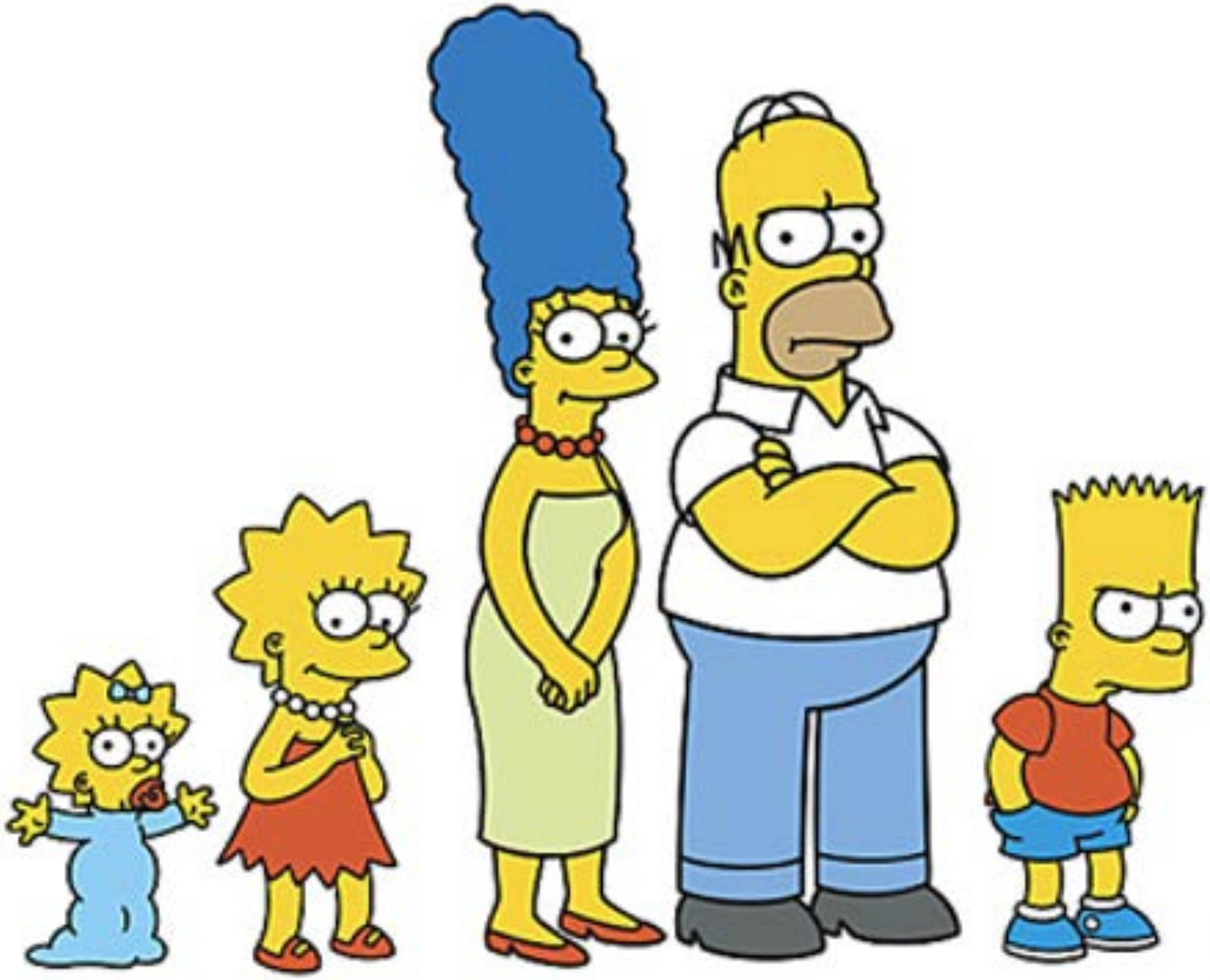
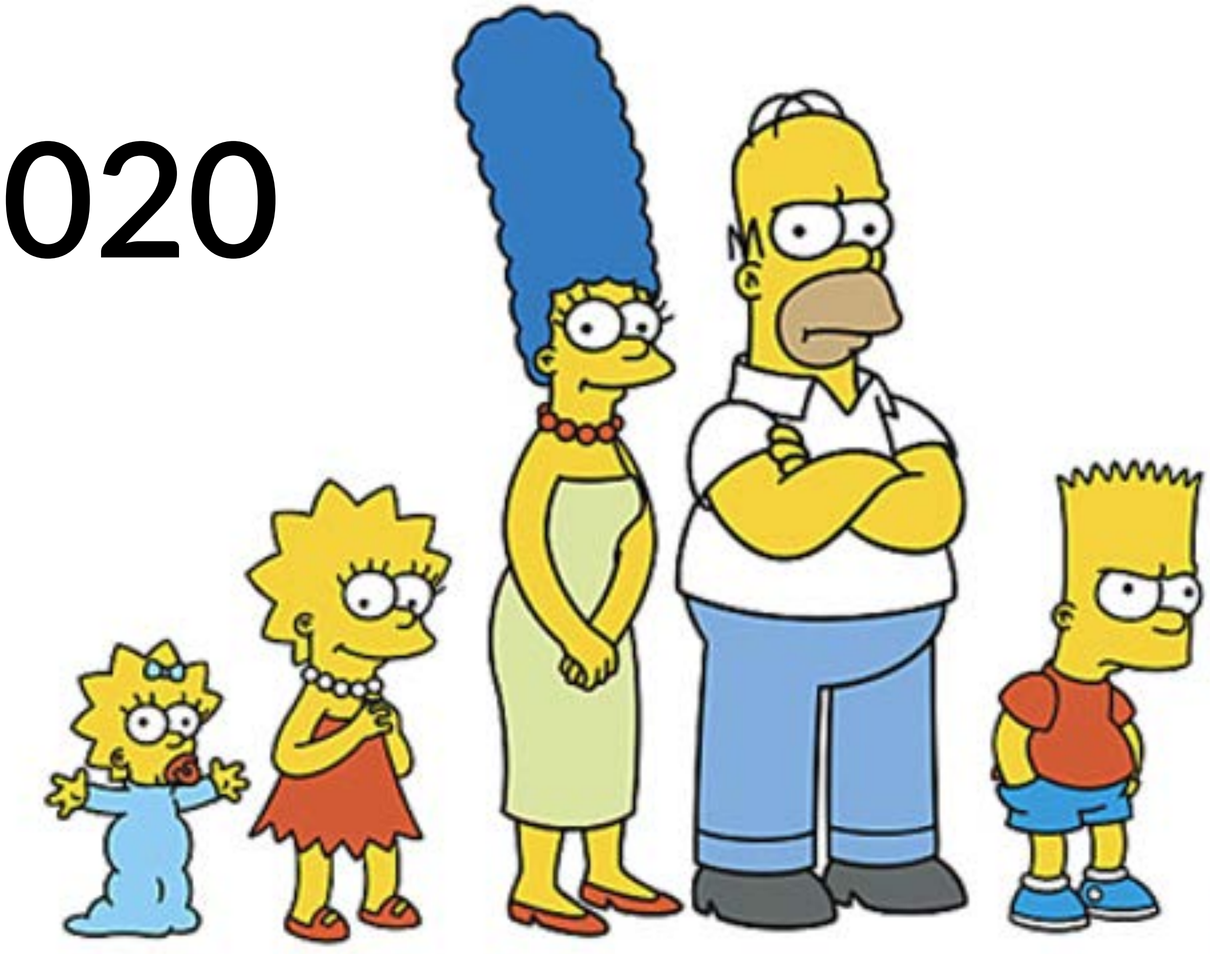
Performance (Really) Matters

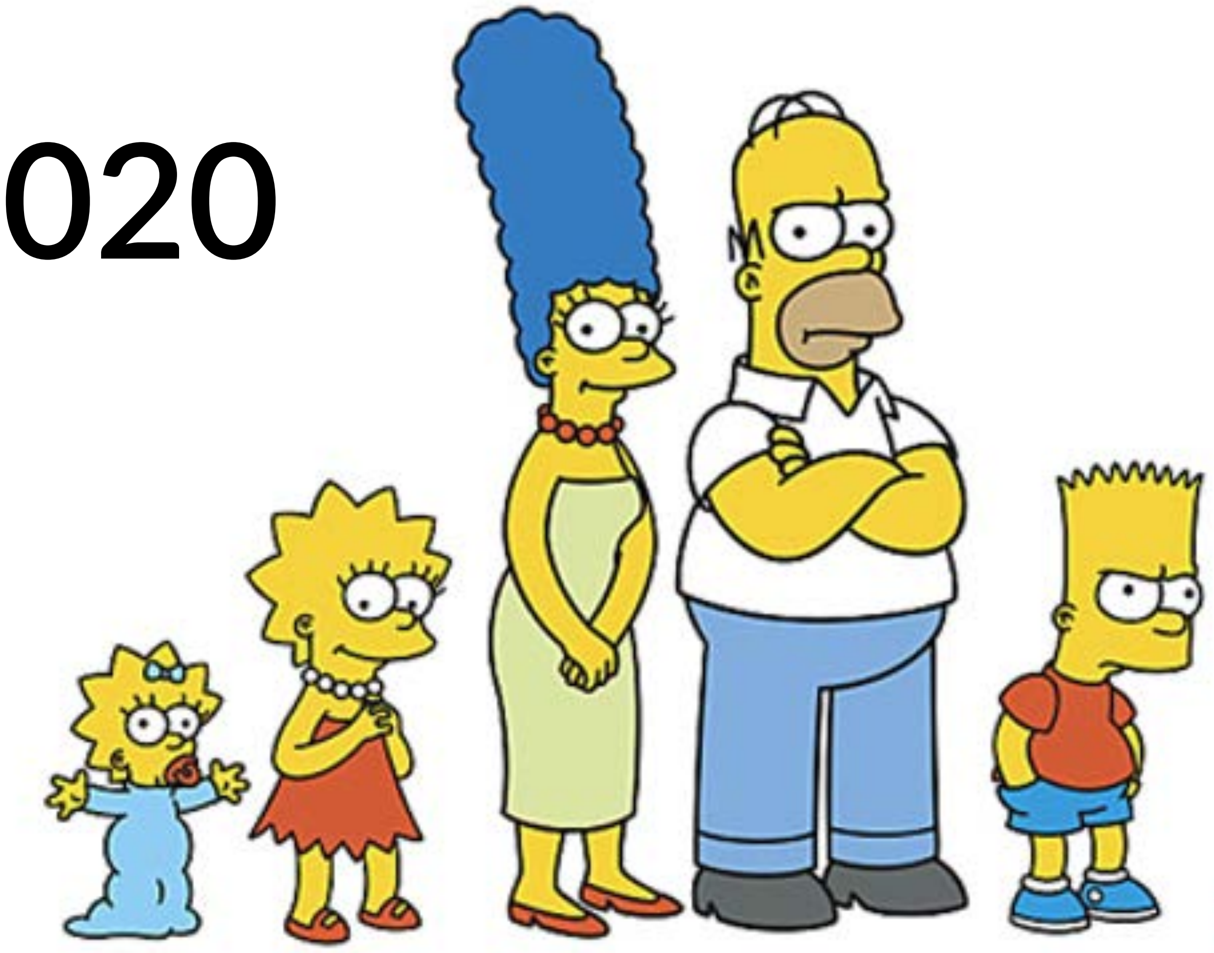
Emery Berger
UMass Amherst
(joint work with Charlie Curtsinger, Grinnell College)
emeryberger.com, @emeryberger



2020

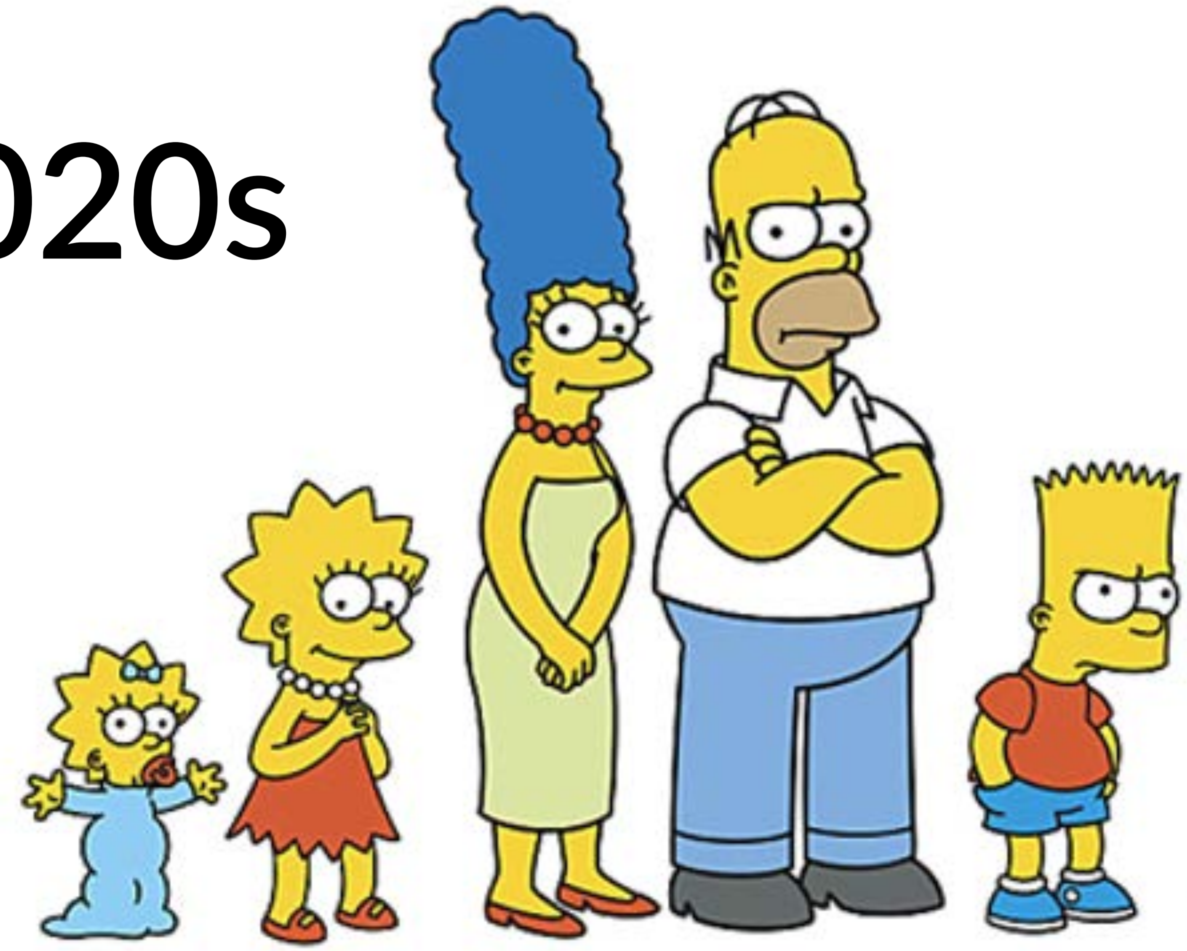


2020

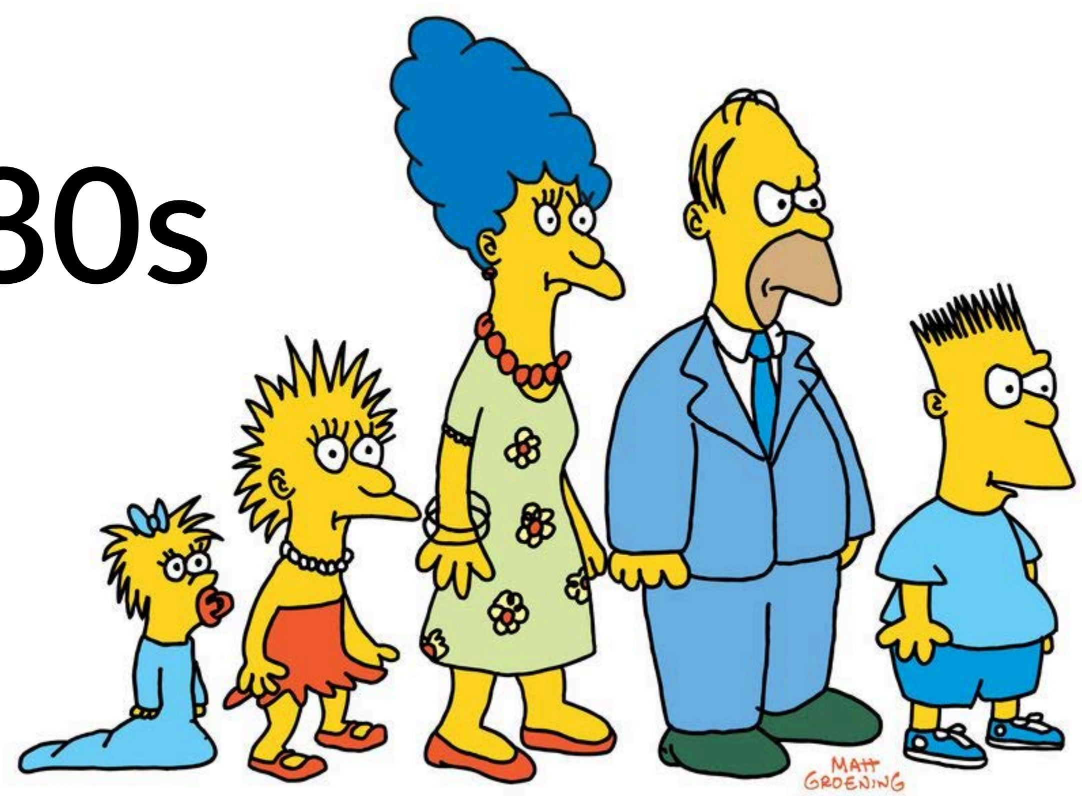


1980s

2020s



1980s



2020

2.65 GHz x 6

64-bit processor

4Gb RAM

1980s

4.77 MHz

16-bit processor

640Kb RAM

2.65 GHz x 6

64-bit processor

4Gb RAM

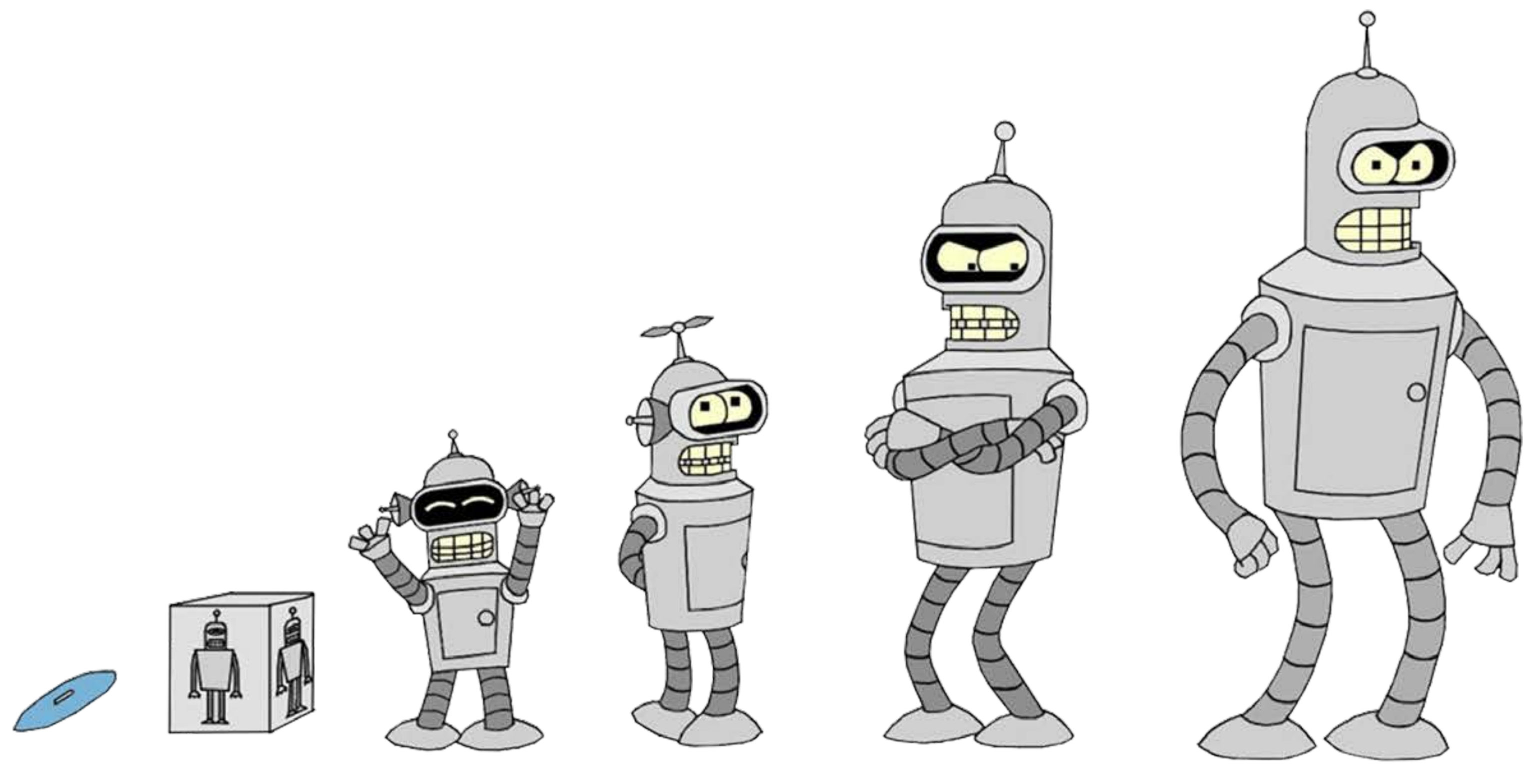
4.77 MHz

16-bit processor

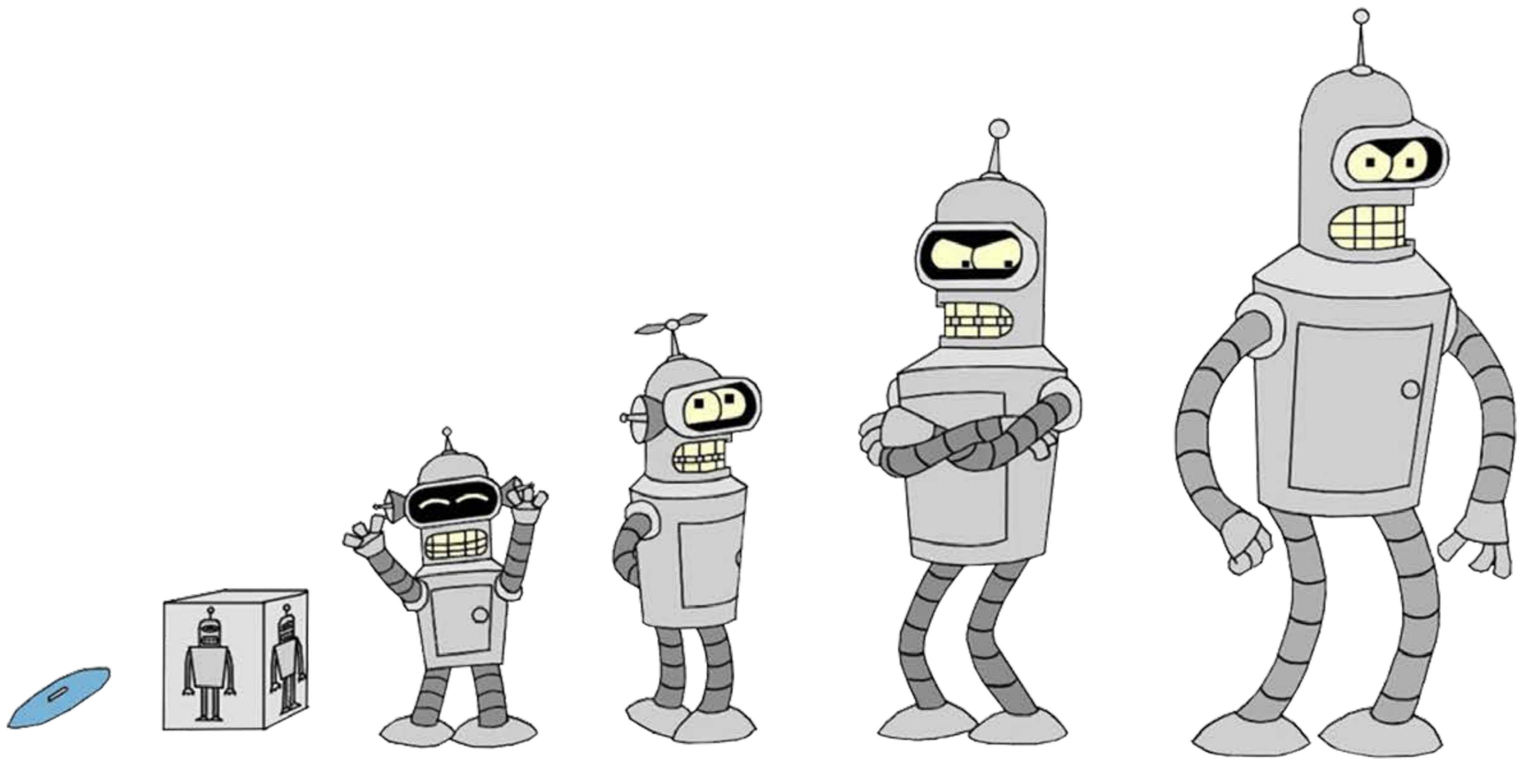
640Kb RAM

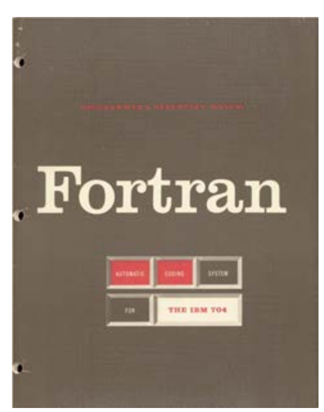
$$\frac{\begin{array}{l}\text{2.65 GHz x 6}\\ \text{64-bit processor}\\ \text{4Gb RAM}\end{array}}{\begin{array}{l}\text{4.77 MHz}\\ \text{16-bit processor}\\ \text{640Kb RAM}\end{array}} \approx \mathbf{3000x}$$

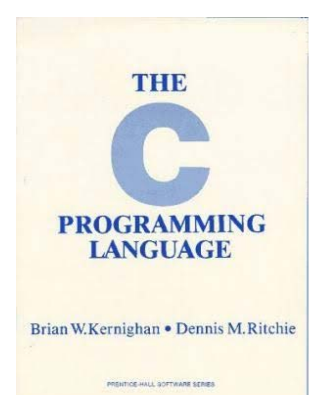
–1980s

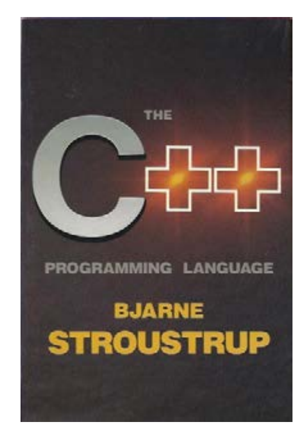


–1980s



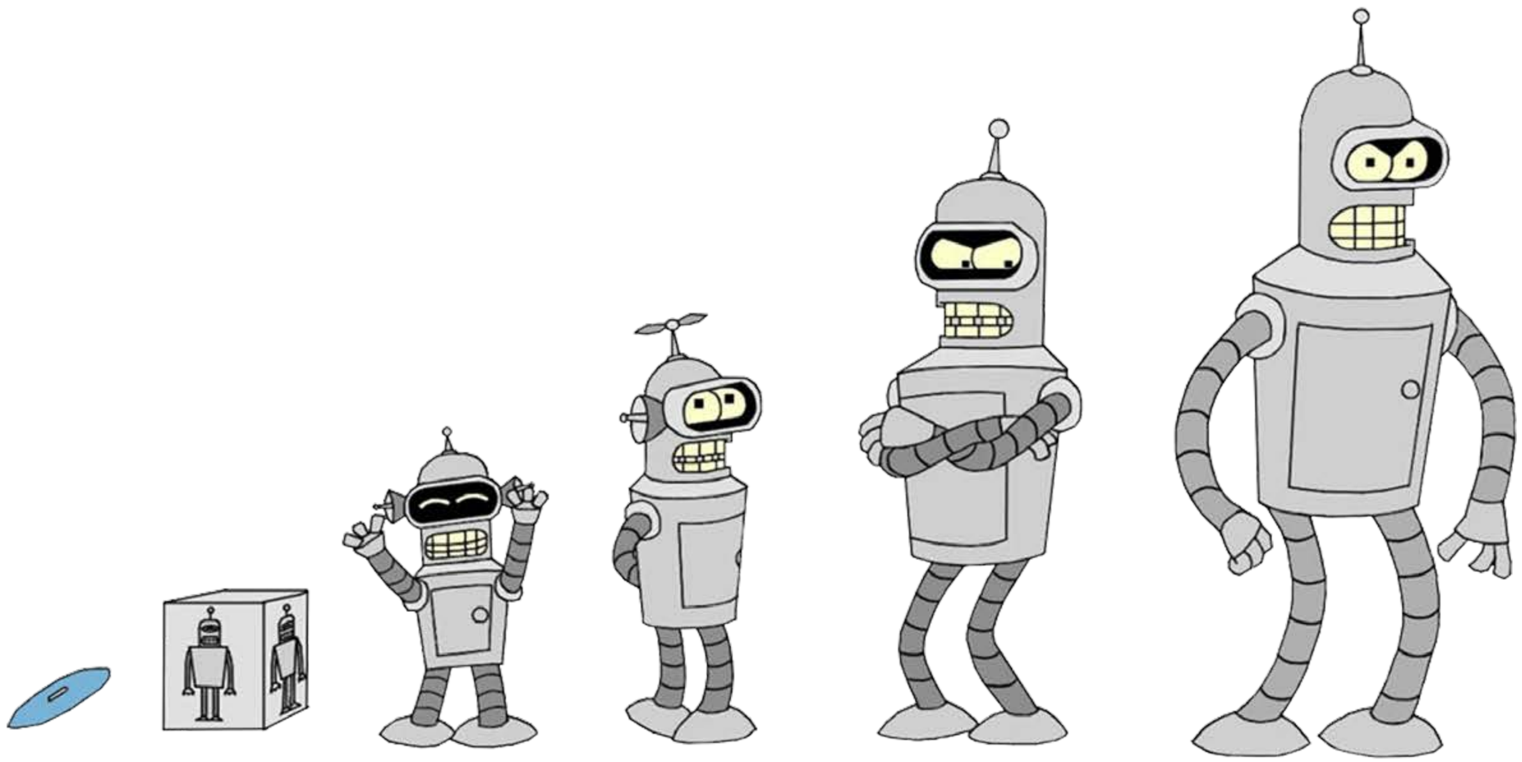


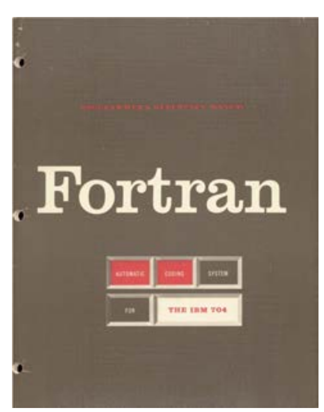


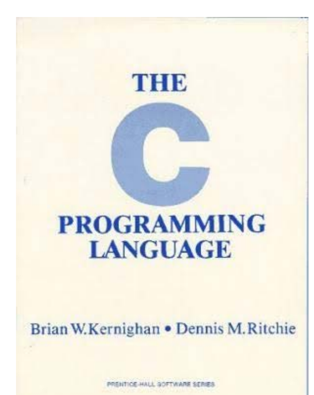


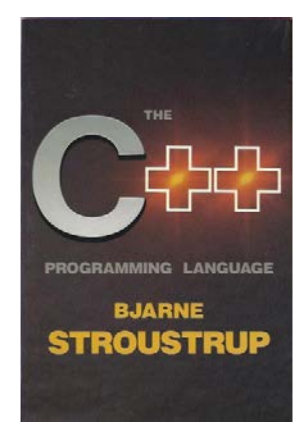
1949 1957 1972 1985

"Metal" Languages









1949 1957 1972 1985

40 Year Performance Ride!

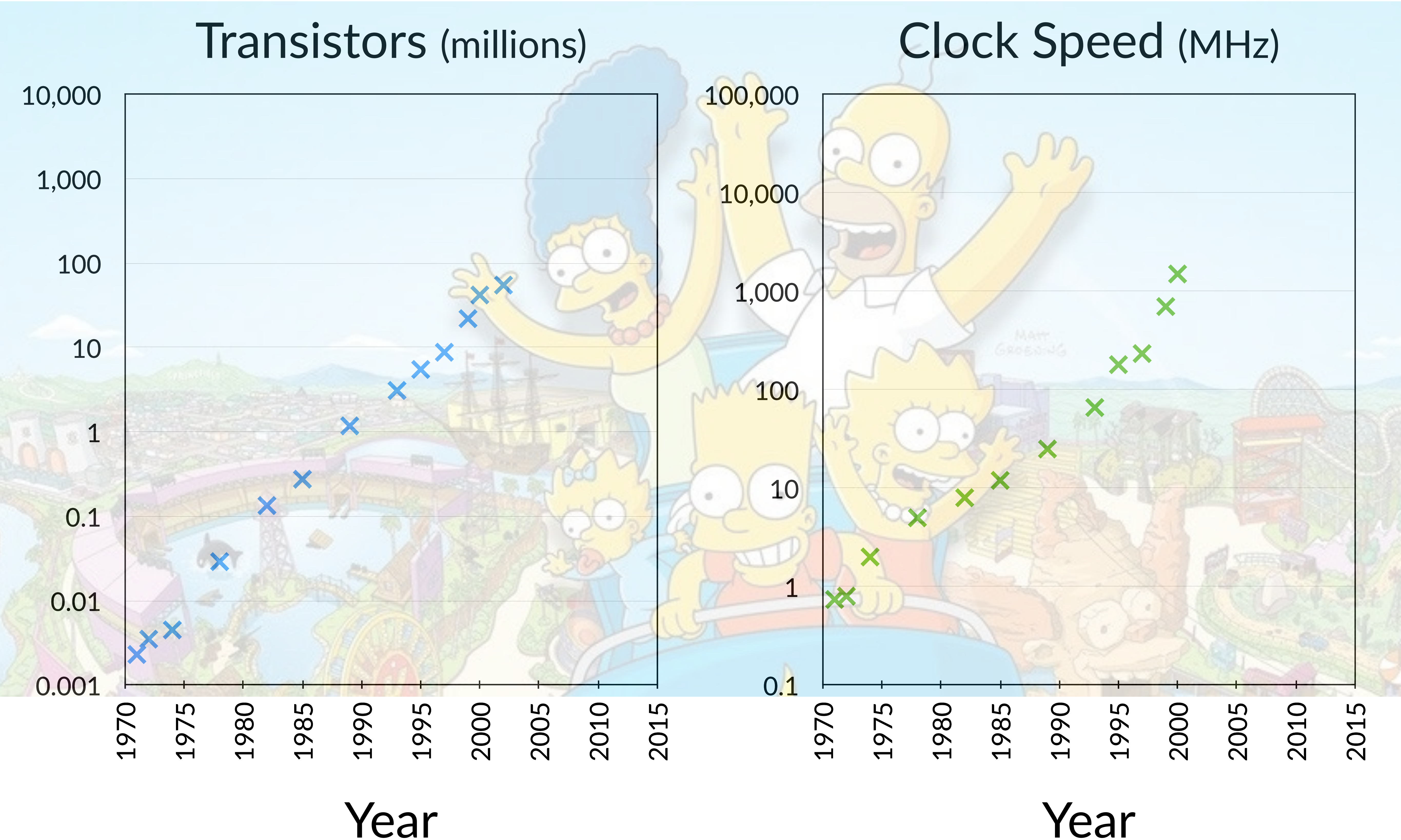
1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

Year

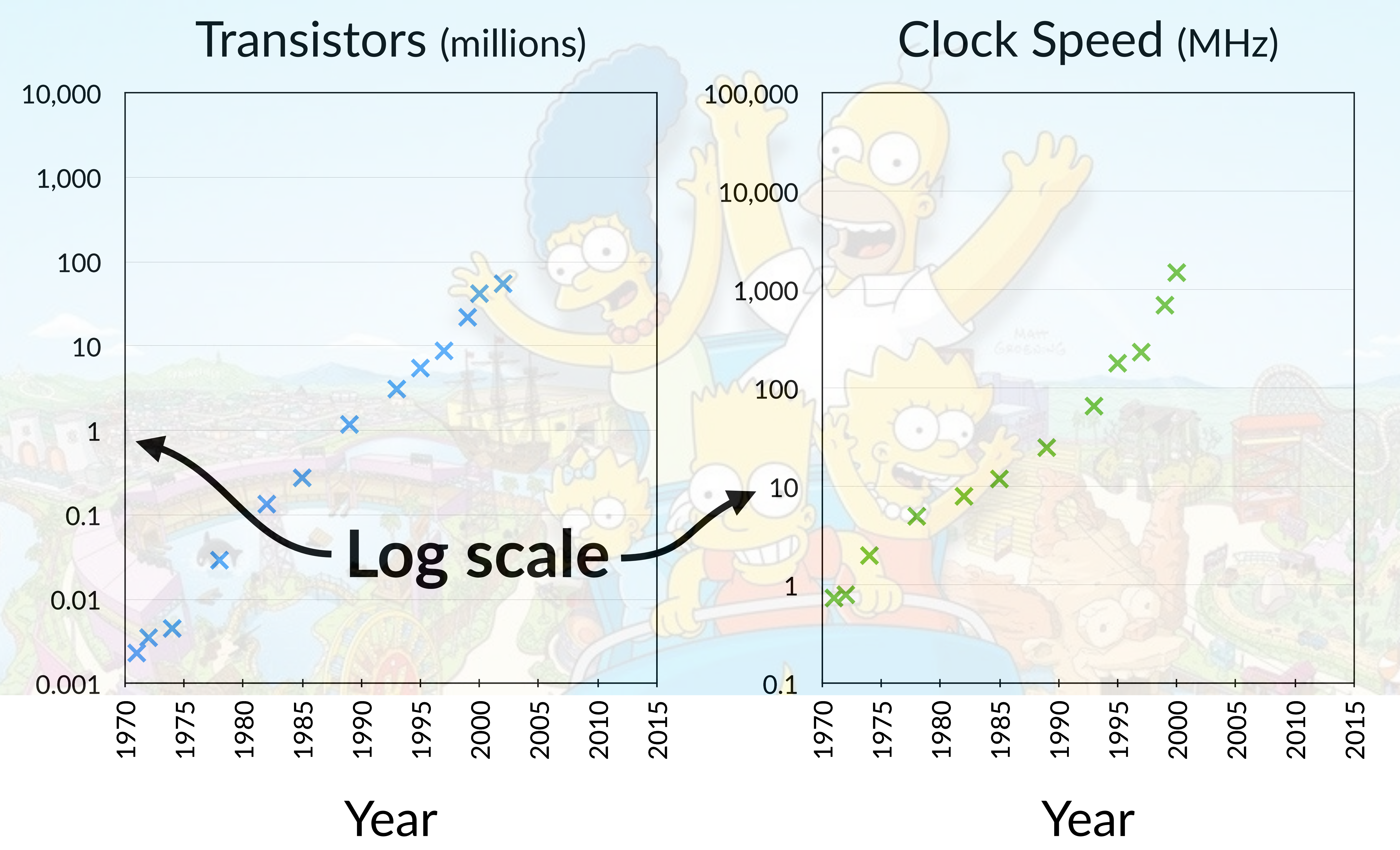
1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

Year

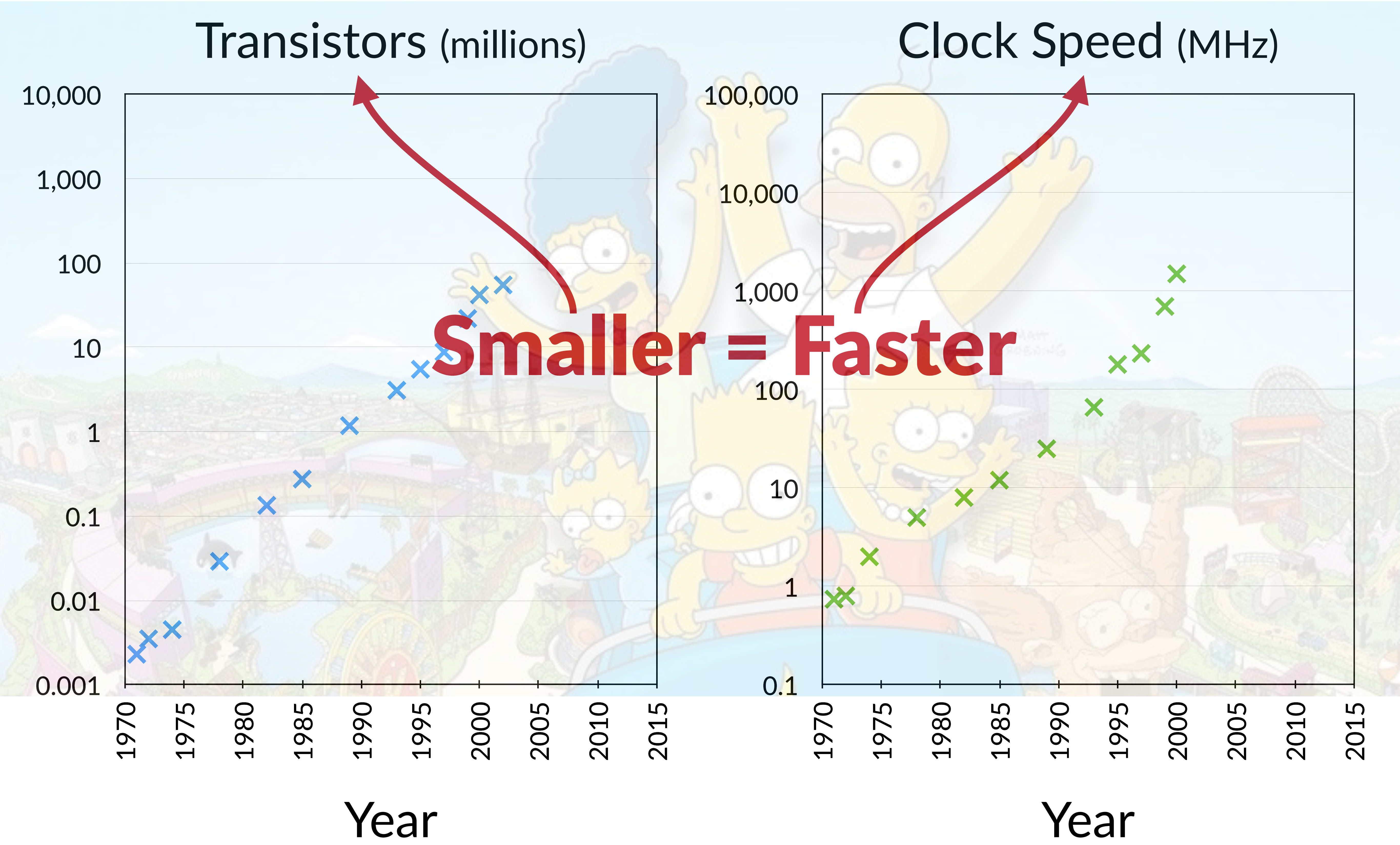
40 Year Performance Ride!



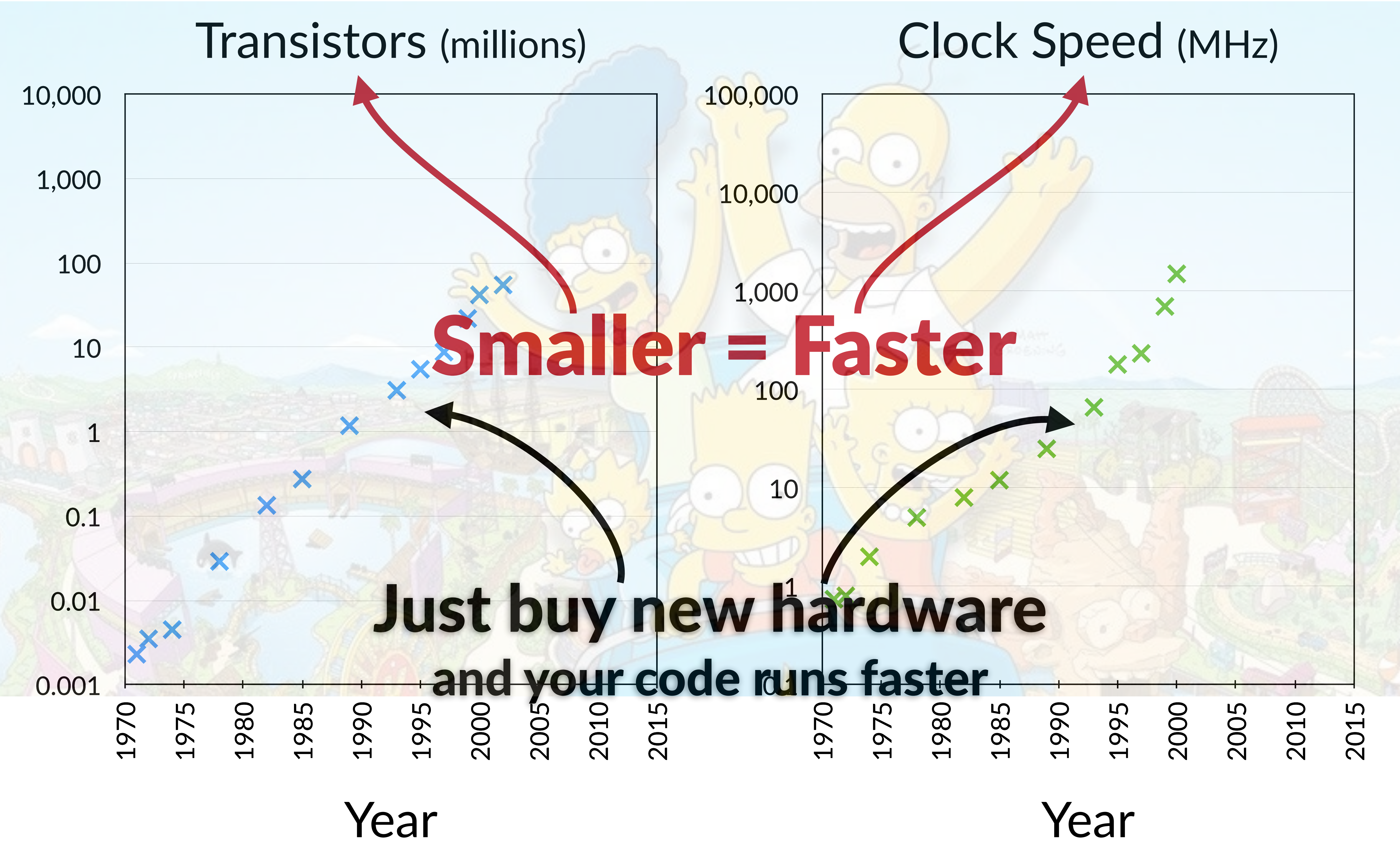
40 Year Performance Ride!



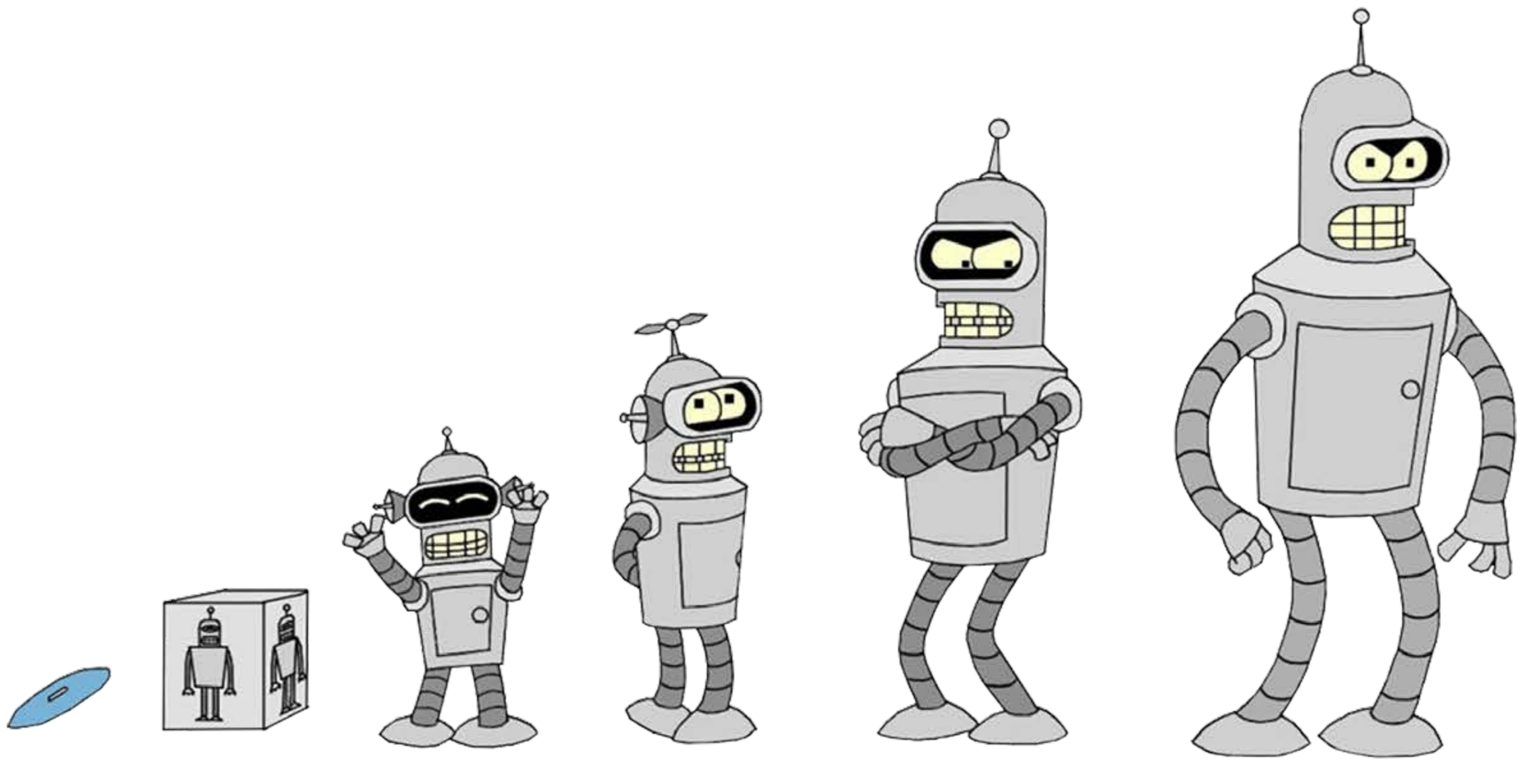
40 Year Performance Ride!

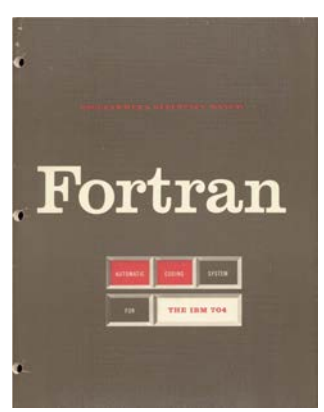


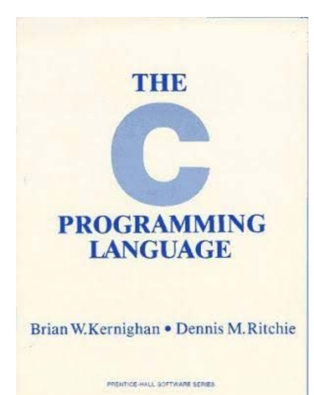
40 Year Performance Ride!

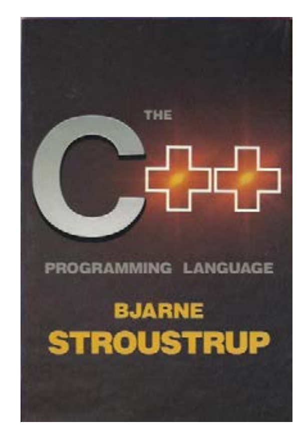


"Metal" Languages



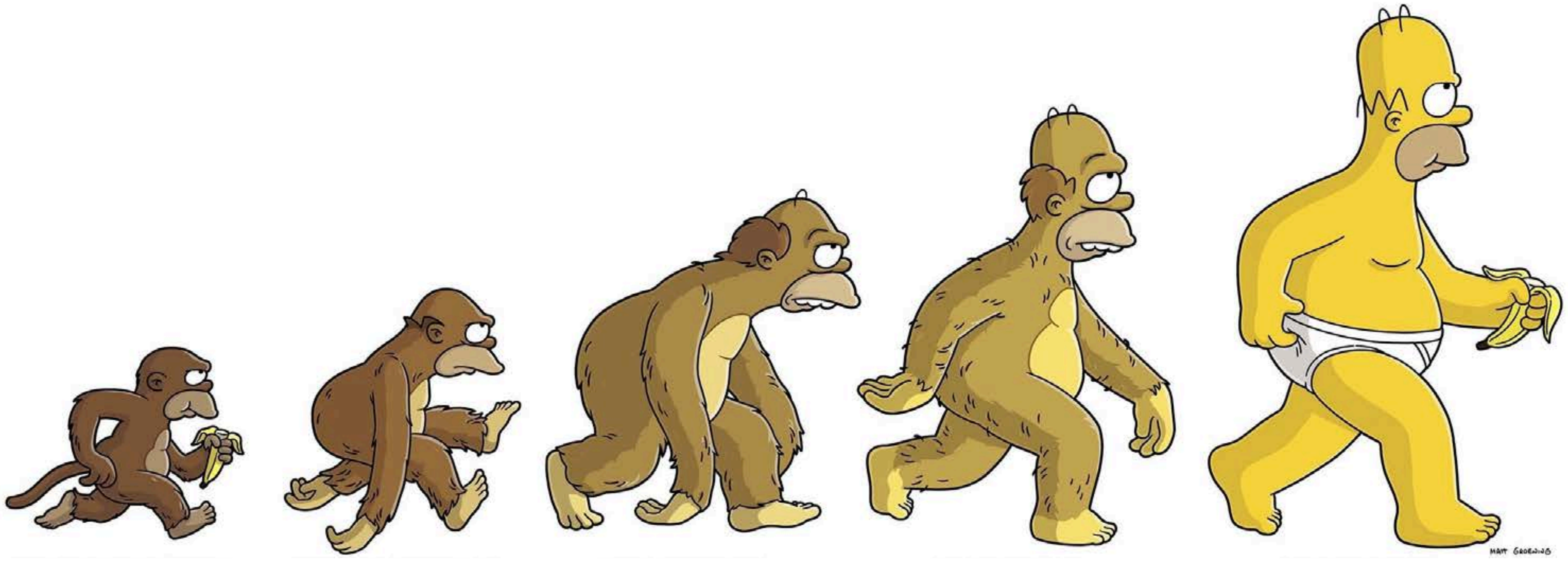




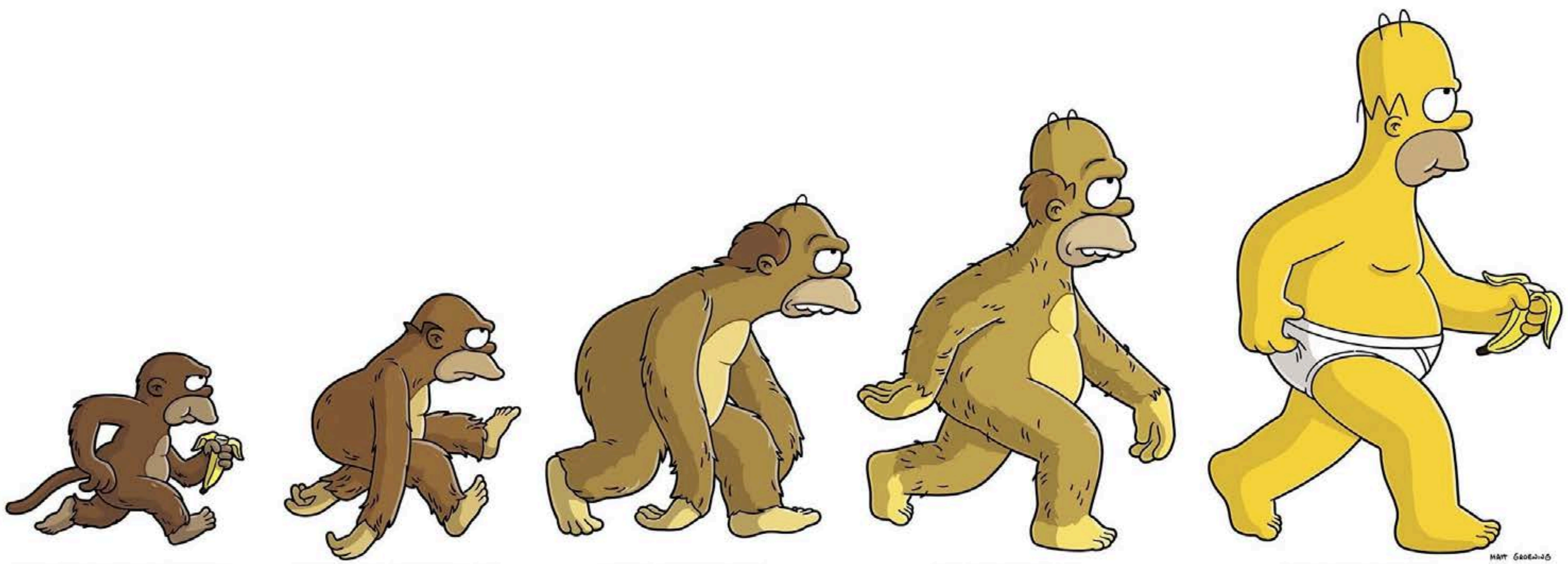


1949 1957 1972 1985

1980-1990s

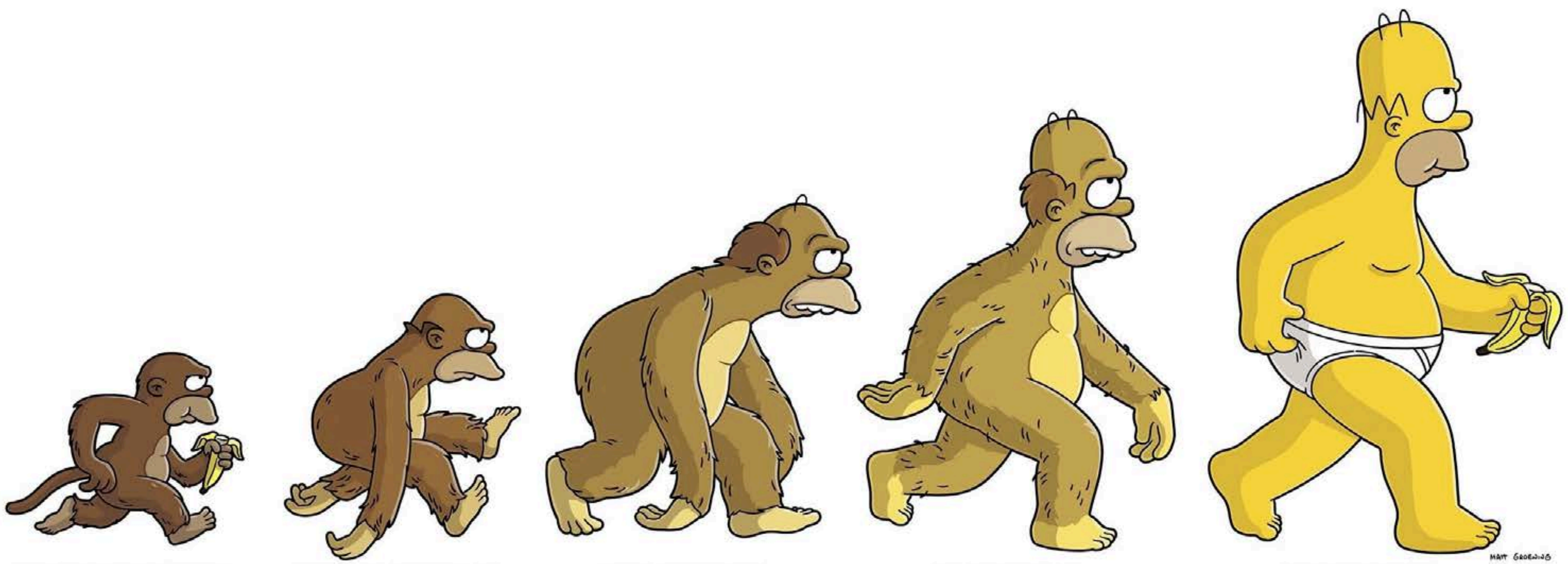


1980-1990s



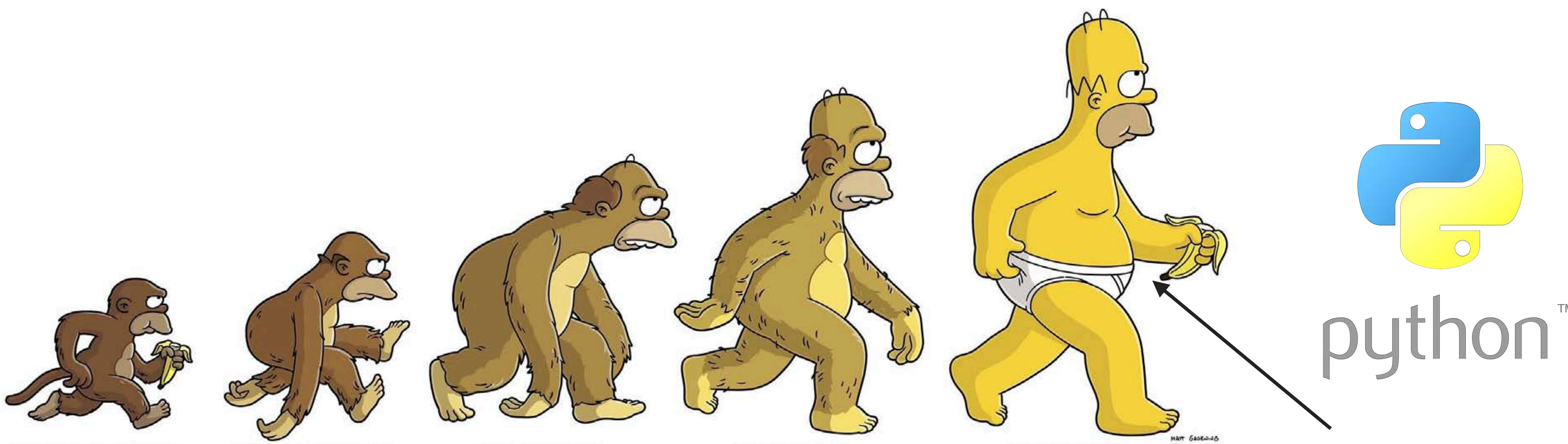
1991 1987 1995 1995 1990

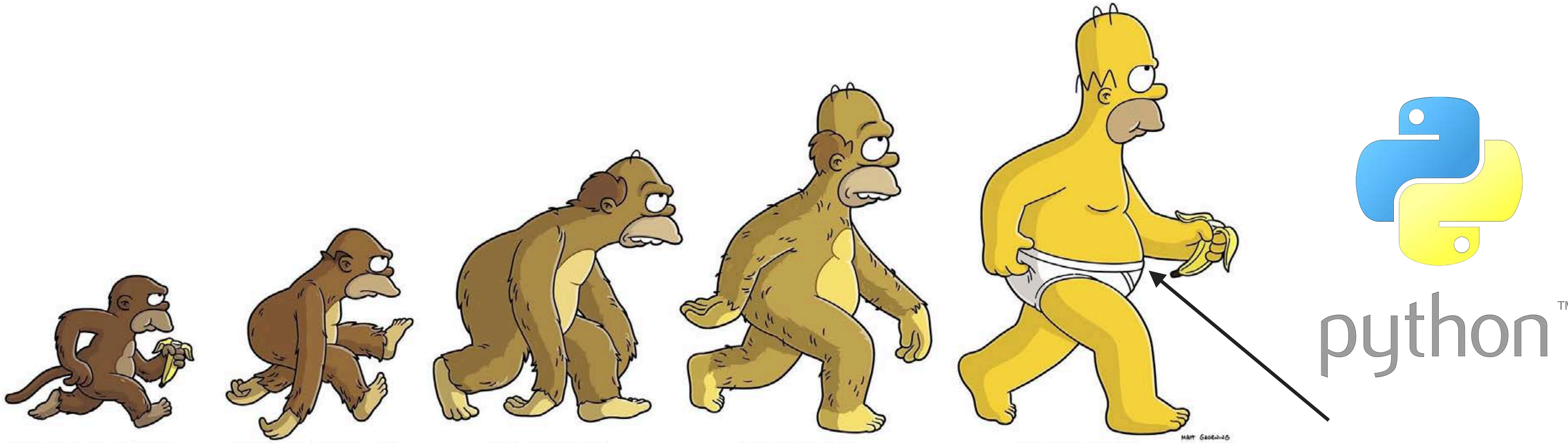
"Irrational Exuberance" Languages



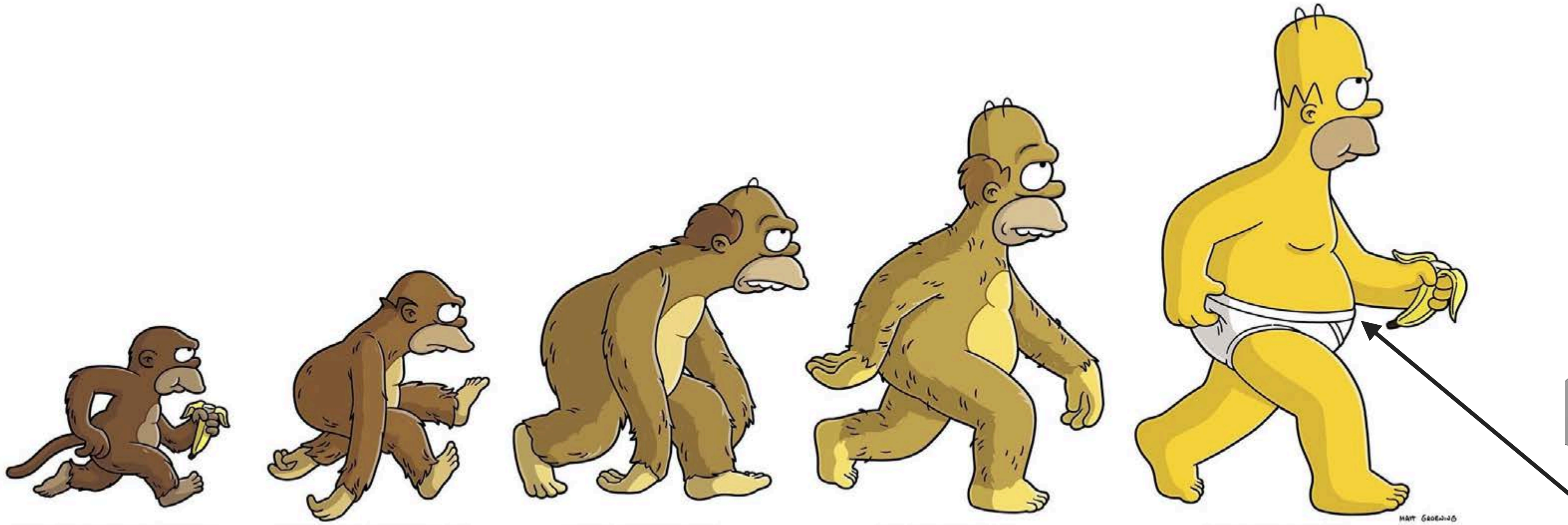
1991 1987 1995 1995 1990

"Irrational Exuberance" Languages

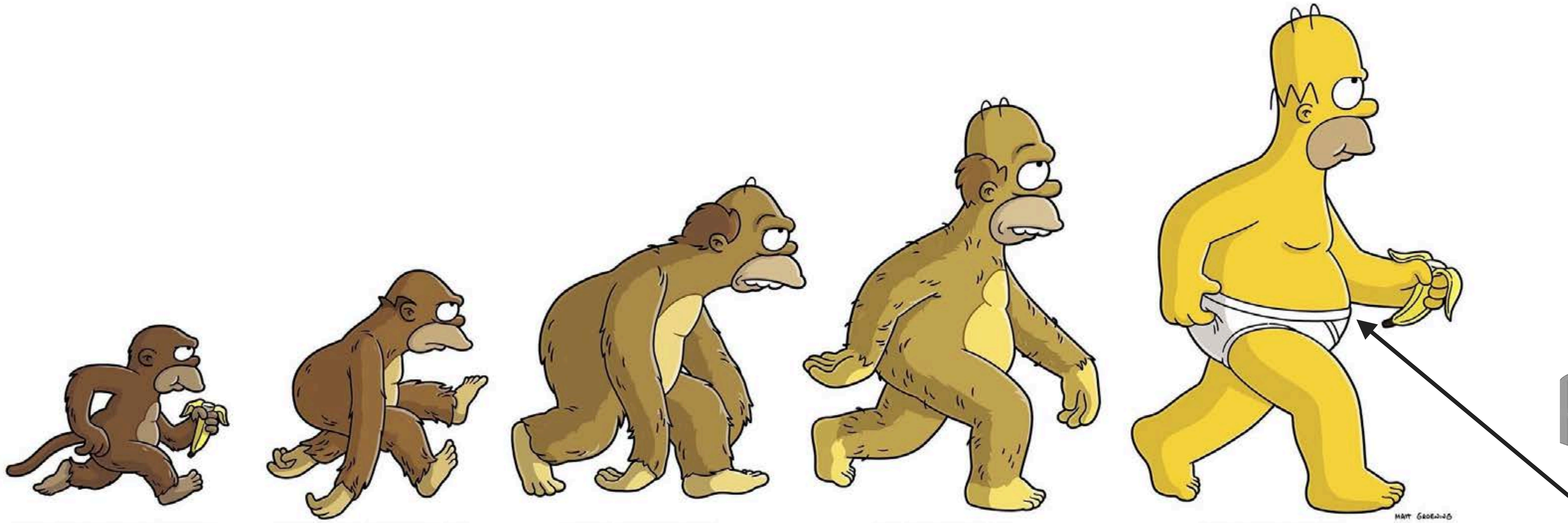




HOW MANY BYTES IN AN…INT?

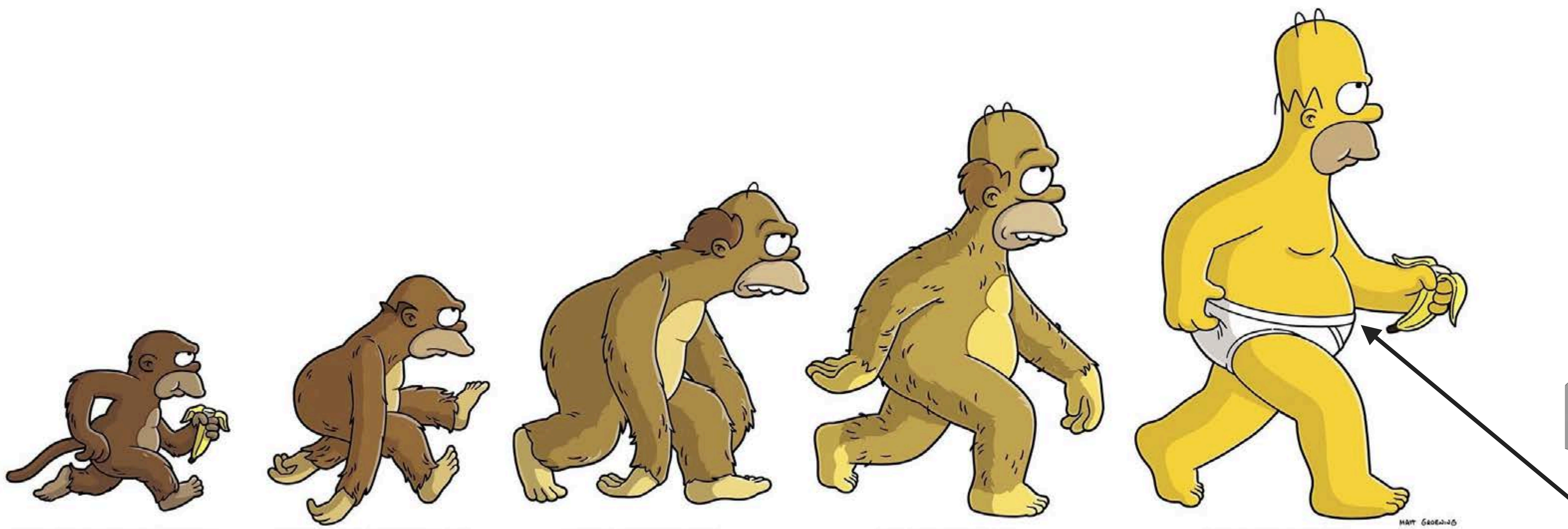


HOW MANY BYTES IN AN…INT?



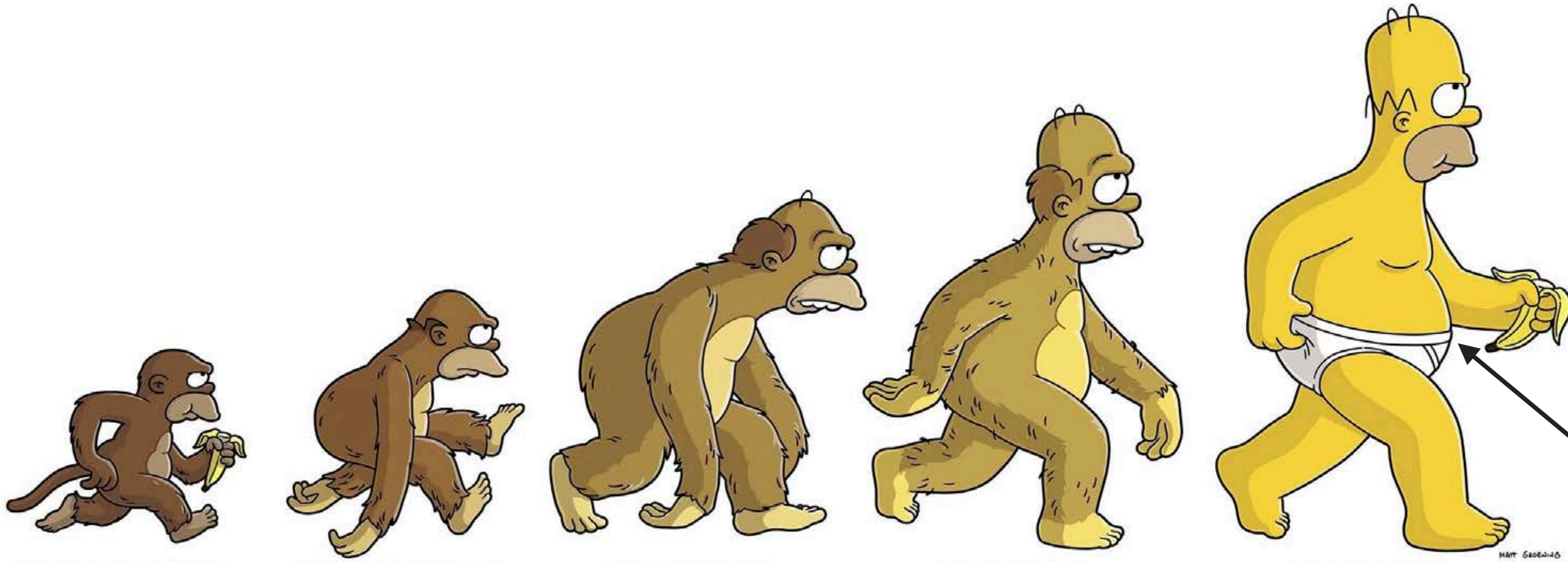
HOW MANY BYTES IN AN…INT?

```
sizeof(1)
```



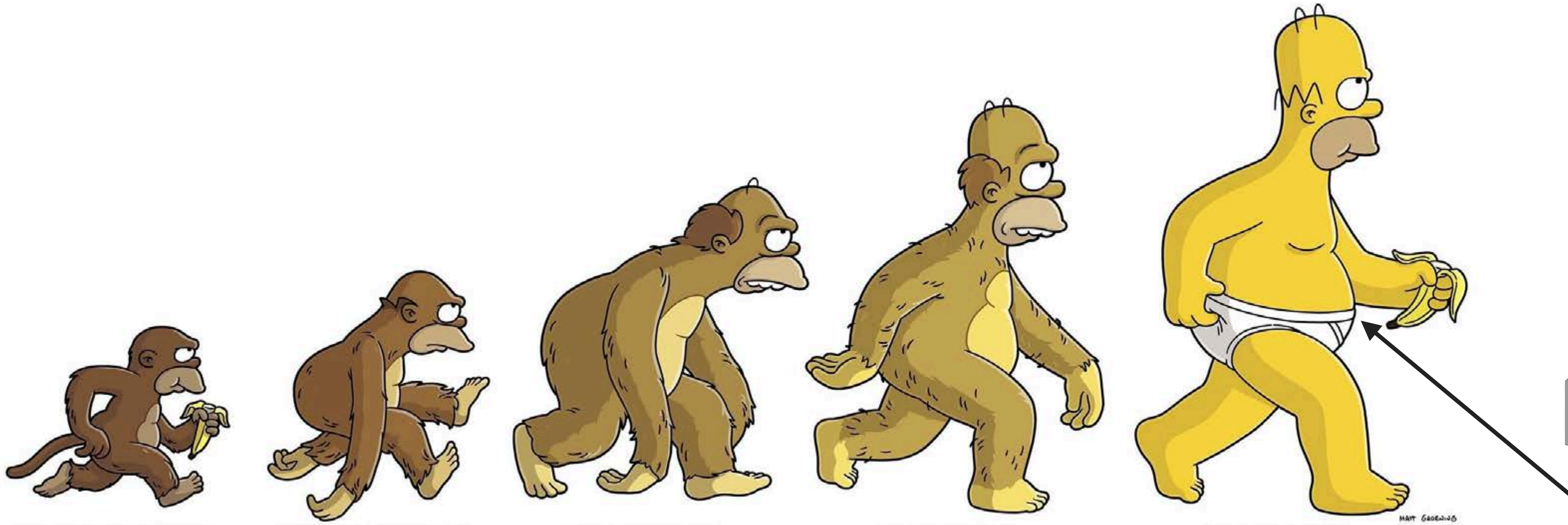
HOW MANY BYTES IN AN…INT?

```
sizeof(1)
→ 4
```



HOW MANY BYTES IN AN…INT?

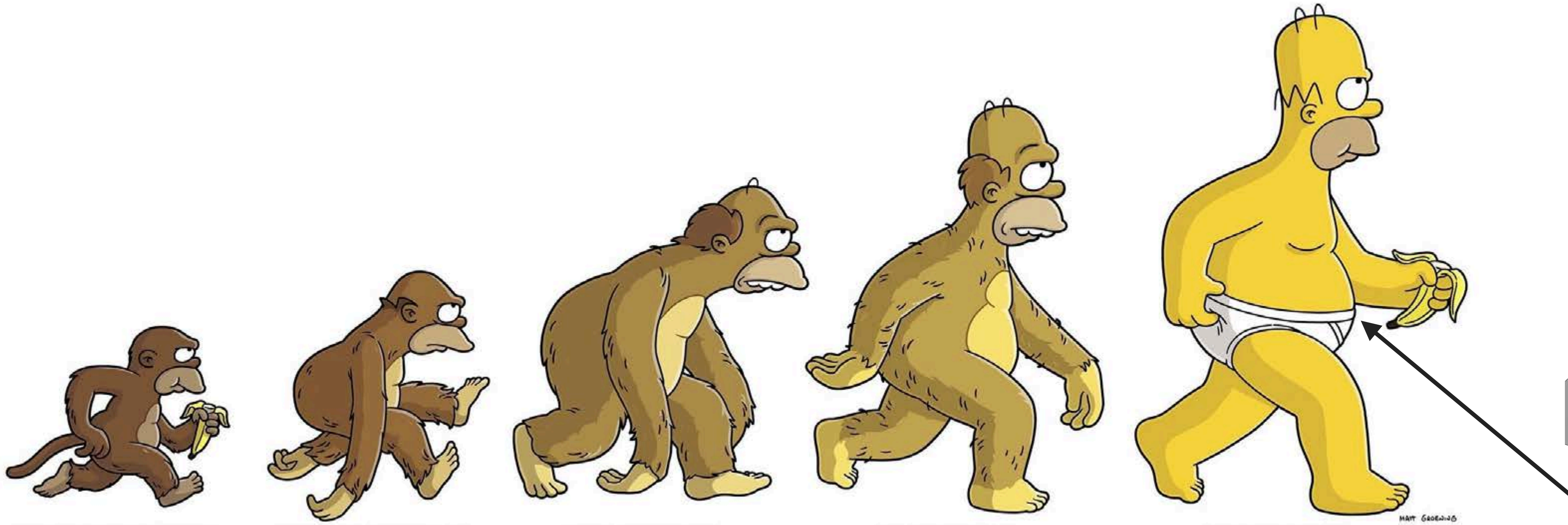
```
sizeof(1)
→ 4
```



HOW MANY BYTES IN AN…INT?

```
sizeof(1)
→ 4
```

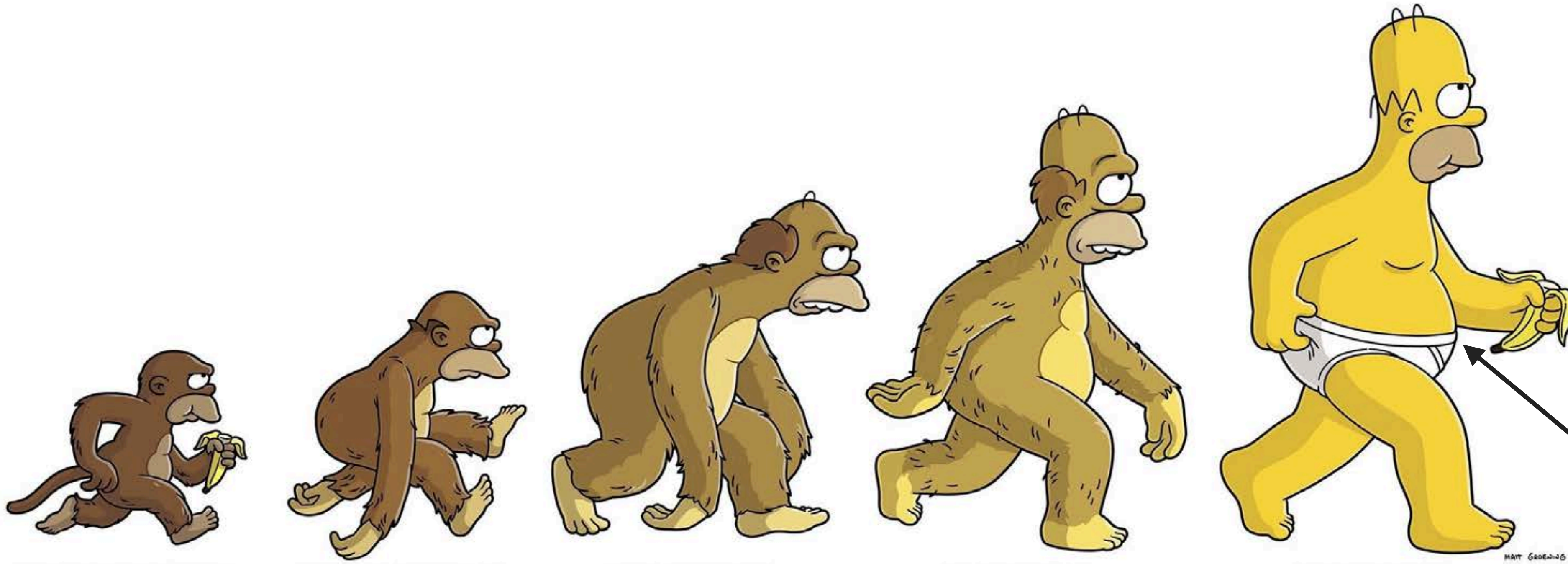
```
>>> sys.getsizeof(1)
```



HOW MANY BYTES IN AN…INT?

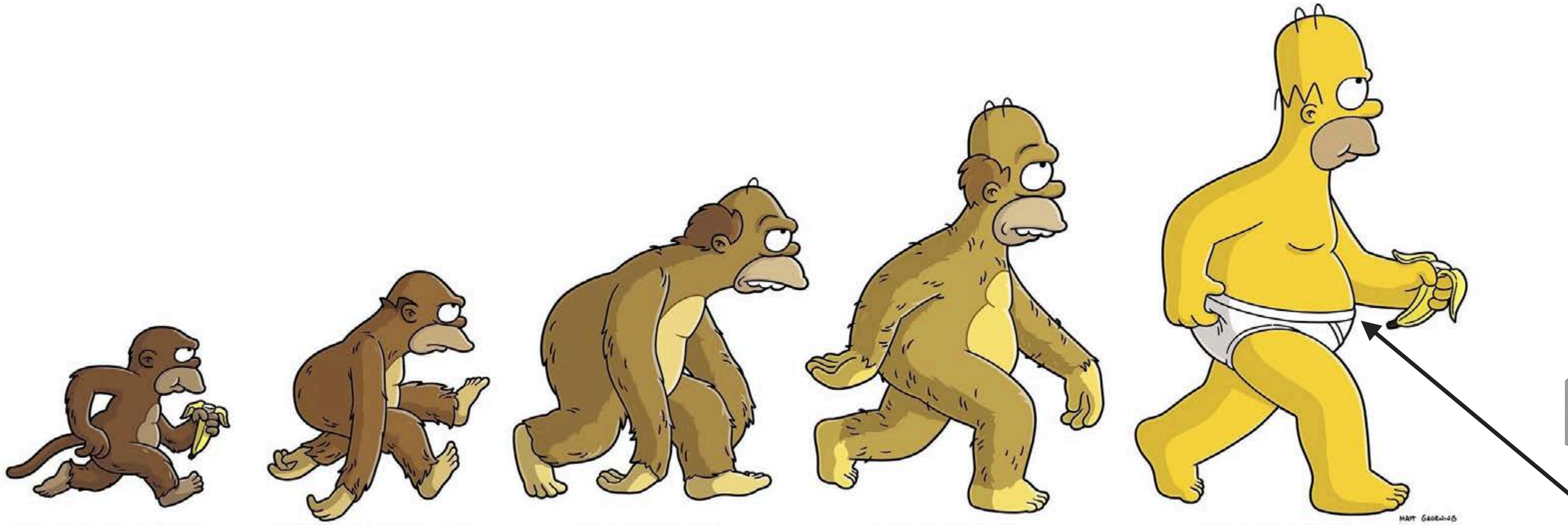
```
sizeof(1)
→ 4
```

```
>>> sys.getsizeof(1)
24
```



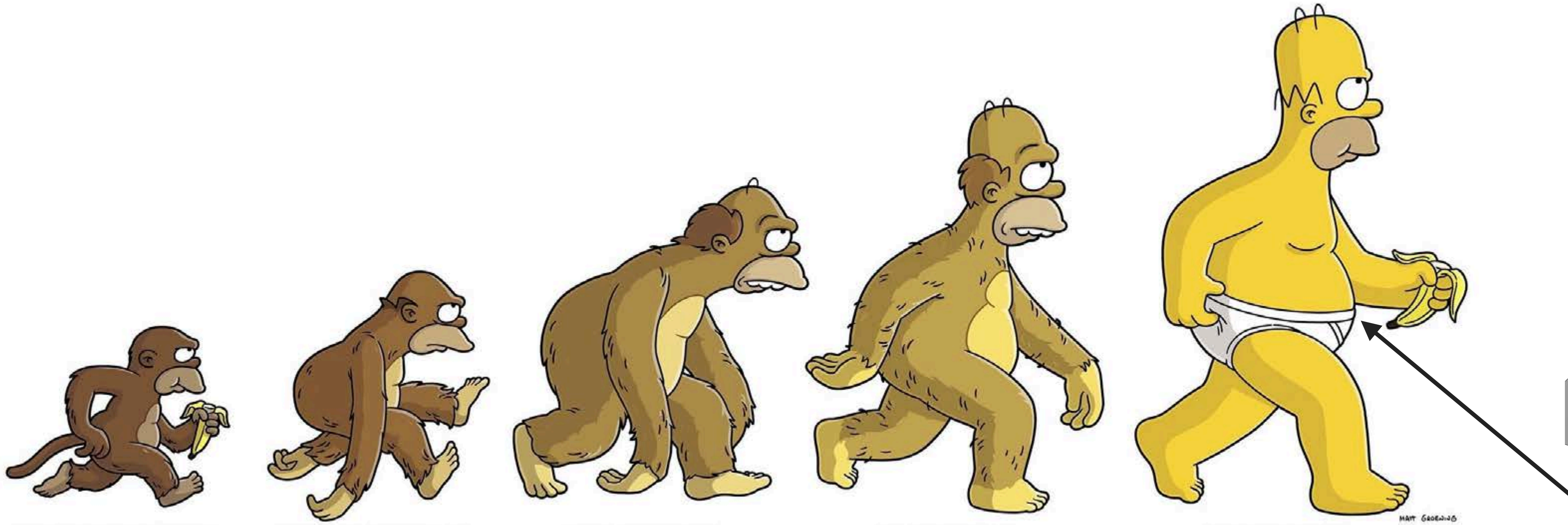
python™

HOW MANY BYTES IN AN…EMPTY LIST?



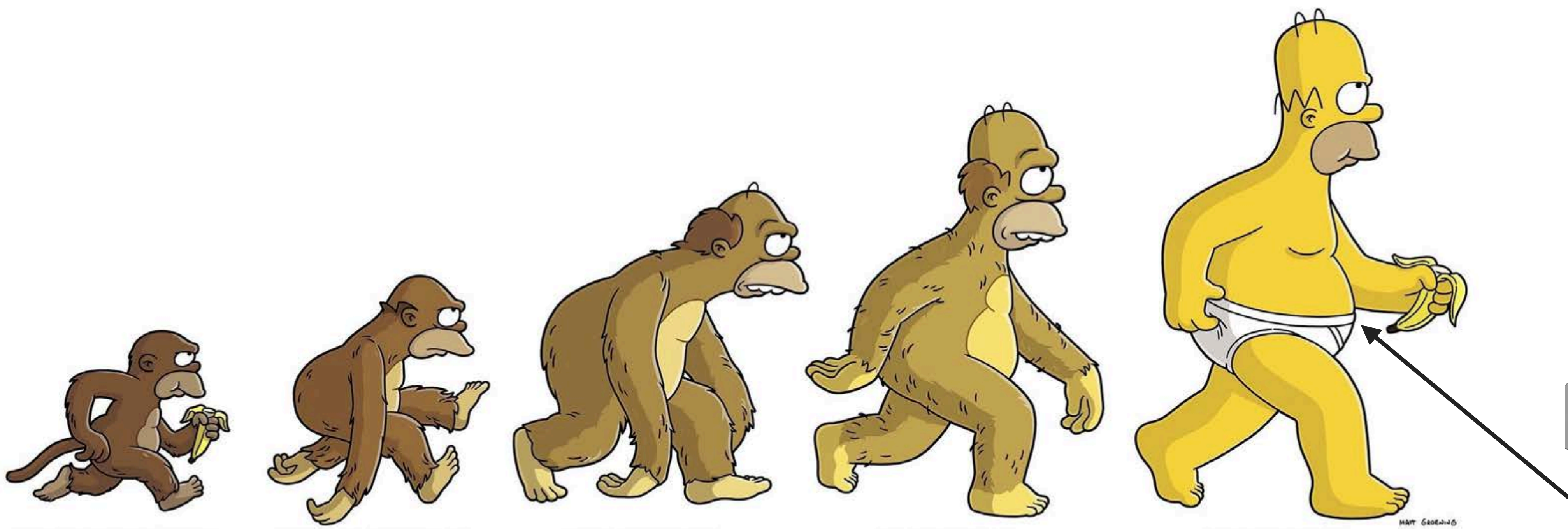
HOW MANY BYTES IN AN…EMPTY LIST?

```
sizeof(list<int>)
```



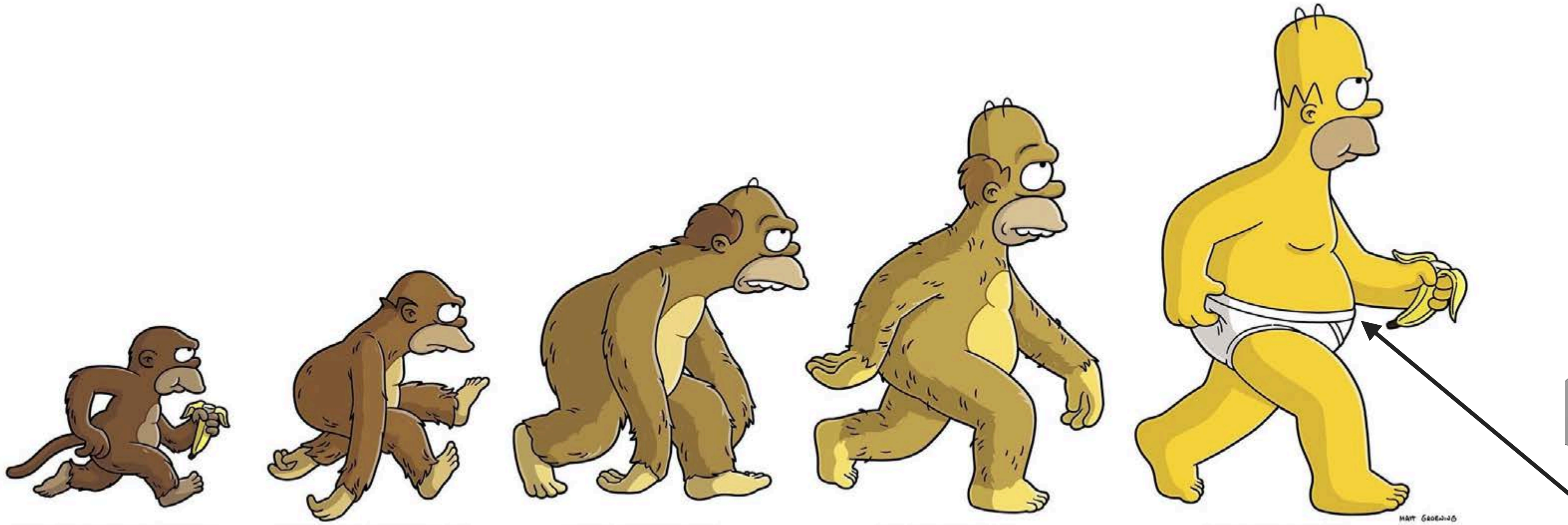
HOW MANY BYTES IN AN…EMPTY LIST?

```
sizeof(list<int>)
→ 24
```



HOW MANY BYTES IN AN…EMPTY LIST?

```
sizeof(list<int>)
→ 24
```



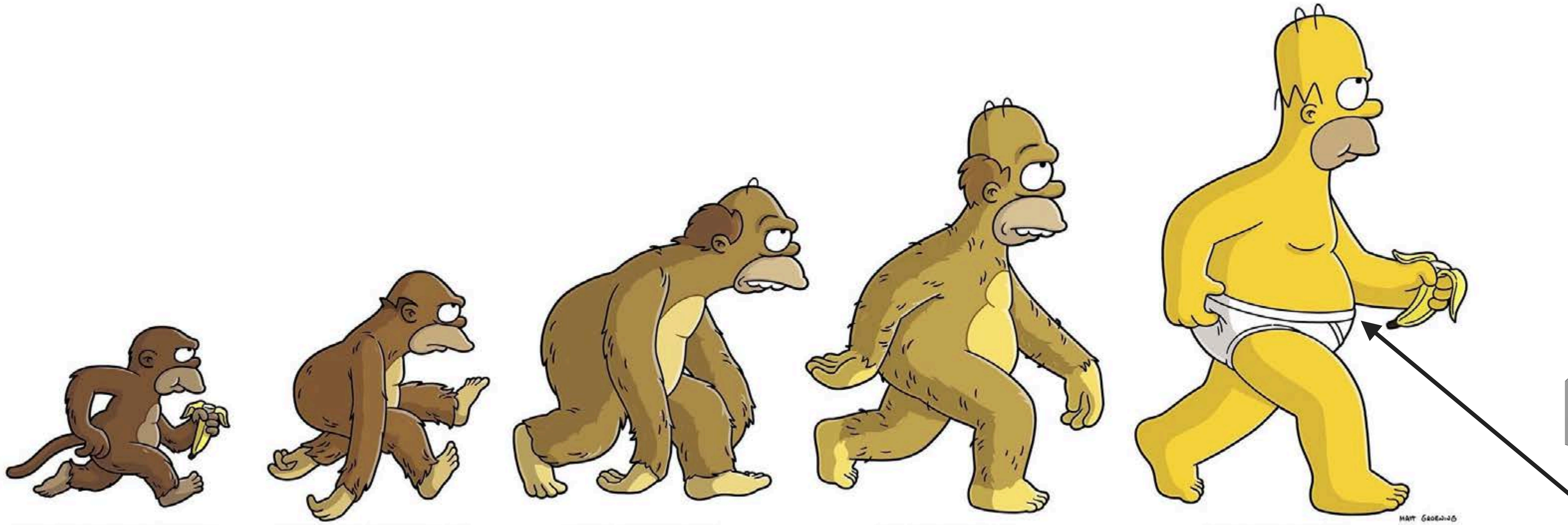
python™

HOW MANY BYTES IN AN…EMPTY LIST?

```
sizeof(list<int>)
→ 24
```

```
>>> sys.getsizeof([])
```

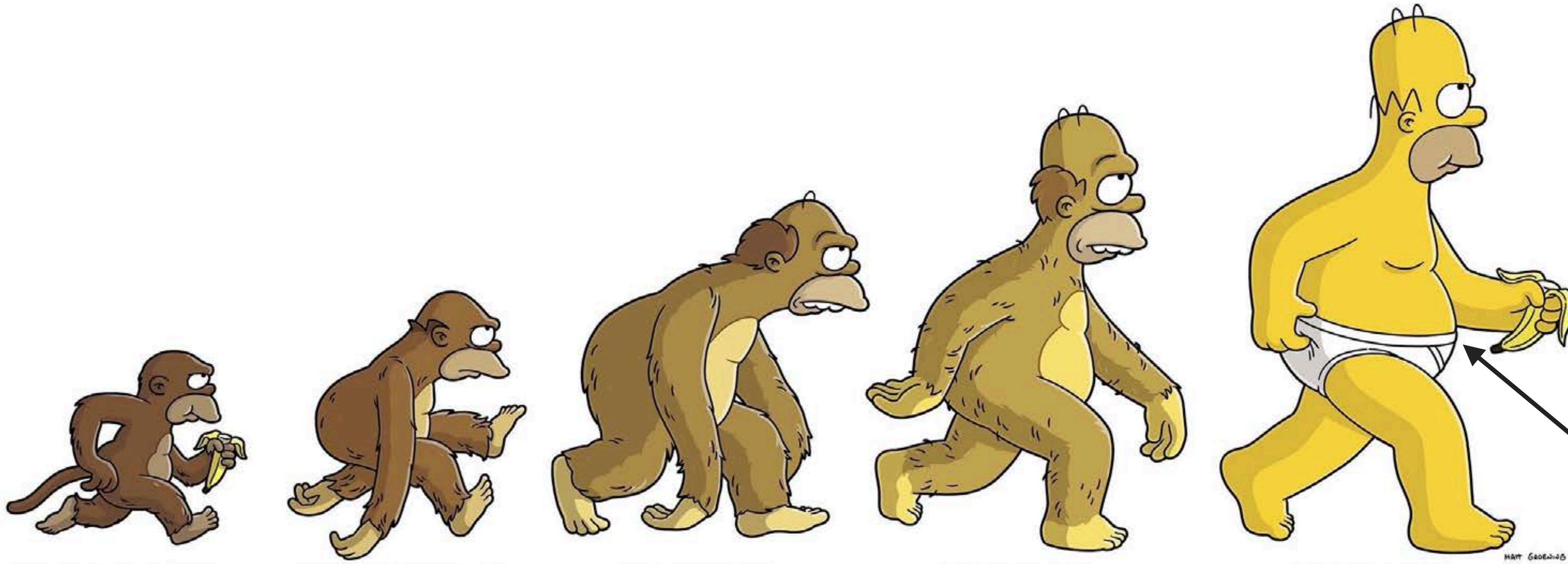
python™



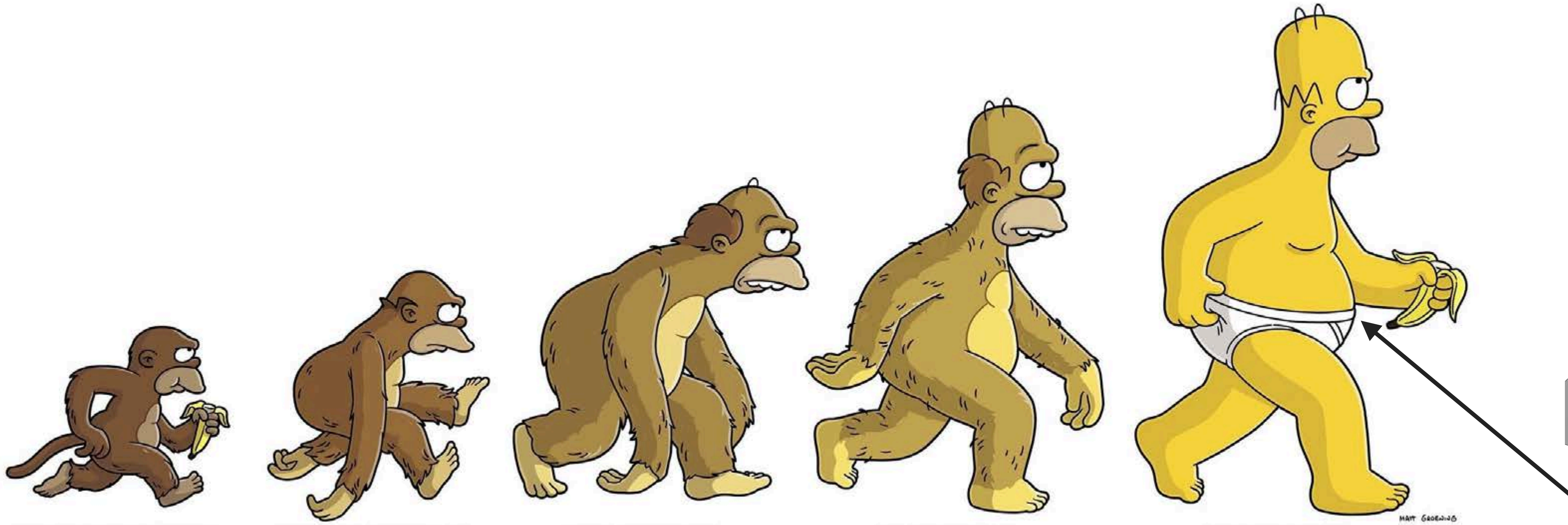
HOW MANY BYTES IN AN…EMPTY LIST?

```
sizeof(list<int>)
→ 24
```

```
>>> sys.getsizeof([])
56
```

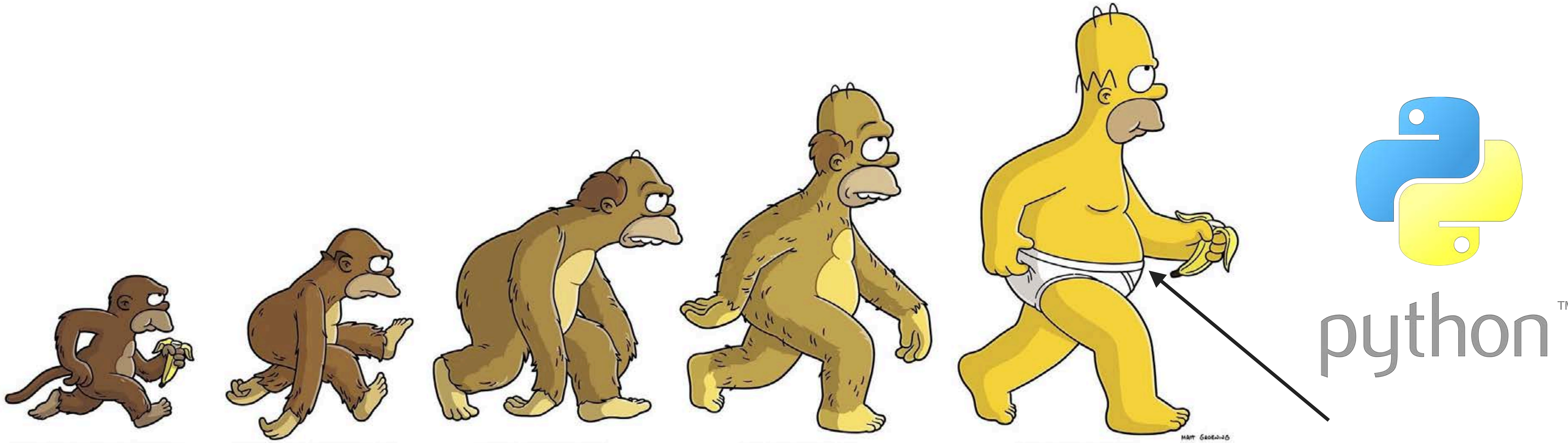


HOW MANY BYTES IN AN…EMPTY DICT?



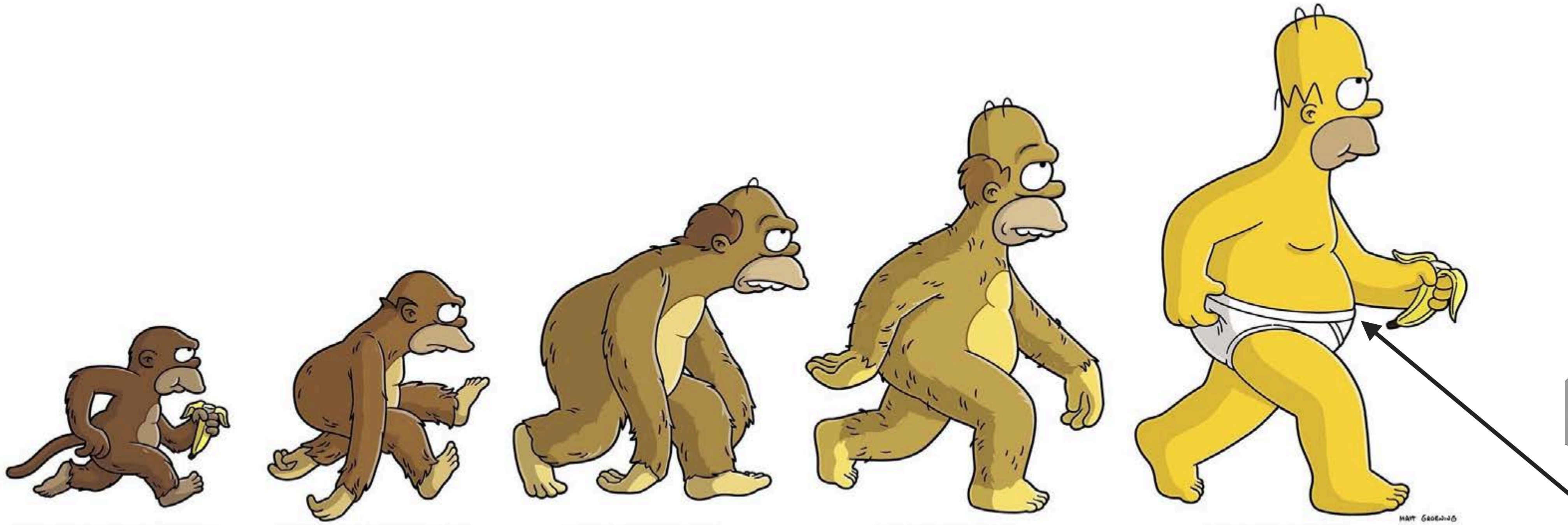
HOW MANY BYTES IN AN…EMPTY DICT?

```
sizeof(map<int, int>)
```



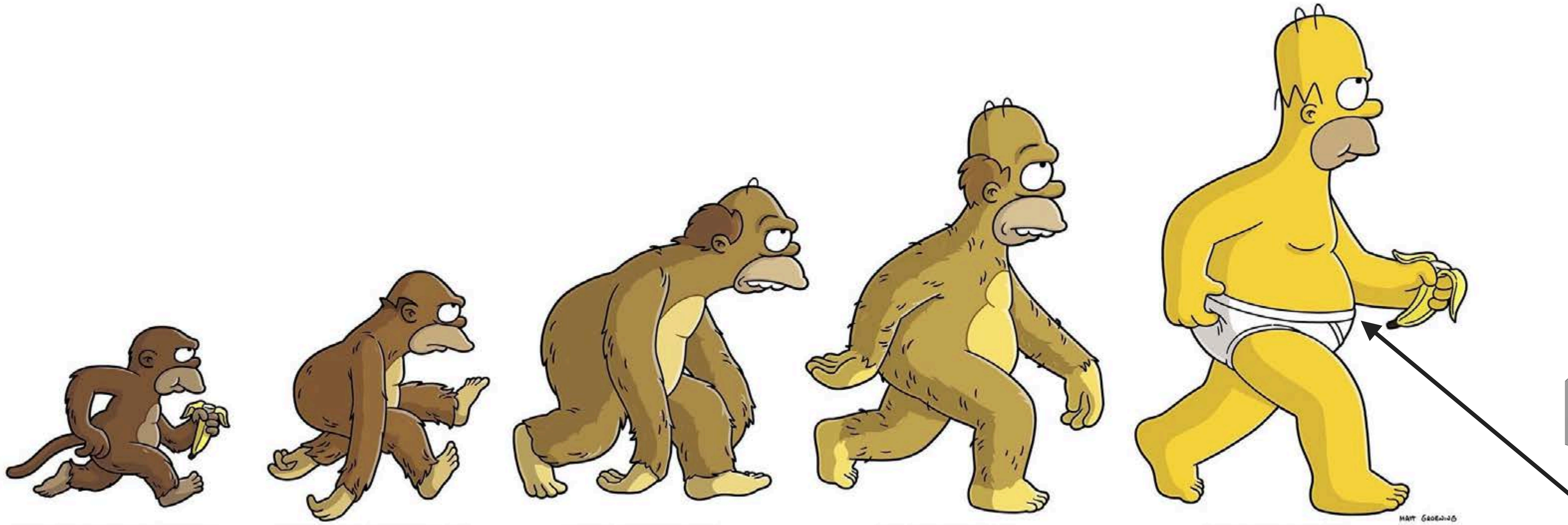
HOW MANY BYTES IN AN…EMPTY DICT?

```
sizeof(map<int,
int>)
→ 24
```



HOW MANY BYTES IN AN…EMPTY DICT?

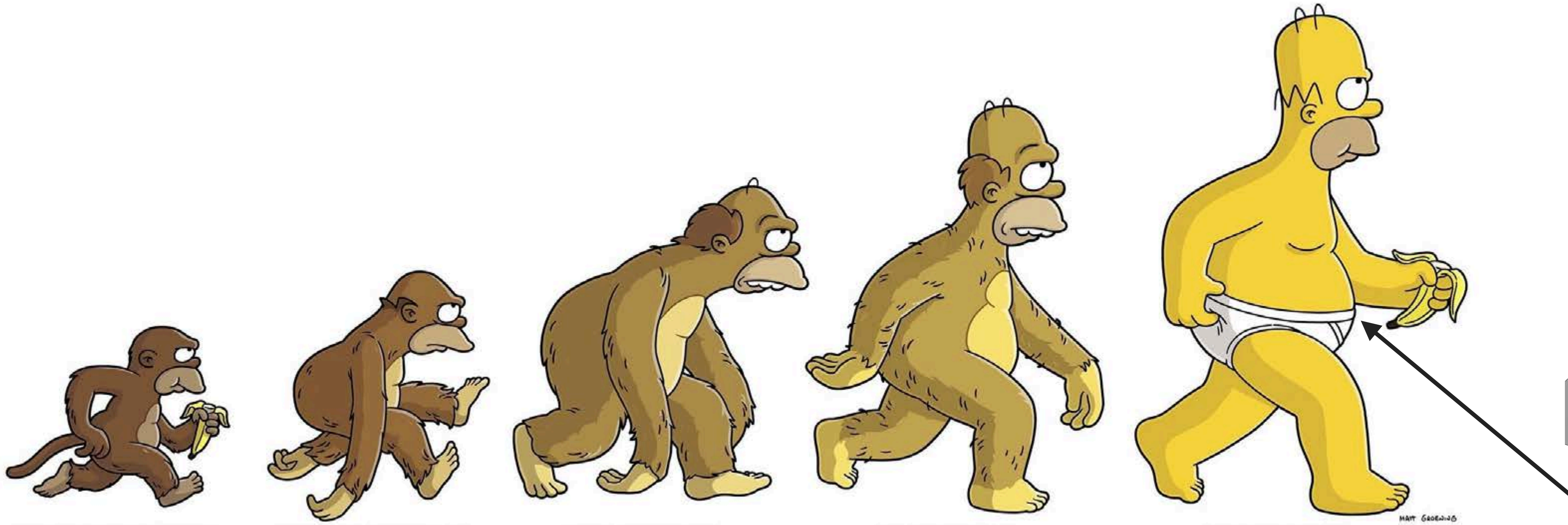
```
sizeof(map<int, int>)
→ 24
```



HOW MANY BYTES IN AN…EMPTY DICT?

```
sizeof(map<int, int>)
→ 24
```

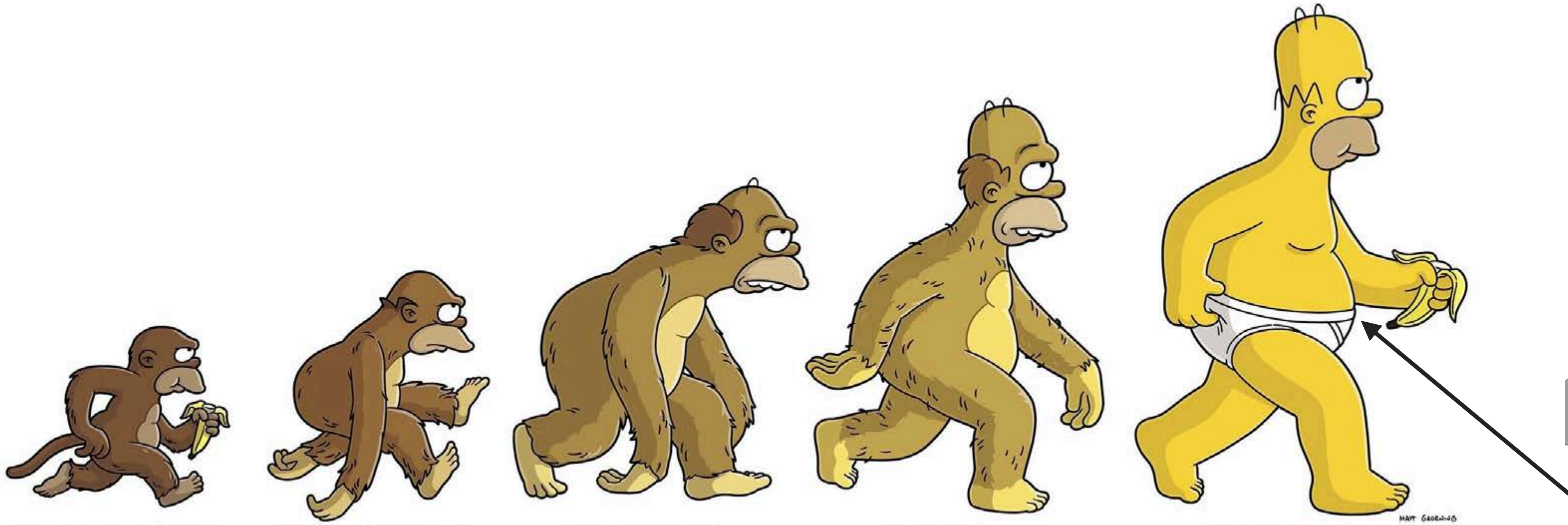
```
>>> sys.getsizeof({})
```



HOW MANY BYTES IN AN…EMPTY DICT?

```
sizeof(map<int, int>)
→ 24
```

```
>>> sys.getsizeof({})
64
```

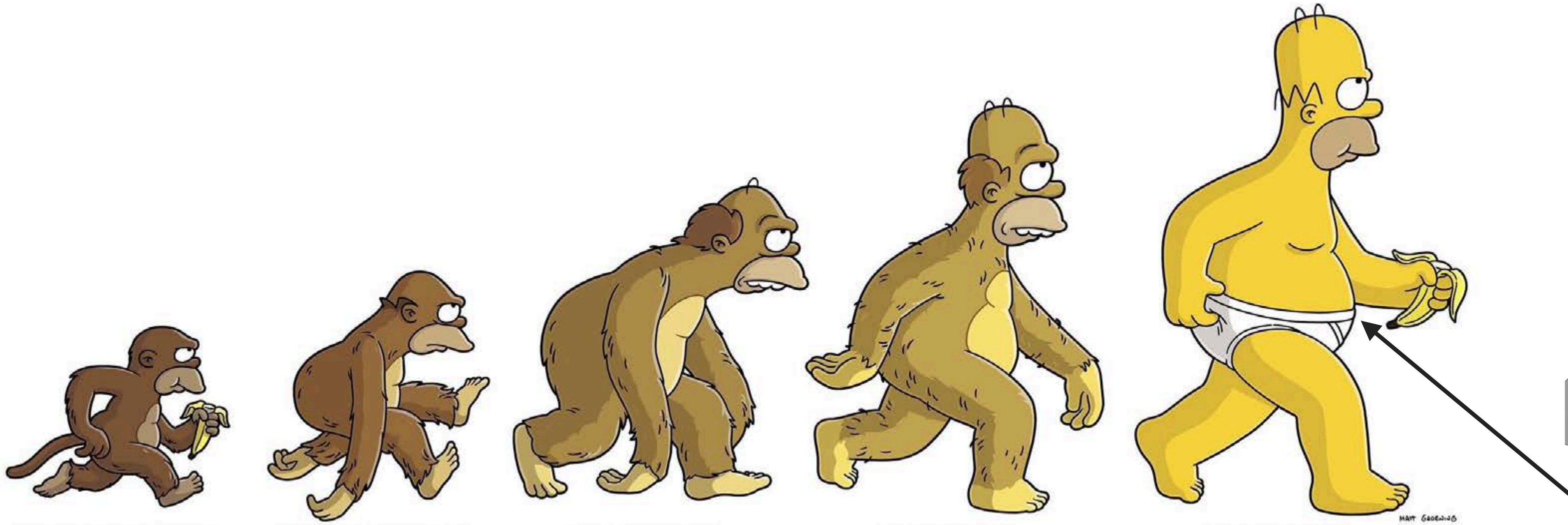


HOW MANY BYTES IN AN…EMPTY DICT?

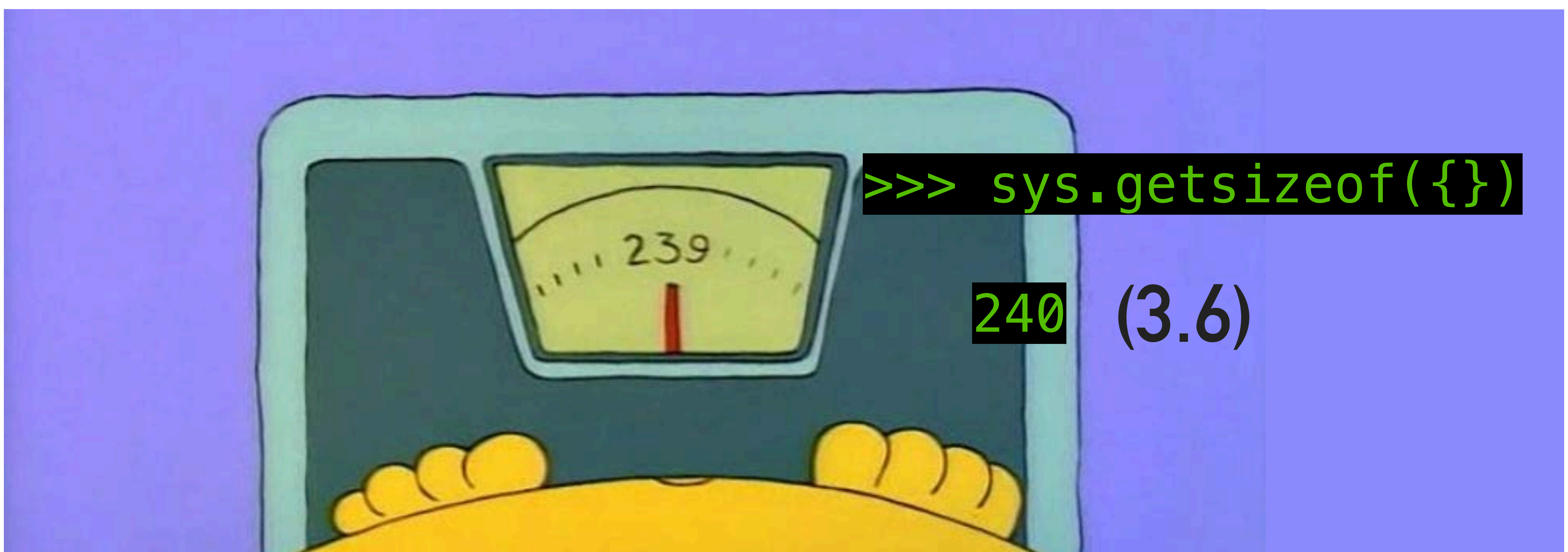
```
sizeof(map<int, int>)
→ 24
```

```
>>> sys.getsizeof({})
64
240
```

(3.6)

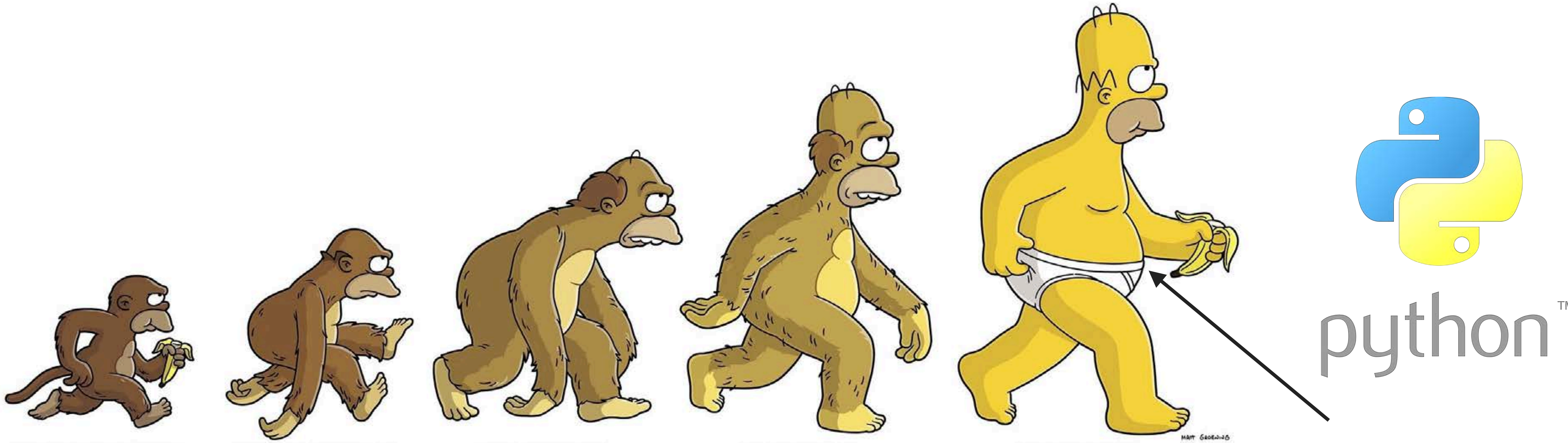


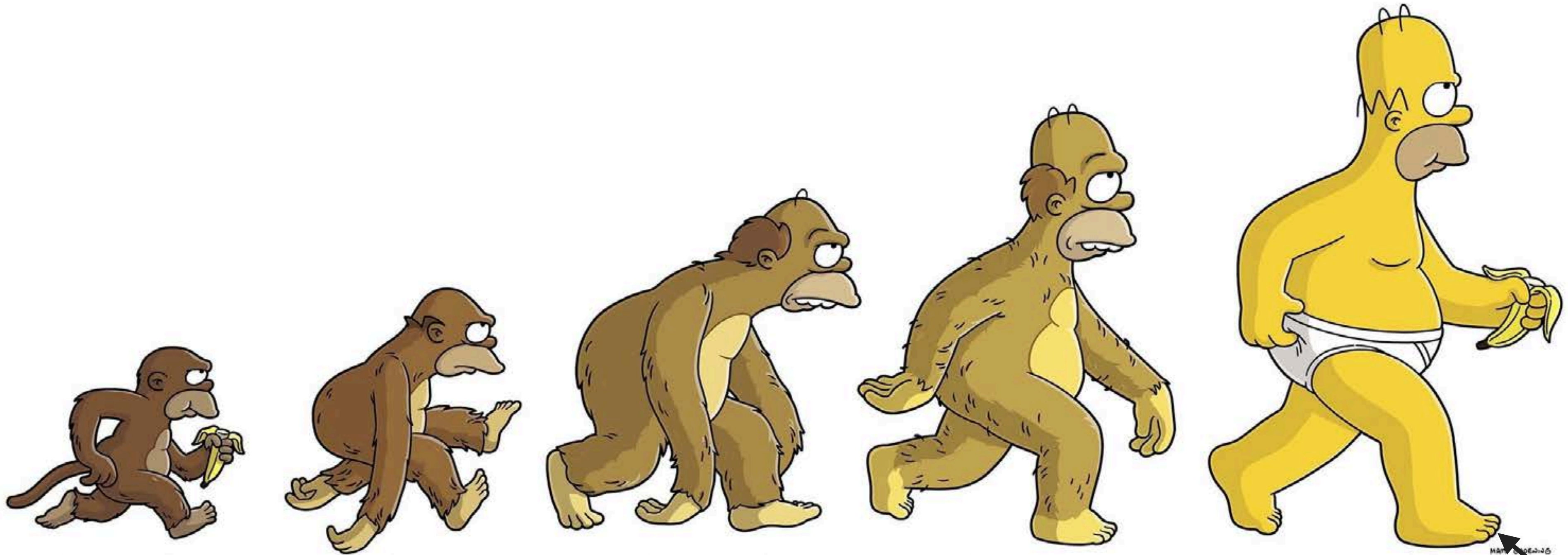
HOW MANY BYTES IN AN…EMPTY DICT?



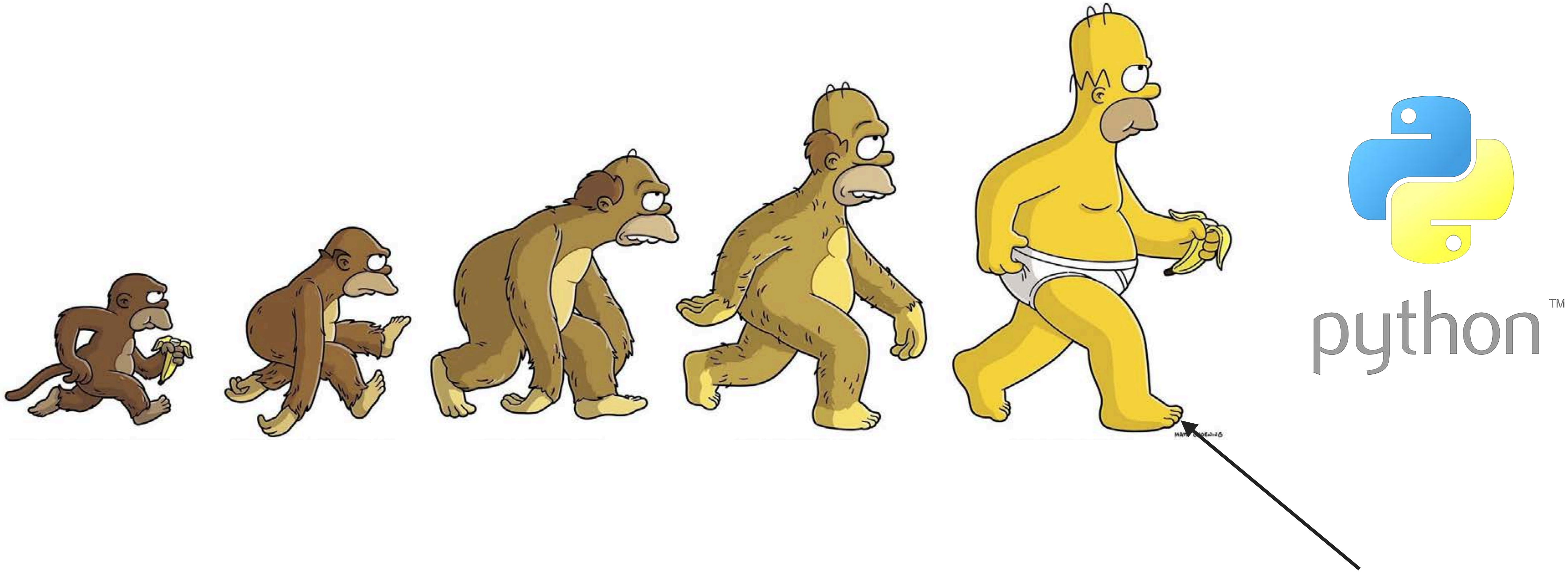

```
>>> sys.getsizeof({})
240
```

(3.6)

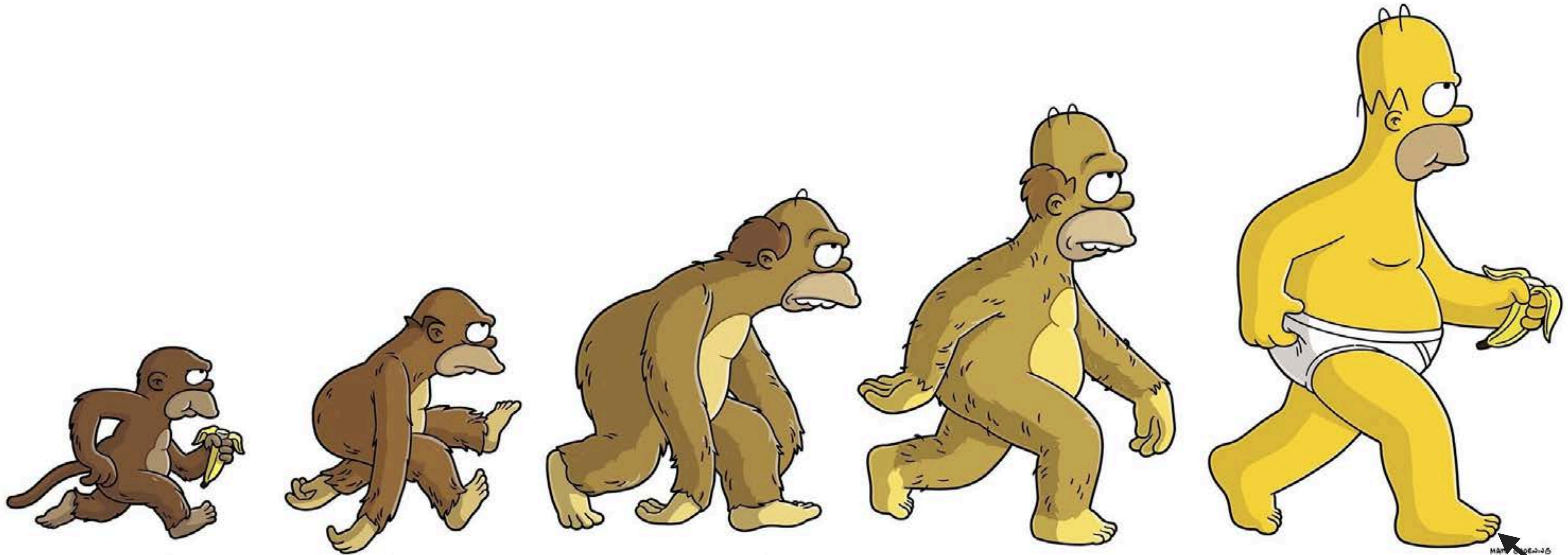

python



python™


python™

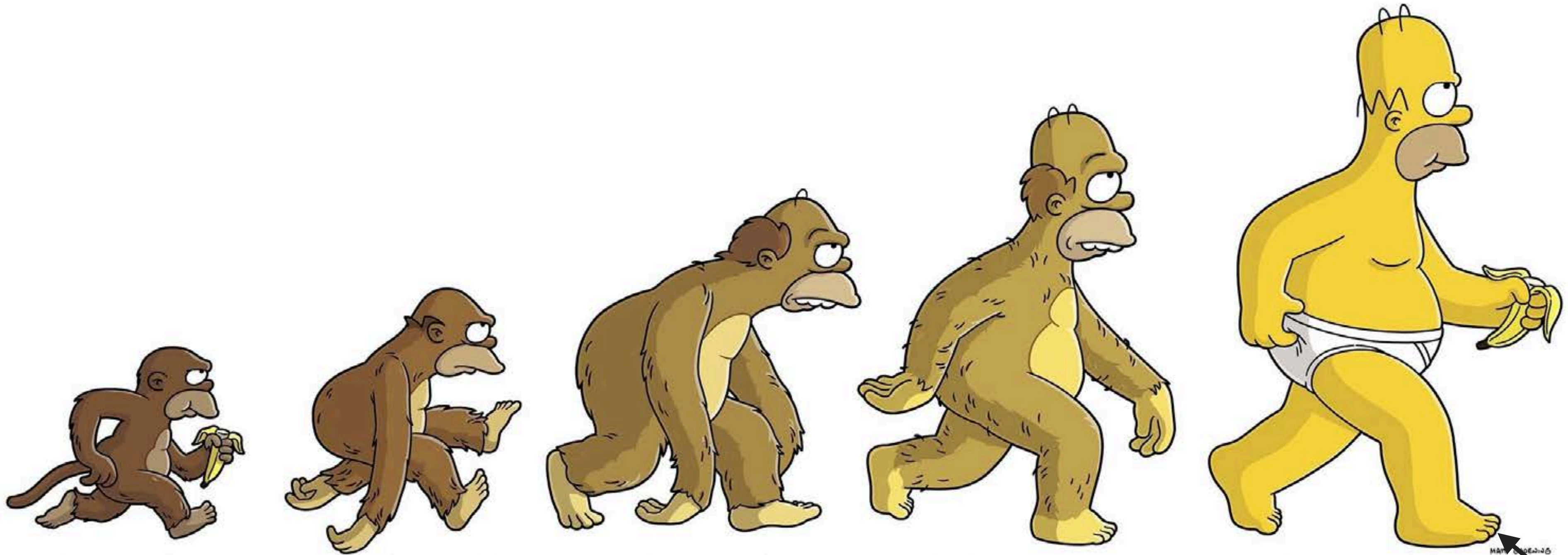

Something was said...not good.
What was it?
Slow!
They called you slow!

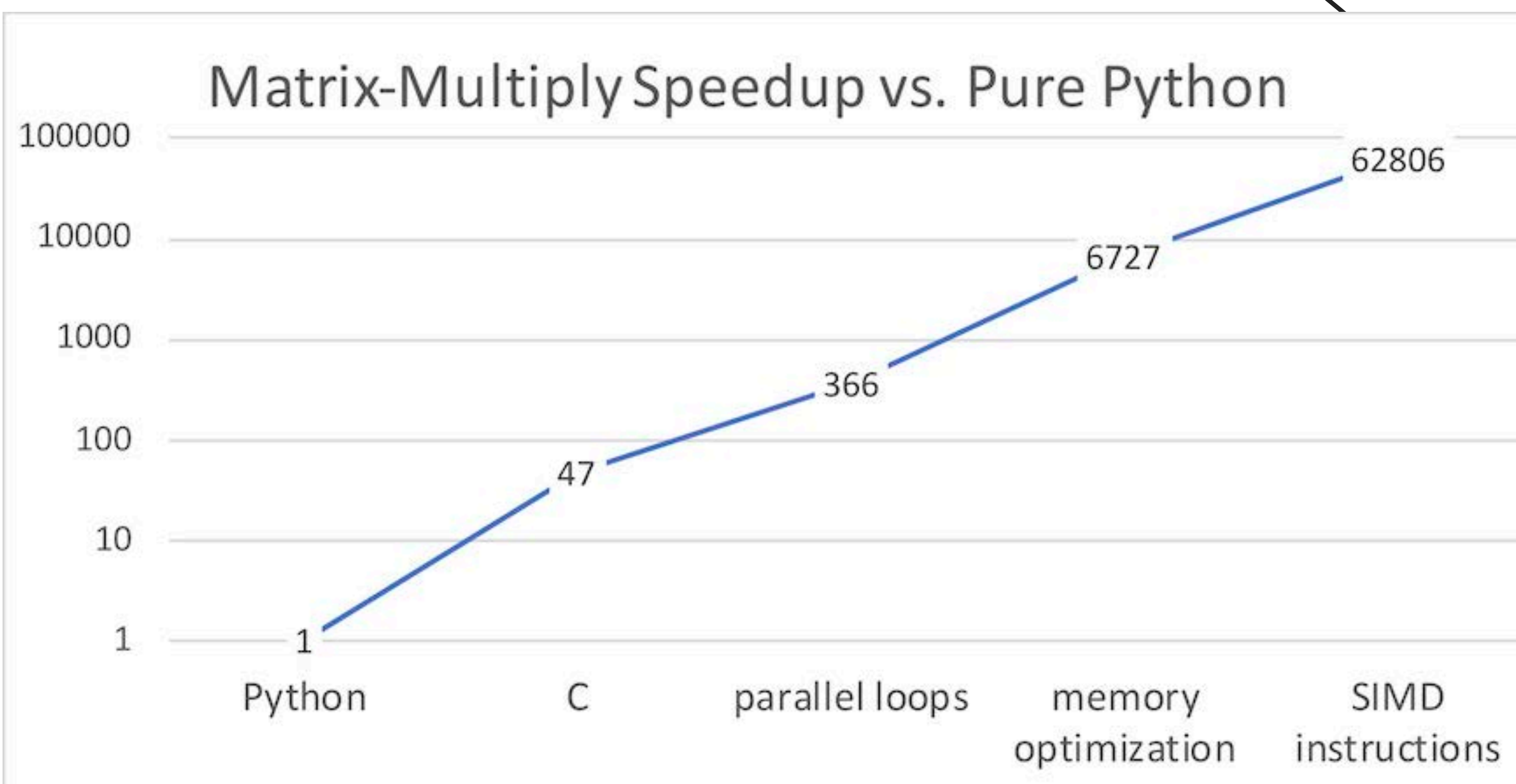


python™

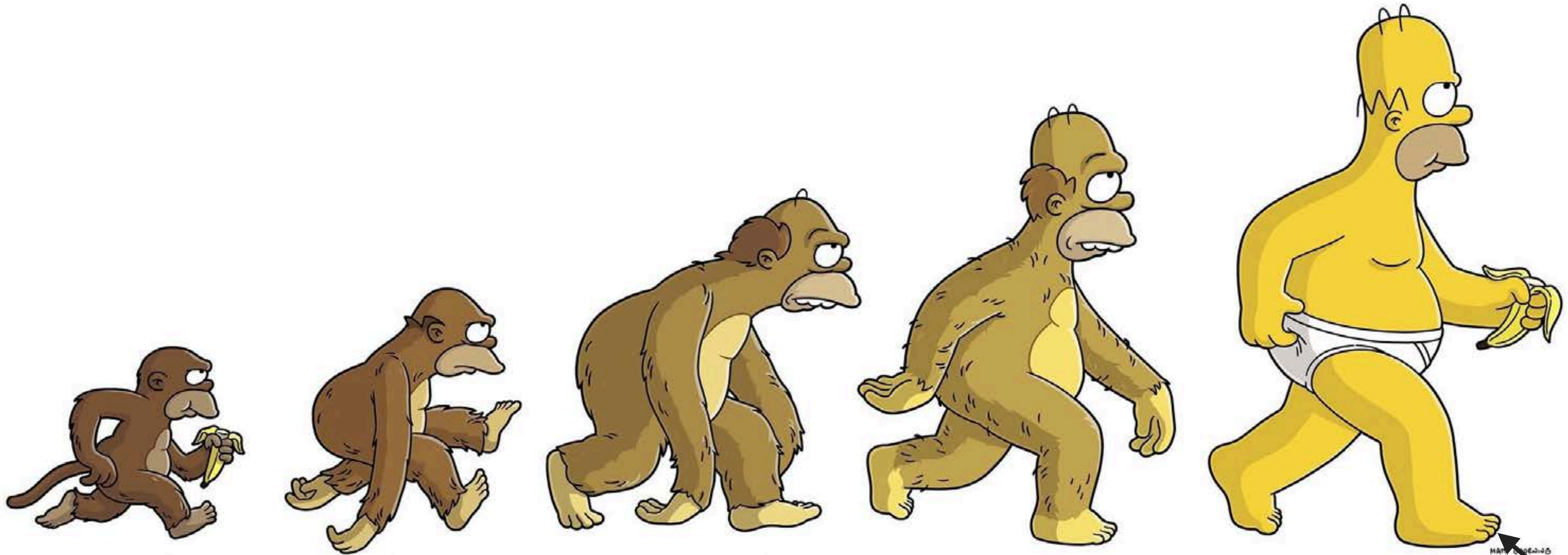
MATRIX-MATRIX MULTIPLY

```
for i in range(n):
  for j in range(n):
    for k in range(n):
      C[i][j] += A[i][k] * B[k][j]
```

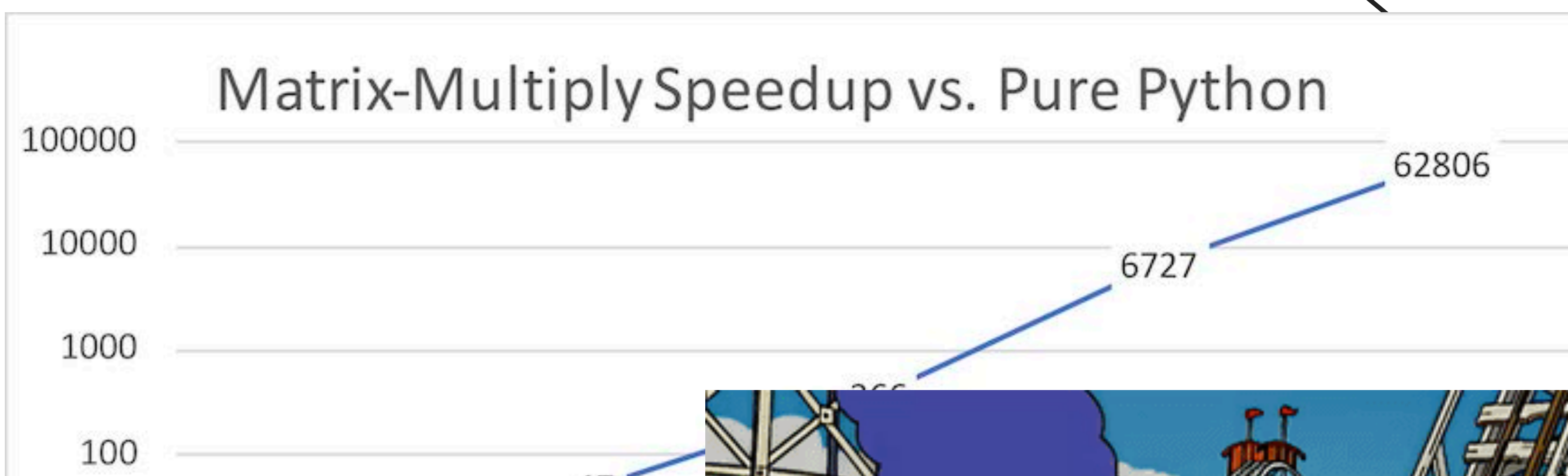




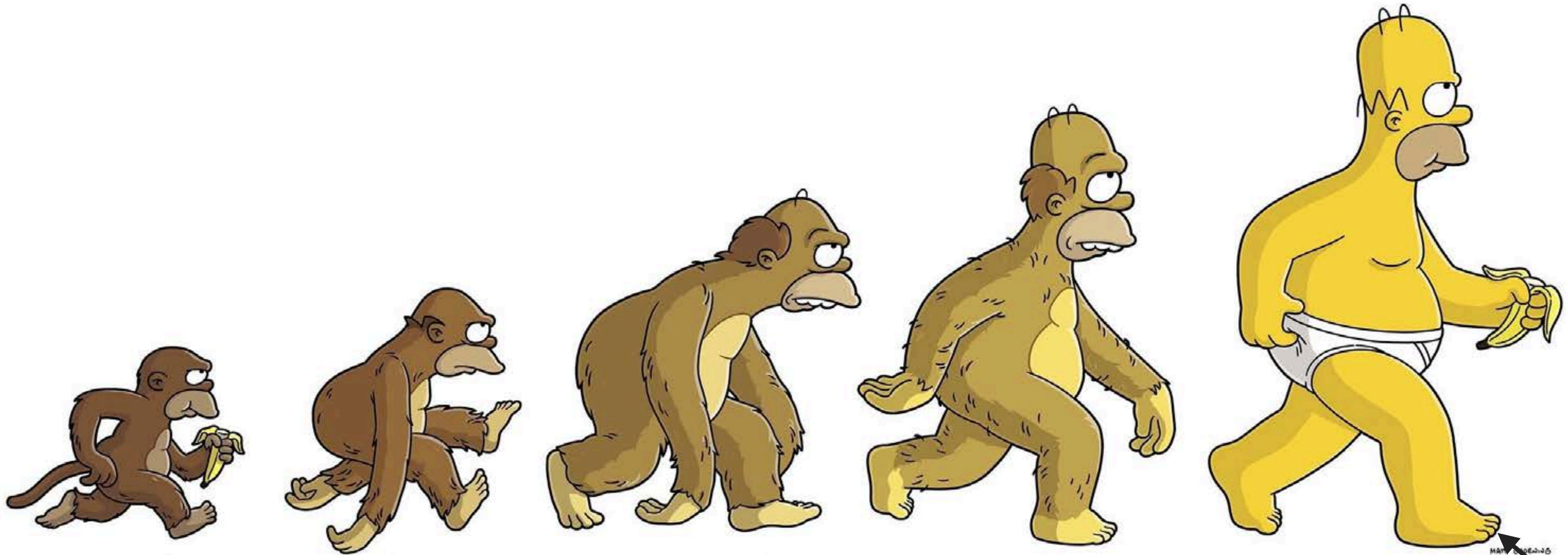
Matrix-Multiply Speedup vs. Pure Python
100000
10000
1000
100
10
1
62806
6727
366
47
1
Python
C
parallel loops
memory optimization
SIMD instructions



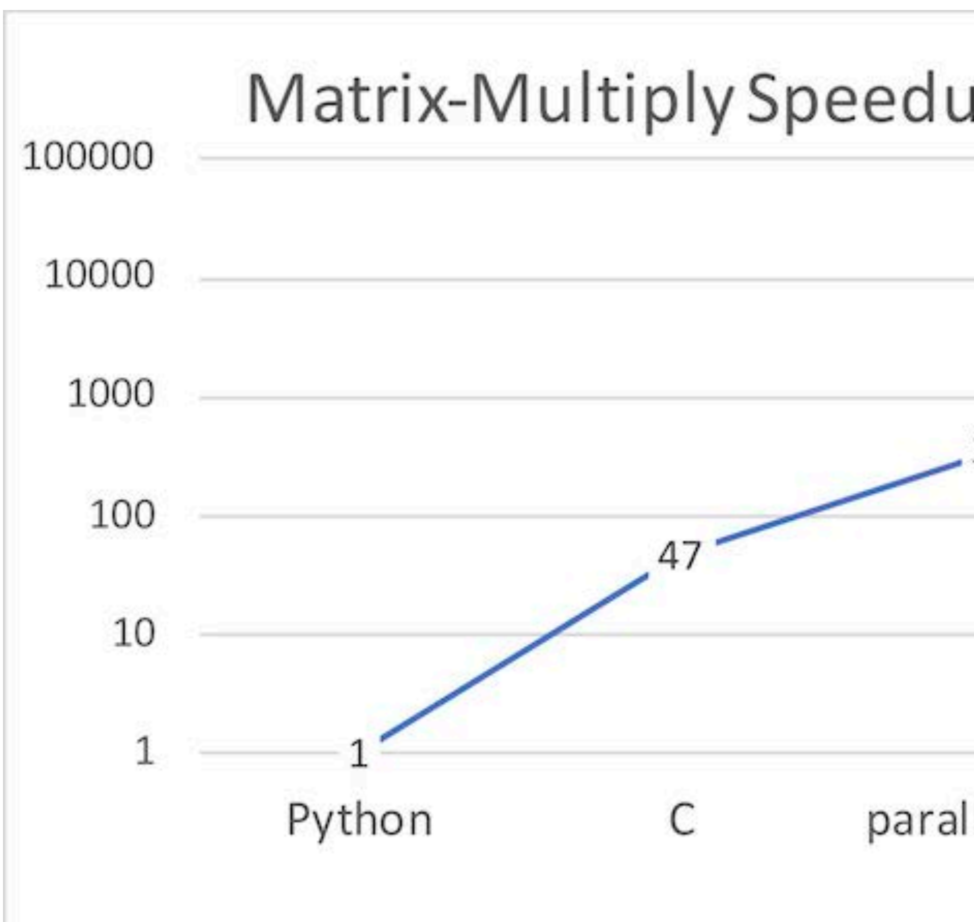
python™



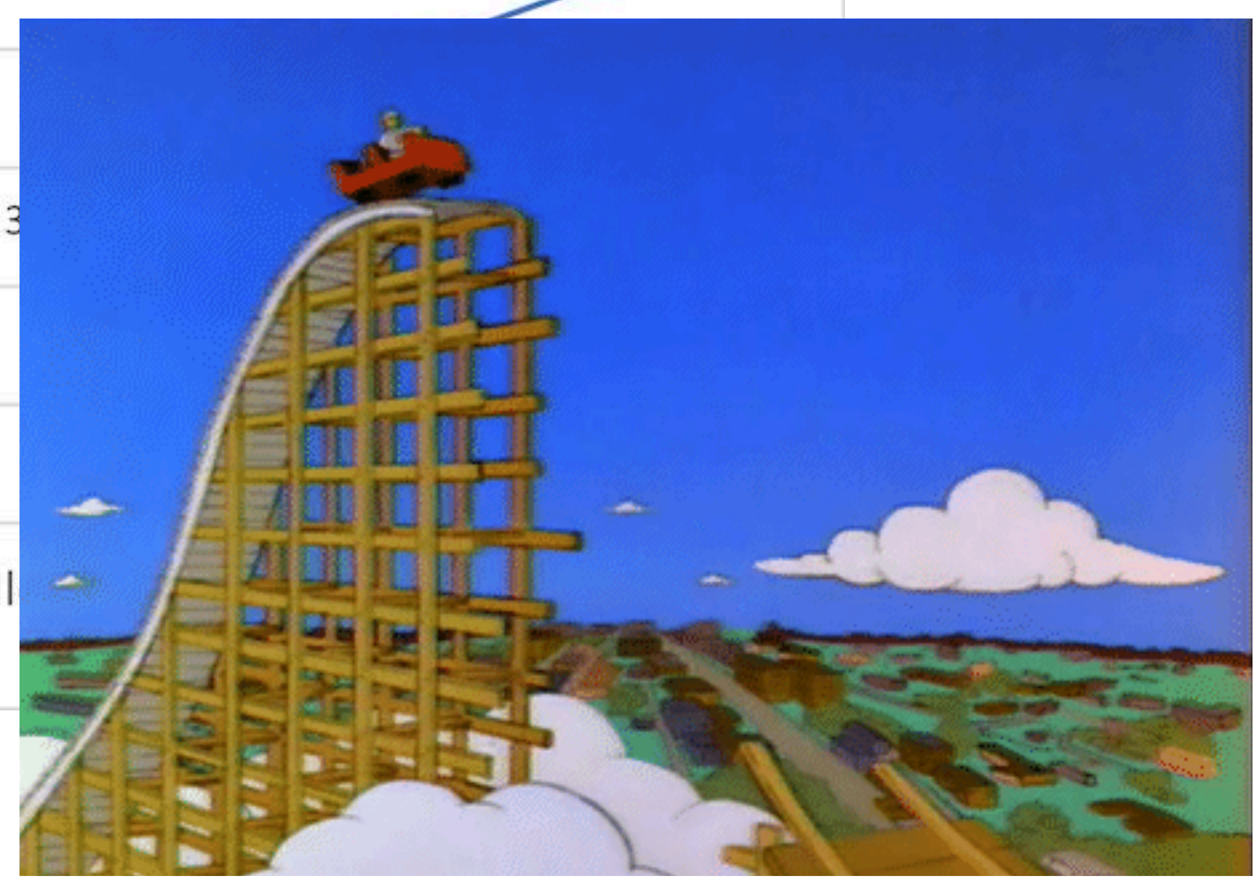
Matrix-Multiply Speedup vs. Pure Python
100000
10000
1000
100
10
1
62806
6727
47
1
Python
C



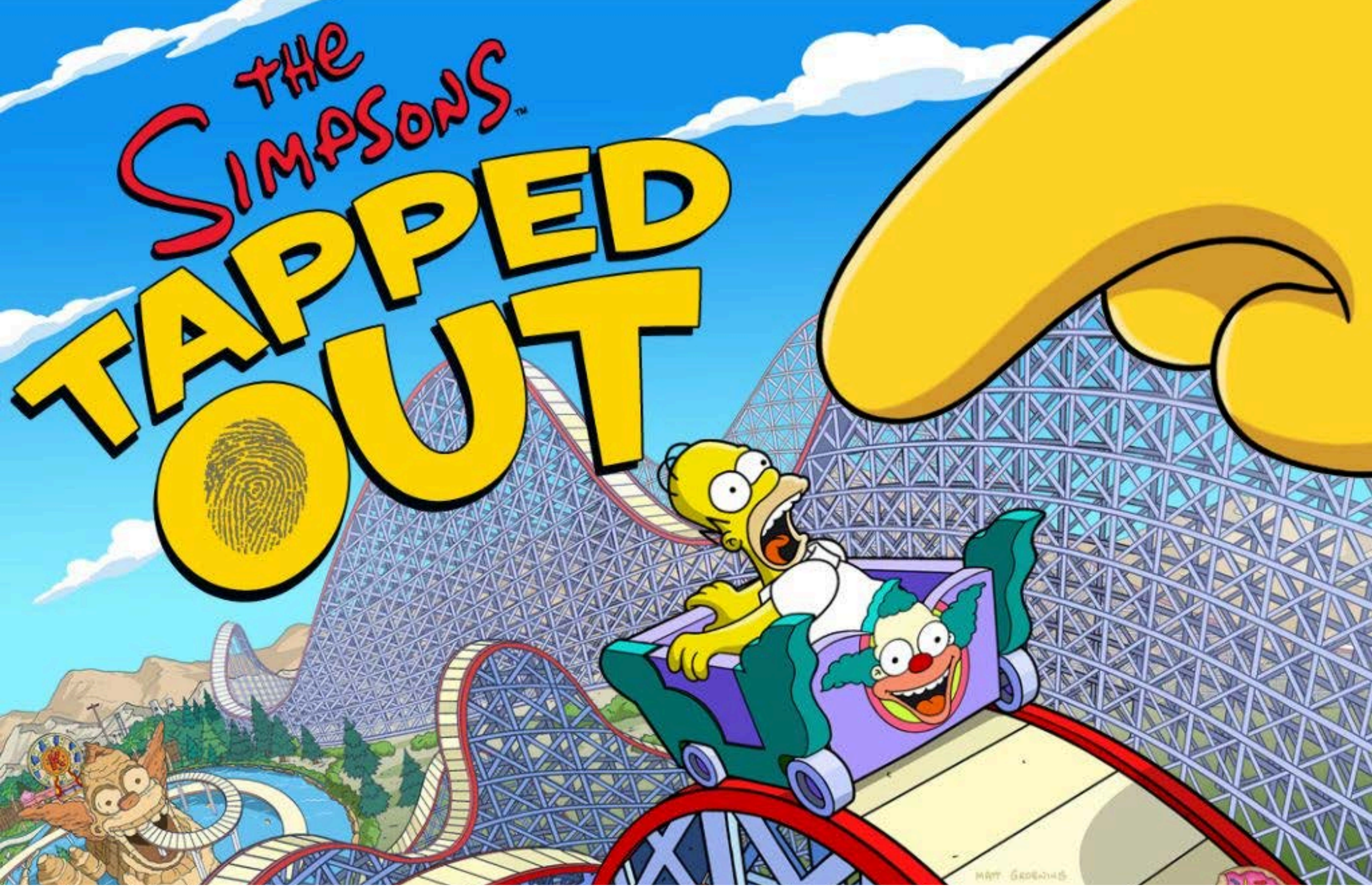
python™



Matrix-Multiply Speedup vs. Pure Python
100000
10000
1000
100
10
1
62806
47
1
Python
C

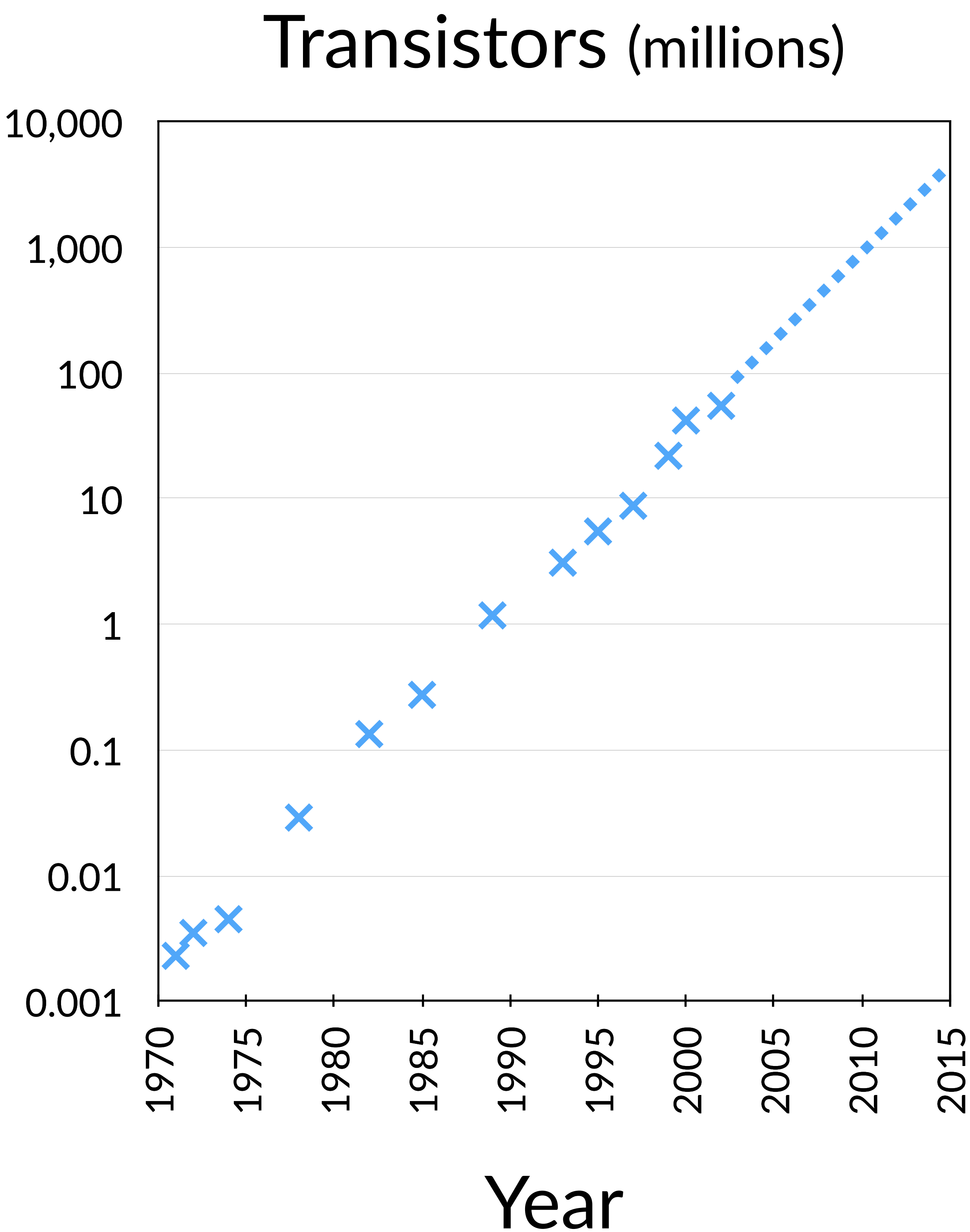


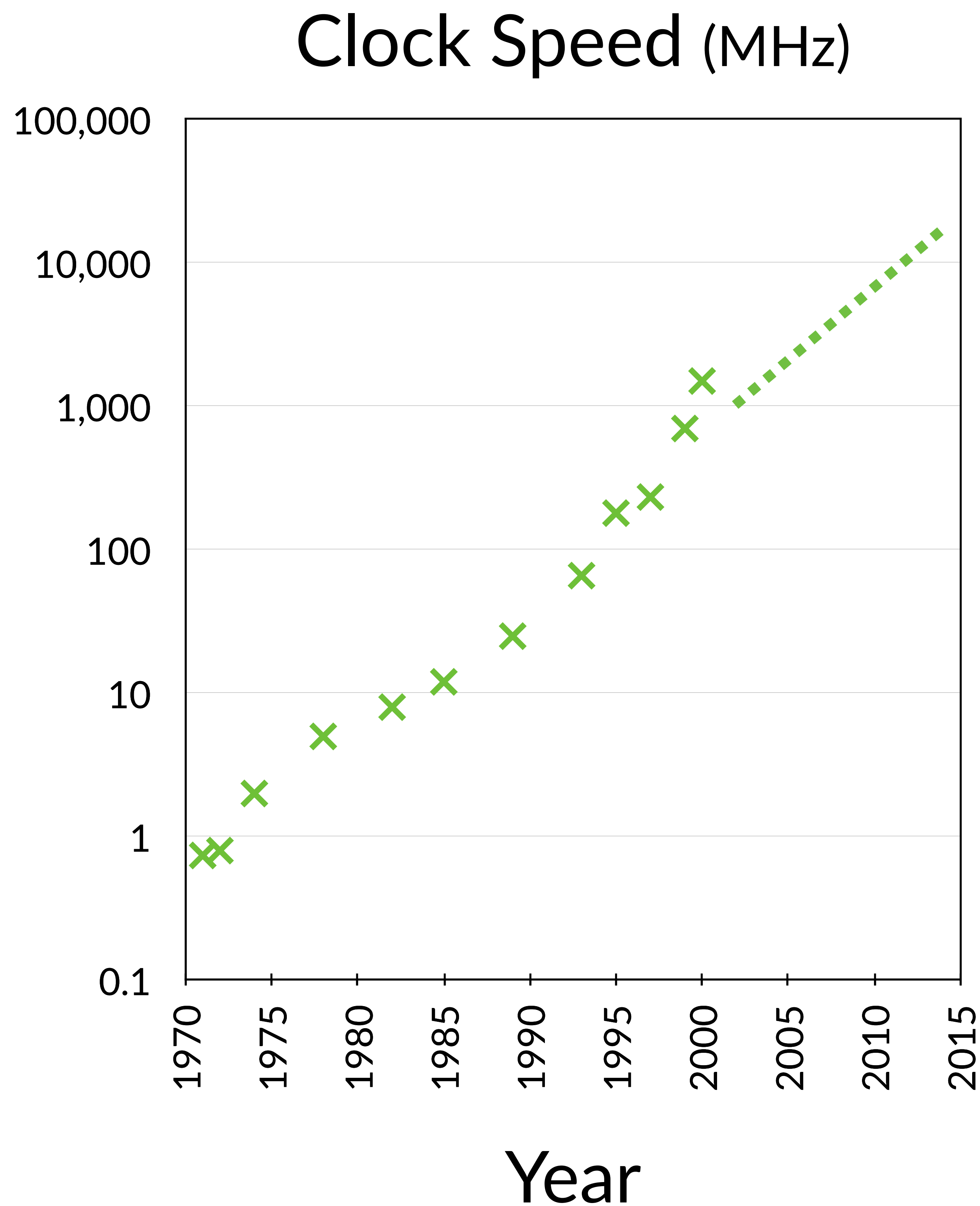

MATT
GROENING



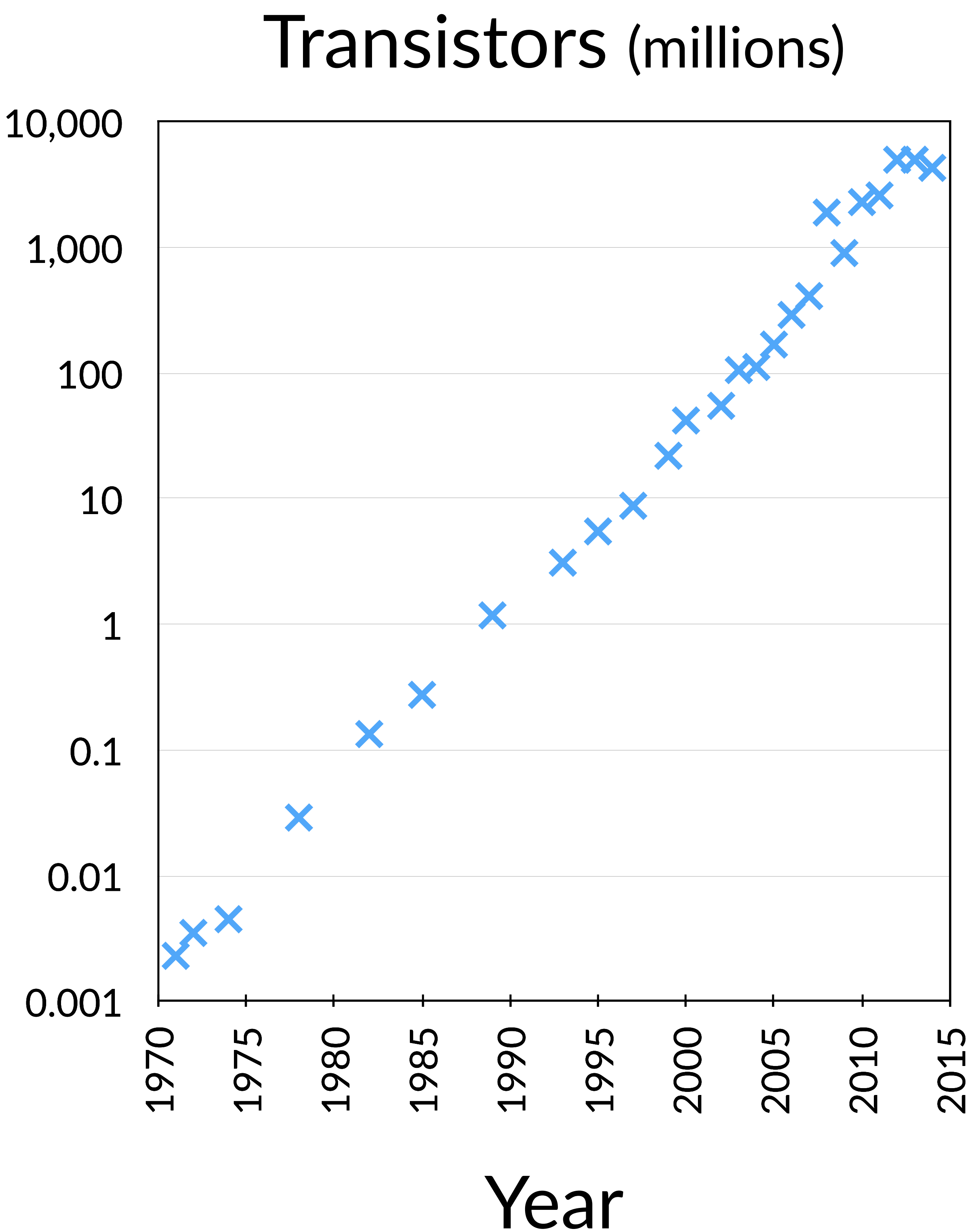
≈2010: THE RIDE IS OVER

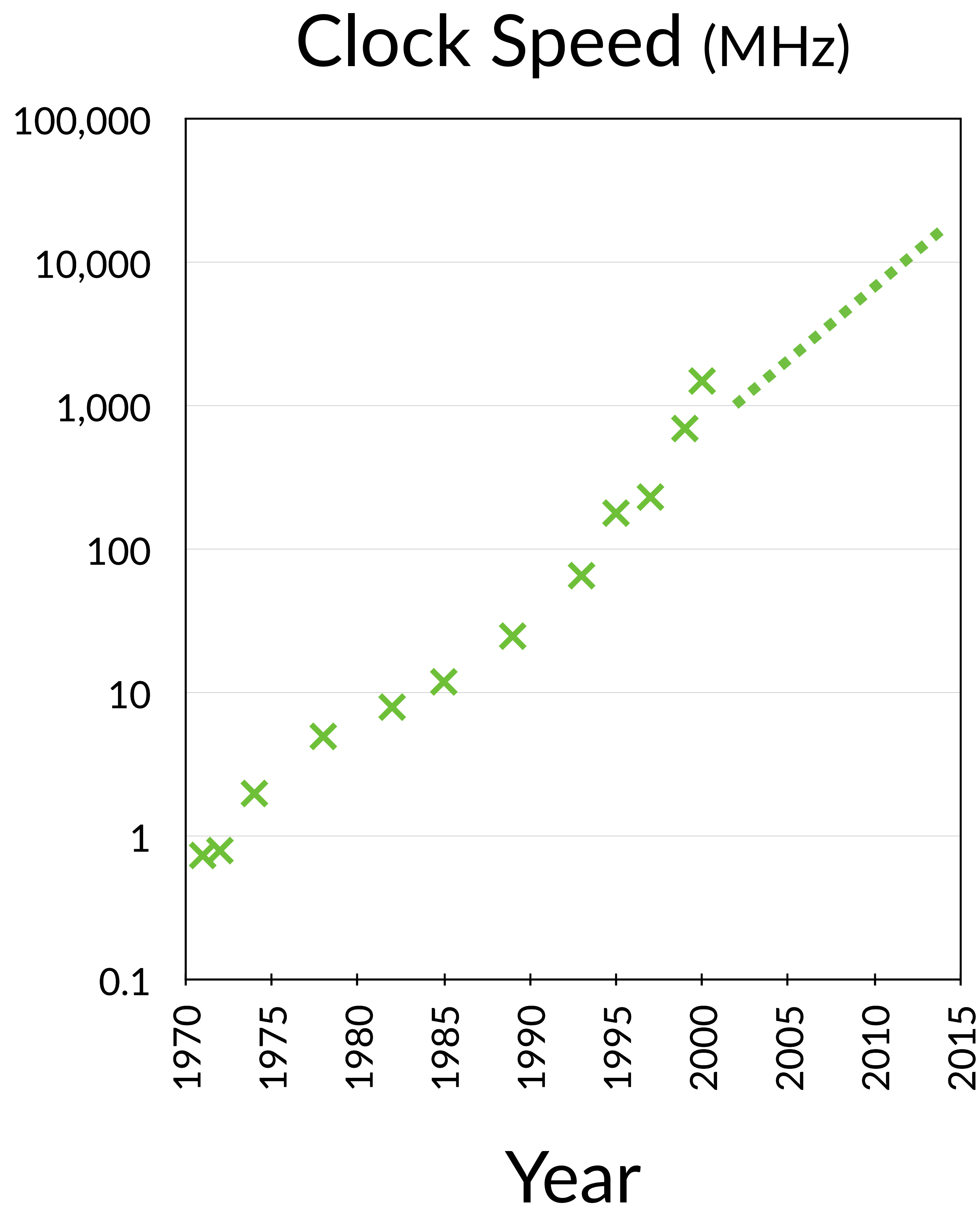
The Ride Is Over



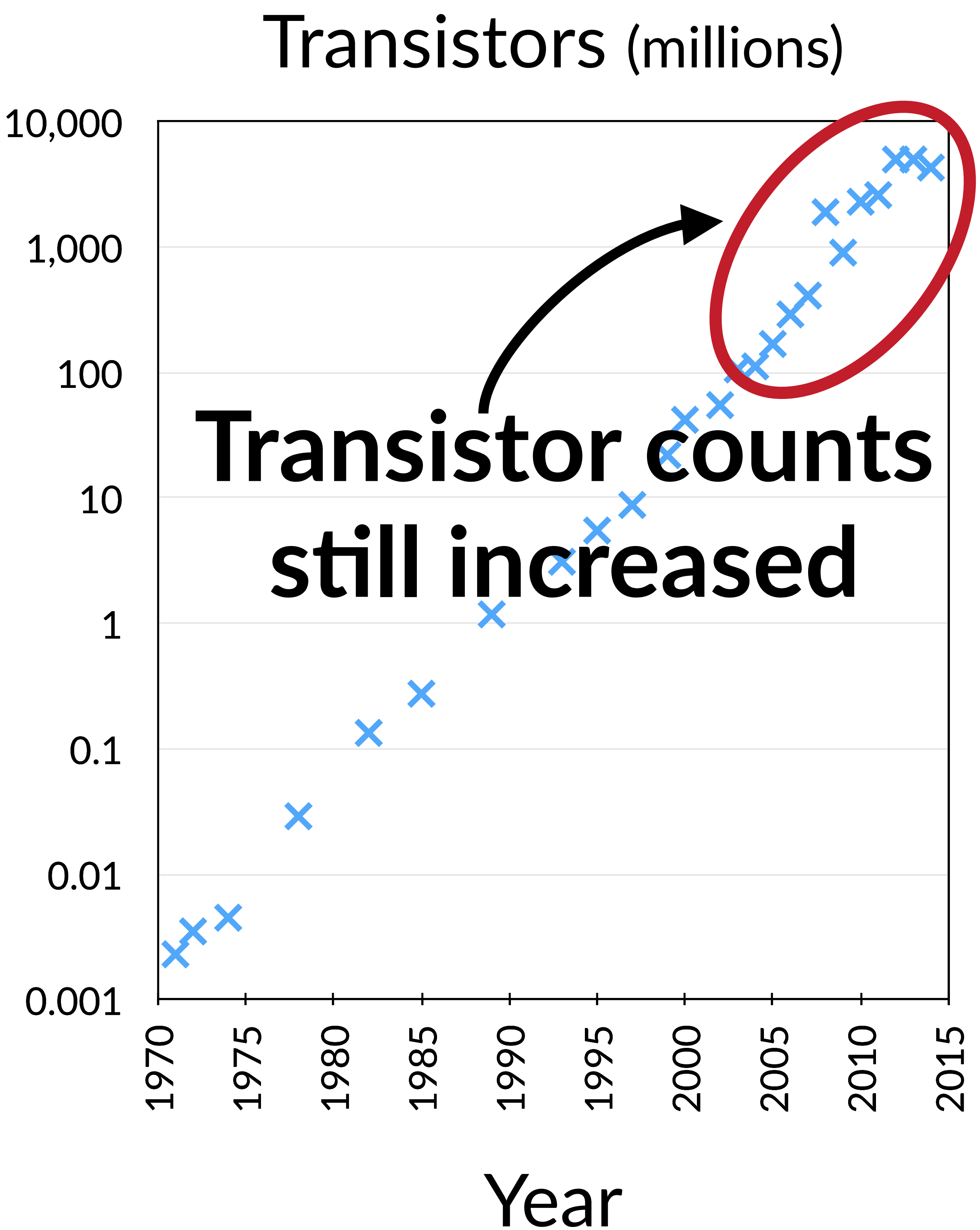


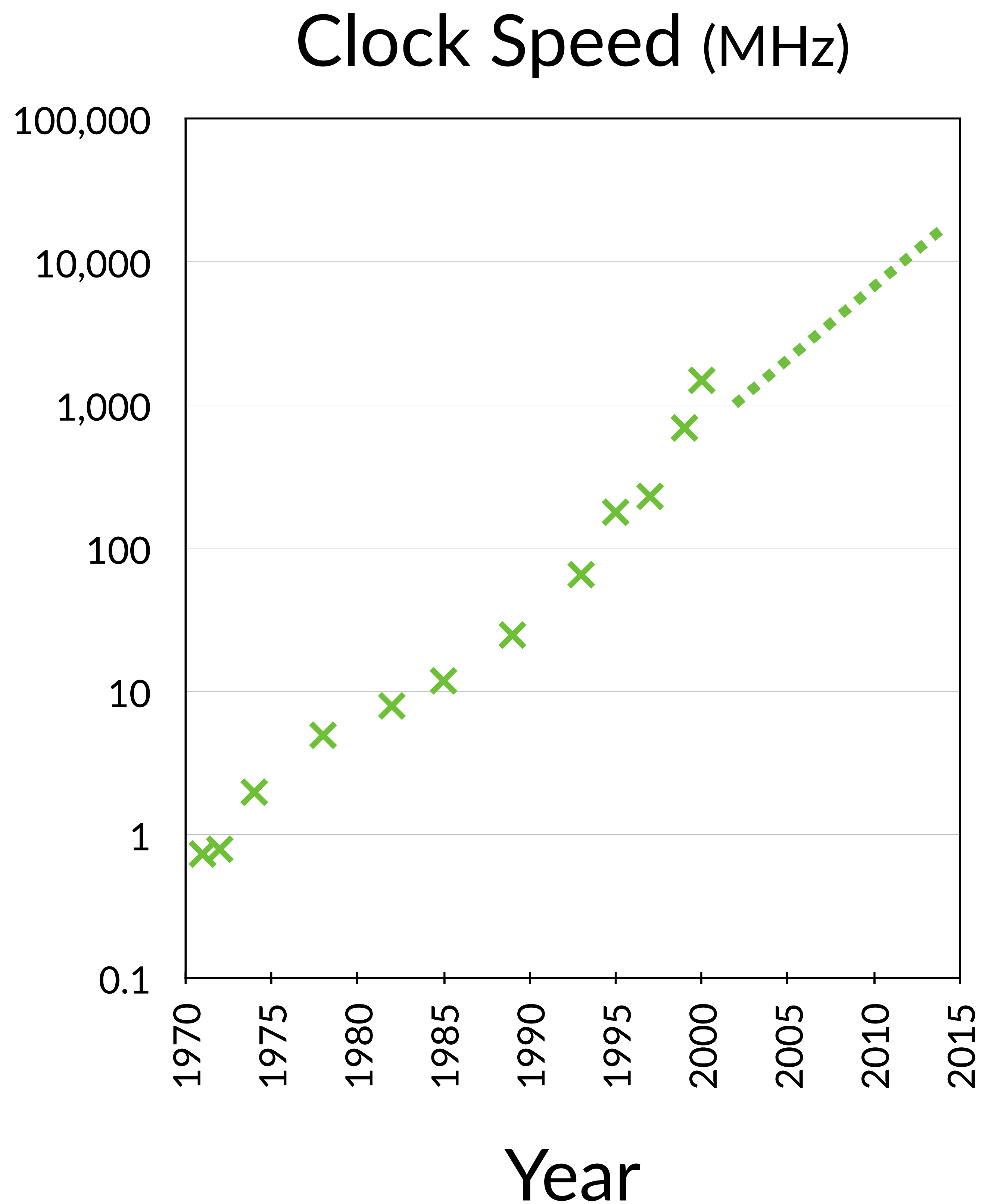
The Ride Is Over



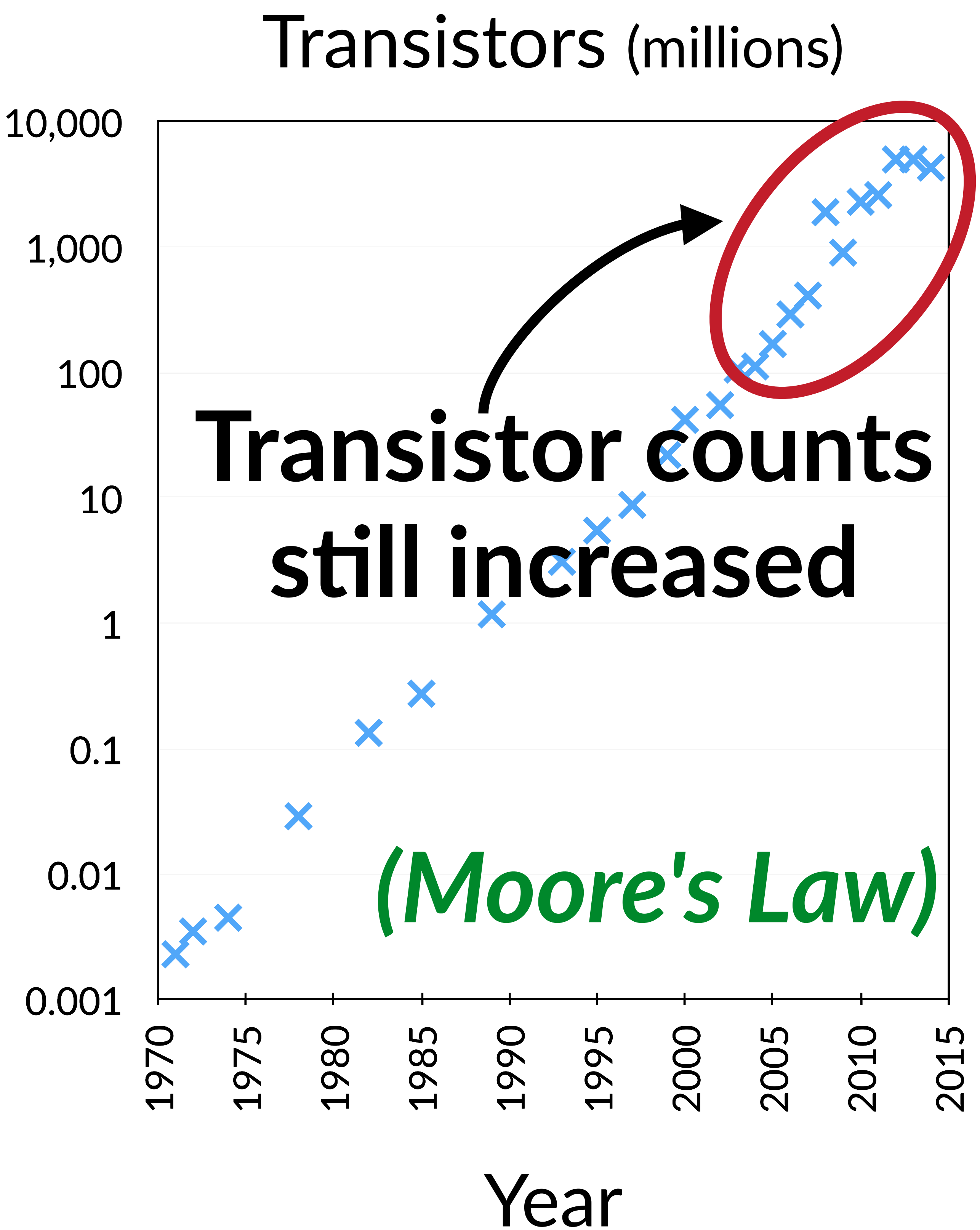


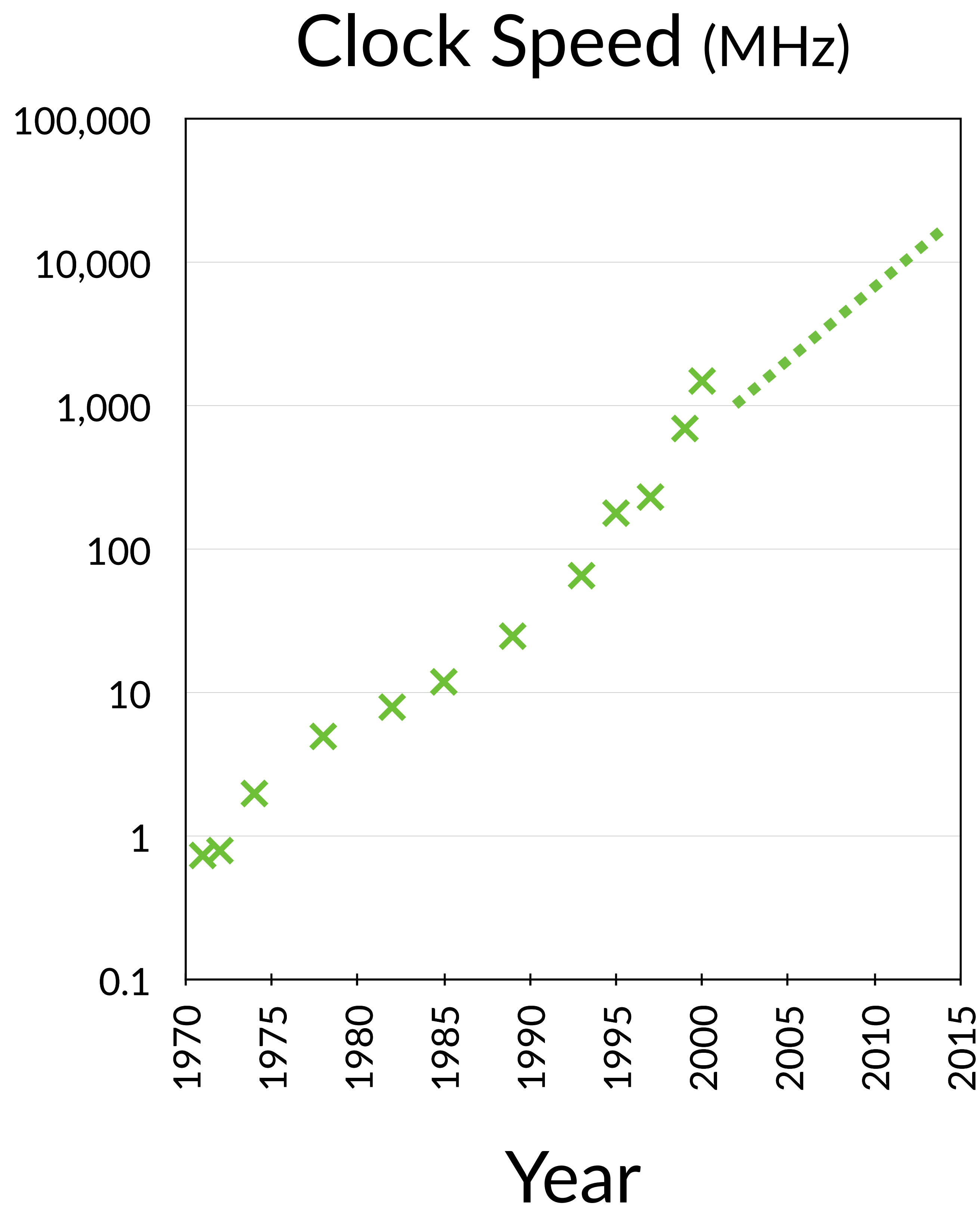
The Ride Is Over



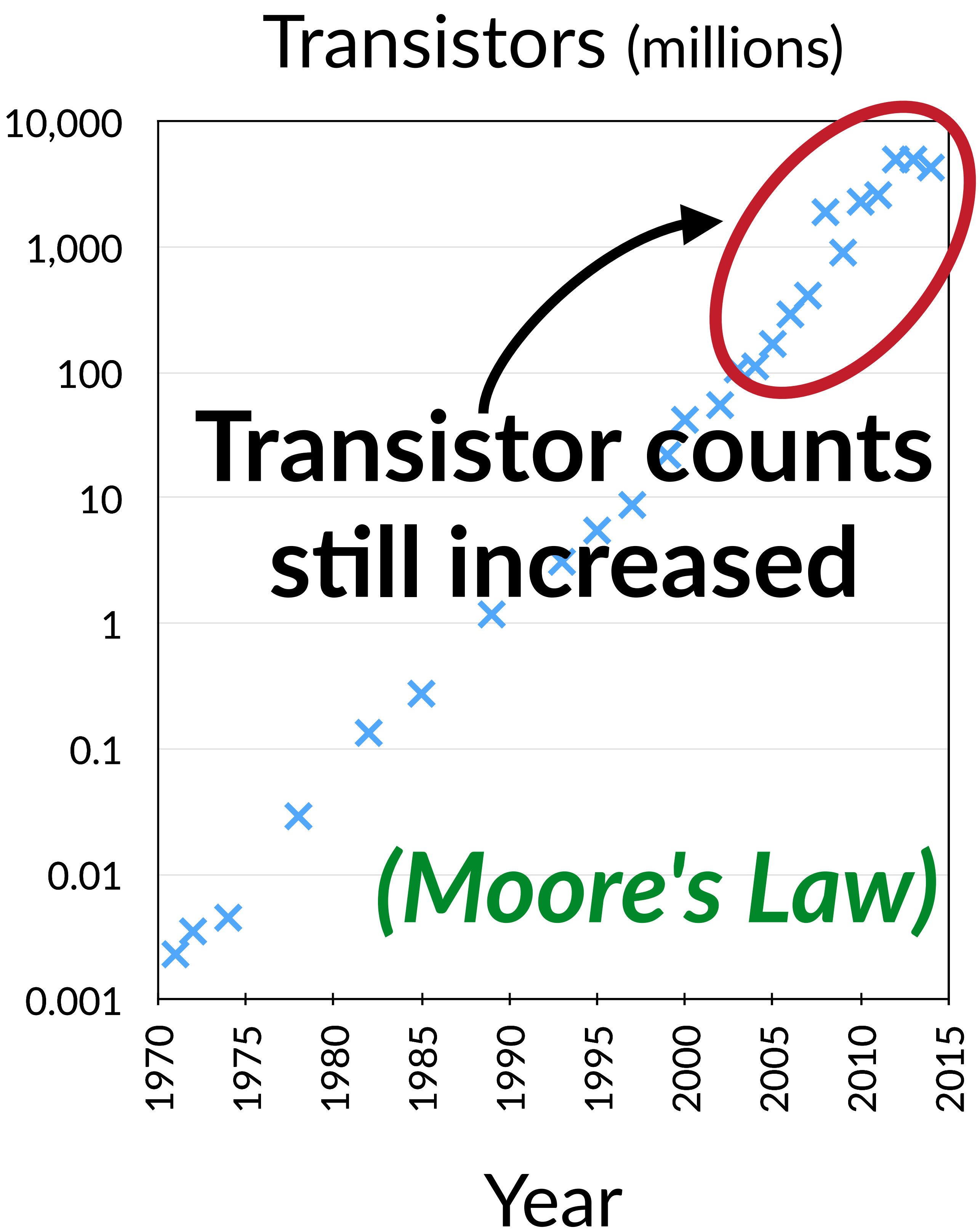


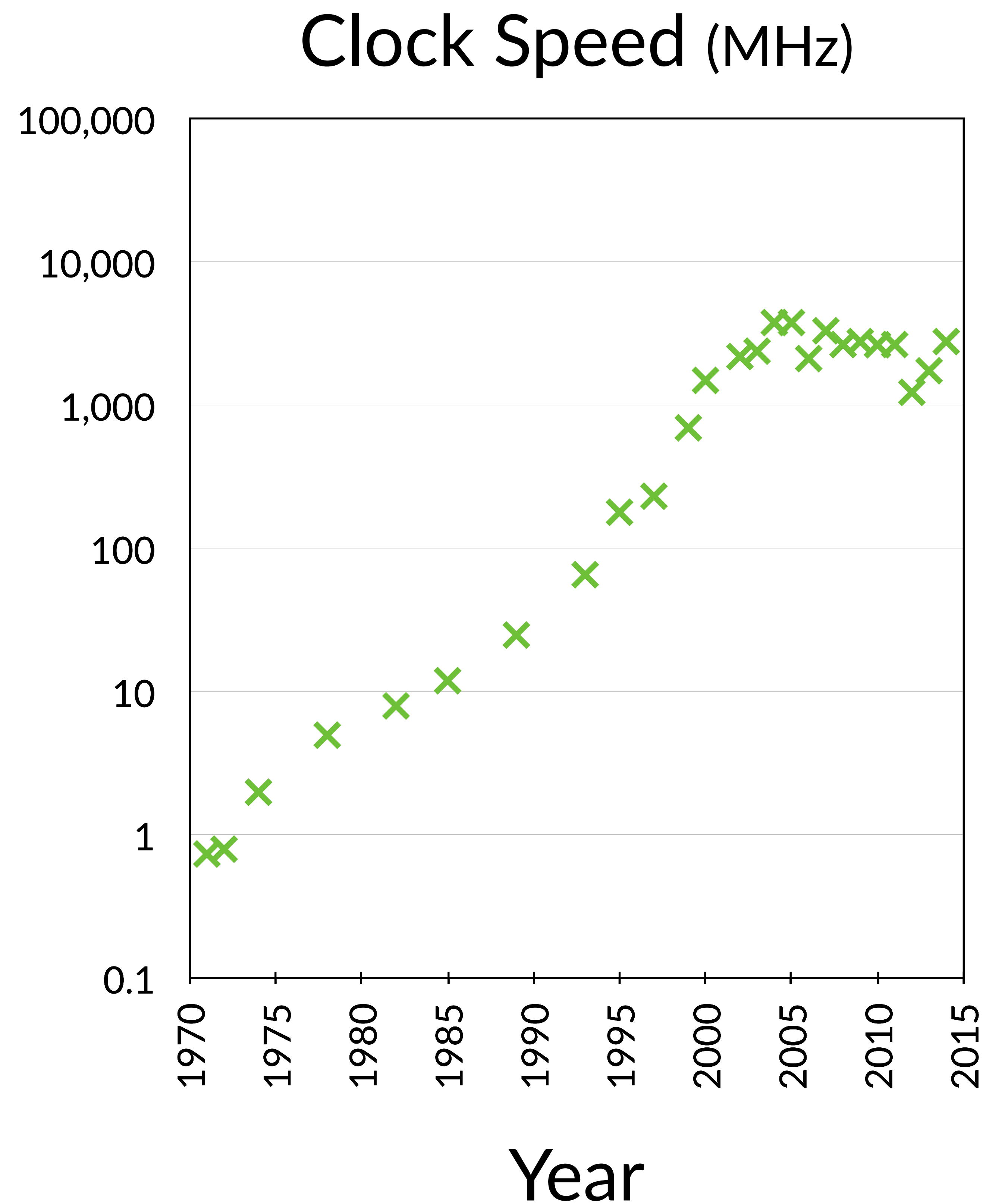
The Ride Is Over



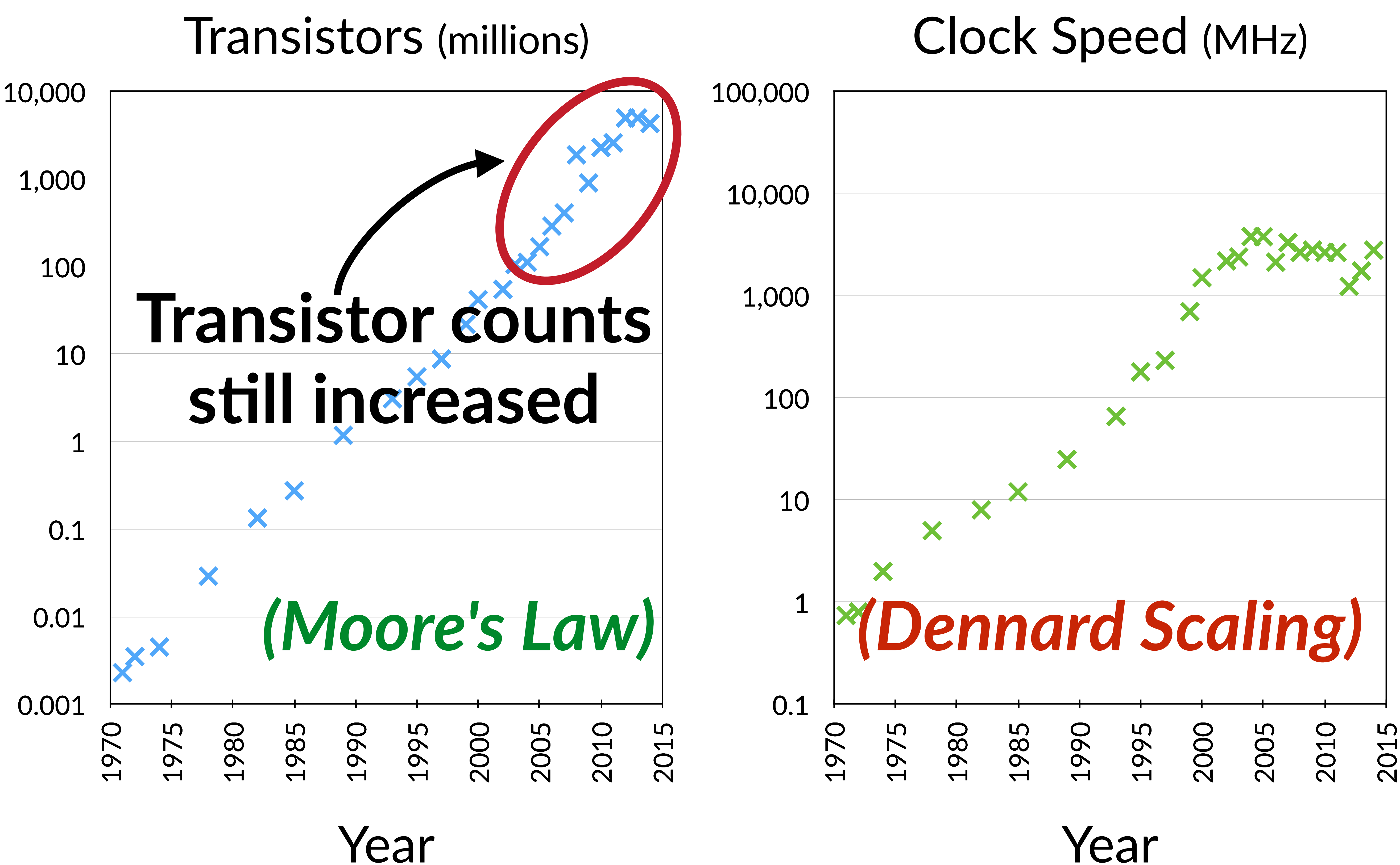


The Ride Is Over

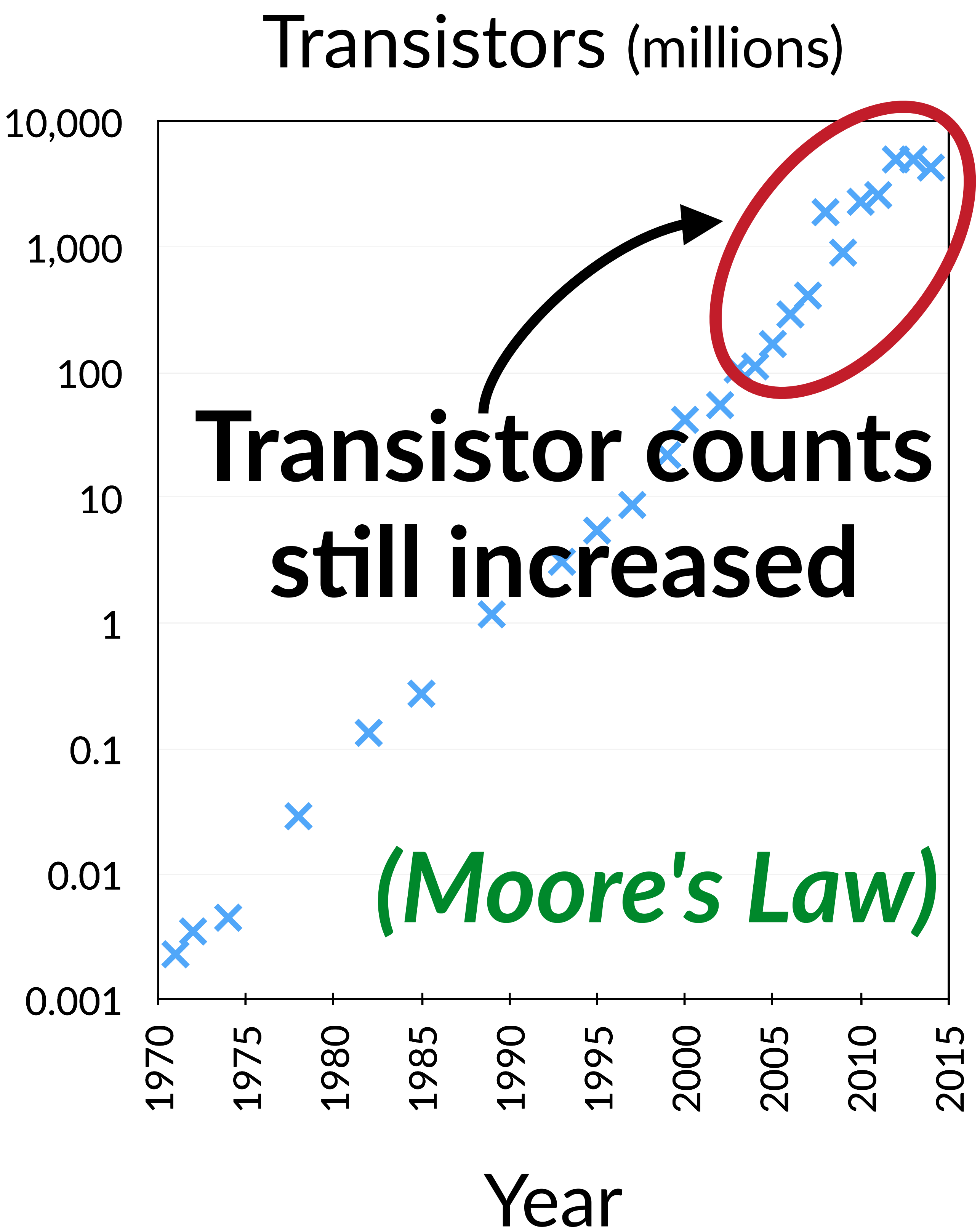


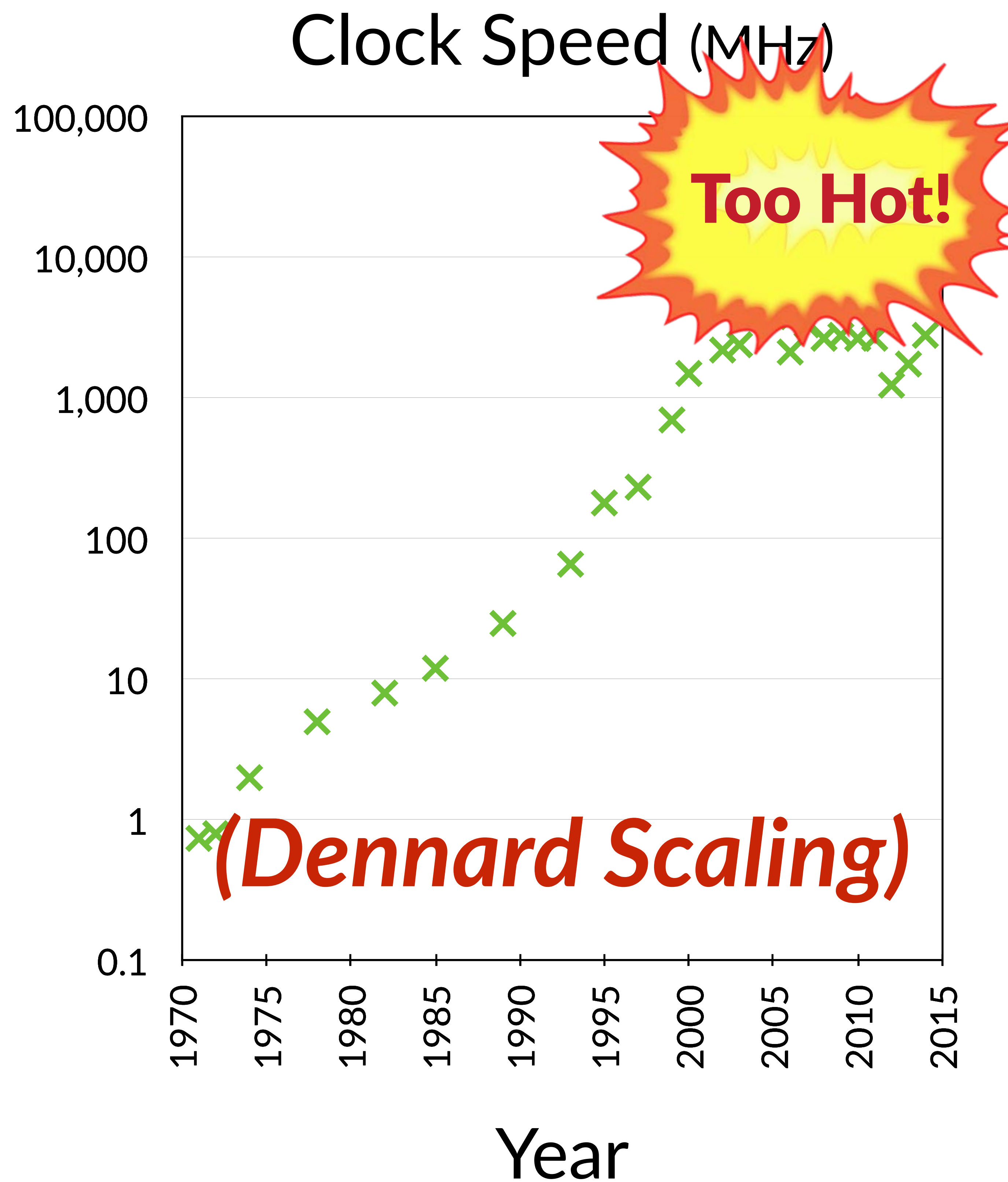


The Ride Is Over



The Ride Is Over



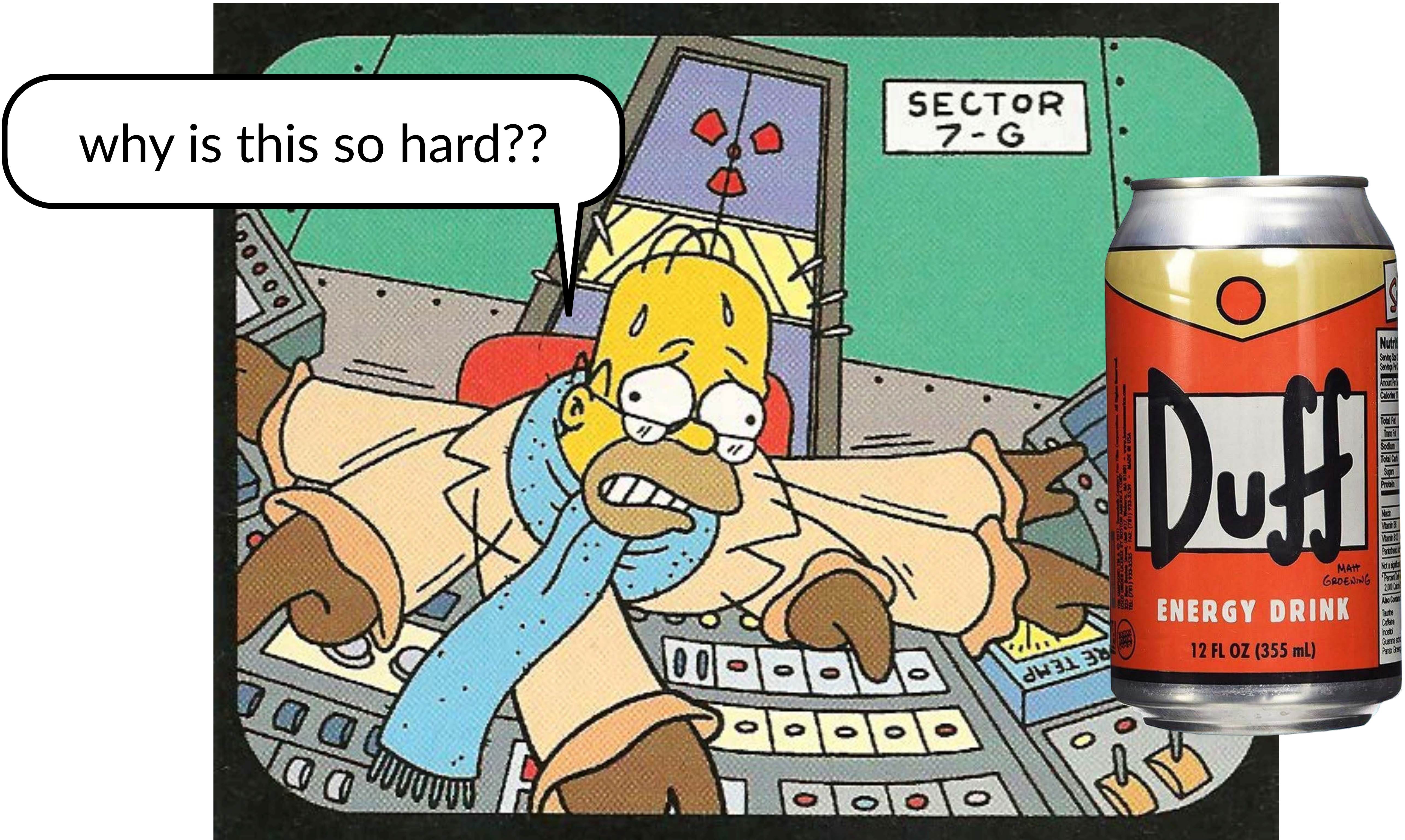


Performance in the ‘80s

just chill and wait for faster hardware!

Modern-day performance

Modern-day performance



Typical performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Typical performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Typical performance evaluation

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Typical performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

Typical performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Typical performance evaluation

A

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Typical performance evaluation

A

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    size_t meaning_of_life=42;
    for (size_t i = 0; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    for (size_t i = 16; i < size; i += 32) {
        asm("icbi 0,%0" : : "r"(p));
        p += 32;
    }
    asm("isync");
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Which is faster?

Which is faster?

Which is faster?

A — 90s

A′ — 87.5s

Which is faster?

A	A′
90s	87.5s

A′ **2.8% faster!**

Why is A′ faster than A?

Why is A′ faster than A?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

The code change!

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}

void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A?

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

The code change!

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A?

Or was it *an accident?*

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```


```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A?

Or was it *an accident?*

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```


```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

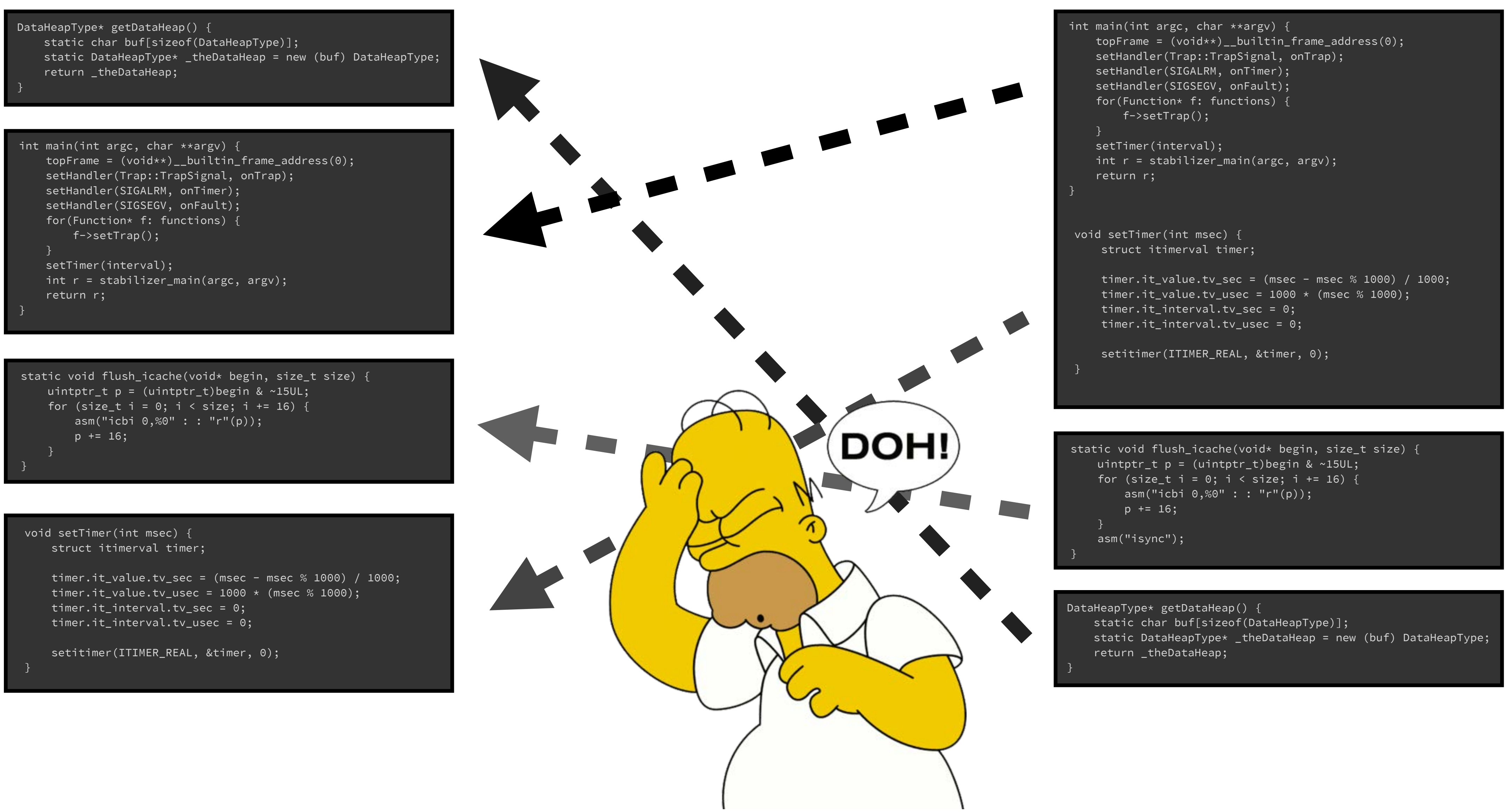
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A?

Layout biases measurement

Mytkowicz et al. (ASPLOS’09)



Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorado.edu

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorado.edu

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
Matthias.Hauswirth@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com

Link Order

Changes function addresses

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Producing Wrong Data Without Doing Anything Obviously Wrong!

Todd Mytkowicz Amer Diwan
Department of Computer Science
University of Colorado
Boulder, CO, USA
{mytkowit,diwan}@colorad[illegible]du

Matthias Hauswirth
Faculty of Informatics
University of Lugano
Lugano, CH
M[illegible]tthias.Hau[illegible]@unisi.ch

Peter F. Sweeney
IBM Research
Hawthorne, NY, USA
pfs@us.ibm.com

Link Order

Changes function addresses

Environment Variable Size

Moves the program stack

Layout biases measurement

Mytkowicz et al. (ASPLOS'09)

Link Order

Changes function addresses

Larger than the impact of -O3! (±40%)

Environment Variable Size

Moves the program stack

Why is A′ faster than A?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

Why is A′ faster than A?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same cache set

Why is A′ faster than A?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same
cache set
conflict

Why is A′ faster than A?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Why is A′ faster than A?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Why is A′ faster than A ?

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to
same set

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

Why is A′ faster than A?

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same set

Nothing here

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
void setTimer(int msec) {
    struct itimerval timer;

    timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
    timer.it_value.tv_usec = 1000 * (msec % 1000);
    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, 0);
}
```

Why is A′ faster than A?

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

map to same set

Nothing here

no conflict

A′

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

It’s the cache

A

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

A’

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

It’s the cache
^ or branch predictor

A

A’

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

It’s the cache

or TLB

or branch predictor

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

It's the cache

or TLB

or branch predictor

or branch target predictor

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}


 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

It's the cache

or TLB

or branch predictor

or branch target predictor

or prefetcher

A

A′

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
    asm("isync");
}
```

```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

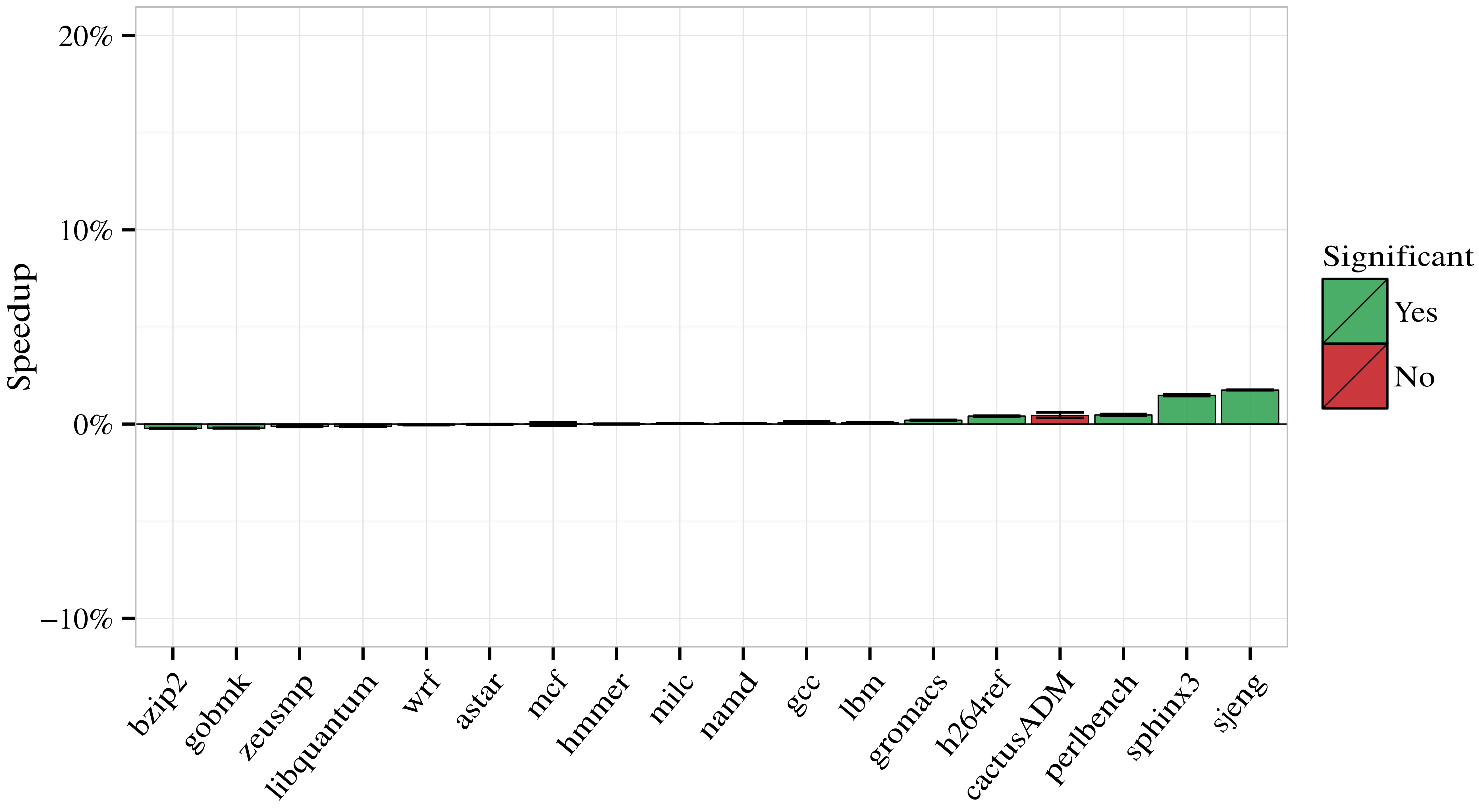
```
DataHeapType* getDataHeap() {
    static char buf[sizeof(DataHeapType)];
    static DataHeapType* _theDataHeap = new (buf) DataHeapType;
    return _theDataHeap;
}
```

```
int main(int argc, char **argv) {
    topFrame = (void**)__builtin_frame_address(0);
    setHandler(Trap::TrapSignal, onTrap);
    setHandler(SIGALRM, onTimer);
    setHandler(SIGSEGV, onFault);
    for(Function* f: functions) {
        f->setTrap();
    }
    setTimer(interval);
    int r = stabilizer_main(argc, argv);
    return r;
}
```

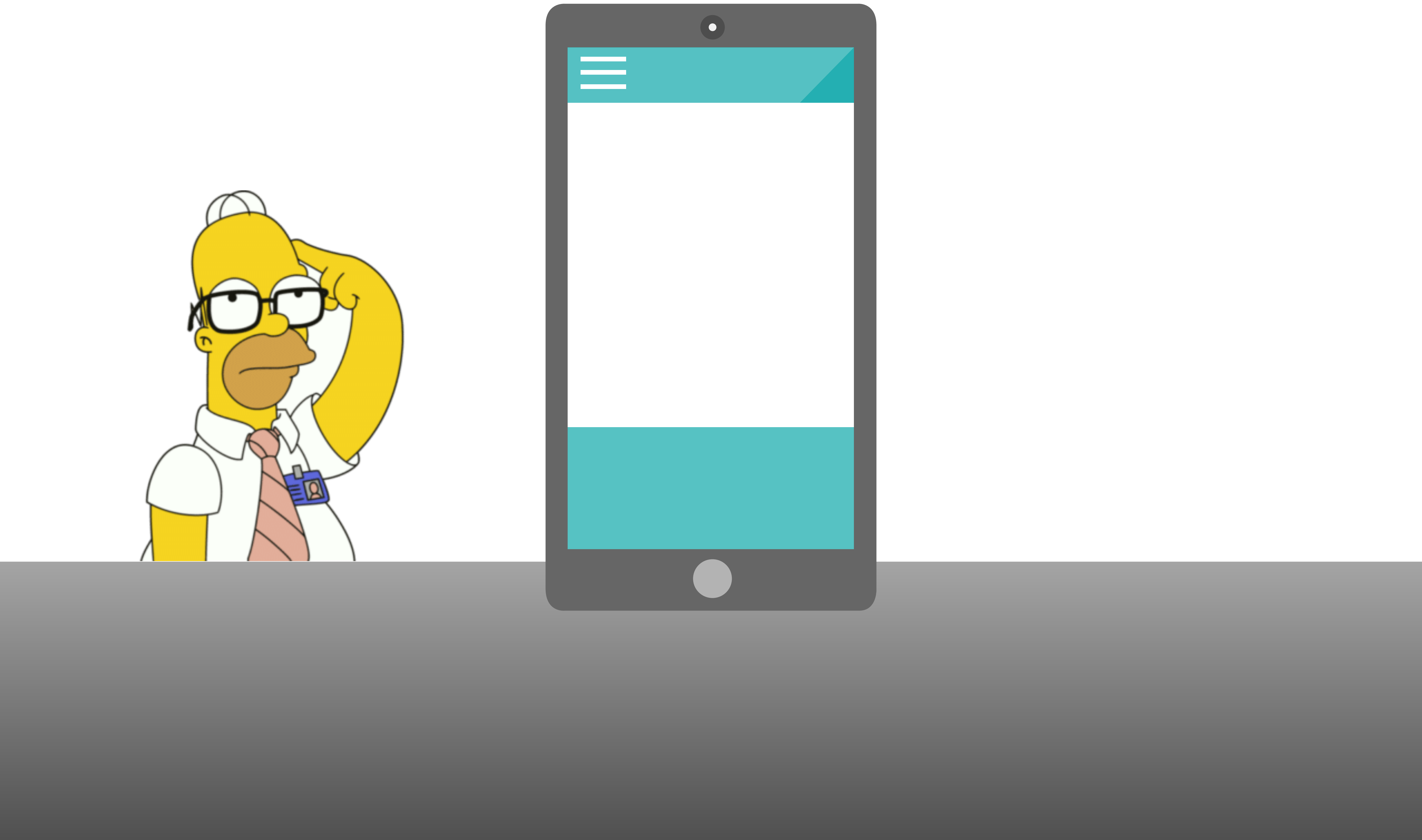
```
static void flush_icache(void* begin, size_t size) {
    uintptr_t p = (uintptr_t)begin & ~15UL;
    for (size_t i = 0; i < size; i += 16) {
        asm("icbi 0,%0" : : "r"(p));
        p += 16;
    }
}
```

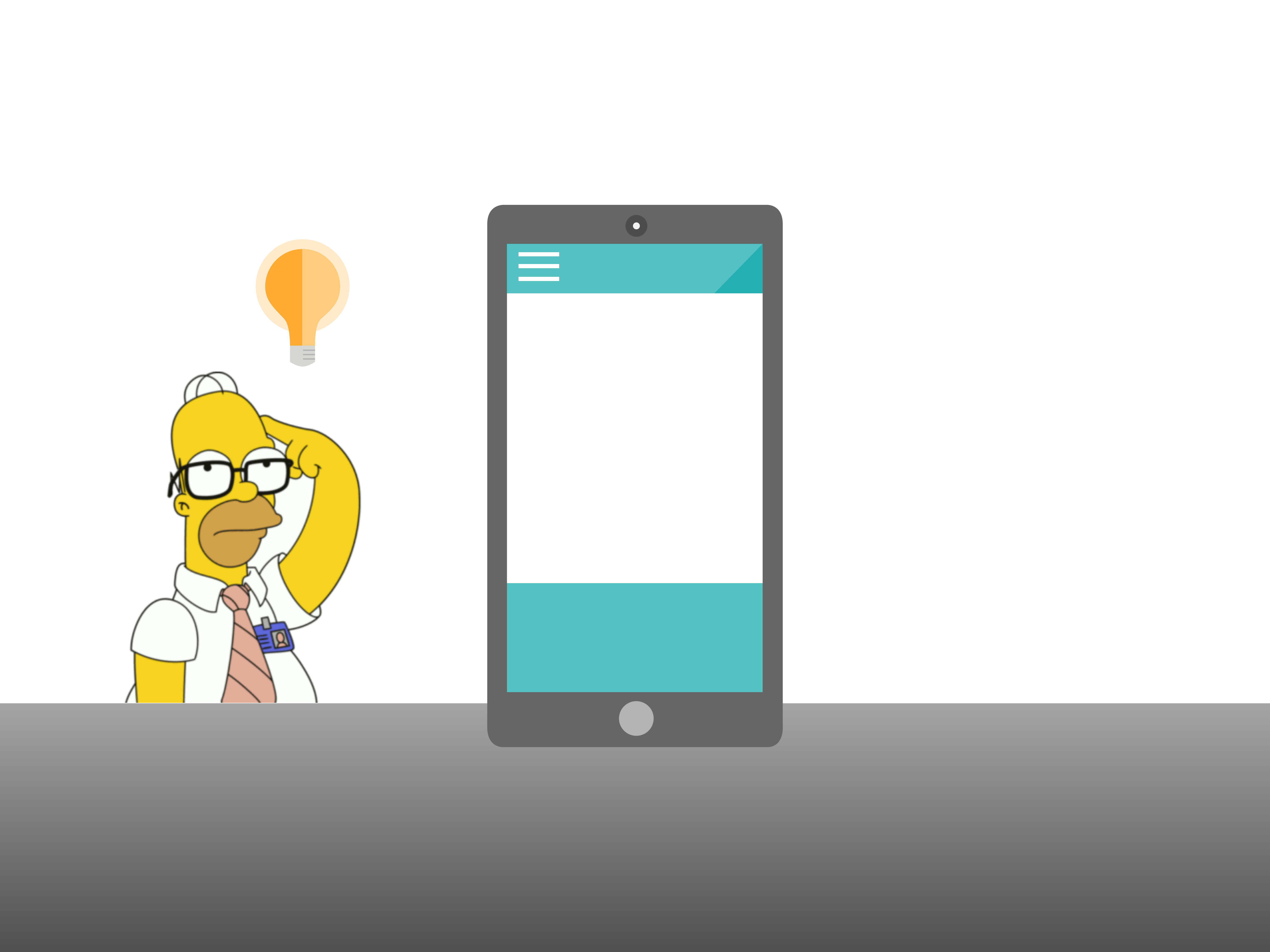
```
 void setTimer(int msec) {
     struct itimerval timer;

     timer.it_value.tv_sec = (msec - msec % 1000) / 1000;
     timer.it_value.tv_usec = 1000 * (msec % 1000);
     timer.it_interval.tv_sec = 0;
     timer.it_interval.tv_usec = 0;

     setitimer(ITIMER_REAL, &timer, 0);
 }
```

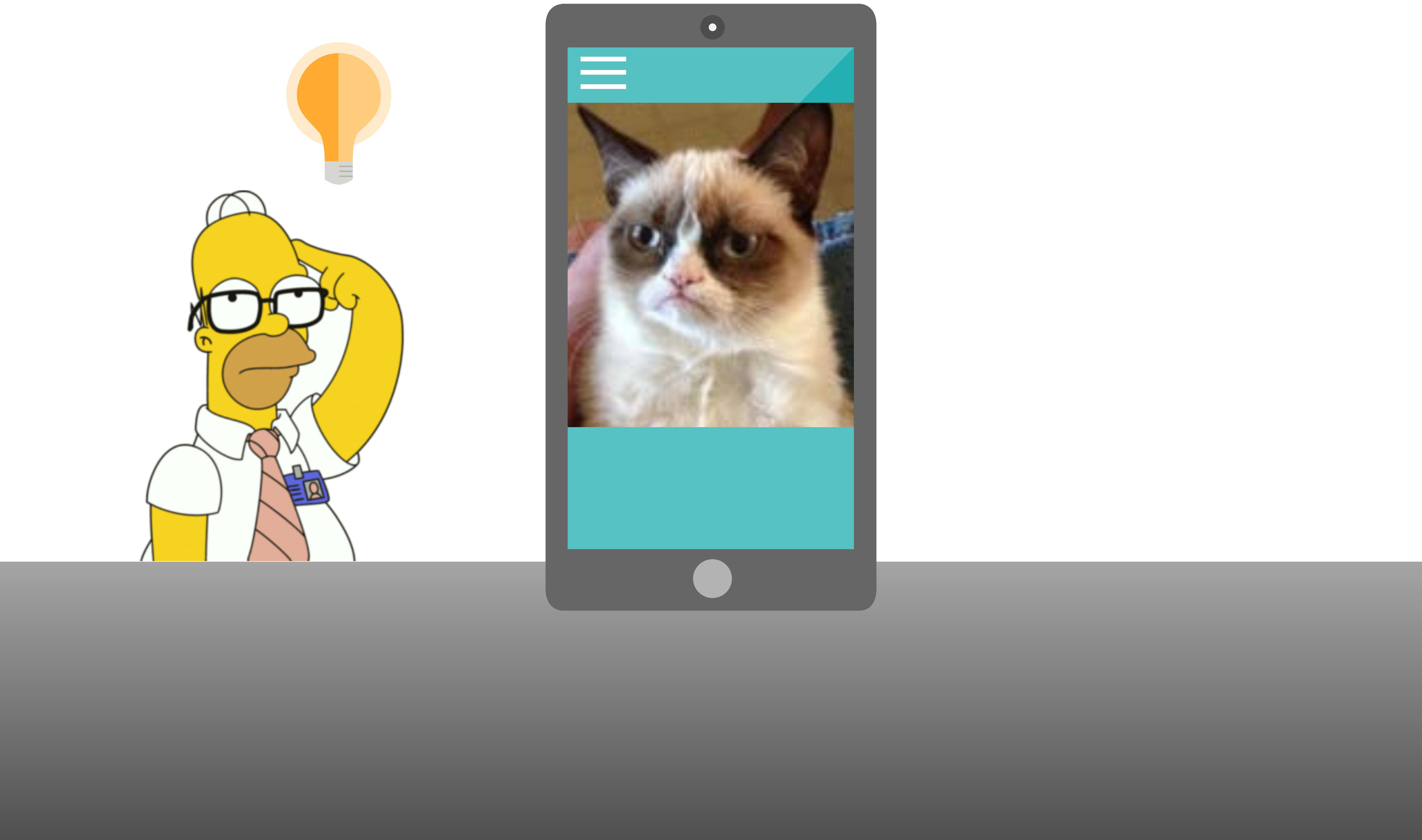
Speedup of −O3 over −O2

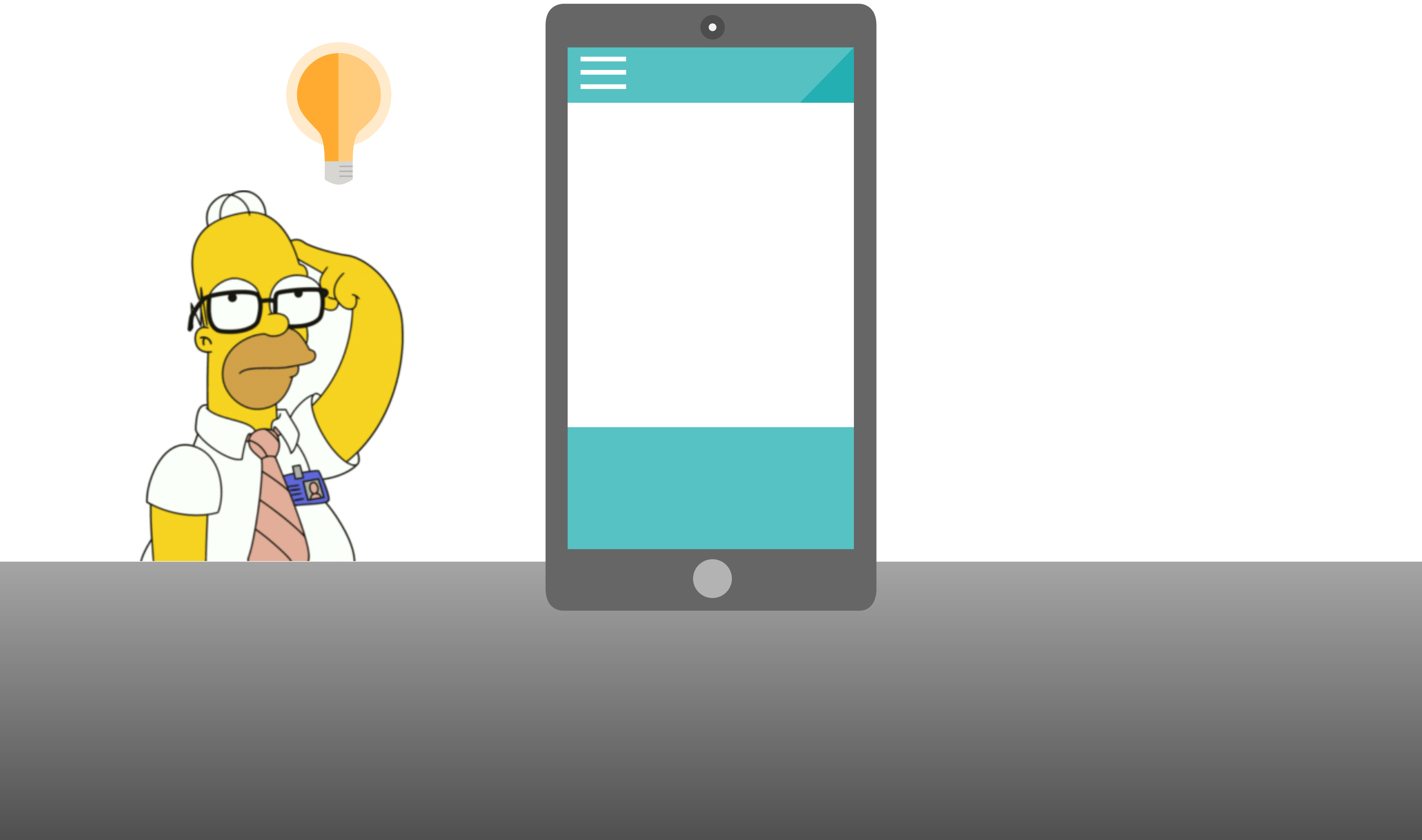


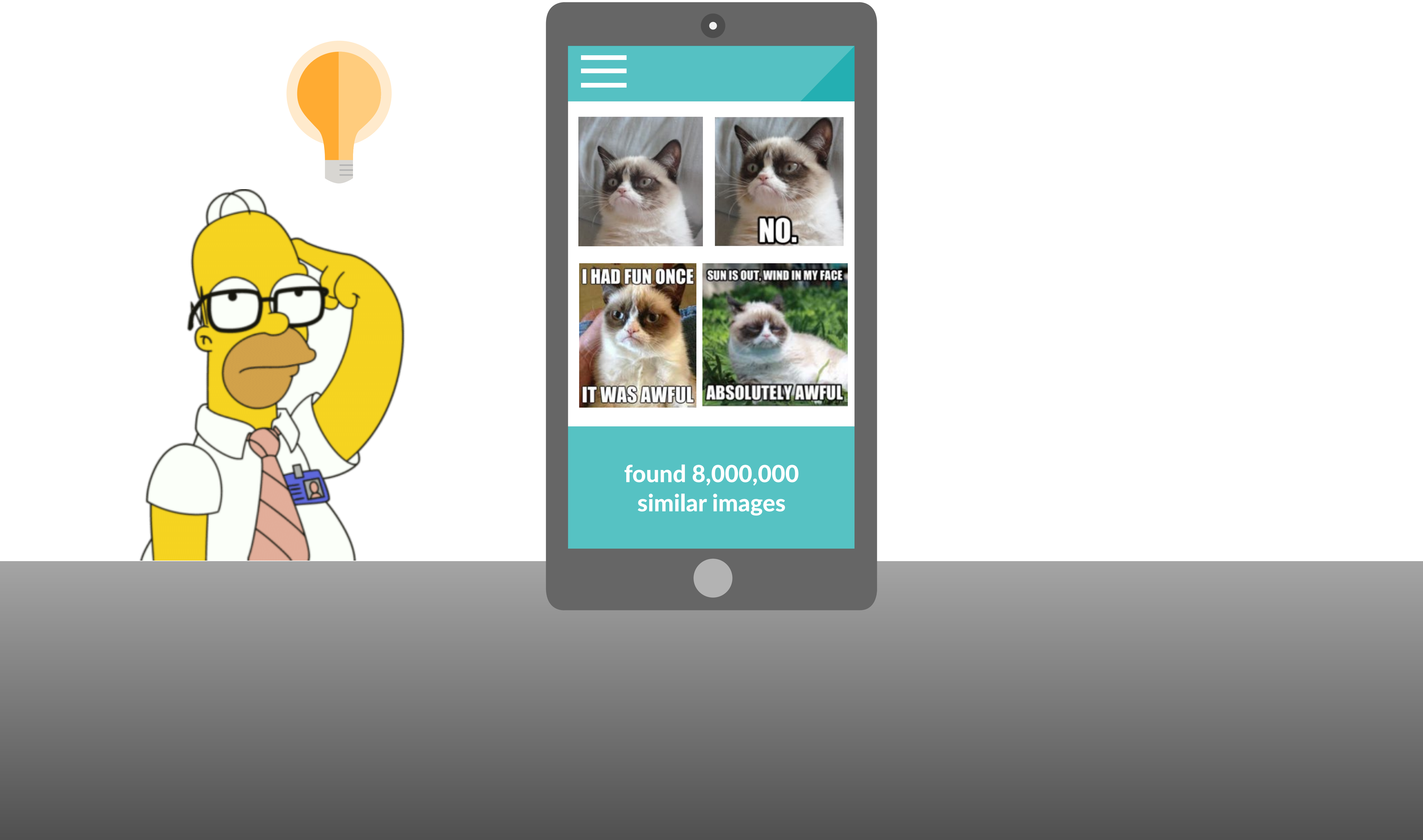

help?







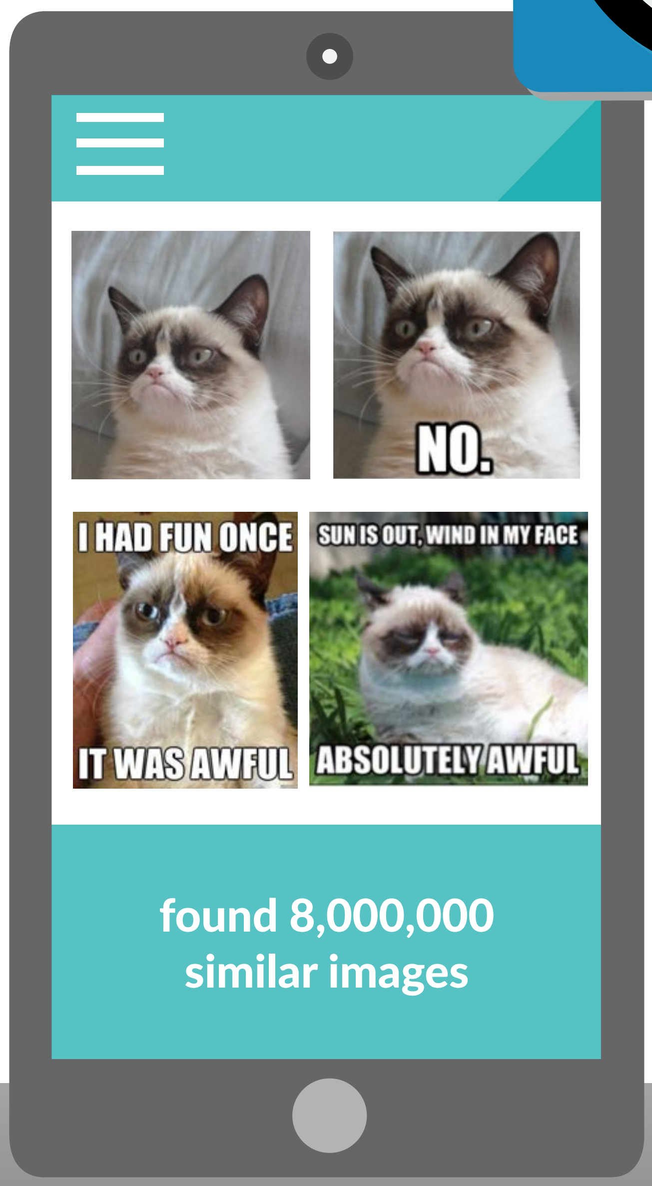



NO.
I HAD FUN ONCE
IT WAS AWFUL
SUN IS OUT, WIND IN MY FACE
ABSOLUTELY AWFUL
found 8,000,000
similar images

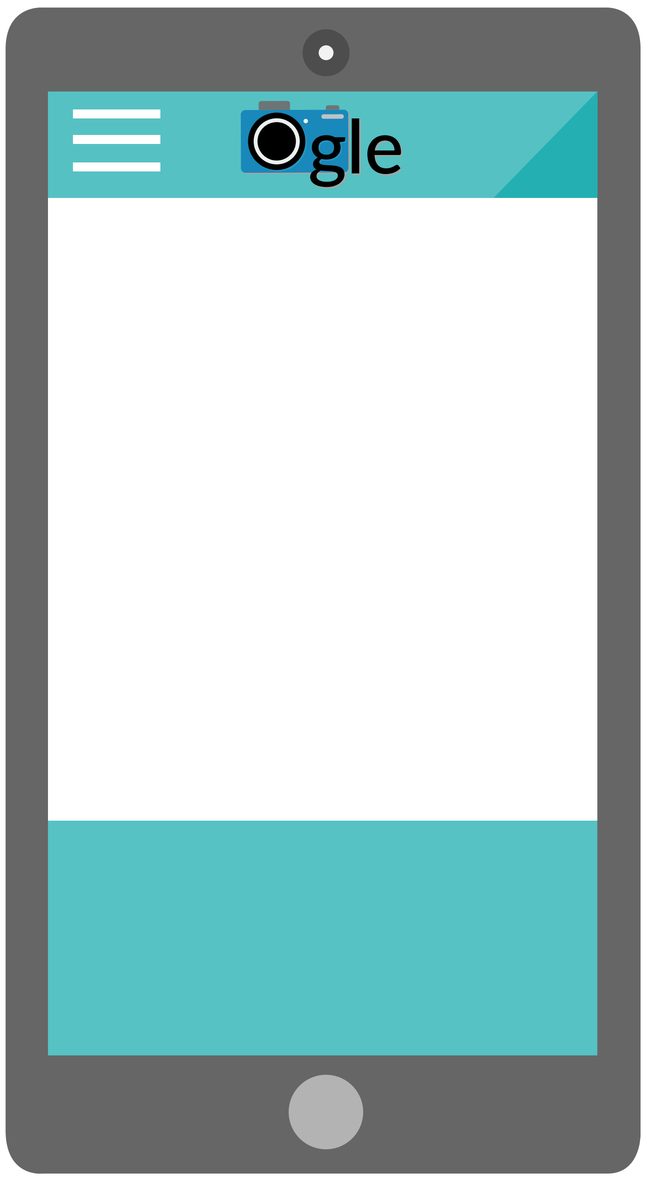

Ogle
NO.
I HAD FUN ONCE
IT WAS AWFUL
SUN IS OUT, WIND IN MY FACE
ABSOLUTELY AWFUL
found 8,000,000
similar images

Ogle

It's going to *totally* disrupt image search.

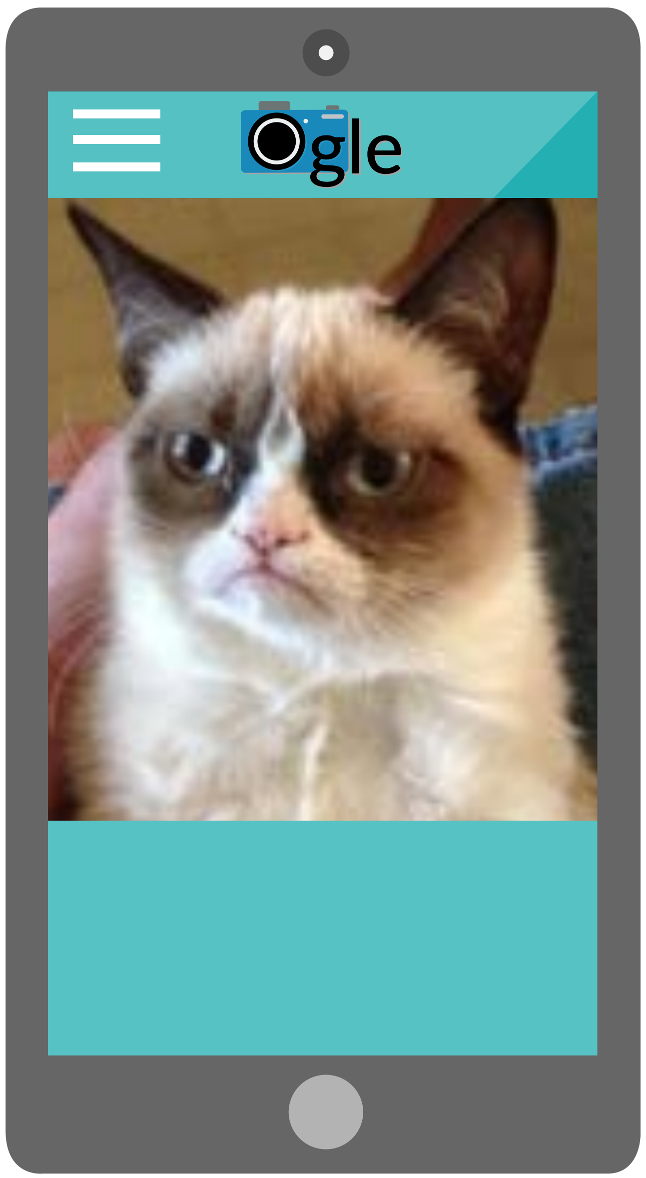


The Prototype Ogle



The Prototype Ogle

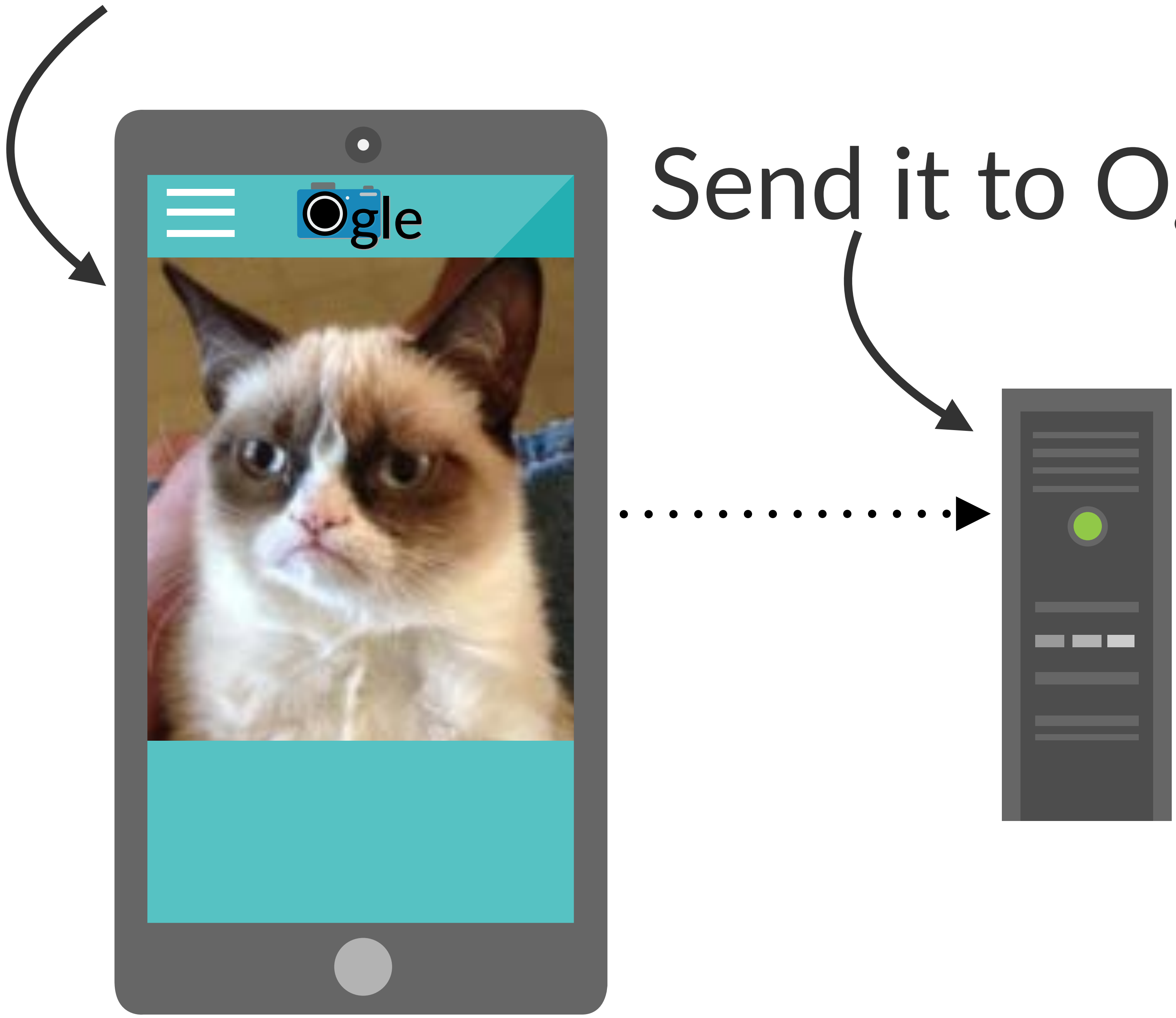
Take a picture



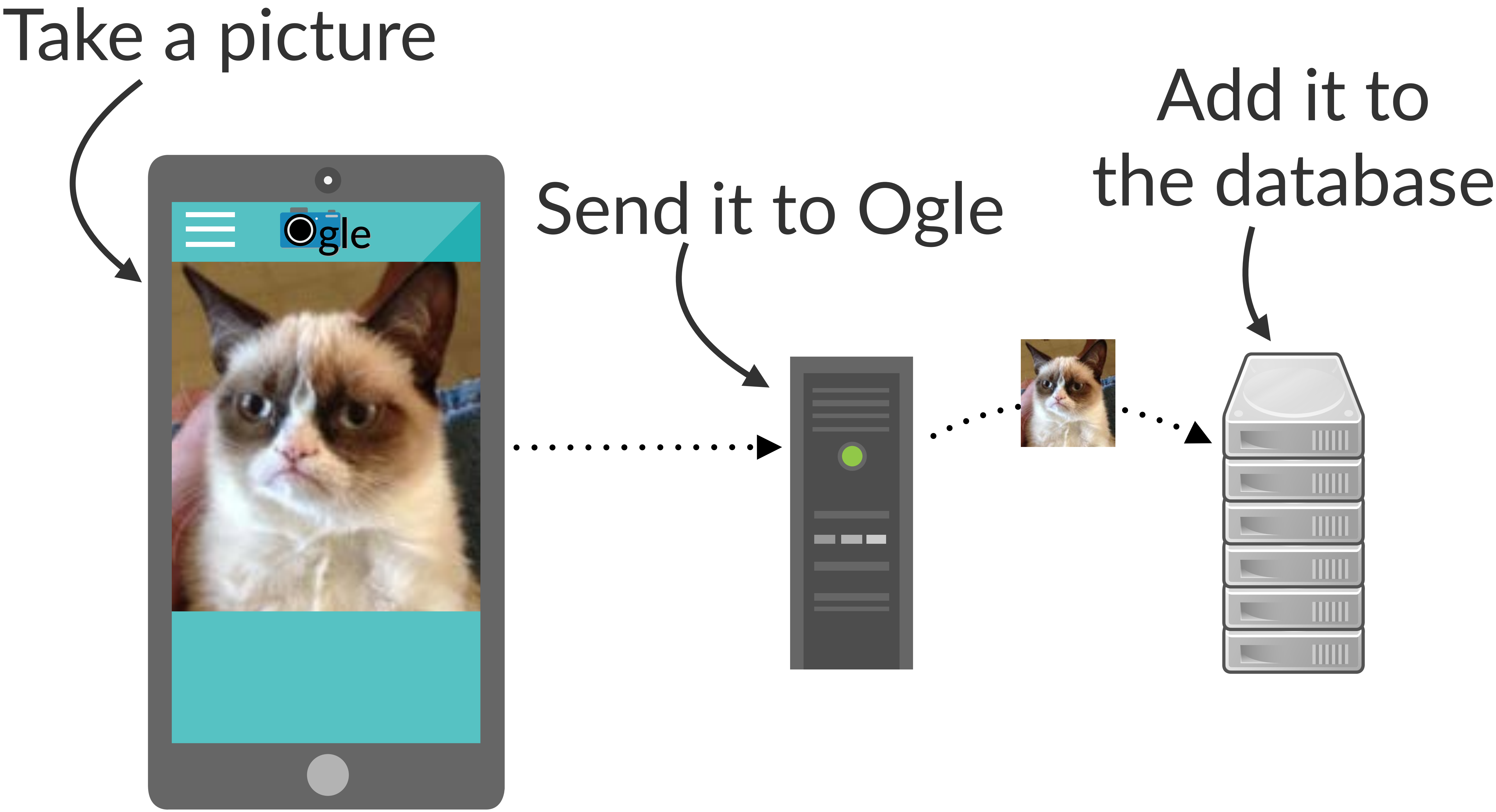
The Prototype Ogle

Take a picture

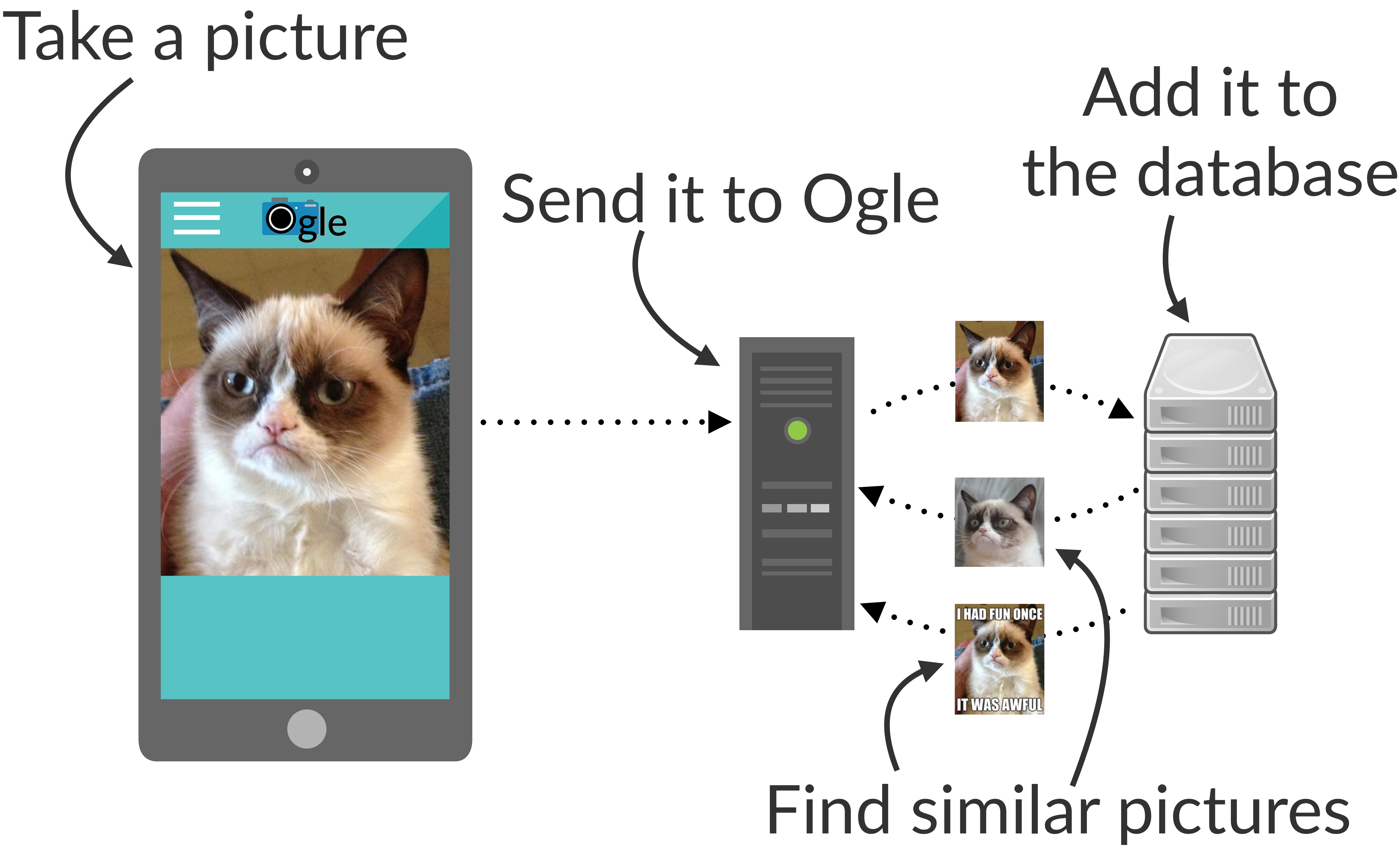
Send it to Ogle



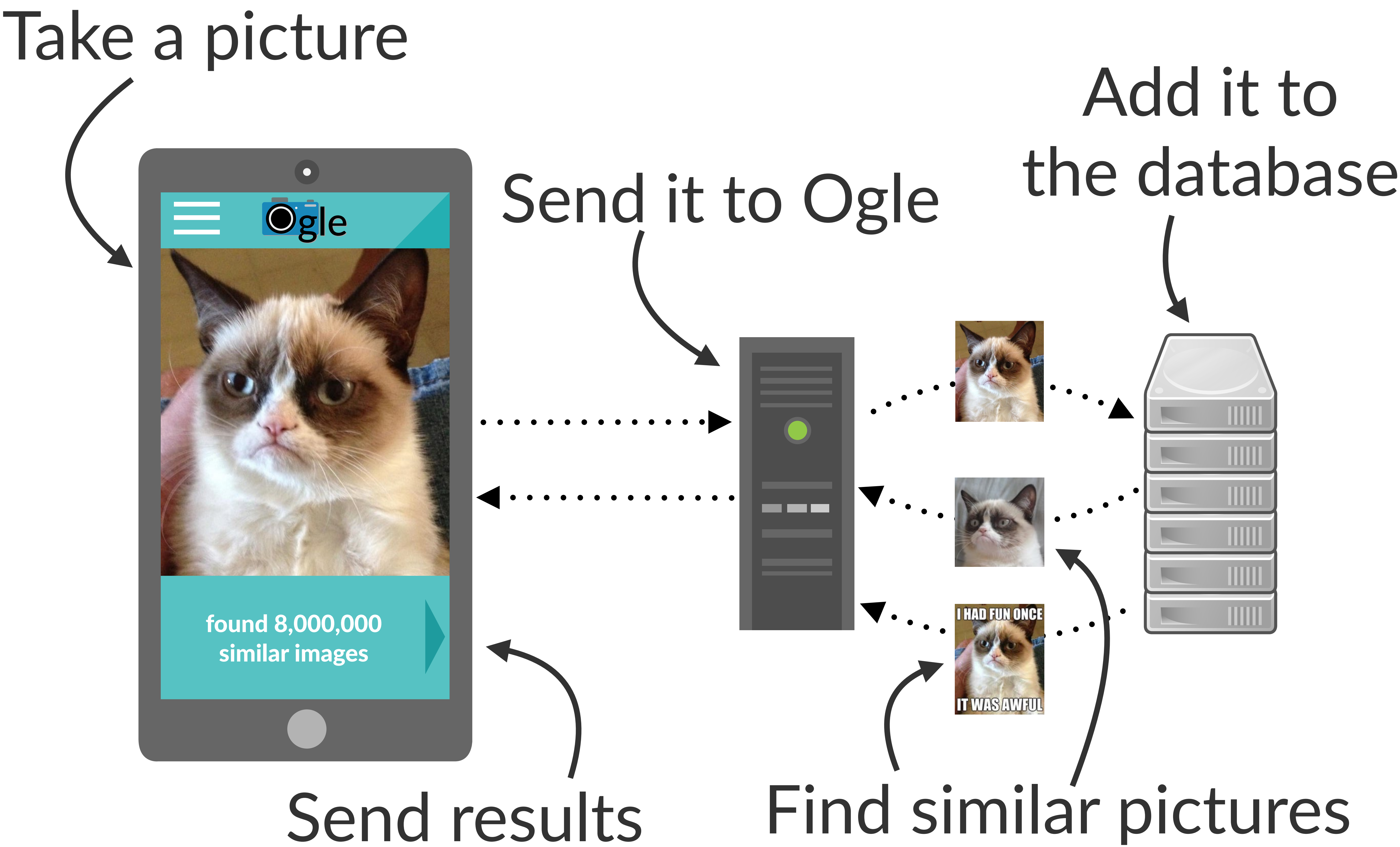
The Prototype Ogle

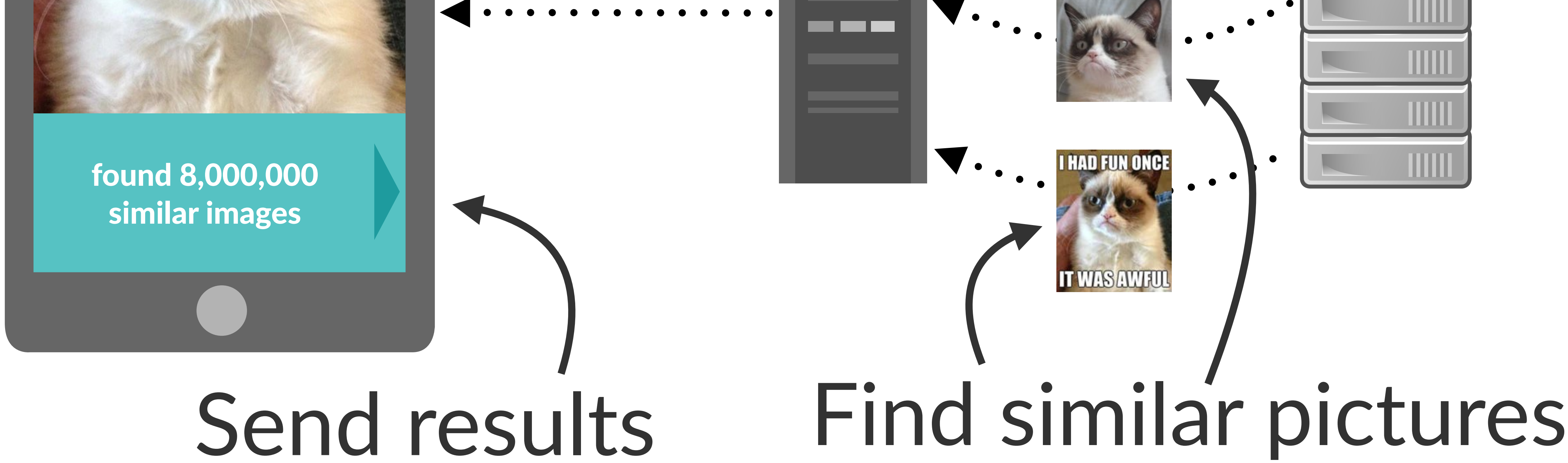


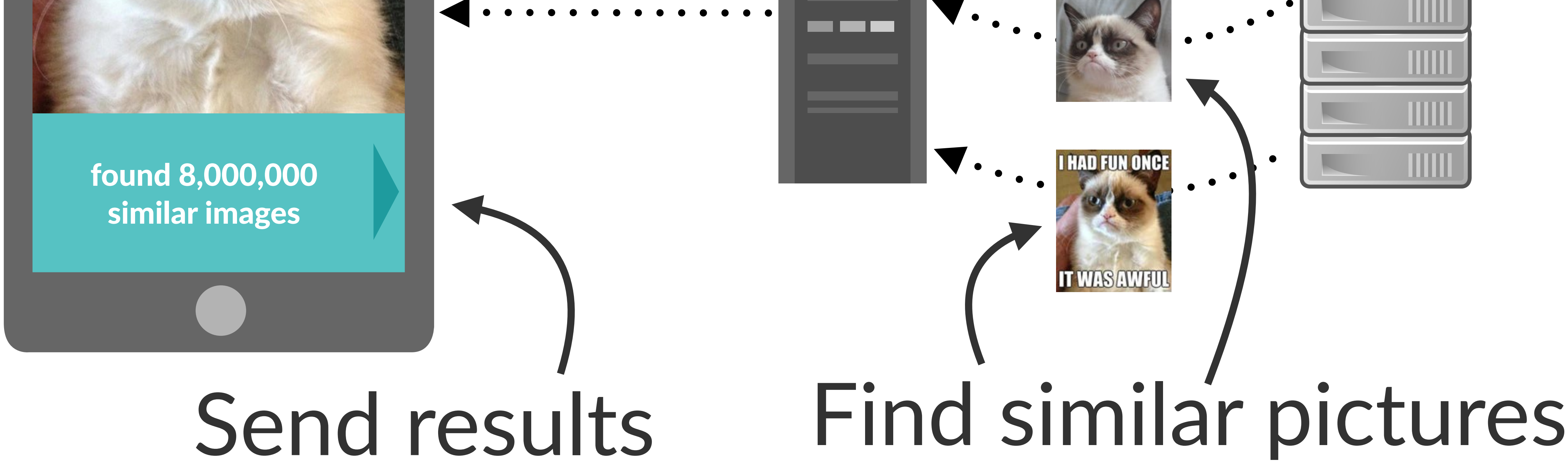
The Prototype Ogle

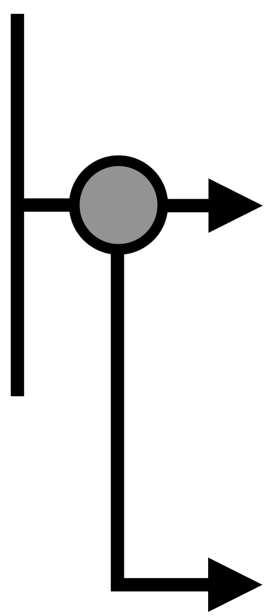


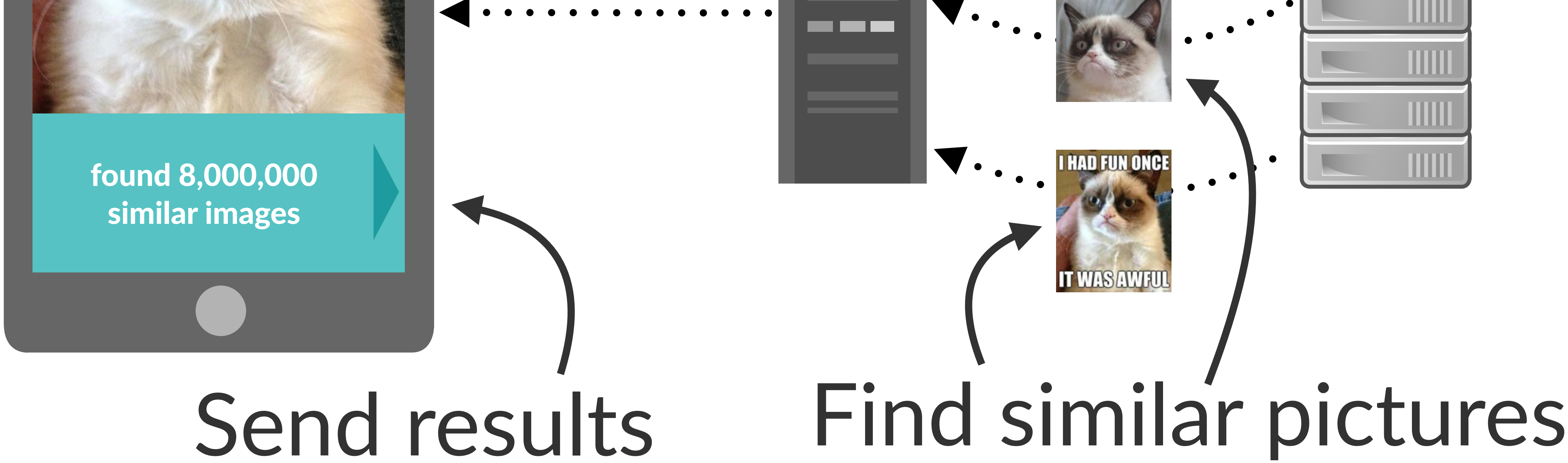
The Prototype Ogle

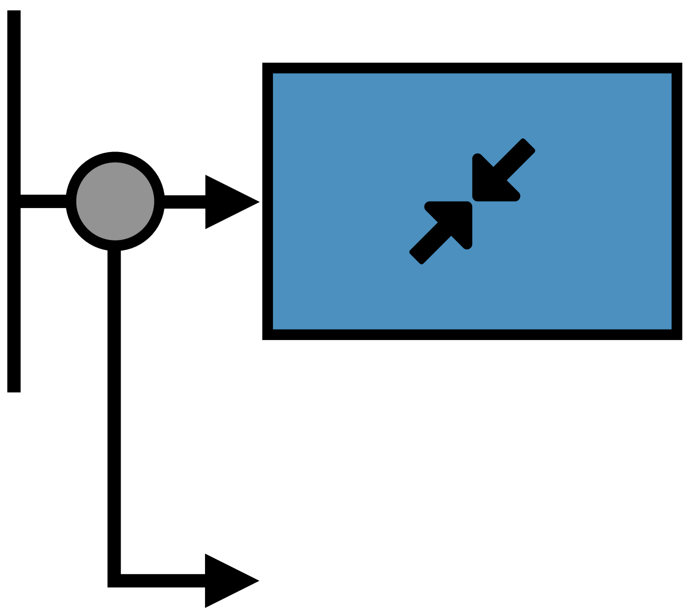


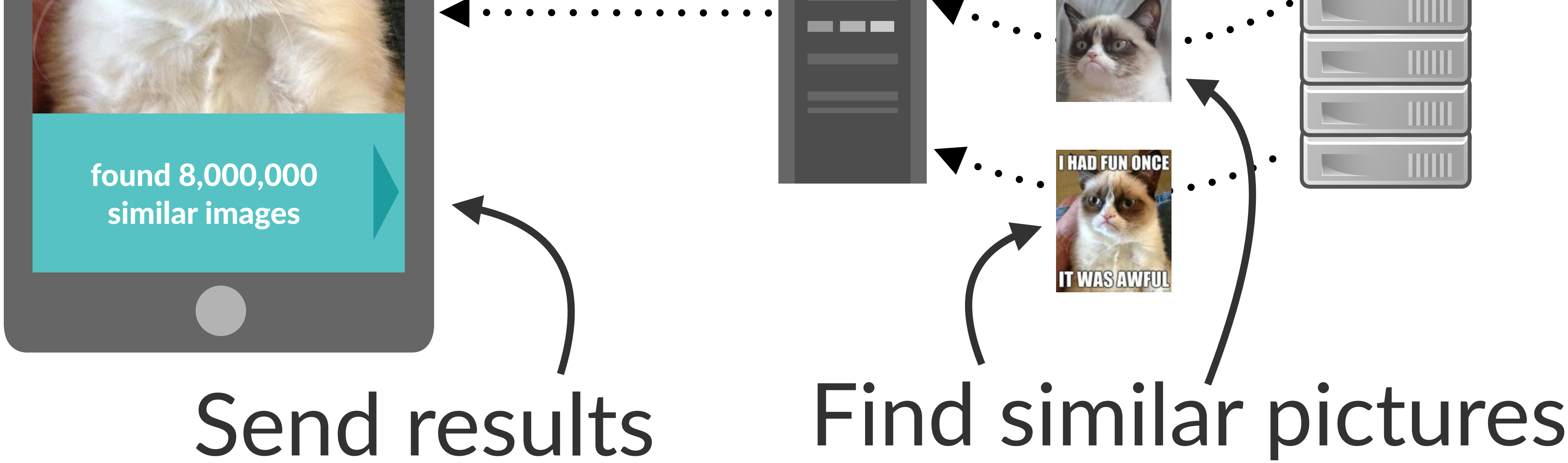

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

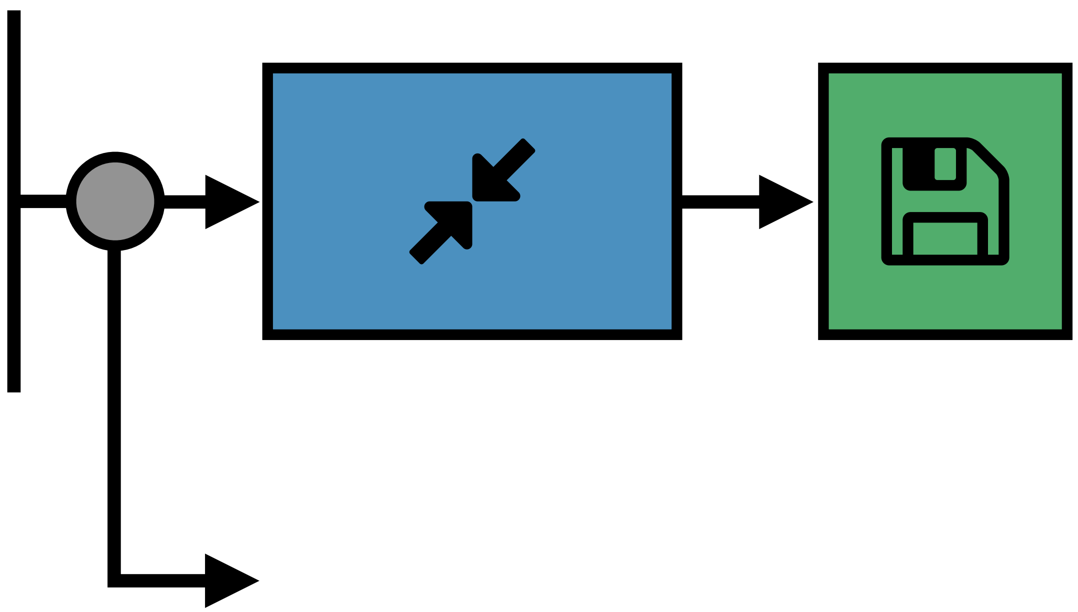

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

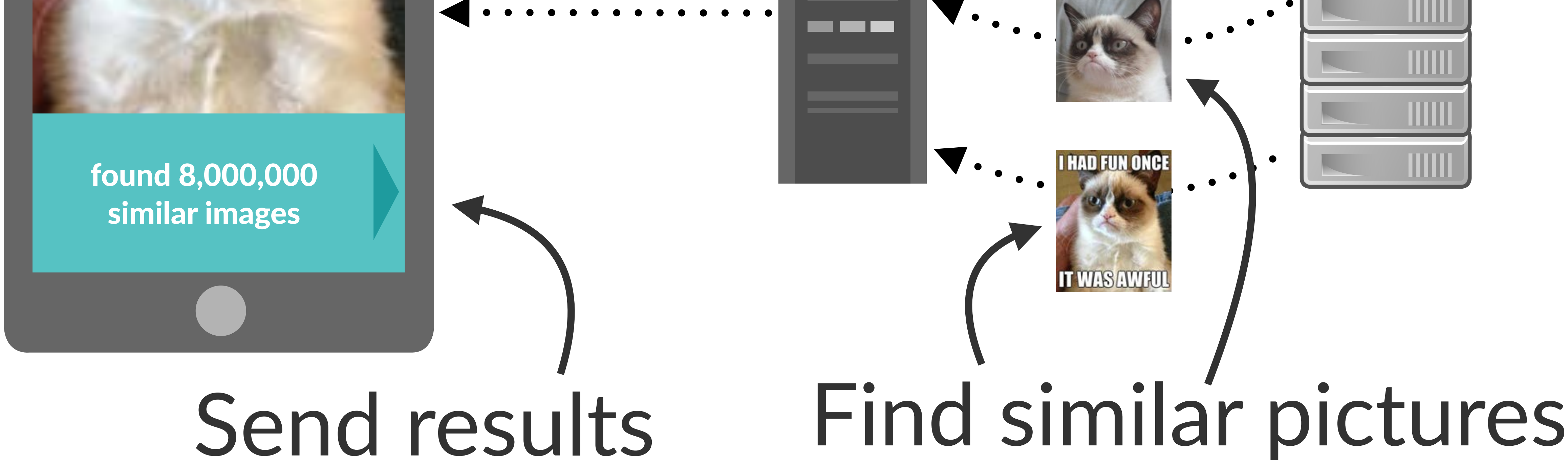


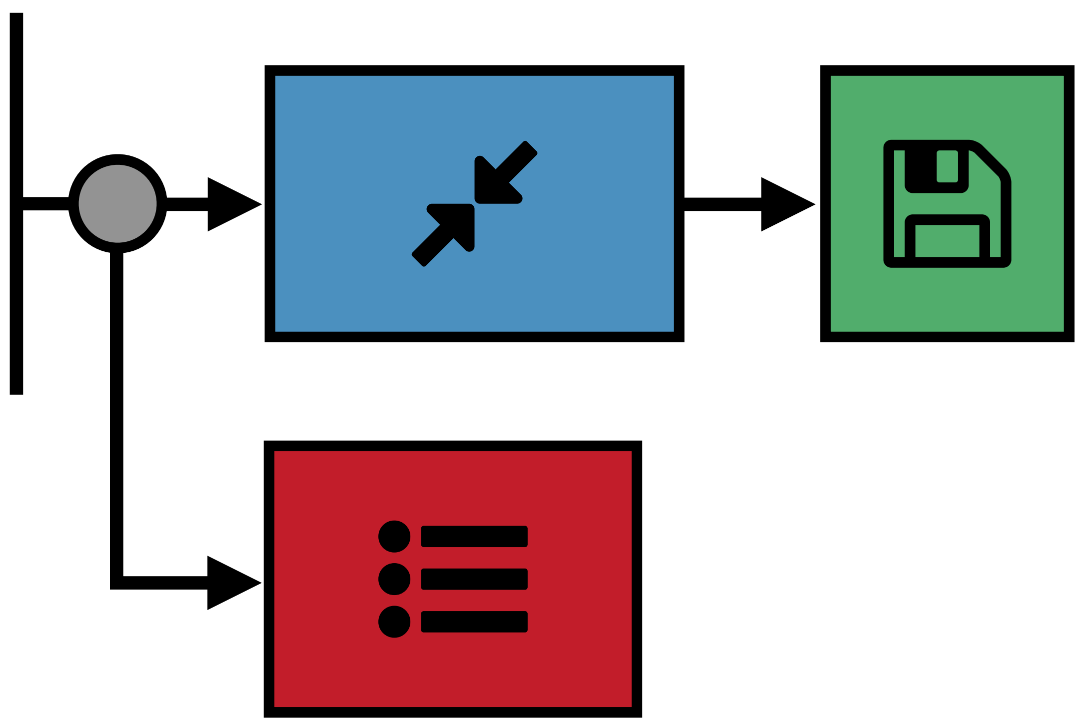

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

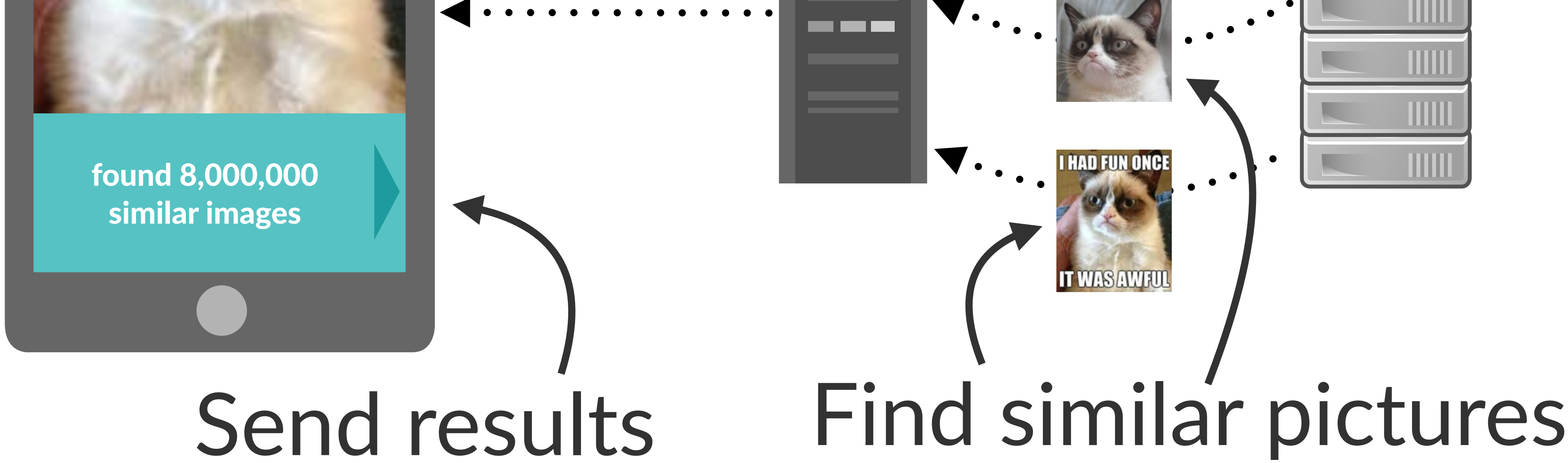


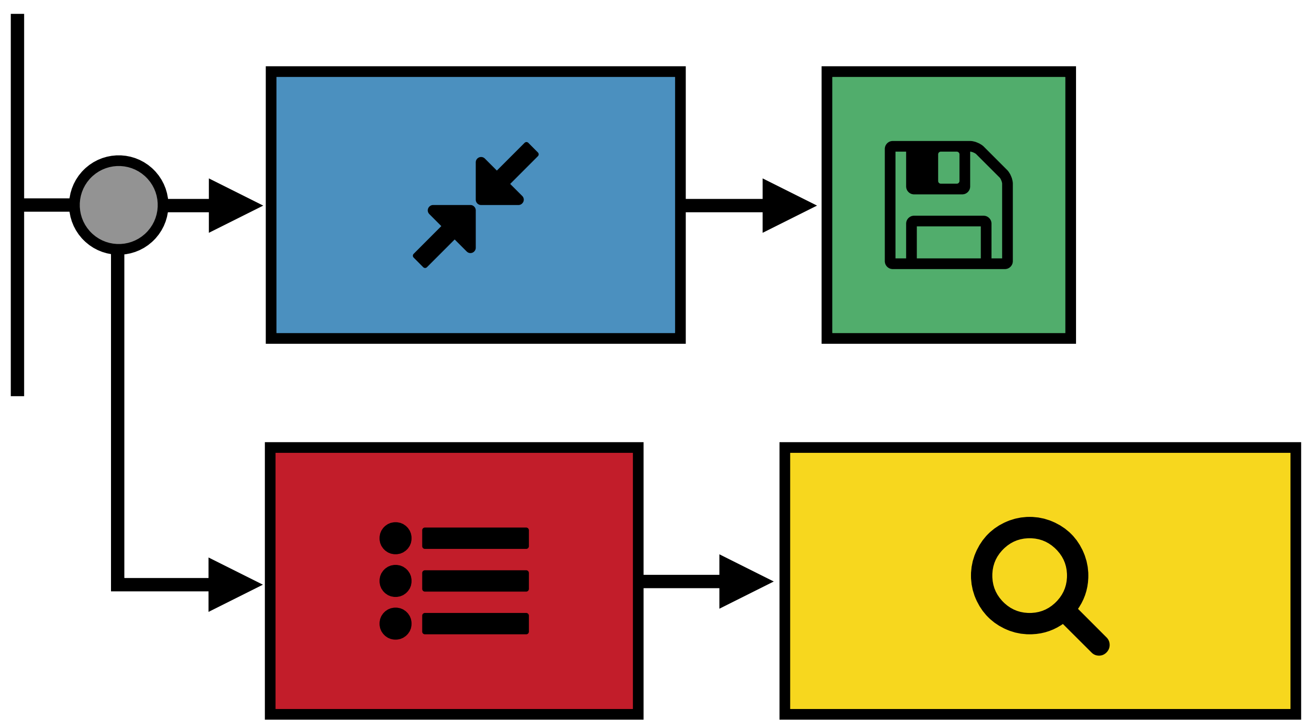

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

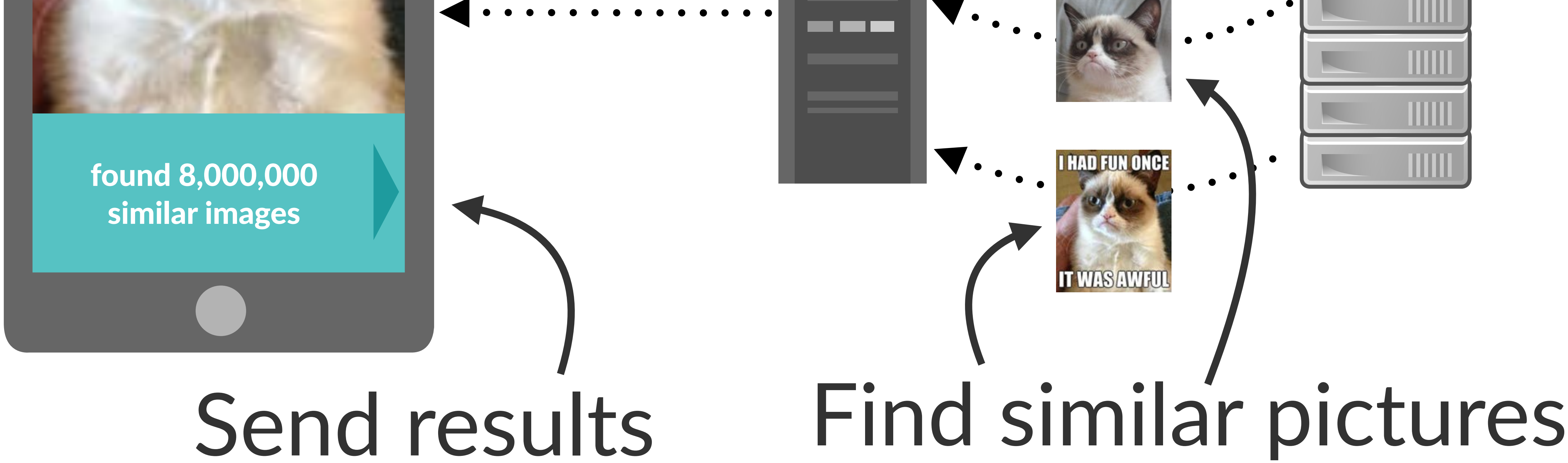


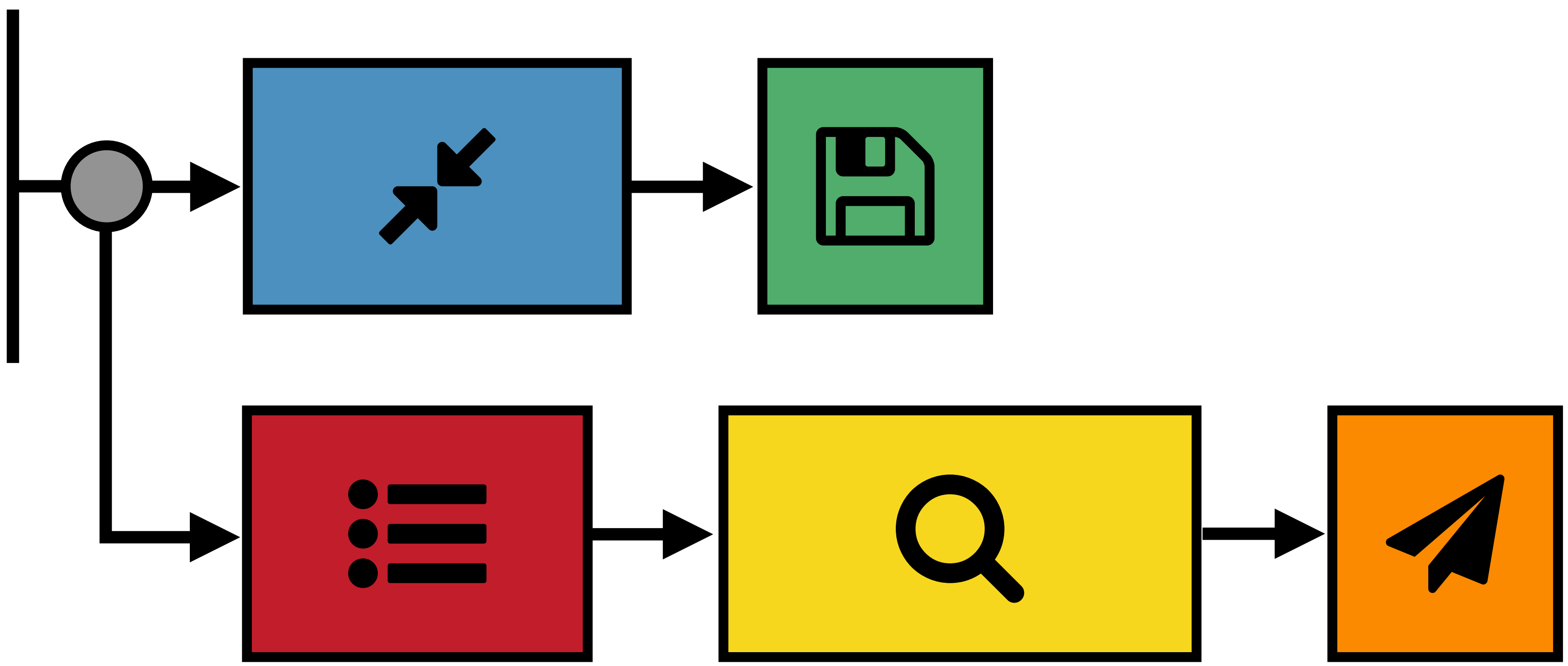

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

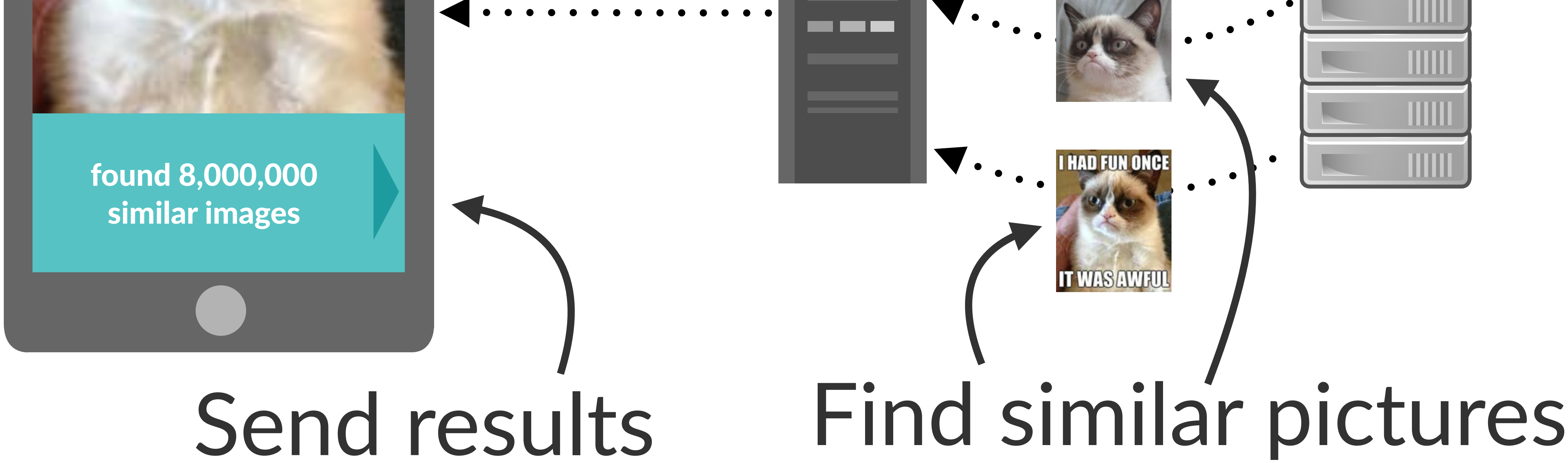


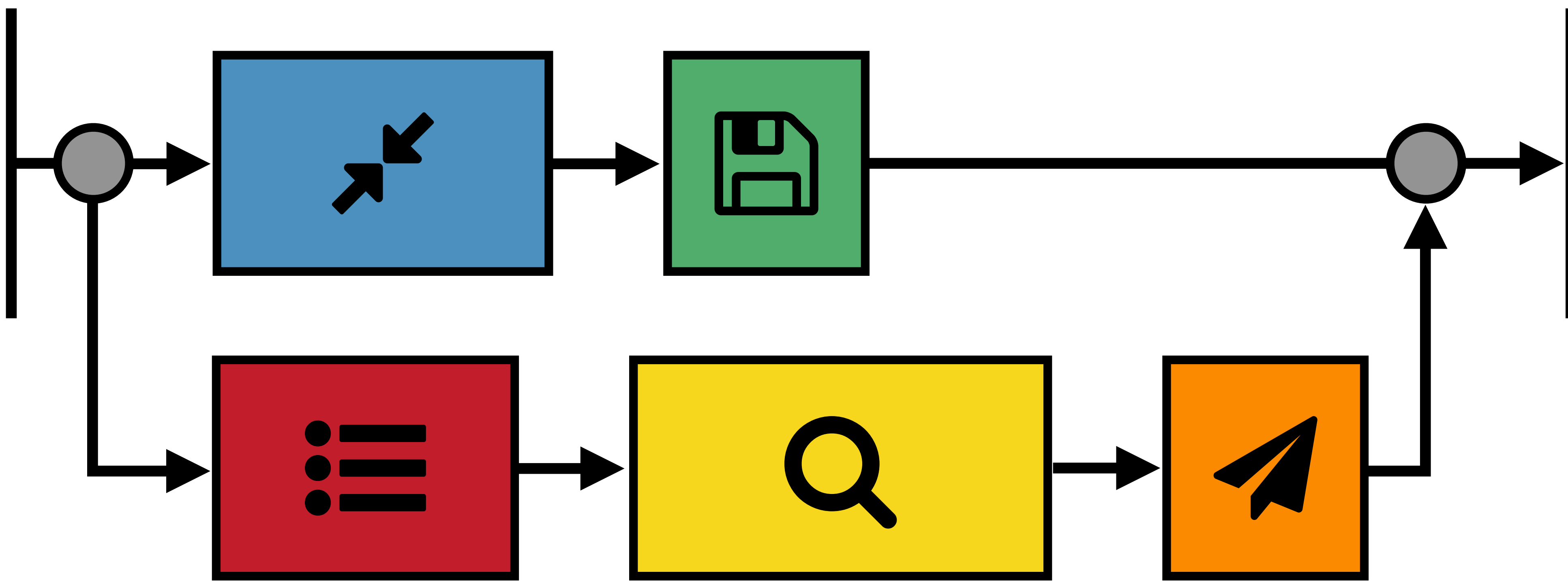

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

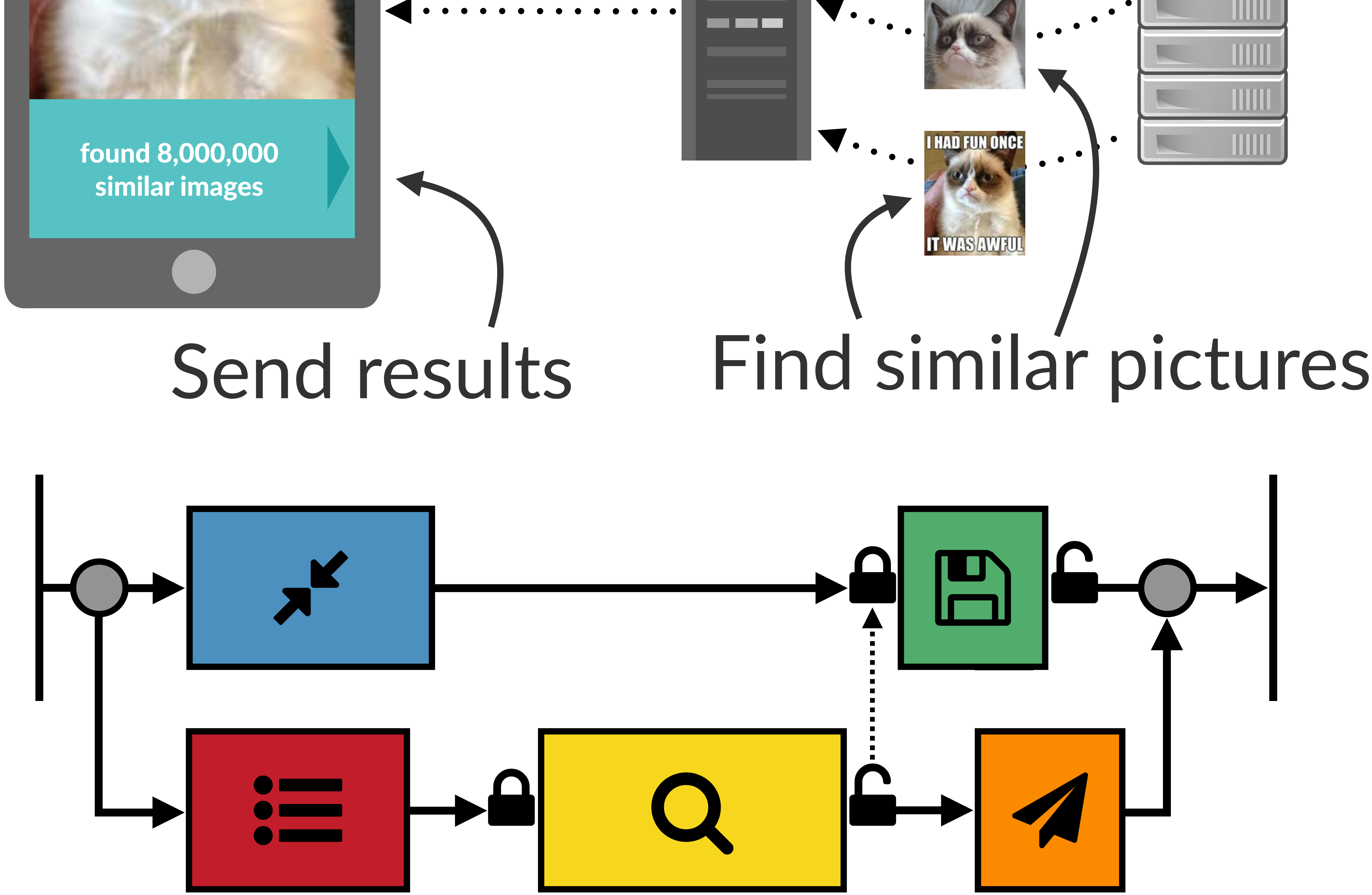


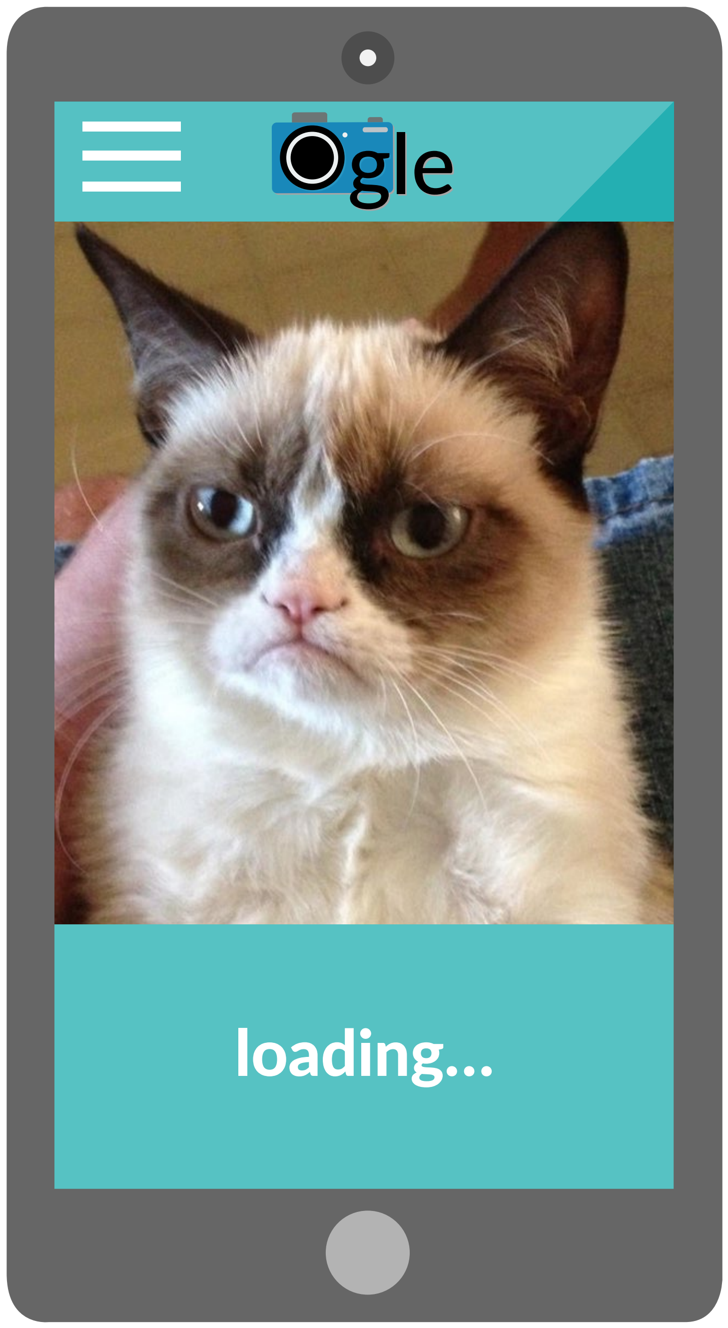

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

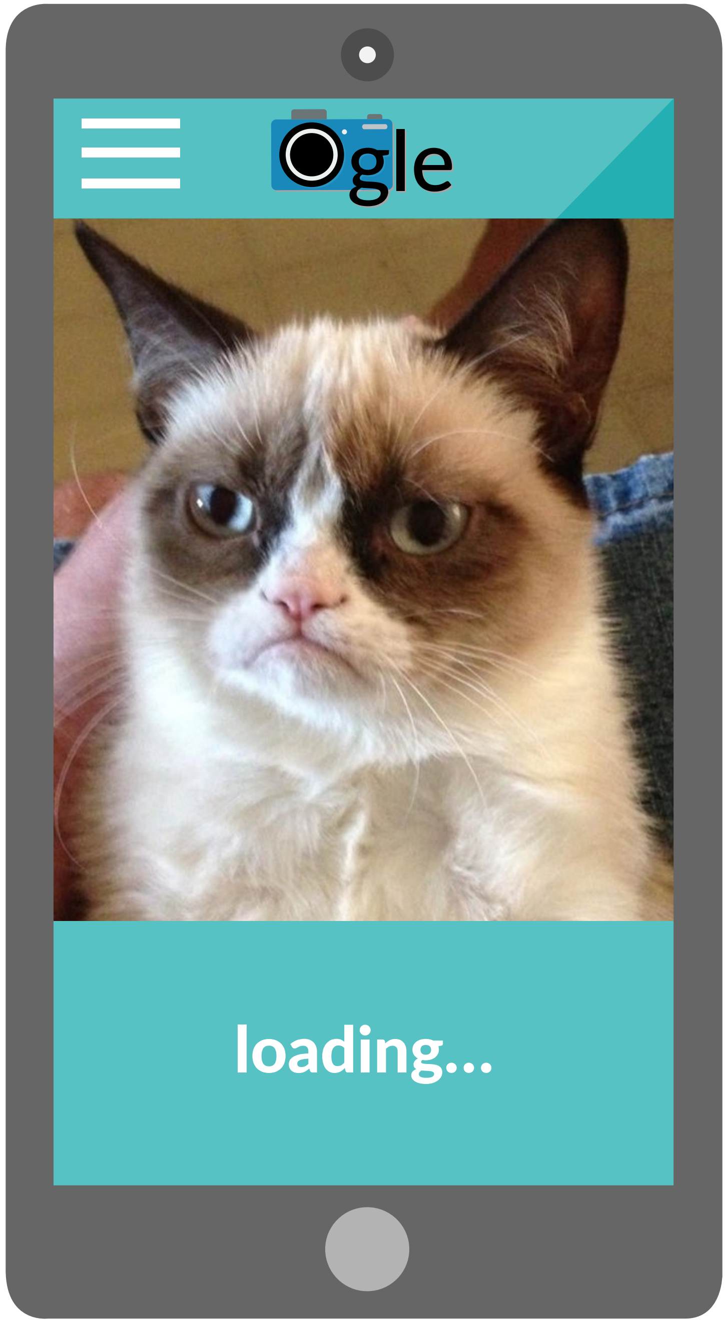


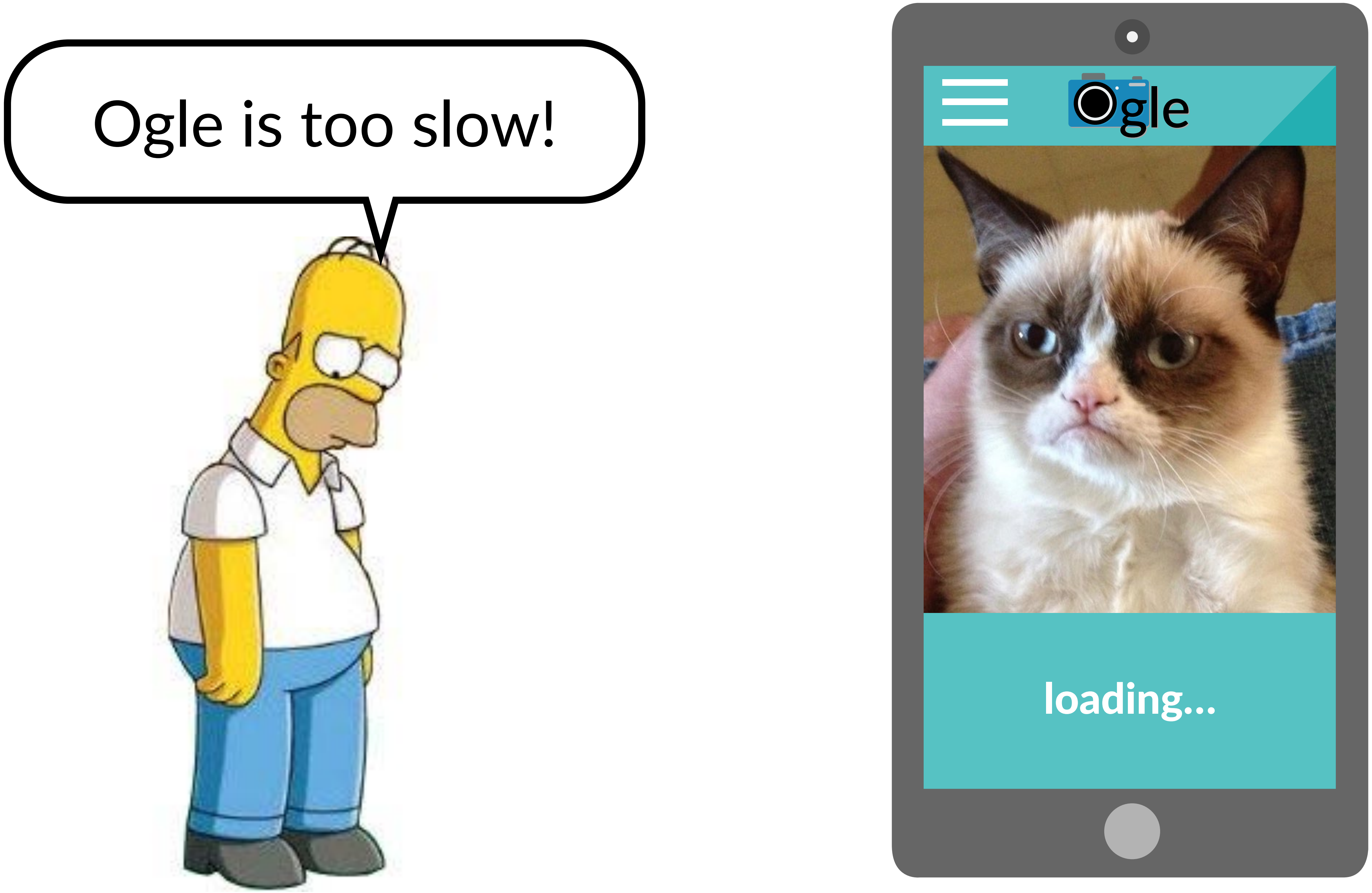

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures



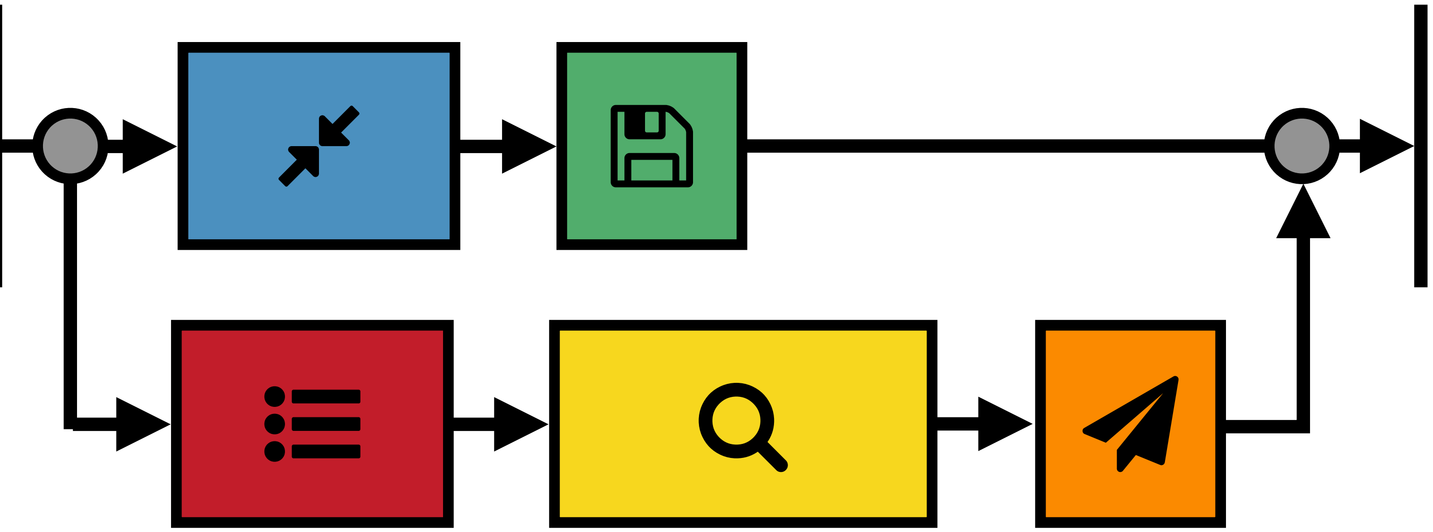

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures


loading...

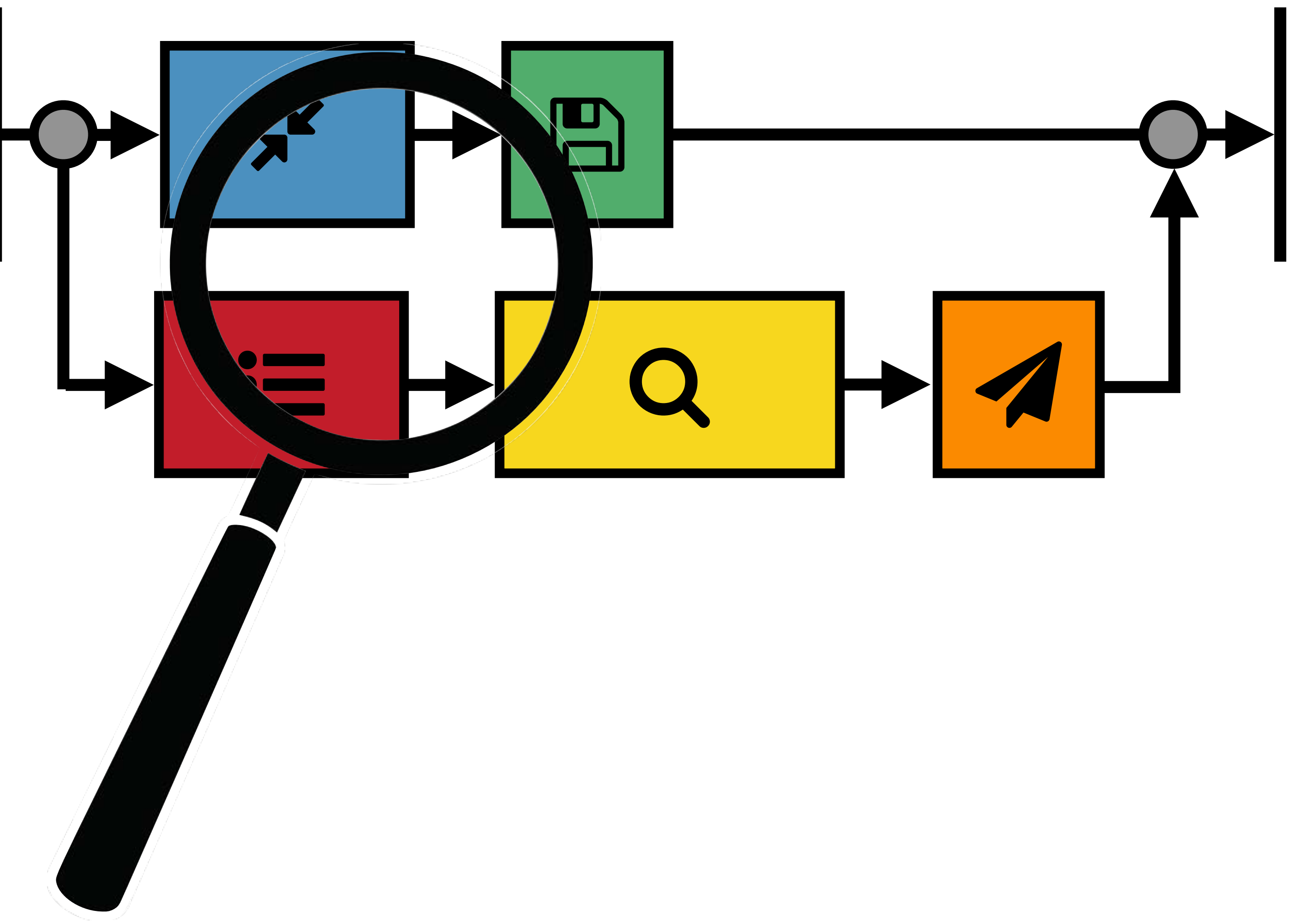

loading...


Ogle is too slow!
Ogle
loading...

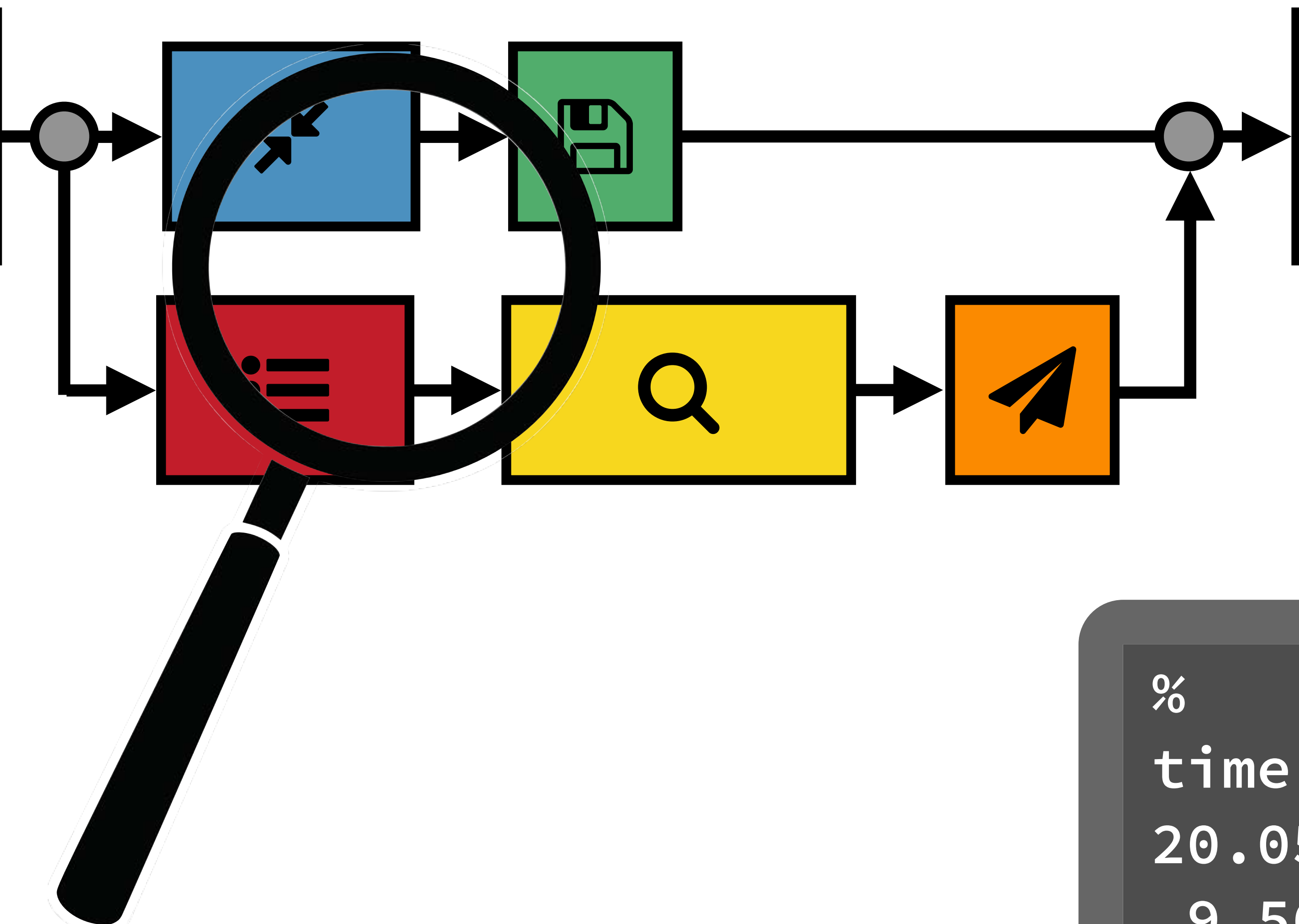
Software Profilers

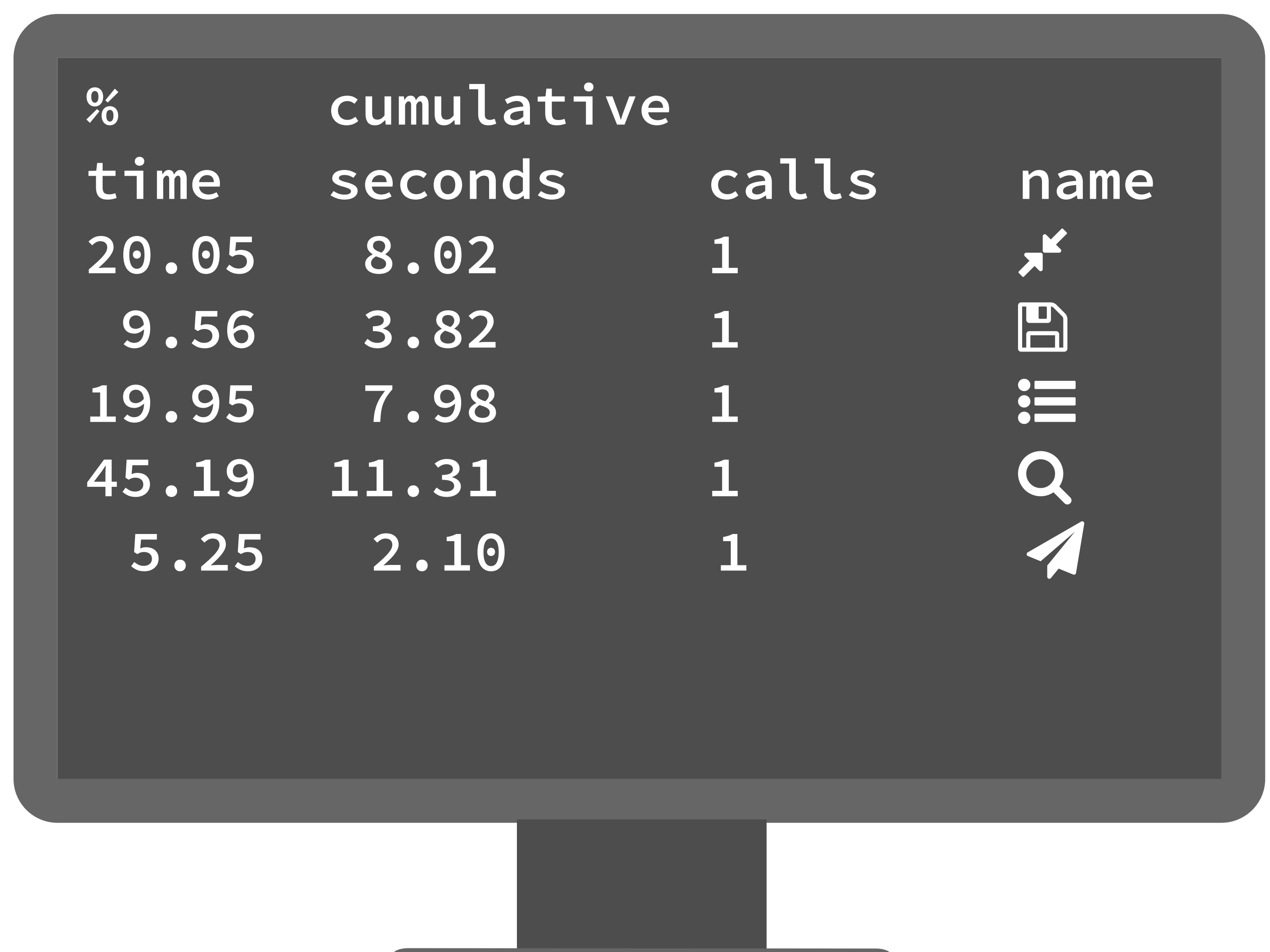


Software Profilers



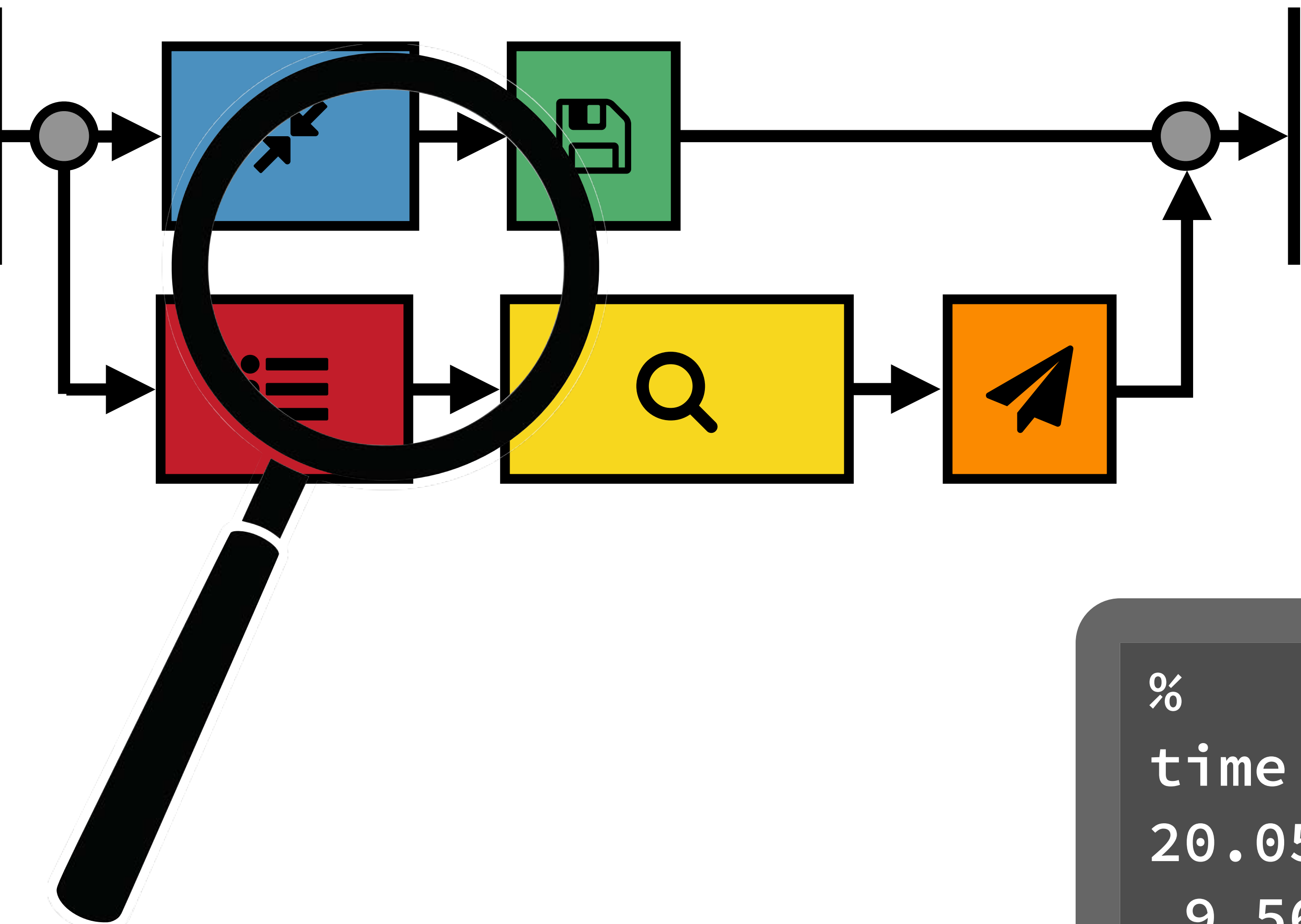
Software Profilers



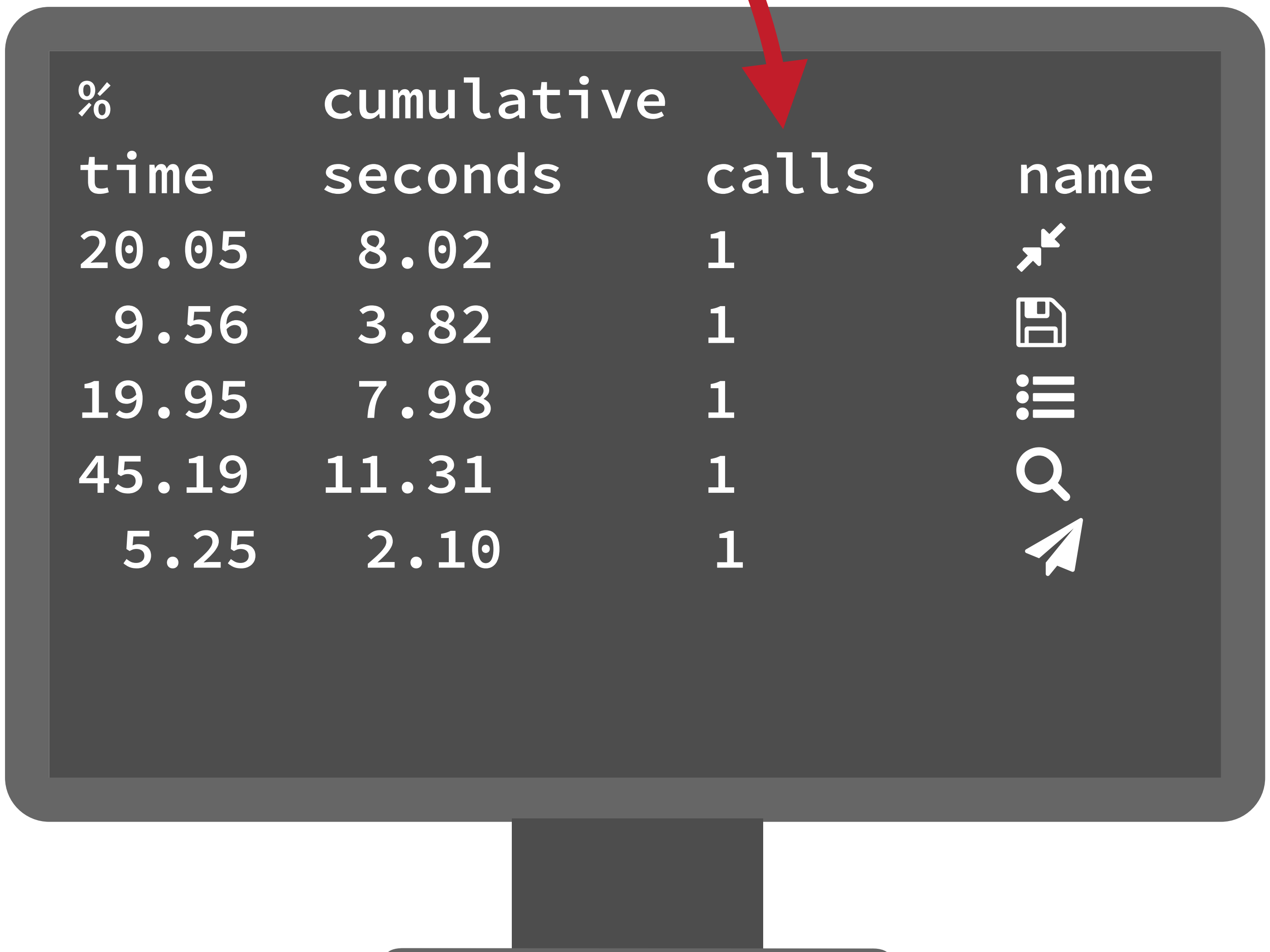


% time	cumulative seconds	calls	name
20.05	8.02	1	
9.56	3.82	1	
19.95	7.98	1	
45.19	11.31	1	
5.25	2.10	1	

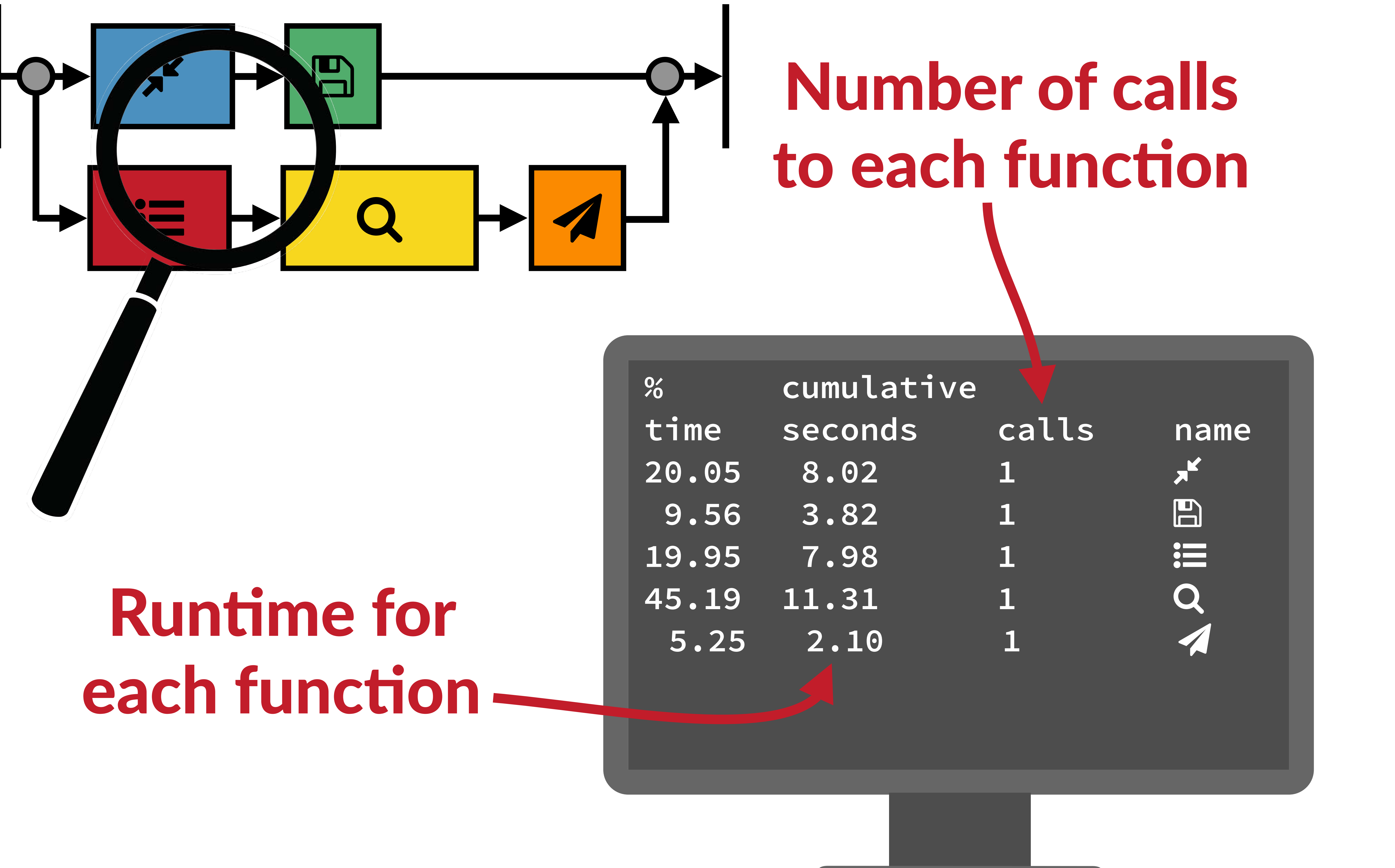
Software Profilers



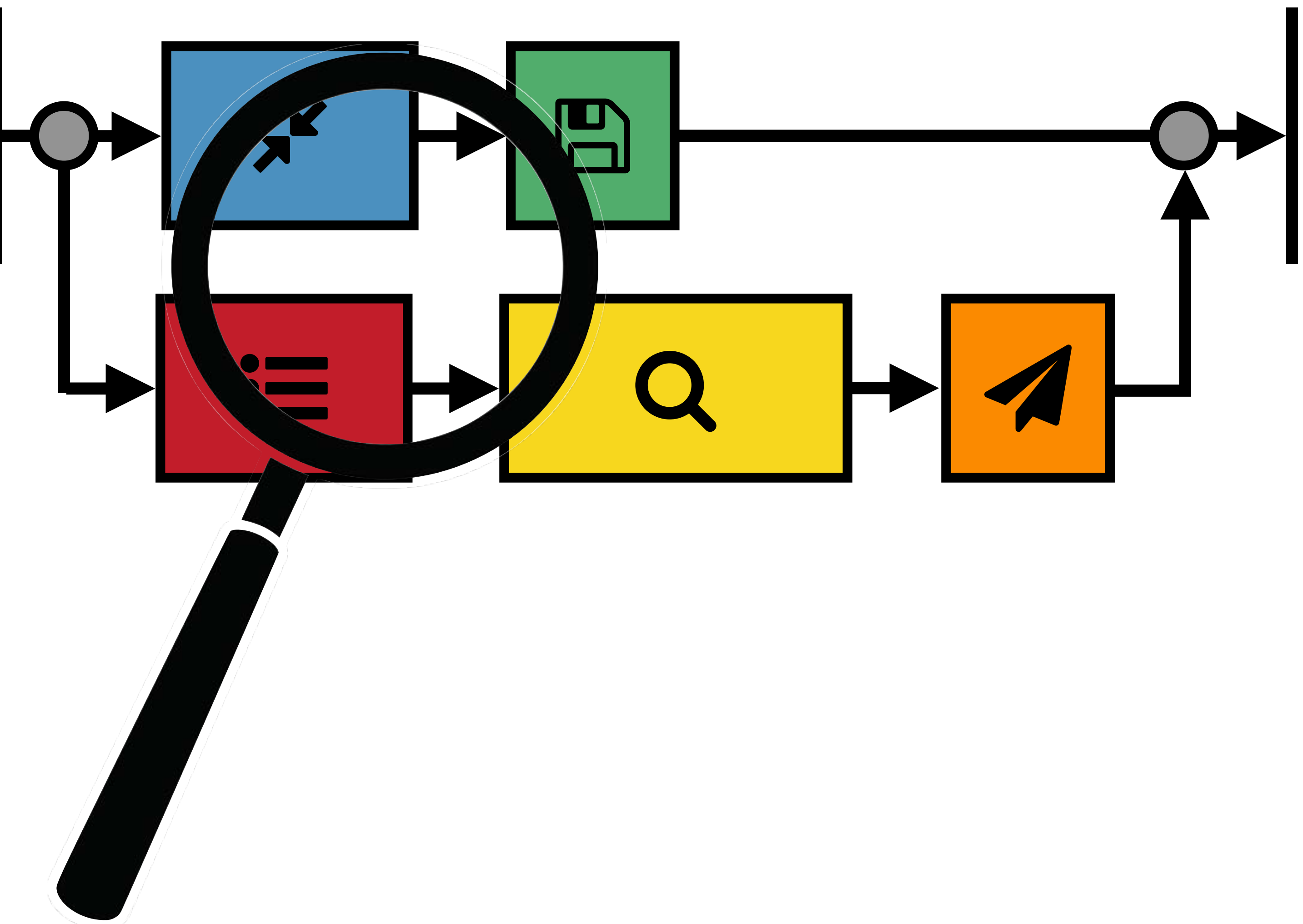
Number of calls to each function

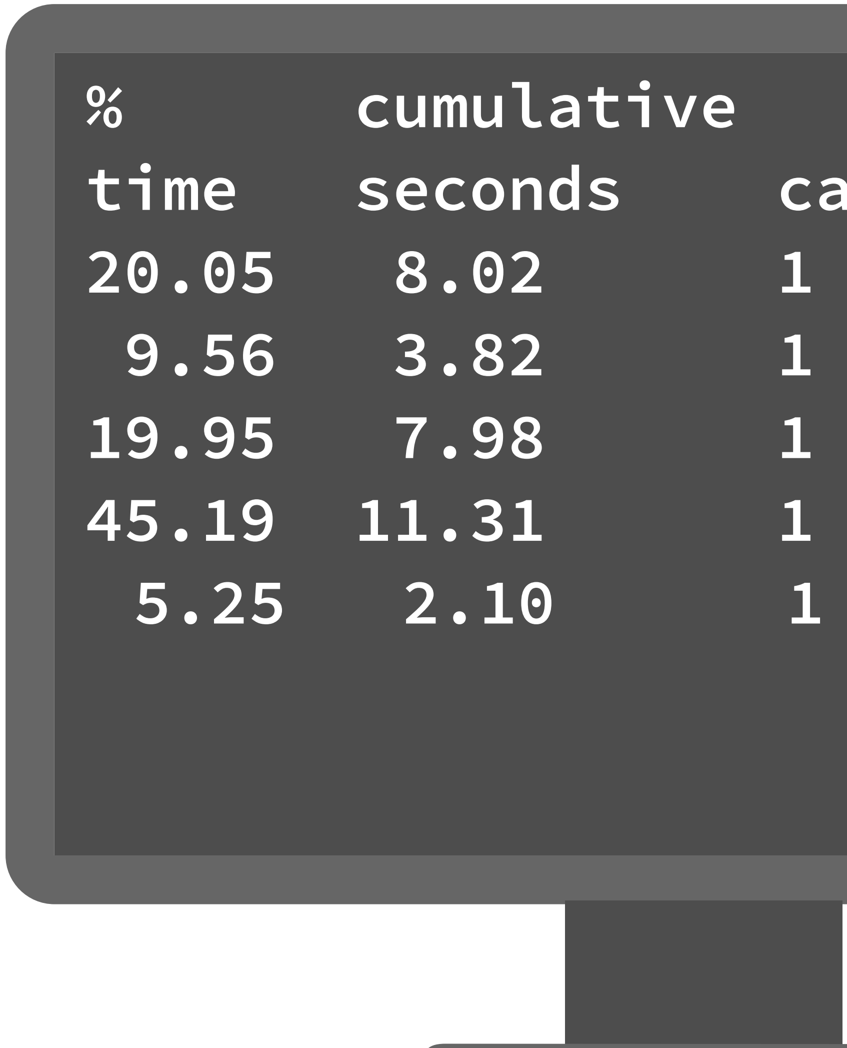


Software Profilers

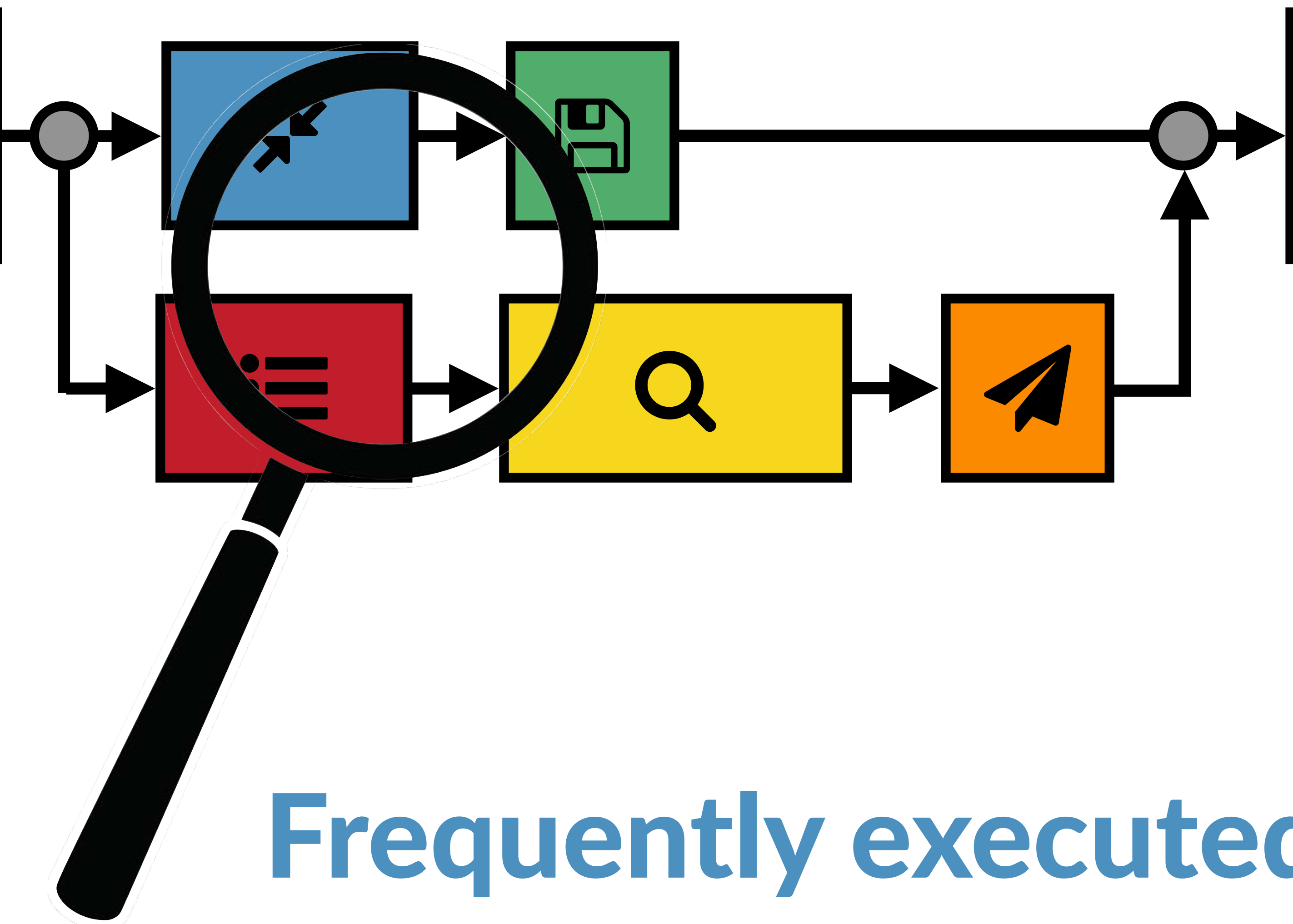


Software Profilers

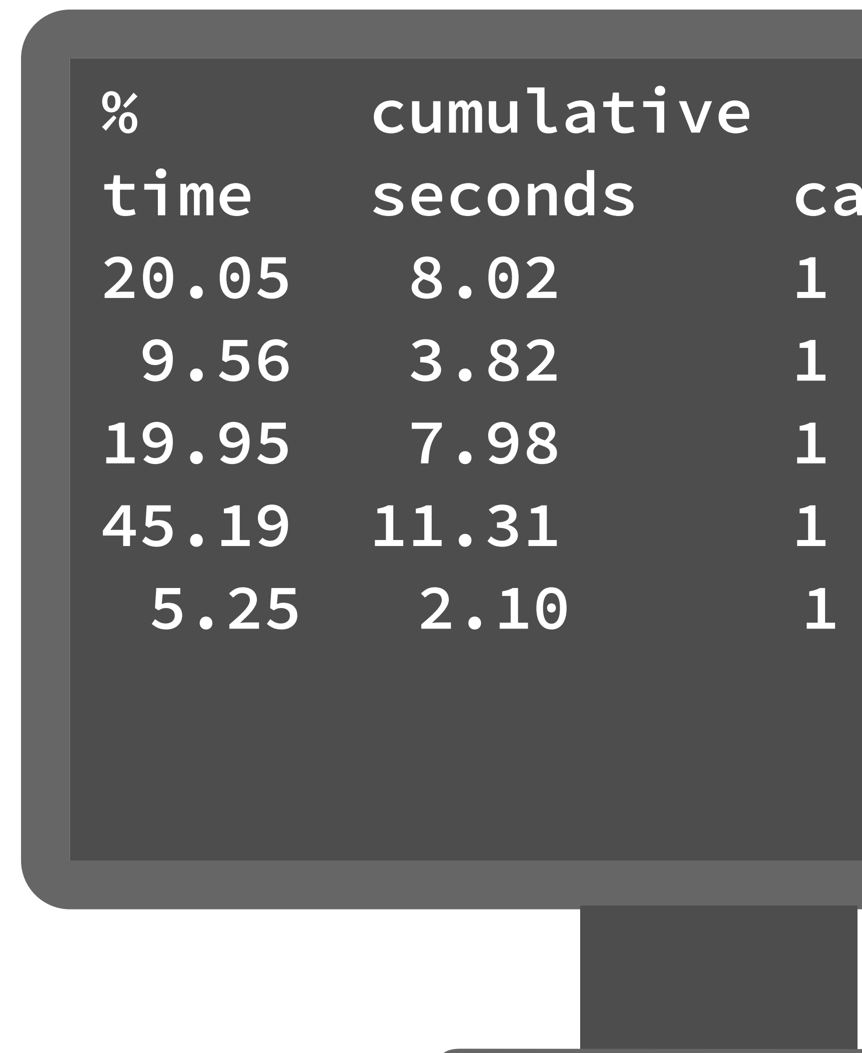




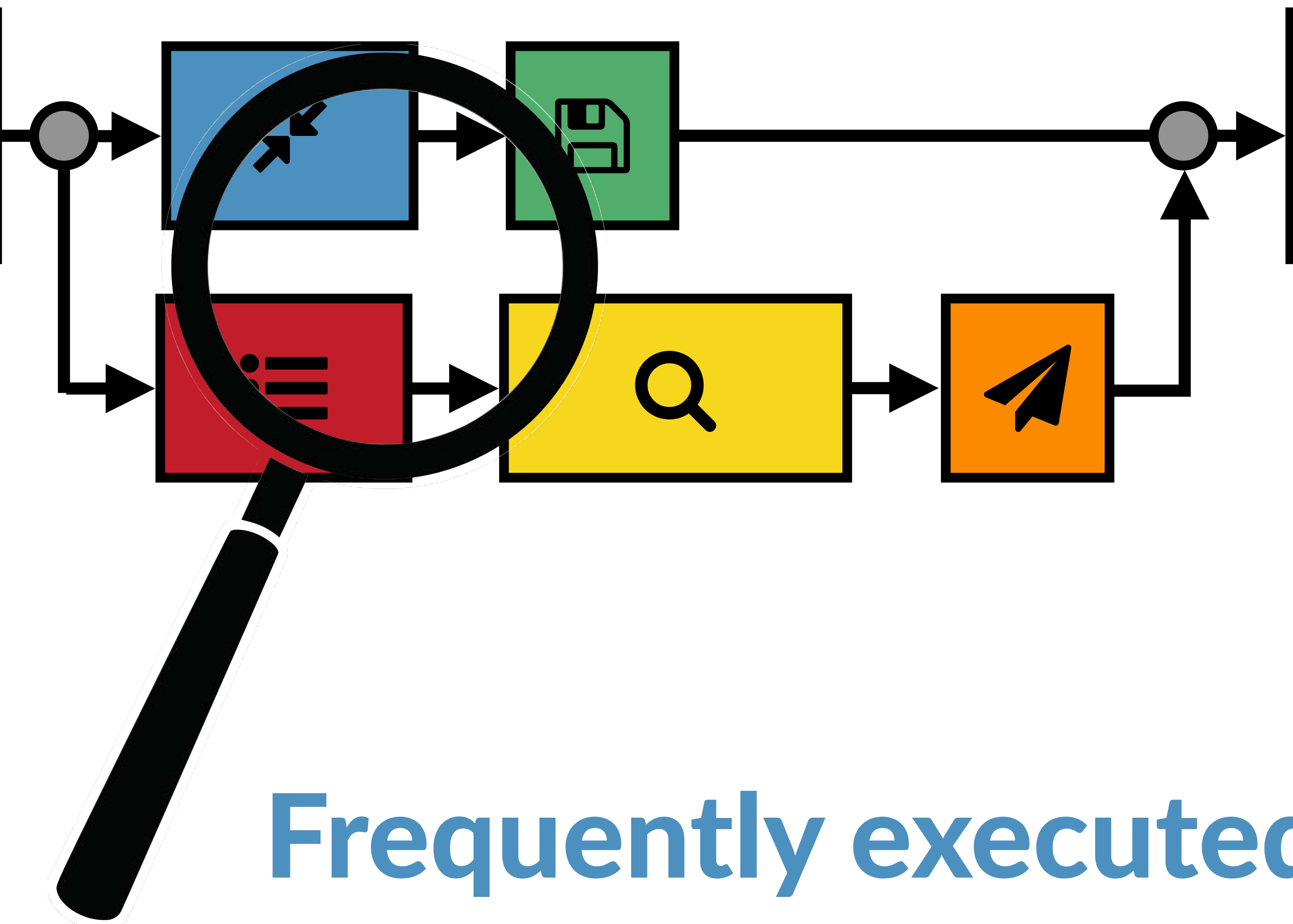
Software Profilers



Frequently executed code

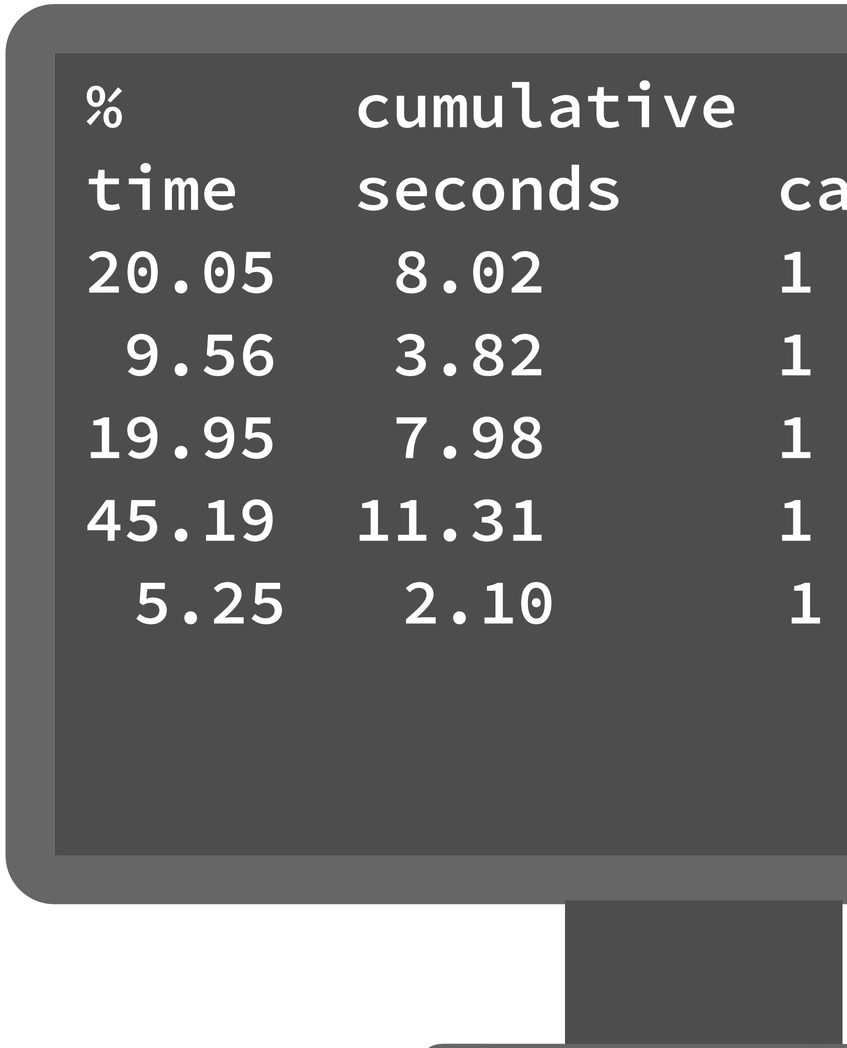


Software Profilers



Frequently executed code

Code that runs for a long time

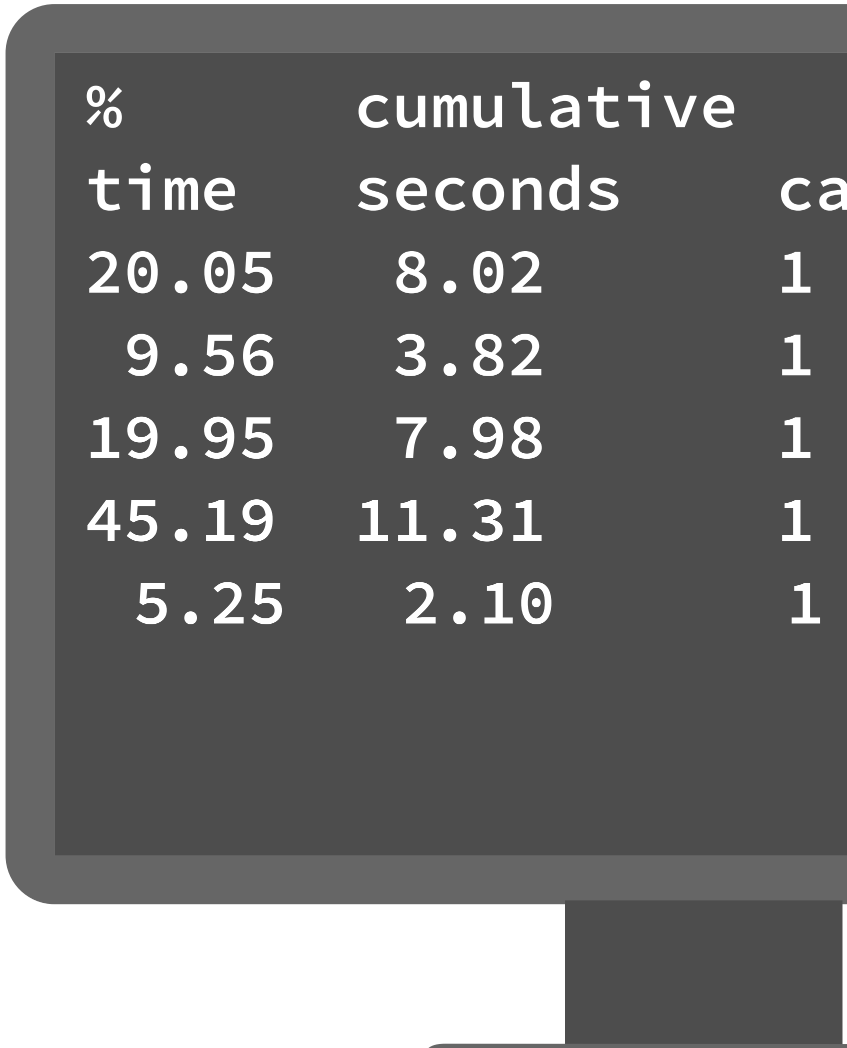


Software Profilers

Are these places where Homer should focus on performance?

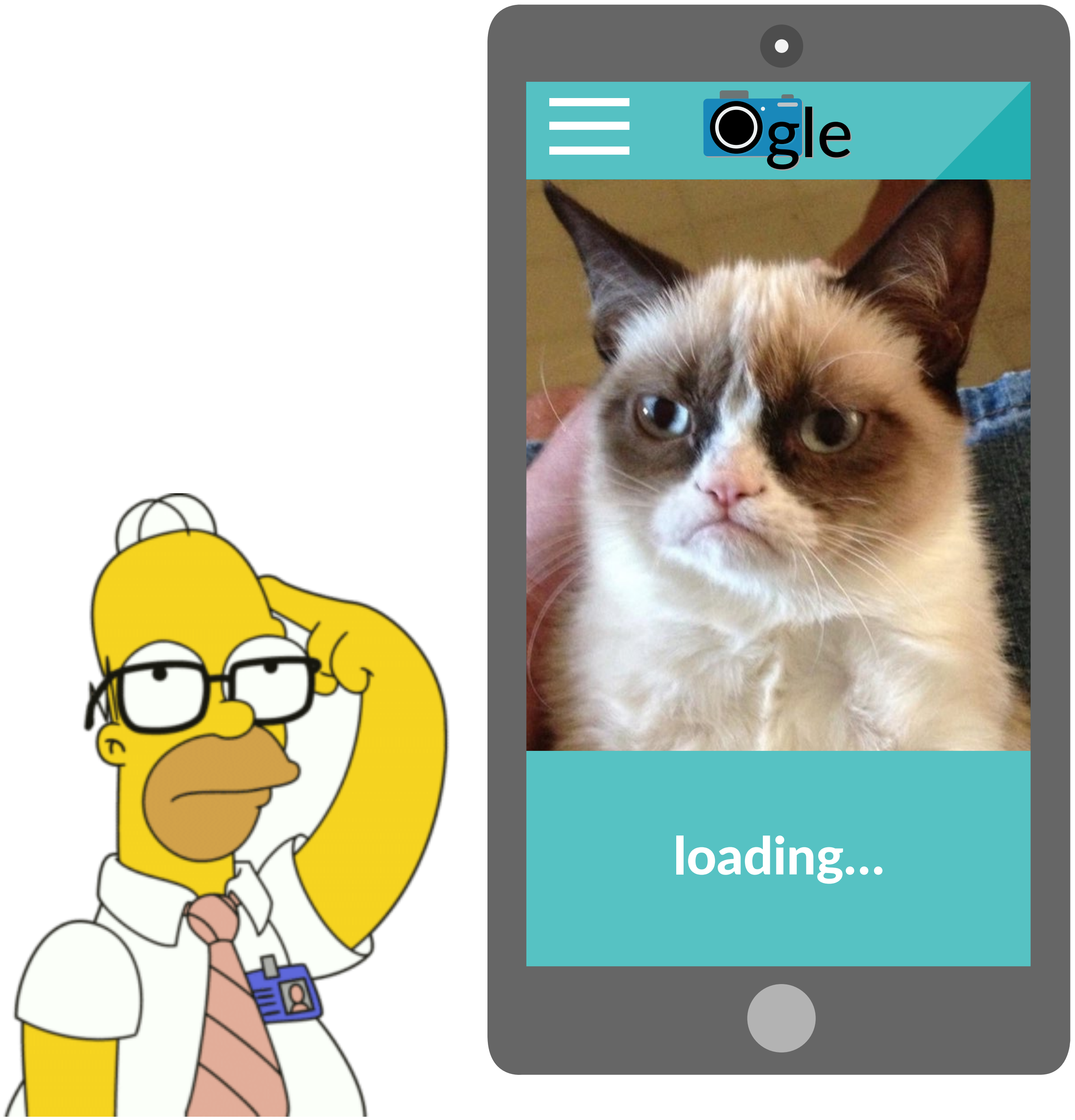
Frequently executed code

Code that runs for a long time

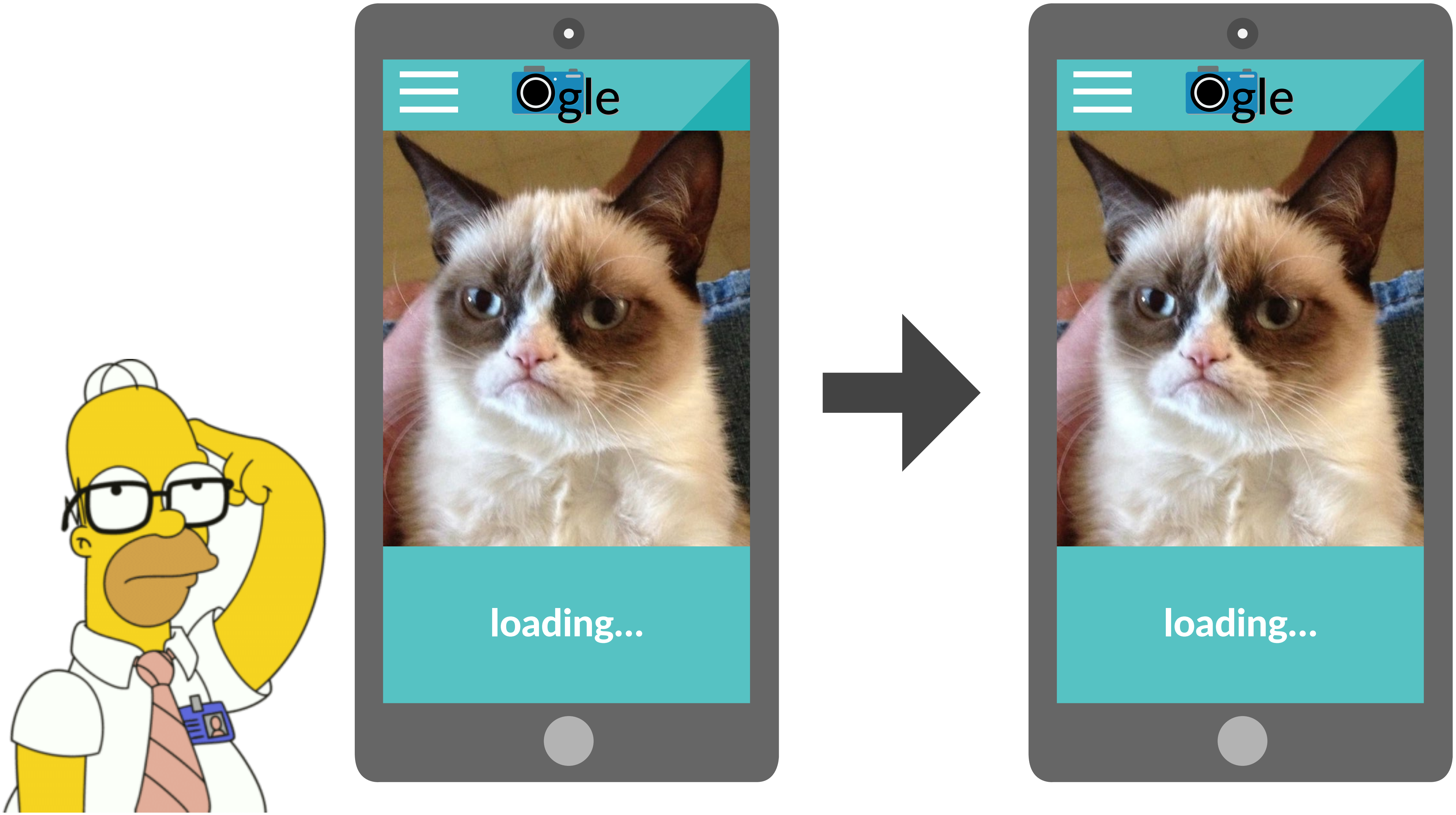


Would this speed up Ogle?

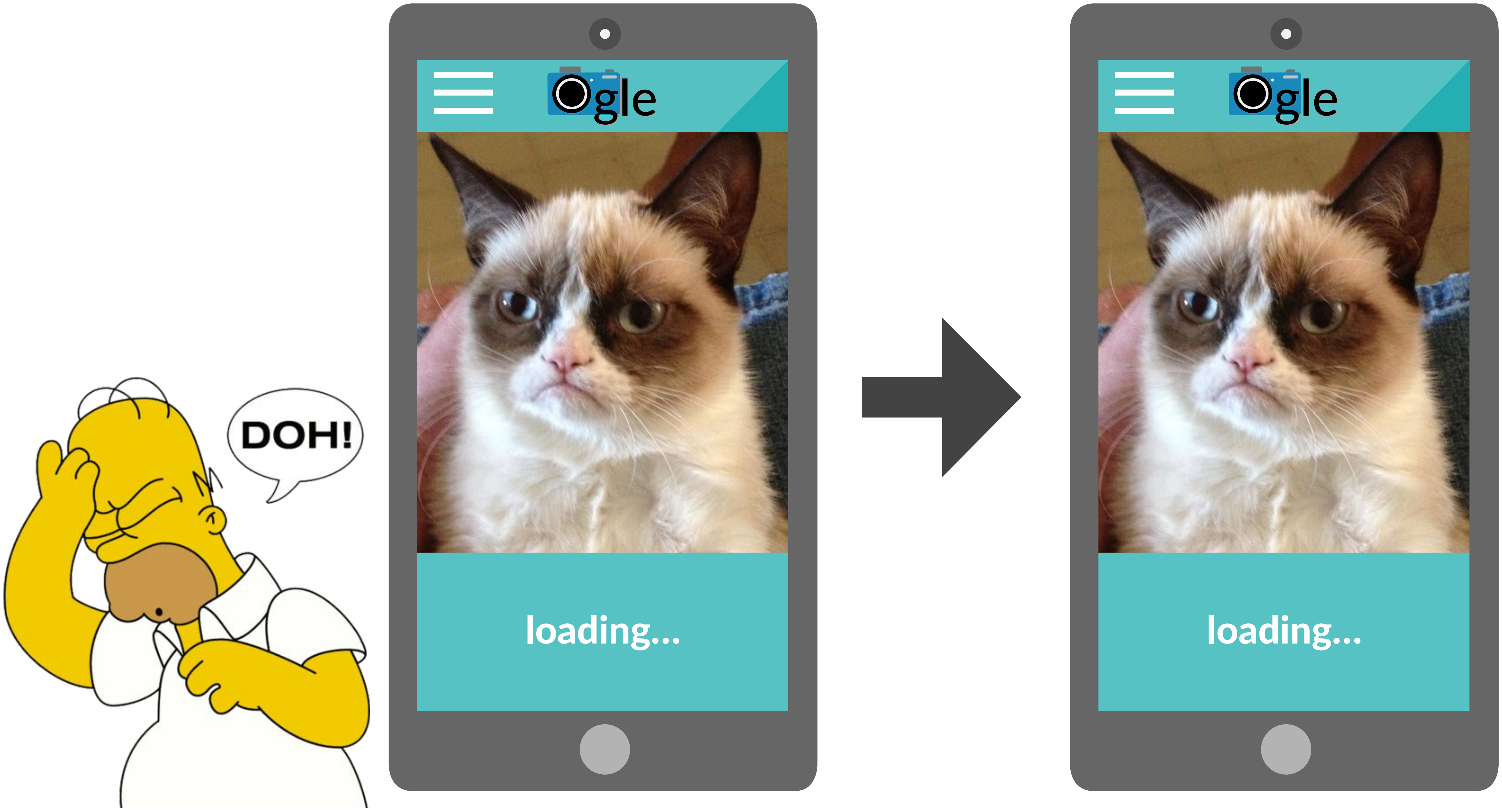
Would this speed up Ogle?



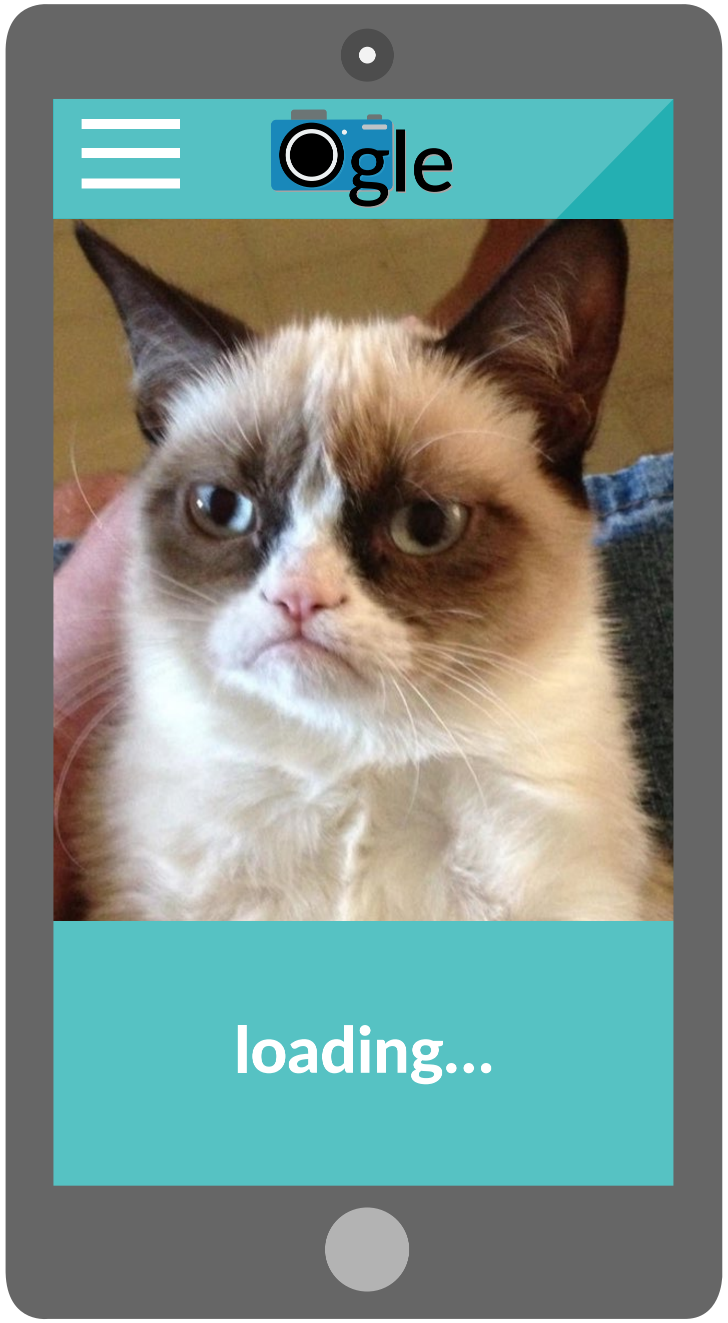
Would this speed up Ogle?



Would this speed up Ogle?

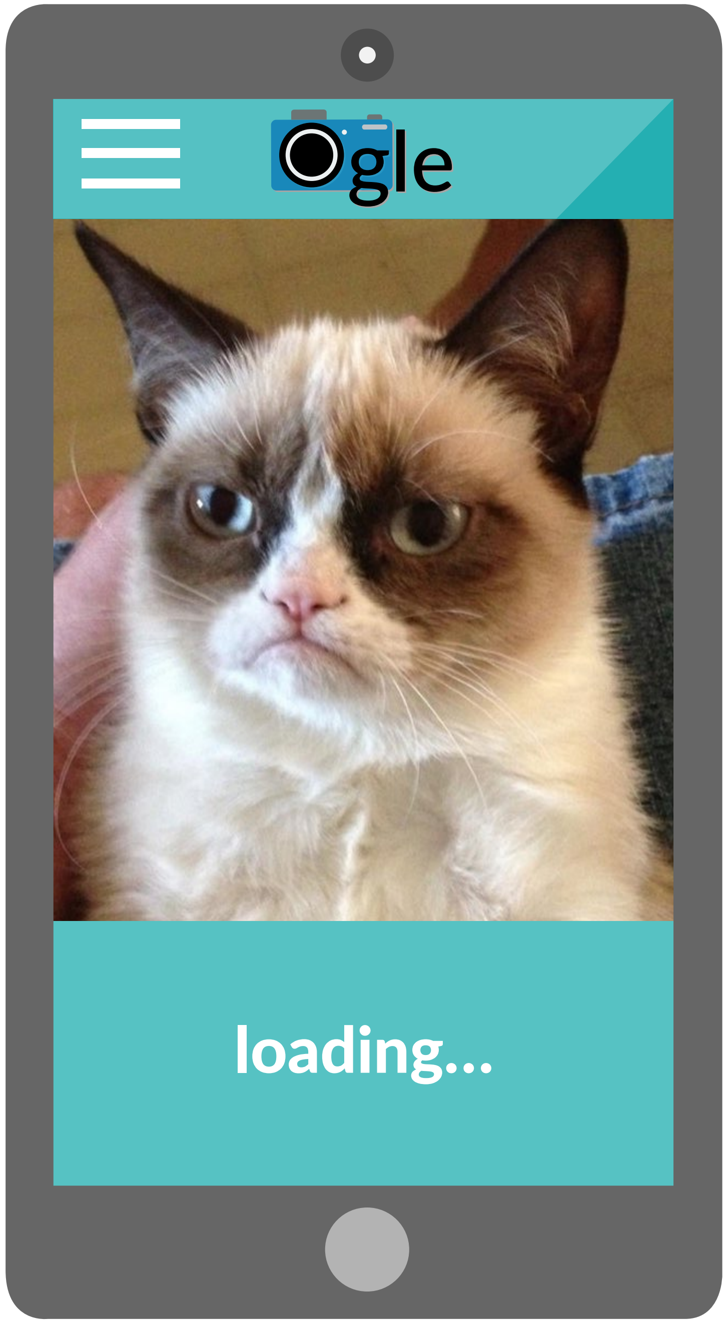


Would this speed up Ogle?



Would this speed up Ogle?

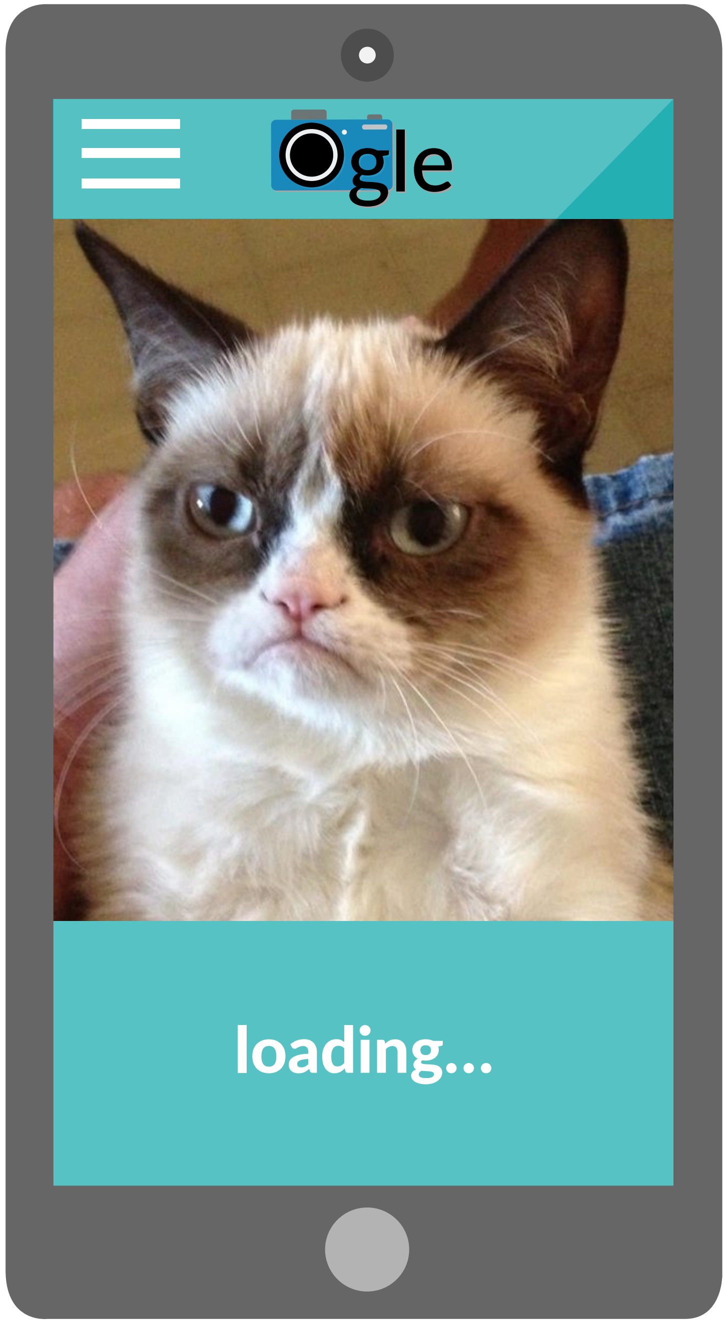
Frequently executed code



Would this speed up Ogle?

Frequently executed code

Code that runs for a long time



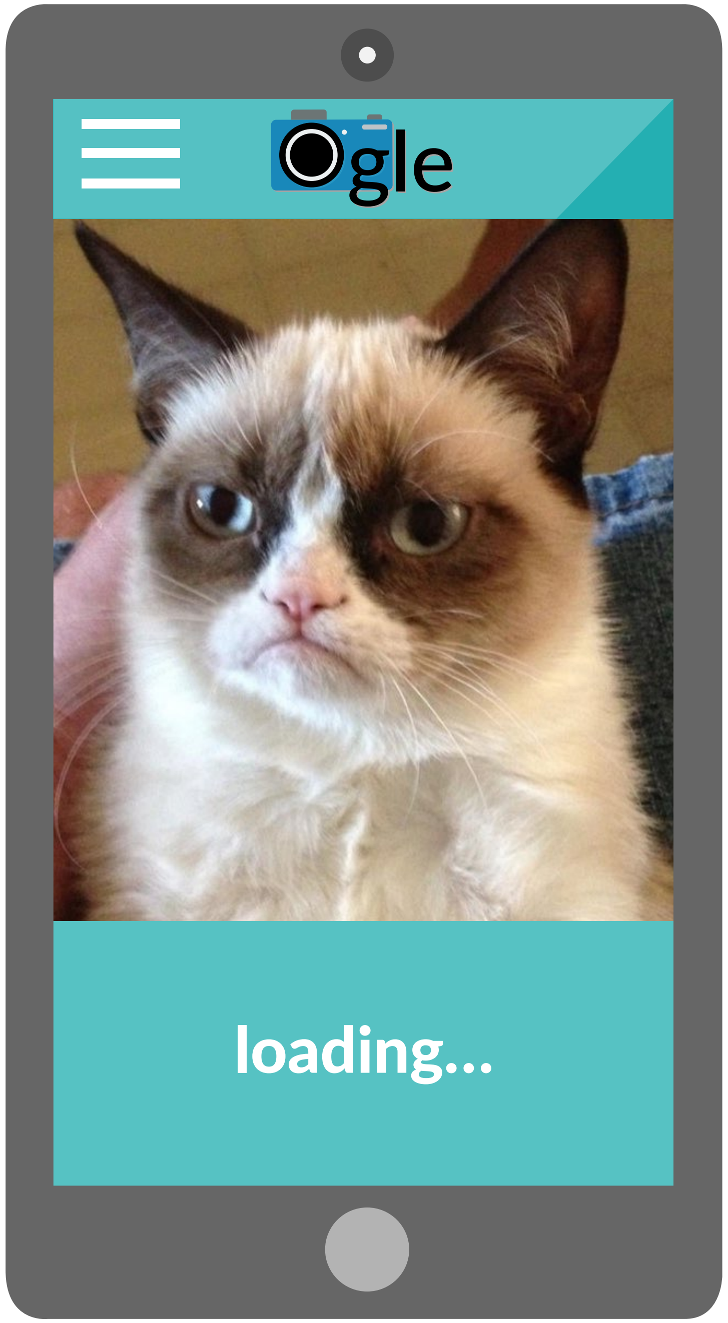
Would this speed up Ogle?

Frequently executed code

Code that runs for a long time

Profilers do a bad job finding important code in modern applications!

parallel / asynchronous / concurrent



Would this speed up Ogle?

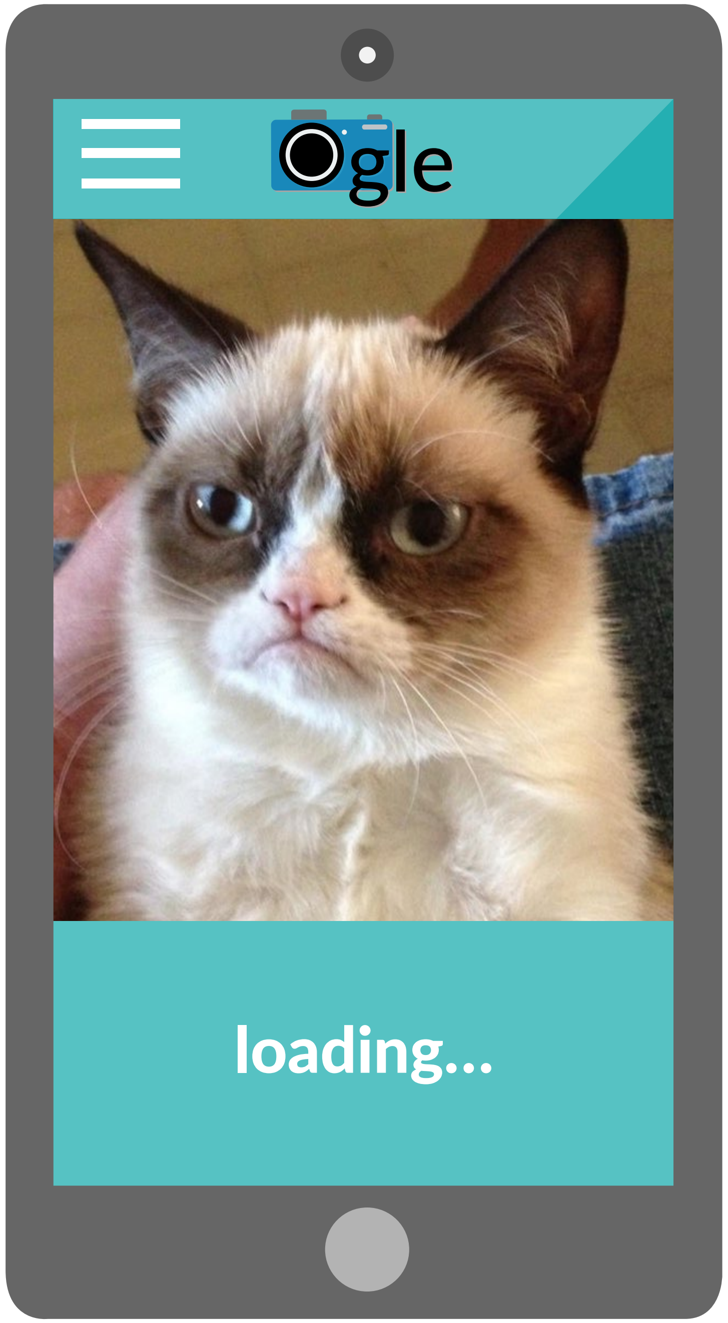
Frequently executed code

Code that runs for a long time

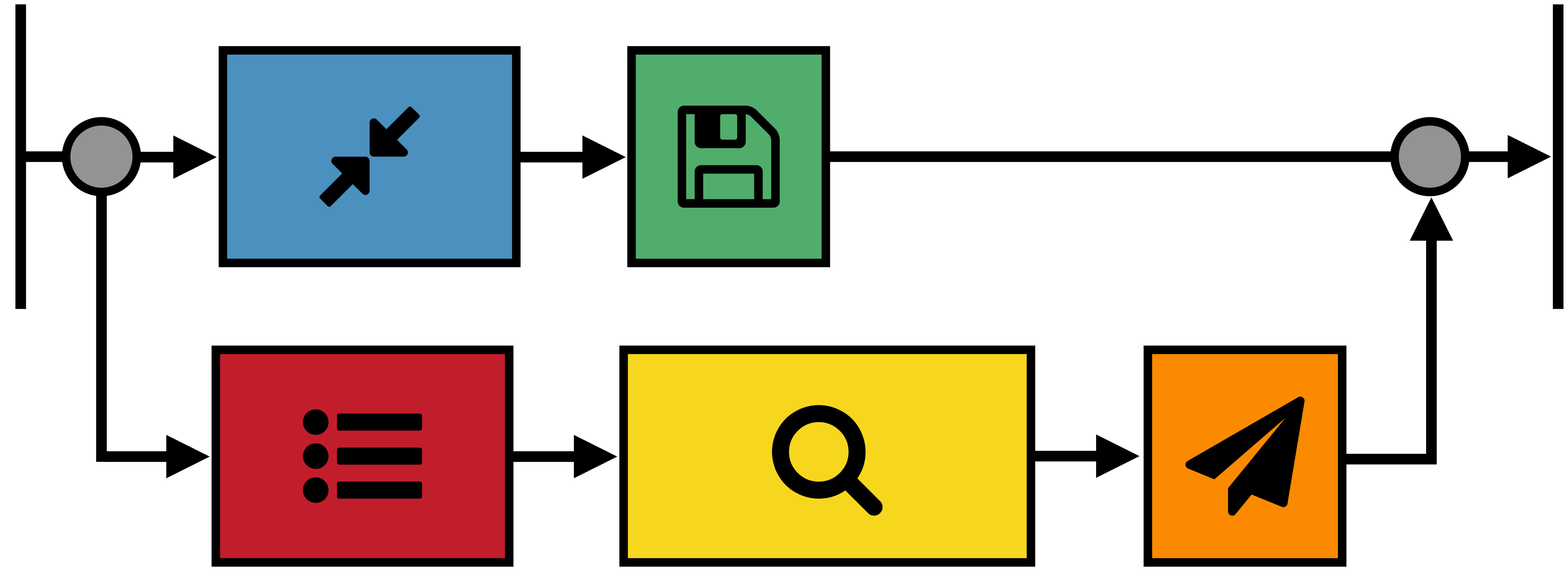
Profilers do a bad job finding important code in modern applications!

parallel / asynchronous / concurrent

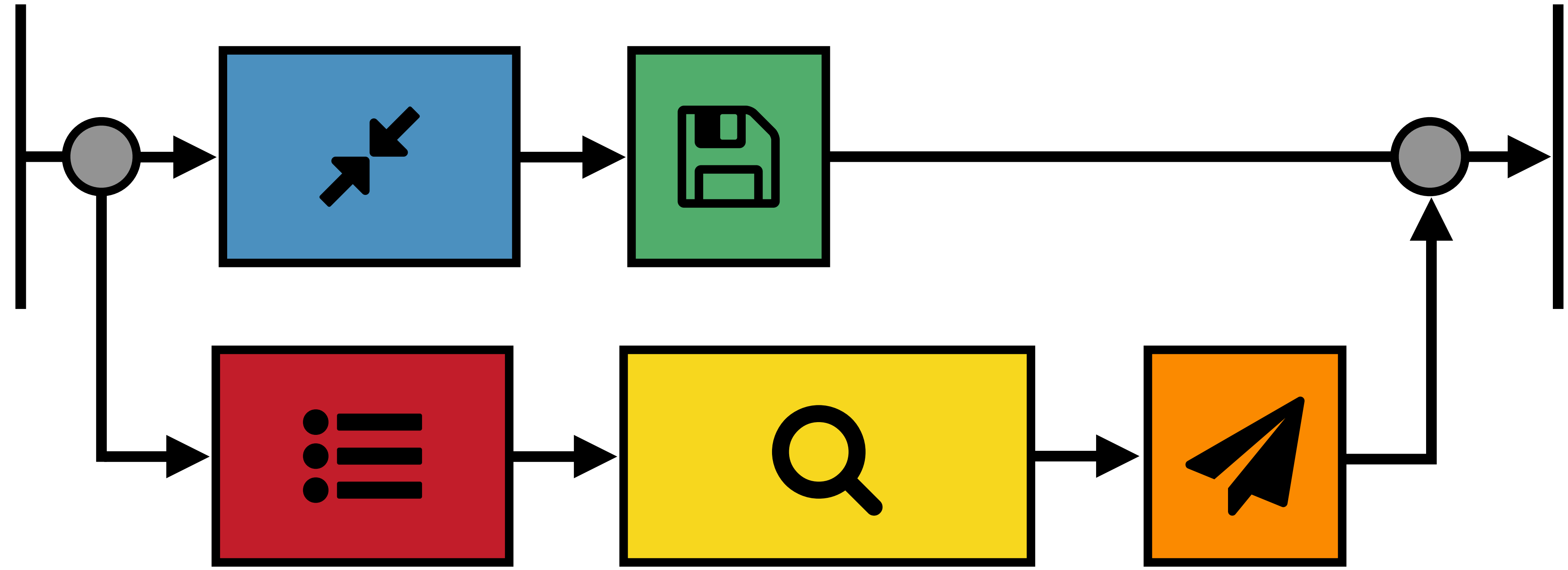
We need to do better!



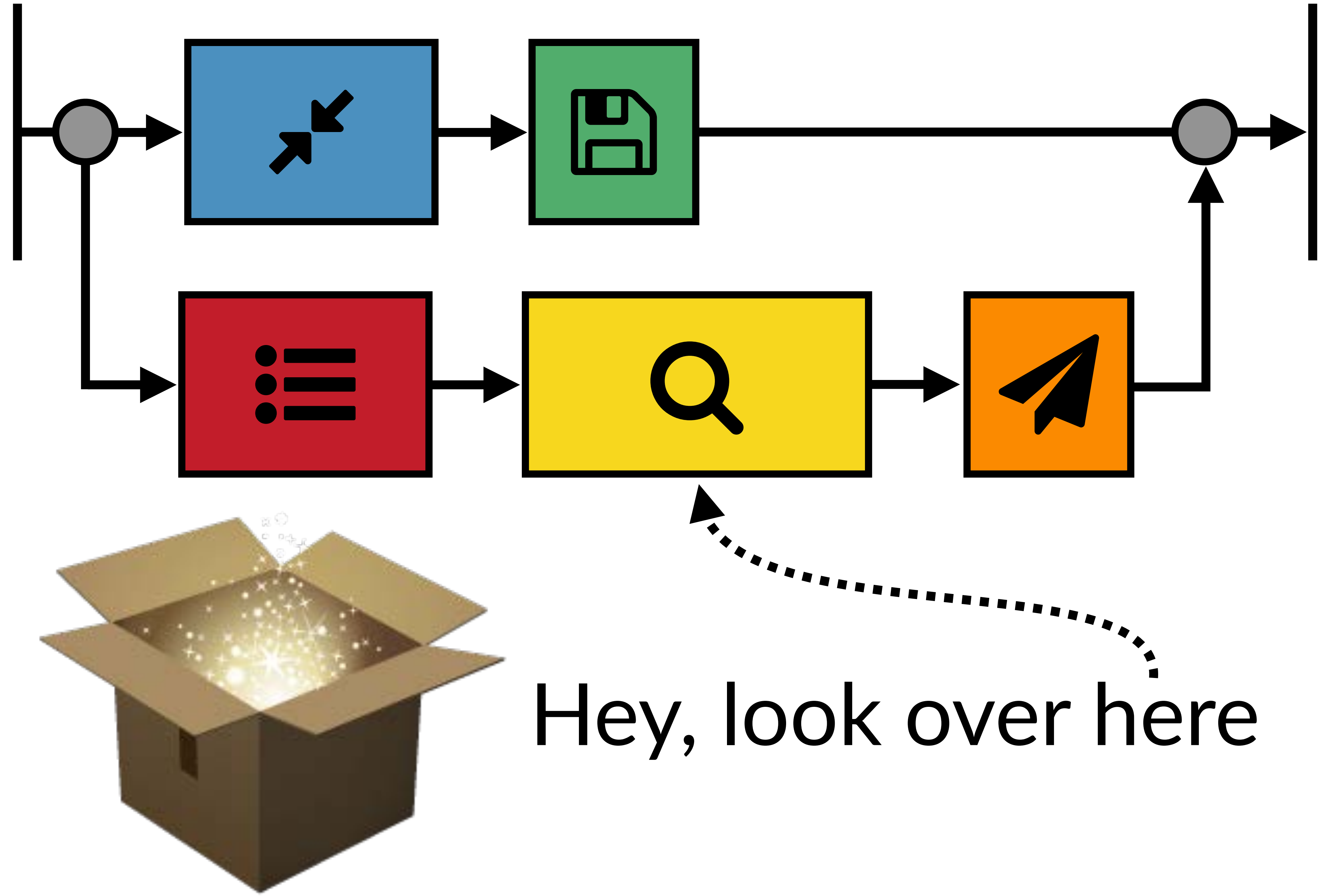
What *would* speed up Ogle?



What *would* speed up Ogle?

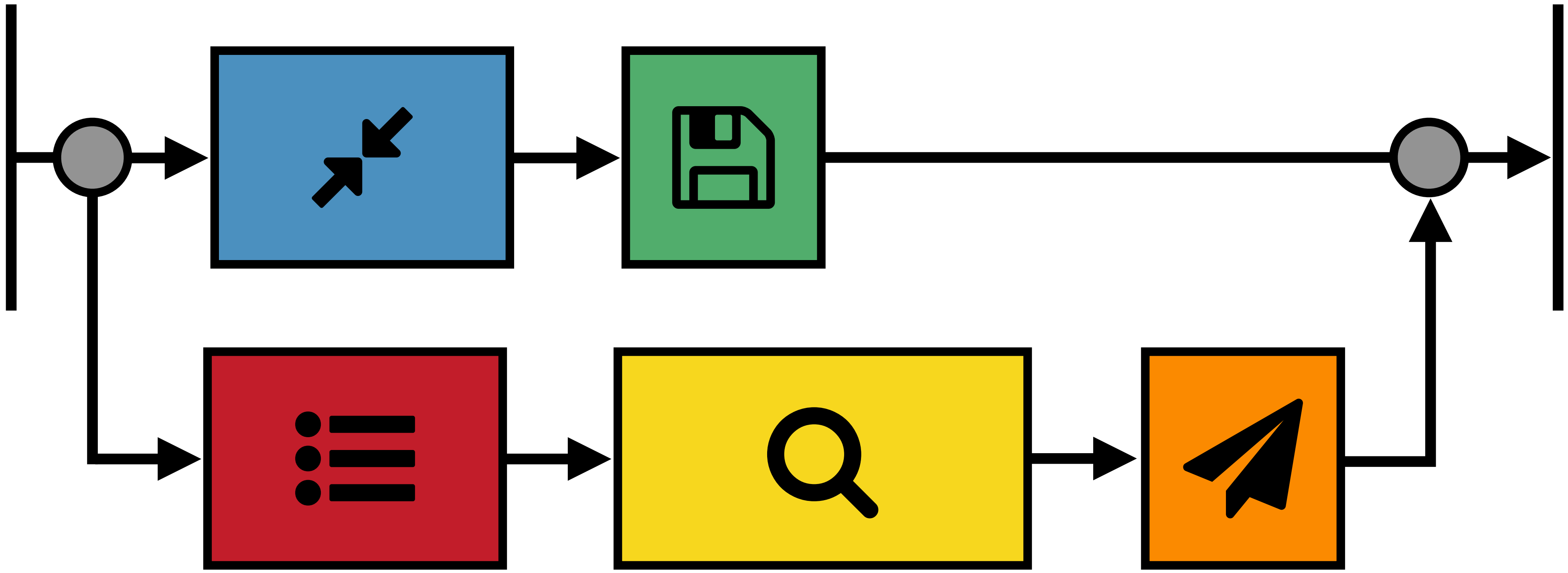


What *would* speed up Ogle?



Hey, look over here

What *would* speed up Ogle?



Hey, look over here

What would this information look like?

Causal Profile

Causal Profile

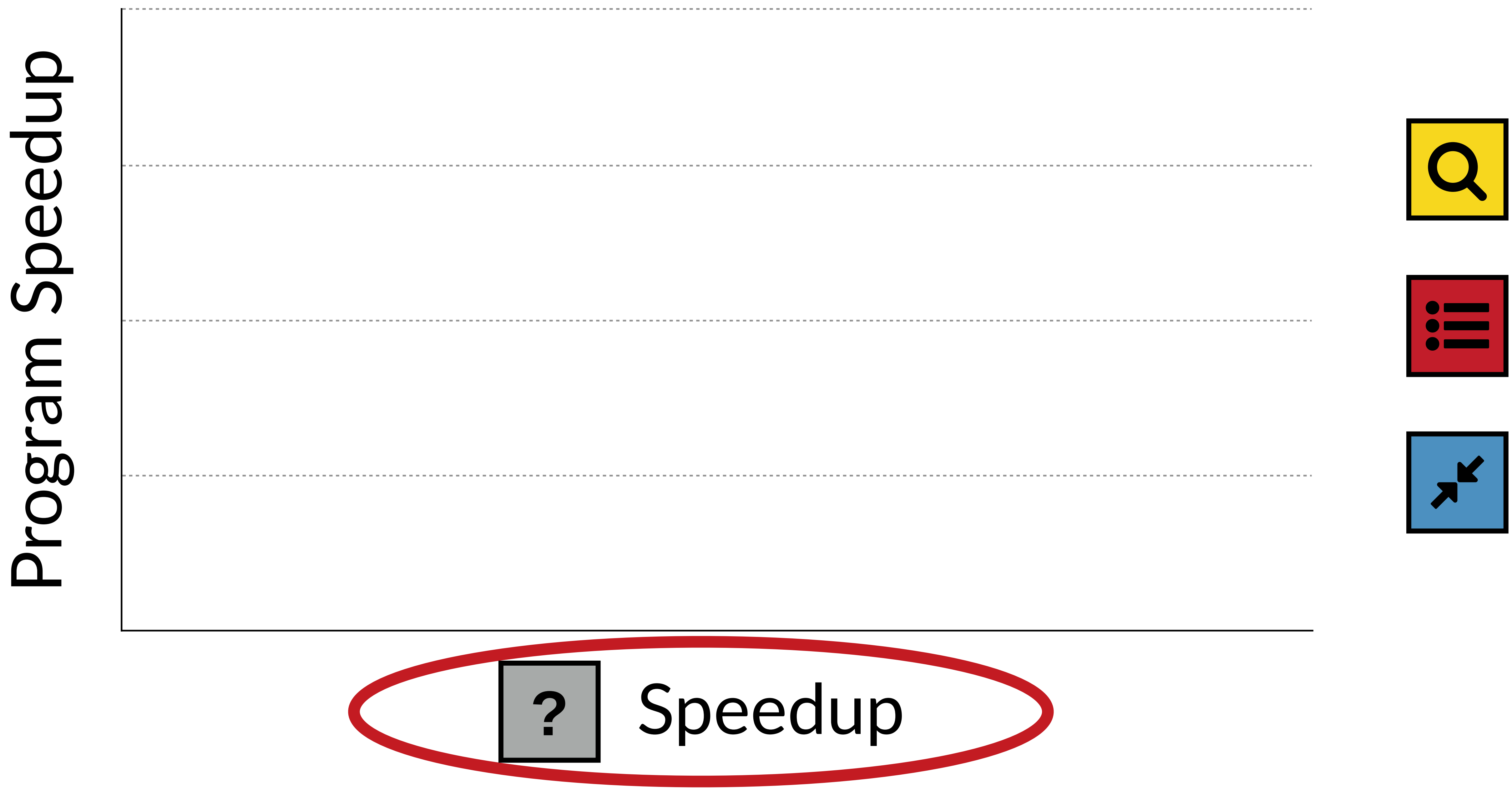
Tells you where optimizations will make a difference

Causal Profile

Tells you where optimizations
will make a difference

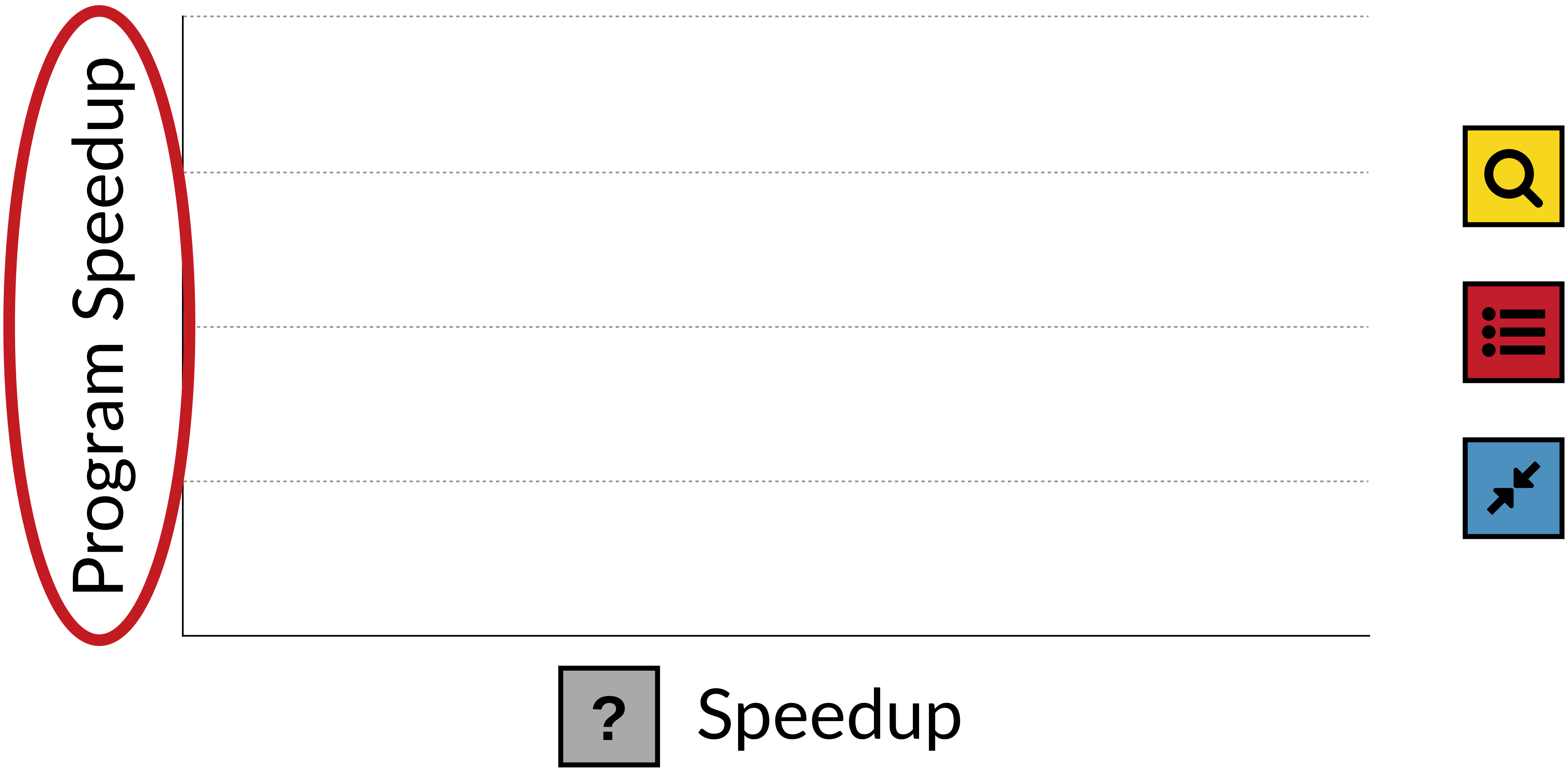
Causal Profile

Tells you where optimizations
will make a difference



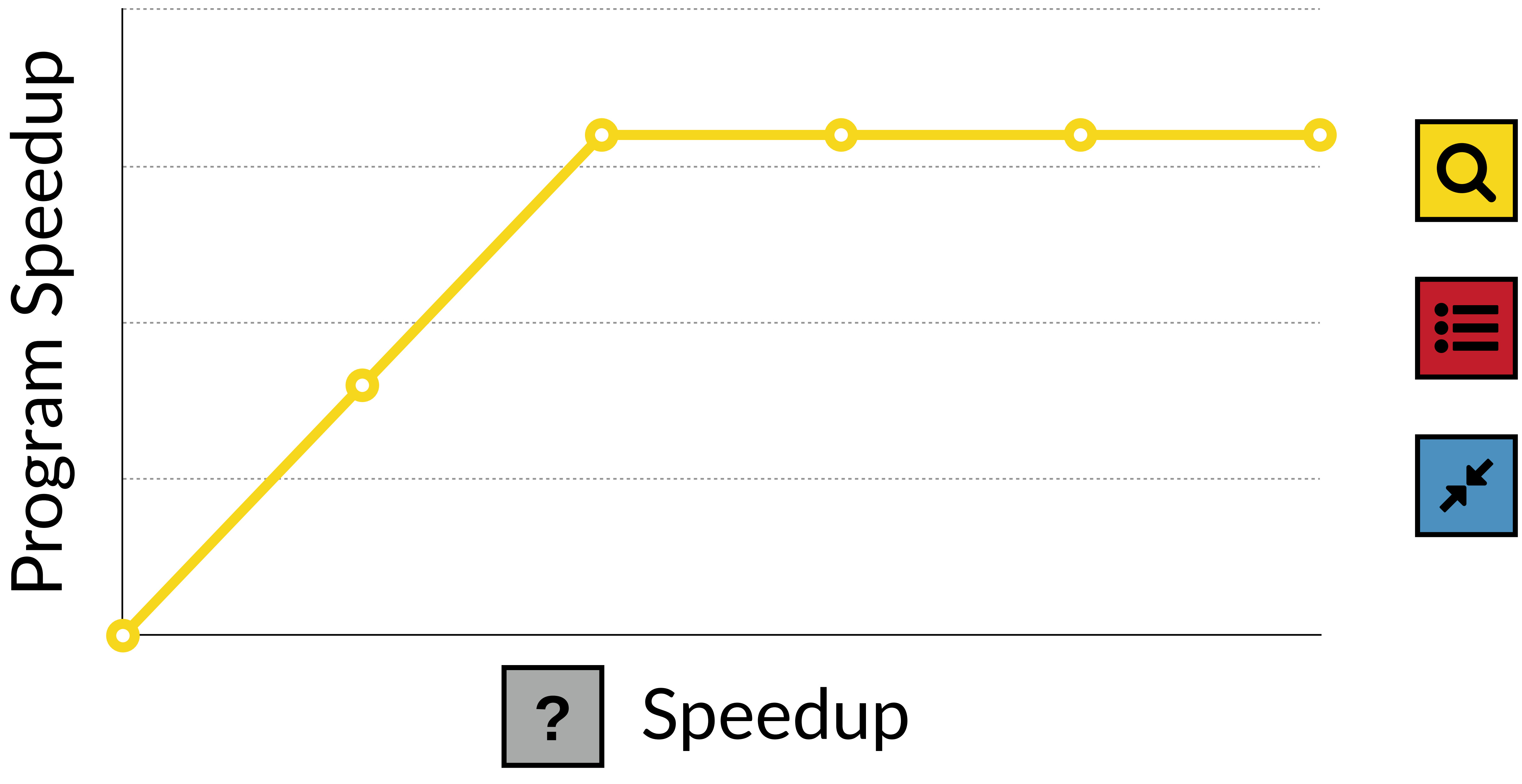
Causal Profile

Tells you where optimizations will make a difference



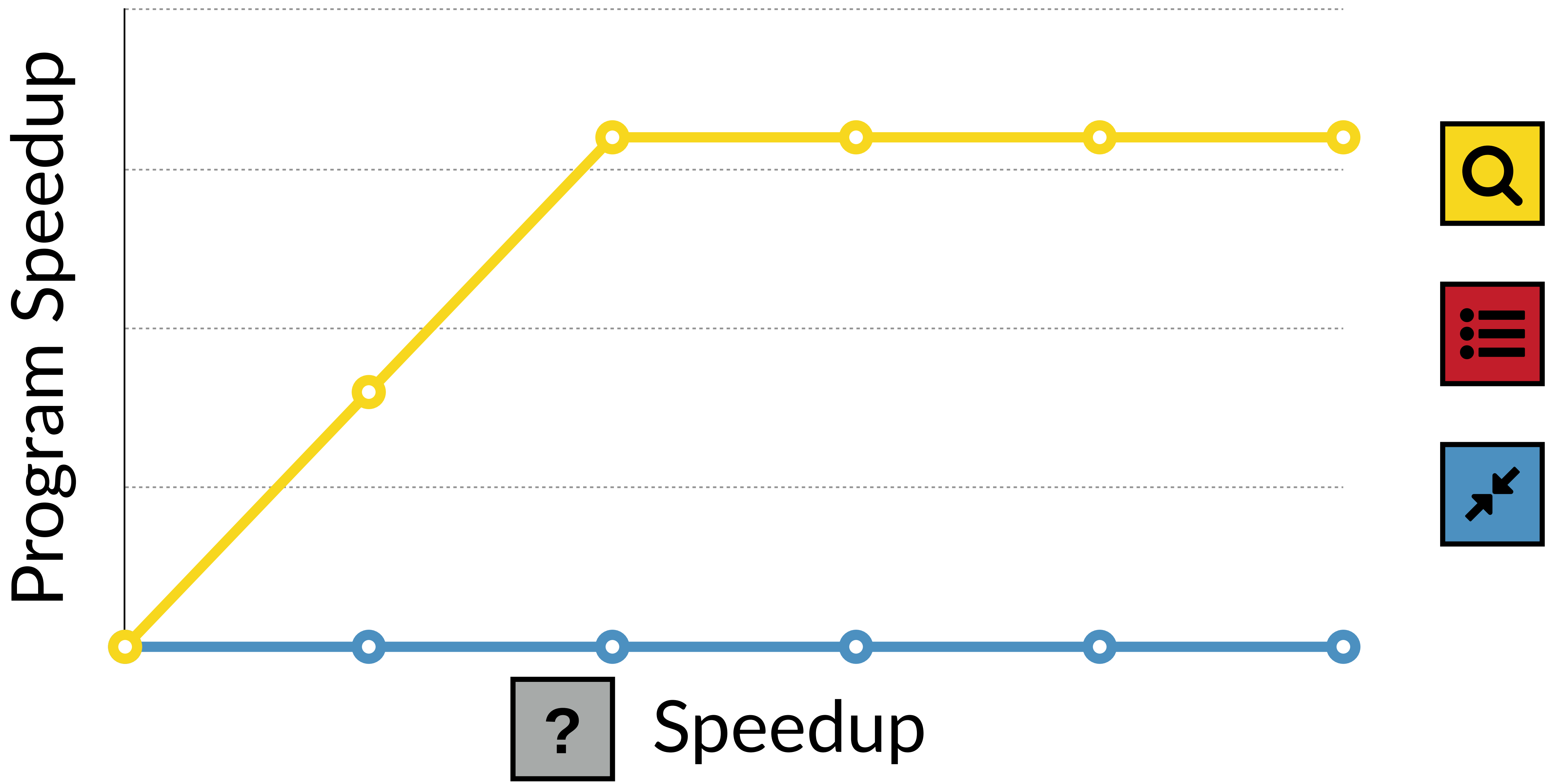
Causal Profile

Tells you where optimizations will make a difference



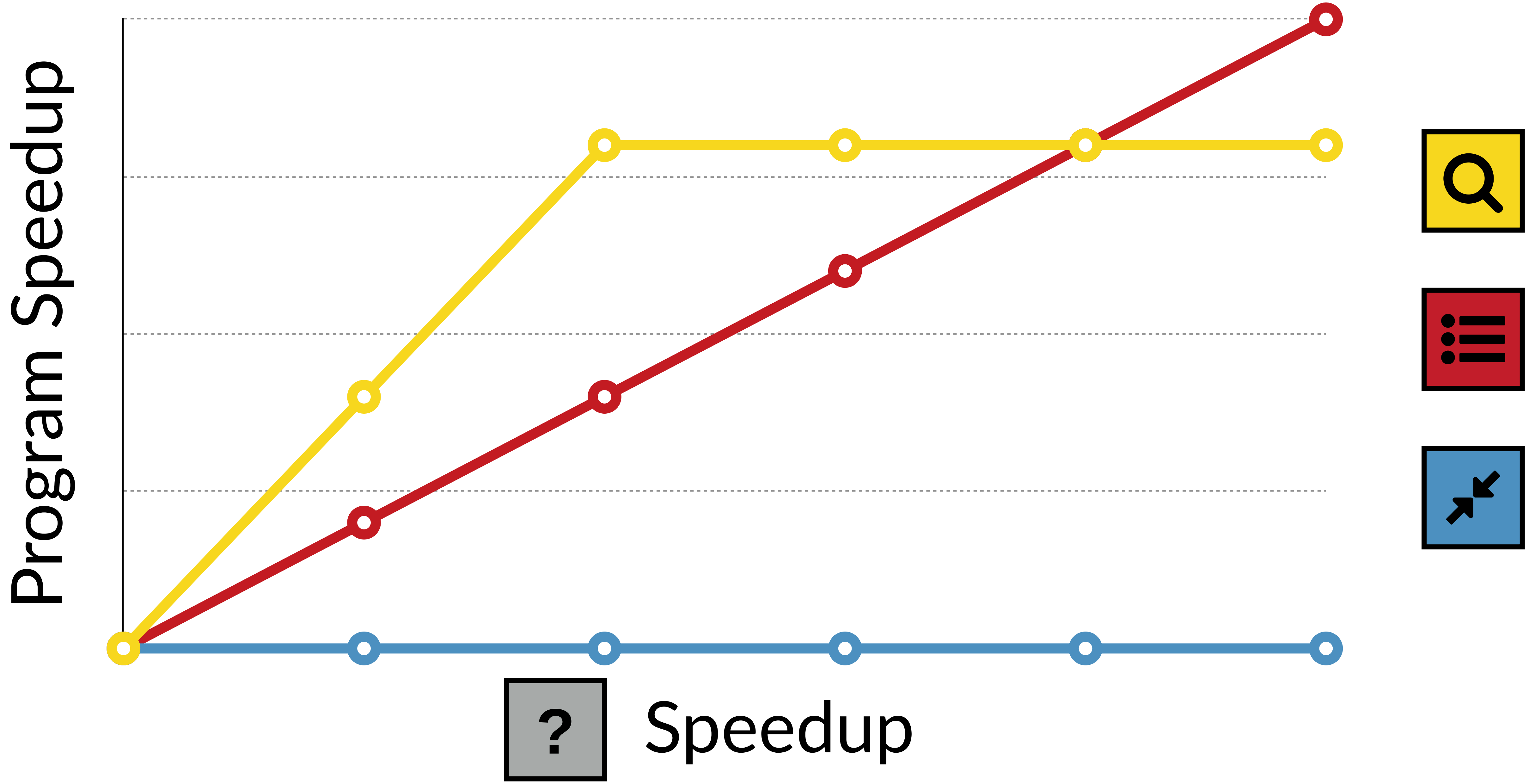
Causal Profile

Tells you where optimizations will make a difference

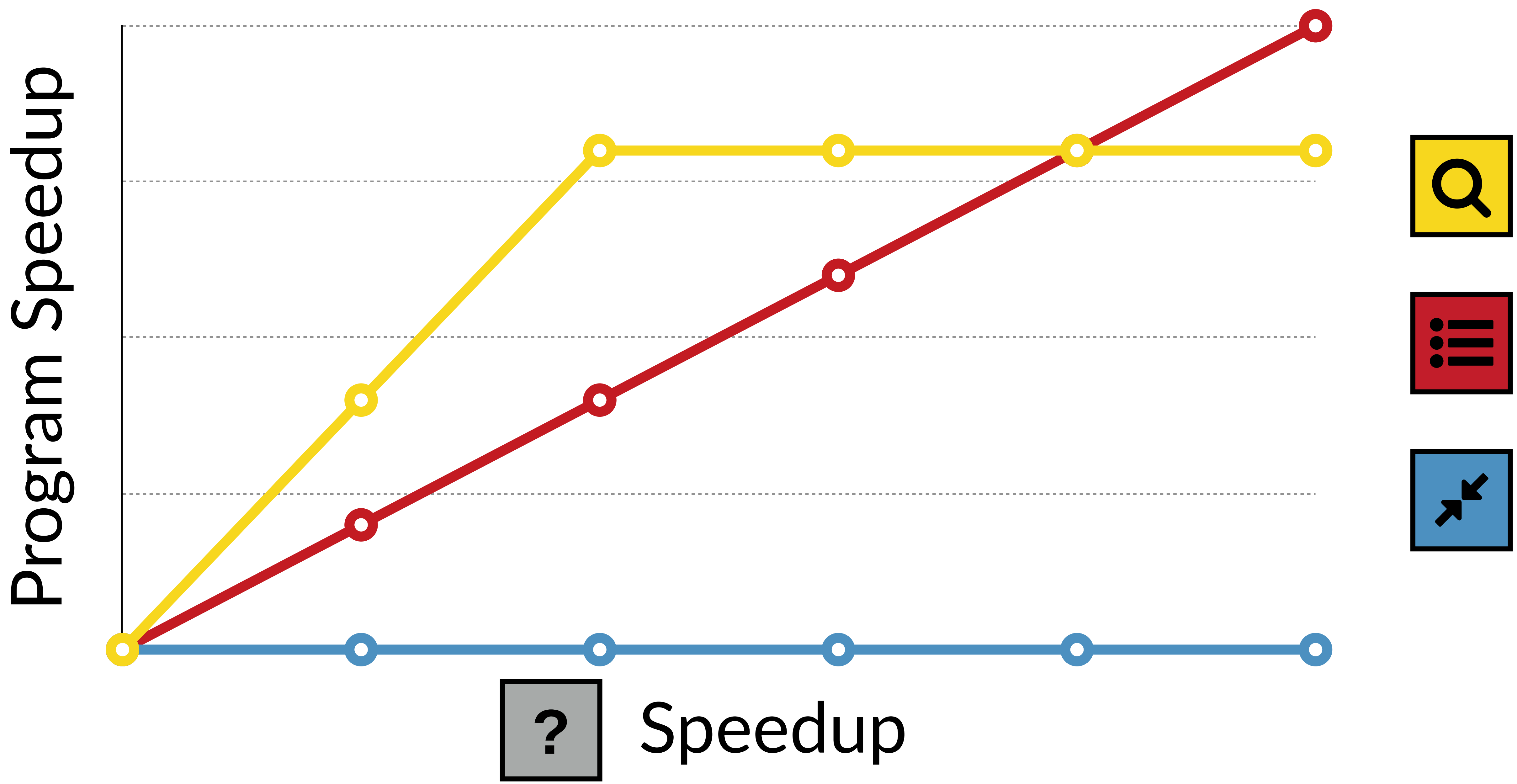


Causal Profile

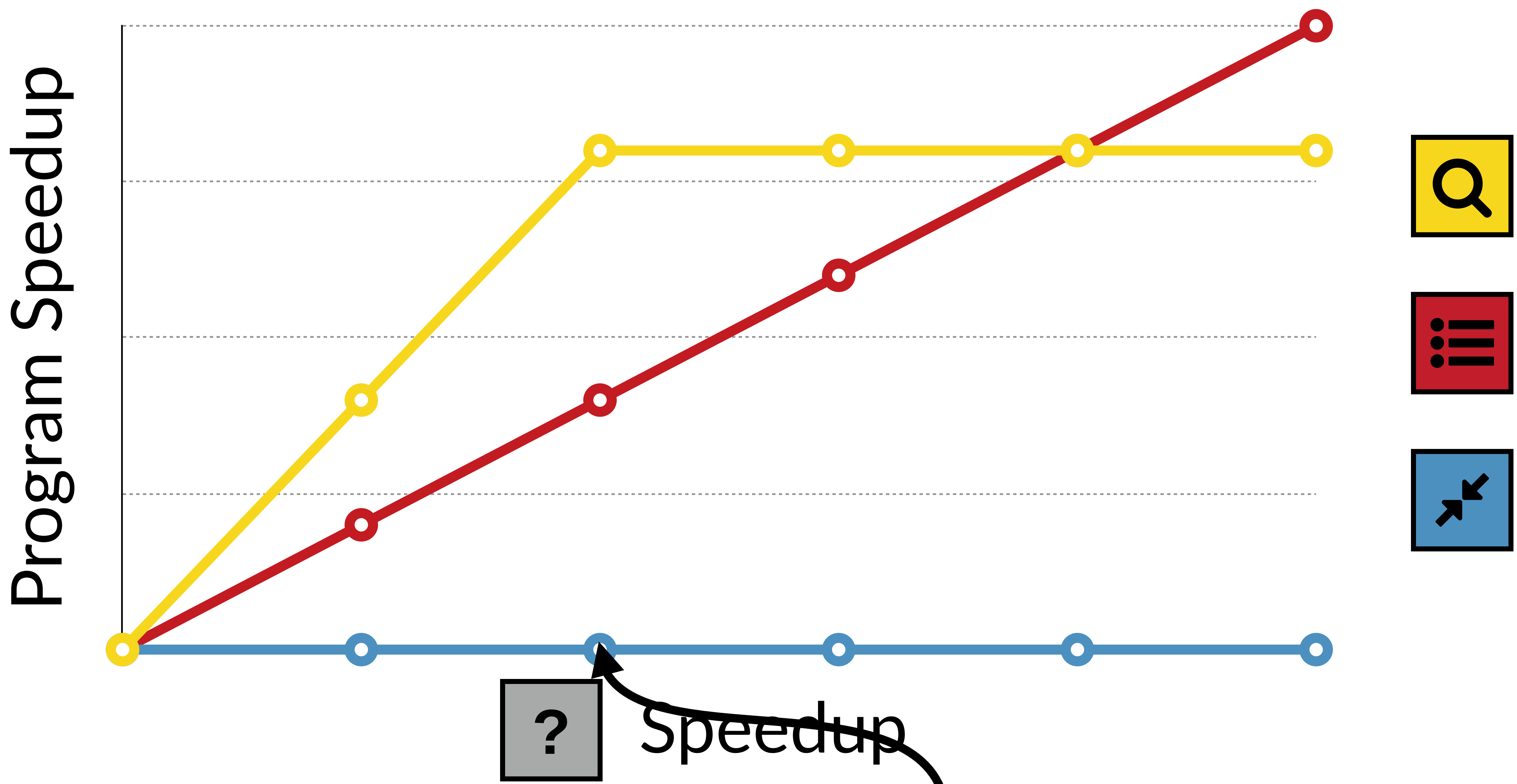
Tells you where optimizations will make a difference



Causal Profile

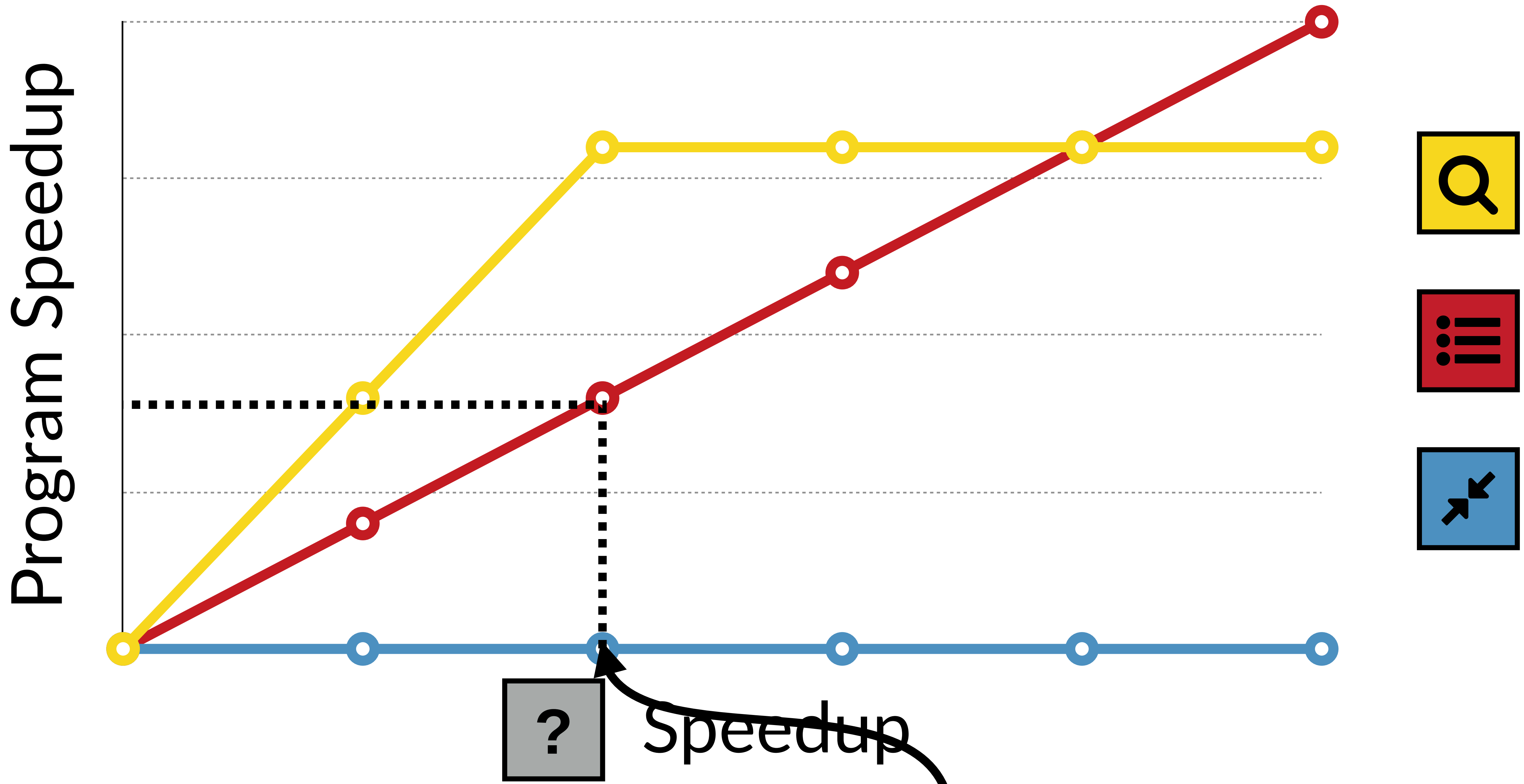


Causal Profile



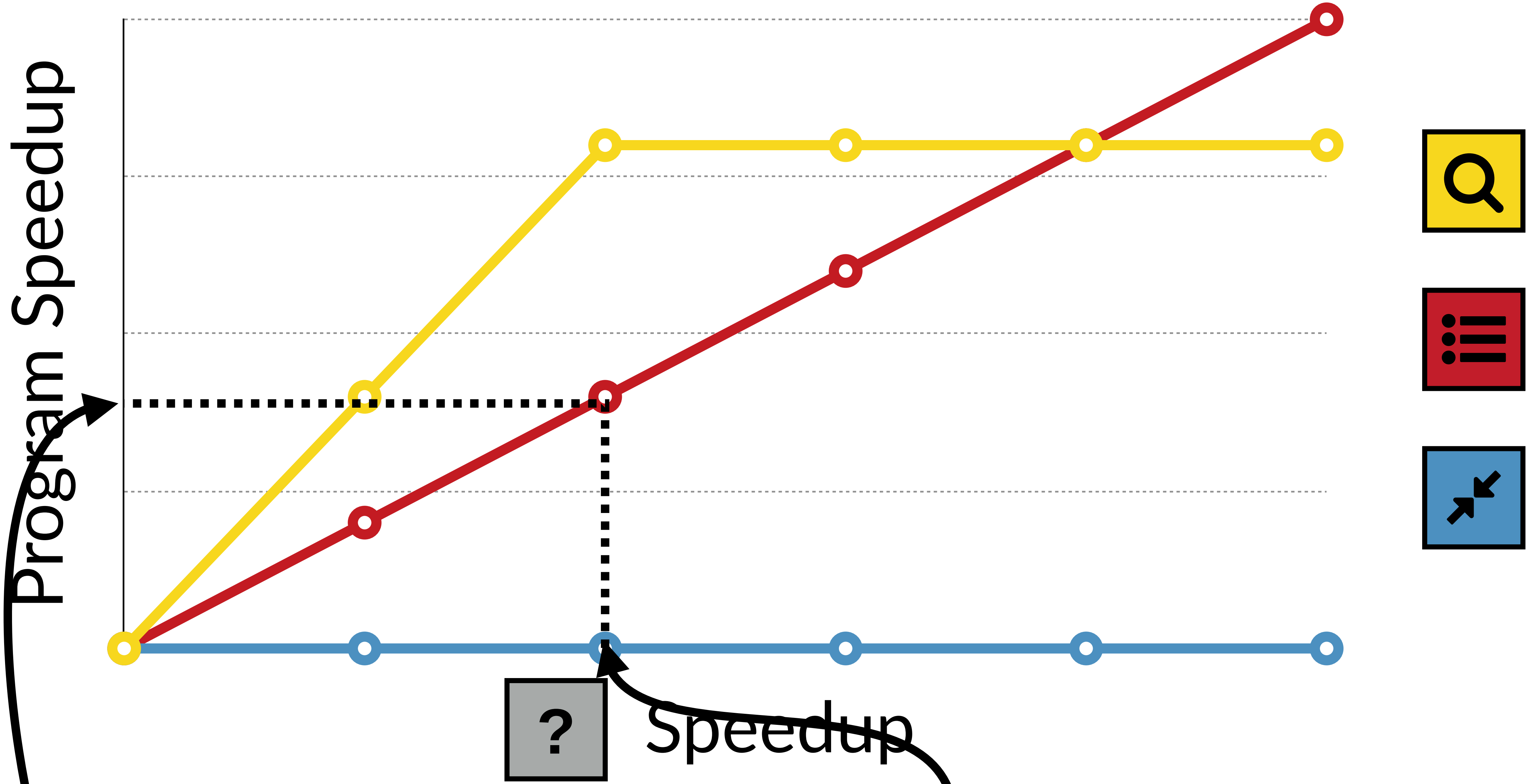
If you speed up this much

Causal Profile



If you speed up this much

Causal Profile

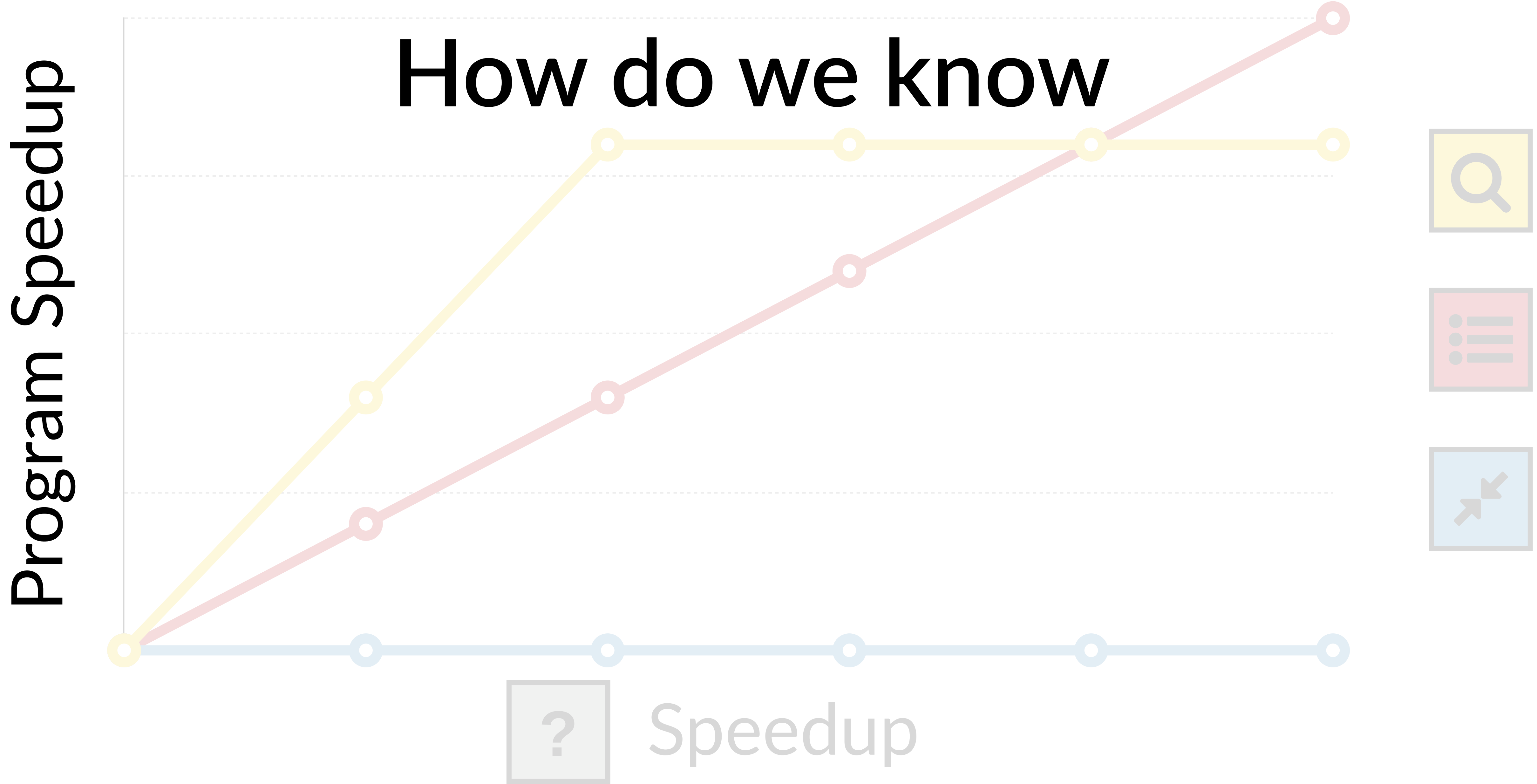


If you speed up this much

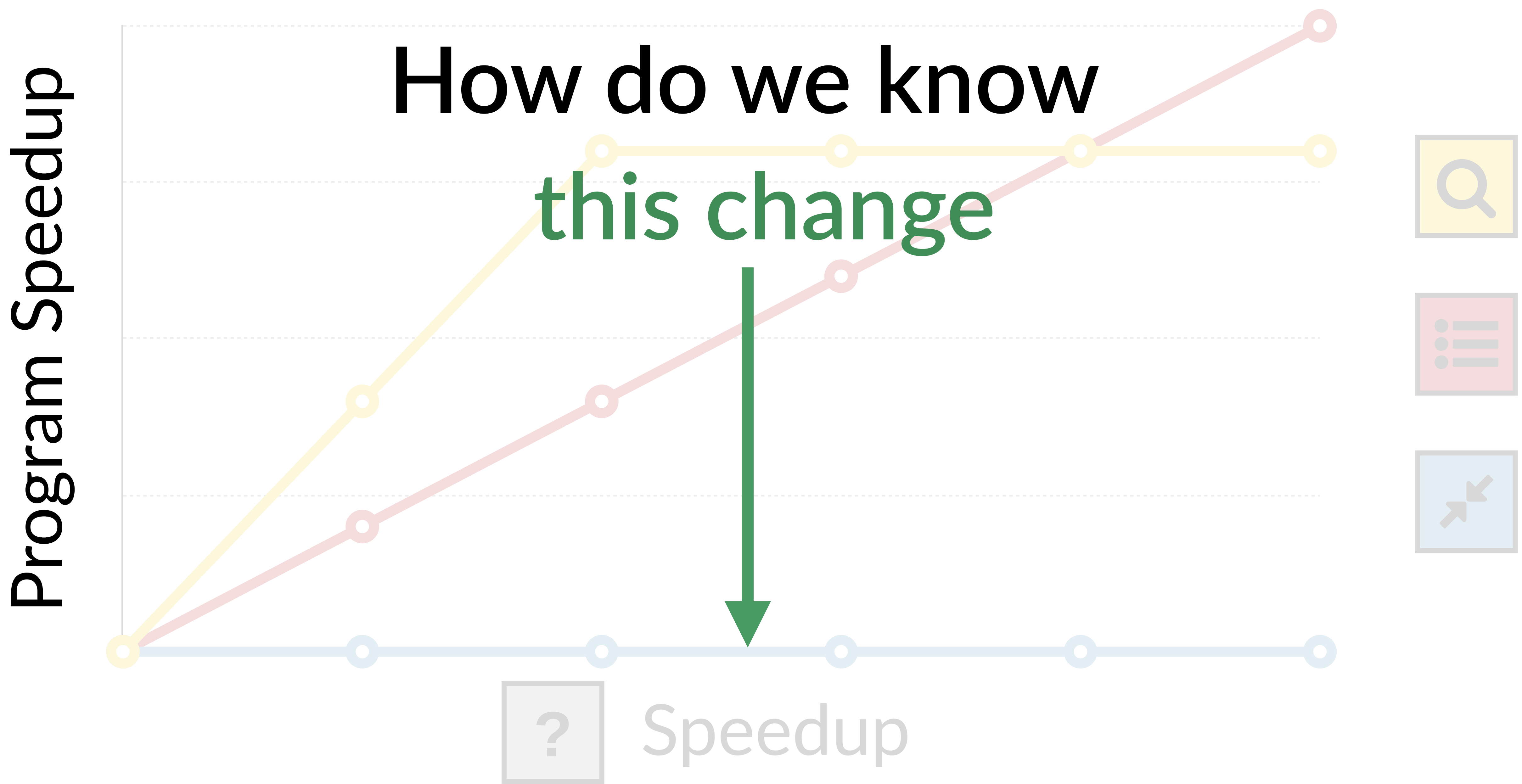
The program will run this much faster

Causal Profile

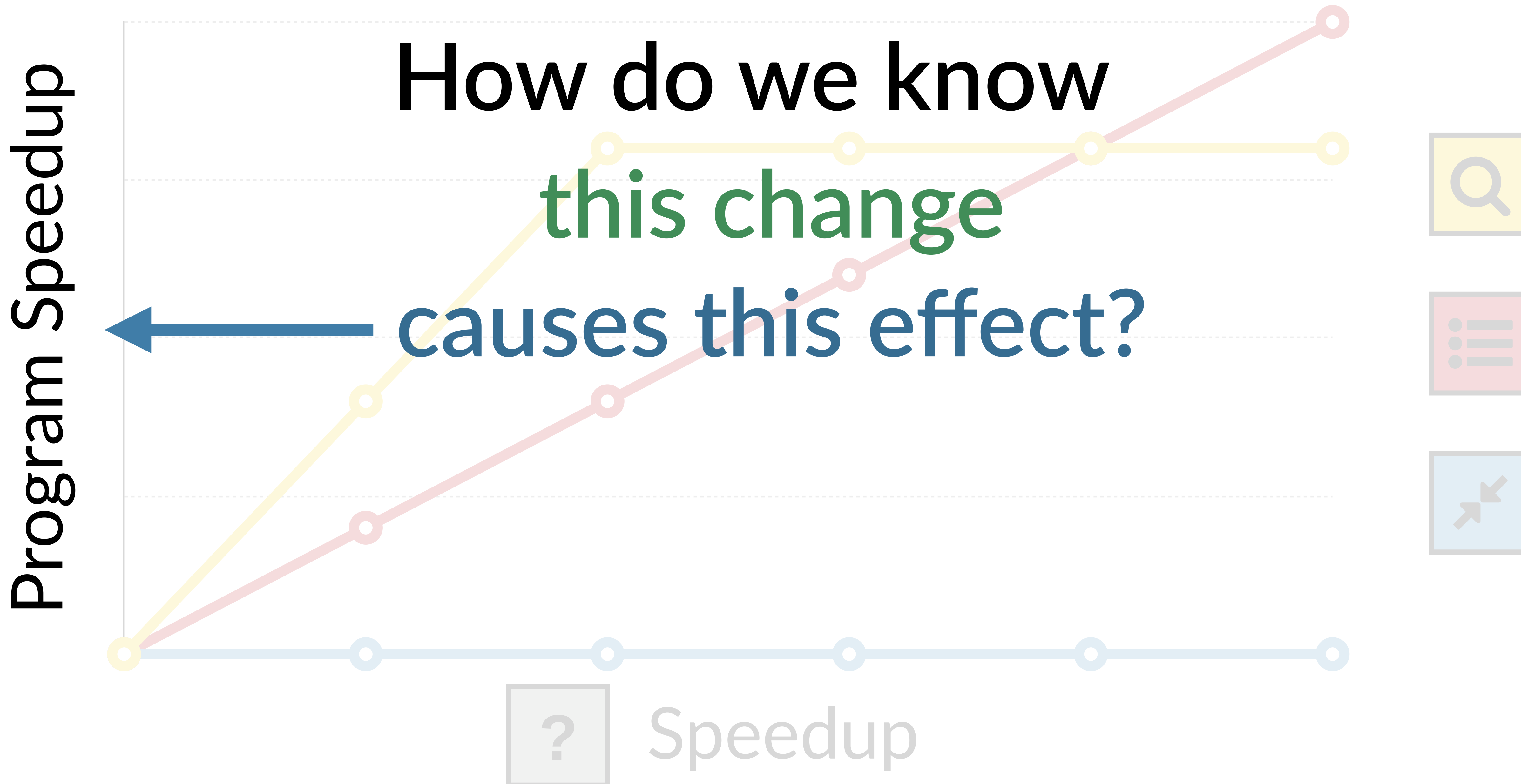
How do we know



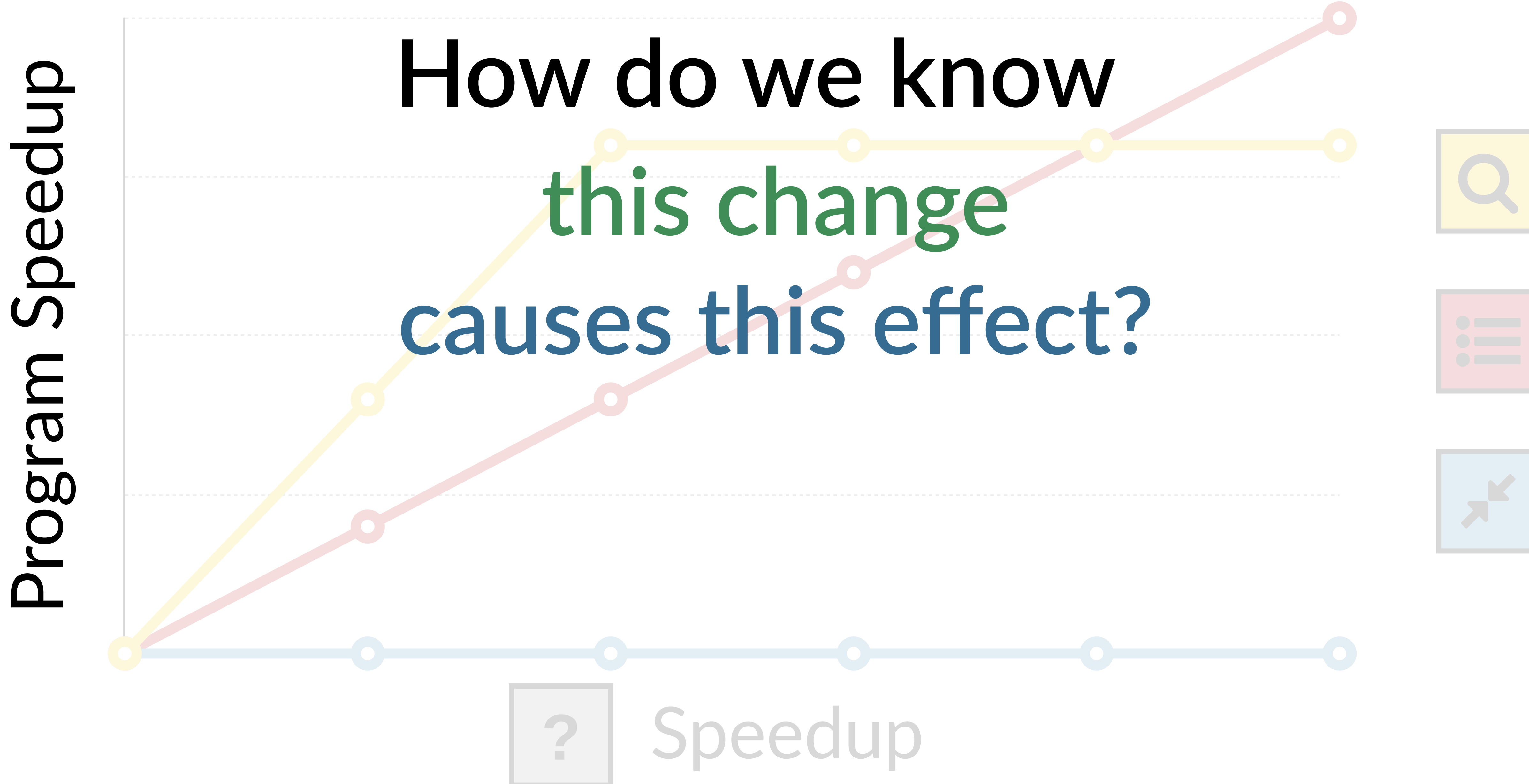
Causal Profile



Causal Profile



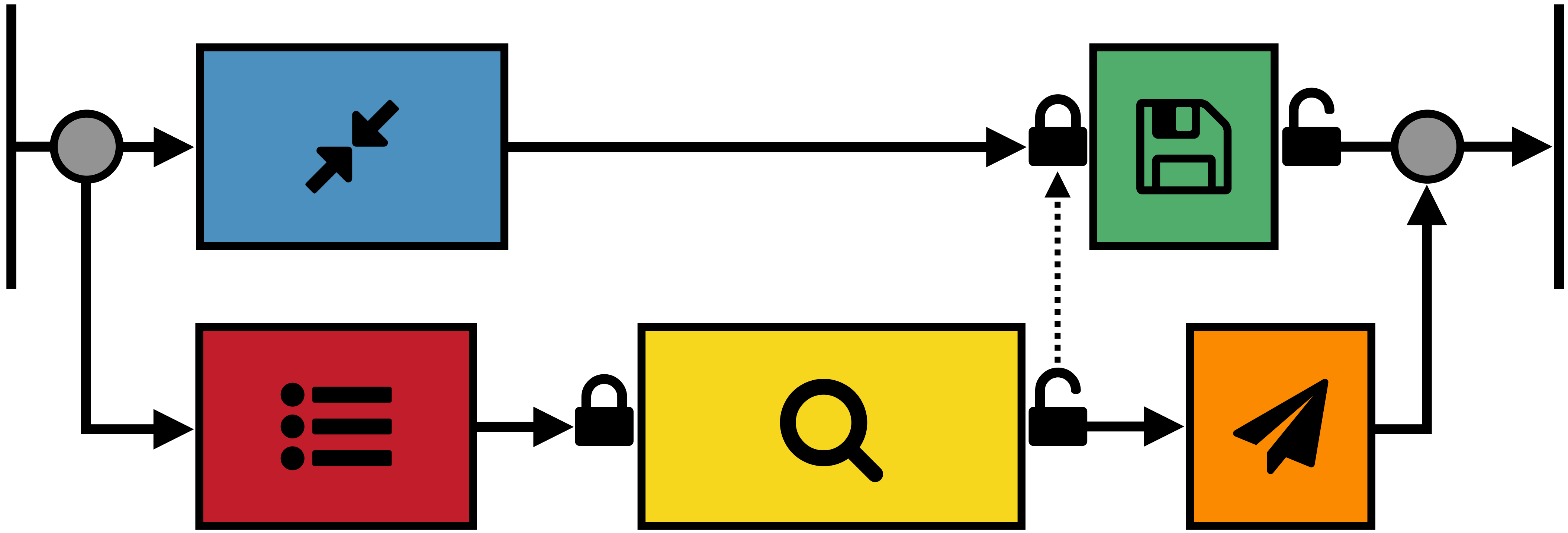
Causal Profile



How do we know
this change
causes this effect?

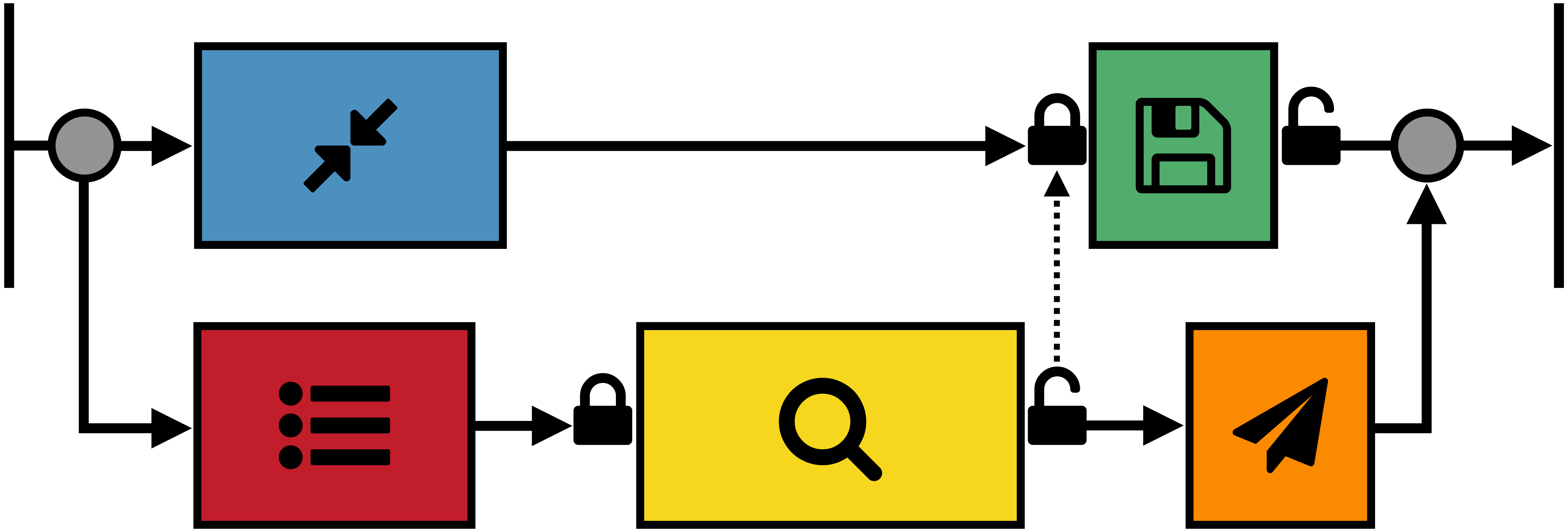
Run an experiment

Performance Experiments



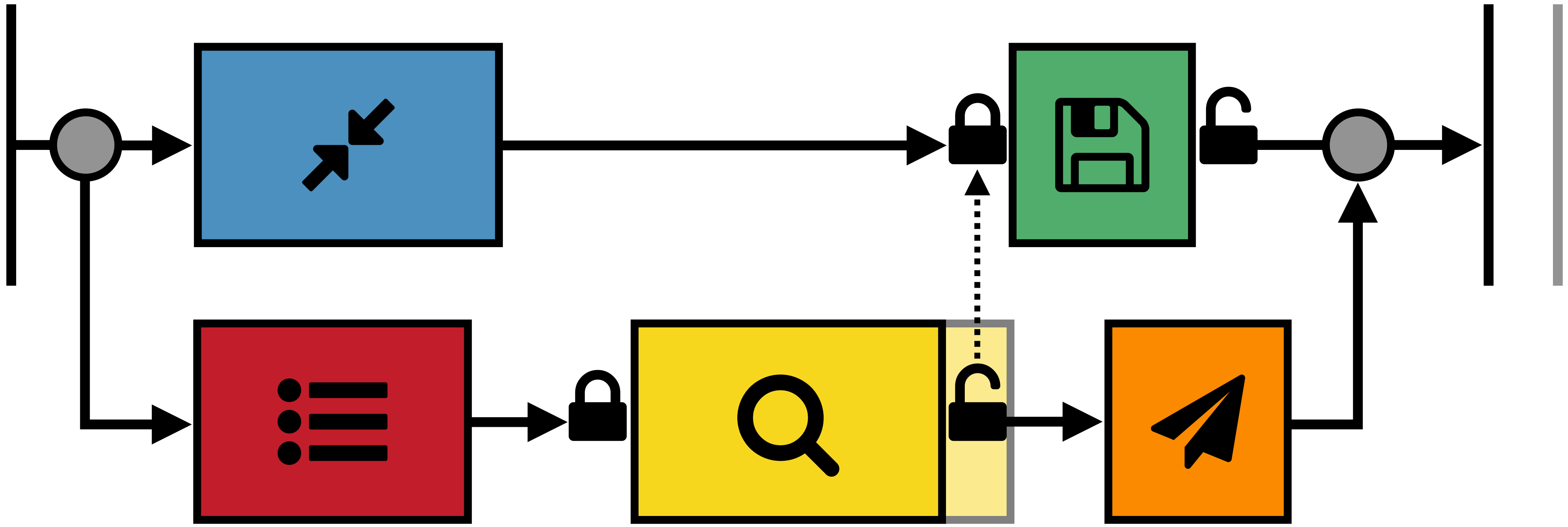
Performance Experiments

If we could magically speed up  ...



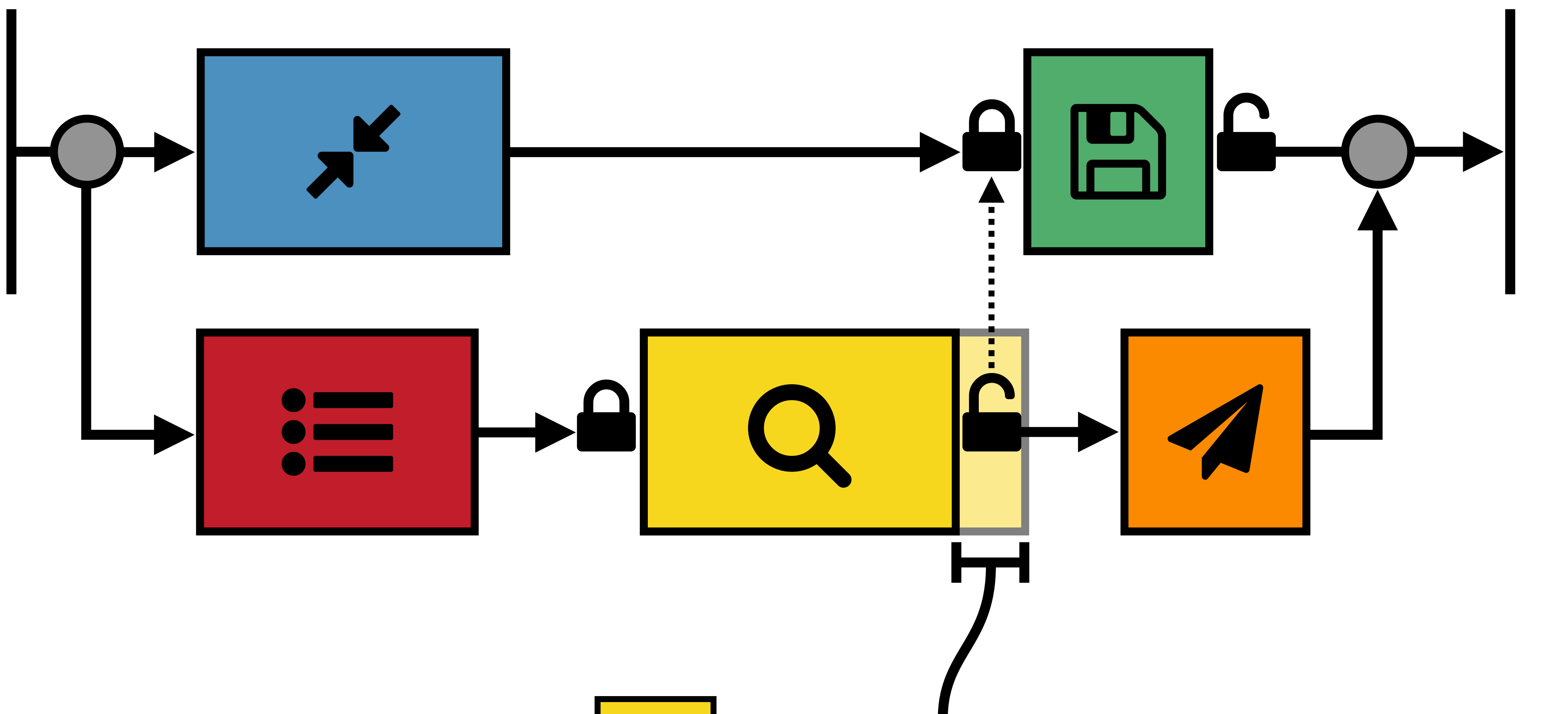
Performance Experiments

If we could magically speed up  ...



Performance Experiments

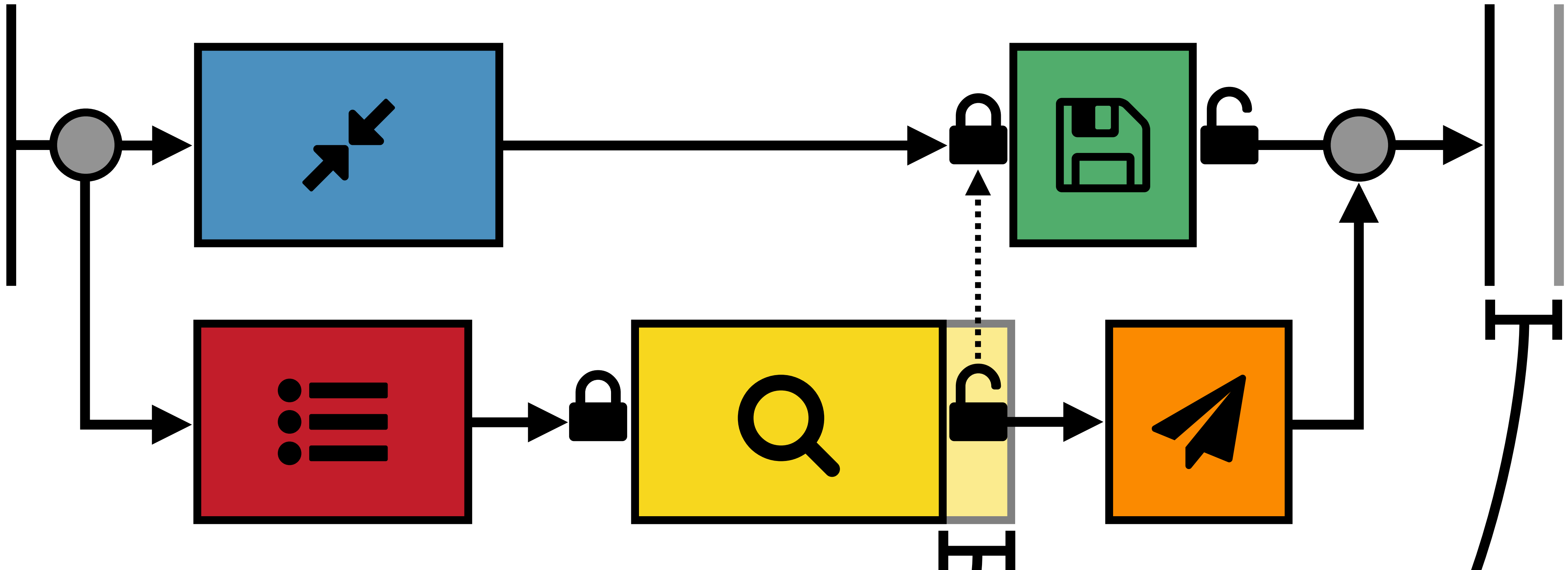
If we could magically speed up …



Speeding up by this much…

Performance Experiments

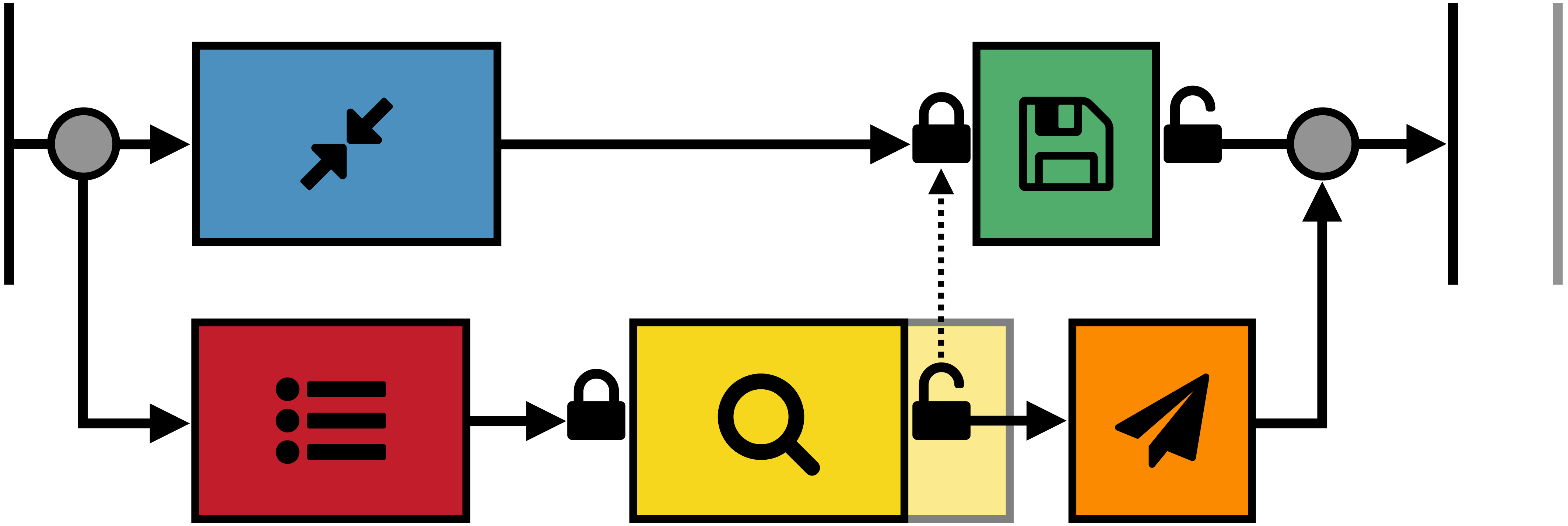
If we could magically speed up  ...



Speeding up  by this much...
speeds up the program by this much.

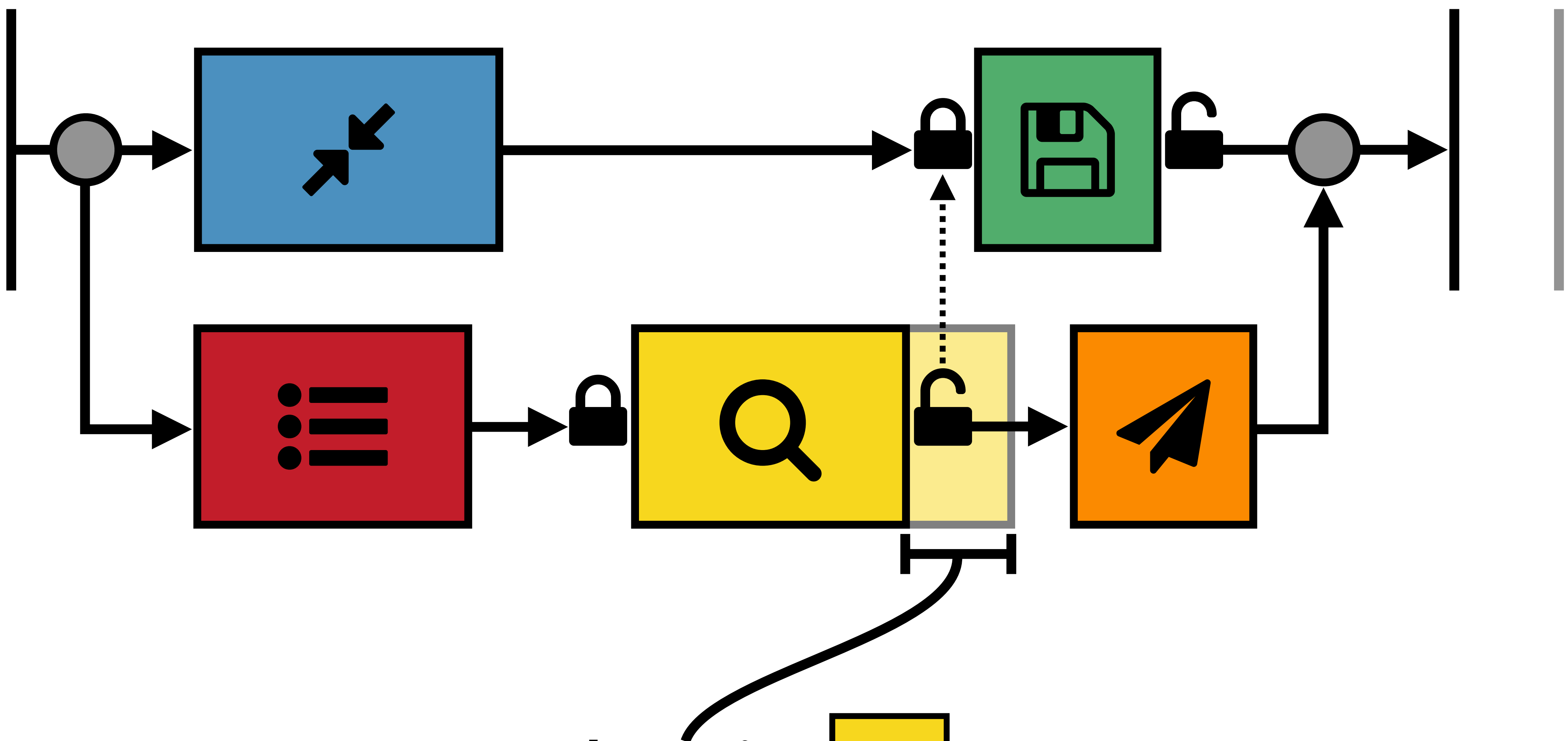
Performance Experiments

If we could magically speed up  ...



Performance Experiments

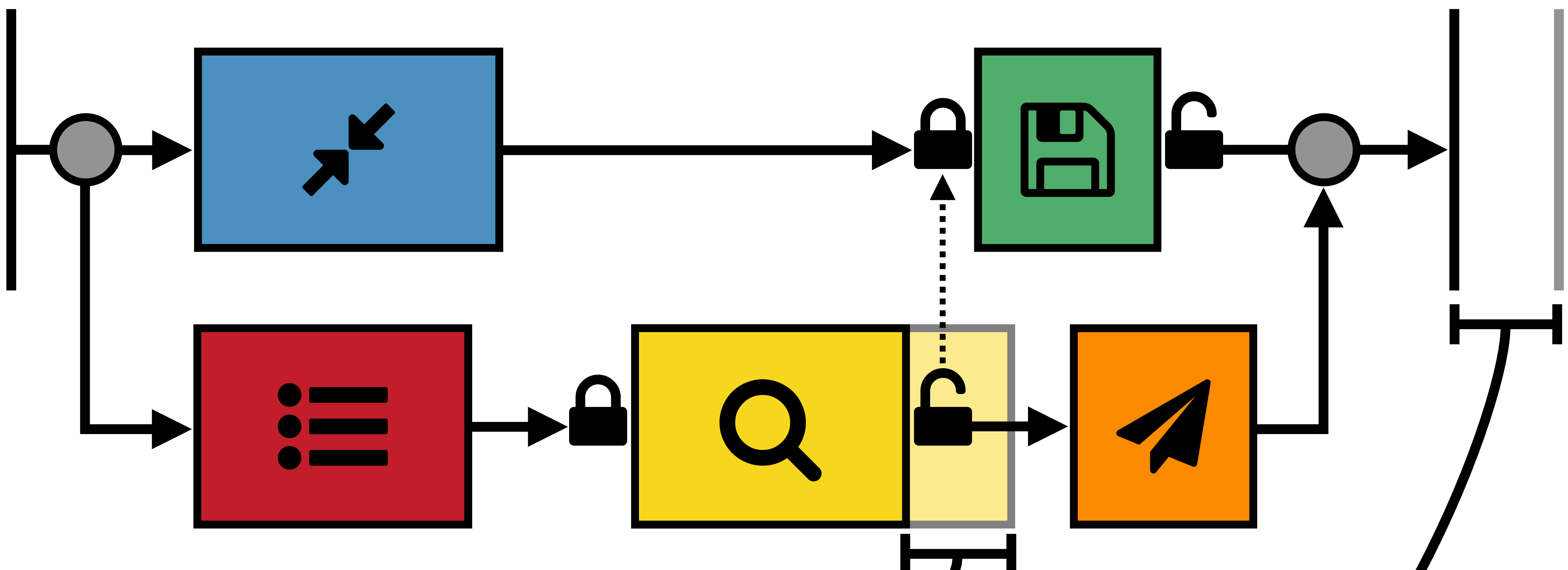
If we could magically speed up …



More speedup in …

Performance Experiments

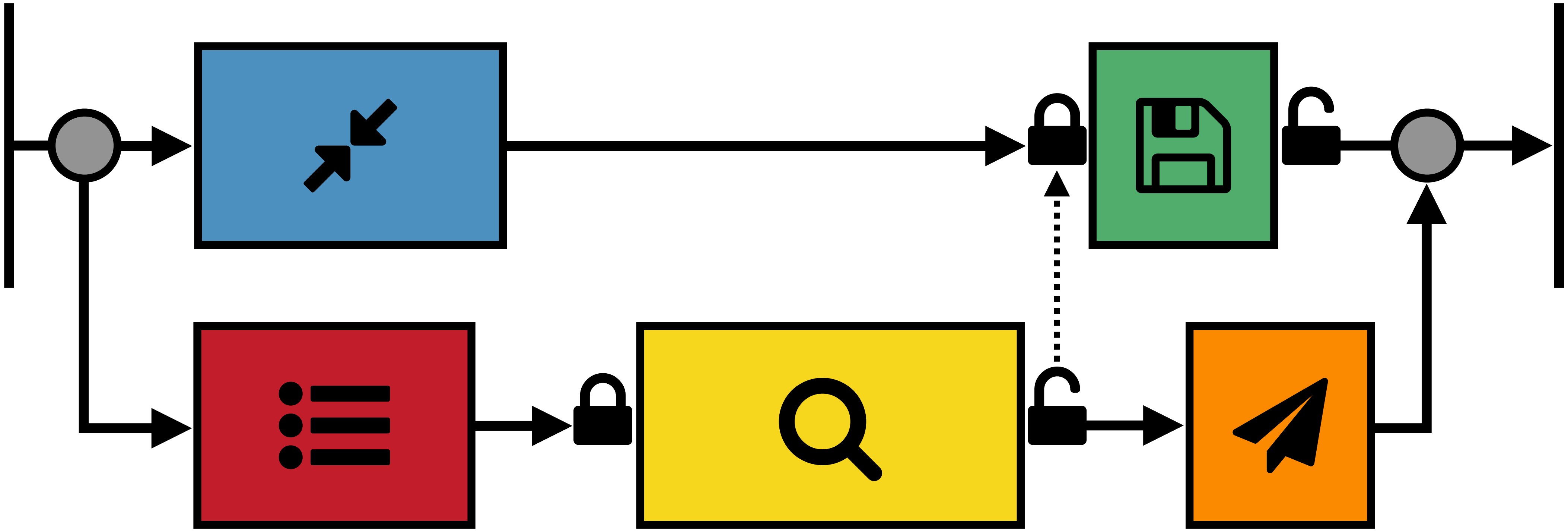
If we could magically speed up …



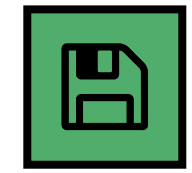
More speedup in …
leads to a larger program speedup.

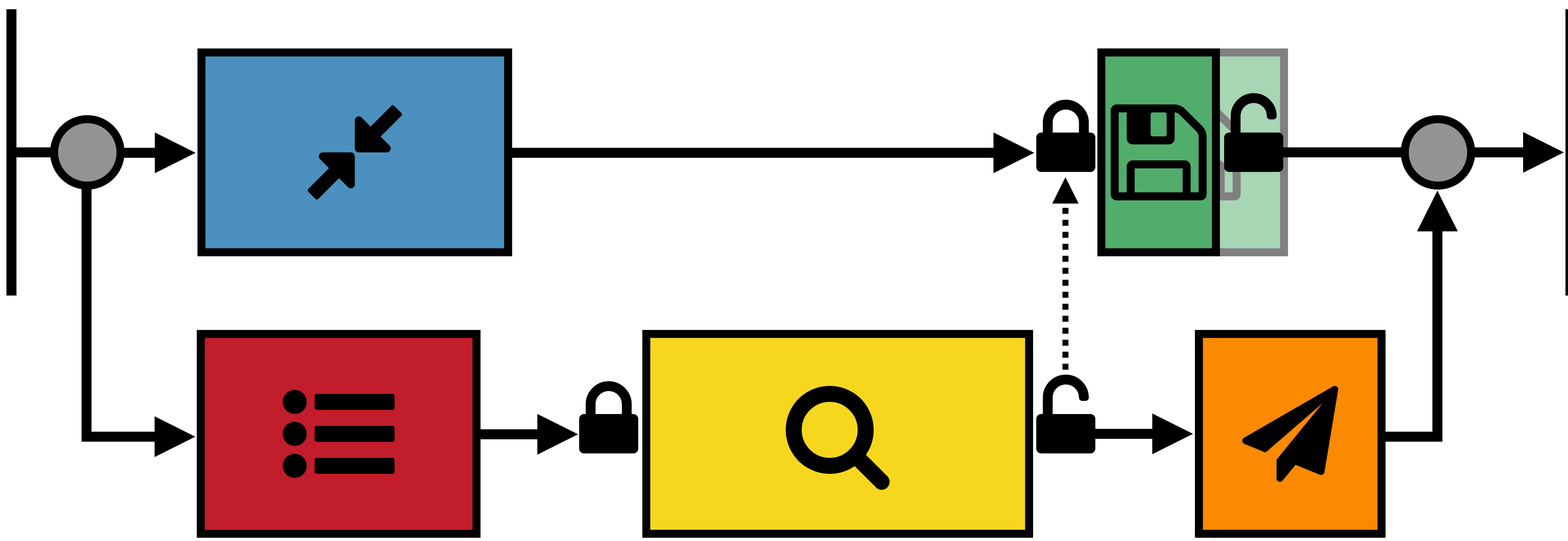
Performance Experiments

If we could magically speed up  ...



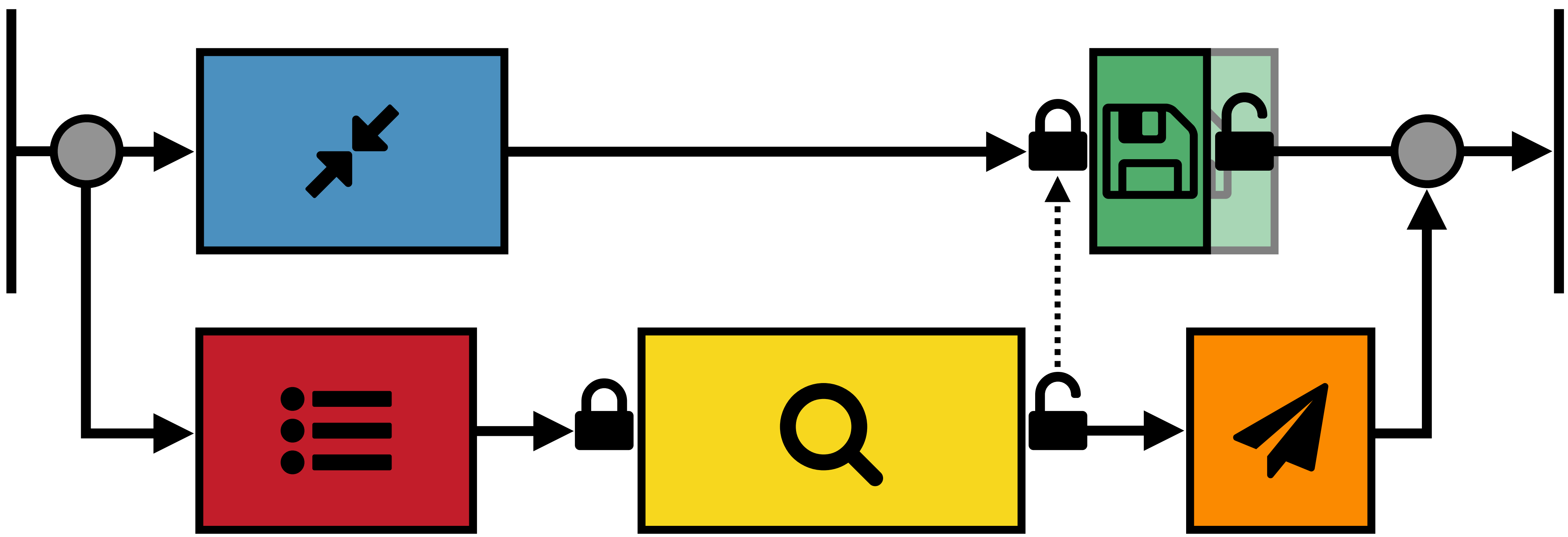
Performance Experiments

If we could magically speed up  ...



Performance Experiments

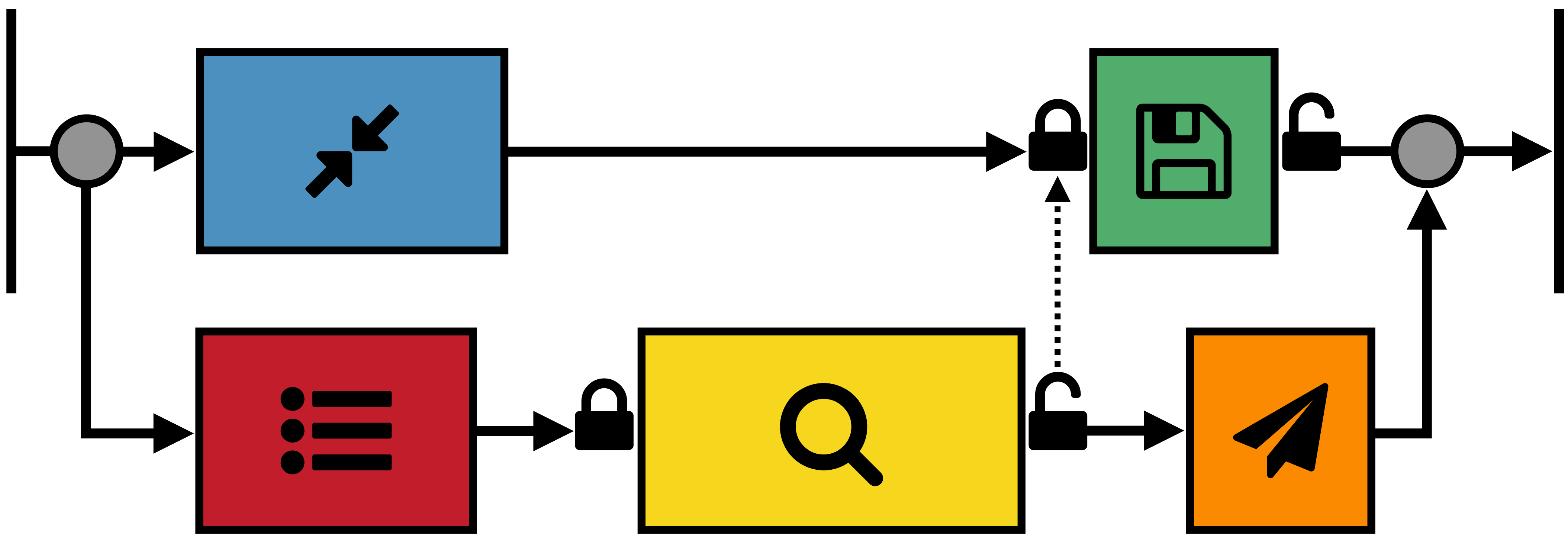
If we could magically speed up …



No program speedup

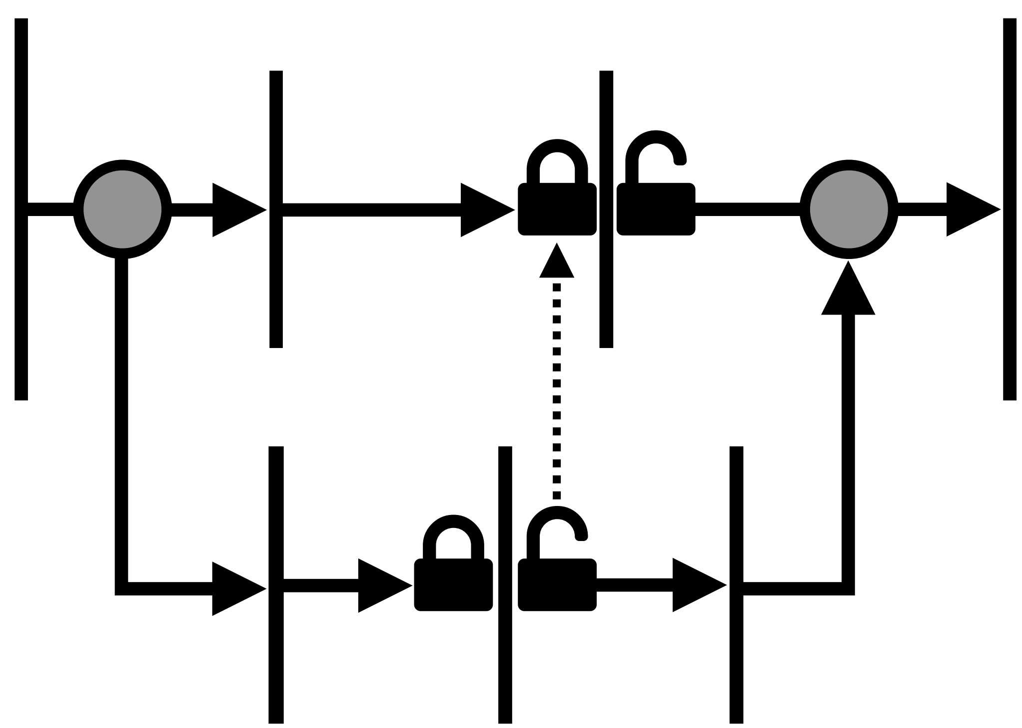
Performance Experiments

We're going to have to do this without magic.



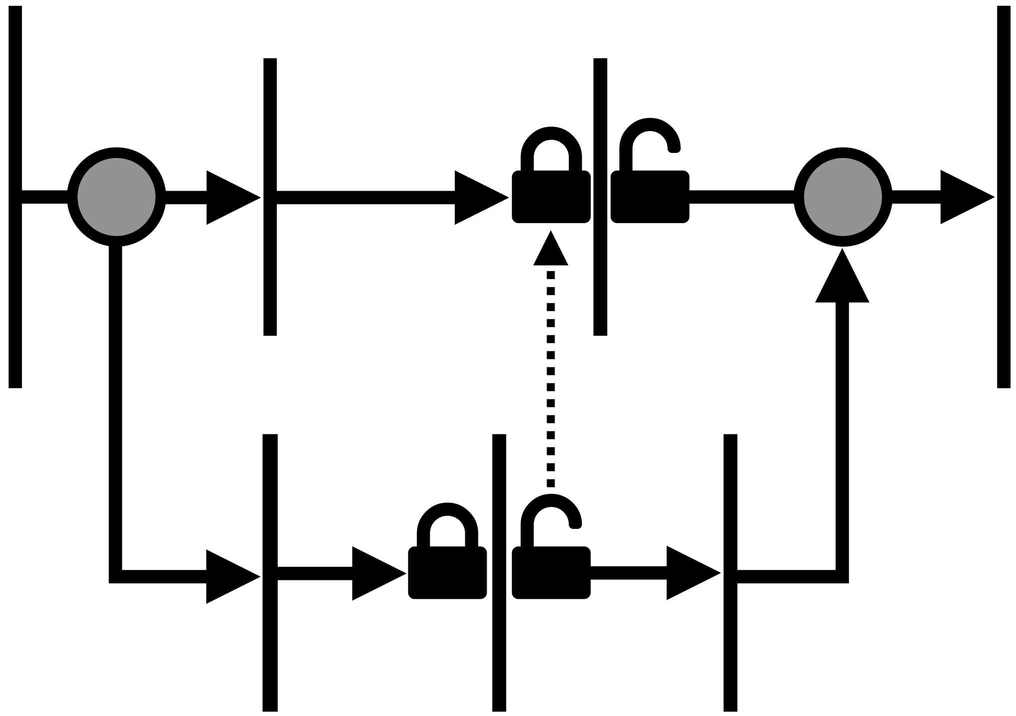
Performance Experiments

We're going to have to do this *without magic*...



Performance Experiments

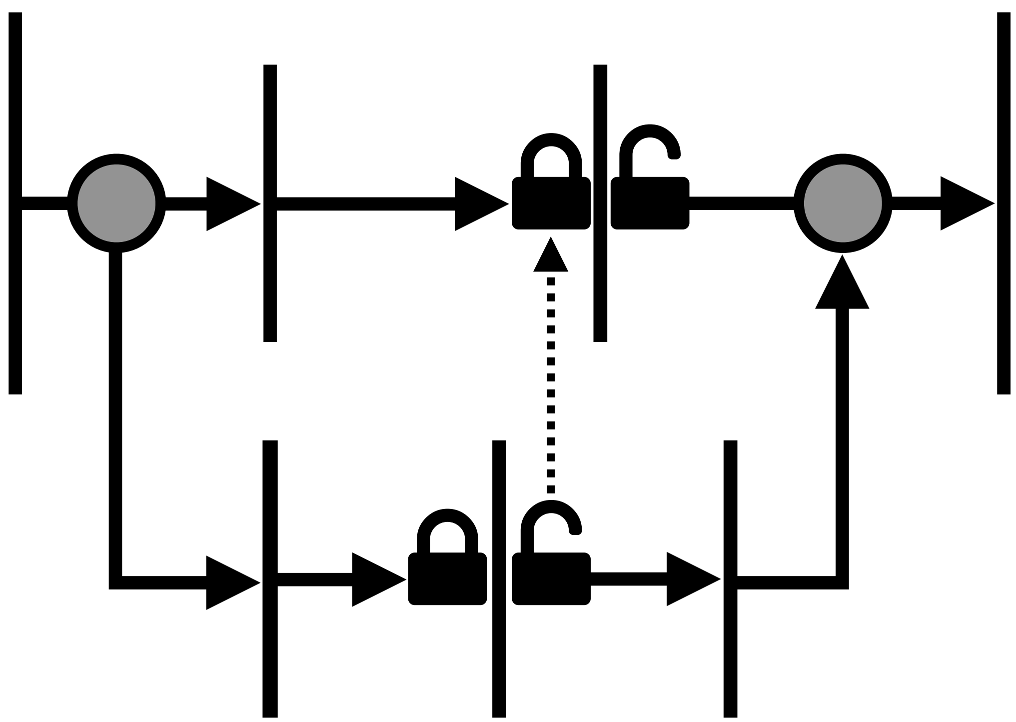
We're going to have to do this *without magic*...



Otherwise, we'd just do this...

Performance Experiments

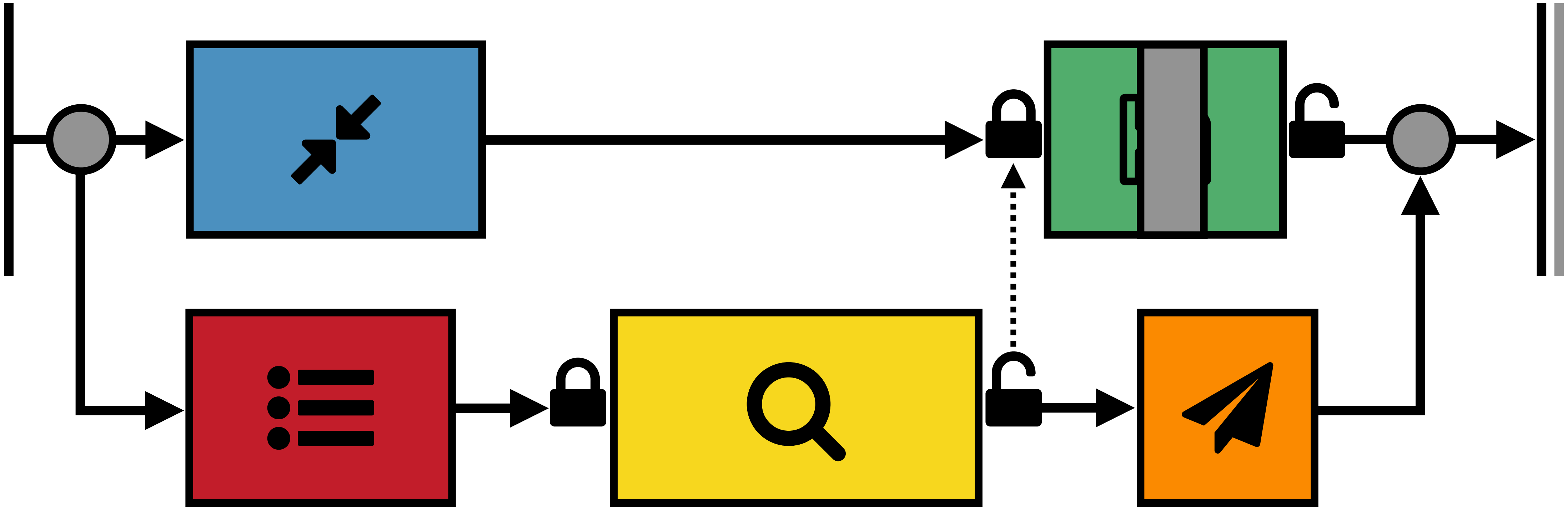
We're going to have to do this *without magic*…



Otherwise, we'd just do this…

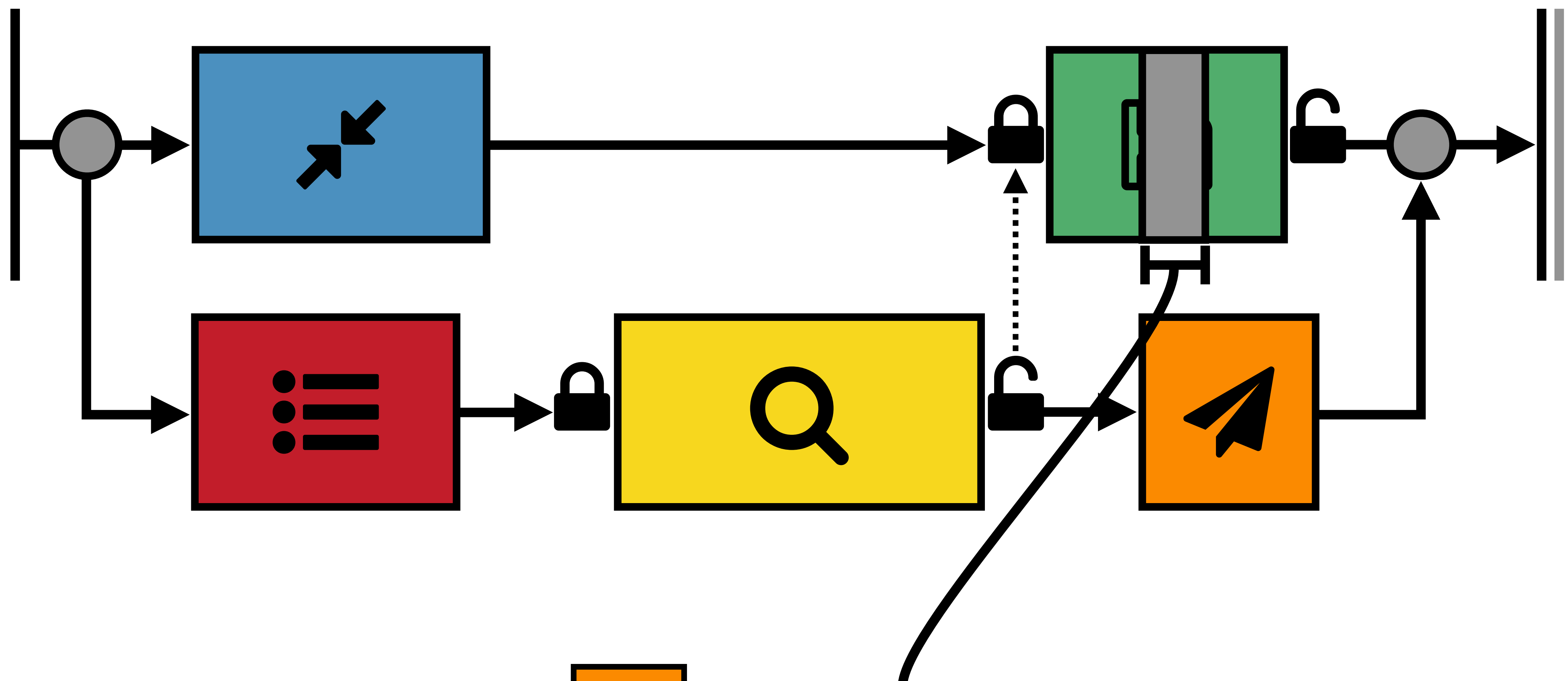
Virtual Speedup

"Speed up"  by slowing everything else down.



Virtual Speedup

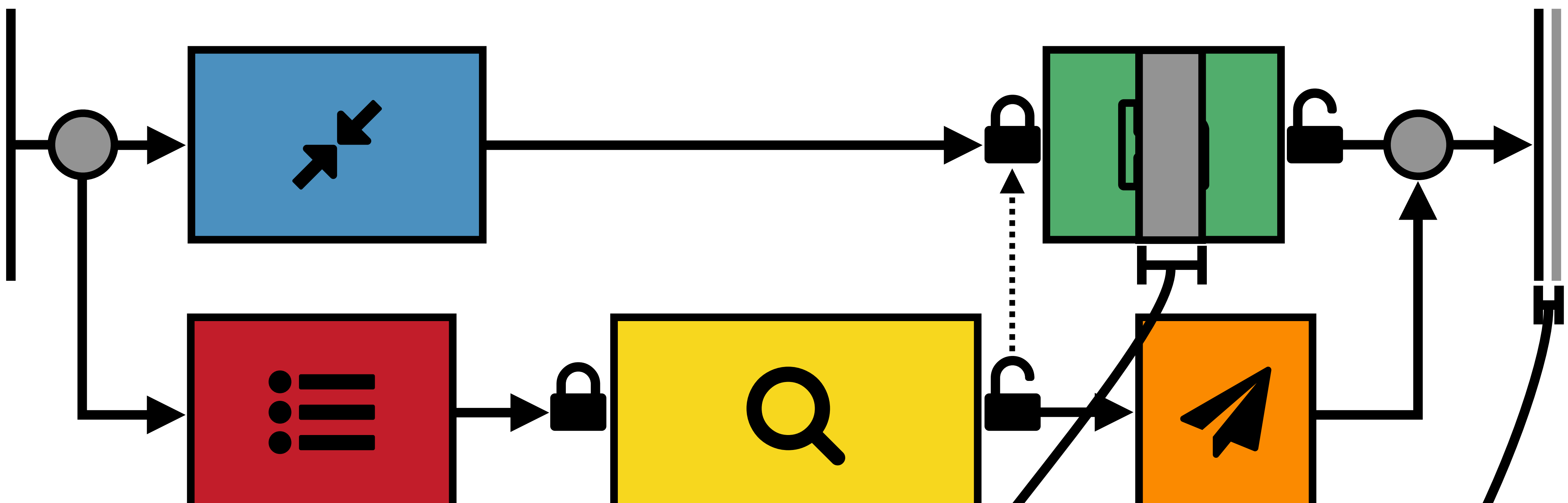
“Speed up” by slowing everything else down.



Speeding up by this much…

Virtual Speedup

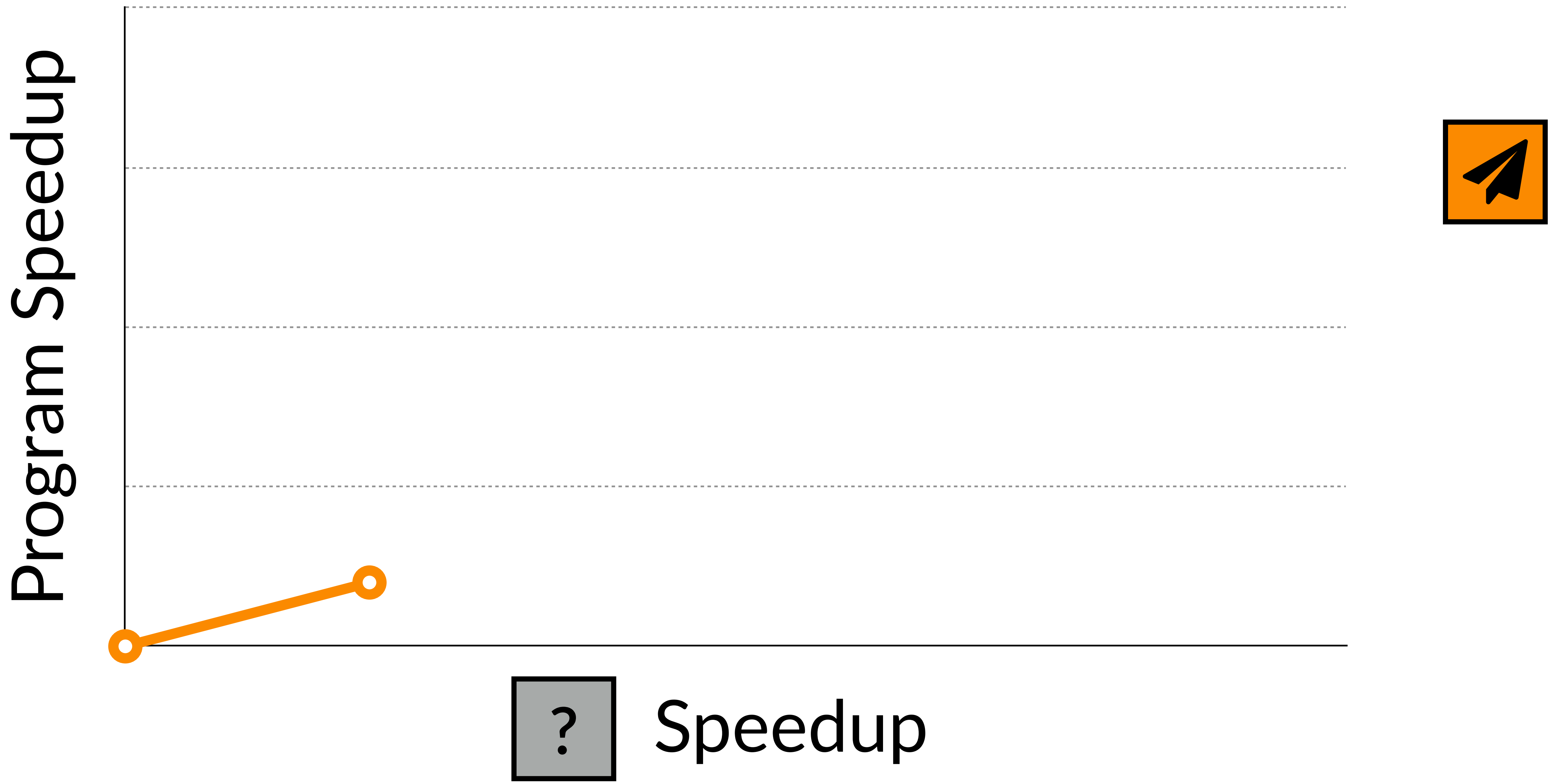
“Speed up”  by slowing everything else down.



Speeding up  by this much…
speeds up the program by this much.

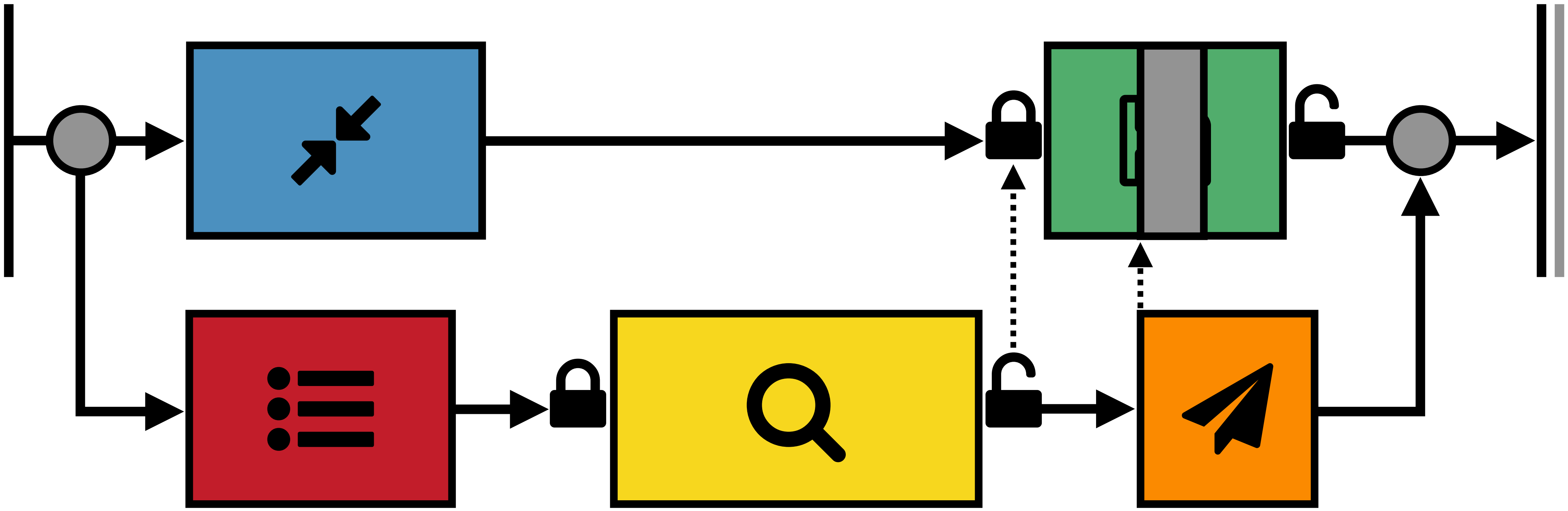
Speedup Results

Speedup Results



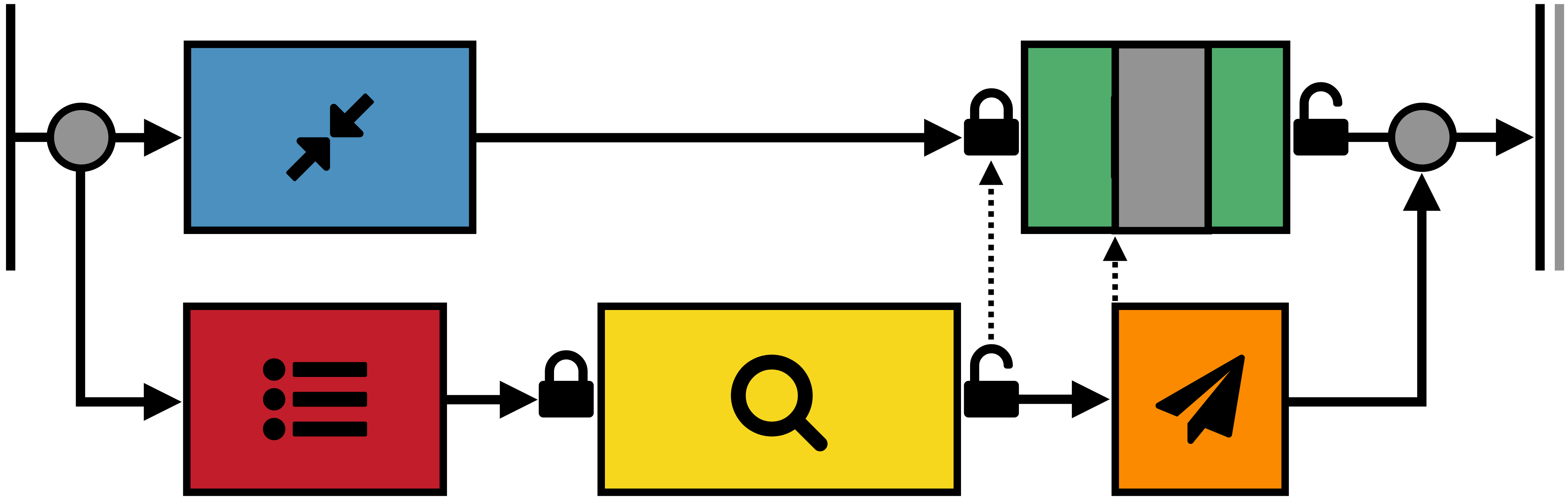
Virtual Speedup

“Speed up” by slowing everything else down.



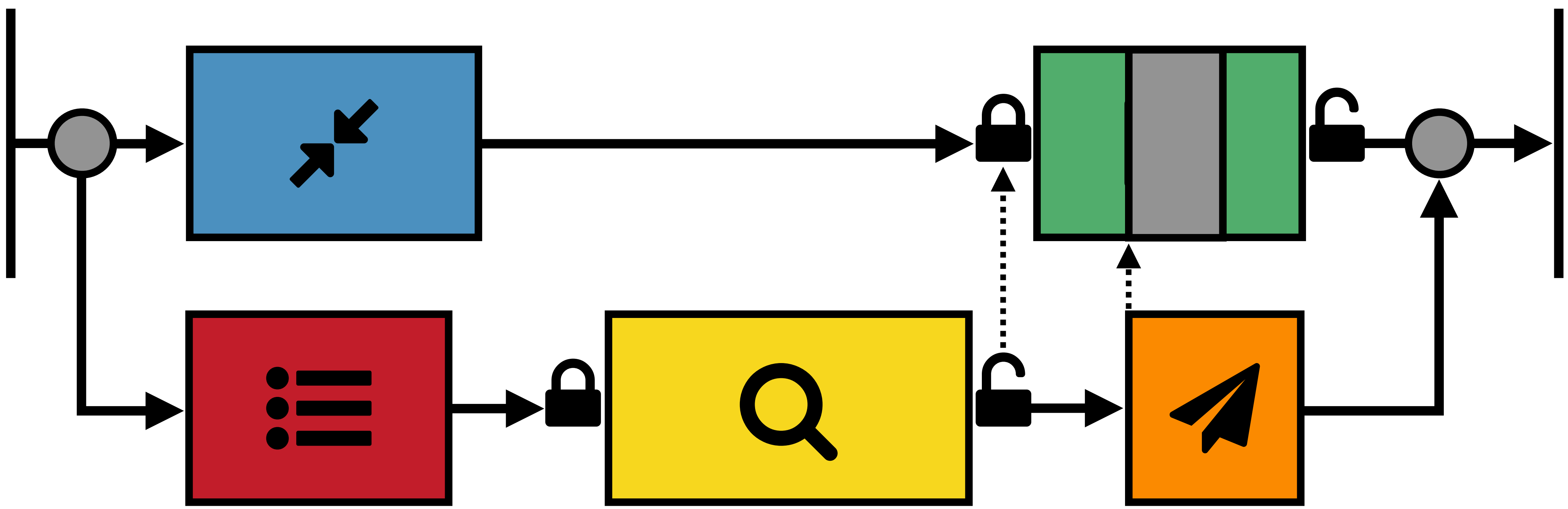
Virtual Speedup

“Speed up” by slowing everything else down.



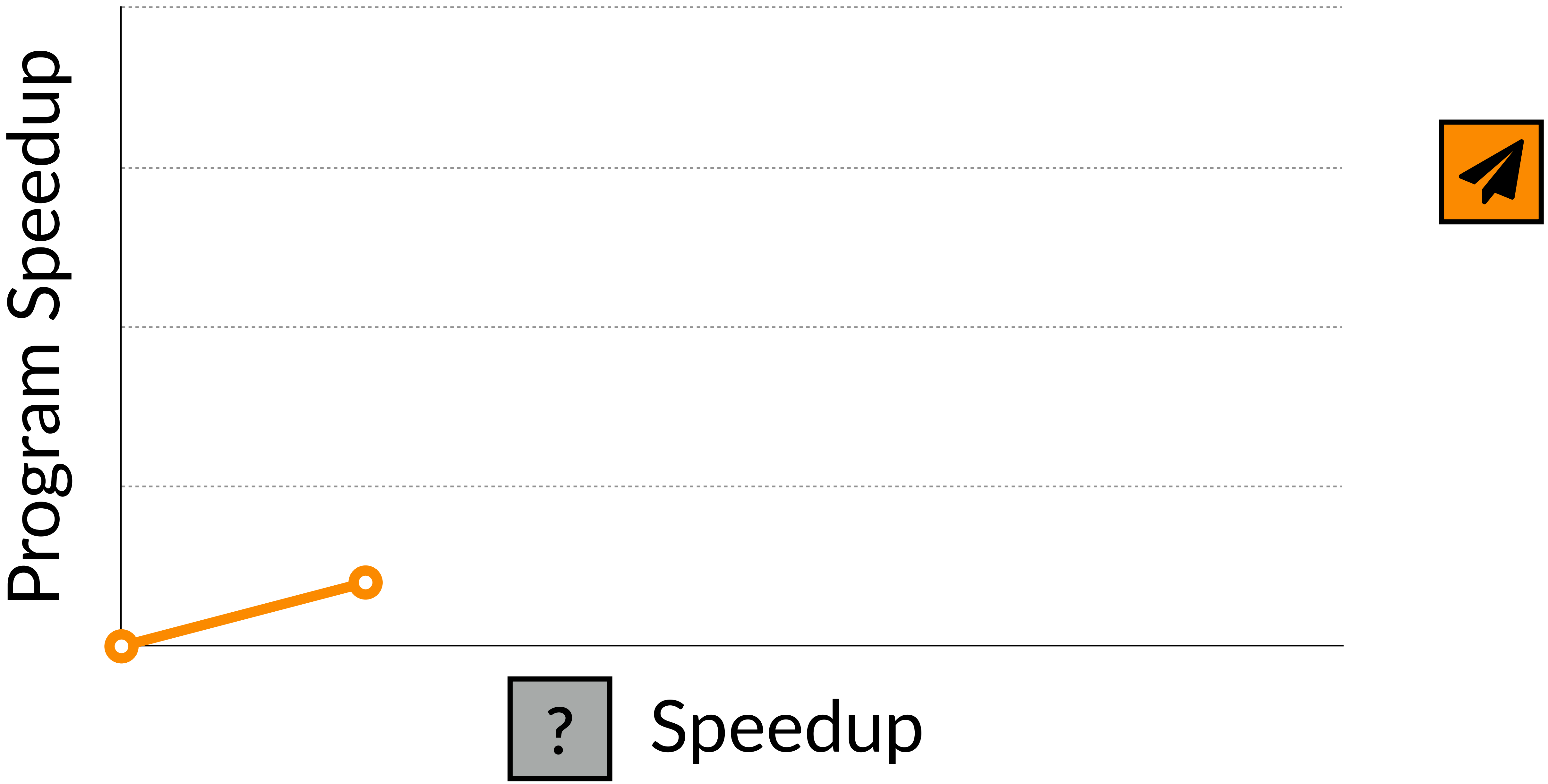
Virtual Speedup

“Speed up”  by slowing everything else down.

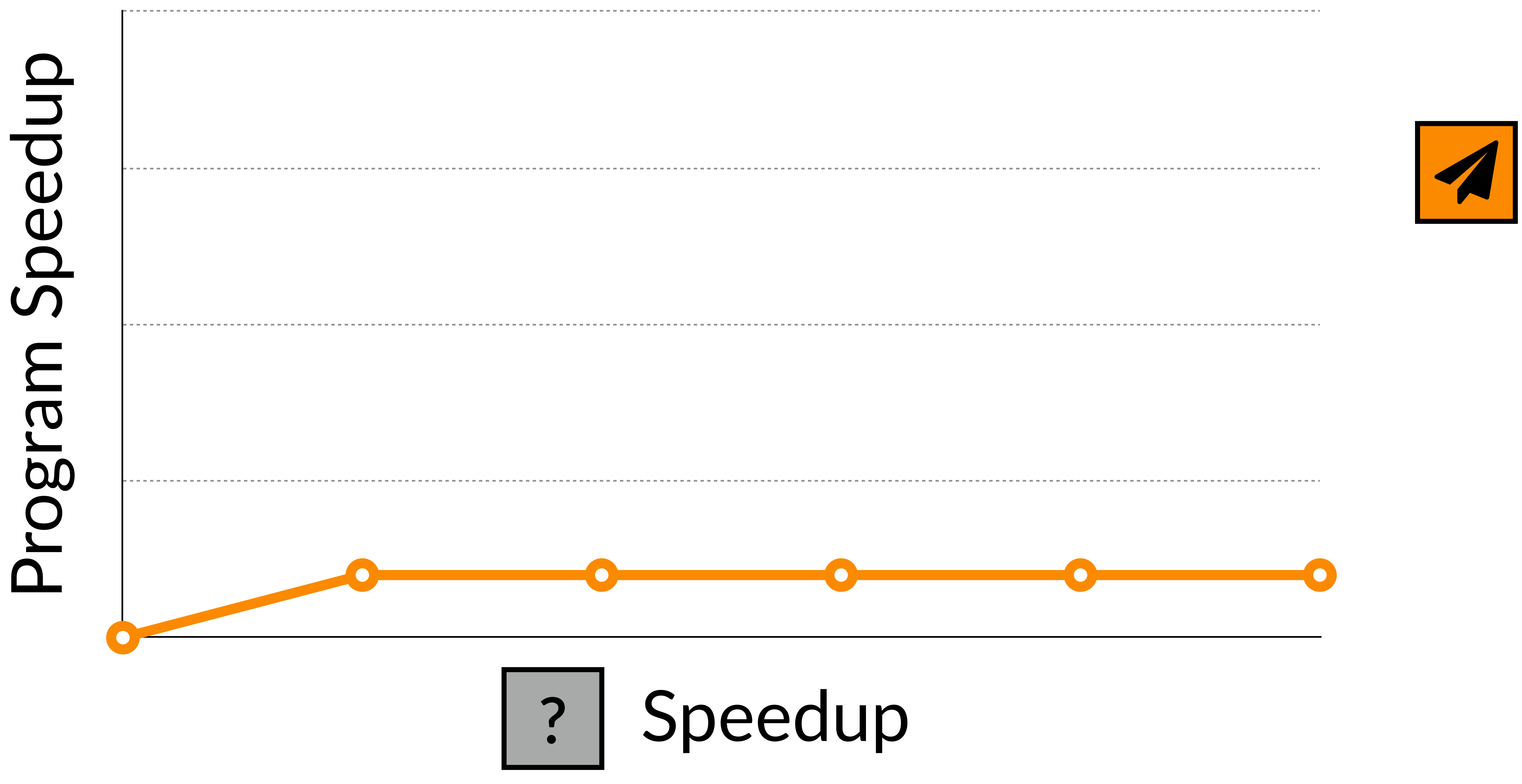


A larger speedup has no additional effect

Speedup Results

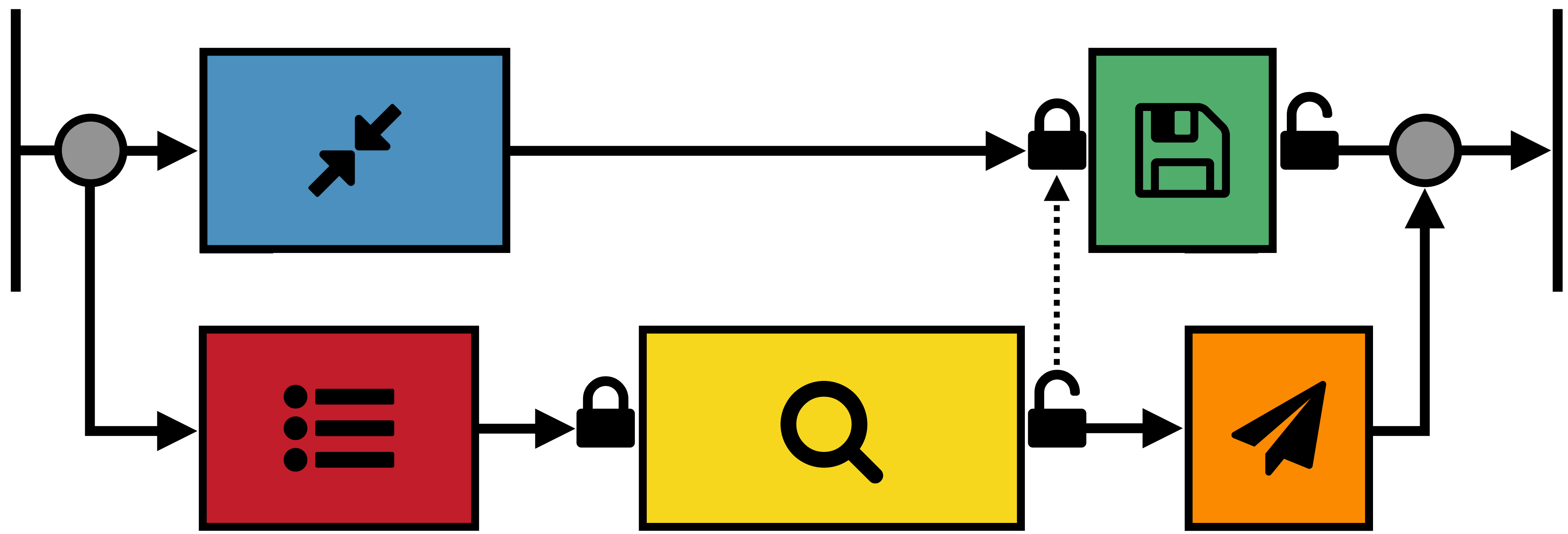


Speedup Results



Virtual Speedup

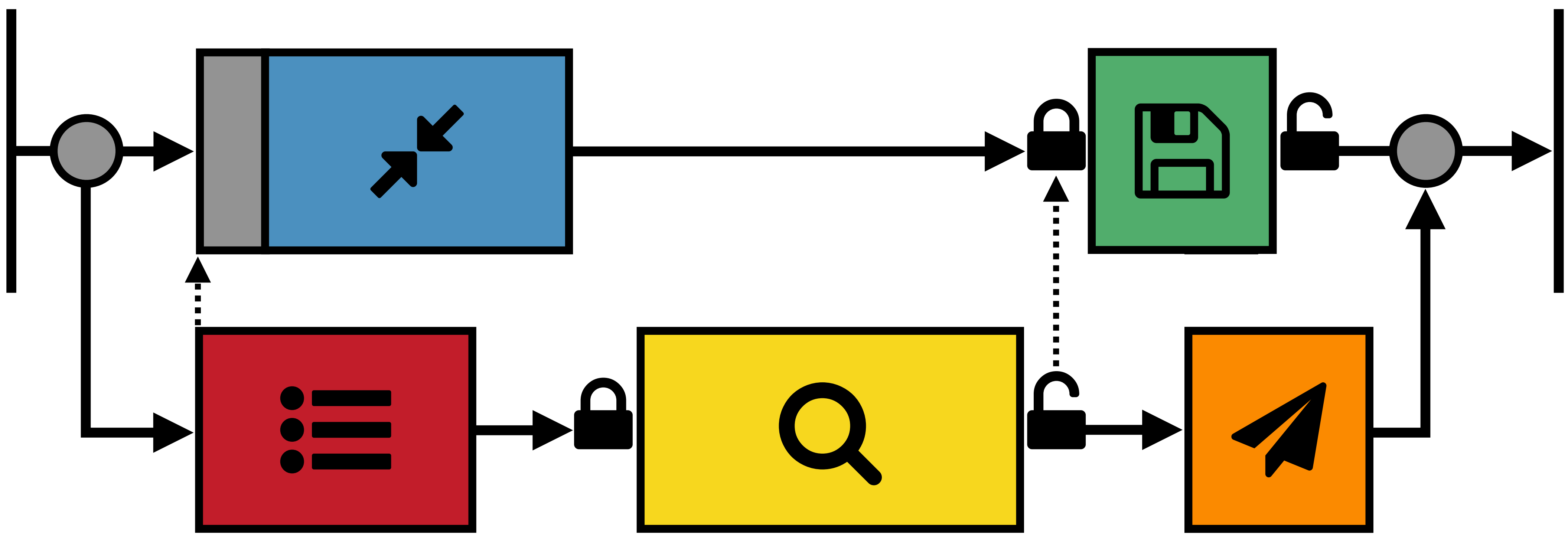
“Speed up” by slowing everything else down.



Each time runs, pause all other threads.

Virtual Speedup

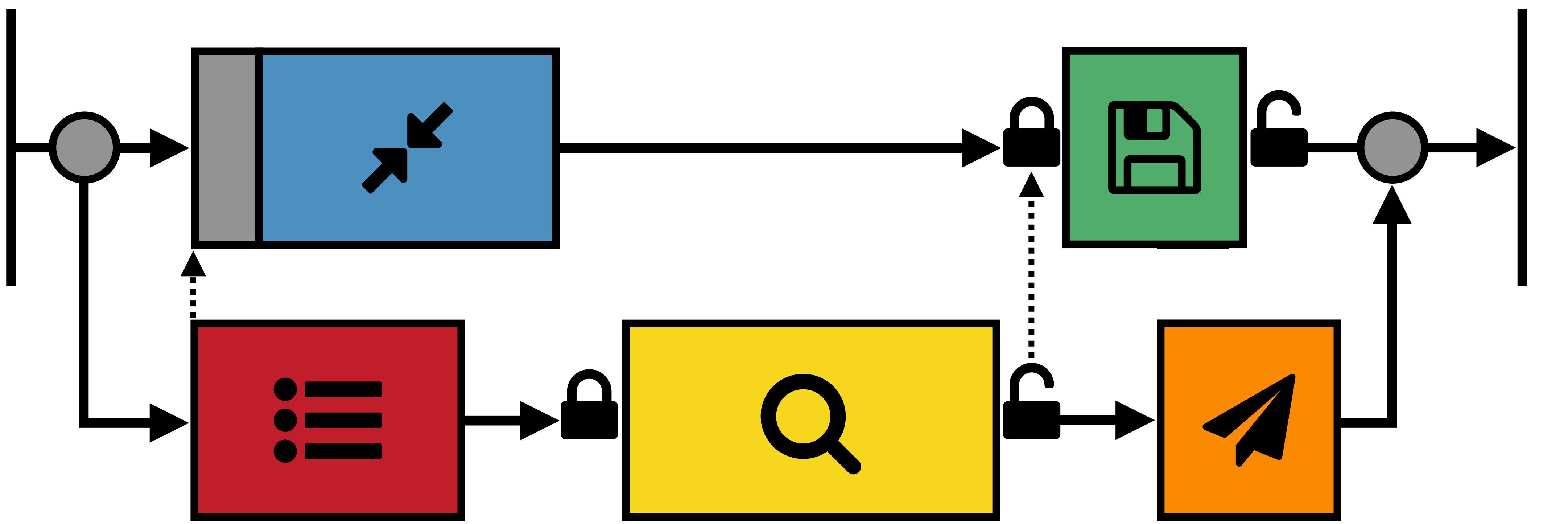
"Speed up" by slowing everything else down.



Each time runs, pause all other threads.

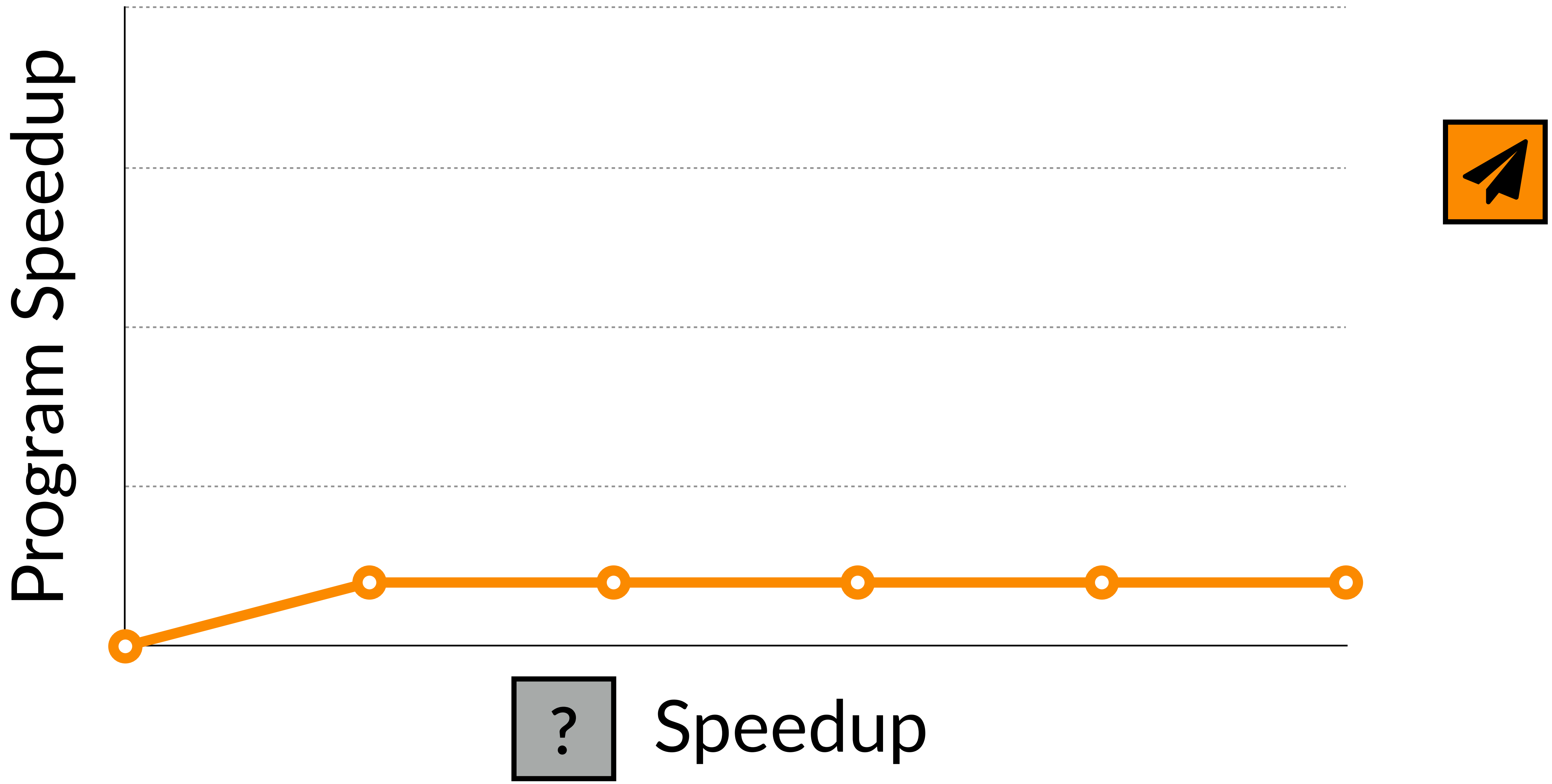
Virtual Speedup

"Speed up" by slowing everything else down.

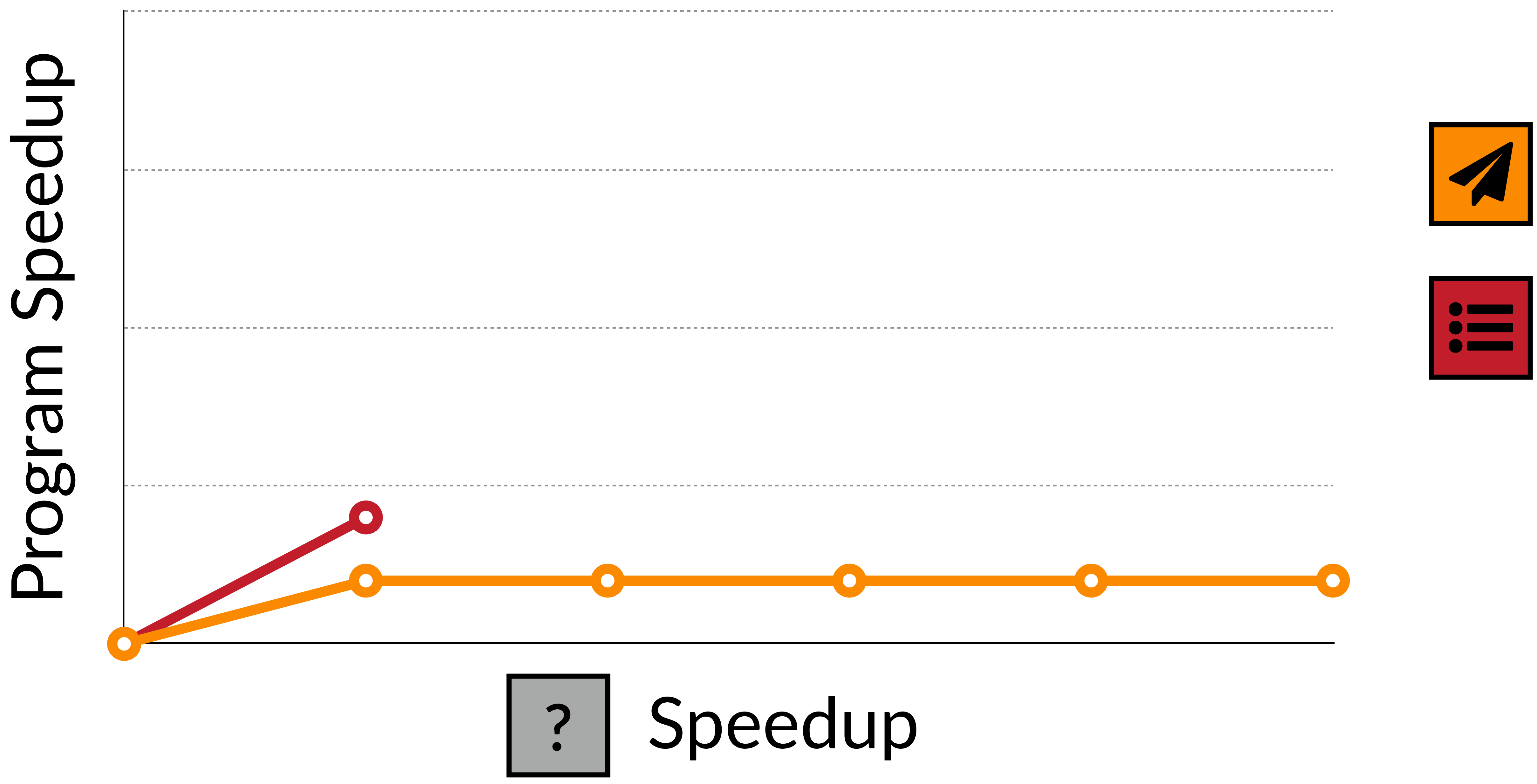


Each time runs, pause all other threads.

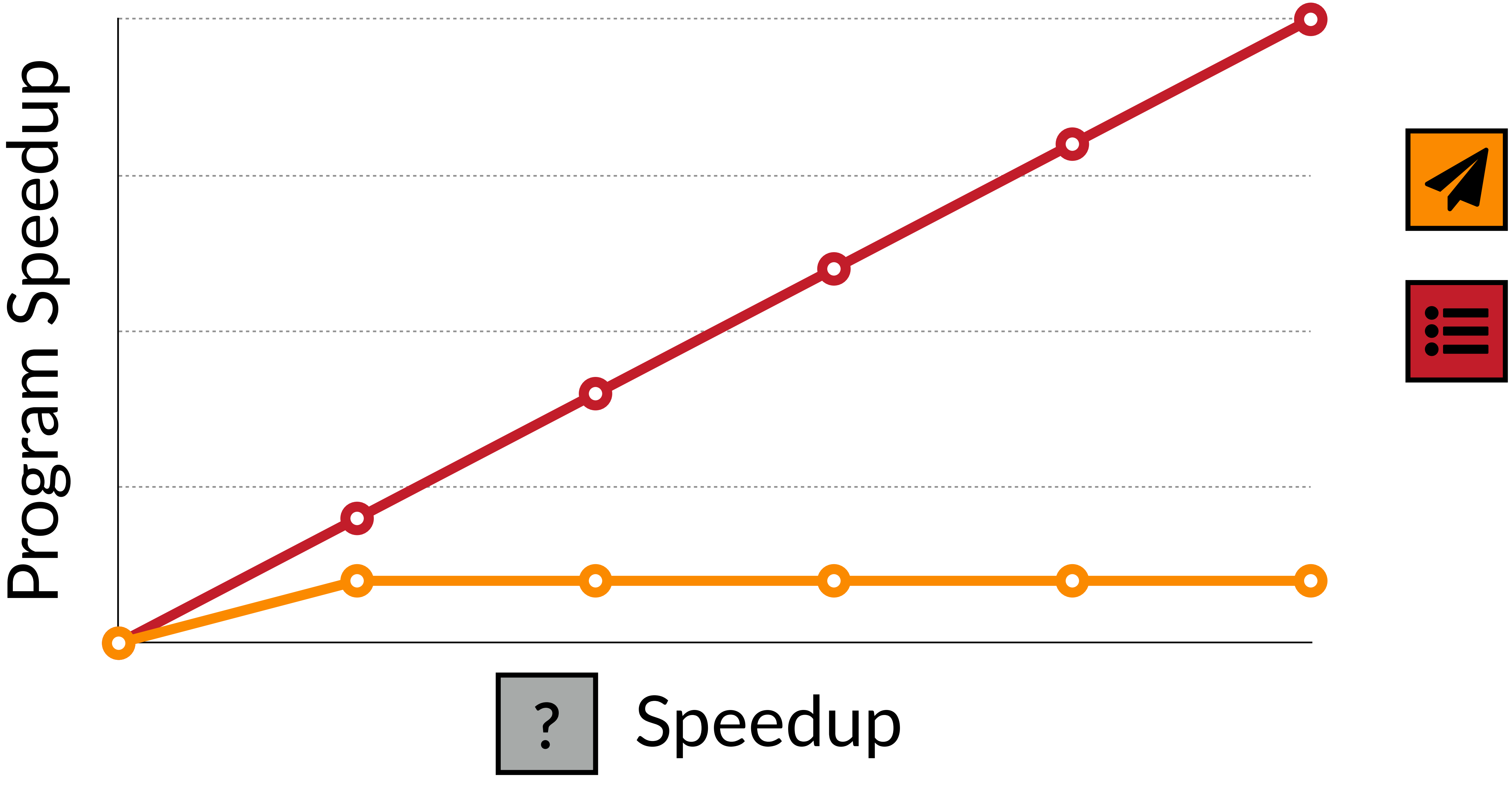
Speedup Results



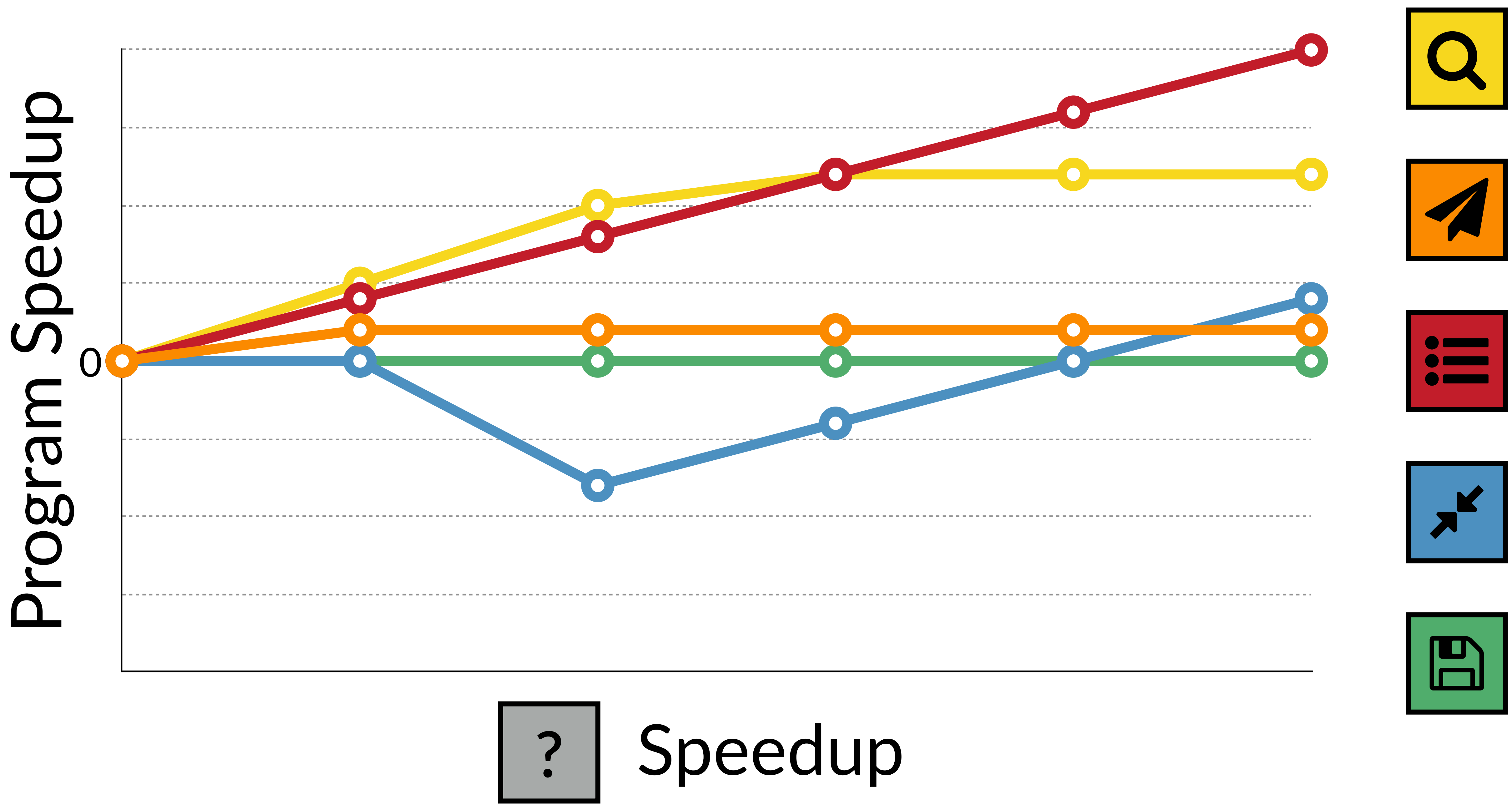
Speedup Results

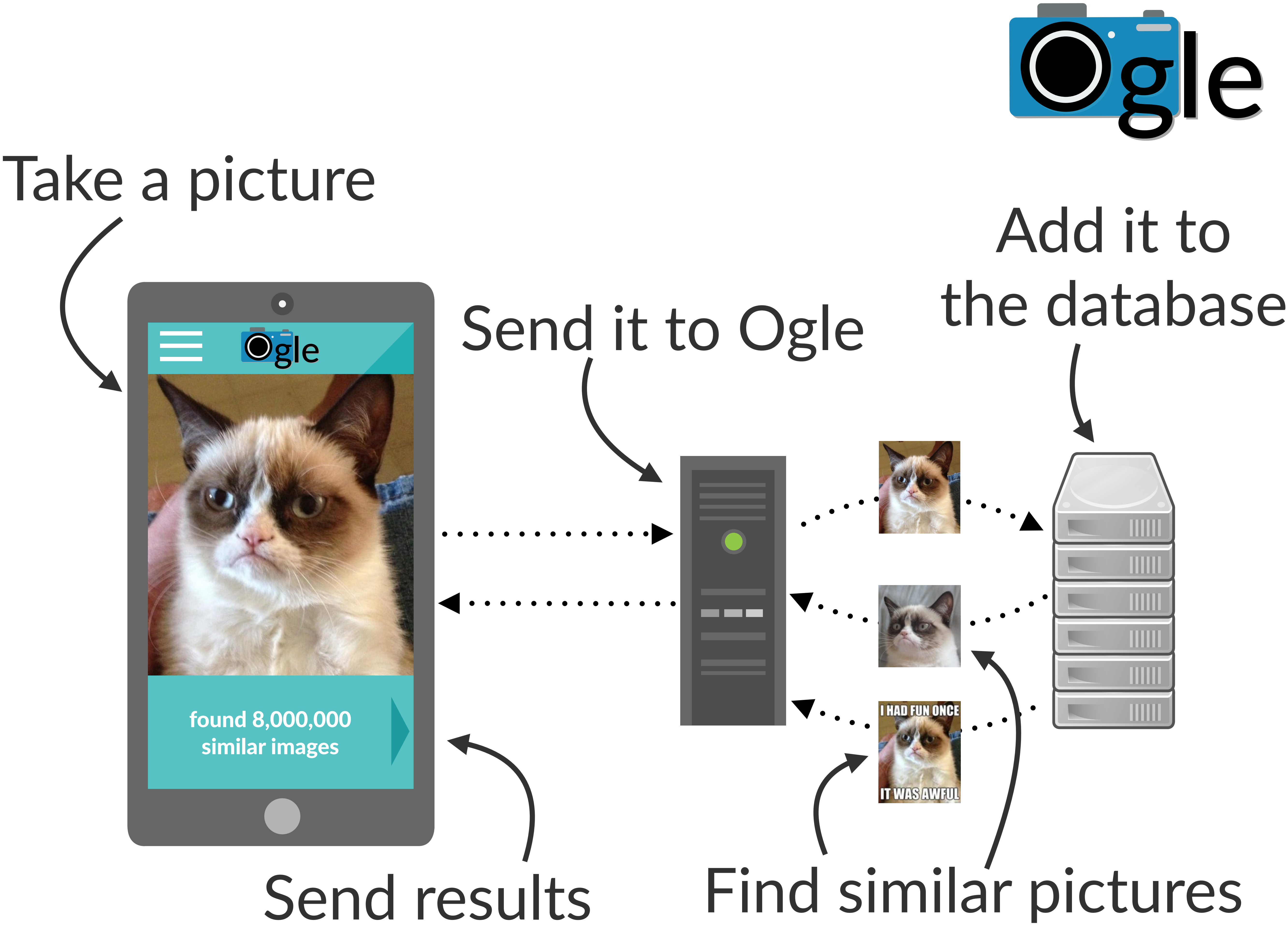


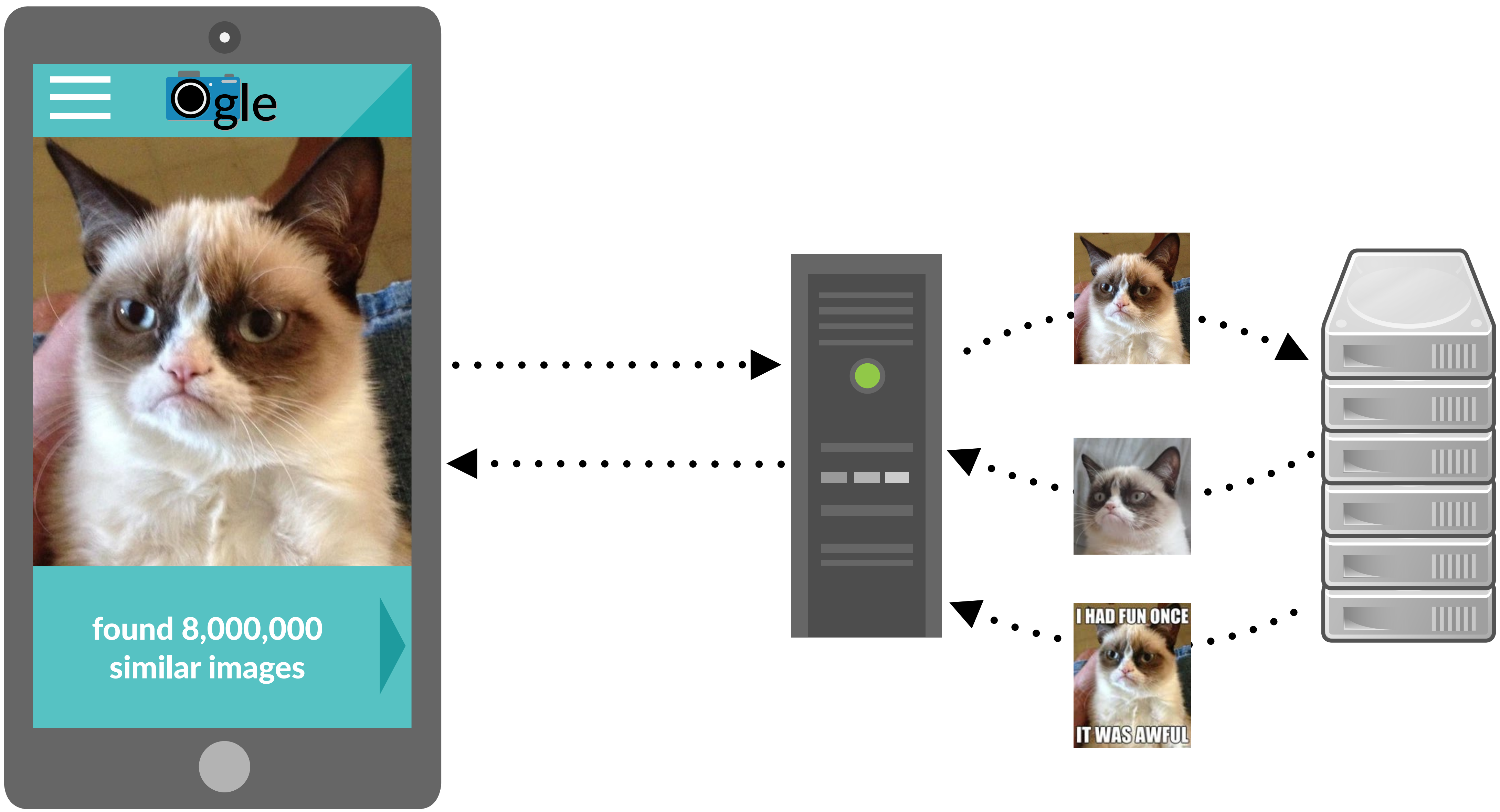
Speedup Results



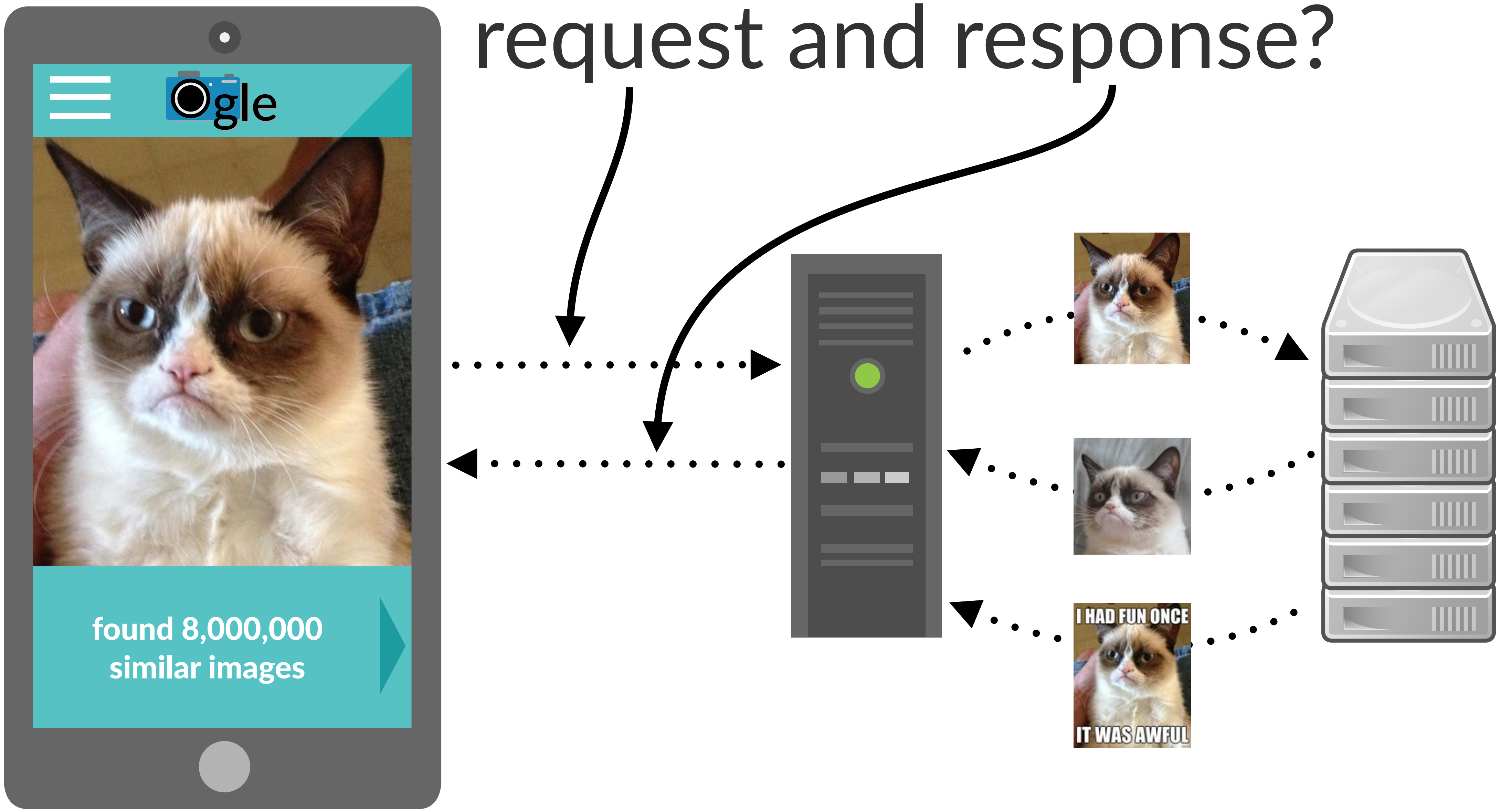
Speedup Results




Ogle
Take a picture
Send it to Ogle
Add it to the database
found 8,000,000 similar images
I HAD FUN ONCE
IT WAS AWFUL
Send results
Find similar pictures

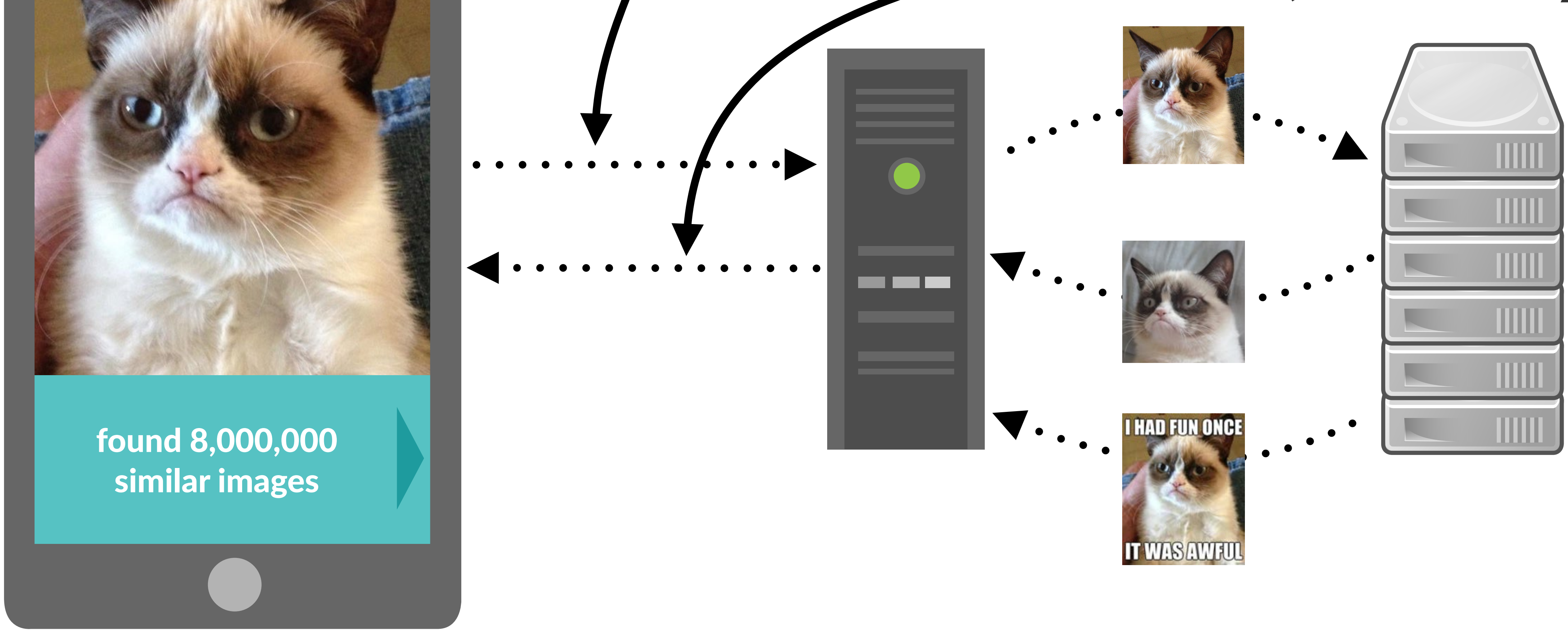

found 8,000,000
similar images
I HAD FUN ONCE
IT WAS AWFUL

How long between request and response?



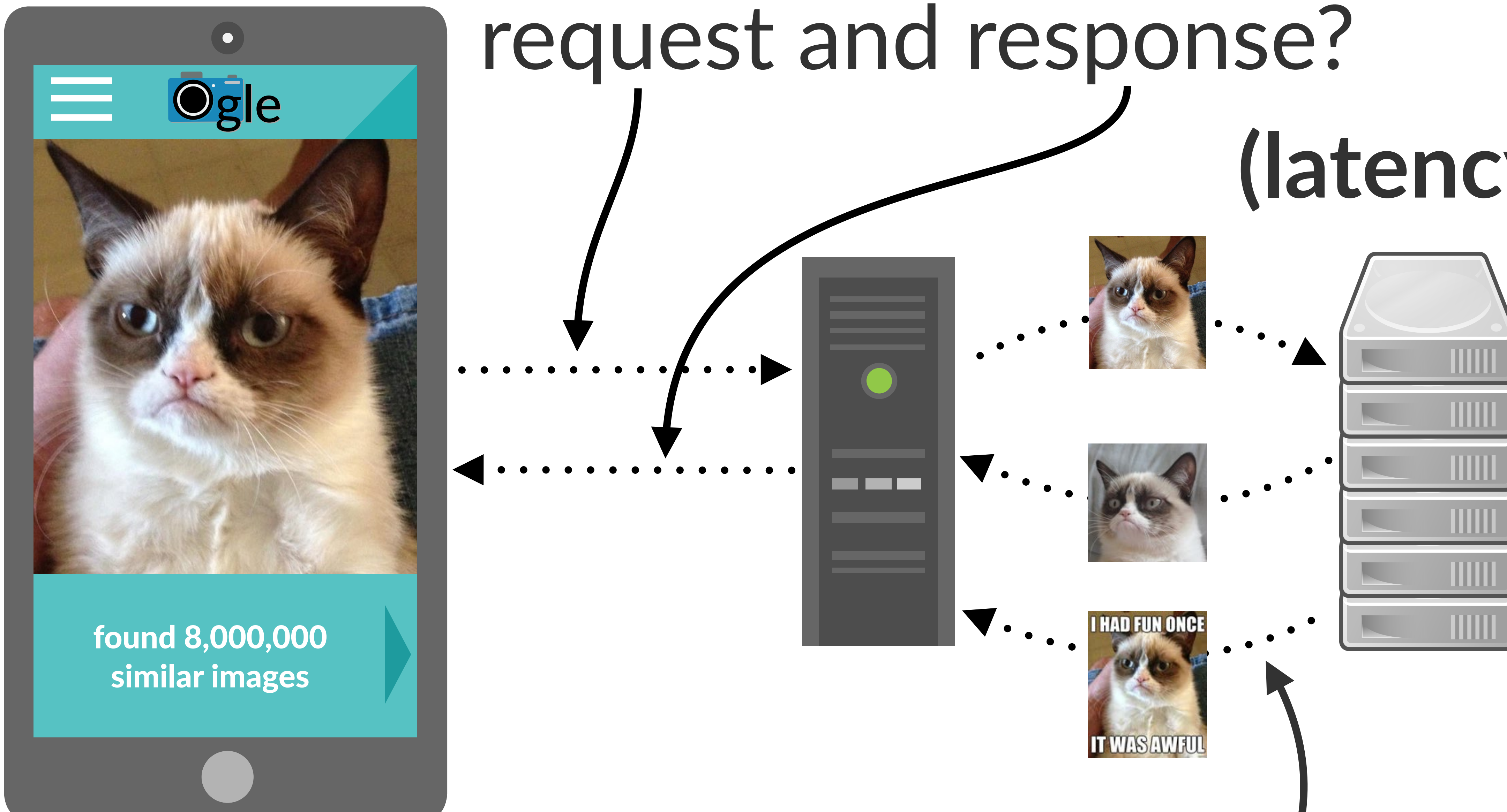
How long between request and response?

(latency)



How long between request and response?

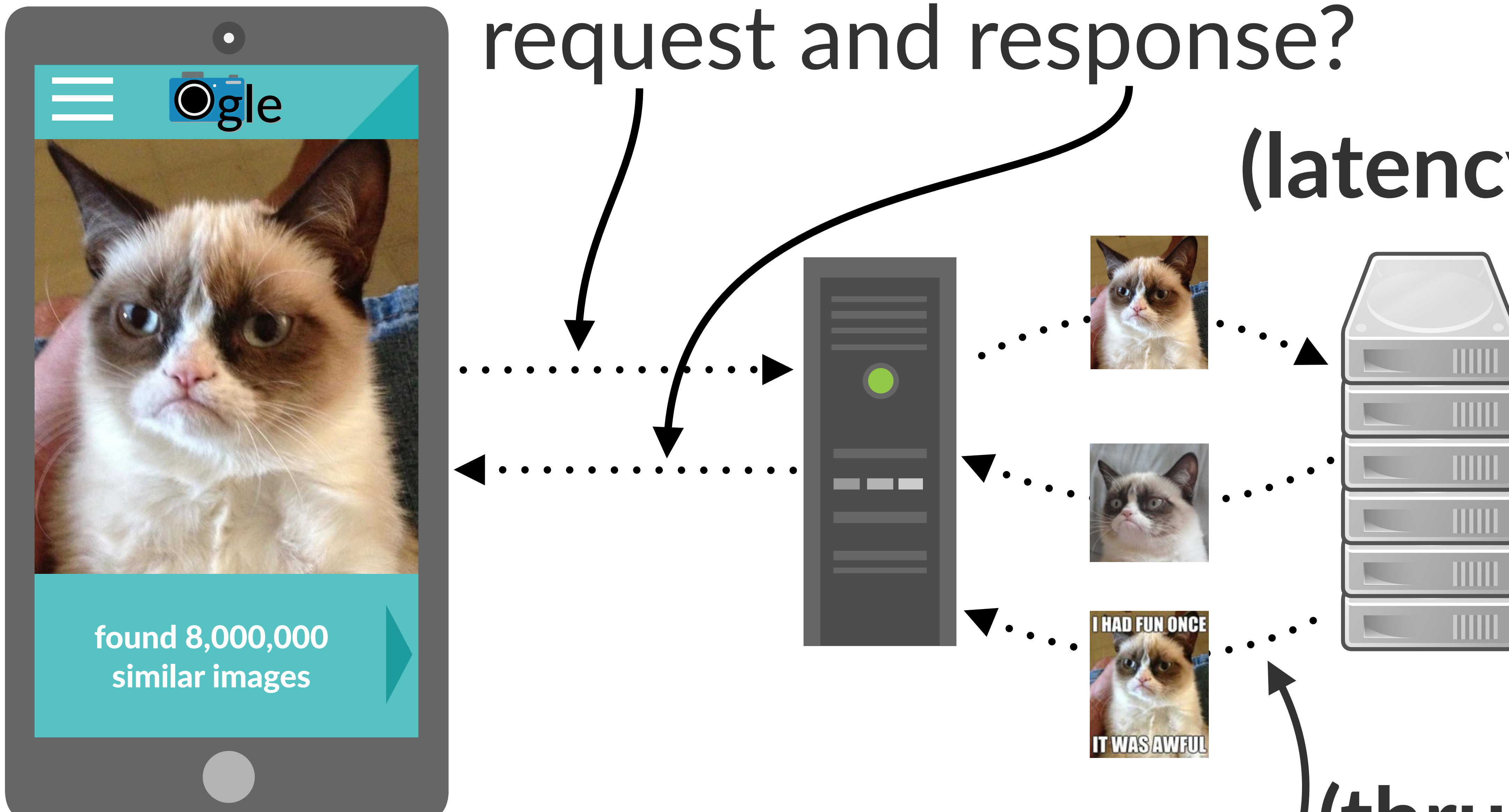
(latency)



How fast do results come back?

How long between request and response?

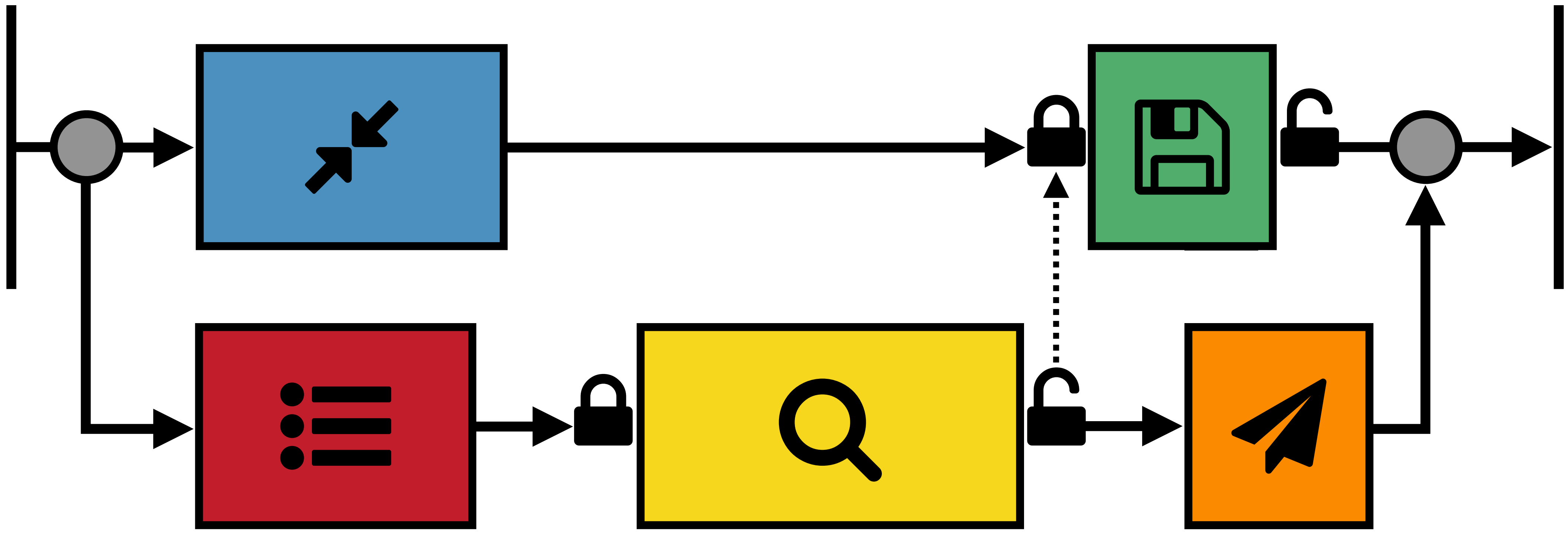
(latency)



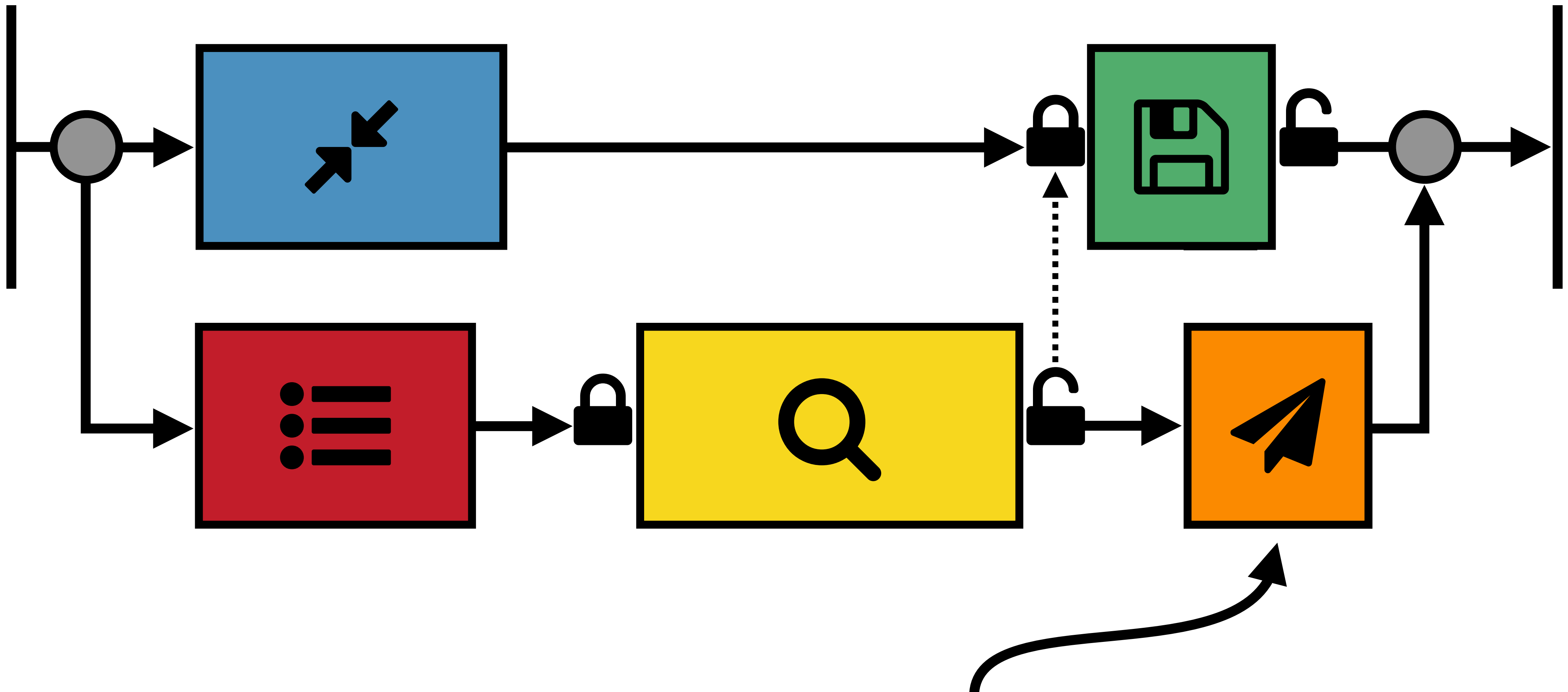
(thruput)

How fast do results come back?

Progress Points

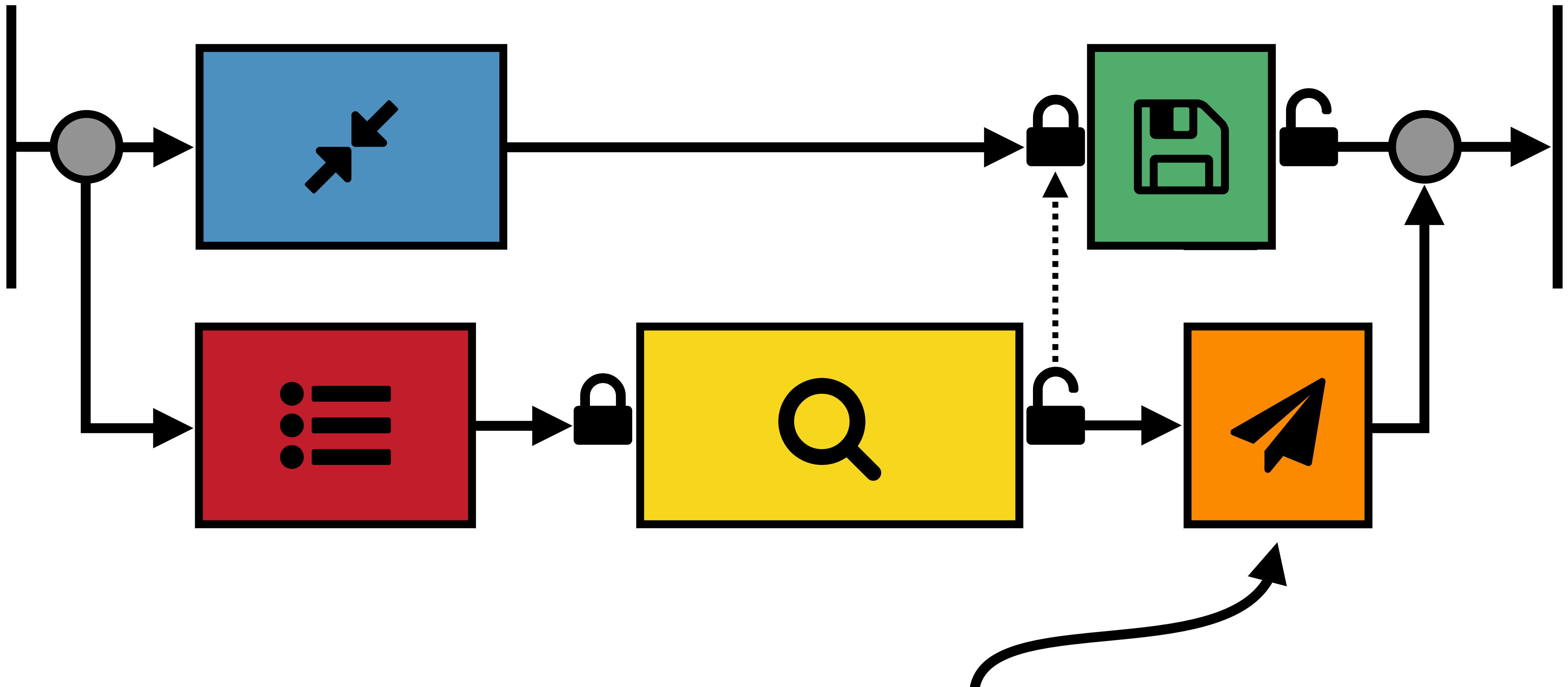


Progress Points



Homer wants to send responses faster.

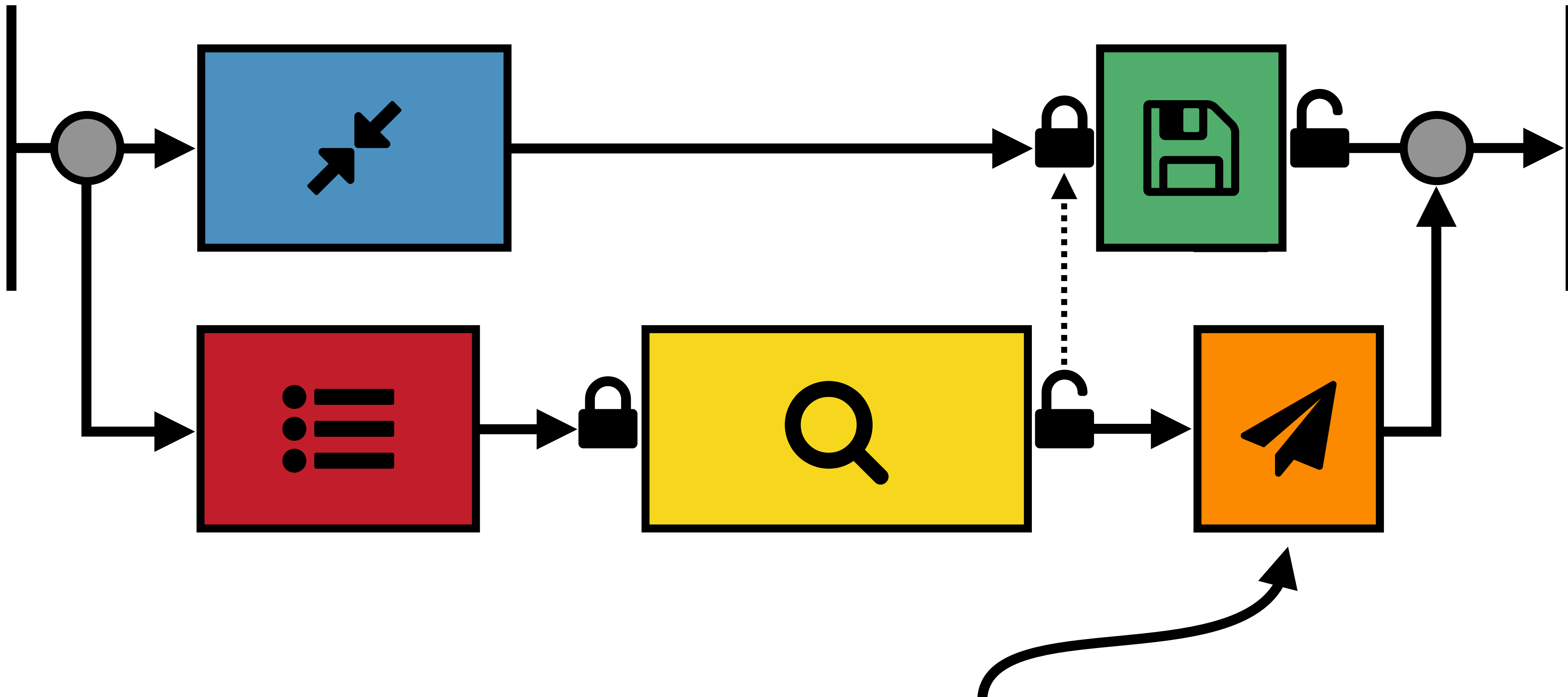
Progress Points



Homer wants to send responses faster.

He marks  as a *progress point*.

Progress Points

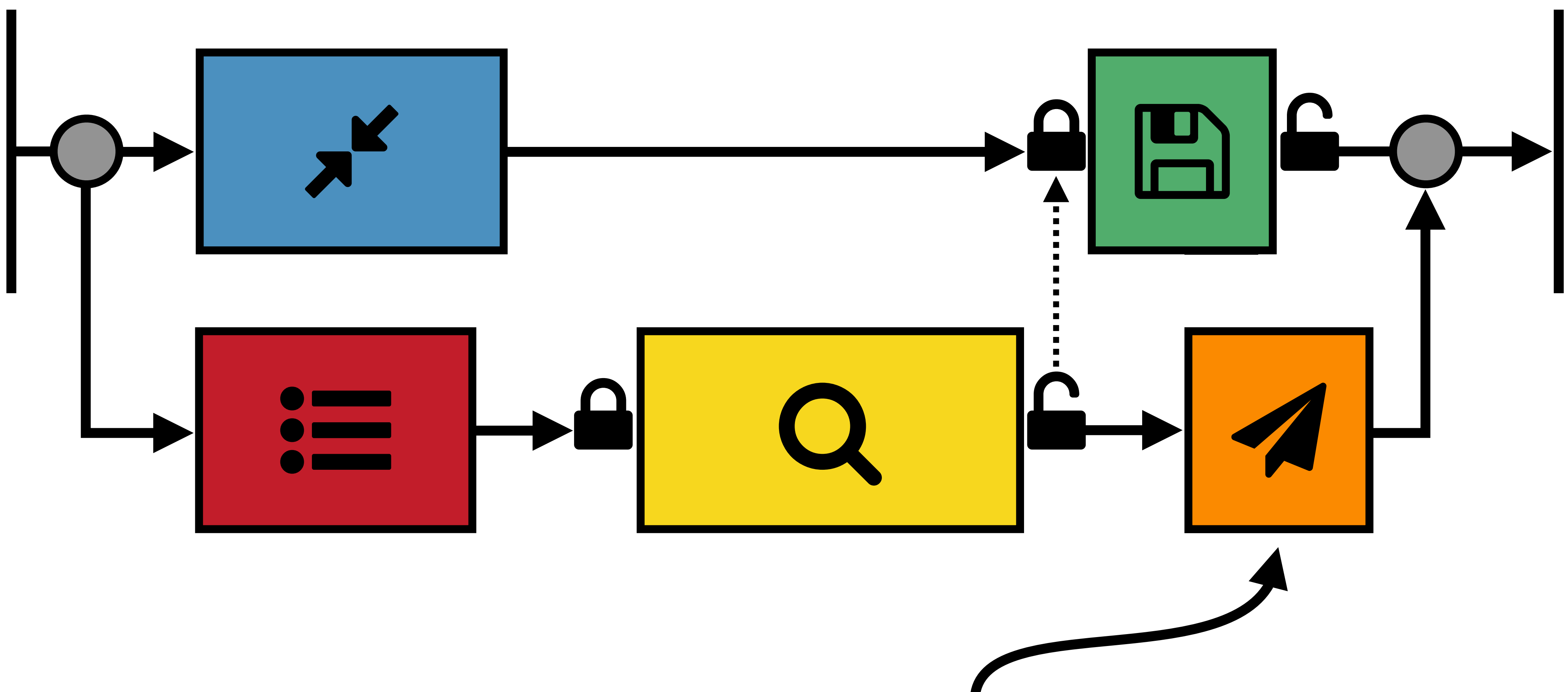


Homer wants to send responses faster.

He marks  as a *progress point*.

Each time this code runs...

Progress Points

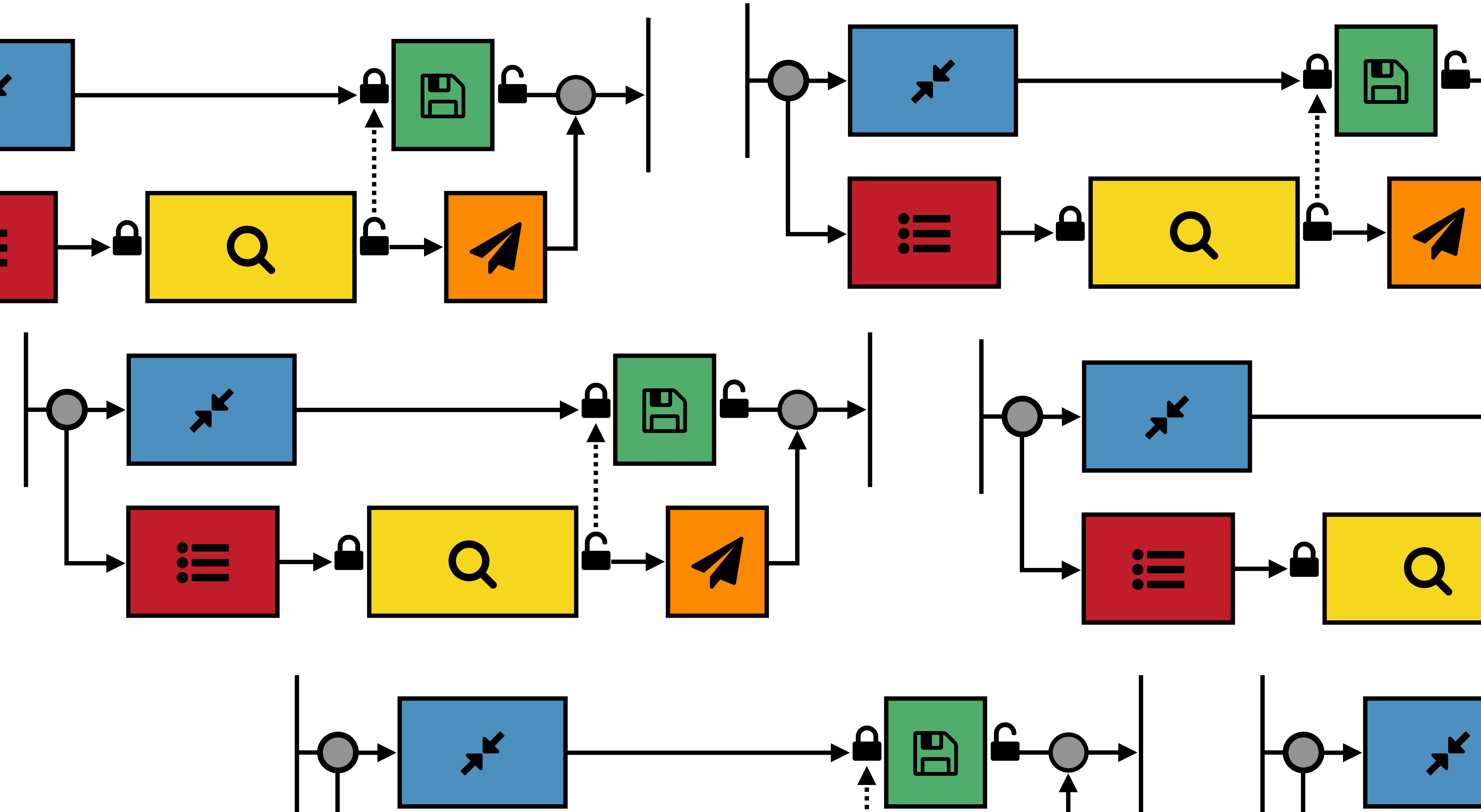


Homer wants to send responses faster.

He marks  as a *progress point*.

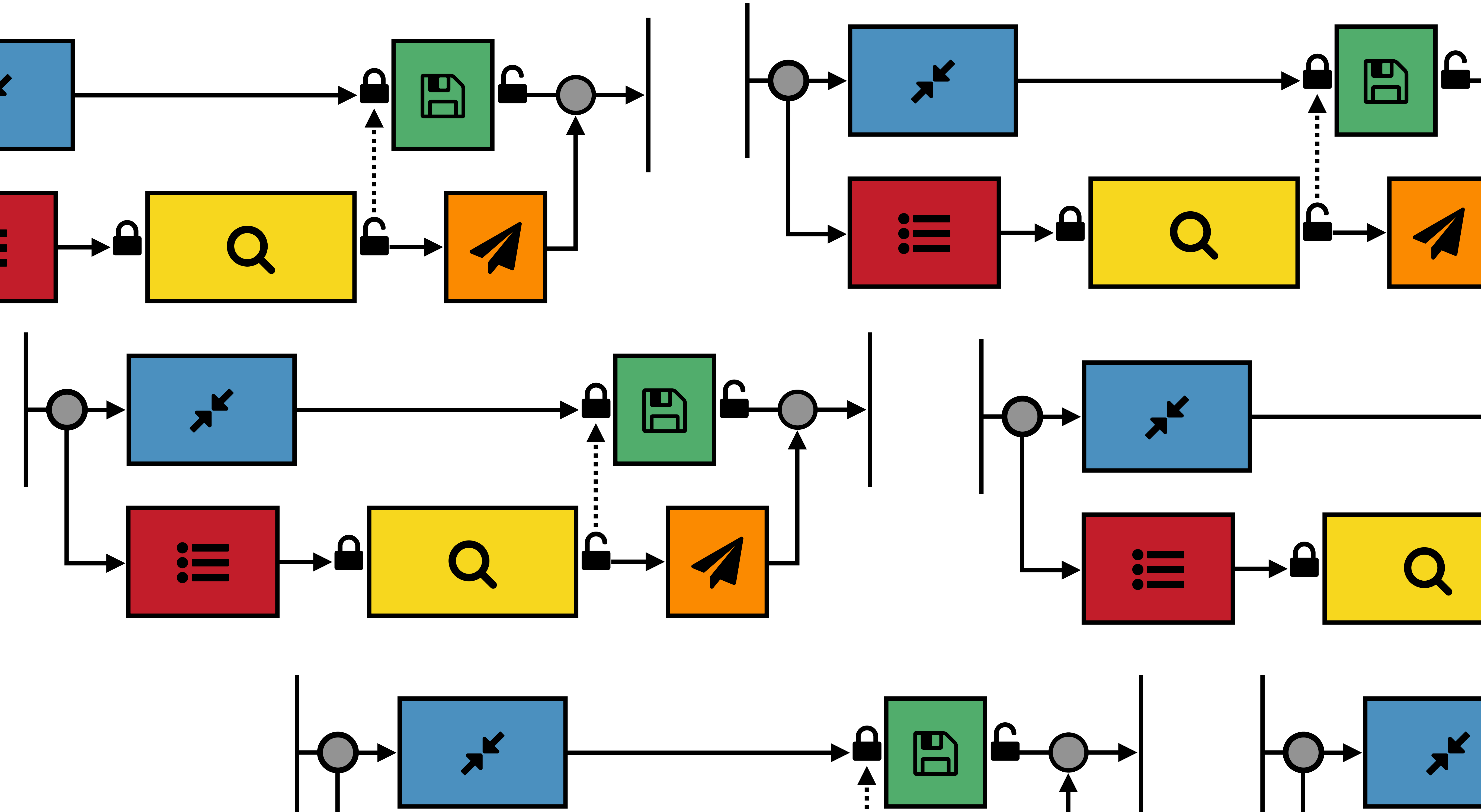
Each time this code runs…

Progress Points



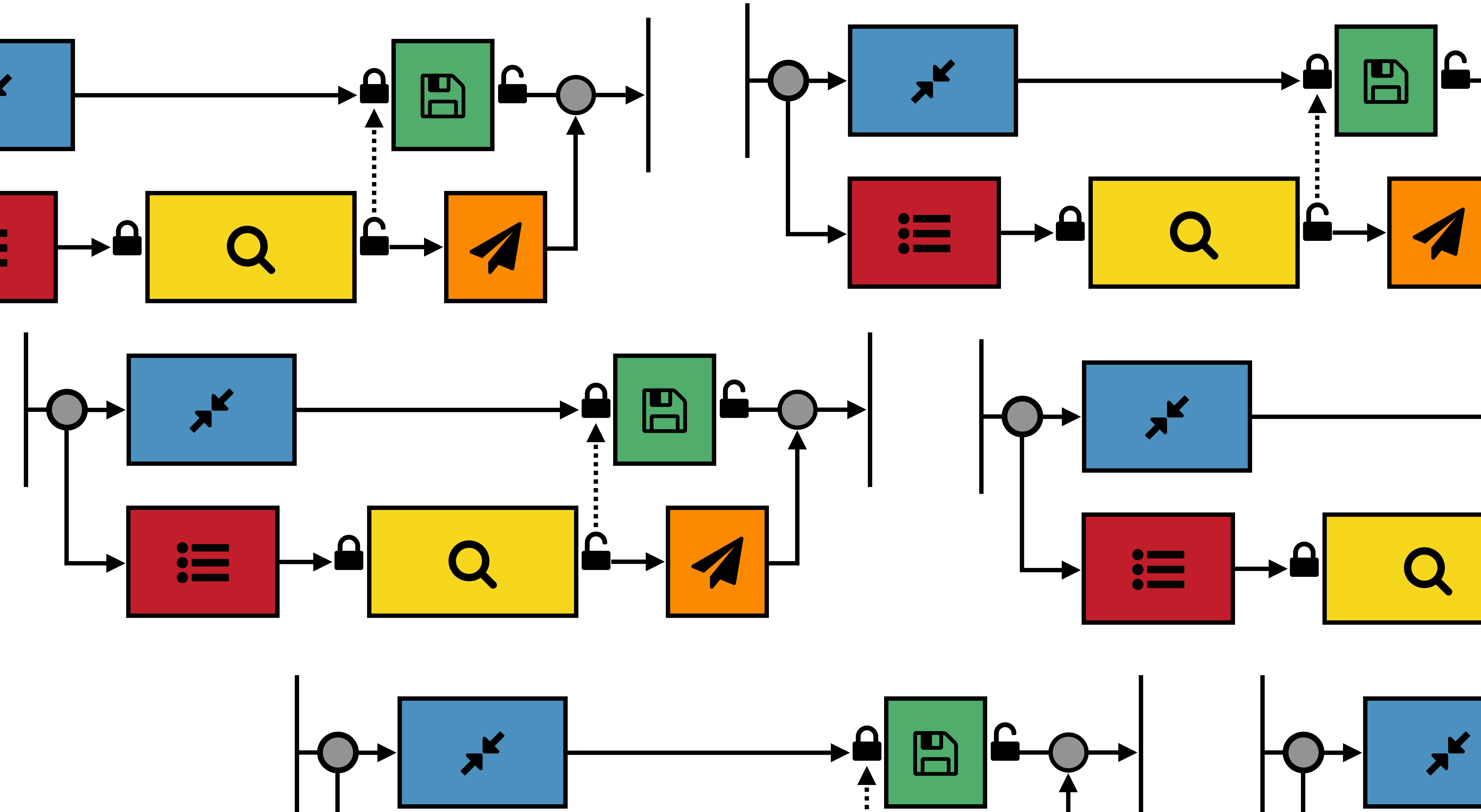
Progress Points

Many requests running for many users.

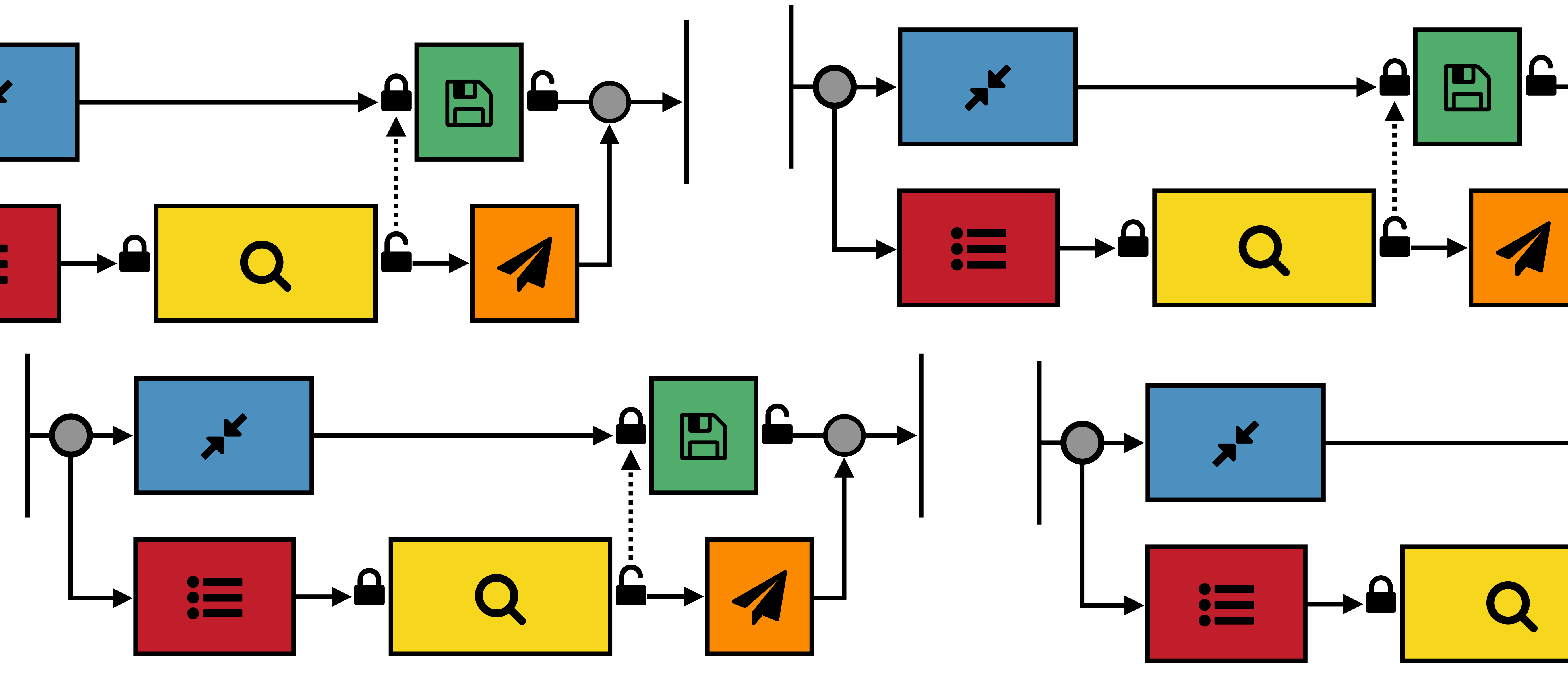


Progress Points

Many requests running for many users.

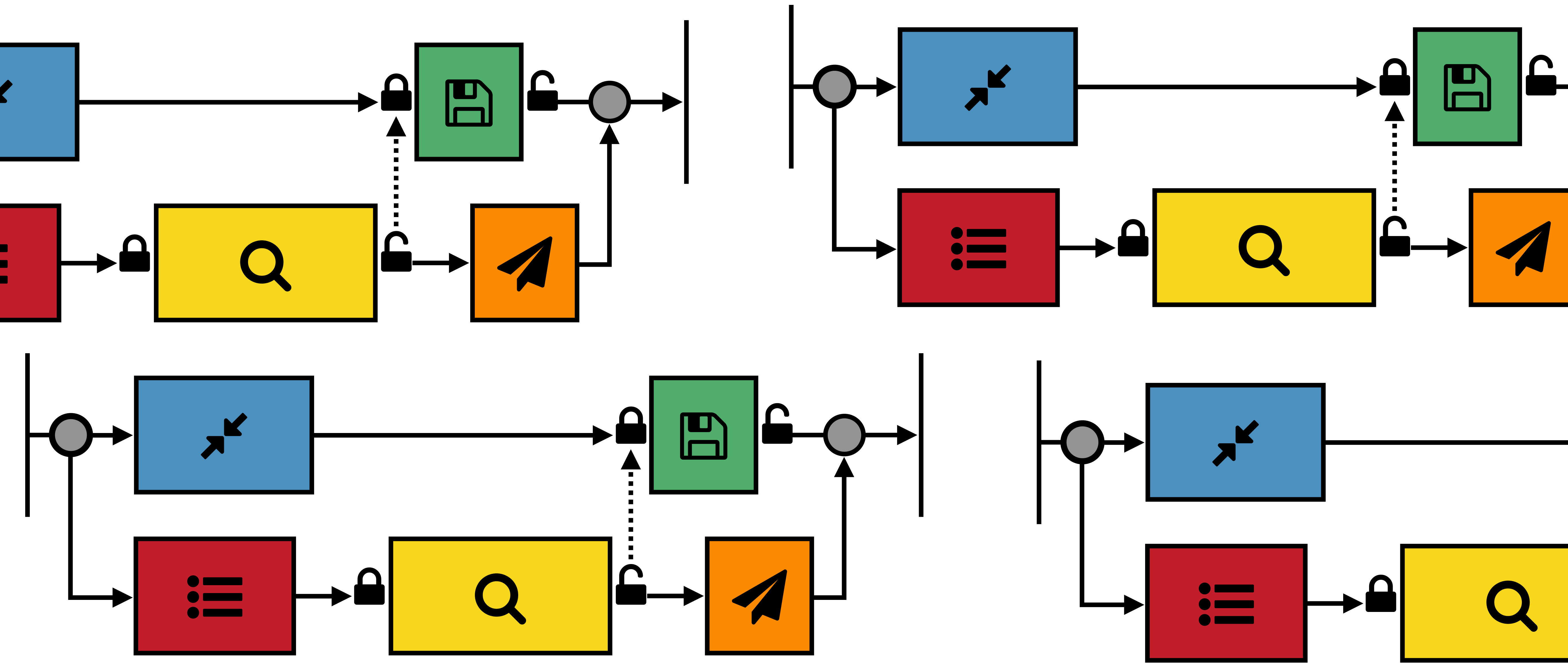


Progress Points



Progress Points

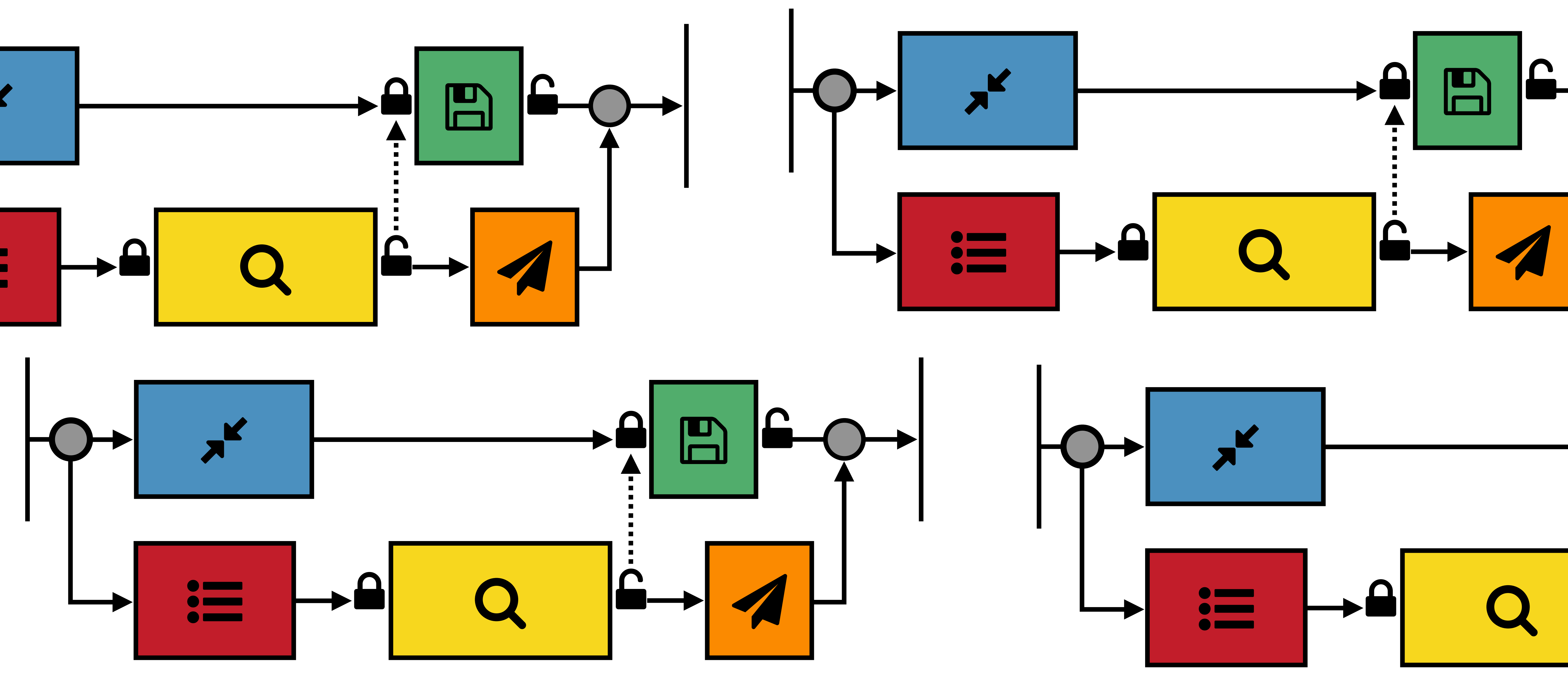
One progress point measures throughput.



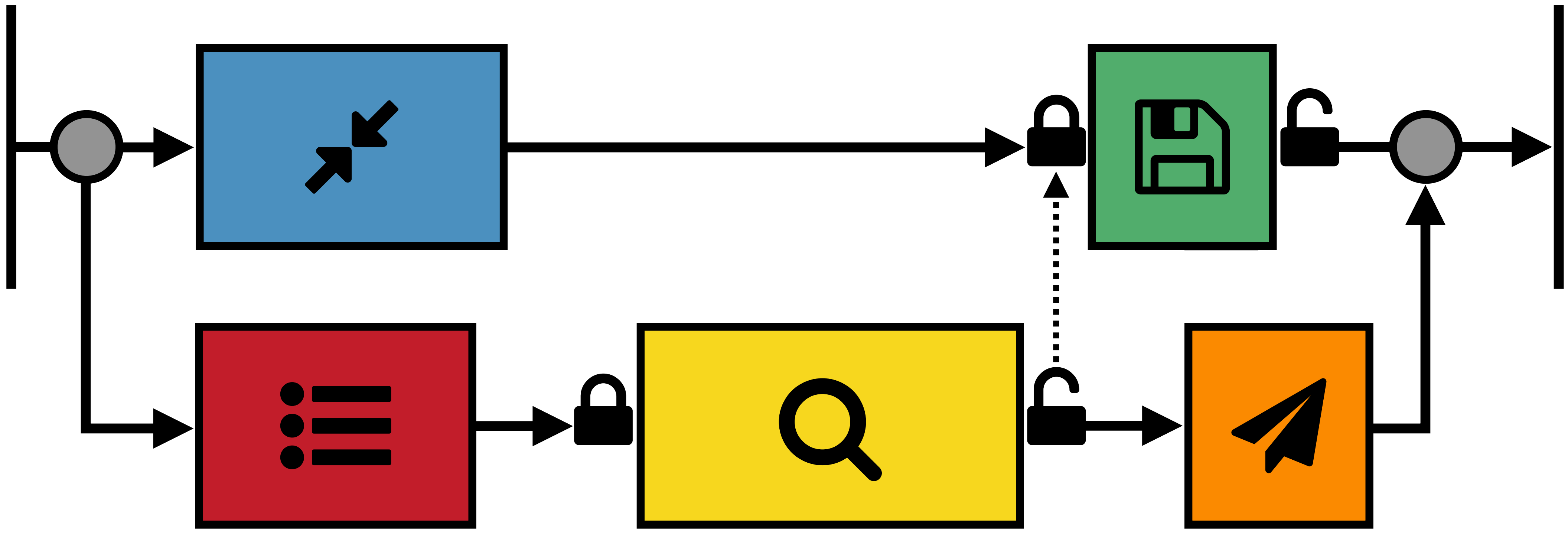
Progress Points

One progress point measures throughput.

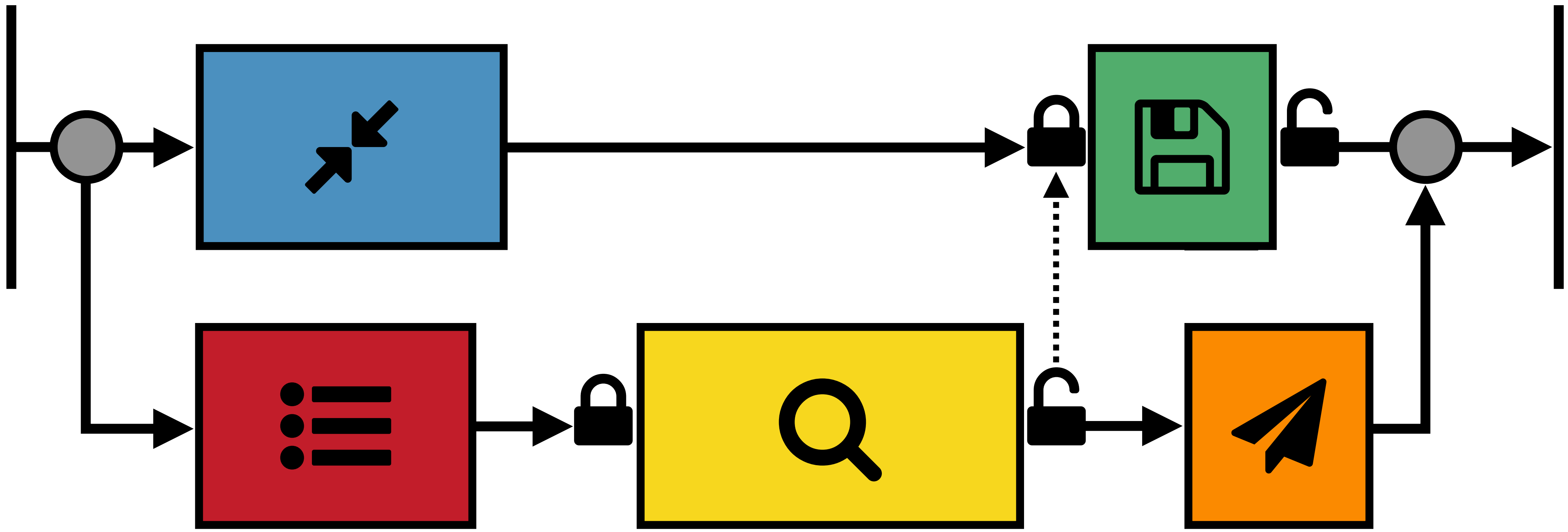
If I speed up ?, how much faster do I run ?



Progress Points

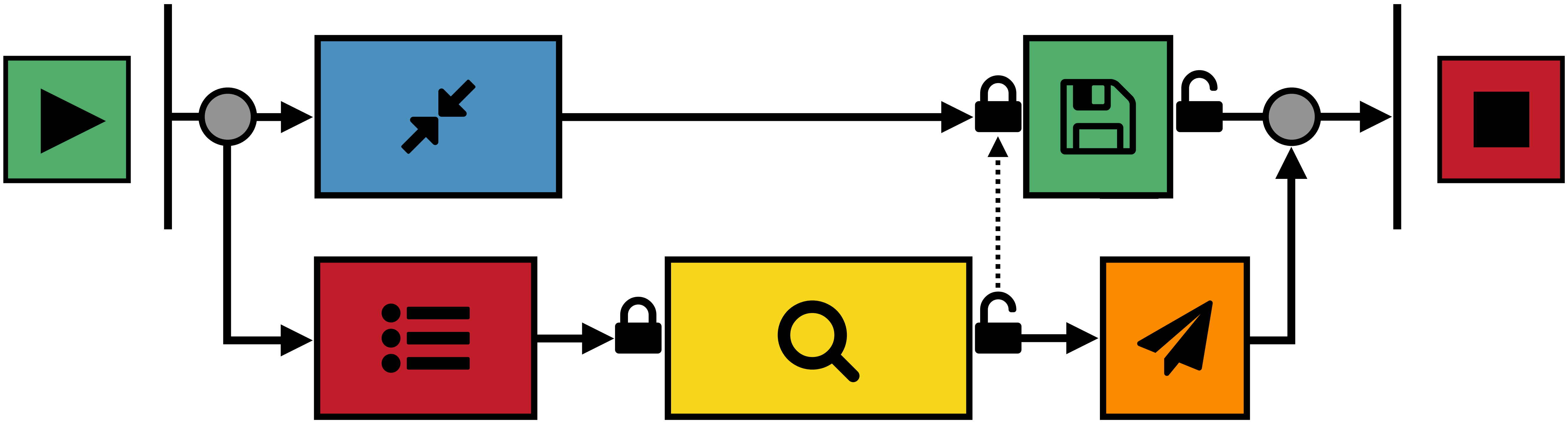


Progress Points



Homer wants to minimize response time.

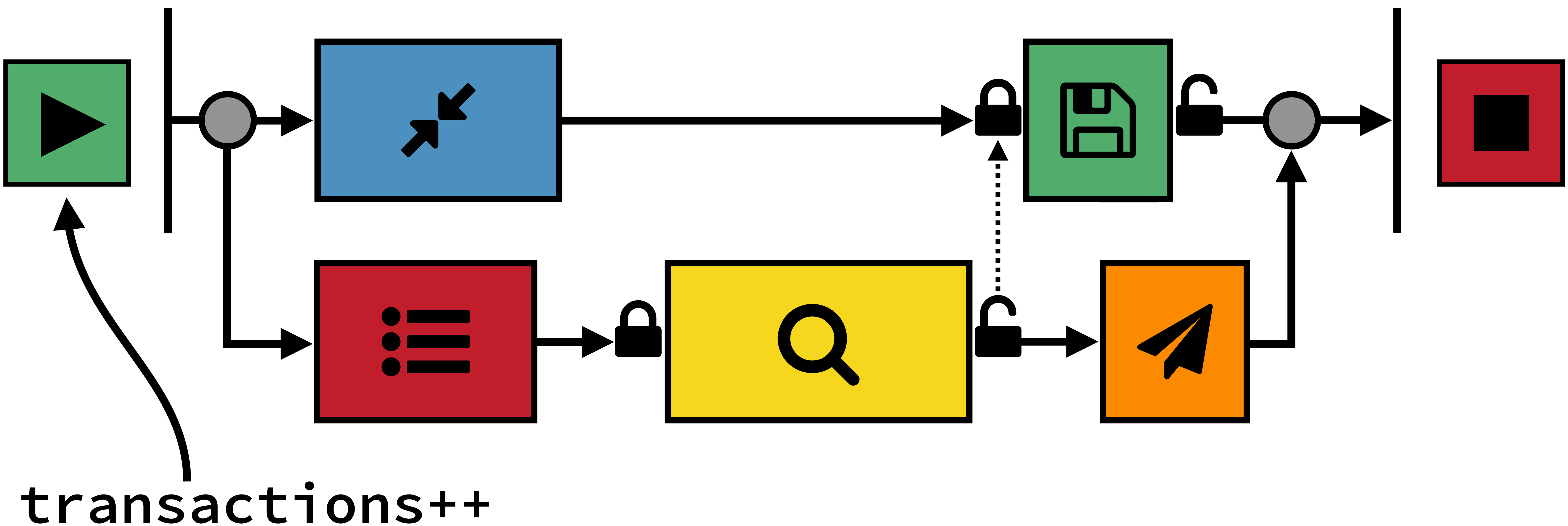
Progress Points



Homer wants to minimize response time.

He adds *latency progress points*.

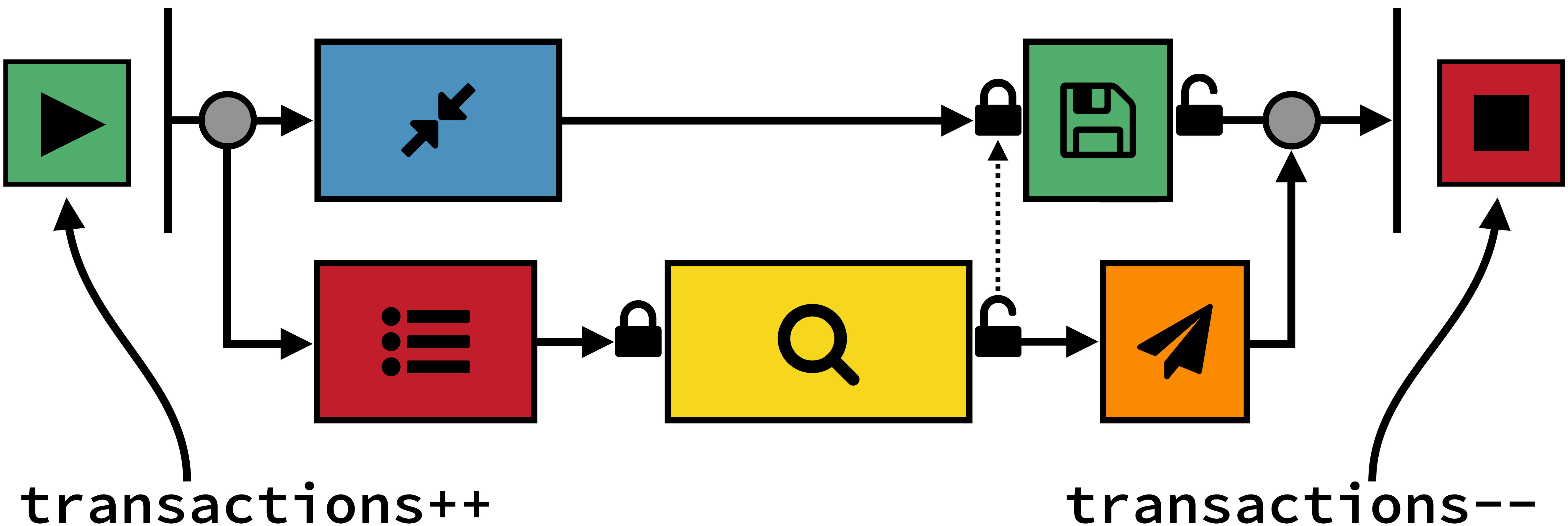
Progress Points



Homer wants to minimize response time.

He adds *latency progress points*.

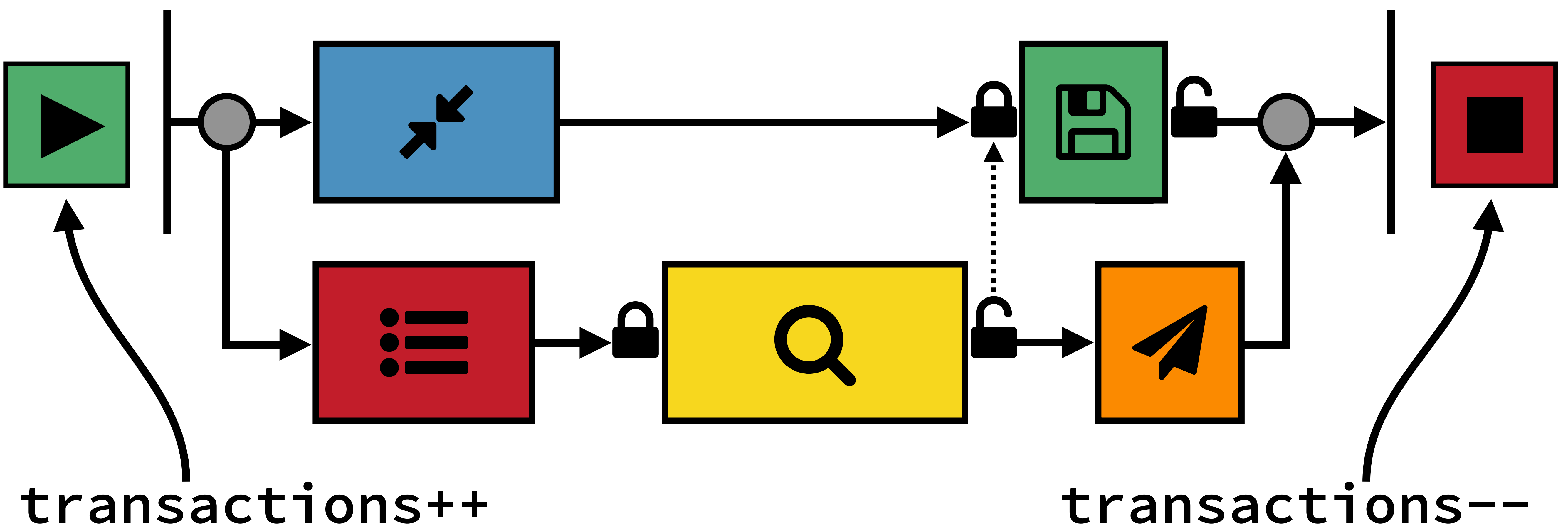
Progress Points



Homer wants to minimize response time.

He adds *latency progress points*.

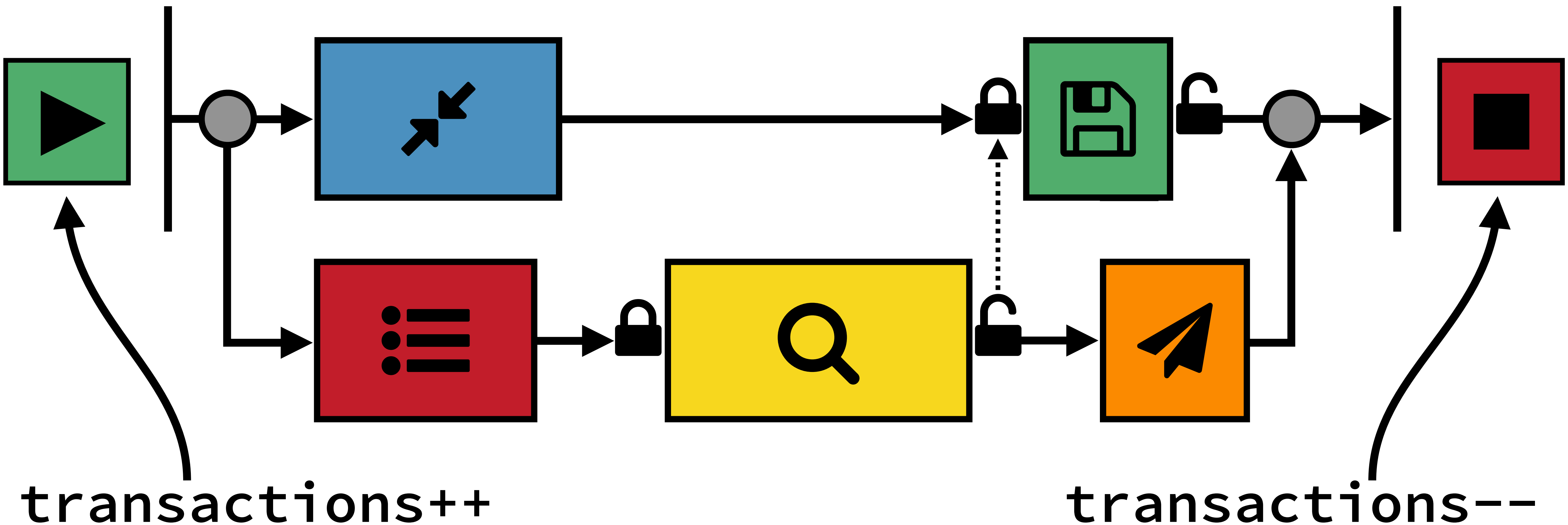
Progress Points



Homer wants to minimize response time.

Progress Points

Little's Law: W= L / λ

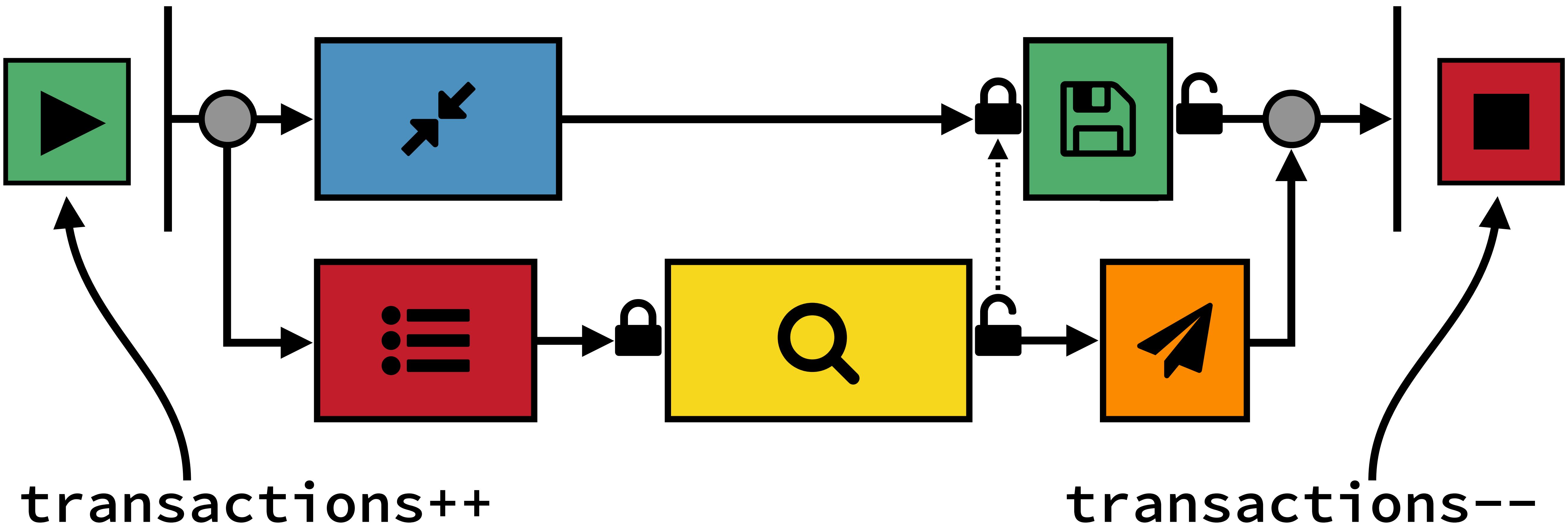


Homer wants to minimize response time.

Progress Points

Little's Law: W = L / λ

latency

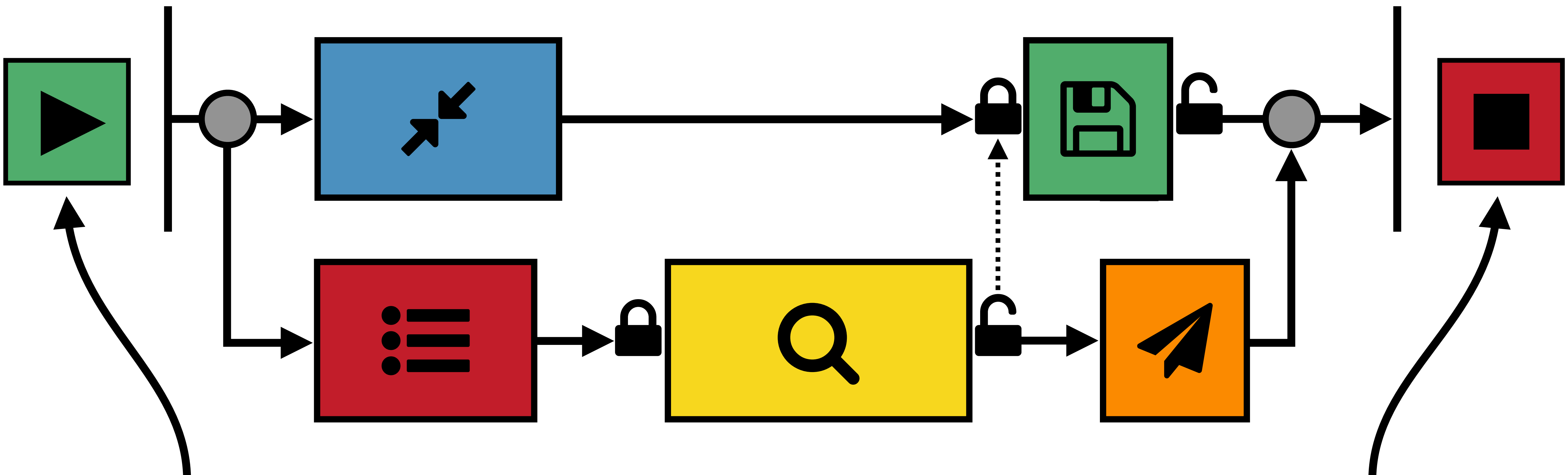


Homer wants to minimize response time.

Progress Points

Little's Law: W= L / λ

latency = `transactions`



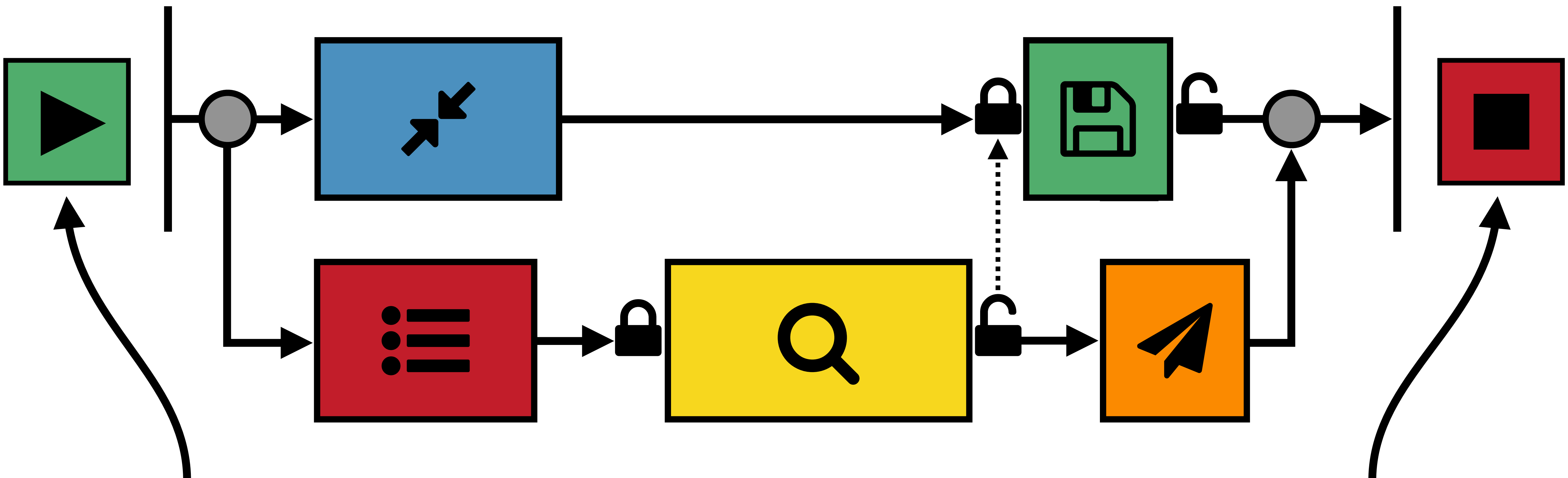
`transactions++` **`transactions--`**

Homer wants to minimize response time.

Progress Points

Little's Law: W= L / λ

latency = `transactions` / throughput



Homer wants to minimize response time.

Coz: a Causal Profiler for Linux

(ships with Debian / Ubuntu)

```
>
```

Coz: a Causal Profiler for Linux

(ships with Debian / Ubuntu)

```
> sudo apt install coz-profiler
```

Coz: a Causal Profiler for Linux

(ships with Debian / Ubuntu)

```
> sudo apt install coz-profiler
> coz run --- ./some_program args
```

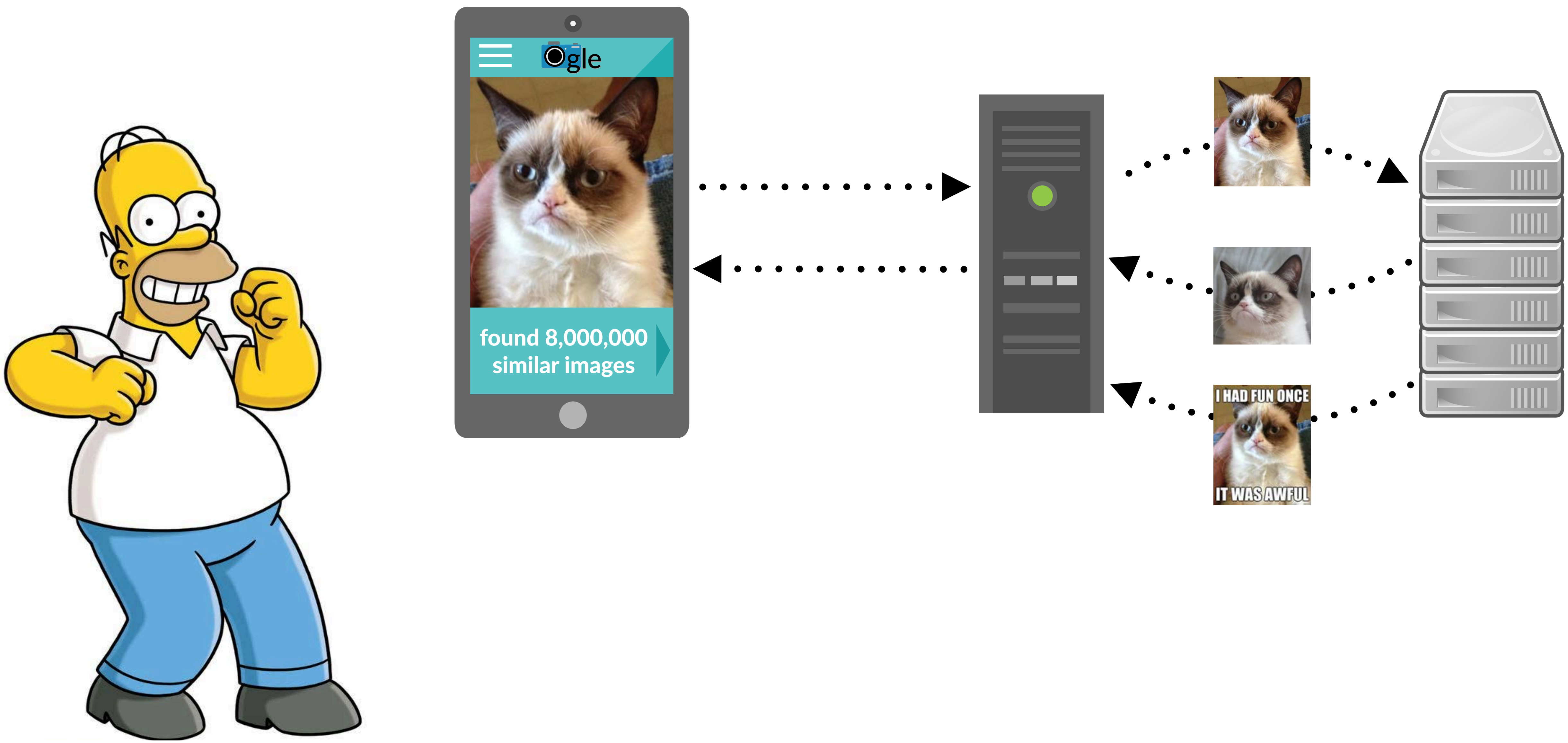
Coz: a Causal Profiler for Linux

(ships with Debian / Ubuntu)

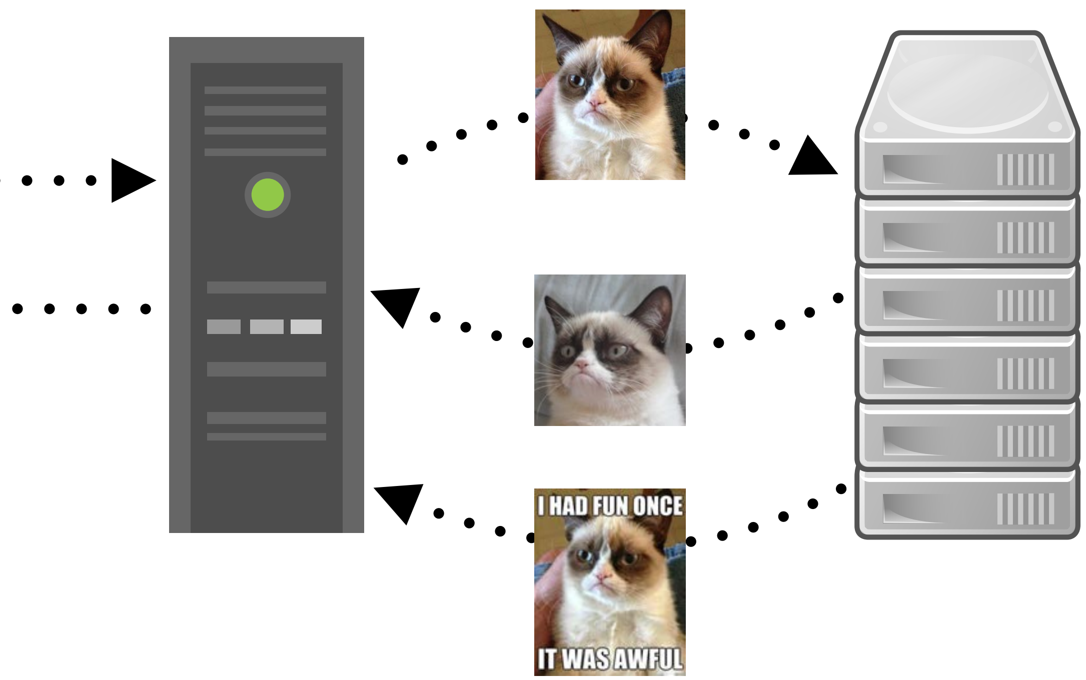
```
> sudo apt install coz-profiler
> coz run --- ./some_program args
```

Random *performance experiments*

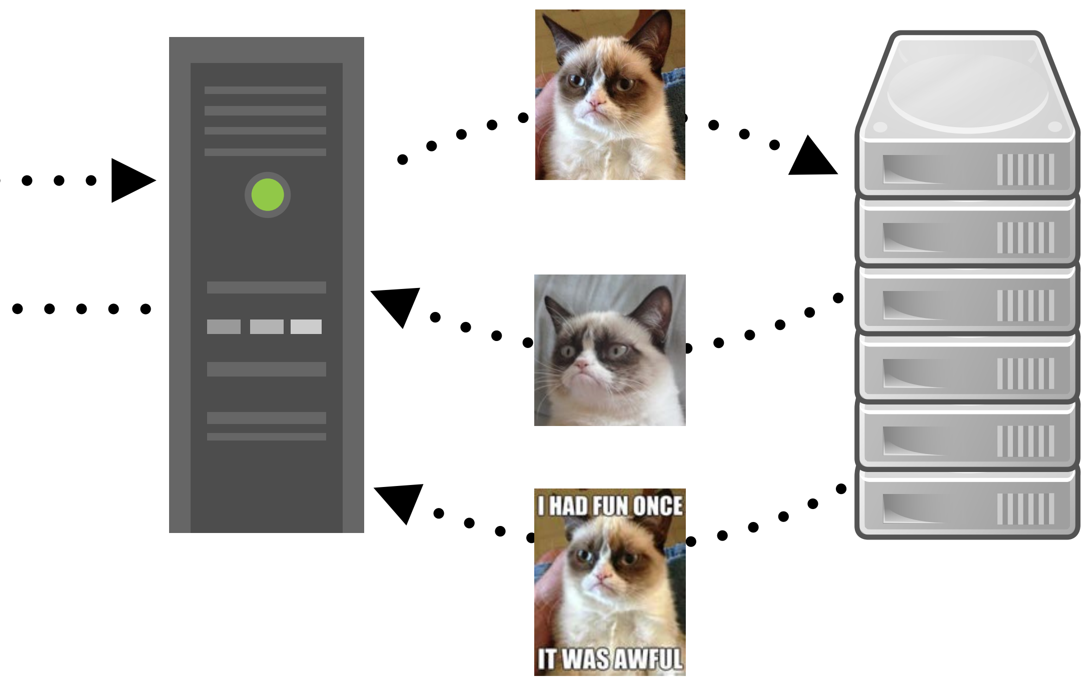
Using Causal Profiling on Ogle



Using Causal Profiling on Ogle

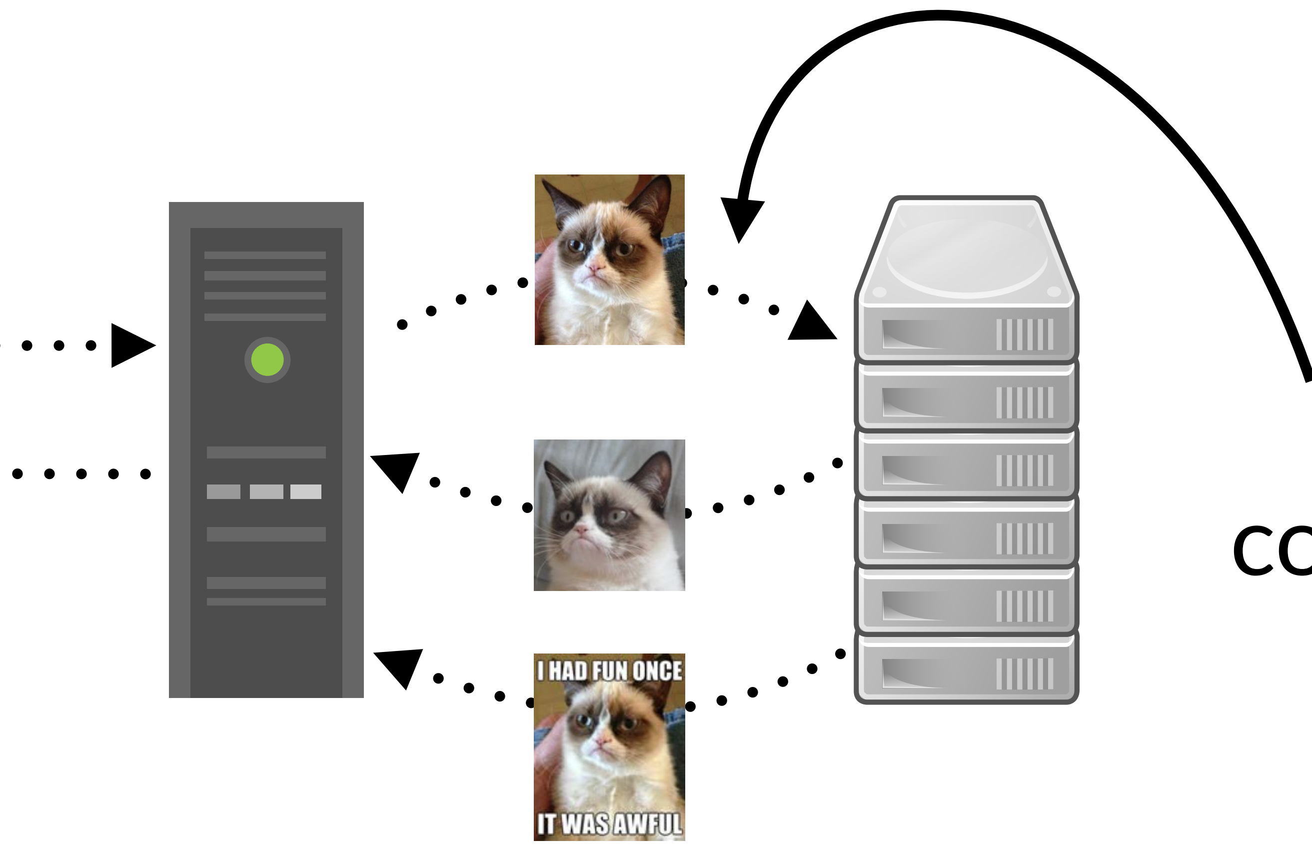


Using Causal Profiling on Ogle



PARSEC
['pär-''sek] A unit of measure

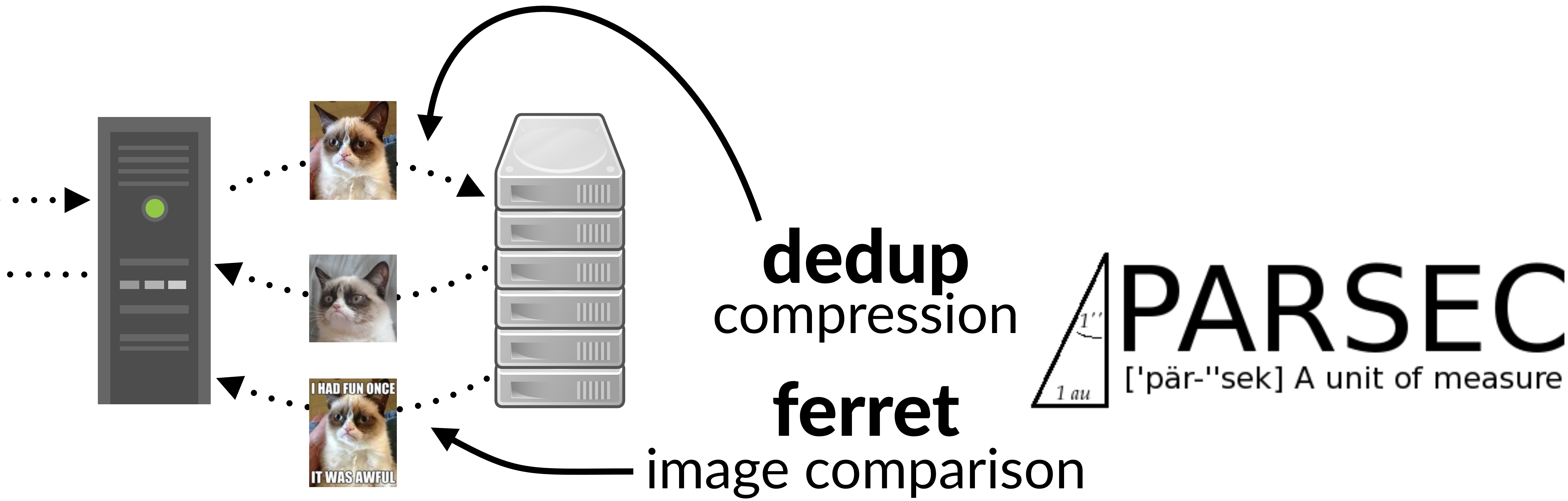
Using Causal Profiling on Ogle



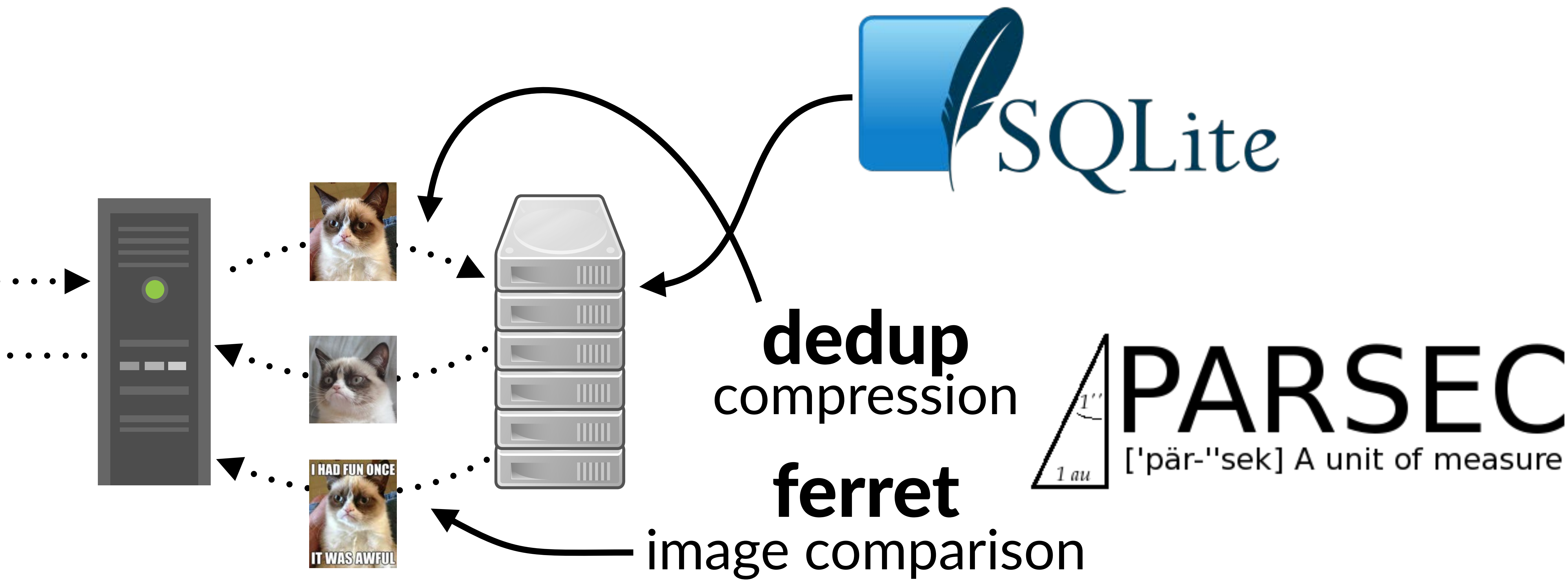
dedup
compression

PARSEC
['pär-''sek] A unit of measure

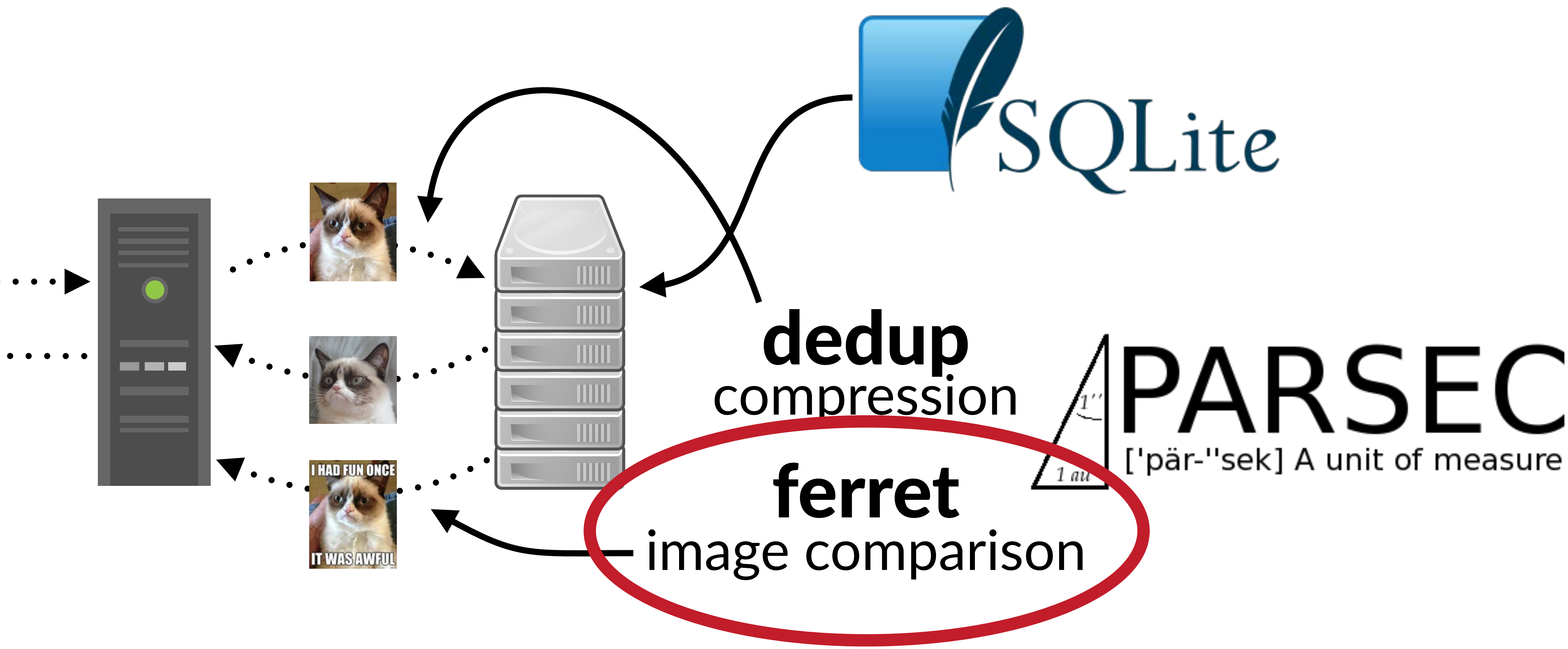
Using Causal Profiling on Ogle

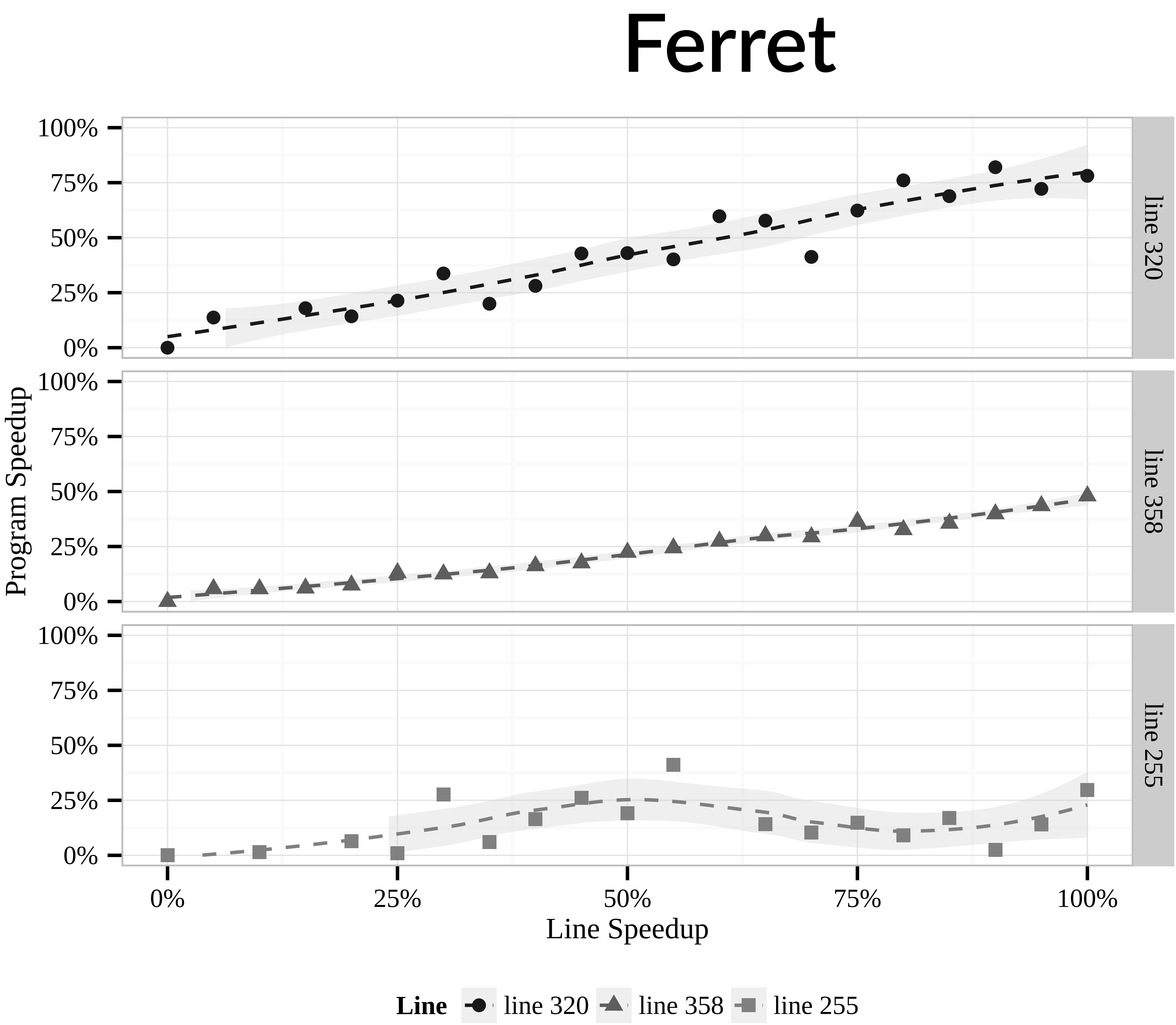


Using Causal Profiling on Ogle

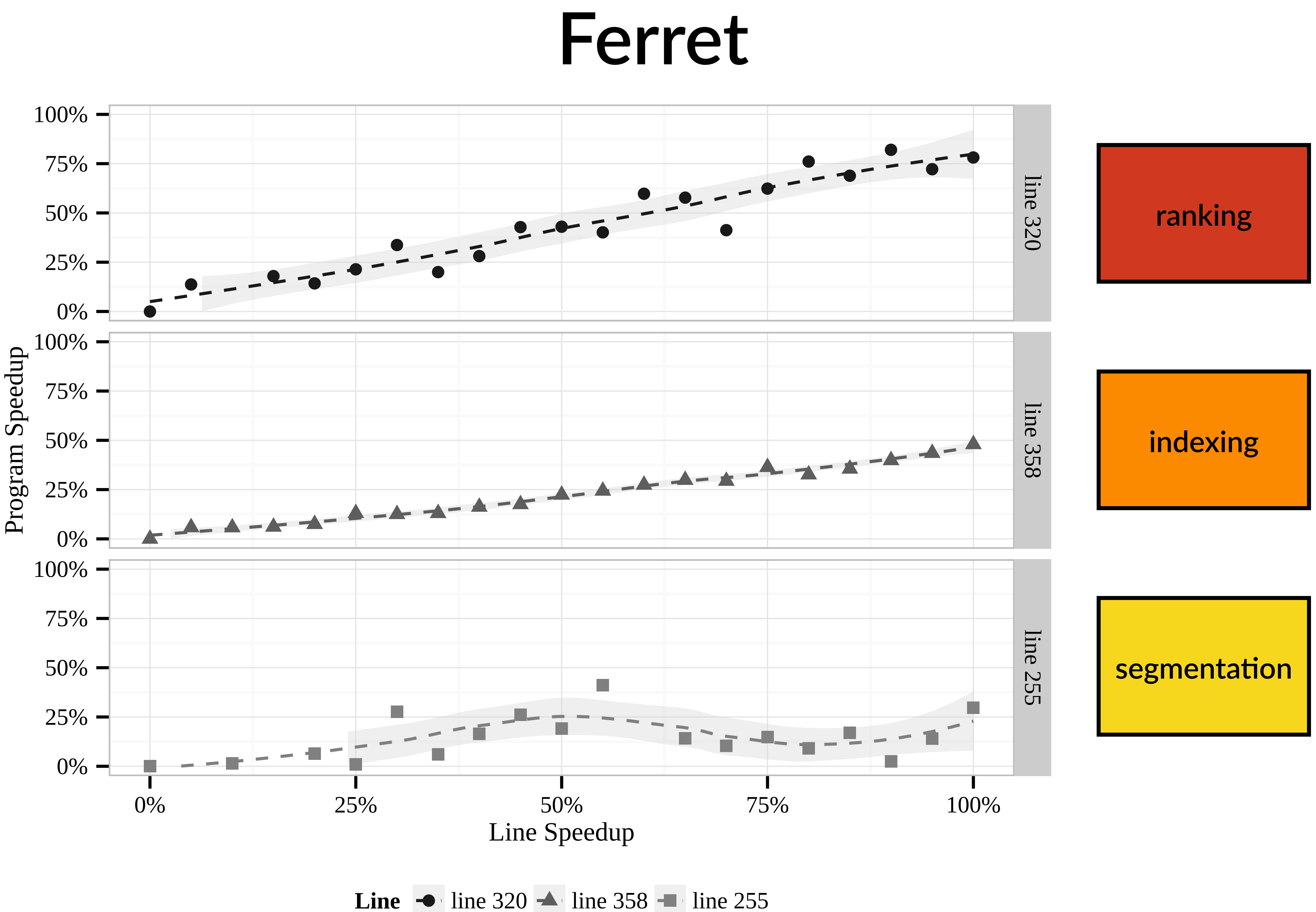


Using Causal Profiling on Ogle



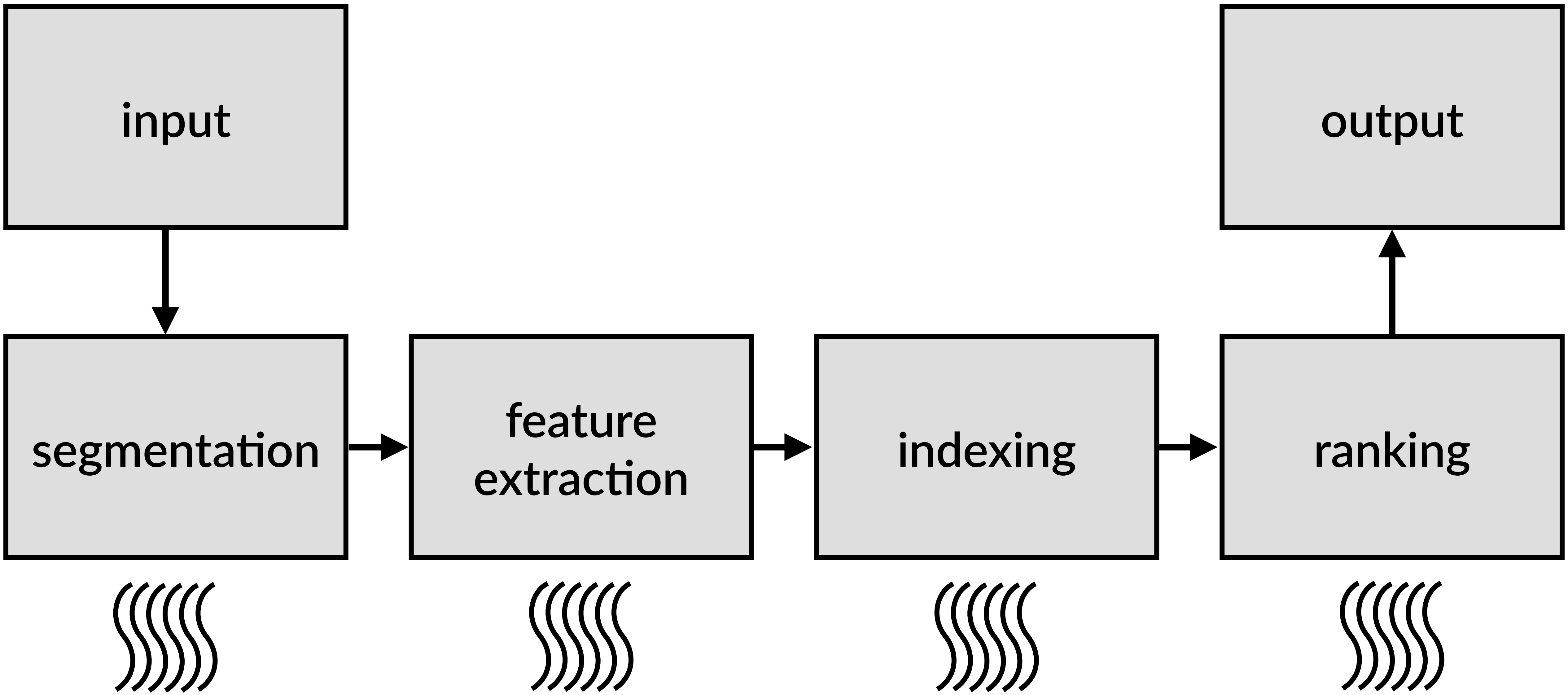


Ferret
Program Speedup
Line Speedup
100%
75%
50%
25%
0%
line 320
line 358
line 255
Line
line 320
line 358
line 255



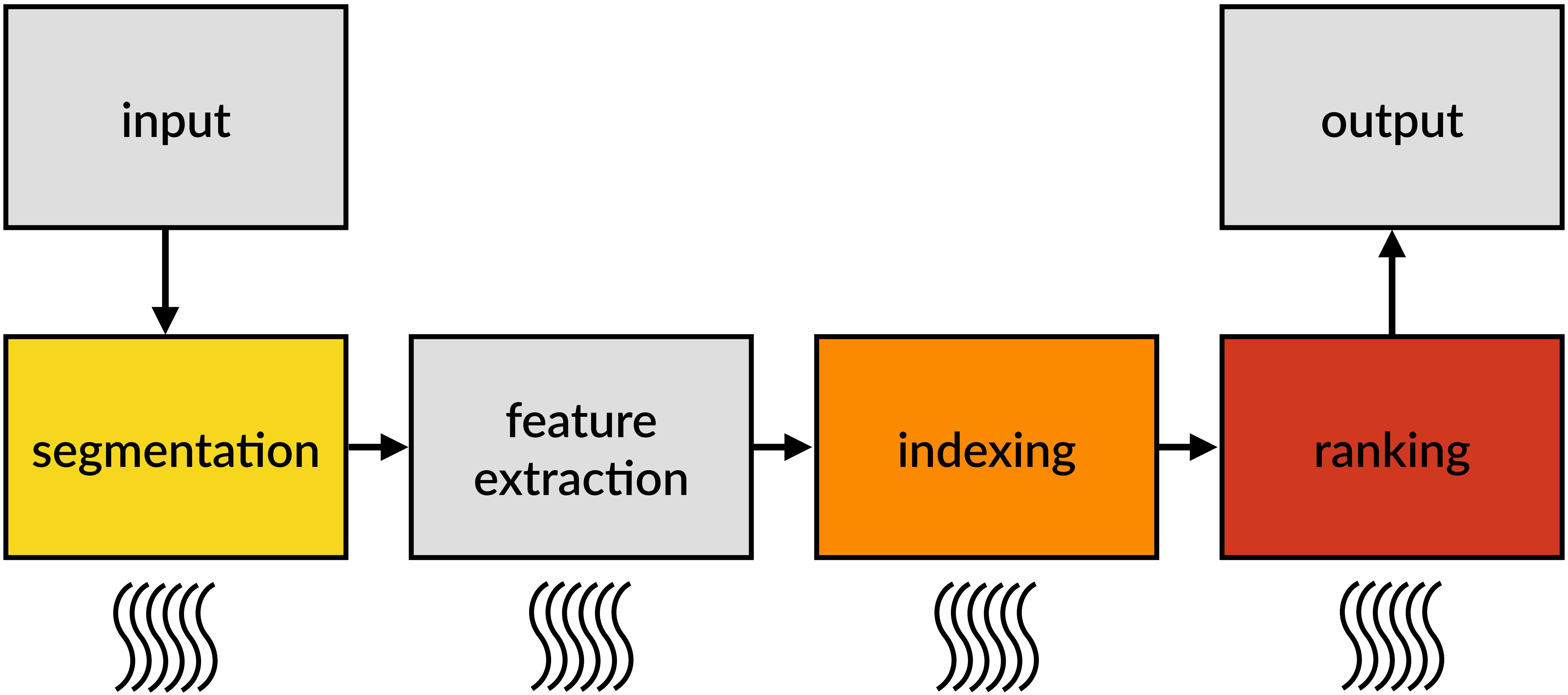
Ferret
Program Speedup
Line Speedup
0%
25%
50%
75%
100%
line 320
line 358
line 255
ranking
indexing
segmentation
Line
line 320
line 358
line 255

Ferret

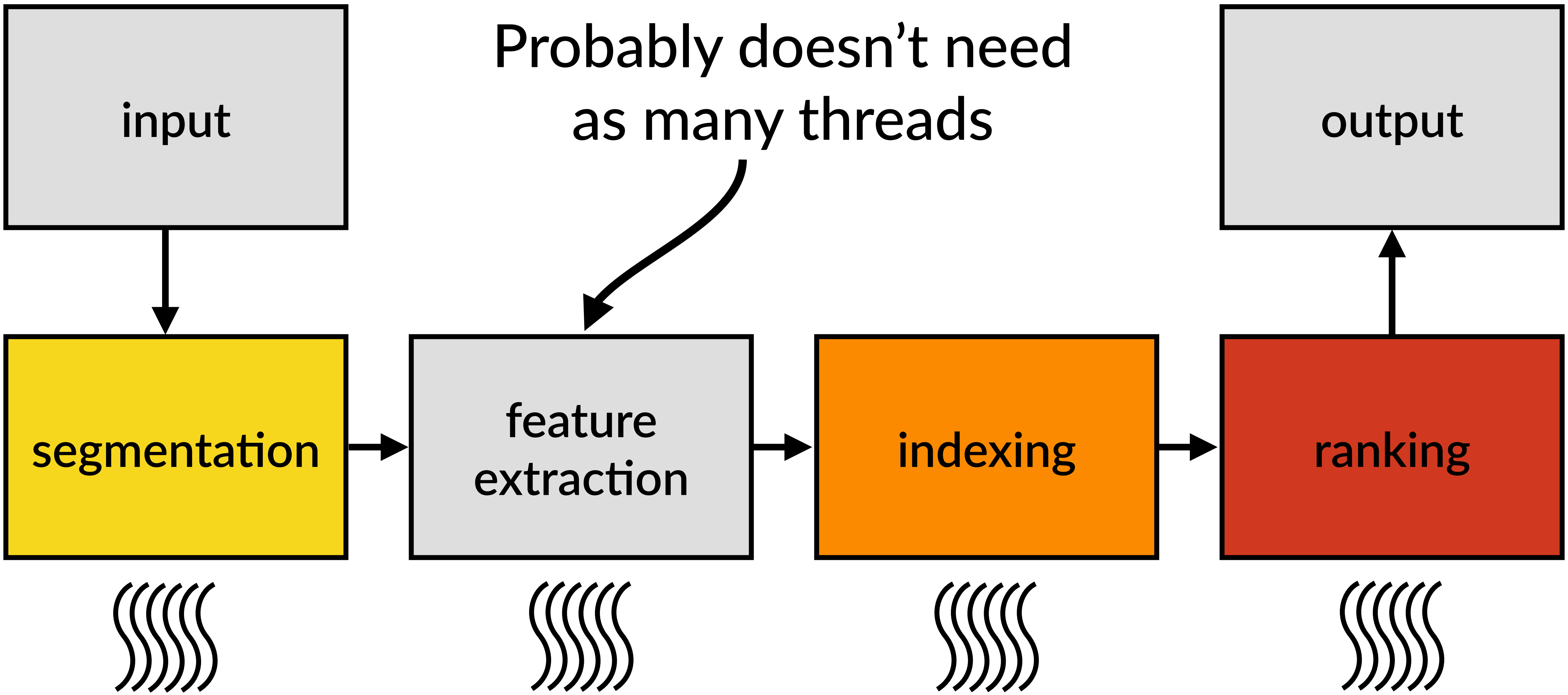


input
output
segmentation
feature extraction
indexing
ranking

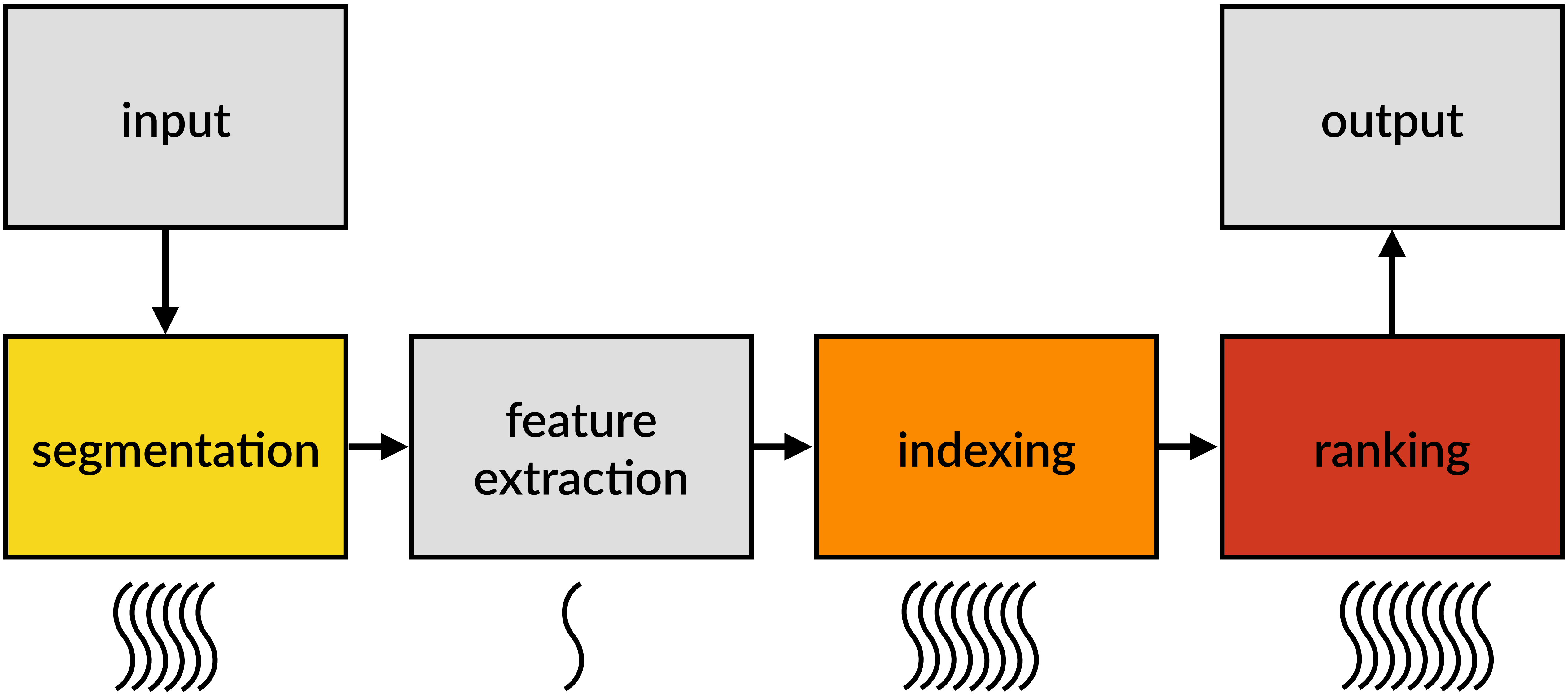
Ferret


input
output
segmentation
feature extraction
indexing
ranking

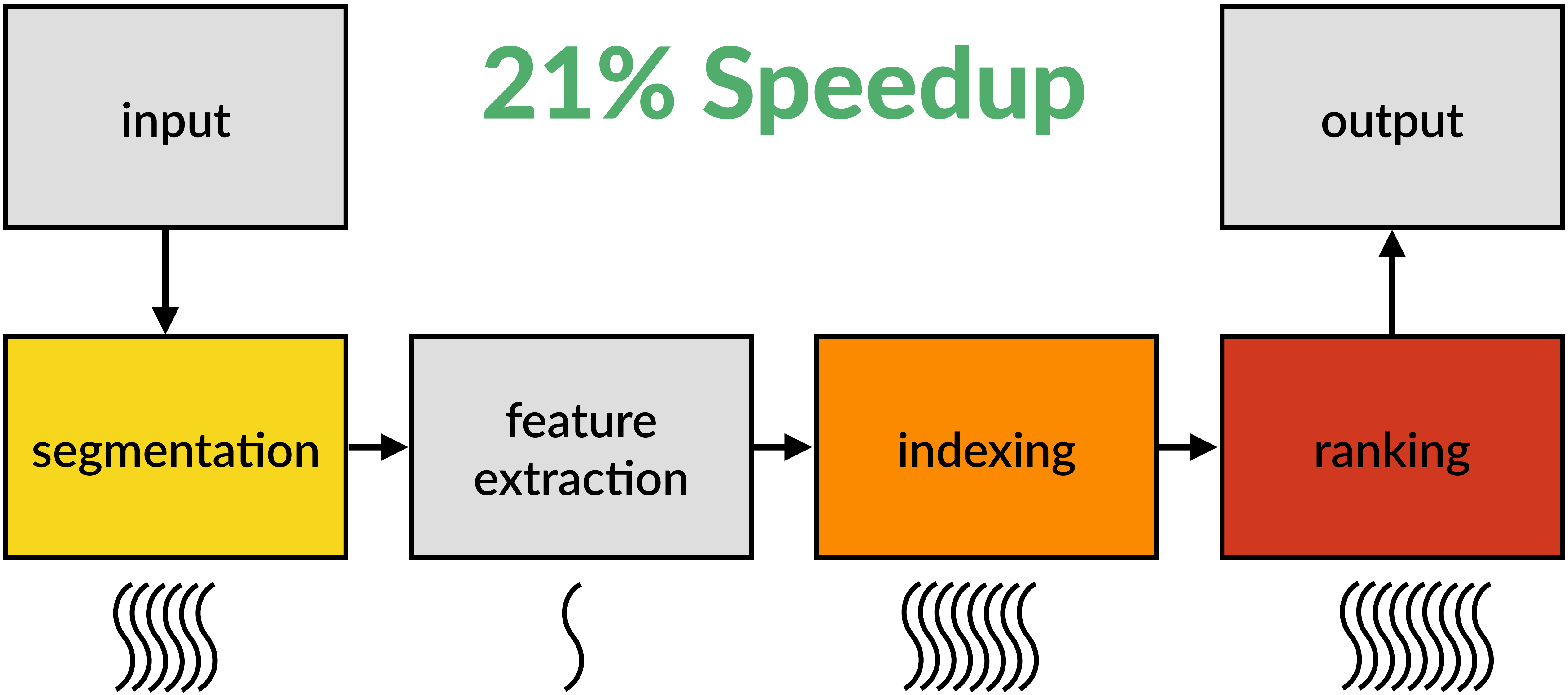
Ferret


input
Probably doesn't need as many threads
output
segmentation
feature extraction
indexing
ranking

Ferret


input
output
segmentation
feature extraction
indexing
ranking

Ferret


input
21% Speedup
output
segmentation
feature extraction
indexing
ranking

What did Coz predict?

ranking

What did Coz predict?

ranking

Increased from 16 to 22 threads

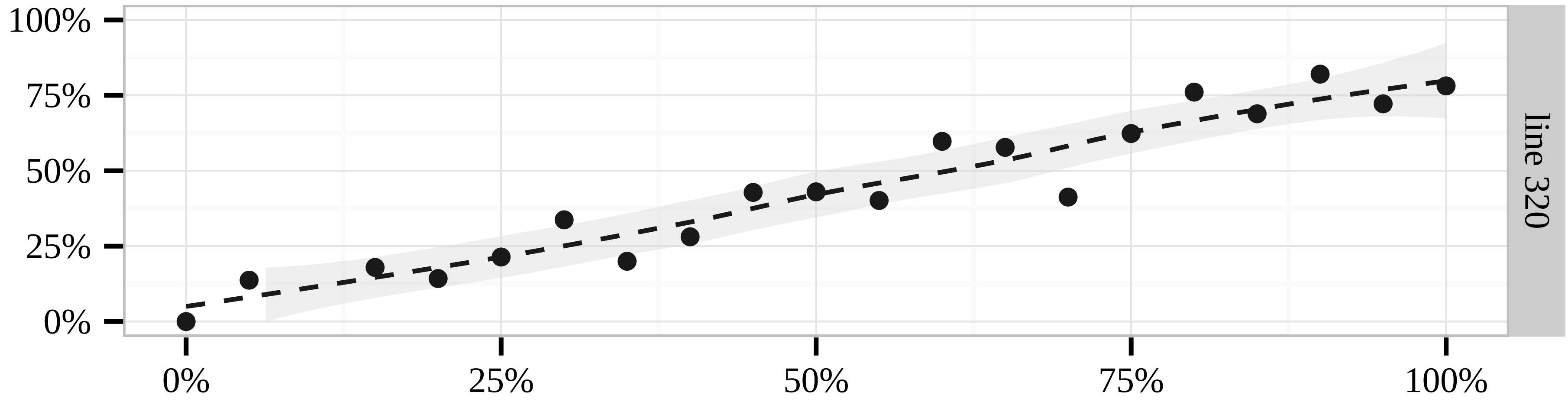
What did Coz predict?

ranking

Increased from 16 to 22 threads

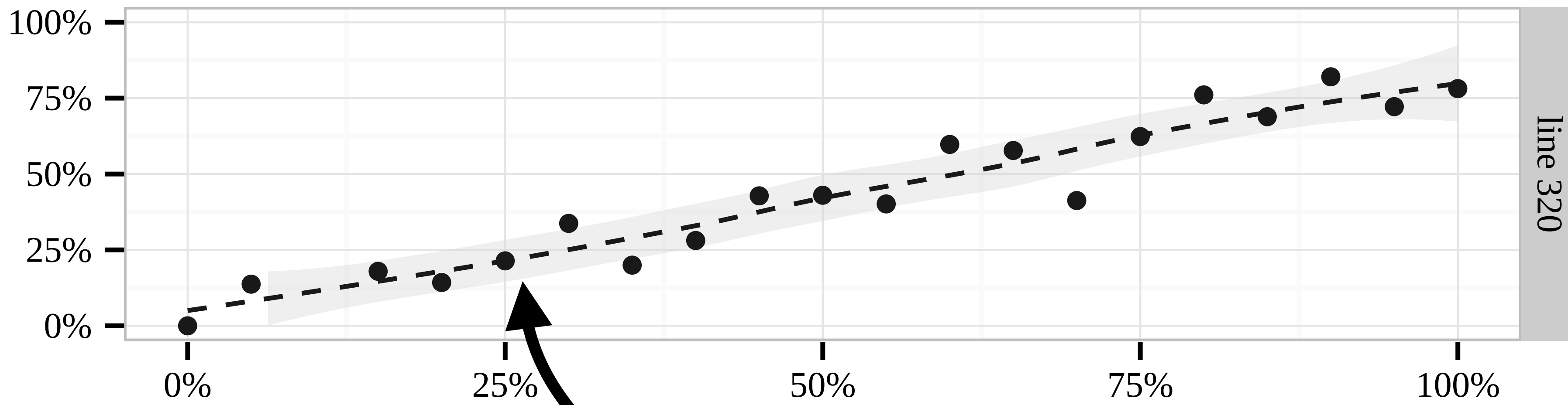
27% increase in ranking throughput

What did Coz predict?



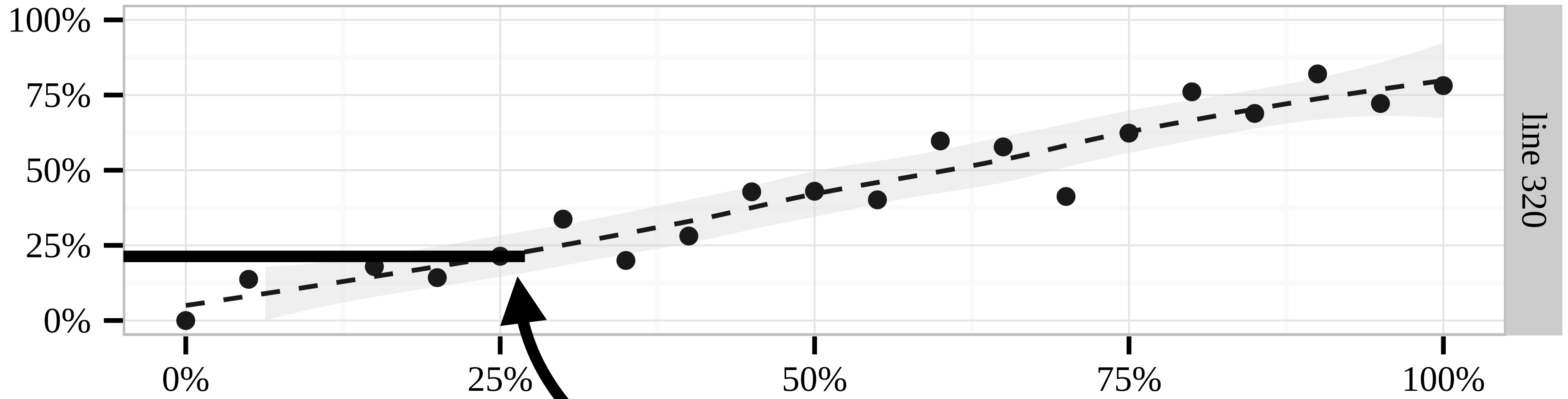
27% increase in ranking throughput

What did Coz predict?



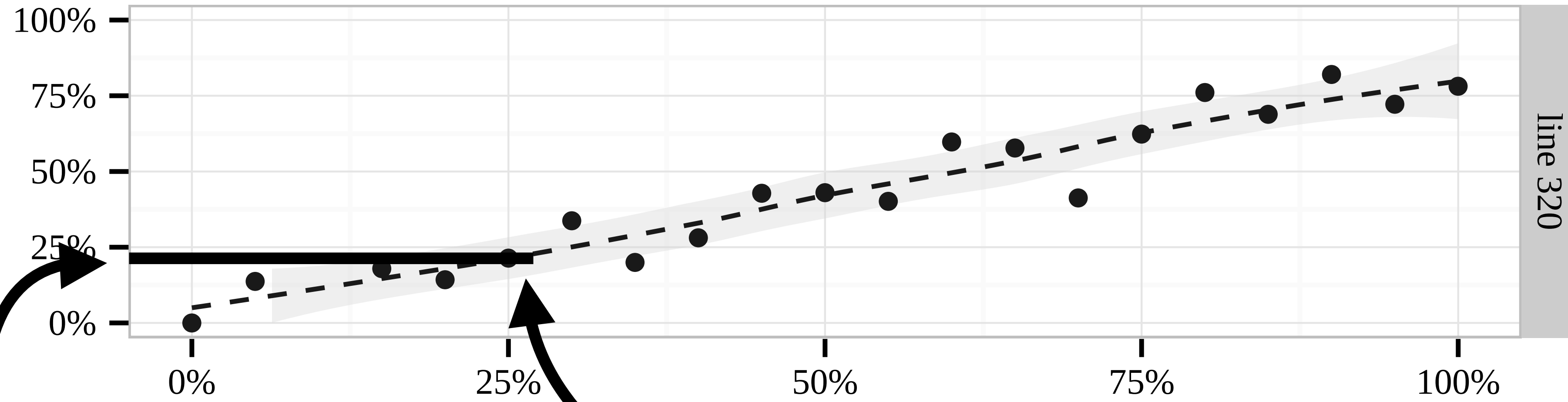
27% increase in ranking throughput

What did Coz predict?



27% increase in ranking throughput

What did Coz predict?

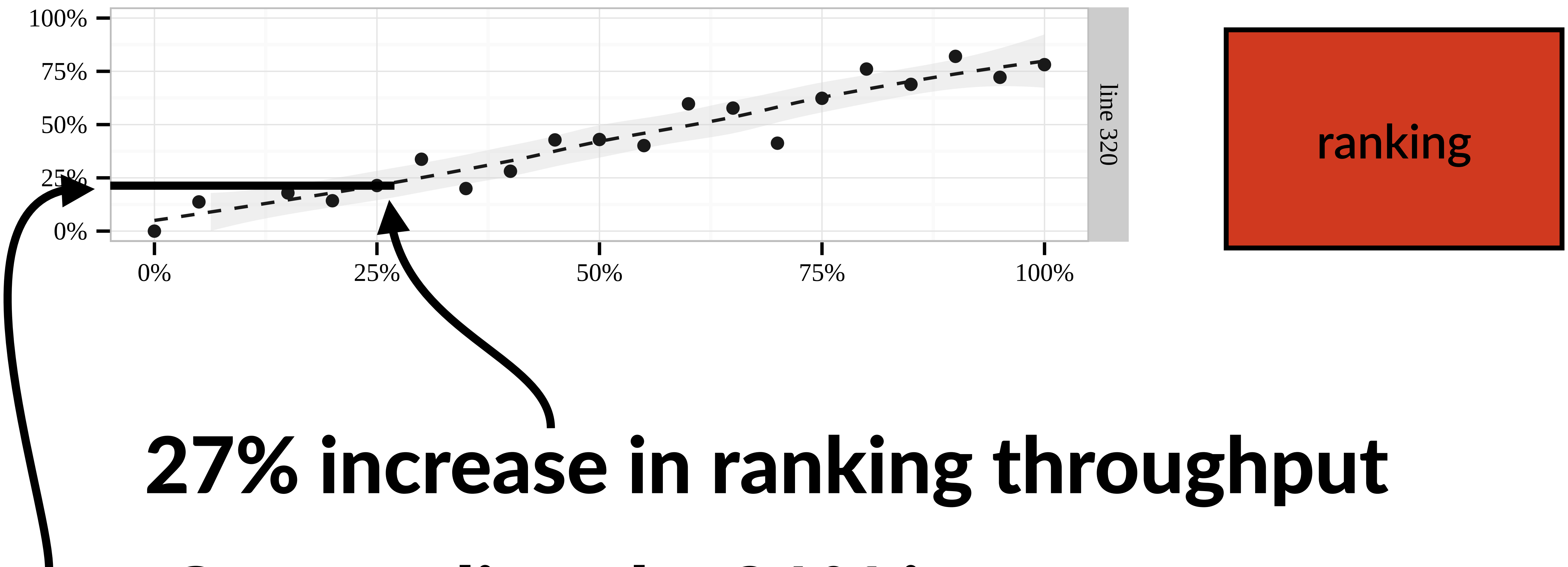


ranking

27% increase in ranking throughput

Coz predicted a 21% improvement

What did Coz predict?

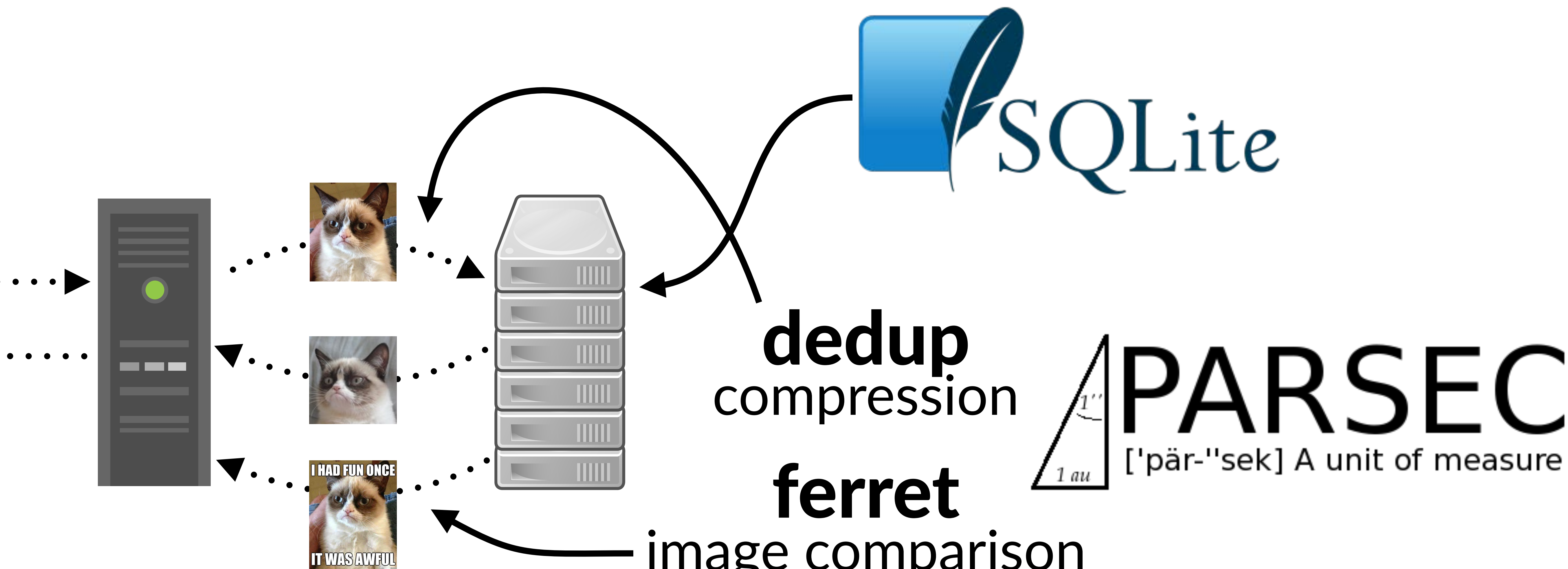


27% increase in ranking throughput

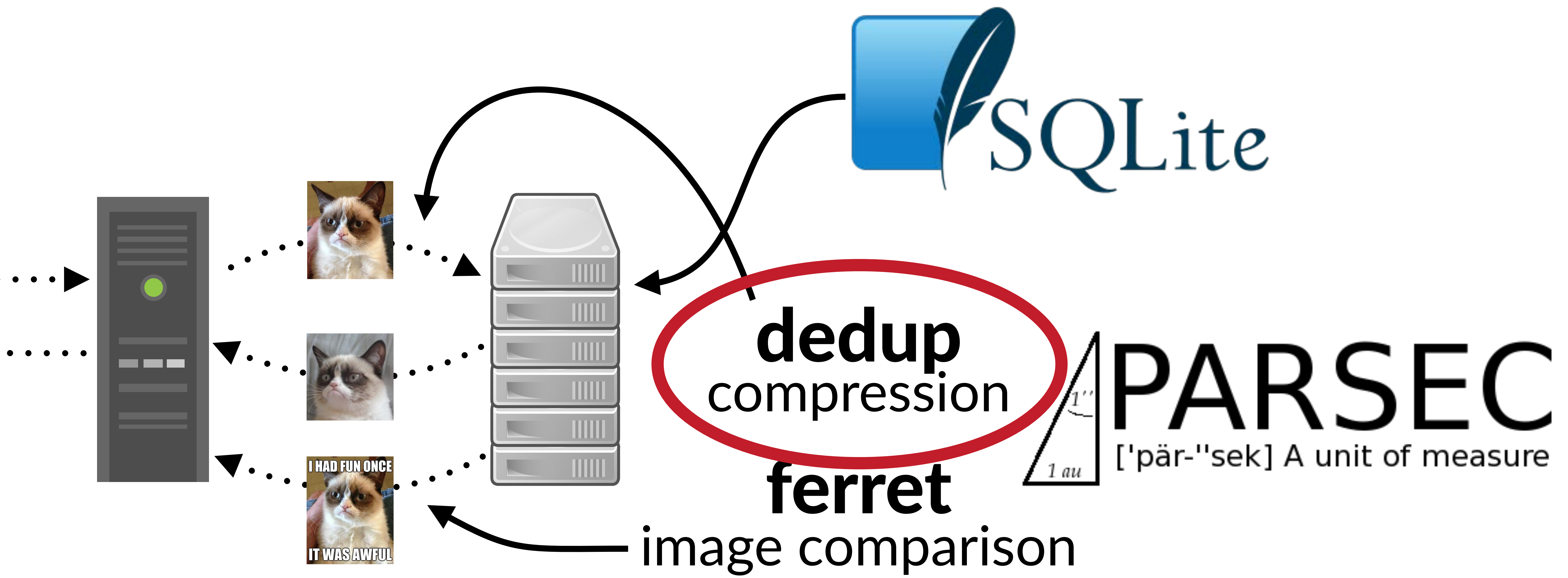
Coz predicted a 21% improvement

Exactly what we observed

Using Causal Profiling on Ogle



Using Causal Profiling on Ogle

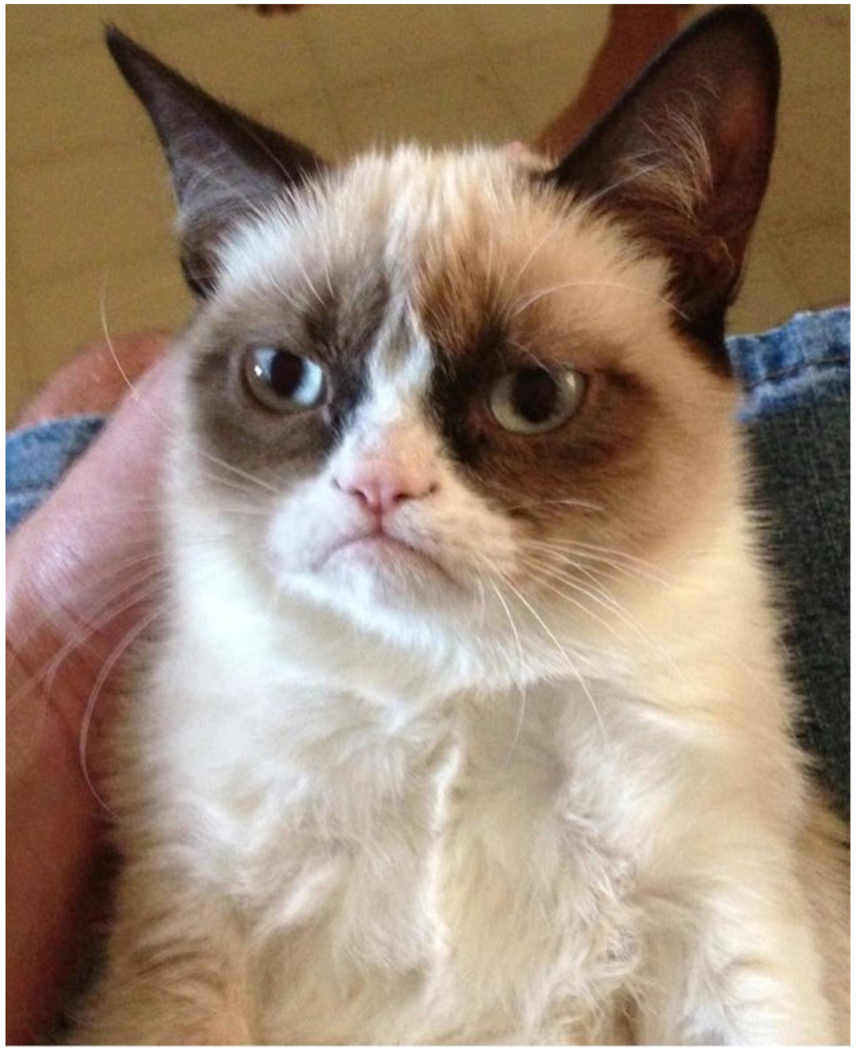


Dedup

Compression via deduplication

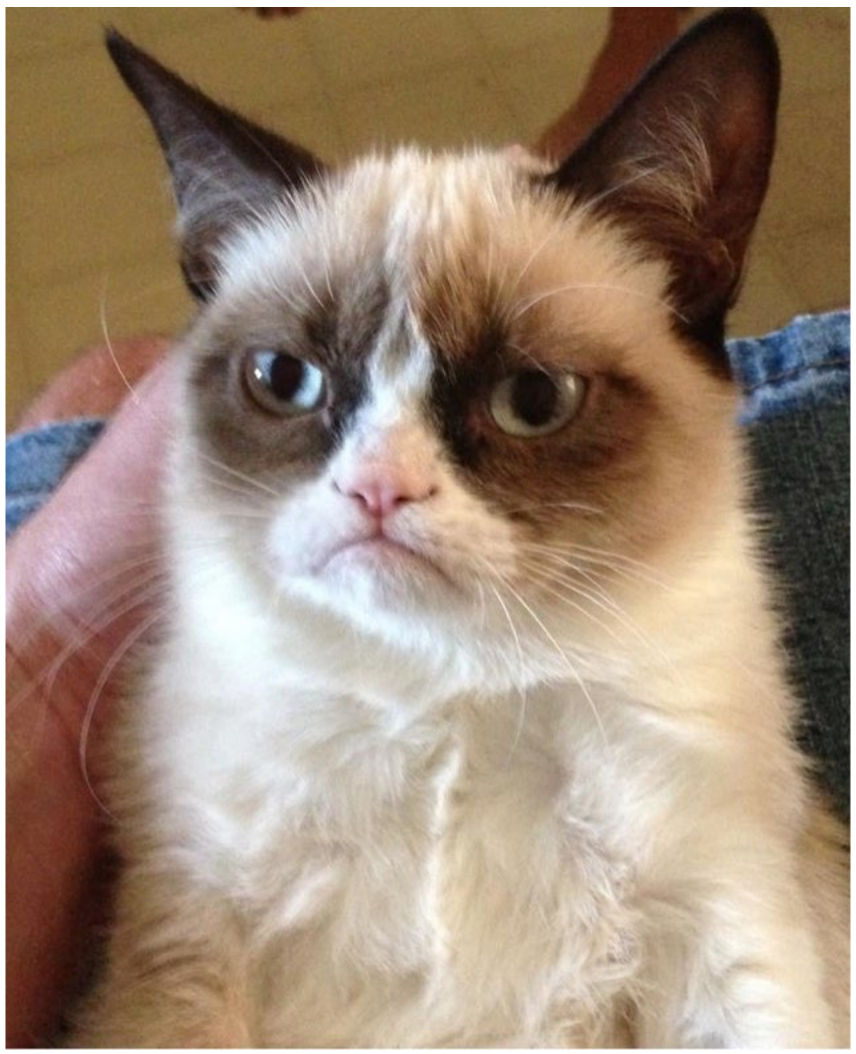
Dedup

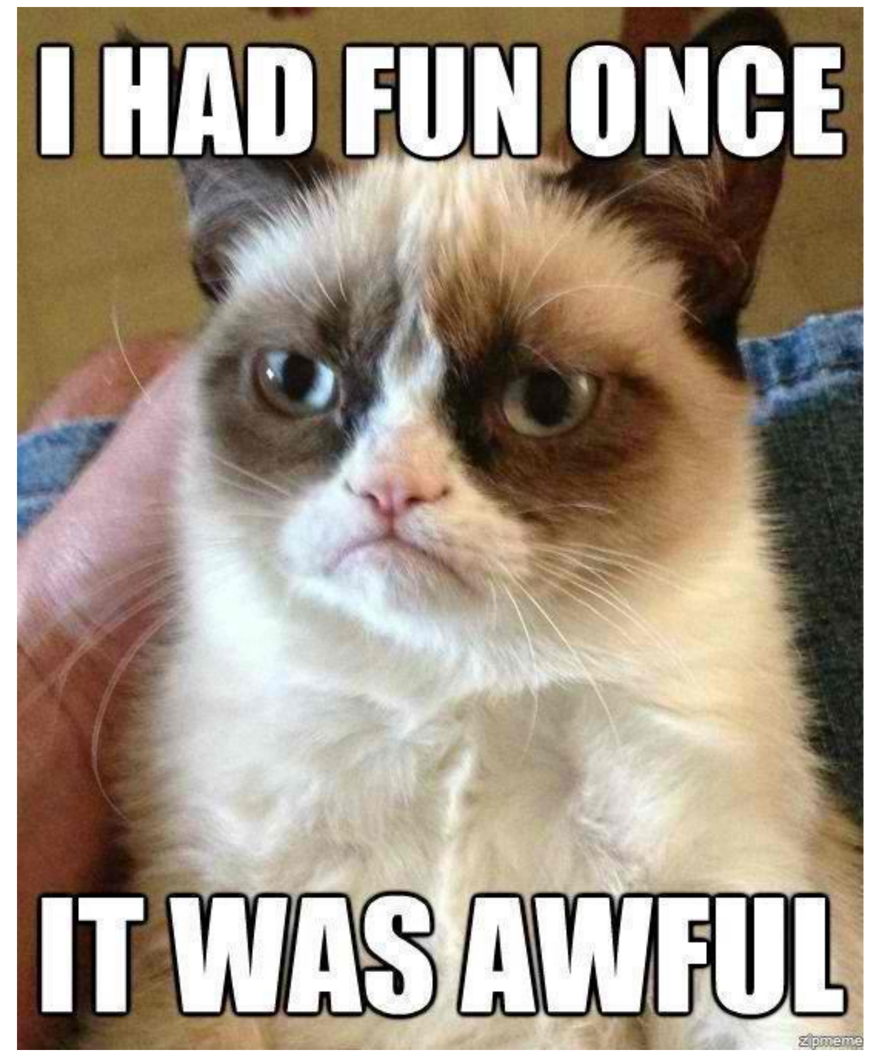
Compression via deduplication



Dedup

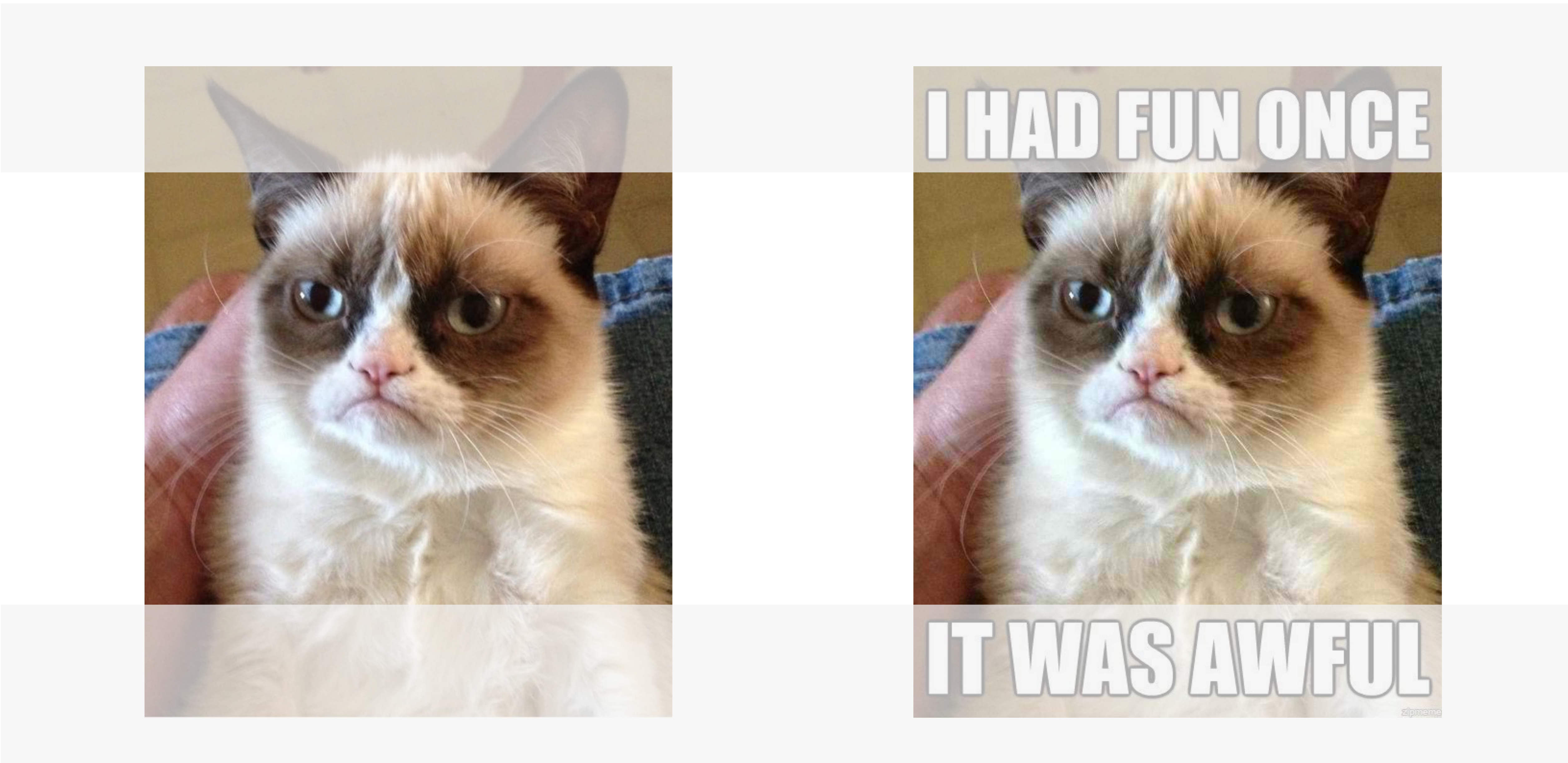
Compression via deduplication





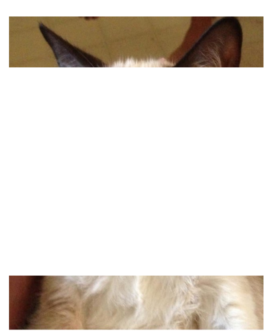
Dedup

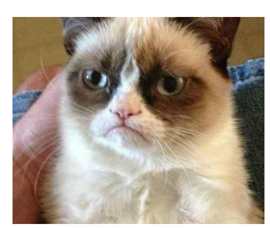
Compression via deduplication



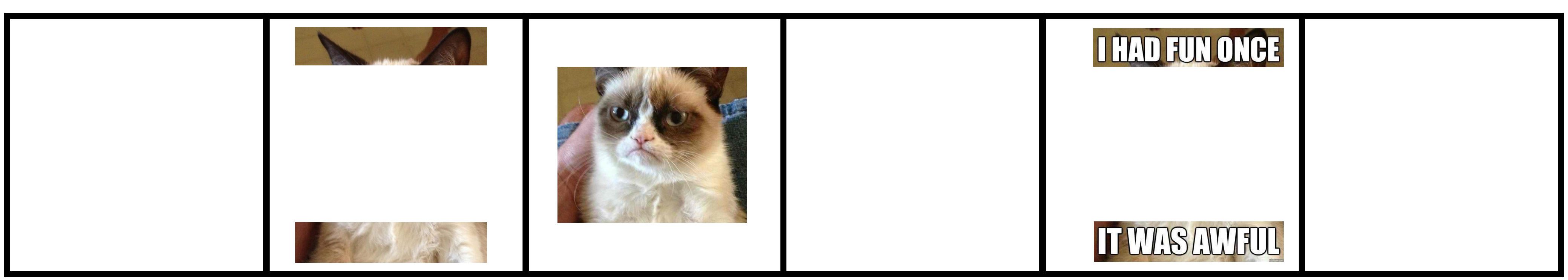
Dedup

Compression via deduplication



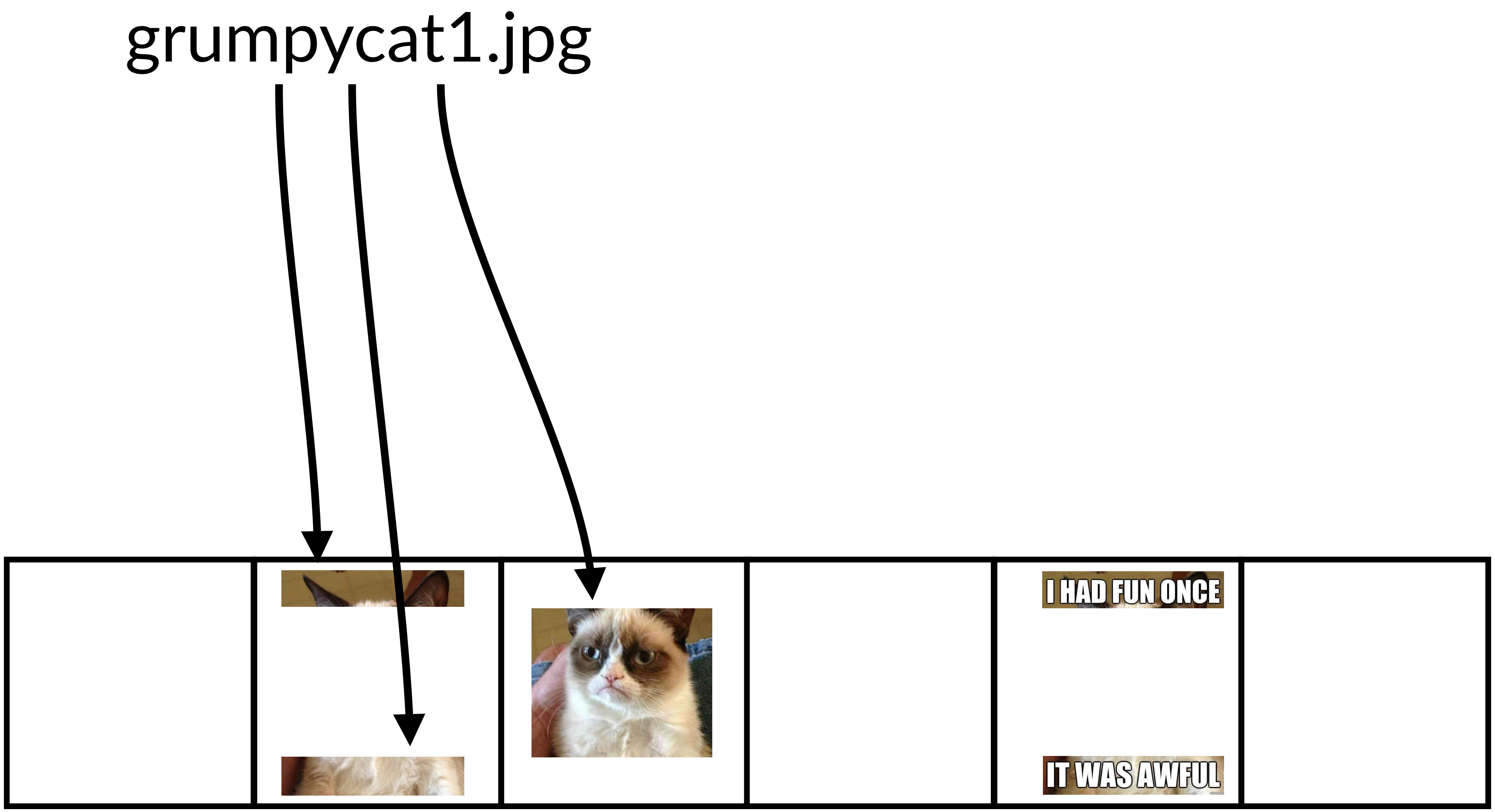


Dedup
Compression via deduplication



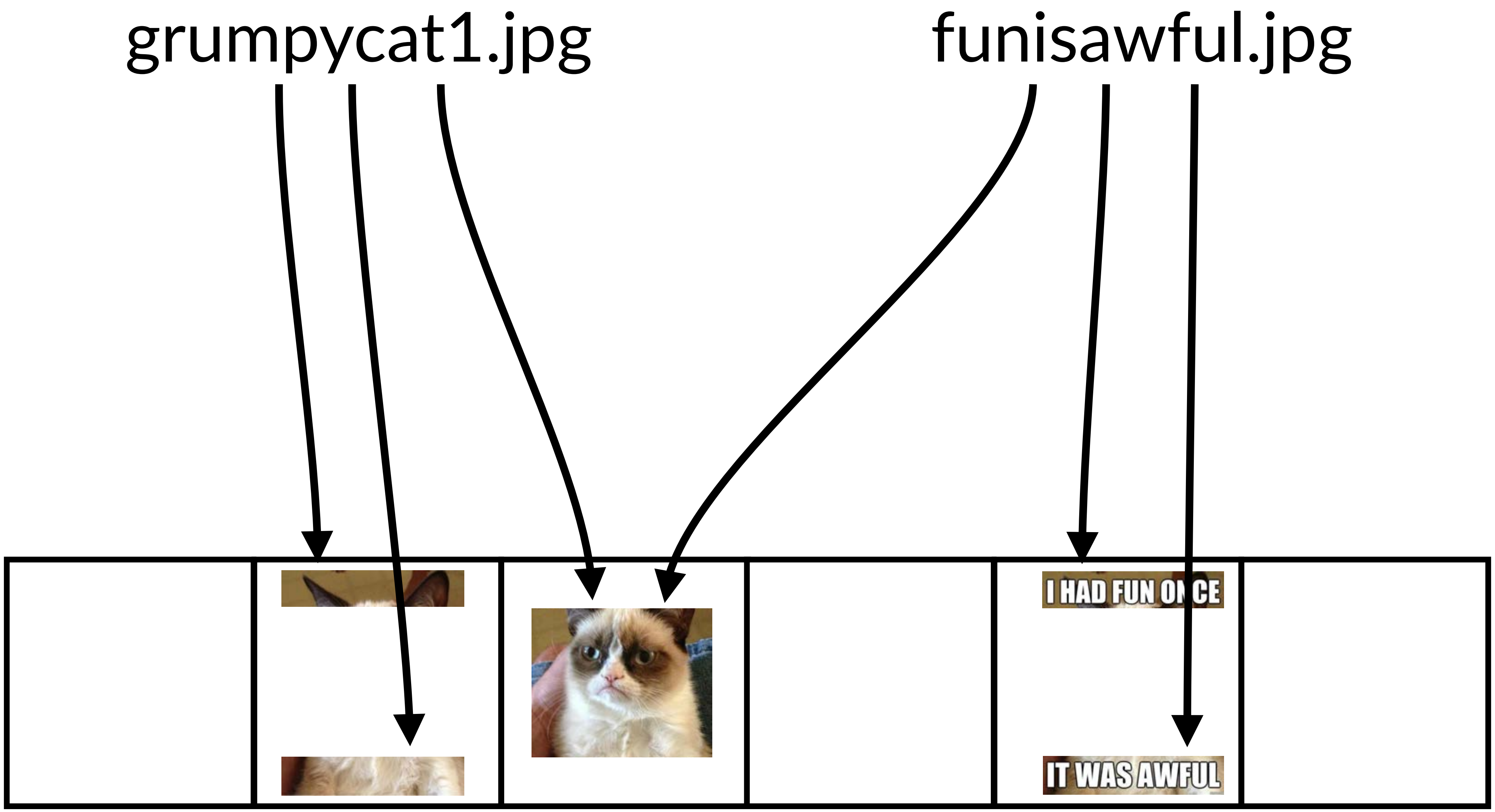
Dedup

Compression via deduplication



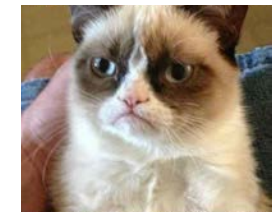
Dedup

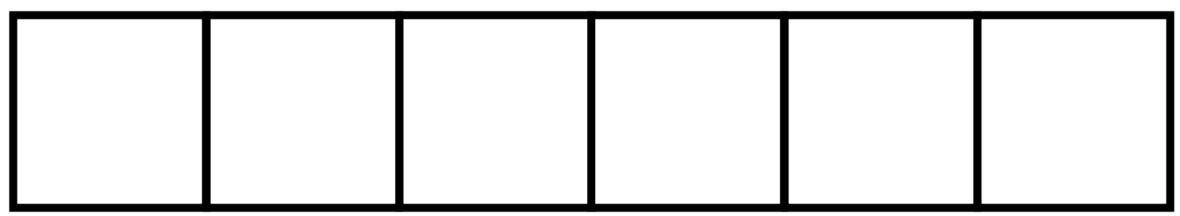
Compression via deduplication



Dedup

Compression via deduplication

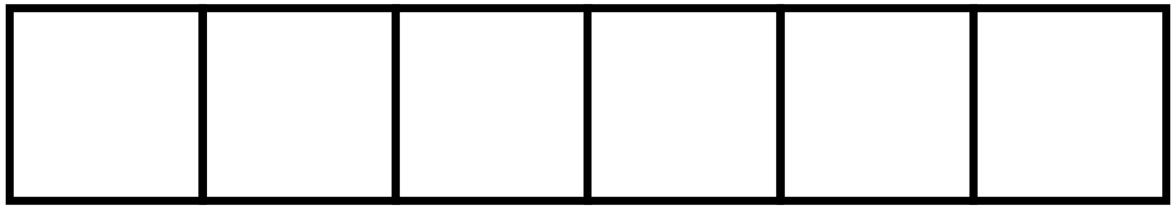




Dedup

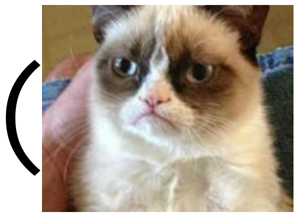
Compression via deduplication

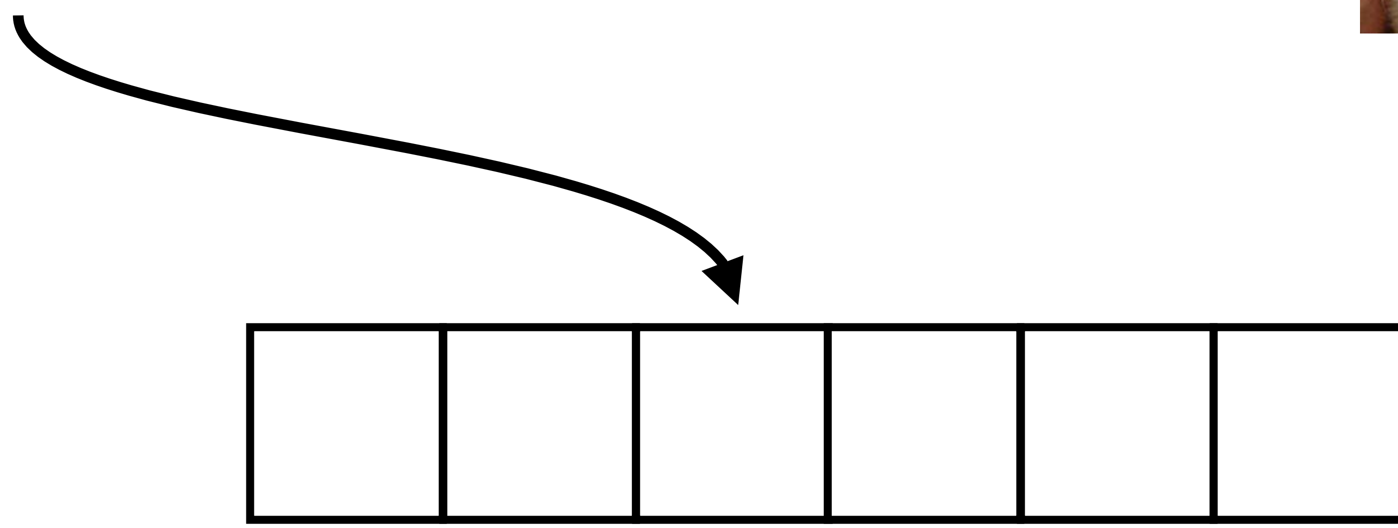
`hash_function(``)`



Dedup

Compression via deduplication

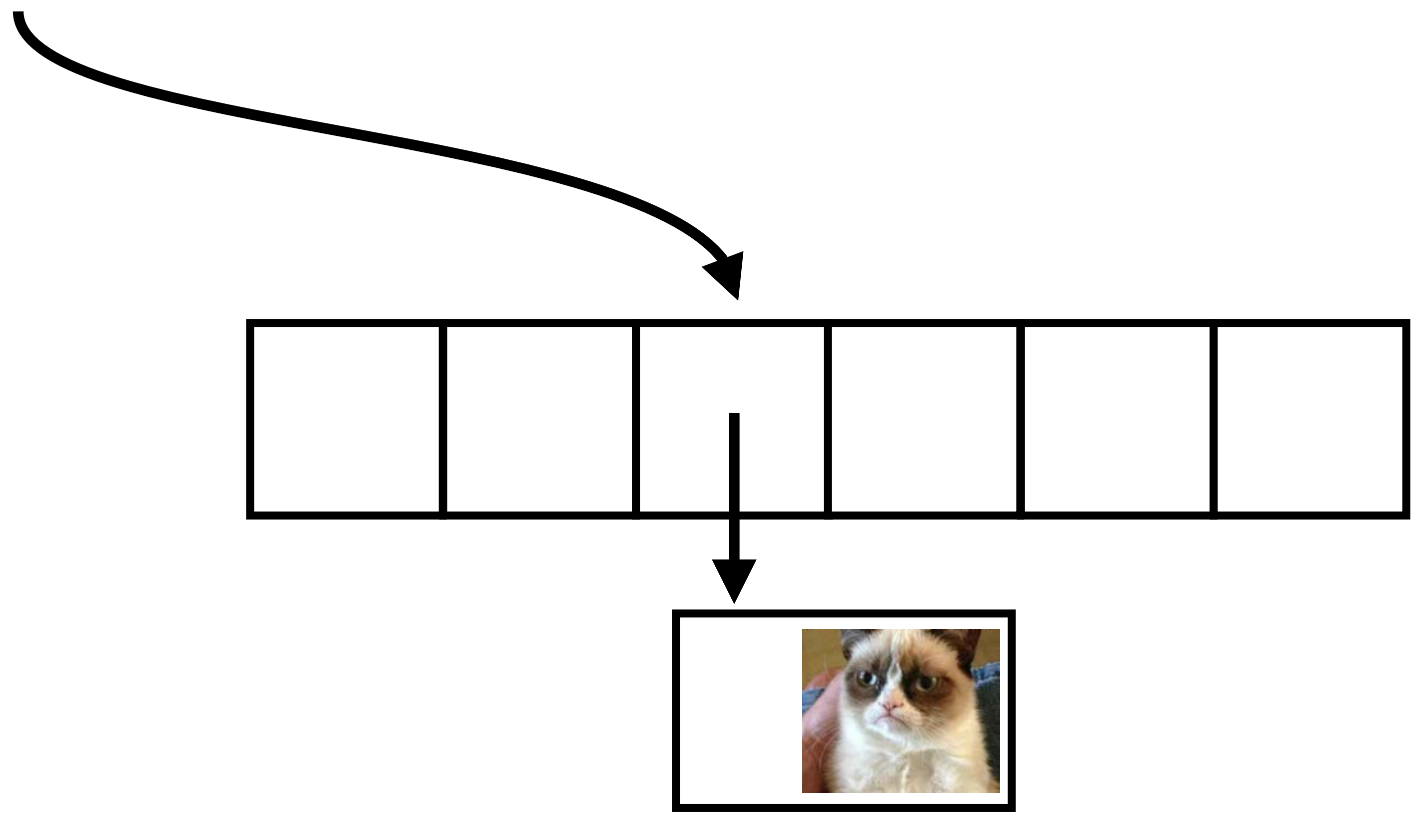
`i = hash_function(`  `)`



Dedup

Compression via deduplication

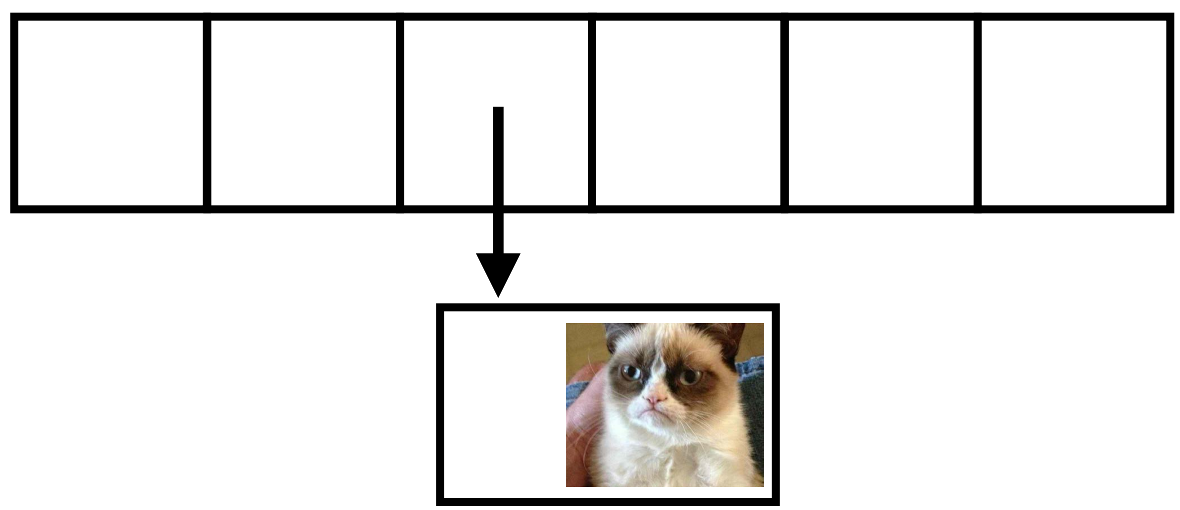
```
i = hash_function(   )
```



Dedup

Compression via deduplication

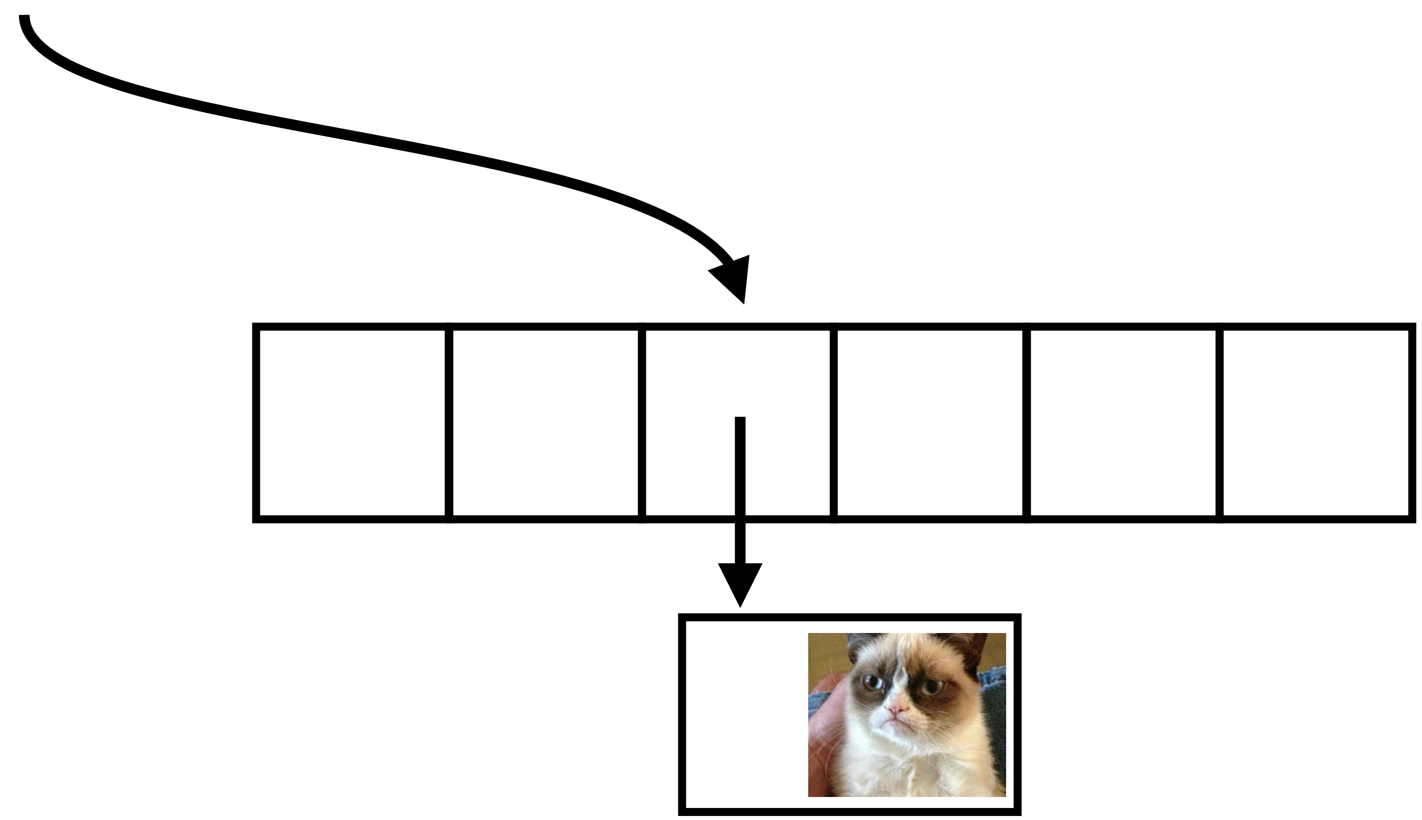
`hash_function(`  `)`



Dedup

Compression via deduplication

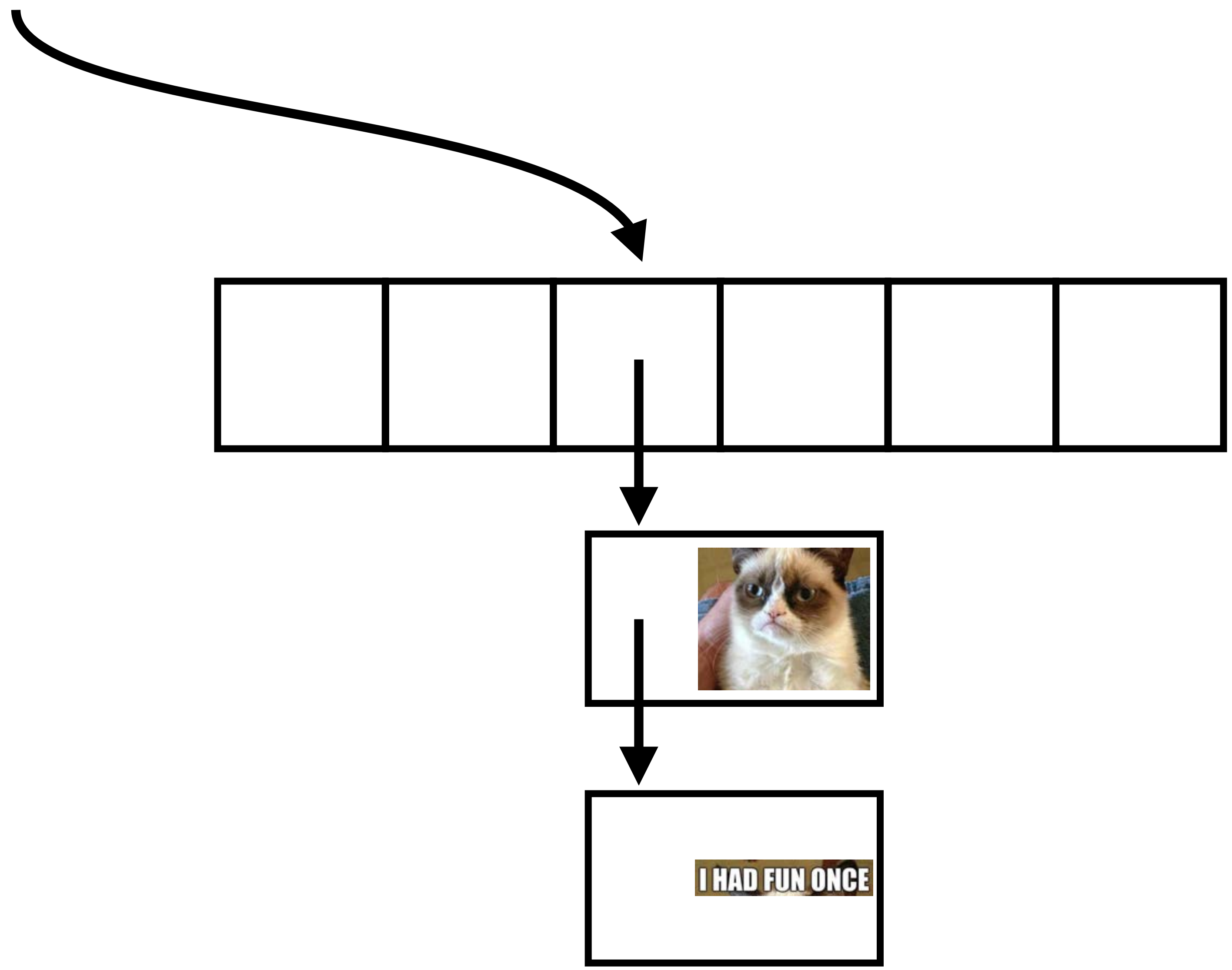
`i = hash_function(`  `)`



Dedup

Compression via deduplication

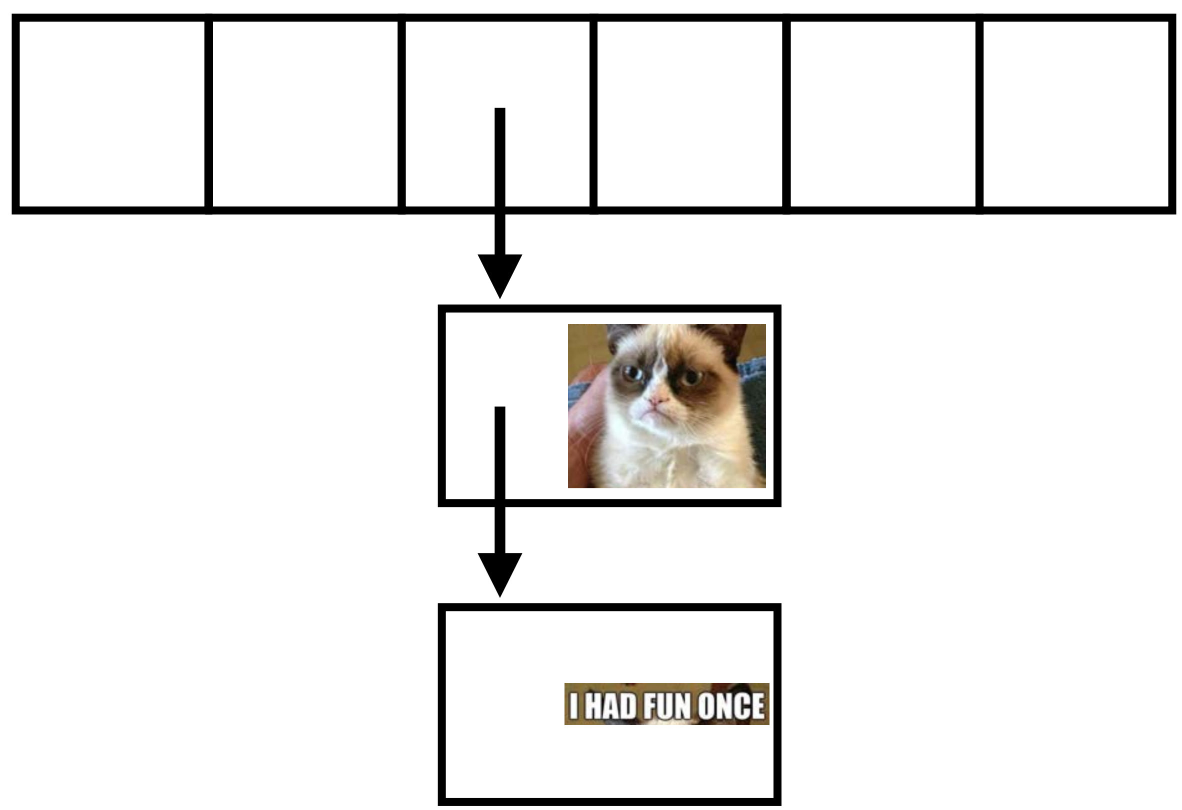
```
i = hash_function(        )
```



Dedup

Compression via deduplication

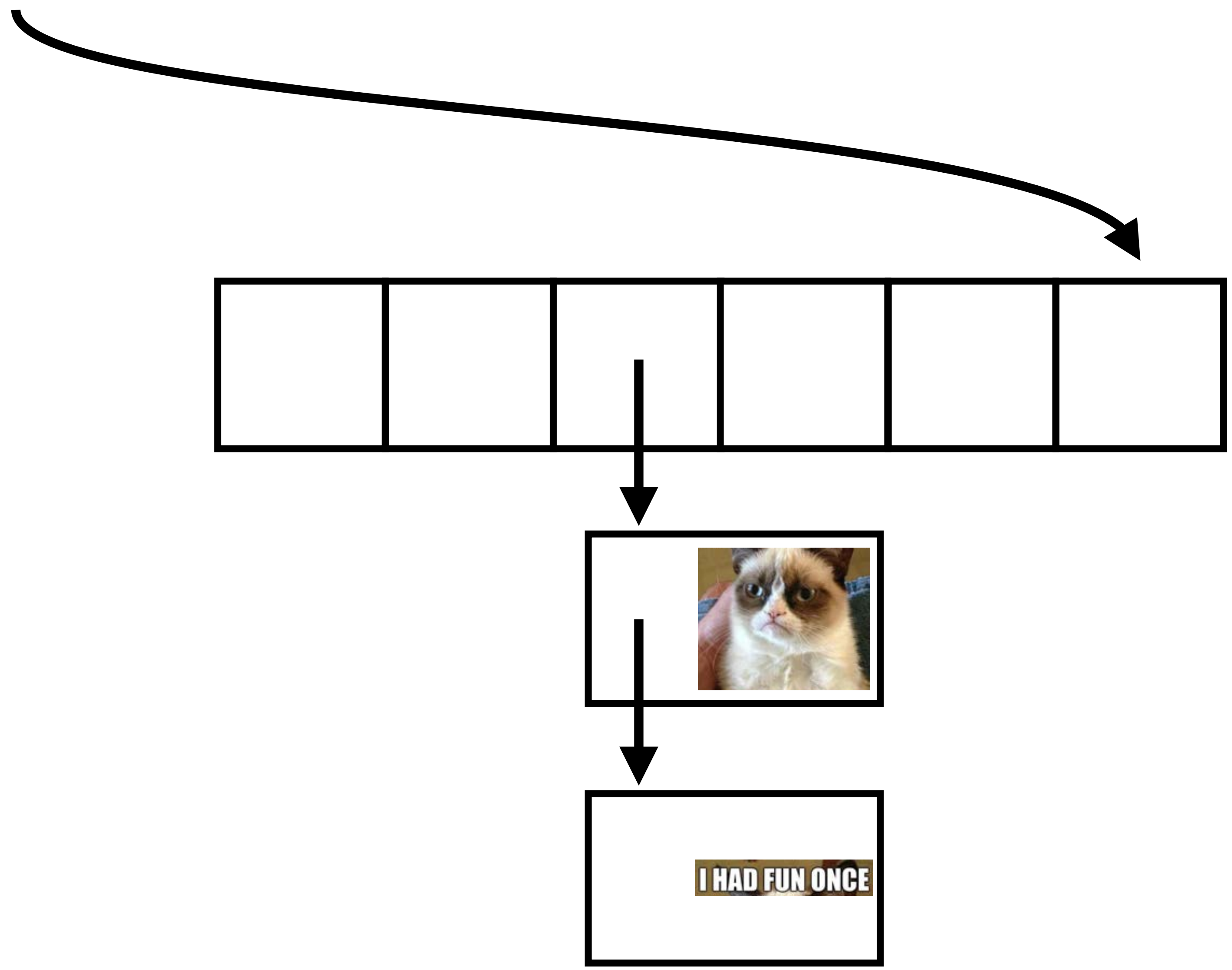
`hash_function(`  `)`



Dedup

Compression via deduplication

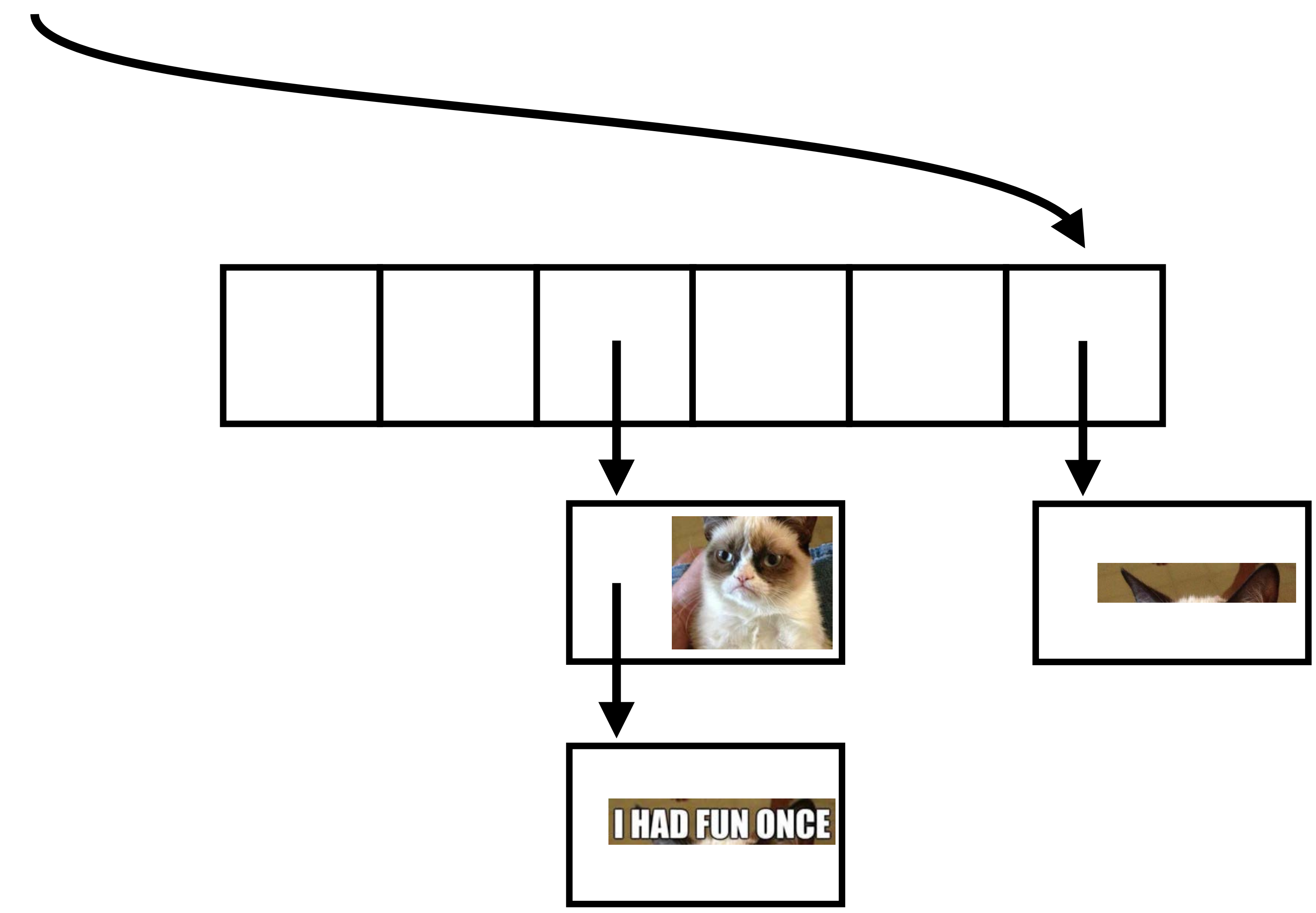
`i = hash_function(`  `)`



Dedup

Compression via deduplication

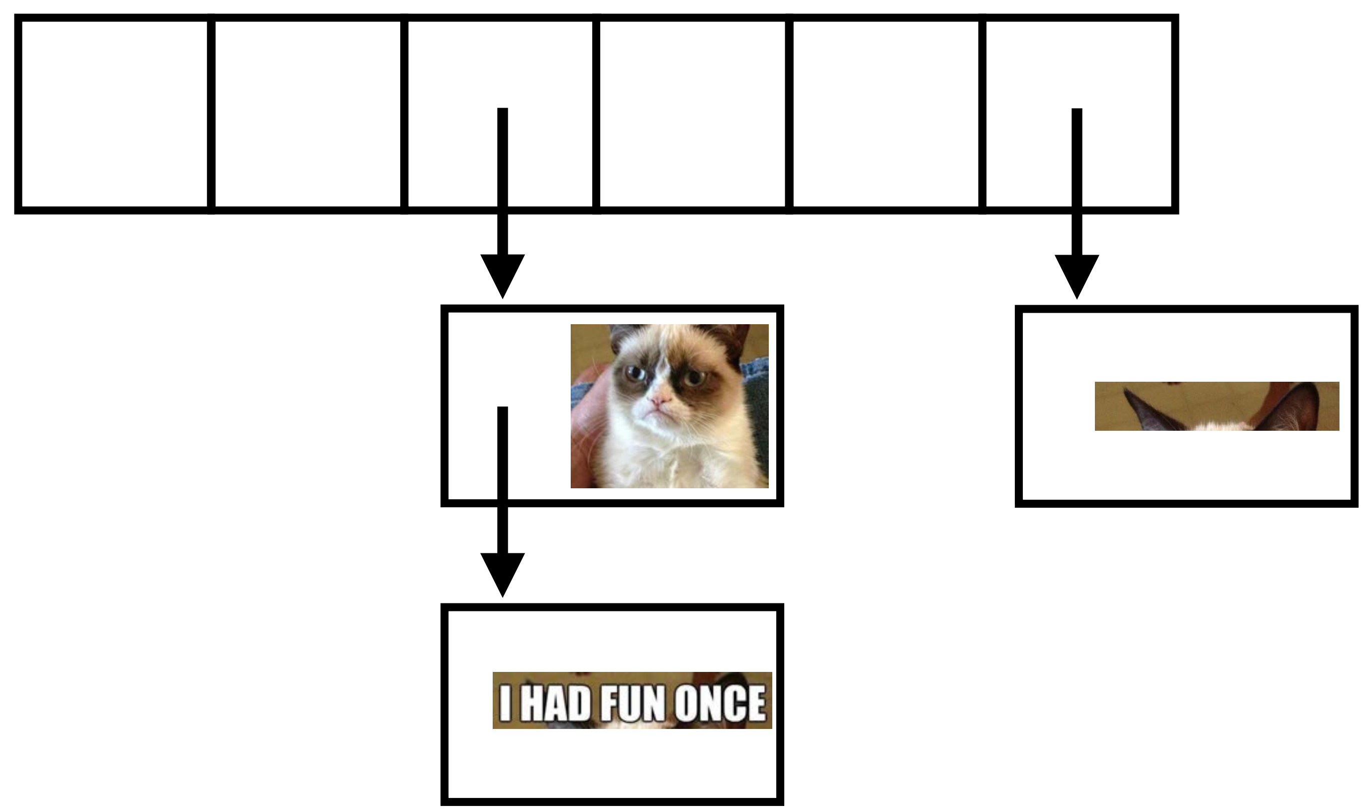
`i = hash_function(          )`



Dedup

Compression via deduplication

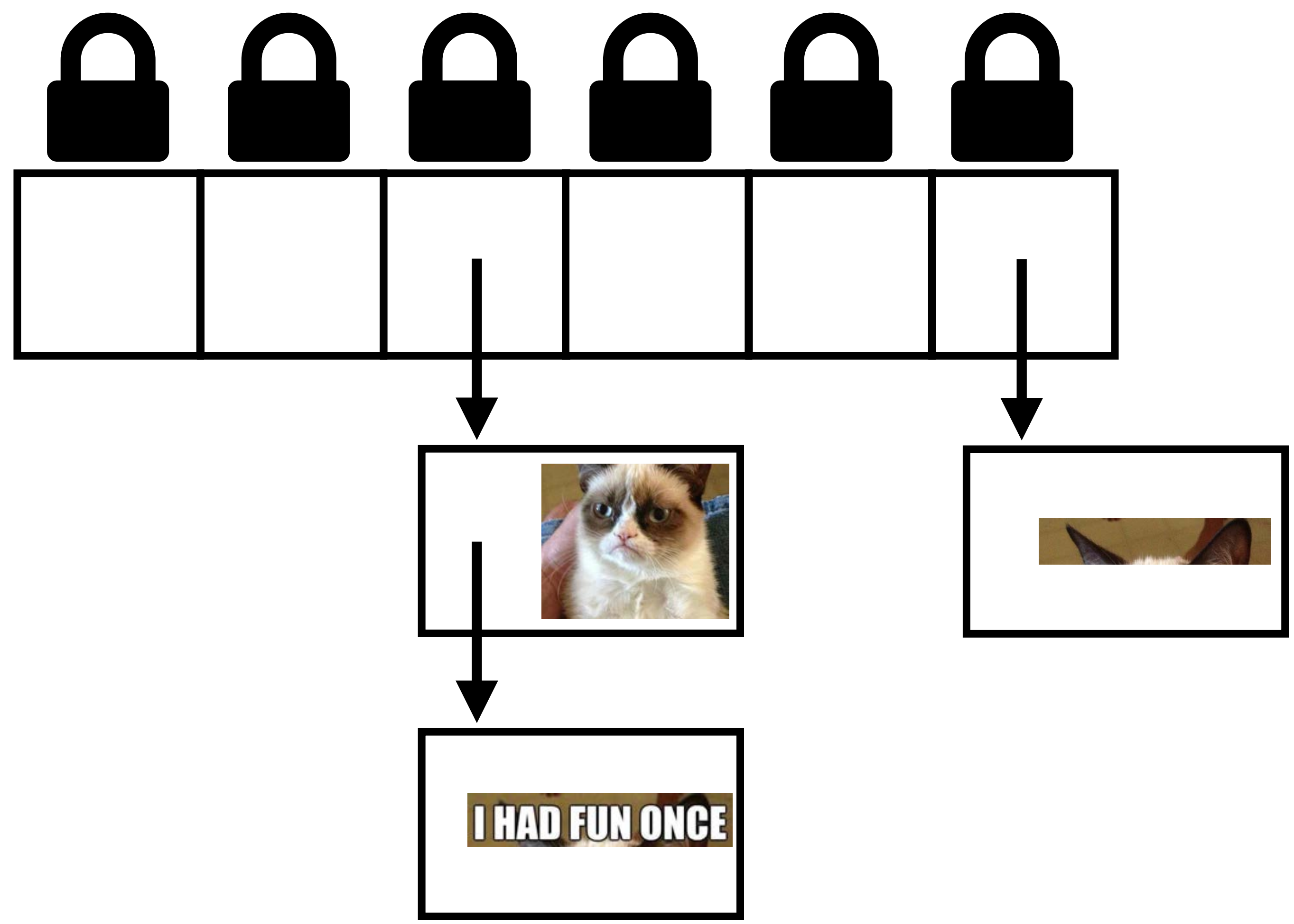
Hash table is accessed concurrently by many threads



Dedup

Compression via deduplication

Hash table is accessed concurrently by many threads

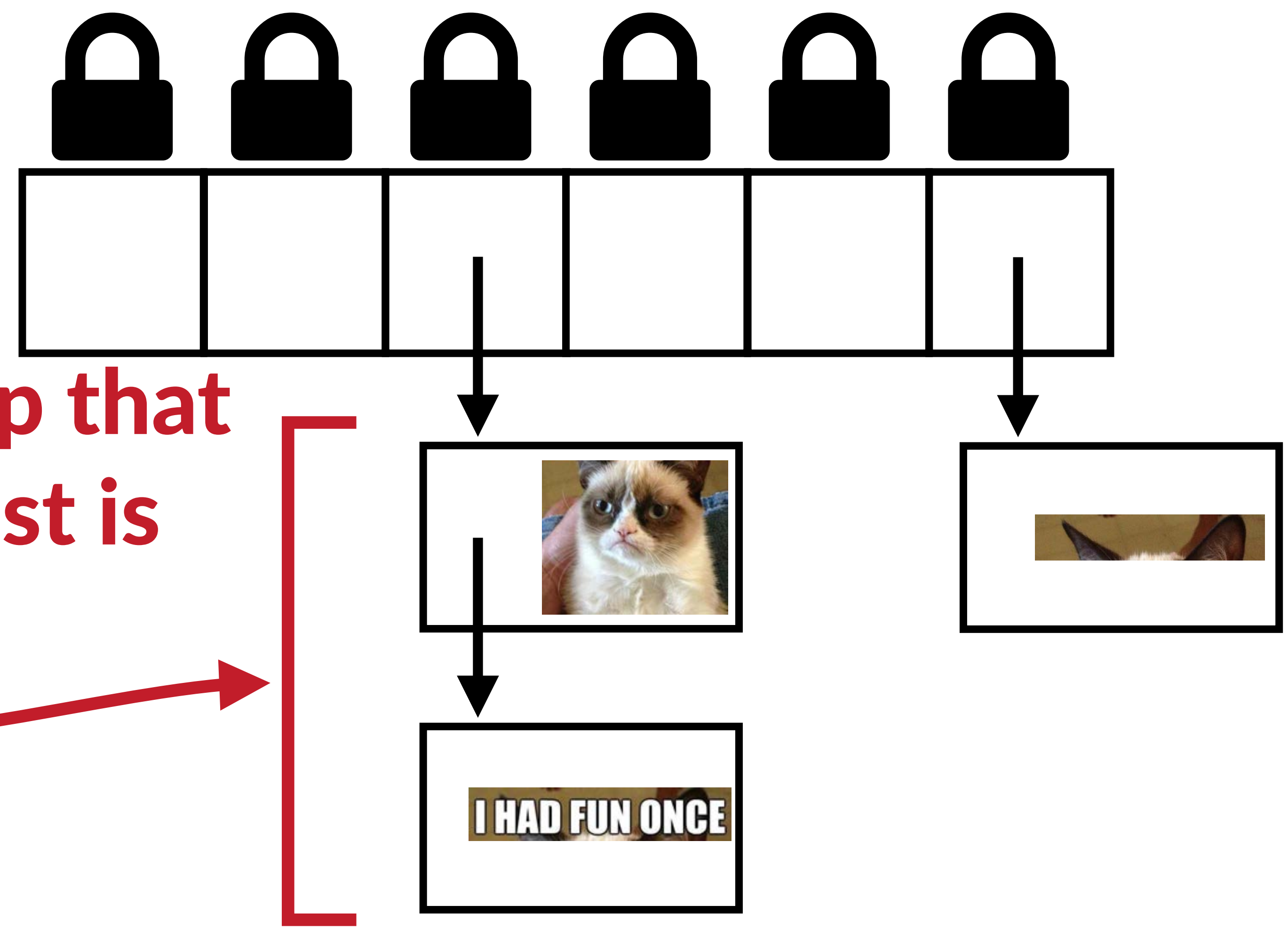


Dedup

Compression via deduplication

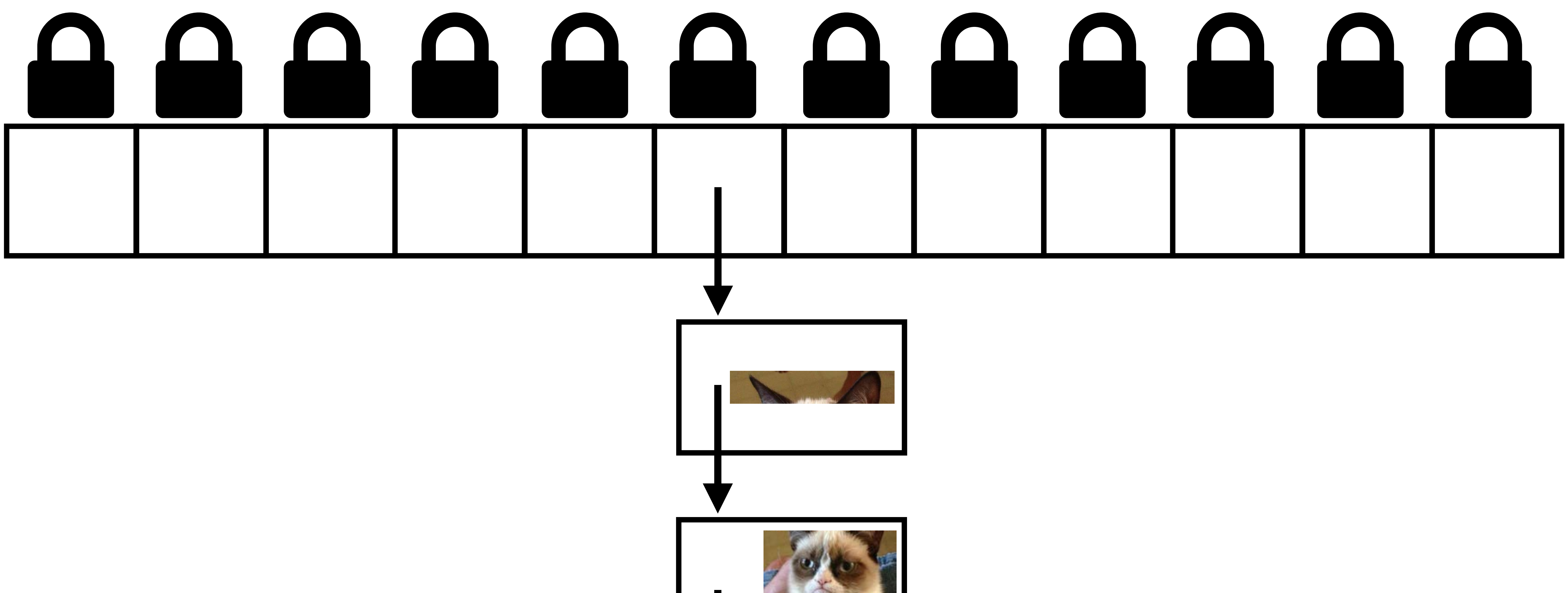
Hash table is accessed concurrently by many threads

Coz says the loop that accesses this list is important



Dedup

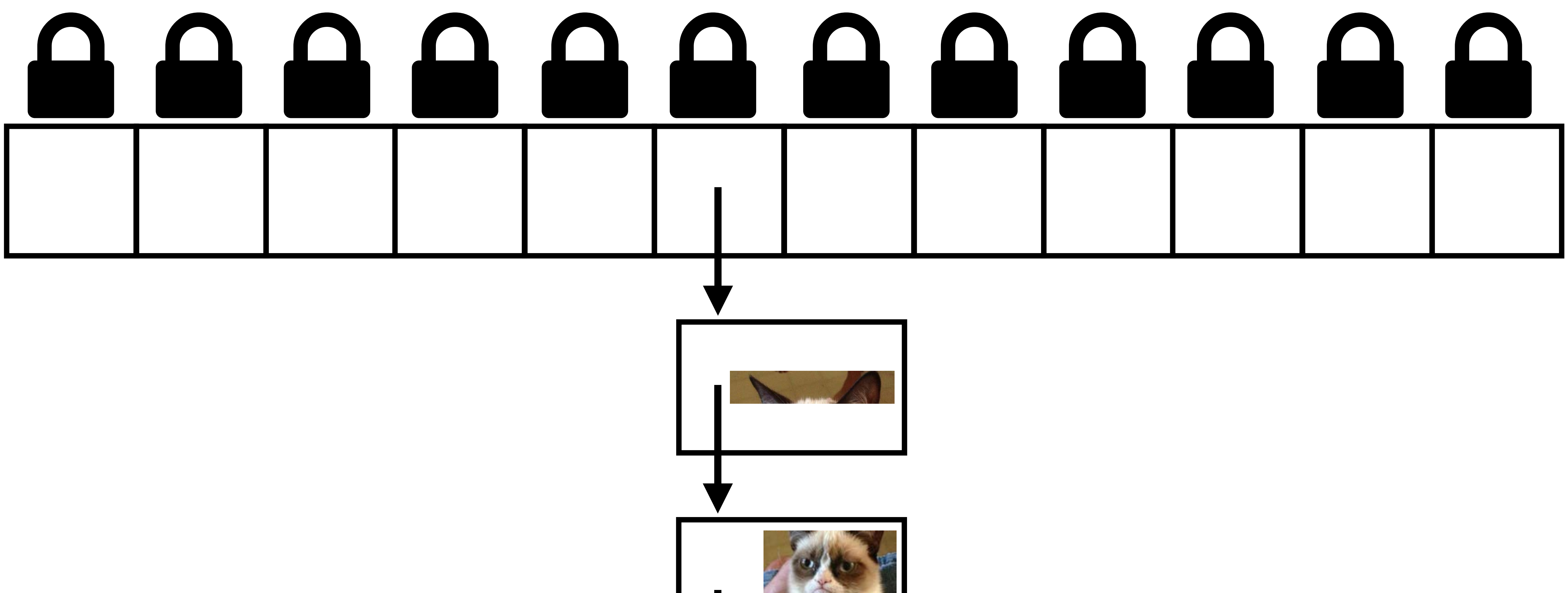
Compression via deduplication



Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

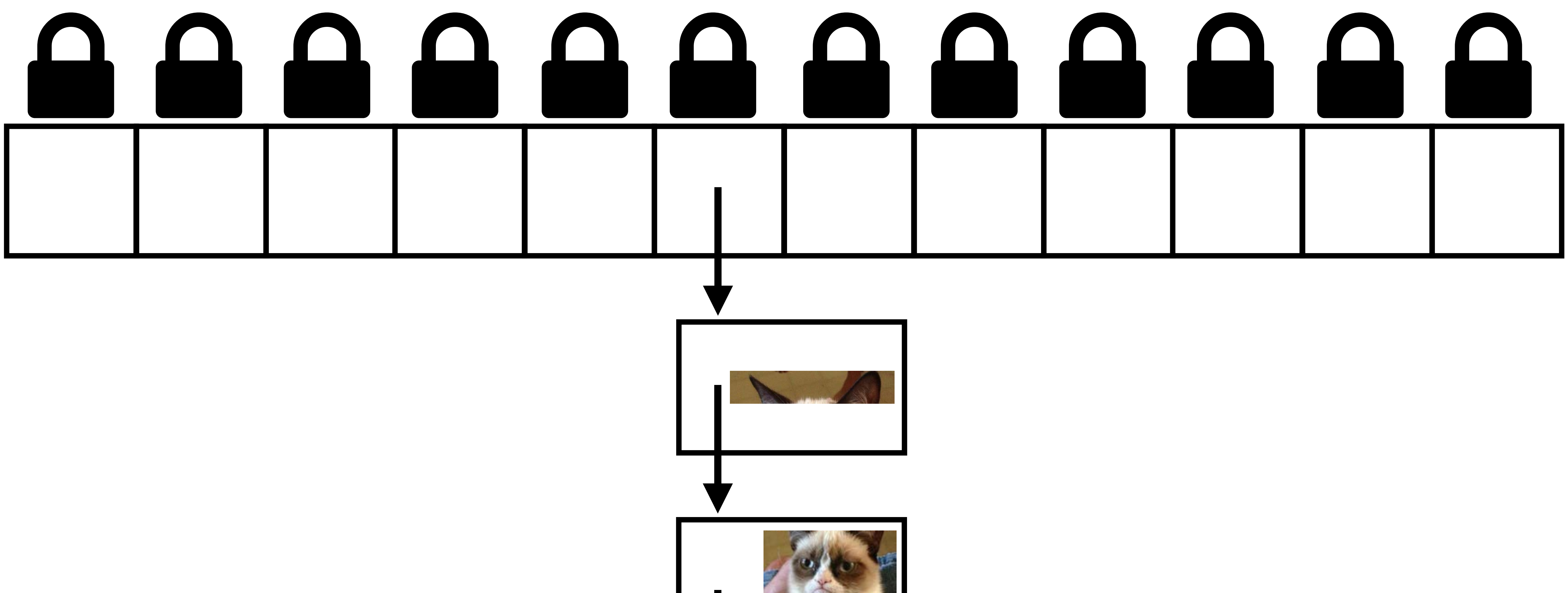


Dedup

Compression via deduplication

More hash buckets should lead to fewer collisions

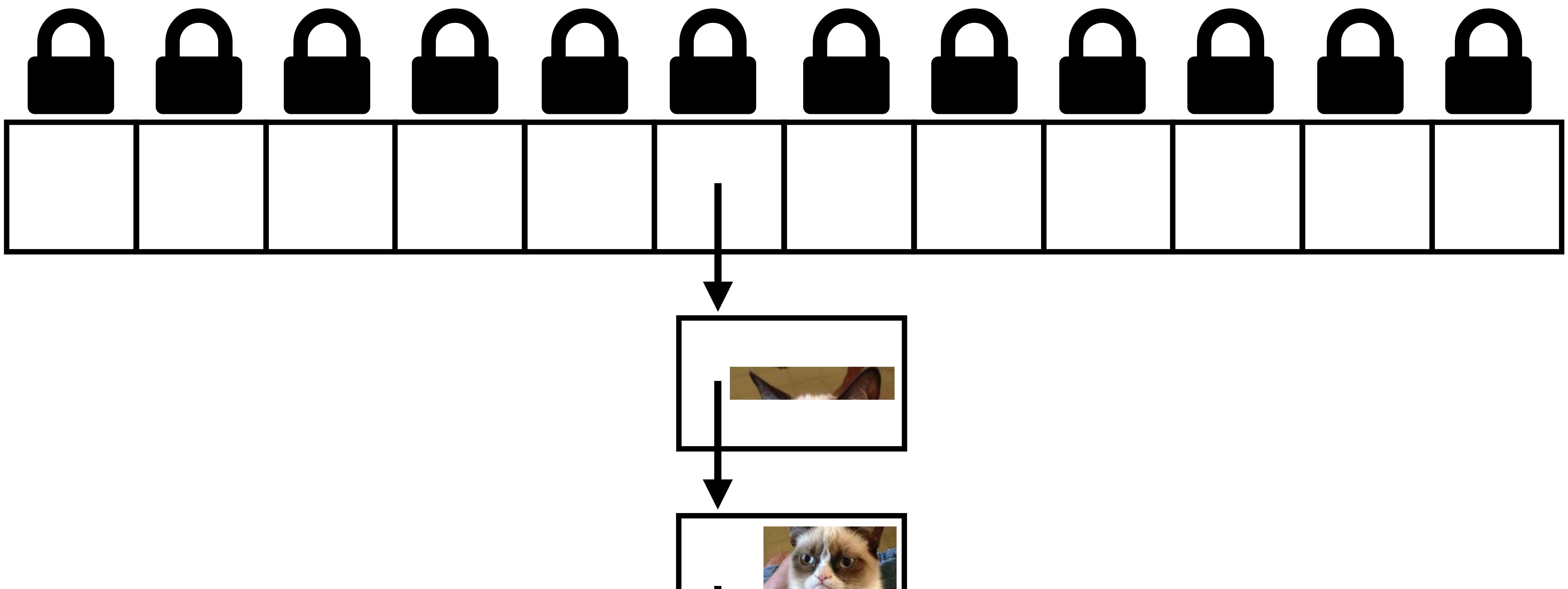
No performance improvement



Dedup

Compression via deduplication

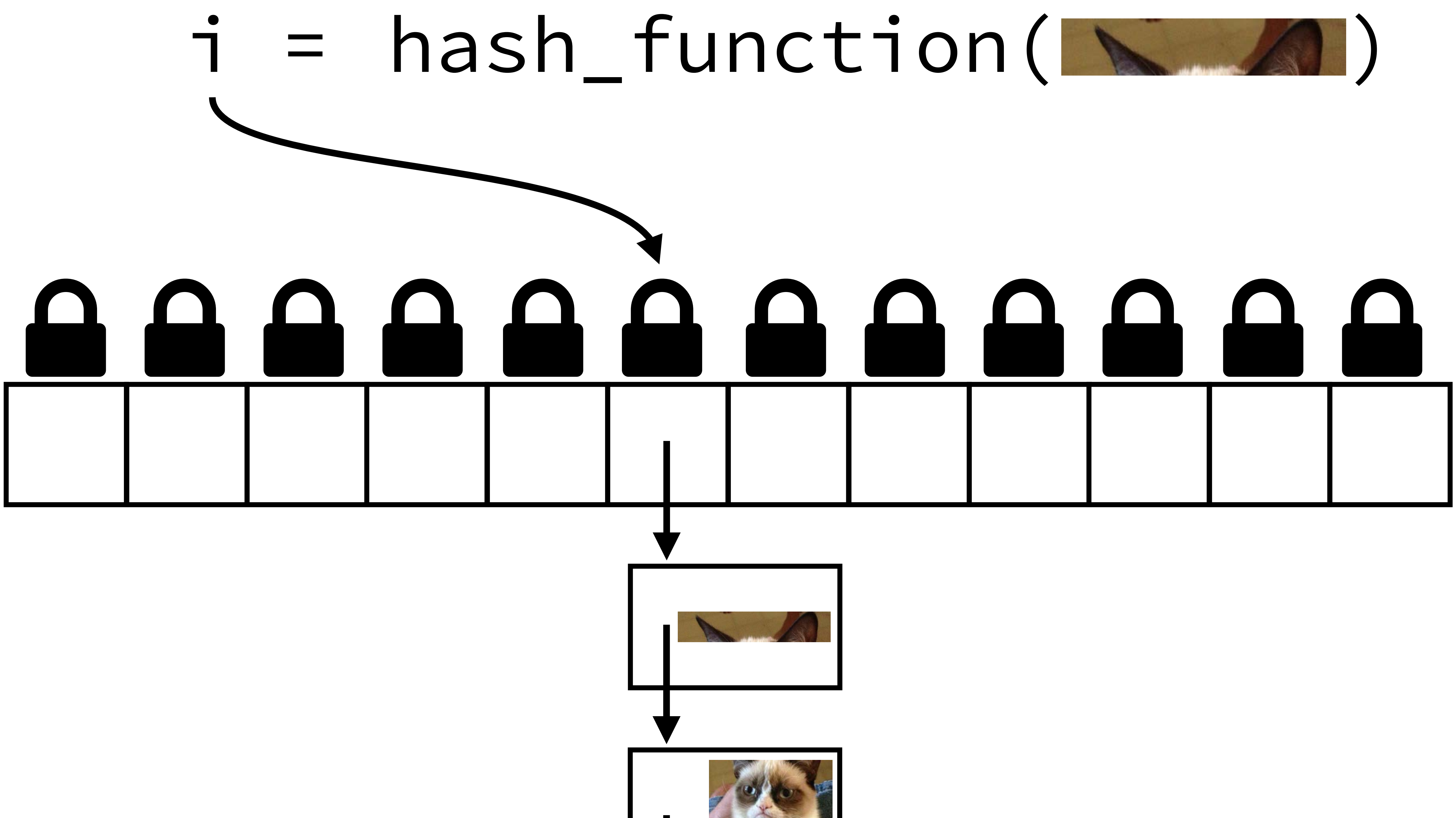
What else could be causing collisions?



Dedup

Compression via deduplication

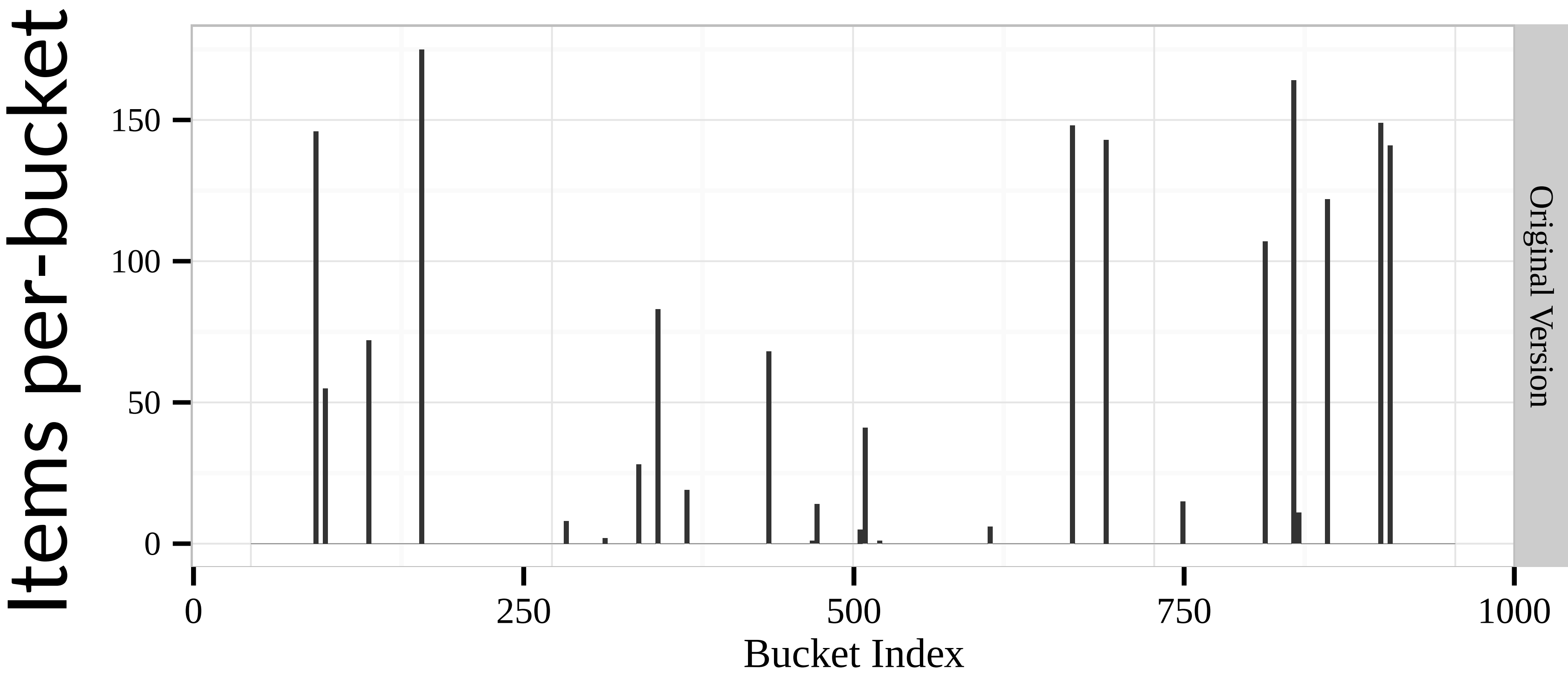
What else could be causing collisions?



Dedup

Compression via deduplication

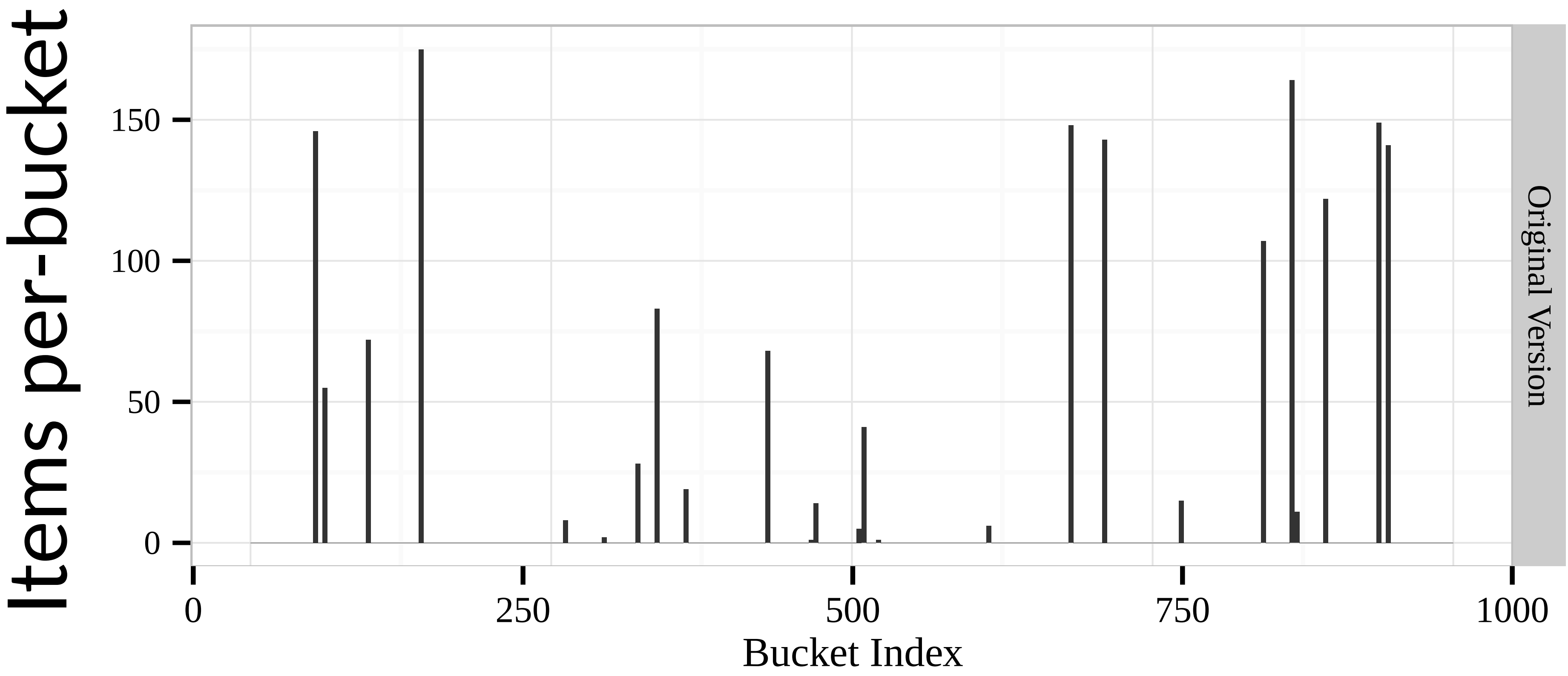
Horrible hash function!



Dedup

Compression via deduplication

Horrible hash function!

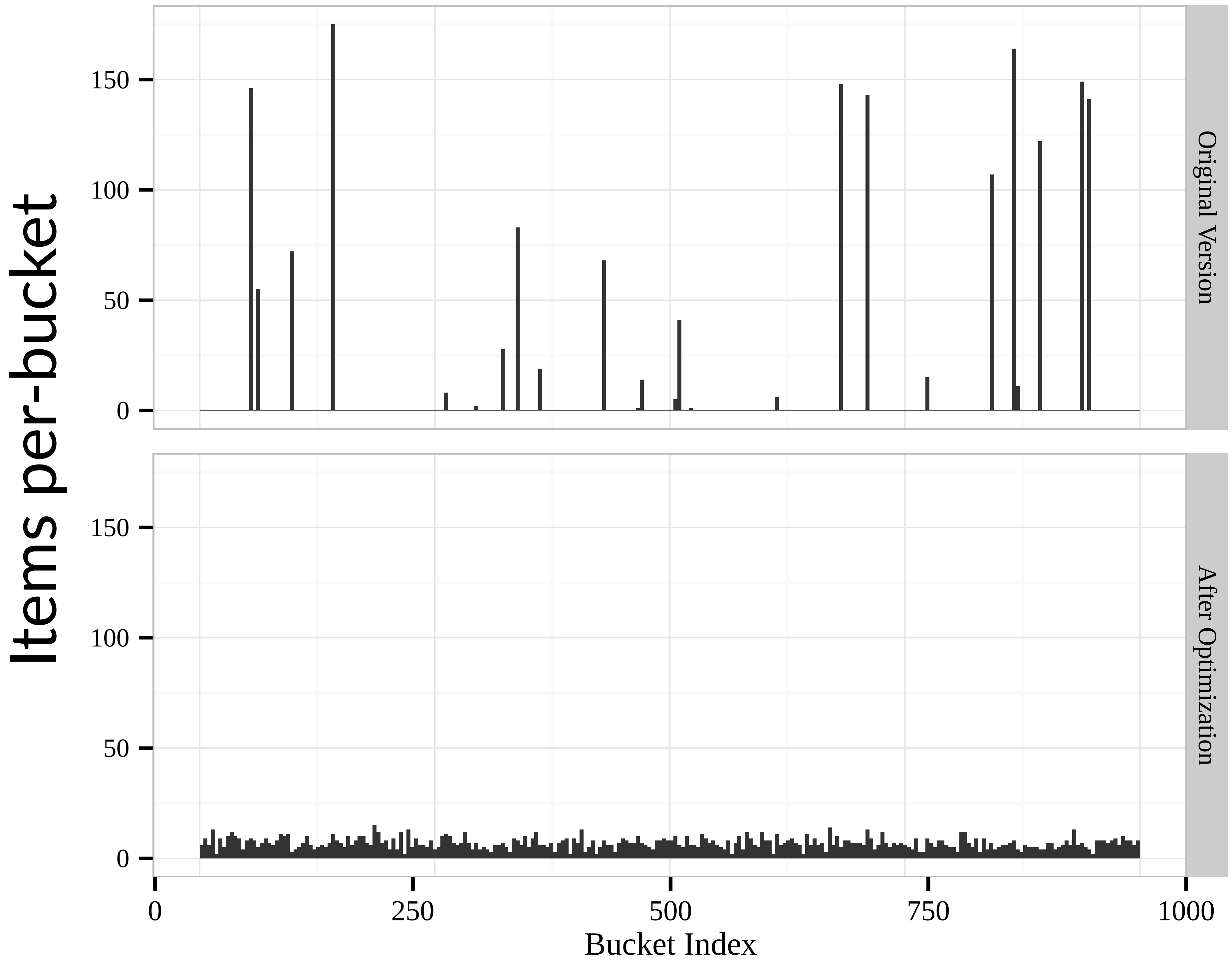


Bin Utilization

2.3%

Dedup

Compression via deduplication

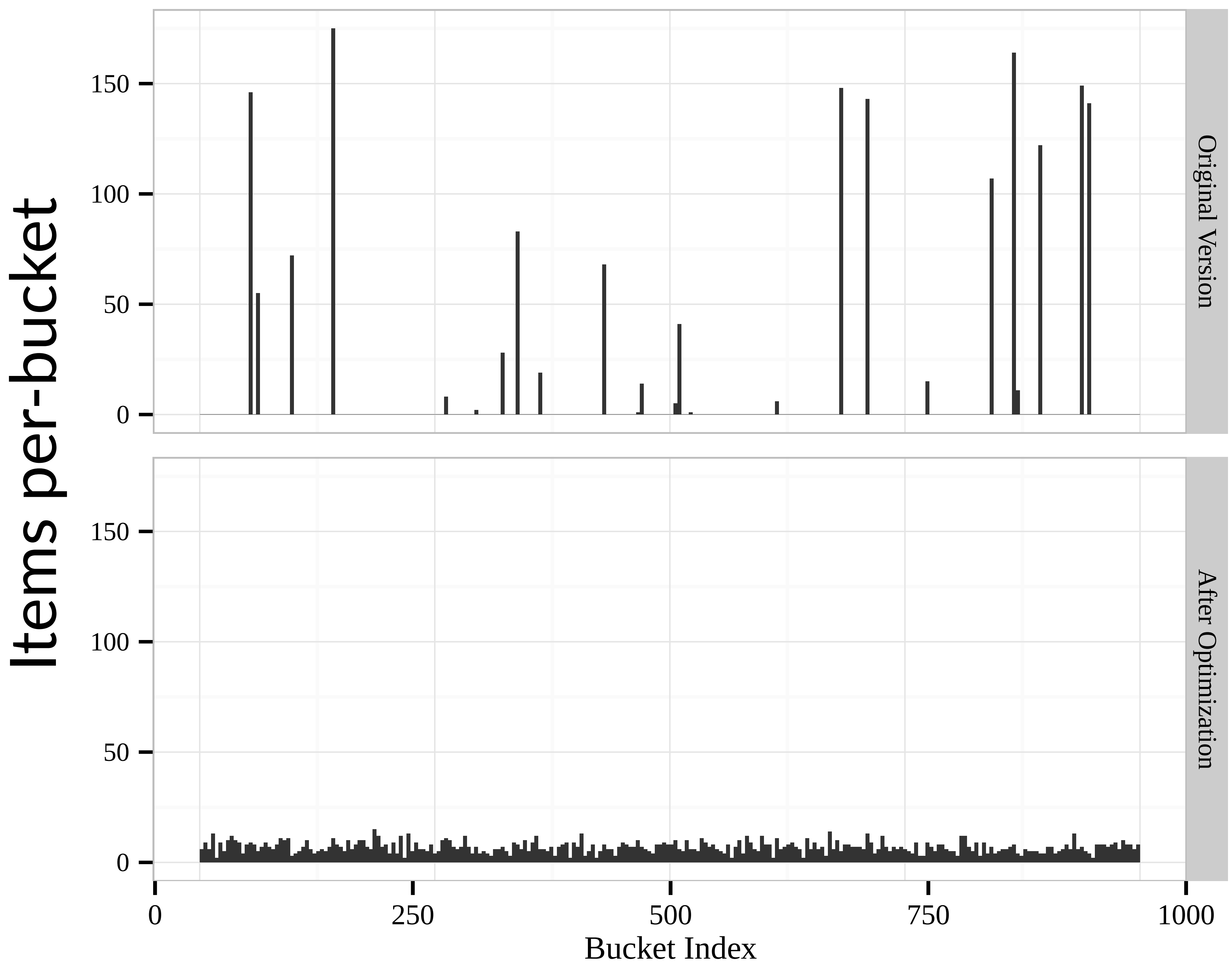


Bin Utilization

2.3%

Dedup

Compression via deduplication



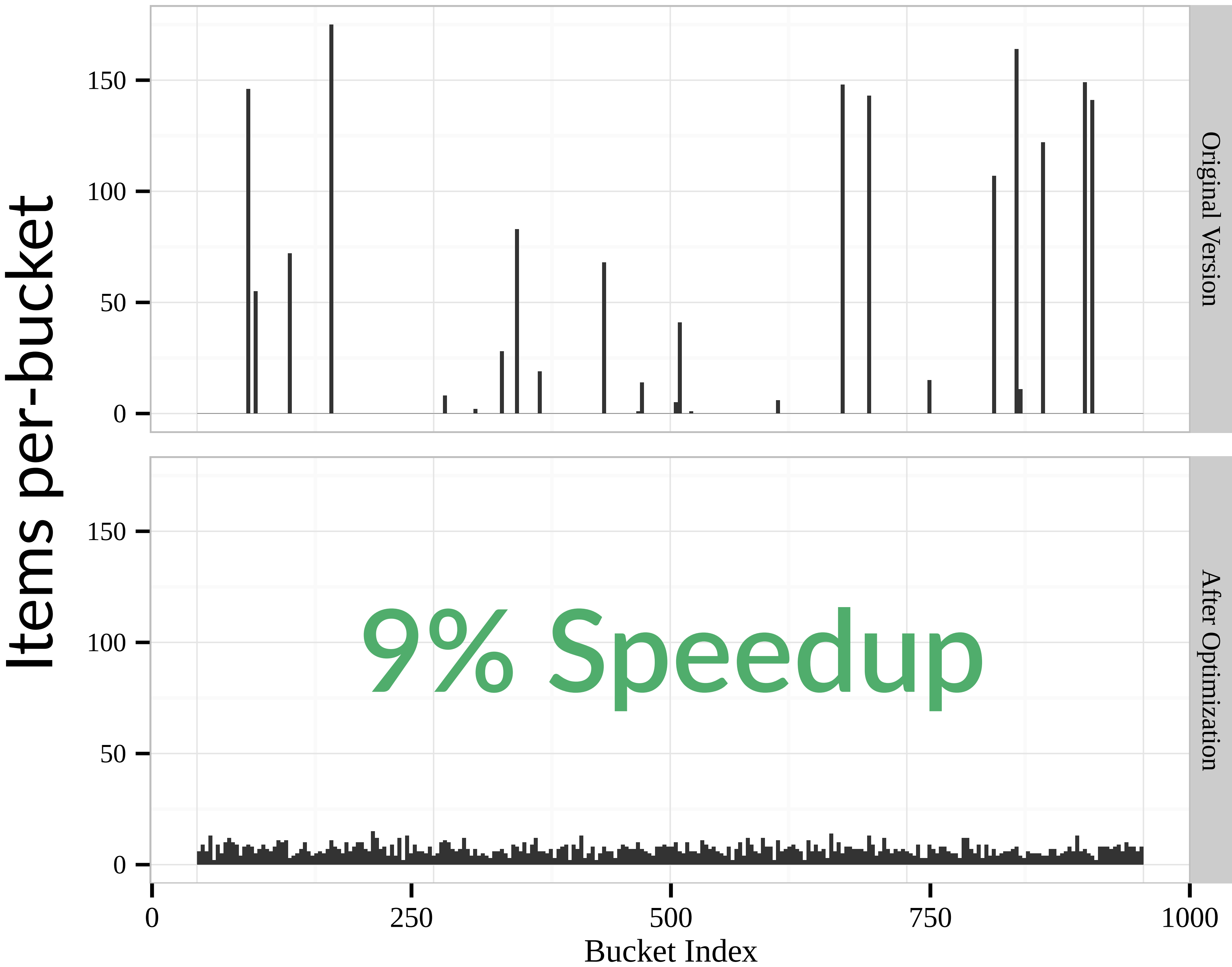
Bin Utilization

2.3%

82%

Dedup

Compression via deduplication



Bin Utilization

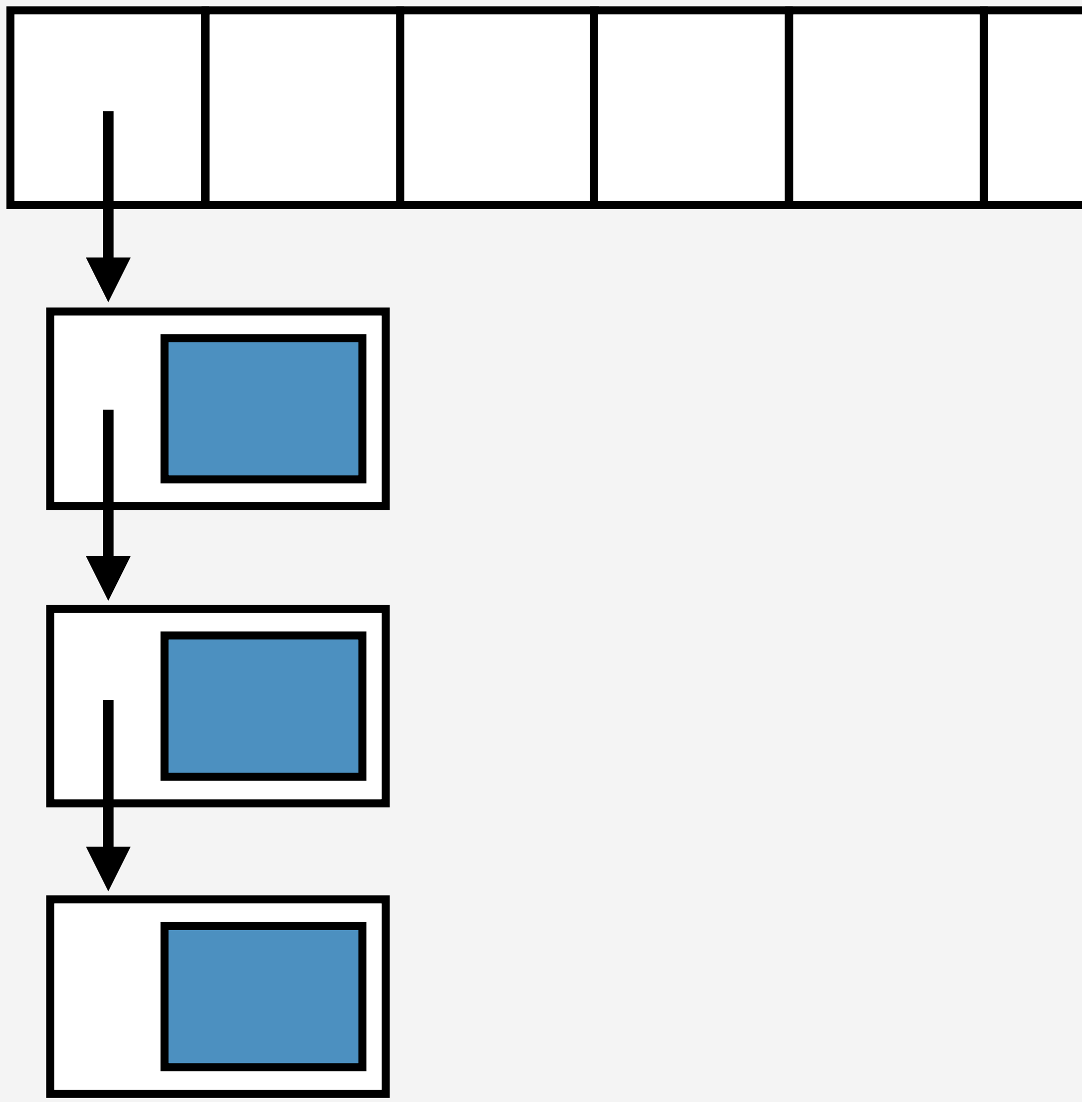
2.3%

82%

Dedup

Compression via deduplication

What did Coz predict?

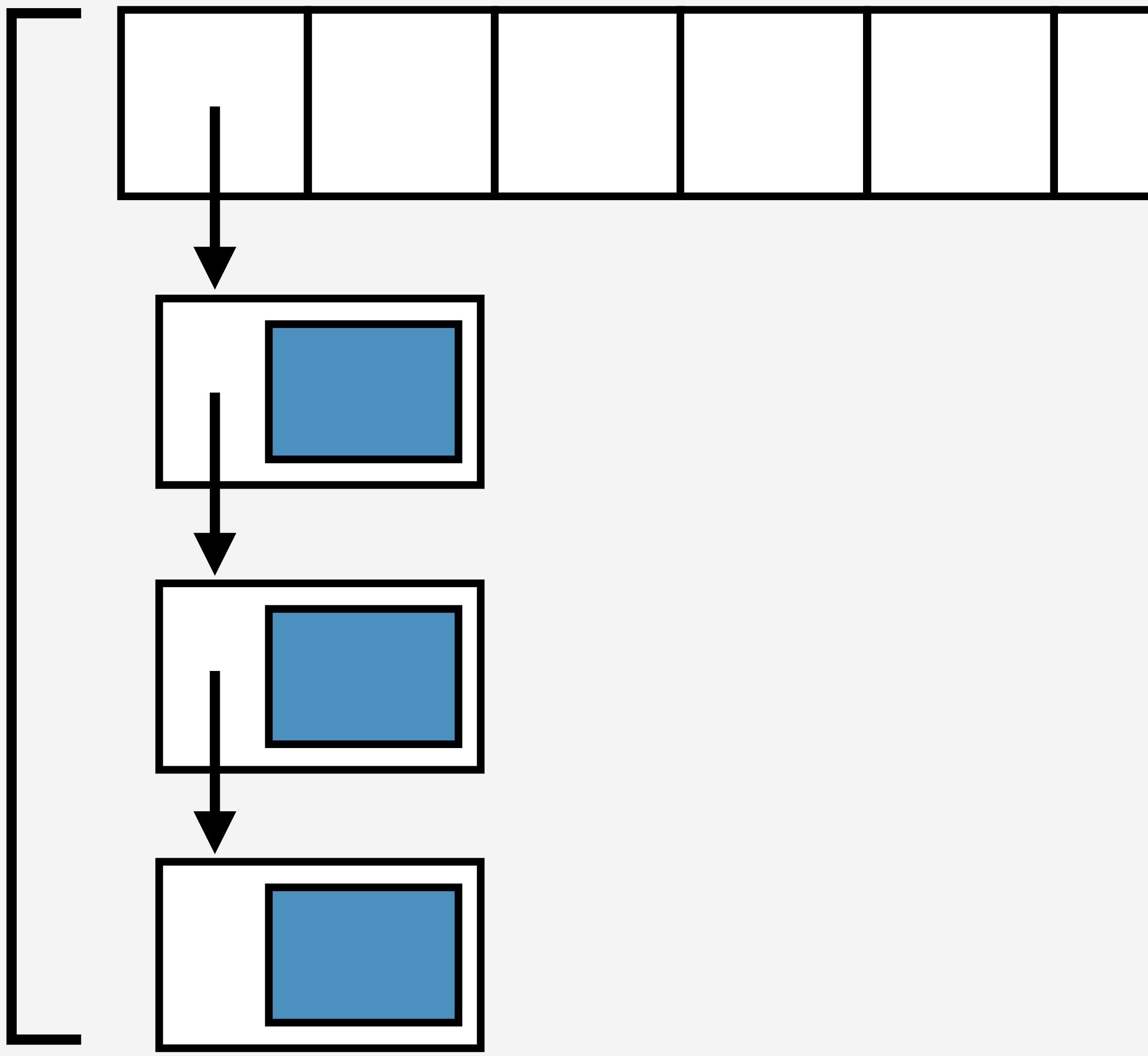


Dedup

Compression via deduplication

What did Coz predict?

Blocks per-bucket



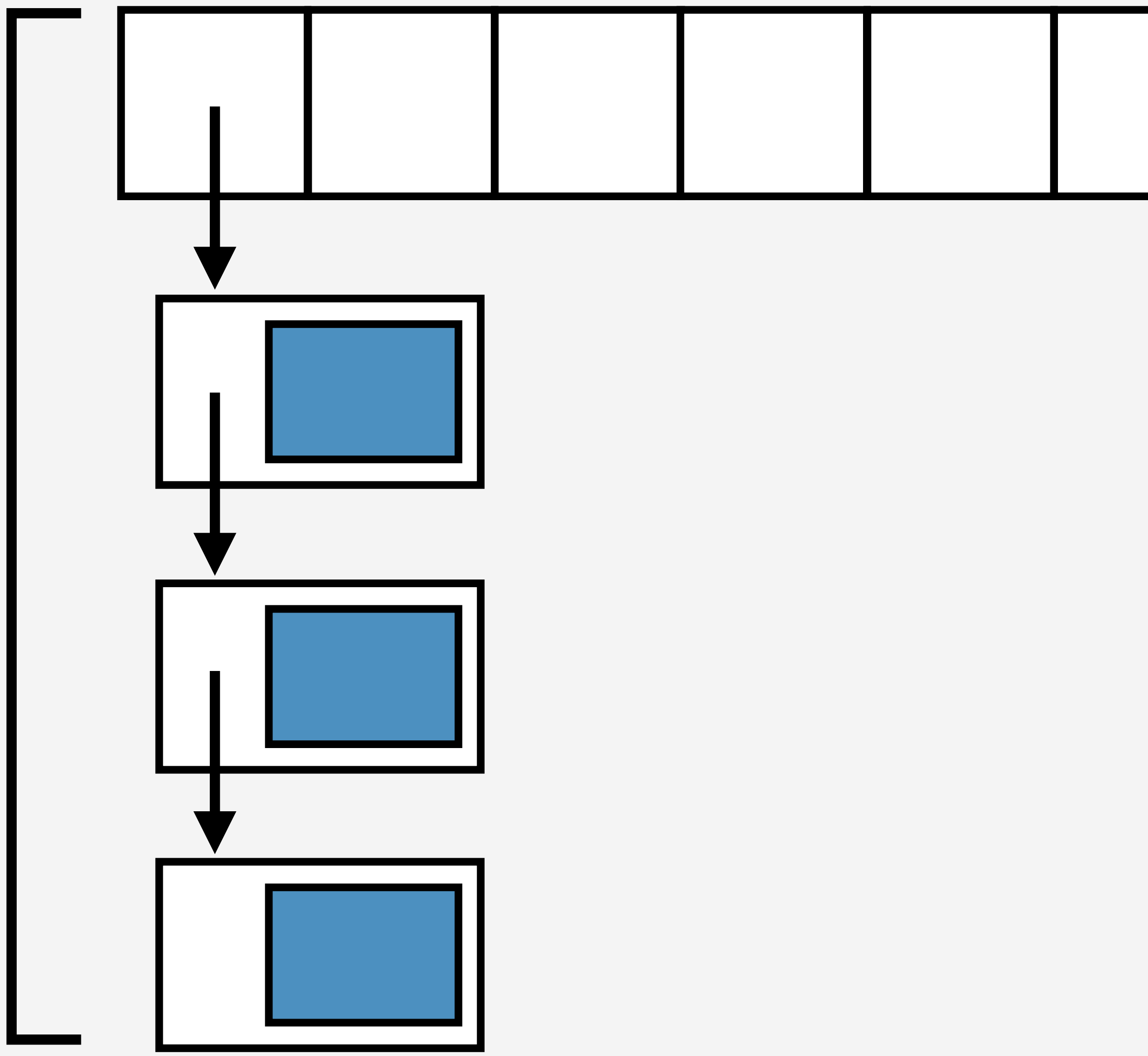
Dedup

Compression via deduplication

What did Coz predict?

Blocks per-bucket

Before: 76.7



Dedup

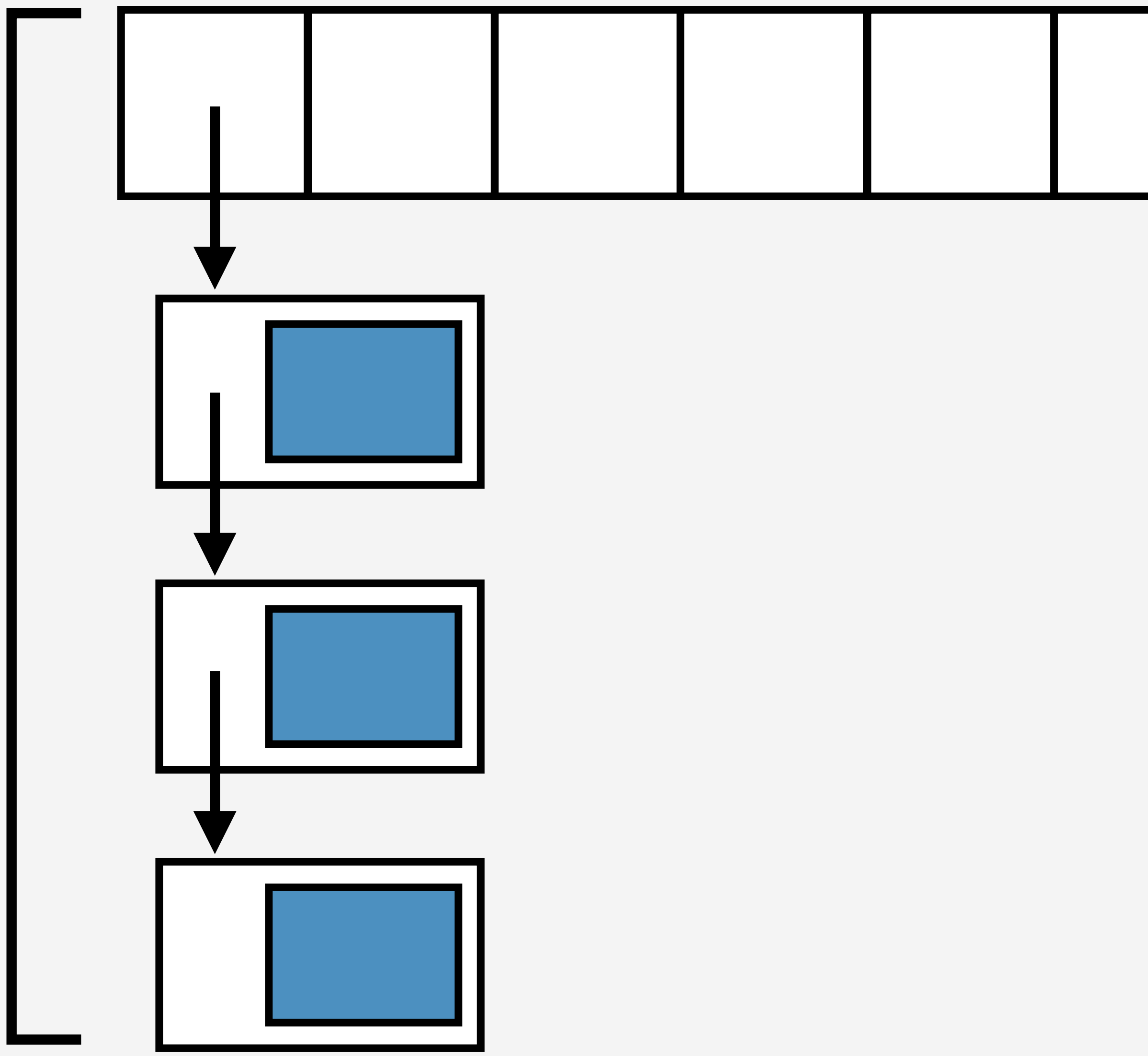
Compression via deduplication

What did Coz predict?

Blocks per-bucket

Before: 76.7

After: 2.09



Dedup

Compression via deduplication

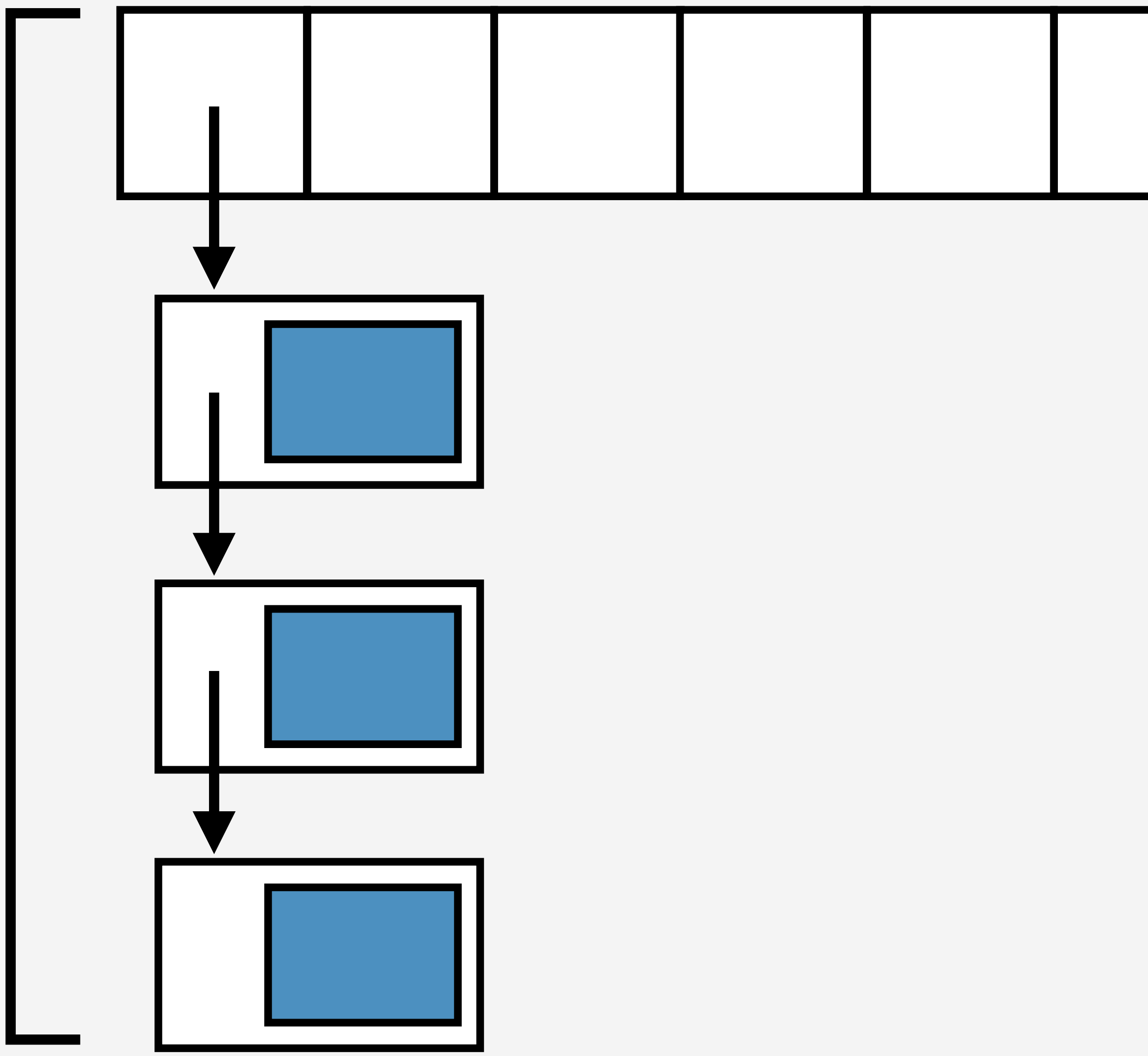
What did Coz predict?

Blocks per-bucket

Before: 76.7

After: 2.09

96% traversal speedup



Dedup

Compression via deduplication

What did Coz predict?

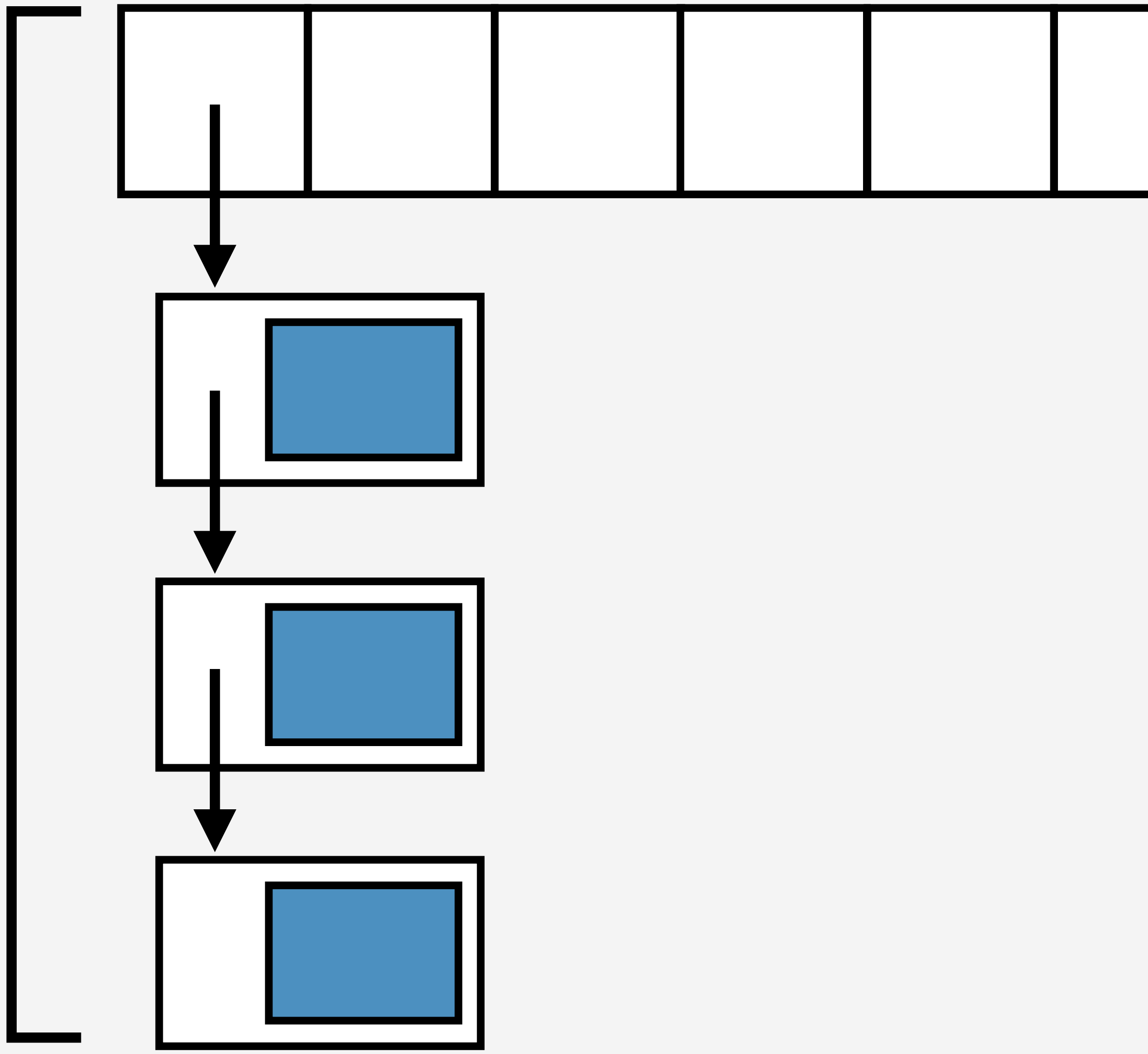
Blocks per-bucket

Before: 76.7

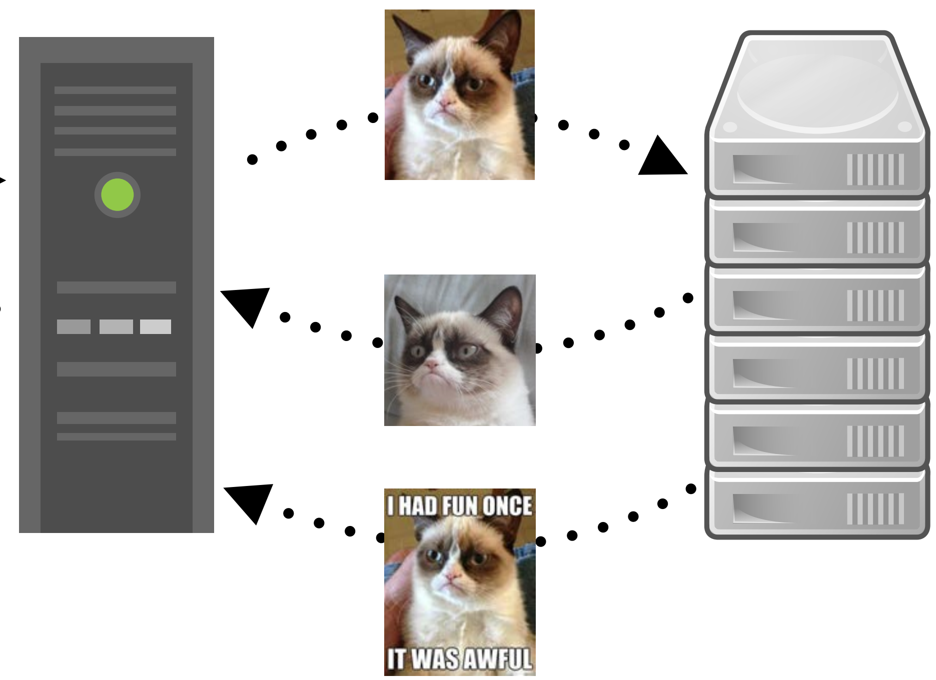
After: 2.09

96% traversal speedup

9% predicted speedup, exactly what we observed



Using Causal Profiling on Ogle

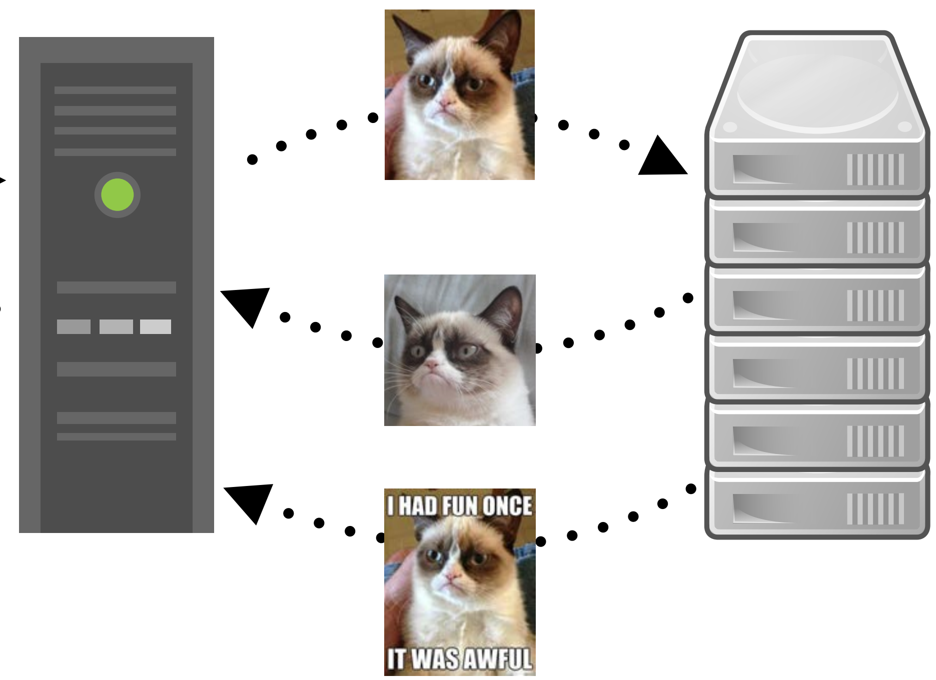


SQLite

dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle

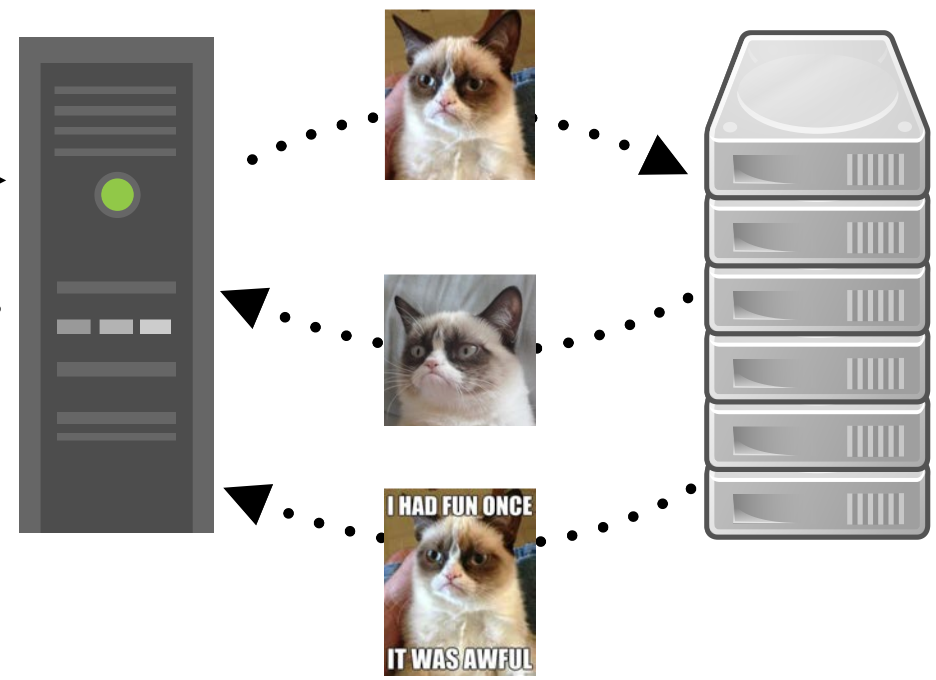


SQLite **25%**

dedup
compression

ferret
image comparison

Using Causal Profiling on Ogle

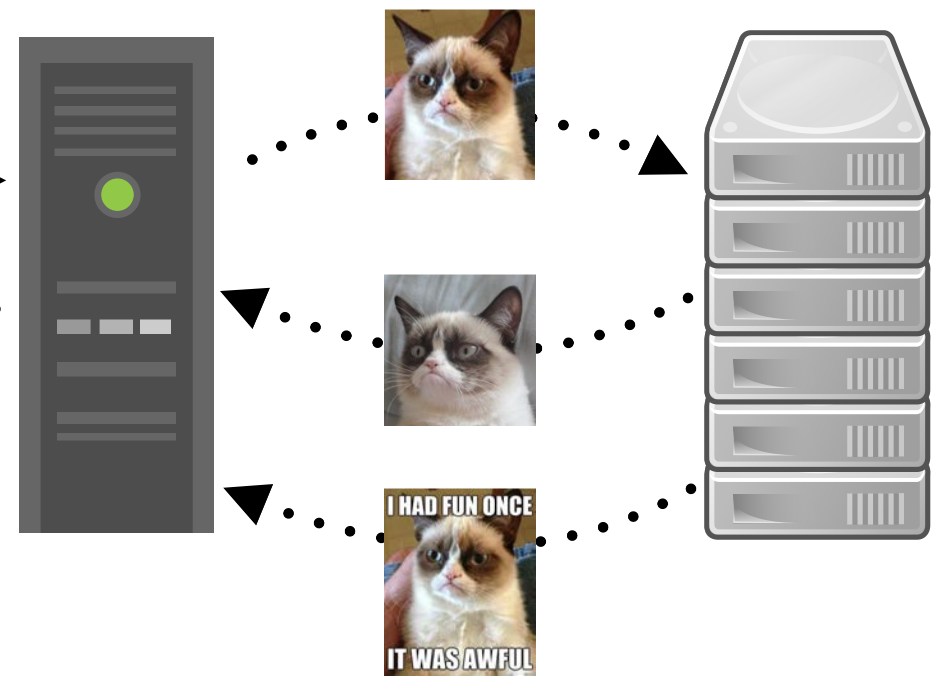


SQLite 25%

dedup
compression 9%

ferret
image comparison

Using Causal Profiling on Ogle



SQLite 25%

dedup
compression 9%

ferret
image comparison 21%

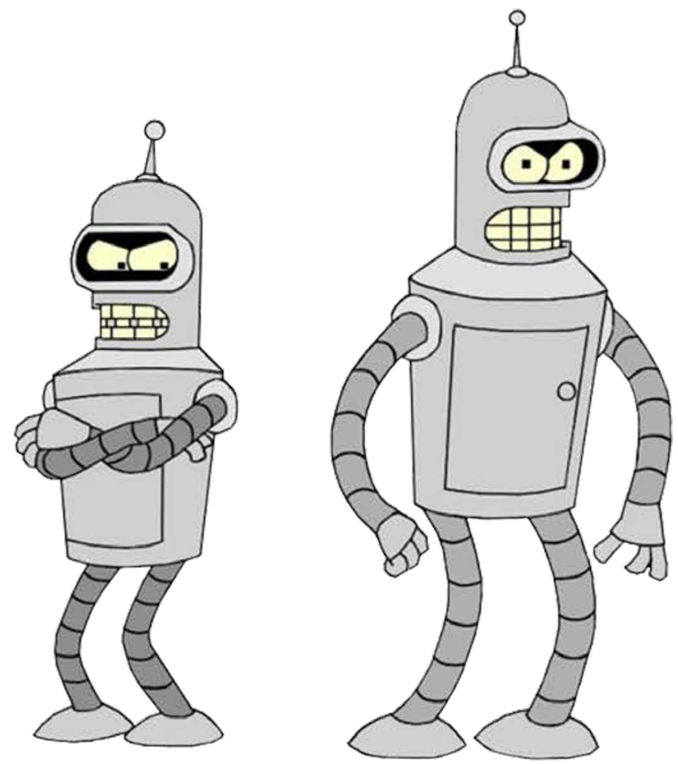
Summary of Optimizations

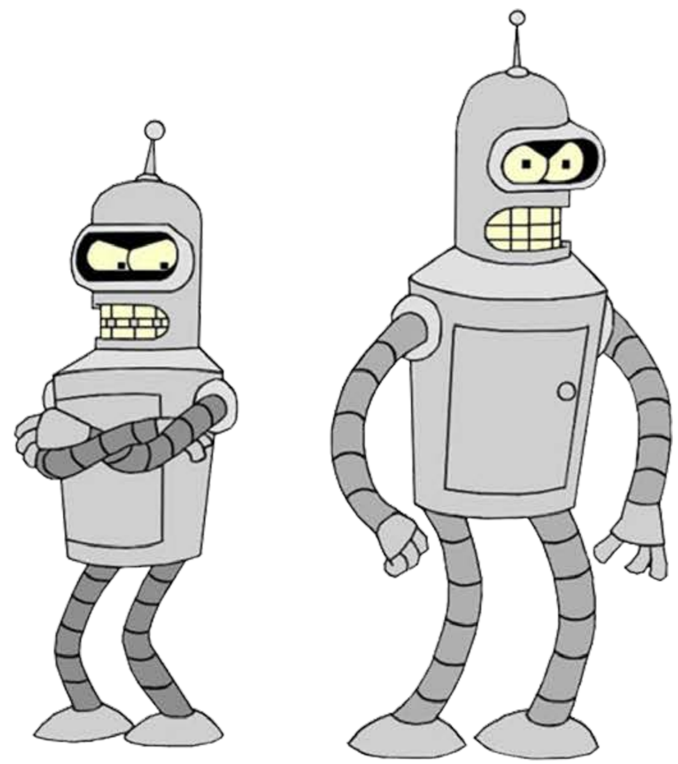
Benchmark	Speedup	Diff Size	Change Summary
memcached	9.39%	-6, +2	removed unnecessary locks
sqlite	25.60%	-3, +3	removed DIY vtable implementation
blackscholes	2.56%	-61, +4	manual common subexpression elimination
dedup	8.95%	-3, +3	fixed degenerate hash function
ferret	21.27%	-4, +4	rebalanced pipeline thread allocation
fluidanimate	37.50%	-1, +0	removed custom barrier with high contention
streamcluster	68.40%	-1, +0	removed custom barrier with high contention
swaptions	15.80%	-10, +16	reordered loop nests

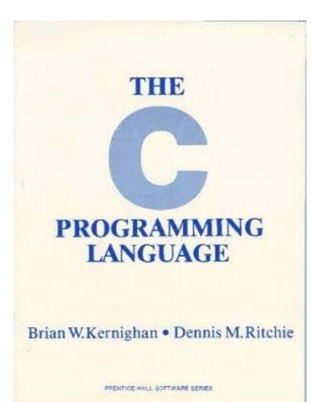
Effective Performance Profiling

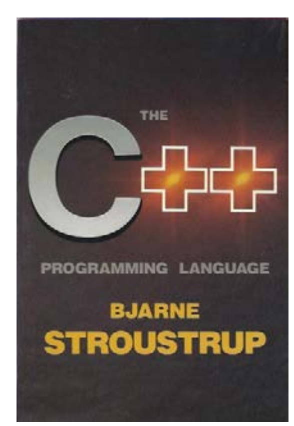
COZ

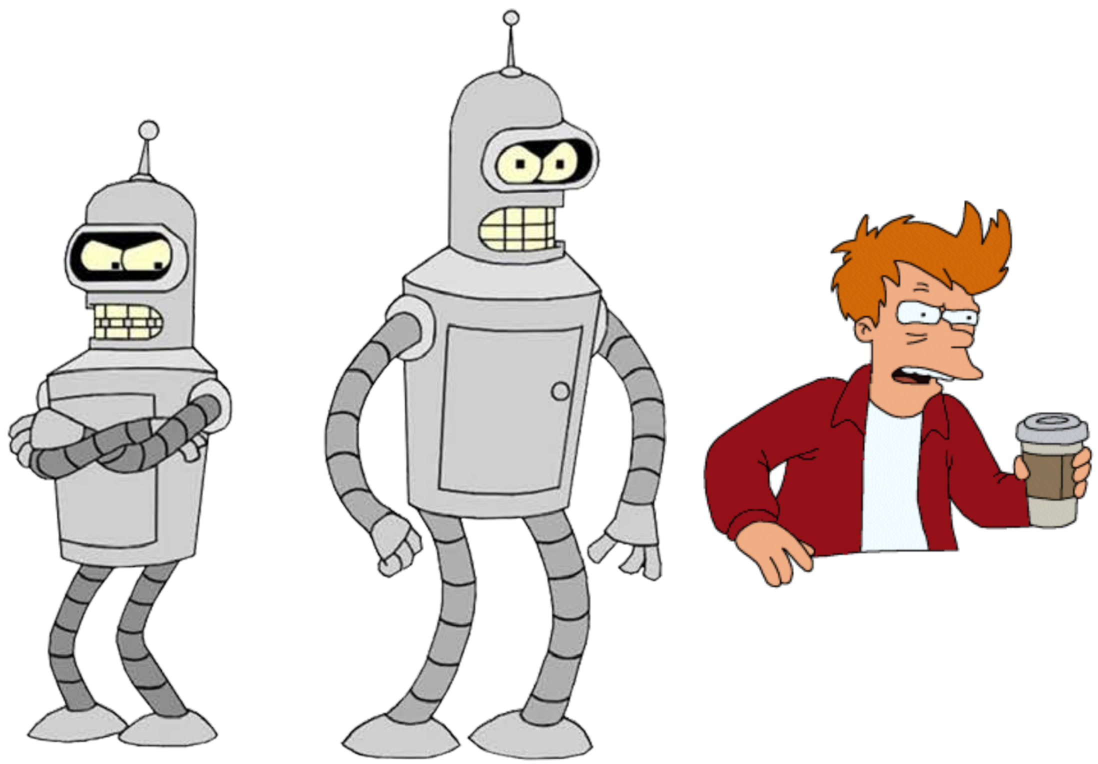
```
% sudo apt install coz-profiler
```

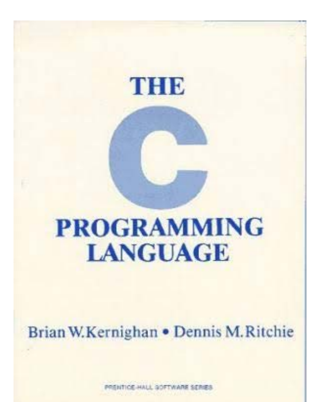


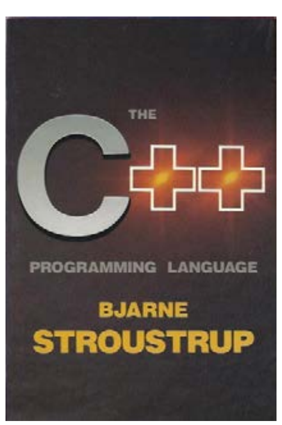


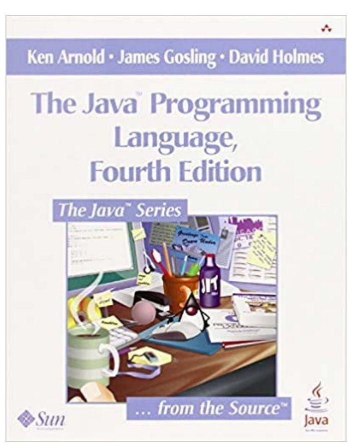

THE
C
PROGRAMMING
LANGUAGE
Brian W.Kernighan • Dennis M.Ritchie

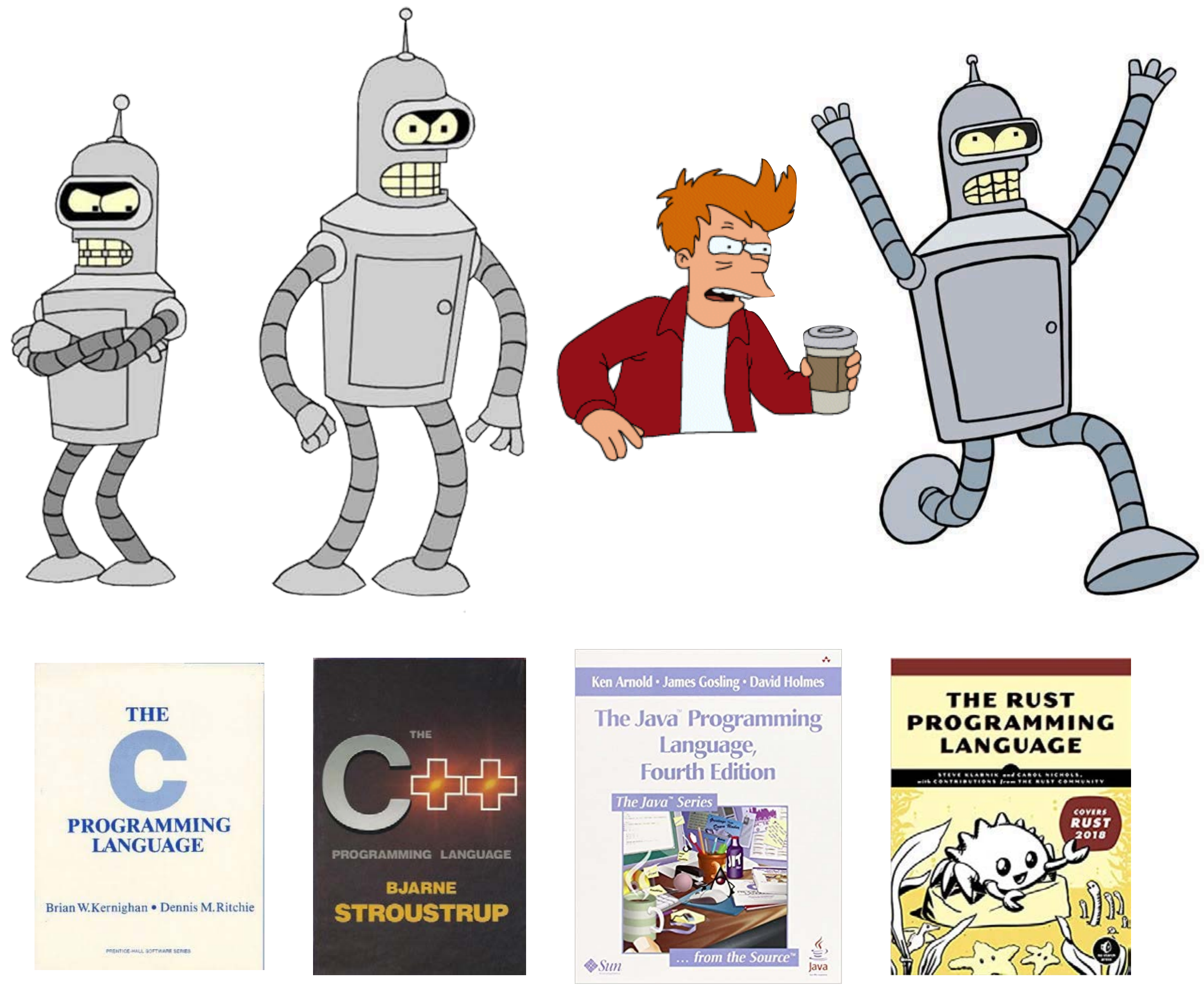

THE
C++
PROGRAMMING LANGUAGE
BJARNE
STROUSTRUP



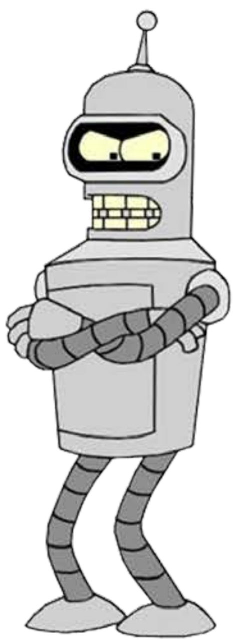

THE
C
PROGRAMMING
LANGUAGE
Brian W.Kernighan • Dennis M.Ritchie

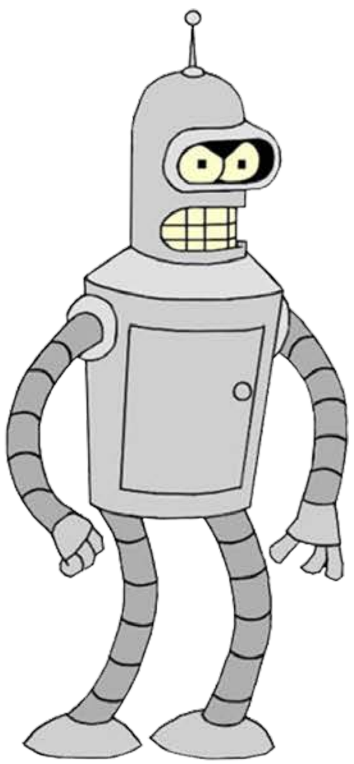

THE
C++
PROGRAMMING LANGUAGE
BJARNE
STROUSTRUP

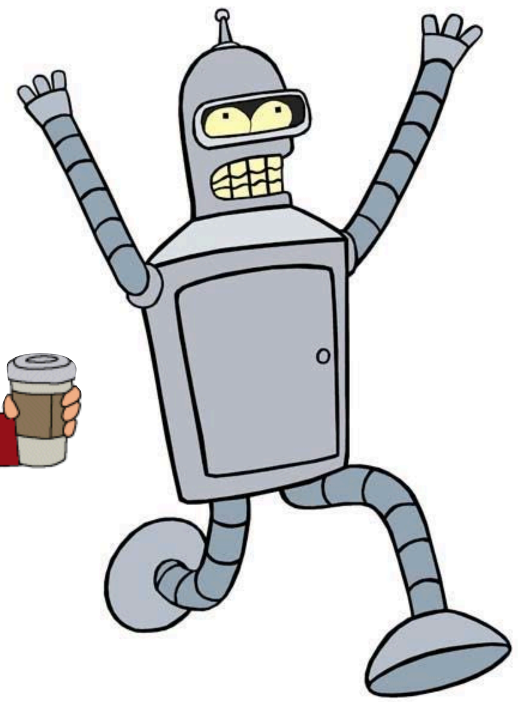

Ken Arnold • James Gosling • David Holmes
The Java™ Programming
Language,
Fourth Edition
The Java™ Series
... from the Source™
Sun
Java

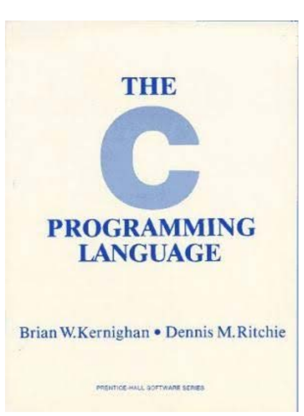

THE
C
PROGRAMMING
LANGUAGE
Brian W.Kernighan • Dennis M.Ritchie
THE
C++
PROGRAMMING LANGUAGE
BJARNE
STROUSTRUP
Ken Arnold • James Gosling • David Holmes
The Java Programming Language, Fourth Edition
The Java Series
... from the Source
Sun
Java
THE RUST PROGRAMMING LANGUAGE
COVERS RUST 2018

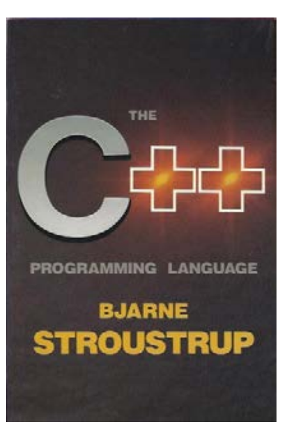

THE C PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
THE C++ PROGRAMMING LANGUAGE
BJARNE STROUSTRUP
Ken Arnold • James Gosling • David Holmes
The Java™ Programming Language, Fourth Edition
The Java™ Series
... from the Source™
Sun
Java
THE RUST PROGRAMMING LANGUAGE
COVERS RUST 2018

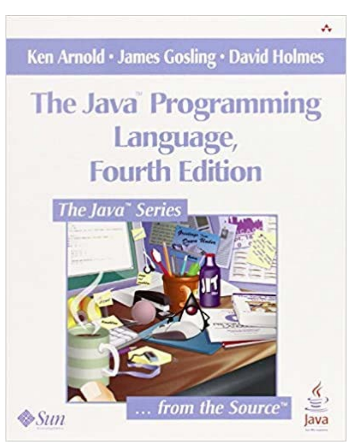


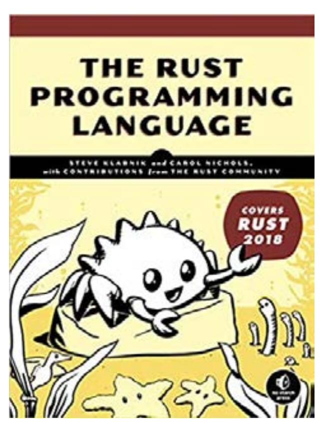





THE
C
PROGRAMMING
LANGUAGE
Brian W.Kernighan • Dennis M.Ritchie


THE
C++
PROGRAMMING LANGUAGE
BJARNE
STROUSTRUP

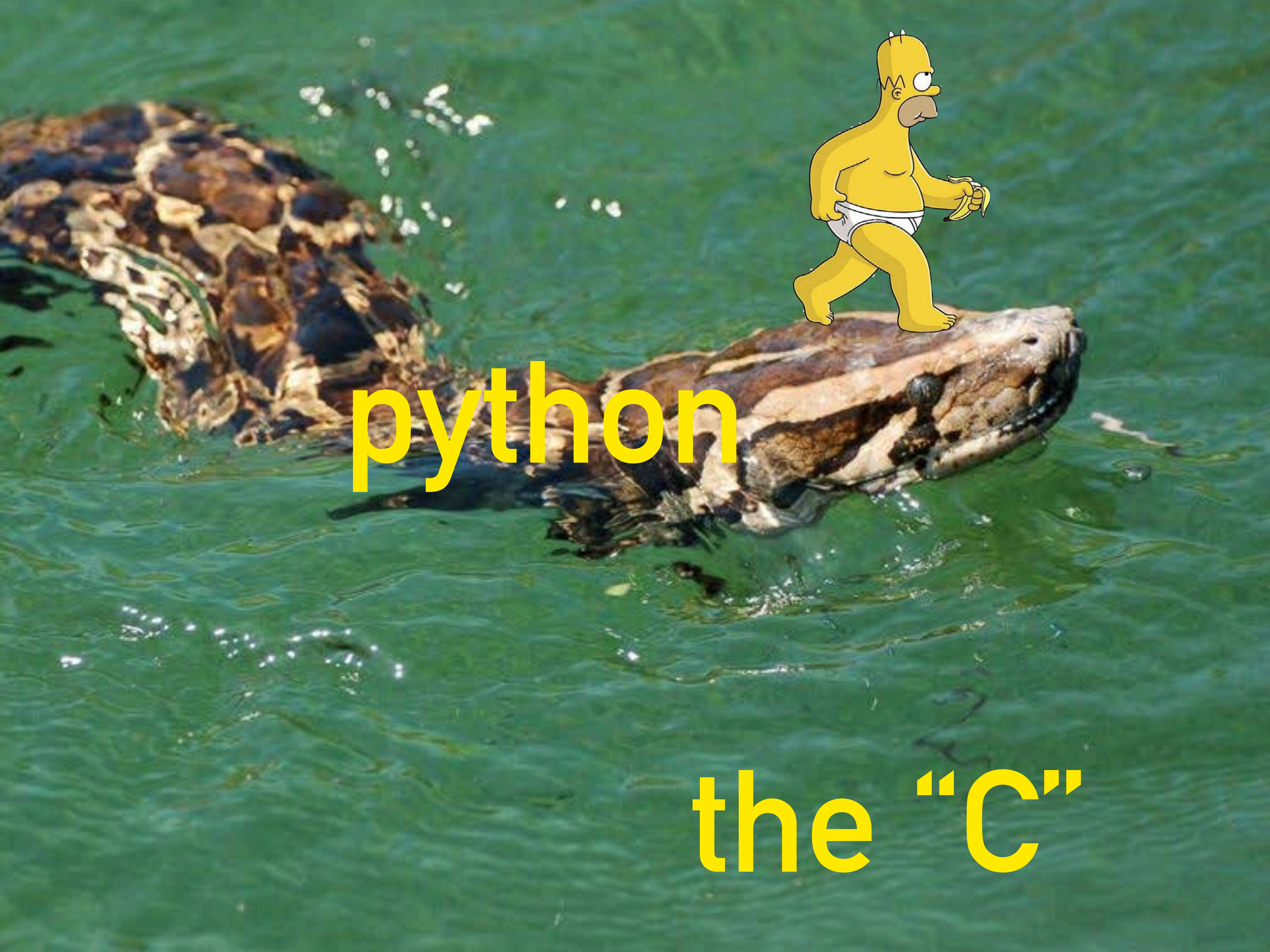

Ken Arnold • James Gosling • David Holmes
The Java Programming Language, Fourth Edition
The Java Series
... from the Source
Sun
Java

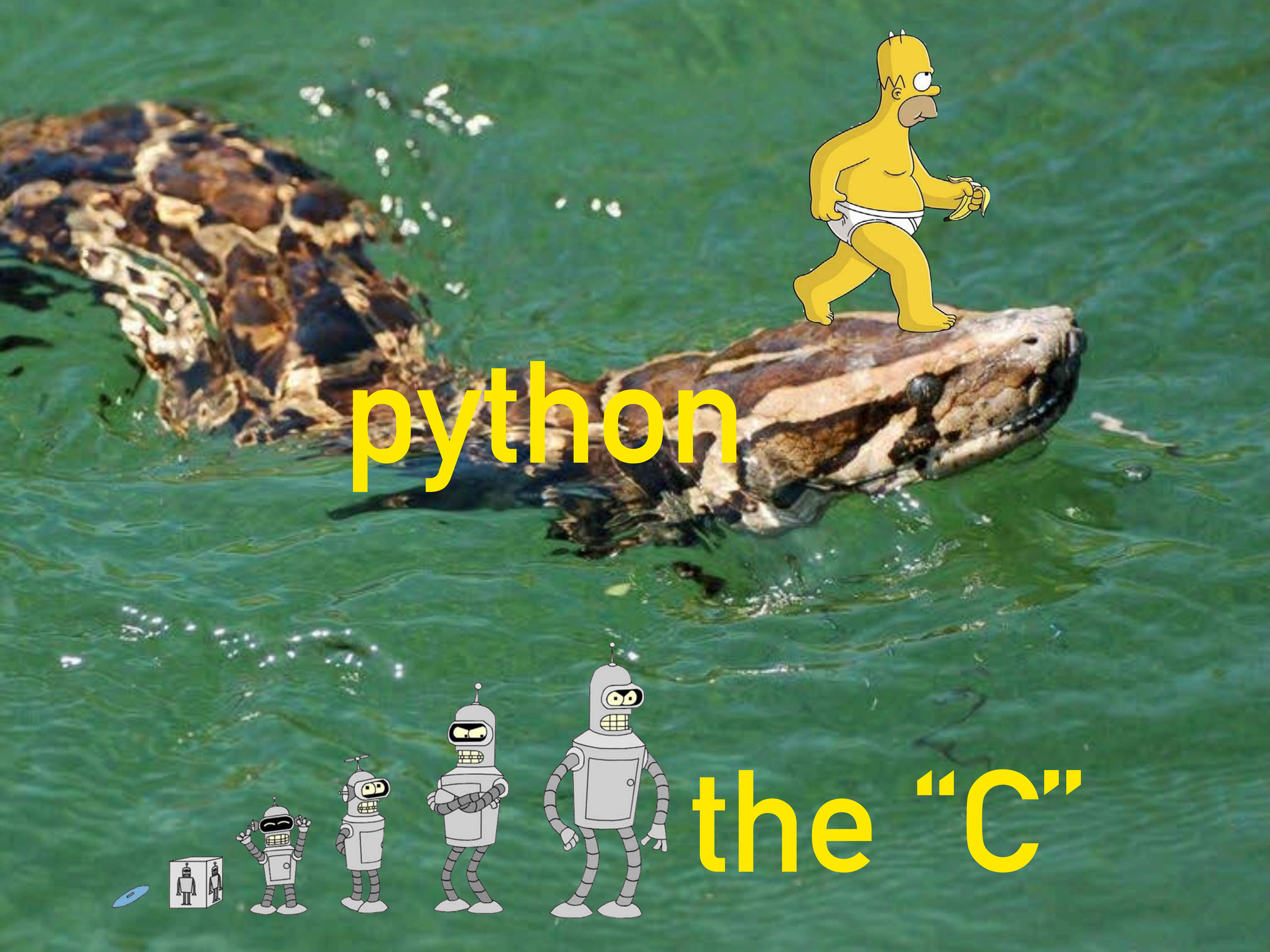

THE RUST
PROGRAMMING
LANGUAGE
COVERS
RUST
2018

python ™


python

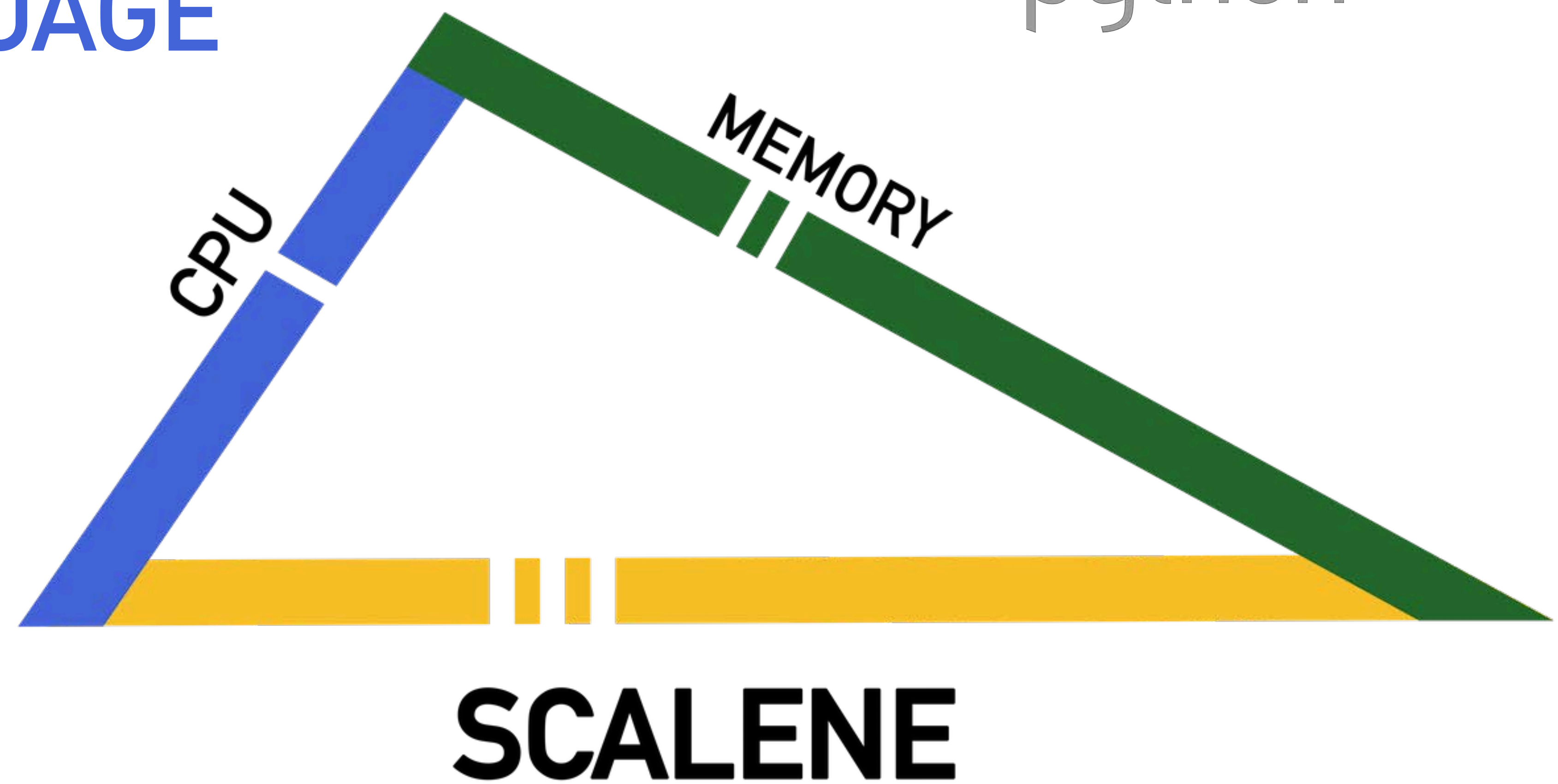

python


python
the "C"


python
the "C"

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING



github.com/emeryberger/scalene

% scalene *yourprogram.py*

```
% scalene --cpu-only yourprogram.py
```

CPU MEMORY SCALENE

```
% scalene --cpu-only --html yourprogram.py
```

```
% scalene --help
```

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))

main()
```

SCALENE

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

CPU
PYTHON vs. NATIVE + SYS%

CPU MEMORY

SCALENE

```
Memory usage:  (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of   6.35s.
```

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

CPU
PYTHON
vs. NATIVE
+ SYS%

MEMORY
PYTHON
vs. NATIVE

SCALENE

MEMORY USAGE OVER TIME

```
Memory usage:                (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of   6.35s.
```

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

CPU PYTHON vs. NATIVE + SYS%

MEMORY PYTHON vs. NATIVE

MEMORY USAGE OVER TIME

% OF MEM ALLOCATED

CPU

MEMORY

SCALENE

MEMORY USAGE OVER TIME

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

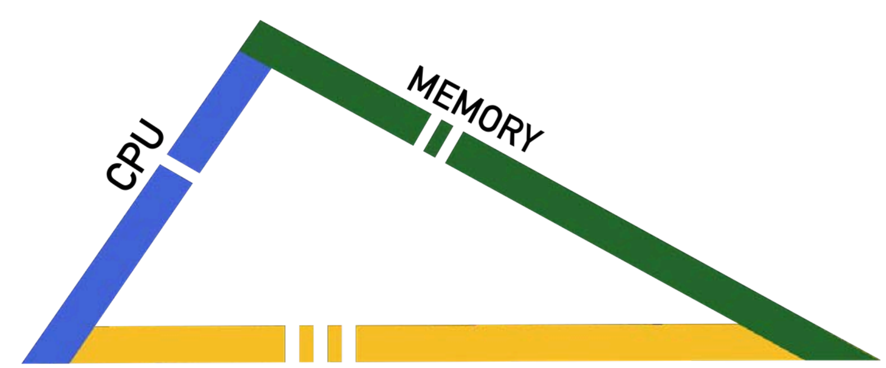
CPU
PYTHON vs. NATIVE + SYS%

MEMORY
PYTHON vs. NATIVE

MEMORY
USAGE OVER TIME

% OF MEM ALLOCATED

COPY
VOLUME (MB/s)



SCALENE

Memory usage: (max: 1618.94MB)

test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

```python
#!/usr/bin/env python3
import numpy as np

def main():
    x = np.array(range(10**7))
    y = np.array(np.random.uniform(0, 100, size=(10**8)))

main()
```

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

34% of runtime in native code

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

34% of runtime in native code

763MB allocated in native code

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

34% of runtime in native code

763MB allocated in native code

87% of memory activity

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								#!/usr/bin/env python3
2	4%	1%	26%	97%	9		3	import numpy as np
3								
4								def main():
5		60%		93%	84			x = np.array(range(10**7))
6	1%	34%			763	87%	120	y = np.array(np.random.uniform(0, 100, size=(10**8)))
7								
8								main()

generated by the scalene profiler

34% of runtime in native code

763MB allocated in native code

87% of memory activity

"sawtooth" pattern

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

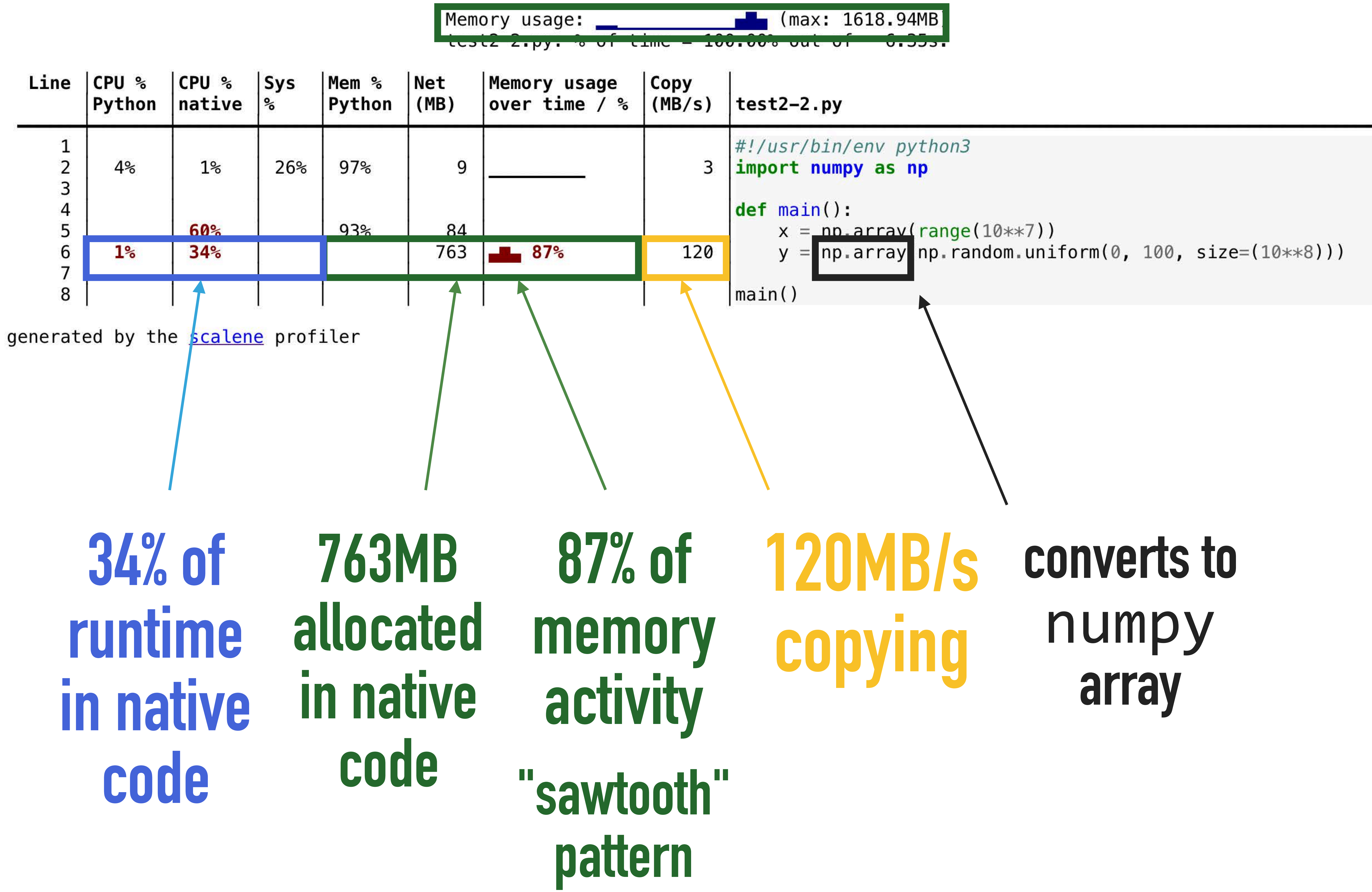
34% of runtime in native code

763MB allocated in native code

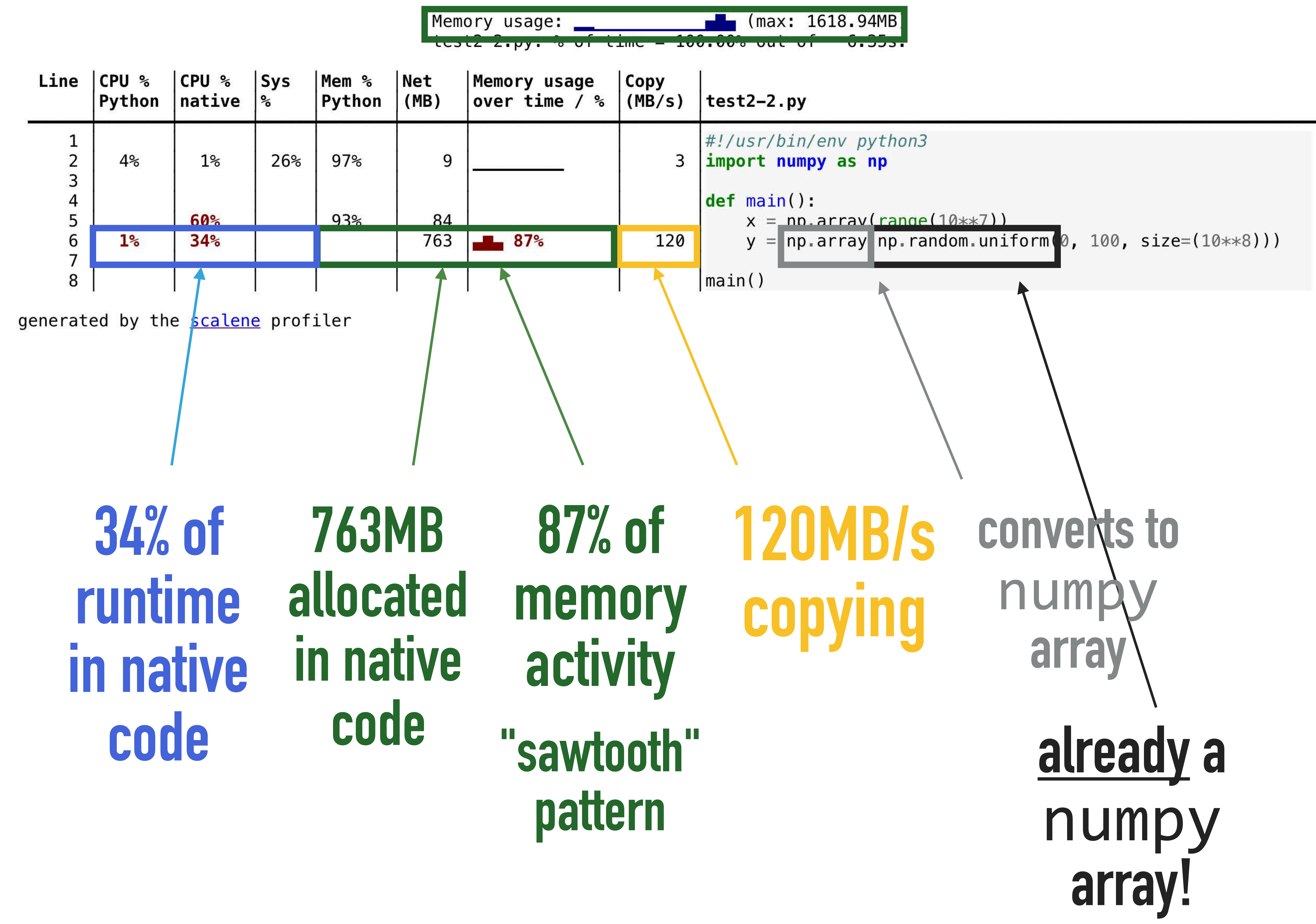
87% of memory activity

"sawtooth" pattern

120MB/s copying



Memory usage: (max: 1618.94MB)
Line
CPU % Python
CPU % native
Sys %
Mem % Python
Net (MB)
Memory usage over time / %
Copy (MB/s)
test2-2.py
1 #!/usr/bin/env python3
2 4% 1% 26% 97% 9 3 import numpy as np
3
4 def main():
5 60% 93% 84 x = np.array(range(10**7))
6 1% 34% 763 87% 120 y = np.array(np.random.uniform(0, 100, size=(10**8)))
7
8 main()
generated by the scalene profiler
34% of runtime in native code
763MB allocated in native code
87% of memory activity
"sawtooth" pattern
120MB/s copying
converts to numpy array



Memory usage: (max: 1618.94MB)
Line
CPU % Python
CPU % native
Sys %
Mem % Python
Net (MB)
Memory usage over time / %
Copy (MB/s)
test2-2.py
1 #!/usr/bin/env python3
2 4% 1% 26% 97% 9 3 import numpy as np
3
4 def main():
5 60% 93% 84 x = np.array(range(10**7))
6 1% 34% 763 87% 120 y = np.array(np.random.uniform(0, 100, size=(10**8)))
7
8 main()
generated by the scalene profiler
34% of runtime in native code
763MB allocated in native code
87% of memory activity
"sawtooth" pattern
120MB/s copying
converts to numpy array
already a numpy array!

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		**60%**		93%	84			`    x = np.array(range(10**7))`
6	**1%**	**34%**			763	**87%**	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 865.97MB)
test2-3.py: % of time = 100.00% out of 5.40s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-3.py
1								`#!/usr/bin/env python3`
2	6%	1%	34%	93%	21		3	`import numpy as np`
3								
4								`#@profile`
5								`def main():`
6		**67%**		94%	82			`    x = np.array(range(10**7))`
7		25%	5%		763	85%		`    y = np.random.uniform(0, 100, size=(10**8))`
8								
9								`main()`

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 865.97MB)
test2-3.py: % of time = 100.00% out of 5.40s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-3.py
1								`#!/usr/bin/env python3`
2	6%	1%	34%	93%	21		3	`import numpy as np`
3								
4								`#@profile`
5								`def main():`
6		67%		94%	82			`    x = np.array(range(10**7))`
7		25%	5%		763	85%		`    y = np.random.uniform(0, 100, size=(10**8))`
8								
9								`main()`

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 865.97MB)
test2-3.py: % of time = 100.00% out of 5.40s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-3.py
1								`#!/usr/bin/env python3`
2	6%	1%	34%	93%	21		3	`import numpy as np`
3								
4								`#@profile`
5								`def main():`
6		67%		94%	82			`    x = np.array(range(10**7))`
7		25%	5%		763	85%		`    y = np.random.uniform(0, 100, size=(10**8))`
8								
9								`main()`

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 865.97MB)
test2-3.py: % of time = 100.00% out of 5.40s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-3.py
1								`#!/usr/bin/env python3`
2	6%	1%	34%	93%	21		3	`import numpy as np`
3								
4								`#@profile`
5								`def main():`
6		67%		94%	82			`    x = np.array(range(10**7))`
7		25%	5%		763	85%		`    y = np.random.uniform(0, 100, size=(10**8))`
8								
9								`main()`

generated by the scalene profiler

Memory usage: (max: 1618.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 865.97MB)
test2-3.py: % of time = 100.00% out of 5.40s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-3.py
1								`#!/usr/bin/env python3`
2	6%	1%	34%	93%	21		3	`import numpy as np`
3								
4								`#@profile`
5								`def main():`
6		67%		94%	82			`x = np.array(range(10**7))`
7		25%	5%		763	85%		`y = np.random.uniform(0, 100, size=(10**8))`
8								
9								`main()`

generated by the scalene profiler

Memory usage: (max: 16[illegible]8.94MB)
test2-2.py: % of time = 100.00% out of 6.35s.

Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-2.py
1								`#!/usr/bin/env python3`
2	4%	1%	26%	97%	9		3	`import numpy as np`
3								
4								`def main():`
5		60%		93%	84			`    x = np.array(range(10**7))`
6	1%	34%			763	87%	120	`    y = np.array(np.random.uniform(0, 100, size=(10**8)))`
7								
8								`main()`

generated by the scalene profiler

Memory usage: (max: 865.97MB)
test2-3.py: % of time = 100.00% out of 5.40s.

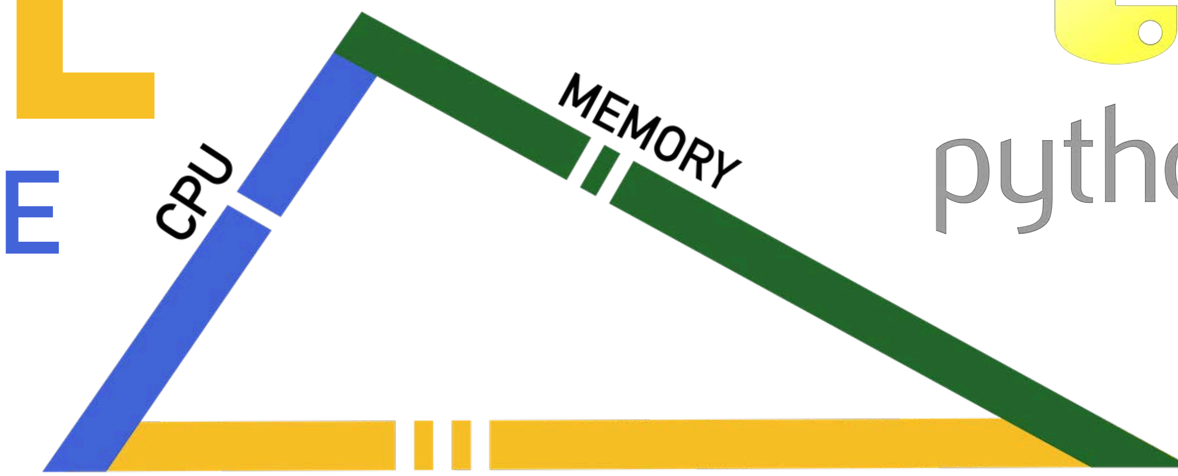
Line	CPU % Python	CPU % native	Sys %	Mem % Python	Net (MB)	Memory usage over time / %	Copy (MB/s)	test2-3.py
1								`#!/usr/bin/env python3`
2	6%	1%	34%	93%	21		3	`import numpy as np`
3								
4								`#@profile`
5								`def main():`
6		67%		94%	82			`    x = np.array(range(10**7))`
7		25%	5%		763	85%		`    y = np.random.uniform(0, 100, size=(10**8))`
8								
9								`main()`

generated by the scalene profiler

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING

python™



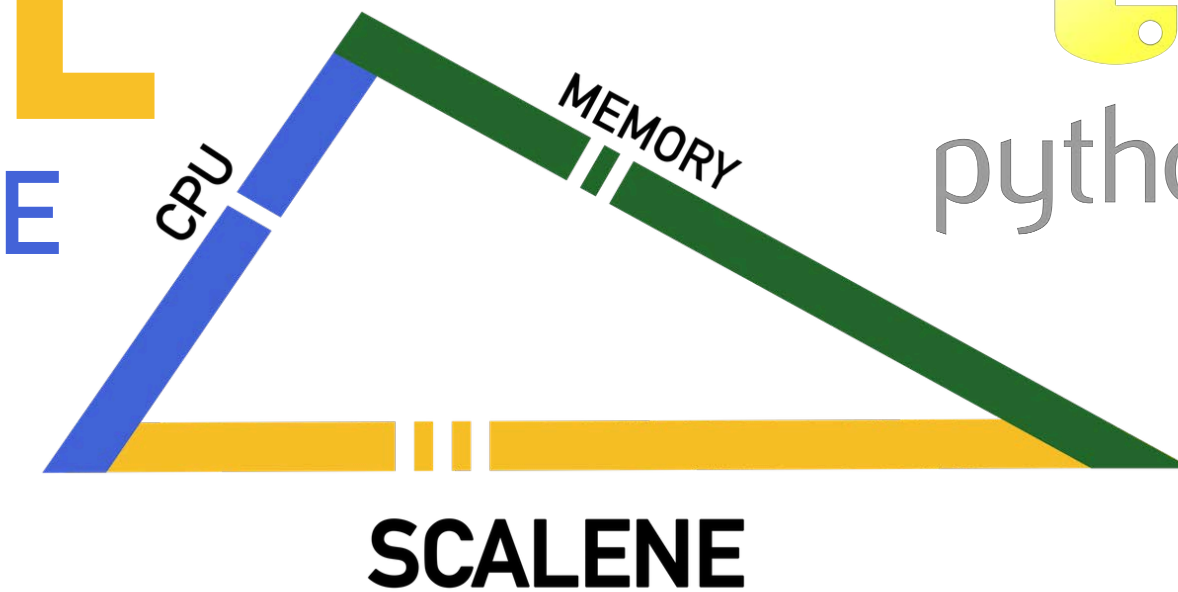
SCALENE

precise

CPU	MEMORY	MEMORY	COPY
PYTHON vs. NATIVE + SYS%	PYTHON vs. NATIVE	USAGE OVER TIME	VOLUME (MB/s)
		% OF MEM ALLOCATED	

SCALENE

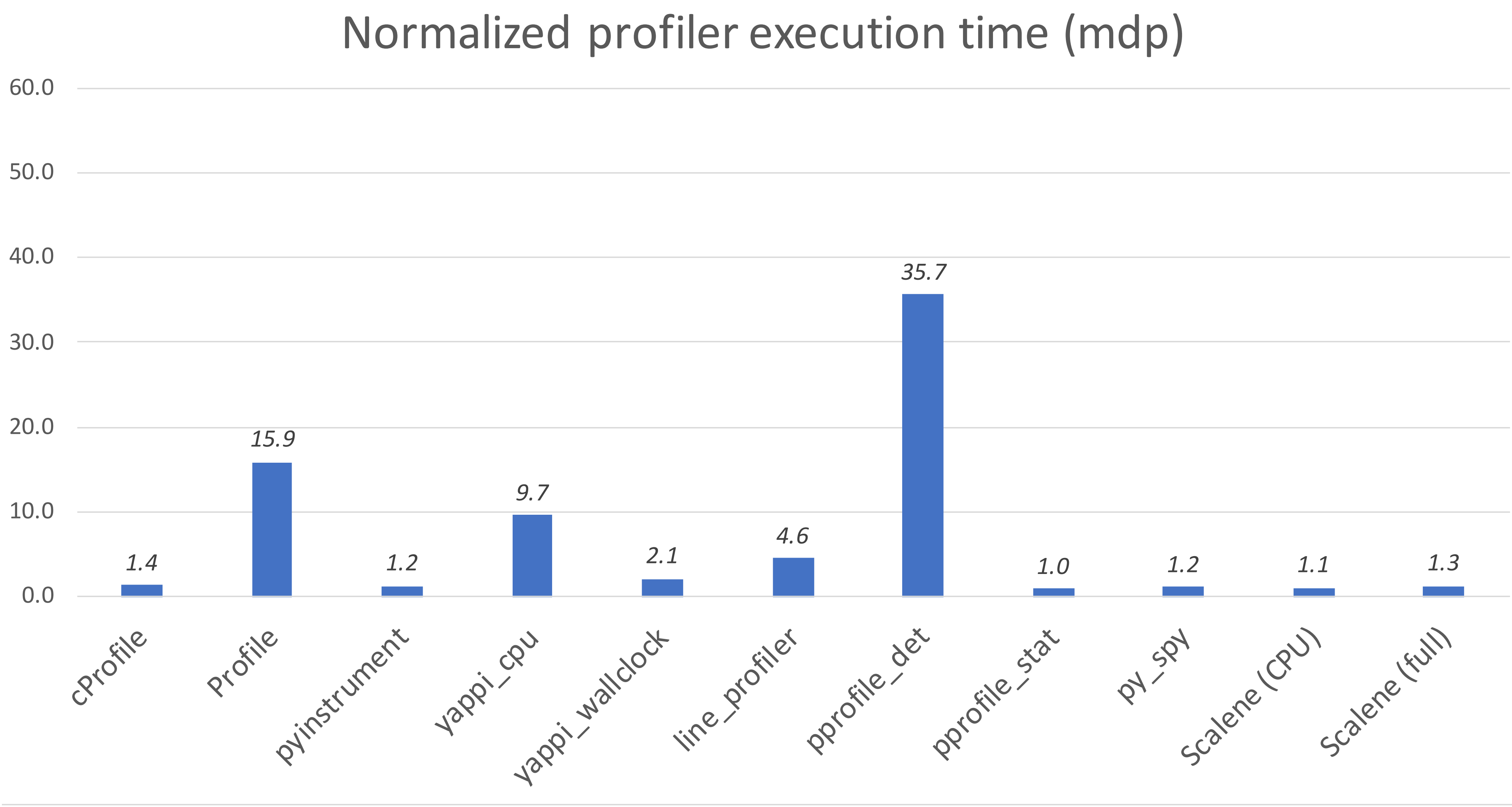
SCRIPTING-LANGUAGE AWARE PROFILING

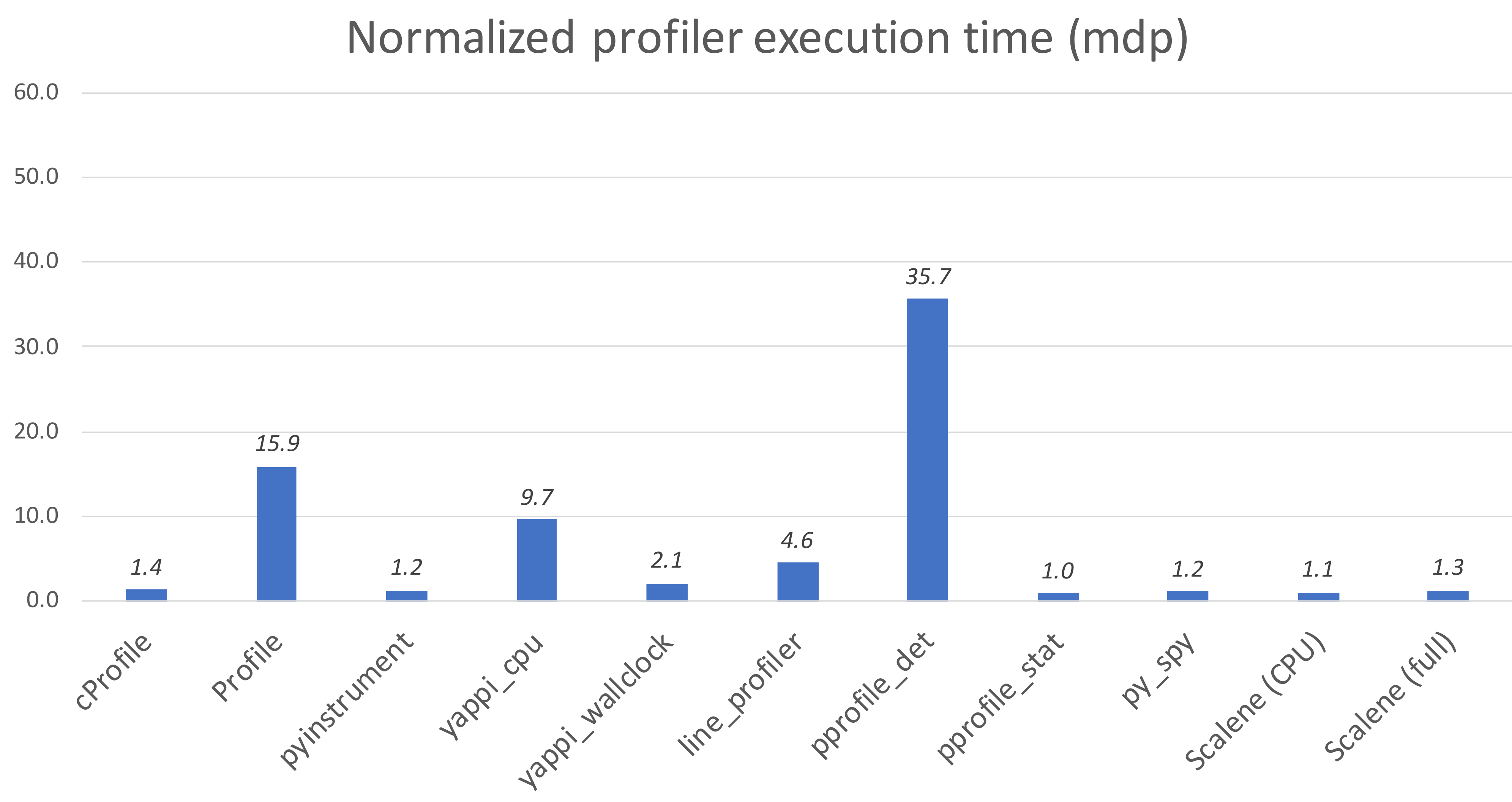


precise

fast

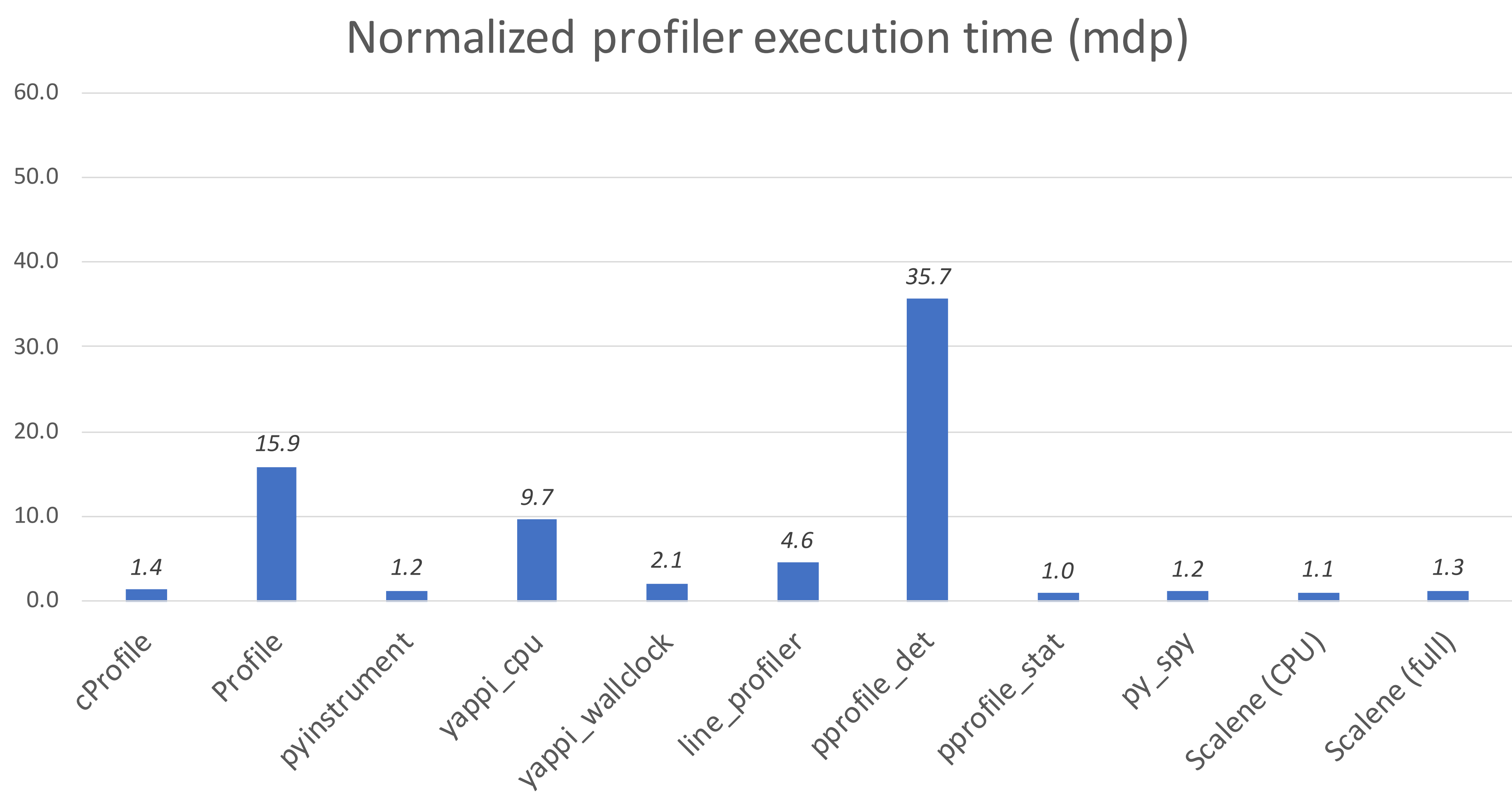
CPU	MEMORY	MEMORY	COPY
PYTHON vs. NATIVE + SYS%	PYTHON vs. NATIVE	USAGE OVER TIME	VOLUME (MB/s)
		% OF MEM ALLOCATED	


Normalized profiler execution time (mdp)
60.0
50.0
40.0
30.0
20.0
10.0
0.0
1.4
15.9
1.2
9.7
2.1
4.6
35.7
1.0
1.2
1.1
1.3
cProfile
Profile
pyinstrument
yappi_cpu
yappi_wallclock
line_profiler
pprofile_det
pprofile_stat
py_spy
Scalene (CPU)
Scalene (full)



memory_profiler: **152x**
(12m38s vs. 5s)

SCALENE



memory_profiler: **152x**
(12m38s vs. 5s)

CPU
MEMORY
SCALENE

DEFERRED SIGNAL DELIVERY

DEFERRED SIGNAL DELIVERY

DEFERRED SIGNAL DELIVERY

DEFERRED SIGNAL DELIVERY




DEFERRED SIGNAL DELIVERY

DEFERRED SIGNAL DELIVERY

DEFERRED SIGNAL DELIVERY

python™ python™ python™ python™




python™

DEFERRED SIGNAL DELIVERY






DEFERRED SIGNAL DELIVERY






INFERRING EXECUTION TIME

(wall clock)

INFERRING EXECUTION TIME

(wall clock)

INFERRING EXECUTION TIME

delay

(wall clock)

```
python_time += interval
c_time += delay - interval
```

DEFERRED SIGNAL DELIVERY

delay

(wall clock)

```
python_time += interval
c_time += delay - interval
```

(virtual time)

```
sys_time = 1 - elapsed_virtual /
elapsed_wallclock
```

DEFERRED SIGNAL DELIVERY

delay

(wall clock)

```
python_time += interval
c_time += delay - interval
```

takesometime.py: % of time = 100.00% out of 1.03s.

Line	CPU % Python	CPU % native	Sys %	takesometime.py
1				`import time`
2				`import taketime`
3				
4				`c_its_per_sec = 250000`
5				`python_its_per_sec = 1080000`
6				
7				`import timeit`
8				
9				`def py_taketime(n):`
10				`    d = 1.01`
11				`    for i in range(n):`
12	7%	1%		`        for j in range(10):`
13	18%	3%		`            d = d * d + d * d`
14				`    return n`
15				
16				`c_secs = .7`
17				`python_secs = .3`
18				
19				`start = time.process_time()`
20	1%	70%		`p = taketime.run(int(c_secs * c_its_per_sec))`
21				`q = py_taketime(int(python_secs * python_its_per_sec))`
22				`end = time.process_time()`
23				`print(end-start)`

DEFERRED SIGNAL DELIVERY

delay

(wall clock)

```
python_time += interval
c_time += delay - interval
```

takesometime.py: % of time = 100.00% out of 1.03s.

≈30%

≈70%

Line	CPU % Python	CPU % native	Sys %	takesometime.py
1				`import time`
2				`import taketime`
3				
4				`c_its_per_sec = 250000`
5				`python_its_per_sec = 1080000`
6				
7				`import timeit`
8				
9				`def py_taketime(n):`
10				`    d = 1.01`
11				`    for i in range(n):`
12	7%	1%		`        for j in range(10):`
13	18%	3%		`            d = d * d + d * d`
14				`    return n`
15				
16				`c_secs = .7`
17				`python_secs = .3`
18				
19				`start = time.process_time()`
20	1%	70%		`p = taketime.run(int(c_secs * c_its_per_sec))`
21				`q = py_taketime(int(python_secs * python_its_per_sec))`
22				`end = time.process_time()`
23				`print(end-start)`

DEFERRED SIGNAL DELIVERY

delay

(wall clock)

```
python_time += interval
c_time += delay - interval
```

takesometime.py: % of time = 100.00% out of 1.03s.

Line	CPU % Python	CPU % native	Sys %	takesometime.py
1				import time
2				import taketime
3				
4				c_its_per_sec = 250000
5				python_its_per_sec = 1080000
6				
7				import timeit
8				
9				def py_taketime(n):
10				d = 1.01
11				for i in range(n):
12	7%	1%		for j in range(10):
13	18%	3%		d = d * d + d * d
14				return n
15				
16				c_secs = .7
17				python_secs = .3
18				
19				start = time.process_time()
20	1%	70%		p = taketime.run(int(c_secs * c_its_per_sec))
21				q = py_taketime(int(python_secs * python_its_per_sec))
22				end = time.process_time()
23				print(end-start)

≈30%

≈70%

error ≈1%

Other "Irrational exuberance" Languages

Scripting Language	**`malloc` interposition**	**Monkey patching**	**Thread enum.**	**Stack inspection**	**Opcode disassembly**
Perl	✓(1)	✓	`threads->list()`	`Devel::StackTrace`	`B::Concise`
Tcl/Tk	✓(2)	✓			
Python	✓(3)	✓	`threading.enumerate()`	`sys._current_frames()`	`dis`
Lua	✓(4)	✓			
PHP	✓(5)	✓			
R	✓	✓		`sys.call`	`disassemble`
Ruby	✓(6)	✓	`Thread.list`	`caller`	`RubyVM::InstructionSequence`

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING

CPU

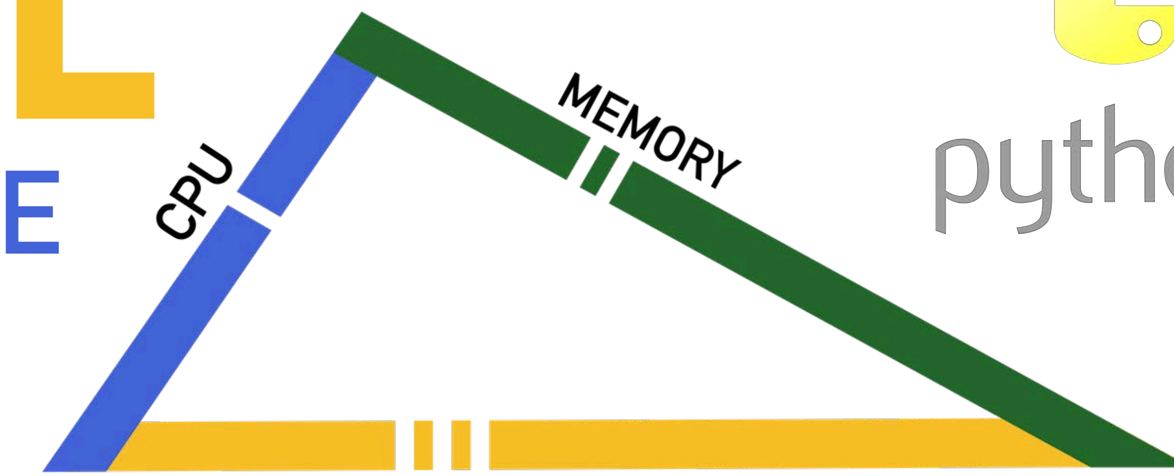
MEMORY

python™

github.com/emeryberger/**SCALENE**

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING



github.com/emeryberger/SCALENE

precise

CPU
PYTHON vs. NATIVE + SYS%

MEMORY
PYTHON vs. NATIVE

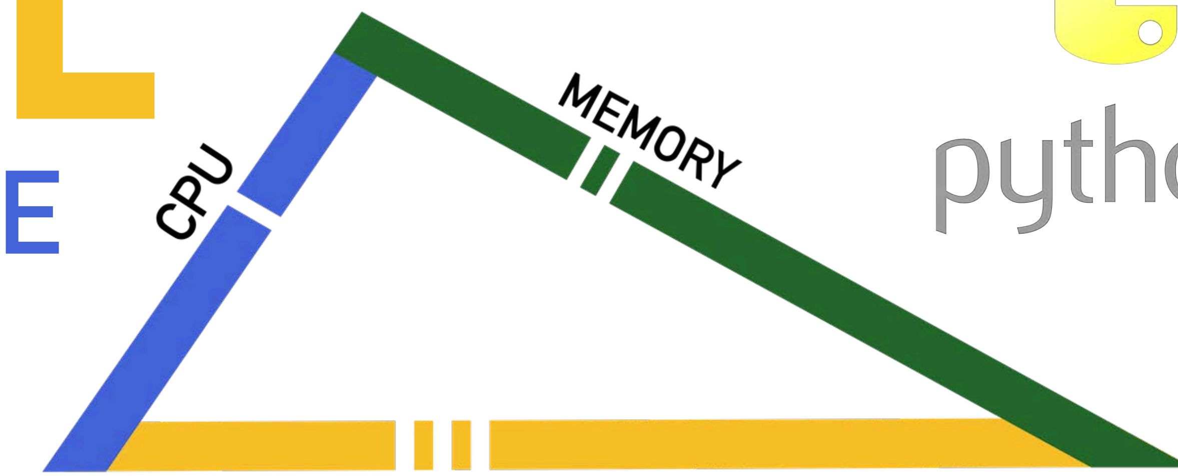
MEMORY
USAGE OVER TIME

% OF MEM ALLOCATED

COPY
VOLUME (MB/s)

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING



github.com/emeryberger/SCALENE

precise

CPU
PYTHON
vs. NATIVE
+ SYS%

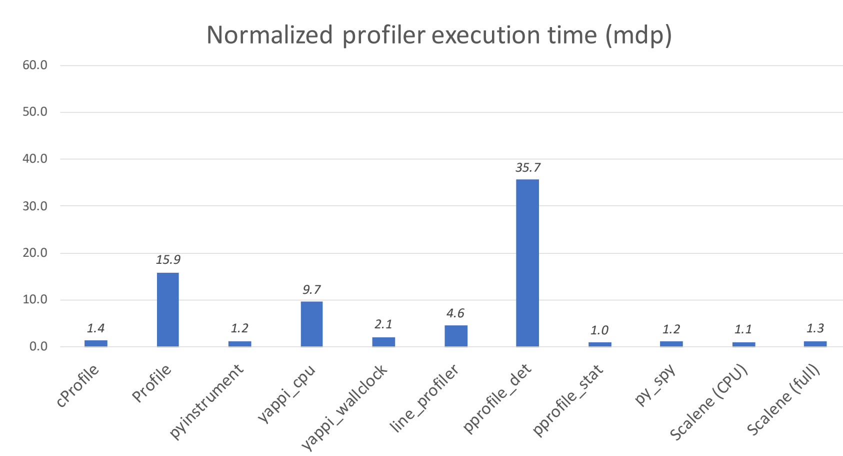
MEMORY
PYTHON
vs. NATIVE

MEMORY
USAGE
OVER
TIME

% OF MEM
ALLOCATED

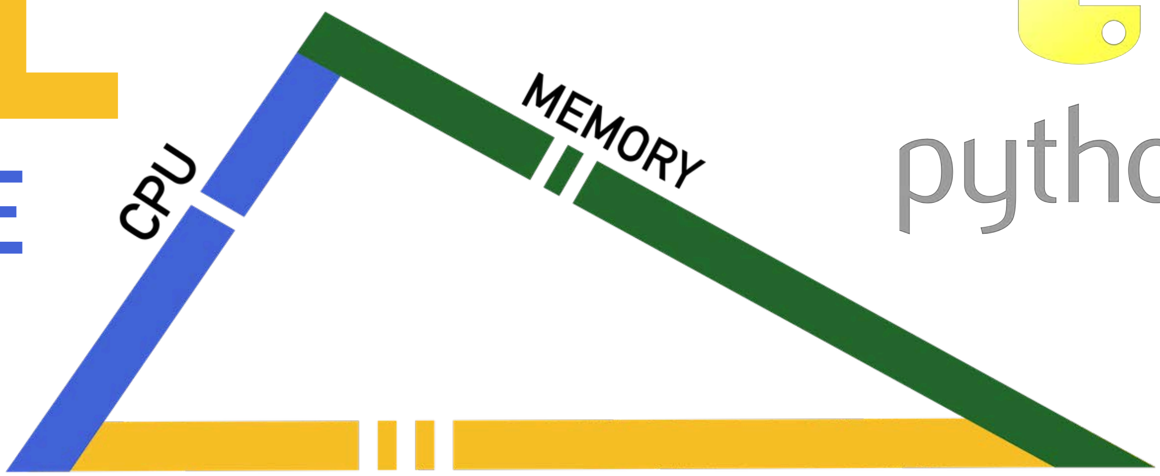
COPY
VOLUME
(MB/s)

fast



SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING



python™

github.com/emeryberger/SCALENE

precise

CPU PYTHON vs. NATIVE + SYS%

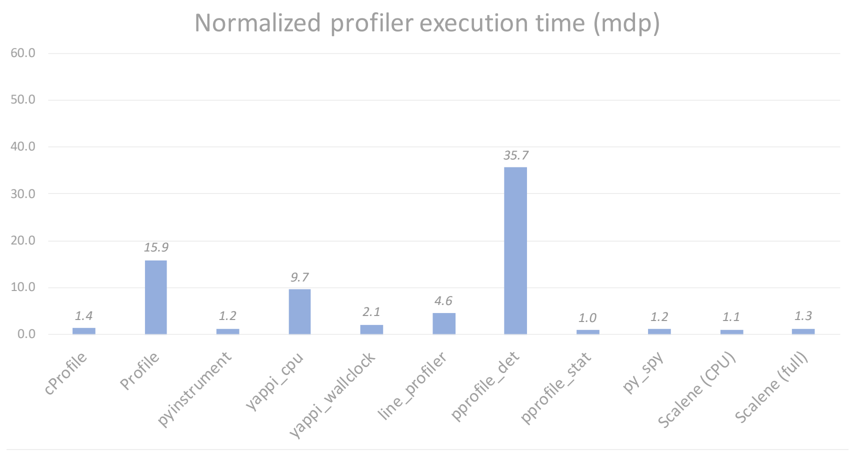
MEMORY PYTHON vs. NATIVE

MEMORY USAGE OVER TIME

% OF MEM ALLOCATED

COPY VOLUME (MB/s)

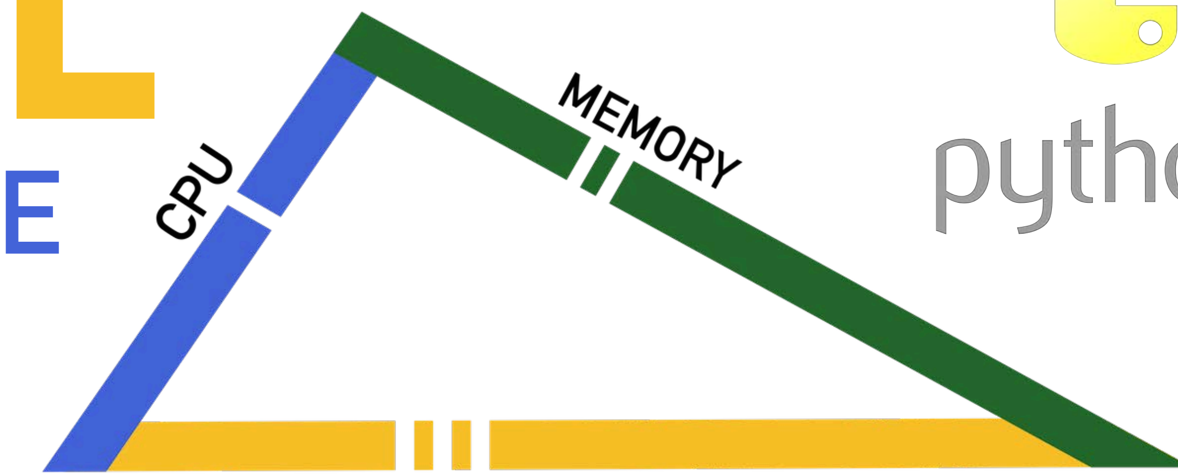
fast



% pip install -U scalene

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING



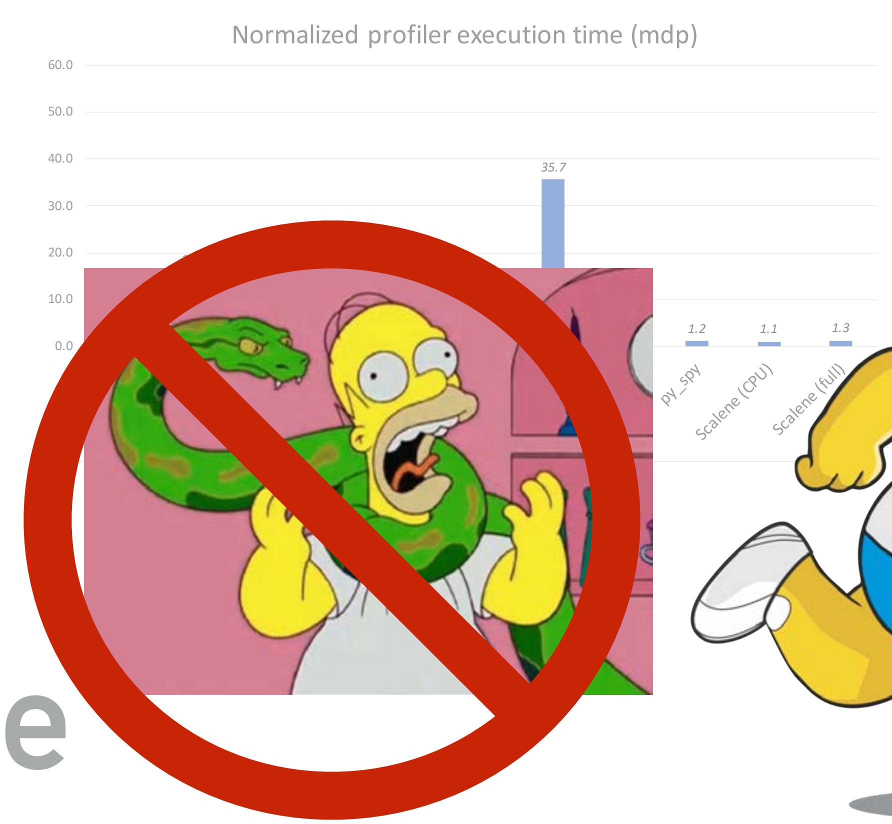
python™

github.com/emeryberger/SCALENE

precise

CPU	MEMORY	MEMORY	COPY
PYTHON vs. NATIVE + SYS%	PYTHON vs. NATIVE	USAGE OVER TIME	VOLUME (MB/s)
		% OF MEM ALLOCATED	

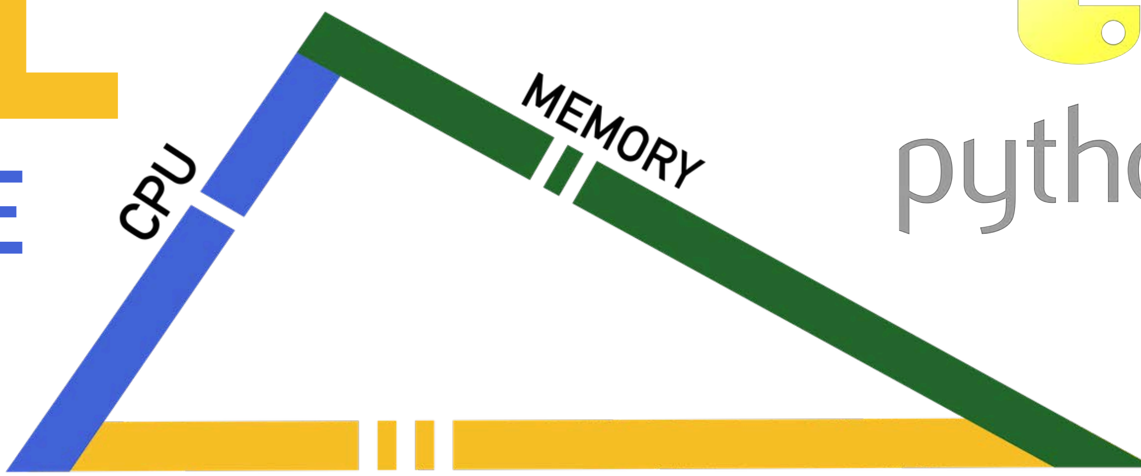
fast



% pip install -U scalene

SCALENE

SCRIPTING-LANGUAGE AWARE PROFILING



github.com/emeryberger/SCALENE

Effective Performance Profiling

Coz

```
% sudo apt install coz-profiler
```

github.com/plasma-umass/coz