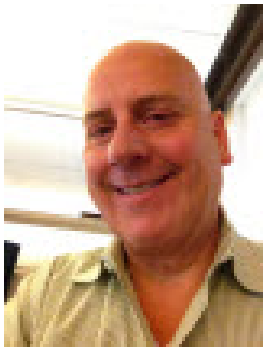
Condos and Clouds

Patterns in SaaS Applications

Thinking about Cloud Computing
by Looking at Condominiums



Presenter
Pat Helland
Salesforce.com

Moderator
Yannis Ioannidis
University of Athens

Extended Version: Jan 2012

ACM Learning Center

http://learning.acm.org

- 1,300+ trusted technical **books and videos** by leading publishers including O'Reilly, Morgan Kaufmann, others
- **Online courses** with assessments and certification-track mentoring, member discounts at partner institutions
- **Learning Webinars** on big topics (Cloud Computing/Mobile Development, Cybersecurity, Big Data, Recommender Systems)
- **ACM Tech Packs** on big current computing topics: Annotated Bibliographies compiled by subject experts
- Learning Paths (accessible entry points into popular languages)
- Popular video tutorials/keynotes from ACM Digital Library, Podcasts with industry leaders/award winners

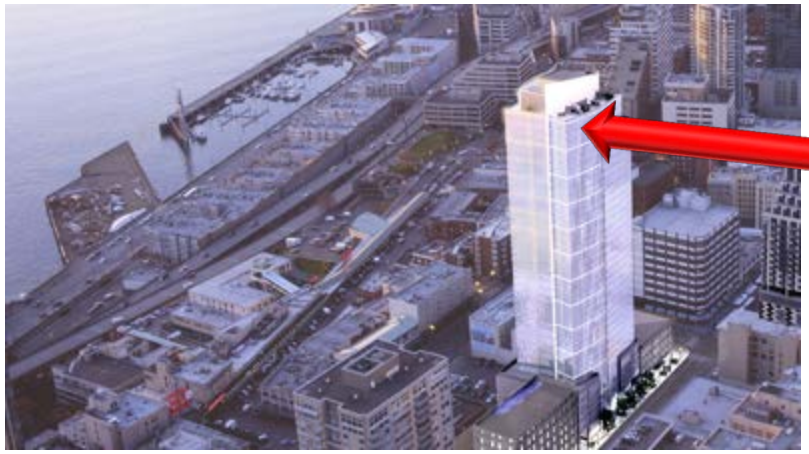
Association for Computing Machinery

Advancing Computing as a Science & Profession

Outline

- Introduction
- Patterns in SaaS Applications: the Front-End
- Patterns in SaaS Applications: the Back-End and Decision Support
- Multi-Tenancy: Making It Work
- Conclusion

I Live in a Condo…



Our home (until about a year ago)!
Great view…. But my ears popped going home!

- Not everyone wants to live in a condo
  - Not good for kids playing in the backyard, dogs running in the backyard, working on cabinetry in your garage, or having a garden
- But it has its advantages!
  - Common heating/air-conditioning "just works"
  - Infinite supply of hot water for long showers
  - Someone else takes the trash out "to the street"
  - It comes with a building engineer who fixes most things!

Sometimes, It's Really Nice to Outsource a Bunch of Hassles

In Exchange, You Live with Some Constraints

Constraints and Concierge Services

Constraints and Concierge Services in Buildings

Building Type	Services	Constraints
Housing	• Reservations • Package/Dry-Cleaning • Shared Exercise & Pool • Shared Engineering	• Limits on Parking, Noise, Pets, & BBQing • No Garage Projects • No Gardening Projects
Office	• Shared Bathrooms • Shared Lobby • Shared Copiers/Coffee (?) • Shared Engineering	• Fixed Office Layout • No Sleeping at Work • (Typically) No Pets • (Maybe) Dress Code
Retail Mall	• Shared Engineering • Shared Parking • Shared Common Space • Shared Security	• Common Mall Hours • Only Retail Activities • Probably Constraints on the Type of Retail

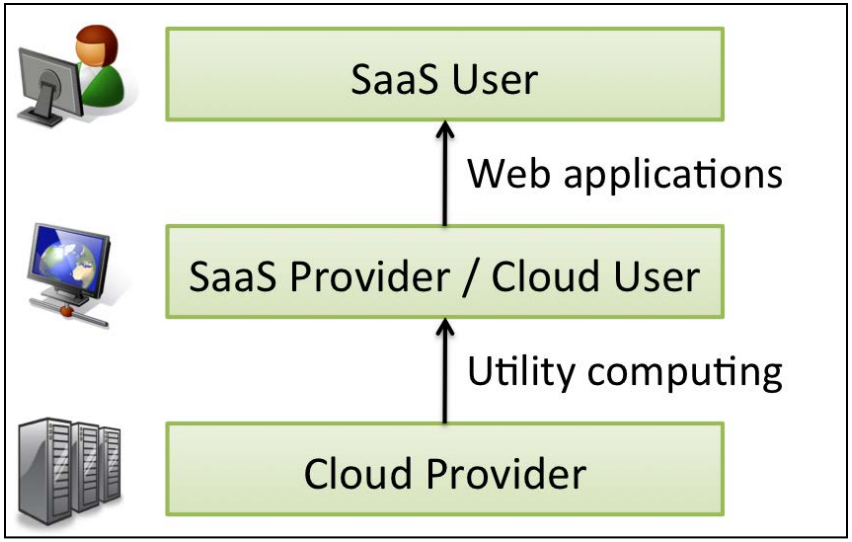
What Are the Constraints and Concierge Services in Cloud Computing?

What Can the Shared Infrastructure Do to Make Life Better for a Sharing App?

What Constraints Must a Sharing App Live within to Fit into the Shared Cloud?

What Is Cloud Computing?

- **Cloud Computing:** Apps delivered as services over the Internet/Intranet
  - **Saas: Software as a Service:** App software delivered over the Net
  - **Utility Computing:** Virtualized hardware and computing delivered over the Net



Three New Aspects to Cloud Computing

The Illusion of Infinite Computing Resources Available on Demand

The Elimination of Upfront Commitment by Cloud Users

The Ability to Pay for Use of Computing Resources on a Short-Term Basis

Cloud Computing Allows Deploying *Software as a Service* – and Scaling on Demand – without Building or Provisioning a Datacenter

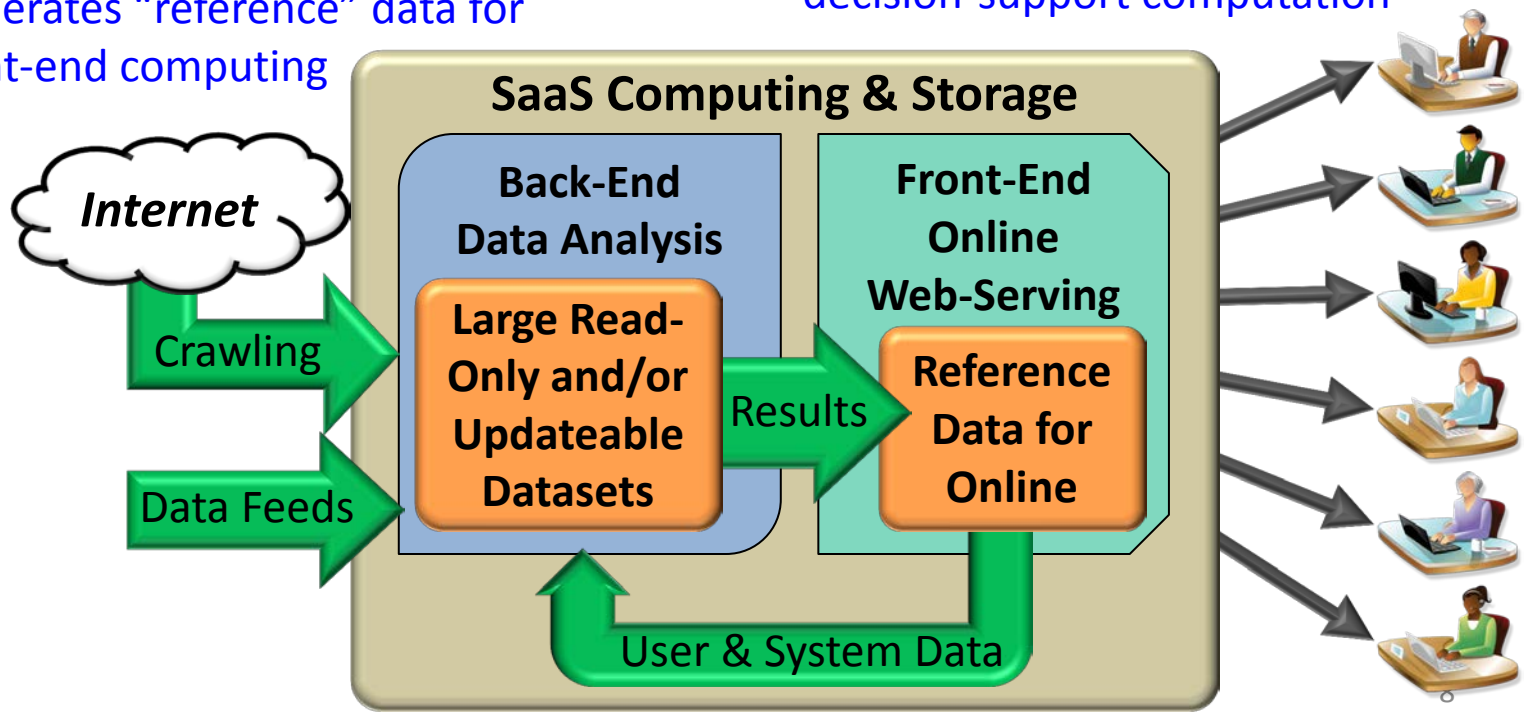
See: *Above the Clouds: a Berkeley View of Cloud Computing*, Feb 2009

The Forces Driving Us to the Cloud

Datacenter Economics	**Utility Computing** • Large Datacenters Are Very Expensive • Sharing Reduces Costs • Chip Foundry Outsource	**Efficiencies of Size** • Electricity where Cheap • Network on Main Lines • Containerized Servers • Shared Standard Admin
Shared Data	**Common "Big-Data" Store** • Analyze Anything-to-Anything for "What-If?" • Value of Serendipitous Discovery Grows with Data Size • Store ALL of an Enterprise's Data in a Common Store • Store the Data, Allow Analysis, and See Surprising Value	
Shared Resources	**Higher Utilization** • Shared Usage of Computation & Storage • Do Low Priority Work During Slack Times	**Stronger SLAs** • High Priority Work Can Preempt Low Priority • Fluid and Fungible Resource Usage

Front-End, Back-End, and Decision-Support

- Front-End Online processing
  - User-facing services called by web-service or HTML
  - SLAs typically less than 500ms
- Back-End processing
  - Consumes crawled data, partner feeds, and logged information
  - Generates “reference” data for front-end computing
- Decision-Support
  - Ad-hoc and planned analysis of the datasets contained in the back-end
- Integrated processing
  - Unclear distinction between back-end processing and decision-support computation

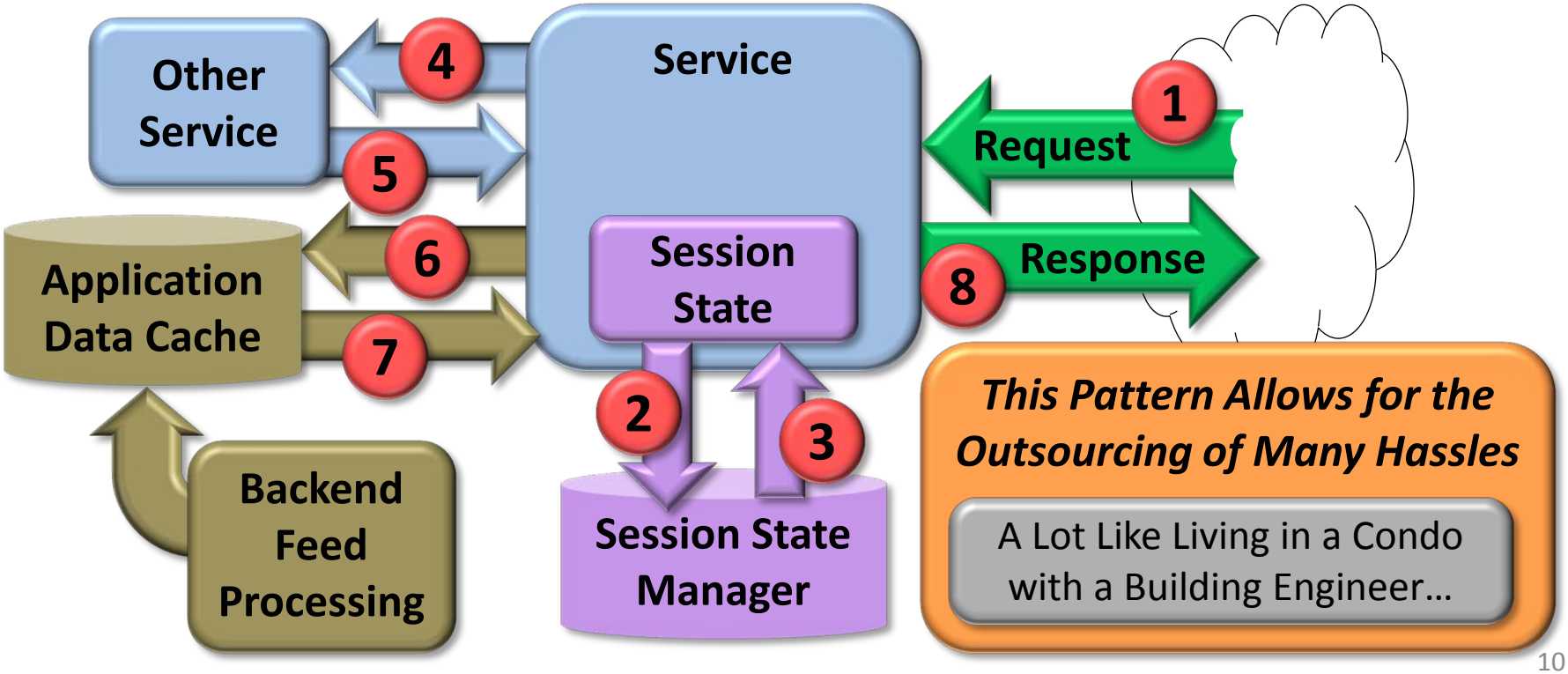


Outline

- Introduction
- Patterns in SaaS Applications: the Front-End
- Patterns in SaaS Applications: the Back-End and Decision Support
- Multi-Tenancy: Making It Work
- Conclusion

Many Service Apps Fit a Pattern

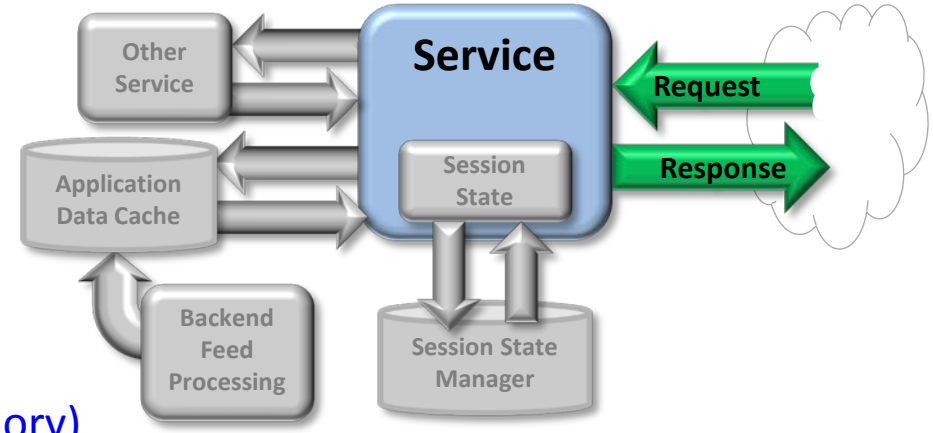
- Many Service applications follow this pattern:
  - Requests come in from the web to a service
  - You optionally fetch state associated with this session
  - The app may or may not invoke other services
  - The app may access a data cache populated from feeds/back-end processing
  - A response is sent back to the user



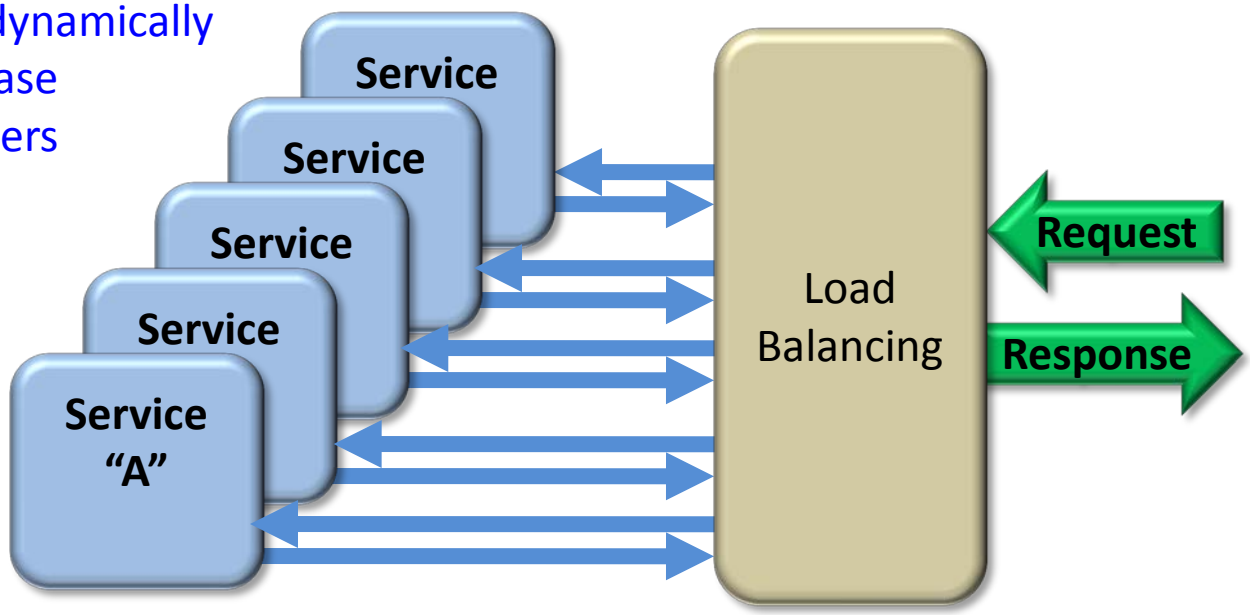
Concierge Services for the Front-End

Auto-Scaling	As the workload rises, additional servers are automatically allocated for this service. Resources taken back when load drops.
Auto-Placement	Deployment, migration, fault boundaries, and geographic transparency are all included. Applications are blissfully ignorant.
Capacity Planning	Analysis of traffic patterns of service usage back to incoming user work load. Trends in incoming user workload are tracked.
Resource Marketplace	Plumbing tracks a service's cost as it directly consumes resources and indirectly consumes them (by calling other services).
A/B-Testing and Experimentation	Plumbing makes it easy to deploy a service on a subset of the traffic and compare the results with the previous version.
Auto-Caching / Data Distribution	Data is fed into a datastore and processed there. This processed data is cached for easy access by services.
Session State Management	User session information is captured before a service completes. The next request easily fetches the state to use.

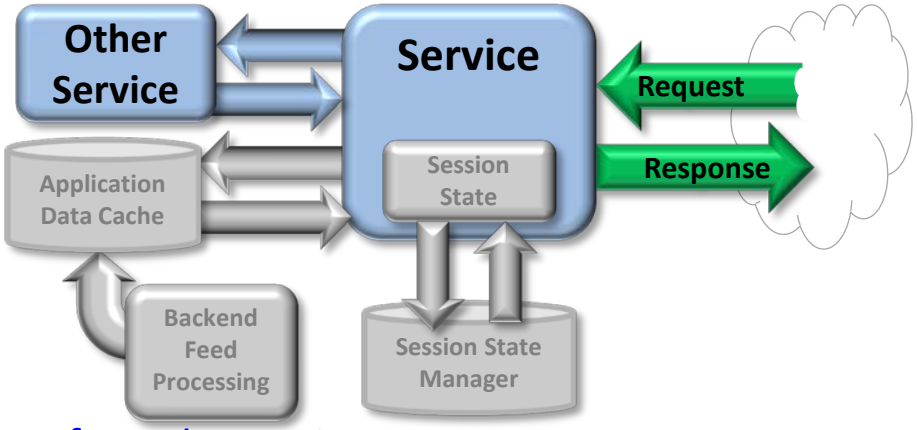
Stateless Request Processing



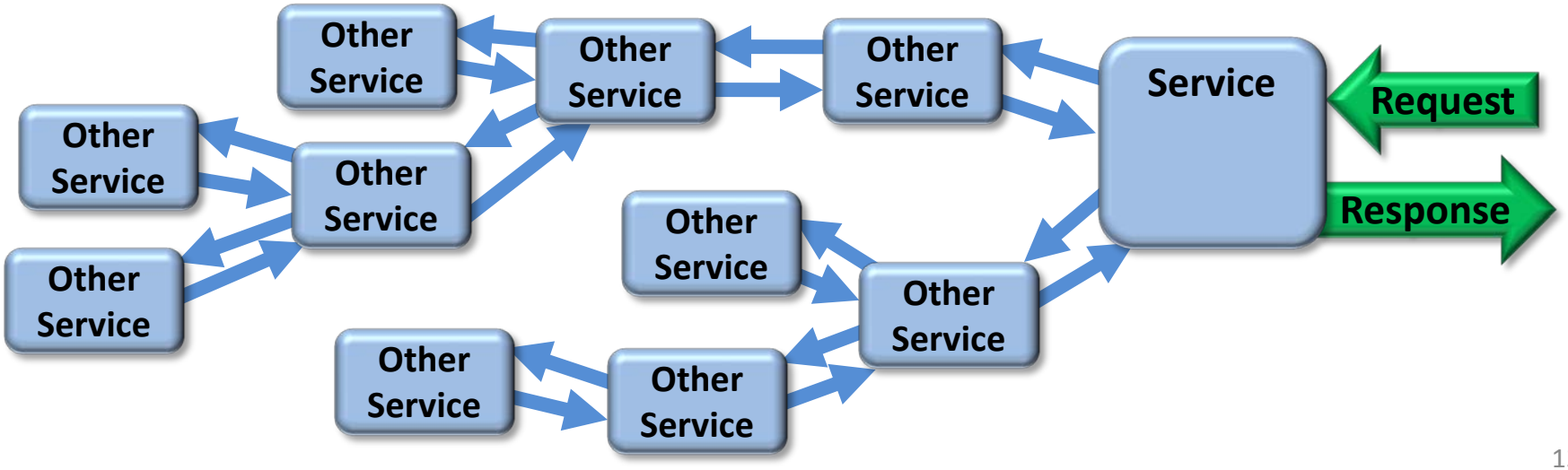
- Incoming requests are routed to one of the copies of the service
  - On arrival, there is no state (or memory) associated with the session
  - *At this point*, we consider the service to be stateless (it may get state later)
- The plumbing keeps a pool of servers capable of implementing the service
  - Incoming requests are dynamically routed to these services
  - The plumbing will dynamically increase and decrease the number of servers implementing this service as needed



Composite Request Processing



- Frequently, a service calls other services to get its job done
  - These may or may not grab session state from the session state manager
- The composite call graph may get complex
  - Sometimes there will be cycles
    - The service must know when to break the cycle
  - Sometimes the depth of the call graph may get very deep
- All of these service calls will need to complete to build the user response
  - The work fans out, processes, and fans back



SLAs and Request Depth

SLA: Service Level Agreement

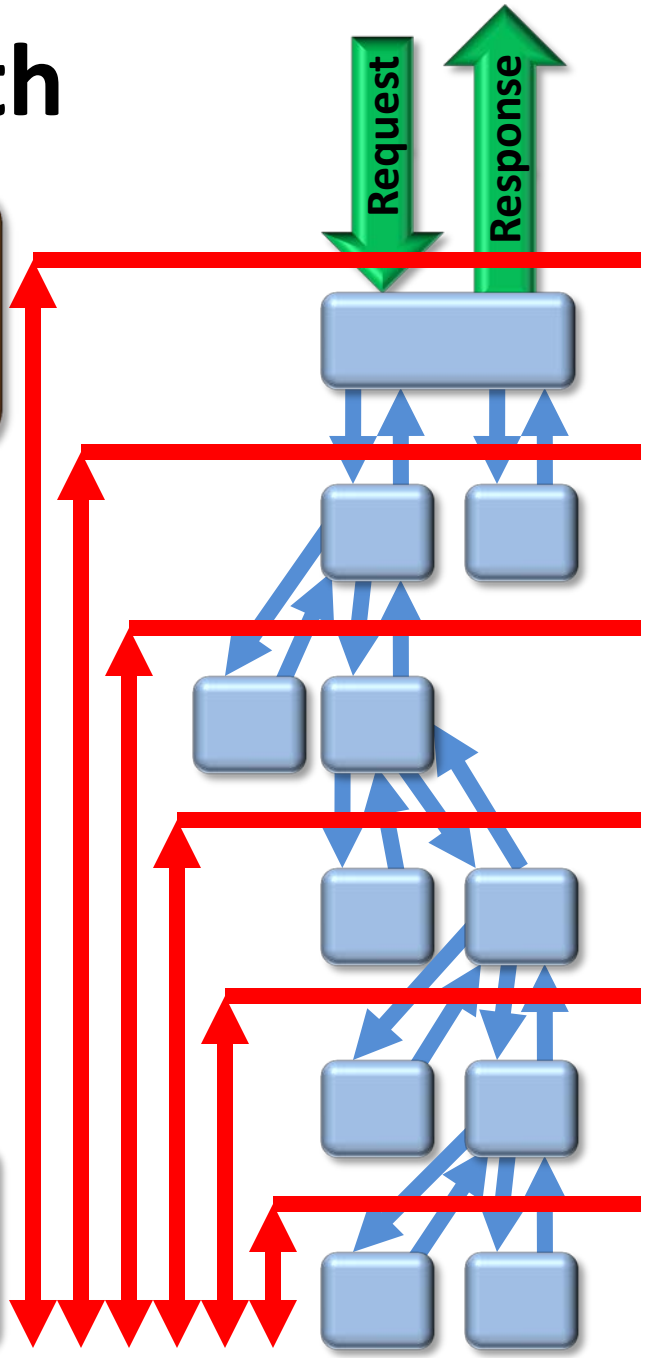
Does the "Service" Provide the Level of Service It Promised?

Is the Response Fast Enough?

- System-wide SLAs need tight component SLAs
  - Multi-level call graph in service structure
- Example SLA:
  - 300ms response for 99.9% of requests with 500 requests per sec
- Cross-service dependencies
  - Force tight SLA responses
- Average versus Percentile
  - Average not strict enough
  - User experience unacceptable

Lots of Pressure on Services at the Bottom of the Stack!

You Need to Answer Fast and Predictably!



Pounding on the Services at the Bottom

- The Deeper the Call Stack, the Tighter the SLA
  - It's Time Is Factored in the Caller's SLA
- Consider Session-State Manager and Auto-Cache Manager
  - They get called a lot
  - They get called from services already deep in the stack!

Deeper the Call Stack → More Pressure

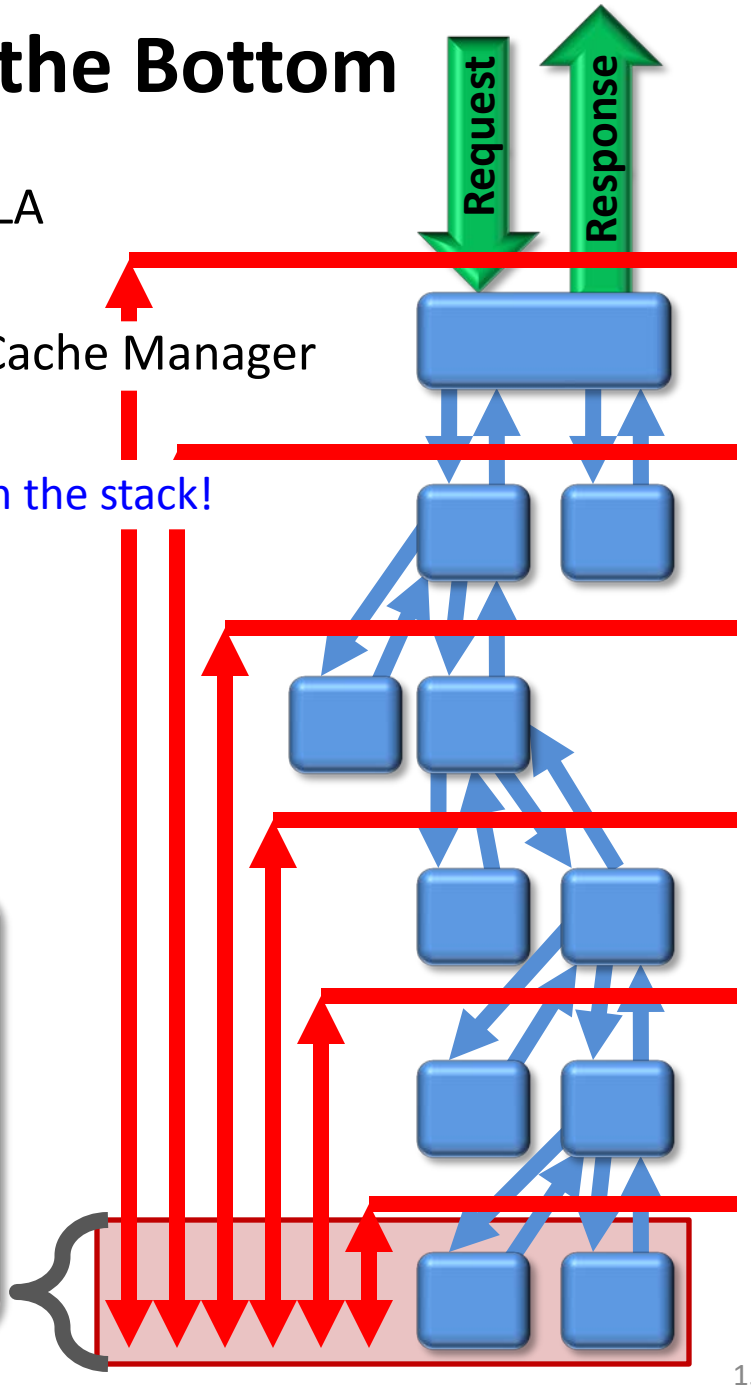
Can Result in Needing to Sacrifice Utilization!

How Are You Going to Provide a Tight SLA?

Short Minimum Response Time?

Really Low Utilization?

Both?...



Automatic Provisioning to Meet SLAs

- The plumbing can know the desired SLA for each service it manages
  - End-user facing services can have their SLAs configured to the plumbing
  - Plumbing can know which services call which other services
    - This can be determined dynamically by watching the interactions
  - Based on the demands of the calling services, a desired SLA can be calculated

It Is Unlikely that SLA Goals Can Be Fully Automated

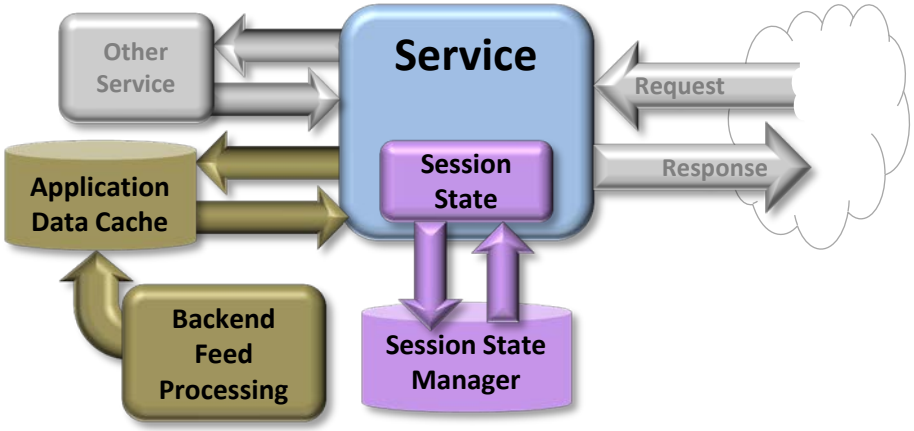
When a Calling Service Calls Many Different Dependent Services,
What Is the Relationship across Their SLAs?

- Within limits, the SLA for a service can be met by increasing the number of servers and, hence, decreasing the utilization of the server pool
  - The plumbing can dynamically increase the server pool to meet the SLA

This Technique Is Useful Sometimes but Not for Everything

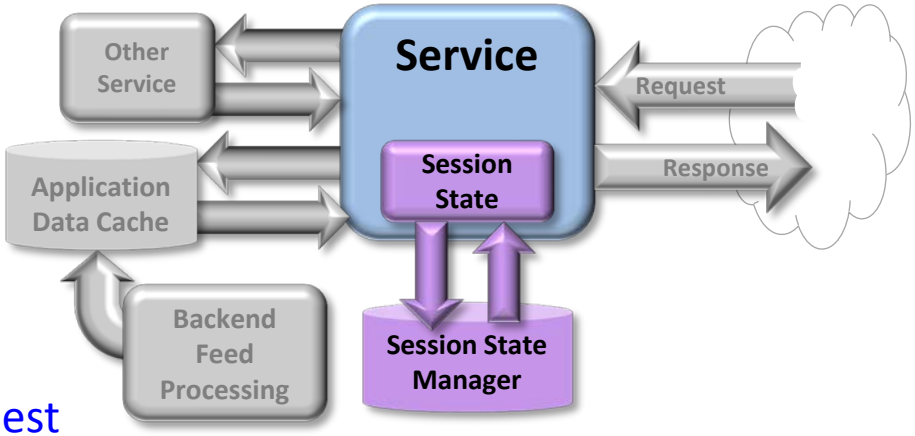
The Minimum Response Time for the Service Must Be Acceptably
Close to the Desired SLA for the Service

Accessing Data and State



- When a request lands into a service, it initially has no state other than what arrives with the request
  - It can fetch session state
  - It can fetch cached data for some application specific data item based on key
- *Session state*: fetched from the session state manager using a session key
  - When changes are made, they are stored back to the session state manager
- *Cached data*: fetched from the cache manager with domain name and key
  - Used to store data which is derived from feeds
  - This application (service) specific data is *read-only by the service*

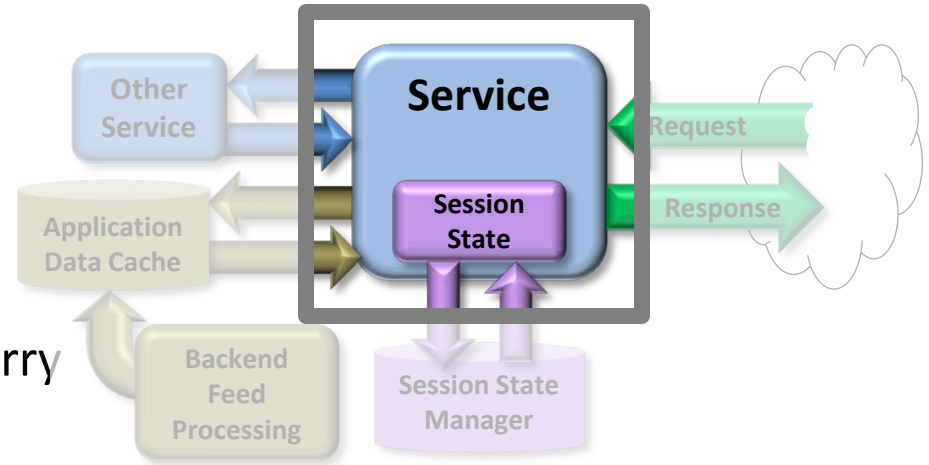
Managing Scalable and Reliable State



- The *Session State* is keyed by the session-id
  - The session-id comes in on the request
  - When the service calls the session-state-manager, a blob of state is returned
- The plumbing manages scaling the session state
  - The session state manager will expand as necessary when the number of sessions grows
- The plumbing manages the durability of the session state
  - It is essential that the state usually survives system failures
  - We should consider schemes which *rarely* lose updates and gain performance
- The plumbing must give excellent responsiveness

Typical Requirement: 5 ms Response 99.9% of the Time
in a First-Class Implementation of the Session State Manager

Automatic Services, State, and Data



- Applications using the plumbing worry about their business code
  - They have session state, application data cache, and calls to other services
  - They simply implement their own business logic, not system issues
- These applications build their apps in two pieces:
  - Back-end feed/crawl processing:
    - Data from feeds and crawls is batch processed to make cache entries
  - Front-end web-serving:
    - Services awaken, (optionally) fetch session state, access cache, call other services, calculate their business logic, and return responses

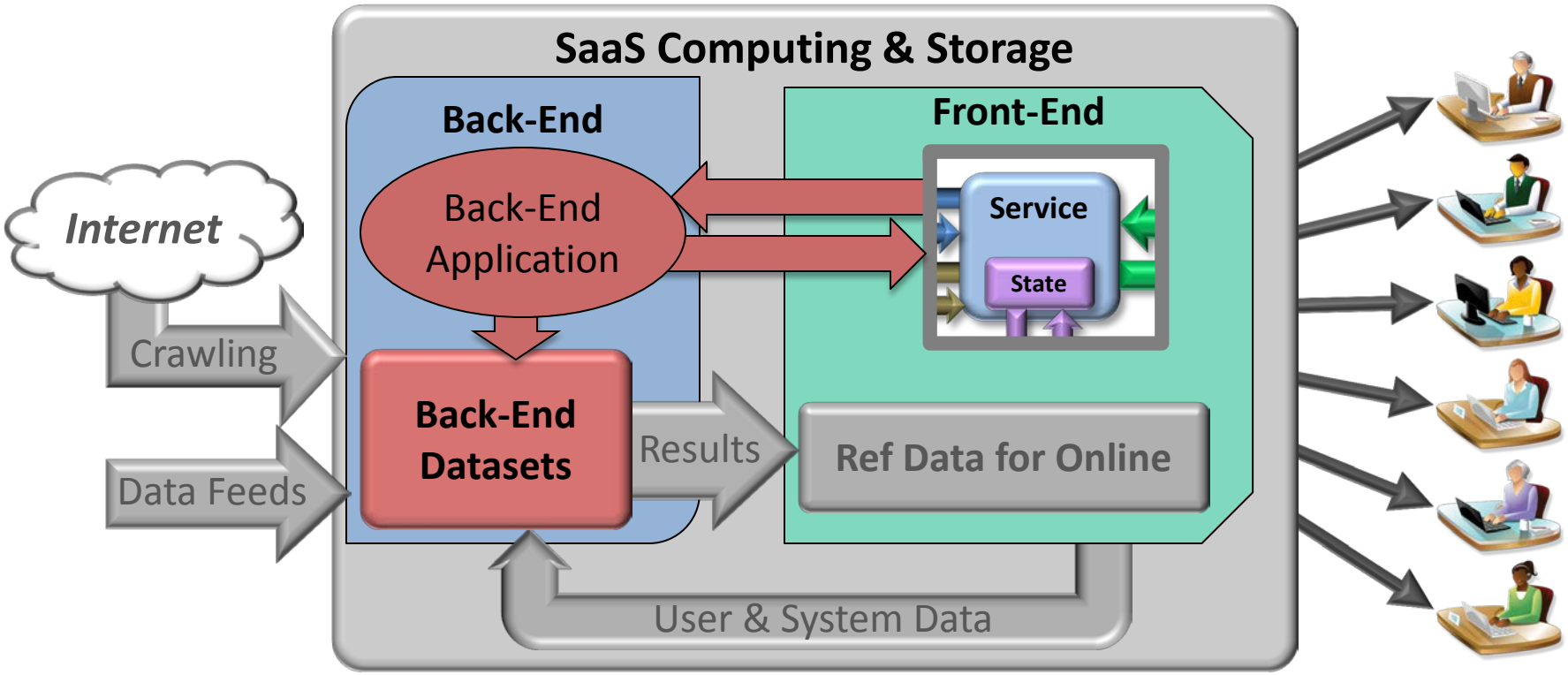
The Plumbing Prescribes HOW to Access These Services

Session State, Cache, and Service Calls Are Done with Special Interfaces

Constrained Application Functionality So the Plumbing Can Provide the Support

Applying Changes to the Back-End

- Sometimes request actually "do-work" and apply application changes
  - For example, pushing "Submit" while shopping at Amazon.com
- Application changes may be synchronous or asynchronous
  - **Synchronous:** the human waits while the back-end gets work done and answers
  - **Asynchronous:** the work is enqueued and processed later
    - Amazon sends an email when the work is completed

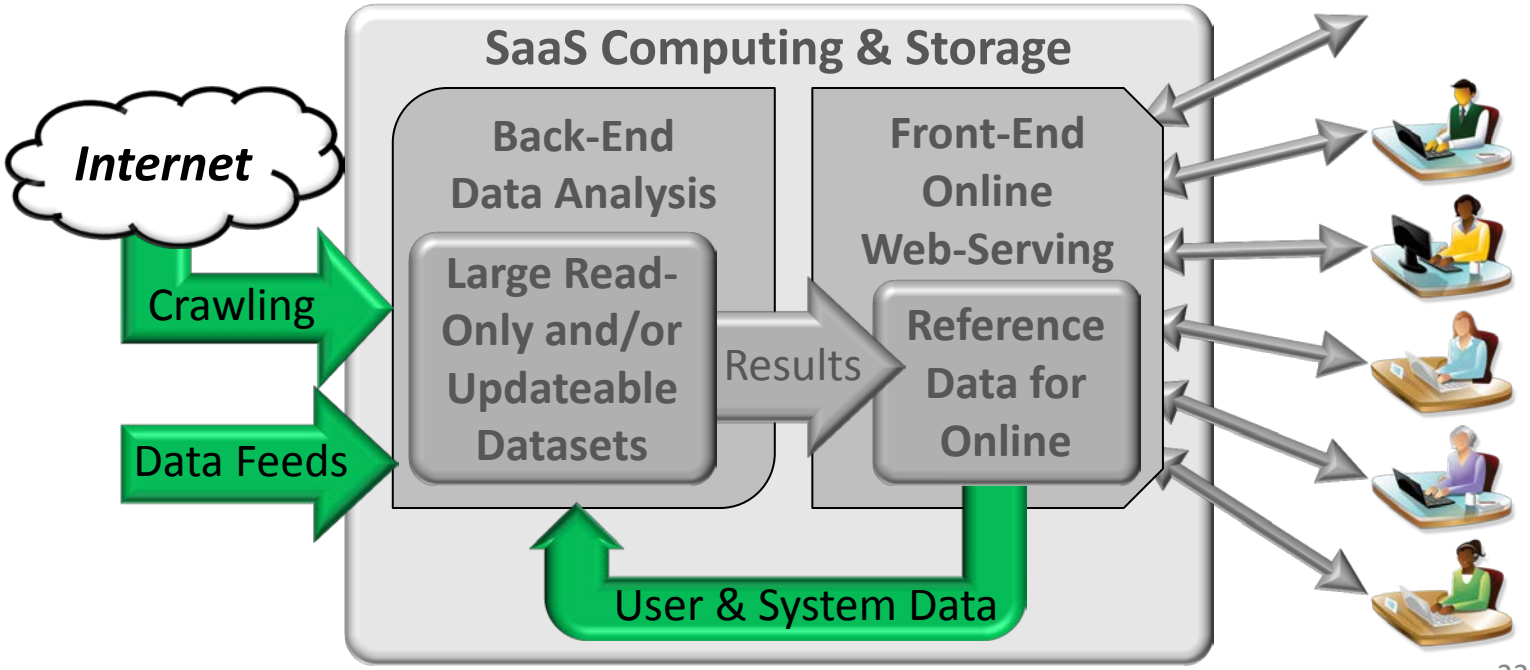


Outline

- Introduction
- Patterns in SaaS Applications: the Front-End
- Patterns in SaaS Applications: the Back-End and Decision Support
- Multi-Tenancy: Making It Work
- Conclusion

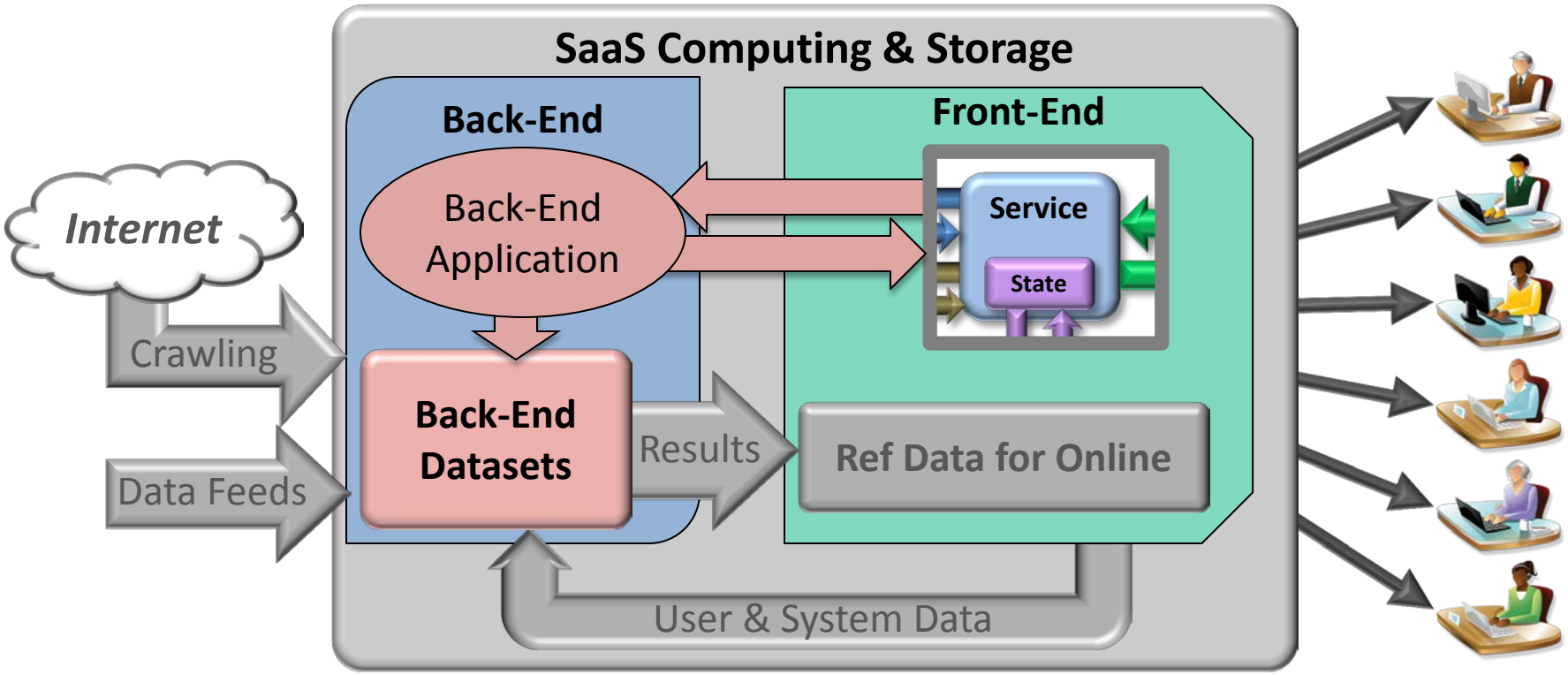
Feeds, Crawling, and Logging

- The Back-End receives data from many sources:
  - **Crawling:** sometimes the back-end has applications which look out at the Internet or other systems to see what can be extracted
  - **Data Feeds:** Partner companies or departments may send data to be ingested into the back-end system
  - **Logging:** Data is accumulated about the behavior of the front-end system. These logs are submitted for analysis by the back-end system



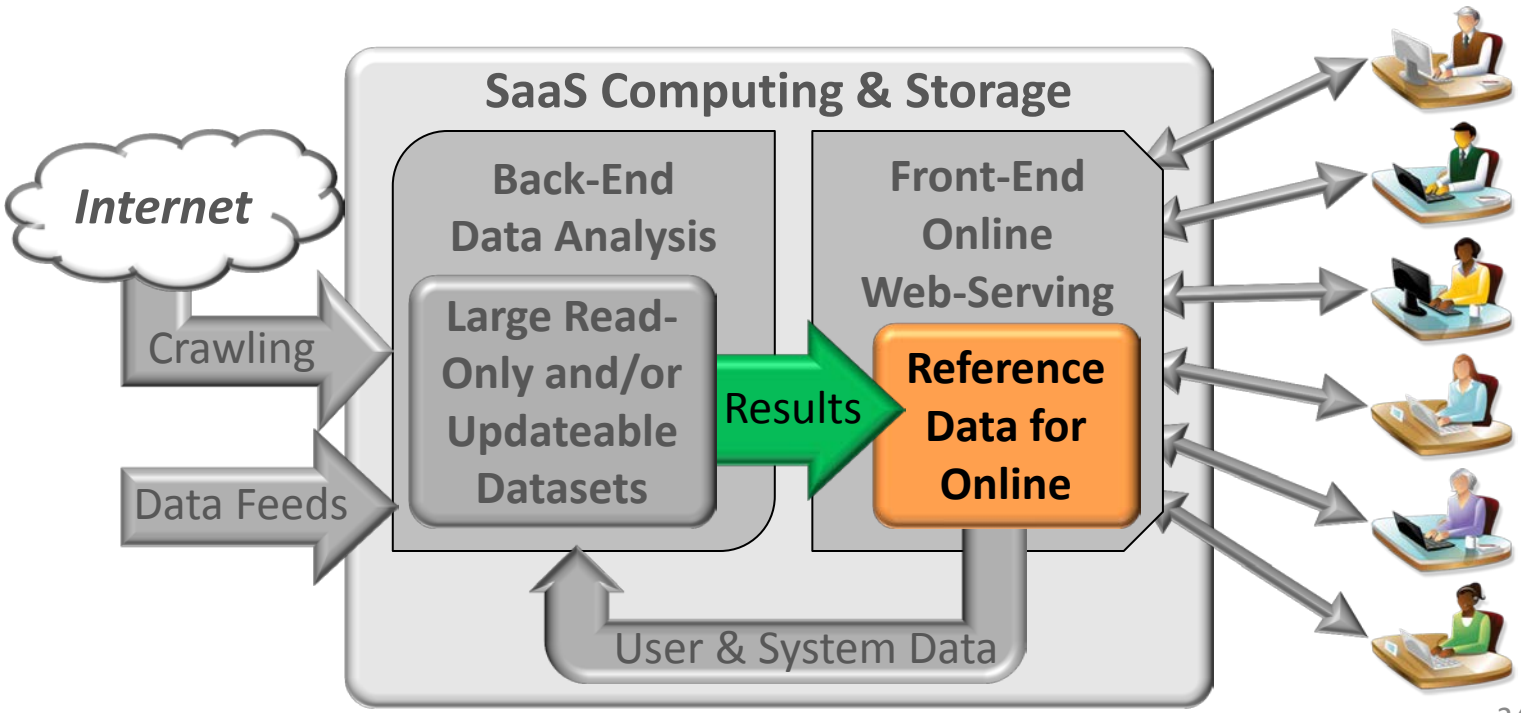
Back-End: Applying Online Work

- Sometimes request actually "do-work" and apply application changes
  - For example, pushing "Submit" while shopping at Amazon.com
- Application changes may be synchronous or asynchronous
  - **Synchronous:** the human waits while the back-end gets work done and answers
  - **Asynchronous:** the work is enqueued and processed later
    - Amazon sends an email when the work is completed



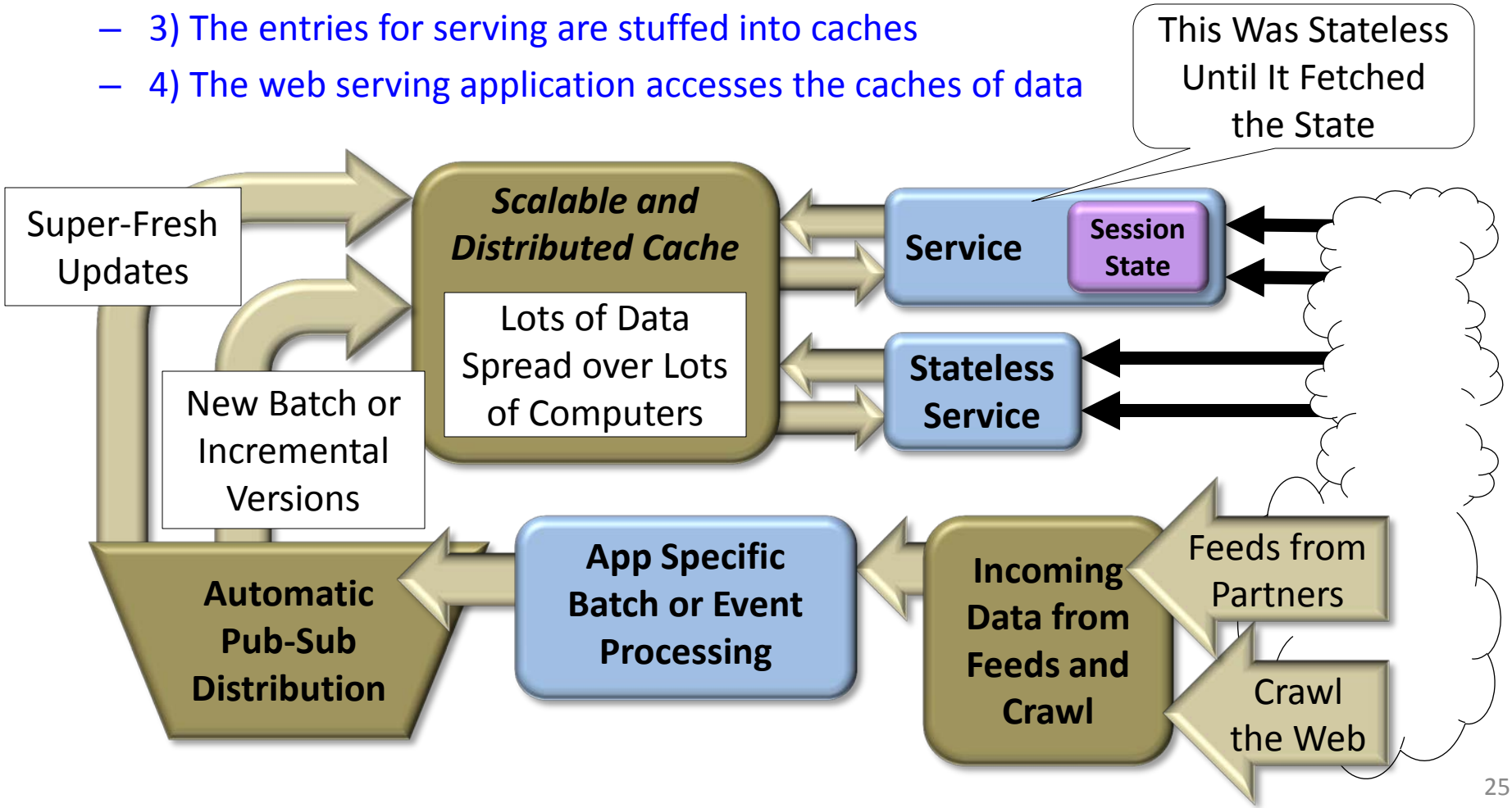
Generating Reference Data for the Front-End

- Many front-end applications use reference-data
  - Reference data is periodically updated by the back-end
  - Applications are designed to deal with reference data that may be stale
- Example uses of reference data:
  - Product catalog & price lists: Online retailing like Amazon.com uses this
  - Search indices for Google, Bing, or enterprise-specific search
  - Maps, insurance rates, ICD9 codes (medical diagnostic codes), and much more



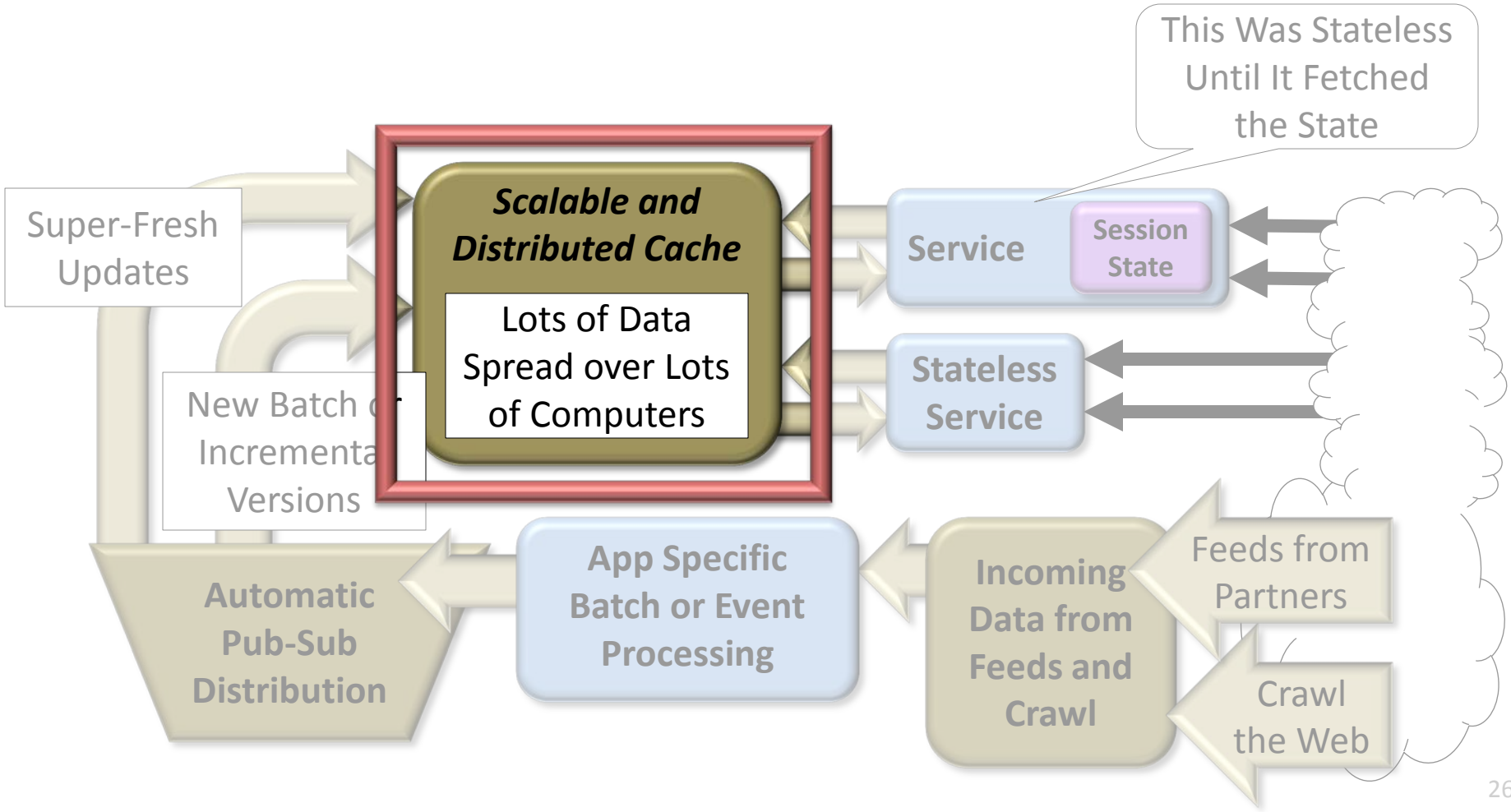
Data Publication and Updates

- The general model for data is:
  - 1) Backend processing receives data from the web by feeds or by crawling
  - 2) Application code on the backend munches the data making entries to serve
  - 3) The entries for serving are stuffed into caches
  - 4) The web serving application accesses the caches of data



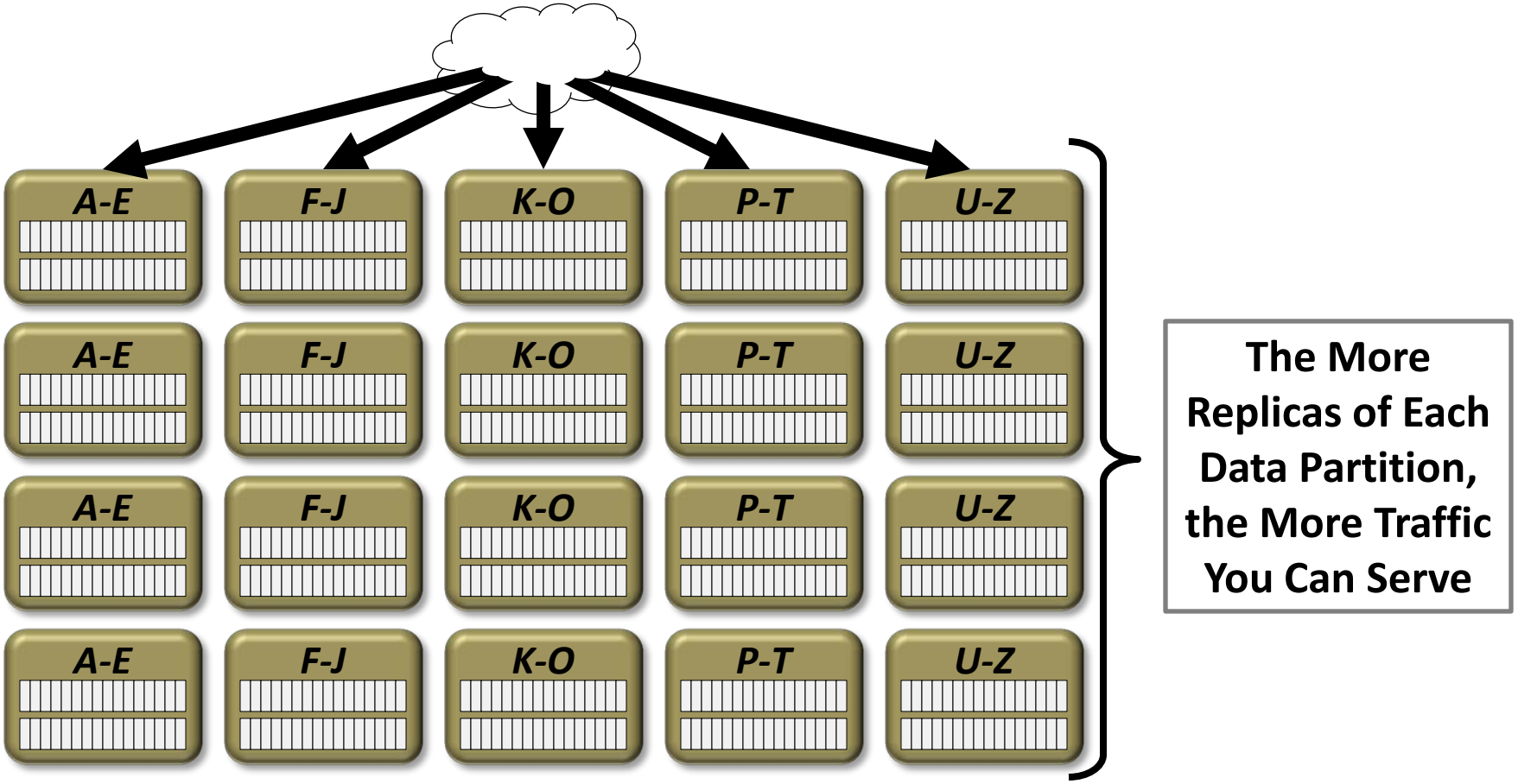
Focusing on the Scalable & Distributed Cache

- Let's zoom in on the reference-data cache stuffed by the back-end and used by the front-end

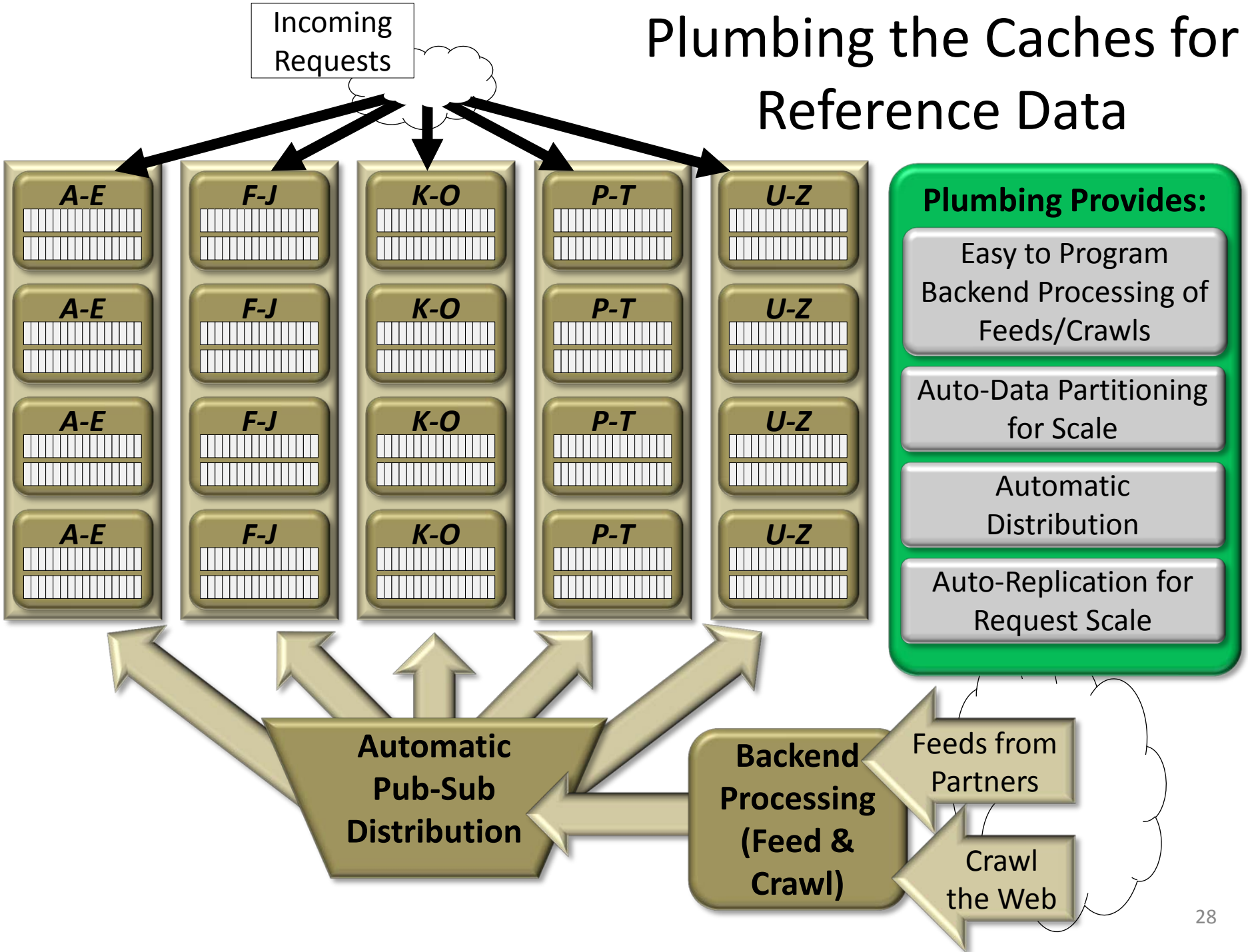


Partitioning and Replication of Reference Data

- Caches for reference data scale in two ways:
  - Partitioning: the data size may need to scale and the partitioning increases
  - Replication: the requests rate may increase and you need more processing



Plumbing the Caches for Reference Data



Styles of Back-End Processing (1)

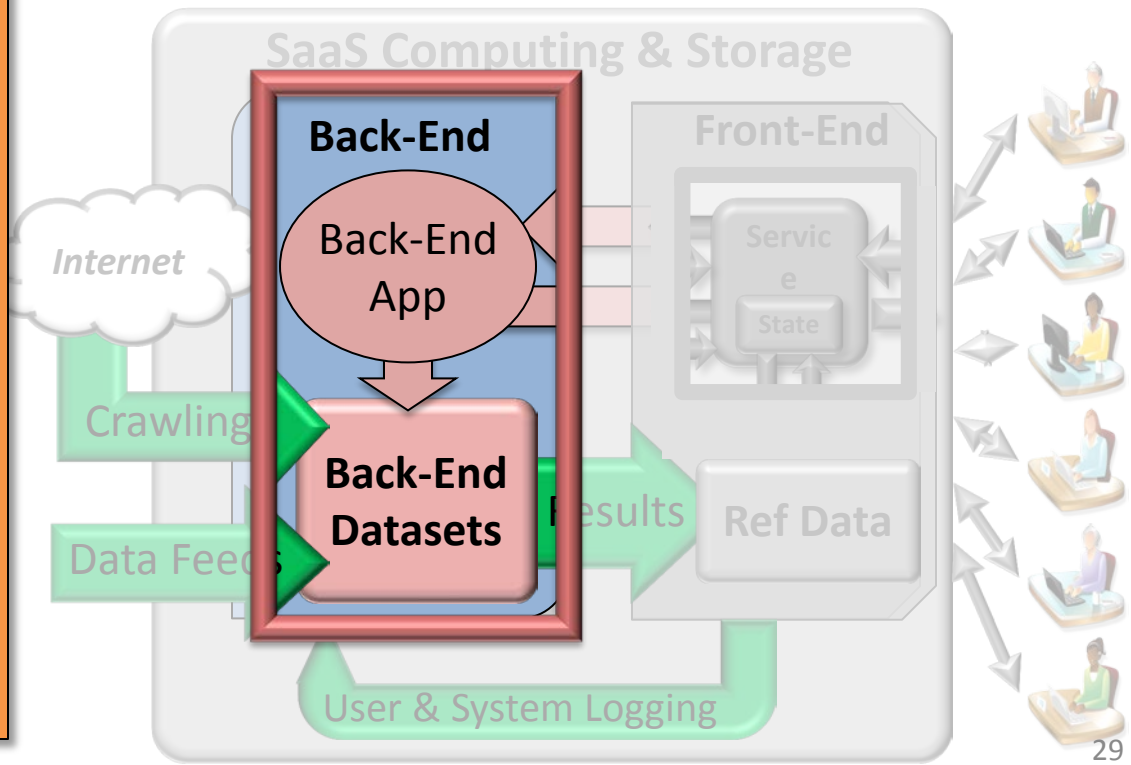
- Back-end input:
  - Crawling, Data Feeds, User & System Logging, and Front-end Calls
- Back-end output:
  - Reference Data Caches, Front-end Responses, Analysis Results, and Output Data Feeds to Others

Implementation Styles

Relational DB & Normal App

"Big-Data" & Parallel Batch

"Big-Data" & Event Pub-Sub



Styles of Back-End Processing (2)

Implementation Styles

Relational DB + Normal App

Application may be:

- DB Trigger: Running in the DB
- N-Tier: Running using VMs
- Other App: Running using VMs

Relational Data Replicated to "Big-Data" Store

Advantages of Relational… Only scales to a Single DB

"Big-Data" & Parallel Batch

Set-Oriented Massively Parallel Batch Processing

- Effectively no limits on the amount of data processed
- Declarative high-level language (easy for analysts)
- Can produce reference data or for BI Analytics

MapReduce/Hadoop over All the Enterprise's Data

"Big-Data" + Event Pub-Sub

Incoming Work Processed within Seconds

- Incoming events enqueued for pub-sub distribution
- Application code processes each subscription event
- Massively parallel transactional record updates

Rapid Processing of Events into Fresh Reference Data

Concierge Services for the Back-End

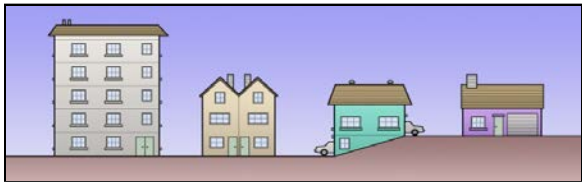
"Big-Data" Unified Data Access	Unified enterprise-wide (and controlled cross-enterprise) data. Anything may be processed with anything (if authorized).
Relational DB for Silos/Services	Relational DBs supporting enterprise apps which work as silos or services. ETL to stage data into the "Big-Data" store.
Fault Tolerant & Scalable Storage	Cloud managed storage for both "Big-Data" and relational. Automatic intra-datacenter and cross-datacenter replication.
Massively Parallel Batch & Event	High-level set-oriented operations. Incoming events call pub-sub style apps which transactionally update "Big-Data".
Automatic Scalable Ref-Data Caching	The Back-End supplies the Front-End with dynamically updated application specific data. Automated high-performance caching.
Multi-Tenanted Access Control	Intra (and inter) enterprise access control to data contained in the "Big-Data" store and the relational store
Prioritized SLA Driven Resources	Relational DBs, Batch, and Events compete for the same stuff. Work is given its SLAs and priorities. Tradeoffs are automated.

Outline

- Introduction
- Patterns in SaaS Applications: the Front-End
- Patterns in SaaS Applications: the Back-End and Decision Support
- Multi-Tenancy: Making It Work
- Conclusion

Landlords, Management Staff, Keys, and Privacy

- In rented apartments, owned condos, or rented houses, the landlord or engineering staff typically have a key to your home
  - There are well established reasons they may enter
  - They can (possibly) get into a world of shit if they come in for other reasons…



- Renting or leasing a property (home, office, retail, or light manufacturing) implies a bidirectional trust relationship
  - Tenants will follow the rules, pay the rent, not trash the property, and not bug the neighbors
  - Landlords will grant access to the property, keep the services running, and respect the privacy of the tenants
- Landlord/Tenant laws and rules took years to establish
  - They are even more complex in condominiums with shared ownership
  - There are different laws and expectations for different usage patterns (e.g. housing, retail, office, …)

Public, Private, Sharing, and Trust

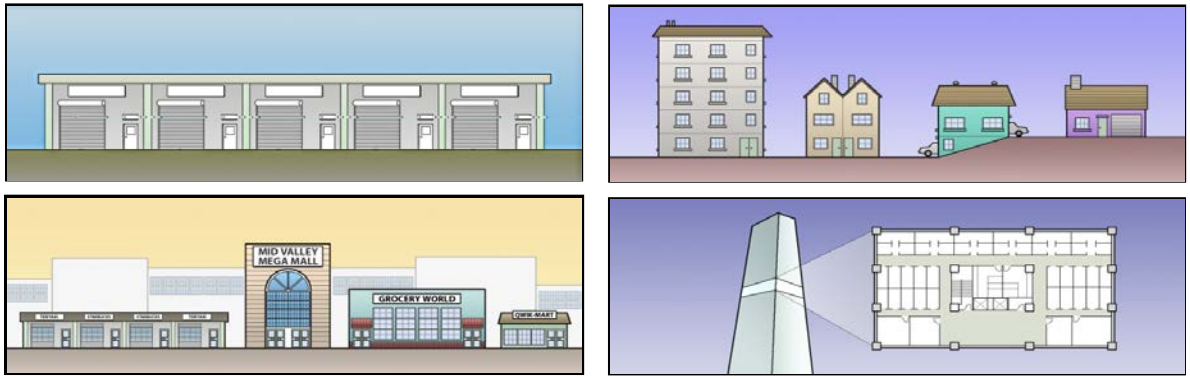
- Public Clouds:
  - Multi-tenancy across companies (typically at the granularity of files and VMs)
    - Some, like Salesforce.com offer finer grain sharing of records in a DB
  - The cloud user *must trust the cloud provider with its confidential data*
  - Precious few laws and conventions around when this is legal and proper
- Private Clouds:
  - On premises at the enterprise; sharing within the enterprise
  - Far fewer legal challenges
  - Harder for the enterprise to gain the value of scale
- Hybrid Enterprise Deployments:
  - Some applications at the IT-Shop's datacenter
  - Some applications in the public cloud
  - Need to work to ensure its OK to trust the public cloud and to make EAI work
- Semi-Private Clouds:
  - Coarse-grained allocation of computing resources (e.g. a rack of servers)
  - Leverage the public cloud's datacenter efficiencies with simple and verifiable allocation of resources to the IT department to run its private cloud

Outline

- Introduction
- Patterns in SaaS Applications: the Front-End
- Patterns in SaaS Applications: the Back-End and Decision Support
- Multi-Tenancy: Making It Work
- Conclusion

Defining Constraints and Empowerment

- Programming platforms define Application Programming Interfaces (APIs)
  - These define their respective models for use and platform support
  - Different platforms may have different models for use and support
- Shared buildings have expectations for usage
  - They are (typically) built without knowing *who* will occupy them
  - They are built with an expectation of *how they will be used*
    - You are not allowed to live and sleep in a retail store or industrial park



- We need to define the expectations for usage for Platform as a Service
  - It is OK for this to address a large class of customers but be useless for others
  - We are designing the platform without knowing who will occupy it
  - It must offer great "concierge services" with acceptable constraints

Concierge Services for the Front-End

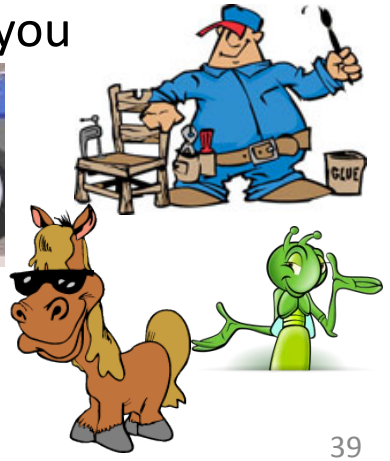
Auto-Scaling	As the workload rises, additional servers are automatically allocated for this service. Resources taken back when load drops.
Auto-Placement	Deployment, migration, fault boundaries, and geographic transparency are all included. Applications are blissfully ignorant.
Capacity Planning	Analysis of traffic patterns of service usage back to incoming user work load. Trends in incoming user workload are tracked.
Resource Marketplace	Plumbing tracks a service's cost as it directly consumes resources and indirectly consumes them (by calling other services).
A/B-Testing and Experimentation	Plumbing makes it easy to deploy a service on a subset of the traffic and compare the results with the previous version.
Auto-Caching / Data Distribution	Data is fed into a datastore and processed there. This processed data is cached for easy access by services.
Session State Management	User session information is captured before a service completes. The next request easily fetches the state to use.

Concierge Services for the Back-End

"Big-Data" Unified Data Access	Unified enterprise-wide (and controlled cross-enterprise) data. Anything may be processed with anything (if authorized).
Relational DB for Silos/Services	Relational DBs supporting enterprise apps which work as silos or services. ETL to stage data into the "Big-Data" store.
Fault Tolerant & Scalable Storage	Cloud managed storage for both "Big-Data" and relational. Automatic intra-datacenter and cross-datacenter replication.
Massively Parallel Batch & Event	High-level set-oriented operations. Incoming events call pub-sub style apps which transactionally update "Big-Data".
Automatic Scalable Ref-Data Caching	The Back-End supplies the Front-End with dynamically updated application specific data. Automated high-performance caching.
Multi-Tenanted Access Control	Intra (and inter) enterprise access control to data contained in the "Big-Data" store and the relational store
Prioritized SLA Driven Resources	Relational DBs, Batch, and Events compete for the same stuff. Work is given its SLAs and priorities. Tradeoffs are automated.

Condos Impose Constraints

- My brother raises pet chickens for eggs… He doesn't live in a condo
  - My condo doesn't allow pet chickens.
- Condo buildings are designed with an expectation of their usage pattern
  - People living with a certain density, certain lifestyle, and certain restrictions
  - Many services are preplanned:
    - We have an exercise room, concierge, wine storage, parking, and a lovely set of rooftop patios for barbequing with nice gas grills and gas fireplaces
- It is the constraints that allow the community to exist
  - Shared services, shared building, shared maintenance, and shared expenses
  - Condos are based on sharing for reduced costs and increased benefits
- If the constraints aren't right for you, condo living isn't right for you
  - If you want to raise chickens or horse, listen to crickets in your backyard, work in a private woodshop, or repair your own automobiles, this isn't for you



- If your lifestyle fits the constraints, condo living can be great!
  - No lawn to mow, gutters to sweep, or trash to take to the street!

What about "The Forces Driving Us to the Cloud"?

Datacenter Economics	**Public Clouds** • Cloud providers: large & efficient datacenters • Fine-grained multi-tenant • *Must trust cloud provider sharing mechanisms*	**Semi-Private Cloud** • Coarse-grained multi-server (rack-level) allocation • VPN between racks • More contained trust • Less fluid resource allocation
Shared Data	**Evolve towards All Data Shared in a "Big Data" Store** • Common, distributed, uniformly addressable data store • Replicated within and across datacenters for high-availability • Existing siloed databases replicated into the common store • Analyze anything-to-anything for "What-If?" • Massively parallel batch and event processing for back-end apps	
Shared Resources	**Coarse-Grained Sharing** • VMs and physical servers offer fungible sharable units • Existing applications can be migrated to coarse-sharing	**Fine-Grained Sharing** • Front-end services share the same platform resources • Back-end batch and event processing shares resources • Prioritized SLAs thru sharing

Takeaways

- Shared buildings became successful by constraining and standardizing usage
  - Apartments, condos, office, retail, & light manufacturing have constraints
  - Not everyone can accept the constraints… if you do, there are efficiencies
  - Standardization allows for outsourcing and sharing many aspects of buildings
  - *Changes in usages patterns, expectations, and the law were required*
- Standardization of usage will empower migration of work to the shared cloud
  - Lower-level standardization (e.g. VMs) supports more apps but with fewer services
  - Higher level "Platform-as-a-Service" is nascent but can offer many advantages
- We must define and constrain usage models for important types of cloud apps
  - This will allow us to offer enhanced sharing with important supporting services
- Enterprises can gain tremendously from public and/or private clouds
  - Sharing across applications (both front-end and back-end) can increase utilization
  - Driving towards fungible resources spurs efficient provisioning and allocation
  - Using the cloud clarifies the relationship between biz-apps and IT control/services
  - "Concierge-services" empower IT and liberate business applications

ACM: The Learning Continues

- Questions about this webinar? learning@acm.org
- ACM Learning Center: http://learning.acm.org
- ACM Tech Pack on Cloud Computing (*open to all, complimentary DL articles for members*): http://techpack.acm.org/cloud/
- ACM SIGMOD: http://www.sigmod.org
- ACM SIGMOD Conference: www.sigmod.org/2013/

Association for Computing Machinery

Advancing Computing as a Science & Profession