

# Netflix in the Cloud

Nov 3, 2010

Adrian Cockcroft

@adriano #netflixcloud

acockcroft@netflix.com

<http://www.linkedin.com/in/adriancockcroft>

The Netflix logo, consisting of the word "NETFLIX" in white, uppercase, sans-serif font, centered within a red rectangular background.

*With more than 16 million subscribers in the United States and Canada, Netflix, Inc. is the world's leading Internet subscription service for enjoying movies and TV shows.*

The Netflix logo, consisting of the word "NETFLIX" in white, uppercase, sans-serif font, centered within a red rectangular background.

# Why Give This Talk?

# Netflix is Path-finding

The Cloud ecosystem is evolving very fast  
Share with and learn from the cloud community



**adriano** adrian cockcroft

This week's choice: buy/sell cloud at [#CloudExpo](#) or build/use cloud at [#QConSF](#), I'm a user, you?

15 hours ago

# We want to use clouds, not build them

Cloud technology should be a commodity

Public cloud and open source for agility and scale

# We are looking for talent

Netflix wants to connect with the very best  
engineers

The Netflix logo, consisting of the word "NETFLIX" in white, uppercase, sans-serif font, centered within a red rectangular background.

# Why Use AWS?



NETFLIX

# We stopped building our own datacenters

Capacity growth rate is accelerating, unpredictable

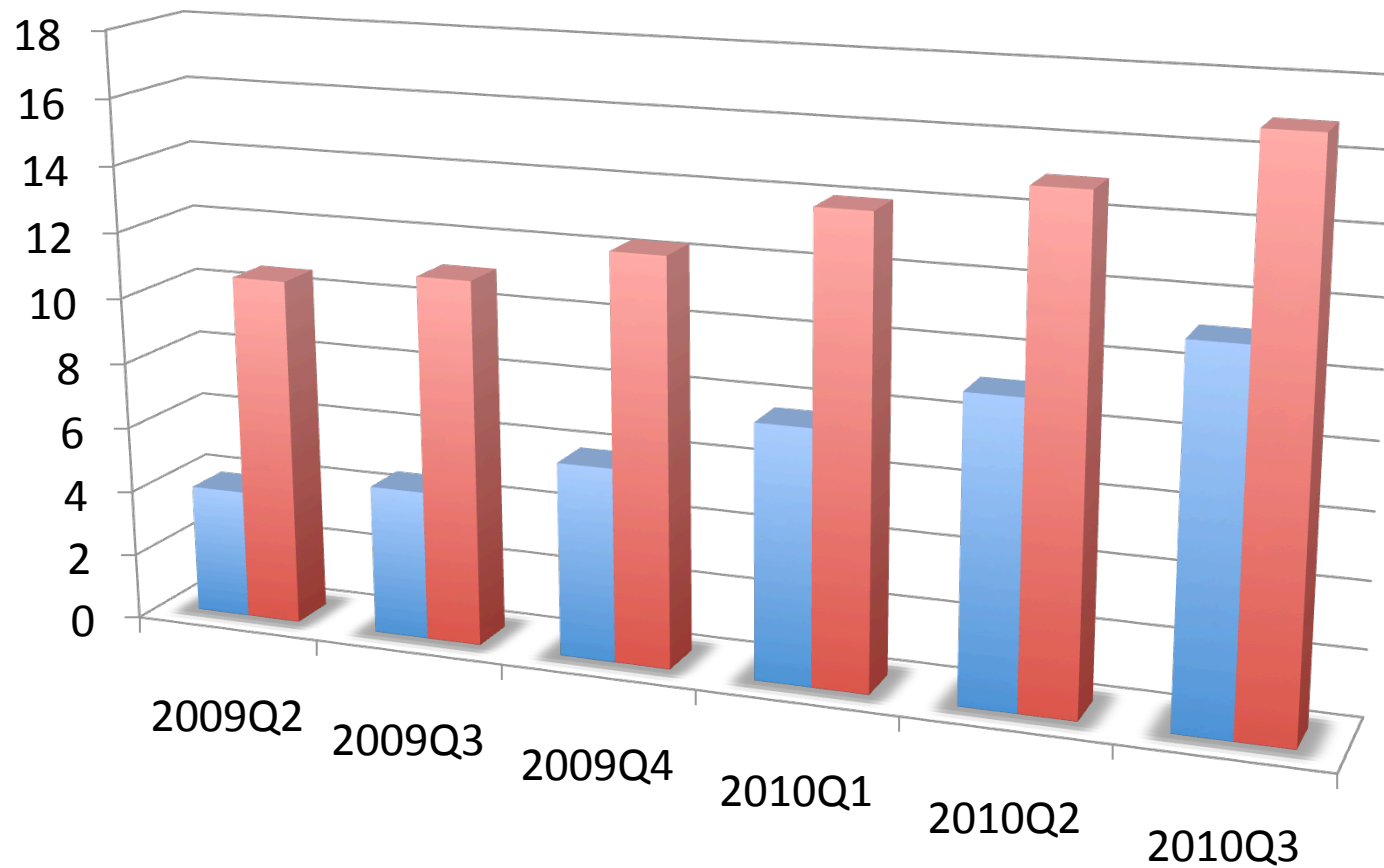
Product launch spikes - iPhone, Wii, PS3, XBox

Datacenter is large inflexible capital commitment



# Customers

Q3 year/year +52% Total and +145% Streaming



Source: <http://ir.netflix.com>



# Leverage AWS Scale

## “the biggest public cloud”

AWS investment in tooling and automation

AWS zones for high availability, scalability

AWS skills are common on resumes...

# Leverage AWS Feature Set “two years ahead of the others”

EC2, S3, SDB, SQS, EBS, EMR, ELB, ASG, IAM, RDB

*“The cloud lets its users focus on delivering differentiating business value instead of wasting valuable resources on the **undifferentiated heavy lifting** that makes up most of IT infrastructure.”*



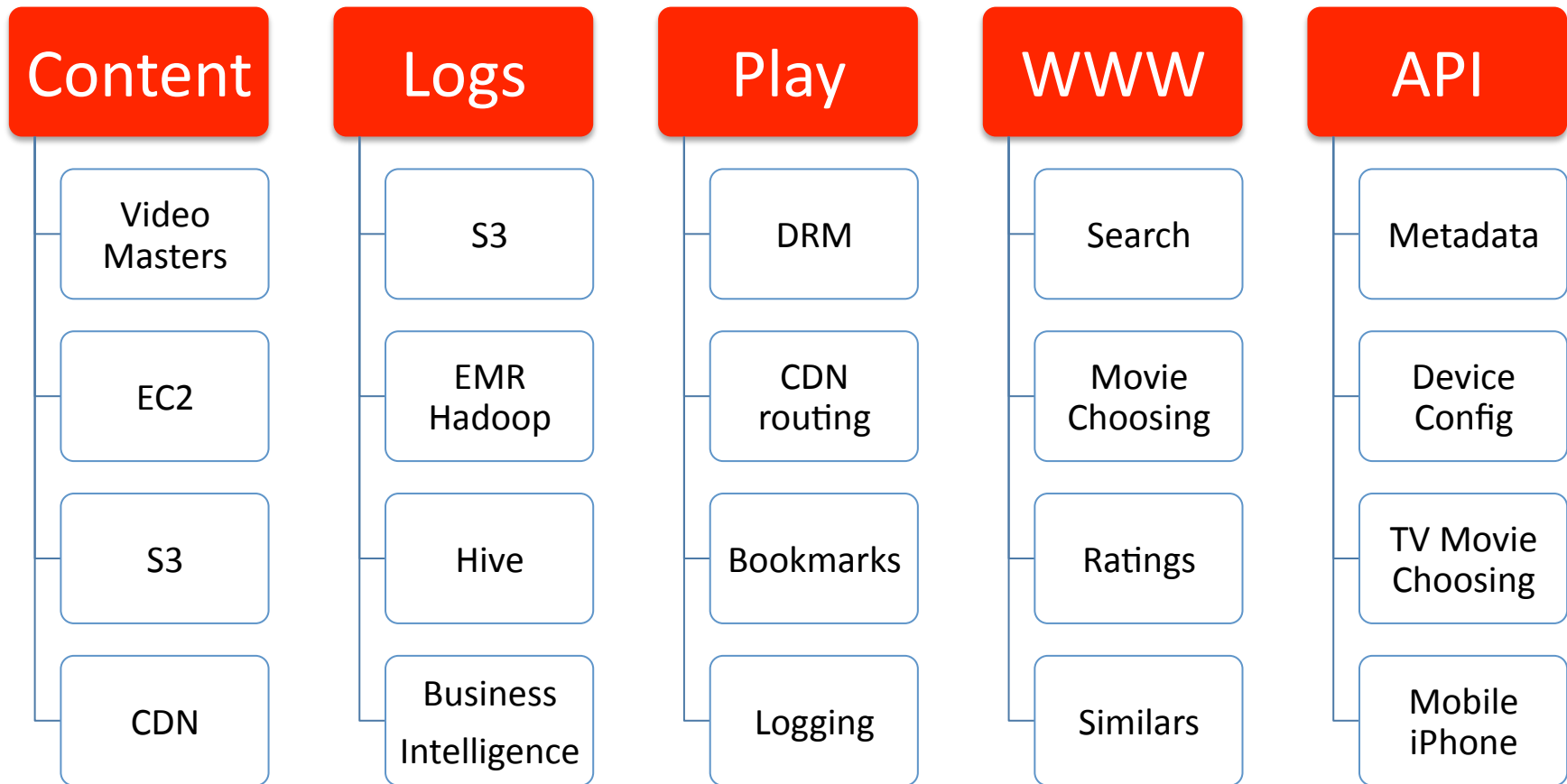
Werner Vogels

Amazon CTO

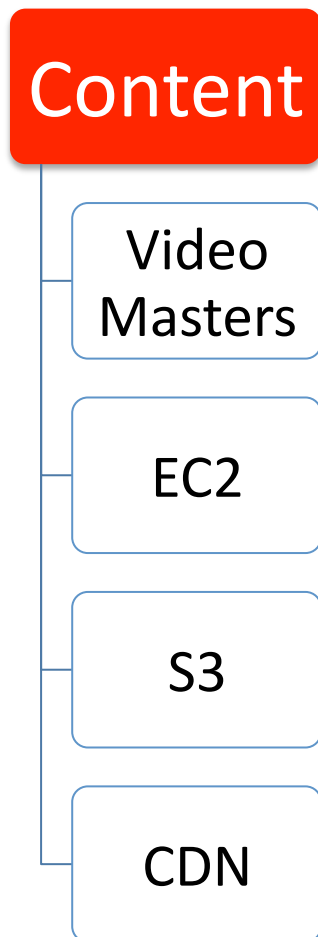


NETFLIX

# Netflix Deployed on AWS



# Movie Encoding farm (2009)

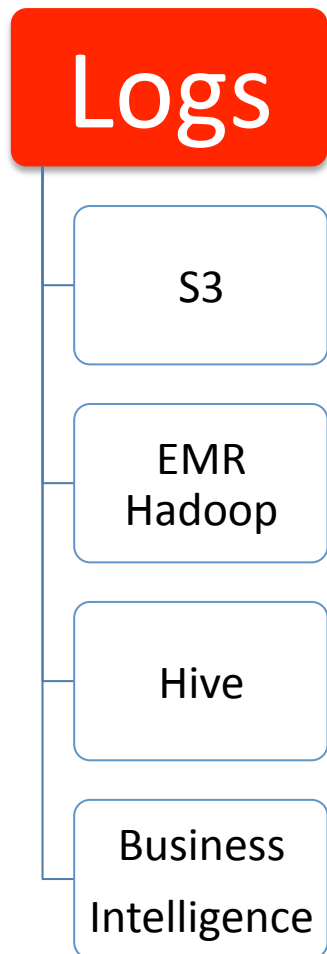


- Tens of thousands of videos
- Thousands of EC2 instances
- Encoding apps on MS Windows
- ~100 speed/format permutations
- Petabytes of S3
- Content Delivery Networks

*“Netflix is one of the largest customers of the biggest CDNs Akamai and Limelight”*



# Hadoop - Elastic Map-Reduce (2009)



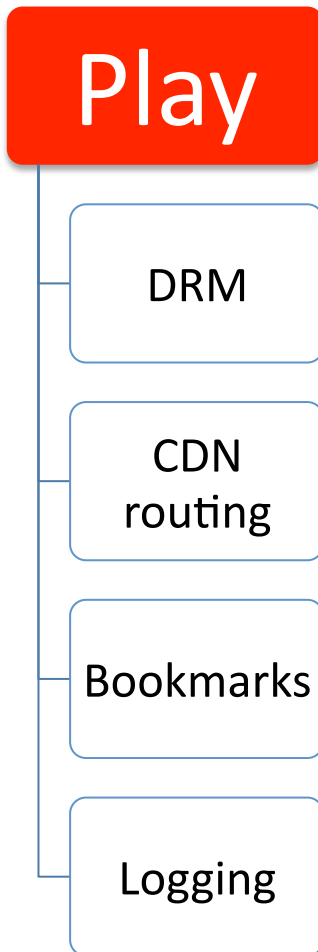
- Web Access Logs
- Streaming Service Logs
- Terabyte per day scale
- Easy Hadoop via Amazon EMR
- Hive SQL “Data Mart”
- Gateway to Datacenter BI

Slideshare.net talks

evamtse “Netflix: Hive User Group” <http://slidesha.re/aqJLAC>

adrianco “Crunch Your Data In The Cloud” <http://slidesha.re/dx4oCK>

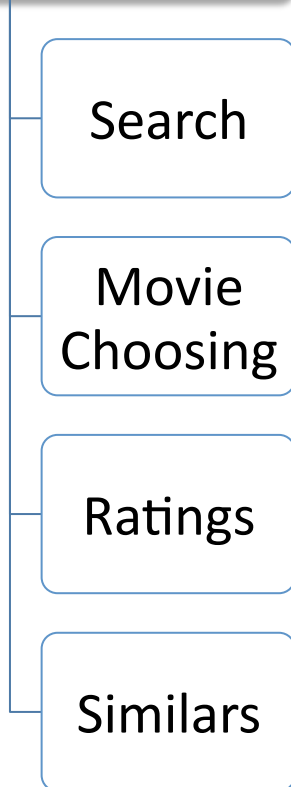
# Streaming Service Back-end (early 2010)



- PC/Mac Silverlight Player Support
- Highly available “play button”
- DRM Key Management
- Generate route to stream on CDN
- Lookup bookmark for user/movie
- Update bookmark for user/movie
- Log quality of service



# Web site, a page at a time (through 2010)



- Clean presentation layer rewrite
- Search auto-complete
- Search backend and landing page
- Movie and genre choosing
- Star ratings and recommendations
- Similar movies
- Page by page to 80% of views

(leave account signup in DC)



# API for TV devices and iPhone etc. (2010)

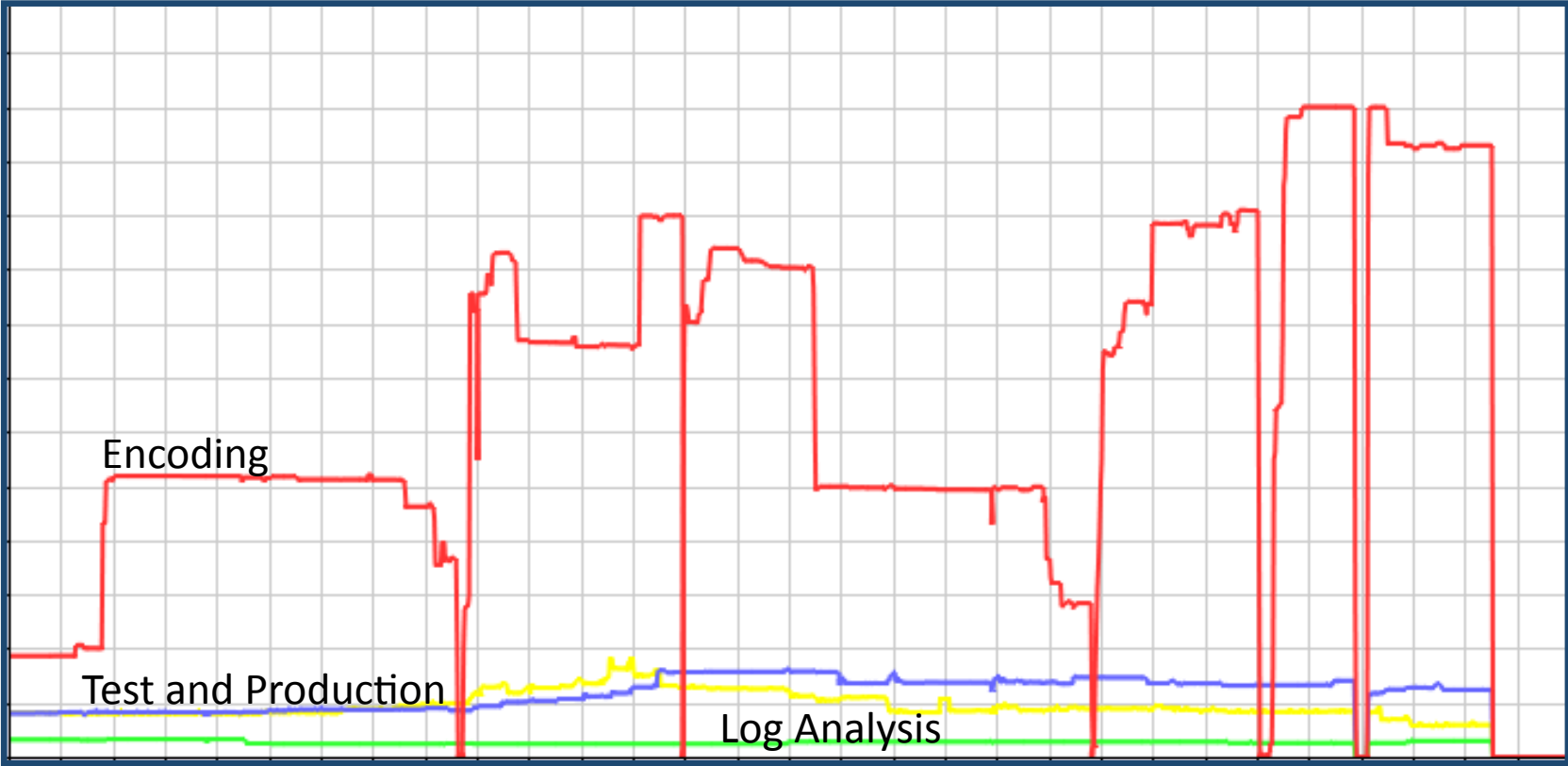


- REST API: [developer.netflix.com](http://developer.netflix.com)
- Interfaces to everything else
- TV Device Configuration
- Personalized movie choosing
- iPhone Launch in the cloud only

*“Netflix is an API for streaming to TVs*

*(we also do DVD’s and a web site)”*

# Netflix EC2 Instances per Account



Learnings...

# Datacenter oriented tools don't work

Ephemeral instances

High rate of change

# Cloud Tools Don't Scale for Enterprise

Too many are “Startup” oriented

Built our own tools

Drove vendors hard

# “fork-lifted” apps don’t work well

Fragile

Too many datacenter oriented  
assumptions

The Netflix logo, consisting of the word "NETFLIX" in white, uppercase, sans-serif font, centered within a red rectangular background.

# Faster to re-code from scratch

- Re-architected and re-wrote most of the code
- Fine grain web services
- Leveraged many open source Java projects
- Systematically instrumented
- “NoSQL” SimpleDB backend



*“In the datacenter, robust code is best practice. In the cloud, it’s essential.”*

# Takeaway

*Netflix is path-finding the use of public AWS cloud to replace in-house IT for non-trivial applications with hundreds of developers and thousands of systems.*

(Pause for questions before we dive into details)



# What, Why and How?

The details...

# Synopsis

- The Goals
  - Faster, Scalable, Available and Productive
- Anti-patterns and Cloud Architecture
  - The things we wanted to change and why
- Cloud Bring-up Strategy
  - Developer Transitions and Tools
- Roadmap and Next Steps

# Goals

- **Faster**
  - **Lower latency** than the equivalent datacenter web pages and API calls
  - Measured as mean and 99<sup>th</sup> percentile
  - For both first hit (e.g. home page) and in-session hits for the same user
- **Scalable**
  - **Avoid needing any more datacenter capacity** as subscriber count increases
  - No central vertically scaled databases
  - Leverage AWS elastic capacity effectively
- **Available**
  - Substantially **higher robustness and availability** than datacenter services
  - Leverage multiple AWS availability zones
  - No scheduled down time, no central database schema to change
- **Productive**
  - Optimize **agility** of a large development team with automation and tools
  - Leave behind complex tangled datacenter code base (~8 year old architecture)
  - Enforce clean layered interfaces and re-usable components

# Cloud Architecture Patterns

Where do we start?

# Datacenter Anti-Patterns

What do we currently do in the datacenter that prevents us from meeting our goals?

# Rewrite from Scratch

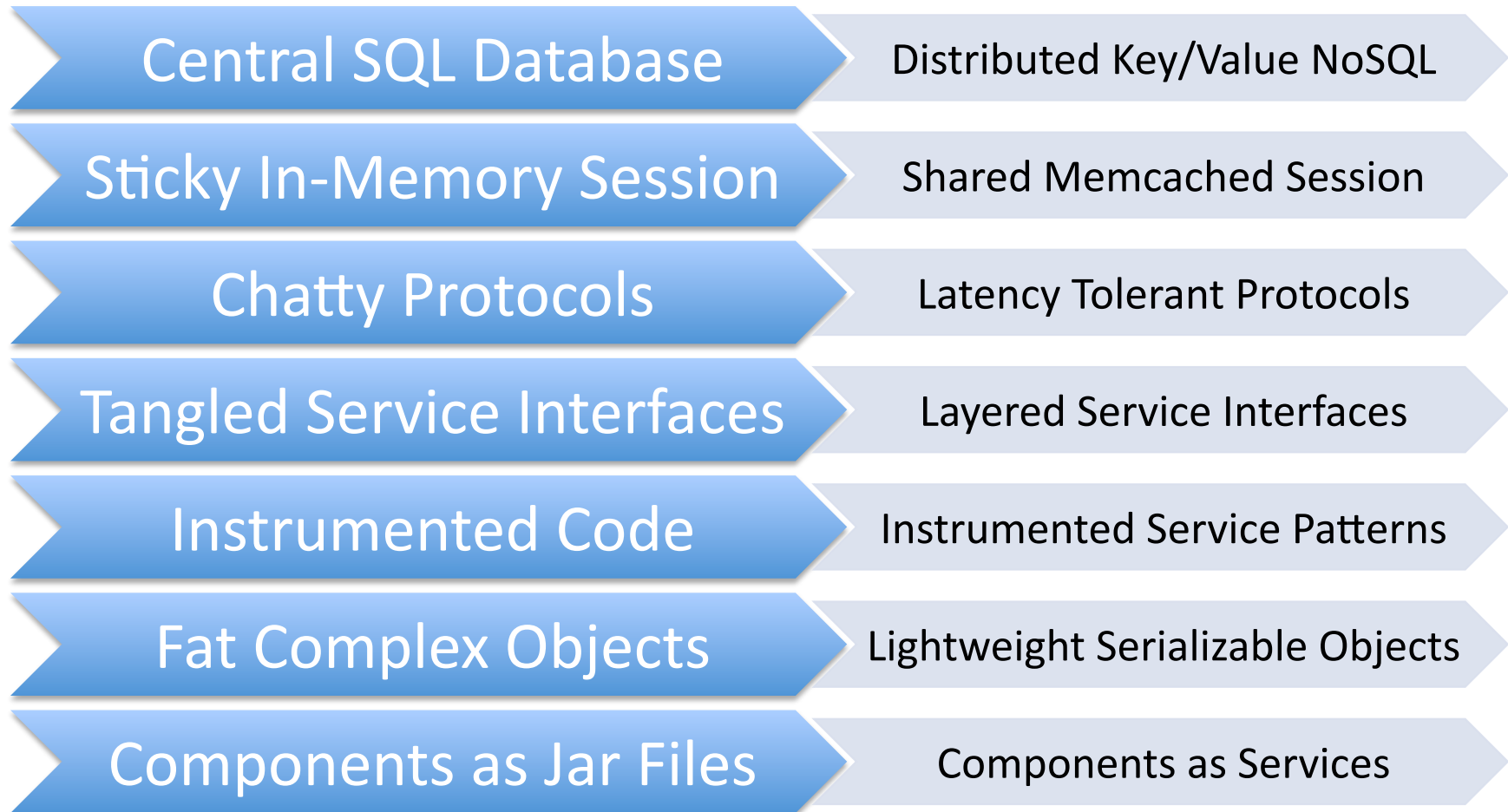
Not everything is cloud specific

Pay down technical debt

Robust patterns



# Old Datacenter vs. New Cloud Arch




# The Central SQL Database

- Datacenter has a central database
  - Everything in one place is convenient until it fails
  - Customers, movies, history, configuration
- Schema changes require downtime

*Anti-pattern impacts scalability, availability*

# The Distributed Key-Value Store

- Cloud has many key-value data stores
  - More complex to keep track of, do backups etc.
  - Each store is much simpler to administer 
  - Joins take place in java code
- No schema to change, no scheduled downtime
- Latency for Memcached vs. Oracle vs. SimpleDB
  - Memcached is dominated by network latency <1ms
  - Oracle for simple queries is a few milliseconds
  - SimpleDB has replication and REST overheads >10ms

# The Sticky Session

- Datacenter Sticky Load Balancing
  - Efficient caching for low latency
  - Tricky session handling code
  - Middle tier load balancer has issues in practice
- Encourages concentrated functionality
  - one service that does everything

*Anti-pattern impacts productivity, availability*

# The Shared Session

- Cloud Uses Round-Robin Load Balancing
  - Simple request-based code
  - External shared caching with memcached
- More flexible fine grain services
  - Works better with auto-scaled instance counts

# Chatty Opaque and Brittle Protocols

- Datacenter service protocols
  - Assumed low latency for many simple requests
- Based on serializing existing java objects
  - Inefficient formats
  - Incompatible when definitions change

*Anti-pattern causes productivity, latency and availability issues*

# Robust and Flexible Protocols

- Cloud service protocols
  - JSR311/Jersey is used for REST/HTTP service calls
  - Custom client code includes service discovery
  - Support complex data types in a single request
- Apache Avro
  - Evolved from Protocol Buffers and Thrift
  - Includes JSON header defining key/value protocol
  - Avro serialization is **half the size** and several times faster than Java serialization, more work to code

# Persisted Protocols

- Persist Avro in Memcached
  - Save space/latency (zigzag encoding, half the size)
  - Less brittle across versions
  - New keys are ignored
  - Missing keys are handled cleanly
- Avro protocol definitions
  - Can be written in JSON or generated from POJOs
  - It's hard, needs better tooling



# Tangled Service Interfaces

- Datacenter implementation is exposed
  - Oracle SQL queries mixed into business logic
- Tangled code
  - Deep dependencies, false sharing
- Data providers with sideways dependencies
  - Everything depends on everything else

*Anti-pattern affects productivity, availability*

# Untangled Service Interfaces

- New Cloud Code With Strict Layering
  - Compile against interface jar
  - Can use spring runtime binding to enforce
- Service interface **is** the service
  - Implementation is completely hidden
  - Can be implemented locally or remotely
  - Implementation can evolve independently

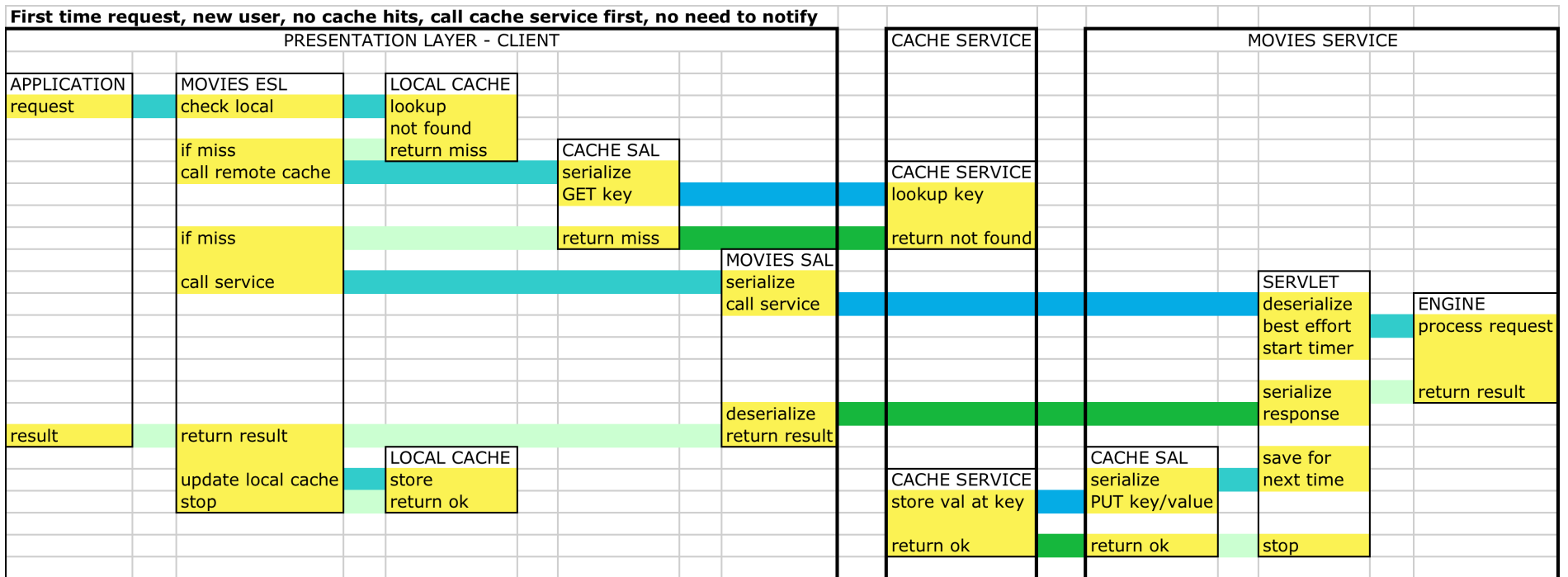
# Untangled Service Interfaces

Two layers:

- SAL - Service Access Library
  - Basic serialization and error handling
  - REST or POJO's defined by data provider
- ESL - Extended Service Library
  - Caching, conveniences
  - Can combine several SALs
  - Exposes faceted type system (described later)
  - Interface defined by data consumer in many cases

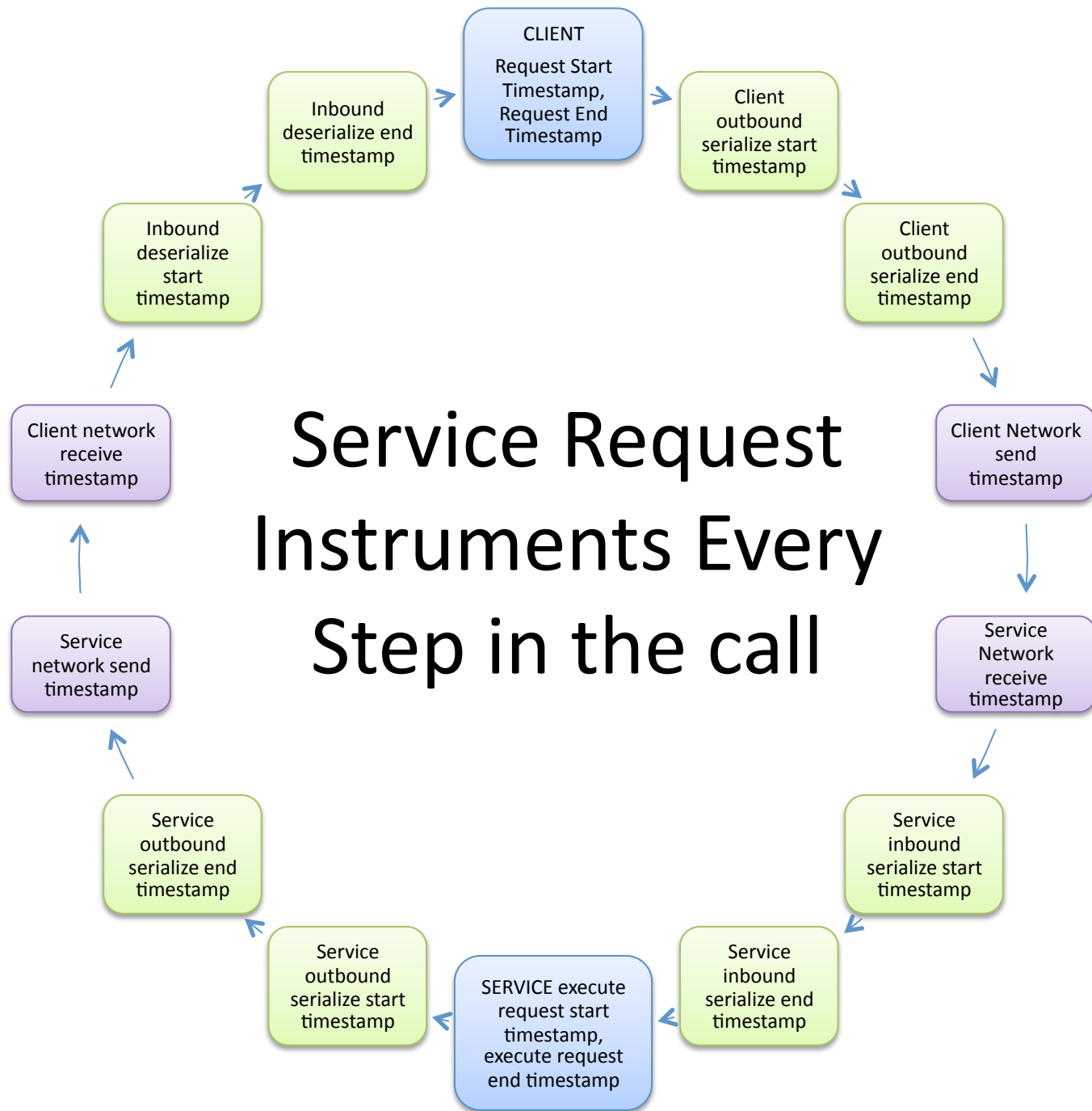
# Service Interaction Pattern

## Swimlane Diagram



# Service Architecture Patterns

- Internal Interfaces Between Services
  - Common patterns as templates
  - Highly instrumented, observable, analytics
  - Service Level Agreements – SLAs
- Library templates for generic features
  - Instrumented Netflix Base Servlet template
  - Instrumented generic client interface template
  - Instrumented S3, SimpleDB, Memcached clients



# Boundary Interfaces

- Isolate teams from external dependencies
  - Fake SAL built by cloud team
  - Real SAL provided by data provider team later
  - ESL built by cloud team using faceted objects
- Fake data sources allow development to start
  - e.g. Fake Identity SAL for a test set of customers
  - Development solidifies dependencies early
  - Helps external team provide the right interface

# One Object That Does Everything

- Datacenter uses a few big complex objects
  - Movie and Customer objects are the foundation
  - Good choice for a small team and one instance
  - Problematic for large teams and many instances
- False sharing causes tangled dependencies
  - Unproductive re-integration work

*Anti-pattern impacting productivity and availability*



# An Interface For Each Component

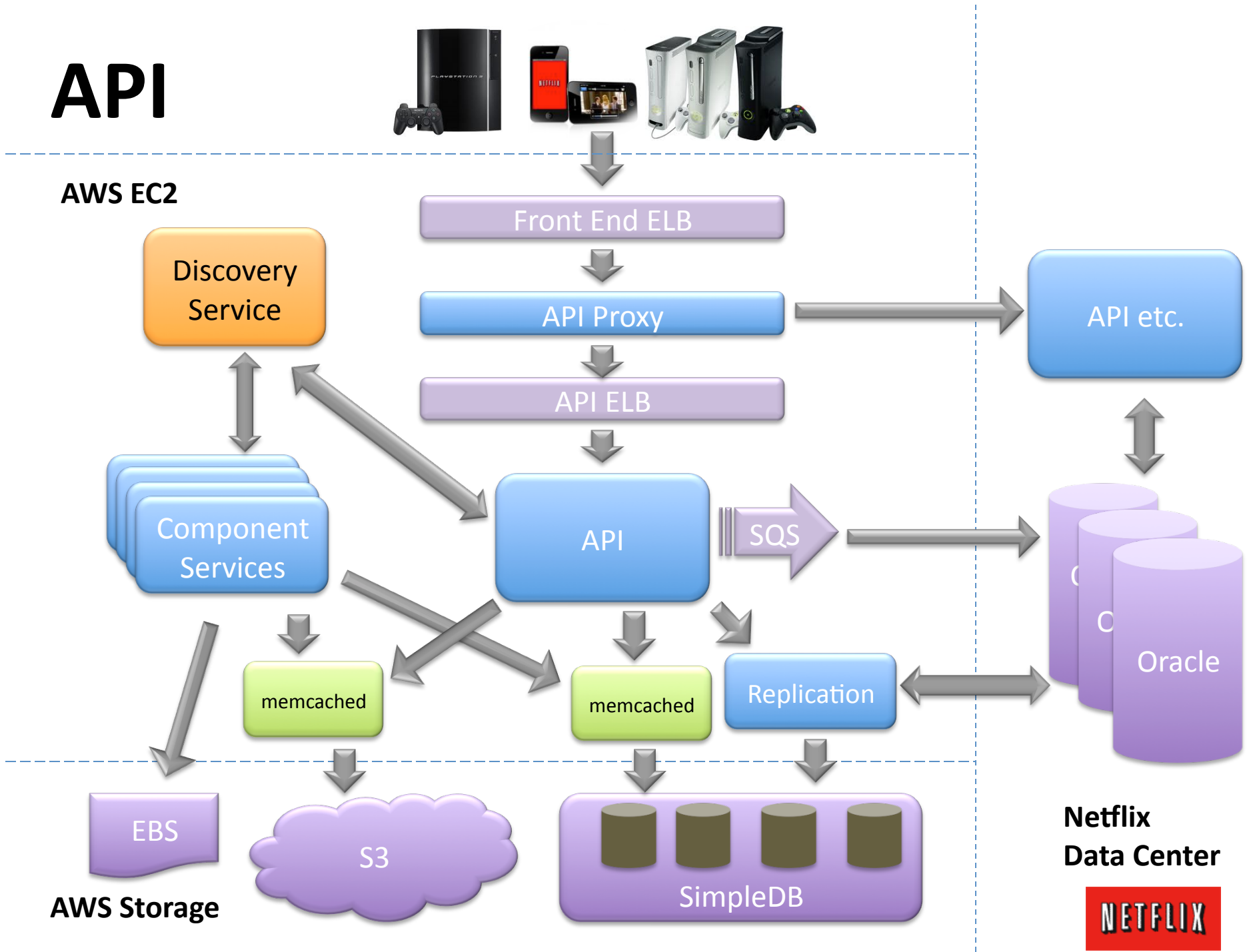
- Cloud uses faceted Video and Visitor
  - Basic types hold only the identifier
  - Facets scope the interface you actually need
  - Each component can define its own facets
- No false-sharing and dependency chains
  - Type manager converts between facets as needed
  - `video.asA(PresentationVideo)` for www
  - `video.asA(MerchableVideo)` for middle tier

# Software Architecture Patterns

- Object Models
  - Basic and derived types, facets, serializable
  - Pass by reference within a service
  - Pass by value between services
- Computation and I/O Models
  - Service Execution using Best Effort
  - Common thread pool management

# Netflix Systems Architecture

# API



# Netflix Undifferentiated Lifting

- Middle Tier Load Balancing
- Discovery (local DNS)
- Encryption Services
- Caching
- Distributed App Management



*We want cloud vendors to do all this for us as well!*

NETFLIX

# Load Balancing in AWS

- Middle tier currently not supported in AWS
  - ELB are public-facing only
  - Cannot apply security group settings
- ELB vertical scalability for concentrated clients
  - Too few proxy IP addresses leads to hot spots
- ELB needs support for balancing heuristics
  - Proportional balance across Availability Zones
  - Weighted Least connections, Weighted Round Robin
- Zone aware routing
  - Default to instances in the same Availability Zone
  - Falls back to cross-zone on failure

# Discovery

- Discovery Service (Redundant instances per zone)
  - Simple REST interface
  - Cloud apps register with Discovery
- Apps send heartbeats every 30 sec to renew lease
  - App evicted after 3 missed heartbeats
  - Can re-register if the problem was transient
- Apps can store custom metadata
  - Version number, AMI id, Availability Zone, etc.
- Software Round-robin Load Balancer
  - Query Discovery for instances of specific application
  - Baked into Netflix REST client (JSR311/Jersey based)

***AWS Middle-tier ELB would eliminate most use cases***

# Database Migration

- Why SimpleDB?
  - No DBA's in the cloud, Amazon hosted service
  - Work started two years ago, fewer viable options
  - Worked with Amazon to speed up and scale SimpleDB
- Alternatives?
  - Investigating adding Cassandra and Membase to the mix
  - Need several options to match use cases well
- Detailed SimpleDB Advice
  - Sid Anand - QConSF Nov 5<sup>th</sup> – Netflix' Transition to High Availability Storage Systems
  - Blog - <http://practicalcloudcomputing.com/>
  - Download Paper PDF - <http://bit.ly/bhOTLu>



# Tools and Automation

- Developer and Build Tools
  - Jira, Eclipse, Hudson, Ivy, Artifactory
  - Builds, creates .war file, .rpm, bakes AMI and launches
- Custom Netflix Application Console
  - AWS Features at Enterprise Scale (hide the keys!)
  - Auto Scaler Group is unit of deployment to production
- Open Source + Support
  - Apache, Tomcat, OpenJDK, CentOS
- Monitoring Tools
  - Keynote – service monitoring and alerting
  - AppDynamics – Developer focus for cloud
  - EpicNMS – flexible data collection and plots <http://epicnms.com>
  - Nimsoft NMS – ITOps focus for Datacenter + Cloud alerting

# Current Status

# WWW Page by Page during Q2/Q3/Q4

- Simplest possible page first
  - Minimal dependencies
- Add pages as dependent services come online
- Home page – most complex and highest traffic
- Leave low traffic pages for later cleanup

*gradual migration from Datacenter pages*

# Big-Bang Transition

- iPhone Launch (August/Sept)
  - No capacity in the datacenter, cloud only
  - App Store gates release, not gradual, can't back out
  - Market is huge (existing and new customers)
  - Has to work at large scale on day one
- Datacenter Shadow Redirect Technique
  - Used to stress back-end and data sources
- SOASTA Cloud Based Load Generation
  - Used to stress test API and end-to-end functionality

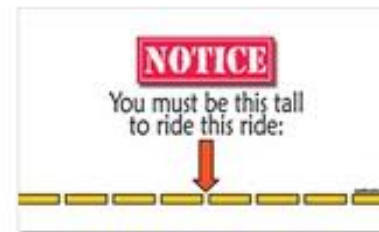
# Current Work for Cloud Platform

- Drive latency and availability goals
  - More Aggressive caching
  - Fault and latency robustness
- Logging and monitoring portal/dashboards
  - Working to integrate tools and data sources
  - Need better observability and automation
- Evaluating a range of NoSQL choices
  - Broad set of use cases, no single winner
  - Good topic for another talk...

# Wrap Up

# Next Few Years...

- “System of Record” moves to Cloud
  - Master copies of data live only in the cloud, with backups etc.
  - Cut the datacenter to cloud replication link
- International Expansion – Global Clouds
  - Rapid deployments to new markets
- GPU Clouds optimized for video encoding
- Cloud Standardization
  - Cloud features and APIs should be a commodity not a differentiator
  - Differentiate on scale and quality of service
  - Competition also drives cost down
  - Higher resilience
  - Higher scalability



*We would prefer to be an insignificant customer in a giant cloud*

# Remember the Goals

Faster

Scalable

Available

Productive

*Track progress against these goals*



# Takeaway

*Netflix is path-finding the use of public AWS cloud to replace in-house IT for non-trivial applications with hundreds of developers and thousands of systems.*

<http://www.linkedin.com/in/adriancockcroft>

@adrianco #netflixcloud

acockcroft@netflix.com



NETFLIX