Globally Distributed Cloud Applications at Netflix

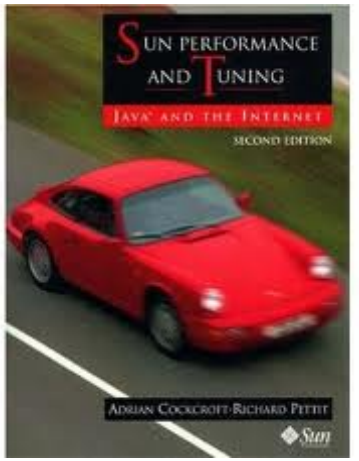
October 2012

Adrian Cockcroft

@adrianco #netflixcloud

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

Adrian Cockcroft



- Director, Architecture for Cloud Systems, Netflix Inc.
  - Previously Director for Personalization Platform
- Distinguished Availability Engineer, eBay Inc. 2004-7
  - Founding member of eBay Research Labs
- Distinguished Engineer, Sun Microsystems Inc. 1988-2004
  - 2003-4 Chief Architect High Performance Technical Computing
  - 2001 Author: *Capacity Planning for Web Services*
  - 1999 Author: *Resource Management*
  - 1995 & 1998 Author: *Sun Performance and Tuning*
  - 1996 Japanese Edition of *Sun Performance and Tuning*
    - SPARC & Solarisパフォーマンスチューニング (サンソフトプレスシリーズ)
- More
  - Twitter @adrianco – Blog http://perfcap.blogspot.com
  - Presentations at http://www.slideshare.net/adrianco

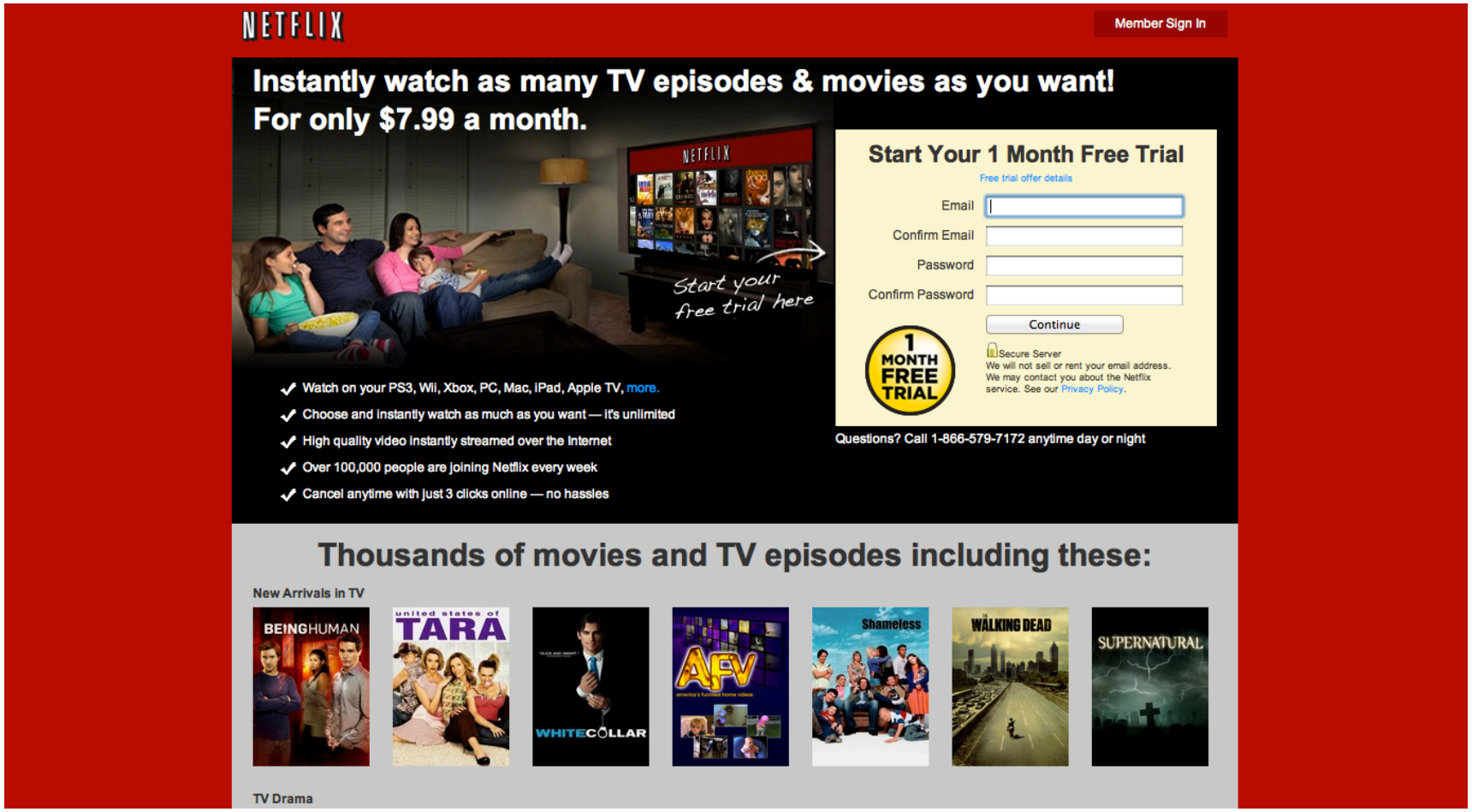
NETFLIX

The Netflix Streaming Service

Now in USA, Canada, Latin America,
UK, Ireland, Sweden, Denmark,
Norway and Finland

US Non-Member Web Site

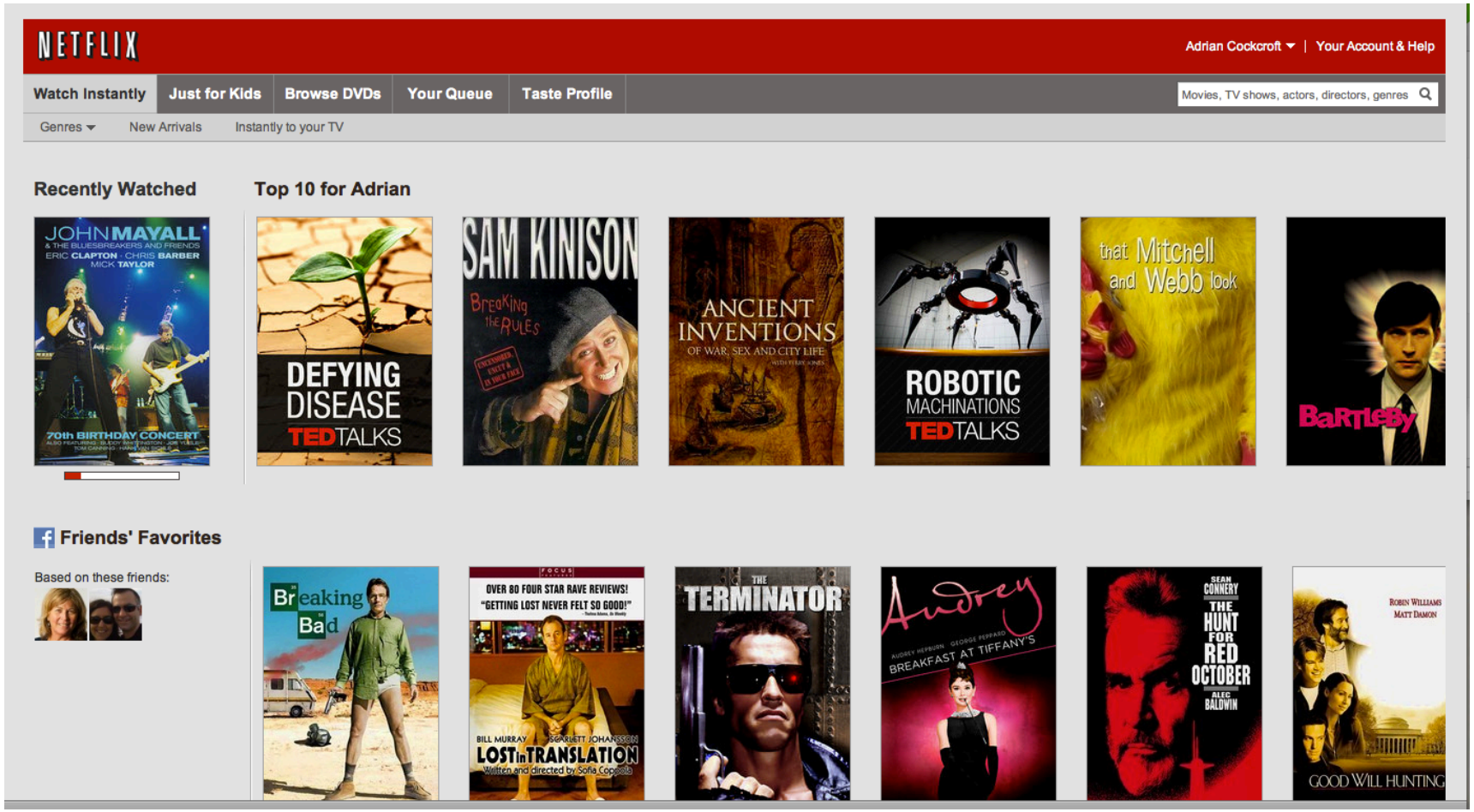
Advertising and Marketing Driven



NETFLIX

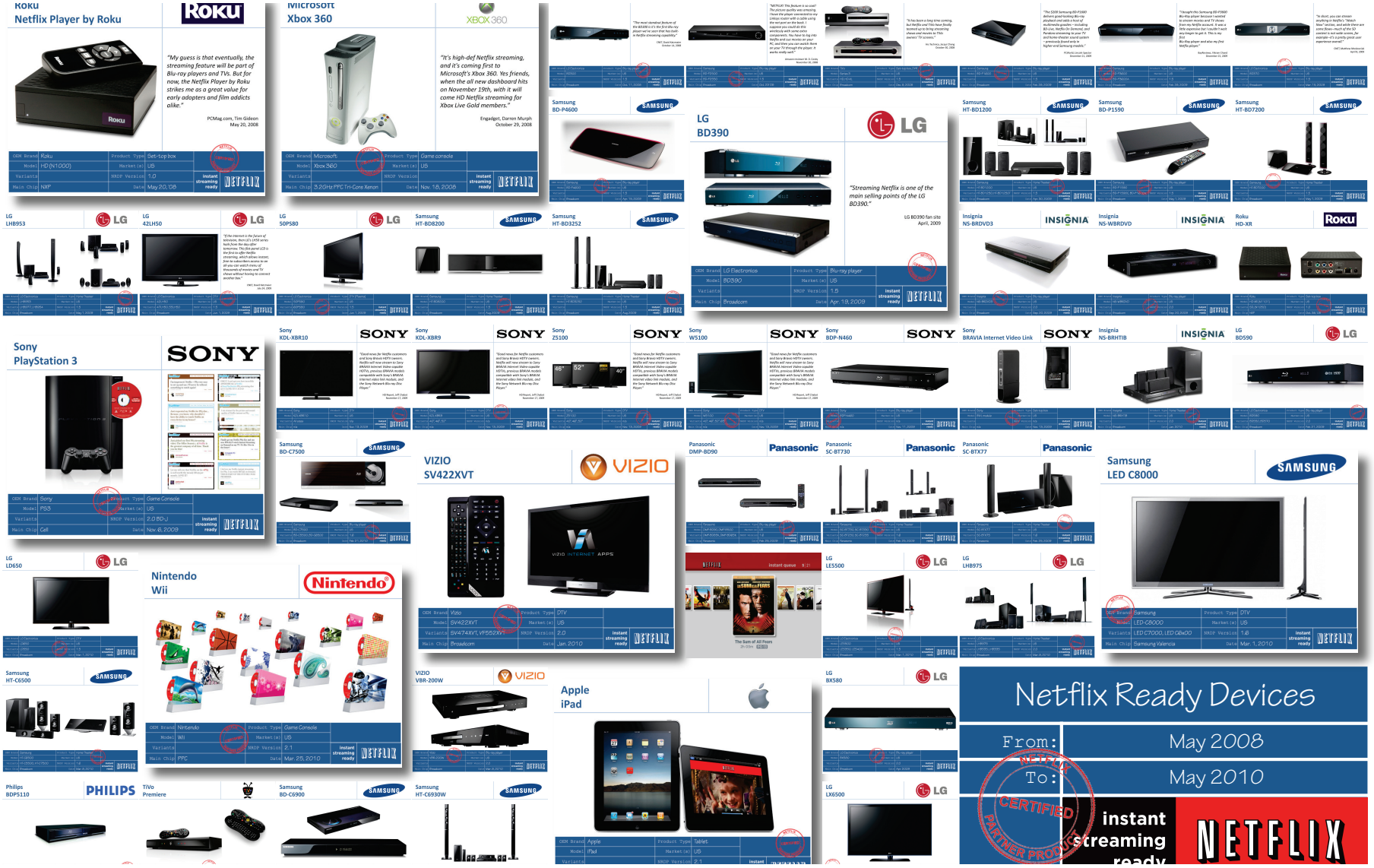
Member Web Site

Personalization Driven



NETFLIX

Streaming Device API



Content Delivery Service

Distributed storage nodes controlled by Netflix cloud services

NETFLIX

Open Connect

- Overview
- FAQ
- Peering Information
- > Hardware Design
- Software Design
- Deployment Guide
- ISP Inquiry

Open Connect Appliance Hardware

Objectives

When designing the Open Connect Appliance Hardware, we focused on these fundamental design goals:

- Very high storage density without sacrificing space and power efficiency. Our target was fitting 100 terabytes into a 4u chassis that is less than 2' deep.
- High throughput: 10 Gbps throughput via an optical network connection.
- Very low field maintenance: the appliance must tolerate a variety of hardware failures including hard drives, network optics, and power supply units.
- Simple racking and installation. Front mounted power and network ports are the only things to connect at install time.

Open Connect Appliances are servers based on commodity PC components (similar to the model used by all large scale content delivery networks). We were influenced by the excellent write-ups from the Backblaze team, and use a custom chassis due to a lack of ready made options for a compact unit.

To achieve over 100 TB of storage, spinning hard drives provide the highest affordable density, in particular 36 3TB SATA units. The hard drives are not hot swappable, as we wish to avoid the operational burden of field service. For lower power utilization and simpler sourcing we select commodity units from two vendors and use software to manage failure modes and avoid field replacement. Dead drives reduce the total storage available for the system, but don't take it offline. We also add 1 TB of flash storage (2 solid state drives) for system files, logs and popular content. To augment the motherboard attached controller, we use two 16 port LSI SAS controller cards that connect directly to the SATA drives. This avoids I/O bottlenecks of SATA multipliers or SAS expanders, and also reduces system complexity.

From a compute point of view, the system has modest requirements moving bits from the storage to network packets on the interface. To reduce the power usage and hence also cooling requirement (which in turn reduces vibration from case fans) we use a single low power 4 core Intel Sandy Bridge CPU on a small form factor Supermicro mATX board with the full 32 GB of RAM installed.

We use redundant, hot swappable power supply units that have interchangeable AC and DC options for maximum installation flexibility. Zippy reversed the fan rotation of the units to allow mounting at the front of the case, and thus allow network and power connects to be positioned here.

The network card has two 10 Gbps modules, which can power a variety of SR and LR optic modules, for installation flexibility and scalable interconnection.

NETFLIX

Abstract

- Netflix on Cloud – What, Why and When
- Globally Distributed Architecture
- Open Source Components

Why Use Cloud?

Get stuck with wrong config

Wait

Wait

File tickets

Ask permission

Wait

Wait

Wait

Things we don't do

Wait

Run out of space/power

Plan capacity in advance

Have meetings with IT

Wait

NETFLIX

What Netflix Did

- Moved to SaaS
  - Corporate IT – OneLogin, Workday, Box, Evernote…
  - Tools – Pagerduty, AppDynamics, EMR (Hadoop)
- Built our own PaaS
  - Customized to make our developers productive
  - Large scale, global, highly available, leveraging AWS
- Moved incremental capacity to IaaS
  - No new datacenter space since 2008 as we grew
  - Moved our streaming apps to the cloud

Keeping up with Developer Trends

	In production at Netflix
• Big Data/Hadoop	2009
• AWS Cloud	2009
• Application Performance Management	2010
• Integrated DevOps Practices	2010
• Continuous Integration/Delivery	2010
• NoSQL	2010
• Platform as a Service; Fine grain SOA	2010
• Social coding, open development/github	2011

AWS specific feature dependence….

Portability vs. Functionality

- Portability – the Operations focus
  - Avoid vendor lock-in
  - Support datacenter based use cases
  - Possible operations cost savings

- Functionality – the Developer focus
  - Less complex test and debug, one mature supplier
  - Faster time to market for your products
  - Possible developer time/cost savings

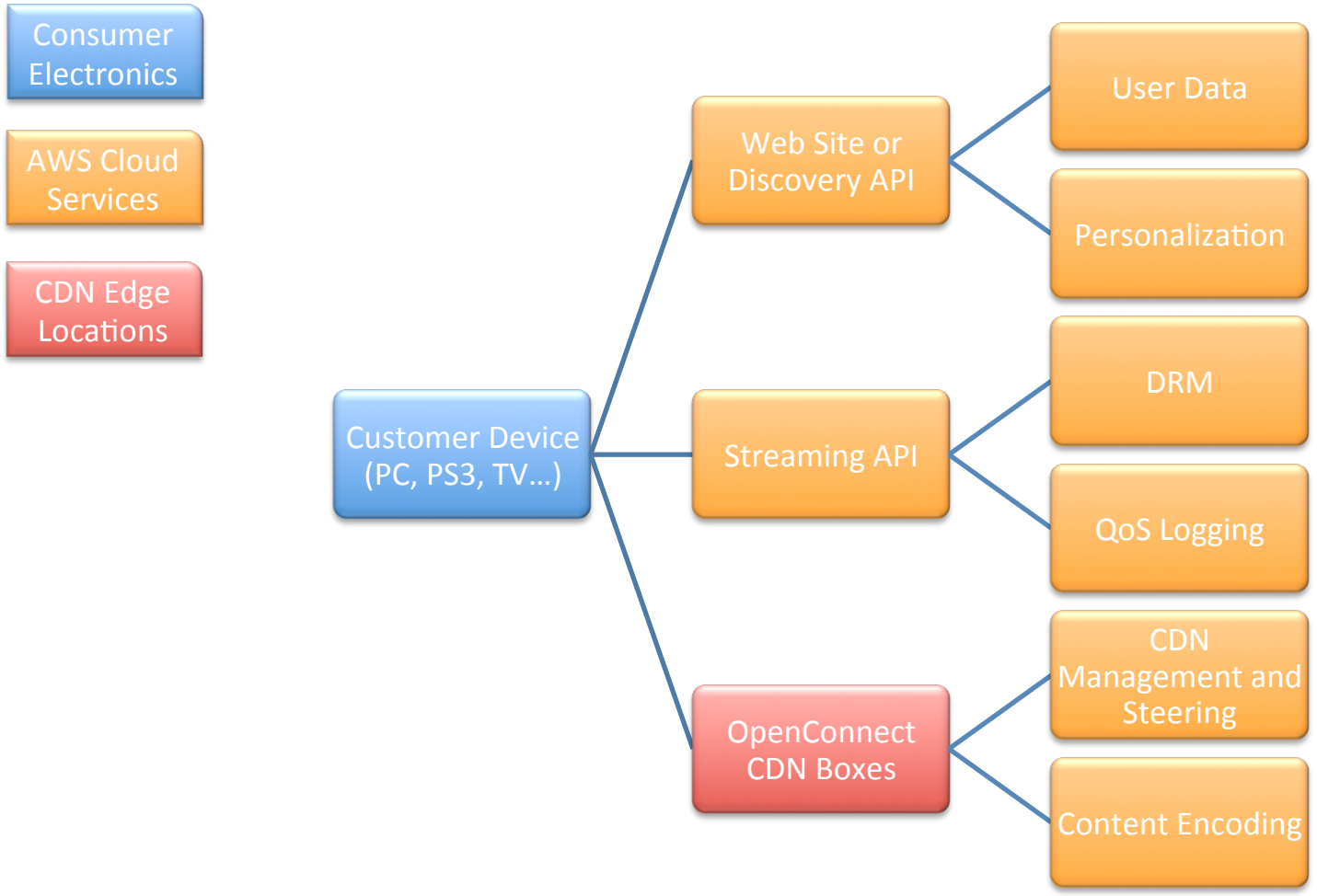
Functional PaaS

- IaaS base - all the features of AWS
  - Very large scale, mature, global, evolving rapidly
  - ELB, Autoscale, VPC, SQS, EIP, EMR, etc, etc.
  - E.g. Large files (TB) and multipart writes in S3

- Functional PaaS – Netflix added features
  - Continuous build/deploy, SOA, HA patterns
  - Asgard console, Monkeys, Big data tools
  - Cassandra/Zookeeper data store automation

NETFLIX

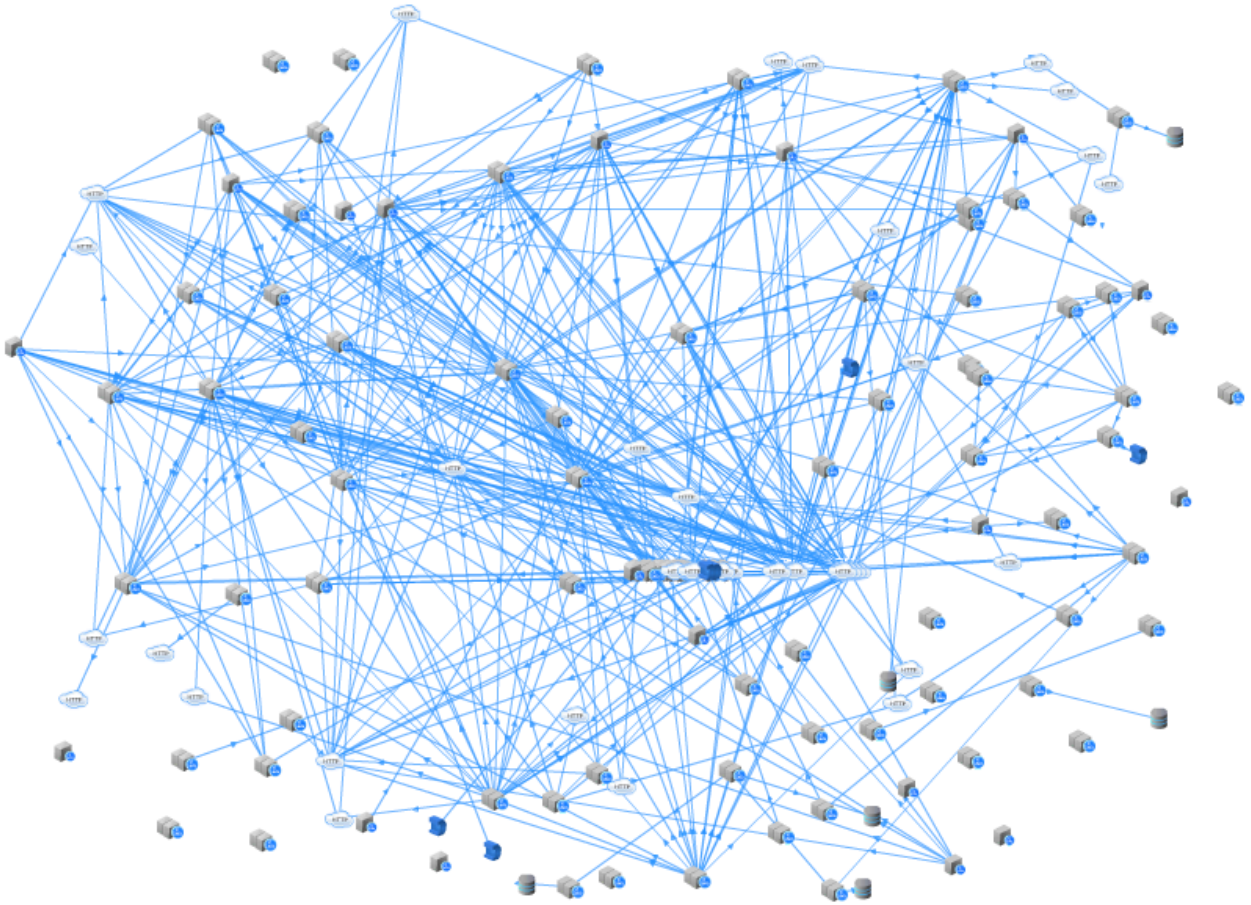
How Netflix Works



NETFLIX

Component Services

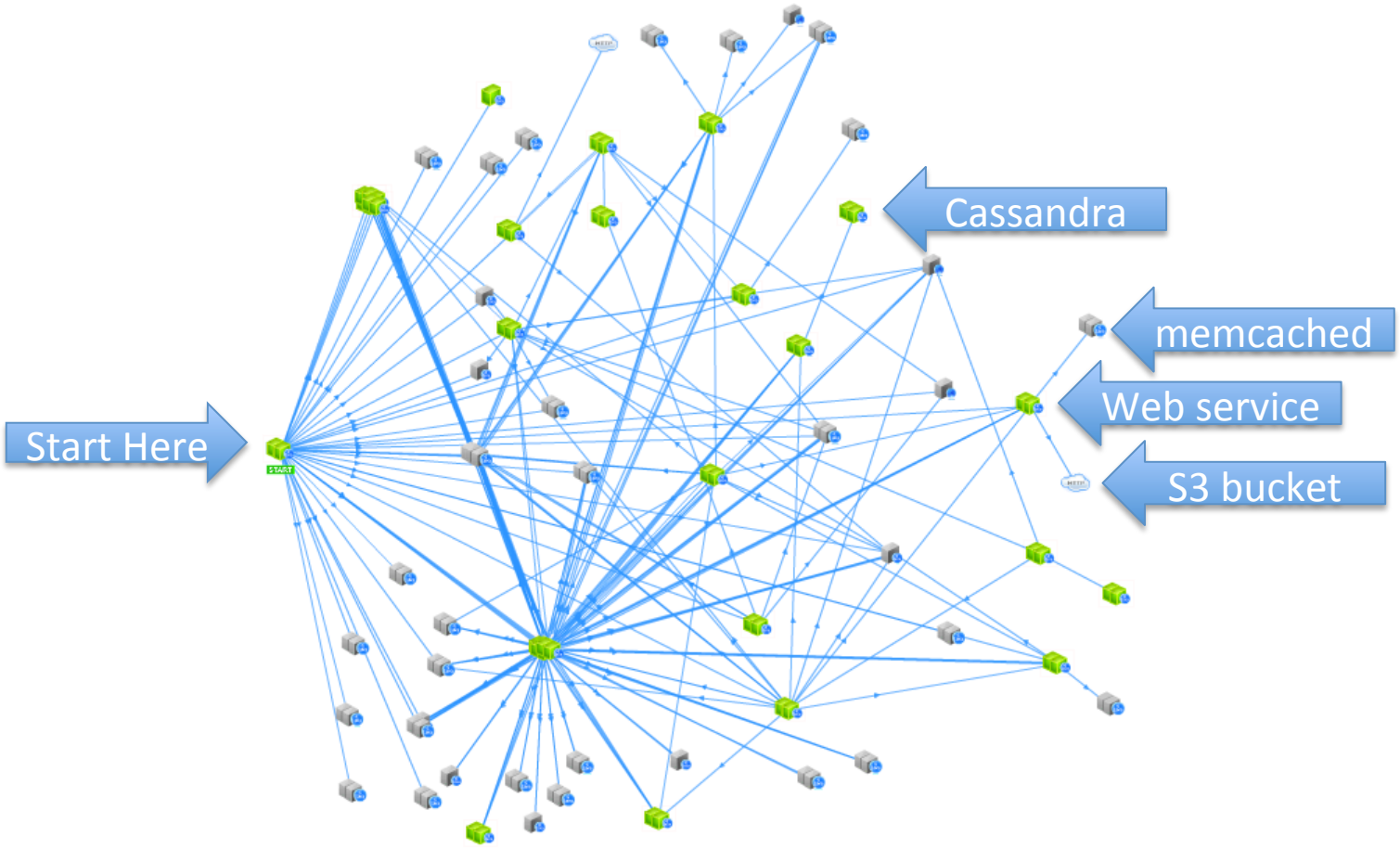
(Simplified view using AppDynamics)



NETFLIX

Web Server Dependencies Flow

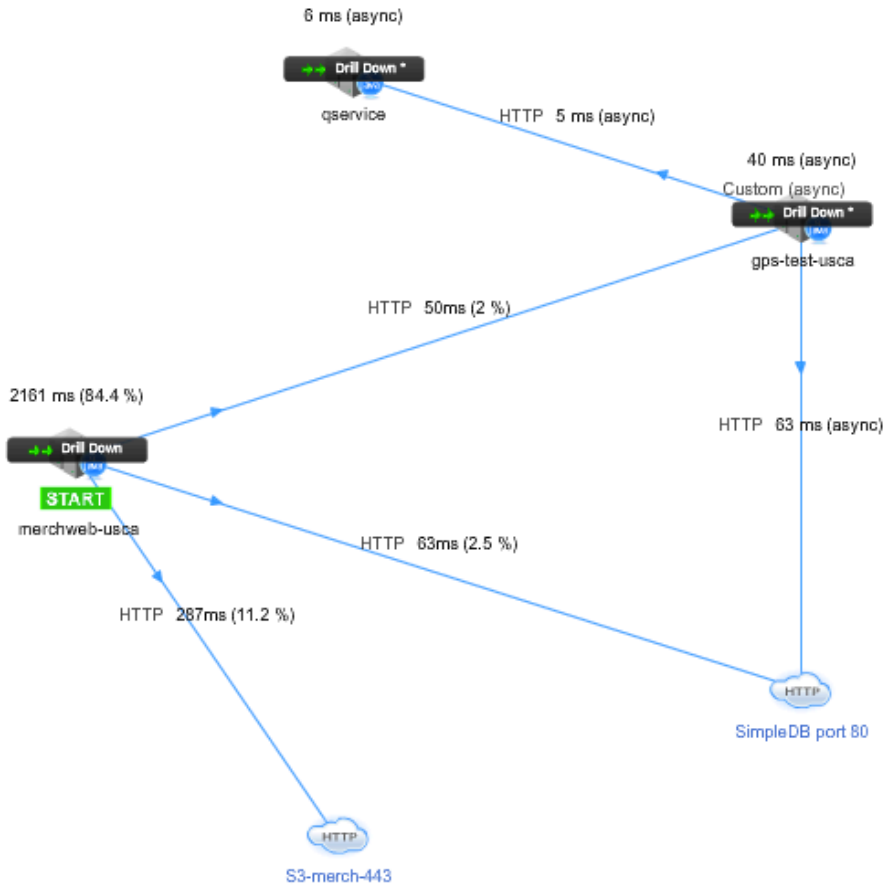
(Home page business transaction as seen by AppDynamics)



NETFLIX

One Request Snapshot

(captured because it was unusually slow)



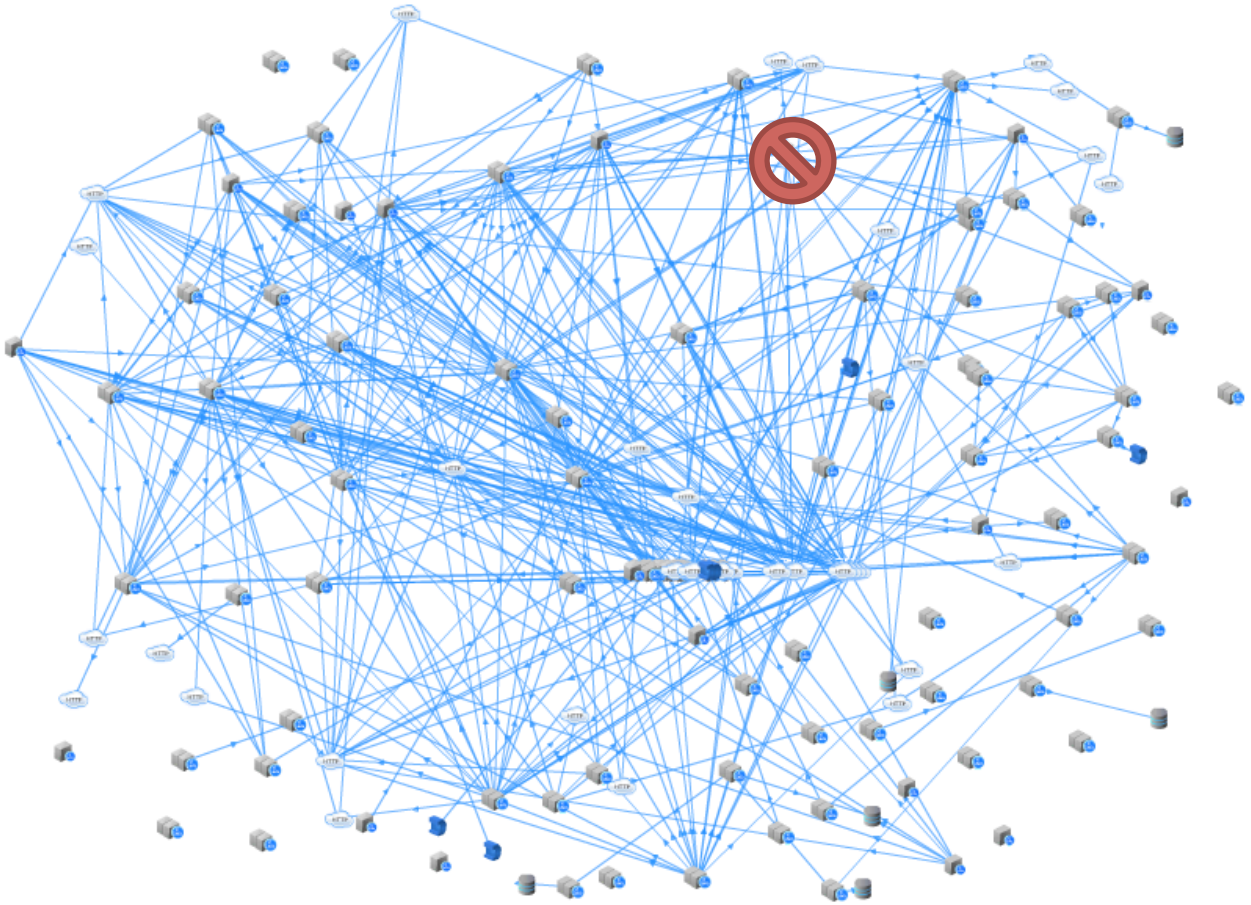
NETFLIX

Current Architectural Patterns for Availability

- Isolated Services
  - Resilient Business logic
- Three Balanced Availability Zones
  - Resilient to Infrastructure outage
- Triple Replicated Persistence
  - Durable distributed Storage
- Isolated Regions
  - US and EU don't take each other down

Isolated Services

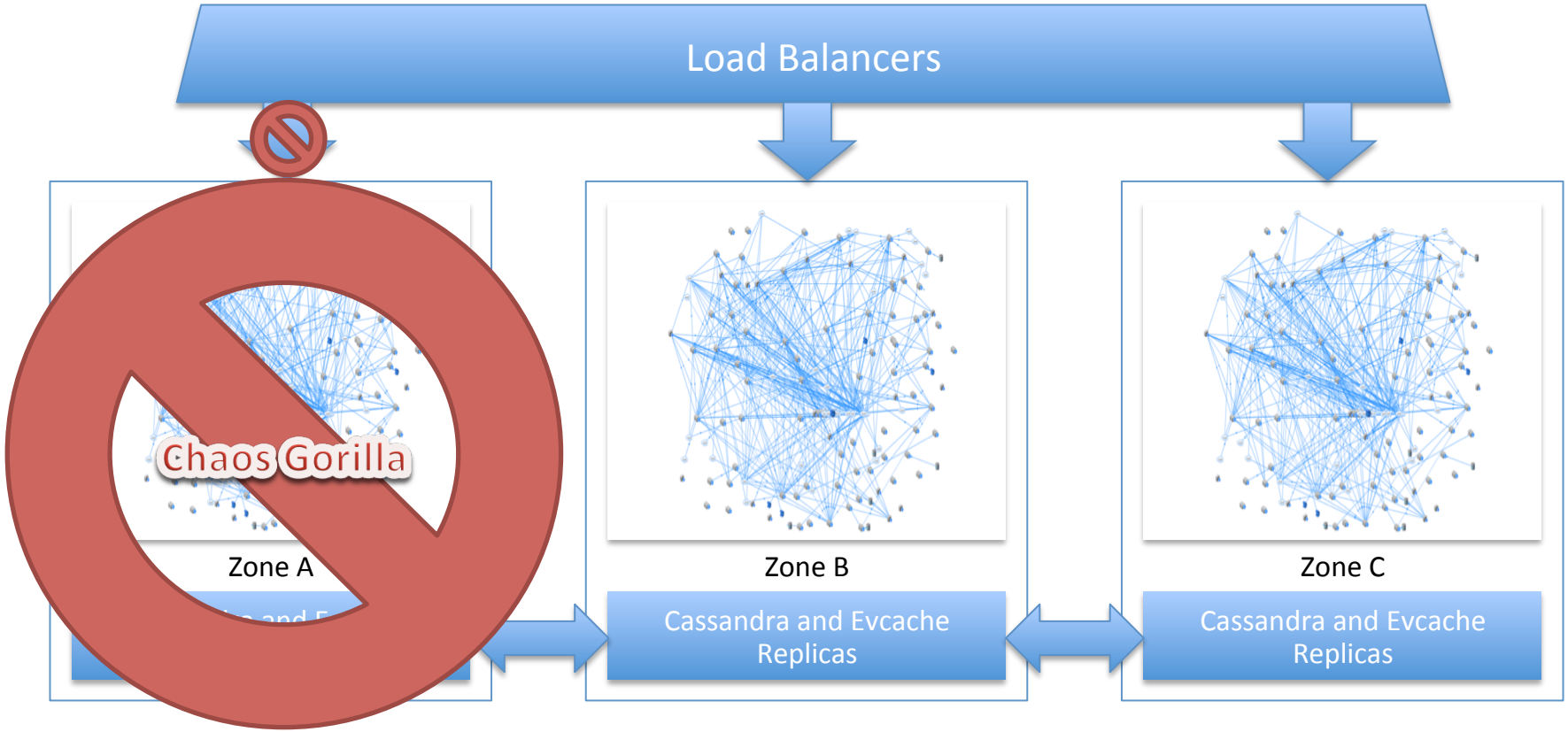
Test With Chaos Monkey, Latency Monkey



NETFLIX

Three Balanced Availability Zones

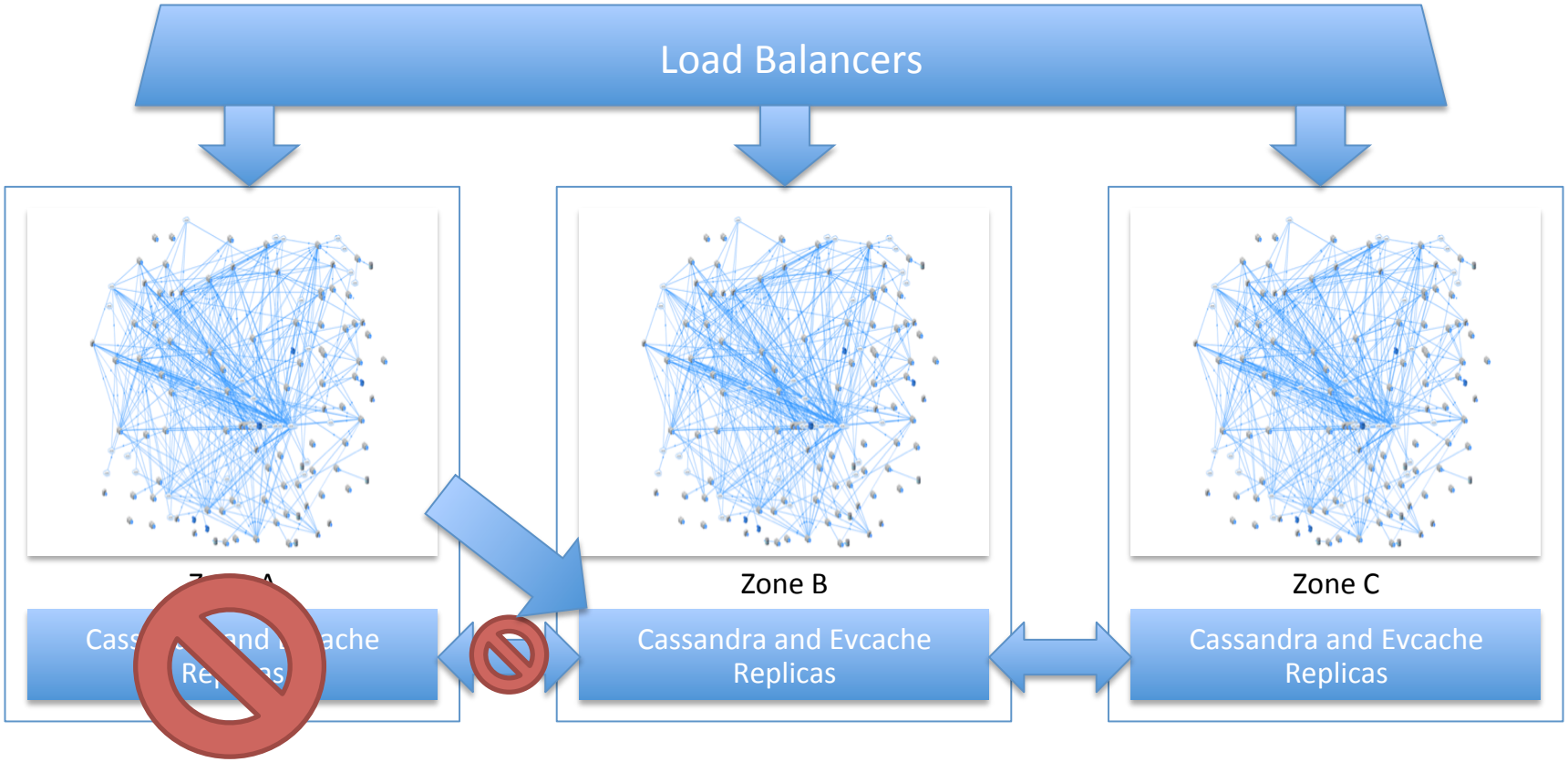
Test with Chaos Gorilla



NETFLIX

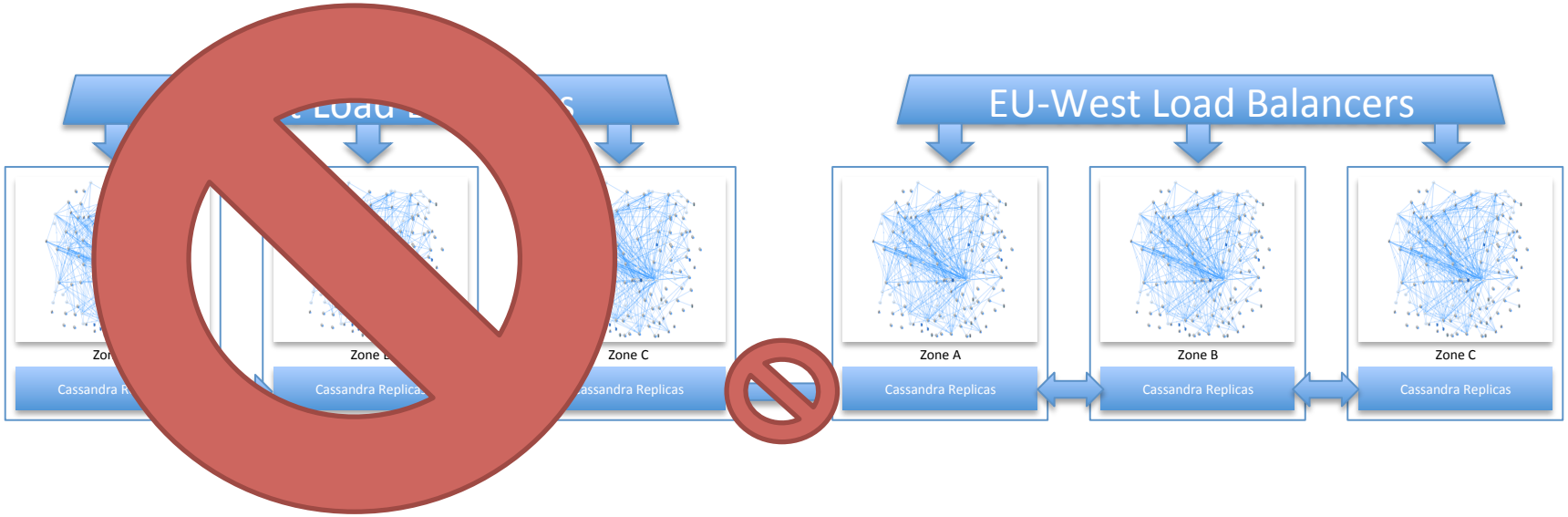
Triple Replicated Persistence

Cassandra maintenance affects individual replicas



NETFLIX

Isolated Regions



NETFLIX

Failure Modes and Effects

Failure Mode	Probability	Mitigation Plan
Application Failure	High	Automatic degraded response
AWS Region Failure	Low	Wait for region to recover
AWS Zone Failure	Medium	Continue to run on 2 out of 3 zones
Datacenter Failure	Medium	Migrate more functions to cloud
Data store failure	Low	Restore from S3 backups
S3 failure	Low	Restore from remote archive

NETFLIX

Netflix Deployed on AWS

2009	2009	2010	2010	2010	2011
Content	Logs	Play	WWW	API	CS
Content Management	S3 Terabytes	DRM	Sign-Up	Metadata	International CS lookup
EC2 Encoding	EMR	CDN routing	Search Solr	Device Config	Diagnostics & Actions
S3 Petabytes	Hive & Pig	Bookmarks	Movie Choosing	TV Movie Choosing	Customer Call Log
	Business Intelligence	Logging	Ratings	Social Facebook	CS Analytics
	hadoop	Cassandra	MySQL	Cassandra	Cassandra

CDNs
ISPs
Terabits
Customers

NETFLIX

Cloud Architecture Patterns

Where do we start?

Datacenter to Cloud Transition Goals

- Faster
  - **Lower latency** than the equivalent datacenter web pages and API calls
  - Measured as mean and 99th 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

NETFLIX

Netflix Datacenter vs. Cloud Arch

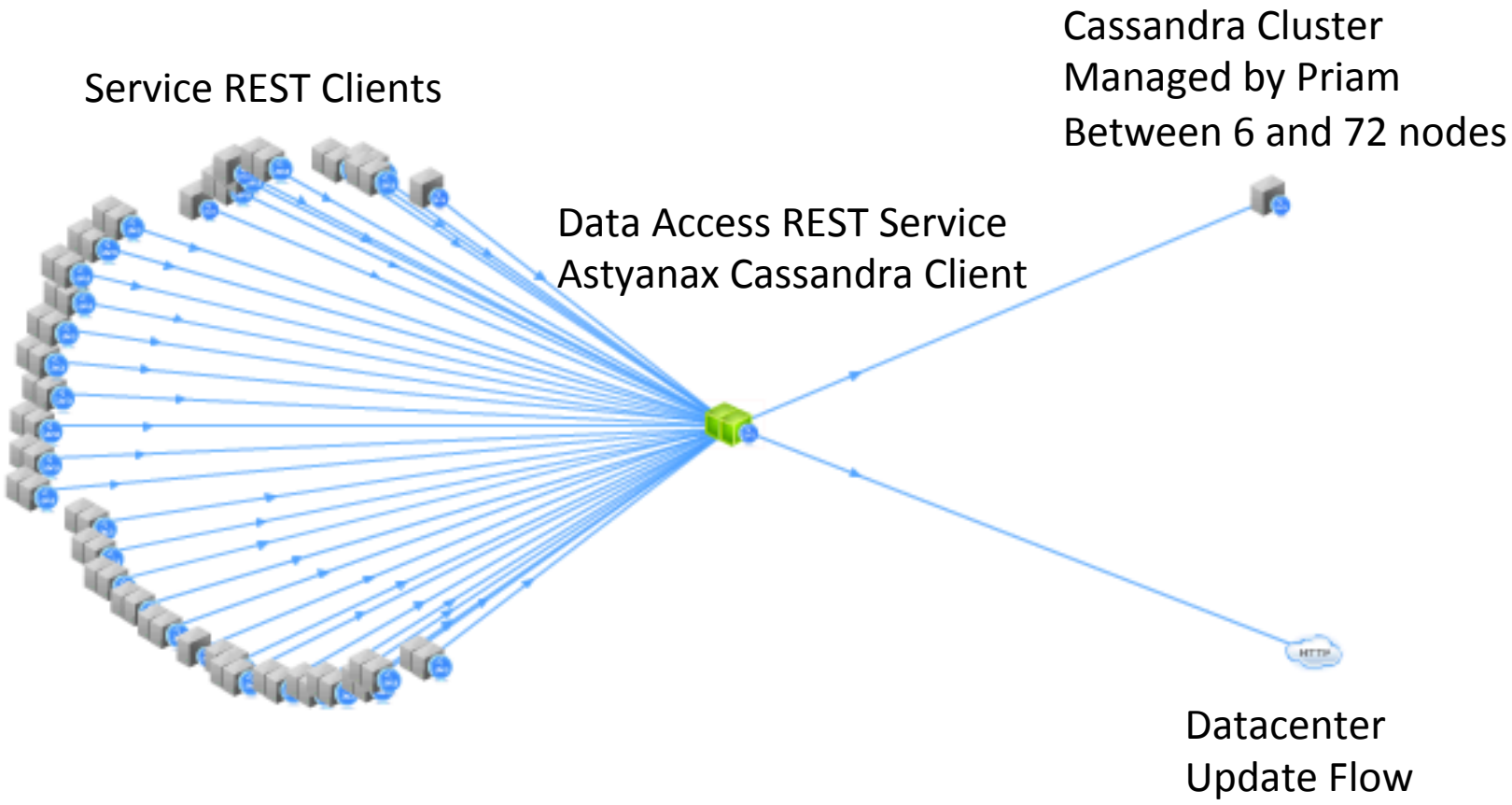
Anti-Architecture

Central SQL Database	Distributed Key/Value NoSQL
Sticky In-Memory Session	Shared Memcached Session
Chatty Protocols	Latency Tolerant Protocols
Tangled Service Interfaces	Layered Service Interfaces
Instrumented Code	Instrumented Service Patterns
Fat Complex Objects	Lightweight Serializable Objects
Components as Jar Files	Components as Services

Cassandra on AWS

A highly available and durable deployment pattern

Cassandra Service Pattern

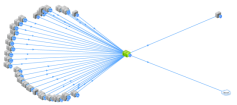


Appdynamics Service Flow Visualization

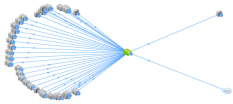
NETFLIX

Production Deployment

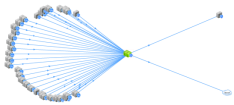
Totally Denormalized Data Model



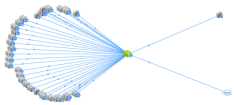
Over 50 Cassandra Clusters



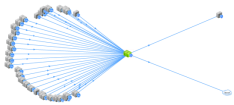
Over 500 nodes



Over 30TB of daily backups



Biggest cluster 72 nodes



1 cluster over 250Kwrites/s

NETFLIX

Astyanax - Cassandra Write Data Flows

Single Region, Multiple Availability Zone, Token Aware

1. Client Writes to local coordinator
2. Coodinator writes to other zones
3. Nodes return ack
4. Data written to internal commit log disks (no more than 10 seconds later)

If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously

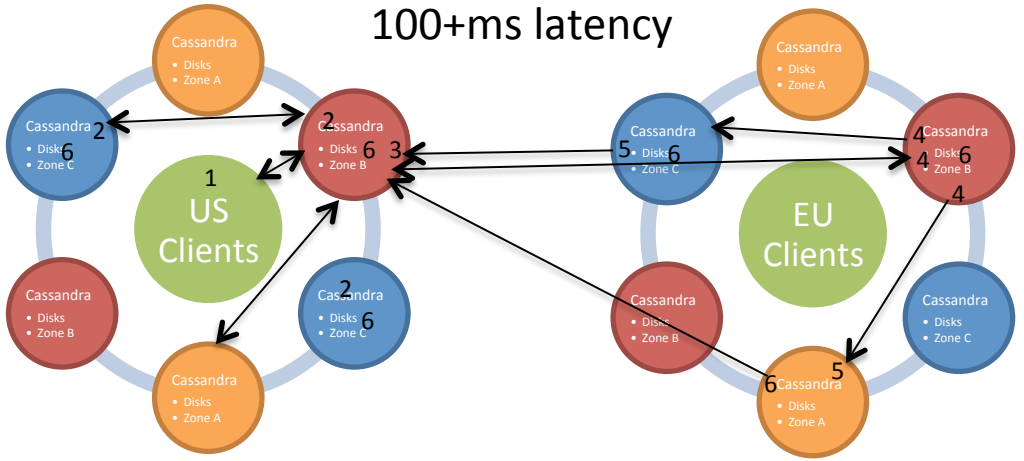
NETFLIX

Data Flows for Multi-Region Writes

Token Aware, Consistency Level = Local Quorum

1. Client writes to local replicas
2. Local write acks returned to Client which continues when 2 of 3 local nodes are committed
3. Local coordinator writes to remote coordinator.
4. When data arrives, remote coordinator node acks and copies to other remote zones
5. Remote nodes ack to local coordinator
6. Data flushed to internal commit log disks (no more than 10 seconds later)

If a node or region goes offline, hinted handoff completes the write when the node comes back up. Nightly global compare and repair jobs ensure everything stays consistent.



NETFLIX

ETL for Cassandra

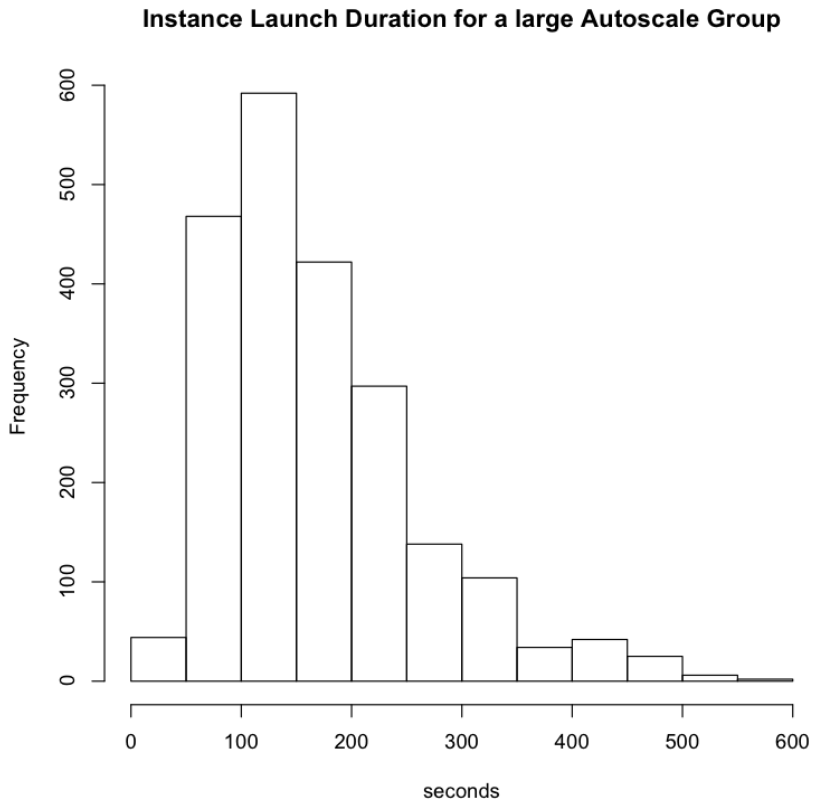
- Data is de-normalized over many clusters!
- Too many to restore from backups for ETL
- Solution – read backup files using Hadoop
- Aegisthus
  - http://techblog.netflix.com/2012/02/aegisthus-bulk-data-pipeline-out-of.html
  - High throughput raw SSTable processing
  - Re-normalizes many clusters to a consistent view
  - Extract, Transform, then Load into Teradata

Benchmarks and Scalability

Cloud Deployment Scalability

New Autoscaled AMI – zero to 500 instances from 21:38:52 - 21:46:32, 7m40s
Scaled up and down over a few days, total 2176 instance launches, m2.2xlarge (4 core 34GB)

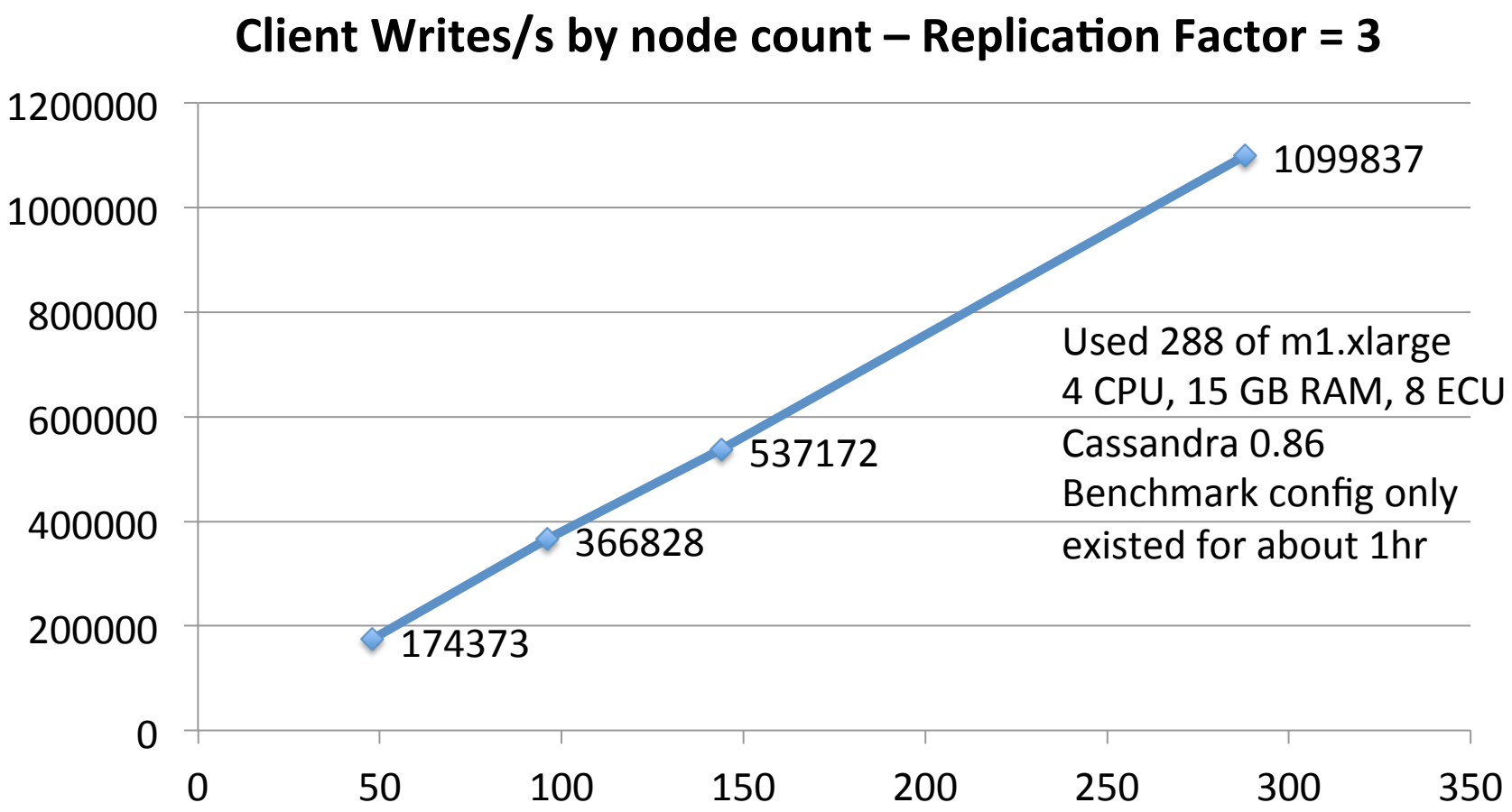
```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
41.0   104.2   149.0   171.8   215.8   562.0
```



NETFLIX

Scalability from 48 to 288 nodes on AWS

http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html



NETFLIX

Cassandra on AWS

The Past

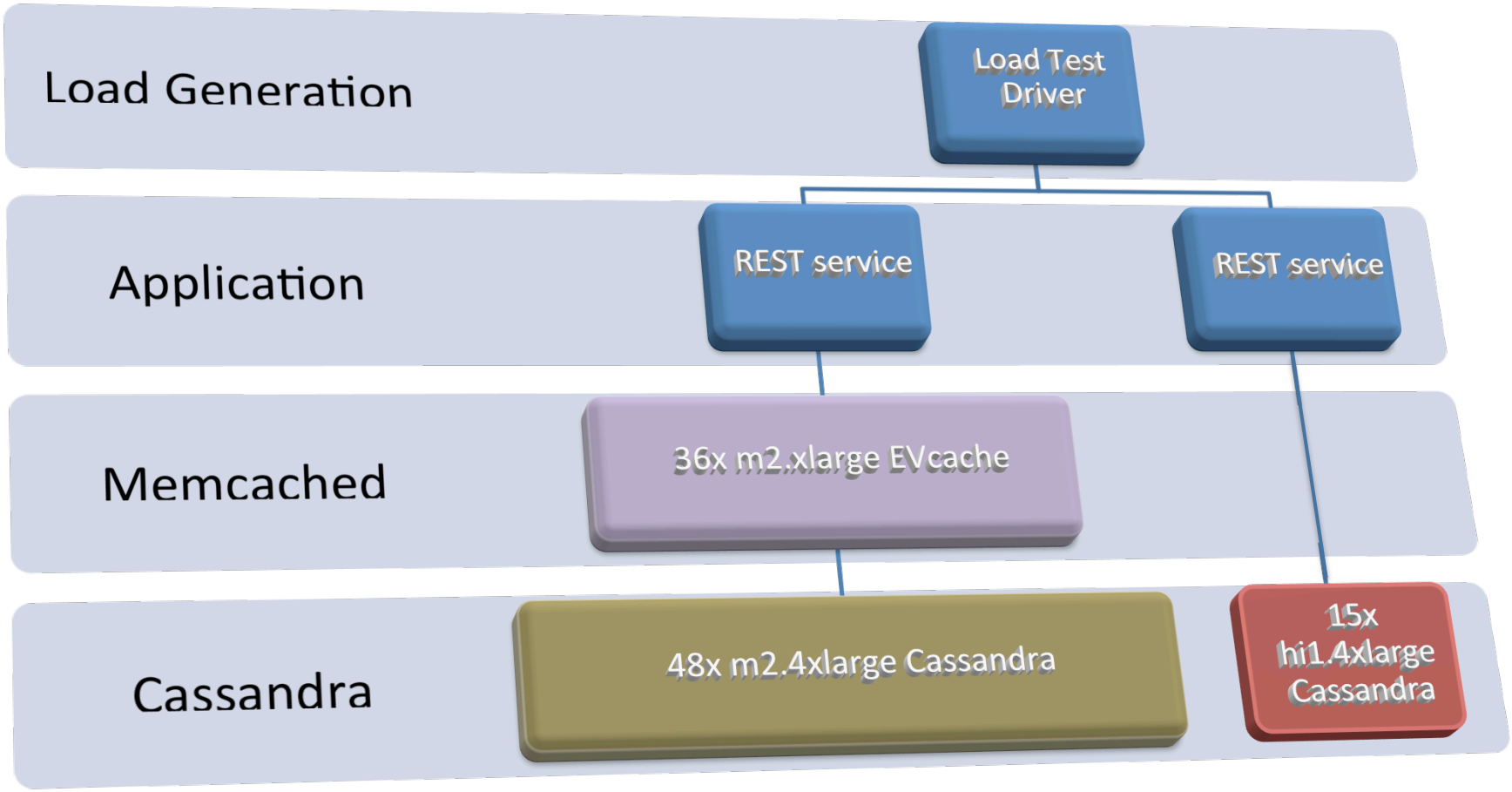
- Instance: m2.4xlarge
- Storage: 2 drives, 1.7TB
- CPU: 8 Cores, 26 ECU
- RAM: 68GB
- Network: 1Gbit
- IOPS: ~500
- Throughput: ~100Mbyte/s
- Cost: $1.80/hr

The Future

- Instance: hi1.4xlarge
- Storage: 2 SSD volumes, 2TB
- CPU: 8 HT cores, 35 ECU
- RAM: 64GB
- Network: 10Gbit
- IOPS: ~100,000
- Throughput: ~1Gbyte/s
- Cost: $3.10/hr

Cassandra Disk vs. SSD Benchmark

Same Throughput, Lower Latency, Half Cost



NETFLIX

Availability and Resilience

Chaos Monkey

http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html

- Computers (Datacenter or AWS) randomly die
  - Fact of life, but too infrequent to test resiliency
- Test to make sure systems are resilient
  - Allow any instance to fail without customer impact
- Chaos Monkey hours
  - Monday-Friday 9am-3pm random instance kill
- Application configuration option
  - Apps now have to opt-out from Chaos Monkey

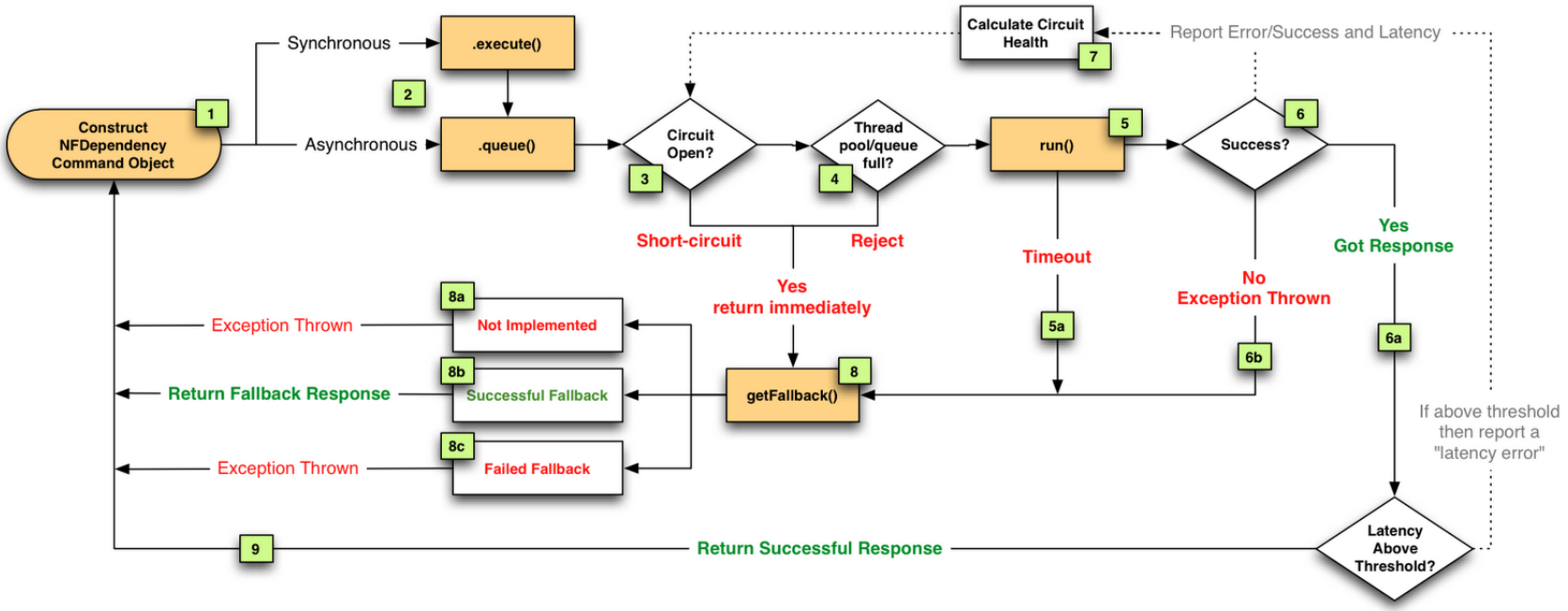
NETFLIX

Responsibility and Experience

- Make developers responsible for failures
  - Then they learn and write code that doesn't fail
- Use Incident Reviews to find gaps to fix
  - Make sure its not about finding "who to blame"
- Keep timeouts short, fail fast
  - Don't let cascading timeouts stack up
- Make configuration options dynamic
  - You don't want to push code to tweak an option

Resilient Design – Circuit Breakers

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html



NETFLIX

Distributed Operational Model

- Developers
  - Provision and run their own code in production
  - Take turns to be on call if it breaks (pagerduty)
  - Configure autoscalers to handle capacity needs

- DevOps and PaaS (aka NoOps)
  - DevOps is used to build and run the PaaS
  - PaaS constrains Dev to use automation instead
  - PaaS puts more responsibility on Dev, with tools

Culture

NETFLIX

Unconventional Culture

See culture deck at http://jobs.netflix.com

- Brave/Aggressive from the top down
- Focus on talent density above everything
- Reduce process, remove complexity
- Freedom and Responsibility
- One product focus for the whole company
- (almost) full information sharing across co.
- Simplified managers role

NETFLIX

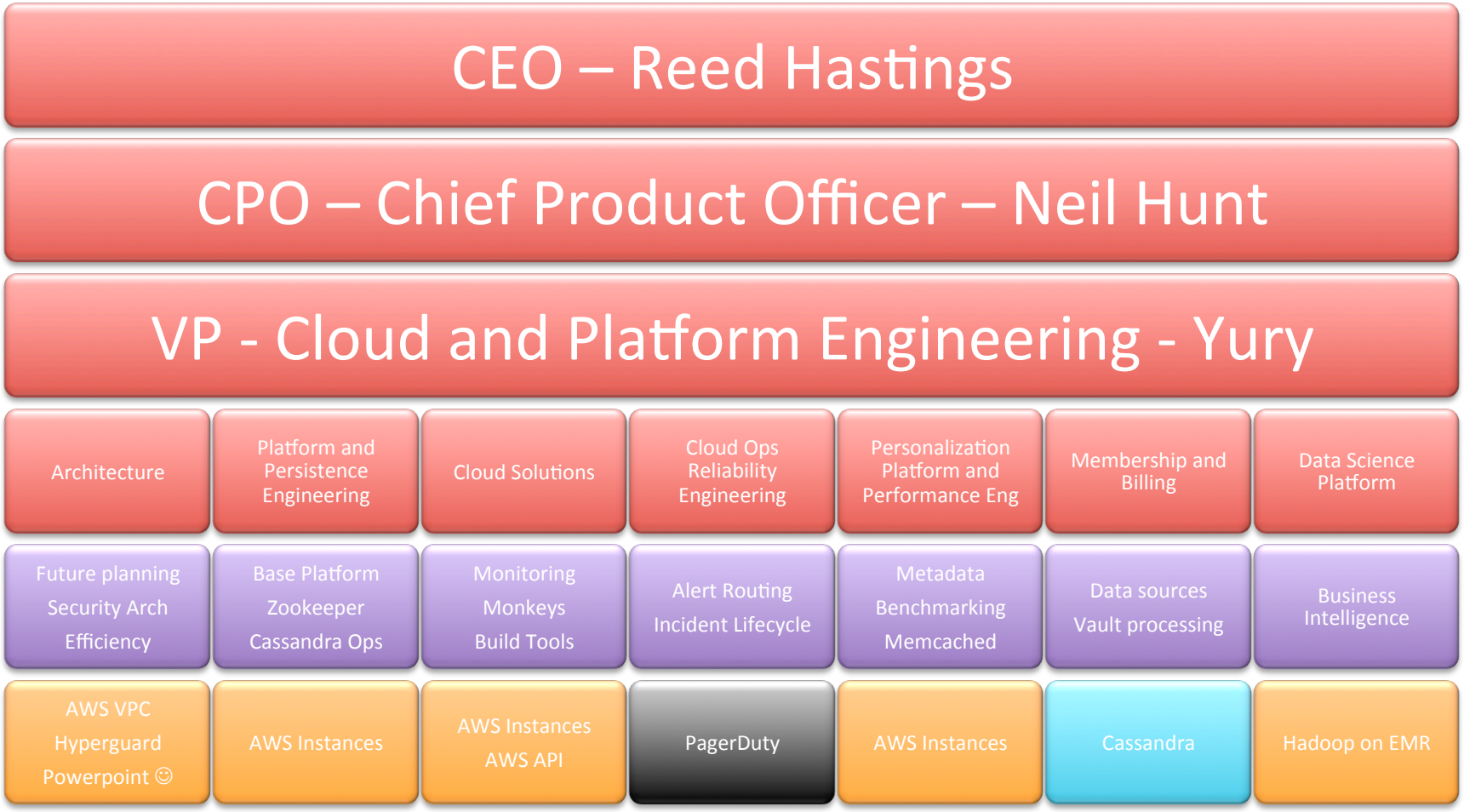
Managers Role

- Hiring, Architecture, Project Management
- No vacation policy to track
- (Almost) no remote employees or contractors
- No bonuses to allocate
- No expenses to approve
- Pay mark to market handled at VP level

NETFLIX

Netflix Organization

DevOps Org Reporting into Product Group, not ITops



NETFLIX

Build Your Own PaaS

NETFLIX

Components

- Continuous build framework turns code into AMIs
- AWS accounts for test, production, etc.
- Cloud access gateway
- Service registry
- Configuration properties service
- Persistence services
- Monitoring, alert forwarding
- Backups, archives

NETFLIX

Netflix Open Source Strategy

- Release PaaS Components git-by-git
  - Source at github.com/netflix – we build from it…
  - Intros and techniques at techblog.netflix.com
  - Blog post or new code every few weeks

- Motivations
  - Give back to Apache licensed OSS community
  - Motivate, retain, hire top engineers
  - "Peer pressure" code cleanup, external contributions

Instance creation

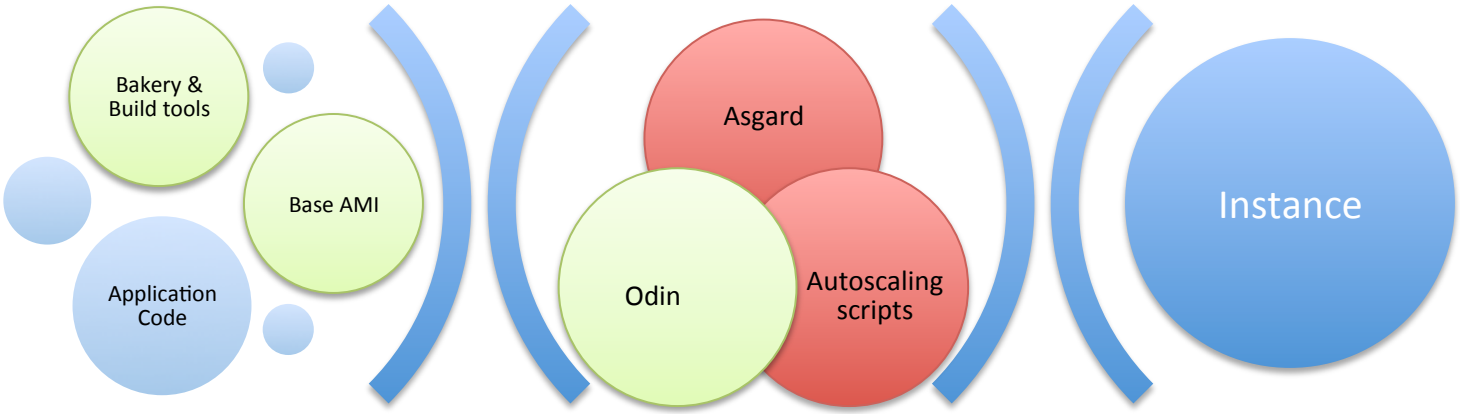


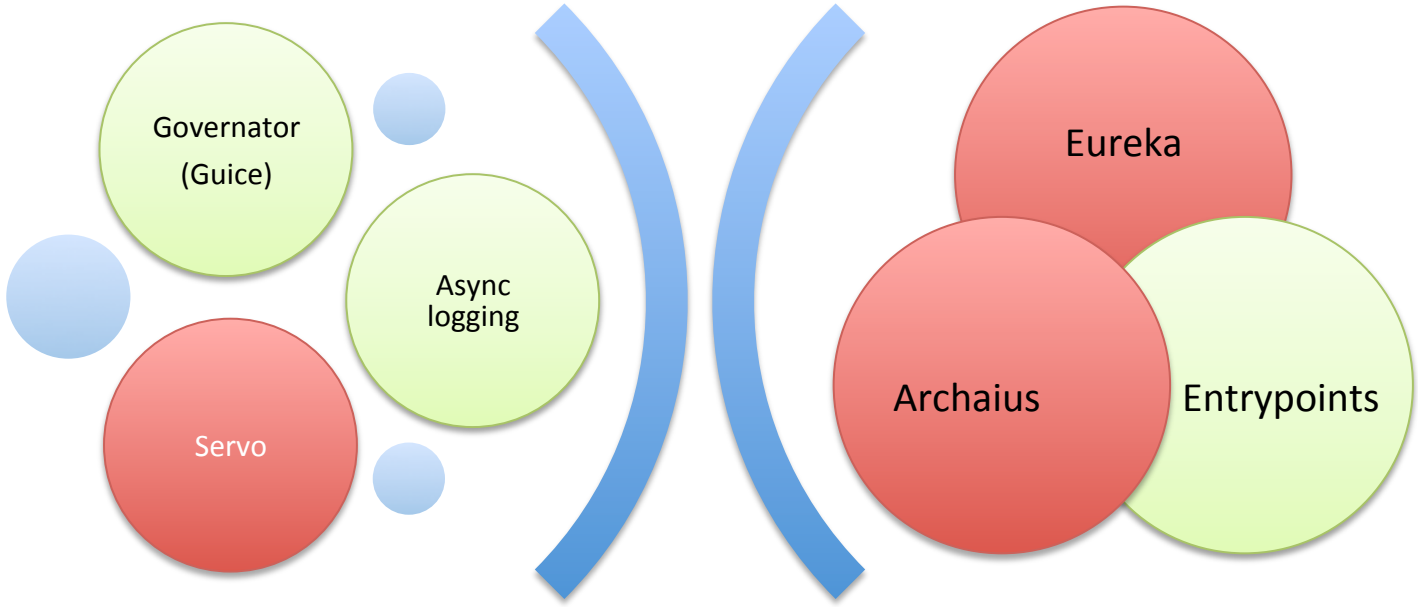
Image baked

ASG / Instance started

Instance Running

NETFLIX

Application Launch

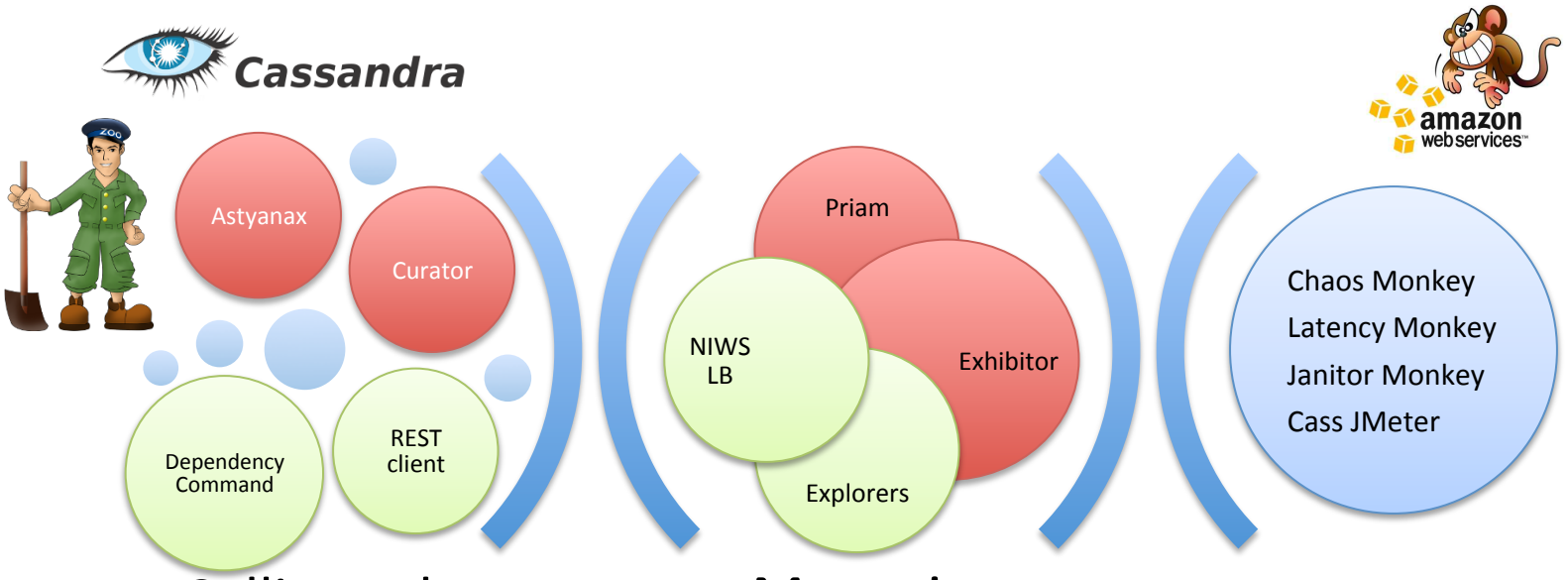


Application initializing

Registering,
configuration

NETFLIX

Runtime



NETFLIX

Open Source Projects

Legend

- Github / Techblog
- Apache Contributions
- Techblog Post
- Coming Soon

Priam Cassandra as a Service	Exhibitor Zookeeper as a Service	Servo and Autoscaling Scripts
Astyanax Cassandra client for Java	Curator Zookeeper Patterns	Honu Log4j streaming to Hadoop
CassJMeter Cassandra test suite	EVCache Memcached as a Service	Circuit Breaker Robust service pattern
Cassandra Multi-region EC2 datastore support	Eureka / Discovery Service Directory	Asgard - AutoScaleGroup based AWS console
Aegisthus Hadoop ETL for Cassandra	Archaius Dynamics Properties Service	Chaos Monkey Robustness verification
Explorers	EntryPoints	Latency Monkey
Governator - Library lifecycle and dependency injection	Server-side latency/error injection	Janitor Monkey
Odin Workflow orchestration	REST Client + mid-tier LB	Bakeries and AMI
Async logging	Configuration REST endpoints	Build dynaslaves

NETFLIX

Roadmap for 2012

- More resiliency and improved availability
- More automation, orchestration
- “Hardening” the platform, code clean-up
- Lower latency for web services and devices
- IPv6 – now running in prod, rollout in process
- More open sourced components
- See you at AWS Re:Invent in November…

NETFLIX

Takeaway

Netflix has built and deployed a scalable global Platform as a Service.

Key components of the Netflix PaaS are being released as Open Source projects so you can build your own custom PaaS.

http://github.com/Netflix
http://techblog.netflix.com
http://slideshare.net/Netflix

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

@adrianco #netflixcloud

Amazon Cloud Terminology Reference

See http://aws.amazon.com/ This is not a full list of Amazon Web Service features

- AWS – Amazon Web Services (common name for Amazon cloud)
- AMI – Amazon Machine Image (archived boot disk, Linux, Windows etc. plus application code)
- EC2 – Elastic Compute Cloud
  - Range of virtual machine types m1, m2, c1, cc, cg. Varying memory, CPU and disk configurations.
  - Instance – a running computer system. Ephemeral, when it is de-allocated nothing is kept.
  - Reserved Instances – pre-paid to reduce cost for long term usage
  - Availability Zone – datacenter with own power and cooling hosting cloud instances
  - Region – group of Avail Zones – US-East, US-West, EU-Eire, Asia-Singapore, Asia-Japan, SA-Brazil, US-Gov
- ASG – Auto Scaling Group (instances booting from the same AMI)
- S3 – Simple Storage Service (http access)
- EBS – Elastic Block Storage (network disk filesystem can be mounted on an instance)
- RDS – Relational Database Service (managed MySQL master and slaves)
- DynamoDB/SDB – Simple Data Base (hosted http based NoSQL datastore, DynamoDB replaces SDB)
- SQS – Simple Queue Service (http based message queue)
- SNS – Simple Notification Service (http and email based topics and messages)
- EMR – Elastic Map Reduce (automatically managed Hadoop cluster)
- ELB – Elastic Load Balancer
- EIP – Elastic IP (stable IP address mapping assigned to instance or ELB)
- VPC – Virtual Private Cloud (single tenant, more flexible network and security constructs)
- DirectConnect – secure pipe from AWS VPC to external datacenter
- IAM – Identity and Access Management (fine grain role based security keys)