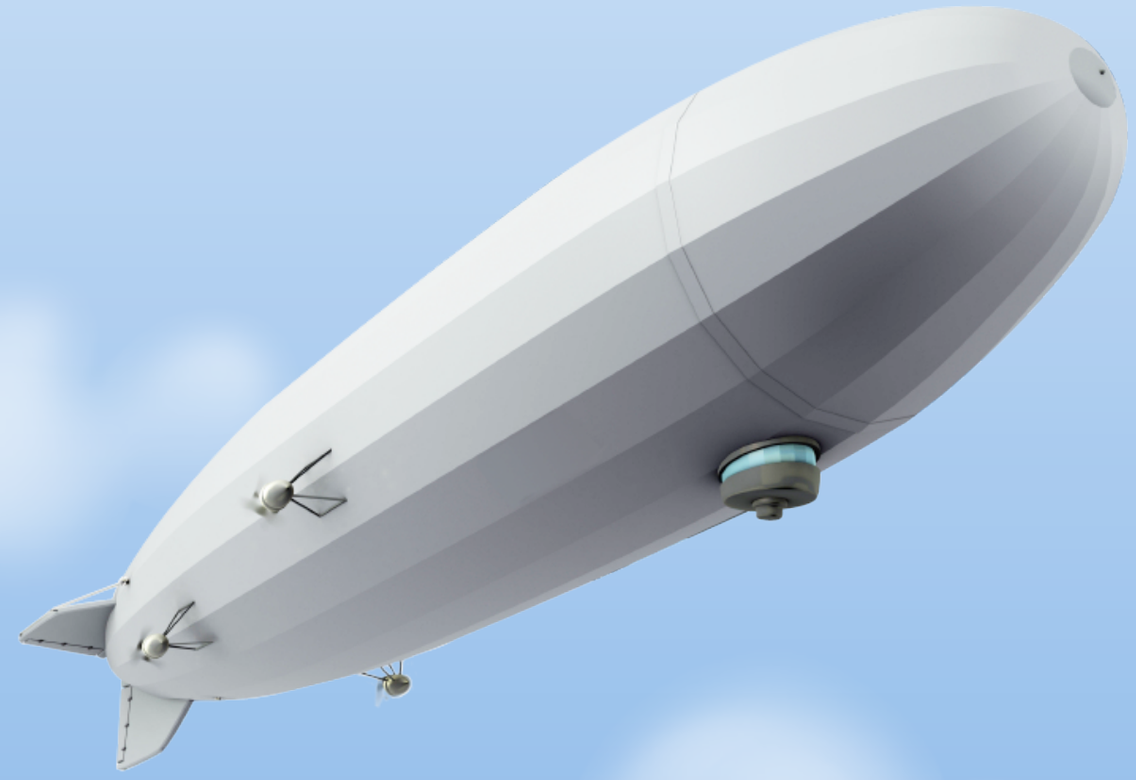


Uptime in High Volume Messaging Systems — Lessons Learned

QConSF, November 2011

Erik Onnen

About Me

- Director of Architecture and Delivery at Urban Airship
- Previously Principal Engineer at Jive Software
- 13 years writing Java, Python, C++ in highly distributed systems

In this Talk

- About Urban Airship and Lean
- Architecture Overview
- Key Learnings - Architecture, Engineering, Operations

What is an Urban Airship?

- Hosting for mobile services that developers should not build themselves
- Unified API for services across platforms
- Push, IAP, Entitlements, Content Delivery all at scale
- SLAs for throughput, latency

UA Is a Lean Company

UA Is a Lean Company

- Specifically, UA is a Lean Startup

UA Is a Lean Company

- Specifically, UA is a Lean Startup
- From Wikipedia:
  - Use of FOSS, employment of Agile Techniques
  - “Ferocious customer-centric rapid iteration, as exemplified by the Customer Development process”

UA Is a Lean Company

- Specifically, UA is a Lean Startup
- From Wikipedia:
  - Use of FOSS, employment of Agile Techniques
  - “Ferocious customer-centric rapid iteration, as exemplified by the Customer Development process”
- Attention to continuous improvement

UA Is a Lean Company

- Specifically, UA is a Lean Startup
- From Wikipedia:
  - Use of FOSS, employment of Agile Techniques
  - “Ferocious customer-centric rapid iteration, as exemplified by the Customer Development process”
- Attention to continuous improvement
- Value the elimination of waste

UA Is a Lean Company

- Specifically, UA is a Lean Startup
- From Wikipedia:
  - Use of FOSS, employment of Agile Techniques
  - “Ferocious customer-centric rapid iteration, as exemplified by the Customer Development process”
- Attention to continuous improvement
- Value the elimination of waste
- Transparent, open processes

UA Is a Lean Company

- Specifically, UA is a Lean Startup
- From Wikipedia:
  - Use of FOSS, employment of Agile Techniques
  - “Ferocious customer-centric rapid iteration, as exemplified by the Customer Development process”
- Attention to continuous improvement
- Value the elimination of waste
- Transparent, open processes
- Doesn’t apply to just Engineering and Product - also Operations and Architecture

UA By The Numbers

UA By The Numbers

- > 20K active developers

UA By The Numbers

- > 20K active developers
- Over 300 million active application installs use our APIs across > 170 million unique devices

UA By The Numbers

- > 20K active developers
- Over 300 million active application installs use our APIs across > 170 million unique devices
- 10s of billions of API requests per month

UA By The Numbers

- > 20K active developers
- Over 300 million active application installs use our APIs across > 170 million unique devices
- 10s of billions of API requests per month
- 10 million direct socket connections to our servers

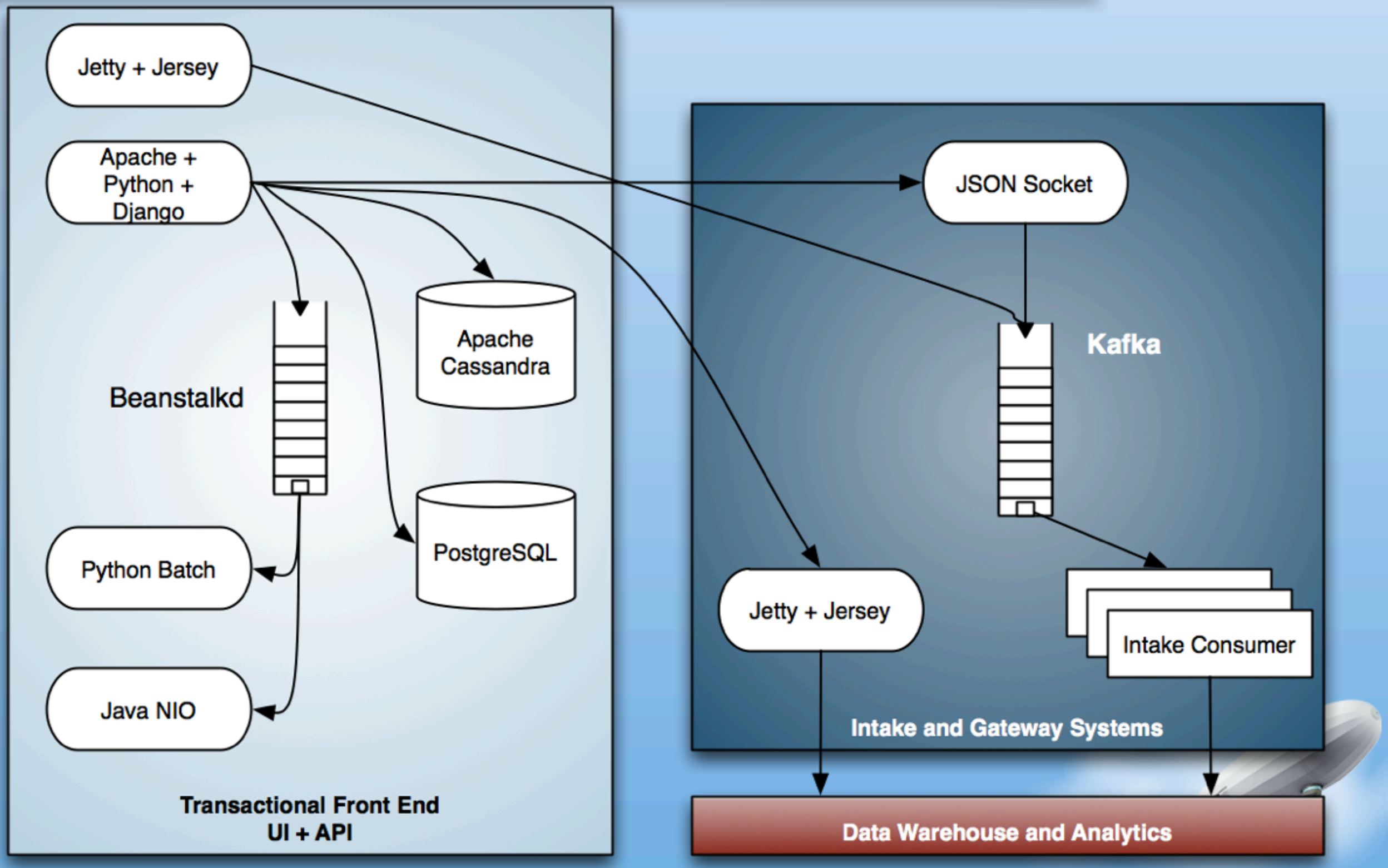
UA By The Numbers

- > 20K active developers
- Over 300 million active application installs use our APIs across > 170 million unique devices
- 10s of billions of API requests per month
- 10 million direct socket connections to our servers
- > 50 TB worth of analytics data

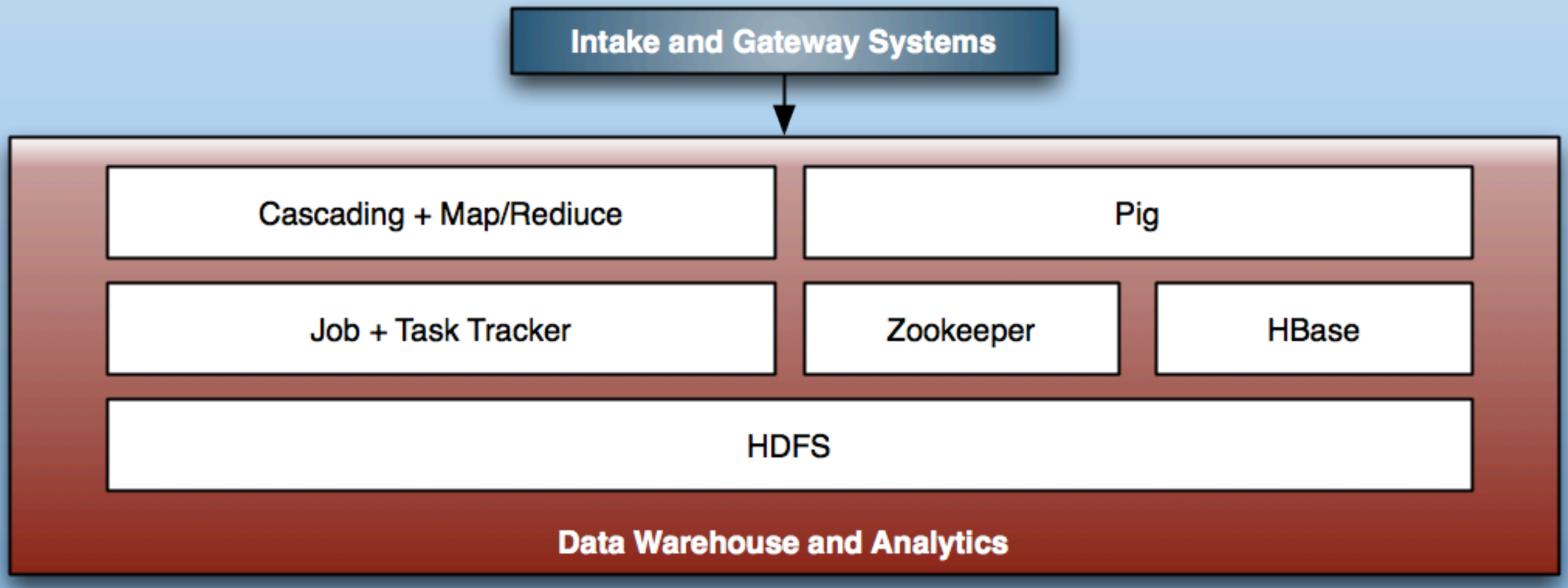
UA By The Numbers

- > 20K active developers
- Over 300 million active application installs use our APIs across > 170 million unique devices
- 10s of billions of API requests per month
- 10 million direct socket connections to our servers
- > 50 TB worth of analytics data
- 30 Engineers, 5 Operations Engineers

Obligatory Architecture Slide

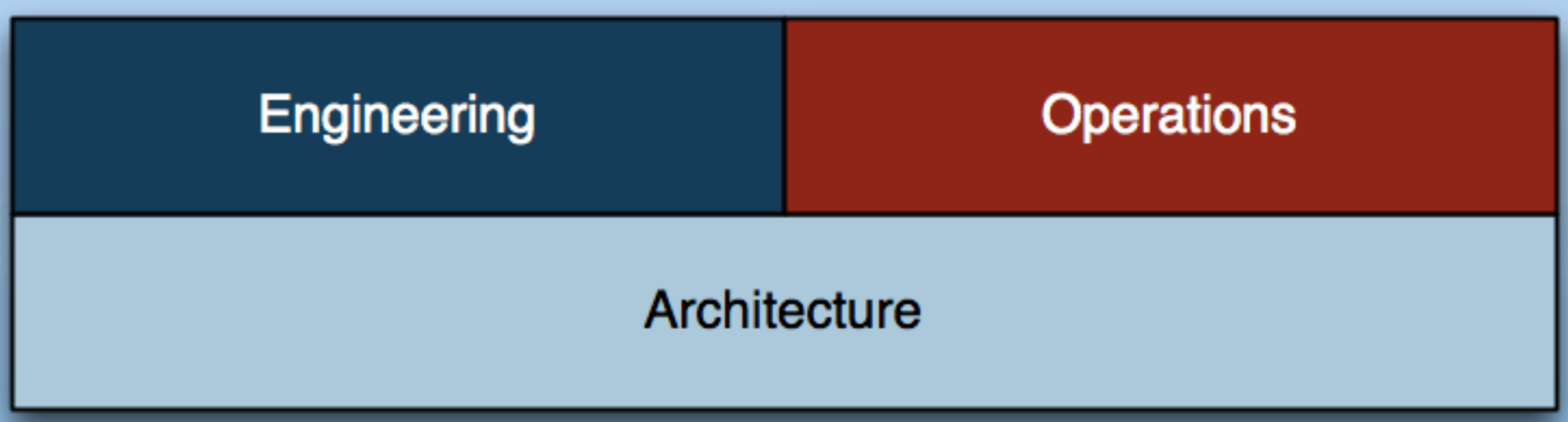


Obligatory Architecture Slide



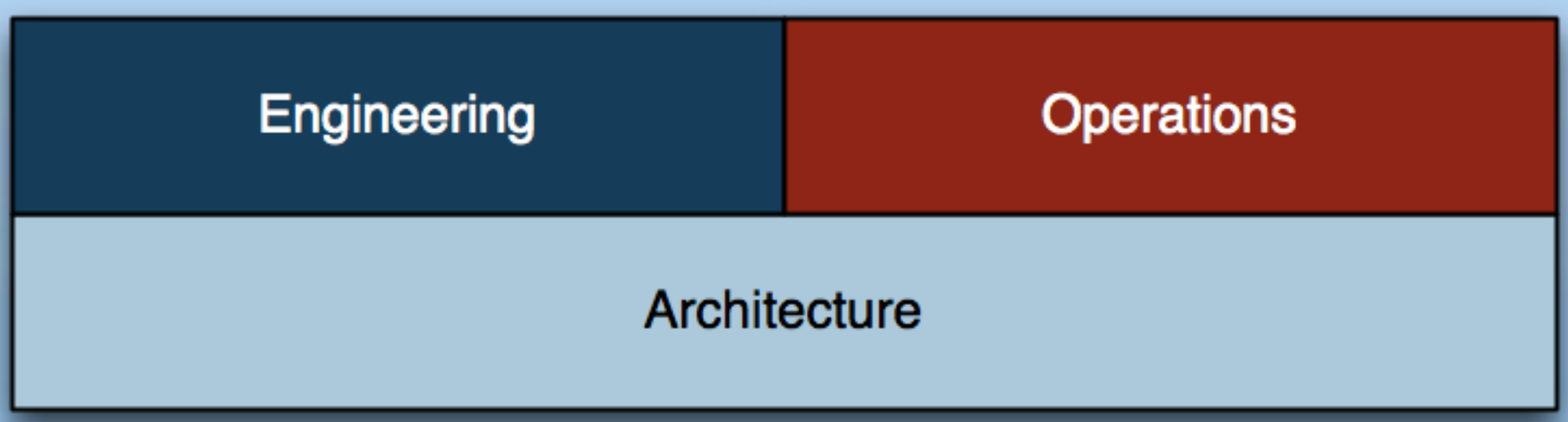
Architecture in Context

Architecture in Context



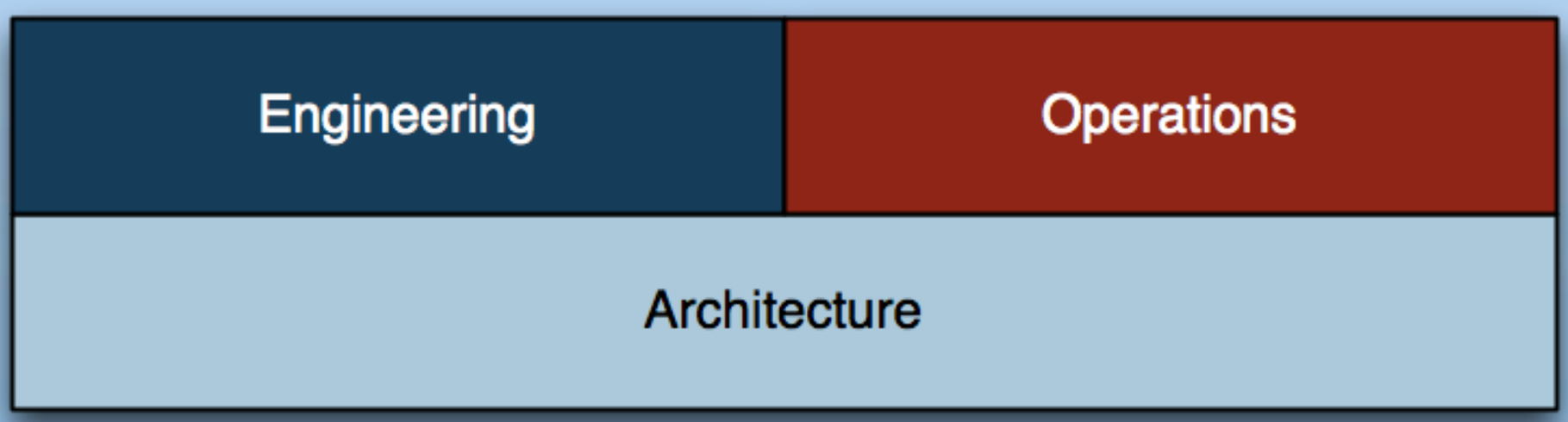
Architecture in Context

Build and Sustain



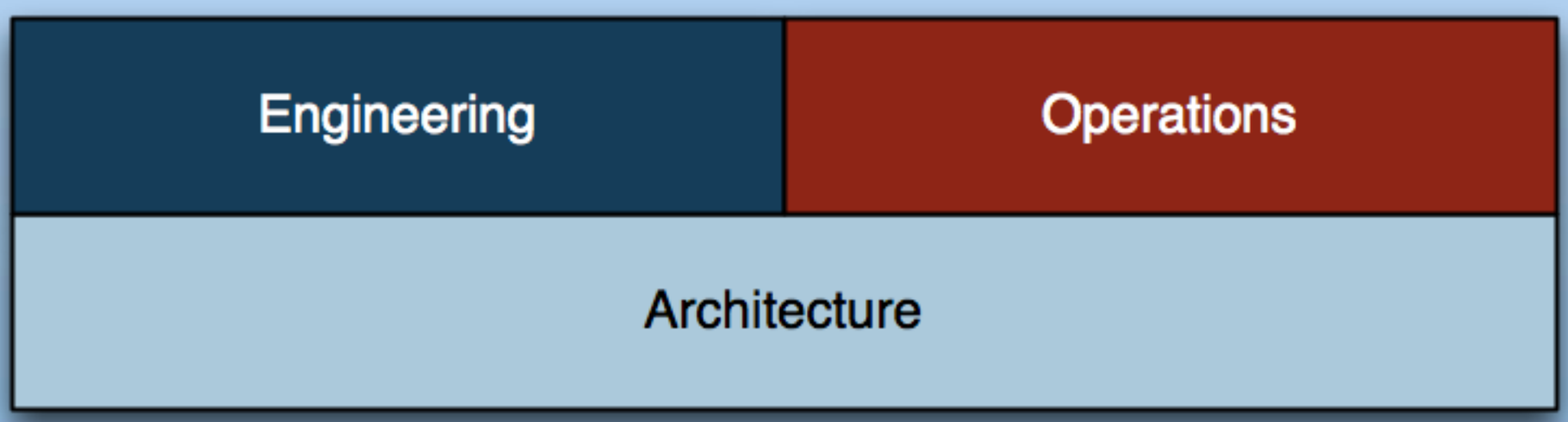
Architecture in Context

Build and Sustain Deploy and Sustain



Architecture in Context

Build and Sustain Deploy and Sustain



Principles, Guidance
and Strategy

Architecture - General Principles

Architecture - General Principles

- Keep everyone moving in the same direction

Architecture - General Principles

- Keep everyone moving in the same direction
- Help discrete teams understand how to interact

Architecture - General Principles

- Keep everyone moving in the same direction
- Help discrete teams understand how to interact
- Think in terms of small, discrete services

Architecture - General Principles

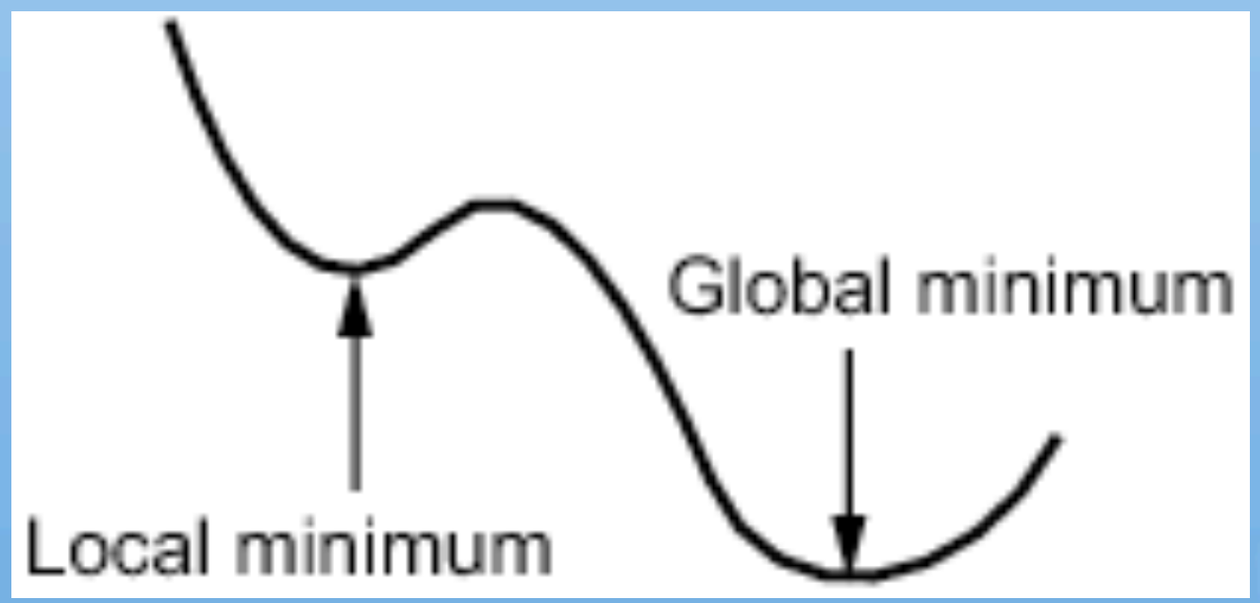
- Keep everyone moving in the same direction
- Help discrete teams understand how to interact
- Think in terms of small, discrete services
- Continuous capacity planning based on real data

Architecture - General Principles

- Keep everyone moving in the same direction
- Help discrete teams understand how to interact
- Think in terms of small, discrete services
- Continuous capacity planning based on real data
- Avoid local optimizations in decision making

Architecture - General Principles

- Keep everyone moving in the same direction
- Help discrete teams understand how to interact
- Think in terms of small, discrete services
- Continuous capacity planning based on real data
- Avoid local optimizations in decision making



Architecture - Services

Architecture - Services

- Trending towards a service-based architecture

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:
  - Minimal exposed functionality

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:
  - Minimal exposed functionality
    - Smallest reasonable surface area to the API

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:
  - Minimal exposed functionality
    - Smallest reasonable surface area to the API
  - Operate on one type of data and do it well

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:
  - Minimal exposed functionality
    - Smallest reasonable surface area to the API
    - Operate on one type of data and do it well
- Simple to operate - start/stop/status

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:
  - Minimal exposed functionality
    - Smallest reasonable surface area to the API
    - Operate on one type of data and do it well
  - Simple to operate - start/stop/status
  - Over exposure of metrics and statistics

Architecture - Services

- Trending towards a service-based architecture
- Critical traits of a service:
  - Minimal exposed functionality
    - Smallest reasonable surface area to the API
    - Operate on one type of data and do it well
  - Simple to operate - start/stop/status
  - Over exposure of metrics and statistics
  - Discoverable via ZooKeeper (future)

Architecture - Services cont.

Architecture - Services cont.

- Critical traits of a service (continued):

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services
  - No shared storage mechanism across services

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services
  - No shared storage mechanism across services
  - Minimize shared state - use ZooKeeper if absolutely necessary

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services
  - No shared storage mechanism across services
  - Minimize shared state - use ZooKeeper if absolutely necessary
  - Consistent logging, configuration

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services
  - No shared storage mechanism across services
  - Minimize shared state - use ZooKeeper if absolutely necessary
  - Consistent logging, configuration
  - Consistent implementation idioms where reasonable

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services
  - No shared storage mechanism across services
  - Minimize shared state - use ZooKeeper if absolutely necessary
  - Consistent logging, configuration
  - Consistent implementation idioms where reasonable
  - Consistent message passing approaches

Architecture - Services cont.

- Critical traits of a service (continued):
  - Zero visibility into inner workings of other services
  - No shared storage mechanism across services
  - Minimize shared state - use ZooKeeper if absolutely necessary
  - Consistent logging, configuration
  - Consistent implementation idioms where reasonable
  - Consistent message passing approaches
  - Convention for on-disk layout and structure

Architecture Waste Reduction

Architecture Waste Reduction

- All back-end services are in Java and Python

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:
  - Same core behaviors for **all** java services

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:
  - Same core behaviors for **all** java services
  - Consistent logging behavior - app, GC and stdx

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:
  - Same core behaviors for **all** java services
  - Consistent logging behavior - app, GC and stdx
  - Consistent thread dump

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:
  - Same core behaviors for **all** java services
  - Consistent logging behavior - app, GC and stdx
  - Consistent thread dump
  - Consistent heap dumps (1.6.0_25 significantly improved)

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:
  - Same core behaviors for **all** java services
  - Consistent logging behavior - app, GC and stdx
  - Consistent thread dump
  - Consistent heap dumps (1.6.0_25 significantly improved)
  - Each service runs as a system user

Architecture Waste Reduction

- All back-end services are in Java and Python
- All Java services are made to use a single set of operational scripts:
  - Same core behaviors for **all** java services
  - Consistent logging behavior - app, GC and stdx
  - Consistent thread dump
  - Consistent heap dumps (1.6.0_25 significantly improved)
  - Each service runs as a system user
  - Leveraged by init scripts

Architecture Waste Reduction

Architecture Waste Reduction

- Always look for new ways to eliminate waste

Architecture Waste Reduction

- Always look for new ways to eliminate waste
- Architectural waste comes in many forms

Architecture Waste Reduction

- Always look for new ways to eliminate waste
- Architectural waste comes in many forms
  - Maintaining lots of data storage engines - PostgreSQL, MongoDB, Cassandra, HBase, etc.

Architecture Waste Reduction

- Always look for new ways to eliminate waste
- Architectural waste comes in many forms
  - Maintaining lots of data storage engines - PostgreSQL, MongoDB, Cassandra, HBase, etc.
  - Using a complex, unfamiliar queuing system was wasteful

Architecture Waste Reduction

- Always look for new ways to eliminate waste
- Architectural waste comes in many forms
  - Maintaining lots of data storage engines - PostgreSQL, MongoDB, Cassandra, HBase, etc.
  - Using a complex, unfamiliar queuing system was wasteful
  - Large diversity in approaches for managing services, worker processes, process management, etc.

Architecture Waste Reduction

- Always look for new ways to eliminate waste
- Architectural waste comes in many forms
  - Maintaining lots of data storage engines - PostgreSQL, MongoDB, Cassandra, HBase, etc.
  - Using a complex, unfamiliar queuing system was wasteful
  - Large diversity in approaches for managing services, worker processes, process management, etc.
  - Developer silos - avoid the bus factor

Architecture - Fault Domains

Architecture - Fault Domains

- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"

Architecture - Fault Domains

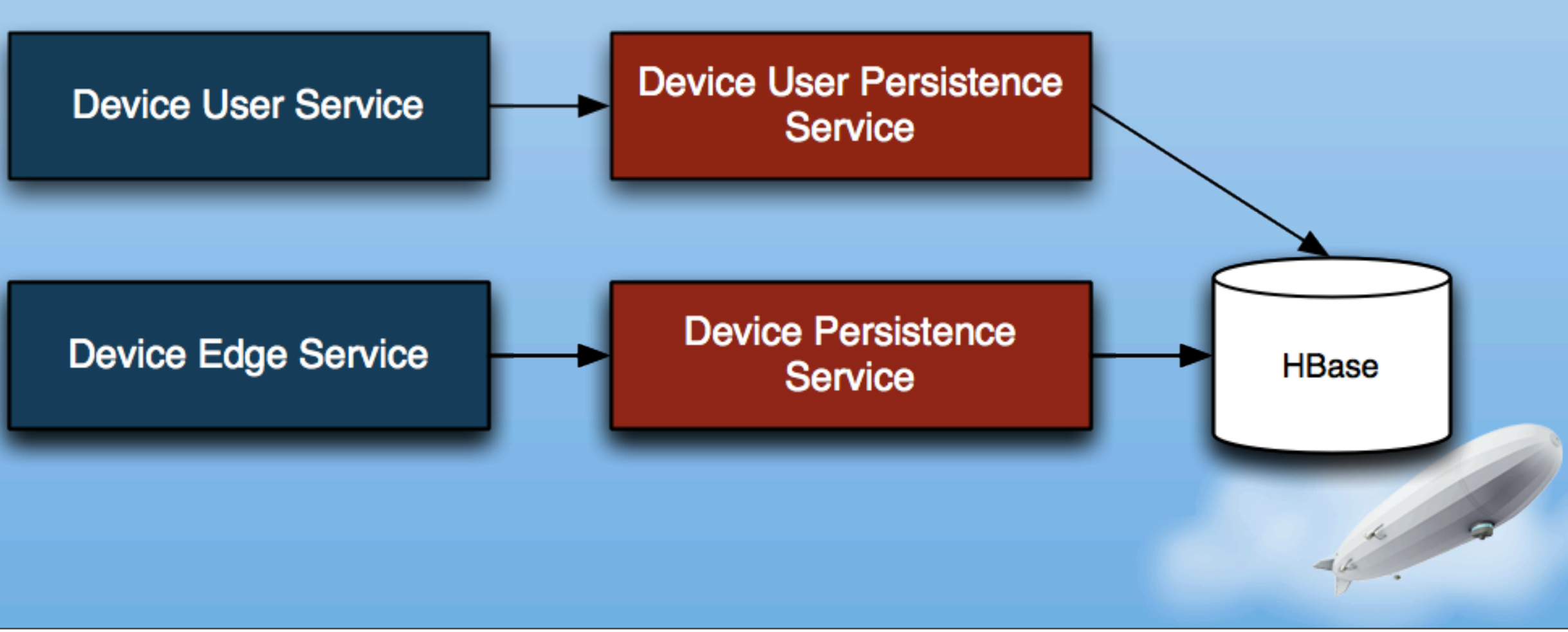
- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"
- When two services are completely unrelated, they should be in isolated fault domains if at all possible

Architecture - Fault Domains

- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"
- When two services are completely unrelated, they should be in isolated fault domains if at all possible

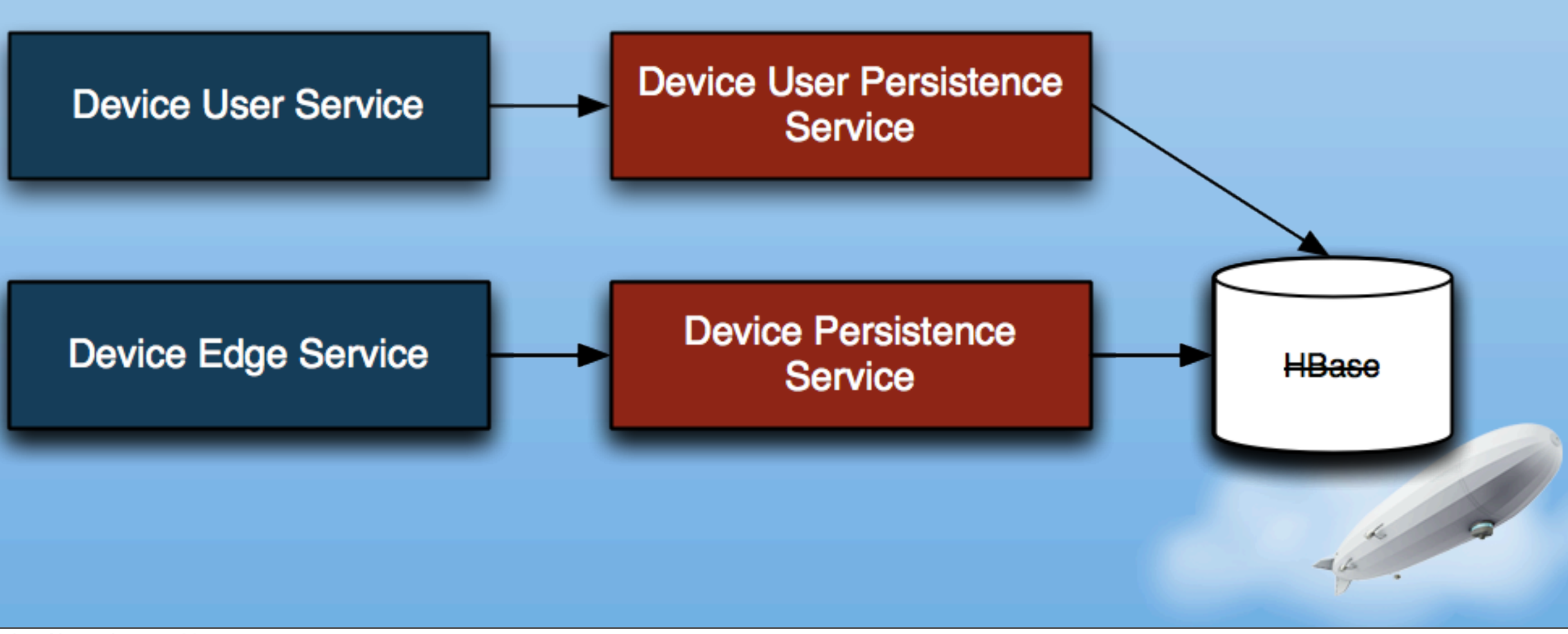
Architecture - Fault Domains

- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"
- When two services are completely unrelated, they should be in isolated fault domains if at all possible



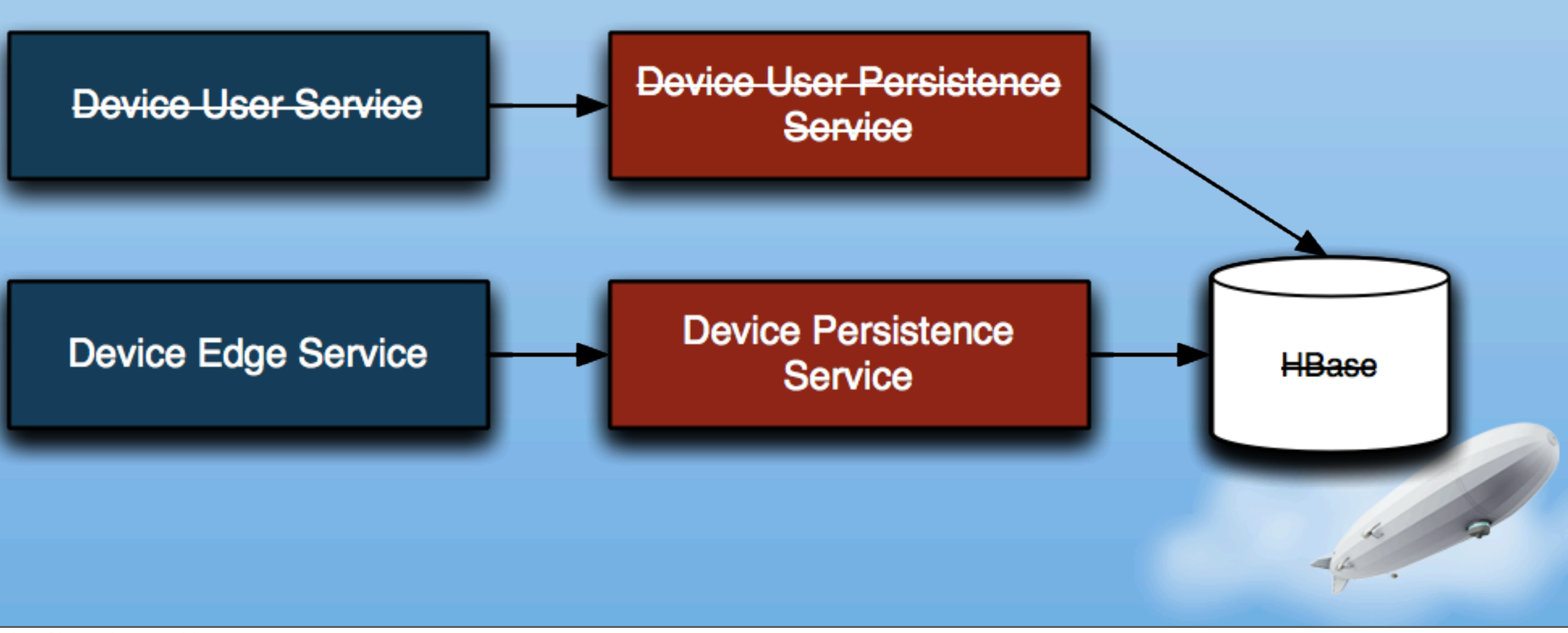
Architecture - Fault Domains

- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"
- When two services are completely unrelated, they should be in isolated fault domains if at all possible



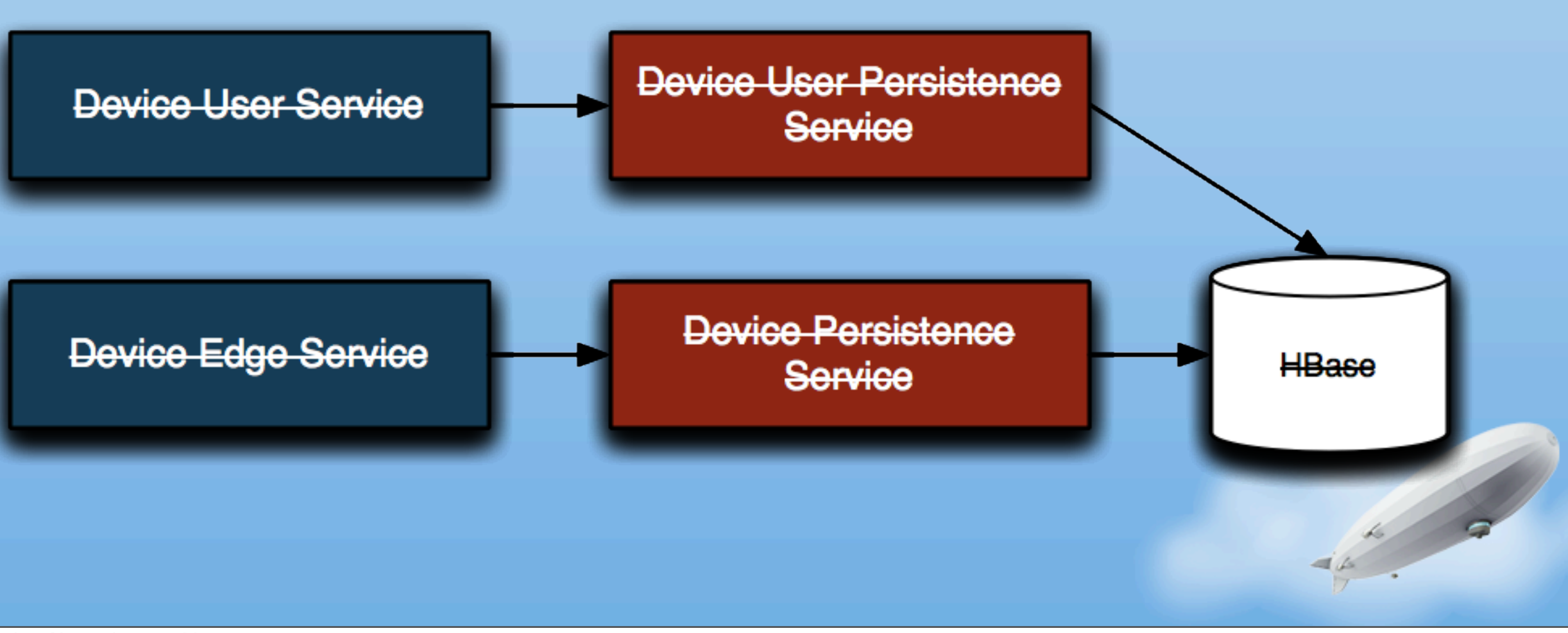
Architecture - Fault Domains

- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"
- When two services are completely unrelated, they should be in isolated fault domains if at all possible



Architecture - Fault Domains

- A Fault Domain is the full extent of peer services that are impacted by a single service outage - "*If we lose the Database, the web site goes down*"
- When two services are completely unrelated, they should be in isolated fault domains if at all possible



Architecture - Fault Domains

Architecture - Fault Domains

- Fault domains extend beyond the firewall

Architecture - Fault Domains

- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint

Architecture - Fault Domains

- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint
  - Weren't able to separate our major classes of traffic:

Architecture - Fault Domains

- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint
  - Weren't able to separate our major classes of traffic:
    - Device - check in, update state, disable services

Architecture - Fault Domains

- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint
  - Weren't able to separate our major classes of traffic:
    - Device - check in, update state, disable services
    - API - customers making requests, sending messages

Architecture - Fault Domains

- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint
  - Weren't able to separate our major classes of traffic:
    - Device - check in, update state, disable services
    - API - customers making requests, sending messages
    - UI - standard web application, push composer, etc.

Architecture - Fault Domains

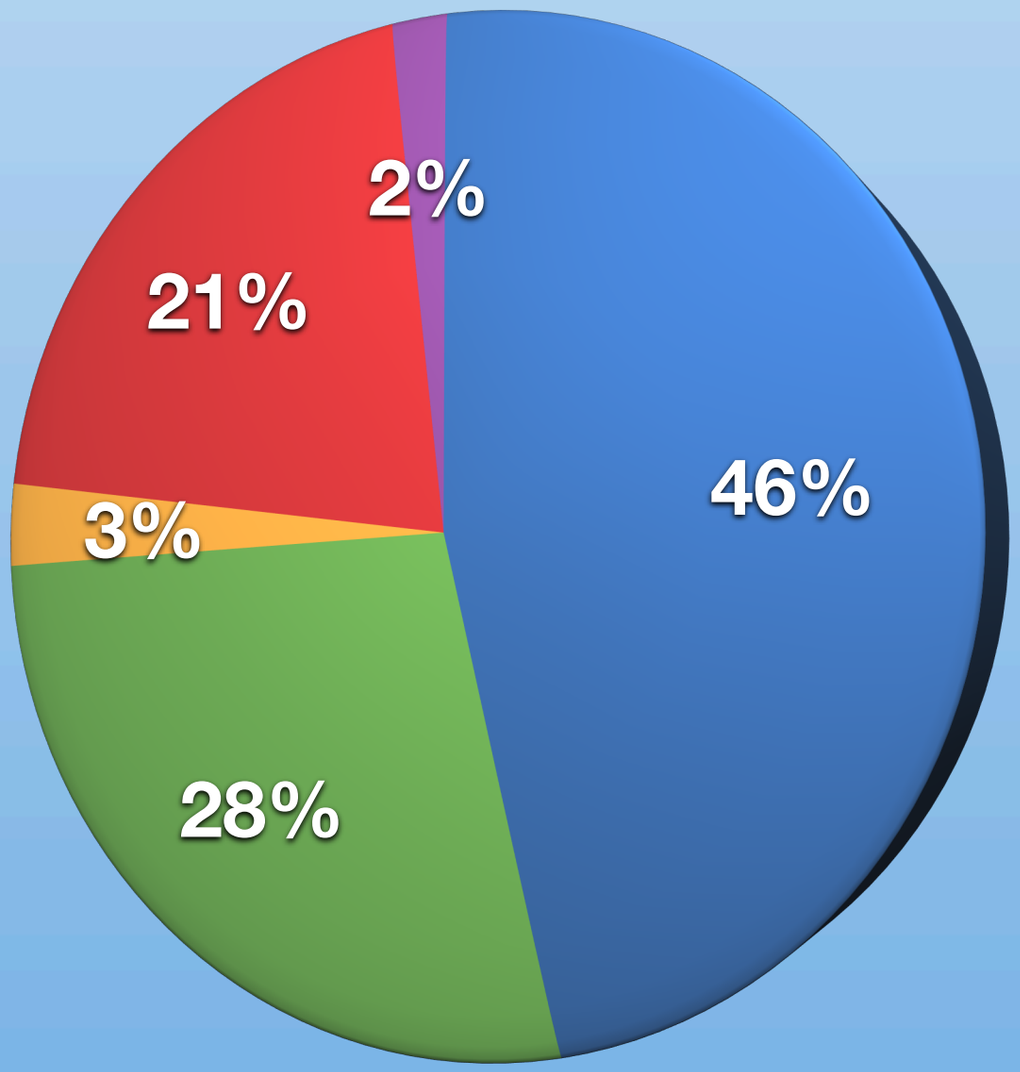
- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint
  - Weren't able to separate our major classes of traffic:
    - Device - check in, update state, disable services
    - API - customers making requests, sending messages
    - UI - standard web application, push composer, etc.
- Large launches would strain the web UI

Architecture - Fault Domains

- Fault domains extend beyond the firewall
- Originally, all of our traffic came in through one HTTPs endpoint
  - Weren't able to separate our major classes of traffic:
    - Device - check in, update state, disable services
    - API - customers making requests, sending messages
    - UI - standard web application, push composer, etc.
  - Large launches would strain the web UI
- Today - align fault domains w/ DNS, keep the distinct traffic classes separate

Engineering at UA

A Day in UA Engineering



- New Feature Development
- Sustaining Engineering
- IRC Tomfoolery
- Production Support
- Beer/Pong

Engineering for Iteration

Engineering for Iteration

- Team of > 30 engineers

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area
- Shortest iterations possible - MVP

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area
- Shortest iterations possible - MVP
- We have no formal QA team - engineers are responsible for integration up through deploy

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area
- Shortest iterations possible - MVP
- We have no formal QA team - engineers are responsible for integration up through deploy
- Frequently pair but it’s not institutionalized as a rule

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area
- Shortest iterations possible - MVP
- We have no formal QA team - engineers are responsible for integration up through deploy
- Frequently pair but it's not institutionalized as a rule
- Always leave the code better than how you found it

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area
- Shortest iterations possible - MVP
- We have no formal QA team - engineers are responsible for integration up through deploy
- Frequently pair but it's not institutionalized as a rule
- Always leave the code better than how you found it
- All bug fixes require a code review in Review Board

Engineering for Iteration

- Team of > 30 engineers
- Organized around functional area
- Shortest iterations possible - MVP
- We have no formal QA team - engineers are responsible for integration up through deploy
- Frequently pair but it's not institutionalized as a rule
- Always leave the code better than how you found it
- All bug fixes require a code review in Review Board
- Large new development undergoes a sit-down code and design review

Engineering For Automation

Engineering For Automation

- All code has three levels of testing:

Engineering For Automation

- All code has three levels of testing:
  - Unit - pure algorithmic testing (parsing, date maths, etc.)

Engineering For Automation

- All code has three levels of testing:
  - Unit - pure algorithmic testing (parsing, date maths, etc.)
  - Functional - local testing of APIs, commonly with mock objects

Engineering For Automation

- All code has three levels of testing:
  - Unit - pure algorithmic testing (parsing, date maths, etc.)
  - Functional - local testing of APIs, commonly with mock objects
  - Integration - relies on external services (goes to the wire)

Engineering For Automation

- All code has three levels of testing:
  - Unit - pure algorithmic testing (parsing, date maths, etc.)
  - Functional - local testing of APIs, commonly with mock objects
  - Integration - relies on external services (goes to the wire)
- Commits are done to a main git branch that is run through automated unit and functional testing

Engineering For Automation

- All code has three levels of testing:
  - Unit - pure algorithmic testing (parsing, date maths, etc.)
  - Functional - local testing of APIs, commonly with mock objects
  - Integration - relies on external services (goes to the wire)
- Commits are done to a main git branch that is run through automated unit and functional testing
- No promotion to the production branch until tests pass

Engineering For Simplicity

Engineering For Simplicity

- Strive for simplicity and consistency

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise
  - Java services use no containers, no EJBs, no WARs

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise
  - Java services use no containers, no EJBs, no WARs
- Vet new components and services

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise
  - Java services use no containers, no EJBs, no WARs
- Vet new components and services
  - Make sure the operational overhead is justified relative to benefits

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise
  - Java services use no containers, no EJBs, no WARs
- Vet new components and services
  - Make sure the operational overhead is justified relative to benefits
  - Make sure the sustaining overhead is justified and even achievable

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise
  - Java services use no containers, no EJBs, no WARs
- Vet new components and services
  - Make sure the operational overhead is justified relative to benefits
  - Make sure the sustaining overhead is justified and even achievable
  - Do they perform as expected?

Engineering For Simplicity

- Strive for simplicity and consistency
  - Classes, methods - small and concise
  - Java services use no containers, no EJBs, no WARs
- Vet new components and services
  - Make sure the operational overhead is justified relative to benefits
  - Make sure the sustaining overhead is justified and even achievable
  - Do they perform as expected?
  - What dependencies do they bring?

Engineering For Simplicity

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating
- We write lots of concurrent code - don’t make it any harder than it needs to be

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating
- We write lots of concurrent code - don’t make it any harder than it needs to be
- Favor immutable data structures

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating
- We write lots of concurrent code - don’t make it any harder than it needs to be
- Favor immutable data structures
  - new Foo(1,2,3) - YES!

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating
- We write lots of concurrent code - don’t make it any harder than it needs to be
- Favor immutable data structures
  - new Foo(1,2,3) - YES!
  - new Foo().setOne(1).setTwo(2).setThree(3) - NO!

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating
- We write lots of concurrent code - don’t make it any harder than it needs to be
- Favor immutable data structures
  - new Foo(1,2,3) - YES!
  - new Foo().setOne(1).setTwo(2).setThree(3) - NO!
- Avoid degenerate failure scenarios

Engineering For Simplicity

- Clean, expected failure conditions in and out of services - exceptions are truly exceptional, not a means of communicating
- We write lots of concurrent code - don't make it any harder than it needs to be
  - Favor immutable data structures
    - new Foo(1,2,3) - YES!
    - new Foo().setOne(1).setTwo(2).setThree(3) - NO!
  - Avoid degenerate failure scenarios
- Avoid asynchronous operations unless absolutely necessary

Engineering For Transparency

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**
- Capture latency for:

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**
- Capture latency for:
  - Service critical operations

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**
- Capture latency for:
  - Service critical operations
  - External service invocation

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**
- Capture latency for:
  - Service critical operations
  - External service invocation
- Capture counters for:

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**
- Capture latency for:
  - Service critical operations
  - External service invocation
- Capture counters for:
  - Service critical operations

Engineering For Transparency

- Simplify metrics and statistics capture and do it **everywhere**
- Capture latency for:
  - Service critical operations
  - External service invocation
- Capture counters for:
  - Service critical operations
  - Service faults

Putting it all Together

```
public OpLogConsumer(RuntimeContext context, ServiceContext services, LagUpdater lagUpdater, Kaf
    super(tg, "OpLog Consumer " + workerCount.incrementAndGet());
    this.setDaemon(false);
    this.configuration = context.radonConfiguration;
    this.registry = services.getApidRegistry();
    this.clusterState = services.getStorageState();
    this.lagUpdater = lagUpdater;
    this.serviceContext = services;

    this.stream = stream;
    this.maxFaults = configuration.opLogMaxFaults;

    this.opLogLag = context.metricsRegistry.newHistogram(this.getClass(), "ConsumerLag", true);
    this.opLogSuccesses = context.metricsRegistry.newCounter(this.getClass(), "Successes");
    this.opLogFaults = context.metricsRegistry.newCounter(this.getClass(), "Faults");
    this.opLogHeartbeats = context.metricsRegistry.newCounter(this.getClass(), "Heartbeats");
    this.opLogMutations = context.metricsRegistry.newCounter(this.getClass(), "Mutations");
    this.opLogMutationRate = context.metricsRegistry.newMeter(this.getClass(), "MutationRate", "
}
```

Engineering For Operations

Engineering For Operations

- Engineers, not Ops create, test deploy scripts

Engineering For Operations

- Engineers, not Ops create, test deploy scripts
  - Tools of the trade include Chef, Puppet and Fabric, Linux distro packages

Engineering For Operations

- Engineers, not Ops create, test deploy scripts
  - Tools of the trade include Chef, Puppet and Fabric, Linux distro packages
  - Part of a work item being done includes its deployment automation

Engineering For Operations

- Engineers, not Ops create, test deploy scripts
  - Tools of the trade include Chef, Puppet and Fabric, Linux distro packages
  - Part of a work item being done includes its deployment automation
- Engineers will commonly do service deployments and updates but always through the automation tools

Engineering For Operations

- Engineers, not Ops create, test deploy scripts
  - Tools of the trade include Chef, Puppet and Fabric, Linux distro packages
  - Part of a work item being done includes its deployment automation
- Engineers will commonly do service deployments and updates but always through the automation tools
- Automation scripts always pull from a prod git branch after passing auto and manual tests

Engineering For Operations

- Engineers, not Ops create, test deploy scripts
  - Tools of the trade include Chef, Puppet and Fabric, Linux distro packages
  - Part of a work item being done includes its deployment automation
- Engineers will commonly do service deployments and updates but always through the automation tools
- Automation scripts always pull from a prod git branch after passing auto and manual tests
- Put the mechanics on the helicopters

Engineering For Responsiveness

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC
  - In-house design based on Netty and Google PBs

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC
  - In-house design based on Netty and Google PBs
  - Supports Sync, Async clients, journaling of messages

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC
  - In-house design based on Netty and Google PBs
  - Supports Sync, Async clients, journaling of messages
- Latency tolerant message paths

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC
  - In-house design based on Netty and Google PBs
  - Supports Sync, Async clients, journaling of messages
- Latency tolerant message paths
  - Use beanstalkd for point-to-point messaging

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC
  - In-house design based on Netty and Google PBs
  - Supports Sync, Async clients, journaling of messages
- Latency tolerant message paths
  - Use beanstalkd for point-to-point messaging
  - Use Kafka for pub-sub messaging

Engineering For Responsiveness

- Low latency, high throughput message paths use RPC
  - In-house design based on Netty and Google PBs
  - Supports Sync, Async clients, journaling of messages
- Latency tolerant message paths
  - Use beanstalkd for point-to-point messaging
  - Use Kafka for pub-sub messaging
  - Generally favor pub-sub - easier to tack on listeners later - dovetails well into system-wide CEP

Engineering For Availability

Engineering For Availability

- The Dark Launch - Roll out a functionality to a subset of customers

Engineering For Availability

- The Dark Launch - Roll out a functionality to a subset of customers
- Take a new service in or out of production with no customer impact

Engineering For Availability

- The Dark Launch - Roll out a functionality to a subset of customers
- Take a new service in or out of production with no customer impact
  - Double writes, single reads, migration, cutover

Engineering For Availability

- The Dark Launch - Roll out a functionality to a subset of customers
- Take a new service in or out of production with no customer impact
  - Double writes, single reads, migration, cutover
- Load balanced HTTP with blended traffic to new and old service

Engineering For Availability

- The Dark Launch - Roll out a functionality to a subset of customers
- Take a new service in or out of production with no customer impact
  - Double writes, single reads, migration, cutover
  - Load balanced HTTP with blended traffic to new and old service
- Service abstraction helps immensely

Engineering For Availability

- The Dark Launch - Roll out a functionality to a subset of customers
- Take a new service in or out of production with no customer impact
  - Double writes, single reads, migration, cutover
  - Load balanced HTTP with blended traffic to new and old service
- Service abstraction helps immensely
- Requires extra discipline and careful thinking about how multiple versions of a thing coexist

Eng. For Continuous Improvement

Eng. For Continuous Improvement

- Lean's 5 Whys when we have a customer-facing incident

Eng. For Continuous Improvement

- Lean's 5 Whys when we have a customer-facing incident
  - Ask "Why" five times in an effort to get to a root cause

Eng. For Continuous Improvement

- Lean's 5 Whys when we have a customer-facing incident
  - Ask "Why" five times in an effort to get to a root cause
  - Make an Engineering or Ops investment proportional to the incident

Eng. For Continuous Improvement

- Lean's 5 Whys when we have a customer-facing incident
  - Ask "Why" five times in an effort to get to a root cause
  - Make an Engineering or Ops investment proportional to the incident
- Almost always results in improved monitoring

Eng. For Continuous Improvement

- Lean's 5 Whys when we have a customer-facing incident
  - Ask "Why" five times in an effort to get to a root cause
  - Make an Engineering or Ops investment proportional to the incident
- Almost always results in improved monitoring
- Often improved testing and code quality

Eng. For Continuous Improvement

- Lean's 5 Whys when we have a customer-facing incident
  - Ask "Why" five times in an effort to get to a root cause
  - Make an Engineering or Ops investment proportional to the incident
- Almost always results in improved monitoring
- Often improved testing and code quality
- Sometimes entire new services

Eng. For Continuous Improvement

Eng. For Continuous Improvement

Example 5 Whys - 13-JUN-2011 Postgres Slave Delay

Eng. For Continuous Improvement

Example 5 Whys - 13-JUN-2011 Postgres Slave Delay

1. **Why did we have a delay in replication?** There was an open connection in "idle in transaction" state for three days.

2. **Why was there an idle transaction?** There was a reports ETL task loading all the data, and it was somewhat slow. While it did work in batches, it did not commit or roll back.

3. **Why didn't we know about the delay in replication?** The slave replication monitors were misconfigured and not reading the correct data.

4. **Why were the replication monitors misconfigured?** After the deploy into production they were not properly reset to the new hostnames.

5. **Why did an idle transaction block replication?** Developer misunderstanding of how query result paging worked relative to PostgreSQL transactions.

Operations at UA

Operations at UA

- Team of 5 operating > 100 servers

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal
  - EC2 for surge capacity

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal
  - EC2 for surge capacity
- We have no “DevOps”

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal
  - EC2 for surge capacity
- We have no “DevOps”
  - Everybody does DevOps - Engineers, Support, Ops

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal
  - EC2 for surge capacity
- We have no “DevOps”
  - Everybody does DevOps - Engineers, Support, Ops
  - But not everybody does Ops

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal
  - EC2 for surge capacity
- We have no “DevOps”
  - Everybody does DevOps - Engineers, Support, Ops
  - But not everybody does Ops
- Responsible for monitoring, alerting, automation tools, security, SOP

Operations at UA

- Team of 5 operating > 100 servers
  - Mostly bare metal
  - EC2 for surge capacity
- We have no “DevOps”
  - Everybody does DevOps - Engineers, Support, Ops
  - But not everybody does Ops
- Responsible for monitoring, alerting, automation tools, security, SOP
- First line of defense in on call rotation

Operations For Transparency

Operations For Transparency

- Measure **everything**, monitor important things

Operations For Transparency

- Measure **everything**, monitor important things
  - Every service exposes internal and external latency

Operations For Transparency

- Measure **everything**, monitor important things
  - Every service exposes internal and external latency
  - Java agent exposes JMX via HTTP/JSON

Operations For Transparency

- Measure **everything**, monitor important things
  - Every service exposes internal and external latency
  - Java agent exposes JMX via HTTP/JSON
  - Python via custom “socket console”

Operations For Transparency

- Measure **everything**, monitor important things
  - Every service exposes internal and external latency
  - Java agent exposes JMX via HTTP/JSON
  - Python via custom “socket console”
- DRY - an operator needs to be L1 & L2 support - consistency is key

Operations For Transparency

- Measure **everything**, monitor important things
  - Every service exposes internal and external latency
  - Java agent exposes JMX via HTTP/JSON
  - Python via custom “socket console”
- DRY - an operator needs to be L1 & L2 support - consistency is key
- This has us to over 1200 services across > 100 servers and we’re still somewhat sane

Ops. Continuous Improvement

Ops. Continuous Improvement

- Company bootstrapped in EC2

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2
  - Erratic network latency

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2
  - Erratic network latency
  - Wide spread outages - EBS and ELB

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2
  - Erratic network latency
  - Wide spread outages - EBS and ELB
  - Strange network behavior at scale

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2
  - Erratic network latency
  - Wide spread outages - EBS and ELB
  - Strange network behavior at scale
  - Database scale getting costly due to poor I/O

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2
  - Erratic network latency
  - Wide spread outages - EBS and ELB
  - Strange network behavior at scale
  - Database scale getting costly due to poor I/O
  - Kernel + Hypervisor issues

Ops. Continuous Improvement

- Company bootstrapped in EC2
- By Q1/Q2 2011- substantial growing pains with EC2
  - Erratic network latency
  - Wide spread outages - EBS and ELB
  - Strange network behavior at scale
  - Database scale getting costly due to poor I/O
  - Kernel + Hypervisor issues
  - Undocumented limitations - Network, VPC, ELB

Ops. Continuous Improvement

Ops. Continuous Improvement

- Step back, revisit the tradeoffs of being a cloud company

Ops. Continuous Improvement

- Step back, revisit the tradeoffs of being a cloud company
- Research alternatives - initially MSPs, no co-lo

Ops. Continuous Improvement

- Step back, revisit the tradeoffs of being a cloud company
- Research alternatives - initially MSPs, no co-lo
- Decide to go bare metal for cost, performance

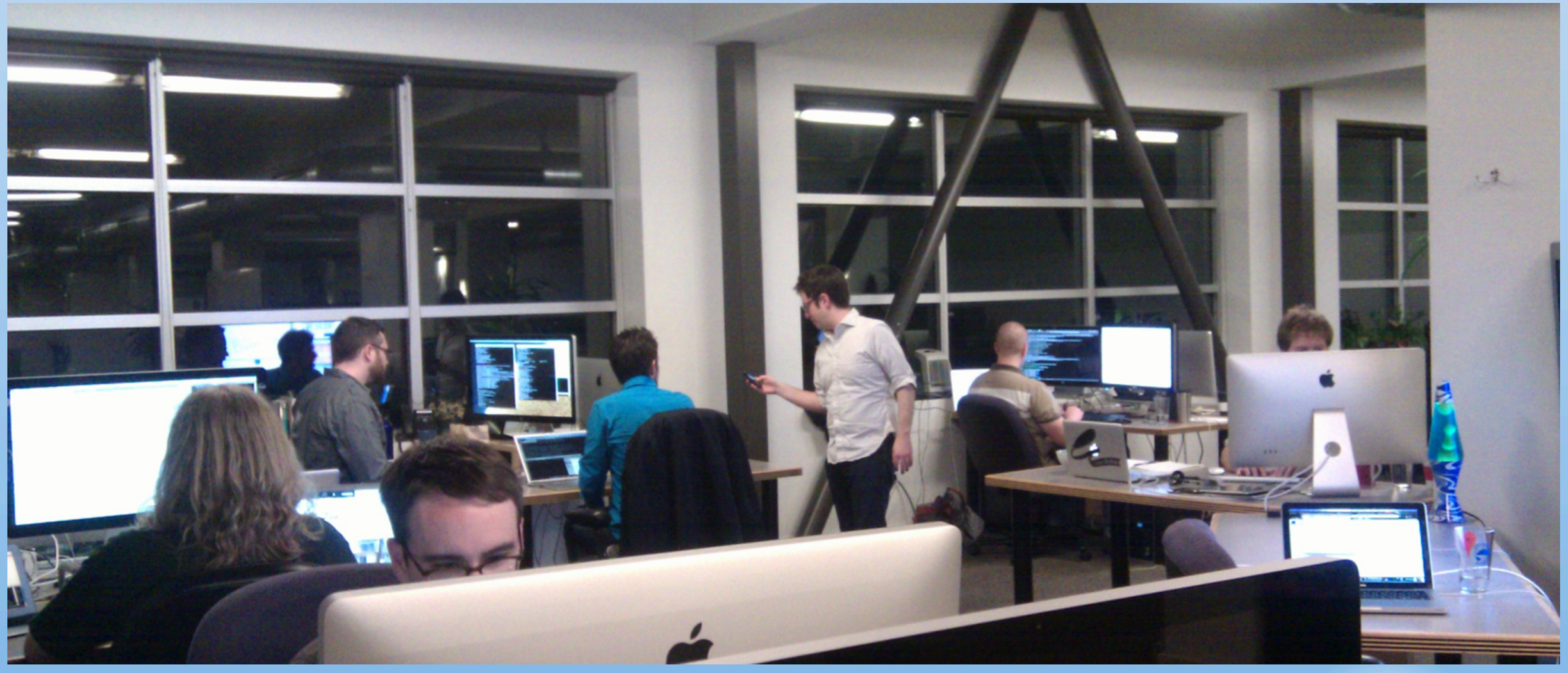
Ops. Continuous Improvement

- Step back, revisit the tradeoffs of being a cloud company
- Research alternatives - initially MSPs, no co-lo
- Decide to go bare metal for cost, performance
- One month of planning, one month of testing

Ops. Continuous Improvement

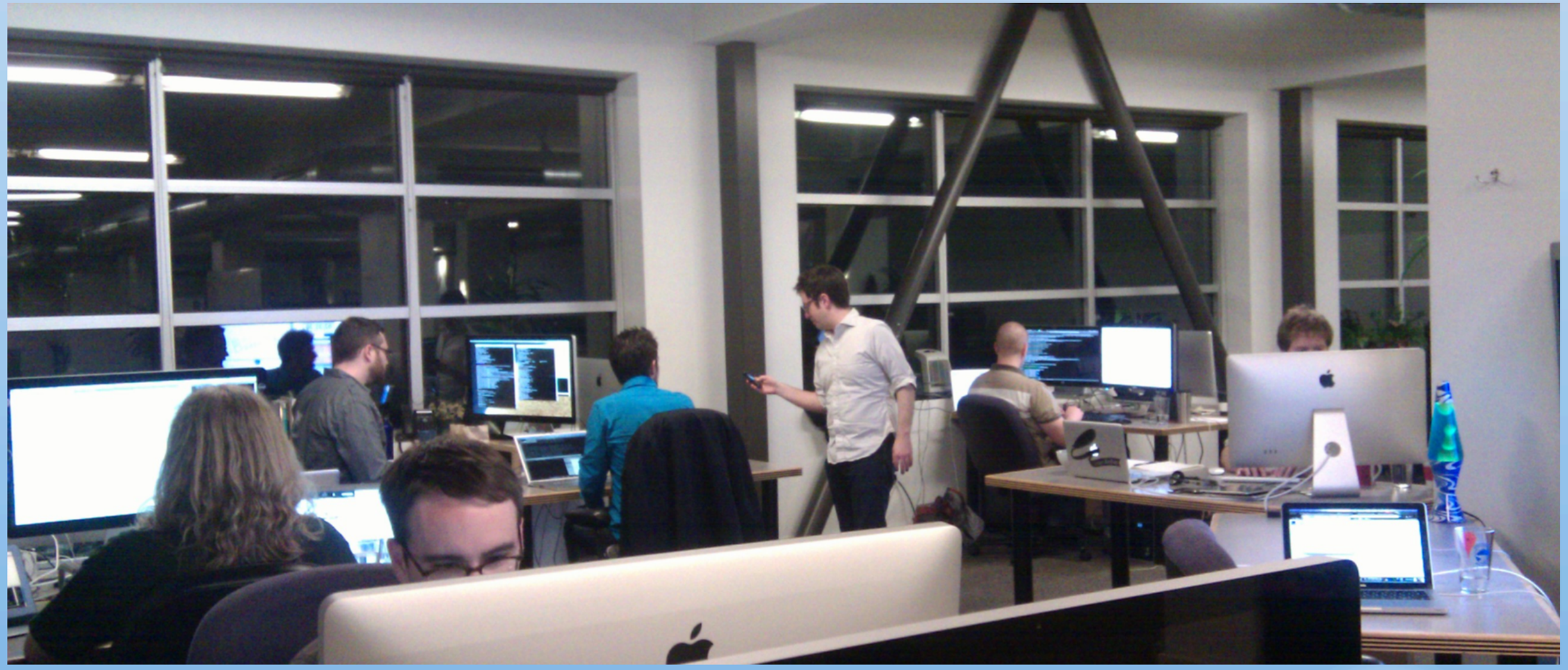
- Step back, revisit the tradeoffs of being a cloud company
- Research alternatives - initially MSPs, no co-lo
- Decide to go bare metal for cost, performance
- One month of planning, one month of testing
- Keep the same automation approach across both environments during the transition

Ops. Continuous Improvement



Ops. Continuous Improvement

One night in Portland...



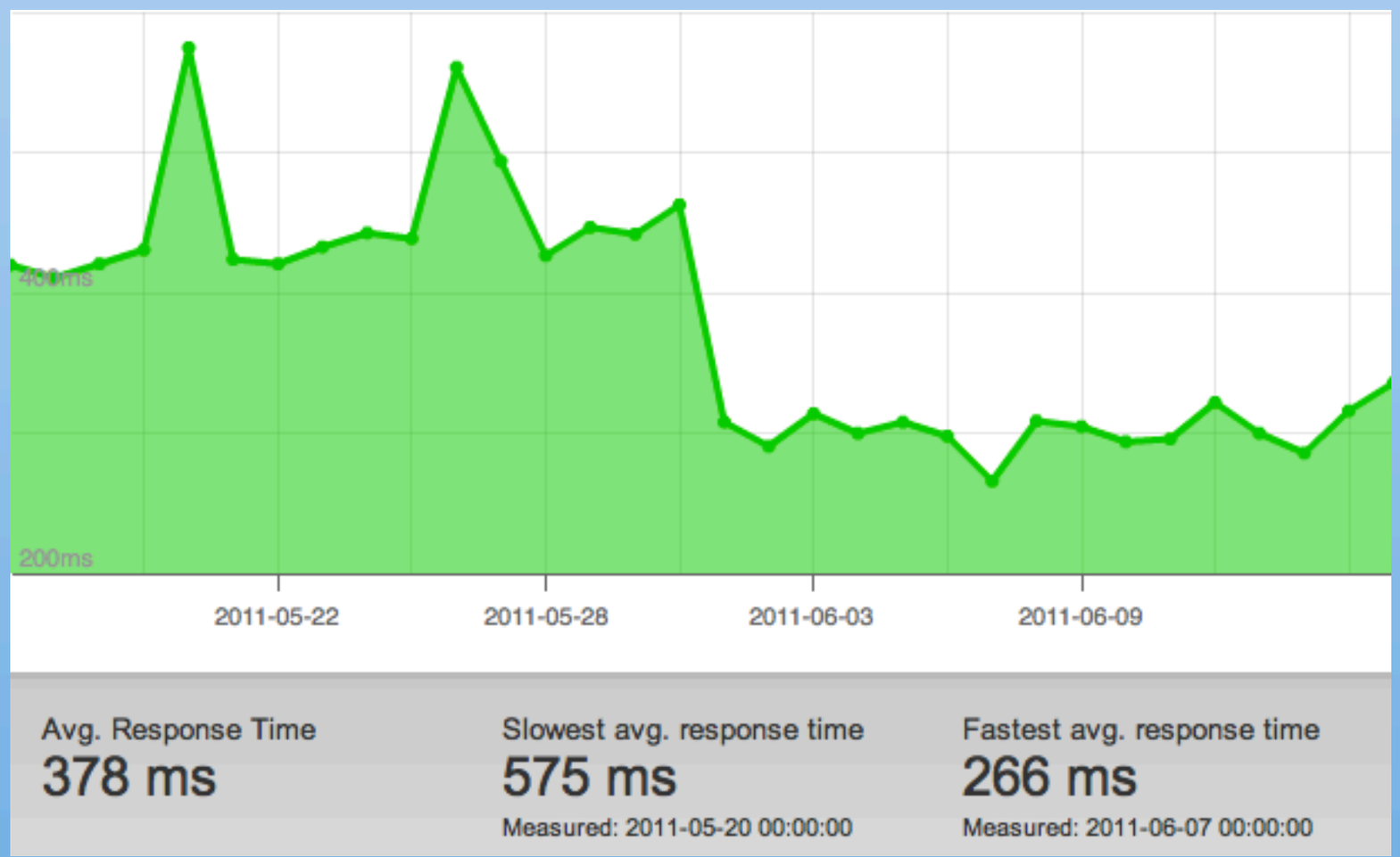
Ops. Continuous Improvement

Ops. Continuous Improvement

Resulted in...

Ops. Continuous Improvement

Resulted in...



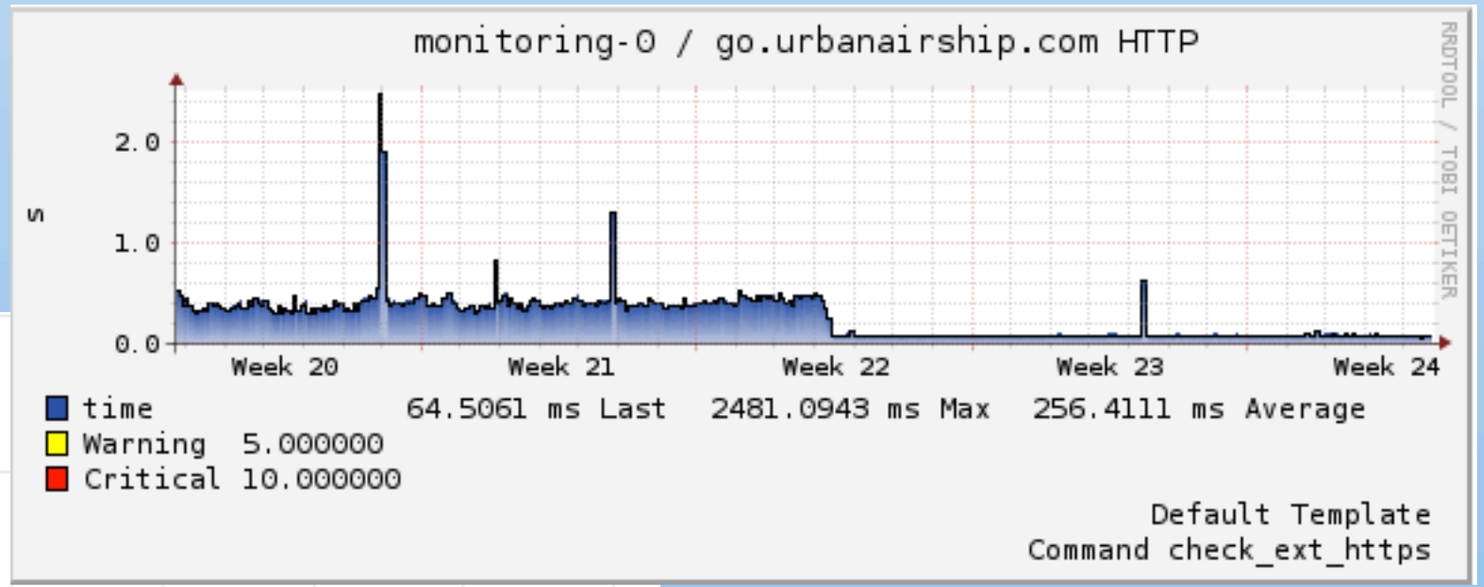
Avg. Response Time
378 ms

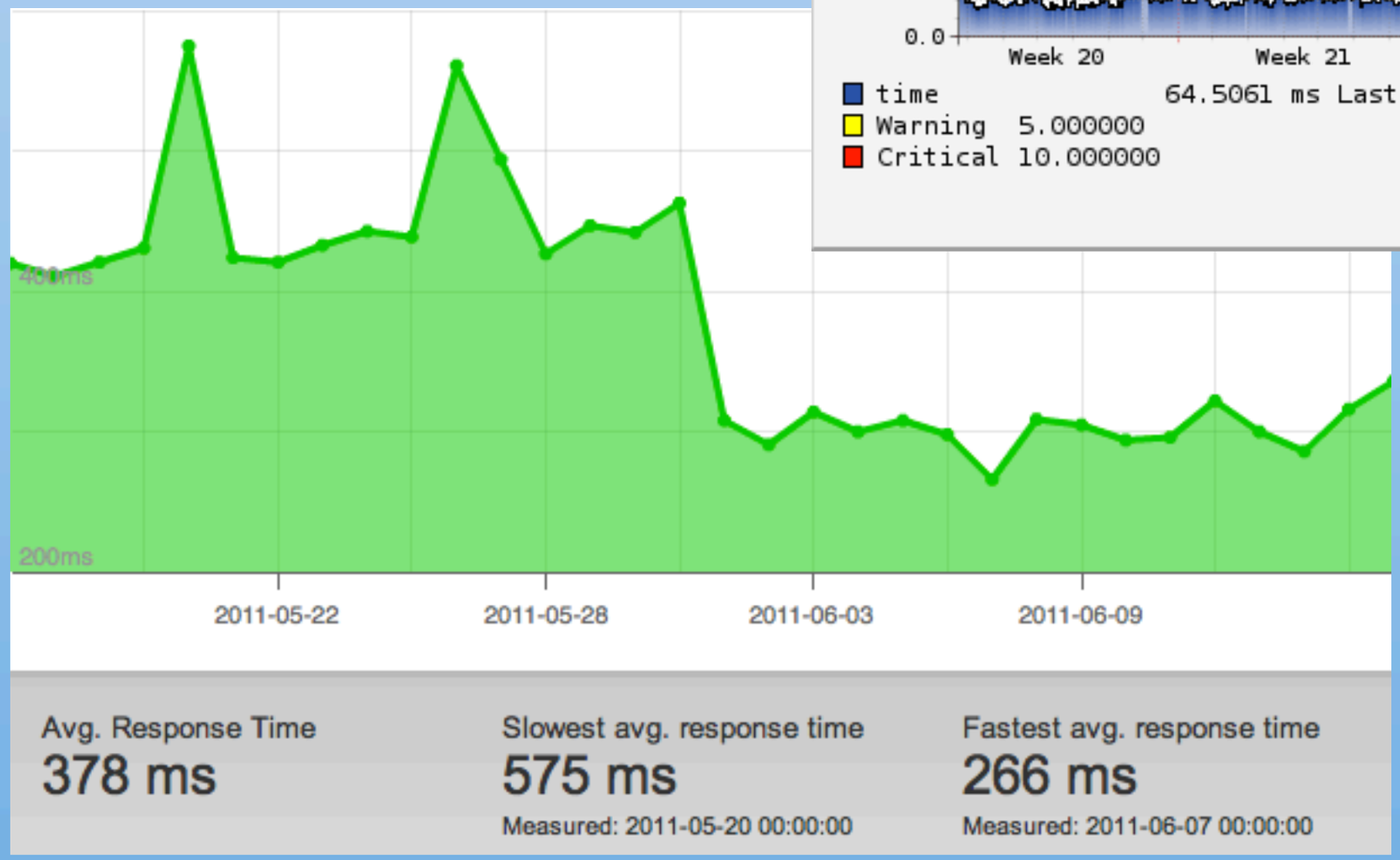
Slowest avg. response time
575 ms
Measured: 2011-05-20 00:00:00

Fastest avg. response time
266 ms
Measured: 2011-06-07 00:00:00

Ops. Continuous Improvement

Resulted in...





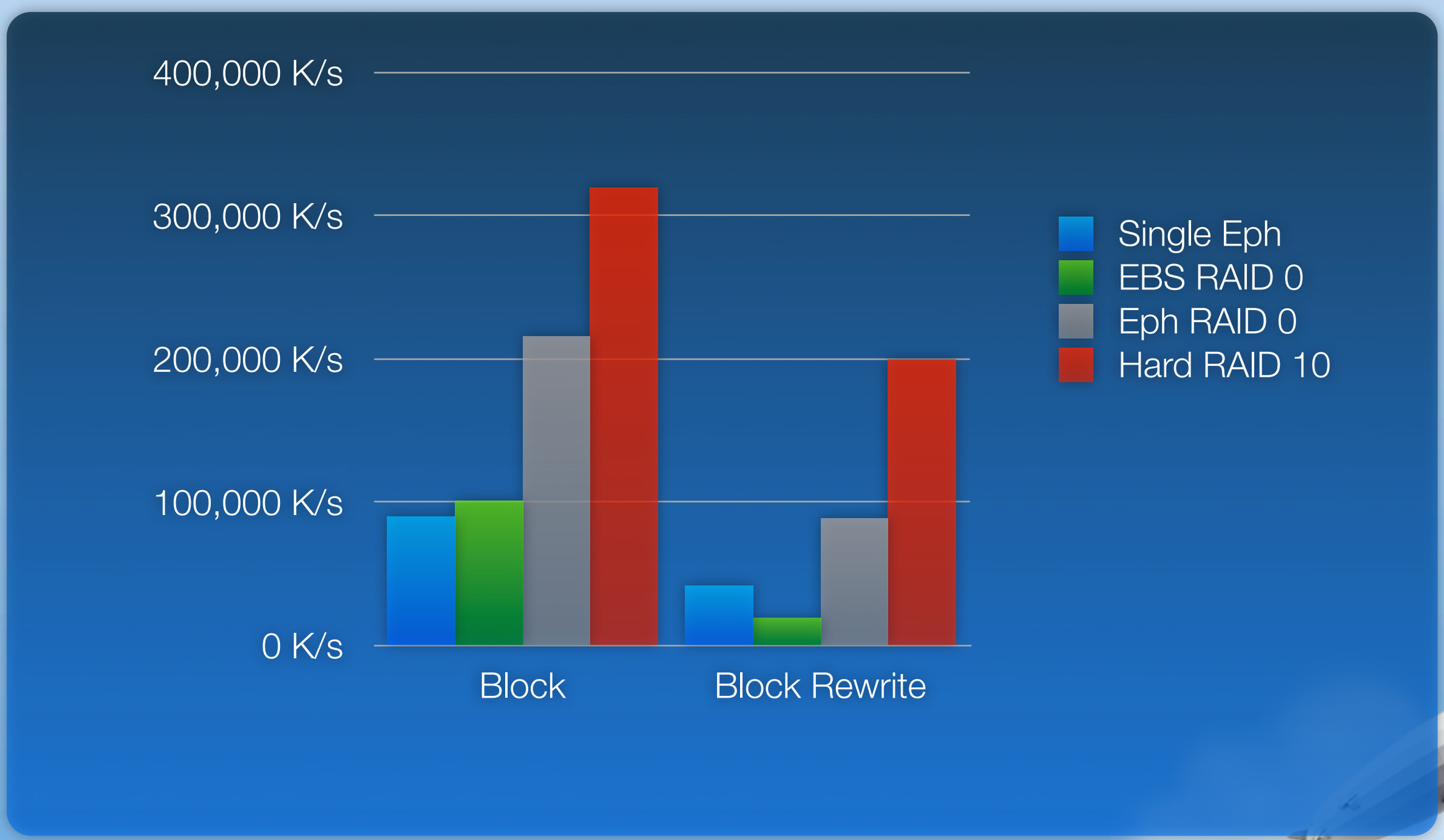
Avg. Response Time
378 ms

Slowest avg. response time
575 ms
Measured: 2011-05-20 00:00:00

Fastest avg. response time
266 ms
Measured: 2011-06-07 00:00:00

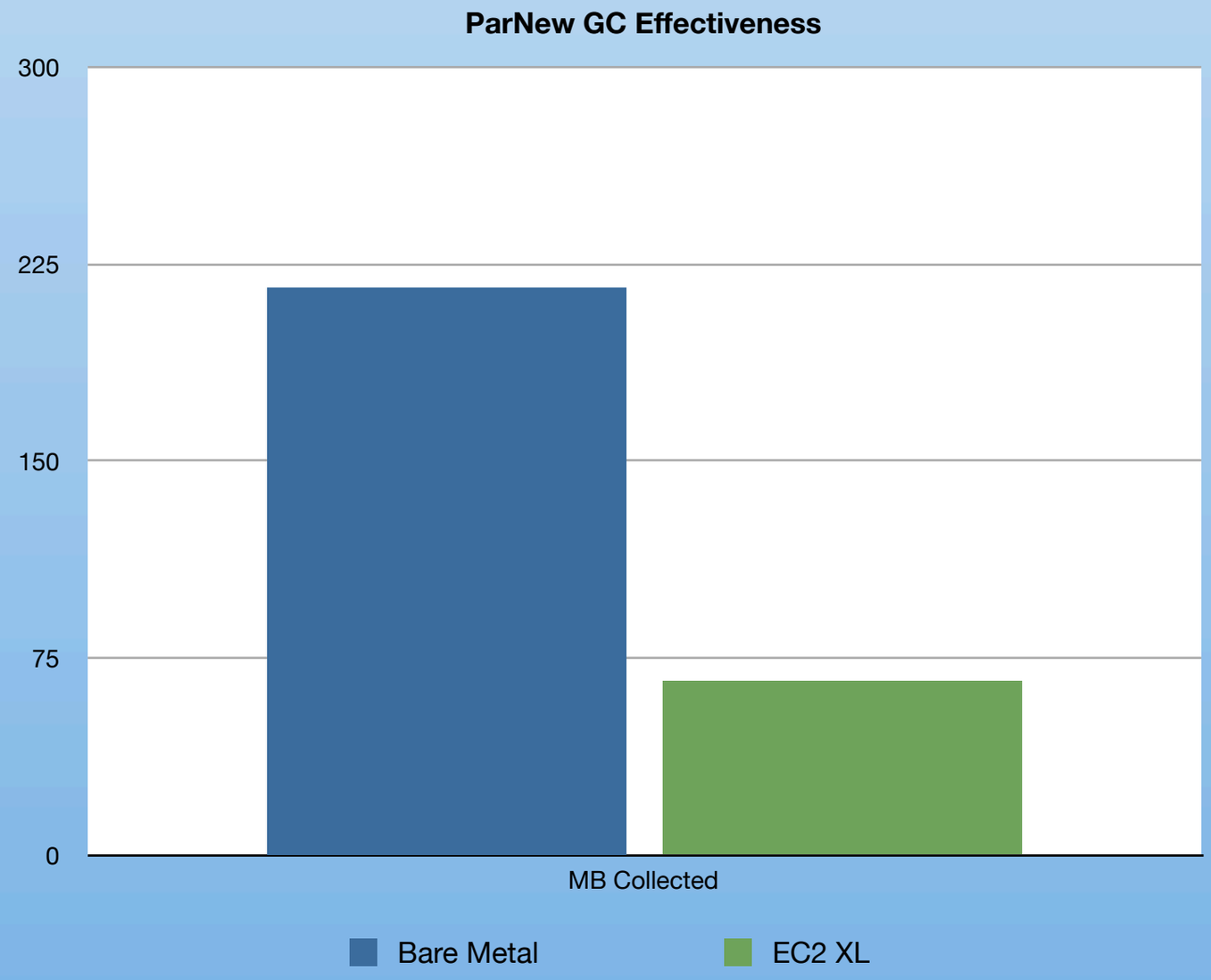
Ops. Continuous Improvement

Sequential Block Writes



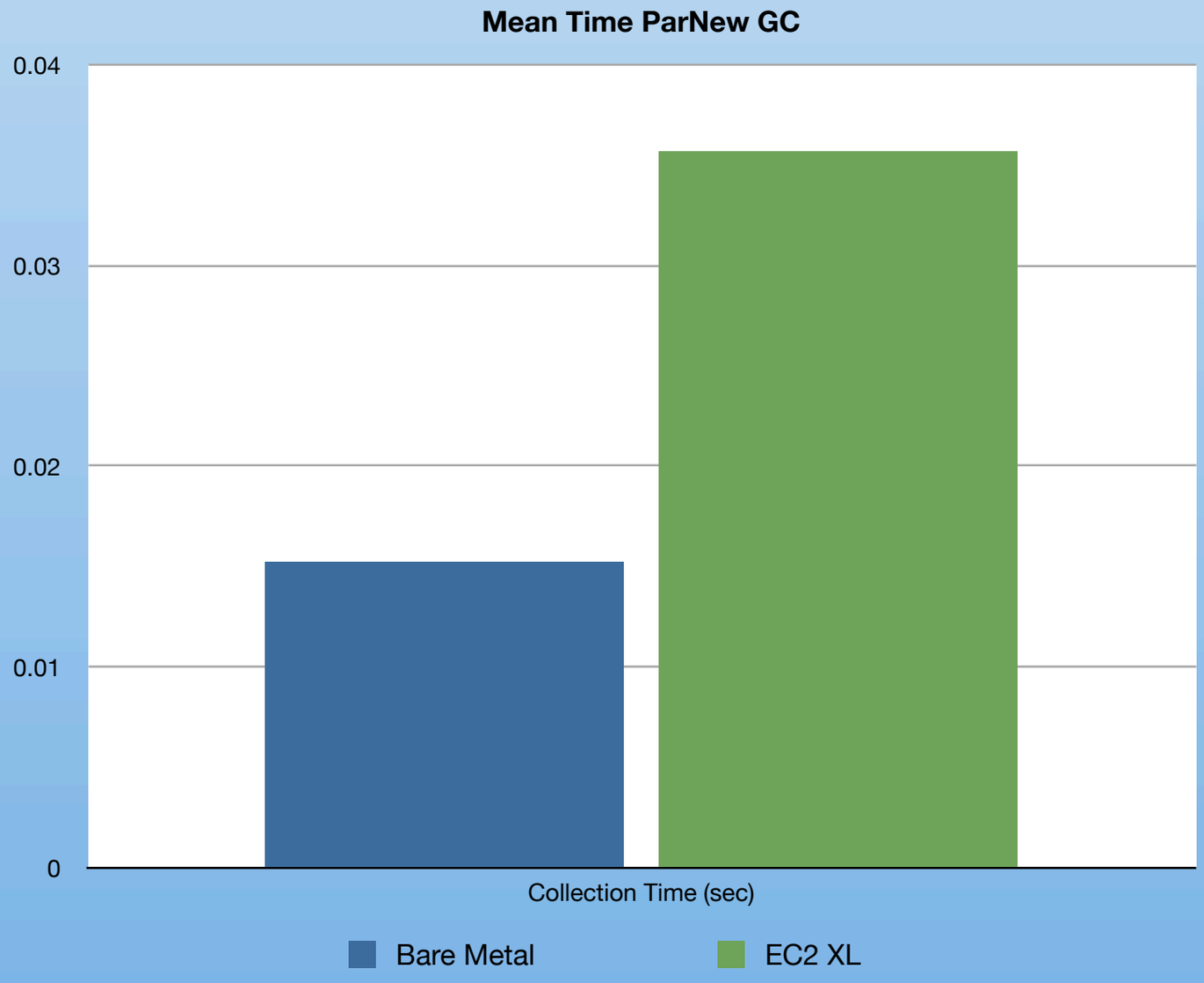
Ops. Continuous Improvement

Performance of Real CPUs - GC Effectiveness



Ops. Continuous Improvement

Performance of Real CPUs - GC Performance



Operations For Scale

Operations For Scale

- Our capacity scale needs tend to come in fits and spurts

Operations For Scale

- Our capacity scale needs tend to come in fits and spurts
  - Successful launches of new apps

Operations For Scale

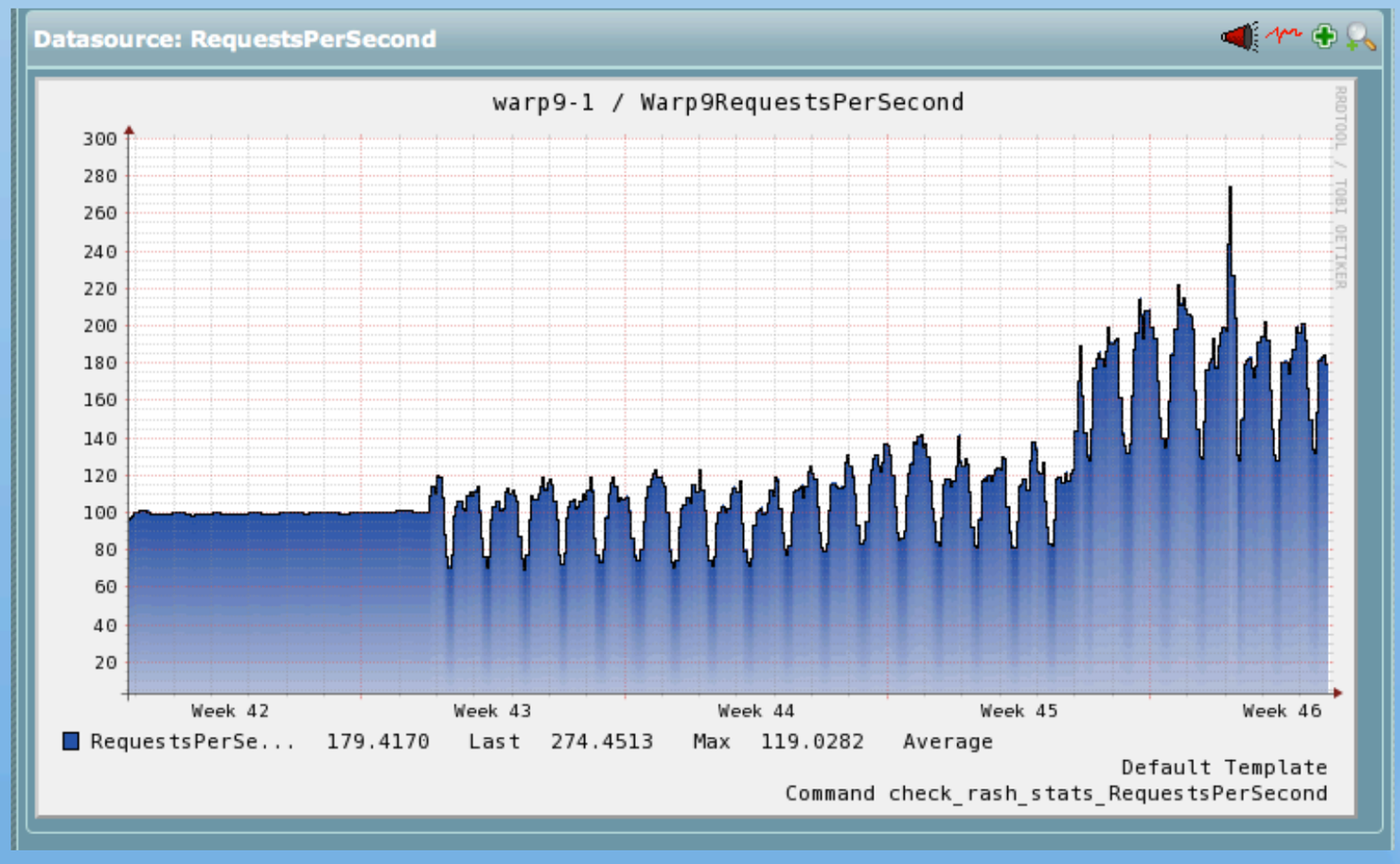
- Our capacity scale needs tend to come in fits and spurts
  - Successful launches of new apps
  - Unexpected viral apps

Operations For Scale

- Our capacity scale needs tend to come in fits and spurts
  - Successful launches of new apps
  - Unexpected viral apps
  - Unexpected herd effects

Operations For Scale

- Our capacity scale needs tend to come in fits and spurts
  - Successful launches of new apps
  - Unexpected viral apps
  - Unexpected herd effects



Operations For Scale

Operations For Scale

- Worked with our MSP and Amazon towards an “Hybrid Cloud”

Operations For Scale

- Worked with our MSP and Amazon towards an “Hybrid Cloud”
- Critical systems on bare metal

Operations For Scale

- Worked with our MSP and Amazon towards an “Hybrid Cloud”
- Critical systems on bare metal
- Surge capacity on Amazon VPC - EC2 nodes on demand, when we need them

Operations For Scale

- Worked with our MSP and Amazon towards an “Hybrid Cloud”
- Critical systems on bare metal
- Surge capacity on Amazon VPC - EC2 nodes on demand, when we need them
- Best of both worlds

Operations For Scale

- Worked with our MSP and Amazon towards an “Hybrid Cloud”
- Critical systems on bare metal
- Surge capacity on Amazon VPC - EC2 nodes on demand, when we need them
- Best of both worlds
  - Databases are happy

Operations For Scale

- Worked with our MSP and Amazon towards an “Hybrid Cloud”
- Critical systems on bare metal
- Surge capacity on Amazon VPC - EC2 nodes on demand, when we need them
- Best of both worlds
  - Databases are happy
  - Extra CPU when we need it - currently using it for SSL offload

Operations For Scale

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data
- In September, some indexers started falling behind

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data
- In September, some indexers started falling behind
  - No new HBase machines for several weeks

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data
- In September, some indexers started falling behind
  - No new HBase machines for several weeks
  - Spin up 10 new HBase Region nodes in EC2

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data
- In September, some indexers started falling behind
  - No new HBase machines for several weeks
  - Spin up 10 new HBase Region nodes in EC2
- Aggregate throughput actually went down

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data
- In September, some indexers started falling behind
  - No new HBase machines for several weeks
    - Spin up 10 new HBase Region nodes in EC2
  - Aggregate throughput actually went down
    - 400Mb network != 1Gb network

Operations For Scale

- Hybrid cloud is not always a panacea - the trials of Big Data
- In September, some indexers started falling behind
  - No new HBase machines for several weeks
    - Spin up 10 new HBase Region nodes in EC2
  - Aggregate throughput actually went down
    - 400Mb network != 1Gb network
    - EC2 I/O was so slow, it became the limiting factor

Room For Improvement

Room For Improvement

- Simulating scale for testing is hard, representative load difficult to achieve

Room For Improvement

- Simulating scale for testing is hard, representative load difficult to achieve
- Automation improvements never end - you can never have enough but it's difficult to make progress in an interrupt driven environment

Room For Improvement

- Simulating scale for testing is hard, representative load difficult to achieve
- Automation improvements never end - you can never have enough but it's difficult to make progress in an interrupt driven environment
- Having lots of services can make integration testing interesting

Room For Improvement

- Simulating scale for testing is hard, representative load difficult to achieve
- Automation improvements never end - you can never have enough but it's difficult to make progress in an interrupt driven environment
- Having lots of services can make integration testing interesting
- Balancing keeping engineers interested and need for specialization

Key Mobile Learnings

Key Mobile Learnings

- Device connectivity is extremely unreliable

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data
- No visibility into iOS kernel

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data
- No visibility into iOS kernel
- Devices can be offline for a long time buffering data

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data
- No visibility into iOS kernel
- Devices can be offline for a long time buffering data
- Use devices as micro storage

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data
- No visibility into iOS kernel
- Devices can be offline for a long time buffering data
- Use devices as micro storage
- Client developers often don’t listen to us which can hurt

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data
- No visibility into iOS kernel
- Devices can be offline for a long time buffering data
- Use devices as micro storage
- Client developers often don’t listen to us which can hurt
- Customers often under estimate the size of their audience and the effectiveness of push notifications

Key Mobile Learnings

- Device connectivity is extremely unreliable
- Carriers incentive is to conserve data
- No visibility into iOS kernel
- Devices can be offline for a long time buffering data
- Use devices as micro storage
- Client developers often don’t listen to us which can hurt
- Customers often under estimate the size of their audience and the effectiveness of push notifications
- Mobile is like browsers but worse

Thanks!

- Urban Airship http://urbanairship.com/
- Me @eonnen or erik at
- We're hiring! http://urbanairship.com/company/jobs/