ARCHITECTURE AT SIMPLEGEO

STAYING AGILE AT SCALE

featuring Mike Malone

SimpleGeo

ABOUT ME

MIKE MALONE
LEAD ARCHITECT

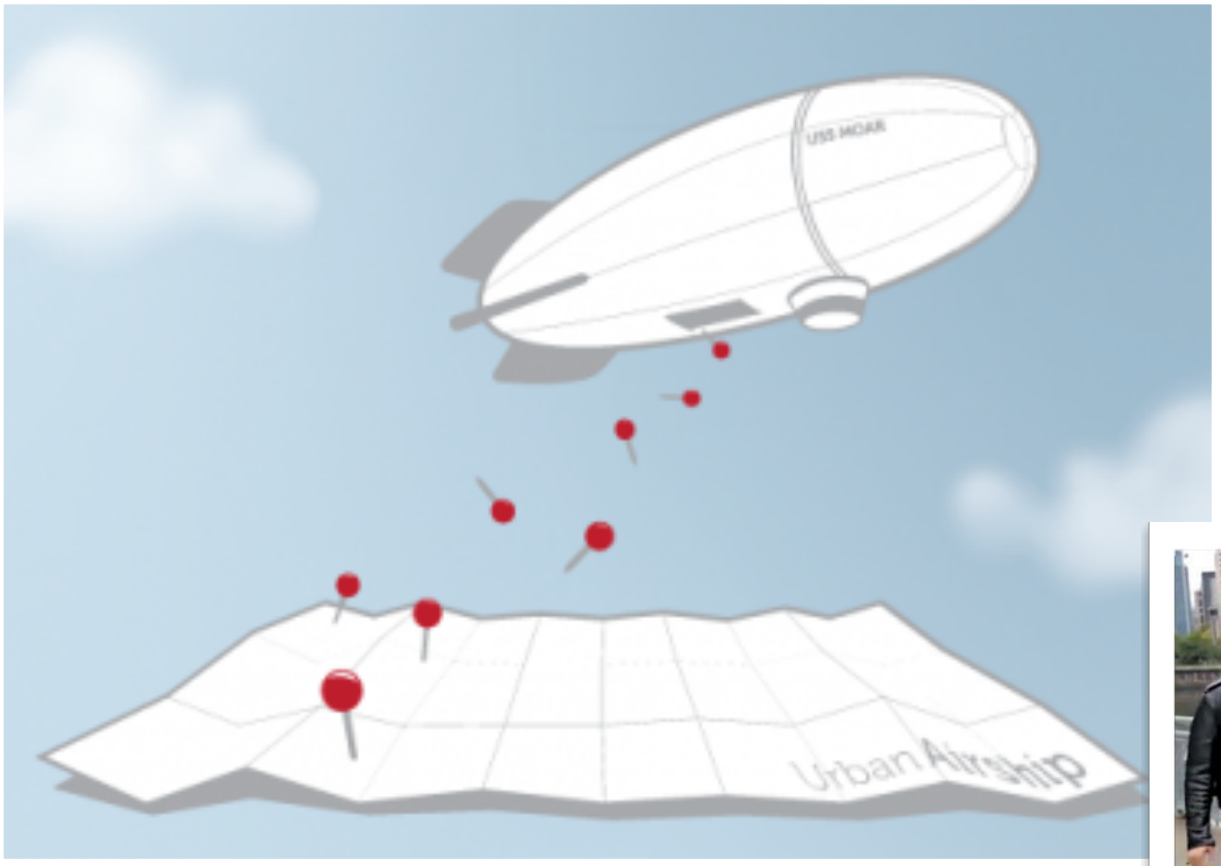
mike@simplegeo.com
@mjmalone

For the last two years I've been helping develop and operate the underlying technologies that power SimpleGeo's location-based services & GIS platform.

SimpleGeo

SG STORAGE SG CONTEXT SG PLACES

I, FOR ONE, WELCOME OUR NEW PORTLANDIAN OVERLORDS



Everything I know about Portland

IN THE NEXT HOUR

ARCHITECTURE

- What is architecture?
- What is good architecture?

SOFTWARE ARCHITECTURE

- What is software architecture?
- What is good software architecture?

Briefly...

SIMPLEGEO'S ARCHITECTURE

- What environmental constraints helped define SimpleGeo's architecture?
- What tenets helped shape SimpleGeo's architecture?
  - What is the rationale for the tenet? How is it consistent with the core architectural philosophy?
  - What is an example of an architectural choice that exemplifies this tenet?

ARCHITECTURE
WHAT DOES IT EVEN MEAN?

SOME ARCHITECTURE?

The **process** and **product** of planning, design, and construction

PART PHILOSOPHICAL

- Based on some value system
- Balances technical, social, environmental, and aesthetic considerations

PART PRAGMATIC

- Scheduling
- Cost estimation
- Construction administration

ARCHITECTURE
WHAT MAKES IT GOOD?

Objective criteria can be used to judge building quality

GOOD ARCHITECTURE IS

- Fit for its intended use
- Cost effective (within budget)
- Environmentally friendly

INTRINSIC VS. INSTRUMENTAL

- **Intrinsic** good in-and-of itself
- **Instrumental** a means to other goods

RYUGYONG HOTEL

ARCHITECTURE
WHAT MAKES IT GOOD?

However, a great deal of subjectivity is also involved

GOOD ARCHITECTURE IS

- Aesthetically pleasing
- Culturally relevant
- A positive influence on its environment

NO ACCOUNTING FOR TASTE

- Hard characteristics to define and measure
- But they're critically important!

FLICKR.COM/PHOTOS/STUCKINCUSTOMS/210118173/

REASONABLE MINDS CAN DIFFER

EXPERIENCE MUSIC PROJECT

FLICKR.COM/PHOTOS/DROCPSU/411079775/

FACT: PROGRAMS AREN'T BUILDINGS

The assertion that software development is like construction is a contentious one...

SOFTWARE DEVELOPMENT VS. CONSTRUCTION

- Incremental vs. Structured & Sequential
- Adaptable vs. Fixed
- Cheap vs. Expensive
- Intangible vs. Tangible

BETTER METAPHOR? - LANGUAGE & WRITING

- But that doesn't mean the construction metaphor is meaningless
- The construction metaphor provides a language that we can use to communicate ideas in software development

SOFTWARE ARCHITECTURE
WHAT DOES IT EVEN MEAN?

The high-level decisions that constrain a system such that its fundamental structure emerges

SOFTWARE ARCHITECTURE IS...

- The high-level breakdown of a system into its parts [FOWLER]
- The set of significant decisions about the organization of a software system [KRUCHTEN, BOOCH, BITTNER, REITMAN]
- The externally visible properties of software elements, and the relationships among them (no internal implementation details) [BASS, CLEMENTS, KAZMAN]
- Whatever the important stuff is [FOWLER]

LOOSELY DEFINED

SOFTWARE ARCHITECTURE
WHAT MAKES IT GOOD?

GOOD ARCHITECTURE IS

- **Robust** lacking bugs and tolerant of faults
- **Maintainable** easy to maintain and extend
- **Useful** utility, beyond the immediate need
- **Scalable** able to grow in capacity
- **Common vision** direction, strategy
- **Agile** simple enough to refactor easily
- **Extensible** able to grow in features
- **Responsive** performant now and after expanding or scaling

WARD'S WIKI <C2.COM/CGI/WIKI?GOODARCHITECTURE>

BEASTIE

SOFTWARE ARCHITECTURE
WHAT MAKES IT GOOD?

FOUNDATION OF QUALITY

- Based on some value system
- Balances technical, social, environmental, and aesthetic considerations

KEEP ASKING "WHY?"

- Provides perspective - what's the point of all of this?
- Promotes "outside the box" thinking
  - Sometimes technical problems have a non-technical solution (and vice-versa)

PLAY-DOH PLATO

<MAYBELOGIC.ORG/MAYBEQUARTERLY/02/0209PLATOPLAY-DOH.HTM>

BEING AN ARCHITECT
WHAT DOES IT EVEN MEAN?

ARCHITECTS ARE GENERALISTS

- Architects must see **the trees** without losing site of **the forest**
- Architects shouldn't work from an **ivory tower**, they should have deep knowledge of systems they work with
- Architects are responsible for **creating an environment** that produces high quality software

ARCHITECTS BALANCE CONFLICTING GOALS

- Architects must communicate with stakeholders to advance their architectural vision
- Architects must understand the broader environment in which their system exists

SIMPLEGEO'S ARCHITECTURE

SimpleGeo's architecture works well for a location-based services platform... but that's not what you do.

VERNACULAR ARCHITECTURE

- Methods of construction that use locally available resources and traditions to address local needs and circumstances
  - Evolves over time to reflect the environmental, cultural, and historical context in which it exists
  - Often dismissed as crude and unrefined
  - Based largely on knowledge achieved through trial and error and handed down through the generations, transmitted by local traditions and culture (e.g., blogs, wikis, conferences)
- Most of the rest of this presentation will be "vernacular architecture"

THE ENVIRONMENT
EXTERNAL CONSTRAINTS ON OUR ARCHITECTURE

PLATFORM AS A SERVICE

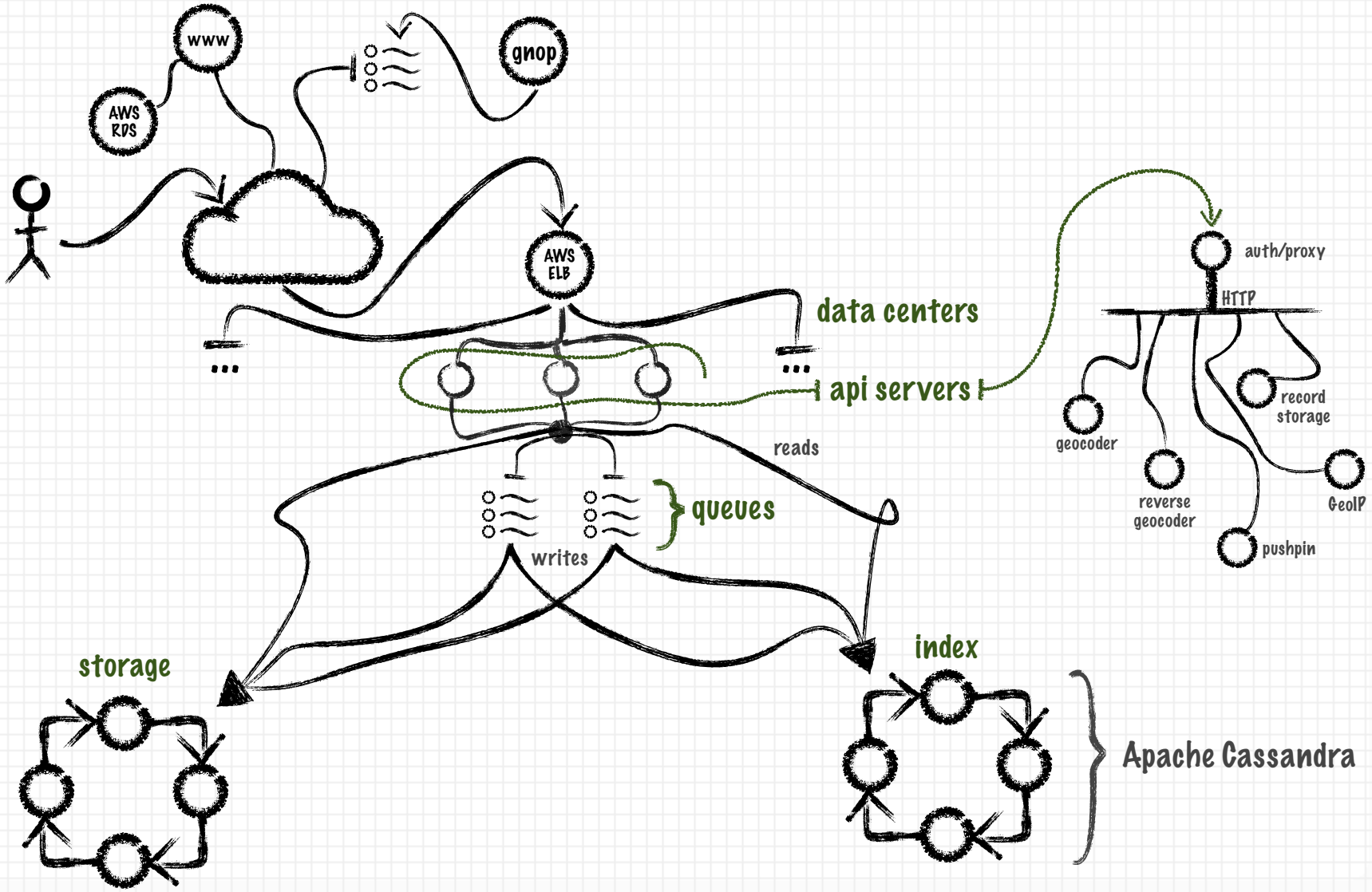
- Web Service API using HTTP for transport
- Massively multi-tenant for economic and operational reasons
- Accounting demands accurate metrics collection
- Low tolerance for outages - unlike consumer web apps, our customers tend to notice even minor outages
- Brand built on scalability and availability

LOCATION (GIS)

- Limited off-the-shelf technology that's usable at scale
- Computationally complex and highly specialized
- Data density has high variance

REAL-TIME

- Large operational workload with frequent updates
- Consumer applications demand low-latency request processing


www
AWS RDS
gnop
AWS ELB
data centers
api servers
auth/proxy
HTTP
record storage
geocoder
reverse geocoder
GeoIP
pushpin
reads
writes
queues
storage
index
Apache Cassandra

THINGS CHANGE
SHIT HAPPENS
DEAL WITH IT

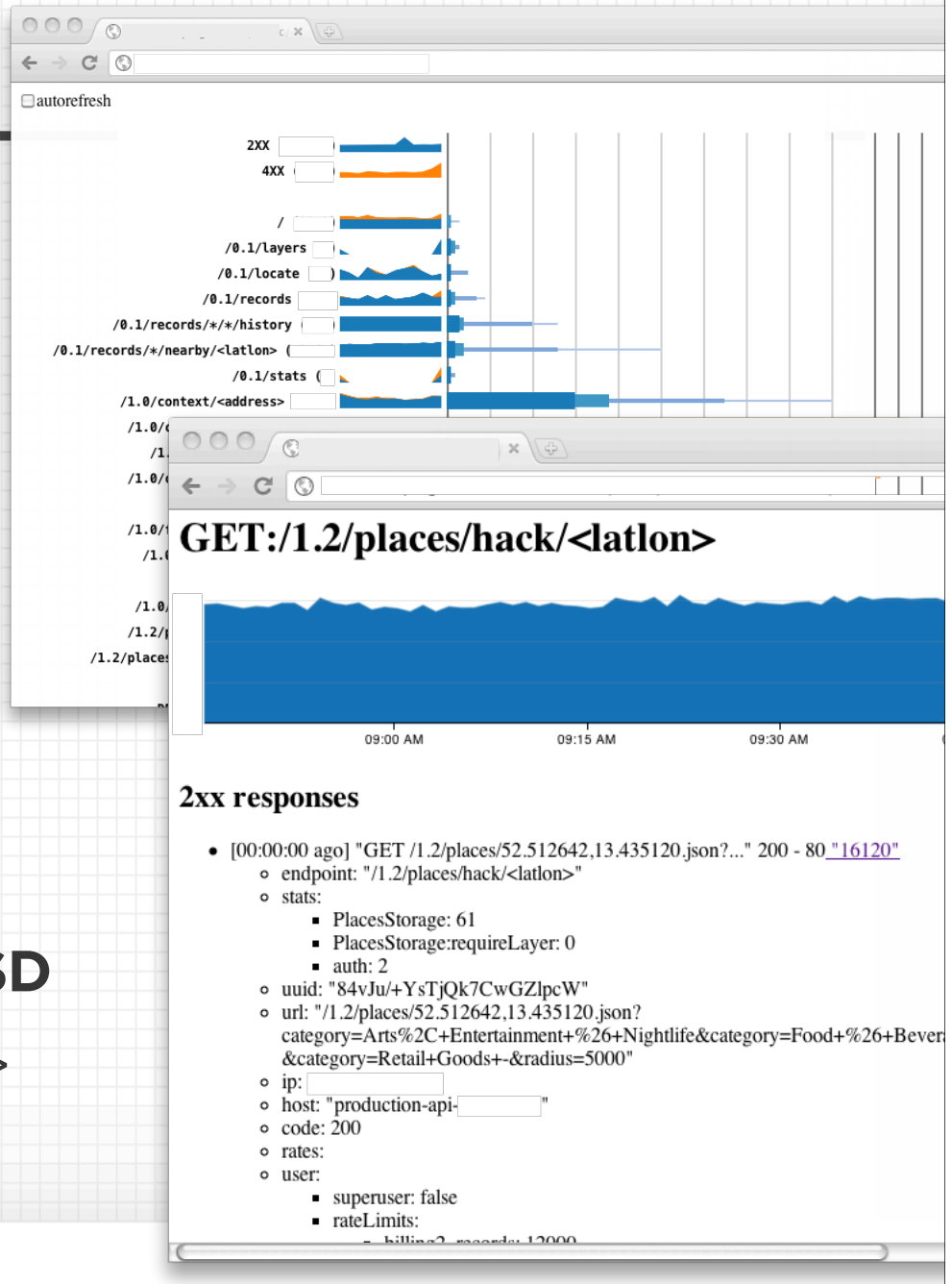
CHANGE HURTS

CHANGE IS RISKY, BUT NECESSARY

- The best we can do is reduce the risk of change
- Measure important stuff
  - Provide fast feedback - the value of information reduces quickly as it ages
  - Think ahead - good metrics and monitoring requires some foresight
    - Build monitoring hooks for key metrics throughout your systems
    - Be aware / beware of observer effect
- Invest in a monitoring system that can scale with your system and org
  - It must be easy to integrate with different code-bases and technologies
  - It must be capable of handling high throughput
  - It must be reasonably real-time (e.g., writing to HDFS and running MR jobs probably isn't good enough)

STATSNY

- **HTTP** service written in **python**, backed by **redis**
- Receives data via TCP or UDP
- Handles **timers** and **counters** grouped by endpoint, method, and response code
  - Computes **average** and **variance**
- Records full request and response headers for lightweight problem diagnosis



SIMILAR TO ETSY'S STATSD

<github.com/etsy/statsd>

SimpleGeo

PUSHING NEW CODE AT SIMPLEGEO

1. PUSH TO GITHUB PENDING BRANCH

- Post-commit hook triggers **Jenkins** build and test
- Once complete, build success/failure notification is sent out via IRC
  - If tests failed, stop
  - If tests pass, push to master branch on GitHub (only Jenkins commits to master)

2. POST-COMMIT HOOK ON MASTER BRANCH TRIGGERS DEB PKG BUILD

- OS package management lets us track system dependencies in the package manifest (could also use configuration management tools for this)
- Packages are hosted in **repoman** - a RESTful service for managing Debian repositories

3A. PUPPET UPGRADES PRODUCTION AUTOMATICALLY FOR MOST SERVICES

3B. FOR CRITICAL SERVICES, PROMOTE PACKAGE

- Promotion simply adjusts config management to install a newer version of a package, and can be done via IRC
- Extra step for critical services that need special attention during upgrade (e.g., Gate, databases)

CLOSE THE LOOP ON PUSHES

THE BIG BOARD

THE BIG BOARD

- **At-a-glance summary** of API health and important business metrics
- Often the first indication of service health problems that aren't caught by monitoring
- Provides critical feedback immediately after code pushes

UPGRADING GATE PHASED ADOPTION

GATE IS THE ENTRY POINT INTO SIMPLEGEO

- Performs auth, communicates with statsny and our billing infrastructure, and routes to backend services
- If it's down, SimpleGeo is down (so it can't go down)
- Lightweight, high speed, shared-nothing service (in node.js)

PHASED ADOPTION

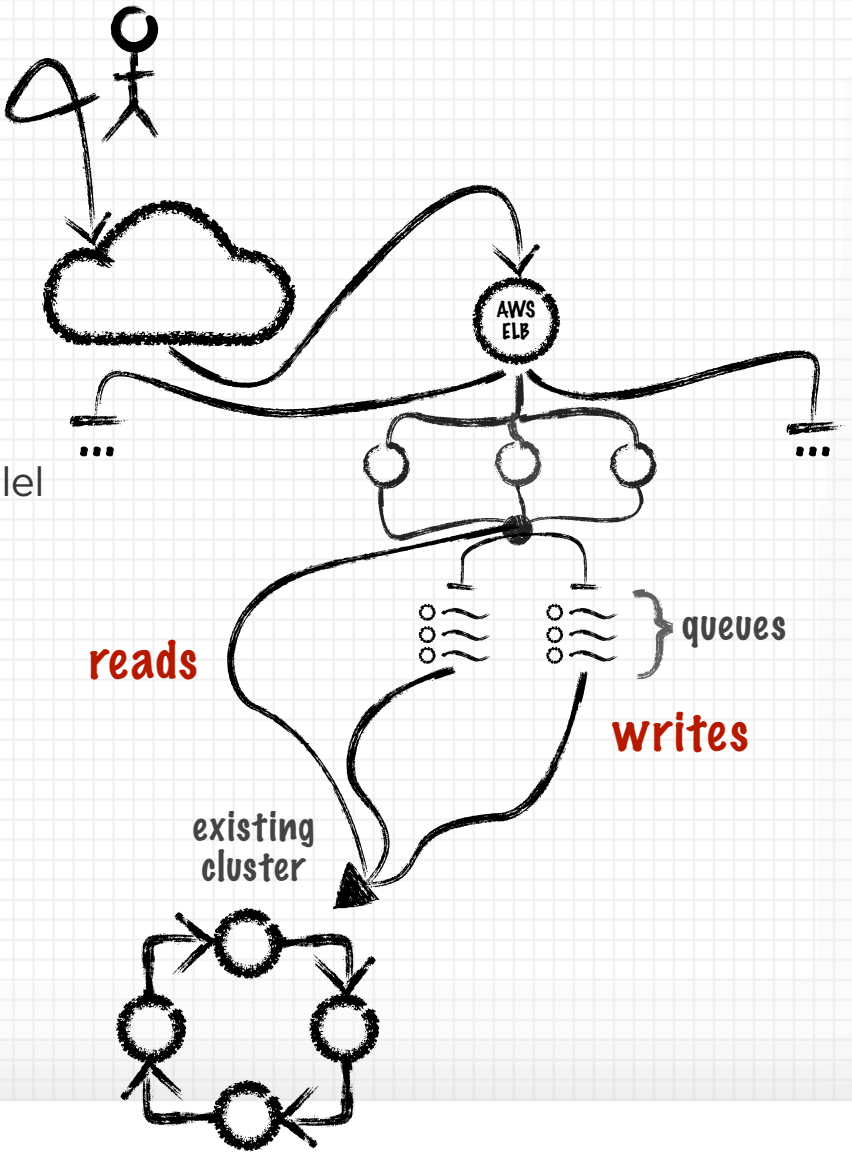
- Remove an availability zone from the ELB
- Upgrade Gate in that availability zone
- Validate that the new version is performing as expected
- Re-associate the availability zone with ELB
- Repeat for other availability zones

UPGRADING CASSANDRA PARALLEL ADOPTION

UPGRADING CASSANDRA

- Cassandra mutations are idempotent and commutative
  - You can apply a mutation multiple times
  - You can apply mutations in any order
- A queue decouples database clusters from end-user visible systems
- Using AWS, it's cheap and easy to provision parallel infrastructure

PARALLEL ADOPTION



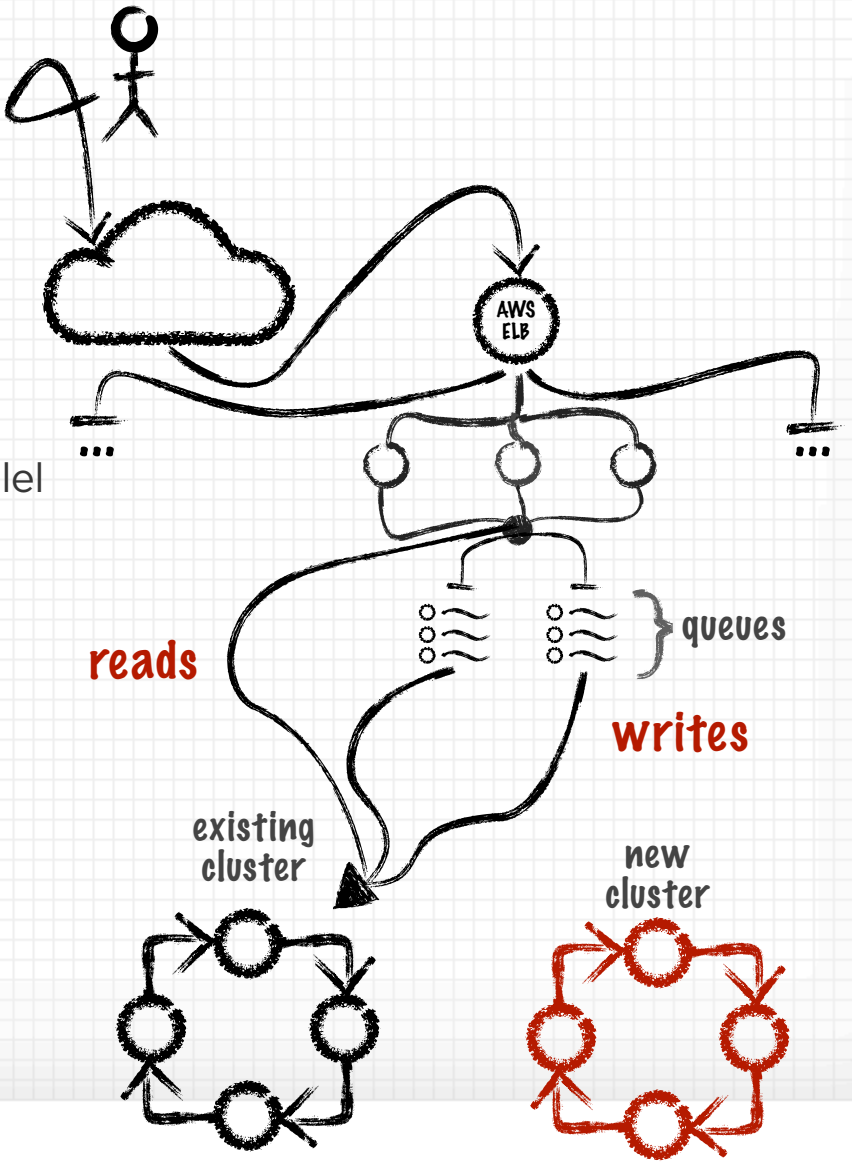
UPGRADING CASSANDRA PARALLEL ADOPTION

UPGRADING CASSANDRA

- Cassandra mutations are idempotent and commutative
  - You can apply a mutation multiple times
  - You can apply mutations in any order
- A queue decouples database clusters from end-user visible systems
- Using AWS, it's cheap and easy to provision parallel infrastructure

PARALLEL ADOPTION

- Provision new cluster running new software



SimpleGeo

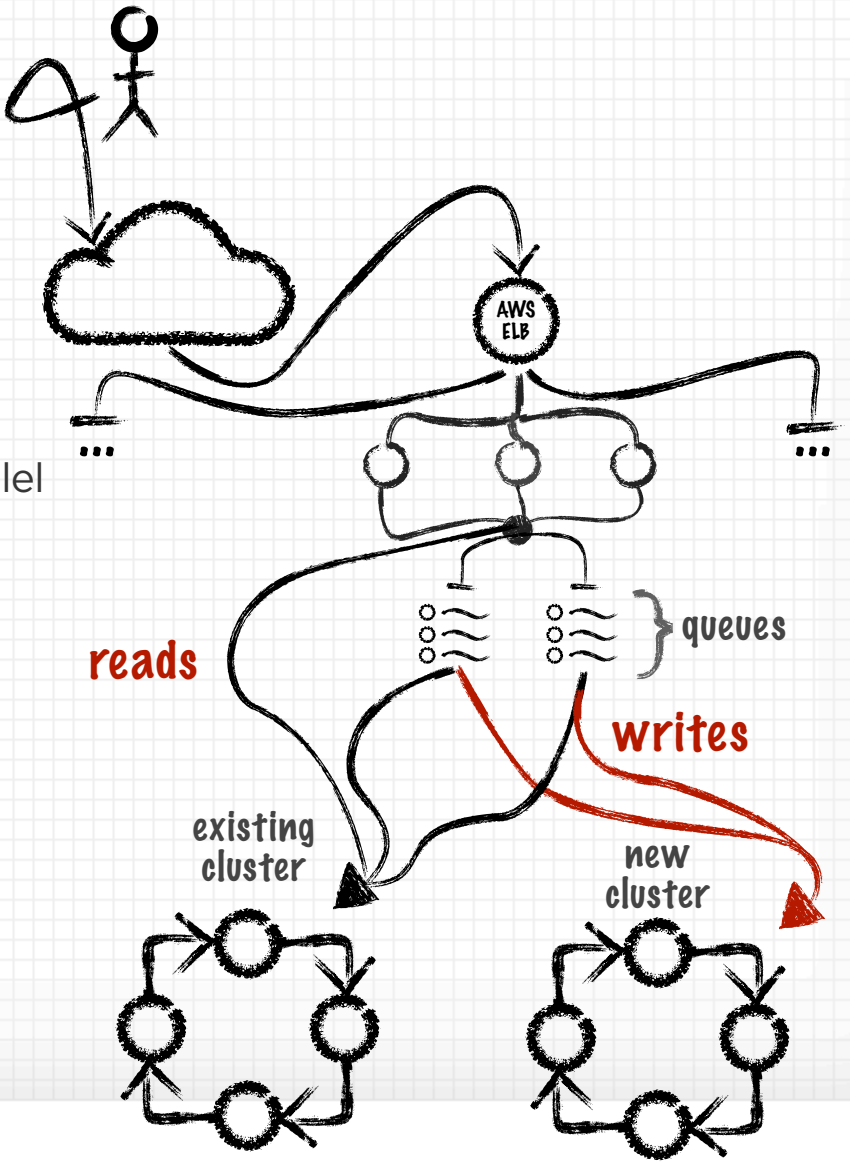
UPGRADING CASSANDRA PARALLEL ADOPTION

UPGRADING CASSANDRA

- Cassandra mutations are idempotent and commutative
  - You can apply a mutation multiple times
  - You can apply mutations in any order
- A queue decouples database clusters from end-user visible systems
- Using AWS, it's cheap and easy to provision parallel infrastructure

PARALLEL ADOPTION

- Provision new cluster running new software
- Start writing new data to both clusters



SimpleGeo

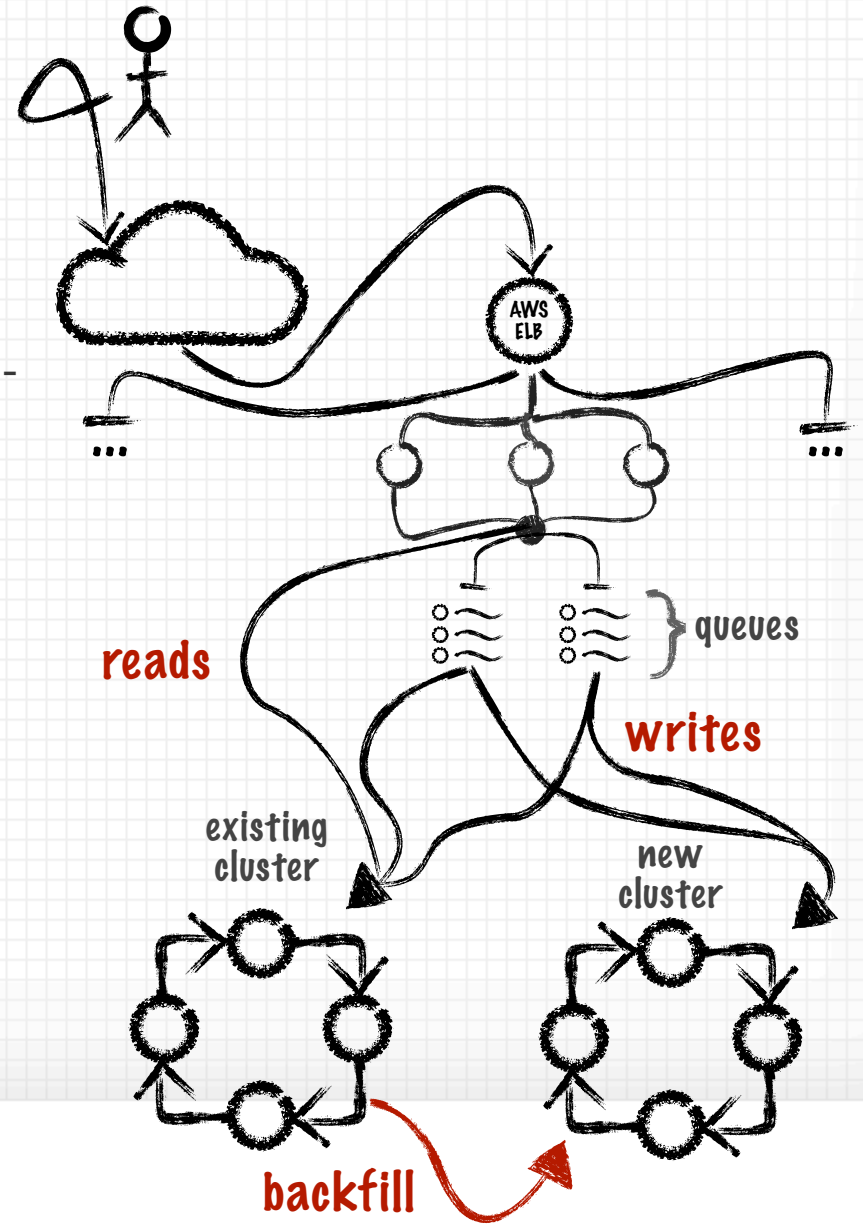
UPGRADING CASSANDRA PARALLEL ADOPTION

UPGRADING CASSANDRA

- Cassandra mutations are idempotent and commutative
  - You can apply a mutation multiple times
  - You can apply mutations in any order
- A queue decouples database clusters from end-user visible systems
- Using AWS, it's cheap and easy to provision parallel infrastructure

PARALLEL ADOPTION

- Provision new cluster running new software
- Start writing new data to both clusters
- Backfill old data from existing cluster to new cluster



SimpleGeo

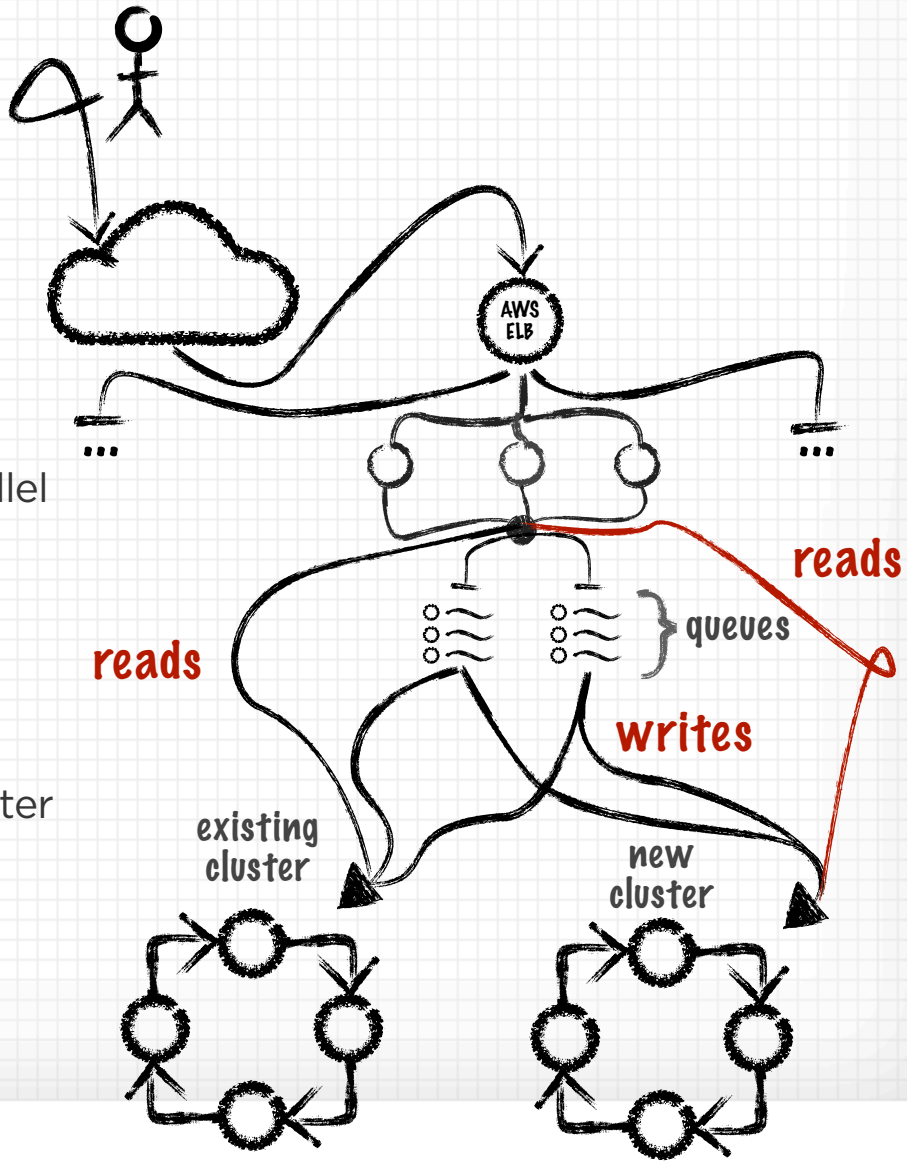
UPGRADING CASSANDRA PARALLEL ADOPTION

UPGRADING CASSANDRA

- Cassandra mutations are idempotent and commutative
  - You can apply a mutation multiple times
  - You can apply mutations in any order
- A queue decouples database clusters from end-user visible systems
- Using AWS, it's cheap and easy to provision parallel infrastructure

PARALLEL ADOPTION

- Provision new cluster running new software
- Start writing new data to both clusters
- Backfill old data from existing cluster to new cluster
- Validate that the new cluster is performing as expected



SimpleGeo

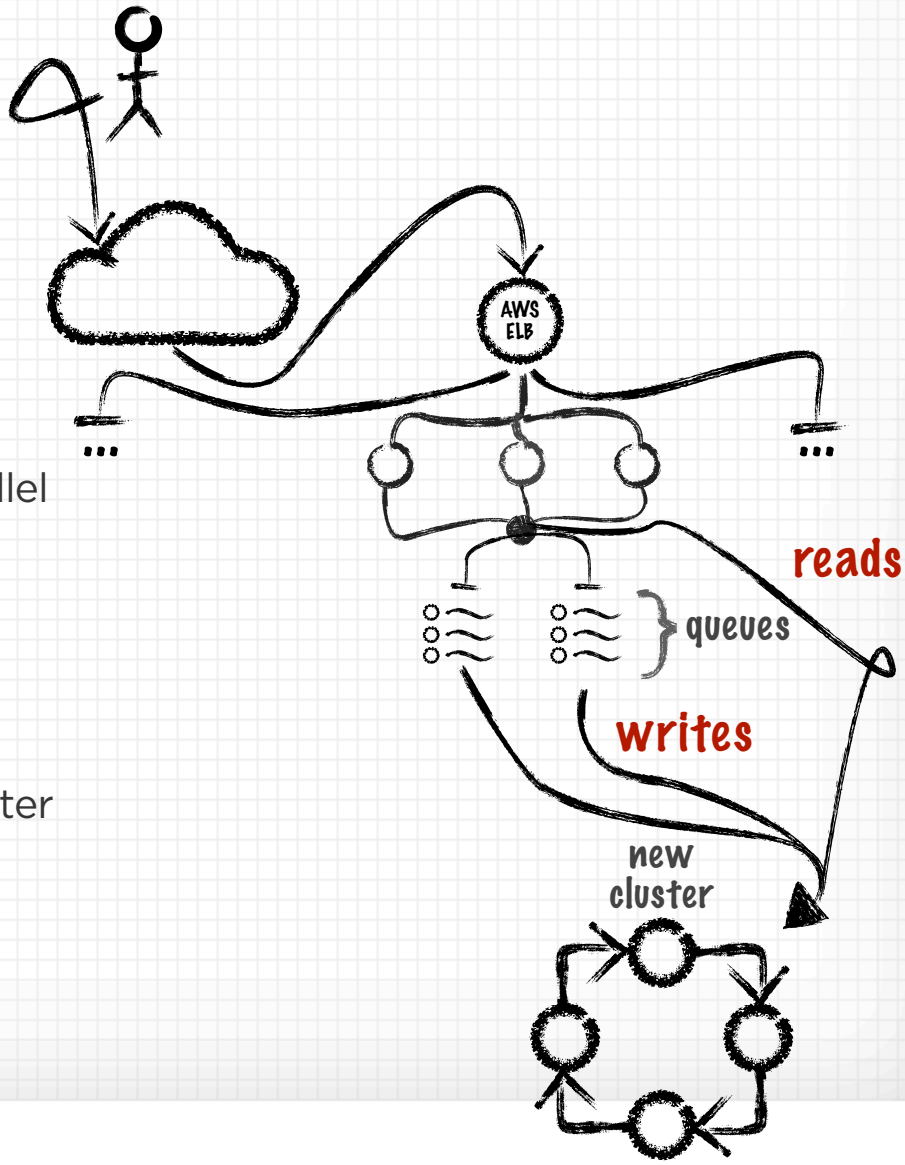
UPGRADING CASSANDRA PARALLEL ADOPTION

UPGRADING CASSANDRA

- Cassandra mutations are idempotent and commutative
  - You can apply a mutation multiple times
  - You can apply mutations in any order
- A queue decouples database clusters from end-user visible systems
- Using AWS, it's cheap and easy to provision parallel infrastructure

PARALLEL ADOPTION

- Provision new cluster running new software
- Start writing new data to both clusters
- Backfill old data from existing cluster to new cluster
- Validate that the new cluster is performing as expected
- Reconfigure clients to read from the new cluster



SimpleGeo

IF IT HURTS, DO IT MORE

ORGANIZATIONS, LIKE PEOPLE, GET BETTER AT THINGS THEY DO OFTEN

- Push all the time - it'll get less painful
- Upgrade all the time - it'll get less painful
- Write tests all the time - it'll get less painful
- Do code reviews all the time - it'll get less painful
- Merge branches all the time - it'll get less painful
- Fail all the time - it'll get less painful

IN GENERAL, IF SOMETHING HURTS, DO IT MORE

ALWAYS BE FAILING TOOLS & SYSTEMS

EXPECT AND EMBRACE FAILURE

- If you think you can prevent failure you're not developing your ability to respond [ALLSPAW, HAMMOND]
- Make the system **resilient** to failure instead trying to **prevent** failure
  - SimpleGeo "database buffer logs" log incoming (idempotent) database operations for replay if a failure occurs

DETECTING FAILURE

- High level monitoring can alert you to user-visible problems
  - Run integration tests in production, alert on failure
- Fail fast and loud so people can quickly see causal relationships between a change and a failure
  - It's ok to throw an exception - when you're coding, if you don't know what to do, resist the temptation to guess
  - If there's a significant lag between something breaking and you knowing it broke there's a larger problem that needs to be addressed

ALWAYS BE FAILING TOOLS & SYSTEMS

ELIMINATE SINGLE POINTS OF FAILURE (SPOFS)

- Component and subsystem failures should not result in total system failures (decouple!)
- If you discover an unexpected correlation between systems then either eliminate it or formalize it
  - In a distributed system, any interaction between systems should be treated as a potential mechanism for cascading failure

KEEP IT SIMPLE AND DON'T REPEAT YOURSELF

- Keep small components and subsystems simple by allowing them to fail (again, they should fail loudly)
- Introduce simple fault-tolerance mechanisms strategically at a high-level to address a range of failure scenarios
  - Gate - the entry point into our API - retries HTTP requests that timeout, so other services don't bother retrying database requests or requests to sub-services

ALWAYS BE FAILING CULTURE

LEARN FROM FAILURE, BUT MAKE SURE PEOPLE AREN'T STIGMATIZED OR AFRAID TO FAIL

- Things move way faster when people aren't afraid to fail
- Make failure culturally acceptable - it's an opportunity for improvement
- Do blame free postmortems
- Be careful of a "culture of shame" - jokes aside, respect and disappointment are way more effective
- Red cards at Urban Airship - invest in improvements in proportion to the cost of the failure

SECURITY
DON'T BE A PEDANT

SECURITY

- There is a natural tension between security and usability (a door with a lock is harder to use) - balance
- There are decreasing returns with increased security
  - If security makes a system too hard to use people start developing techniques for defeating the security (e.g., leaving the door propped open)

PERIMETER SECURITY

- Like fault-tolerance, we've seen the best results when security is centralized, not deep - in other words, when the system is secured at its perimeter
- Give developers sudo privileges in production, but with accountability (no single root account, proxy box with non-erasable logs)
  - We distribute SSH keys with Puppet and lock down access to a single EC2 host (SSH can be configured to make this easy)

WHAT'S THE PROBLEM

- Like failure, security incidents highlight organizational shortcomings - if you're worried that your engineers will nuke all of your production boxes you have bigger issues

SECURITY
SOME EXCEPTIONS

GOLDEN GATE FOR CRITICAL STUFF

- Web-service proxy that adds accounting, time-lock and two-person authentication
- A fail-safe system isn't really fail-safe unless it's procedurally fail-safe (example: ELB)
  - Multi-factor authentication is a good way to add a procedural fail-safe to a system that doesn't come that way out of the box
    - Take the MFA keyfob, put it in a safe-deposit box, swallow the key
    - Good for: ELB, S3, or your own physical hardware

SOFTWARE ARCHITECTURE
WHAT MAKES IT GOOD?

GOOD ARCHITECTURE IS

- **Robust** lacking bugs and tolerant of faults
- **Maintainable** easy to maintain and extend
- **Useful** utility, beyond the immediate need
- **Scalable** able to grow in capacity
- **Common vision** direction, strategy
- **Agile** simple enough to refactor easily
- **Extensible** able to grow in features
- **Responsive** performant now and after expanding or scaling

WARD'S WIKI <C2.COM/CGI/WIKI?GOODARCHITECTURE>

BEASTIE

THINKING AHEAD

DON'T CORNER YOURSELF CONCEPTUALLY

BUILD FOR ONE YEAR, PLAN FOR TEN

- Time-to-market is important, but you also need to be able to manage and scale your systems after you've gotten there
  - Architectural assumptions are hard to change
  - If you don't plan ahead your future product roadmap will be driven by technological limitations
- Technology is like crappy furniture - it has a tendency to remain in our lives much longer than it should

RELATED...

- In the information technology industry we tend to overestimate what we can do in a year and underestimate what we can do in a decade **[Marc Benioff]**

AVOID THE UGLY COUCH - SIT ON THE FUCKING FLOOR.

There is a lesson here for ambitious system architects: the most dangerous enemy of a better solution is an existing codebase that is just good enough.

ERIC S. RAYMOND
THE ART OF UNIX PROGRAMMING

INTERFACES OVER IMPLEMENTATIONS

IF YOU'RE SHORT ON RESOURCES, INVEST IN INTERFACES FIRST

- "Crappy" implementations are often good enough
  - If you're off the critical path a simple implementation that's slow is often better than complex one that's more efficient
- Interfaces are forever (or as close as you can get in this business)
  - Interfaces shape the system in unanticipated ways
  - Changing an interface requires rewriting tests, risks related to backwards compatibility, extensive manual QA, etc.

ABSTRACT ABSTRACTIONS

COMMUNICATE CONCEPTS, NOT JUST CODE

- It's worth designing and discussing systems using terminology that doesn't depend on a particular technology
- Decouples the solution from a particular technology choice, which gives engineers a larger toolbox and more freedom

BIG BOARD STATISTICS AT SIMPLEGEO

- Went through several major iterations
  - In memory => memcachedb => SimpleDB => redis
- The interface never changed - it was designed with a simple goal in mind and the backing store was insignificant
  - Client code didn't have to change
  - Unit tests remained the same, allowing us to validate changes

BEAUTY IN SIMPLICITY

DEFINE & EMPHASIZE CORE CONCEPTS

- Simplicity in computer science often means reducing an idea to its conceptual core
  - Emphasizing the concept reduces debate / butthurtedness about specific technologies
  - Having a clear, well articulated, shared understanding of the conceptual foundation for a piece of technology tends to produce better code
- Additional features can be added as “decoration” on top of this core (postmodern software development)
- Example: SimpleGeo Storage
  - When we explain the technology we emphasize the ideas, not the fact that it’s written in Java, built on top of Cassandra, etc.

PENELOPE
DHT SECONDARY INDEXES

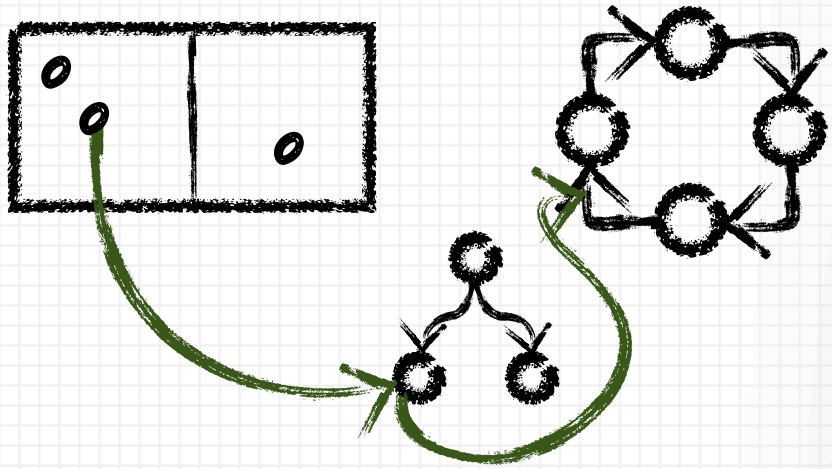
SECONDARY INDEXES

- In a distributed hash table (DHT) since they offer straightforward fault tolerance and operational simplicity
- Notoriously difficult to implement properly since they necessarily span nodes (physical machines)
  - If your secondary indexes are on the same physical machine as the data being indexed they're not very useful (you need to query every machine)
  - If your secondary indexes are distributed separately from the data being indexed you're spanning physical hardware

PENELOPE
DHT SECONDARY INDEXES

HOW IT WORKS

- Tree nodes are identified by DHT keys
- Data value and key are inverted and stored in appropriate leaf nodes
  - {id: 25, name: “Mike”} => “Mike:25”
- When leaf nodes exceed a configured capacity, they’re split
- Querying proceeds by passing function continuations amongst nodes to be executed “next to” the data
- Network hops are reduced by caching “approximations” of the tree in local memory



PENELOPE
DHT SECONDARY INDEXES

IDEA IS SEPARABLE FROM IMPLEMENTATION

- When we discuss Penelope we emphasize the idea, not the implementation
- To get to market faster we've made implementation compromises, but retained a solid conceptual foundation
- Decoupling has bled into the implementation, where the indexing mechanism is almost entirely decoupled from the underlying DHT

SOFTWARE ARCHITECTURE
WHAT MAKES IT GOOD?

GOOD ARCHITECTURE IS

- **Robust** lacking bugs and tolerant of faults
- **Maintainable** easy to maintain and extend
- **Useful** utility, beyond the immediate need
- **Scalable** able to grow in capacity
- **Common vision** direction, strategy
- **Agile** simple enough to refactor easily
- **Extensible** able to grow in features
- **Responsive** performant now and after expanding or scaling

WARD'S WIKI <C2.COM/CGI/WIKI?GOODARCHITECTURE>

BEASTIE

PLAYING WELL WITH OTHERS

(WE'RE ALL ON THE SAME TEAM)

SimpleGeo

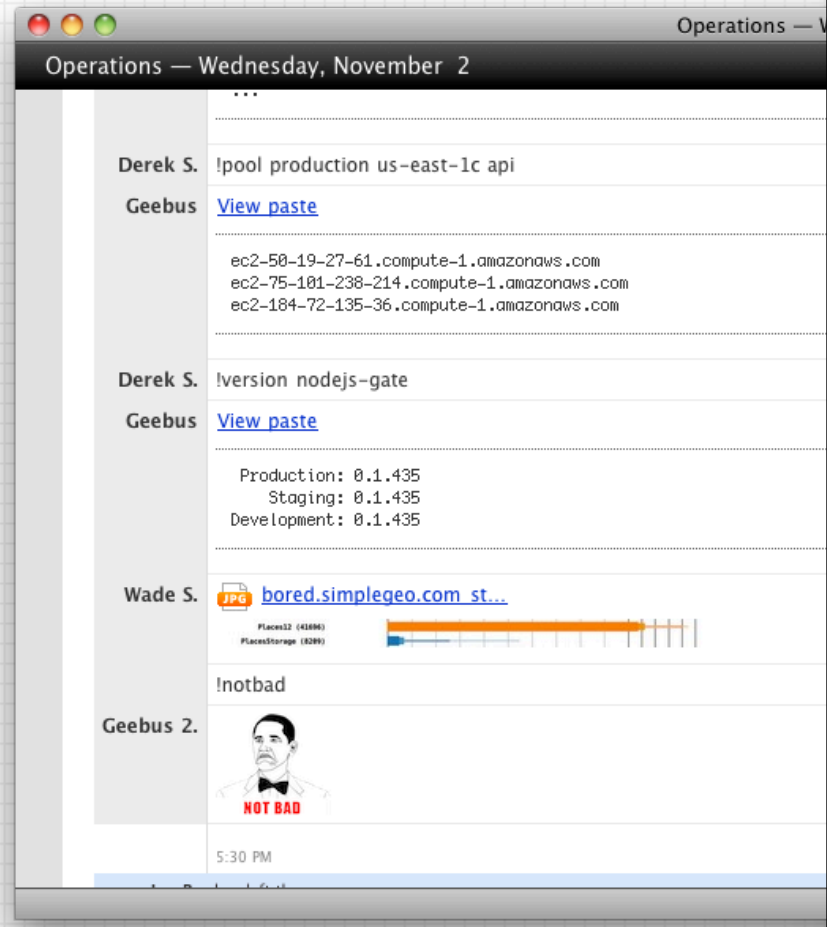
COLLABORATION

GITHUB

- We've had great success using GitHub for all of our repositories (public and private)
  - Excellent interface for code navigation and repository management
  - Code reviews become pull requests

COMMUNICATION

- At various times we've used Campfire and IRC - both work
- Either way, we use chat to communicate with each other and with our systems
  - **Botriot.com** is a productized version of the bot that we run at SimpleGeo



A CONVERSATION WITH GEEBUS

OPERATIONAL CONCERNS ARE ARCHITECTURAL CONCERNS

IF YOUR SYSTEM IS

- Hard to deploy, configure, or integrate with other systems
- Hard to upgrade without causing outages or introducing a lot of organizational risk
- Hard to monitor, lacking test hooks and other mechanisms for introspection
- Hard to debug when problems occur
- Otherwise causing operational headaches

YOU'VE GET AN ARCHITECTURAL PROBLEM

MORE SCIENCE
LESS RELIGION

IN GOD WE TRUST, ALL OTHERS BRING DATA

- For a discipline that's based in the sciences, practitioners of computer science have an awful lot of religious debates
- Just because someone "doesn't think that change will make a difference" doesn't mean we shouldn't try it - "prove it"

INNOVATE

- Just because nobody else is doing something doesn't mean it's a bad idea
- A lot of large enterprise process-driven development assumes that nothing is ever new and yesterday's solutions are good enough for tomorrow
  - These guys are easily displaced by a competitor who innovates

THE LANGUAGE OF BUSINESS

SELLING IDEAS TO STAKEHOLDERS

- Engineers and architects need to communicate their architectural vision to all stakeholders - if you can't speak their language that's your problem
- Business stakeholders typically make decisions based on utilization of time, money, and other **scarce resources**; economic **reward**; and **risk**

BUILD VS. BUY
A HERETIC'S OPINION

OPERATION VS. DEVELOPMENT

- At scale, the cost of operating and maintaining a system quickly dominates the cost of developing it
- If you have smart, passionate engineers (who you think will stick around for a while) you may be better off building something yourself vs. using something off-the-shelf
  - Your solution only needs to solve your problem - so it will be simpler
  - Your solution is developed by you so you'll know how it works, how to fix it, and how to extend it
  - Your system will have someone internal who is accountable for it - so it will get fixed when it breaks and bugs can't be blamed on external developers or policies
- Most of these considerations also apply to build vs. off-the-shelf open-source software

SOFTWARE ARCHITECTURE
WHAT MAKES IT GOOD?

GOOD ARCHITECTURE IS

- **Robust** lacking bugs and tolerant of faults
- **Maintainable** easy to maintain and extend
- **Useful** utility, beyond the immediate need
- **Scalable** able to grow in capacity
- **Common vision** direction, strategy
- **Agile** simple enough to refactor easily
- **Extensible** able to grow in features
- **Responsive** performant now and after expanding or scaling

WARD'S WIKI <C2.COM/CGI/WIKI?GOODARCHITECTURE>

BEASTIE

GOOD SOFTWARE ARCHITECTURE

Software architecture is about more than just building a system that can do a billion requests per second

PRODUCT

- Anticipate and enable change by providing
  - Monitoring and metrics hooks
  - Mechanisms for aggregating and visualizing data
- Let components fail, but make them fail loudly
- Just enough security is enough security
- Interface over implementation
- Simplify and decouple - small layers composed to create cool stuff

GOOD SOFTWARE ARCHITECTURE

In order to create qualify software you must first create an environment in which quality software can be created

PROCESS

- The right amount is just enough, automate what you can
- If it's important, measure it, and provide timely feedback
- If it hurts, do it more
- Communicate using concepts, not just code
- Play nice with others, if someone else has a problem you have a problem (though it might be solved by communication)
- More science, less religion

BUILD BEAUTIFUL THINGS

IF THERE IS ONE THING YOU DO...

VITRUVIUS' "DE ARCHITECTURA"

- Earliest surviving written work on the subject of architecture
- A **good** building should satisfy three principles [WIKIPEDIA]
  - **Durability** it should stand up robustly and remain in good condition
  - **Utility** it should be useful and function well for the people using it
  - **Beauty** it should delight people and raise their spirits

BEAUTY

- Programming is a art, and **good** software should be beautiful
- Beauty is the ultimate defense against complexity
- People exposed to your software should be delighted, and their spirits should be raised, if that's not the case you're doing it wrong
  - Includes users, stakeholders, programmers, operators, sales staff, competitors, customer support, quality assurance, and many others
- Hard to define, but you know what I mean

QUESTIONS?

MIKE MALONE
LEAD ARCHITECT

mike@simplegeo.com
@mjmalone


SimpleGeo™