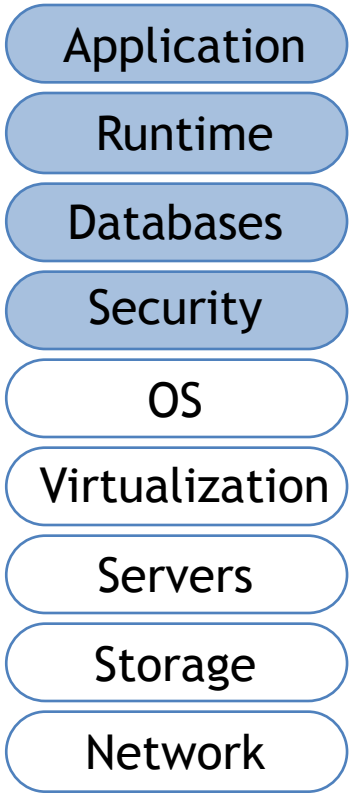
Cloud Computing
#2 - Cloud Applications

Last Session

- Types of cloud
- Drivers behind cloud
- Inside a datacenter
- IaaS & AWS



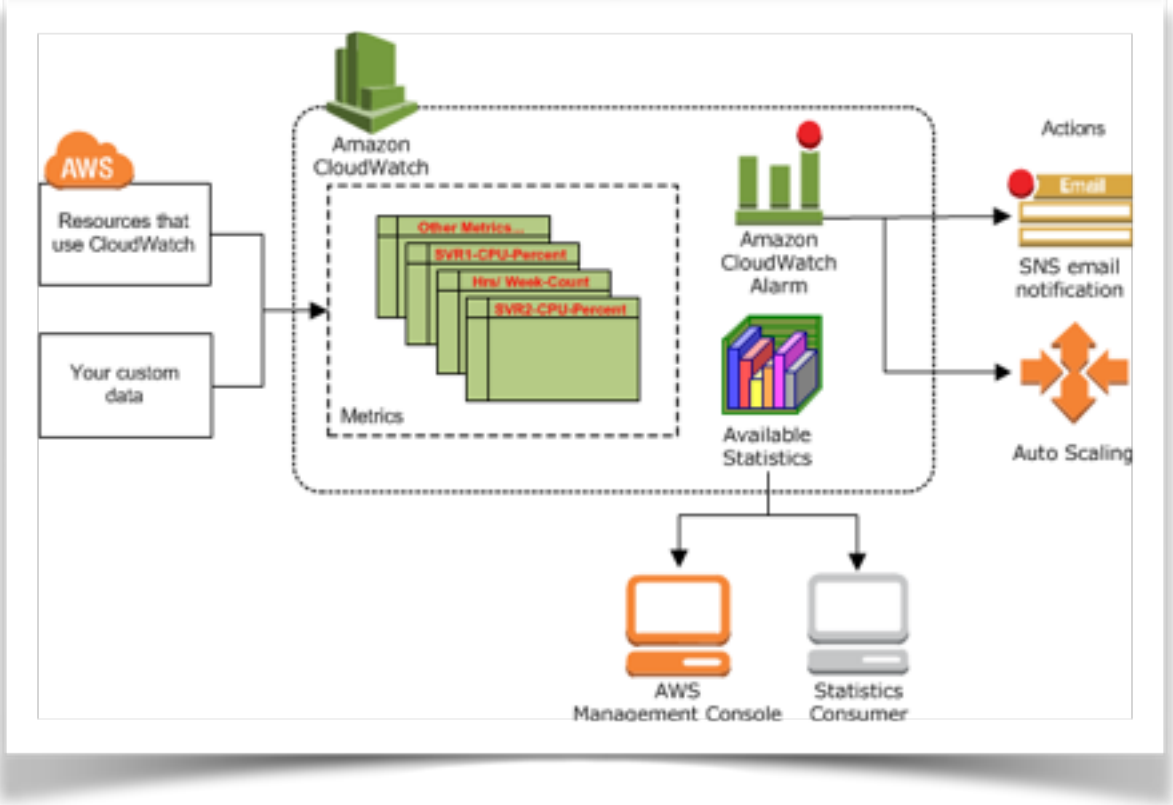
This Session

- Some more on AWS
- PaaS & SaaS
- Discuss cloud applications
  - Identify the important technical components
  - Look some at Netflix & Googe
- Home assignments
- Participant session planning

Scaling up or Scaling out?

- Scale up (or vertical scaling)
  - Adding more memory and CPUs to a single box.
  - Add more threads
  - Ideal for stateful components
- Scale out (or horizontal scaling)
  - Adding more boxes of similar memory and CPU.
  - Ideal for web-tier, and has some of the following characteristics:

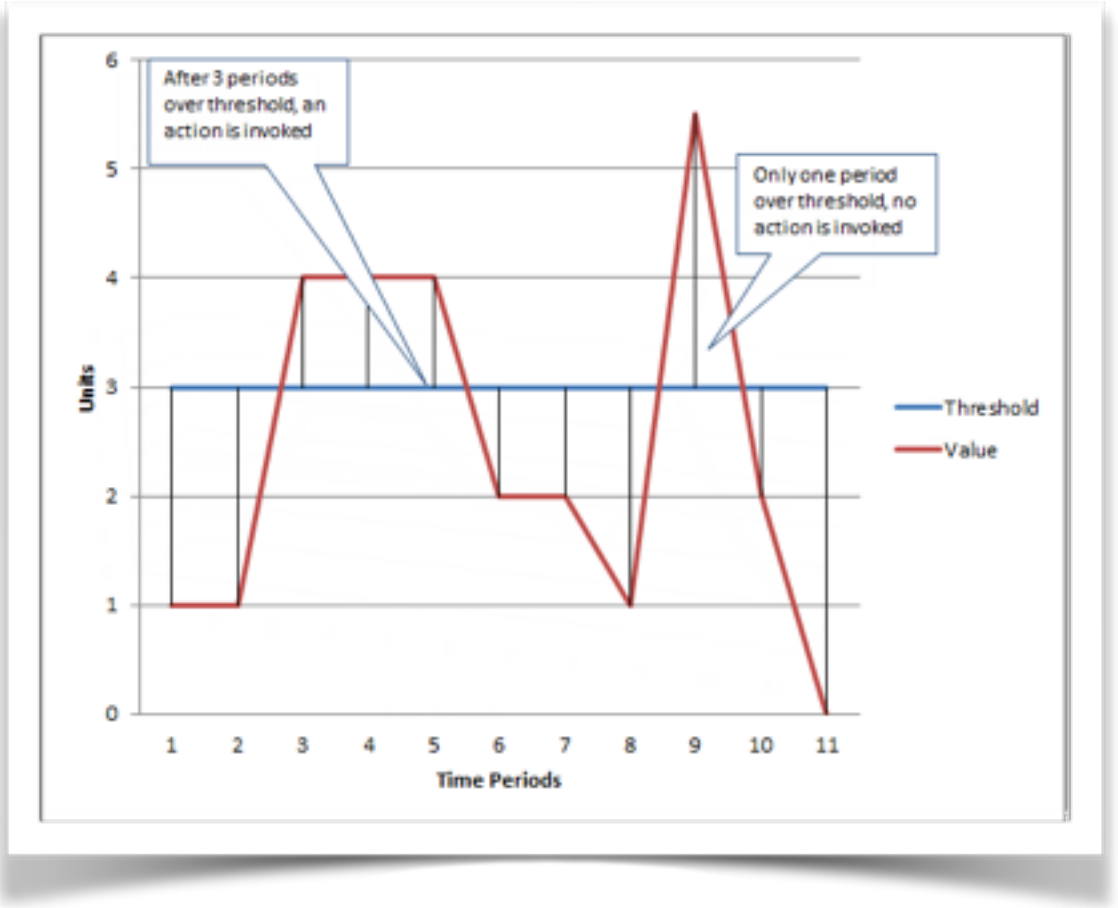
AWS CloudWatch



Monitor CPU utilization, data transfer, disk usage, latency, etc.

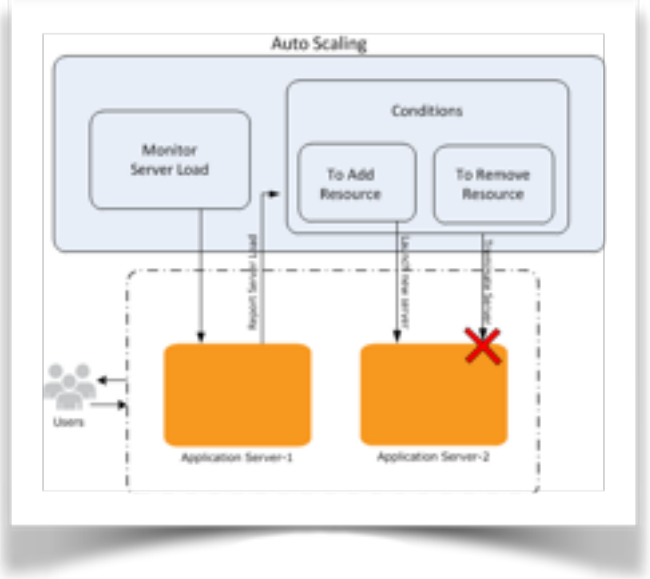
A metric is a time ordered set of data-points such as EC2 CPU usage

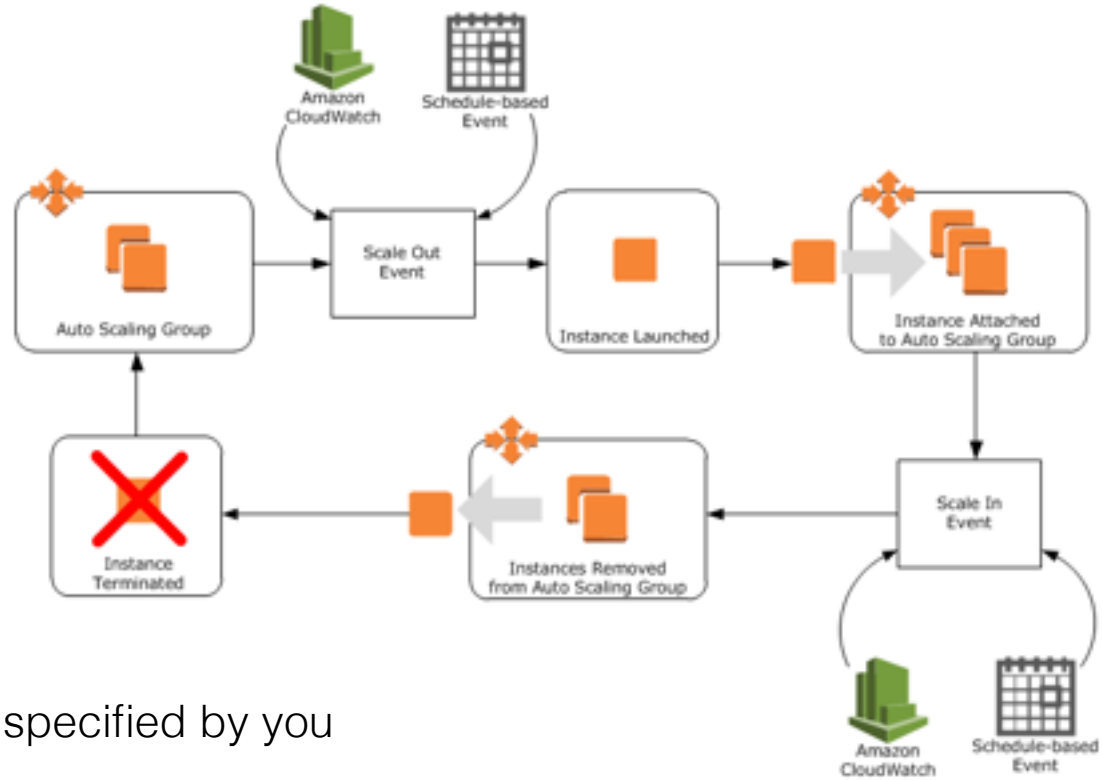
AWS CloudWatch



Set alarms to invoke an action, e.g. start, stop or terminate a VM (EC2)

AWS Autoscaling

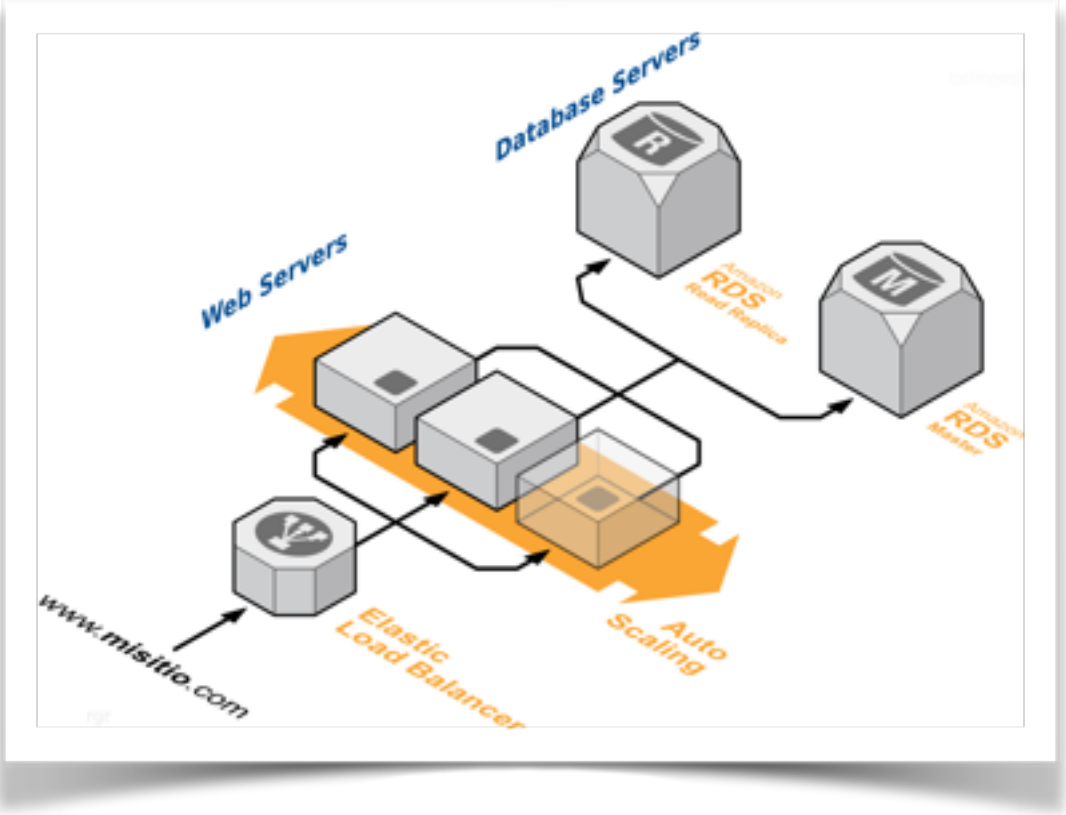




- Dynamically based on conditions specified by you
- Predictably according to a schedule defined by you (every Friday at 13:00:00).
- Scale-out = adding more machines

AWS Load Balancing

- The load balancer is the only computer visible to the outside
- Distribute traffic across multiple VMs
- VMs may be added or removed dynamically
- Monitors health continuously

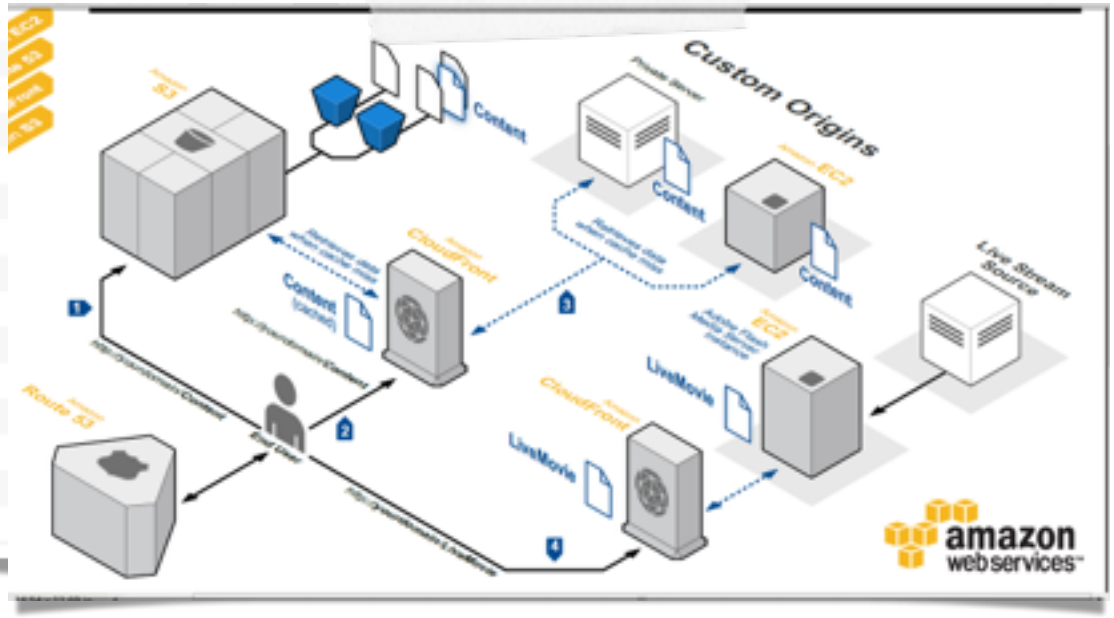


AWS CloudFront

AWS Edge Locations

Amazon CloudFront and Amazon Route 53 services are offered at AWS Edge Locations:

North America	South America	Europe / Middle East / Africa	Asia Pacific
Atlanta, GA	Rio de Janeiro, Brazil	Amsterdam, The Netherlands (2)	Chennai, India
Ashburn, VA (3)	São Paulo, Brazil	Dublin, Ireland	Hong Kong, China (2)
Dallas/Fort Worth, TX (2)		Frankfurt, Germany (3)	Manila, the Philippines
Hayward, CA		London, England (3)	Melbourne, Australia
Jacksonville, FL		Madrid, Spain	Mumbai, India
Los Angeles, CA (2)		Marseille, France	Osaka, Japan
Miami, FL		Milan, Italy	Seoul, Korea
New York, NY (3)		Paris, France (2)	Singapore (2)
Newark, NJ		Stockholm, Sweden	Sydney, Australia
Palo Alto, CA		Warsaw, Poland	Taipei, Taiwan
San Jose, CA			Tokyo, Japan (2)
Seattle, WA			
South Bend, IN			
St. Louis, MO			



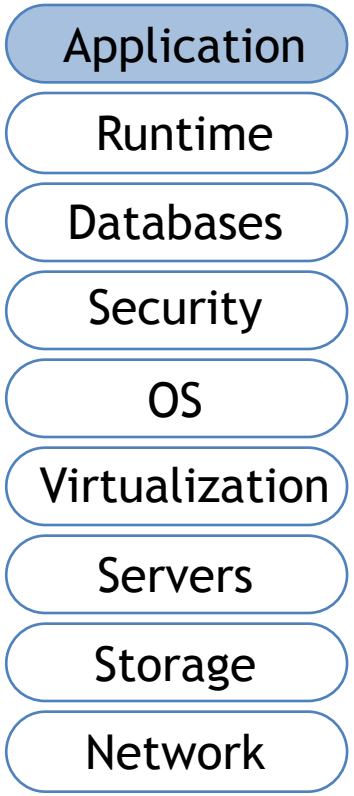
- Content distribution network (CDN)
- Caches S3 content at edge locations for low-latency delivery
- Some similarities to other CDNs like Akamai, Limelight, etc.

AWS Dynamo

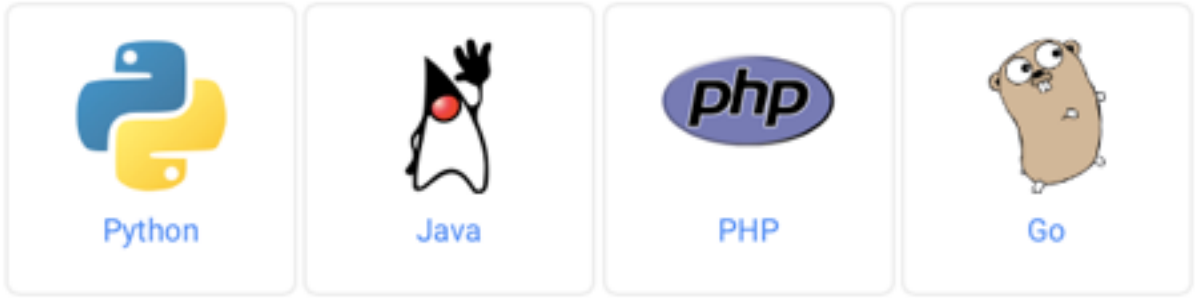
- A NoSQL data base
  - Used for shopping cart: a very high-load application
- Built over a version of Chord DHT
  - Basic idea is to offer a key-value API (like memcached, S3)
  - Support for thousands of service instances
- Basic innovation?
  - To speed things up (think BASE), Dynamo sometimes puts data at the “wrong place”
  - Idea is that if the right nodes can’t be reached, put the data *somewhere* in the DHT, then allow repair mechanisms to migrate the information to the right place asynchronously

Platform-as-a-Service

- Cloud provides runtime/middleware
  - Java VM, Python VM, JS VM
  - Databases, communication, etc.
- User does not manage/control application infrastructure (network, servers, OS, etc.)
- PaaS handles scale-out
- Customer pays SaaS provider for the service; SaaS provider pays the cloud for the infrastructure
- Example: Windows Azure, Google App Engine, *Examples*: Google App Engine, Node.js, Map Reduce



Google App Engine



- App Engine invokes your app's servlet classes to handle requests and prepare responses in this environment.
- Add
  - Servlet classes, (*.java)
  - JavaServer Pages (*.jsp),
  - Your static files and data files,
  - A deployment descriptor (the web.xml file)
- Auto scale to 7 billion requests per day

Lots of APIs

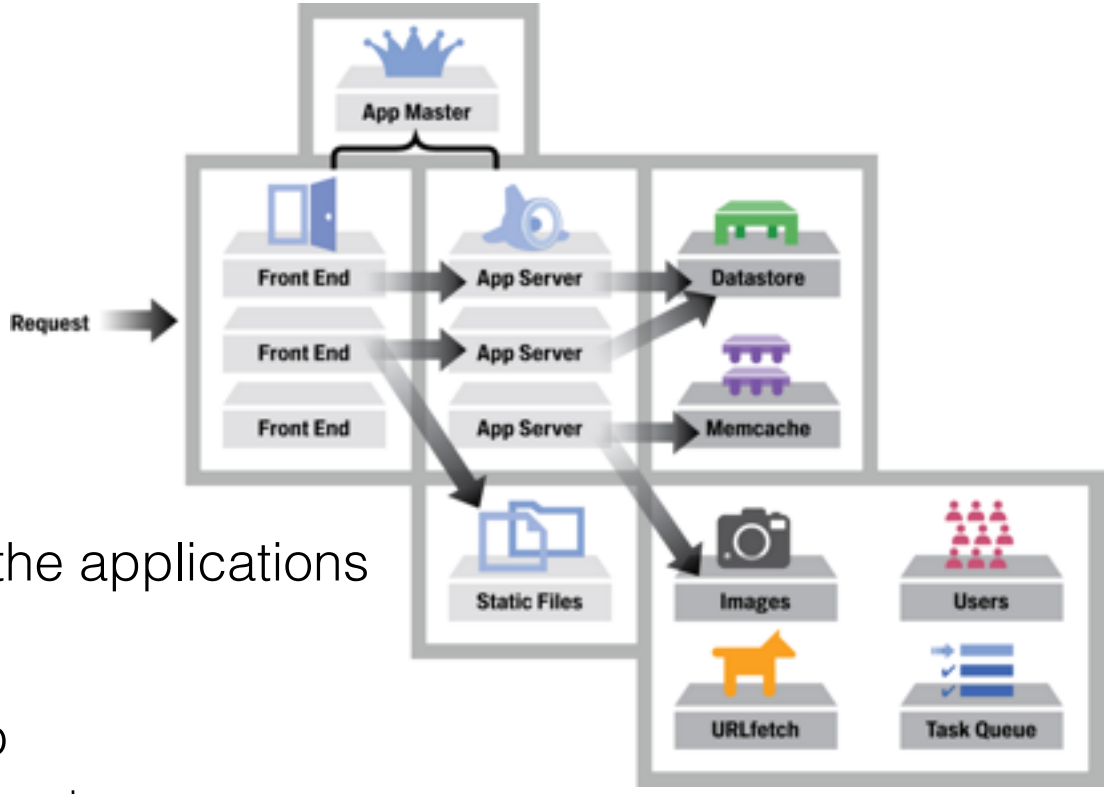
- A large set of scalable features:
  - Mail: APIs to gmail
  - Users: APIs to google user account info
  - Image: APIs to manipulate images, resize, crop, …
  - URLfetch: fetch other URLs
  - Task Queue: support multiple threads in App, allow it to perform background tasks while handling user request
  - XMPP: APIs to google talk
  - https://cloud.google.com/appengine/features/
- GAE handles all the tricky stuff
  - scaling, redundancy, load balancing

```
public class GuestbookServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws IOException {

    UserService userService = UserServiceFactory.getUserService();
    User currentUser = userService.getCurrentUser();

    if (currentUser != null) {
      resp.setContentType("text/plain");
      resp.getWriter().println("Hello, " + currentUser.getNickname());
    } else {
      resp.sendRedirect(userService.createLoginURL(req.getRequestURI()));
    }
  }
}
```

```
import webapp2

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.write('Hello, World!')

application = webapp2.WSGIApplication([('/', MainPage),], debug=True)
```

```
application: your-app-id
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /.*
  script: helloworld.application
```

Features



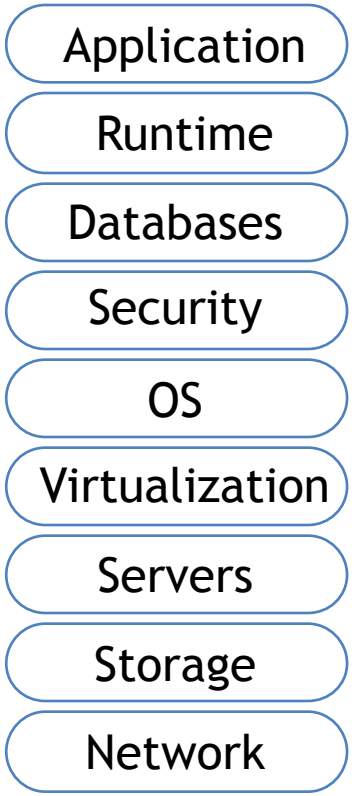
- App Master
  - Schedules applications
  - Manage the replication of the applications
- Front Ends
  - Route dynamic requests to
    - Static files: if accessing static web pages
    - App Servers: if accessing dynamic contents
  - Load balancing
    - Select the nearest and lightest-loaded server (for both static and dynamic contents)

App Server

- Datastore provides persistent storage
  - Replicated for fault tolerance (geographically distributed)
  - Cache is maintained automatically
- App server deployment
  - One process per app, many apps per CPU
  - Inactive processes are killed eventually
- Busy apps (many QPS) get assigned to multiple CPUs
  - This is automatically assigned, as long as CPUs are available
- Constrain direct OS functionality for security
  - No processes, threads, dynamic library loading
  - No sockets (use urlfetch API)
  - Can't write files (use datastore)
- Limited quota to avoid DoS (denial of service) attacks
  - Constraints on CPU time and storage size

Software-as-a-Service

- Cloud provides an entire application
  - Often running in the browser
- Application and data hosted centrally
  - No installation, zero maintenance
  - No control (no need for a sysop)
- Example:
  - Google Apps, Word processor, presentation, spreadsheet, calendar, photo, CRM

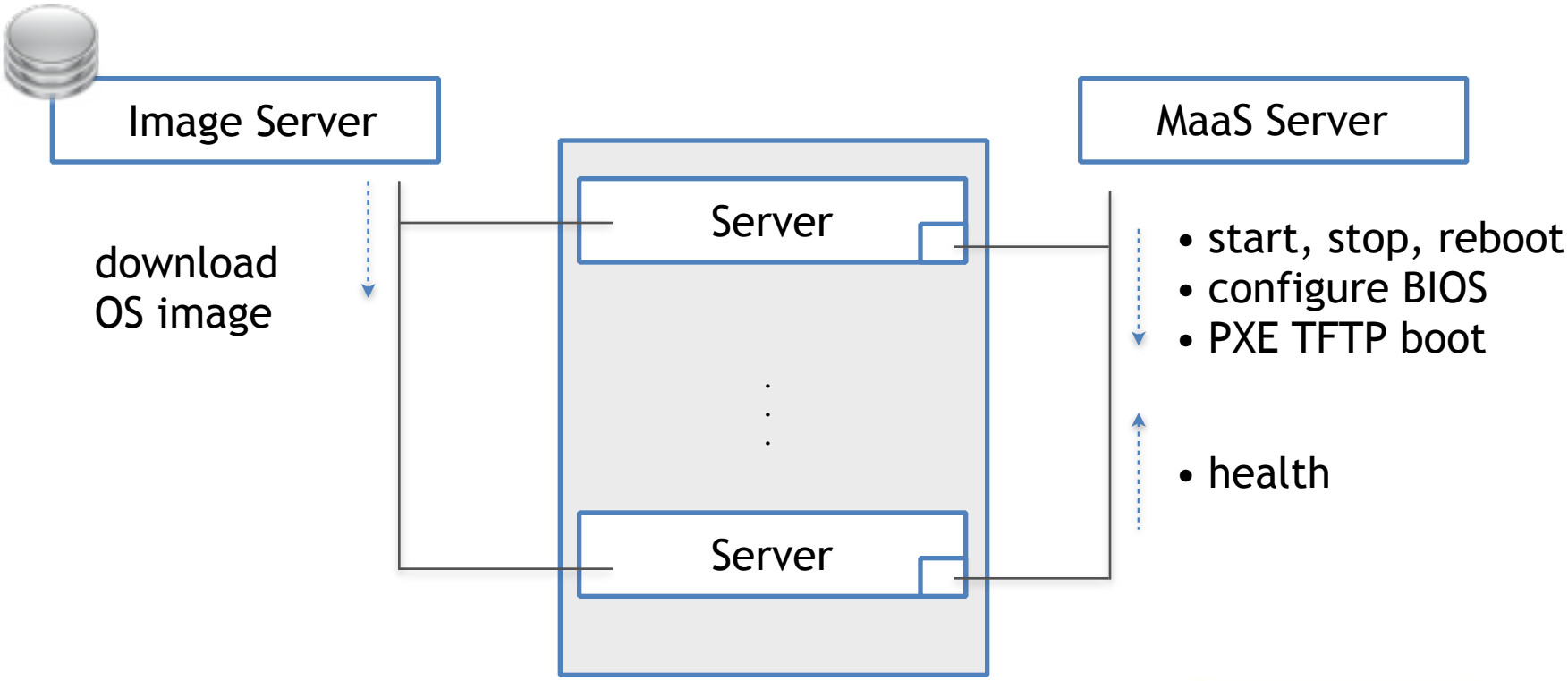


Differences between SaaS and traditional model

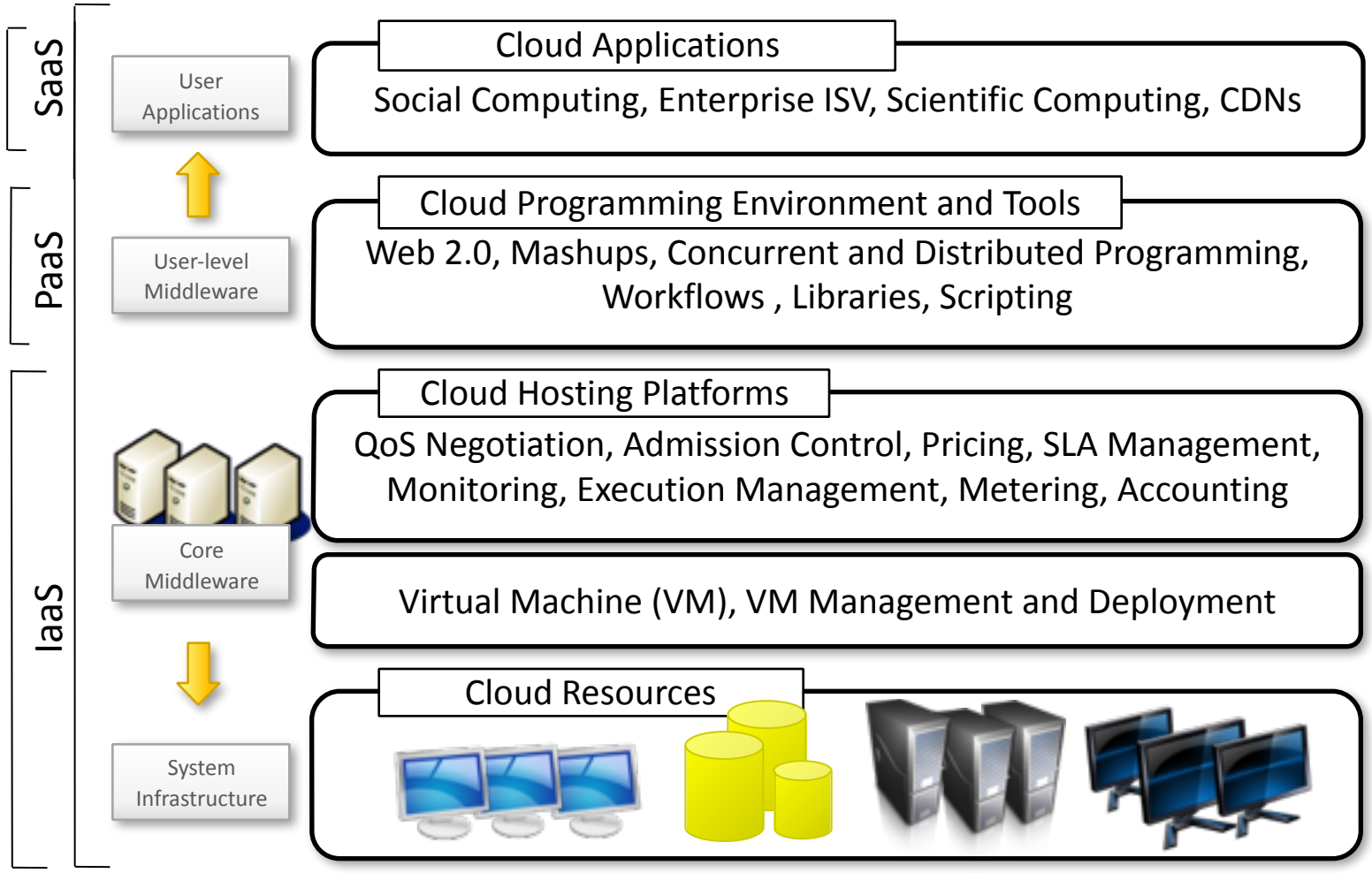
	Hosted (Traditional)	Cloud (SaaS)
Price	One time – large upfront cost	Subscription – low initial cost
Accessed from	Local hardware	Any mobile device
Upgrades	Manual	Automatic
User Adoption	Normal	Higher, due to mobile access
Deployment	IT Know-how and resources required	Minimal IT Setup Required
Security	Handled by local IT	Handled by Service provider
Data Storage	Your local servers	Service provider's servers
Incentive for Software Vendor	Make initial sale	High re-subscription rate

Configuring the Datacenter

"Metal-as-a-Service"



IPMI - Intelligent Platform Management


SaaS
PaaS
IaaS
User Applications
User-level Middleware
Core Middleware
System Infrastructure
Cloud Applications
Social Computing, Enterprise ISV, Scientific Computing, CDNs
Cloud Programming Environment and Tools
Web 2.0, Mashups, Concurrent and Distributed Programming, Workflows , Libraries, Scripting
Cloud Hosting Platforms
QoS Negotiation, Admission Control, Pricing, SLA Management, Monitoring, Execution Management, Metering, Accounting
Virtual Machine (VM), VM Management and Deployment
Cloud Resources

The Size of It

- Large data sets
  - ~50 billion web pages index by Google
  - Average size of webpage = 20KB
  - 50 billion * 20KB = 1 PB (1^15 B)
  - Disk read bandwidth = 1 GB/sec
  - Time to read = 10^6 seconds = 11+ days
- Node failures:
  - A single server can stay up for 3 years (1000 days)
  - 1000 servers in cluster => 1 failure/day
  - 1M servers in cluster => 1000 failures/day
- Network
  - Network bandwidth = 1 Gbps => Moving 10TB takes ~1 day
  - Solution: Push computation to the data
- Round trip between Europe & US ~200 ms

Cloud Native Applications

- Design for failure Avoid single points of failure
  - Assume everything fails, and design backwards
- Goal: Applications should continue to function even if the underlying physical hardware fails or is removed or replaced
- Couple loosely to make it scale
  - Independent components
  - Design everything as a black box
  - De-couple interactions
  - Load-balance clusters

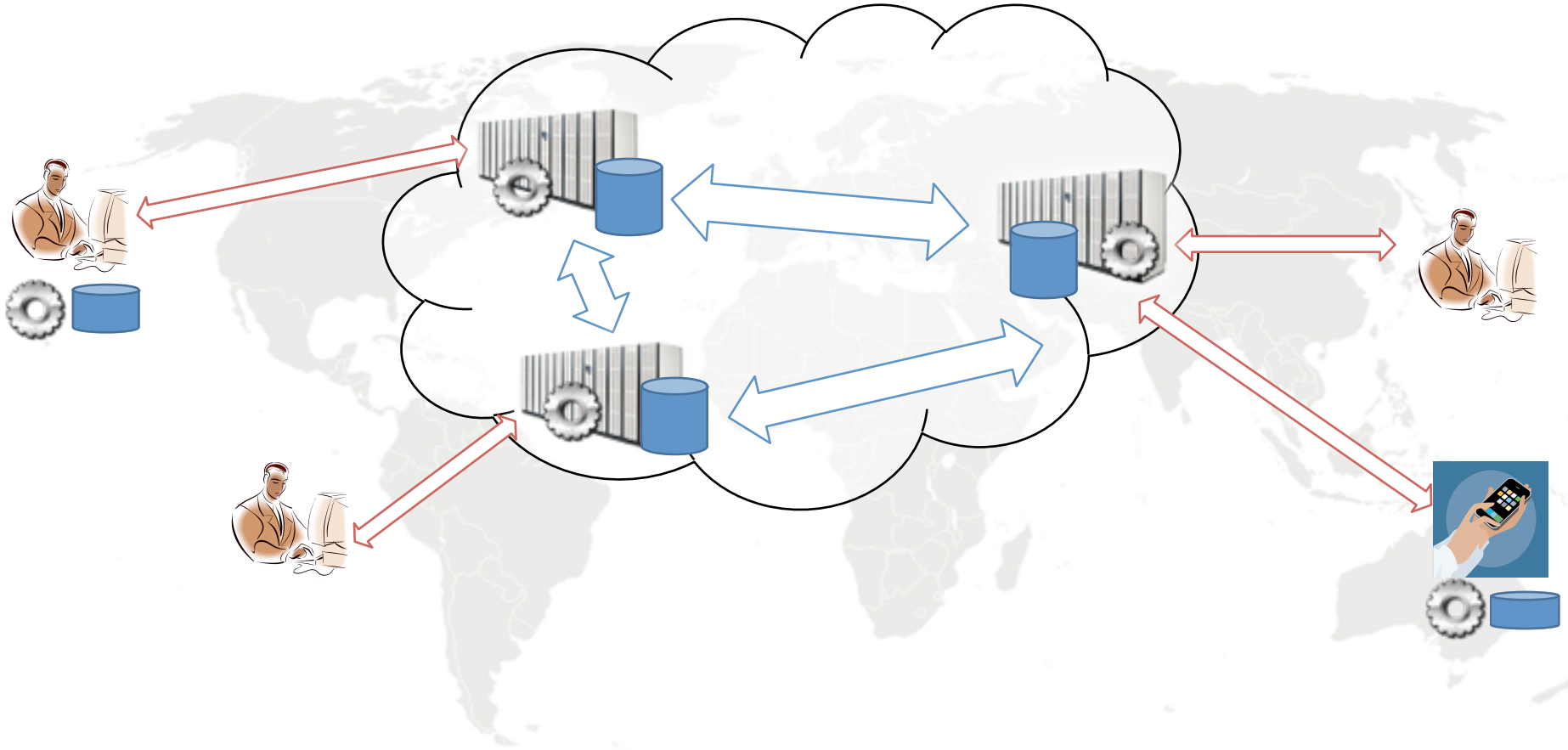
eBay Architecture Guidelines

- Scale Out, Not Up
  - Horizontal scaling at every tier.
  - Functional decomposition.
- Prefer Asynchronous Integration
  - Minimize availability coupling.
  - Improve scaling options.
- Virtualize Components
  - Reduce physical dependencies.
  - Improve deployment flexibility.
- Design for Failure
  - Automated failure detection and notification.
  - “Limp mode” operation of business features.

Stateless Software Architecture

- Do not retain information about the last session into the next – e.g. user data, parameters, logic outcomes.
  - Don’t keep any state in the quickly scalable part of the application
- Idempotence - The property of an operation whereby it can be applied multiple times without changing the result beyond the initial application. It keeps you “safe” because executing the same thing twice (or more) has no more affect than doing it once.

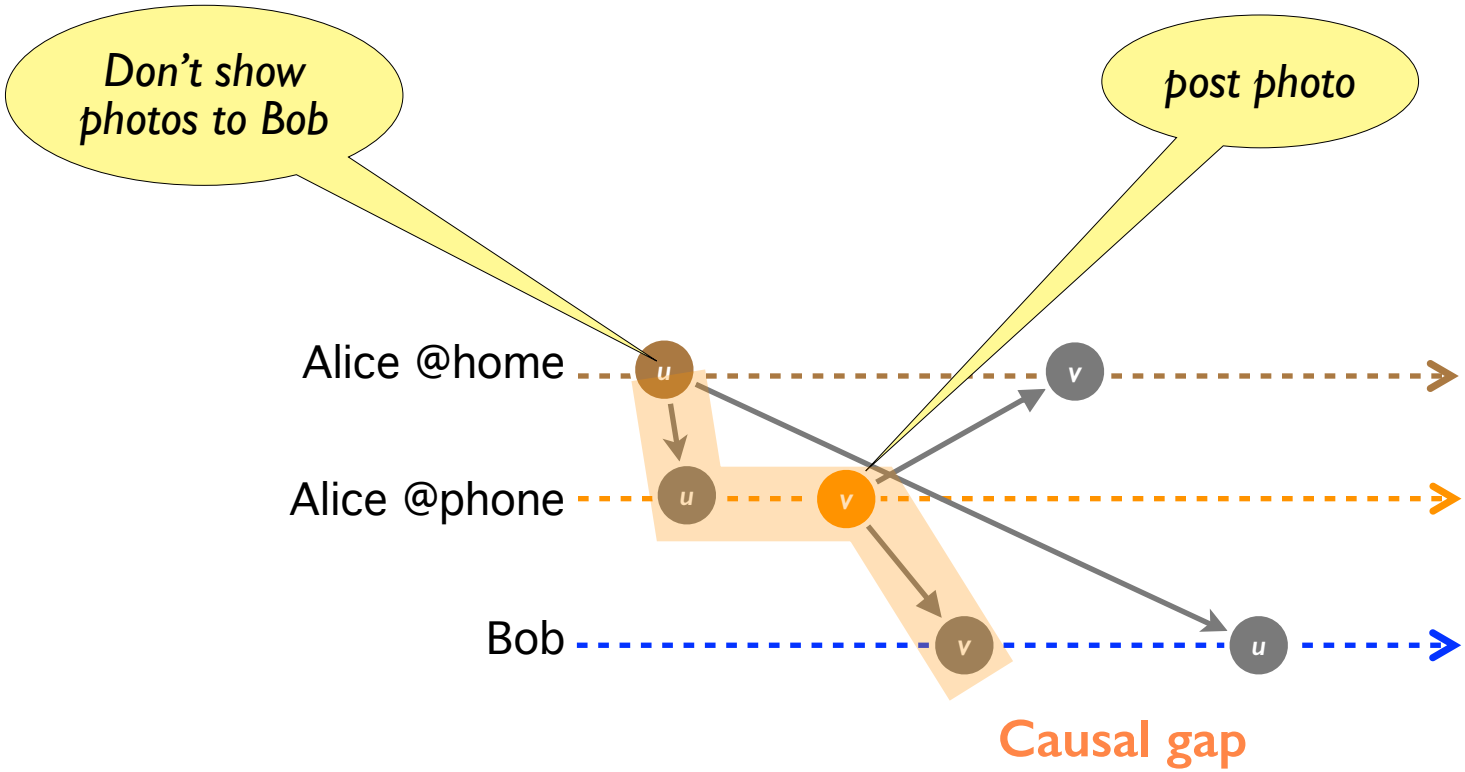
Data Replication



Consistency of shared mutable data

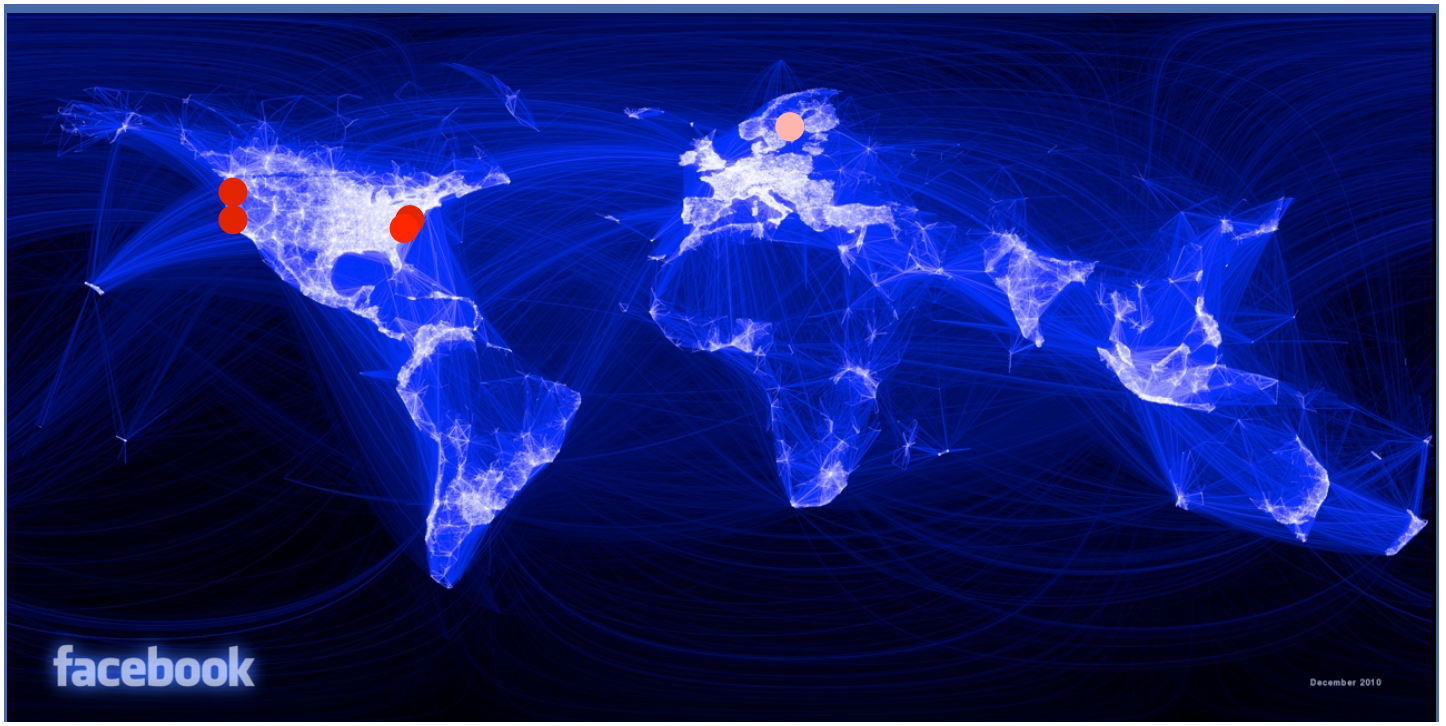
Slide: Marc Shapiro

~~Causal-order delivery~~



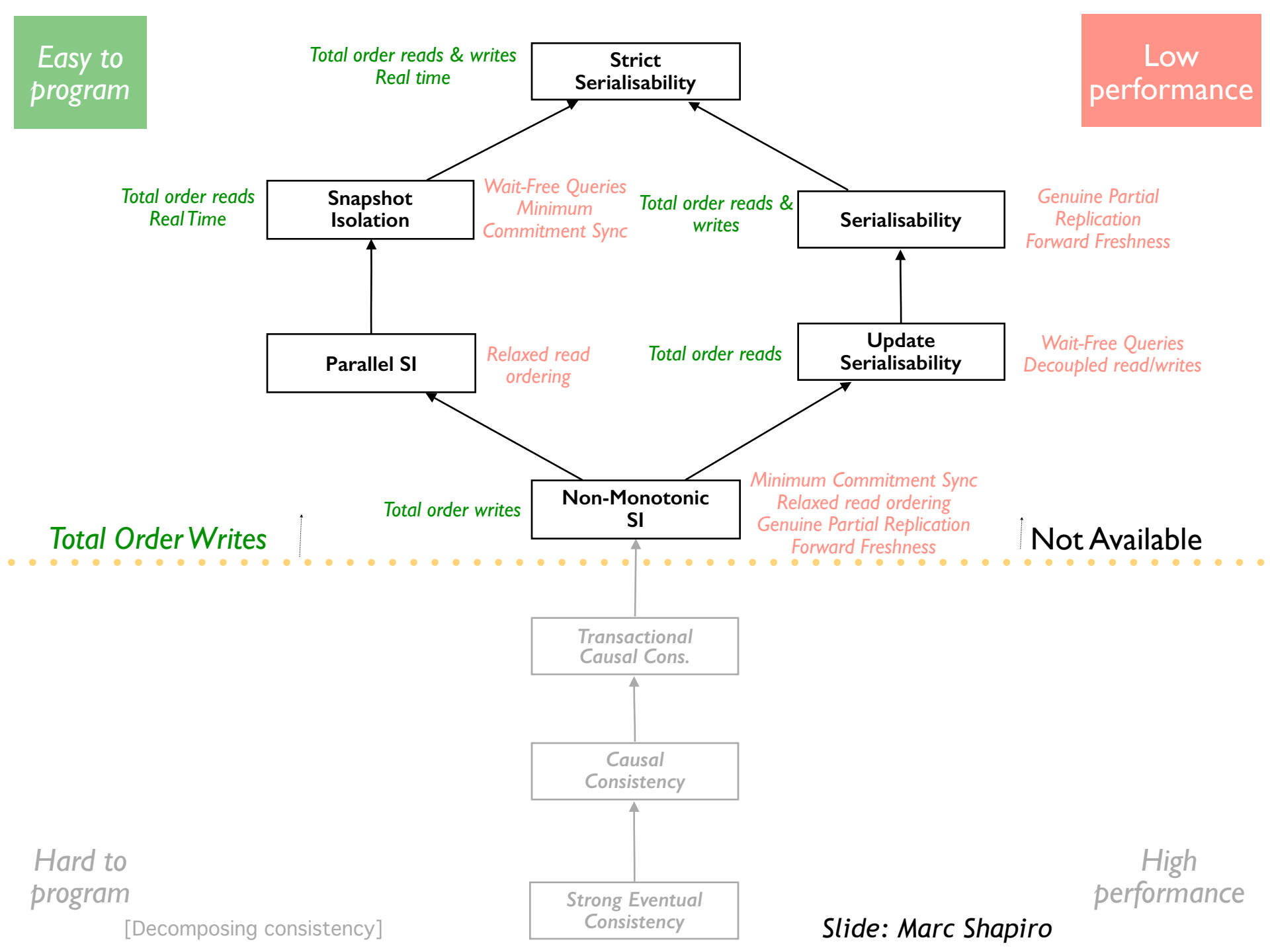
Slide: Marc Shapiro

Example: Facebook

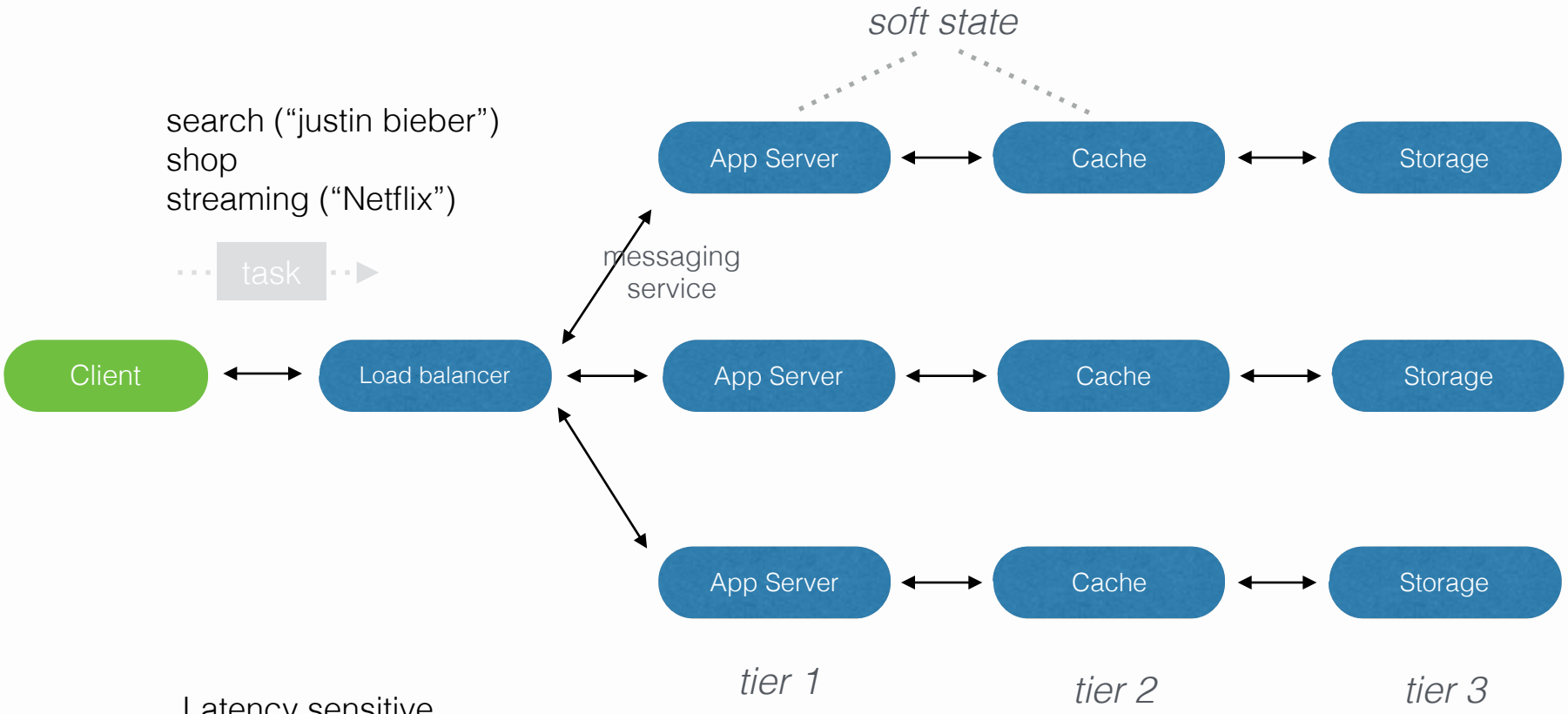


- Read from closest server
- Write to California
- Other servers update cache every 15 minutes
- After write: read from CA for 15 minutes

Slide: Marc Shapiro


Easy to program
Low performance
Total order reads & writes
Real time
Strict Serialisability
Total order reads
Real Time
Snapshot Isolation
Wait-Free Queries
Minimum
Commitment Sync
Total order reads & writes
Serialisability
Genuine Partial Replication
Forward Freshness
Parallel SI
Relaxed read ordering
Total order reads
Update Serialisability
Wait-Free Queries
Decoupled read/writes
Total order writes
Non-Monotonic SI
Minimum Commitment Sync
Relaxed read ordering
Genuine Partial Replication
Forward Freshness
Total Order Writes
Not Available
Transactional Causal Cons.
Causal Consistency
Strong Eventual Consistency
Hard to program
High performance
[Decomposing consistency]
Slide: Marc Shapiro

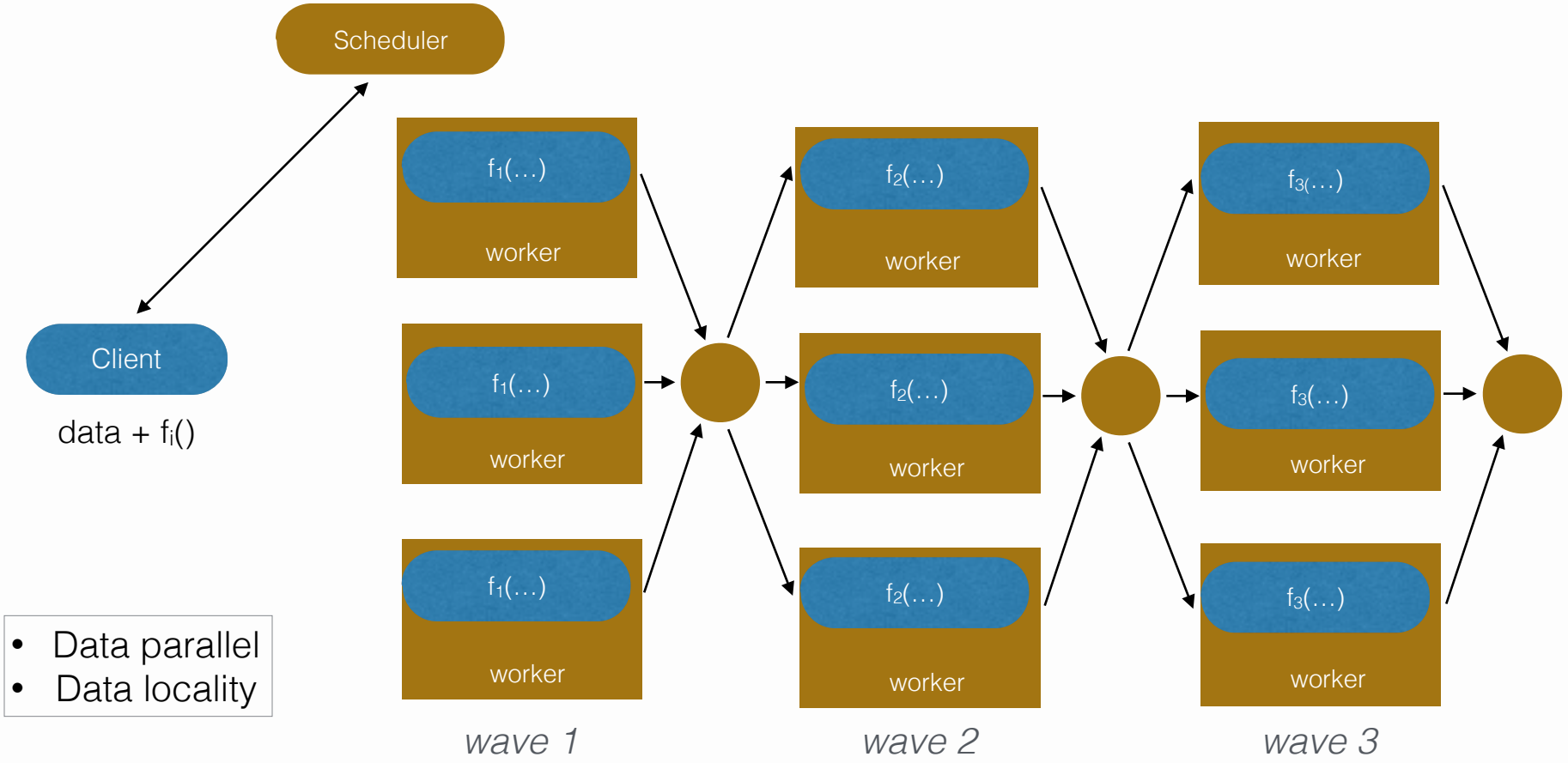
Service Job



Can be a set of *microservices*

Batch Job

Frameworks: MapReduce, Spark, Dryad, Pregel



- Data parallel
- Data locality

Optimize for throughput

MPI Jobs

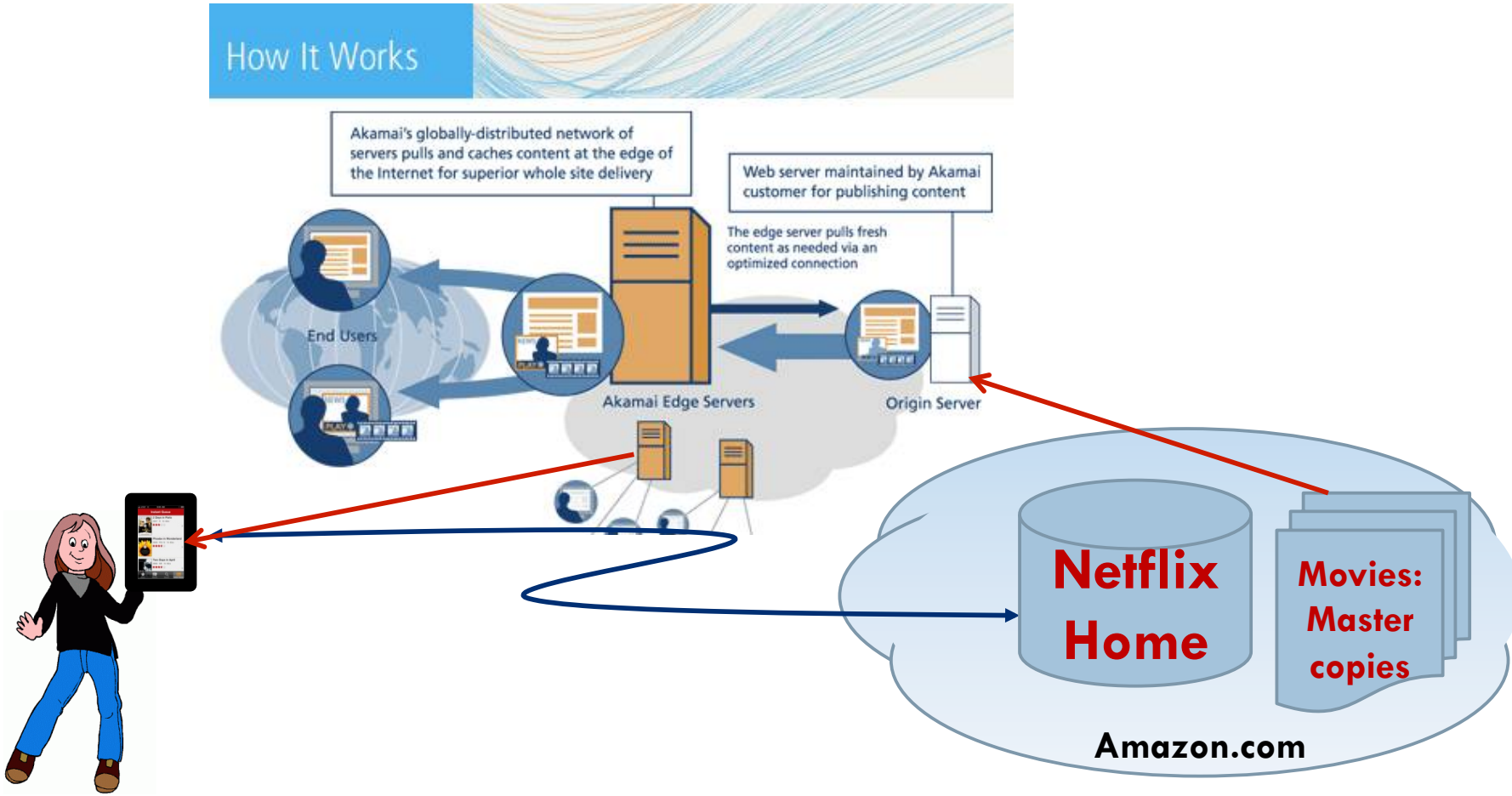
- Can be either service or batch jobs
- MPI = Message Passing Interface
  - Standardized and portable message-passing system
- Does not automatically scale well, static in nature.
- High Performance Computing (HPC) / Grid computing
  - Legacy type applications

Cloud Applications

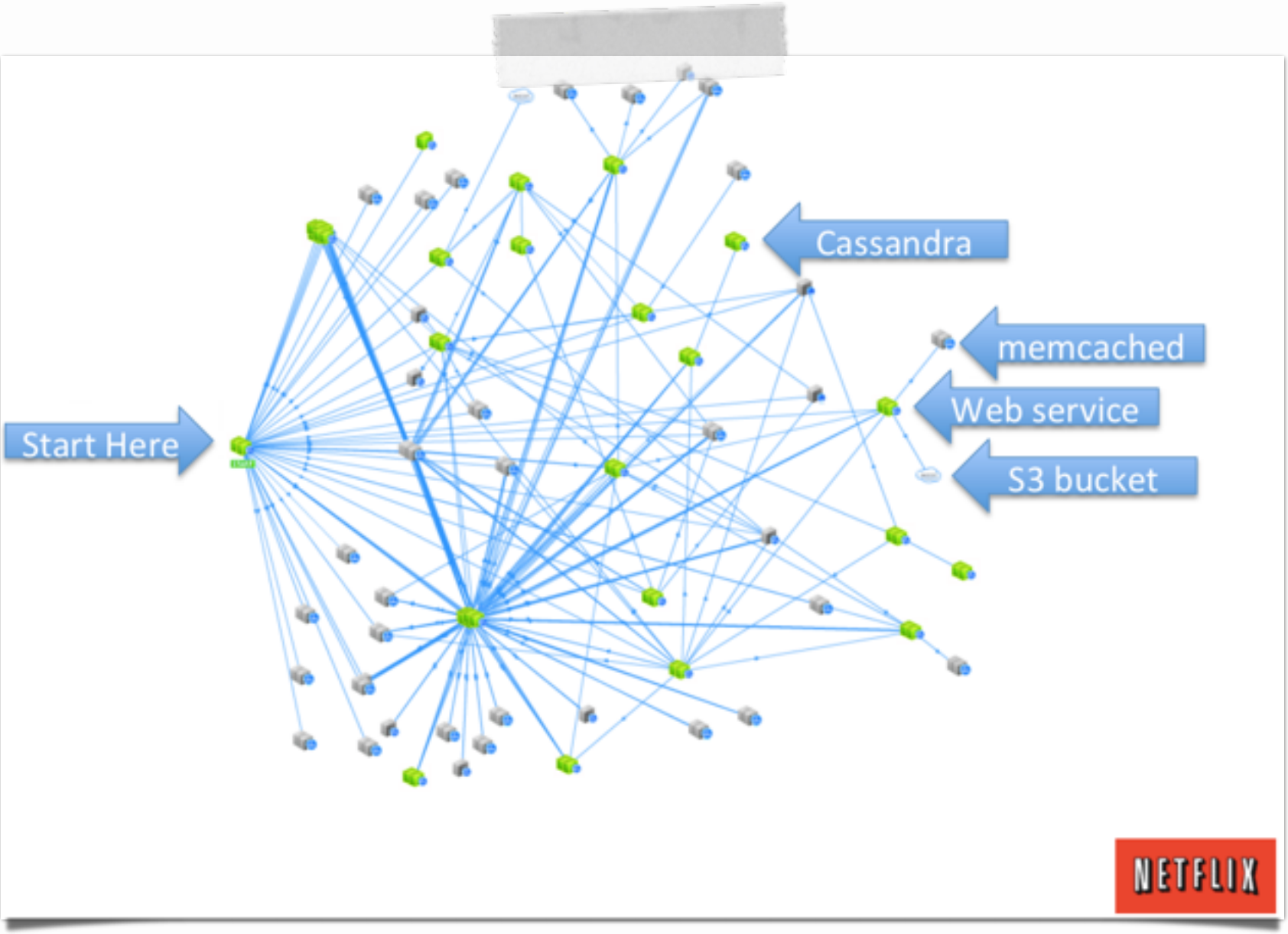
- Service Jobs
  - Duration ~30 days*
  - 12-20 min* inter arrival time
- Batch Jobs
  - Duration ~2-15 minutes*
  - 4-7 seconds* inter arrival time

**80th percentile, from Omega presentation*

Netflix 1.0



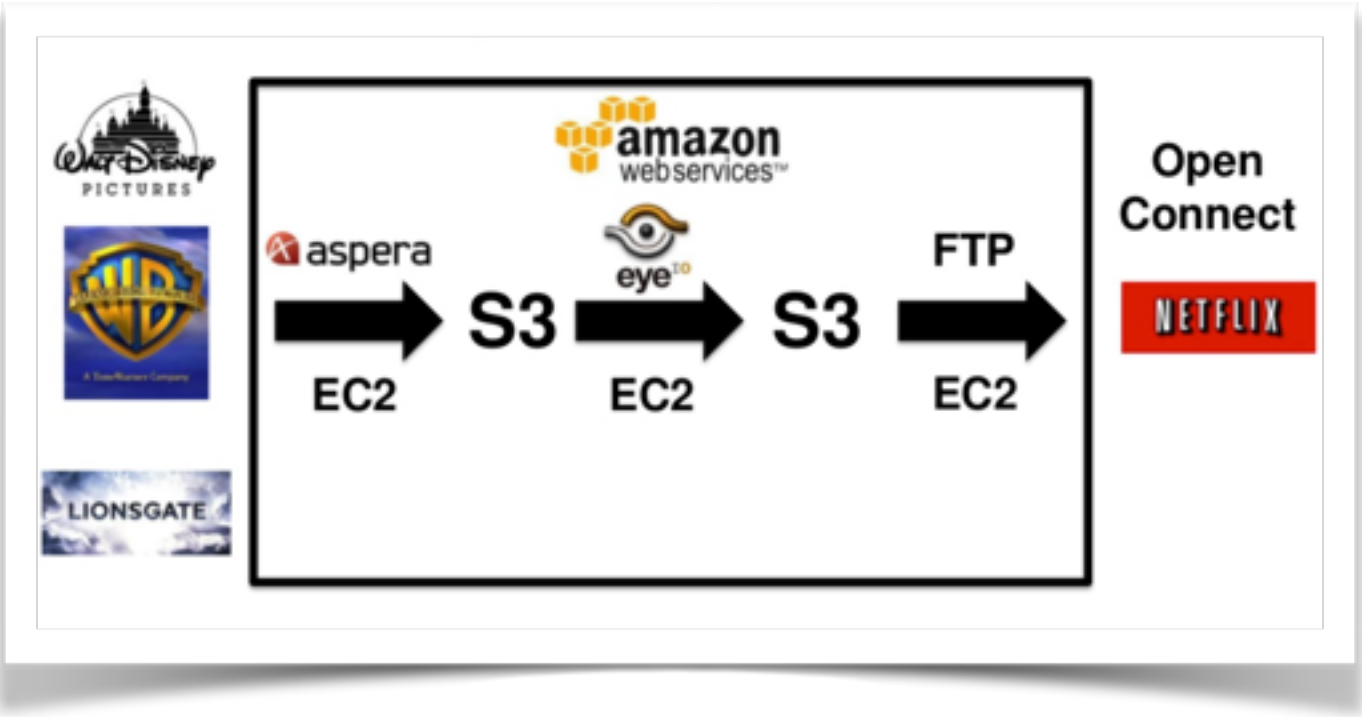
Source: Ken Birman



source: Adrian Cockcroft

Netflix Media Management

- 5 Regional catalogs
- 4 formats supported (VC-1, 3 H.264) at multiple bit rates
- Uses 6000 virtual machines (EC2) for transcoding
- Petabytes of storage
  - Stores the originals at “cold storage” (Amazon Glacier)

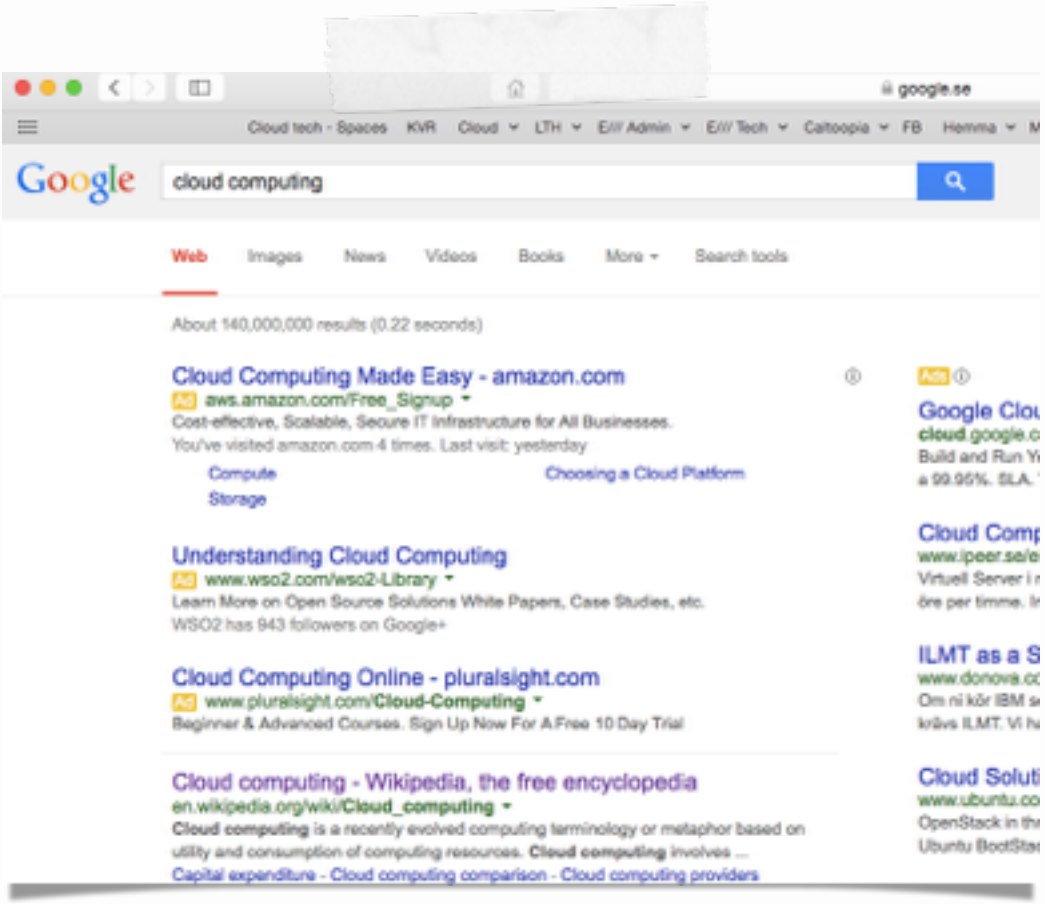


How does Google do it?

‘instant search’ really is instant most of the time

What does it look like under the hood?

We don’t know, but let’s guess a little



Google Search

- In 2003 a single query on Google reads 100 of MB, and consumes tens of billions of CPU cycles
  - And that was before ‘instant search’
- Design factors: energy efficiency & price-performance
  - Power & cooling are limiting factors for DCs
  - Easy parallelization: Different queries can run on different processors, and the a single query can use multiple processors. => commodity hardware will do just fine

- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

Getting there

```
lanhost496:~ johan$  dig +noall +answer google.com
google.com.          1    IN   A    173.194.78.102
google.com.          1    IN   A    173.194.78.100
google.com.          1    IN   A    173.194.78.138
google.com.          1    IN   A    173.194.78.101
google.com.          1    IN   A    173.194.78.113
google.com.          1    IN   A    173.194.78.139
lanhost496:~ johan$
lanhost496:~ johan$  dig +noall +answer google.com
google.com.          300  IN   A    173.194.65.113
google.com.          300  IN   A    173.194.65.101
google.com.          300  IN   A    173.194.65.102
google.com.          300  IN   A    173.194.65.138
google.com.          300  IN   A    173.194.65.100
google.com.          300  IN   A    173.194.65.139
lanhost496:~ johan$  dig +noall +answer google.com
google.com.          299  IN   A    173.194.65.139
google.com.          299  IN   A    173.194.65.100
google.com.          299  IN   A    173.194.65.138
google.com.          299  IN   A    173.194.65.102
google.com.          299  IN   A    173.194.65.101
google.com.          299  IN   A    173.194.65.113
```

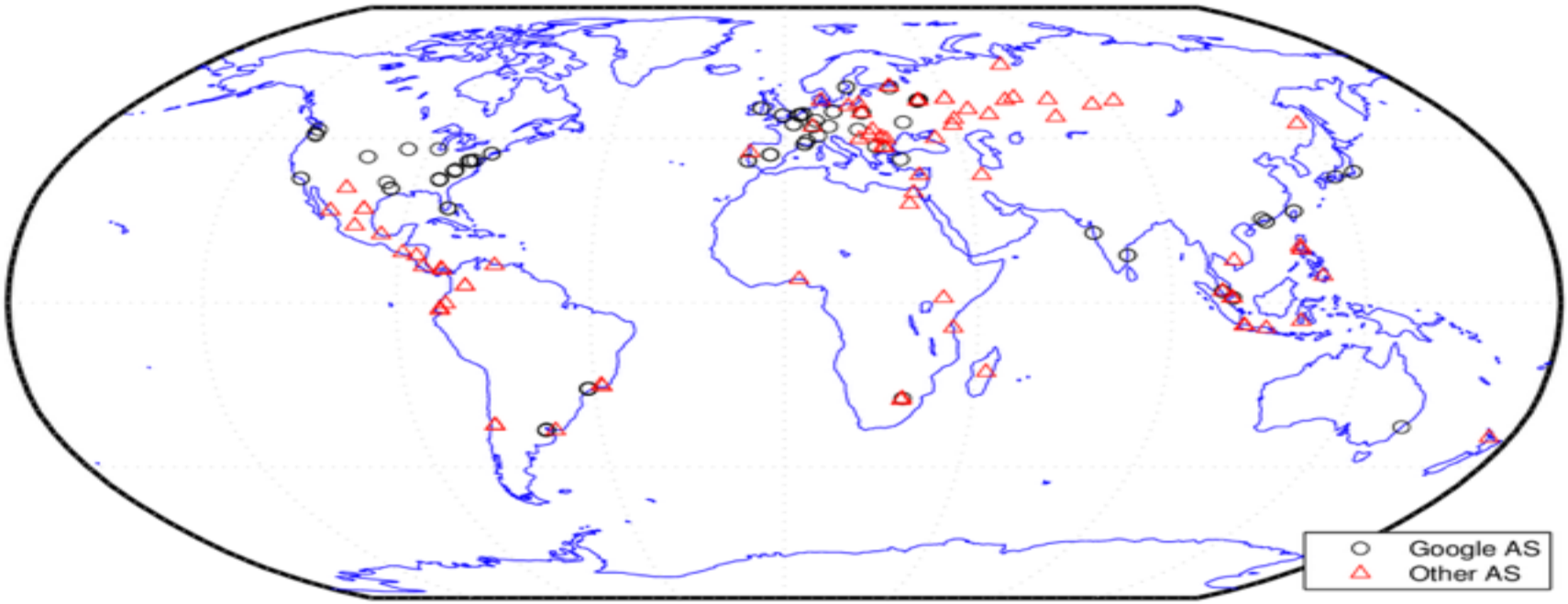
- DNS load balancing
- TTL < 5 minutes
- 500+ IP addresses for ‘Search’

The Datacenters

- 36+ data centers worldwide (probably lots more)
- Estimated 900.000 machines in 2011
- Energy footprint 2010
  - Less than 1% of all energy for data centers
- All IPv6 inside
- Zero trust inside
- Everything's compressed
- Everything's cached

http://www.datacenterknowledge.com/archives/2012/05/15/google-data-center-faq/
http://www.koomey.com/post/8323374335

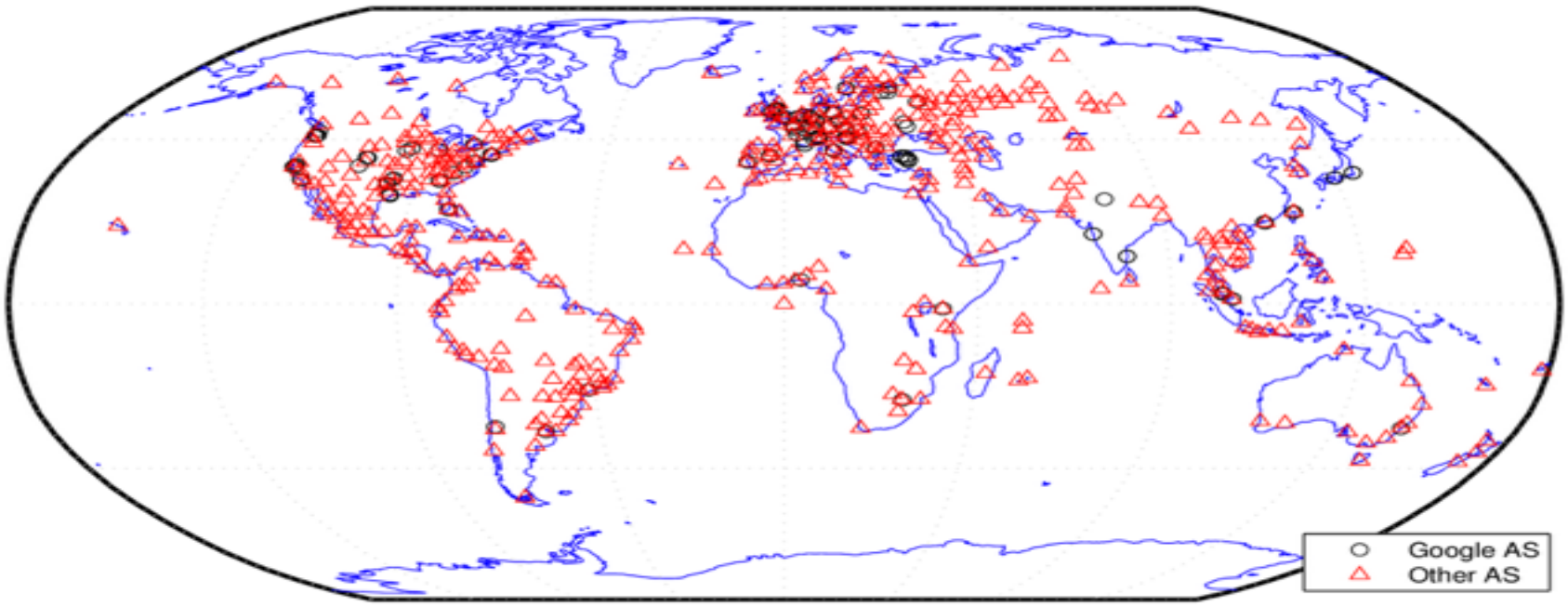
Google Servers Oct'12



- 200 sites in 60 countries and 100 ASes
- Large % are in Google's AS

source: http://nsl.cs.usc.edu/Talks/?action=download&upname=calder_imc13_mapping.pdf

Google Servers Oct'13



- 1400 (7x) sites in 130+ (2.3x) countries, 800+ (8x) ASes
- Growth is outside Google's AS

source: http://mappinggoogle.cs.usc.edu/

Google Global Cache (GGC)

- GGC was designed for eyeball-heavy networks with greater than 300Mbps peak Google traffic.
- Cache hit rate will vary by network based on the number of users served by the cache, their usage patterns, the size and type of GGC node, and the number of nodes deployed in your network. We have typically seen hit rates of 70% to 90%.
- Typically, a majority of the traffic routed through the GGC node is static content such as YouTube videos and Android Market downloads. Other Google web services, such as Google Search, may also be proxied and/or cached based on a number of factors, including legal requirements, available capacity and expected improvement in performance for end users. These services could include (but are not limited to):
  - YouTube
  - Google Search
  - Google Plus
  - Google Maps (including map tiles and street view)
  - Google Earth
  - Google Docs
  - Google Scholar
  - Google News
  - Android Market
  - Picasa Web Albums
  - DoubleClick by Google

Source https://peering.google.com/about/faq.html

The Google Network(s)

- External network
  - Heterogenous
- Internal network - B4
  - Few nodes, well known traffic
  - More traffic than external
- Traffic engineering using SDN
  - 2-3x efficiency improvement

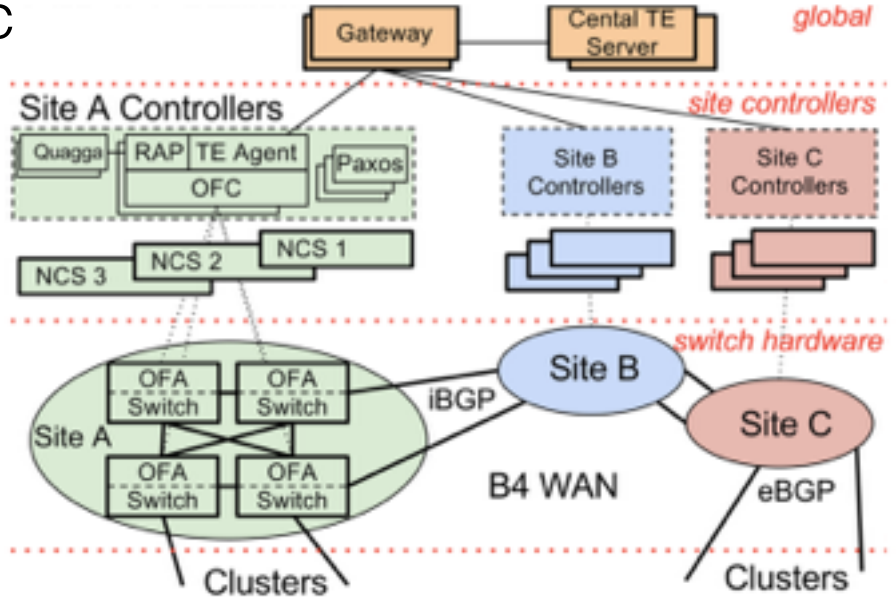


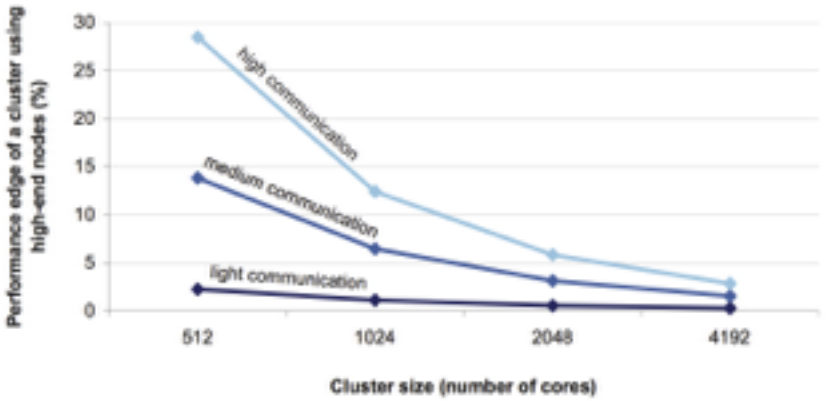
Figure 2: B4 architecture overview.

Google

- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

The Servers

	HP INTEGRITY SUPERDOME-ITANIUM2	HP PROLIANT ML350 G5
Processor	64 sockets, 128 cores (dual-threaded), 1.6 GHz Itanium2, 12 MB last-level cache	1 socket, quad-core, 2.66 GHz X5355 CPU, 8 MB last-level cache
Memory	2,048 GB	24 GB
Disk storage	320,974 GB, 7,056 drives	3,961 GB, 105 drives
TPC-C price/performance	$2.93/tpmC	$0.73/tpmC
price/performance (server HW only)	$1.28/transactions per minute	$0.10/transactions per minute
Price/performance (server HW only) (no discounts)	$2.39/transactions per minute	$0.12/transactions per minute



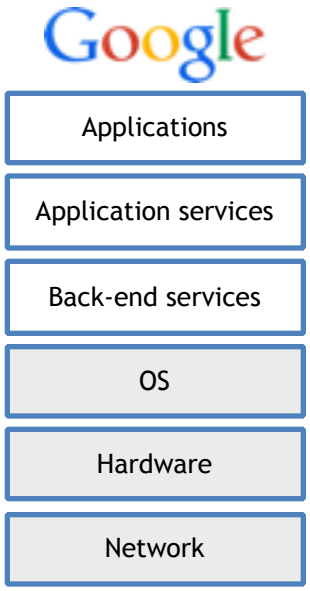
- Clusters of unreliable commodity hardware, not server grade
- Focus on price
- Everything is automized, minimize staff

Google: We're One of the World's Largest Hardware Makers

BY ROBERT MCMILLAN 06.22.12 | 3:25 PM | PERMALINK

"Google spent about $3.4 billion on capital expenditures last year, and a large chunk of that went to the x86 servers that run its data centers. To put that in perspective, number-four X86 server vendor Fujitsu sold $1.3 billion worth of servers in 2011, according to research firm Gartner."

http://www.wired.com/2012/06/google_makes_servers/



The Node OS

- Linux - Started with Red Hat 6.2 (2000) and moved to 7.1(2001) (2.6.X kernel) and stayed with that until 2013
  - Updates using file level sync from gold master (rsync-ish)
  - Kernel stayed the same besides security patches
- Now home-cooked Linux by called *ProdNG* based on Debian
- All software is in-house or open-source
  - No licenses makes scaling cheap

source: http://marc.merlins.org/linux/talks/ProdNG-LISA/

Google

Applications

Application services

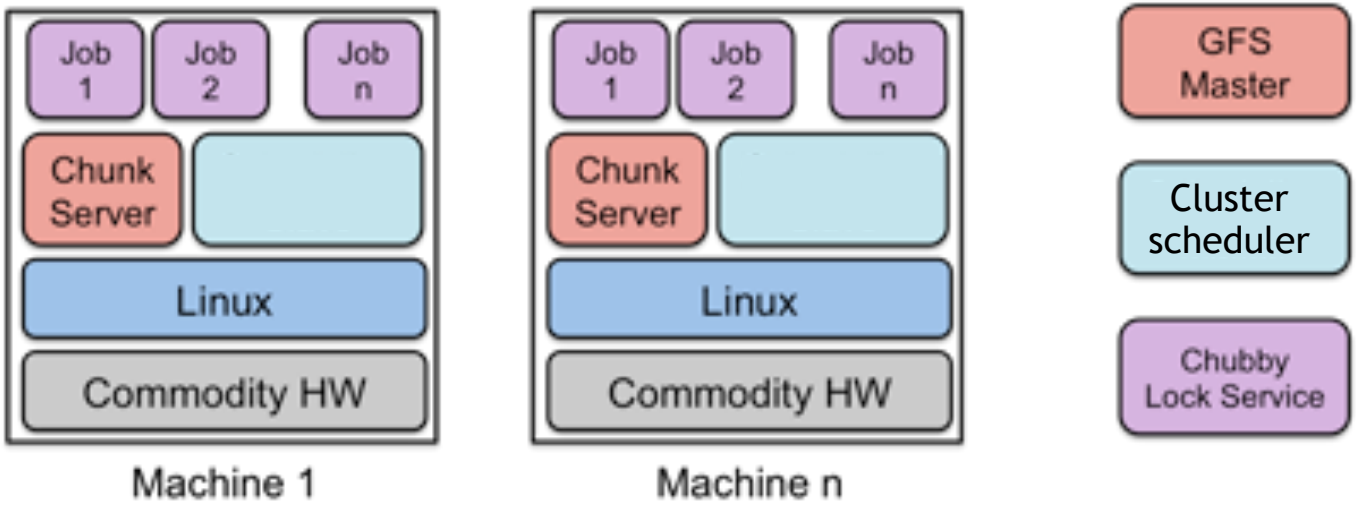
Back-end services

OS

Hardware

Network

The Datacenter OS

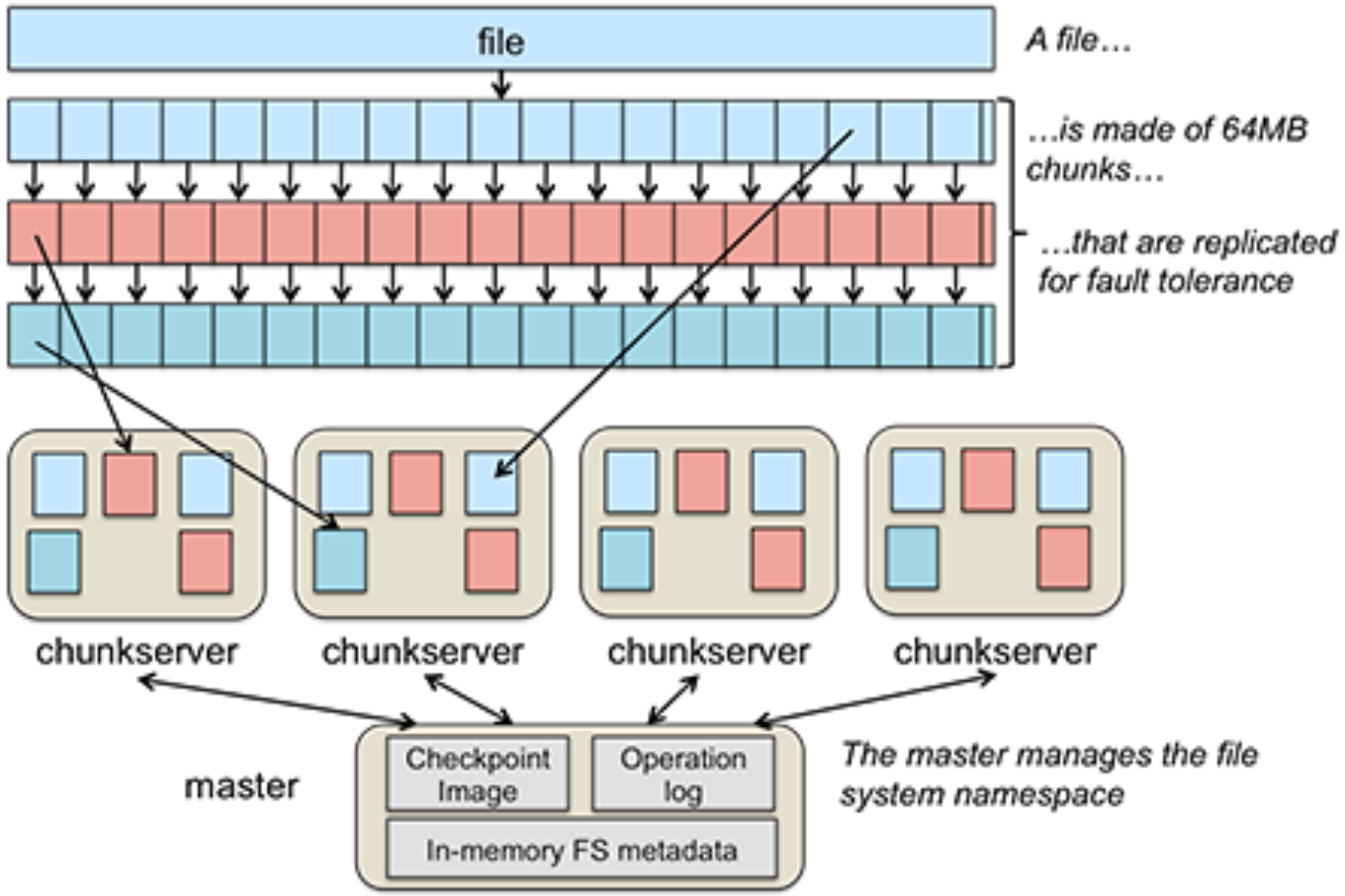


source: P. Krzyzanowski cs.rutgers.edu

Google

- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

Distributed File System



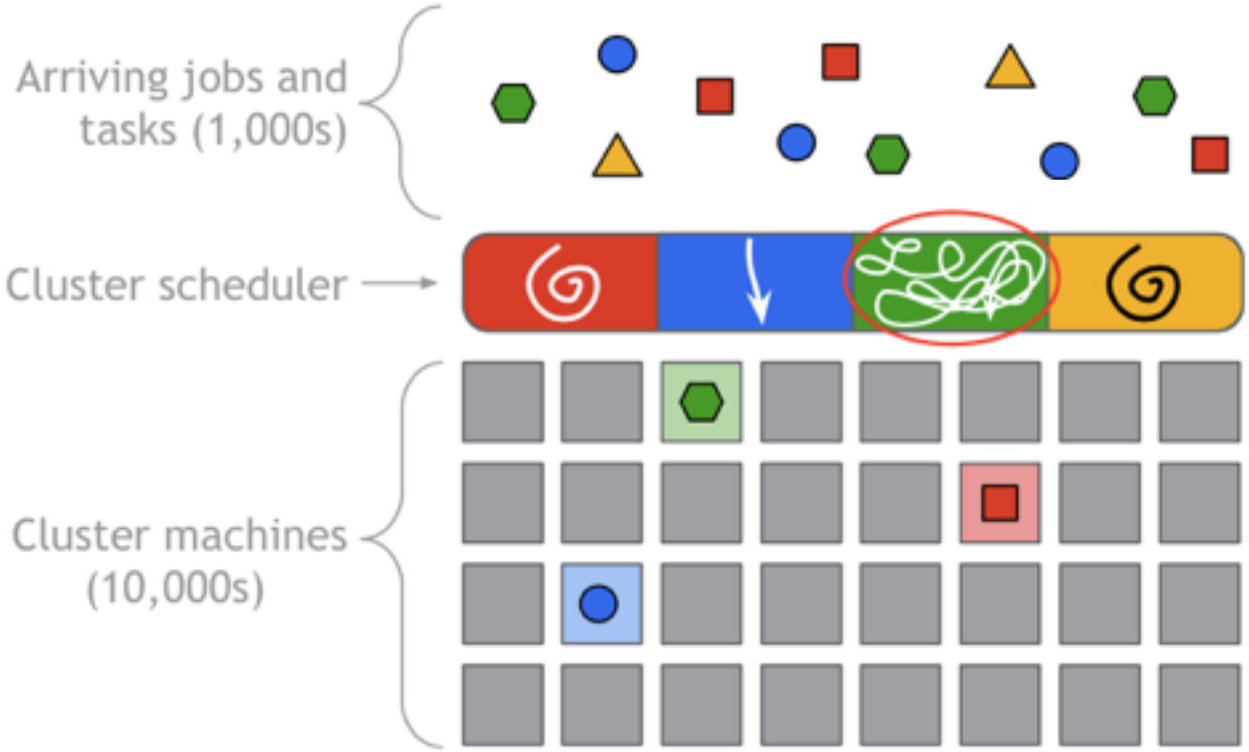
Designed for writes!

source: P. Krzyzanowski cs.rutgers.edu

Google

- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

The Cluster Scheduler

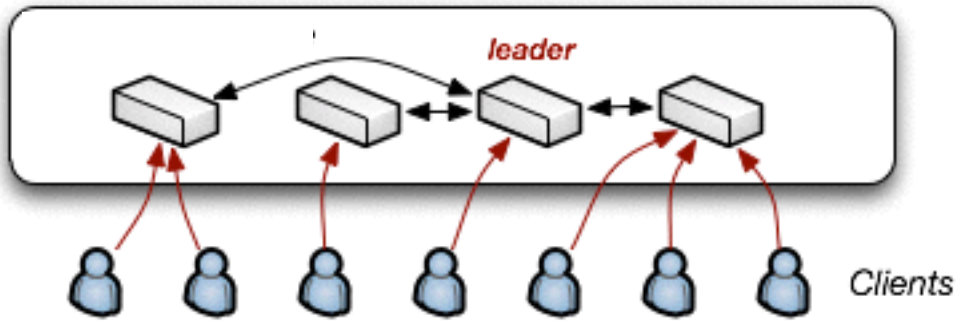


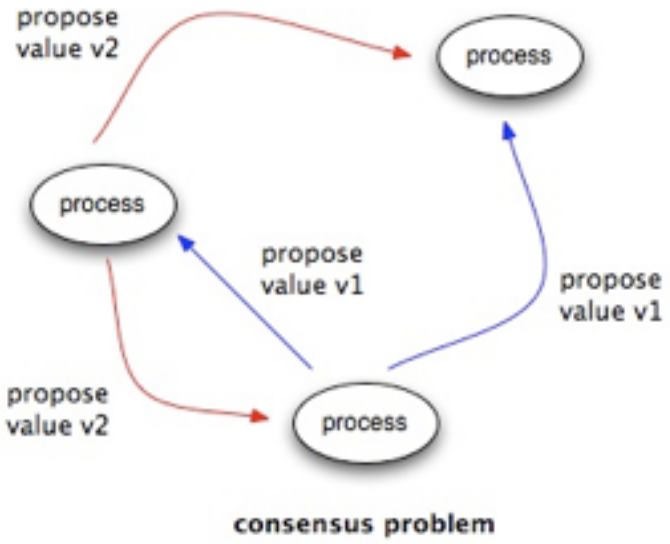
'I PREFER TO CALL IT THE SYSTEM THAT WILL NOT BE NAMED.'

— JOHN WILKES

Lock Service

- Chubby lock server
  - Implementation of Paxos
  - Cluster of 5
- Locks with varying levels of durability
- Leadership election
- Store configuration data

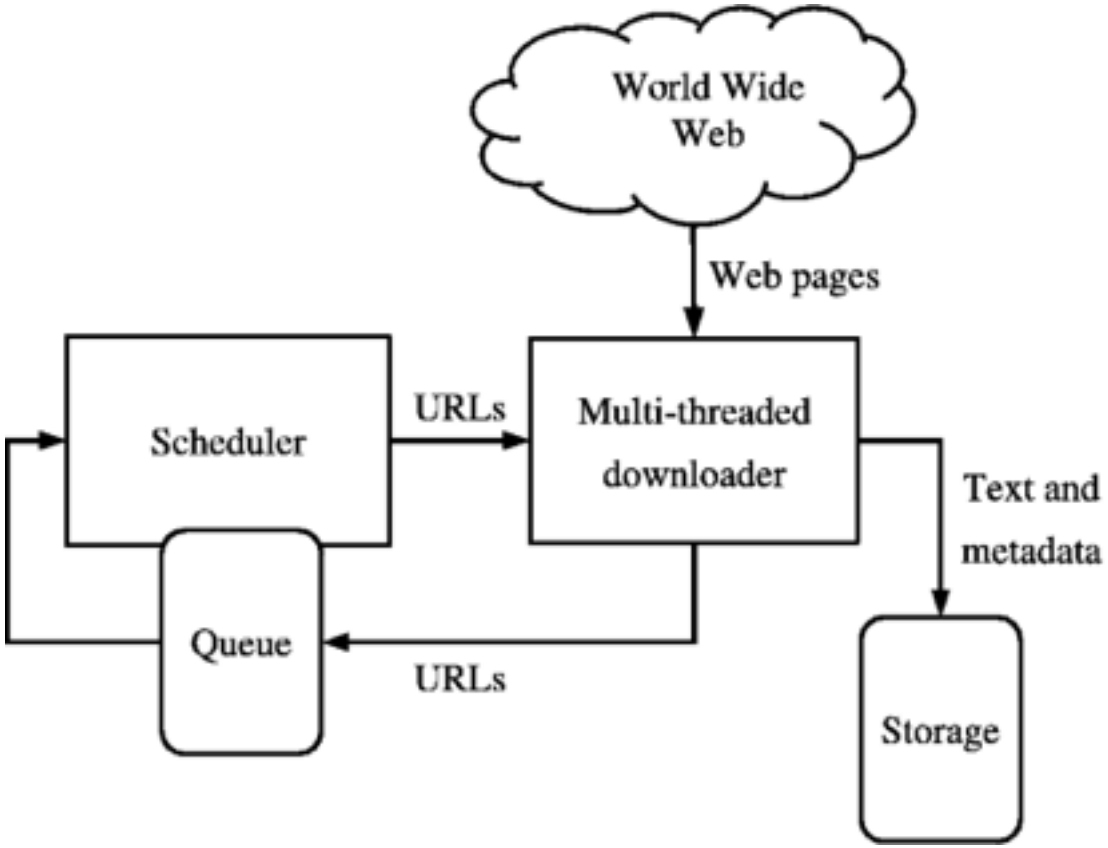




Google

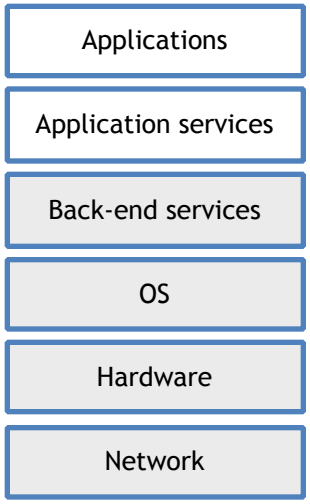
- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

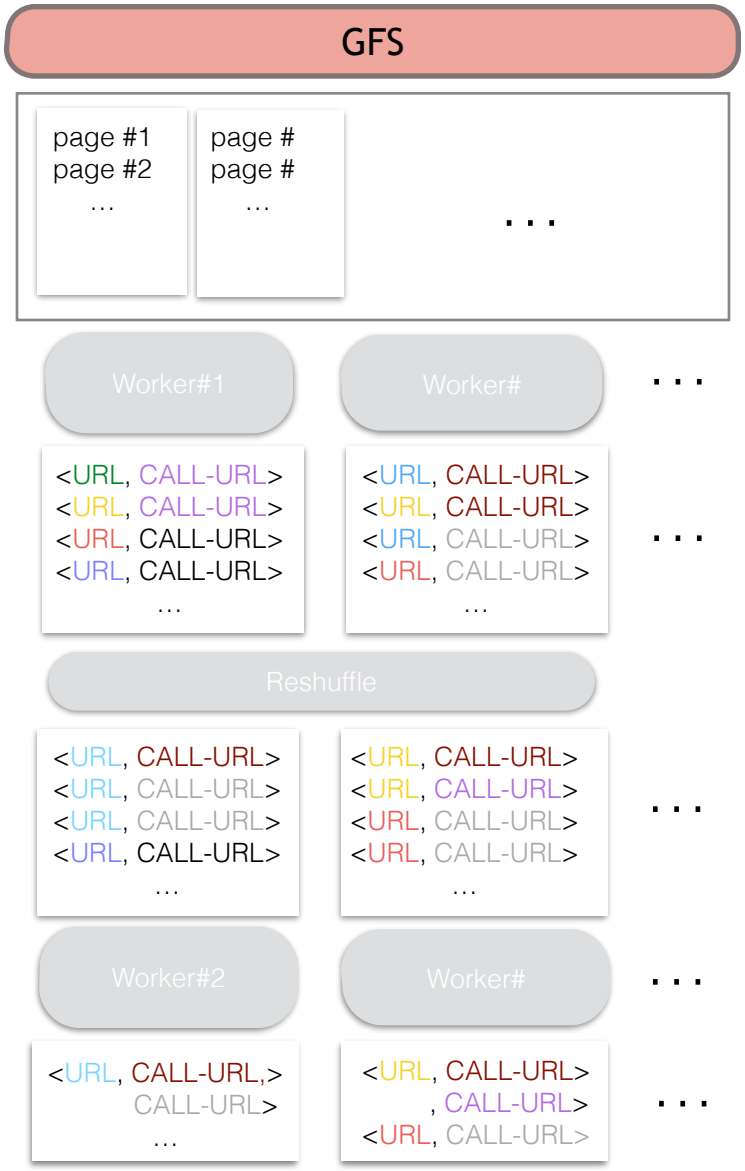
Collecting the Pages

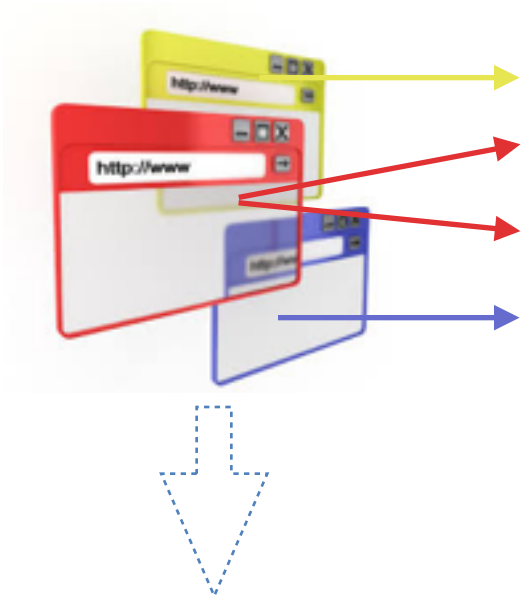


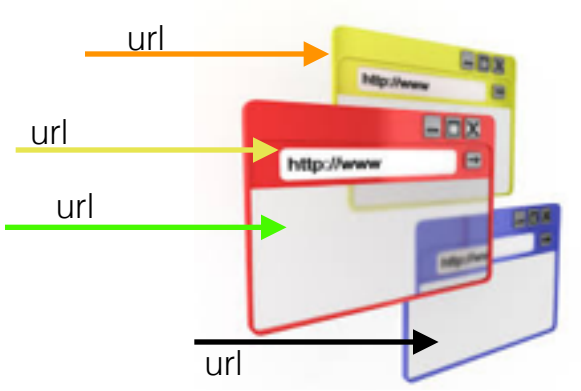
The task of the GoogleBot

Link Reversal









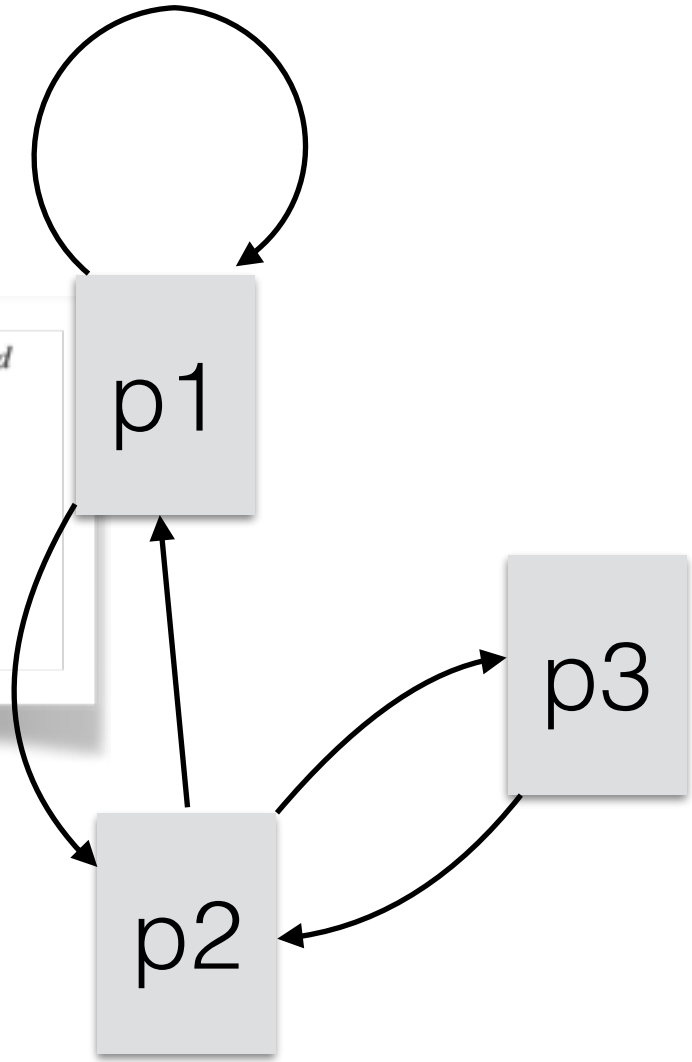
Input is all the web pages in the world stored on GFS chunks

Rank the Pages

We assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:

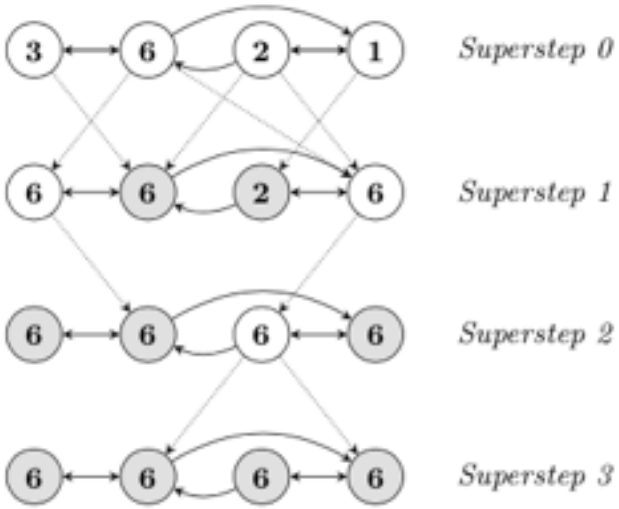
$$PR(A) = (1-d) + d\,(PR(T1)/C(T1) + \ldots + PR(Tn)/C(Tn))$$

Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages' PageRanks will be one.



Compute Page Rank

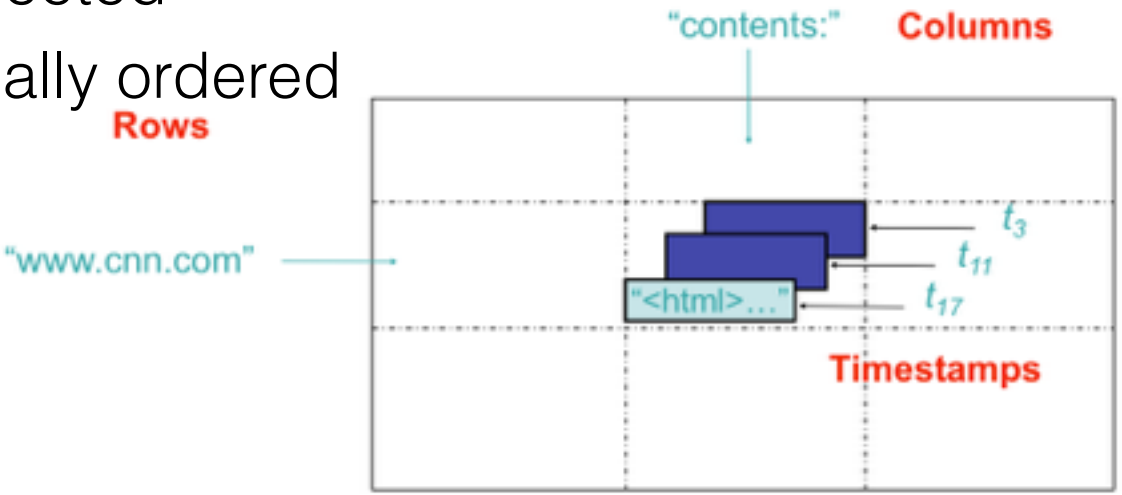
- Recreate the web in Pregel
  - A google graph language
- Pages as vertices
- Links as edges
- Stepwise iteration

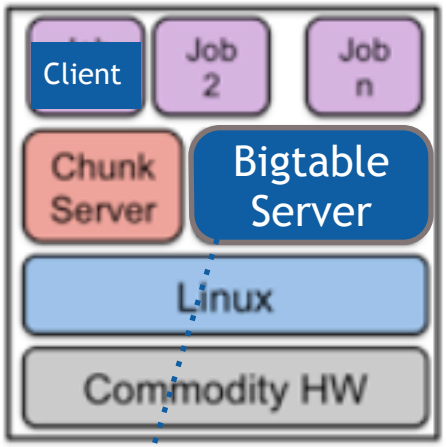


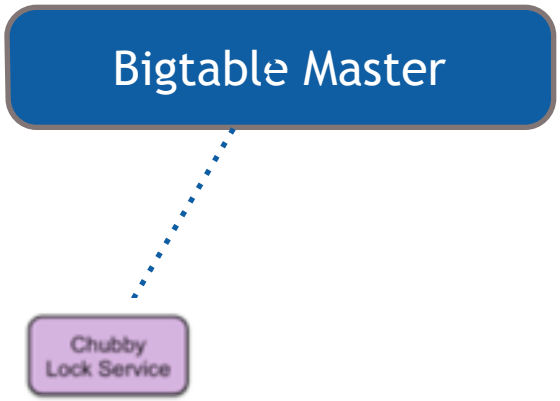

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

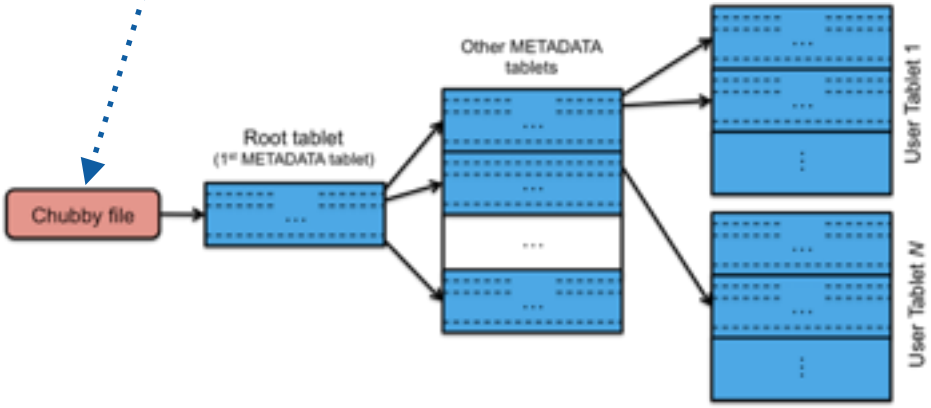
BigTable

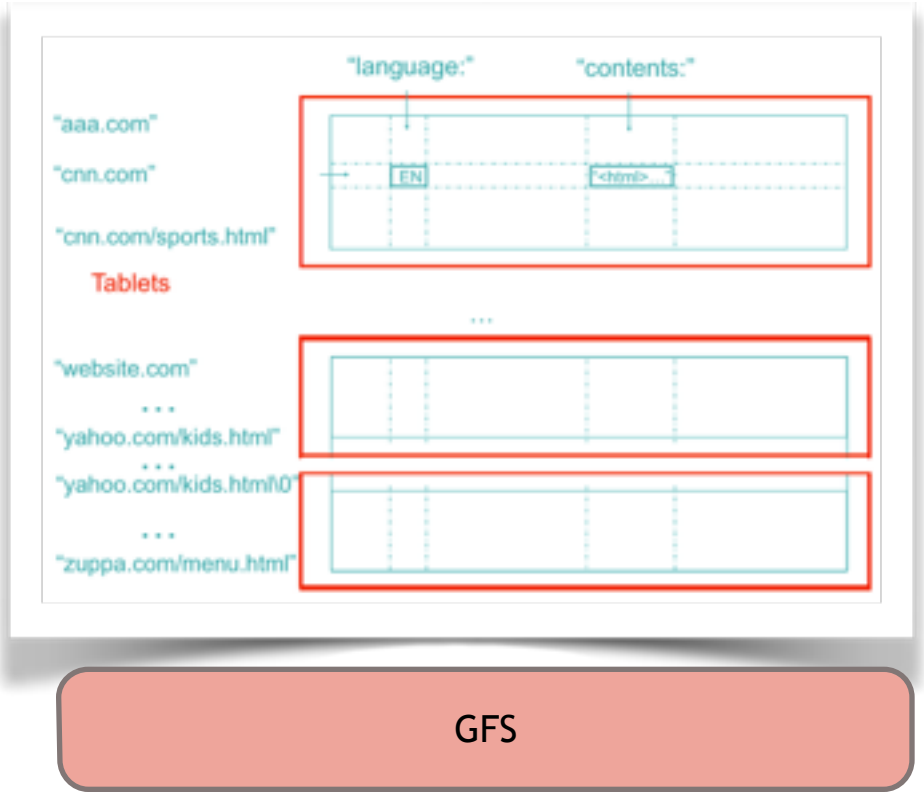
- Sparse distributed database
- A big map <Row, Column, Timestamp> ⇒ value
  - Arbitrary “columns” on a row-by-row basis
  - No multirow transactions (reads or writes)
  - Per row mutation
- Garbage collected
- Row are lexically ordered











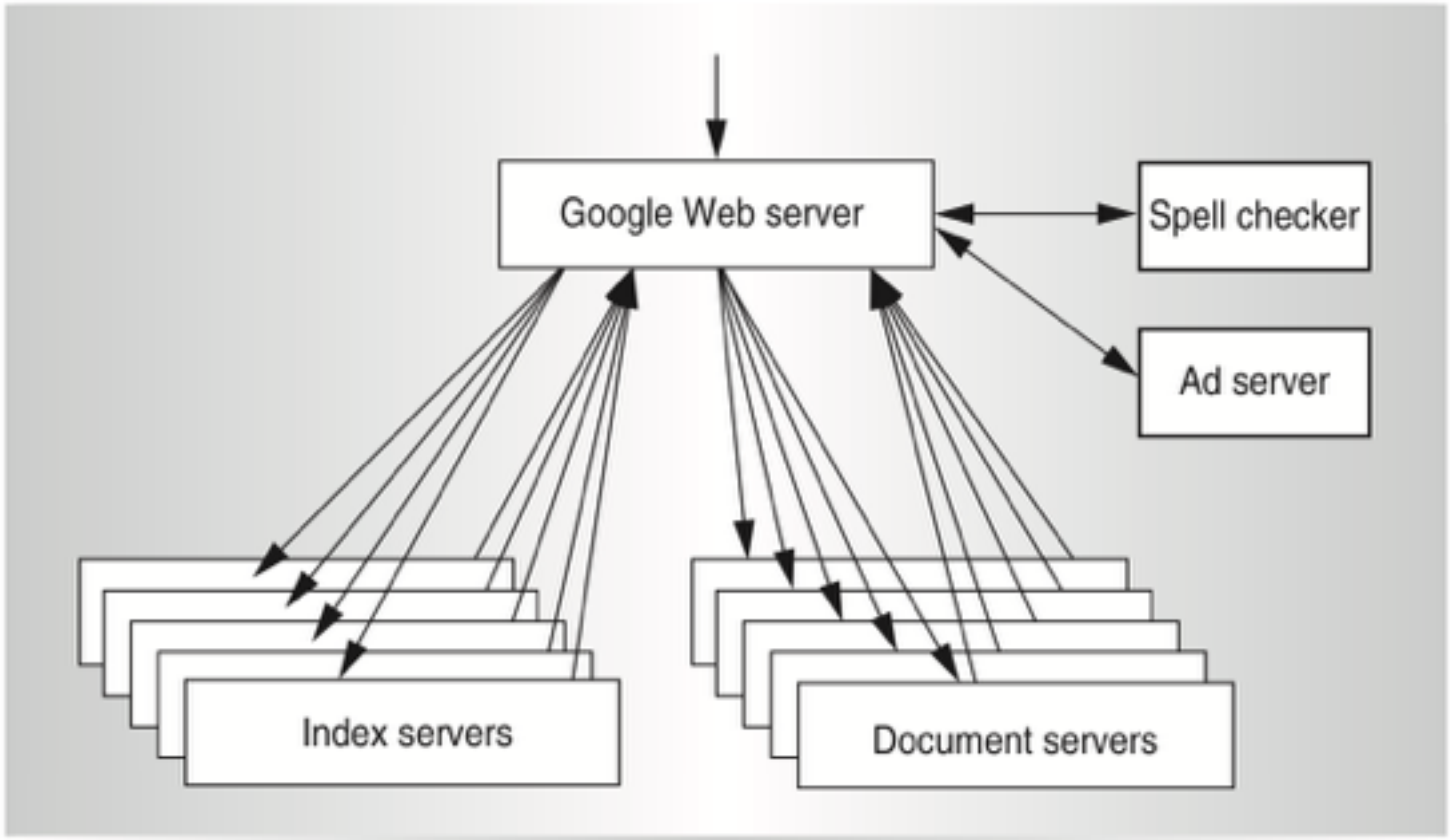
GFS

Servers can be added/removed dynamically

Google

- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

Search



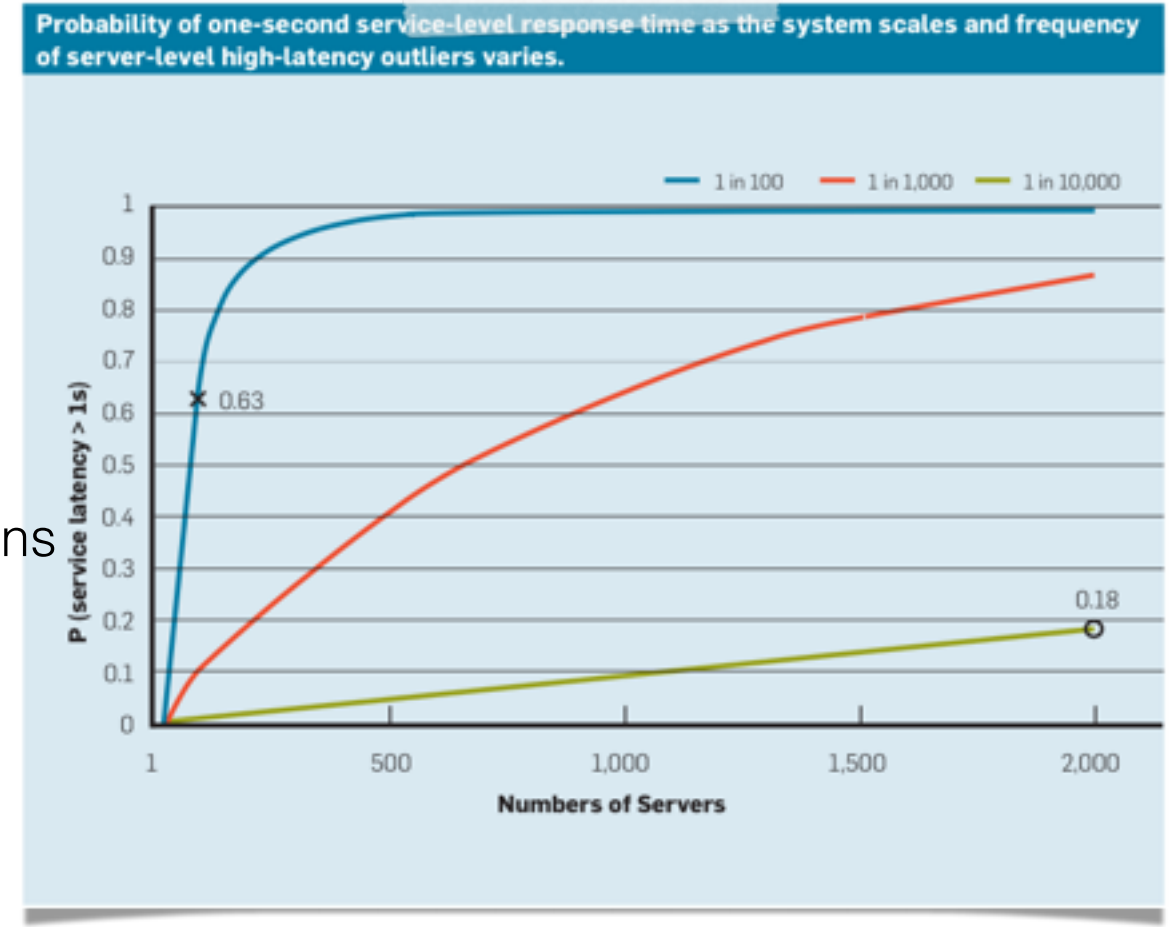
- A web search touches 50+ separate services, 1000s machines
- Everything is heavily cached

Google

- Applications
- Application services
- Back-end services
- OS
- Hardware
- Network

Latency

- Latency sources
  - Resource sharing
    - Local & cluster
  - SSD GC & compactations
- Server response
  - Typical: 10 ms
  - One out of *x*: 1000 ms
- Hedging



Building Our Own

- Linux & KVM
- HDFS GFS: Google File System
- Mesos instead of Borg
- Memcached
- Riak instead of Dynamo
- Cassandra instead of BigTable
- Zookeeper instead of Chubby
- Hadoop instead of MapReduce

Enabling Technologies

- Distributed computing
  - Consensus, time, locks, failure detection
- Networking
  - Traffic engineering, multi tenancy, migration
- Virtualization
  - Compute density, isolation, management/migration, security
- Storage
  - Replication, caching, DB, key-value, block storgae
- Datacenter OS & Application architecture
  - Microservices, utilization
- Programming models
  - Huge data sets, non reliable environment

Summary

- Utility computing not really new
  - Share the access time of mainframes (1960's)
- Build large datacenter and rent access to customers
  - Sun, IBM, HP, Intel, and many others built datacenters (1990's)
  - Then, Google & Amazon selling spare capacity
- We have a new usage model:
  - No initial investment; pay-as-you-go model
  - No long legal negotiations of SLAs and long-term contracts
  - Saving on CapEx and OpEx
- Centralization and sharing
  - Reduce the overall facility costs, power consumption, etc.
  - Statistical multiplexing —> Better utilization!
  - Economies of scale - Bigger is better
- Scalability
  - Cost associativity: 1,000 computers for 1 hour has the same price as 1 computer for 1,000 hours
  - Unbounded computational power and storage

Home Assignment #2