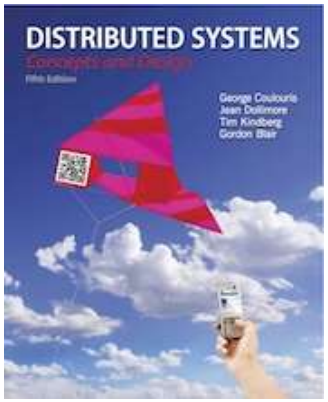
Google Case Study

Since 1998



From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2012

Agenda

1. Google
2. Platfom
    1. Servers
    2. Clusters
    3. Data centers
    4. Networks
3. Middleware
    1. Communication paradigms
    2. Data storage and coordination services
    3. Distributed computation services
4. Hadoop

1. Google

Google is a corporation specializing in Internet-related services and products

Most of its profits are derived from AdWords, an online advertising service that places advertising near the list of search results

Google was founded by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University

http://en.wikipedia.org/wiki/Google

Papers by googlers

Sergey Brin and Lawrence Page (1998), **The Anatomy of a Large-Scale Hypertextual Web Search Engine**, *Computer Networks and ISDN Systems*, Vol. 30, pp. 107-117.

Luiz André Barroso, Jeffrey Dean, and Urs Hölzle (2003), **Web Search for a Planet: The Google Cluster Architecture**, *IEEE Micro*, Vol. 23, No. 2, pp. 22-28.

Dennis Abts and Bob Felderman (2012), **A Guided Tour of Datacenter Networking**, *Communications of the ACM*, Vol. 55, No. 6, pp. 44-51.

Jeffrey Dean and Luiz André Barroso (2013), **The Tail at Scale**, *Communications of the ACM*, Vol. 56, No. 2, pp. 74-80.

Wojciech Golab, et al (2014), **Eventually Consistent: Not What You Were Expecting?** *Communications of the ACM*, Vol. 57, No. 3, pp. 38-44.

https://research.google/pubs/?area=distributed-systems-and-parallel-computing

Papers by googlers

Abhishek Verma, et al (2015), **Large-scale cluster management at Google with Borg**, *Proceedings of the 10th European Conference on Computer Systems (EuroSys'15)*, pp. 1-17, April 21–24, 2015, Bordeaux, France.

Arjun Singh, et al (2015), **Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network**, *SIGCOMM*, pp.183-197, August 17-21, London, UK.

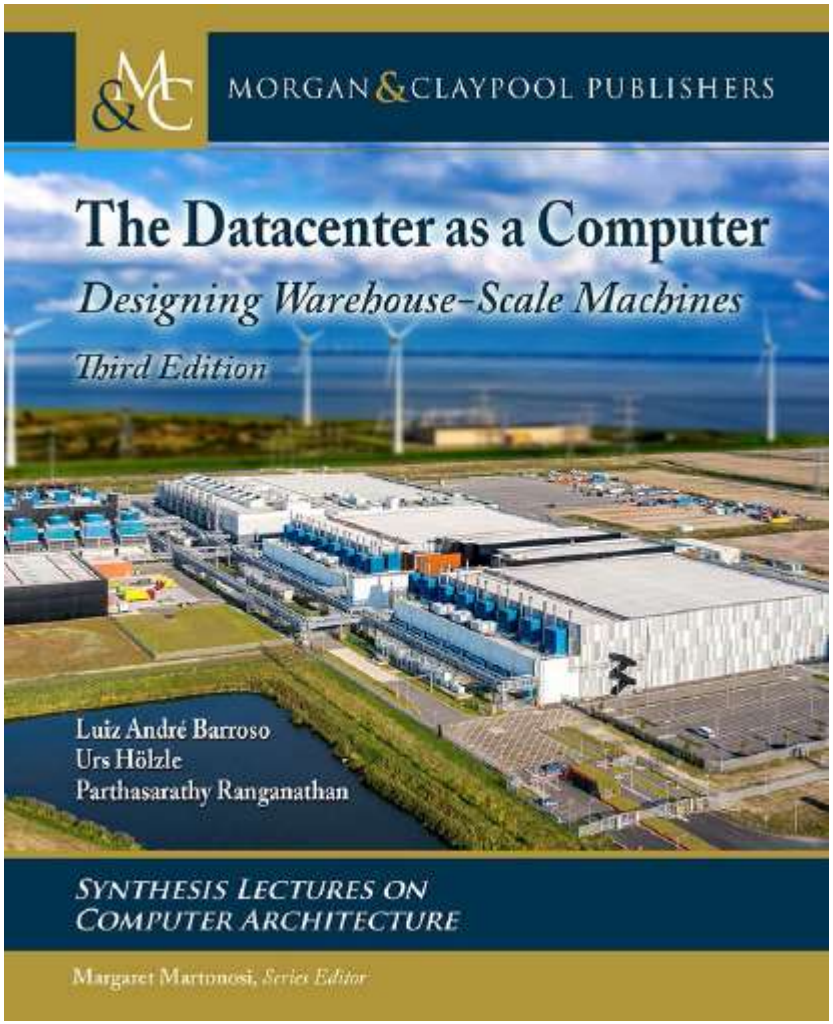
Alok Kumar, et al (2015), **BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing**, *SIGCOMM*, pp. 1-14, August 17-21, London, UK.

Erick Brewer (2017), **Spanner, TrueTime & the CAP Theorem**, *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pp. 251-264, October 8-10, Hollywood, CA.

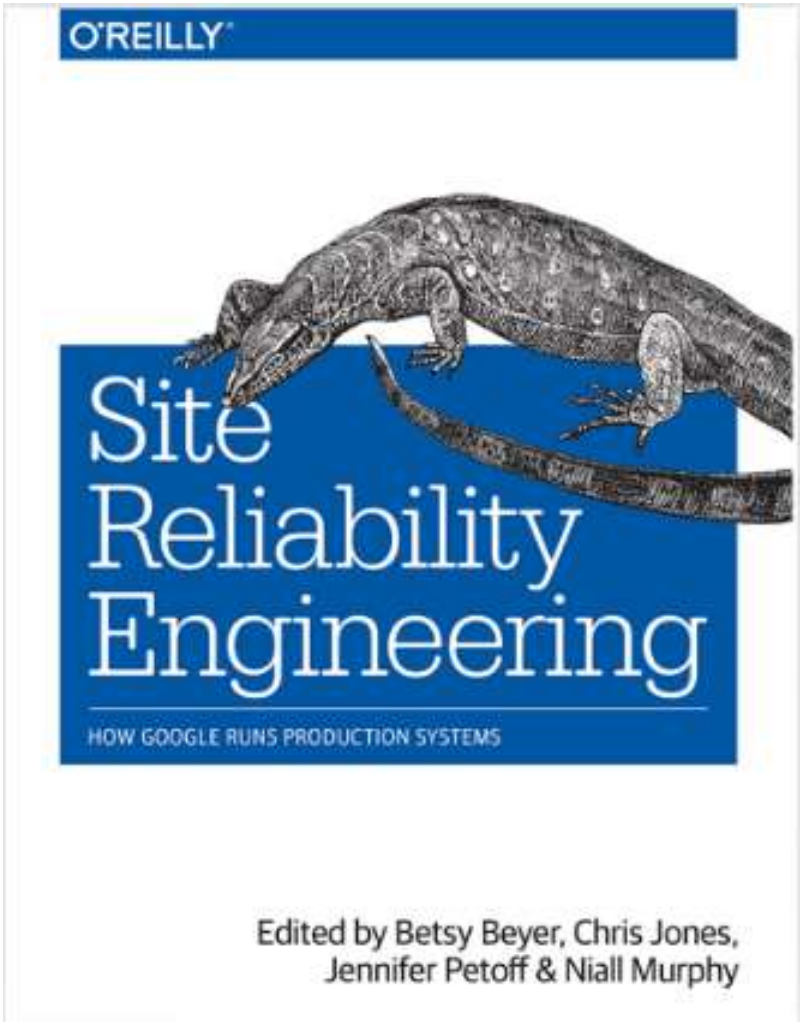
Googlish way: fruitful melding of advanced theory and technology.

http://highscalability.com/blog/2012/9/24/google-spanners-most-surprising-revelation-nosql-is-out-and.html

Books by googlers

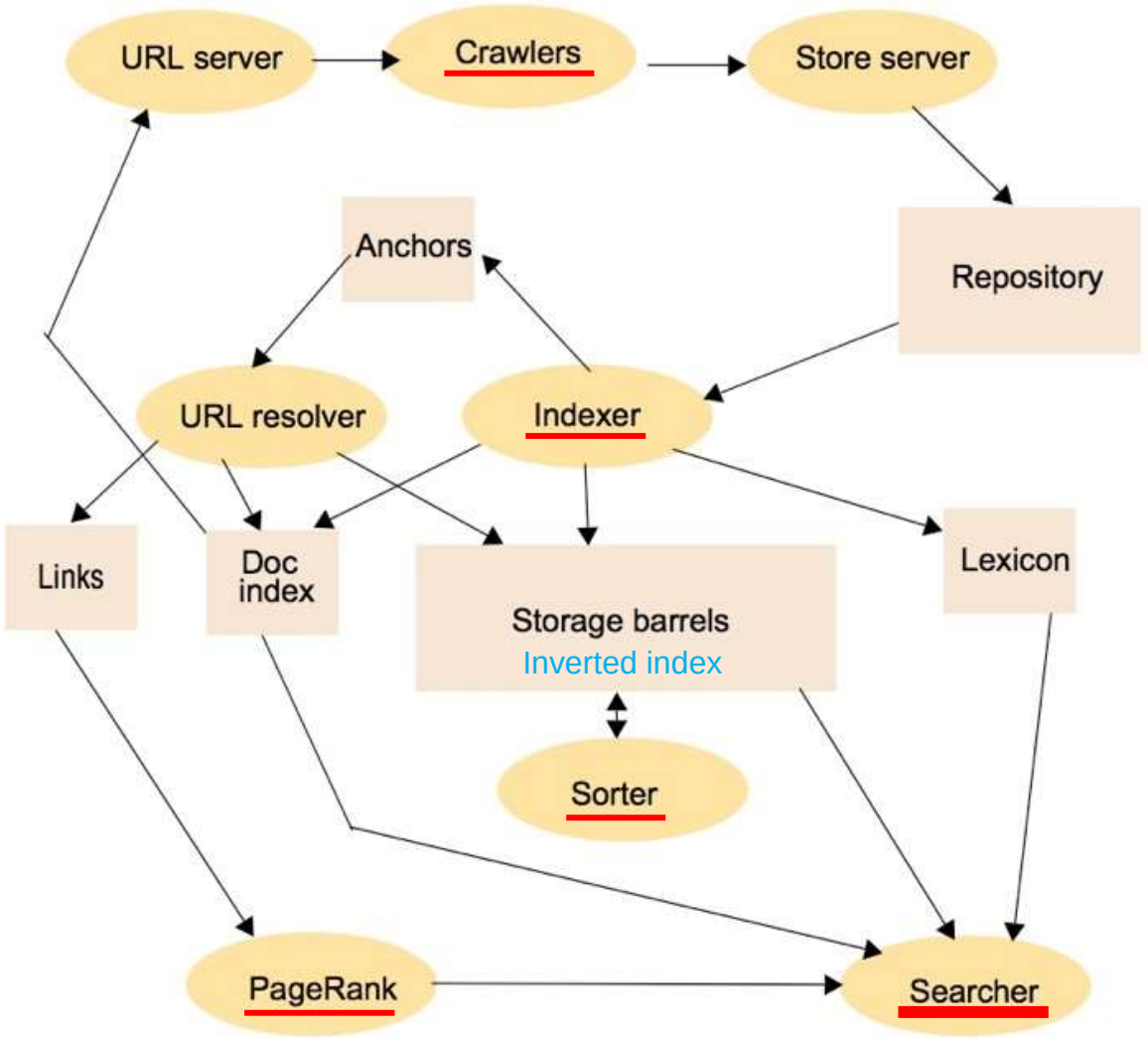


Books by googlers



https://landing.google.com/sre/sre-book/toc/index.html

Original search engine



Applications

Application	*Description*
Gmail	Mail system with messages hosted by Google but desktop-like message management.
Google Docs	Web-based office suite supporting shared editing of documents held on Google servers.
Google Sites	Wiki-like web sites with shared editing facilities.
Google Talk	Supports instant text messaging and Voice over IP.
Google Calendar	Web-based calendar with all data hosted on Google servers.
Google Wave	Collaboration tool integrating email, instant messaging, wikis and social networks.
Google News	Fully automated news aggregator site.
Google Maps	Scalable web-based world map including high-resolution imagery and unlimited user-generated overlays.
Google Earth	Scalable near-3D view of the globe with unlimited user-generated overlays.
Google App Engine	Google distributed infrastructure made available to outside parties as a service (platform as a service).

Googleplex

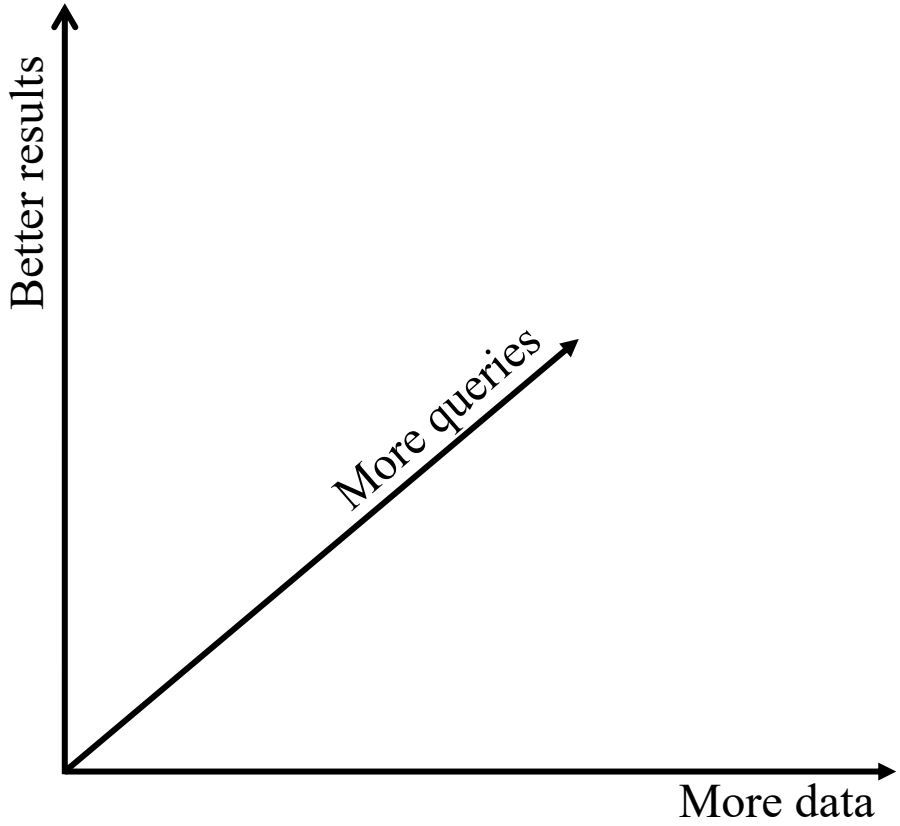
Store it all on “one big disk”

Process users’ requests on “one big CPU”

$$$$$

Doesn’t scale

System architecture requirements



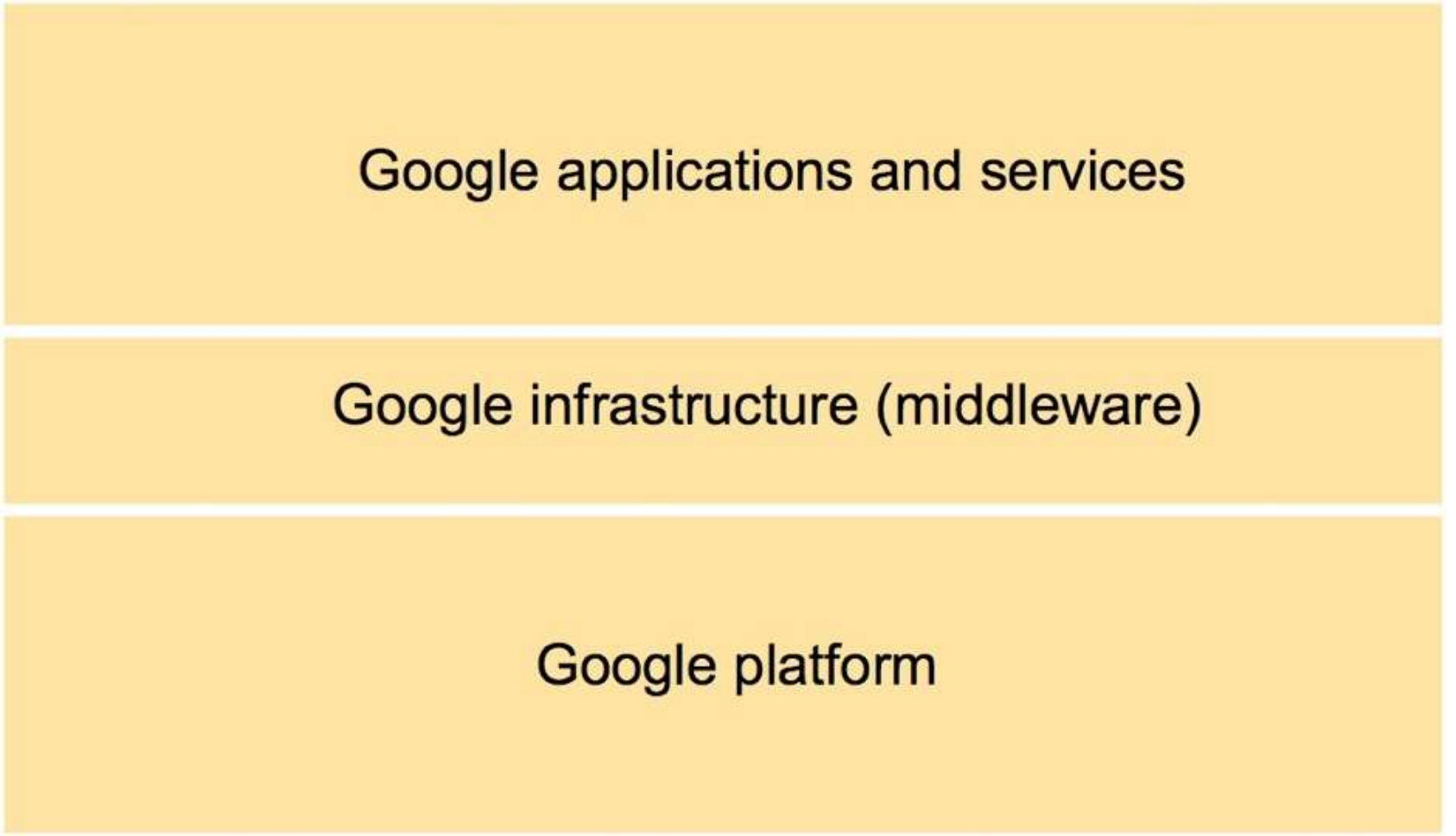
Availability

Performance

Scalability

Openness

Overall architecture



2. Platform

Servers

Clusters

Data centers

Network

2.1 Servers

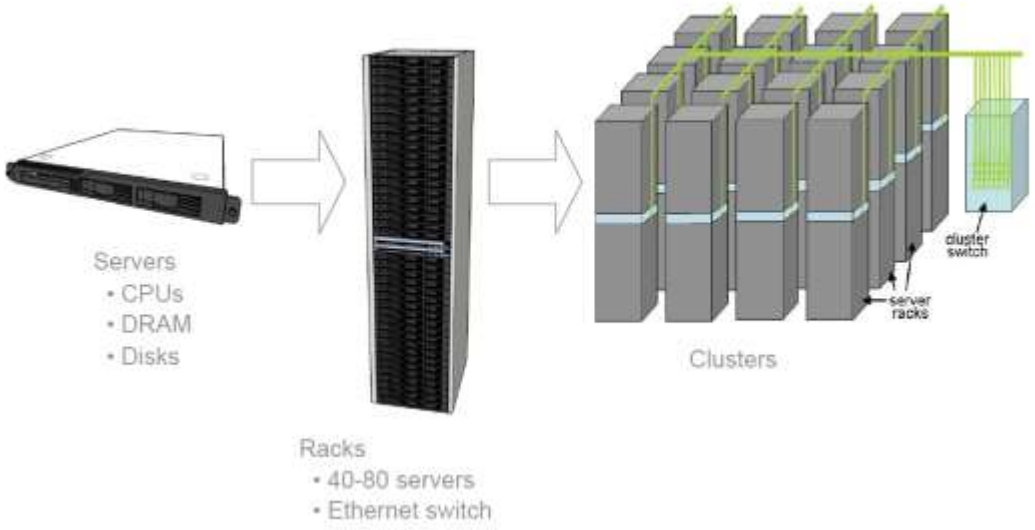
Google provides reliability in software rather than in server-class hardware, so commodity PCs can be used to build a high-end computing cluster at a low-end price

Google provides reliability by replicating services across many different machines and automatically detecting and handling failures

2.2 Clusters

To provide sufficient capacity to handle query traffic, the service consists of multiple clusters distributed in several data centers worldwide

Each cluster has around a few thousand machines, and the geographically distributed setup protects Google against catastrophic data center failures



2.3 Data centers

North America

Berkeley County, South Carolina

Council Bluffs, Iowa

The Dalles, Oregon

Douglas County, Georgia

Henderson, Nevada

Jackson County, Alabama

Lenoir, North Carolina

Loudoun County, Virginia

Mayes County, Oklahoma

Midlothian, Texas

Montgomery County, Tennessee

South America

Quilicura, Chile

Europe

Dublin, Ireland

Eemshaven, Netherlands

Fredericia, Denmark

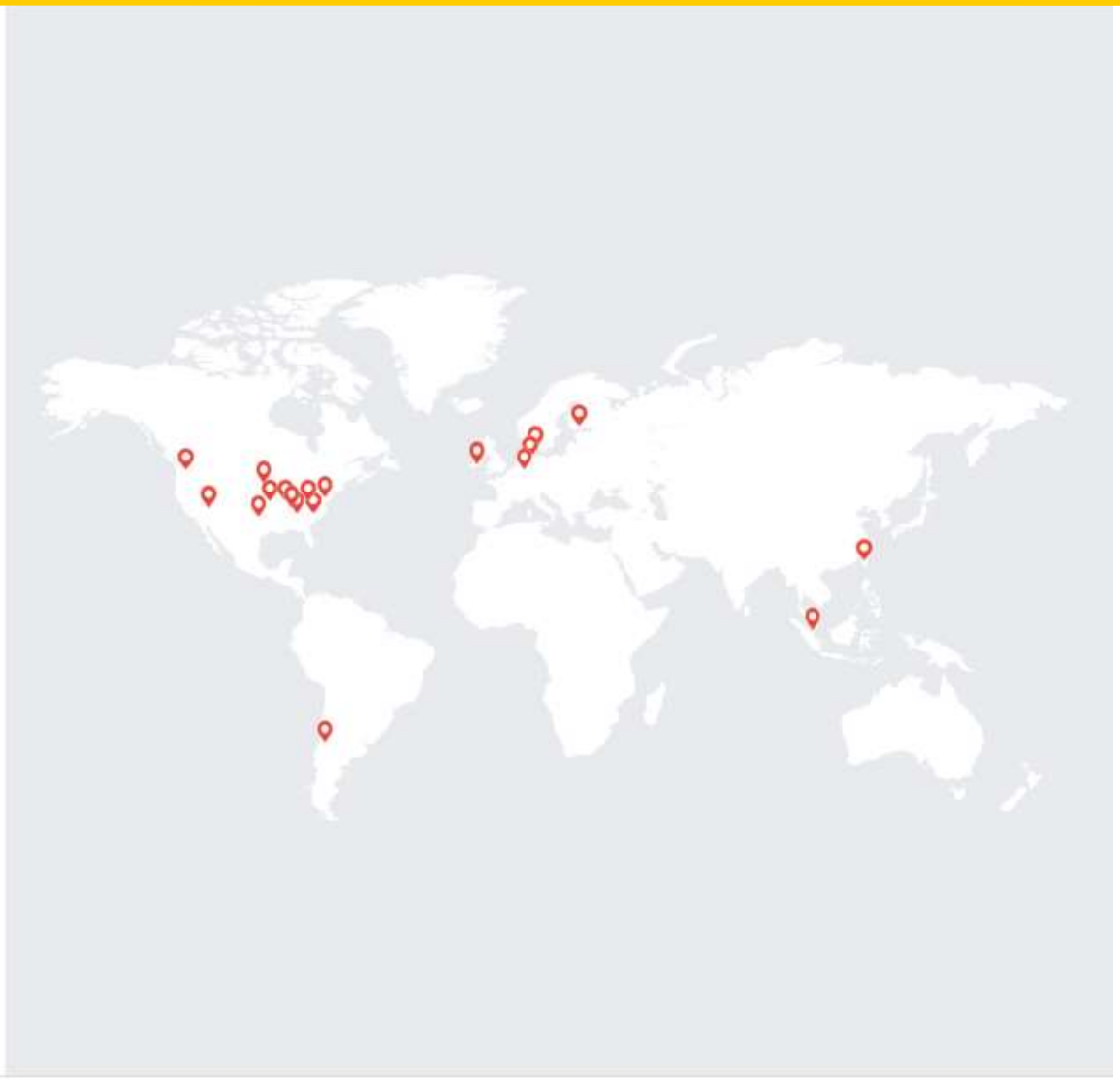
Hamina, Finland

St. Ghislain, Belgium

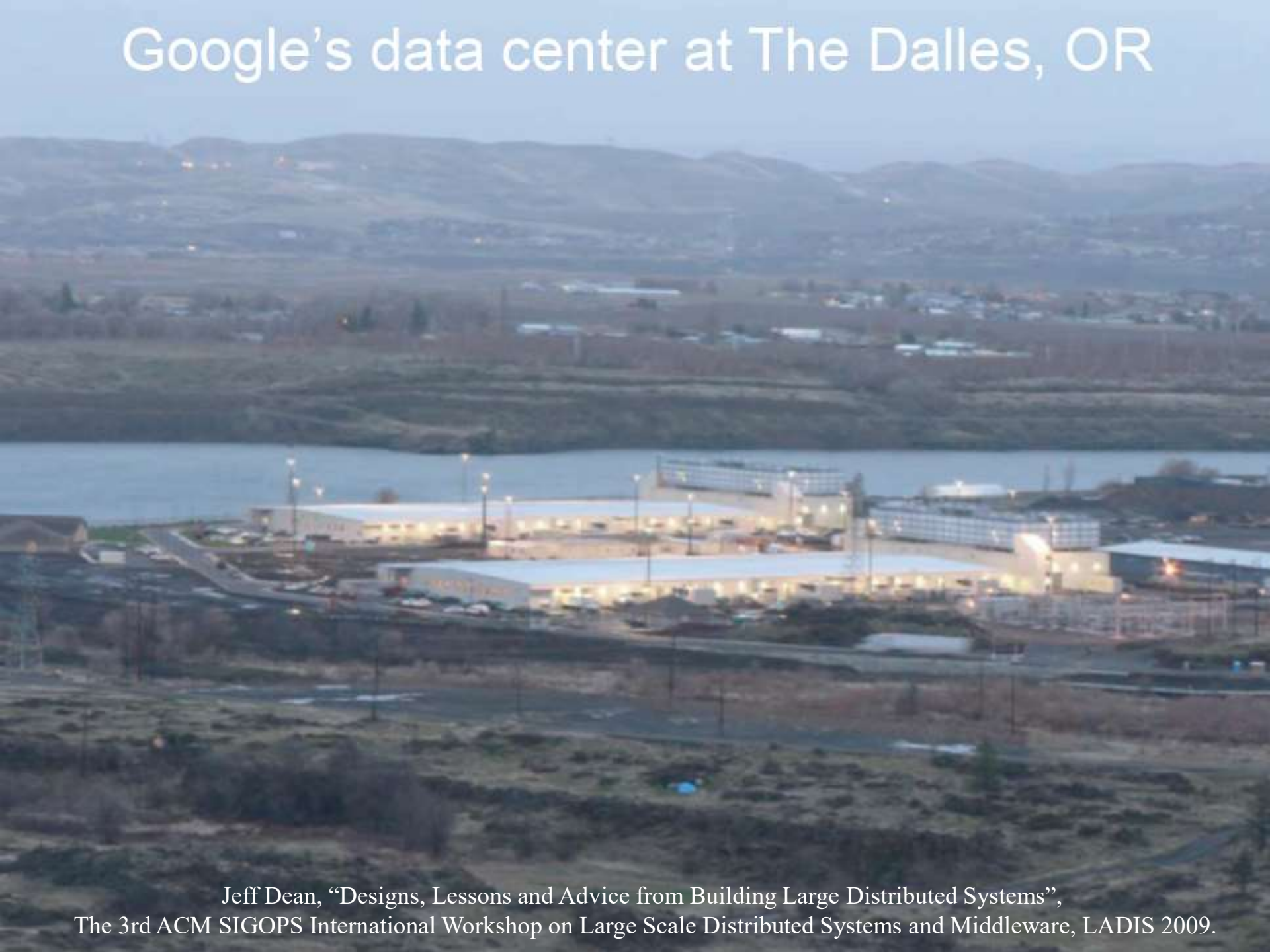
Asia

Changhua County, Taiwan

Singapore

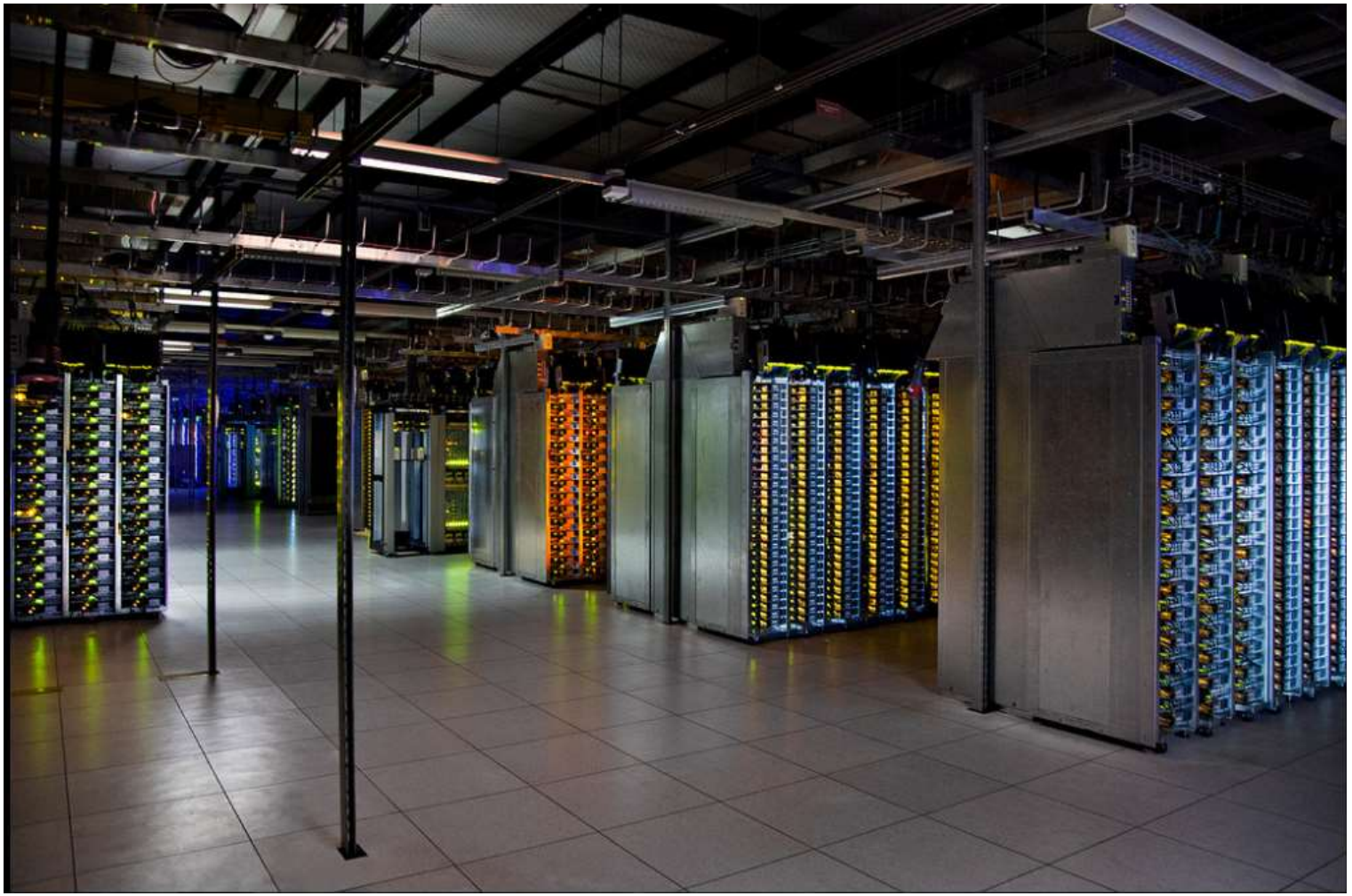


http://www.google.com/about/datacenters/inside/locations/



Jeff Dean, "Designs, Lessons and Advice from Building Large Distributed Systems",
The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, LADIS 2009.

Data center servers in The Dalles, Oregon.



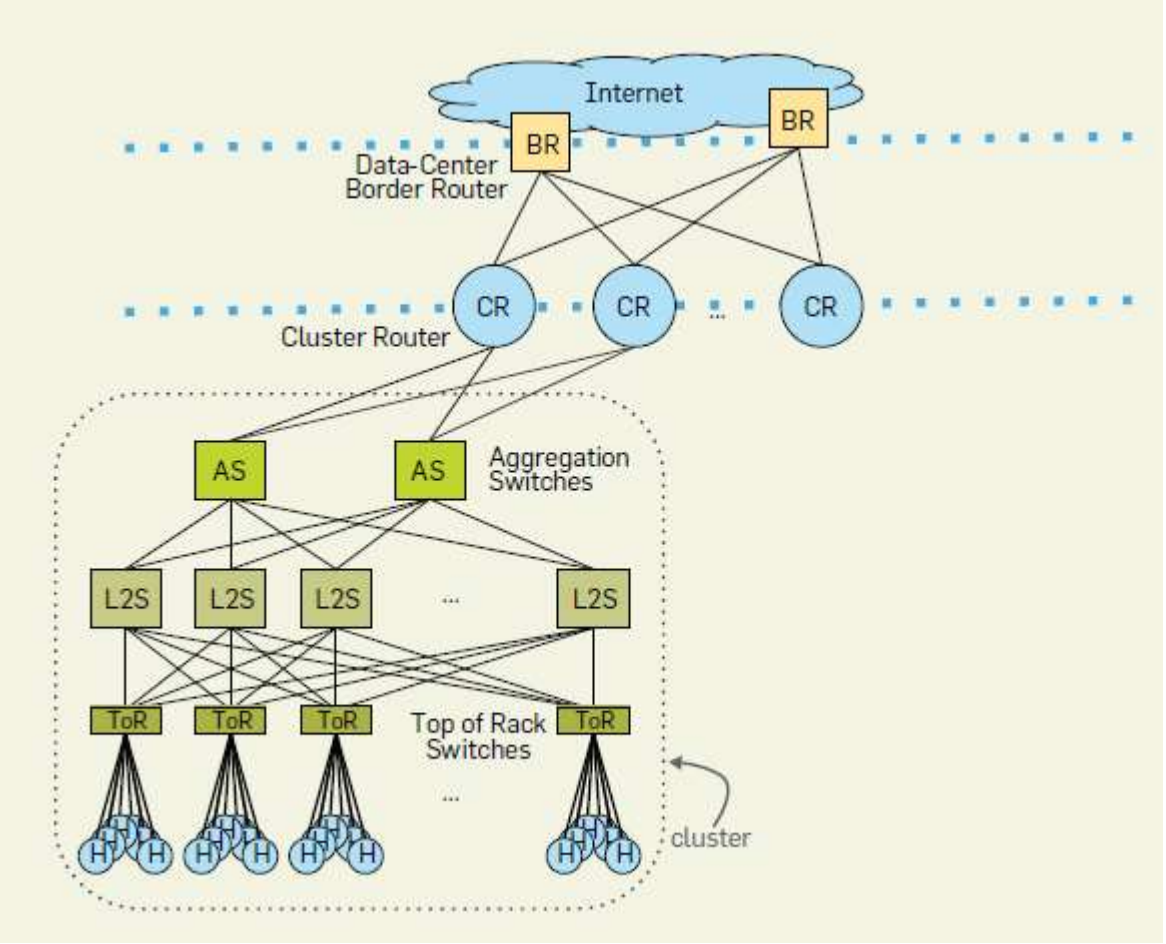
http://www.google.com/about/datacenters/gallery/#/tech

A “server room” in Council Bluffs, Iowa.

http://www.wired.com/wiredenterprise/2012/10/ff-inside-google-data-center/all/

Inside a Google Data Center
https://www.youtube.com/watch?v=XZmGGAbHqa0

2.4 Data center network - Jupiter



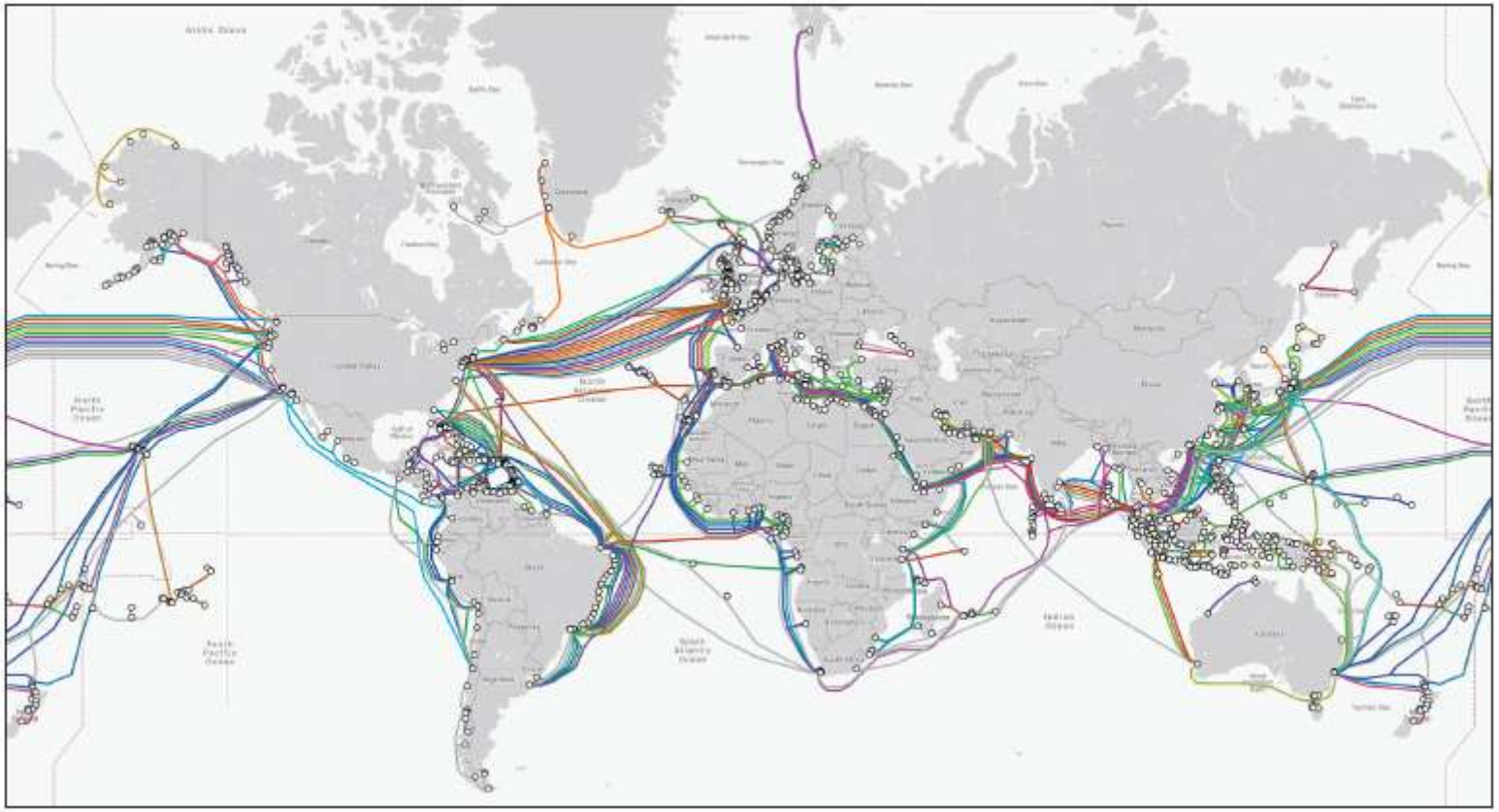
Fat-tree or folded-Clos topology

Wide area network – B4

Google runs its own private global network (switches and links)

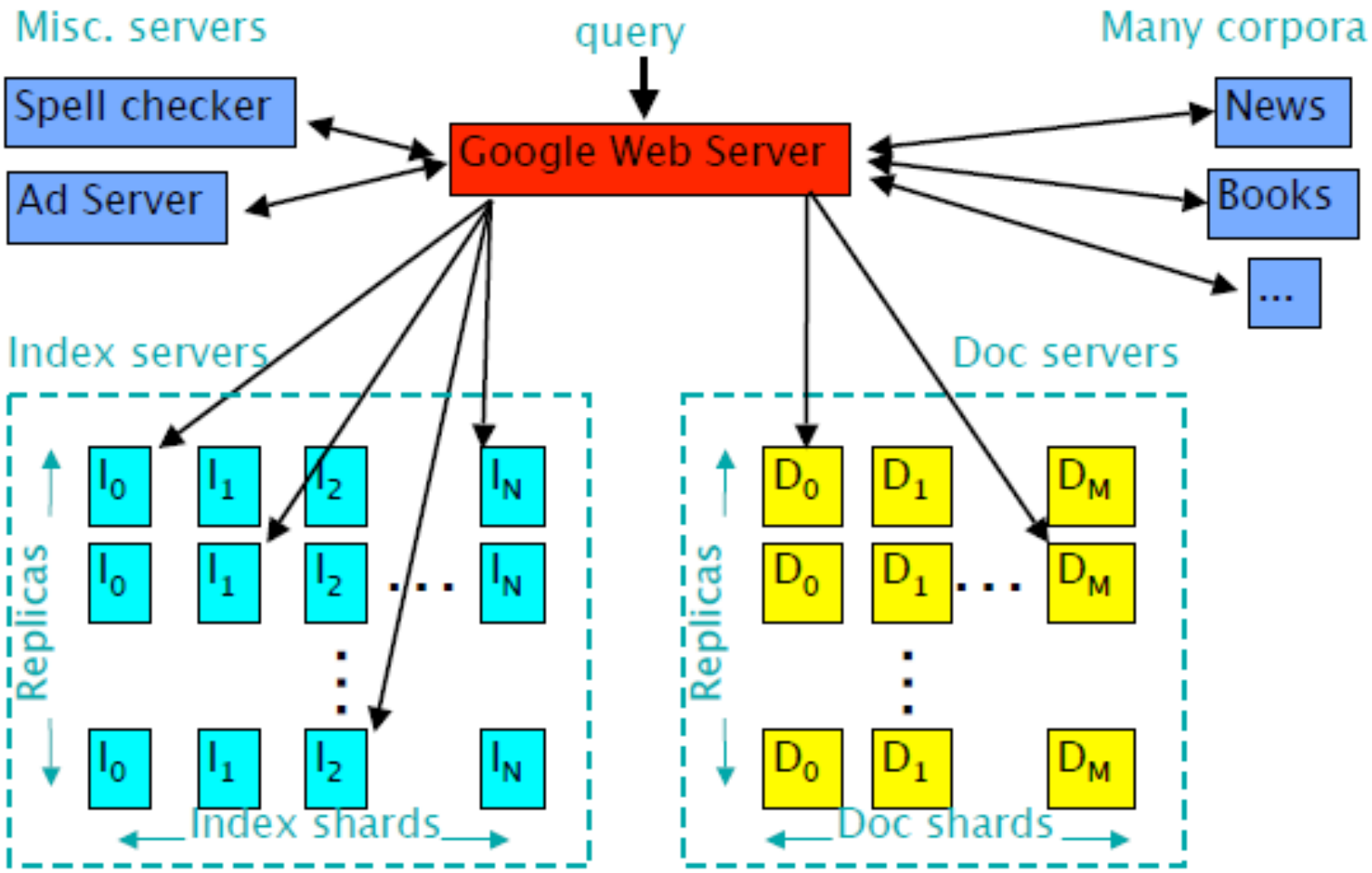
Each data center typically has at least three independent fibers connecting it to the private global network, thus ensuring path diversity for every pair of data centers

How the Internet spans the globe



Kugler, L. (2020). *Communications of the ACM*, Vol. 63, No. 1, pp. 14-16.

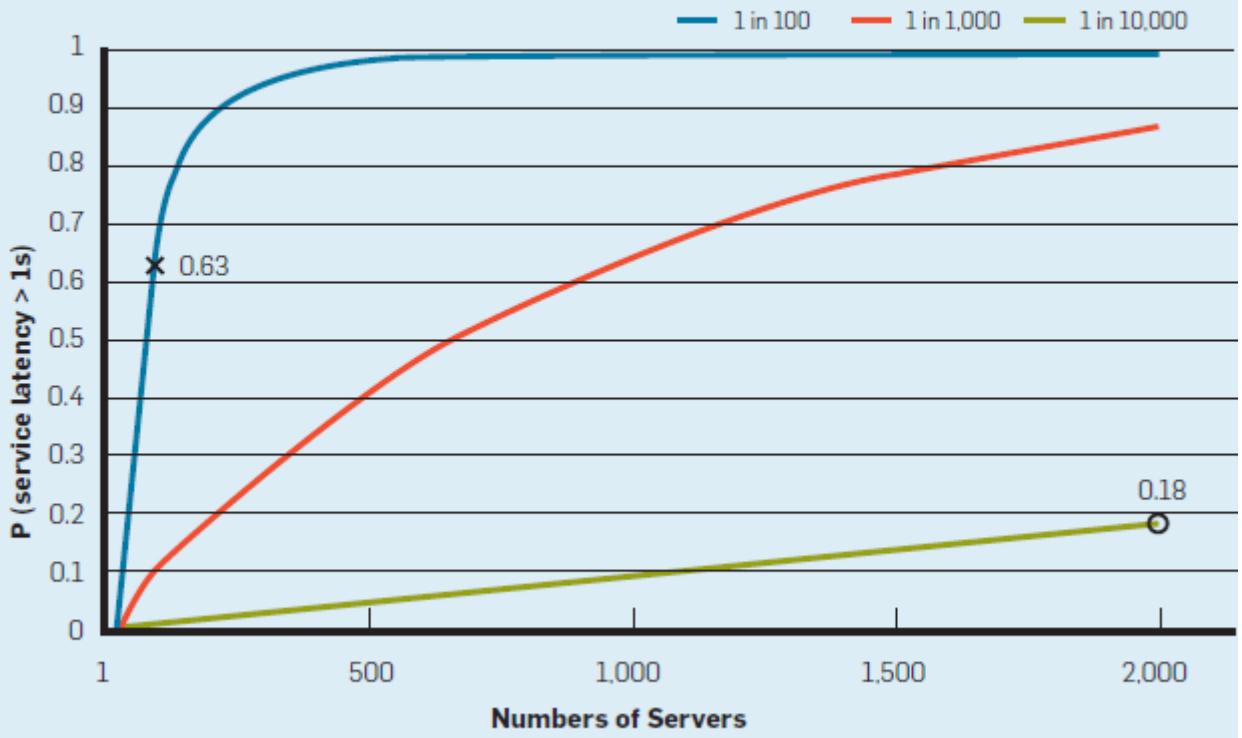
Query-serving architecture



Elapsed time: 0.25s, machines involved: 1000s+

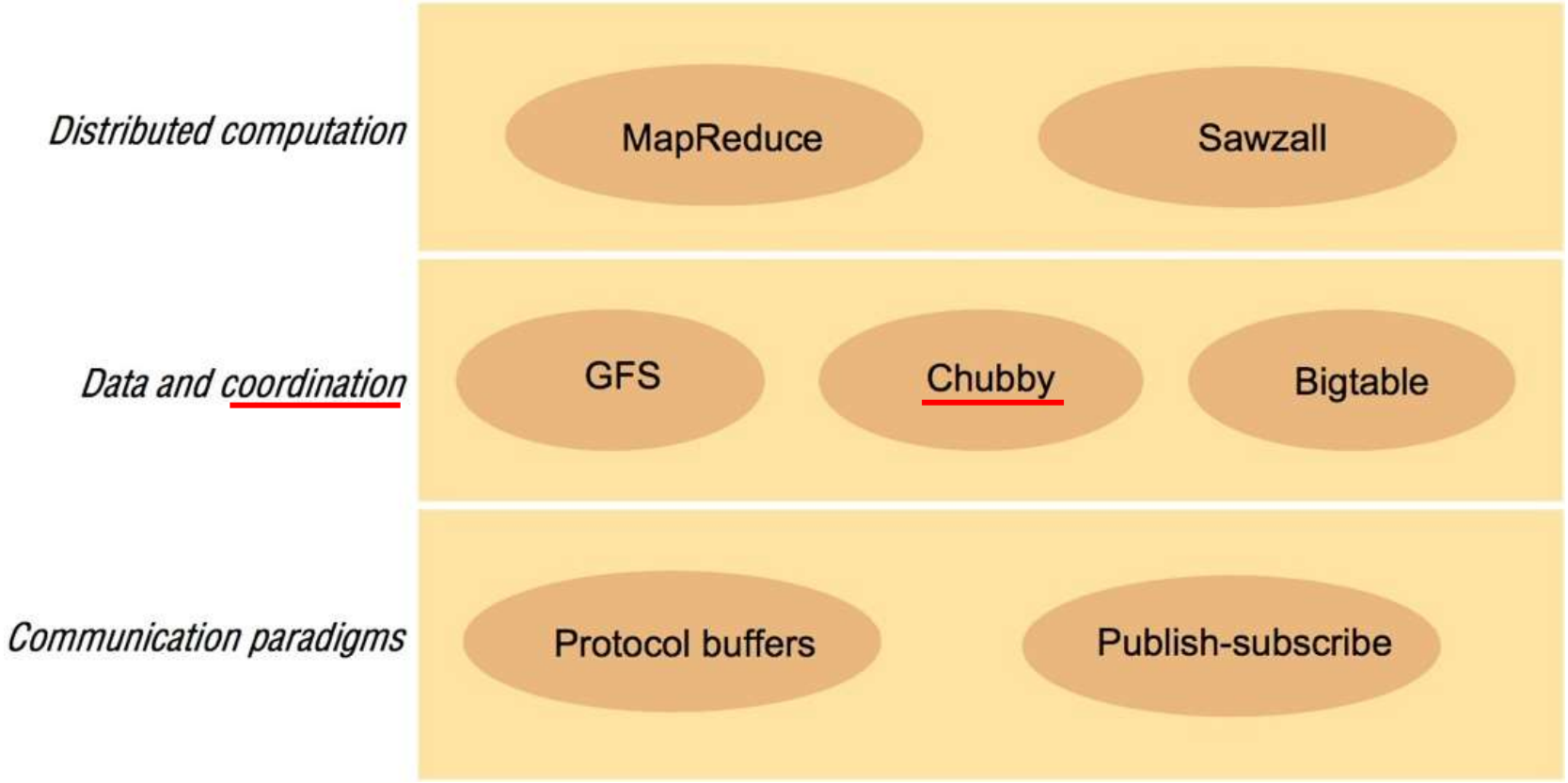
Tail at scale – solving latency variability

Probability of one-second service-level response time as the system scales and frequency of server-level high-latency outliers varies.



https://www.youtube.com/watch?v=1Qxnrf2pW10

3. Middleware



Design principles: simplicity, performance, stringent testing.

3.1. Communication paradigms

Element	*Design choice*	*Rationale*	*Trade-offs*
Protocol buffers	The use of a language for specifying data formats	Flexible in that the same language can be used for serializing data for storage or communication	-
	Simplicity of the language	Efficient implementation	Lack of expressiveness when compared, for example, with XML
	Support for a style of RPC (taking a single message as a parameter and returning a single message as result)	More efficient, extensible and supports service evolution	Lack of expressiveness when compared with other RPC or RMI packages
	Protocol-agnostic design	Different RPC implementations can be used	No common semantics for RPC exchanges

protobuf
Protocol Buffers - Google's data interchange format

Communication paradigms

Publish-subscribe	Topic-based approach	Supports efficient implementation	Less expressive than content-based approaches (mitigated by the additional filtering capabilities)
	Real-time and reliability guarantees	Supports maintenance of consistent views in a timely manner	Additional algorithmic support required with associated overhead

3.2. Data storage and coordination

Element	*Design choice*	*Rationale*	*Trade-offs*
GFS	The use of a large chunk size (64 megabytes)	Suited to the size of files in GFS; efficient for large sequential reads and appends; minimizes the amount of metadata	Would be very inefficient for random access to small parts of files
	The use of a centralized master	The master maintains a global view that informs management decisions; simpler to implement	Single point of failure (mitigated by maintaining replicas of operations logs)
	Separation of control and data flows	High-performance file access with minimal master involvement	Complicates the client library as it must deal with both the master and chunkservers
	Relaxed consistency model	High performance, exploiting semantics of the GFS operations	Data may be inconsistent, in particular duplicated
Chubby	Combined lock and file abstraction	Multipurpose, for example supporting elections	Need to understand and differentiate between different facets
	Whole-file reading and writing	Very efficient for small files	Inappropriate for large files
	Client caching with strict consistency	Deterministic semantics	Overhead of maintaining strict consistency
Bigtable	The use of a table abstraction	Supports structured data efficiently	Less expressive than a relational database
	The use of a centralized master	As above, master has a global view; simpler to implement	Single point of failure; possible bottleneck
	Separation of control and data flows	High-performance data access with minimal master involvement	-
	Emphasis on monitoring and load balancing	Ability to support very large numbers of parallel clients	Overhead associated with maintaining global states

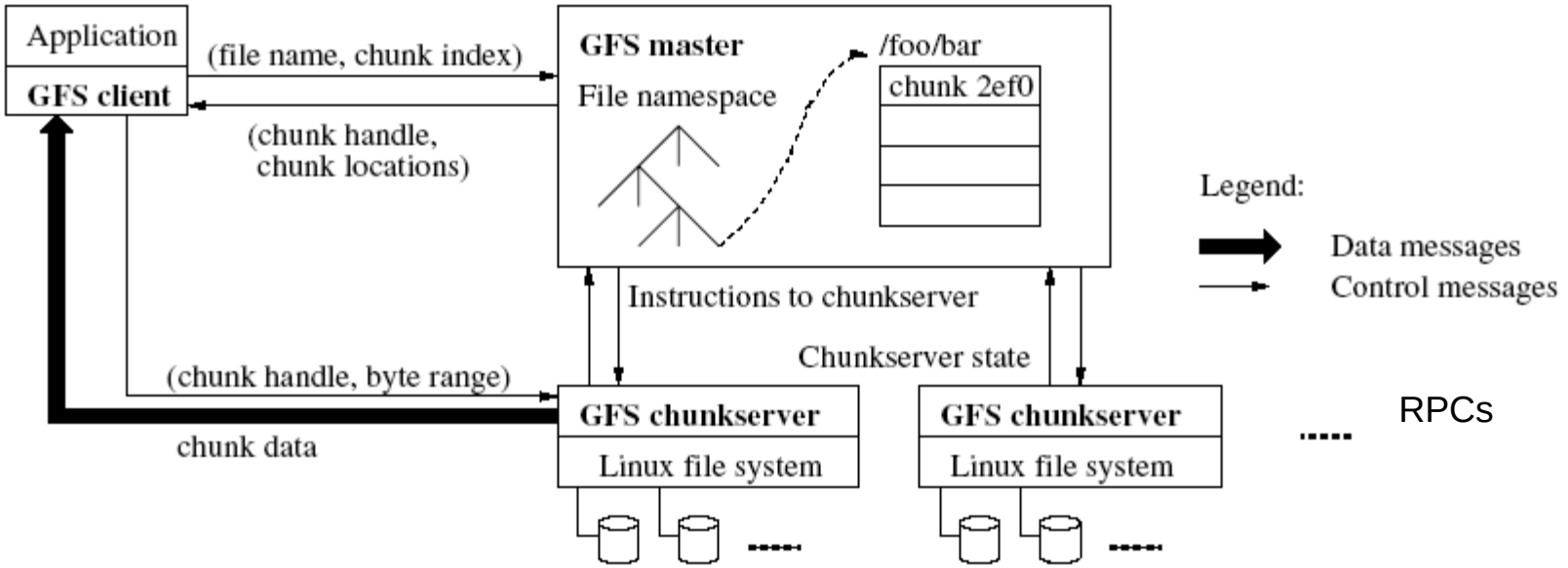
Colossus

Spanner

Infrastructure software

7th Extremely Larga Data Bases Conference XLDB
Stanford University, September 9-12, 2013
https://www.youtube.com/watch?v=gCGvneeHbPQ

GFS

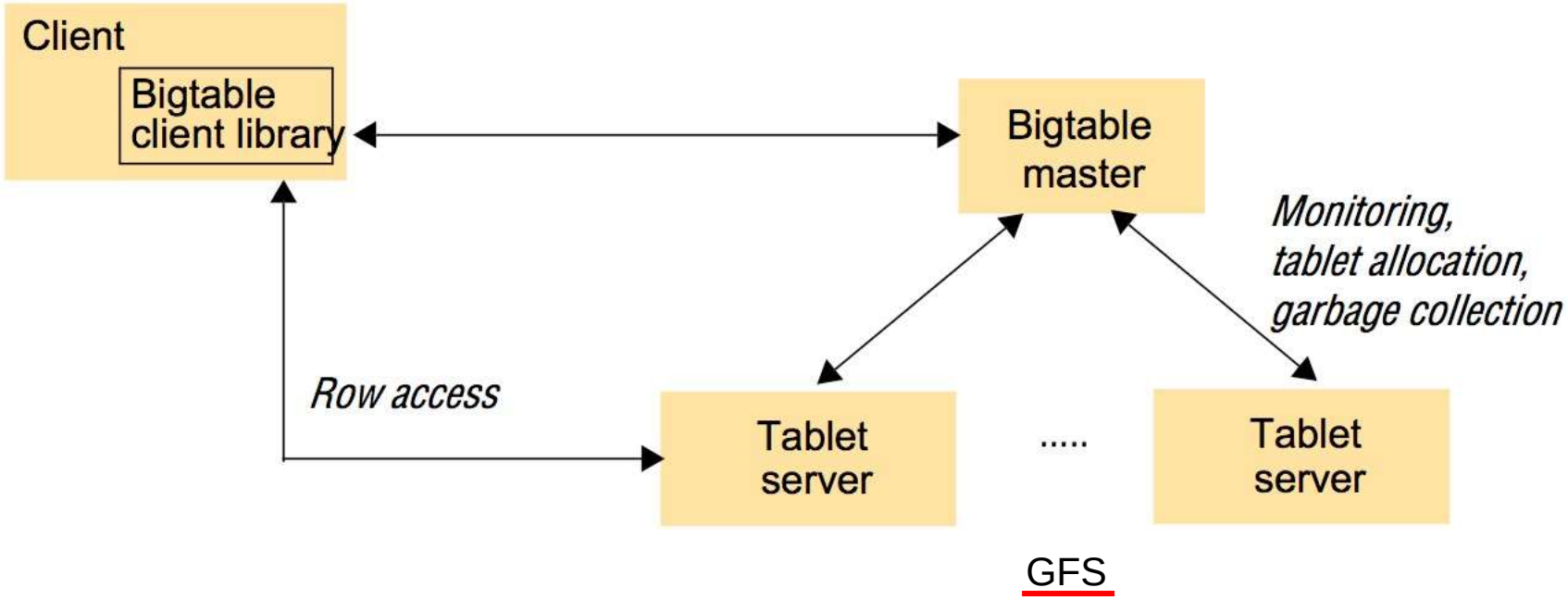


Master election

Replicas: 1) availability, 2) consistency

Ghemawat, S., et al. (2003). The Google File System, *SOSP'03*, pp. 29-43, November 3-6, Farmington, PA.

Bigtable



Chang, F., et al. (2006). Bigtable: A Distributed Storage System for Structured Data, *OSDI'06*, pp. 205-218, November 6-8, Seattle, WA.

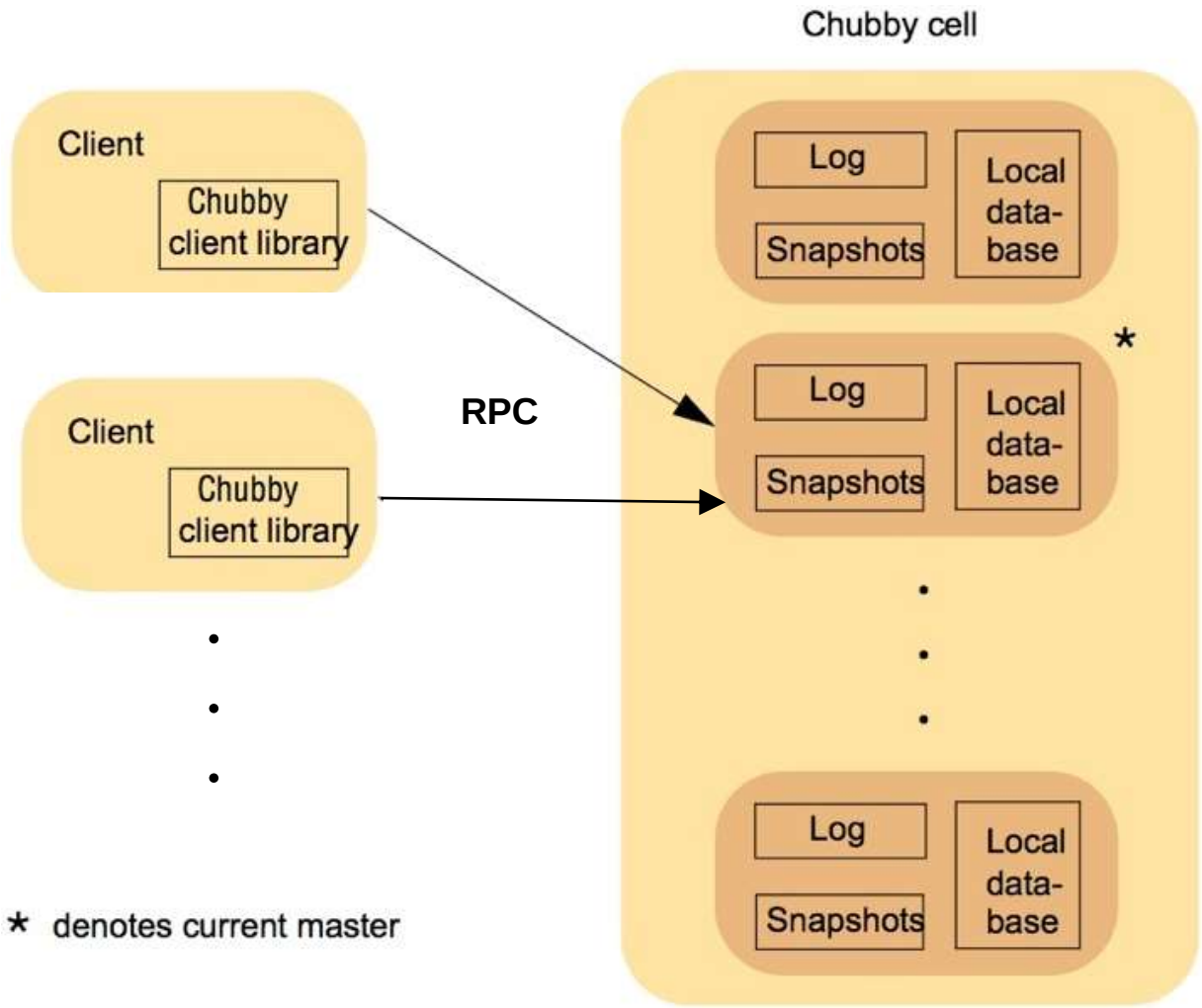
Chubby

Chubby is a fault-tolerant system intended to provide coarse-grained **locking** and (low-volume) reliable storage

Chubby provides an interface much like a distributed file system with advisory locks, but the design emphasis is on reliability, as opposed to high performance

Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems, *OSDI'06*, pp. 335-350, November 6-8, Seattle, WA.

Overall architecture of Chubby



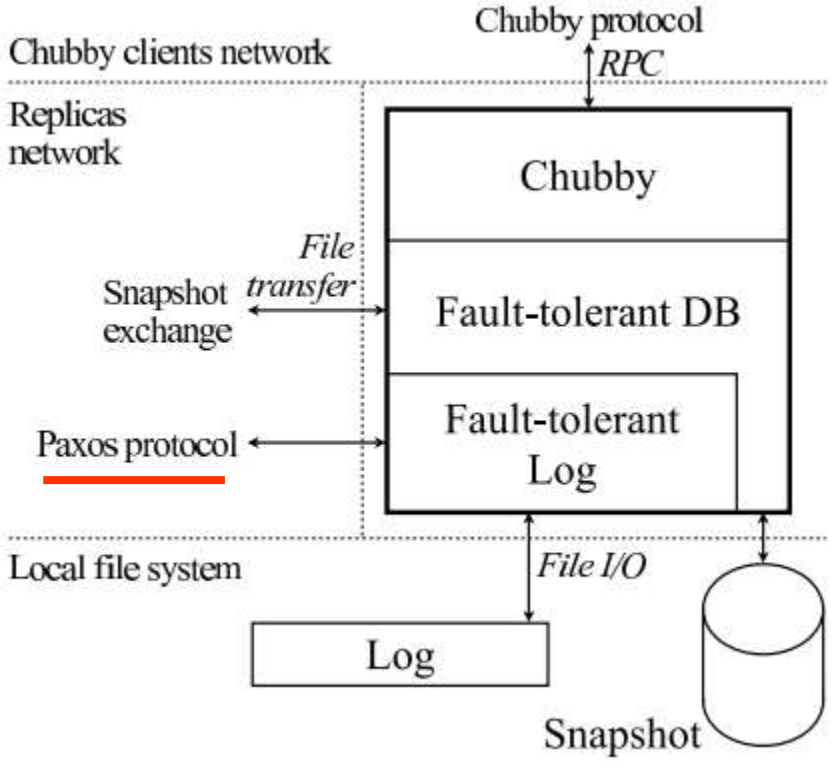
Availability

Fault-tolerant **replicated** log

Coordination Consistency

Distributed consensus

Overall architecture of Chubby



Stable storage

3.3. Distributed computation

Element	*Design choice*	*Rationale*	*Trade-offs*
MapReduce	The use of a common framework	Hides details of parallelization and distribution from the programmer; improvements to the infrastructure immediately exploited by all MapReduce applications	Design choices within the framework may not be appropriate for all styles of distributed computation
	Programming of system via two operations, *map* and *reduce*	Very simple programming model allowing rapid development of complex distributed computations	Again, may not be appropriate for all problem domains
	Inherent support for fault-tolerant distributed computations	Programmer does not need to worry about dealing with faults (particularly important for long-running tasks running over a physical infrastructure where failures are expected)	Overhead associated with fault-recovery strategies
Sawzall	Provision of a specialized programming language for distributed computation	Again, support for rapid development of often complex distributed computations with complexity hidden from the programmer (even more so than with MapReduce)	Assumes that programs can be written in the style supported (in terms of filters and aggregators)

Cloud dataflow

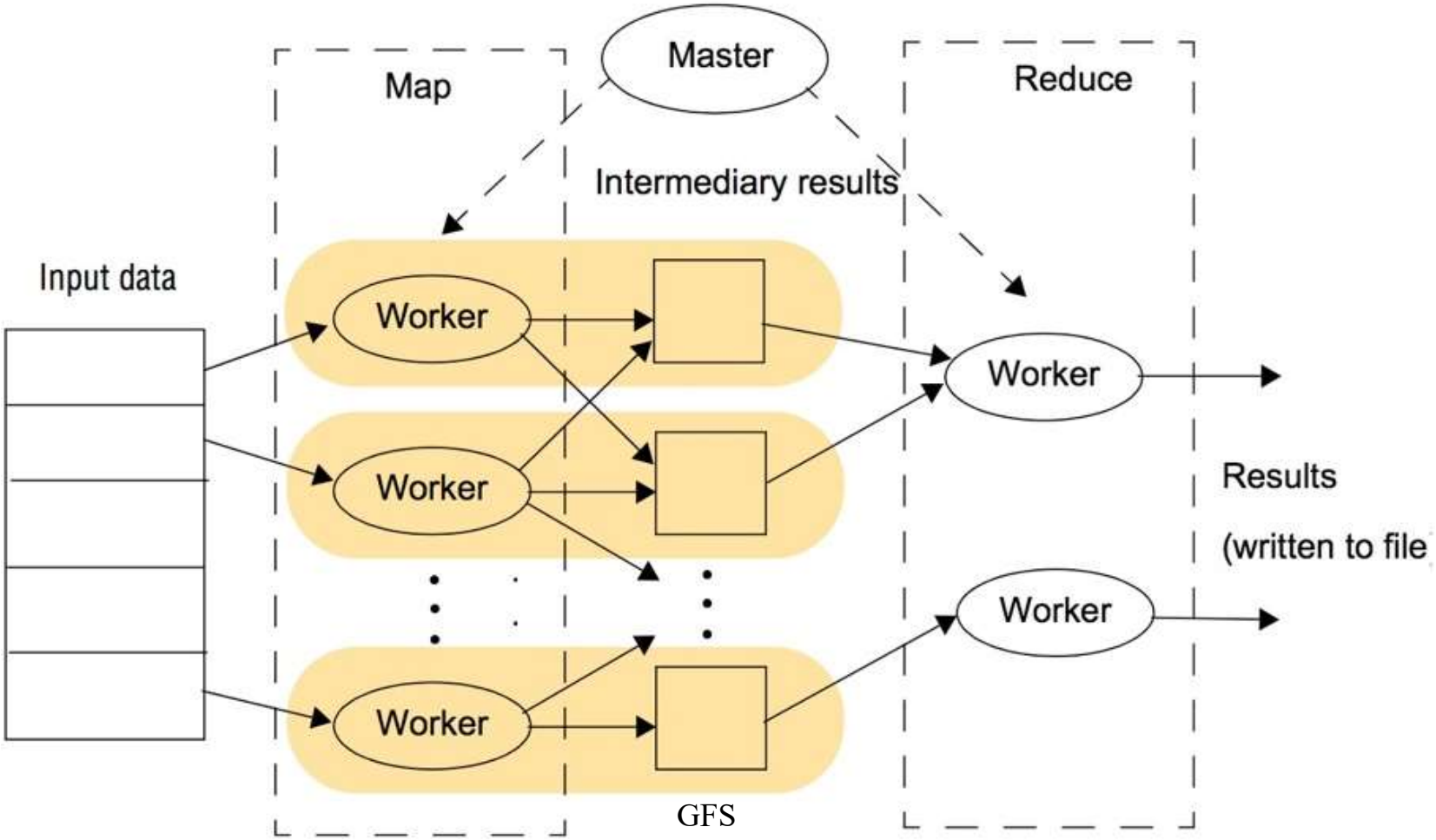
Lingo

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster

Jeffrey Dean and Sanjay Ghemawat (2004). MapReduce: Simplified Data Processing on Large Clusters, *OSDI'04*, pp. 137-150, December 6-8, San Francisco, CA.

The overall execution of a MapReduce program

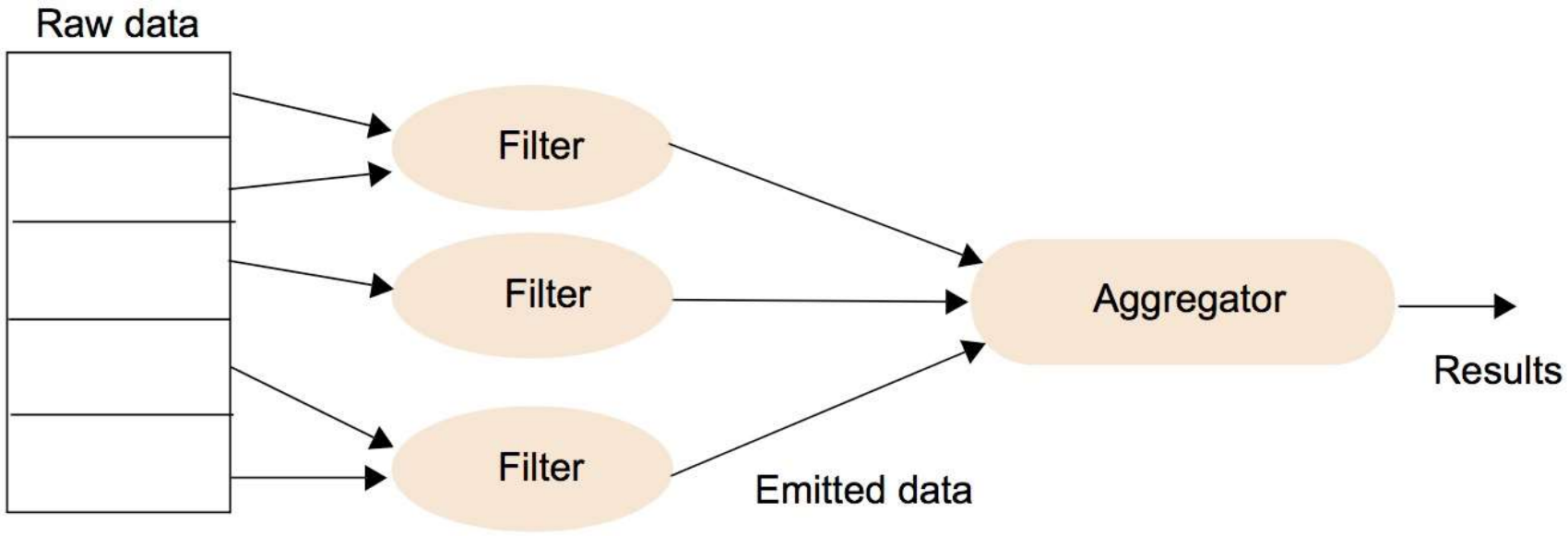


Fault tolerance

Examples of the use of MapReduce

Function	*Initial step*	*Map phase*	*Intermediate step*	*Reduce phase*
Word count	*Partition data into fixed-size chunks for processing*	For each occurrence of word in data partition, emit <*word*, 1>	*Merge/sort all key-value keys according to their intermediary key*	For each word in the intermediary set, count the number of 1s
Grep		Output a line if it matches a given pattern		Null
Sort *N.B. This relies heavily on the intermediate step*		For each entry in the input data, output the key-value pairs to be sorted		Null
Inverted index		Parse the associated documents and output a <*word*, document ID> pair wherever that word exists		For each word, produce a list of (sorted) document IDs

The overall execution of a Sawzall program



Summary

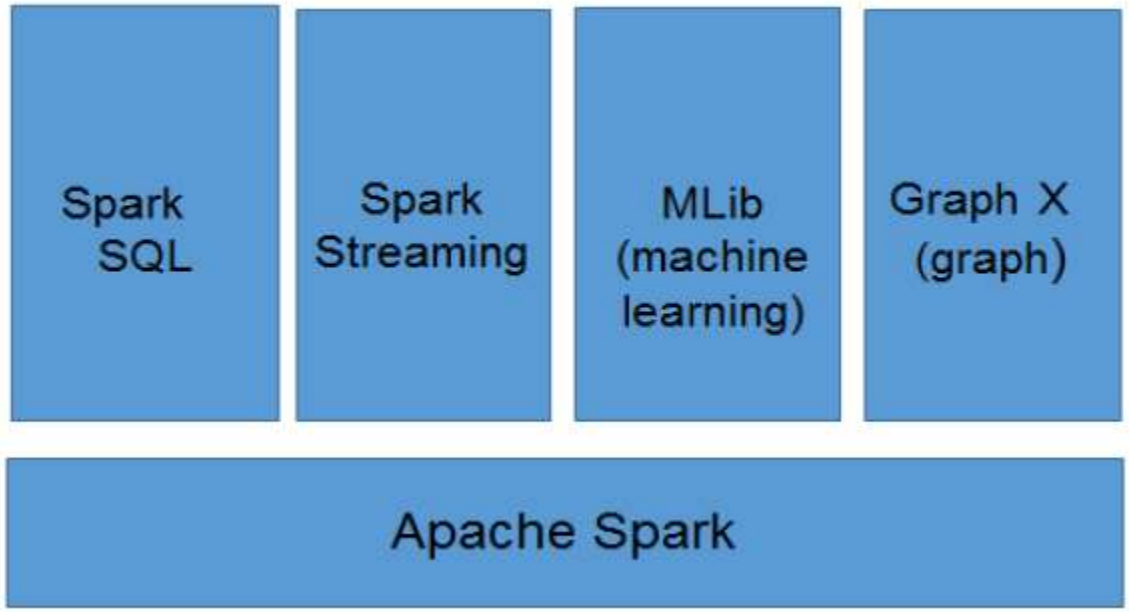
- One key lesson to be taken from this case study is the importance of really understanding your application domain, deriving a core set of underlying design principles and applying them consistently
- In the case of Google, this manifests itself in a strong advocacy of simplicity and low-overhead approaches coupled with an emphasis on testing, logging and tracing
- The end result is an architecture that is highly scalable, reliable, high performance and open in terms of supporting new applications and services

4. (Apache) Hadoop

Google	Hadoop
Sawzall	Pig
MapReduce	Hadoop MapReduce
Bigtable	HBase
GFS	HDFS
Chubby	ZooKeeper
Protocol Buffers	Avro

Spark

Apache Spark is a unified analytics engine for large-scale data processing.



Anexo

Chubby - Paxos

Chubby

Chubby's design is based on well-known (research and practical) ideas that have meshed well:

- distributed consensus among a few replicas for fault tolerance
- consistent client-side caching to reduce server load while retaining simple semantics
- timely notification of updates
- a familiar file system interface

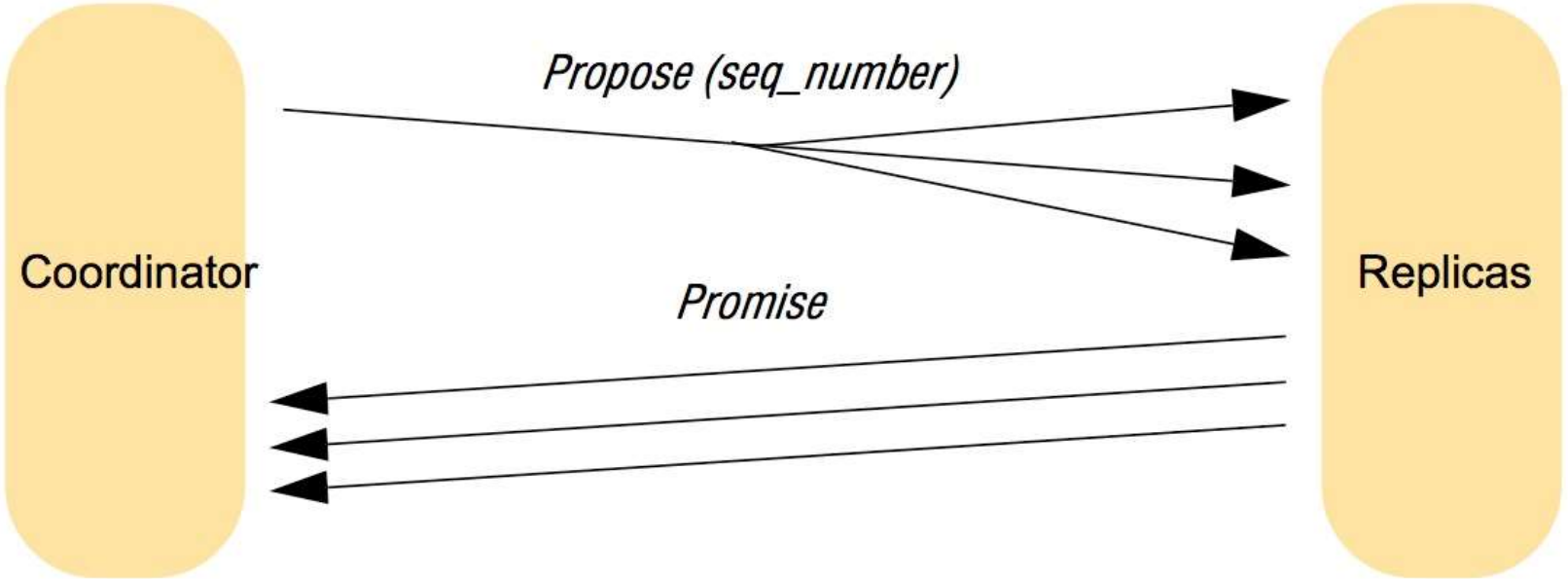
Chubby API

Role	*Operation*	*Effect*
General	*Open*	Opens a given named file or directory and returns a handle
	Close	Closes the file associated with the handle
	Delete	Deletes the file or directory
File	*GetContentsAndStat*	Returns (atomically) the whole file contents and metadata associated with the file
	GetStat	Returns just the metadata
	ReadDir	Returns the contents of a directory – that is, the names and metadata of any children
	SetContents	Writes the whole contents of a file (atomically)
	SetACL	Writes new access control list information
Lock	*Acquire*	Acquires a lock on a file
	TryAquire	Tries to acquire a lock on a file
	Release	Releases a lock

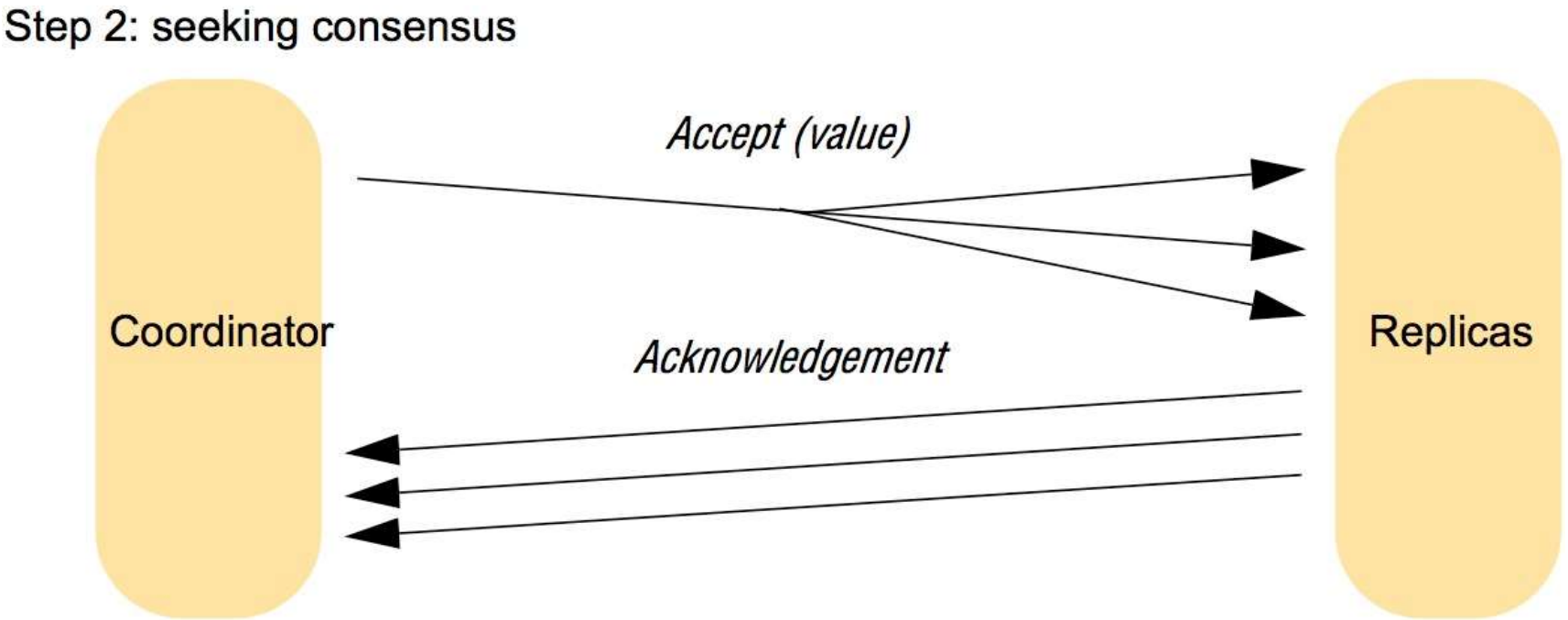
Every data object is a file

Message exchanges in Paxos (in absence of failures)

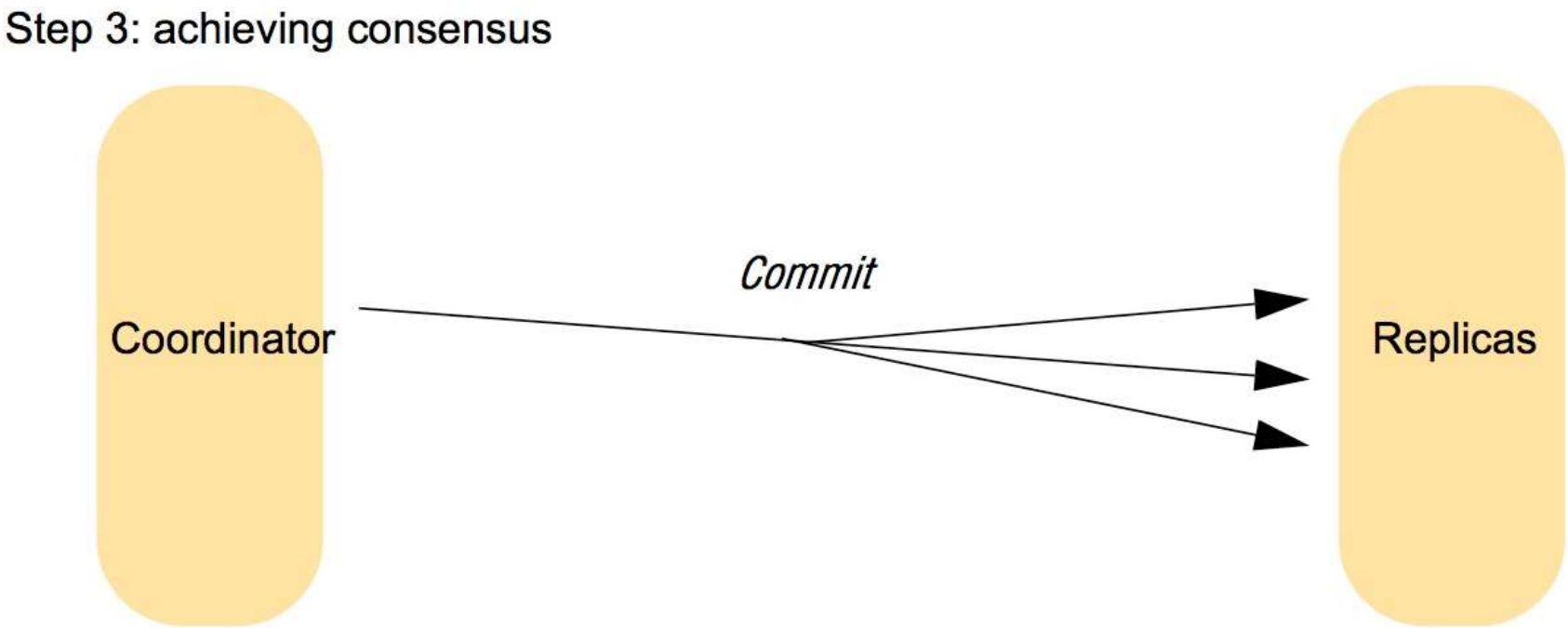
Step 1: electing a coordinator



Message exchanges in Paxos (in absence of failures)



Message exchanges in Paxos (in absence of failures)



Paxos in Chubby

- MultiPaxos:
  - Step 1 (Propose and Promise) done once
  - Steps 2 (Accept) and 3 (Commit) repeated multiple times by the same coordinator

- Leader election: Replicas in a cell initially use Paxos to establish the coordinator (the majority of replicas must agree)
- Master Lease: Replicas promise not to try to elect a new master for at least a few seconds; master lease is periodically renewed
- Master failure: If replicas lose contact with master, they wait for a grace period (4-6 s); on timeout, they hold a new election