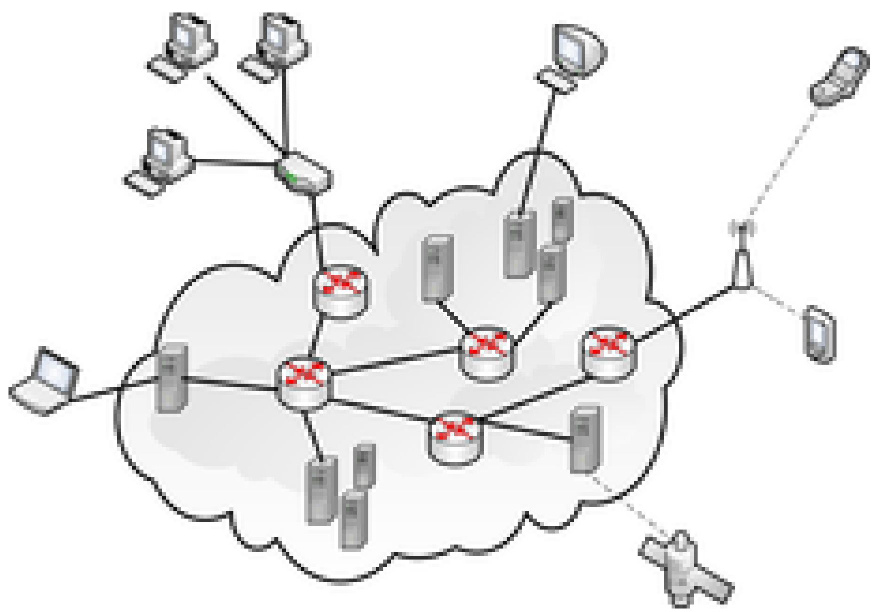
ECE419 - Distributed Systems: Introduction, Motivation & Overview

http://ece419.msrg.utoronto.ca

Slides available in Quercus

(**before** or after lecture)



ECE 419 in a Nutshell

- Introduction to distributed systems
  - Principles, foundations
  - Algorithms, protocols
  - Case studies, applications, architectures

- Learn to build and run distributed systems

- My **office hour:** Monday, 6-7 PM (by appointment)

ECE 419 in a Nutshell

Lectures

Occasional Tutorial Lecture

Our Project:
Four Milestones

Four Assignments

Exams (midterm & final)

TAs

Arnamoy Bhattacharyya
b.arnamoy@mail.utoronto.ca

Genrui Zhang, **Head TA**
gengrui.zhang@mail.utoronto.ca

Fei Pan
Fei.pan@mail.utoronto.ca

Shashank Motepalli

Dai Qin
dai.qin@mail.utoronto.ca

Yuqiu Zhang

Agenda This Week

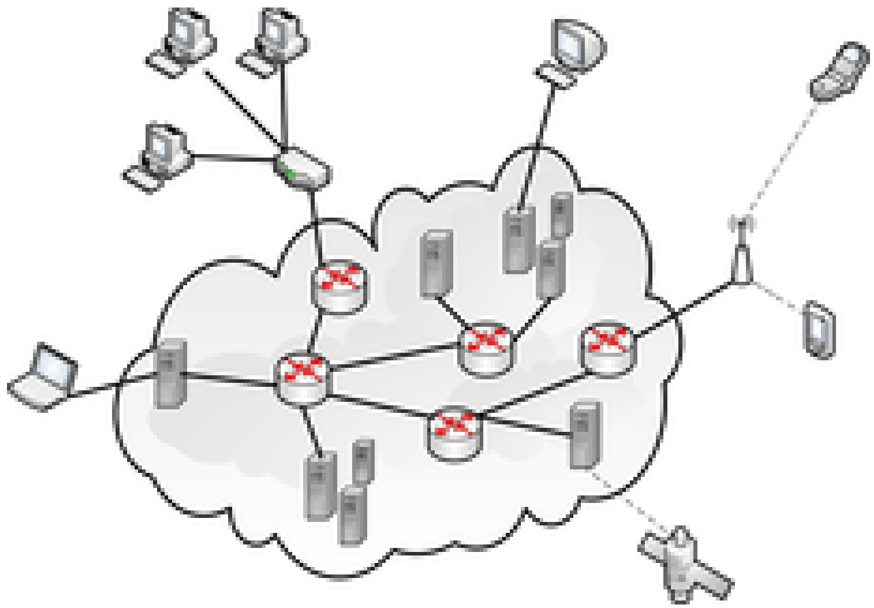
- Distributed systems overview
  - *Roadmap for semester*

- Administrative overview

- Course project (milestones)

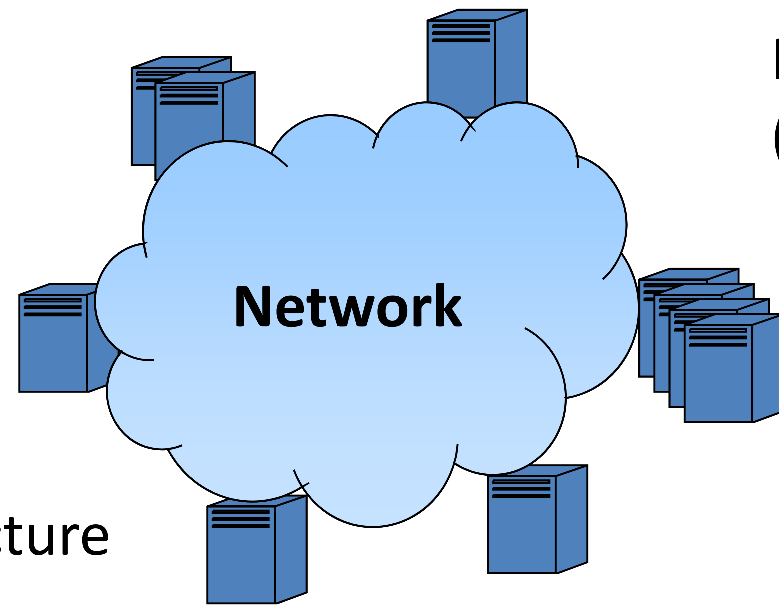
DISTRIBUTED SYSTEMS OVERVIEW

What is a Distributed System?



Working Definition

A distributed system is a system that is comprised of several **physically disjoint compute resources** interconnected by a **network**.



Peer-to-peer
(Bitcoin, BitTorrent)

MapReduce
(Hadoop)

World Wide Web
(Akamai CDN)

Google infrastructure
(BigTable)

Other Definitions & Views

- A distributed system is one in which **hardware** or **software components** located at **networked computers communicate** and **coordinate** their actions only by **passing messages**.
  - *By Coulouris et al.*

- A distributed system is a **collection of independent computers** that appears to its users as **a single coherent system**.
  - *By Tanenbaum & van Steen.*

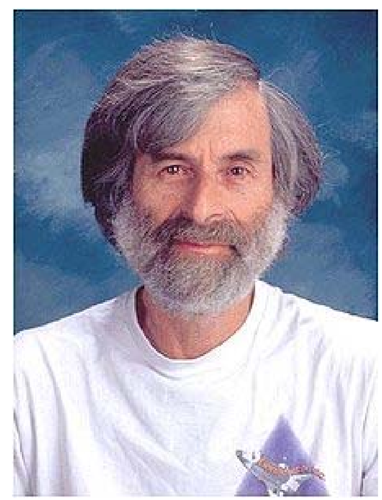
“Introduction to Distributed Systems Design”

- A distributed system is an **application** that executes a collection of **protocols** to **coordinate the actions** of multiple **processes** on a **network,** such that all components **cooperate** together to perform a **single** or small set of **related tasks**.
  - By Google Code University

Leslie Lamport's Anecdotal Remark

- *"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*. " (While at DEC SRC, 1985-2001)

- "Father" of distributed systems
  – Turing Award 2013
  – Inventor of LaTeX, ☺



Leslie Lamport

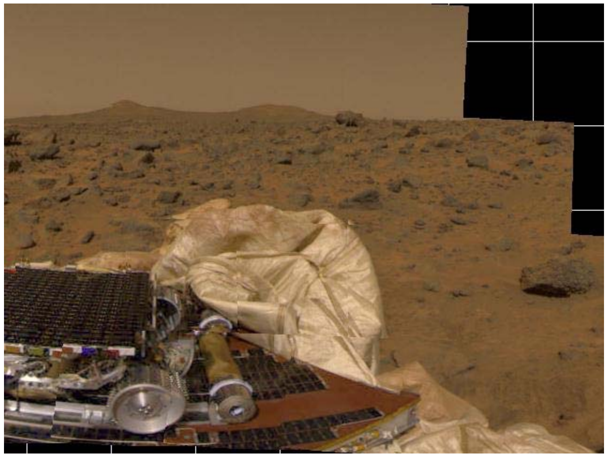
Why Build a Distributed System?

- Centralized system is simpler in all respects:
  - Local memory, storage
  - Failure model
  - Maintenance
  - Data security

Why Build a Distributed System?

But…

Mars Pathfinder, July 1997

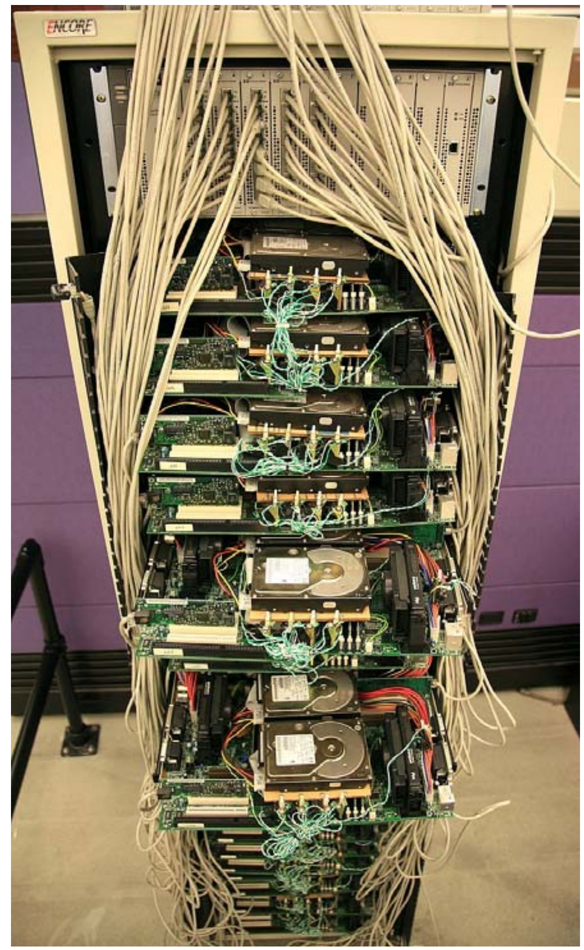


- **Vertical scaling** costs more than **horizontal scaling**
- Availability and redundancy
- Single point of failure

- Many resources are inherently distributed
- Many resources used in a shared fashion

First Google Computer, Cluster

(http://www.computerhistory.org/collections/catalog/102662167)



Related Disciplines …

Networking
- Latency
- Communication

Databases
- Transactions
- Consistency

Distributed Systems

Security
- Faulty stacks
- *Privacy, encryption*

Parallel computing
- Concurrency
- Decomposition

Characteristics of distributed systems

- Reliable
- Fault-tolerant
- Highly available
- Recoverable
- Consistent
- Scalable
- Predictable performance
- Secure
- Heterogeneous
- Open

Also known as the ***ilities***

(non-functional requirements)

Many of them still pose ***significant challenges*** *in theory and in practice!*

Reliability

- Probability of a system to **perform** its **required functions** under **stated conditions** for a **specified period of time**.
- To run continuously without failure

- Expressed as
  Mean Time Between Failure
  (MTBF), failure rate

Availability & High-availability

- Proportion of time a system is in a **functioning state**, i.e., can be used, **(1 – unavailable).**
- **Ratio** of time usable over entire time
  - Informally, uptime / (uptime + downtime)
  - System that **can be used 100 hrs** out of **168 hrs** has **availability of 100/168**
- Specified as decimal or percentage
  - Five nines is 0.99999 or 99.999% available

Nines or Class of 9

# Nines	Avail. (%)	Downtime per			
		year	month	week	day
1 x 9	90	36.5 d	3 d	16.8 h	2.4 h
2 x 9	99	3.65 d	7.2 h	1.68 h	14.4 mins
4 x 9	99.99	52.56 min	4.32 min	60.48 s	8.64 s
5 x 9	99.999	5.256 min	25.92 s	6.048 s	864 ms
6 x 9	99.9999	31.536 s	2.592 s	604.8 ms	86.4 ms
9 x 9	99.9999999	31.536 ms	2.592 ms	604.8 µs	86.4 µs

Nines or Class of 9

- Frequently used for telecommunication systems et al.
- Favorite marketing term
- Does not capture impact or cost of downtime

*"According to Google, its Gmail service was **available** 99.984 percent of the time in 2010 …"* by P. Lilly

Availability ≠ Reliability

- System going down 1 ms every 1 hr has an availability of more than 99.9999%
  - Highly available, but also highly unreliable

- A system that never crashes, but is taken down for two weeks
  - Highly reliable, but only about 96% available

Distributed Systems Design Fallacies

- **Assumptions** (novice) designers of distributed systems often make **that turn out to be false**
- Originated in 1994 by Peter Deutsch, Sun Fellow, Sun Microsystems

- **The 8 fallacies**
  - The network is reliable.
  - Latency is zero.
  - Bandwidth is infinite.
  - The network is secure.
  - Topology doesn't change.
  - There is one administrator.
  - Transport cost is zero.
  - The network is homogeneous

DISTRIBUTED SYSTEM EXAMPLE: BIGTABLE

Key-value stores

- *What is a key-value-store?*
- *Why is a key-value store needed*?
- Key-value-store client interface
- Key-value stores in practice
- Common features & non-features
- Apache HBase
- Apache Cassandra

What mechanisms make them work?

What are key-value stores?

- Container for key-value pairs (databases)
- Distributed, multi-component, systems
- NoSQL semantics (non-relational)
- KV-Stores offer **simpler query semantics** in exchange for **increased scalability**, **speed**, **availability**, and **flexibility**.

DBMS (SQL)

Students Table

Student	ID*
John Smith	084
Jane Bloggs	100
John Smith	182
Mark Antony	219

Activities Table

ID*	Activity*	Cost
084	Swimming	$17
084	Tennis	$36
100	Squash	$40
100	Swimming	$17
182	Tennis	$36
219	Golf	$47
219	Swimming	$15
219	Squash	$40

- Relational data schema
- Data types
- Foreign keys
- Full SQL support

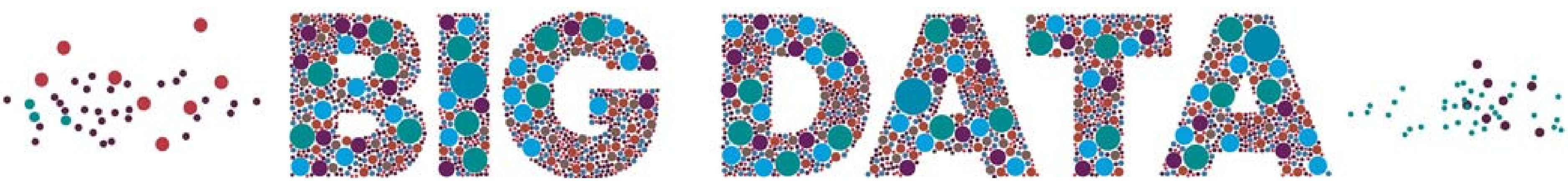
Key-value store

Key	Value
John Smith	{Activity:Name=Swimming}
Jane Bloggs	{Activity:Cost=57}
Mark Anthony	{ID=219}

- No data schema
- Raw byte access
- No relations
- Single-row operations

Why are key-value stores needed?

- Today's internet applications
  - Huge amounts of stored data (1 PB = 10^{15} bytes)
  - Huge number of Internet users (e.g., 3.4 billion)
  - Frequent updates
  - Fast retrieval of information
  - Rapidly changing data definitions
- Ever more users, ever more data



Why are key-value stores needed?

- Horizontal scalability
  - User growth, traffic patterns change
  - Adapt to number of requests, data size
- Performance
  - High speed for single-record read and write operations
- Flexibility
  - Adapt to changing data definitions

Why are key-value stores needed?

- Reliability
  - Thousands of components at play
  - Uses commodity hardware: failure is the norm
  - Provide failure recovery
- Availability and geo-distribution
  - Users are worldwide
  - Guarantee fast access

Key-value store client interface

- Main operations
  - Write/update **put**(key, value)
  - Read **get**(key)
  - Delete **delete**(key)

- Usually no aggregation, no table joins, no transactions!

Hbase: Key-value store client interface

```
Configuration conf = HBaseConfiguration.create();
conf.set("hbase.zookeeper.quorum", "192.168.127.129");

HTable table = new HTable(conf, „M

Put put = new Put(Bytes.toBytes("k                 ));
put.add(Bytes.toBytes("colfam1"), Bytes.toBytes(„value"), Bytes.toBytes(200));
table.put(put);

Get get = new Get(Bytes.toBytes("key1"));
Result result = table.get(get);
byte[] val = result.getValue(Bytes.toBytes("colfam1"), Bytes.toBytes(„value"));
System.out.println("Value: " + Bytes.toInt(val));
```

Initialization Using ZooKeeper

Column Family: “Schema”

Column: Defined at run-time (“wide column” stores)

Key-value store in practice

- BigTable
- Apache HBase
- Apache Cassandra
- Redis
- Amazon Dynamo
- Yahoo! PNUTS

Common elements of key-value stores

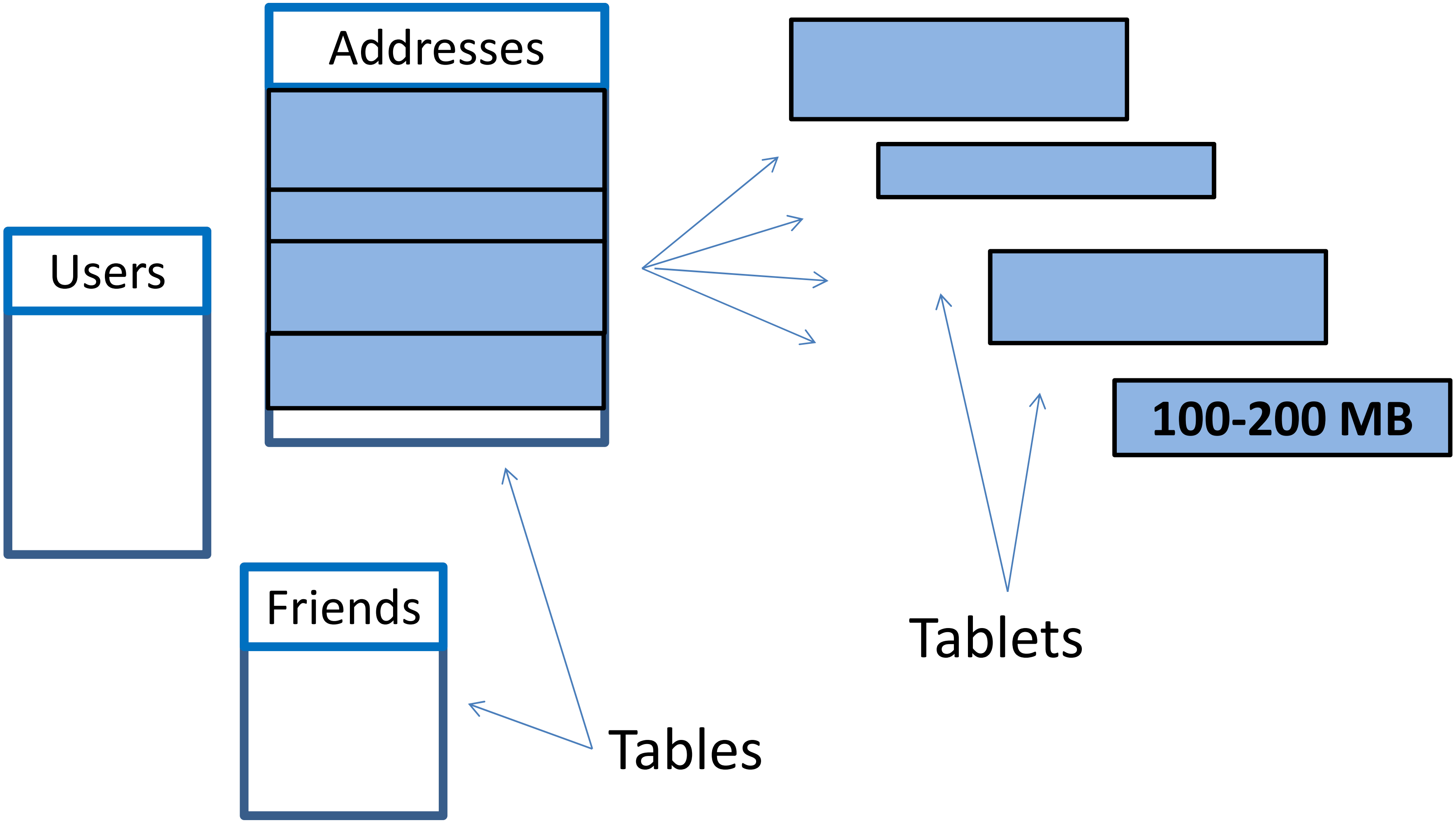
- Failure detection & failure recovery *(cf. DS Models Lecture)*
- Replication *(cf. Replication Lecture)*
  - Store and manage multiple copies of data
- Memory store & write ahead log (WAL) *(cf. Web Caching, Consistent Hashing Lecture)*
  - Keep data in memory for fast access
  - Keep a commit log as ground truth
- Versioning (*cf. Time in DS Lecture*)
  - Store different versions of data
  - Timestamping

BIGTABLE / HBASE

BigTable

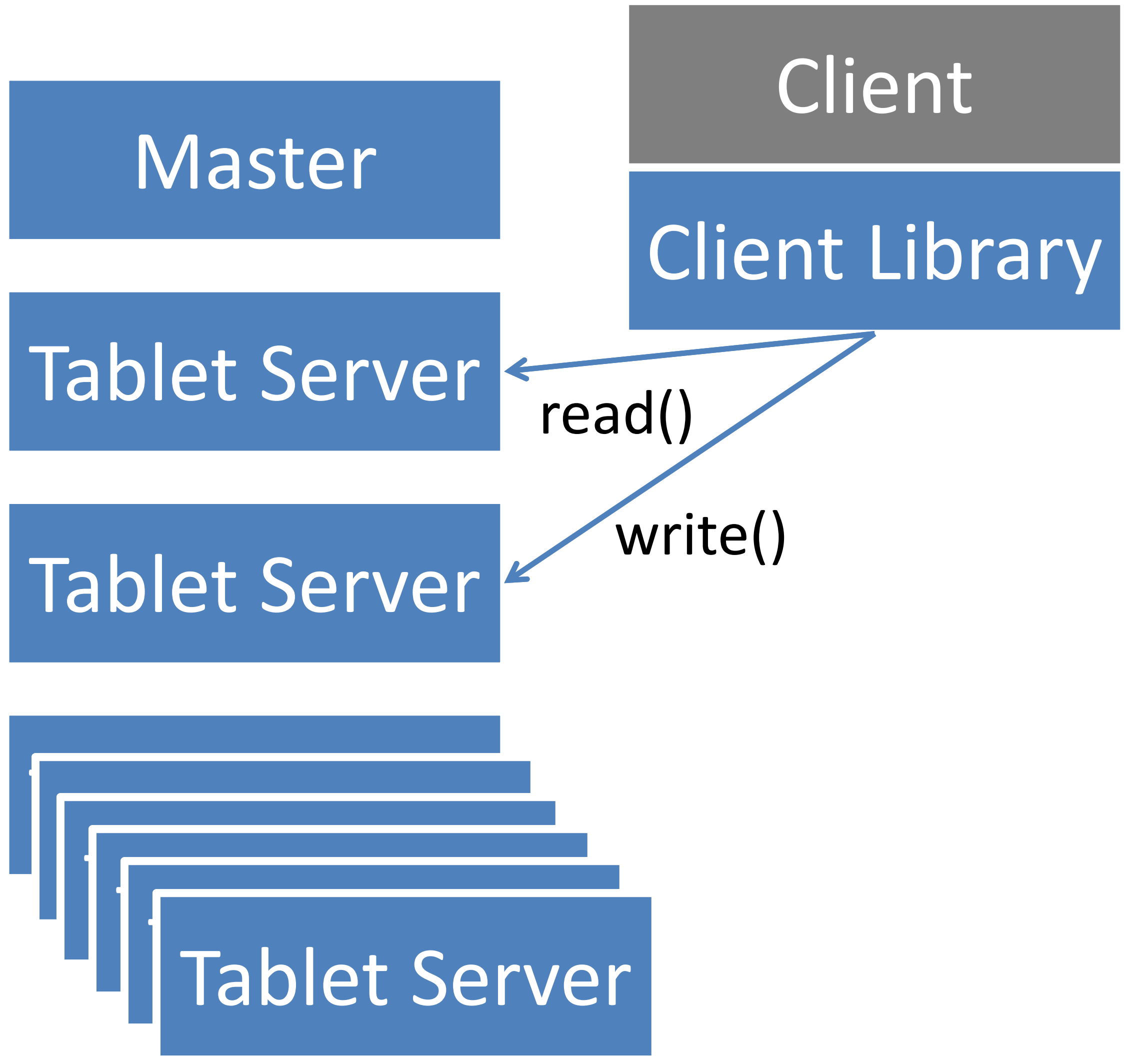
- Engineered at Google, 2004
- Designed for petabyte scale
- Internal use for web indexing, personalized search, Google Earth, Google Analytics, Google Finance
- Based on Google File System (GFS), *cf. GFS et al. Lecture*

BigTable: Tables & Tablets
(Logical Organization)



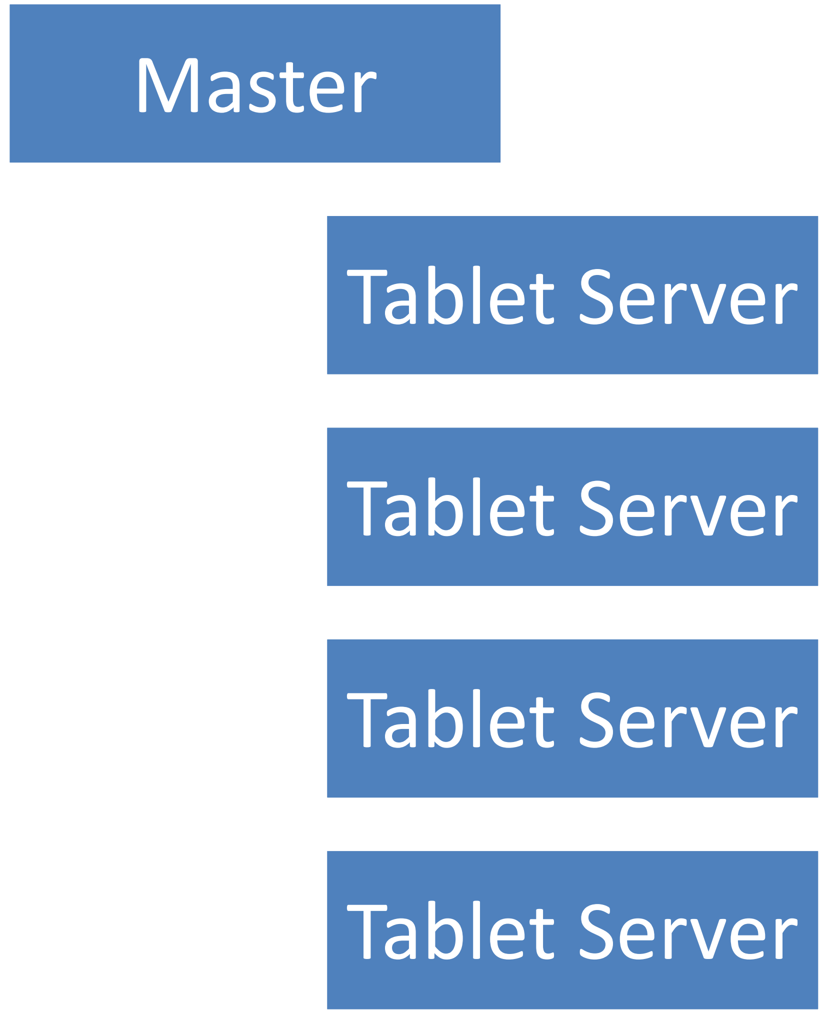
BigTable Components

- Client library
- Master
  - Metadata operations
  - Load balancing
- Tablet server
  - Data operations



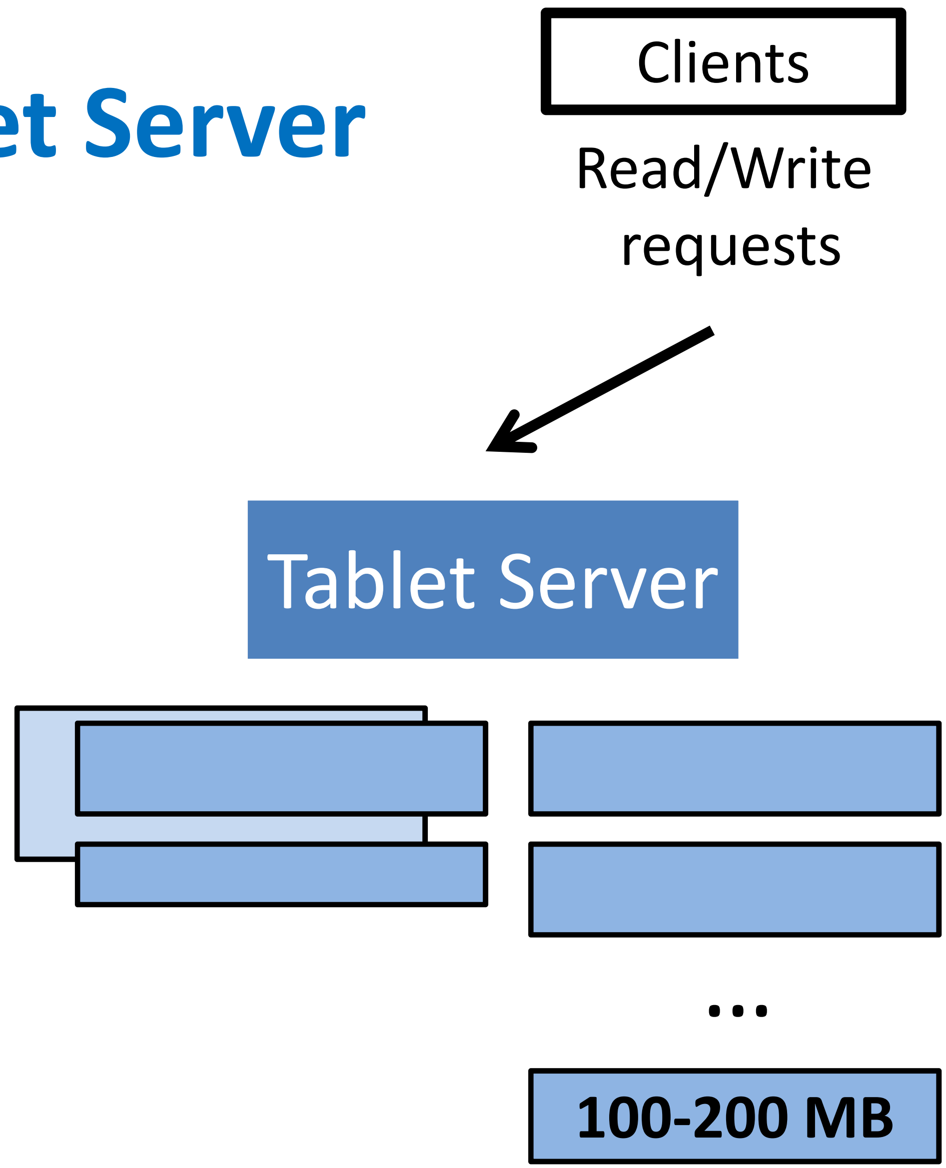
Master

- Assigns tablets to tablet servers
- Detects addition and expiration of tablet servers
- Balance tablet server load



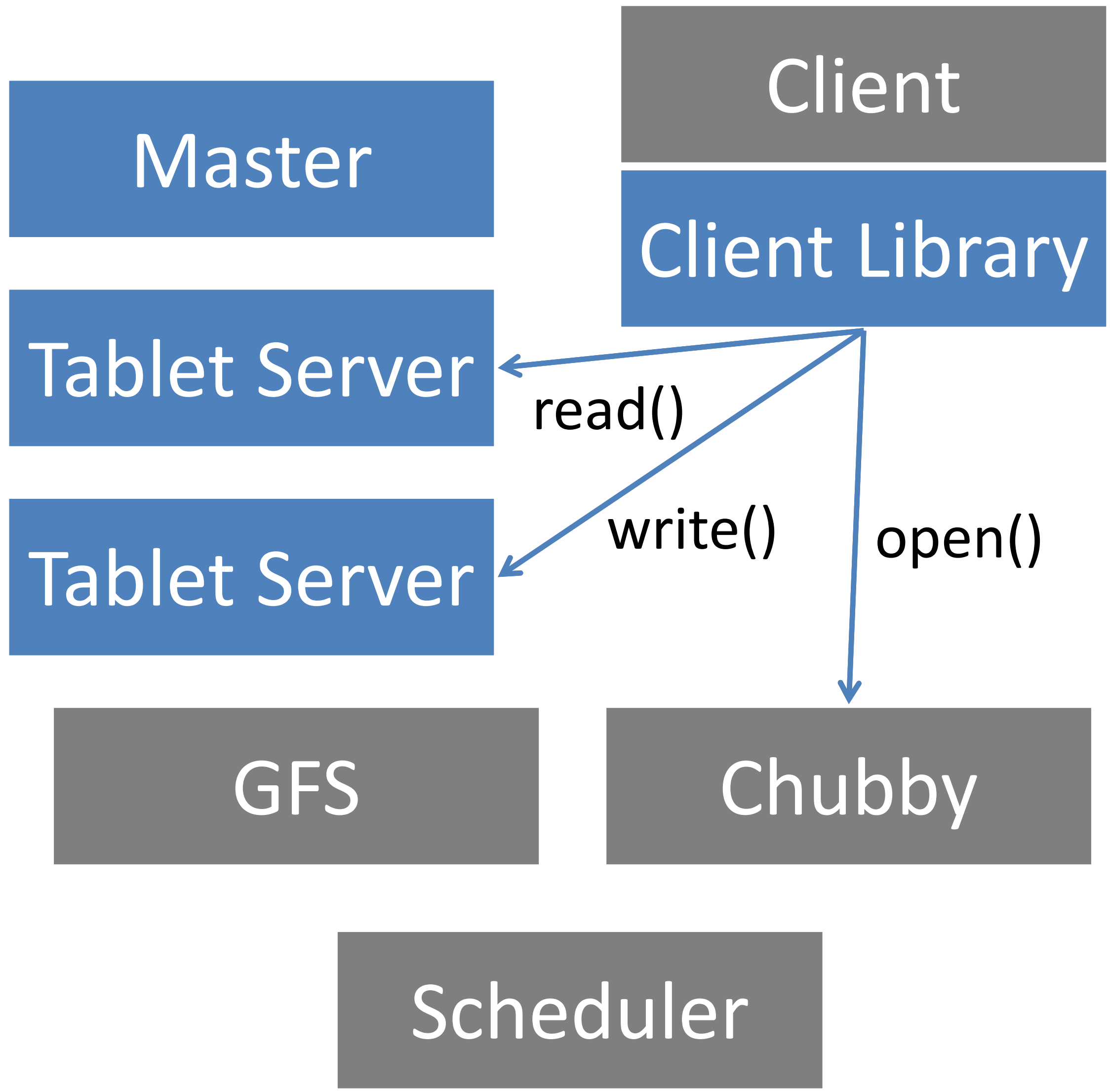
Tablet Server

- Manages a set of tablets (up to a thousand)
- Handles read and write requests for the tablets it manages
- Splits tablets that have grown too large



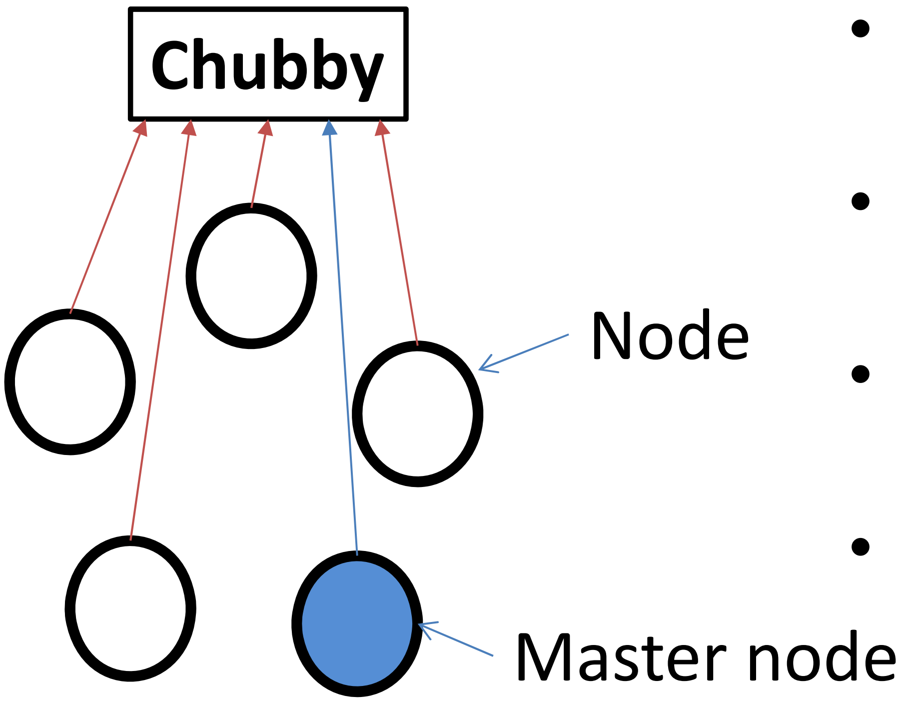
BigTable Building Blocks

- Chubby *(cf. Paxos Lecture)*
  - Lock service
  - Metadata storage
- GFS *(cf. GFS et al. Lecture)*
  - Data, log storage
  - Replication
  - Uses Sorted Strings Table files (SSTables)
- Scheduler
  - Monitoring
  - Failover



Chubby Lock Service

Highly-available, persistent, distributed lock, coordination service



Sample use in BigTable

- Ensure at most one active BigTable master at any time
- Store bootstrap location of data (root tablet)
- Discover tablet servers (manage their lifetime)
- Store schema information

Chubby Lock Service

Hig
coo

Cf. Coordination and Agreement Lecture

Cf. The Paxos Consensus Algorithm Lecture

Cf. Coordination with Zookeeper Lecture

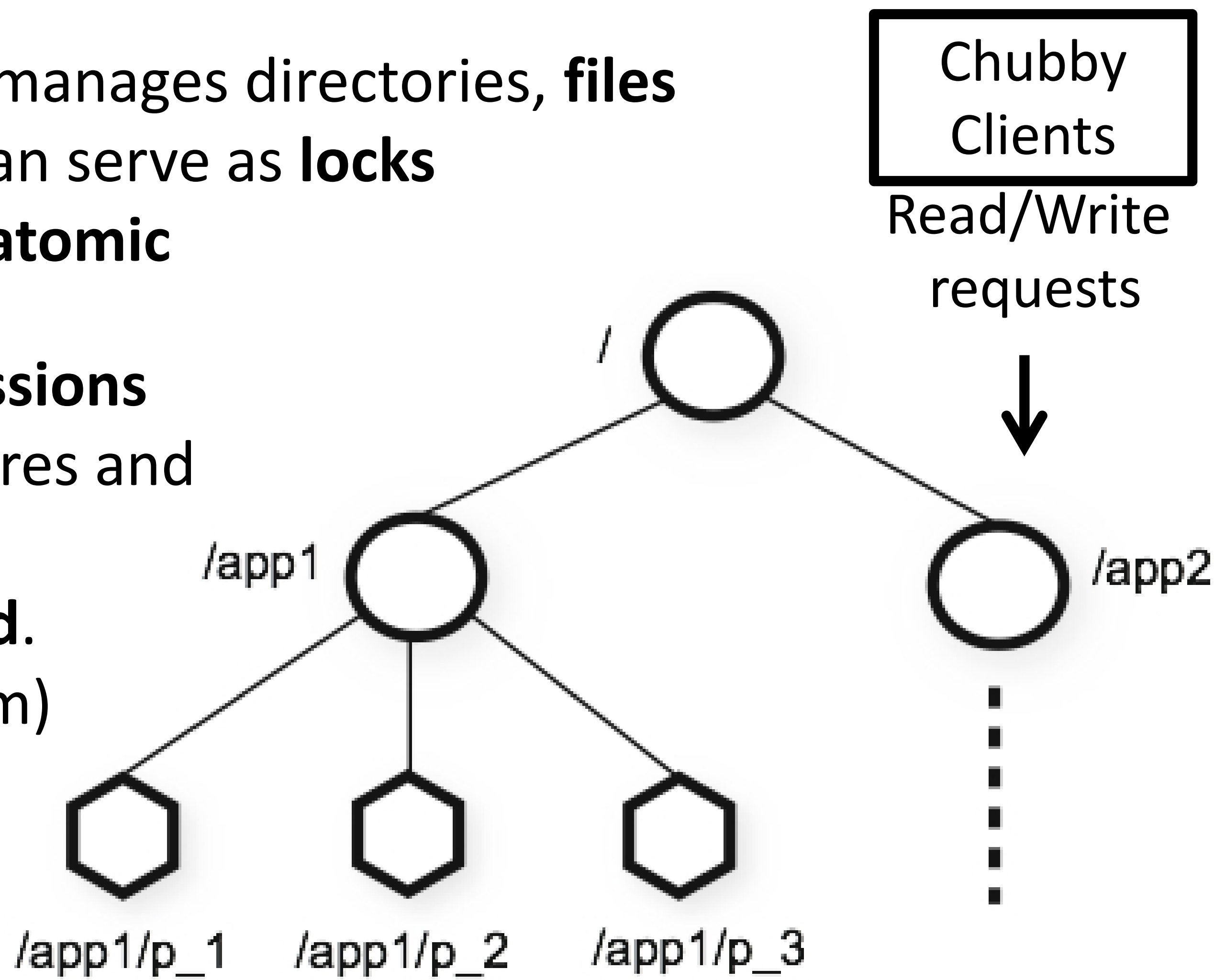
active
ny time
tion of

ers
ne)
ation

Lock Service Operational Model

- Knows about and manages directories, **files**
- Directories, files can serve as **locks**
- Reads, writes are **atomic**

Clients maintain **sessions**
If session lease expires and can't be renewed, **locks** are **released**.
(timeout mechanism)



Lock Service Availability

- Comprised of **five** active **replicas**
  - **Consistently replicate writes (***cf. Replication Lecture***)**
- One replica is designated as master
  - Need to **elect** master (**leader**) (*cf. Coordination Lecture*)
  - Chubby master is different from BigTable master!
- Service is up when:
  - Majority of replicas are running and
    - A **quorum** of replicas is established
  - Can communicate with one another

Core Mechanisms

- Ensure one active BigTable master at any time
  - **Leader election**, but in a distributed setting
  - *Cf. Coordination and Agreement Lecture*
- Keep replicas consistent in face of failures
  - **Paxos algorithm** based on replicated state machines (RSM)
  - Atomic broadcast
  - *Cf. Paxos Lecture, Replication Lecture*

Chubby Example: leader election

- Electing a leader node: supported by acquiring an exclusive lock on a file (**clients represent partaking nodes**)
- Clients concurrently **open a file** and attempt to acquire the file lock in write mode
- One client **succeeds** (i.e., becomes the **leader**) and writes its name to the file
- Other clients **fail** (i.e., become **replicas**) and discover the name of the leader by reading the file

Chubby Example: leader election

```
Open("/ls/cell1/somedir/file1",
      "write mode")

if (successful) {   // leader
      setContents(primary_identity)
} else {            // replica
      Open("/ls/cell1/somedir/file1",
            "read mode",
            "file-modification event")
      On modification notification
            primary = getContentsAndStat()
}
```

obtain file handle

write to file

subscribe to modification event

read from file

Apache HBase

- Open-source implementation of BigTable
- E.g., Facebook Messenger uses Hbase

- Different names for similar components
  - GFS → HDFS
  - Chubby → Zookeeper
  - BigTable → Hbase
  - MapReduce → Hadoop

BigTable vs. MapReduce

BigTable

- Layered on top of GFS
- Data storage and access
- BigTable: read/write web data

MapReduce

- Layered on top of GFS
- Batch analytics
- MapReduce: offline batch processing *(cf. MapReduce Lecture)*

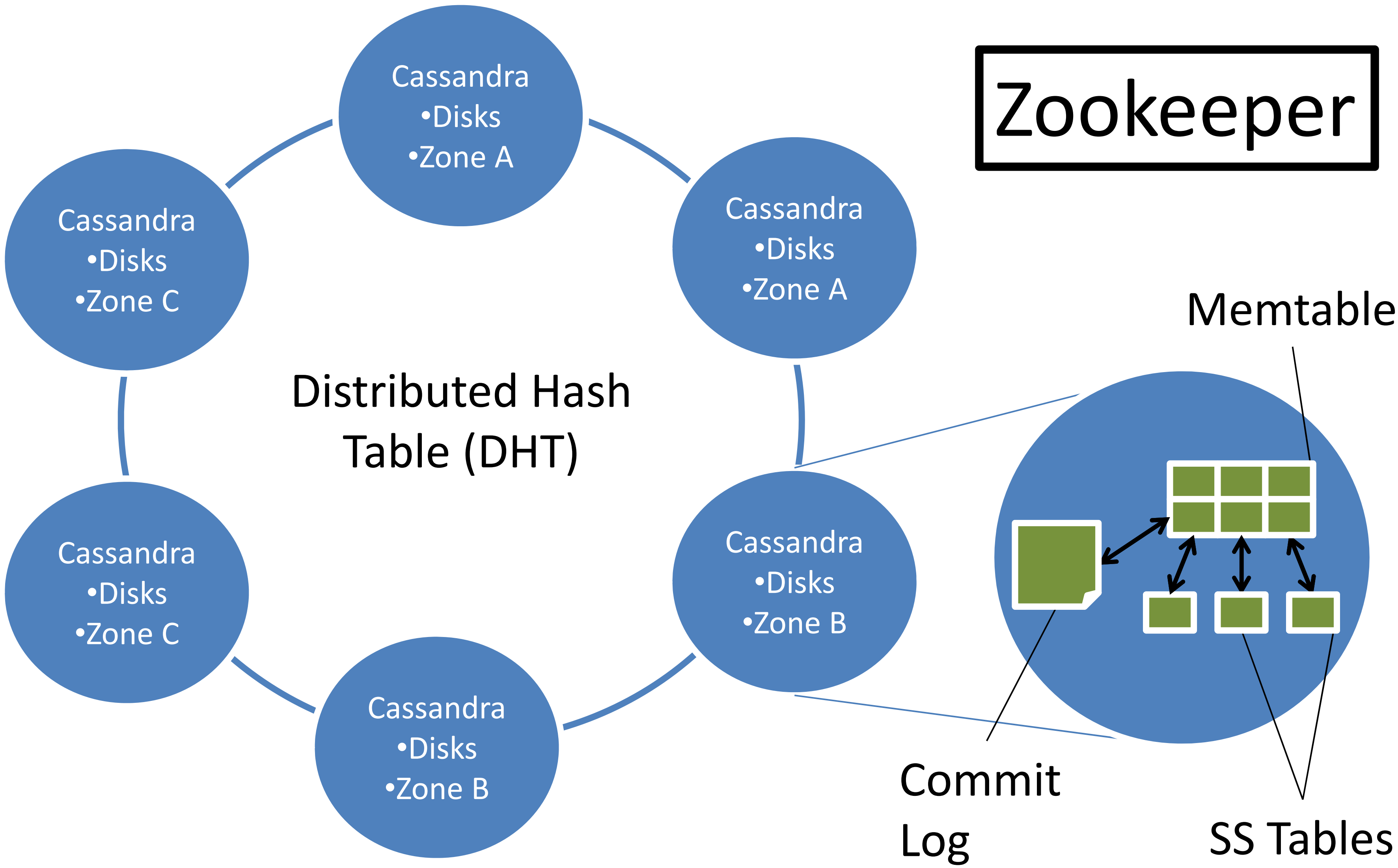
Google File System: common persistent storage layer *(cf. GFS et al. Lecture)*

DYNAMO / CASSANDRA

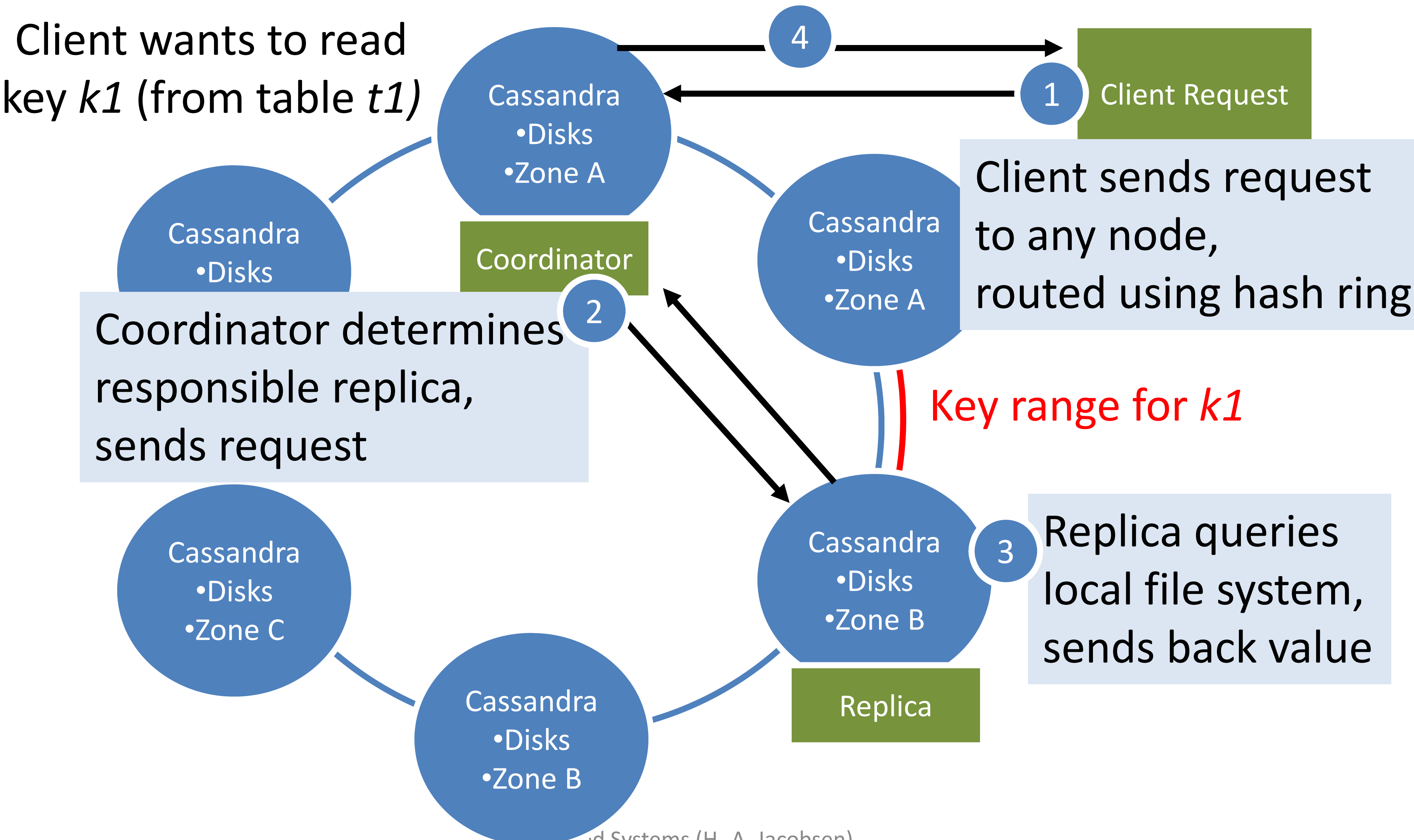
Cassandra

- Developed by Facebook
- Based on Amazon Dynamo (but open-source)
- Structured storage nodes (**no GFS** used)
- **Decentralized** architecture (no master assignment)
- **Consistent hashing** for load balancing
- Eventual consistency
- **Gossiping** to exchange information

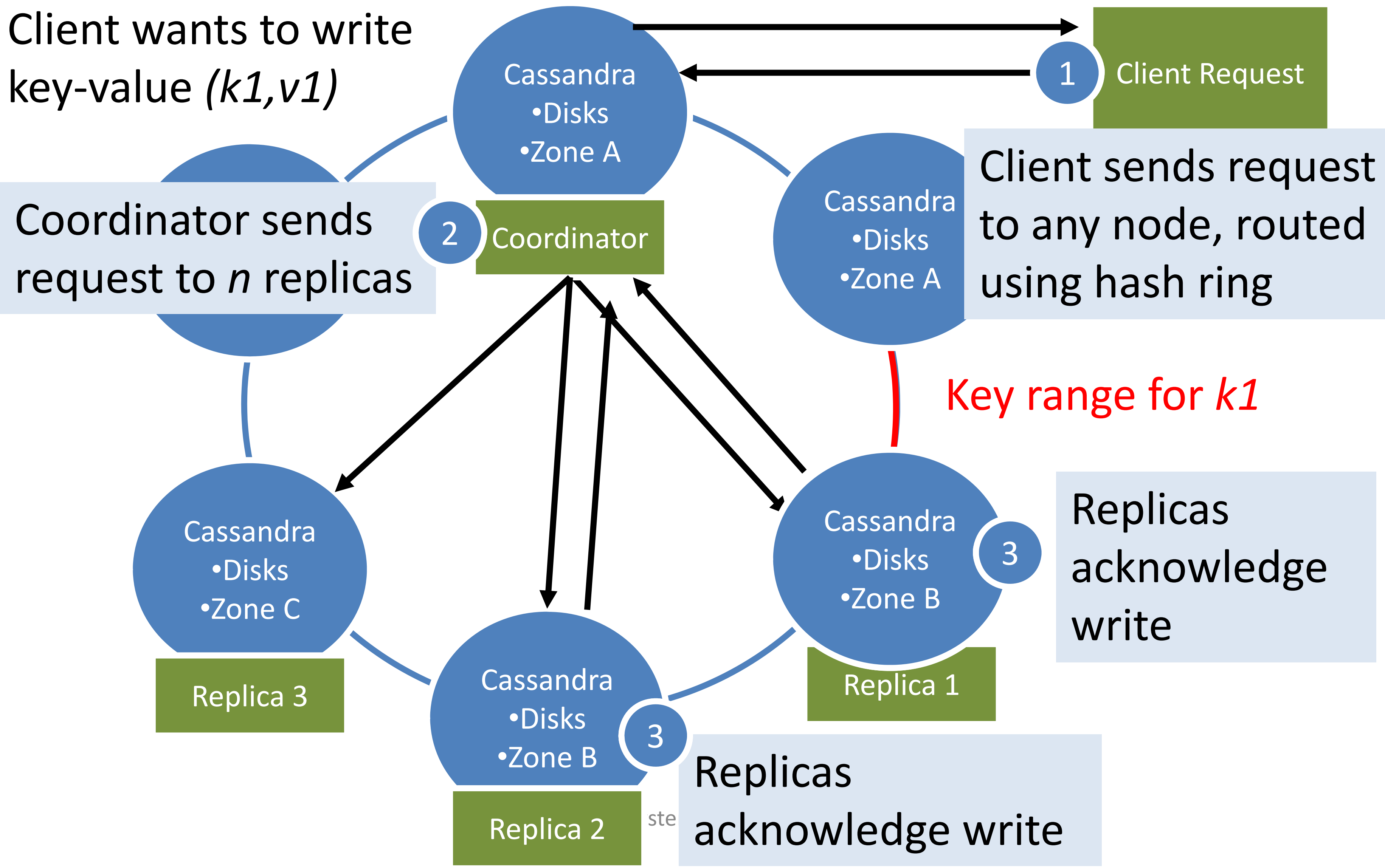
Cassandra Architecture Overview



Cassandra global read-path

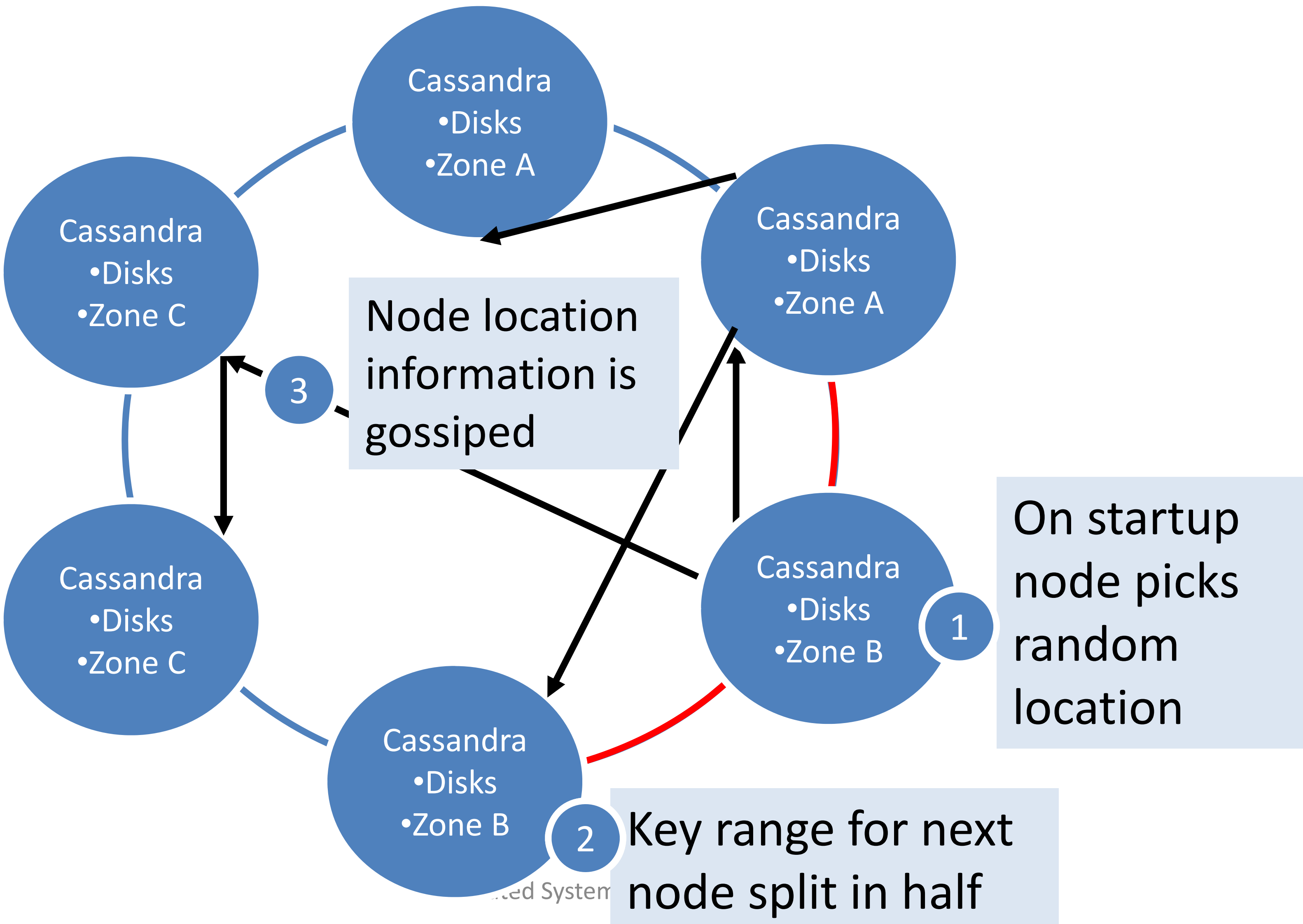


Cassandra Global Write-path

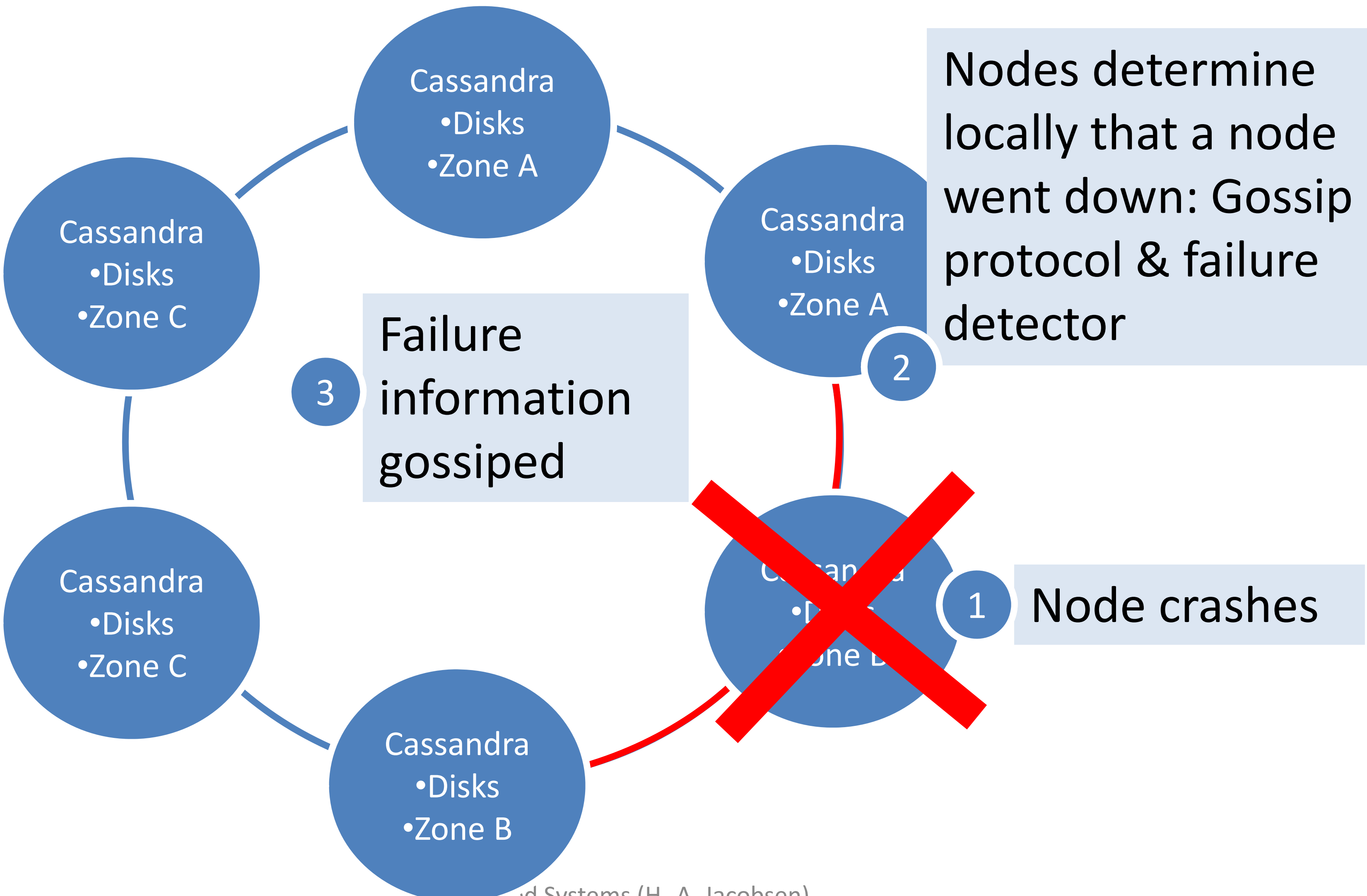


Incremental Scaling in Cassandra

(i.e., adding a storage unit)



Storage Unit Failure



Core Mechanisms

- Decentralized load balancing and scalability
  - *Cf. Consistent Hashing Lecture*
- Read/write reliability
  - *Cf. Replication Lecture*
- Membership management
  - *Cf. Gossip in Replication Lecture*
- Eventual consistency model
  - *Cf. Consistency Lecture*

Tentative Course Outline

- **Time** in distributed systems
- **Coordination** and **agreement**
- **Consensus** with Paxos
- **Replication**
- **Consistency** and transactions
- Consistent hashing, **CAP theorem**, web caching
- Distributed file systems (GFS)
- **MapReduce**, Spark
- Peer-to-peer systems, **distributed hash tables (DHTs)**
- **Blockchains**