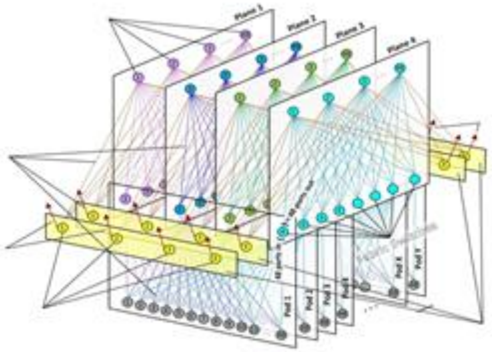
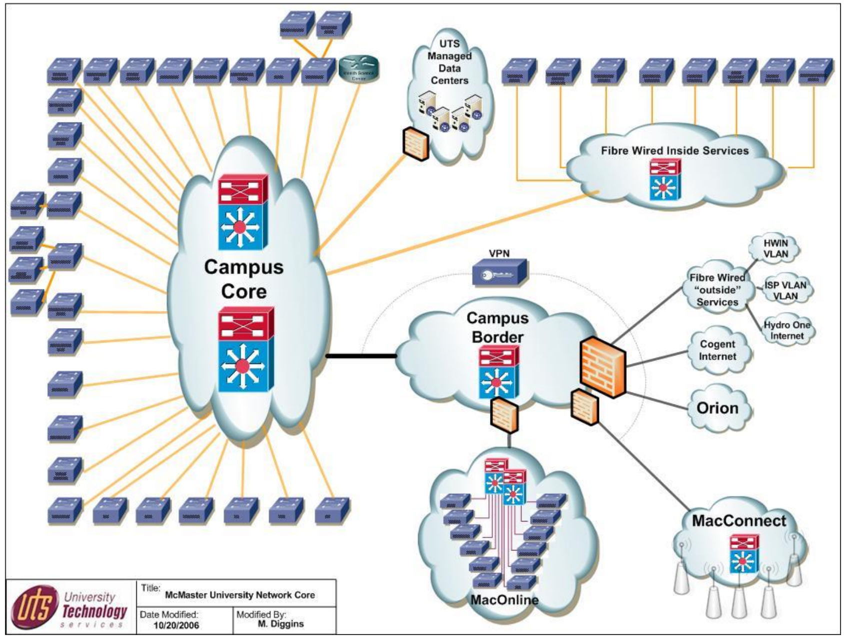
UNIVERSITEIT VAN AMSTERDAM

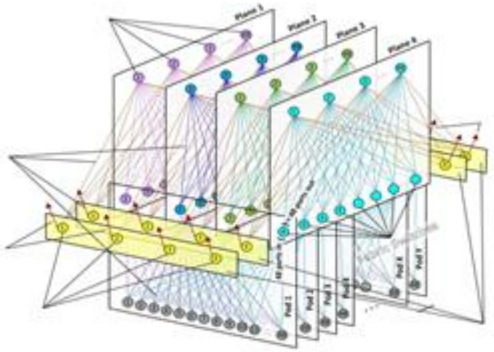


Large Systems:

Design +
Implementation +
Administration

2024-2025

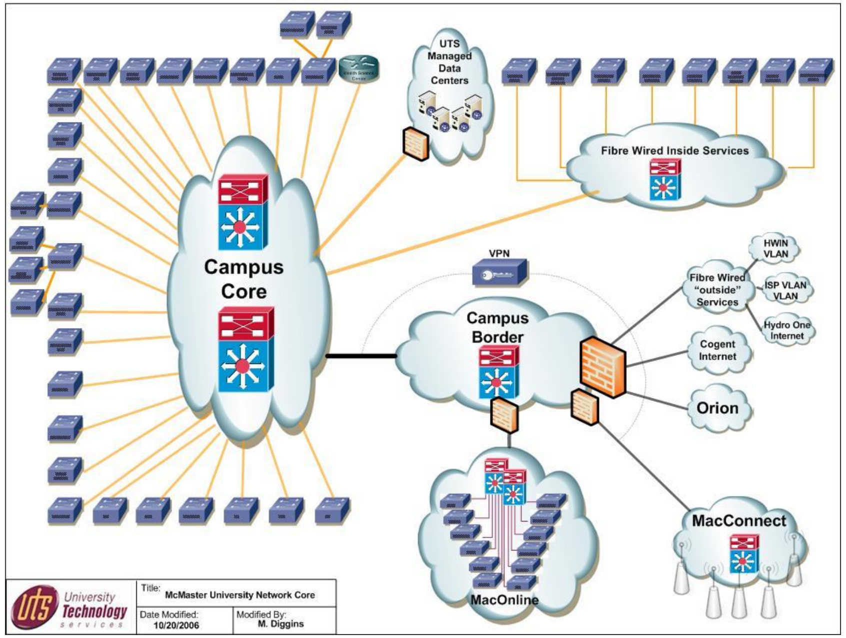




Large Systems:

Design + Implementation:

- Week3-L6: Replication, Caching & Partitioning



Shashikant Ilager
shashikantilager.com

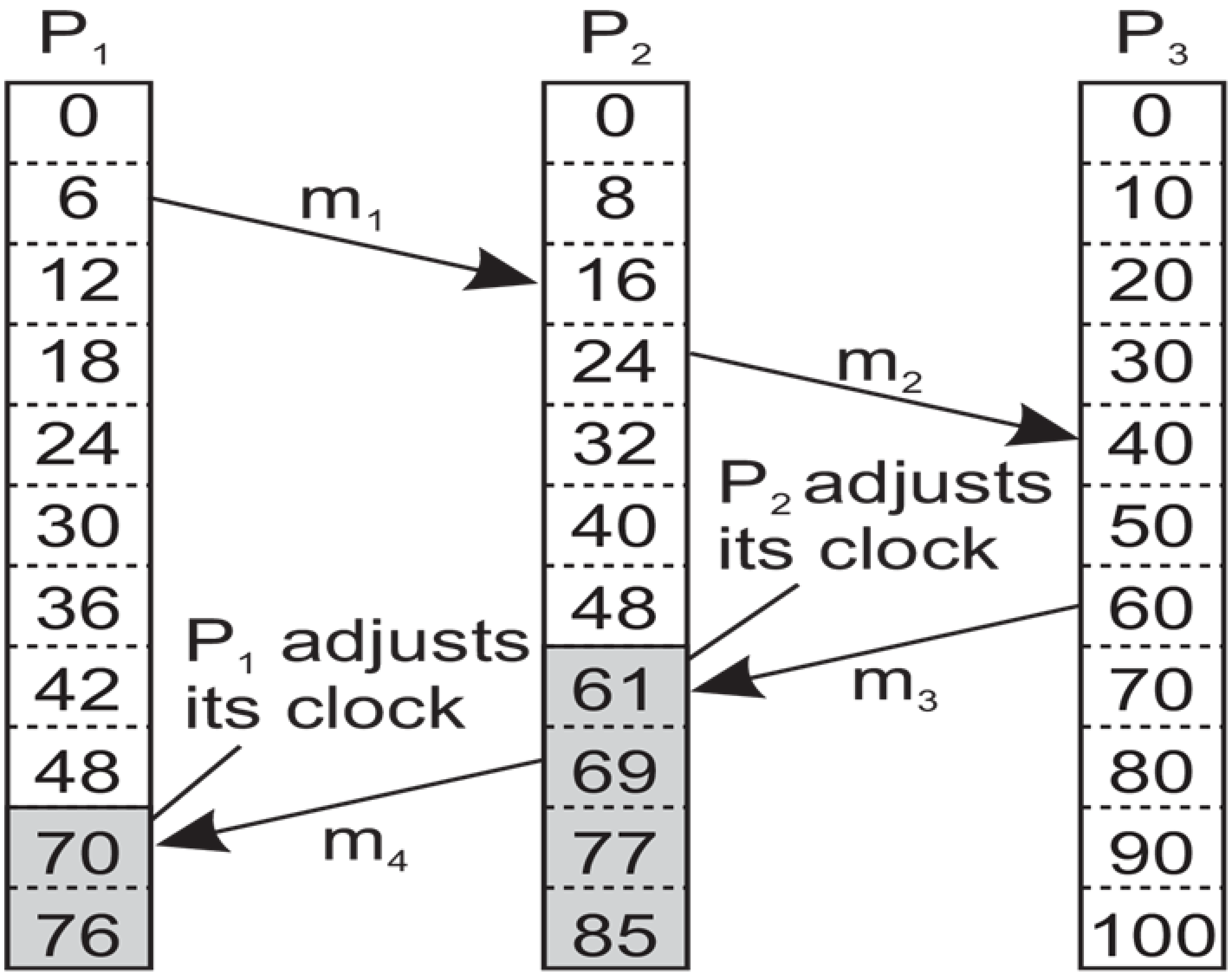
Admin tasks

- Paper presentation
  - Make sure you have already finalized the group member and paper
- Data Center visit signup [November 21st]
  - Use the canvas calendar to sign up for the slot. I see still one slot completely empty
- Updated schedule
  - We have updated the lecture schedule
  - Nov 25th, AWS personal guest lecture

Recap…

- Communication abstractions
  - RPC, RMI, Middleware
- Coordination
  - Message ordering: Casual and total order
  - Logical clock and Lamport’s clock synchronization algorithm

Recap…



Source: DS3, Fig. 6-8b

Scaling Techniques

- Bigger machines
- Virtualization
- Asynchronous communication
- Replication & Caching
- Partitioning

Replication (Ch. 7)

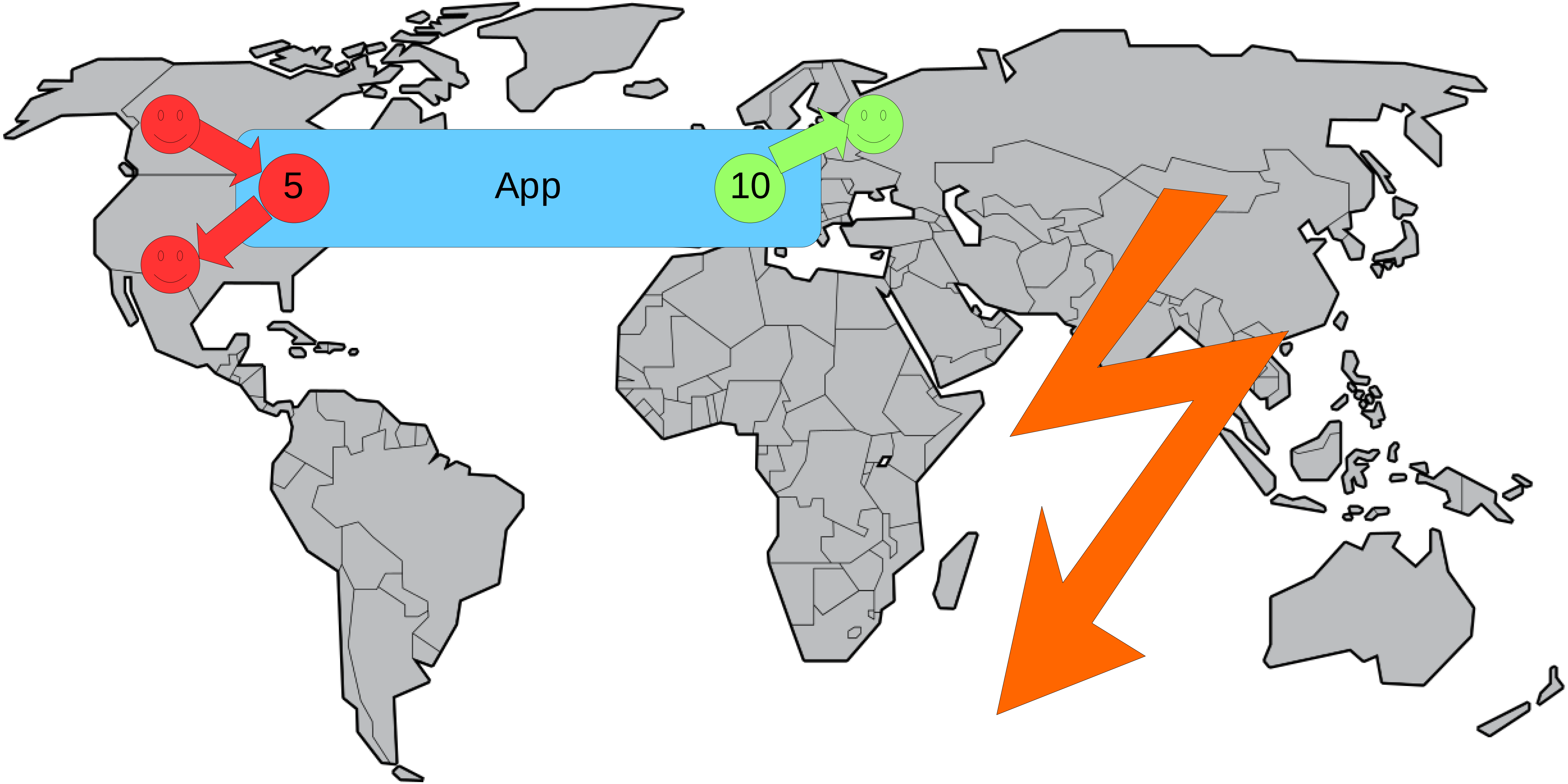
- Duplicate data or functionality on another server
- Reason: Performance
  - Scale in size: More capacity
  - Scale geographically: Closer to client = lower latency
- Reason: Redundancy
  - Have other copies when a component fails

Replication

- Redundancy → Availability
  - Availability also means the ability to progress
  - Not only read data
  - Also able to do writes (e.g., If you can access your bank app, but you cannot do transactions, then service is unavailable)

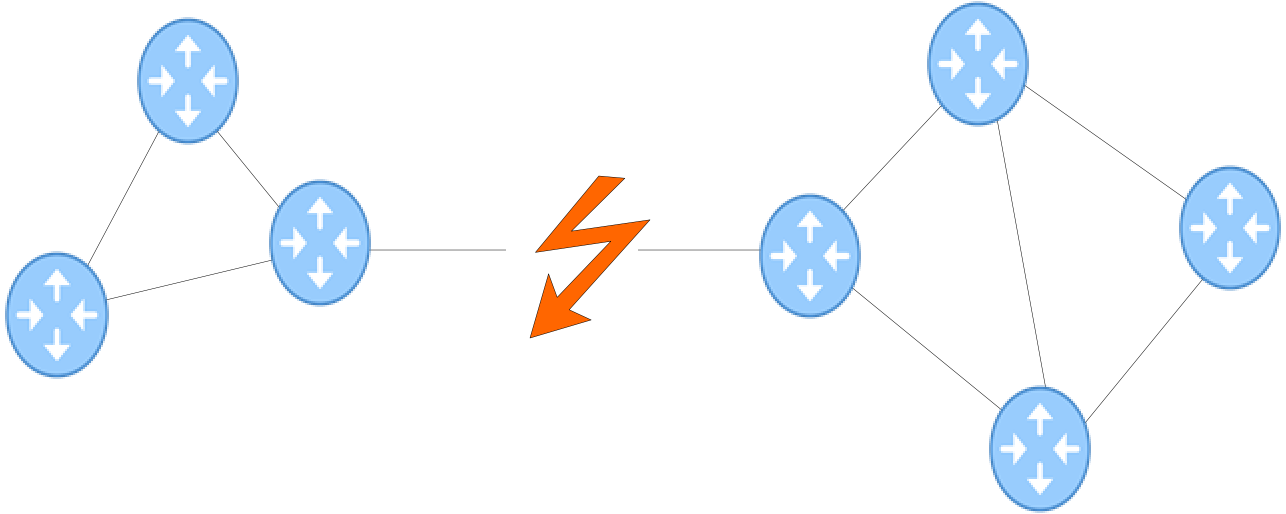
Replication Issues 1

- Problem 1: Replicas must be kept consistent
  - Make sure clients see the same data
  - Distribution transparency!
  - Strong consistency costs performance
  - Delay all operations until all replicas are the same


5
App
10

Replication Issues 2

- Problem 2: Network may become partitioned
  - How to keep replicas consistent if in different partitions?
- Want Consistency, Availability, and Partition (CAP) Resistance for replicated data

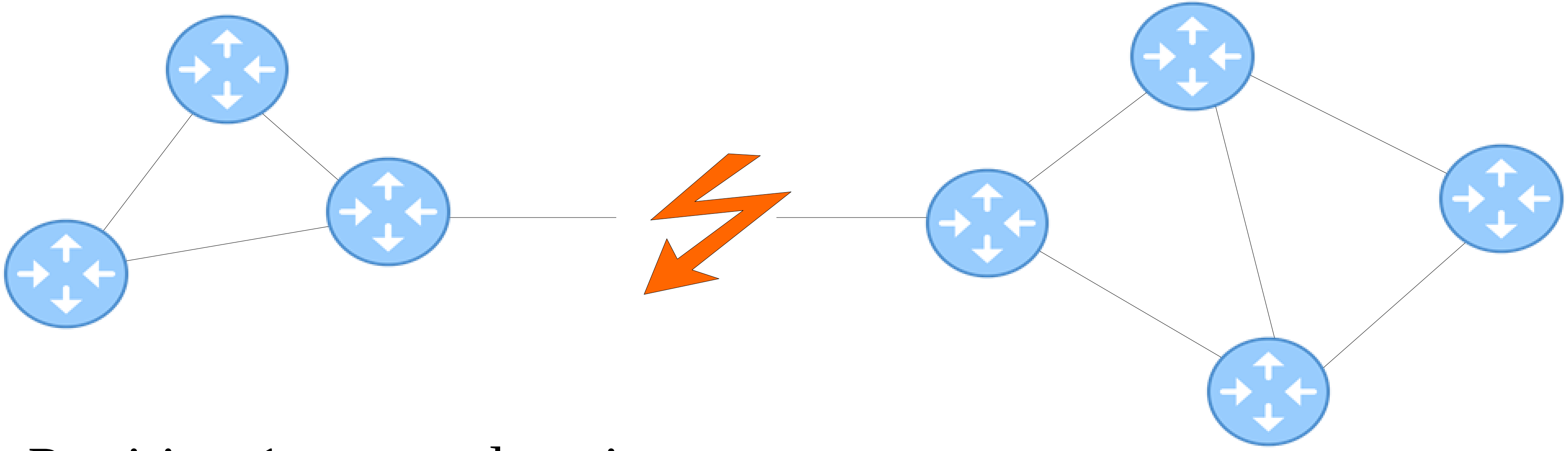


CAP Principle

- CAP Principle / Theorem
  - Can only have 2 out of 3!

See: A. Fox and E. A. Brewer, "Harvest, yield, and scalable tolerant systems," Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, 1999, pp. 174-178, doi: 10.1109/HOTOS.1999.798396.

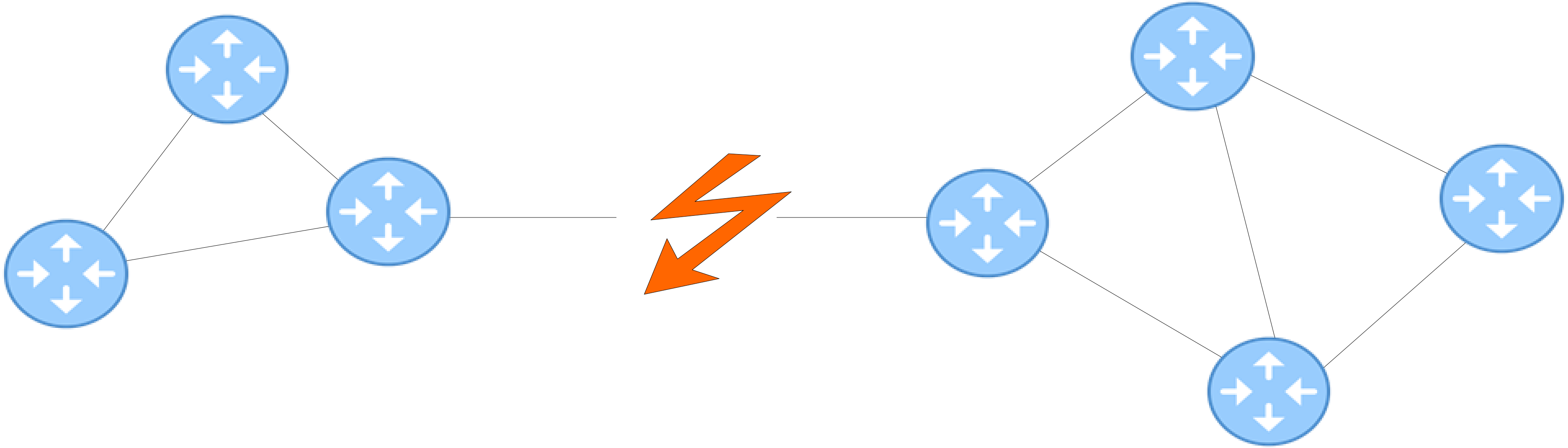
C+P



Partition 1 can read+write
Partition 2 must not respond

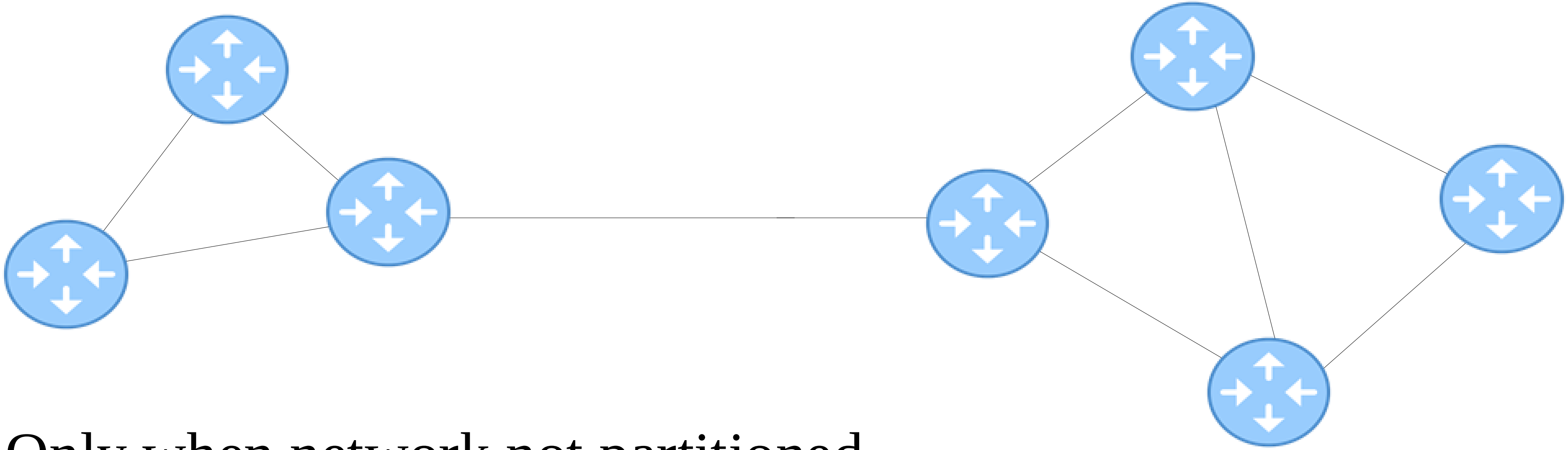
Or: All go read-only-> No availability, as no writes means no progress

A+P



Both partitions respond to reads
Both partitions can do updates (write)
Fix inconsistencies later

C+A



Only when network not partitioned

CAP Principle

- C+A
  - Traditional database
  - MySQL, Spanner
- C+P
  - Read-only, or non-responding when partitioned
  - MongoDB, HBase, Redis, Memcachedb
- A+P
  - Always respond, even when outdated
  - CouchDB, Voldemort, Cassandra

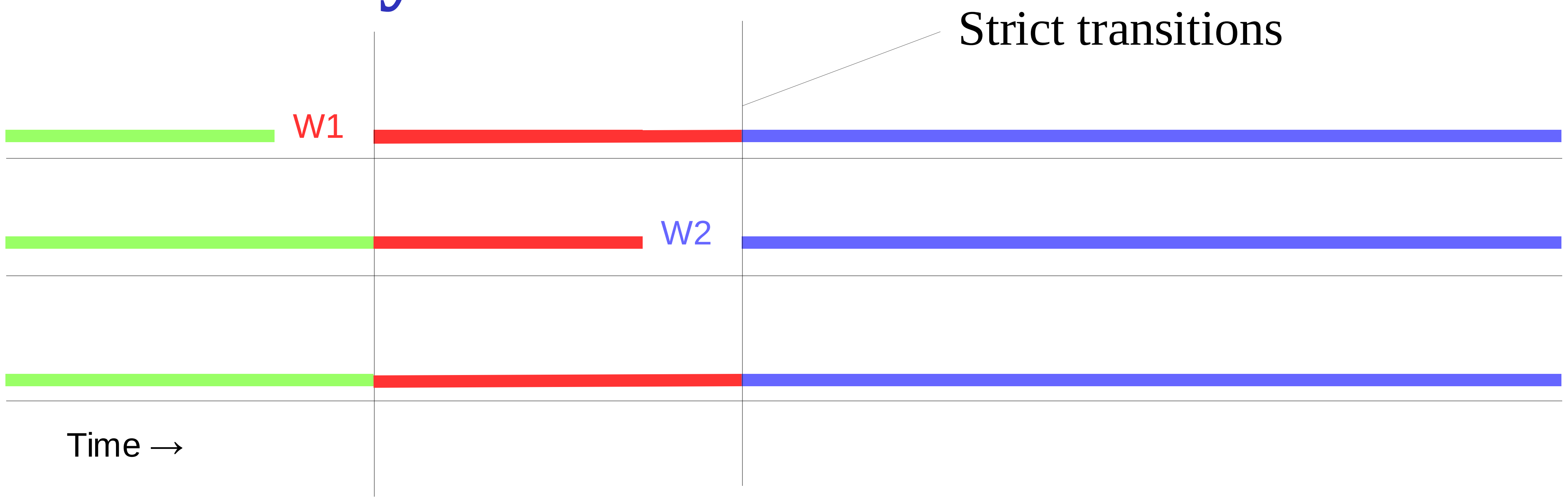
Implication

- Have to make a choice, dictated by the application
- Can it handle weaker consistency?
  - Bank account: No
  - Facebook/insta posts: Yes
- Acceptable to be unavailable for a while?

Consistency Models

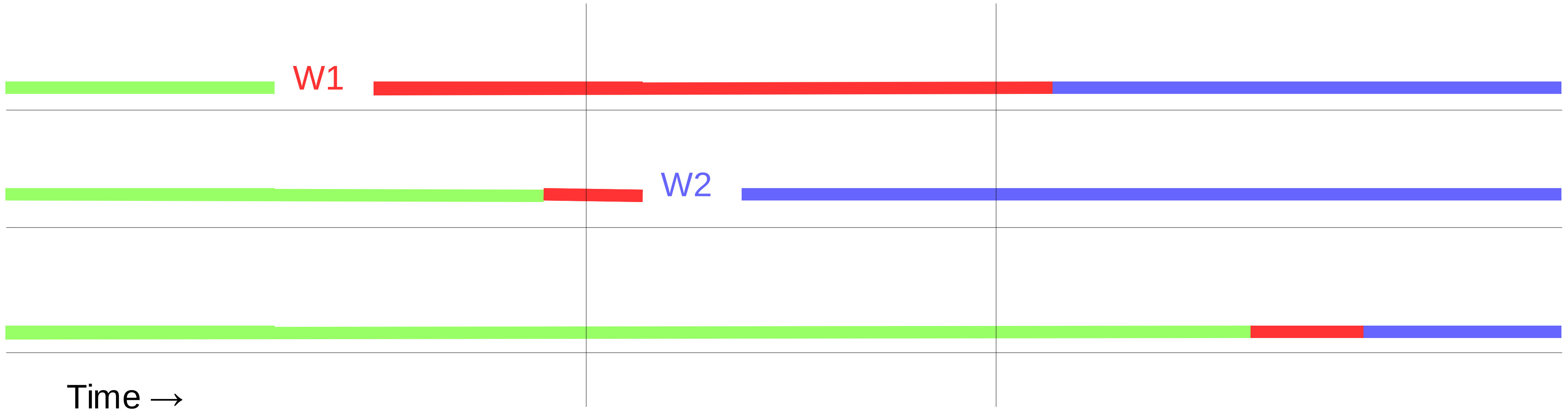
- Linearizability
  - Clients see atomic writes in same order
- Sequential consistency
  - Clients see writes in same order
- Causal consistency
  - Clients see causally related writes in same order, unrelated writes possibly in different order
- Eventual consistency
  - If few or resolvable write-write conflicts
  - Eventually all replicas converge
  - Client may see differences in meanwhile

Linearizability



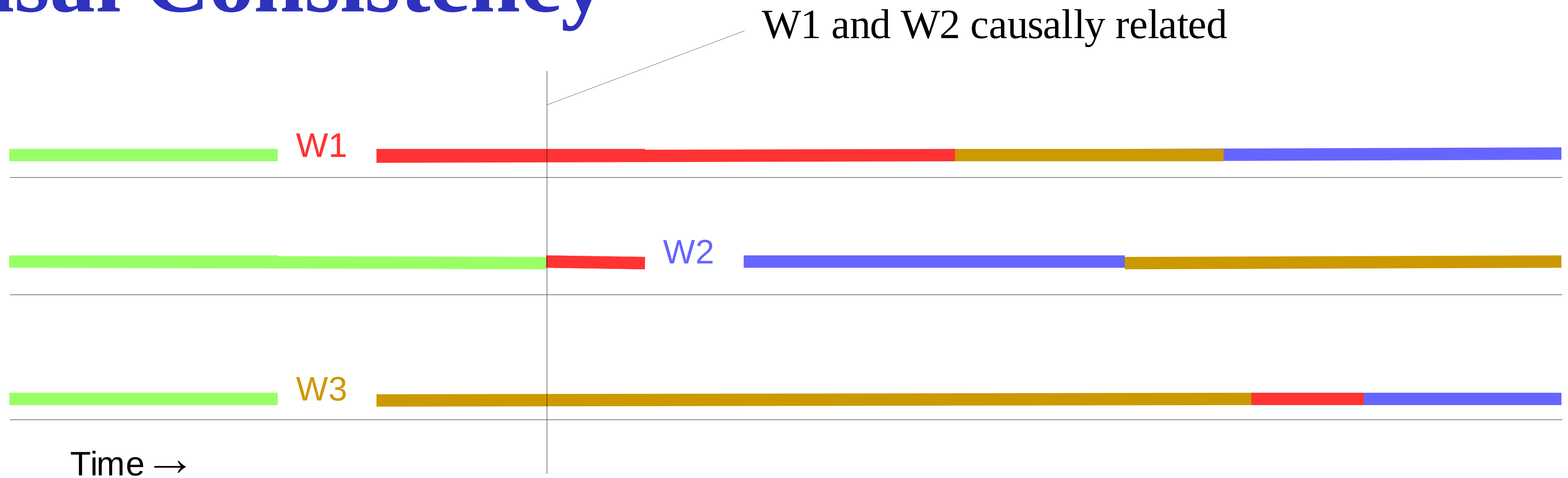
- Process P1, P2, P3 all hold replica of data item X
- Colours indicate what each process would read as value of X.
- Linearizability: All processes must read same value at all times

Sequential Consistency



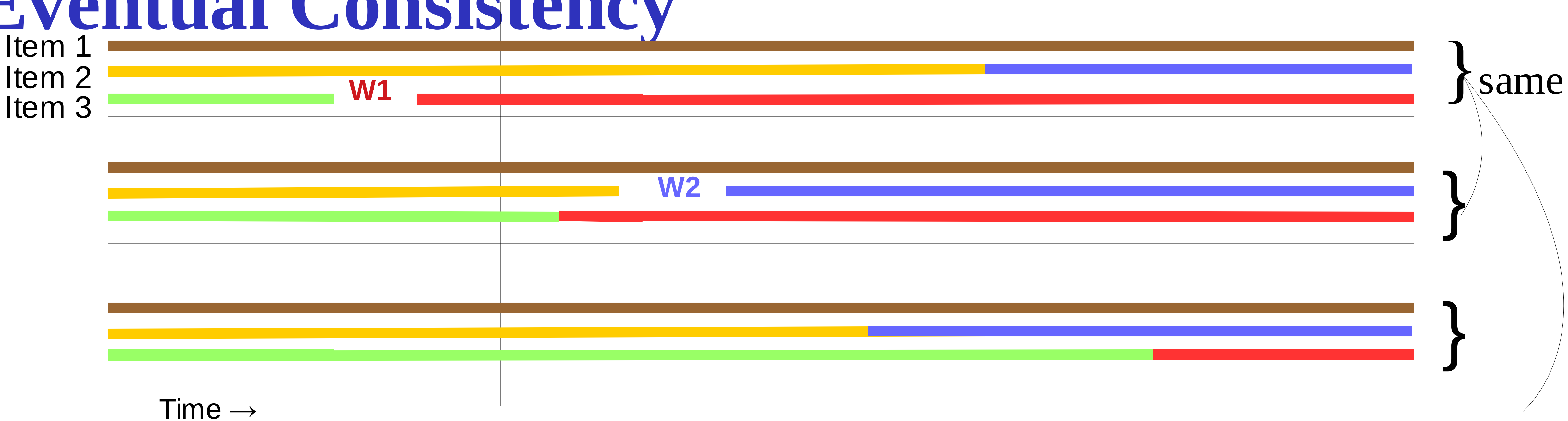
- SeqCons: Processes may read different colours at different times
- As long as the global (and per-process) order of writes is the same (Green, Red, Blue)

Causal Consistency



- W2 depends on W1, i.e. W1 and W2 causally related
- CausalCons: All processes must see W1 before W2 (Red before Blue)
- Concurrent writes, e.g. W3 may be seen in diff order

Eventual Consistency



- Simple EvtCons: Processes write to different items (few write-write conflicts)
- Processes may read different colours at different times
- As long as eventually the colours become all the same

Examples Consistency Models

- Bank account:
  - Linearizability
  - All processes must see same balance
- C+A Database:
  - Sequential consistency
  - Global order on transactions
- A+P Database:
  - Eventual consistency
- Facebook: Causally + eventually consistent
  - You do not see comments before post
  - May not see all comments right away

Implementing Consistency Models

- Linearizability:
  - Global lock on replicas [temporal unavailability of system]
  - Not scalable
- Sequential Consistency:
  - Need total order on writes
  - Hard, but some capacity possible [DS3, p.401]
- Causal Consistency
  - Causal relations via Vector clocks [DS3, p.316]
  - Hard, need per-process info

E.g. Sequential Consistency

- Sequential Consistency: Impose total order
  - Use MutEx solutions
    - Single coordinator (sequencer)
    - Logical Clocks
  - Or combination thereof

Replication: Doubly Doomed?

1. Must to choose properties (C,A,P)
2. If we choose consistency
   - Does not scale
   - Slow and expensive

Doom in Reality

- Many apps can do with a choice of C,A,P
- If data is static, or servers stateless:
  - Replication still easy way to increase capacity
- Do apps need strongest consistency?

Example: Reservation System

- Read/write ratio of 400:1
- 100,000 items to book in the system
- 10% booked every day
- Assuming even distribution: once every 0.86 s
- Clients read from a read-only copy
- Synced once every 90 seconds from main DB

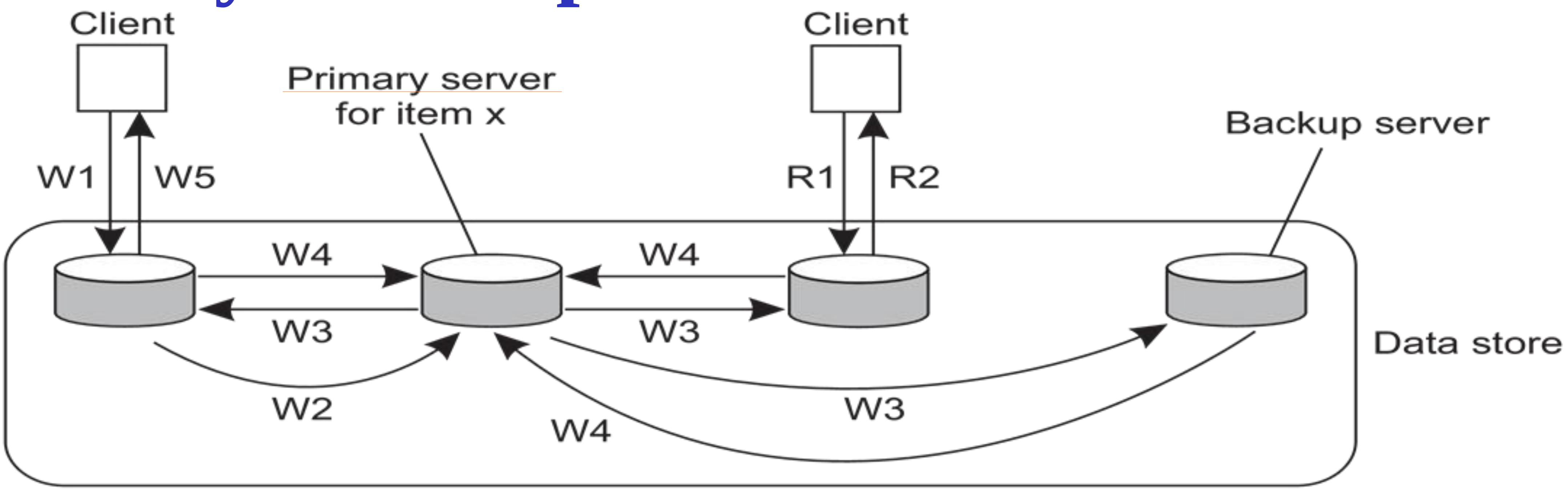
- Result: Chance of double booking is in the order of two decimal points (< 0.1 %)

Source: Scalability Rules, 2nd Ed. Abbott and Fischer, 2017, p. 23

Replication Protocols

- Primary / Backup
  - Primary is implicit sequencer
  - Variations:
    - Send write operations to backups
    - Send result of write operation to backups
    - Send invalidation to backups
- Active Replication
  - All replicas perform the write operation
  - Need explicit sequencer / total ordering
- Quorum-Based

Primary / Backup

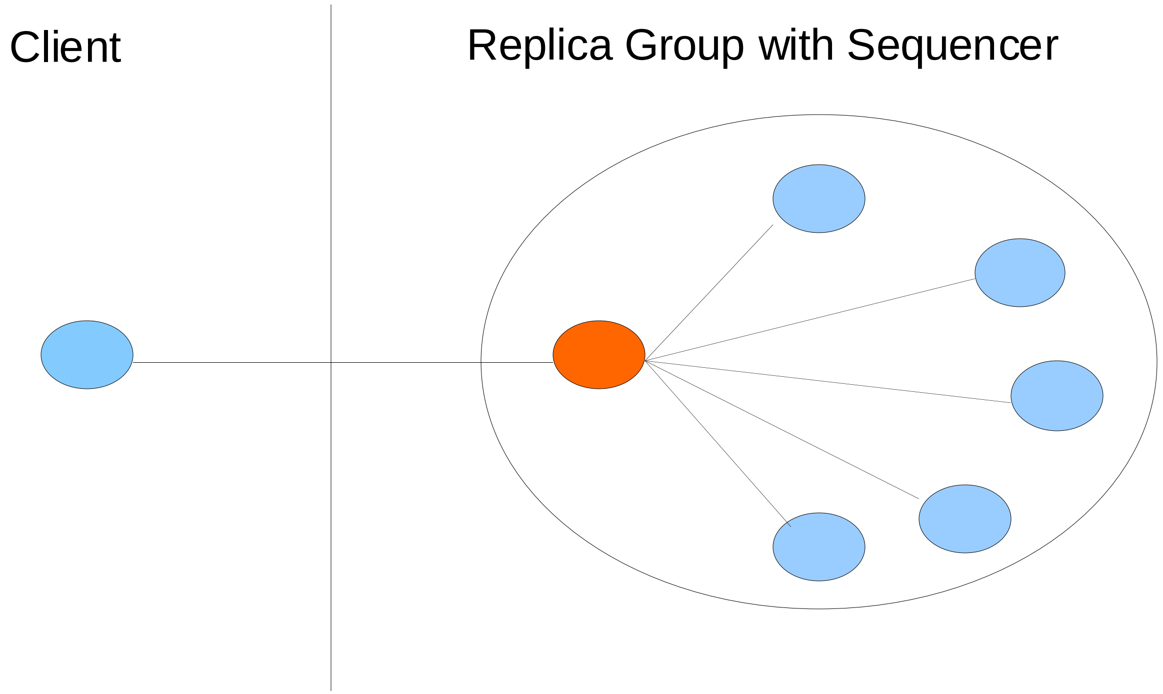


W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

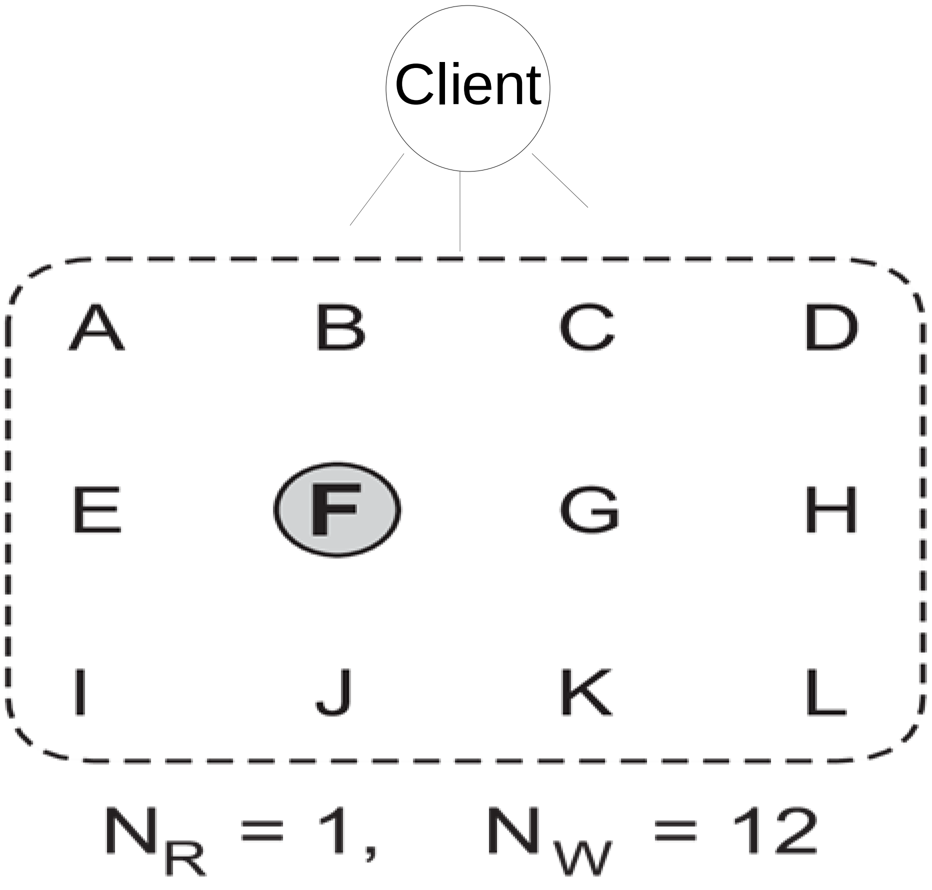
R1. Read request
R2. Response to read

Source: DS3, Fig 7-27

Active Replication

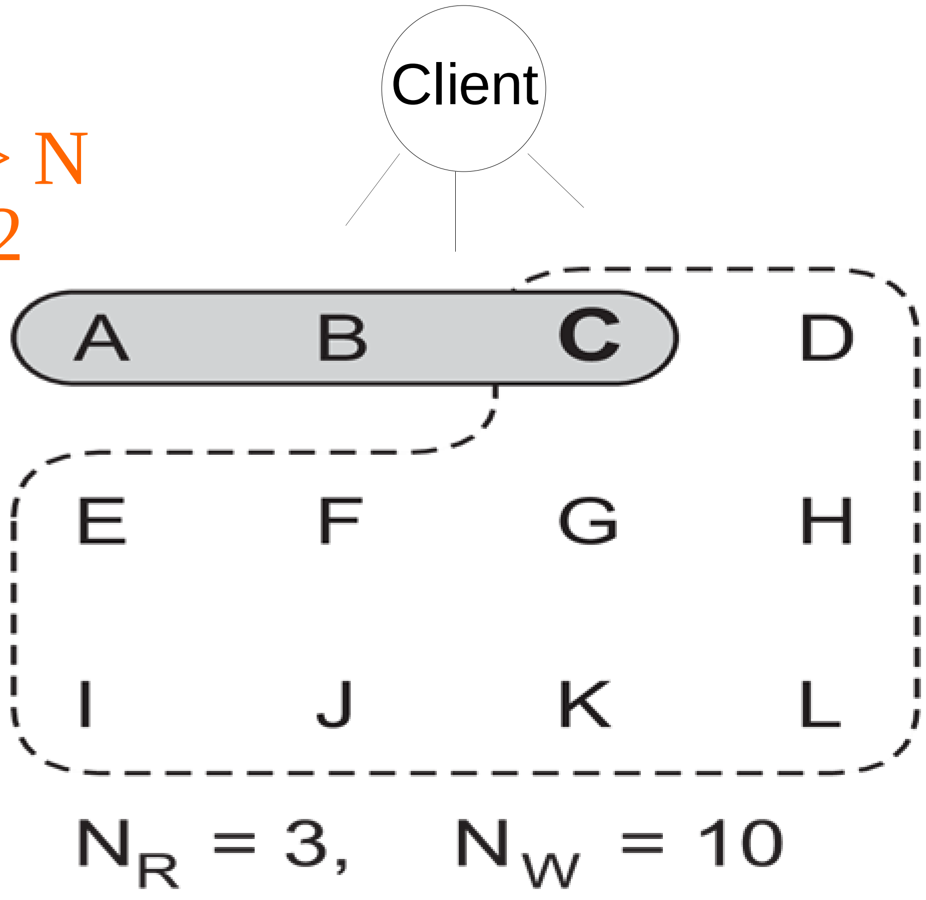


Quorum-Based Replication



Rules:

Nr + Nw > N
Nw > N / 2



Source: DS3, Fig 7-29 (c,a)

Questions to Answer about Replication

1. What to replicate (granularity)?
2. How to keep them consistent and at what level?
3. Where to place replicas?
4. How to direct clients to replicas?

Finding Replicas

- Global Redirection
  - AnyCast
  - DNS redirection

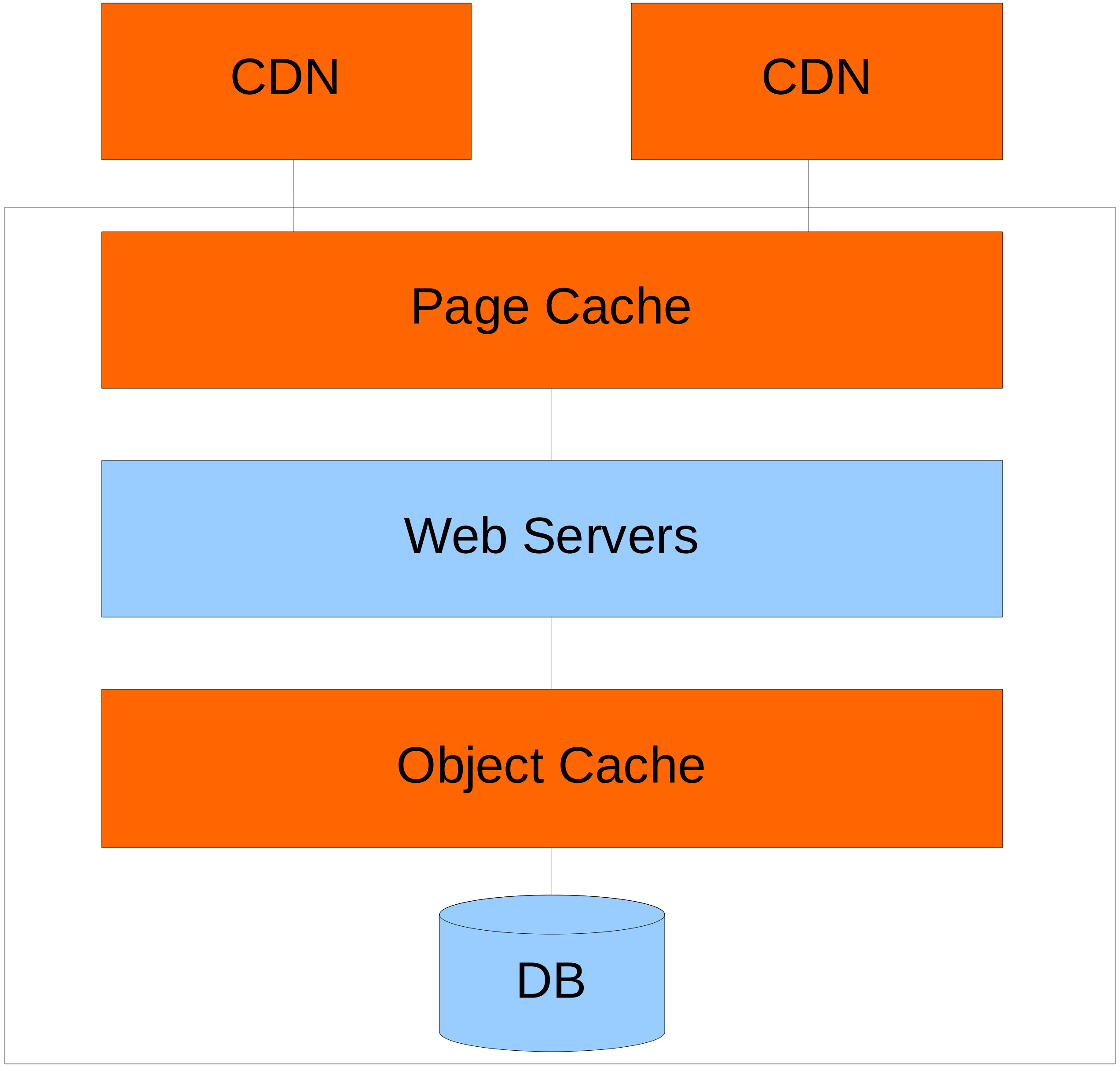
- Local Redirection:
  - In a cluster, the local load balancer directs to the server
  - DNS round-robin

Caching

- = Replicate temporarily
- E.g. results of queries
- When to expire?
  - Associate Time To Live (TTL)
  - Allow invalidations from authoritative source
- Very effective
  - Google Search Talk: hit rate 30-60%
  - Scalability Rules Book: “Cache is King”

Cache Control Mechanisms

- DNS?
- HTTP?



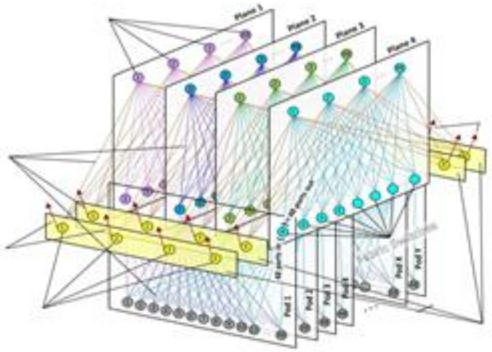
E.g., Static Content: Images, CSS

E.g., composed pages

E.g., queries from DB

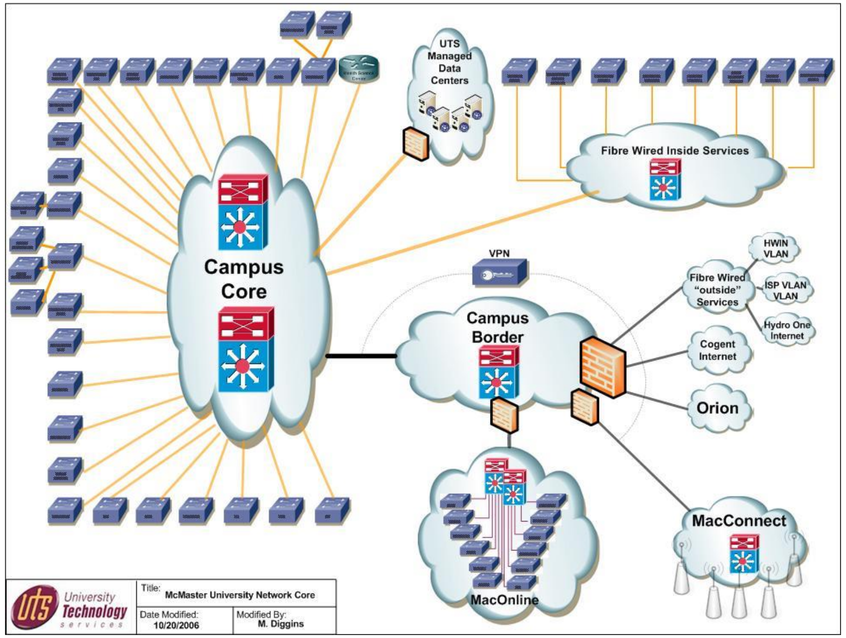
UNIVERSITEIT VAN AMSTERDAM

Large Systems:



Design + Implementation:

Partitioning



Shashikant Ilager
shashikantilager.com

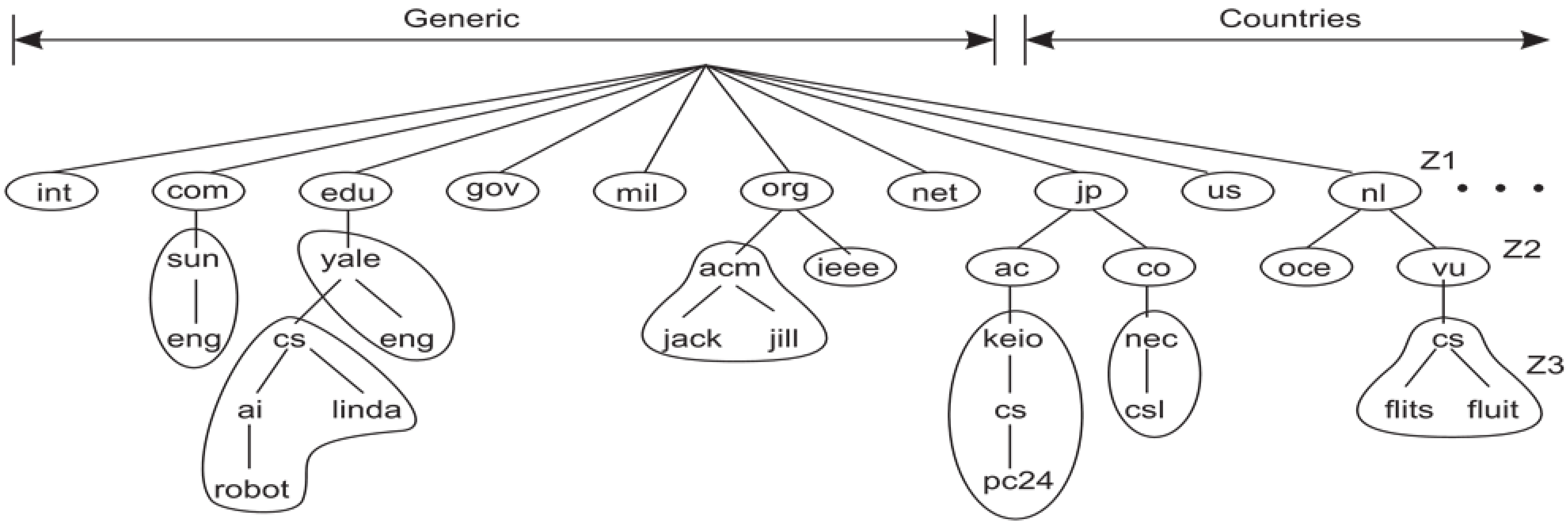
14 november 2024

Scaling Techniques

- Bigger machines
- Virtualization
- Asynchronous communication
- Replication & Caching
- Partitioning

Ways of Partitioning

- Split work/responsibilities over a set of servers
- Common method: Hierarchical
- E.g. DNS → Administrative scale!

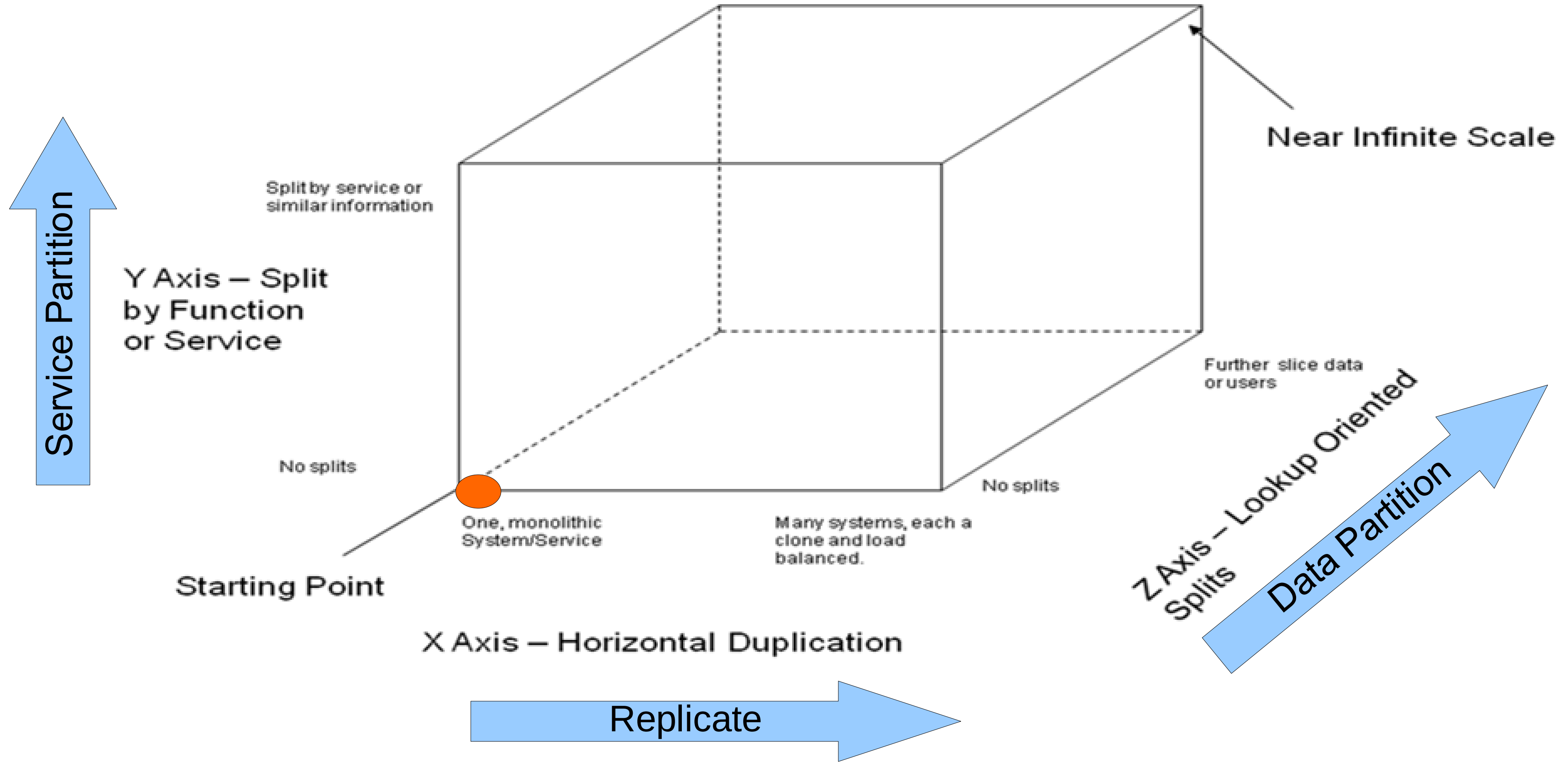


Source: DS3, Fig. 1-5

Ways of Partitioning (cont'd)

- Another common method: Formulaic
- Data is divided into parts according to some formula
- Parts are distributed to servers
  - Via another formula using the server ID
  - A meta server remembers which parts are where
- Check Consistent Hashing

Recall: AKF Scale Cube



Source: http://akfpartners.com/growth-blog/splitting-applications-or-services-for-scale

Recall: AKF Scale Cube

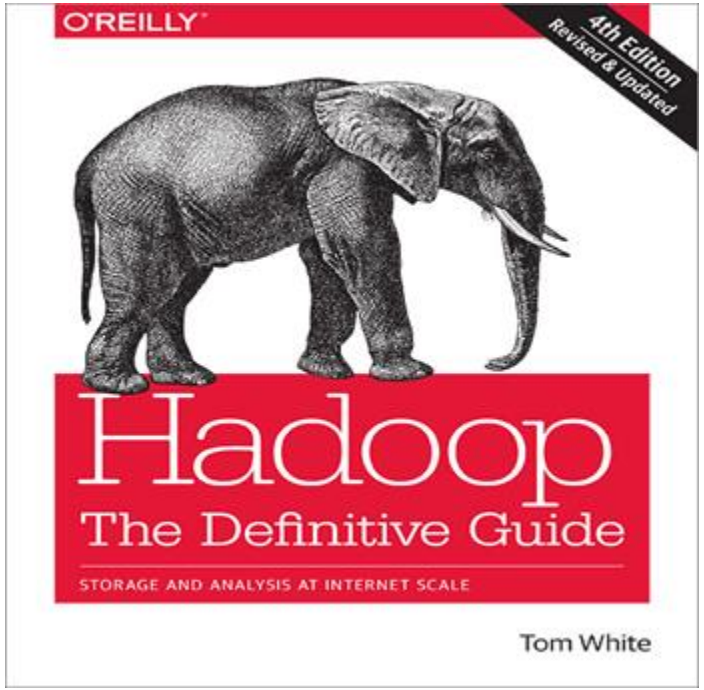
- Y-Axis: Service Partition: “Split different things”
  - Obvious: Web, DB, DNS separately
  - Others: signup, login, search (by verb)
  - Or: catalog, inventory, user accounts, marketing (by noun)

- Z-Axis: Data Partition: “Split similar things”
  - Split by Customer ID, last name, geographical location, device/network carrier, or some other property
  - Hierarchically
  - Splits often equal in size, e.g. by using hashing

Source: http://akfpartners.com/growth-blog/splitting-applications-or-services-for-scale

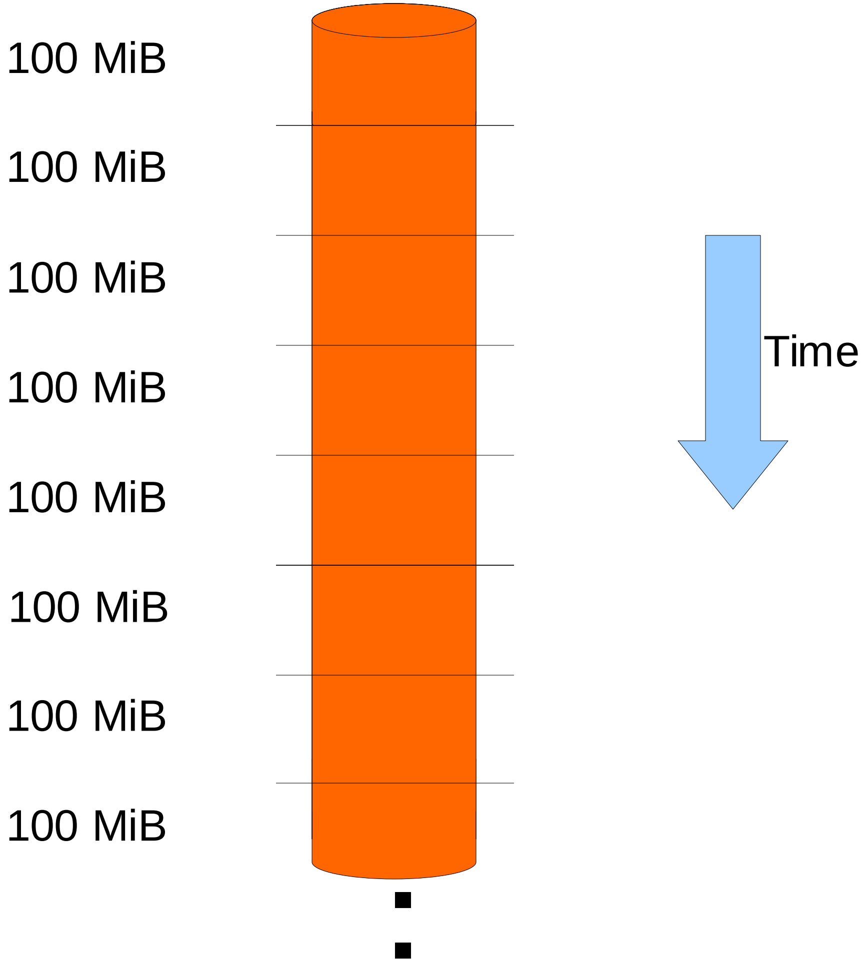
Case Study: Hadoop

- Spotify: 3,000 nodes, storing 100 petabytes of data ...
- Slides based on:
- "Hadoop: The Definitive Guide", 4th Ed. by Tom White

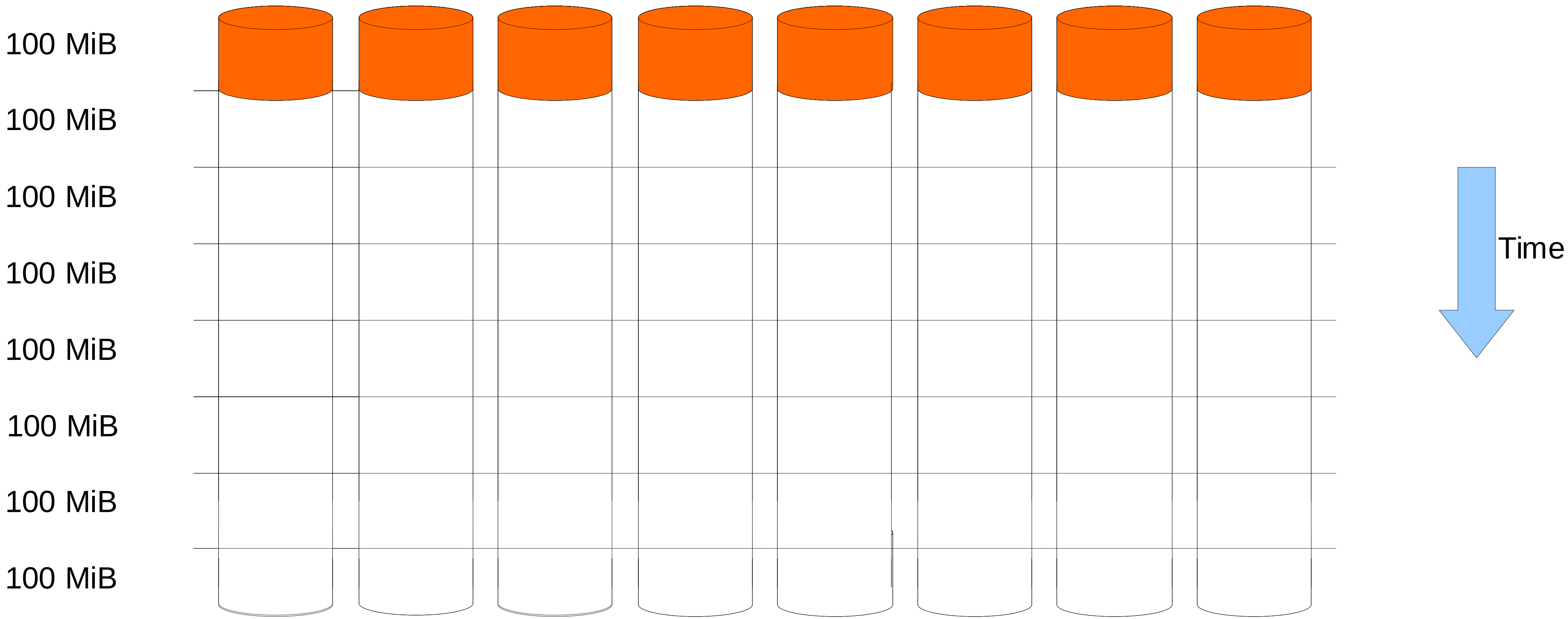


Reading Data

- Single magnetic disk can read ~100 MiB per second
- Takes ~3h to read a 1 TiB disk :-(



Reading Data (cont'd)



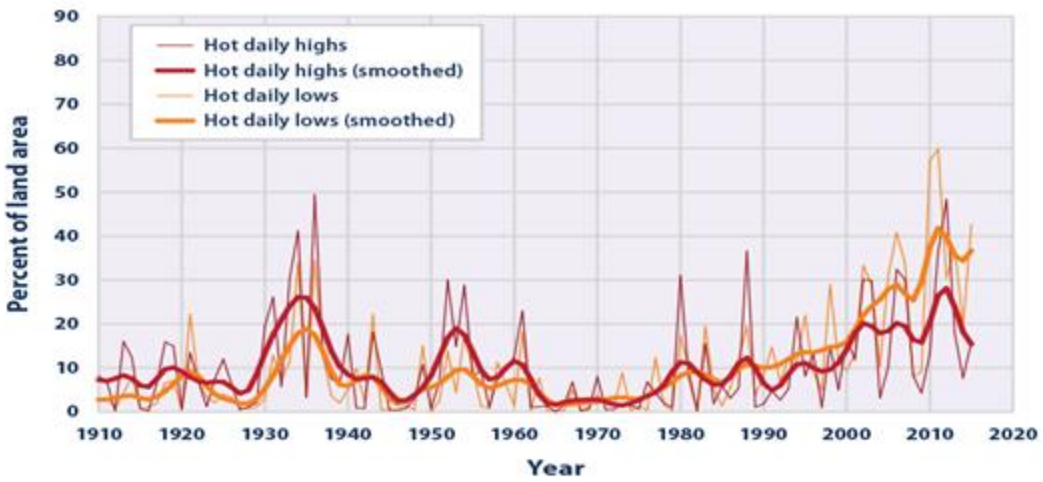
With N disks, reading is N times faster: 100 disks → 1 TiB / ~2 min!

Hadoop Idea

- Partition data over many disks
- Process data in parallel
- Components:
  - Various storage solutions
  - Most common: Hadoop File System (HDFS)
  - Various programming environments
  - Most common: MapReduce

Processing Weather Data

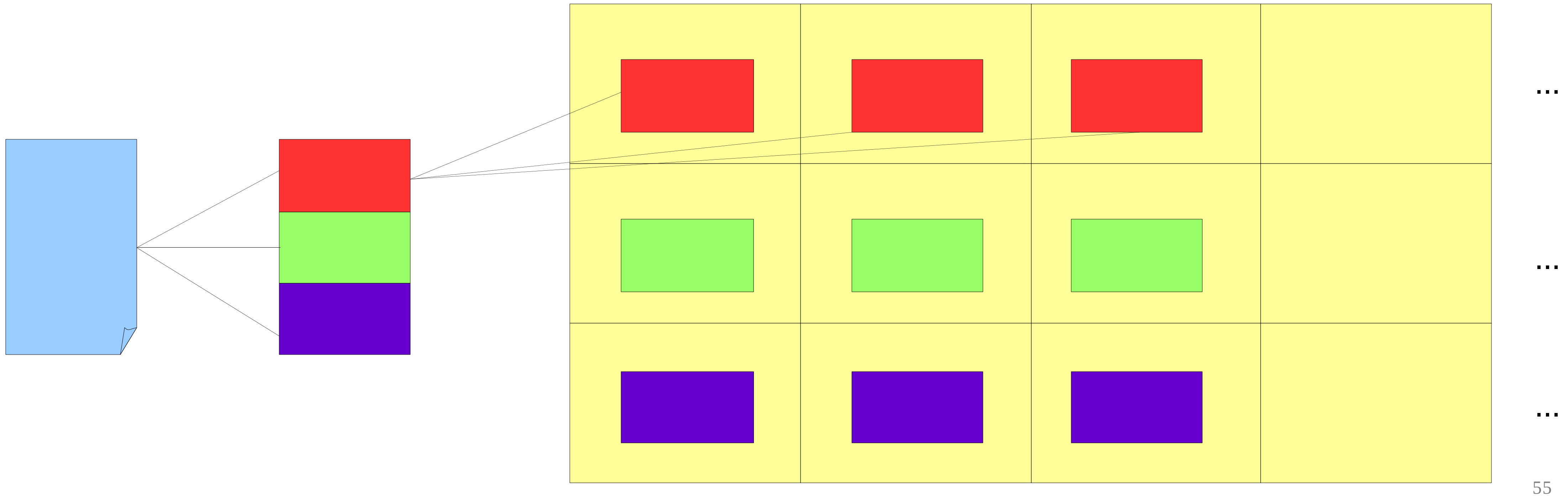
- We have weather readings from 10,000+ stations (temperature, wind direction, wind speed)
- For every year from 1901-2000
- Stored in big files: one per year
- Goal: Highest temperature measured for each year



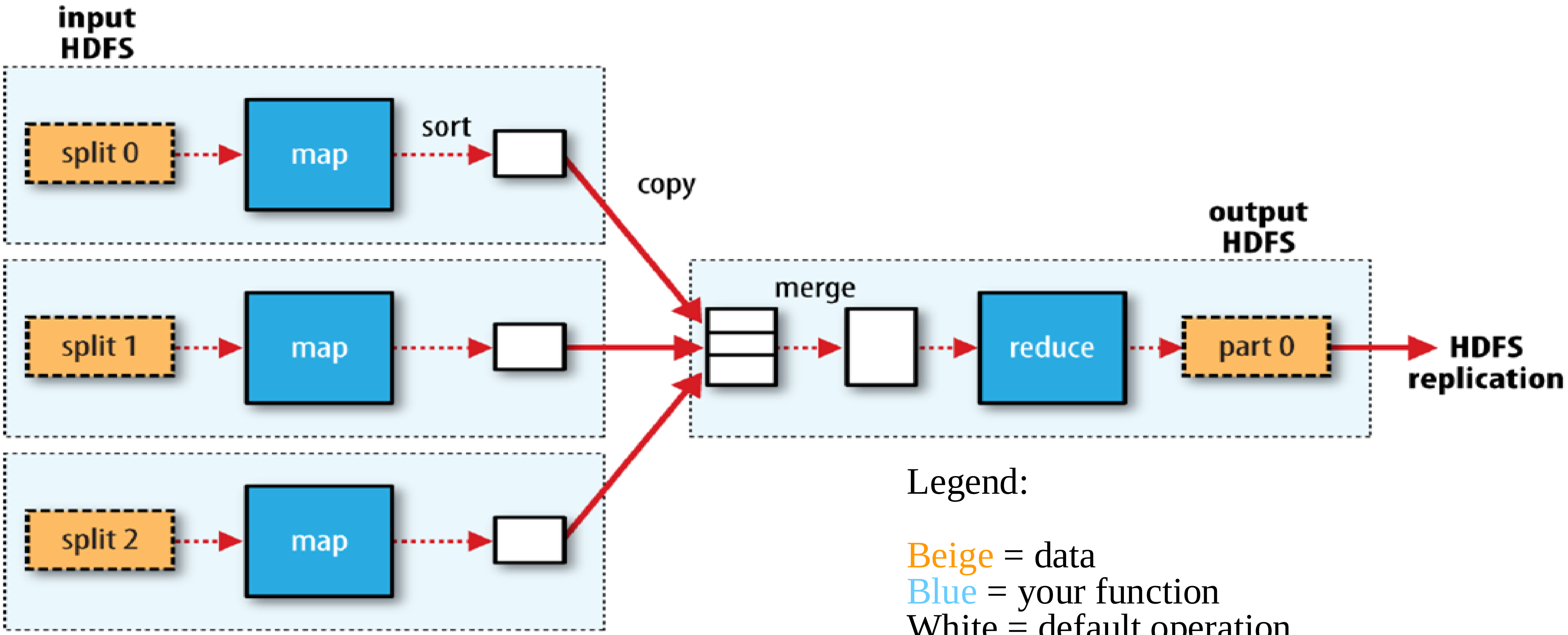
Source: *epa.gov*

Step 1: Upload Data

- Per year data file split into 128 MiB blocks
- Blocks stored on different machines
- Blocks replicated on 3+ machines



MapReduce Data Flow



Legend:

Beige = data
Blue = your function
White = default operation

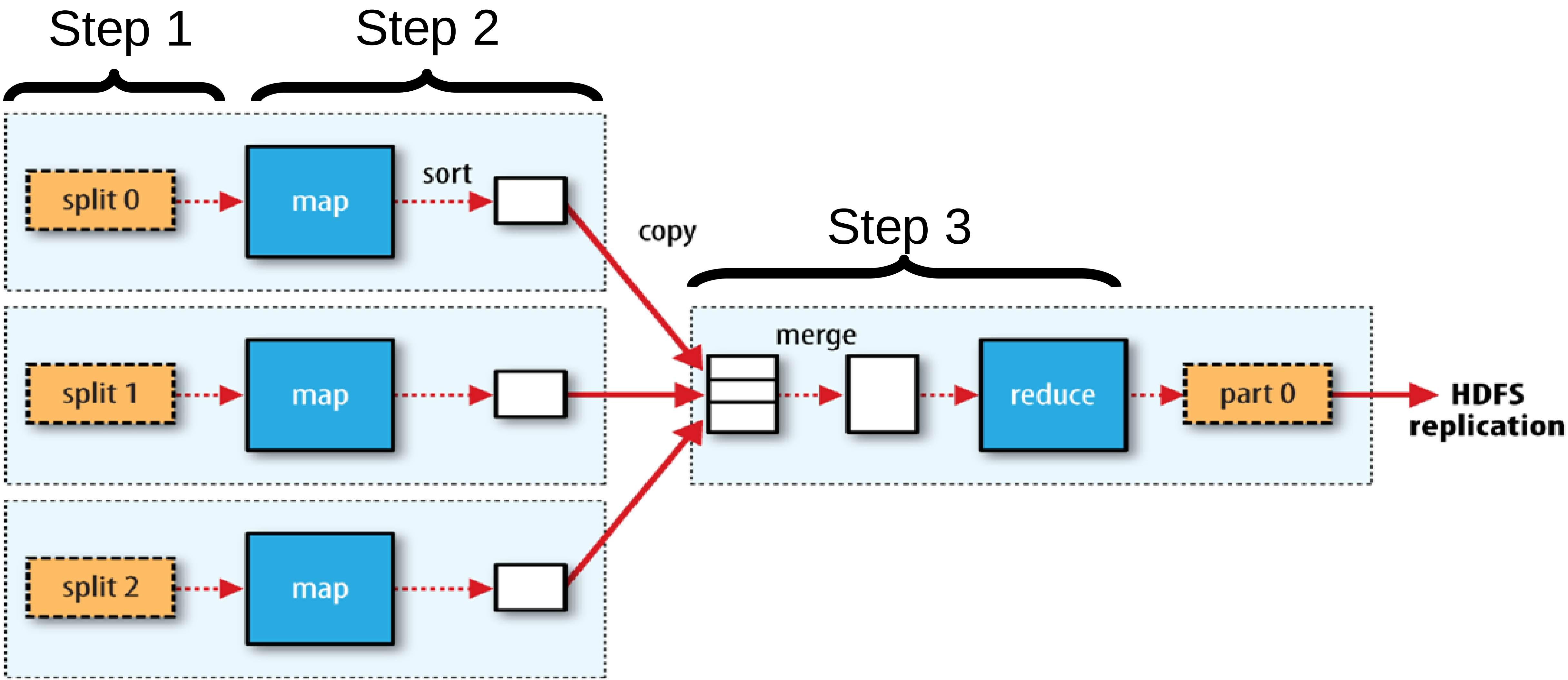
Step 2: Summarize Blocks

- Tasks are started on each machine that contains a weather block
- Task reads block and extracts (year, temperature) for each weather station
- Task summarizes and sorts data into
  - (year, [temp1,temp2,temp3])
- Map phase
- Data locality
  - Data is read by the machine that stores it
  - No network bandwidth wasted

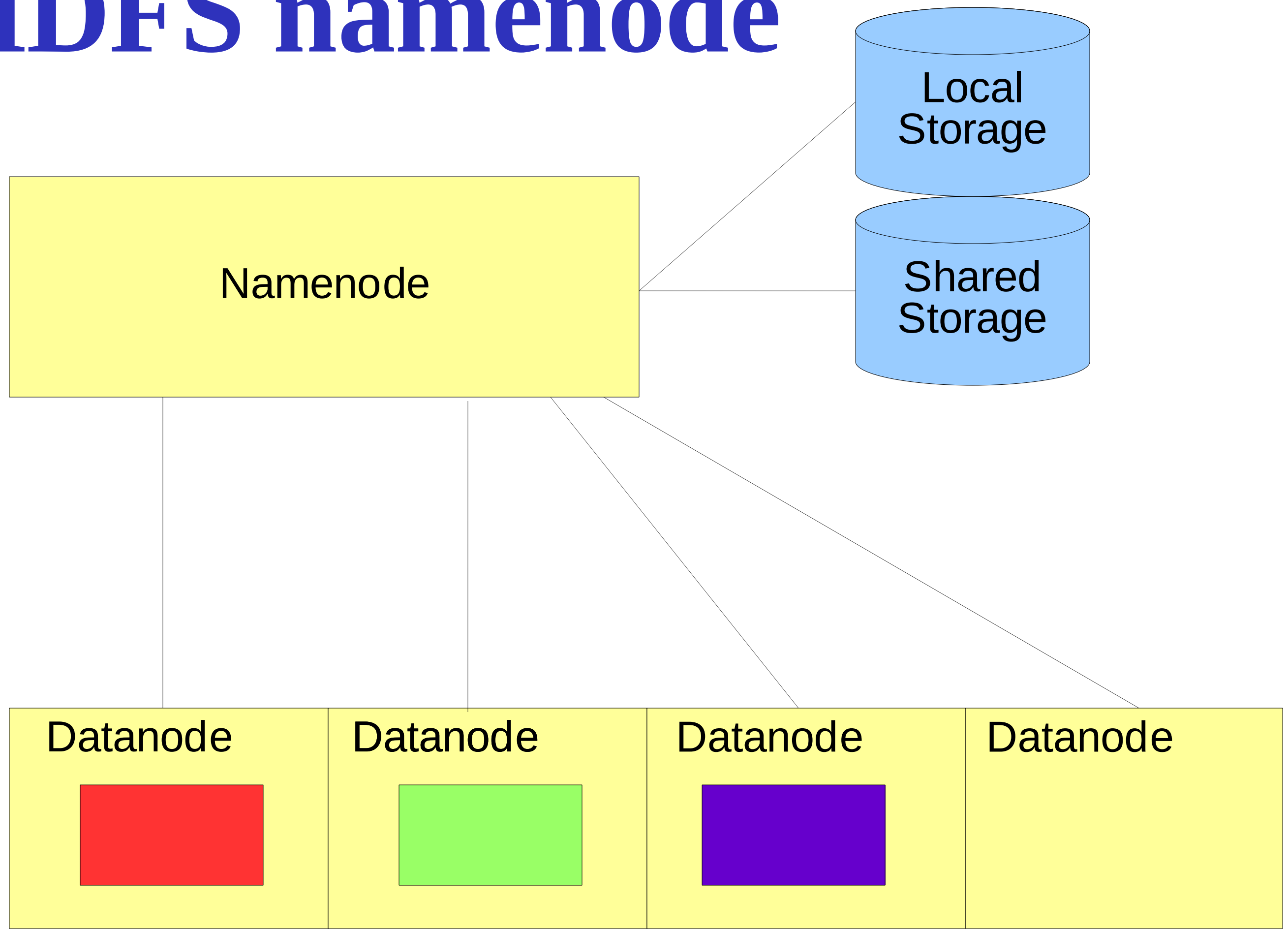
Step 3: Merge Per-Block Results

- Output from tasks is sent to a single machine
- Reducer task merges all (year, [temps]) lists
- Determines highest in list for each year
- Output written to Hadoop Filesystem
- Reduce phase

MapReduce Data Flow



HDFS namenode



Stores metadata:
/home/foo/data → Blocks
Saved to local and shared storage

Report block ownership to namenode

Hadoop Fault Tolerance

- Blocks stored on multiple machines
- If task fails on Copy1, create new task on Copy2
- Namenode:
- Primary w/fault-tolerant logging (slow)
- Active/Standby (fast)