More Primary/Backup

Doug Woos

Logistics notes

Problem set 1 posted…real soon now

- due next Friday, 9pm

Lab 1 due 9pm

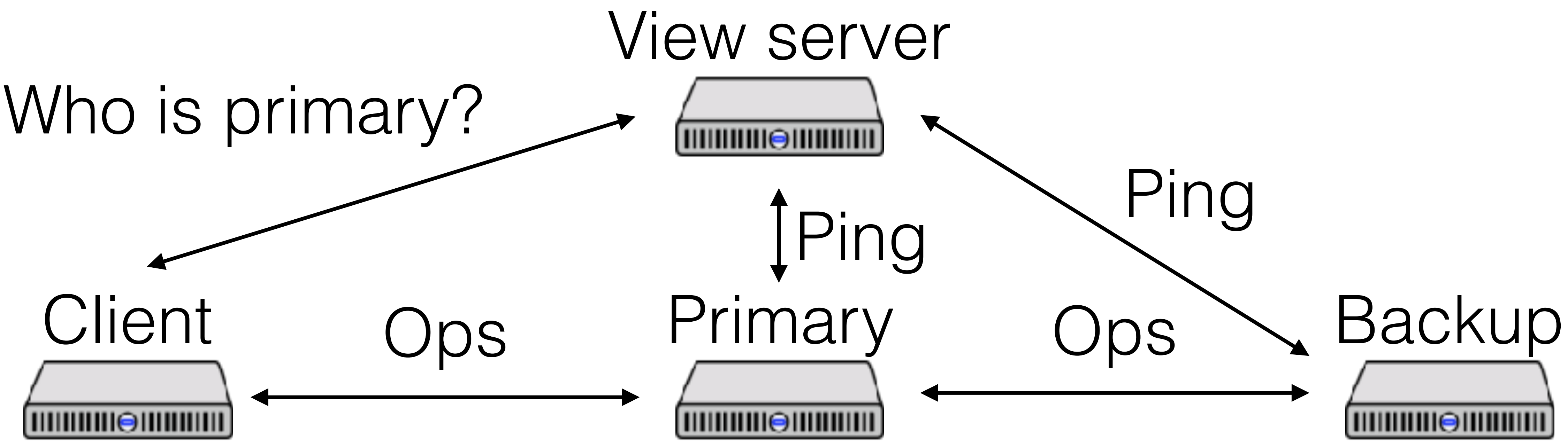
Question experiment

Today

More Primary/Backup

Intro to logical clocks

Primary/Backup Architecture



Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Split brain

1: A, B

A is still up, but can’t reach view server

2: C, D

C learns it is promoted to primary
A still thinks it is primary

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*

2. Primary must wait for backup to accept/execute each op before doing op and replying to client

3. Backup must accept forwarded requests only if view is correct

4. Non-primary must reject client requests

5. Every operation must be before or after state transfer

1. Missing writes

1: A, B

Client writes to A, receives response
A crashes before writing to B

2: B, C

Client reads from B
Write is missing

2. “Fast” Reads?

Does the primary need to forward reads to the backup?

(This is a common “optimization”)

Stale reads

1: A, B

A is still up, but can’t reach view server

2: B, C

Client 1 writes to B
Client 2 reads from A
A returns outdated value

Reads vs. writes

Reads treated as state machine operations too

But: can be executed more than once

RPC library can handle them differently

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*

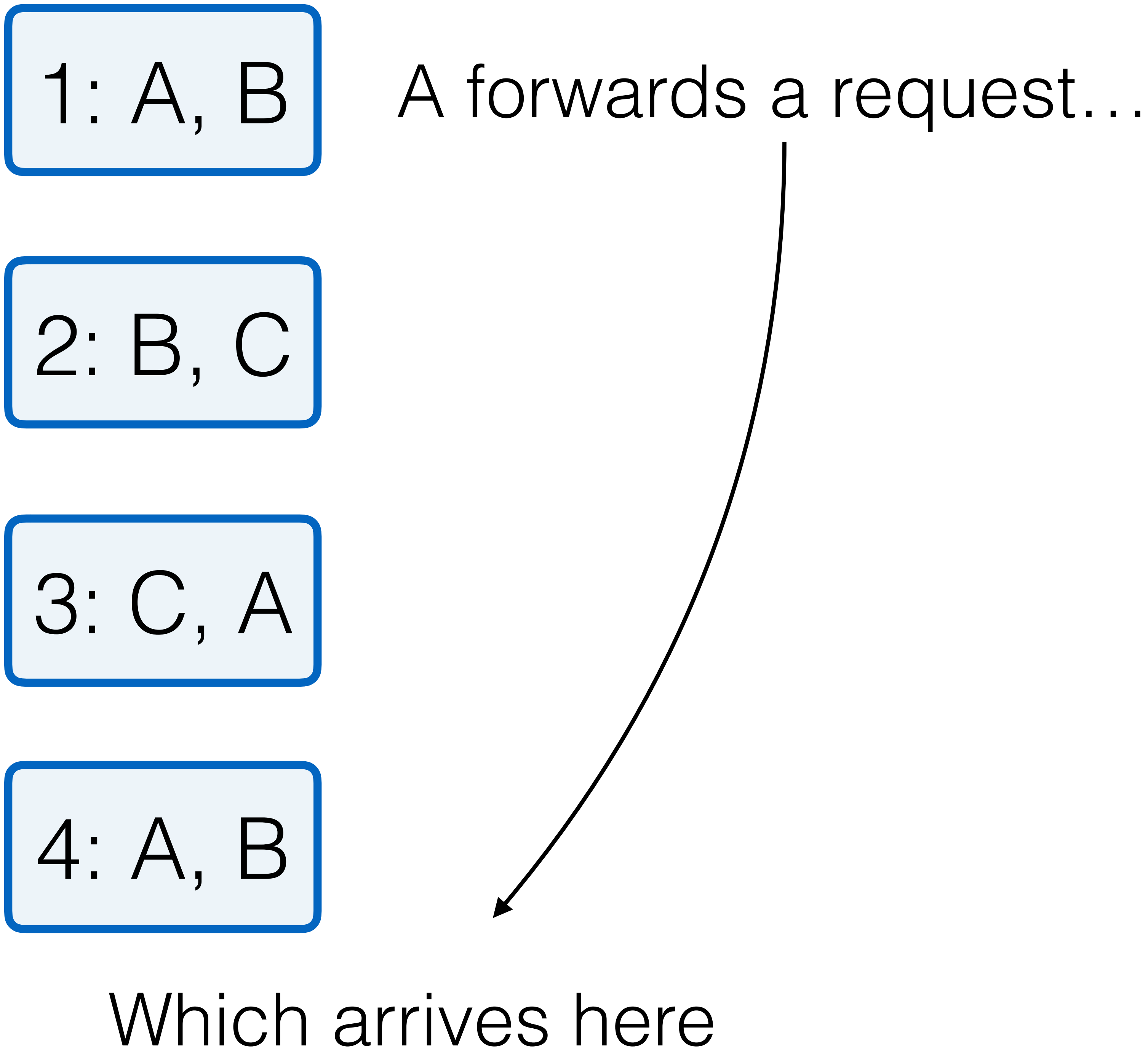
2. Primary must wait for backup to accept/execute each op before doing op and replying to client

3. Backup must accept forwarded requests only if view is correct

4. Non-primary must reject client requests

5. Every operation must be before or after state transfer

Old messages



1: A, B

2: B, C

3: C, A

4: A, B

A forwards a request…

Which arrives here

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*

2. Primary must wait for backup to accept/execute each op before doing op and replying to client

3. Backup must accept forwarded requests only if view is correct

4. Non-primary must reject client requests

5. Every operation must be before or after state transfer

Inconsistencies

1: A, B

2: B, C

2: B, A

Outdated client sends request to A
A shouldn’t respond!

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*
2. Primary must wait for backup to accept/execute each op before doing op and replying to client
3. Backup must accept forwarded requests only if view is correct
4. Non-primary must reject client requests
5. Every operation must be before or after state transfer

Inconsistencies

1: A, B

A starts sending state to B
Client writes to A
A forwards op to B
A sends rest of state to B

Rules

1. Primary in view *i+1* must have been backup or primary in view *i*

2. Primary must wait for backup to accept/execute each op before doing op and replying to client

3. Backup must accept forwarded requests only if view is correct

4. Non-primary must reject client requests

5. Every operation must be before or after state transfer

Progress

Are there cases when the system can’t make further progress (i.e. process new client requests)?

Progress

- View server fails
- Network fails entirely (hard to get around this one)
- Clients can’t reach primary but it can ping VS
- No backup and primary fails
- Primary fails before ack’ing view change

State transfer and RPCs

State transfer must include RPC data

Duplicate writes

1: A, B

Client writes to A
A forwards to B
A replies to client
Reply is dropped

2: B, C

B transfers state to C, crashes

3: C, D

Client resends write. Duplicated!

One more corner case

1: A, B

View server stops hearing from A
A and B, and clients, can still communicate

2: B, C

B hasn’t heard from view server
Client in view 1 sends a request to A
What happens?
Client in view 2 sends a request to B
What happens?

Logical time

Distinct from physical time

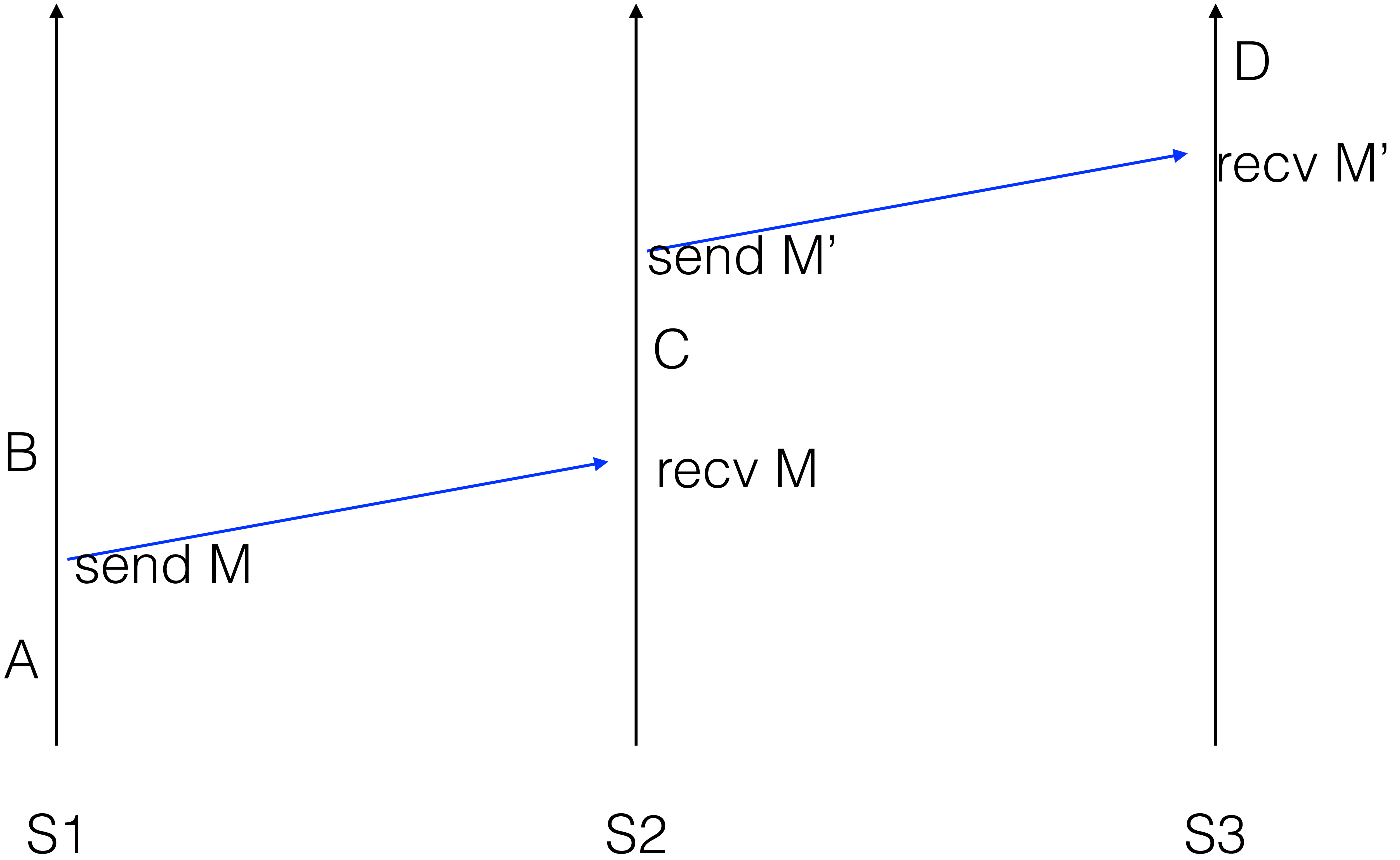
How can we order events at different nodes?

What does it mean for an event to happen before another one?

Happens-before

1. Happens at same location, earlier
2. Transmission before receipt

Space-time diagrams



Lamport clocks

Idea: timestamp on each event

When to advance timestamp, and to what?

How to implement a lock using logical clocks?

Tune in next time