Lamport Clocks

Lamport Clocks

Framework for *reasoning* about event ordering

Assign timestamps to events

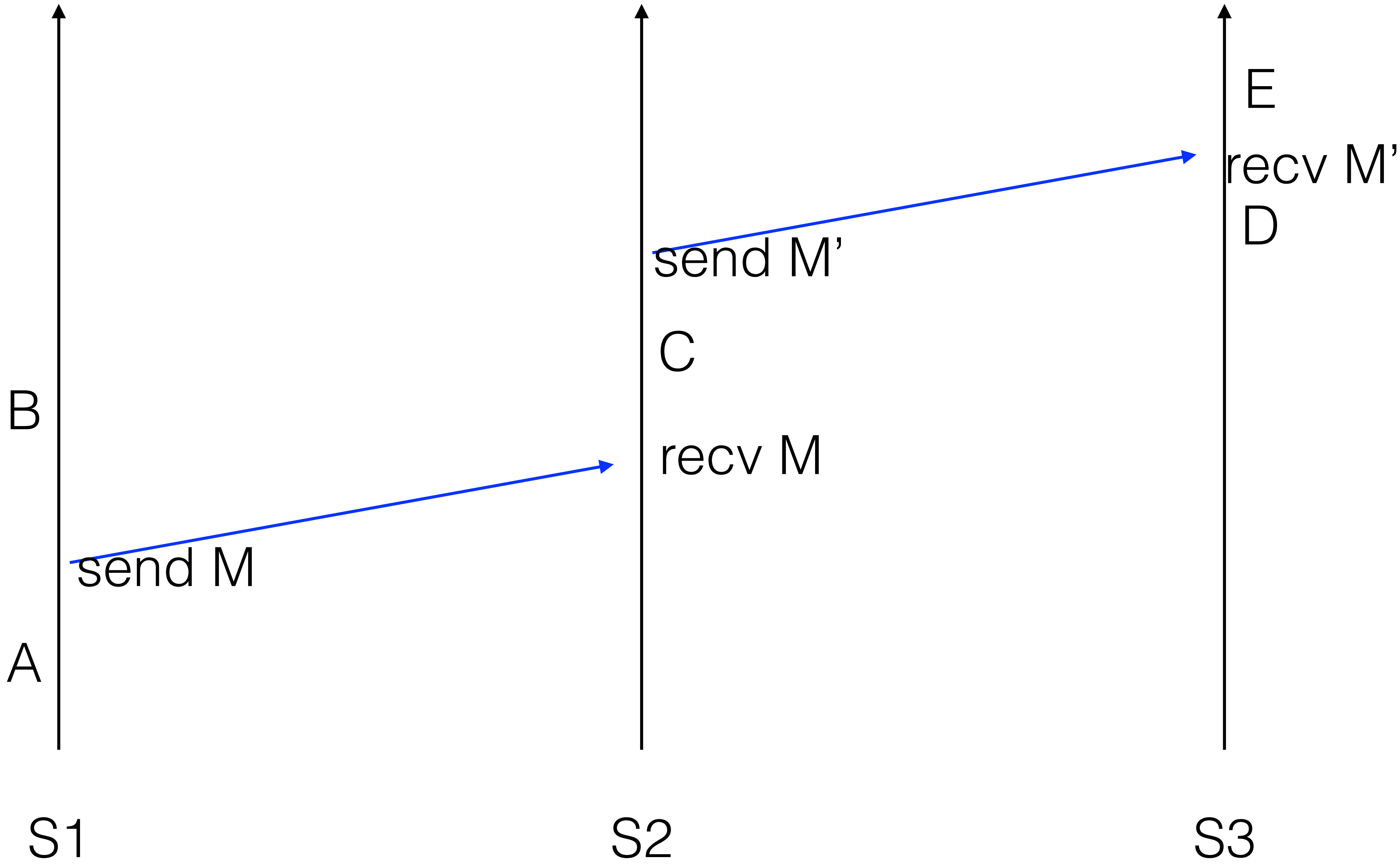
- Globally valid
- Respects causality
- Using only local information
- No physical clock

What does it mean for *a* to happen before *b*?

Happens-before

1. Happens earlier at same location
2. Transmission before receipt
3. Transitivity

Example



Logical clock implementation

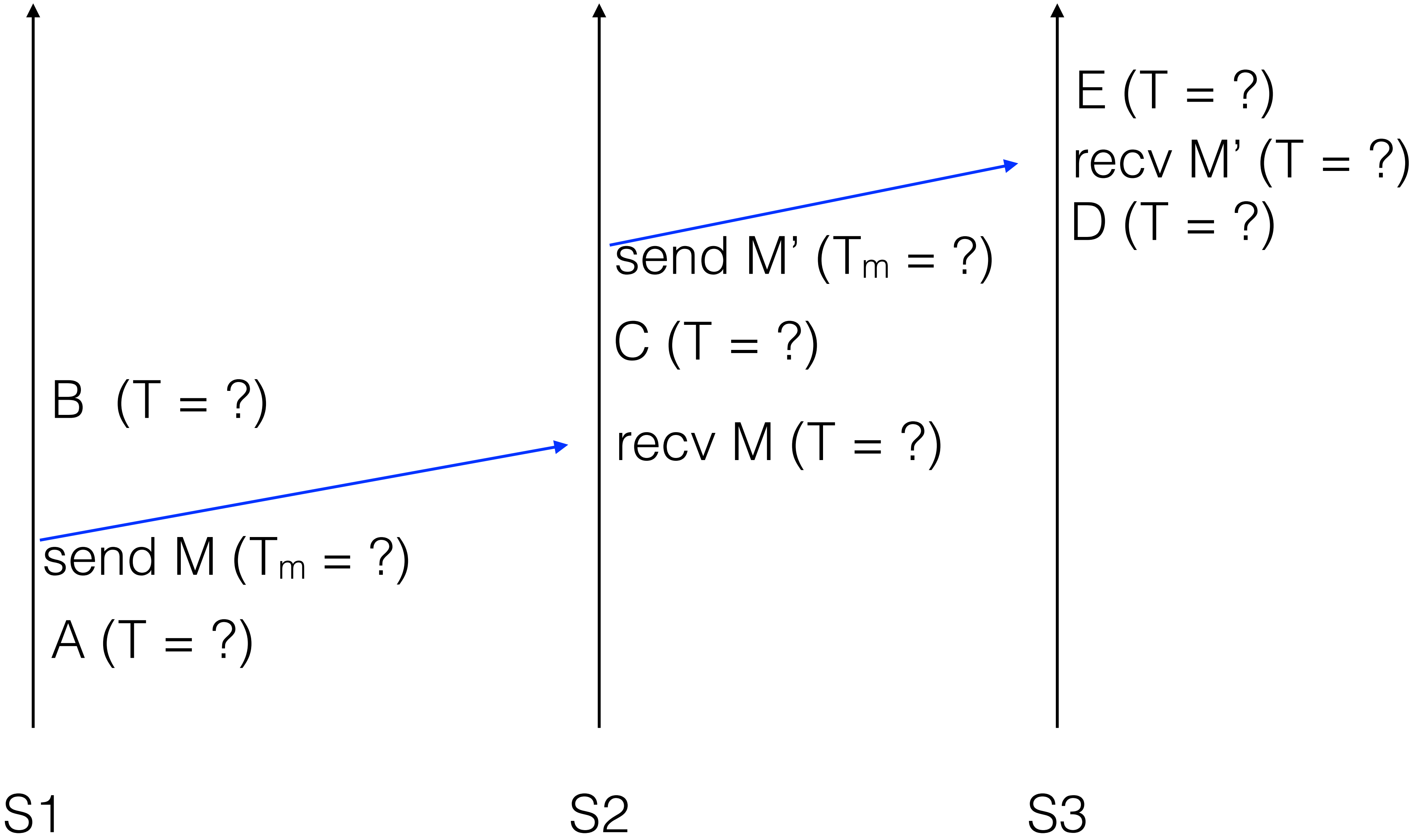
Keep a local clock T

Increment T whenever an event happens

Send clock value on all messages as T_m

On message receipt: $T = max(T, T_m) + 1$

Example



B (T = ?)

send M (T_m = ?)

A (T = ?)

send M' (T_m = ?)

C (T = ?)

recv M (T = ?)

E (T = ?)

recv M' (T = ?)

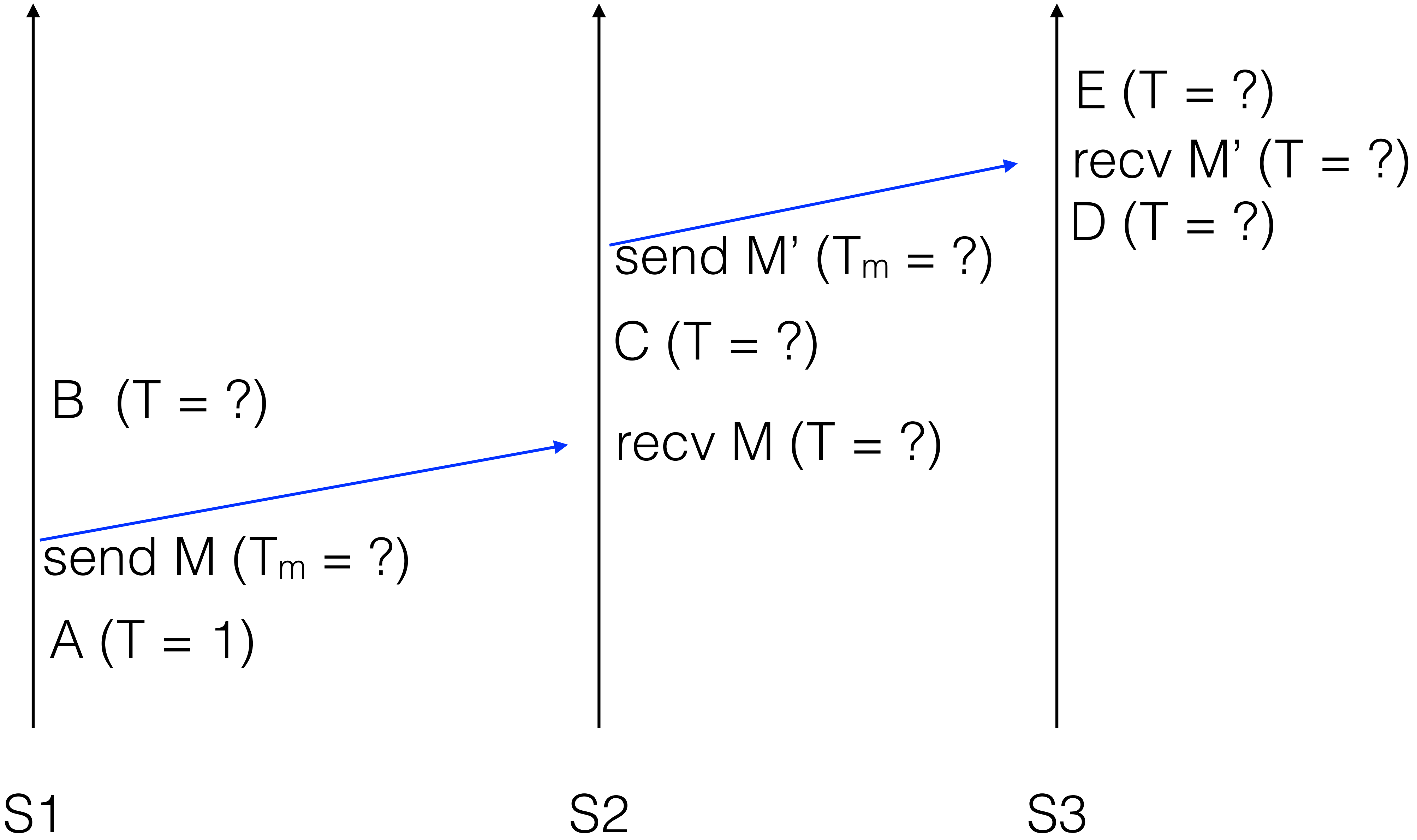
D (T = ?)

S1

S2

S3

Example



E (T = ?)

recv M’ (T = ?)

D (T = ?)

send M’ (T_m = ?)

C (T = ?)

B (T = ?)

recv M (T = ?)

send M (T_m = ?)

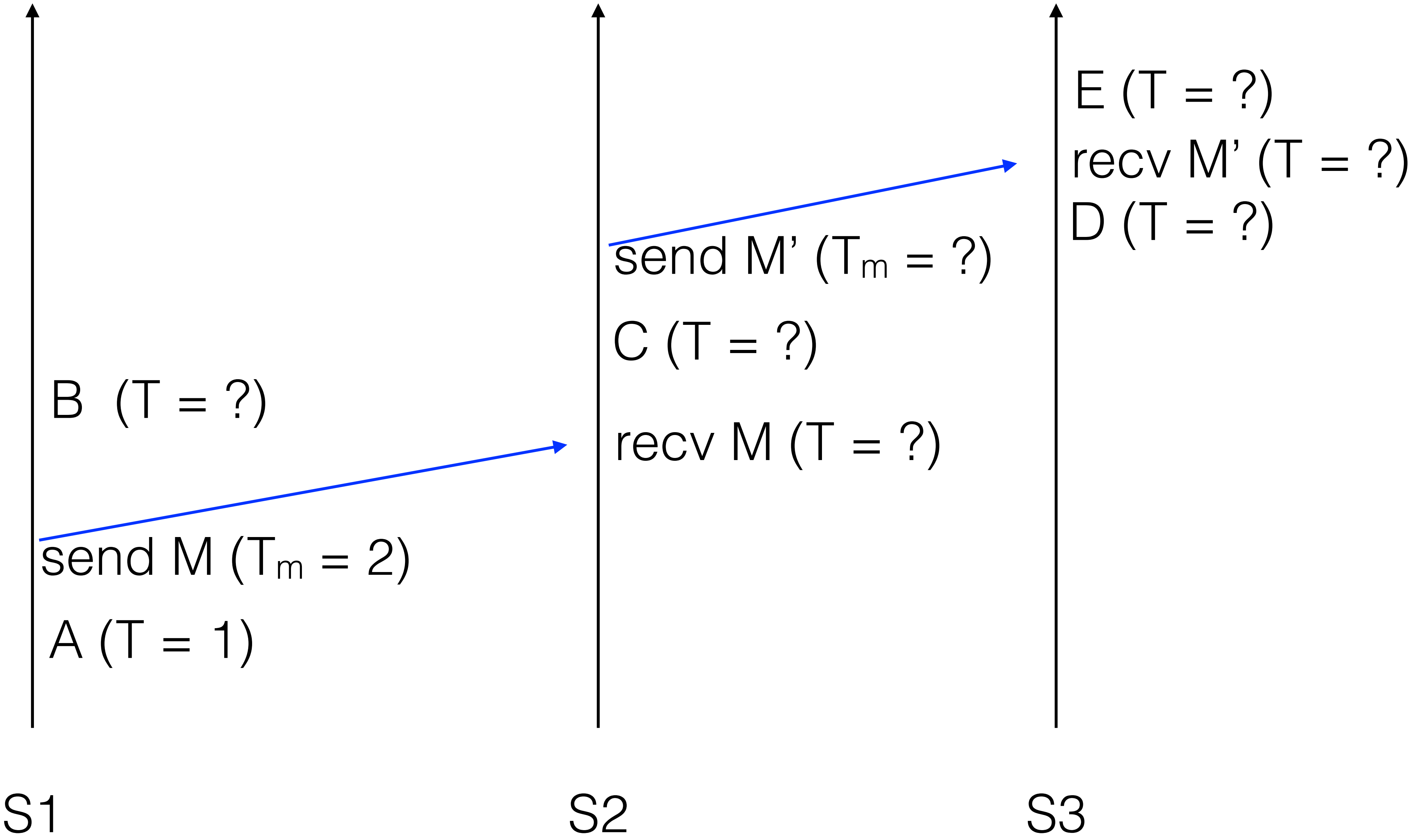
A (T = 1)

S1

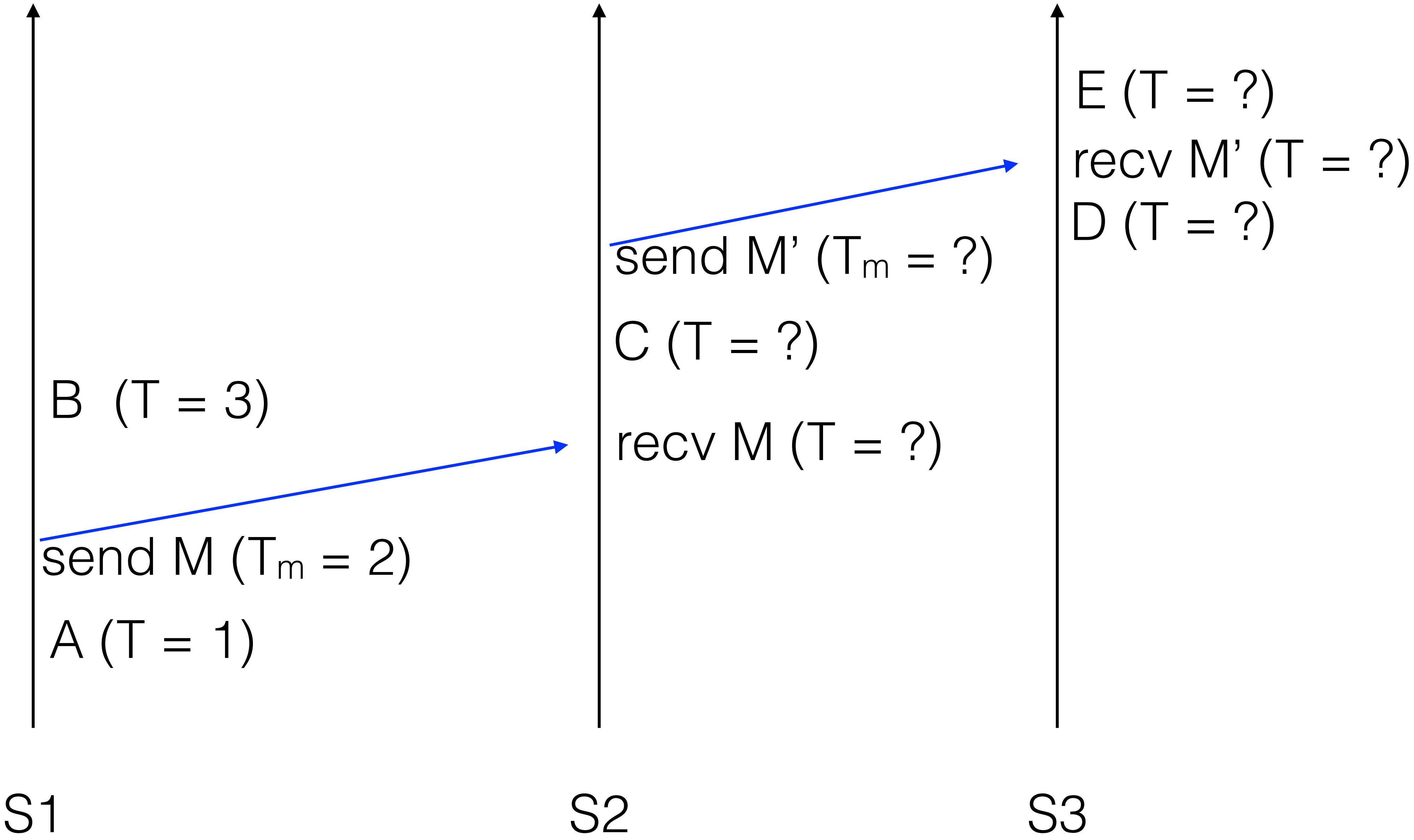
S2

S3

Example



Example



B (T = 3)

send M ($T_m = 2$)

A (T = 1)

send M’ ($T_m = ?$)

C (T = ?)

recv M (T = ?)

E (T = ?)

recv M’ (T = ?)

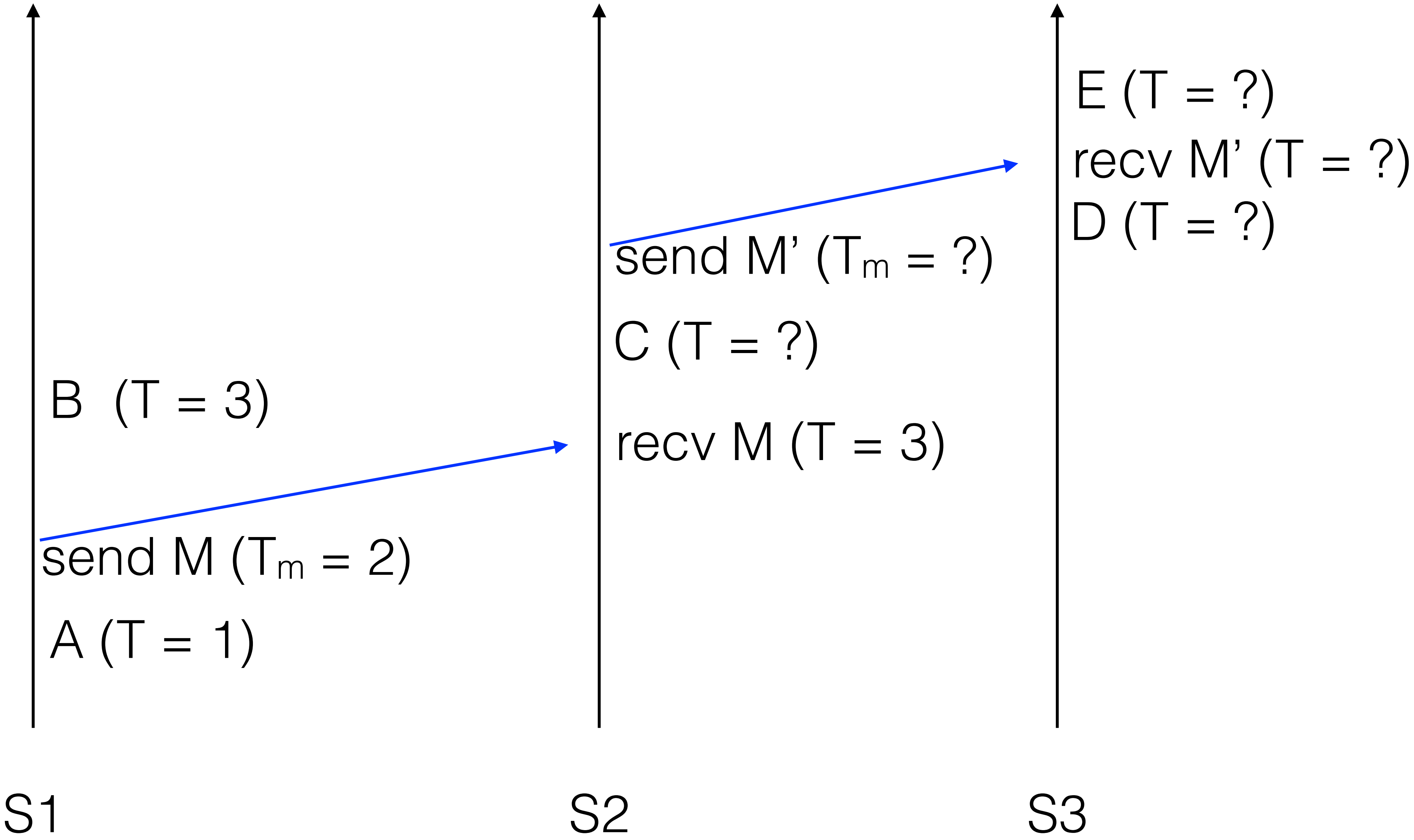
D (T = ?)

S1

S2

S3

Example



B (T = 3)

send M (T_m = 2)

A (T = 1)

send M’ (T_m = ?)

C (T = ?)

recv M (T = 3)

E (T = ?)

recv M’ (T = ?)

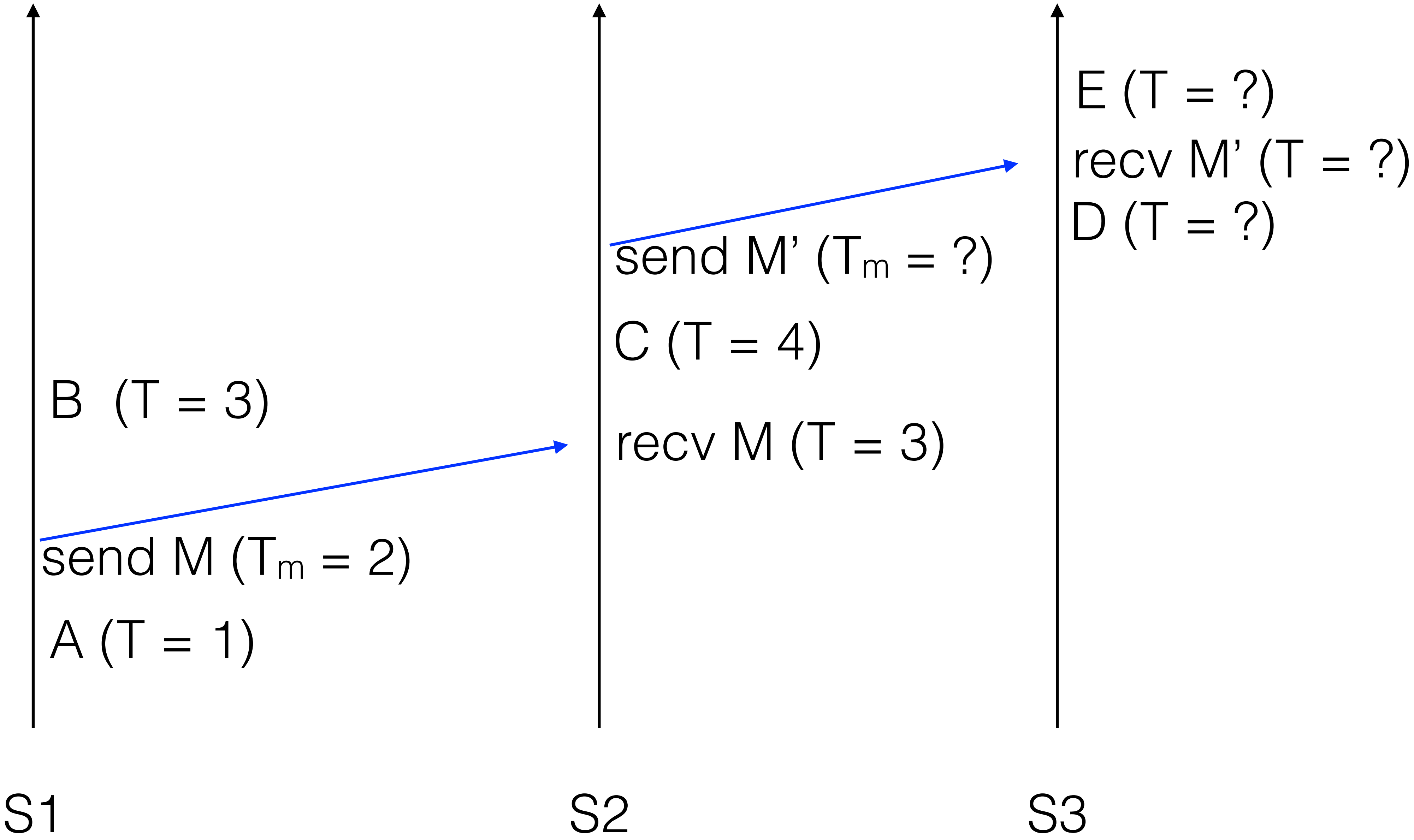
D (T = ?)

S1

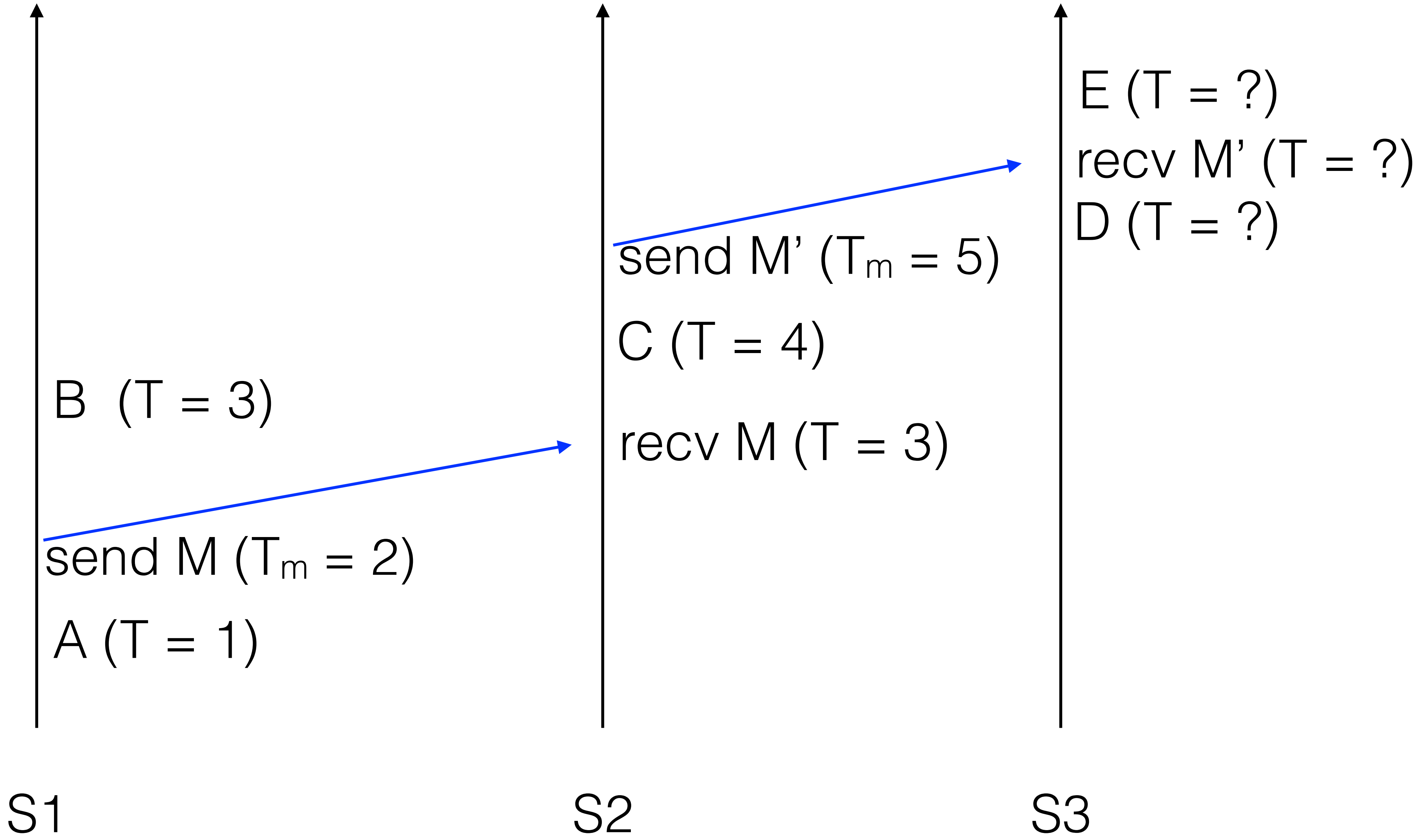
S2

S3

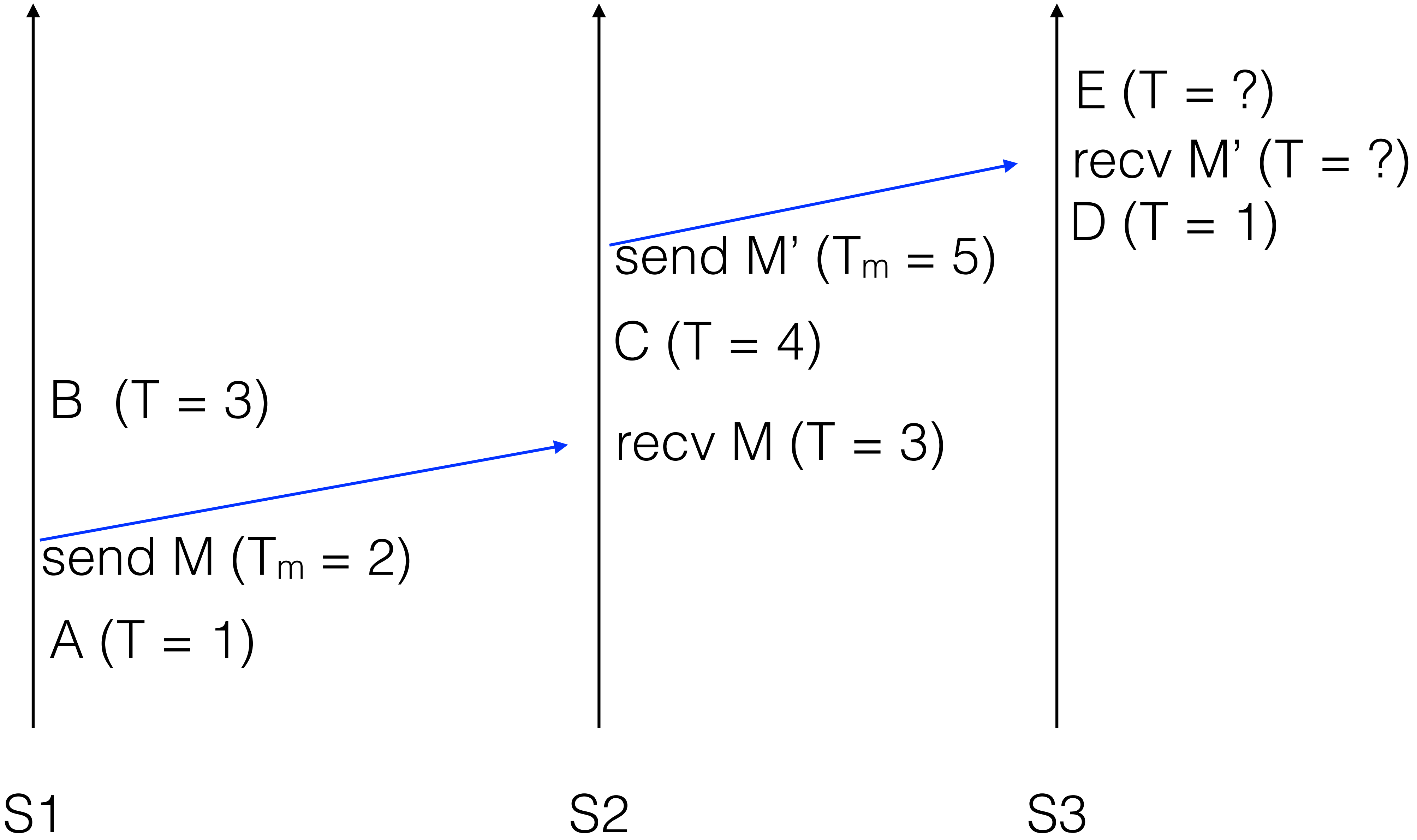
Example



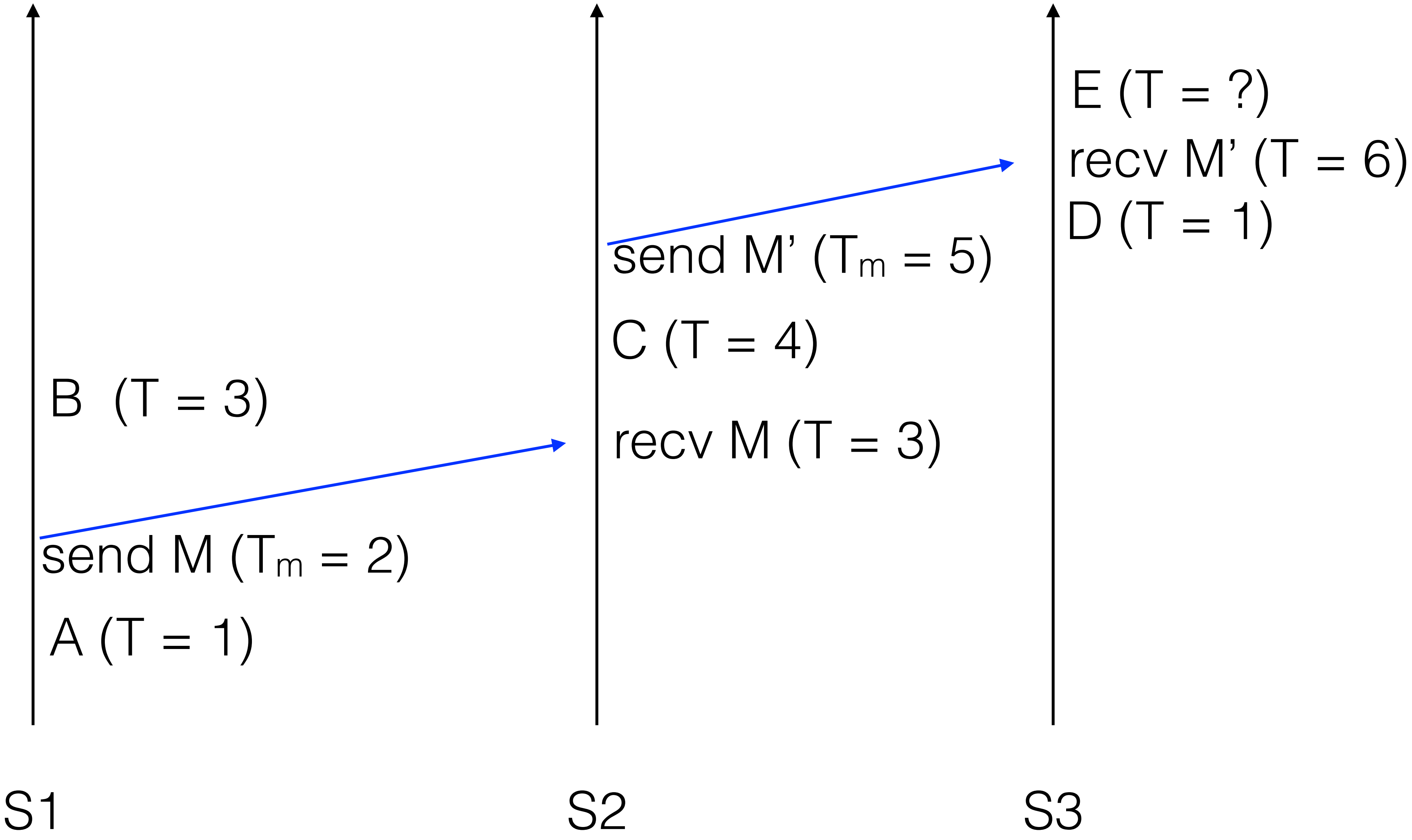
Example



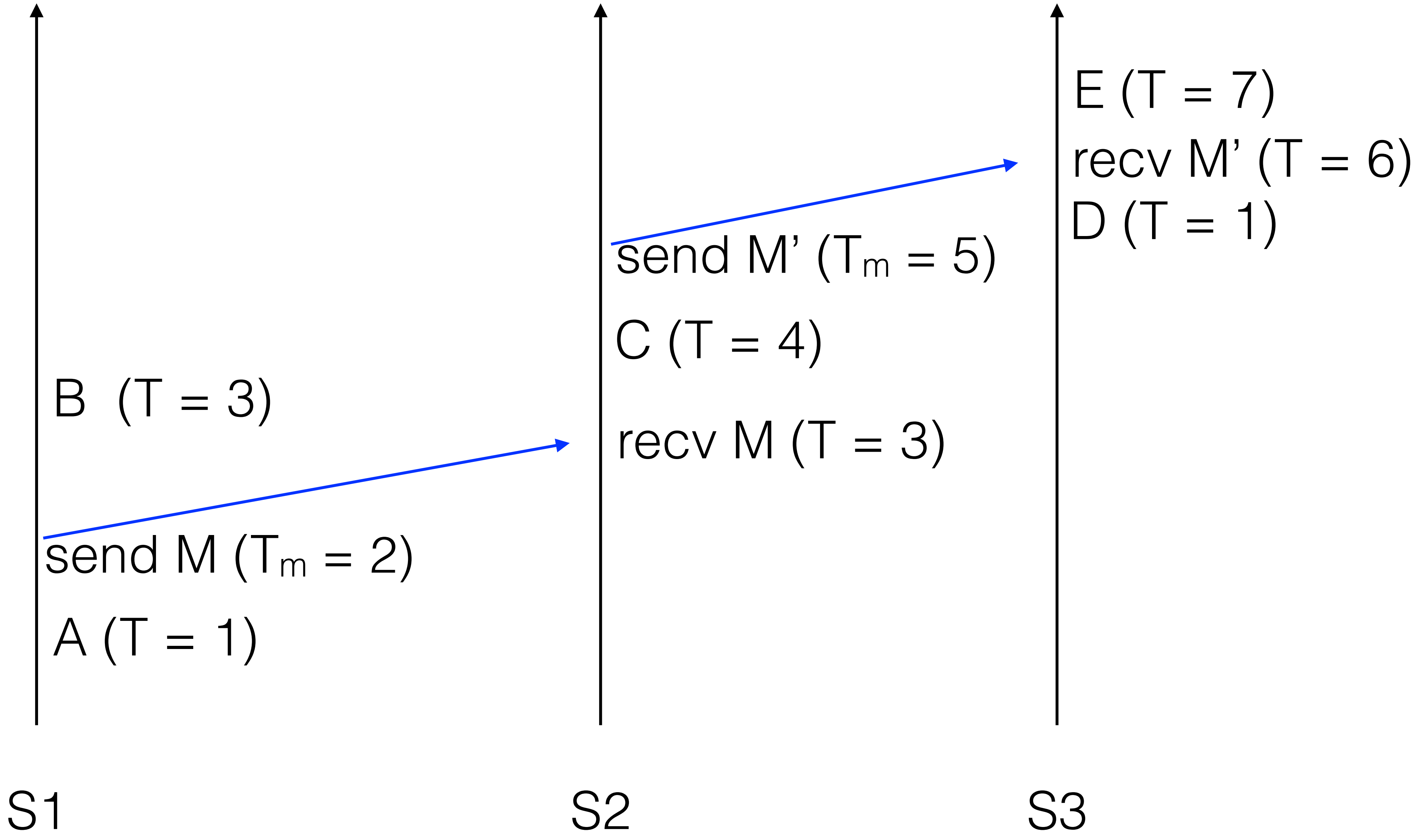
Example



Example


E (T = ?)
recv M’ (T = 6)
D (T = 1)
send M’ (T$_m$ = 5)
C (T = 4)
B (T = 3)
recv M (T = 3)
send M (T$_m$ = 2)
A (T = 1)
S1
S2
S3

Example


E (T = 7)
recv M’ (T = 6)
D (T = 1)
send M’ (T_m = 5)
C (T = 4)
B (T = 3)
recv M (T = 3)
send M (T_m = 2)
A (T = 1)
S1
S2
S3

Goal of Lamport clocks

happens-before(A, B) -> *T*(A) < *T*(B)

Does T(A) < T(B) -> happens-before(A, B)?

Mutual exclusion

Use clocks to implement a lock

- Using state machine replication

Goals:

- Only one process has the lock at a time
- Requesting processes eventually acquire the lock

Assumptions:

- In-order point-to-point message delivery
- No failures, all messages delivered

Mutual exclusion implementation

Each message carries a timestamp T_m (and a seq #)

Three message types:

- *request* (broadcast)
- *release* (broadcast)
- *acknowledge* (on receipt)

Each node's state:

- A queue of *request* messages, ordered by T_m
- The latest message it has received from each node

Mutual exclusion implementation

On receiving a *request*:

- Record message timestamp
- Add request to queue

On receiving a *release*:

- Record message timestamp
- Remove corresponding request from queue

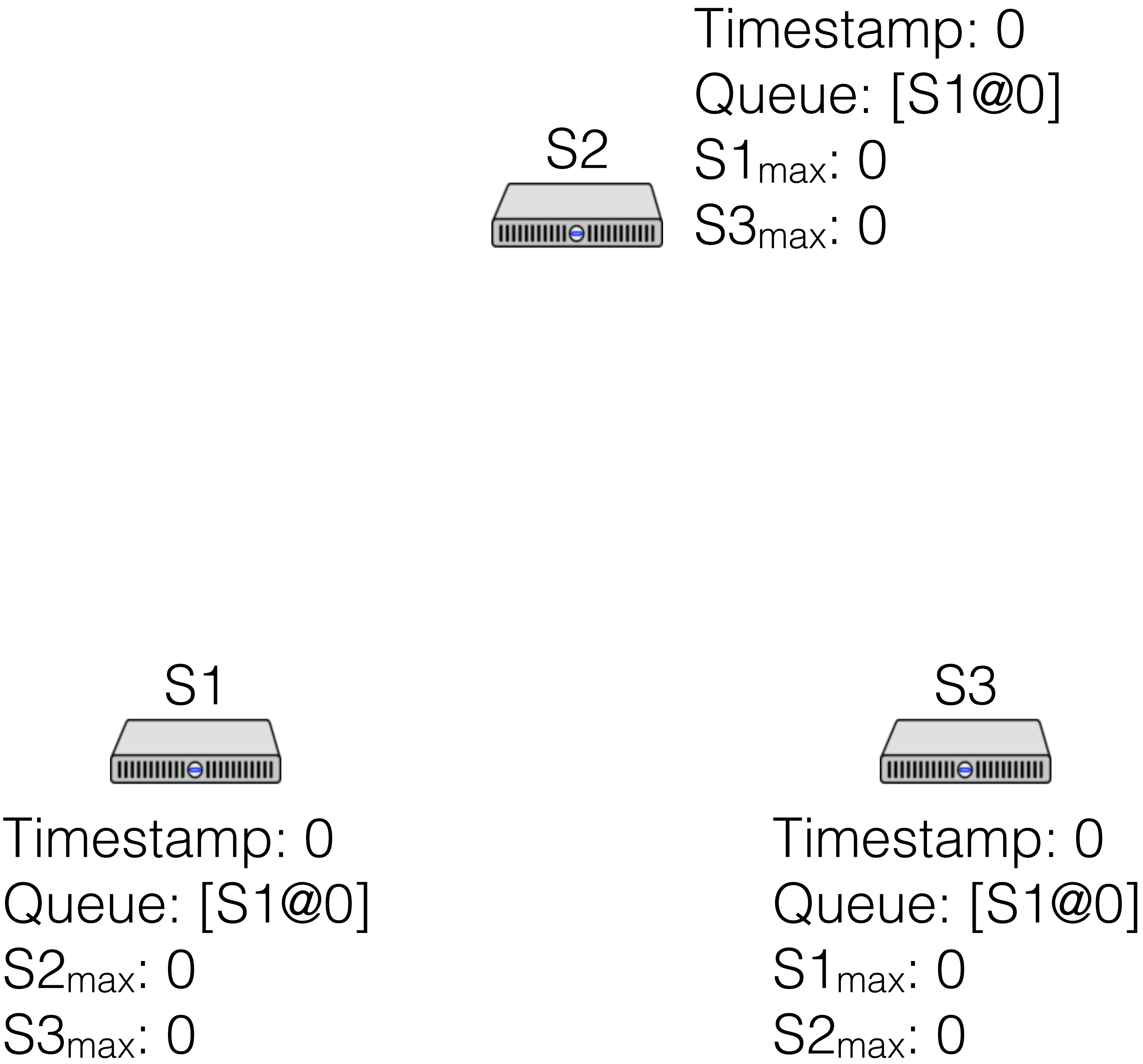
On receiving an *acknowledge*:

- Record message timestamp

Mutual exclusion implementation

To acquire the lock:

- Send *request* to everyone, including self
- The lock is acquired when:
  - My request is at the head of my queue, and
  - I’ve received higher-timestamped messages from everyone
  - So my request must be the earliest



S2

Timestamp: 0

Queue: [S1@0]

$S1_{max}$: 0

$S3_{max}$: 0

S1

Timestamp: 0

Queue: [S1@0]

$S2_{max}$: 0

$S3_{max}$: 0

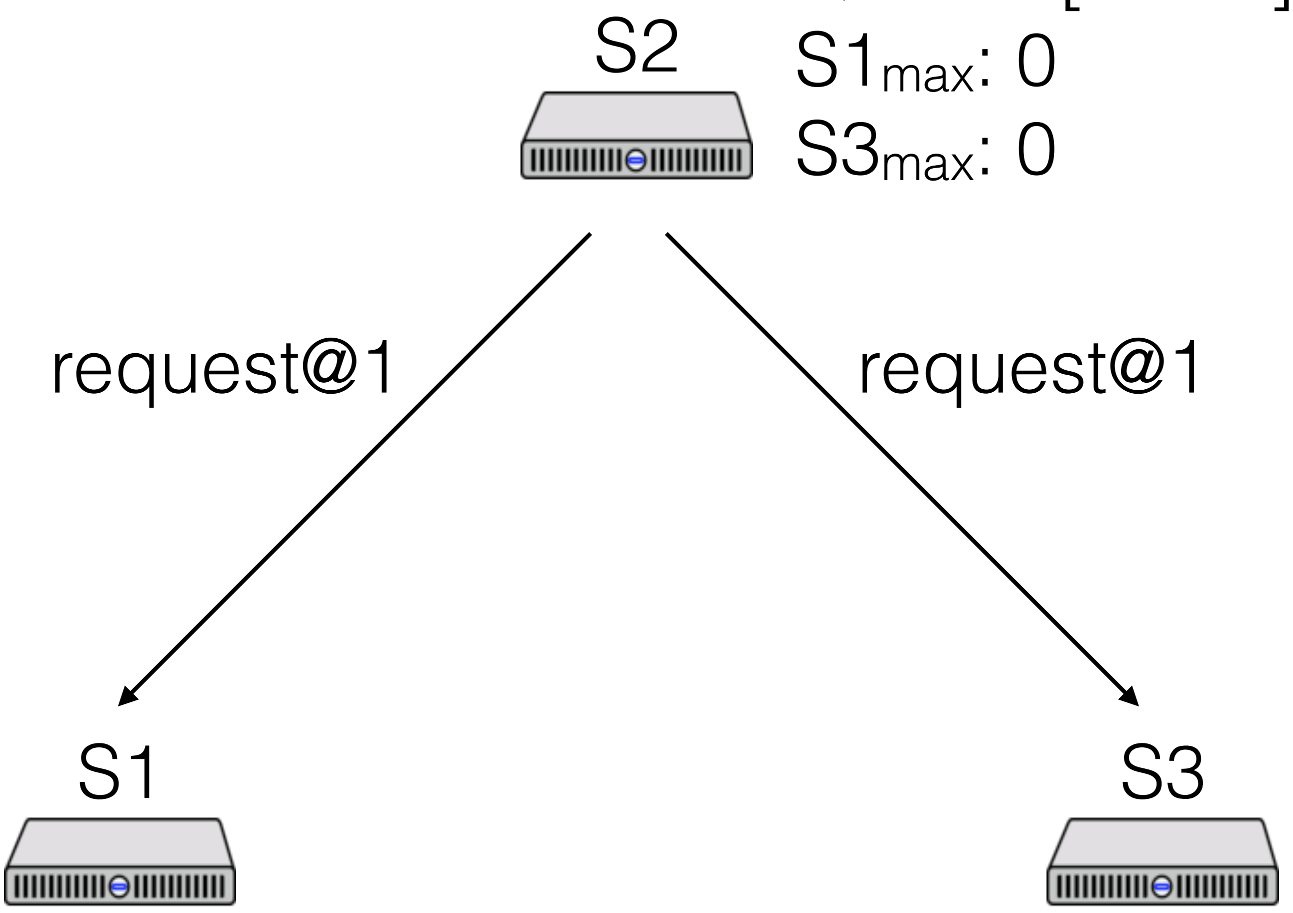
S3

Timestamp: 0

Queue: [S1@0]

$S1_{max}$: 0

$S2_{max}$: 0



S2:

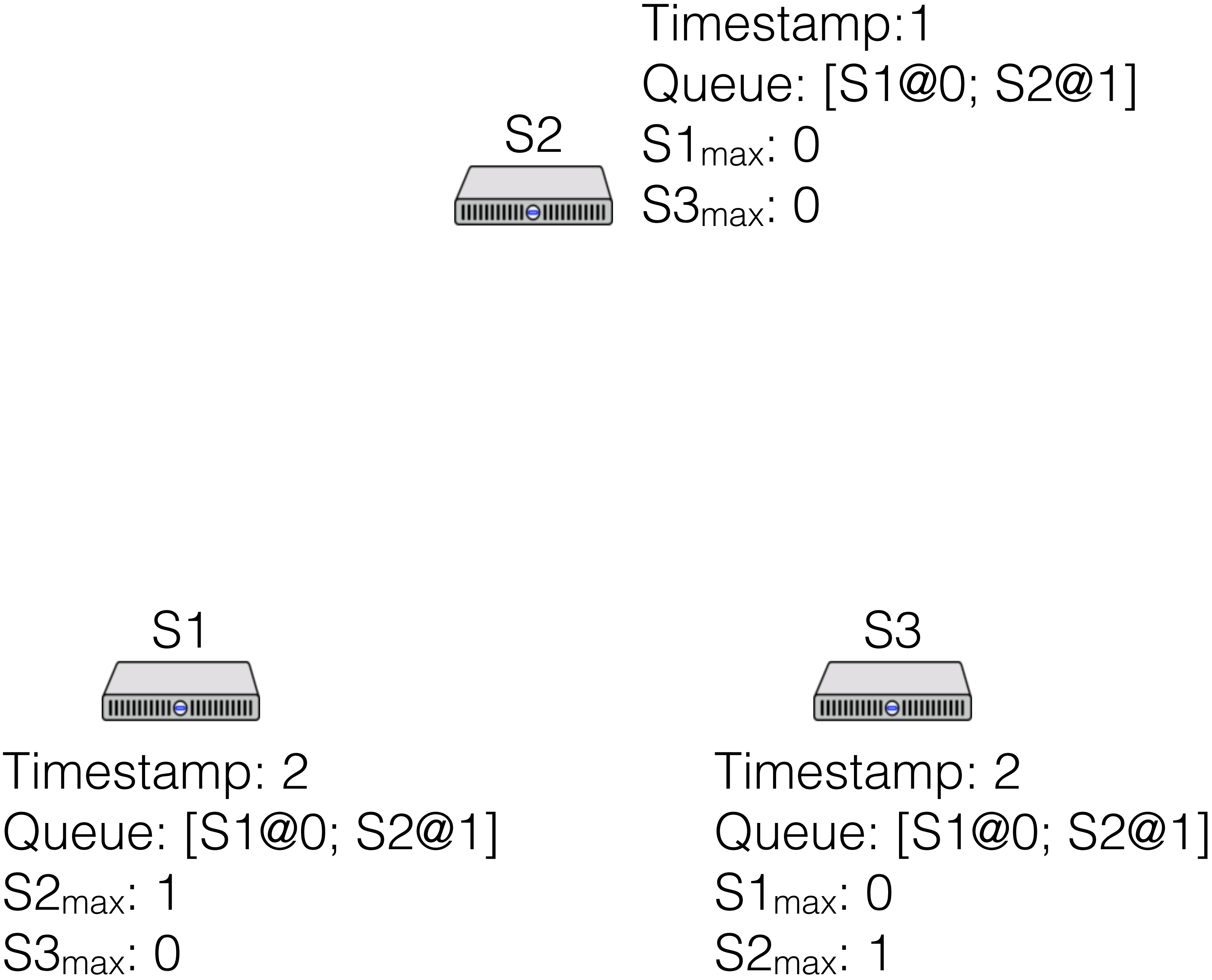
Timestamp: 1
Queue: [S1@0]
$S1_{max}$: 0
$S3_{max}$: 0

S1:

Timestamp: 0
Queue: [S1@0]
$S2_{max}$: 0
$S3_{max}$: 0

S3:

Timestamp: 0
Queue: [S1@0]
$S1_{max}$: 0
$S2_{max}$: 0



Timestamp:1

Queue: [S1@0; S2@1]

$S1_{max}$: 0

$S3_{max}$: 0

Timestamp: 2

Queue: [S1@0; S2@1]

$S2_{max}$: 1

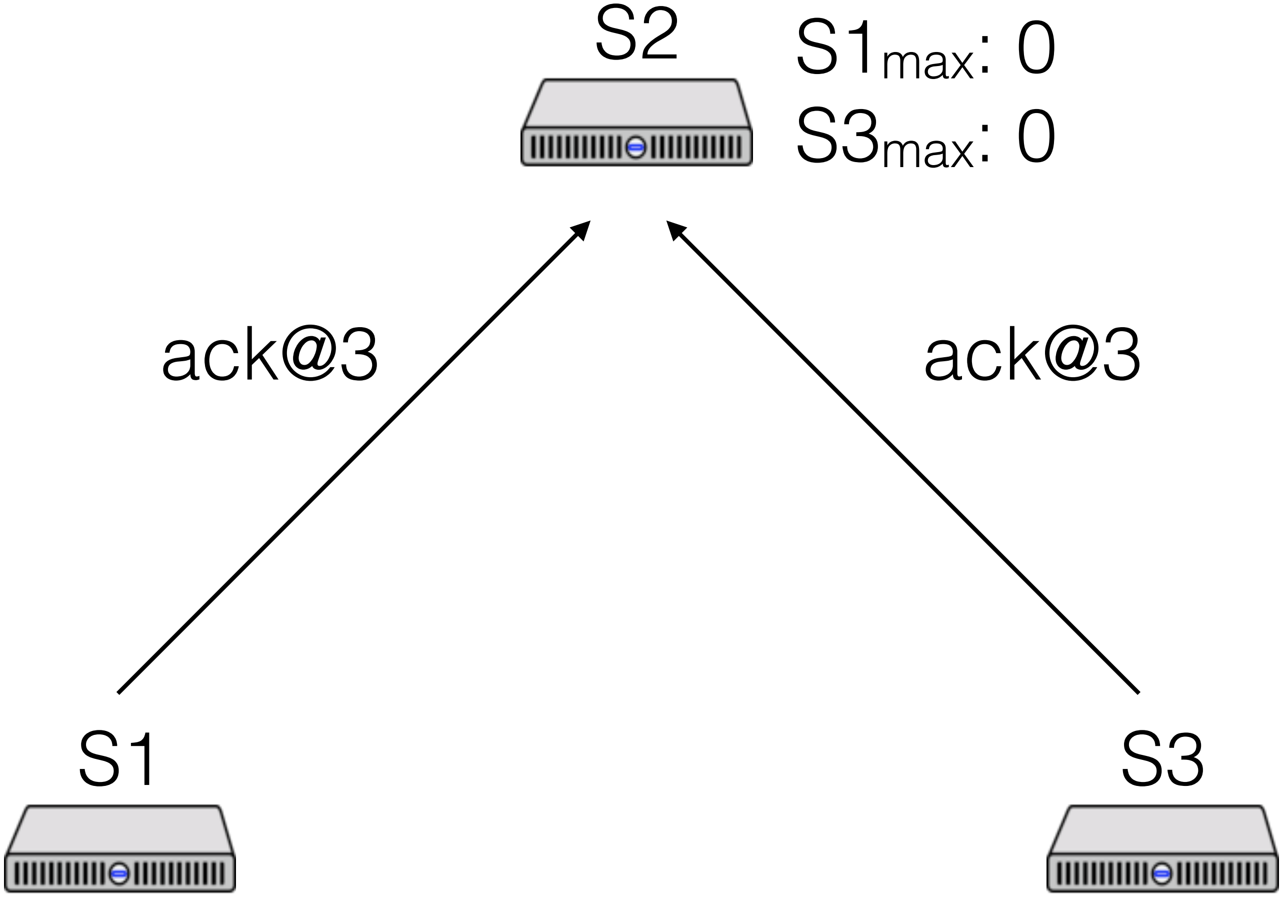
$S3_{max}$: 0

Timestamp: 2

Queue: [S1@0; S2@1]

$S1_{max}$: 0

$S2_{max}$: 1



S2

Timestamp:1
Queue: [S1@0; S2@1]
$S1_{max}$: 0
$S3_{max}$: 0

ack@3

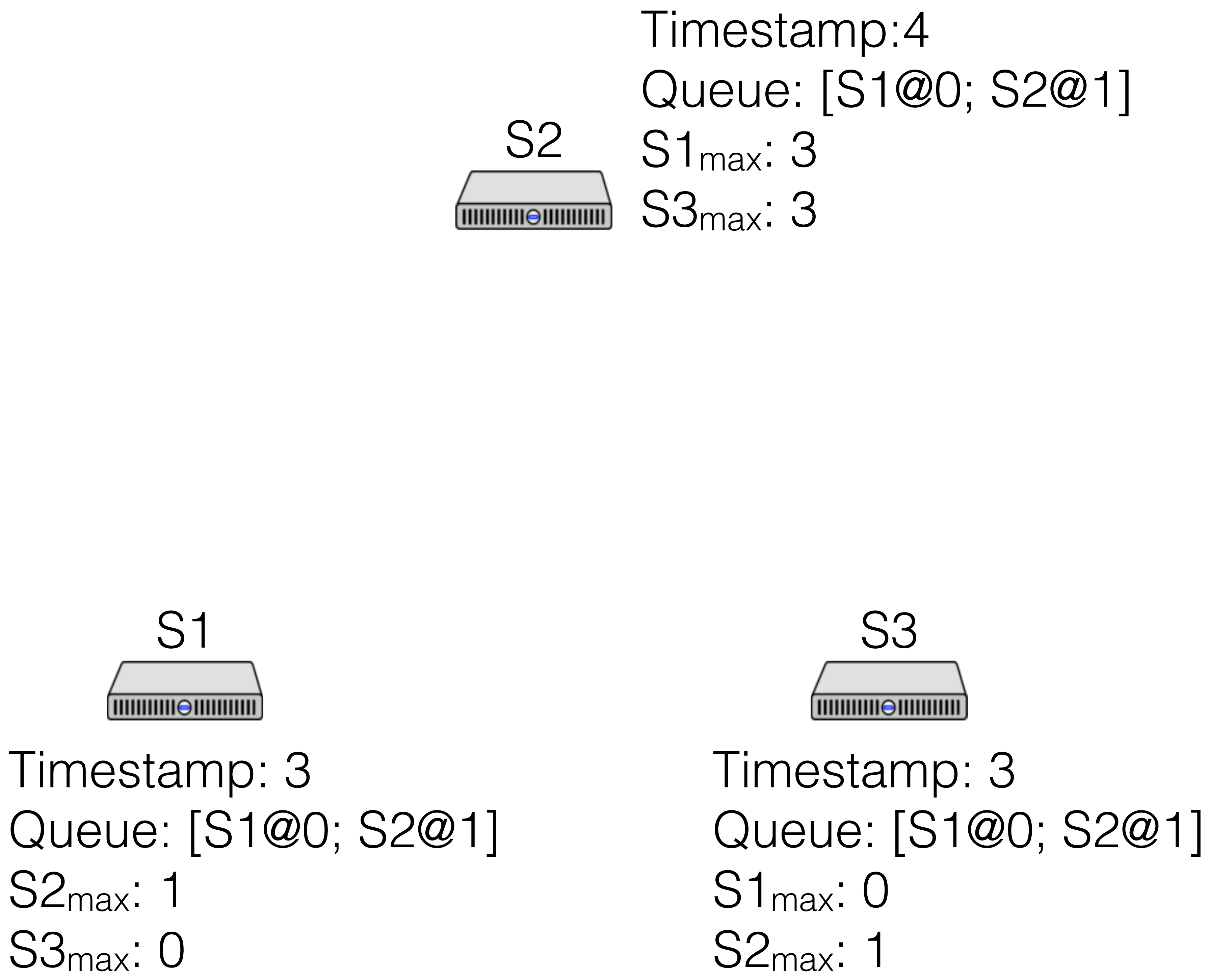
ack@3

S1

Timestamp: 3
Queue: [S1@0; S2@1]
$S2_{max}$: 1
$S3_{max}$: 0

S3

Timestamp: 3
Queue: [S1@0; S2@1]
$S1_{max}$: 0
$S2_{max}$: 1



S2

Timestamp:4

Queue: [S1@0; S2@1]

$S1_{\text{max}}$: 3

$S3_{\text{max}}$: 3

S1

Timestamp: 3

Queue: [S1@0; S2@1]

$S2_{\text{max}}$: 1

$S3_{\text{max}}$: 0

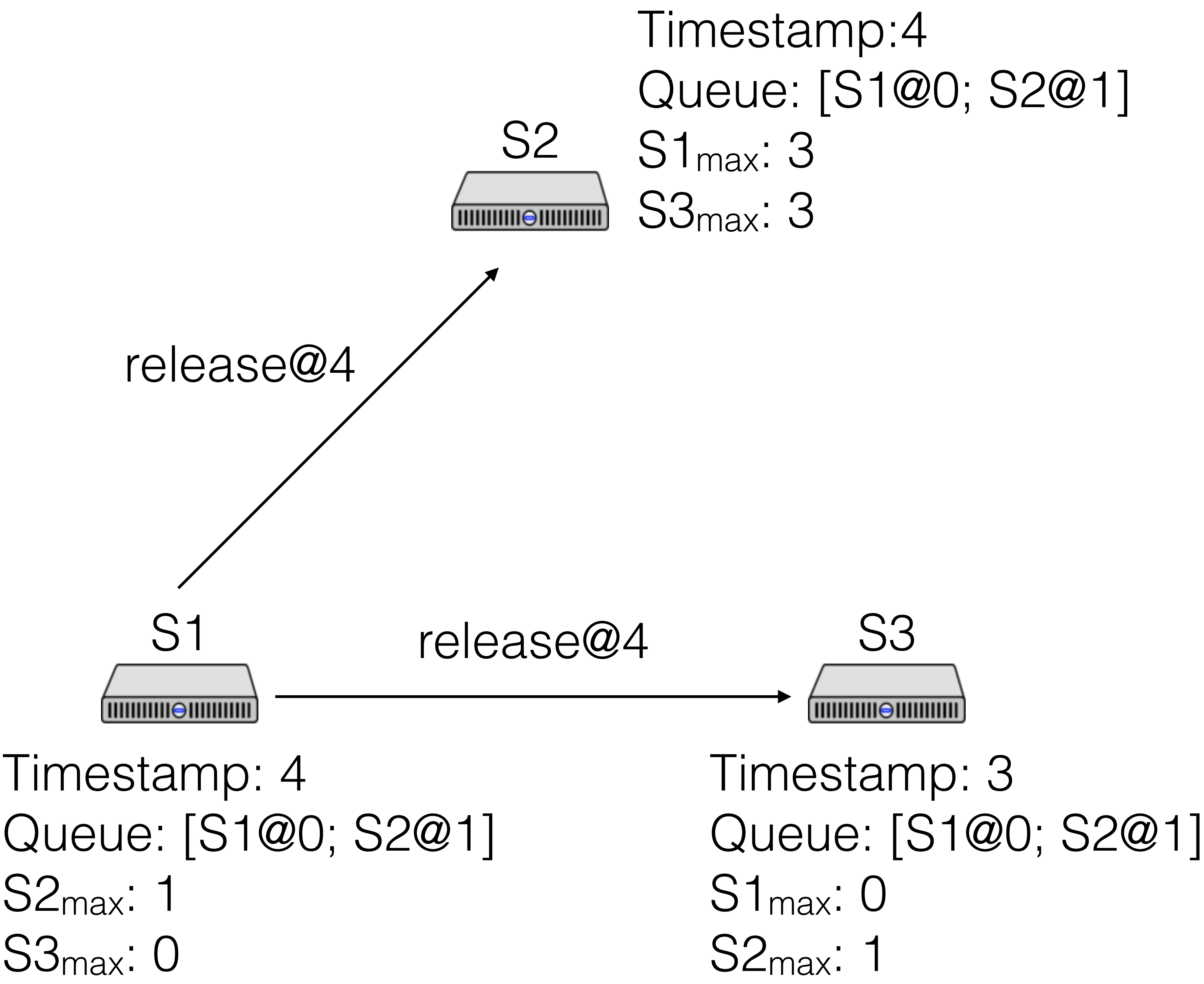
S3

Timestamp: 3

Queue: [S1@0; S2@1]

$S1_{\text{max}}$: 0

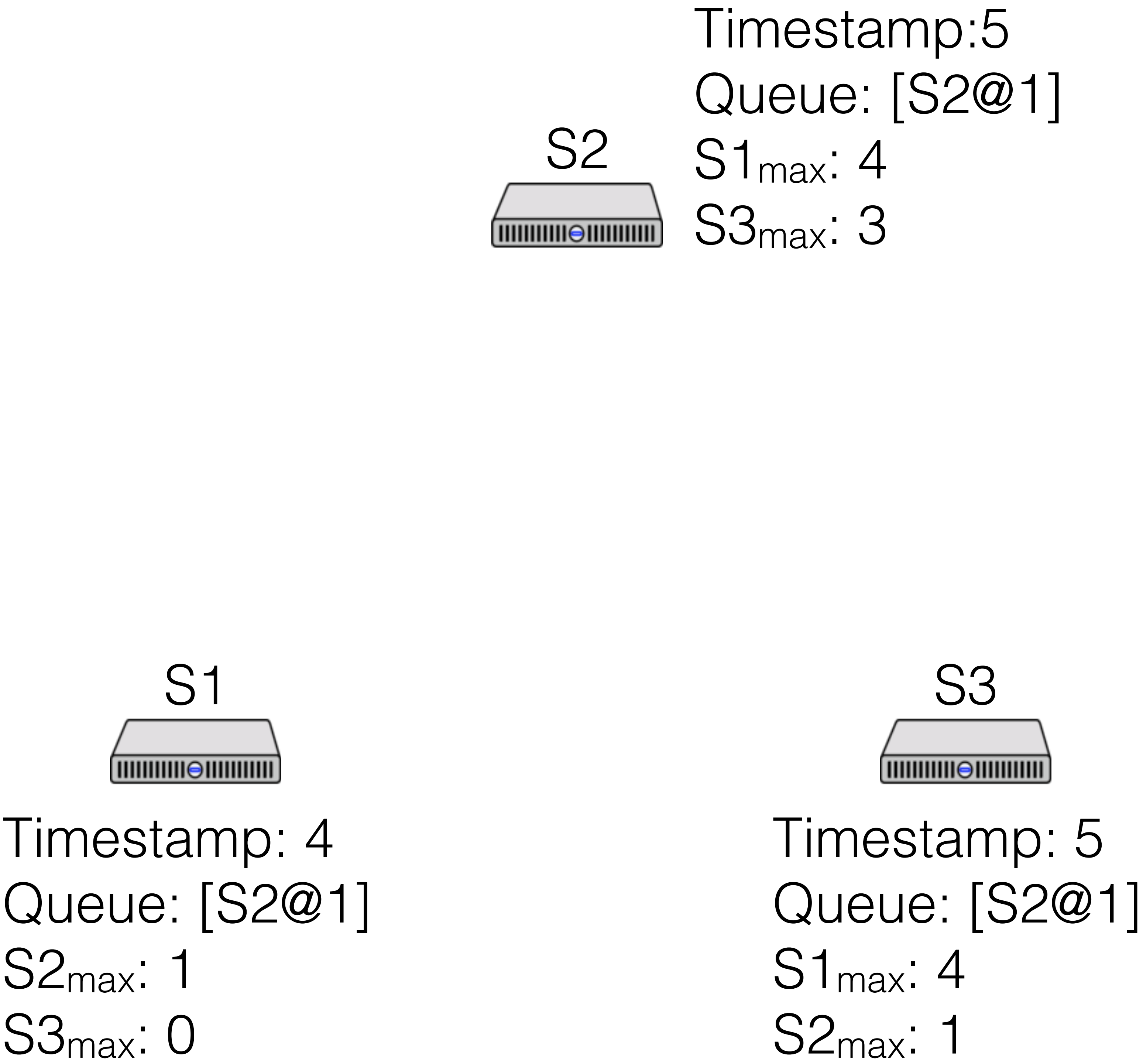
$S2_{\text{max}}$: 1



Timestamp:4
Queue: [S1@0; S2@1]
$S1_{max}$: 3
$S3_{max}$: 3

Timestamp: 4
Queue: [S1@0; S2@1]
$S2_{max}$: 1
$S3_{max}$: 0

Timestamp: 3
Queue: [S1@0; S2@1]
$S1_{max}$: 0
$S2_{max}$: 1



S2

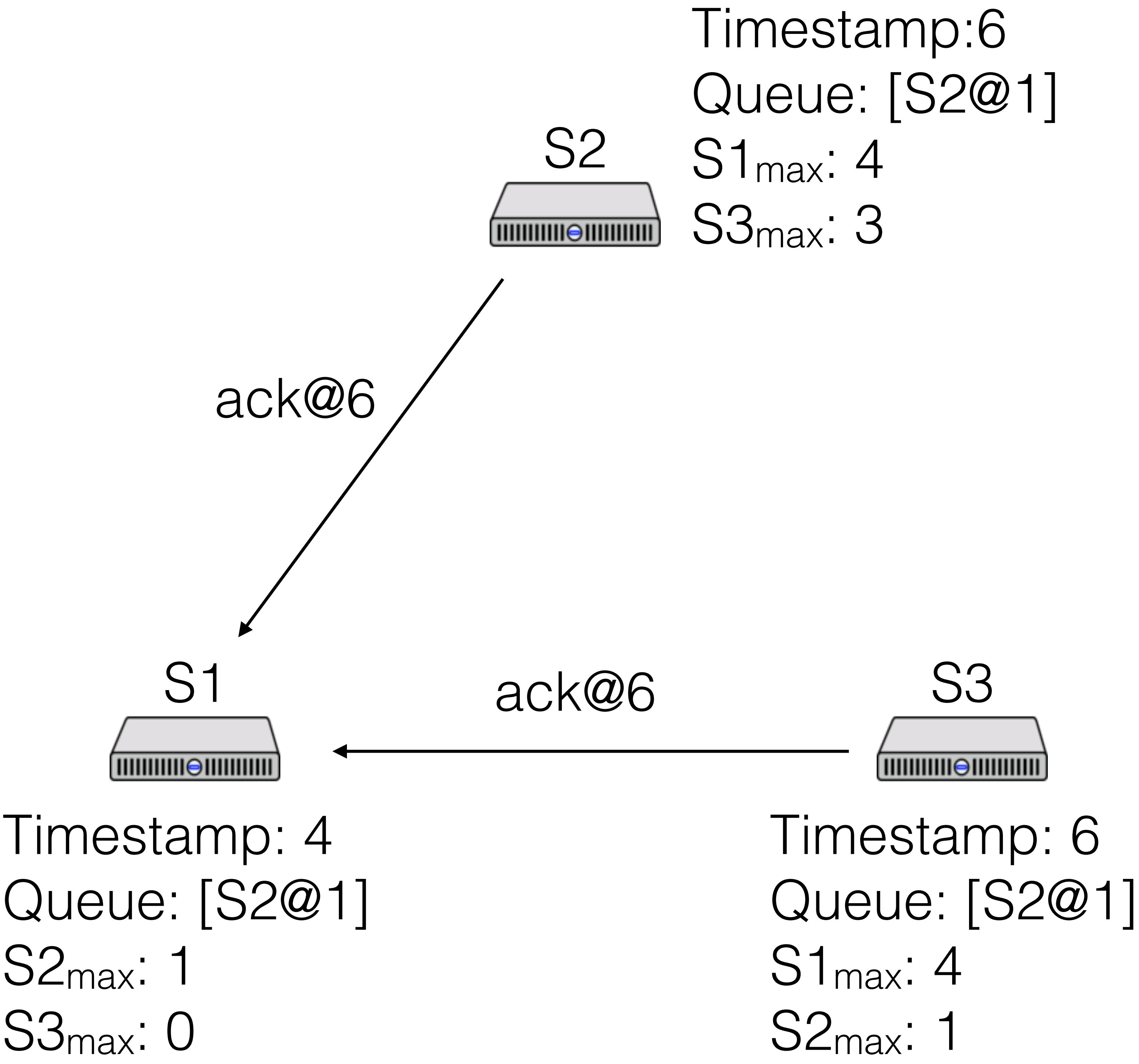
Timestamp:5
Queue: [S2@1]
$S1_{max}$: 4
$S3_{max}$: 3

S1

Timestamp: 4
Queue: [S2@1]
$S2_{max}$: 1
$S3_{max}$: 0

S3

Timestamp: 5
Queue: [S2@1]
$S1_{max}$: 4
$S2_{max}$: 1



S2

Timestamp:6
Queue: [S2@1]
$S1_{max}$: 4
$S3_{max}$: 3

ack@6

S1

Timestamp: 4
Queue: [S2@1]
$S2_{max}$: 1
$S3_{max}$: 0

ack@6

S3

Timestamp: 6
Queue: [S2@1]
$S1_{max}$: 4
$S2_{max}$: 1

S2

Timestamp:6
Queue: [S2@1]
$S1_{max}$: 4
$S3_{max}$: 3

S1

Timestamp: 6
Queue: [S2@1]
$S2_{max}$: 6
$S3_{max}$: 6

S3

Timestamp: 6
Queue: [S2@1]
$S1_{max}$: 4
$S2_{max}$: 1

Questions

- What happens if you don’t have in-order delivery?
- What happens if you eliminate the ack for the request?
- What happens when nodes fail?

Generic State Machine Replication (SMR)

In mutual exclusion:

- State: queue of processes who want the lock
- Commands: P_i *requests*, P_i *releases*

Approach generalizes to other “state machines"

Process a command iff we’ve seen all commands w/ lower timestamp