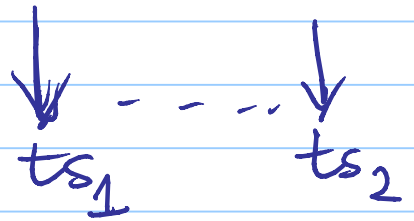
Time – real time → reference clock
– computer time → counter
↓
clock freq
↓
CPU clock

used for – timestamps
– logging
– checkpoints etc

Computer time → never decreases
(monotonic)

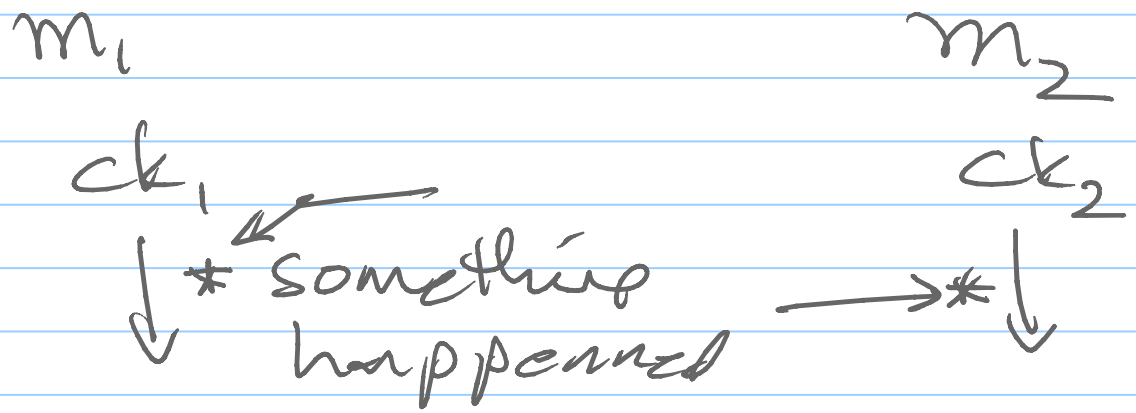
↳ does not sync with real time

events, timestamps

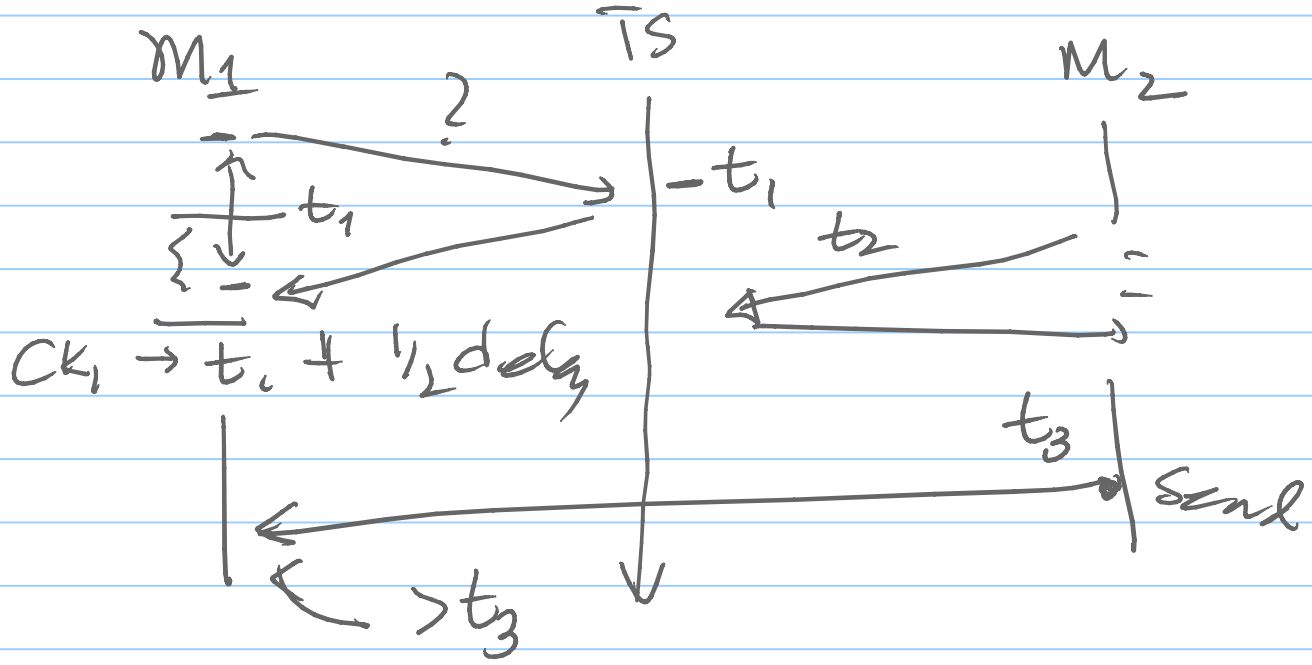


Distributed System

— what is the notion of time



Time servers (NTP)



Leslie Lamport

"Time, Clocks & Event Ordering in a distributed System" – 1978

Event → things that happen
→ an instruction execution
→ file creation / update
→ etc

events in one system are totally ordered.

if e_1 & e_2 are 2 event

then $e_1 \rightarrow e_2$ (or $e_2 \rightarrow e_1$)

'$\rightarrow$' = happened before

Such event ordering also happens in a distributed system —

if e_1 & e_2 happen on same system

$e_1 \rightarrow e_2$ if e_1 happens b4 e_2

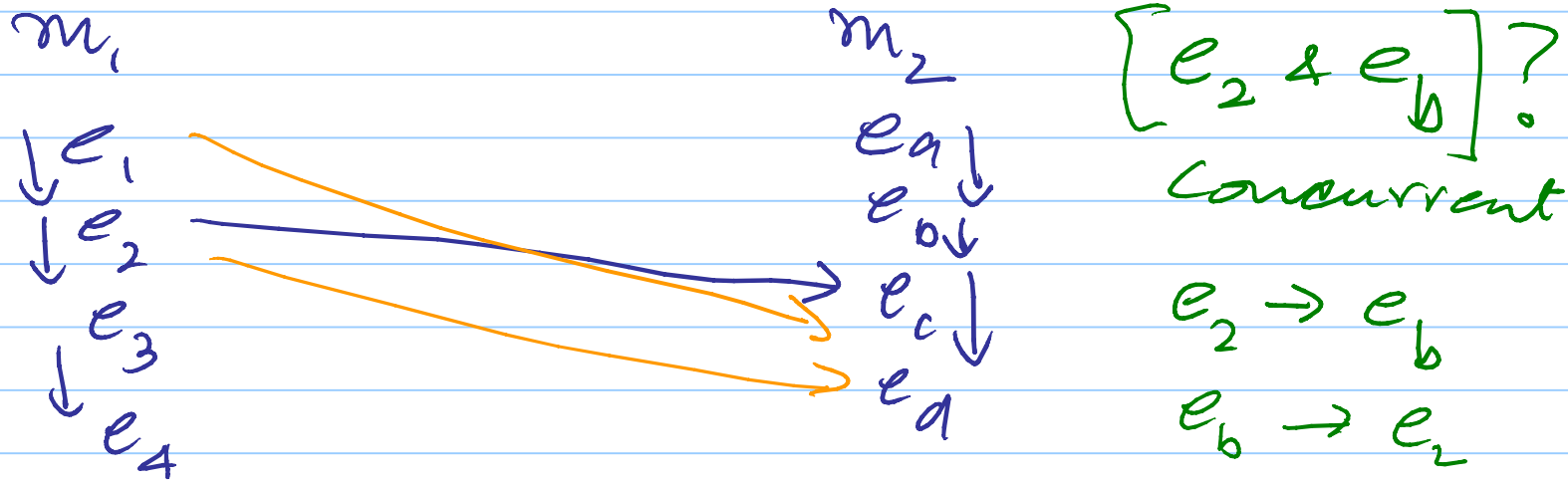
if e_1 & e_2 are on different systems and

$e_1 \equiv send(m)$ & $e_2 \equiv rcv(m)$ ← same msg

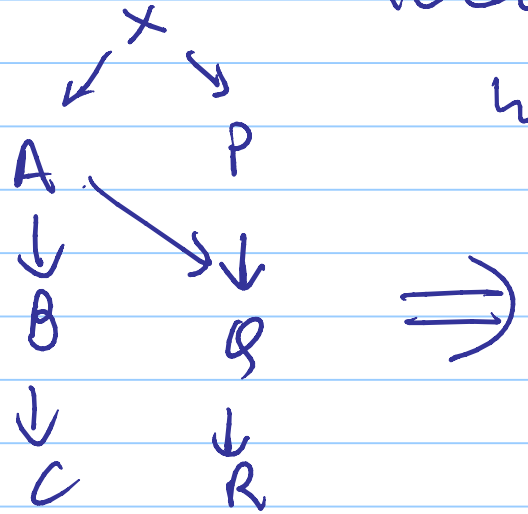
then $e_1 \rightarrow e_2$

ordering
event $\wedge$ is transitive

$$e_1 \to e_2 \;\&\; e_2 \to e_3 \Rightarrow e_1 \to e_3$$



Partial orders can be converted to total orders that are consistant with the partial order



$\Rightarrow$ $X \to A \to P \to B \to Q \to C \to R$

Clocks

→ use a local clock to timestamp every (or some) event.

[if e_i happen at (local) Clock time T then $ts_{e_i} = T$]

[if $e_i \rightarrow e_j$ then $ts_{e_i} < ts_{e_j}$]

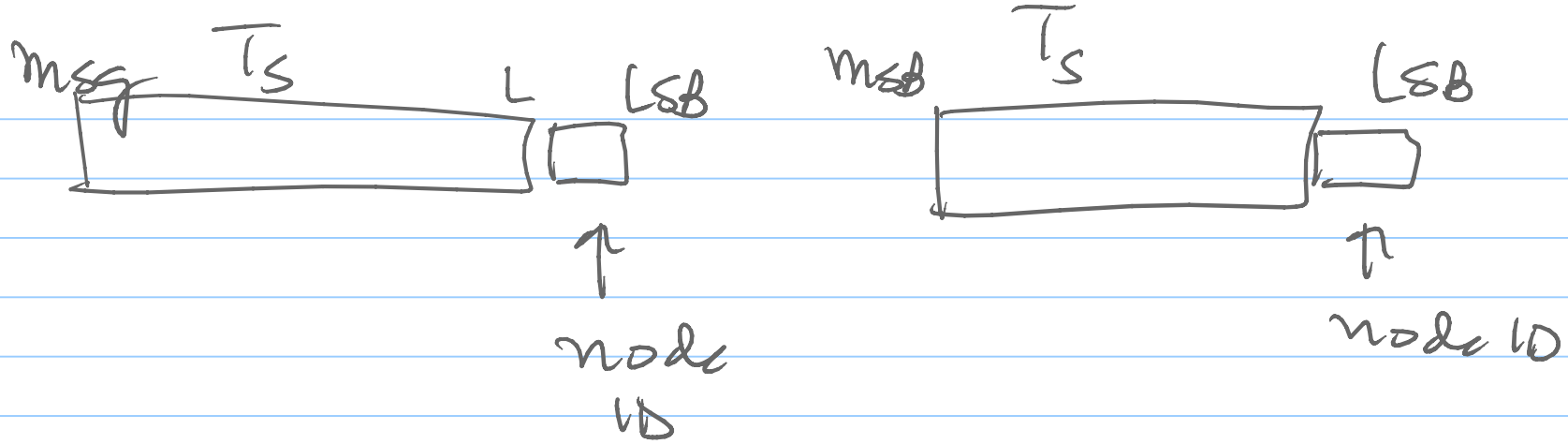
⇒ clocks are correct

Lamport clocks

send (p, m)

→ when m is send, attach local clock value to the msg data

rcv (p, m) →
- get msg from buff
- if msg timestamp > local clock
  - local clock = msg_ts + 1
- return msg


msg
Ts
L
LSB
node ID
MSB
Ts
LSB
node ID

- All events have a time stamp
  (local clock)
  - total order over dist system
  - All timestamps are correct
    (consistent with real event ordering)
    - local clocks are "synchronized"
    - " " can have unbounded SKEW.

C1
200
V
Whats
201
50
201
C2

Lamport clocks — jump

Vector clocks — do not jump

Lamport suggested clocks can be used for dist mutual exclusion (DME)

DME solution

① Lock server (centralized)

② Distributed → no lock server

↳ N processes → they coordinate the critical section

A process wants to enter critical section

→ send n msgs to all processes (include itself) → Request msg

→ wait for acks from everyone

→ keep a Q of requests in TS order (LOCAL)

→ wait till own req @ head of Q.

→ enter CS → at exit send release msg → to everyone

A process gets REQ msg from P_i

→ Add P_i's req to its request Q

→ Send ACK

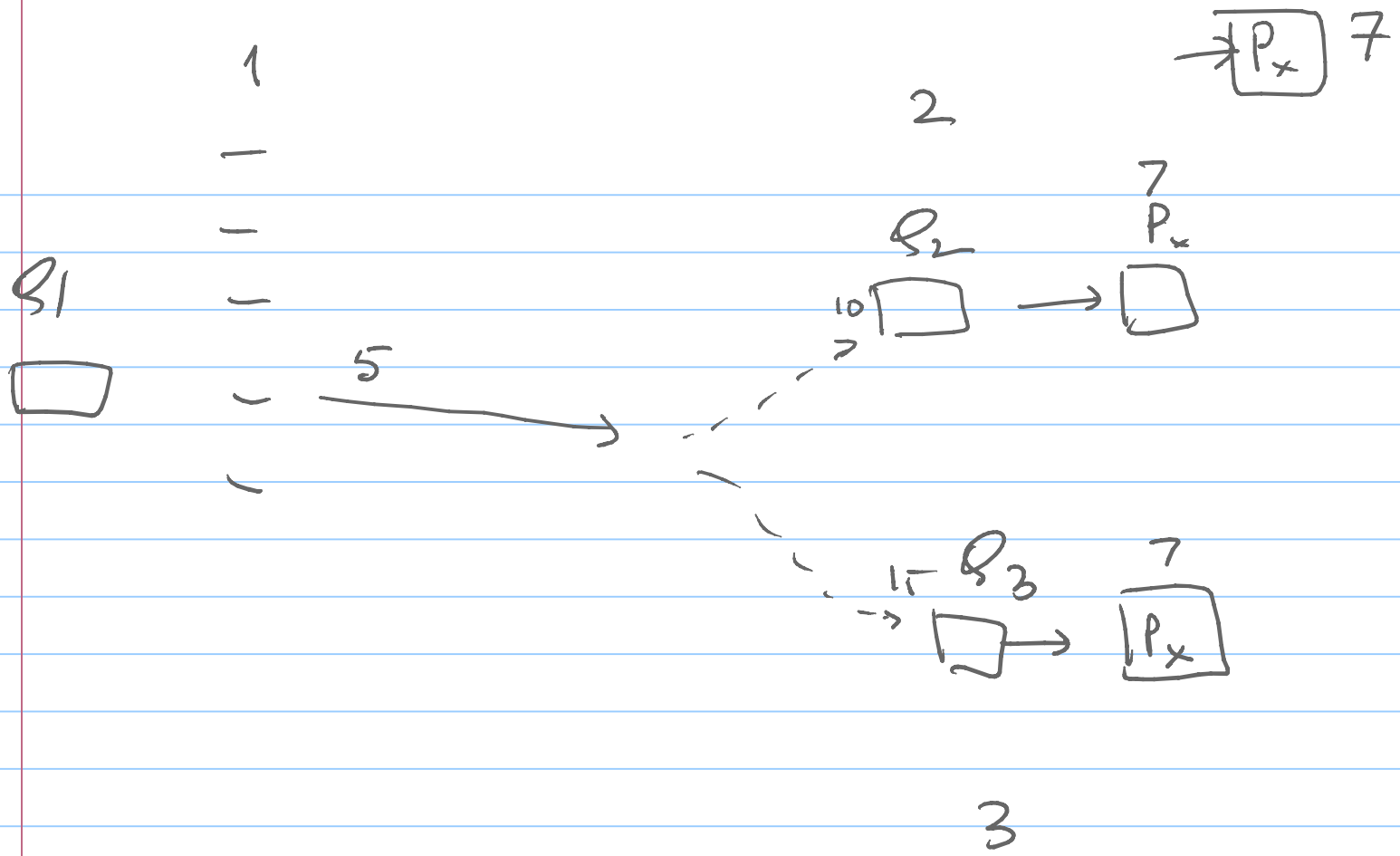
A process gets RELEASE msg from P_i

→ delete P_i's req from Q

→ NOTE → P_i req <u>will be</u> @ head of Q

CS is granted in the order
the req msgs are sent
(i.e. local TS of req msg)

[Central Lock server
→ CS granted in order rcvd]


1
2
3
P_x 7
7
P_x
Q_2
10
Q_1
5
Q_3
15
7
P_x

Lamport → needs $3(n-1)$ msgs

Ricart & Agrawal → $2(n-1)$ msgs

Maekawa → $O\sqrt{n}$ msgs