Time and Ordering of Events

Dominic Duggan
Stevens Institute of Technology
Partly based on material by
Sape Mullender and Ken Birman

1

What time is it?

- Agree that update A occurred before update B
- Offer a “lease” on a resource that expires at time 10:10.0150
- Guarantee that a time critical event will reach all interested parties within 100ms

2

What does time “mean”?

- Time on a global clock?
  - E.g. with GPS receiver
- Machine’s local clock
  - But was it set accurately?
  - And could it drift, e.g. run fast or slow?
  - What about faults, like stuck bits?
- Or try to agree on time

3

3

LOGICAL TIME

4

4

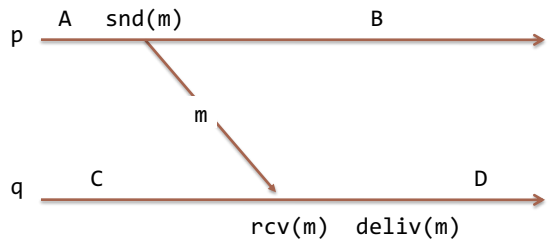
Lamport: Logical Time

- Time lets a system ask "Which came first: event A or event B?"
- Time is a means of labeling events so that…
  - If A happened before B, TIME(A) < TIME(B)
  - If TIME(A) < TIME(B), A happened before B

5

5

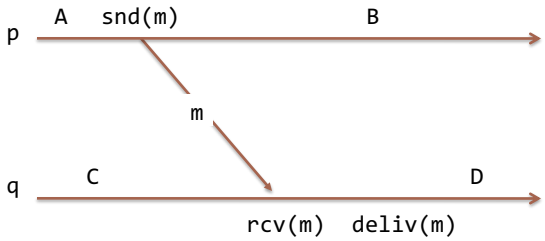
Drawing time-line pictures:



6

6

Drawing time-line pictures:

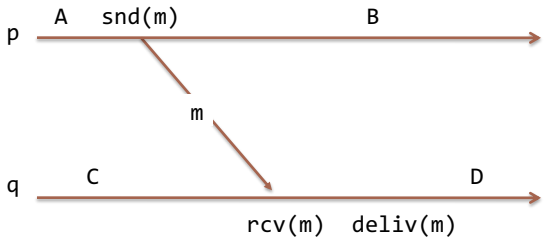


- A, B, C and D are "events".
  - So are snd(m) and rcv(m) and deliv(m)
- What ordering claims are meaningful?

7

7

Drawing time-line pictures:

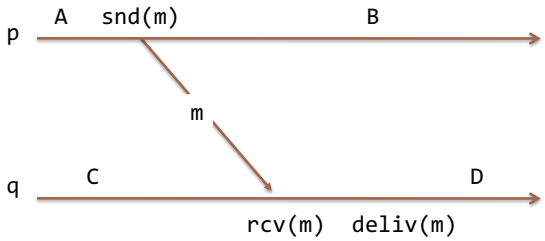


- A happens before B, and C before D
  - *Local ordering* at a single process
  - Write A $\rightarrow^{p}$ B and C $\rightarrow^{q}$ D

8

8

Drawing time-line pictures:

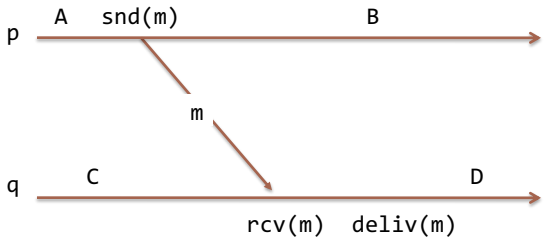


- snd(m) also happens before rcv(m)
  - *Distributed ordering* introduced by a message
  - Write snd(m) → recv(m)

9

9

Drawing time-line pictures:

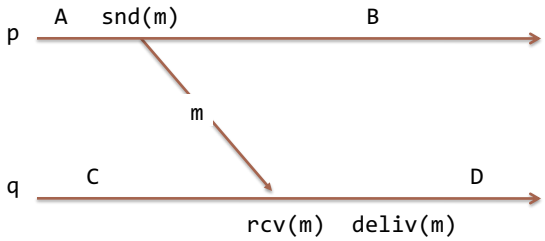


- A happens before D
  - *Transitivity:* A happens before snd(m), which happens before rcv(m), which happens before D

10

10

Drawing time-line pictures:



- B and D are concurrent
  - Looks like B happens first, but D has no way to know. No information flowed…

11

11

"Happens before" relation

- We'll say that "*A happens before B*", written A→B, if
  - $A \rightarrow^{p} B$ according to the local ordering, or
  - A is snd(m) and B is rcv(m) and A→B, or
  - A and B are related under the transitive closure of rules (1) and (2)

12

12

LOGICAL CLOCKS

13

13

Logical clocks

- First version: uses just a single integer
  - Designed for big (64-bit or more) counters
  - Each process p maintains LT_p, a local counter
  - A message m will carry timestamp TS(m)

14

14

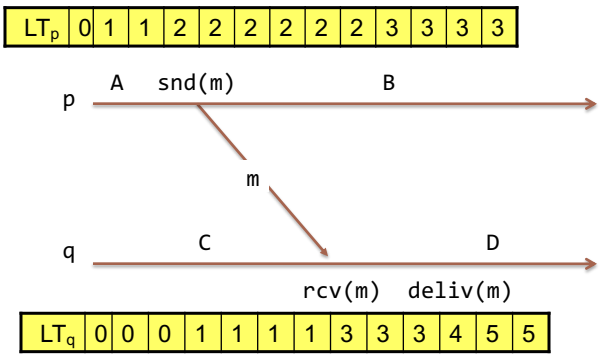
Rules for managing logical clocks

- When an event happens at a process p it increments LT_p
  - Any event that matters to p
  - Normally, also snd and rcv events (since we want receive to occur "after" the matching send)
- When p sends m, set
  - $TS(m) = LT_p$
- When q receives m, set
  - $LT_q = \max(LT_q, TS(m))+1$

15

15

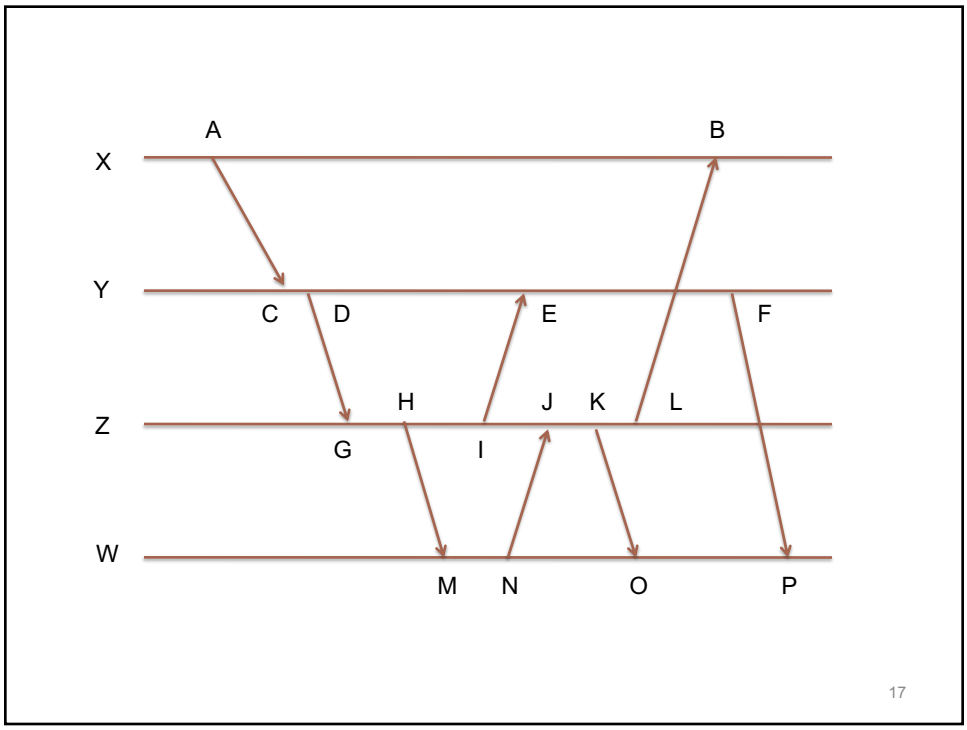
Time-line with LT annotations



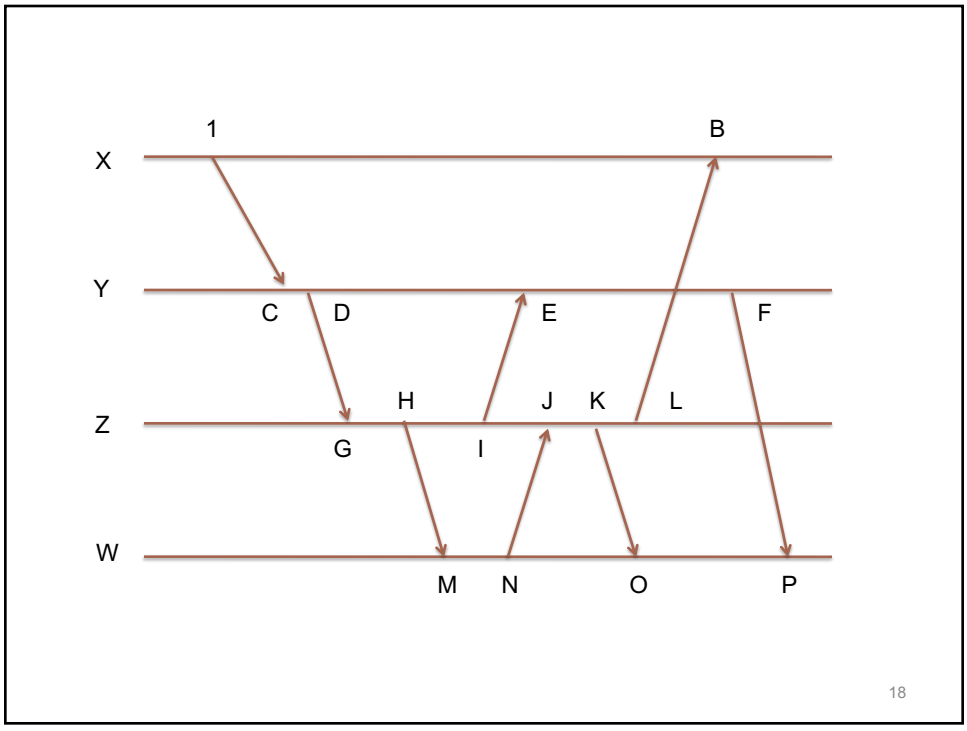
- LT(A) = 1, LT(snd(m)) = 2, TS(m) = 2
- LT(rcv(m))=max(1,2)+1=3, etc…

16

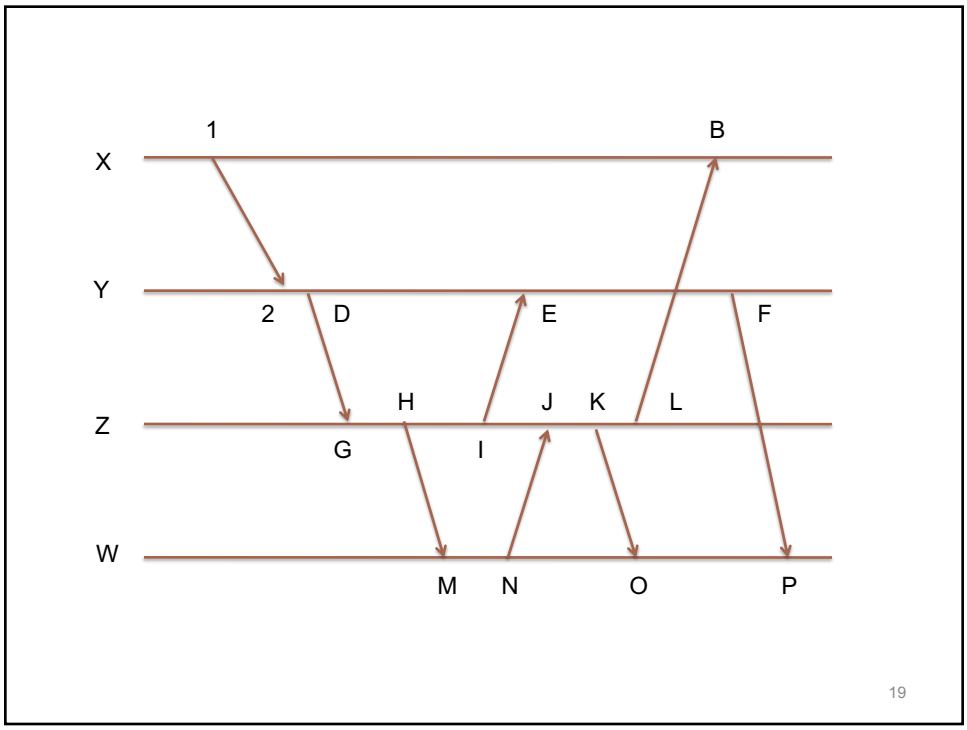
16



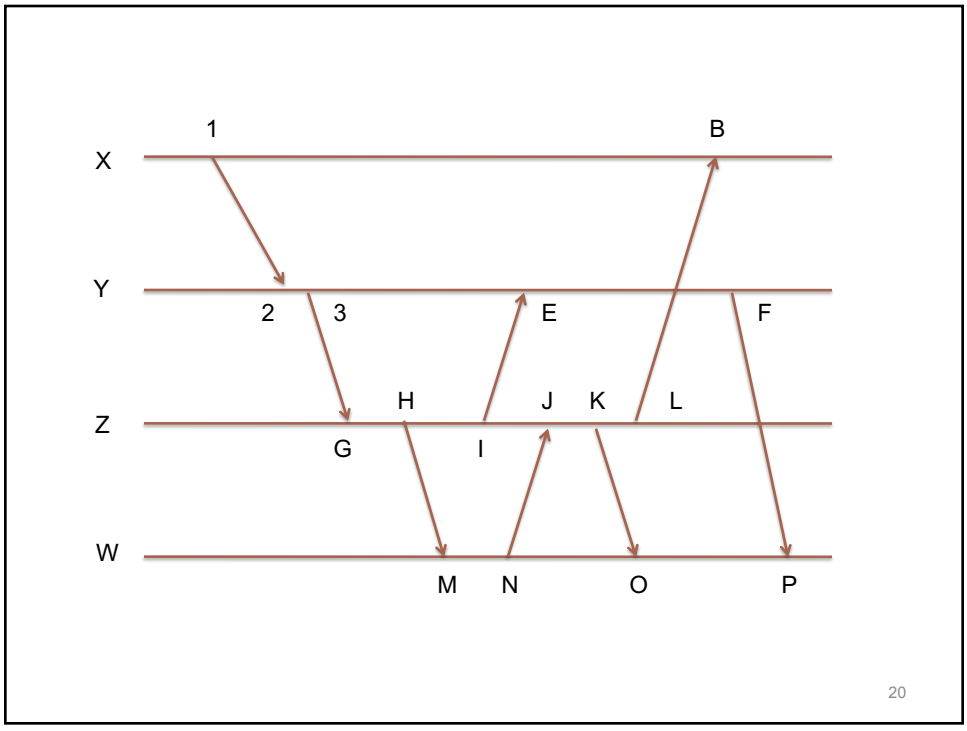
17



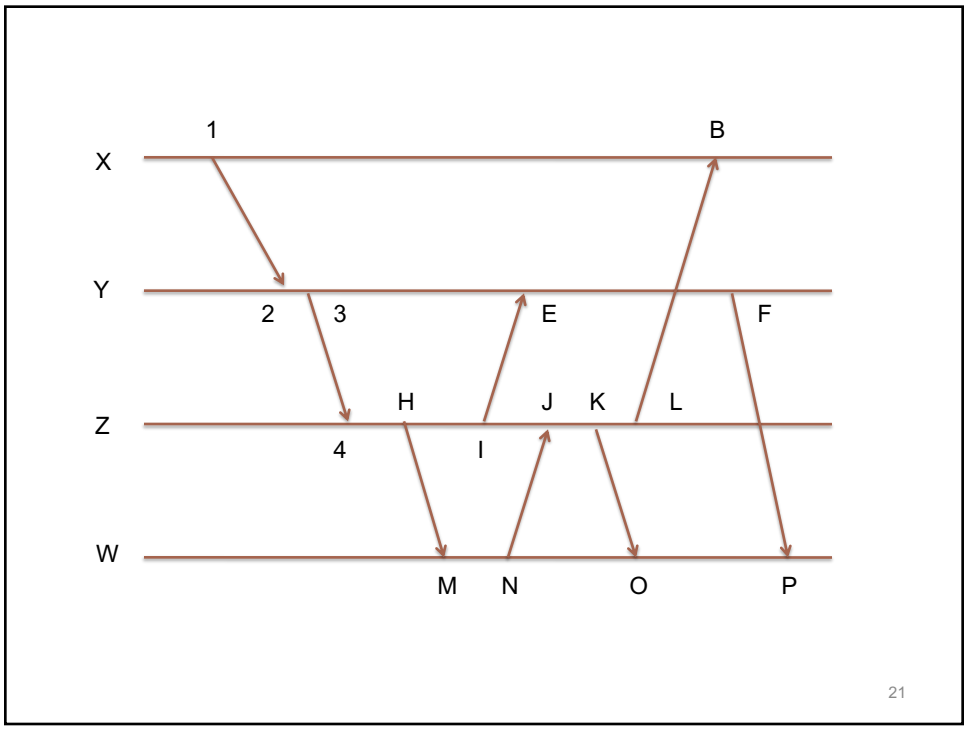
18



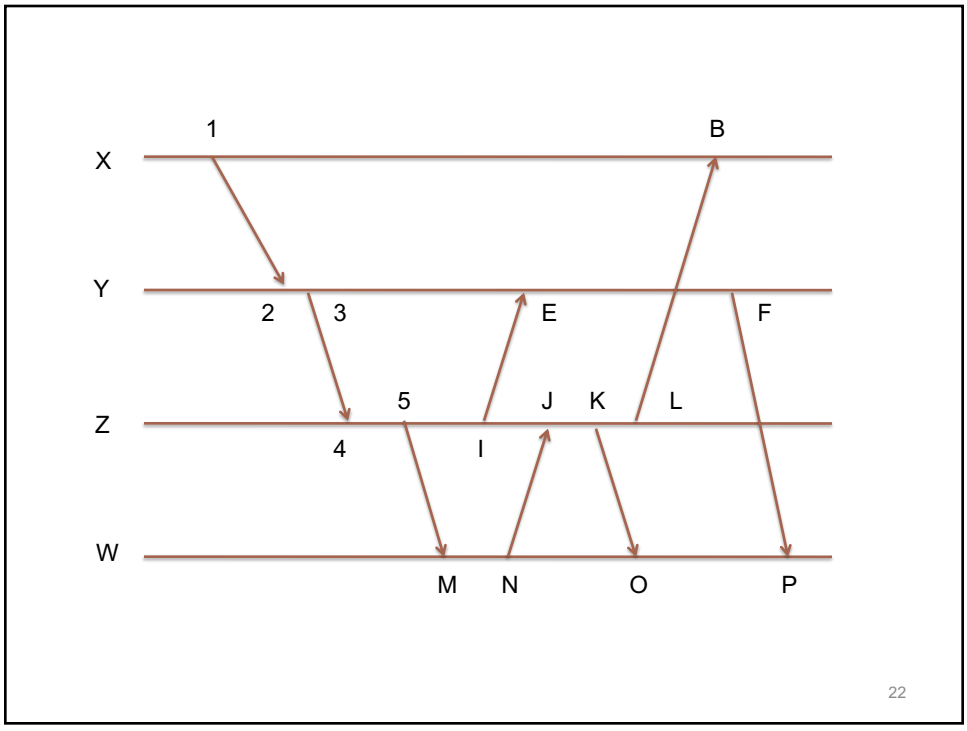
19



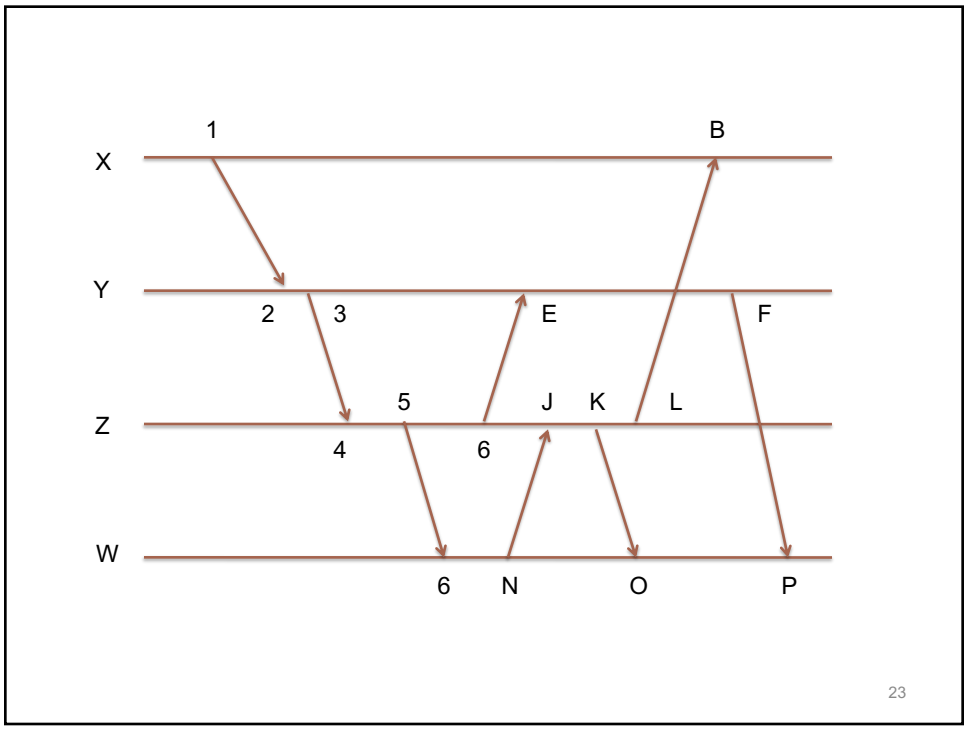
20



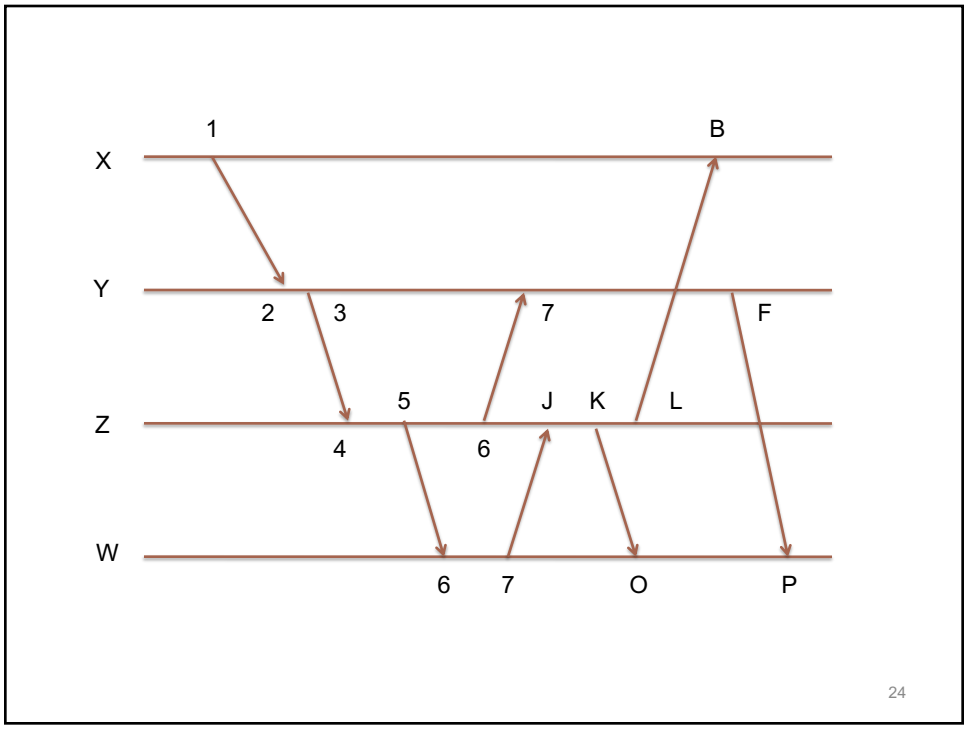
21



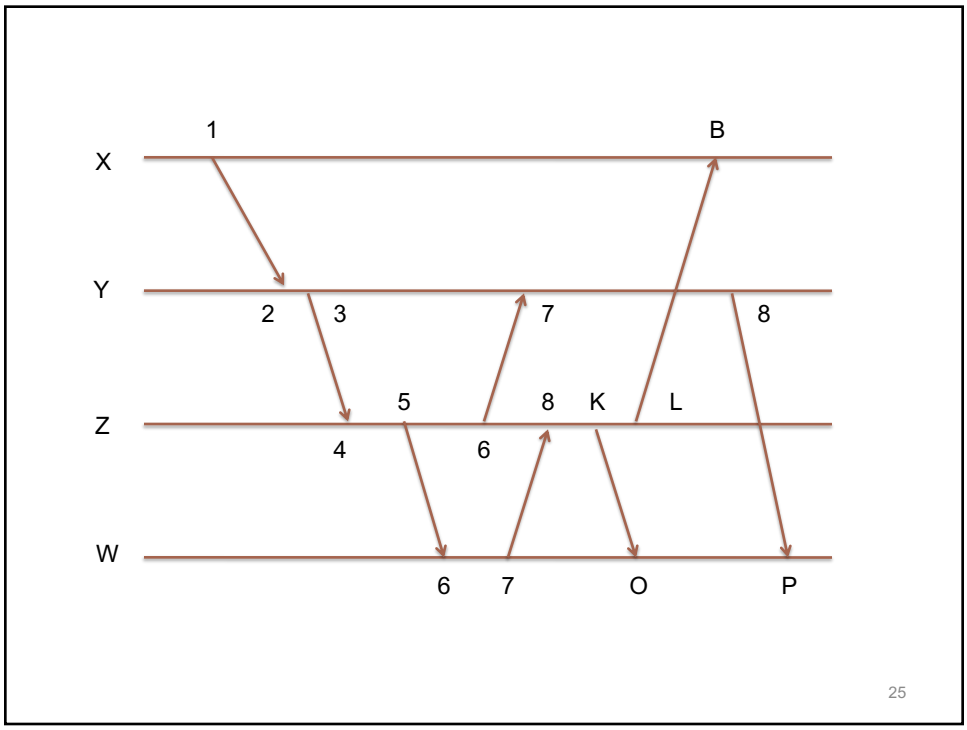
22



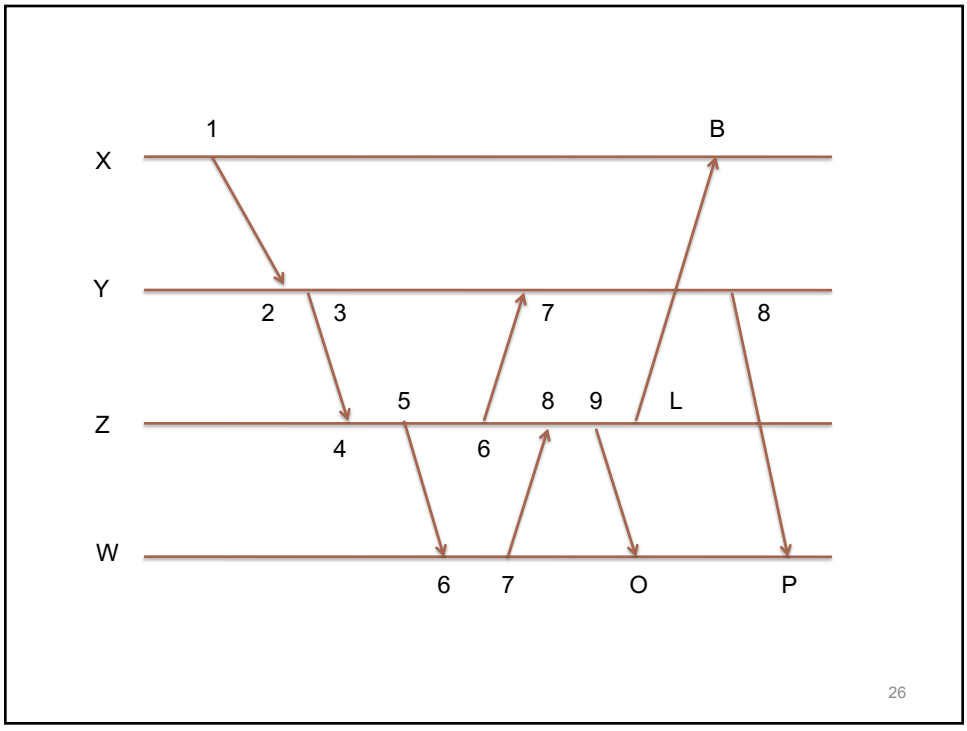
23



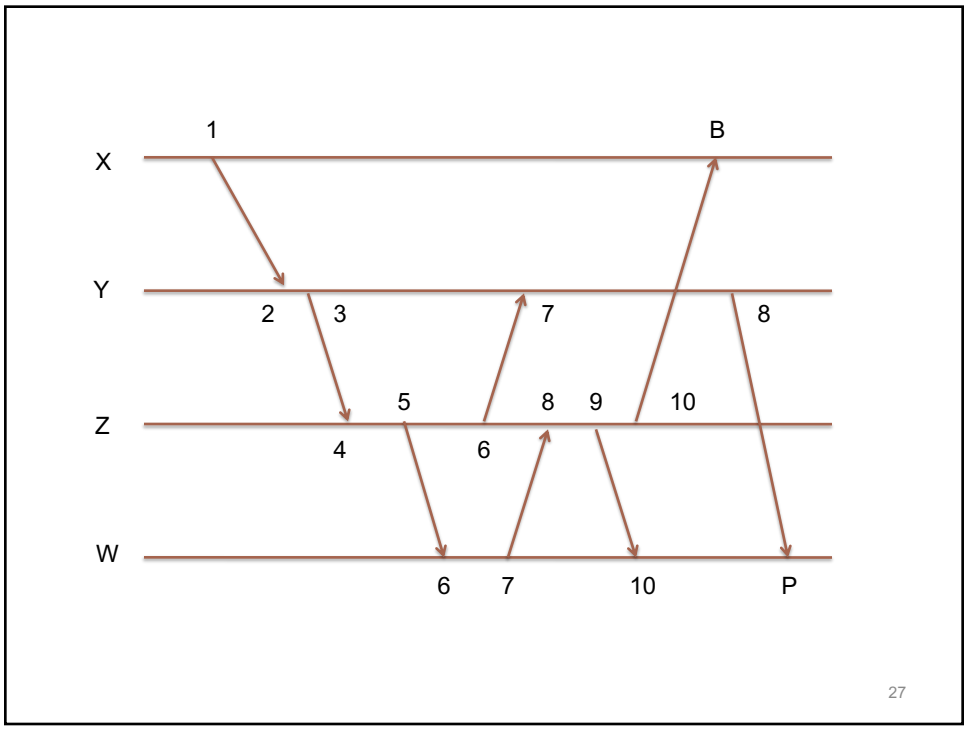
24



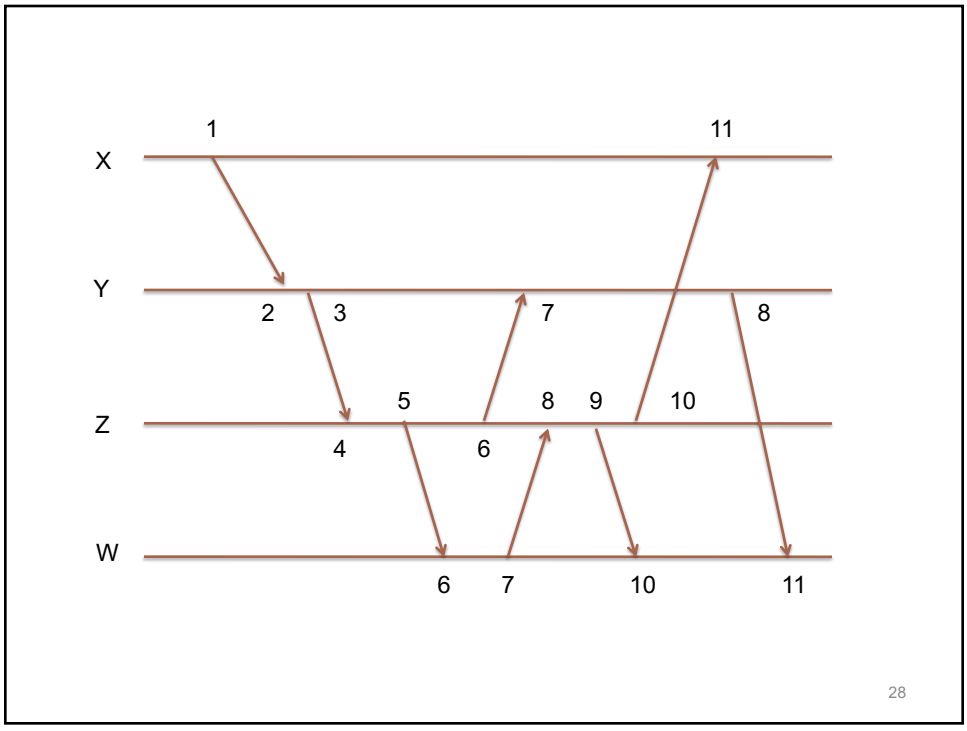
25



26


1
B
X
Y
2
3
7
8
5
8
9
10
Z
4
6
W
6
7
10
P
27

27


1
11
X
Y
2
3
7
8
5
8
9
10
Z
4
6
W
6
7
10
11
28

28

DISTRIBUTED MUTUAL EXCLUSION

29

29

Distributed Mutual Exclusion

- Idea: purely distributed protocol for mutually exclusive access to a resource
  - No central coordinator

30

30

Distributed Mutual Exclusion

- Idea: purely distributed protocol for mutually exclusive access to a resource
  - No central coordinator
- Requests are ordered using logical time
  - Use (ts, pid) with pid to break ties
  - (m, p) < (n, q) if m < n or (m = n and p < q)

31

31

Distributed Mutual Exclusion

- Idea: purely distributed protocol for mutually exclusive access to a resource
  - No central coordinator
- Requests are ordered using logical time
  - Use (ts, pid) with pid to break ties
  - (m, p) < (n, q) if m < n or (m = n and p < q)
- Data structures
  - Logical time LT_p
  - Request queue, ordered by request timestamp
  - LT[i], timestamp of last message received from process p_i

32

32

Request a Resource

- Process p_i
  - Increments its logical clock
  - Adds request m (TS(m)=LT_i) to its request queue
  - Broadcasts request to every other process

33

33

Request a Resource

- Process p_i
  - Increments its logical clock
  - Adds request m (TS(m)=LT_i) to its request queue
  - Broadcasts request to every other process
- Process p_j ($j \neq i$)
  - Acknowledges receipt of request (TS(ack)=LT_j)

34

34

Request a Resource

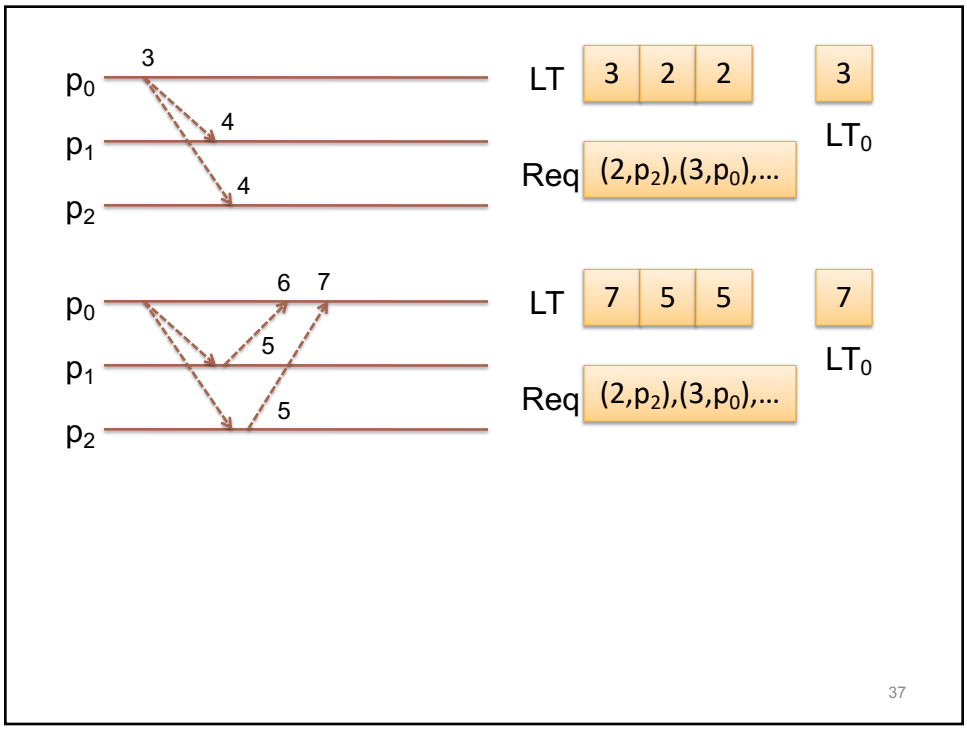
- Process p_i
  - Increments its logical clock
  - Adds request m (TS(m)=LT_i) to its request queue
  - Broadcasts request to every other process
- Process p_j ($j \neq i$)
  - Acknowledges receipt of request (TS(ack)=LT_j)
- Process pi has access when:
  - Its request is in the front of its request queue
  - LT[i] $\geq$ TS(m) for all i=1,…,n

35

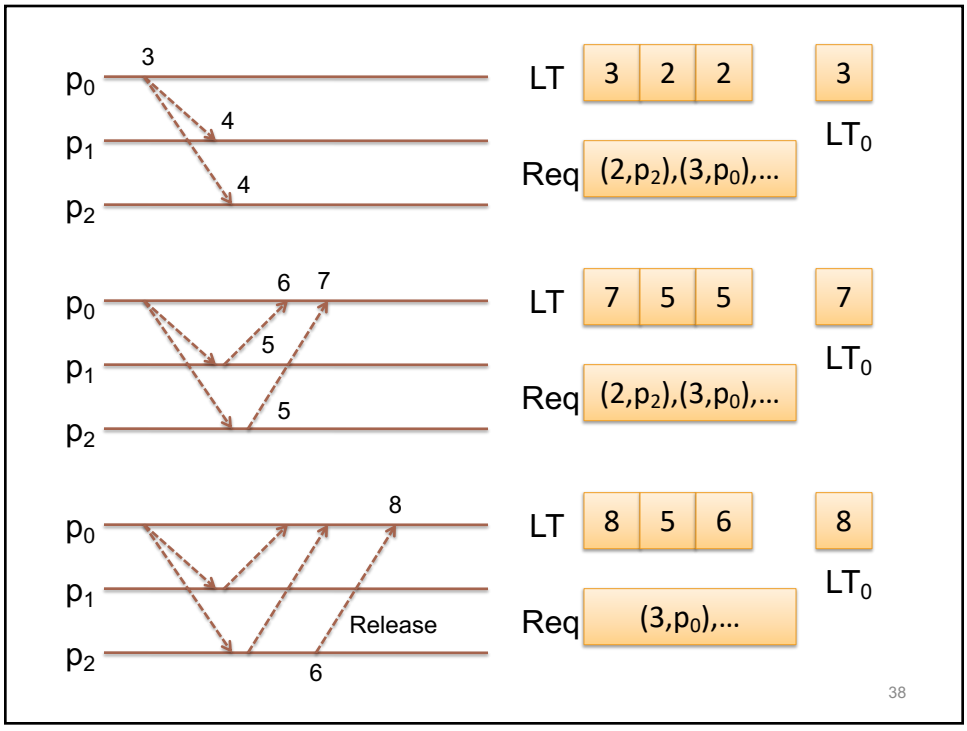
35

36

36


p0
p1
p2
3
4
4
LT
3 2 2
3
LT0
Req
(2,p2),(3,p0),...
6 7
5
5
LT
7 5 5
7
LT0
Req
(2,p2),(3,p0),...
37

37


p0
p1
p2
3
4
4
LT
3 2 2
3
LT0
Req
(2,p2),(3,p0),...
6 7
5
5
LT
7 5 5
7
LT0
Req
(2,p2),(3,p0),...
8
Release
6
LT
8 5 6
8
LT0
Req
(3,p0),...
38

38