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”?

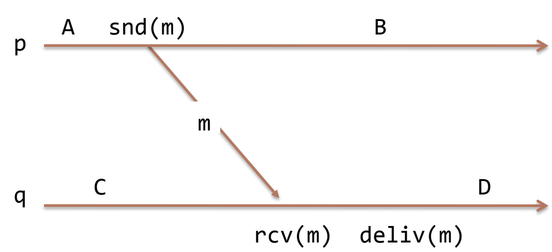
- Time on a global clock?
  - E.g. with GPS receiver
- Machine’s local clock
  - But was it set accurately?
  - And could it drift, e.g. run fast or slow?
  - What about faults, like stuck bits?
- Or try to agree on time

LOGICAL TIME

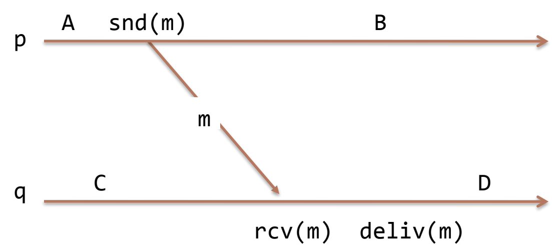
Lamport: Logical Time

- 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

Drawing time-line pictures:

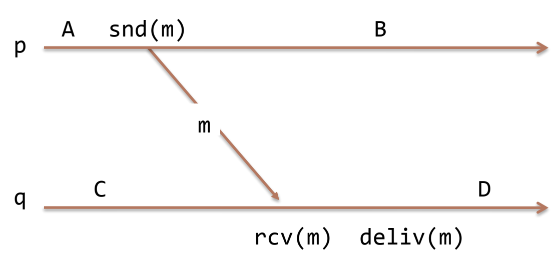


Drawing time-line pictures:



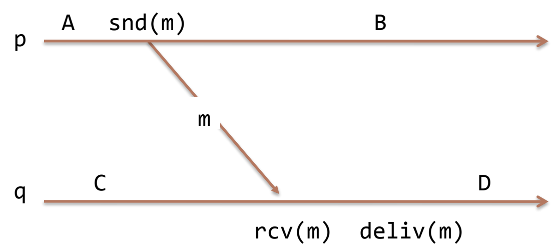
- A, B, C and D are “events”.
  - So are snd(m) and rcv(m) and deliv(m)
- What ordering claims are meaningful?

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

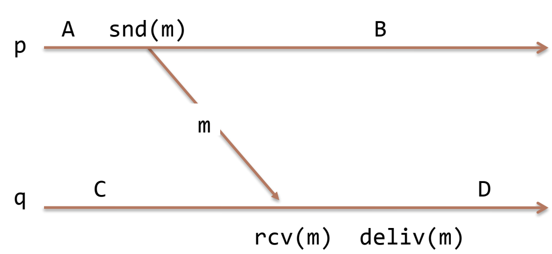
Drawing time-line pictures:



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

9

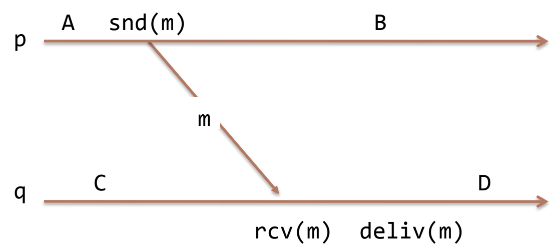
Drawing time-line pictures:



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

10

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

"Happens before" relation

- We'll say that "*A happens before B*", written A→B, if
  - A→^{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

LOGICAL CLOCKS

13

Logical clocks

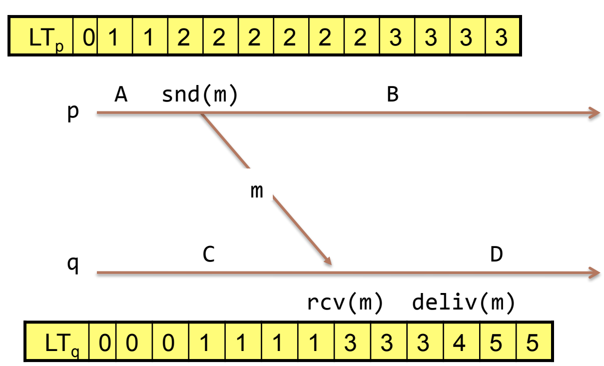
- First version: uses just a single integer
  - Designed for big (64-bit or more) counters
  - Each process p maintains LT_p, a local counter
  - A message m will carry timestamp TS(m)

14

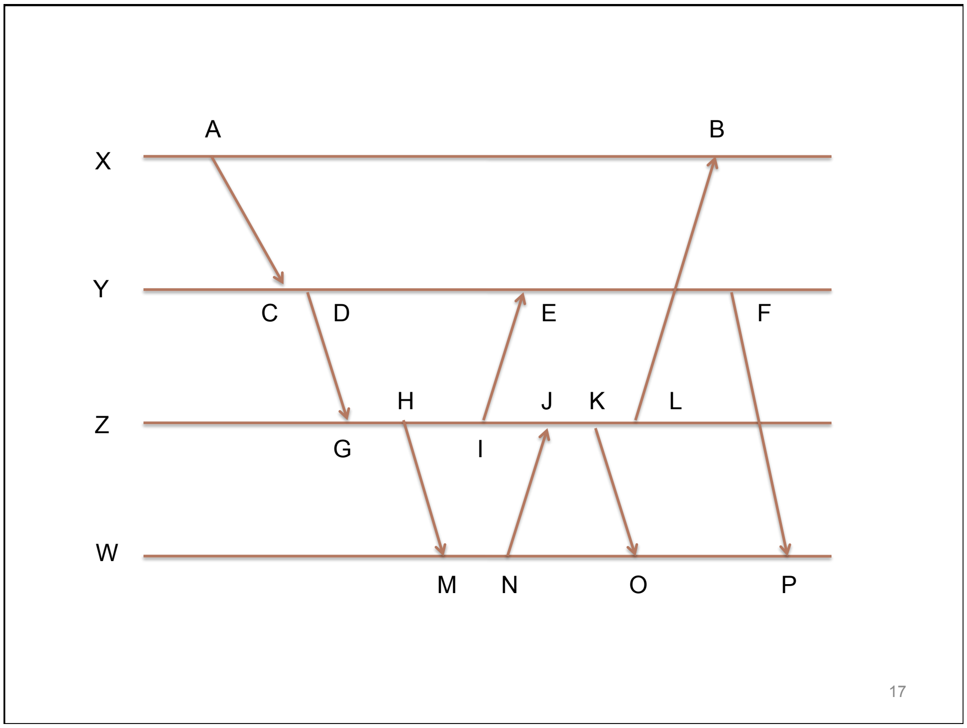
Rules for managing logical clocks

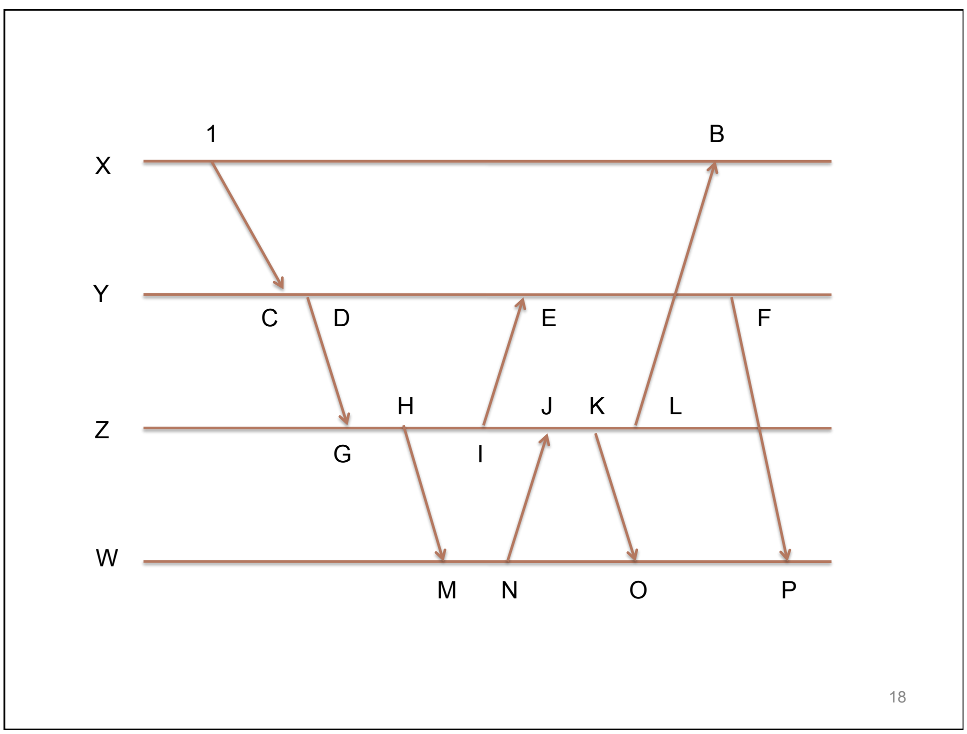
- When an event happens at a process p it increments LT_p
  - Any event that matters to p
  - Normally, also snd and rcv events (since we want receive to occur "after" the matching send)
- When p sends m, set
  - $TS(m) = LT_p$
- When q receives m, set
  - $LT_q = \max(LT_q, TS(m))+1$

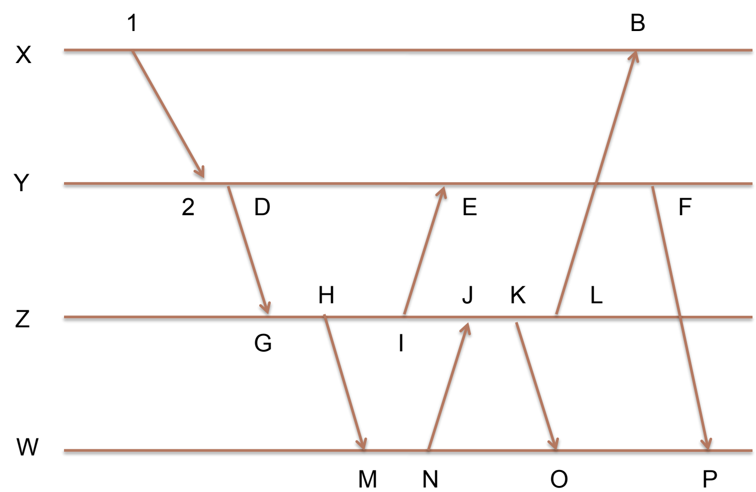
Time-line with LT annotations

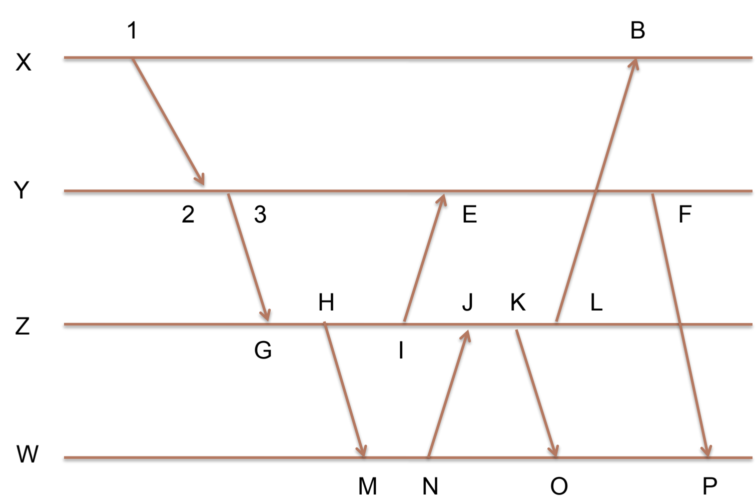


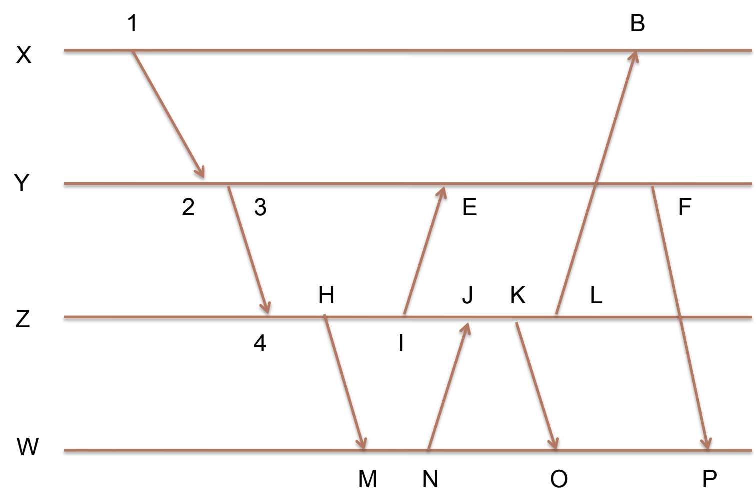
- LT(A) = 1, LT(snd(m)) = 2, TS(m) = 2
- LT(rcv(m))=max(1,2)+1=3, etc…


A
B
X
Y
C
D
E
F
H
J
K
L
Z
G
I
W
M
N
O
P
17

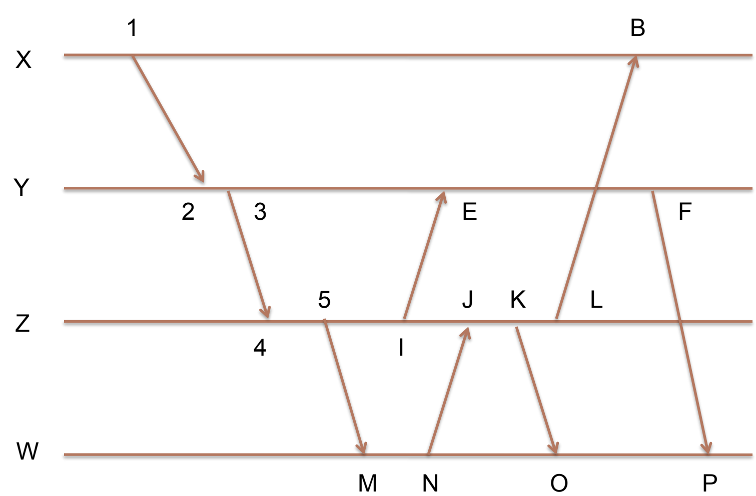

1
B
X
Y
C
D
E
F
H
J
K
L
Z
G
I
W
M
N
O
P
18


1
B
X
Y
2
D
E
F
H
J
K
L
Z
G
I
W
M
N
O
P

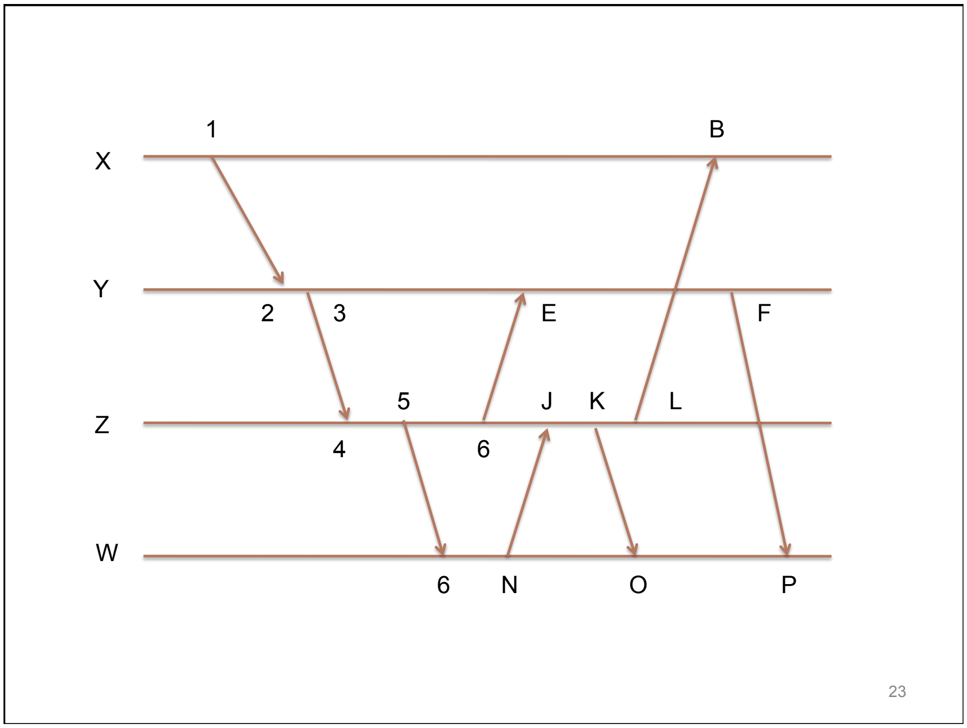

1
B
X
Y
2
3
E
F
H
J
K
L
Z
G
I
W
M
N
O
P

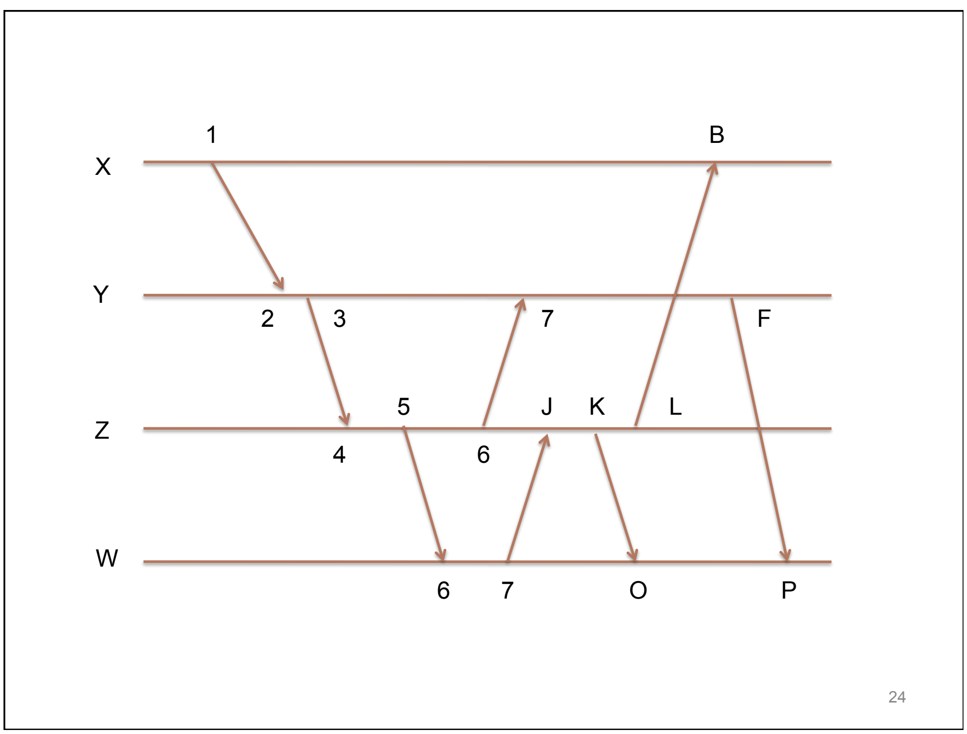


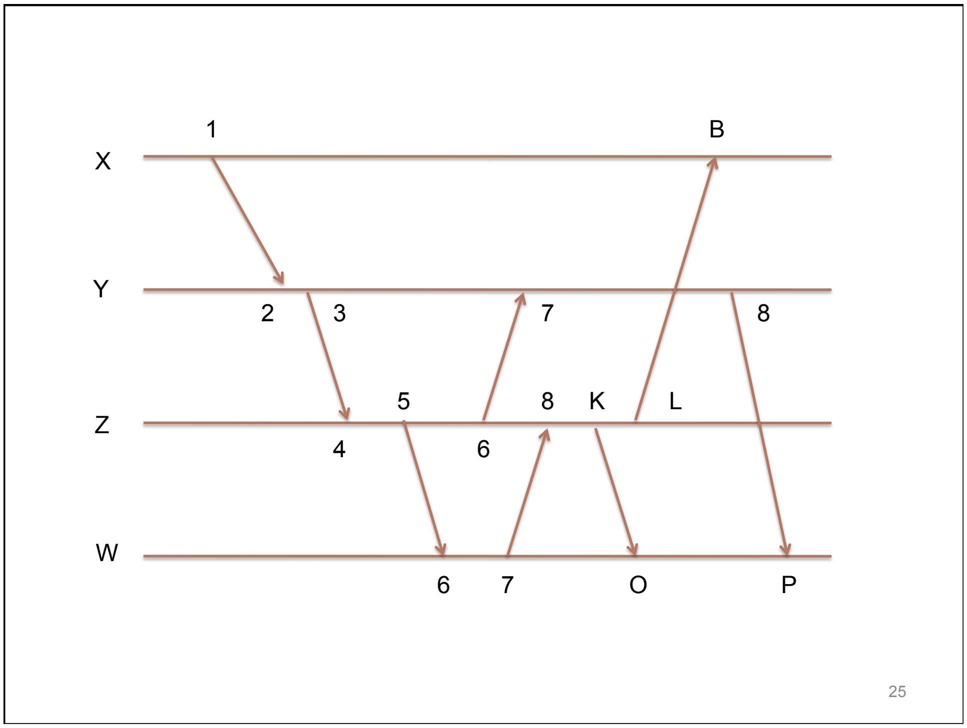
21

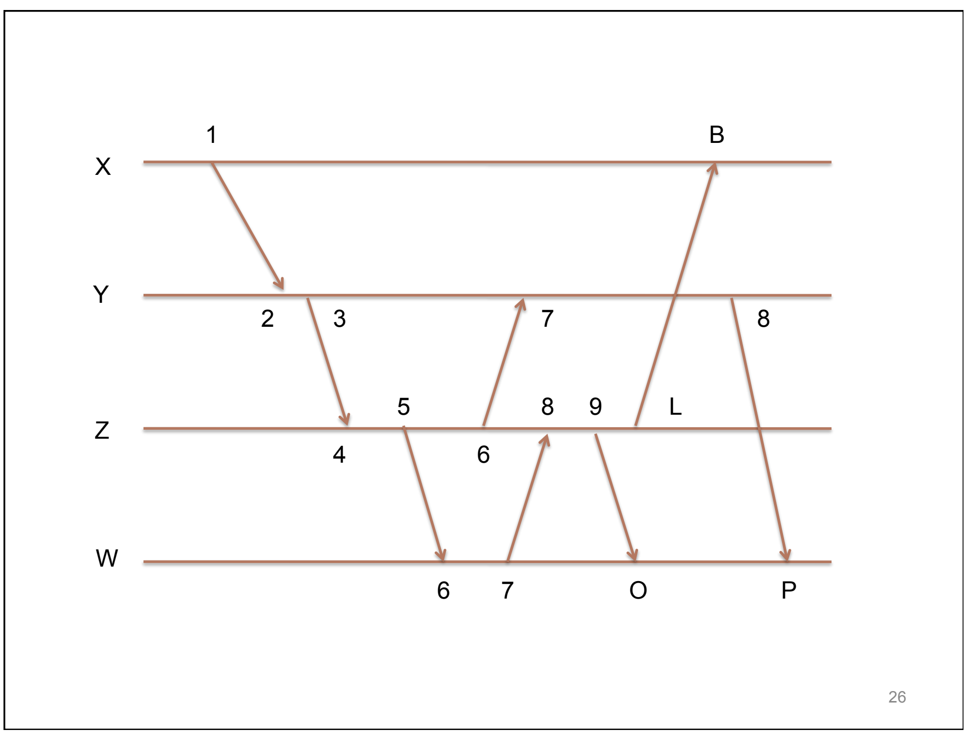


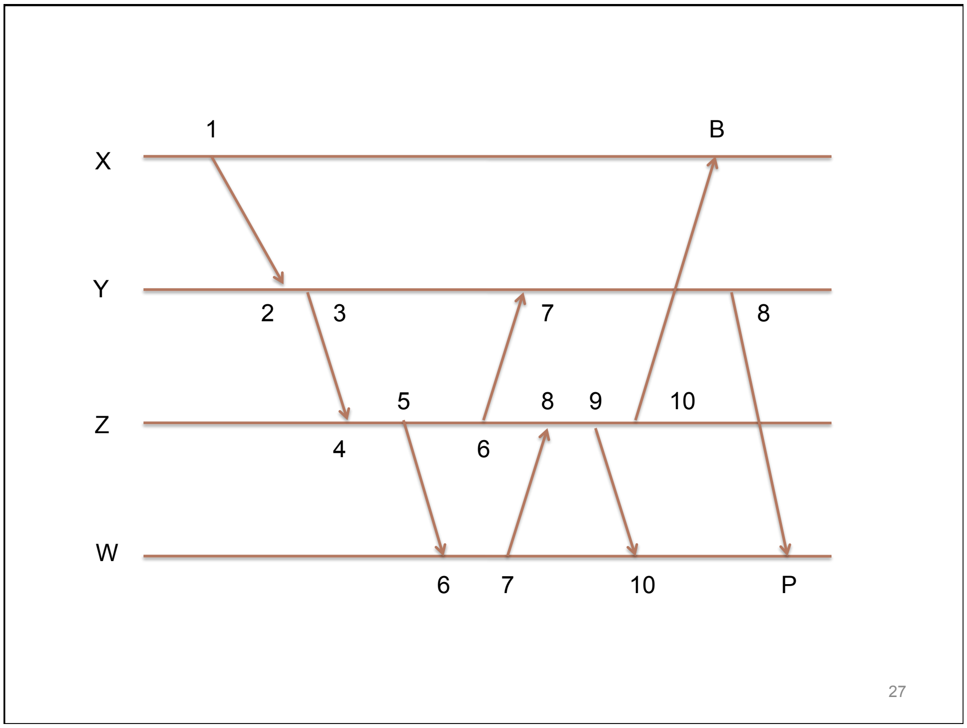
22

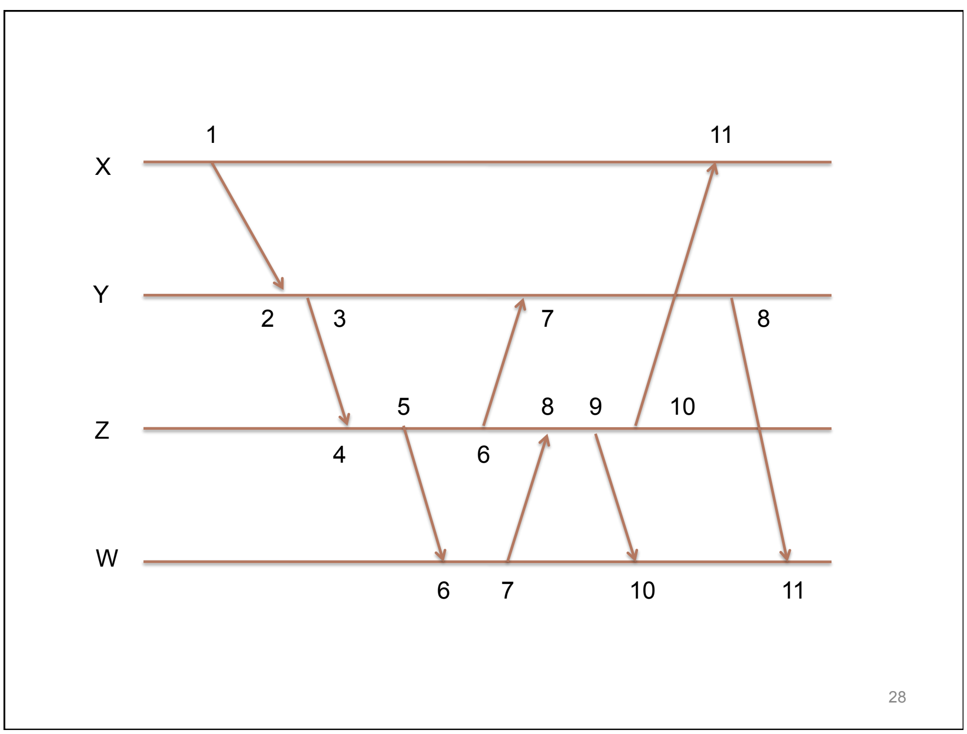

1
B
X
Y
2
3
E
F
5
J
K
L
Z
4
6
W
6
N
O
P
23


1
B
X
Y
2
3
7
F
5
J
K
L
Z
4
6
W
6
7
O
P
24


1
B
X
Y
2
3
7
8
5
8
K
L
Z
4
6
W
6
7
O
P
25


1
B
X
Y
2
3
7
8
5
8
9
L
Z
4
6
W
6
7
O
P
26


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


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

DISTRIBUTED MUTUAL EXCLUSION

Distributed Mutual Exclusion

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

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

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

Request a Resource

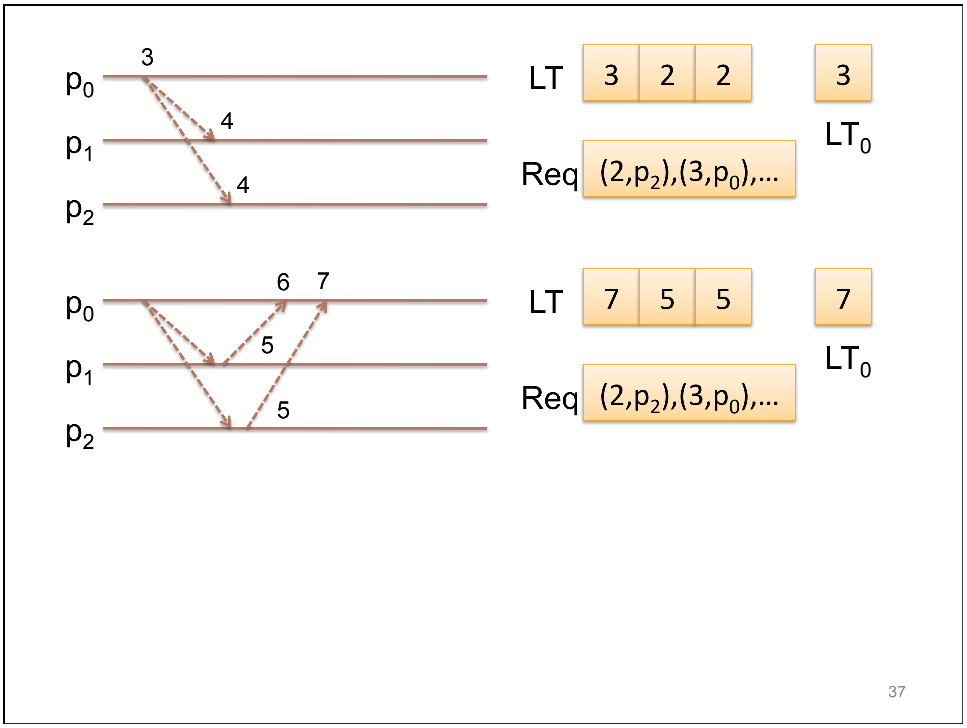
- Process p_i
  - Increments its logical clock
  - Adds request m ($TS(m)=LT_i$) to its request queue
  - Broadcasts request to every other process

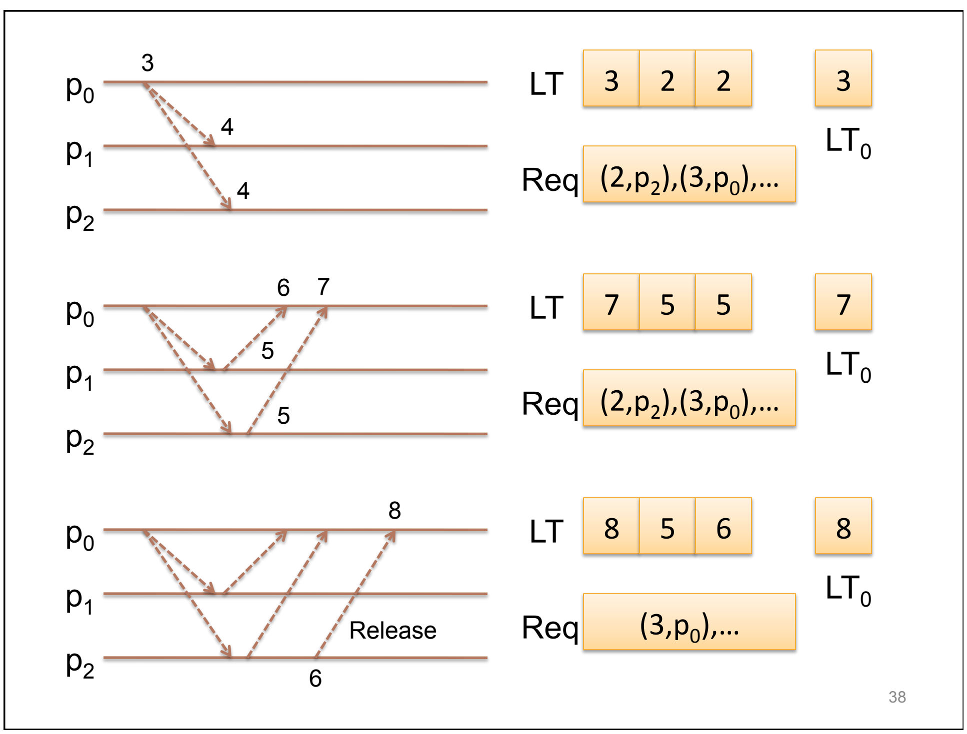
Request a Resource

- 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 ($TS(ack)=LT_j$)

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


p_0
p_1
p_2
3
4
4
LT
3 2 2
3
LT_0
Req
$(2,p_2),(3,p_0),\ldots$
p_0
p_1
p_2
6 7
5
5
LT
7 5 5
7
LT_0
Req
$(2,p_2),(3,p_0),\ldots$
37

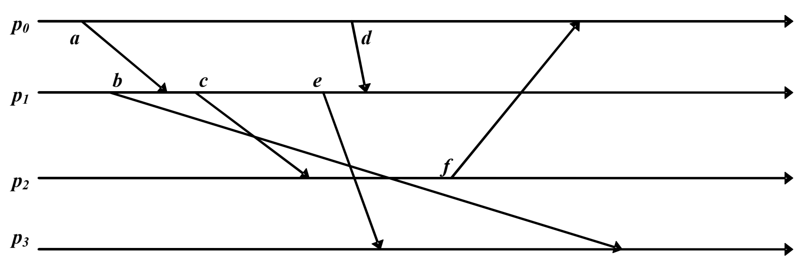

p_0
p_1
p_2
3
4
4
LT
3 2 2
3
LT_0
Req
$(2,p_2),(3,p_0),\ldots$
p_0
p_1
p_2
6 7
5
5
LT
7 5 5
7
LT_0
Req
$(2,p_2),(3,p_0),\ldots$
p_0
p_1
p_2
8
Release
6
LT
8 5 6
8
LT_0
Req
$(3,p_0),\ldots$
38

CONSISTENT CUTS

39

Temporal distortions

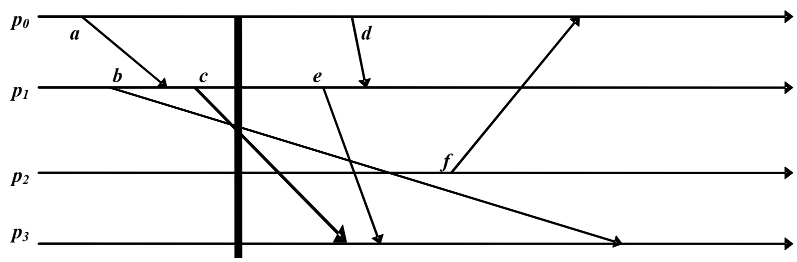
- What does "now" mean?



40

Temporal distortions

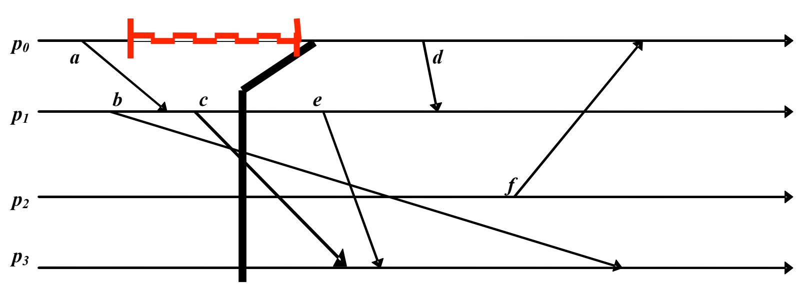
- What does "now" mean?



41

Temporal distortions

- Timelines can "stretch"…

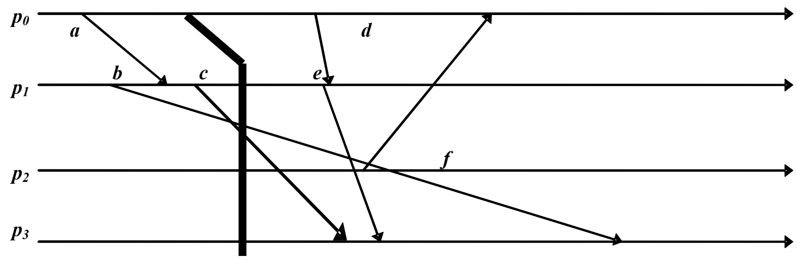


- … caused by scheduling effects, message delays, message loss…

42

Temporal distortions

- Timelines can “shrink”

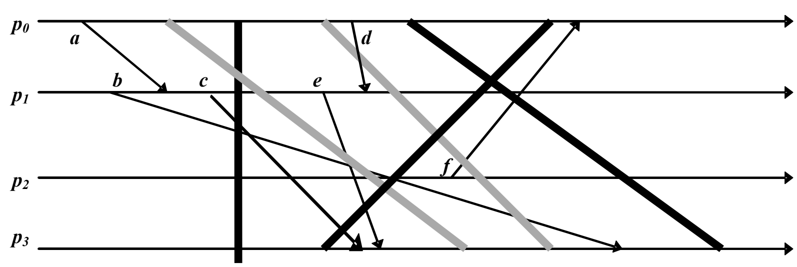


- E.g. something lets a machine speed up

43

Temporal distortions

- Cuts represent instants of time.



- But not every “cut” makes sense

44

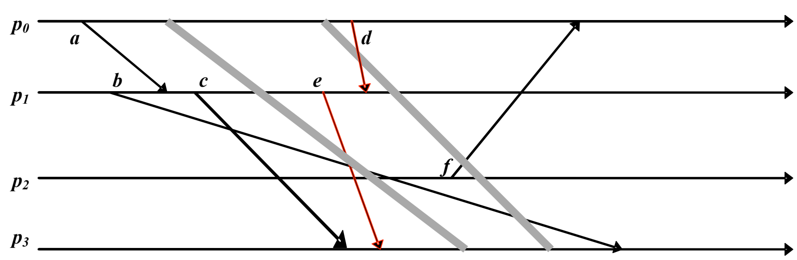
Consistent cuts and snapshots

- Identify system states that “might” have occurred in real-life
  - Avoid capturing “inconsistent” states
    - Receive without a send
  - This is the problem with the gray cuts

45

Temporal distortions

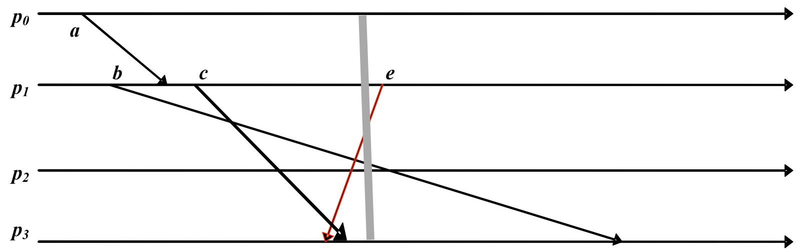
- Red messages cross gray cuts “backwards”



46

Temporal distortions

- Red messages cross gray cuts “backwards”



- In a nutshell: the cut includes a message that “was never sent”

47

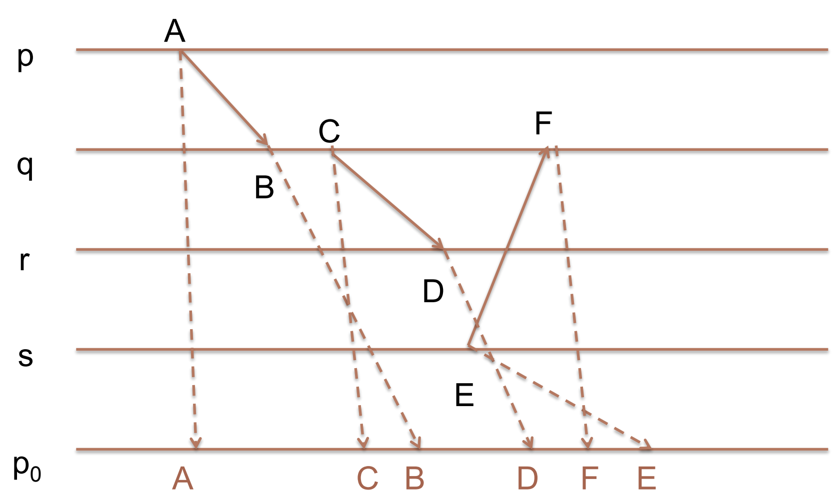
DISTRIBUTED LOGGING

48

Distributed Logging

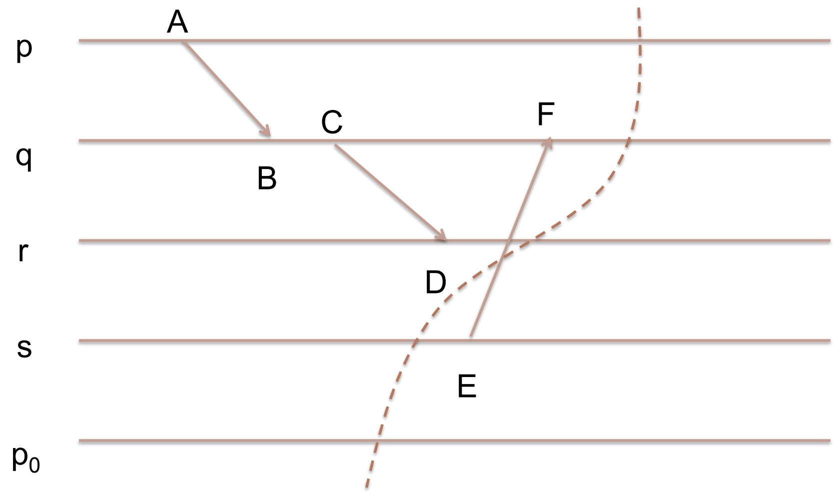
- We have n processes $p_1,\ldots,p_n$
- We want to use a monitor process p_0 to build a trace of the system for debugging purposes
- Protocol: every time an event e happens at a process p_i, it sends a notification of that event to p_0

49



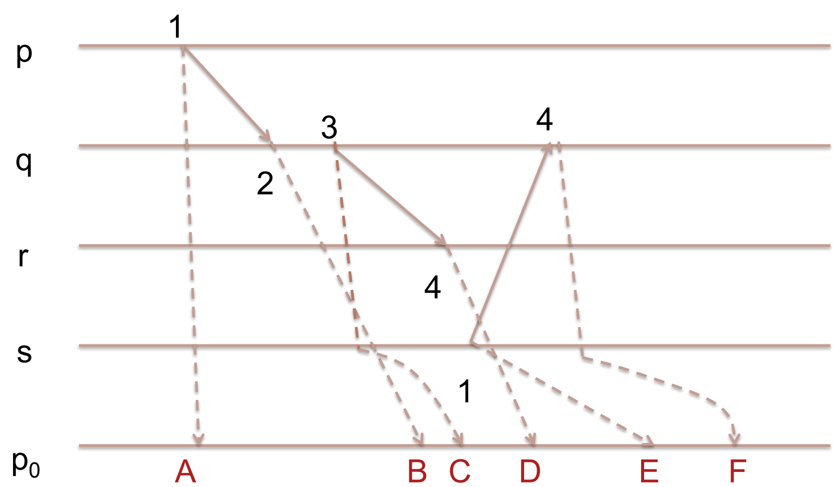
50

Inconsistent Cut



51

Delay Delivery for Consistency



52

Clock Condition

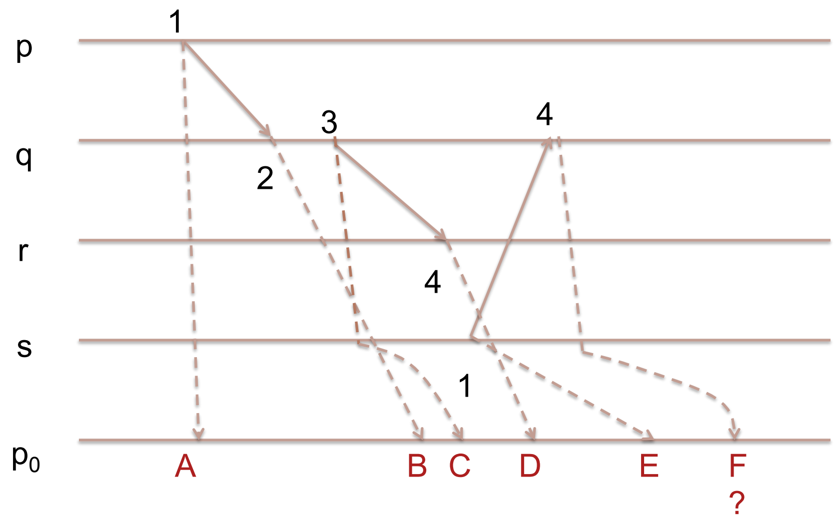
- **Clock Condition:**
  - $e \rightarrow e'$ implies $LT(e) < LT(e')$
- **Delivery Rule 1 (DR1):**

  At time t, deliver all received messages with timestamps up to t, in increasing timestamp order
- Clock condition ensures consistent observations

Gap Detection

- We cannot deliver a message m with $TS(m) = t$ unless we are certain that no message m' with $TS(m') < t$ can be received
- **Gap Detection:**
  - Given two events e and e'
  - Given $LT(e) < LT(e')$
  - Determine whether an event e'' exists such that $LT(e) < LT(e'') < LT(e')$

Safe to Deliver?

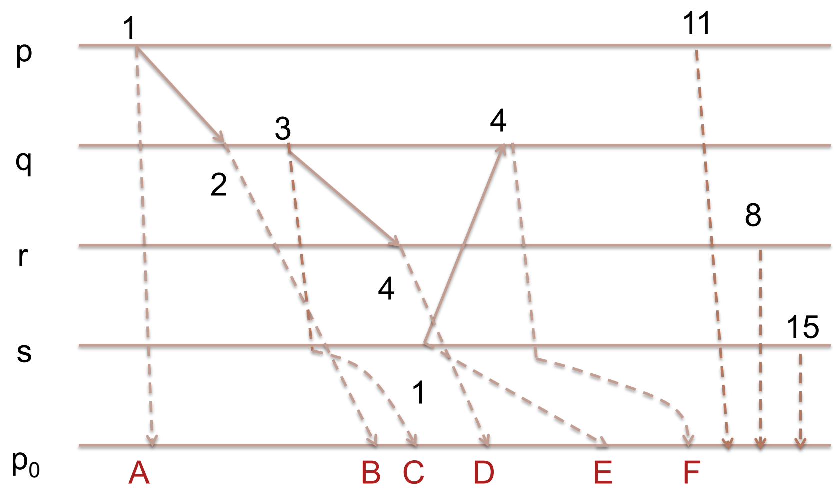


55

Stable Messages

- Message m received at p is **stable** if no future messages with smaller timestamps will be received at p
- **Delivery Rule 2 (DR2):** Deliver all stable messages at p0 in increasing time-stamp order
- With *FIFO* channels, stability is assured once messages with greater timestamps are received from every other process

56



Problem

- Delivery Rule 2 is too conservative
  - We have to see later messages from every other process before delivering a message from p

VECTOR TIME

59

Ordering Relations

- Ordered set (A, ≤)
- **Total order:** order is
  - *Total*: for all x,y, either x≤y or y≤x
  - *Symmetric*: x≤y and y≤x implies x=y
  - *Transitive*: x ≤y and y≤z implies x≤z
- **Partial order:** weaken totality to *reflexivity*: x≤x for all x
- **Preorder:** ordering relation is transitive, not refl
  - x<y and y<z implies x<z

60

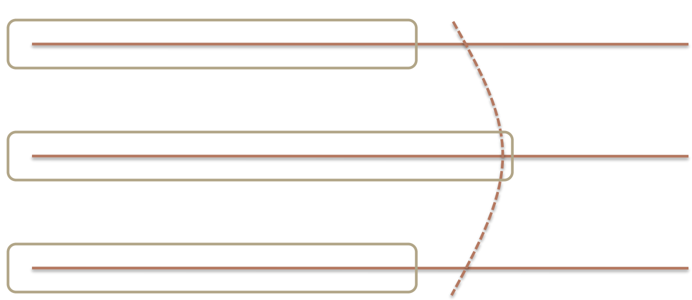
Potential Causality

- If A happens before B, A→B, then LT(A)<LT(B)
- But converse might not be true:
  - If LT(A)<LT(B) can't be sure that A→B
  - Total order placed on what is a partial order

61

Vector Clocks

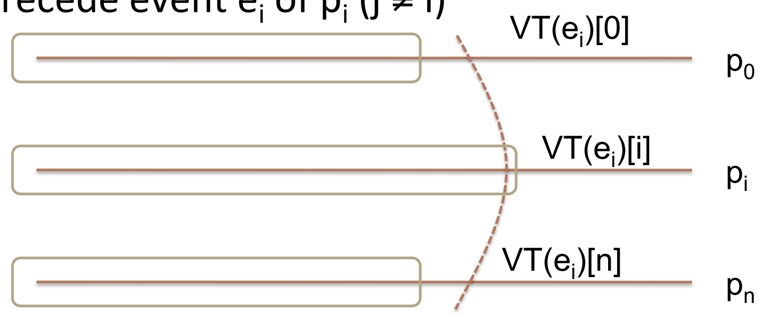
- Here we treat timestamps as a list
  - One counter for each process
  - Vector of n counters represents a "cut" of the executions of n processes



62

Operational Interpretation

- $VT(e_i)[i]$ = number of events p_i has executed up to and including e_i
- $VT(e_i)[j]$ = number of events of p_j that causally precede event e_i of p_i ($j \neq i$)

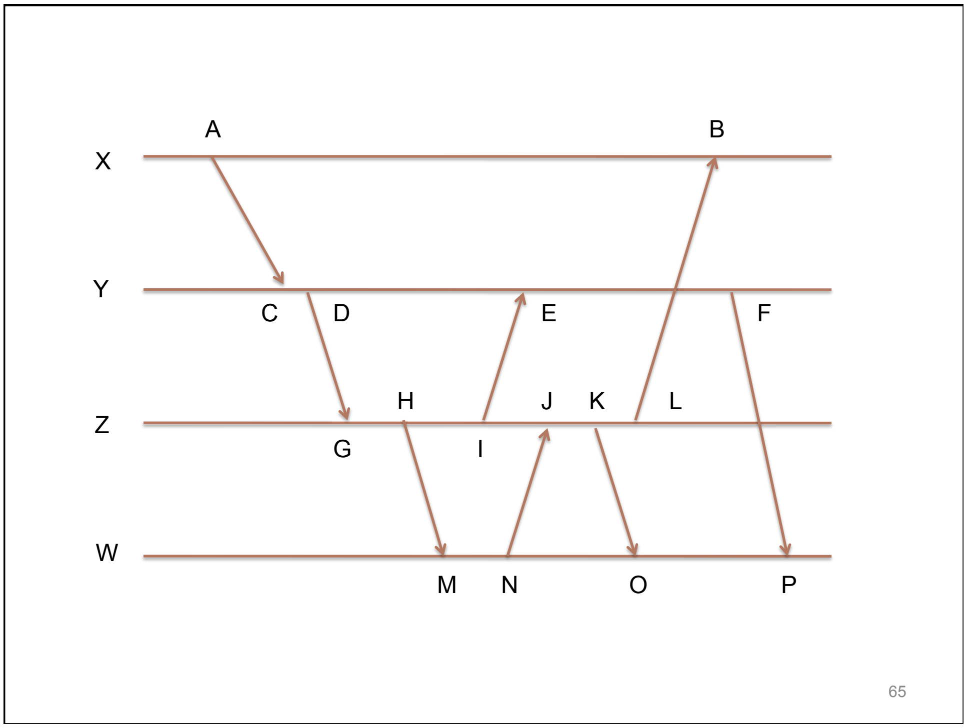


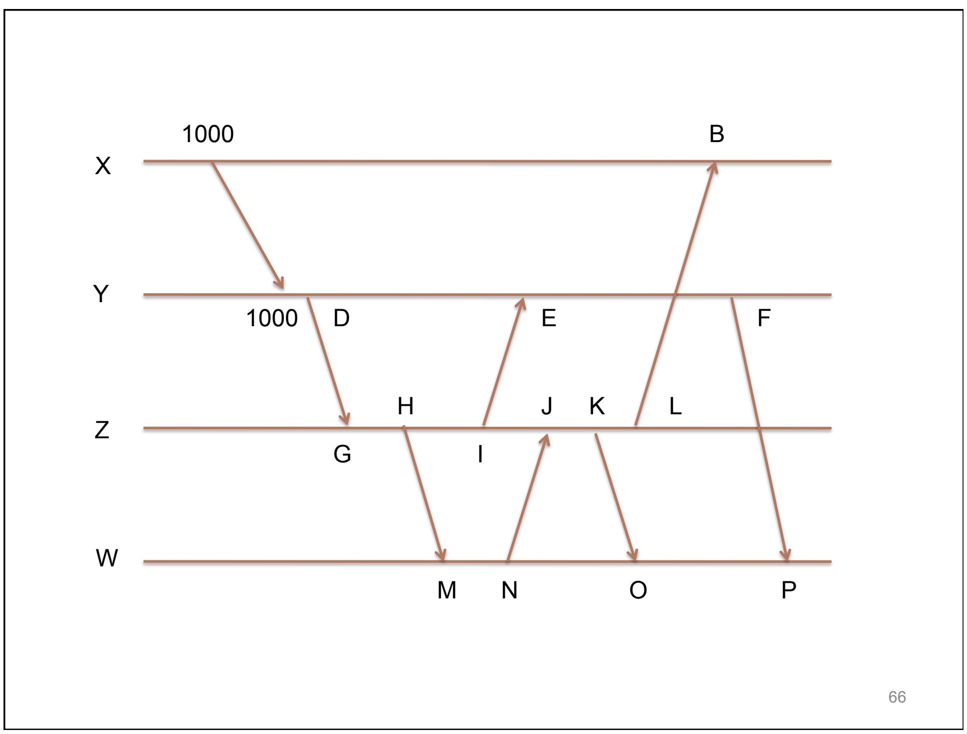
63

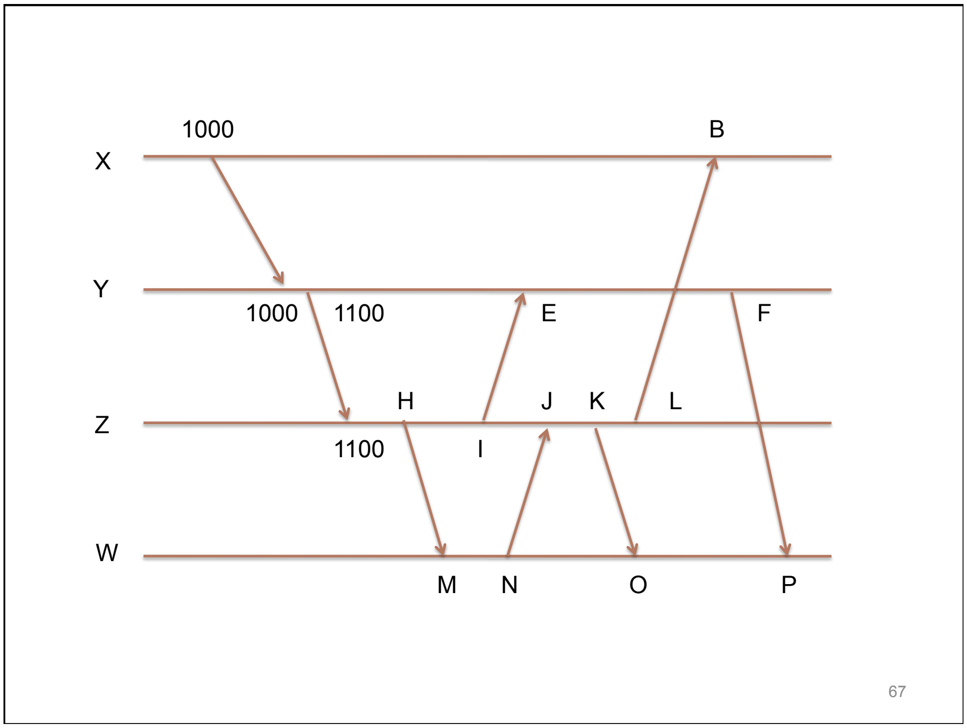
Vector Clocks

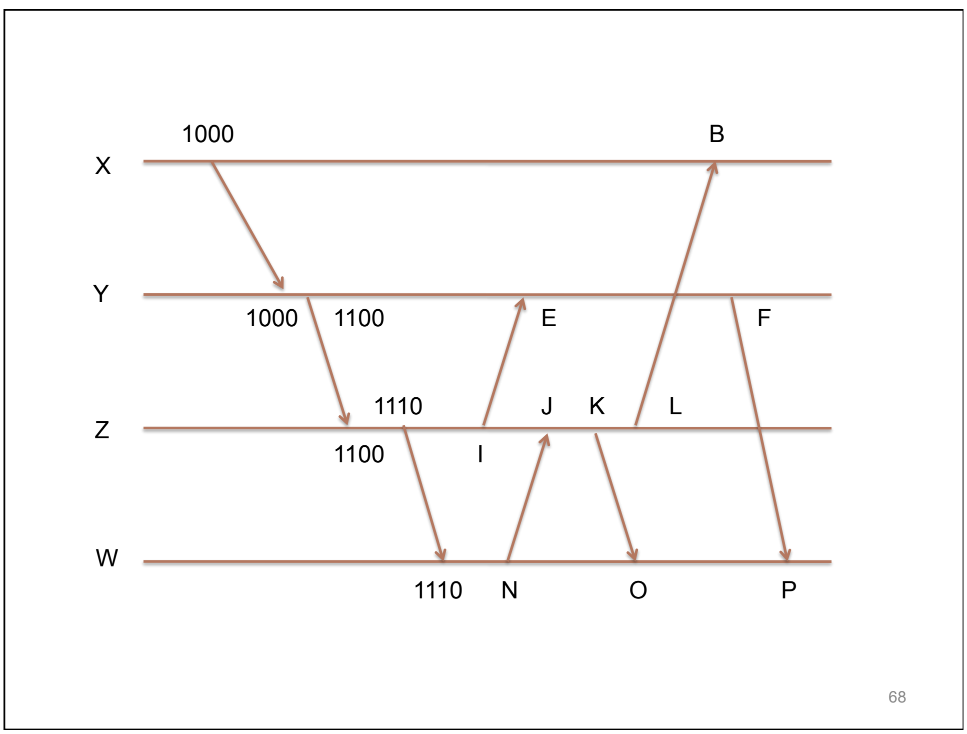
- Rules for managing vector clock
  - When event happens at p, increment $VT_p[index_p]$
    - Normally, also increment for snd and rcv events
  - When sending a message, set $TS(m)=VT_p$
  - When receiving, set $VT_q=max(VT_q, TS(m))$

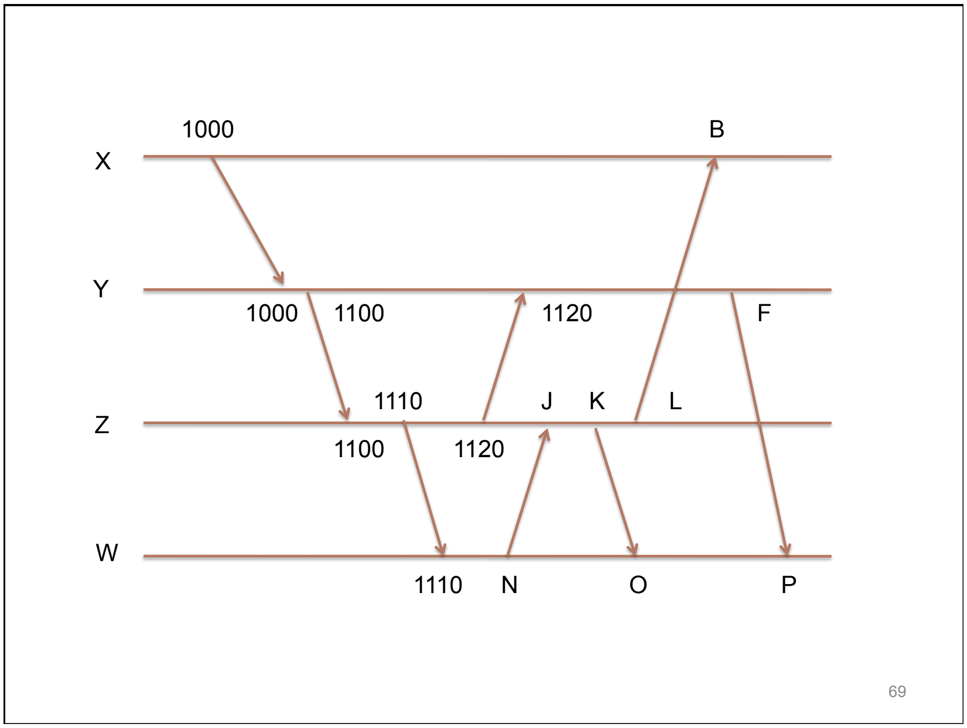
64

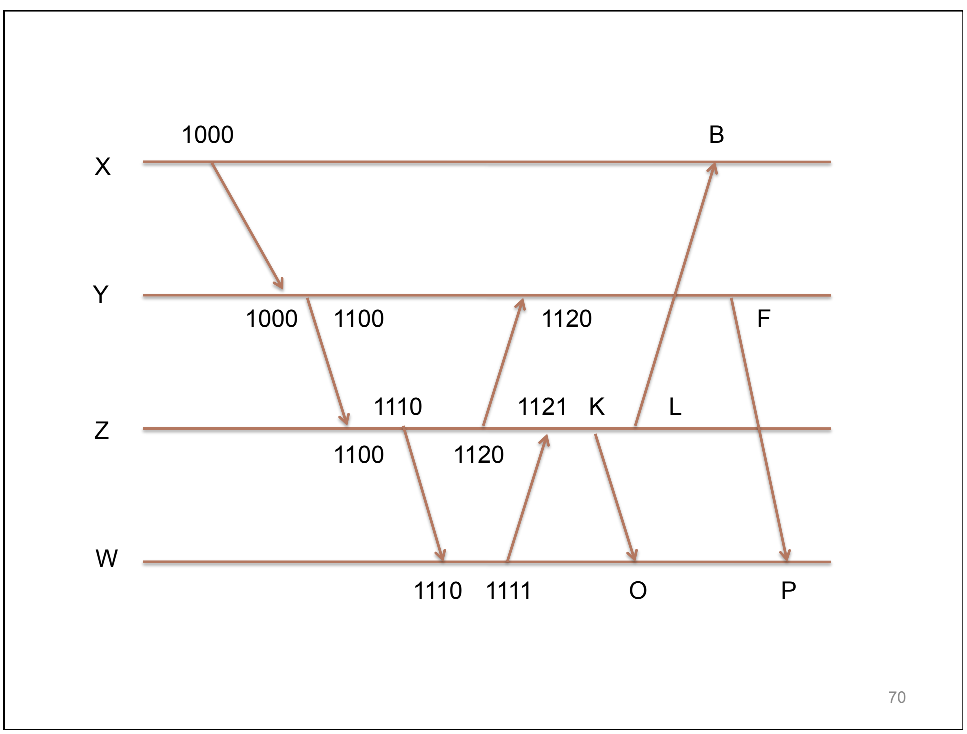

A
B
X
Y
C
D
E
F
H
J
K
L
Z
G
I
W
M
N
O
P
65

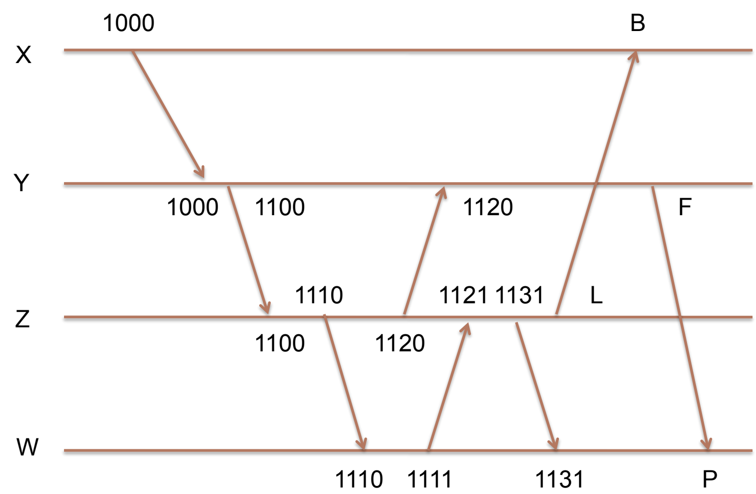

1000
B
X
Y
1000
D
E
F
H
J
K
L
Z
G
I
W
M
N
O
P
66


1000
B
X
Y
1000
1100
E
F
H
J
K
L
Z
1100
I
W
M
N
O
P
67

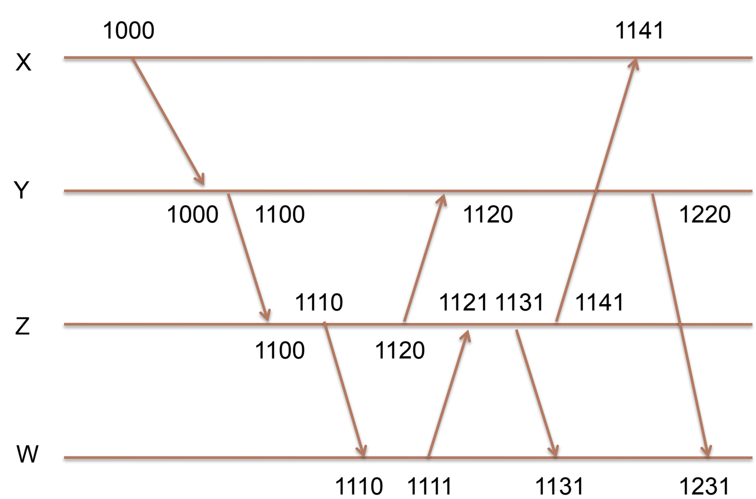

1000
B
X
Y
1000
1100
E
F
1110
J
K
L
Z
1100
I
W
1110
N
O
P
68


1000
B
X
Y
1000
1100
1120
F
1110
J
K
L
Z
1100
1120
W
1110
N
O
P
69


1000
B
X
Y
1000
1100
1120
F
1110
1121
K
L
Z
1100
1120
W
1110
1111
O
P
70


1000
B
X
Y
1000
1100
1120
F
1110
1121 1131
L
Z
1100
1120
W
1110
1111
1131
P

71


1000
1141
X
Y
1000
1100
1120
1220
1110
1121 1131
1141
Z
1100
1120
W
1110
1111
1131
1231

72

Rules for comparison of VTs

- We'll say that $VT_A \leq VT_B$ if
  - $VT_A[i] \leq VT_B[i]$ for all i
- And we'll say that $VT_A < VT_B$ if
  - $VT_A \leq VT_B$ but $VT_A \neq VT_B$
  - That is, for some i, $VT_A[i] < VT_B[i]$
- Examples?
  - [2,4] ≤ [2,4]
  - [1,3] < [7,3]
  - [1,3] is "incomparable" to [3,1]

73

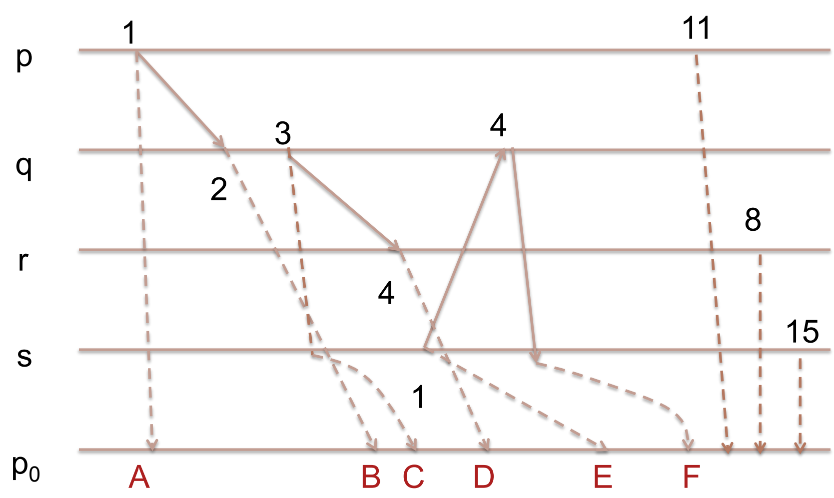
Vector time and happens before

- If A→B, then $VT_A < VT_B$
  - Write a chain of events from A to B
  - Step by step the vector clocks get larger
- If $VT_A < VT_B$ then A→B
  - Two cases: if A and B both happen at same process p, trivial
  - If A happens at p and B at q, can trace the path back by which q "learned" $VT_A[p]$
- Otherwise A and B happened concurrently

74

DISTRIBUTED LOGGING AND VECTOR TIME

75



76

Clock Condition

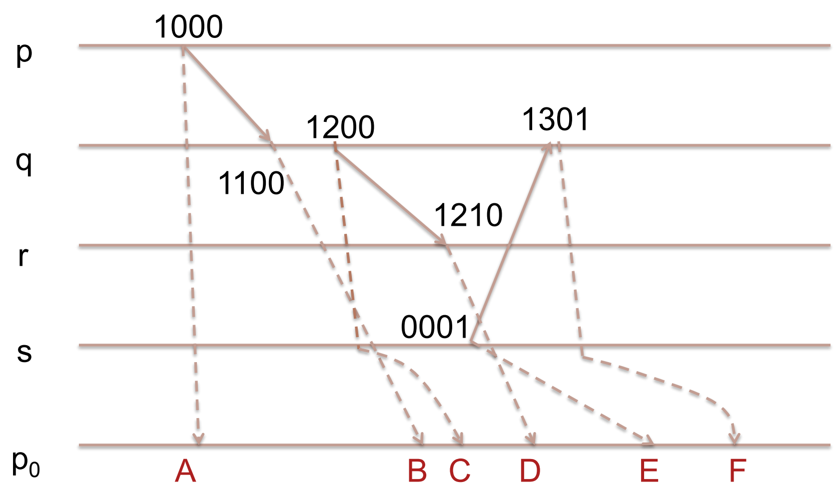
- Delivery Rule 2 too conservative
- Clock Condition:
  - e → e’ implies LT(e) < LT(e’)
  - Possible: LT(e) < LT(e’), but not (e → e’)
- Logical clocks give potential causality
  - Hence the need to wait for stability

77

Strong Clock Condition

- **Delivery Rule 3 (DR3):**
  - Deliver messages all of whose causal predecessors have been delivered
- Relies on: **Strong Clock Condition**
  - e → e’ if and only if VT(e) < VT(e’)

78



F can be delivered once we have received
1 message from p, 2 from q, one from s

79

Causal Delivery

- We have used causal delivery at monitor process to construct consistent observations
- Causal delivery may also be used in general message delivery
- Deadlock problems with point-to-point (requires matrix clock)
- Vector clock used with causal broadcast

80

ORDERED MESSAGE DELIVERY

81

FIFO Order

- FIFO Delivery Rule:
  - send(m) on **i** → send(m’) on **i**

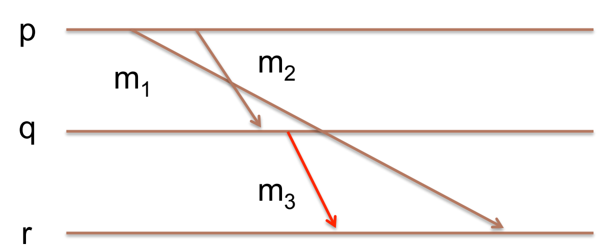
    implies that

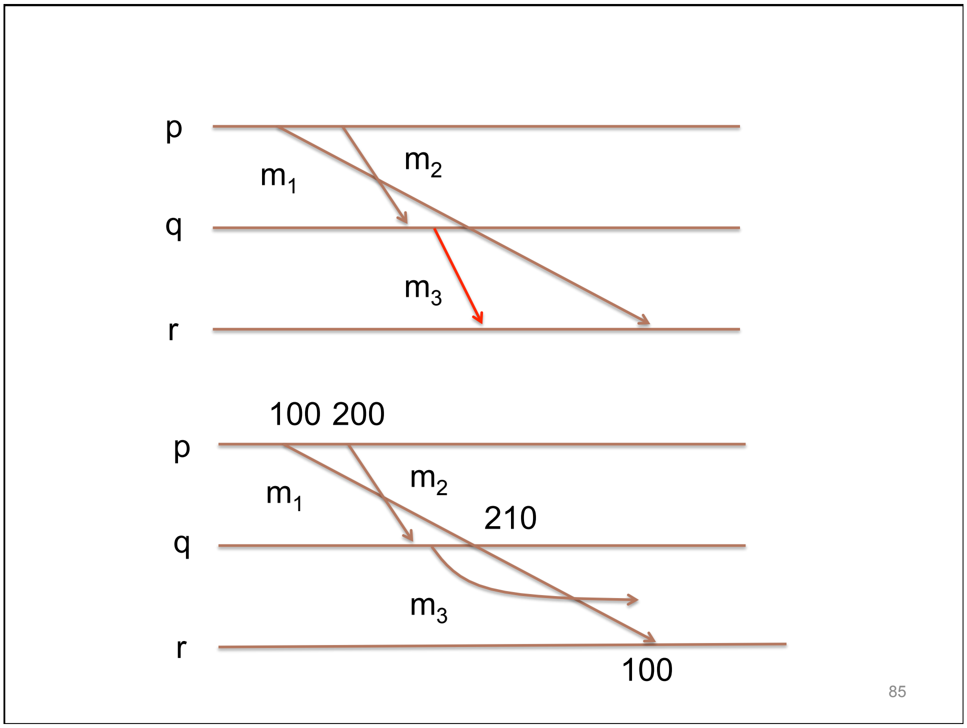
    deliver(m) on **k** → deliver(m’) on **k**

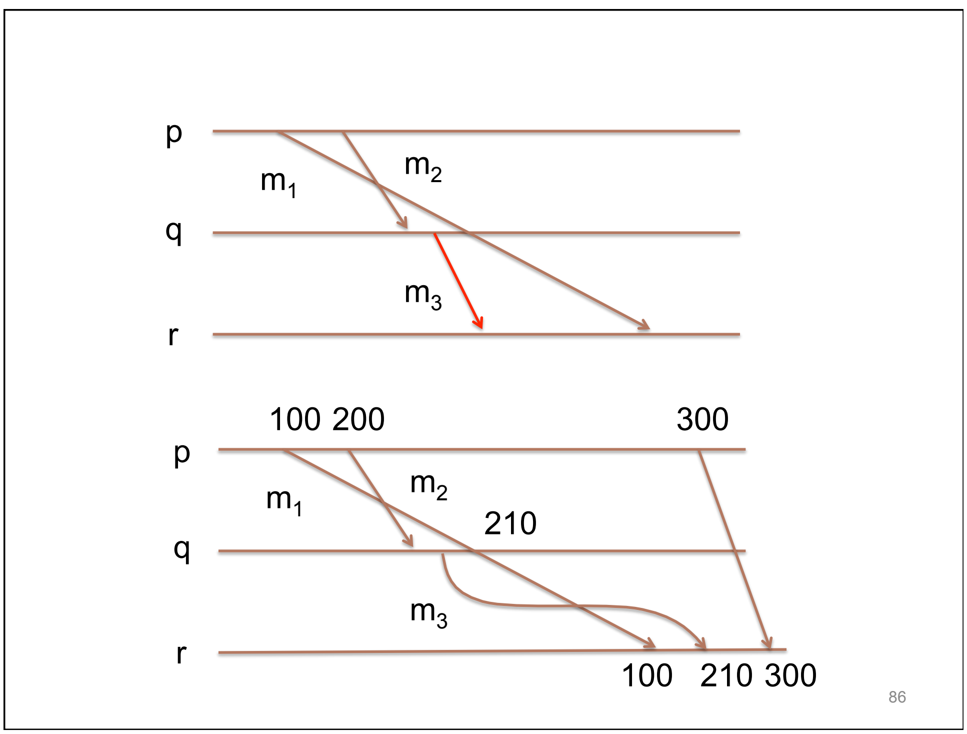
82

Causal Order

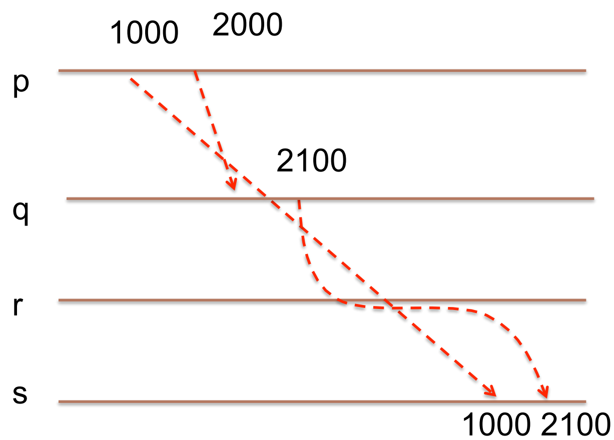
- Causal Delivery Rule:
  - send(m) on **i** → send(m') on **j**
    implies that
    deliver(m) on **k** → deliver(m') on **k**
- FIFO Order not sufficient to ensure Causal Order




p
q
r
m_1
m_2
m_3
100 200
210
100
85

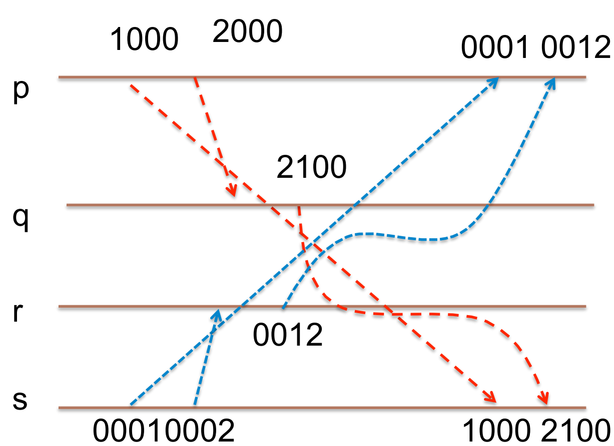

p
q
r
m_1
m_2
m_3
100 200
300
210
100 210 300
86

Causal Point-to-Point May Deadlock



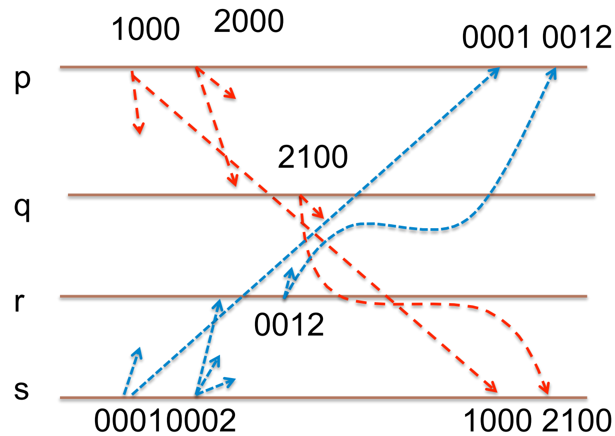
87

Causal Point-to-Point May Deadlock



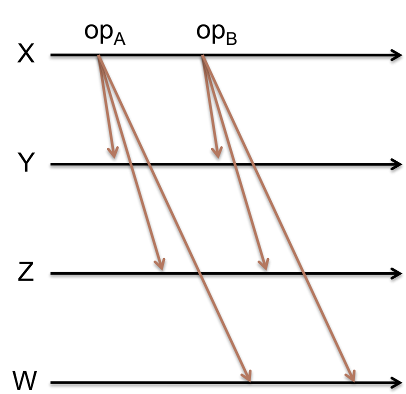
88

Causal Multicast avoids Deadlock



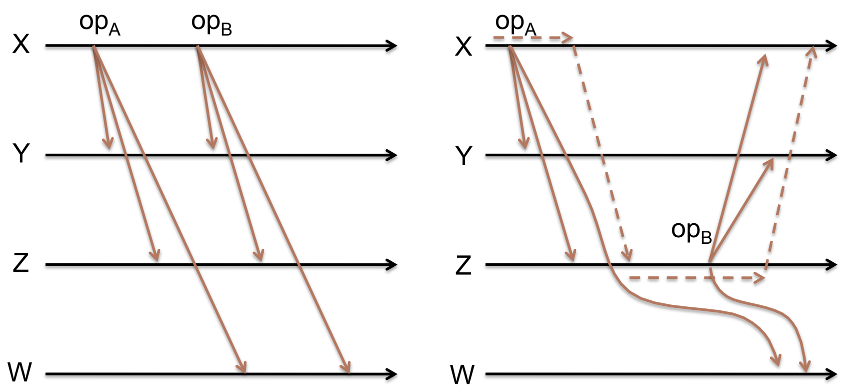
89

FIFO With One Sender



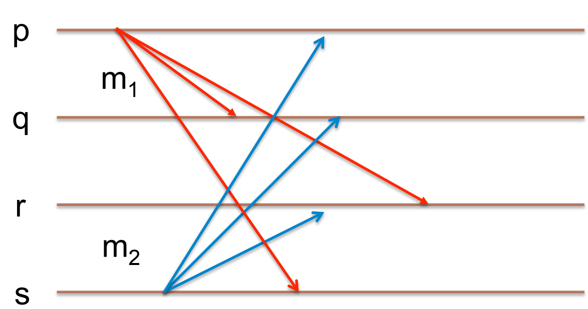
90

Causal with One Thread



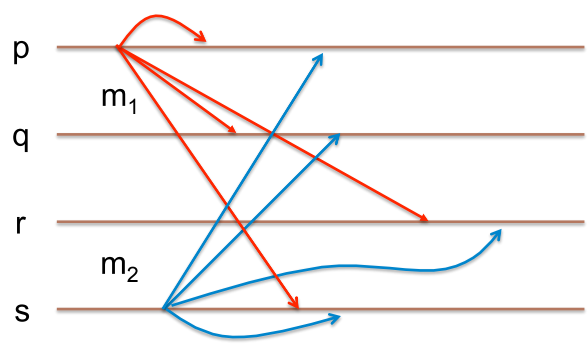
91

Causal but not Total



92

Total Ordered Multicast



93