Causal, Partial and Total Ordering of Messages in Distributed Systems

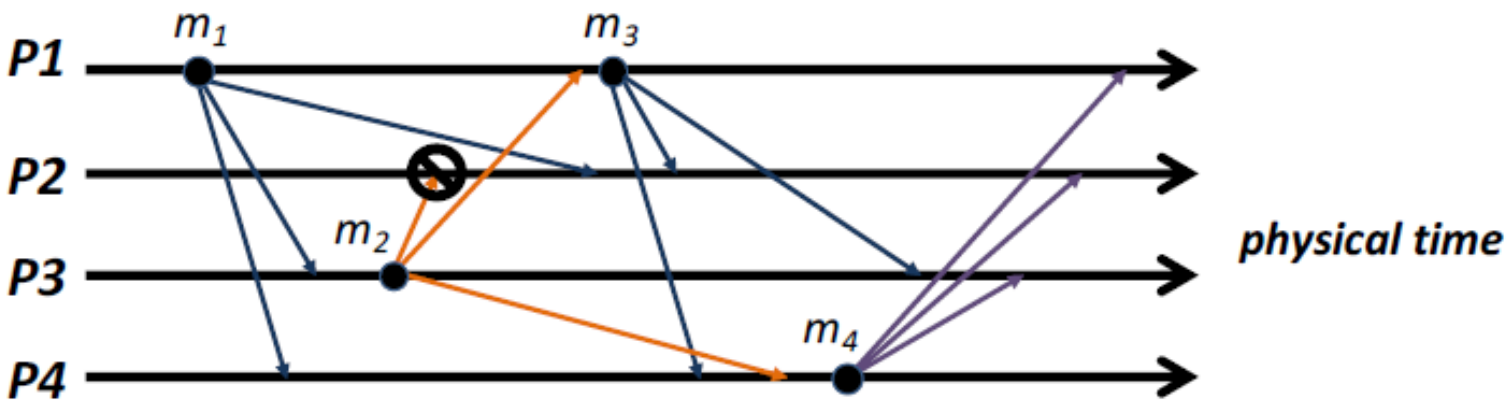
Causal and Total Order

Stronger orderings

- Can also implement FIFO ordering by just using a reliable FIFO transport like TCP/IP
- But the general 'receive versus deliver' model also allows us to provide **stronger** orderings:
  - **Causal ordering**: if event *multicast(**g**, m_1)* → *multicast(**g**, m_2)*, then all processes will see m_1 before m_2
  - **Total ordering**: if any processes delivers a message m_1 before m_2, then all processes will deliver m_1 before m_2
- Causal ordering implies FIFO ordering, since any two multicasts by the same process are related by →
- Total ordering (as defined) does *not* imply FIFO (or causal) ordering, just says that all processes must agree
  - Often want **FIFO-total** ordering (combines the two)

Causal Ordering

Causal ordering

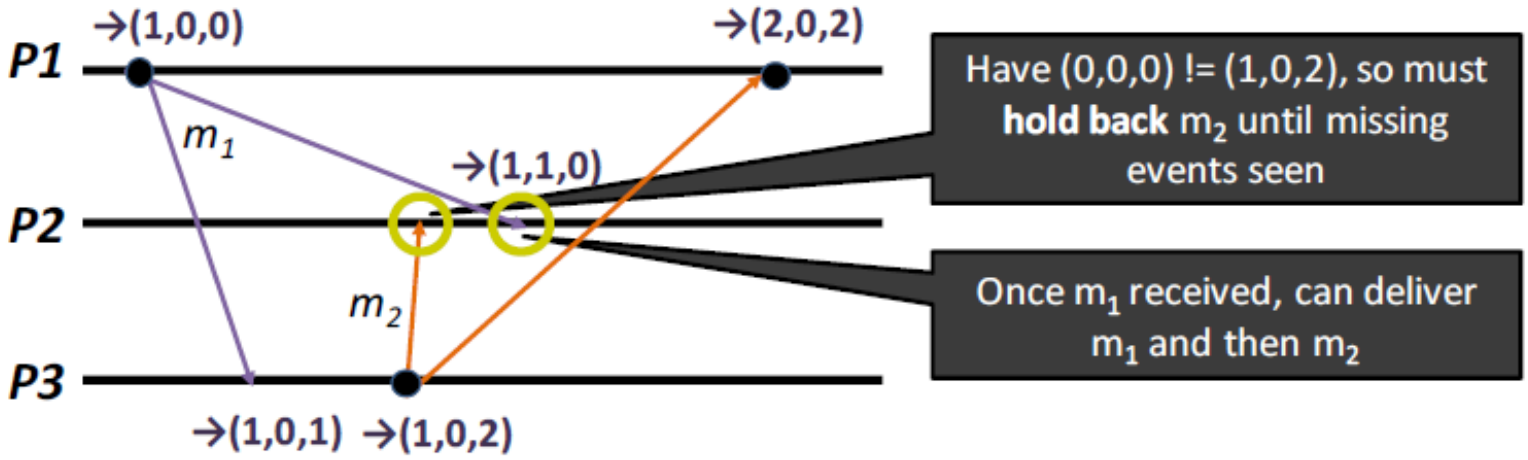


- Same example as previously, but now causal ordering means that
  (*a*) everyone must see $\mathbf{m_1}$ before $\mathbf{m_3}$ (as with FIFO), **and**
  (*b*) everyone must see $\mathbf{m_1}$ before $\mathbf{m_2}$ (due to happens-before)
- Is this ok?
  - No! $\mathbf{m_1} \rightarrow \mathbf{m_2}$, but **P2** sees $\mathbf{m_2}$ before $\mathbf{m_1}$
  - To be correct, must hold back (delay) delivery of $\mathbf{m_2}$ at **P2**
  - But how do we know this?

Causal Ordering

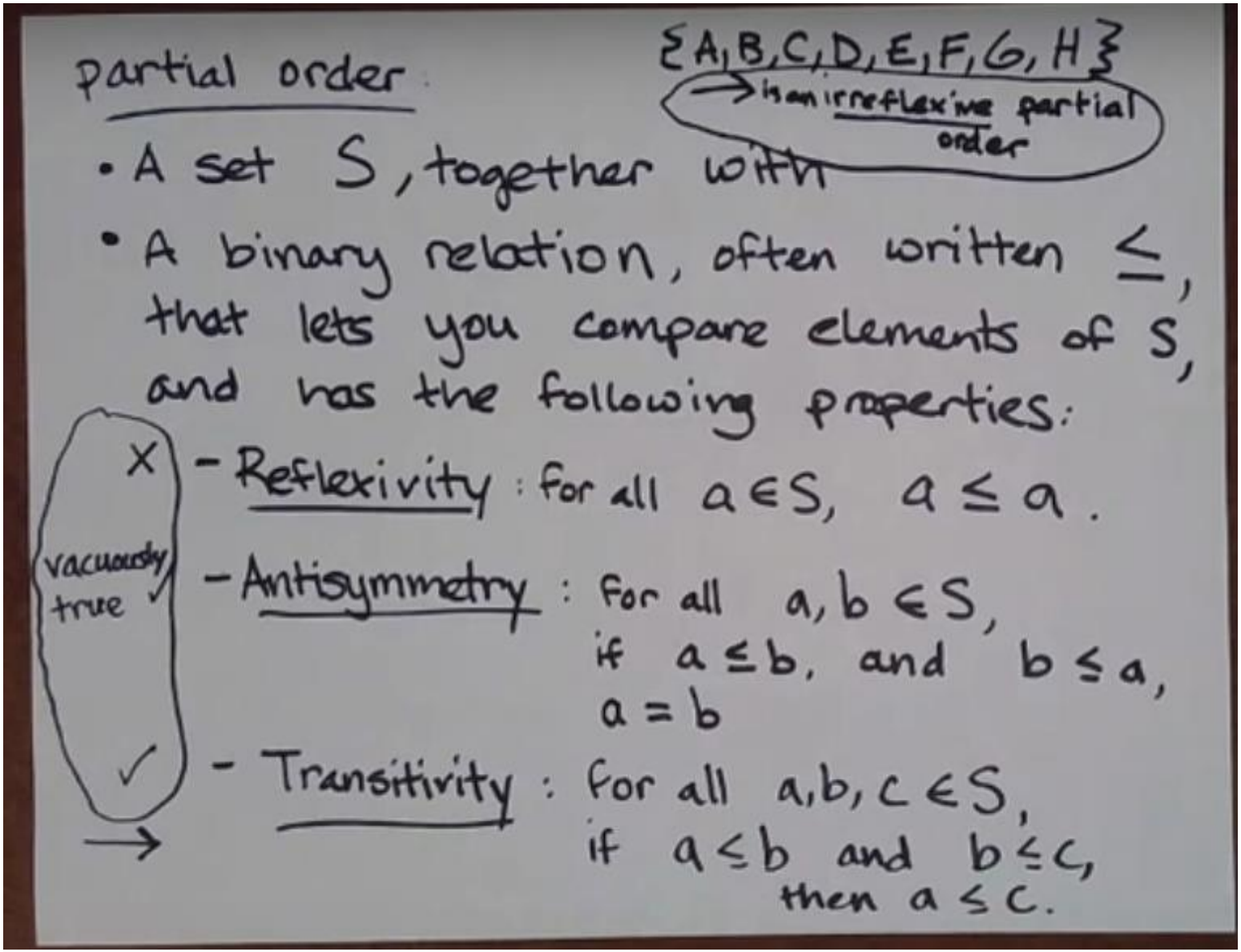
Implementing causal ordering

- Turns out this is pretty easy!
  - Start with receive algorithm for FIFO multicast…
  - and replace sequence numbers with vector clocks



- Some care needed with dynamic groups

Partial Order



Total Order

Total ordering of events

- A system of clocks that satisfy the Clock Condition can be used to totally order system events.
- To totally order the events in a system, the events are ordered according to their times of occurrence. In case two or more events occur at the same time, an arbitrary total ordering $\prec$ of processes is used. To do this, the relation $\Rightarrow$ is defined as follows:

  If a is an event in process P_i and b is an event in process P_j, then $a \Rightarrow b$ if and only if either:

  i. $C_i\langle a\rangle < C_j\langle b\rangle$ or

  ii. $C_i\langle a\rangle = C_j\langle b\rangle$ and $P_i \prec P_j$

  There is total ordering because for any two events in the system, it is clear which happened first.
- The total ordering of events is very useful for distributed system implementation.