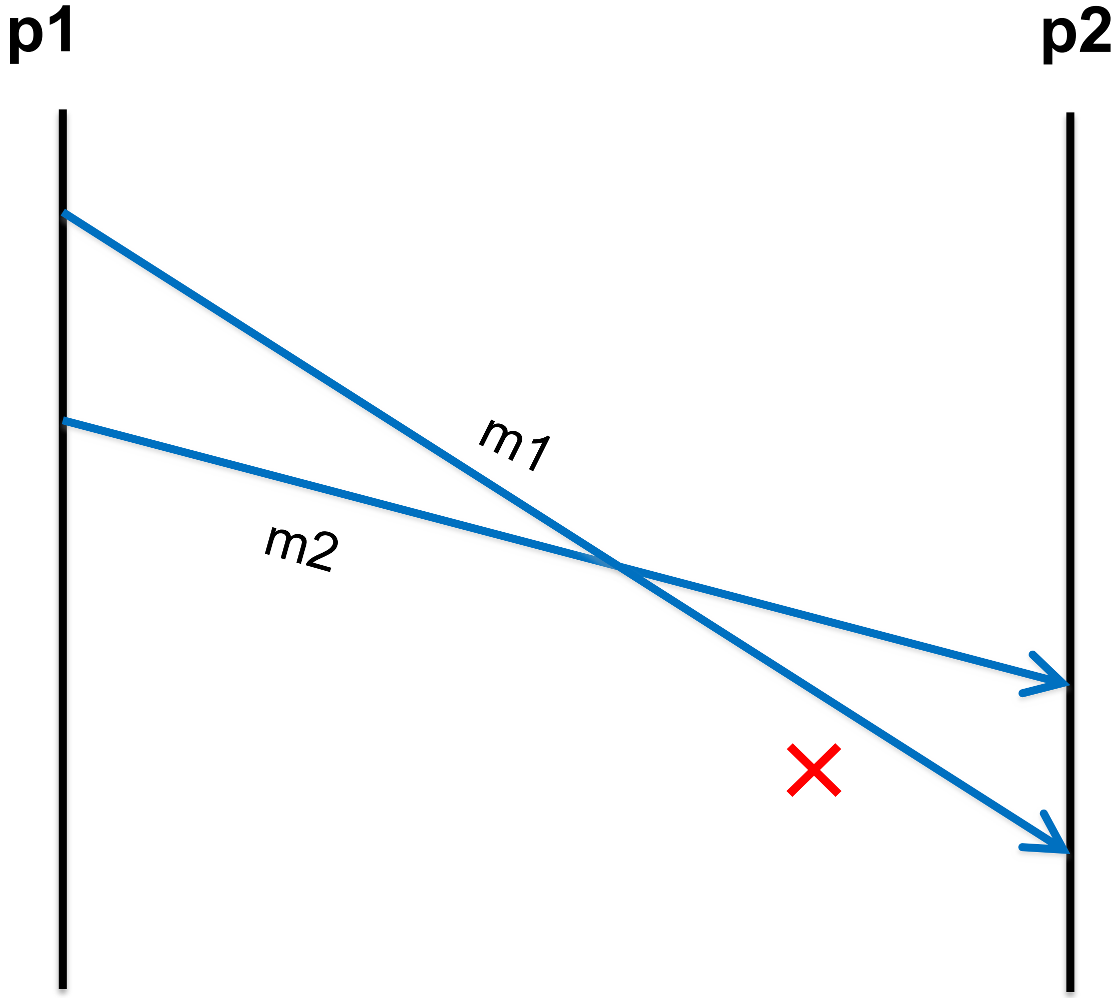
Fault Tolerance and Reliable Multicast

Jingzhu He

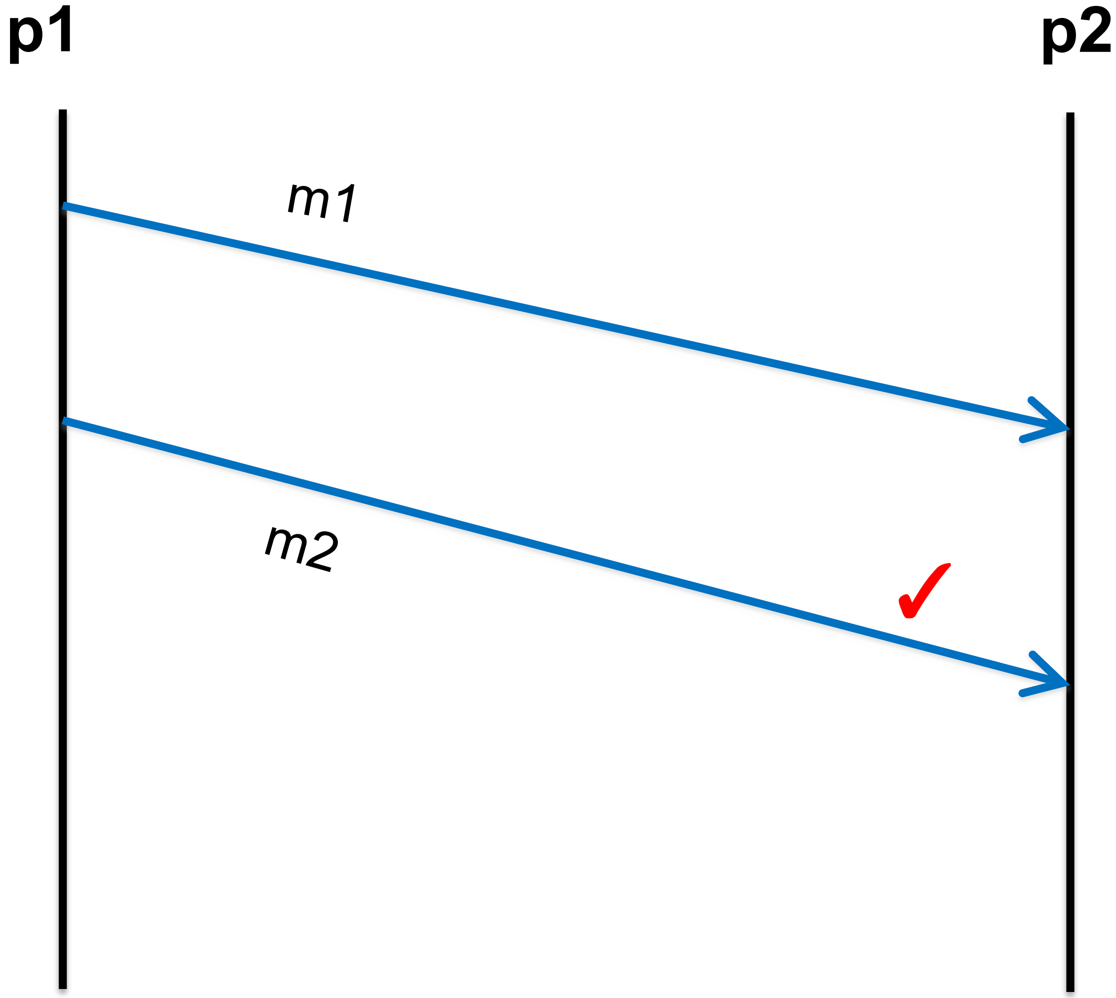
Three Delivery Properties

- FIFO ordering
  – If a process sends a message m2 after m1, any process delivering both deliver m1 first.
- Causal ordering
  – If m1 is sent **happen-before** m2 is sent, then m1's delivery must **happen-before** m2's delivery.
- Total ordering
  – If a process delivers message m1 before m2, then all processes delivering both m1 and m2 deliver m1 before m2.

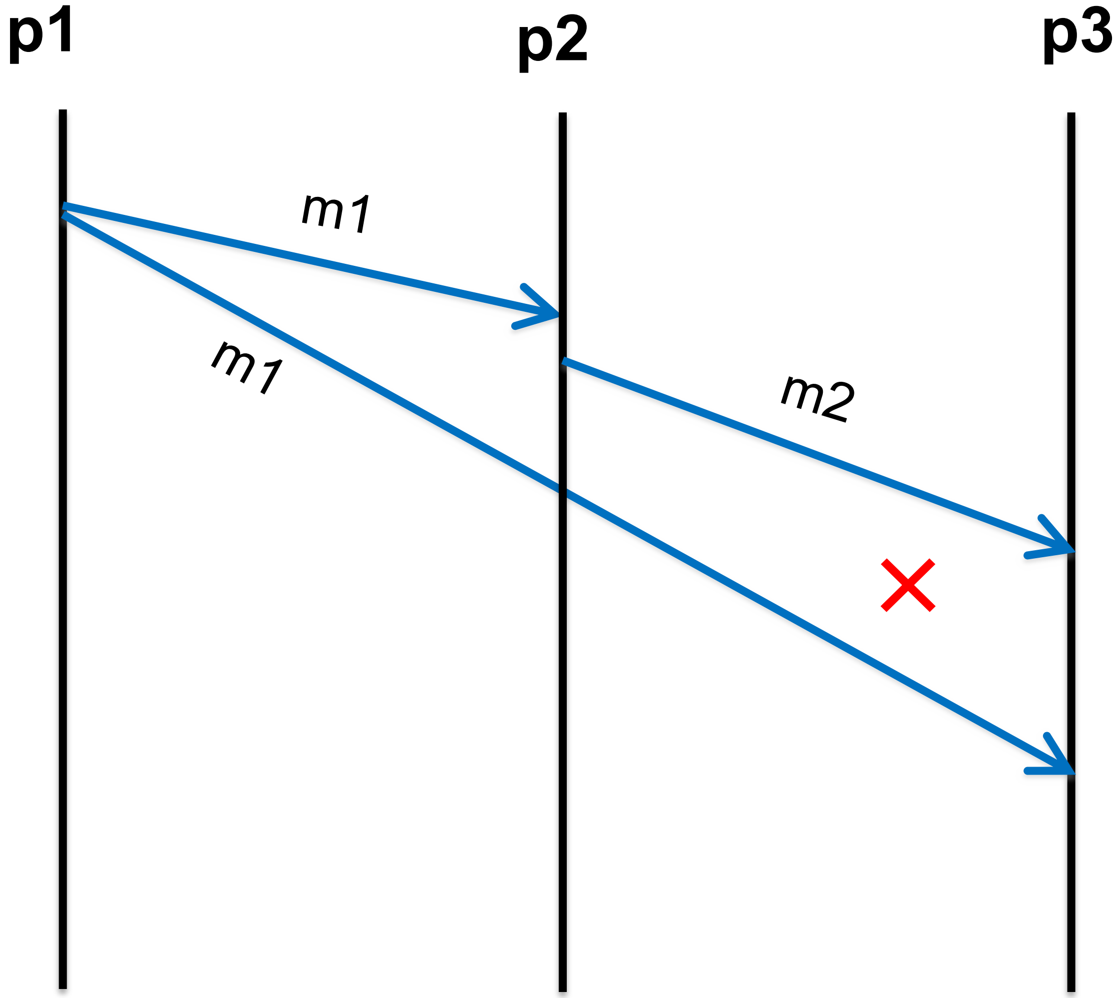
FIFO Delivery



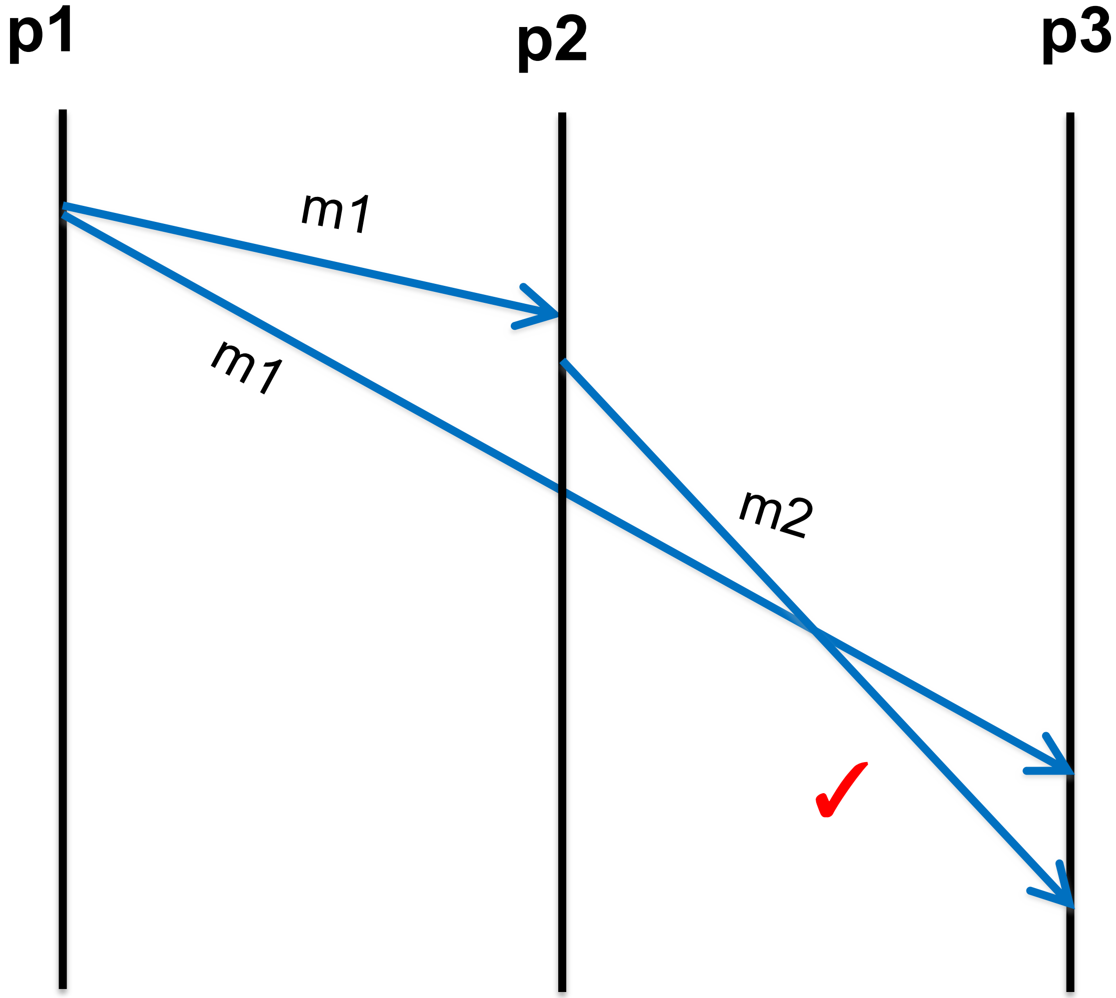
FIFO Delivery



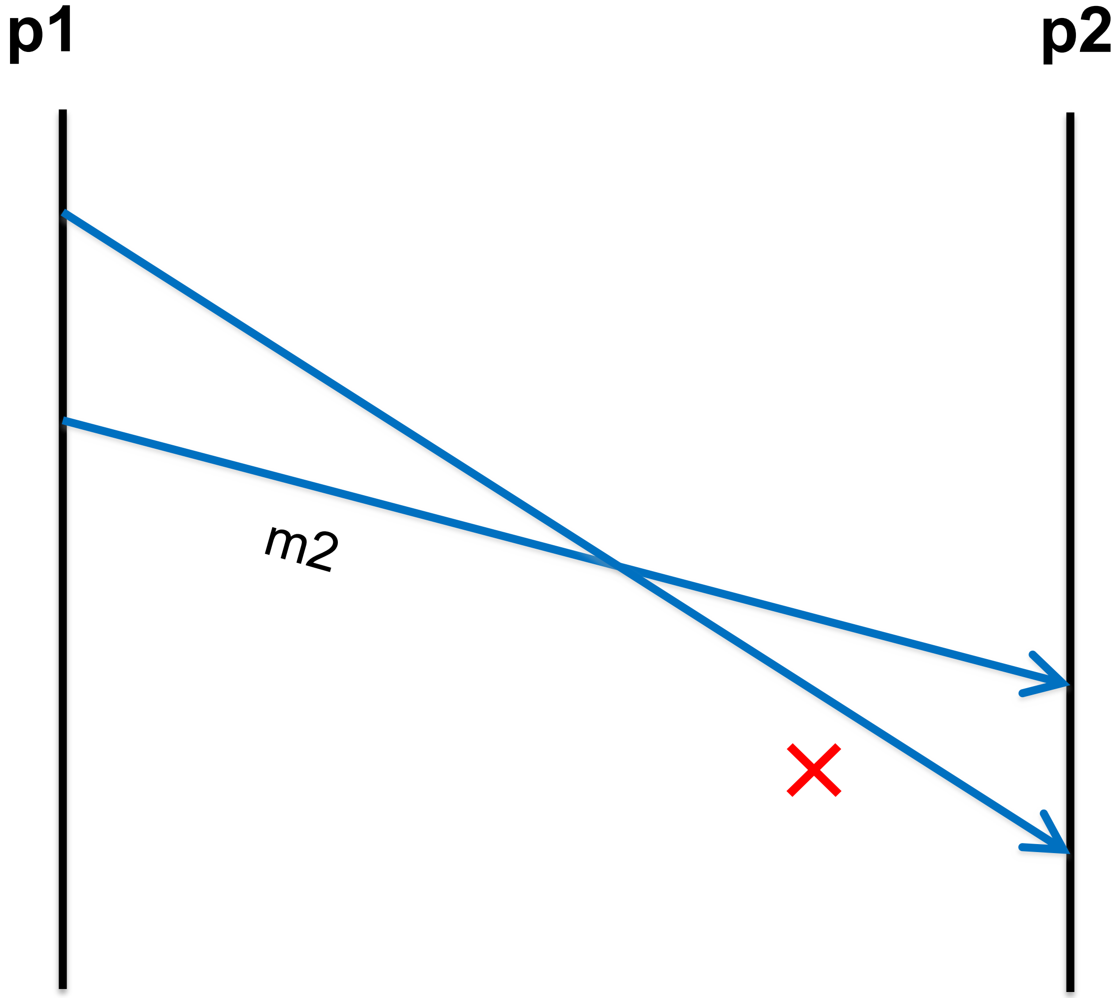
Causal Delivery



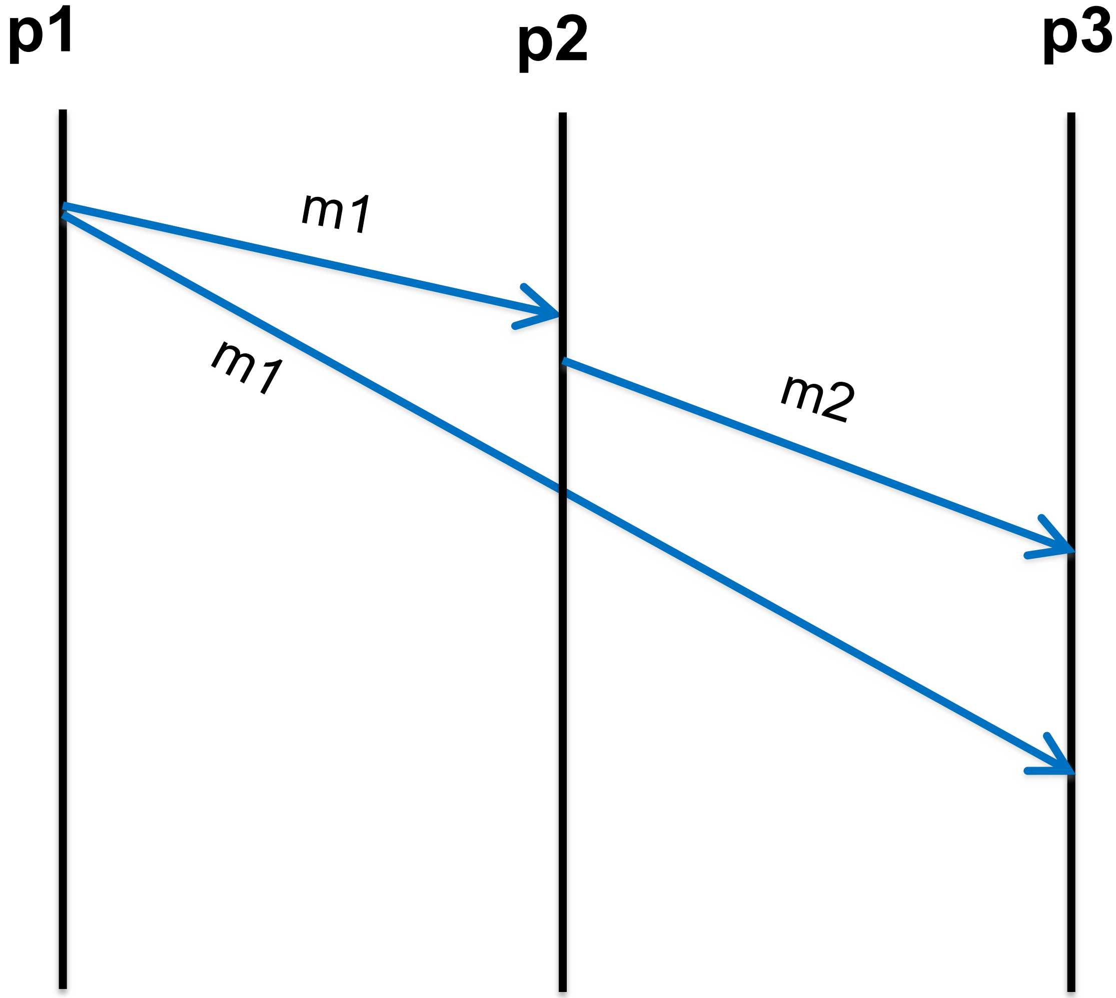
Causal Delivery



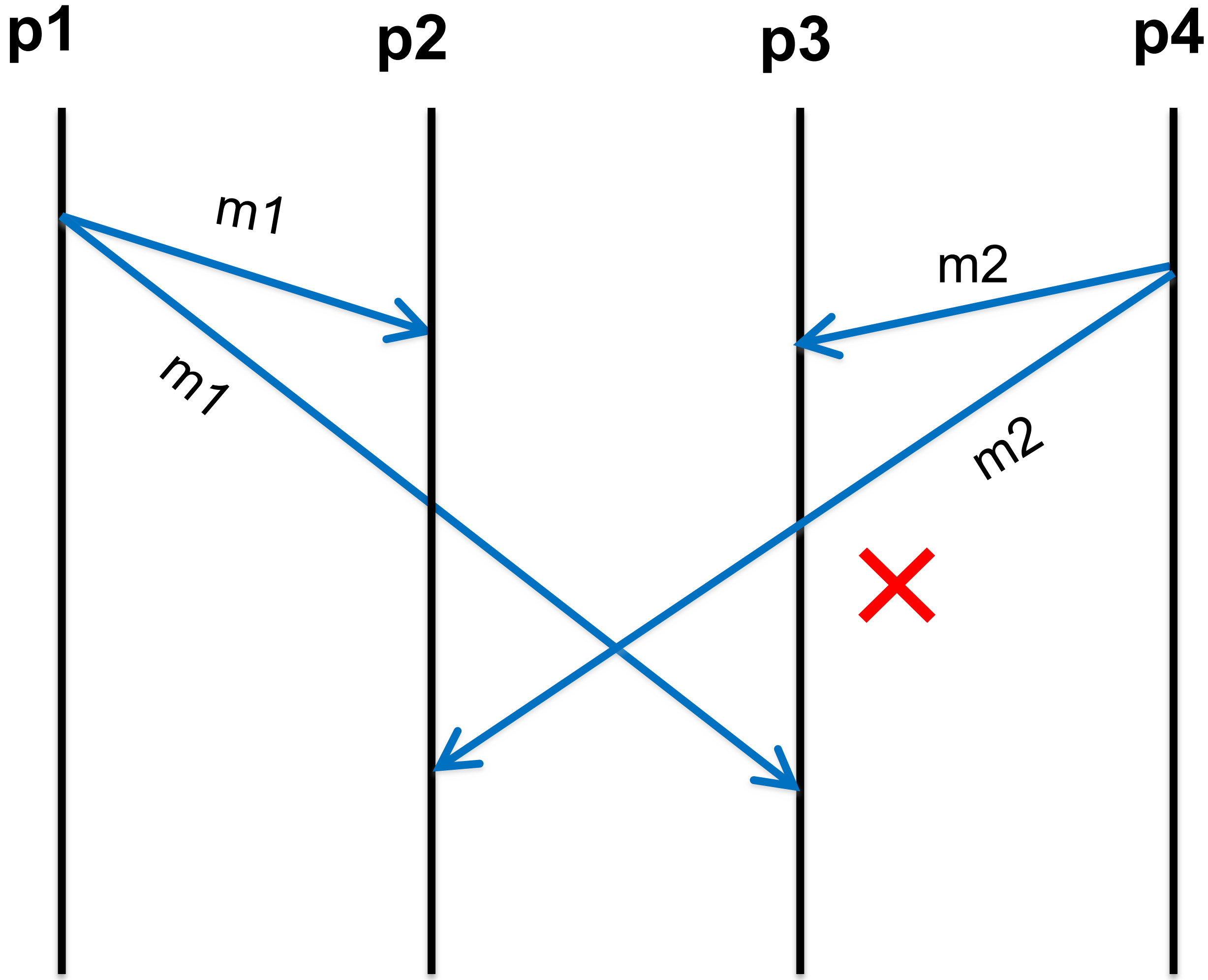
Is FIFO Delivery Violation Also Causal Delivery Violation?



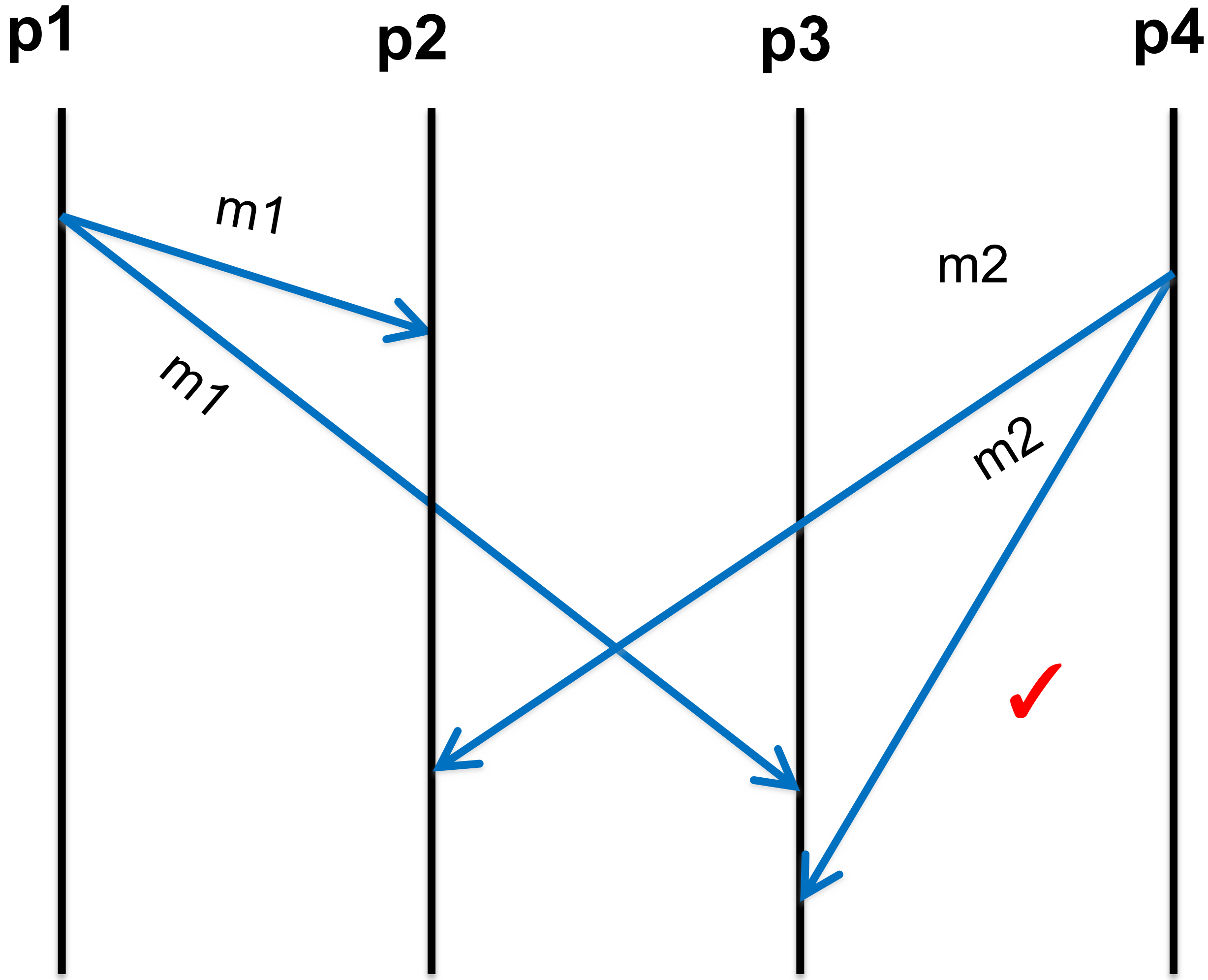
Relationship Between FIFO Delivery and Causal Delivery



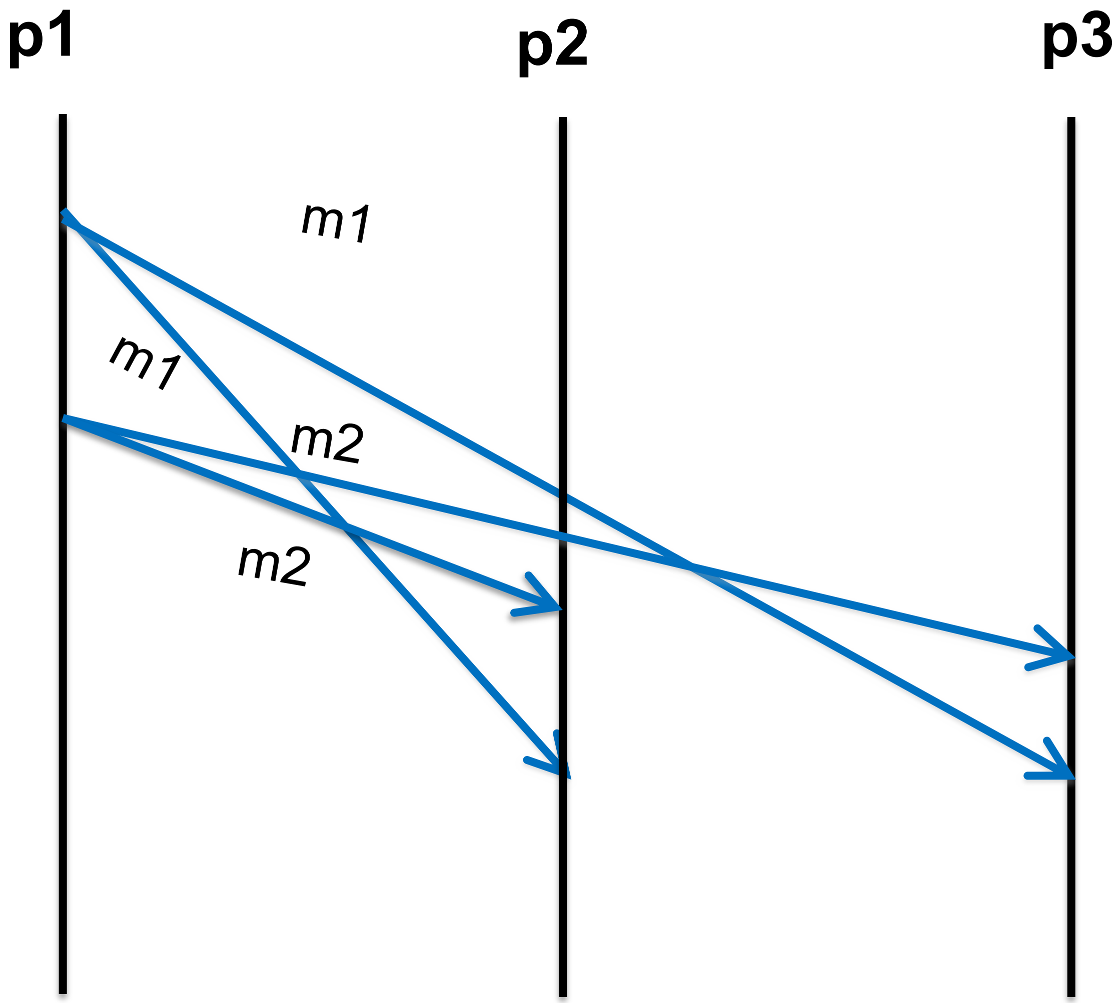
Totally-ordered Delivery



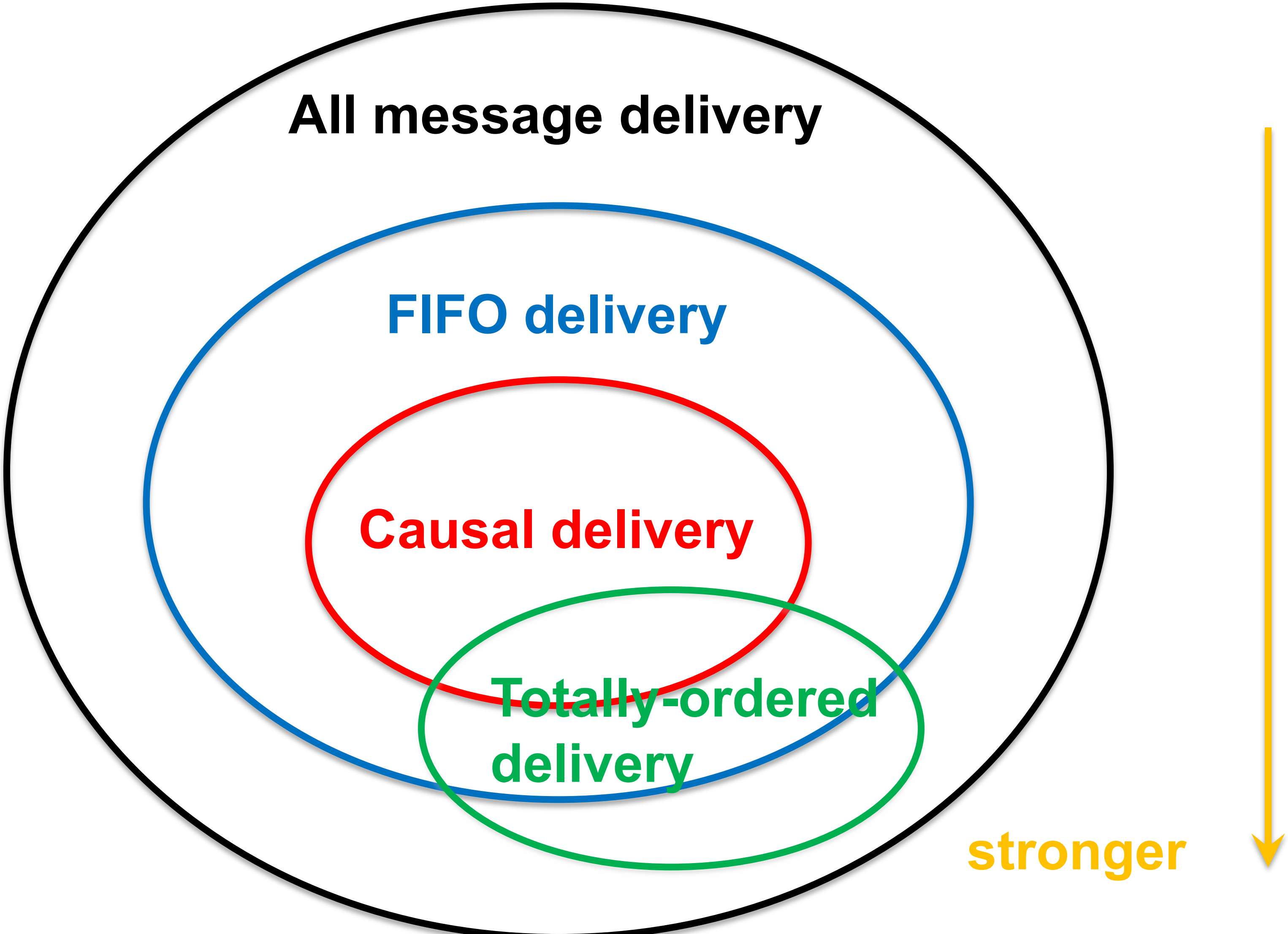
Totally-ordered Delivery



Does Totally-ordered Delivery Comply FIFO Delivery?



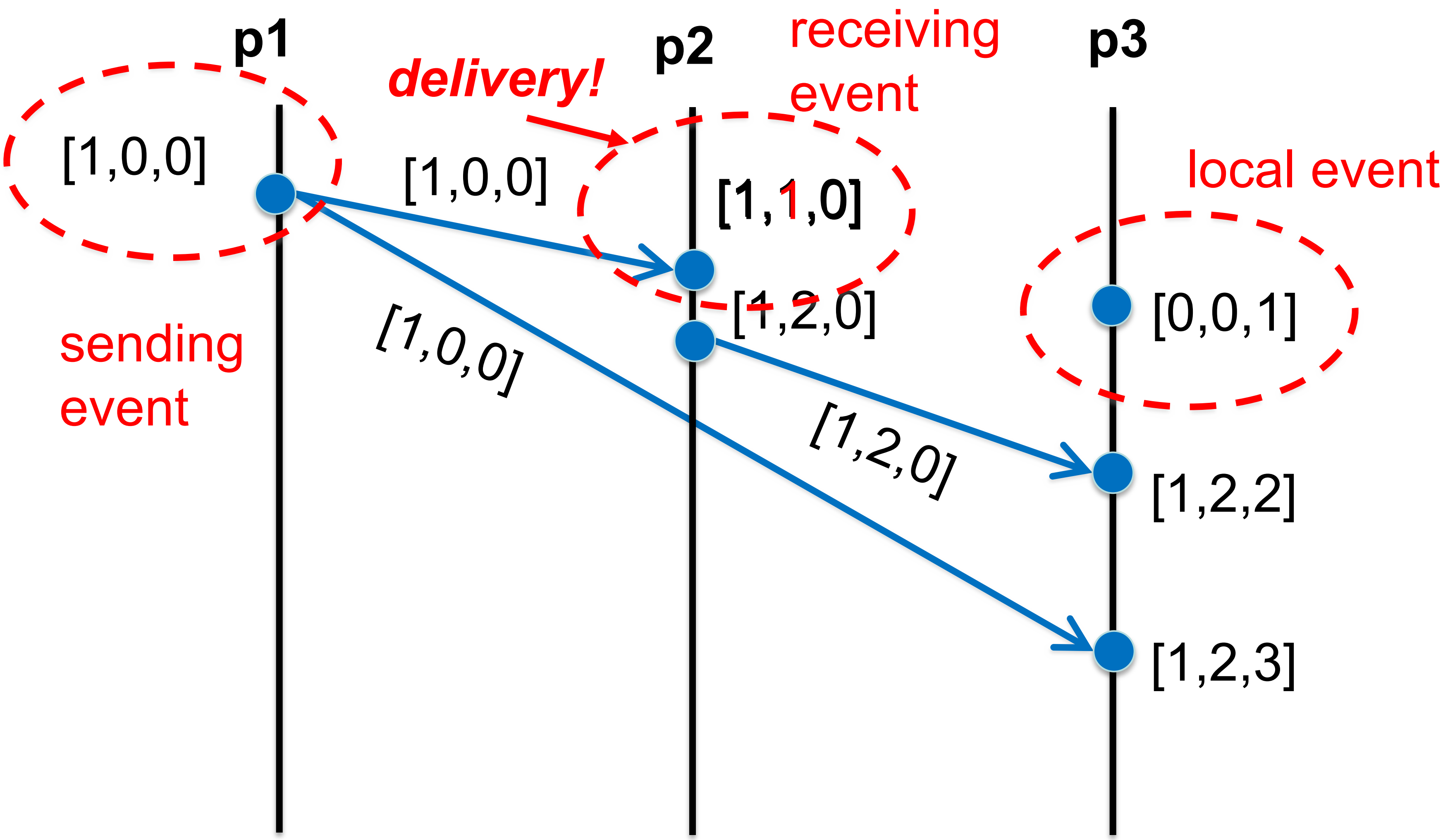
Relationships



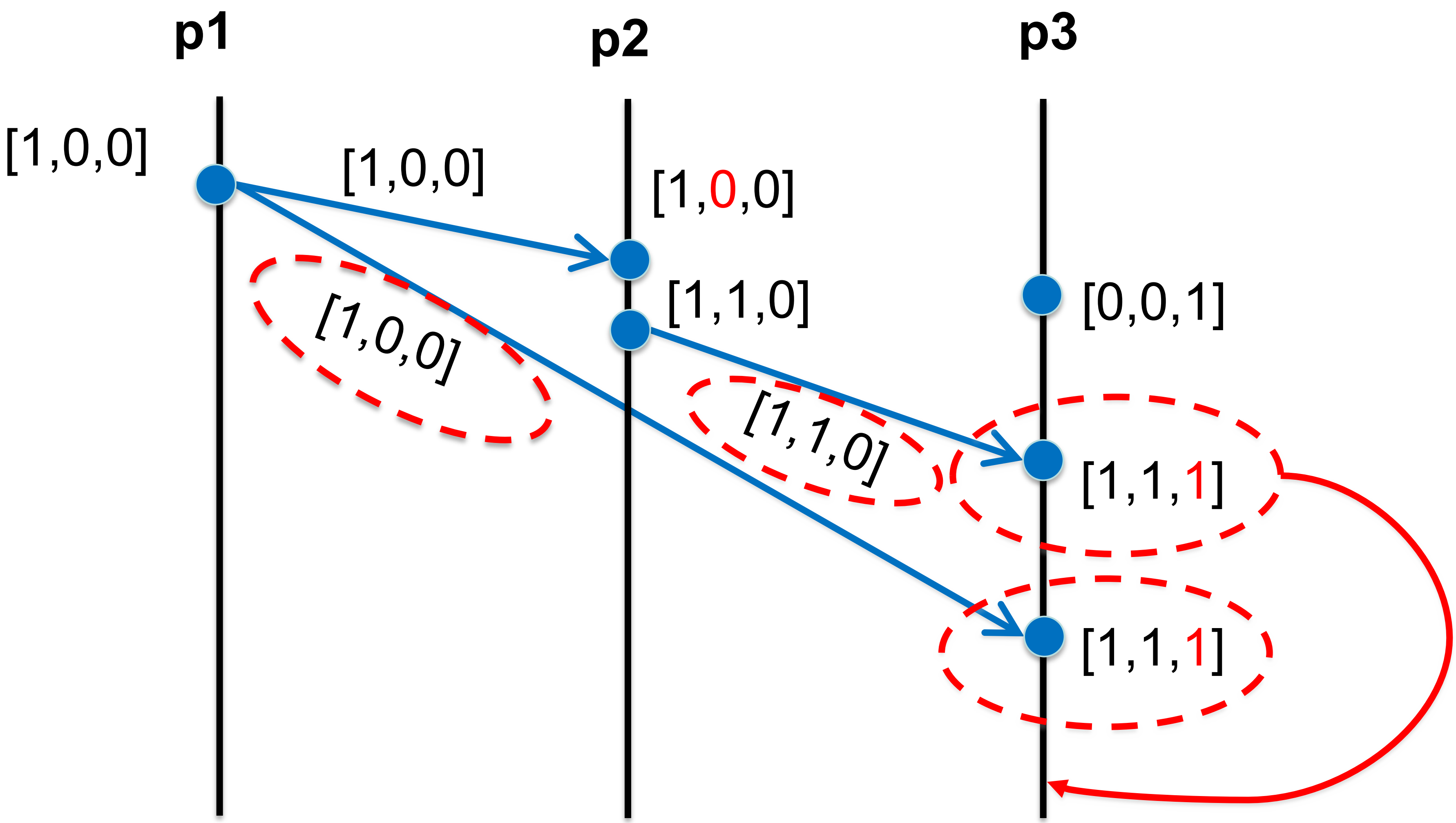
Broadcast

- Unicast
  - one sender, one receiver, point to point
- Multicast
  - a sender, multiple receiver
- Broadcast
  - a sender, everyone is receiver
  - How to implement <u>**causal**</u> broadcast?

Review of Vector Clock



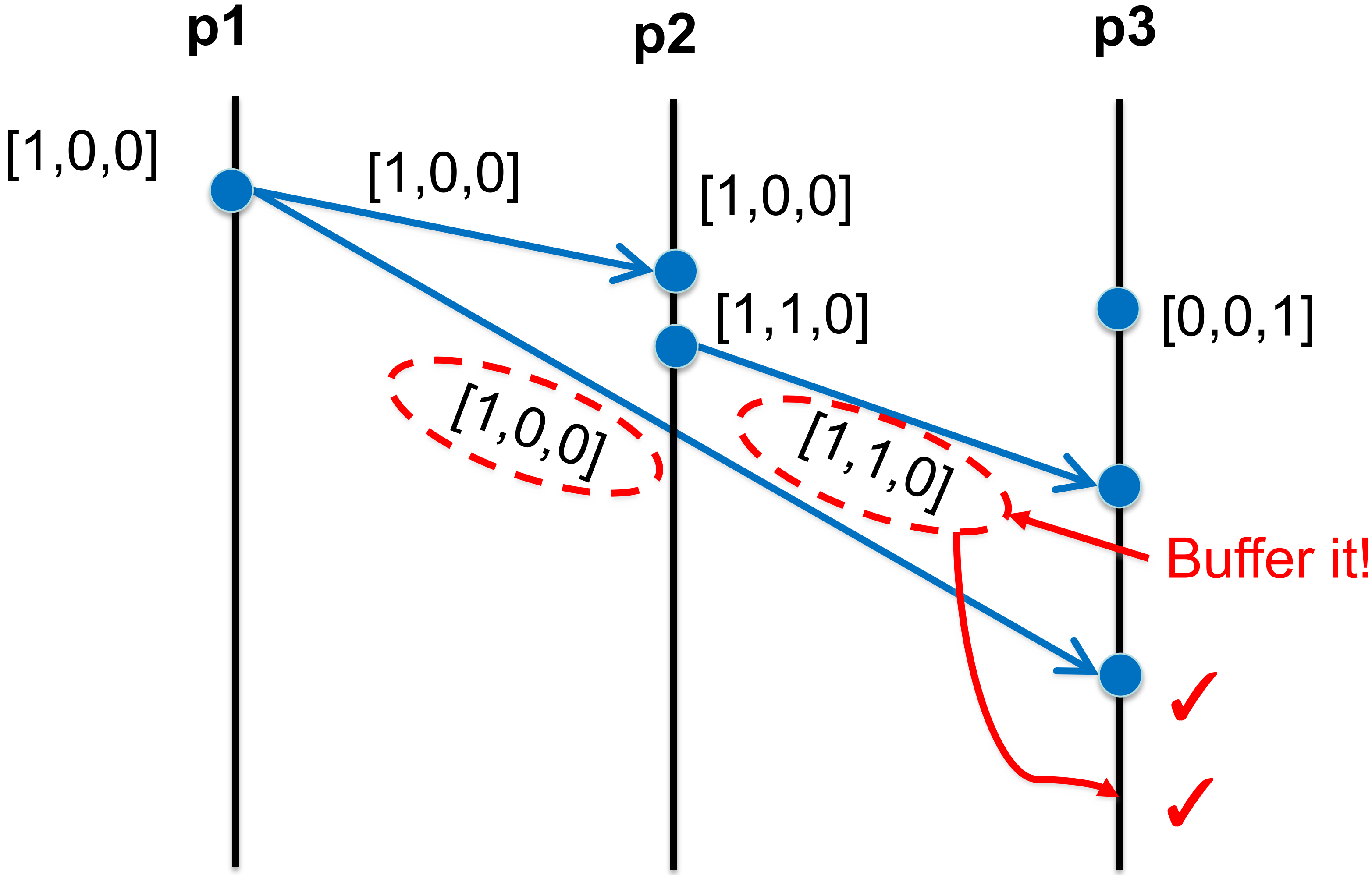
Considering Receiving Rather Than Delivery



Delivery Condition

- A message m can be delivered at process p if
  - for the sender process k, $VC_m[k] = VC_p[k] + 1$
  - for other processes l, $VC_m[l] \leq VC_p\ [l]$
- The condition ensures
  - for the sending process, messages sent before the current message should be delivered before it.
  - besides the sender process, no event happens before the the message on all other processes.

Implementing Causal Delivery

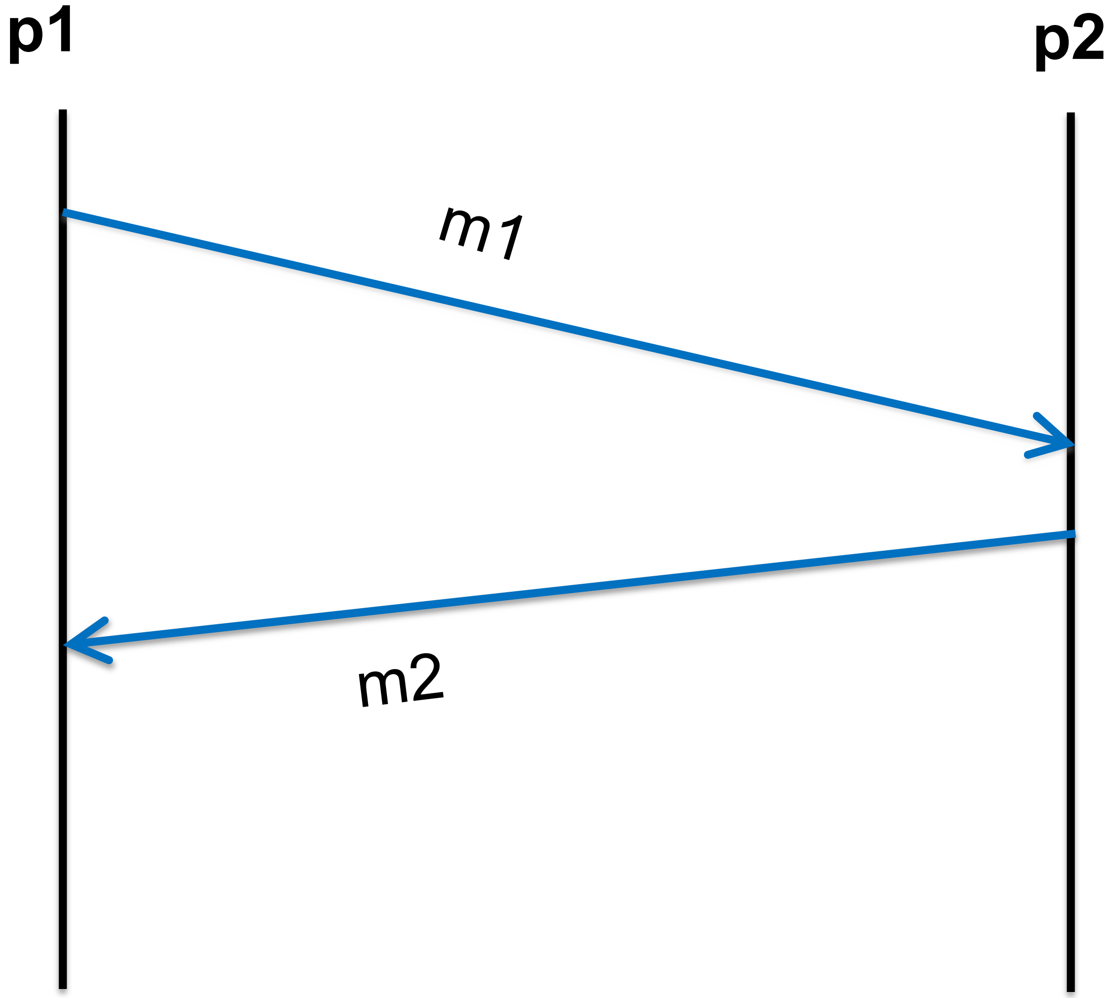


Safety and Liveness

- Safety property: nothing **bad** happens during execution
  - e.g., FIFO/Causal/Totally-ordered delivery, mutual exclusion

- Liveness property: something good **eventually** happens
  - e.g., reliable delivery, view synchronous communication

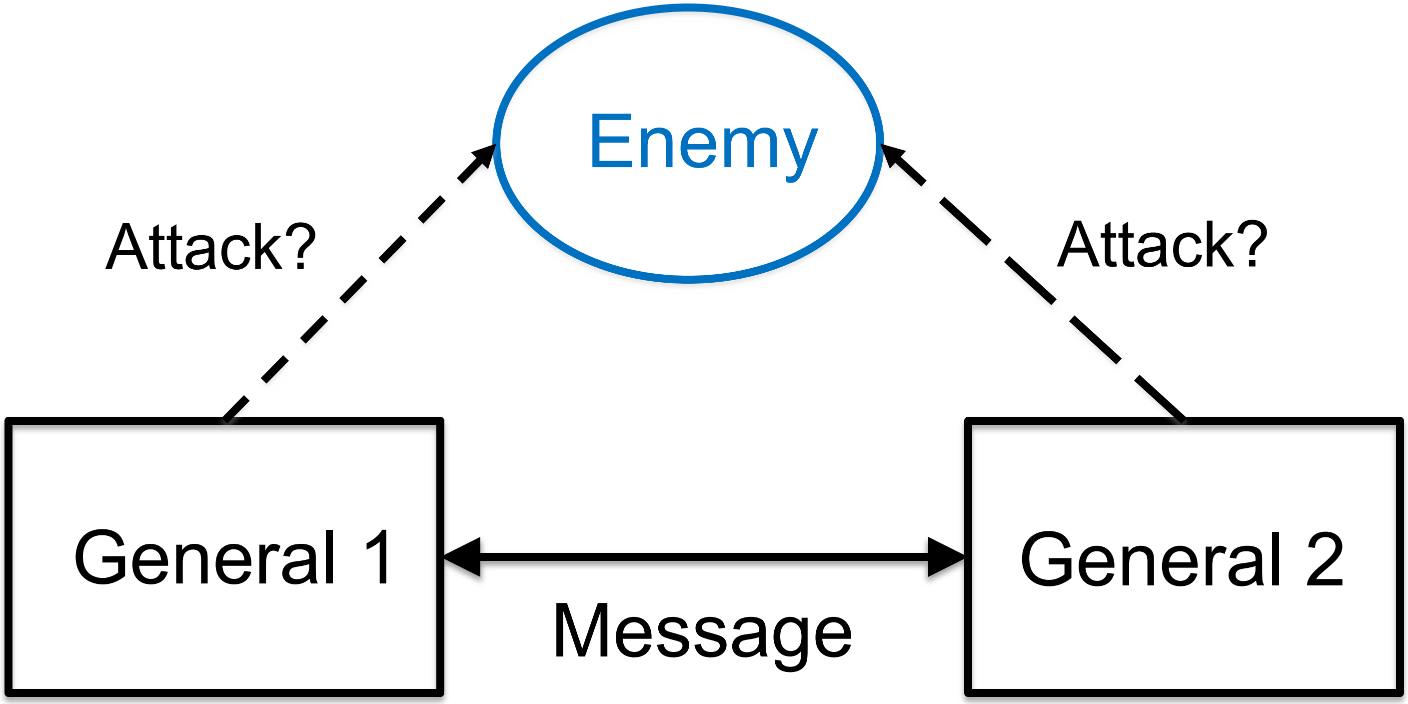
Any Possible Faults in the Simple Example



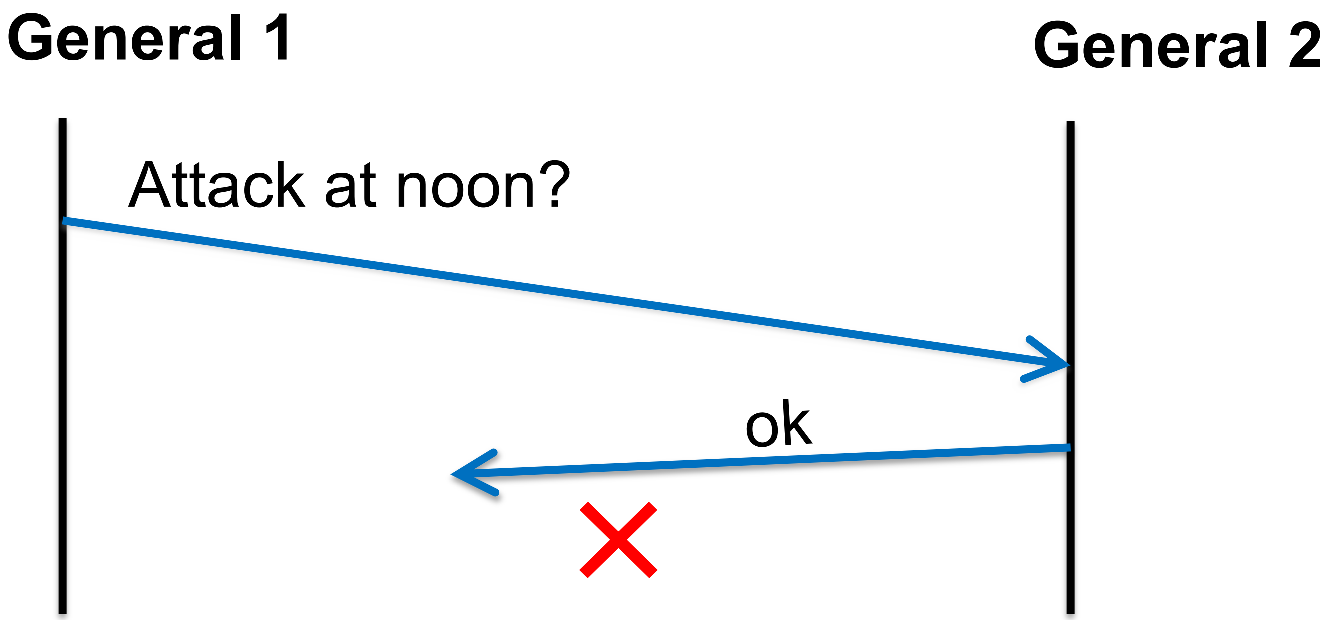
Fault Models

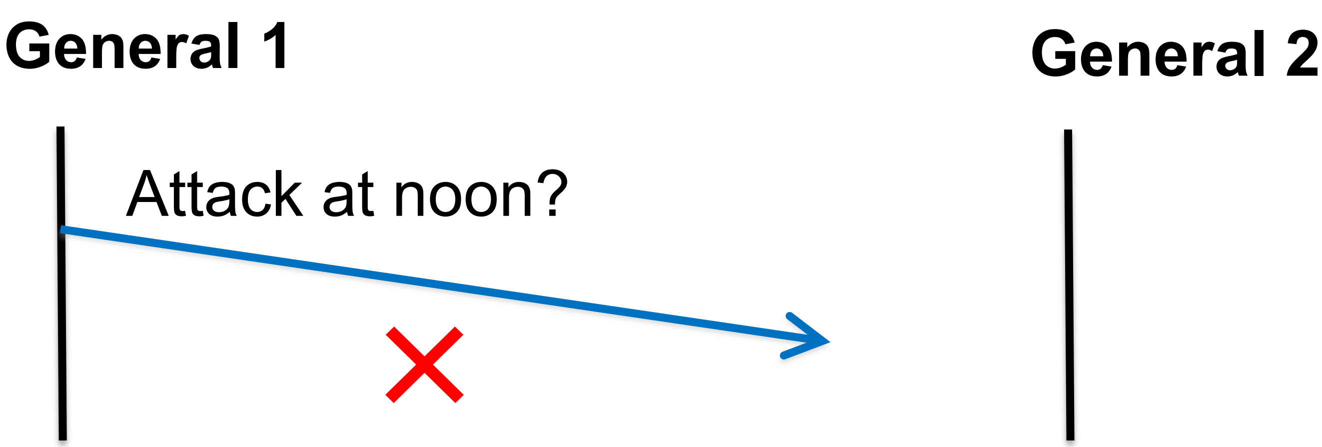
- Omission fault: a message gets lost. Process fails to send/receive one particular message.
  - message m1 or m2 gets lost
- Timing fault: process responds too late or too early.
  - message m1 or m2 is slow
- Crash fault: a process fails by halting. Stop sending/receiving messages.
  - process p2 crashes
- Byzantine fault: process behaves in an arbitrary way.
  - process p2 lies

Two Generals Problem

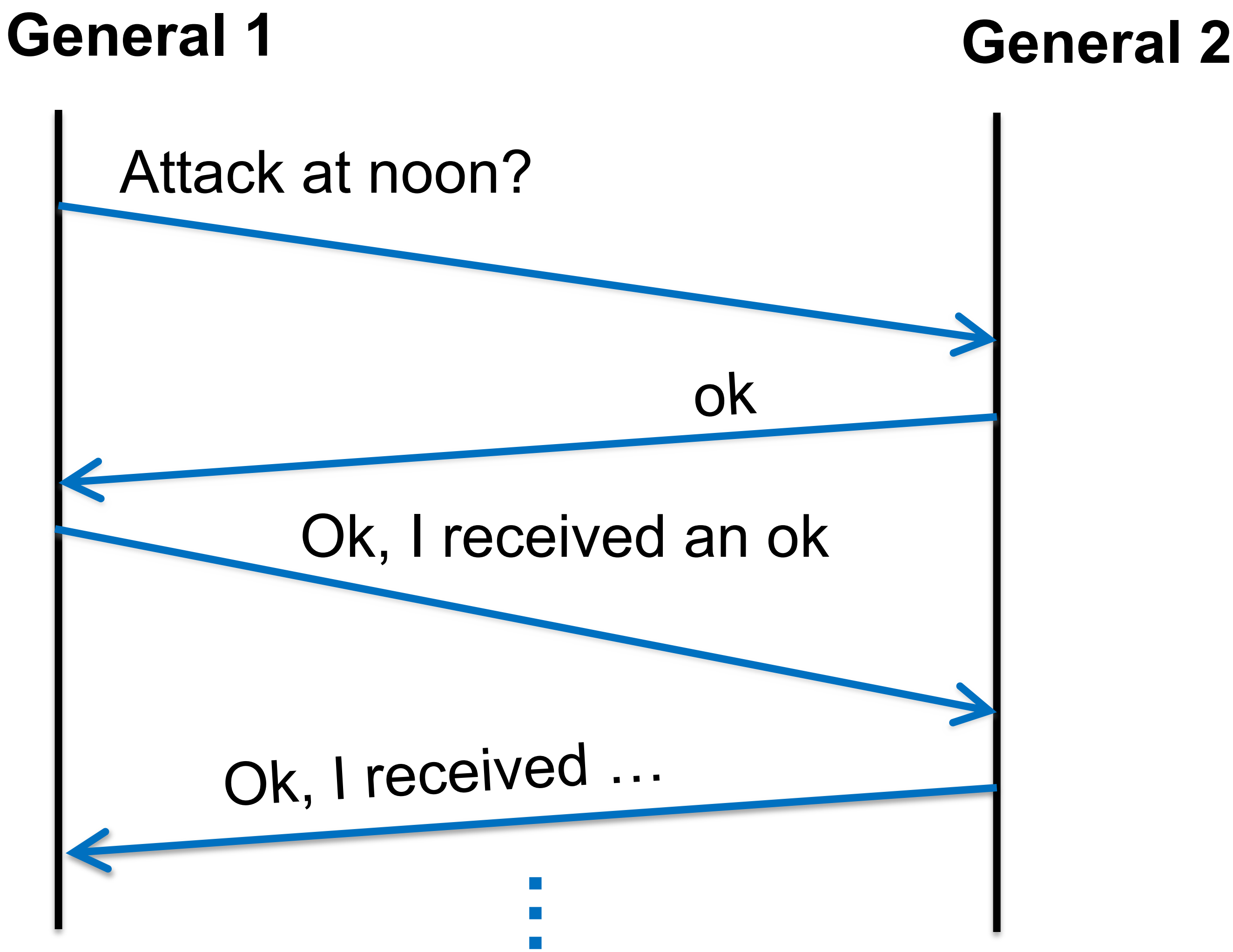


Two Generals Problem





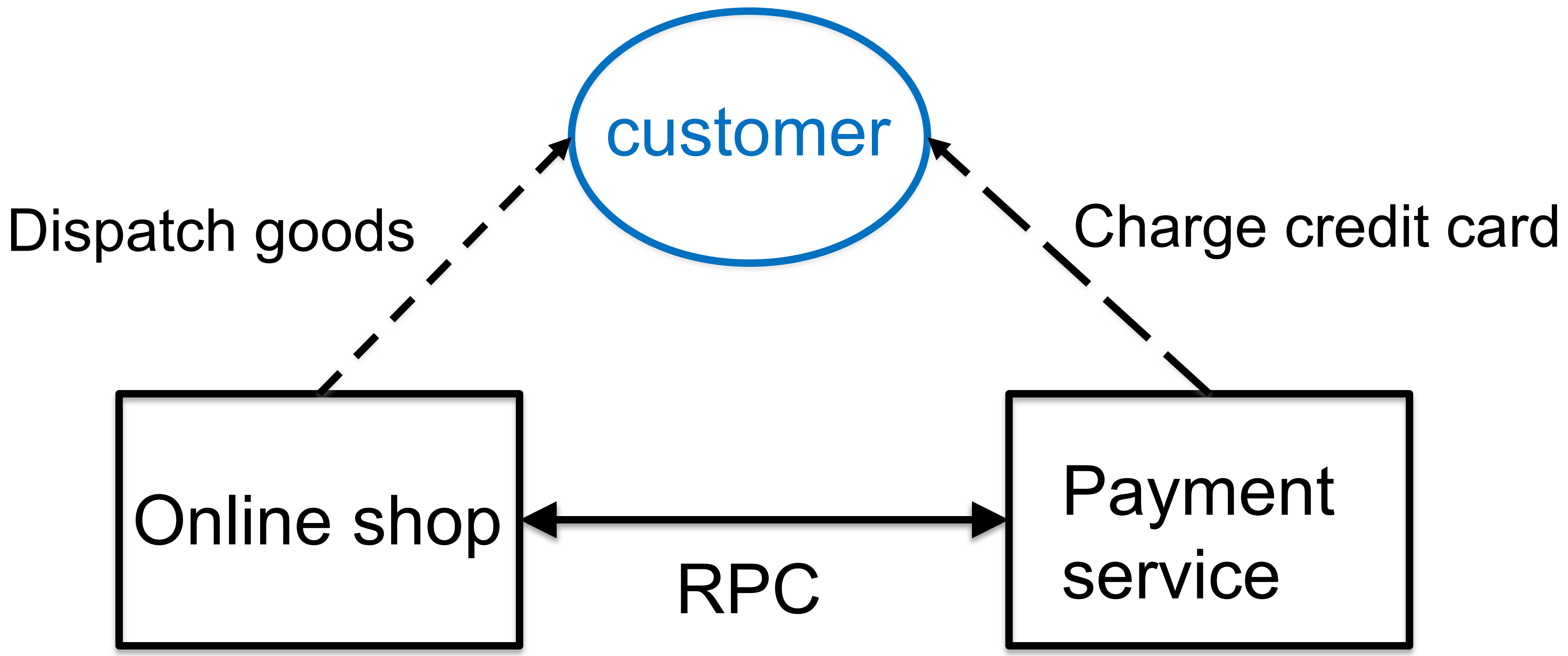
Two Generals Problem



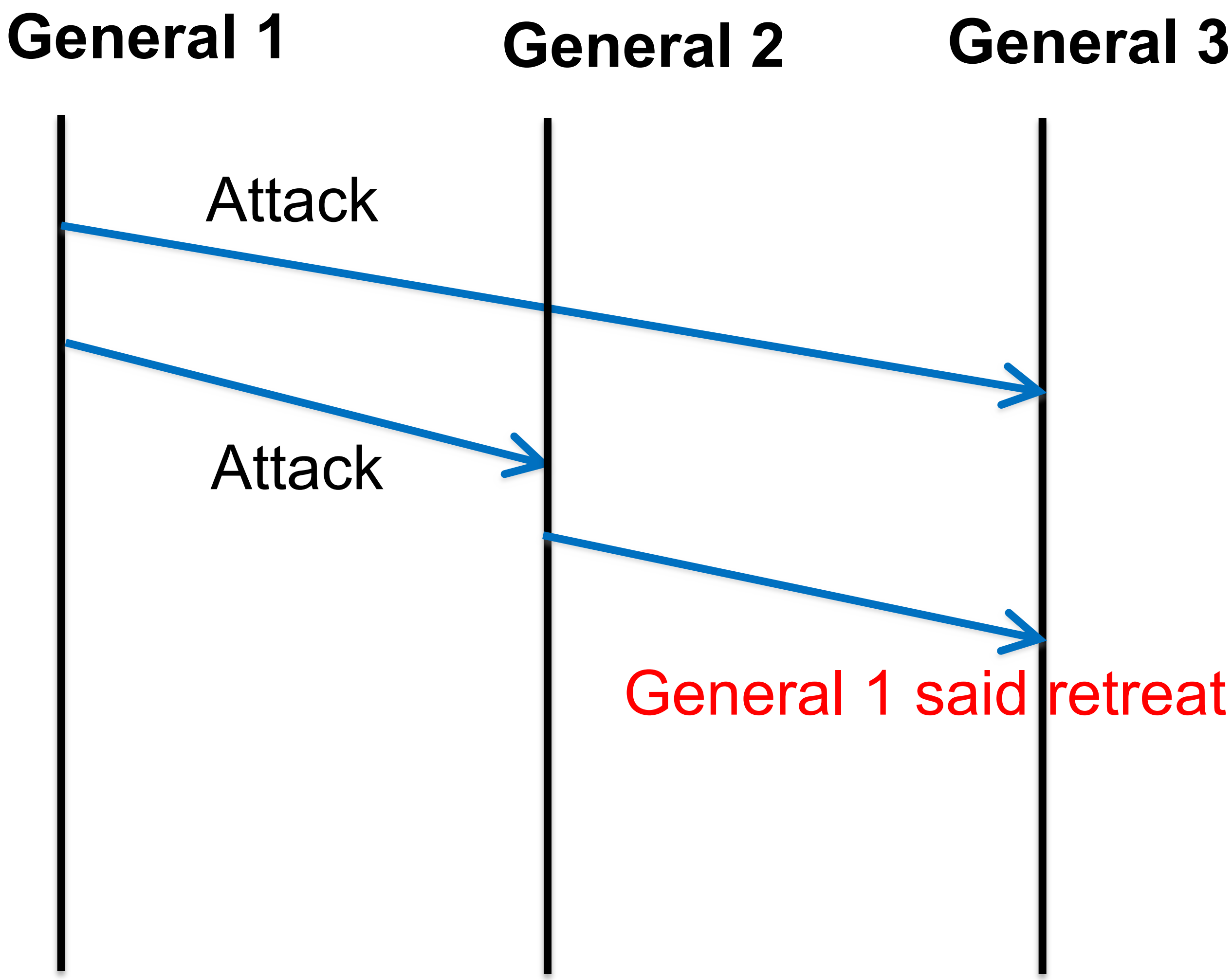
Two Generals Problem

- In the omission model, it is **impossible** for general 1 and general 2 to attack and for sure that the other will.
- How should the generals decide?
  - Send lots of messages to **increase probability** that one will get through
  - **common knowledge** before the attack

Application



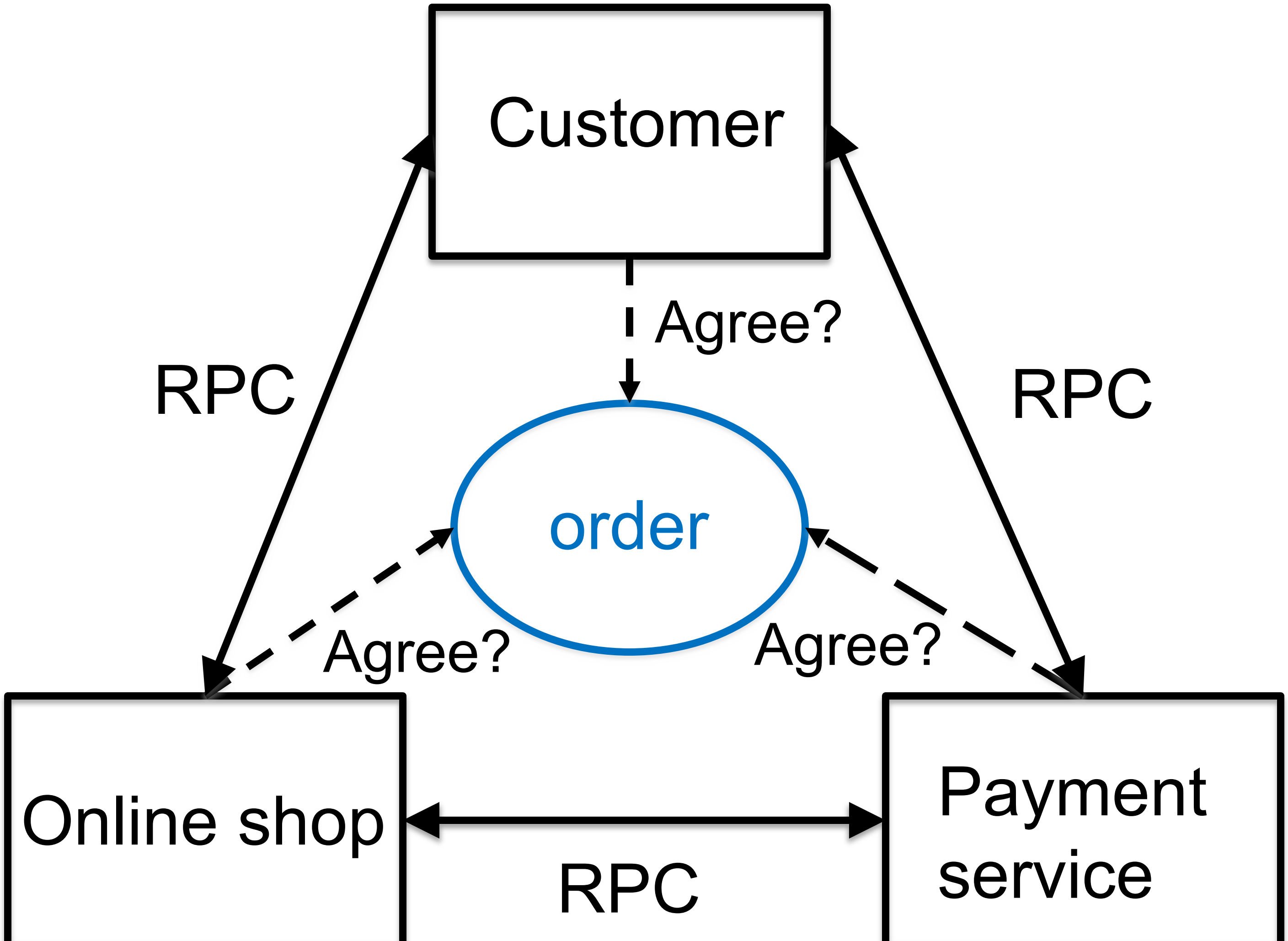
Byzantine Generals Problem



Byzantine Generals Problem

- Honest generals do not know who the malicious are
- The malicious generals may collude
- Honest generals must agree on the plan
- Need 3f+1 generals in total to tolerate f malicious generals
- Cryptography can help

Application



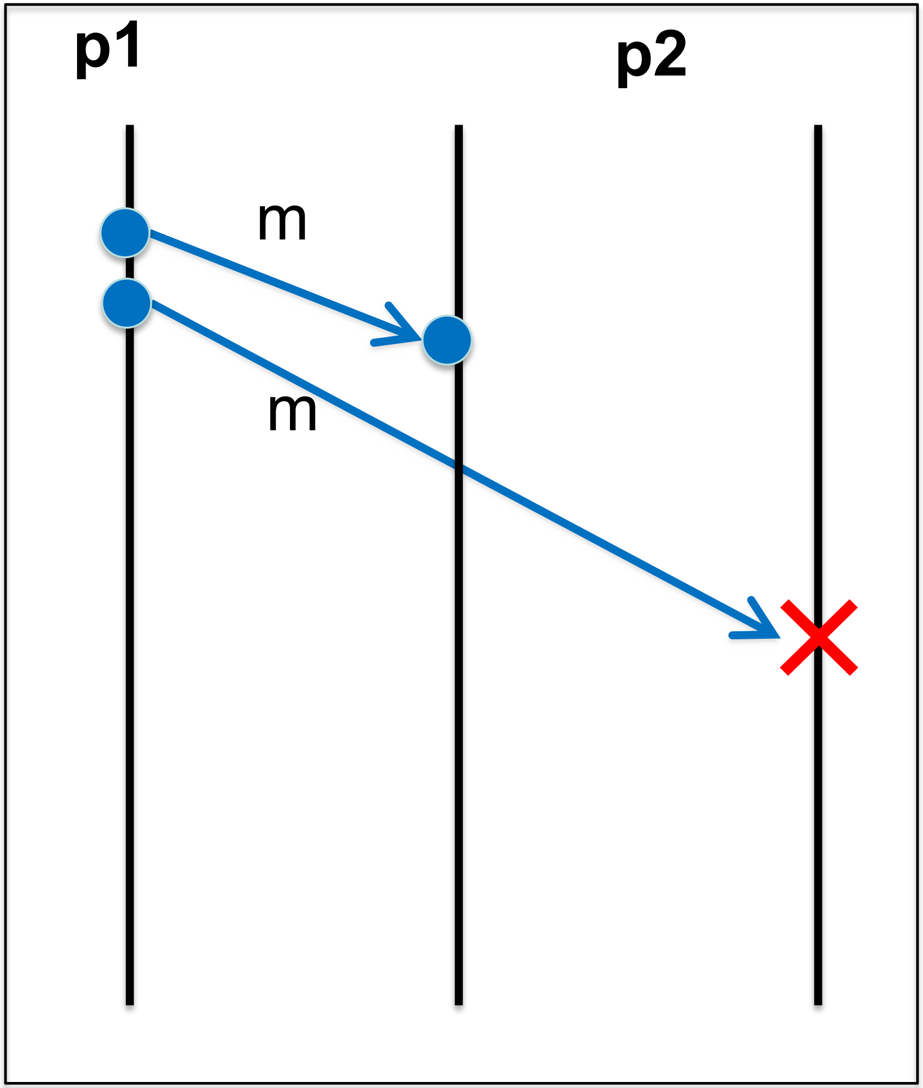
Fault Tolerance

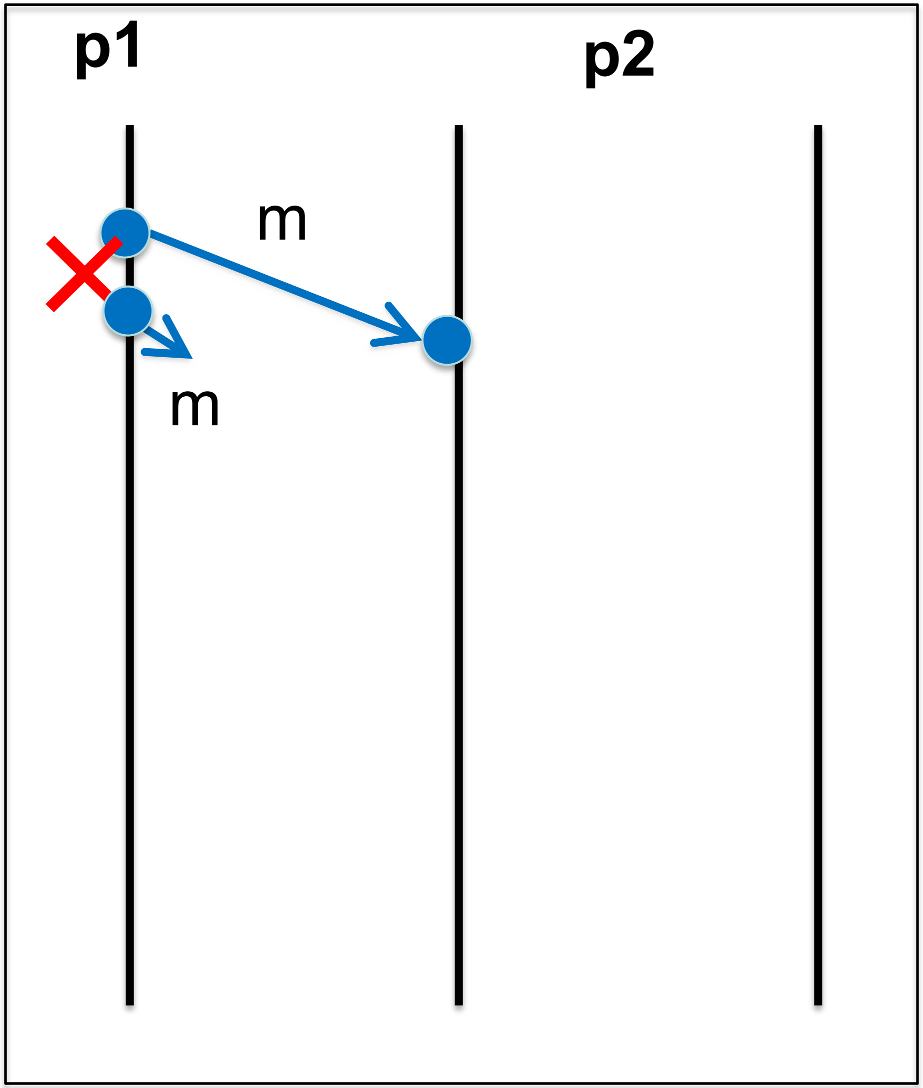
- What does it mean to tolerate a fault?
  - a correct system satisfies both **safety** and **liveness**
- System is both safety and liveness: masking.
- System is not safety but liveness: non-masking.
- System is safety but not liveness: fail-safe.

Broadcast Algorithms

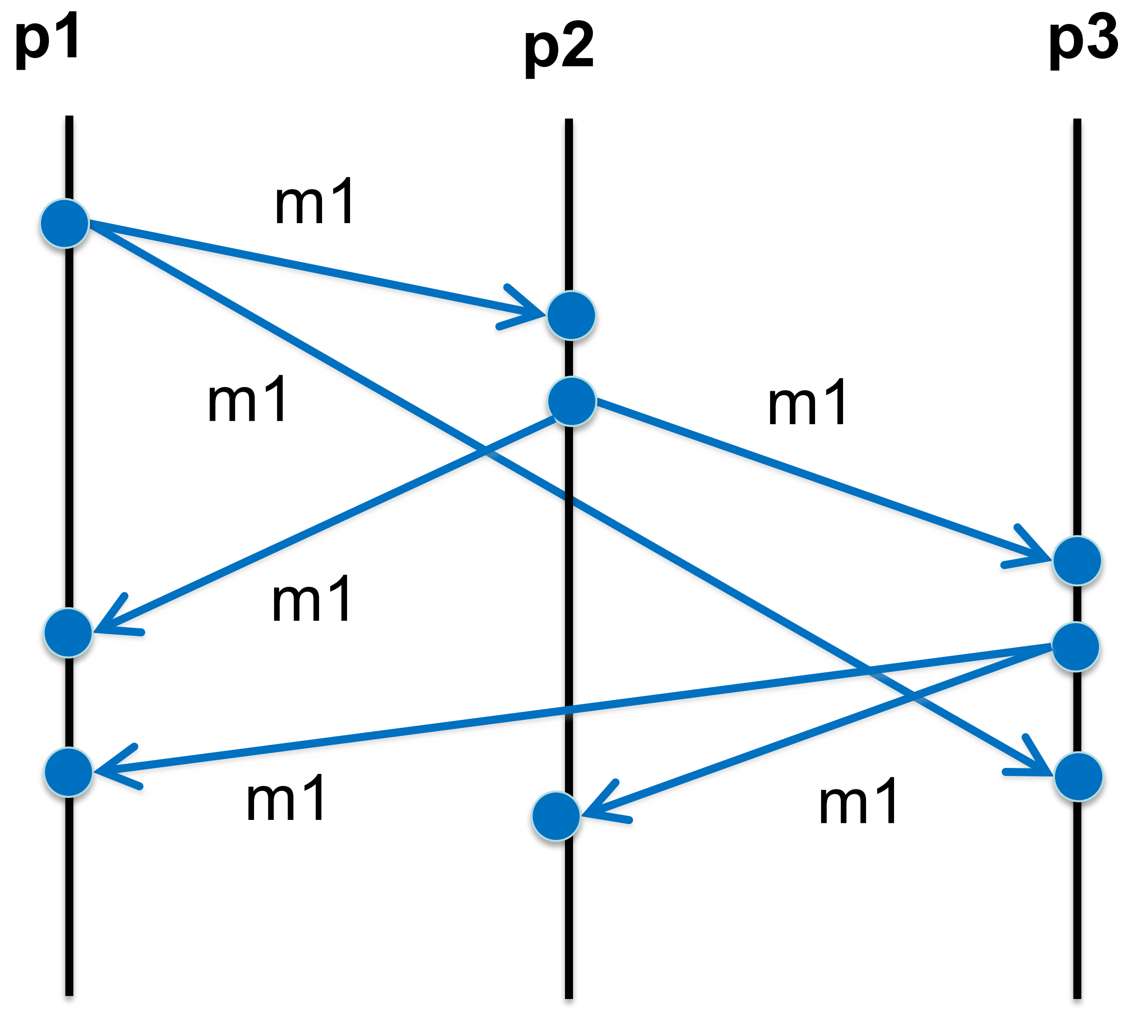
- Break down into two layers:
  - Make best-effort broadcast reliable by **retransmitting** dropped messages
  - idempotent and non-idempotent operations
  - Enforce delivery order on top of reliable broadcast
- Reliable broadcast: If a non-faulty process sends a message m, then all the non-faulty processes eventually deliver m.
- First attempt: broadcasting node sends message directly to every other node
  - Use reliable links (**retry** + **deduplicate**)
  - Problem: node may crash before all messages delivered

Two Examples

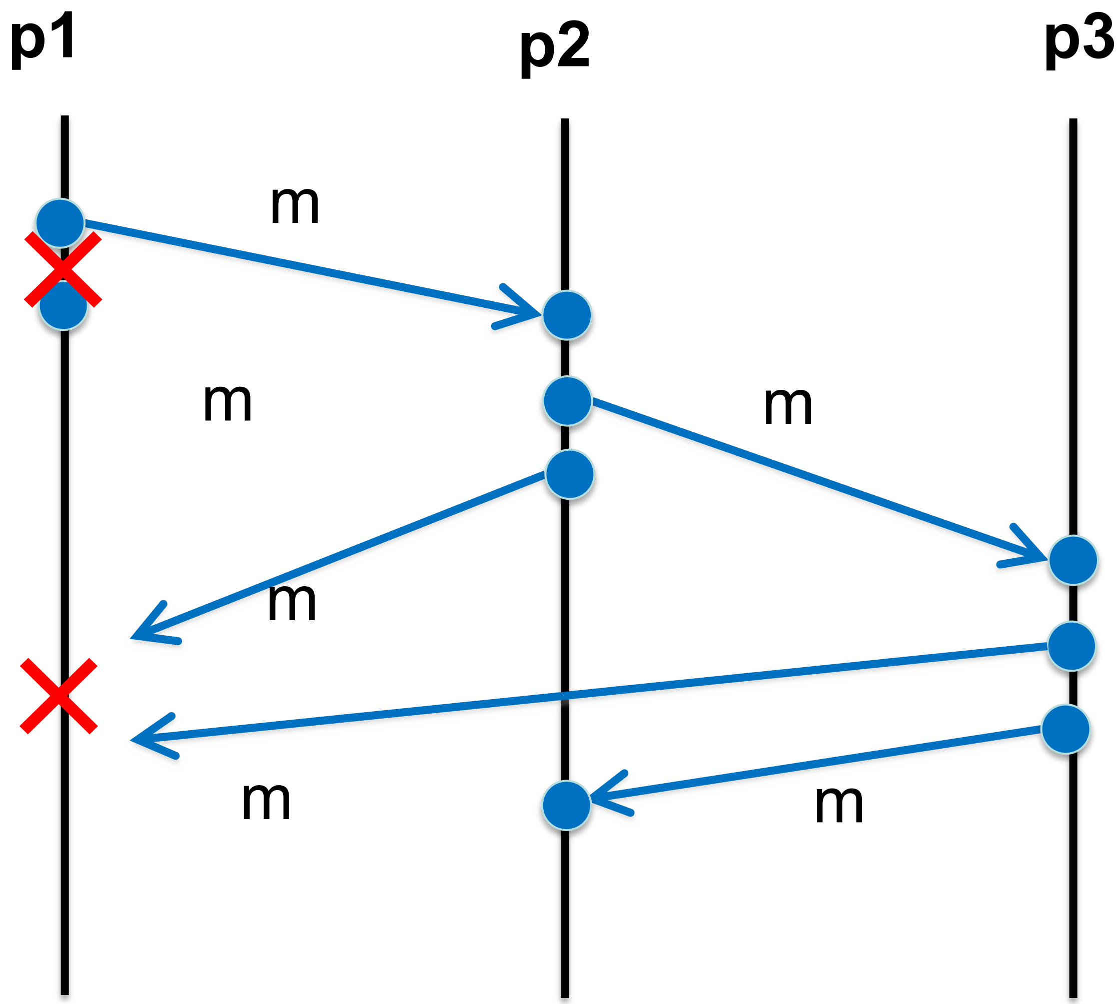




Eager Reliable Broadcast



Eager Reliable Broadcast



Implementing Totally-ordered broadcast

- Single leader approach:
  - One node is designated as **leader** (sequencer)
  - To broadcast message, send it to the leader; leader broadcasts it via FIFO broadcast.
  - Problem: leader crashes
  - Changing the leader safely is difficult
- Lamport clocks approach:
  - Attach **Lamport timestamp** to every message
  - Deliver messages in total order of timestamps
  - Problem: how do you know if you have seen all messages with timestamp < T? Need to use FIFO links and wait for message with timestamp ≥ T from every node

Fault-tolerant Totally-ordered Broadcast

- Consensus and total order broadcast are formally equivalent
  - Traditional formulation of consensus: several nodes want to come to agreement about a **single value**
  - In context of total order broadcast: this value is the **next message to deliver**
  - Once one node decides on a certain message order, all nodes will decide the same order
- Consensus algorithms
  - **Paxos:** single-value consensus
  - **Multi-Paxos**: generalization to total order broadcast
  - **Raft, Viewstamped Replication, Zab**: FIFO-total order broadcast by default

Summary

- Delivery properties
  - FIFO delivery, causal delivery, totally-ordered delivery
  - Implementing causal delivery
- Fault tolerance
  - Safety and liveness
  - Fault models
  - Two generals problem
  - Byzantine generals problem
- Reliable broadcast