Virtual Synchrony

Ki Suh Lee

Some slides are borrowed from Ken, Jared (cs6410 2009) and Justin (cs614 2005)

The Process Group Approach to Reliable Distributed Computing

- Ken Birman
  - Professor, Cornell University

  - Isis
  - Quicksilver
  - Live Object

Understanding the Limitations of Causally and Totally Ordered Communication

- David Cheriton
  - Stanford
  - PhD – Waterloo
  - Billionaire

- Dale Skeen
  - PhD – UC Berkeley
  - Distributed pub/sub communication
  - 3-phase commit protocol

Recap…

- End-to-End Argument
- Multicast
- Partial/Total Ordering
  - Happens-before relation
- Logical/Physical Clocks
- Distributed snapshop
- Consensus

Recap

- Asynchronous vs. synchronous
- Failure model
  - Crash-stop (fail-stop) Failures
  - Byzantine Failures

Distributed computing

- 1978 Lamport's
  "
  Time, Clocks, and the Ordering of Events in a Distributed System"
- 1983 Schneider's *State machine replication*
- 1985 FLP's *the impossibility of asynchronous fault-tolerant consensus*

- 1981 transactional serializability (2PC)
- 1981 Non-blocking 3PC

Motivation

- Distributed system with
  - Fault-tolerance
  - Reliability
  - Easy programmability

Virtual Synchrony

- In the early 1980's
- Key idea: equate "group" with "data abstraction"
  - Each group implements some object
  - An application can belong to many groups

Virtual Synchrony

- The user sees what looks like a synchronous execution
  - Simplifies the developer's task

- Process groups with state transfer, automated fault detection and membership reporting
- Ordered reliable multicast, in several flavors
- Extremely good performance

Historical Aside

- Isis (Virtual synchrony)
  - Weaker properties – not quite "FLP consensus"
  - Much higher performance (orders of magnitude)
  - Simple Dynamic membership control
- Paxos (state machine)
  - Closer to FLP definition of consensus
  - Slower (by orders of magnitude)
  - Sometimes can make progress in partitioning situations where virtual synchrony can't
  - Complex dynamic membership control

Programming with groups

- Many systems just have one group
  - E.g. replicated bank servers
  - Cluster mimics one highly reliable server
- But we can also use groups at finer granularity
  - E.g. to replicate a shared data structure
  - Now one process might belong to many groups
- A further reason that different processes might see different inputs and event orders

ISIS

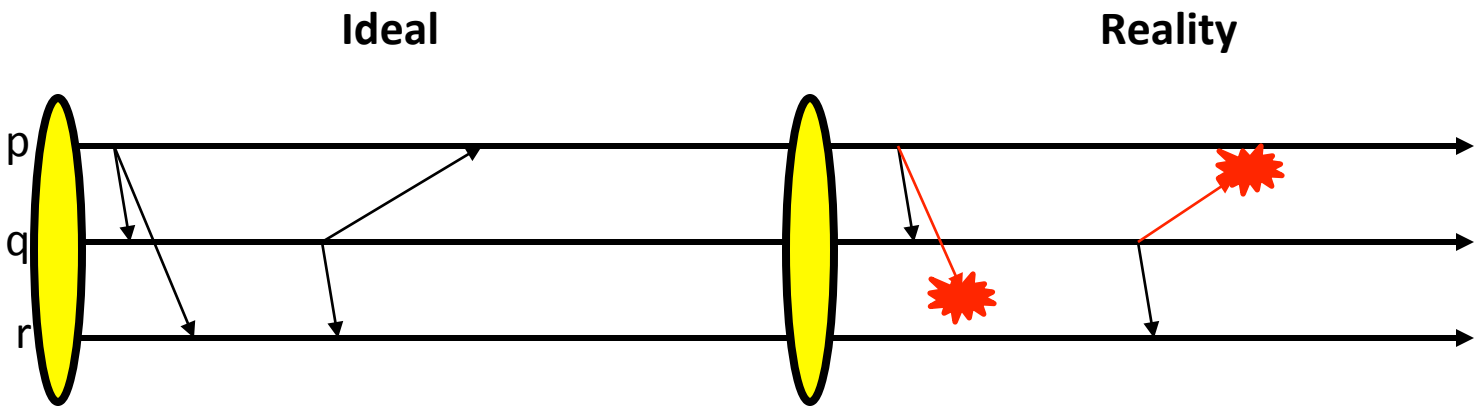
Assumptions

- Fail-stop model
- Clocks are not synchronized
- Unreliable network
- Network partitions is rare
- Failure detection subsystem
  - Consistent system-wide view

Difficulties

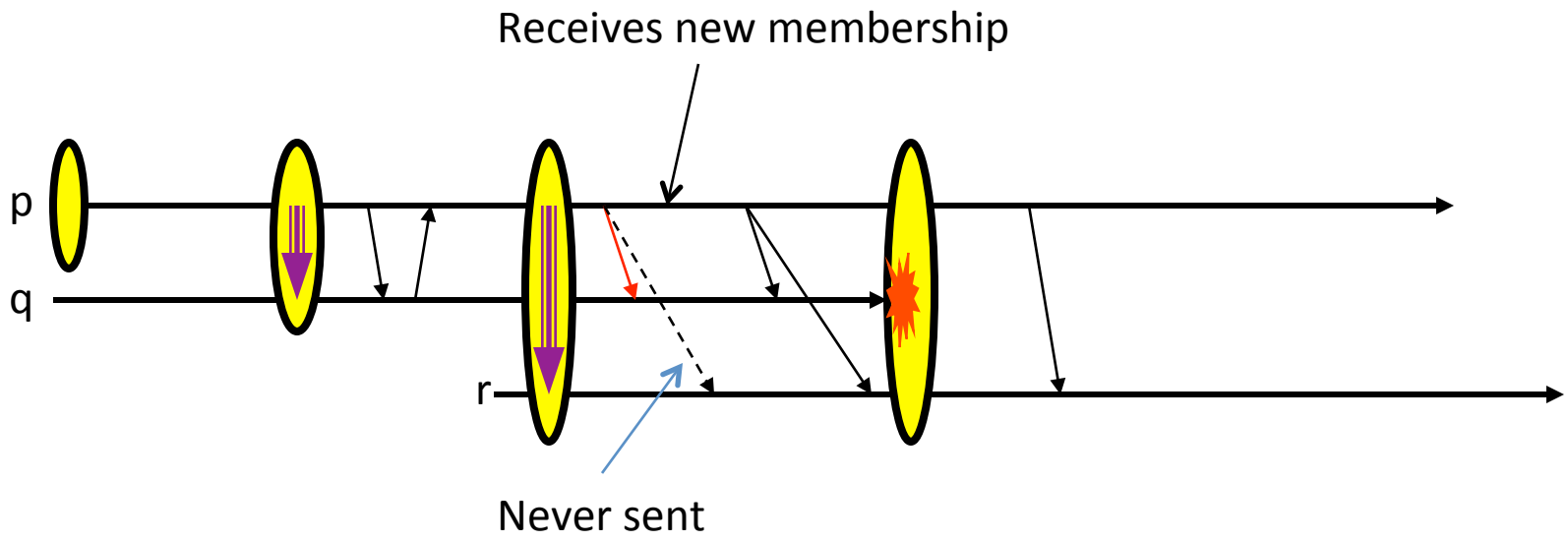
- Conventional message passing technologies
  - TCP, UDP, RPC, …
- Group addressing
- Logical time and causal dependency
- Message delivery ordering
- State transfer (membership change)
- Fault tolerance
- …

No Reliable Multicast



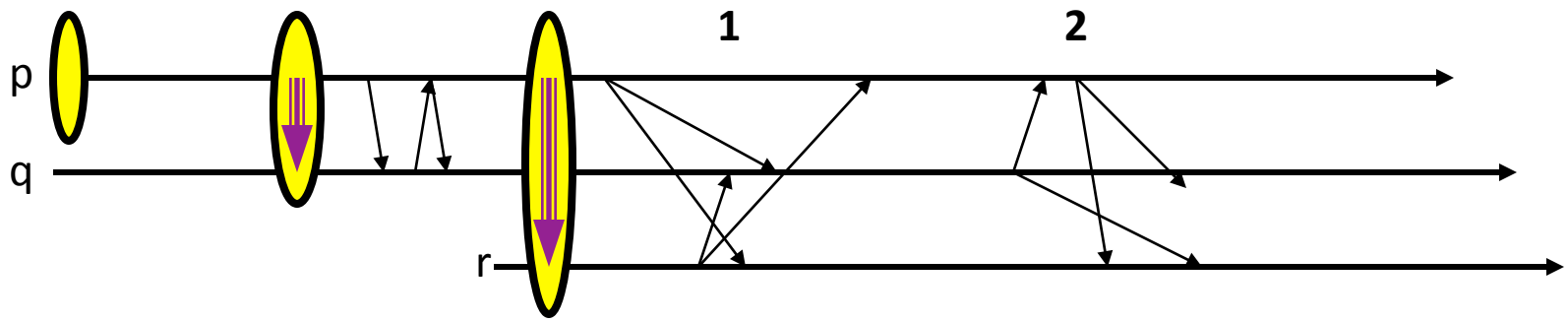
- UDP, TCP, Multicast not good enough
- *What is the correct way to recover?*

Membership Churn



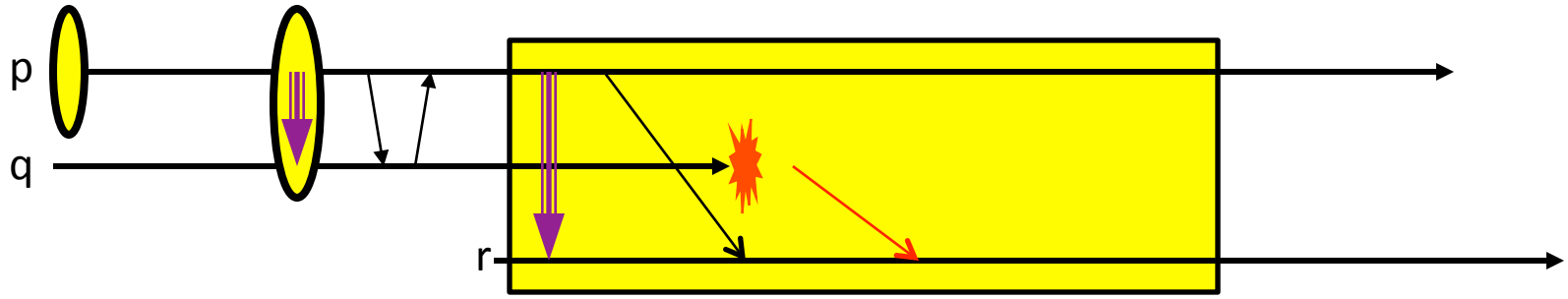
- Membership changes are not instant
- How to handle failure cases?

Message Ordering



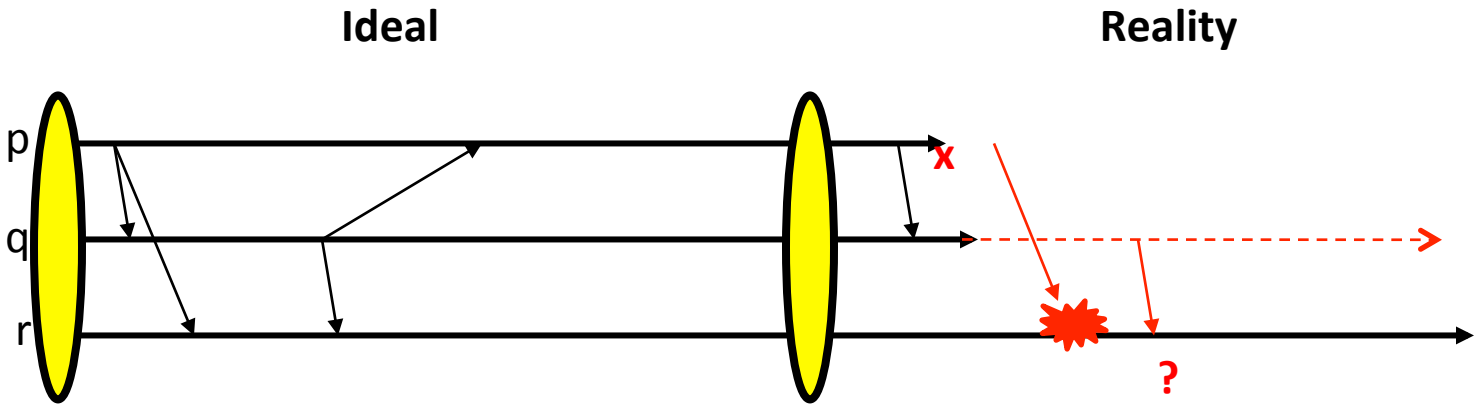
- Everybody wants it!
- How can you know if you have it?
- How can you get it?

State Transfers



- New nodes must get current state
- Does not happen *instantly*
- How do you handle nodes failing/joining?

Failure Atomicity



- Nodes can fail mid-transmit
- Some nodes receive message, others do not
- Inconsistencies arise!

Process Groups

- Distributed groups of cooperating programs
- Pub/sub style of interaction

- Requirements
  - Group communication
  - Group membership as input
  - Synchronization

Process Groups

- Anonymous group
  - Group addressing
  - All or none delivery
  - Message Ordering
- Explicit group
  - Members cooperate directly
  - Consistent views of group membership

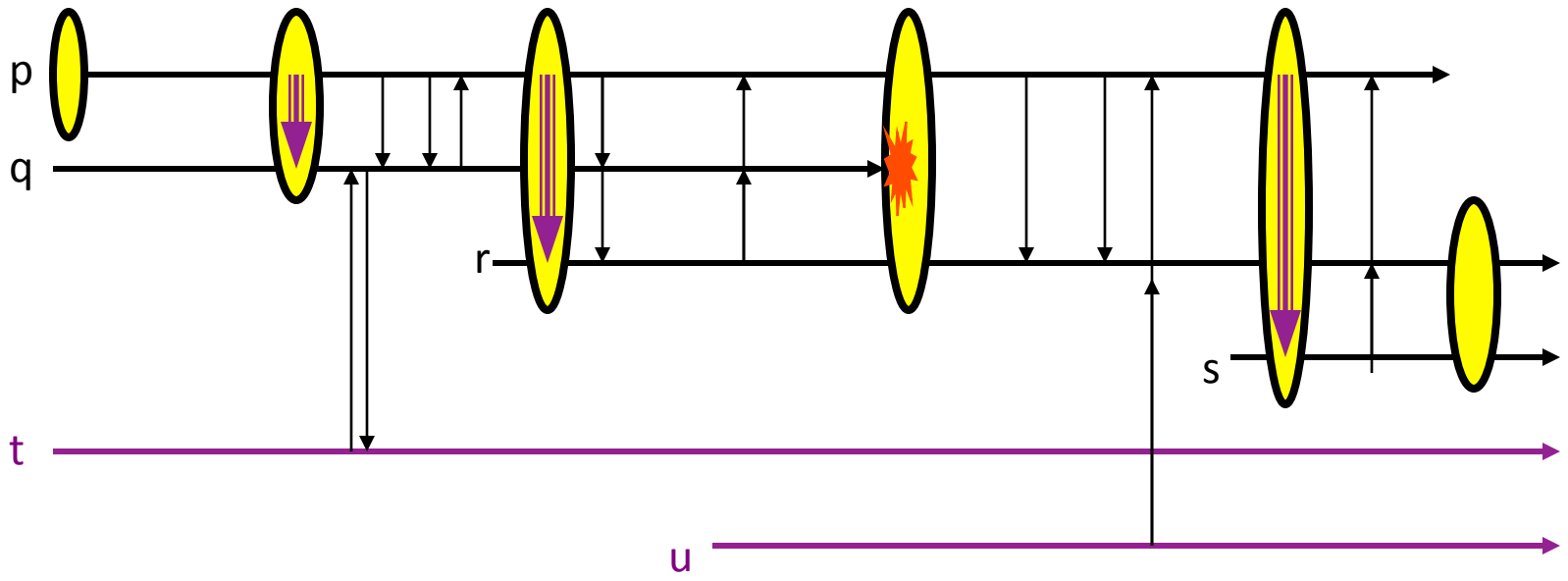
Process groups

- The group view gives a simple leader election rule
- A group can easily solve consensus
- A group can easily do consistent snapshot

Close Synchrony

- Lock-step execution model
  - Implementing synchronous model in asynchronous environment
  - Order of events is preserved
  - A multicast is delivered to its full membership

Close Synchrony



Close Synchrony

- Not practical
  - Impossible in the presence of failures
  - Expensive

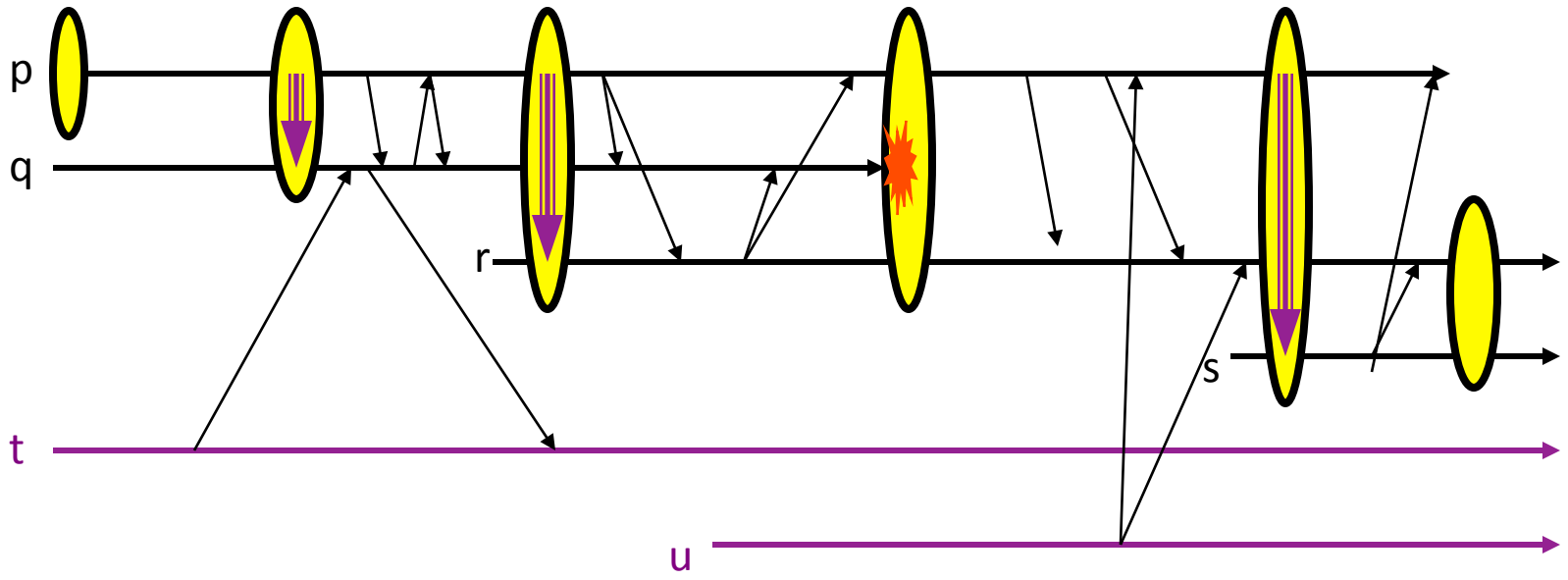
- We want close synchrony with high throughput.

  => *Virtual Synchrony*

Virtual Synchrony

- Relax synchronization requirements where possible
  - Different orders among concurrent events won't matter as long as they are delivered.

Asynchronous Execution



ABCAST

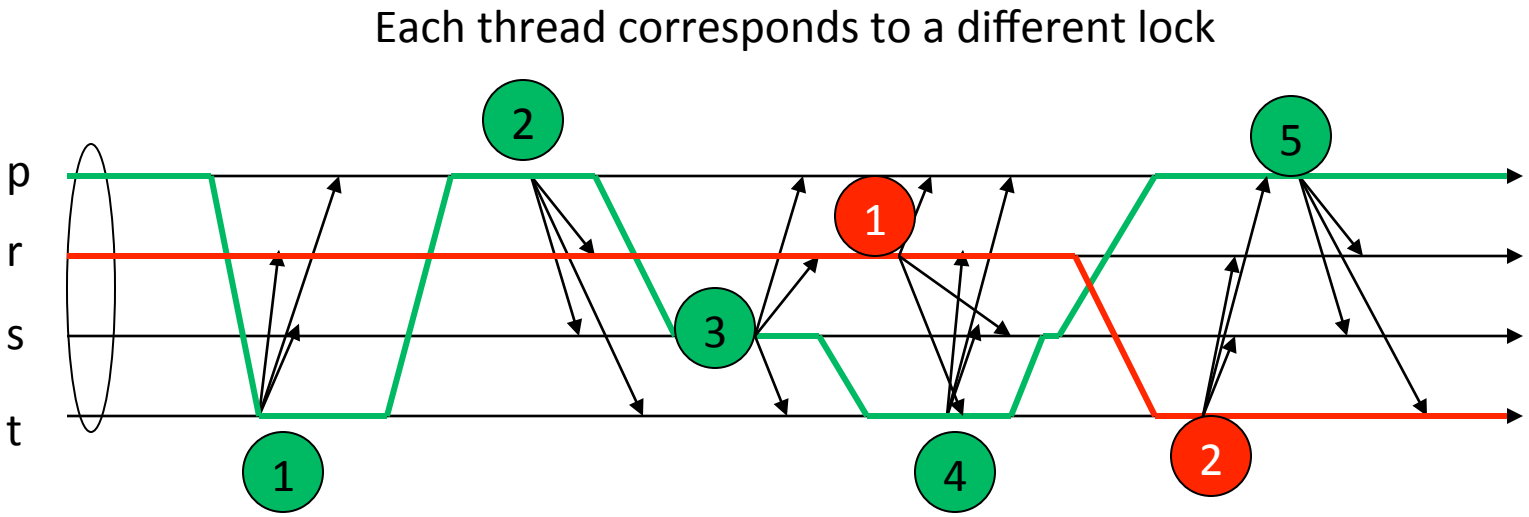
- *Atomic delivery ordering*
- Stronger Ordering, but costly
- locking or token passing

- Not all applications need this…

CBCAST

- Two messages can be sent to concurrently *only when their effects on the group are independent*
- If m1 causally precedes m2, then m1 should be delivered before m2.
- Weaker then ABCAST
- Fast!

When to use CBCAST?



- When any conflicting multicasts are uniquely ordered along a single causal chain
- …..This is Virtual Synchrony

Benefits

- Assuming a closely synchronous execution model
- Asynchronous, pipelined communication
- Failure handling through a system membership list

Isis toolkit

- A collection of higher-level mechanisms for process groups
- Still used in
  - New York and Swiss Stock Exchange
  - French Air Traffic Control System
  - US Navy AEGIS

Problems

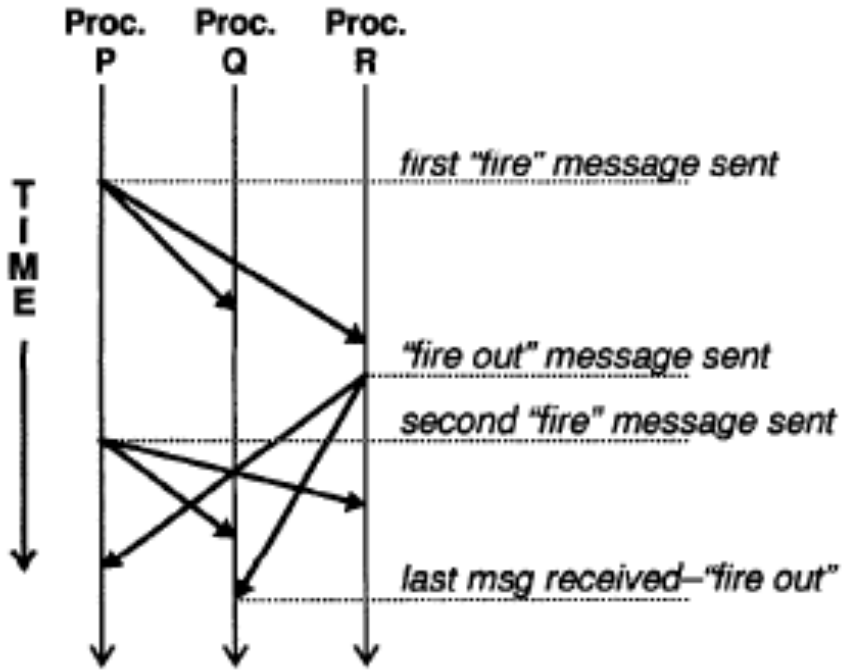
- Message delivery is atomic, but not *durable*
- Incidental ordering
  - Limited to ensure communication-level semantics
  - Not enough to ensure application-level consistency.
- Violates end-to-end argument.

Limitations

- Can't say "*for sure*"
- Can't say the "*whole story*"
- Can't say "*together*"
- Can't say "*efficiently*"

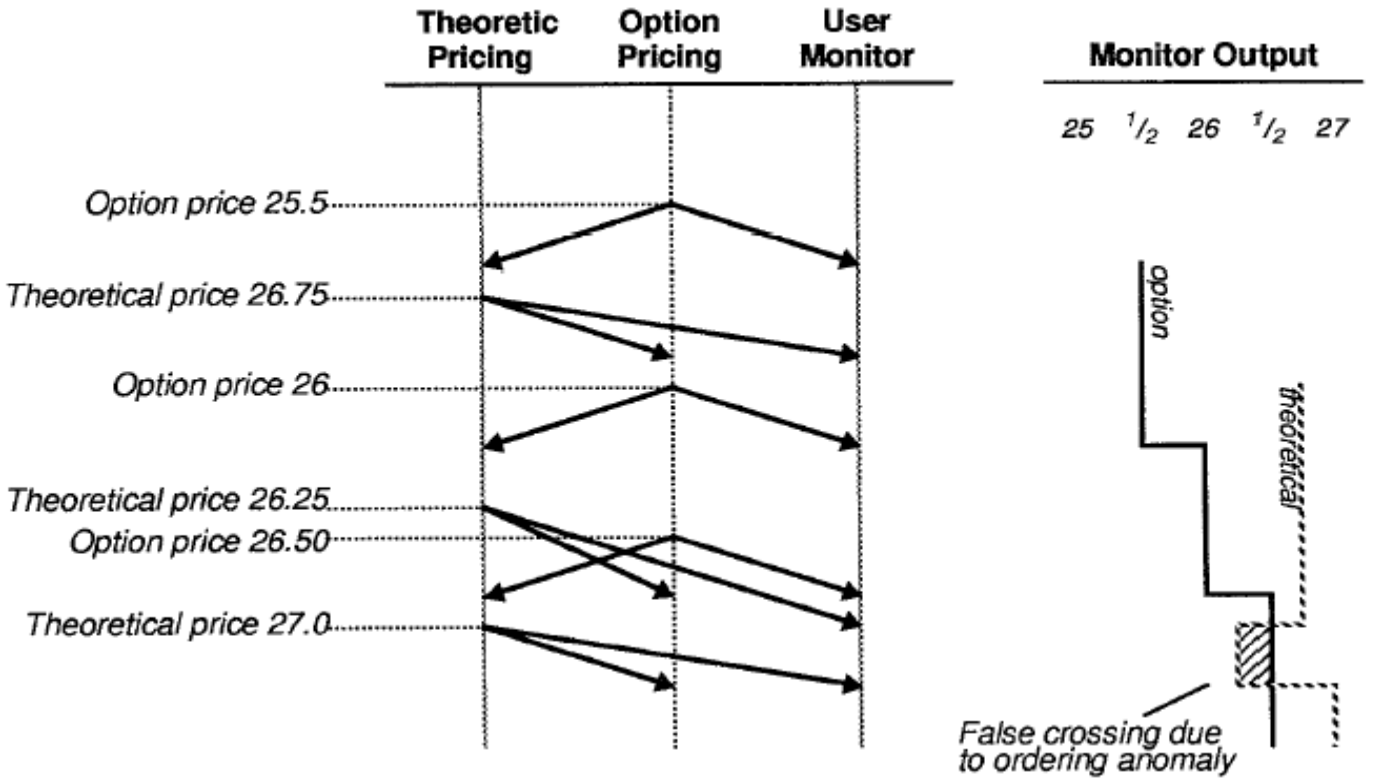
Can't say "*for sure*"

- Causal relationships at semantic level are not recognizable
- External or 'hidden' communication channel.



Can't say "*together*", "whole story"

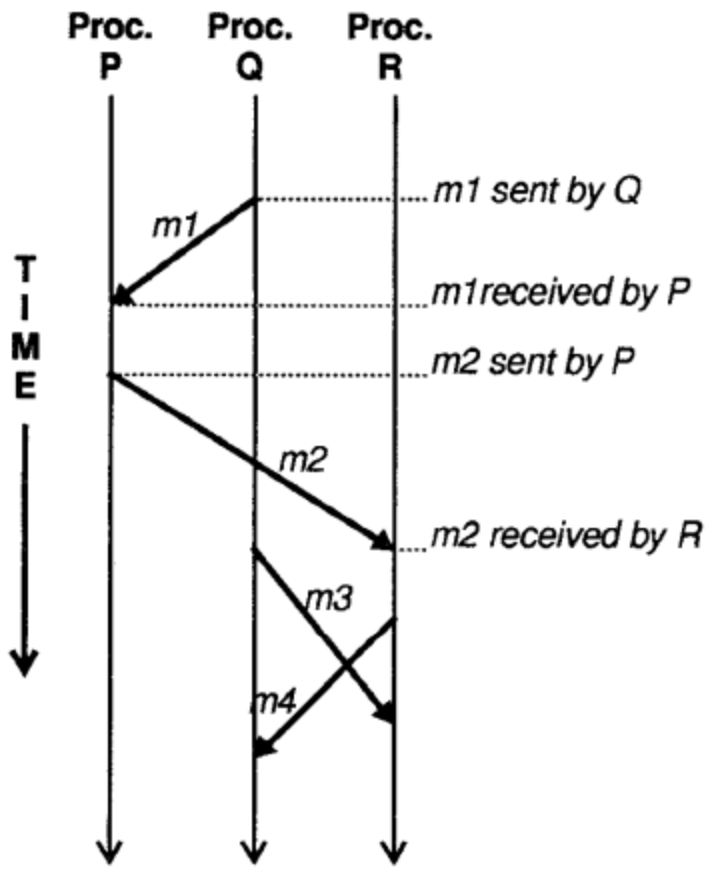
- Serializable ordering, semantic ordering are not ensured



Can't say "*efficiently*"

- No efficiency gain over state-level techniques
- False Causality
- Not scalable
  - Overhead of message reordering
  - Buffering requirements grow quadratically

False Causality



- What if m2 happened to follow m1, but was not causally related?

Discussion

- Virtual Synchrony good!
- But, not perfect