Virtual Synchrony

Scott Phung
Nov 15, 2011

Some slides borrowed from Jared (‘09)

Motivation

- Build Distributed Systems with:
  - Fault-Tolerance
  - Consistency
  - Concurrency
  - Easy programmability

Timeline

Year	**Event**	**Author**
1975	ARPANET	ARPANET
1978	Time, Clocks, and the **Ordering of Events** in a Distributed System	Lamport
1978, 84, 90	**State Machine Replication**	Lamport, Schneider
1981	Database serializability, 2PC, 3PC	Berstein, Goodman, Skeen
1982	Byzantine General's Problem	Lamport, Shostak, Pease
1983	Impossibility of Distributed Consensus with One Faulty Process	Fischer, Lynch, Paterson
1983+	**Virtual Synchrony**	Birman et al
1985	Group Communication primitives, **"process group"** OS construct	Cheriton, Deering, Zwaenepoel
1990	Paxos	Lamport

Source: A History of the Virtual Synchrony Replication Model ('93)

The Process Group Approach to Reliable Distributed Computing ('93)

- Ken Birman
  - Professor, Cornell University
  - Virtual Synchrony / Isis / $Isis^2$
  - Quicksilver
  - Live Object

Assumptions

- Asynchronous communication
- Message Passing
- Fail-Crash Failure Model
  - Timeout suspects stopped or slow processes through
  - Processes considered to have failed
- WAN of LANs

Virtual Synchrony

- Distributed execution model that gives the appearance of synchronous execution
  - Eases program development
  - will talk more later

- Features
  - Process Groups
  - Ordered and Concurrent Message Delivery
  - Reliable Multicast

Motivation

- Build Distributed Systems with:
  - Fault-Tolerance
  - Consistency
  - Concurrency
  - Easy programmability

- How to achieve Fault-Tolerance, Consistency and Easy Programmability? Process Groups.

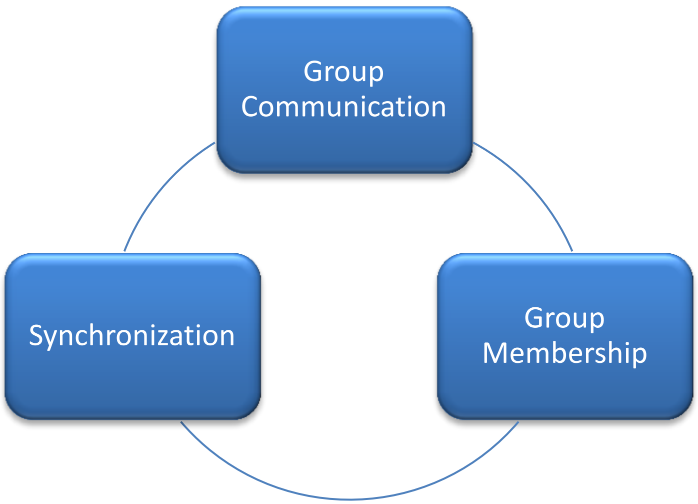
Outline

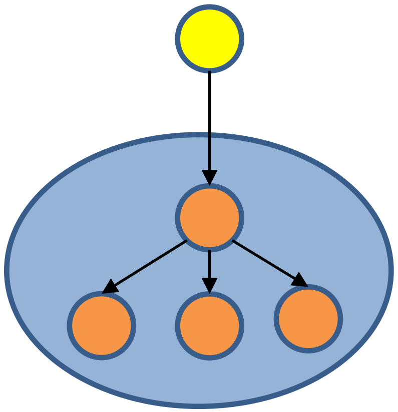
- Problem
  - Process Groups (Implementation)
- Solution
  - Close Synchrony
  - Virtual Synchrony
  - Isis

Process Groups

Communication framework that structures members of a distributed system into groups:

- Provides an easy development framework:

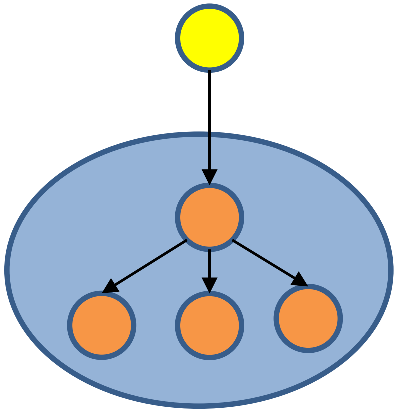




Process Groups

Process Groups provides:

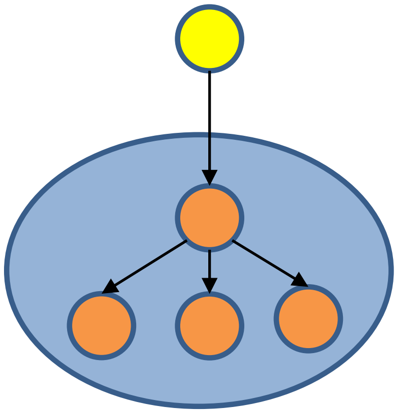
- Fault Tolerance
  - State Machine Replication
- Consistency
  - Membership changes, Message Delivery Order



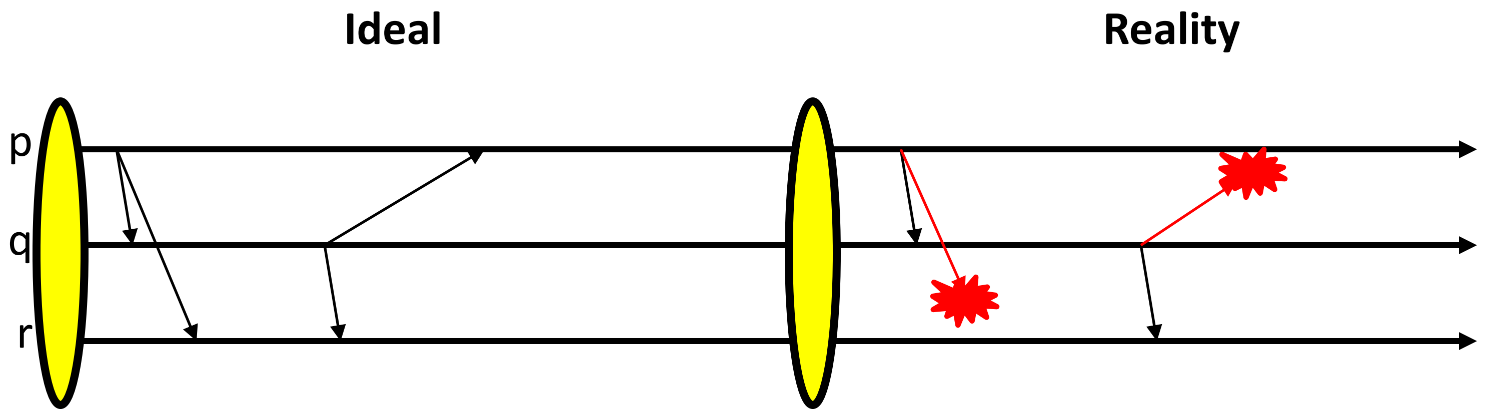
Process Groups Issues

Problems building using Conventional Technologies (UDP, RPC, TCP):

- No reliable multicast (Group Communication)
- Membership churn (Group Membership)
- Message ordering (Synchronization)
- State transfers (Group Membership)
- Failure atomicity (Group Membership)

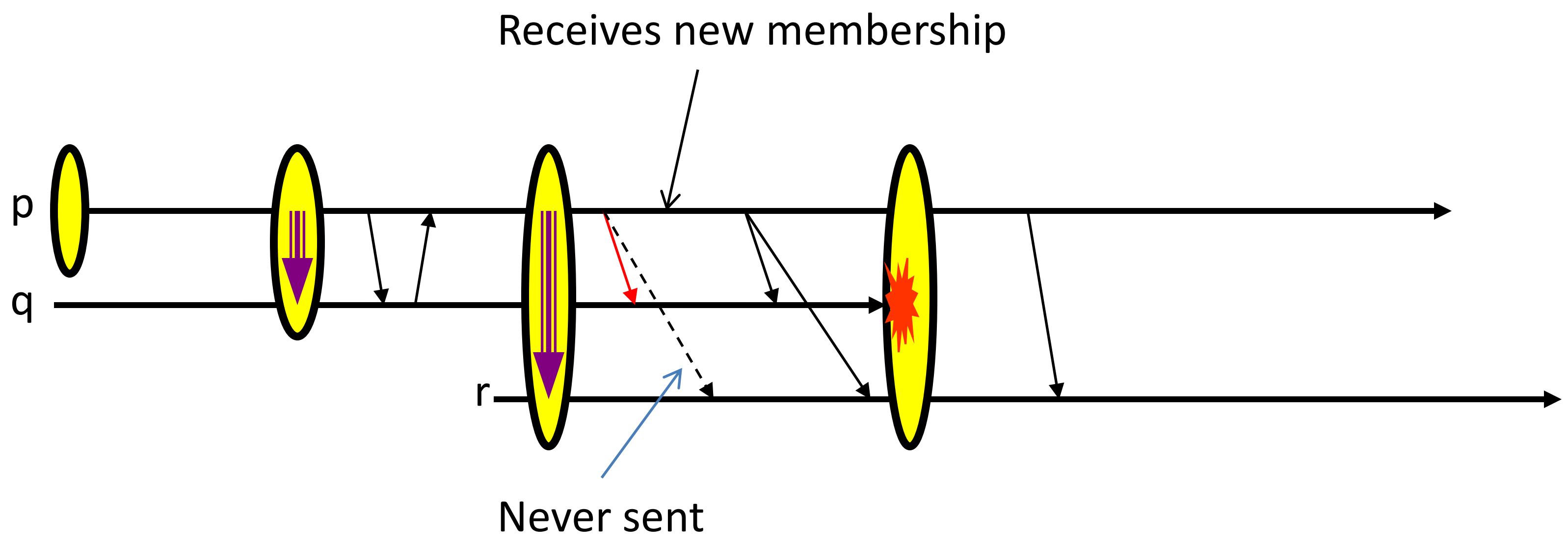


No Reliable Multicast



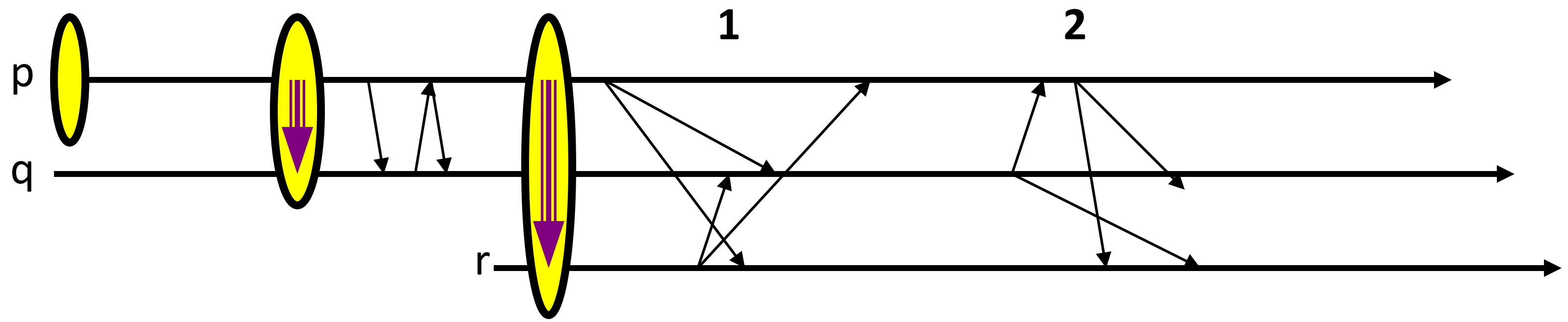
- UDP, TCP, Multicast not good enough
- *What is the correct way to recover?*

Membership Churn



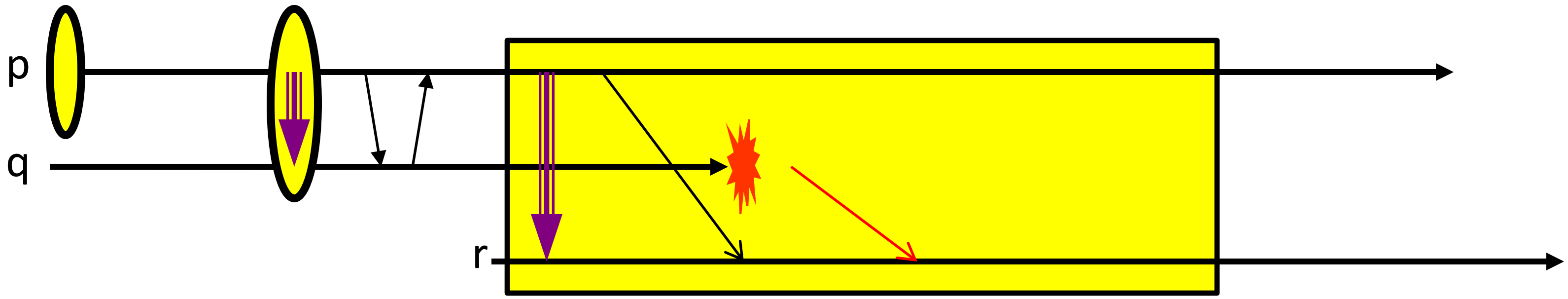
- Membership changes are not instant
- How to handle failure cases?

Message Ordering



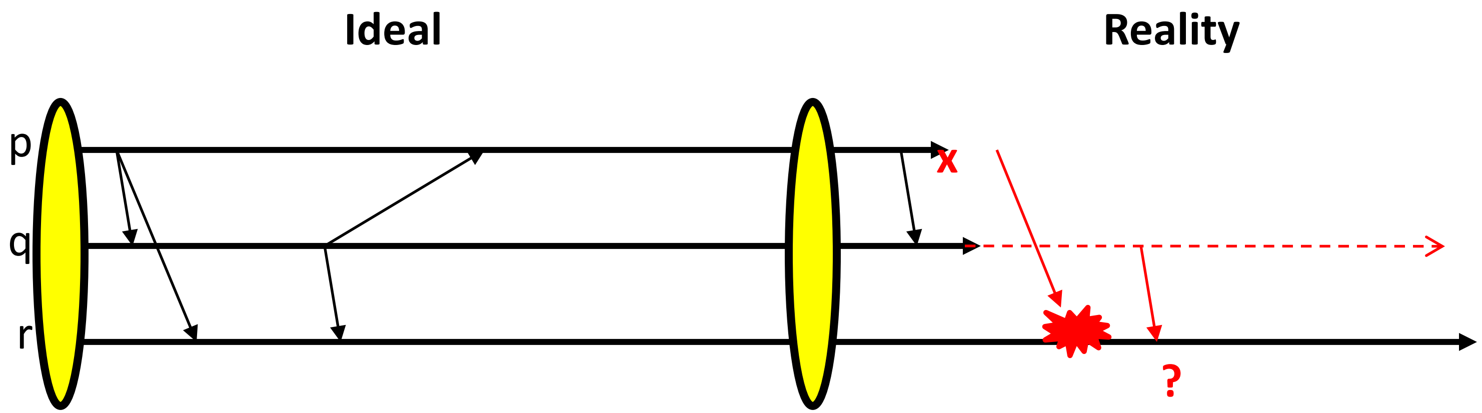
- Lamport's Notion of Time: Causality
- How to prevent causal messages delivered out of order (Ex 2)?

State Transfers



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

Failure Atomicity

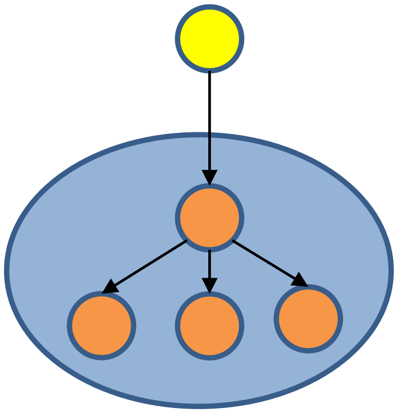


- Nodes can fail mid-transmit
- Some nodes receive message, others do not
- Inconsistencies arise!

Process Groups Issues Recap

Problems building using Conventional Technologies (UDP, RPC, TCP):

- No reliable multicast (Group Communication)
- Membership churn (Group Membership)
- Message ordering (Synchronization)
- State transfers (Group Membership)
- Failure atomicity (Group Membership)



Can we build a system that solves these?

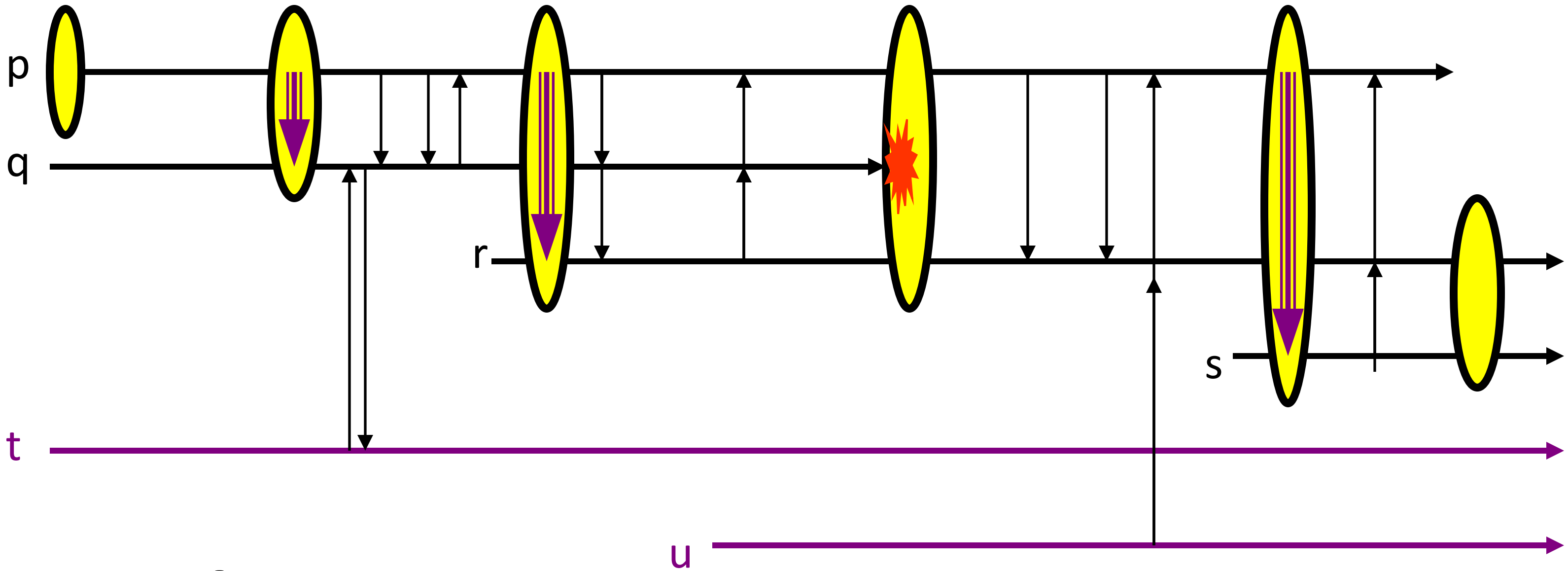
Outline

- Problem
  - Process Groups (Implementation)
- Solution
  - Close Synchrony
  - Virtual Synchrony
  - Isis

Close Synchrony

- Synchronous Execution Model
- Multicast delivered to all group members as a single, reliable ***instantaneous*** event.
  - Solves all Process Group problems!

Close Synchrony



- ***Synchronous*** execution
  - Execution moves in lock-step

Close Synchrony

Process Group problems solved:

- No Reliable Multicast
  - Multicast is always reliable
- Membership Churn
  - Membership is always consistent
- Message Ordering
  - Totally ordered message delivery
- State Transfers
  - State-transfer happens *instantaneously*
- Failure Atomicity
  - Multicast is a *single event*

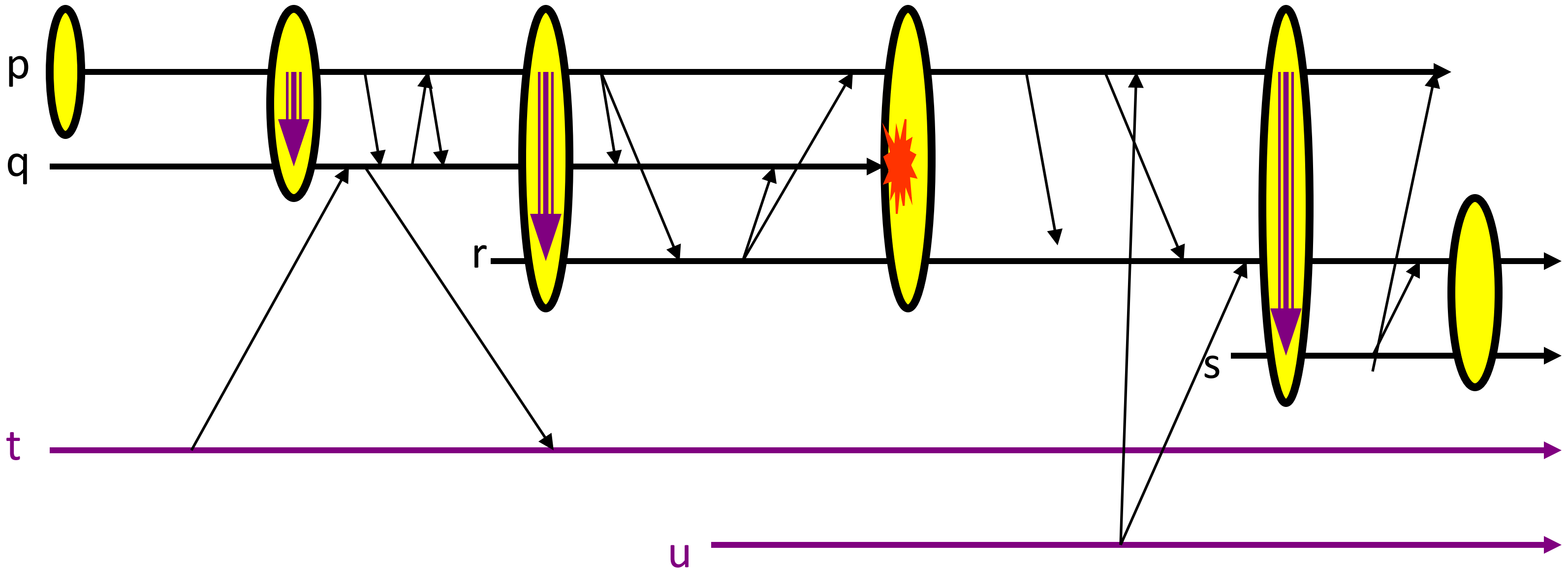
Close Synchrony

Problem

- We don't have instantaneous events
- It is impossible in the presence of failures
- Expensive (waits for slowest member)

What can we do?

Asynchronous Execution



Virtual Synchrony

Close Synchrony using Asynchronous protocols

Group Communication

- Notion of time: Use Lamport's Happens-Before relationship
- Causal & Concurrent Ordered Message Delivery (CBCAST)
- This causal order matches some equivalent Close Synchronous execution (total order).

Group Membership

- Synchronized Membership View Changes
- Replicated Group Membership Service sends final word on failures & joins to all members

Causal Message Ordering

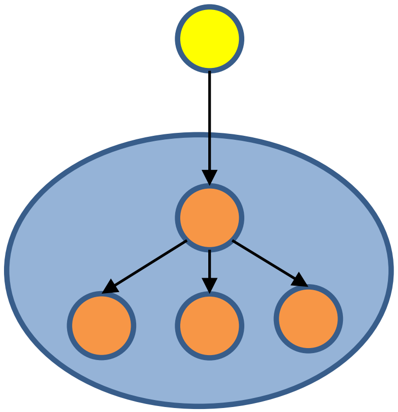
- CBCAST (Casual Atomic Broadcast Primitive)
- Asynchronous, **fast**
- Causal Order Delivery (within group)
  - Vector clock, delay of messages
- Concurrent messages can be delivered OOO
- Batch multiple messages
- Most-used primitive in Virtual Synchrony

Total Message Ordering

- ABCAST (Atomic Broadcast Primitive)
- Synchronous, slow
- Total Order Delivery (within a group)
- No message can be delivered to any user until all previous ABCAST messages have been delivered

Distributed Algorithms

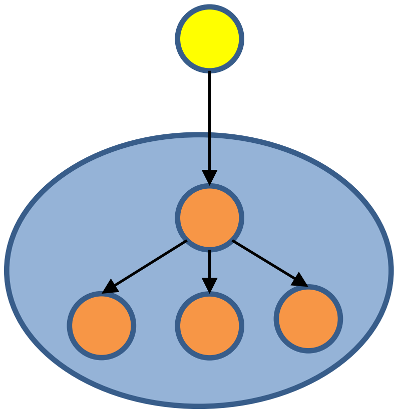
- How can Process Groups solve Consensus?



From Ken's Slides - 2006

Distributed Algorithms

- How can Process Groups perform Distributed Snapshots?



Isis

- Framework that offers Group communication with Virtual Synchrony
- Takes care of group communication, membership changes and failures through a single, event oriented execution model (Virtual Synchrony).
- You just concentrate on the member code!

Isis

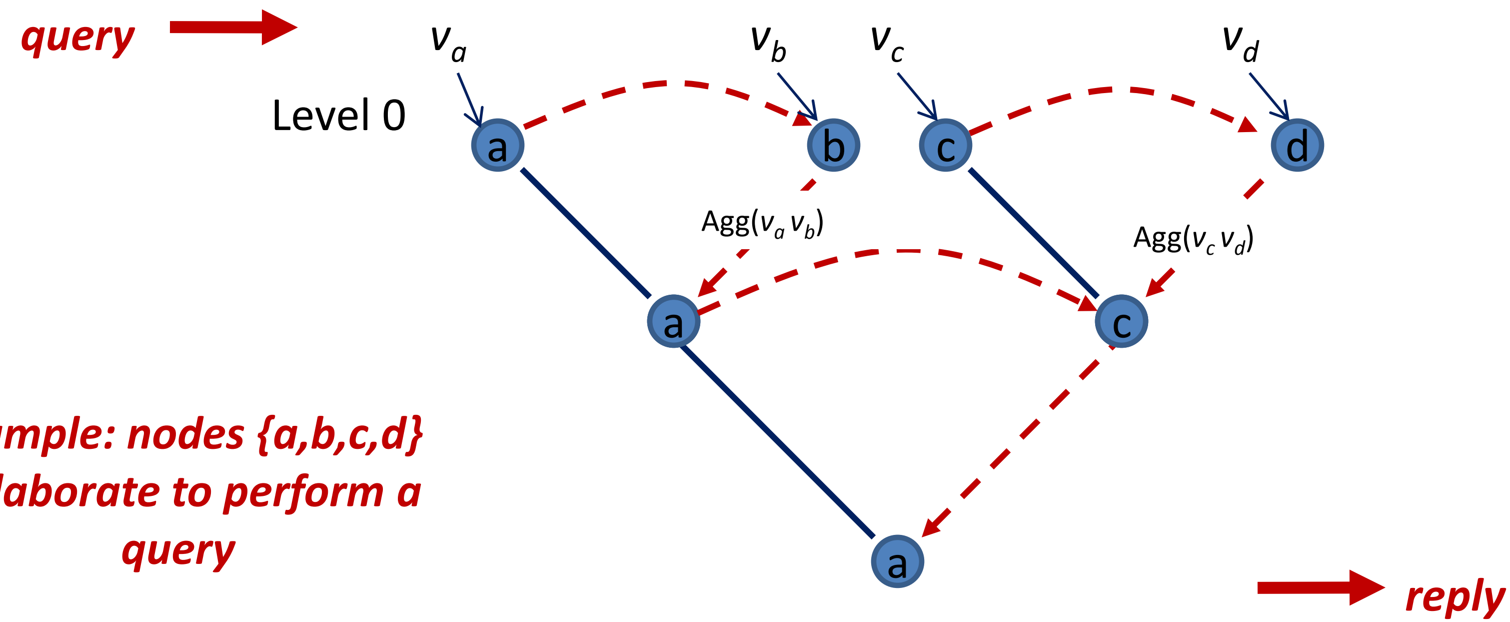
- Used In:
  - NYSE, Swiss Stock Exchange
  - French Air Traffic Control System
  - US Navy AEGIS

Isis - Weakness

- Large Groups - Multicast reply explosion
  - Isis^2 Group Aggregation, Dr. Multicast
- No reduction ability within Groups
  - Isis^2 Group Aggregation
- Messages sent are not durable
  - Isis^2 SafeSend (Paxos Mode)

Isis² Group Aggregation

- Used if group is *really big*
- Request, updates: still via multicast
- *Response* is aggregated within a tree



Example: nodes {a,b,c,d} collaborate to perform a query

Takeaways

- Virtual Synchrony Benefits
  - Group Communication, Membership Changes, State Transfers and Failures in a single event execution model (Close Synchrony)
- Key Contributions
  - Dynamic Group Membership
  - Integration of Failure detection into communication subsystems
  - Ordered and Total Message Delivery

Understanding the Limitations of Causally and Totally Ordered Communication ('93)

- David Cheriton
  - Professor, Stanford
  - PhD – Waterloo
  - V Operating System

- Dale Skeen
  - PhD – UC Berkeley, former Cornell Assistant Prof.
  - Distributed pub/sub communication, 3PC
  - Co-founded TIBCO, Vitria

CATOCS Problems

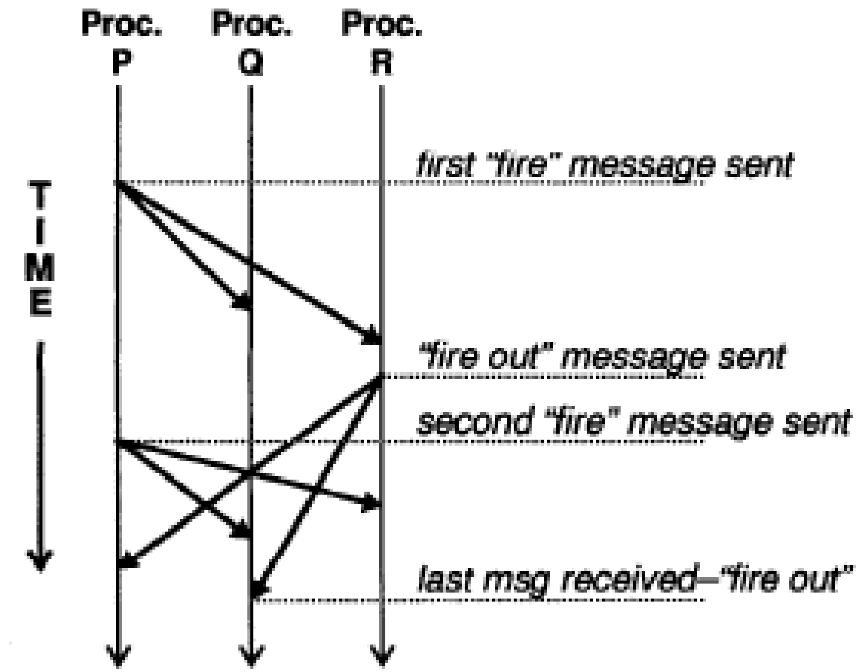
- Causal And Totally Ordered Communication Support
- Message delivery is atomic, but not *durable*
- Incidental ordering
  - CATOCS is at communication level but consistency requirements are at application state
- Violates end-to-end argument.

Limitations of CATOCS in communication layer

- Unrecognized Causality
  - Can’t say “*for sure*”
- No Semantic Ordering
  - Can’t say the “*whole story*”
- Lack of serialization ability
  - Can’t say “*together*”
- Lack of Efficiency Gain over State-level Techniques
  - Can’t say “*efficiently*”

Unrecognized Causality
Can't say "*for sure*"

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

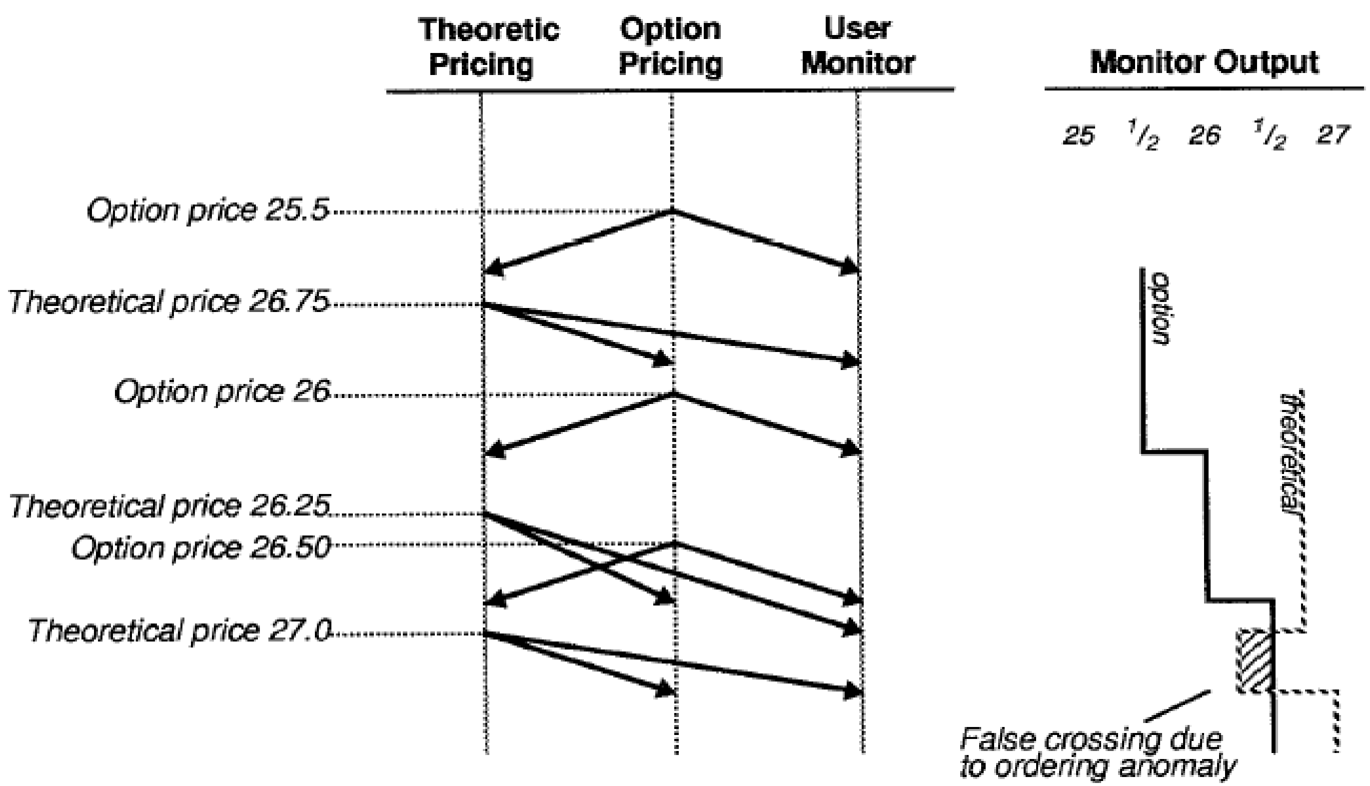


Can’t say “*together*”

- Serializable ordering, cannot order a group of messages together
  - Seems to only provide shared-memory w/lock examples, do other Message Passing systems offer serializable ordering?

Can’t say “whole story”

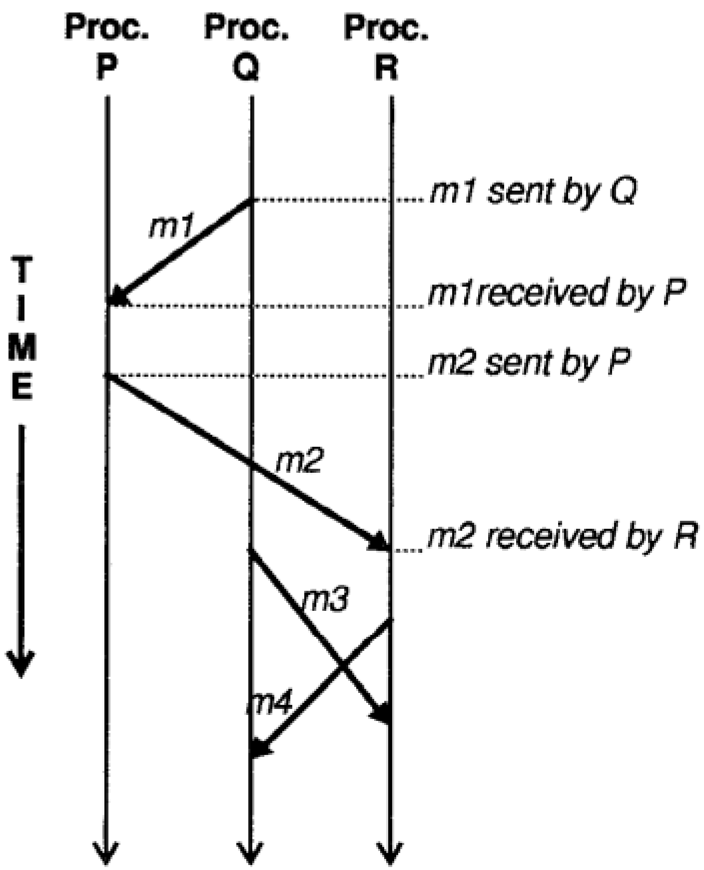
- Semantic ordering are not ensured



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?

Birman's Response ('93)

- Ordering is important to guarantee consistency
  - when **combined** with an Execution model (Virtual Synchrony) produces a system with powerful reliability guarantees.
  - This point was completely neglected.
- Causal ordering
  - is cheap and prevents some failures.
  - flow control and congestion handling more important.
- Hidden Channels
  - Rare, mostly in Shared Memory, which you protect with a lock.
  - No system can say for sure for the example constructed.

Birman’s Response (‘93)

- Semantic vs Causal Ordering
  - Causal order provides some ordering guarantees.
  - Tag with timestamps or create causal dependency from theoretical price to actual price.
- Can Say “efficiently”
  - Buffering requirements do not grow quadratic, they are usu. constant.
  - VS is efficient, otherwise leave group membership, communication, synchronization to application developer ==> less efficient system
- Theoretical Proofs carry little weight in this domain
  - FLP, yet systems are still built that solve consensus.
  - 3PC, yet most DB systems use 2PC.