DISTRIBUTED SYSTEMS: GROUP COMMUNICATION

CS6410

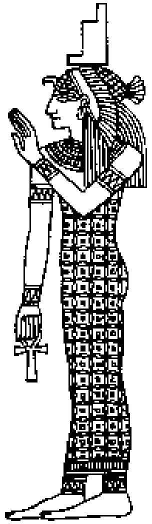
Hakim Weatherspoon

Slides borrowed liberally from past presentations from Julia Proft, Utkarsh Mall, Scott Phung, and Jared Cantwell

The Process Group Approach to Reliable Distributed Computing

Communications of the ACM, Dec. 1993

Ken Birman, Cornell University



Reviews a decade of research on the Isis system.

> By naming our system ‘The Isis Toolkit’ we wanted to evoke this very old image of something that picks up the pieces and restores a computing system to life.

Timeline

Year	Event	Contributor(s)
1978	Time, Clocks, and the Ordering of Events in a Distributed System	Lamport
1982	Byzantine Generals Problem	Lamport, Shostak, and Pease
1983	Impossibility of Distributed Fault Tolerant Consen	Fischer, Lynch, and Patterson
1983	Virtual Synchrony and the Isis Toolkit	Birman et al.
1984	State Machine Replication	Lamport, Schneider
1985	Distributed Process Groups (V System)	Cheriton, Deering, and Zwaenepoel
1987-1993	Bulk of development on the Isis Toolkit	Birman et al.

Motivation

Problem: the construction of **reliable distributed software.**

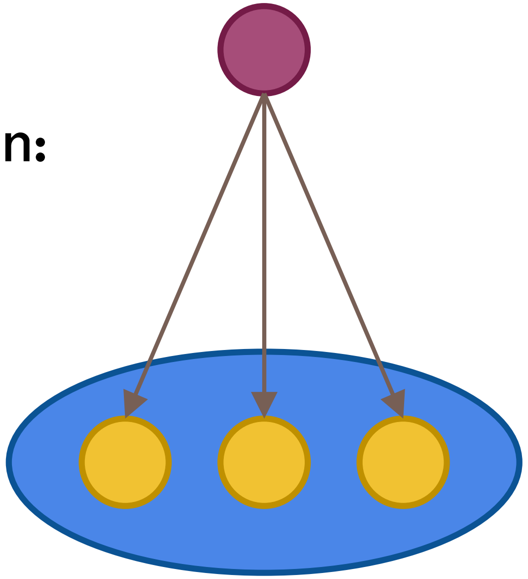
- Issues of reliability have been left to the application programmers, who are "largely unable to respond to the challenge"; solutions to the problems are "probably beyond the ability of a typical distributed applications programmer."

Solution: programming with **distributed groups of cooperating programs,** implemented in the computing environment itself or the operating system.

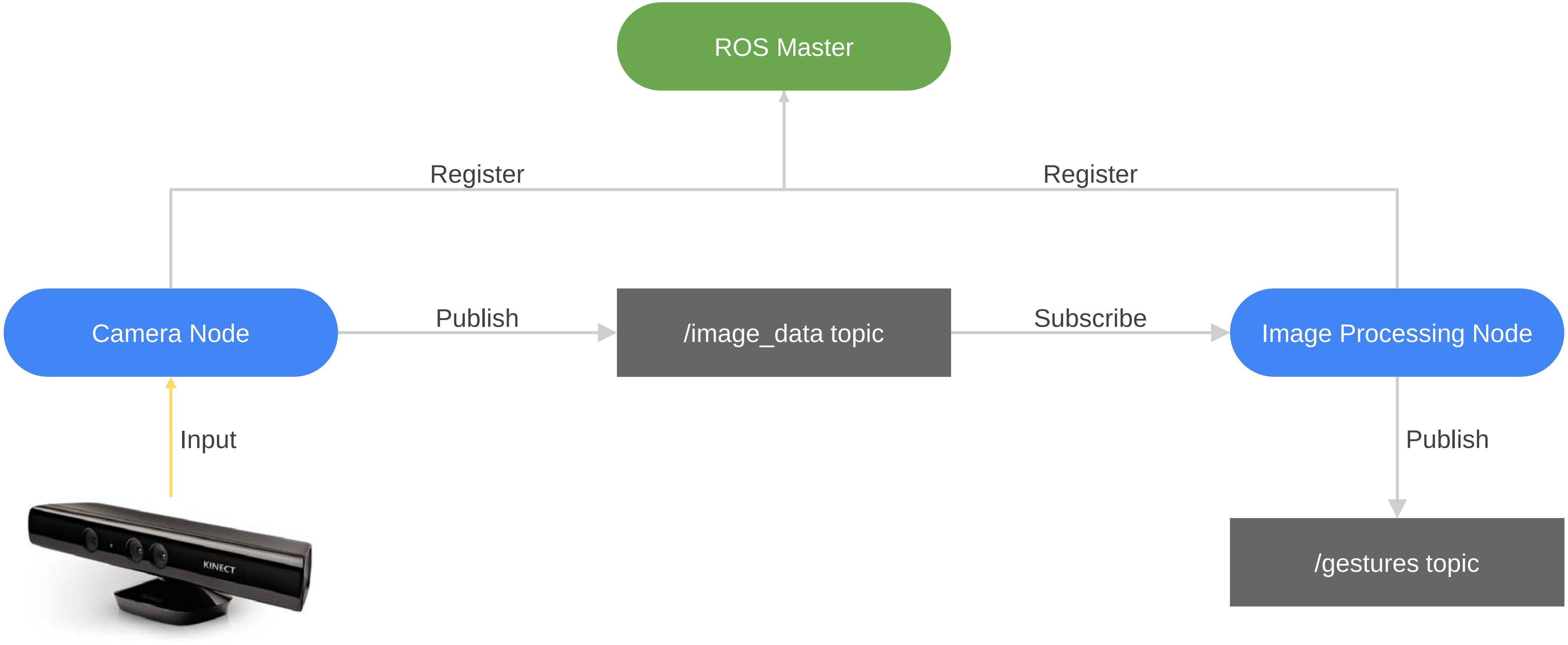
- *"The only practical approach"!*

Process Groups

- Anonymous groups
  - Application *publishes* data to a *topic*
  - Other processes *subscribe* to this topic
  - Properties needed for automatic, reliable operation:
    - Ability to address group
    - Atomic message delivery
    - Ordered message delivery
    - Access to history of group
- Explicit groups
  - Direct cooperation between members
  - Share responsibility for responding to requests
  - Membership changes published to the group



Example: the Robot Operating System (ROS)



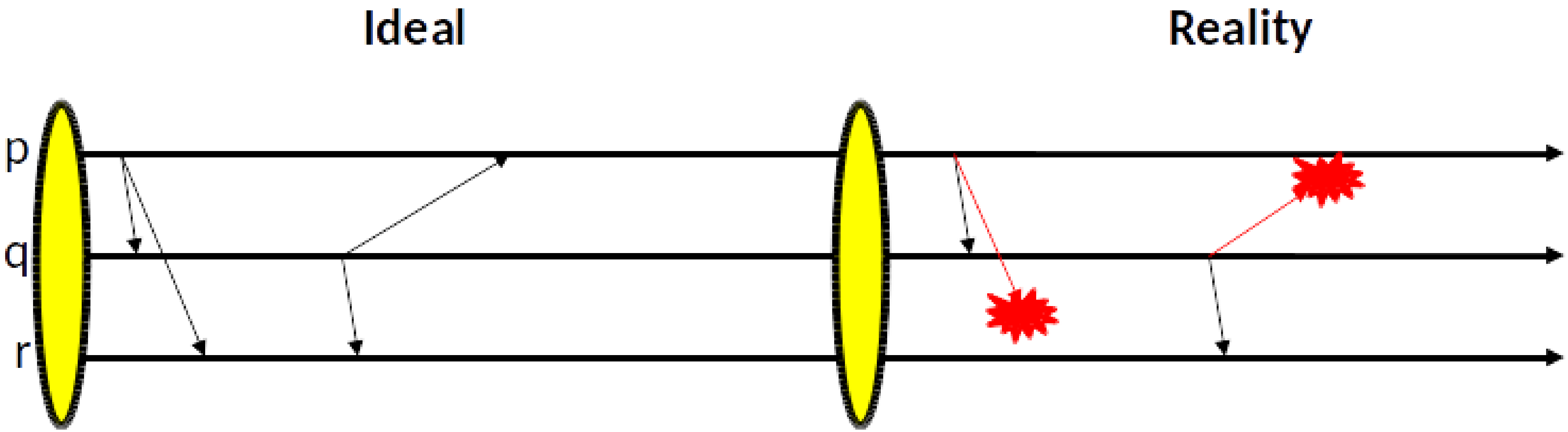
Advantages

- Fault tolerance
  - Transparent adaptation to failure and recovery
  - State machine replication
- Consistency
  - Ordered and atomic message delivery
  - Consistent view of group membership
- Ease of development
  - Need not worry about communication protocol
  - Leave fault tolerance and consistency to the OS

Problems

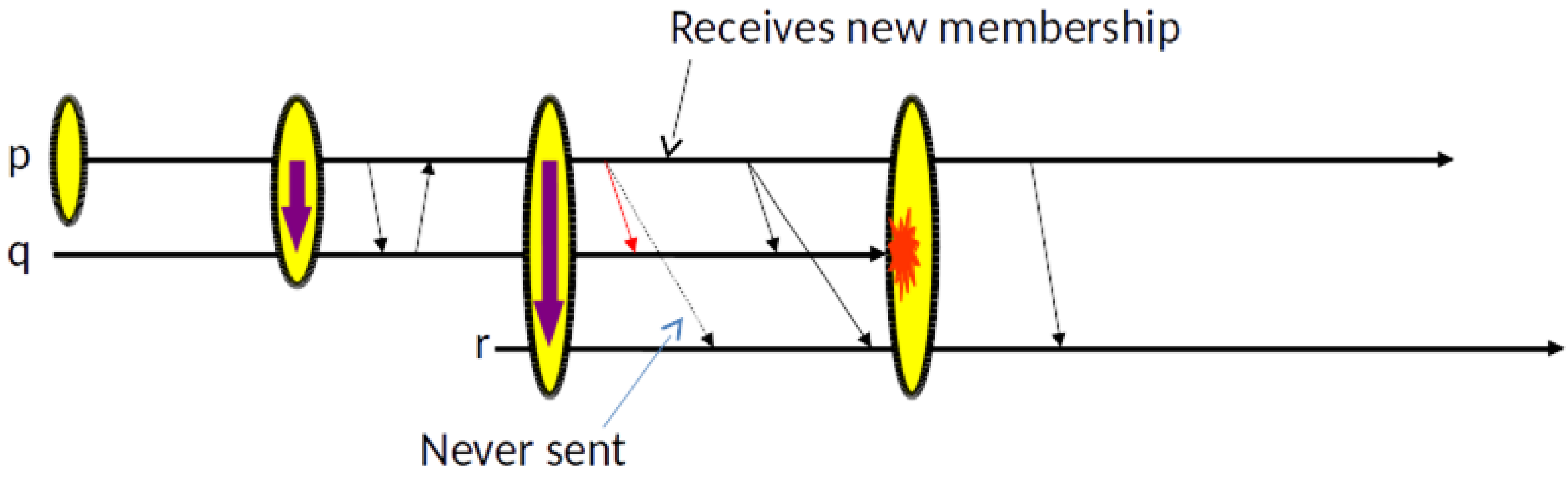
- Unreliable communication
- Membership changes
- Delivery ordering
- State transfer
- Failure atomicity

Unreliable communication



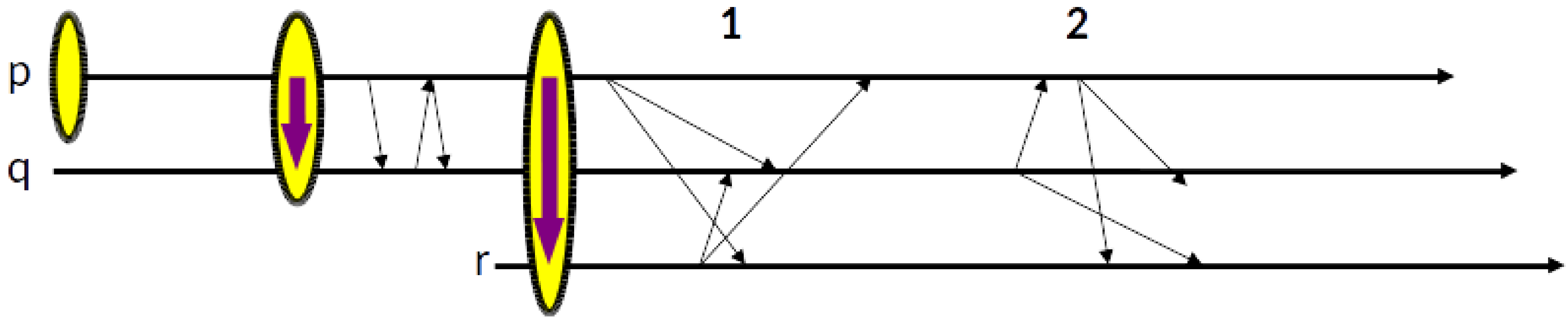
- UDP: packets lost, duplicated, delivered out of order
- RPC: sender cannot distinguish reason for failure
- TCP: broken channels result in inconsistent behavior
- How to recover consistently from message loss?

Membership changes



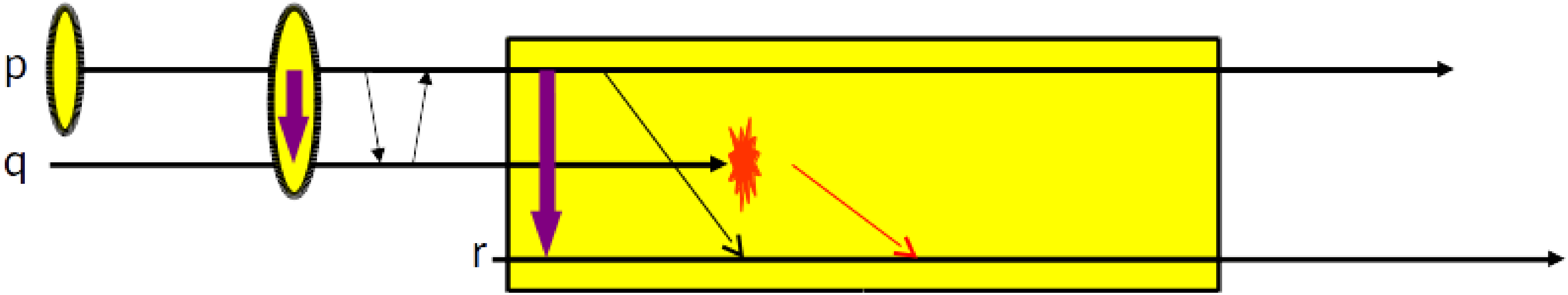
- Group membership changes do not happen instantaneously
- How to make sure messages reach the latest group members?

Delivery ordering



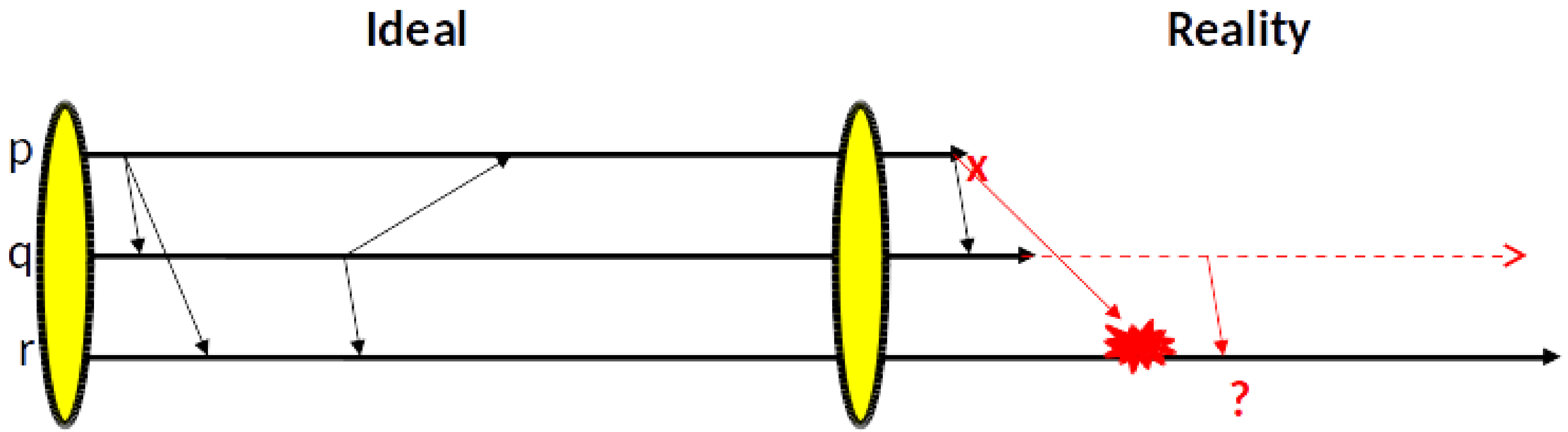
- Messages need to be ordered by causality
- How to deliver in causal ordering?

State transfer



- Processes joining group must get latest state
- How to handle inconsistencies from concurrent messages?

Failure atomicity



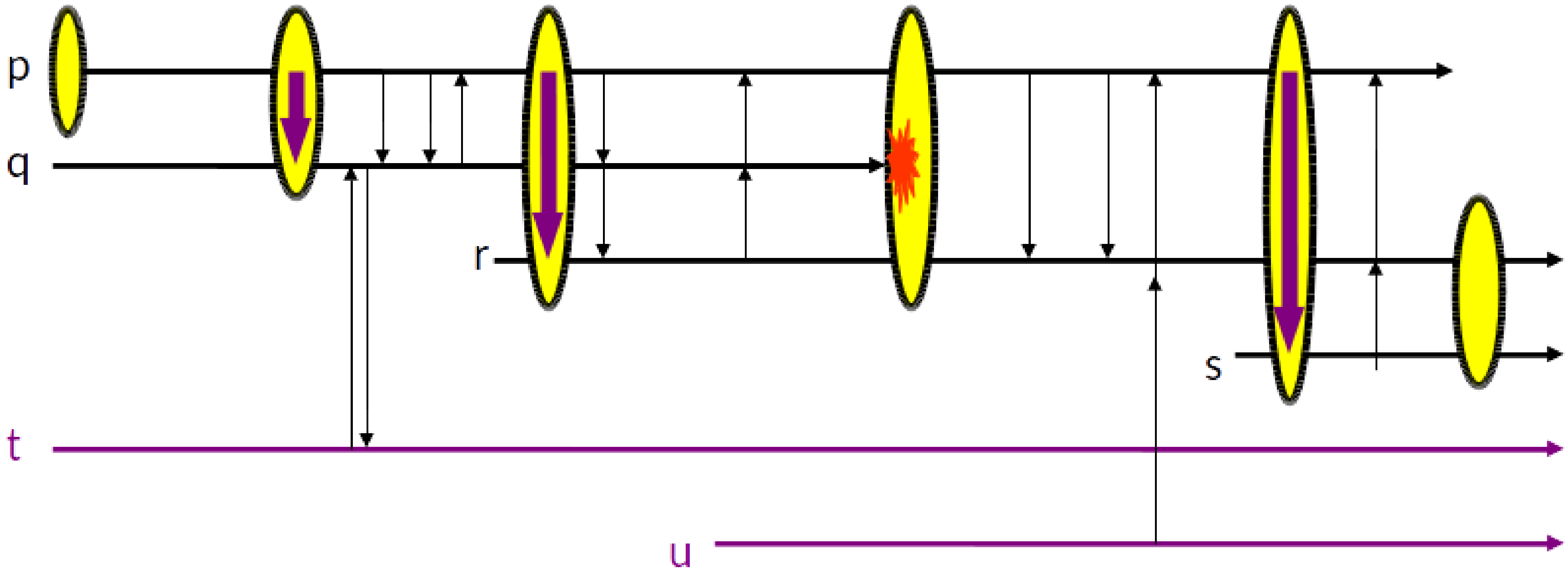
- Need to achieve all-or-nothing message delivery
- How to handle mid-transmission failures?

Close Synchrony

A **synchronous** execution model.

- *Multicasts* to a process group are delivered to all members
- Send and delivery events occur as a single, instantaneous event

Close Synchrony



- Execution runs in genuine lockstep.

Close Synchrony

- Unreliable Communication
  - Multicast is always reliable
- Membership changes
  - Consistent membership at any logical instant
- Delivery Ordering
  - Concurrent multicasts are distinct events
- State Transfer
  - Happens instantaneously
- Failure Atomicity
  - Multicast is a single logical event

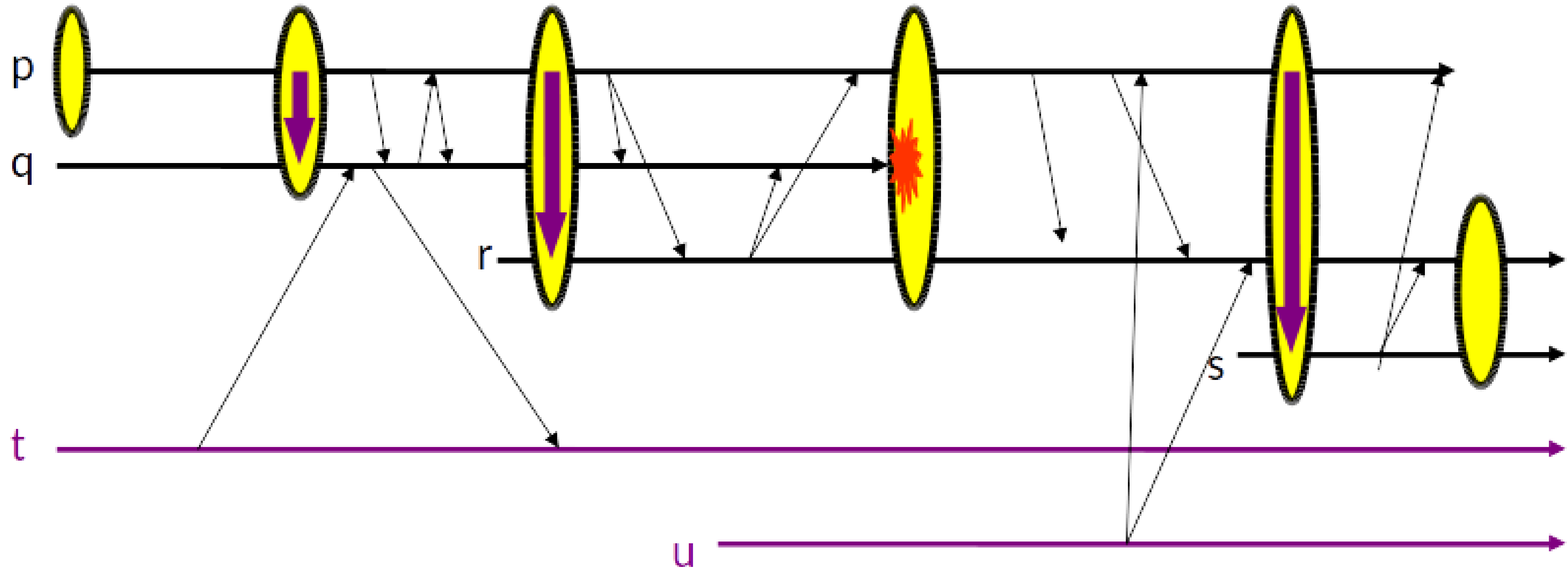
Problems with Close Synchrony

- In the real world, events are not instantaneous!
- Expensive: execution runs in genuine lockstep!
- Impossible to achieve in presence of failures (why?)

What do we do?

Virtual Synchrony

- **Asynchronous** Close Synchrony
- Synchronization needed only for events sensitive to ordering



Virtual Synchrony

- Group Membership Service
  - Replicated service within the process group itself
  - Membership change needs to be done synchronously
- Group Communication Service
  - Uses Lamport's happened before relationship
  - CBcast (Causal Broadcast) or ABcast (Atomic Broadcast)
  - Multicasts are going to be a total event ordering equivalent to some close synchrony execution

Vector Clocks

- Array of clocks, indexed by processes in the process group
- Protocol:
  - $VT(p_i)$ = clock maintained by process p_i
  - $VT(p_i)$ initialized to zero
  - For each send(m) at p_i, $VT(p_i)[i] += 1$ and $VT(m) = VT(p_i)$
  - If p_j delivers a message, received from p_i:
    - For k in 1..n: $VT(p_j)[k] = \max(VT(m)[k], VT(p_i)[k])$
- Ordering
  - $VT_1 \leq VT_2$ iff $\forall i$, $VT_1[i] \leq VT_2[i]$
  - $VT_1 < VT_2$ iff $VT_1 \leq VT_2$ and $\exists i$, $VT_1[i] < VT_2[i]$

CBcast

- Uses vector clocks to detect causality
- Delivery of received messages delayed until "happened before" messages are delivered
- Protocol:
  - p_j on receiving message m from p_i, delays delivery until
    - $VT(m)[k] = VT(p_j)[k]+1$ if k=i
    - $VT(m)[k] \leq VT(p_j)[k]$ otherwise
  - When m is delivered follow vector clock protocol
- Delayed messages stored in CBcast delay queue
- Concurrent messages delivered out of order
- Fast because asynchronous

ABcast

- Stronger ordering guarantee than CBcast
- Total message ordering within a group
- Messages can only be delivered if, no prior ABcast is undelivered
- Slow
- Protocol:
  - A process p_i holding token CBcasts message
  - If p_i is not holding the token
    - CBcast but mark undeliverable
    - Token holder delivers and CBcasts a set-order
    - Other follow the set-order

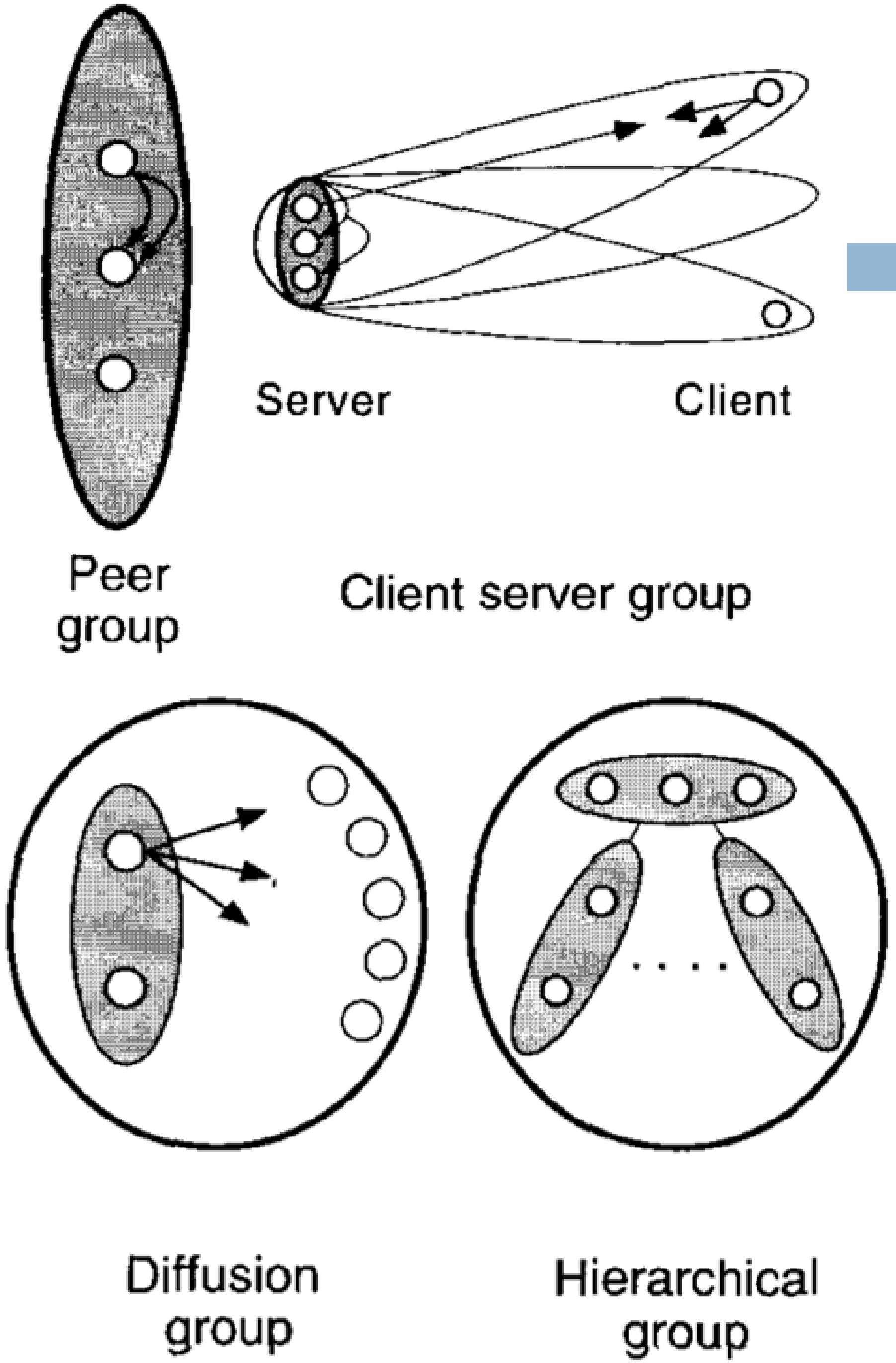
Virtual Synchrony

- Unreliable Communication
  - Group communication service
- Membership changes
  - Group membership service
- Delivery Ordering
  - ABcast, CBcast
- State Transfer
  - Group membership service
- Failure Atomicity
  - Group communication service, group membership service

Isis

An implementation of virtual synchrony

- Used by
  - New York/Swiss stock exchange
  - French air traffic control system (PHIDIAS)
- Also provides
  - monitoring facilities: site failures, triggers
  - Automated recovery
  - Styles of group



Discussion Questions

- How is virtual synchrony with ABcast different from close synchrony?

Takeaways

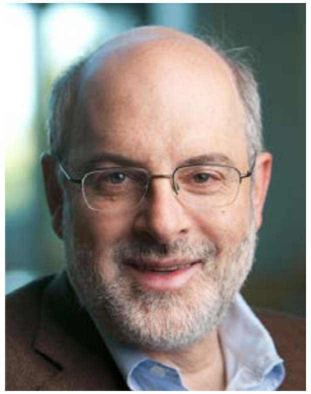
Close synchrony with process groups provides:

- Ease of development
- Consistency
- Fault tolerance

Virtual synchrony:

- Faster asynchronous system

Bimodal Multicast (1999)



Ken Birman
PhD Berkeley ‘81
→ Cornell University

Mark Hayden
PhD Cornell ‘98
→ Compaq Research
→ North Fork Networks
→ Lefthand Networks
→ Ventura Networks

Öznur Özkasap
PhD Ege ‘00
→ Koç University

Spent two years (and completed dissertation) at Cornell

Zhen Xiao
PhD Cornell ‘01
→ AT&T Research
→ IBM Research
→ Peking University

Mihai Budiu
PhD CMU ‘03
→ Microsoft Research
→ Barefoot Networks
→ VMware Research

Spent a year at Cornell

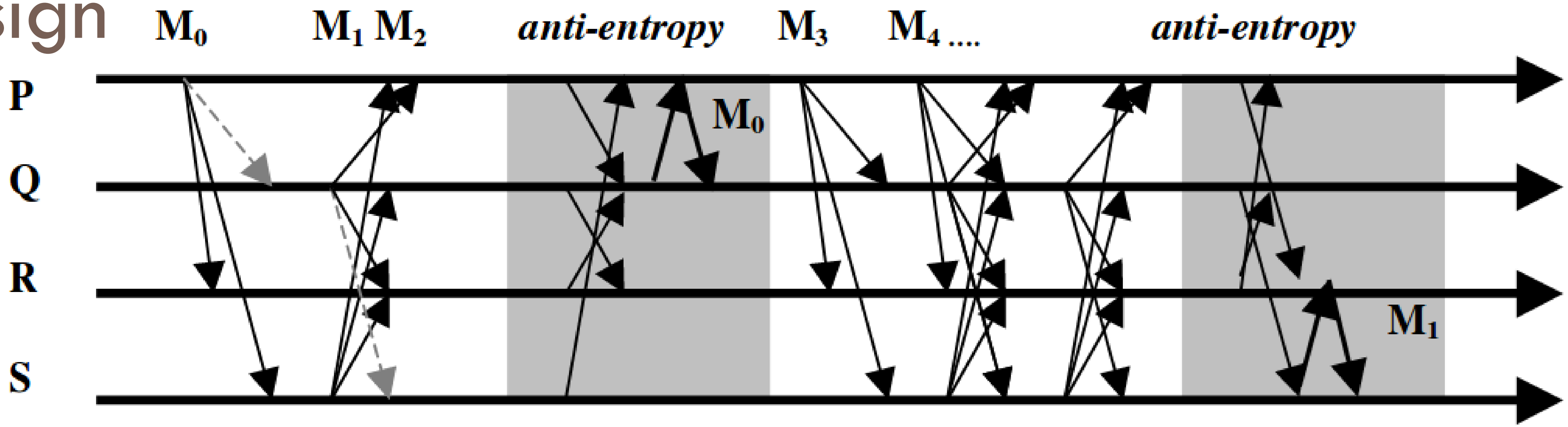
Yaron Minsky
PhD Cornell ‘02
→ Jane Street

Fun fact: introduced Jane Street to OCaml

Motivation

- Virtual synchrony
  - Costly protocol
  - Unstable under stress
  - Not scalable
- Best effort reliability protocols
  - Scalable
  - Starts re-multicasting under low levels of noise
  - No membership check
  - No end-to-end guarantee
- Multicast with stable throughput
  - e.g. Streaming Media, teleconferencing

Design

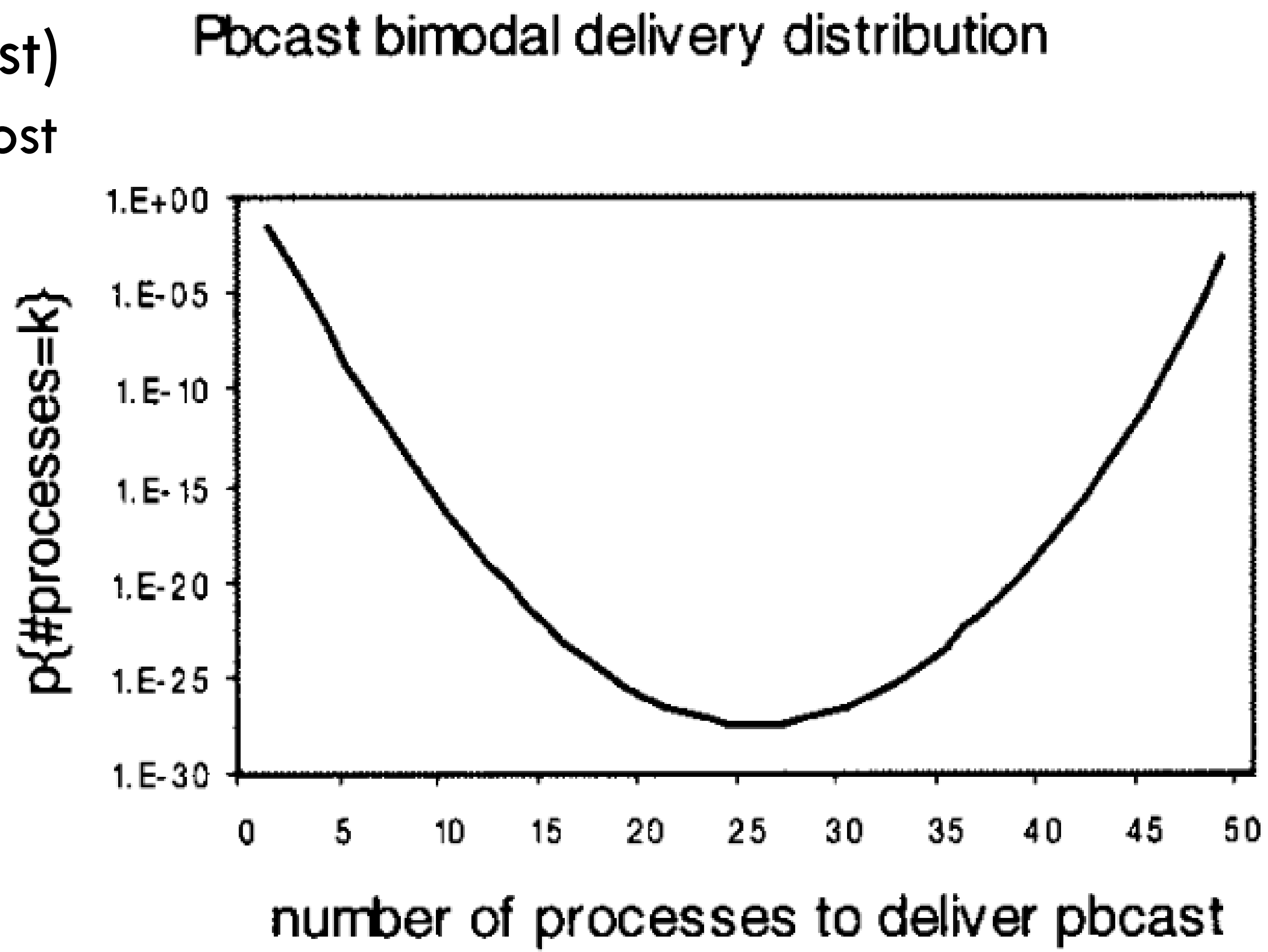


Two step protocol

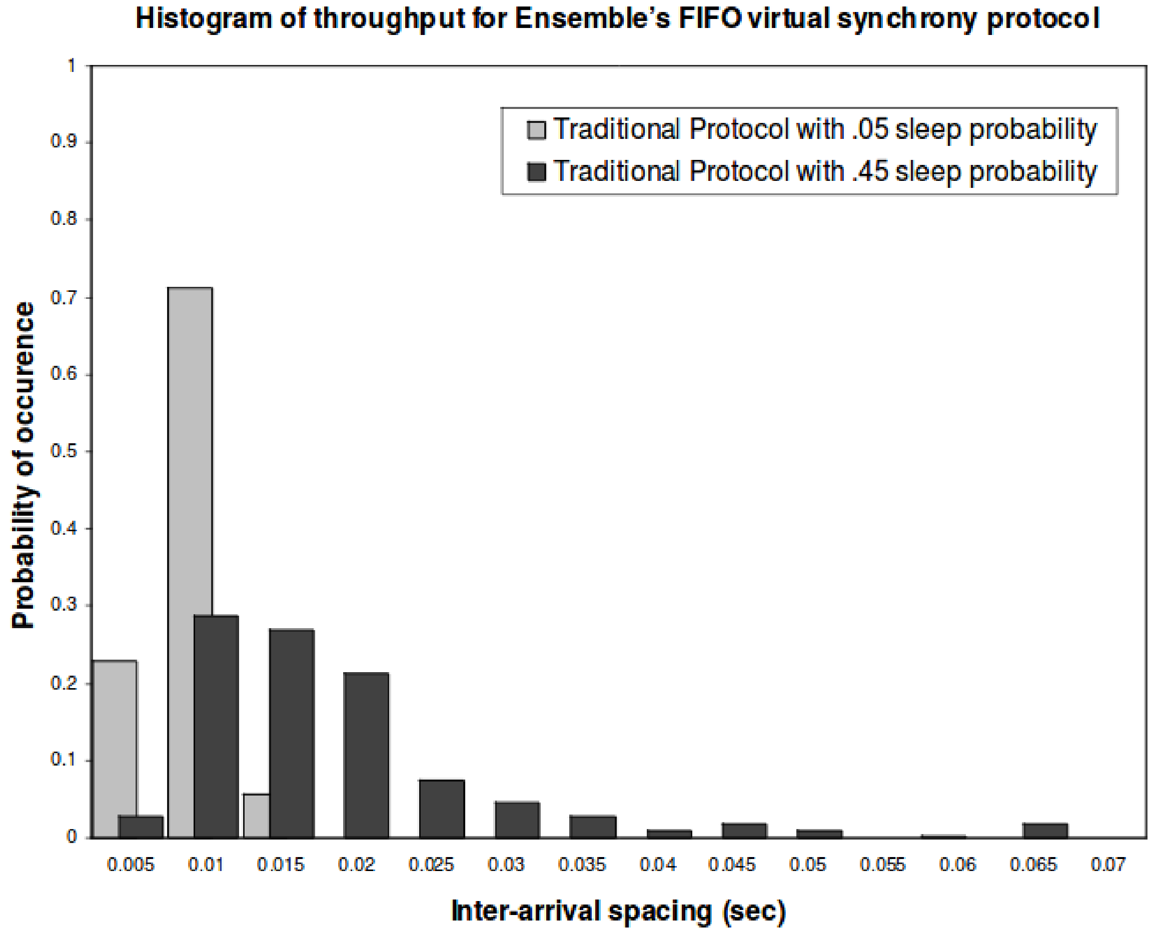
1. Optimistic Dissemination Protocol
   - Unreliable Multicast like IP multicast
2. Two-Phase Anti-Entropy Protocol
   - Random gossip
   - Unicast lost messages
   - Cheaper than re-multicasting

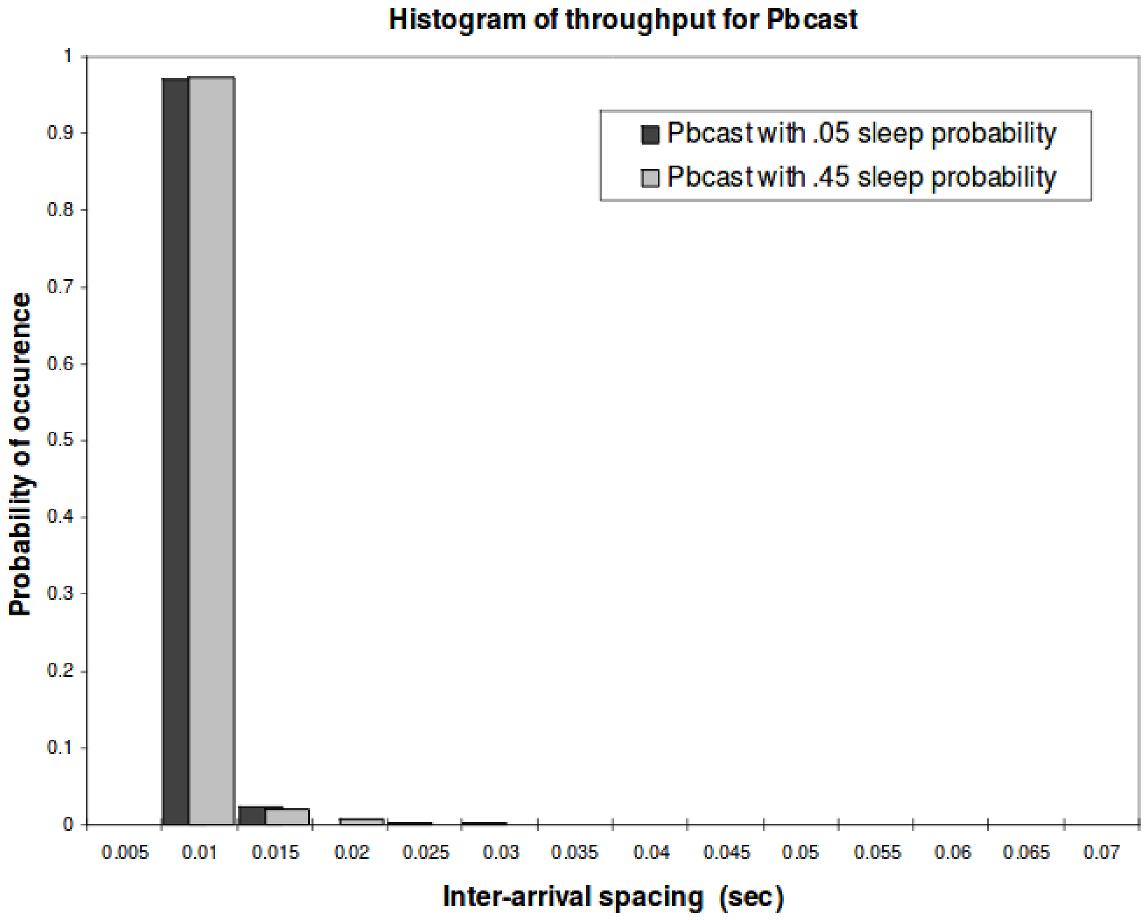
Advantages

- PBcast (Probabilistic Broadcast)
  - Atomicity (Almost all or almost none)
  - Scalability
  - Throughput Stability

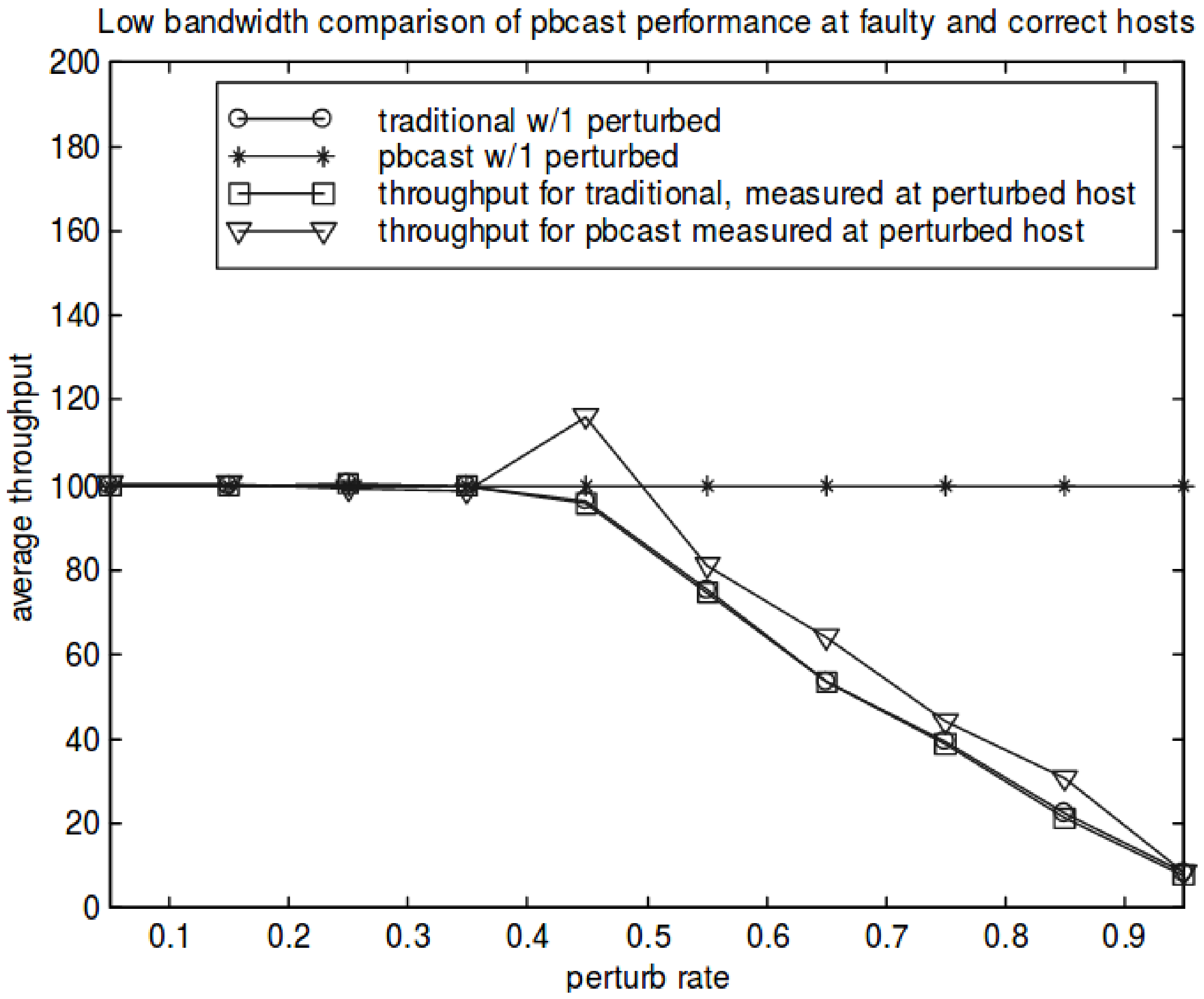


Performance





Performance



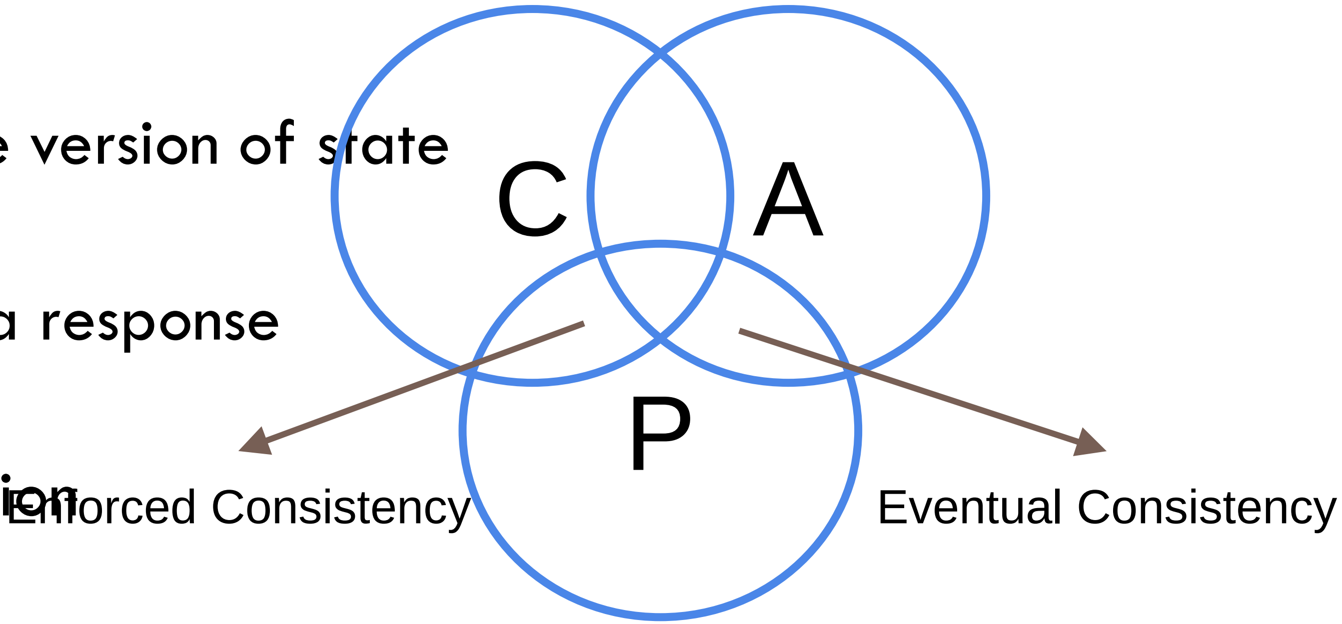
Takeaways

Bimodal Multicast

- Stable throughput
- Scalability at cost of “weaker” reliability
- Predictable reliability
- Predictable load

CAP Conjecture

- Consistency
  - Client receives the latest the version of state
- Availability
  - Client request always gets a response
- Partition Tolerance
  - Can tolerate network partition



- In presence of partition, choose a trade-off between Consistency and Availability.

Acknowledgments

Many slides/diagrams borrowed from Julia Proft and Utkarsh Mall, CS 6410 Fall 2017, Scott Phug, CS 6410 Fall 2011, Ken Birman, CS 614 Fall 2006

Vector Clock, CBcast and ABcast borrowed from Birman, Schiper, Stephenson, *Lightweight causal and atomic group multicast*, 1991