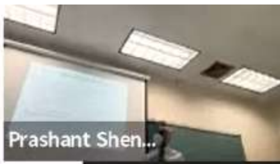
Agreement in Faulty Systems

- How should processes agree on results of a computation?
- *K-fault tolerant*: system can survive k faults and yet function
- Assume processes fail silently
  - Need (k+1) redundancy to tolerant k faults
- *Byzantine failures*: processes run even if sick
  - Produce erroneous, random or malicious replies
    - Byzantine failures are most difficult to deal with
  - Need ? Redundancy to handle Byzantine faults

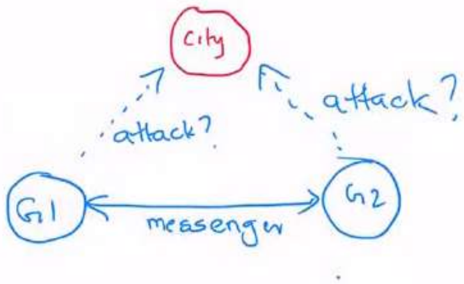
Byzantine Faults



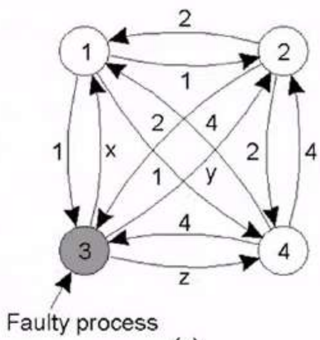
- Simplified scenario: two perfect processes with unreliable channel
  - Need to reach agreement on a 1 bit message
- **Two Generals Problem:** Two armies waiting to attack
  - Each army coordinates with a messenger
  - Messenger can be captured by the hostile army
  - Can generals reach agreement?
  - Property: Two perfect process can never reach agreement in presence of unreliable channel
  - Concept of **Common knowledge**
- **Byzantine generals problem**: Can N generals reach agreement with a perfect channel?
  - M generals out of N may be traitors

2:38 PM Mon Apr 4

Lecture 18 notes



Byzantine Generals Problem



(a)

1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

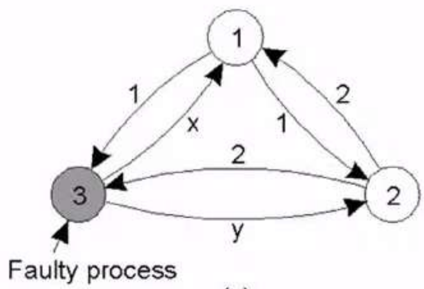
- Recursive algorithm by Lamport
- The Byzantine generals problem for 3 loyal generals and 1 traitor.

a) The generals announce their troop strengths (in units of 1 kilosoldiers).
b) The vectors that each general assembles based on (a)
c) The vectors that each general receives in step 3.

UMassAmherst

Powered by Zoom

Byzantine Generals Problem Exampl



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

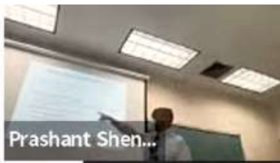
(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

- The same as in previous slide, except now with 2 loyal generals and one traitor.
- Property: With m faulty processes, agreement is possible only if $2m+1$ processes function correctly out of $3m+1$ total processes. [Lamport 82]
  - Need more than two-thirds processes to function correctly (for m=1, 3 out of 4 processes)

Byzantine Fault Tolerance



- Detecting a faulty process is easier
  - 2k+1 to detect k faults
- Reaching agreement is harder
  - Need 3k+1 processes (2/3rd majority needed to eliminate the faulty processes)
- Implications on real systems:
  - How many replicas?
  - Separating agreement from execution provides savings

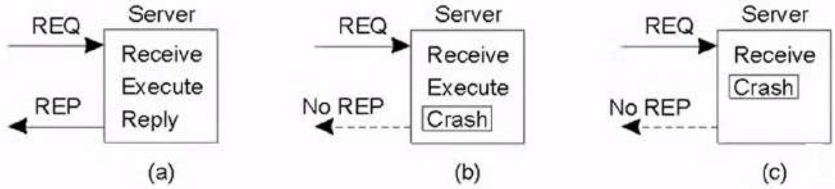
Reaching Agreement

- If message delivery is unbounded,
  - No agreement can be reached even if one process fails
  - Slow process indistinguishable from a faulty one

- BAR Fault Tolerance
  - Until now: nodes are byzantine or collaborative
  - New model: Byzantine, Altruistic and Rational
  - Rational nodes: report timeouts etc

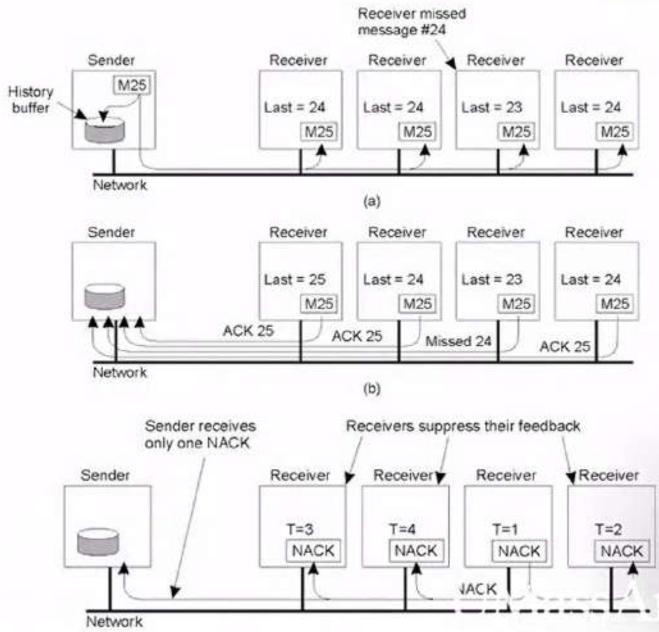
Reliable One-One Communicatio

- Issues were discussed in Lecture 3
  - Use reliable transport protocols (TCP) or handle at the application layer
- RPC semantics in the presence of failures
- Possibilities
  - Client unable to locate server
  - Lost request messages
  - Server crashes after receiving request
  - Lost reply messages
  - Client crashes after sending request



Reliable One-Many Communicatio

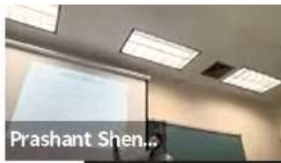
- Reliable multicast
  - Lost messages => need to retransmit
- Possibilities
  - ACK-based schemes
    - Sender can become bottleneck
  - NACK-based schemes



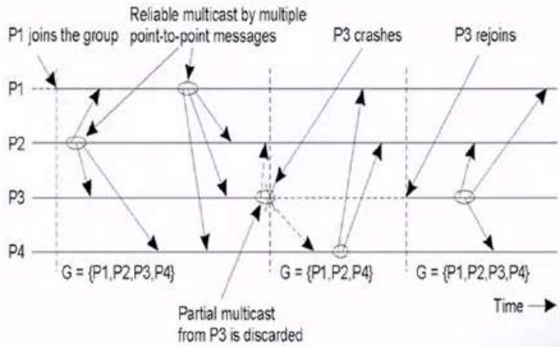
Broadcast Ordering

- Broadcast (or multicast) ordered important for replication
- FIFO broadcast: if a process sends m1 and then m2, all other processes receive m1 before m2
- Totally ordered: If a process receives m1 before m2 (regardless of sender), all processes receive m1 before m2
  - Does not imply FIFO, all processes just agree on order
- Causally ordered: if send(m1)->send (m2) => recv(m1)-> recv (m2)
- **State machine replication (SMR)**
  - Broadcast requests to all replicas using totally ordered broadcast; replicas apply requests in order.

Atomic Multicast

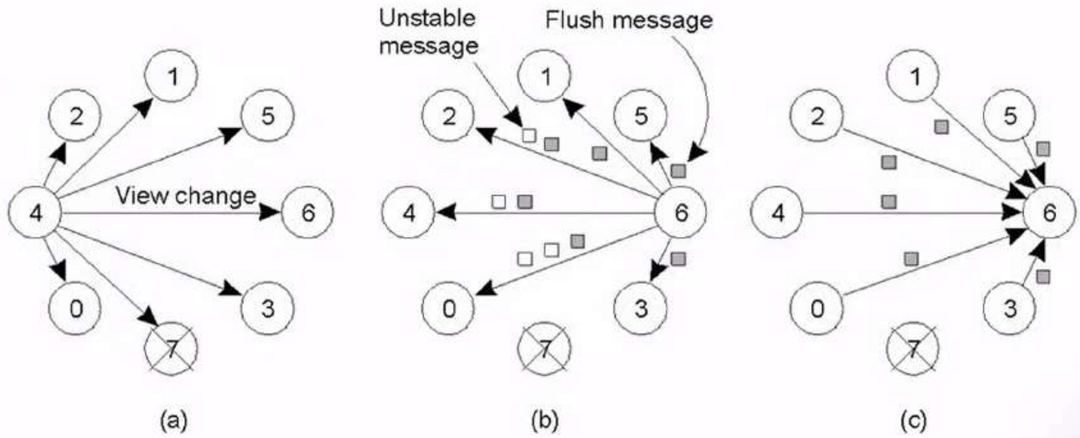


- Atomic multicast: a guarantee that all process received the message or none at all
  - Replicated database example
  - Need to detect which updates have been missed by a faulty process
- Problem: how to handle process crashes?
- Solution: *group view*
  - Each message is uniquely associated with a group of processes
    - View of the process group when message was sent
    - All processes in the group should have the same view (and agree on it)



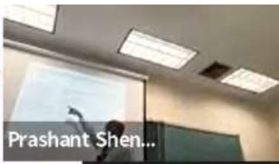
Virtually Synchronous Multicast

Implementing Virtual Synchrony in Is



a) Process 4 notices that process 7 has crashed, sends a view change
b) Process 6 sends out all its unstable messages, followed by a flush message
c) Process 6 installs the new view when it has received a flush message from everyone else

Implementing Virtual Synchrony

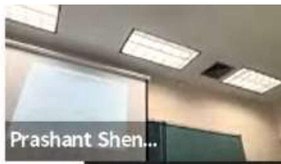


Multicast	Basic Message Ordering	Total-Ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

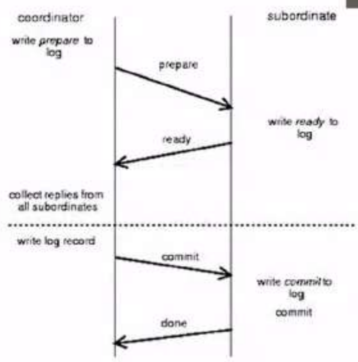
Distributed Commit

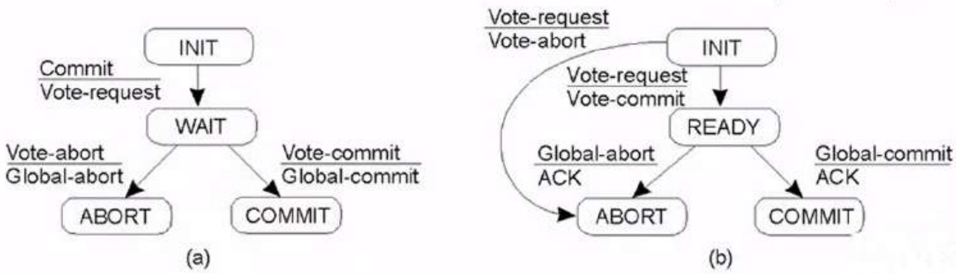
- Atomic multicast example of a more general problem
  - All processes in a group perform an operation or not at all
  - Examples:
    - Reliable multicast: Operation = delivery of a message
    - Distributed transaction: Operation = commit transaction
- Problem of distributed commit
  - All or nothing operations in a group of processes
- Possible approaches
  - Two phase commit (2PC) [Gray 1978]
  - Three phase commit

Two Phase Commit



- Coordinator process coordinates the operation
- Involves two phases
  - Voting phase: processes vote on whether to commit
  - Decision phase: actually commit or abort



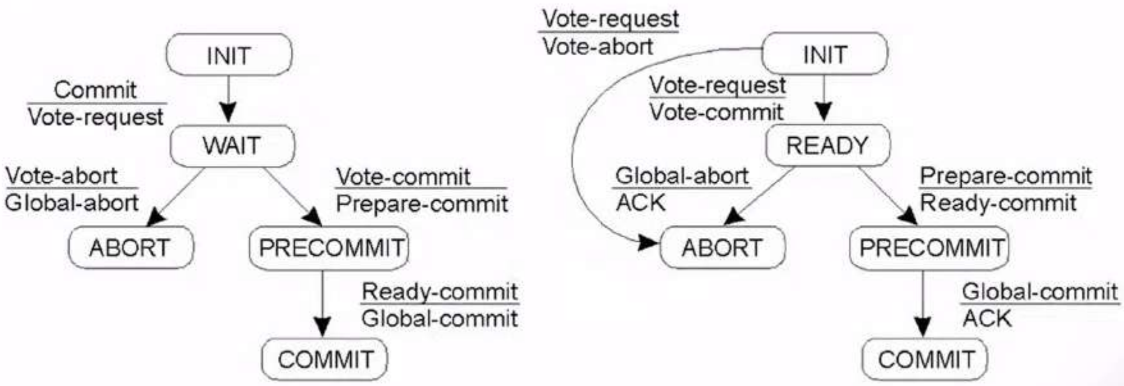


Recovering from a Crash

- If INIT : abort locally and inform coordinator
- If Ready, contact another process Q and examine Q's state

State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

Three-Phase Commit



(a) (b)

Two phase commit: problem if coordinator crashes (processes block)
Three phase commit: variant of 2PC that avoids blocking