Today: Fault Tolerance

- Agreement in presence of faults
  - Two army problem
  - Byzantine generals problem
- Reliable communication
- Distributed commit
  - Two phase commit
  - Three phase commit
- Paxos
- Failure recovery
  - Checkpointing
  - Message logging

Fault Tolerance

- Single machine systems
  - Failures are all or nothing
    - OS crash, disk failures
- Distributed systems: multiple independent nodes
  - Partial failures are also possible (some nodes fail)
- *Question:* Can we automatically recover from partial failures?
  - Important issue since probability of failure grows with number of independent components (nodes) in the systems
  - Prob(failure) = Prob(Any one component fails)=1-P(no failure)

A Perspective

- Computing systems are not very reliable
  - OS crashes frequently (Windows), buggy software, unreliable hardware, software/hardware incompatibilities
  - Until recently: computer users were "tech savvy"
    - Could depend on users to reboot, troubleshoot problems
  - Growing popularity of Internet/World Wide Web
    - "Novice" users
    - Need to build more reliable/dependable systems
  - Example: what is your TV (or car) broke down every day?
    - Users don't want to "restart" TV or fix it (by opening it up)
- Need to make computing systems more reliable
  - Important for online banking, e-commerce, online trading, webmail…

Basic Concepts

- Need to build *dependable* systems
- Requirements for dependable systems
  - Availability: system should be available for use at any given time
    - 99.999 % availability (five 9s) => very small down times
  - Reliability: system should run continuously without failure
  - Safety: temporary failures should not result in a catastrophic
    - Example: computing systems controlling an airplane, nuclear reactor
  - Maintainability: a failed system should be easy to repair

Basic Concepts (contd)

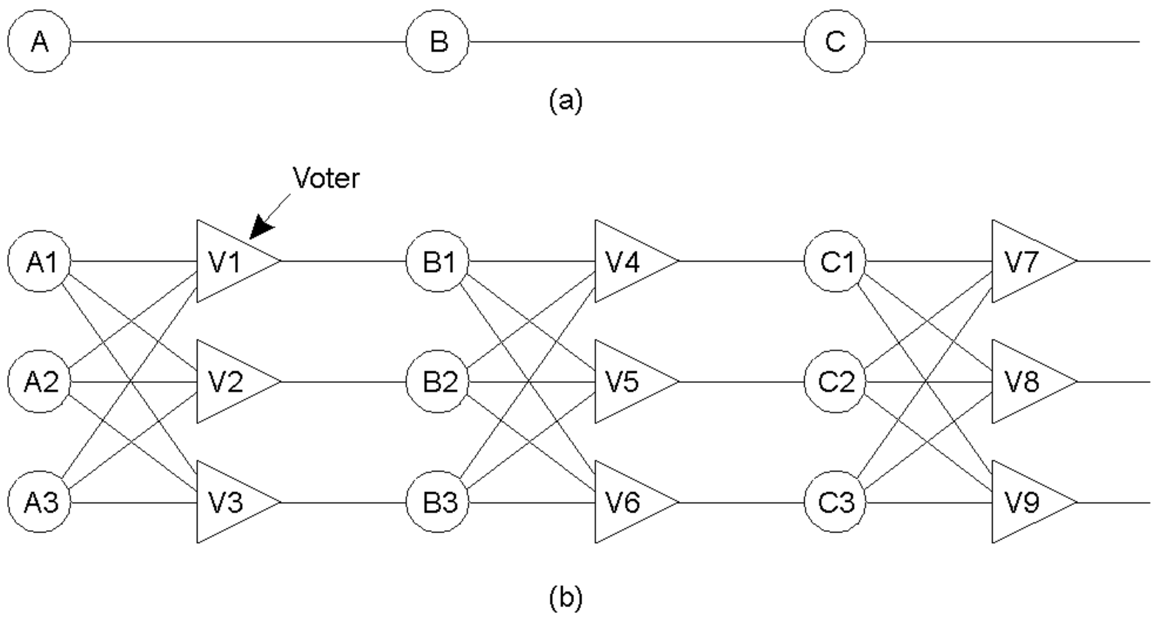
- Fault tolerance: system should provide services despite faults
  - Transient faults
  - Intermittent faults
  - Permanent faults

Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

- Different types of failures.

Failure Masking by Redundancy



- Triple modular redundancy.

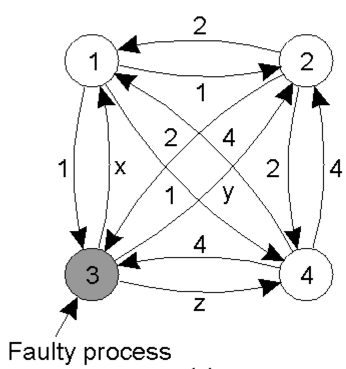
Agreement in Faulty Systems

- 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 army 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
- Byzantine generals problem: Can N generals reach agreement with a perfect channel?
  - M generals out of N may be traitors

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)

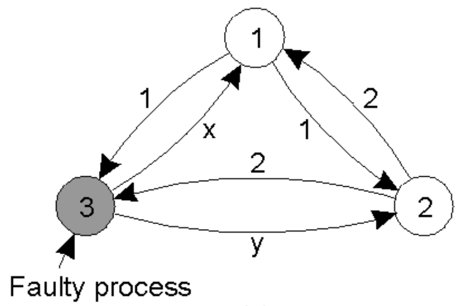
1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(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.

Byzantine Generals Problem Example



(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 2*m*+1 processes function correctly out of 3*m*+1 total processes. [Lamport 82]
  – Need more than two-thirds processes to function correctly

Byzantine Fault Tolerance

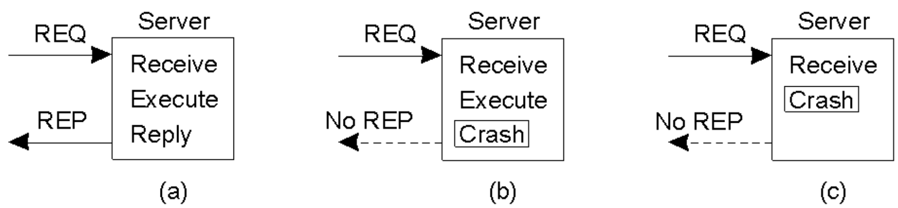
- Detecting a faulty process is easier
  – 2k+1 to detect k faults
- Reaching agreement is harder
  – Need 3k+1 processes (2/3rd majority needed to eliminate the faulty processes)
- Implications on real systems:
  – How many replicas?
  – Separating agreement from execution provides savings

Reaching Agreement

- If message delivery is unbounded,
  - No agreeement 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

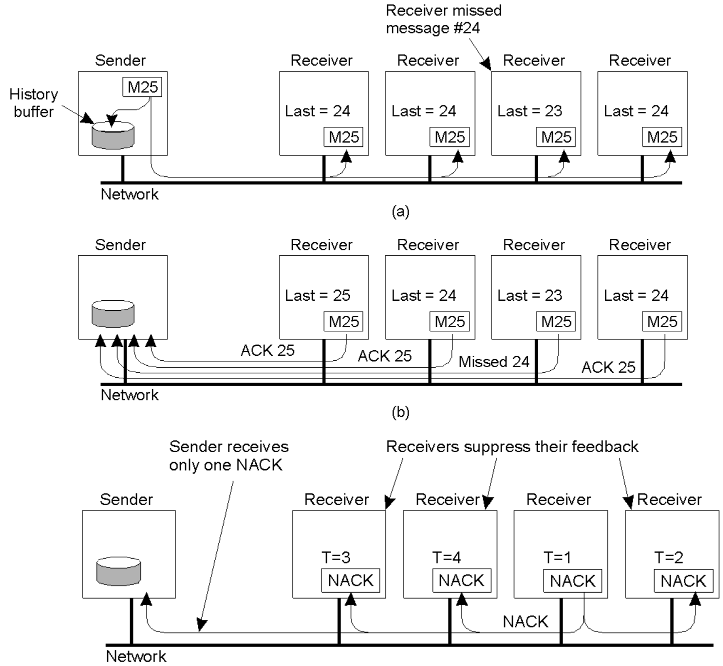
Reliable One-One Communication

- 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



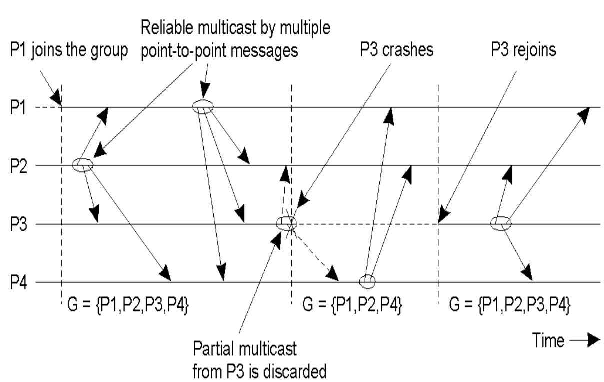
Reliable One-Many Communication

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



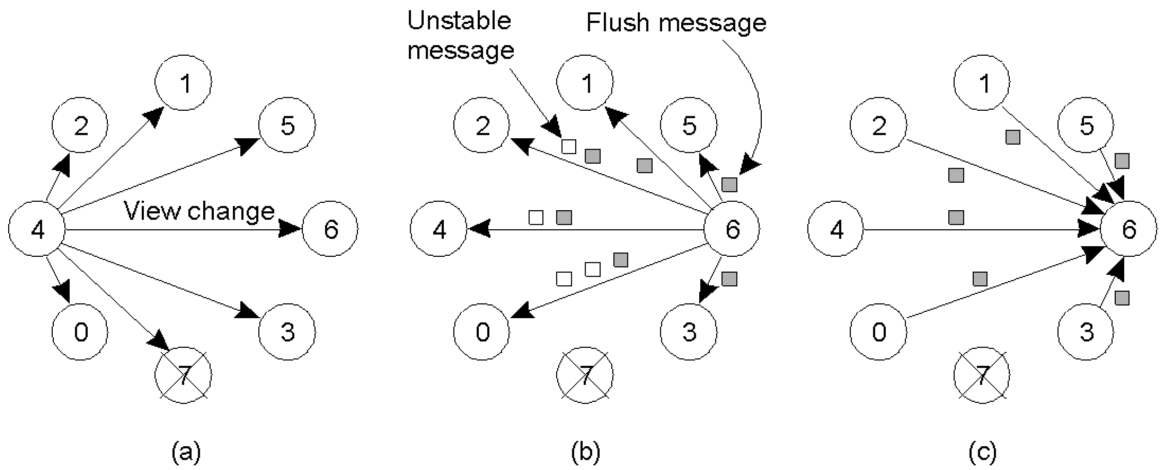
Atomic Multicast

- •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 Isis



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

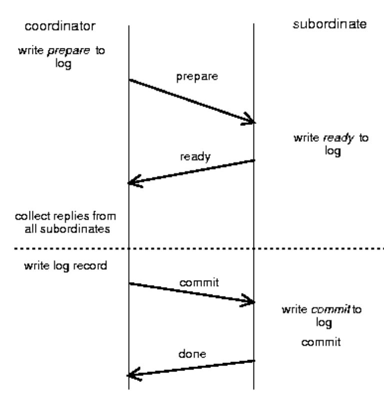
Multicast	Basic Message Ordering	Total-Ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

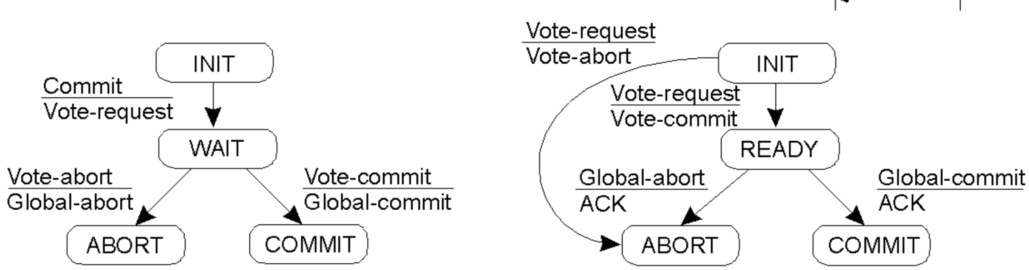
Distributed Commit

- 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





(a) (b)

Implementing Two-Phase Commit

actions by coordinator:

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        while GLOBAL_ABORT to local log;
        multicast  GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
    write GLOBAL_COMMIT to local log;
    multicast GLOBAL_COMMIT to all participants;
} else {
    write GLOBAL_ABORT  to local log;
    multicast GLOBAL_ABORT to all participants;
}
```

- Outline of the steps taken by the coordinator in a two phase commit protocol