Last Class: Weak Consistency

- Eventual Consistency and epidemic protocols
- Implementing consistency techniques
  - Primary-based
  - Replicated writes-based
    - Quorum protocols

Today: Fault Tolerance

- Basic concepts in fault tolerance
- Masking failure by redundancy
- Process resilience

Motivation

- 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

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)

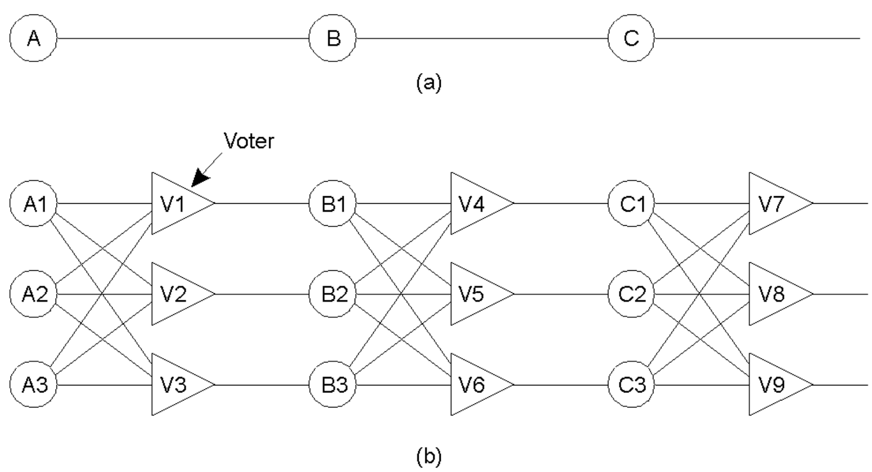
- Fault tolerance: system should provide services despite faults
  - Transient faults
  - Intermittent faults
  - Permanent faults

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

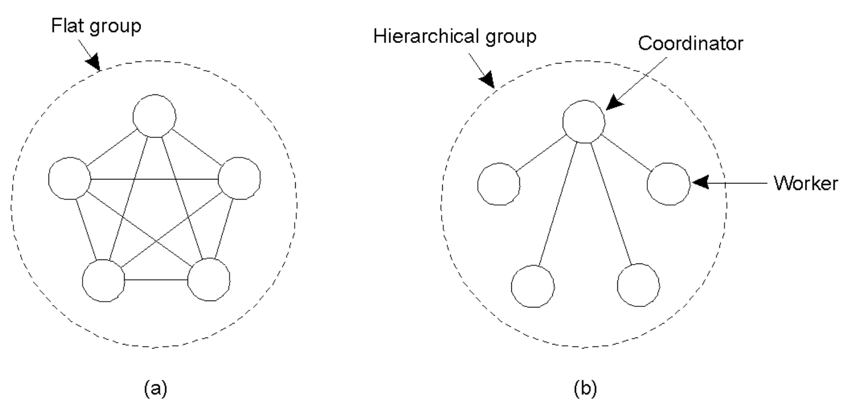


- Triple modular redundancy.

Process Resilience

- Handling faulty processes: organize several processes into a group
  - All processes perform same computation
  - All messages are sent to all members of the group
  - Majority need to agree on results of a computation
  - Ideally want multiple, independent implementations of the application (to prevent identical bugs)
- Use *process groups* to organize such processes

Flat Groups versus Hierarchical Groups



Advantages and disadvantages?

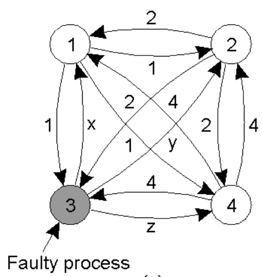
Agreement in Faulty Systems

- 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)

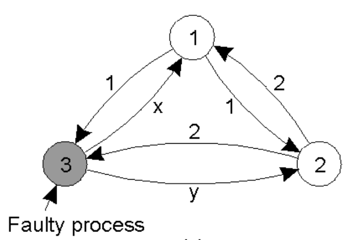
1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(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 [Lamport 82]
  - Need more than two-thirds processes to function correctly