Fault Tolerance

Topics

- What is fault tolerance?
- Why we need fault tolerant systems?
- Existing fault tolerant techniques/models (and where they are applied)?
- Challenges in developing fault tolerant systems.
- Future of fault tolerance.

Software Services

- People use software services and applications everyday.
- They will continue to use them only if these services/applications are

DEPENDABLE.

Dependability

- **Specified Service** : How the behavior of the service should be
- **Derived Service** : Actual behavior of the service
- If the dependability goes down, Derived service will start to deviate from the Specified Service

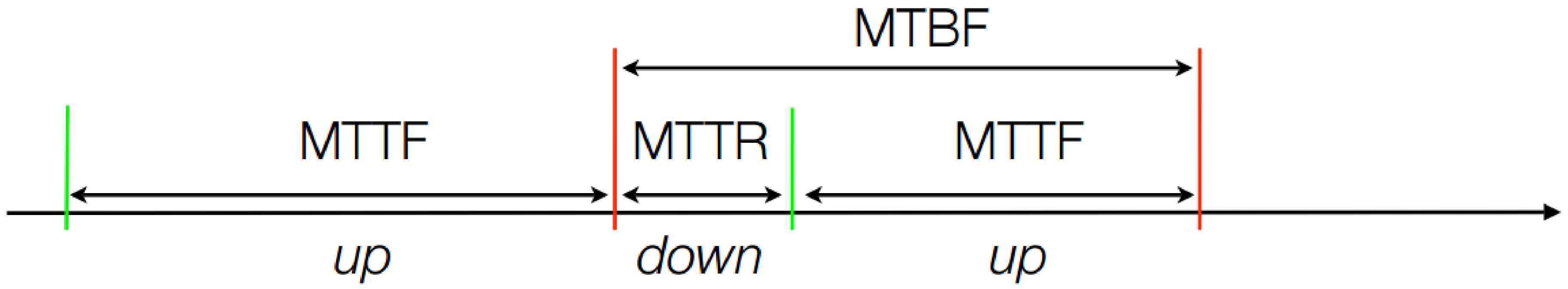
Dependability

- *Availability* – the system is ready to be used immediately.
- *Reliability* – the system can run continuously without failure.
- *Safety* – if a system fails, nothing catastrophic will happen.
  major harm
- *Maintainability* – when a system fails, it can be repaired easily and quickly (and, sometimes, without its users noticing the failure).

Reliability and Availability

- **Mean time to failure (MTTF)** – Average time it takes for the system to fail
- **Mean time to recover (MTTR)** – Average time it takes to recover
- **Mean time between failures (MTBF)** – Average time between failures

Reliability and Availability



$$A = \frac{Uptime}{Uptime + Downtime} = \frac{MTTF}{MTTF + MTTR}$$

Reliability and Availability

Availability	Downtime per year	Downtime per week
90.0 % (1 nine)	36.5 days	16.8 hours
99.0 % (2 nines)	3.65 days	1.68 hours
99.9 % (3 nines)	8.76 hours	10.1 min
99.99 % (4 nines)	52.6 min	1.01 min
99.999 % (5 nines)	5.26 min	6.05 s
99.9999 % (6 nines)	31.5 s	0.605 s
99.99999 % (7 nines)	0.3 s	6 ms

Faults, Errors and Failures

- A fault is the cause of an error. It is associated with a defect.
- An error is part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service.
- A system failure is an event that occurs when the derived service deviates from specified service.
- A fault originally causes an error within the state of one (or more) components, but system failure will not occur as long as the error does not reach the service interface of the system.

What is fault tolerance?

- The approach of fault-tolerance expect faults to be present during system operation, but employs design techniques which insure the continued correct execution of the computing process

Why do we need fault tolerant systems?

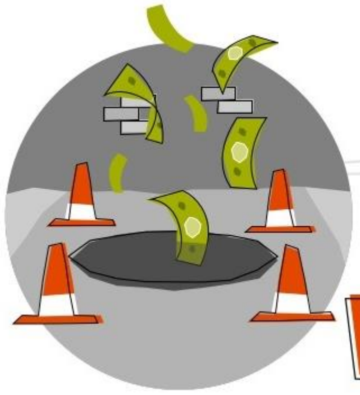
- Average costs per hour of downtime (Gartner 1998)
  - Brokerage operations in finance: $6.5 million
  - Credit card authorization: $2.6 million
  - Home catalog sales: $90.000
  - Airline reservation: $89.500

- 22-hour service outage of eBay in June 1999
  - Interruption of around 2.3 million auctions
  - 9.2% stock value drop

Why we need fault tolerant systems?

For the Fortune 1000, the average total cost of unplanned application downtime per year is

\$1.25 billion - \$2.5 billion



The average cost of a critical application failure per hour is

\$500,000 - \$1 million

The average hourly cost of an infrastructure failure is

\$100,000 per hour

Stats from 2015

Fault Classification

- By cause
  - Hardware Faults
  - Design Faults
  - Operation Faults
  - Environment Faults

- By duration
  - Permanent Faults
  - Intermittent Faults
  - Transient Faults

Failure Classification

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 (Byzantine failure)	A server may produce arbitrary responses at arbitrary times

Fault Tolerance Strategies

- Fault tolerance in computer system is achieved through redundancy in hardware, software, information, and/or time.
  Such redundancy can be implemented in static, dynamic, or hybrid configurations.

- Fault tolerance can be achieved by many techniques:
  - Fault masking is any process that prevents faults in a system from introducing errors. Example: Error correcting memories and majority voting.
  - Reconfiguration is the process of eliminating faulty component from a system and restoring the system to some operational state.

Reconfiguration Approach

- **Fault detection** is the process of recognizing that a fault has occurred. Fault detection is often required before any recovery procedure can be initiated.
- **Fault location** is the process of determining where a fault has occurred so that an appropriate recovery can be initiated.
- **Fault containment** is the process of isolating a fault and preventing the effects of that fault from propagating throughout the system.
- **Fault recovery** is the process of regaining operational status via reconfiguration even in the presence of faults.

The Concept of Redundancy

- *Redundancy* is simply the addition of information, resources, or time beyond what is needed for normal system operation.

- **Software redundancy** is the addition of extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults.

- **Hardware redundancy** is the addition of extra hardware, usually for the purpose either detecting or tolerating faults.

- **Information redundancy** is the addition of extra information beyond that required to implement a given function; for example, error detection codes.

The Concept of Redundancy

- **Time redundancy** uses additional time to perform the functions of a system such that fault detection and often fault tolerance can be achieved. *Transient faults* are tolerated by this.

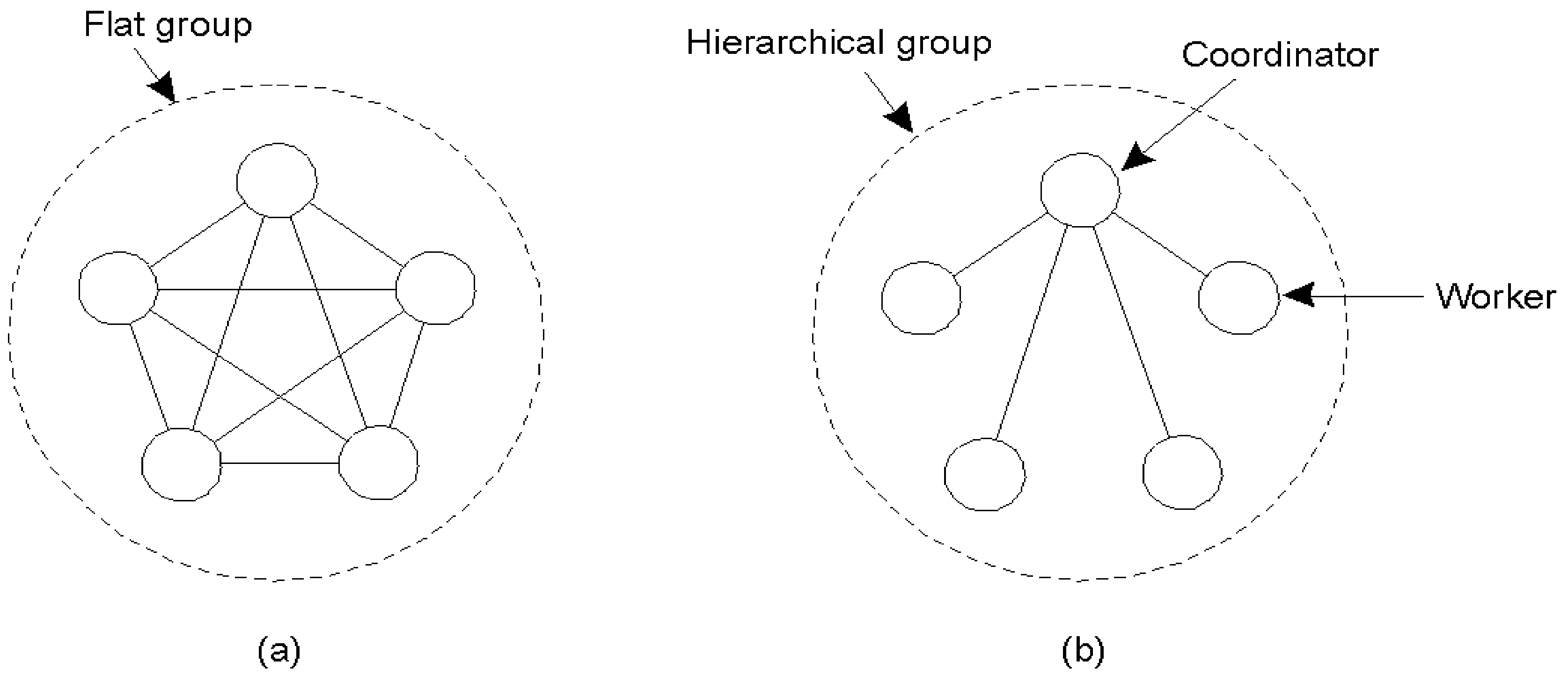
- The use of redundancy can provide additional capabilities within a system. But, redundancy can have very important impact on a system's performance, size, weight and power consumption.

Software Redundancy

Process Resilience

- Mask process failures by replication
- Organize processes into groups, a message sent to a group is delivered to all members
- If a member fails, another should fill in

Flat Groups versus Hierarchical Groups



a) Communication in a flat group.
b) Communication in a simple hierarchical group

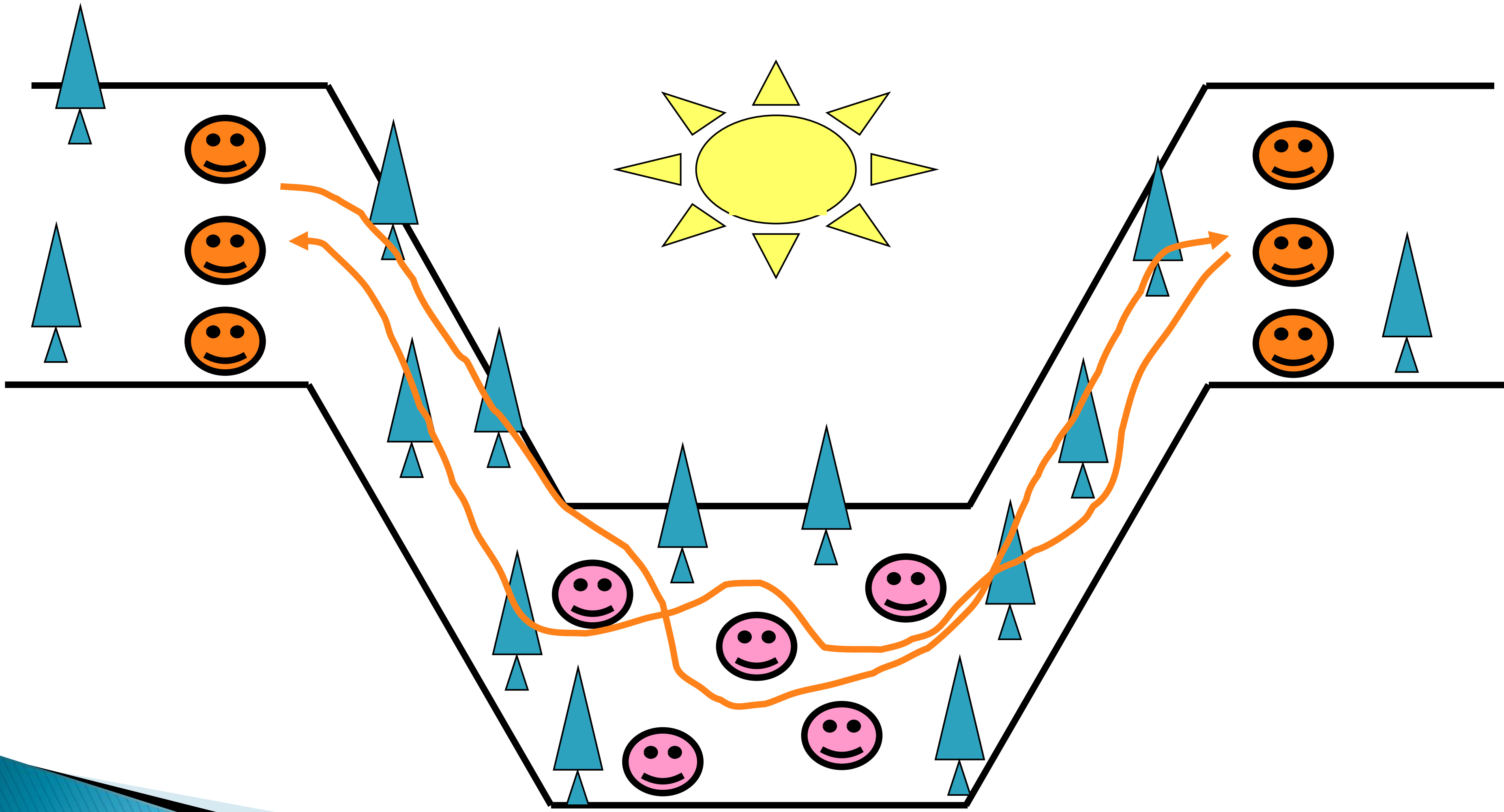
Process Replication

- Replicate a process and group replicas in one group
- A system is k fault-tolerant if it can survive and function even if it has k faulty processes
  - For crash failures (a faulty process halts, but is working correctly until it halts)
    - k+1 replicas
  - For Byzantine failures (a faulty process may produce arbitrary responses at arbitrary times)
    - 2k+1 replicas

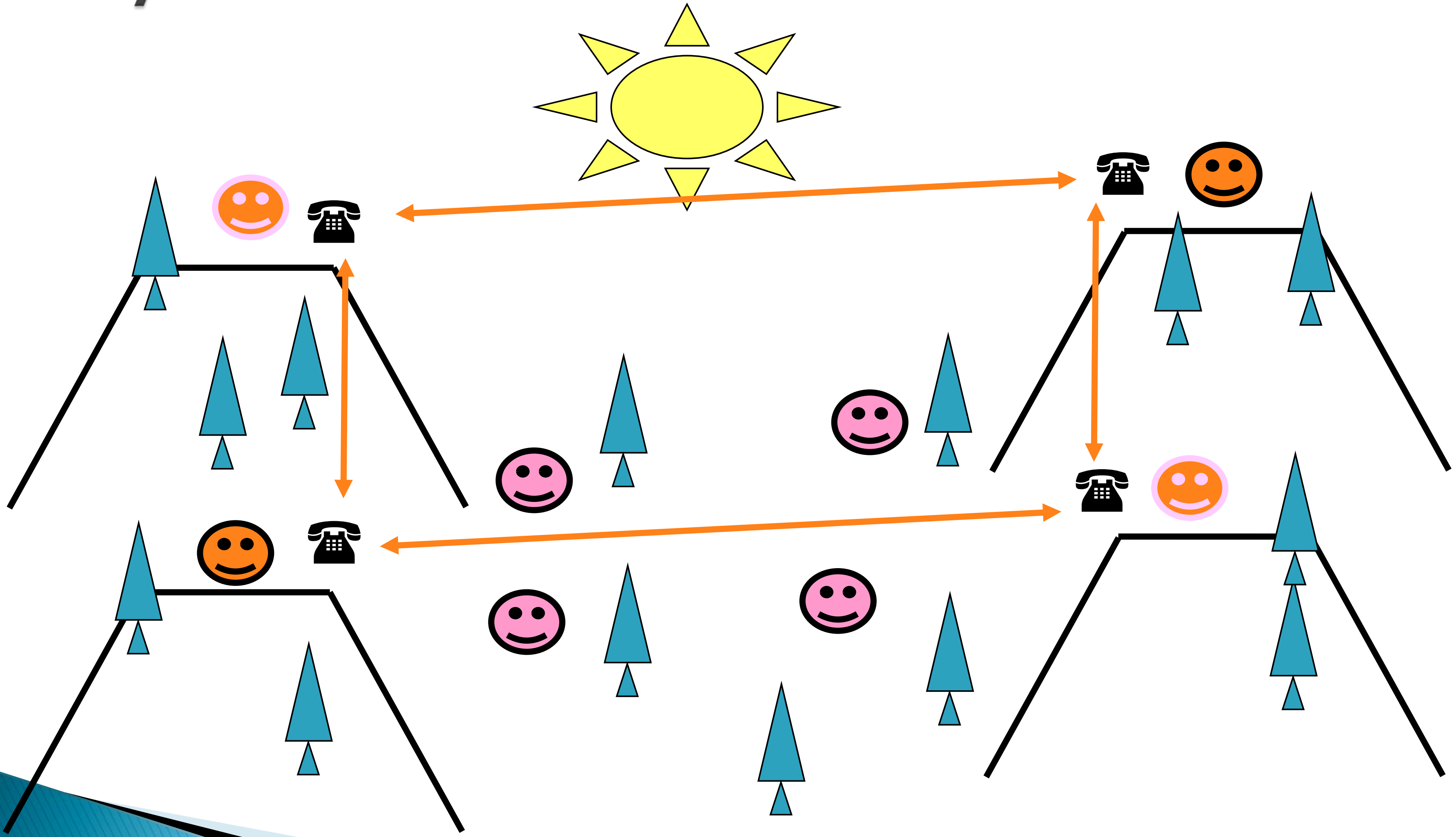
Agreement

- Need agreement in DS:
  - Leader, commit, synchronize
- **Distributed Agreement algorithm**: all non-faulty processes achieve consensus in a finite number of steps
- Perfect processes, faulty channels: two-army
- Faulty processes, perfect channels: Byzantine generals

Two-Army Problem



Byzantine Generals Problem



Distributed COMMIT

- **General Goal:**
  - *We want an operation to be performed by all group members, or none at all.*

- There are three types of “commit protocol”: single-phase, two-phase and three-phase commit.

Commit Protocols

- **One-Phase Commit Protocol**:
  - An elected co-ordinator tells all the other processes to perform the operation in question.
  - But, what if a process cannot perform the operation? There's no way to tell the coordinator! Whoops …
- **The solutions**:
  - The *Two-Phase* and *Three-Phase Commit Protocols*.

The Two-Phase Commit Protocol

- First developed in 1978!!!
- *Summarized: GET READY, OK, GO AHEAD.*

1. The coordinator sends a *VOTE_REQUEST* message to all group members.
2. The group member returns *VOTE_COMMIT* if it can commit locally, otherwise *VOTE_ABORT*.
3. All votes are collected by the coordinator. A *GLOBAL_COMMIT* is sent if all the group members voted to commit. If one group member voted to abort, a *GLOBAL_ABORT* is sent.
4. The group members then **COMMIT** or **ABORT** based on the last message received from the coordinator.

Big Problem with Two-Phase Commit

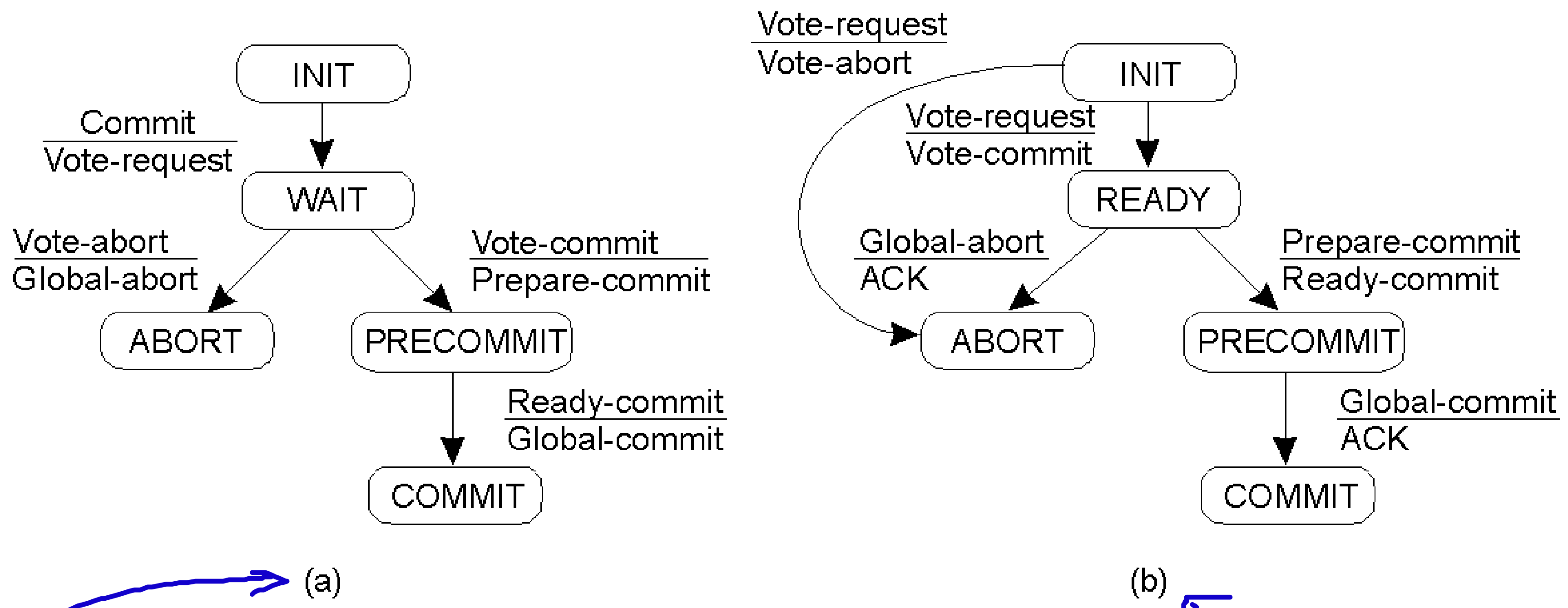
- It can lead to both the coordinator and the group members **blocking**, which may lead to the dreaded *deadlock*.

- If the coordinator crashes, the group members may not be able to *reach a final decision*, and they may, therefore, block until the coordinator *recovers* …

- Two-Phase Commit is known as a **blocking-commit protocol** for this reason.

- The solution? *The Three-Phase Commit Protocol.*

Three-Phase Commit


INIT
Commit / Vote-request
WAIT
Vote-abort / Global-abort
Vote-commit / Prepare-commit
ABORT
PRECOMMIT
Ready-commit / Global-commit
COMMIT
(a)
Vote-request / Vote-abort
INIT
Vote-request / Vote-commit
READY
Global-abort / ACK
Prepare-commit / Ready-commit
ABORT
PRECOMMIT
Global-commit / ACK
COMMIT
(b)

a) Finite state machine for the coordinator.

b) Finite state machine for a group member.

c) **Main point**: although 3PC is generally regarded as *better* than 2PC, it is *not applied often in practice*, as the conditions under which 2PC blocks rarely occur.

Software Redundancy – to Detect Hardware Faults

- **Consistency checks** use a priori knowledge about the characteristics of the information to verify the correctness of that information.
  Example: Range checks, overflow and underflow checks.

- **Capability checks** are performed to verify that a system possesses the capability expected.
  Examples: Memory test – a processor can simply write specific patterns to certain memory locations and read those locations to verify that the data was stored and retrieved properly.

Software Redundancy – to Detect Hardware Faults

- **ALU tests:** Periodically, a processor can execute specific instructions on specific data and compare the results to known results stored in ROM.

- **Testing of communication** among processors, in a multiprocessor, is achieved by periodically sending specific messages from one processor to another or writing into a specific location of a shared memory.

Software Redundancy – to Detect Hardware Faults.

- All modern day microprocessors use instruction retry
- Any transient fault that causes an exception such as parity violation is retried
- Very cost effective and is now a standard technique

Software Redundancy – to Detect Software Faults

- There are two popular approaches: **N-Version Programming** (NVP) and **Recovery Blocks (RB)**.
- NVP is a forward recovery scheme – it masks faults.
- RB is a backward error recovery scheme.
- In NVP, multiple versions of the same task is executed concurrently, whereas in RB scheme, the versions of a task are executed serially.
- NVP relies on *voting.*
- RB relies on *acceptance test.*

Single Version Fault Tolerance: Software Rejuvenation

- Example: Rebooting a PC
- As a process executes
  - it acquires memory and file-locks without properly releasing them
  - memory space tends to become increasingly fragmented
- The process can become faulty and stop executing
- To head this off, proactively halt the process, clean up its internal state, and then restart it
- Rejuvenation can be time-based or prediction-based
- Time-Based Rejuvenation - periodically
- Rejuvenation period - balance benefits against cost

HARDWARE REDUNDANCY

Hardware Redundancy

- **Static techniques** use the concept of fault masking. These techniques are designed to achieve fault tolerance without requiring any action on the part of the system. Relies on voting mechanisms.

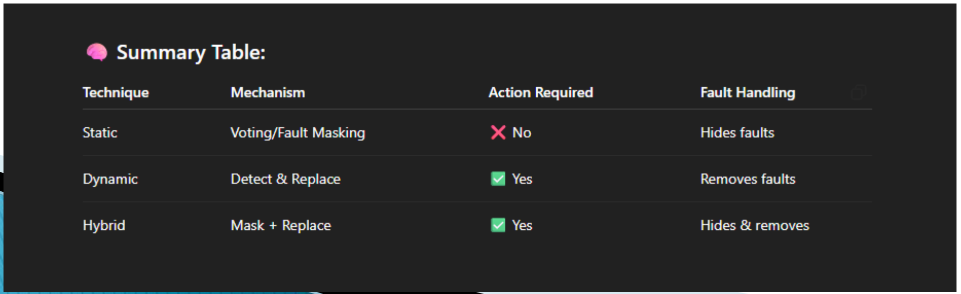
  (also called passive redundancy or fault-masking)

- **Dynamic techniques** achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty hardware from the system. That is, active techniques use fault detection, fault location, and fault recovery in an attempt to achieve fault tolerance.

  (also called active redundancy)

Hardware Redundancy (Cont'd)

- **Hybrid techniques** combine the attractive features of both the passive and active approaches.
  - Fault masking is used in hybrid systems to prevent erroneous results from being generated.
  - Fault detection, location, and recovery are also used to improve fault tolerance by removing faulty hardware and replacing it with spares.



🧠 Summary Table:

Technique	Mechanism	Action Required	Fault Handling
Static	Voting/Fault Masking	✗ No	Hides faults
Dynamic	Detect & Replace	✓ Yes	Removes faults
Hybrid	Mask + Replace	✓ Yes	Hides & removes

Future of Fault Tolerance

- Apache Zookeeper
  - ZooKeeper Atomic Broadcasts
- ETCD
- Consul
  - Gossip Protocol
- Doozer
- Apache Akka