CS 550:
Advanced Operating Systems

Fault Tolerance

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
March 22nd, 2011

Outline: Fault Tolerance

- Basic concepts in fault tolerance
- Masking failure by redundancy
- Process resilience
- Reliable communication
  - One-one communication
  - One-many communication
- Distributed commit
  - Two phase commit
- Failure recovery
  - Checkpointing
  - Message logging

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 if 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 <u>at any given time</u> (99.999% means ?)
  - *Reliability:* system should run continuously without failure (<u>over a time interval</u>)
  - *Safety:* temporary failures should not result in a catastrophe
  - *Maintainability:a failed system should be easy to repair*

Basic Concepts (cont.)

- Fault tolerance: system should provide services despite faults

- Three types:
  - Transient faults
  - Intermittent faults
  - Permanent faults

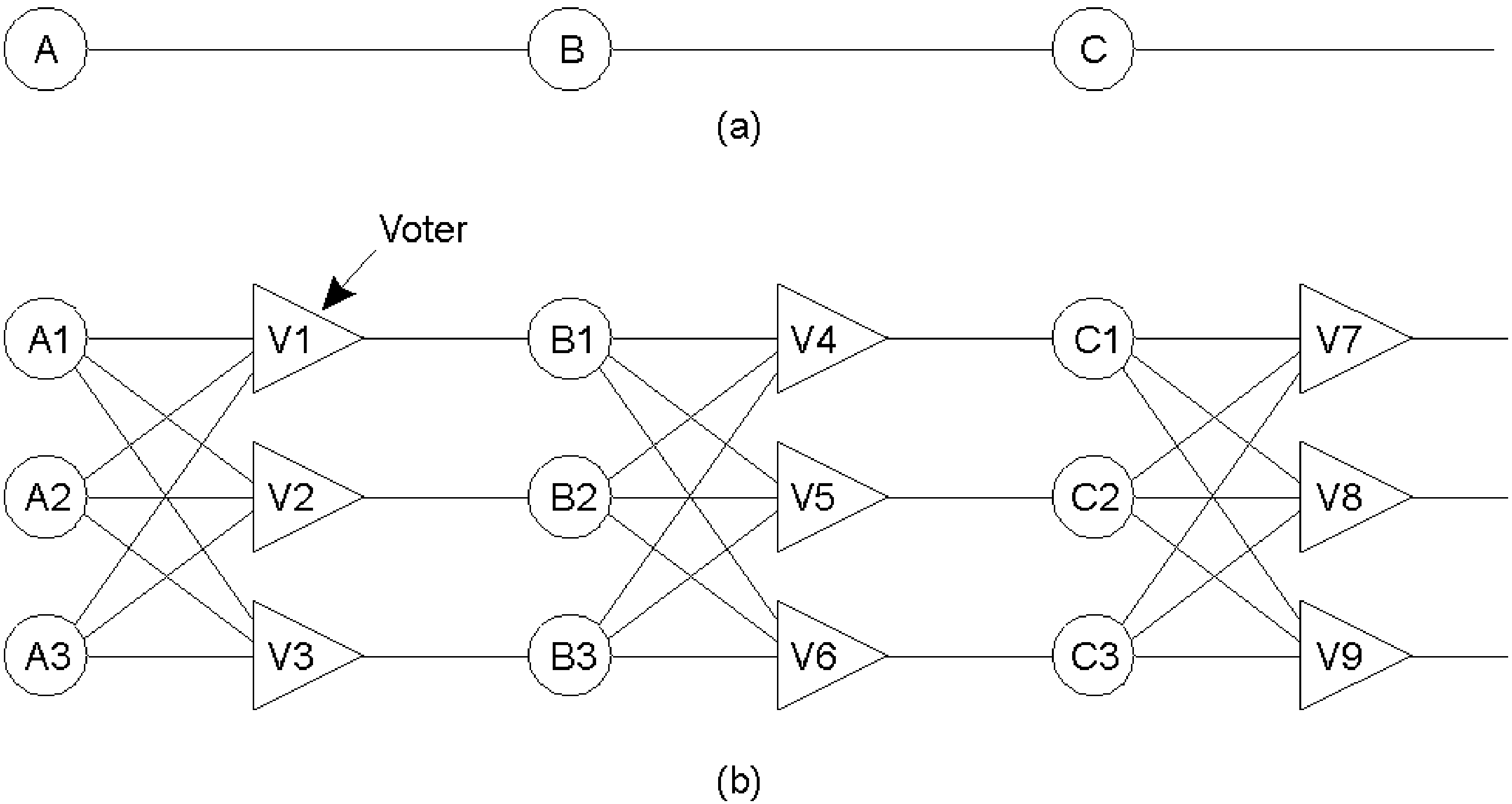
Failure Models

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

Process Resilience

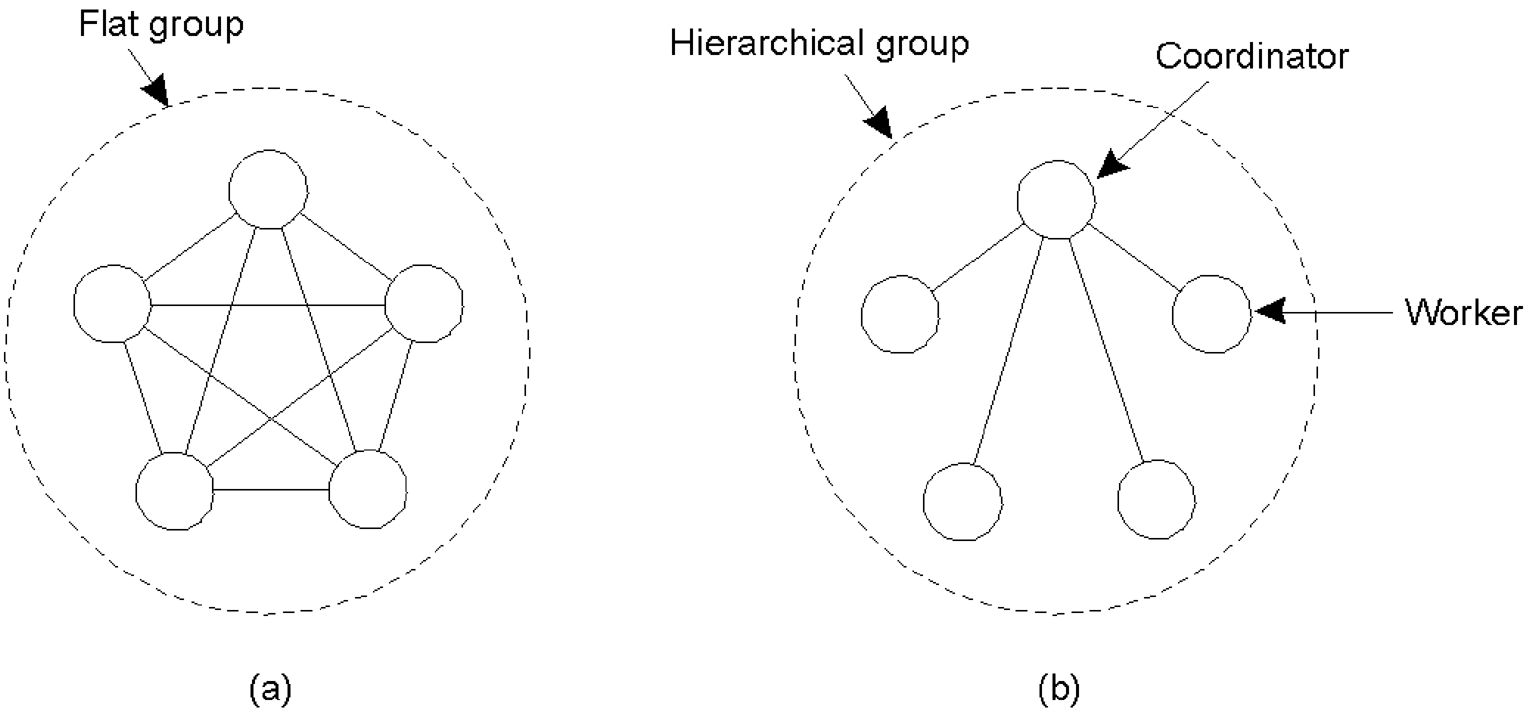
- Handling faulty processes:
  - Use process group:
    - All processes perform the same operations
    - All messages are sent to all members of the group
    - Majority need to agree on results of an operation

Failure Masking by Redundancy



Triple Modular Redundancy (TMR)

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
- (1) If processes fail silently: (k+1) components
- *(2) if Byzantine failures*: (2k+1) components

Byzantine Fault Tolerance

- Defend against Byzantine failures
- Components of a system fail in arbitrary ways
  - not just by stopping or crashing but by processing requests incorrectly, corrupting their local state, and/or producing incorrect or inconsistent outputs
- Correct functionality assuming not too many Byzantine faulty components

Byzantine Faults

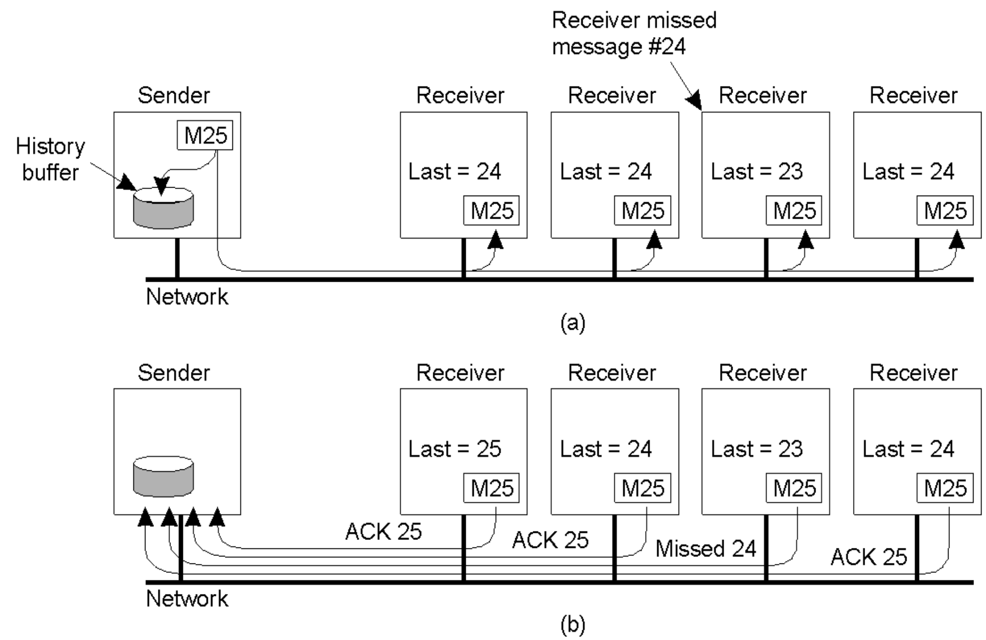
- Two army problem:
  - Each army coordinates with a messenger
  - Messenger can be captured by the hostile army
  - Can generals reach agreement?
  - Conclusion: ?

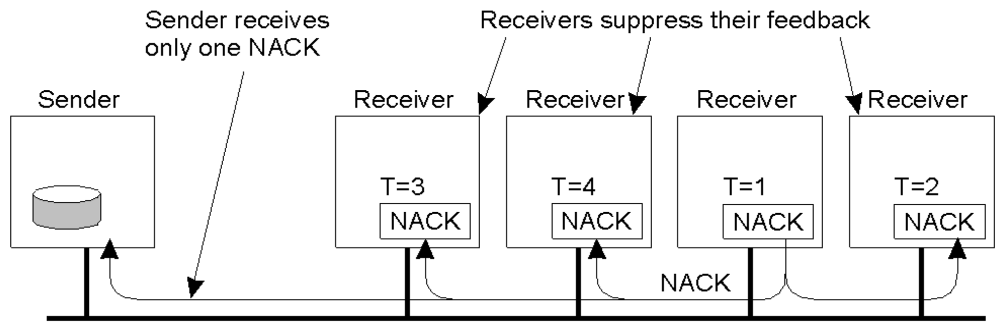
Reliable One-One Communication

- Make use of a reliable transport protocol (TCP) or handle at the application layer
- In Chapter 4, we summarized five different classes of failures in RPC systems:
  - 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

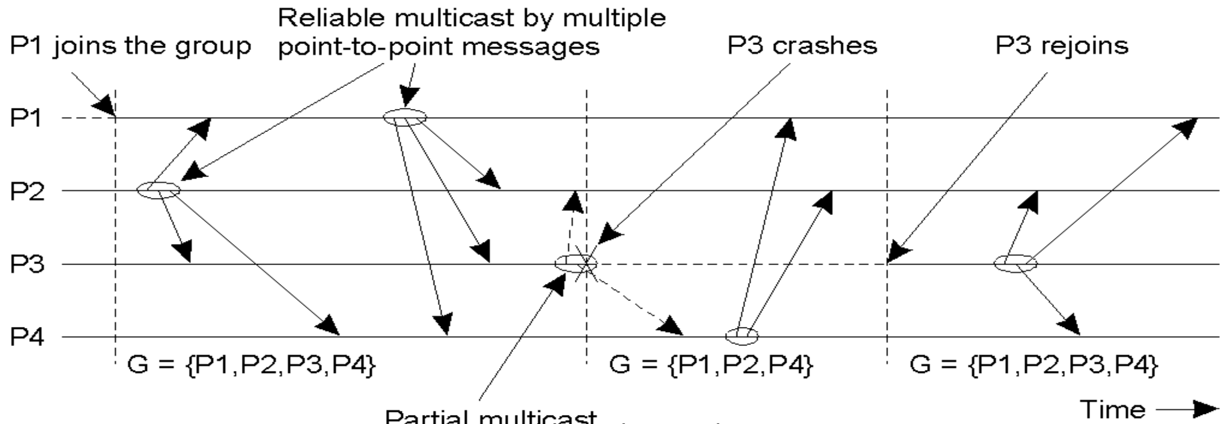
- Reliable multicast
  - Lost messages => need to retransmit
- Approaches:
  - ACK-based schemes
    - Problems?
  - NACK-based schemes





Atomic Multicast

- Atomic multicast: all processes received the message or none at all
- Solution: Group view & View change
  - Each msg is uniquely associated with a group of processes
  - View of the process group when message was sent
  - All procs in the group should have the same view
  - Virtually synch property

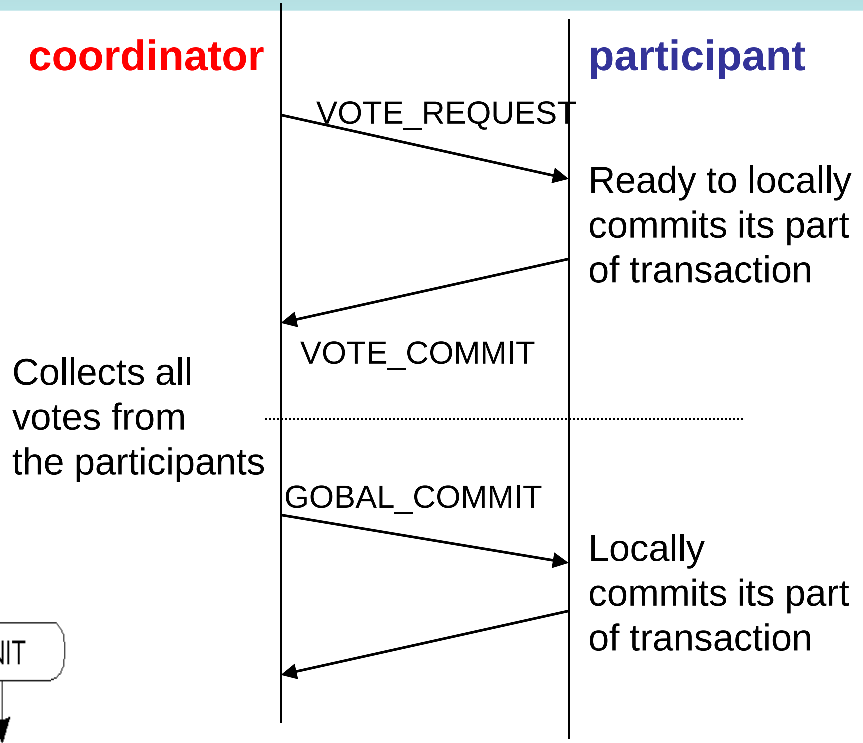


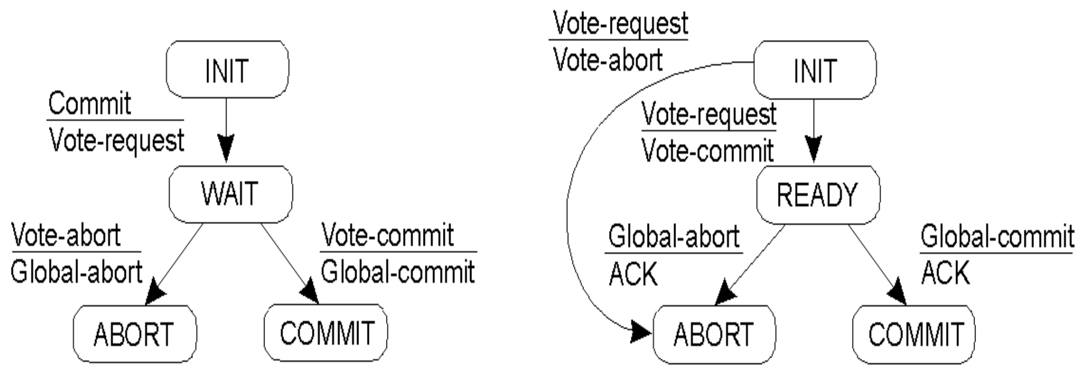
Distributed Commit

- Distributed commit: 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
- Possible approaches
  - One phase commit
  - Two phase commit (2PC) [Gray 1978]
  - Three phase commit

Two Phase Commit

- Coordinator: coordinates the operation
- Involves two phases
  - Voting phase
  - Decision phase





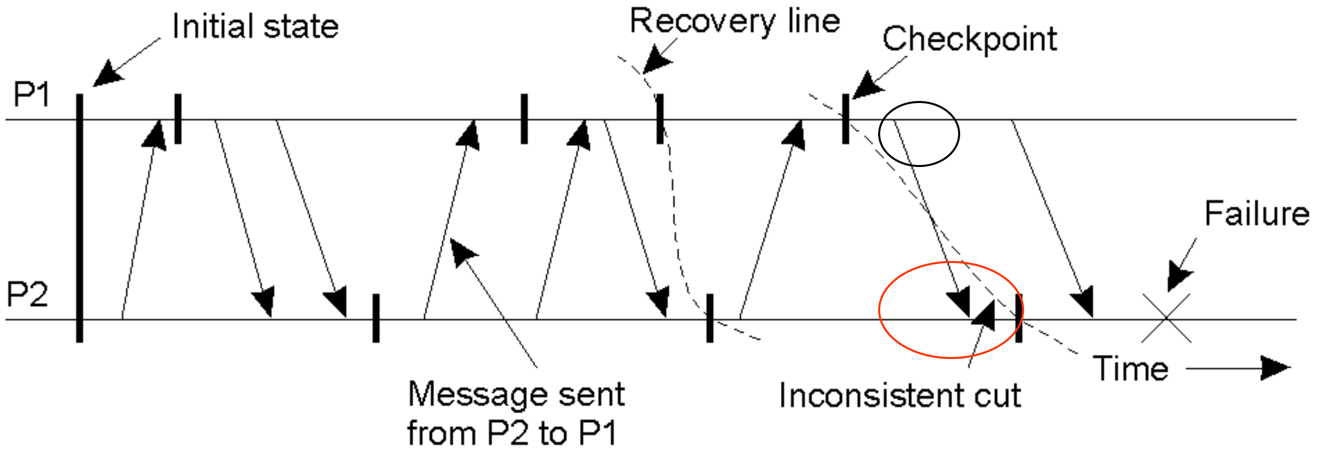
(a) (b)

Recovery

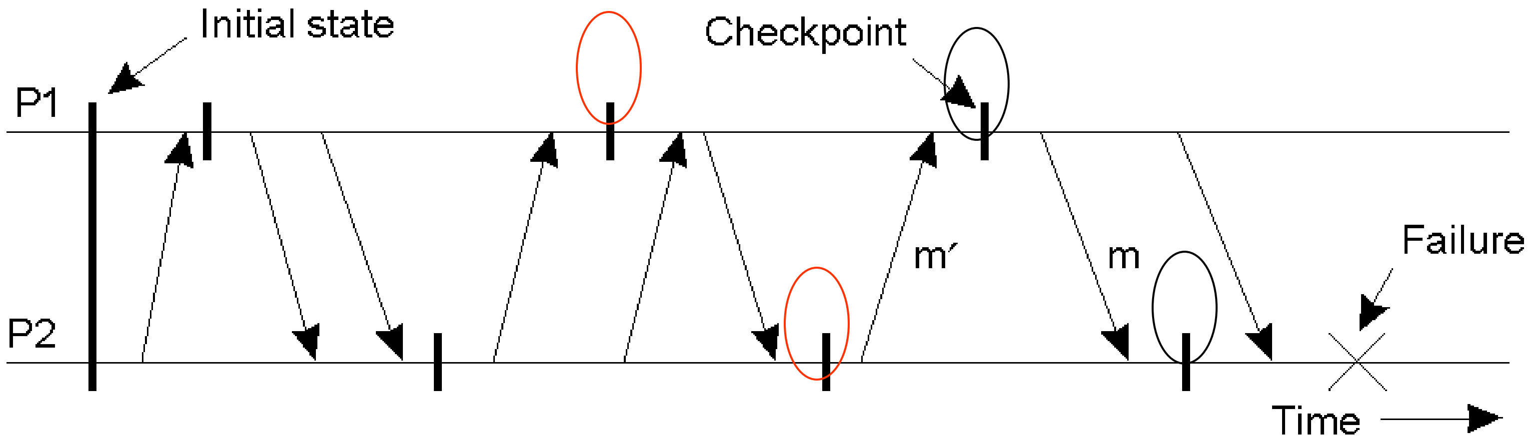
- Techniques thus far allow failure handling
- Recovery: operations that must be performed after a failure to recover to a correct state
- Techniques:
  - Backward recovery
  - Forward recovery
- Storage types:
  - RAM
  - Disk
  - Stable storage

Checkpointing

- Steps:
  - ?
- Key issue: consistent cut & recovery line



Independent Checkpointing



- Each processes periodically checkpoints independently of other processes
- Upon a failure, work backwards to locate a consistent cut
- Problem: ?

Message Logging

- Checkpointing is expensive
  - All procs restart from previous consistent cut
  - Taking a snapshot is expensive

- Combine checkpointing (expensive) with message logging (cheap)
  - Take infrequent checkpoints
  - Log all msgs between checkpoints to local stable storage
  - To recover: simply replay msgs from previous checkpoint
    - Avoid recomputations from previous checkpoint

Summary

- Basic concepts in fault tolerance
- Reliable communication
  - One-one communication
  - One-many communication
- Distributed commit
  - Two phase commit
- Failure recovery
  - Checkpointing
  - Message logging
- Reading materials:
  - AST chpt 8

Questions

?