# CSED601
# Dependable Computing
# Lecture 5

Jong Kim

Dept. of CSE

POSTECH

# Review of Previous Lecture

- Fault Tolerance
- Fault-Tolerance Idea
- Hardware redundancy
  - Passive H/W redundancy
    - TMR, NMR
  - Active H/W redundancy
    - Duplication with comparison, Stand-by sparing, Pair and a spare, Watchdog timer
  - Hybrid H/W redundancy
    - NMR with sparing, Self-purging redundancy, Sift-out modular redundancy, Triple-duplex architecture

# Software redundancy

- Concept
  - Many fault-detection and fault-tolerance techniques can be implemented in software
  - Require minimal redundant hardware / substantial redundant software
    - Redundant software does not need to be a complete copy.

# Software redundancy methods

- Methods
  - Consistency checks
  - Capability checks
  - N-version programming
  - Algorithm construction
  - Forward error recovery
  - Checkpointing
  - Journaling
  - Recovery blocks
  - Self-checking programming

# Consistency checks

- Concept
  - Uses a priori knowledge about the characteristics of information to verify the correctness of that information.
  - Commonly used for error detection
  - Eg: Memory address, detection of invalid instruction
- Software implementation
  - Compare the predicted performance with the actual performance
    - If the deviation is too large, fault (physical of man-made) exists.
    - Two problems of this approach
      - Prediction accuracy
      - Deviation tolerance
  - Counting the number of words transferred on the bus.

# Capability checks

- Concept
  - Verification of capability that a system is expected to possess
  - Commonly used for fault detection
- Implementation examples
  - Memory test : write and read
  - ALU test : execute specific instruction on specific data and compare the results with the known result
  - Communication check with other processors in multiprocessors
  - Access right check for detecting man-made fault

# N-version programming

- Concept
  - Software implementation of NMR
  - Used for detecting and/or tolerating faults that occur in the software of a system
  - Software fault types
    - Design fault, coding mistake
- Implementation
  - Design and code the software module N times and to compare the N results produced by these modules
  - Each module is designed and coded by a separate group of programmers
  - N-version can avoid the common-mode failure that can occur by software fault.

# N-version programming

- Problem
  - Designer (also coder) usually do the same mistakes usually.
  - Developed from common specifications (cannot detect specification mistakes)

# Algorithm construction

- Concept
  - A set of high-level faults are identified, and algorithm is designed that tolerate those faults.
  - Most of the work on algorithm construction has focused on distributed system.
- Examples
  - Reaching agreement among a pool of processors
    - Goal : Total system agreement in a distributed system
    - Fault : a failure of a single system in a distributed system
    - Failure model : Failed-Notified, Failed-stop, Message omission, Timing (out-of-order, delay), Byzantine failure
    - Known as Byzantine generals' problem

# Algorithm construction

- Examples
  - Consistency among multiple copies of database
    - Goal : Maintaining consistency among multiple copies of database
    - Use two phase protocol
      - 1st phase : Every process either prepare state or abort message
      - 2nd phase: commit if everything commits.
        - » Select commit coordinator -> record updates -> notify all process if every process acknowledges its commit
    - Cannot tolerate byzantine failure
      - To reach an agreement when there are t byzantine faulty nodes, there must be at least 3t nodes in the system.
  - Leader selection problem
    - Choose a leader among processors by exchanging messages.

# Forward error recovery

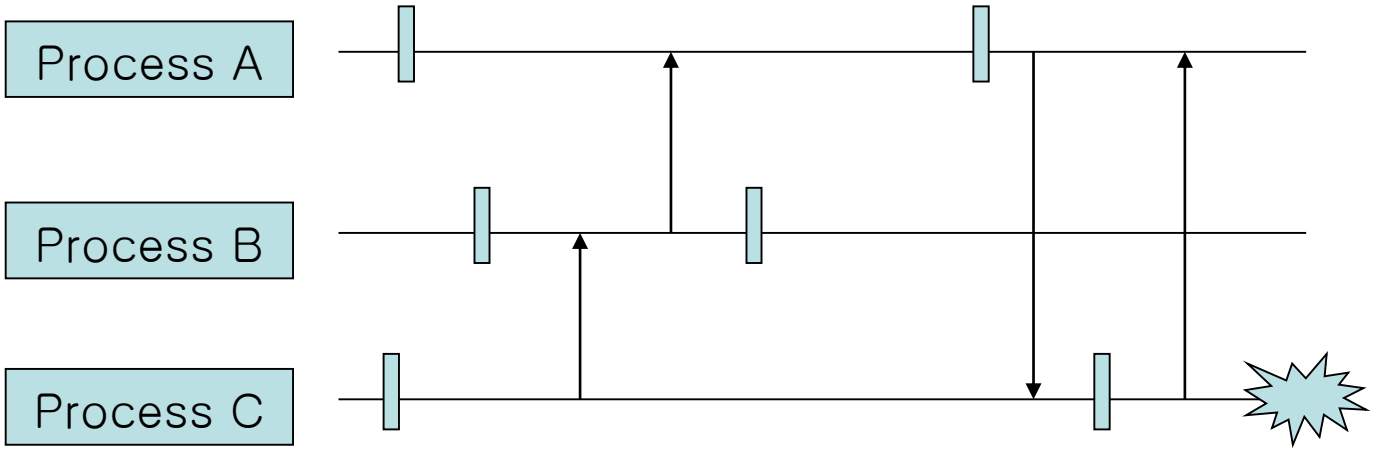
- Concept
  - When an error occurred, proceed to the point that the result is known.
  - The forward error recovery point must be well-defined.
  - Eg.: An occasional missed response to a sensor input is tolerable.
  - Highly application dependant

# Checkpointing

- Concept
  - Some subset of the system is saved at specific points during process execution.
  - Rollback is part of the actual recovery process and occur after the repair (or by reconfiguration).
  - Comparison of # of ckpts and overhead
    - # of ckpts high low
    - Execution overhead high low
    - Fault recovery overhead low high
- Relevant issues
  - Selecting ckpts to minimize the amount of information
  - Deciding which information must be backed up
  - Restraining multiple rollbacks that arise when multiple concurrent processes communicate each other (in multiprocessing or distributed computing environments) ➔ known as domino effect
  - Avoiding error latency situations in which the validity of the state saved at the checkpoint is jeopardized by the undetected error.

# Checkpointing

- Domino effect
  - Happens among communicating processes
  - Eg: Fault occurs at process C.
    - Process A returns to the 2nd ckpt. (no message guarantee)
    - Process C returns to the 1st ckpt (no message guarantee)
    - All processes returns to their 1st ckpts.

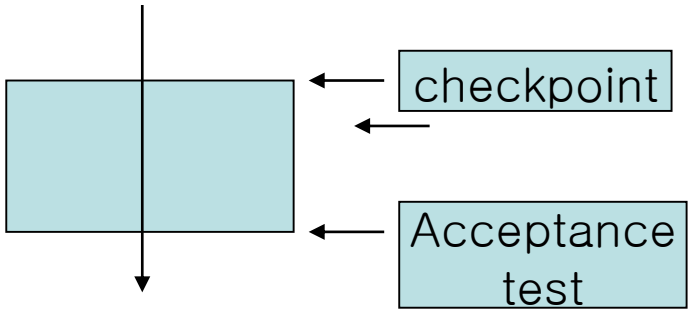


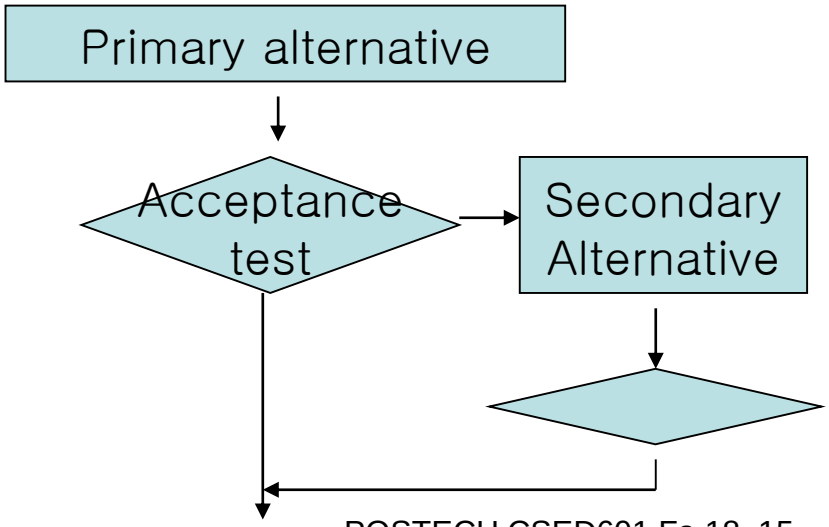
# Journaling

- Concept
  - Copy initial data and record all transactions after that.
  - Recreation of environment when an error occurs.
  - Simplest and least efficient method.
  - Require the largest recovery time.
  - Commonly used as the secondary backup solution.
  - Can be used for other purpose (auditing)

# Recovery blocks

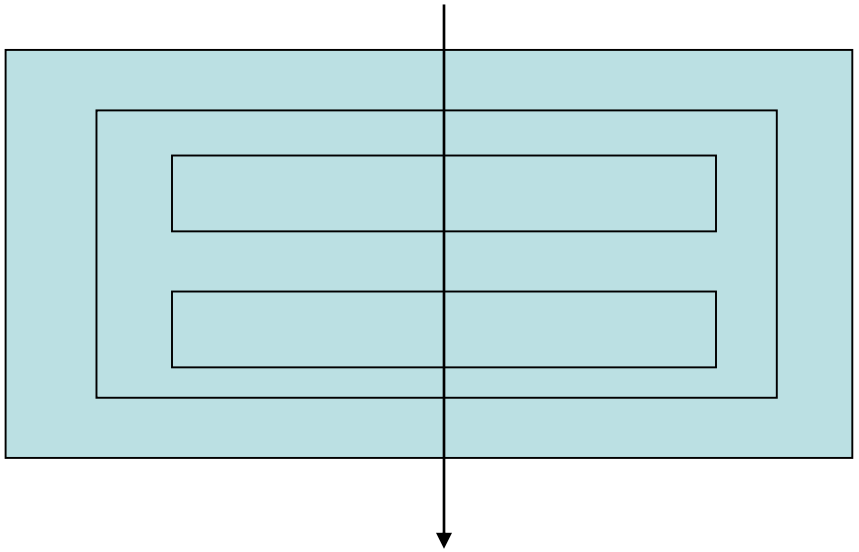
- Concept
  - Combines checkpointing and backup alternative in one block that are similar in nature to the blocks in modern block-structured language
  - Tolerate software design faults, recovery from hard failures and transient errors.
  - Each recovery block contains variables global to the block that will be automatically checkpointed.
  - Graphical presentation



# Recovery blocks

- Properties
  - The success of recovery blocks depends on the ability of acceptance test to successfully detect errors.
  - The extent of acceptance test should be equal to all alternatives.
  - The recovery blocks can be nested.
  - Graphical presentation of nested recovery blocks



# Self-checking programming

- Concept
  - Include some extra code which checks the dynamic behavior of program.
  - Composed of one variant and one acceptance test or two variants.

# Fault-avoidance in software

- Concept
  - Based on software engineering techniques
- Methods
  - Modularity
    - System design is broken into a number of modules that provide a concise interface to a function.
    - Information hiding and defensive programming technique
  - Object-oriented programming
    - A logical growth of modular programming
    - Extra overhead is required in making subroutine calls and returns every time a module boundary is crossed.
  - Capability-based programming
    - Places the protection on the access path.
  - Formal proofs
    - A mechanism proposed to eliminate design errors.

# Fault-avoidance in software

- Other methods
  - Error reporting
  - Regression testing
    - Test values and test structures are systematically selected from a list of alternatives.
  - Reviews
    - Desk checking, peer review, design review, change review
  - Top-down designs
    - Hierarchically structuring a problem and successfully refining each node.

# Summary of software redundancy

- Fault-avoidance
  - Software engineering techniques
- Fault-masking
  - N-version programming
- Fault-tolerance
  - Algorithm construction
  - Forward error recovery
  - Backward error recovery
    - Checkpointing
    - Journaling
    - Recovery blocks