IAF0530/IAF9530

Dependability and fault tolerance

Lectures 5 and 6
Redundancy (Hardware and software)

Gert Jervan
gert.jervan@ati.ttu.ee

Lecture Outline

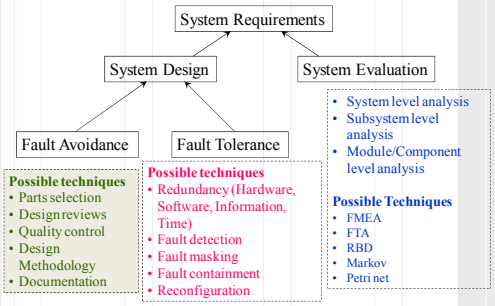
- ✓ **Introduction**
- ✓ **Hardware Redundancy**
- ✓ **Software Redundancy**
- ✓ **Information Redundancy**
- ✓ **Time Redundancy**

Some materials from:
Kewal Saluja
Hongyu Sun
Zaipeng Xie
Meng-Lai Yin
Rajesh Gupta
Elena Dubrova

Fault Tolerance

- A fault-tolerant system is one that can continue to correctly perform its specified tasks in the presence of hardware failures and/or software errors.
- Fault tolerance is the attribute that enables a system to achieve fault-tolerant operation.
- Fault tolerance is not a new field:
  - 1949, the EDVAC computer duplicated the ALU and compare the results
  - 1955, the UNIVAC computer incorporated parity check for data transfers
  - 1952, John von Neumann, lectures on the use of replicated logic modules to improve system reliability,
  - etc.

System Design & Evaluation Top-Level View



Hardware Redundancy

Hardware Redundancy

- 3 basic forms: passive, active, and hybrid
  - Passive: Mask faults rather than detect faults without requiring any system or operator action
  - Active: Fault has to be detected before it can be tolerated. Actions: location, containment, recovery (for component removal)

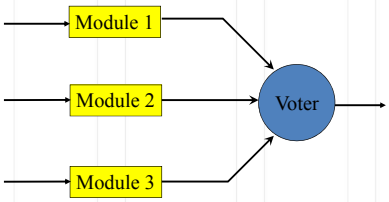
Passive Hardware Redundancy

- Use fault masking to hide the occurrence of faults and prevent the faults from resulting in errors
- Mask faults rather than detect faults
- Achieve fault tolerance without requiring any system or operator action
- Voting mechanisms, majority voting
- Do not need fault detection or reconfiguration
- Many drawbacks

Passive Hardware Redundancy

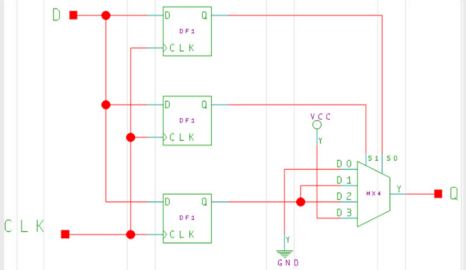
- N-Modular Redundancy (generalization of TMR or Triple Modular Redundancy)
- TMR: Triplicate the hardware and perform a majority vote to determine the output of the system
  - If one of the modules becomes faulty, the 2 remaining fault-free modules mask the results of the faulty module when the majority vote is performed

TMR Technique



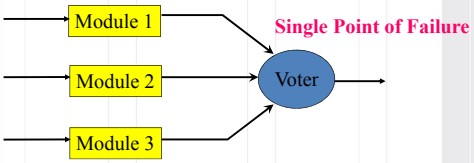
Tolerates N/2 faults

TMR/Voter Structures



Fault-Tolerance Capability

- Assuming perfect voter, how many module faults can the TMR technique tolerate?
- What if 2 modules fail the same way?
- Does TMR technique provide fault detection capability?
- How about imperfect voter?
- Performance impacts from the voter in the TMR technique

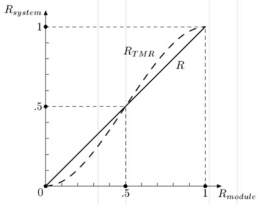


Reliability of a TMR System

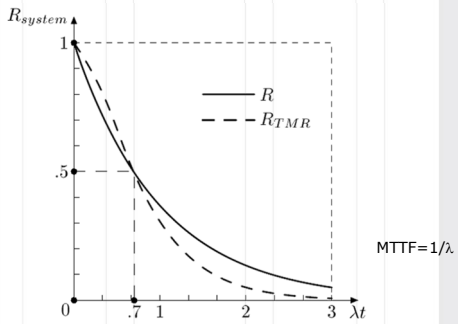
$$R_{TMR} = R_1R_2R_3 + (1-R_1)R_2R_3 + R_1(1-R_2)R_3 + R_1R_2(1-R_3)$$

$$R_1 = R_2 = R_3 = R$$

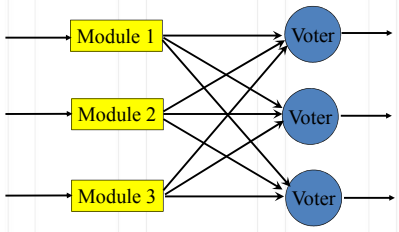
$$R_{TMR} = 3R^2 - 2R^3$$



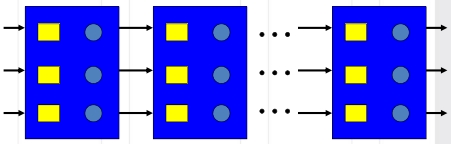
Reliability of a TMR System



TMR with Triplicated Voters



Cascading TMR modules



Passive hardware redundancy

- Types of voting
  - Majority
    - in many practical situations it is meaningless
  - Average
    - can have poor performance if a sensor always provide very low value
  - Mid value
    - a good choice - can be very costly to implement in HW

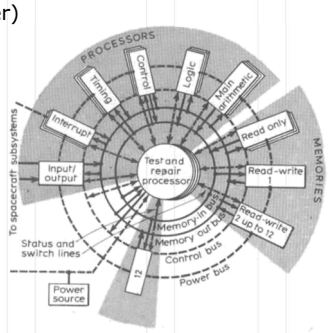
Passive Hardware Redundancy

- Comparison between hw and sw voter schemes

	HW	**SW**
cost	high	low
flexibilty	inflex	flex
synch.	tightly	loosely
perform.	high (fast)	low (slow)
types of voting	majority (others costly)	diff (no extra cost)

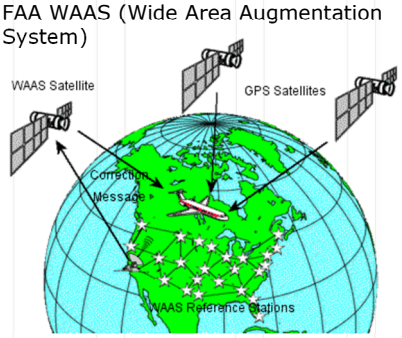
Example Systems Using TMR Technique

- JPL STAR (Self-Testing And Repairing computer)

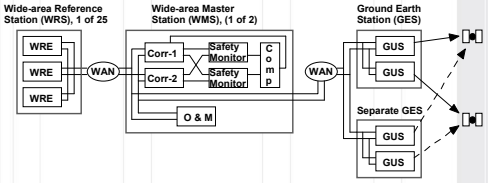


Example Systems Using TMR Technique

- FAA WAAS (Wide Area Augmentation System)



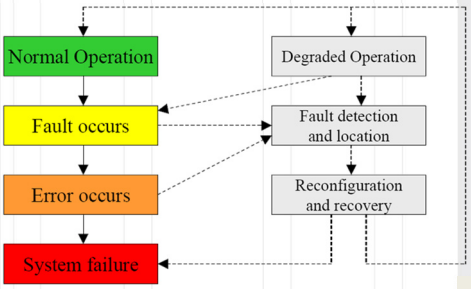
WAAS Block Diagram



Active Hardware Redundancy

- Achieve fault tolerance by detecting the existence of faults and performing some action to remove the faulty parts
- Require the system be reconfigured to tolerate faults
- 3 steps: fault detection, fault location, and fault recovery

Active Hardware Redundancy

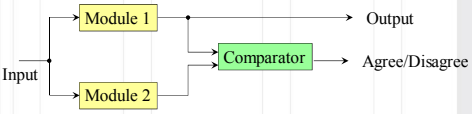


Dynamic Redundancy

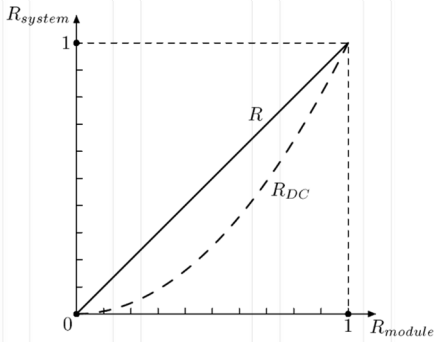
- Uses Extra Components
- Only 1 Copy Operates At A Times
  - Fault Detection
  - Fault Recovery
- Spares Are On "Standby"
  - Hot Spares
  - Cold Spares

Duplication with Comparison

- Both modules perform the same computations in parallel and compare the results
- An error message is generated if the two results disagree
- Only fault detection, no fault tolerance
- Can be used as a fundamental fault detection technique in active redundancy approach, for example, the pair-and-a-spare technique



Reliability of duplication with comparison



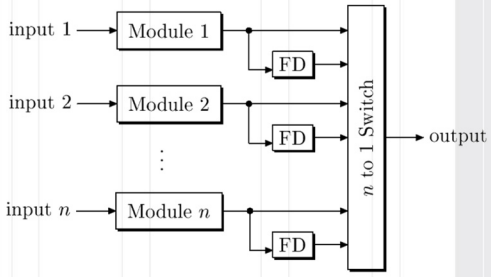
Duplication with Comparison

- Problems:
  - if there is a fault on input line, both modules will receive the same erroneous signal and produce the erroneous result
  - comparator may not be able to perform an exact comparison
    - synchronisation
    - no exact matching
  - comparator is a single point of failure

Implementation of comparator

- In hardware, a bit-by-bit comparison can be done using two-input exclusive-or gates
- In software, a comparison can be implemented with a COMPARE instruction
  - commonly found in instruction sets of almost all microprocessors

Standby Sparing



Spares

- Hot spares
  - all modules are powered up
  - spares can be switched into use immediately after the primary module becomes failed
- Cold spares
  - the primary modules are powered up
  - the spares are powered down, which are powered up and switched into use when the primary modules fail
- Warm spares

Standby Sparing (standby replacement)

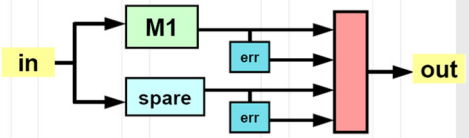
- Active hardware redundancy
- One module is operational and one or more modules serve as standbys (or spares)
- Various fault detection or error detection schemes are used to determine whether a module has become faulty
- Fault location is used to determine exactly which module, if any, is faulty.

Standby Sparing (standby replacement)

- If a fault is detected and located, then the faulty module is removed from operation and replaced with a spare
- The reconfiguration can be viewed as a switch.
- Can bring a system back to full operation after the occurrence of a fault.
- Require momentary disruption in performance when reconfiguration is performed.

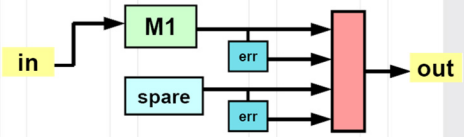
Hot Standby Sparing

- In hot standby sparing spares operate in synchrony with on-line module and are prepared to take over any time



Cold Standby Sparing

- In cold standby sparing spares are unpowered until needed to replace a faulty module



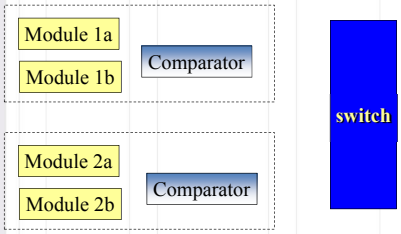
Hot & Cold Standby Sparing

- Hot standby sparing can minimize the performance disruption. The spares operate in synchrony with the on line modules and are prepared to take over at any time.
- In cold standby sparing, the spares are unpowered until needed to replace a faulty module. Hence extra time is required to bring the module back to operation. The advantage is that spares do not consume power until needed. Satellite application is a good example for cold standby sparing.

Pair-and-a-spare Technique

- Combine the features in standby sparing and duplication with comparison
- 2 modules are operated in parallel at all times and their results are compared to provide the error protection capability
- The error signal from the comparison is used to initiate the reconfiguration process (switch) that removes faulty modules and replaces them with spares

Pair-and-a-spare scheme



http://www.stratus.com/

Example Systems

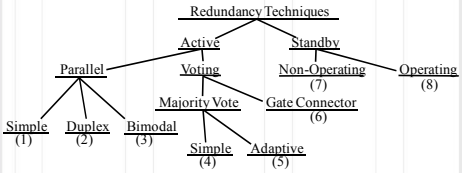
- Apollo telescope mount pointing computer
- Saturn 5 LVDC memory section
- Compaq Himalaya architecture

Types of Redundancy

NASA Office of Logic Design - klabs.org

- Classified on how the redundant elements are introduced into the circuit
- Choice of redundancy type is application specific
- Active or Static Redundancy
  - External components are not required to perform the function of detection, decision and switching when an element or path in the structure fails.
- Standby or Dynamic Redundancy
  - External elements are required to detect, make a decision and switch to another element or path as a replacement for a failed element or path.

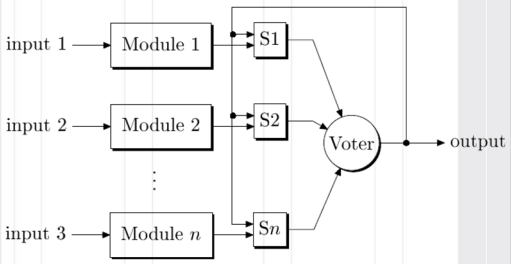
Redundancy Techniques



Hybrid Hardware Redundancy

- Hybrid:
  - combine the attractive features of both the passive and active approaches
    - fault masking
    - fault detection
    - fault location
    - recovery

Self-Purging Redundancy



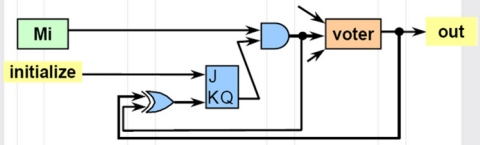
Can mask n-2 module faults

Self Purging Redundancy

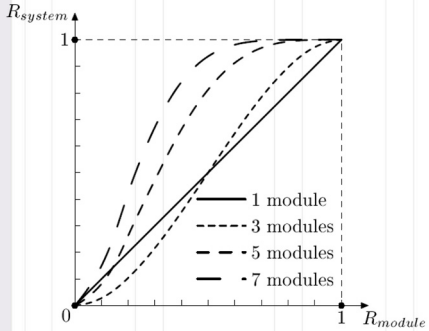
- Initially start with NMR
- Purge one unit at at time till arrive at TMR
  - can tolerate more faults initially compared to NMR with spare
  - cost of the switch - higher?

Basic Structure of a Switch

- If output of a module disagrees with the output of the system, its contribution to the voter is forced to be 0 (threshold voter)



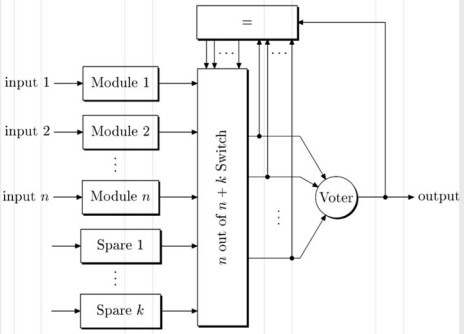
Reliability of Self-Purging System



N-Modular Redundancy with Spares

- Most hybrid redundancy are based on the concept of N-modular redundancy (NMR) with spares
- The idea is to provide N modules arranged in a voting configuration
- Spares are provided to replace failed modules
- The advantage of NMR with spares is that a voting configuration can be restored after a fault has occurred

N-Modular Redundancy with Spares



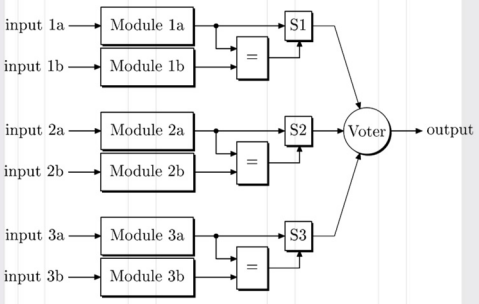
NMR with Spares

- System remains in the basic NMR configuration until the disagreement vector determines a fault
- The output of the voter is compare to the individual outputs of the modules
- Module which disagrees is labeled as faulty and removed from the NMR core
- Spare is switched to replace it

NMR with Spares

- The reliability is maintained as long as the pool of spares is not exhausted
- 3-modular redundancy with 1 spare can tolerate 2 faults
- To do it in a passive approach, we would need to have 5 modules

Triplex-duplex Redundancy



Triplex-duplex Redundancy

- TMR allows faults to be masked
  - performance without interruption
- Duplication with comparison allows faults to be detected and faulty module removed form voting
  - removal of faulty module allows to tolerate future faults
- Two module faults can be tolerated

Software Fault Tolerance

Introduction

- Less understood and less mature than in hardware
- Software does not degrade over time
- Design faults
- Environment

Introduction

- Many current techniques for software fault tolerance attempt to leverage the experience of hardware redundancy schemes
  - software N-version programming closely resembles hardware N-modular redundancy
  - recovery blocks use the concept of retrying the same operation in expectation that the problem is resolved after the second try.

Problems

- Traditional hardware fault tolerance techniques were developed to fight
  - permanent components faults primarily
  - transient faults caused by environmental factors secondarily.
- They do not offer sufficient protection against design and specification faults, which are dominant in software.

Concepts for Traditional SFT

- Software design and implementation errors cannot be detected by simple replication of identical software units, assuming the same inputs are provided to each copy.
- Some form of diversity must accompany the redundancy
  - Software redundancy → Design diversity
  - Information or data redundancy → Data diversity
  - Temporal redundancy → Temporal diversity
  - Environment diversity
  - Hardware redundancy

Single- and multi-version

- Software fault-tolerance techniques can be divided into two groups:
  - single-version
  - multi-version
- Single version techniques aim to improve fault tolerant capabilities of a single software module
  - fault detection, containment and recovery mechanisms
- Multi-version techniques employ redundant software modules, developed following design diversity rules

Redundancy Allocation

- A number of possibilities have to be examined:
  - at which level the redundancy need to be provided
- Redundancy can be applied to a procedure, or to a process, or to the whole software system
  - which modules are to be made redundant
- Usually, the components which have high probability of faults are chosen to be made redundant.
- The increase in complexity caused by redundancy can be quite severe and may diminish the dependability improvement

Single-Version (Dynamic) Techniques

- Dynamic redundancy kicks in only when an error is detected.
- Four phases
  - 1. Error detection:
    fault tolerance techniques effective only when an error is detected
  - 2. Damage assessment and containment:
    to what extent the "damage" has spread because of the delay between a fault and its manifestation/detection?
  - 3. Error recovery:
    techniques to reach from a corrupted to a safe state
  - 4. Fault treatment and continued service:
    error correction.

1 - Error Detection

- The goal is to determine that a fault has occurred within a system.
- Various types of acceptance tests are used to detect faults
  - the result of a program is subjected to a test
  - if the result passes the test, the program continues its execution
  - a failed test indicates a fault

Acceptance Test

- Acceptance test is most effective if it can be calculated in a simple way and if it is based on criteria that can be derived independently of the program application.
- The existing techniques include
  - timing checks
  - coding checks
  - reversal checks
  - reasonableness checks
  - structural checks
  - replication checks
  - dynamic reasonableness checks

Timing Checks

- Timing checks are applicable to system whose specification include timing constrains
- Based on these constrains, checks are developed to indicate a deviation from the required behavior.
  - Watchdog timer is an example of a timing check
  - Watchdog timers are used to monitor the performance of a system and detect lost or locked out modules.

61

Coding Checks

- Coding checks are applicable to system whose data can be encoded using information redundancy techniques
- Usually used in cases when the information is merely transported from one module to another without changing it content.
  - Arithmetic codes can be used to detect errors in arithmetic operations

62

Reversal Checks

- In some system, it is possible to reverse the output values and to compute the corresponding input values.
- A reversal checks compares the actual inputs of the system with the computed ones.
  - a disagreement indicates a fault.

63

Reasonableness Checks

- Reasonableness checks use semantic properties of data to detect fault.
  - a range of data can be examined for overflow or underflow to indicate a deviation from system's requirements
- Maximum withdrawal sum in bank's teller machine
- Address generated by a computer should lie inside the range of available memory

64

Structural Checks

- Structural checks are based on known properties of data structures
  - a number or elements in a list can be counted, or links and pointer can be verified
- Structural checks can be made more efficient by adding redundant data to a data structure,
  - attaching counts on the number of items in a list, or adding extra pointers

65

2 - Damage Assessment & Containment

- Necessary due to the delay between fault and error
- Goal of containment is to minimize damage caused by a faulty component
  - "firewalling"
- Assessment closely related to containment techniques used
- Techniques for fault containment:
  - modularization
  - partitioning
  - system closure
  - atomic actions

66

Modularization

- Software system is divided into modules with few or no common dependencies between them
- Modularization attempts to prevent the propagation of faults
  - by limiting the amount of communication between modules to carefully monitored messages
  - by eliminating shared resources

Partitioning

- Modular hierarchy of a software architecture is partitioned in horizontal or vertical dimensions
- Horizontal partitioning separates the major software functions into independent branches
  - The execution of the functions and the communication between them is done using control modules
- Vertical partitioning distributes the control and processing function in a top-down hierarchy.
  - High-level modules normally focus on control functions, while low-level modules perform processing

System Closure

- System closure technique is based on a principle that no action is permissible unless explicitly authorized
- In an environment with many restrictions and strict control all the interactions between the elements of the system are visible
  - prison
- It is easier to locate and disable any fault.

Atomic Action

- An atomic action among a group of components in an activity in which the components interact exclusively with each other.
  - no interaction with the rest of the system
- Two possible outcomes of an atomic action:
  - it terminates normally
  - it is aborted upon a fault detection
- Fault containment area is defined and fault recovery is limited to atomic action components

3 Fault Recovery

- Once a fault is detected and contained, a system attempts to recover from the faulty state and regain operational status
  - If fault detection and containment mechanisms are implemented properly, the effects of the faults are contained within a particular set of modules at the moment of fault detection.
- The knowledge of fault containment region is essential for the design of effective fault recovery mechanism

Exception Handling

- Exception handling is the interruption of normal operation to handle abnormal responses
- Possible events triggering the exceptions:
  - Interface exceptions
    - signaled by a module when it detects an invalid service request
  - Local exceptions
    - signaled by a module when its fault detection mechanism detects a fault
  - Failure exceptions
    - signaled by a module when it has detected that its fault recovery mechanism is enable to recover successfully

Recovery

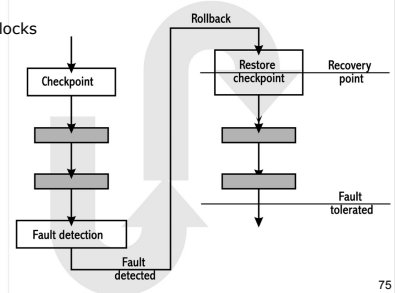
- Forward or Backward
- Forward: continues from an erroneous state by making selective corrections to the system state
  - includes making safe the controlled environment which may be hazardous or damaged because of failure
  - system specific and depends upon accurate predictions
  - e.g., redundant pointers in data structures, self-correcting codes

Recovery

- Backward: relies on restoring the system to a previous safe state and executing an alternative section of the program
  - safe functionality but different algorithm
  - the point to which a process is restored is called a recovery point and the act of establishing it is called checkpointing.
  - BER can be used to recover from unanticipated faults including design errors.
  - State restoration is not always possible in (real-time) embedded systems.

Backward Recovery

- ✓ Attempts to return the system to a correct or error-free state.
- ✓ For transient faults
- ✓ Example: recovery blocks (RcB)



Static Checkpoints

- A static checkpoint takes a single snapshot of the system state at the beginning of the program execution and stores it in the memory.
  - If a fault is detected, the system returns to this state and starts the execution from the beginning.
  - Fault detection checks are placed at the output of the module

Dynamic Checkpoints

- Dynamic checkpoints are created dynamically at various points during the execution
  - If a fault is detected, the system returns to the last checkpoint and continues the execution.
  - Fault detection checks need to be embedded in the code and executed before the checkpoints are created

Static vs. Dynamic

- In static approach, the expected time to complete the execution grows exponentially with the execution requirements.
  - static checkpointing is effective only if the processing requirement is relatively small.
- In dynamic approach, it is possible to achieve linear increase in execution time as the processing requirements grow

Strategies for dynamic checkpointing

- Equidistant
  - places checkpoints at deterministic fixed time intervals
  - the time between checkpoints is chosen depending on the expected fault rate
- Modular
  - places checkpoints at the end of the sub-modules in a module, after the fault detection checks for the submodule are completed
  - the execution time depends on the distribution of the sub-modules and expected fault rate
- Random

Advantages

- Conceptually simple
- Independent of the damage caused by a fault
- Applicable to unanticipated faults
- General enough to be used at multiple levels in a system

Problems

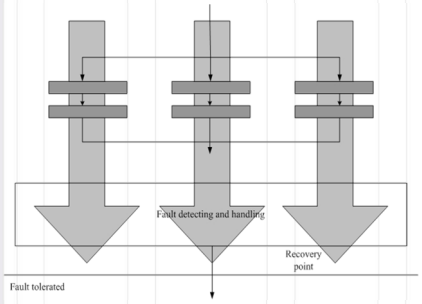
- Non-recoverable actions exist in some systems
  - these actions cannot be compensated by simply reloading the state and restarting the system
    - firing a missile
    - soldering a pair of wires
- The recovery from such actions can be done
  - by compensating for their consequences
    - undoing a solder
  - by delaying their output until after additional confirmation checks are completed
    - do a friend-or-foe confirmation before firing

Forward Recovery

- Attempts to find a new state from which the system can continue operation.
- Utilize error compensation based on redundancy to select or derive the correct answer or an acceptable answer.
- Example: N-version programming (NVP), N-copy programming (NCP), and the distributed recovery block (DRB)

Forward Recovery

- Efficient for predictable errors



4 - Fault Treatment and Continued Service

- Even with recovery, the error may recure. Need to eradicate the fault from the system
- Automatic treatment of faults is very application specific
- Make some assumptions. For instance:
  - all faults are transient
- Fault treatment in two stages
  - Fault location
  - System repair
- Fault location
  - use error detection techniques to trace a fault to a component (hardware or software)
  - System repair
    - sometimes it has to be done while the system is in operation.