

Fault Tolerance 101

Basis Concepts of Dependable Computing

Dr. Zbigniew Kalbarczyk

Center for Reliable and High-Performance Computing

Coordinated Science Laboratory

University of Illinois at Urbana-Champaign

kalbarcz@illinois.edu

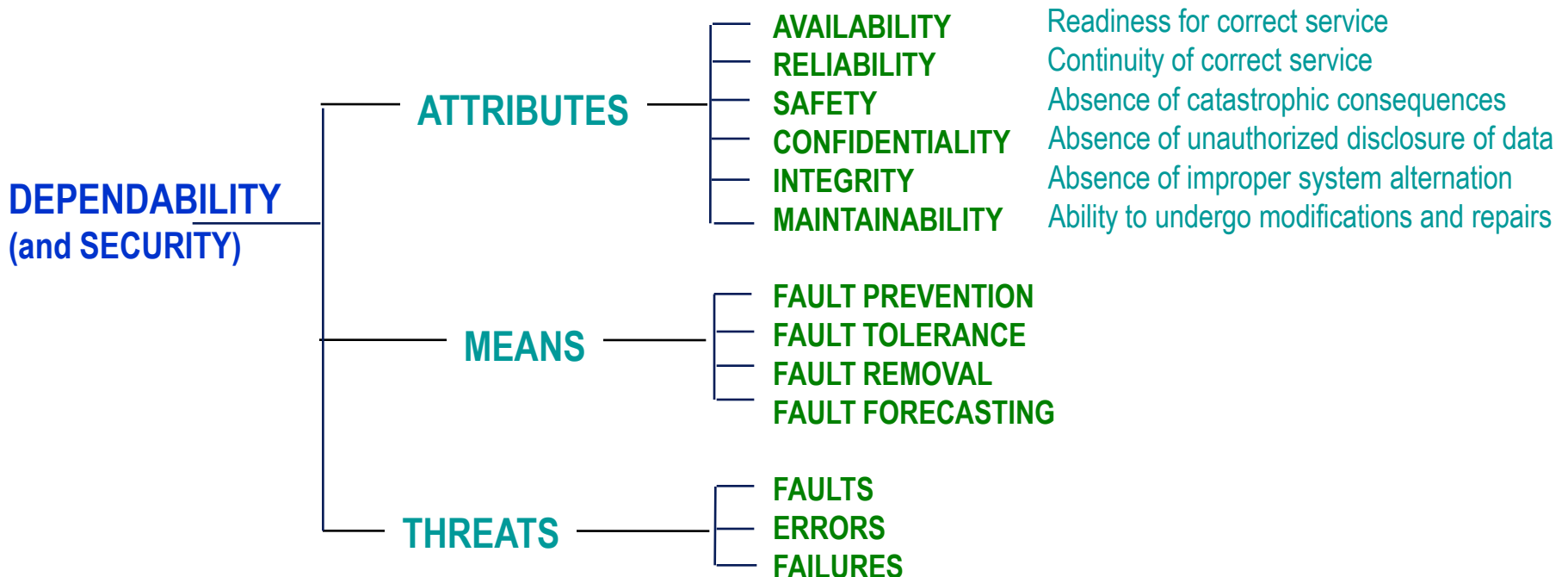


ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

<http://www.crhc.uiuc.edu/DEPEND>

Dependable Computing

- Original definition of dependability (that stresses the need for justification of trust) states that: *the dependability is the ability to deliver service that can justifiably be trusted*
- The alternate definition (that provides the criterion for deciding if the service is dependable) states that: *the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable*



Fault Classes

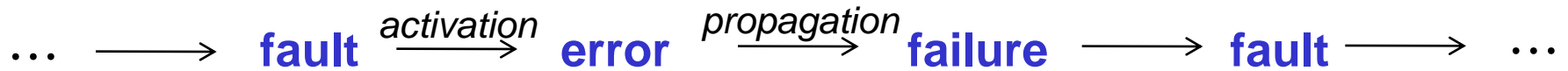
Based on the temporal persistence

- **Permanent** faults, whose presence is continuous and stable.
- **Intermittent** faults, whose presence is only occasional due to unstable hardware or varying hardware and software states (e.g., as a function of load or activity).
- **Transient** faults, resulting from temporary environmental conditions.

Based on the origin

- **Physical** faults, stemming from physical phenomena internal to the system, such as threshold change, shorts, opens, etc., or from external changes, such as environmental, electromagnetic, vibration, and radiation.
- **Human-made** faults, which may be either design faults, introduced during system design, modification, or establishment of operating procedures, or interaction faults, which are violation of operating or maintenance procedures.

Fundamental Chain of Dependability



Example 1

- A short in an integrated circuit is a failure (with respect to the function of the circuit)
- The consequence (e.g., stuck at a Boolean value) is a fault that stays dormant until activated
- Upon activation (invoking the faulty component by applying an input) the fault becomes active and produces an error
- If and when the propagated error affects the delivered service (e.g., information content), a failure occurs

Example 2

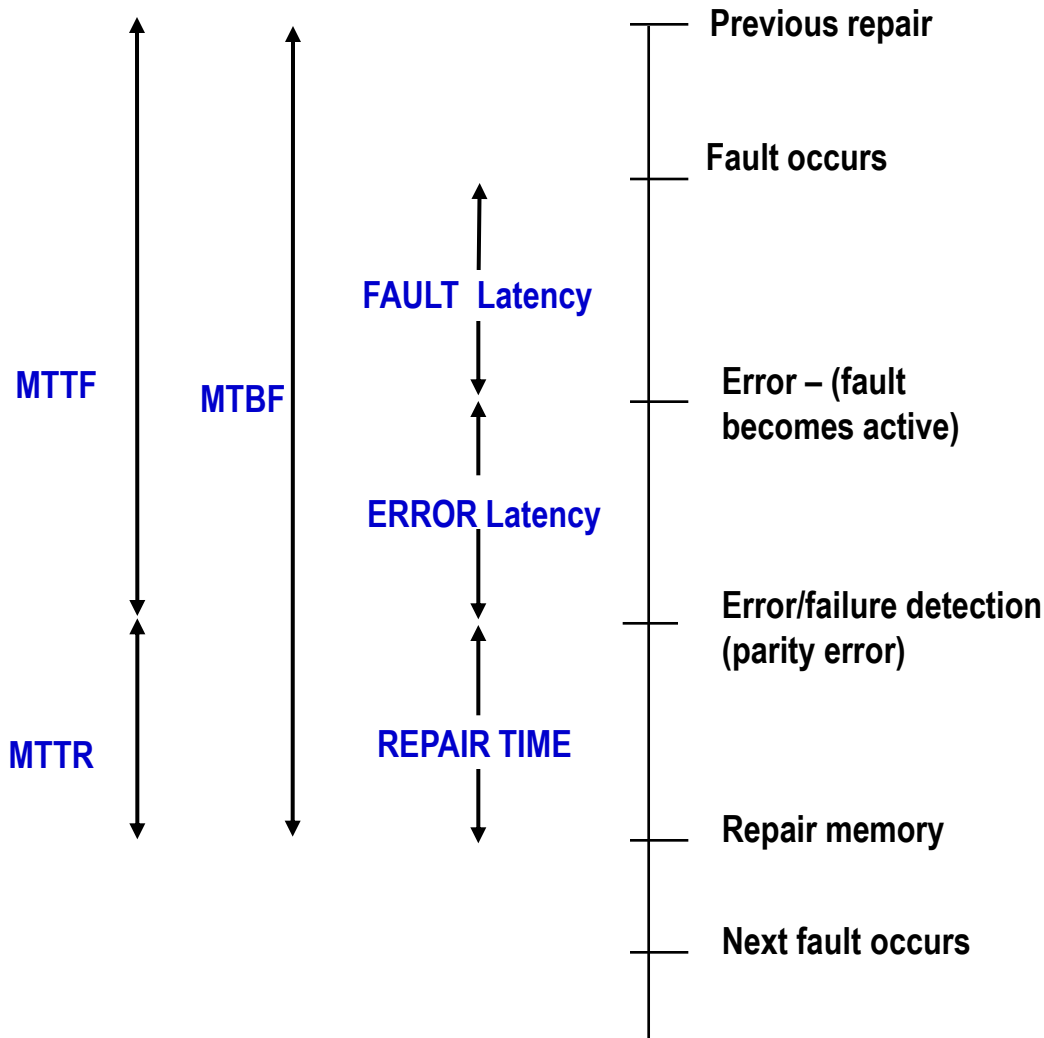
- The result of an error by a programmer leads to a failure to write the correct instruction or data
- This results in a dormant fault in the written software (e.g., faulty instruction)
- Upon activation the fault becomes active and produces an error
- When the error affects the delivered service, a failure occurs

Example 3

- An inappropriate human-system interaction performed by an operator is an external fault (from the system view point)
- Resulting altered processed data is an error,

.....

Fault Cycle & Dependability Measures



Reliability:

a measure of the continuous delivery of service;
R(t) is the probability that the system survives (does not fail) throughout $[0, t]$;
 expected value: **MTTF (Mean Time To Failure)**

Maintainability:

a measure of the service interruption
M(t) is the probability that the system will be repaired within a time less than t ;
 expected value: **MTTR (Mean Time To Repair)**

Availability:

a measure of the service delivery with respect to the alternation of the delivery and interruptions
A(t) is the probability that the system delivers a proper (conforming to specification) service at a given time t .
 expected value: **$EA = MTTF / (MTTF + MTTR)$**

Safety:

a measure of the time to catastrophic failure
S(t) is the probability that no catastrophic failures occur during $[0, t]$;
 expected value:
MTTCF (Mean Time To Catastrophic Failure)

Spatial Redundancy: Hardware

- **Passive hardware redundancy**
 - relies on voting to mask the occurrence of errors
 - can operate without need for error detection or system reconfiguration
 - triple modular redundancy (TMR)
 - N-modular redundancy (NMR),
- **Active hardware redundancy**
 - achieves fault tolerance by error detection, error location, and error recovery
 - duplication and comparison
 - standby sparing – one module is operational and one or more modules serve as standbys or spares
- **Hybrid hardware redundancy**
 - Fault masking used to prevent the system from producing erroneous results
 - fault detection, location, and recovery used to reconfigure the system in the event of an error
 - N-modular redundancy with spares.

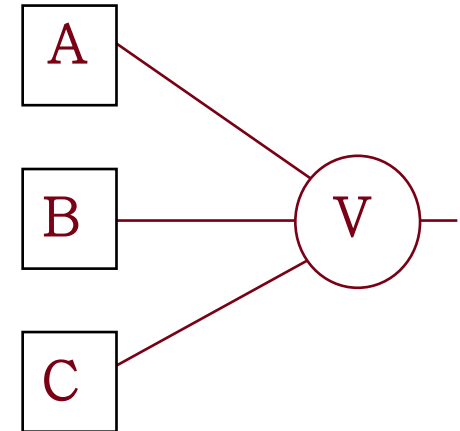
M-out-of-N Systems

- **Passive or masking redundancy**
- Consider *TMR* (Triple Modular Redundancy) system (*2-out-of-3*)
- Out of 3 modules, two need to function for the system to operate correctly

$$R_{TMR} = R_m^3 + \binom{3}{2} R_m^2 (1 - R_m)$$

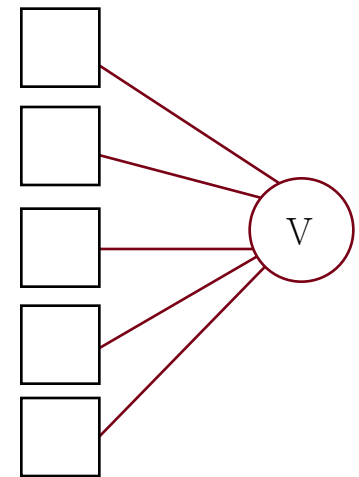
- For general *M-out-of-N* system
- Out of N modules, need M to function

$$R_{MN} = \sum_{i=0}^{N-M} \binom{N}{i} R_m^{N-i} (1 - R_m)^i$$



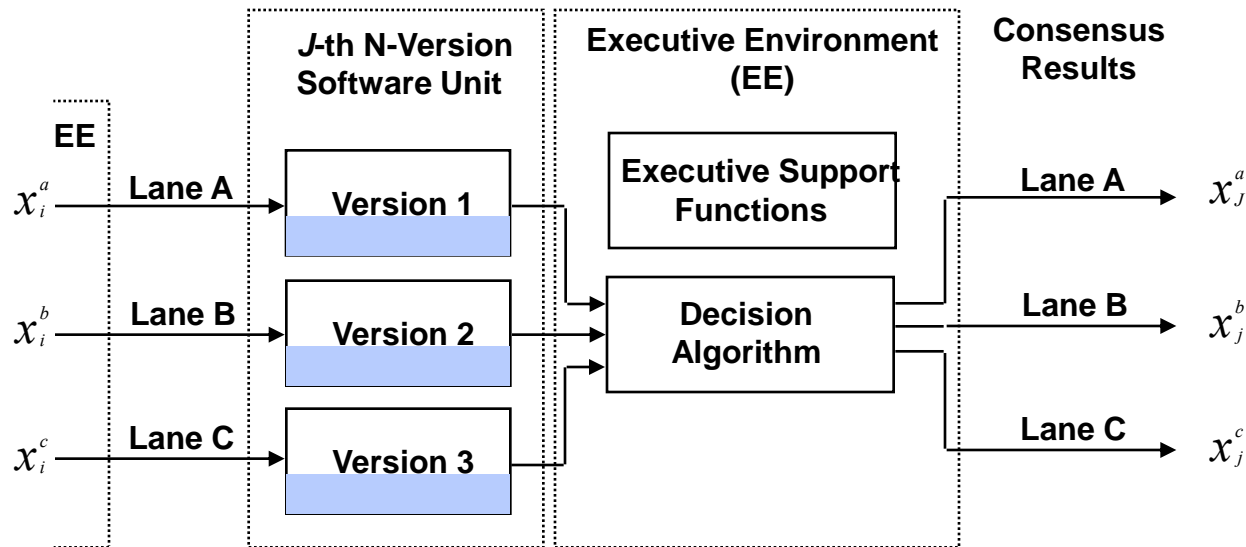
N working
N-1 working
N-2 working
N-M working

Failed



Spatial Redundancy: Software

- Software Replication
 - Concurrent execution of N copies of same software and voting
- N-Version programming
 - Concurrent execution of N version of software and voting



The N-version software model with n=3

Effect of Coverage and Voter

Impact of coverage

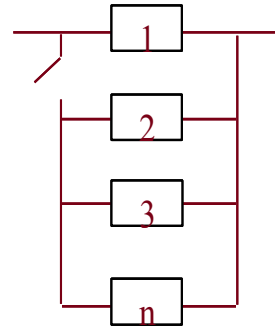
- Error detection is not perfect
- Reconfiguration may not succeed
- Attach a coverage "c"

One spare system

$$R_{sys} = R_1 + c(1-R_1)R_2$$

n-1 spare system

$$R_{sys} = R_m \sum_{i=0}^{n-1} c^i (1-R_m)^i$$



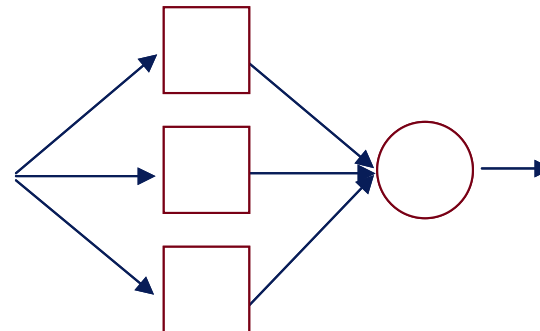
With low coverage, reliability saturates

	R _m = 0.9	R _m = 0.7	R _m = 0.5
C=0.99, n=2	0.989	0.908	0.748
C=0.99, n=4	0.999	0.988	0.931
C=0.99, n=inf	0.999	0.996	0.990
C= 0.8 , n=2	0.972	0.868	0.700
C= 0.8 , n=4	0.978	0.918	0.812
C=0.8, n=inf	0.978	0.921	0.833

Impact of Voter

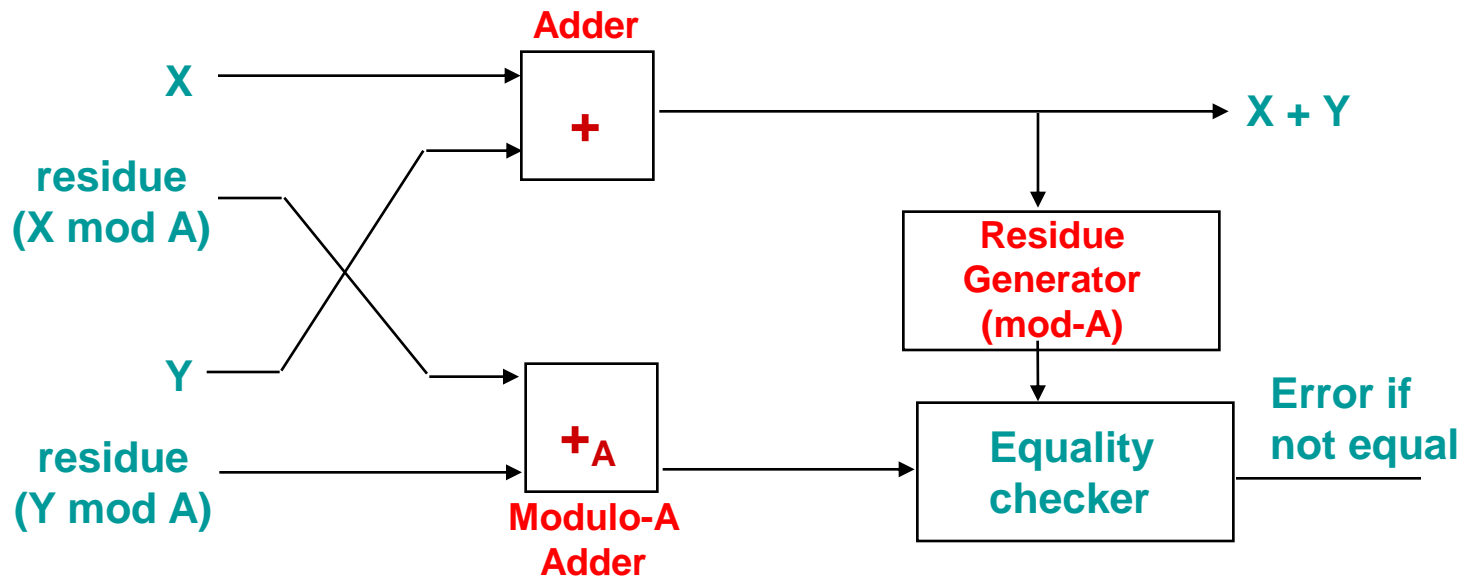
- Assume voter reliability R_v

$$R_{TMRV} = R_v (R_m^3 + \binom{3}{2} R_m^2 (1 - R_m))$$



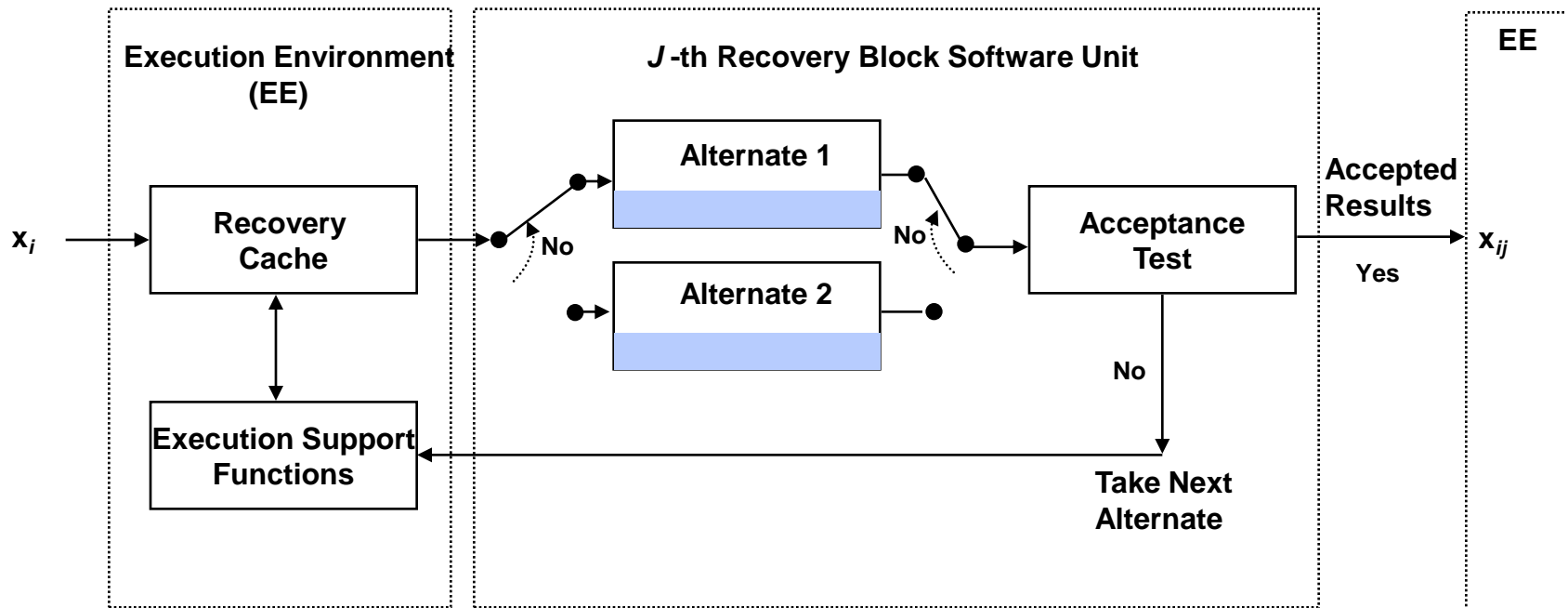
Information Redundancy: Coding Techniques

- *Parity, error correcting codes (e.g., ECC)* – memory protection and disk protection
- *Cyclic redundancy codes* – storage and data communication protection
- *Residue codes* – arithmetic operation protection
 - The property used: $((x \bmod A) + (y \bmod A)) \bmod A = (x+y) \bmod A$



Time/Information Redundancy: Recovery Blocks

- Failure of the primary software version (alternate 1) triggers execution of a new alternate from the checkpoint (recovery cache) on the same hardware



Time/Information Redundancy: Checkpoint and Rollback

- **Applicability**
 - When time redundancy is allowed
 - To transient hardware and many software design faults (e.g., timing faults)
 - To both nonredundant and redundant architectures
- **Checkpointing**
 - Maintains/saves precise system state or a “snapshot” at regular intervals
 - Snapshot can be as small as one instruction
 - Typically, checkpoint interval includes many instructions
 - May not be ideal when there is much error detection latency
- **Rollback recovery**
 - When error is detected
 - Roll back (or restore) process(es) to the saved state, i.e., a checkpoint
 - Restart the computation

Recovery in Distributed Systems: Synchronous & Asynchronous Checkpointing

Synchronous Checkpointing

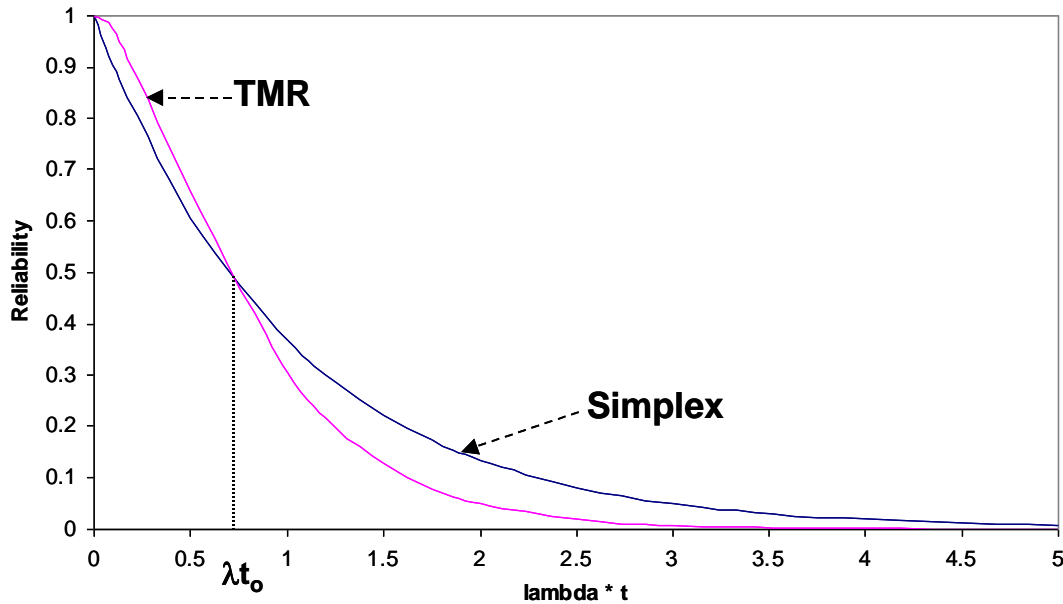
- All process synchronized their actions of taking checkpoints
- The set of local checkpoints constitutes a consistent system state
- **Disadvantages:**
 - additional messages must be exchanged to coordinate checkpointing
 - Synchronization delays are introduced during normal operations.

Asynchronous Checkpointing

- Checkpoints at each process are taken independently without any synchronization among the processors.
- **Disadvantages:**
 - There is no guarantee that a set of local checkpoints taken will be a consistent set of checkpoints.
 - The recovery algorithm must search for the most recent consistent set of checkpoints before it initiates recovery.

Pitfalls Using Single Measure

Compare reliability of simplex and TMR systems



$$R_{\text{simplex}}(t) = e^{-\lambda t}$$

$$MTTF_{\text{simplex}} = \int e^{-\lambda t} dt = 1 / \lambda$$

$$R_{TMR}(t) = e^{-3\lambda t} + \binom{3}{2} e^{-2\lambda t} (1 - e^{-\lambda t})$$

$$MTTF_{TMR} = \frac{3}{2\lambda} - \frac{2}{3\lambda} = \frac{5}{6\lambda}$$

$$MTTF_{\text{simplex}} > MTTF_{TMR}$$

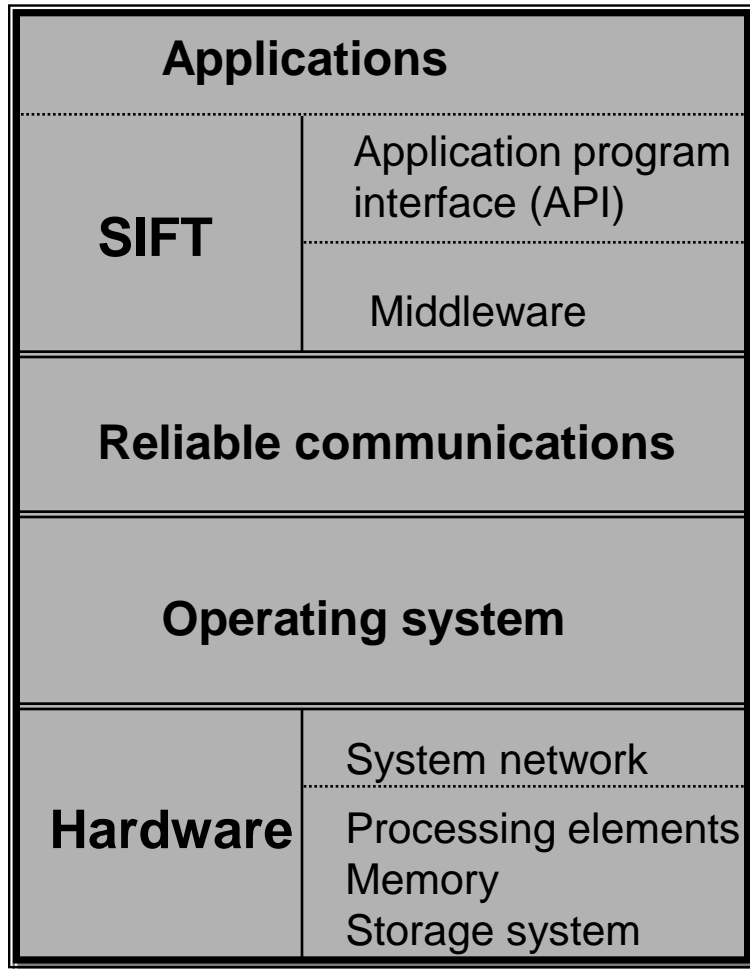
$$R_{TMR}(t) \geq R(t) \quad 0 \leq t \leq t_0$$

$$R_{TMR}(t) \leq R(t) \quad t_0 \leq t < \infty$$

$$\text{where } t_0 = \frac{\ln 2}{\lambda} \approx \frac{0.7}{\lambda}$$

- Instead of MTTF, look at mission time
- Reliability of M-out-of-N systems very high in the beginning
 - spare components tolerate failures
- Reliability sharply falls down in end
 - system exhausted redundancy, more hardware can possibly fail
- Such systems useful in aircraft control
 - very high short time reliability
 - 0.99999 over 10 hour period

System View of Dependable Computing



What can be provided in software and application itself?

What can be provided in the communication layer?

What can be provided in the operating system?

What can be provided in hardware to ensure fail-silent behavior of system components ?

How to combine hardware and software fault tolerance techniques - (1) fast error detection in hardware, (2) high efficiency detection and recovery in software
How to assess whether the achieved availability meets system requirements

Recommended Texts

- [Prad96] D.K. Pradhan, ed., *Fault Tolerant Computer System Design*, Prentice-Hall, 1996
- [John89] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison Wesley, 1989
- [SiSw92] D.P. Siewiorek and R.S. Swarz, *Reliable Computer Systems - Design and Evaluation*, Digital Press (distributed by Butterworth), 1992, 2nd edition.
- [Lyu95a] M.R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1995
- [Lyu95b] M.R. Lyu, ed., *Software Fault Tolerance*, J. Wiley & Sons, 1995
- [SiSh94] M. Singhal and N.G. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw-Hill, 1994