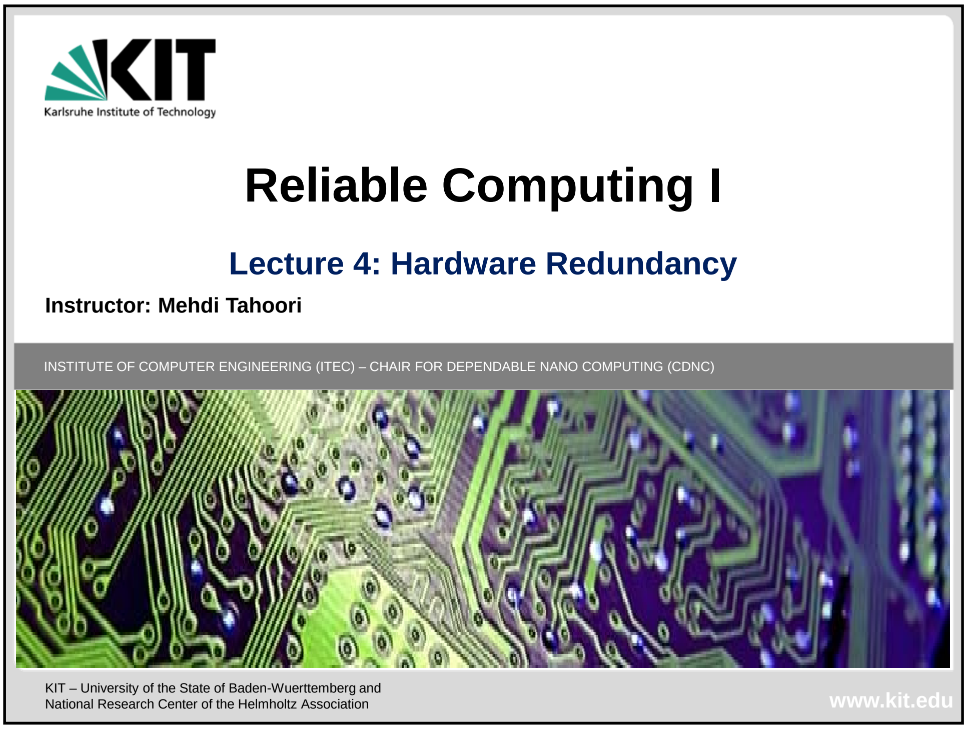


Reliable Computing I

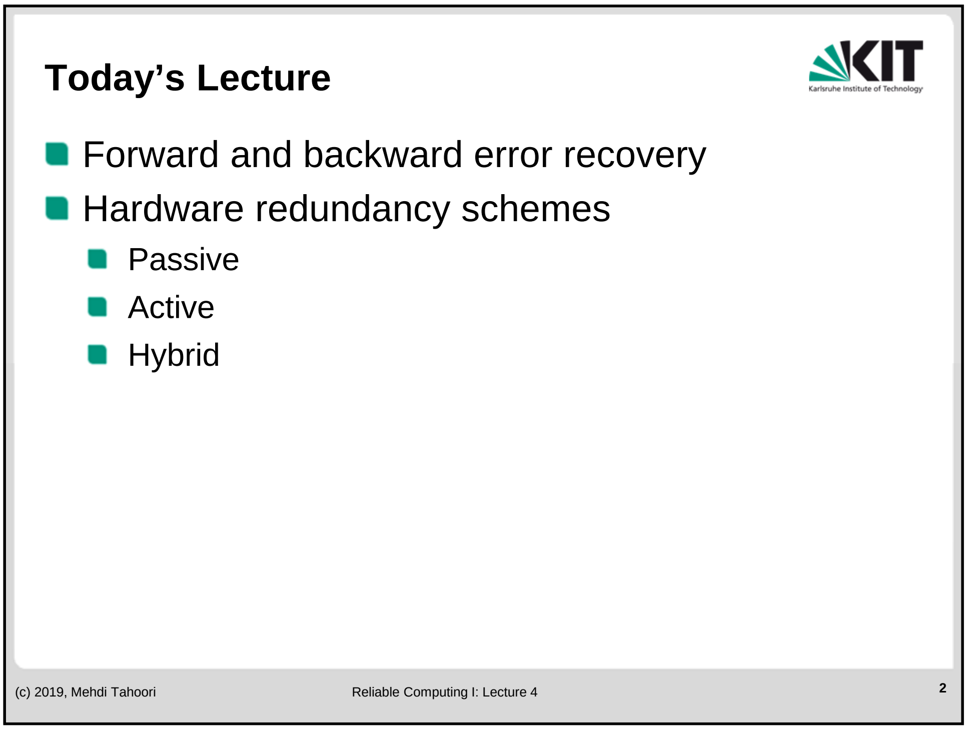
Lecture 4: Hardware Redundancy

Instructor: Mehdi Tahoori

INSTITUTE OF COMPUTER ENGINEERING (ITEC) – CHAIR FOR DEPENDABLE NANO COMPUTING (CDNC)

KIT – University of the State of Baden-Wuerttemberg and
National Research Center of the Helmholtz Association

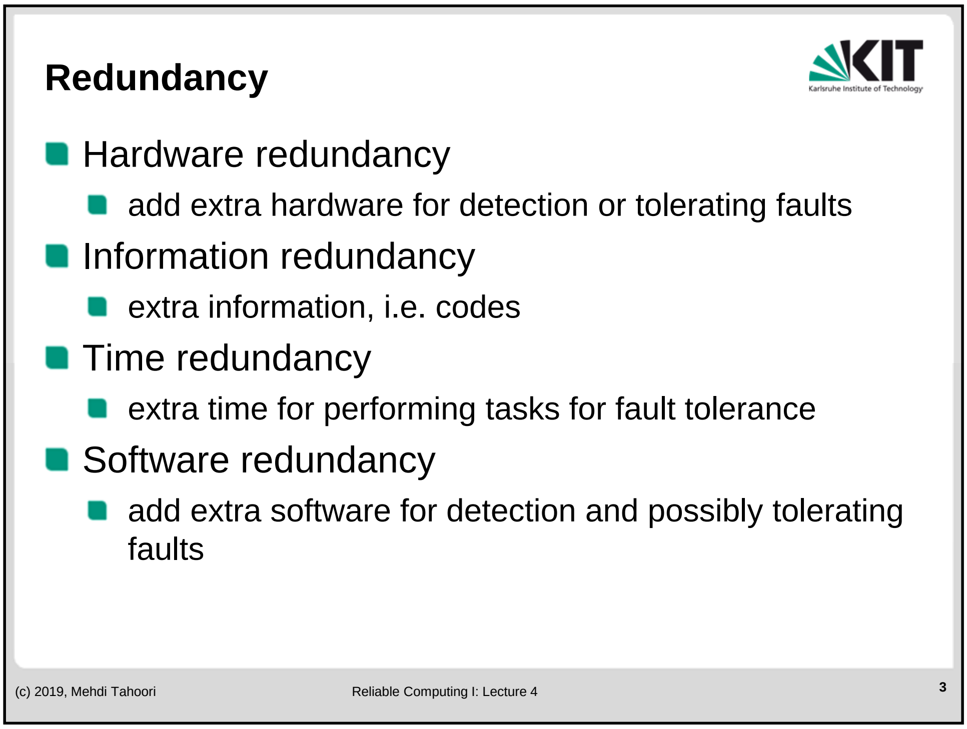
www.kit.edu



Today’s Lecture

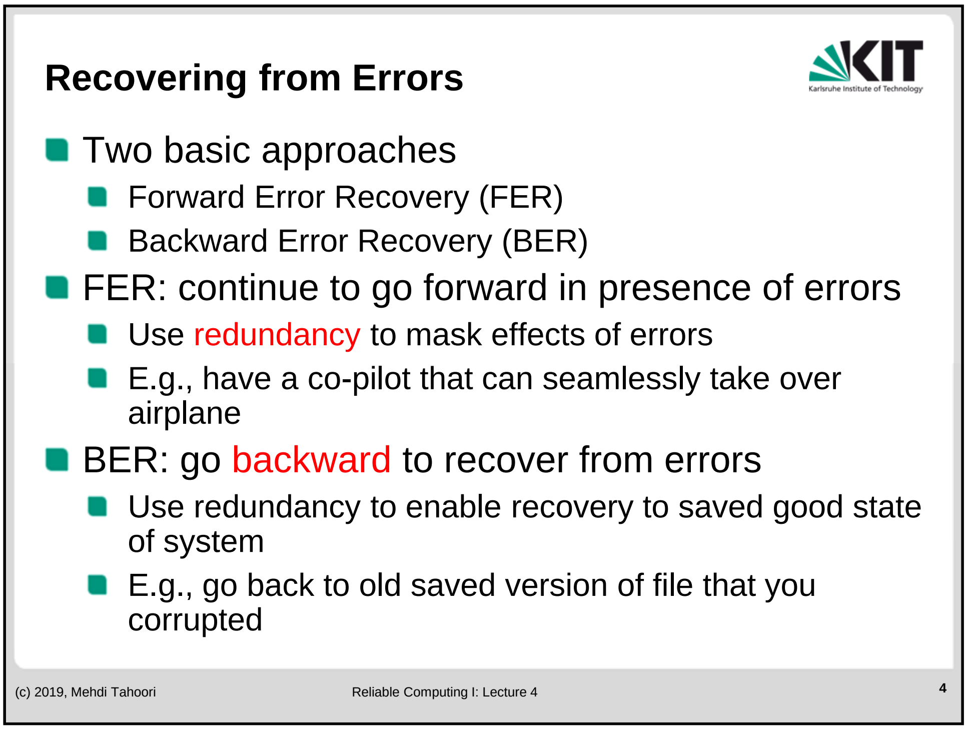
- Forward and backward error recovery
- Hardware redundancy schemes
  - Passive
  - Active
  - Hybrid

Reliable Computing I: Lecture 4 2



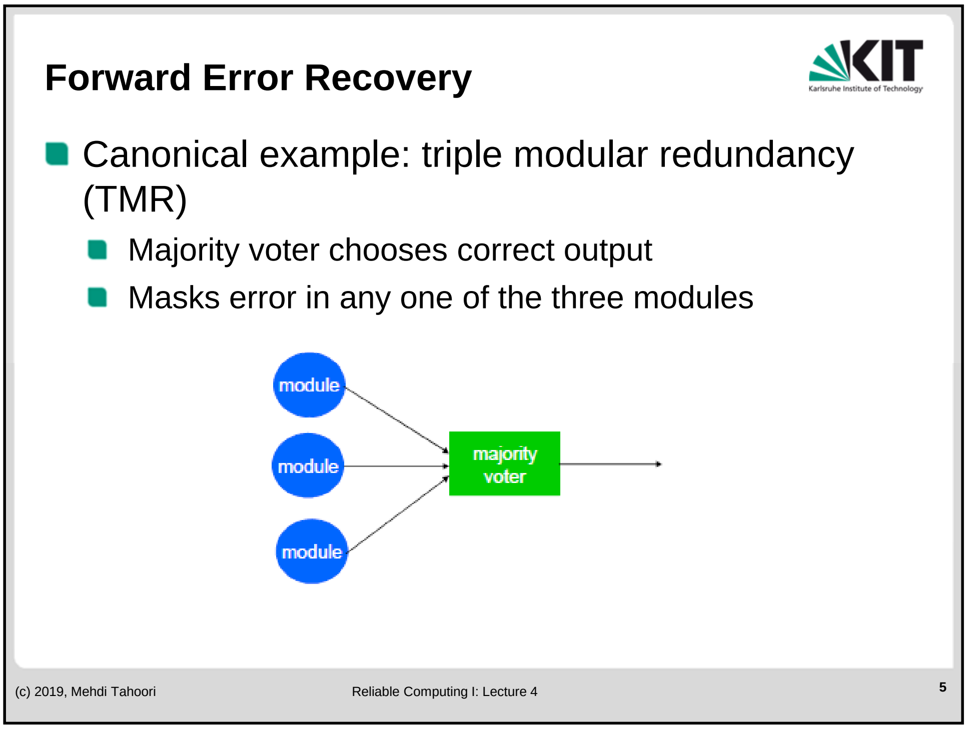
Redundancy

- Hardware redundancy
  - add extra hardware for detection or tolerating faults
- Information redundancy
  - extra information, i.e. codes
- Time redundancy
  - extra time for performing tasks for fault tolerance
- Software redundancy
  - add extra software for detection and possibly tolerating faults



Recovering from Errors

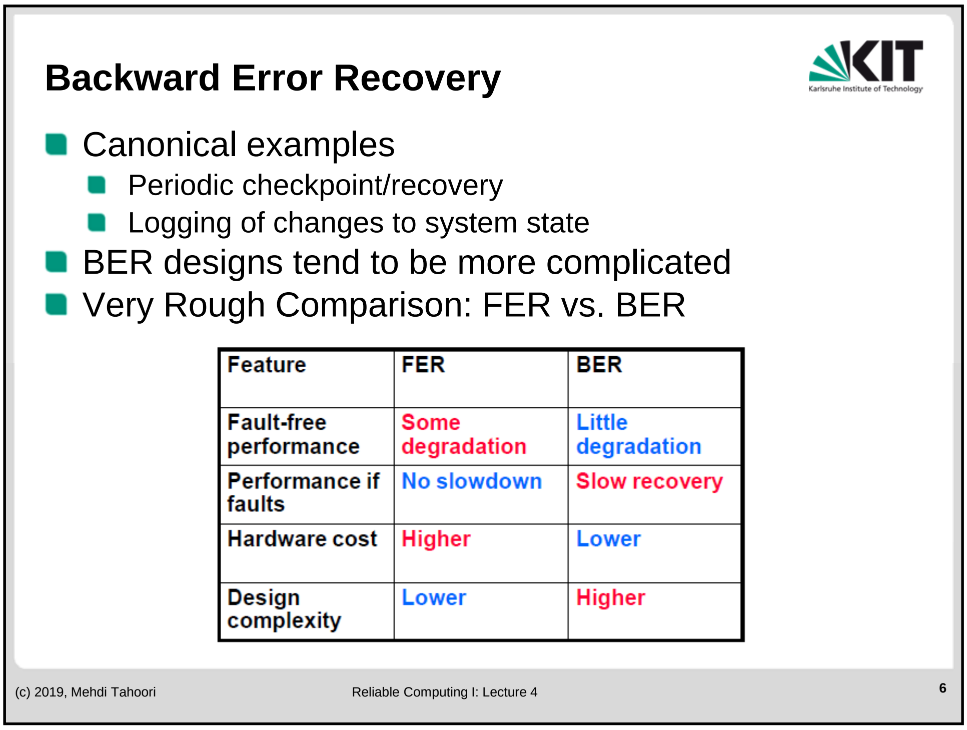
- Two basic approaches
  - Forward Error Recovery (FER)
  - Backward Error Recovery (BER)
- FER: continue to go forward in presence of errors
  - Use redundancy to mask effects of errors
  - E.g., have a co-pilot that can seamlessly take over airplane
- BER: go backward to recover from errors
  - Use redundancy to enable recovery to saved good state of system
  - E.g., go back to old saved version of file that you corrupted



Forward Error Recovery

- Canonical example: triple modular redundancy (TMR)
  - Majority voter chooses correct output
  - Masks error in any one of the three modules

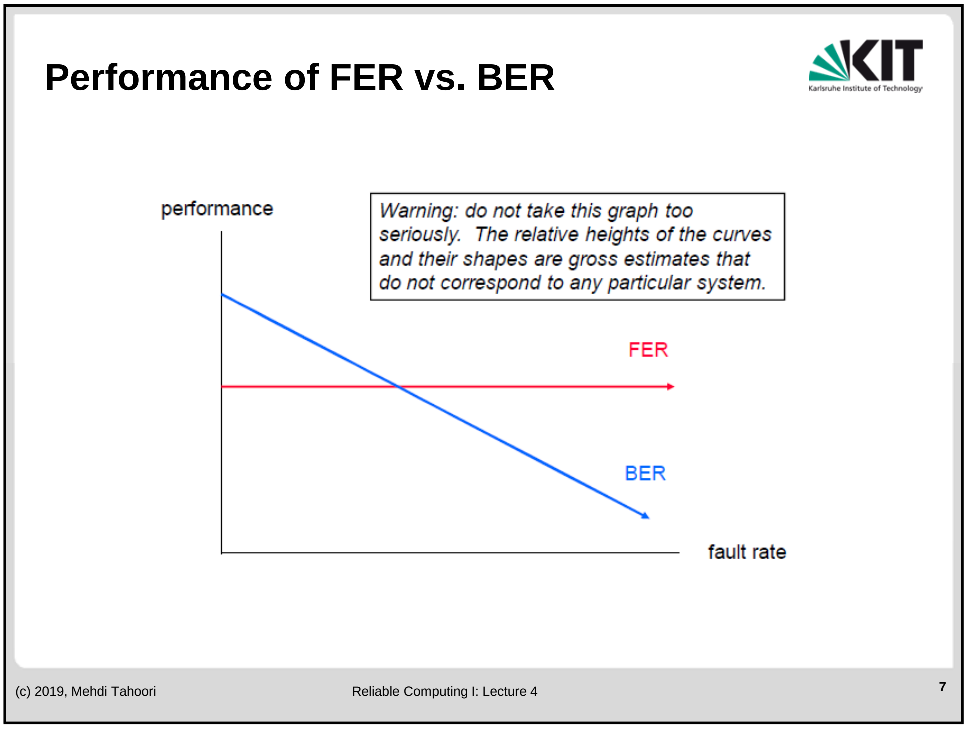
module
module
module
majority voter



Backward Error Recovery

- Canonical examples
  - Periodic checkpoint/recovery
  - Logging of changes to system state
- BER designs tend to be more complicated
- Very Rough Comparison: FER vs. BER

Feature	FER	BER
Fault-free performance	Some degradation	Little degradation
Performance if faults	No slowdown	Slow recovery
Hardware cost	Higher	Lower
Design complexity	Lower	Higher



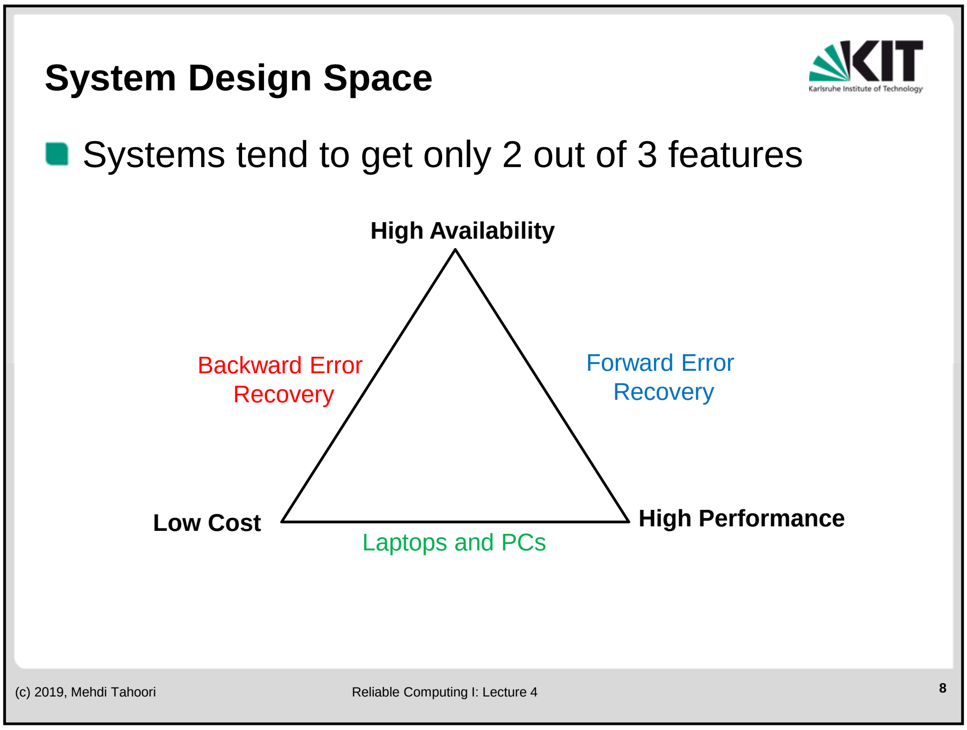
Performance of FER vs. BER

Warning: do not take this graph too seriously. The relative heights of the curves and their shapes are gross estimates that do not correspond to any particular system.

(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 4

7



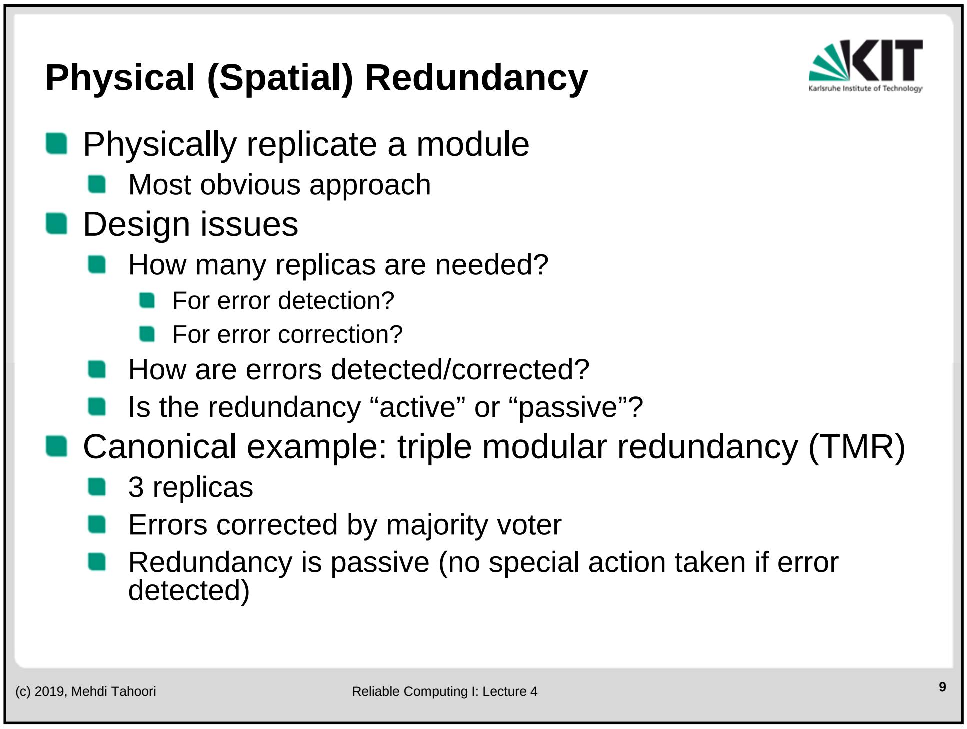
System Design Space

- Systems tend to get only 2 out of 3 features

(c) 2019, Mehdi Tahoori

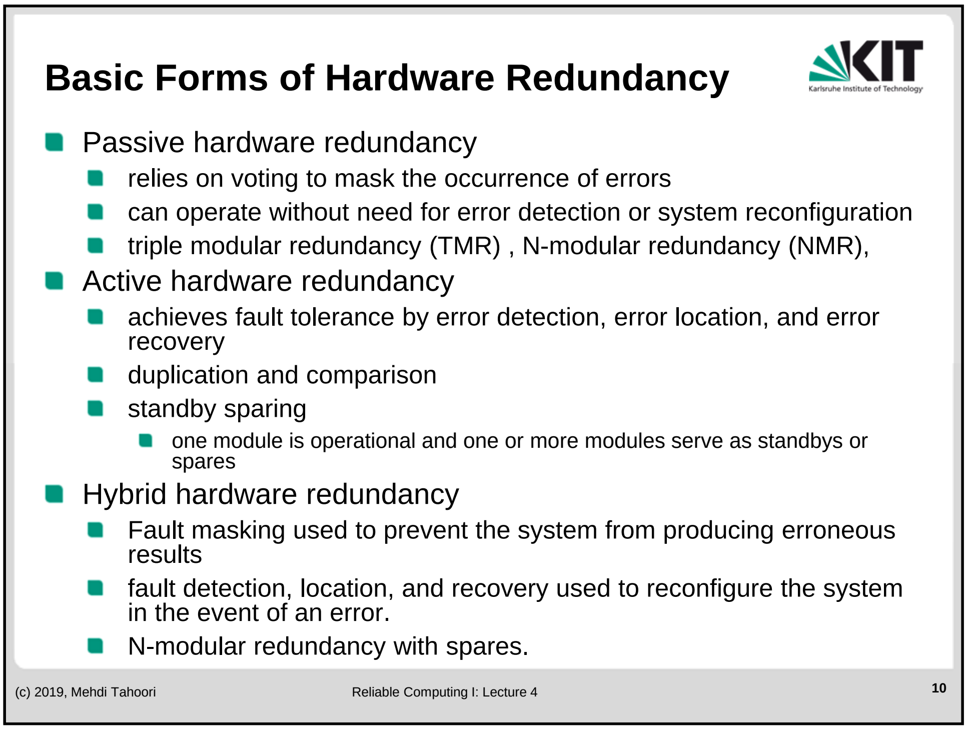
Reliable Computing I: Lecture 4

8



Physical (Spatial) Redundancy

- Physically replicate a module
  - Most obvious approach
- Design issues
  - How many replicas are needed?
    - For error detection?
    - For error correction?
  - How are errors detected/corrected?
  - Is the redundancy “active” or “passive”?
- Canonical example: triple modular redundancy (TMR)
  - 3 replicas
  - Errors corrected by majority voter
  - Redundancy is passive (no special action taken if error detected)



Basic Forms of Hardware Redundancy

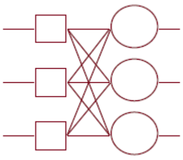
- Passive hardware redundancy
  - relies on voting to mask the occurrence of errors
  - can operate without need for error detection or system reconfiguration
  - triple modular redundancy (TMR) , N-modular redundancy (NMR),
- Active hardware redundancy
  - achieves fault tolerance by error detection, 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.

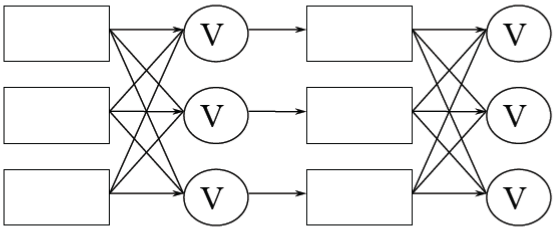
Physical Redundancy: TMR

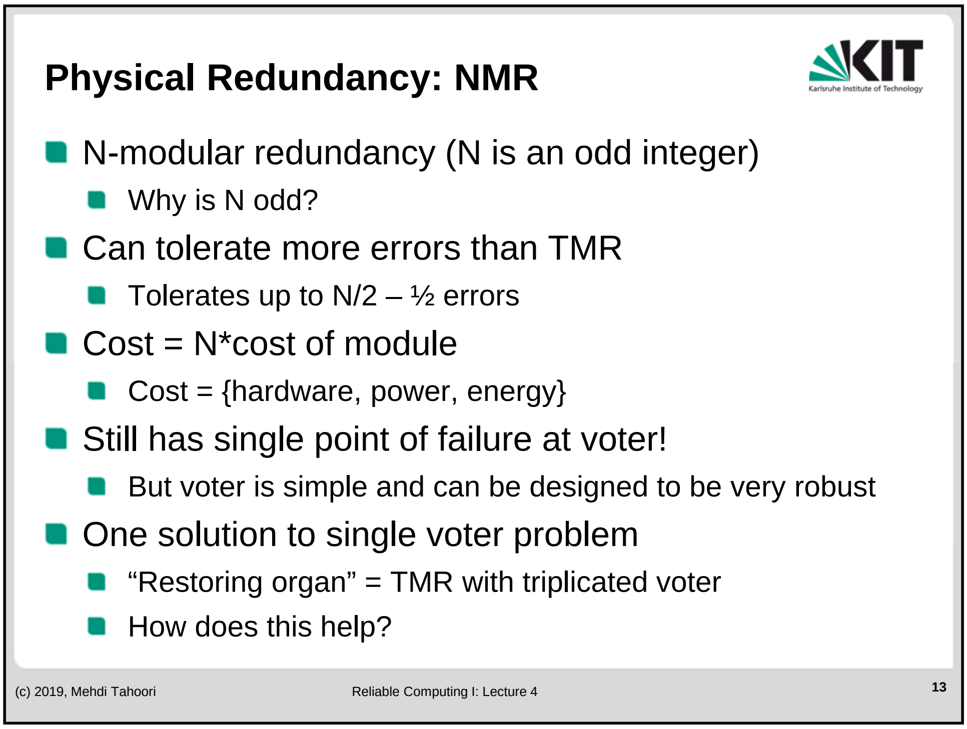
- Strengths
  - Tolerates an error in any single module
  - Tolerates soft and hard errors
  - Simple design
  - Small performance penalty, even when faults occur
- Weaknesses
  - Can’t tolerate multiple faults
    - Can’t tolerate any faults after a latent hard fault
  - Expensive hardware (3x cost)
  - Uses lots of power (approx 3x power of unprotected)
  - Also a 3x energy cost
  - Single point of failure at voter
  - Can’t tolerate errors due to design faults … why not?

TMR with 3 Voters

- Remove single point of failure
- Use TMR with 3 voters
  - Restoring organ
- Cascade such systems
  - Multistage TMR with replicate voters

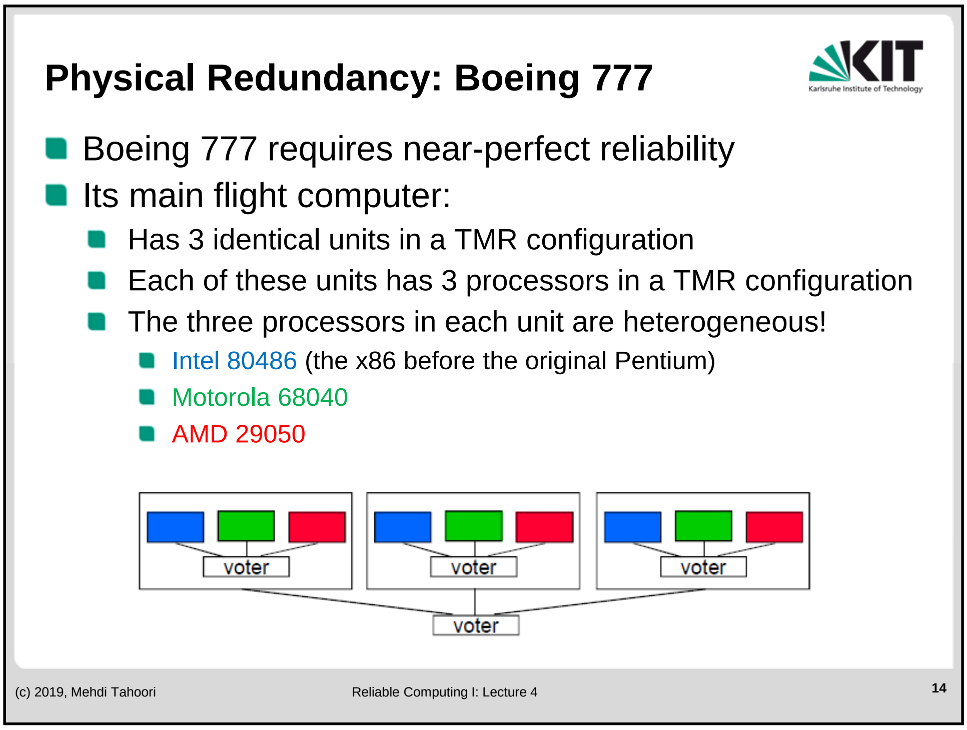






Physical Redundancy: NMR

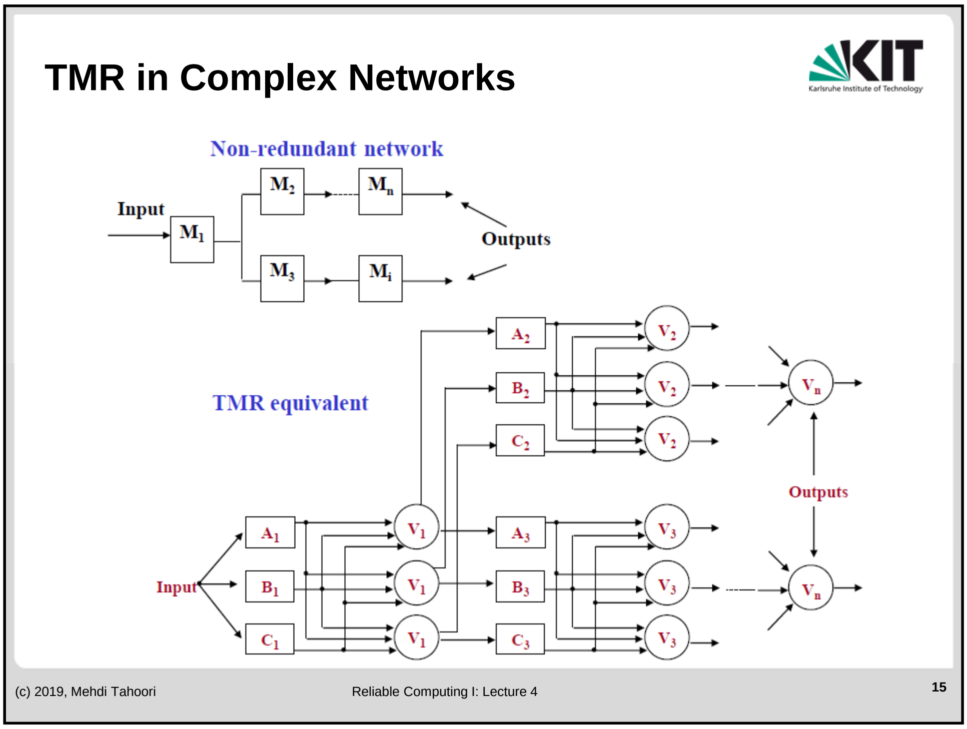
- N-modular redundancy (N is an odd integer)
  - Why is N odd?
- Can tolerate more errors than TMR
  - Tolerates up to N/2 – ½ errors
- Cost = N*cost of module
  - Cost = {hardware, power, energy}
- Still has single point of failure at voter!
  - But voter is simple and can be designed to be very robust
- One solution to single voter problem
  - “Restoring organ” = TMR with triplicated voter
  - How does this help?



Physical Redundancy: Boeing 777

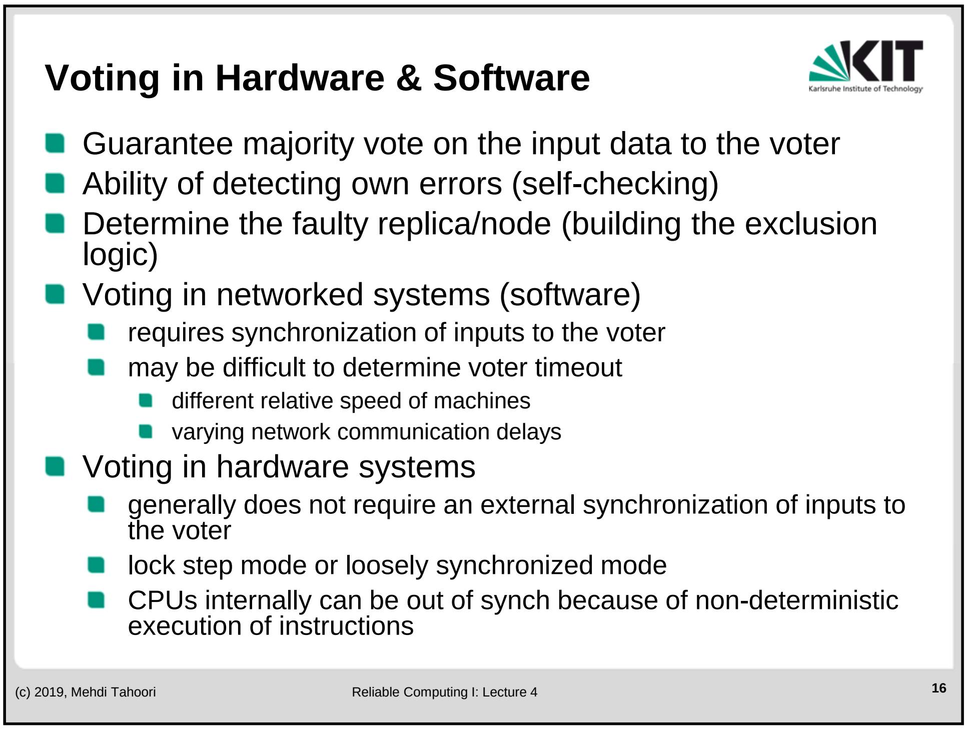
- Boeing 777 requires near-perfect reliability
- Its main flight computer:
  - Has 3 identical units in a TMR configuration
  - Each of these units has 3 processors in a TMR configuration
  - The three processors in each unit are heterogeneous!
    - Intel 80486 (the x86 before the original Pentium)
    - Motorola 68040
    - AMD 29050

voter
voter
voter
voter



TMR in Complex Networks

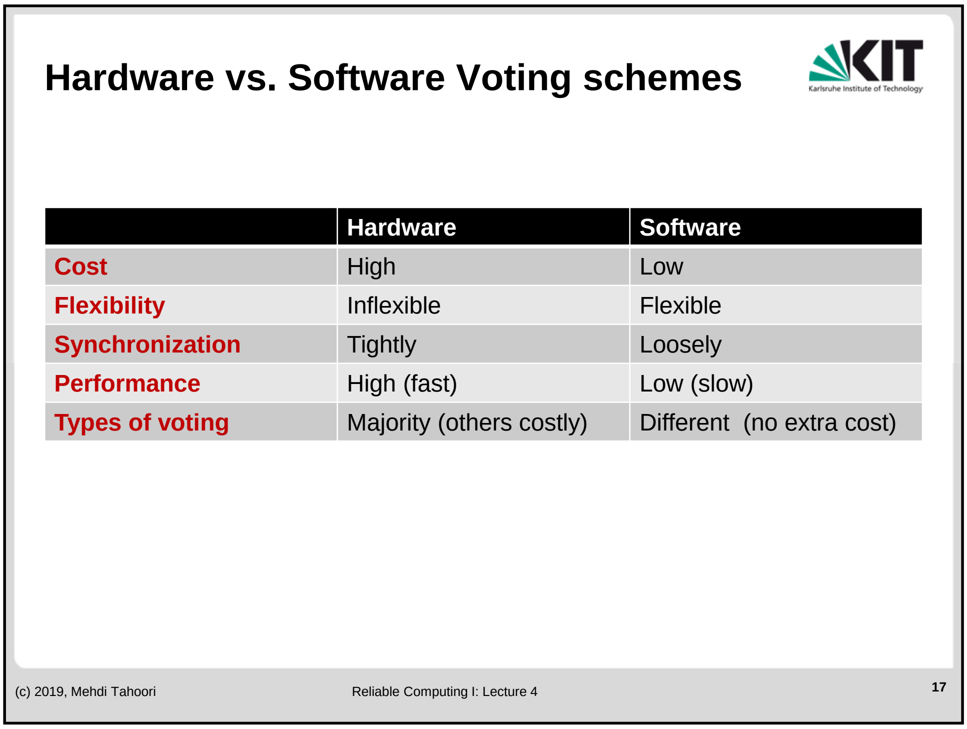
(c) 2019, Mehdi Tahoori Reliable Computing I: Lecture 4 15



Voting in Hardware & Software

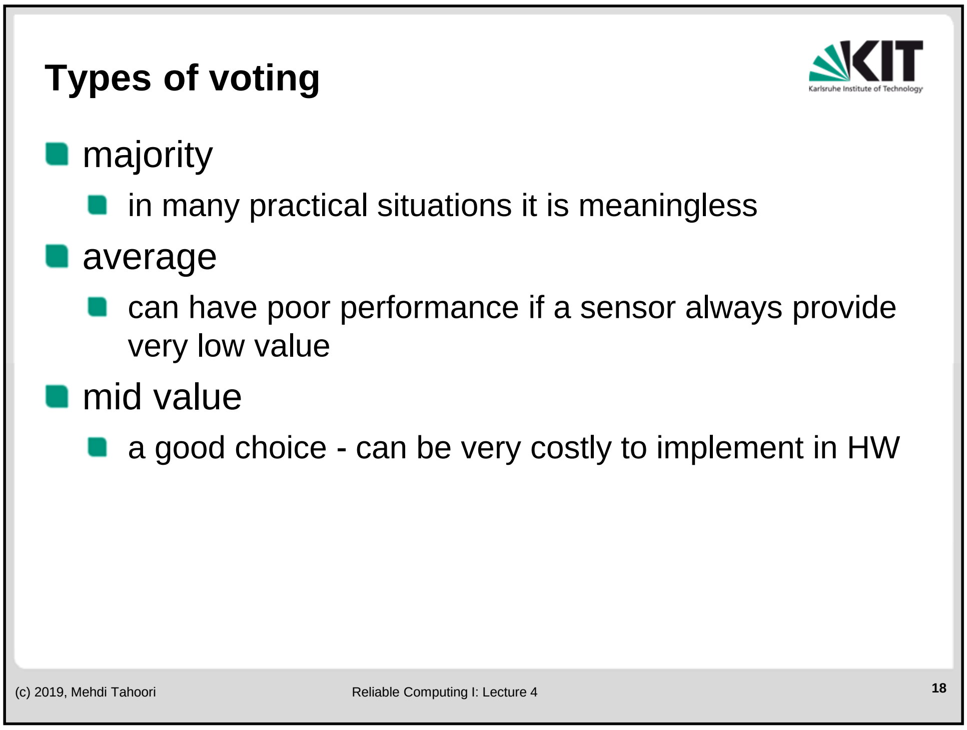
- Guarantee majority vote on the input data to the voter
- Ability of detecting own errors (self-checking)
- Determine the faulty replica/node (building the exclusion logic)
- Voting in networked systems (software)
  - requires synchronization of inputs to the voter
  - may be difficult to determine voter timeout
    - different relative speed of machines
    - varying network communication delays
- Voting in hardware systems
  - generally does not require an external synchronization of inputs to the voter
  - lock step mode or loosely synchronized mode
  - CPUs internally can be out of synch because of non-deterministic execution of instructions

(c) 2019, Mehdi Tahoori Reliable Computing I: Lecture 4 16



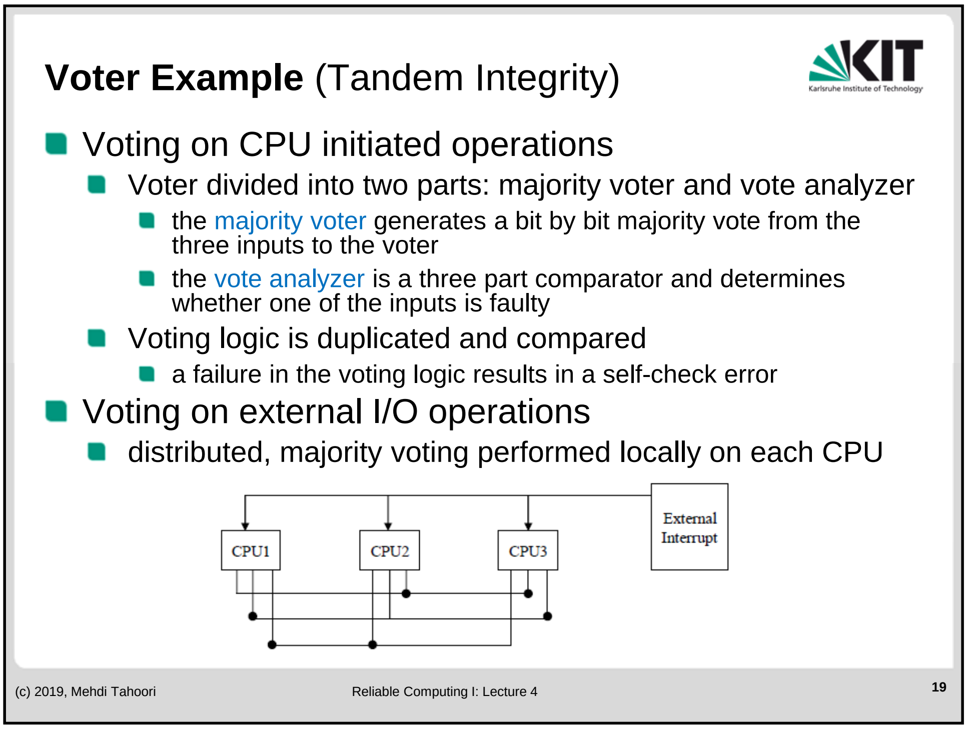
Hardware vs. Software Voting schemes

	Hardware	Software
Cost	High	Low
Flexibility	Inflexible	Flexible
Synchronization	Tightly	Loosely
Performance	High (fast)	Low (slow)
Types of voting	Majority (others costly)	Different (no extra cost)



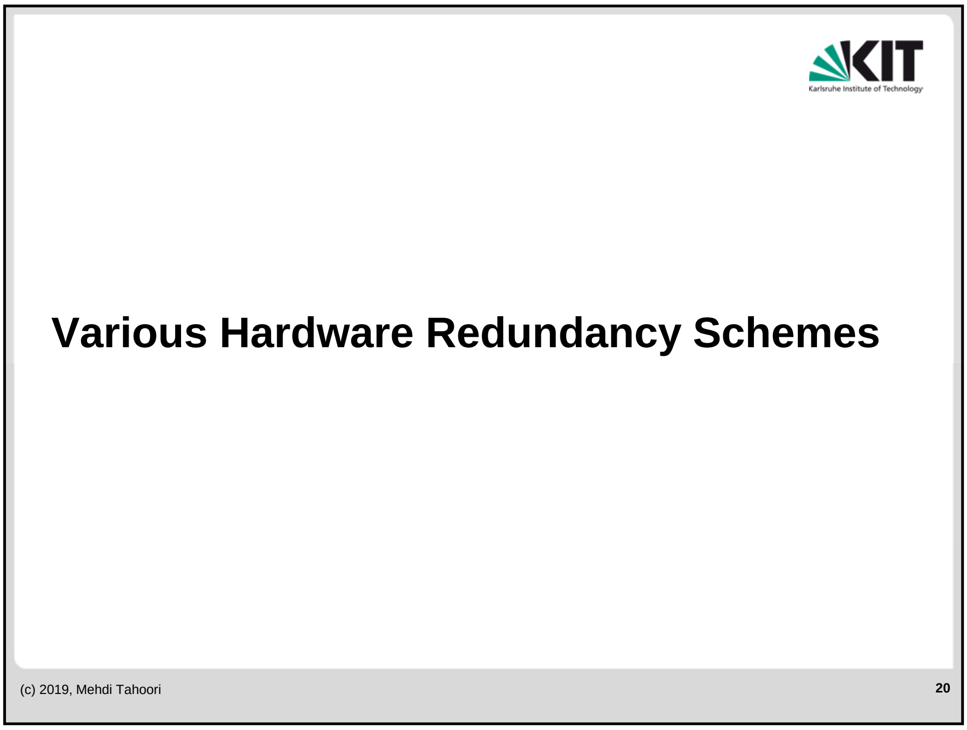
Types of voting

- 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

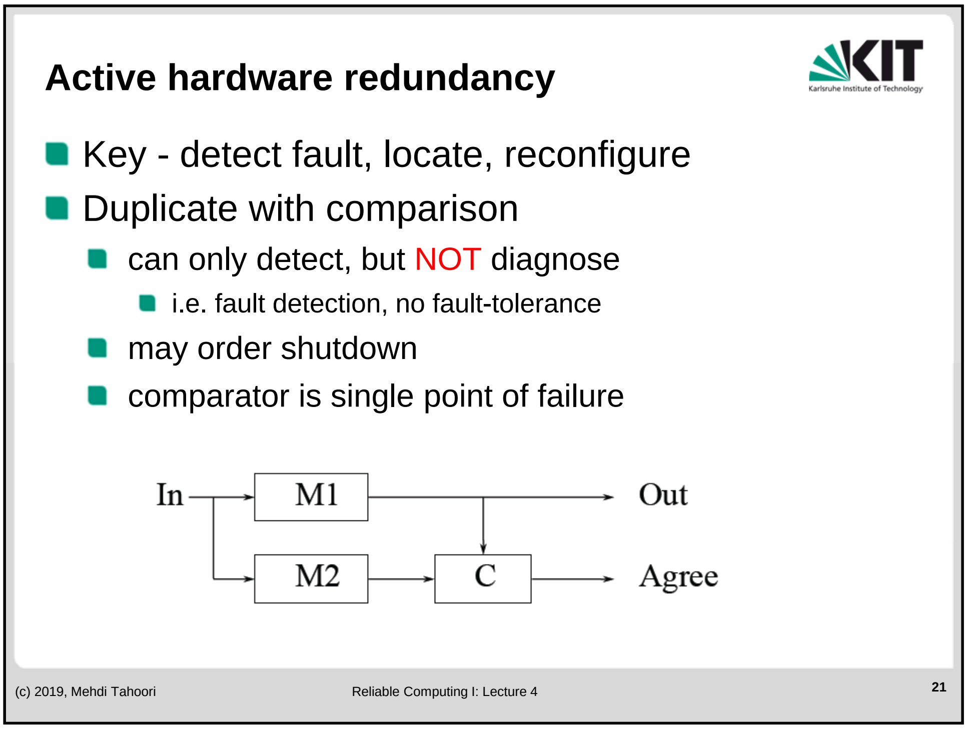


Voter Example (Tandem Integrity)

- Voting on CPU initiated operations
  - Voter divided into two parts: majority voter and vote analyzer
    - the majority voter generates a bit by bit majority vote from the three inputs to the voter
    - the vote analyzer is a three part comparator and determines whether one of the inputs is faulty
  - Voting logic is duplicated and compared
    - a failure in the voting logic results in a self-check error
- Voting on external I/O operations
  - distributed, majority voting performed locally on each CPU



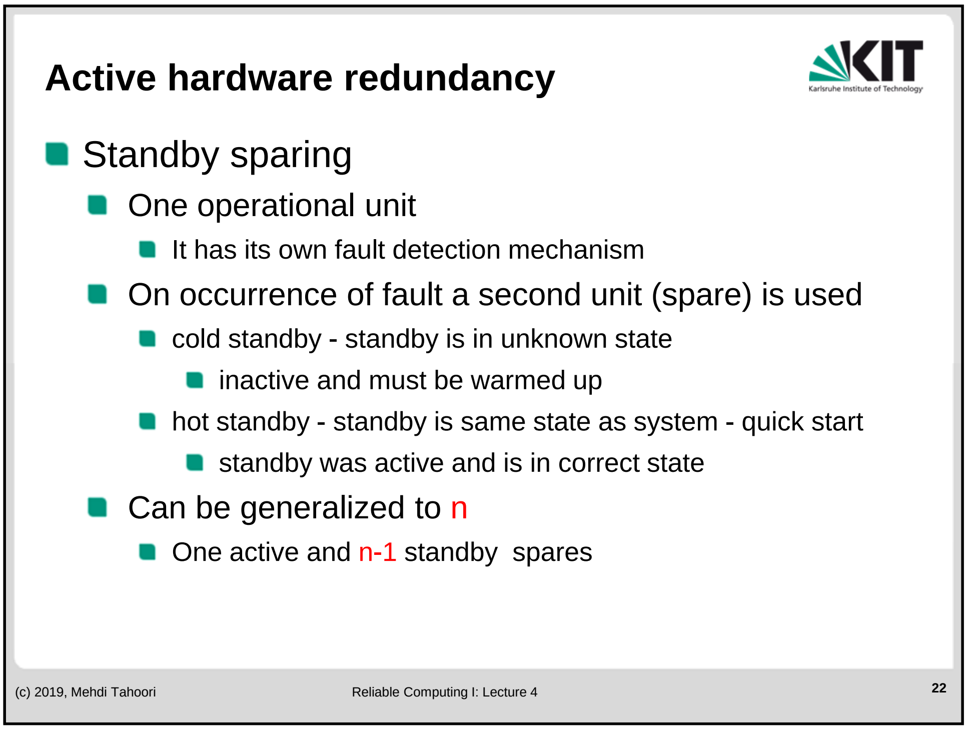
Various Hardware Redundancy Schemes



Active hardware redundancy

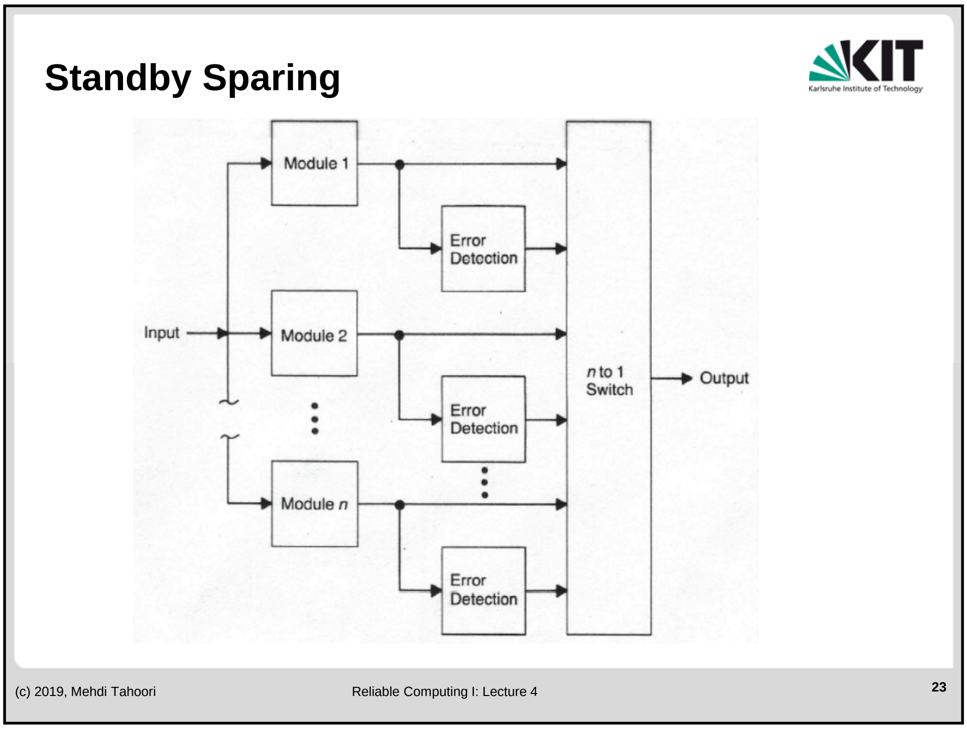
- Key - detect fault, locate, reconfigure
- Duplicate with comparison
  - can only detect, but NOT diagnose
    - i.e. fault detection, no fault-tolerance
  - may order shutdown
  - comparator is single point of failure

In
M1
M2
C
Out
Agree



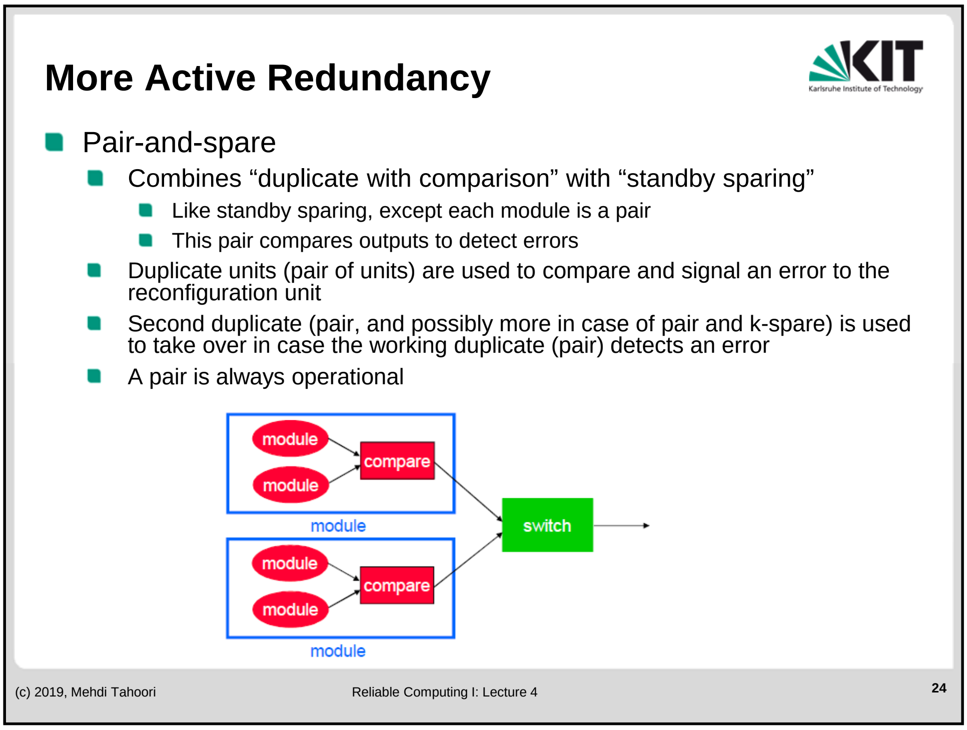
Active hardware redundancy

- Standby sparing
  - One operational unit
    - It has its own fault detection mechanism
  - On occurrence of fault a second unit (spare) is used
    - cold standby - standby is in unknown state
      - inactive and must be warmed up
    - hot standby - standby is same state as system - quick start
      - standby was active and is in correct state
  - Can be generalized to n
    - One active and n-1 standby spares



Standby Sparing

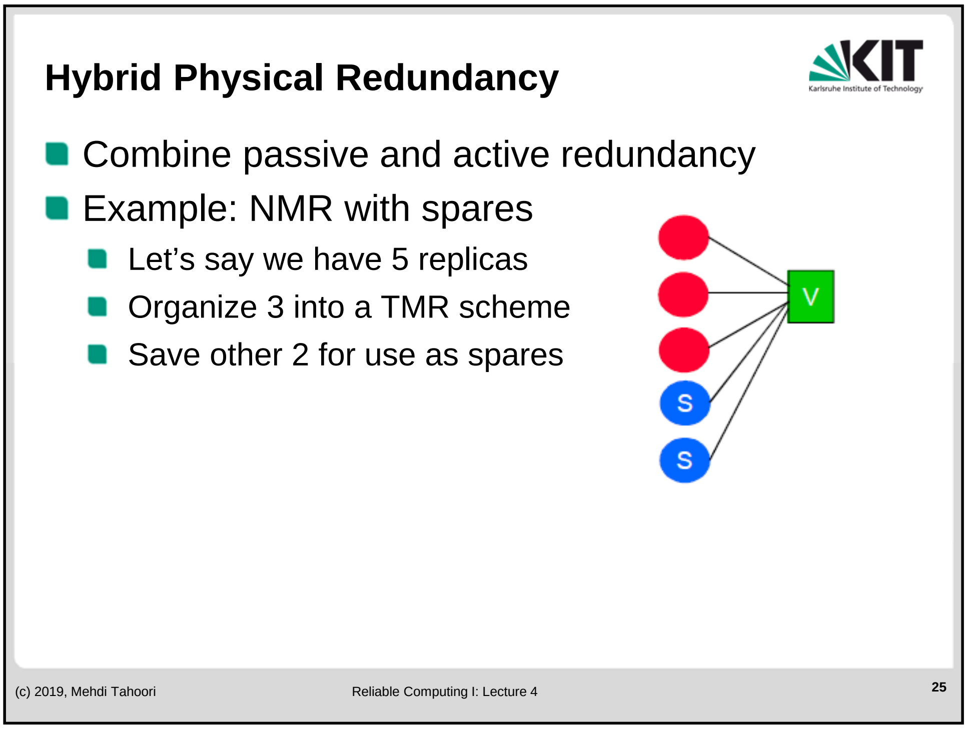
(c) 2019, Mehdi Tahoori Reliable Computing I: Lecture 4 23



More Active Redundancy

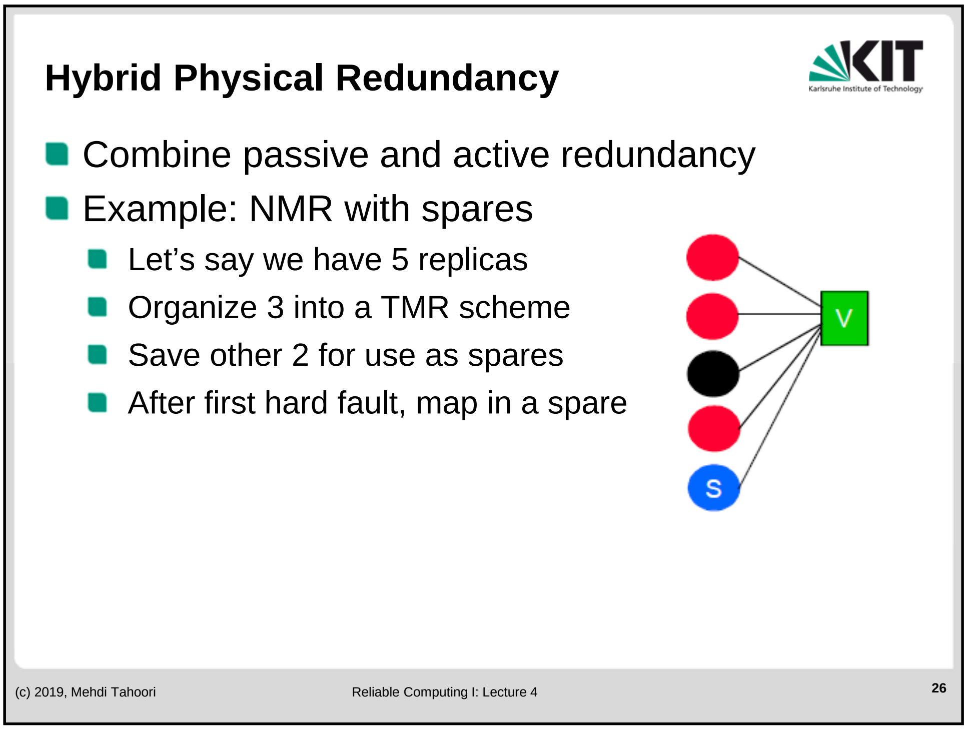
- Pair-and-spare
  - Combines “duplicate with comparison” with “standby sparing”
    - Like standby sparing, except each module is a pair
    - This pair compares outputs to detect errors
  - Duplicate units (pair of units) are used to compare and signal an error to the reconfiguration unit
  - Second duplicate (pair, and possibly more in case of pair and k-spare) is used to take over in case the working duplicate (pair) detects an error
  - A pair is always operational

(c) 2019, Mehdi Tahoori Reliable Computing I: Lecture 4 24



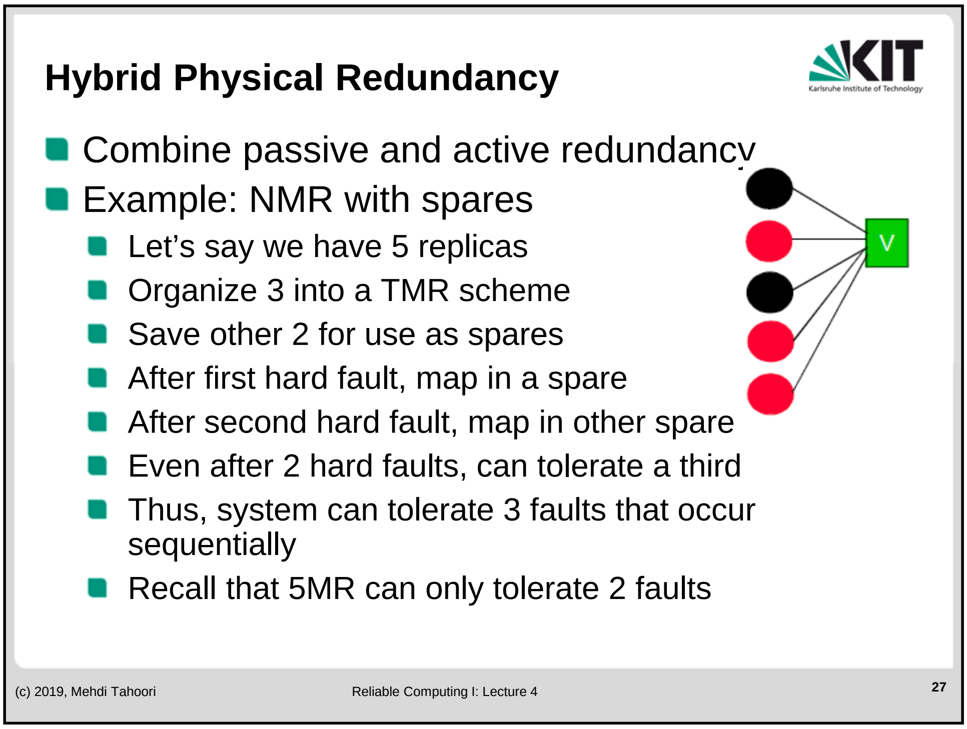
Hybrid Physical Redundancy

- Combine passive and active redundancy
- Example: NMR with spares
  - Let’s say we have 5 replicas
  - Organize 3 into a TMR scheme
  - Save other 2 for use as spares



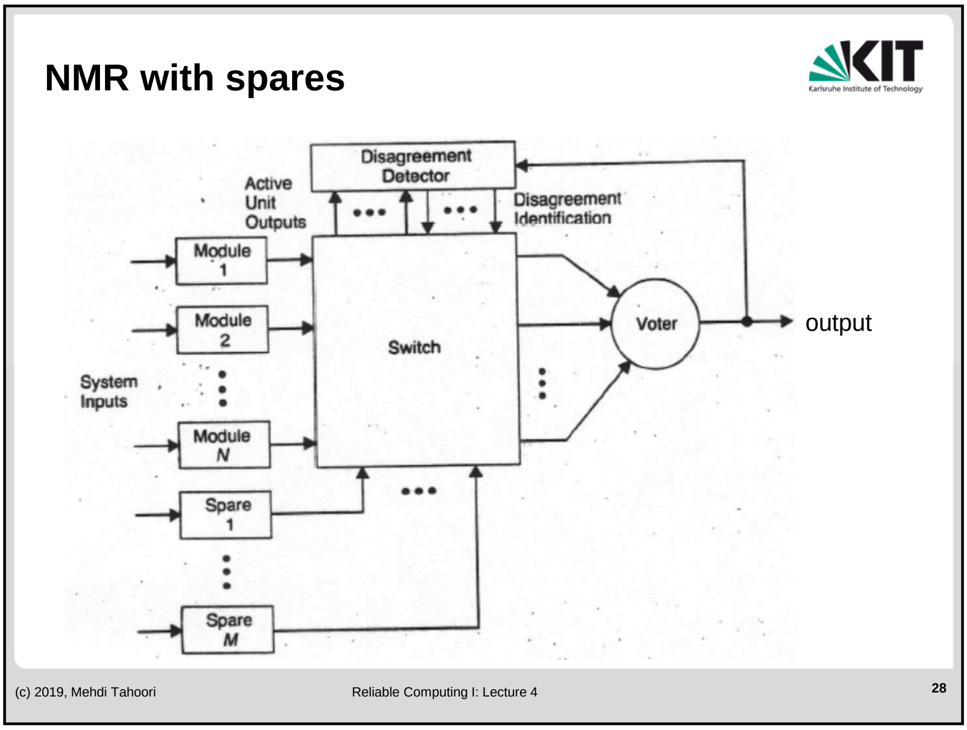
Hybrid Physical Redundancy

- Combine passive and active redundancy
- Example: NMR with spares
  - Let’s say we have 5 replicas
  - Organize 3 into a TMR scheme
  - Save other 2 for use as spares
  - After first hard fault, map in a spare

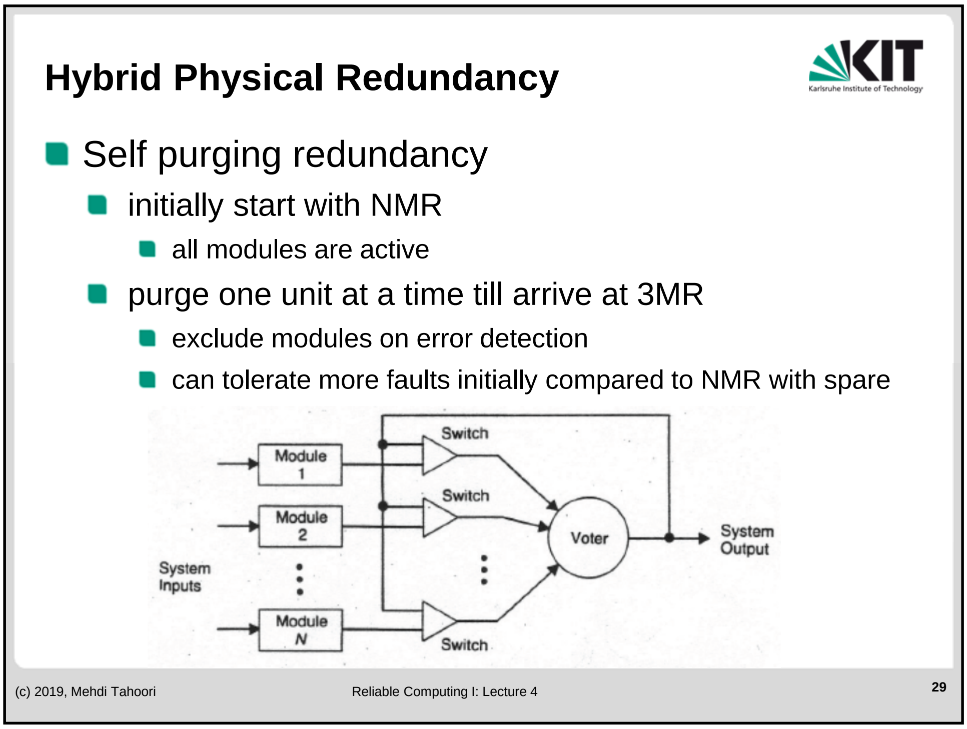


Hybrid Physical Redundancy

- Combine passive and active redundancy
- Example: NMR with spares
  - Let’s say we have 5 replicas
  - Organize 3 into a TMR scheme
  - Save other 2 for use as spares
  - After first hard fault, map in a spare
  - After second hard fault, map in other spare
  - Even after 2 hard faults, can tolerate a third
  - Thus, system can tolerate 3 faults that occur sequentially
  - Recall that 5MR can only tolerate 2 faults

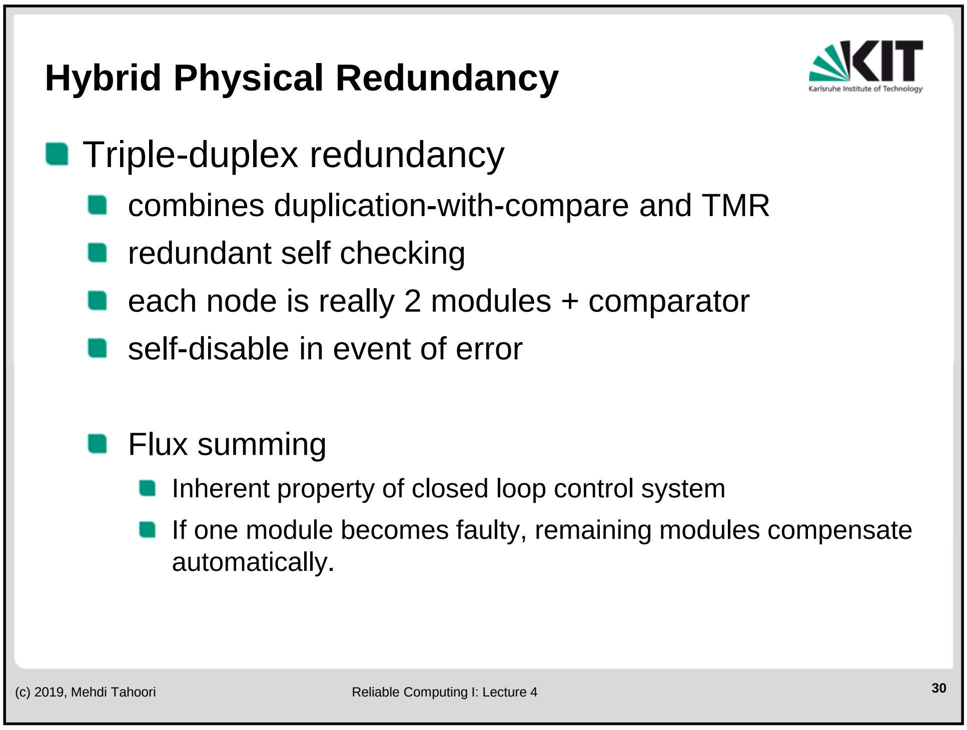


NMR with spares



Hybrid Physical Redundancy

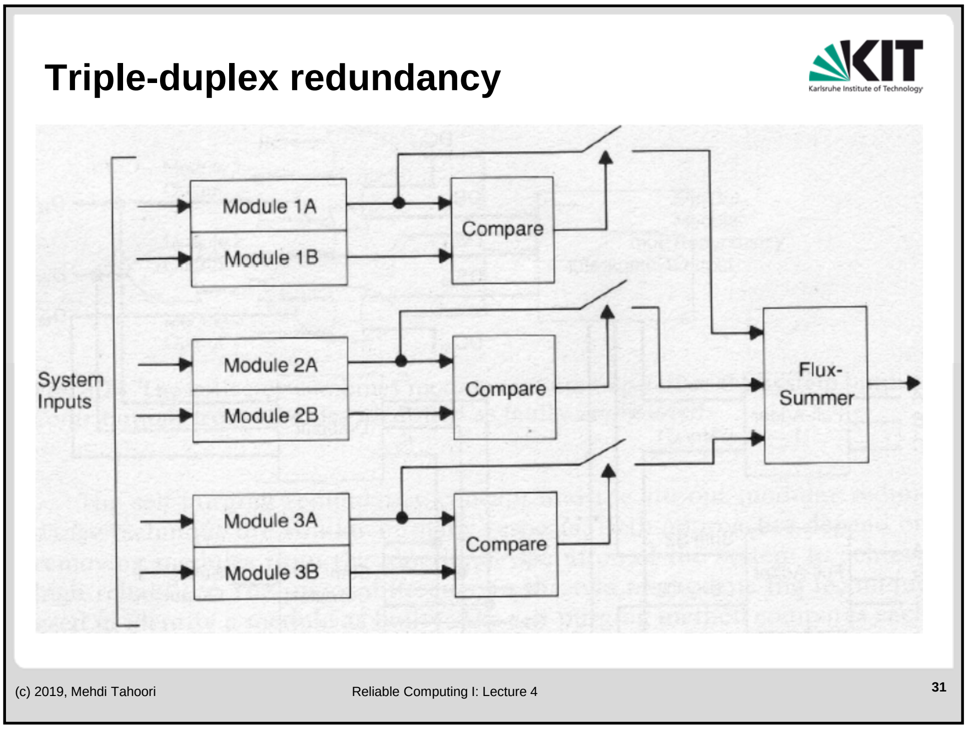
- Self purging redundancy
  - initially start with NMR
    - all modules are active
  - purge one unit at a time till arrive at 3MR
    - exclude modules on error detection
    - can tolerate more faults initially compared to NMR with spare



Hybrid Physical Redundancy

- Triple-duplex redundancy
  - combines duplication-with-compare and TMR
  - redundant self checking
  - each node is really 2 modules + comparator
  - self-disable in event of error

  - Flux summing
    - Inherent property of closed loop control system
    - If one module becomes faulty, remaining modules compensate automatically.



Triple-duplex redundancy

(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 4

31