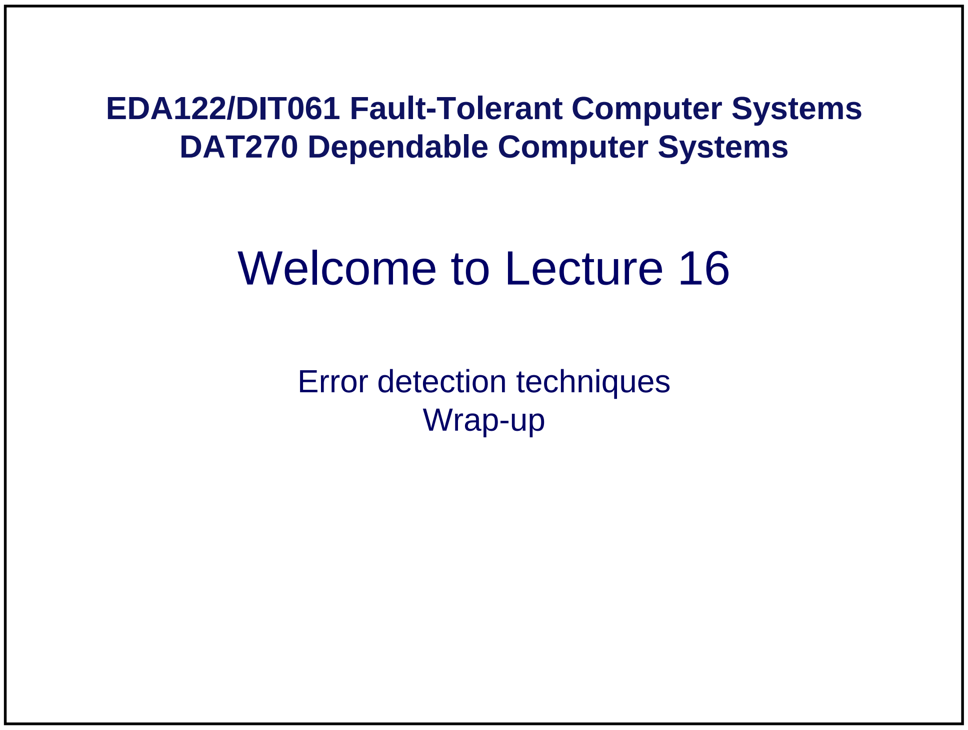
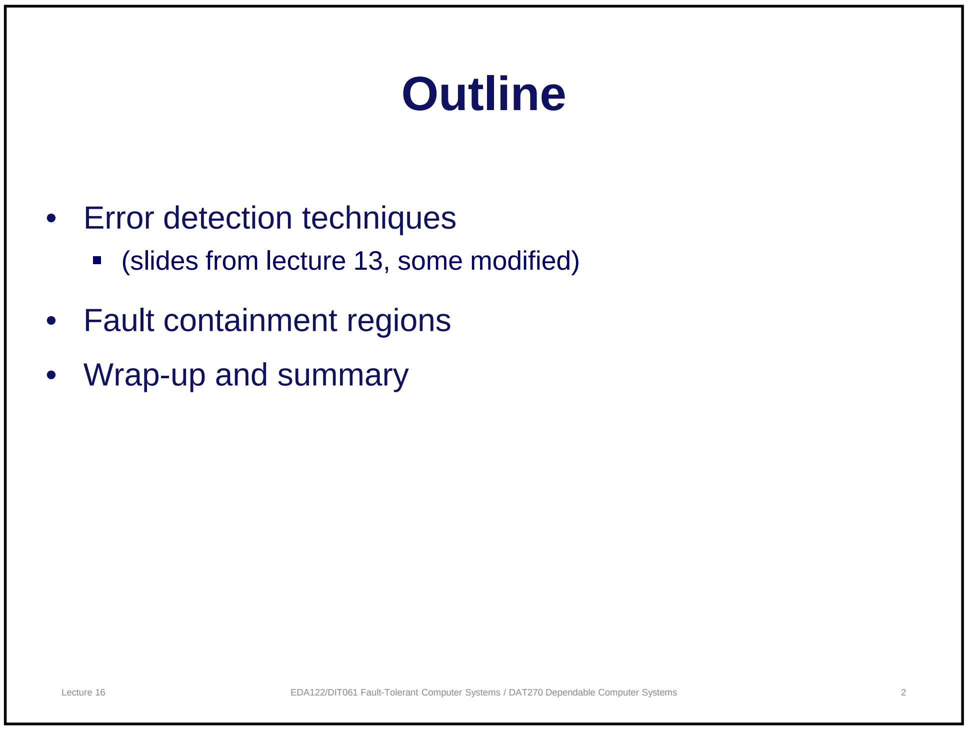


EDA122/DIT061 Fault-Tolerant Computer Systems
DAT270 Dependable Computer Systems

Welcome to Lecture 16

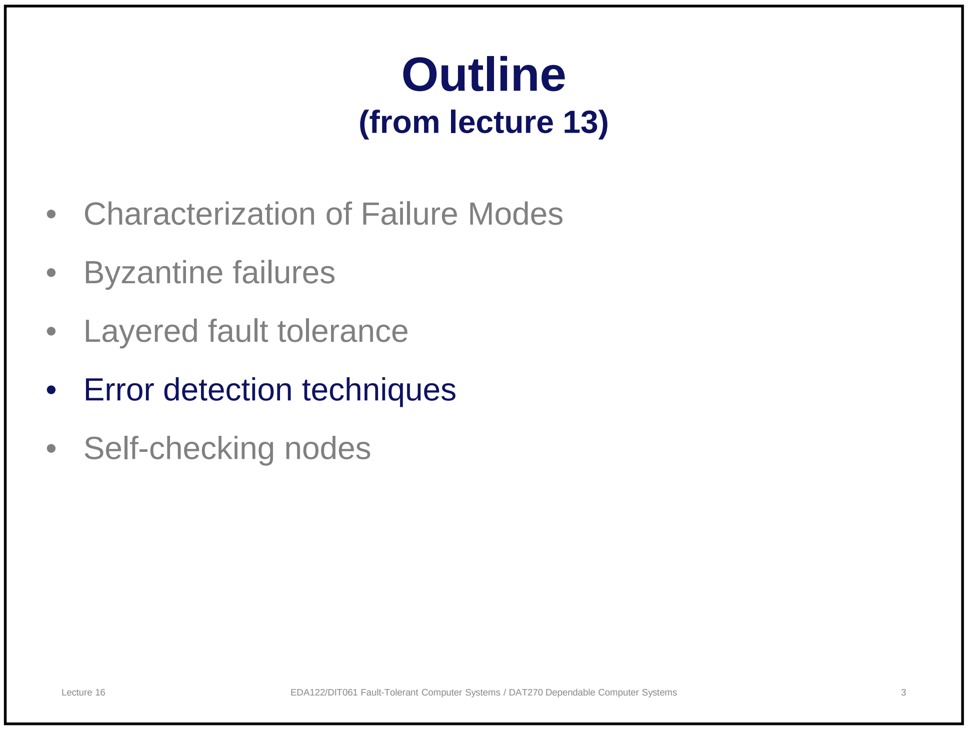
Error detection techniques
Wrap-up



Outline

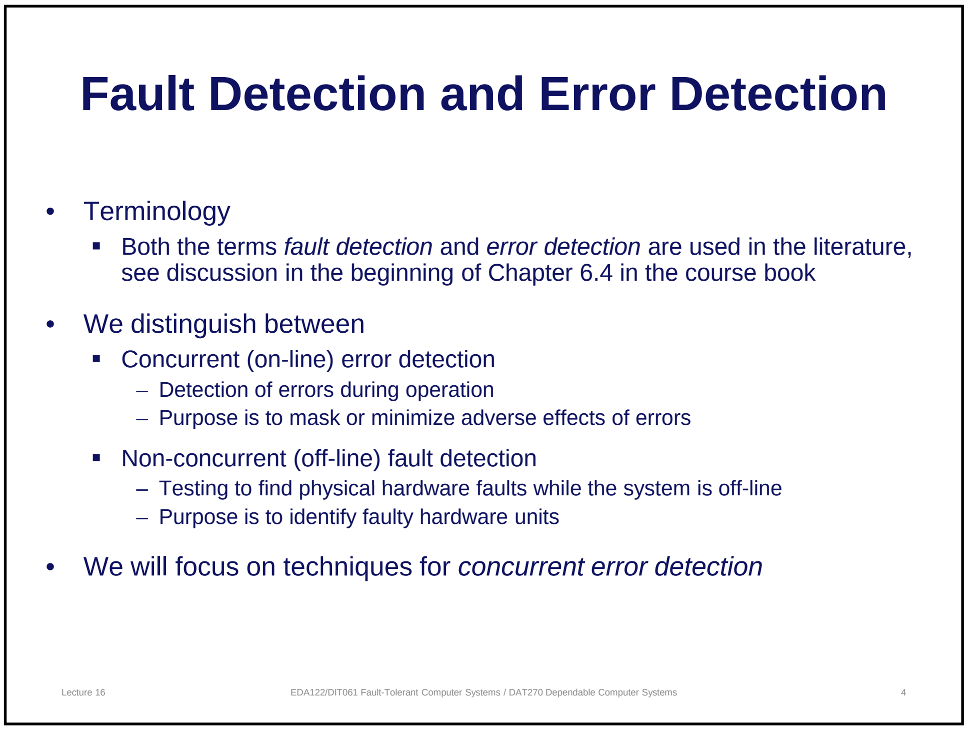
- Error detection techniques
  - (slides from lecture 13, some modified)
- Fault containment regions
- Wrap-up and summary

Lecture 16 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 2



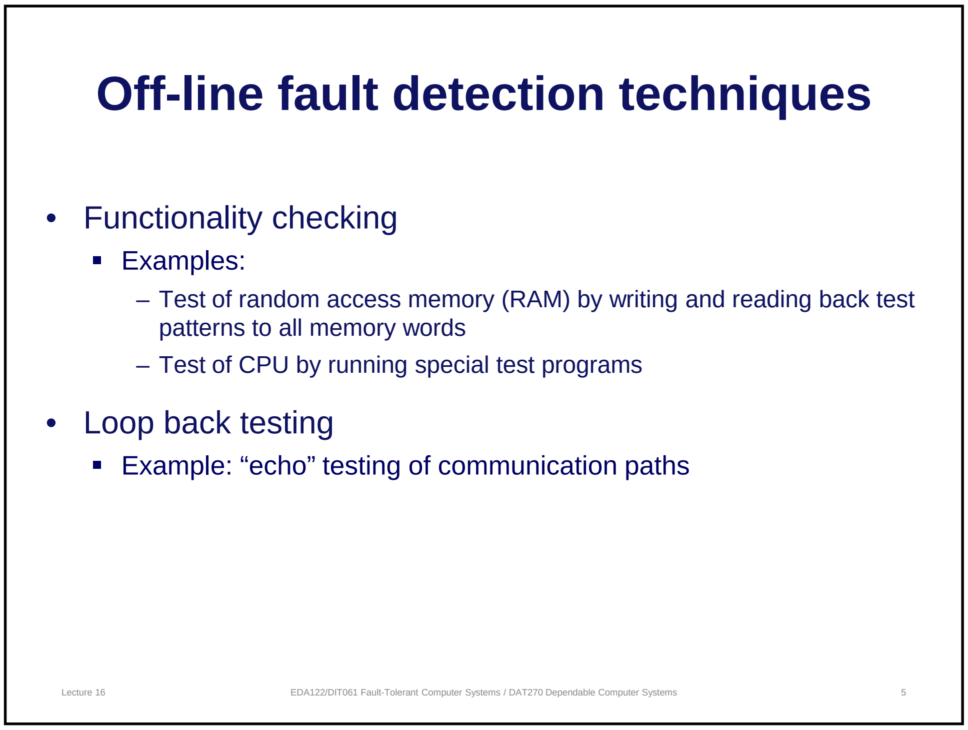
Outline
(from lecture 13)

- Characterization of Failure Modes
- Byzantine failures
- Layered fault tolerance
- Error detection techniques
- Self-checking nodes



Fault Detection and Error Detection

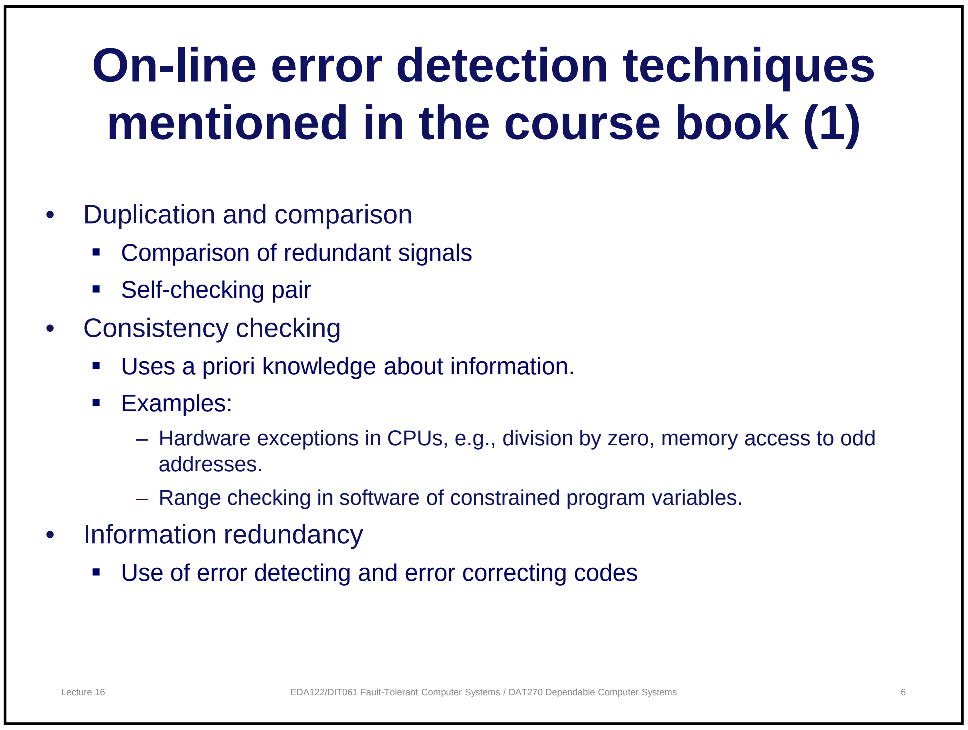
- Terminology
  - Both the terms *fault detection* and *error detection* are used in the literature, see discussion in the beginning of Chapter 6.4 in the course book
- We distinguish between
  - Concurrent (on-line) error detection
    - Detection of errors during operation
    - Purpose is to mask or minimize adverse effects of errors
  - Non-concurrent (off-line) fault detection
    - Testing to find physical hardware faults while the system is off-line
    - Purpose is to identify faulty hardware units
- We will focus on techniques for *concurrent error detection*



Off-line fault detection techniques

- Functionality checking
  - Examples:
    - Test of random access memory (RAM) by writing and reading back test patterns to all memory words
    - Test of CPU by running special test programs
- Loop back testing
  - Example: “echo” testing of communication paths

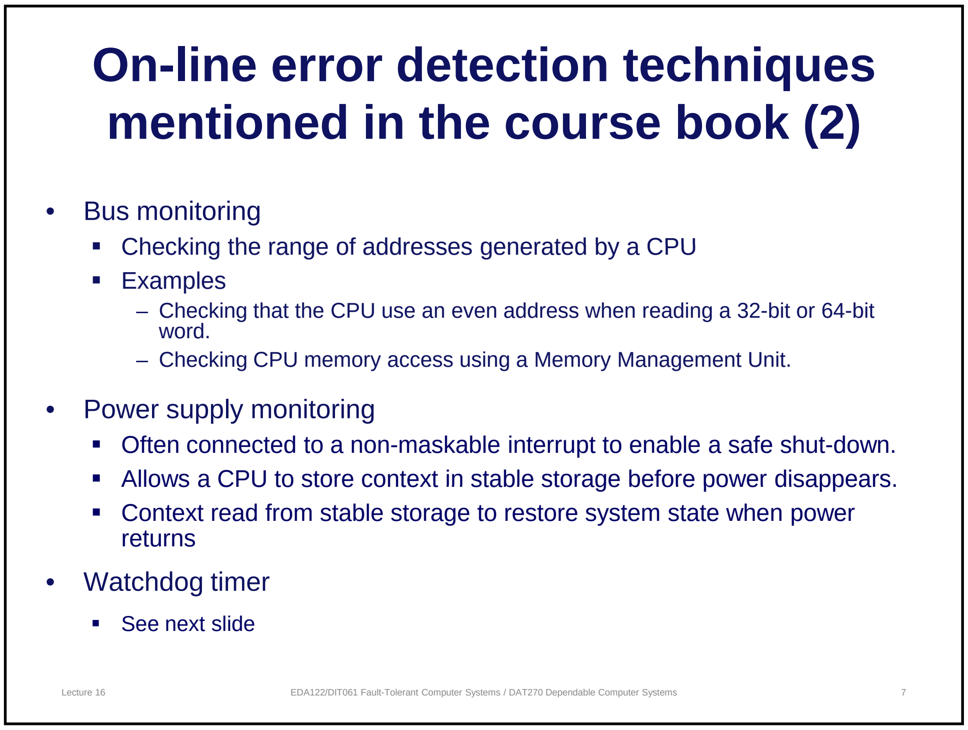
Lecture 16 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 5



On-line error detection techniques mentioned in the course book (1)

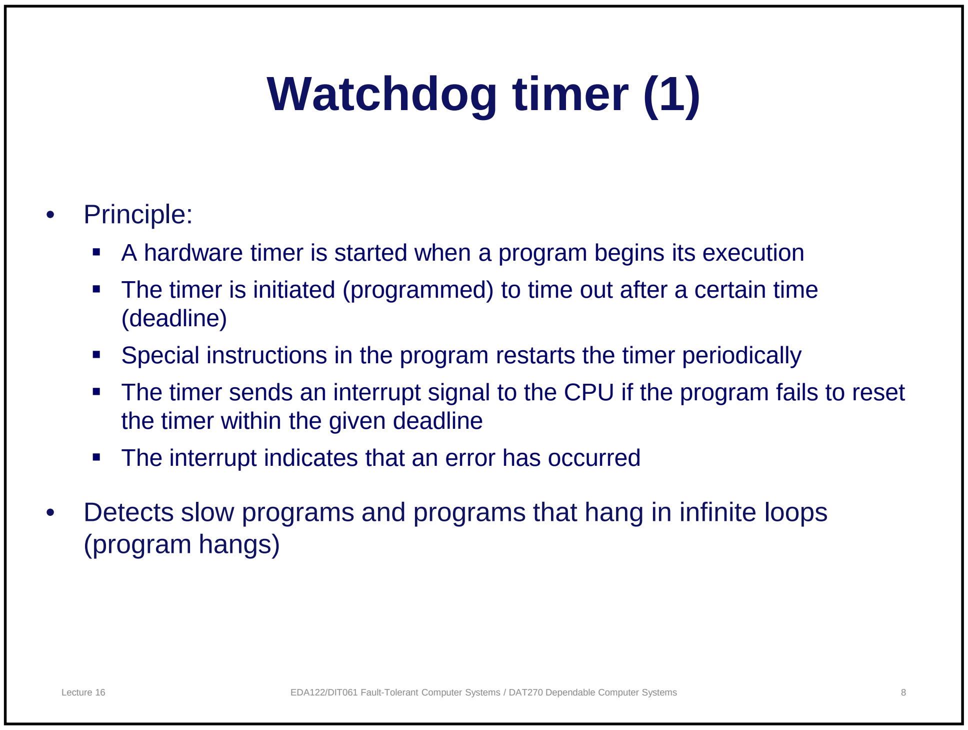
- Duplication and comparison
  - Comparison of redundant signals
  - Self-checking pair
- Consistency checking
  - Uses a priori knowledge about information.
  - Examples:
    - Hardware exceptions in CPUs, e.g., division by zero, memory access to odd addresses.
    - Range checking in software of constrained program variables.
- Information redundancy
  - Use of error detecting and error correcting codes

Lecture 16 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 6



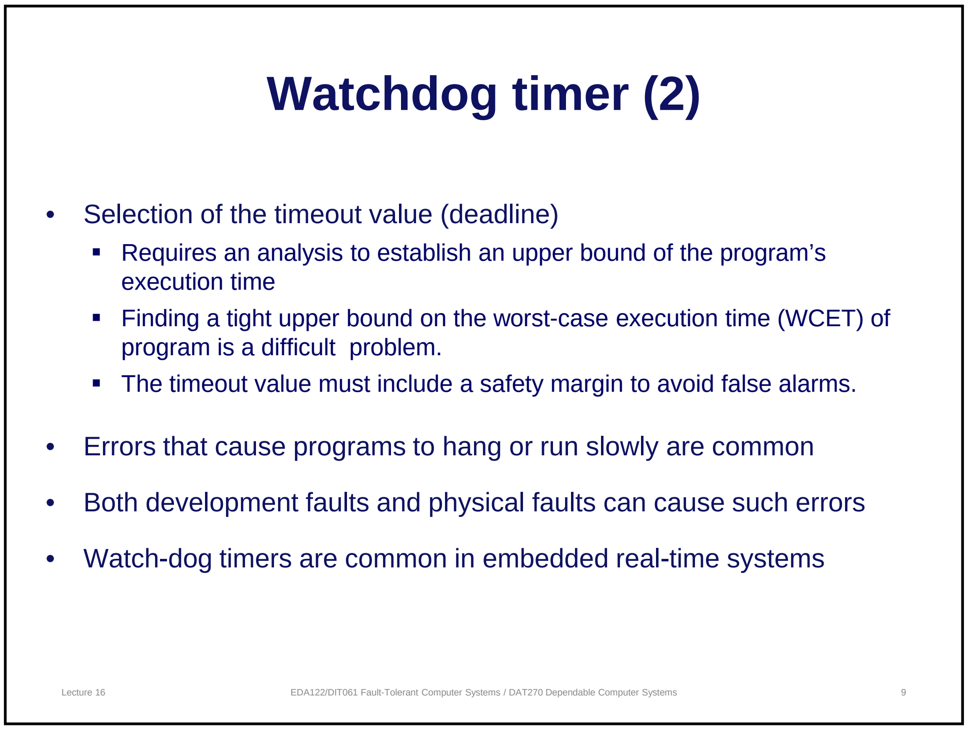
On-line error detection techniques mentioned in the course book (2)

- Bus monitoring
  - Checking the range of addresses generated by a CPU
  - Examples
    - Checking that the CPU use an even address when reading a 32-bit or 64-bit word.
    - Checking CPU memory access using a Memory Management Unit.
- Power supply monitoring
  - Often connected to a non-maskable interrupt to enable a safe shut-down.
  - Allows a CPU to store context in stable storage before power disappears.
  - Context read from stable storage to restore system state when power returns
- Watchdog timer
  - See next slide



Watchdog timer (1)

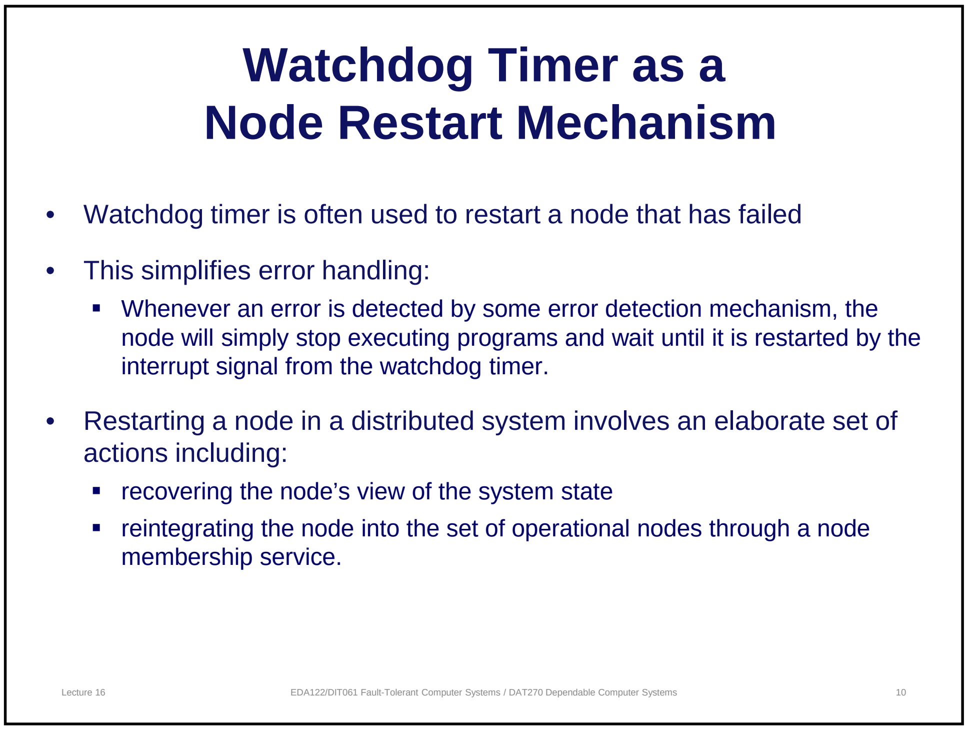
- Principle:
  - A hardware timer is started when a program begins its execution
  - The timer is initiated (programmed) to time out after a certain time (deadline)
  - Special instructions in the program restarts the timer periodically
  - The timer sends an interrupt signal to the CPU if the program fails to reset the timer within the given deadline
  - The interrupt indicates that an error has occurred
- Detects slow programs and programs that hang in infinite loops (program hangs)



Watchdog timer (2)

- Selection of the timeout value (deadline)
  - Requires an analysis to establish an upper bound of the program's execution time
  - Finding a tight upper bound on the worst-case execution time (WCET) of program is a difficult problem.
  - The timeout value must include a safety margin to avoid false alarms.
- Errors that cause programs to hang or run slowly are common
- Both development faults and physical faults can cause such errors
- Watch-dog timers are common in embedded real-time systems

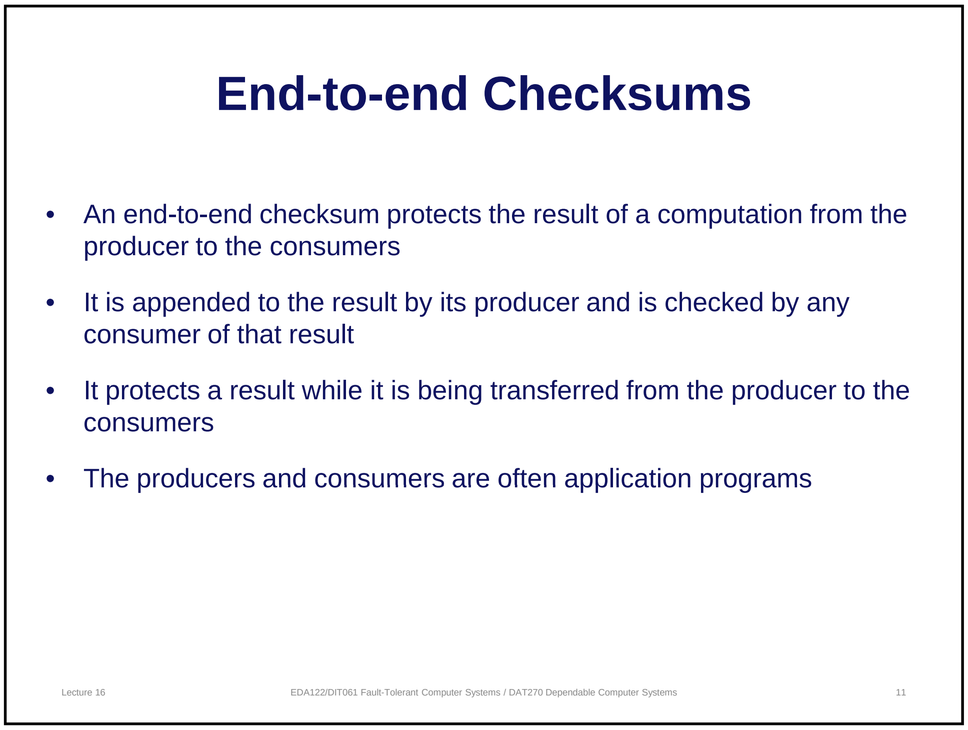
Lecture 16 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 9



Watchdog Timer as a Node Restart Mechanism

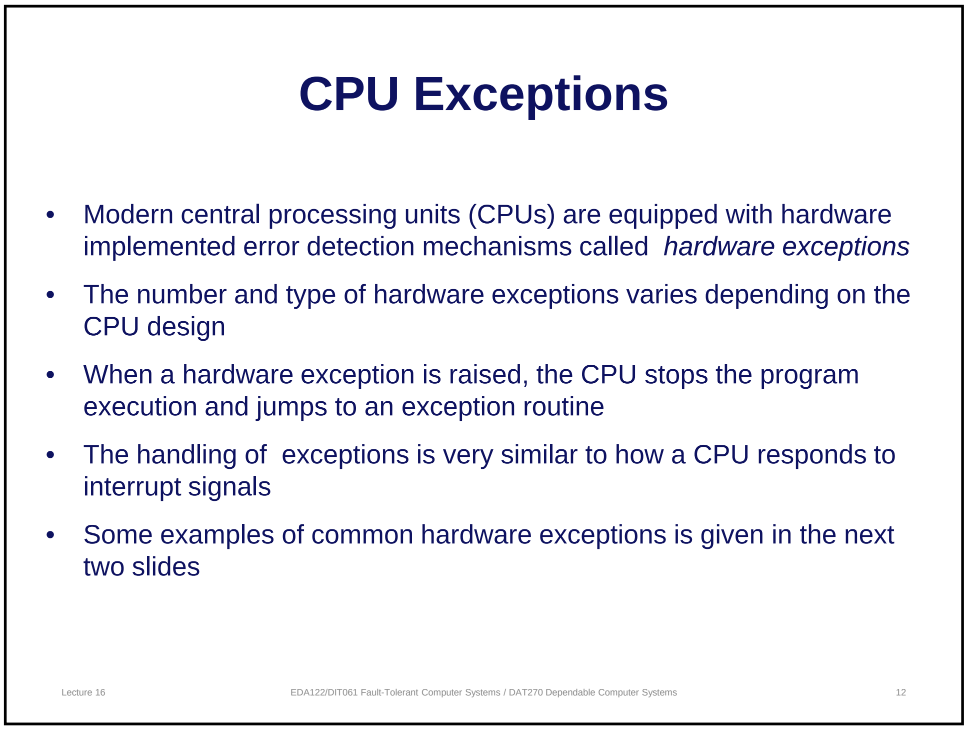
- Watchdog timer is often used to restart a node that has failed
- This simplifies error handling:
  - Whenever an error is detected by some error detection mechanism, the node will simply stop executing programs and wait until it is restarted by the interrupt signal from the watchdog timer.
- Restarting a node in a distributed system involves an elaborate set of actions including:
  - recovering the node's view of the system state
  - reintegrating the node into the set of operational nodes through a node membership service.

Lecture 16 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 10



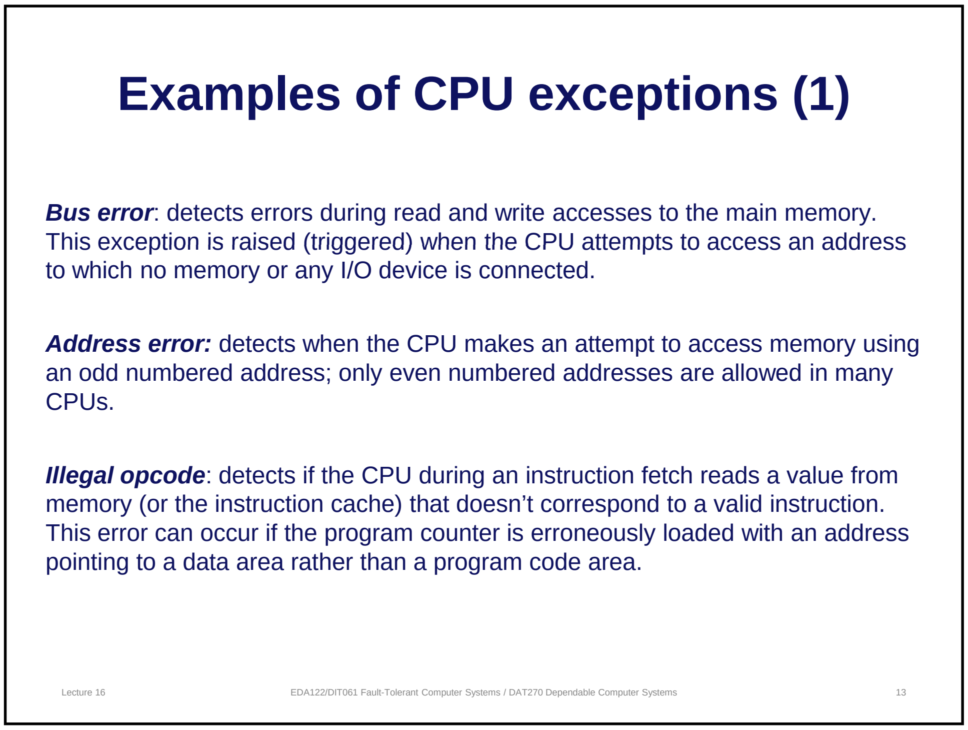
End-to-end Checksums

- An end-to-end checksum protects the result of a computation from the producer to the consumers
- It is appended to the result by its producer and is checked by any consumer of that result
- It protects a result while it is being transferred from the producer to the consumers
- The producers and consumers are often application programs



CPU Exceptions

- Modern central processing units (CPUs) are equipped with hardware implemented error detection mechanisms called *hardware exceptions*
- The number and type of hardware exceptions varies depending on the CPU design
- When a hardware exception is raised, the CPU stops the program execution and jumps to an exception routine
- The handling of exceptions is very similar to how a CPU responds to interrupt signals
- Some examples of common hardware exceptions is given in the next two slides

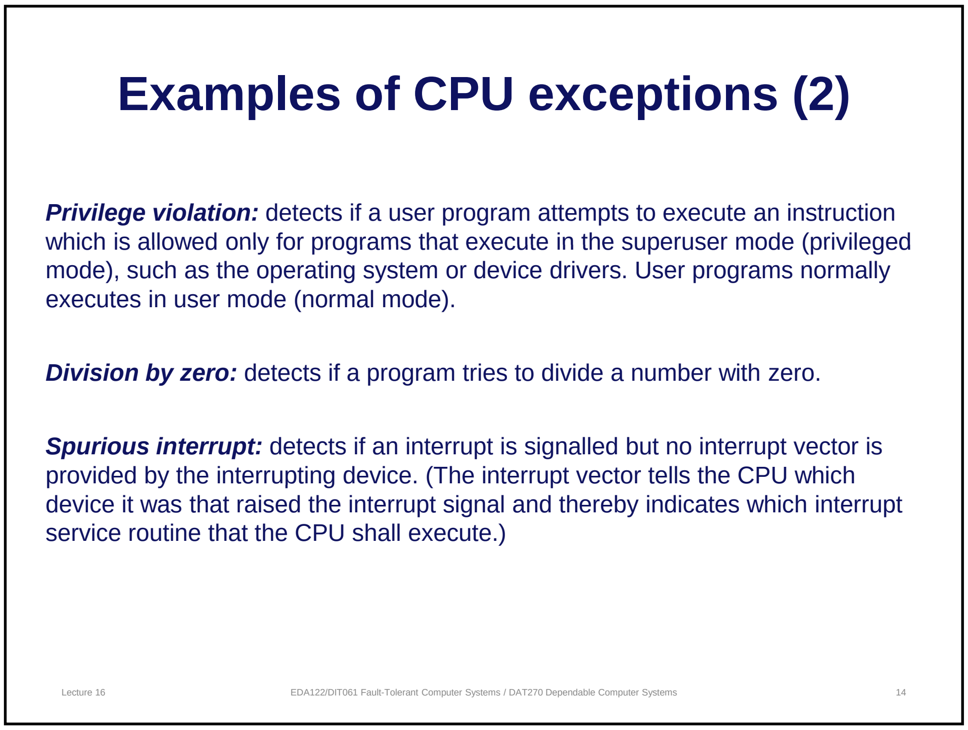


Examples of CPU exceptions (1)

Bus error: detects errors during read and write accesses to the main memory. This exception is raised (triggered) when the CPU attempts to access an address to which no memory or any I/O device is connected.

Address error: detects when the CPU makes an attempt to access memory using an odd numbered address; only even numbered addresses are allowed in many CPUs.

Illegal opcode: detects if the CPU during an instruction fetch reads a value from memory (or the instruction cache) that doesn't correspond to a valid instruction. This error can occur if the program counter is erroneously loaded with an address pointing to a data area rather than a program code area.



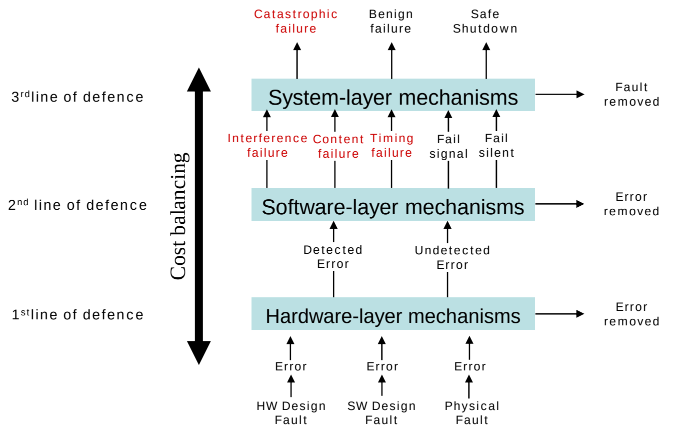
Examples of CPU exceptions (2)

Privilege violation: detects if a user program attempts to execute an instruction which is allowed only for programs that execute in the superuser mode (privileged mode), such as the operating system or device drivers. User programs normally executes in user mode (normal mode).

Division by zero: detects if a program tries to divide a number with zero.

Spurious interrupt: detects if an interrupt is signalled but no interrupt vector is provided by the interrupting device. (The interrupt vector tells the CPU which device it was that raised the interrupt signal and thereby indicates which interrupt service routine that the CPU shall execute.)

Layered fault tolerance in distributed systems



Examples of error detection mechanisms at different layers

Hardware layer

- **CPU hardware exceptions:** *Bus error, Address error, Illegal opcode, Privilege violation, Division by zero, Spurious interrupt, etc.*
- **Error detecting and correcting codes in main memory, caches, internal buffers**
- **Special hardware circuits (often connected to the non-maskable interrupt signal of a CPU):** *Power supply monitor, Network (bus) guardian.*
- **Watchdog timer (sometimes implemented as combination of HW and SW)**

Software layer

- **Compiler:** *Type checking of constrained variables, Value range overflow, Loop iteration bound overflow*
- **OS:** *Processing time overflow, consistency checks on OS data,*
- **Application:** *time-redundant execution of tasks, application specific consistency checks*

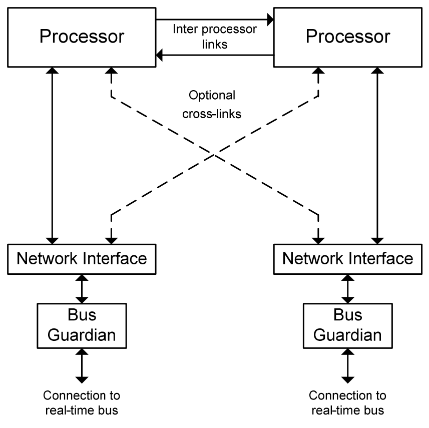
System layer

- **End-to-end checksums**
- **Comparison of results produced by two nodes**
- **Voting on results produced by three or more nodes**

Outline

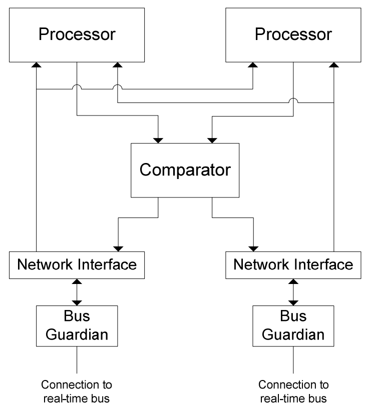
- Characterization of Failure Modes
- Byzantine failures
- Layered fault tolerance
- Error detection techniques
- Self-checking nodes

Self-checking node supporting software implemented message comparison



- The processors executes the same programs and exchange copies of outgoing messages via the inter processor links
- They compare the message copies and stops execution if the copies do not match.
- An error counter stores the number of mismatches that has occurred.
- The node is restarted after a mismatch only if the value of the error counter is below a predefined threshold
- The bus guardian protects the bus from erratic behavior (e.g., babbling idiot) of the network interfaces

Self-checking node with message comparison in hardware

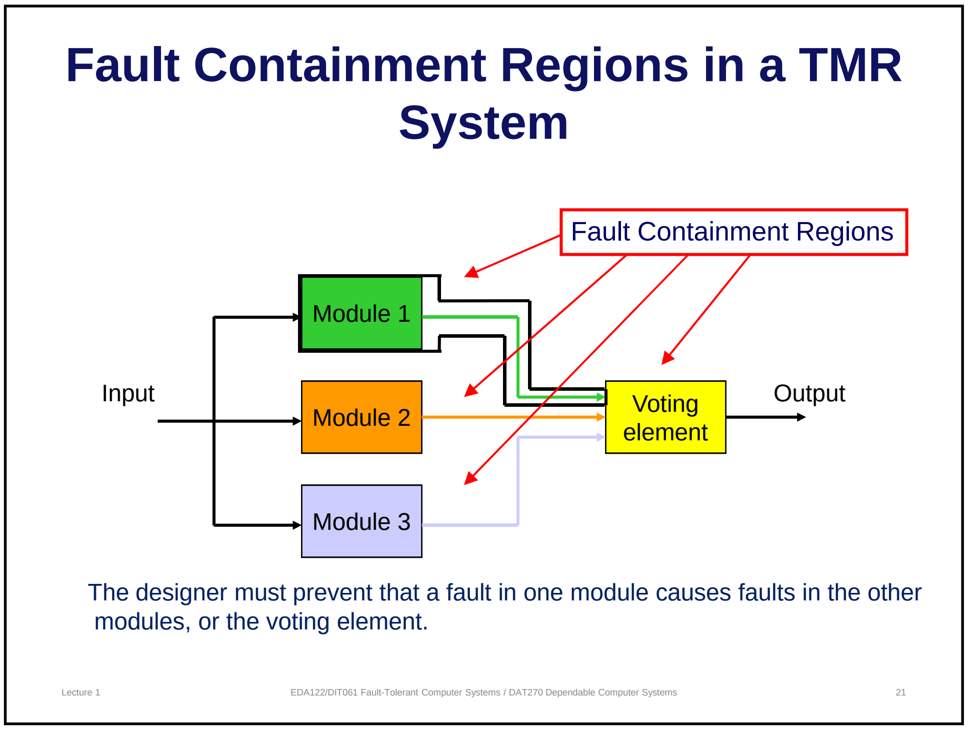


- Processor failures are detected by duplication and comparison
- The processors produce replicated messages that are compared by the comparator.
- The network interfaces receive messages from the comparator and send them to other nodes via two redundant real-time busses.
- The payload in the messages are protected by end-to-end checksums added by the processors.
- End-to-end checksums can be used to ensure that faults in the comparator and network interfaces are detectable by the service users (other nodes).

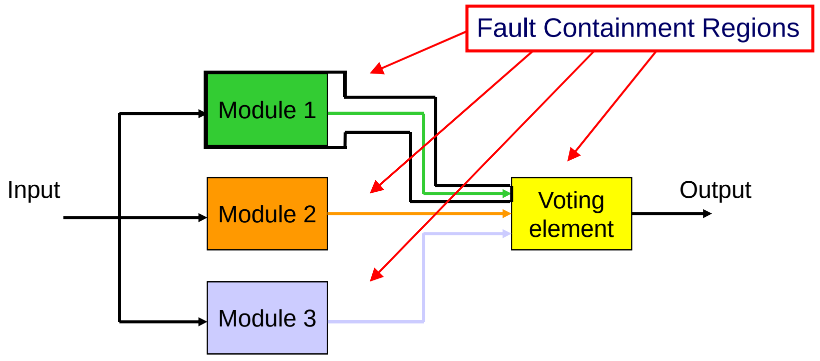
Fundamental Concepts
Fault/Error Containment

Fault/Error containment aims at preventing faults/errors from affecting other (redundant) units in the system.

- A fault-tolerant system consist of several ***fault/error containment regions***

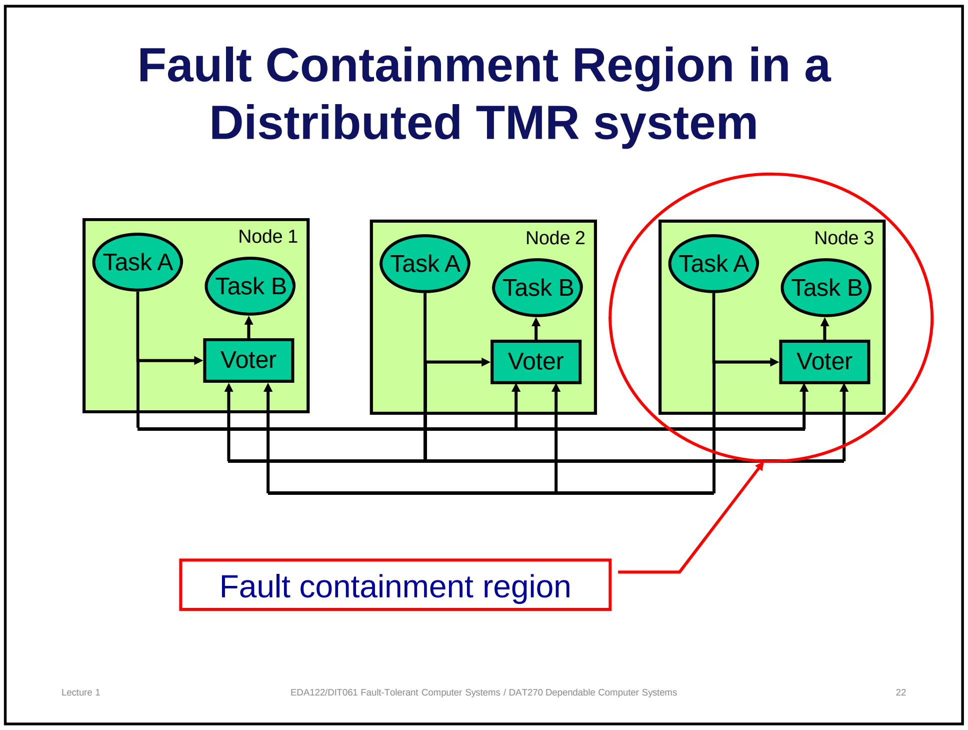


Fault Containment Regions in a TMR System

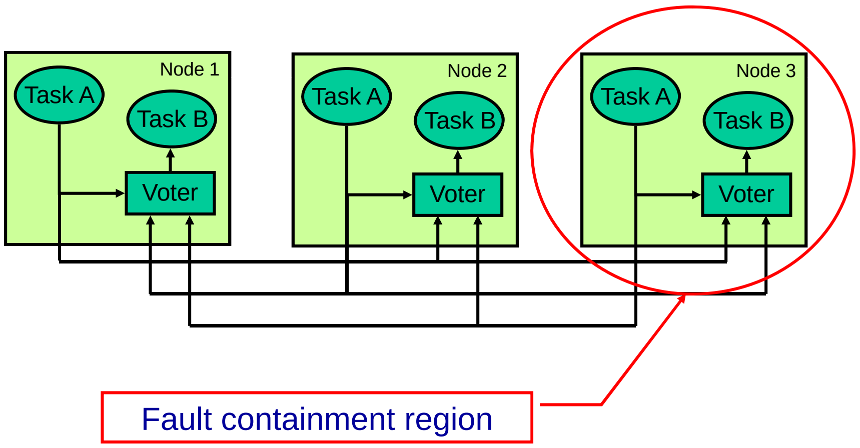


The designer must prevent that a fault in one module causes faults in the other modules, or the voting element.

Lecture 1 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 21

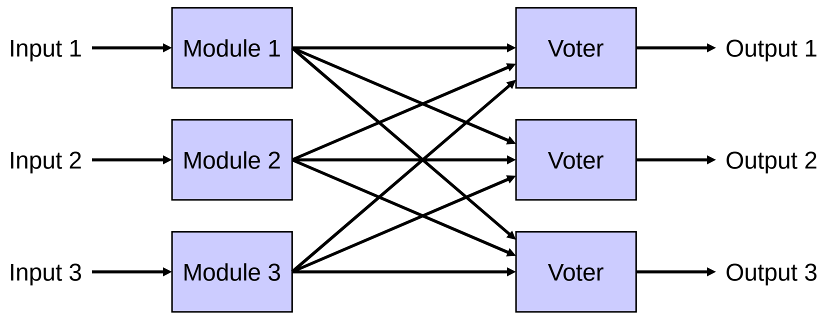


Fault Containment Region in a Distributed TMR system



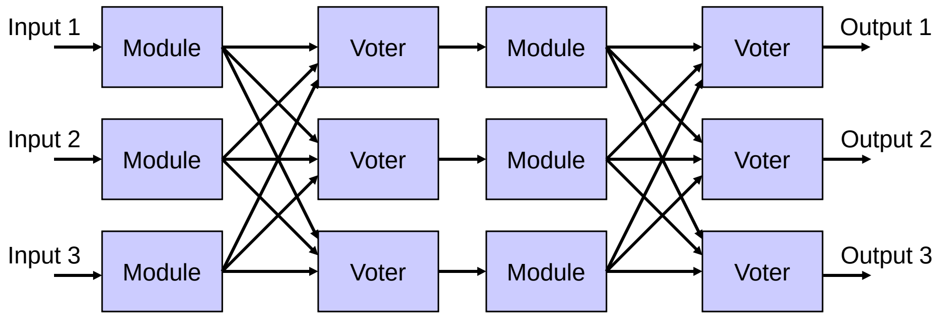
Lecture 1 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 22

Masking Redundancy
TMR with triple voting and triple inputs

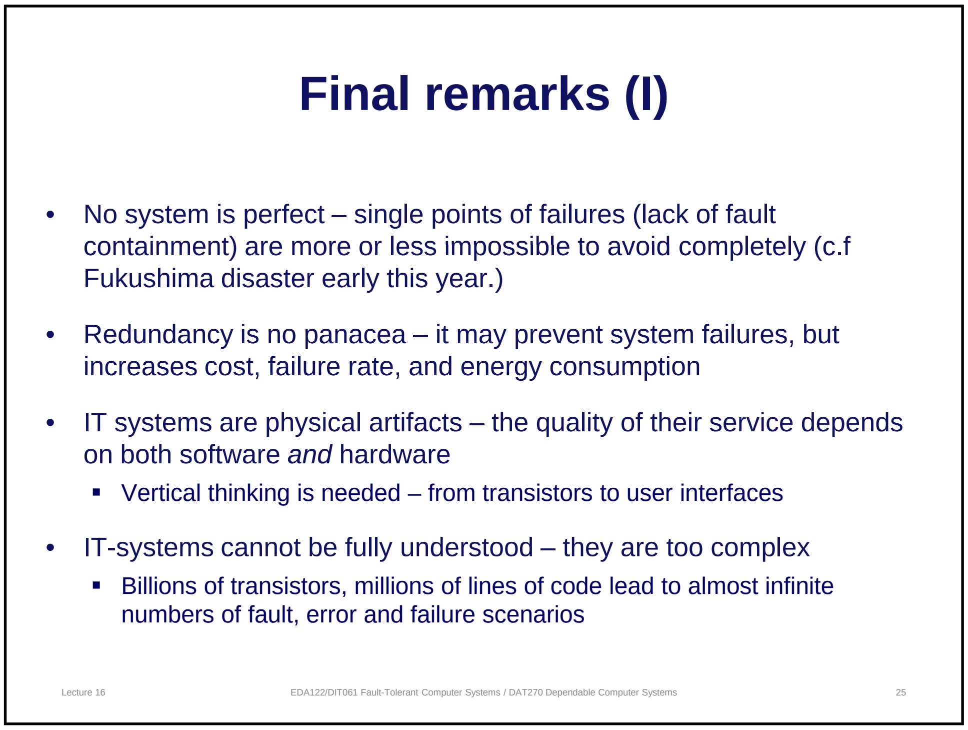


Lecture 1 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 23

Masking Redundancy
Multi-stage TMR

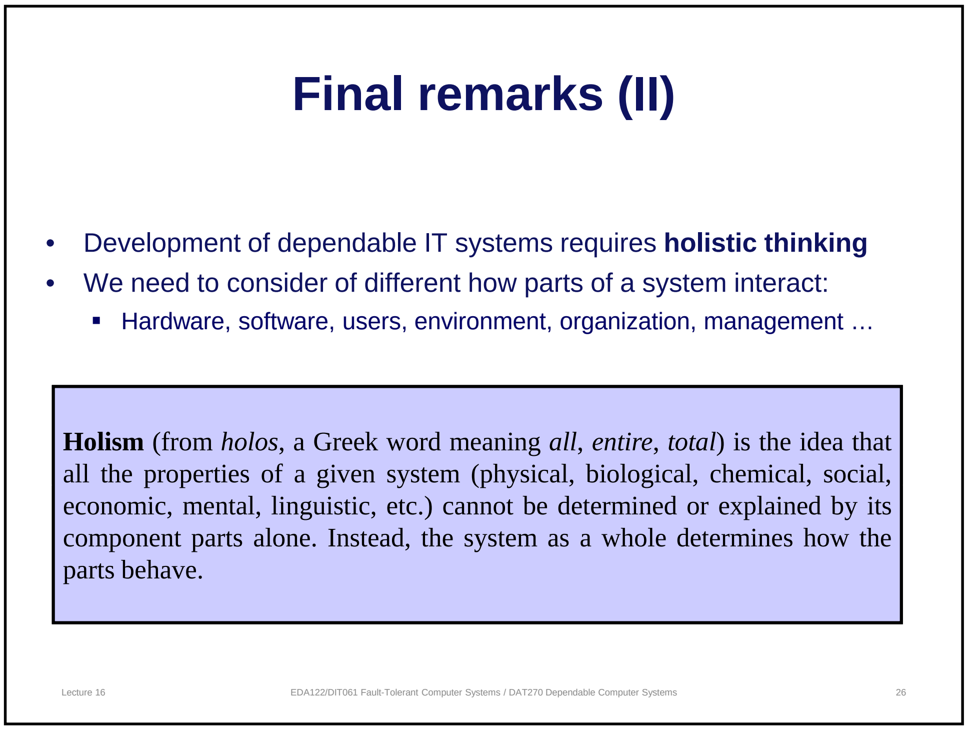


Lecture 1 EDA122/DIT061 Fault-Tolerant Computer Systems / DAT270 Dependable Computer Systems 24



Final remarks (I)

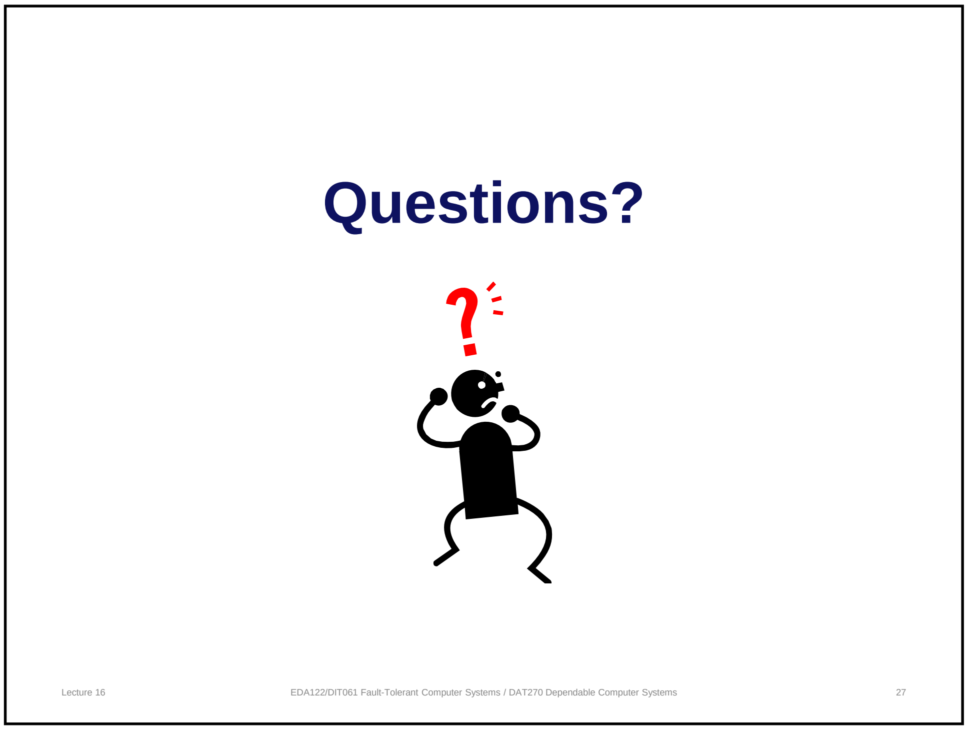
- No system is perfect – single points of failures (lack of fault containment) are more or less impossible to avoid completely (c.f Fukushima disaster early this year.)
- Redundancy is no panacea – it may prevent system failures, but increases cost, failure rate, and energy consumption
- IT systems are physical artifacts – the quality of their service depends on both software *and* hardware
  - Vertical thinking is needed – from transistors to user interfaces
- IT-systems cannot be fully understood – they are too complex
  - Billions of transistors, millions of lines of code lead to almost infinite numbers of fault, error and failure scenarios



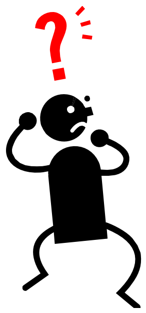
Final remarks (II)

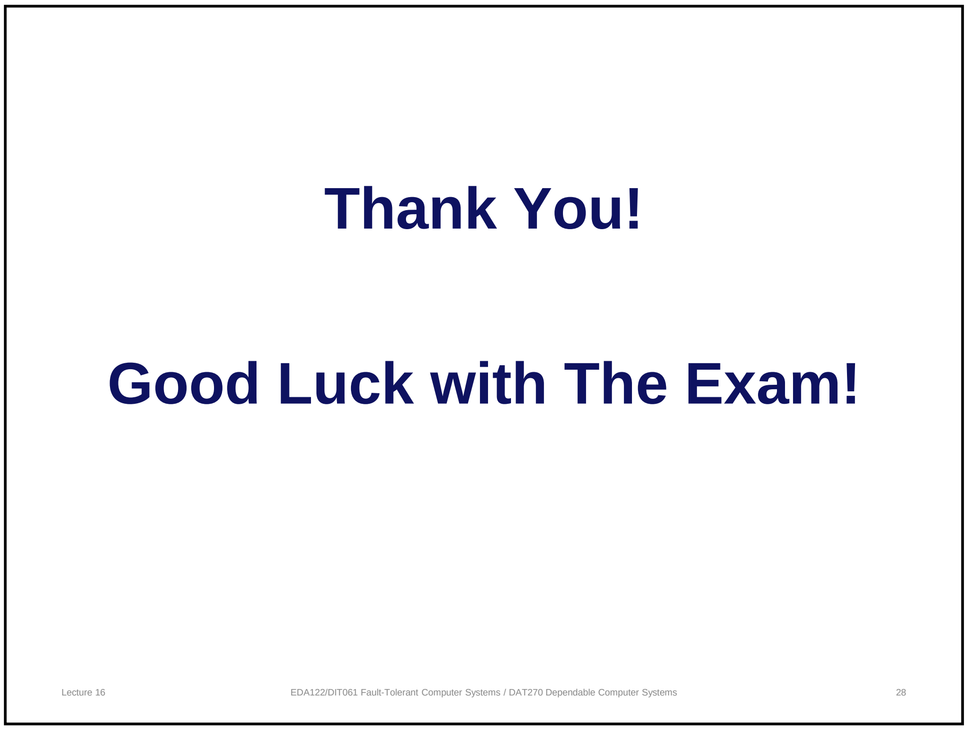
- Development of dependable IT systems requires **holistic thinking**
- We need to consider of different how parts of a system interact:
  - Hardware, software, users, environment, organization, management …

Holism (from *holos*, a Greek word meaning *all*, *entire*, *total*) is the idea that all the properties of a given system (physical, biological, chemical, social, economic, mental, linguistic, etc.) cannot be determined or explained by its component parts alone. Instead, the system as a whole determines how the parts behave.



Questions?





Thank You!

Good Luck with The Exam!