Sources of Faults in Computer Systems

EECE 513: Fault-tolerant digital systems

Learning Objectives

- Specify fault models for different techniques

- List faults in each layer of the system stack
  - Why they occur ?
  - How do they manifest ?

- Apply fault-tolerance techniques at the appropriate layer of the system stack

What is a fault model ?

- A concrete description of
  - What faults can occur
  - Where they can occur
  - When they can occur

- **Example**: When a packet is read from a noisy channel, it can have a **single word** corrupted by an error in its **header or body**

Why do we need fault models ?

- Principled way to reason about faults

- Need to qualitatively outline the space of faults before we can quantify their occurrence

- Every fault-tolerance technique is targeted to and evaluated against a fault-model
  - Even if one is not explicitly specified in the paper
  - Example: ECC targets single bit flips in memory

Examples of fault models - 1

Same Input

Processor 1

Processor 2

Voter
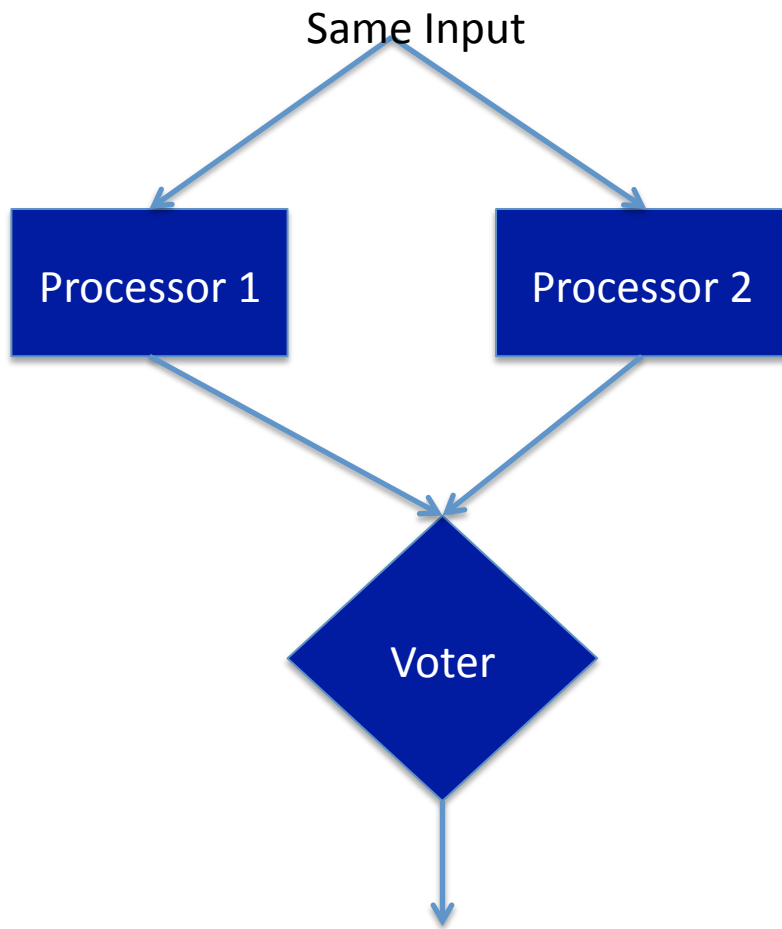
- **What ?**
  - Fault in either processor but not correlated faults in both

- **When ?**
  - Anytime during the execution of the program after input is given, but before voting starts

- **Where ?**
  - Any non-correlated fault in the processor as well as non-deterministic faults in S/W

Examples of fault models - 2

ECC-
protected
Memory

ECC-
Circuitry

Update code after write
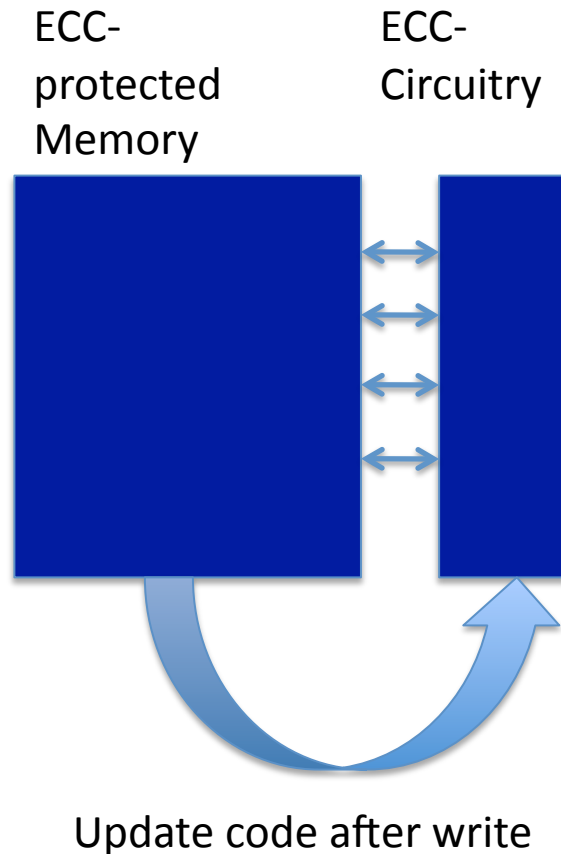
- **What ?**
  - Faults that cause corruption of memory values independently, i.e., no correlation in space

- **When ?**
  - Anytime after a value has been written to memory OR before it is read from memory

- **Where ?**
  - Anywhere in the ECC-protected memory (not including circuits)

Examples of fault models - 3

Recovery block

Switch

input

Primary Module
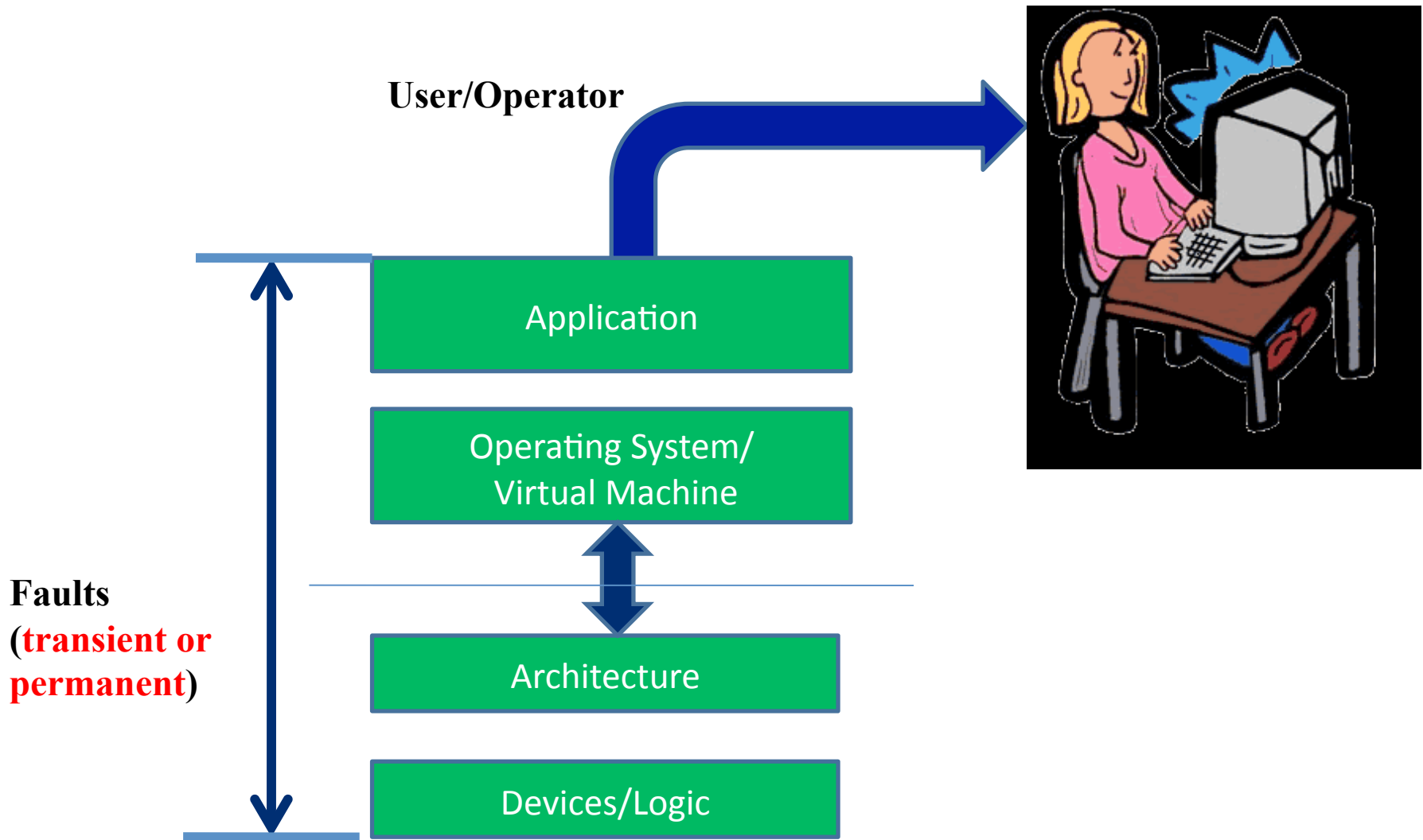
Secondary Module
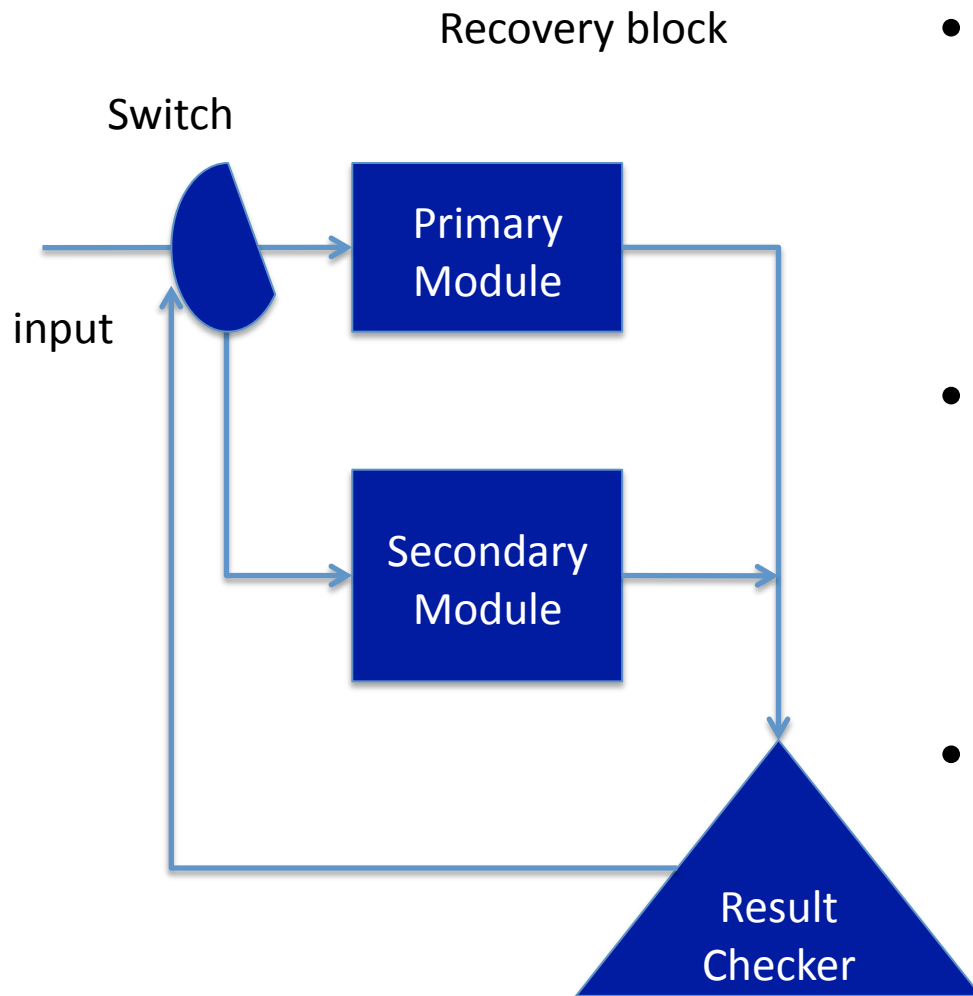
Result Checker

- **What ?**
  - Faults in the primary that are detected successfully by the result checker

- **When ?**
  - During the execution of primary, but before the result is checked

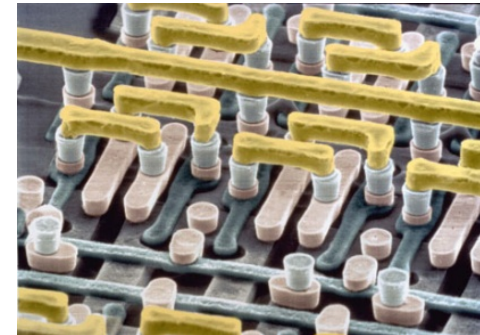- **Where ?**
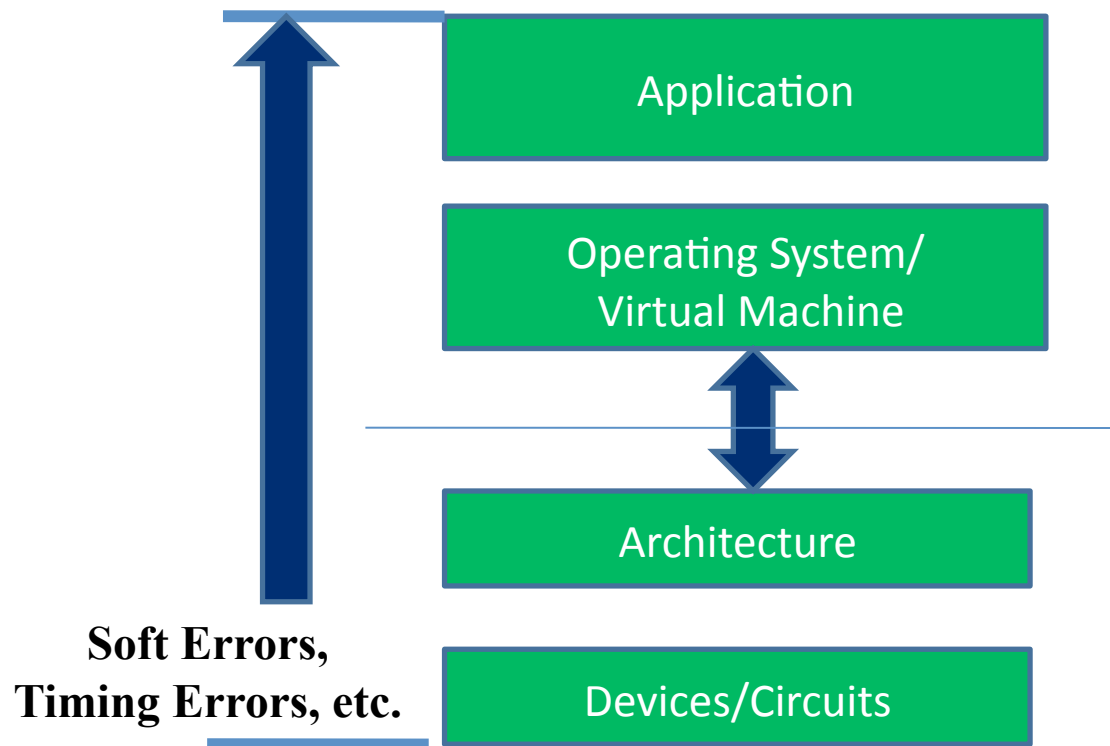 - H/W and S/W of primary

Learning Objectives

- Specify fault models for different techniques

- List faults in each layer of the system stack
  - Why do they occur ?
  - How do they manifest ?

- Apply fault-tolerance techniques at the appropriate layer of the system stack

Typical System Stack

User/Operator

Faults (transient or permanent)

Application

Operating System/ Virtual Machine

Architecture

Devices/Logic

System Stack: Logic Level

Soft errors

Timing errors

Application

**Operating System/
Virtual Machine**

Architecture

Devices/Circuits

**Soft Errors,
Timing Errors, etc.**

System Stack: Architectural Level

Application

Operating System/
Virtual Machine

Architecture

Devices/Circuits

Design defects +
Wearout-related defects

BUGS

intel inside ®

The Errata Series™
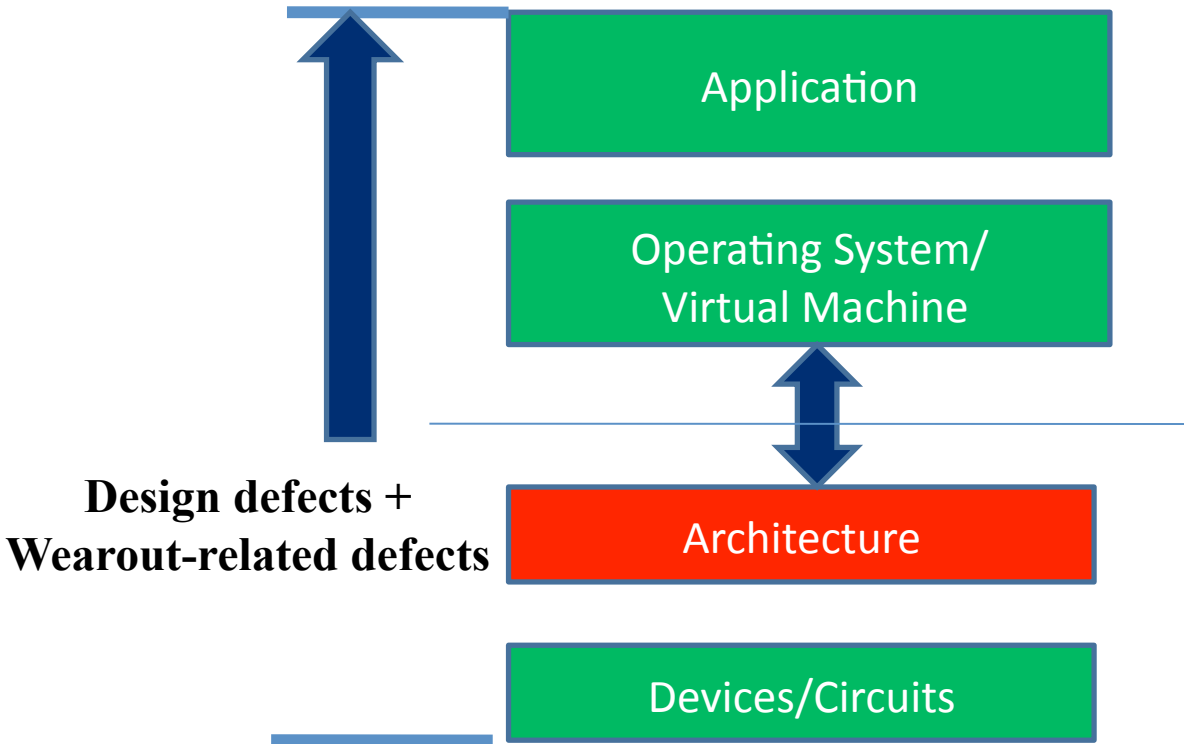
Design Defects
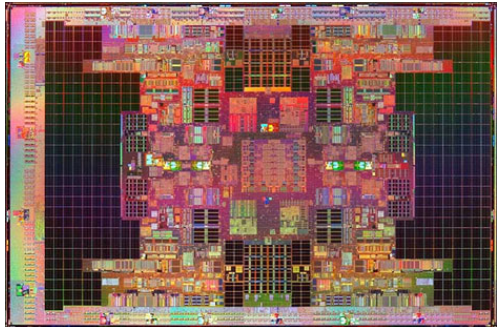
Wearout-related defects

System Stack: OS/VM Level

Kernel Error

Errors in kernel or device drivers

| Application |
| Operating System/ Virtual Machine |
| Architecture |
| Devices/Circuits |

Driver error

12

System Stack: Application Level

Concurrency bugs and Memory corruption errors

| Application |

| Operating System/ Virtual Machine |

| Architecture |

| Devices/Circuits |

Concurrency bugs
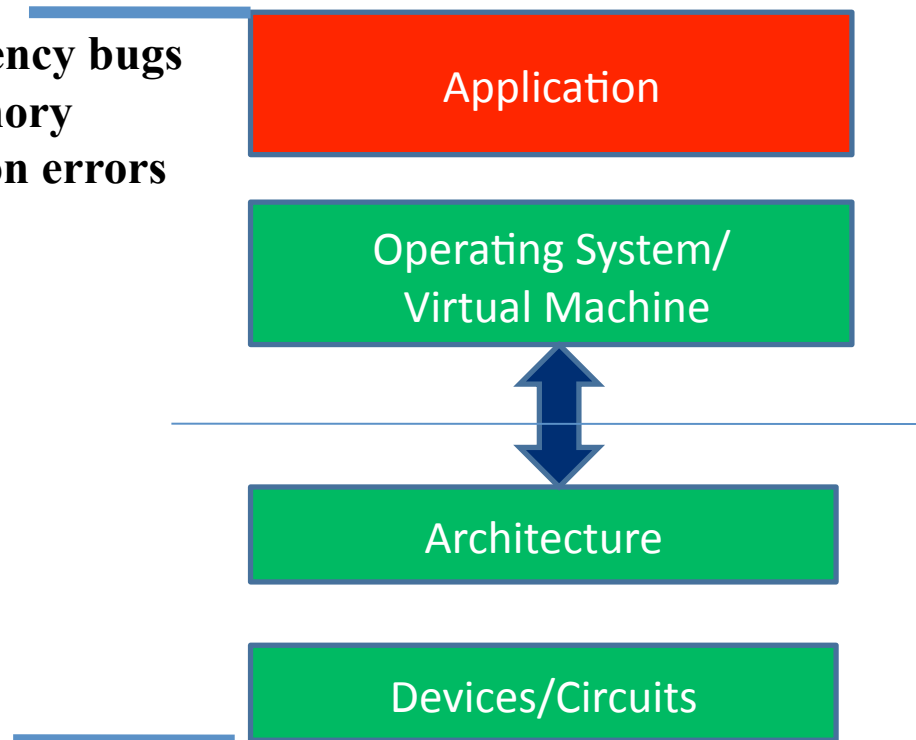
Buffer Overflow Attacks
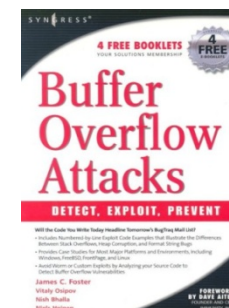
Memory corruption

Learning Objectives

- Specify fault models for different techniques

- List faults in each layer of the system stack
  - Why do they occur ?
  - How do they manifest ?

- Apply fault-tolerance techniques at the appropriate layer of the system stack

Manifestation of Faults

- **Fault effects may be permanent or temporary**
  - Same fault may result in different effects depending on where/when it occurs
  - A soft error in the code segment is a permanent error while one in the data segment may be temporary

- **Faults may affect different layers differently**
  - A permanent fault in the logic level may manifest as a temporary fault at the architectural level if the functional unit in which it occurs is often unused

Logic Level Fault Models

- **Stuck-at-fault (Permanent)**
  - Assume that some gate/line gets "stuck"
  - Can be stuck-at-0 or stuck-at-1
  - May not correspond to real physical faults
  - Very useful for evaluating test cases in ATPG

- **Bit-Flip Model (Transient)**
  - Can be caused by cosmic rays/alpha particles striking flip-flops or logic gates (Soft errors)
  - Leads to one or more bits getting "flipped"

Architectural Level Fault Models

- **Permanent Errors**
  - Some functional unit in the processor fails (e.g., an ALU stops working, cache line has a stuck-at-fault)
  - Certain instructions are always executed incorrectly due to design errors (e.g., adds always encounter errors when value overflows register width)

- **Transient Errors**
  - Some unit experiences an error for 1 cycle/instruction (e.g., an entry in the ROB has a bit-flip for 1 cycle)
  - Cache line has a single bit-flip due to cosmic ray strike

OS Level Fault Models

- **Permanent Error**
  - An instruction or data item was corrupted by a fault in the disk image of the OS
  - Device experiences a permanent failure

- **Transient Error**
  - An OS data/code page in memory is corrupted
  - A device experiences a transient malfunctioning
  - The kernel experiences deadlock/livelock

Application/Program Level

- **Permanent Errors**
  - Programming errors in application logic - mutation of source code or binary file
  - Corruption of configuration files/data-bases needed by the application

- **Transient Errors**
  - Software memory corruption errors -> corruption of memory locations
  - Race conditions/Atomicity violations -> lock elision

Operator/User Level

- **Permanent Errors**
  - Errors in configuration files/databases due to user's carelessness or misunderstanding of parameters
 - Wrong semantic model of the application

- **Transient Errors**
  - User types in incorrect command/GUI action due to carelessness or oversight
  - Operator attempts to upgrade hardware or software and upgrades the wrong component/package

Learning Objectives

- Specify fault models for different techniques

- List faults in each layer of the system stack
  - Why do they occur ?
  - How do they manifest ?

- Apply fault-tolerance techniques at the appropriate layer of the system stack

Detection Latency

Typical Recovery Latencies for a Hierarchical Fault Tolerant Design
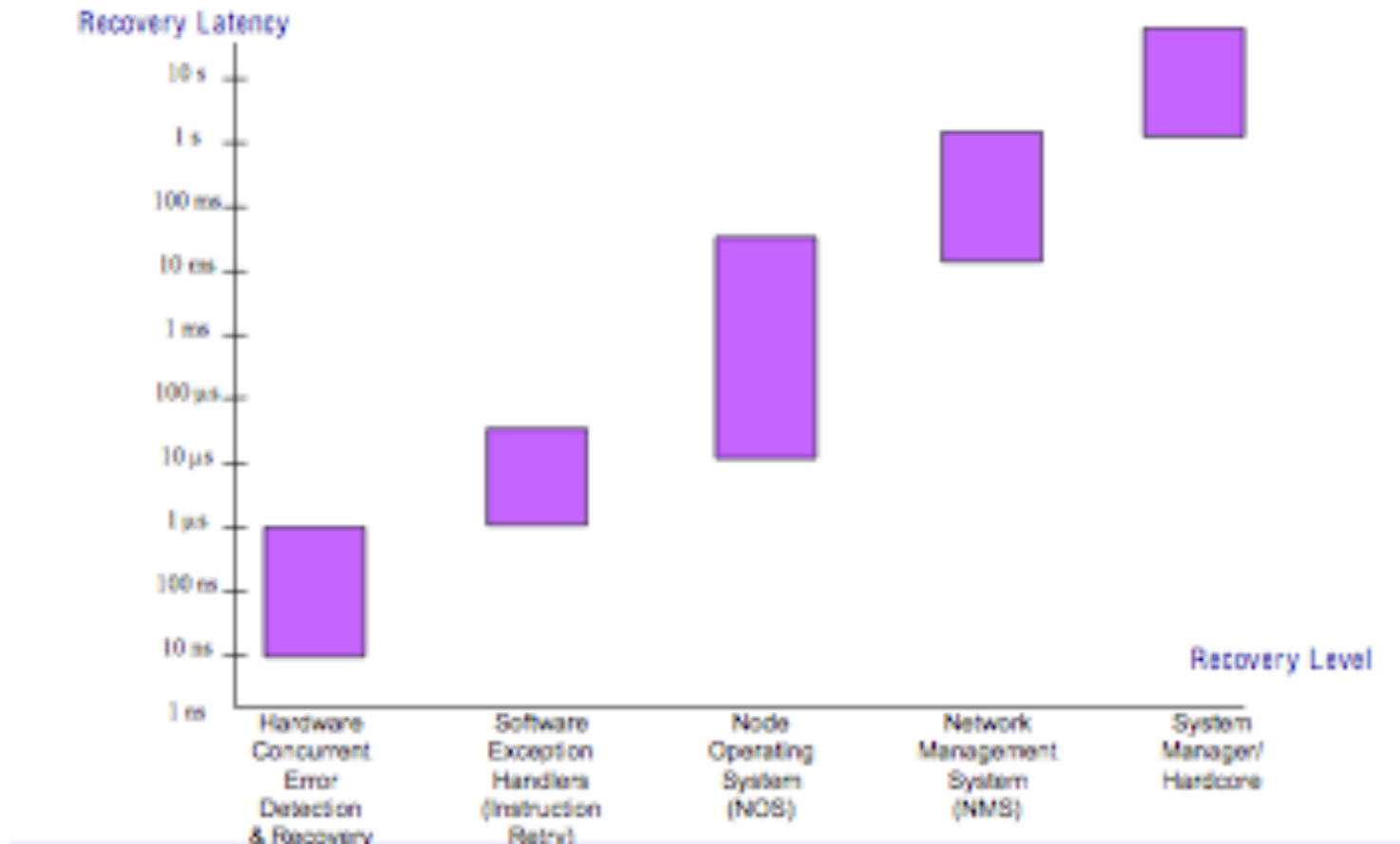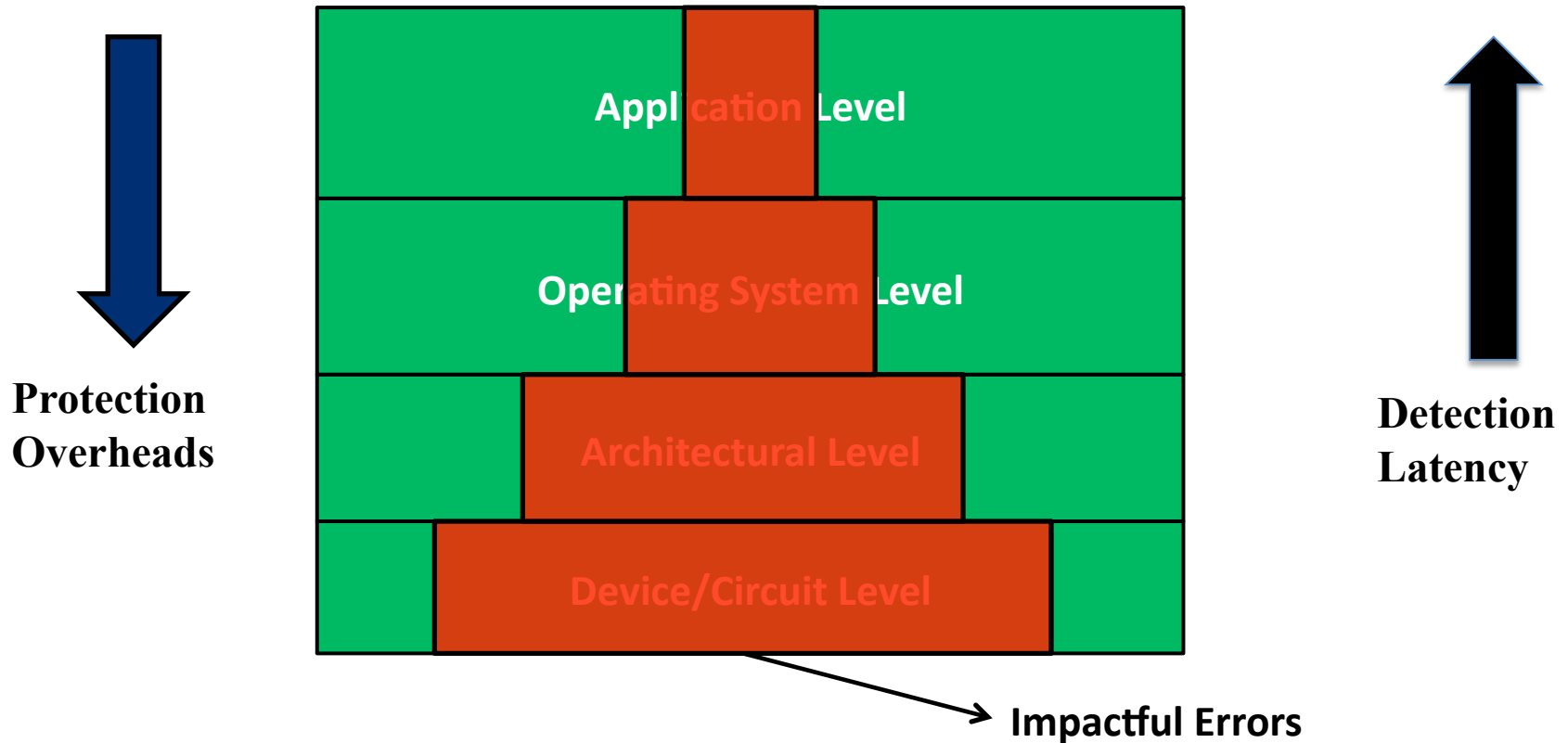
Why is detection latency important ?

- **Long-latency errors lead to more severe, harder-to-recover failures**

  – Corruption of checkpoint or file-system state

  – Propagation of errors in distributed systems

- **Early detection facilitates fault isolation**

  – Need not perform system-wide restart

  – Easier to diagnose problem's root cause

Filtering of Errors

- **Only a fraction of errors in each layer makes it to the top layer**
  - Not all state is used in the layer above
  - Some errors may be masked/overwritten

- **Example: Less than 15% of errors in flip-flops make it to the architectural state [Saggesse'05]**
  - Of these, only about 30 to 40 % affect programs [Nakka'05]
  - Not all errors that affect programs are impactful [Pattabiraman'06, Pattabiraman'09]

Multi-layer Fault tolerance

Need to balance overheads with detection latency

Application Level

Operating System Level

Architectural Level

Device/Circuit Level

Protection Overheads

Detection Latency

Impactful Errors

Learning Objectives

- Specify fault models for different techniques

- List faults in each layer of the system stack
  - Why do they occur ?
  - How do they manifest ?

- Apply fault-tolerance techniques at the appropriate layer of the system stack

Summary

- Fault models are important for qualifying (and quantifying) the scope of reliability techniques

- Faults occur at different layers of the system stack
  - Same fault manifested differently at different layers
  - Higher layer faults may be filtered by lower layers

- Fault tolerance techniques should judiciously balance error detection latency and overheads