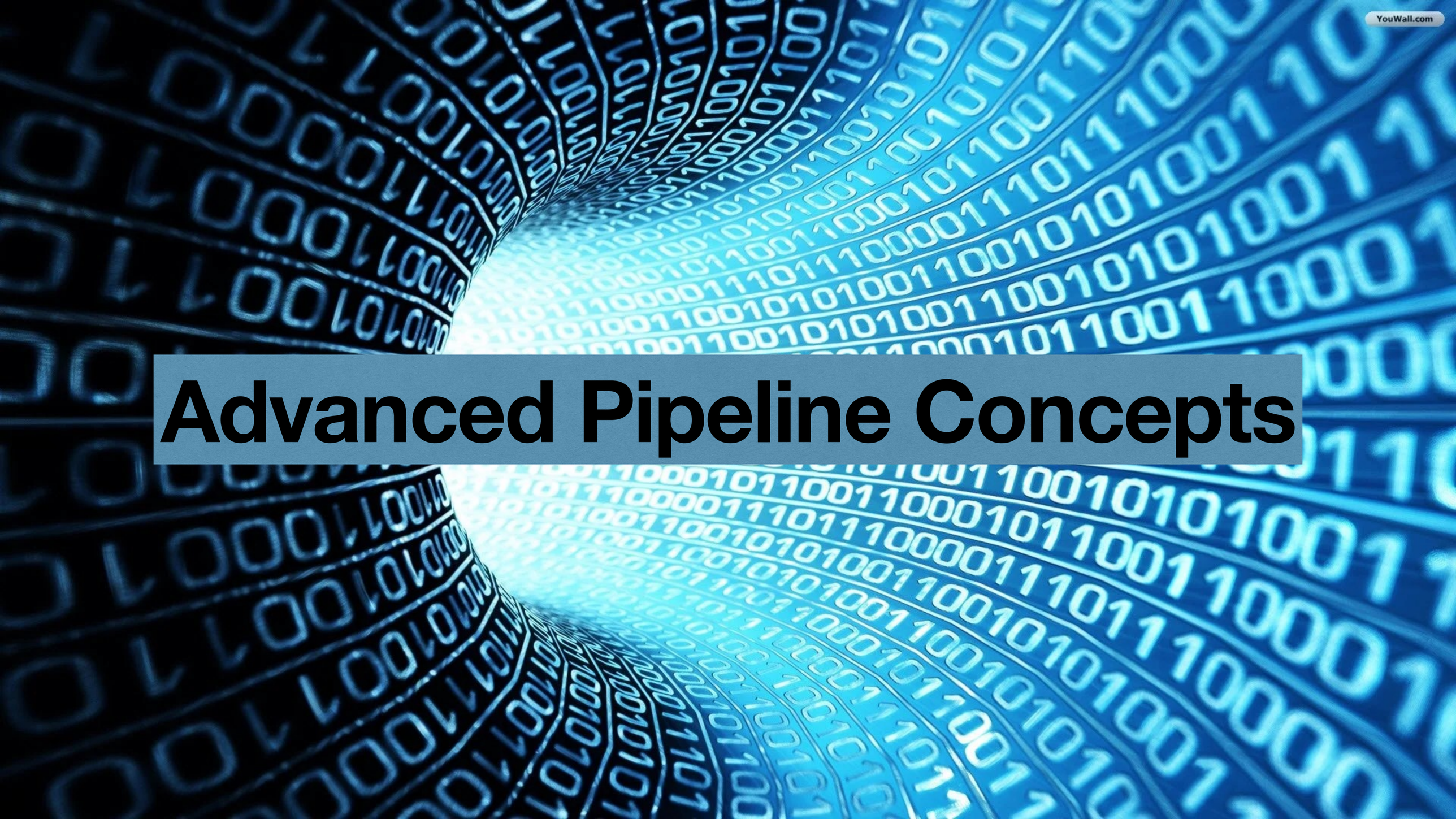
Digital Logic Design + Computer Architecture

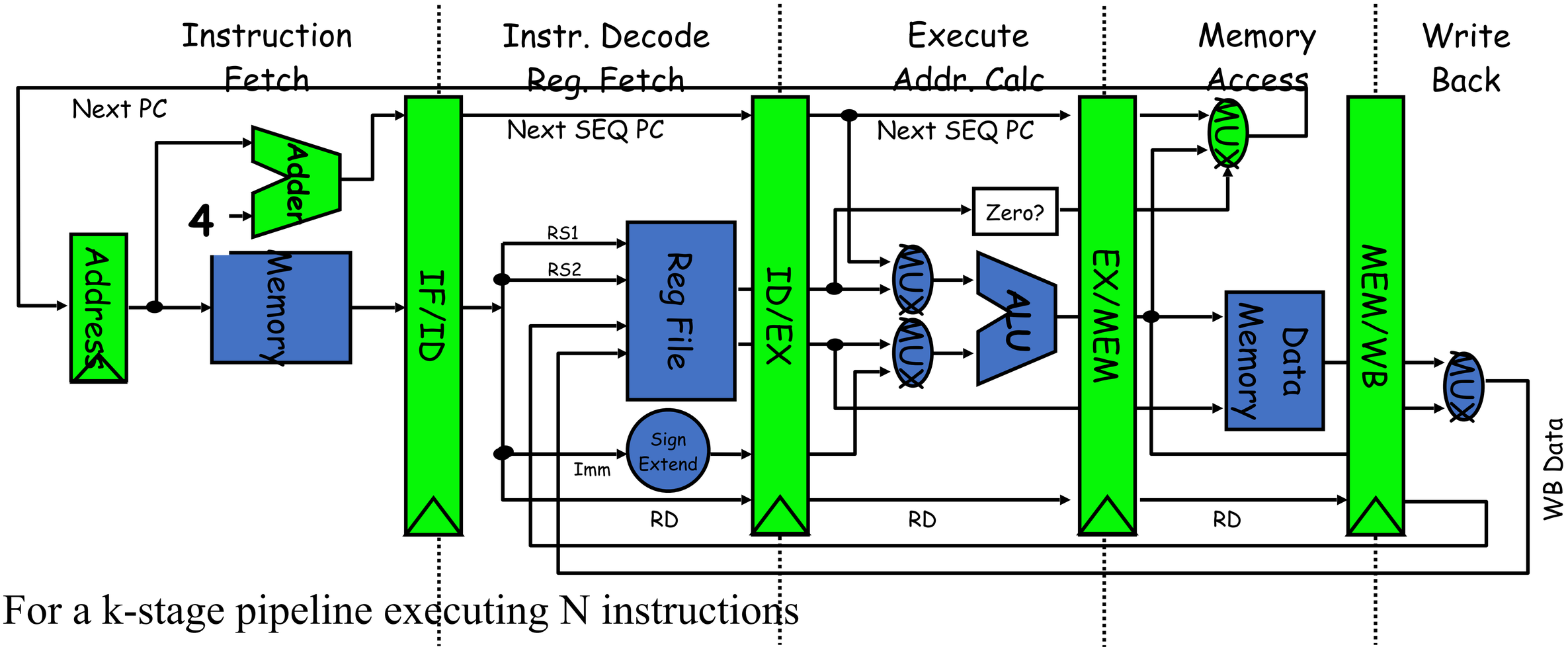
Sayandeep Saha

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Bombay



Advanced Pipeline Concepts

Pipeline Recap…



For a k-stage pipeline executing N instructions

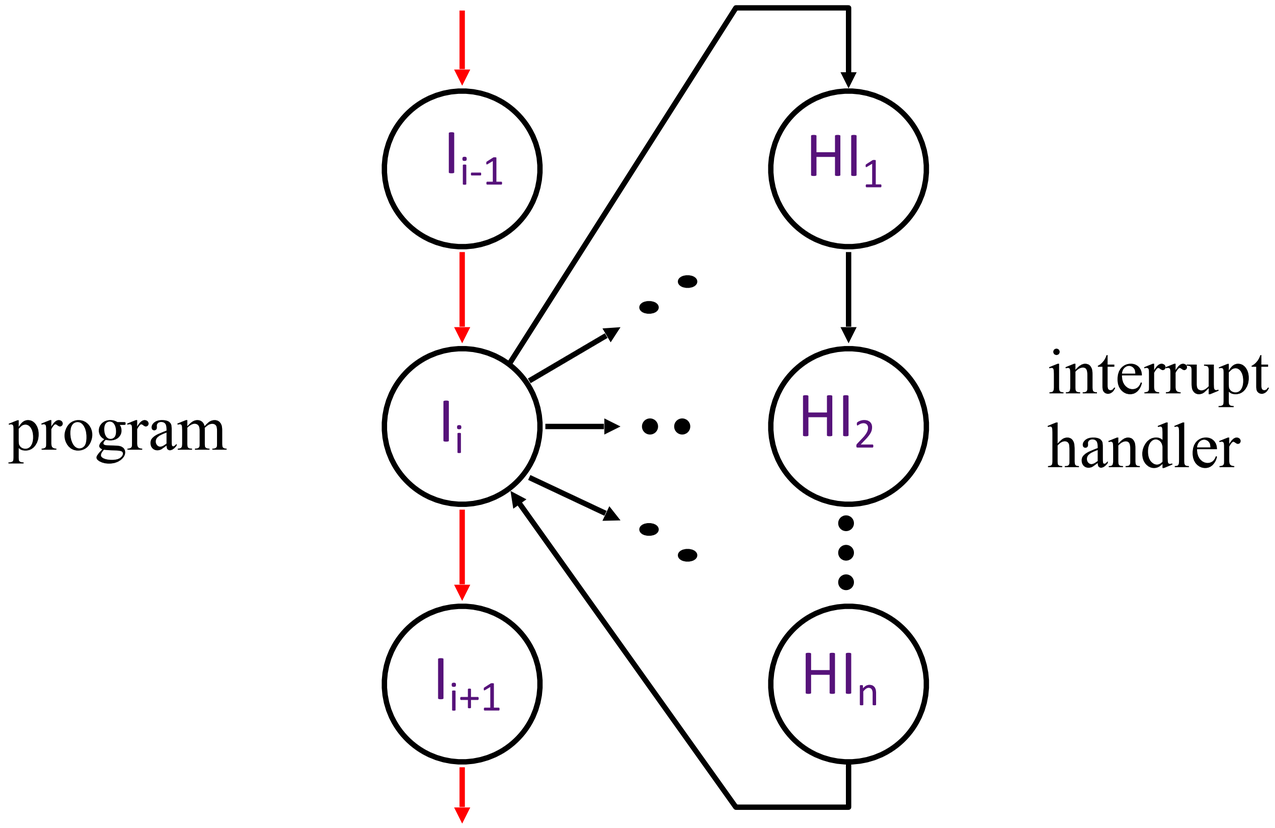
first instruction: k cycles

Next N-1 instructions: N-1 cycles, total = K + (N-1) cycles

Exception/Interrupt

An unscheduled event that disrupts program (instructions) in action.

Interrupt Handling

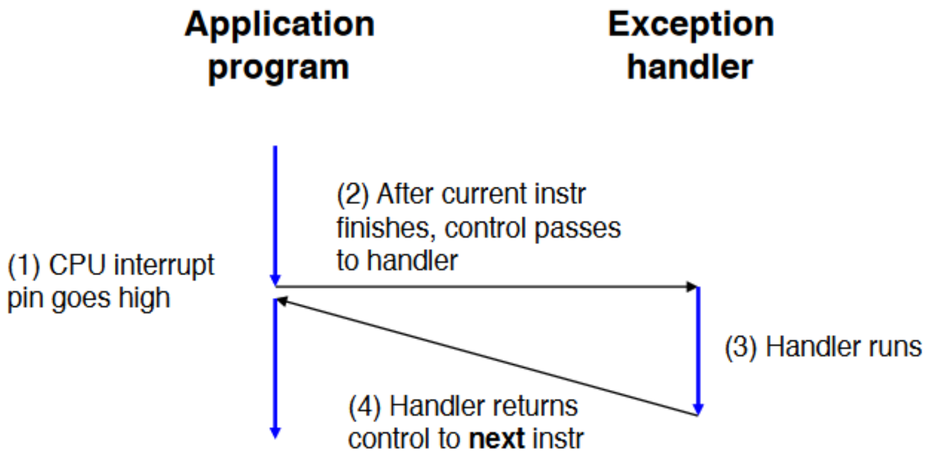


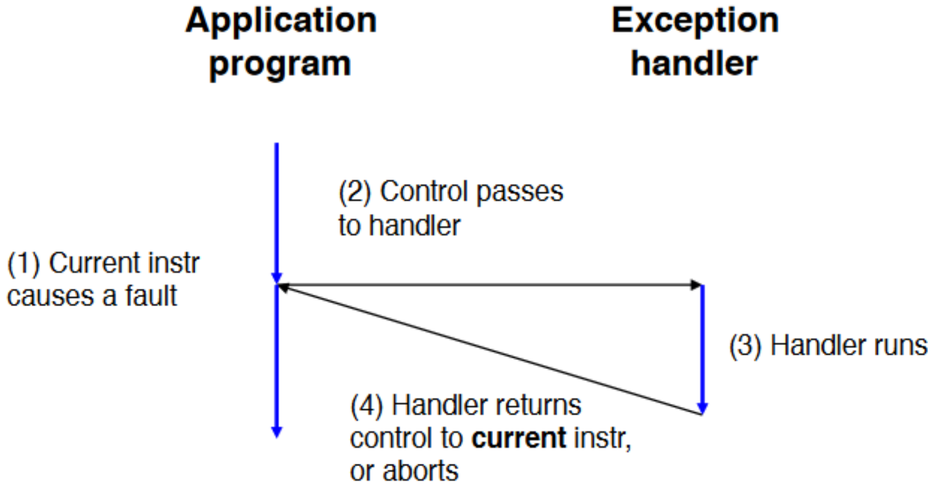
An *external or internal event* that needs to be processed. The event is usually unexpected or rare from program's point of view.

Types of Interrupts

- Asynchronous: an *external event*
  - input/output device service-request
  - timer expiration
  - power disruptions, hardware failure
- Synchronous: an *internal event (a.k.a. traps or exceptions)*
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception, misaligned memory access
  - *virtual memory exceptions:* page faults, TLB misses, protection violations
  - system calls, e.g., jumps into kernel

Interrupt and Exception





Interrupt Handler

Exception program counter (EPC): address of the offending instruction,

Saves EPC before enabling interrupts to allow nested interrupts

Need to mask further interrupts at least until EPC can be saved

Need to read a *status register* that indicates the cause of the interrupt

Handshake between processor and the OS

Processor:

stops the offending instruction,

makes sure all prior instructions complete,

flushes all the future instructions (in the pipeline)

Sets a register to show the cause

Saves EPC

Disables further interrupts

Jumps to pre-decided address (cause register or vectored)

Handshake between processor and the OS

OS:

Looks at the cause of the exception

Interrupt handler saves the GPRs

Handles the interrupt/exception

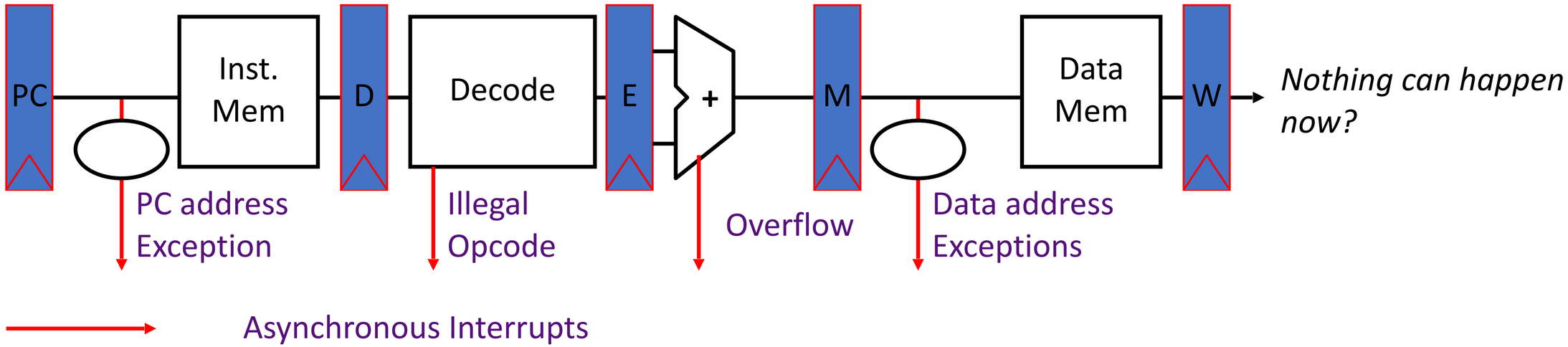
Calls RFE

Contd.

Uses a special indirect jump instruction RFE (*return-from-exception*) which

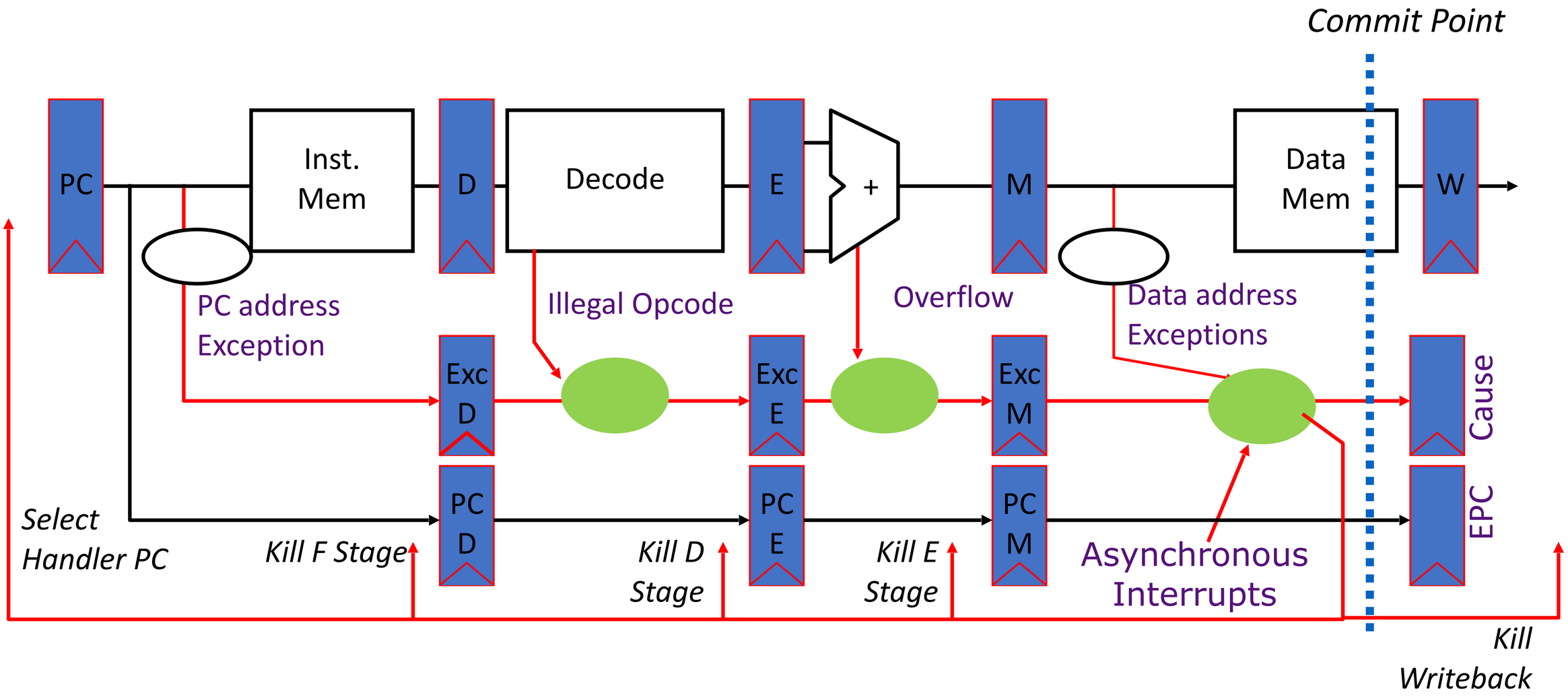
- enables interrupts
- restores the processor to the user mode

Exception handling and Pipelining



- When do we stop the pipeline for *precise* interrupts or exceptions?
- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

Exception handling and Pipelining



Contd.

- Hold exception flags in pipeline until commit point for instructions that will be killed
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- If exception at commit: update cause and EPC registers, kill all stages, inject handler PC into fetch stage

SuperScalar: More Instructions Per Cycle (IPC)

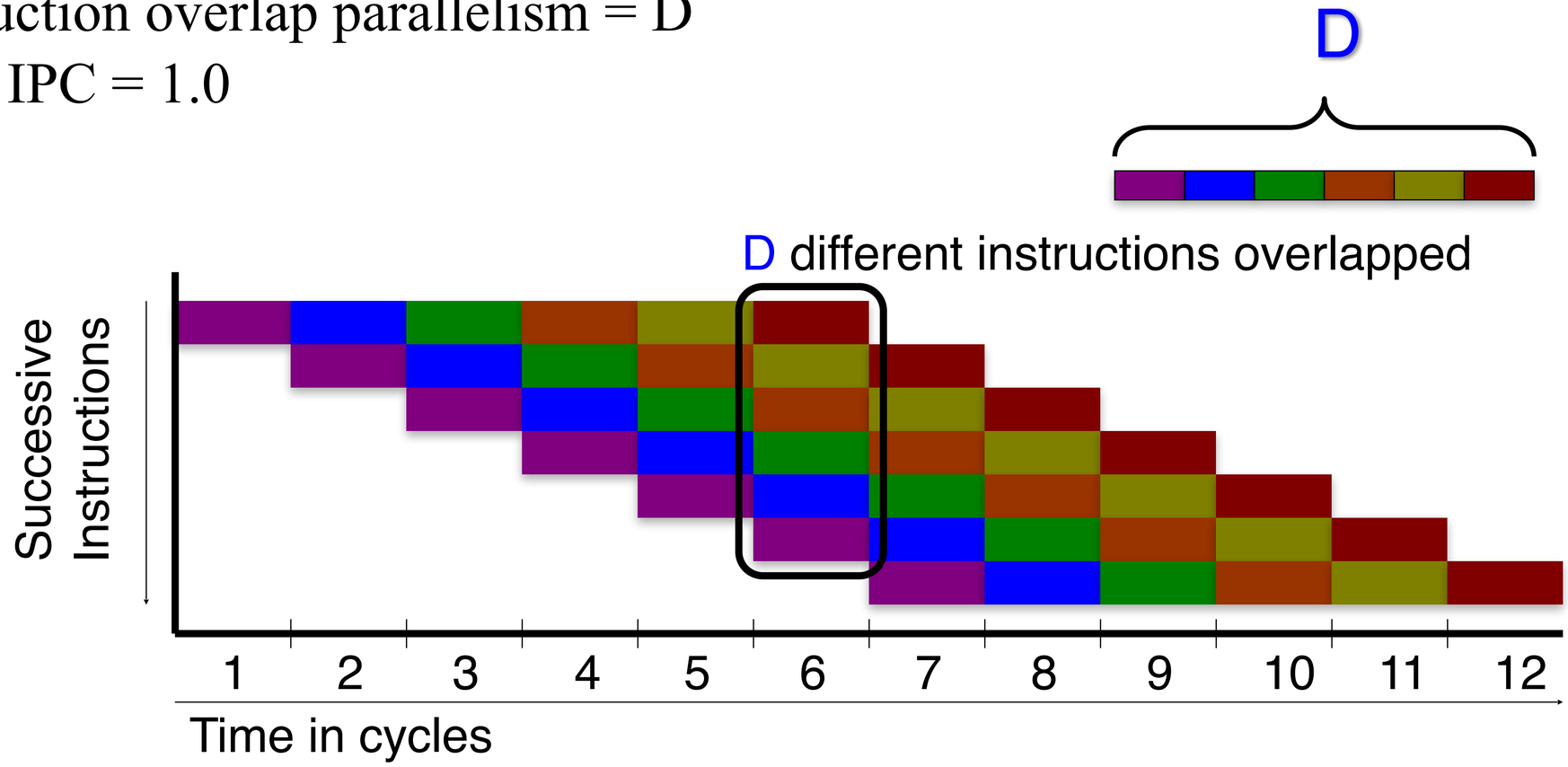
Beyond Scalar

- Scalar pipeline limited to CPI ≥ 1.0
  - Can never run more than 1 insn per cycle

- "Superscalar" can achieve CPI ≤ 1.0 (i.e., IPC ≥ 1.0)
  - *Superscalar* means executing multiple insns in parallel

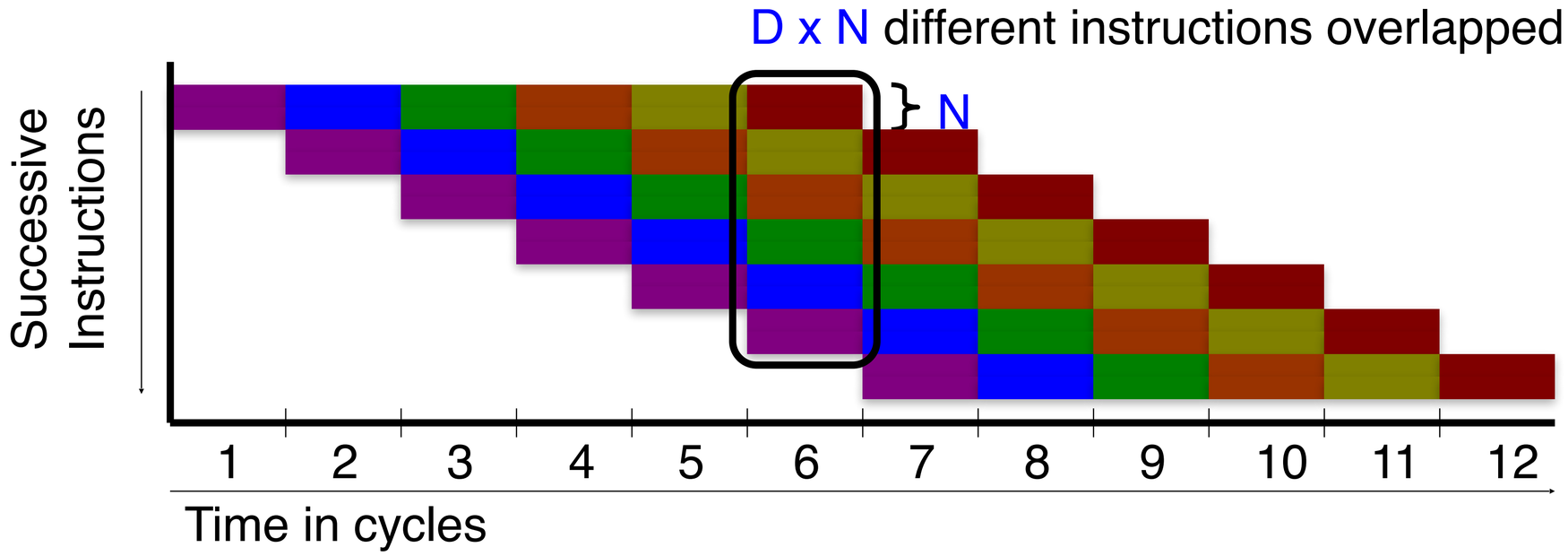
Instruction Level Parallelism (ILP)

- Scalar pipeline (baseline)
  - Instruction overlap parallelism = D
  - Peak IPC = 1.0



Superscalar Processor

- Superscalar (pipelined) Execution
  - Instruction parallelism = D x N
  - Peak IPC = N per cycle



What is the deal?

We get an IPC boost if the number of instructions fetched in one cycle are independent ☺

Complicates datapaths, multi-ported structures, complicates exception handling