CASPER

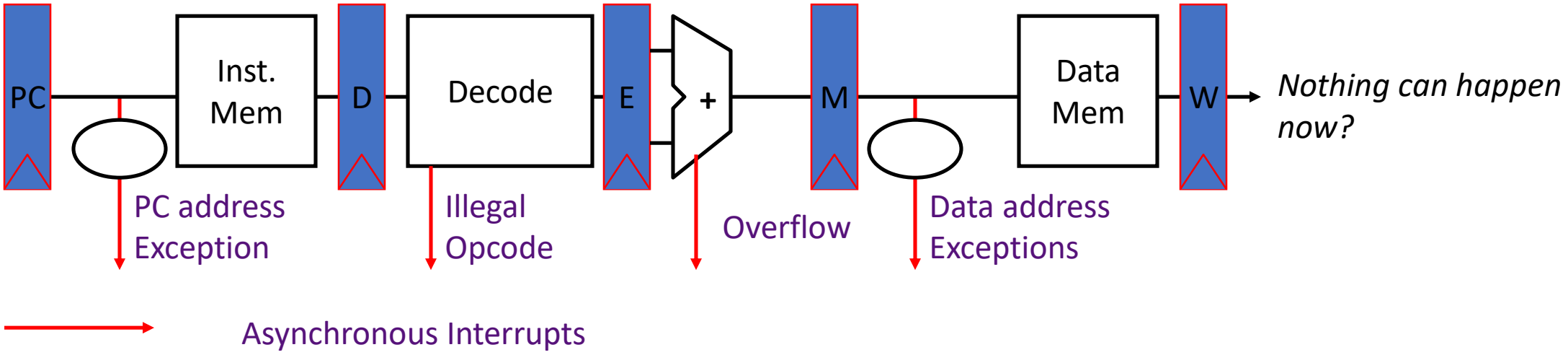
CS230: Digital Logic Design and Computer Architecture

Lecture 16: Beyond scalar and performance evaluation

https://www.cse.iitb.ac.in/~biswa/courses/CS230/autumn23/main.html

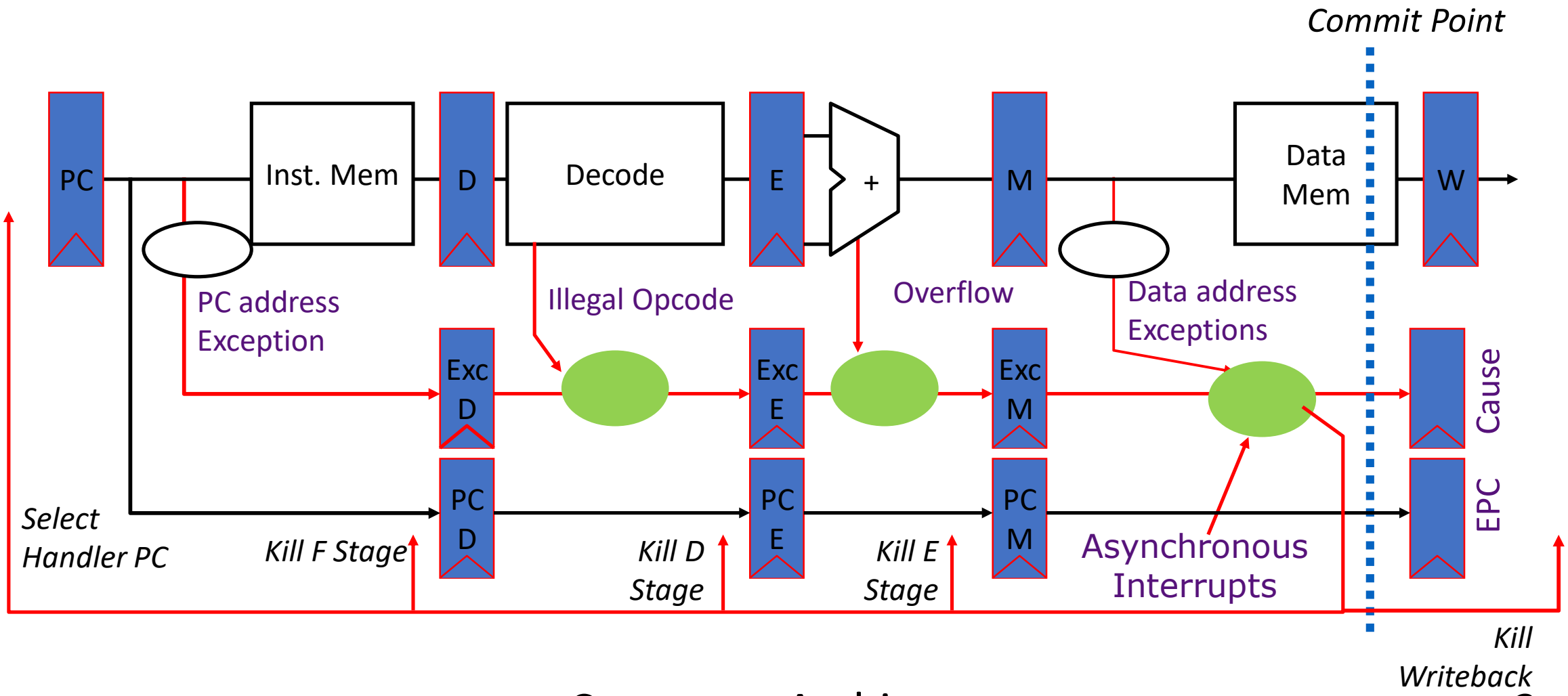
https://www.cse.iitb.ac.in/~biswa/

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?

Contd.


Commit Point
PC
Inst. Mem
D
Decode
E
+
M
Data Mem
W
PC address Exception
Illegal Opcode
Overflow
Data address Exceptions
Exc D
Exc E
Exc M
Cause
PC D
PC E
PC M
EPC
Select Handler PC
Kill F Stage
Kill D Stage
Kill E Stage
Asynchronous Interrupts
Kill Writeback

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

Moving on in the pursuit of IPC++

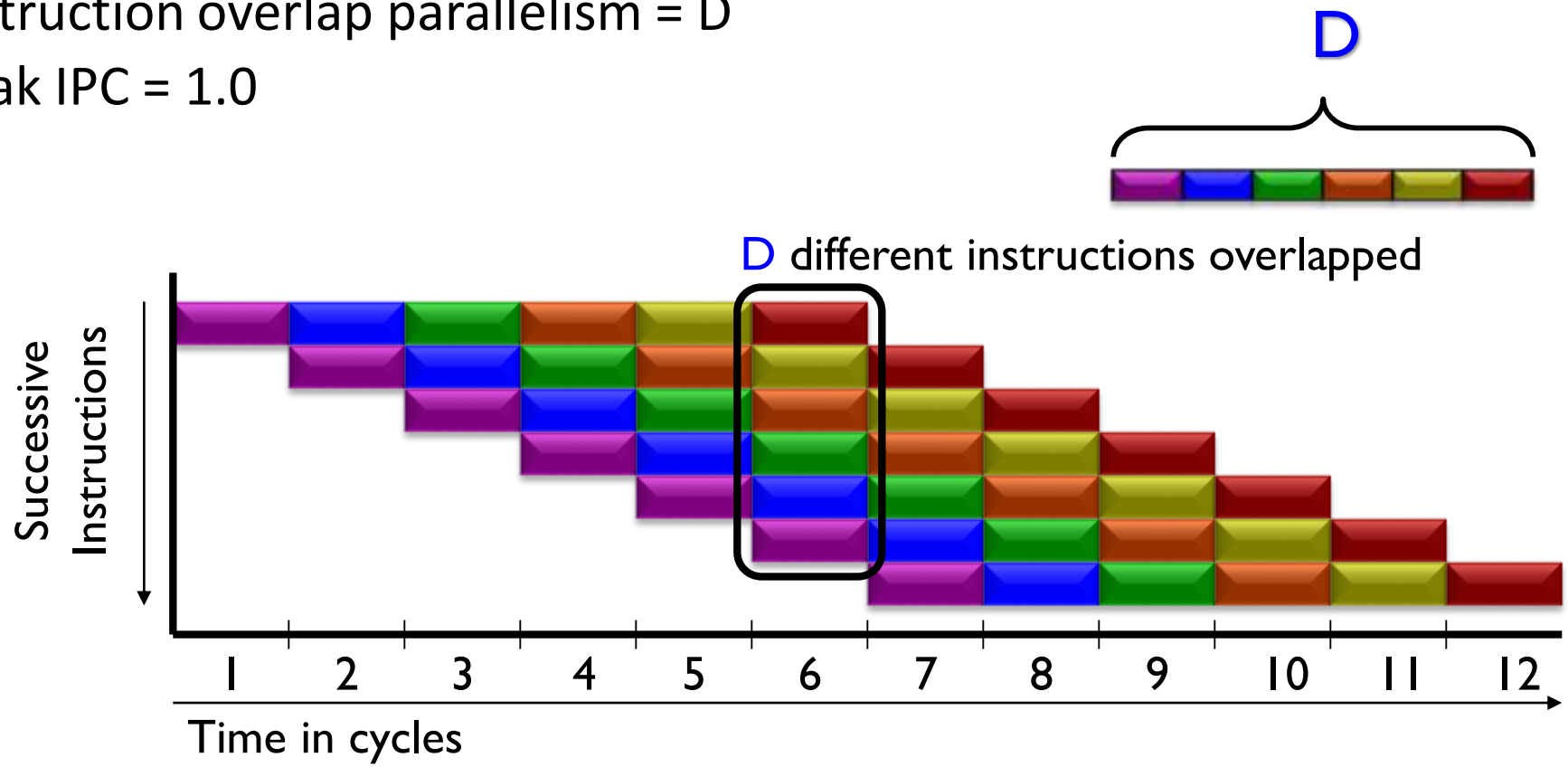
Beyond Scalar

- 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

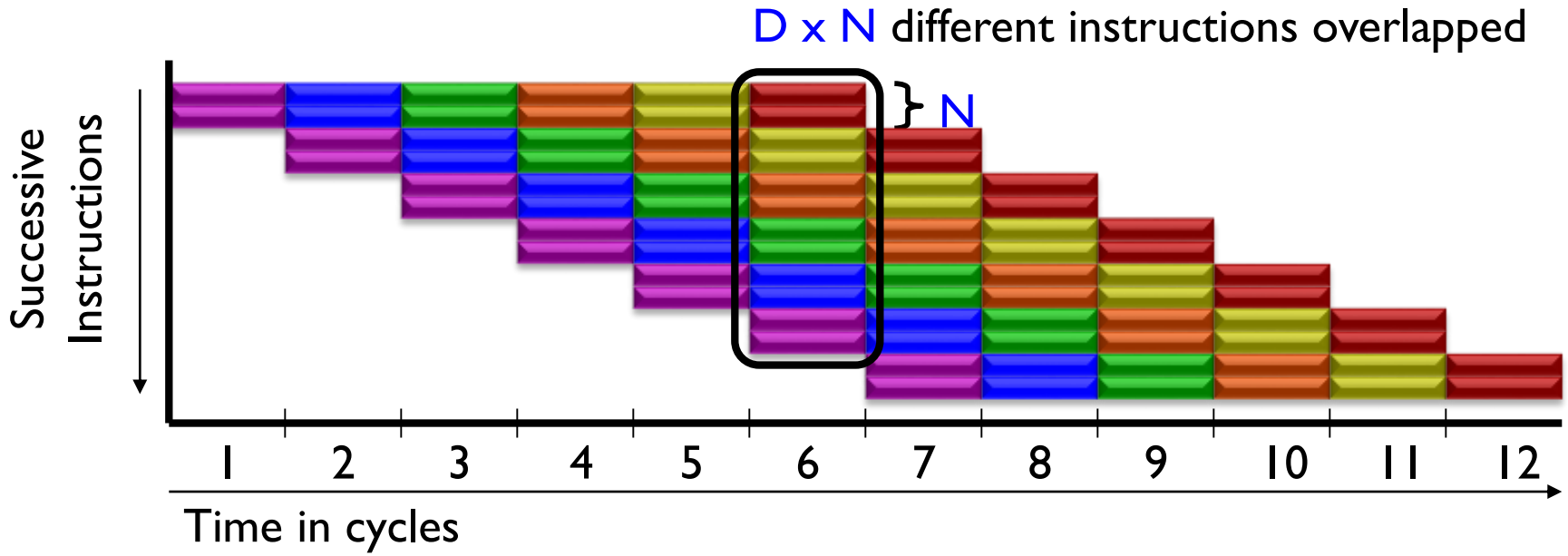
Instruction Level Parallelism (ILP)

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



Superscalar Processor

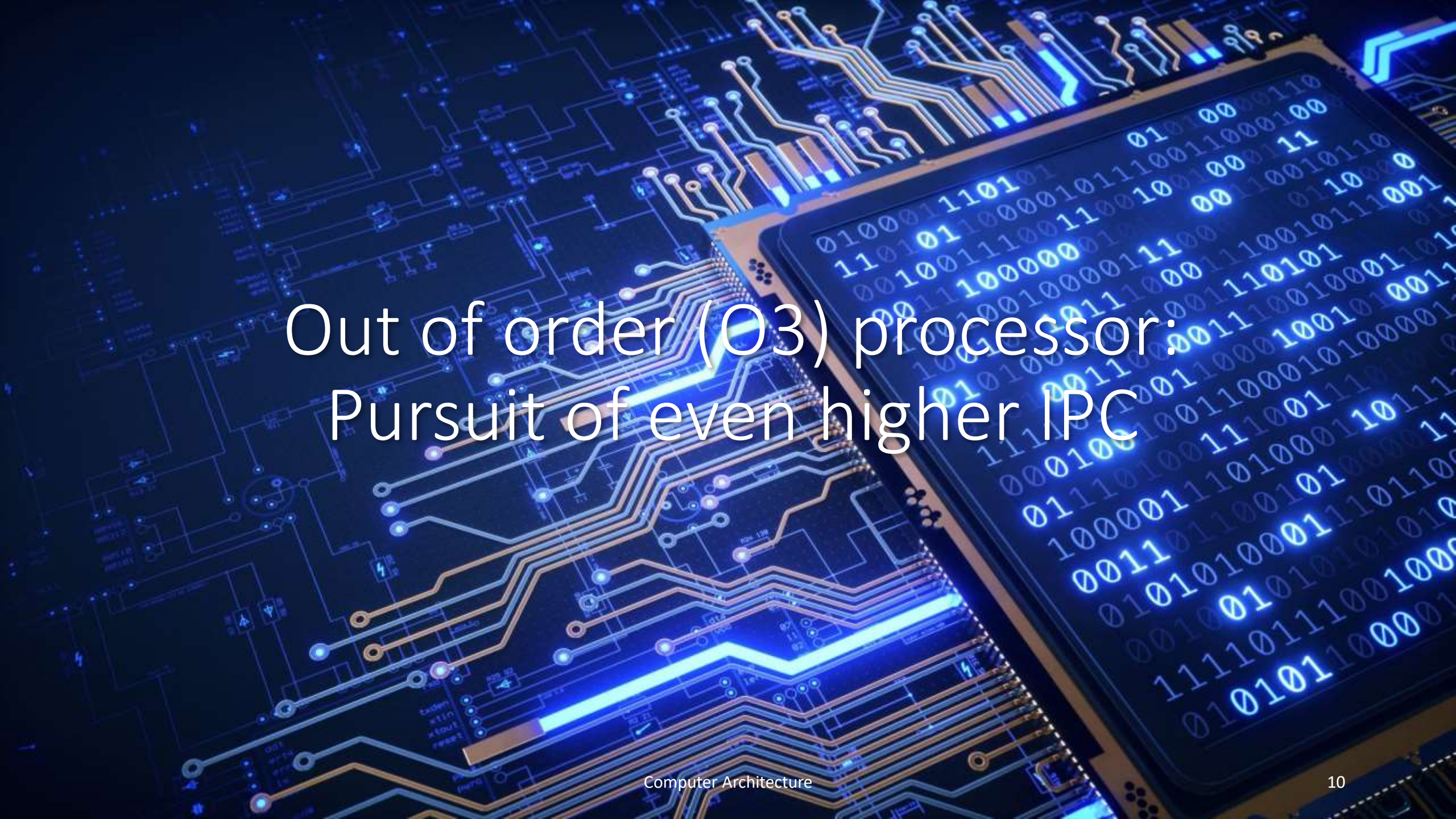
- Superscalar (pipelined) Execution
  - Instruction parallelism = D x N
  - Peak IPC = N per cycle



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



Out of order (O3) processor: Pursuit of even higher IPC

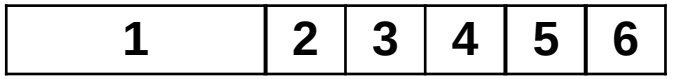
Out-of-order follows data-flow order

Example:

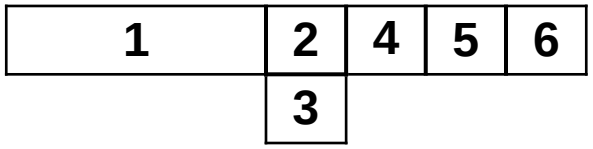
```
(1) r1 ← r4 / r7
(2) r8 ← r1 + r2
(3) r5 ← r5 + 1
(4) r6 ← r6 - r3
(5) r4 ← r5 + r6
(6) r7 ← r8 * r4
```

/* assume division takes 20 cycles */

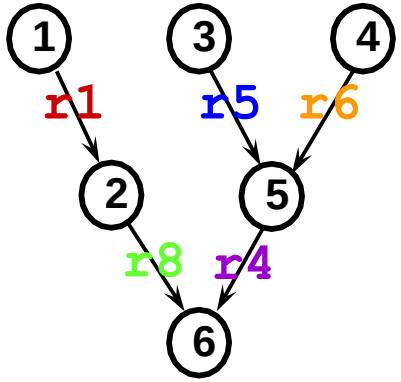
In-order execution



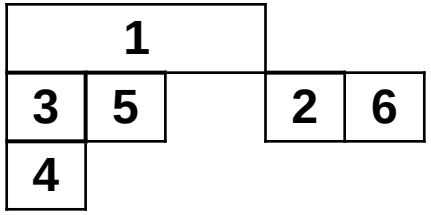
In-order (2-way superscalar)



Data Flow Graph



Out-of-order execution

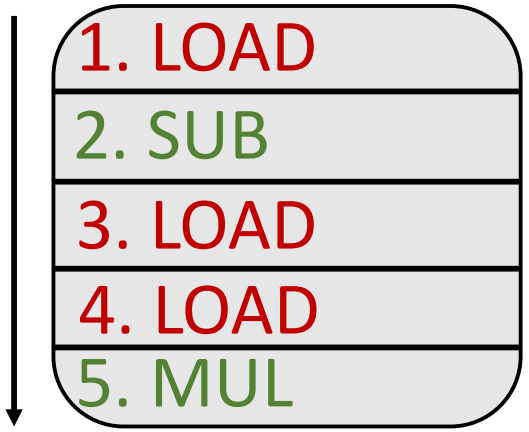


O3

Two or more instructions can execute in any order if they have no dependences (RAW, WAW, WAR)

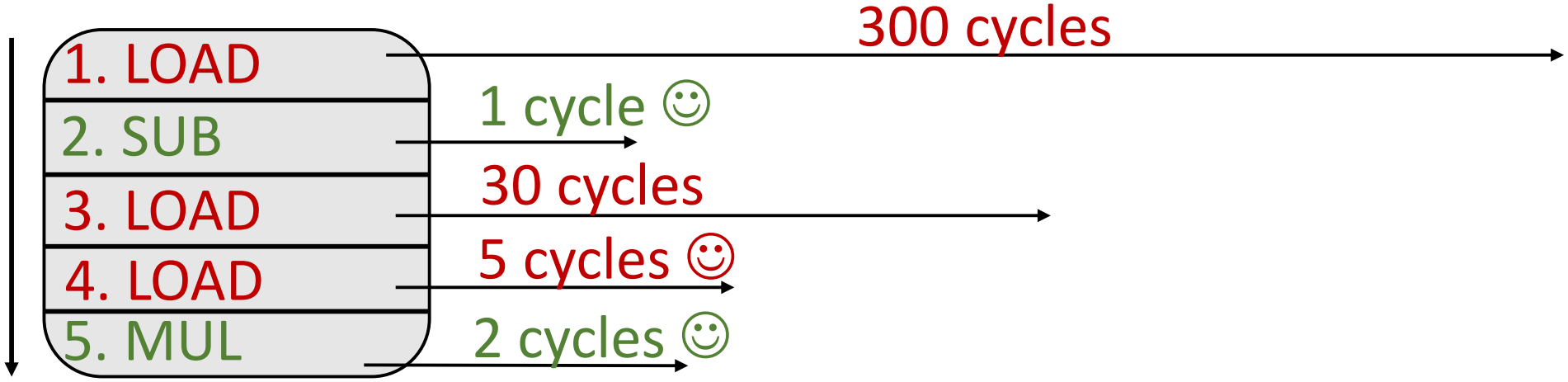
Completely orthogonal to superscalar/pipelining

O3 + Superscalar



In-order Instruction Fetch
(Multiple fetch in one cycle)

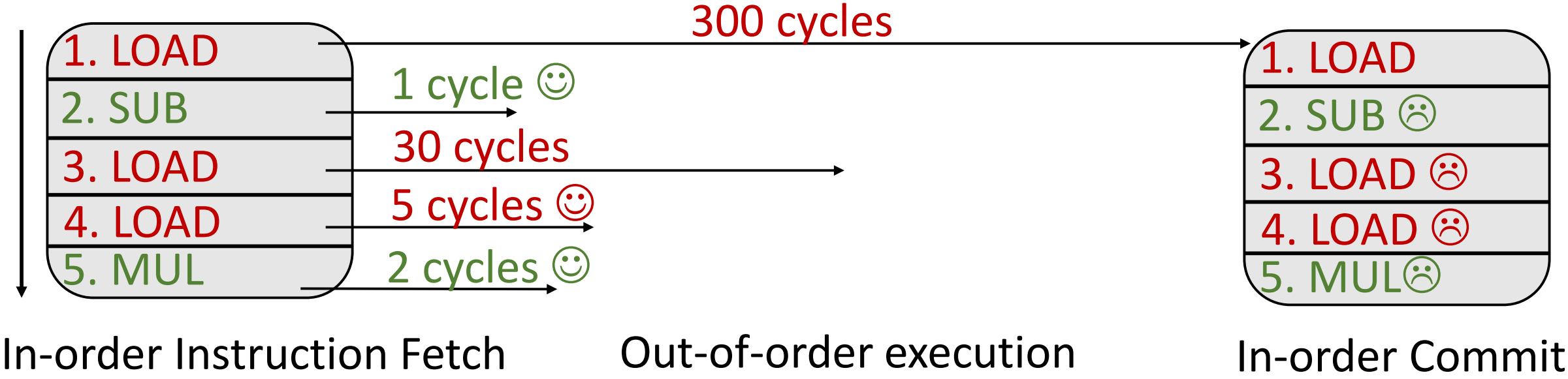
O3 + Superscalar



In-order Instruction Fetch
(Multiple fetch in one cycle)

Out-of-order execution

O3 + Superscalar



The Big Picture

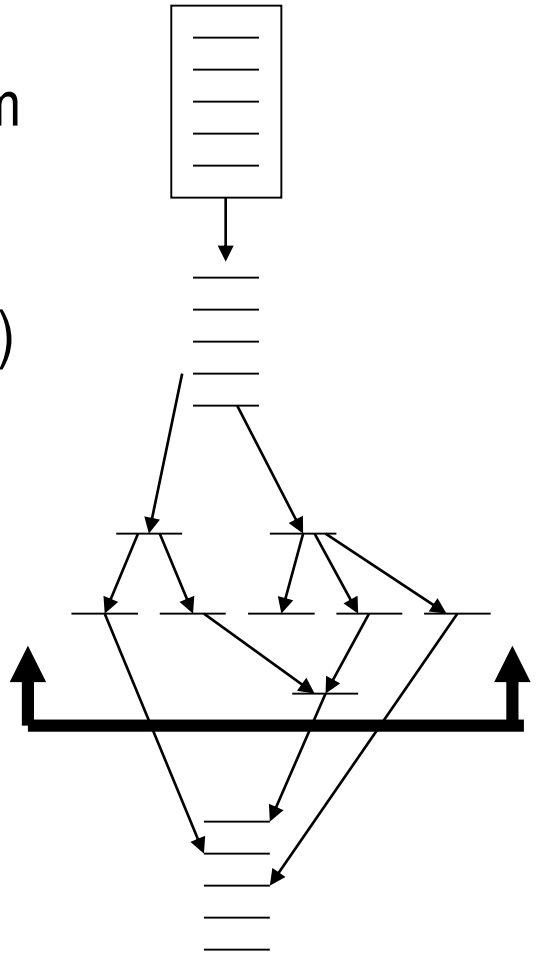
Program Form

Static program

dynamic inst.
Stream (trace)

execution
window

completed
instructions



Processing Phase

Dispatch/ dependences

inst. Issue

inst execution

inst. Reorder &
commit

The notion of Commit

After commit, the results of a committed instruction is visible to the programmer

and

the order at which instructions are fetched is also visible.

Why we need in-order commit?

Think about exceptions and precise exceptions

We should know till when we are done as per the programmer's view.

Quantifying Performance

Performance: Time (Iron Law)

Time/Program =

Instructions/program X cycles/instruction X Time/cycle

Source code	ISA	microarch.
Compiler	microarch.	technology
ISA		

Performance: Time (Iron Law)

Time/Program =

Instructions/program X cycles/instruction X Time/cycle

(∑ IC(i) X CPI (i)) X Time/cycle

Example

Program p = one billion instructions

Processor takes one cycle per instruction

Processor clock is 1GHz

CPU time = $10^{9 \text{ instructions}}$ X 1 cycle/instruction X 1 ns

= 1 second

Example

Program p = one billion instructions

Processor takes one cycle per instruction

Processor clock is 4 GHz

CPU time = $10^{9 \text{ instructions}}$ X 1 cycle/instruction X 1/4 ns

= 0.25 second (4X faster)

Example

Program p = one billion instructions

Processor processes 10 instructions in one cycle

Processor clock is 4 GHz

CPU time = $10^{9 \text{ instructions}}$ X 0.10 cycle/instruction X 1/4 ns

= 0.025 second (40X faster)

Example (Role of compiler/programmer)

Program p = one million instructions

Processor processes 1 instruction in one cycle

Processor clock is 4 GHz

CPU time = $10^{6 \text{ instructions}}$ X 1 cycle/instruction X 1/4 ns

= 0.00025 second (4000X faster)

A bit deeper

Program p has 10 billion instructions

* 2 billion branches (CPI of 4)
* 3 billion Loads (CPI of 2)
* 1 billion Stores (CPI of 3)
* Rest 4 billion, arithmetic instructions (CPI of 1)

Clock rate 4GHz, What is the execution time?

Which one ?

Processor IMTEL: CPI 2, Clock rate 2GHz

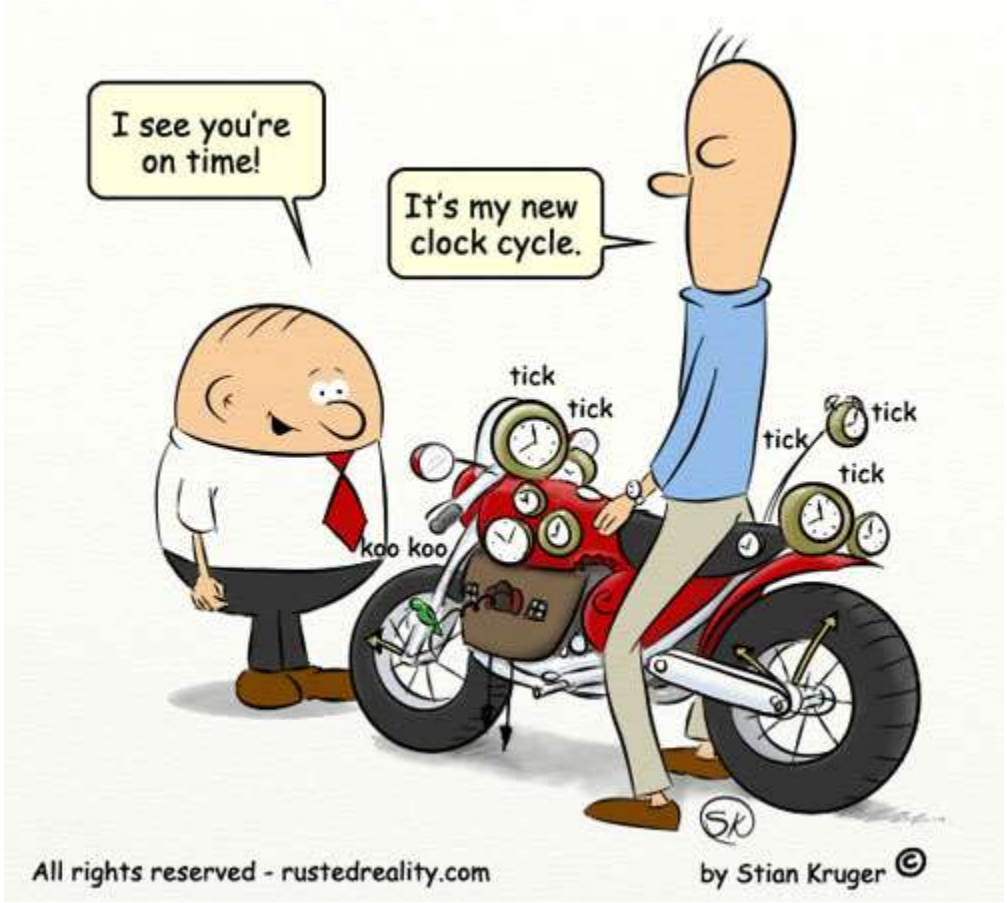
Processor AND: CPI 1, Clock rate 1GHz

Assume compiler/ISA/… are the same.

IMTEL: 2 X 0.5 ns = 1 ns per instruction

AND: 1 X 1ns = 1ns per instruction ☺

Empirical Evaluation



- Benchmarks
- Metrics
- Simulators
- Latency and bandwidth