



Computer Hardware Engineering (IS1200)

Computer Organization and Components (IS1500)

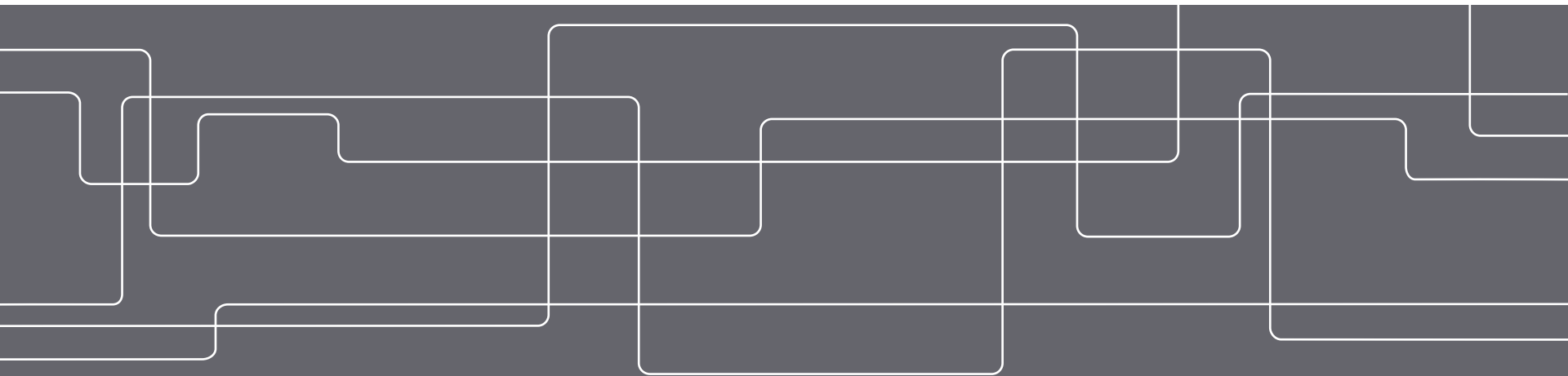
Spring 2021

Lecture 12: Parallelism, Concurrency, Speedup, and ILP

Artur Podobas

Researcher, KTH Royal Institute of Technology

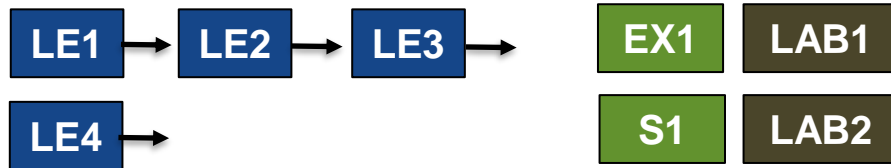
Slides by David Broman, KTH (Extensions by Artur Podobas)



Course Structure



Module 1: C and Assembly Programming



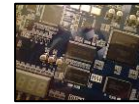
Module 2: I/O Systems



Module 3: Logic Design (IS1500 only)



**PROJ
START**



Module 4: Processor Design



Module 5: Memory Hierarchy



Module 6: Parallel Processors and Programs



Proj. Expo

LE14

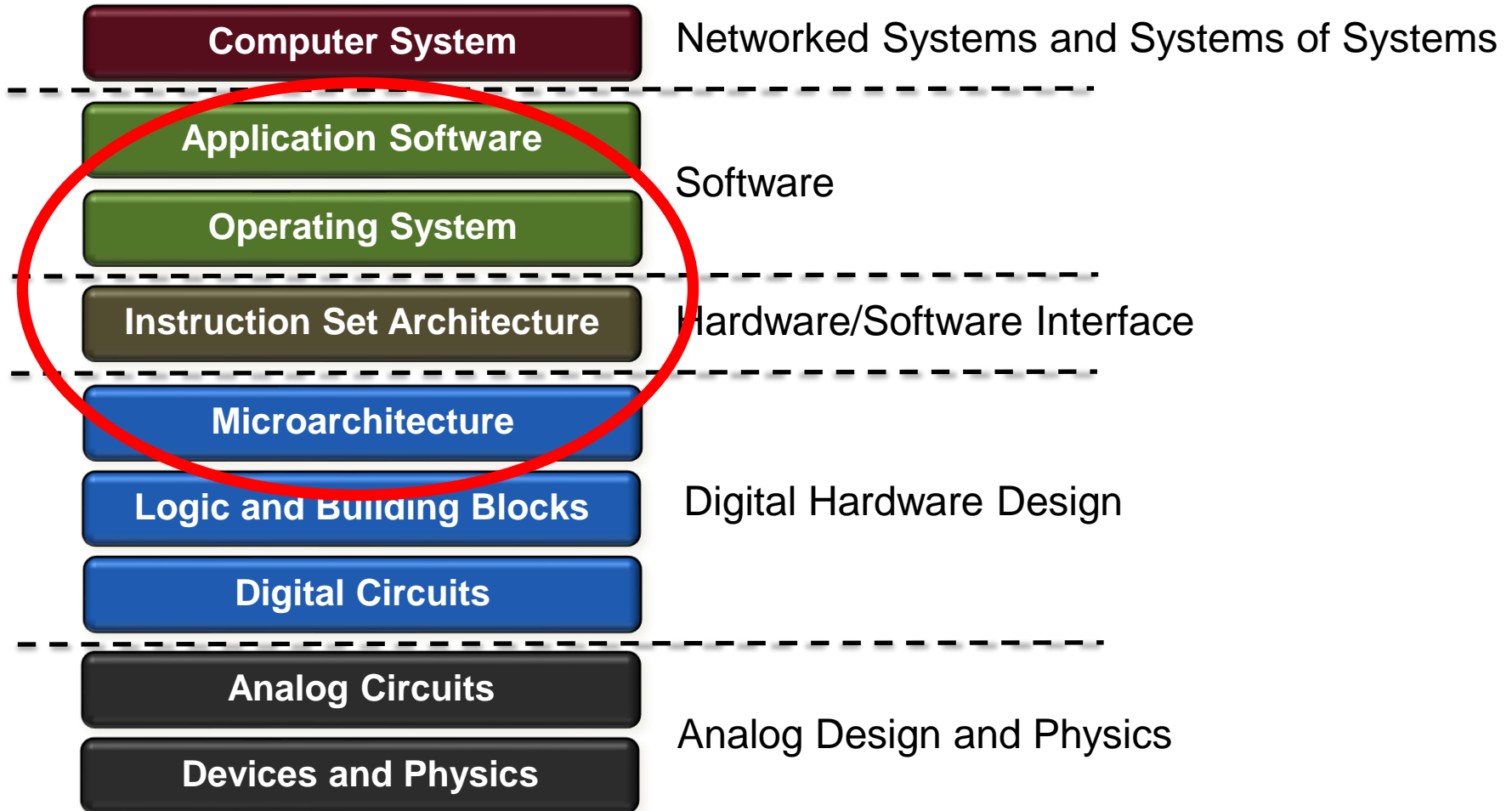
Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism

Abstractions in Computer Systems



Agenda

Part I

**Multiprocessors, Parallelism,
Concurrency, and Speedup**



Part II

Instruction-Level Parallelism



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism

Part I

Multiprocessors, Parallelism, Concurrency, and Speedup



Acknowledgement: The structure and several of the good examples are derived from the book “Computer Organization and Design” (2014) by David A. Patterson and John L. Hennessy



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism

How is this computer revolution possible? (Revisited)



Moore's law:

- Integrated circuit resources (transistors) double every 18-24 months.
- By Gordon E. Moore, Intel's co-founder, 1960s.
- Possible because of refined manufacturing processes. E.g., 4th generation Intel Core i7 processors uses 22nm manufacturing.
- Sometimes considered a *self-fulfilling prophecy*. Served as a goal for the semiconductor industry.

Dennard's scaling:

- As transistor densities grows (Moore's law), their power densities remain constant
- Robert H. Dennard, 1974
- Voltage (and thus current) scale down



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

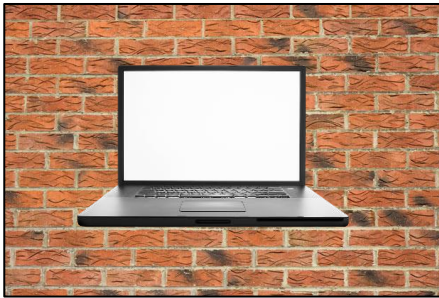
Part II

Instruction-Level
Parallelism

Have we reached the limit? (Revisited)

Why?

The Power Wall



<http://www.publicdomainpictures.net/view-image.php?image=1281&picture=tegelvagg>

**Increased clock rate
implies increased power**

**We cannot cool the system enough to
increase the clock rate anymore...**

**During the last decade, the clock rate has
increased dramatically.**

- 1989: 80486, 25MHz
- 1993: Pentium, 66Mhz
- 1997: Pentium Pro, 200MHz
- 2001: Pentium 4, 2.0 GHz
- 2004: Pentium 4, 3.6 GHz

2016: Intel Core i7, 3.1 GHz (Turbo 3.4Ghz)

2020: Fujitsu ARM A64FX, 2 GHz

“New” trend since 2006: Multicore

- Moore’s law still holds (*but not for long*)
- More processors on a chip: multicore
- **“New” challenge: parallel programming**



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

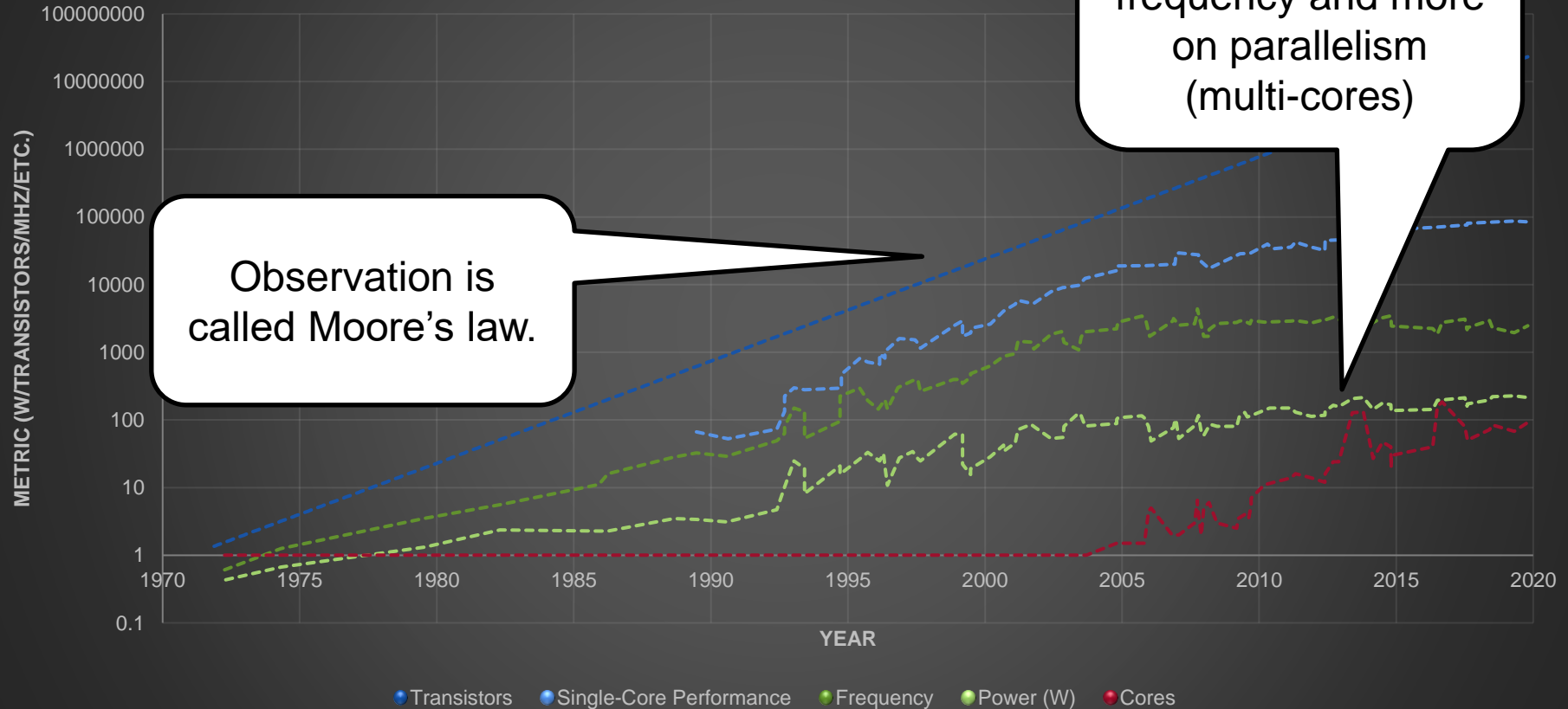
Part II

Instruction-Level
Parallelism

Microprocessor Trends

Plot redrawn using data by Karl Rupp
(<https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/>)

Microprocessor Trends



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

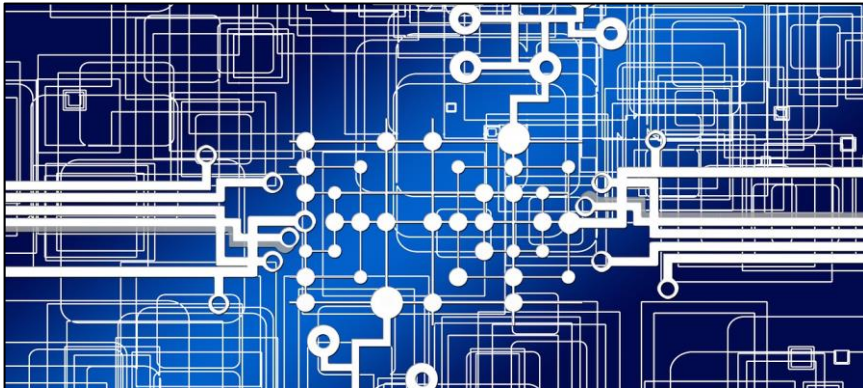


Part II

Instruction-Level
Parallelism

Moore's law: we might soon have reached the limit

There is a limit for how small transistors we can manufacture.



By the early 2020s:

“even with superaggressive efforts, we’ll get to the 2–3nanometre limit, where features are just 10 atoms across. Is that a device at all?”

According to Paolo Gargini,
(citation from Waldrop, 2016)

Part of the course literature (especially important for advanced part):
Mitchell Waldrop. **More than Moore**, *Nature*, Vol 530, 2016. (See the course website)



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism

What is a multiprocessor?

A **multiprocessor** is a computer system with two or more processors.



By contrast, a computer with one processor is called a **uniprocessor**.



by Eric Gaba, CC BY-SA 3.0. No modifications made.

Multicore microprocessors are multiprocessors where all processors (cores) are located on a single integrated circuit.



Photo by Robert Harker

A **cluster** is a set of computers that are connected over a local area network (LAN). May be viewed as one large multiprocessor. Communication between nodes (computers) often a bottleneck.



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Different Kinds of Computer Systems (Revisited)



**Embedded
Real-Time Systems**



Photo by Kyro

**Personal Computers and
Personal Mobile Devices**



Photo by Robert Harker

**Warehouse
Scale Computers /
Supercomputers**

Dependability

Energy

Performance



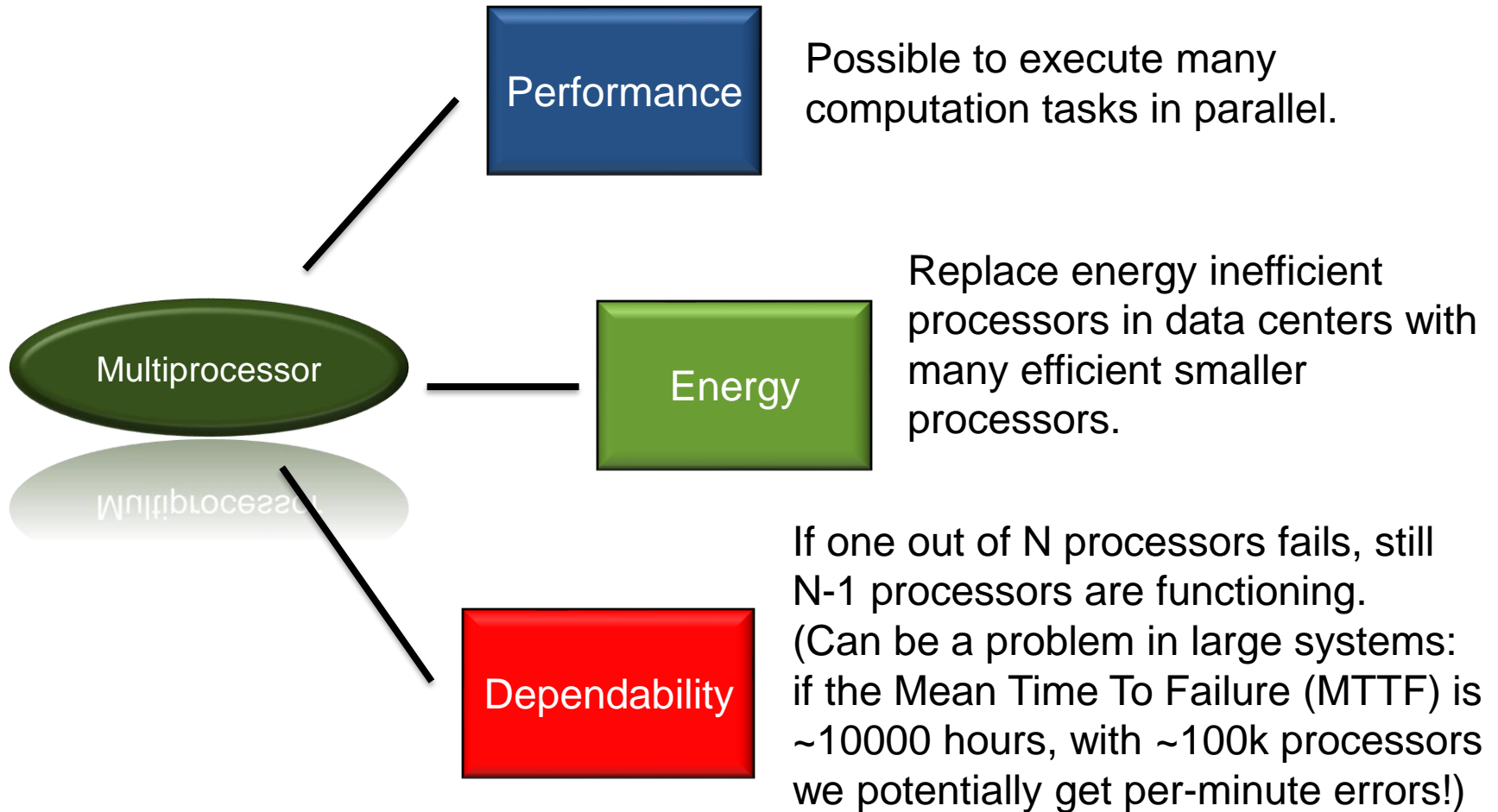
Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism

Why multiprocessors?



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Parallelism and Concurrency – what is the difference?

Concurrency is about **handling** many things at the same time.
Concurrency may be viewed from the **software** viewpoint.

Parallelism is about **doing (executing)** many things at the same time. Parallelism may be viewed from the **hardware** viewpoint.

		Software	
		Sequential	Concurrent
Hardware	Serial	Example: matrix multiplication on a uniprocessor.	Example: A Linux OS running on a uniprocessor .
	Parallel	Example: matrix multiplication on a multicore processor.	Example: A Linux OS running on a multicore processor.

Note: As always, everybody does not agree on the definitions of concurrency and parallelism. The matrix is from H&P 2014 and the informal definitions above are similar to what was said in a talk by Rob Pike.



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

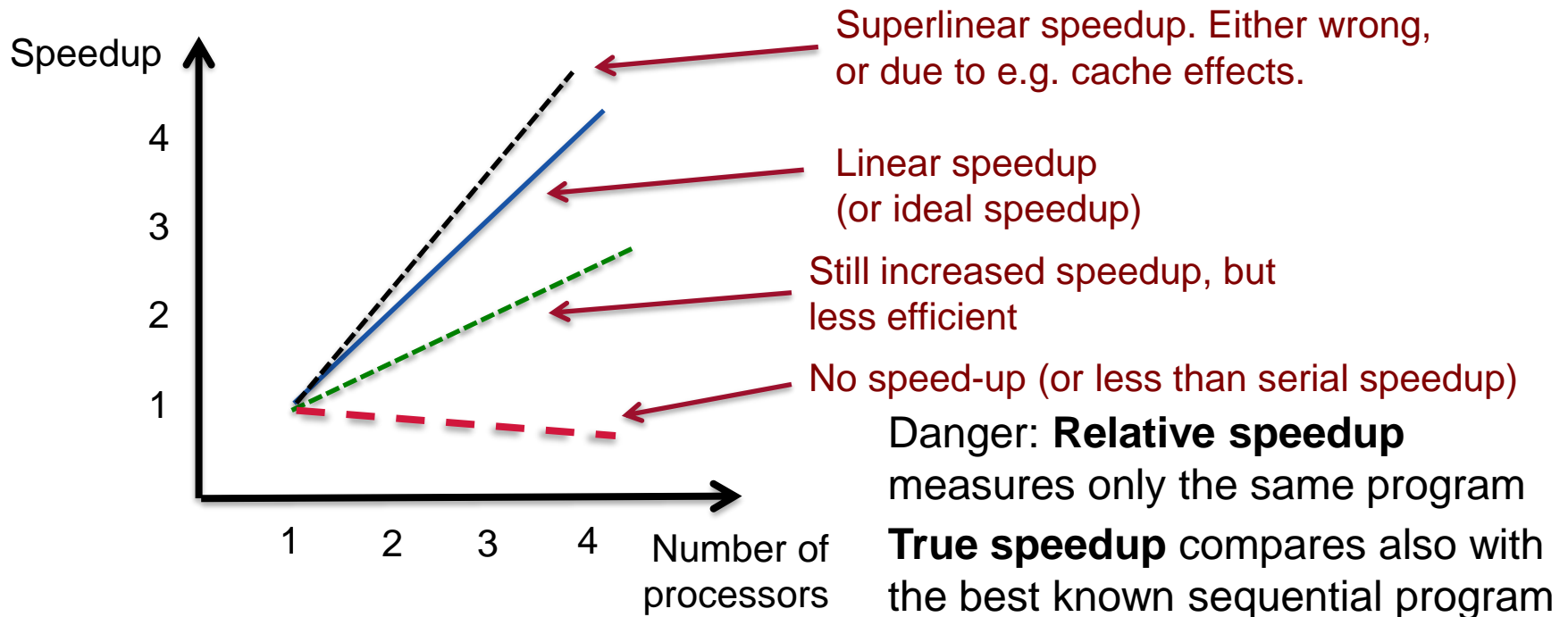
Speedup

How much can we improve the performance using parallelization?

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}}$$

Execution time of one program **before** improvement

Execution time **after** improvement



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Amdahl's Law (1/5)

Can we achieve linear speedup?

Divide execution time before improvement into two parts.

Time affected by the improvement of parallelization

Time unaffected of improvement (sequential part)

$$T = T_{\text{affected}} + T_{\text{unaffected}}$$

Execution time after improvement

$$T_{\text{after}} = \frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}$$

Amount of improvement (N times improvement)

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}} = \frac{T_{\text{before}}}{\frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}}$$

This is sometimes referred to as **Amdahl's law**



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Amdahl's Law (2/5)

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}} = \frac{T_{\text{before}}}{\frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}}$$

Exercise: Assume a program consists of an image analysis task, sequentially followed by a statistical computation task. Only the image analysis task can be parallelized. How much do we need to improve the image analysis task to be able to achieve 4 times speedup?

Assume that the program takes 80ms in total and that the image analysis task takes 60ms out of this time.

Solution:

$$4 = 80 / (60 / N + 80 - 60)$$

$$60/N + 20 = 20$$

$$60/N = 0$$

It is impossible to achieve this speedup!



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Amdahl's Law (3/5)

$$\text{Speedup} = \frac{T_{\text{before}}}{T_{\text{after}}} = \frac{T_{\text{before}}}{\frac{T_{\text{affected}}}{N} + T_{\text{unaffected}}}$$



Assume that we perform 10 scalar integer additions, followed by one matrix addition, where matrices are 10x10. Assume additions take the same amount of time and that we can only parallelize the matrix addition.

Exercise A: What is the speedup with 10 processors?

Exercise B: What is the speedup with 40 processors?

Exercise C: What is the maximal speedup?

Solution A:

$$(10+10*10) / (10*10/10 + 10) = 5.5$$

Solution B:

$$(10+10*10) / (10*10/40 + 10) = 8.8$$

Solution C:

$$(10+10*10) / (10*10/N + 10) = 11 \text{ when } N \rightarrow \text{infinity}$$

Alternative solution for C

$$(10+10*10) / (10*10/100 + 10) = 10$$

if we assume that one add instruction cannot be parallelized



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Amdahl's Law (4/5)

Example continued. What if we change the size of the problem (make the matrices larger)?

		Number of processors	
		10	40
Size of matrices	10x10	Speedup 5.5	Speedup 8.8
	20x20	Speedup 8.2	Speedup 20.5

But was not the maximal speedup 11 when $N \rightarrow \infty$?

Strong scaling = measuring speedup while keeping the problem size fixed.

Weak scaling = measuring speedup when the problem size grows proportionally to the increased number of processors (*Gustafson's law*).



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Amdahl's Law (5/5)

TOP500 event

- ☐ Announced two times a year (at ISC and SC conferences)
- ☐ Progress of supercomputers
- ☐ Performance based on LINPACK (solve dense system of linear equations)

Current #1: Fugaku, Japan

- ☐ ~7 million cores
- ☐ Require a lot of parallelism to use (Amdahl's law)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646

Source: Top500.org



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism

Main Classes of Parallelisms

DLP

Data-Level Parallelism (DLP)

Many data items can be processed at the same time.



Example – Sheep shearing

Assume that sheep are data items and the task for the farmer is to do sheep shearing (remove the wool). Data-level parallelism would be the same as using several farm hands to do the shearing.

TLP

Task-Level Parallelism (TLP)

Different tasks of work that can work in independently and in parallel



Example – Many tasks at the farm

Assume that there are many different things that can be done on the farm (fix the barn, sheep shearing, feed the pigs etc.) Task-level parallelism would be to let the farm hands do the different tasks in parallel.



Part I

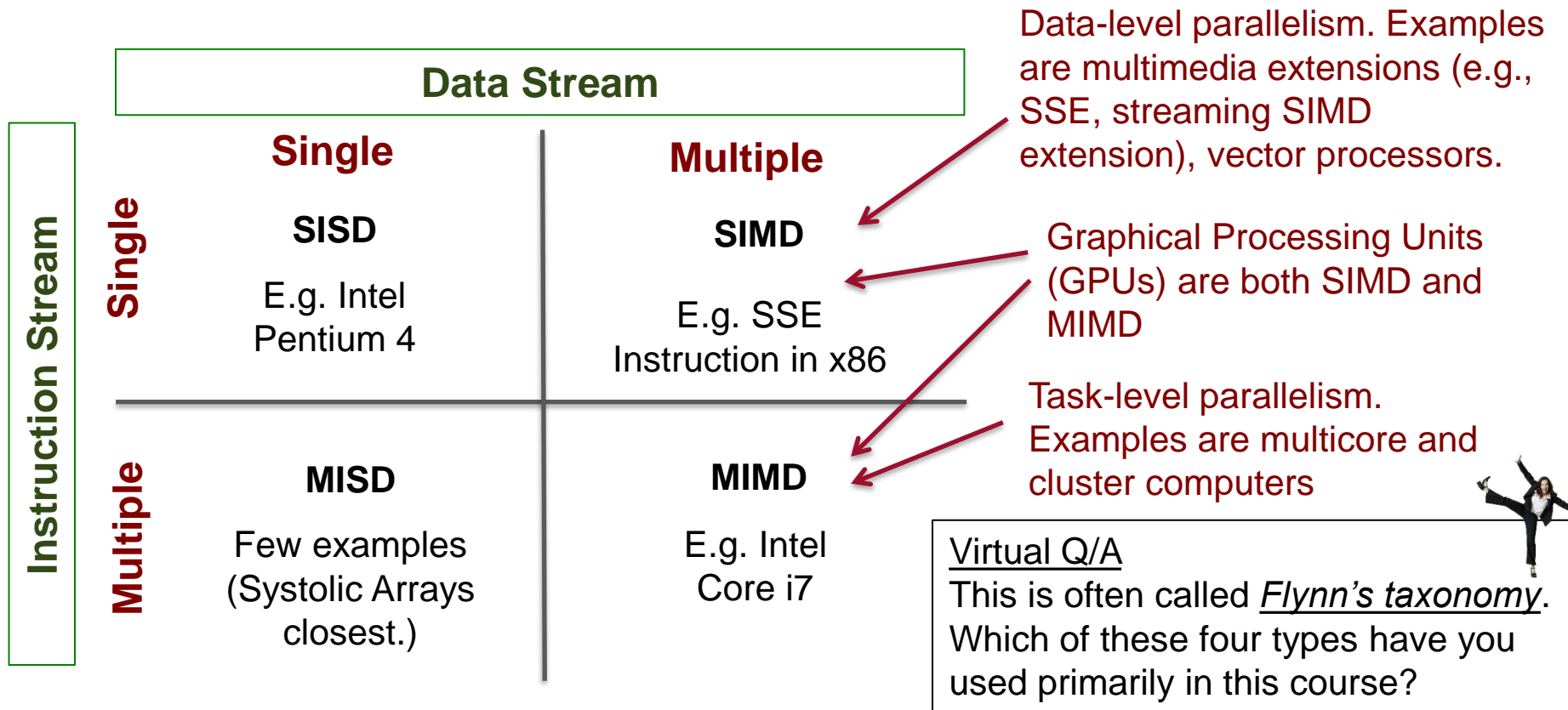
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism

SISD, SIMD, and MIMD

An old (from the 1960s) but still very useful classification of processors uses the notion of instruction and data streams.



Part I

Multiprocessors, Parallelism, Concurrency, and Speedup

Part II

Instruction-Level Parallelism


Part II

Instruction-Level Parallelism



Acknowledgement: The structure and several of the good examples are derived from the book “Computer Organization and Design” (2014) by David A. Patterson and John L. Hennessy

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup



Part II
Instruction-Level
Parallelism

What is Instruction-Level Parallelism?

Instruction-Level Parallelism (ILP) may increase performance without involvement of the programmer. It may be implemented in a SISD, SIMD, and MIMD computer.

Two main approaches:



1. Deep pipelines with more pipeline stages

If the length of all pipeline stages are balanced, we may increase performance by increasing the clock speed.



2. Multiple issue

A technique where multiple instructions are issued in each in cycle.

ILP may decrease the CPI to lower than 1, or using the inverse metric *instructions per clock cycle (IPC)* increase it above 1.



Part I

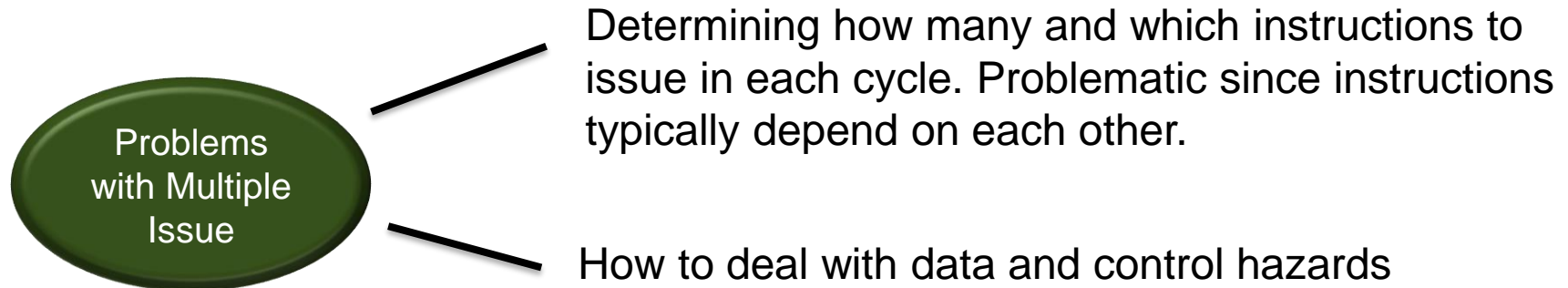
Multiprocessors, Parallelism,
Concurrency, and Speedup



Part II

Instruction-Level
Parallelism

Multiple Issue Approaches



The two main approaches of multiple issue are

1. Static Multiple Issue

Decisions on when and which instructions to issue at each clock cycle is determined by the compiler.

2. Dynamic Multiple Issue

Many of the decisions of issuing instructions are made by the processor, dynamically, during execution.

Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup



Part II

Instruction-Level
Parallelism

Static Multiple Issue (1/3)

VLIW

Very Long Instruction Word (VLIW)

processors issue several instructions in each cycle using **issue packages**.

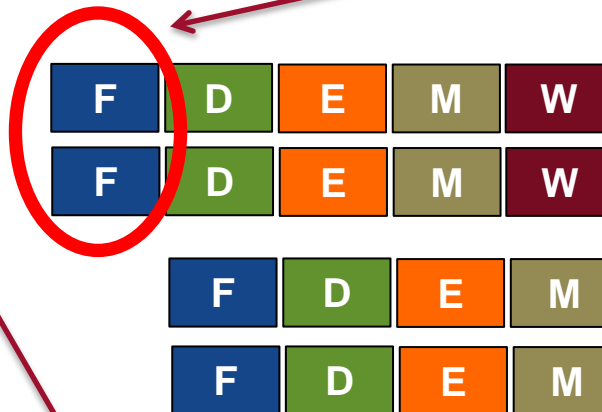
An **issue package** may be seen as one large instruction with multiple operations.

add \$s0, \$s1, \$s2

add \$t0, \$t0, \$0

and \$t2, \$t1, \$s0

lw \$t0, 0(\$s0)



Each issue package has two **issue slots**.

The compiler may insert **no-ops** to avoid hazards.

How is VLIW affecting the hardware implementation?

Part I

Multiprocessors, Parallelism, Concurrency, and Speedup



Part II

Instruction-Level Parallelism

Static Multiple Issue (2/3)

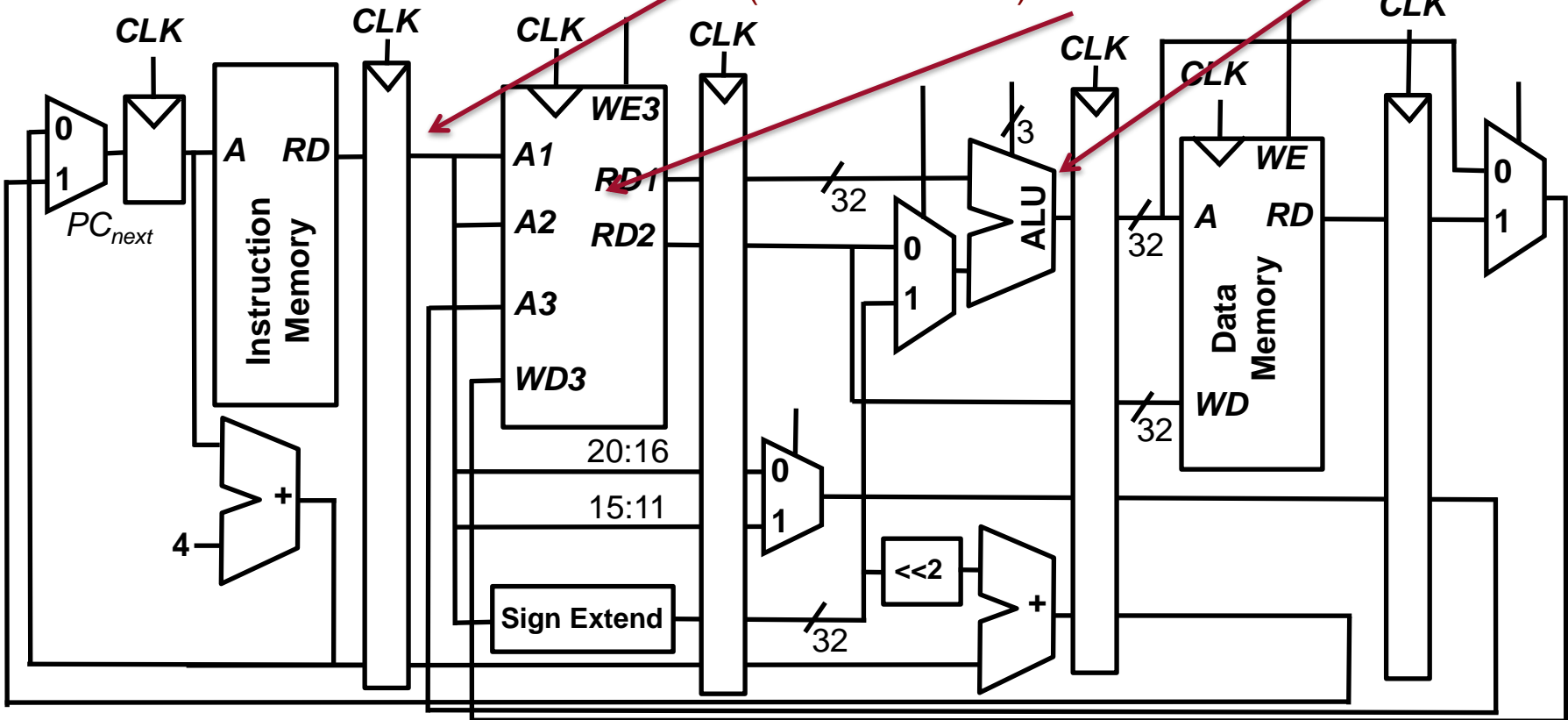
Changing the hardware

To issue two instructions in each cycle, we need the following (not shown in picture):

Fetch and decode 64-bit (two instructions)

Double the number of ports for the register file

Add another ALU



Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup



Part II

Instruction-Level
Parallelism

Static Multiple Issue (3/3)

VLIW

Exercise: Assume we have a VLIW processor that can issue two instructions in the same clock cycle. For the following code, schedule the instructions into issue slots and compute IPC. Assume that no hazards occurs.

```
addi $s1, $0, 10
L1: lw  $t0, 0($s2)
    ori $t1, $t0, 7
    sw  $t1, 0($s2)
    addi $s2, $s2, 4
    addi $s1, $s1, -1
    bne $s1, $0, L1
```

Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup



Part II

Instruction-Level
Parallelism

Step 1) How many instructions?

Answer: $1 + (10 \cdot 6) = 61$ instructions

Step 2) Schedule the instructions onto the VLIW.

Step 3) Calculate IPC.

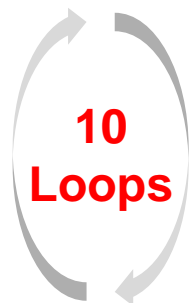
Each loop takes $(5-1=)$ 4 cycles.

Total cycles: $1 + (10 \cdot 4) = 41$ cycles.

61 instructions / 41 cycles \approx 1.487 IPC

```

        addi $s1, $0, 10
L1:    lw    $t0, 0($s2)
        ori   $t1, $t0, 7
        sw    $t1, 0($s2)
        addi $s2, $s2, 4
        addi $s1, $s1, -1
        bne   $s1, $0, L1
    
```



	Slot 1	Slot 2	Cycle
	addi \$s1, \$0, 10		1
L1	lw \$t0, 0(\$s2)		2
	addi \$s2, \$s2, 4	addi \$s1, \$s1, -1	3
	ori \$t1, \$t0, 7		4
	bne \$s1, \$0, L1	sw \$t1, -4(\$s2)	5

Blank can also be used to mean NOP.

Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup



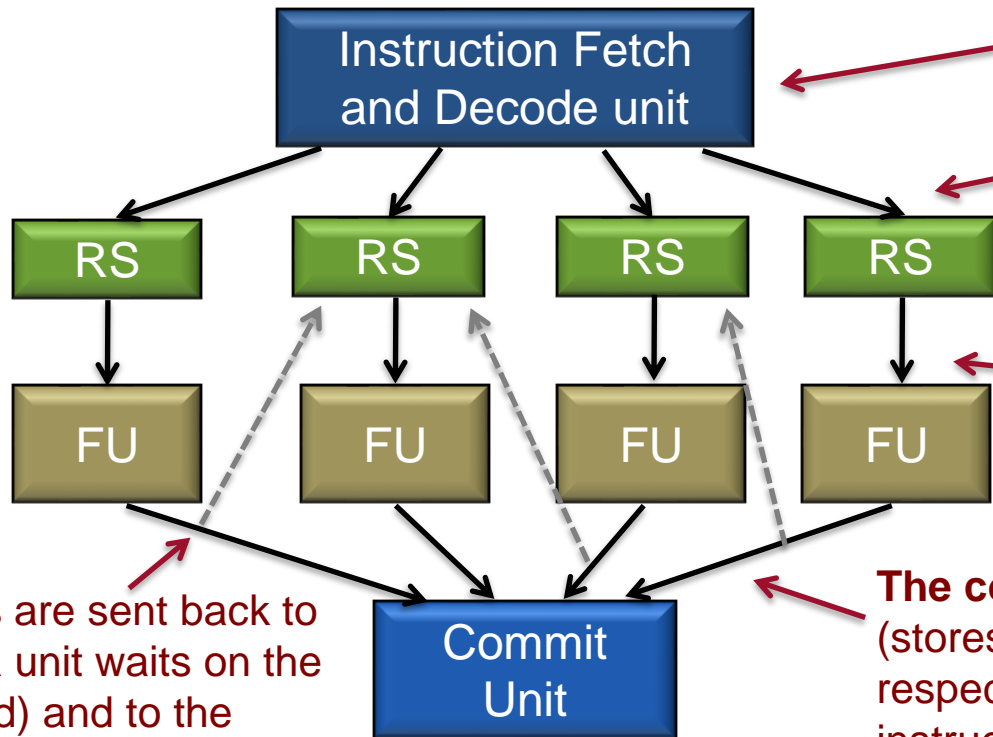
Part II

Instruction-Level
Parallelism

Dynamic Multiple Issue Processors (1/2)

Superscalar Processors

In many modern processors (e.g., Intel Core i7), instruction issuing is performed dynamically by the processor while executing the program. Such processor is called **superscalar**.



The first unit fetches and decodes several instruction **in-order**

Reservation Stations (RS) buffer operands to the FU before execution. Data dependencies are handled dynamically.

Functional Units (FU) execute the instruction. Examples are integer units, floating point units, and load/store units.

The commit unit commits instructions (stores in registers) conservatively, respecting the observable order of instructions. Called **in-order commit**.

Results are sent back to RS (if a unit waits on the operand) and to the commit unit.

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

Dynamic Multiple Issue Processors (2/2)

Out-of-Order Execution, RAW, WAR

If the superscalar processor can reorder the instruction execution order, it is an **out-of-order execution** processor.

```
lw    $t0, 0($s2)
addi  $t1, $t0, 7
```

Example of a **Read After Write (RAW)** dependency (dependency on \$t0). The superscalar pipeline must make sure that the data is available before read.

```
sub    $t0, $t1, $s0
addi  $t1, $s0, 10
```

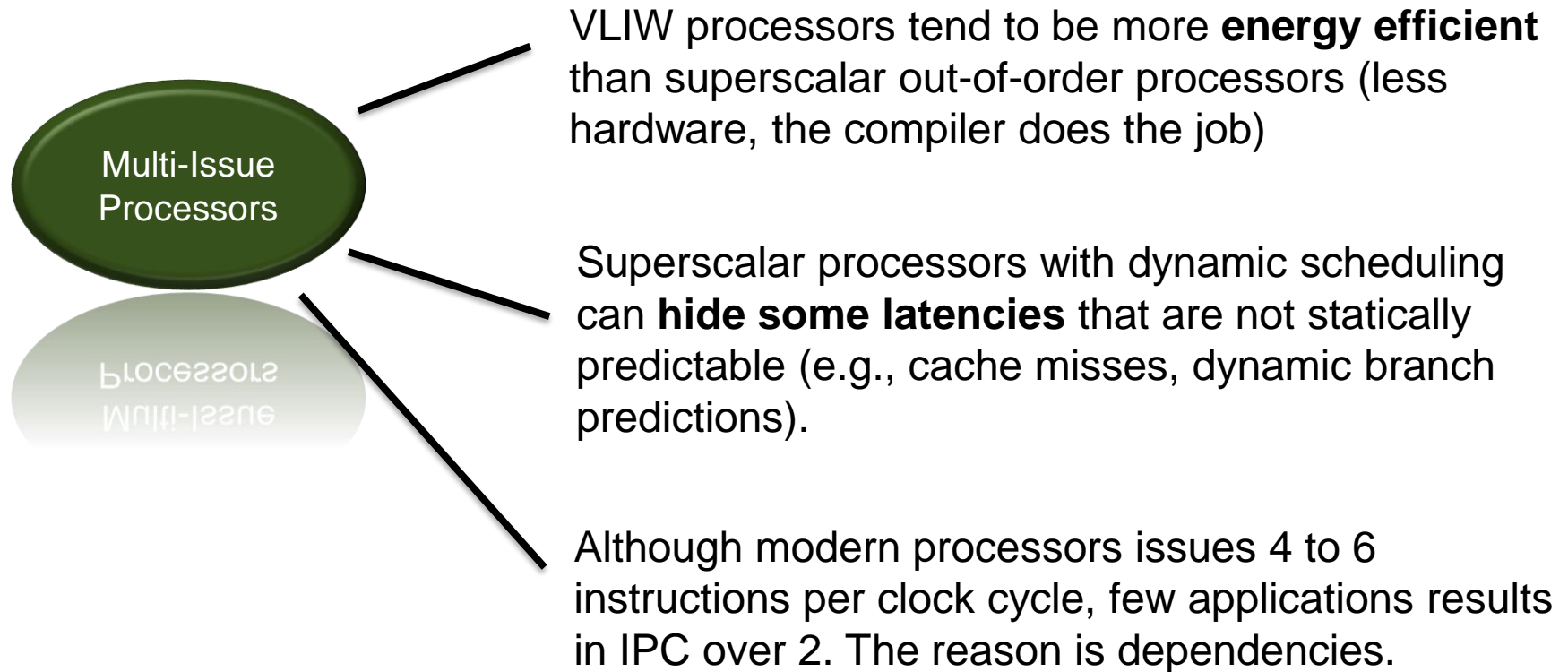
Example of a **Write After Read (WAR)** dependency (dependency on \$t1). If the order is flipped due to out-of-order execution, we have a hazard.

```
addi  $r1, $s0, 10
sub    $t0, $t1, $s0
```

WAR dependencies can be resolved using **register renaming**, where the processor writes to a nonarchitectural renaming register (here in the pseudo asm code called \$r1, not accessible to the programmer)

Note that out-of-order processor is not rewriting the code, but keeps track of the renamed registers during execution.

Some observations on Multi-Issue Processors





Intel Microprocessors, some examples

Processor	Year	Clock Rate	Pipeline Stages	Issue Width	Cores	Power
Intel 486	1989	25 MHz	5	1	1	5 W
Intel Pentium	1993	66 MHz	5	2	1	10W
Intel Pentium Pro	1997	200 MHz	10	3	1	29 W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	1	75W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	1	103W
Intel Core	2006	2930 MHz	14	4	2	75W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	1	87W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	8	77W

Clock rate increase stopped (the power wall) around 2006

Pipeline stages first increased and then decreased, but the number of cores increased after 2006.

The power consumption peaked with Pentium 4

Source: Patterson and Hennessey, 2014, page 344.

Part I

Multiprocessors, Parallelism, Concurrency, and Speedup



Part II

Instruction-Level Parallelism

Soon time for coffee.

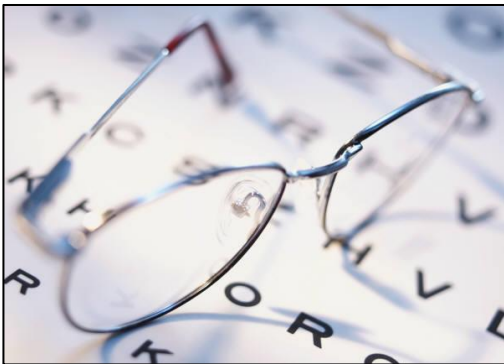


Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism

Reading Guidelines

Module 6: Parallel Processors and Programs



Lecture 12: Parallelism, Concurrency, Speedup, and ILP

- H&H Chapter 1.8, 3.6, 7.8.3-7.8.5
- P&H5 Chapters 1.7-1.8, 1.10, 4.10, 6.1-6.2
or P&H4 Chapters 1.5-1.6, 1.8, 4.10, 7.1-7.2

Lecture 13: SIMD, MIMD, and Parallel Programming

- H&H Chapter 7.8.6-7.8.9
- P&H5 Chapters 2.11, 3.6, 5.10, 6.3-6.7
or P&H4 Chapters 2.11, 3.6, 5.8, 7.3-7.7

Reading Guidelines

See the course webpage
for more information.

Part I

Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II

Instruction-Level
Parallelism



Summary

Some key take away points:

- **Amdahl's law** can be used to estimate maximal speedup when introducing parallelism in parts of a program.
- **Instruction-Level Parallelism (ILP)** has been very important for performance improvements over the years, but improvements have not been as significant lately.



Thanks for listening!

Part I
Multiprocessors, Parallelism,
Concurrency, and Speedup

Part II
Instruction-Level
Parallelism