Course Review

Section I

Xinfei Guo
xinfei.guo@sjtu.edu.cn

July 22nd, 2024

Grading Policy

Participation & Etiquette*	10%
Labs	30%
Final Project & Report**	35%
Final Exam	25%
Total	**100%**

- *Participation is highly recommended as there will be NO recordings, it includes lecture (3%) and lab participation (3%). The rest includes piazza questions and answers (2%), lecture/lab/OH involvement (2%), etc.
- **It also includes the literature search and review part.
- Creative ideas in projects will get bonus.
- Final letter grades might be curved, but are not guaranteed.

Final Exam Format

- Scope: Everything we have covered in the course (T1 – T8)
- Covers **All lectures**.
- Time: 2024/08/06, TUE., 18:20-20:00 (Beijing) Location **TBD**

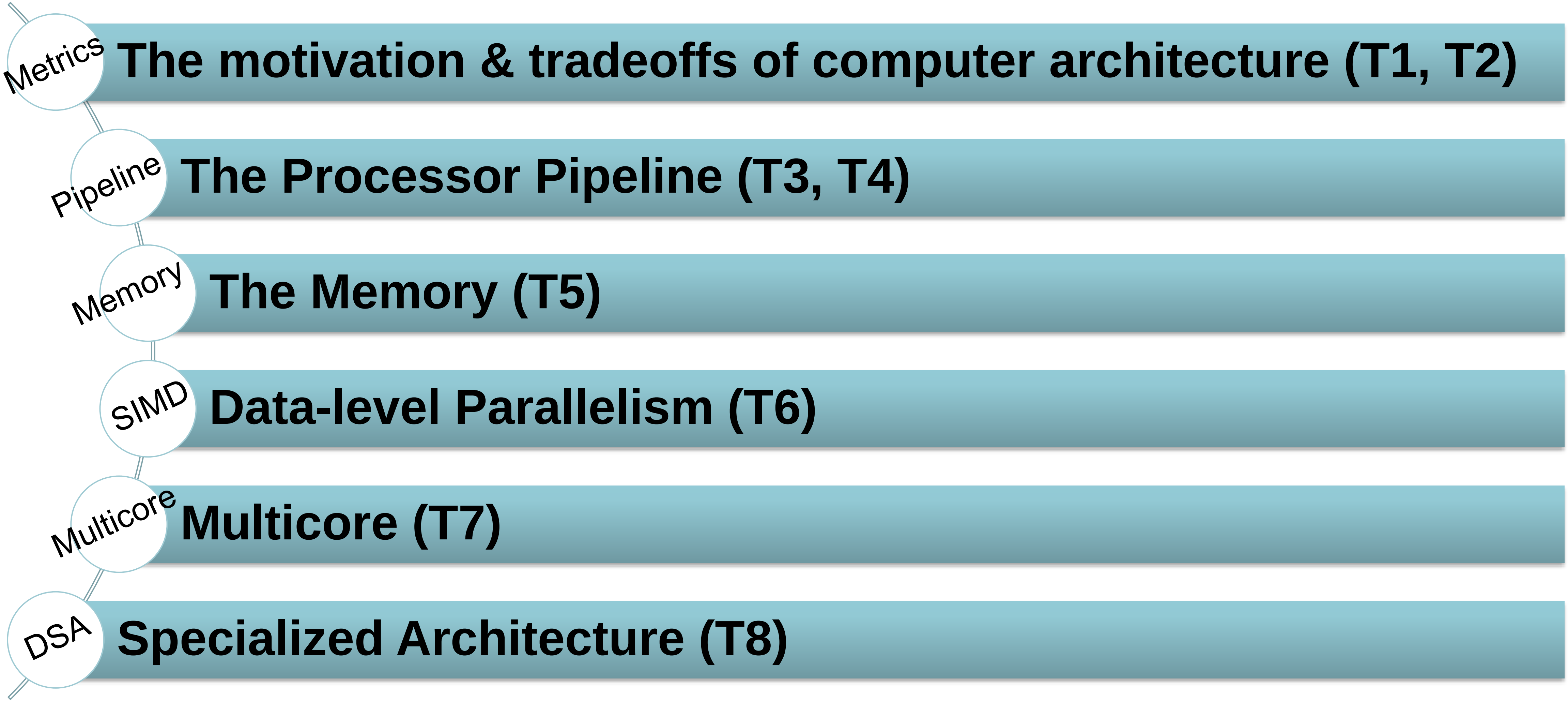
Only in-person format is available.

- Types of questions
  - Freebies: remember the materials
  - Probing: understand the materials
  - Applied: apply the materials in original interpretation
- **Close book, close notes, no cell phone, calculator allowed**
- Supplementary materials will be provided as appendix if needed
- Sample questions will be provided and released later this week, they might be lectured depending on the time left for this course.

What to expect in final exam

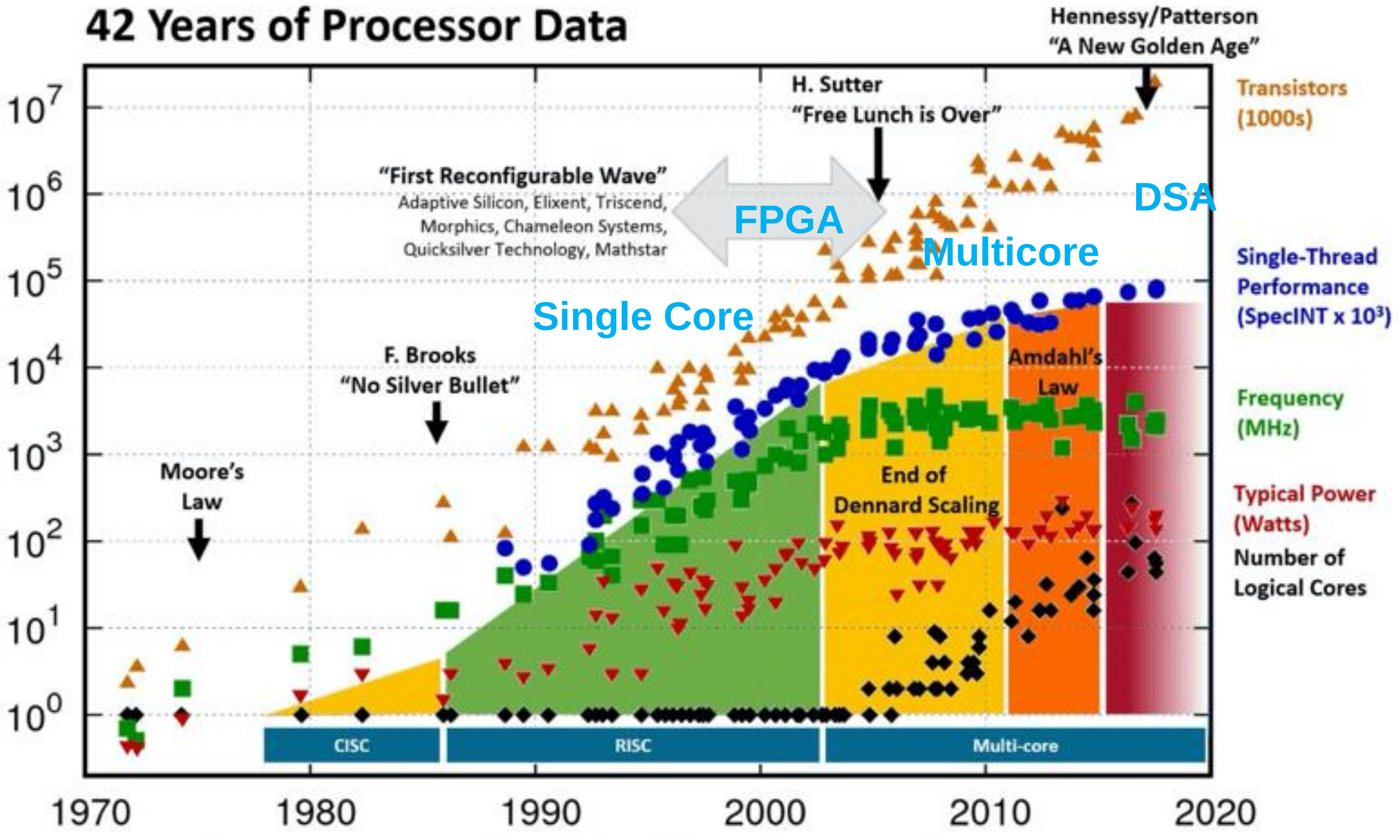
- Mixture of selection, short answer and long answer questions
- Will have questions with no fixed answers (subjective opinions)
- Questions with light calculation (equations will be provided for certain problems)
- Recommended strategy
  - Be prepared and start reviewing the course materials earlier
  - Learning in a systematic way
  - Focus on **understanding** instead of memorizing
  - Go through **examples** on slides and the book

What we have learned in this course



- Metrics: The motivation & tradeoffs of computer architecture (T1, T2)
- Pipeline: The Processor Pipeline (T3, T4)
- Memory: The Memory (T5)
- SIMD: Data-level Parallelism (T6)
- Multicore: Multicore (T7)
- DSA: Specialized Architecture (T8)

What we have learned in this course



Hennessy and Patterson, Turing Lecture 2018, overlaid over "42 Years of Processors Data"
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/;"First Wave" added by Les Wilson, Frank Schirrmeister
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Recap: Course Objectives

- To teach students fundamental design tradeoffs of the modern computer architectures; ✓ (T1, T2)
- To teach advanced computer architecture concepts such as Multiprocessors, Superscalar, GPUs and advanced caches; ✓ (part of T5, T6, T7)
- To give students an exposure to state-of-the-art computer architecture simulators through hands-on assignments and/or a project that will help understand the tradeoffs; ✓ (Lab 4, final project)
- To offer student hands-on experiences of implementing or optimizing major portions of a substantial processor with either Verilog or High-level design approaches; ✓ (Lab 1 – 3, final project)
- To introduce students the most recent computer architecture advances such as Domain specific architectures (DSA) and AI accelerators; ✓ (T8)
- To provide experiences of executing a project as a team from concept to finish effectively and professionally, and to offer opportunities to present ideas to the public. ✓ (final project)

Recap: Course Outcomes

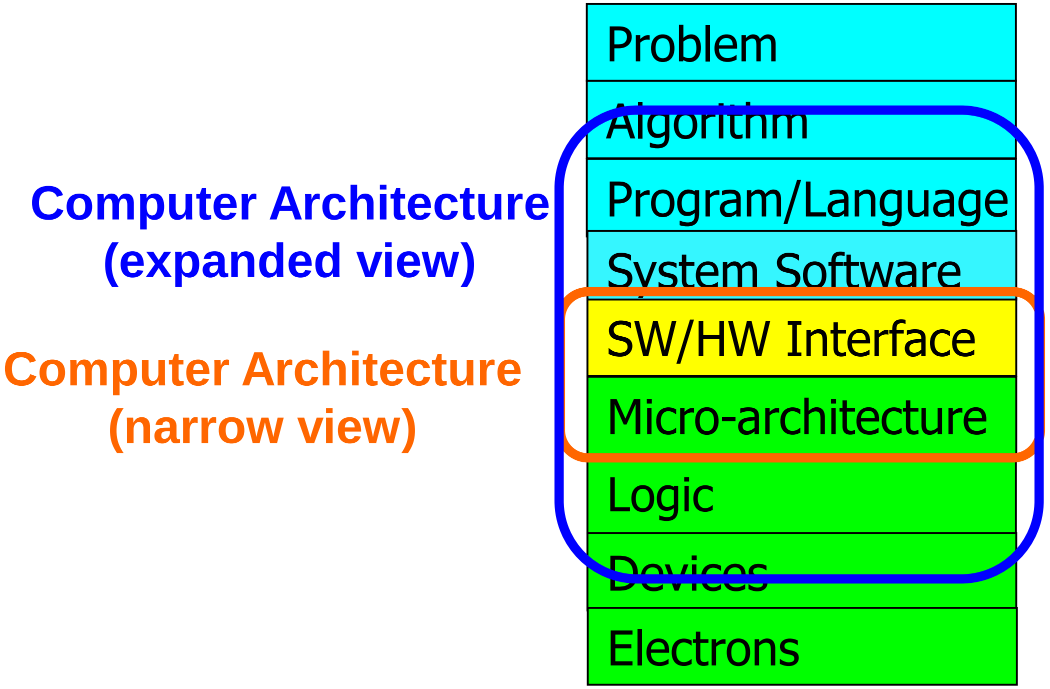
- Understand fundamentals of designing a modern processor and the key metrics;
- Gain fundamental knowledge and understandings of advanced computer architectures such as multiprocessors, superscalars, advanced caches, and prefetching schemes;
- Learn about state-of-the-art computer architecture simulators or frameworks;
- Develop understandings about certain design tradeoffs in implementing the modern computer architectures;
- Learn about the concept of high-level design languages, AI accelerators or domain-specific architectures;
- Work as a team to implement and optimize a given functional description of a major portion of a substantial processor with learned knowledge and tools;
- Be able to present the results and conclusions of an experimental project in a clear, logical, succinct, and informative written format.

What We Did Not Cover

- A LOT …
  - Virtual Machines
  - Request-level parallelism (RLP)
  - Interrupt and Exceptions
  - Other Dynamic Scheduling Algorithms
  - Distributed Systems
  - Warehouse Computing (Ch. 6 of textbook) / Cloud computing
  - Details about memory consistency
  - …
- But, I hope you can start from this course and develop learnings further (Read the textbook, take advanced courses, etc.).

Computer Architecture Design Philosophy (T1, T2)

Levels of Abstraction



- Each level of design imposes different requirements and constraints, which change over time.
- History and economics: there is commercial pressure to evolve in a way that minimizes disruption and possible costs to the ecosystem (e.g., software).
- There is also a need to look forward and not design for yesterday’s technology and workloads!
- Design decisions should be carefully justified through experimentation.

Figure: Prof. Onur Mutlu @ ETH

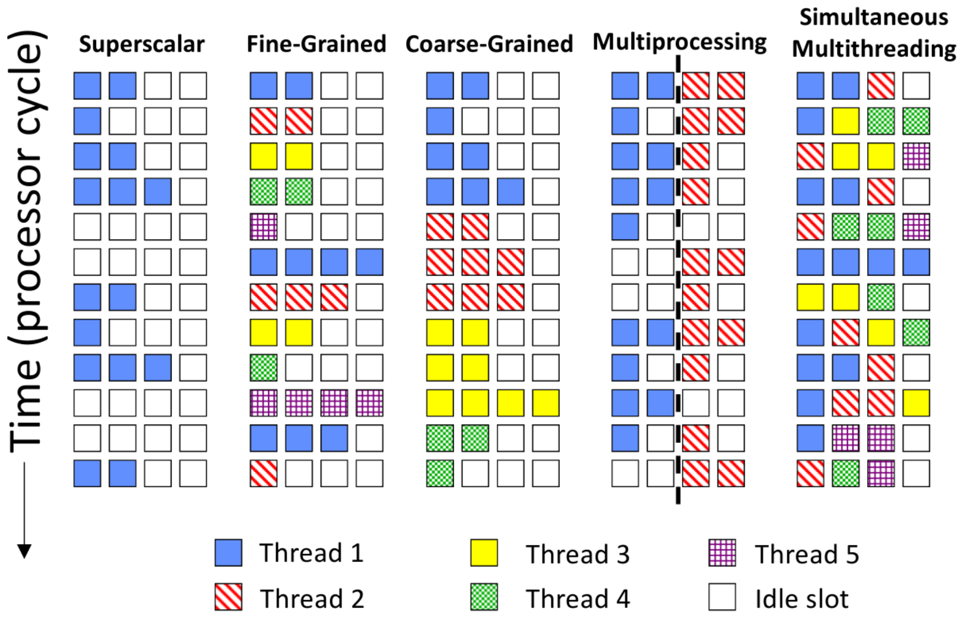
Computer Architecture

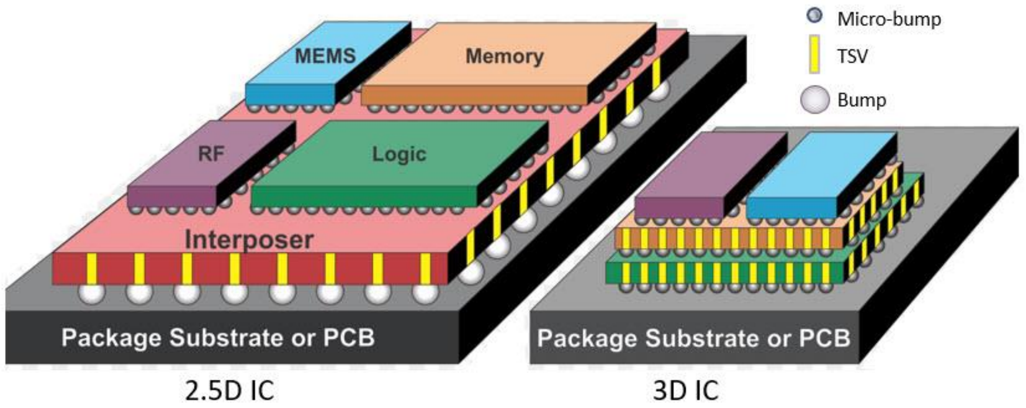
- Computer architecture is concerned with how best to exploit fabrication technology to meet marketplace demands.
  - *e.g., how best might we use five billion transistors and a power budget of two watts to design the chip at the heart of a mobile phone?*
- Computer architecture builds on a few simple concepts, but is challenging as we must constantly seek new solutions.
- What constitutes the “best” design changes over time and depending on our use-case. It involves considering many different trade-offs.

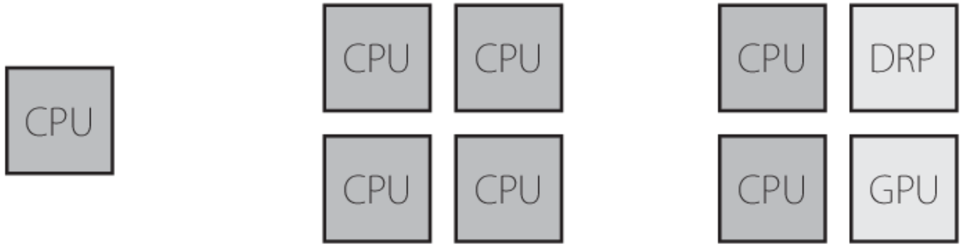
The trending architectures

- From sequential to parallel
- From horizontal to vertical
- From homogeneous to heterogenous
- From single die to multidie

The trending architectures



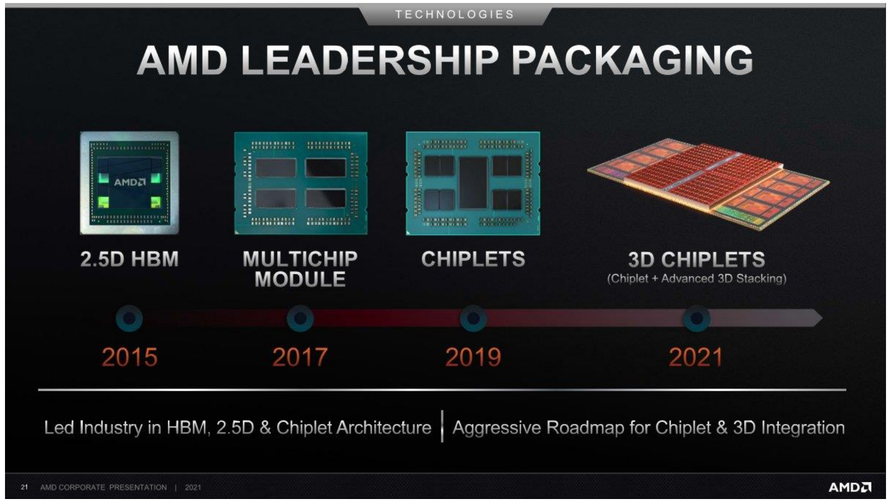




(a) Single-Core Processor

(b) Homogeneous Multi-Core Processor

(c) Heterogeneous Multi-Core Processor



TECHNOLOGIES

AMD LEADERSHIP PACKAGING

2.5D HBM	MULTICHIP MODULE	CHIPLETS	3D CHIPLETS (Chiplet + Advanced 3D Stacking)
2015	2017	2019	2021

Led Industry in HBM, 2.5D & Chiplet Architecture | Aggressive Roadmap for Chiplet & 3D Integration

21 AMD CORPORATE PRESENTATION | 2021

AMD

RISC vs CISC

RISC Philosophy

- Regularity & simplicity
- Leaner means faster
- Optimize the common case

CISC Rebuttal

- Compilers can be smart
- Transistors are plentiful
- Legacy is important
- Code size counts
- Micro-code!

Energy efficiency

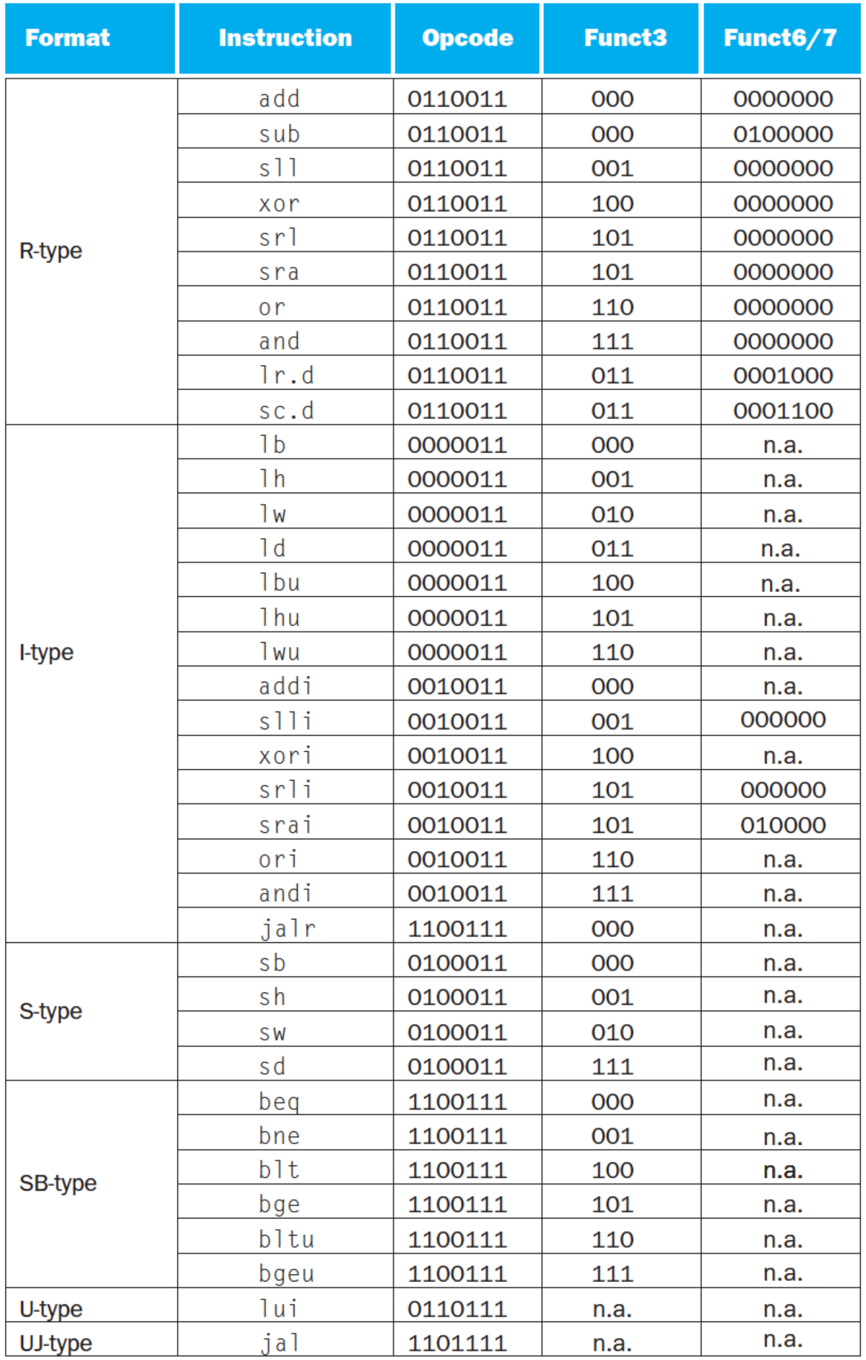
Embedded Systems

Phones/Tablets

Desktops/Servers

source: Prof. Hakim Weatherspoon, CS 3410

RISC-V Encoding Cheat Sheet



Format	Instruction	Opcode	Funct3	Funct6/7
R-type	add	0110011	000	0000000
	sub	0110011	000	0100000
	sll	0110011	001	0000000
	xor	0110011	100	0000000
	srl	0110011	101	0000000
	sra	0110011	101	0000000
	or	0110011	110	0000000
	and	0110011	111	0000000
	lr.d	0110011	011	0001000
	sc.d	0110011	011	0001100
I-type	lb	0000011	000	n.a.
	lh	0000011	001	n.a.
	lw	0000011	010	n.a.
	ld	0000011	011	n.a.
	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	lwu	0000011	110	n.a.
	addi	0010011	000	n.a.
	slli	0010011	001	000000
	xori	0010011	100	n.a.
	srli	0010011	101	000000
	srai	0010011	101	010000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.
S-type	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
	sd	0100011	111	n.a.
SB-type	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.
U-type	lui	0110111	n.a.	n.a.
UJ-type	jal	1101111	n.a.	n.a.

Borrowed from VE370 2021 Summer

CPU Performance

- Latency vs. Throughput
  - You can add latencies, but not throughput
    - Latency(P1+P2, A) = Latency(P1,A) + Latency(P2,A)
    - Throughput(P1+P2,A) != Throughput(P1,A) + Throughput(P2,A)

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

MIPS as a Performance Metric

- **MIPS**: Millions of Instructions Per Second (recall IPC)

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}\end{aligned}$$

- Doesn’t account for
  - Differences in ISAs between computers
  - Differences in complexity between instructions
- Example:
  - Compiler removes instructions, program faster
  - However, “MIPS” goes down (misleading)

Danger: Partial Performance Metrics

- Micro-architects often ignore dynamic instruction count
  - Typically work in one ISA/one compiler → treat it as fixed
- Micro-architects often ignore instructions/program
- General public (mostly) also ignores CPI
  - Equates clock frequency with performance!!
- Example: which processor would you buy?
  - A: CPI=2, f=500MHz, B: CPI=1, f=300MHz (assume the same ISA/compiler)
  - A? But B is faster!

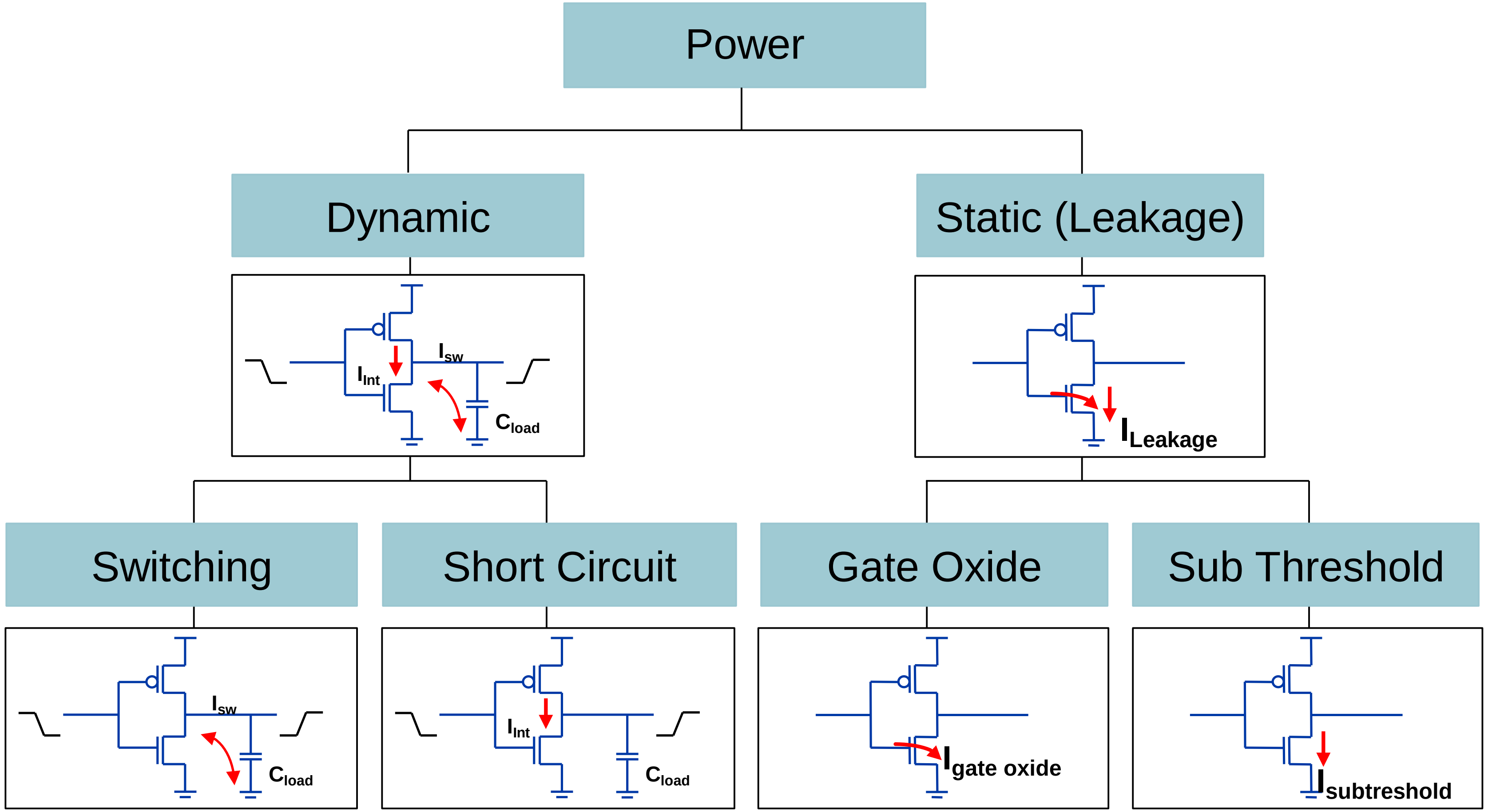
Source: Prof. Ron Dreslinski @ UMich, EECS 470

Quantitative Principles of Computer Design

- Take Advantage of Parallelism
  - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units
- Principle of Locality
  - Reuse of data and instructions
- Speculation
  - Guess you are right/wrong all the time
- Memorization
  - Programs do the same thing over and over again.
- Focus on the Common Case
  - … but speedup ultimately limited by the uncommon case
  - Amdahl’s Law

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

Sources of Power



Power

Dynamic

Static (Leakage)

Switching

Short Circuit

Gate Oxide

Sub Threshold

Source: Synopsys

Power Equation

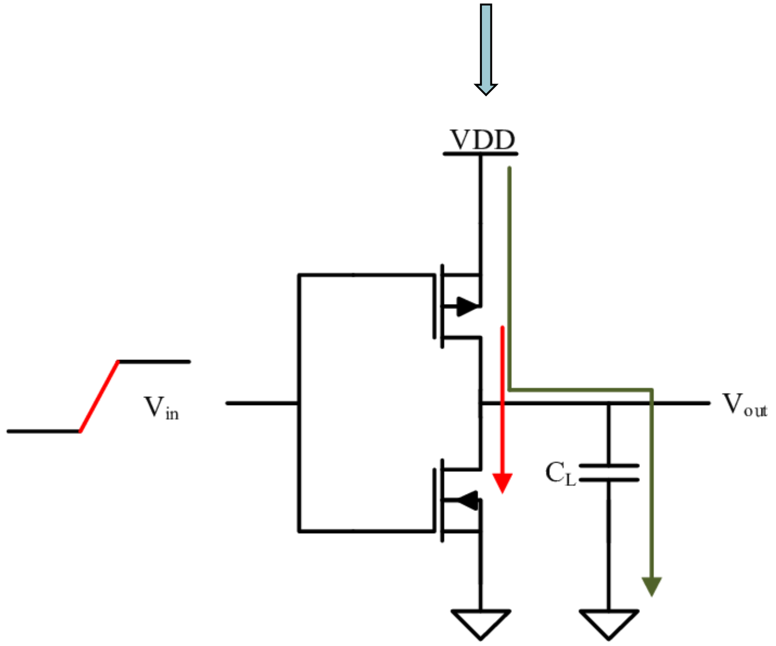
1/2 typically, but not always

$$P = \alpha.f.C_L.V_{DD}^2 + f.I_{peak}.V_{DD} + V_{DD}.I_{static}$$

Dynamic power

Short-circuit power

Static power



Borrowed from Xinfei Guo@JI, ECE4810J, 21FA

Dynamic Power

Capacitance:
Function of fan-out, wire length, transistor sizes

Supply Voltage:
Has been dropping with successive generations

$$P_{dyn} = C_L \, V_{DD}^2 \, \alpha f$$

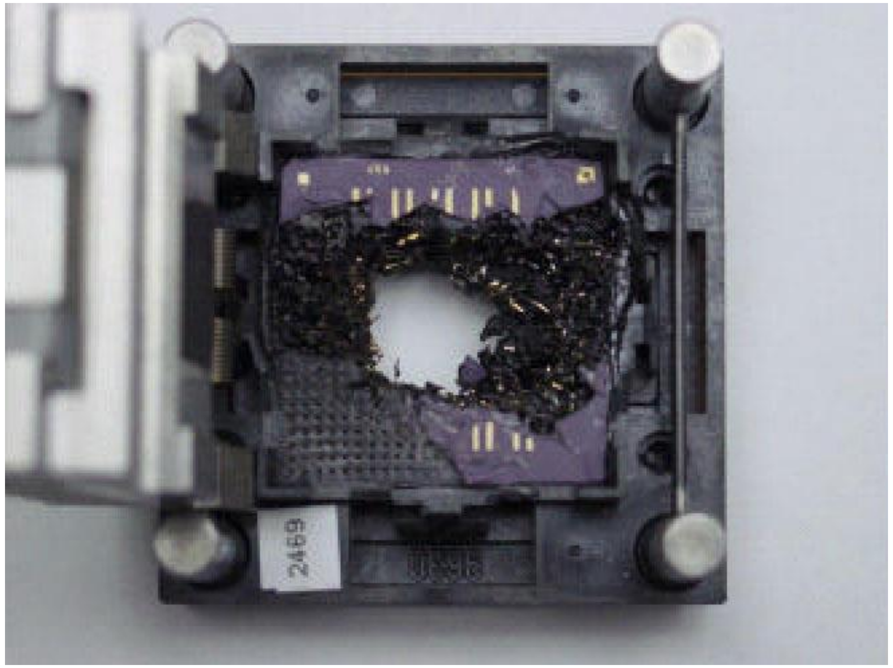
Activity factor:
How often, on average, do wires switch?

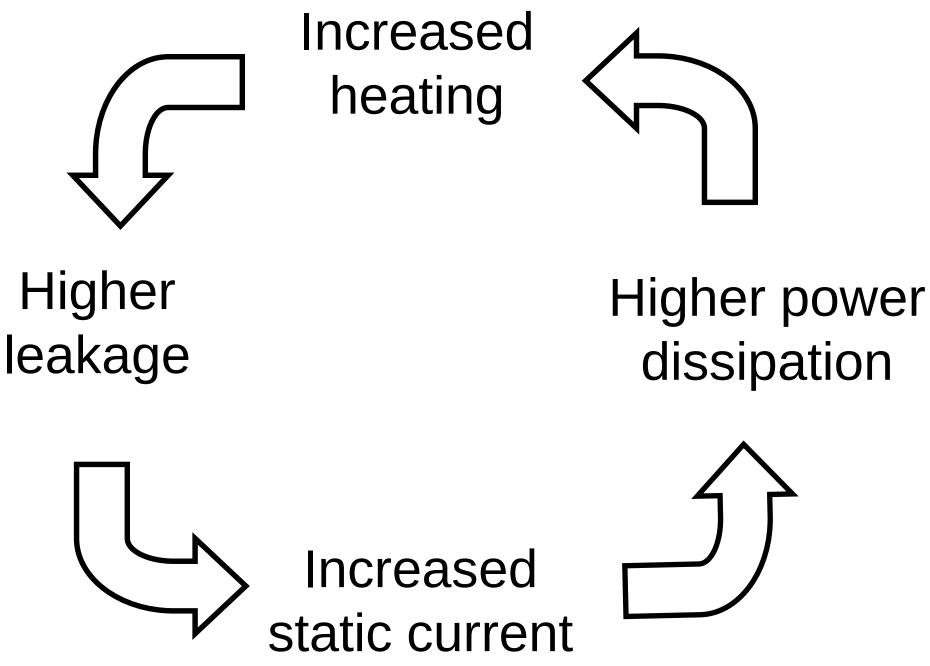
Clock frequency:
Increasing…

Static Power

- Concept
- Static Power vs. Temp.
- Thermal Runaway

Thermal Runaway

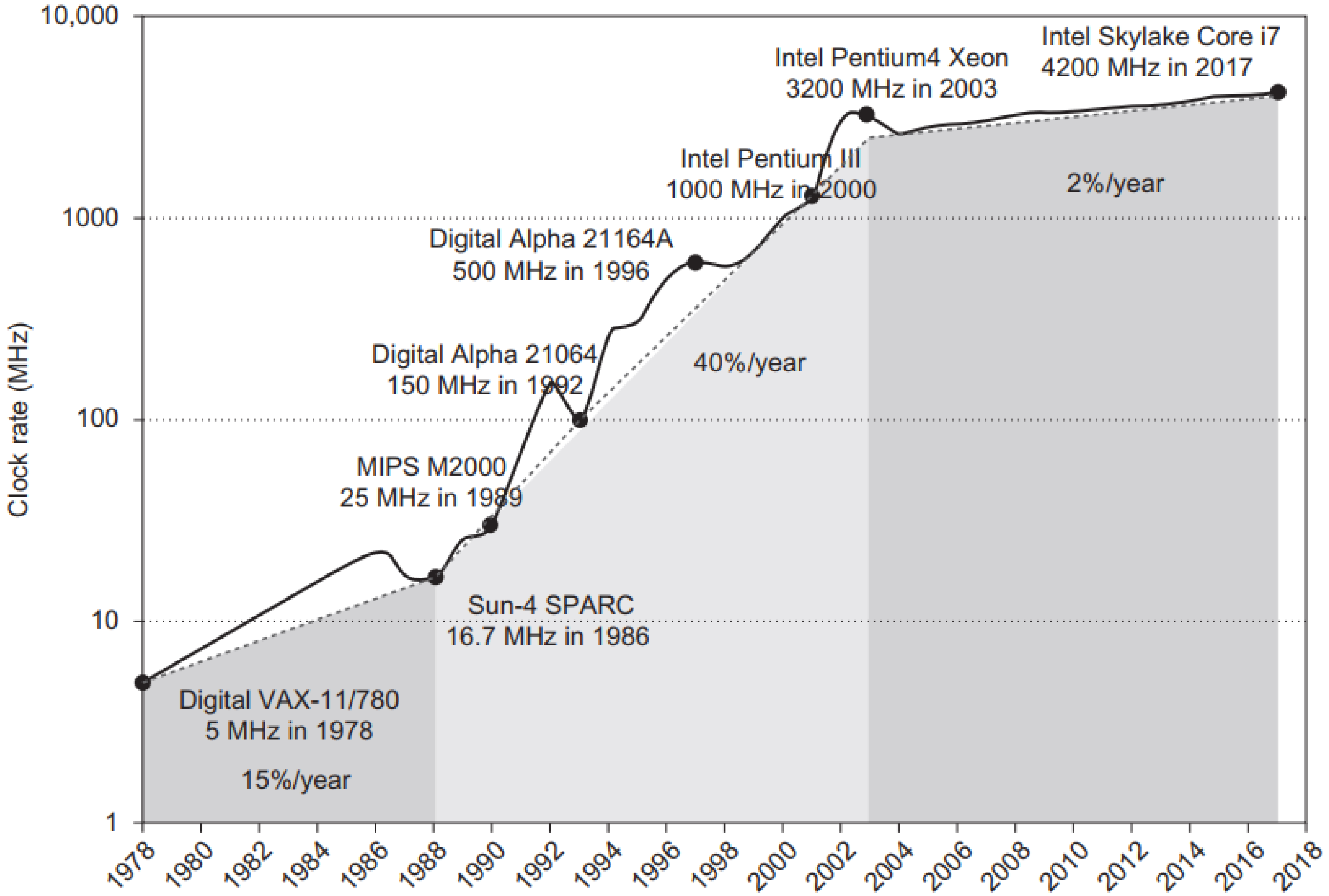




- Destructive positive feedback mechanism
- Leakage increases exponentially with temperature
- May destroy the test socket → thermal sensors required

source: synopsys

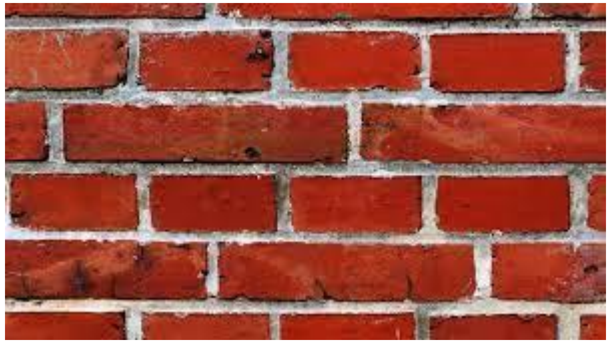
What happened after 2003?

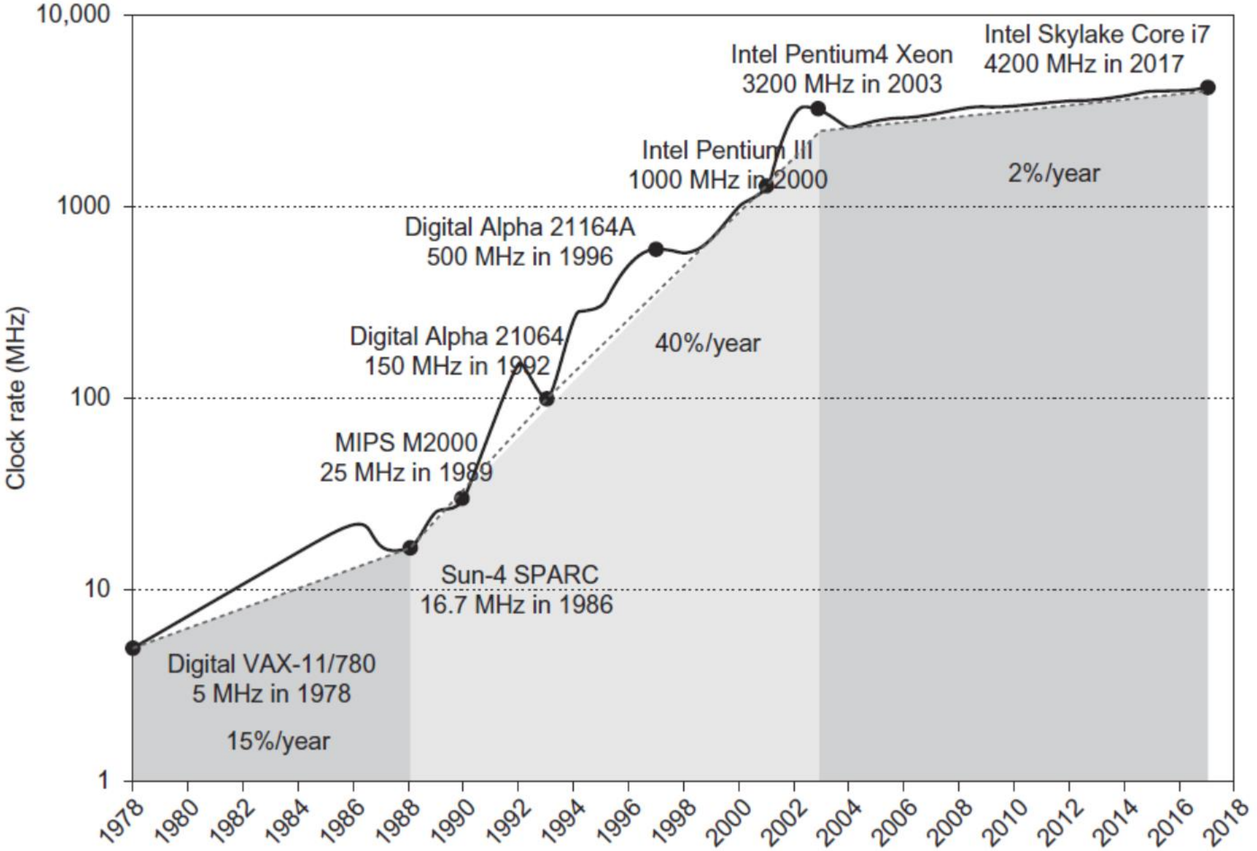


Heat a wall due to power issues (will be covered soon)…

Power Wall

- Unfortunately, the increased the power dissipation of the CPU chip beyond the capacity of inexpensive cooling techniques.
- 3.3 GHz Intel Core i7 consumes 130W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air
- We hit a wall!





https://thermtest.com/how-the-thermal-conductivity-of-clay-bricks-contributes-to-their-success-as-a-building-material

Power vs. Energy

- Power consumption in Watts
  - Sets packaging limits
  - Indicates heat
  - TDP
  - Power Wall
- Energy efficiency in joules
  - Rate at which energy is consumed over Time
  - Determines battery life in hours
  - Energy = power * delay (joules = watts * seconds)
  - Lower energy number means less power to perform a computation at same frequency

Combined Power-Performance Metrics

- Power-delay Product (PDP or Energy) = Pavg * t
  - PDP is the average energy consumed per switching event
- Energy-delay Product (EDP) = PDP * t
  - Takes into account that one can trade increased delay for lower energy/operation
- Energy-delay^2 Product (EDDP) = EDP * t
  - Why do we need so many formulas?!
  - We want a voltage-invariant efficiency metric! Why?
  - Power ~ ½ CV^2f, Performance ~ f (and V)

TDP – Thermal Design Power

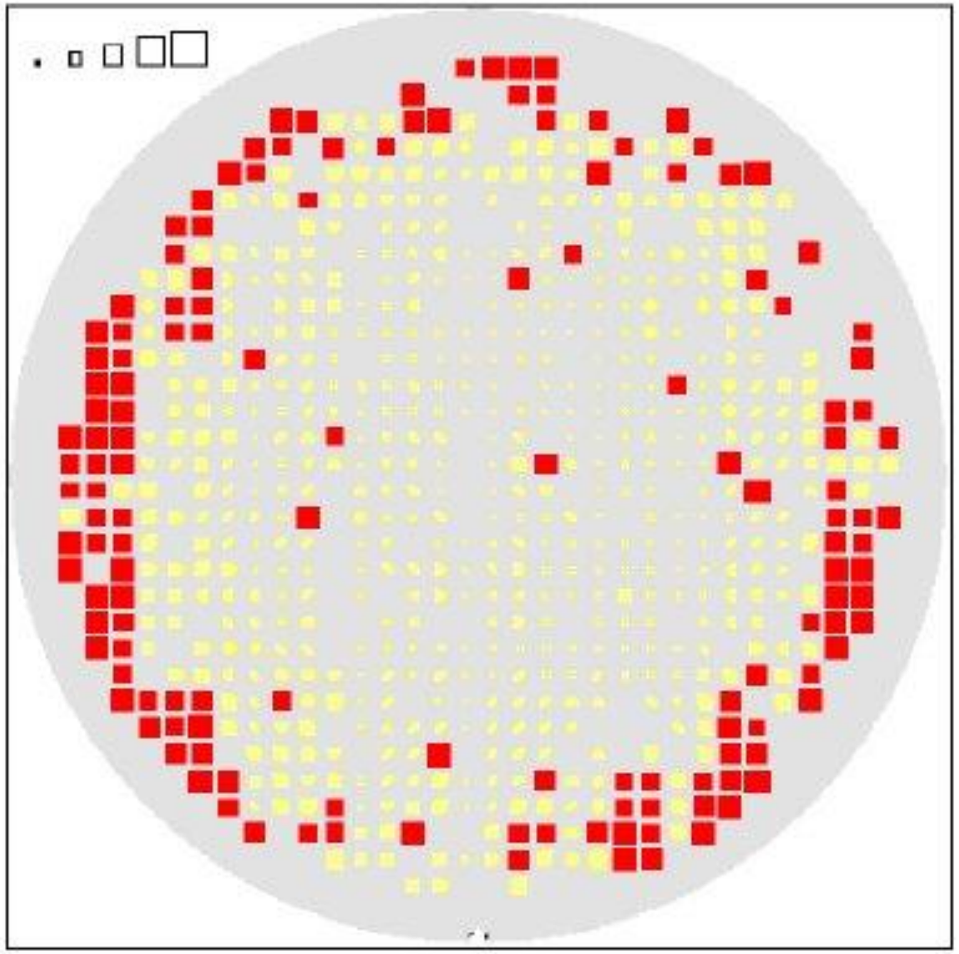
- A metric that is expressed in watts.
- It refers to the amount of power/heat a cooling system (i.e. heatsink, fan) is expected to dissipate to prevent overheating.
- The thermal design power is the maximum power a processor can draw for a thermally significant period while running commercially useful software.
- The TDP does not typically indicate the most power the chip could ever draw, but how much it draws under real load.
- Lower than peak power (1.5X higher), higher than average power consumption
- You can, generally, use TDP to gauge power consumption of a chip, in that the smaller the number for TDP, the lower power consumption by the CPU.

References

- https://www.intel.com/content/dam/doc/white-paper/resources-xeon-measuring-processor-power-paper.pdf

Area and Cost

- Concept of Yield
- How yield affects area & cost



🟥 Failing Die

Reliability

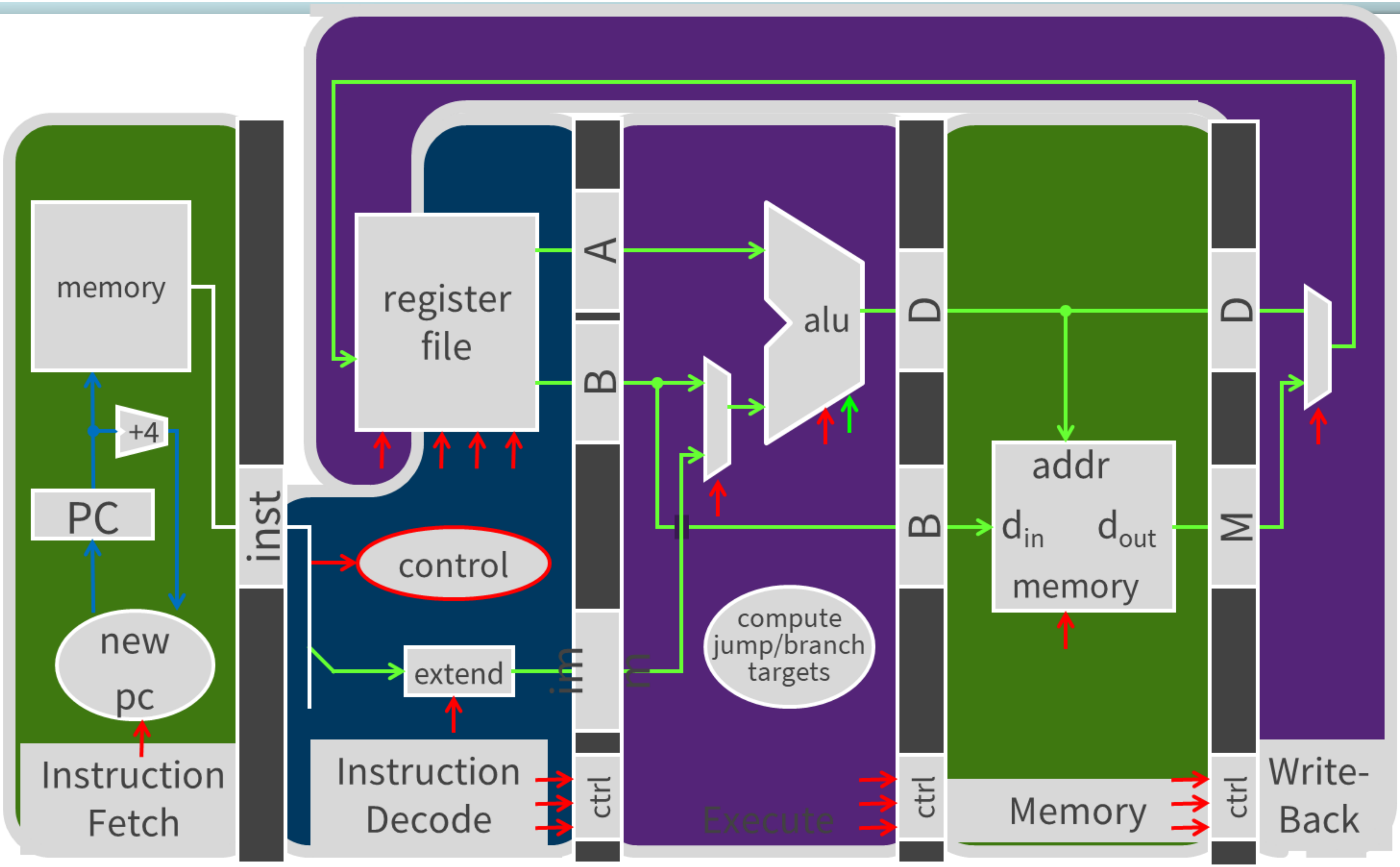
- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

Factors

- Reliability is related to die area, clock frequency, and power.
- Die area increases the amount of circuitry and the probability of a fault.
- It also allows the use of error correction and detection techniques.
- Higher clock frequencies increase electrical noise and noise sensitivity.
- Faster circuits are smaller and more susceptible to radiation.

The Pipeline (T3, T4)

A five-stage pipeline



A single cycle RISC-V Processor

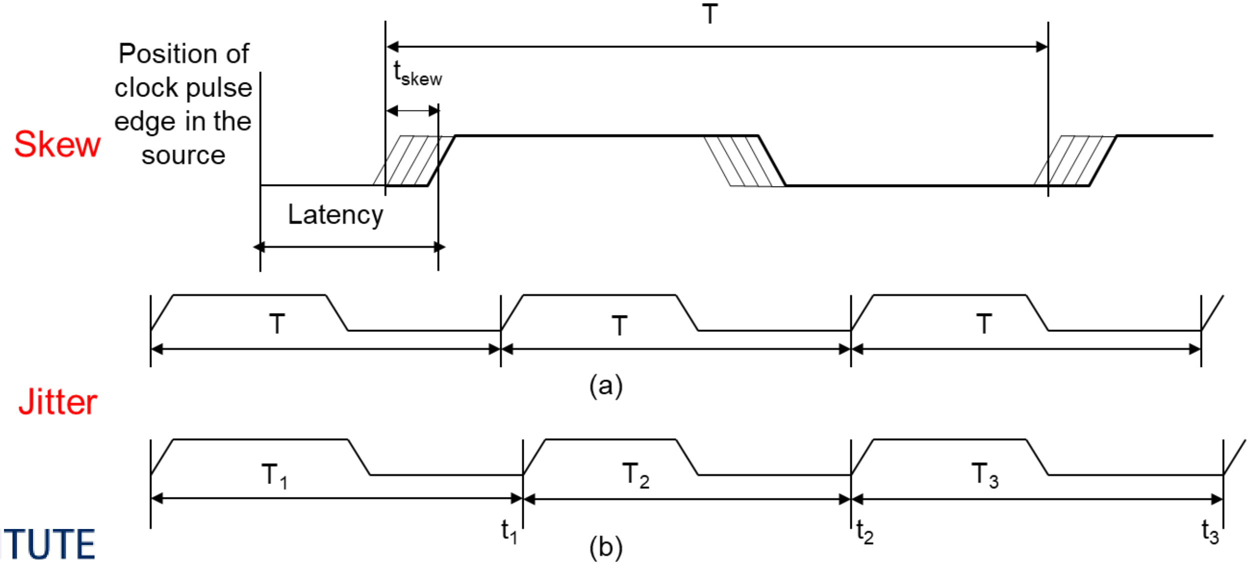
source: Prof. Hakim Weatherspoon @Cornell

Principles of Pipelined Implementation

- Break datapath into multiple cycles (e.g. 5 in VE370)
  - Parallel execution increases throughput
  - **Balanced pipeline very important**
    - Slowest stage determines clock rate
    - Imbalance kills performance
- Add pipeline registers (flip-flops) for isolation
  - Each stage begins by reading values *from* latch
  - Each stage ends by writing values *to* latch
- We will also need to be careful to pipeline our control signals so that decoded control information accompanies each instruction as they progress down the pipeline.
- Resolve hazards

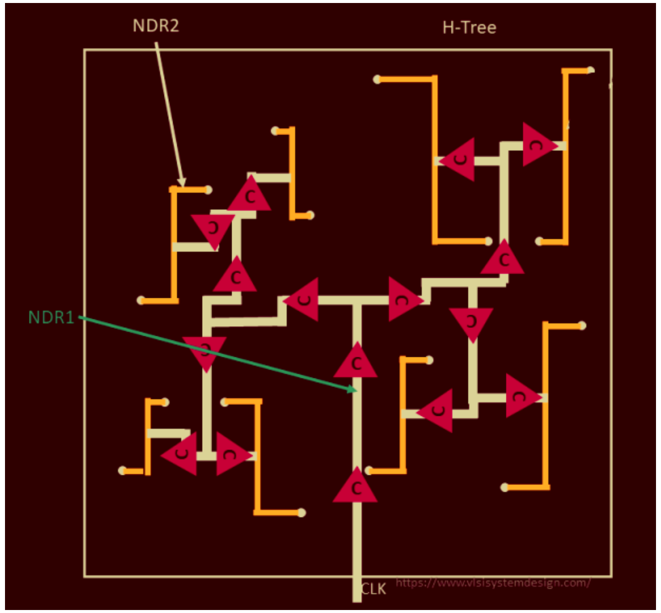
Q: The more pipeline stages, the faster?

- So far we have assumed pipeline registers are “free”
- The fact
  - Clocks have skews and jitters (can be added to each register)
    - Clock skew = systematic clock edge variation between sites
      - Mainly caused by delay variations introduced by manufacturing variations
    - Random variation
      - Clock jitter = variation in clock edge timing between clock cycles
      - Mainly caused by noise



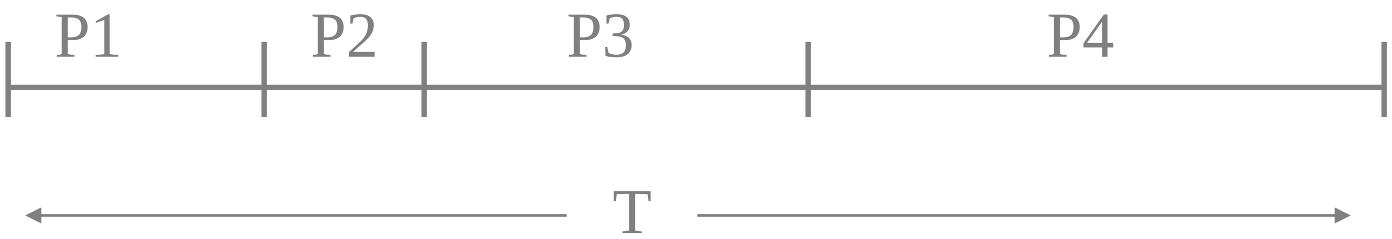
(a)

(b)



https://www.vlsiguide.com/2018/07/clock-tree-synthesis-cts.html

Optimum pipelining



$P_{\max\ i}$ = delay of the i th functional unit

suppose $T = \Sigma_i P_{\max\ i}$ without clock overhead

S = number of pipeline segments

C = clock overhead

$T/S \geq \max\ (P_{\max\ i})$ [quantization]

Taken from VE481, 2021 FA

$$\overset{\text{Cycle time}}{\Delta t} = \overset{\text{Avg. Time / segment}}{T/S} + \overset{\text{Clock overhead}}{C}$$

performance = 1/ (1+(S - 1)b) [IPC]

throughput = G = performance / Δt [IPS]

$$G = \left(\frac{1}{1+(S-1)\,b}\right) \times \left(\frac{1}{(T/S)+C}\right)$$

Find S for optimum performance by solving for S:

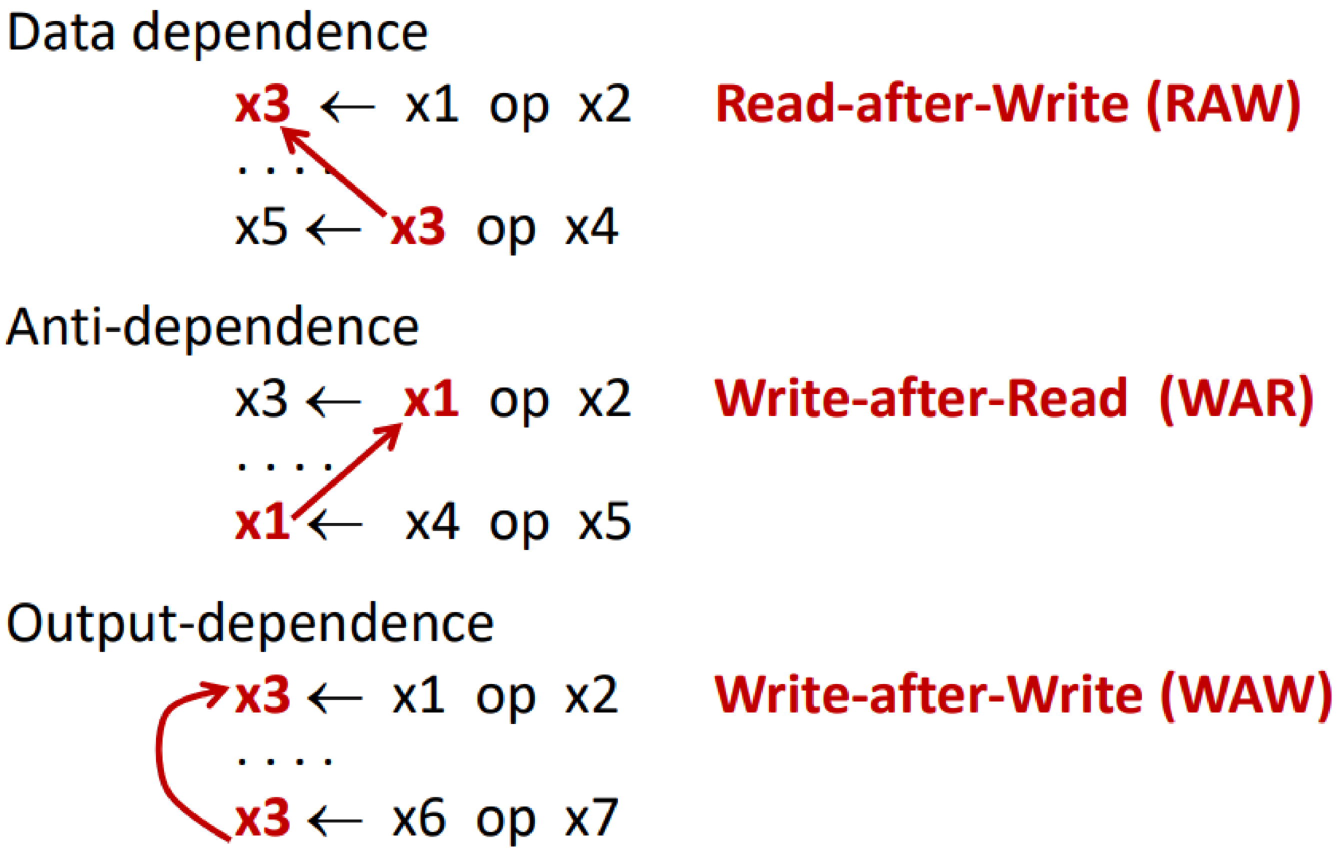
$$\frac{dG}{dS} = 0$$

we get $S_{\text{opt}} = \sqrt{\dfrac{(1-b)\,T}{bC}}$

Taken from VE481, 2021 FA

Types of Data hazards

- Consider executing a sequence of register-register instructions of type:



Data dependence

x3 ← x1 op x2 **Read-after-Write (RAW)**

. . . .

x5 ← x3 op x4

Anti-dependence

x3 ← x1 op x2 **Write-after-Read (WAR)**

. . . .

x1 ← x4 op x5

Output-dependence

x3 ← x1 op x2 **Write-after-Write (WAW)**

. . . .

x3 ← x6 op x7

Note that the RAR (read after read) case is not a hazard.

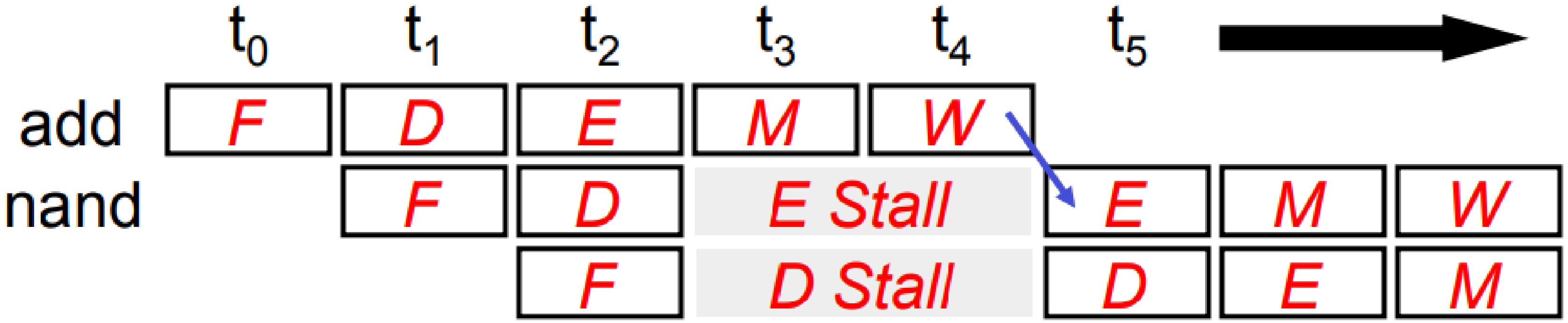
source: James C. Hoe, CMU/ECE/CALCM, 2022

The Cost of Deeper Pipelines

- Instruction pipelines are not ideal
  - Assume

```
add 3 1 2
nand 5 3 4
```

RAW!!



	t0	t1	t2	t3	t4	t5		
add	F	D	E	M	W			
nand		F	D	E Stall		E	M	W
			F	D Stall		D	E	M

source: Prof. Ronald Dreslinski @ Umich, EECS470

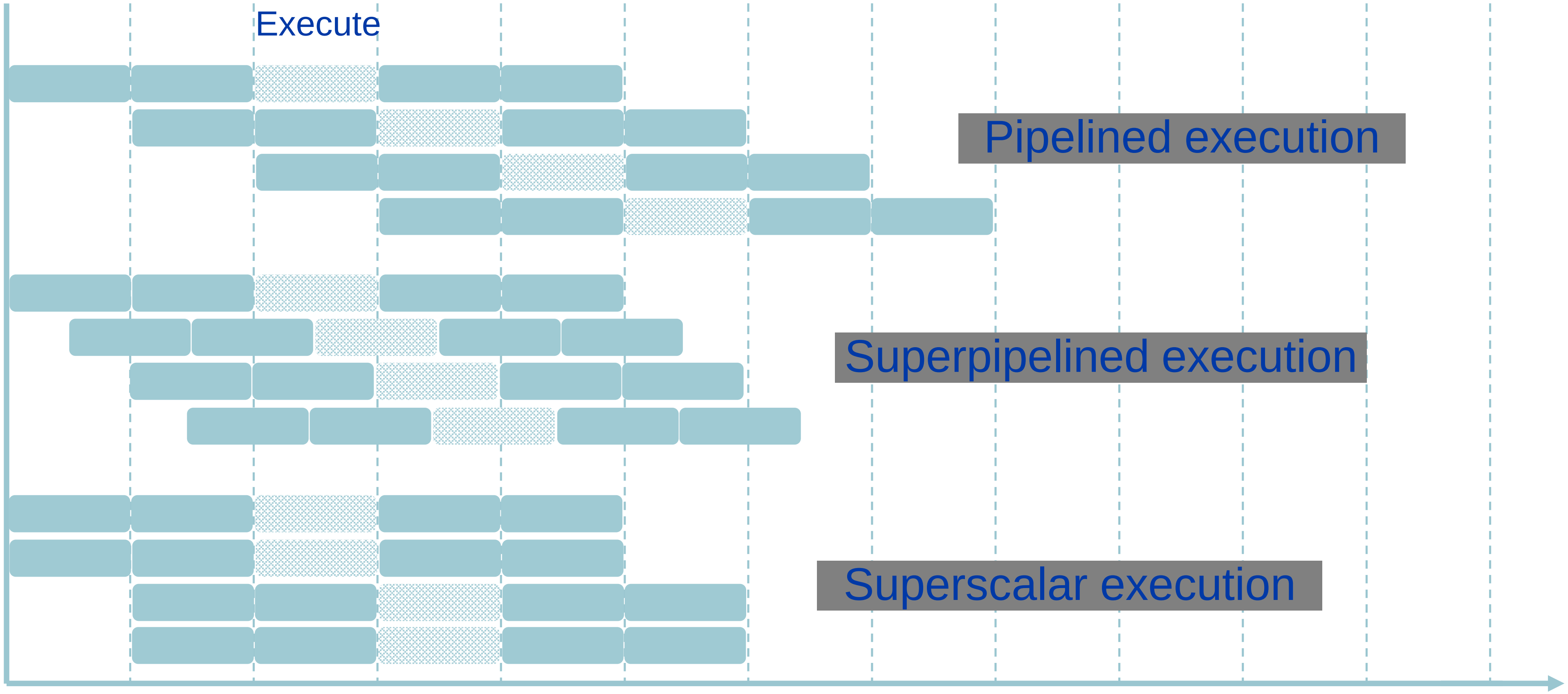
Handling Data Hazards

- Avoidance (static)
  - Make sure that there are no hazards in the code
- Interlock/stall (dynamic)
  - Stall until earlier instructions finish
- forward/bypass (dynamic)
  - Resolve hazard earlier by bypassing value as soon as available
- Second and third require detection
- Load-Use Hazard Detection

Handling Control Hazards

- Stall on branch
- Always assume branch not taken or taken
- Branch prediction
- Delayed Branch

Superpipelined and Superscalar Processors

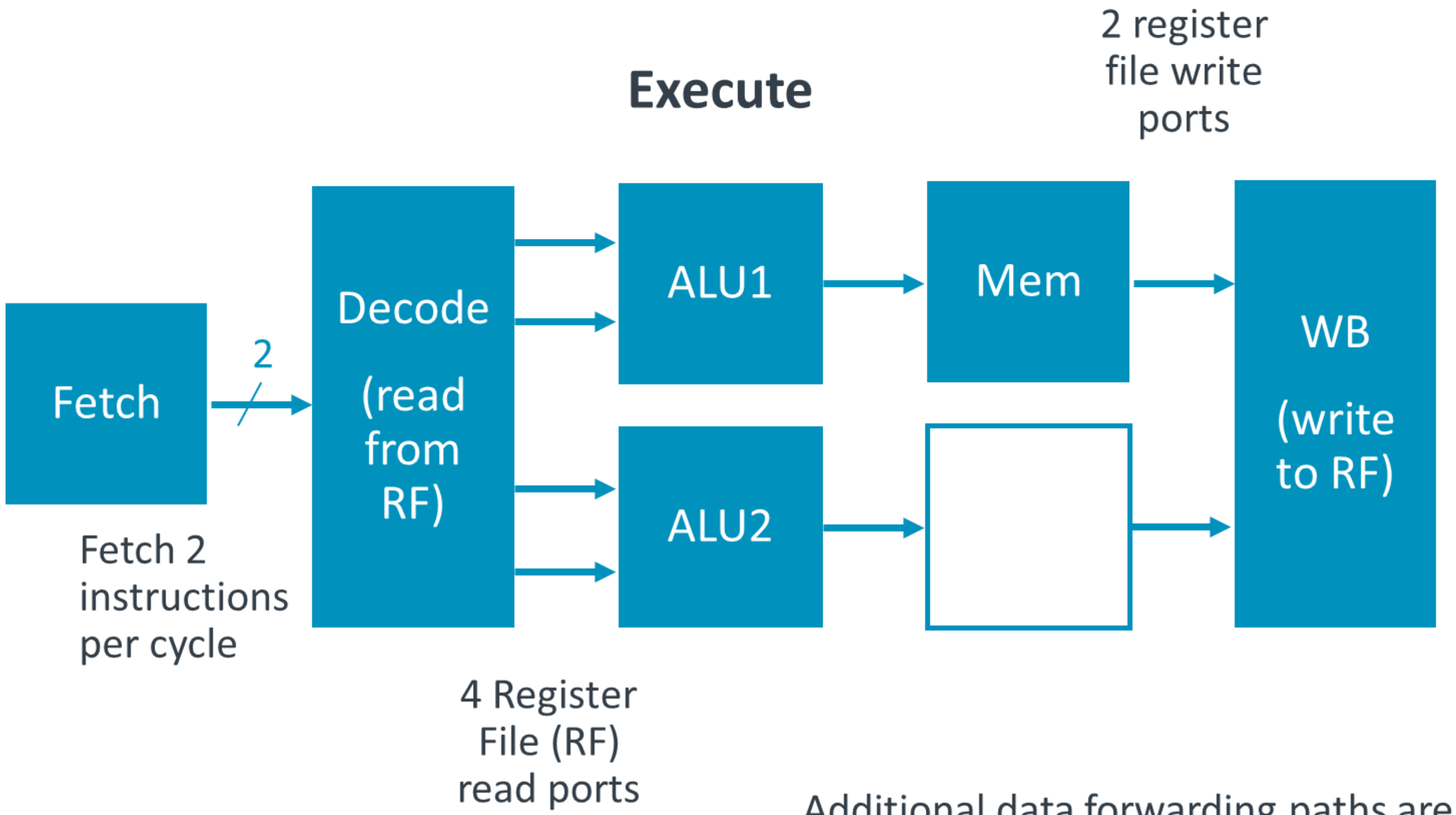


Time in “base” clock cycles

In-order vs. Out-of-order Execution

- In-order Execution
  - Instructions are fetched, executed and computed in compiler-generated order
  - one stalls, they all stall
  - instructions are “statically scheduled” by the hardware
- Out-of-order (OoO or O3 or O^3) Execution
  - Instructions are fetched in compiler-generated order
  - Instructions are completed in order
  - They might be executed in some other order
  - instructions behind a stalled instruction can bypass it
  - instructions are “dynamically scheduled” by the hardware

Simple In-order Superscalar Processors



Additional data forwarding paths are also required (not shown here), from and to both ALUs.

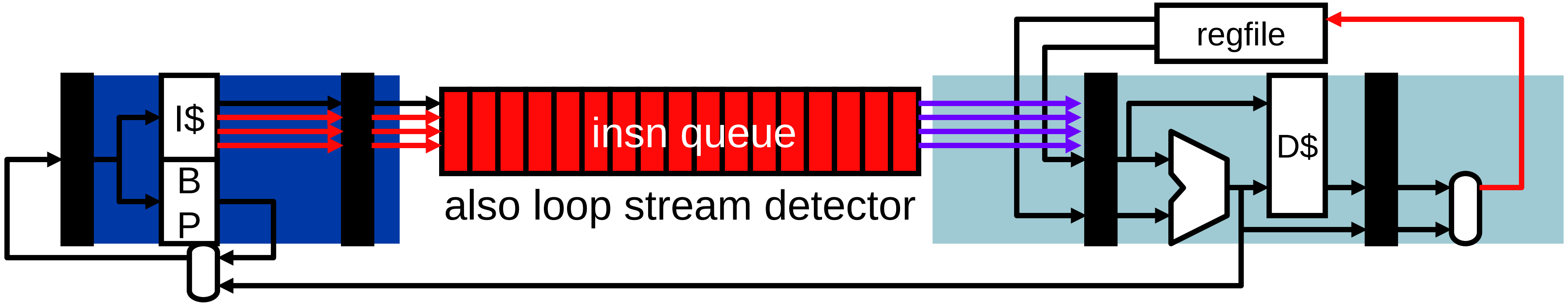
source: ARM

Superscalar Challenges - Front End

- **Superscalar instruction fetch**
  - Modest: fetch multiple instructions per cycle
  - Aggressive: buffer instructions and/or predict multiple branches
- **Superscalar instruction decode**
  - Replicate decoders
- **Superscalar instruction issue**
  - Determine when instructions can proceed in parallel
  - More complex stall logic - order **N^2** for *N*-wide machine
  - Not all combinations of types of instructions possible
- **Superscalar register read**
  - Port for each register read (4-wide superscalar ➔ 8 read "ports")
  - Each port needs its own set of address and data wires
    - Latency & area $\propto$ #ports^2

Source: Dr. Joseph Devietti, CIS 571: Comp. Org & Design

Increasing Superscalar Fetch Rate



- Option #1: over-fetch and buffer
- Option #2: “loop stream detector” (Core 2, Core i7)
- Other options: next-*next*-block prediction, “trace cache”

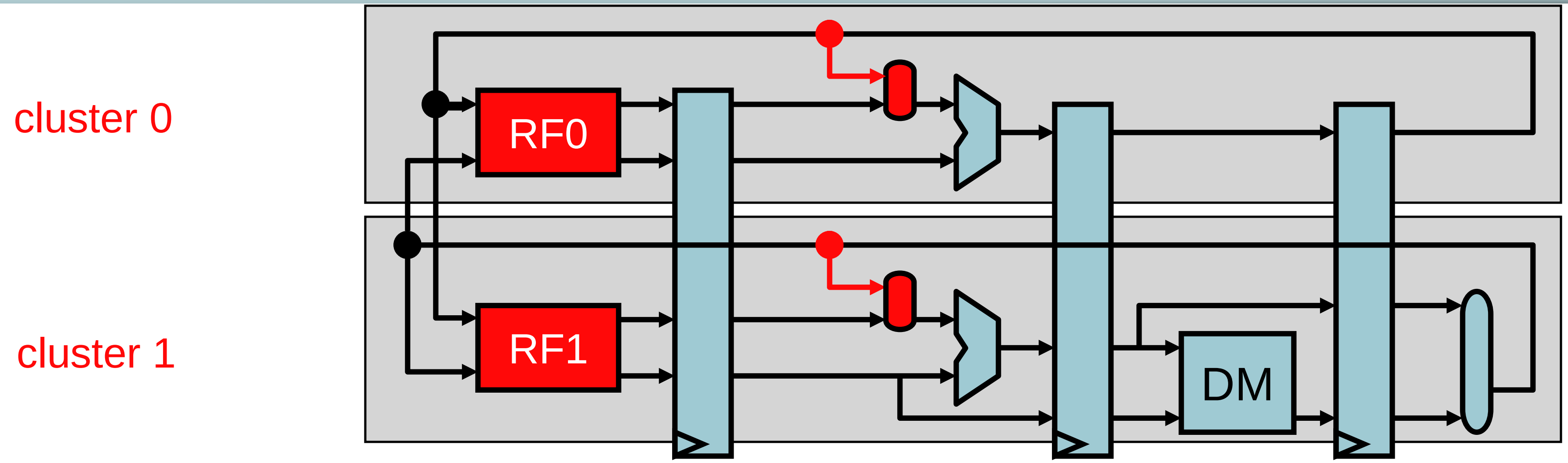
Source: Dr. Joseph Devietti, CIS 571: Comp. Org & Design

Superscalar Challenges - Back End

- **Superscalar instruction execution**
  - Replicate arithmetic units (but not all, for example, integer divider)
  - Perhaps multiple cache ports (slower access, higher energy)
    - Only for 4-wide or larger (why? only ~35% are load/store insn)
- **Superscalar bypass paths**
  - More possible sources for data values
  - Order (N^2 * P) for *N*-wide machine with execute pipeline depth *P*
- **Superscalar instruction register writeback**
  - One write port per instruction that writes a register
  - Example, 4-wide superscalar ➔ 4 write ports
- **Fundamental challenge:**
  - Amount of ILP (instruction-level parallelism) in the program
  - Compiler must schedule code and extract parallelism

Source: Dr. Joseph Devietti, CIS 571: Comp. Org & Design

Mitigating N^2 RegFile with Clustering

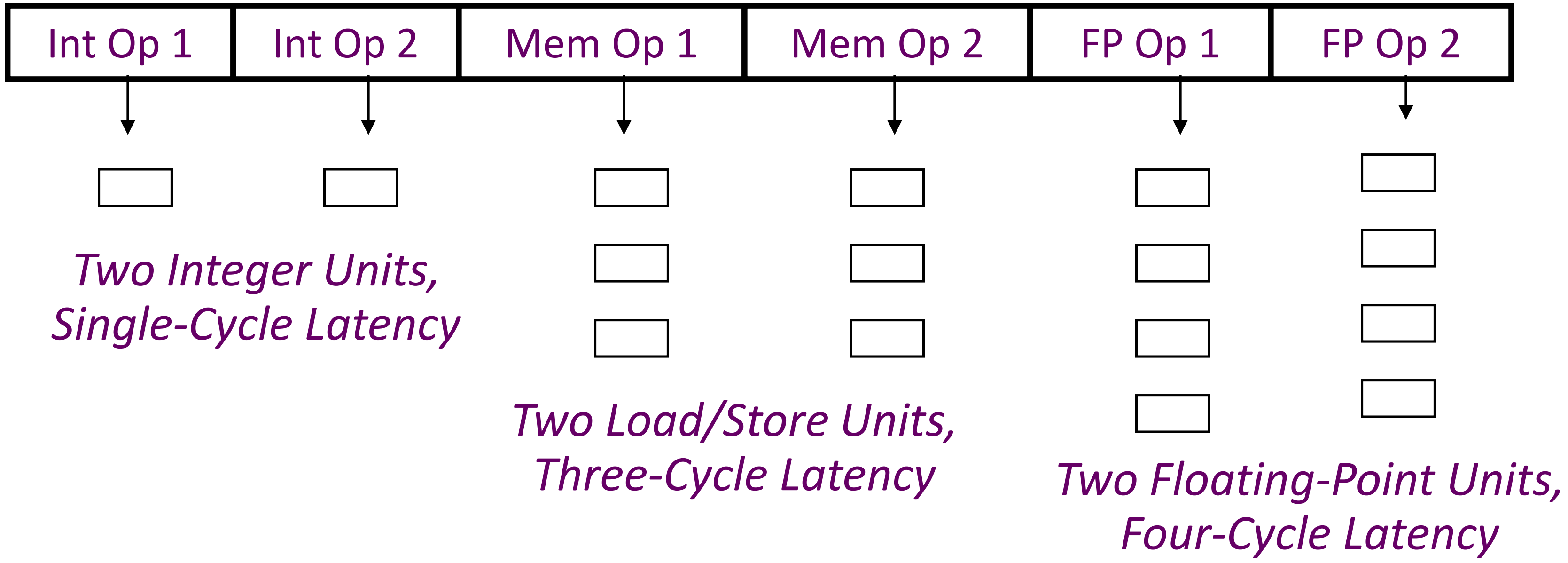


- **Clustering**: split **N**-wide execution pipeline into **K** clusters
  - With centralized register file, 2N read ports and N write ports

- **Clustered register file**: extend clustering to register file
  - Replicate the register file (one replica per cluster)
  - Register file supplies register operands to just its cluster
  - All register writes go to all register files (keep them in sync)
  - Advantage: fewer read ports per register!

Source: Dr. Joseph Devietti, CIS 571: Comp. Org & Design

VLIW: Very Long Instruction Word


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single-Cycle Latency
Two Load/Store Units, Three-Cycle Latency
Two Floating-Point Units, Four-Cycle Latency

- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
  - Parallelism within an instruction => no cross-operation RAW check
  - No data use before data ready => no data interlocks

Out-of-Order Pipeline (high-level view)

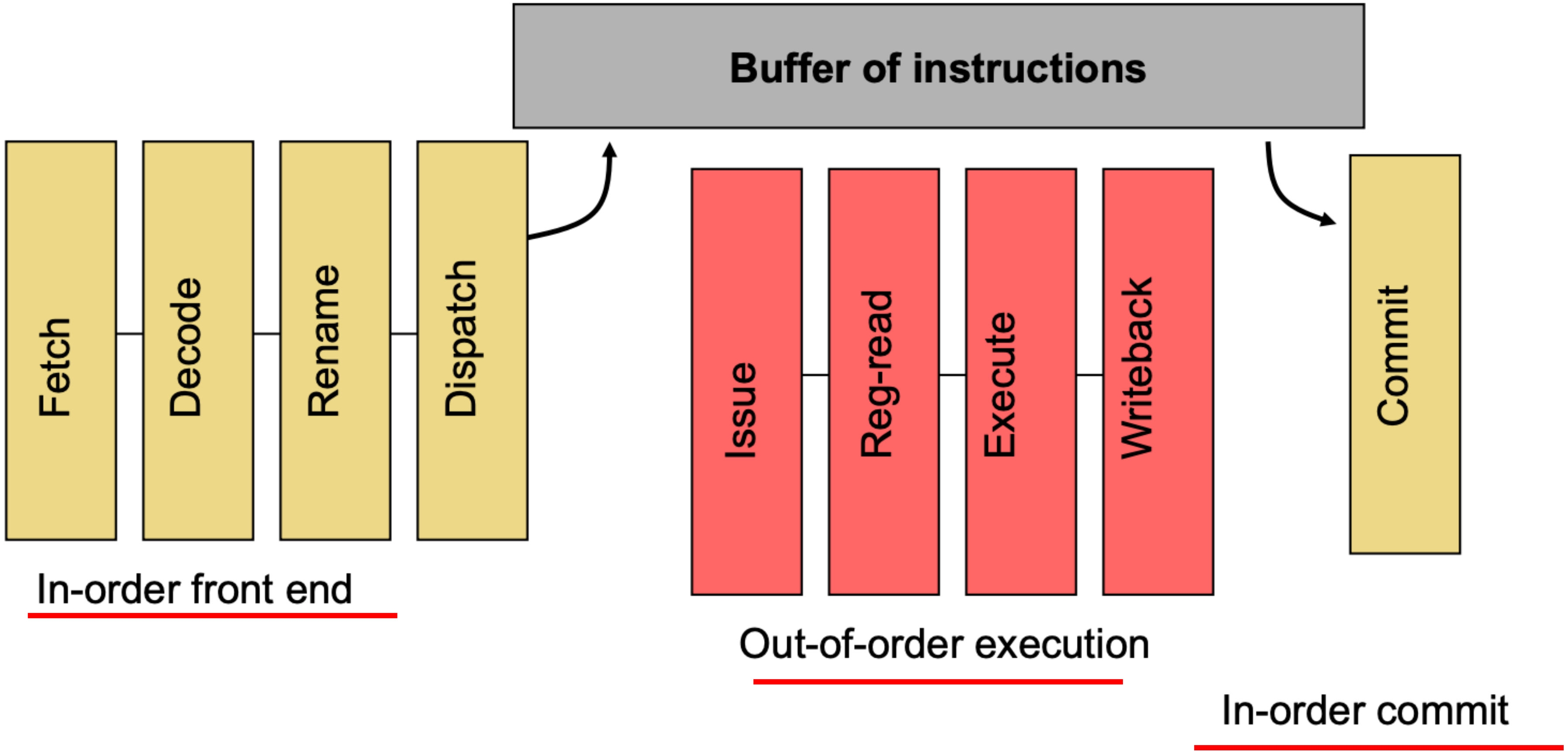


Image: cis.upenn.edu/~cis571/

Out-of-Order Execution

- Also called "Dynamic scheduling"
  - Done by the hardware on-the-fly during execution

- Looks at a "window" of instructions waiting to execute
  - HW examines a sliding window of consecutive instructions
  - Each cycle, picks the next ready instruction(s)

- Two steps to enable out-of-order execution:

  Step #1: Register renaming – to avoid "false" dependencies

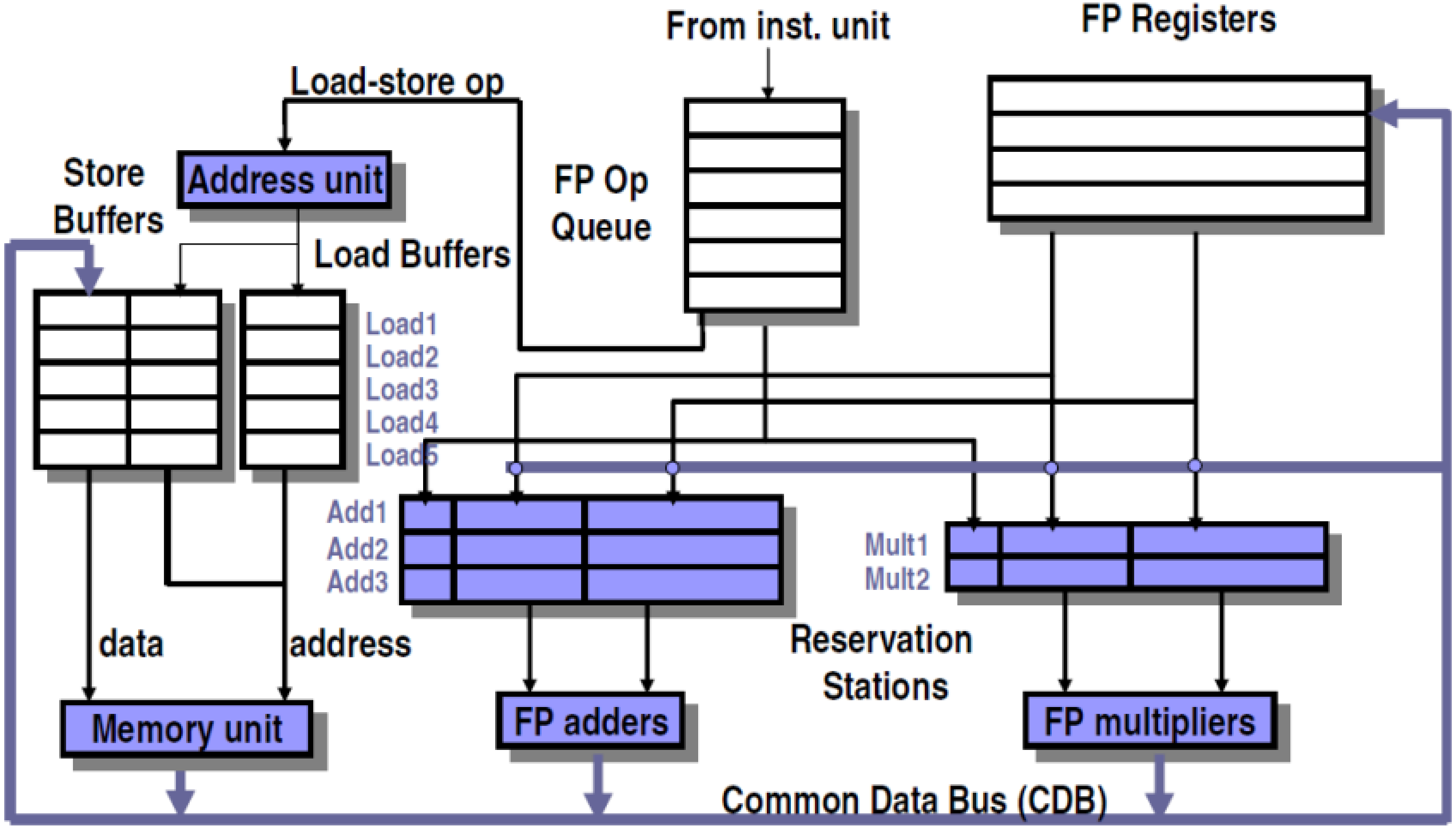
  Step #2: Dynamically schedule – to enforce "true" dependencies

- Key to understanding out-of-order execution:
  - **Data dependencies**

OoO Important Steps

- Register Renaming Algorithm
- Dispatch Steps
- Dynamic Scheduling/Issue Algorithm
- Issue
- Re-order Buffer (ROB)
- Commit
- **Check more examples on the slide and book**

Tomasulo Organization



Tomasulo Organization

Adapted from Nanjing U, CS252@UC Berkeley and CS 246@Harvard University

Store Queue + Load Queue

- Store Queue: handles forwarding, allows OoO stores
  - Entry per store (allocated @ dispatch, deallocated @ commit)
  - Written by stores (@ execute)
  - Searched by loads (@ execute)
  - Read from SQ to write data cache (@ commit)

- Load Queue: detects ordering violations
  - Entry per load (allocated @ dispatch, deallocated @ commit)
  - Written by loads (@ execute)
  - Searched by stores (@ execute)

- Both together
  - Allows aggressive load scheduling
  - Stores don’t constrain load execution

Where are we Heading?

- RC2: Course Review II