Lecture 1
Introduction / Overview

EEC 171 Parallel Architectures

John Owens
UC Davis

Credits

Administrivia

Teaching Staff

- Instructor
  - John Owens | jowens@ece | www.ece/~jowens/
    - Office hour: Th 2:30-3:30p (office: Kemper 3175) or by appointment
- TA
  - Terry O’Neill | toneill@ucdavis
    - Office hour: M 3–4p (Kemper 2101)

Electronic Resources

- https://canvas.ucdavis.edu/courses/114024/
  - Your instructor is ~new to Canvas, help him out
- Slack for most communication!
  - Signup link on the syllabus page
  - Do not send mail to my personal account if you want a timely reply.
  - You'll want to address to-the-instructor comments to both Terry and me
  - In general you can post questions to specific channels (#hw1, #ilpproject, etc.) so that all students can benefit from them

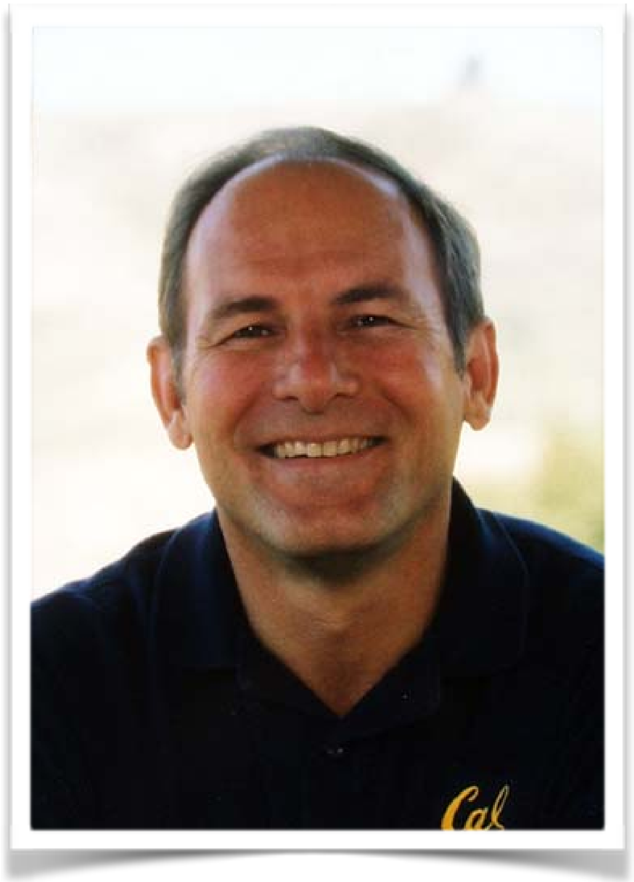
Classroom Time

- Class is TTh 10–2
- 10–11:30: “Lecture”
- 11:30–12: “Discussion”
- In reality: merged together
  - Discussion (~60%) will be used for lecture
  - Also for (~40%) project descriptions (TA), etc.
- Small class—let’s make it interactive!
- Administrative announcements: In middle of class

Lecture Style

- Students sometimes prefer blackboard
- It’s hard to show complex diagrams on a blackboard though
  - Plus I like slides better
  - I’ll give you my notes—spend your time thinking not writing
- Also will throw in some discussion questions
  - Not in your lecture notes! Gray backgrounds.
- Will use board when appropriate

Textbook



- John L. Hennessy and David Patterson, “Computer Architecture: A Quantitative Approach”, 5th edition (Morgan Kaufmann), 2011
  - 5th edition should match lectures a little better.
  - Don’t get the 4th edition
  - You do need the CD (but if you don’t have it your classmates will)
  - Problem sets are good but not always well-written

Grading

- 3 segments to class:
  - Instruction level parallelism
  - Thread level parallelism
  - Data level parallelism

- Homework: 10% (3)
- Projects: 30% (3)
- Midterms: 15% each
- Final: 30% (cumulative)
- Goals:
  - Reduce homework dependence
  - Projects are important!

Course Philosophy

- Ninth time I'm teaching this class
- Here's what I hope you'll get for any given technique in this class:
  - Understand *what* that technique is
  - Understand *why* that technique is important
  - Not understand (necessarily) *how* that technique is implemented

Important Dates

- Midterms
  - Th 11 May (concentrates on instruction-level parallelism)
  - Th 1 June (concentrates on thread-level parallelism)
- Final
  - Fri 9 June (8–10a)
  - Cumulative
- Exams are open-book, open-note
- Makeups on midterms or final are oral only

Homework Turn-In

- Homework goes in 2131 Kemper
- Homework is due at 10 am BEFORE CLASS
- DO NOT BRING HOMEWORK TO CLASS
- Written homework must be done individually
- Homework and exam solutions will be handed out in class
    - We'll try to make solutions ASAP after due date
    - Please do not pass these solutions on to other students
    - Use of online solutions to homework/projects is cheating

Project Turnin

- Projects will be turned in electronically (Canvas)
- Project deliverable will be a *writeup*
  - Ability to communicate is important!
  - Writeups will be short (1–2 pages)
  - PDF
- Projects are individual

What is cheating?

- Cheating is claiming credit for work that is not your own.
- Cheating is disobeying or subverting instructions of the instructional staff.
    - Homework deadlines, online solutions, etc.
- It is OK to work in (small) groups on homework.
    - All work you turn in must be 100% yours, and you must be able to explain all of it.
    - Give proper credit if credit is due.

Things You Should Do

- *Ask questions!*
  - Especially when things aren’t clear
- Give feedback!
  - Email or face-to-face
  - Tell me what I’m doing poorly
  - Tell me what I’m doing well
  - Tell the TA too
- Start projects early

Things You Shouldn’t Do

- Cheat
- Skip class (I’ll know when you’re not there!)
- Be late for class
- Read the paper in class
- Open your laptop in class unless there’s a good reason
- Allow your cell phone to ring in class

Getting the Grade You Want

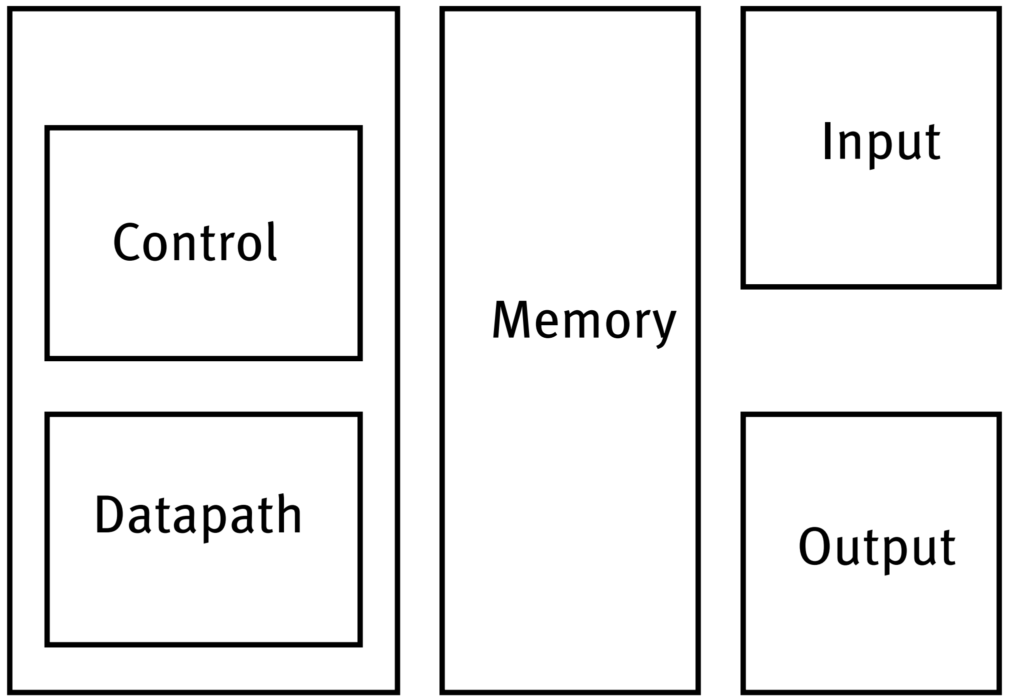
- Come to class!
- Ask questions in class when you don’t understand
- Come to office hours (mine and Terry’s)
- Start the hw and projects early
- Use the projects as a vehicle for learning
- Understand the course material

My Expectations

- This class was hard
- I learned a lot
- The work I did in this class was worthwhile
- The instructor was fair
- The instructor was effective
- The instructor cared about my learning

Review

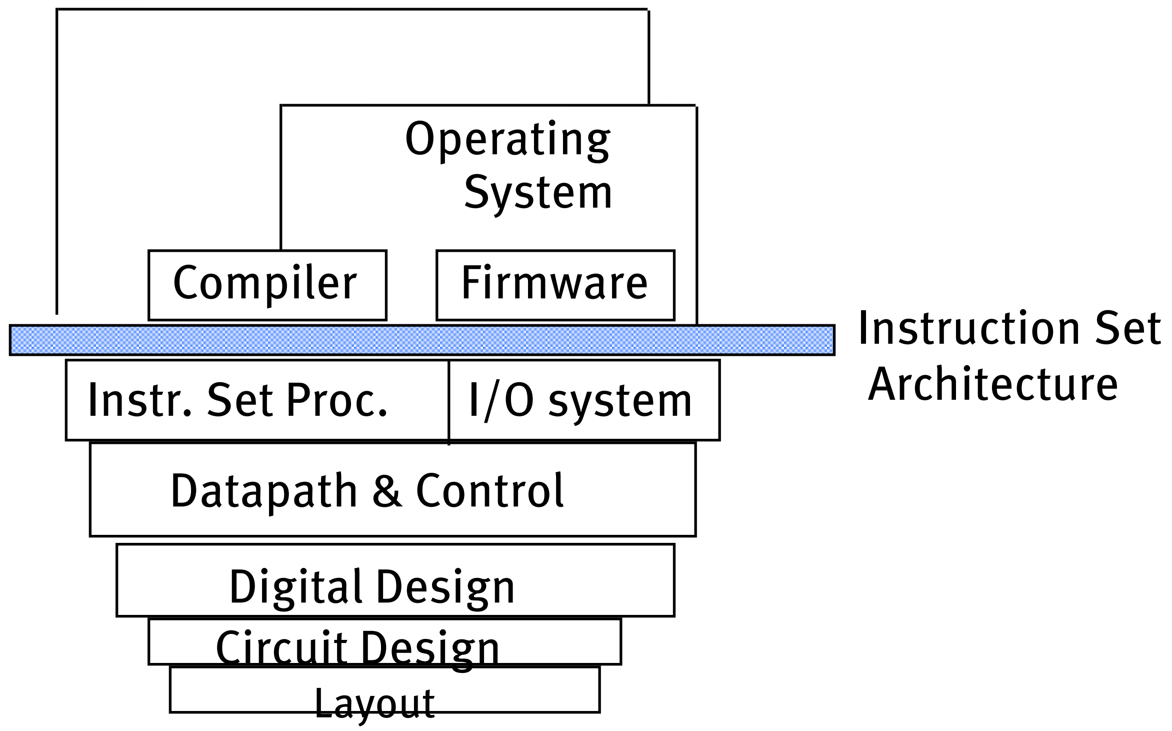
The Big Picture



- Since 1946 all computers have had 5 components

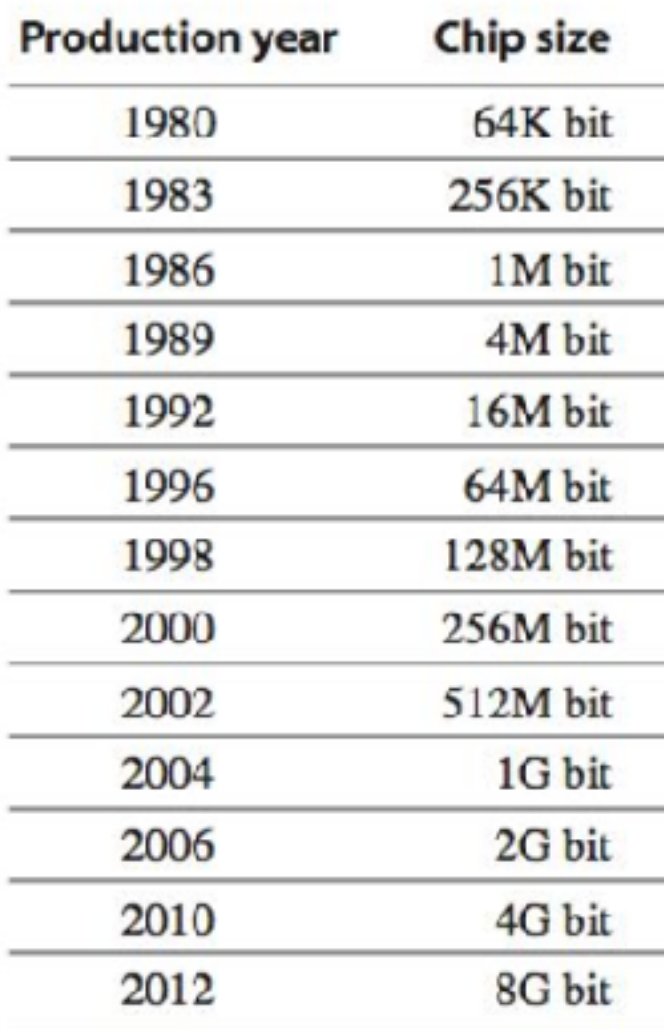
What is “Computer Architecture”?

- Coordination of many levels of abstraction
- Under a rapidly changing set of forces
- Design, Measurement, and Evaluation

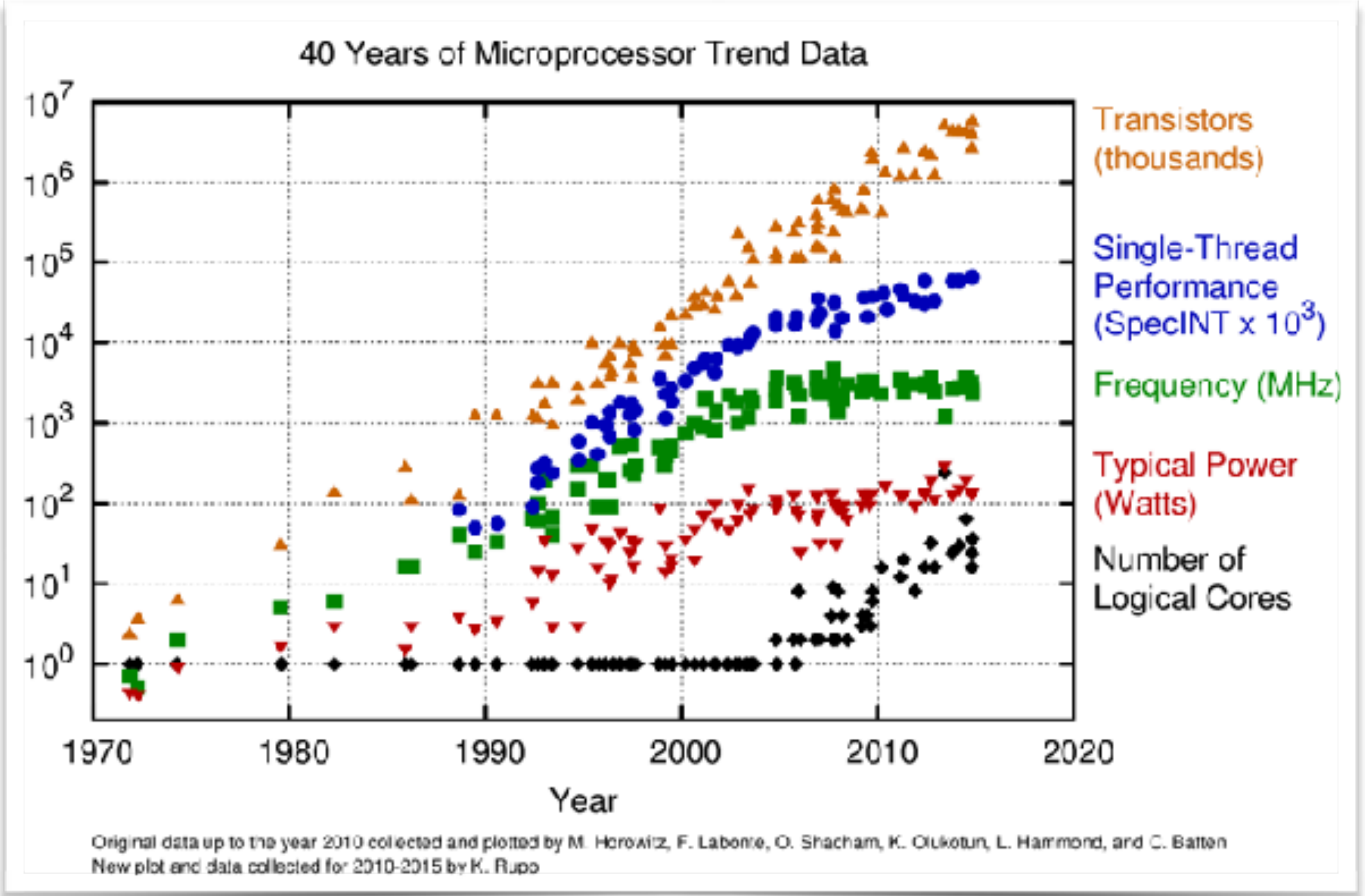


Technology

DRAM chip capacity



Production year	Chip size
1980	64K bit
1983	256K bit
1986	1M bit
1989	4M bit
1992	16M bit
1996	64M bit
1998	128M bit
2000	256M bit
2002	512M bit
2004	1G bit
2006	2G bit
2010	4G bit
2012	8G bit



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 2010-2015 by K. Rupp

The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software
By Herb Sutter, Dr Dobbs Journal 30(3), March 2005. Data through Aug 2009. Newest data from https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

- In ~1985 the single-chip processor (32-bit) and the single-board computer emerged
  - => workstations, personal computers, multiprocessors have been riding this wave since

Technology rates of change

- Processor
  - logic capacity: about 30% per year
  - clock rate: about 20% per year
- Memory
  - DRAM capacity: about 60% per year (4x every 3 years)
  - Memory speed: about 10% per year
  - Cost per bit: improves about 25% per year
- Disk
  - capacity: about 60% per year
  - Total use of data: 100% per 9 months!
- Network Bandwidth increasing more than 100% per year!

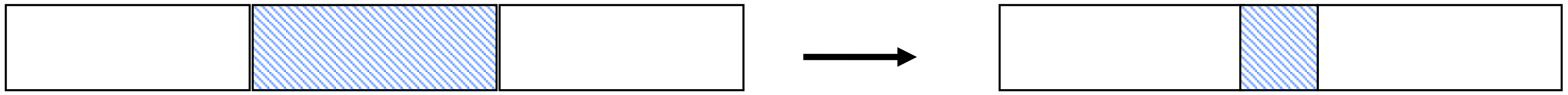
The Performance Equation

- Time = Clock Speed * CPI * Instruction Count
  - = seconds/cycle * cycles/instr * instrs/program
  - => seconds/program
- “The only reliable measure of computer performance is time.”

Amdahl's Law

- Speedup due to enhancement E:

$$\text{Speedup}(E) = \frac{\text{Execution Time without } E}{\text{Execution Time with } E} = \frac{\text{Performance with } E}{\text{Performance without } E}$$



- Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected:

$$\text{Execution time (with } E) = ((1-F) + F/S) \cdot \text{Execution time (without } E)$$

$$\text{Speedup (with } E) = \frac{1}{(1-F) + F/S}$$

- Design Principle: Make the common case fast!

Amdahl's Law example

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\begin{aligned}\text{Speedup}_{\text{overall}} &= \frac{1}{(1-\text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} \\ &= \frac{1}{(1-0.4)+\frac{0.4}{10}} = \frac{1}{0.64} = 1.56\end{aligned}$$

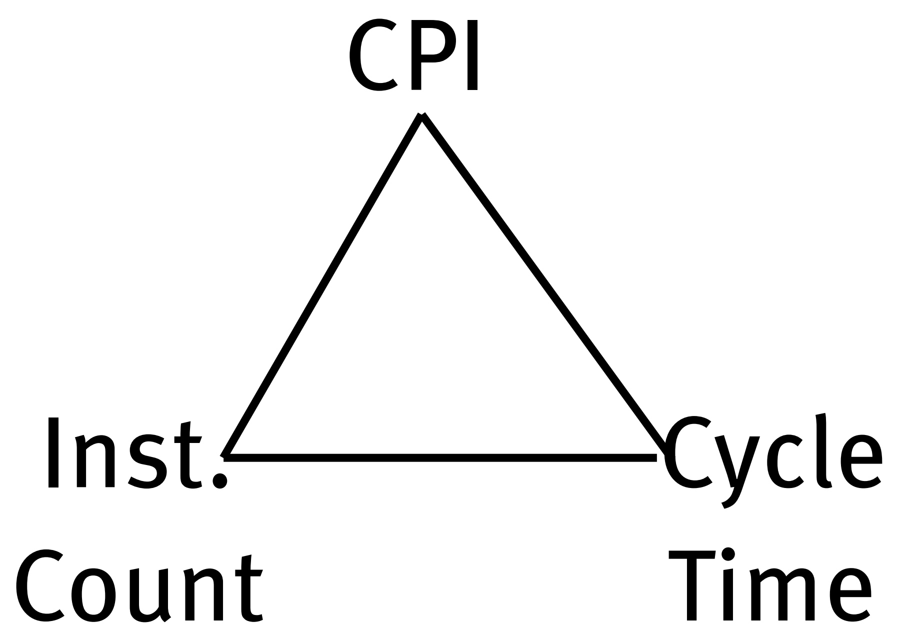
- Apparently, it's human nature to be attracted by 10X faster, vs. keeping in perspective it's just 1.6X faster

Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	• less representative
• easy to run, early in design cycle	Small “kernel” benchmarks	• easy to “fool”
• identify peak capability and potential bottlenecks	Microbenchmarks	• “peak” may be a long way from application performance

Evaluating Instruction Sets

- Design-time Metrics:
    - Can it be implemented, in how long, at what cost?
    - Can it be programmed? Ease of compilation?
- Static Metrics:
    - How many bytes does the program occupy in memory?
- Dynamic Metrics:
    - How many instructions are executed?
    - How many bytes does the processor fetch to execute the program?
    - How many clocks are required per instruction?
    - How "lean" a clock is practical?
- Best Metric: Time to execute the program!



MIPS Instruction Set

- 32-bit fixed format inst (3 formats)
- 32 32-bit GPR (R0 contains zero); 32 FP registers (and HI LO)
  - partitioned by software convention
- 3-address, reg-reg arithmetic instr.
- Single address mode for load/store: base+displacement
  - no indirection, scaled
- 16-bit immediate plus LUI
- Simple branch conditions
  - compare against zero or two registers for =,≠
  - no integer condition codes
- Delayed branch
  - execute instruction after a branch (or jump) even if the branch is taken
  - Compiler can fill branch delay slot ~50% of the time

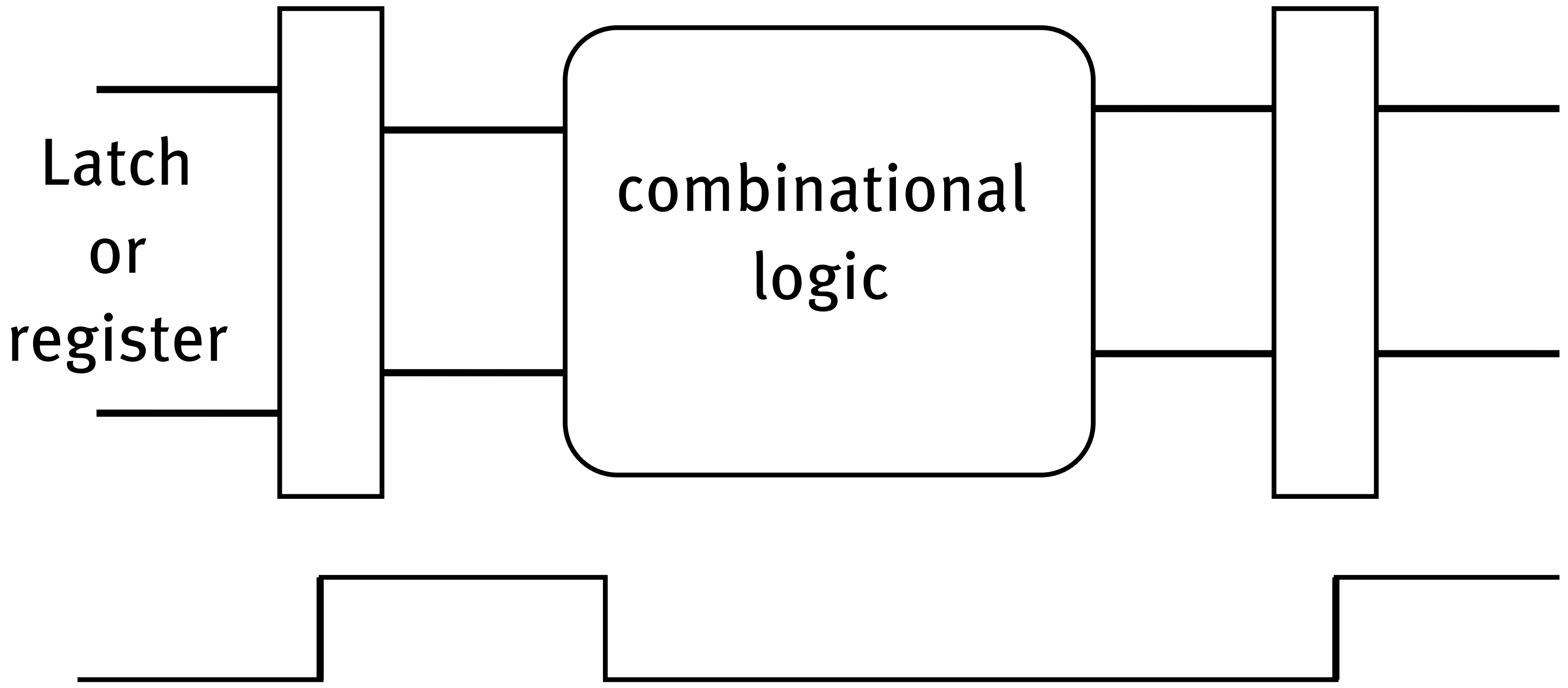
RISC Philosophy

- Instructions all same size
- Small number of opcodes (small opcode space)
- Opcode in same place for every instruction
- Simple memory addressing
- Instructions that manipulate data don’t manipulate memory, and vice versa
- Minimize memory references by providing ample registers

Computer Arithmetic

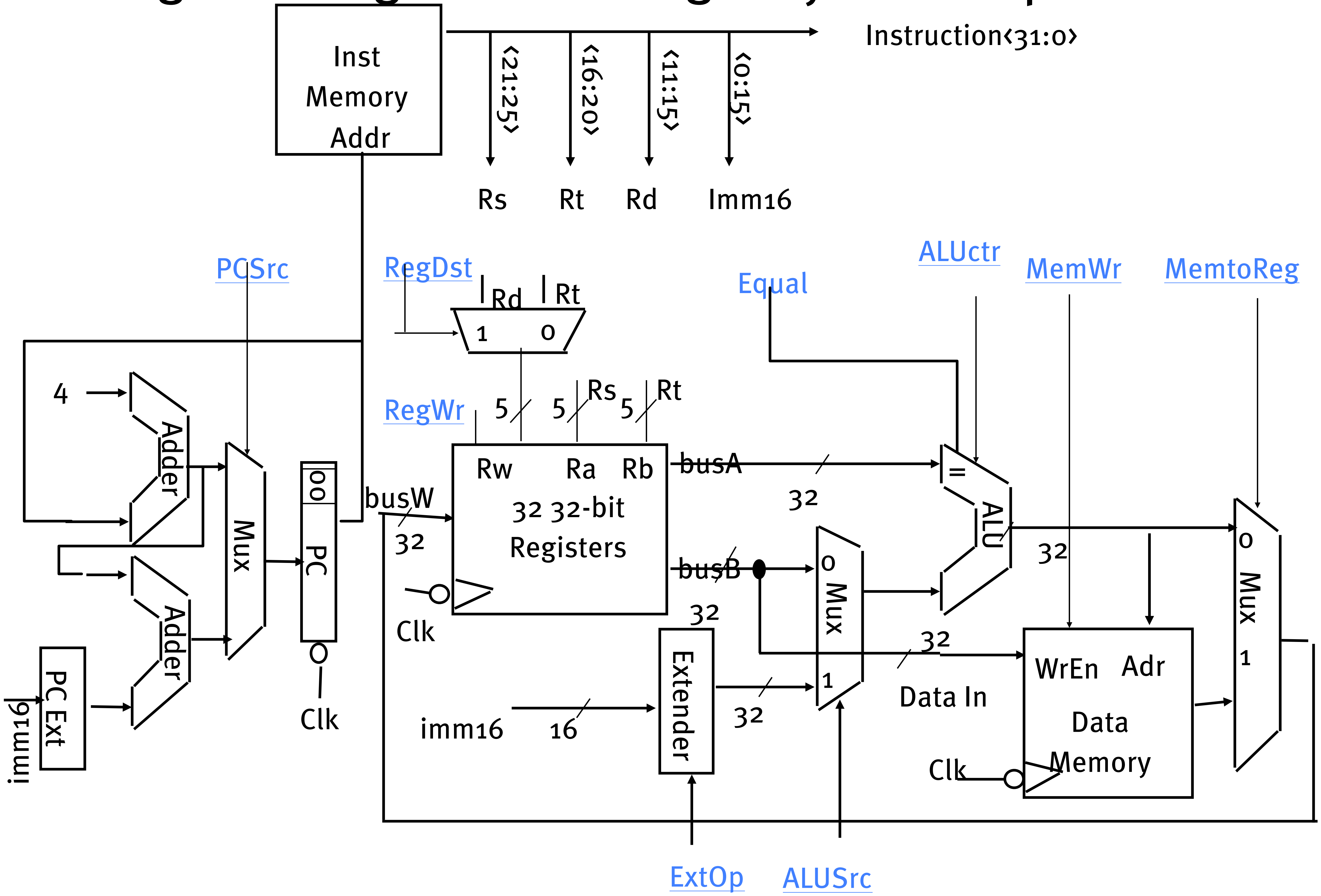
- Bits have no inherent meaning: operations determine whether really ASCII characters, integers, floating point numbers
- 2's complement
- Hardware algorithms for arithmetic:
  - Carry lookahead/carry save addition (parallelism!)
- Floating point

What's a Clock Cycle?

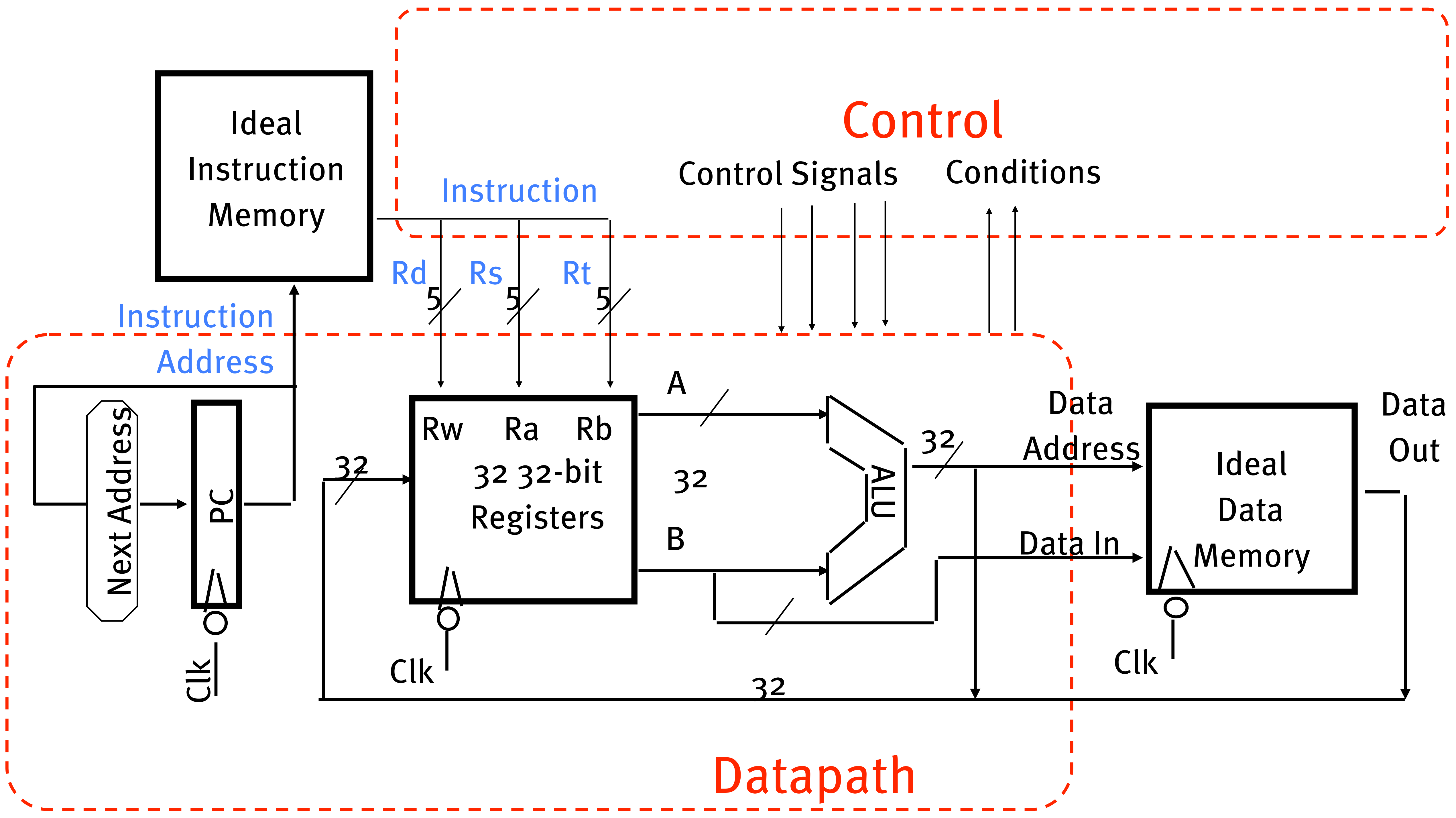


- Old days: ~10 levels of gates
- Today: determined by numerous time-of-flight issues + gate delays
  - clock propagation, wire lengths, drivers

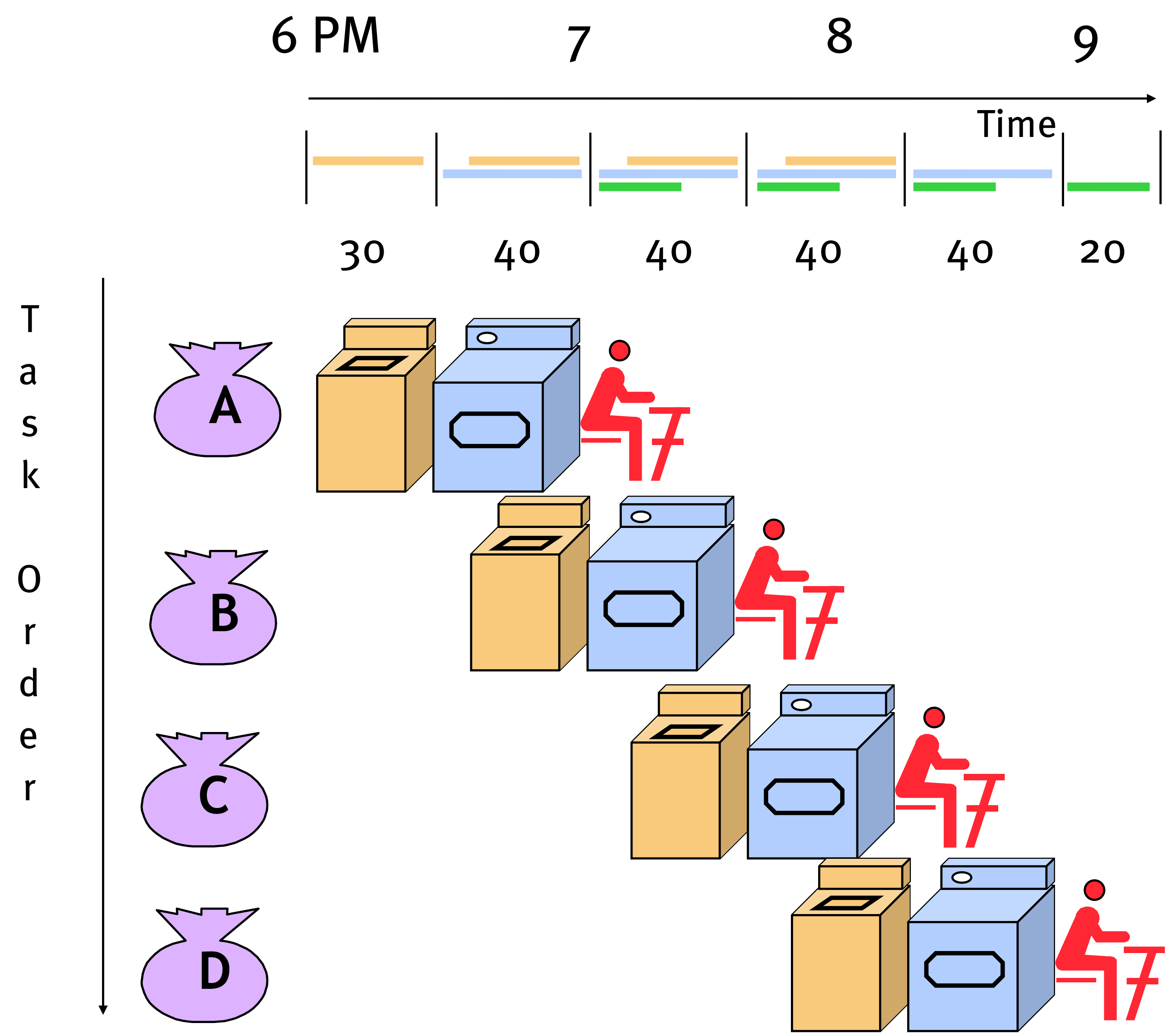
Putting it All Together: A Single Cycle Datapath



An Abstract View of Single Cycle

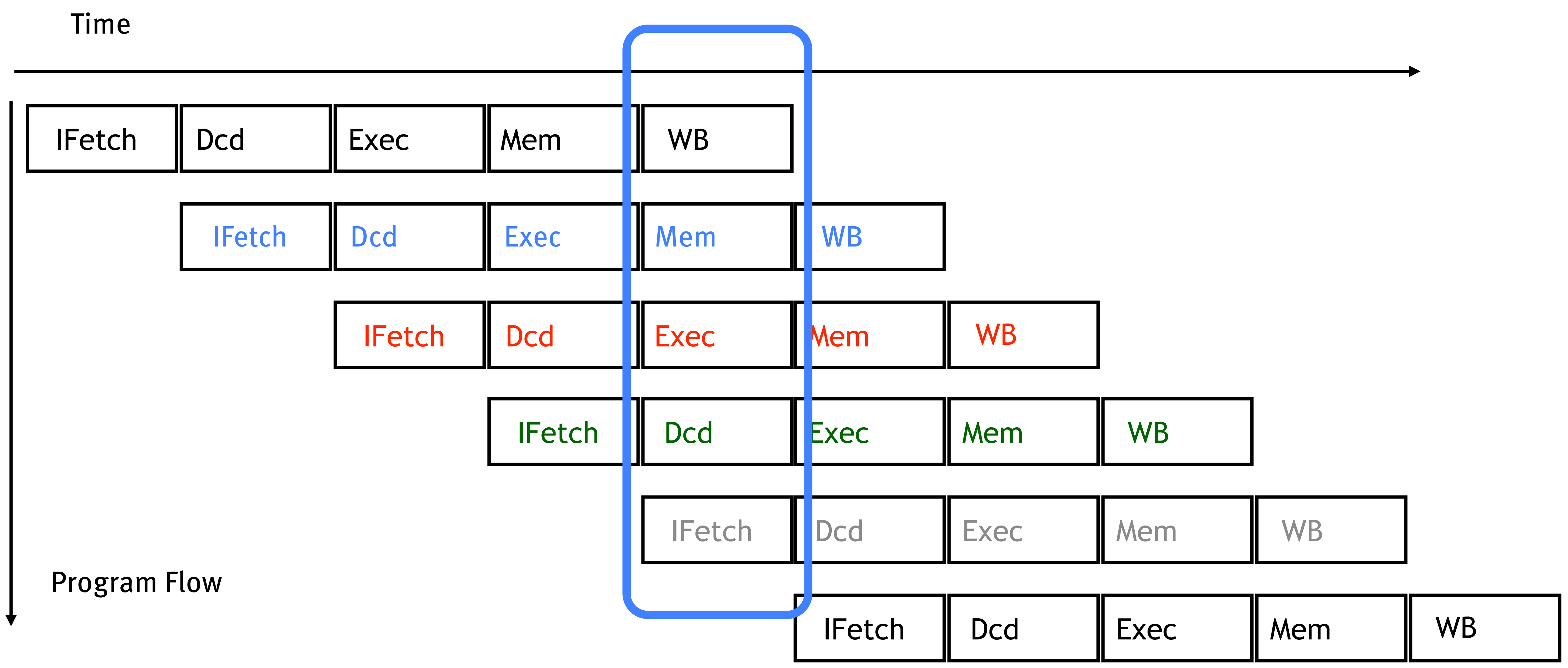


Pipelining Overview

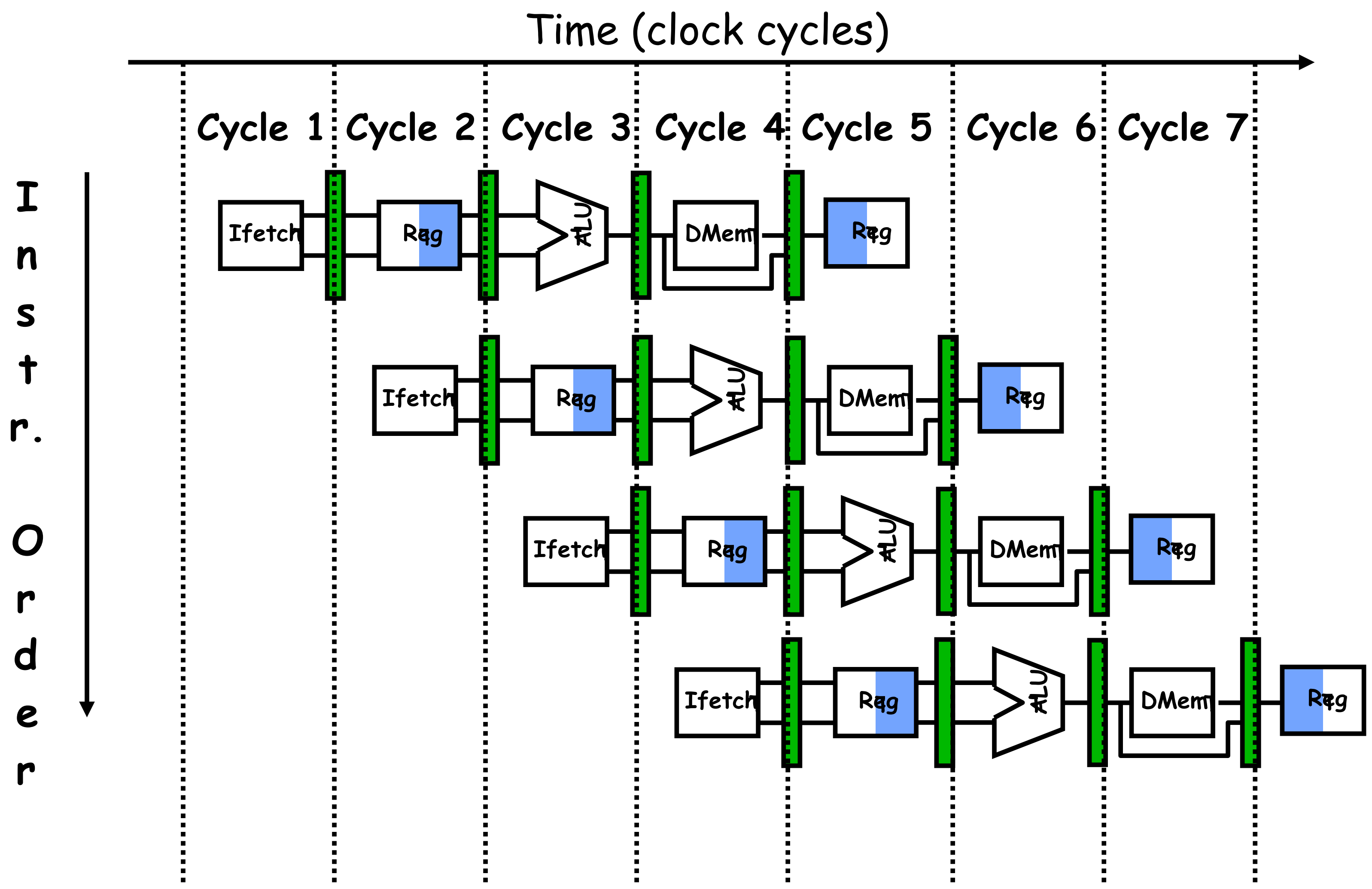


- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for dependencies

Conventional Pipelined Execution Representation



Why Pipeline? Because we can!

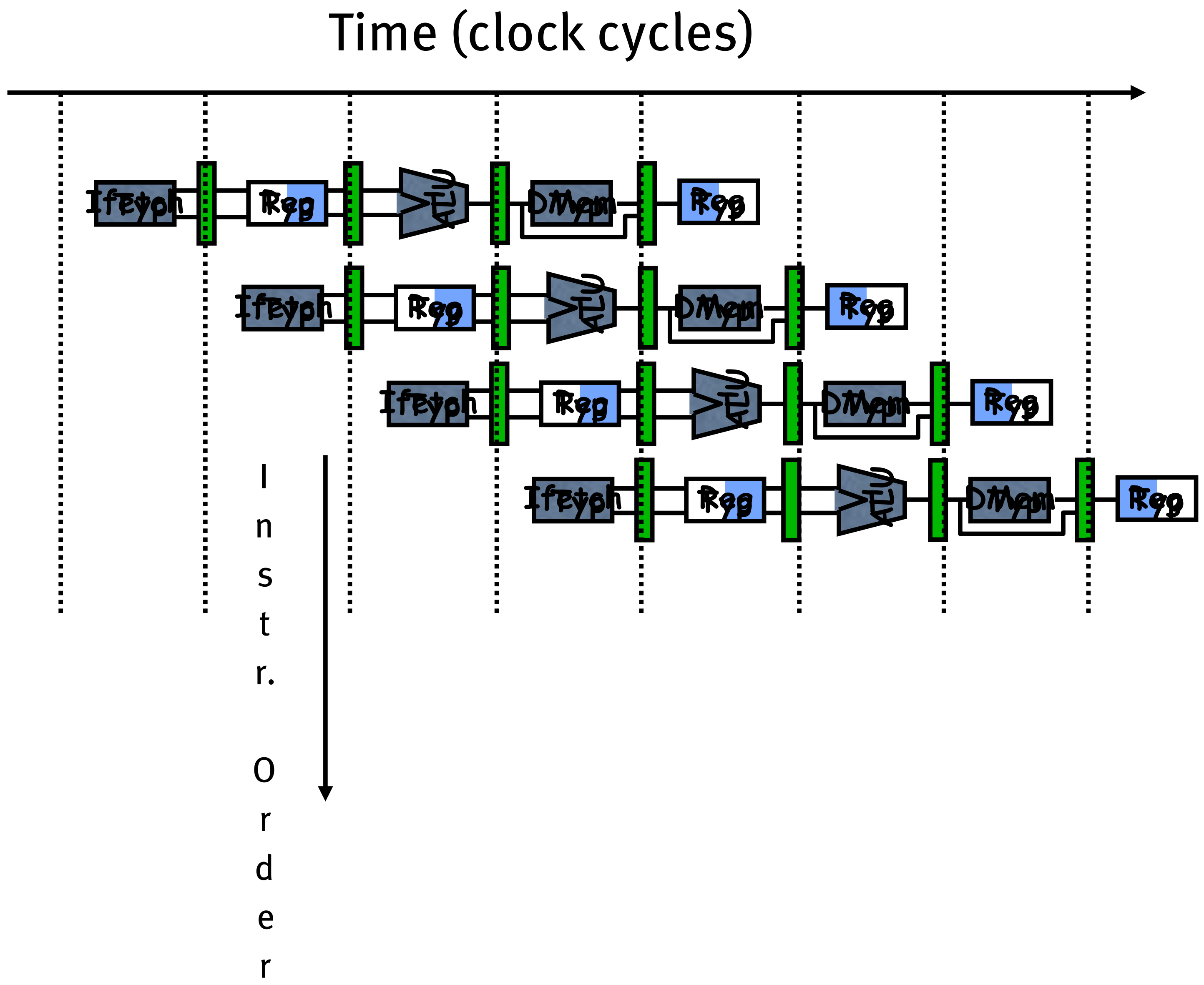


Why is MIPS great for pipelining?

- All MIPS instructions same length
- Source registers located in same place for every instruction
  - Overlap register fetch and instruction decode
- Simple memory operations
  - MIPS: execute calculates memory address, memory load/store in next stage
  - X86: can operate on result of load: execute calculates memory address, memory load/store in next stage, THEN ALU stage afterwards
- All instructions aligned in memory — 1 access for each instruction

Limits to pipelining

- Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: attempt to use the same hardware to do two different things at once
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).



Focus on the Common Case

- Common sense guides computer design
    - Since it's engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
    - e.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
    - e.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
    - e.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more for the common case of no overflow
    - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster?
  $\Rightarrow$ Amdahl's Law

Pipeline Summary

- Simple 5-stage pipeline: F D E M W
- Pipelines pass control information down the pipe just as data moves down pipe
- Resolve data hazards through forwarding.
- Forwarding/Stalls handled by local control
- MIPS I instruction set architecture made pipeline visible (delayed branch, delayed load)
- More performance from deeper pipelines, parallelism

Design Philosophies Change

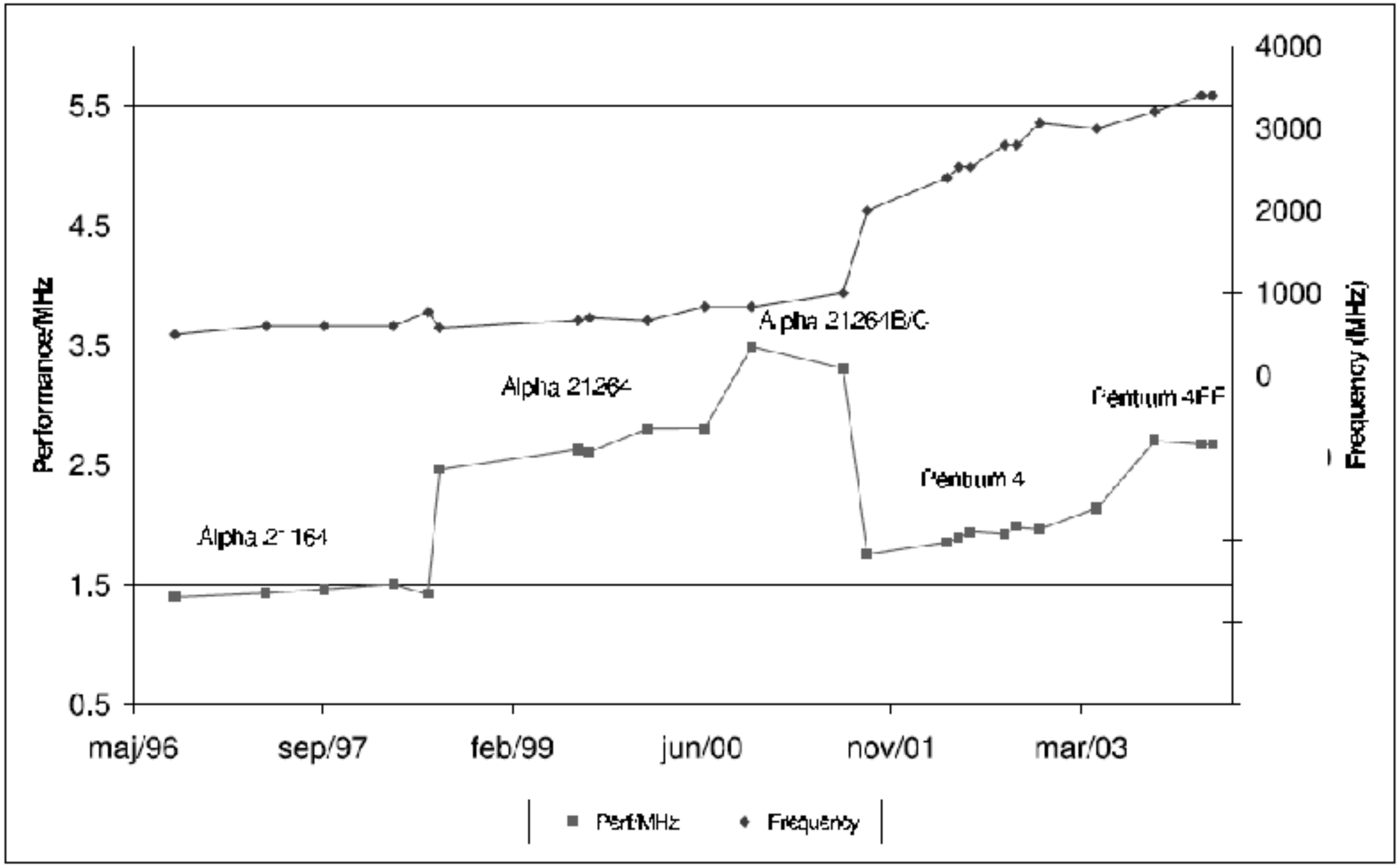
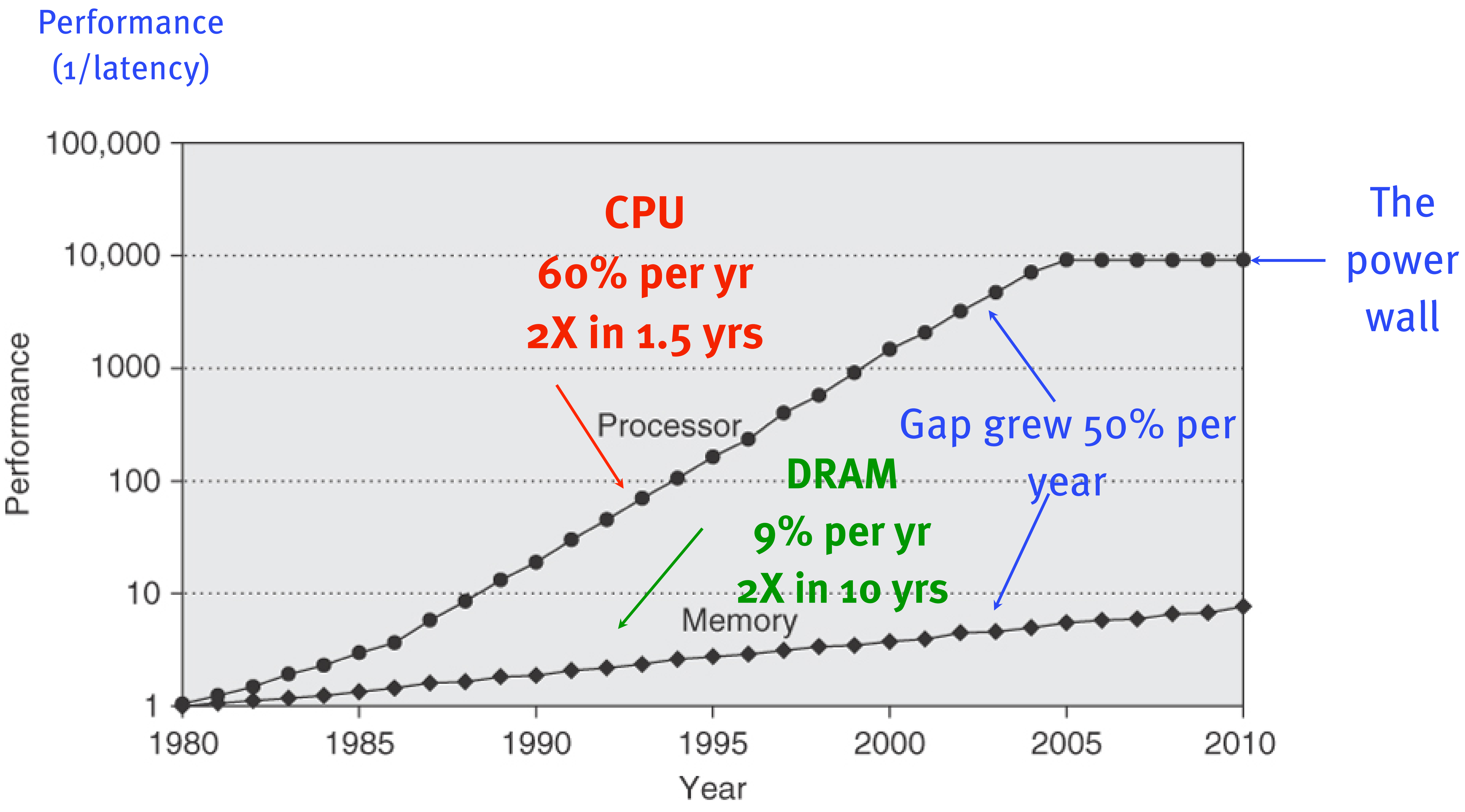


Fig. 3. Performance per MHz 1993-2004.

- Ekman et al., "An In-Depth Look at Computer Performance Growth"

Why Do We Care About the Memory Hierarchy?



Levels of the Memory Hierarchy

Capacity
Access Time
Cost

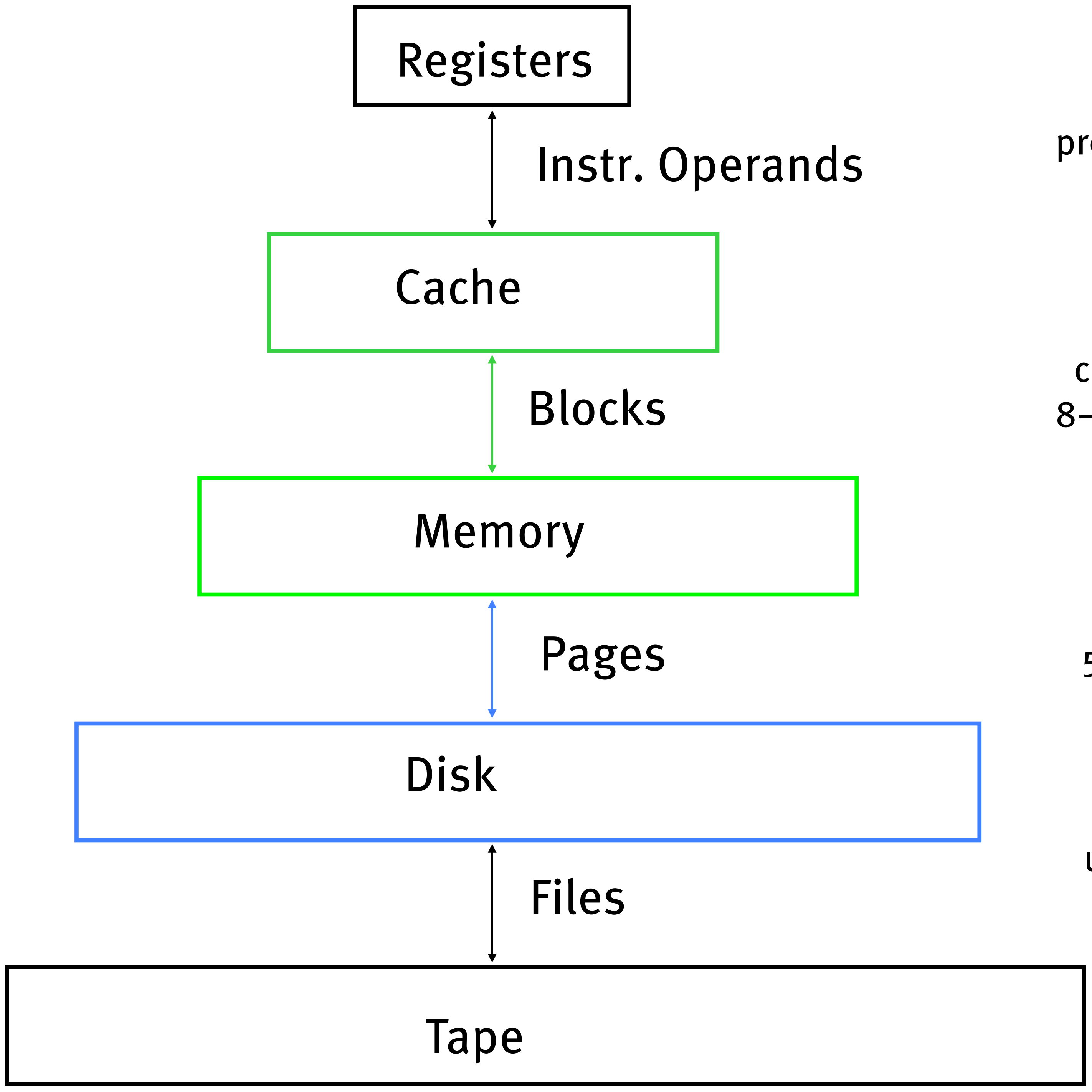
CPU Registers
100s Bytes
< 2 ns

Cache
K Bytes SRAM
2–100 ns
$.01–.001/bit

Main Memory
M Bytes DRAM
100ns–1us
$.01–.001

Disk
G Bytes
ms
10^{-3}–10^{-4} cents

Tape
infinite
sec-min
10^{-6}



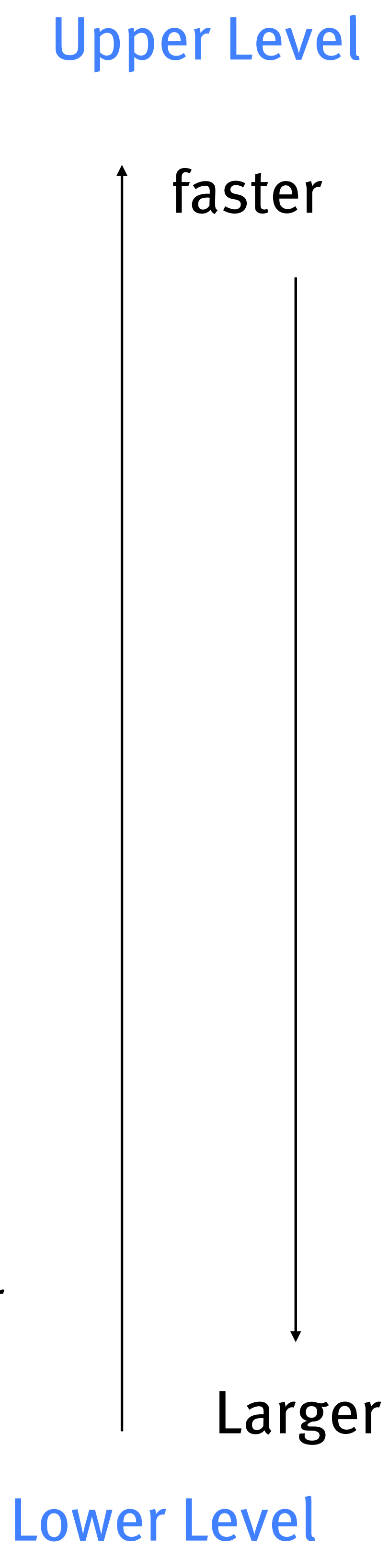
Staging
Xfer Unit

prog./compiler
1–8 bytes

cache cntl
8–128 bytes

OS
512–4K bytes

user/operator
Mbytes



Memory Hierarchy

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
  - Capacity Misses: increase cache size

Looking Forward

“Fast enough” ?

- 10 years ago: buy fastest computer you can afford
- Today: No longer the case
- No killer apps!
    - 3D graphics, full-screen video, Internet, speech …
- You should invent some.

Gelsinger’s Law

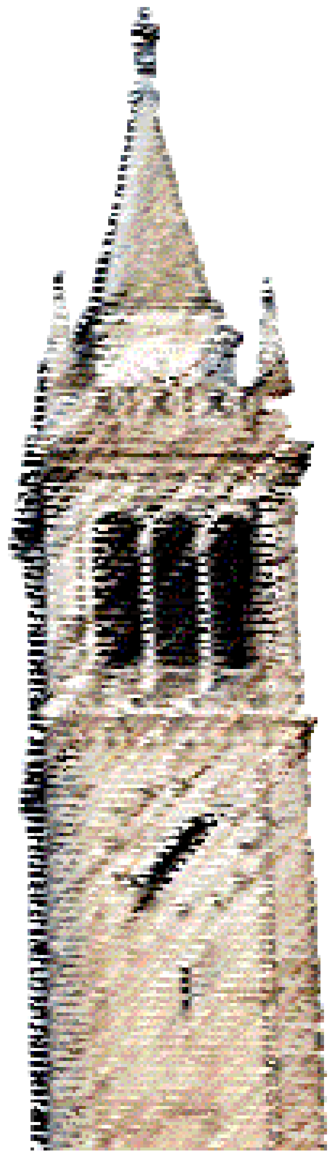
- “New generation microarchitectures use twice as many transistors for a 40% increase in performance.”

Cost of Fabs

- Rock’s Law: Cost of fabs double every 4 years
- $14B for Samsung’s “18 Line” Pyeongtaek fab (2017 opening)
  - Rise of fabless design houses
  - Rise of for-hire fabs (TSMC, etc.)
  - $14B fab means $28B in revenue is required

Looking To The Future

The Landscape of Parallel Computing Research: A View from Berkeley



Krste Asanovic
Ras Bodik
Bryan Christopher Catanzaro
Joseph James Gebis
Parry Husbands
Kurt Keutzer
David A. Patterson
William Lester Plishker
John Shalf
Samuel Webb Williams
Katherine A. Yelick

Electrical Engineering and Computer Sciences
University of California at Berkeley

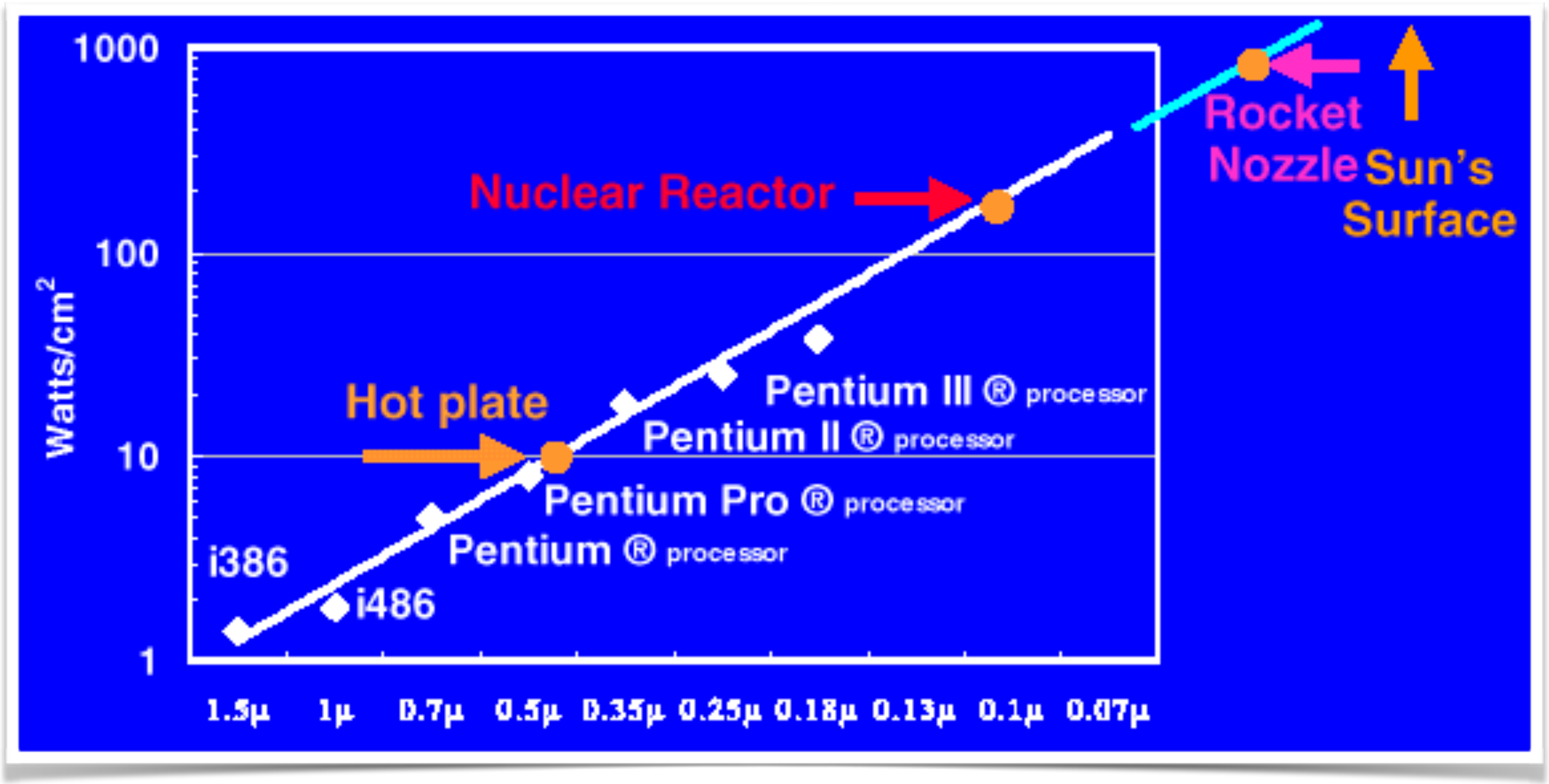
Technical Report No. UCB/EECS-2006-183
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

December 18, 2006

Power

- Old CW: Power is free, but transistors are expensive.
- New CW is the “Power Wall”: Power is expensive, but transistors are free. We can put more transistors on a chip than we have power to turn on.

Power Limits Performance



- Bob says: We're pushing perf & clk rates too hard

[courtesy of Bob Colwell]

Compute vs. Memory

- Old CW: Multiply is slow, but load and store is fast.
- New CW is the *Memory Wall*. Load and store is slow, but multiply is fast. Modern microprocessors can take 200 clocks to access Dynamic Random Access Memory (DRAM), but even floating-point multiplies may take only four clock cycles.

Computation vs. Communication

- 20 years ago: computation expensive, wires free
  - To first order: ignore wire delay
- Light moves 1 foot/ns in vacuum
  - Wires are also getting thinner
  - Wire delay now significant even on chip!
- Moore's Law implies:
  - Computation gets cheaper
  - Speed of light doesn't change
  - Compute don't communicate!

Reliability and Test

- Design team size growing at Moore’s Law rates
- Processors:
  - Getting larger
  - Getting more complicated
- How to test processors?
- How to prove processor designs correct?

Intel Directions

- (Intel Developer Forum, September 2004)
- "What Intel announced at this IDF was no less than a total rethinking of their approach to microprocessors."
  - Moore's-Law-driven performance scaling will come not from increases not in MHz ratings but in machine width.
    - Power wall
    - Threading
    - MIPS/watt instead of MIPS
  - Datasets are growing in size, and so are the network pipes that connect those datasets.
    - Intel claims "doubling of digital data every 18 months"
    - More integration? WiMax?

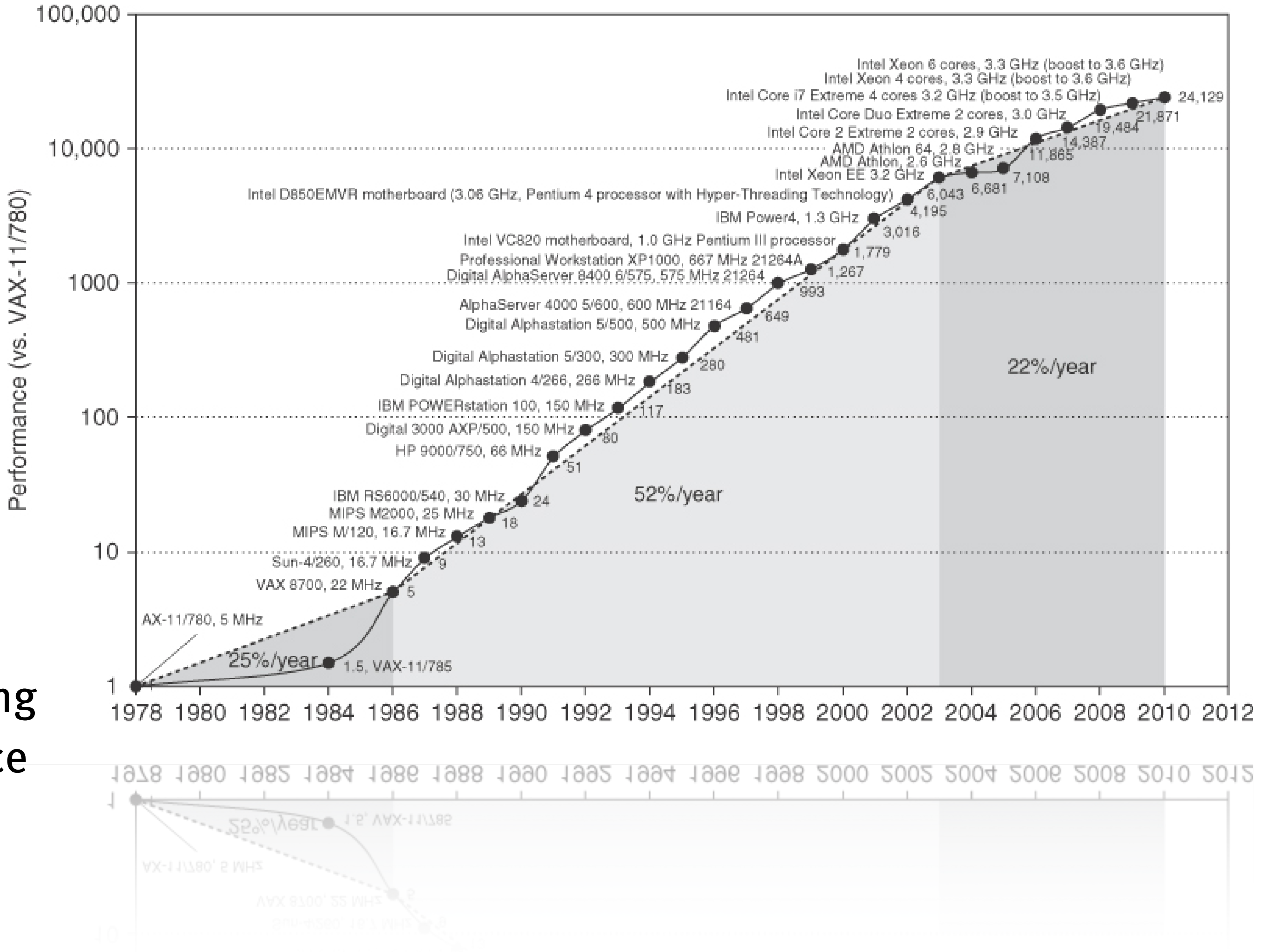
http://arstechnica.com/articles/paedia/cpu/intel-future.ars/1

Instruction-Level Parallelism

- Old CW: We can reveal more instruction-level parallelism (ILP) via compilers and architecture innovation. Examples from the past include branch prediction, out-of-order execution, speculation, and Very Long Instruction Word systems.
    - This is our first three weeks.
- New CW is the “ILP wall”: There are diminishing returns on finding more ILP.

Uniprocessor Performance

- Old CW: Uniprocessor performance doubles every 18 months.
- New CW is Power Wall + Memory Wall + ILP Wall = Brick Wall. In 2006, performance is a factor of three below the traditional doubling every 18 months that we enjoyed between 1986 and 2002. The doubling of uniprocessor performance may now take 5 years.



Moore’s Law reinterpreted

- Number of cores per chip can double every two years
- Clock speed will not increase (possibly decrease)
- Need to deal with systems with millions of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism
- But Moore’s Law is not forever… industry consortium predicts end in 2021

Courtesy Kathy Yelick

Why EEC 171?

- Old CW: Don’t bother parallelizing your application, as you can just wait a little while and run it on a much faster sequential computer.
- New CW: It will be a very long wait for a faster sequential computer.
- Old CW: Increasing clock frequency is the primary method of improving processor performance.
- New CW: Increasing parallelism is the primary method of improving processor performance.
- Old CW: Less than linear scaling for a multiprocessor application is failure.
- New CW: Given the switch to parallel computing, any speedup via parallelism is a success.