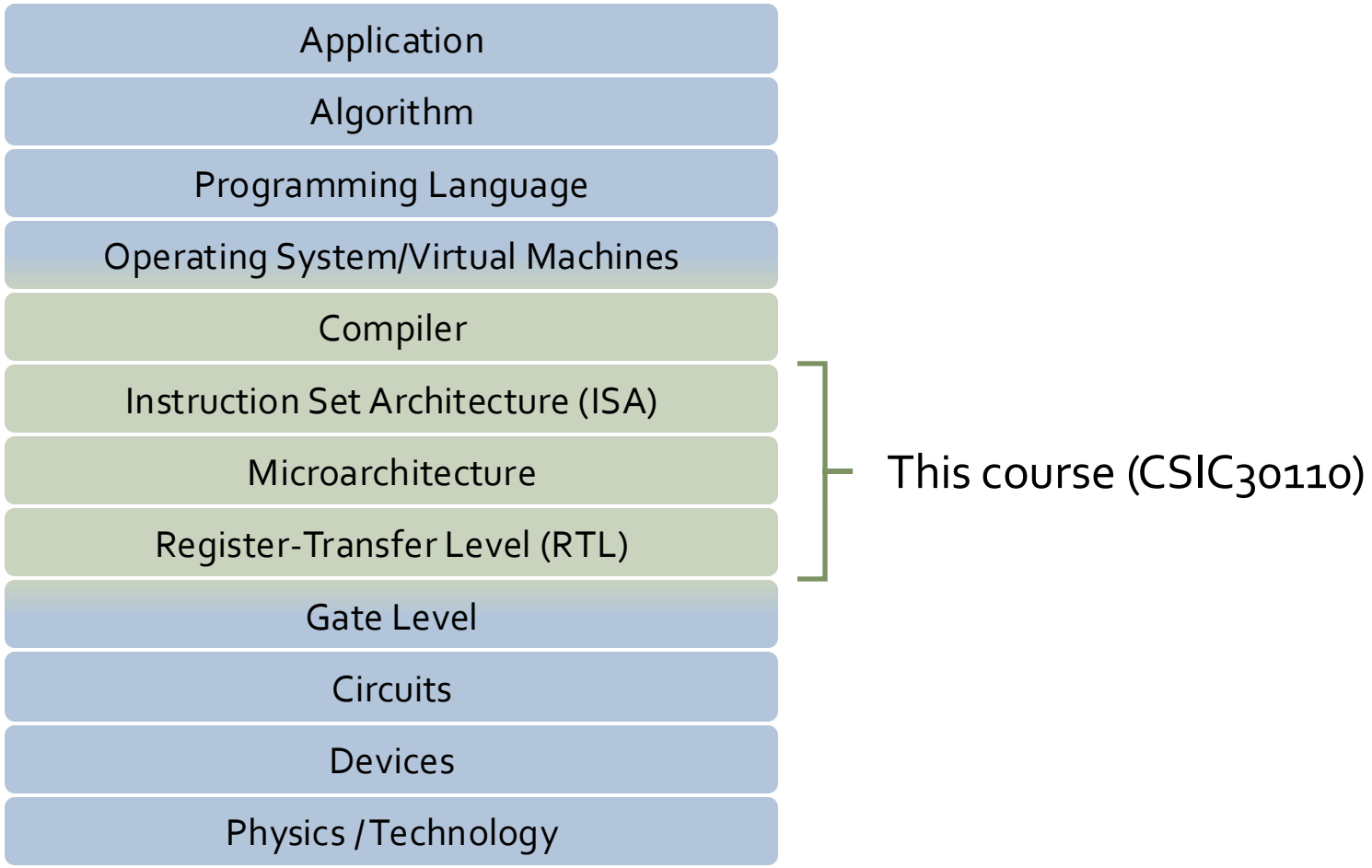
Computer Architecture
Pipeline and Cache Review

Ting-Jung Chang

NYCU CS

Recap


Application
Algorithm
Programming Language
Operating System/Virtual Machines
Compiler
Instruction Set Architecture (ISA)
Microarchitecture
Register-Transfer Level (RTL)
Gate Level
Circuits
Devices
Physics / Technology
This course (CSIC30110)

Recap: Real World Instruction Sets

Arch	Type	# Oper	# Mem	Data Size	# Regs	Addr Size	Use
Alpha	Reg-Reg	3	0	64-bit	32	64-bit	Workstation
ARM	Reg-Reg	3	0	32/64-bit	16	32/64-bit	Cell Phones, Embedded
MIPS	Reg-Reg	3	0	32/64-bit	32	32/64-bit	Workstation, Embedded
SPARC	Reg-Reg	3	0	32/64-bit	24-32	32/64-bit	Workstation
TI C6000	Reg-Reg	3	0	32-bit	32	32-bit	DSP
IBM 360	Reg-Mem	2	1	8/16/32/64-bit	16	24/31/64-bit	Mainframe
x86	Reg-Mem	2	1	8/16/32/64-bit	4/8/16	16/32/64-bit	Personal Computers, HPC
RISC-V	Reg-Reg	3	0	32/64/128-bit	32	32/64-bit	Embedded

Quick Check

- Suppose the number of registers in the ISA is halved. Which of the following statements is true?
  A. There must be more R-type instructions.
  B. The shift amount field would range from 0 to 63.
  C. I-type instructions could include 2 additional immediate bits.
  D. Fewer bits are needed to encode register specifiers in the instruction format.

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

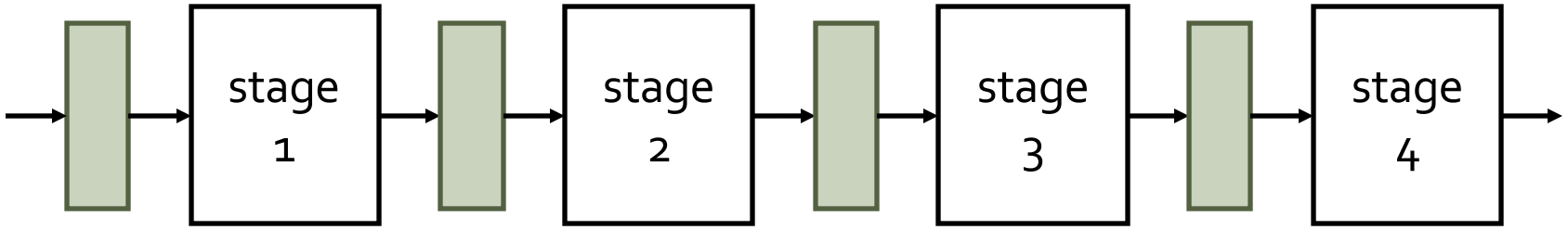
Course Administration

- Course video uploaded
- Labo, Lab1 released
- PS1 released

Agenda

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

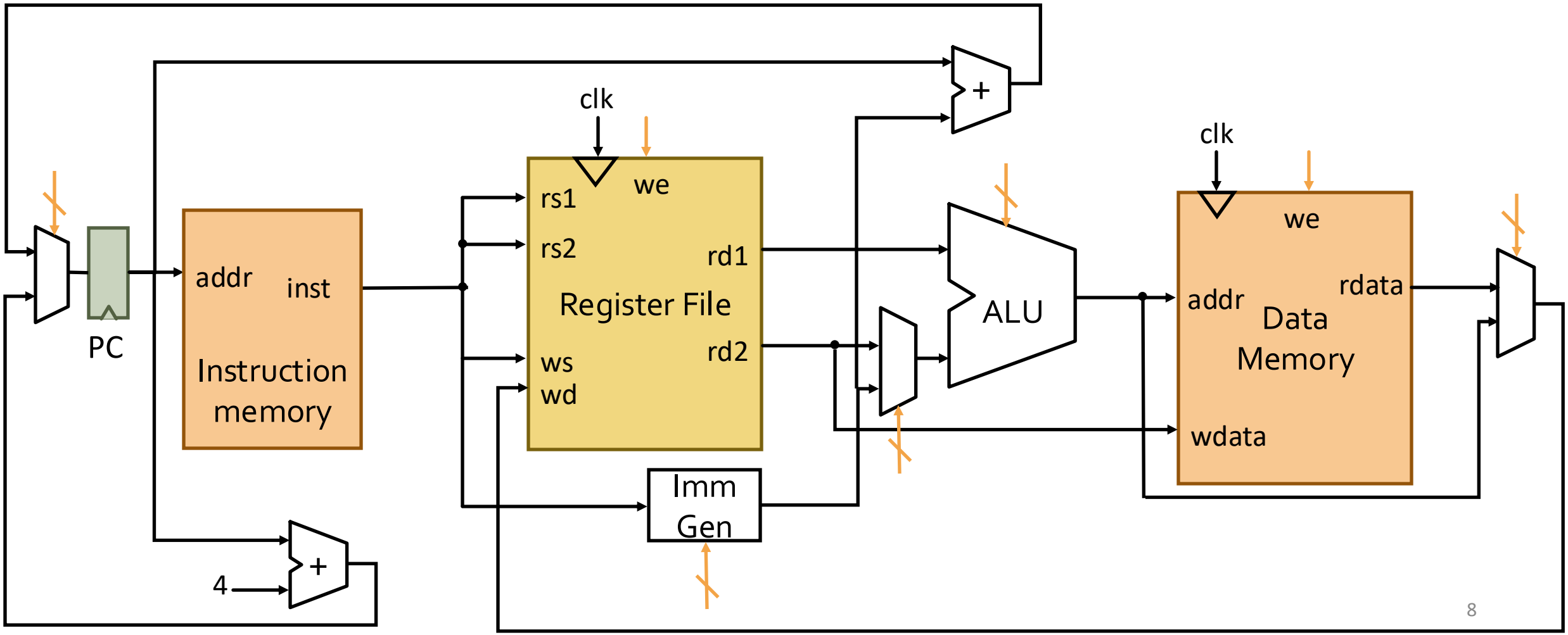
An Ideal Pipeline



- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- The scheduling of an object entering the pipeline is not affected by the objects in other stages

These conditions generally hold for industrial assembly lines, but instructions depend on each other!

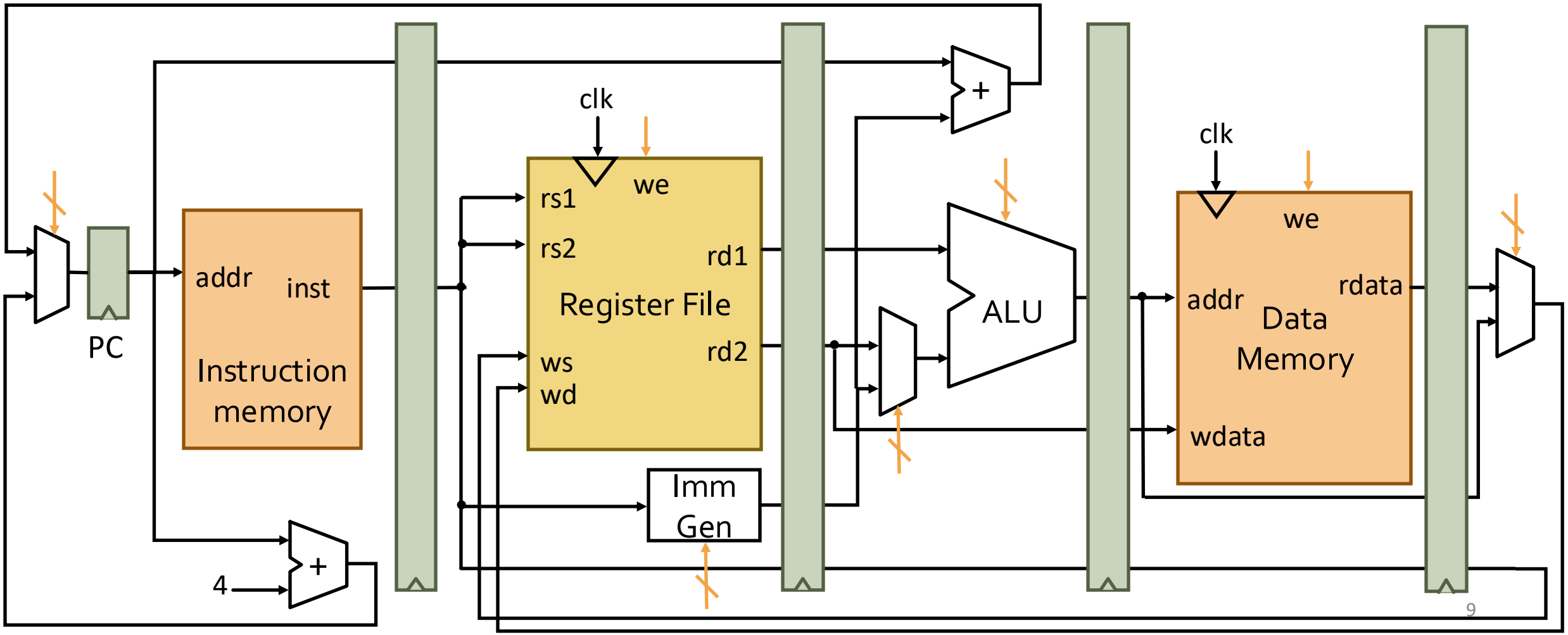
Single Cycle Datapath



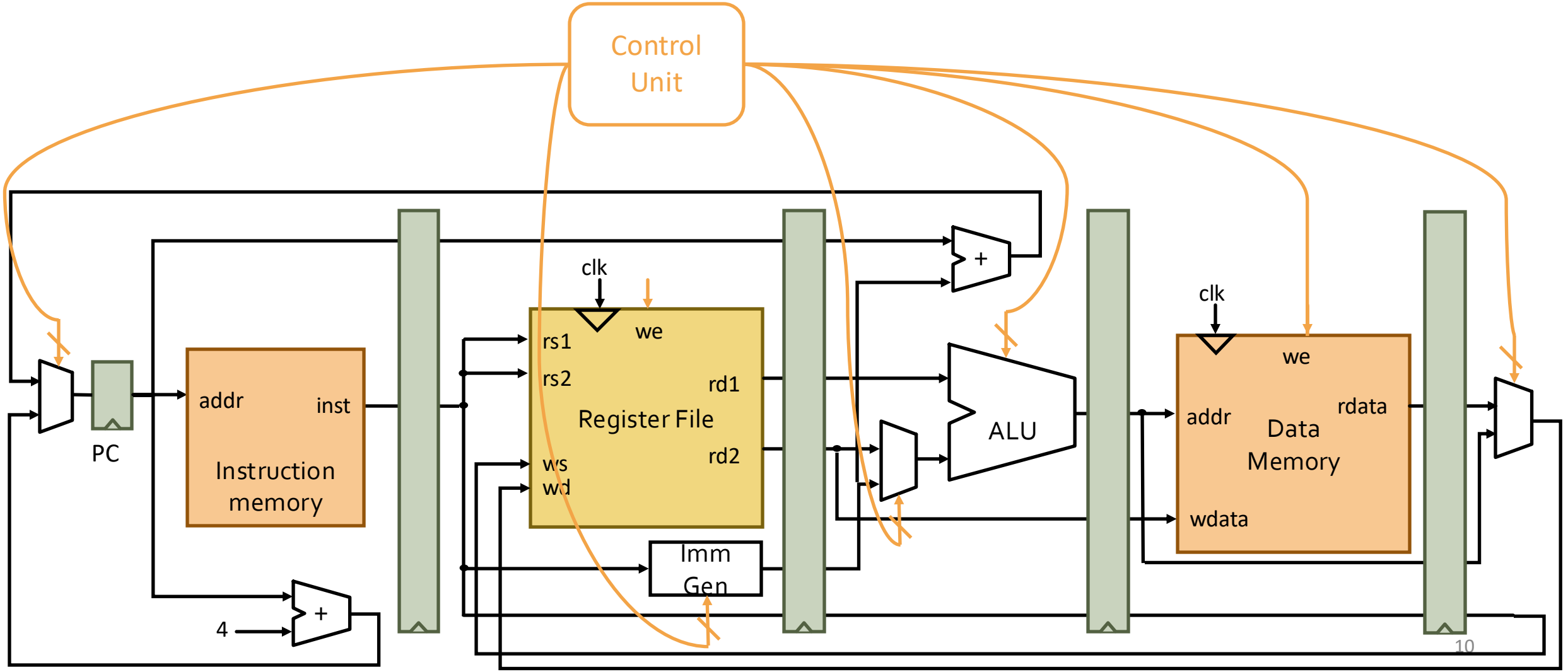
Pipelined Datapath

Clock period can be reduced by dividing the execution of an instruction into multiple cycles

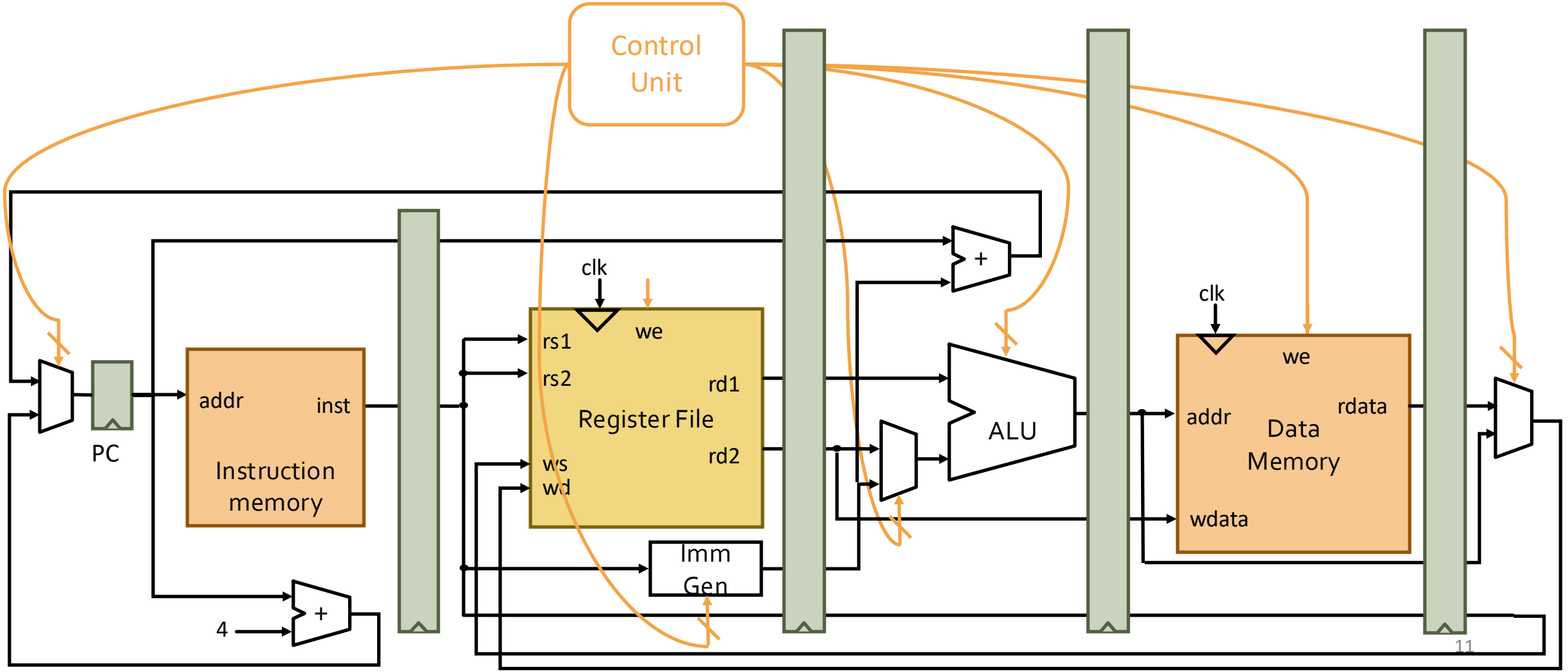
$t_C > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\}$ (= t_{DM} **probably**)



Pipelined Control



Pipelined Control



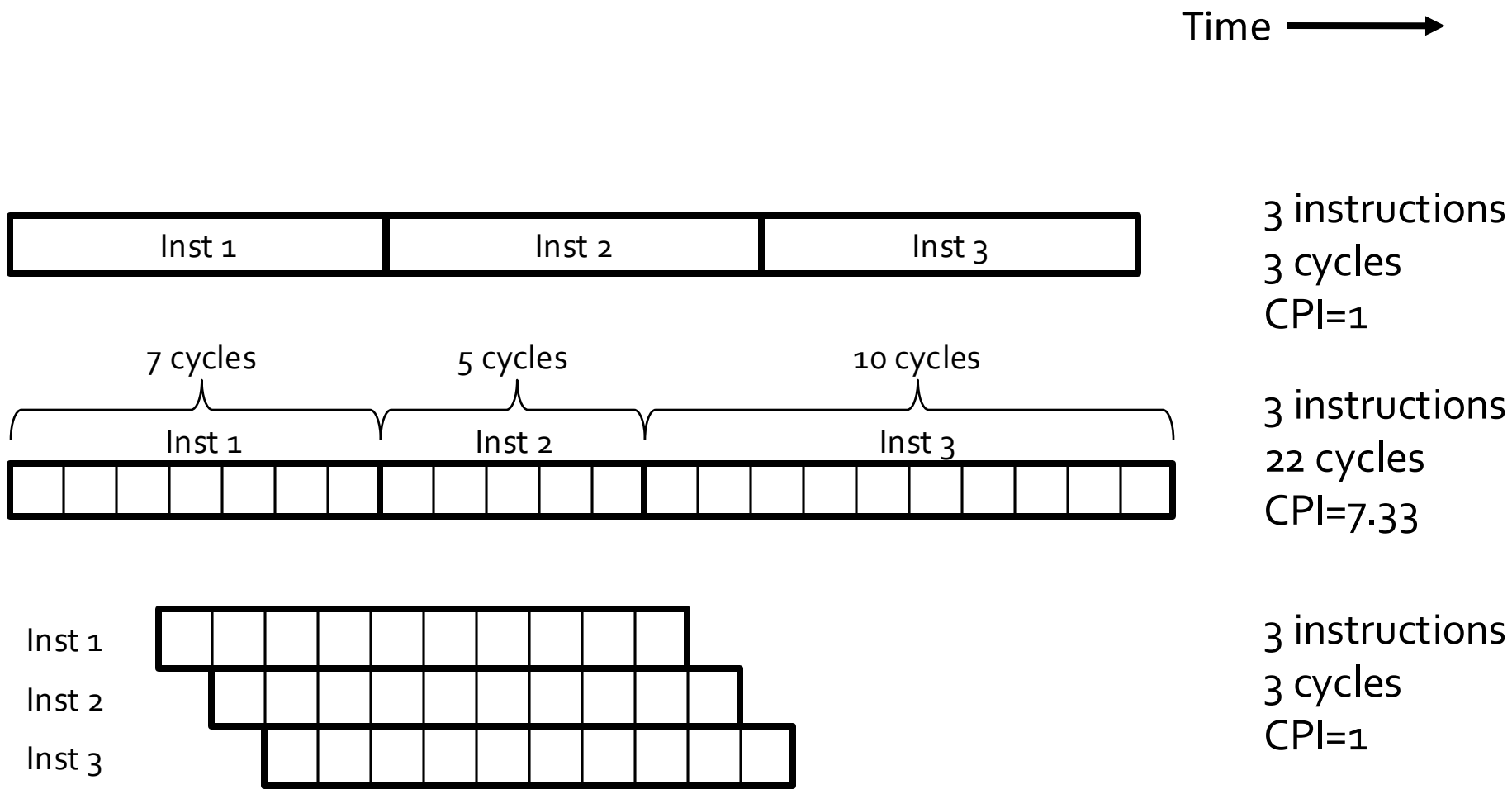
Iron Law of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Instructions per program depends on source code, compiler technology, and ISA
- Cycles per instructions (CPI) depends on ISA and μarchitecture
- Time per cycle depends upon the μarchitecture and base technology

Microarchitecture	CPI	cycle time
Single-cycle unpipelined	1	long
Multi-cycle, unpipelined control	>1	short
Pipelined	1	short

CPI Examples



Instructions Interact With Each Other in Pipeline

- **Structural Hazard:** An instruction in the pipeline may need a resource being used by another instruction in the pipeline
- **Data Hazard:** An instruction depends on a data value produced by an earlier instruction
- **Control Hazard:** Whether an instruction should be executed depends on a control decision made by an earlier instruction

Agenda

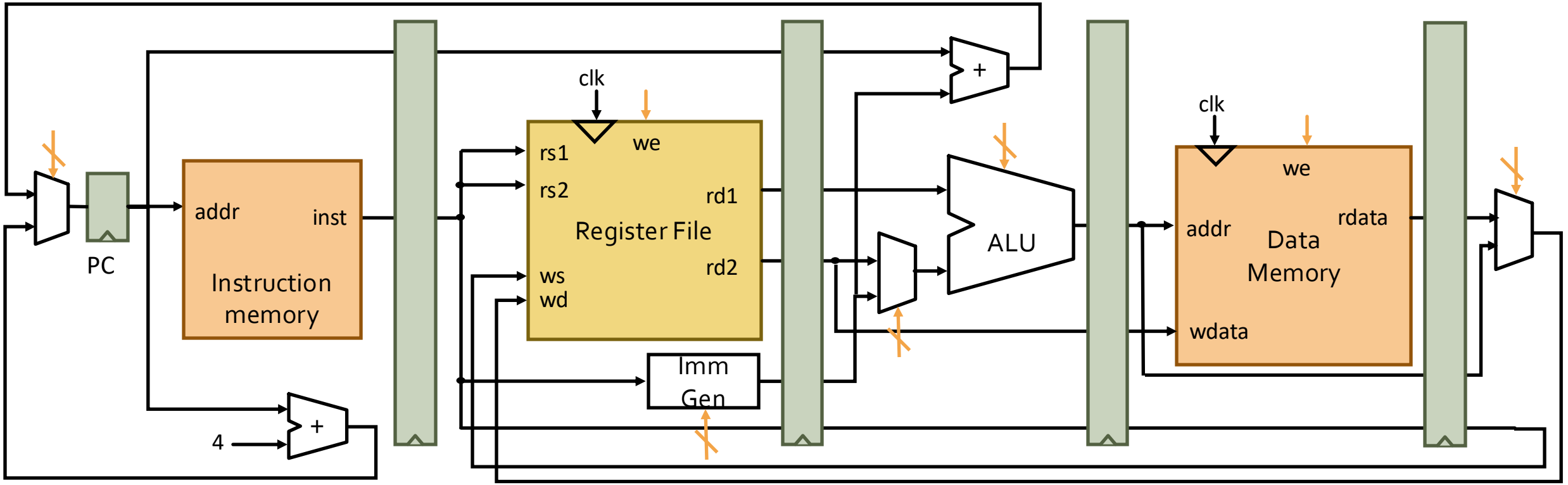
- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

Overview of Structural Hazards

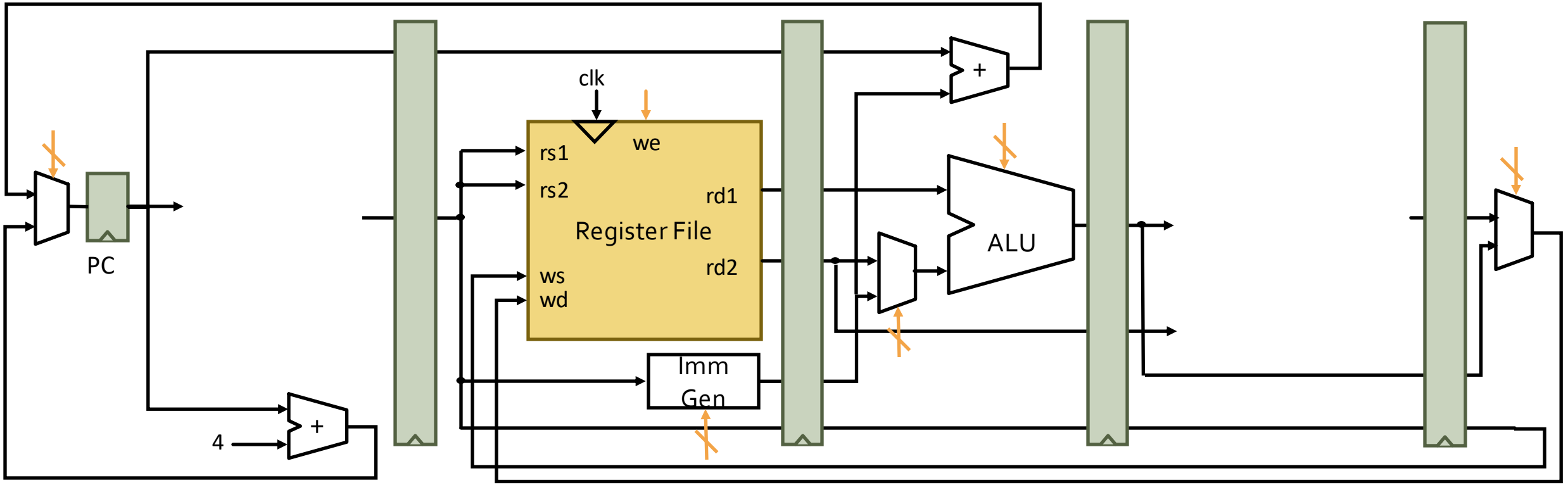
- Structural hazard occurs when two instructions need same hardware resource at same time
- Approaches to resolving structural hazards
  - **Schedule:** Programmer explicitly avoids scheduling instructions that would create structural hazards
  - **Stall:** Hardware includes control logic that stalls until earlier instruction is no longer using contended resources
  - **Duplicate:** Add more hardware to design so that each instruction can access independent resources at the same time
- Our 5-stage pipeline has no structural hazards by design

Example Structural Hazard

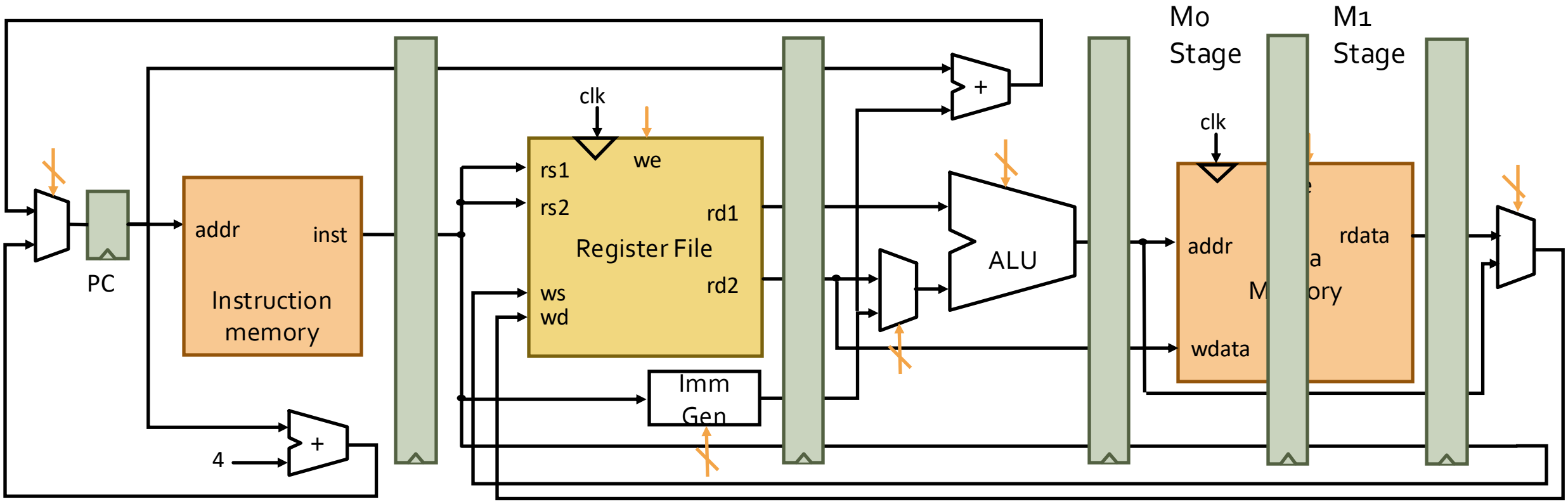
Unified Memory



Example Structural Hazard
Unified Memory



Example Structural Hazard
2-Cycle Memory



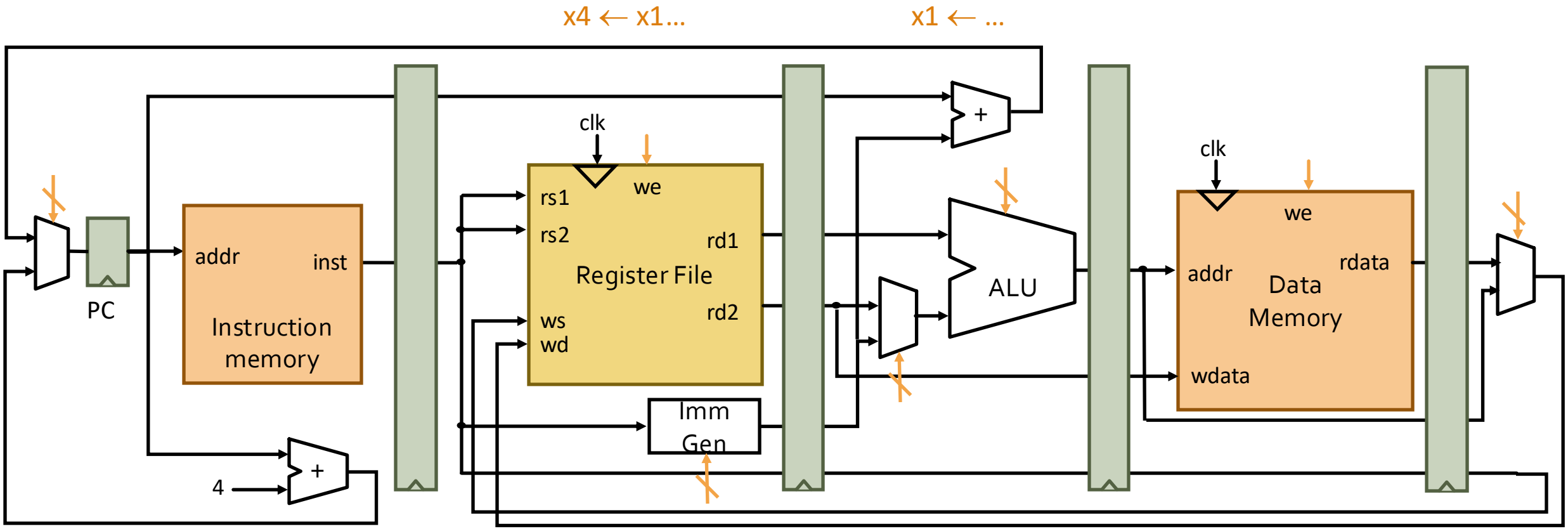
Agenda

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

Overview of Data Hazards

- Data hazard occurs when one instruction depends on a data value produced by a preceding instruction still in the pipeline
- Approaches to resolving data hazards
  - **Schedule:** Programmer explicitly avoids scheduling instructions that would create data hazards
  - **Stall:** Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
  - **Bypass:** Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
  - **Speculate:** Guess that there is not a problem, if incorrect kill speculative instruction and restart

Example Data Hazard



…

x1 ← x0 + 10

x4 ← x1 + 17

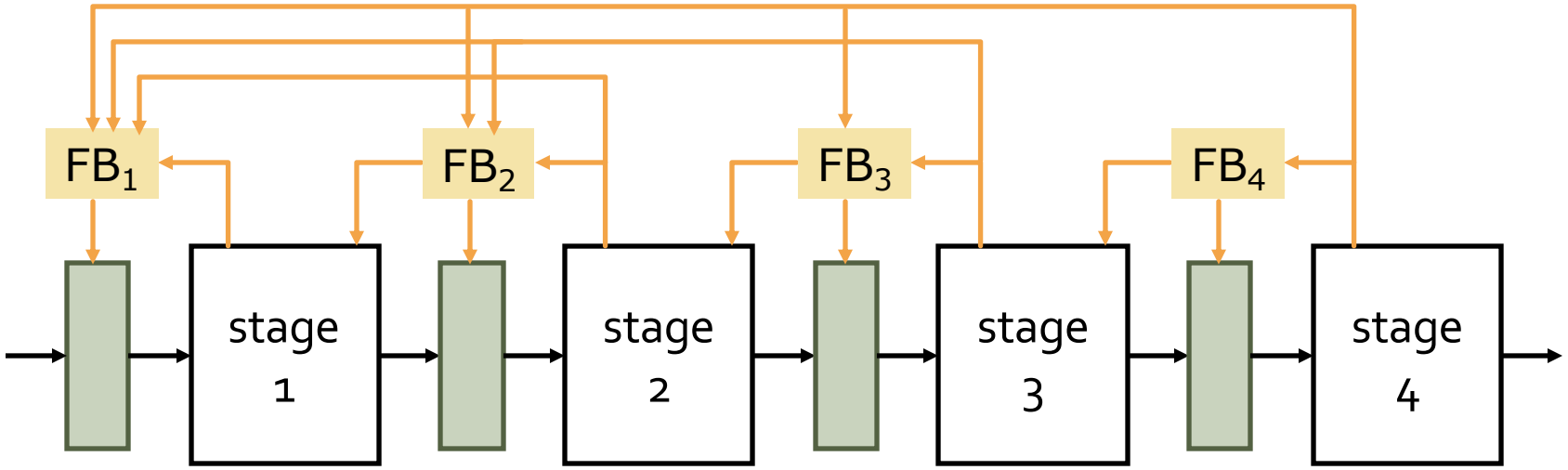
…

x1 is stale. Oops!

Overview of Data Hazards

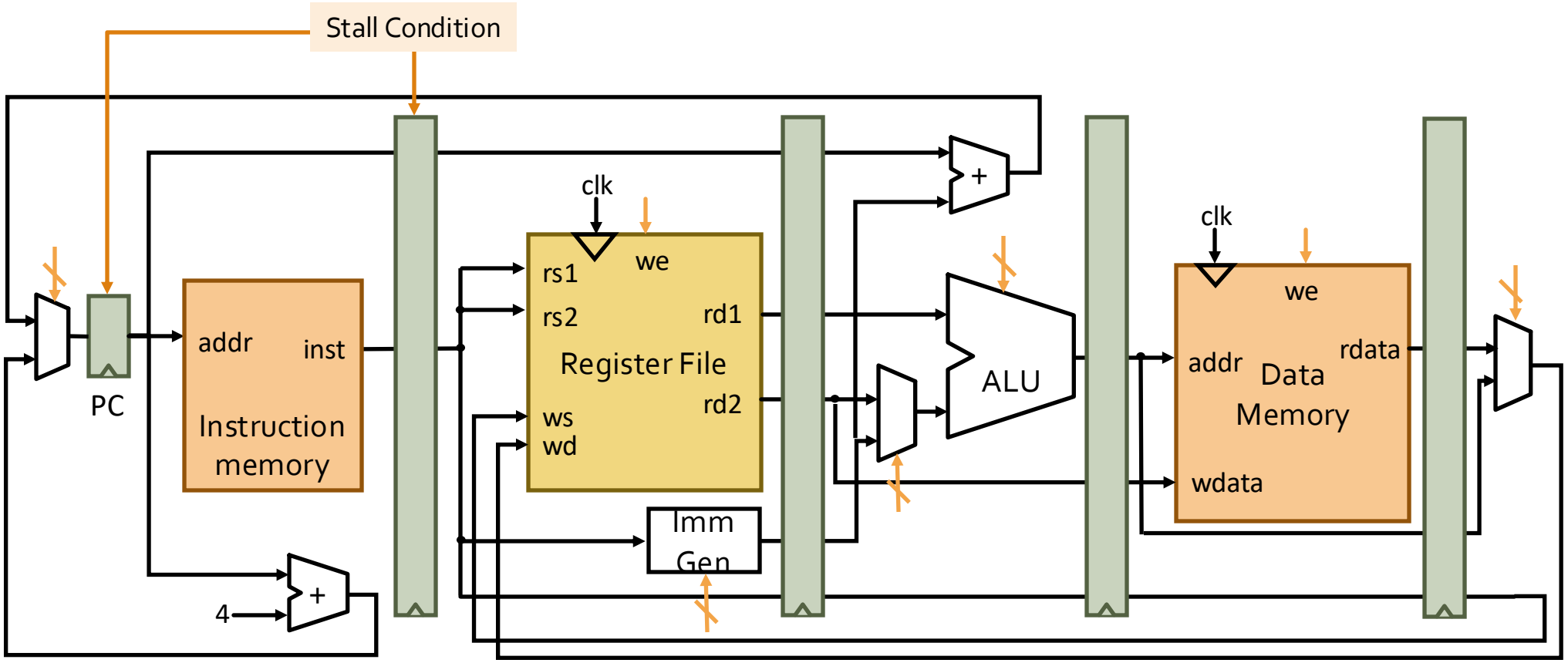
- Data hazard occurs when one instruction depends on a data value produced by a preceding instruction still in the pipeline
- Approaches to resolving data hazards
  - **Schedule:** Programmer explicitly avoids scheduling instructions that would create data hazards
  - **Stall:** Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
  - **Bypass:** Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
  - **Speculate:** Guess that there is not a problem, if incorrect kill speculative instruction and restart

Feedback to Resolve Hazards



- Later stages provide dependence information to earlier stages which can **stall (or kill)** instructions
- Controlling a pipeline in this manner works provided the instruction at stage i+1 can complete without any interaction from instructions in stages 1 to i (otherwise deadlock)

Resolving Data Hazards with Stalls (Interlocks)



...

x1 ← x0 + 10

x4 ← x1 + 17

...

Stalled Stages and Pipeline Bubbles

time		t0	t1	t2	t3	t4	t5	t6	t7	…			
I_1	x1 ← x0 + 10	F_1	D_1	X_1	M_1	W_1							
I_2	x4 ← x1 + 17		F_2	D_2	D_2	D_2	D_2	X_2	M_2	W_2			
I_3				F_3	F_3	F_3	F_3	D_3	X_3	M_3	W_3		
I_4								F_4	D_4	X_4	M_4	W_4	
I_5									F_5	D_5	X_5	M_5	W_5

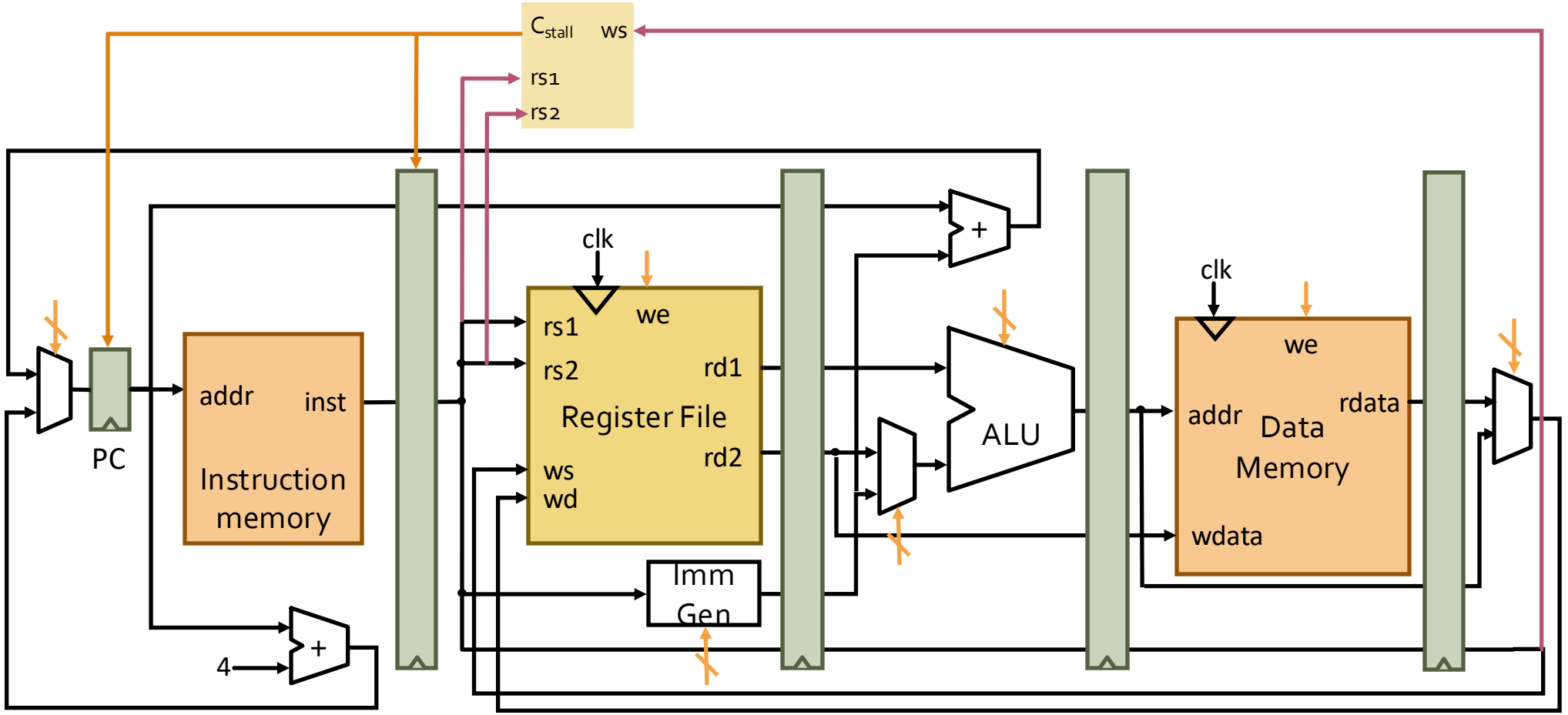
stalled stages

Resource Usage

time	t0	t1	t2	t3	t4	t5	t6	t7	…			
F	I_1	I_2	I_3	I_3	I_3	I_3	I_4	I_5				
D		I_1	I_2	I_2	I_2	I_2	I_3	I_4	I_5			
X			I_1	-	-	-	I_2	I_3	I_4	I_5		
M				I_1	-	-	-	I_2	I_3	I_4	I_5	
W					I_1	-	-	-	I_2	I_3	I_4	I_5

- ⇨ pipeline bubble

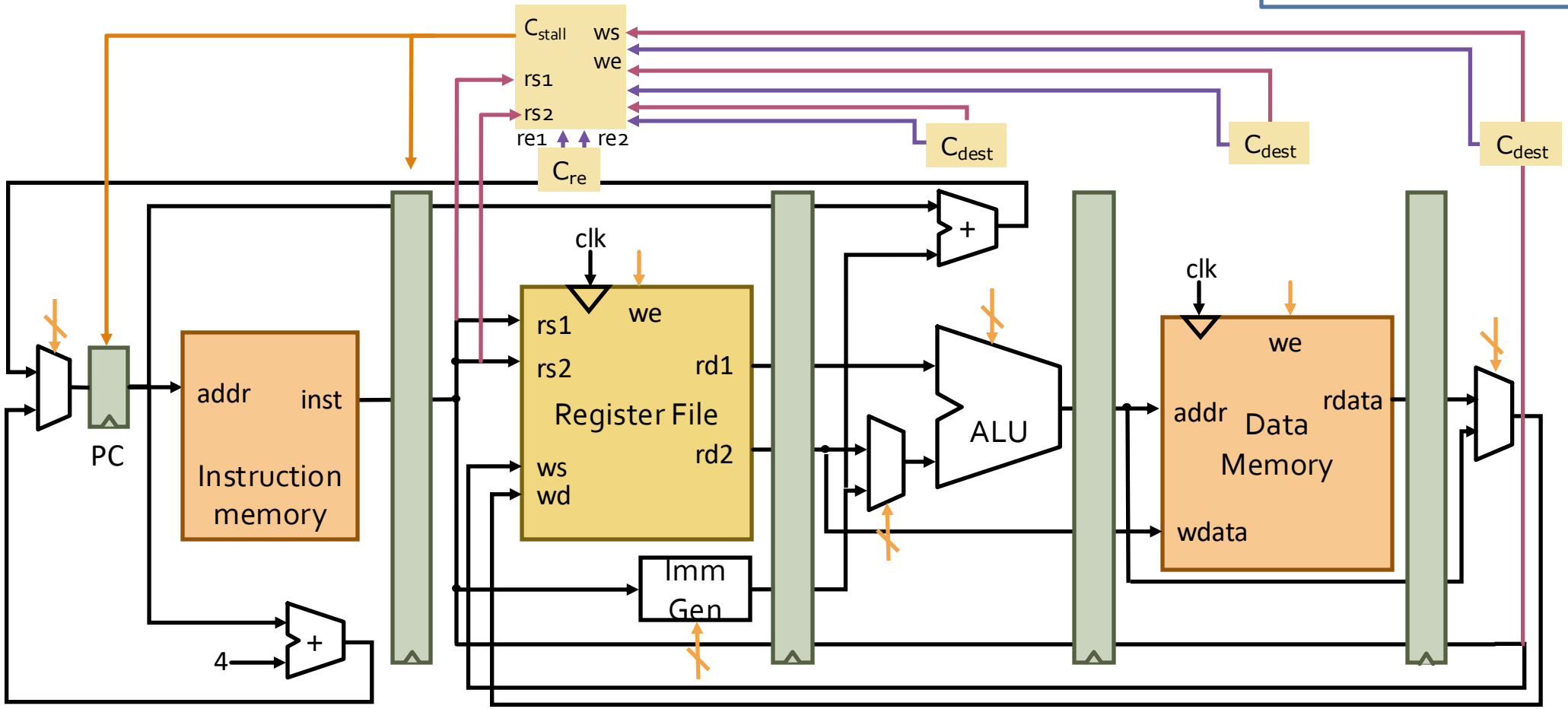
Stall Control Logic



Compare the source registers of the instruction in the decode stage with the destination register of the **uncommitted** instructions.

Stall Control Logic

we: write enable, 1-bit on/off
ws: write select, 5-bit register number
re: read enable, 1-bit on/off
rs: read select, 5-bit register number



Should we always stall if the rs field matches some rd?

not every instruction writes a register→we
not every instruction reads a register→re

Source & Destination Registers

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

And more!

		Source	Destination
ALU	rd ⟵ (rs1) func (rs2)	rs1, rs2	rd
ALUi	rd ⟵ (rs1) func immediate	rs1	rd
LW	rd ⟵ M [(rs1) + immediate]	rs1	rd
SW	M [(rs1) + immediate] ⟵ (rs2)	rs1, rs2	
JAL	rd ⟵ (PC+4), PC ⟵ (PC) + immediate		rd
BEQ	If (rs1) == (rs2) then PC ⟵ (PC) + immediate else PC ⟵ (PC) + 4	rs1, rs2	

Deriving the Stall Signal

C_{dest}

ws = **Case** opcode

ALU	⇒ rd
ALUi, LW	⇒ rd
JAL, JALR	⇒ rd(x1)

we = **Case** opcode

ALU, ALUi, LW	⇒ on (ws ≠ 0)
JAL, JALR	⇒ on (ws ≠ 0)
…	⇒ off

C_{re}

re1 = **Case** opcode

ALU, ALUi, LW, SW, BZ, JR, JALR	⇒ on
J, JAL	⇒ off

re2 = **Case** opcode

ALU, SW	⇒ on
…	⇒ off

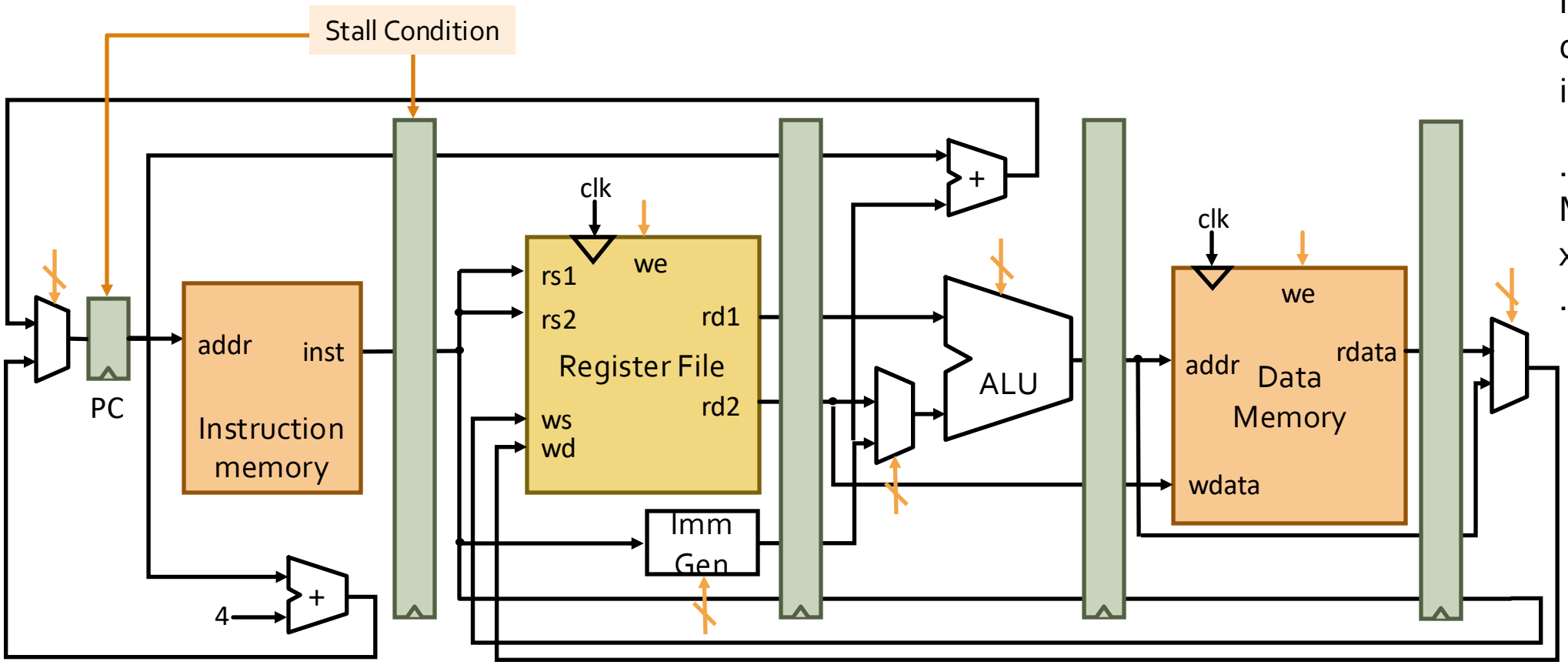
C_{stall}

$$stall = ((rs1_D = ws_E).we_E + (rs1_D = ws_M).we_M + (rs1_D = ws_W).we_W) \,.\, re1_D + ((rs2_D = ws_E).we_E + (rs2_D = ws_M).we_M + (rs2_D = ws_W).we_W) \,.\, re2_D$$

This is not the full story !

Hazards due to Loads and Stores

What if x1+7 = x3+5 ?



Is there any possible data hazard in this instruction sequence?

...
M[x1+7] ← x2
x4 ← M[x3+5]
...

Load & Store Hazards

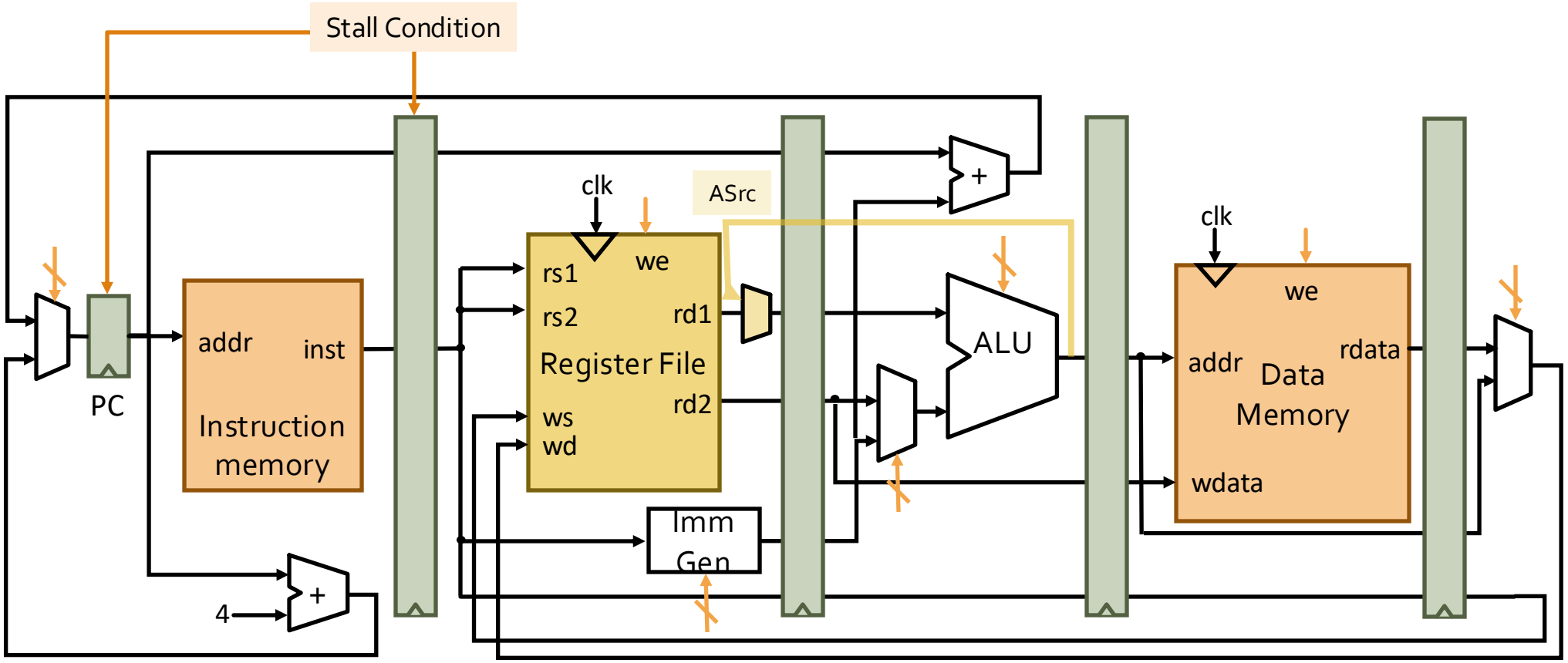
- Example instruction sequence
  - M[x1+7] ← x2
  - x4 ← M[x3+5]

- What if x1+7 == x3+5 ?
  - Writing and reading to/from the same address
  - Hazard is avoided because our memory system completes writes in a **single cycle**
  - More realistic memory system will require more careful handling of data hazards due to loads and stores

Overview of Data Hazards

- Data hazard occurs when one instruction depends on a data value produced by a preceding instruction still in the pipeline
- Approaches to resolving data hazards
  - **Schedule:** Programmer explicitly avoids scheduling instructions that would create data hazards
  - **Stall:** Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
  - **Bypass:** Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
  - **Speculate:** Guess that there is not a problem, if incorrect kill speculative instruction and restart

Adding Bypassing to the Datapath



When does **this** bypass help?

x1 ← x0 + 10
x4 ← x1 + 17

---

x1 ← M[x0+10]
x4 ← x1 + 17

JAL x1, 500
x4 ← x1 + 17

Deriving the Bypass Signal

time		t0	t1	t2	t3	t4	t5	t6	t7	…		
I_1	x1 ← x0 + 10	F_1	D_1	X_1	M_1	W_1						
I_2	x4 ← x1 + 17		F_2	D_2	D_2	D_2	D_2	X_2	M_2	W_2		
I_3				F_3	F_3	F_3	F_3	D_3	X_3	M_3	W_3	
I_4				stalled stages				F_4	D_4	X_4	M_4	W_4
I_5									F_5	D_5	X_5	M_5	W_5

Each **stall** *or* **kill** introduces a bubble in the pipeline ⇨ *CPI > 1*
A new datapath, i.e., *a* **bypass**, can get the data from the output of the ALU to its input

time		t0	t1	t2	t3	t4	t5	t6	t7	…
I_1	x1 ← x0 + 10	F_1	D_1	X_1	M_1	W_1				
I_2	x4 ← x1 + 17		F_2	D_2	X_2	M_2	W_2			
I_3				F_3	D_3	X_3	M_3	W_3		
I_4					F_4	D_4	X_4	M_4	W_4	
I_5						F_5	D_5	X_5	M_5	W_5

The Bypass Signal

Deriving it from the Stall Signal

$$stall = (((rs1_D = ws_E).we_E + (rs1_D = ws_M).we_M + (rs1_D = ws_W).we_W).re1_D + ((rs2_D = ws_E).we_E + (rs2_D = ws_M).we_M + (rs2_D = ws_W).we_W).re2_D)$$

ws = **Case** opcode

ALU	⇒ rd
ALUi, LW	⇒ rd
JAL , JALR	⇒ rd

we = **Case** opcode

ALU, ALUi, LW	⇒ on (ws ≠ 0)
JAL, JALR	⇒ on (ws ≠ 0)
…	⇒ off

$$ASrc = (rs1_D = ws_E).we_E.re1_D$$

Is this correct?

No! because only ALU and ALUi instructions can benefit from this bypass. Split we_E into two components: we-bypass, we-stall.

Bypass and Stall Signals

Split we_E into two components: we-bypass, we-stall

$we\text{-}bypass_E$ = **Case** opcode
- ALU, ALUi ⇒ on (ws ≠ 0)
- … ⇒ off

$we\text{-}stall_E$ = **Case** opcode
- LW ⇒ on (ws ≠ 0)
- JAL, JALR ⇒ on (ws ≠ 0)
- … ⇒ off

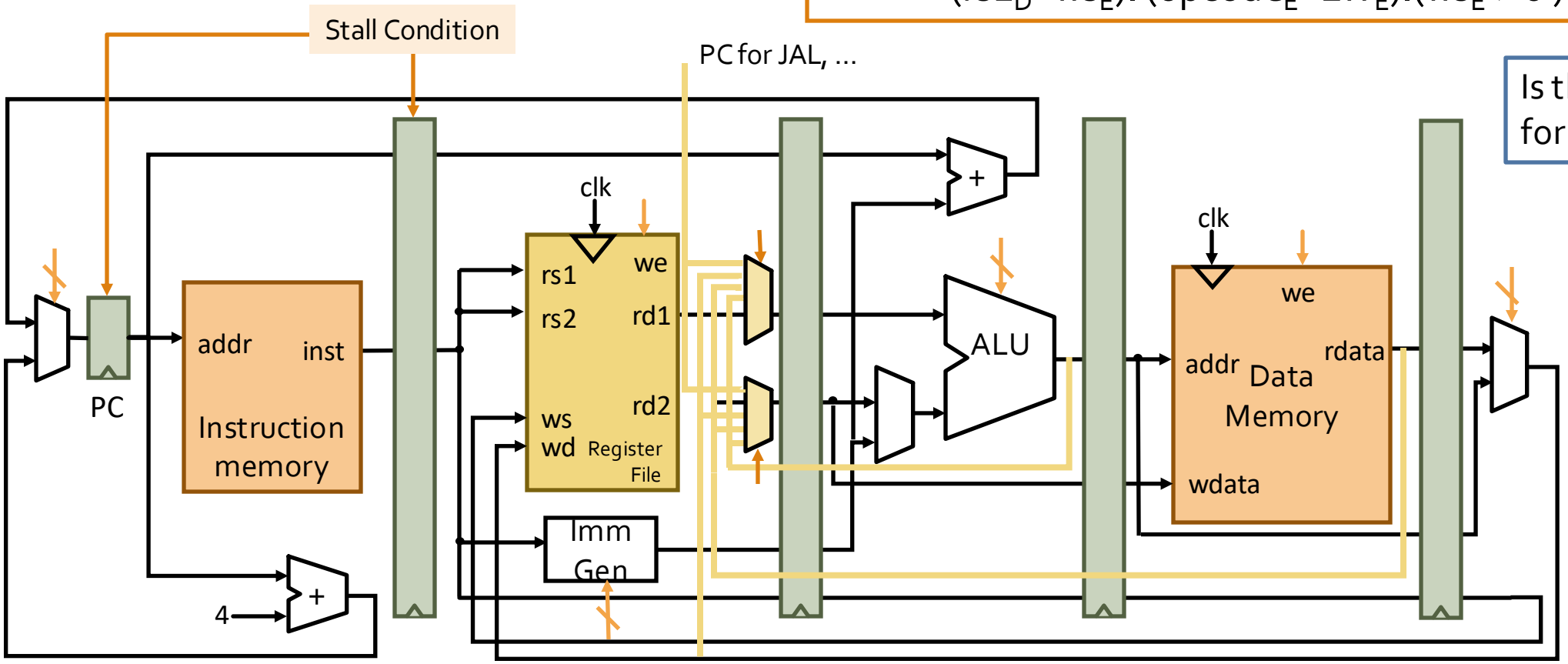
$ASrc = (rs1_D = ws_E).we\text{-}bypass_E.re1_D$

$stall = (((rs1_D = ws_E).we\text{-}stall_E + (rs1_D = ws_M).we_M + (rs1_D = ws_W).we_W).re1_D$
$+((rs2_D = ws_E).we_E + (rs2_D = ws_M).we_M + (rs2_D = ws_W).we_W).re2_D)$

Fully Bypassed Datapath

$$\text{stall} = (rs1_D = ws_E).(opcode_E = LW_E).(ws_E \neq 0).re1_D + (rs2_D = ws_E).(opcode_E = LW_E).(ws_E \neq 0).re2_D$$

Is there still a need for the stall signal ?



Overview of Data Hazards

- Data hazard occurs when one instruction depends on a data value produced by a preceding instruction still in the pipeline
- Approaches to resolving data hazards
  - **Schedule:** Programmer explicitly avoids scheduling instructions that would create data hazards
  - **Stall:** Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
  - **Bypass:** Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
  - **Speculate:** Guess that there is not a problem, if incorrect kill speculative instruction and restart

Agenda

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

Control Hazards

- What do we need to calculate next PC?
  - For Jumps
    - Opcode, PC and offset
  - For Jump Register
    - Opcode, Register value, and PC
  - For Conditional Branches
    - Opcode, Register (for condition), PC and offset
  - For all other instructions
    - Opcode and PC
      - have to know it's not one of above

Opcode Decoding Bubble

Speculate next address is PC+4

time		to	t1	t2	t3	t4	t5	t6	t7	…		
I_1	x1 ⟵ x0 + 10	F_1	D_1	X_1	M_1	W_1						
I_2	x3 ⟵ x2 + 17		F_2	F_2	D_2	X_2	M_2	W_2				
I_3					F_3	F_3	D_3	X_3	M_3	W_3		
I_4							F_4	F_4	D_4	X_4	M_4	W_4

Resource Usage

time	to	t1	t2	t3	t4	t5	t6	t7	…			
F	I_1	-	I_2	-	I_3	-	I_4	-				
D		I_1	-	I_2	-	I_3	-	I_4	-			
X			I_1	-	I_2	-	I_3	-	I_4	-		
M				I_1	-	I_2	-	I_3	-	I_4	-	
W					I_1	-	I_2	-	I_3	-	I_4	-

CPI = 2!

- ⇨ pipeline bubble

Branch Pipeline Diagrams (resolved in execute stage)

time		to	t1	t2	t3	t4	t5	t6	t7	…
I_1	096:ADD	F_1	D_1	X_1	M_1	W_1				
I_2	100: BEQZ + 200		F_2	D_2	X_2	M_2	W_2			
I_3	104: ADD			F_3	D_3	-	-	-		
I_4	108:				F_4	-	-	-	-	
I_5	300: ADD					F_5	D_5	X_5	M_5	W_5

Resource Usage

time	to	t1	t2	t3	t4	t5	t6	t7	…
IF	I_1	I_2	I_3	I_4	I_5				
ID		I_1	I_2	I_3	-	I_5			
EX			I_1	I_2	-	-	I_5		
MA				I_1	I_2	-	-	I_5	
WB					I_1	I_2	-	-	I_5

- ⇨ pipeline bubble

Reducing Branch Penalty

- One pipeline bubble can be removed if an extra comparator is used in the Decode stage
  - But might elongate cycle time

Branch Delay Slots (expose control hazard to software)

- Change the ISA semantics so that the instruction that follows a jump or branch is always executed
  - Gives compiler the flexibility to put in a useful instruction where normally a pipeline bubble would have resulted

I_1	096:	ADD	
I_2	100:	BEQZ + 200	
I_3	104:	ADD	← Delay Slot instruction executed regardless of branch outcome
I_4	304:	ADD	

- Other techniques include more advanced branch prediction, which can dramatically reduce the branch penalty… (to come later)

Branch Pipeline Diagrams (branch delay slot)

time		to	t1	t2	t3	t4	t5	t6	t7	…
I_1	096:ADD	F_1	D_1	X_1	M_1	W_1				
I_2	100: BEQZ + 200		F_2	D_2	X_2	M_2	W_2			
I_3	104: ADD			F_3	D_3	X_3	M_3	W_3		
I_4	300: ADD				F_4	D_4	EX_4	M_4	W_4	

Resource Usage

time	t0	t1	t2	t3	t4	t5	t6	t7	…
IF	I_1	I_2	I_3	I_4					
ID		I_1	I_2	I_3	I_4				
EX			I_1	I_2	I_3	I_4			
MA				I_1	I_2	I_3	I_4		
WB					I_1	I_2	I_3	I_4	

Why an Instruction may not be dispatched every cycle (CPI>1)?

- Full bypassing may be too expensive to implement
  - Typically, all frequently used paths are provided
  - Some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
- Loads have two cycle latency
  - Instruction after load cannot use load result
- Jumps/Conditional branches may cause bubbles
  - Kill following instruction(s) if no delay slots

Machines with software-visible delay slots may execute significant number of NOP instructions inserted by the compiler. NOPs not counted in useful CPI (alternatively, increase instructions/program)

Other Control Hazards

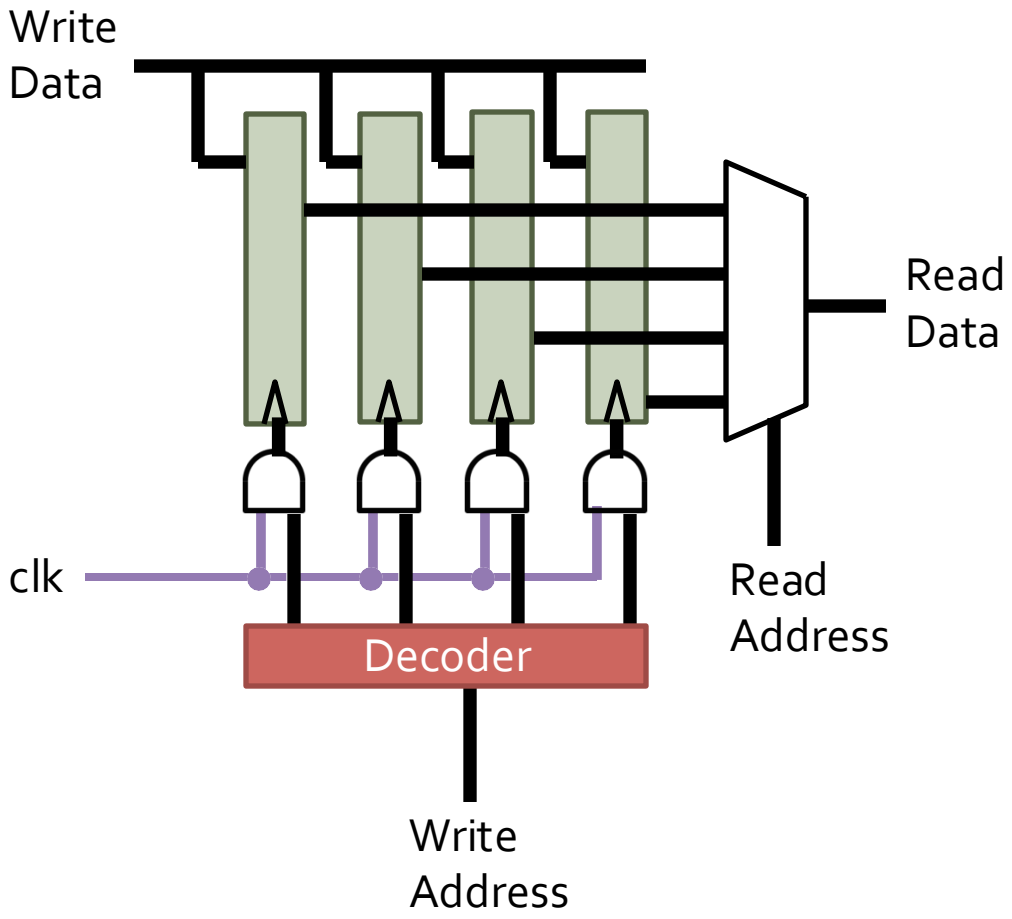
- Exceptions
- Interrupts

More on this later in the course!

Agenda

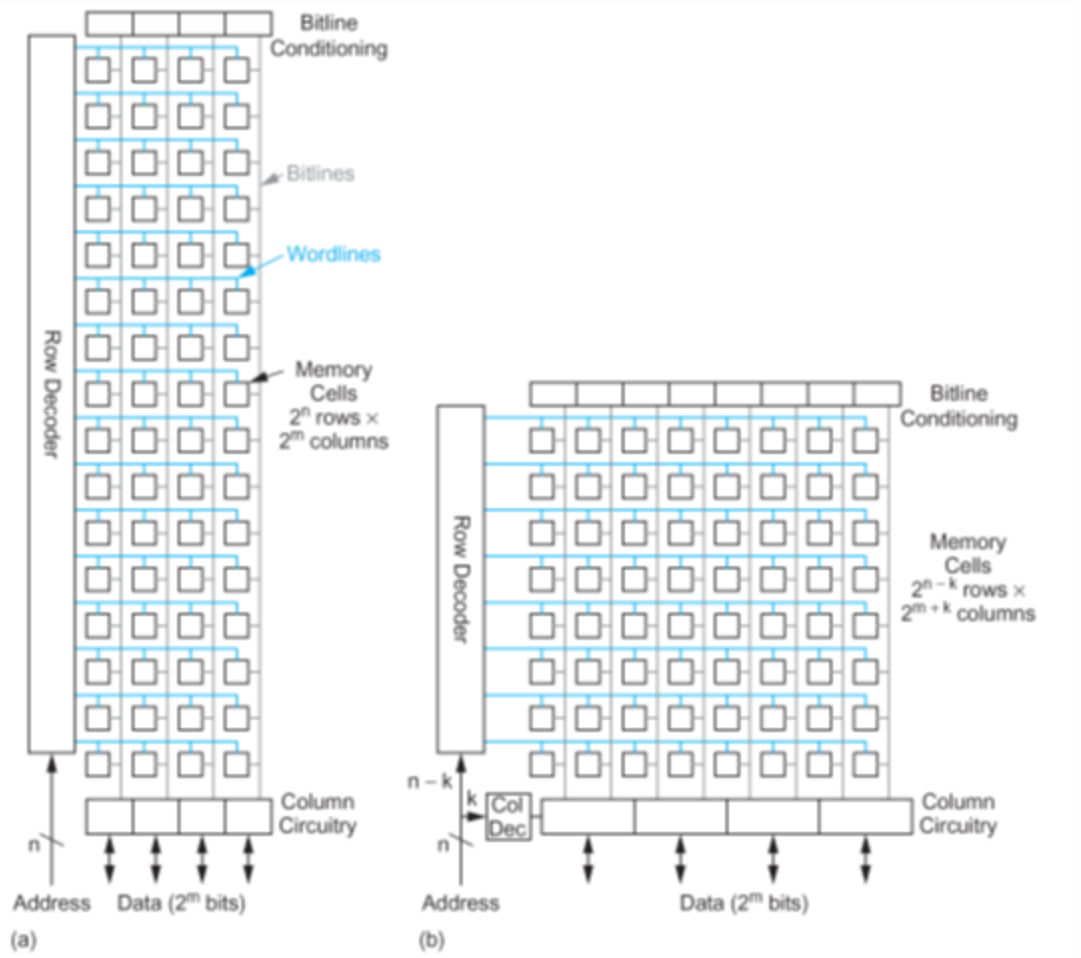
- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

Naive Register File



Array Architecture

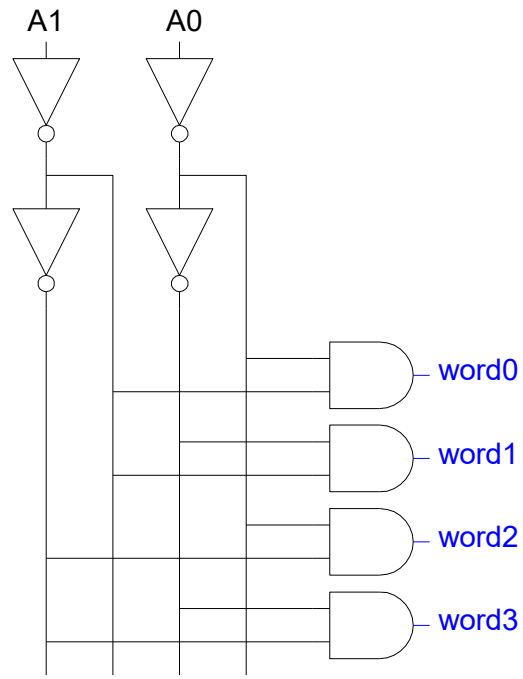
- 2^n words of 2^m bits each
- If n >> m, fold by 2^k into fewer rows of more columns
- Good regularity – easy to design
- Very high density if good cells are used



From Weste and Harris "CMOS VLSI Design : A circuits and systems perspective" Image

Decoders

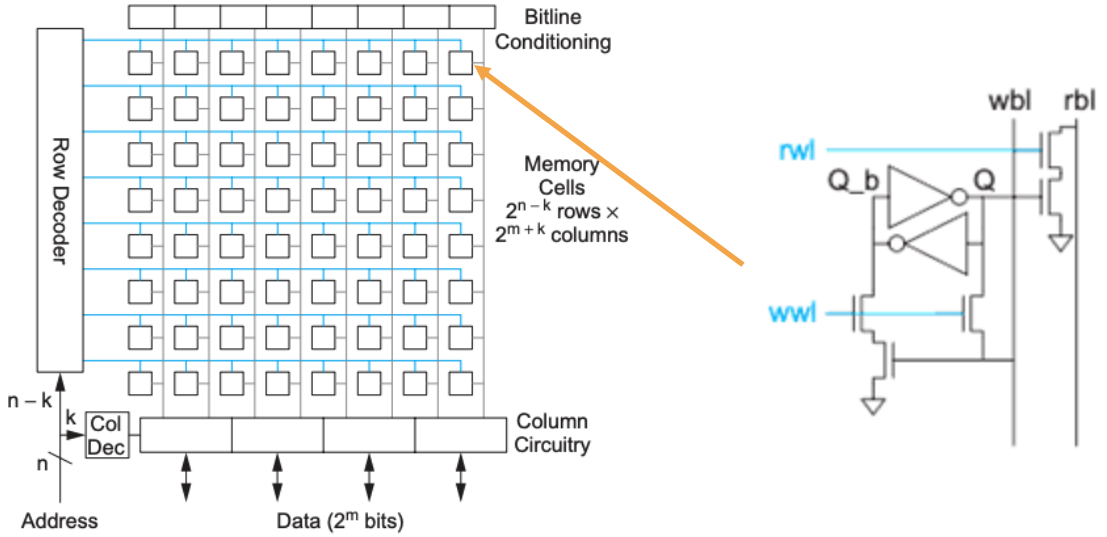
- $n:2^n$ decoder consists of 2^n n-input AND gates
  - One needed for each row of memory
  - Build AND from NAND or NOR gates



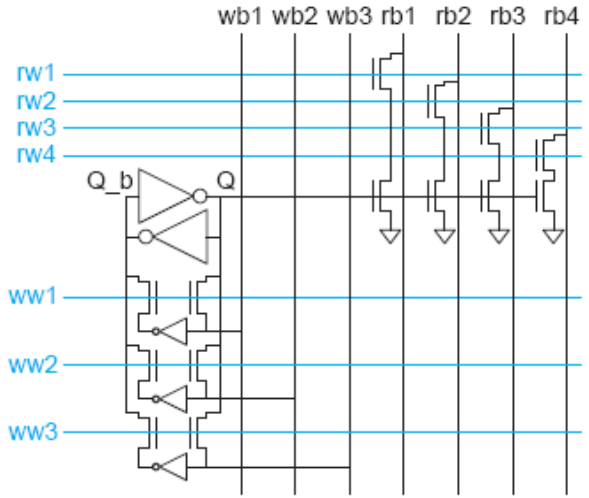
Column Circuitry

- Bitline conditioning
  - Precharge bitlines high before reads
  - Equalize bitlines to minimize voltage difference when using sense amplifiers
- Sense amplifiers
  - Sense amplifiers are triggered on small voltage swing (reduce ΔV)
- Column multiplexing
  - Recall that array may be folded for good aspect ratio
  - Ex: 2 kword x 16 folded into 256 rows x 128 columns
    - Must select 16 output bits from the 128 columns
    - Requires 16 8:1 column multiplexers

Memory Arrays: Register File

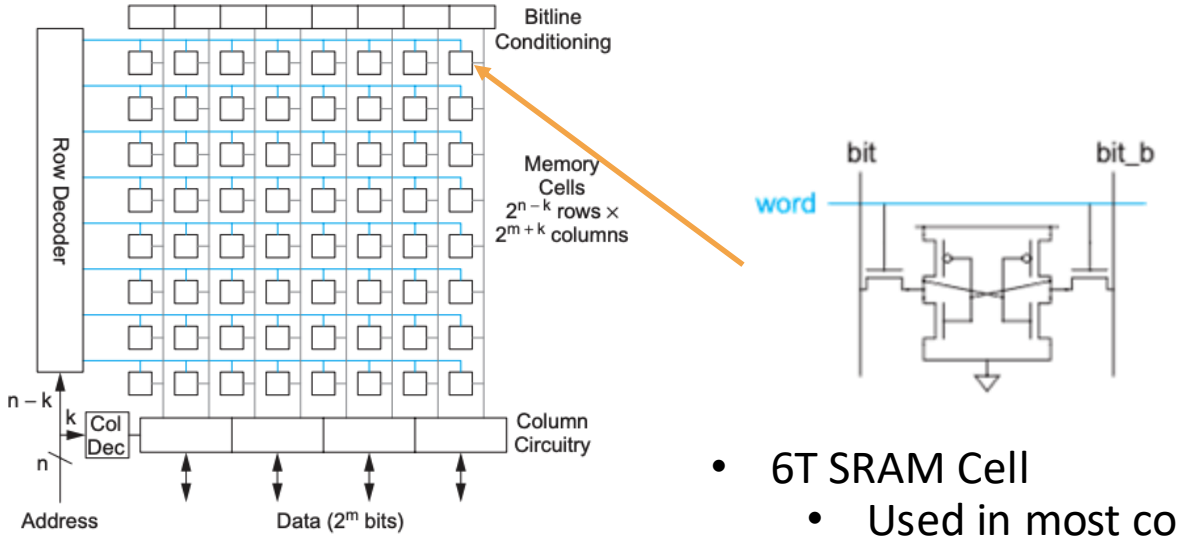


Multiple ports



From Weste and Harris "CMOS VLSI Design : A circuits and systems perspective" Image

Memory Arrays: SRAM

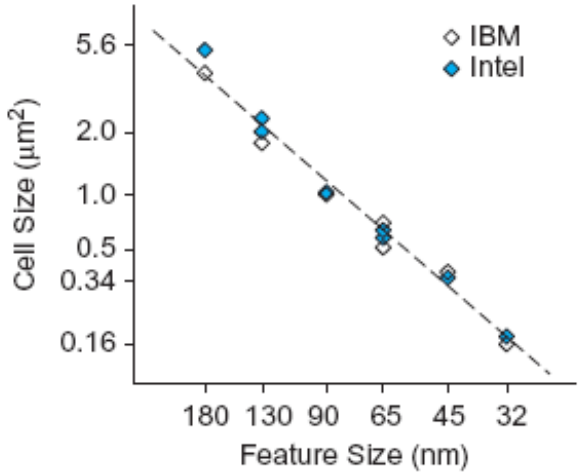


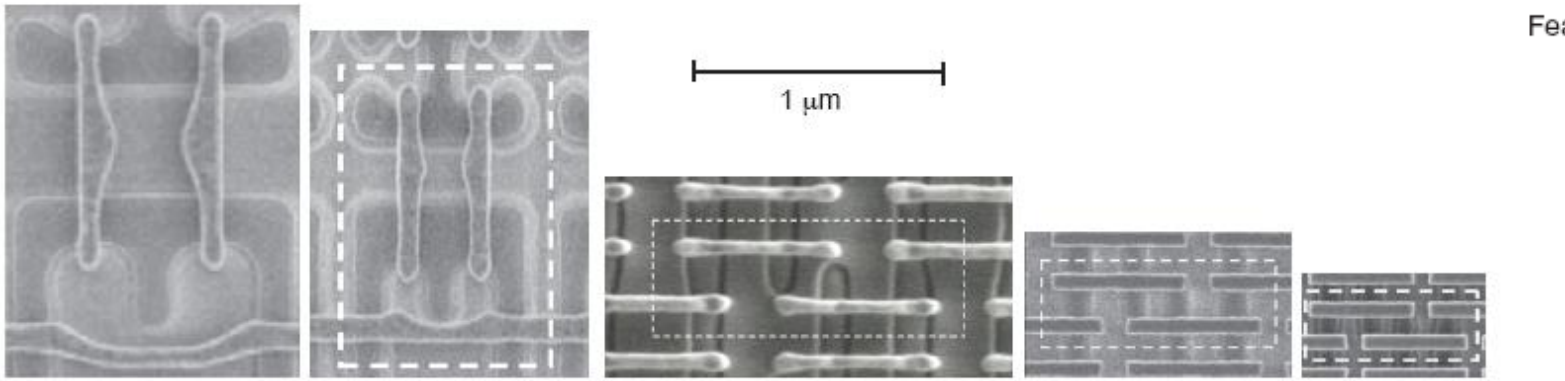
- 6T SRAM Cell
  - Used in most commercial chips
  - Data stored in cross-coupled inverters
- Read:
  - Precharge bit, bit_b
  - Raise wordline
- Write:
  - Drive data onto bit, bit_b
  - Raise wordline

From Weste and Harris "CMOS VLSI Design : A circuits and systems perspective" Image

Commercial SRAMs

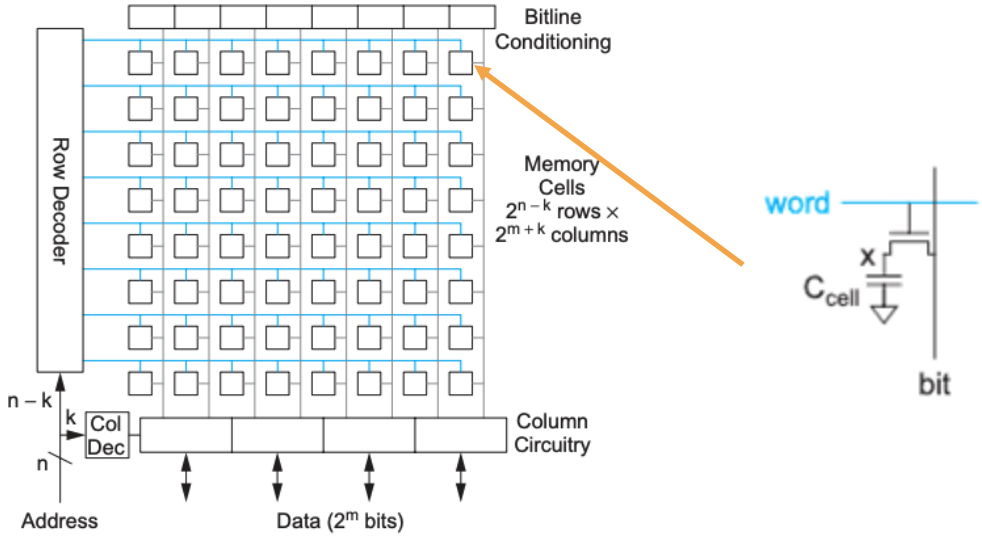
- Five generations of Intel SRAM cell micrographs
  - Transition to thin cell at 65 nm
  - Steady scaling of cell area

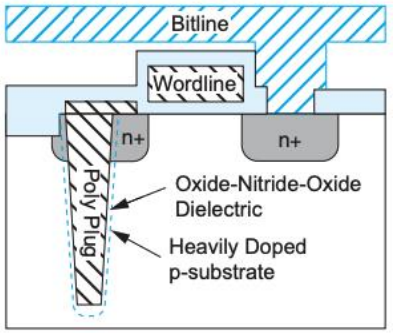


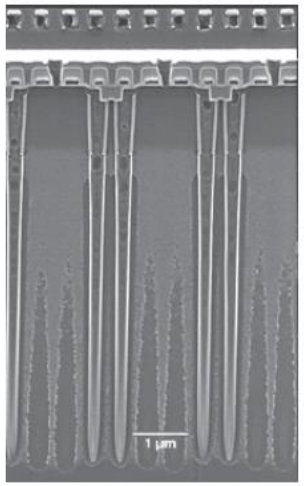


From Weste and Harris "CMOS VLSI Design : A circuits and systems perspective" Image

Memory Arrays: DRAM



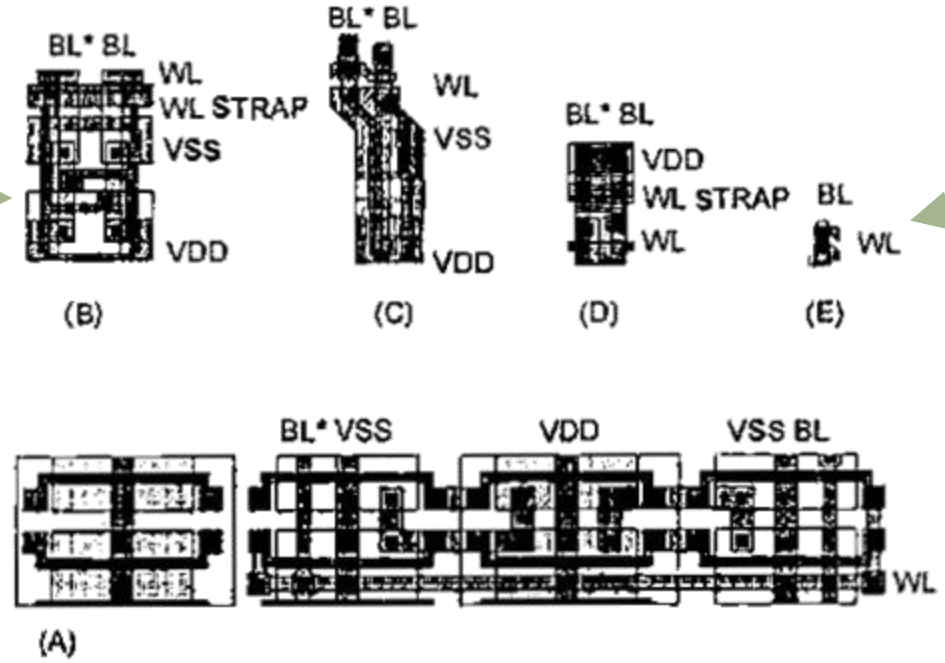




From Weste and Harris "CMOS VLSI Design : A circuits and systems perspective" Image

Relative Memory Sizes of SRAM vs. DRAM

On-Chip SRAM on logic chip



DRAM on memory chip

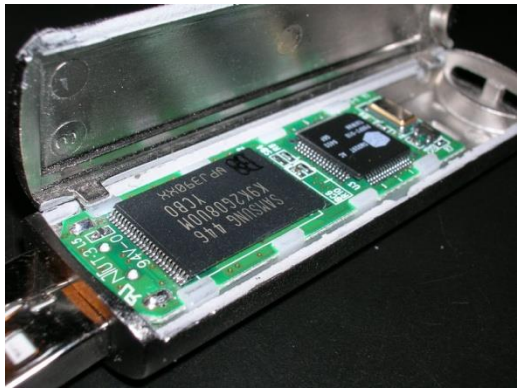
1 Memory cell in 0.5µm processes

a) Gate Array SRAM
b) Embedded SRAM
c) Standard SRAM (6T cell with local interconnect)
d) ASIC DRAM
e) Standard DRAM (stacked cell)

[From Foss, R.C. "Implementing Application-Specific Memory", ISSCC 1996]

Flash Storage

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

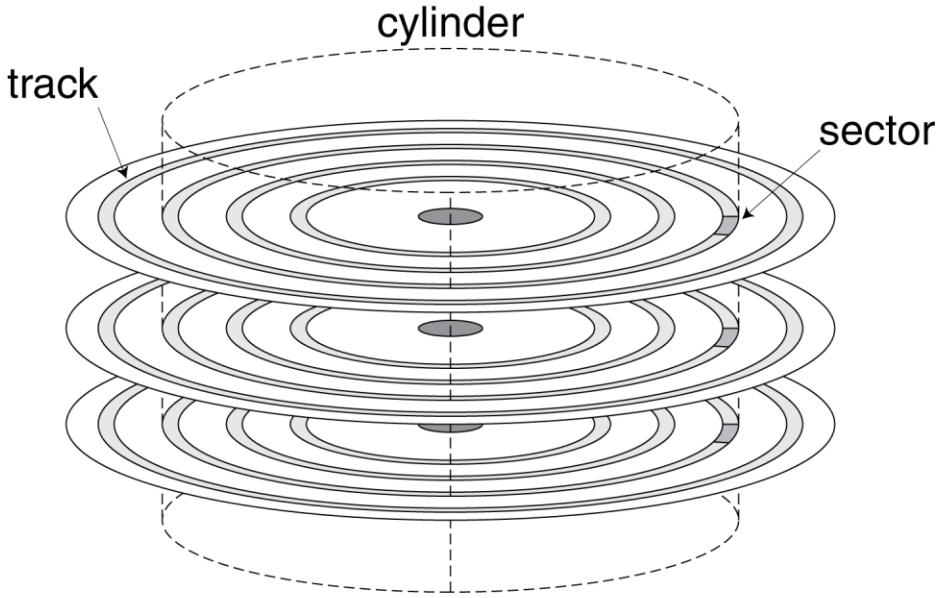


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage



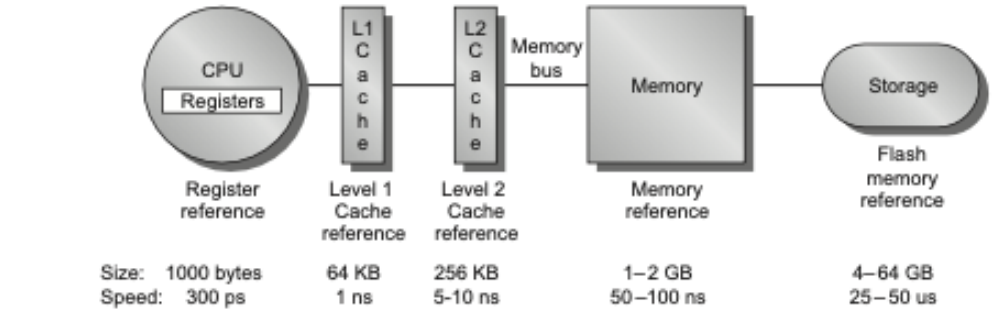
Summary: Memory Technologies

	Capacity	Latency	Cost/GB
Register	1000s of bits	20 ps	$$$$
SRAM	~10 KB – 10 MB	1 – 10 ns	~$1000
DRAM	~ 10 GB	80 ns	~$10
Flash	~ 100 GB	100 us	~$1
Hard disk	~ 1TB	10 ms	~$0.1

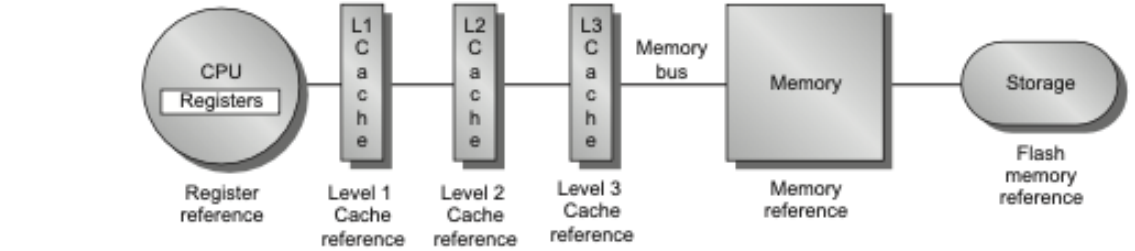
Low Capacity
Low Latency
High Bandwidth

High Capacity
High Latency
Low Bandwidth

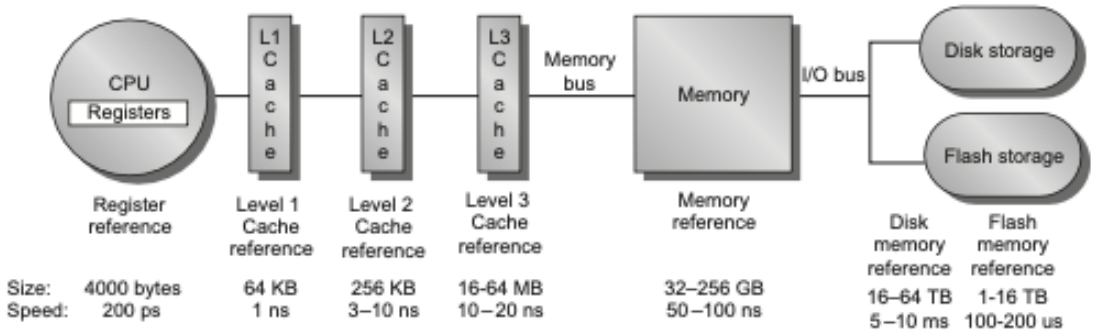
Can we get the best of both worlds? (large, fast, cheap)



(A) Memory hierarchy for a personal mobile device



(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

Apple M1 Memory System

Storage and DRAM

Two [redacted] chips on same package as system SoC

Two SSD chips

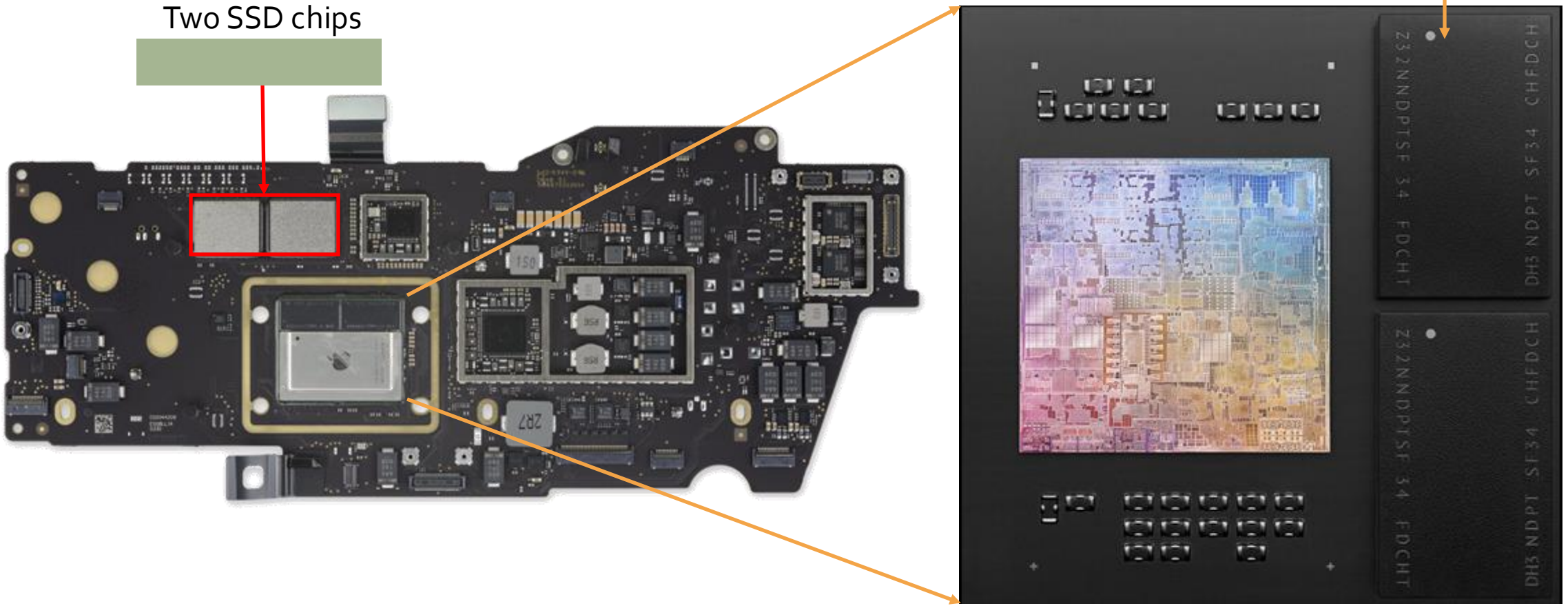
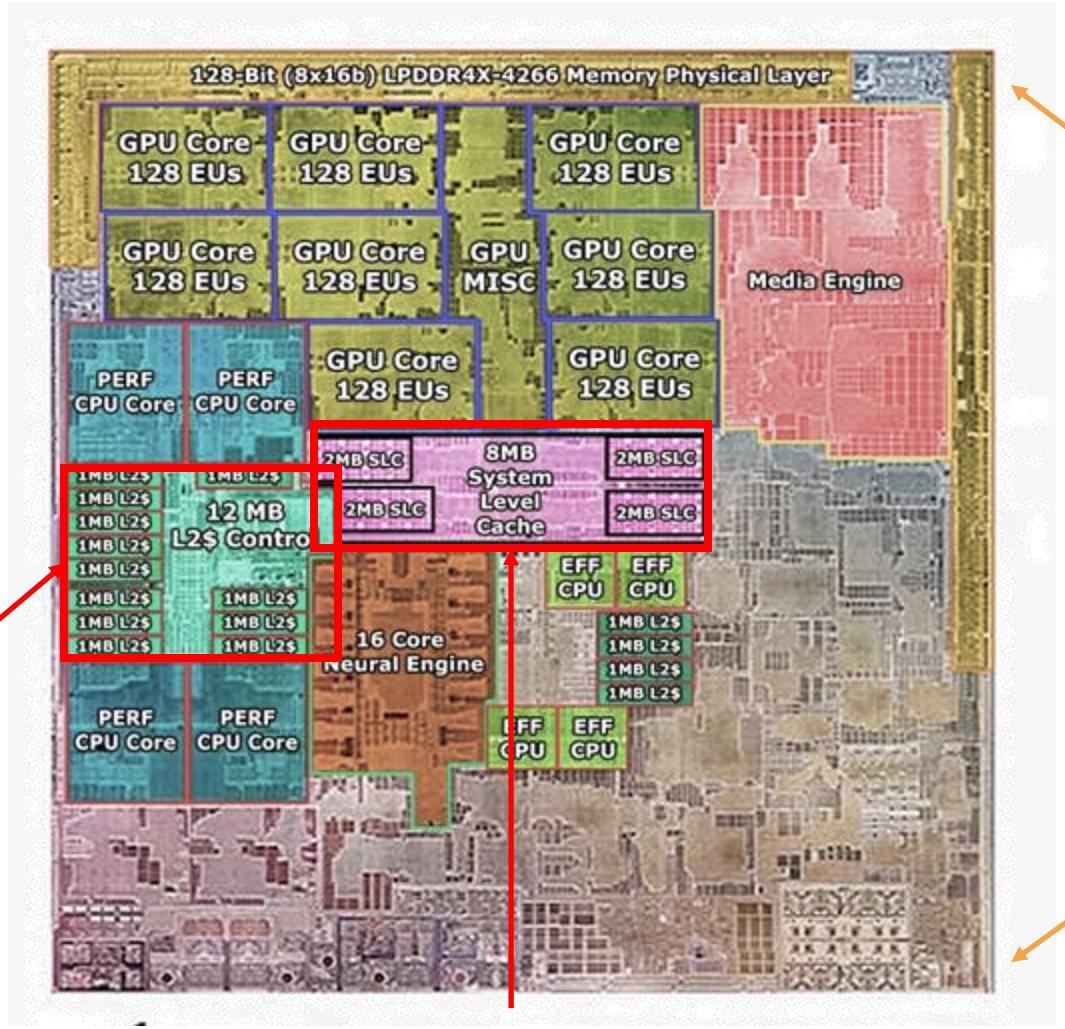


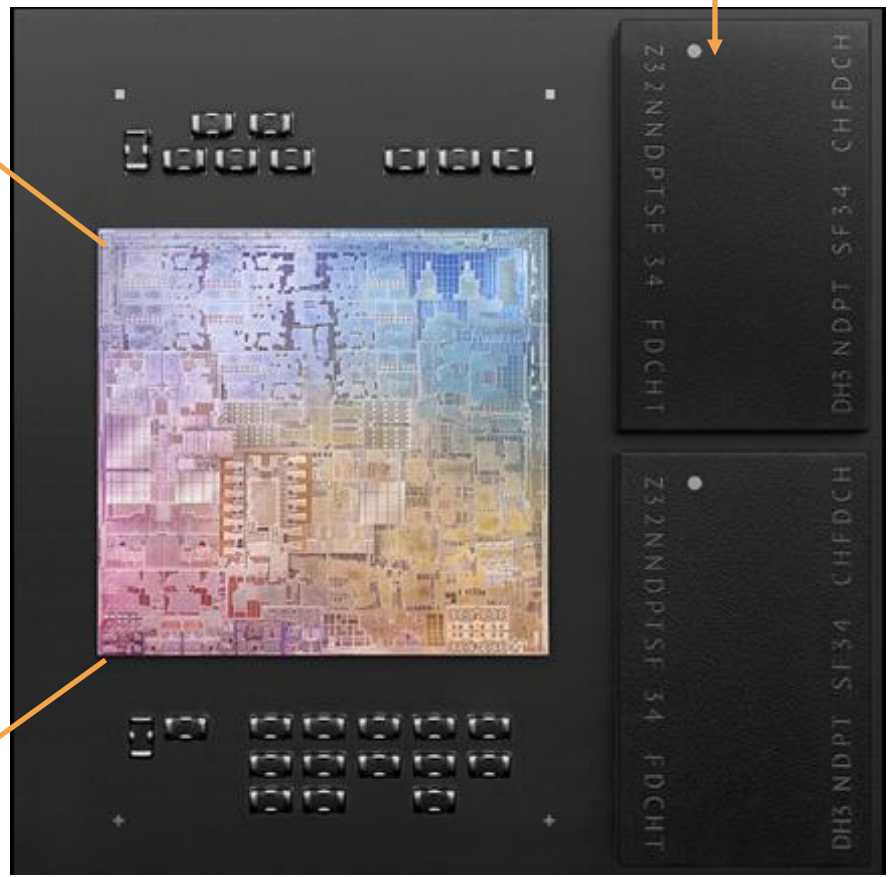
Image Credit: Apple

Apple M1 Memory System

Cache

Two DRAM chips on same package as system SoC





L2 Cache

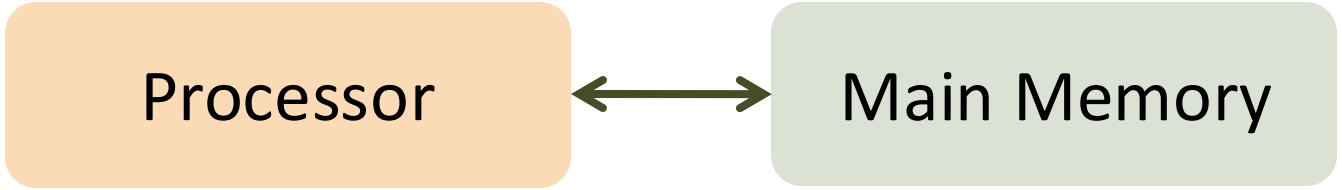
System Level Cache (SLC)

Image Credit: Apple

Agenda

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

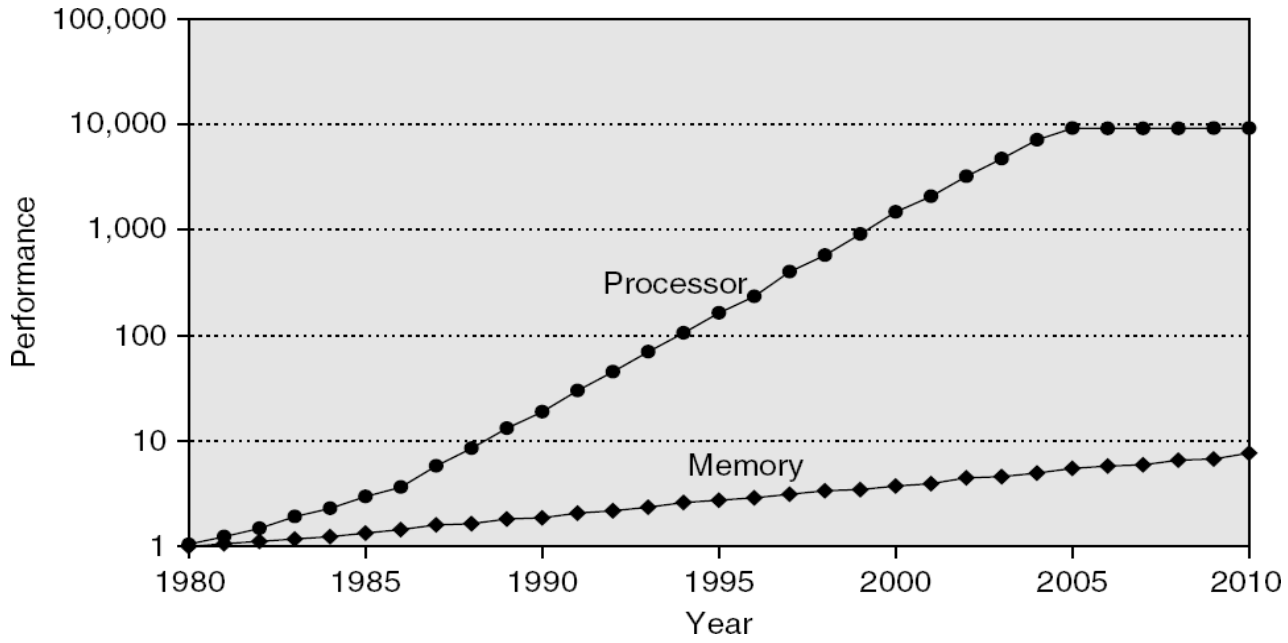
CPU-Memory Bottleneck



- Computer performance is often limited by memory speed (latency and bandwidth)
- **Latency** = delay of one access (memory >> CPU cycle time)
- **Bandwidth** = number of accesses per unit time

Fast CPUs get stuck waiting for slow memory!!

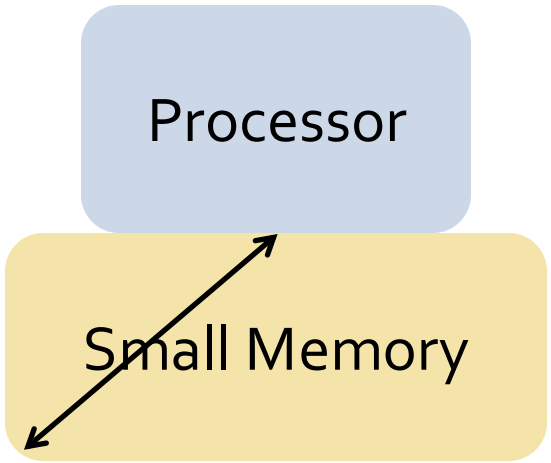
CPU-Memory Latency Gap

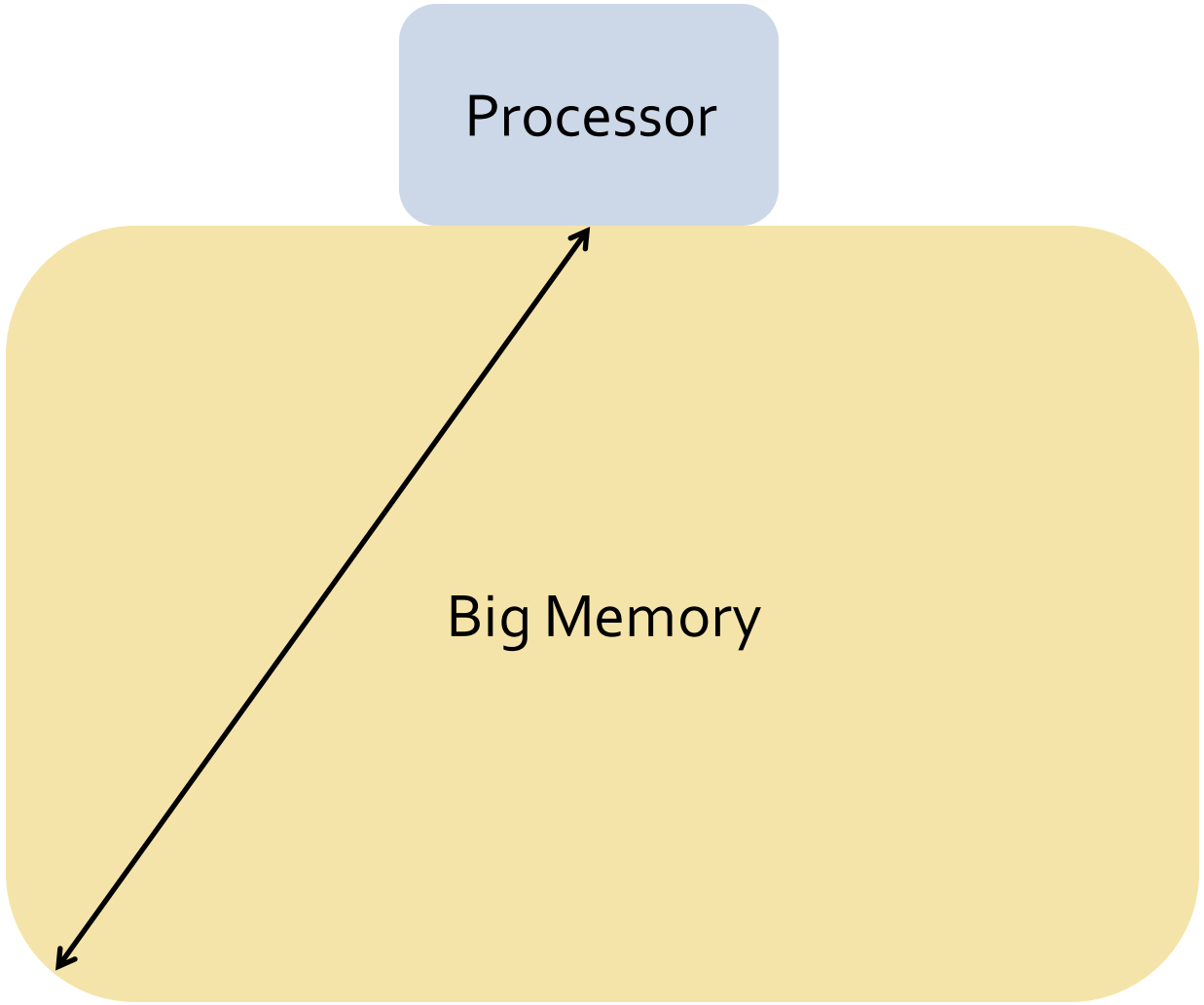


[Hennessy & Patterson 2011]

- Slow memory (DRAM) access severely limits CPU performance
  - 1980 processor exec. ~1 instruction at the same time as DRAM access
  - 2000 processor exec. ~1000 instructions at the same time as DRAM access

Physical Size Affects Latency



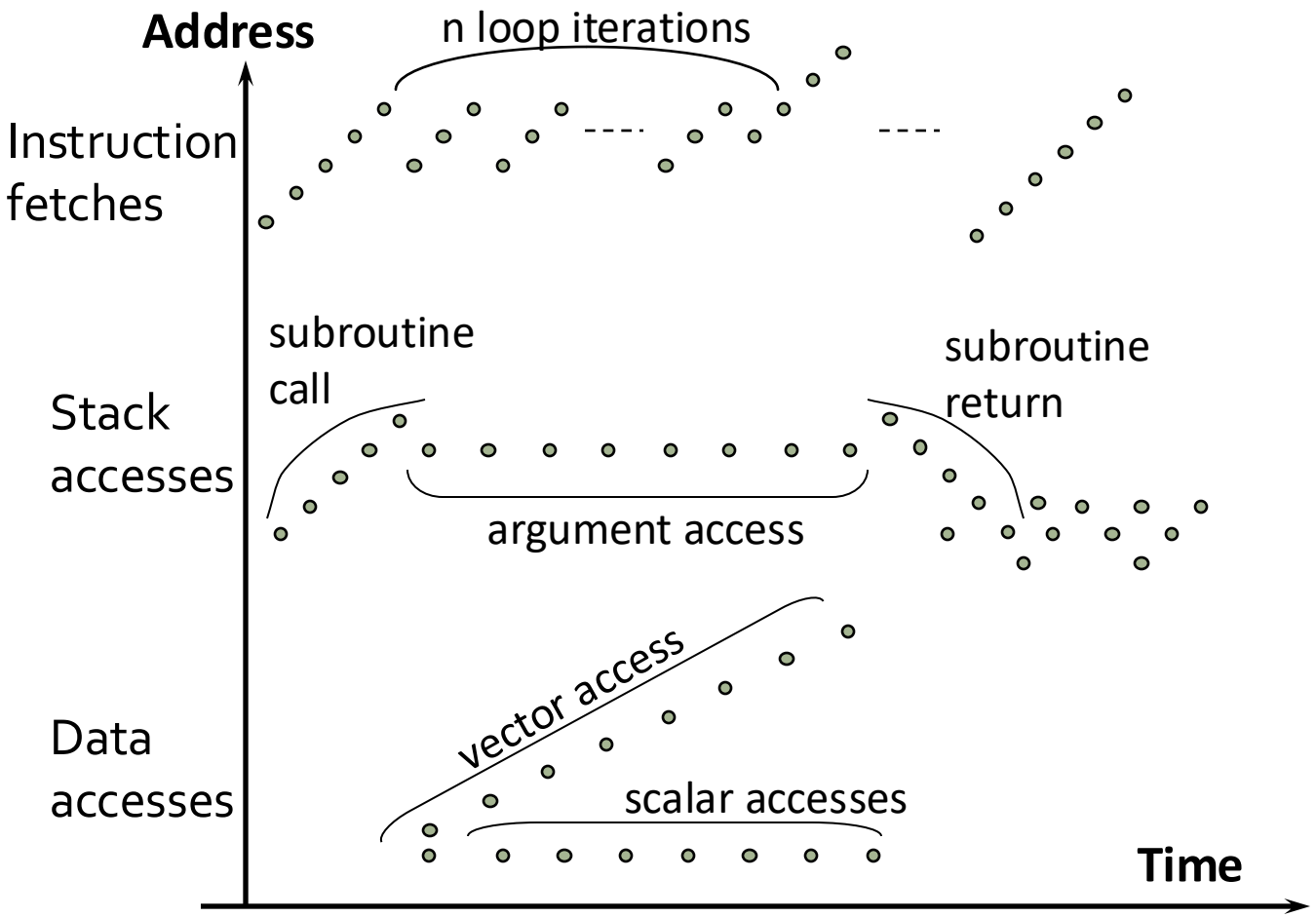


- Signals have further to travel
- Fan out to more locations

Memory Hierarchy

- Capacity: Register << SRAM << DRAM
- Latency: Register << SRAM << DRAM
- Bandwidth: on-chip >> off-chip
- On a data access:
  - if data is in fast memory → low-latency access to SRAM
  - if data is not in fast memory →long-latency access to DRAM
- Memory hierarchies only work if the small, fast memory actually stores data that is reused by the processor

Common And Predictable Memory Reference Patterns



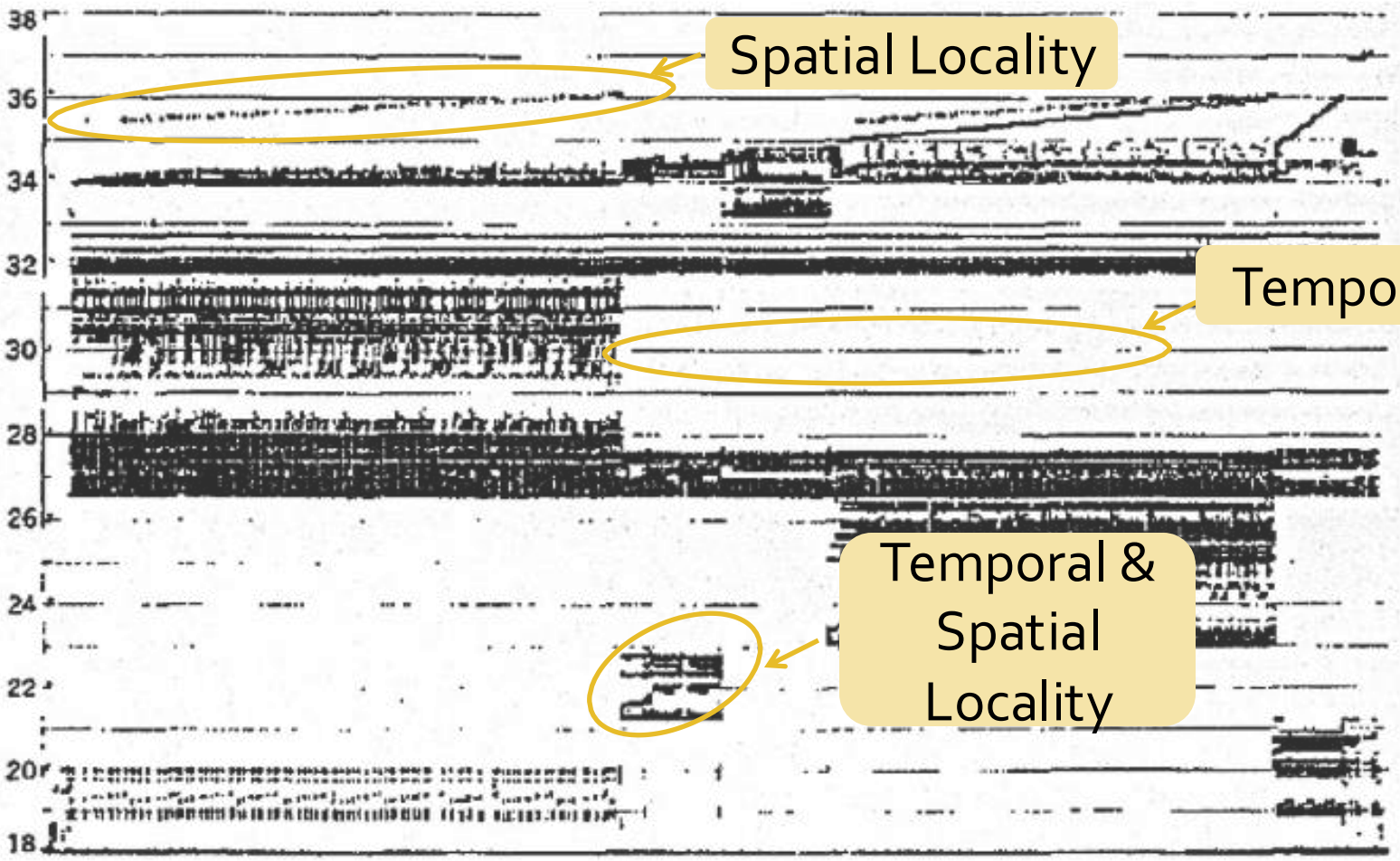
Temporal Locality:

If a location is referenced, it is likely to be referenced again in the near future

Spatial Locality:

If a location is referenced, it is likely that locations near it will be referenced in the near future

Real Memory Reference Patterns



Time (one dot per access to that address at that time)

[From Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168--192 (1971)]

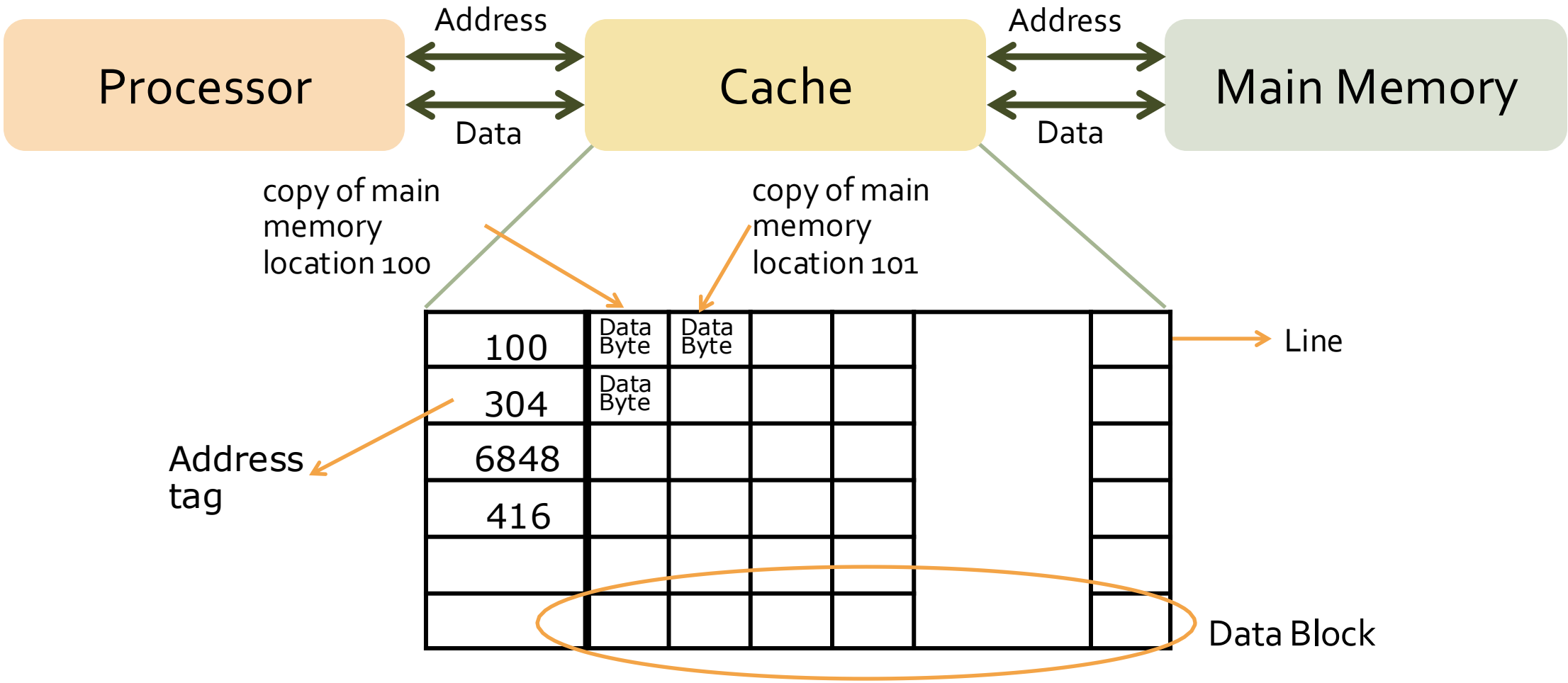
Caches Exploit Both Types of Locality

- Exploit temporal locality by remembering the contents of recently accessed locations
- Exploit spatial locality by fetching blocks of data around recently accessed locations

Agenda

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

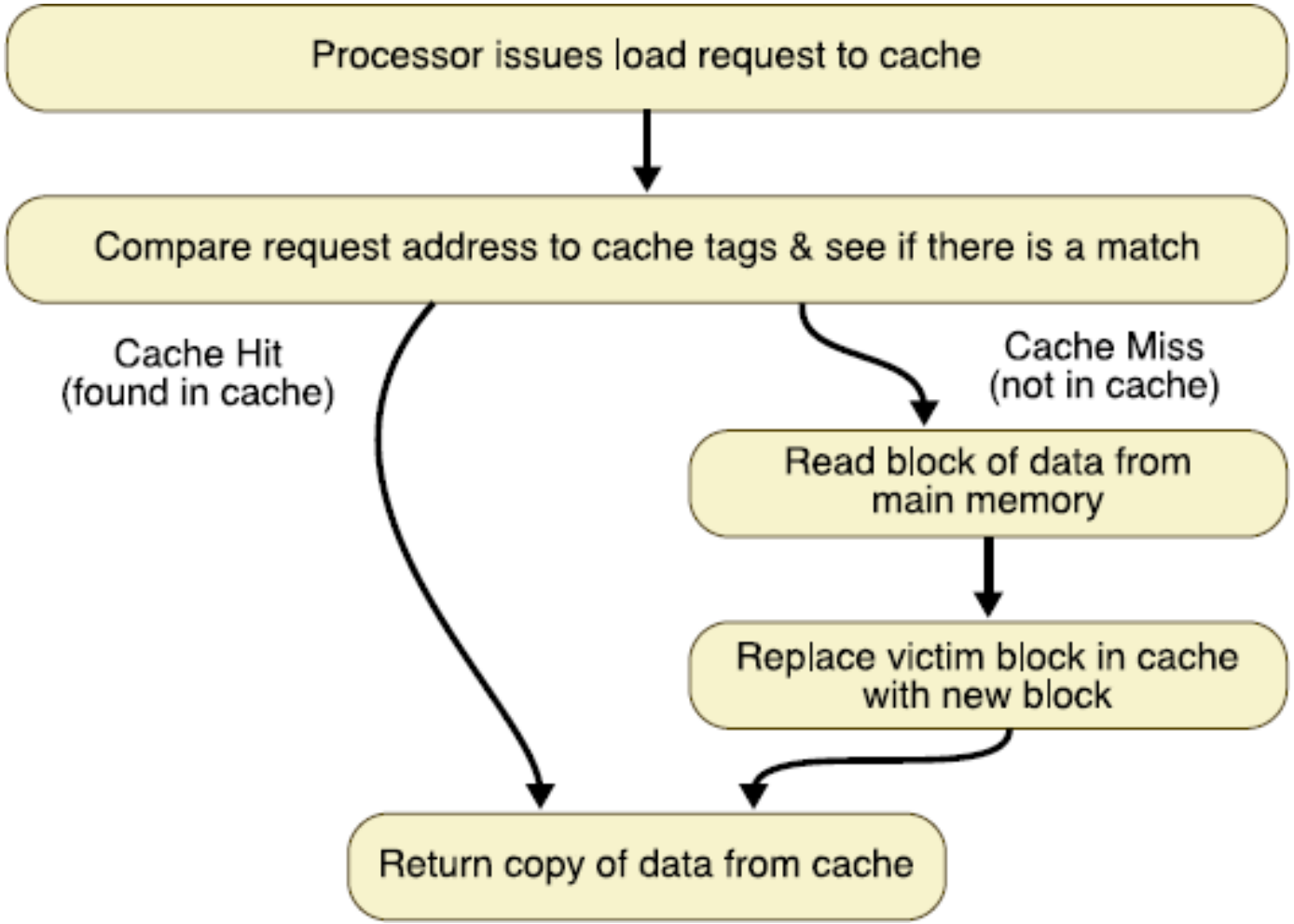
Inside a Cache



Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Basic Cache Algorithm for a Load

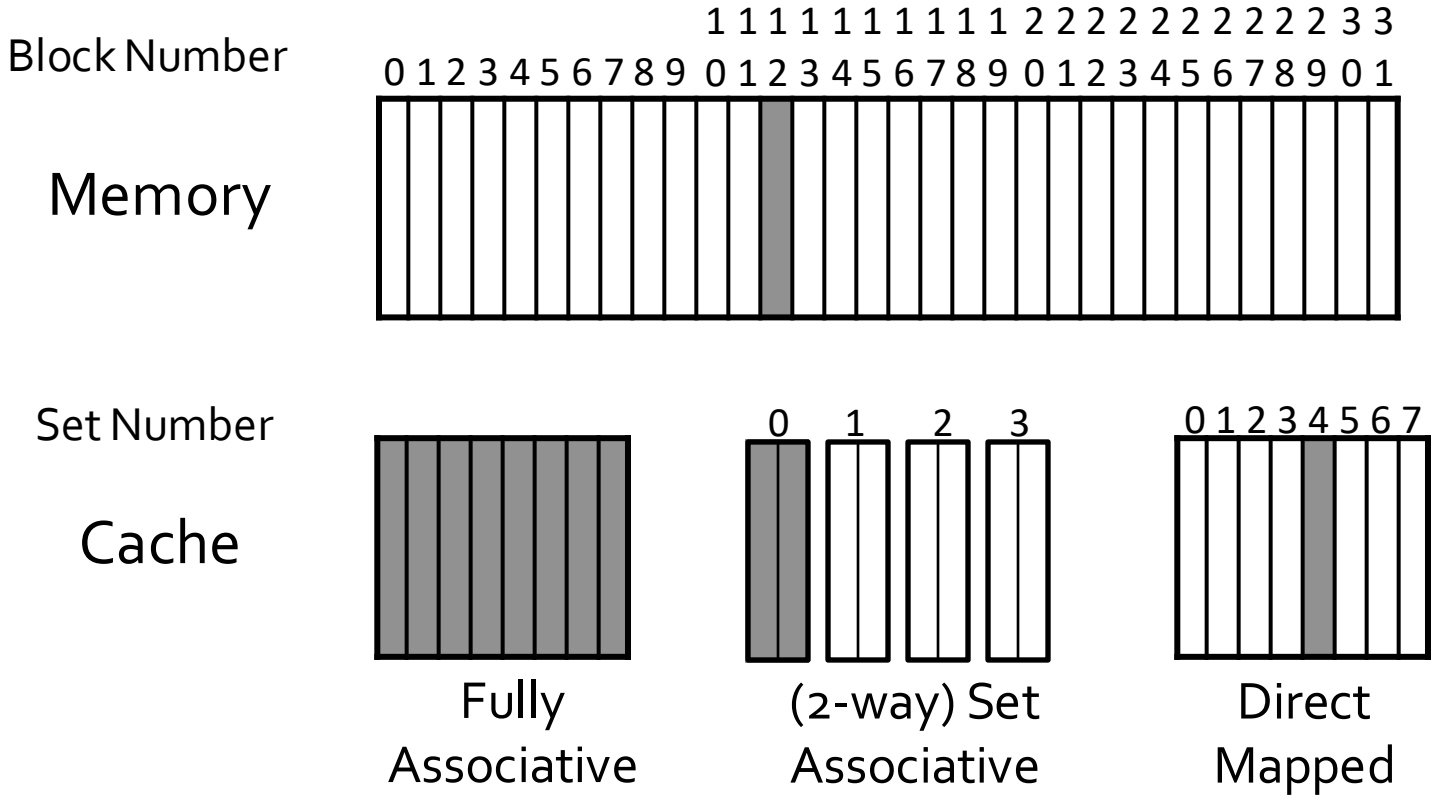


Classifying Caches

- **Block Placement**: Where can a block be placed in the cache?
- **Block Identification**: How a block is found if it is in the cache?
- **Block Replacement**: Which block should be replaced on a miss?
- **Write Strategy**: What happens on a write?

Block Placement:

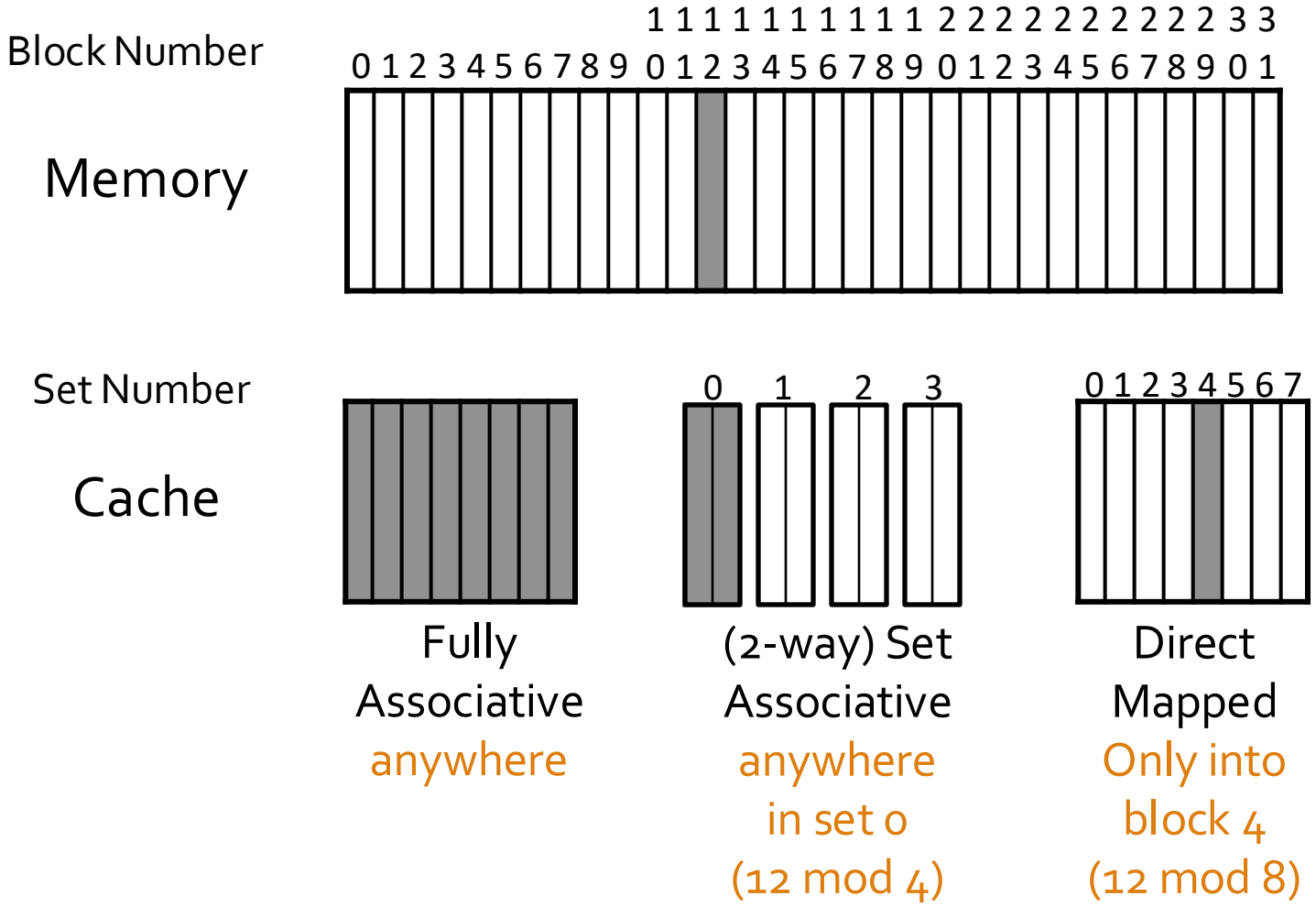
Where Place Block in Cache?



block 12
can be placed

Block Placement:

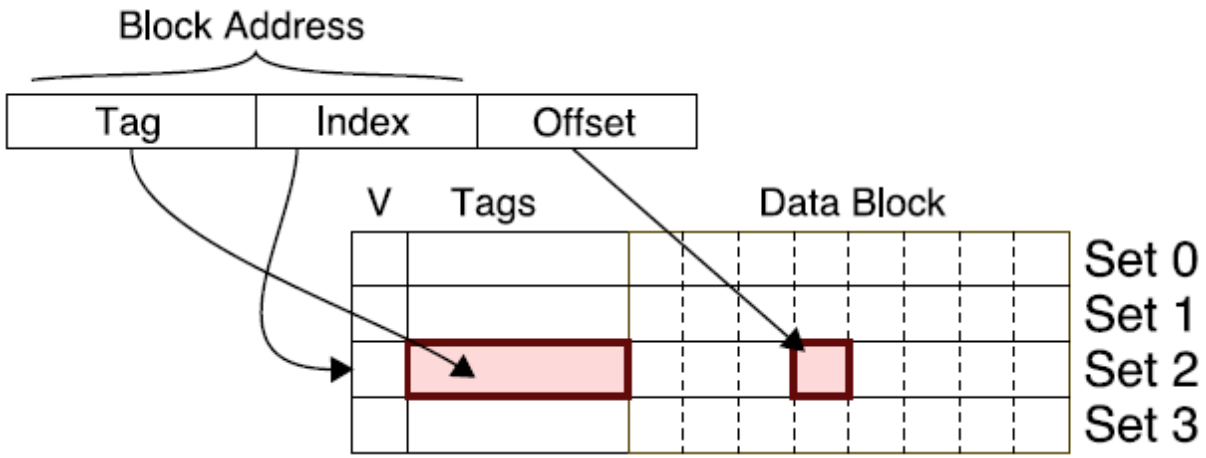
Where Place Block in Cache?



block 12
can be placed

Block Identification:

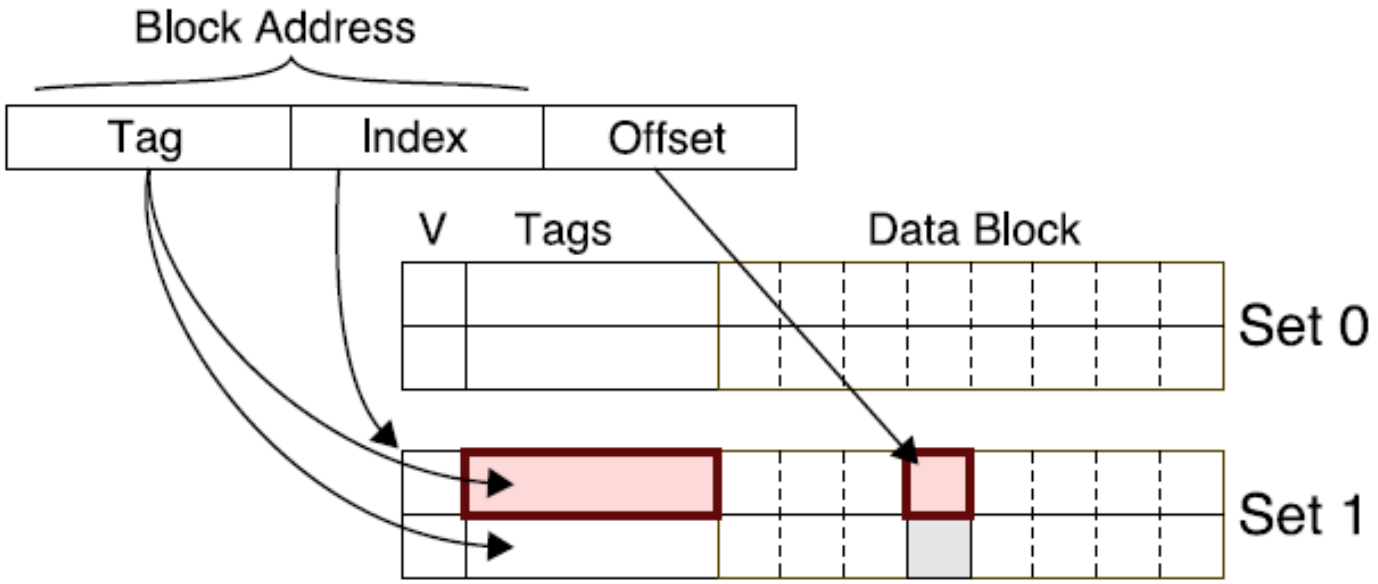
How to find block in cache?



- Cache uses index and offset to find potential match, then checks tag
- Tag check only includes higher order bits
- In this example (Direct--mapped, 8B block, 4 line cache)

Block Identification:

How to find block in cache?



- Cache checks all potential blocks with parallel tag check
- In this example (2-way associative, 8B block, 4 line cache)

Block Replacement:

Which block to replace?

- No choice in a direct mapped cache
- In an associative cache, which block from set should be evicted when the set becomes full?
- **Random**
- **Least Recently Used (LRU)**
  - LRU cache state must be updated on every access
  - True implementation only feasible for small sets (2--way)
  - Pseudo--LRU binary tree often used for 4--8 way
- **First In, First Out (FIFO) aka Round--Robin**
  - Used in highly associative caches
- **Not Most Recently Used (NMRU)**
  - FIFO with exception for most recently used block(s)

Write Strategy:

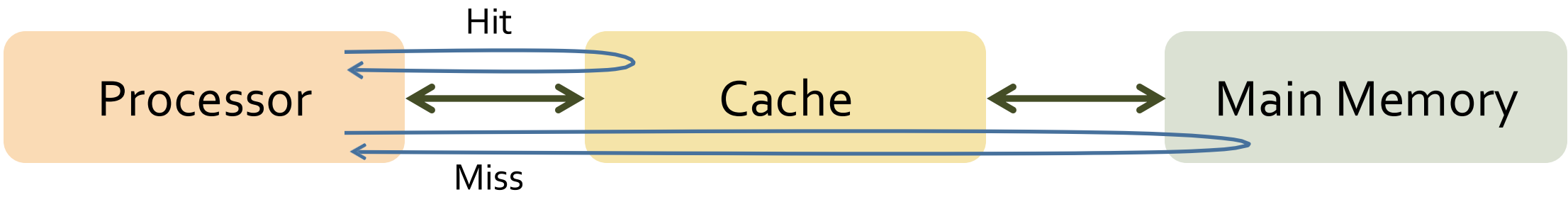
How are writes handled?

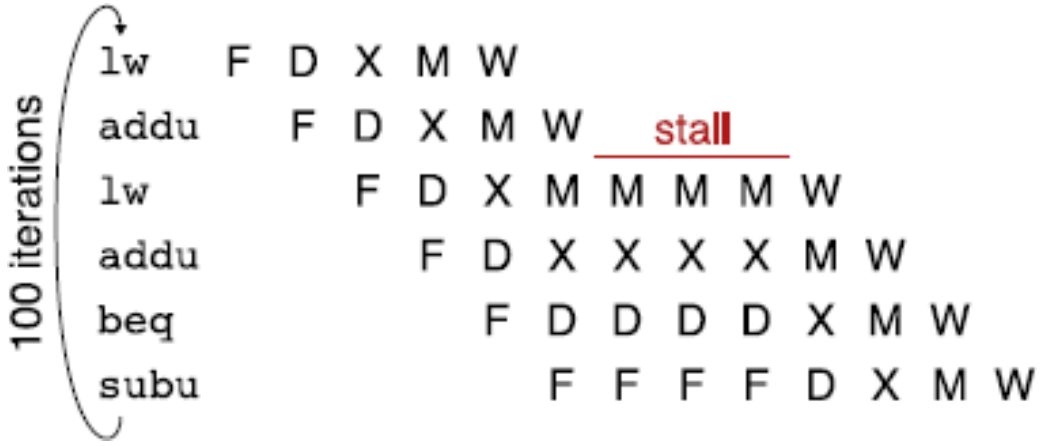
- Cache Hit
  - **Write Through** – write both cache and memory, generally higher traffic but simpler to design
  - **Write Back** – write cache only, memory is written when evicted, dirty bit per block avoids unnecessary write backs, more complicated
- Cache Miss
  - **No Write Allocate** – only write to main memory
  - **Write Allocate** – fetch block into cache, then write
- Common Combinations
  - Write Through & No Write Allocate
  - Write Back & Write Allocate

Agenda

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

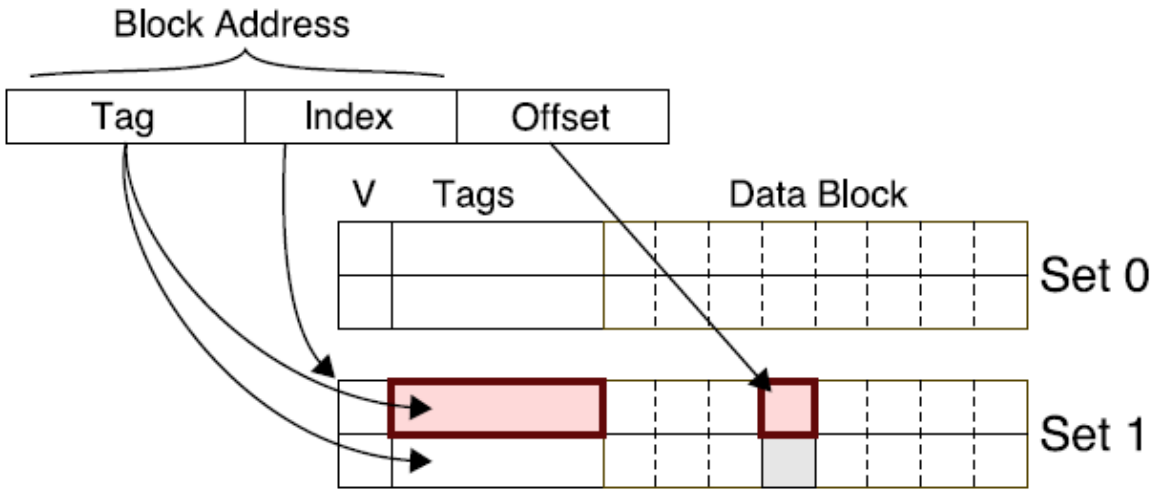
Average Memory Access Time





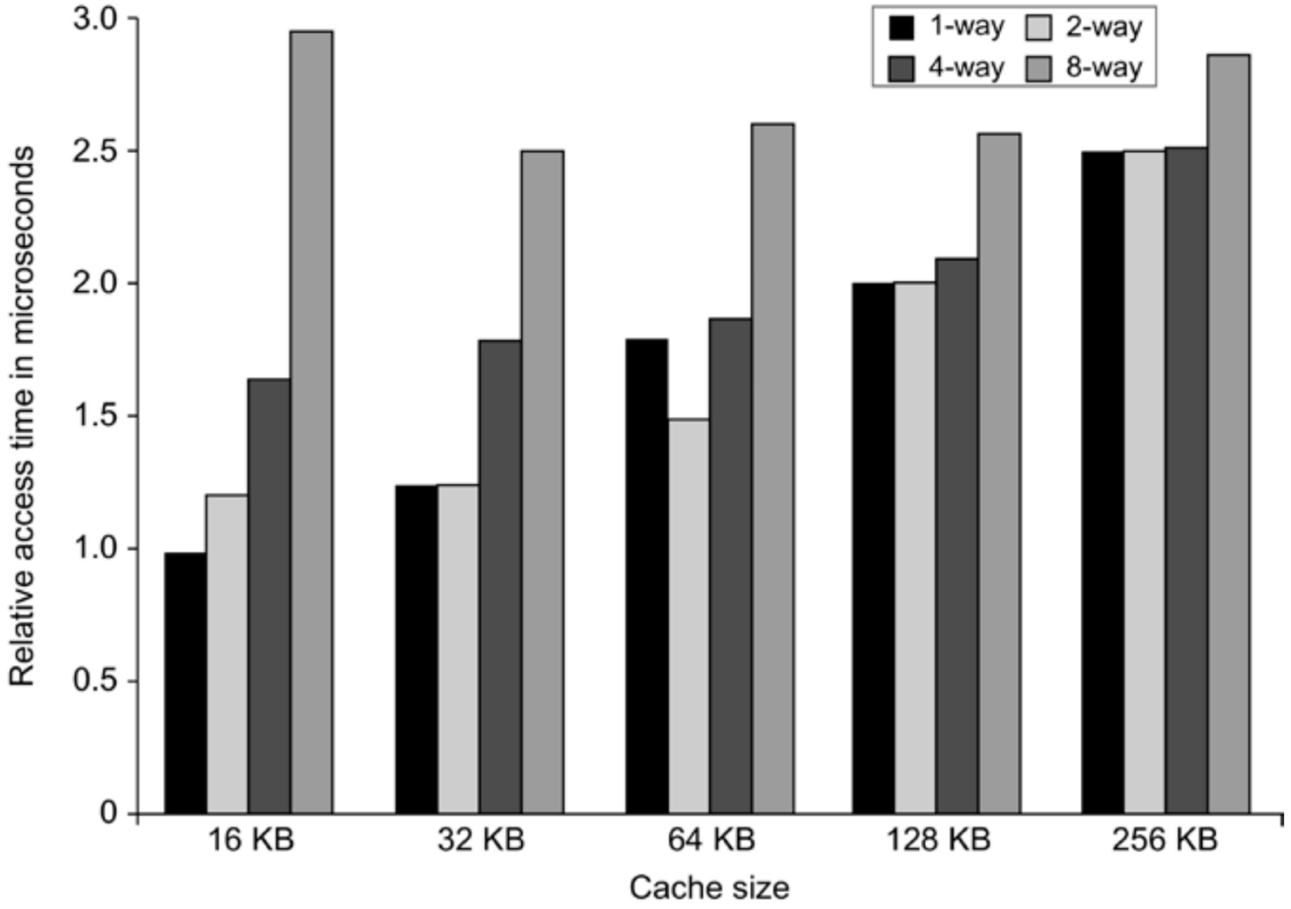
- Average Memory Access Time = Hit Time + (Miss Rate * Miss Penalty)

Categorizing Misses: The Three C's



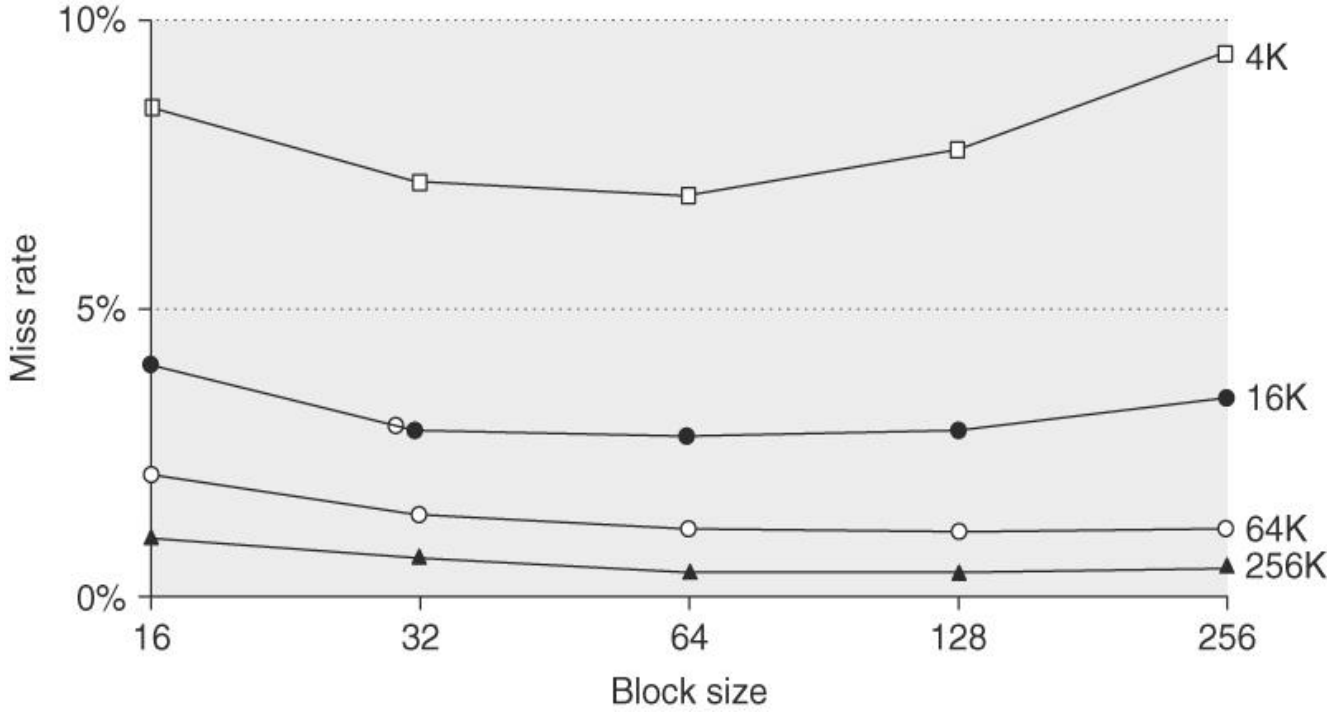
- **Compulsory** – first-reference to a block, occur even with infinite cache
- **Capacity** – cache is too small to hold all data needed by program, occur even under perfect replacement policy (loop over 5 cache lines)
- **Conflict** – misses that occur because of collisions due to less than full associativity (loop over 3 cache lines)

Reduce Hit Time:
Small & Simple Caches



Reduce Miss Rate:

Large Block Size

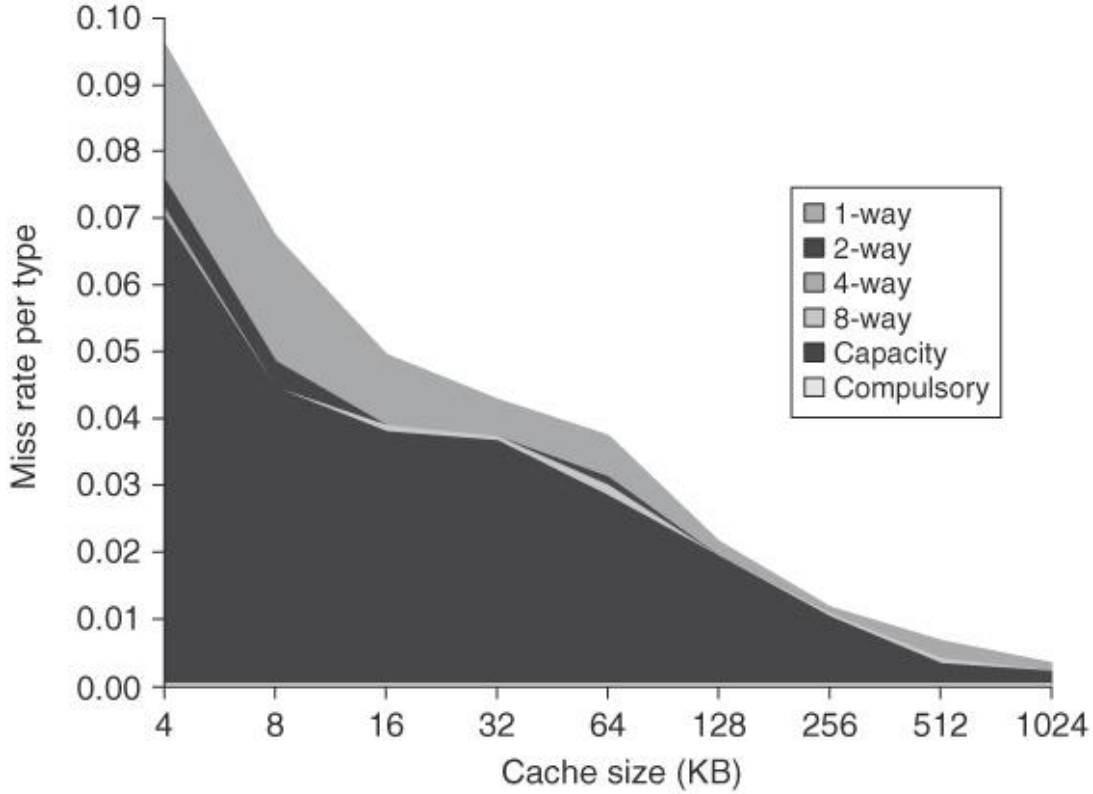


- Less tag overhead
- Exploit fast burst transfers from DRAM
- Exploit fast burst transfers over wide on-chip busses

- Can waste bandwidth if data is not used
- Fewer blocks -> more conflicts

Reduce Miss Rate:

Large Cache Size

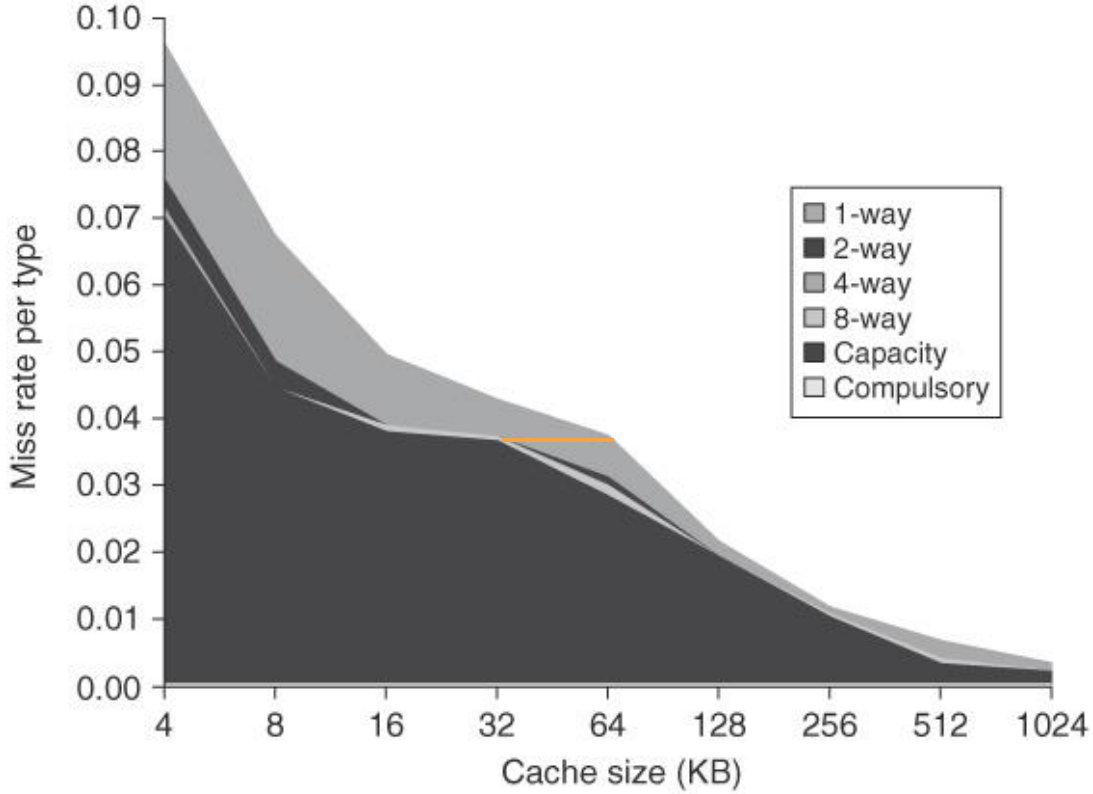


Empirical Rule of Thumb:

If cache size is doubled, miss rate usually drops by about $\sqrt{2}$

Reduce Miss Rate:

High Associativity



Empirical Rule of Thumb:

Direct-mapped cache of size N has about the same miss rate as a two-way set-associative cache of size N/2

Recap

- Pipeline Review
  - Pipelining Basics
  - Hazards
    - Structural Hazards
    - Data Hazards
    - Control Hazards
- Cache Review
  - Memory Technology
  - Motivation for Caches
  - Classifying Caches
  - Cache Performance

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)
  - David Wentzlaff (Princeton)
- MIT material derived from course 6.823
- UCB material derived from course CS252
- Cornell material derived from course ECE 4750
- Princeton material derived from course ECE 475