

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2022

Advanced Processor Architecture

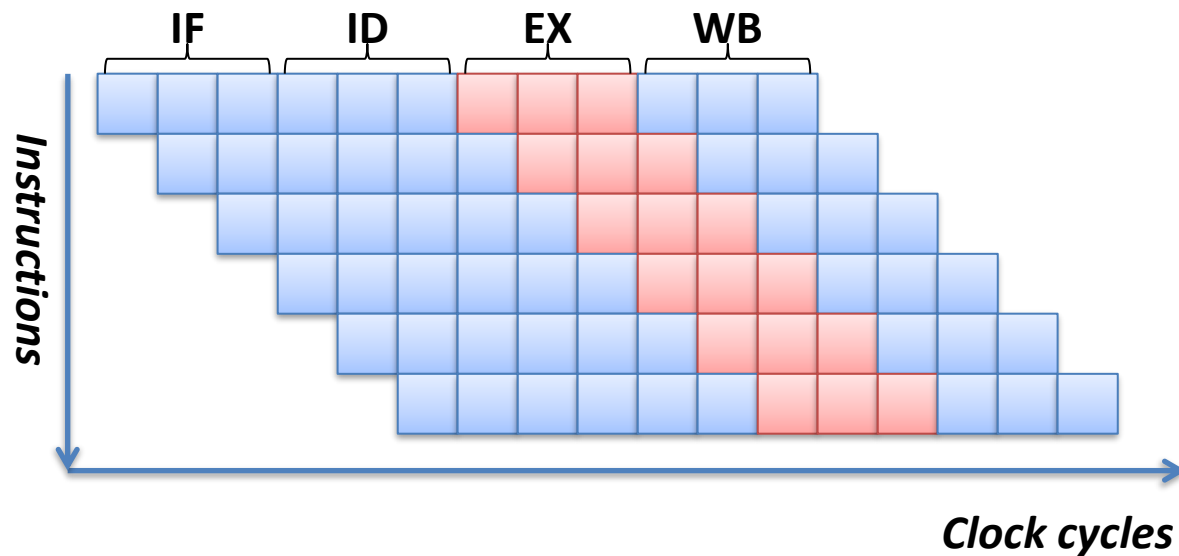
Chap. 4.10 – 11, 4.14 – 15



Superpipelined vs. Multiple-Issue

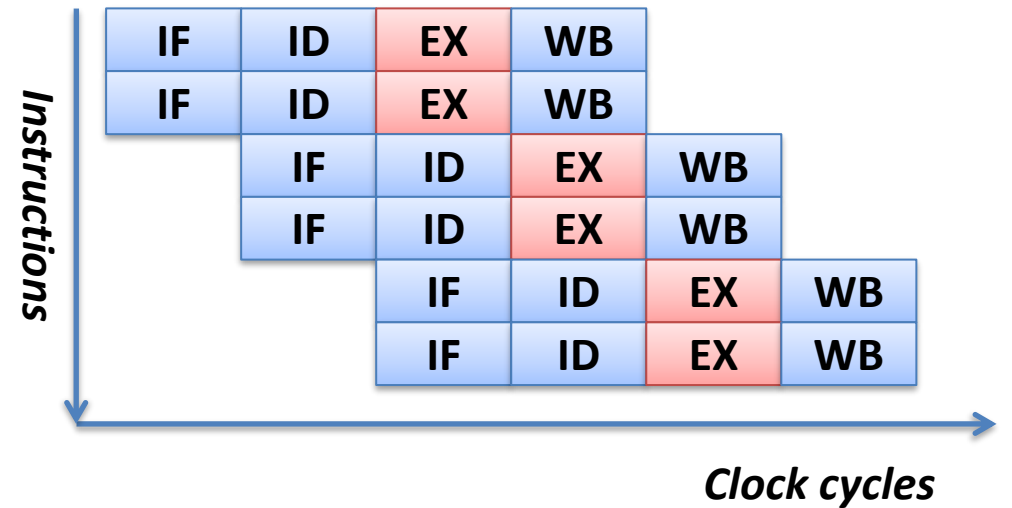
▪ Superpipelined

- Subdivide each pipeline stage
- Higher clock speed



▪ Multiple-issue

- Execute multiple instructions in parallel
- The EX stage has many functional units

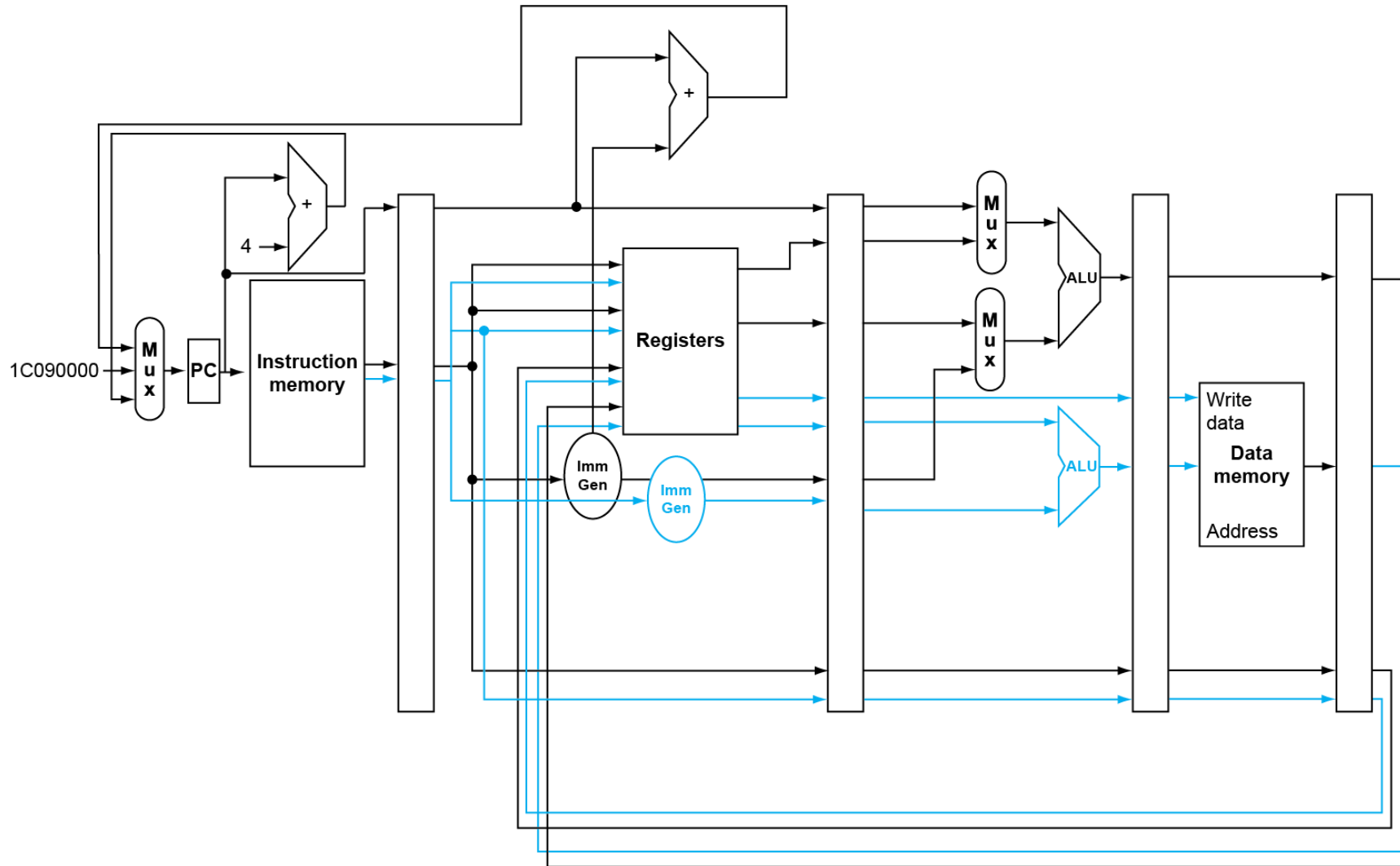


RISC-V with Static Dual Issue

- Two-issue packets
 - 64-bit aligned: One ALU/branch instruction + One load/store instruction
 - Pad an unused instruction with nop
 - Additional hardware:
 - +2 read / +1 write ports in register file
 - Separate adder for computing the effective address for memory

Address	Instruction type	Pipeline stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

RISC-V with Static Dual Issue



Scheduling Example

- Schedule this for dual-issue RISC-V

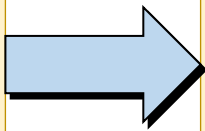
```
Loop: lw    x31, 0(x20)    // x31 = array element
      add   x31, x31, x21  // add scalar in x21
      sw    x31, 0(x20)    // store result
      addi  x20, x20, -4   // decrement pointer
      blt   x22, x20, Loop // branch if x22 < x20
```

	ALU/branch	Load/store	Cycle
Loop:	nop	lw x31, 0(x20)	1
	addi x20, x20, -4	nop	2
	add x31, x31, x21	nop	3
	blt x22, x20, Loop	sw x31, 4(x20)	4

- $IPC = 5/4 = 1.25$ (cf. peak $IPC = 2$)

Loop Unrolling

```
Loop: lw    x31, 0(x20)
      add   x31, x31, x21
      sw    x31, 0(x20)
      addi  x20, x20, -4
      blt   x22, x20, Loop
```



```
Loop: lw    x31, 0(x20)
      add   x31, x31, x21
      sw    x31, 0(x20)

      lw    x31, -4(x20)
      add   x31, x31, x21
      sw    x31, -4(x20)

      lw    x31, -8(x20)
      add   x31, x31, x21
      sw    x31, -8(x20)

      lw    x31, -12(x20)
      add   x31, x31, x21
      sw    x31, -12(x20)

      addi  x20, x20, -16
      blt   x22, x20, Loop
```

Loop Unrolling Scheduled Example

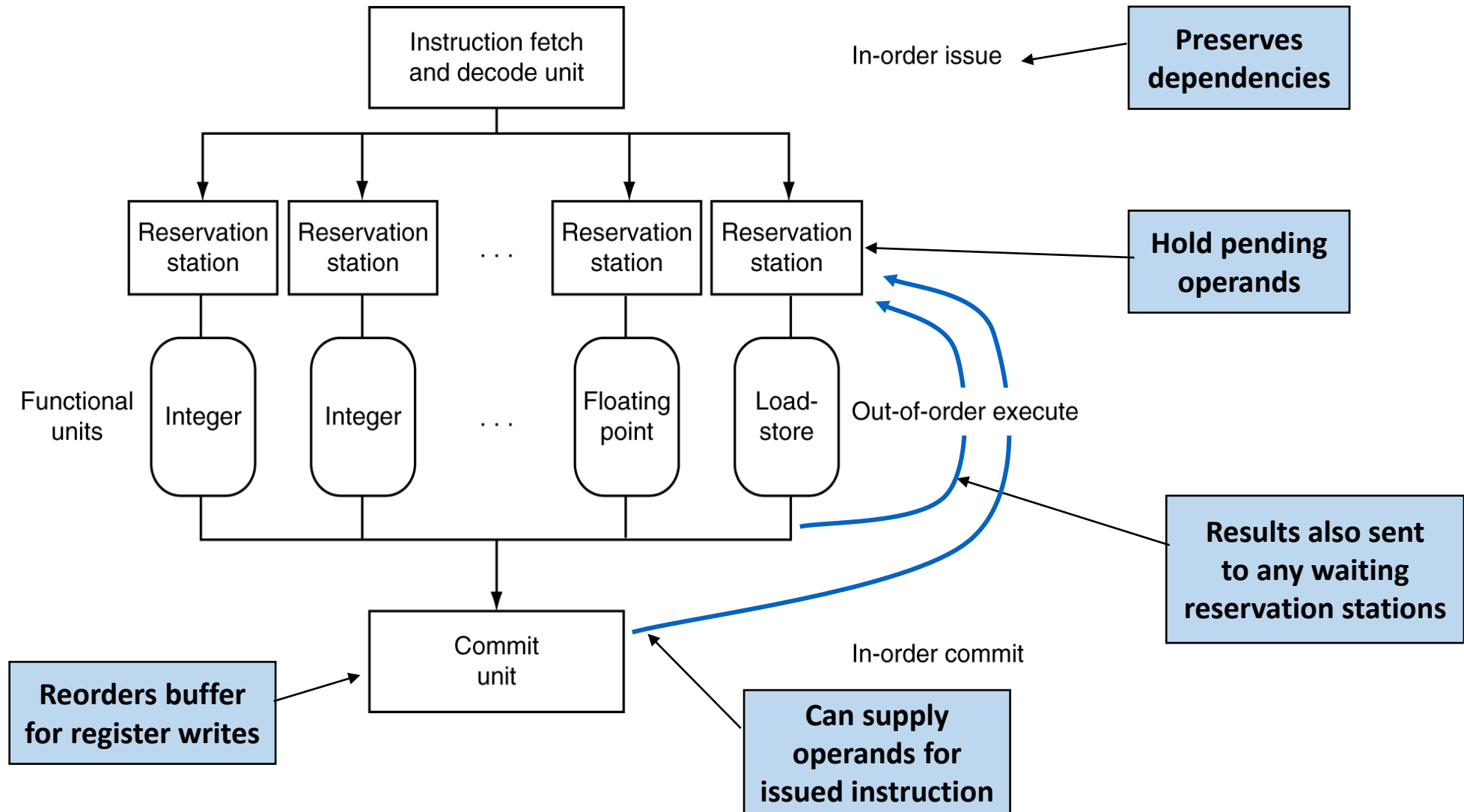
	ALU/branch	Load/store	Cycle
Loop:	addi x20, x20, -16	lw x28, 0(x20)	1
	nop	lw x29, 12(x20)	2
	add x28, x28, x21	lw x30, 8(x20)	3
	add x29, x29, x21	lw x31, 4(x20)	4
	add x30, x30, x21	sw x28, 16(x20)	5
	add x31, x31, x21	sw x29, 12(x20)	6
	nop	sw x30, 8(x20)	7
	blt x22, x20, Loop	sw x31, 4(x20)	8

- $IPC = 14/8 = 1.75$
 - Closer to 2, but at cost of registers and code size

Why Dynamic Scheduling?

- Why not just let the compiler schedule code?
- Not all stalls are predictable
 - e.g., cache misses
- Can't always schedule around branches
 - Branch outcome is dynamically determined
- Different implementations of an ISA have different latencies and hazards

Dynamically Scheduled CPU



Speculation

- "Guess" what to do with an instruction
 - Start operation as soon as possible
 - Check whether guess was right → If not, roll-back and do the right thing
- Speculate on branch outcome
 - Predict branch and continue issuing
 - Don't commit until branch outcome determined
- Speculate on load
 - Avoid load and cache miss delay
 - Predict the effective address or loaded value
 - Load before completing outstanding stores
 - Bypass stored values to load unit
 - Don't commit load until speculation cleared

Beyond Pipelining

- Hyperthreading
- SIMD (MMX, SSE)
- Vector processor (AVX-512)

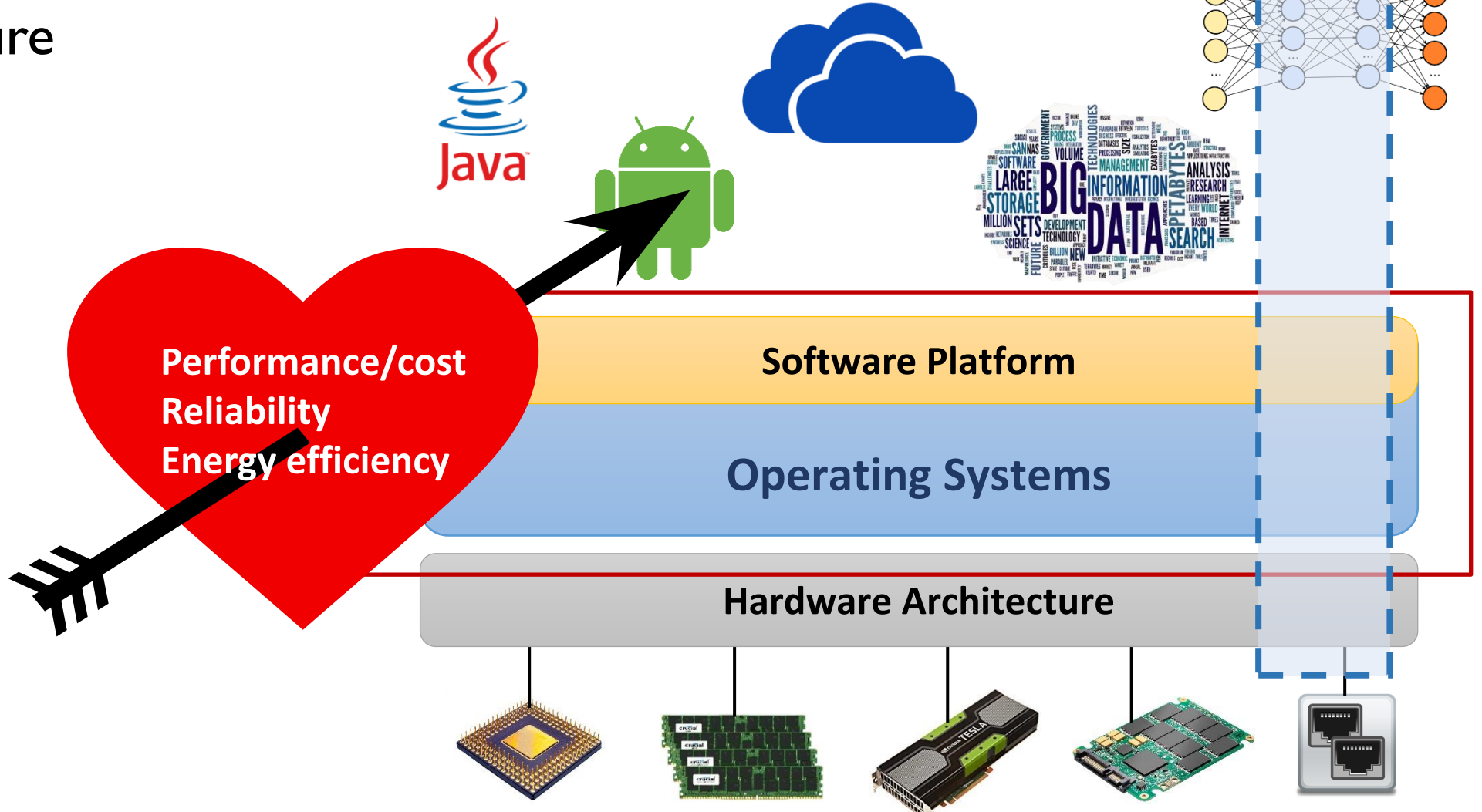
- Manycore (Xeon Phi: Knights Ferry / Corner / Landing / Hill)
 - Knights Landing (KNL): 72 cores, 4 threads / core
- GPUs
 - Nvidia H100 (Hopper): 16896 CUDA cores + 528 tensor cores
- Accelerators
 - Amazon EC2 F1 instances: up to 8 Xilinx FPGAs (each with 2.5M logic elements)

What We Have Learned

- Instruction Set Architecture (i.e., abstraction of hardware)
- Representing numbers: integer and floating-point
- RISC-V assembly and how to translate C program into it
- Basic processor organization
- Pipelining
- Branch prediction
- Locality and memory hierarchy
- Caches
- Program optimization for caches
- Virtual memory
- Performance evaluation

Computer Systems Research

- Architecture
- OS
- Compiler
- Network
- Database
- ...
- +
- Security



*"domain-specific computing"
or "vertical optimization"*