Lecture 25: Interrupt Handling and Multi-Data Processing

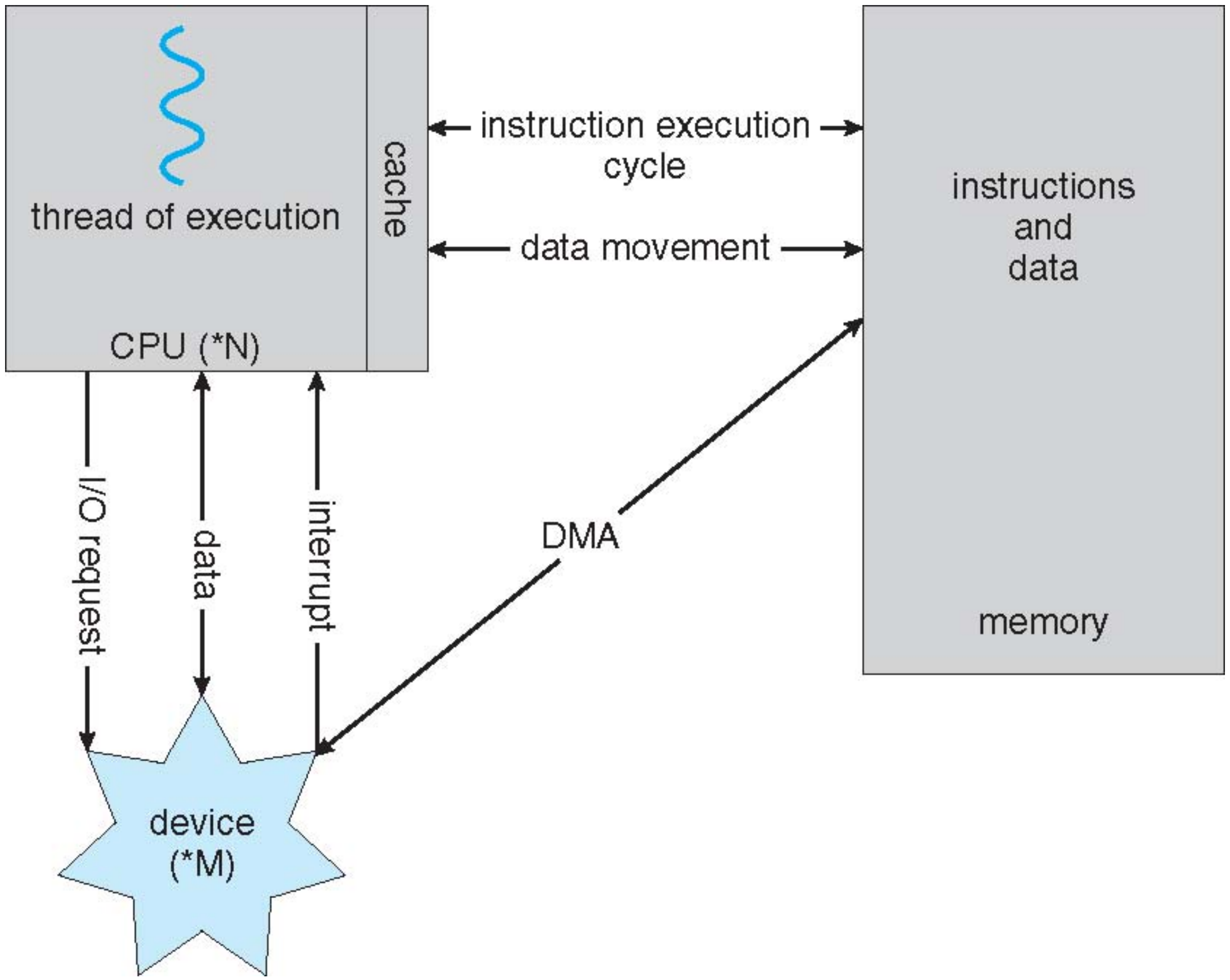
Spring 2024
Jason Tang

Topics

- Interrupt handling
- Vector processing
- Multi-data processing

I/O Communication

- Software needs to know when:
  - I/O device has completed an operation
  - I/O device had an error
- Software can either:
  - Repeatedly poll device (using programmed I/O)
  - Wait for I/O interrupt notification



Operating System Concepts, Silberschatz, Galvin, and Gagne

Data Transfers

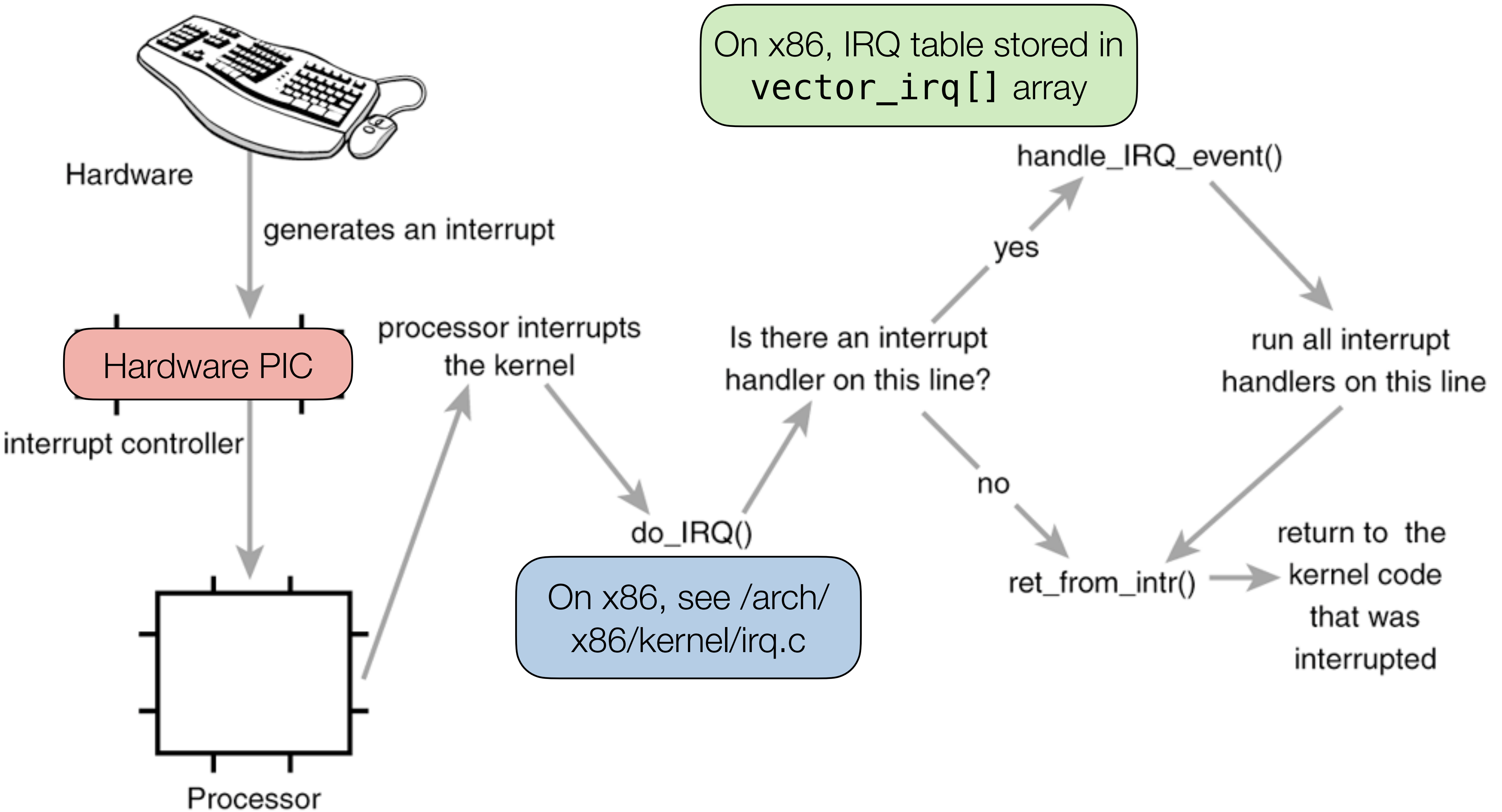
	Programmed I/O	Interrupt Driven
Advantage	Simple to implement, processor in complete control	Main software keeps running while data actual transfers
Disadvantage	No software processing while waiting for I/O response	Device must raise interrupt, processor must detect and handle interrupt

- Programmed I/O is best for frequent, small data transfers
- Interrupt driven is best when transfers are infrequent, and when a dedicated DMA engine handles transfers of large blocks of data
    - Use PIO when the amount of data to transfer is less than overhead of creating and initiating a TxD

Interrupt Handling

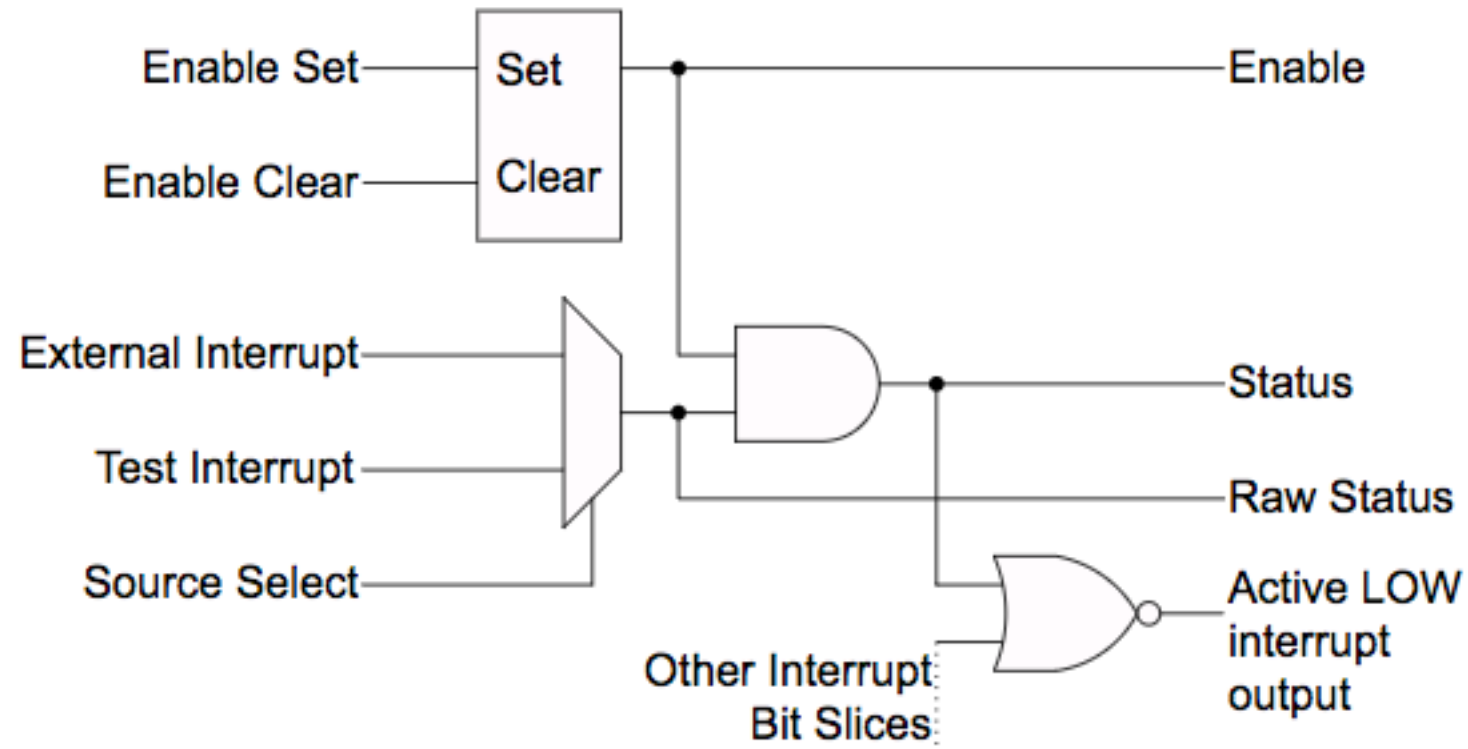
- Hardware sends an electrical signal on a physical interrupt line
- Processor detects that signal and translates it into an interrupt request (IRQ) number
- Processor then jumps to interrupt handling code
- Software searches through its interrupt request table (stored in RAM) for entry or entries that match the IRQ
- If found, software jumps to the registered interrupt service routine (ISR)
- If not found, software ignores interrupt

Linux Kernel IRQ Handling



Programmable Interrupt Controller

- Hardware component that collects signals from peripherals

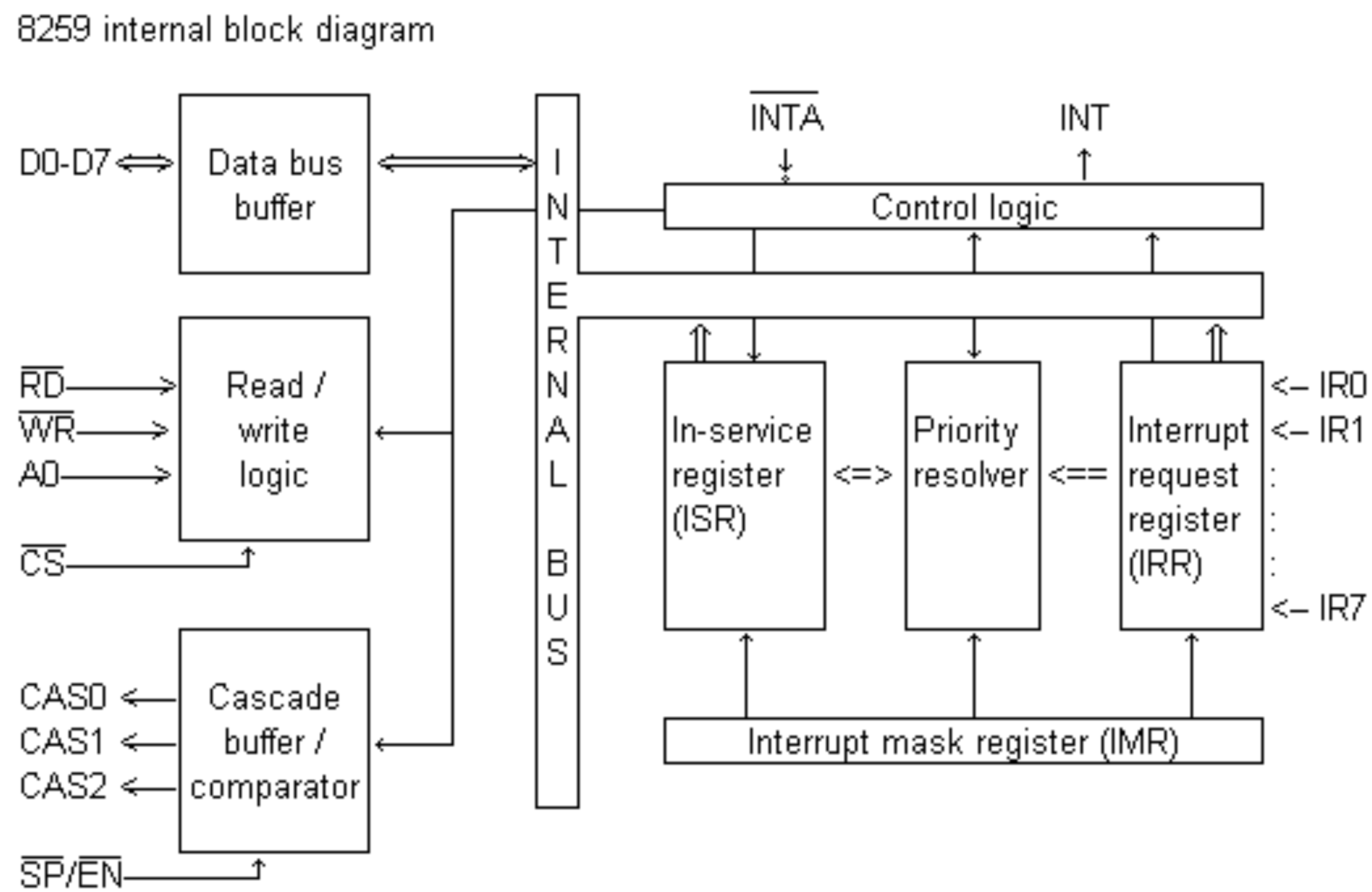


- Contains multiple enable registers, one per interrupt source
    - PIC forwards enabled interrupts to the processor
    - PIC ignores masked interrupts
- Can also prioritize output, when multiple devices raise interrupts simultaneously

ASB Example AMBA SYstem Technical Reference Manual, §4.1.2

8259 PIC

- Original programmable interrupt controller for Intel-based computers
- Has 8 inputs, organized by priority
  - When an unmasked input is raised and an no other interrupt is pending, then PIC raises interrupt line to CPU
  - Superseded by Advanced Programmable Interrupt Controller (APIC)



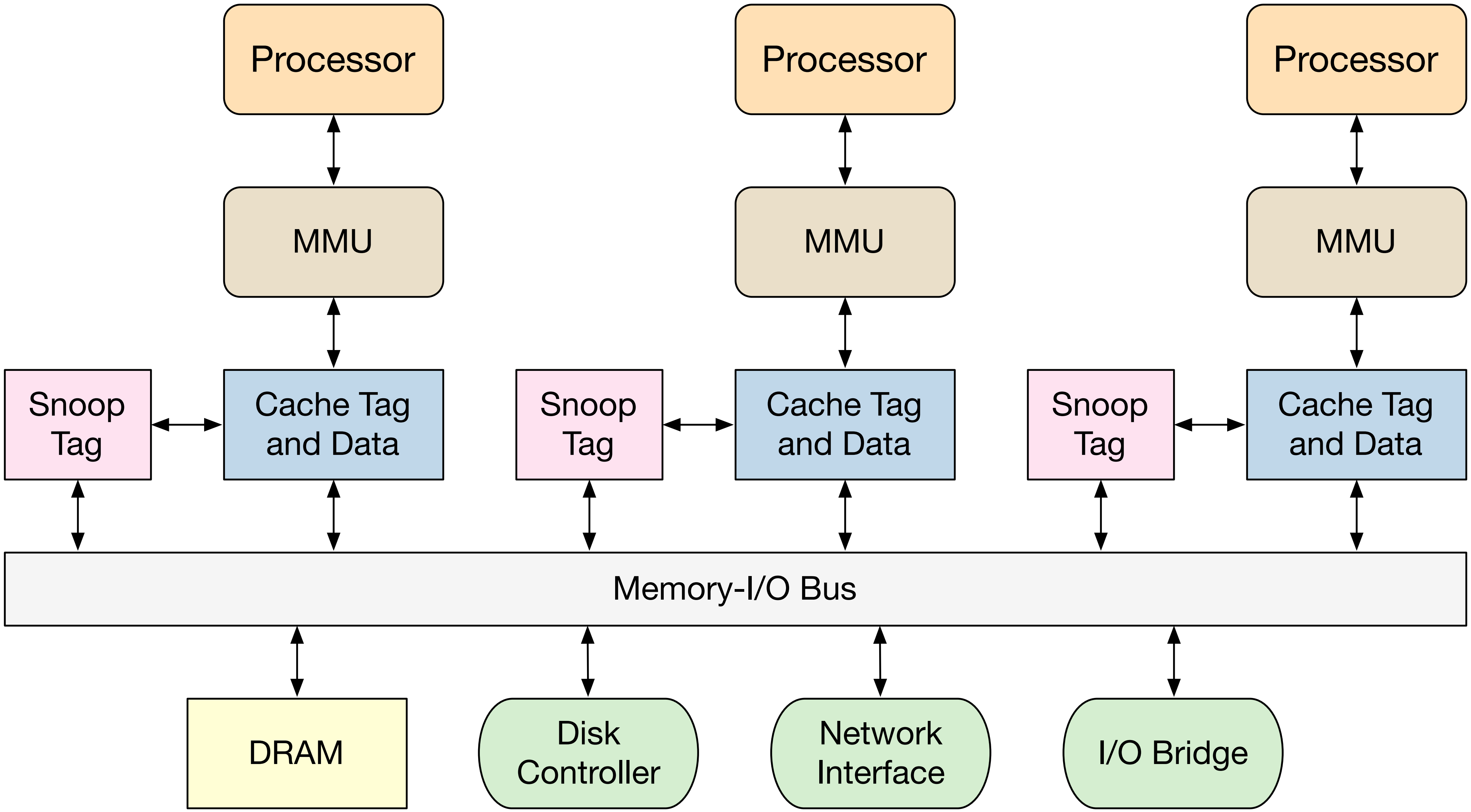
http://www.thesatya.com/8259.html

Message Signaled Interrupts

- Newer alternative to line-based interrupts
- Instead of having dedicated wires to trigger interrupts, a device triggers interrupt by *writing* to a special memory address
  - Number of interrupts no longer constrained by size of PIC
  - Operating system does not need to poll devices to determine source of interrupt, when multiple devices are on a shared interrupt line
- Used by modern buses, like PCIe

https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/using-message-signaled-interrupts

Parallel Processing



- Modern computers are multiprocessors, to simultaneously execute multiple programs

Multiprocessors

	A4	A8	A10X Fusion	A12 Bionic	A15 Bionic
Device	iPhone 4	iPhone 6 / 6+	iPad Pro (2nd Gen)	iPhone XS / XR	iPhone 13
CPU Core(s)	Cortex-A8	Typhoon	Hurricane / Zephyr	Vortex / Tempest	Avalanche / Bionic
CPU Freq	0.8 GHz	1.1 GHz	2.36 GHz	2.49 GHz	3.2 GHz
Cores	1	2	3 / 3	2 / 4	2 / 4

- Multicore systems common in modern computers
- Improvement in throughput by adding more cores is limited by **Amdahl's Law**
  - Modern software can be written to take advantage of multiple processors

Parallel Processing

- Many scientific and engineering problems involve looping over an array, to perform some computation over each element

```c
for (i = 0; i < 64; i++)
    a[i] = b[i] + s;
```

```
// x0 = s, x1 = i,
// x3 = a, x4 = b
top:
    ldr w2, [x1, x3]
    add w2, w2, w0
    str w2, [x1, x4]
    add x1, x1, 4
    cmp x1, #64
    b.ne top
```

- Repeatedly fetching the same instruction wastes a lot of cycles
    - More efficient to have processor automatically perform operation across a vector of data

Flynn's Taxonomy

		Instruction Streams	
		One	Many
Data Streams	One	SISD	MISD
	Many	SIMD	MIMD

- Classification of computer architectures, based upon how the processor/processors handle datum/data
  - Instruction Stream: number of processing unit(s), executing instruction(s)
  - Data Stream: number of data value(s) that the processing unit(s) are acting upon
- Traditional single core system is SISD

https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/

SIMD

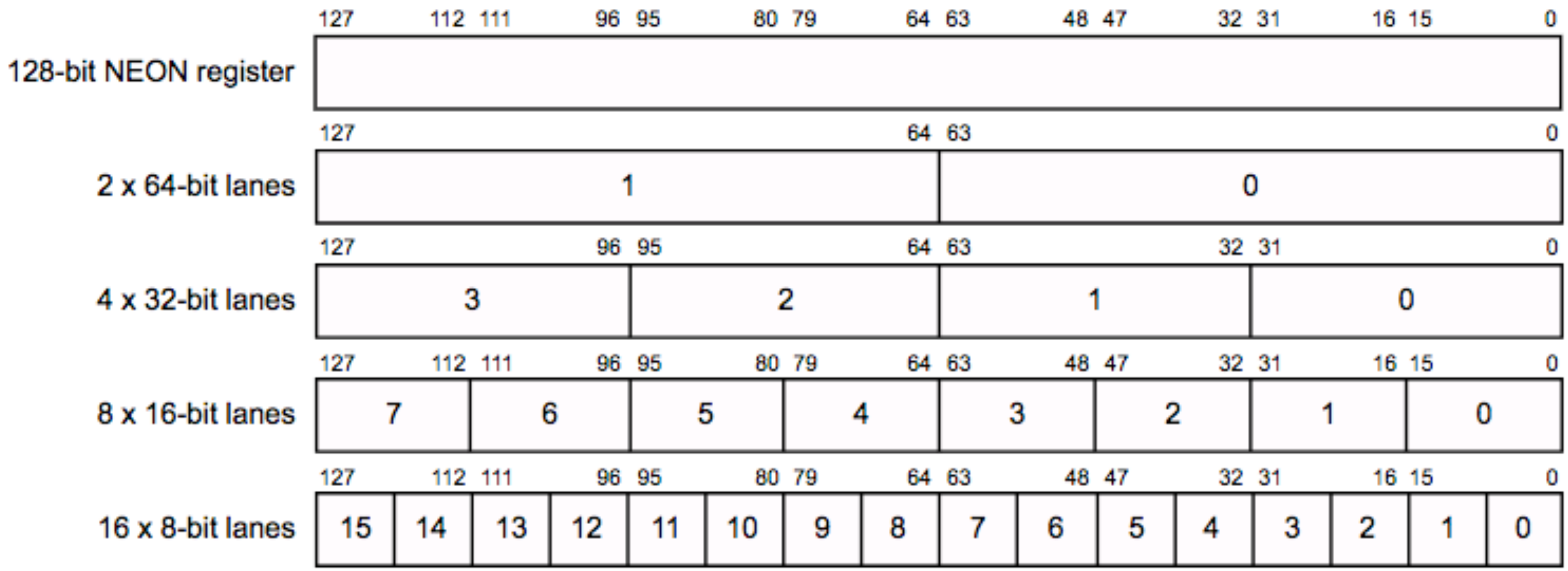
- Single instruction that operates on multiple data, either stored in registers or in memory
- Common on modern systems, to perform vector arithmetic:
  - x86-64: MMX, SSE, SSE2, SSE3, SSSE3
  - PowerPC: AltiVec
  - ARM: NEON
- Fewer instructions to fetch, but requires more hardware

SIMD Subtypes

- “True” Vector Architecture: instruction specifies starting source and destination memory addresses, and how many times to execute the instruction
  - Pipelined processor still executes only one calculation per cycle
  - Only one instruction fetch, but multiple cycles of execution, memory accesses, and write backs
- Short-Vector Architecture: execute a single instruction across a few registers, treating each register as containing multiple independent data
  - Example: ARM’s NEON has 32 SIMD 128-bit registers, which can be treated as 2x 64-bit, 4x 32-bit, 8x 16-bit, or 16x 8-bit integers (signed or unsigned), or as 2x 64-bit or 4x 32-bit floating point values

ARM Cortex-A Series Programmer’s Guide, §7.2

ARM NEON registers



- Extension to ARMv7 and ARMv8, intended to accelerate common audio and video processing

- Performs the same operation in all lanes of a vector

https://developer.arm.com/technologies/neon

ARMv8-A NEON Example

```
/* add an array of floating
   point pairs */
void add_float_neon2
   (float *dst, float *src1,
    float *src2, int count);
```

```
add_float_neon2:
  ld1    {v0.4s}, [x1], #16
  ld1    {v1.4s}, [x2], #16
  fadd   v0.4s, v0.4s, v1.4s
  subs   x3, x3, #4
  st1    {v0.4s}, [x0], #16
  bgt    add_float_neon2
  ret
```

- `ld1` loads 1 element to one lane of a SIMD register, `st1` stores data from a SIMD register
    - `.4s` suffix means treat the register as having 4 single-precision floats
- `fadd` performs a vector floating-point add

https://community.arm.com/android-community/b/android/posts/arm-neon-programming-quick-reference

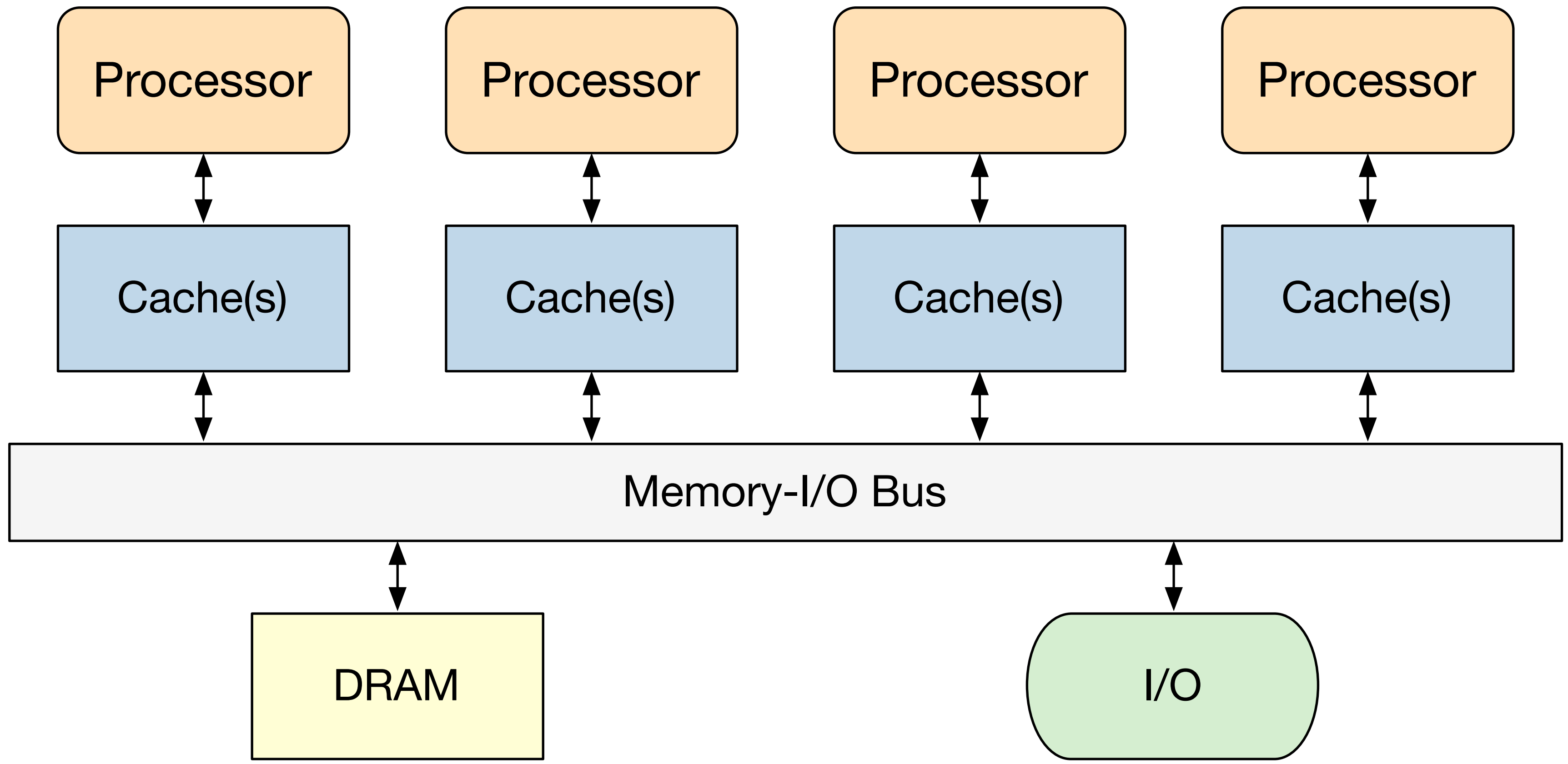
MISD

- Multiprocessor machine, executing different instructions upon the same dataset
- Built for fault tolerance systems
    - Example: Space Shuttle flight computer
    - Otherwise, very rare

MIMD

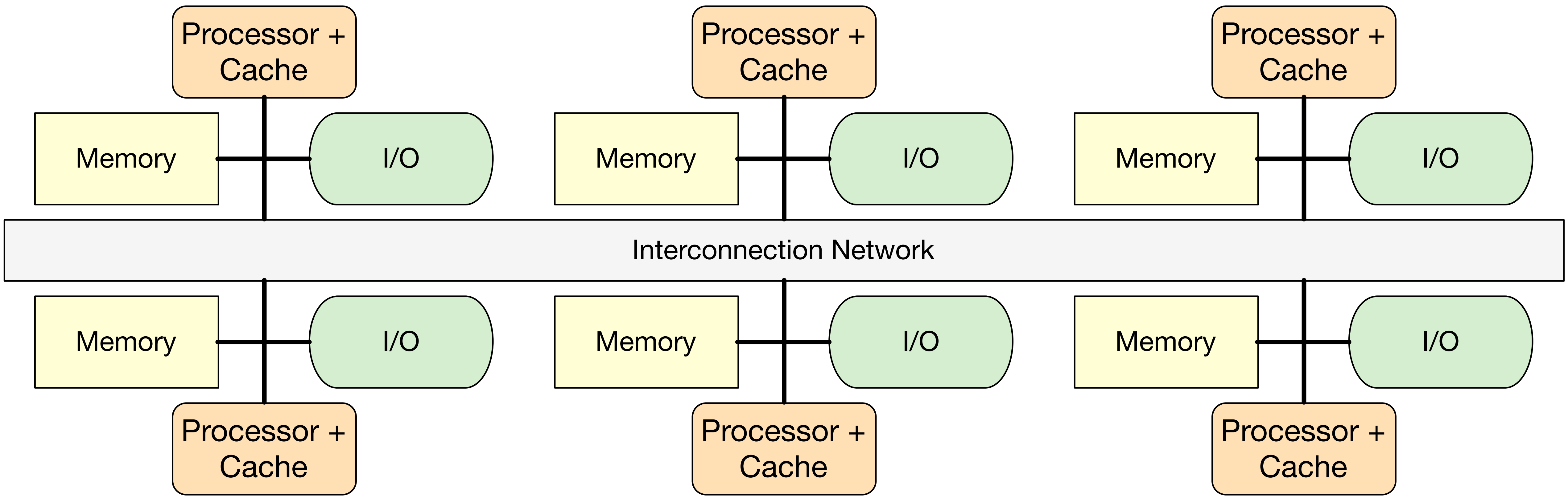
- Multiprocessor machine, executing different instructions on different data independently
  - Most modern systems are some type of MIMD
- Subtypes based upon memory model:
  - Centralized shared memory
  - Distributed shared memory

Centralized Shared Memory MIMD



- Uniform Memory Access (UMA): Processors share a single centralized memory through a single bus interconnect, with a snoopers
- Feasible for systems with few processors, when memory contention is infrequent

Distributed Memory MIMD



- Physically distributed memory, to avoid memory contention given a system with many processors, but [typically] no snooping between nodes
    - Processor nodes can have some local I/O (clustering)
- Difficult to synchronize separate nodes