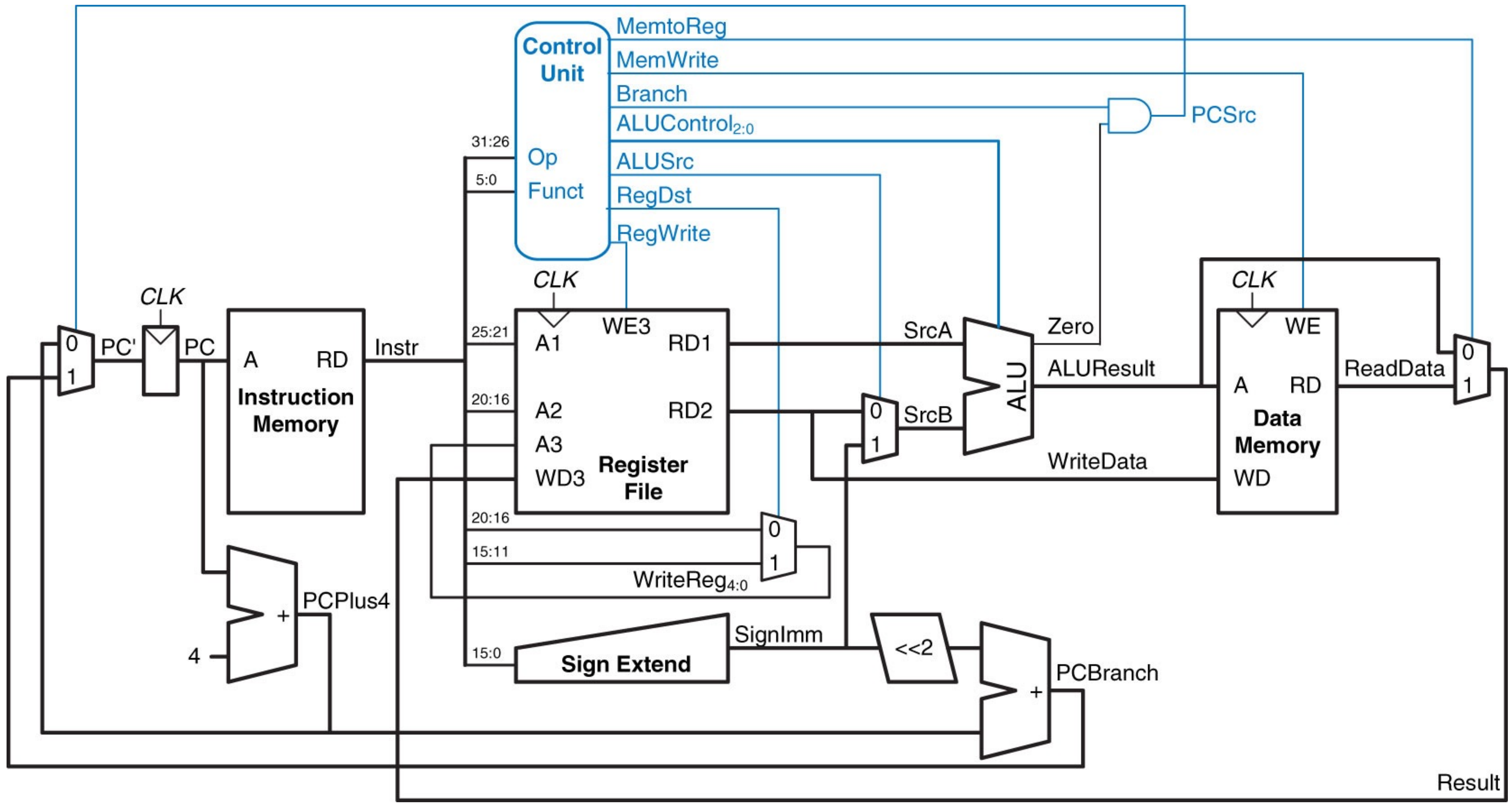
Cache


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
31:26
5:0
Op
Funct
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
WriteReg4:0
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
Data Memory
WE
ReadData
WD
Sign Extend
SignImm
<<2
PCBranch
PCPlus4
4
Result

Introduction

CPU (central processing unit) performs processing

- Registers are part of the CPU

RAM (random access memory) holds data

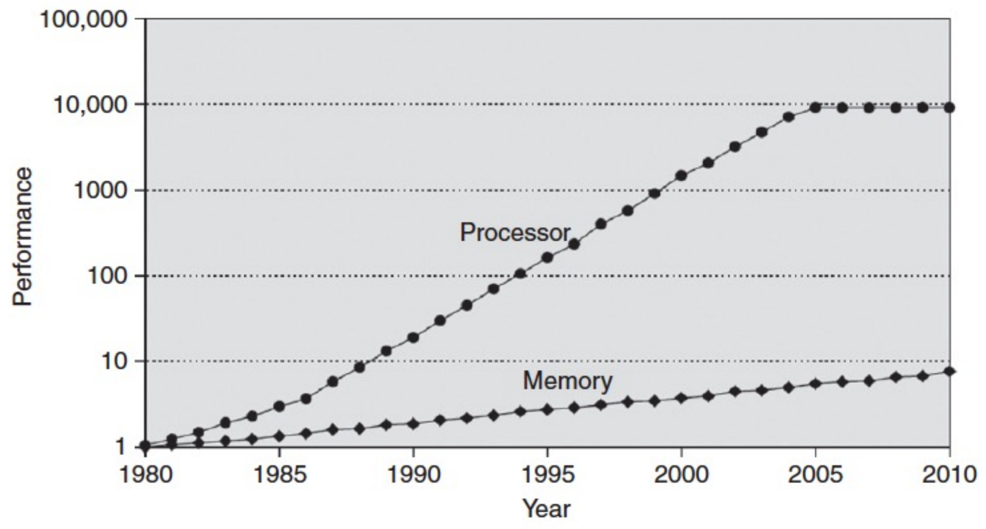
Time Cost of Memory

Memory is *much* slower than the processor

Everything needs to happen in one clock cycle*

* for now

Making the CPU wait for data means we're wasting time



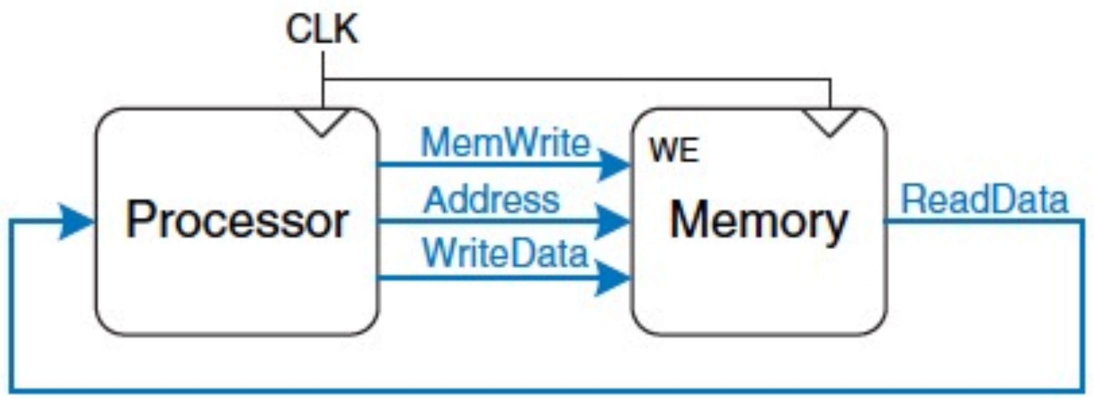
What if loads and stores were 100x slower?

```
# assume arr is a static array

la $s0, arr
addi $t0, $0, 0
for:
        slti $t1, $t0, 10
        beq $t1, $0, done
        sll $s1, $t1, 2
        add $t3, $s0, $s1
        lw $t2, 0($t3)
        addi $t2, $t2, 5
        sw $t2, 0($t3)
        addi $t0, $t0, 1
        j for
done:
```

- Clock cycle would be 100x slower, so *everything* would be 100x slower

Memory Interface



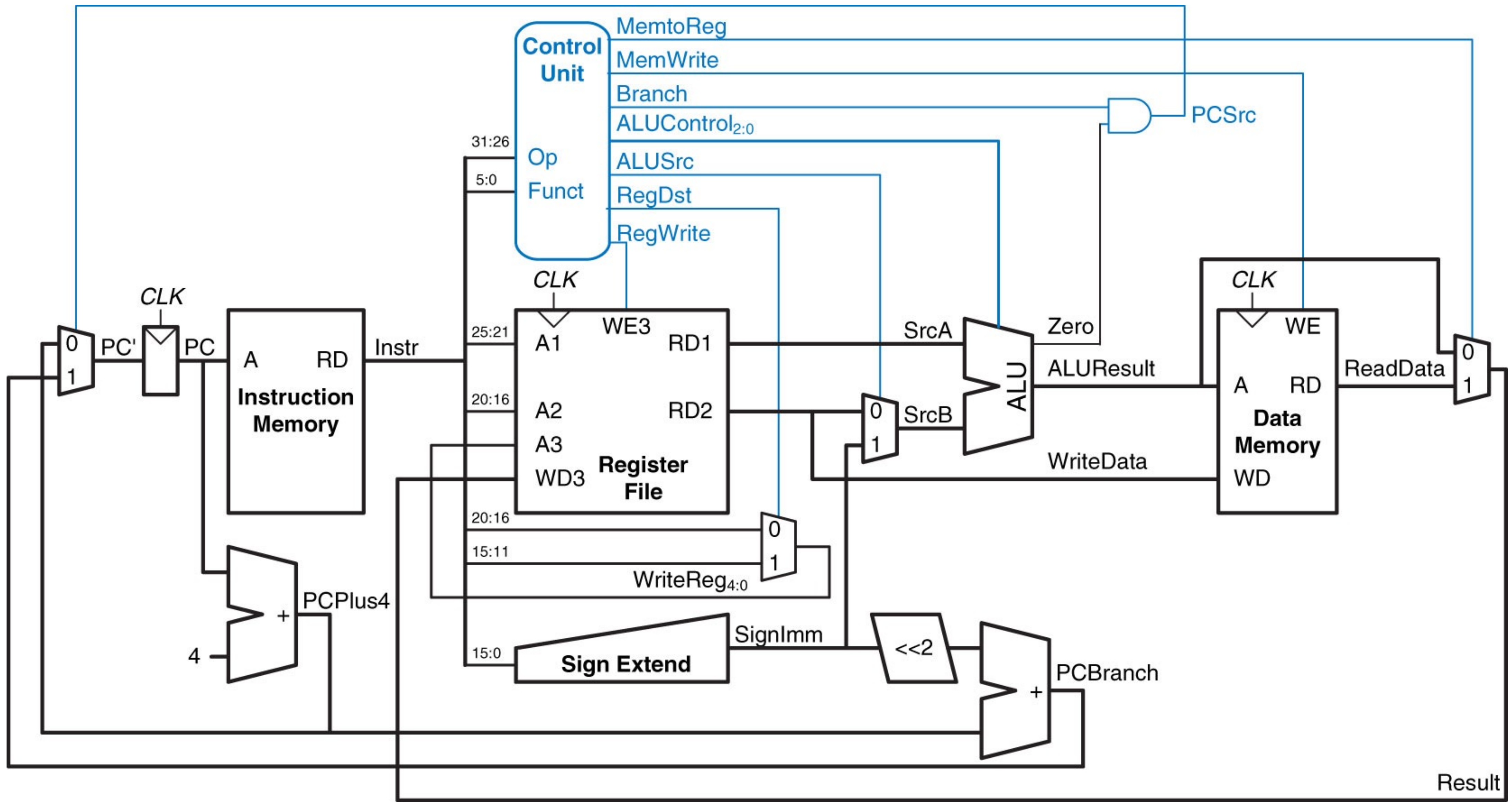
Processor communicates with memory through a memory interface

Decouples CPU from memory to allow clock to stay at CPU speeds

- Memory no longer accessed in one clock cycle

CPU still needs to do something while waiting for memory (the clock is ticking)

- We will assume it stalls



Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
WriteReg4:0
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
A
RD
WE
WD
Data Memory
ReadData
Result
Sign Extend
SignImm
<<2
PCBranch
PCPlus4
4
+
0
1

What if loads and stores took 100 clock cycles?

```
# assume arr is a static array

la $s0, arr
addi $t0, $0, 0
for:
        slti $t1, $t0, 10
        beq $t1, $0, done
        sll $s1, $t1, 2
        add $t3, $s0, $s1
        lw $t2, 0($t3)
        addi $t2, $t2, 5
        sw $t2, 0($t3)
        addi $t0, $t0, 1
        j for
done:
```

- Code is slower in proportion to number of lw and sw operations
- In the case of the loop to the left, there are 9 instructions, and only 2 access memory
  - Assume cycle time is 1 ns
  - Each loop iteration would cost 207 ns
  - If we had slowed down the clock instead, would have been 900 ns per loop iteration
- A big savings over slowing down the clock, but still much too slow

Just have the engineers do it

Further improvements will be minimal unless we have faster memory

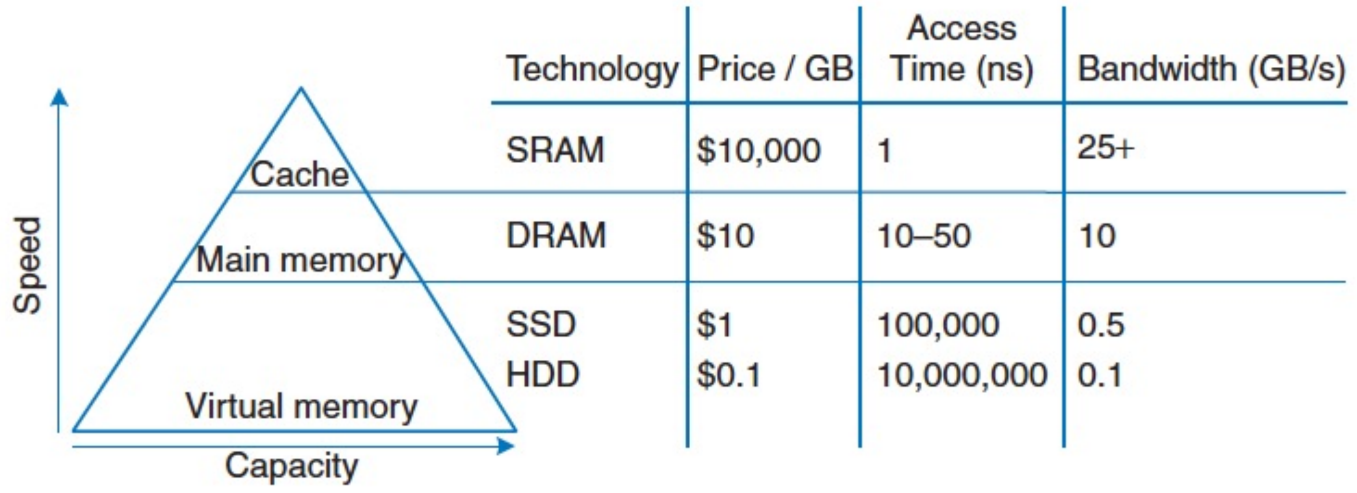
Fortunately, we do!

Unfortunately, it’s not cheap.

Memory Tradeoffs

Ideally, memory would be fast, large, and cheap

- In practice, you can have two of the three
- Either slow, small, or expensive



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10–50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

But…

We can combine two memory systems with different attributes.

- Fast, small, expensive with slow, large, cheap
- Minimize $$$$!

Fast memory serves the commonly used data and instructions

- This small, fast memory is the *cache*

Slow memory holds the remainder giving us a much larger capacity.

This is the same tradeoff we already make between registers and memory

Cache

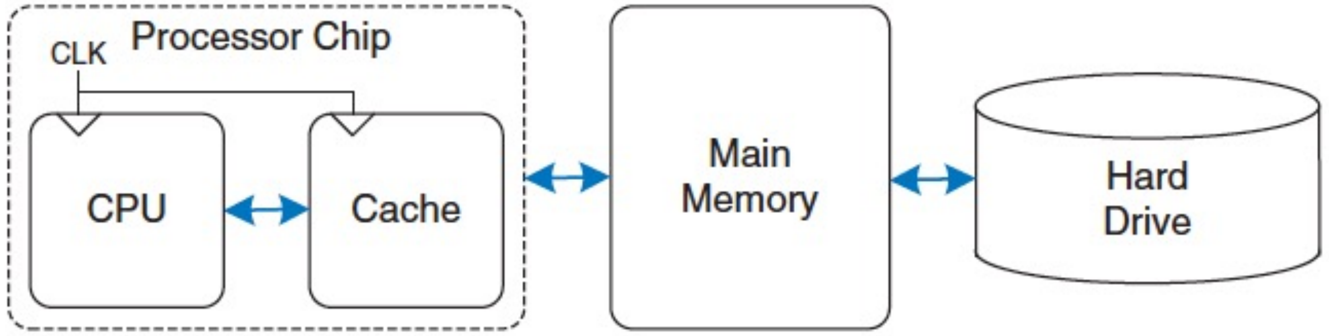
Computers store the most commonly used instructions and data in fast, but small, memory (Cache)

- Cache is located in SRAM on the CPU.
- Cache speed comparable to CPU as SRAM is faster than DRAM

No delay caused by going out to a separate chip.

SRAM cost $10,000/GB in 2012

Cache is thus relatively small (kilobytes to several MB)



Library

Why do caches work?

Temporal and Spatial locality

- Temporal: If you have used a book recently, you are likely to use it again soon.
- Spatial: When you use a particular book, use others in the same area.

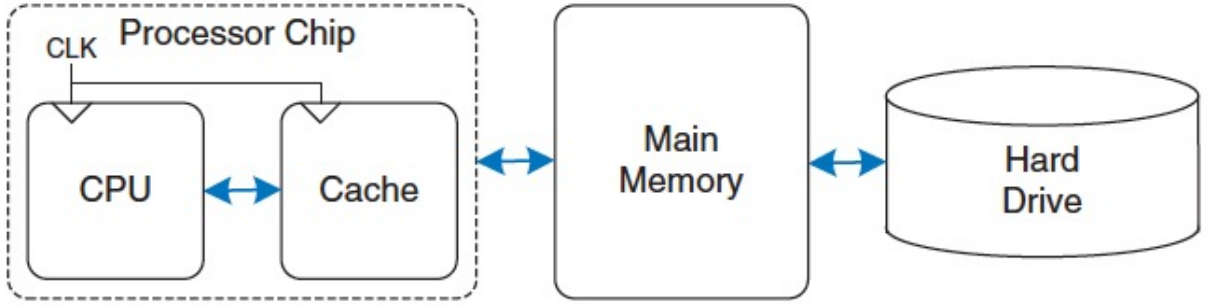
Speed up by exploiting both of these commonalities

Same idea comes up everywhere!

Cache Usage

Processor makes a request for data.
Hierarchy of looking for data:

1. Cache
2. DRAM
3. HD



If data is found in the cache, we have a hit

- Returned quickly

If not, miss

- Must go to main memory

Key: Maximizing the hit ratio makes our average access time low (Good)

What Data is Held in the Cache

Ideally a cache would predict what data is needed and keep it ready

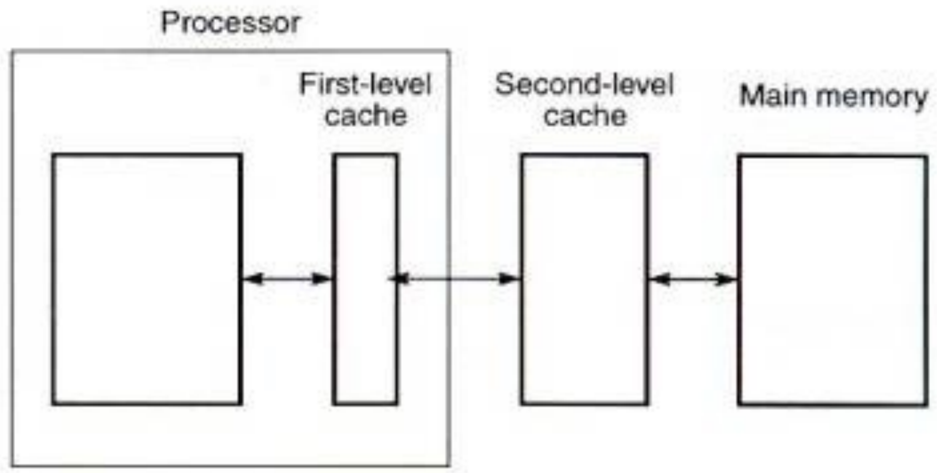
- Sadly, we don't have a crystal ball
- Must make a guess at what data will be needed

How might we make this guess? Locality.

Temporal: Processor is likely to access a piece of data again soon if it has accessed the data recently.

Spatial: when data is accessed, data near it is also likely to be accessed

Multi-Level Caches



Virtual memory

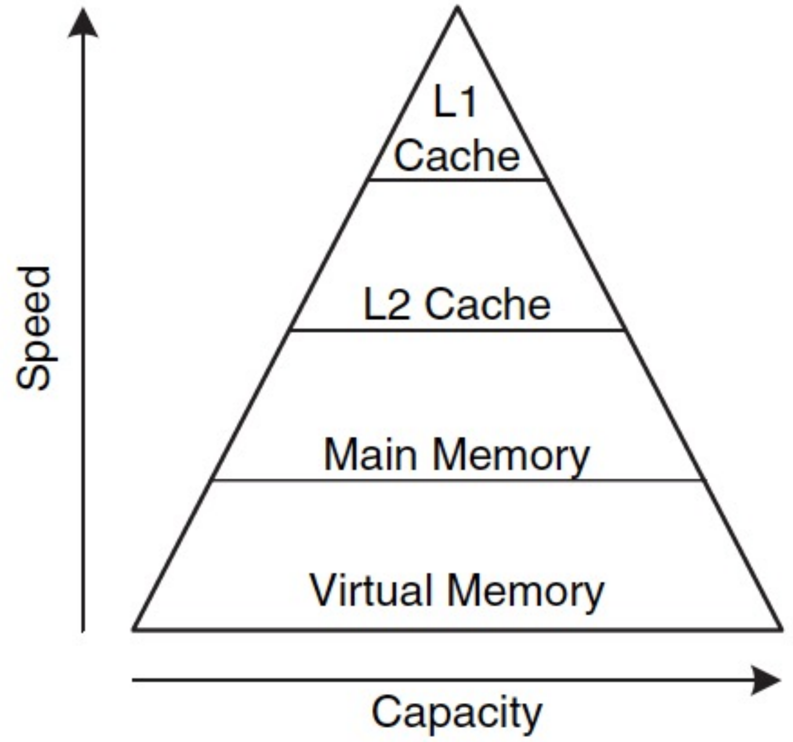
Hard drives hold most of our data but access times are slow.

- ”Virtual Memory"
- Slow access but lots of it

Virtual memory is to DRAM as DRAM is to cache

Multiple-Level Caches

- Large caches hold data of interest and have a lower miss rate
  - Bigger sizes
  - They are slower than smaller caches however.
- L1 cache
  - Small, one or two-cycle access time
  - Typically "split" – instructions and data
- L2 cache
  - Larger and slower
- Some systems may even have more levels of cache



Intel Pentium III die

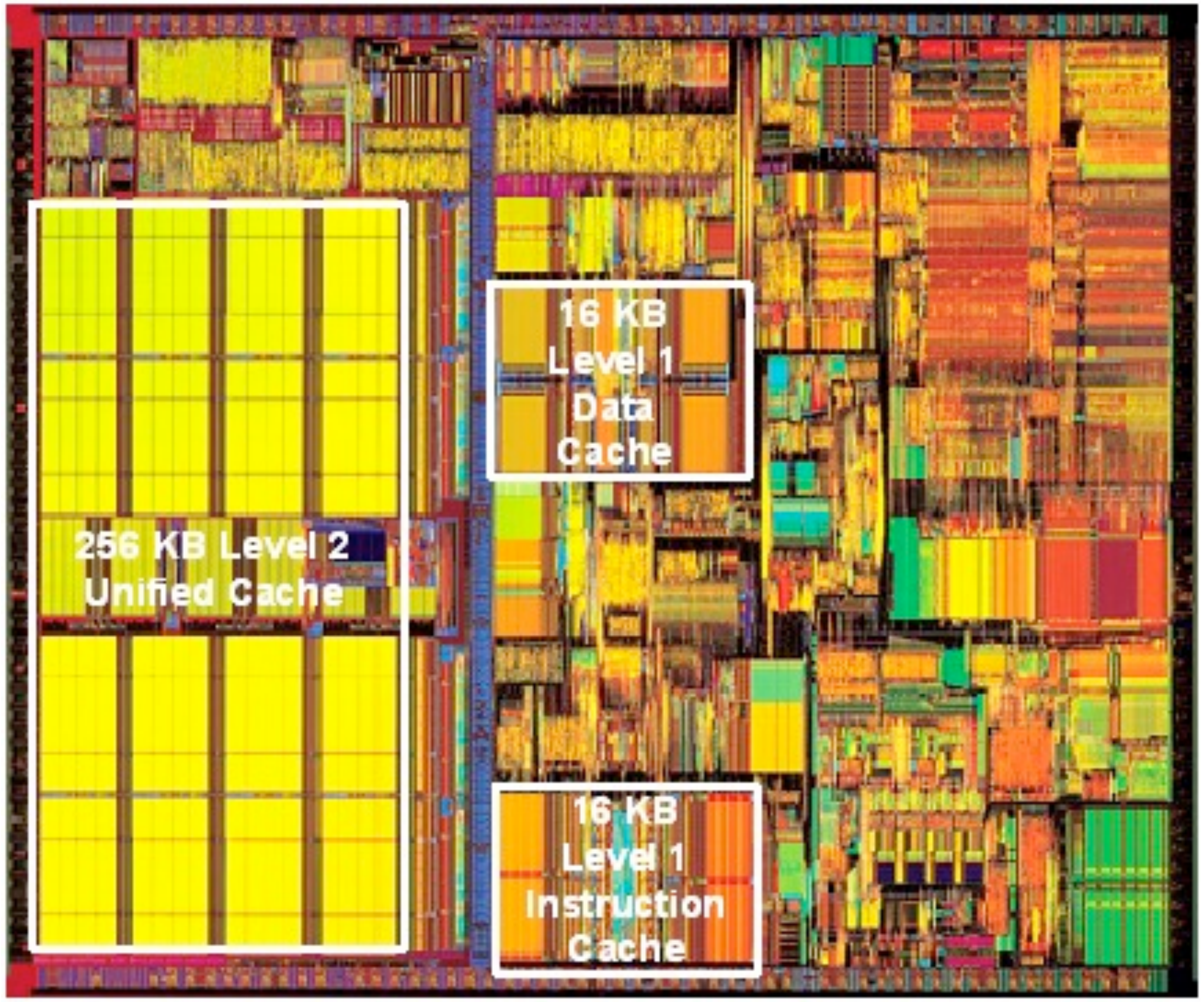


Figure from Harris and Harris

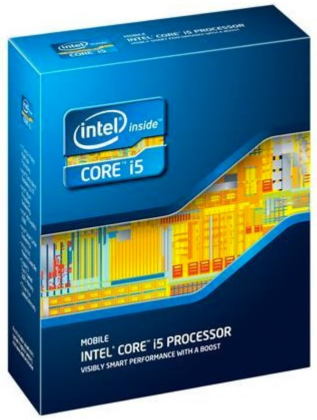
Real World Specs

Server

$9,999.99

Details

Core Name	SkyLake
# of Cores	28-Core
# of Threads	56
Operating Frequency	2.5 GHz
Max Turbo Frequency	3.8 GHz
L3 Cache	38.5MB



Mobile

$147.98

Details

Core Name	Sandy Bridge
Multi-Core	Dual-Core
Operating Frequency	2.6GHz (3.3GHz Turbo Boost)
L2 Cache	2 x 256KB
L3 Cache	3MB

Desktop

$239.99

Details

CPU Socket Type	LGA 1151 (300 Series)
Core Name	Coffee Lake
# of Cores	6-Core
# of Threads	6
Operating Frequency	3.6 GHz
Max Turbo Frequency	4.3 GHz
L3 Cache	9MB

Performance improvements are not always as good as they sound.

Improving memory to be 10X as fast will not necessarily make a program run ten times as fast.

Caching is more important the more a program accesses memory

Amdahl's Law: Effort spent on increasing the performance of a subsystem is worthwhile only if the subsystem affects a large percentage of the overall performance.