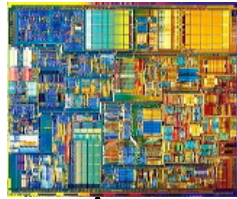

The Memory Hierarchy

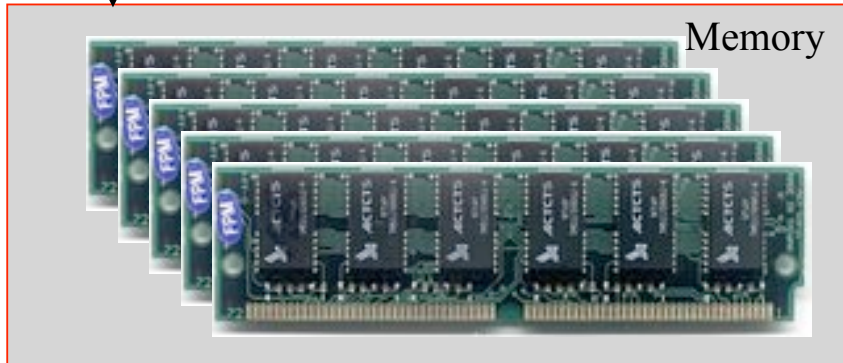
Memory



CPU



Abstraction: Big array of bytes



Memory

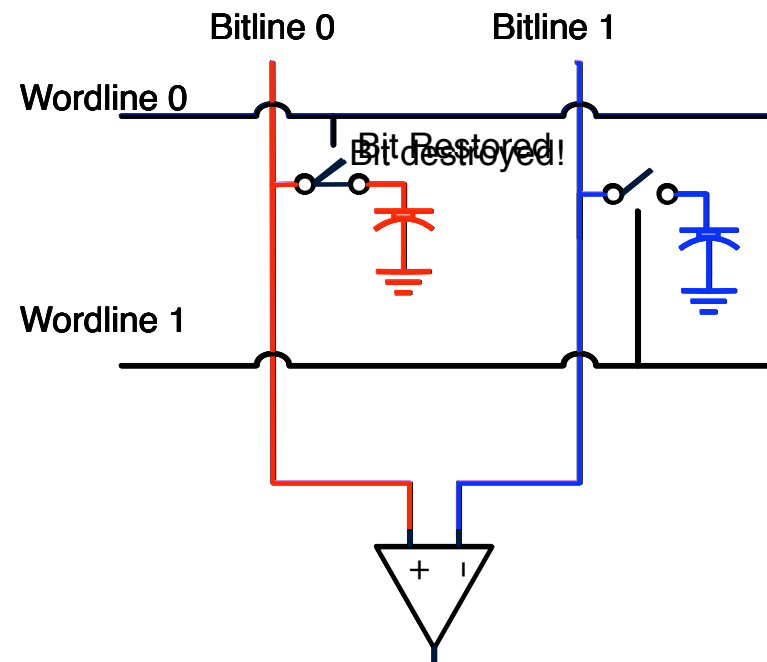
Dynamic Random Access Memory (DRAM)

- Storage

- Charge on a capacitor
- Decays over time (us-scale)
- This is the “dynamic” part.
- About $6F^2$: 20x better than SRAM

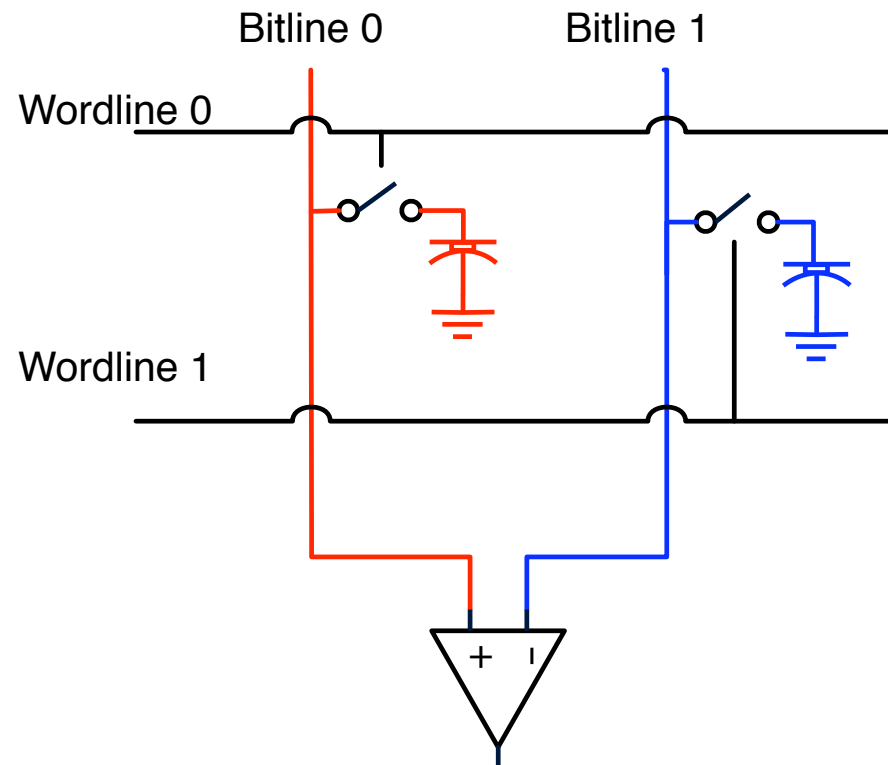
- Reading

- Precharge
- Assert word line
- Sense output
- Refresh data

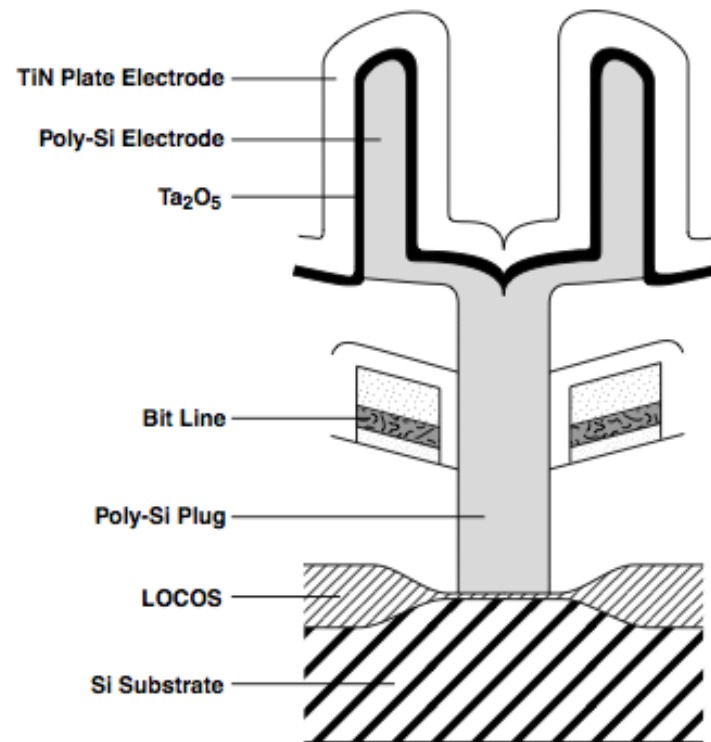


DRAM: Write and Refresh

- **Writing**
 - Turn on the wordline
 - Override the sense amp.
- **Refresh**
 - Every few micro-seconds, read and re-write every bit.
 - Consumes power
 - Takes time

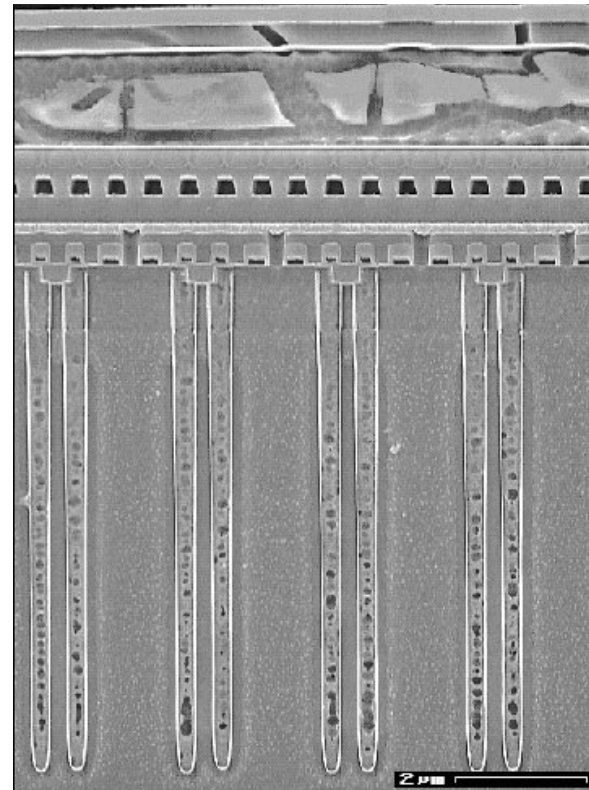


DRAM Lithography



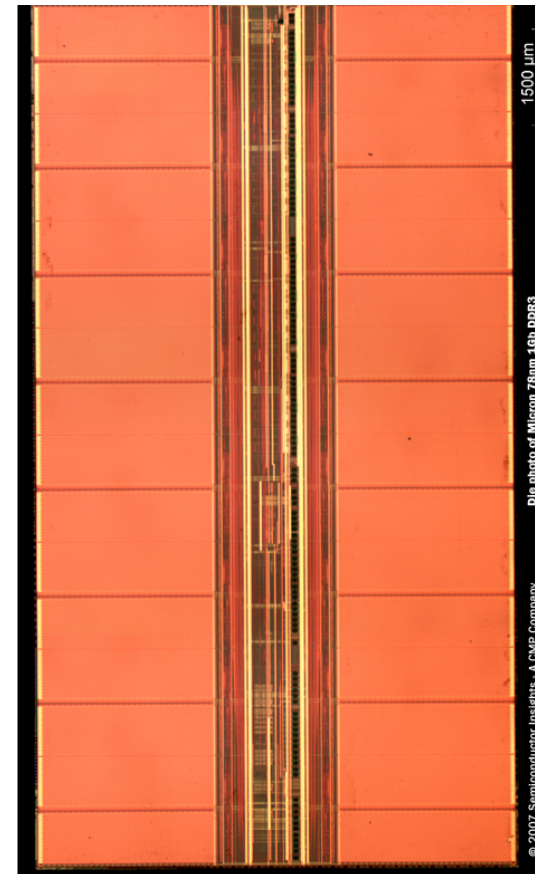
Source: Hitachi/ICE, "Memory 1997"

20766



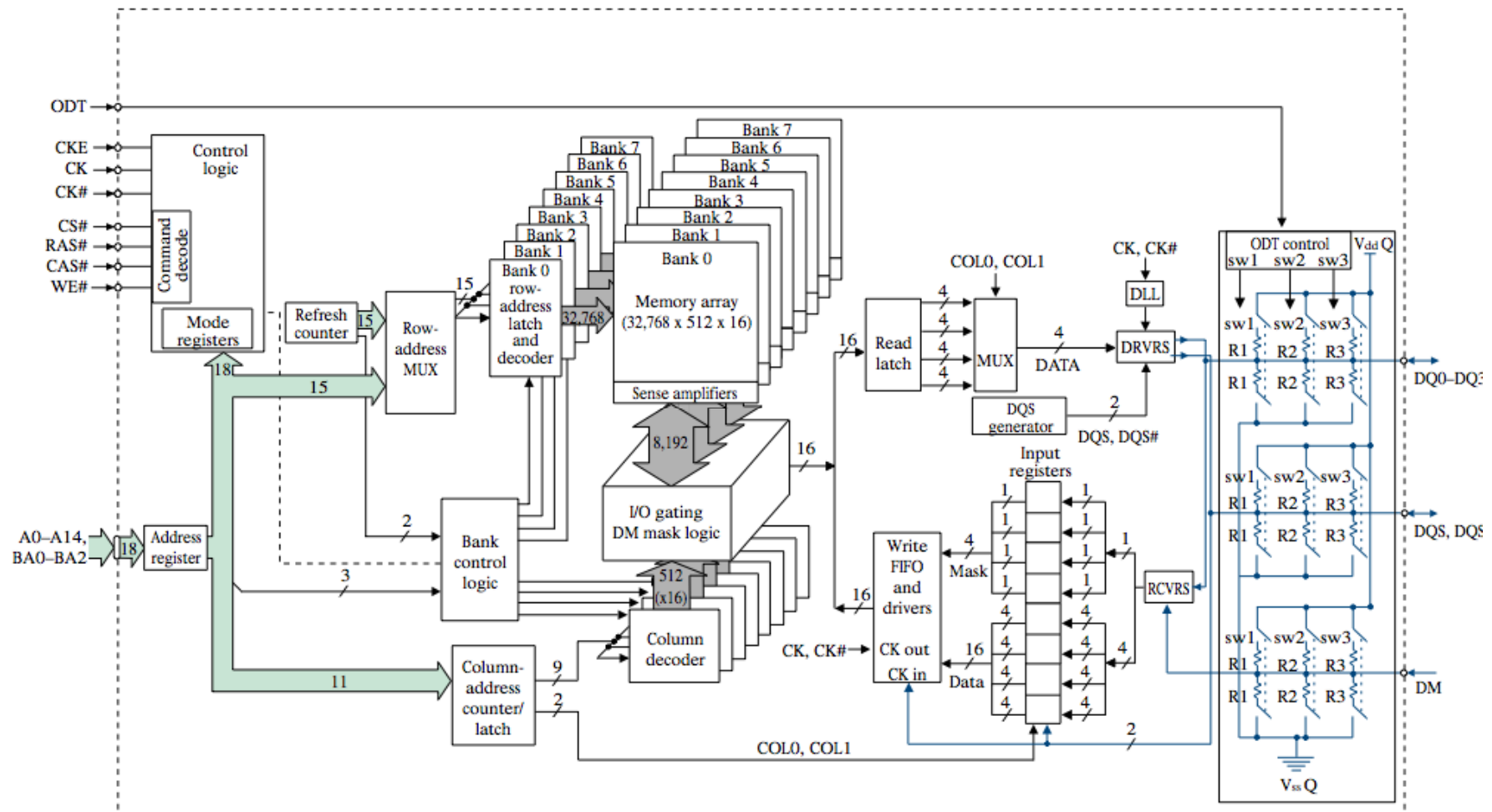
DRAM Devices

- There are many banks per die (16 at left)
 - Multiple can be active at once to hide latencies
 - Parallelism!!!
- Example
 - open bank 1, row 4
 - open bank 2, row 7
 - open bank 3, row 10
 - read bank 1, column 8
 - read bank 2, column 32
 - ...



Micron 78nm 1Gb DDR3

DRAM: Micron MT47H512M4

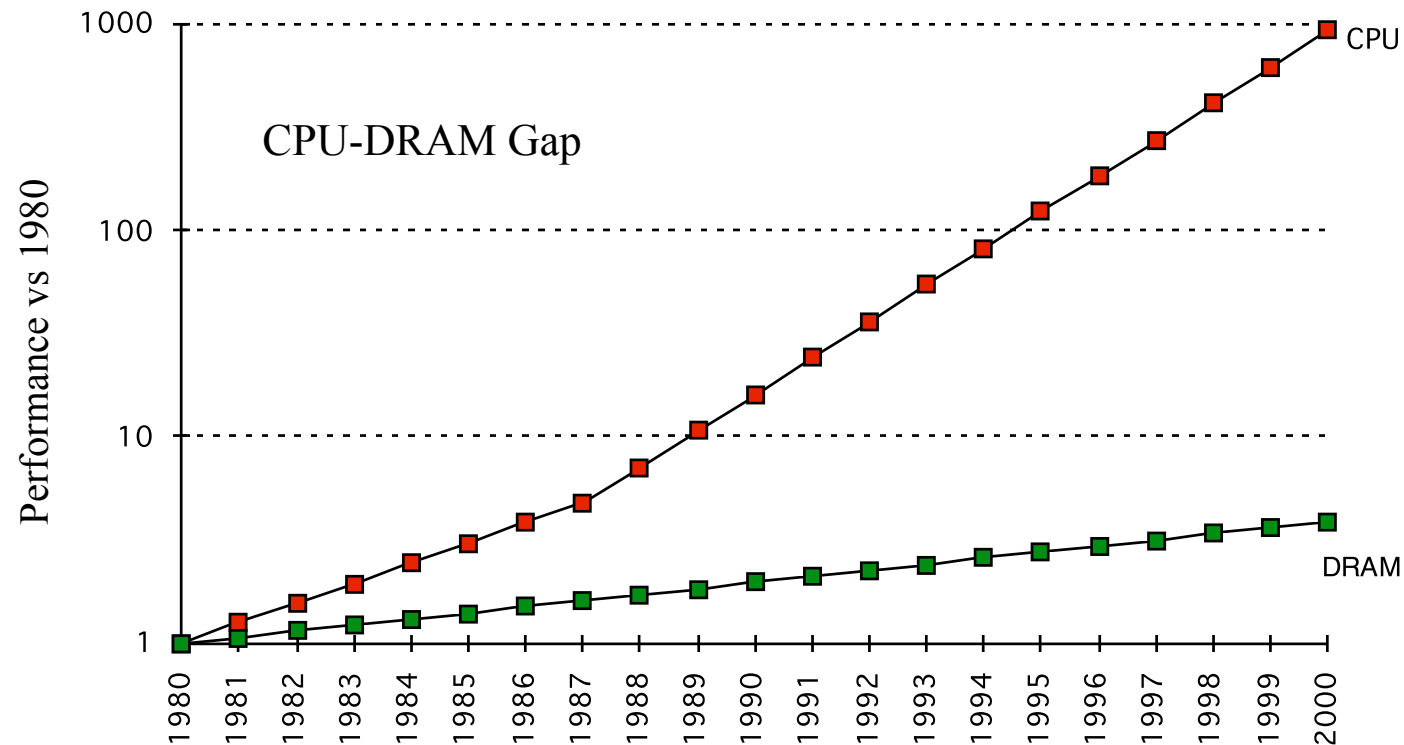


Main points for today

- What is a memory hierarchy?
- What is the CPU-DRAM gap?
- What is locality? What kinds are there?
- Learn a bunch of caching vocabulary.

Who Cares about Memory Hierarchy?

■ Processor vs Memory Performance



1980: no cache in microprocessor;
1995 2-level cache

Memory's impact

$M = \% \text{ mem ops}$

$M_{lat} (\text{cycles}) = \text{memory latency}$

$BCPI = \text{base CPI}$

$CPI =$

Memory's impact

$M = \% \text{ mem ops}$

$Mlat \text{ (cycles)} = \text{memory latency}$

$BCPI = \text{base CPI}$

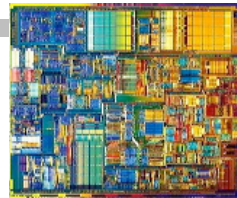
$$CPI = BCPI + M * Mlat$$

$BCPI = 1; M = 0.2; Mlat = 100$ (somewhat aggressive today)

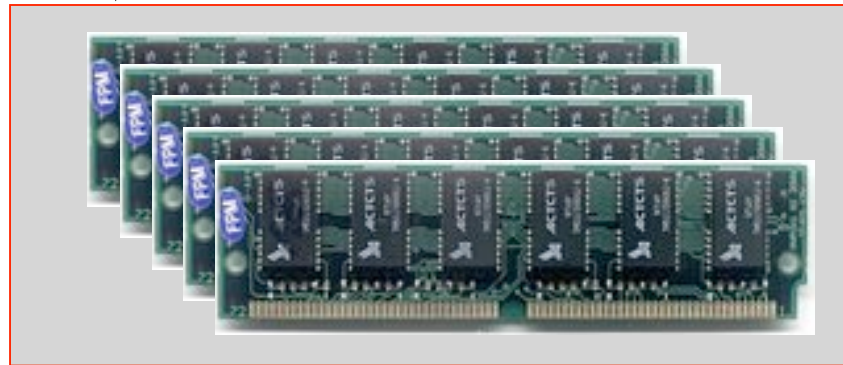
$CPI = 21 \rightarrow 20x \text{ slow down}$

Remember!: Amdahl's law does not bound the slowdown. Poor memory performance can make your program arbitrarily slow.

Memory Cache



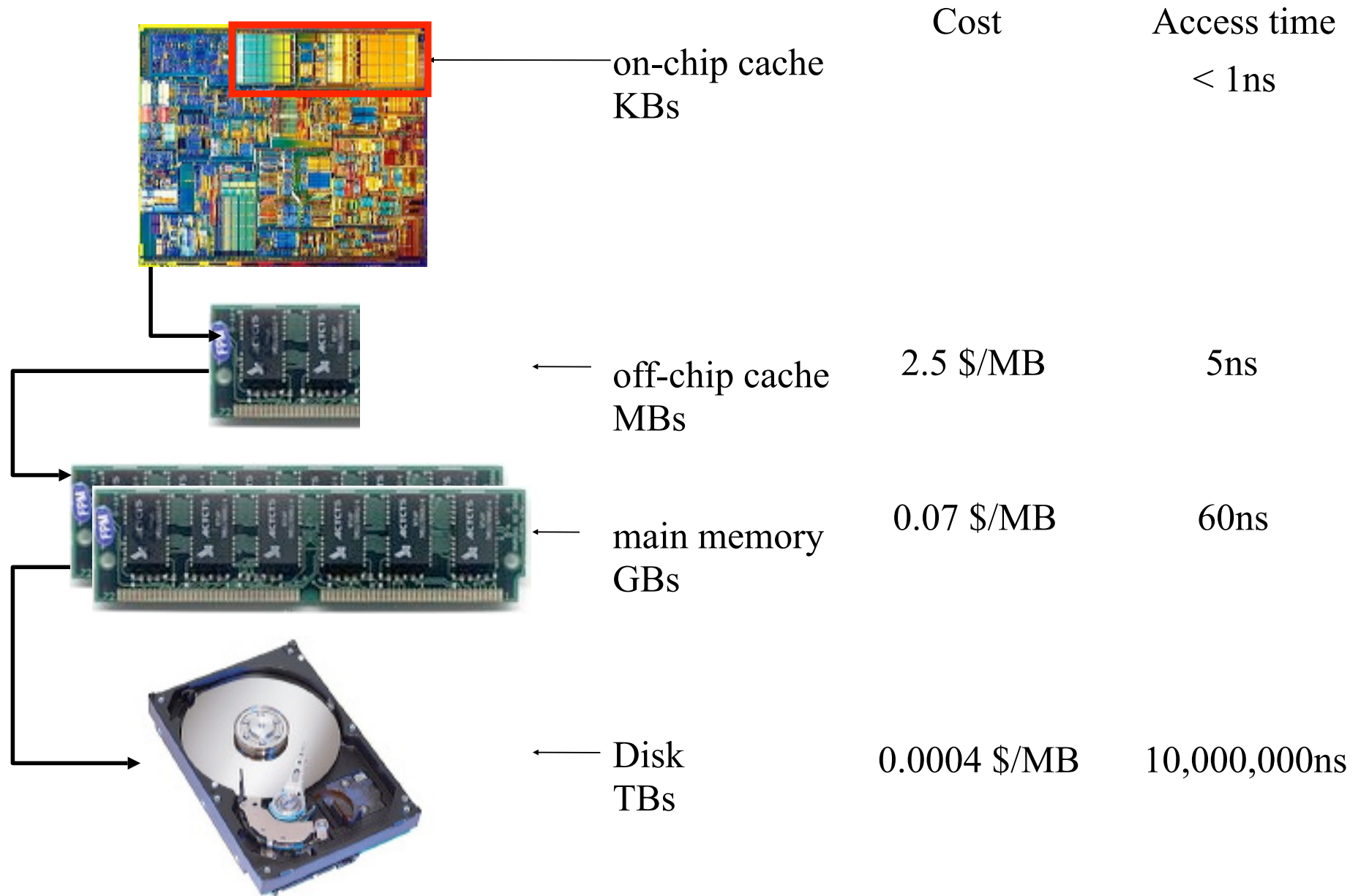
Small fast memory

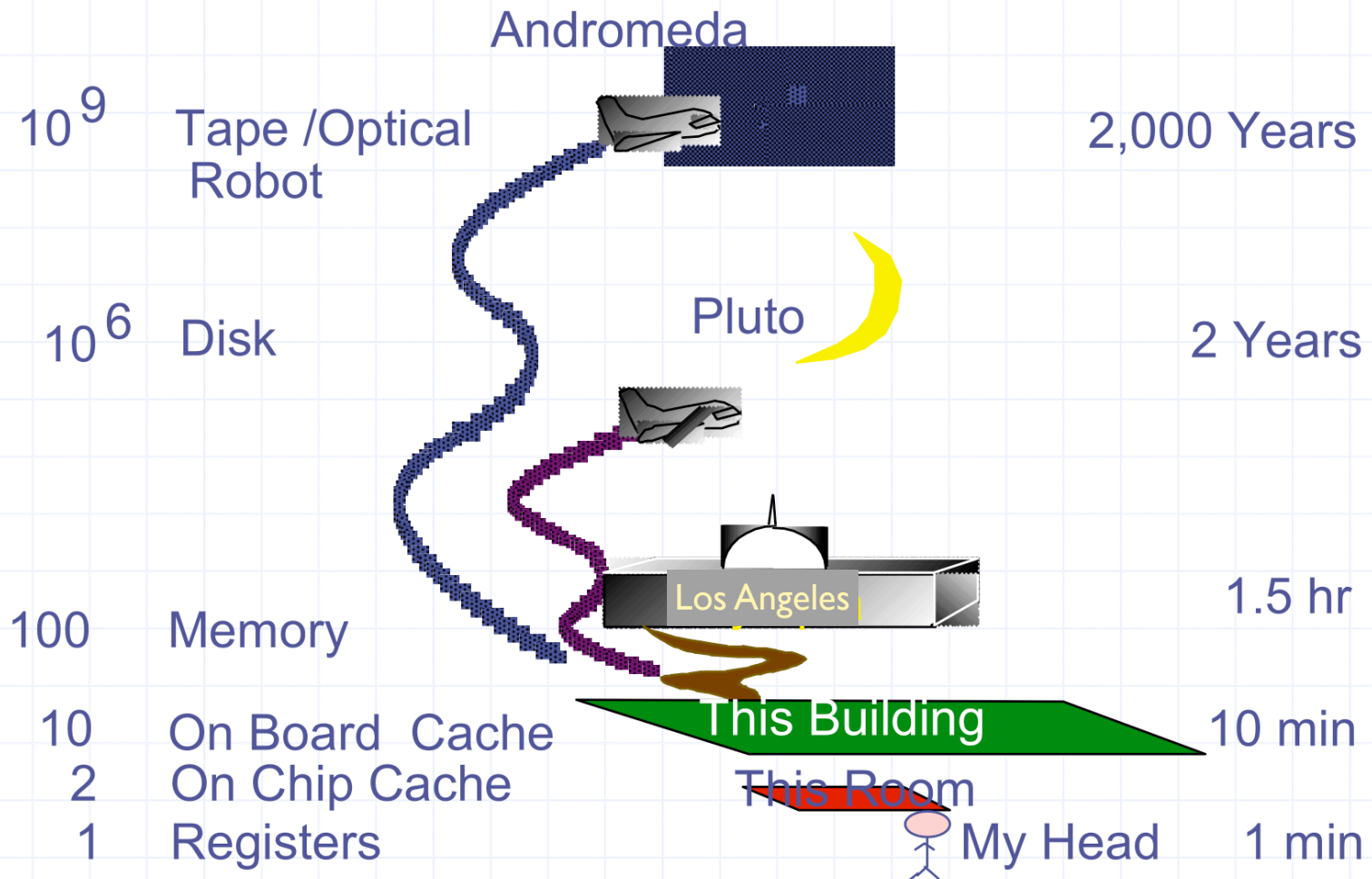


Big slow memory

- Memory tradeoffs:
 - Speed vs. Density vs. Cost
- Build several memories with different trade-offs
- How do you use it? Build a “memory hierarchy”
- What should it mean for the memory abstraction?

A typical memory hierarchy

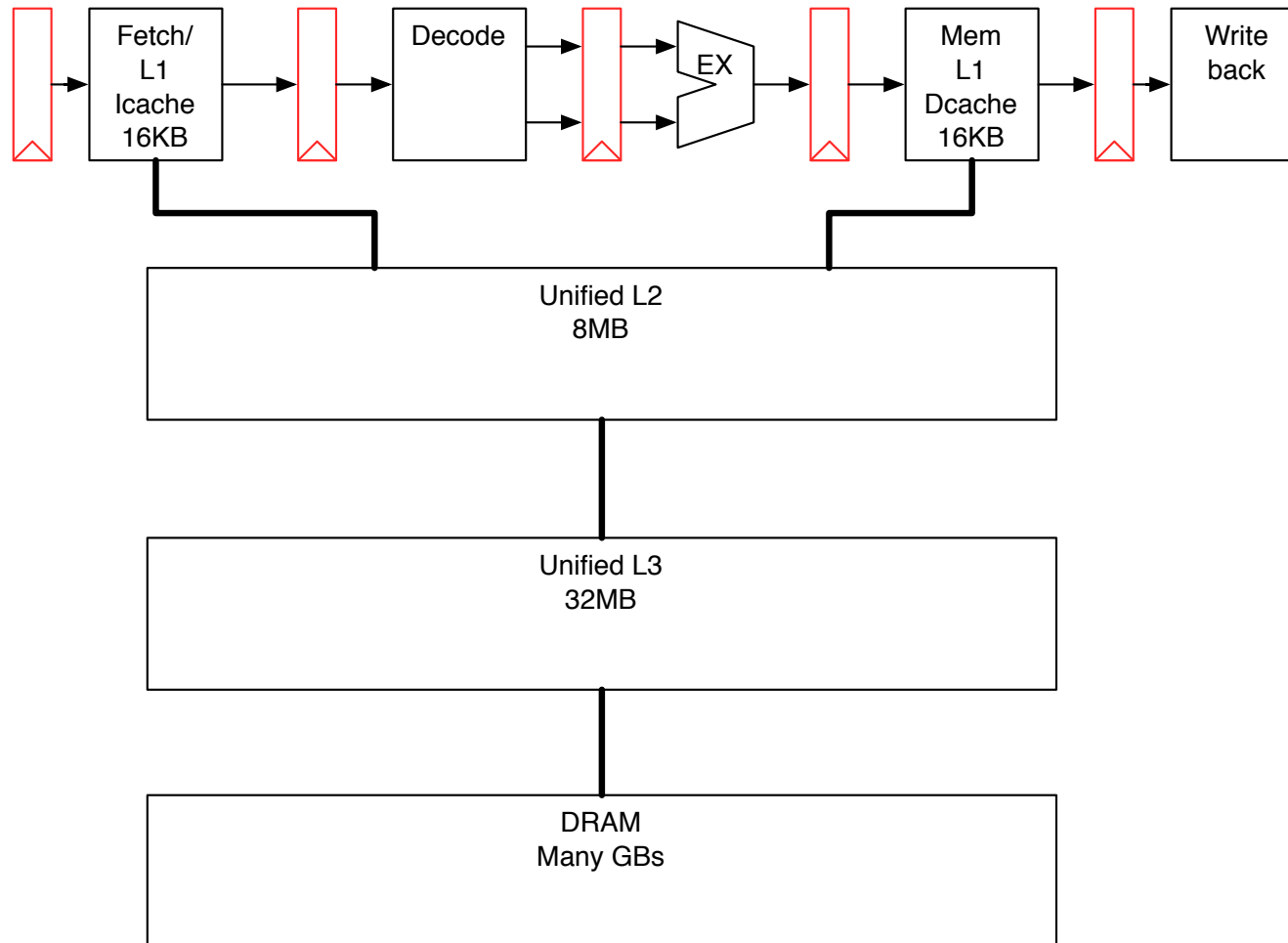




Why should we expect caching to work?

- Why did branch prediction work?
 - able to exploit predictability in computation

Typical Cache Hierarchy



Data vs Instruction Caches

- Why have different I and D caches?

Data vs Instruction Caches

- Why have different I and D caches?
 - Different areas of memory
 - Different access patterns
 - I-cache accesses have lots of spatial locality. Mostly sequential accesses.
 - I-cache accesses are also predictable to the extent that branches are predictable
 - D-cache accesses are typically less predictable
 - Not just different, but often across purposes.
 - Sequential I-cache accesses may interfere with the data the D-cache has collected.
 - This is “interference” just as we saw with branch predictors
 - At the LI level it avoids a structural hazard in the pipeline
 - Writes to the I cache by the program are rare enough that they can be prohibited (i.e., self modifying code)