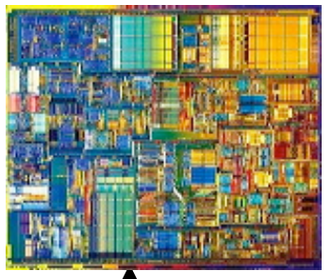
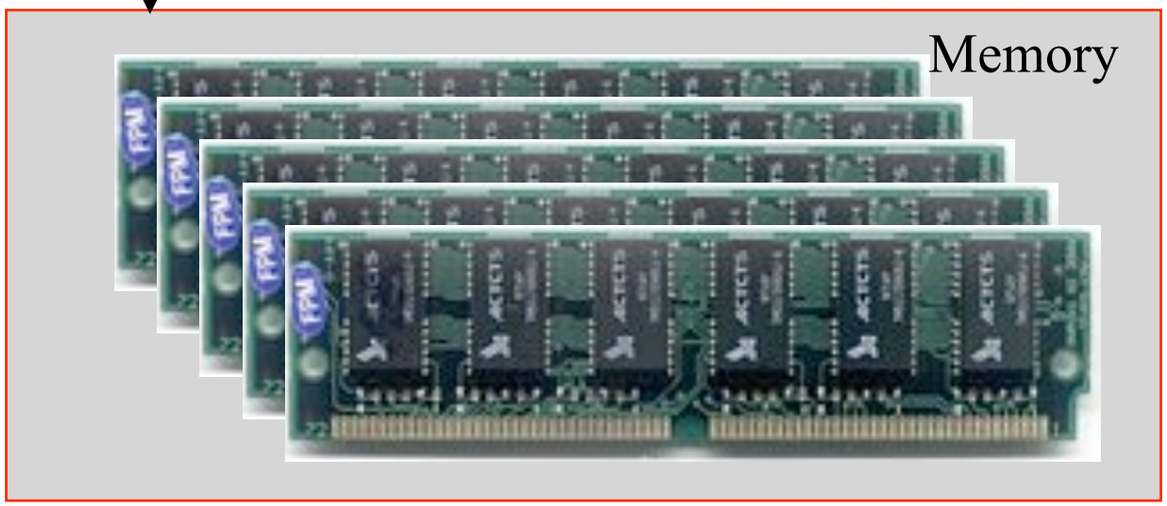
The Memory Hiearchy

Memory



CPU

Abstraction: Big array of bytes

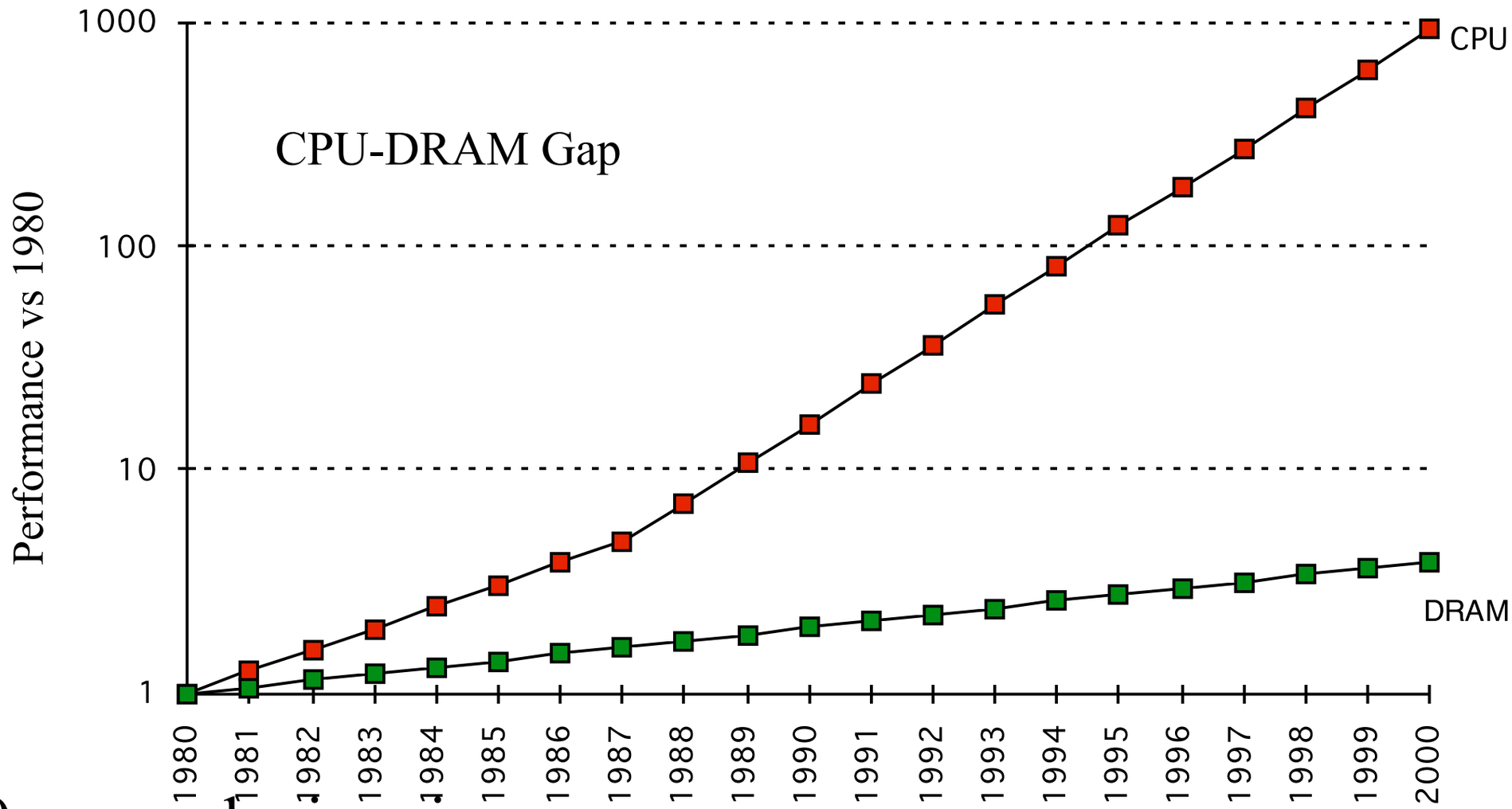


Main points for today

- 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 = % mem ops

Mlat (cycles) = memory latency

BCPI = base CPI

CPI =

Memory's impact

M = % mem ops

Mlat (cycles) = memory latency

BCPI = base CPI

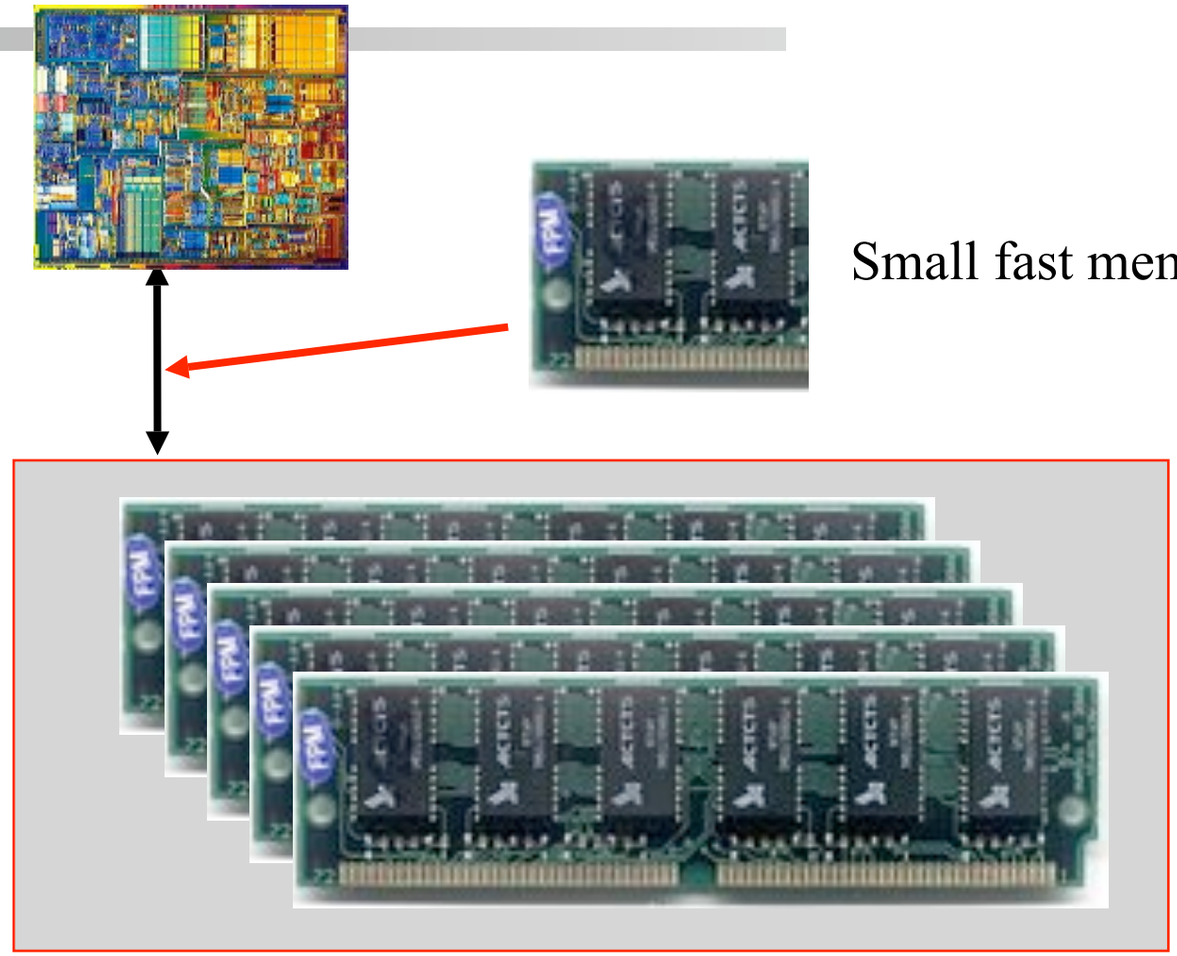
CPI = BCPI + M*Mlat

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

CPI = 21 -> 20x slow down

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

Memory Cache

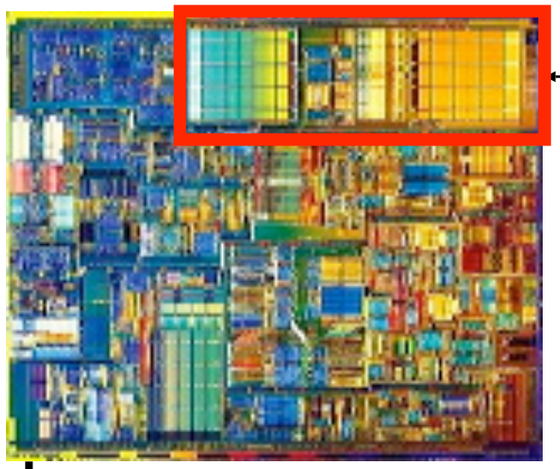


Small fast memory

Big slow memory

- Memory cost
  - >> capacity -> more $$
  - >> speed/bw -> more $$
  - >> speed -> larger (less dense)
- 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



on-chip cache
KBs

Cost

Access time

< 1ns

off-chip cache
MBs

2.5 $/MB

5ns

main memory
GBs

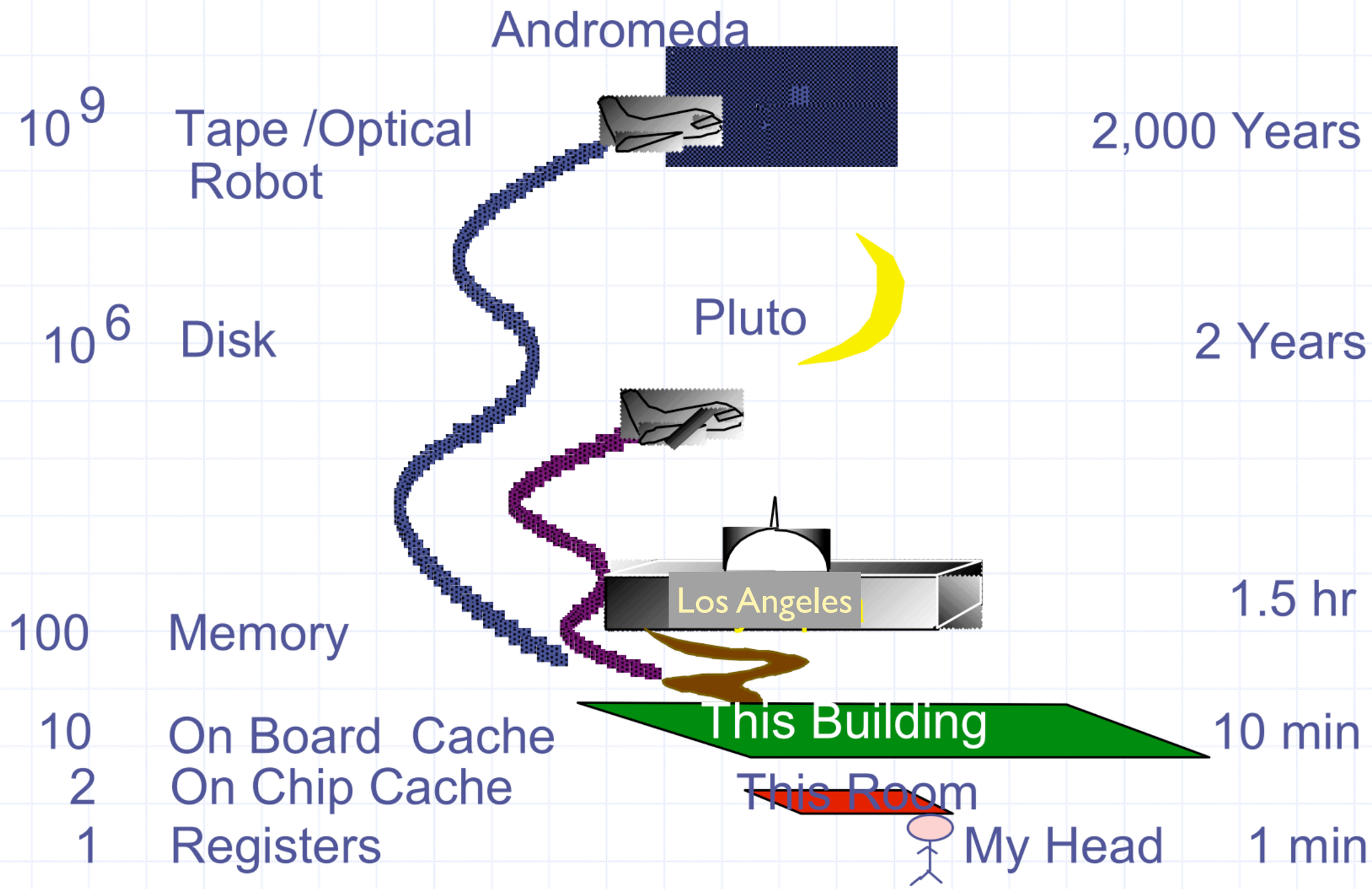
0.07 $/MB

60ns

Disk
TBs

0.0004 $/MB

10,000,000ns



Why should we expect caching to work?

- Why did branch prediction work?

Why should we expect caching to work?

- Why did branch prediction work?
- Where is memory access predictable
  - Predictably accessing the same data
    - In loops: for(i = 0; i < 10; i++) {s += foo[i];}
    - foo = value + configuration_parameter;
    - bar = another_value+ configuration_parameter;
  - Predictably accessing different data
    - In linked lists: while(l != NULL) {l = l->next;}
    - In arrays: for(i = 0; i < 10000; i++) {s += data[i];}
    - structure access: foo(some_struct.a, some_struct.b);

The Principle of Locality

- "Locality" is the tendency of data access to be predictable. There are two kinds:

- Spatial locality: The program is likely to access data that is close to data it has accessed recently

- Temporal locality: The program is likely to access the same data repeatedly.

Locality in Action

- Label each access with whether it has temporal or spatial locality or neither
  - 1
  - 2
  - 3
  - 10
  - 4
  - 1800
  - 11
  - 30
  - 1
  - 2
  - 3
  - 4
  - 10
  - 190
  - 11
  - 30
  - 12
  - 13
  - 182
  - 1004

Locality in Action

- Label each access with whether it has temporal or spatial locality or neither
  - 1 n
  - 2 s
  - 3 s
  - 10 n
  - 4 s
  - 1800 n
  - 11 s
  - 30 n
  - 1 t
  - 2 s, t
  - 3 s,t
  - 4 s,t
  - 10 s,t
  - 190 n
  - 11 s,t
  - 30 s
  - 12 s
  - 13 s
  - 182 n?
  - 1004 n

There is no hard and fast rule here. In practice, locality exists for an access if the cache performs well.

To Build a Cache

- You get 128 bytes
- Main memory is 1GB
- What should we do?

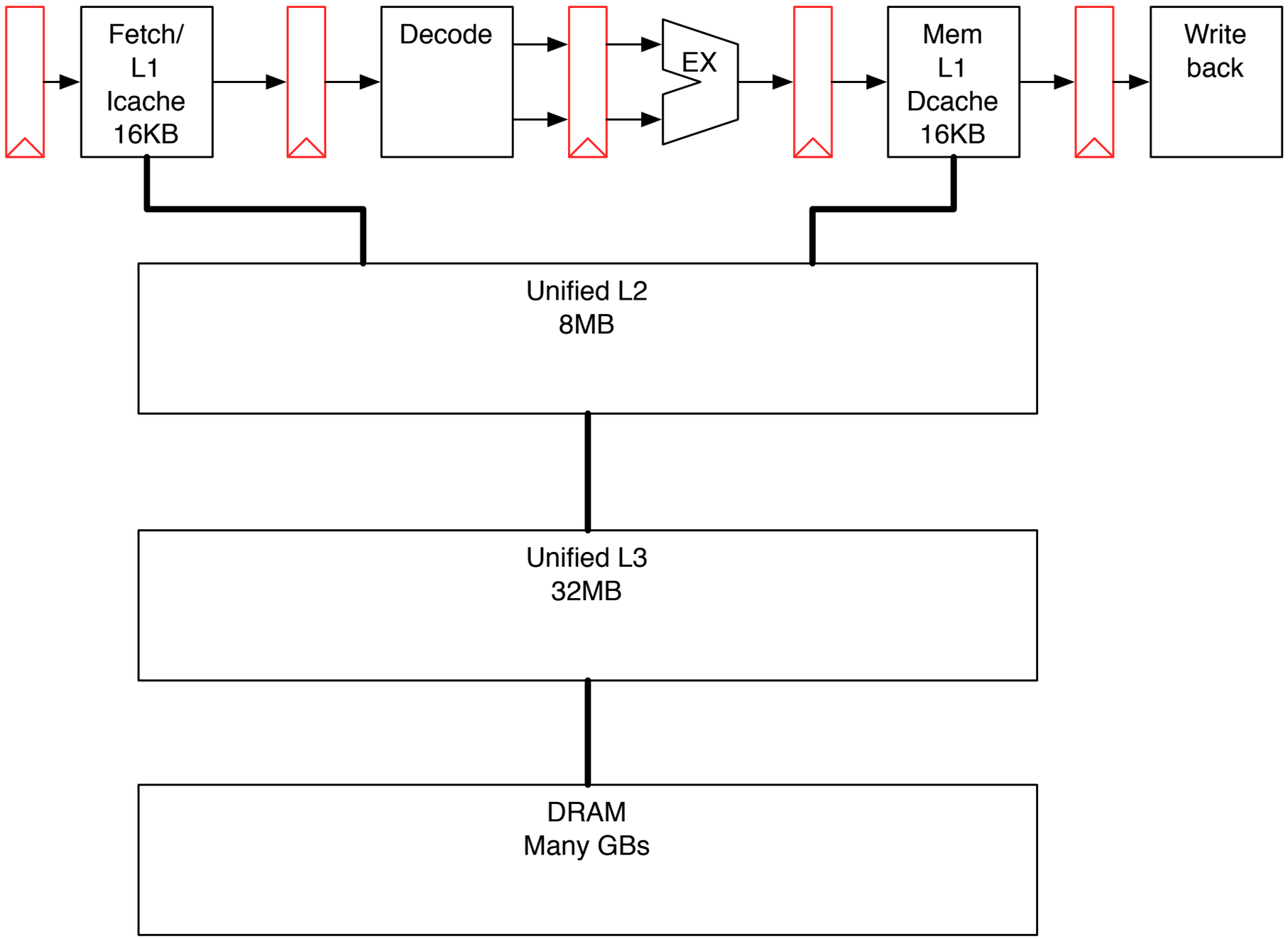
Cache Vocabulary

- Hit - The data was found in the cache
- Miss - The data was not found in the cache
- Hit rate - hits/total accesses
- Miss rate = 1- Hit rate
- Locality - see previous slides
- Cache line - the basic unit of data in a cache. generally several words.
- Tag - the high order address bits stored along with the data to identify the actual address of the cache line.
- Hit time -- time to service a hit
- Miss time -- time to service a miss (this is a function of the lower level caches.)

Cache Vocabulary

- There can be many caches stacked on top of each other
- if you miss in one you try in the "lower level cache" Lower level, mean higher number
- There can also be separate caches for data and instructions. Or the cache can be "unified"
- to whit:
  - the L1 data cache (d-cache) is the one nearest processor. It corresponds to the "data memory" block in our pipeline diagrams
  - the L1 instruction cache (i-cache) corresponds to the "instruction memory" block in our pipeline diagrams.
  - The L2 sits underneath the L1s.
  - There is often an L3 in modern systems.

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 L1 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)