# CMPE 110: Computer Architecture

## Week 6
## Memory Hierarchy / Cache

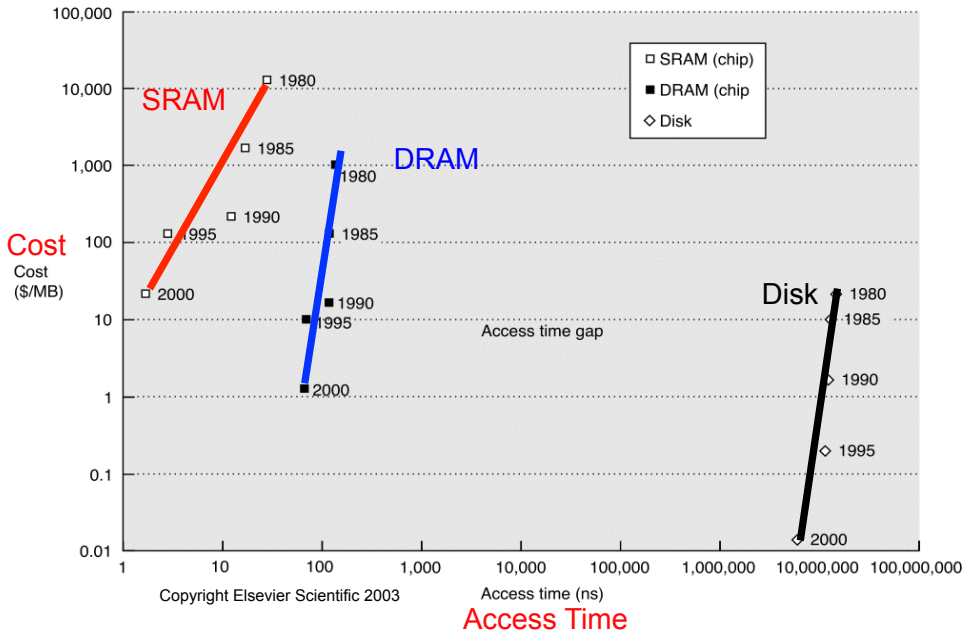
Jishen Zhao (http://users.soe.ucsc.edu/~jzhao/)

[Adapted in part from Jose Renau, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

## Reminder

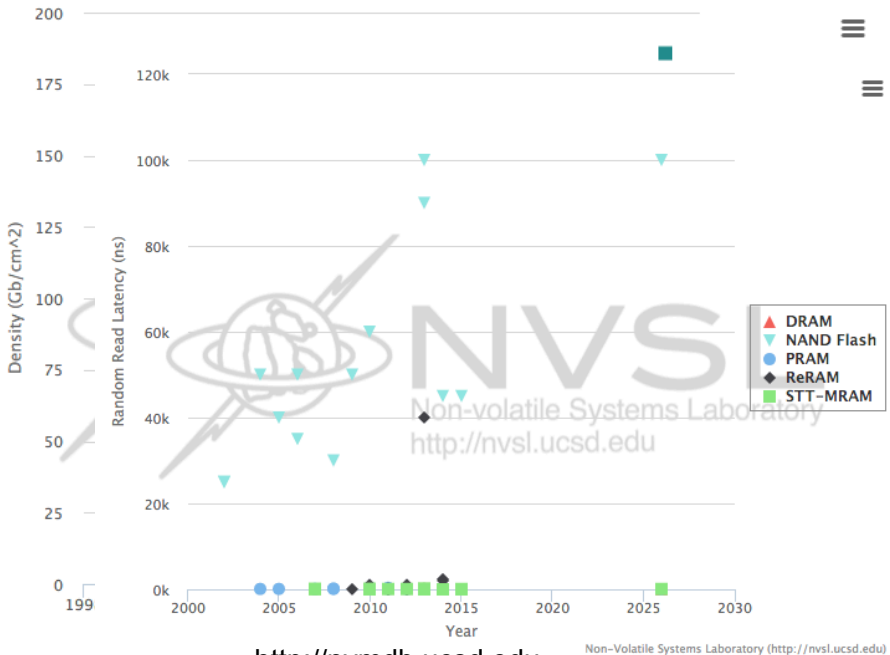
- Midterm1 exam papers can be picked up this week
  - Rebecca: A-G (ready)
  - Xin: H-M (ready)
  - Aziz: N-S
  - Narendra: T-Z
- Quiz 2 will be posted today
  - Due on Wednesday midnight

# Review: Memory Technology Trends



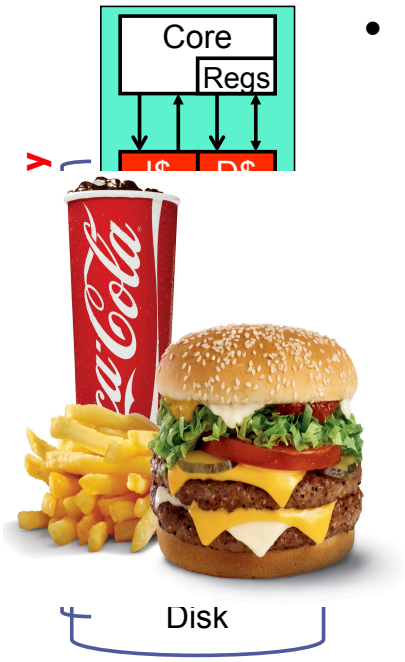



# Review: Memory Technology Trends




http://nvmdb.ucsd.edu

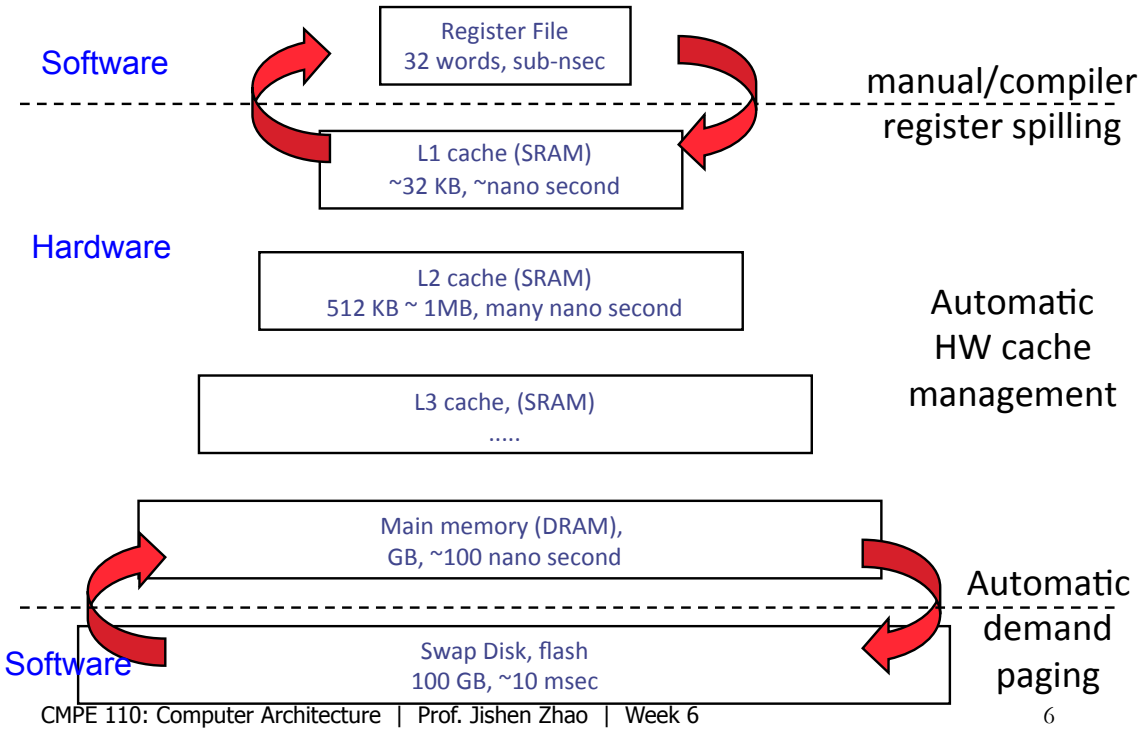
# Review: Memory Hierarchy: what, where, why




- What are the components in a memory hierarchy?
  - Caches
  - Main memory
  - Storage devices (e.g., hard drives, SSD, etc.)
- Where do these components locate?
  - On processor chip
  - Off processor chip
- Why do we need a hierarchy of memory
  - Goals: High **speed**, large **capacity**, low **cost**
  - No single memory technology can simultaneously achieve all three goals
  - Solution: build a Combo



# A Modern Memory Hierarchy



# Memory Hierarchy and Locality



## Big Observation: Locality (in general)

- One's recent past is a very good predictor of his/her near future.

- Temporal Locality: If you just did something, it is very likely that you will do the same thing again soon
  - since you are here today, there is a good chance you will be here again and again regularly

- Spatial Locality: If you did something, it is very likely you will do something similar/related (in space)
  - every time I find you in this room, you are probably sitting close to the same people

# Library Analogy


- Consider books in a library
- Library has lots of books, but it is slow to access
  - Far away (time to walk to the library)
  - Big (time to walk within the library)
- How can you avoid these latencies?
  - Check out books, take them home with you
    - Put them on desk, on bookshelf, etc.
  - But desks & bookshelves have limited capacity
    - Keep recently used books around (**temporal locality**)
    - Grab books on related topic at the same time (**spatial locality**)
    - Guess what books you'll need in the future (**prefetching**)



# Memory Locality

- A "typical" program has a lot of locality in memory references
  - typical programs are composed of "loops"

- Temporal: A program tends to reference the same memory location many times and all within a small window of time

- Spatial: A program tends to reference a cluster of memory locations at a time
  - most notable examples:
    - 1. instruction memory references
    - 2. array/data structure references

# Example: Locality & Caching

- Which memory accesses demonstrate spatial locality?
- Which memory accesses demonstrate temporal locality?

```
int sum = 0;
int x[1000];

for(int c = 0; c < 1000; c++) {
  sum += c;  // sum is access again and again for 1000 times

  x[c] = 0;   // if x[0] is accessed, we know that x[1], x[2], …,
              // x[999] will all be accessed
}
```



# Temporal and Spatial Locality

- Which memory accesses demonstrate spatial locality?
- Which memory accesses demonstrate temporal locality?

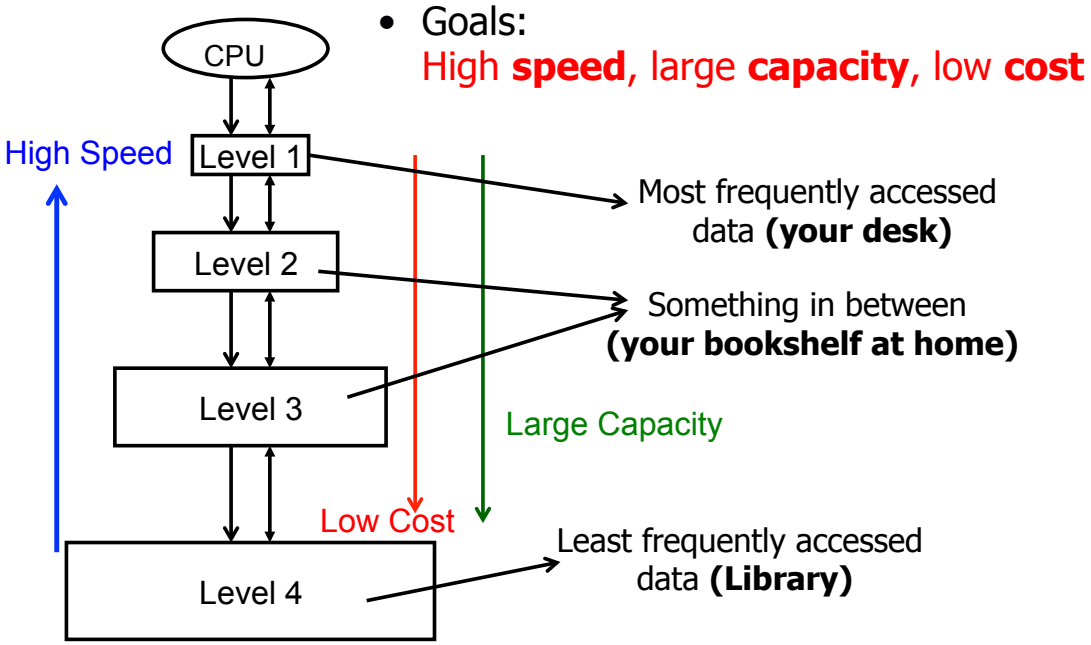
```
int sum = 0;
int x[1000];

for(int c = 0; c < 1000; c++) {
  sum += c;  // Temporal locality: A program tends to access
             the same memory location many times and all
             within a small window of time

  x[c] = 0;  // Spatial locality: A program tends to reference a
             cluster of memory locations at a time
}
```

# What we learned

- "Temporal locality"
- "Spatial locality"

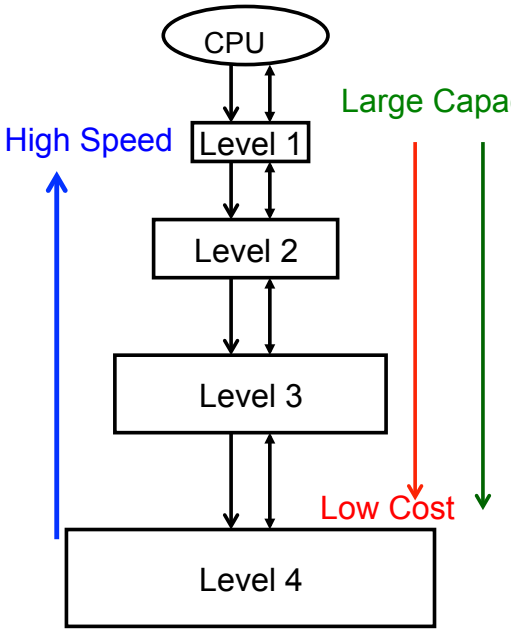


# Summary: basic idea of memory hierarchy




- Goals:
  High **speed**, large **capacity**, low **cost**

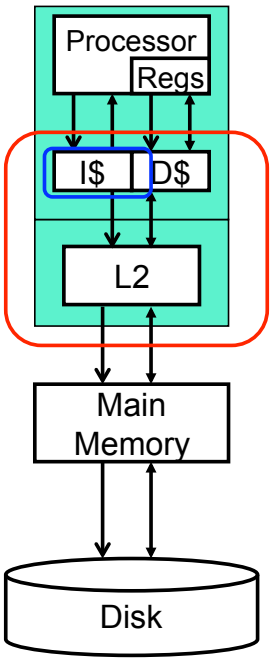
# Summary: basic idea of memory hierarchy




- Registers ↔ books on your desk
  - Actively being used, small capacity
- Caches ↔ bookshelves
  - Moderate capacity, pretty fast to access
- Main memory ↔ library
  - Big, holds almost all data, but slow
- Disk (virtual memory) ↔ inter-library loan
  - Very slow, but hopefully really uncommon



**Cache (not cash)**

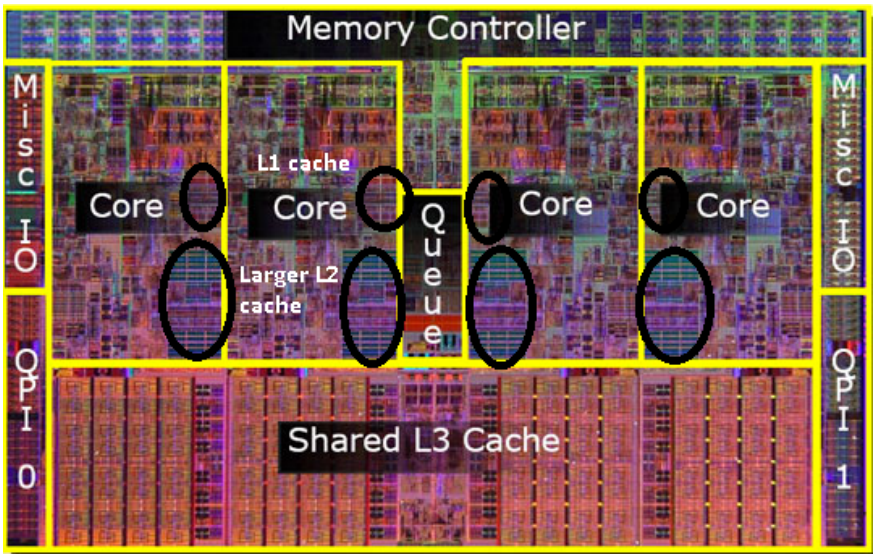



- I$ (instruction cache): read only
- D$ (data cache): read/write
- L2 (L3, L4, ..) are lower-level caches for both instruction and data

# Cache

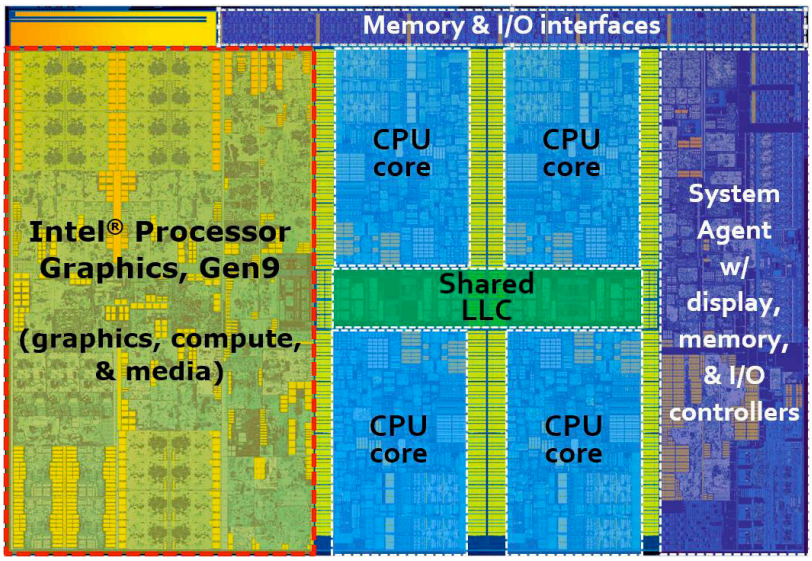


# Where are L1, L2, L3 caches?




Intel Core i7 (Nehalem, 2008, 45nm)

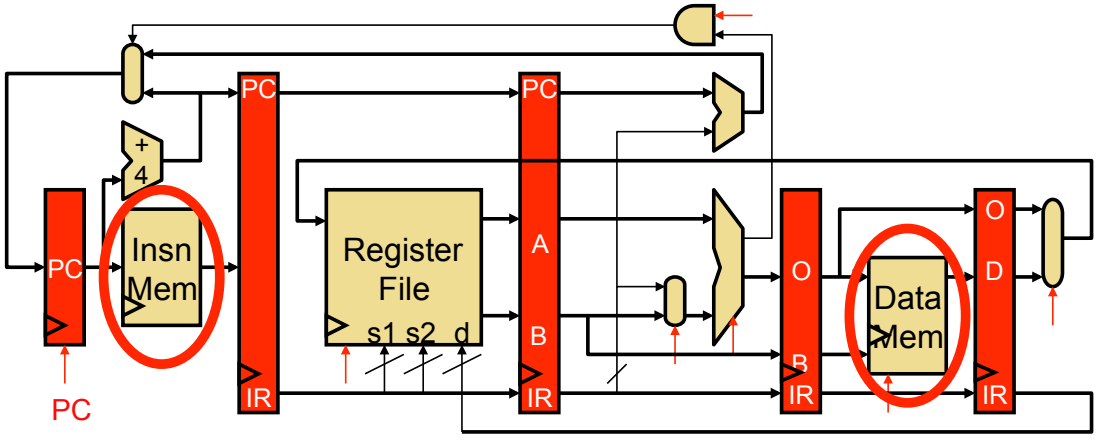
# Caches in the latest processors?




Intel Core i7 (Skylake, 2015, 14nm)



# Caches in the pipeline




lw R2, 100(R5)
sw R3, 30(R7)

Fetch Stage

Memory Stage

# How to exploit Locality with caches

## Exploit Temporal Locality

(Recently accessed data will be again accessed in the near future)

- Idea: Store recently accessed data in automatically managed fast memory (called cache)
- Anticipation: the data will be accessed again soon

## Exploit Spatial Locality

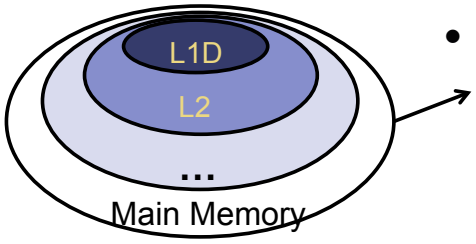
(Nearby data in memory will be accessed in the near future)

- Idea: Store addresses adjacent to the recently accessed one in automatically managed fast memory
  - Logically divide memory into equal size blocks
  - Fetch to cache the accessed block in its entirety
- Anticipation: nearby data will be accessed soon



# Cache basics

# Inclusive vs. exclusive caches

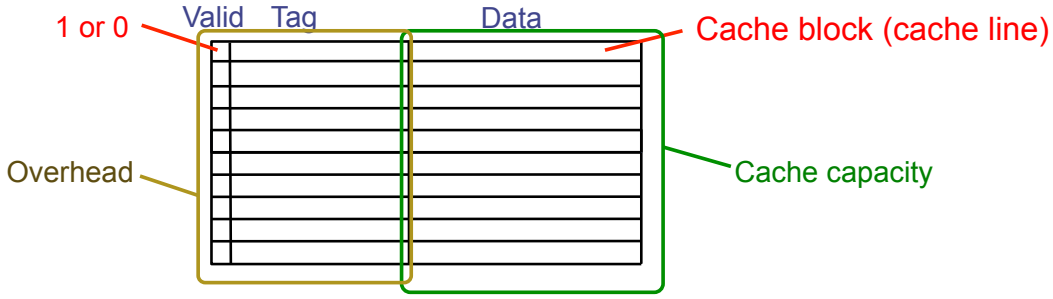



- Inclusive cache
  - A higher-level of cache stores a subset of lower-level caches
- Exclusive cache (e.g., AMD Athlon)
  - Data is guaranteed to be in at most one of the L1 and L2 caches, never in both
- Non-inclusive cache (e.g., Intel Pentium II, III, and 4)
  - Intermediate policy
  - Do not require that data in the L1 cache also reside in the L2 cache, although it may often do so

*Cheng et al., "LAP: Loop-Block Aware Inclusion Properties for Energy-Efficient Asymmetric Last Level Caches", ISCA 2016.*



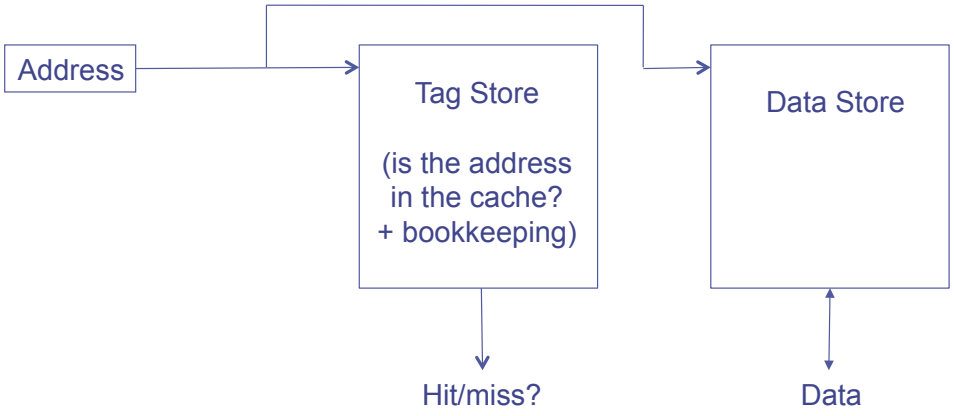
# Cache Basis




- When data referenced
  - HIT: If in cache, use cached data instead of accessing memory
  - MISS: If not in cache, bring block into cache (invalid → miss)
    - Go to the next level of cache to bring this data up
    - Have to kick something else out to do it, if it is full

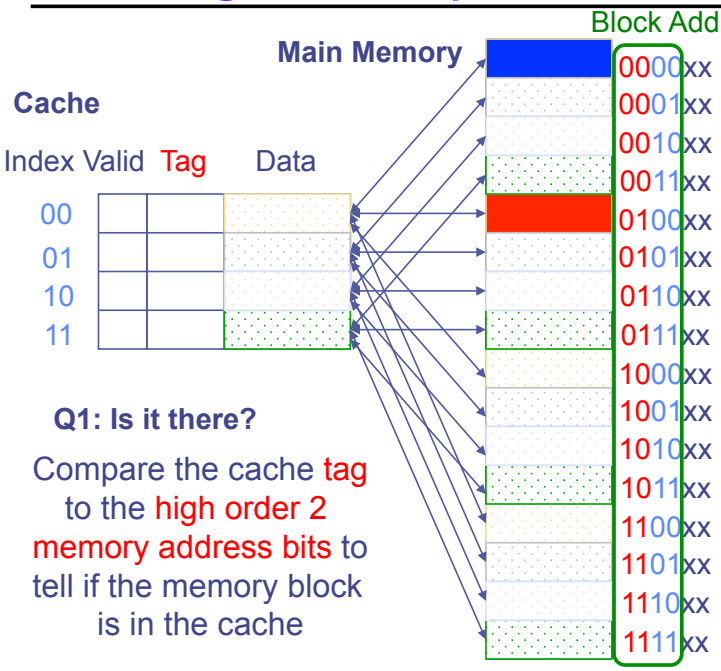
# Cache basis (cont.)

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?






# Caching: A Simple First Example




**Q1: Is it there?**

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

**32-bit (4-byte) data (1 word)**
**6-bit address**

- Need 2 bits (xx) to identify the 4 bytes of data
- The rest 4 bits used as "block address"

**Q2: How do we find it?**

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Index = (block address) modulo (# of blocks in the cache)