Caches: placement policy, replacement policy, and cache hierarchies

Yipeng Huang

Rutgers University

April 6, 2021

Table of contents

Looking ahead

Class plan

1. Today, Tuesday, 4/6: Caches: design parameters, direct mapped, fully associative, set associative.
2. Wednesday, 4/7: PA5 cache simulator and performance released
3. Thursday, 4/8: PA4 binary bomb lab due.
4. Monday, 4/12: Quiz due.

Table of contents

Cache, memory, storage, and network hierarchy trends

- Assembly programming view of computer: CPU and memory.
- Full view of computer architecture and systems: +caches, +storage, +network

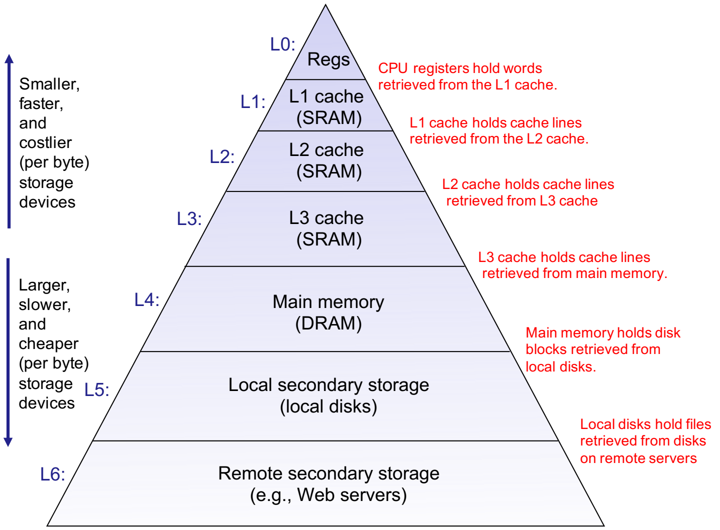


Figure: Memory hierarchy. Image credit CS:APP

Cache, memory, storage, and network hierarchy trends

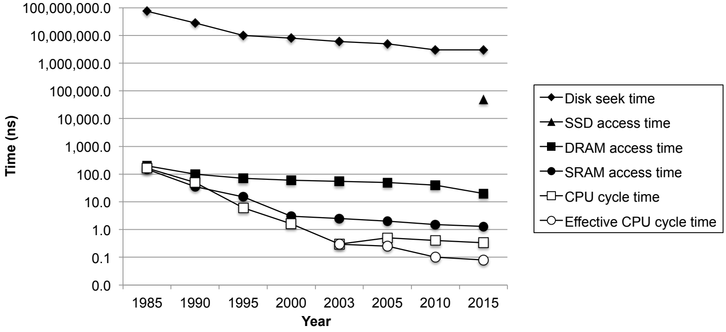


Figure: Widening gap: CPU processing time vs. memory access time. Image credit CS:APP

Topic of this chapter:

- Technology trends that drive CPU-memory gap.
- How to create illusion of fast access to capacious data.

Dynamic random-access memory (main memory)

- Needs refreshing every 10s of milliseconds
- 8GB typical in laptop; 1TB on each ilab machine
- Access time: 100 CPU clock cycles
- Memory gap: DRAM technological improvement slower relative to CPU/SRAM.

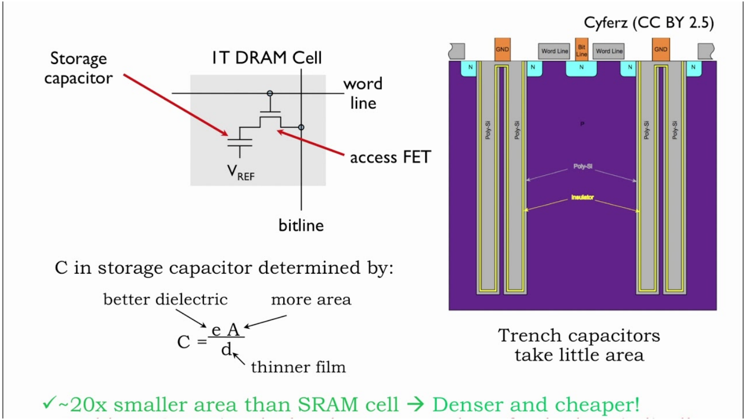


Figure: DRAM operating principle. Image credit ocw.mit.edu

Static random-access memory (caches)

- SRAM is bistable logic
- Access time: 1 to 10 CPU clock cycles
- Implemented in the same transistor technology as CPUs, so improvement has matched pace.

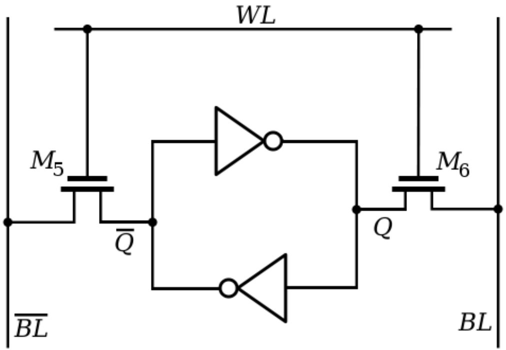


Figure: SRAM operating principle. Image credit Wikimedia

CPU / cache / DRAM main memory interface

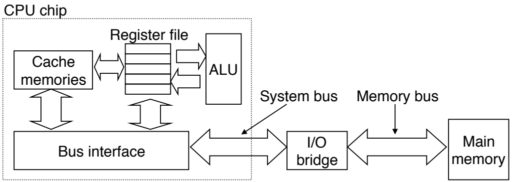
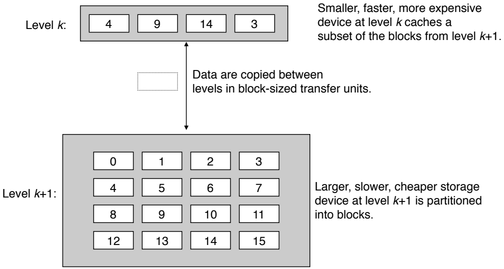


Figure: Cache resides on CPU chip close to register file. Image credit CS:APP



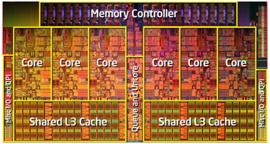


Figure: Intel 2020 Gulftown die shot. Image credit AnandTech

Figure: Cache stores a temporary copy from the slower main memory. Image credit CS:APP

Locality: How to create illusion of fast access to capacious data

From the perspective of memory hierarchy, locality is using the data in at any particular level more frequently than accessing storage at next slower level.

Well-written programs maximize locality

- Spatial locality
- Temporal locality

CPU / cache / DRAM main memory interactions

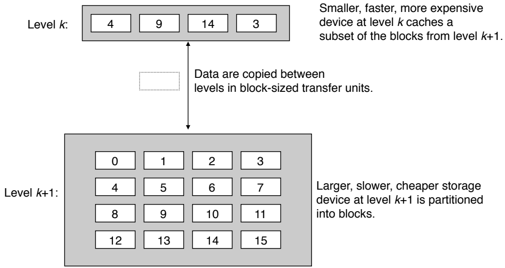


Figure: Cache stores a temporary copy from the slower main memory. Image credit CS:APP

When CPU loads (LD) from memory

- Cache read hit
- Cache read miss

When CPU stores (ST) to memory

- Cache write hit
- Cache write miss

Table of contents

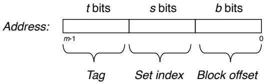
Cache placement policy (how to find data at address for read and write hit)

Several designs for caches

- Fully associative cache
- Direct-mapped cache
- N-way set-associative cache

Cache design options use *m*-bit memory addresses differently

- t-bit tag
- s-bit set index
- b-bit block offset



$m = t+s+b$

Figure: Memory addresses. Image credit CS:APP

Fully associative cache

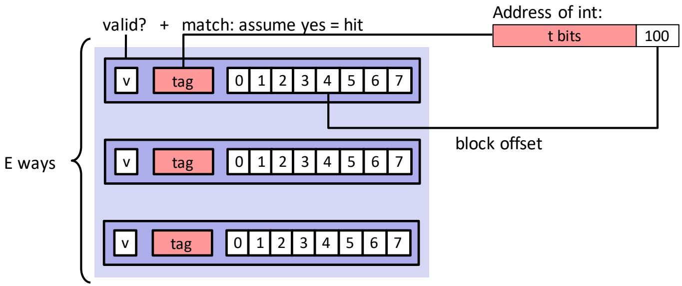


Figure: Fully associative cache. Image credit CS:APP

m-bit memory address split into:

- *t*-bit tag
- *b*-bit block offset

(s=0)

Fully associative cache

4-byte integer at address 0b00001100 (m=8)
always store 8 bytes in a row
4-byte integer at address 0b00001100+4

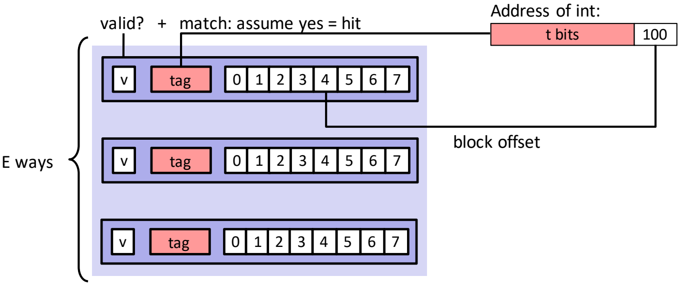


Figure: Fully associative cache. Image credit CS:APP

b-bit block offset

- here, $b = 3$
- The number of bytes in a block is $B = 2^b = 2^3 = 8$
- A block is the minimum number of bytes that can be cached
- Good for capturing spatial locality, short strides

Fully associative cache

READ 4-byte integer at address 0b00001100 (m=8)
tag = 0b00001 (t=5)
For fully associative cache design, the tag is a ID for blocks

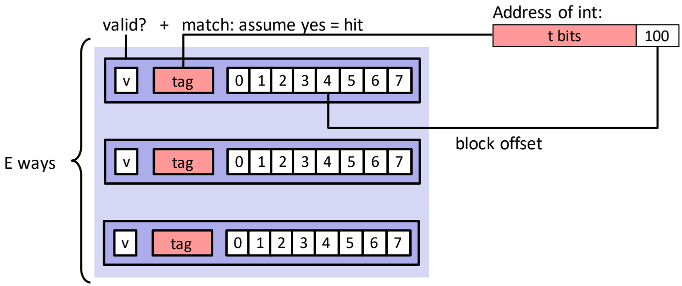


Figure: Fully associative cache. Image credit CS:APP

t-bit tag

- here,
  $t = m - b = m - 3$
- When CPU wants to read from or write to memory, all t-bits in tag need to match for read/write hit.

Fully associative cache

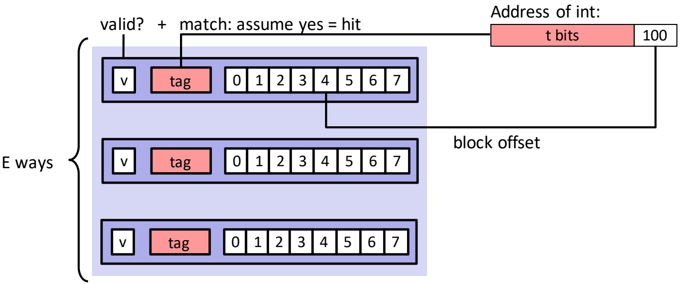


Figure: Fully associative cache. Image credit CS:APP

Full associativity

- Blocks can go into any of E ways
- Here, $E = 3$
- Good for capturing temporal locality: cache hits can happen even with intervening reads and writes to other tags.

Fully associative cache

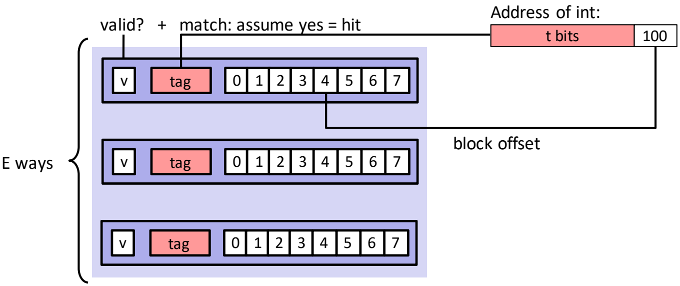


Figure: Fully associative cache. Image credit CS:APP

Capacity of cache

- Total capacity of fully associative cache in bytes: $C = EB = E * 2^b$
- Here, $C = E * 2^b = 3 * 2^3 = 24$ bytes

Fully associative cache

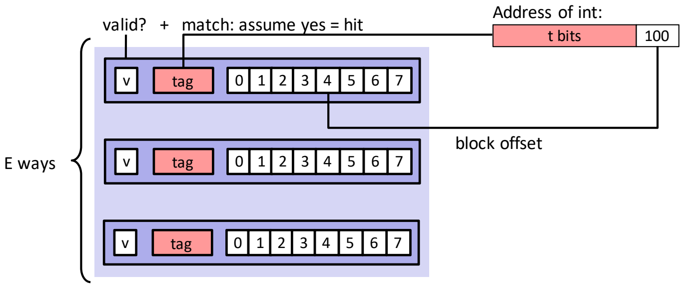


Figure: Fully associative cache. Image credit CS:APP

Strengths

- Blocks can go into any of E-ways.
- Hit rate is only limited by total capacity.

Weaknesses

- Searching across all valid tags is expensive.
- Figuring out which block to evict on read/write miss is also expensive.
- Requires a lot of

Direct-mapped cache

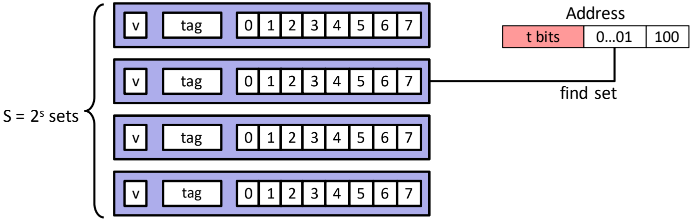


Figure: Direct-mapped cache. Image credit CS:APP

m-bit memory address split into:

- t-bit tag
- s-bit set index
- b-bit block offset

Direct-mapped cache

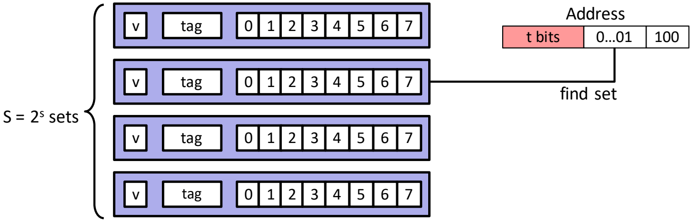


Figure: Direct-mapped cache. Image credit CS:APP

b-bit block offset

- here, $b = 3$
- The number of bytes in a block is $B = 2^b = 2^3 = 8$
- A block is the minimum number of bytes that can be cached
- Good for capturing spatial locality, short strides

Direct-mapped cache

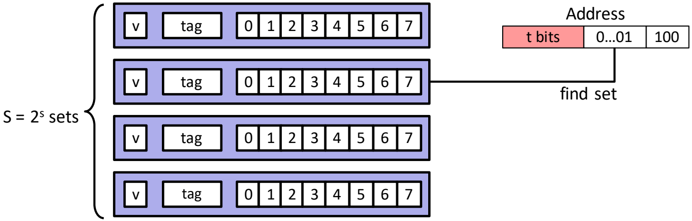


Figure: Direct-mapped cache. Image credit CS:APP

s-bit set index

- here, $s = 2$
- The number of sets in cache is $S = 2^s = 2^2 = 4$
- A hash function that limits exactly where a block can go
- Good for further increasing ability to exploit spatial locality, short strides

Direct-mapped cache

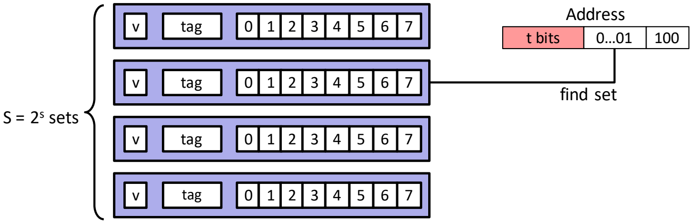


Figure: Direct-mapped cache. Image credit CS:APP

t-bit tag

- here,
  $t = m - s - b = m - 2 - 3$
- When CPU wants to read from or write to memory, all t-bits in tag need to match for read/write hit.

Direct-mapped cache

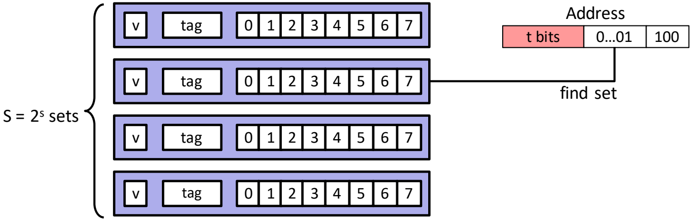


Figure: Direct-mapped cache. Image credit CS:APP

Full associativity

- In direct-mapped cache, blocks can go into only one of $E = 1$ way

Direct-mapped cache

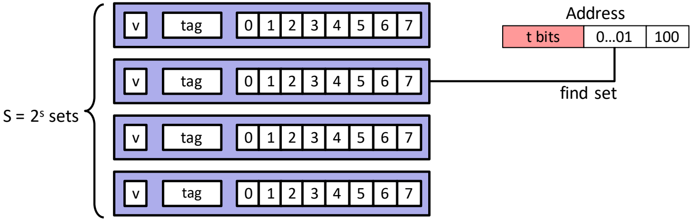


Figure: Direct-mapped cache. Image credit CS:APP

Capacity of cache

- Total capacity of fully associative cache in bytes: $C = SEB = 2^s * E * 2^b$
- Here, $C = 2^s * E * 2^b = 2^2 * 1 * 2^3 = 32$ bytes

Direct-mapped cache

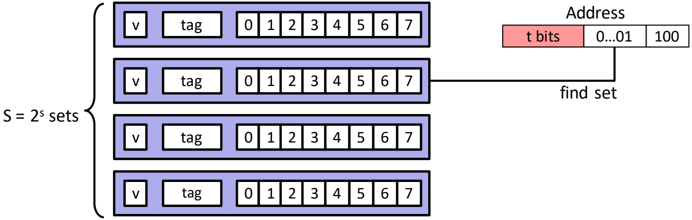


Figure: Direct-mapped cache. Image credit CS:APP

Strengths

- Simple to implement.
- No need to search across tags.

Weaknesses

- Can lead to surprising thrashing of cache with unfortunate access patterns.
- Unexpected conflict misses independent of cache capacity.