Introduction to Caching

COS 316 Lecture 8

Amit Levy

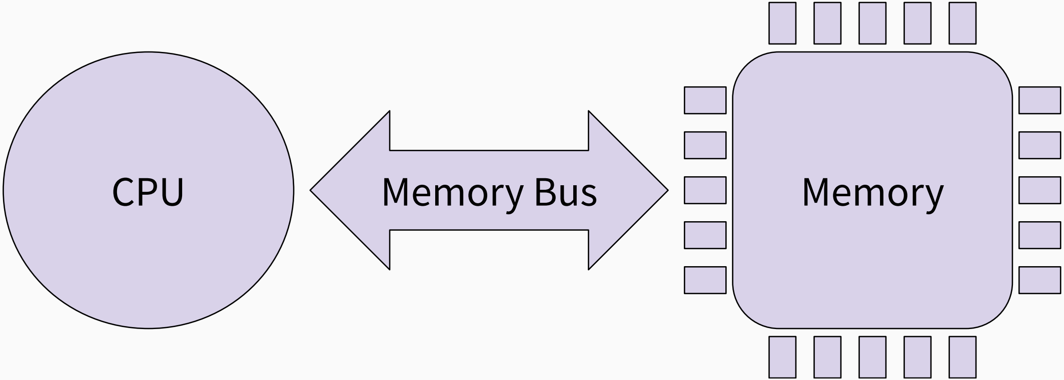


Figure 1: CPU Connected Directly to Memory

How long to run this code?

Characteristics

- CPU Instructions: 0.5ns (2GHz)
- Memory access: 100ns

```
int arr[1000];
for (i = 0; i < arr.len(); i++) { ++arr[i]; }
```

```
loop: ldr r2, [r0, #0]
      add r2, r2, #1
      str r2, [r0, #0]
      subs r0, r0, #4
      bne <loop>
```

```
loop: ldr r2, [r0, #0]
      add r2, r2, #1
      str r2, [r0, #0]
      subs r0, r0, #4
      bne <loop>
```

1. $2.5\mu S\ (2,500nS)$
2. $30\mu S\ (30,000nS)$
3. $201.5\mu S\ (201,500ns)$

Why not just make everything fast?

Type	Access Time	Typical Size	$/MB
Registers	$< 0.5ns$	~256 bytes	$\$1000$
SRAM/"Cache"	$5ns$	1-4MB	$\$100$
DRAM/"Memory"	$50ns$	GBs	$\$0.01$
Magnetic Disk	$5ms$	TBs	$\$0.000001$

- High cost of fast storage
- Physical limitations
- Not necessarily possible—e.g. accessing a web page across the world

A Solution: Caching

What is caching?

- Keep *all* data in bigger, cheaper, slower storage
- Keep *copies* of “active” data in smaller, more expensive, faster storage

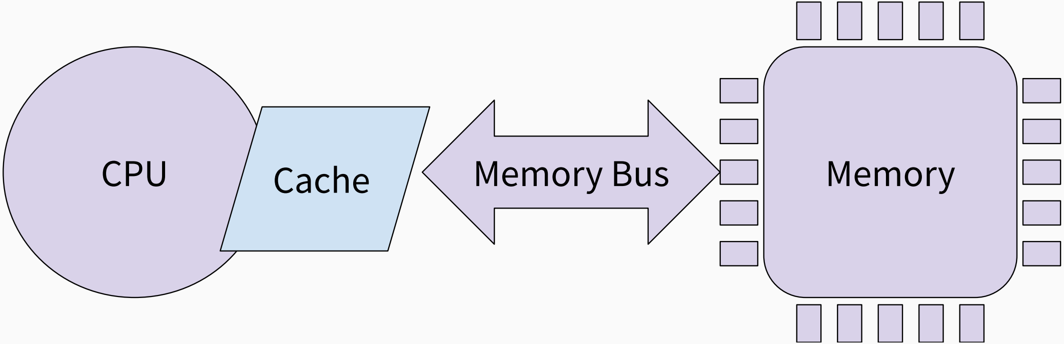


Figure 2: CPU + Cache + Memory

What do we cache?

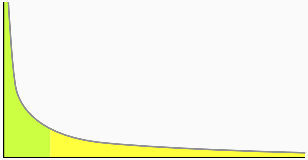
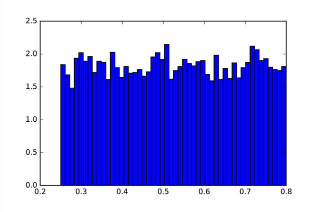
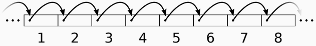
- Data stored verbatim in slower storage
- Previous computations—recomputation is also a kind of slow storage
- Examples:
  - CPU memory hierarchy
  - File system page buffer
  - Content distribution network
  - Web application cache
  - Database cache
  - Memoization

Locality

- Temporal locality: nearness in time
    - Data accessed now probably accessed recently
    - Useful data tends to continue to be useful
- Spatial locality: nearness in name
    - Data accessed now “near” previously accessed data
    - Memory addresses, files in the same directory, frames in a video...

When is caching effective?

Which of these workloads could we cache effectively?

Repeated Access	Random Access	Sequential access
		
A few popular items	No pattern to accesses	Access items in order
E.g. most social media	E.g. large hash tables	E.g. streaming a video

Caching Terminology

- **Hit**: when a requested item was in the cache
- **Miss**: when a requested item was *not* in the cache
- **Hit rate** and **Miss rate**: proportion of hits and misses, respectively
- **Hit time** and **Miss time**: time to access item in cache and not in cache, respectively

Effective access time

Effective aceess time is a function of:

- Hit and miss rates
- Hit and miss times

$$t_{effective} = (hit\_rate)t_{hit} + (1 - hit\_rate)t_{miss}$$

aka, Average Memory Access Time (AMAT)

Characterizing a Caching System

- *Effective access time*
- Look-aside vs. Look-through
- Write-through vs. Write-back
- Allocation policy
- Eviction Policy

Who handles misses?

What happens when a requested item is not in the cache?

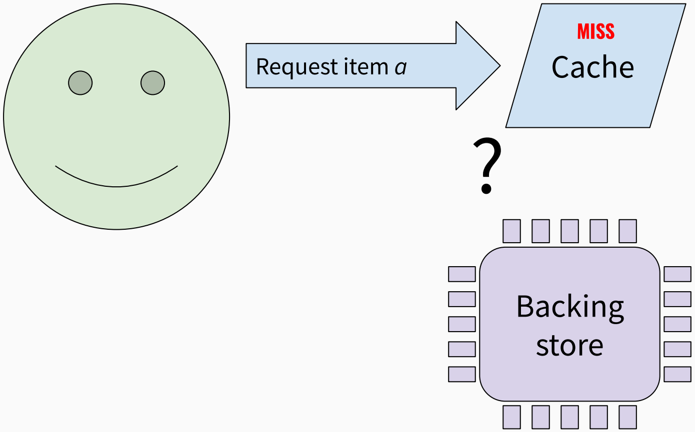


Figure 3: User requests an item not in the cache

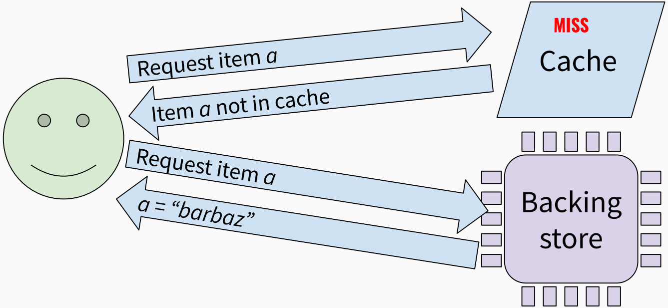


Figure 4: Look-aside Cache

- Advantages: easy to implement, flexible
- Disadvantages: application handles consistency, can be slower on misses

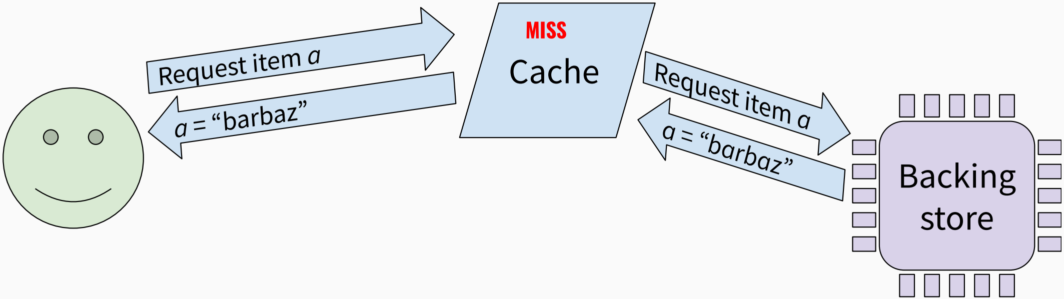


Figure 5: Look-through Cache

- Advantages: helps maintain consistency, simple to program against
- Disadvantages: harder to implement, less flexible

Handling Writes

- Caching creates a replica/copy of the data
- When you write, the data needs to be synchronized *at some point*
    - But when?

Write-through

Write to backing store on every update

- Advantages:
  - Cache and memory are always consistent
  - Eviction is cheap
  - Easy to implement
- Disadvantages:
  - Writes are at least as slow as writes to the backing store

Write-back

Update only in the cache. Write "back" to the backing store only when evicting item from cache

- Advantages:
  - Writes always at cache speed
  - Multiple writes to same item combined
  - Batch writes of related items
- Disadvantages:
  - More complex to maintain consistency
  - Eviction is more expensive

Write-allocate vs. Write-no-allocate

When writing to items *not* currently in the cache, do we bring them into the cache?

Yes == Write-Allocate

- Advantage: Exploits temporal locality: written data likely to be access again soon

No == Write-No-Allocate

- Advantage: Avoids spurious evictions if data not accessed soon

Eviction policies

Which items to we evict from the cache when we run out of space?

Many possible algorithms:

- Least Recently Used (LRU), Most Recently Used (MRU)
- Least Frequently Used (LFU)
- First-In-First-Out (FIFO), Last-In-First-Out (LIFO)
- ...

Deciding factors include:

- Workload
- Performance

Challenges in Caching

- Speed: making the cache itself fast
- Cache Coherence: dealing with out-of-sync caches
- Performance: maximizing hit rate
- Security: avoiding information leakage through the cache

Remainder of this Section

- Caching in the CPU Memory Hierarchy
- File system page buffer
- Caching in the Web (Prof. Freedman)
- Assignment 3: Implement a look-aside, write-allocate cache

References