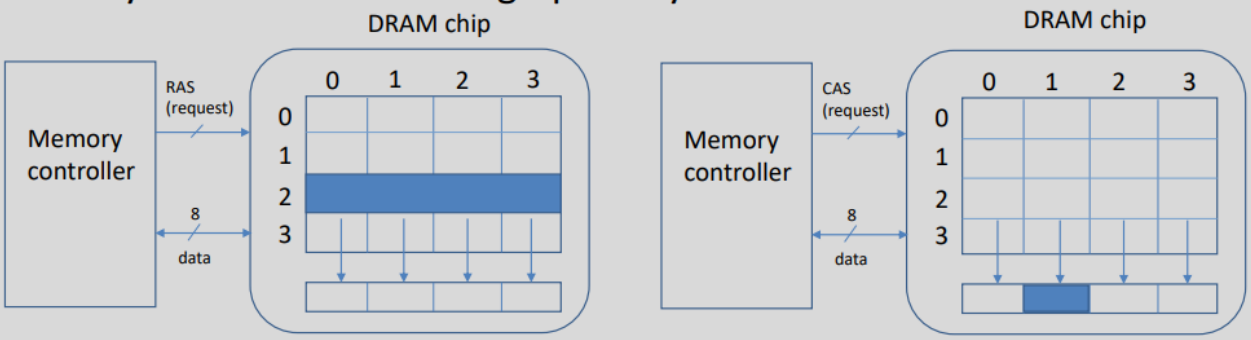
Caches and Memory Hierarchy

Storage technology

- ***RAM (Random Access Memory):*** two types, *static* (SRAM) and *dynamic* (DRAM). Information remains stored as long is powered. SRAM used for cache memory and DRAM for main memory and frame buffer of graphics system.



- Enhanced RAMs: *SDRAM, FPM DRAM, EDO DRAM, DDR DRAM, VRAM.*

RAM is volatile

- ***ROM (Read Only Memory):*** The non-volatile memories retain the information stored even if it is power off. Some type of ROMs can also be written to. ROMs are differentiate by the number of times they can be reprogrammed and by the mechanism to reprogram them.

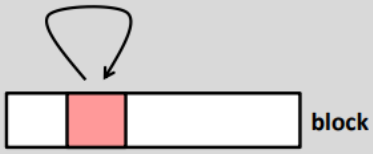
- Type of ROMs: PROM, EPROM, EEPROM, flash memory.

- Hard Disk Drives: PATA, SATA, SCSI,SSD

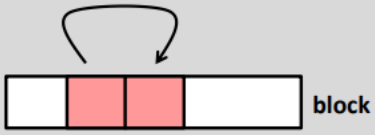
Principle of locality

- ***Locality***: Programs tend to use data and instructions with addresses near or equal to those they have used recently

- ***Temporal locality***:
  - Recently references items are likely to be referenced again in the near future



- ***Spatial locality***:
  - Items with nearby addresses tend to be referenced close together in time



Example: Locality? Data? Instructions?

- **Data:**
  - Temporal: `sum` referenced in each iteration
  - Spatial: array `a[]` accessed in stride-1 pattern

- **Instructions:**
  - Temporal: cycle through loop repeatedly
  - Spatial: reference instructions in sequential memory order

```
sum = 0;
for (i = 0; i < n; i++)
{
    sum += a[i];
}
return sum;
```

Locality Example #1

```
int sum_array_cols(int a[M][N])
{
  int i, j, sum = 0;
  for (j = 0; j < N; j++) {
    for (i = 0; i < M; i++) {
      sum += a[i][j];
    }
  }
  return sum;
}
```

M = 3, N = 4

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Access Pattern:
stride = ?

1) a[0][0]
2) a[1][0]
3) a[2][0]
4) a[0][1]
5) a[1][1]
6) a[2][1]
7) a[0][2]
8) a[1][2]
9) a[2][2]
10) a[0][3]
11) a[1][3]
12) a[2][3]

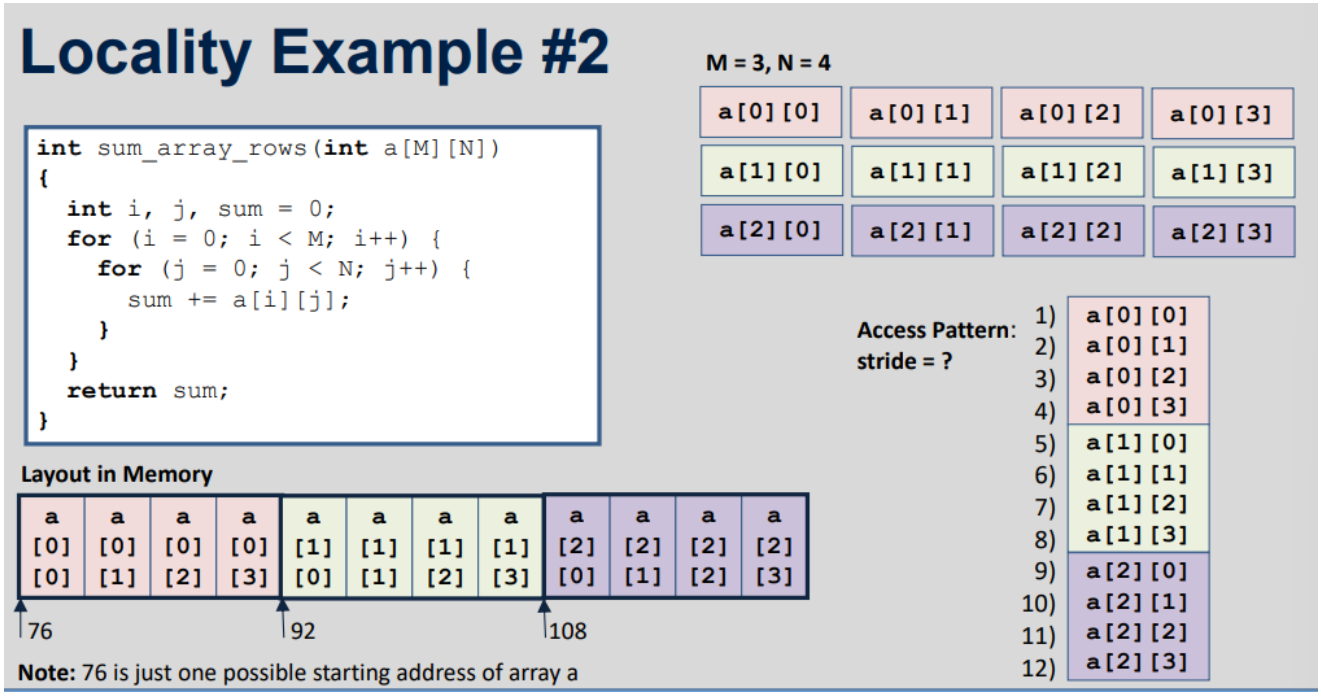
Layout in Memory

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

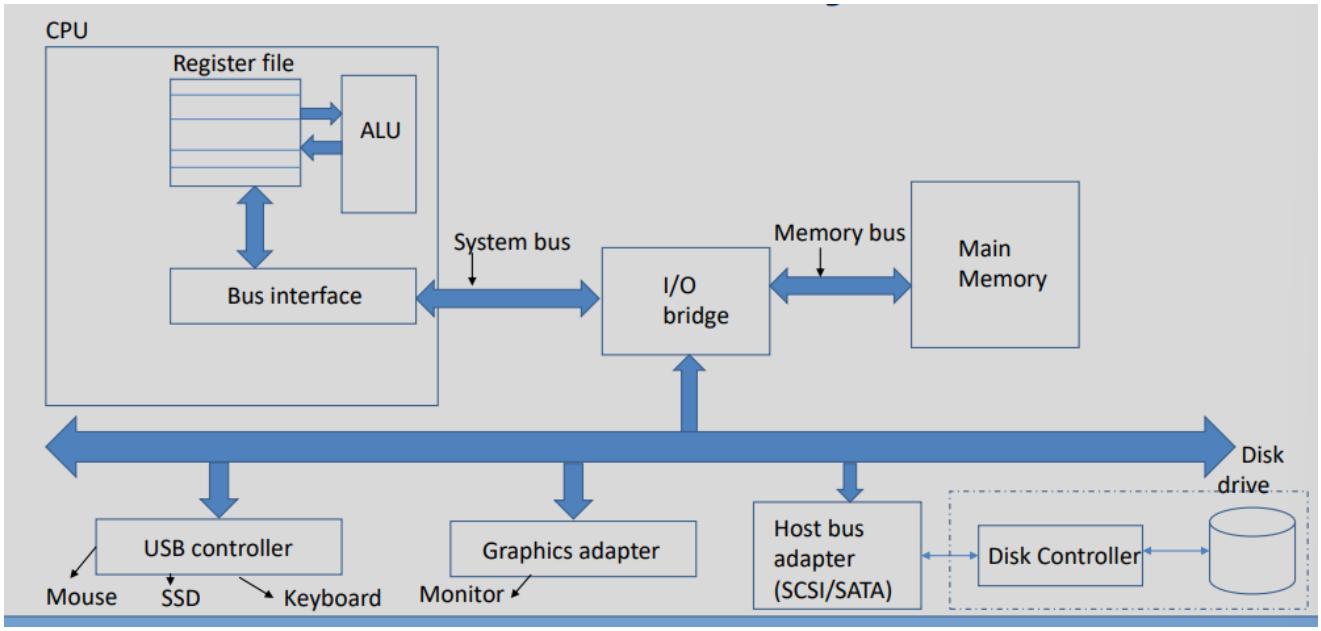
76 92 108

Note: 76 is just one possible starting address of array a

Locality Example #2



Problem: Processor-memory bottleneck



???

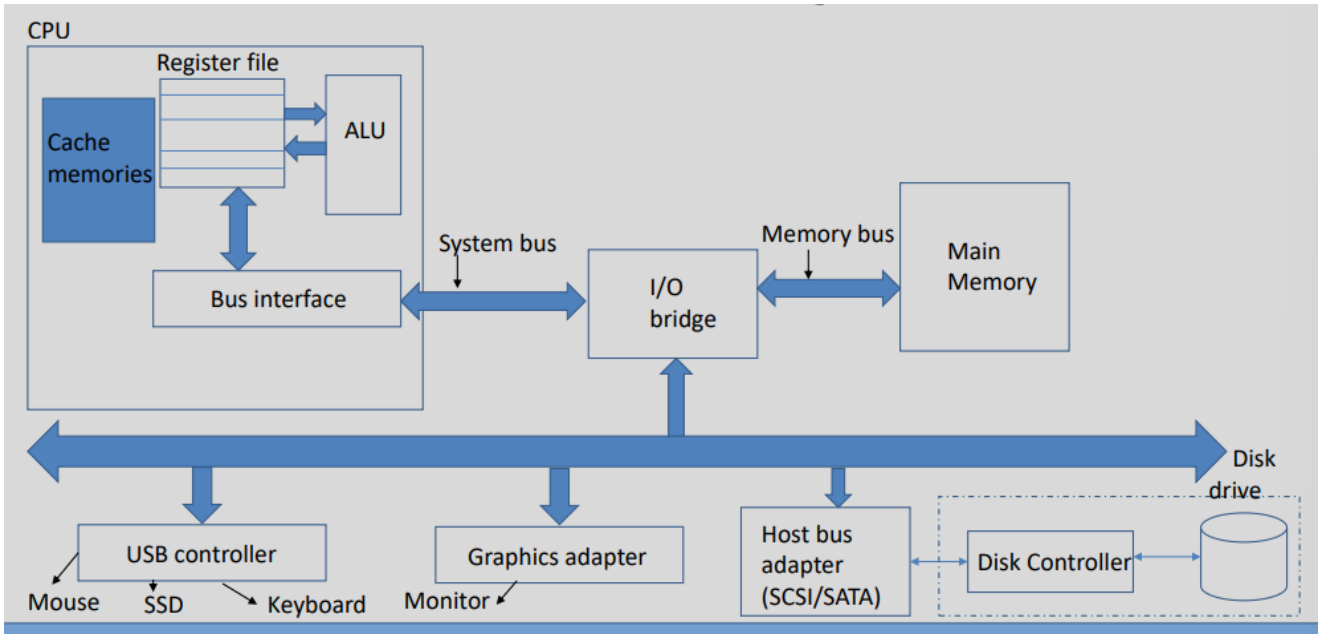
Cache

- ***Pronunciation:*** "cash". We will abbreviate it as "$".

- ***Definition (computer):***
  Computer memory with short access time used for the storage of frequently or recently used instructions (code) or data.

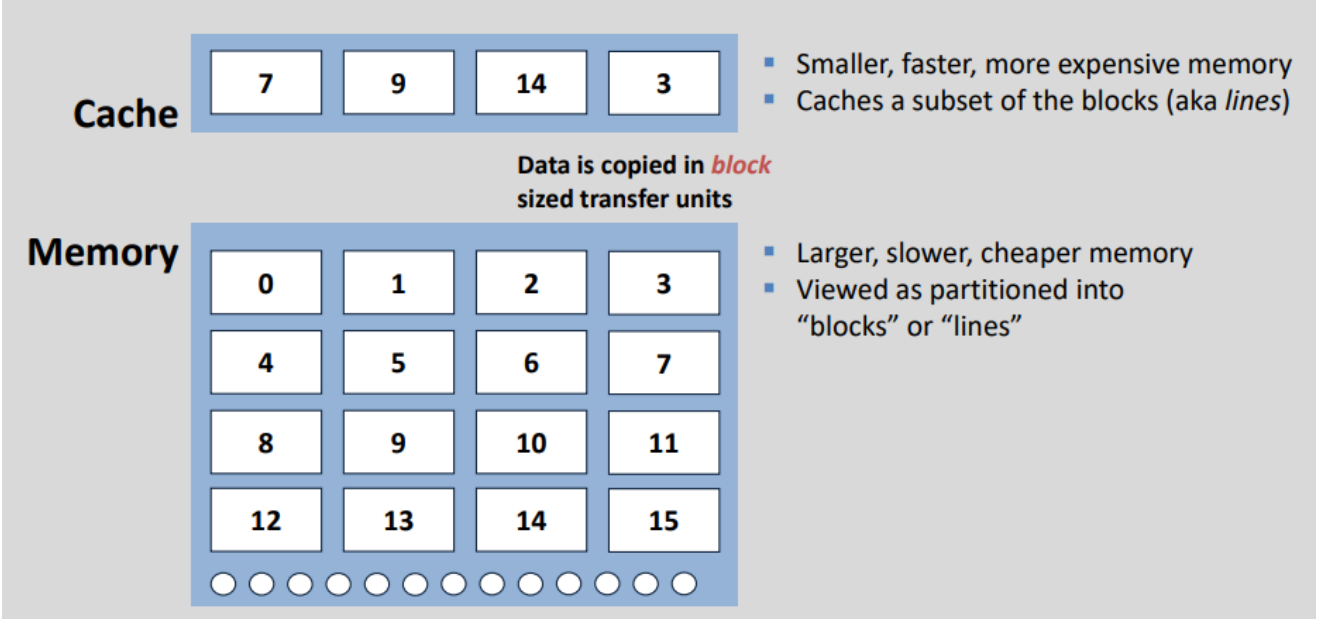
- ***Definition (more general computing):***
  Used to optimise data transfers between any system elements with different characteristics (e.g., network interface cache, I/O cache, etc.).

Solution: Processor-memory bottleneck



???

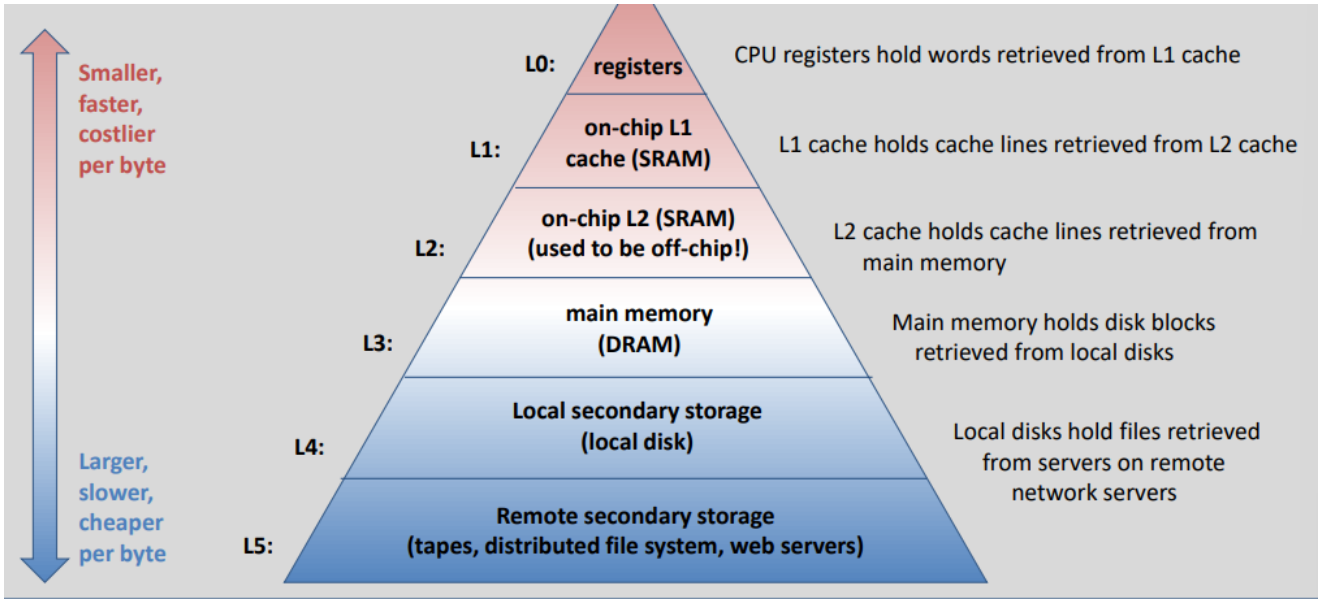
General Cache Mechanics



Memory Hierarchies

- **Some fundamental and enduring properties of hardware and software systems:**
  - Faster storage technologies almost always cost more per byte and have lower capacity
  - The gap between memory technology speeds are widening
    - True for: registers ↔ cache, cache ↔ DRAM, DRAM ↔ disk, etc.
  - Well-written programs tend to exhibit good locality
- **These properties complement each other**
  - They suggest an approach for organizing memory and storage systems known as ***memory hierarchy***

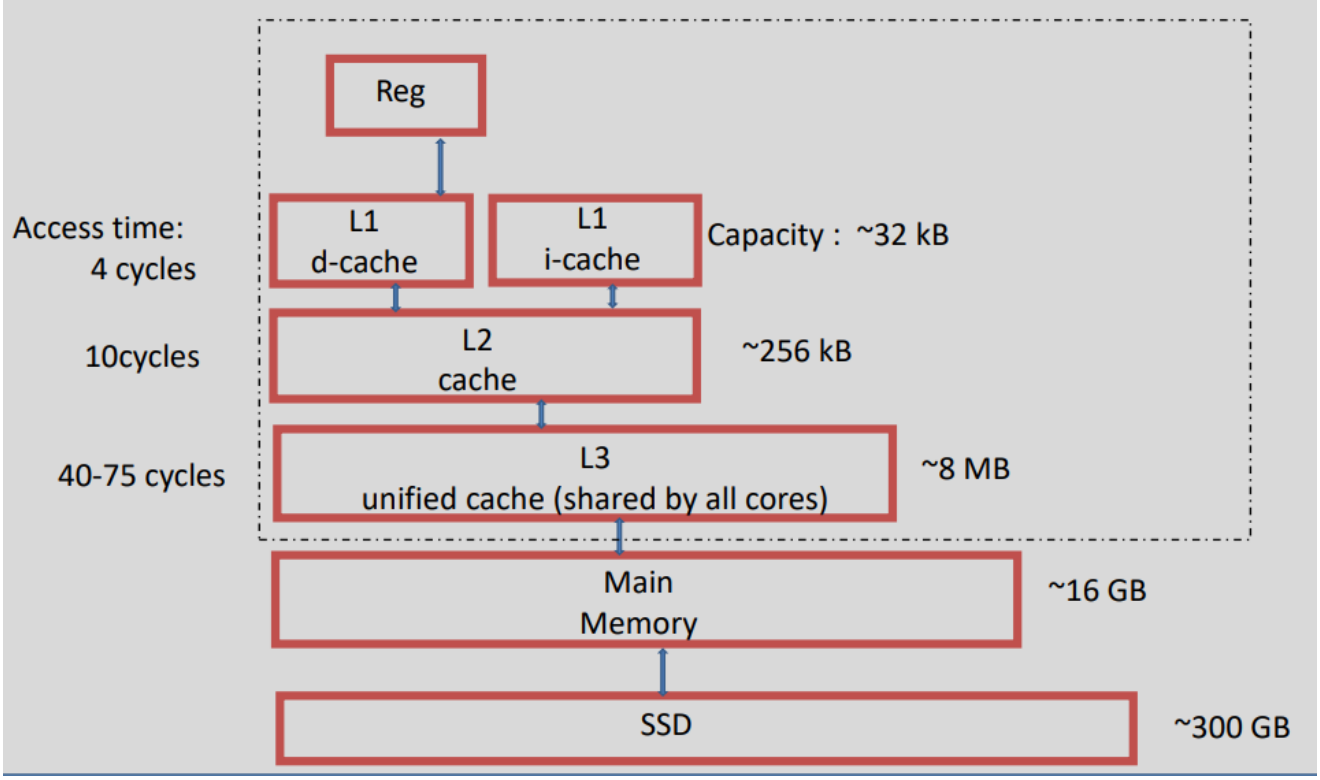
An Example Memory Hierarchy



Examples of caching in the hierarchy

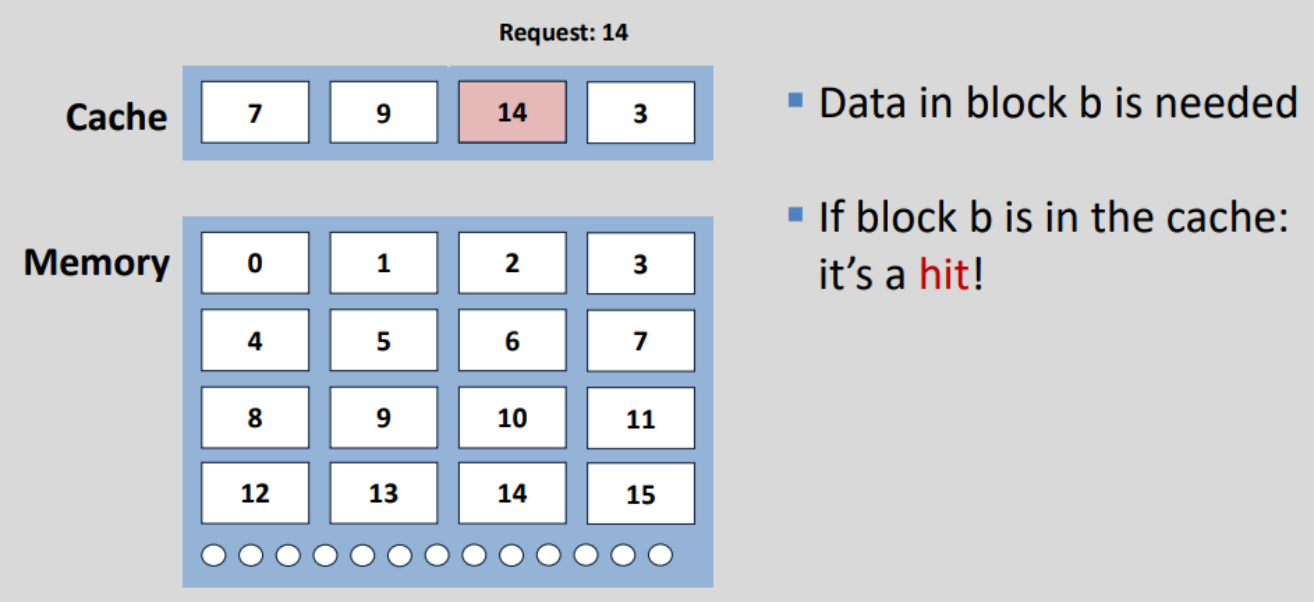
Cache type	What is cached?	Where is it cached?	Latency (cycles)	Managed by
Registers	4/8-byte words	CPU core	0	Compiler
TLB	Address translation	On-chip TLB	0	Hardware
L1 cache	64-byte blocks	On-chip L1	4	Hardware
L2 cache	64-byte blocks	On-chip L2	10	Hardware
Virtual Memory	4kB page	Main memory (RAM)	200	Hardware + OS
Buffer cache	4kB sectors	Main memory	200	OS
Network buffer cache	Parts of files	Local disk	10’000’000	SMB/NSF client
Browser cache	Web pages	Local disk	10’000’000	Web browser
Web cache	Web pages	Remote server disks	1’000’000’000	Web proxy server

Cache hierarchy: Intel Core i7

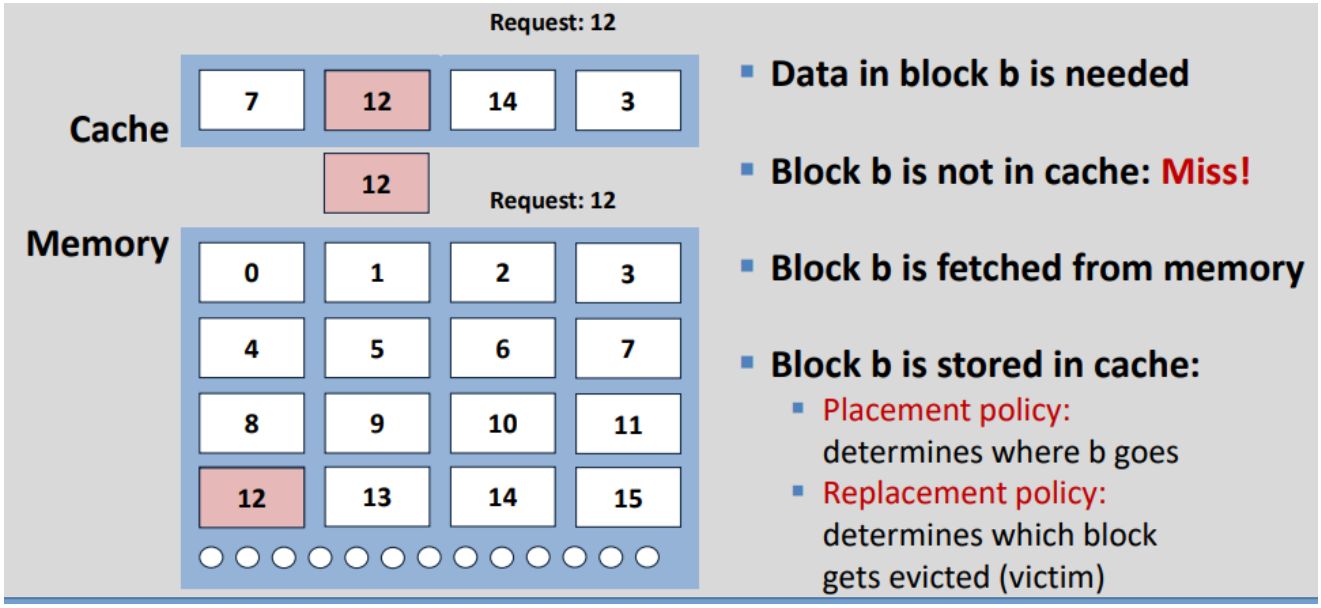


Cache Basics

General Cache Concepts: Hits



General Cache Concepts: Miss



Cache Performance Metrics

- **Huge difference between the cost of a cache hit and a cache miss**
  - Could be 100x speed difference between accessing cache and main memory (measured in *clock cycles*)
- **Miss Rate (MR)**
  - Fraction of memory references not found in cache (misses / accesses) = 1 – Hit Rate
- **Hit Time (HT)**
  - Time to deliver a block in the cache to the processor
    - Includes time to determine whether the block is in the cache
- **Miss Penalty (MP)**
  - Additional time required because of a miss

Cache Performance

- **Two things hurt the performance of a cache:**
  - Miss rate and miss penalty
- **Average Memory Access Time (AMAT)**: average time to access memory considering both hits and misses

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- 99% hit rate is twice as good as 97% hit rate!
  - Assume HT of 1 clock cycle and MP of 100 clock cycles
  - 97%: AMAT = $1 + (1 - 0.97) \times 100 = 1 + 3 = 4$ clock cycles
  - 99%: AMAT = $1 + (1 - 0.99) \times 100 = 1 + 1 = 2$ clock cycles

Can we have more than one cache?

- **Why would we want to do that?**
  - Avoid going to memory!
- **Typical performance numbers:**
  - Miss Rate:
    - L1 MR = 3-10%
    - L2 MR = Quite small (e.g., <1%), depending on parameters, etc.
  - Hit Time:
    - L1 HT = 4 clock cycles
    - L2 HT = 10 clock cycles
  - Miss Penalty:
    - P = 50-200 cycles for missing in L2 and going to main memory
    - Trend: increasing!

Summary

- **Memory hierarchy**
  - Successively higher levels contain "most used" data from lower levels
  - Exploits *temporal* and *spatial* locality
  - Caches are intermediate storage levels used to optimise data transfers between any system elements with different characteristics
- **Cache Performance**
  - Ideal case: found in cache (hit)
  - Bad case: not found in cache (miss), search in next level
  - Average Memory Access Time (AMAT) = $\mathrm{HT} + \mathrm{MR} \times \mathrm{MP}$
    - Hurt by Miss rate and Miss Penalty

Cache organization

Cache Organisation (1)

- **Cache Set (S)**
  - A cache with M = 2^m unique addresses is organised as a group of S = 2^s *cache sets*. Each cache set consists of E *cache lines*.
- **Block Size (B)**
  - Each cache lines consists of B = 2^b *blocks* of data.
  - Given in bytes and always a power of 2 (e.g., 64 B)
  - Blocks consist of adjacent bytes, for spatial locality!
- **Offset field**
  - Low-order $\log_2(\mathbf{B})$ = **b** bits of address tell you which byte within a block
    - (address) $mod\ 2^n = n$ lowest bits of address
  - (address) modulo (# of bytes in a block)

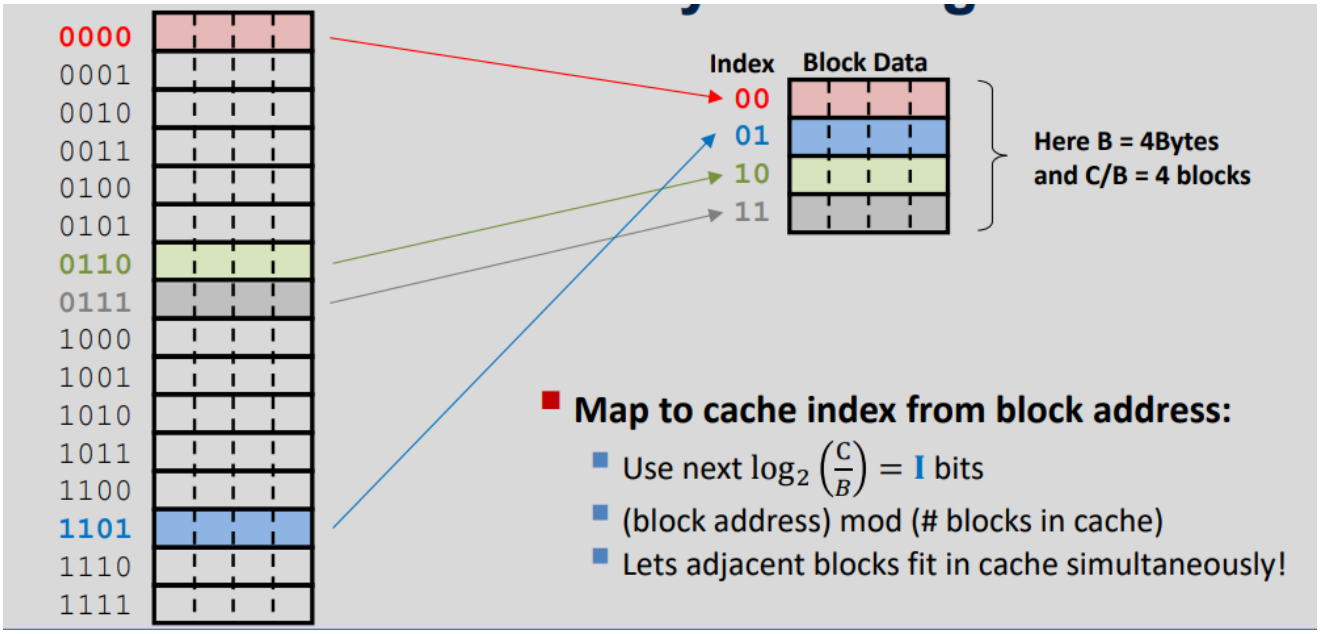
Cache Organisation (2)

- **Cache Size (C):** amount of *data* in bytes that the cache can store
  - Cache can only hold so much data (subset of next level)
  - The cache size is determine by the number of sets per the number of lines (N) in each set and the block size, thus C = S x N x B.
  - The number of blocks can be computed given the cache size and the block size (C/B)
    - Example: C = 32 KiB = 512 blocks if using 64Bytes blocks size
  - Cache organisation can be defined by the tuple (S, N, B, m)

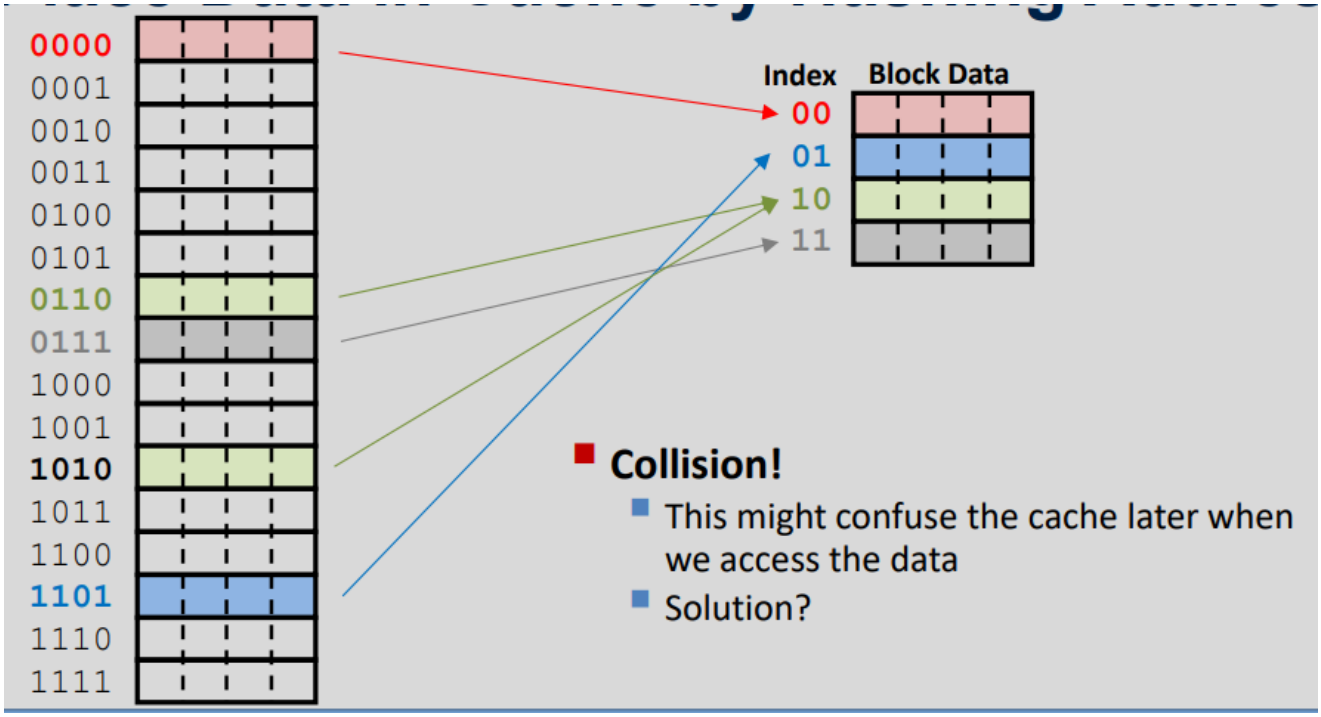
- **Where should data go in the cache?**
  - We need a mapping from memory addresses to specific locations in the cache to make checking the cache for an address **fast**.

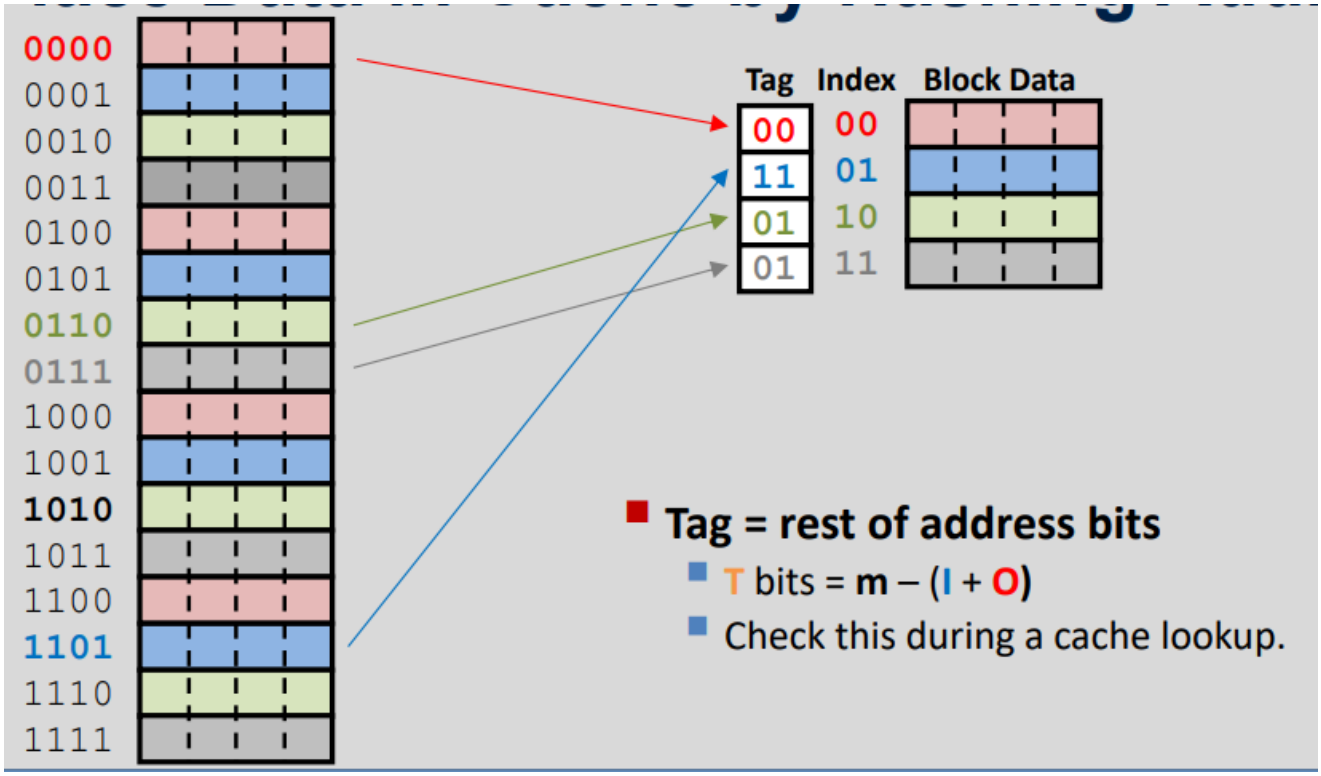
- **What is a data structure that provides fast lookup?**
  - Hash table!

Place Data in Cache by Hashing Address



- **Map to cache index from block address:**
  - Use next $\log_2\left(\frac{C}{B}\right) = \mathbf{I}$ bits
  - (block address) mod (# blocks in cache)
  - Lets adjacent blocks fit in cache simultaneously!

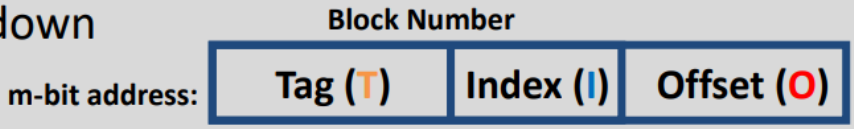




Checking for a Requested Address

- CPU sends address request for chunk of data
  - Address and requested data are not the same thing!
  - Analogy: your friend ≠ his or her phone number

- TIO address breakdown



  - **Index** field tells you where to look in cache
  - **Tag** field lets you check that data is the block you want
  - **Offset** field selects specified start byte within block

  - Note: T and I sizes will change based on hash function