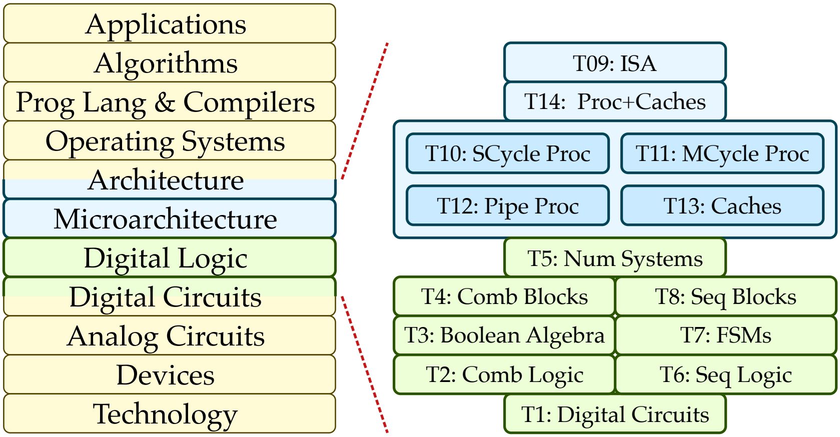
ECE 2300 Digital Logic and Computer Organization
Fall 2025

Topic 13: Caches

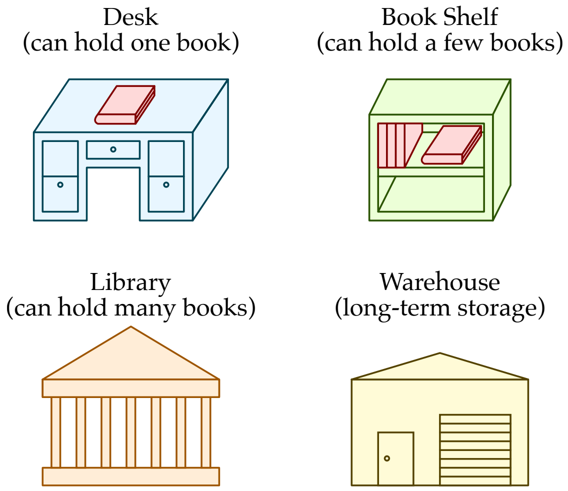
School of Electrical and Computer Engineering
Cornell University

revision: 2025-12-04-10-07


Applications
Algorithms
Prog Lang & Compilers
Operating Systems
Architecture
Microarchitecture
Digital Logic
Digital Circuits
Analog Circuits
Devices
Technology
T09: ISA
T14: Proc+Caches
T10: SCycle Proc
T11: MCycle Proc
T12: Pipe Proc
T13: Caches
T5: Num Systems
T4: Comb Blocks
T8: Seq Blocks
T3: Boolean Algebra
T7: FSMs
T2: Comb Logic
T6: Seq Logic
T1: Digital Circuits

1. Memory/Library Analogy

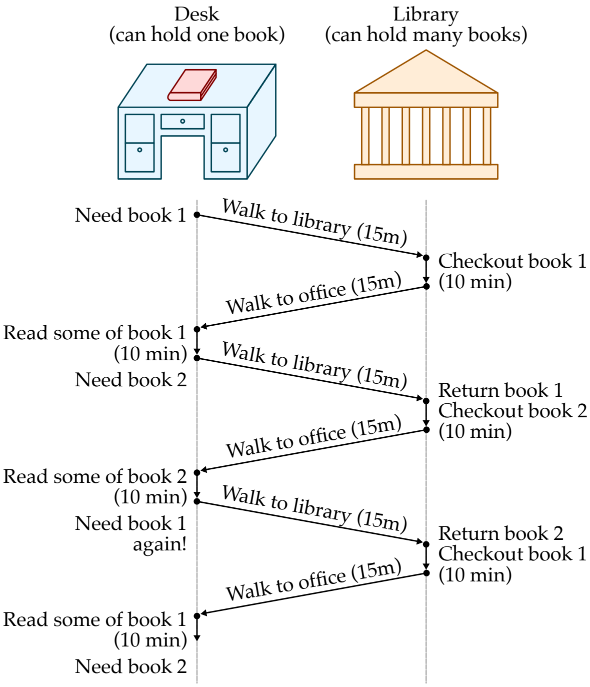
Our goal is to do some research on a new computer organization, and so we wish to consult the literature to learn more about past computer systems. The library contains most of the literature we are interested in, although some of the literature is stored off-site in a large warehouse. There are too many distractions at the library, so we prefer to do our reading in our doorm room or office. Our doorm room or office has an empty bookshelf that can hold ten books or so, and our desk can hold a single book at a time.



1.1. Three Example Scenarios

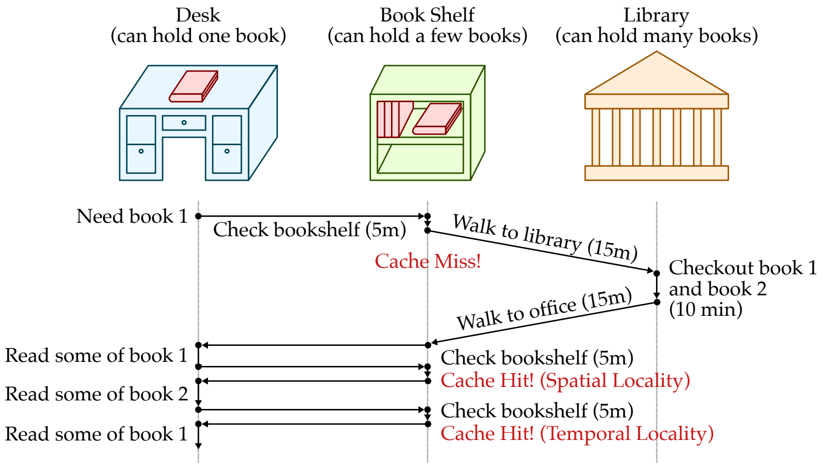
- Use desk and library
- Use desk, book shelf, and library
- Use desk, book shelf, library, and warehouse

Books from library with no bookshelf "cache"



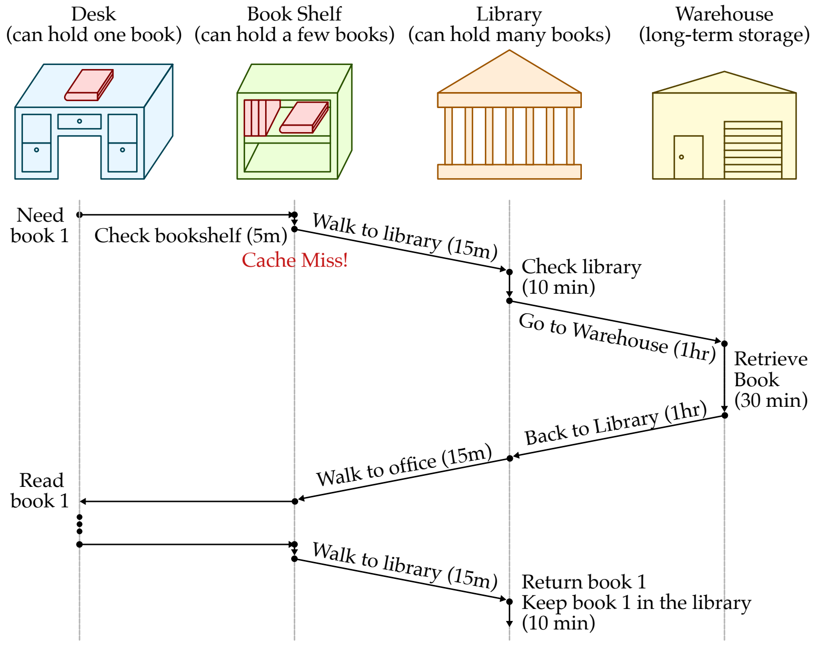
- Some inherent "translation" since we need to use the online catalog to translate a book author and title into a physical location in the library (e.g., floor, row, shelf)
- Average latency to access a book: 40 minutes
- Average throughput including reading time: 1.2 books/hour
- Latency to access library limits our throughput

Books from library with bookshelf "cache"



- Average latency to access a book: <20 minutes
- Average throughput including reading time: ≈2 books/hour
- Bookshelf acts as a small "cache" of the books in the library
  - Cache Hit: Book is on the bookshelf when we check, so there is no need to go to the library to get the book
  - Cache Miss: Book is not on the bookshelf when we check, so we need to go to the library to get the book
- Caches exploit structure in the access pattern to avoid the library access time which limits throughput
  - Temporal Locality: If we access a book once we are likely to access the same book again in the near future
  - Spatial Locality: If we access a book on a given topic we are likely to access other books on the same topic in the near future

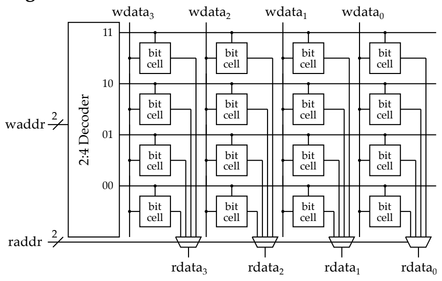
Books from warehouse



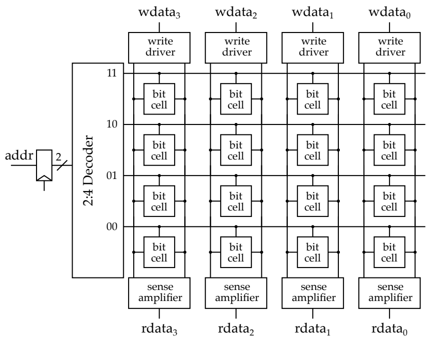
- Keep very frequently used books on book shelf, but also keep books that have recently been checked out in the library before moving them back to long-term storage in the warehouse
- We have created a "book storage hierarchy"
- Book Shelf : low latency, low capacity
- Library : high latency, high capacity
- Warehouse : very high latency, very high capacity

1.2. Review: Memory Arrays

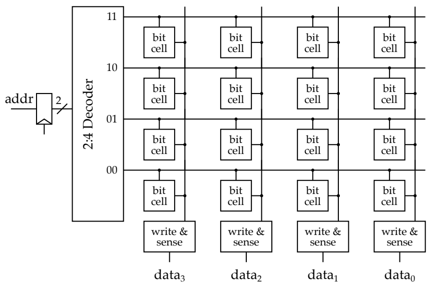
Register File



Static Random Access Memory (SRAM)



Dynamic Random Access Memory (DRAM)



Comparing different types of memory

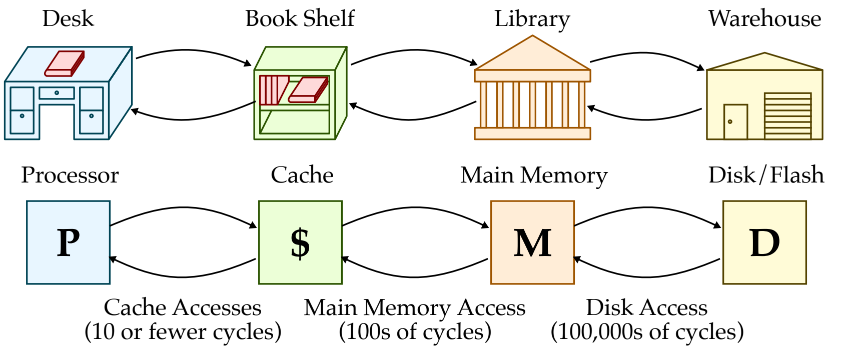
- Capacity: How many bits can we store per unit area?
- Latency: How long does it take to read or write data?
- Bandwidth: How many bits can we read or write at once?

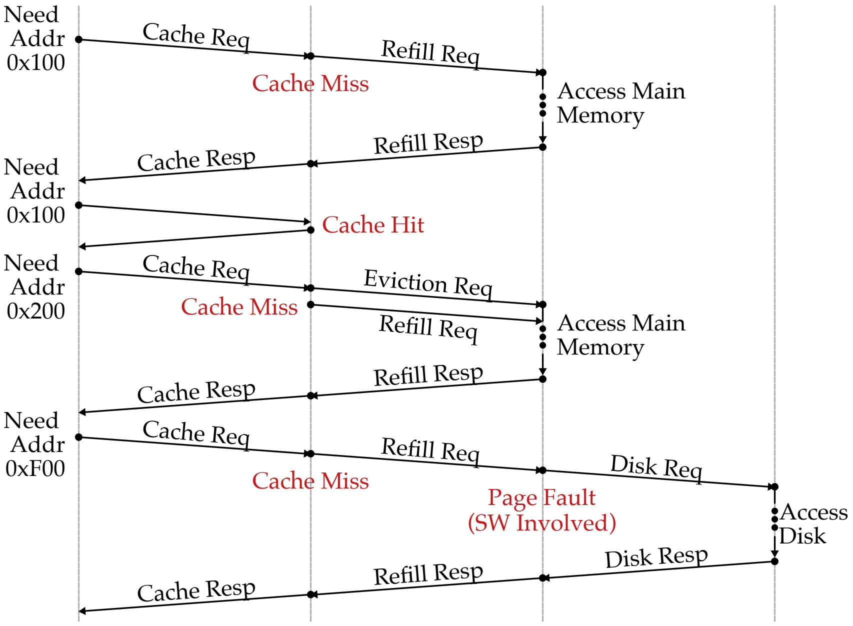
Memory Type	Capacity	Latency	Bandwidth
Register			
Register File			
SRAM			
DRAM			
SSD/Disk			

Latency numbers every computer engineer should know

Small SRAM access	1	ns
Branch mispredict	3	ns
Large SRAM access	4	ns
Mutex lock/unlock	17	ns
Send 2KB over commodity network	22	ns
DRAM access	100	ns
Compress 1KB with zip	2	us
Read 1MB sequentially from DRAM	2	us
SSD random read	16	us
Read 1MB sequentially from SSD	31	us
Round trip in datacenter	500	us
Read 1MB sequentially from disk	625	us
Disk random read	2	ms
Packet roundtrip from CA to Netherlands	150	ms

1.3. Cache Memories in Computer Architecture



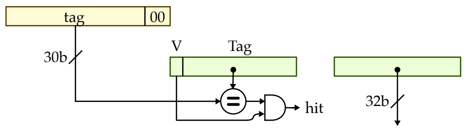


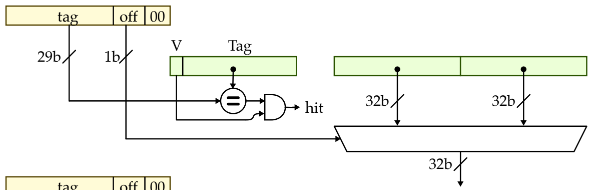
2. Cache Concepts

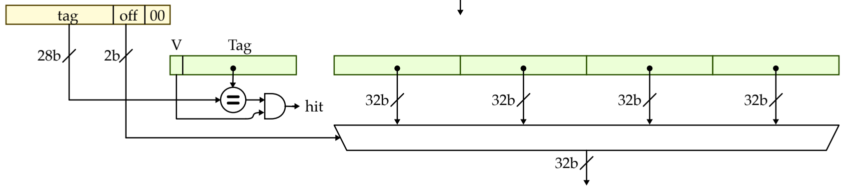
- Single-line cache
- Multi-line cache
- Replacement policies
- Write Policies

2.1. Single-Line Cache

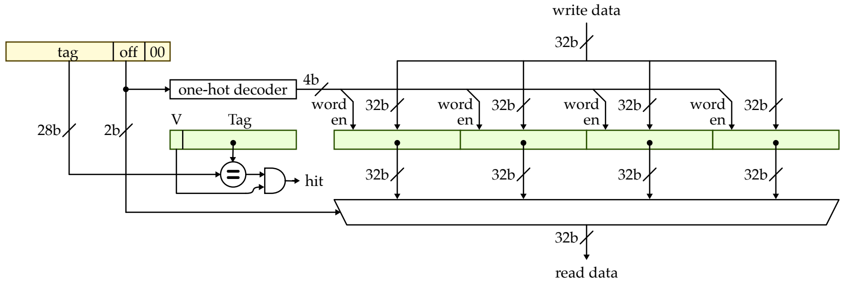
Consider only 4B word accesses and only the read path for three single-line cache designs:







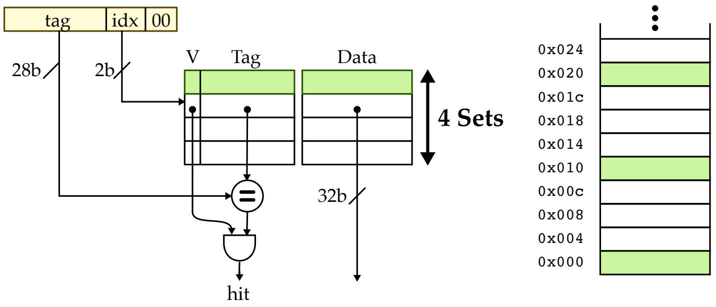
What about writes?



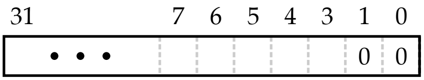
- Spatial Locality: Refill entire cache line at once
- Temporal Locality: Reuse word multiple times

2.2. Multi-Line Cache

Consider a four-line direct-mapped cache with 4B cache lines



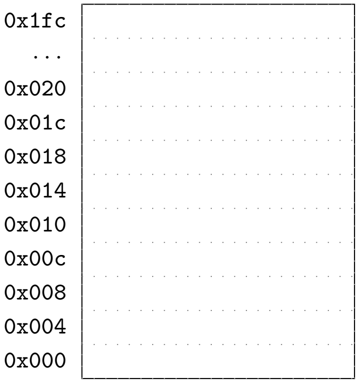
Consider a 16B direct-mapped cache with 4B cache lines.



	Tag	Idx	Hit?
☐☐☐ `rd 0x000`			
☐☐☐ `rd 0x004`			
☐☐☐ `rd 0x010`			
☐☐☐ `rd 0x000`			
☐☐☐ `rd 0x004`			
☐☐☐ `rd 0x020`			

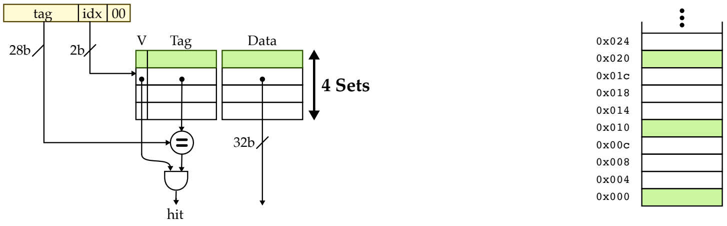
	V	Tag	Data
Set 0			
Set 1			
Set 2			
Set 3			

Memory

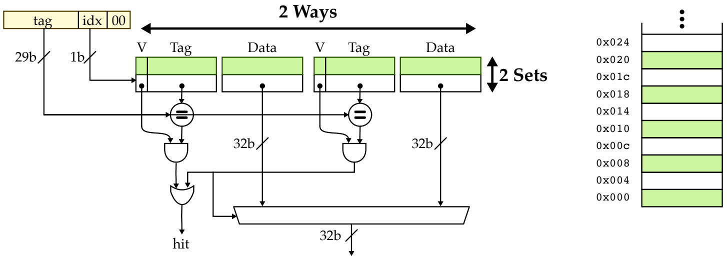


Increasing cache associativity

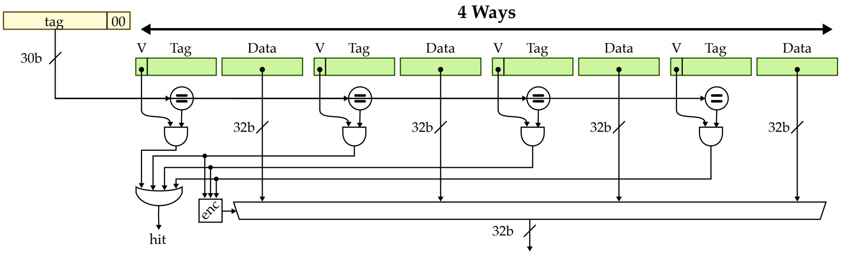
Four-line direct-mapped cache with 4B cache lines



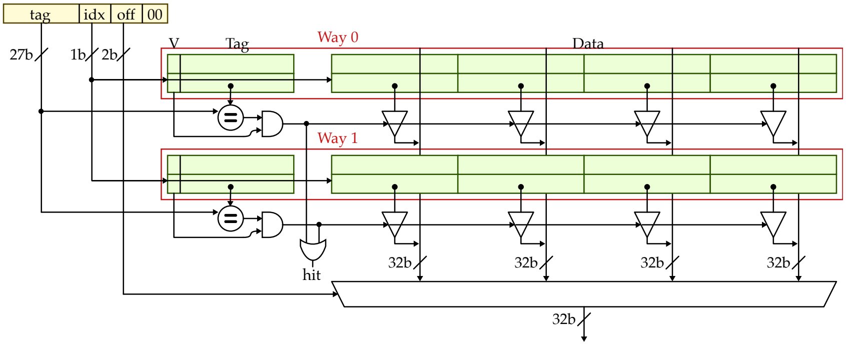
Four-line two-way set-associative cache with 4B cache lines



Four-line fully-associative cache with 4B cache lines



Combining associativity with longer cache lines

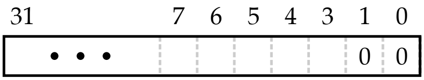


- Spatial Locality: Refill entire cache line + simple indexing to find set
- Temporal Locality: Reuse word multiple times + replacement policy

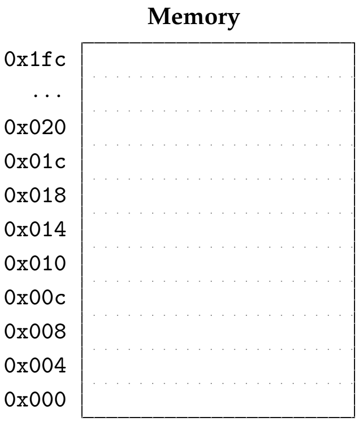
2.3. Replacement Policies

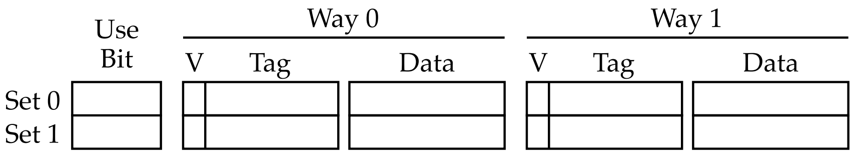
- No choice in a direct-mapped cache
- Random
  - Good average case performance, but difficult to implement
- Least Recently Used (LRU)
  - Replace cache line which has not been accessed recently
  - LRU cache state must be updated on every access which is expensive
  - True implementation only feasible for small sets
  - Two-way cache can use a single “last used bit”
  - Pseudo-LRU uses binary tree to approximate LRU for higher associativity
- First-In First-Out (FIFO, Round Robin)
  - Simpler implementation, but does not exploit temporal locality
  - Potentially useful in large fully associative caches

Consider a 16B 2-way set-associative cache with 4B cache lines.



	Tag	Idx	Hit?
□ □ □ `rd 0x000`			
□ □ □ `rd 0x004`			
□ □ □ `rd 0x010`			
□ □ □ `rd 0x000`			
□ □ □ `rd 0x004`			
□ □ □ `rd 0x020`			





2.4. Write Policies

Write-Through with No Write Allocate

- On write miss, write memory but do not bring line into cache
- On write hit, write both cache and memory
- Requires more memory bandwidth, but simpler to implement

Write-Back with Write Allocate

- On write miss, bring cache line into cache then write
- On write hit, only write cache, do not write memory
- Only update memory when a dirty cache line is evicted
- More efficient, but more complicated to implement

3. Analyzing Memory Performance

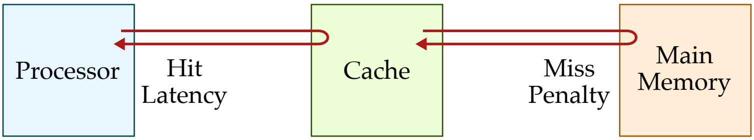
$$\frac{\text{Time}}{\text{Mem Access Sequence}} = \frac{\text{Mem Accesses}}{\text{Sequence}} \times \frac{\text{Avg Cycles}}{\text{Mem Access}} \times \frac{\text{Time}}{\text{Cycle}}$$

$$\frac{\text{Avg Cycles}}{\text{Mem Access}} = \frac{\text{Avg Cycles}}{\text{Hit}} + \left(\frac{\text{Num Misses}}{\text{Num Accesses}} \times \frac{\text{Avg Extra Cycles}}{\text{Miss}} \right)$$

- Mem access / sequence depends on program and translation
- Time / cycle depends on microarchitecture and implementation

- Also called the average memory access latency (AMAL)
- Avg cycles / hit is called the hit latency
- Number of misses / number of accesses is called the miss rate
- Avg extra cycles / miss is called the miss penalty

- Avg cycles per hit depends on microarchitecture
- Miss rate depends on microarchitecture
- Miss penalty depends on microarchitecture, rest of memory system



Estimating average memory access latency

Consider the following sequence of memory acceses which might correspond to copying 4 B elements from a source array to a destination array. Each array contains 64 elements. Assume two-way set associative cache with 16 B cache lines, hit latency of 1 cycle and 10 cycle miss penalty. What is the AMAL in cycles?

```
rd 0x1000
wr 0x2000
rd 0x1004
wr 0x2004
rd 0x1008
wr 0x2008
...
rd 0x1040
wr 0x2040
```

Consider the following sequence of memory acceses which might correspond to incrementing 4 B elements in an array. The array contains 64 elements. Assume two-way set associative cache with 16 B cache lines, hit latency of 1 cycle and 10 cycle miss penalty. What is the AMAL in cycles?

```
rd 0x1000
wr 0x1000
rd 0x1004
wr 0x1004
rd 0x1008
wr 0x1008
...
rd 0x1040
wr 0x1040
```