Memory Organization (part 1)

Chapter 8

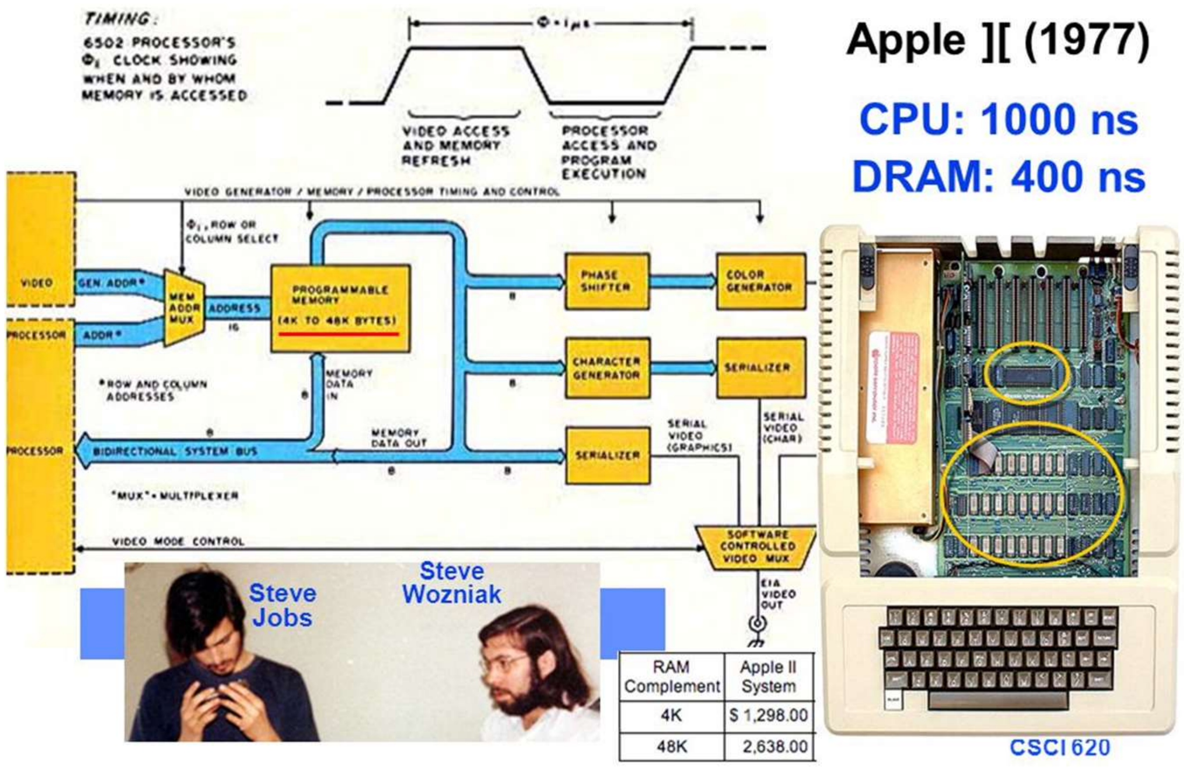
Memory Organization (part 1)

- ★ Technology Trends and Memory Issues
- ★ Design Goal
- ★ Principle of Locality
  - ○ Temporal, Spatial
- ★ Cache: Terminology and Organization
- ★ Performance
  - ○ Sources of Miss
- ★ Configuration
  - ○ Cache Size, Block Size, Miss Penalty
- ★ Write Management
  - ○ Write Buffer
- ★ Software and Cache

Technology Trends and Memory Issues

- ★ In 70s, memory is faster than a processor.
- ★ In the text mode/dos era, 640KB was more than enough.
- ★ Now, minimum requirement is 4-8 GB.

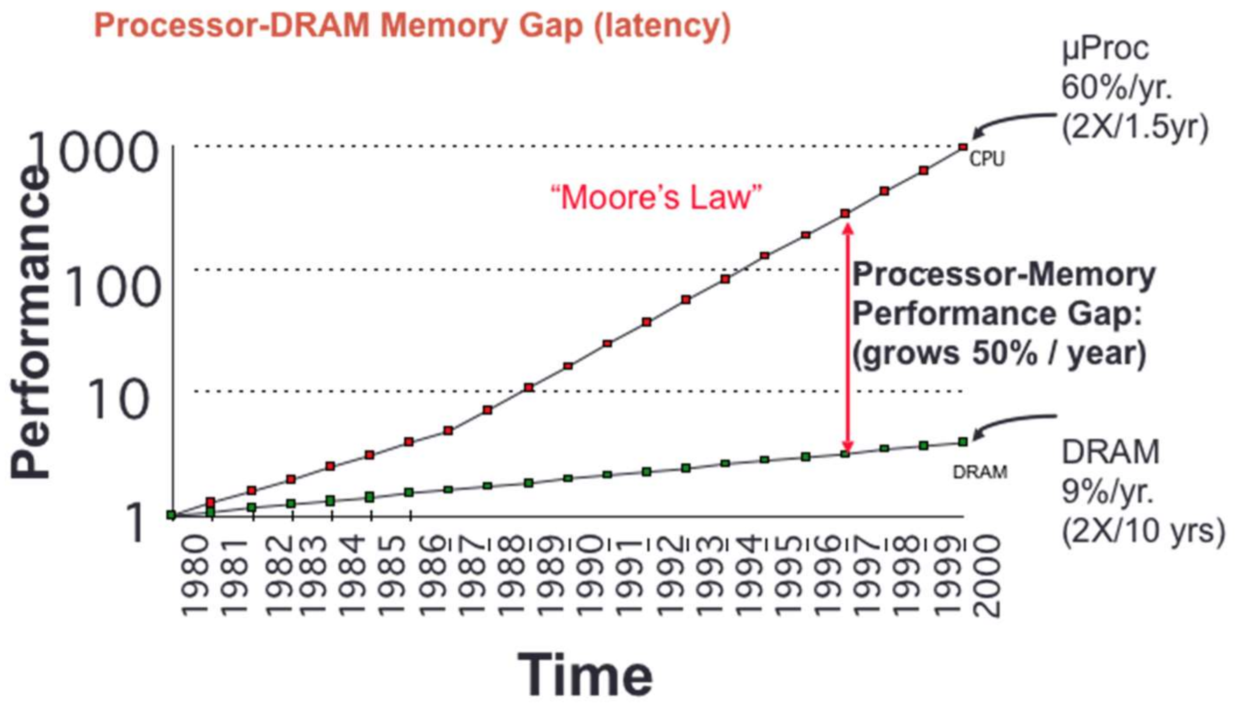
1977: At one time, DRAM was faster than microprocessors 7



RAM Complement	Apple II System
4K	$ 1,298.00
48K	2,638.00

Technology Trends and Memory Issues

- ★ Now, memory is relatively slow, comparing to processor.
- ★ In 1980, CPU has no cache.
- ★ In 1995, CPU has 2-level caches.



Issues on Performance

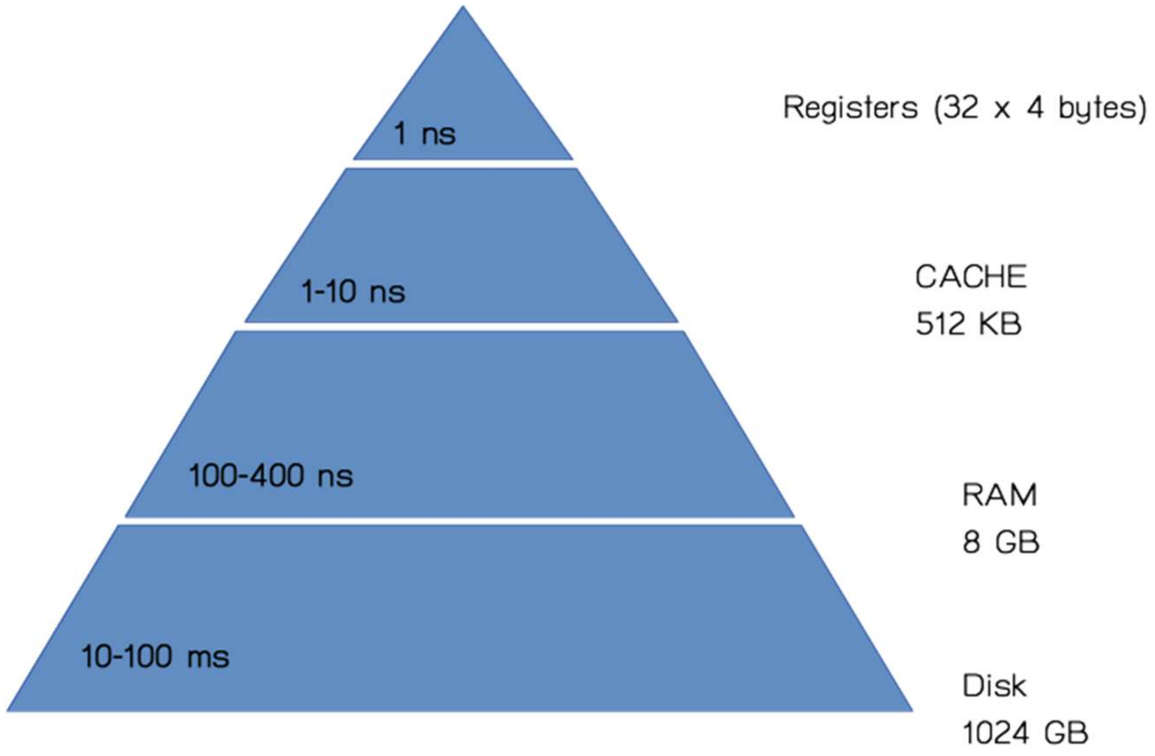
- ★ Assuming that a memory operation takes 50 cycles and other operations take 1 cycle (with pipeline). With 10% of memory operation, what is the percentage of time that a processor has to wait for memory operations?
- ★ CPU Time = IC x CPI x Tc
- ★ CPU Time = ((0.9 IC x 1) + (0.1 IC x 50)) x Tc
- ★ Memory Wait = (0.1 IC x 50) x Tc
- ★ Wait time = (0.1 IC x 50) / ((0.9 IC x 1) + (0.1 IC x 50)) = 5 / 5.9 ≅ 85%

Design Goals

- ★ To have large, fast, (and cheap) memory for software
- ★ How do we create a memory that is large, cheap and fast (most of the time)?
  - Hierarchy
  - Parallelism
- ★ Keep in mind that large memory is slow. (Physical rules)

Memory Hierarchy

★ We have levels of memory.
★ Fewer fast memory and larger cheap memory.
★ Locality may make memory access faster.
★ Lower level is closer to the processor.



Management of Hierarchy

- ★ Registers ↔ Memory
  - ○ By software (compiler, programmer?)
- ★ Cache ↔ Memory
  - ○ By hardware
- ★ Memory ↔ Disks
  - ○ By hardware with support from operating system (virtual memory/paging)
  - ○ By the programmer (via files)

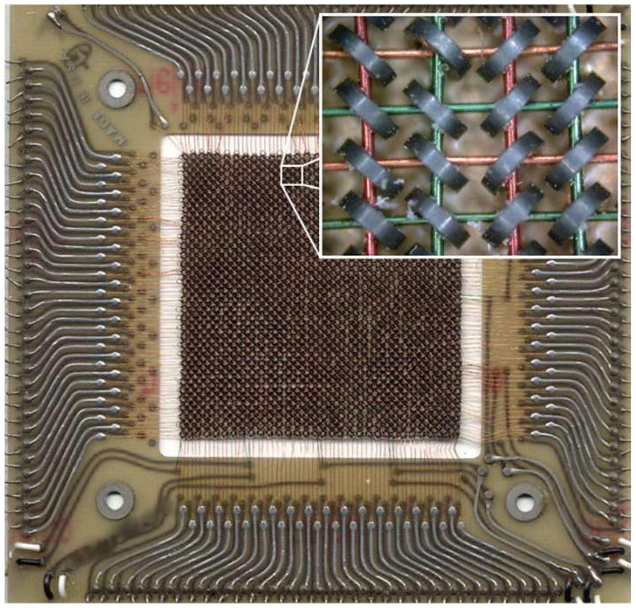
Memory Technology

★ Random Access Memory

★ Random means constant time for all locations.

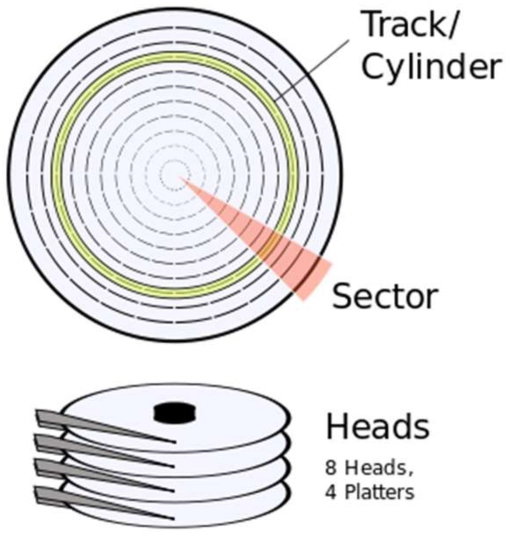
★ DRAM: Dynamic Random Access Memory
  - Capacitor Technology
  - High density, low power, cheap, slow
  - Need to be refreshed regularly.

★ SRAM: Static Random Access Memory
  - Transistor Technology
  - Low density, high power, expensive, fast
  - Content will last “forever” (while power on).



Memory Technology (ctd.)

- ★ Semi-random (Not-so-random) Access Technology
  - Seek, Rotate
  - Disk CDROM, DVDROM
- ★ Sequential Access Technology
  - Access time is linear to location
  - Tape



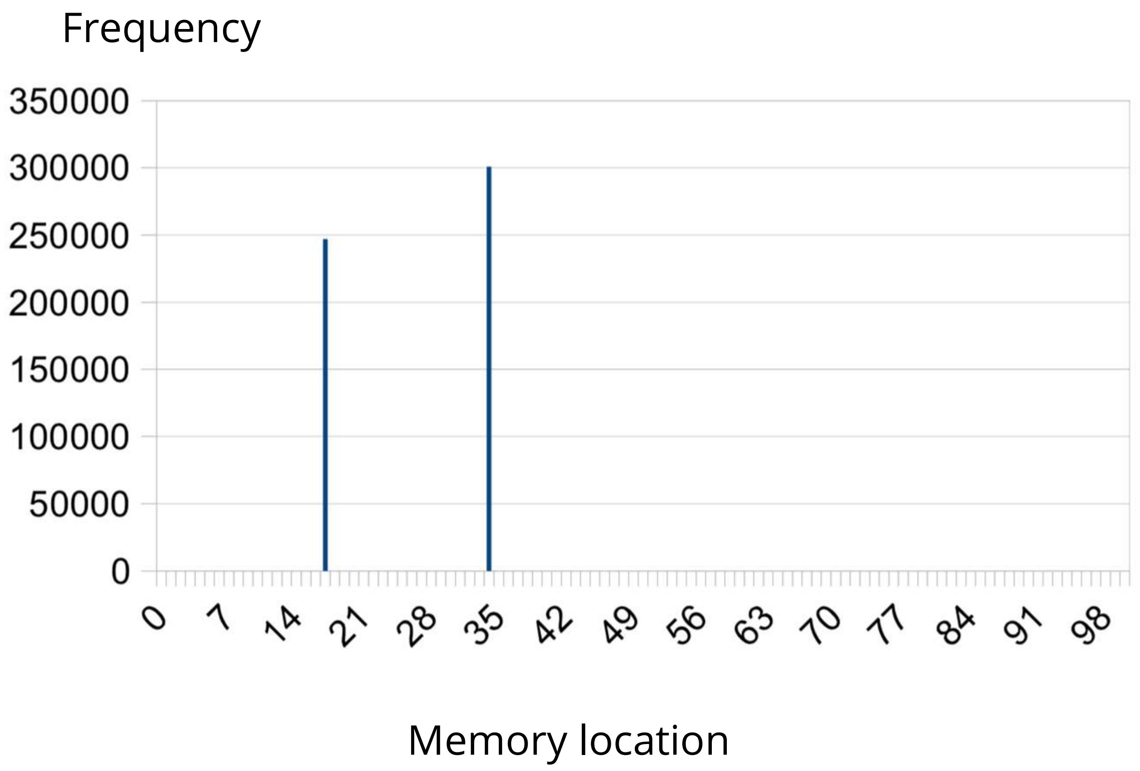
Picture from wikipedia

CACHE & Memory

- ★ DRAM is slow, but cheap and dense.
  - ○ Good choice for memory
- ★ SRAM is fast, but expensive and not very dense.
  - ○ Good choice for CACHE

Principle of Locality

- ★ At a particular time, software only access a small chunk of addresses.
- ★ If we can put this chunk in the lower level, we will feel that it is faster.
- ★ Two kinds of locality
  - Temporal
  - Spacial



Locality in real world

- ★ Temporal Locality- tempo is "time" in Italian.
- ★ Spatial Locality - Spatial means space.
- ★ If I just called you, chance is that I might call you again soon. Your number should be in a recent list (on the top). --- Temporal
- ★ If I just called you, chance is that I might want to call your friend as well. (Maybe you want to hit on a friend of your friend by using your friend as a bridge to her friend.)
  I should pick up your number and a number of your friend at the same time. --- Spatial
- ★ I always use pencil and rubber together. So I should carry them both in my pocket. -- Spatial

Cache: Terminology and Organization

- ★ Hit - when we find data in a Cache
    - ○ Hit Rate (aka. Hit Ratio) - Probability that you will find data in a Cache
    - ○ Hit Time - Time to access data in a Cache
- ★ Miss - when we cannot find data in a Cache
    - ○ Miss Rate (aka. Miss Ratio) - Probability that you will not find data in a Cache
      Miss Rate = (1-Hit Rate)
    - ○ Miss Penalty - Time for fetching data into a Cache
- ★ Block
    - ○ A small chunk of data in a line of Cache.
    - ○ We always take a whole block in/out of Cache.

Cache Organization

★ With memory hierarchy, there are 4 questions to address.
  - Q1. Where can a block be placed in the upper level?
  - Q2. How is a block found if it is in the upper level?
  - Q3. Which block should be replaced on a miss?
  - Q4. What happens on a write?

★ Cache organization will address these issues.

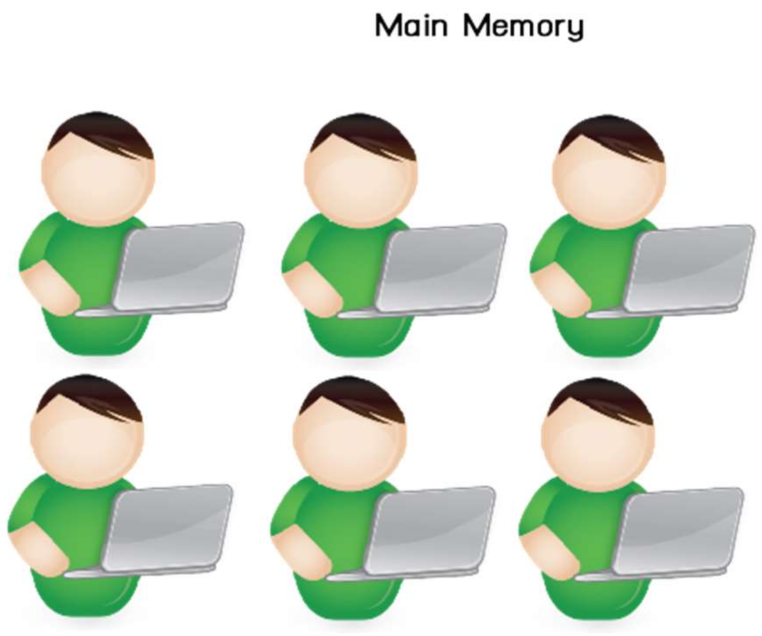
★ We will start with the most simple one.
  - Directed Mapped Cache

Directed Mapped Cache in real life

★ Suppose that a class of 100 students are in the auditorium (main memory). Each student got a seat number (e.g. id)

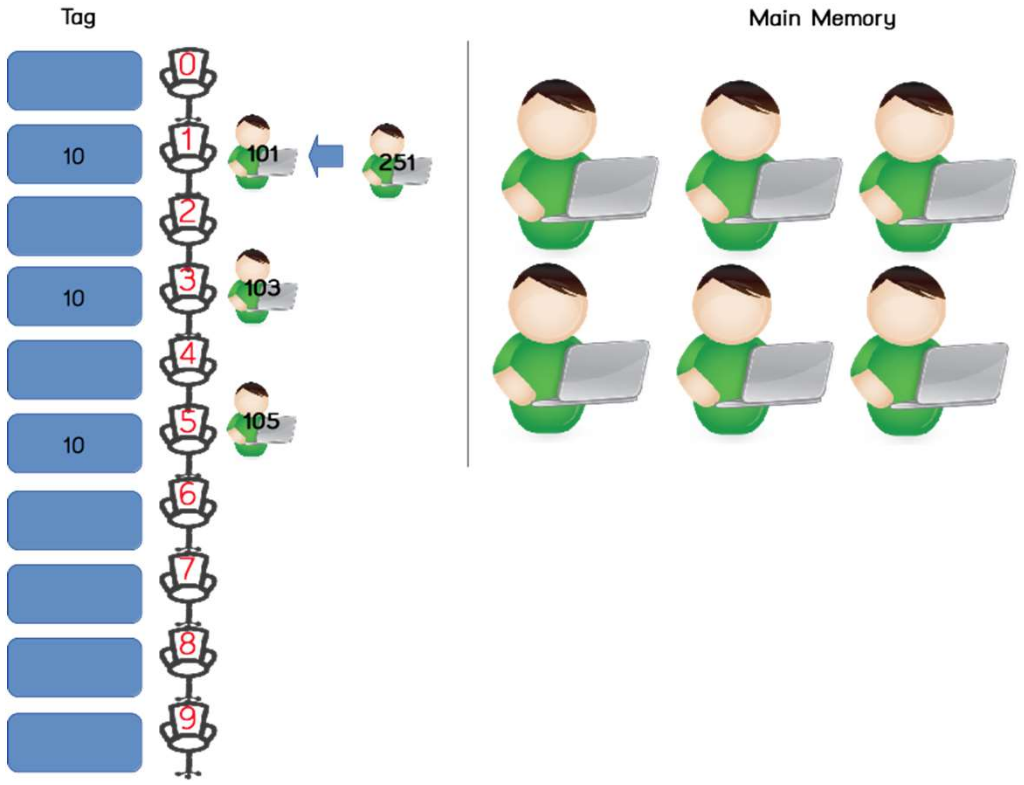
★ To work (e.g. exam), a student must first get into the office of a professor.

★ To avoid having students walked between the auditorium and the office, ten chairs are installed in the office.

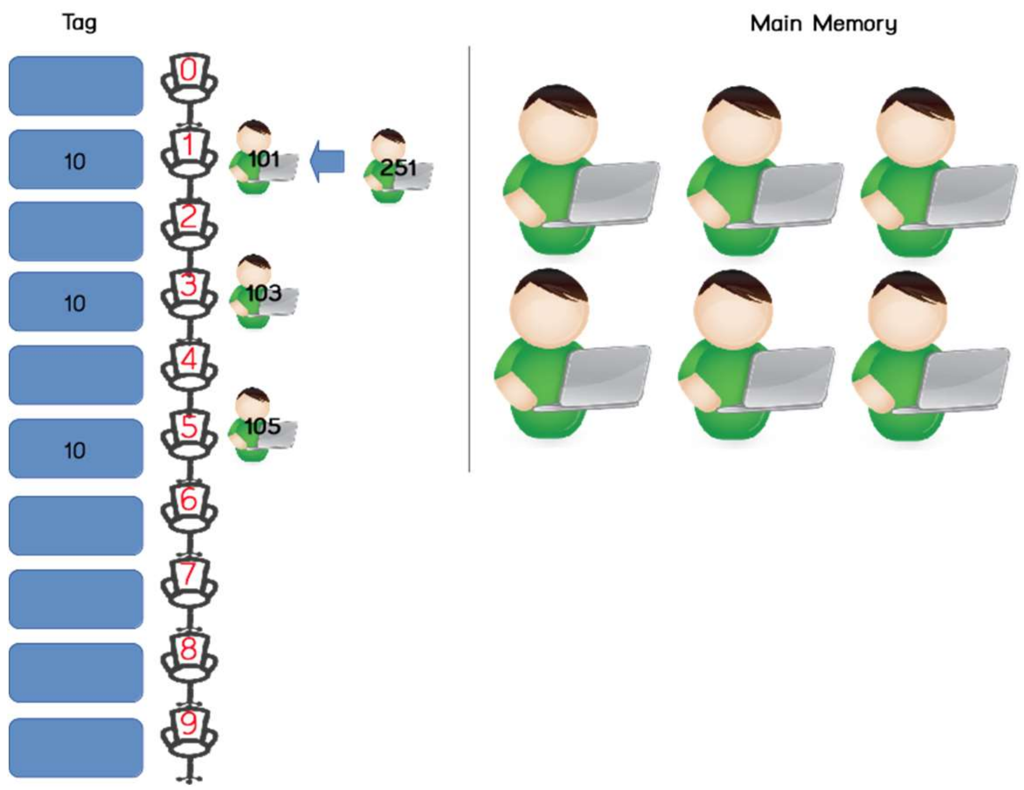


Directed Mapped Cache in real life (ctd)

- ★ A directed mapped cache means there is a place in locality for each person.
- ★ Given that each person has an address (id), we can use a digit to **index** a chair.
- ★ To figure out who is sitting at a particular chair, we put a **tag** at each seat.



Tag and Index

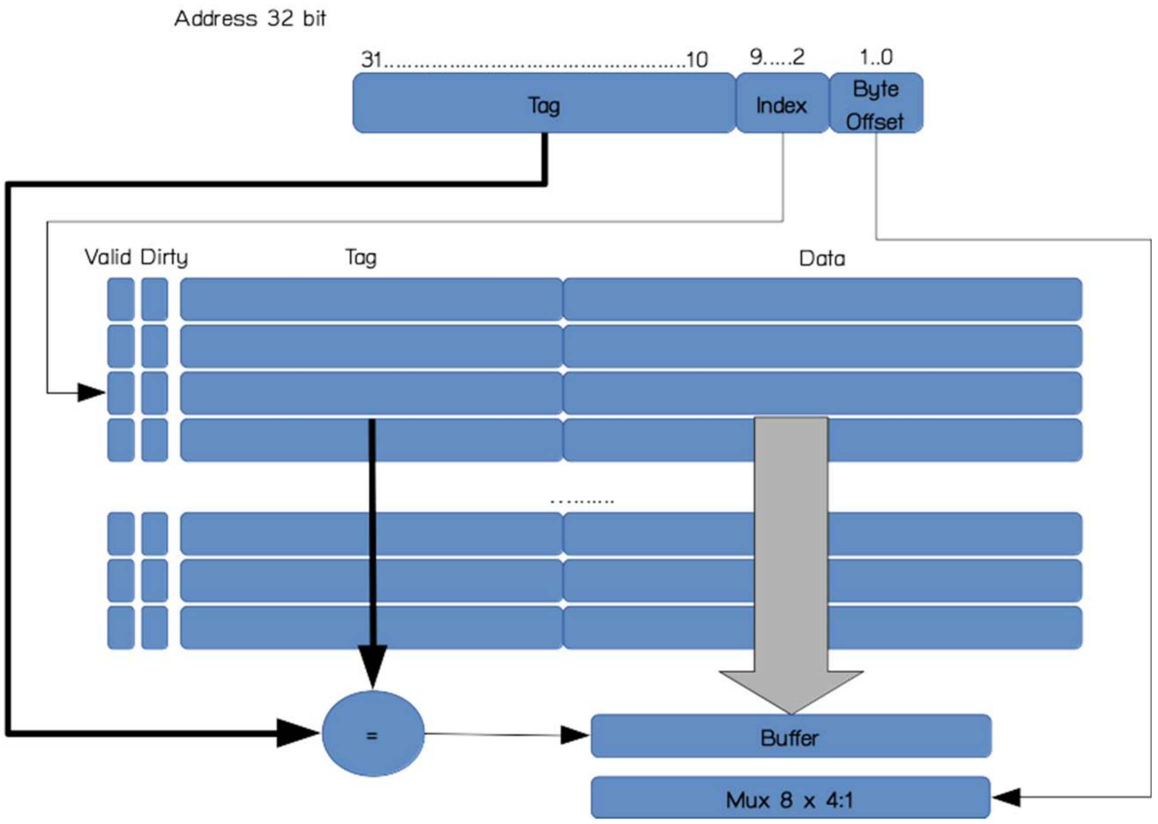


- ★ Index = a place to sit
  - = (address mod cache size) div block size.
- ★ Tag = an id of a person
  - = address div cache size.
- ★ With 10x1 chairs (10 rows, each with 1 chair) means cache size=10 and block size=1.
- ★ A person with id 105, will have to sit at seat index 5 (105 mod 10) and put up a tag 10 (105 div cache size).

Directed Mapped Cache in Memory

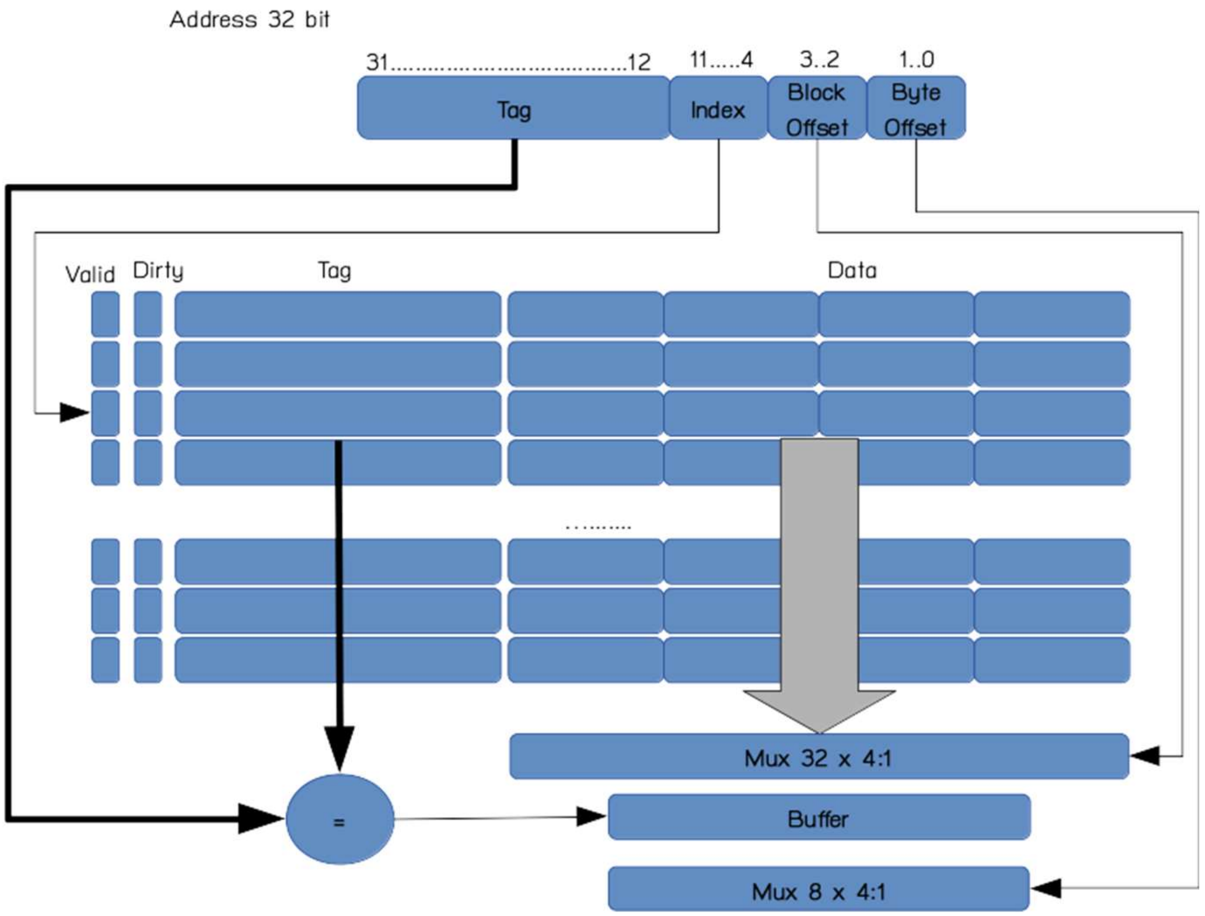
- Find the following values
  - Block size
  - Cache size
- What is
  - Valid bit?
  - Dirty bit?
- Please locate 0x00004098 in this cache.

This is good for temporal.
Can we add more spatial?



Directed Mapped Cache with Spatial

- ★ A larger block size means more spatial locality.
- ★ Find the following values
  - ○ Block size
  - ○ Cache size



Performance (revisit)

- ★ Memory Access Time = Hit Time + (Miss Rate x Miss Penalty)
- ★ Hit Time and Miss Penalty usually report in cycle.
- ★ CPI = ideal CPI + (Miss Rate x Miss Penalty)
- ★ CPU Time = ??
- ★ Note. We previously assume that memory access time is 1 cycle.

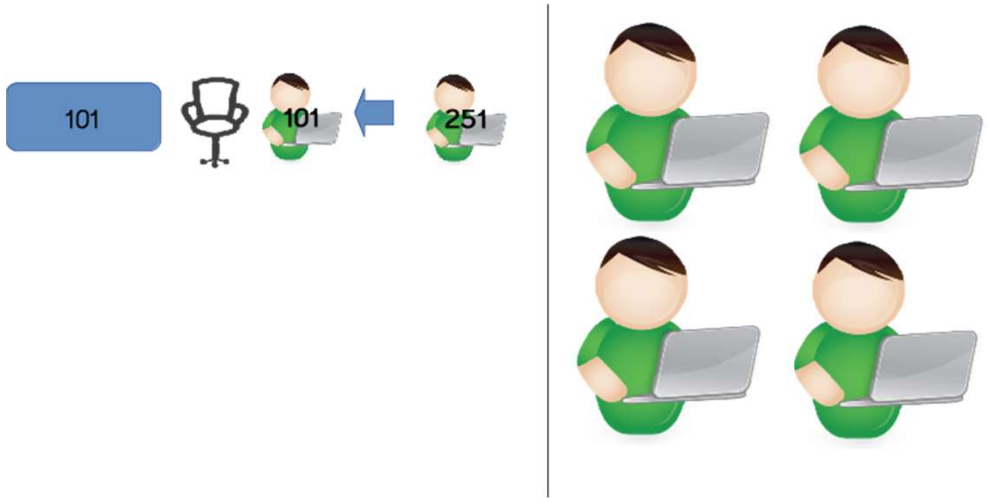
Optimizing Cache

★ To get the best performance, we have to minimize memory access time.
★ Since Memory Access Time = Hit Time + (Miss Rate x Miss Penalty), How can we minimize:
  ○ Hit Time
  ○ Miss Rate
  ○ Miss Penalty
★ To minimize Hit Time, use small cache. Why?
★ How about Miss Rate and Miss Penalty?

Sources of Miss

- ★ To minimize Miss Rate, we have to first understand sources of miss.
- ★ 3 Sources of miss
- ★ Compulsory Miss (aka. Cold Miss)
  - ○ Cold start. No data in cache. We can do nothing.
  - ○ Maybe a prefetch can help. (Beyond the scope of this class.)
- ★ Capacity Miss
  - ○ The cache size (locality) is too small for the workload.
  - ○ See ping-pong effect in the next slide.
  - ○ Naive solution is to increase the cache size.
- ★ Conflict Miss
  - ○ There is a free space in a Cache. However, the mapping mechanism prevents us from using them.
  - ○ Solutions: (1) Increase cache size, (2) multiple entries for the same index.

Ping-Pong effect (extreme capacity miss)



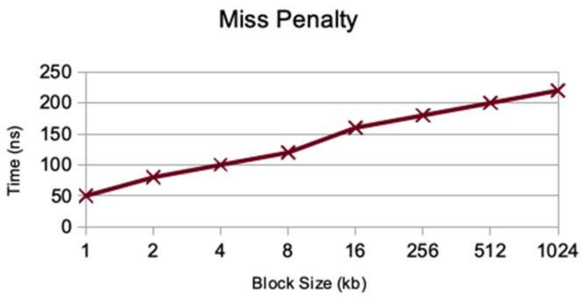
- ★ If there is only one line of cache, an access to a new address always result in a miss.
- ★ Assuming that we can only put a page of paper on a desk, every page changes will result in a miss.
- ★ The CPU is always busy swapping memory in and out.

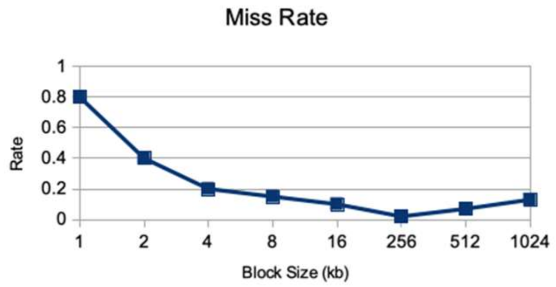
Cache Configurations

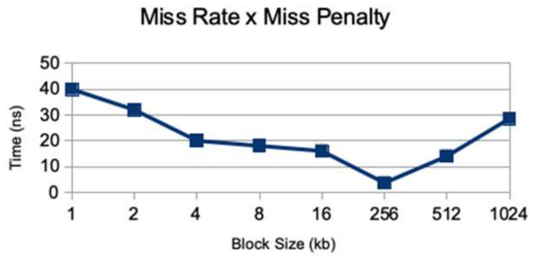
- ★ 5 Configurations
  - ○ Cache Size
  - ○ Block Size
  - ○ Associativity
  - ○ Replacement Algorithm
  - ○ Write Management (Write Buffer)
- ★ Let’s optimize Miss Rate

Block Size

- ★ Intuitively, larger block size means more spatial locality.
- ★ However, larger block also means higher miss penalty
- ★ Too large a block also means less temporal locality (when cache size is fixed.)
- ★ Finding an optimal block size requires a lots of benchmarking.
- ★ See assignment II.

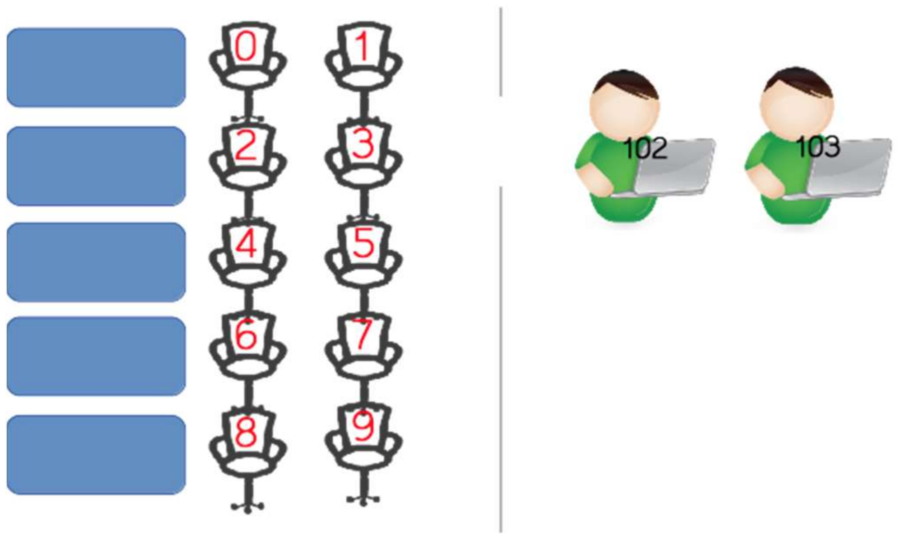




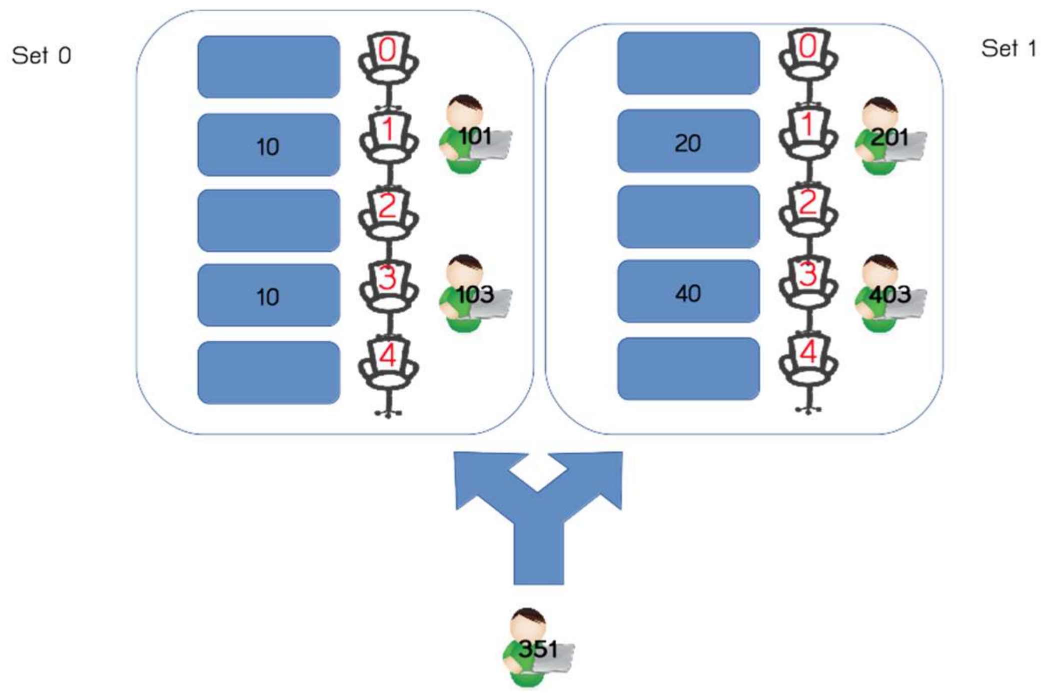


Block Size (ctd.)

★ Unless increasing a bus size, larger block means higher miss penalty. (Bringing several person into a room with small door.)



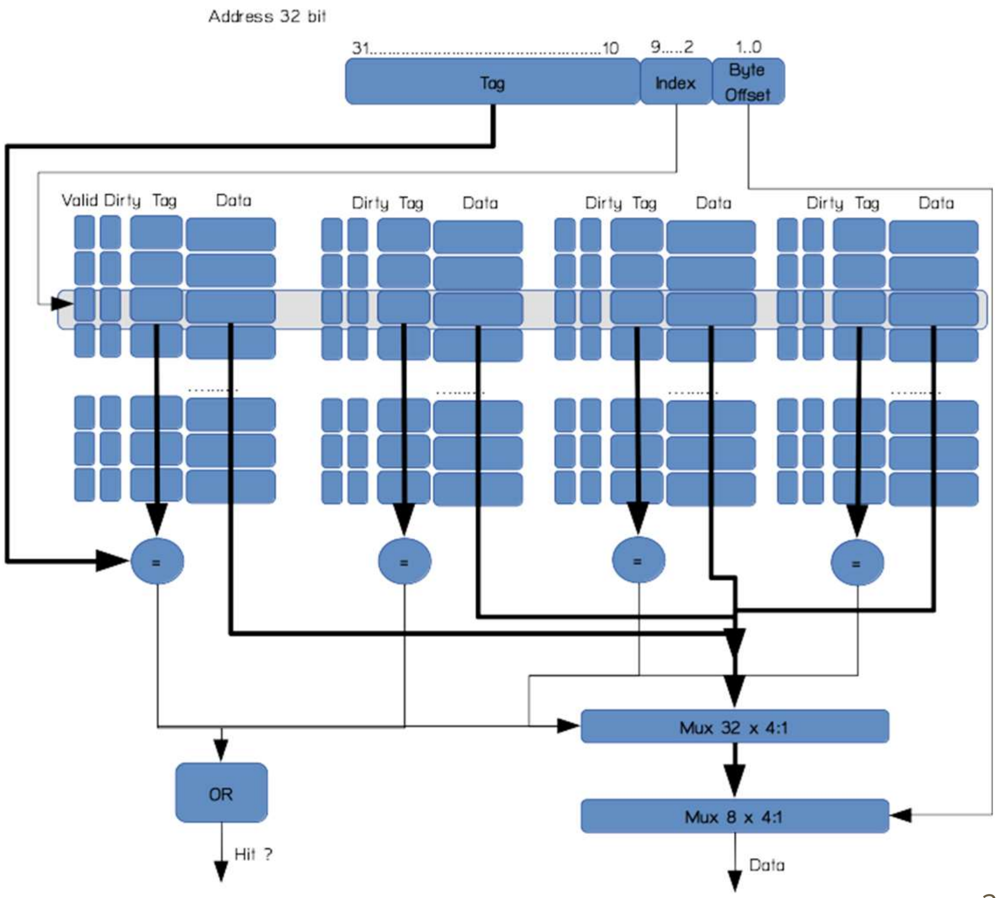
Two-way set Associativity



- ★ Two sets of chairs in an office.
- ★ Two entries for the same index.
- ★ Less conflict miss.
- ★ Can also increase to four ways, and eight ways.

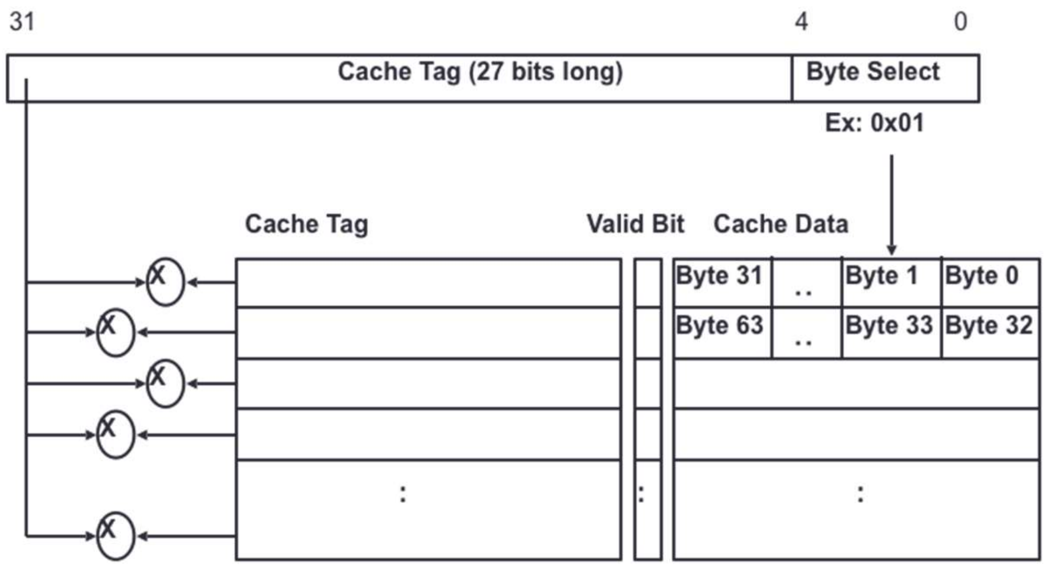
4-way set Associativity Cache

★ More complex circuit for accessing data
★ 4 comparators
★ Large multiplexer
★ Higher Hit Time? (Slower?)



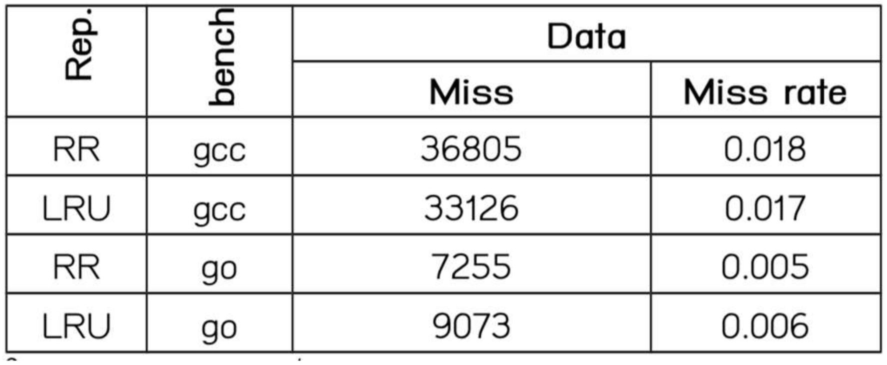
Fully Associative (zero conflict miss)

- ★ Another extreme
- ★ No index. Data can be placed anywhere.
- ★ How many comparators do we need?
- ★ Conflict Miss = 0



Replacement Algorithm

- ★ Least Recently Used (LRU)
  - ○ Discards the least recently used items first
- ★ Round Robin
  - ○ Taking turn for replacement.
- ★ How about First In First Out (FIFO), and Random Replacement?
- ★ Practically, code and data use split caches. There is more spatial locality in code.



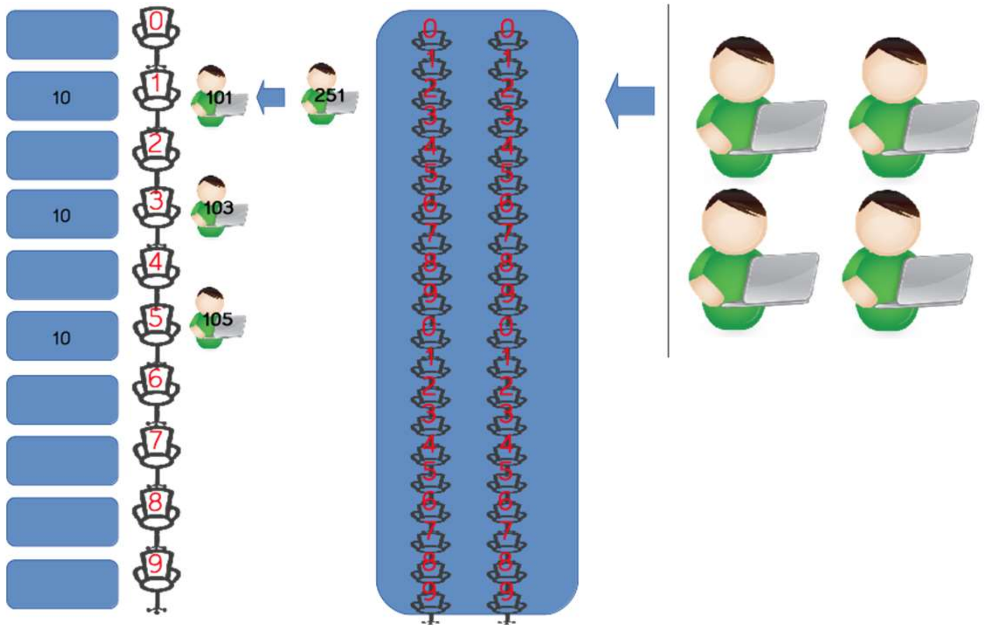
Rep.	bench	Data	
		Miss	Miss rate
RR	gcc	36805	0.018
LRU	gcc	33126	0.017
RR	go	7255	0.005
LRU	go	9073	0.006

Pop Quiz

With directed map, what kind of replacement algorithm should we use?
Please provide your analysis.

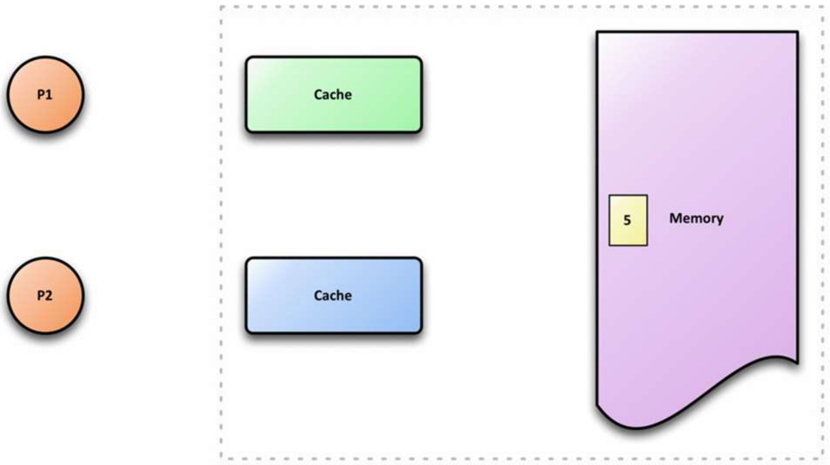
Reducing Miss Penalty

- ★ Between the professor's office and auditorium, we can install chairs in a hall.
- ★ A miss from an office may hit in the hall.
- ★ Less miss penalty comparing to going to the auditorium.



Write Management

- ★ Two choices
  - ○ Write Back
  - ○ Write Through
- ★ Write Back
  - ○ Write to Cache only. Will update main memory on replacement.
  - ○ Good for repeated write
  - ○ Fast, but with incoherent data in multiprocessor system.
- ★ Write Through
  - ○ Always write to cache and main memory
  - ○ Bad for repeated write
  - ○ Slow, but preferred for multiprocessor system



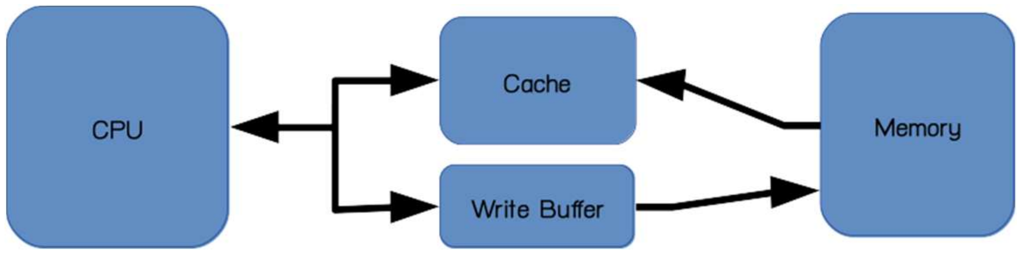
Write Management (ctd.)

- ★ Write allocate vs. Write not allocate
- ★ Write not allocate means write directly without first loading.
- ★ On write miss, write allocate means loading data before write (equivalent to miss penalty + write time).
  - How about writing to a part of the block?
  - Use subblock concept? (beyond the scope of this class)

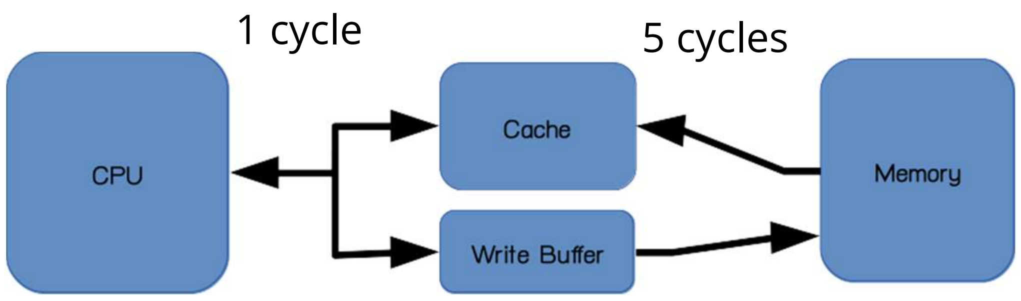
- ★ For simplicity, we will use write through and write allocate.

Write Buffer (for Write Through)

★ With write through, a write access is bad for performance. (equivalent to miss penalty?)
★ To make write faster, use write buffer.
★ Write Buffer
★ Since write through is like running to auditorium on each write. A write buffer is having someone to run to the auditorium to tell the main person for you.



Write Buffer Saturation

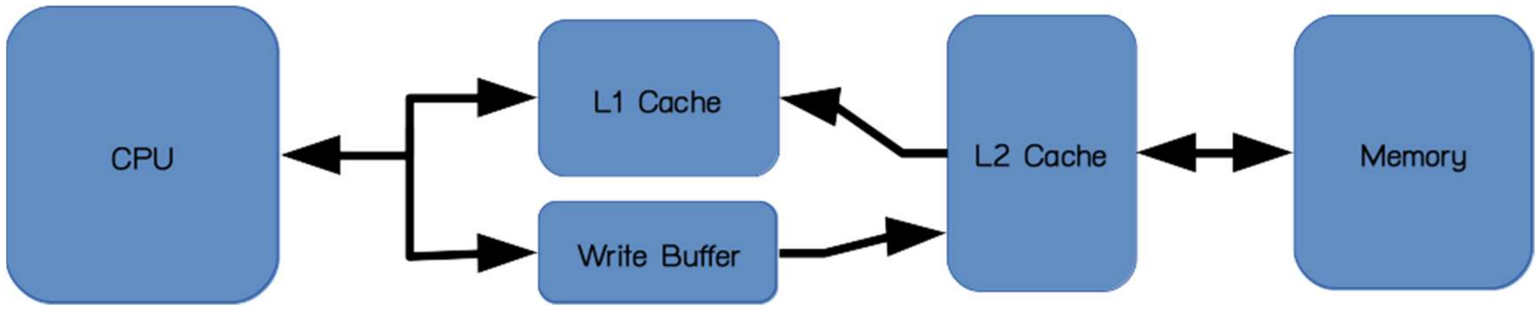


- ★ Write Buffer is basically a small FIFO (eg. 4 words)
- ★ If memory is much more slower than main memory, write buffer will soon be saturated (overflowed) regardless the size of write buffer.
  (Nobody to run to the auditorium for the professor.)
- ★ How to solve this problem?

Write buffer saturation (ctd.)

★ A naive solution is to introduce a second level cache (L2).

★ Analogy: Rather than running to the auditorium, a write means running to the hall. (less saturation)



Software and Cache

- ★ Cache Management is transparent to software.
  (We gain benefits from cache without any software modification.)
- ★ As a programmer, can we programmatically take advantage of Cache?
- ★ Can a program exploit more temporal and more spatial?
  - ○ Continuously use variables in the scope (more temporal).
- ★ Can a program avoid conflict miss?

More Spatial

★ A software can programmatically take advantage of spatial locality.

★ These two snippets are logically equivalent.

★ However, one is faster Why?

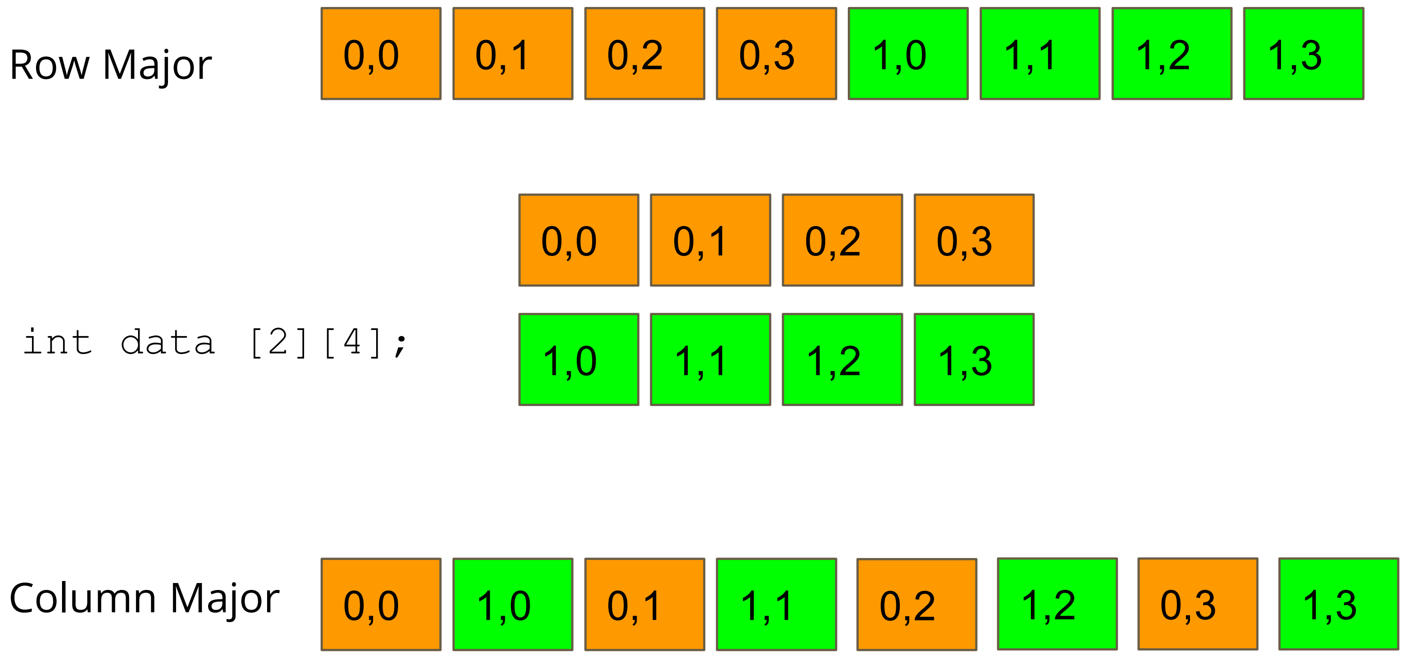
```
// row-major order
int data [1000][1000];
int sum;
for (int r=0; r<1000; r++){
  for (int c=0; c<1000; c++){
      sum += data[r][c];
  }
}
```

```
real    0m0.010s
user    0m0.004s
sys     0m0.004s
```

```
// column-major order
int data [1000][1000];
int sum;
for (int c=0; c<100; c++){
  for (int r=0; r<100; r++){
      sum += data[r][c];
  }
}
```

```
real    0m0.013s
user    0m0.007s
sys     0m0.004s
```

Row-Major Order vs. Column-Major Order



```
int data [2][4];
```

Reduce Conflict Miss

★ Likely, accessing different arrays with same index may result in a conflict miss. (same cache index)

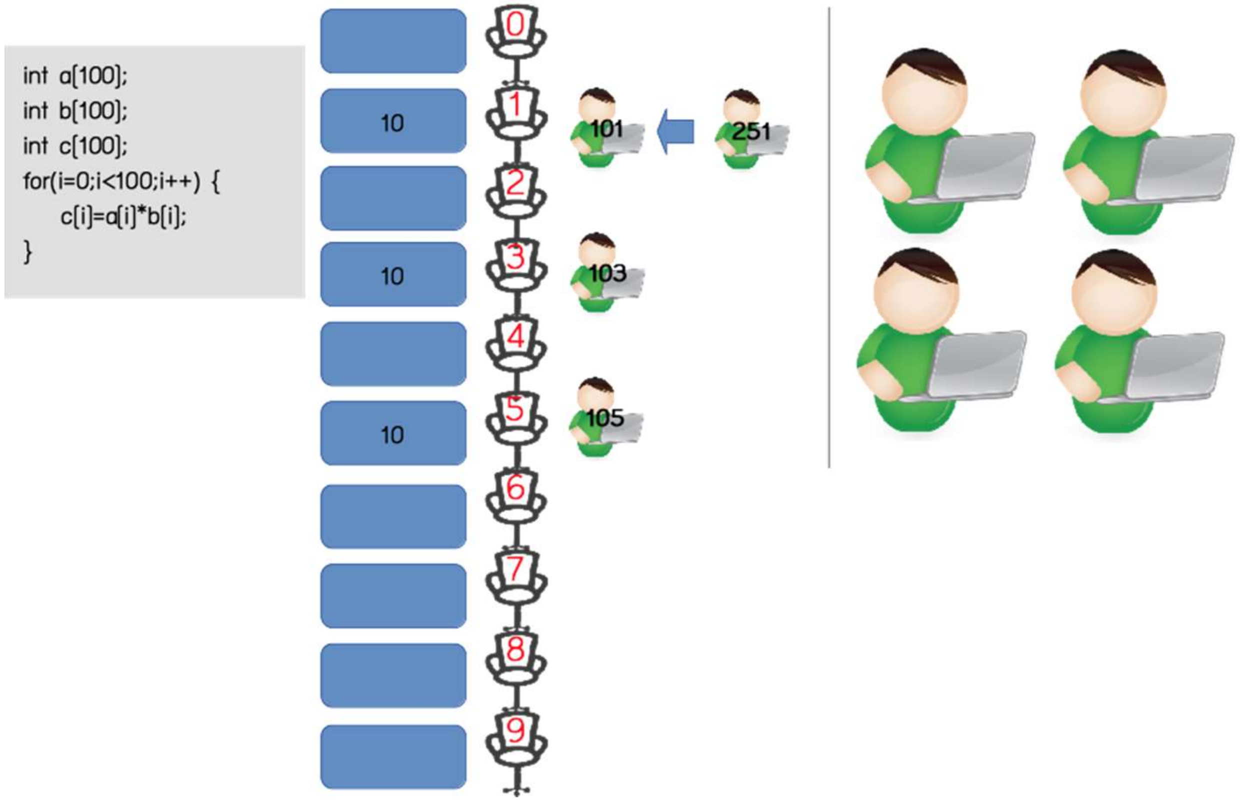
★ To reduce conflict miss, an array of structure may help.

```
// three arrays
int a[100];
int b[100];
int c[100];
for (int i=0; i<100; i++) {
     c[i] = a[i] * b[i];
}
```

```
// array of struct
struct Data {
     int a;
     int b;
     int c;
}
data[100];
for (int i=0;i<100;i++) {
     data[i].c = data[i].a * data[i].b;
}
```

Reduce Conflict Miss (ctd.)

- ★ Same index of an array is likely to share the same index of Cache.



```
int a[100];
int b[100];
int c[100];
for(i=0;i<100;i++) {
    c[i]=a[i]*b[i];
}
```

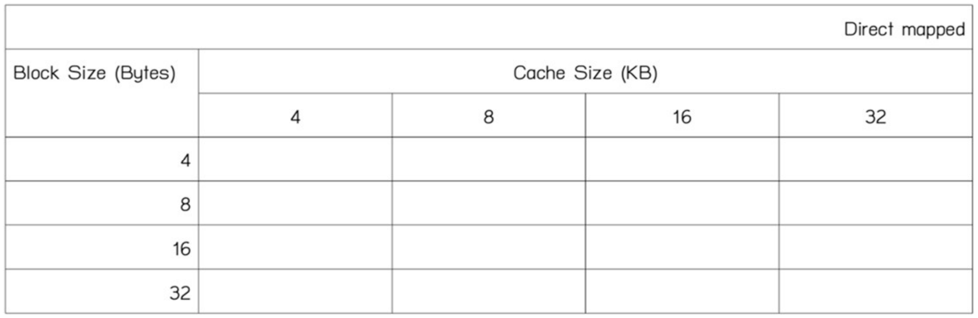
Summary

- ★ Principle of locality
  - Program likely accesses a relatively small chunk of memory at any instant of time.
  - Temporal locality - Locality in Time
  - Spatial locality - Locality in Space
- ★ Reducing Miss
  - Cold Miss, Capacity Miss, Conflict Miss
- ★ Cache organization
  - Cache size
  - Block size
  - Associativity
  - Replacement Algorithm
  - Write Management

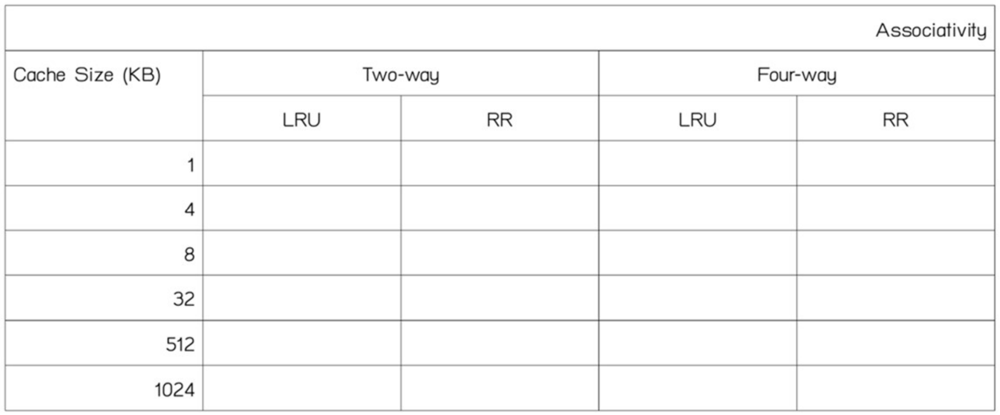
Exercises

Assignment II

★ Use the cache simulator for studying the factors that affects the performance of cache accesses. The cache simulator will read an address trace file from gcc_ld_trace.txt or go_ld_trace.txt (You can use genAddr.c for generating more address traces). The simulator will show us a number of cache hit, a number of cache miss, miss rate, and access time. Please use only one address trace and modify the code of simulator for calculating the miss rate. Please fill your results and plot graph of each table.

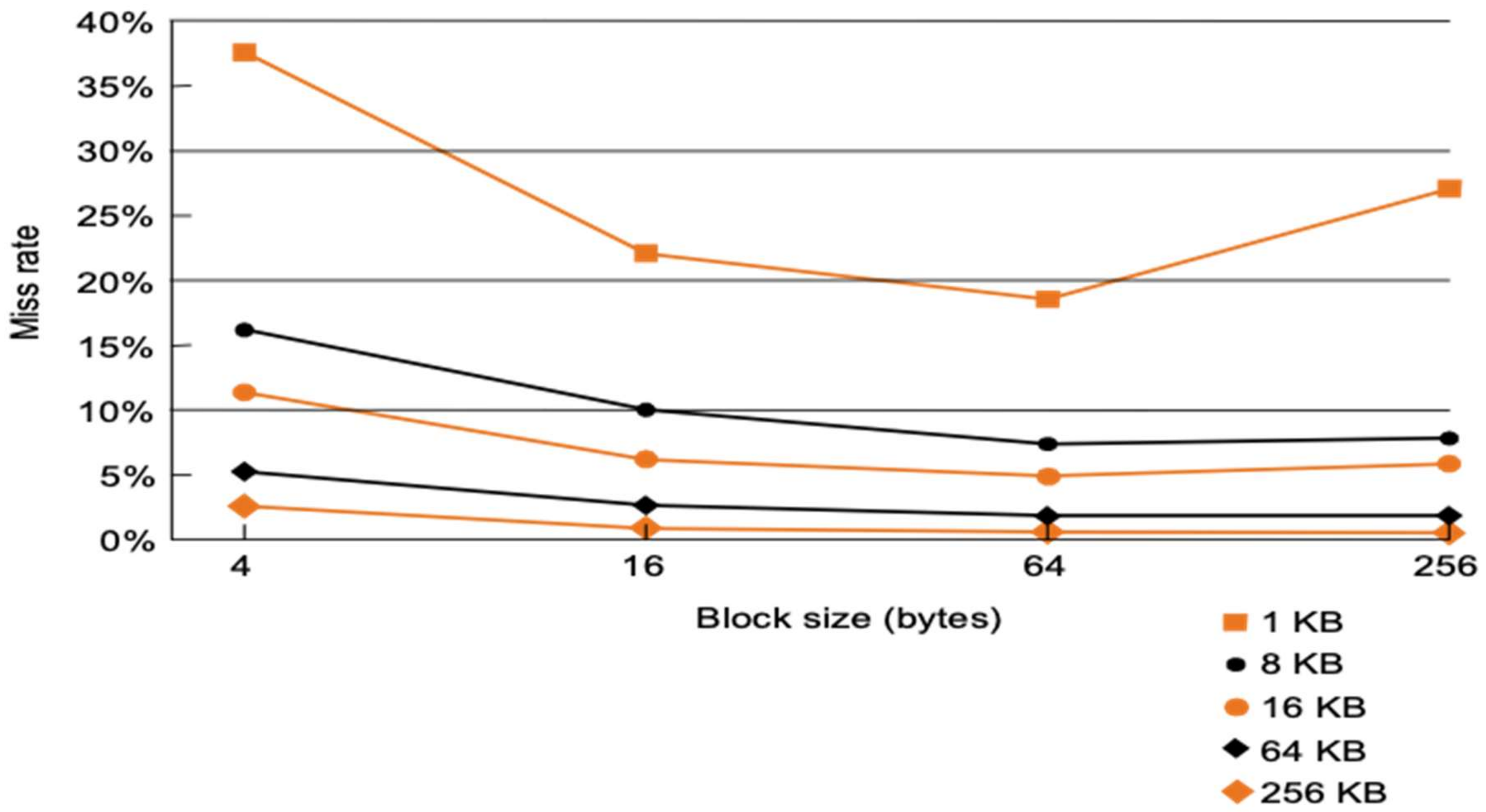


				Direct mapped
Block Size (Bytes)	Cache Size (KB)			
	4	8	16	32
4				
8				
16				
32				

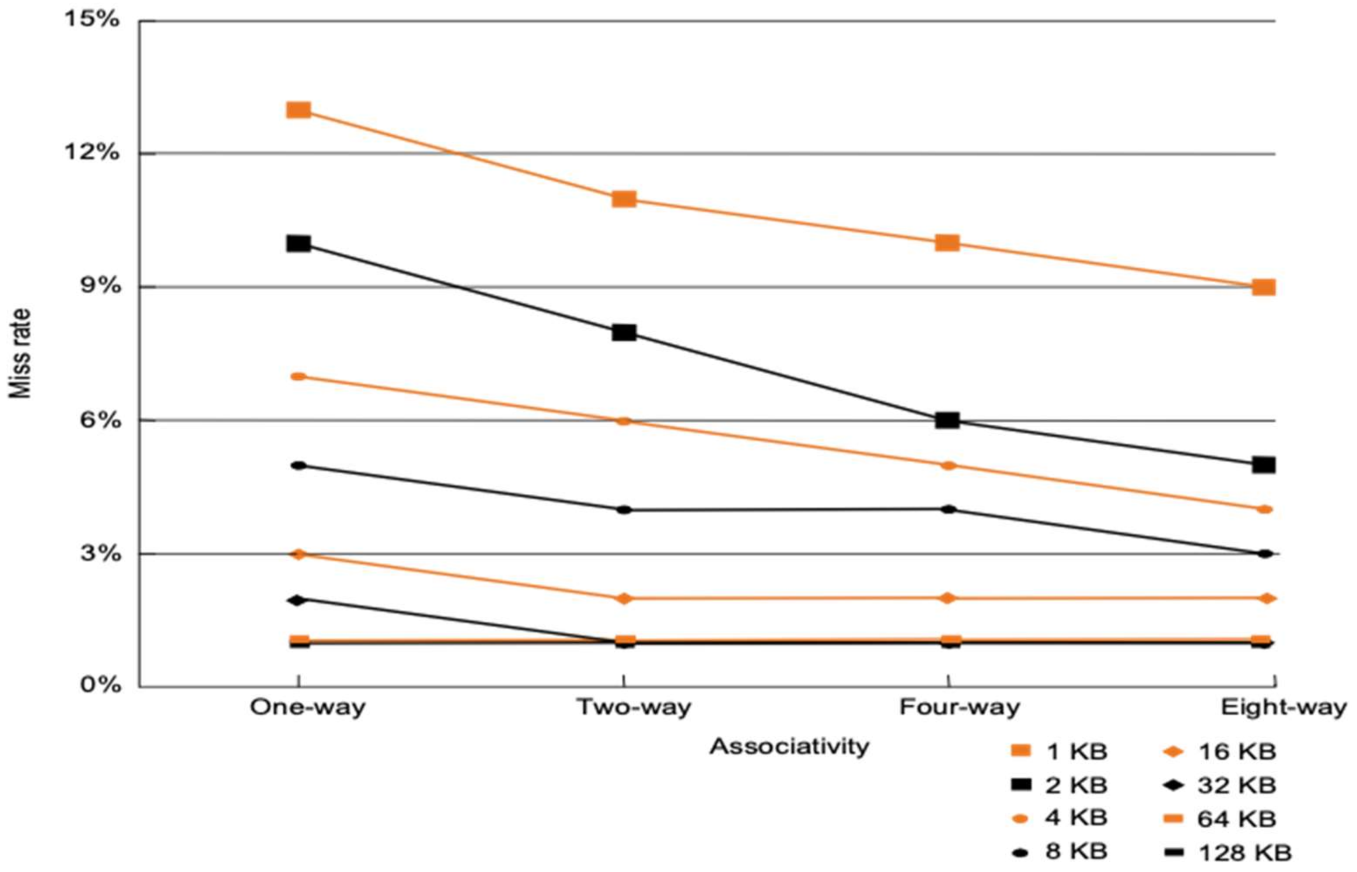


				Associativity
Cache Size (KB)	Two-way		Four-way	
	LRU	RR	LRU	RR
1				
4				
8				
32				
512				
1024				

Expected Result (Block size)



Expected Result (Associativity)



Multilevel Cache

- ★ What is effective access time of multilevel Cache?

- ★ $\text{Access Time}_0 = \text{Hit Time}_0 + (\text{Miss Rate}_0 \times \text{Miss Penalty}_0)$
- ★ $\text{Miss Penalty}_0 = \text{Access Time}_1$?

End of Chapter 8 (part 1)