Review Session

Memory Hierarchy, Systems Fundamentals, Virtual Memory

Aadam Lokhandwala, Joe Chiu

UMass Amherst | College of Information & Computer Sciences

How to prepare for the exam?

- Read through all the lecture slides
- Review all the weekly quiz
- Use office hours if you have any doubt about any concept

Memory Hierarchy

Things you should know from this chapter:

- Cache Miss vs Hit and Locality
- Tag bits, set bits, and offset bits in memory address
- Cache Placement Policy: Direct Mapped, Fully Associative, and E-way Set Associative Cache
- Replacement Algorithm: LRU and LFU

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory

What is the minimal number of block loads from grid if the block size is 8 bytes?

```
int total = 0;
for (i = 0; i < 3; ++i) {
    for (j = 0; j < 3; ++j) {
        total = grid[i][j];
    }
}
```

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory

What is the minimal number of block loads from grid if the block size is 8 bytes?

Answer: 5

```
int total = 0;
for (i = 0; i < 3; ++i) {
    for (j = 0; j < 3; ++j) {
        total = grid[i][j];
    }
}
```

grid	(0, 0)	(0, 1)	(0, 2)	(1, 0)	(1, 1)	(1, 2)	(2, 0)	(2, 1)	(2, 2)
Block	B1	B1	B2	B2	B3	B3	B4	B4	B5

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)
    - Miss Penalty: Extra Time to load data to the cache if it is a cache miss

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)
    - Miss Penalty: Extra Time to load data to the cache if it is a cache miss

-> Hit Rate = 1 - Miss Rate

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)
    - Miss Penalty: Extra Time to load data to the cache if it is a cache miss

-> Miss Time = Hit Time + Miss Penalty

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit

- Temporal Locality vs Spatial Locality

    - A program with good locality can significantly reduce miss rate and their penalty

- Block is the basic unit when transfer between memory

- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)
    - Miss Penalty: Extra Time to load data to the cache if it is a cache miss

-> Memory Access Time = Average Memory Access Time * (Number of Hit + Number of Miss)

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit
- Temporal Locality vs Spatial Locality
    - A program with good locality can significantly reduce miss rate and their penalty
- Block is the basic unit when transfer between memory
- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)
    - Miss Penalty: Extra Time to load data to the cache if it is a cache miss

-> Memory Access Time = Average Memory Access Time * (Number of Hit + Number of Miss)
= Hit Time * Number of Hit + Miss Time * Number of Miss

Aadam Lokhandwala, Joe Chiu

Cache Hit / Miss and Locality

- Cache Miss vs Cache Hit

- Temporal Locality vs Spatial Locality

    - A program with good locality can significantly reduce miss rate and their penalty

- Block is the basic unit when transfer between memory

- Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty
    - Hit Time: Time to load data from cache if it is a cache hit
    - Miss Rate: Number of Miss / (Number of Hit + Number of Miss)
    - Miss Penalty: Extra Time to load data to the cache if it is a cache miss

-> Memory Access Time = Average Memory Access Time * (Number of Hit + Number of Miss)
= Hit Time * Number of Hit + Miss Time * Number of Miss
= Hit Time * Number of Hit + (Hit Time + Miss Penalty) * Number of Miss

Tag, set, and offset bits in memory address

In a 8-bit memory address system. Let $C[s][l][b]$ denote the value in cache in set s , line l , and byte b line l , all of which are zero-indexed, for instance C[0][1][2] refers to the third byte in the second line of the first set. Let S=2 be the number of sets, E=8 be the number of lines per set, and B=16 be the number of bytes per block (the block being the data portion of a given line). Assume the tag field is in the most-significant bits [leftmost] and the block field is the least-significant bit [rightmost], What is the memory address (in hexadecimal) of that data in C[0][2][7] with the tag bit 2?

Tag, set, and offset bits in memory address

In a 8-bit memory address system. Let $C[s][l][b]$ denote the value in cache in set s , line l , and byte b line l , all of which are zero-indexed, for instance $C[0][1][2]$ refers to the third byte in the second line of the first set. Let S=2 be the number of sets, E=8 be the number of lines per set, and B=16 be the number of bytes per block (the block being the data portion of a given line). Assume the tag field is in the most-significant bits [leftmost] and the block field is the least-significant bit [rightmost], What is the memory address (in hexadecimal) of that data in $C[0][2][7]$ with the tag bit 2?

Cache

S=0	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes
S=1	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes

Aadam Lokhandwala, Joe Chiu

Tag, set, and offset bits in memory address

In a 8-bit memory address system. Let $C[s][l][b]$ denote the value in cache in set s , line l , and byte b line l , all of which are zero-indexed, for instance $C[0][1][2]$ refers to the third byte in the second line of the first set. Let S=2 be the number of sets, E=8 be the number of lines per set, and B=16 be the number of bytes per block (the block being the data portion of a given line). Assume the tag field is in the most-significant bits [leftmost] and the block field is the least-significant bit [rightmost], What is the memory address (in hexadecimal) of that data in C[0][2][7] with the tag bit 2?

8 bits		
	Set=1	

Aadam Lokhandwala, Joe Chiu

Tag, set, and offset bits in memory address

In a 8-bit memory address system. Let $C[s][l][b]$ denote the value in cache in set s , line l , and byte b line l , all of which are zero-indexed, for instance $C[0][1][2]$ refers to the third byte in the second line of the first set. Let S=2 be the number of sets, E=8 be the number of lines per set, and B=16 be the number of bytes per block (the block being the data portion of a given line). Assume the tag field is in the most-significant bits [leftmost] and the block field is the least-significant bit [rightmost], What is the memory address (in hexadecimal) of that data in $C[0][2][7]$ with the tag bit 2?

8 bits		
	Set=1	Offset=4

Aadam Lokhandwala, Joe Chiu

Tag, set, and offset bits in memory address

In a 8-bit memory address system. Let $C[s][l][b]$ denote the value in cache in set s , line l , and byte b line l , all of which are zero-indexed, for instance C[0][1][2] refers to the third byte in the second line of the first set. Let S=2 be the number of sets, E=8 be the number of lines per set, and B=16 be the number of bytes per block (the block being the data portion of a given line). Assume the tag field is in the most-significant bits [leftmost] and the block field is the least-significant bit [rightmost], What is the memory address (in hexadecimal) of that data in C[0][2][7] with the tag bit 2?

8 bits		
Tag=8-1-4=3	Set=1	Offset=4

Tag, set, and offset bits in memory address

In a 8-bit memory address system. Let $C[s][l][b]$ denote the value in cache in set s , line l , and byte b line l , all of which are zero-indexed, for instance C[0][1][2] refers to the third byte in the second line of the first set. Let S=2 be the number of sets, E=8 be the number of lines per set, and B=16 be the number of bytes per block (the block being the data portion of a given line). Assume the tag field is in the most-significant bits [leftmost] and the block field is the least-significant bit [rightmost], What is the memory address (in hexadecimal) of that data in C[0][2][7] with the tag bit 2?

8 bits		
Tag=3	Set=1	Offset=4
010	0	0111

Answer is 0100 0111 in binary, which is 47 in hexadecimal

Aadam Lokhandwala, Joe Chiu

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

11010

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ssb

11010

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ss b
11 01 0

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ss b

11 01 0

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ss b

11 01 0

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

t t ssb

11010

HIT, load the first byte.

Aadam Lokhandwala, Joe Chiu

UMass Amherst | College of Information & Computer Sciences

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

01110

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ssb
01110

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ssb

$01\boxed{11}0$

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ss b

01 $\boxed{11}$ 0

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

t t ss b

0 1 1 1 0

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

Cache Placement Policy

Consider a 2-way set associative cache running LRU replacement algorithm that services a memory space of 32 byte locations. 5-bit memory addresses are divided into the format of [ttssb]. In each set, the top row is most recently used and bottom row is least recently used. Indicate whether each of the following memory accesses was a hit or a miss. Noticed that all given addresses are independent, so the state of the cache is not changed after accessing each of the given addresses.

Set	Valid	Tag	Block
0	1	10	
	1	11	
1	1	11	
	0	00	
2	1	00	
	0	11	
3	0	01	
	0	00	

01110

MISS, check the next level.

Cache Replacement Policy

A = [4, 1, 4, 3, 4, 1, 2, 3, 3, 4, 3, 1, 3, 2, 1, 3]

Suppose A, a list of address requests. We execute those requests starting with the request at index 0 in the list. We are using the **Least Recently Used** replacement policy to cache those request. The capacity of the cache is 3 addresses.

Suppose A, a list of address requests. We execute those requests starting with the request at index 0 in the list. We are using the **Least Frequently Used** replacement policy to cache those request. The capacity of the cache is 3 addresses. If there is a tie, the smallest number is removed.

Cache Replacement Policy

A = [4, 1, 4, 3, 4, 1, 2, 3, 3, 4, 3, 1, 3, 2, 1, 3]

Suppose A, a list of address requests. We execute those requests starting with the request at index 0 in the list. We are using the **Least Recently Used** replacement policy to cache those request. The capacity of the cache is 3 addresses.

A: 1, 2, 3 are the final data in the cache after visiting all the data since they are the most recent 3 elements

Suppose A, a list of address requests. We execute those requests starting with the request at index 0 in the list. We are using the **Least Frequently Used** replacement policy to cache those request. The capacity of the cache is 3 addresses. If there is a tie, the smallest number is removed.

Cache Replacement Policy

A = [4, 1, 4, 3, 4, 1, 2, 3, 3, 4, 3, 1, 3, 2, 1, 3]

Suppose A, a list of address requests. We execute those requests starting with the request at index 0 in the list. We are using the **Least Recently Used** replacement policy to cache those request. The capacity of the cache is 3 addresses.

A: 1, 2, 3 are the final data in the cache after visiting all the data since they are the most recent 3 elements

Suppose A, a list of address requests. We execute those requests starting with the request at index 0 in the list. We are using the **Least Frequently Used** replacement policy to cache those request. The capacity of the cache is 3 addresses. If there is a tie, the smallest number is removed.

A: 1 3 4 are the final data in the cache after visiting all the data since the frequency of them are 1 5 4. Notice that the frequency is reset to 0 when the data is evicted from the cache during the replacement process.

Systems Fundamentals

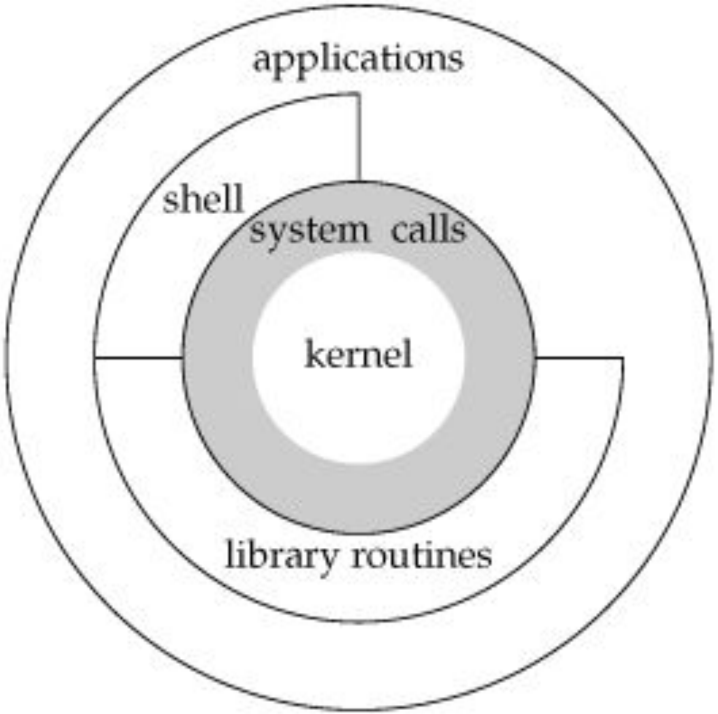
Things you should know from this chapter:

- What are system calls and library calls?
- How to do error handling for system calls? [errno]
- What are 3 OS abstraction?
- What are the 3 table used to manipulate files?
- System calls used for manipulating files:
    - open
    - read
    - write
    - close
    - lseek

Aadam Lokhandwala, Joe Chiu

System Calls vs Library Calls

Figure taken from lecture slides:



Are system calls always faster than their library counterparts?

False

ERRNO

Taken from lecture slides:

- When a system call fails, it sets the global integer variable errno to a positive value that identifies the specific error
- Including the <errno.h> header file provides a declaration of errno, as well as a set of constants for the various error numbers
- The section headed ERRORS in each manual page provides a list of possible errno values that can be returned by each system call

ERRNO

Taken from lecture slides:

- When a system call fails, it sets the global integer variable errno to a positive value that identifies the specific error

- Including the <errno.h> header file provides a declaration of errno, as well as a set of constants for the various error numbers

- The section headed ERRORS in each manual page provides a list of possible errno values that can be returned by each system call

errno gets reset to 0, everytime a system call is successful?

ERRNO

Taken from lecture slides:

- When a system call fails, it sets the global integer variable errno to a positive value that identifies the specific error

- Including the <errno.h> header file provides a declaration of errno, as well as a set of constants for the various error numbers

- The section headed ERRORS in each manual page provides a list of possible errno values that can be returned by each system call

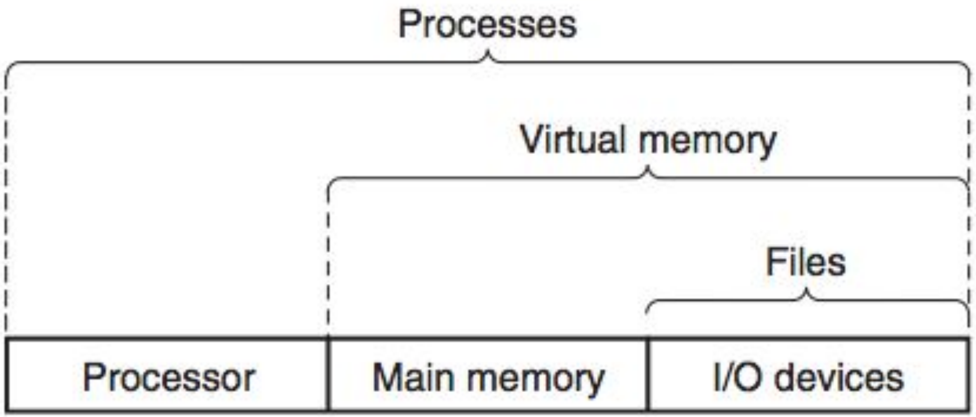
errno gets reset to 0, everytime a system call is successful?

False

What are the 3 OS Abstractions?

What are the 3 OS Abstractions?

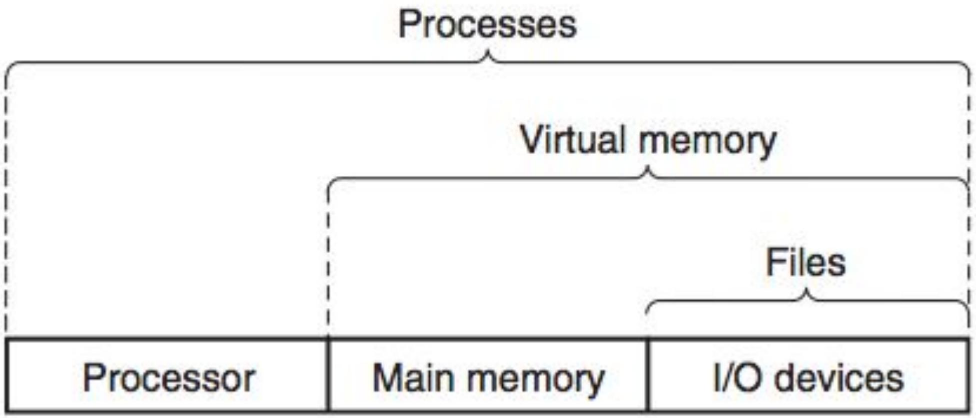
Figure taken from lecture slides:



Aadam Lokhandwala, Joe Chiu

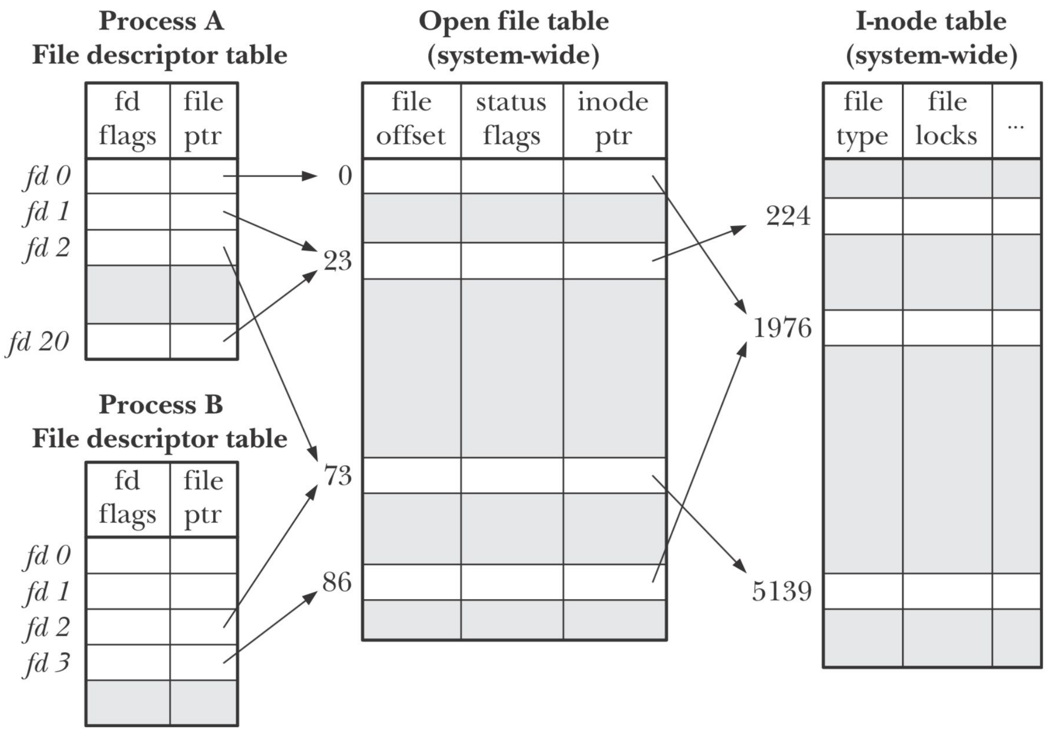
What are the 3 OS Abstractions?

Figure taken from lecture slides:



Now recap from the worksheet 6!

Three tables behind the files system



Aadam Lokhandwala, Joe Chiu

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1)
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

file1.txt:

ABCD

Process A File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…

Open File Table (System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .

What are the first 3 entry in file descriptor table?

Figure taken from lecture slides:

What will be the final output of this code?

File descriptor	Purpose	POSIX name	*stdio* stream
0	standard input	STDIN_FILENO	*stdin*
1	standard output	STDOUT_FILENO	*stdout*
2	standard error	STDERR_FILENO	*stderr*

Aadam Lokhandwala, Joe Chiu

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1)
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20

Open File Table
(System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
0	…	100

file1.txt:

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1)
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21

Open File Table
(System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
0	…	100
0	…	250

file1.txt:

ABCD
Aadam Lokhandwala, Joe Chiu
UMass Amherst | College of Information & Computer Sciences

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20

Open File Table
(System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
0	…	100
0	…	250

file1.txt:

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

file1.txt:

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table
(System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
0	…	100
0	…	250
0	…	100

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table (System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
0 => 6	…	100
0	…	250
0	…	100

file1.txt:

ABCD => CS230

Aadam Lokhandwala, Joe Chiu

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

file1.txt:

|CS230 || => |CS230 Nikko||

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table
(System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
6 => 11	…	100
0	…	250
0	…	100

Aadam Lokhandwala, Joe Chiu

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table (System Wide)

file offset	status flag	inode pointer
. . .	. . .	. . .
11	…	100
0	…	250
0 => 5	…	100

file1.txt:

|CS230 Nikko|| => Joe C| Nikko||

Aadam Lokhandwala, Joe Chiu

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table
(System Wide)

file offset	status flag	inode
. . .	. . .	. . .
11 - 3 => 8	…	100
0	…	250
5	…	100

file1.txt:

Joe C| Nikko|| => Joe C| Ni||kko

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

Process A
File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table
(System Wide)

file offset	status flag	inode
. . .	. . .	. . .
8	…	100
0	…	250
5 => 11	…	100

file1.txt:

Joe C| Ni|| kko => Joe C Ni|| kko|

Aadam Lokhandwala, Joe Chiu

What happens when I run this code?

```
int fd1 = open("file1.txt", O_RDWR);
int fd2 = open("file2.txt", O_RDWR);
int fd3 = dup(fd1);
int fd4 = open("file1.txt", O_RDWR);

write(fd1, "CS230 ", 6);
write(fd3, "Nikko", 5);
write(fd4, "Joe C", 5);

lseek(fd3, -3, SEEK_CUR);
lseek(fd4, 0, SEEK_END);
```

file1.txt:

Joe C Ni|||kko|

Process A File Descriptor Table

fd flags	file ptr
….	…
…	…
…	…
RD,WR	20
RD,WR	21
RD,WR	20
RD,WR	22

Open File Table (System Wide)

file offset	status flag	inode
. . .	. . .	. . .
8	…	100
0	…	250
11	…	100

Virtual Memory

Things you should know from this chapter:

- How does Page Table works
- How to identify a page is Allocated/ Unallocated, Cached/Uncached
- What happens when page is not cached? (Page Fault)
- Address Translation (How to convert VA to PA)

Virtual Memory

Virtual Memory

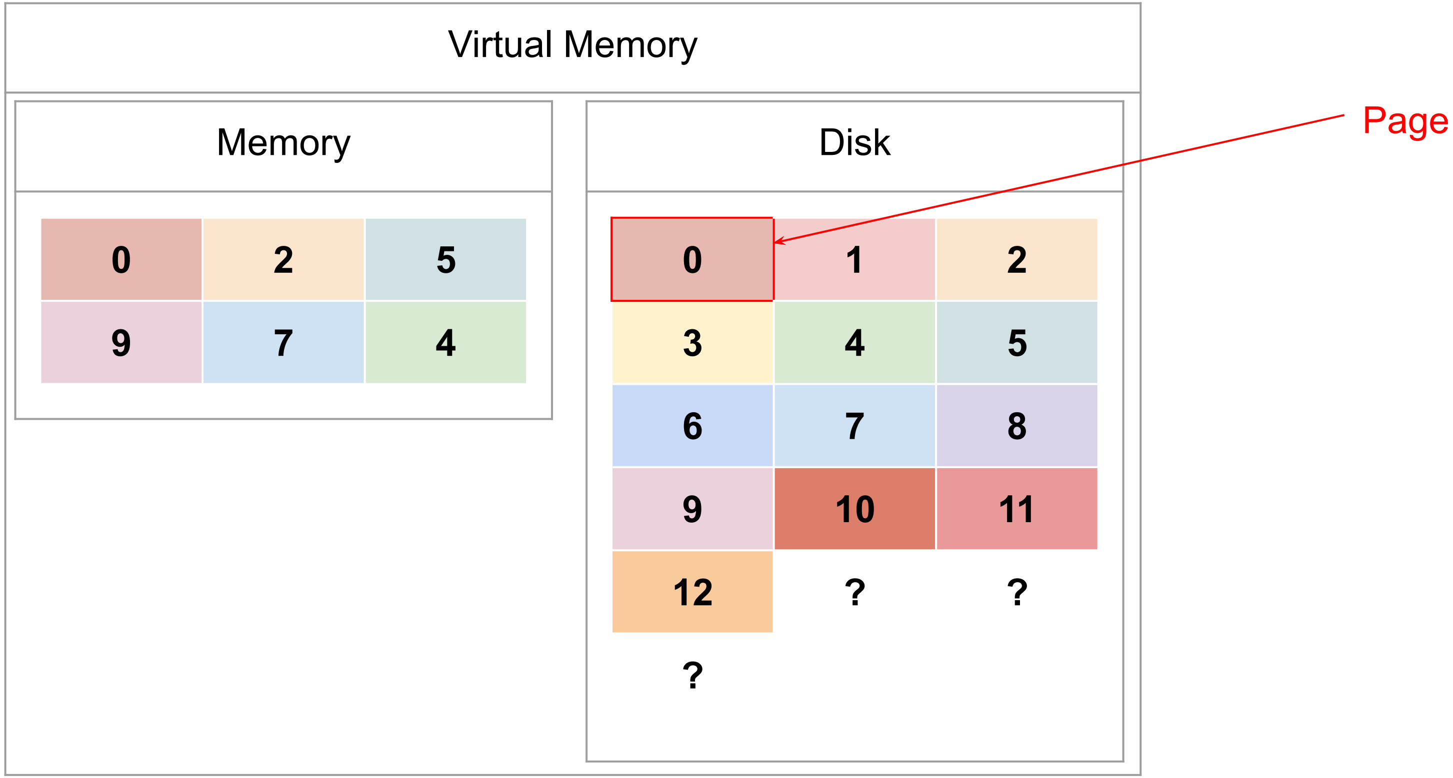
Memory

0	2	5
9	7	4

Disk

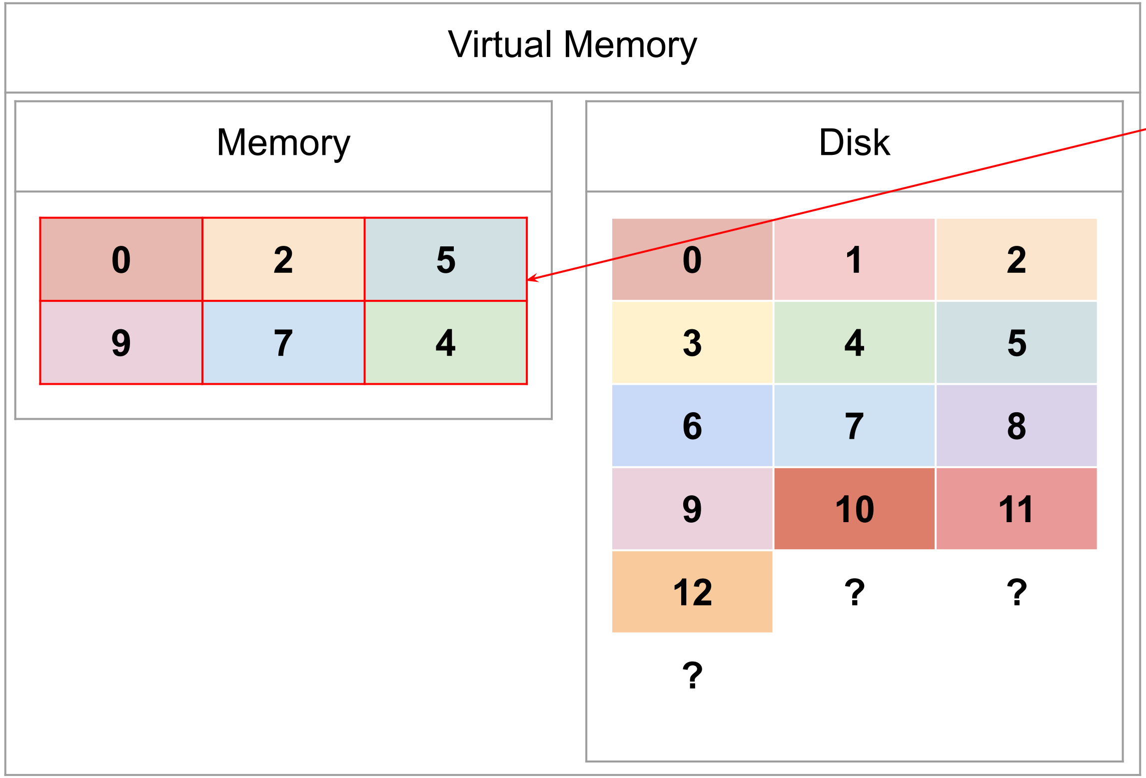
0	1	2
3	4	5
6	7	8
9	10	11
12	?	?
?		

Virtual Memory



Aadam Lokhandwala, Joe Chiu

Virtual Memory

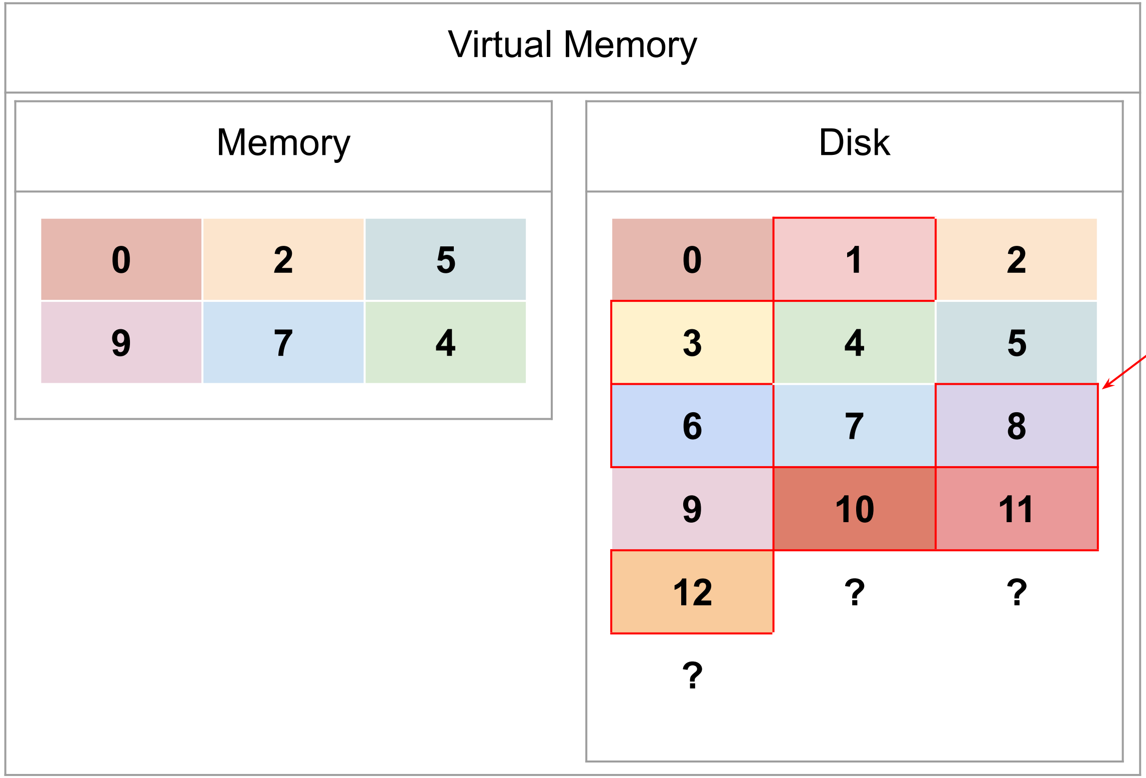


Allocated & Cached

Allocated & Un-Cached

Unallocated

Virtual Memory



Allocated & Cached

Allocated & Un-Cached

Unallocated

Aadam Lokhandwala, Joe Chiu

Virtual Memory

Virtual Memory

Memory

0	2	5
9	7	4

Disk

0	1	2
3	4	5
6	7	8
9	10	11
12	?	?
?		

Allocated & Cached

Allocated & Un-Cached

Unallocated

UMass Amherst | College of Information & Computer Sciences

Aadam Lokhandwala, Joe Chiu

Virtual Memory

Virtual Memory

Memory

0	2	5
9	7	4

Disk

0	1	2
3	4	5
6	7	8
9	10	11
12	?	?
?		

Page Table

1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

Virtual Memory

Virtual Memory

Memory

0	2	5
9	7	4

Disk

0	1	2
3	4	5
6	7	8
9	10	11
12	?	?
?		

Page Table

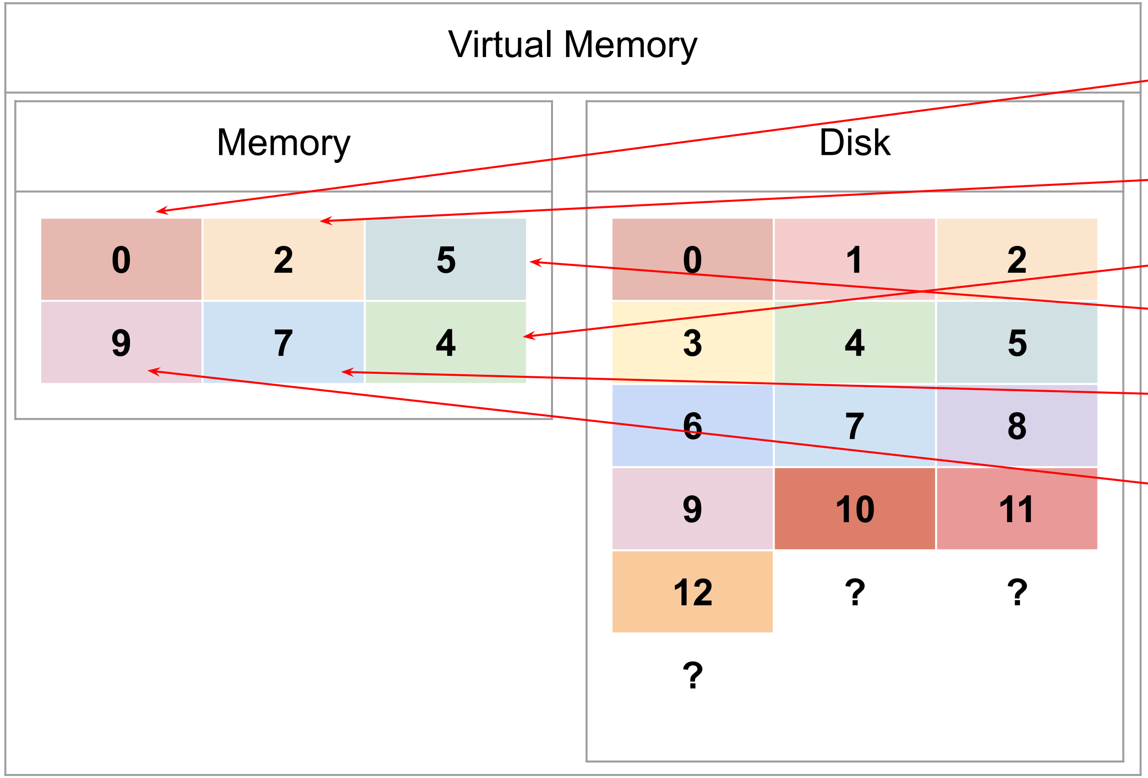
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

Page Table Entry (PTE)

Valid Bit

Address

Virtual Memory

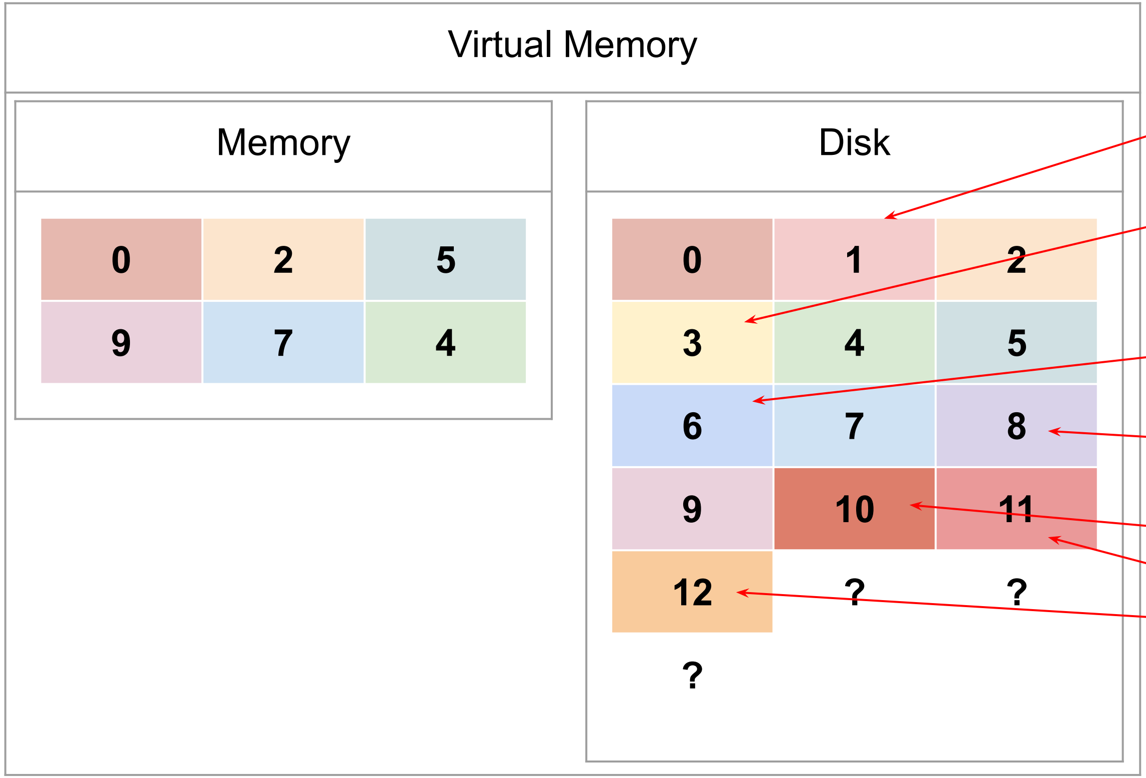


Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

When **valid bit is 1**, it means the page is in memory. Hence, it's **Cached**

Aadam Lokhandwala, Joe Chiu

Virtual Memory



Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

When **valid bit is 0 and address is not null**, it means the page is not is memory. Hence, it's **Un-Cached**

Virtual Memory

Virtual Memory

Memory

0	2	5
9	7	4

Disk

0	1	2
3	4	5
6	7	8
9	10	11
12	?	?
?		

Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

When **valid bit is 0 and address is null**, it means the page is not allocated yet. Hence, it's **Un-Allocated**

Address Translation

Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

Address Translation

Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

Which piece of hardware is responsible for converting VA to PA?

Address Translation

Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

Which piece of hardware is responsible for converting VA to PA?

Memory Management Unit (MMU)

Aadam Lokhandwala, Joe Chiu

Address Translation

Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

Which piece of hardware is responsible for converting VA to PA?

Memory Management Unit (MMU)

Can you convert from PA to VA?

Aadam Lokhandwala, Joe Chiu

Address Translation

Page Table	
1	0000
0	0001
1	0001
0	0011
1	0010
1	0101
0	0110
1	0100
0	1000
1	0011
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

Which piece of hardware is responsible for converting VA to PA?

Memory Management Unit (MMU)

Can you convert from PA to VA?

No, Address translation only works one way.

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

How many bits will VA and PA require? Let's say we are given M = 128, and virtual memory uses 6 bits.

Aadam Lokhandwala, Joe Chiu

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (n-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (m-p bits)	Page Offset(PO) (p bits)

How many bits will VA and PA require? Let's say we are given M = 128, and virtual memory uses 6 bits.

As $M = 2^m \Rightarrow 128 = 2^7 \Rightarrow m = 7$

And as virtual memory uses 6 bits $\Rightarrow n = 6$

Aadam Lokhandwala, Joe Chiu

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (6-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (7-p bits)	Page Offset(PO) (p bits)

How many bits will VA and PA require? Let's say we are given M = 128, and virtual memory uses 6 bits.

As $M = 2^m \Rightarrow 128 = 2^7 \Rightarrow m = 7$

And as virtual memory uses 6 bits $\Rightarrow n = 6$

Aadam Lokhandwala, Joe Chiu

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (6-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (7-p bits)	Page Offset(PO) (p bits)

How many bits do we need for VPN? That is how many bits do we require to index the page table?

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (6-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (7-p bits)	Page Offset(PO) (p bits)

How many bits do we need for VPN? That is how many bits do we require to index the page table?

As page table has 16 PTE, so we need 16 = 2^4 index, hence we will need 4 bits.

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (6-p bits)	Page Offset(PO) (p bits)

- Physical Address:

Physical Page Number(PPN) (7-p bits)	Page Offset(PO) (p bits)

How many bits do we need for VPN? That is how many bits do we require to index the page table?

As page table has 16 PTE, so we need 16 = 2^4 index, hence we will need 4 bits.

So VPN will require 4 bits $\Rightarrow$ 6-p = 4 $\Rightarrow$ p = 2

Aadam Lokhandwala, Joe Chiu

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (6-2 bits)	Page Offset(PO) (2 bits)

- Physical Address:

Physical Page Number(PPN) (7-2 bits)	Page Offset(PO) (2 bits)

How many bits do we need for VPN? That is how many bits do we require to index the page table?

As page table has 16 PTE, so we need $16 = 2^4$ index, hence we will need 4 bits.

So VPN will require 4 bits ⇒ 6-p = 4 ⇒ p = 2

Address Translation

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

How can we convert Virtual Address to Physical Address?

First let's see how VA and PA looks like:

- Virtual Address:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)

- Physical Address:

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
?	?

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

In decimal is 4

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

In decimal is 4

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

In decimal is 4

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Valid bit is 1, means it's cached

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

In decimal is 4

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
00010	?

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

In decimal is 4

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
00010	10

Aadam Lokhandwala, Joe Chiu

What will be 010010 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0100	10

In decimal is 4

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
00010	10

So, PA will be: 0b0001010

What will be 000100 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
?	?

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

What will be 000110 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0001	00

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

What will be 000110 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0001	00

In decimal is 1

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Aadam Lokhandwala, Joe Chiu

What will be 000110 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0001	00

In decimal is 1

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

UMass Amherst | College of Information & Computer Sciences

Aadam Lokhandwala, Joe Chiu

What will be 000110 in PA?

Page Table	
1	0
0	1
1	1
0	11
1	10
1	101
0	110
1	100
0	1000
1	11
0	1010
0	1011
0	1100
0	Null
0	Null
0	Null

So we have:

Virtual Page Number(VPN) (4 bits)	Page Offset(PO) (2 bits)
0001	00

In decimal is 1

Physical Page Number(PPN) (5 bits)	Page Offset(PO) (2 bits)
?	?

Valid bit is 0, means it's either Un-Cached, or Un-Allocated. In both cases we will get **Page Fault**!

UMass Amherst | College of Information & Computer Sciences

Aadam Lokhandwala, Joe Chiu