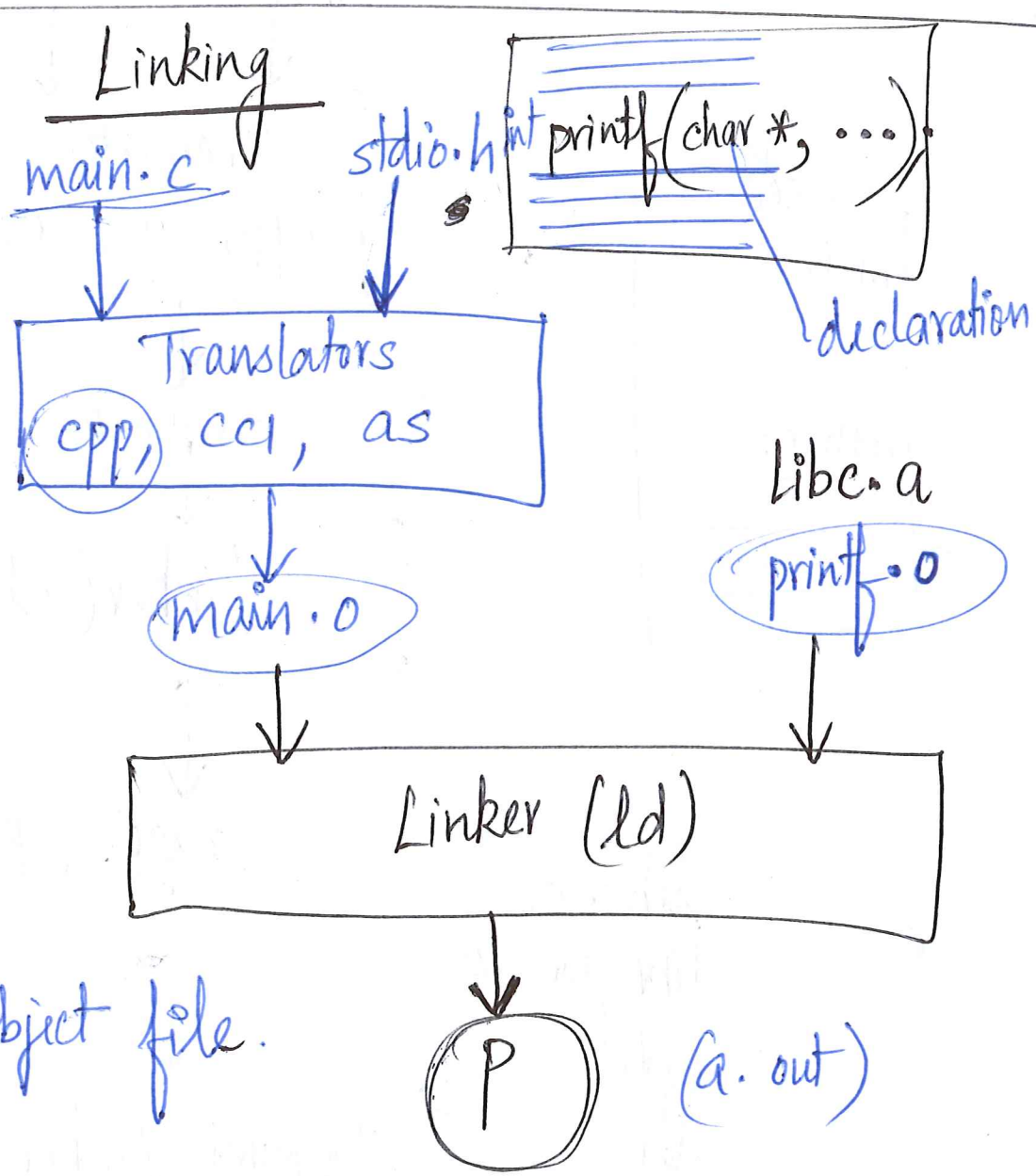
CS 354 - Lecture 14

Last Lecture!

1. Linking
2. Review.
3. What's next?

(1) "Course evals"

Linking



② Linking

static

main.c

addvec();
main() {
}

libvector.a

addvec
multvec

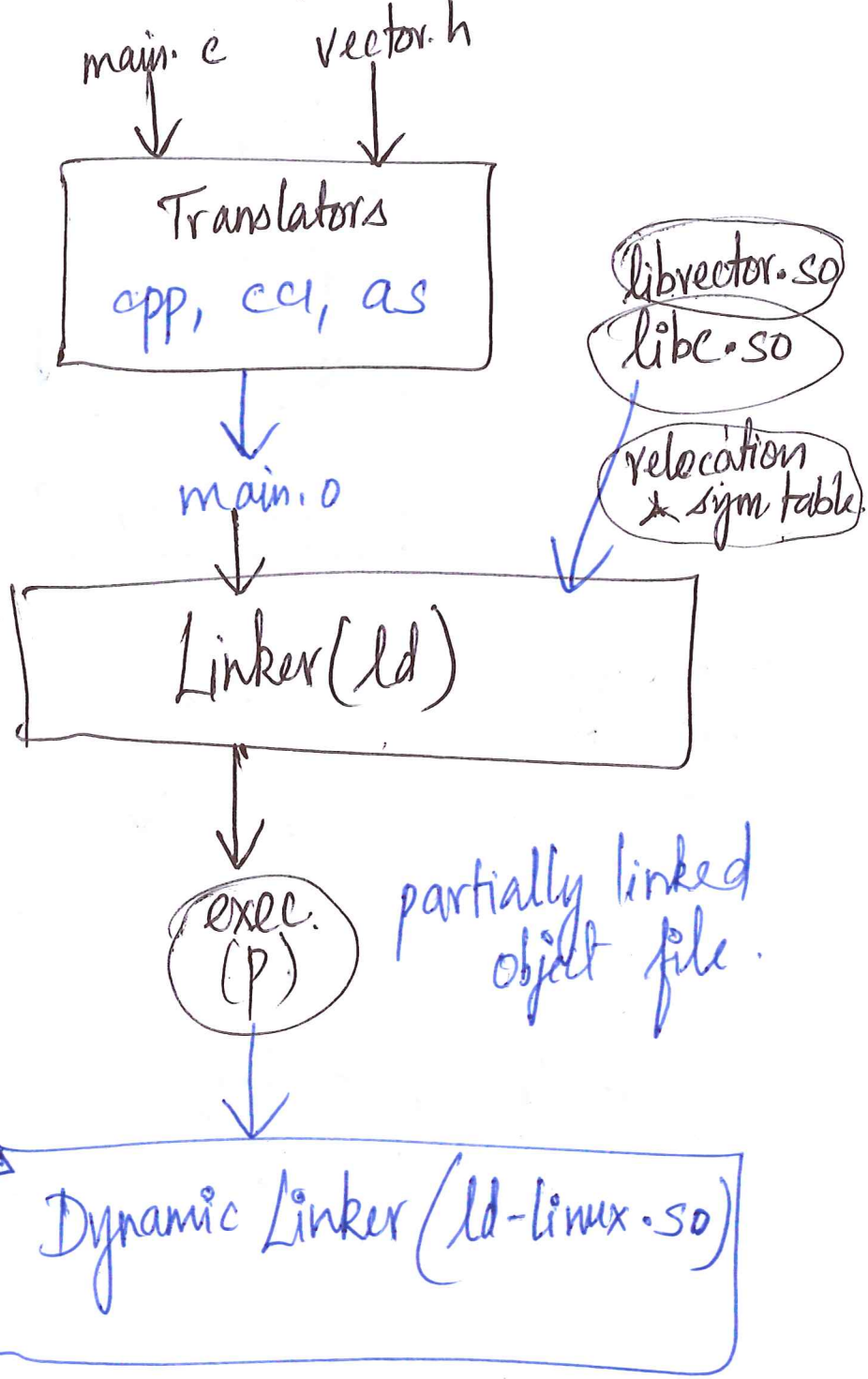
exec

main:

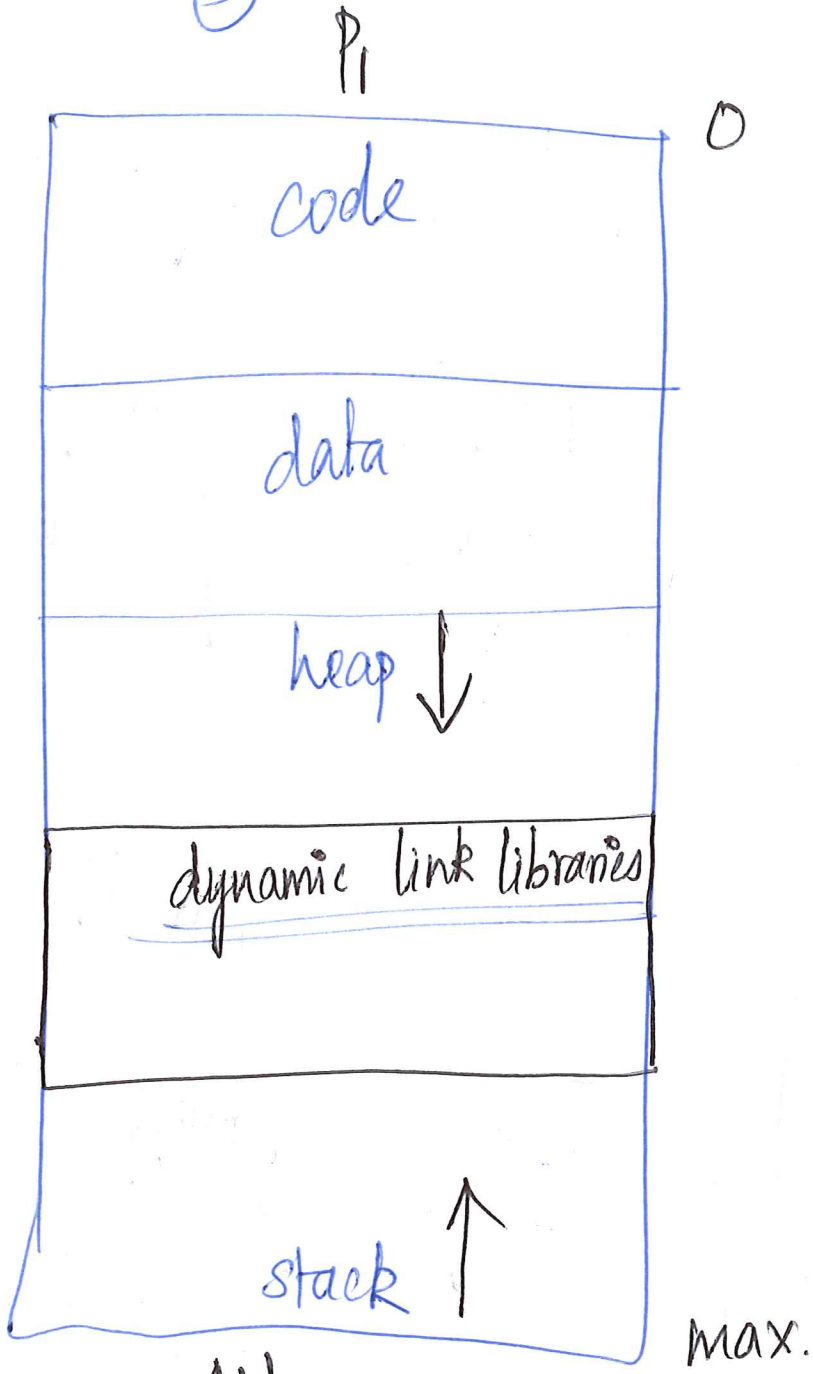
addvec:

dynamic

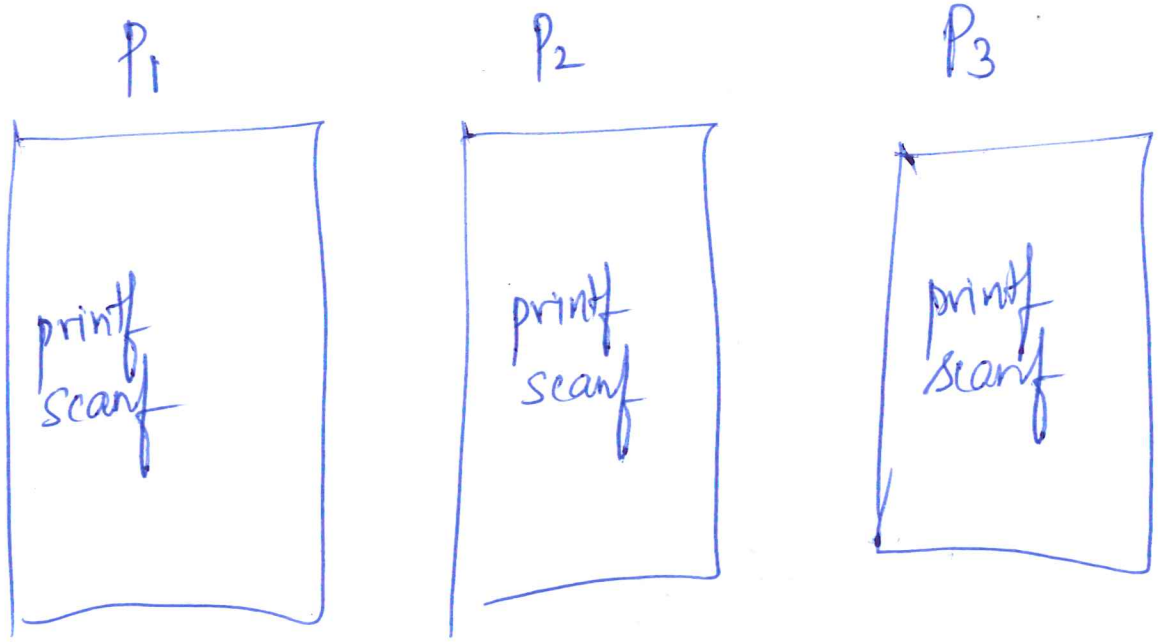
- .so (shared object) — Linux
- .dll (Dyn. link lib) — Windows

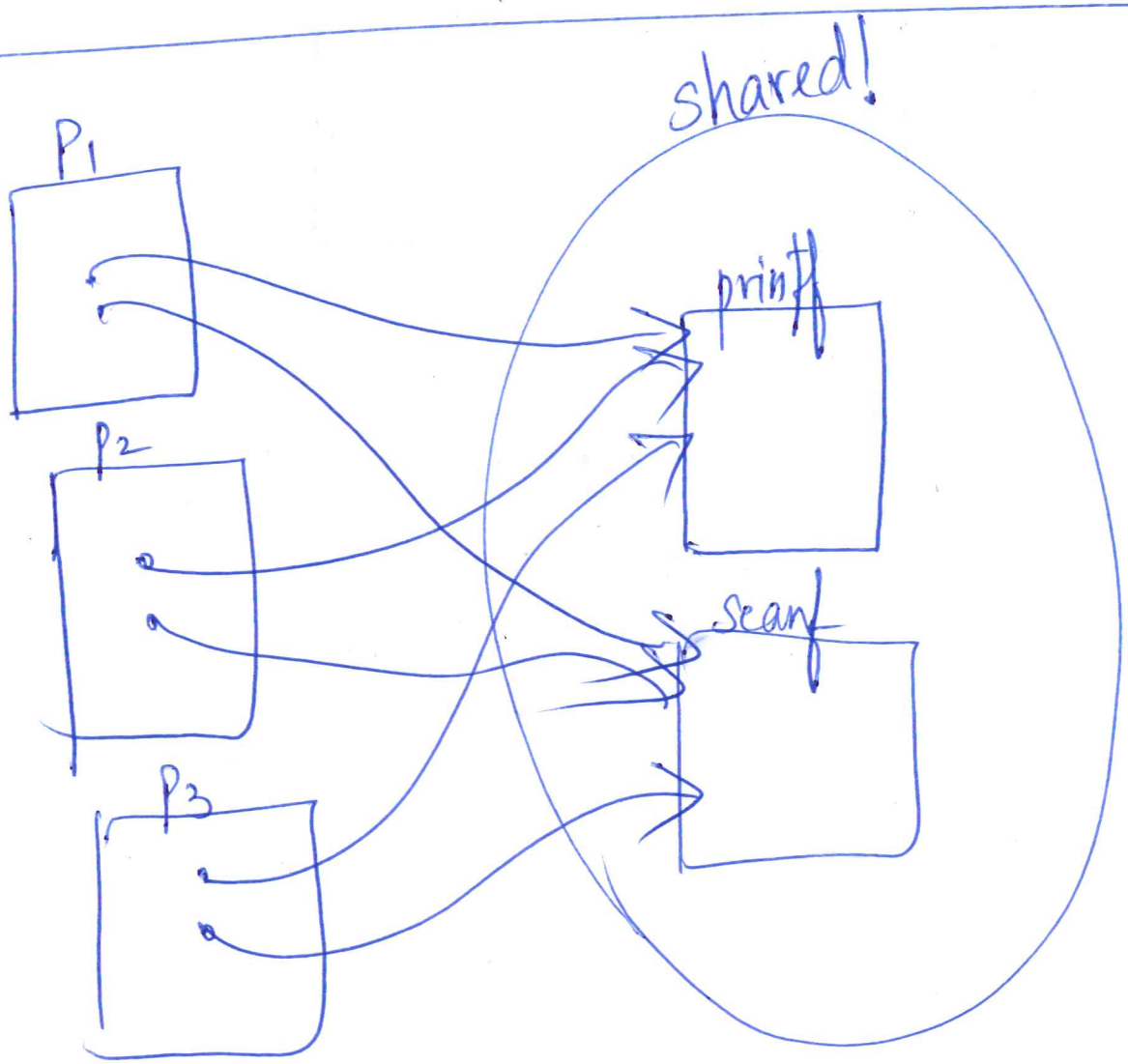


③



Address space of program P_1


P1
P2
P3
printf
scanf
printf
scanf
printf
scanf


shared!
P1
P2
P3
printf
scanf

2. Review – What did we learn? (5)
What to study for the final?

I. C programming (2)

- data types, if/else, functions, loops.
- arrays, pointers, structures
- dynamic memory alloc, linked lists
- low-level programming ("bits")
- Recursion, func. ptr, tools ← gdb, make, valgrind

II. Assembly

- Registers, mov, arithmetic, memory.
- if/else, loops
- Functions (1)
- Pointers (1)
- Arrays & structures (1)
- Stack smashing (1)

III. Systems

- Dynamic memory allocator. (2)
- Cache Memories (2)
- Linking (1)

6

Linking

Consider the three files **sum.h, sum.c** and **main.c** as shown below and answer the questions that follow.

sum.h

```
=====
#ifndef SUM_H
#define SUM_H
extern int sum(int, int);
#endif
```

sum.c

```
=====
#include "sum.h"
extern int num_ops;
static int global_sum = 0;

int sum(int x, int y)
{
    global_sum += (x+y);
    num_ops++;
    return x+y;
}
```

declaration.

main.c

```
======
#include "sum.h"
#define SUCCESS 0

int num_ops = 0;

int main()
{
    int a = 10;
    int b = 3;
    int result = sum(a,b);
    return SUCCESS;
}
```

declaration

defines

assignment

int x;

The final executable **a.out** is produced by running the following command:

```
% gcc sum.c main.c -m32
```

Questions:

A. The variable num_ops is **ONLY declared but not defined** in the file sum.c

B. The variable num_ops is **defined** in the file main.c

C. Do the following variables / functions need **relocation** when the executable (a.out) is formed? (Just answer: **Yes** or **No**).

a. Variable global_sum in sum.c - Y

b. Variable result in main.c - N

c. Variable num_ops in main.c - Y

d. Function sum() in sum.c - Y

e. Function main() in main.c - Y

D. Can the variable global_sum be accessed in the file main.c without changing the type of the variable global_sum in sum.c? Yes OR No.

E. What type of object file is **sum.o** ?
sum.o is generated using the command: gcc -c sum.c -m32

i. **Relocatable Object File**
ii. **Executable Object File**

F. Which part of **program memory** (code, data, stack, heap) are the following variables / functions stored?

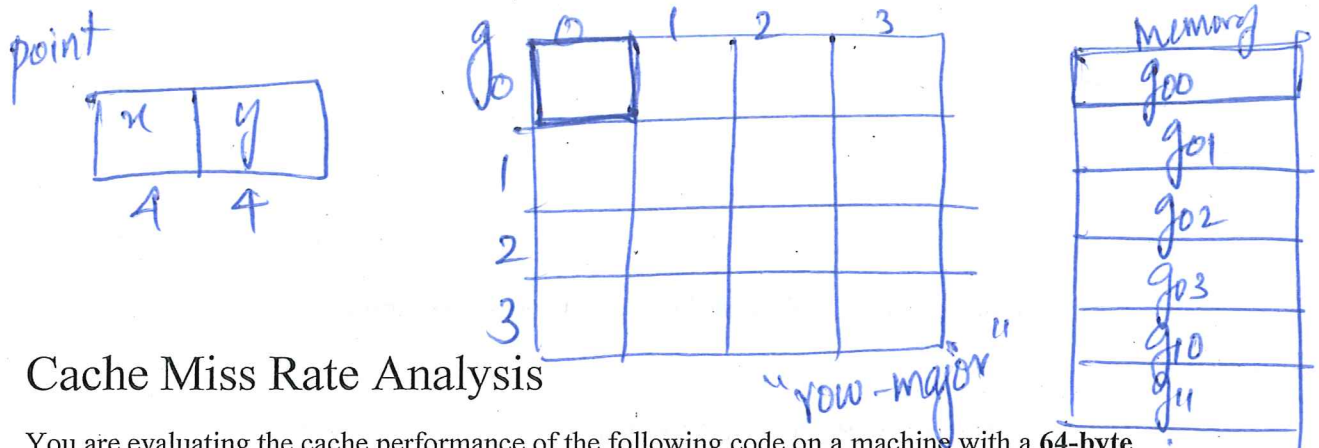
a. Variable global_sum in sum.c - Data

b. Variable result in main.c - Stack

c. Variable num_ops in main.c - Data

d. Function sum() in sum.c - Code/Text

e. Function main() in main.c - "

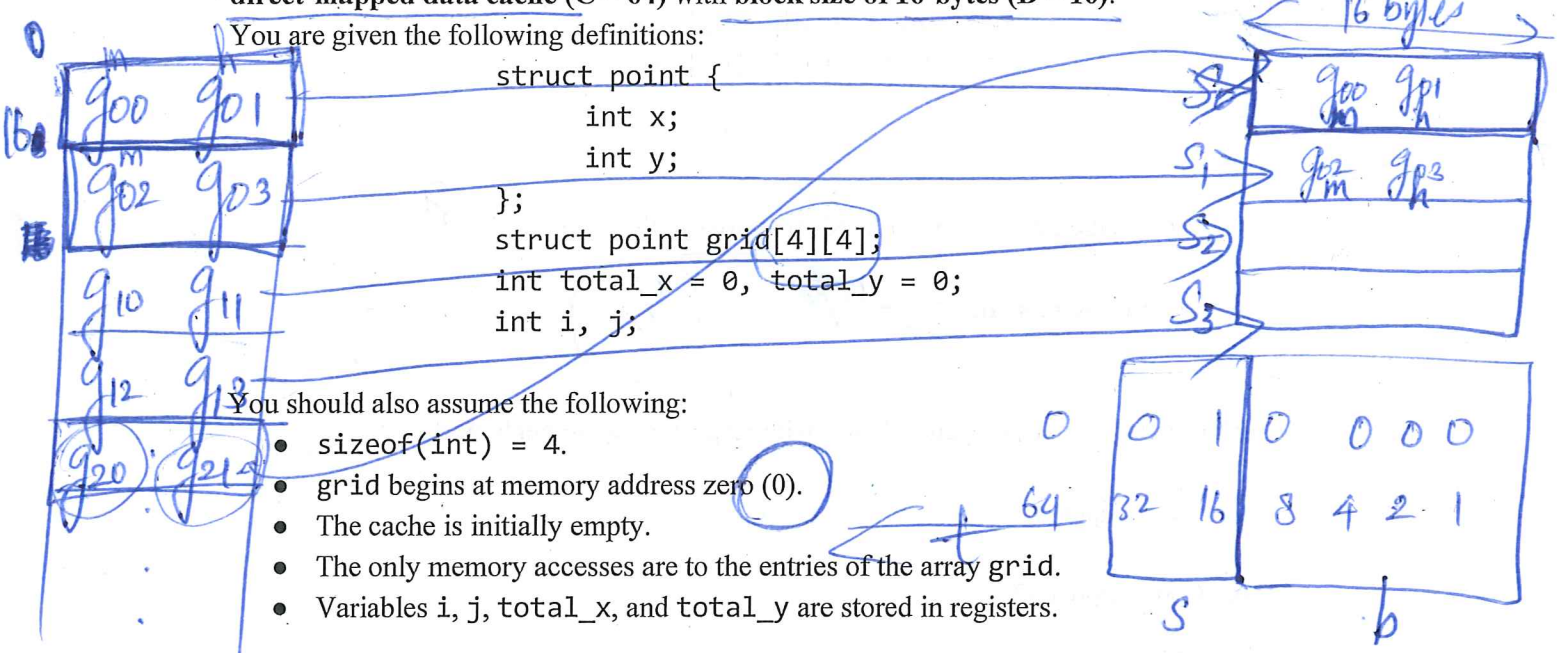


Cache Miss Rate Analysis

You are evaluating the cache performance of the following code on a machine with a **64-byte direct-mapped data cache (C = 64)** with **block size of 16-bytes (B = 16)**.

You are given the following definitions:

```
struct point {
        int x;
        int y;
};
struct point grid[4][4];
int total_x = 0, total_y = 0;
int i, j;
```



You should also assume the following:

- `sizeof(int) = 4`.
- `grid` begins at memory address zero (0).
- The cache is initially empty.
- The only memory accesses are to the entries of the array `grid`.
- Variables `i`, `j`, `total_x`, and `total_y` are stored in registers.

A. Determine the cache performance for the following **code snippet 1**:

Code snippet 1:

```
for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
                total_x += grid[i][j].x;
        }
}
```

16 reads → 8 misses

```
for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
                total_y += grid[i][j].y;
        }
}
```

16 reads → 8 misses

1. What is the total number of reads that **miss** in the cache? 16

2. What is the **miss rate**? 50%

B. Determine the cache performance for the following **code snippet 2**:

Code snippet 2:

```
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        total_x += grid[i][j].x;
        total_y += grid[i][j].y;
    }
}
```

1. What is the total number of reads that **miss** in the cache? 8

2. What is the **miss rate**? 25% (1/4)

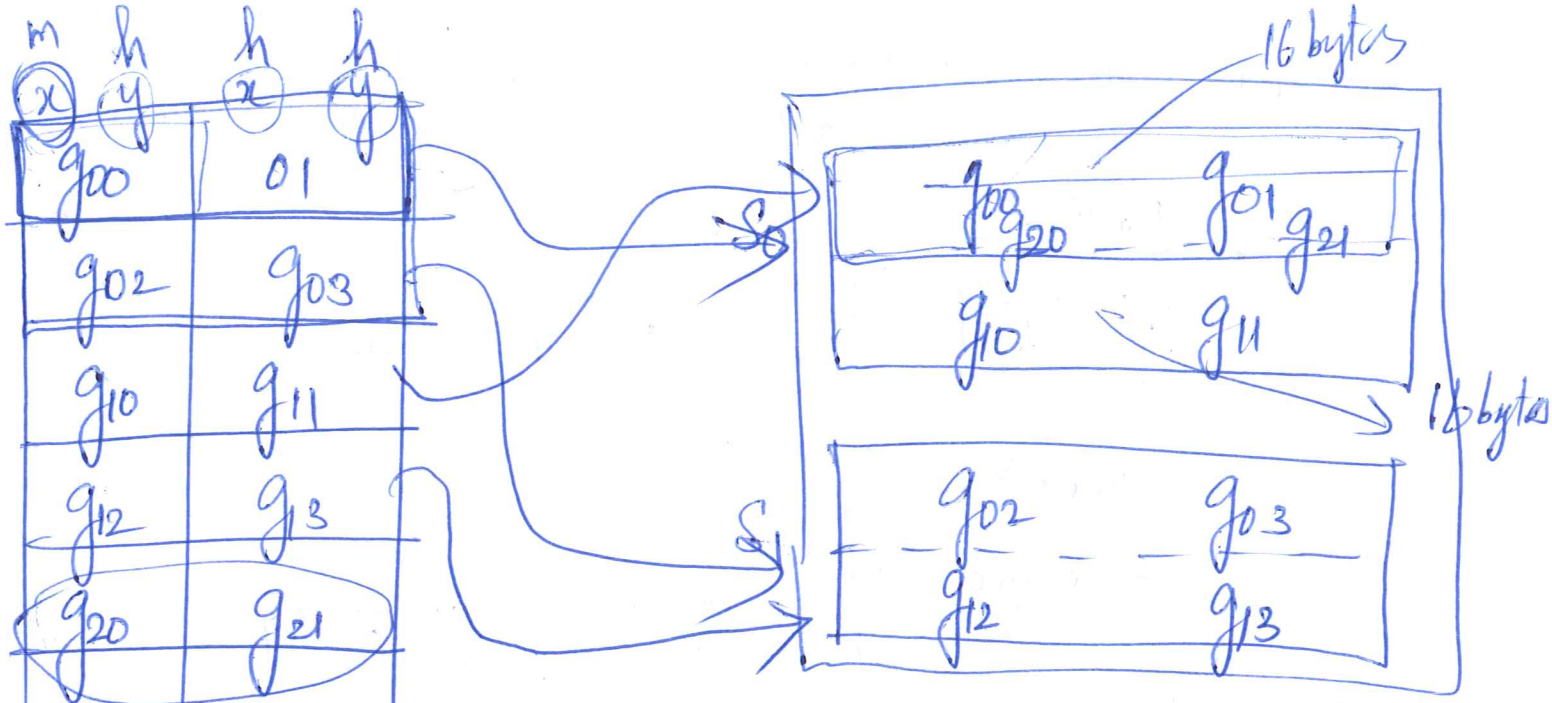
C. Which of these 2 code snippets is better with respect to **cache performance**?

a. Code snippet #1

b. Code snippet #2

Why? (just a single line explanation is sufficient)

_____



C = 64 bytes

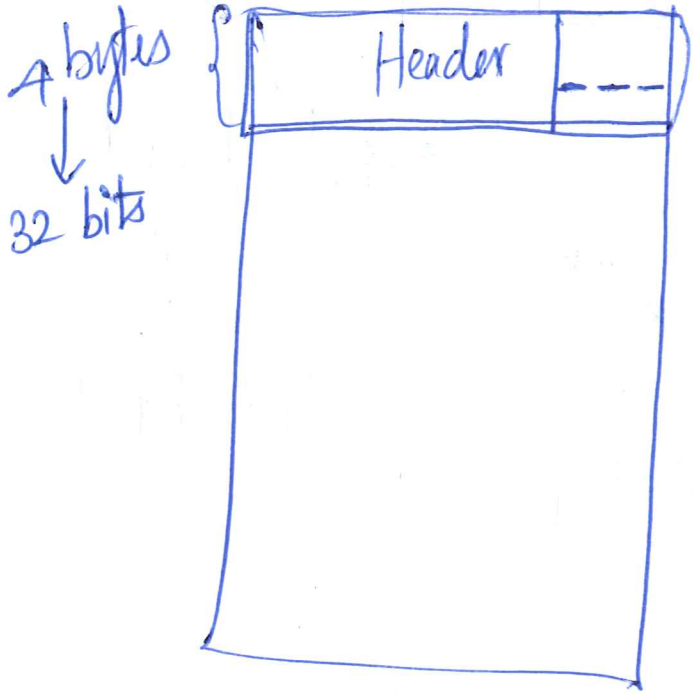
B = 16 bytes

E = 2

LRU

1 1 1 1

$2^4 - 1 = 15$



double word aligned

8

0000 1000

0001 0000

$2^{32} - 1$

$(2^{32} - 1) - 7 = 2^{32} - 8$

001
010
011
100
101
110
111

Pointers in assembly

%eax

0x3004

%ebx

0x3008

0x3004	0x10
0x3008	0x3004

int n = *p;

movl (%eax), %ecx

leal (%eax), %ecx

%ecx

0x10

0x3004

int *p = &n;

$c_1(R_1, R_2, c_2)$

movl (%ebx), %ecx

$c_1 + R_1 + c_2 * R_2$

0x3004

$\underbrace{0X\ FF\ FF\ FF\ F4}_{32\ bits} = 0XF4$

1	1	1	1	0	1	0	0
-128	64	32	16	8	4	2	1

-16

func(int x, int y)

```
pushl %ebp
movl %esp, %ebp
```

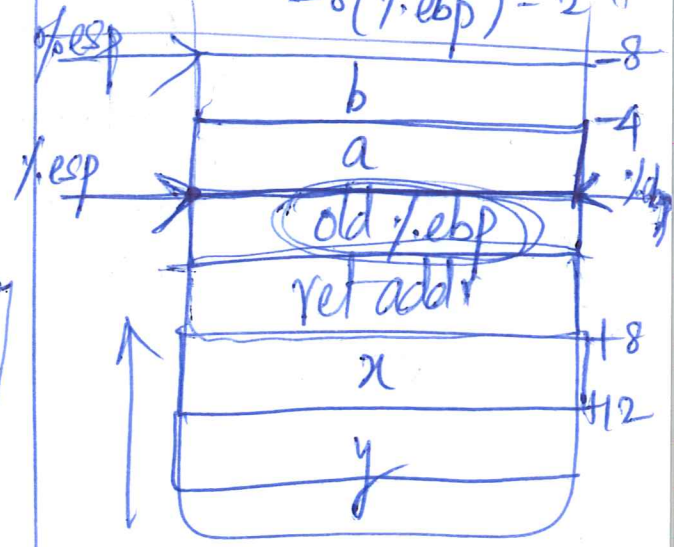
leave

```
movl %ebp, %esp
popl %ebp
```

ret

args → 8(%ebp) - 1st
12(%ebp) - 2nd

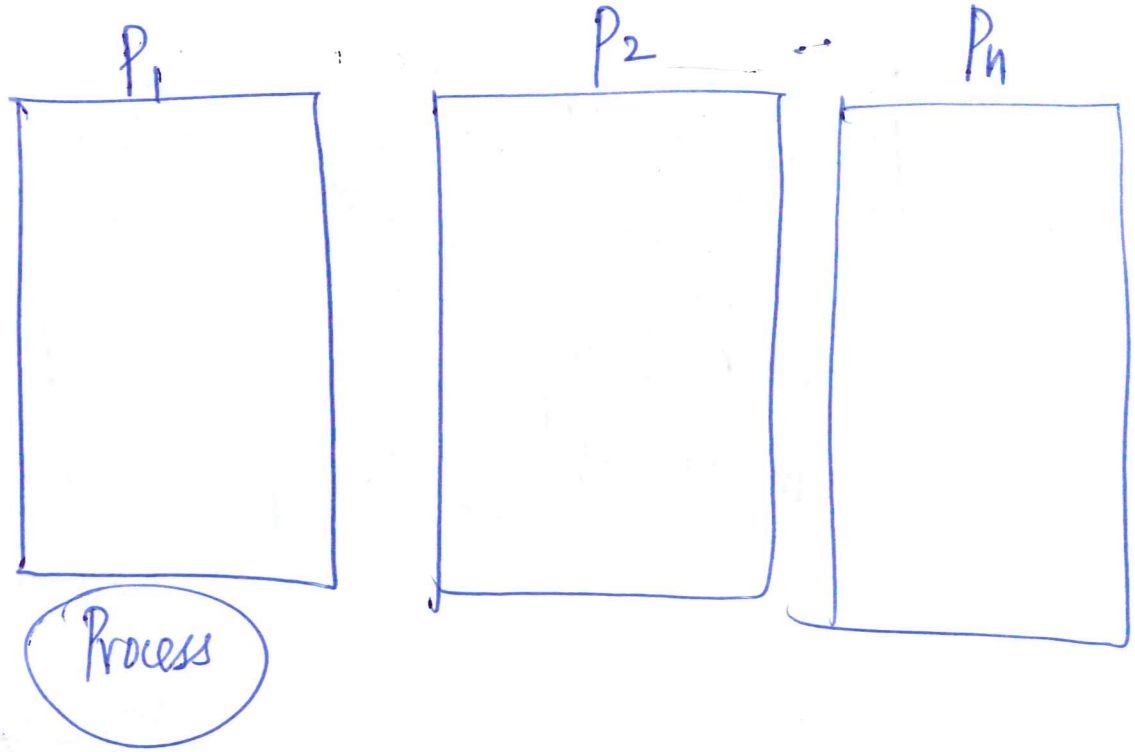
locals → -4(%ebp) - 1st
-8(%ebp) - 2nd

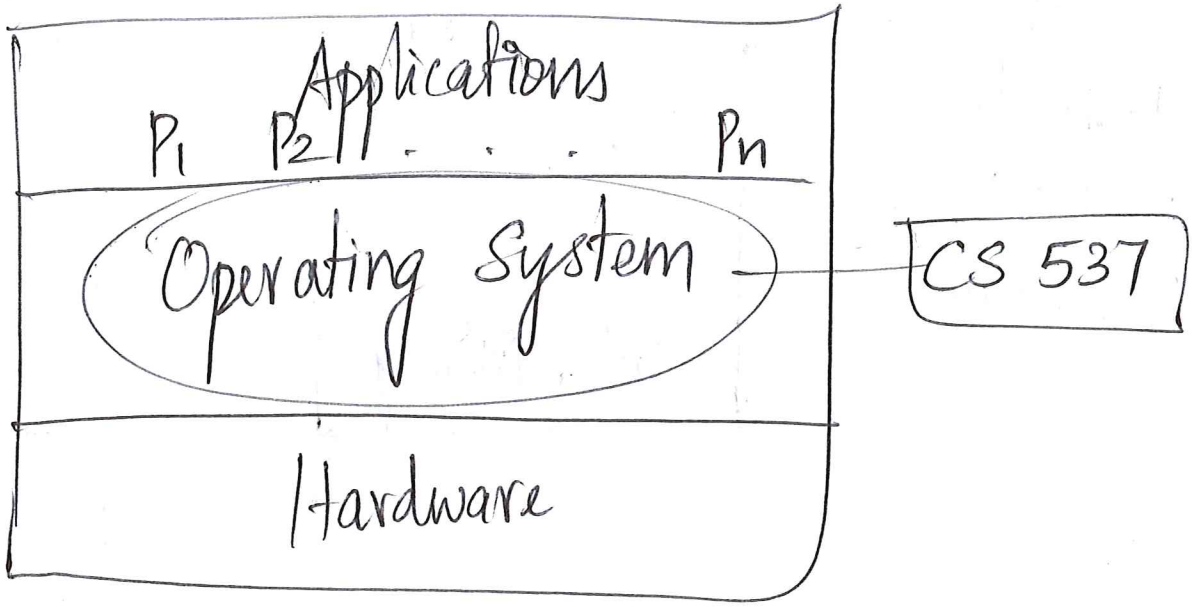


Course Evals

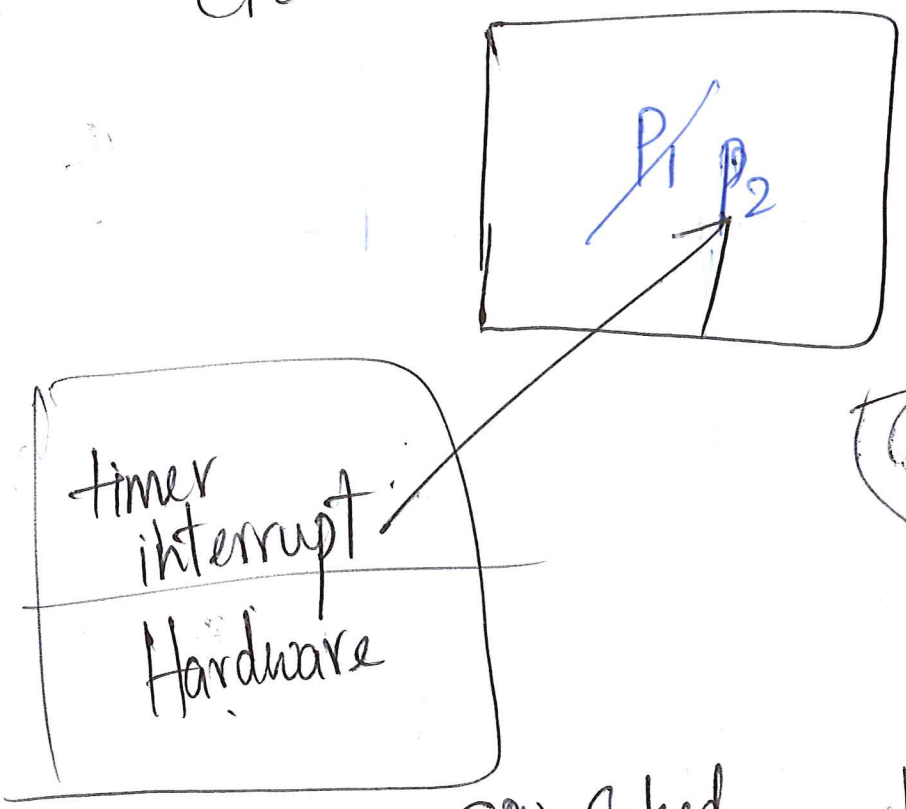
Please complete your UW-Madison course survey evaluations

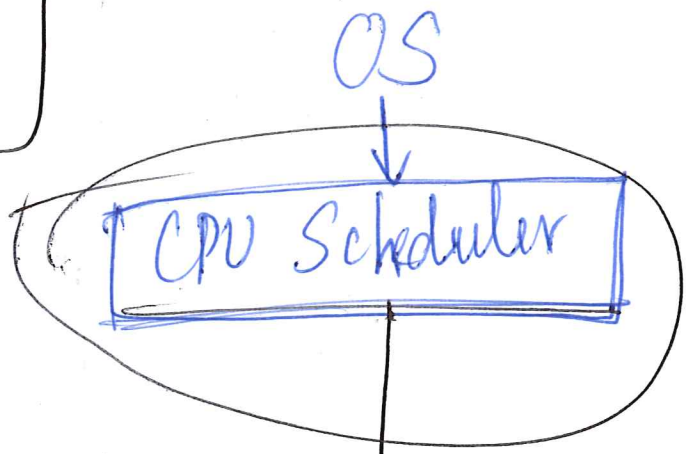
3. What next?

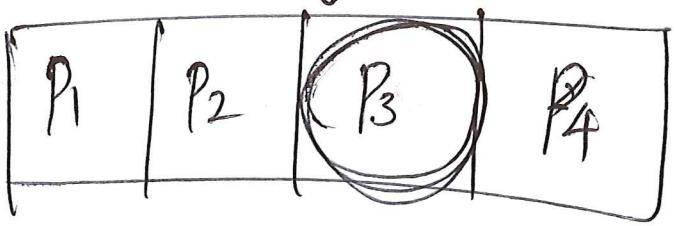




CPU





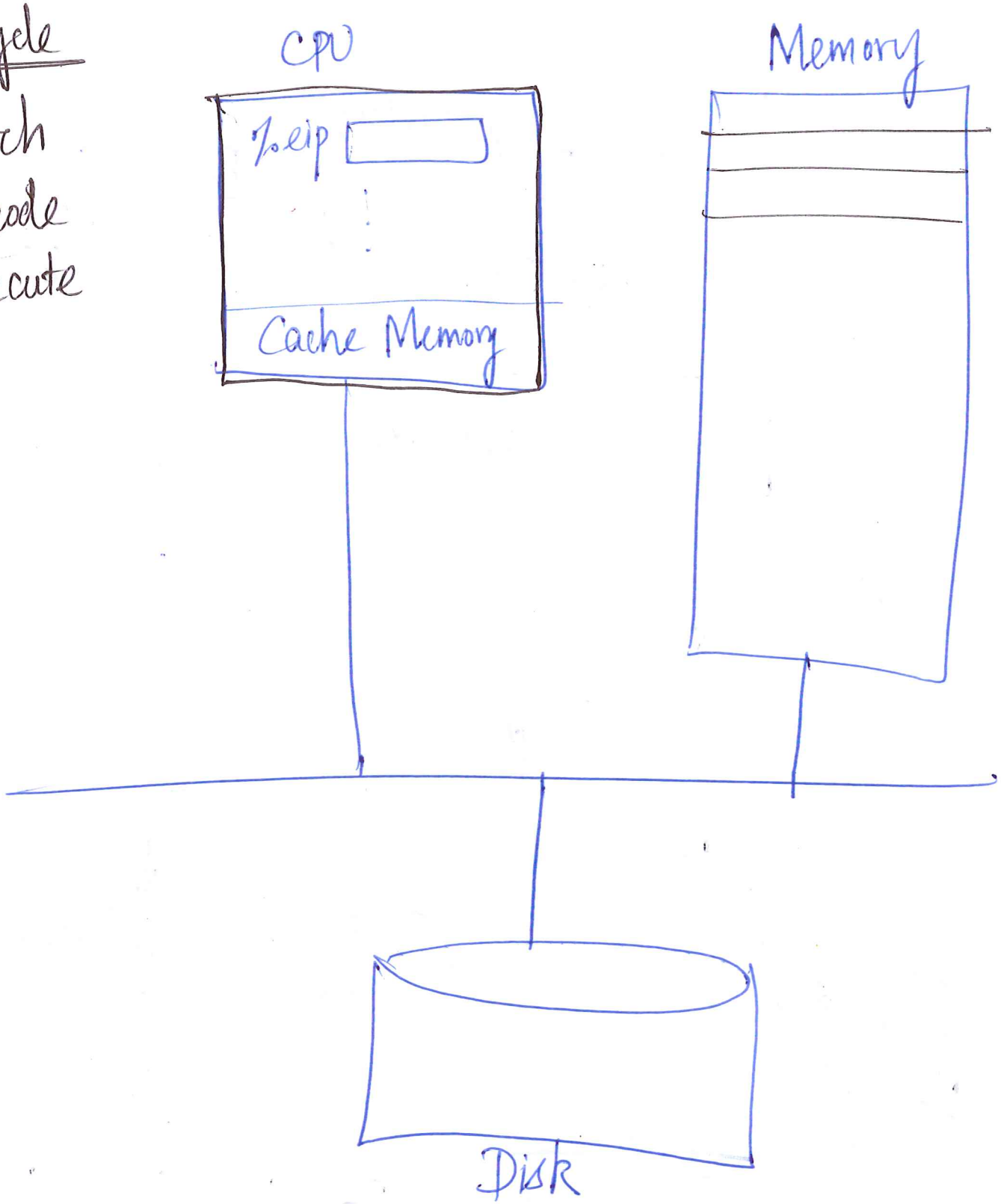


CPU Sched

Mechanisms Policies

cpu cycle

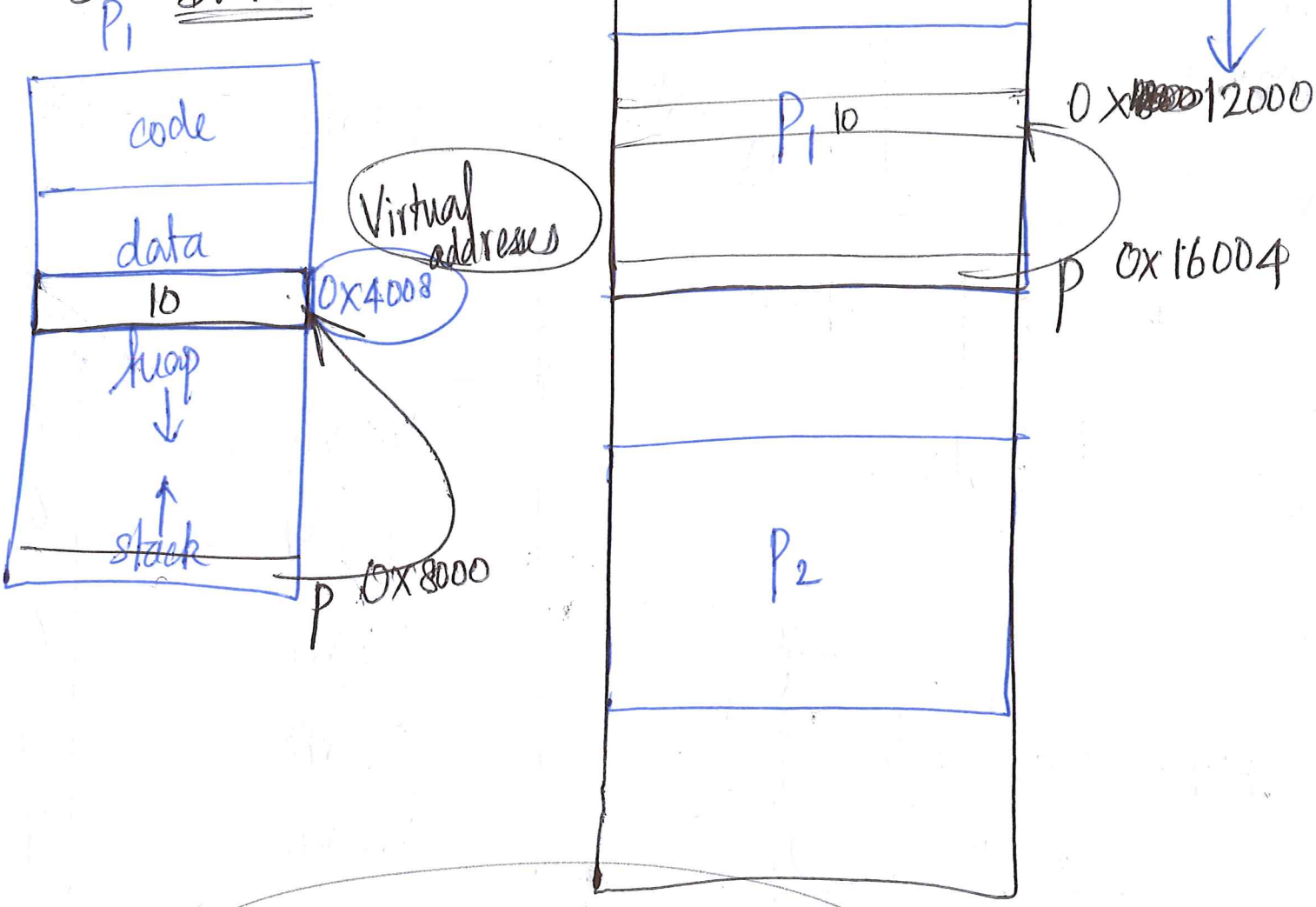
1. fetch
2. decode
3. execute



1. CPU

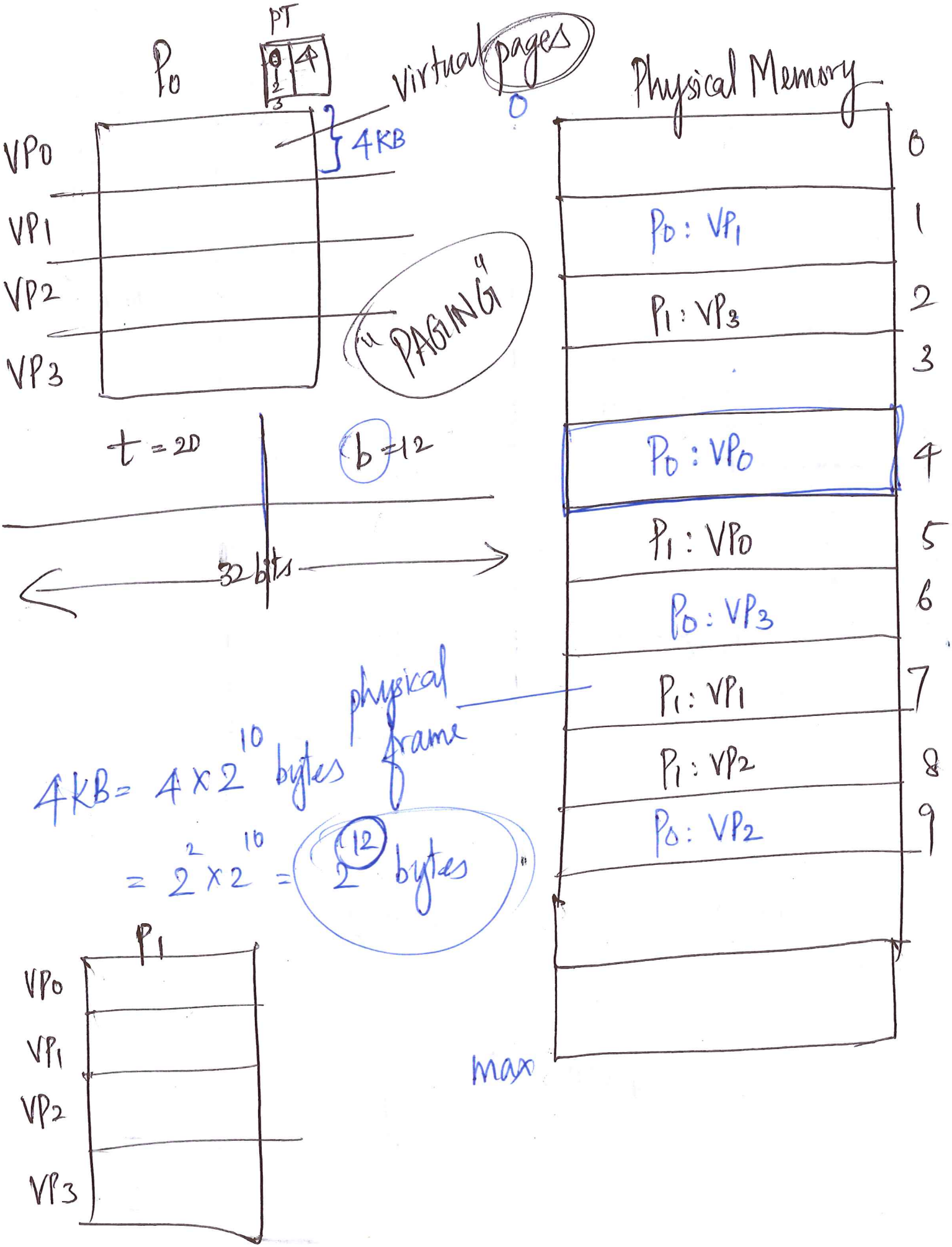
2. Memory

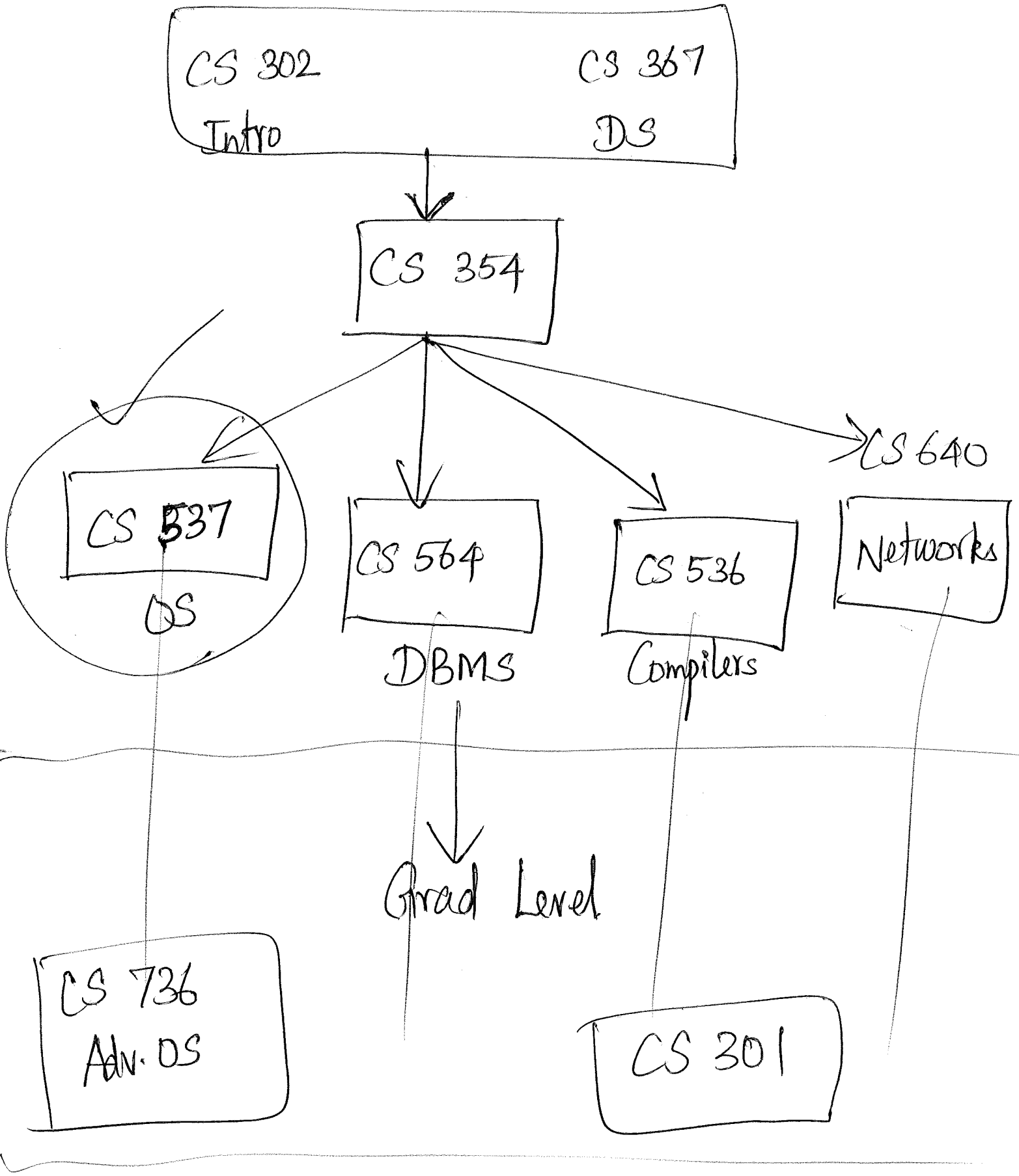
3. DISK



V.A ⟶ P.A.

"Virtual Memory"


PT
P0
0
4
2
3
virtual pages
0
4KB
VP0
VP1
VP2
VP3
"PAGING"
t = 20
b = 12
32 bits
Physical Memory
0
1
2
3
4
5
6
7
8
9
P0 : VP1
P1 : VP3
P0 : VP0
P1 : VP0
P0 : VP3
P1 : VP1
P1 : VP2
P0 : VP2
max
physical frame
4KB = 4 x 2^10 bytes
= 2^2 x 2^10 = 2^12 bytes
P1
VP0
VP1
VP2
VP3


CS 302
Intro
CS 367
DS
CS 354
CS 537
OS
CS 564
DBMS
CS 536
Compilers
CS 640
Networks
Grad Level
CS 736
Adv. OS
CS 301