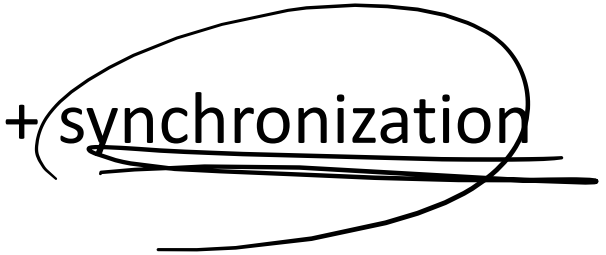
Monitors

Monitor --- Class

- Monitor == data + operation + synchronization



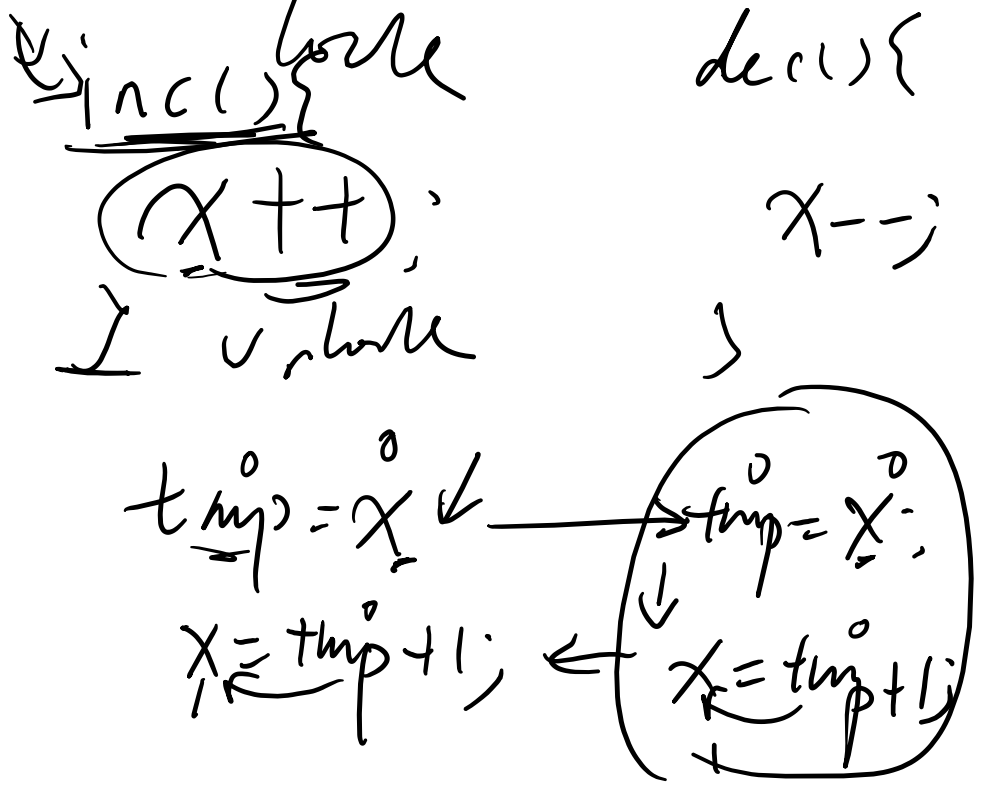
The history of synchronization

- System level resource-access synchronization
- Application level synchronization
- THE, Nucleus, Hoare Monitors, …

Two basic types of synchronization

- Mutual exclusion
  - What it is …?
  - Example
  - How supported by Monitor
- Ordering
  - What it is
  - How supported by Monitor

been completed. Otherwise, if two procedure bodies were in simultaneous execution, the effects on the local variables of the monitor could be chaotic. The proce-



Two basic types of synchronization

- Mutual exclusion
  - What it is
  - How supported by Monitor
- Ordering
  - What it is
  - How supported by Monitor

been completed. Otherwise, if two procedure bodies were in simultaneous execution, the effects on the local variables of the monitor could be chaotic. The proce-

Two basic types of synchronization

- Mutual exclusion
  - What it is
  - How supported by Monitor
- Ordering
  - What it is
  - Example
  - How supported by Monitor

been completed. Otherwise, if two procedure bodies were in simultaneous execution, the effects on the local variables of the monitor could be chaotic. The proce-

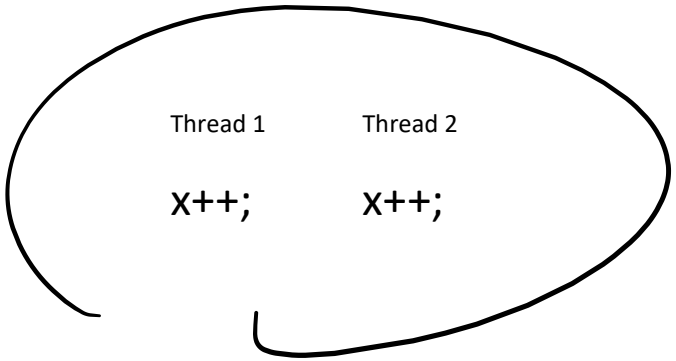
Any dynamic resource allocator will sometimes need to delay a program wishing to acquire a resource which is not currently available, and to resume that program after some other program has released the resource

Semantics of monitor procedures

Semantics of monitor wait & signal

Synchronization types

Be careful about
Global variables &
Heap variables
In multi-threaded programs!

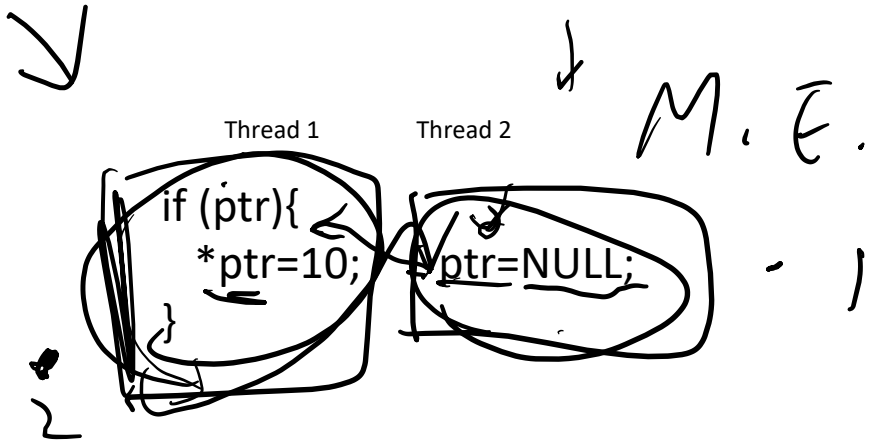


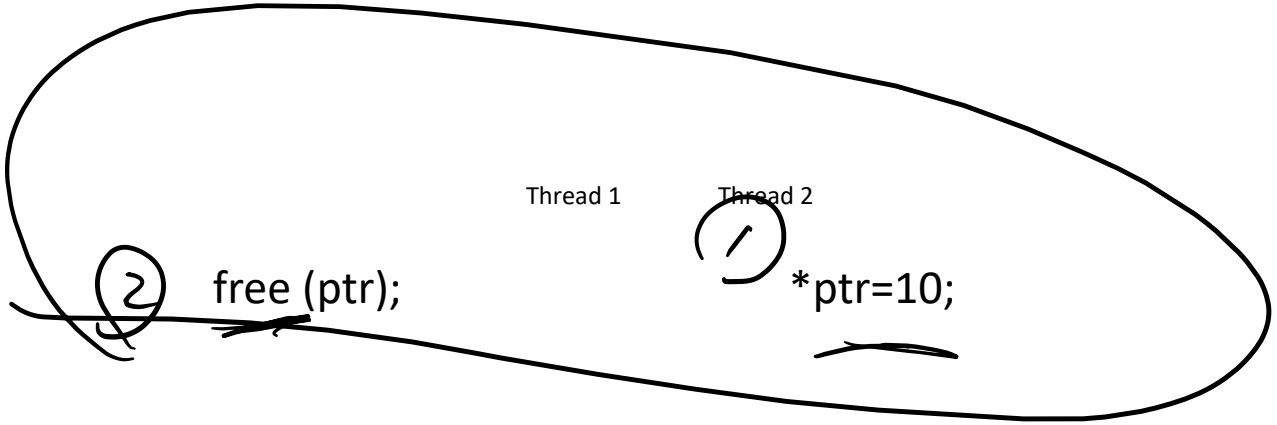
Thread 1 Thread 2

ptr=NULL;

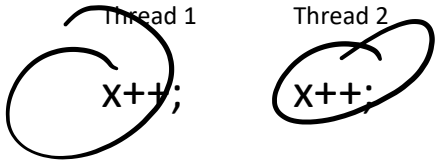
…

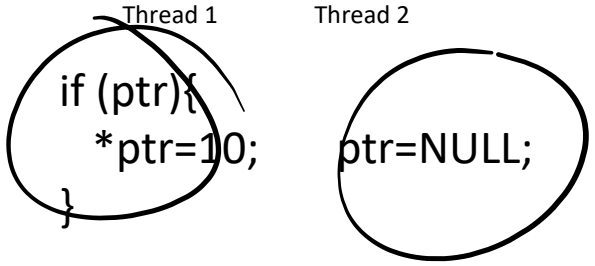
ptr=malloc(sizeof(int)); *ptr=10;





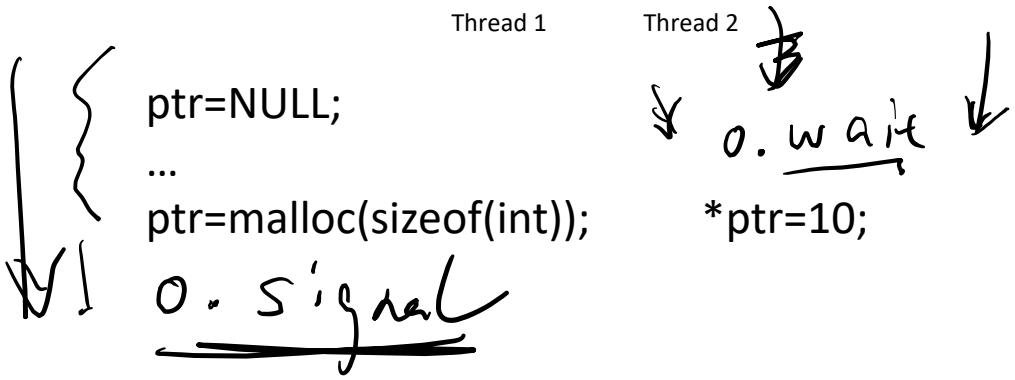
How to enforce mutual exclusion?





How to enforce ordering?

o.wait
o.signal



How to synchronize: Example 1

Thread 1	Thread 2
ptr=NULL;	wait (&CV)
…	
ptr=malloc(sizeof(int));	*ptr=10;
signal (&CV)	

Is this correct?

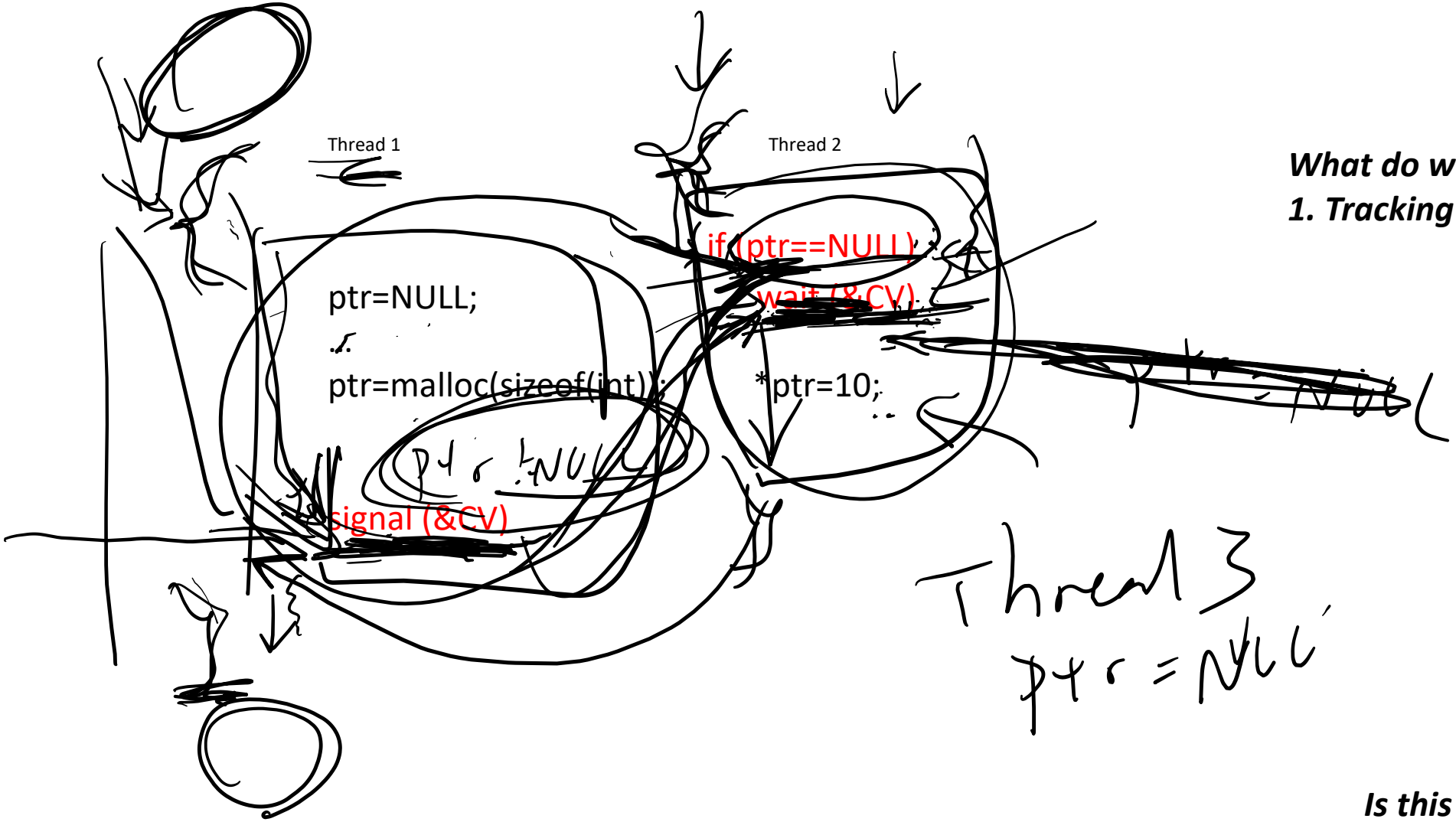
How to synchronize: Example 1

Thread 1	Thread 2
ptr=NULL;	wait (&CV)
…	
ptr=malloc(sizeof(int));	*ptr=10;
signal (&CV)	

What do we need?

1. Tracking if signal already occurred

How to synchronize: Example 1



What do we need?

1. Tracking if signal already occurred

Is this correct?

How to synchronize: Example 1

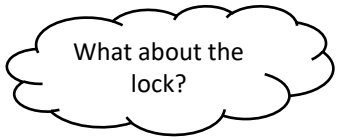
Thread 1	Thread 2
	if (ptr==NULL)
ptr=NULL;	wait (&CV)
…	
ptr=malloc(sizeof(int));	*ptr=10;
signal (&CV);	

What do we need?

1. ***Tracking if signal already occurred***
2. ***A lock!***

How to synchronize: Example 1

Thread 1	Thread 2
	Lock(&L);
	if (ptr==NULL)
ptr=NULL;	wait (&CV)
Lock(&L);	
ptr=malloc(sizeof(int));	*ptr=10;
	Unlock(&L);
signal (&CV);	
Unlock (&L);	



What do we need?

1. ***Tracking if signal already occurred***
2. ***A lock!***

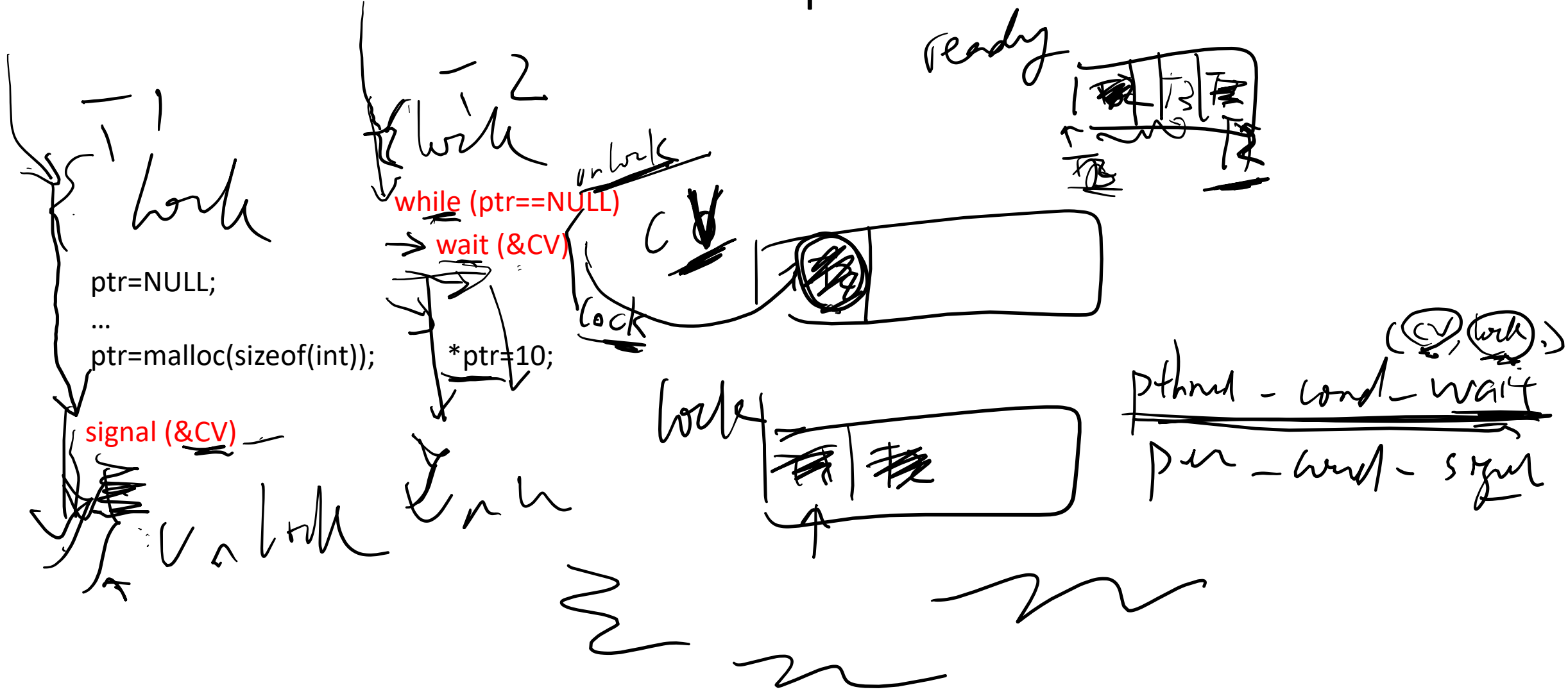
Hoare monitor vs. Mesa monitor

Thread 1	Thread 2
	Lock(&L);
	if (ptr==NULL)
ptr=NULL;	wait (&CV)
Lock(&L);	
ptr=malloc(sizeof(int));	*ptr=10;
	Unlock(&L);
signal (&CV);	
Unlock (&L);	

Mesa monitor and its implementation

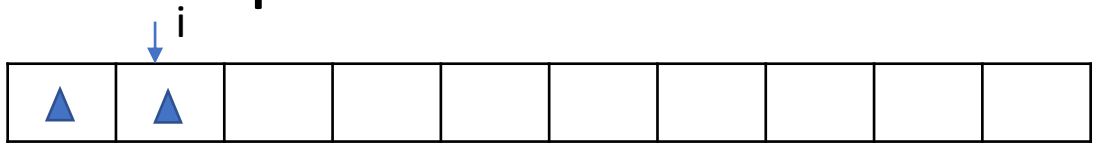
```
ptr=NULL;
...
ptr=malloc(sizeof(int));
signal (&CV)
```

```
while (ptr==NULL)
  wait (&CV)
*ptr=10;
```



Today’s …

Example 2: producer – consumer queue



Producer

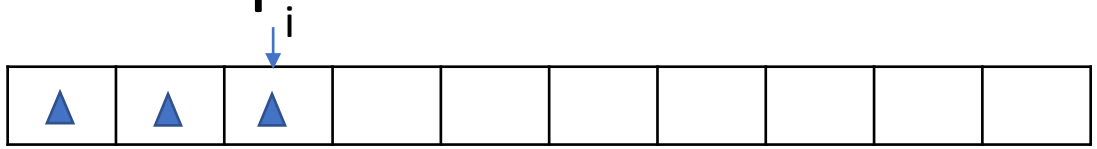
```
Item=produce();
Q[++i]=Item;
```

Consumer

```
Item = Q[i--];
Consume (Item);
```

What problems?
Mutual exclusion?
Ordering?

Example 2: producer – consumer queue



Producer

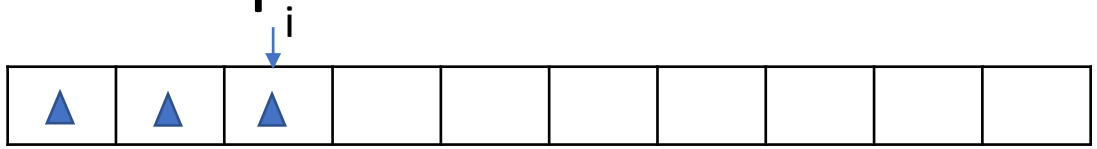
```
Item=produce();
Lock (&L);
Q[++i]=Item;
signal(&CV1);
Unlock(&L);
```

Consumer

```
Lock(&L);
While(…)
    wait(&CV1);
Item = Q[i--];
Unlock(&L);
Consume (Item);
```

What problems?
Mutual exclusion?
Ordering?

Example 2: producer – consumer queue



Producer

```
Item=produce();
Lock (&L);
While(…)
    wait(&CV2);
Q[++i]=Item;
signal(&CV1);
Unlock(&L);
```

Consumer

```
Lock(&L);
While(…)
    wait(&CV1);
Item = Q[i--];
signal(&CV2);
Unlock(&L);
Consume (Item);
```

What problems?
Mutual exclusion?
Ordering?
Deadlock?

Other types of synchronization

Synchronization primitives

	Mutual Exclusion	Ordering
C (pthread library)	pthread_mutex_lock	pthread_cond_signal pthread_cond_wait
Java (monitor)	synchronized methods	wait, notify
semaphore	**sem_post** **sem_wait**	**sem_wait** **sem_post**

Thread 1 Thread 2

x++; x++;

Thread 1 Thread 2

```
if (ptr){
  *ptr=10;     ptr=NULL;
}
```

Thread 1 Thread 2

ptr=NULL;
…
ptr=malloc(sizeof(int)); *ptr=10;

Thread 1 Thread 2

free (ptr); *ptr=10;

Our 4/15 lecture ends here

Semantics & mechanisms of semaphore

- Semaphore{
  int cnt;
  Queue blocked_q;
}

- Sem_init

- Sem_post
  - Cnt + +
  - Wake a waiting queue up, if any

- Sem_wait
  - If cnt>0
    - Cnt- -
  - Else
    - Wait in blocked_q
    - Cnt--

How to enforce mutual exclusion?

Sem_init(??)

X+ +;

X+ +;

Public semaphore

Sem_init(??)

Sem_wait

X+ +;

Sem_post

Sem_wait

X+ +;

Sem_post

Private semaphore: how to make B waits for A

Sem_init(??)

B

A

Semaphore vs. Condition variable

semaphore

- Has a counter

Condition variable

- No counter

Semaphore vs. Condition variable

semaphore

- Has a counter
- Sem_post
  - What happens when no one is waiting?

Condition variable

- No counter
- Signal
  - What happens when no one is waiting?

Semaphore vs. Condition variable

semaphore

- Has a counter
- Sem_post
  - What happens when no one is waiting?
- Sem_wait
  - Work by itself
  - No lock associated

Condition variable

- No counter
- Signal
  - What happens when no one is waiting?
- Wait
  - Need a while(flag)
  - Need a lock
  - Release lock while waiting

Example 1

Semaphore

Thread 1

```
sem_init(&s, 0);

ptr=NULL;
…
ptr=malloc(sizeof(int));

sem_post(&s);
```

Thread 2

```
sem_wait(&s);
*ptr=10;
```

Condition variable

Thread 1

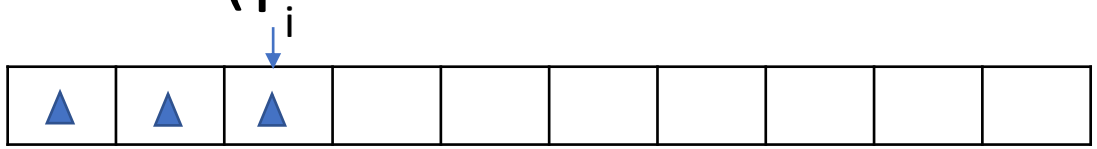
```
ptr=NULL;
…
ptr=malloc(sizeof(int));

Lock(&L);
signal (&CV);
Unlock (&L);
```

Thread 2

```
Lock(&L);
if (ptr==NULL)
  wait (&CV)
Unlock(&L);
*ptr=10;
```

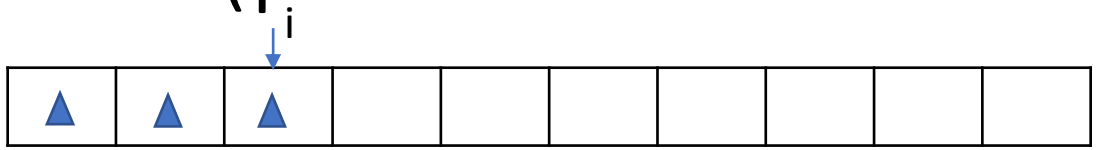
Example 2 (producer—consumer queue)



- Producer
- Consumer

What problems?
Mutual exclusion?
Ordering?
Deadlock?

Example 2 (producer—consumer queue)



- Producer

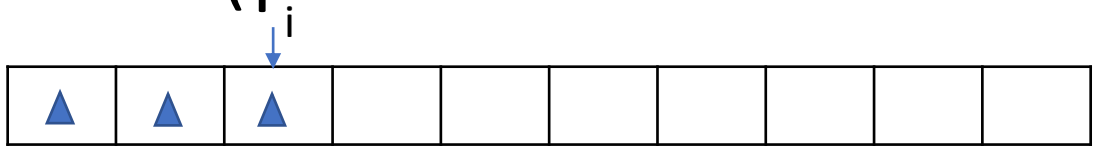
```
Item=produce();
Lock (&L);
While(…)
    wait(&CV2);
Q[++i]=Item;
signal(&CV1);
Unlock(&L);
```

- Consumer

```
Lock(&L);
While(…)
    wait(&CV1);
Item = Q[i--];
signal(&CV2);
Unlock(&L);
Consume (Item);
```

What problems?
Mutual exclusion?
Ordering?
Deadlock?

Example 2 (producer—consumer queue)



- Producer

Sem_init(&cell, MAX); sem_init(&item, 0);

```
Item=produce();
Sem_wait(&cell);
Sem_wait(&mutex);
Q[++i]=Item;
Sem_post(&mutex);
Sem_post(&item);
```

- Consumer

```
Sem_wait(&item);
Sem_wait(&mutex);
Item = Q[i--];
Sem_signal(&mutex);
Sem_signal(&cell);
```

What problems?
Mutual exclusion?
Ordering?
Deadlock?

What is used today?

Other issues raised by Mesa monitor paper

- Deadlocks