COS 318: Operating Systems

Semaphores, Monitors and Condition Variables

Kai Li

Computer Science Department
Princeton University

(http://www.cs.princeton.edu/courses/cos318/)

Today's Topics

- ◆ Semaphores
- ◆ Monitors
- ◆ Mesa-style monitor and its idioms
- ◆ Barriers

Semaphores (Dijkstra, 1965)

- ◆ Initialization
  - ● Initialize a semaphore s
- ◆ P (or Down or Wait or "Probeer") definition
  - ● Atomic operation
  - ● Wait for semaphore to become positive and then decrement
    ```
    P(s){
        while (s <= 0)
            ;
        s--;
    }
    ```
- ◆ V (or Up or Signal or "Verhoog") definition
  - ● Atomic operation
  - ● Increment semaphore by 1
    ```
    V(s){
        s++;
    }
    ```

Bounded Buffer with Semaphores

```
producer() {                    consumer() {
  while (1) {                     while (1) {
    produce an item                 P(fullCount);
    P(emptyCount);
                                    P(mutex);
    P(mutex);                       take an item from buffer
    put the item in buffer          V(mutex);
    V(mutex);
                                    V(emptyCount);
    V(fullCount);                   consume the item
  }                               }
}                               }
```

◆ Initialization: emptyCount = N; fullCount = 0
◆ Are **P(mutex)** and **V(mutex)** necessary?

Example: Interrupt Handler

Device thread

```
...
Acquire(m);
...


Release(m);
...
```

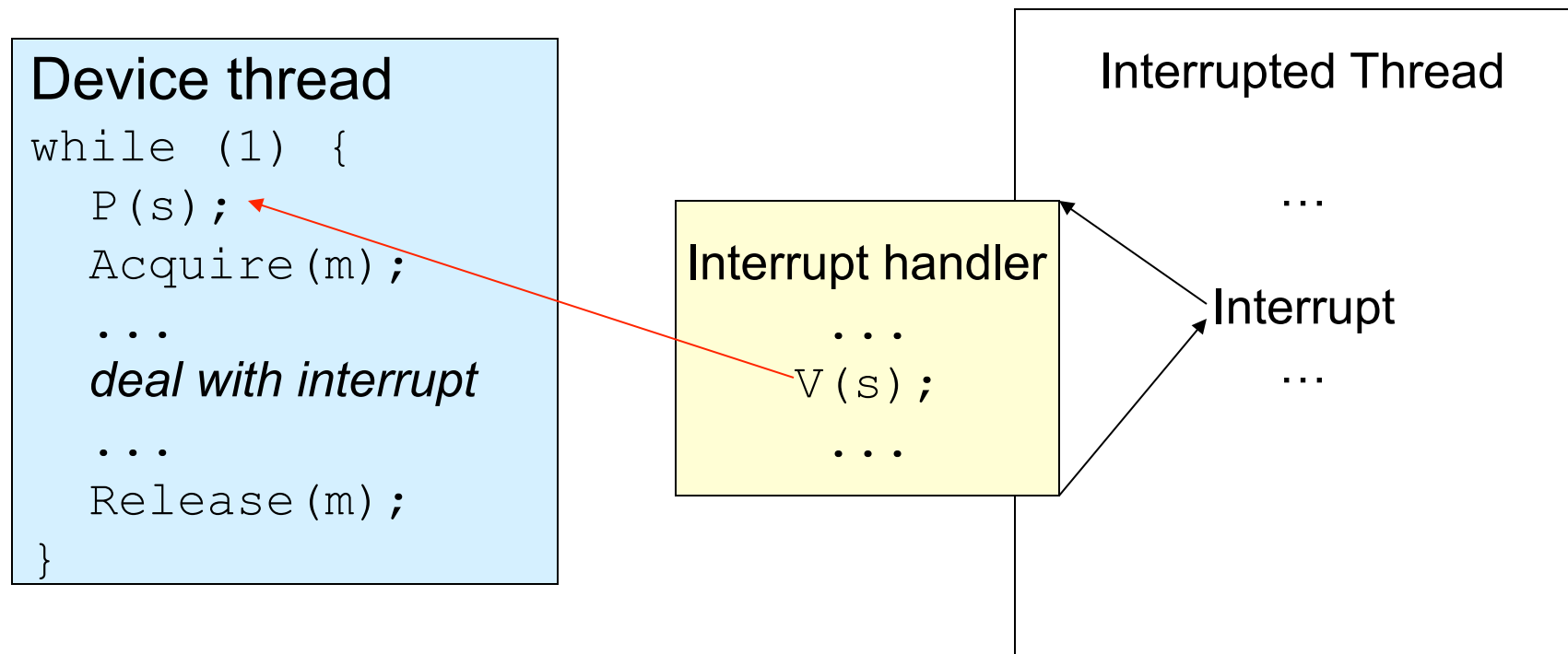**Interrupt handler**

```
...
Acquire(m);
...

Release(m);
...
```
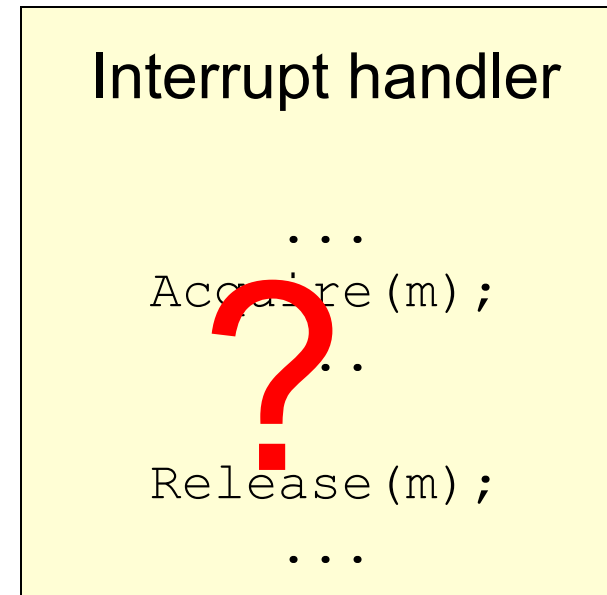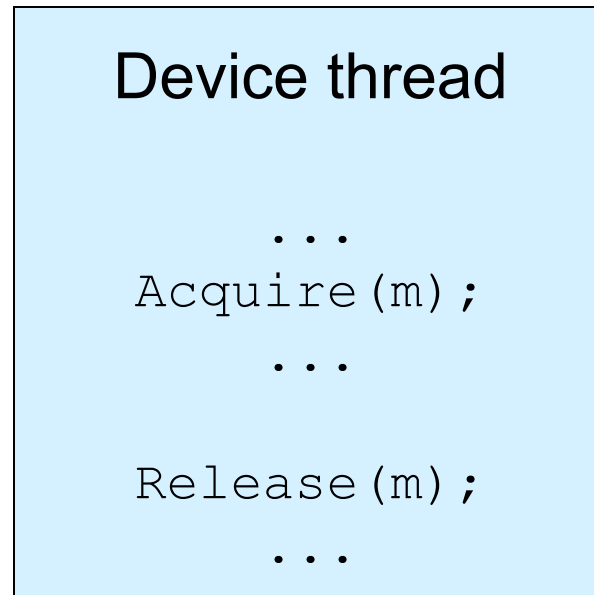
◆ A device thread works with an interrupt handler

◆ What to do with shared data?

◆ What if "m" is held by another thread or by itself?

Use Semaphore

```
Init(s,0);
```

Device thread
```
while (1) {
  P(s);
  Acquire(m);
  ...
  deal with interrupt
  ...
  Release(m);
}
```

Interrupt handler
```
  ...
  V(s);
  ...
```

Interrupted Thread
```
  …
  Interrupt
  …
```

Semaphores Are Not Always Convenient

◆ A shared queue has Enqueue and Dequeue:

```
Enqueue(q, item)              Dequeue(q)
{                             {
  Acquire(mutex);               Acquire(mutex);
  put item into q;              take an item from q;
  Release(mutex);               Release(mutex);
}                               return item;
                             }
```
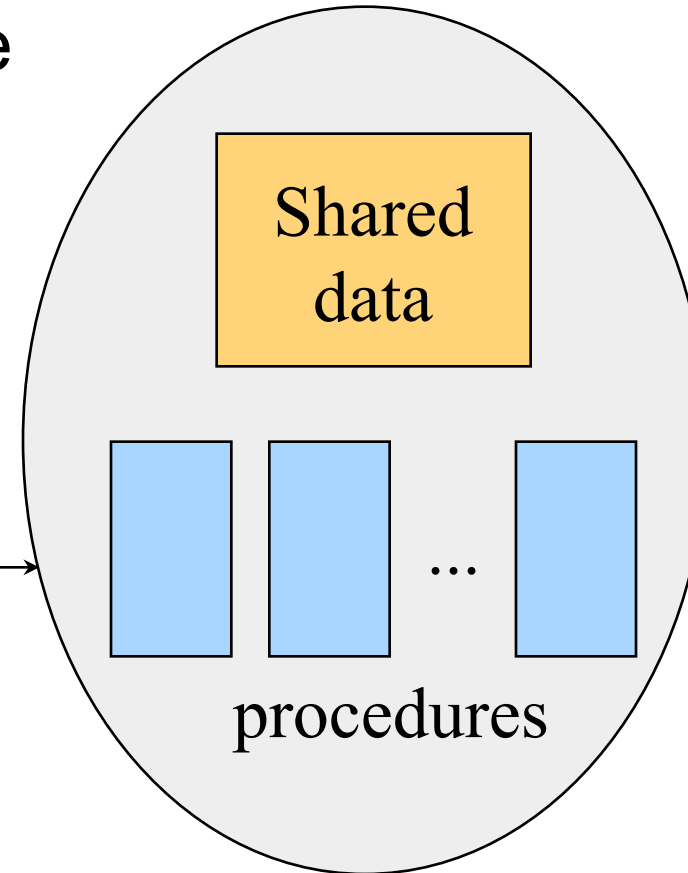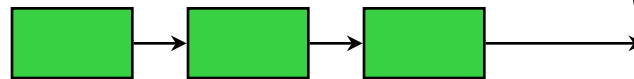
◆ What if we want **Dequeue(q)** to block until **q** is not empty?
  ● It is a consumer and producer problem

◆ Semaphores are difficult to use
  ● Orders are important

Monitor: Hide Mutual Exclusion

◆ Brinch-Hansen (73), Hoare (74)

◆ Procedures are mutual exclusive

Queue of waiting processes
trying to enter the monitor

Shared
data

... procedures

Condition Variables in A Monitor

◆ Wait(condition)
- Block on "condition"

◆ Signal(condition)
- Wakeup a blocked process on "condition"

Queues associated with x, y conditions

x

y

Shared data

Entry queue

procedures

...
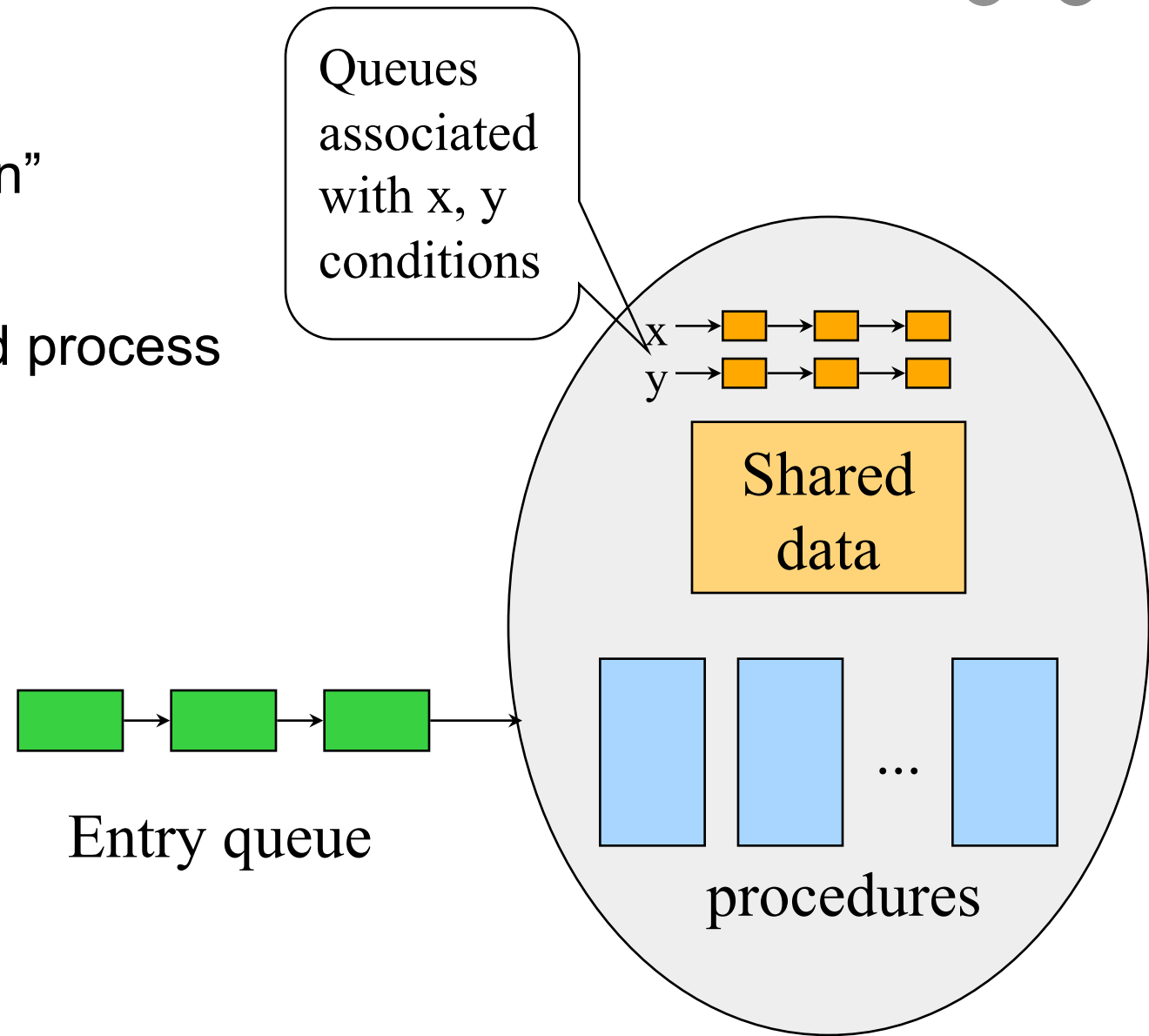
Producer-Consumer with Monitors

```
procedure Producer
begin
  while true do
  begin
    produce an item
    ProdCons.Enter();
  end;
end;

procedure Consumer
begin
  while true do
  begin
    ProdCons.Remove();
    consume an item;
  end;
end;
```

```
monitor ProdCons
  condition full, empty;

  procedure Enter;
  begin
    if (buffer is full)
      wait(full);
    put item into buffer;
    if (only one item)
      signal(empty);
  end;

  procedure Remove;
  begin
    if (buffer is empty)
      wait(empty);
    remove an item;
    if (buffer was full)
      signal(full);
  end;
```

Hoare's Signal Implementation

◆ Run the signaled thread immediately and suspend the current one (Hoare)

◆ What if the current thread has more things to do?

```
  if (only one item)
      signal(empty);
  something else
 end;
```

```
monitor ProdCons
  condition full, empty;

  procedure Enter;
  begin
    if (buffer is full)
      wait(full);
    put item into buffer;
    if (only one item)
      signal(empty);
  end;


  procedure Remove;
  begin
    if (buffer is empty)
      wait(empty);
    remove an item;
    if (buffer was full)
      signal(full);
  end;
```

Hansen's Signal Implementation

- ◆ Signal must be the last statement of a monitor procedure

- ◆ Exit the monitor

- ◆ Any issue with this approach?

```
monitor ProdCons
  condition full, empty;

  procedure Enter;
  begin
    if (buffer is full)
      wait(full);
    put item into buffer;
    if (only one item)
      signal(empty);
  end;

  procedure Remove;
  begin
    if (buffer is empty)
      wait(empty);
    remove an item;
    if (buffer was full)
      signal(full);
  end;
```

Mesa Signal Implementation

◆ Continues its execution

```
if (only one item)
    signal(empty);
something else
end;
```

● B. W. Lampson and D. D. Redell, "Experience with Processes and Monitors in Mesa," Communiction of the ACM, 23(2):105-117. 1980.

◆ This is easy to implement!

◆ Issues?

Mesa Style "Monitor" (Birrell's Paper)

- ◆ Associate a condition variable with a mutex

- ◆ Wait(mutex, condition)
  - Atomically unlock the mutex and enqueued on the condition variable (block the thread)
  - Re-lock the lock when it is awaken

- ◆ Signal(condition)
  - No-op if there is no thread blocked on the condition variable
  - Wake up at least one if there are threads blocked

- ◆ Broadcast(condition)
  - Wake up all waiting threads

Consumer-Producer with Mesa-Style Monitor

```
static count = 0;
static Cond full, empty;
static Mutex lock;
```

```
Enter(Item item) {
   Acquire(lock);
   if (count==N)
     Wait(lock, full);
   insert item into buffer
   count++;
   if (count==1)
     Signal(empty);
   Release(lock);
}
```

```
Remove(Item item) {
   Acquire(lock);
   if (!count)
     Wait(lock, empty);
   remove item from buffer
   count--;
   if (count==N-1)
     Signal(full);
   Release(lock);
}
```

Any issues with this?

Consumer-Producer with Mesa-Style Monitor

```
static count = 0;
static Cond full, empty;
static Mutex lock;

Enter(Item item) {                  Remove(Item item) {
   Acquire(lock);                      Acquire(lock);
   while (count==N)                    while (!count)
     Wait(lock, full);                   Wait(lock, empty);
   insert item into buffer            remove item from buffer
   count++;                            count--;
   if (count==1)                       if (count==N-1)
     Signal(empty);                      Signal(full);
   Release(lock);                      Release(lock);
}                                   }
```

The Programming Idiom

◆ Waiting for a resource

```
Acquire( mutex );
while ( no resource )
    wait( mutex, cond );
...
(use the resource)
...
Release( mutex);
```

◆ Make a resource available

```
Acquire( mutex );
...
(make resource available)
...
Signal( cond );
/* or Broadcast( cond );
Release( mutex);
```

Revisit the Motivation Example

```
Enqueue(Queue q,
        Item item) {

    Acquire(lock);

    insert an item to q;

    Signal(Empty);
    Release(lock);
}
```

```
Item GetItem(Queue q) {
    Item item;

    Acquire( lock );
    while ( q is empty )
        Wait( lock, Empty);

        remove an item;

    Release( lock );
    return item;
}
```

◆ Does this work?

Condition Variables Primitives

- **Wait(mutex, cond)**
  - Enter the critical section (min busy wait)
  - Release mutex
  - Put my TCB to cond's queue
  - Call scheduler
  - Exit the critical section
    . . . (blocked)

  - Waking up:
    - Acquire mutex
    - Resume

- **Signal(cond)**
  - Enter the critical section (min busy wait)
  - Wake up a TCB in cond's queue
  - Exit the critical section

More on Mesa-Style Monitor

◆ Signaler continues execution

◆ Waiters simply put on ready queue

  ● Must reevaluate the condition

◆ No constraints on when the waiting thread must run

◆ No constrains on signaler

  ● Can execute after signal call (Hansen's cannot)

 ● Do not need to relinquish control to awaken thread/process

Evolution of Monitors

- ◆ Brinch-Hansen (73) and Hoare Monitor (74)
  - Concept, but no implementation
  - Requires Signal to be the last statement (Hansen)
  - Requires relinquishing CPU to signaler (Hoare)
- ◆ Mesa Language (77)
  - Monitor in language, but signaler keeps mutex and CPU
  - Waiter simply put on ready queue, with no special priority
- ◆ Modula-2+ (84) and Modula-3 (88)
  - Explicit LOCK primitive
  - Mesa-style monitor
- ◆ Pthreads (95)
  - Started standard effort around 1989
  - Defined by ANSI/IEEE POSIX 1003.1 Runtime library
- ◆ Java threads
  - James Gosling in early 1990s without threads
  - Use most of the Pthreads primitives
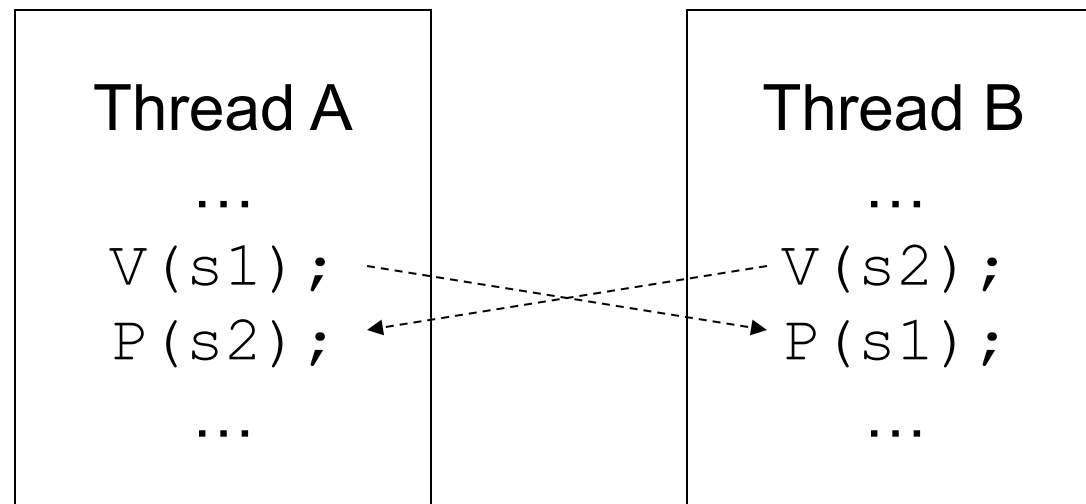
Example: A Simple Barrier

◆ Thread A and Thread B want to meet at a particular point

◆ Then both go forward

◆ How would you program this with a monitor?

Thread A   Thread B
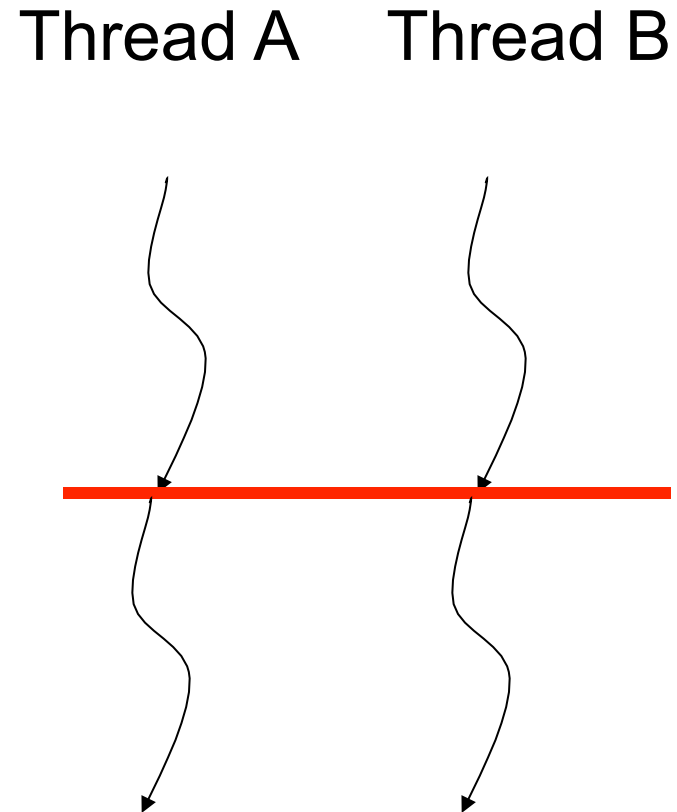
Using Semaphores as A Barrier

- ◆ Use two semaphore?

```
init(s1, 0);
init(s2, 0);
```

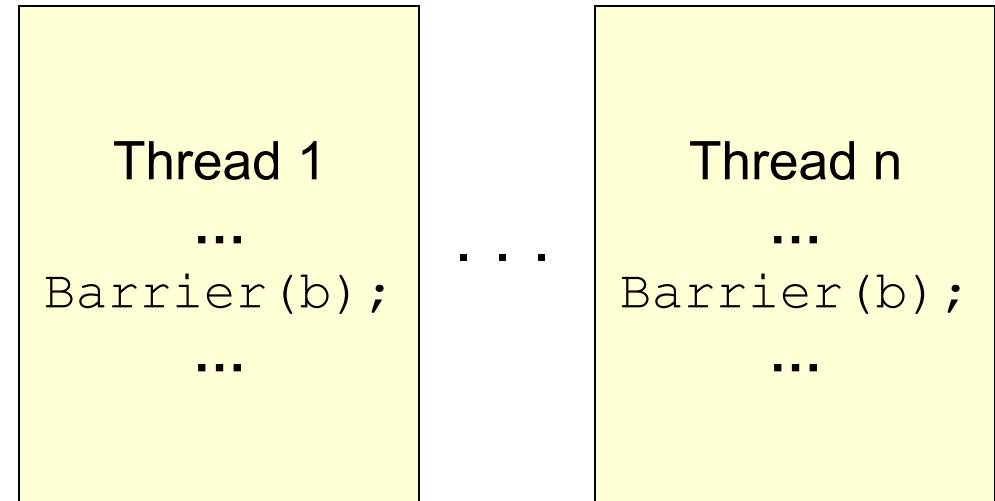| Thread A	Thread B
...	...
V(s1);	V(s2);
P(s2);	P(s1);
...	...

- ◆ What about more than two threads?
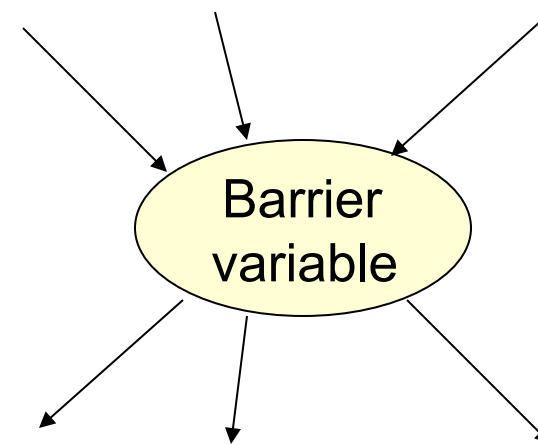
Barrier Primitive

◆ **Functions**

- Take a barrier variable
- Broadcast to n-1 threads
- When barrier variable has reached n, go forward

◆ **Hardware support on some parallel machines**

- Multicast network
- Counting logic
- User-level barrier variables

```
Thread 1
...
Barrier(b);
...
```

. . .

```
Thread n
...
Barrier(b);
...
```

Barrier variable

Equivalence

- ◆ **Semaphores**
  - Good for signaling
  - Not good for mutex because it is easy to introduce a bug
- ◆ **Monitors**
  - Good for scheduling and mutex
  - Maybe costly for a simple signaling

Summary

- ◆ Semaphores
- ◆ Monitors
- ◆ Mesa-style monitor and its idiom
- ◆ Barriers