Condition Variables

Questions answered in this lecture:

How can threads enforce ordering across operations?

How can thread_join() be implemented?

How can condition variables be used to support producer/consumer apps?

Concurrency Objectives

- **Mutual exclusion (e.g., A and B don't run at the same time)**
  - solved with locks

- **Ordering (e.g., B runs after A does something)**
  - solved with condition variables and semaphores

Ordering Example: Join

```
pthread_t p1, p2;
Pthread_create(&p1, NULL, mythread, "A");
Pthread_create(&p2, NULL, mythread, "B");
// join waits for the threads to finish
Pthread_join(p1, NULL);
Pthread_join(p2, NULL);
printf("main: done\n [balance: %d]\n [should: %d]\n",
        balance, max*2);
return 0;
```

how to implement join()?

Condition Variables

- **Condition Variable: queue of waiting threads**
- **B waits for a signal on CV before running**
  - wait(CV, …)
- **A sends signal to CV when time for B to run**
  - signal(CV, …)

Condition Variables

- **wait(cond_t *cv, mutex_t *lock)**
  - assumes the lock is held when wait() is called
  - puts caller to sleep + releases the lock (atomically)
  - when awoken, reacquires lock before returning
  - This complexity stems from the desire to prevent certain race conditions from occurring when a thread is trying to put itself to sleep

- **signal(cond_t *cv)**
  - wake a single waiting thread (if >= 1 thread is waiting)
  - if there is no waiting thread, just return, doing nothing

Join Implementation: Attempt 1

Parent:

```
void thread_join() {
   Mutex_lock(&m);        // x
   Cond_wait(&c, &m);     // y
   Mutex_unlock(&m);      // z
}
```

Child:

```
void thread_exit() {
   Mutex_lock(&m);        // a
   Cond_signal(&c);       // b
   Mutex_unlock(&m);      // c
}
```

Example schedule:

Parent:	x	y				z
Child:			a	b	c	

Works!

Join Implementation: Attempt 1

Parent:

```
void thread_join() {
   Mutex_lock(&m);        // x
   Cond_wait(&c, &m);     // y
   Mutex_unlock(&m);      // z
}
```

Child:

```
void thread_exit() {
   Mutex_lock(&m);        // a
   Cond_signal(&c);       // b
   Mutex_unlock(&m);      // c
}
```

Can you construct ordering that does not work?

Example broken schedule:

Parent: x y

Child: a b c

Parent waits forever!

Rule of Thumb 1

- Keep state in addition to CVs
- CVs are used to signal threads when state changes
- If state is already as needed, thread doesn't wait for a signal!

Join Implementation: Attempt 2

Parent:

```
void thread_join() {
   Mutex_lock(&m);        // w
   if (done == 0)         // x
      Cond_wait(&c, &m); // y
   Mutex_unlock(&m);     // z
}
```

Child:

```
void thread_exit() {
   done = 1;        // a
   Cond_signal(&c);    // b
}
```

Fixes previous broken ordering:

Parent: w x y z

Child: a b

Join Implementation: Attempt 2

Parent:

```
void thread_join() {
  Mutex_lock(&m);         // w
  if (done == 0)          // x
    Cond_wait(&c, &m); // y
  Mutex_unlock(&m);     // z
}
```

Child:

```
void thread_exit() {
  done = 1;        // a
  Cond_signal(&c);    // b
}
```

Can you construct ordering that does not work?

Parent: w x y ... sleep forever ...

Child: a b

Join Implementation: Correct

Parent:

```
void thread_join() {
   Mutex_lock(&m);        // w
   if (done == 0)         // x
      Cond_wait(&c, &m); // y
   Mutex_unlock(&m);     // z
}
```

Child:

```
void thread_exit() {
   Mutex_lock(&m);       // a
   done = 1;             // b
   Cond_signal(&c);      // c
   Mutex_unlock(&m);    // d
}
```

Parent: w x y z

Child: a b c

hold the lock when calling signal or wait, and you will always be in good shape

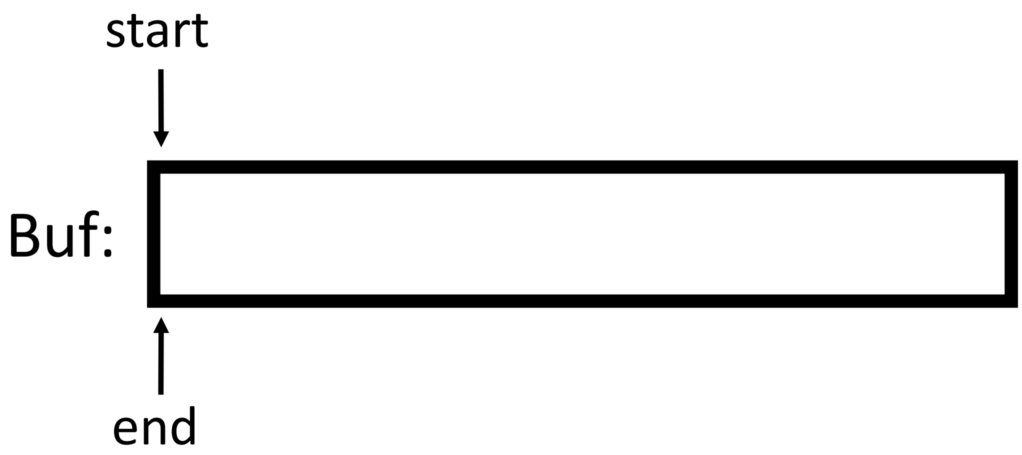
- Use mutex to ensure no race between interacting with state and wait/signal

Producer/Consumer Problem

Example: UNIX Pipes

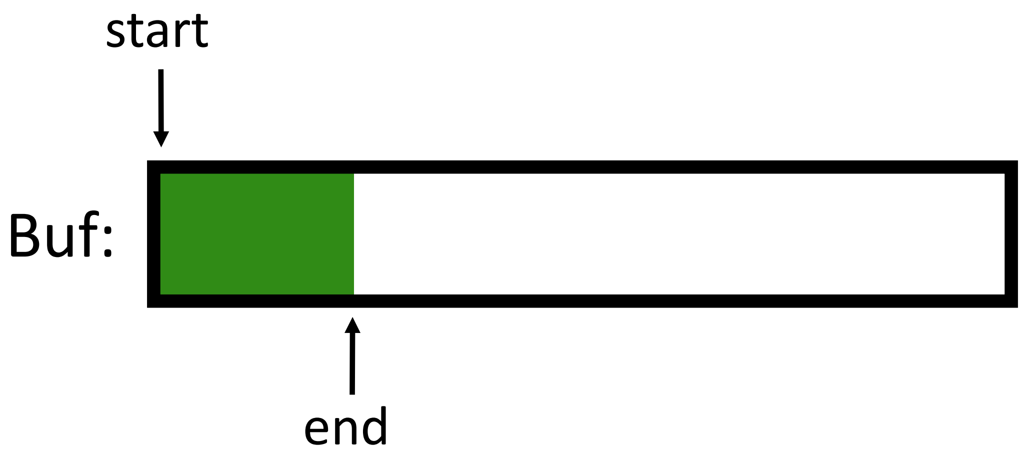
- **A pipe may have many writers and readers**
- **Internally, there is a finite-sized buffer**
- **Writers add data to the buffer**
  - Writers have to wait if buffer is full
- **Readers remove data from the buffer**
  - Readers have to wait if buffer is empty

Example: UNIX Pipes

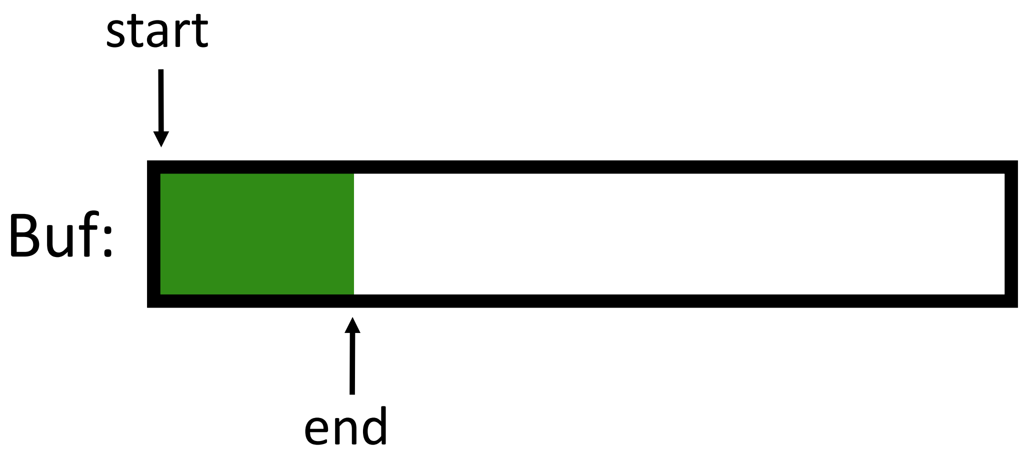


Example: UNIX Pipes

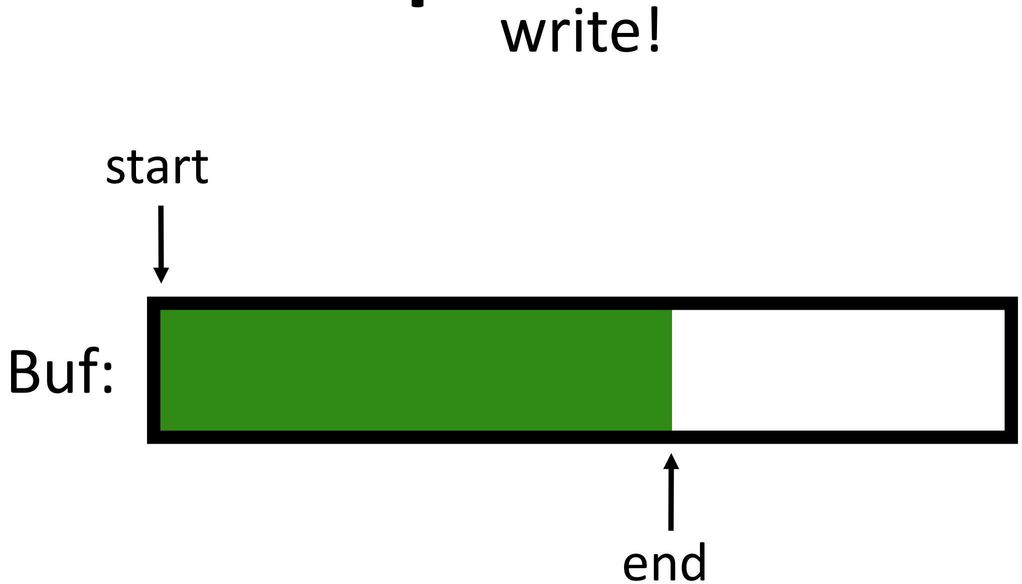
write!



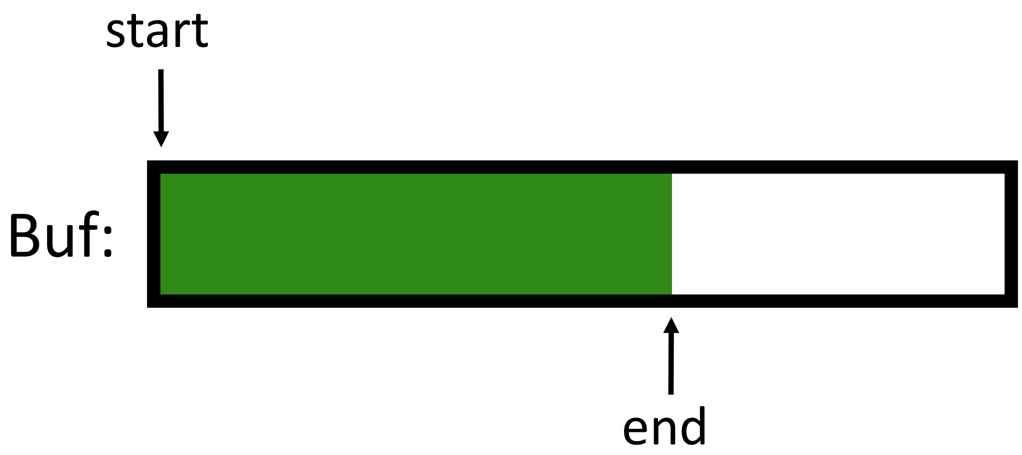
Example: UNIX Pipes



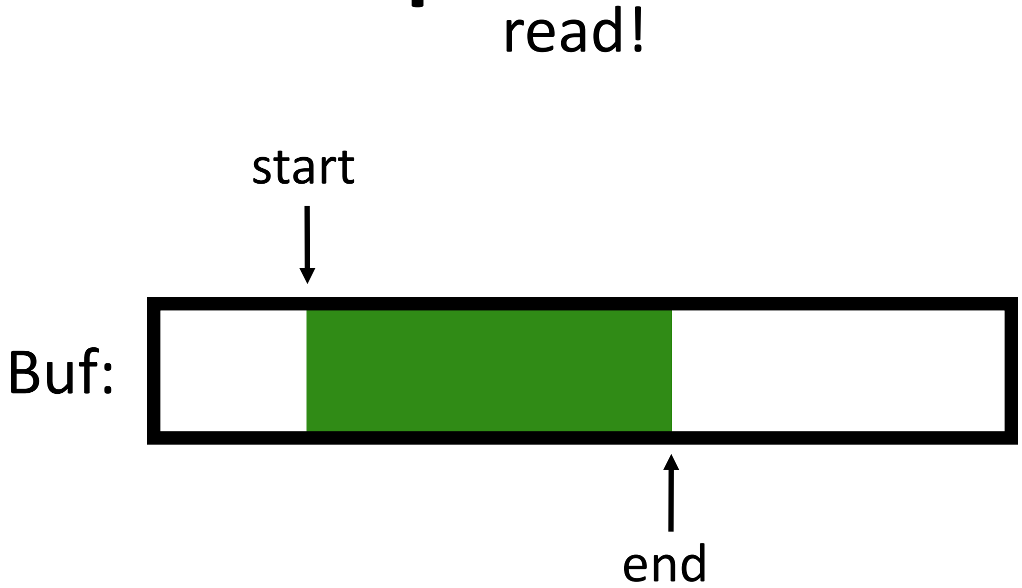
Example: UNIX Pipes



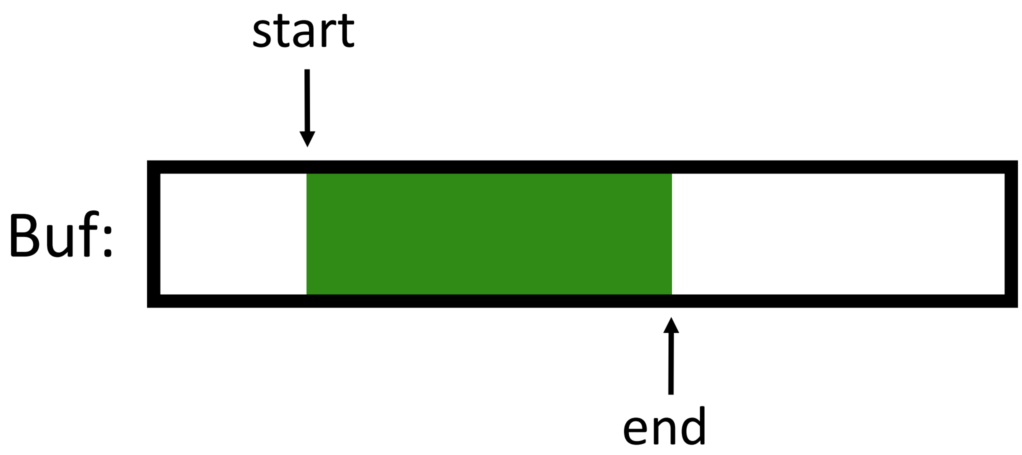
Example: UNIX Pipes



Example: UNIX Pipes

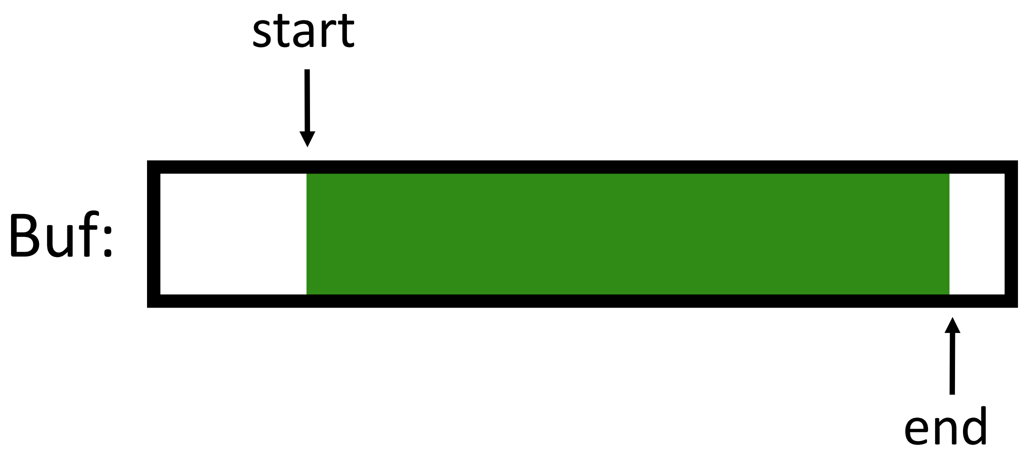


Example: UNIX Pipes

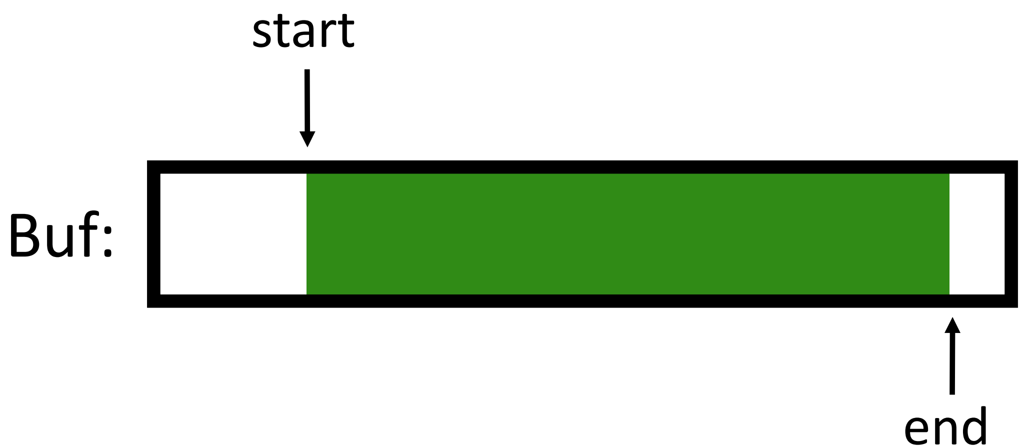


Example: UNIX Pipes

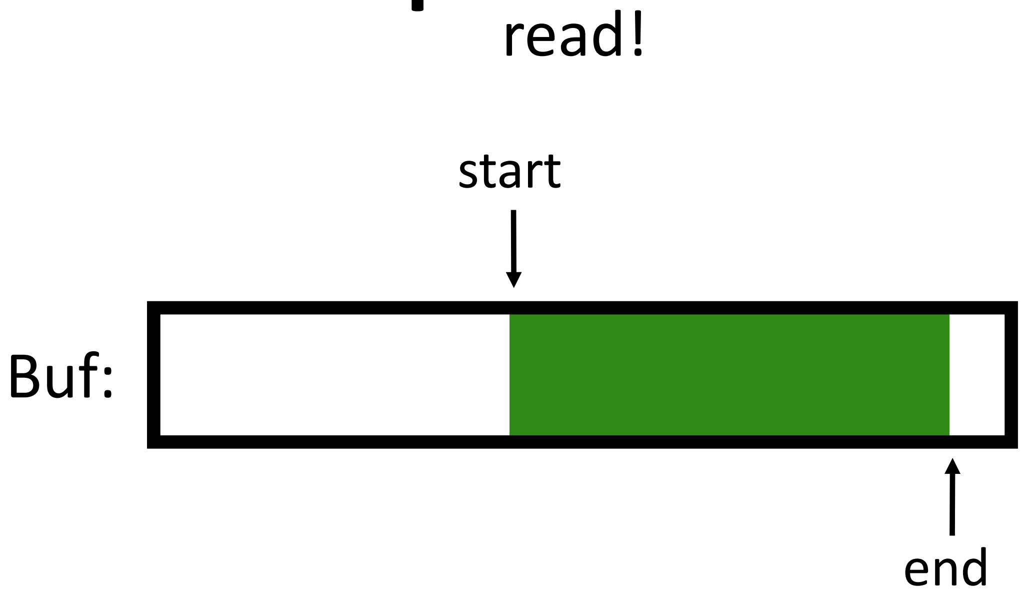
write!



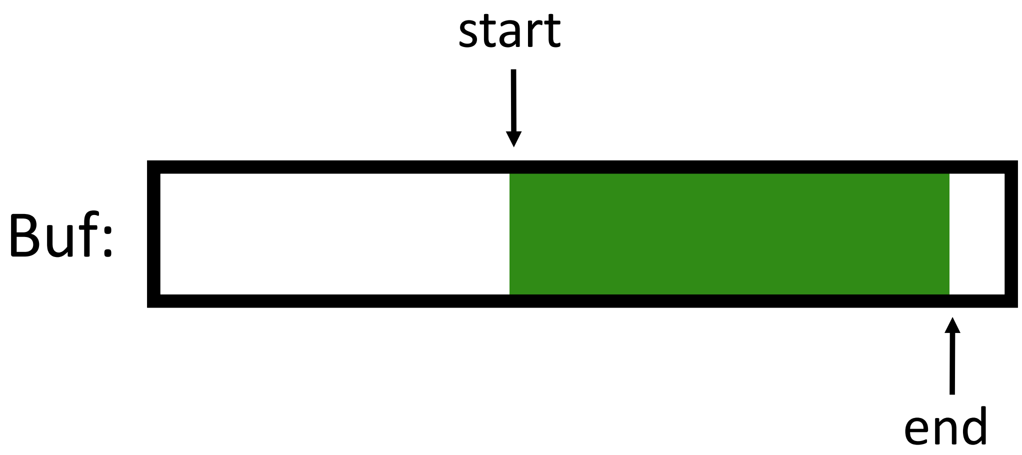
Example: UNIX Pipes



Example: UNIX Pipes

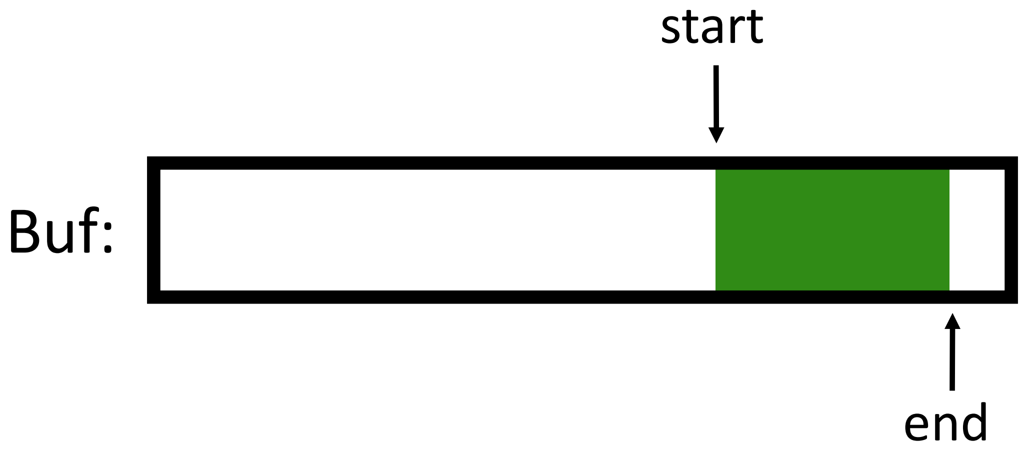


Example: UNIX Pipes

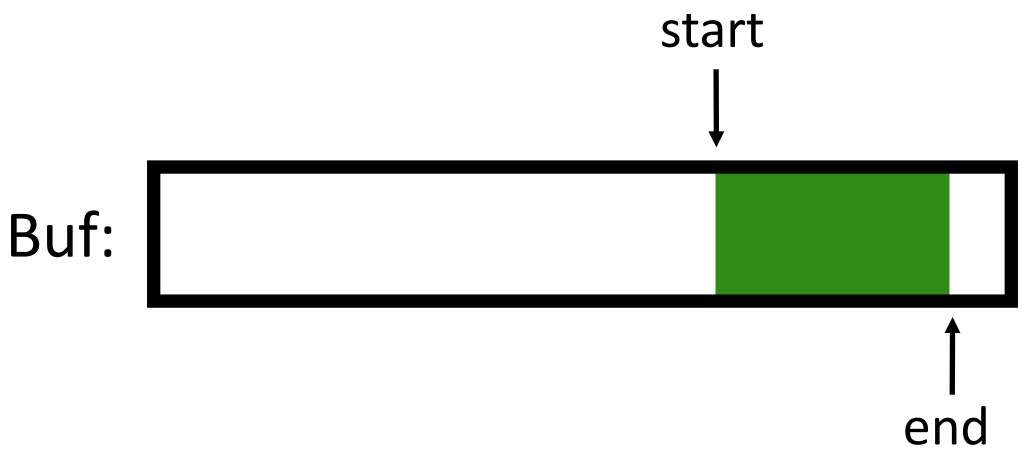


Example: UNIX Pipes

read!

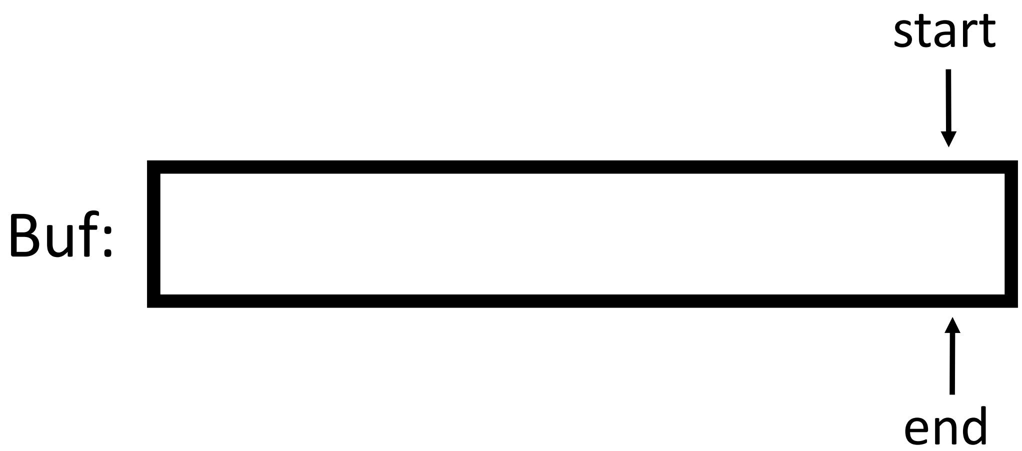


Example: UNIX Pipes



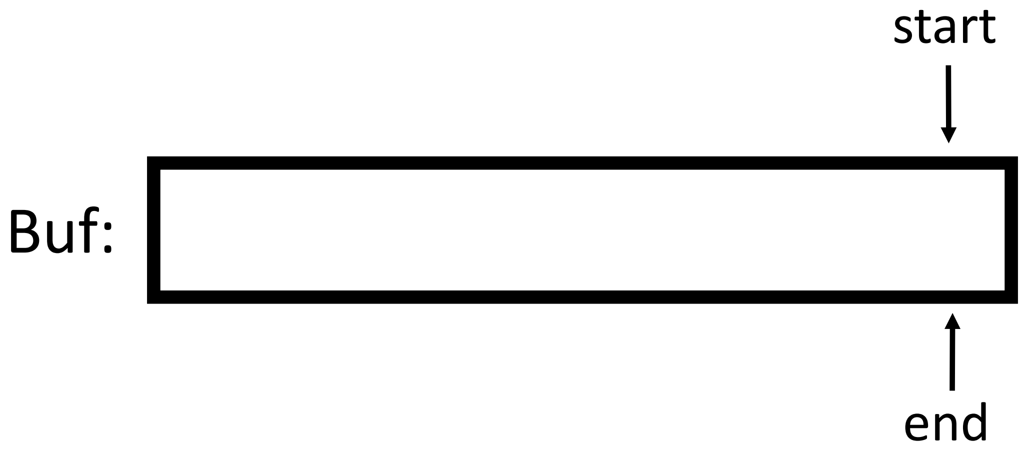
Example: UNIX Pipes

read!



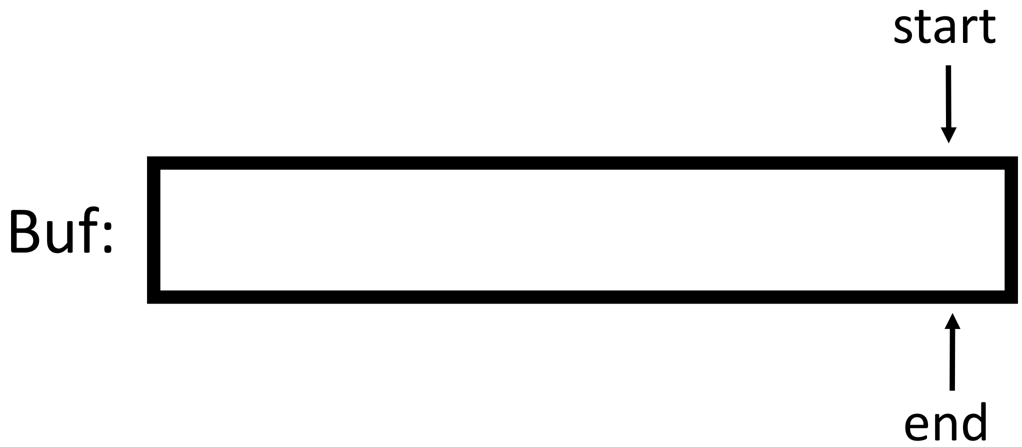
Example: UNIX Pipes

read!



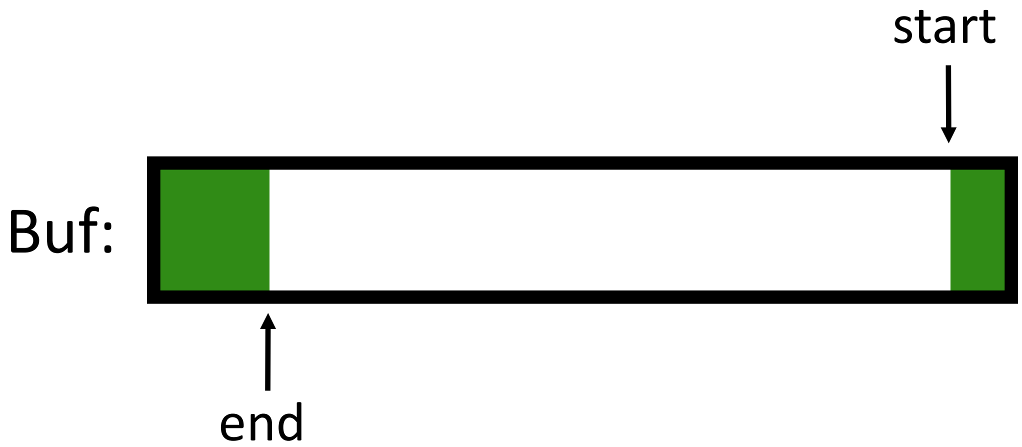
note: readers must wait

Example: UNIX Pipes

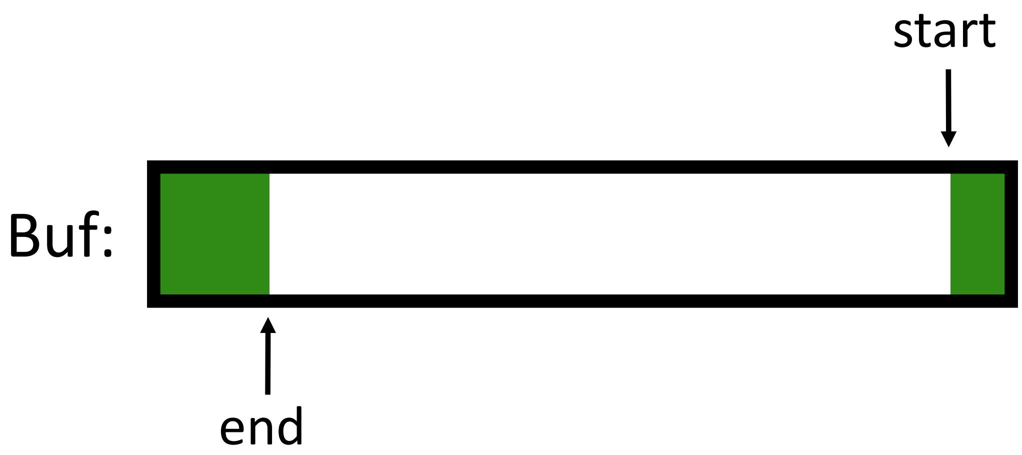


Example: UNIX Pipes

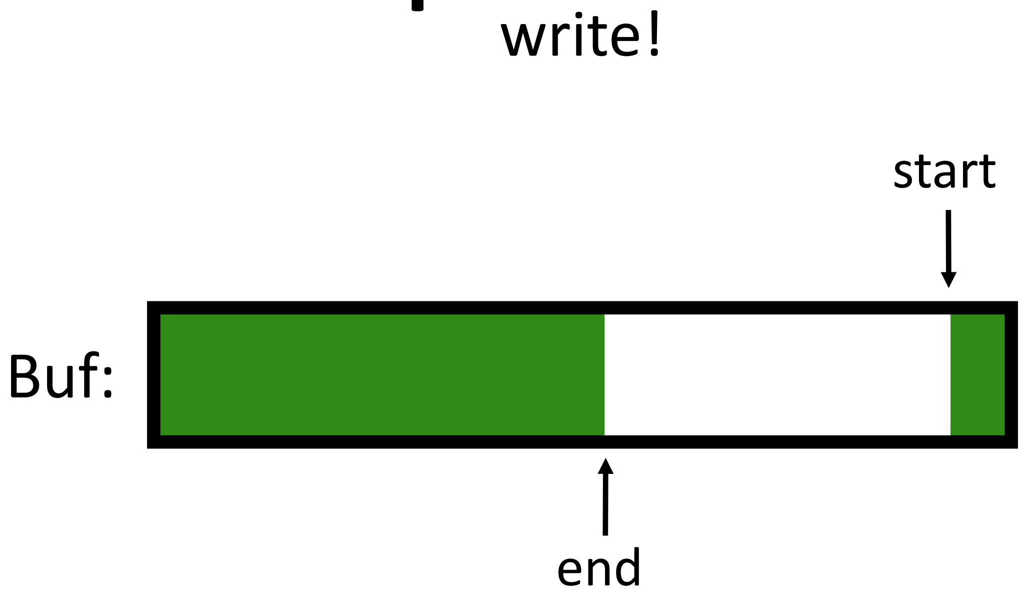
write!



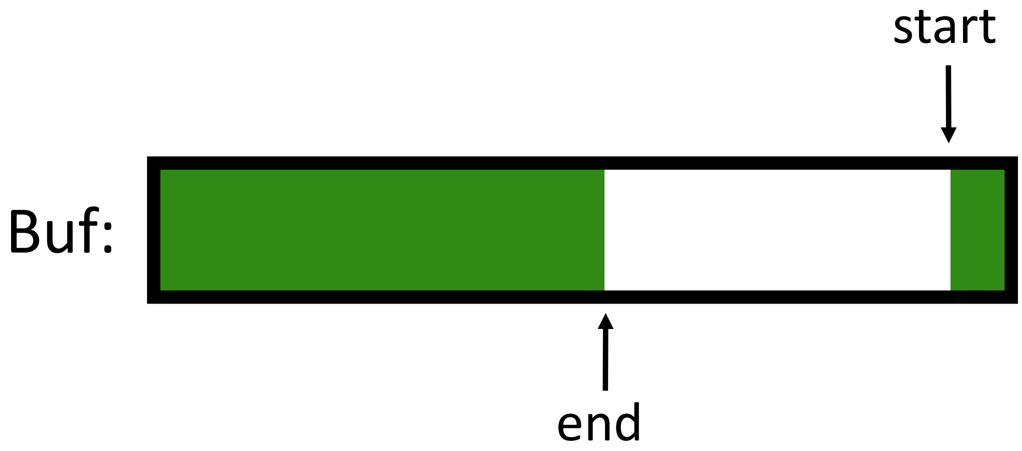
Example: UNIX Pipes



Example: UNIX Pipes



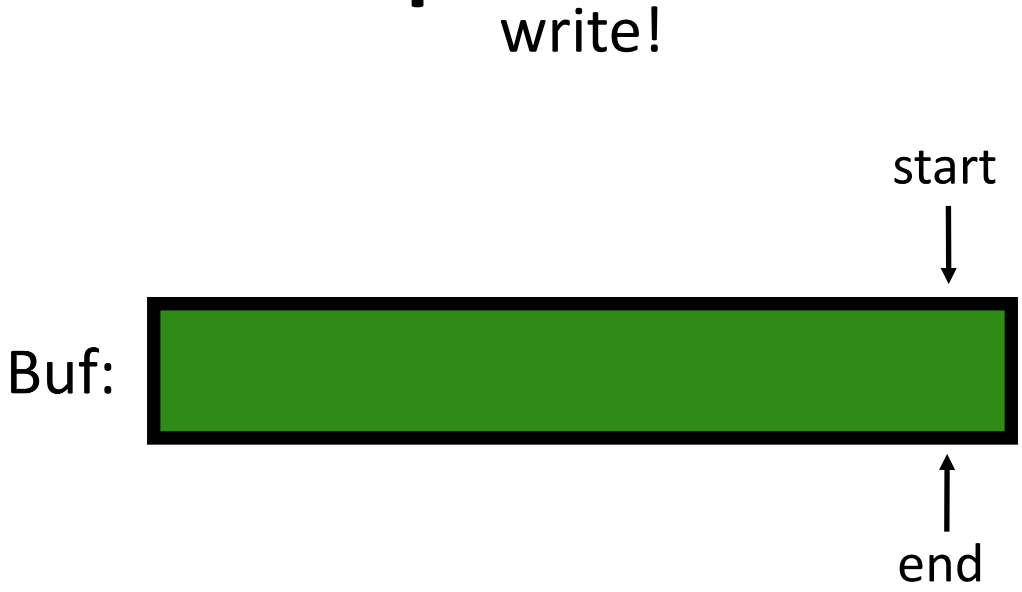
Example: UNIX Pipes



Example: UNIX Pipes

write!

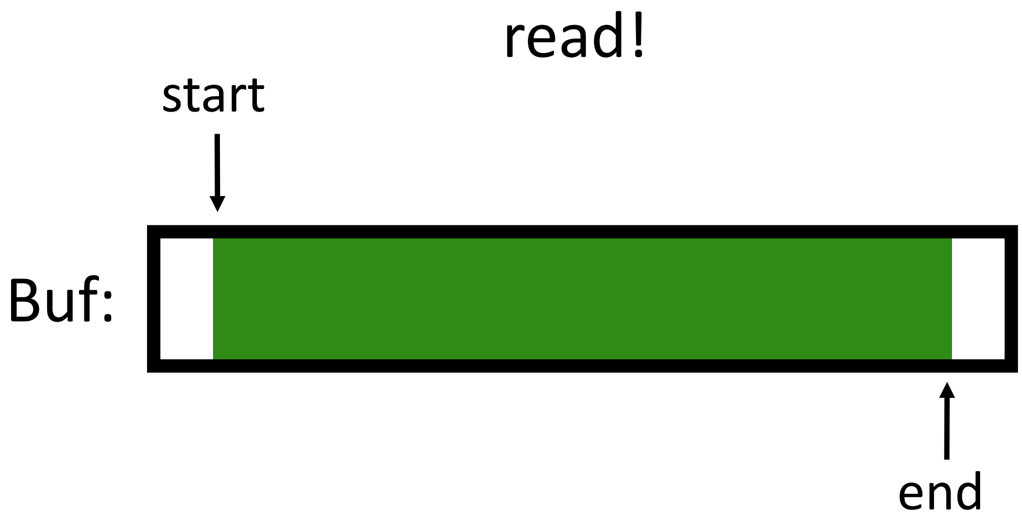
Example: UNIX Pipes



note: writers must wait

Example: UNIX Pipes

Example: UNIX Pipes



Example: UNIX Pipes

- **Implementation:**
  - reads/writes to buffer require locking
- **when buffers are full, writers must wait**
- **when buffers are empty, readers must wait**

Producer/Consumer Problem

- **Producers generate data (like pipe writers)**

- **Consumers grab data and process it (like pipe readers)**

- **Producer/consumer problems are frequent in systems**
- **Web servers**
  - General strategy use condition variables to:
  - make producers wait when buffers are full
  - make consumers wait when there is nothing to consume

Produce/Consumer Example

- **Start with easy case:**
  - 1 producer thread
  - 1 consumer thread
  - 1 shared buffer to fill/consume (max = 1)

- **Numfill = number of buffers currently filled**
- **Examine slightly broken code to begin…**

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
  for (int i=0; i<loops; i++) {
    Mutex_lock(&m);
    if (numfull == max)
      Cond_wait(&cond, &m);
    do_fill(i);
    Cond_signal(&cond);
    Mutex_unlock(&m);
  }
}
```

[RUNNING]

```
void *consumer(void *arg) {
  while(1) {
    Mutex_lock(&m);
    if (numfull == 0)
      Cond_wait(&cond, &m);
    int tmp = do_get();
    Cond_signal(&cond);
    Mutex_unlock(&m);
    printf("%d\n", tmp);
  }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
  for (int i=0; i<loops; i++) {
    Mutex_lock(&m);
    if (numfull == max)
      Cond_wait(&cond, &m);
    do_fill(i);
    Cond_signal(&cond);
    Mutex_unlock(&m);
  }
}
```

[RUNNING]

```
void *consumer(void *arg) {
  while(1) {
    Mutex_lock(&m);
    if (numfull == 0)
      Cond_wait(&cond, &m);
    int tmp = do_get();
    Cond_signal(&cond);
    Mutex_unlock(&m);
    printf("%d\n", tmp);
  }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
    for (int i=0; i<loops; i++) {
        Mutex_lock(&m);
        if (numfull == max)
            Cond_wait(&cond, &m);
        do_fill(i);
        Cond_signal(&cond);
        Mutex_unlock(&m);
    }
}
```

[RUNNING]

```
void *consumer(void *arg) {
    while(1) {
        Mutex_lock(&m);
        if (numfull == 0)
            Cond_wait(&cond, &m);
        int tmp = do_get();
        Cond_signal(&cond);
        Mutex_unlock(&m);
        printf("%d\n", tmp);
    }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
  for (int i=0; i<loops; i++) {
    Mutex_lock(&m);
    if (numfull == max)
      Cond_wait(&cond, &m);
    do_fill(i);
    Cond_signal(&cond);
    Mutex_unlock(&m);
  }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
  while(1) {
    Mutex_lock(&m);
    if (numfull == 0)
      Cond_wait(&cond, &m);
    int tmp = do_get();
    Cond_signal(&cond);
    Mutex_unlock(&m);
    printf("%d\n", tmp);
  }
}
```

numfull=0

[RUNNING]

```
void *producer(void *arg) {
  for (int i=0; i<loops; i++) {
    Mutex_lock(&m);
    if (numfull == max)
      Cond_wait(&cond, &m);
    do_fill(i);
    Cond_signal(&cond);
    Mutex_unlock(&m);
  }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
  while(1) {
    Mutex_lock(&m);
    if (numfull == 0)
      Cond_wait(&cond, &m);
    int tmp = do_get();
    Cond_signal(&cond);
    Mutex_unlock(&m);
    printf(“%d\n”, tmp);
  }
}
```

numfull=0

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
  for (int i=0; i<loops; i++) {
    Mutex_lock(&m);
    if (numfull == max)
      Cond_wait(&cond, &m);
    do_fill(i);
    Cond_signal(&cond);
    Mutex_unlock(&m);
  }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
  while(1) {
    Mutex_lock(&m);
    if (numfull == 0)
      Cond_wait(&cond, &m);
    int tmp = do_get();
    Cond_signal(&cond);
    Mutex_unlock(&m);
    printf("%d\n", tmp);
  }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[SLEEPING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[SLEEPING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[SLEEPING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf(“%d\n”, tmp);
   }
}
```

numfull=0

[SLEEPING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
  for (int i=0; i<loops; i++) {
    Mutex_lock(&m);
    if (numfull == max)
      Cond_wait(&cond, &m);
    do_fill(i);
    Cond_signal(&cond);
    Mutex_unlock(&m);
  }
}
```

[RUNNING]

```
void *consumer(void *arg) {
  while(1) {
    Mutex_lock(&m);
    if (numfull == 0)
      Cond_wait(&cond, &m);
    int tmp = do_get();
    Cond_signal(&cond);
    Mutex_unlock(&m);
    printf("%d\n", tmp);
  }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNABLE]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf(“%d\n”, tmp);
   }
}
```

numfull=0

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=0

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[SLEEPING]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

numfull=1

[RUNNING]

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m);
      if (numfull == max)
         Cond_wait(&cond, &m);
      do_fill(i);
      Cond_signal(&cond);
      Mutex_unlock(&m);
   }
}
```

[RUNNABLE]

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&cond, &m);
      int tmp = do_get();
      Cond_signal(&cond);
      Mutex_unlock(&m);
      printf("%d\n", tmp);
   }
}
```

What about 2 consumers?

- Can you find a problematic timeline with 2 consumers (still 1 producer)?

```
void *producer(void *arg) {
    for (int i=0; i<loops; i++) {
        Mutex_lock(&m); // p1
        if (numfull == 1) //p2
            Cond_wait(&cond, &m); //p3
        do_fill(i); // p4
        Cond_signal(&cond); //p5
        Mutex_unlock(&m); //p6
    }
}
```

```
void *consumer(void *arg) {
    while(1) {
        Mutex_lock(&m);  // c1
        if (numfull == 0) // c2
            Cond_wait(&cond, &m); // c3
        int tmp = do_get(); // c4
        Cond_signal(&cond); // c5
        Mutex_unlock(&m); // c6
        printf("%d\n", tmp); // c7
    }
}
```

another consumer sneaks in

get nothing!

Producer:				p1	p2	p4	p5	p6	p1	p2	p3						
Consumer1:	c1	c2	c3				[Runable]										c4
Consumer2:												c1	c2	c4	c5	c6	

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m); // p1
      while(numfull == 1) //p2
         Cond_wait(&cond, &m); //p3
      do_fill(i); // p4
      Cond_signal(&cond); //p5
      Mutex_unlock(&m); //p6
   }
}
```

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);  // c1
      while(numfull == 0) // c2
         Cond_wait(&cond, &m); // c3
      int tmp = do_get(); // c4
      Cond_signal(&cond); // c5
      Mutex_unlock(&m); // c6
      printf("%d\n", tmp); // c7
   }
}
```

Customer1 wakes up and rechecks the state (c2)

```
void *producer(void *arg) {
   for (int i=0; i<loops; i++) {
      Mutex_lock(&m); // p1
      while(numfull == 1) //p2
         Cond_wait(&cond, &m); //p3
      do_fill(i); // p4
      Cond_signal(&cond); //p5
      Mutex_unlock(&m); //p6
   }
}
```

```
void *consumer(void *arg) {
   while(1) {
      Mutex_lock(&m);  // c1
      while(numfull == 0) // c2
         Cond_wait(&cond, &m); // c3
      int tmp = do_get(); // c4
      Cond_signal(&cond); // c5
      Mutex_unlock(&m); // c6
      printf("%d\n", tmp); // c7
   }
}
```

wait() wait() signal() wait() signal()

Producer: p1 p2 p4 p5 p6 p1 p2 p3

Consumer1: c1 c2 c3

Consumer2: c1 c2 c3 c2 c4 c5

Does last signal wake producer or consumer1?

How to wake the right thread?

- **One solution:**

  wake all the threads!

Waking All Waiting Threads

- **wait(cond_t *cv, mutex_t *lock)**
  - assumes the lock is held when wait() is called
  - puts caller to sleep + releases the lock (atomically)
  - when awoken, reacquires lock before returning
- **signal(cond_t *cv)**
  - wake a single waiting thread (if >= 1 thread is waiting)
  - if there is no waiting thread, just return, doing nothing
- **broadcast(cond_t *cv)**
  - wake all waiting threads (if >= 1 thread is waiting)
  - if there are no waiting thread, just return, doing nothing

any disadvantage?

Example Need for Broadcast

```
void *allocate(int size) {
    mutex_lock(&m);
    while (bytesLeft < size)
        cond_wait(&c);
    …
}
```

```
void free(void *ptr, int size) {
    …
    cond_broadcast(&c)
    …
}
```

How to wake the right thread?

- **One solution:**

  wake all the threads!

- **Better solution (usually): use two condition variables**

Producer/Consumer: Two CVs

```
void *producer(void *arg) {
   for (int i = 0; i < loops; i++) {
      Mutex_lock(&m); // p1
      if (numfull == max) // p2
         Cond_wait(&empty, &m); // p3
      do_fill(i);  // p4
      Cond_signal(&fill); // p5
      Mutex_unlock(&m); //p6
   }
}
```

```
void *consumer(void *arg) {
   while (1) {
      Mutex_lock(&m);
      if (numfull == 0)
         Cond_wait(&fill, &m);
      int tmp = do_get();
      Cond_signal(&empty);
      Mutex_unlock(&m);
   }
}
```

- Is this correct? Can you find a bad schedule?

  1. consumer1 waits because numfull == 0
  2. producer increments numfull, wakes consumer1
  3. before consumer1 runs, consumer2 runs, grabs entry, sets numfull=0.
  4. consumer1 then reads bad data.

Good Rule of Thumb 3

- Whenever a lock is acquired, recheck assumptions about state!
- Possible for another thread to grab lock in between signal and wakeup from wait
- Note that some libraries also have “spurious wakeups” (may wake multiple waiting threads at signal or at any time)

Producer/Consumer: Two CVs and While

```
void *producer(void *arg) {
   for (int i = 0; i < loops; i++) {
      Mutex_lock(&m); // p1
      while (numfull == max) // p2
         Cond_wait(&empty, &m); // p3
      do_fill(i);  // p4
      Cond_signal(&fill); // p5
      Mutex_unlock(&m); //p6
   }
}
```

```
void *consumer(void *arg) {
   while (1) {
      Mutex_lock(&m);
      while (numfull == 0)
         Cond_wait(&fill, &m);
      int tmp = do_get();
      Cond_signal(&empty);
      Mutex_unlock(&m);
   }
}
```

- Is this correct? Can you find a bad schedule?
- **Correct!**
  - no concurrent access to shared state
  - every time lock is acquired, assumptions are reevaluated
  - a consumer will get to run after every do_fill()
  - a producer will get to run after every do_get()

Summary: rules of thumb for CVs

- **Keep state** in addition to CV's
- Always do wait/signal **with lock held**
- Whenever thread wakes from waiting, **recheck state**