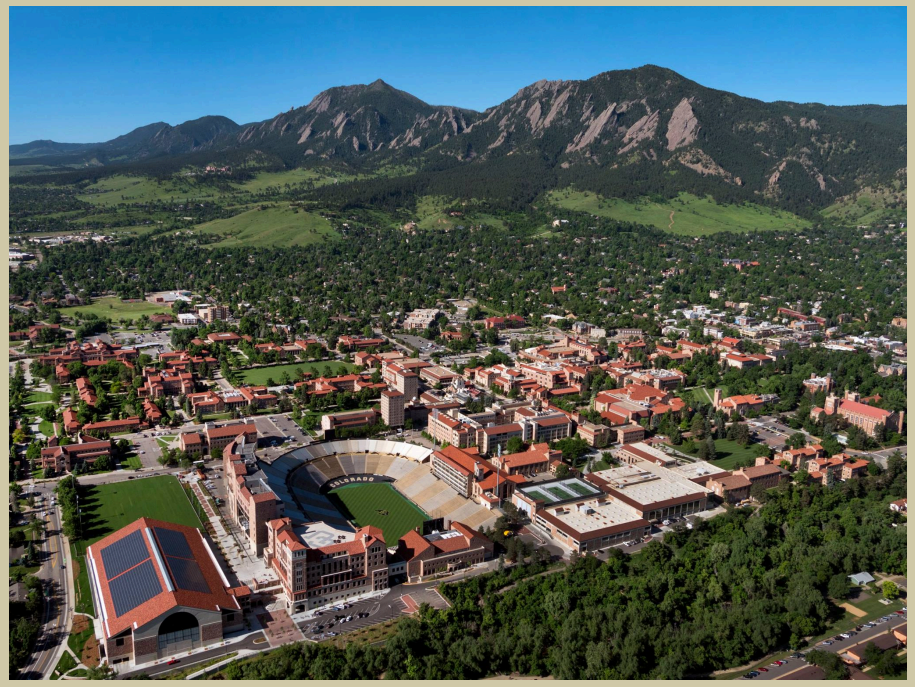
Design and Analysis of Operating Systems
CSCI 3753

Dr. David Knox
University of Colorado Boulder

Bounded Buffer Problem



Design and Analysis of Operating Systems
CSCI 3753

Dr. David Knox
University of Colorado Boulder

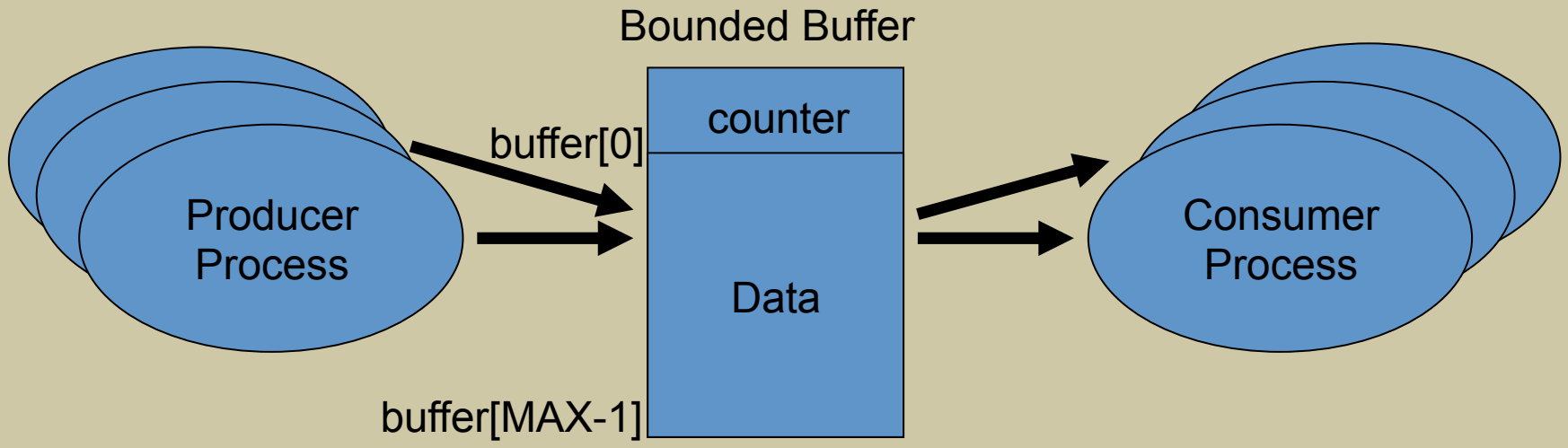
Producer-Consumer Problem

also known as

Bounded Buffer Problem

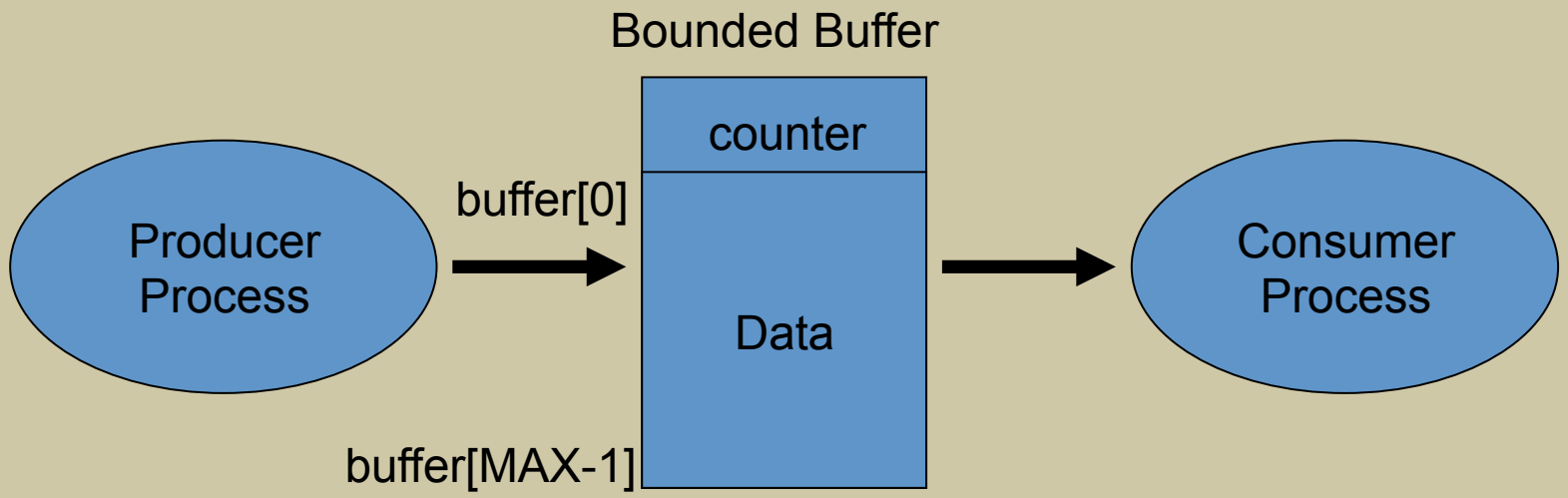
- We have already seen this problem with one producer and one consumer
- General problem: multiple producers and multiple consumers
- **Producers** puts new information in the buffer
- **Consumers** takes out information from the buffer

Bounded-Buffer Problem



- Producer places data into a buffer at the next available position
- Consumer takes information from the earliest item

Prior Bounded-Buffer Approach




```
while(1) {
  produce (nextdata)
  while(count==MAX);
  Acquire(lock);
  buffer[count] = nextdata;
  count++;
  Release(lock);
}
```

```
while(1) {
  while(count==0);
  Acquire(lock);
  data = buffer[count-1];
  count--;
  Release(lock);
  consume(data);
}
```

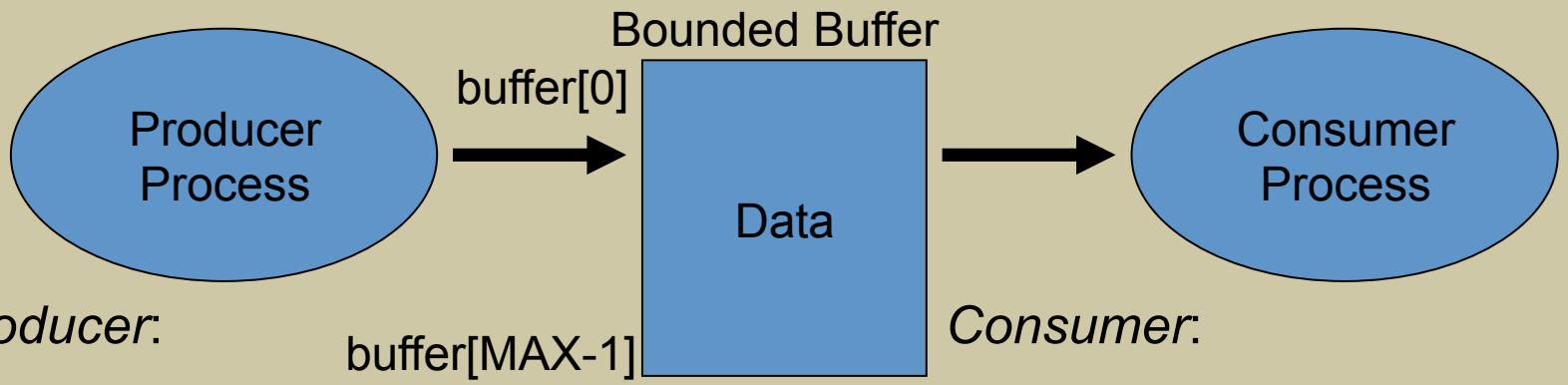
while(TS(&lock)) ;

Busy-wait!

Bounded-Buffer Goals

- In a prior approach, both the producer and consumer are ***busy-waiting*** using locks
- Instead, want both to sleep as necessary
  - Goal #1: Producer should block when buffer is full
  - Goal #2: Consumer should block when the buffer is empty
  - Goal #3: Mutual exclusion when buffer is partially full
    - Producer and consumer should access the buffer in a synchronized mutually exclusive way

Bounded Buffer Solution



Producer:

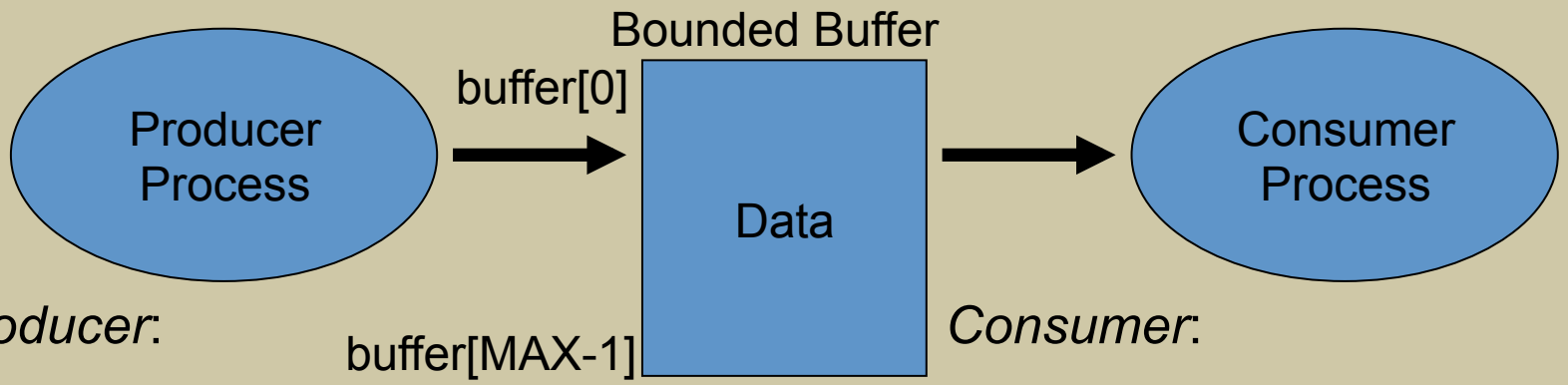
```
while(1) {
    …
    P(mutex)  // Goal 3: mutual
                 exclusion
       add item to buffer
    V(mutex)
    …
}
```

Consumer:

```
while(1) {
    ...
    P(mutex)  // Goal 3: mutual
                 exclusion
       remove item from buffer
    V(mutex)
    ...
}
```

Assume $mutex_{init} = 1$

Bounded Buffer Solution (2)



Producer:

Consumer:

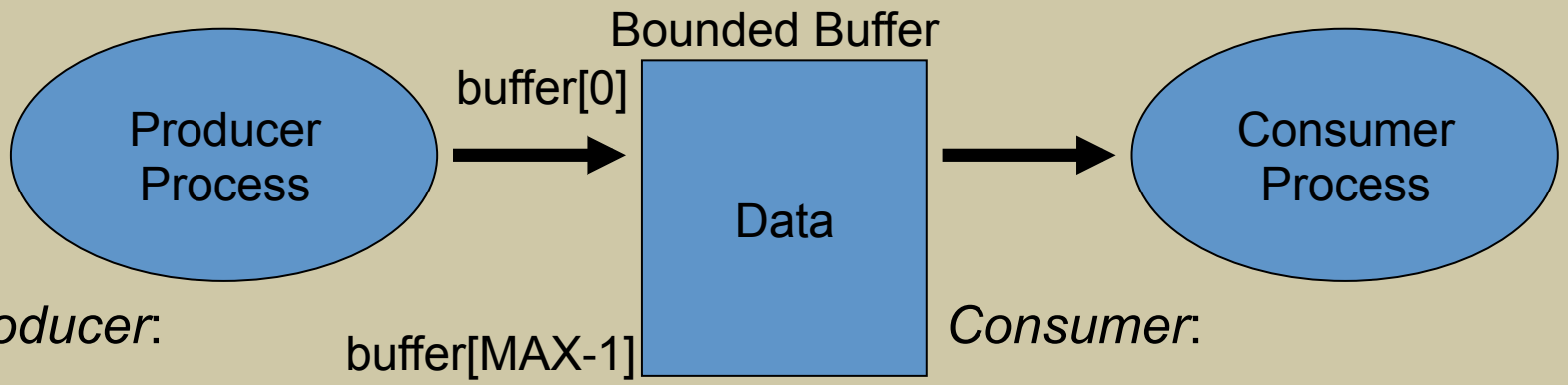
Goal #1: Producer should block when buffer is full

```
while(1) {
  P(empty) // Goal 1: full buffer
              blocks producer
    P(mutex)
       add item to buffer
    V(mutex)
    …
}
```

```
while(1) {
    ...
    P(mutex)
       remove item from buffer
    V(mutex)
  V(empty) // Goal 1: full buff
              blocks producer
}
```

Assume $mutex_{init}$=1, $empty_{init}$=MAX

Bounded Buffer Solution (3)



Producer:

Consumer:

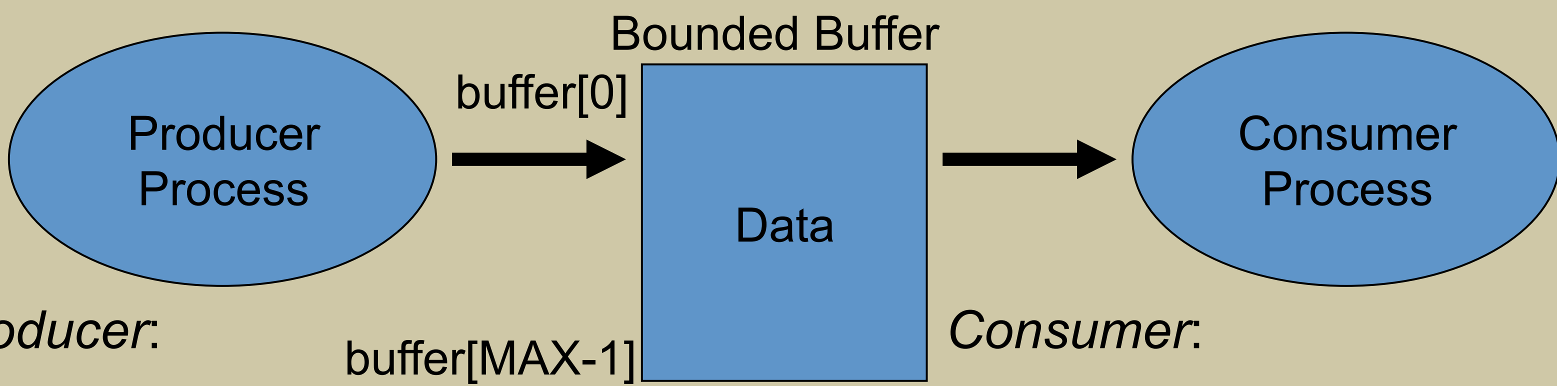
Goal #2: Consumer should block when the buffer is empty

```
while(1) {
  P(empty)
    P(mutex)
       add item to buffer
    V(mutex)
  V(full) // Goal 2: empty buffer
             blocks consumer
}
```

```
while(1) {
  P(full) // Goal 2: empty
            buff blocks consumer
    P(mutex)
       remove item from buffer
    V(mutex)
  V(empty)
}
```

Assume $mutex_{init}$=1, $empty_{init}$=MAX, $full_{init}$=0

Bounded Buffer Solution (4)



Producer:

```
while(1) {
  P(empty)
    P(mutex)
      add ite
    V(mutex)
  V(full)
}
```

Consumer:

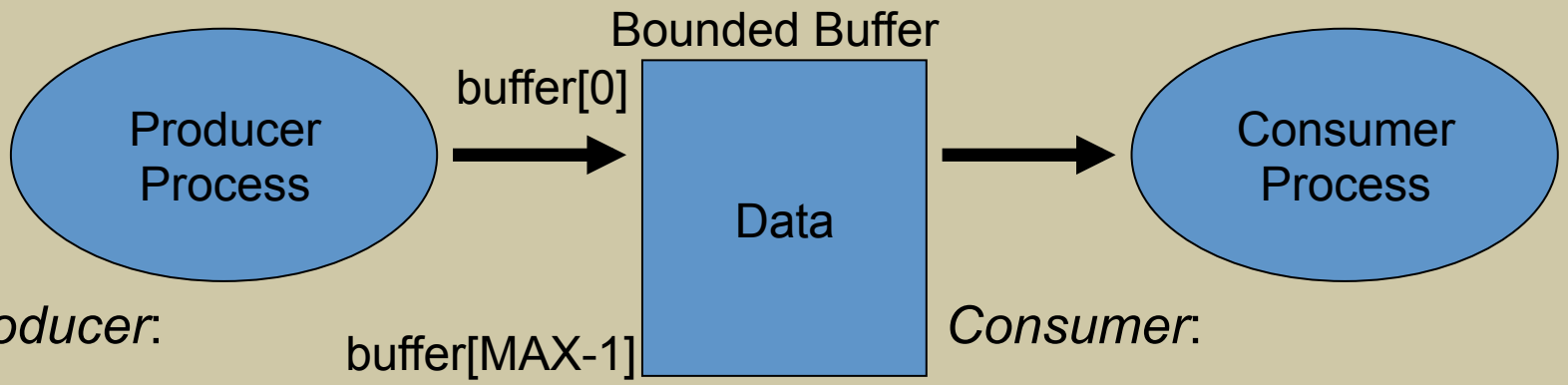
```
while(1) {
    
    :ex)
      emove item from buffer
    :ex)
  )
}
```

Achieves:

1) mutual exclusion
2) blocked producer if buffer full
3) blocked consumer if buffer empty
4) no deadlock
5) is efficient (no busy wait)

Assume $mutex_{init}$=1, $empty_{init}$=MAX, $full_{init}$=0

Bounded Buffer Solution (5)



Producer:

```
while(1) {
  P(space_avail)
    P(mutex)
       add item to buffer
    V(mutex)
  V(items_avail)
}
```

Consumer:

```
while(1) {
  P(items_avail)
    P(mutex)
       remove item from buffer
    V(mutex)
  V(space_avail)
}
```

Assume $mutex_{init}$=1,
$space\_avail_{init}$=MAX,
$items\_avail_{init}$=0

Bounded-Buffer Design

- **Goal #1: Producer should block when buffer is full**
  - Use a counting semaphore called *space_avail* that is initialized to $space\_avail_{init}$ = MAX
  - Each time the producer adds an object to the buffer, this decrements the # of empty slots, until it hits 0 and the producer blocks
- **Goal #2: Consumer should block when the buffer is empty**
  - Define a counting semaphore *items_avail* that is initialized to $items\_avail_{init}$ = 0
  - *items_avail* tracks the # of full slots and is incremented by the producer
  - Each time the consumer removes a full slot, this decrements *items_avail*, until it hits 0, then the consumer blocks
- **Goal #3: Mutual exclusion when buffer is partially full**
  - Use a mutex semaphore to protect access to buffer manipulation, $mutex_{init}$ = 1

Bounded Buffer Solution (6)

Will this solution work for multiple Producers and multiple Consumers?

Producer:

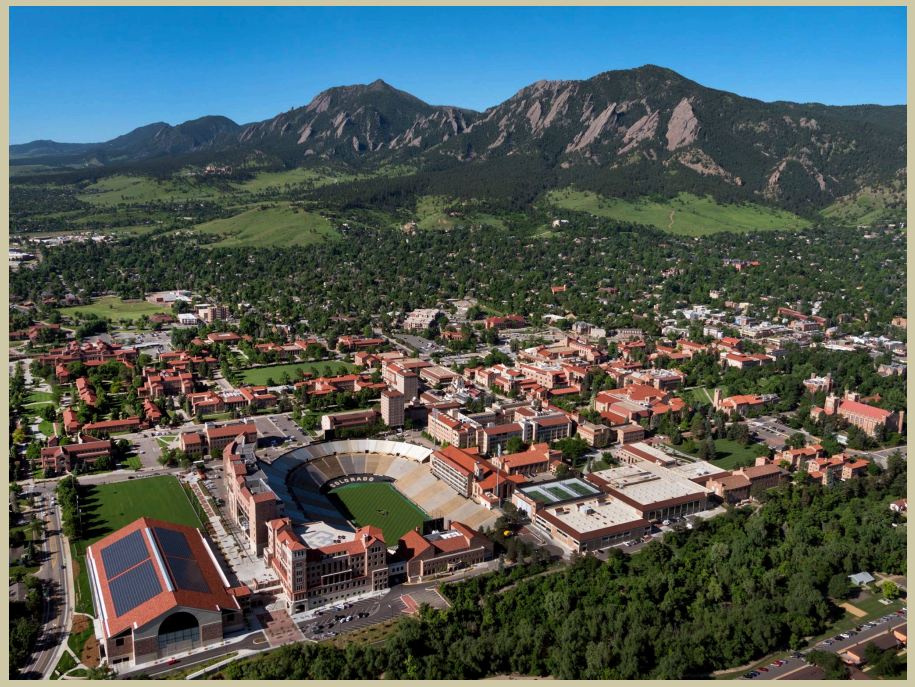
```
while(1) {
  P(space_avail)
    P(mutex)
       add item to buffer
    V(mutex)
  V(items_avail)
}
```

r:

```
while(1) {
  P(items_avail)
    P(mutex)
       remove item from buffer
    V(mutex)
  V(space_avail)
}
```

Assume $mutex_{init}$=1,
$space\_avail_{init}$=MAX,
$items\_avail_{init}$=0

Design and Analysis of Operating Systems CSCI 3753



Dr. David Knox

University of Colorado
Boulder