CSCI-UA.0201

Computer Systems Organization

Concurrency – Condition Variables

Thomas Wies
wies@cs.nyu.edu
https://cs.nyu.edu/wies

Producer/consumer based on a FIFO Queue

```
queue_t queue;
pthread_mutex_t mu;
...
void produce(int x) {
  pthread_mutex_lock(&mu);
  enqueue(&queue, x);
  pthread_mutex_unlock(&mu);
}
```

The Need for Modular Synchronization

Suppose queue is bounded:

- enqueue may block until queue has room
- decision whether to block depends on internal state of the queue

Multiple producers/consumers:

- every thread needs to keep track of the lock, the queue state, etc.

The Need for Modular Synchronization

Suppose queue is bounded:

- enqueue may block until queue has room
- decision whether to block depends on internal state of the queue

Multiple producers/consumers:

- every thread needs to keep track of the lock, the queue state, etc.

not scalable

Modular Synchronization

Let queue handle its own synchronization

- queue has its own lock
  - acquired by each enqueue/dequeue call
  - released when the call returns
- if thread enqueues on a full queue
  - queue itself detects the problem
  - suspend the caller and resume when the queue has room

Condition Variables

- A mechanism to block a thread until some condition becomes true
- Condition variables allow a thread to
  - temporarily release the lock and suspend itself until awoken by another thread
  - awake other threads that are currently suspended waiting for that condition

Monitors

The combination of

- a data structure and its operations
- a mutual exclusion lock
- and the lock's condition variables is called a **monitor**

Monitors enable modular synchronization.

Condition Variables in the pthread lib

- `pthread_cond_t`
- `pthread_cond_wait` / `pthread_cond_timedwait`
- `pthread_cond_signal`
- `pthread_cond_broadcast`

pthread_cond_wait

```
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
```

- Atomically releases `mutex` and causes the calling thread to be put on an internal waiting queue for `cond`.
- On successful return, `mutex` is locked (which the calling thread should unlock later)

pthread_cond_wait

```
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
```

- Atomically releases `mutex` and causes the calling thread to be put on an internal waiting queue for `cond`.
- On successful return, `mutex` is locked (which the calling thread should unlock later)

No other thread can grab the released mutex before the calling thread is put in the waiting queue

pthread_cond_signal

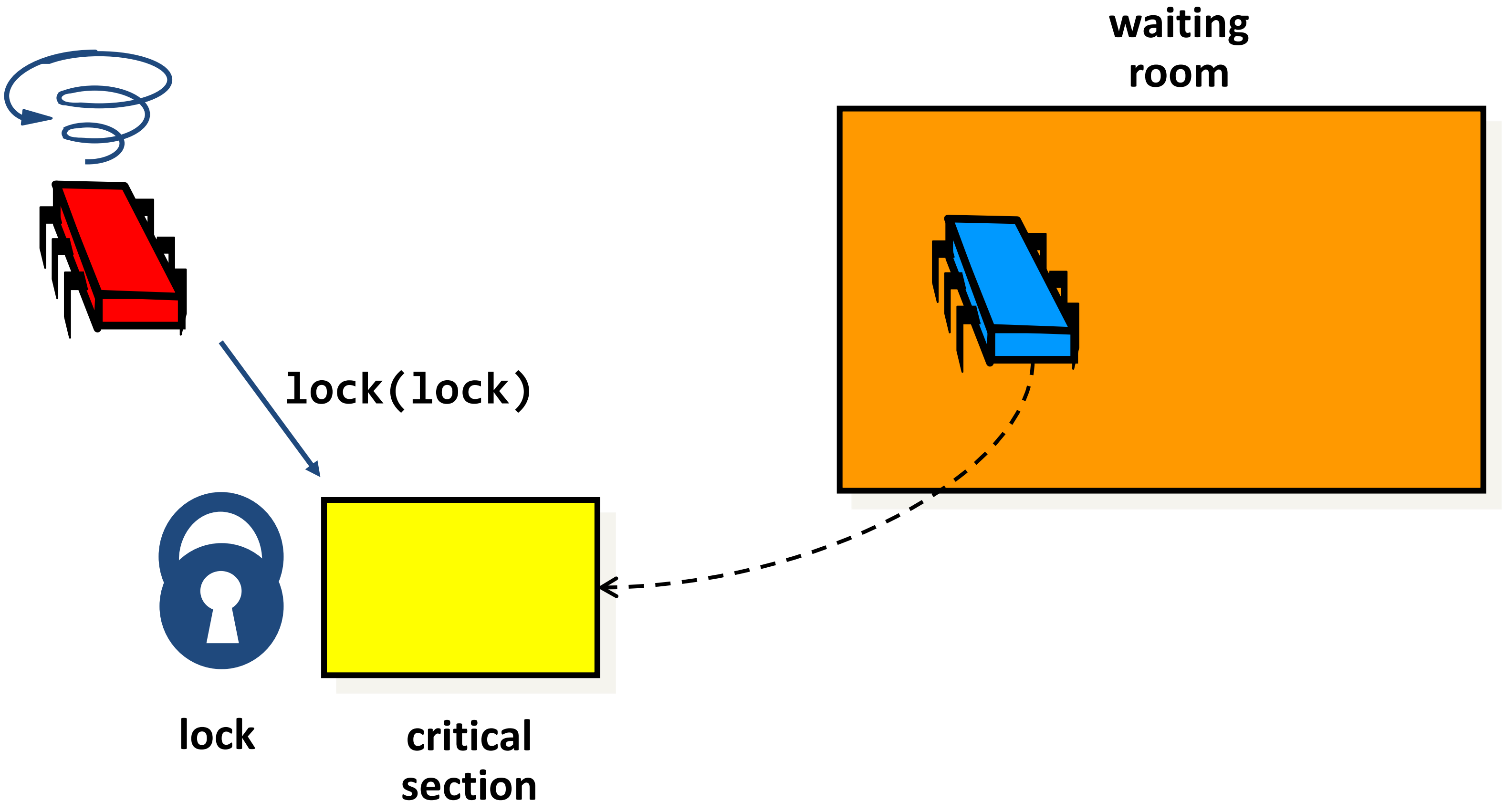
```
int pthread_cond_signal(pthread_cond_t *cond);
```

- Unblock at least one of the threads waiting on `cond`

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

- Unblock all threads waiting on `cond`.

A Typical Monitor Execution



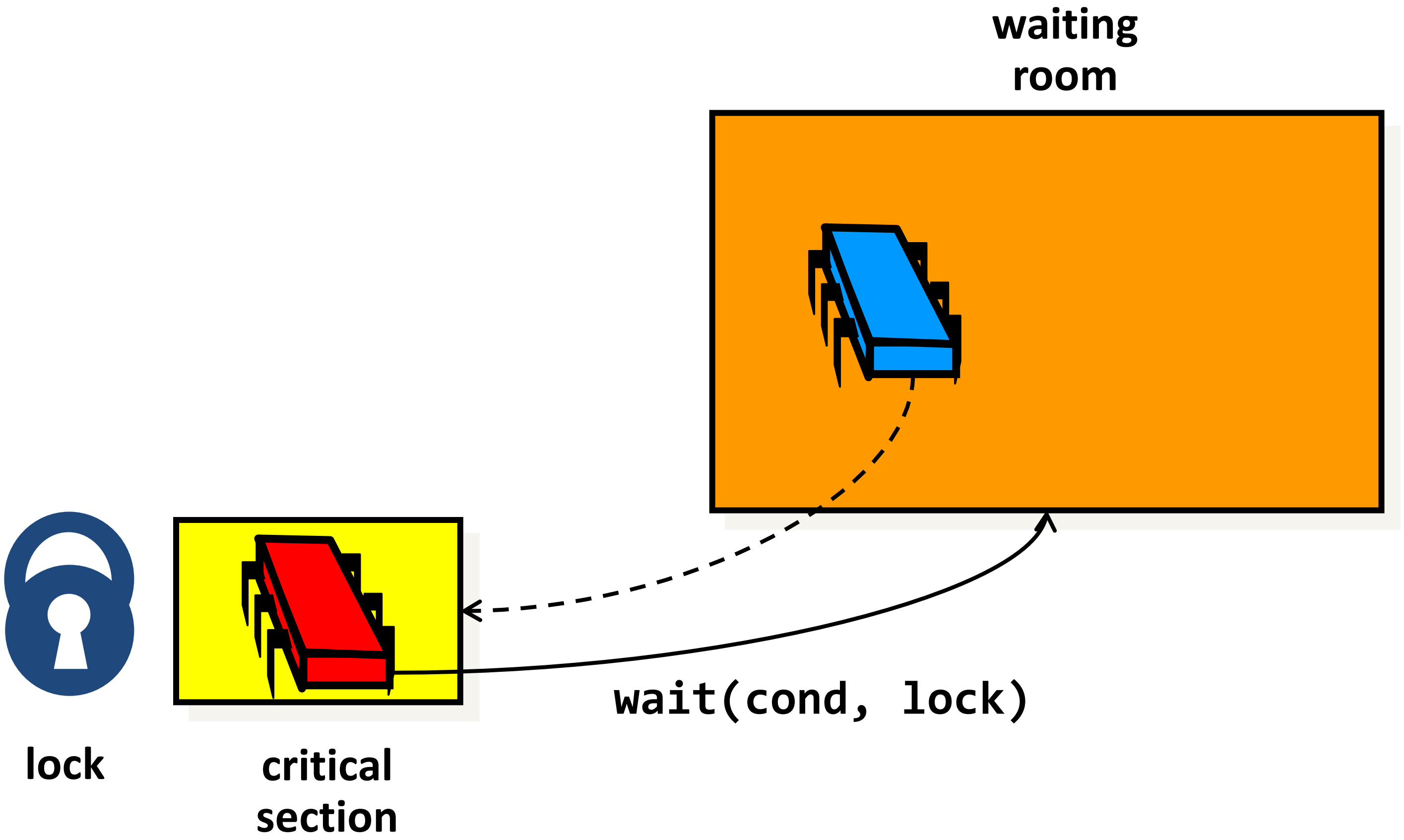
waiting room

`lock(lock)`

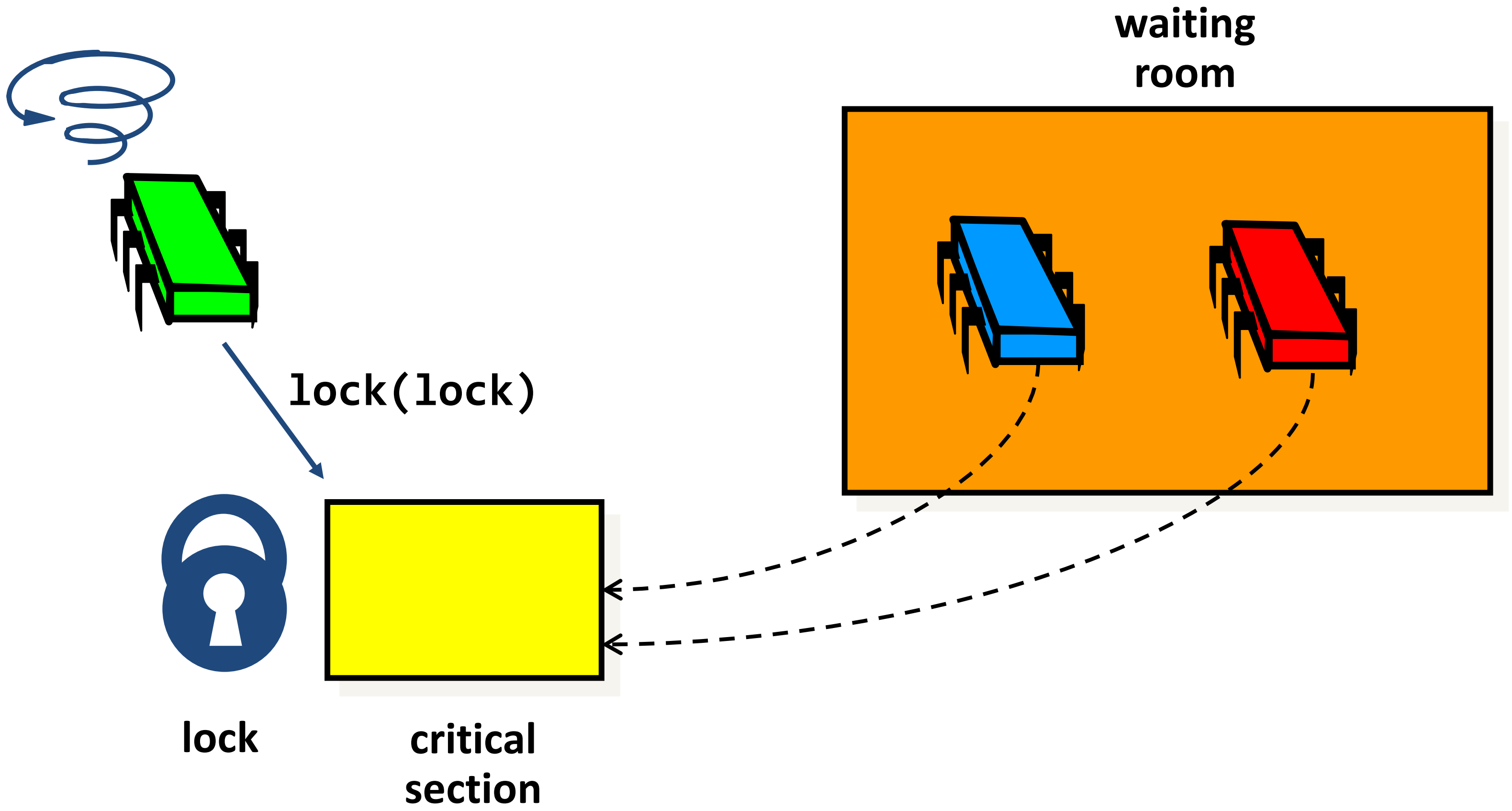
lock

critical section

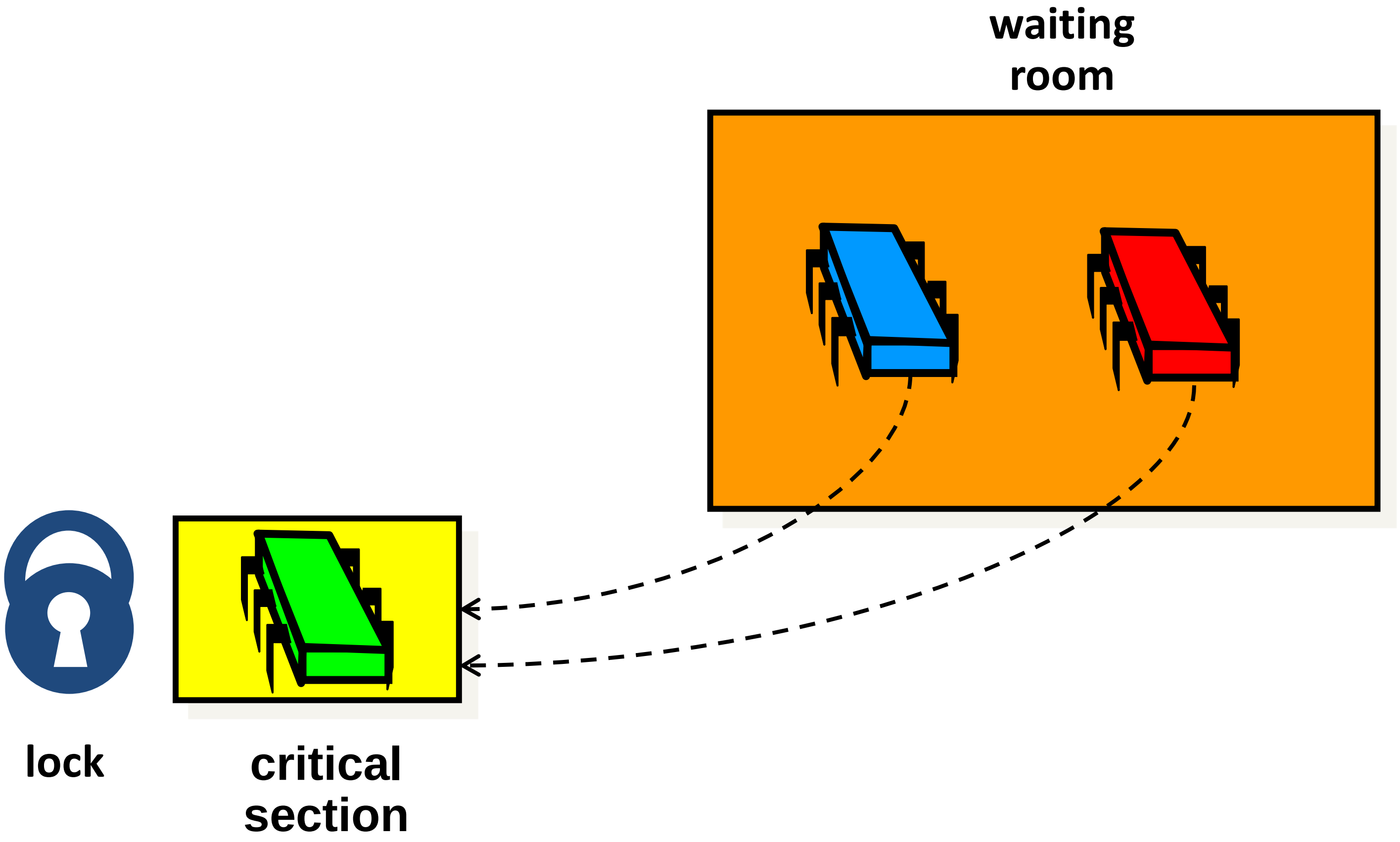
A Typical Monitor Execution



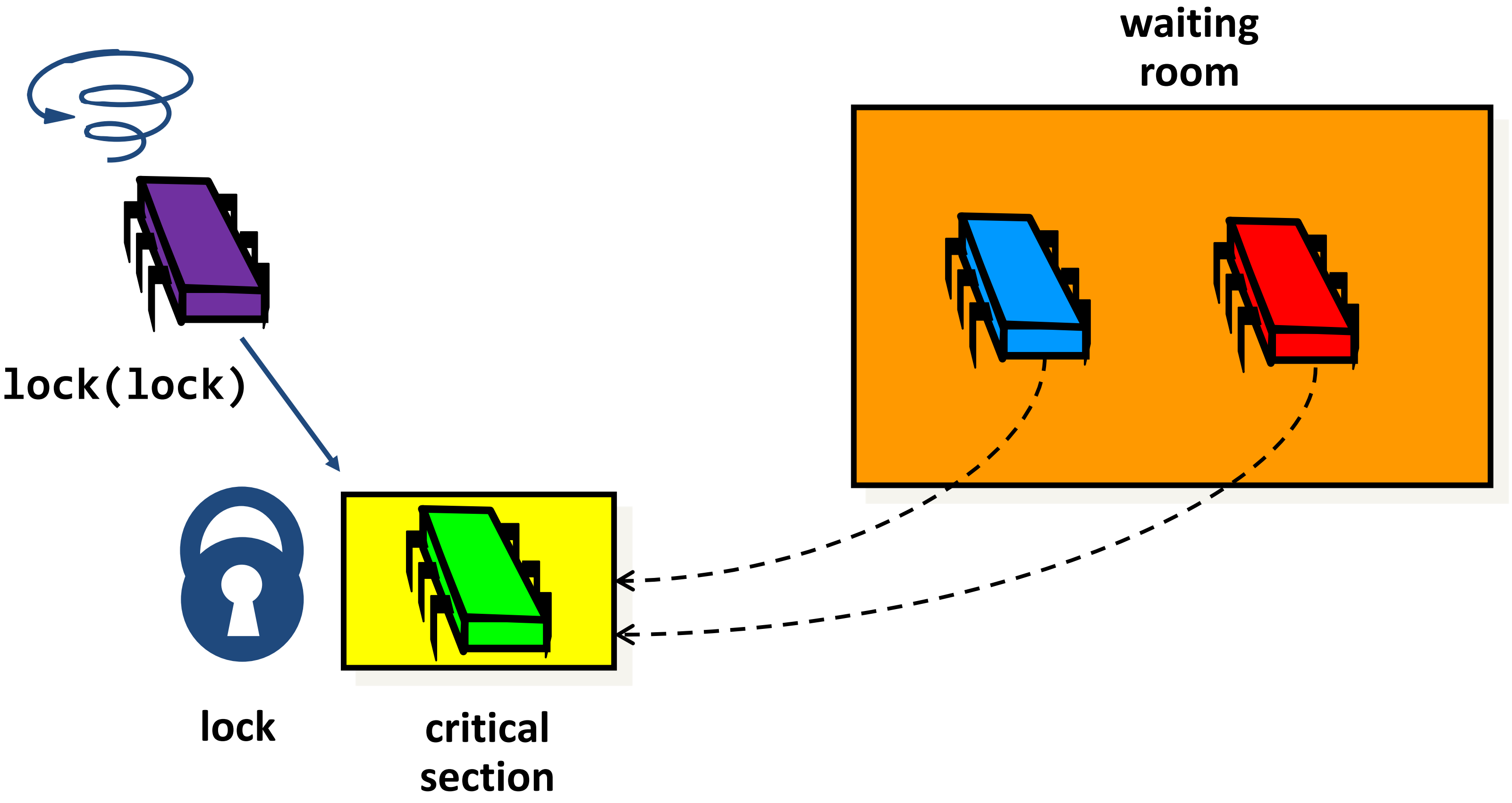
A Typical Monitor Execution



A Typical Monitor Execution



A Typical Monitor Execution



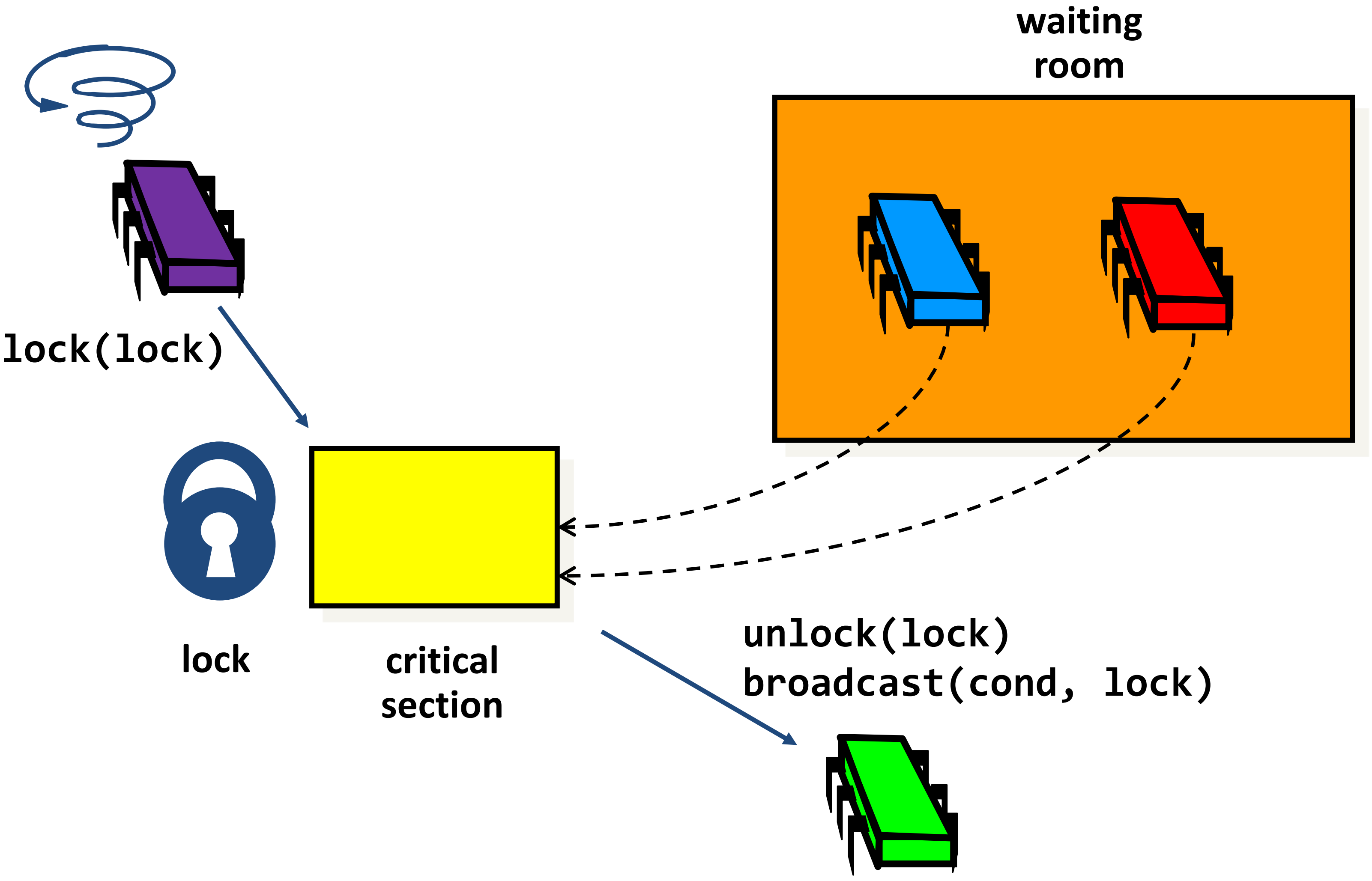
waiting room

lock(lock)

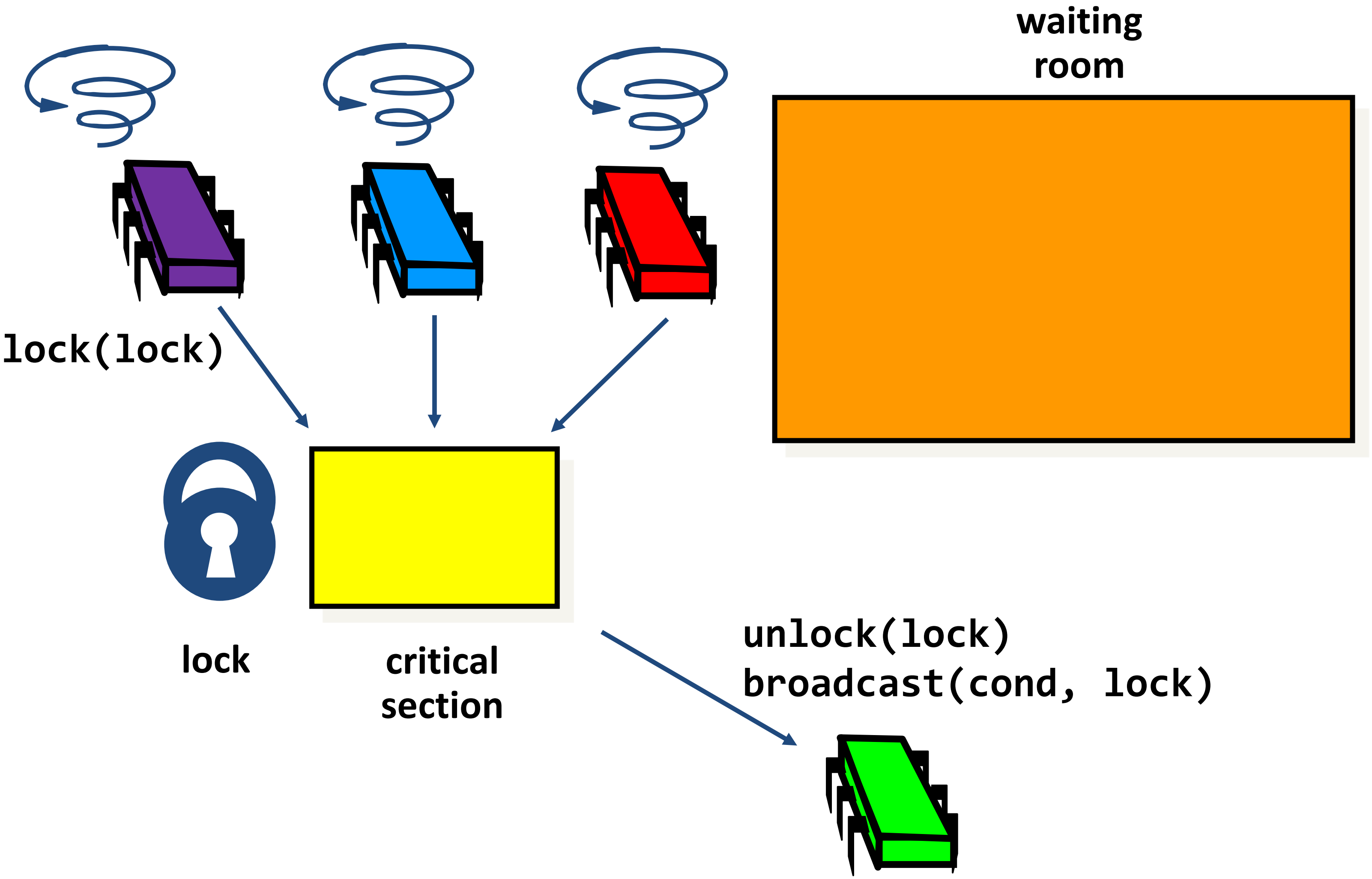
lock

critical section

A Typical Monitor Execution



A Typical Monitor Execution



waiting room

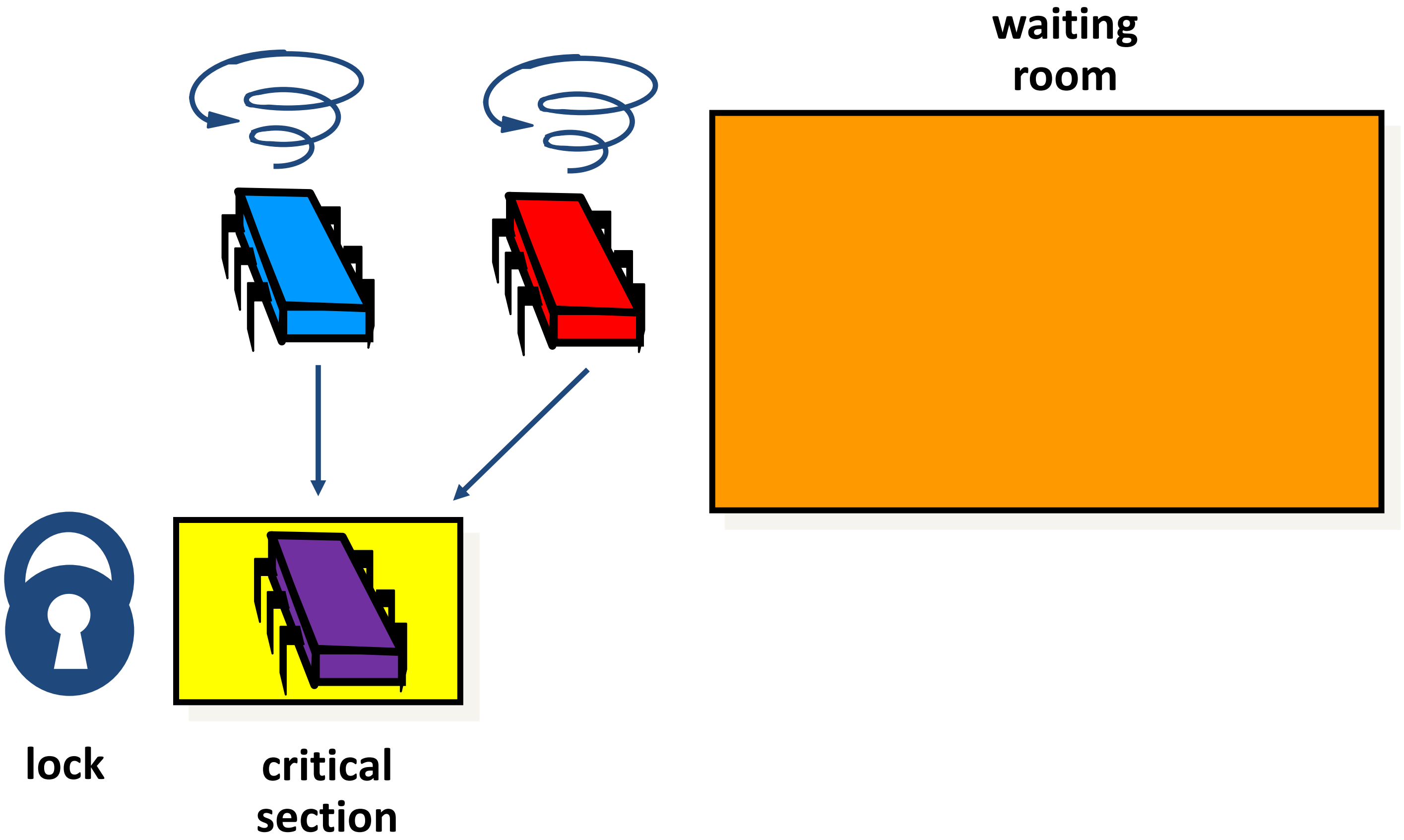
`lock(lock)`

lock

critical section

```
unlock(lock)
broadcast(cond, lock)
```

A Typical Monitor Execution



Using Condition Variables

```
pthread_mutex_t mu;
pthread_cond_t cond;
...
void foo() {
  pthread_mutex_lock(&mu);
  while (!property)
    pthread_cond_wait(&cond, &mu);
  ...
  pthread_mutex_unlock(&mu);
}
```

Using Condition Variables

```
pthread_mutex_t mu;
pthread_cond_t cond;
...
void foo() {
  pthread_mutex_lock(&mu);
  while (!property)
    pthread_cond_wait(&cond, &mu);
  ...
  pthread_mutex_unlock(&mu);
}
```

create new condition variable

Using Condition Variables

```
pthread_mutex_t mu;
pthread_cond_t cond;
...
void foo() {
  pthread_mutex_lock(&mu);
  while (!property)
    pthread_cond_wait(&cond, &mu);
  ...
  pthread_mutex_unlock(&mu);
}
```

acquire the lock

Using Condition Variables

```
pthread_mutex_t mu;
pthread_cond_t cond;
...
void foo() {
  pthread_mutex_lock(&mu);
  while (!property)
    pthread_cond_wait(&cond, &mu);
  ...
  pthread_mutex_unlock(&mu);
}
```

not happy

Using Condition Variables

```
pthread_mutex_t mu;
pthread_cond_t cond;
...
void foo() {
  pthread_mutex_lock(&mu);
  while (!property)
    pthread_cond_wait(&cond, &mu);
  ...
  pthread_mutex_unlock(&mu);
}
```

release the lock and suspend until notified

Using Condition Variables

```
pthread_mutex_t mu;
pthread_cond_t cond;
...
void foo() {
  pthread_mutex_lock(&mu);
  while (!property)
    pthread_cond_wait(&cond, &mu);
  ...
  pthread_mutex_unlock(&mu);
}
```

happy: property must hold

Example: Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t notFull;
  pthread_cond_t notEmpty;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

Example: Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t notFull;
  pthread_cond_t notEmpty;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

mutex lock for queue

Example: Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t notFull;
  pthread_cond_t notEmpty;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

condition to wait on if queue is full

Example: Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t notFull;
  pthread_cond_t notEmpty;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

condition to wait on if queue is empty

Example: Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t notFull;
  pthread_cond_t notEmpty;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

internal queue state protected by lock

Blocking Queue: enqueue

```
void enqeue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->notFull, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  pthread_cond_signal(&queue->notEmpty, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

Blocking Queue: enqueue

```
void enqueue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->notFull, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  pthread_cond_signal(&queue->notEmpty, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

wait until queue has space

Blocking Queue: enqueue

```
void enqueue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->notFull, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  pthread_cond_signal(&queue->notEmpty, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

queue has space!
insert element

Blocking Queue: enqueue

```
void enqeue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->notFull, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  pthread_cond_signal(&queue->notEmpty, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

wake up one waiting consumer

Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->notEmpty, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->notFull, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->notEmpty, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->notFull, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

wait until queue is nonempty

Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->notEmpty, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->notFull, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

Queue nonempty!
retrieve next element

Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->notEmpty, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->notFull, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

wake up one waiting producer

Improved enqueue?

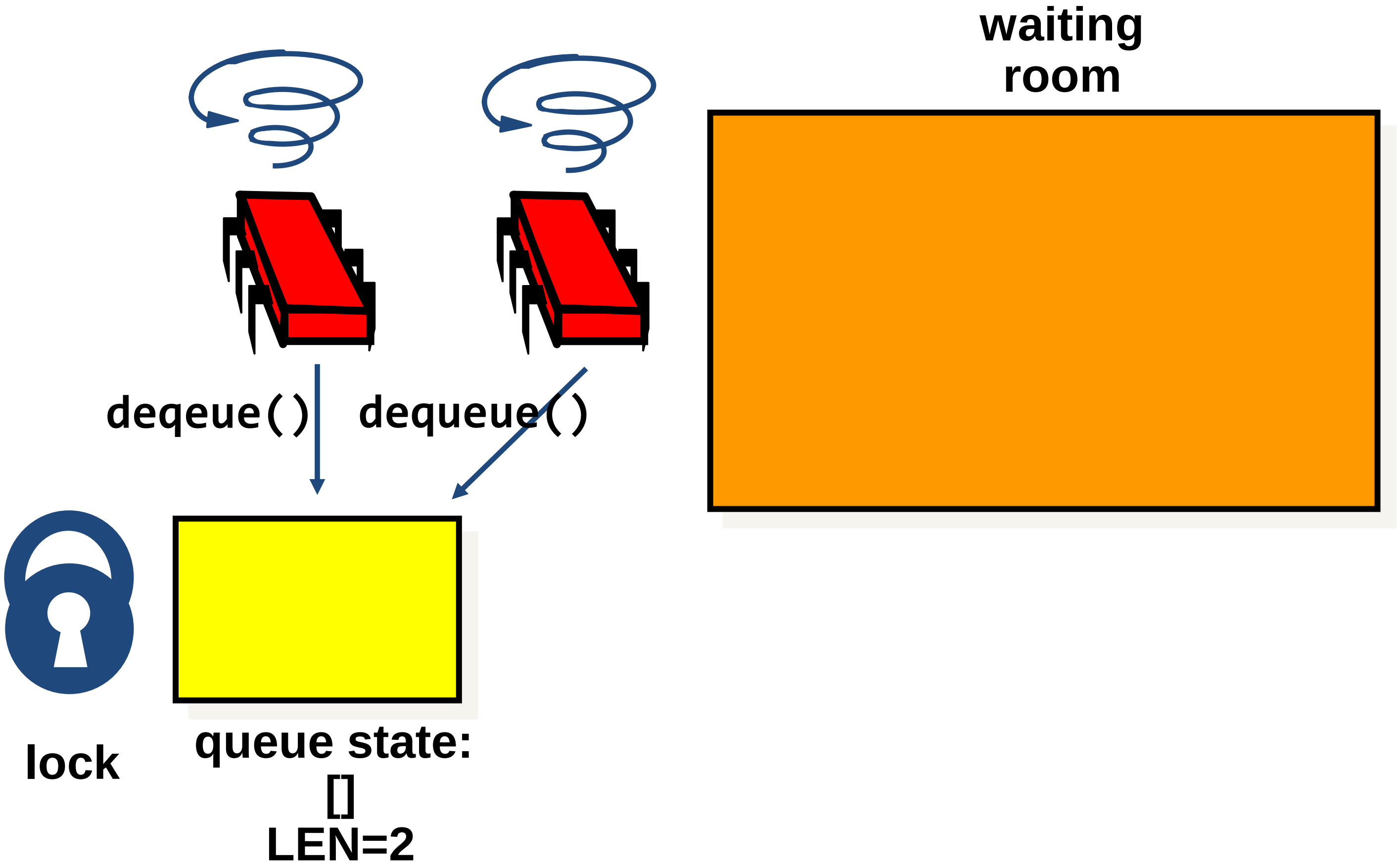
```
void enqueue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->notFull, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  if (queue->count == 1)
    pthread_cond_signal(&queue->notEmpty, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

Improved enqueue?

```
void enqueue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->notFull, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  if (queue->count == 1)
    pthread_cond_signal(&queue->notEmpty, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

lost wakeups

Lost Wakeup



waiting
room

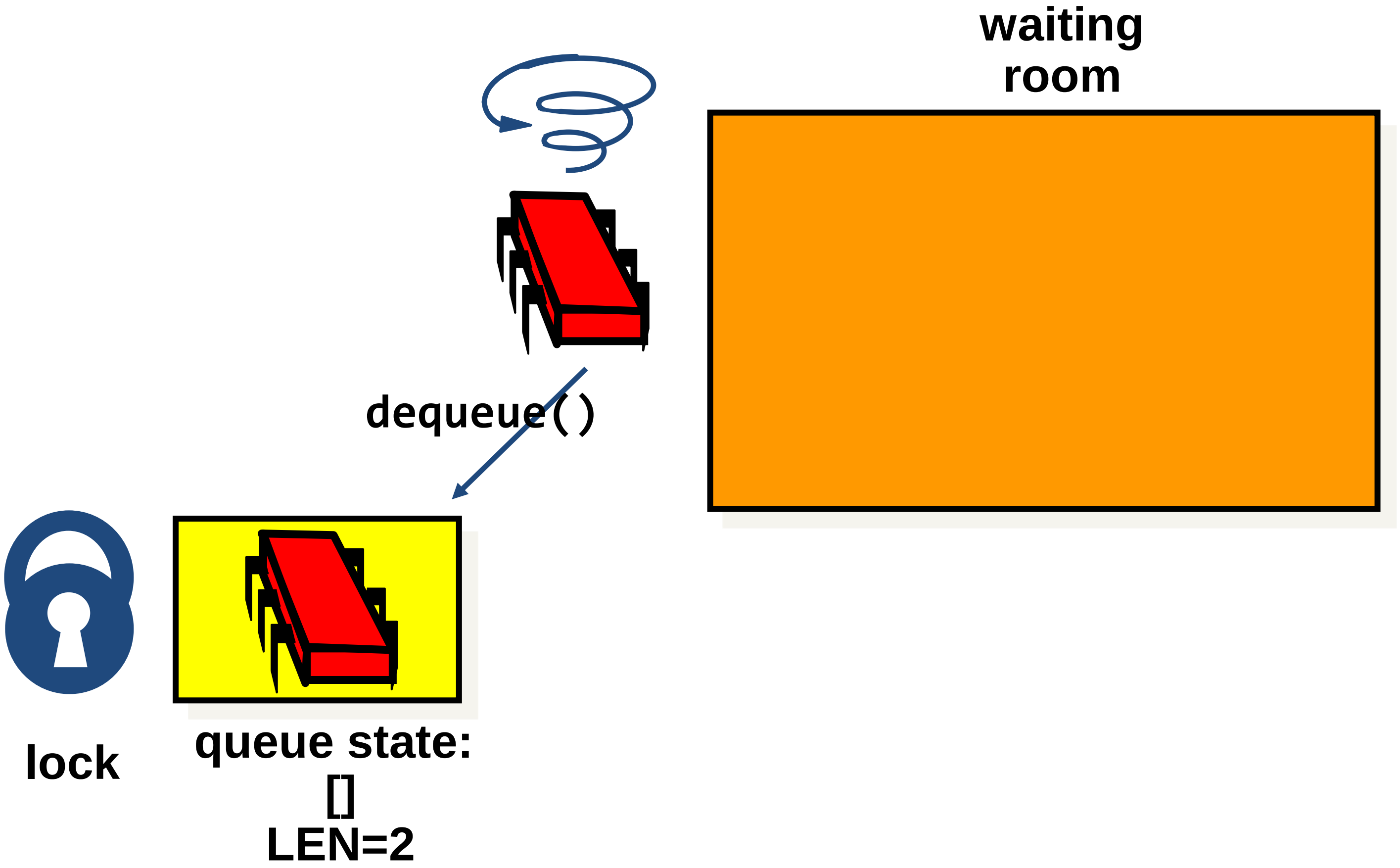
deqeue()

dequeue()

lock

queue state:
[]
LEN=2

Lost Wakeup



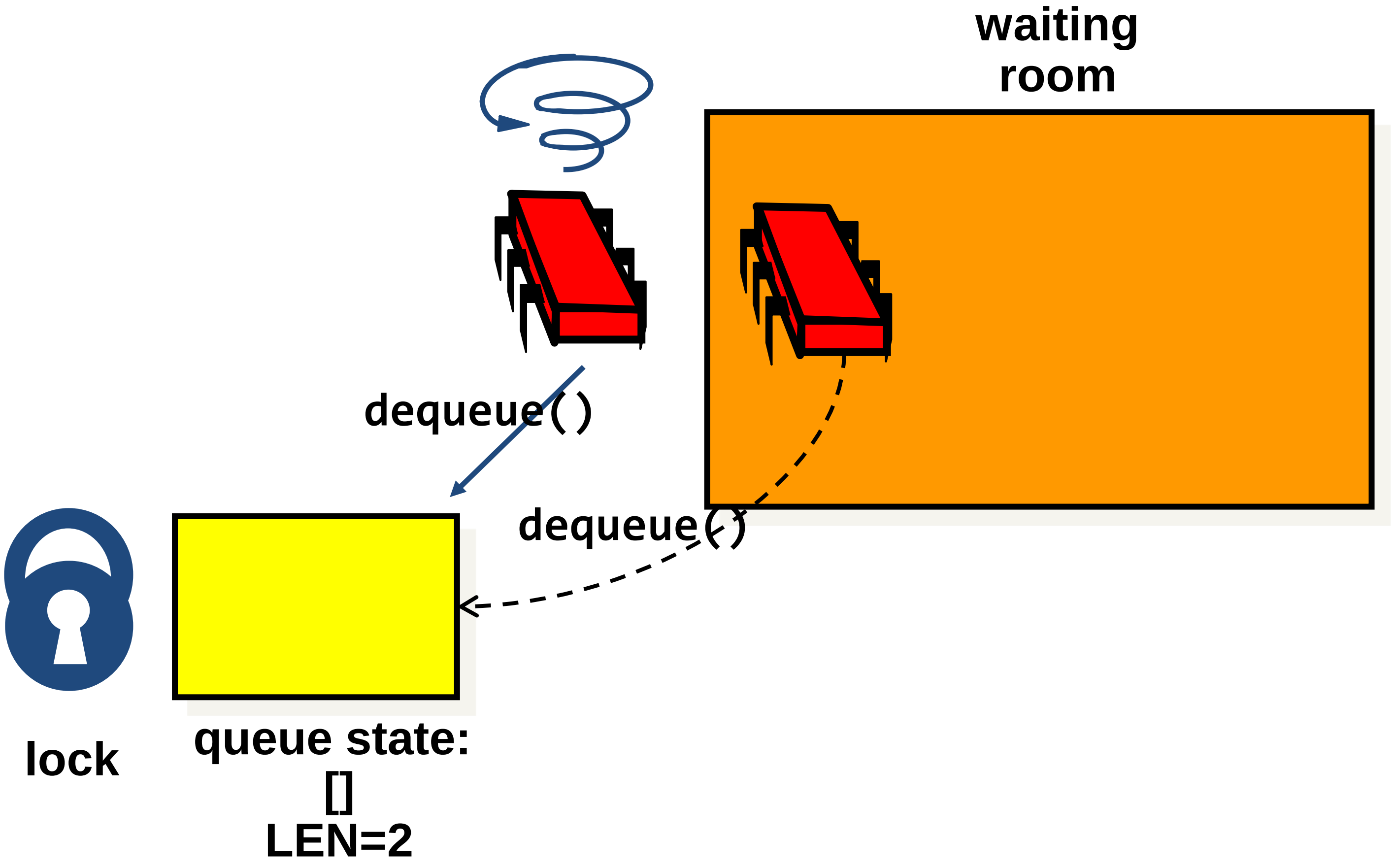
waiting room

dequeue( )

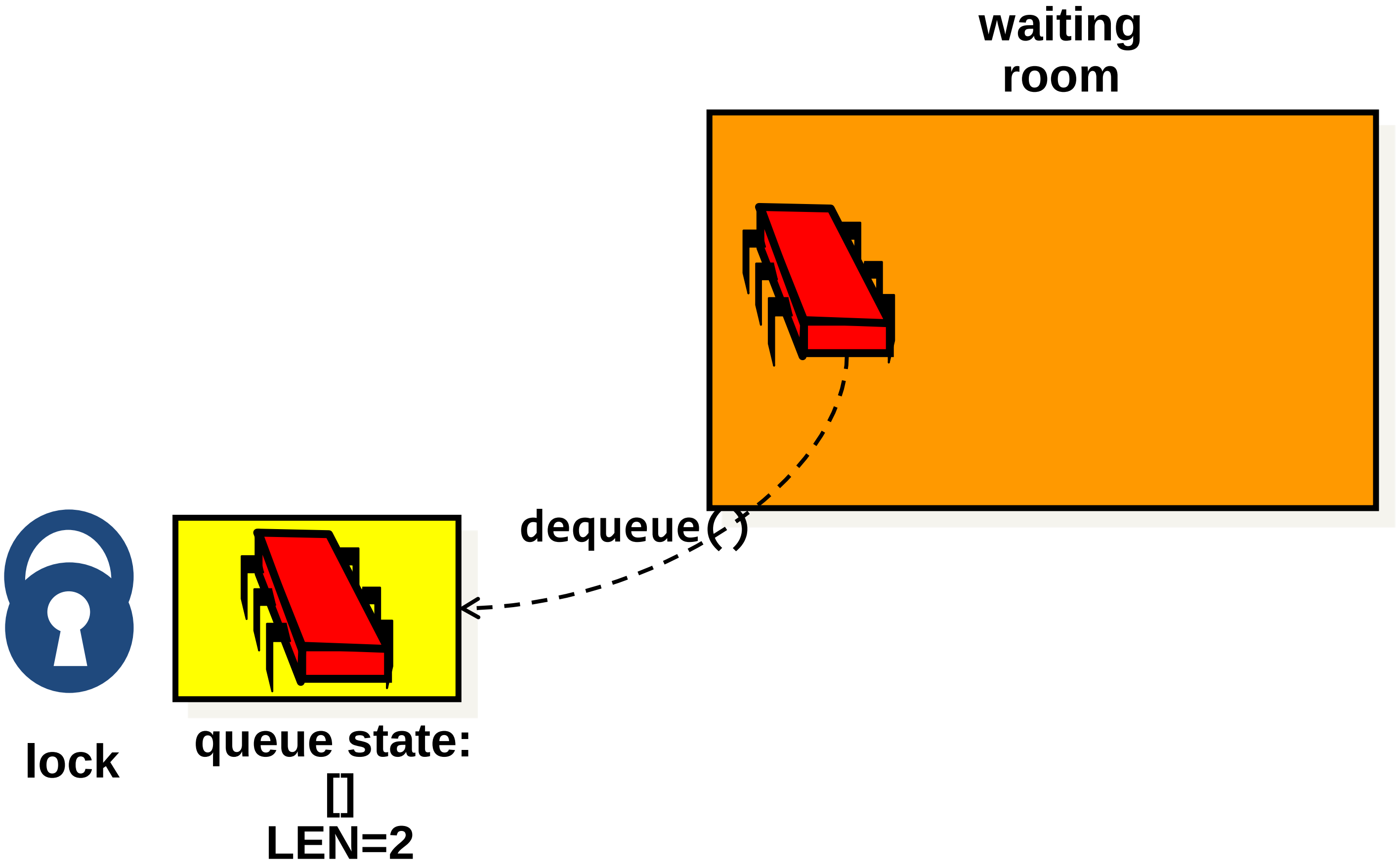
lock

queue state:
[]
LEN=2

Lost Wakeup



Lost Wakeup



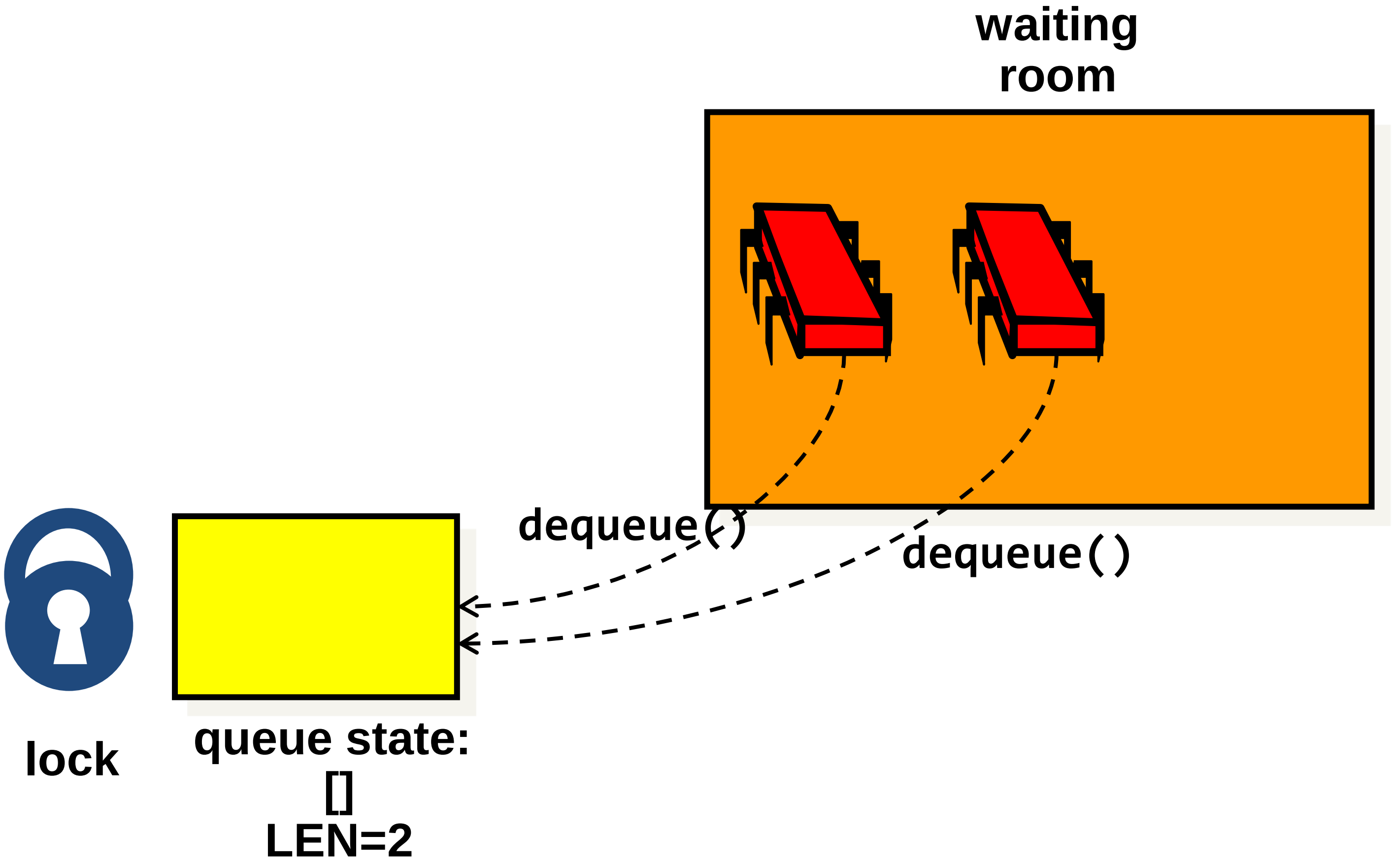
waiting room

dequeue()

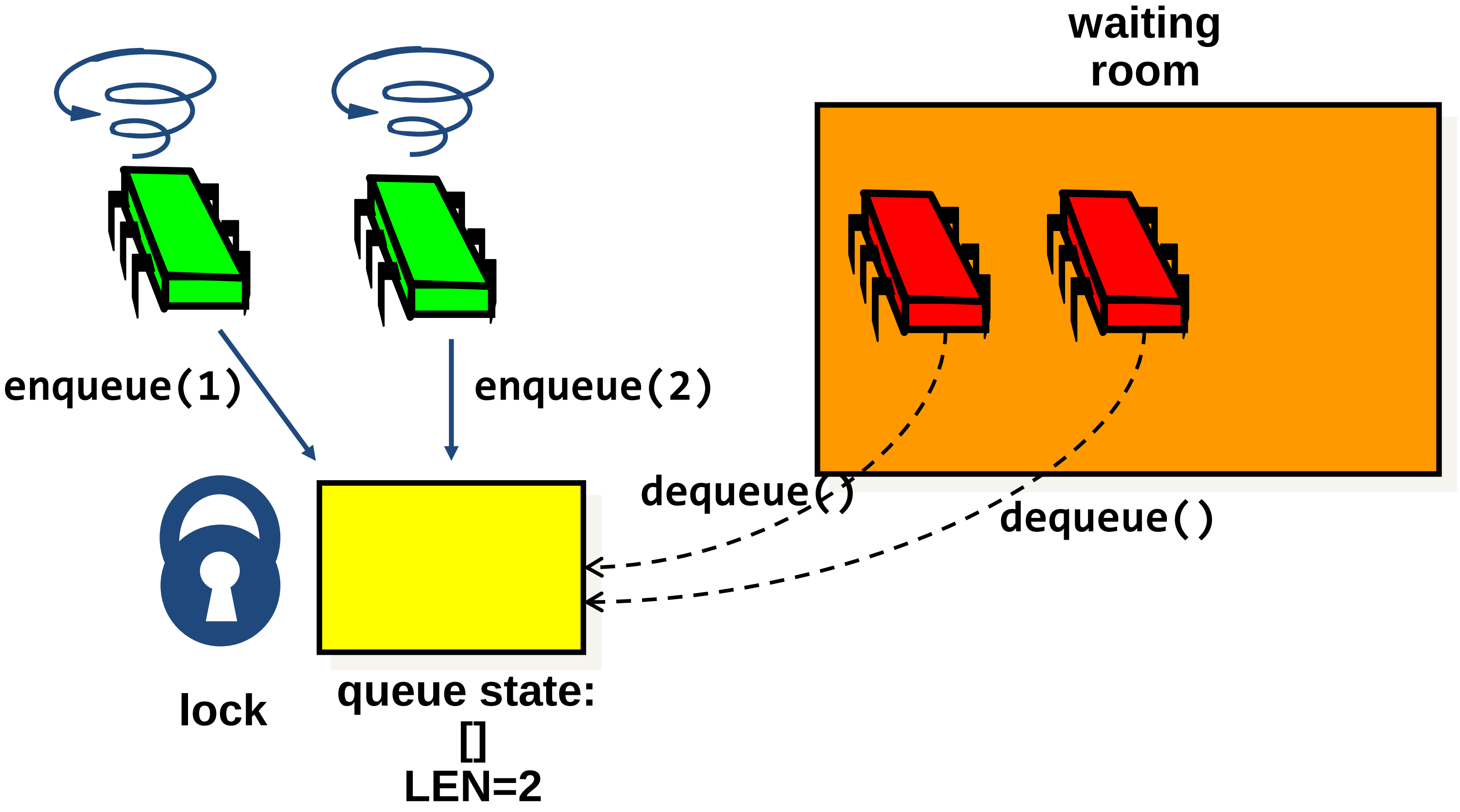
lock

queue state:
[]
LEN=2

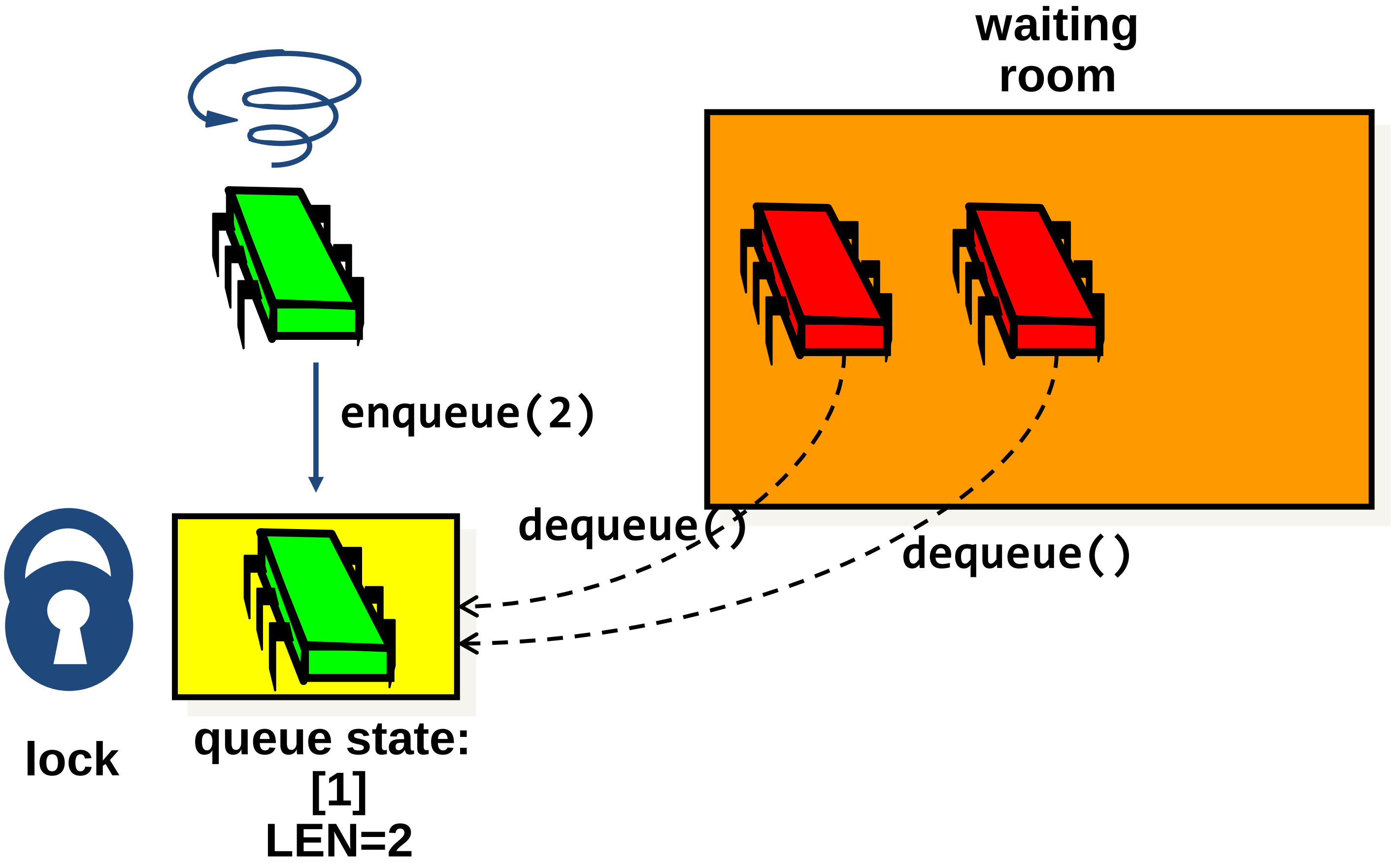
Lost Wakeup



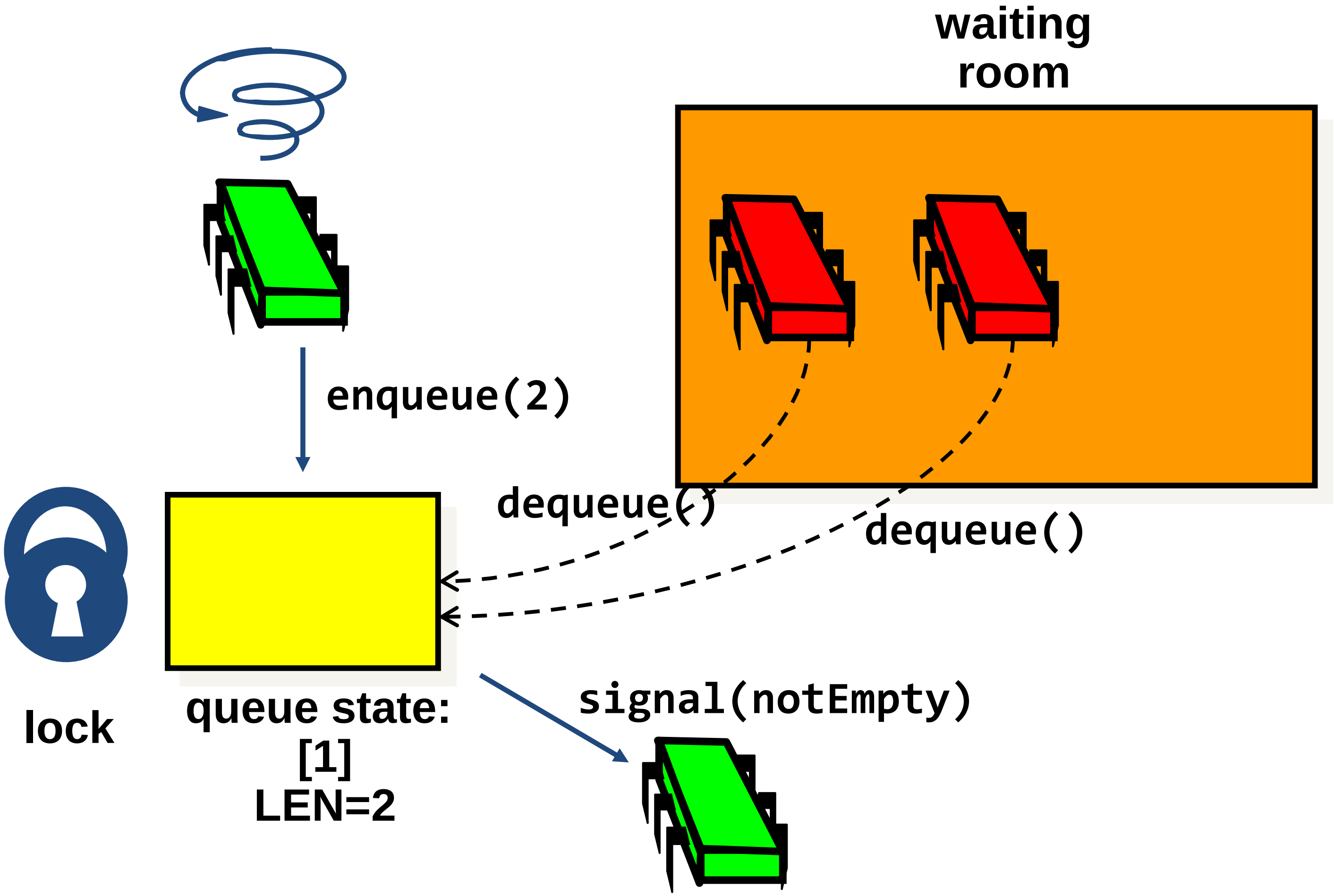
Lost Wakeup


waiting
room
enqueue(1)
enqueue(2)
dequeue()
dequeue()
lock
queue state:
[]
LEN=2

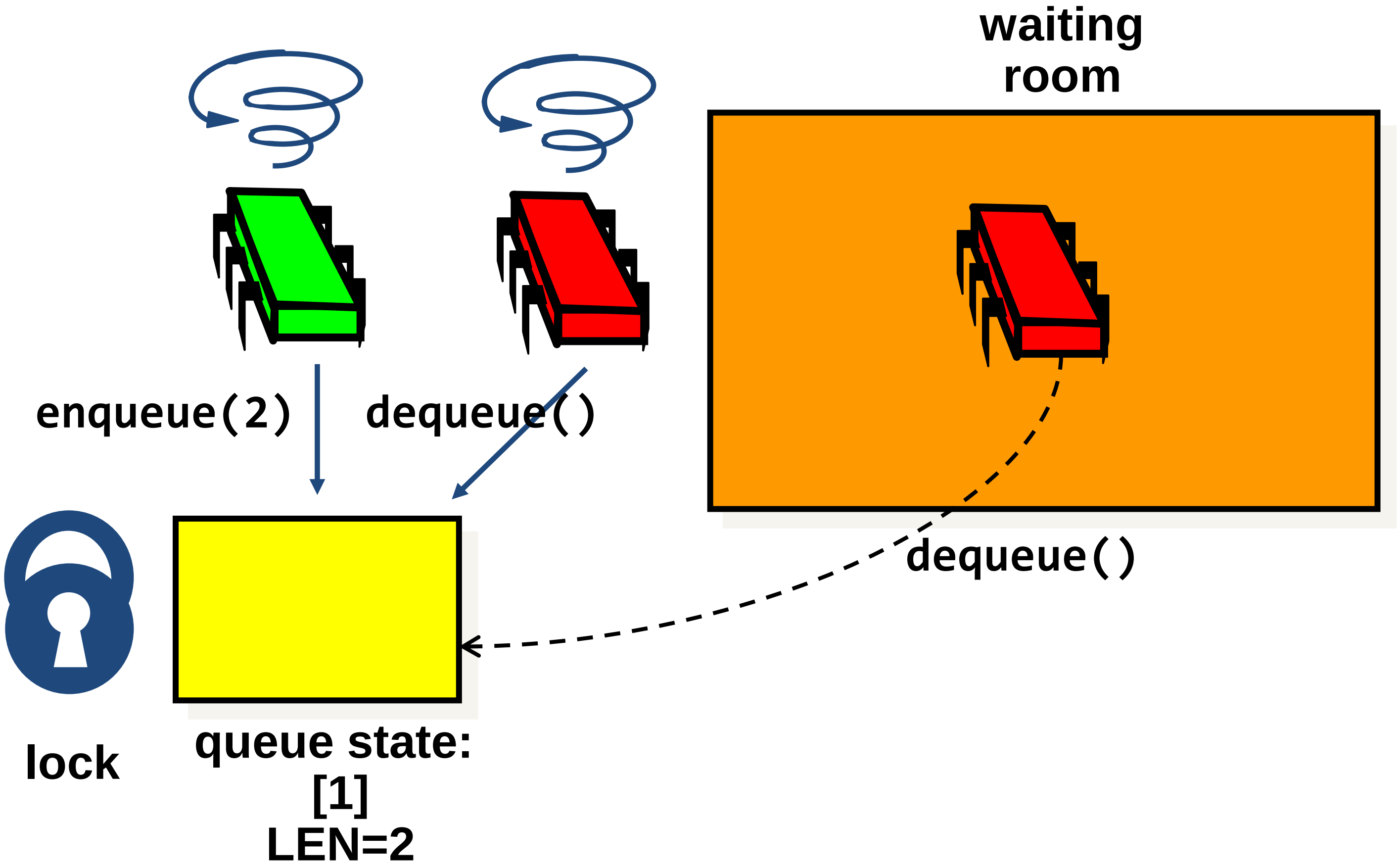
Lost Wakeup


waiting
room
enqueue(2)
dequeue()
dequeue()
lock
queue state:
[1]
LEN=2

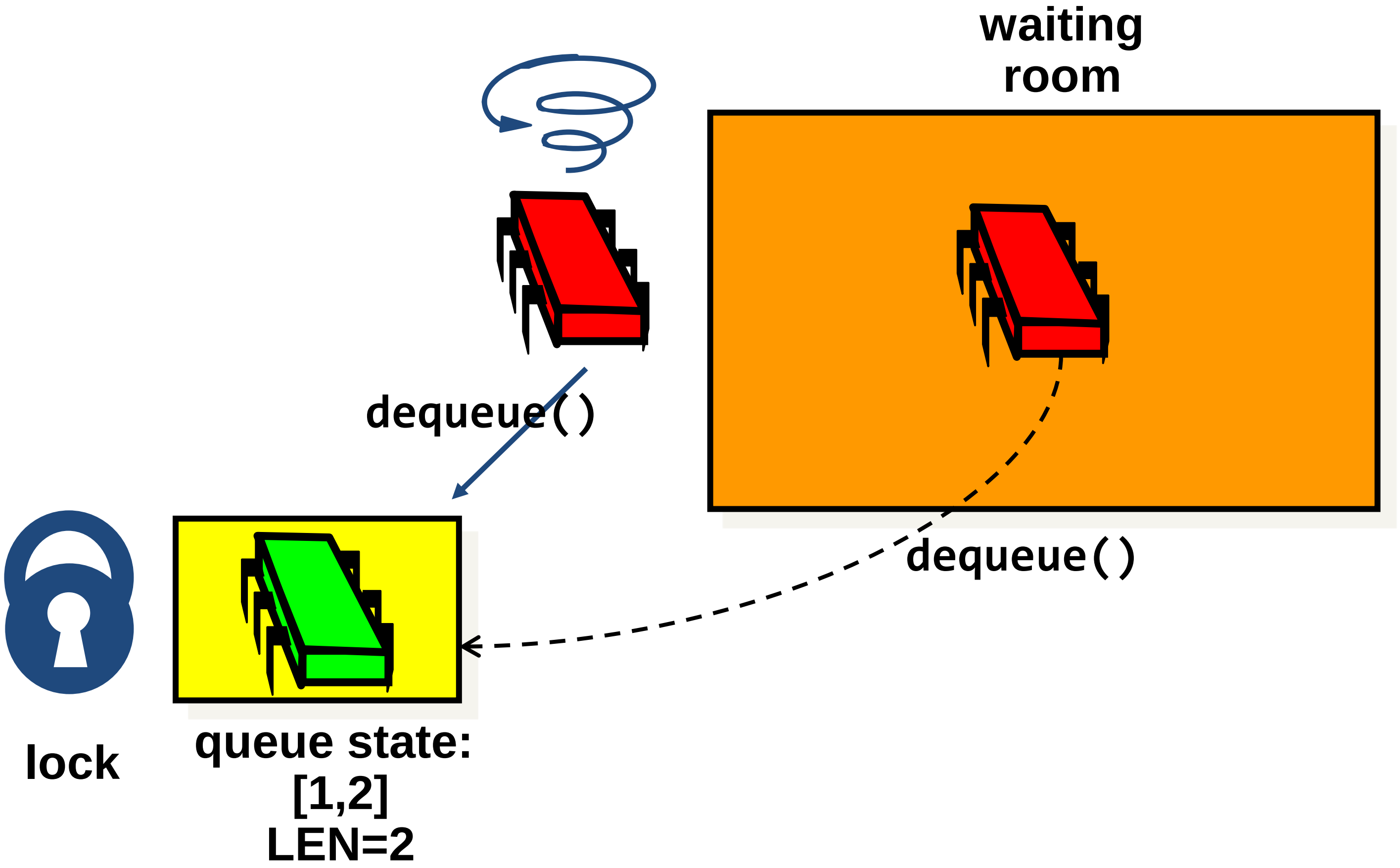
Lost Wakeup


waiting room
enqueue(2)
dequeue()
dequeue()
lock
queue state:
[1]
LEN=2
signal(notEmpty)

Lost Wakeup


waiting
room
enqueue(2)
dequeue()
dequeue()
lock
queue state:
[1]
LEN=2

Lost Wakeup



waiting room

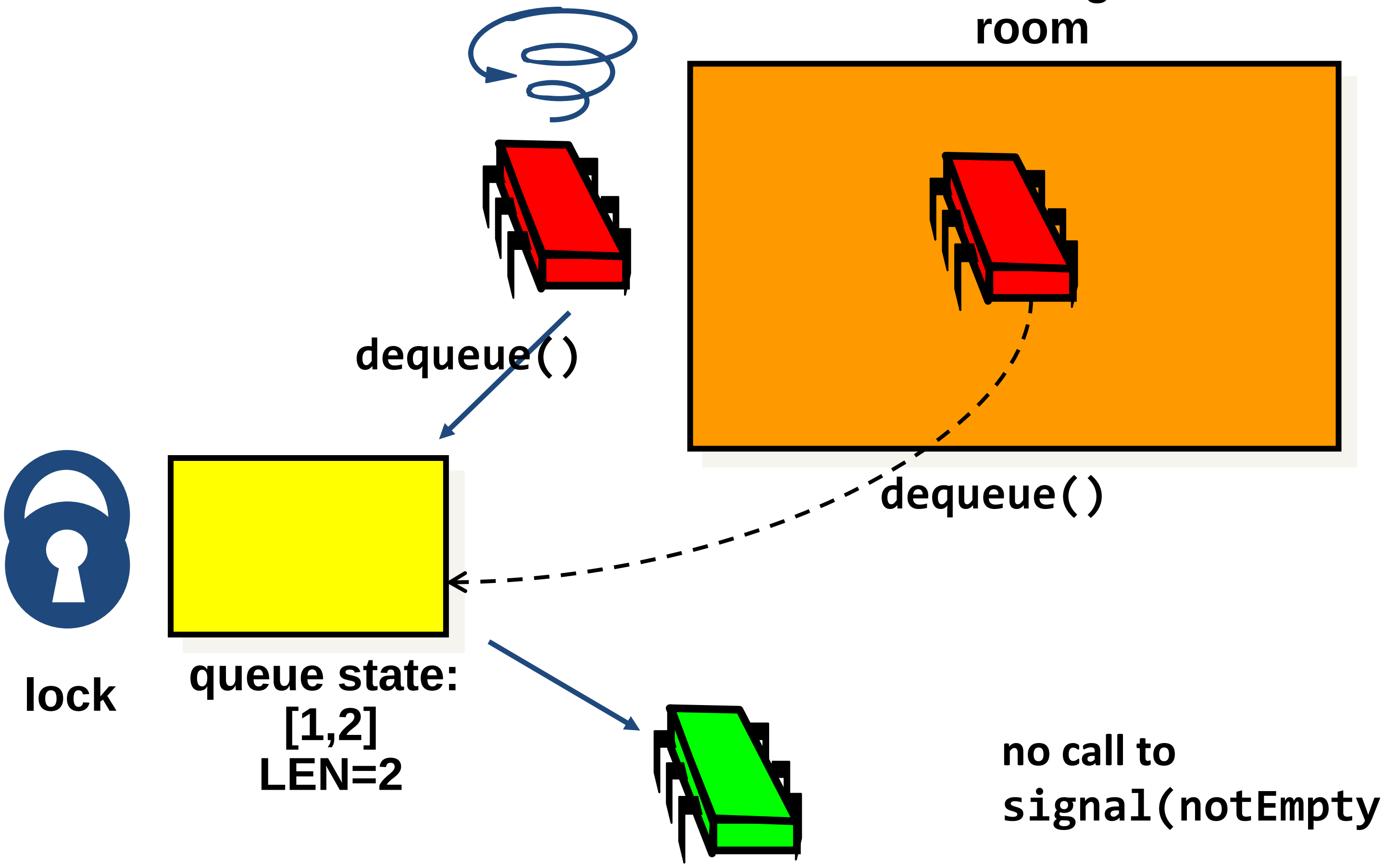
dequeue()

dequeue()

lock

queue state: [1,2] LEN=2

Lost Wakeup



waiting room

dequeue()

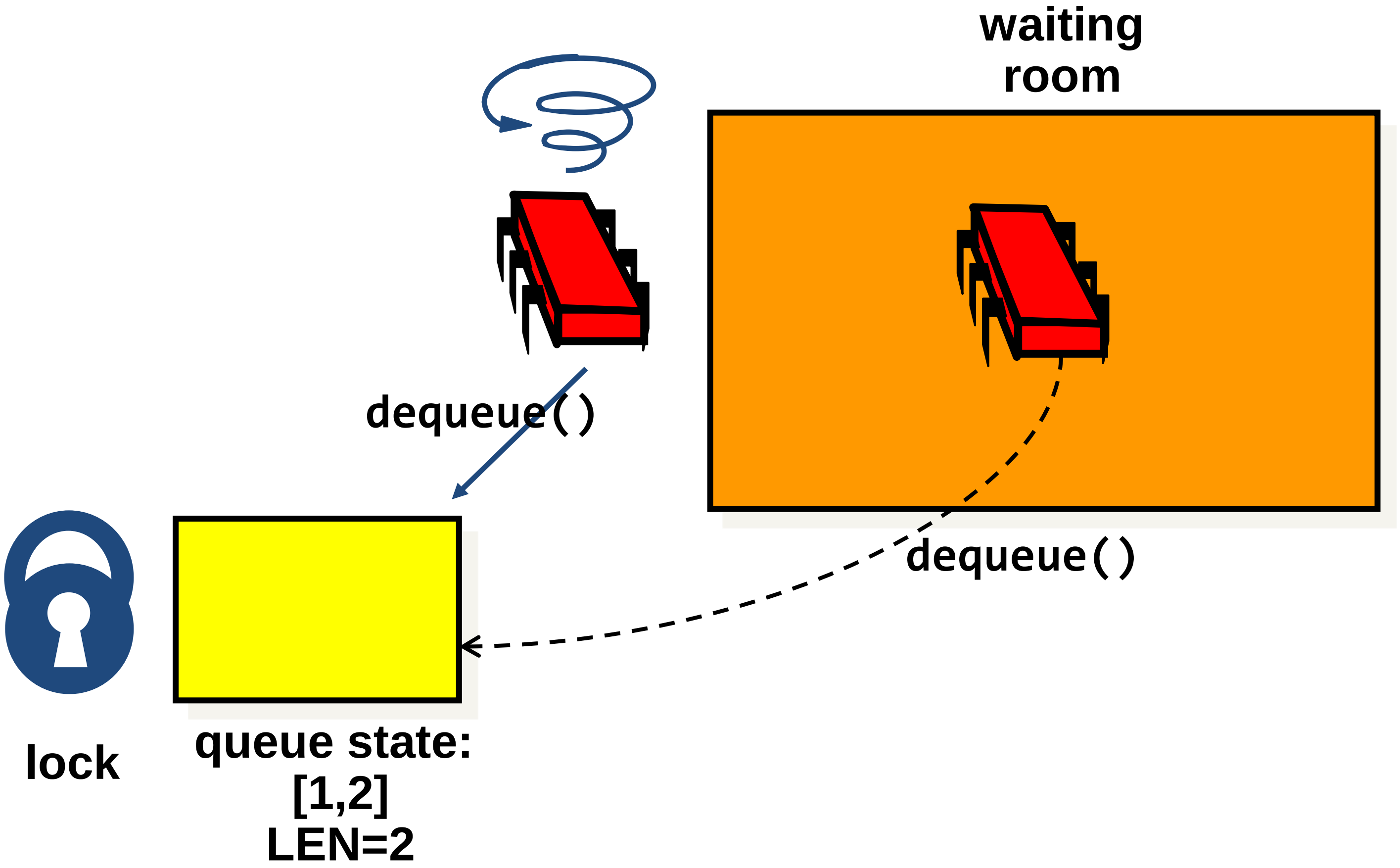
dequeue()

lock

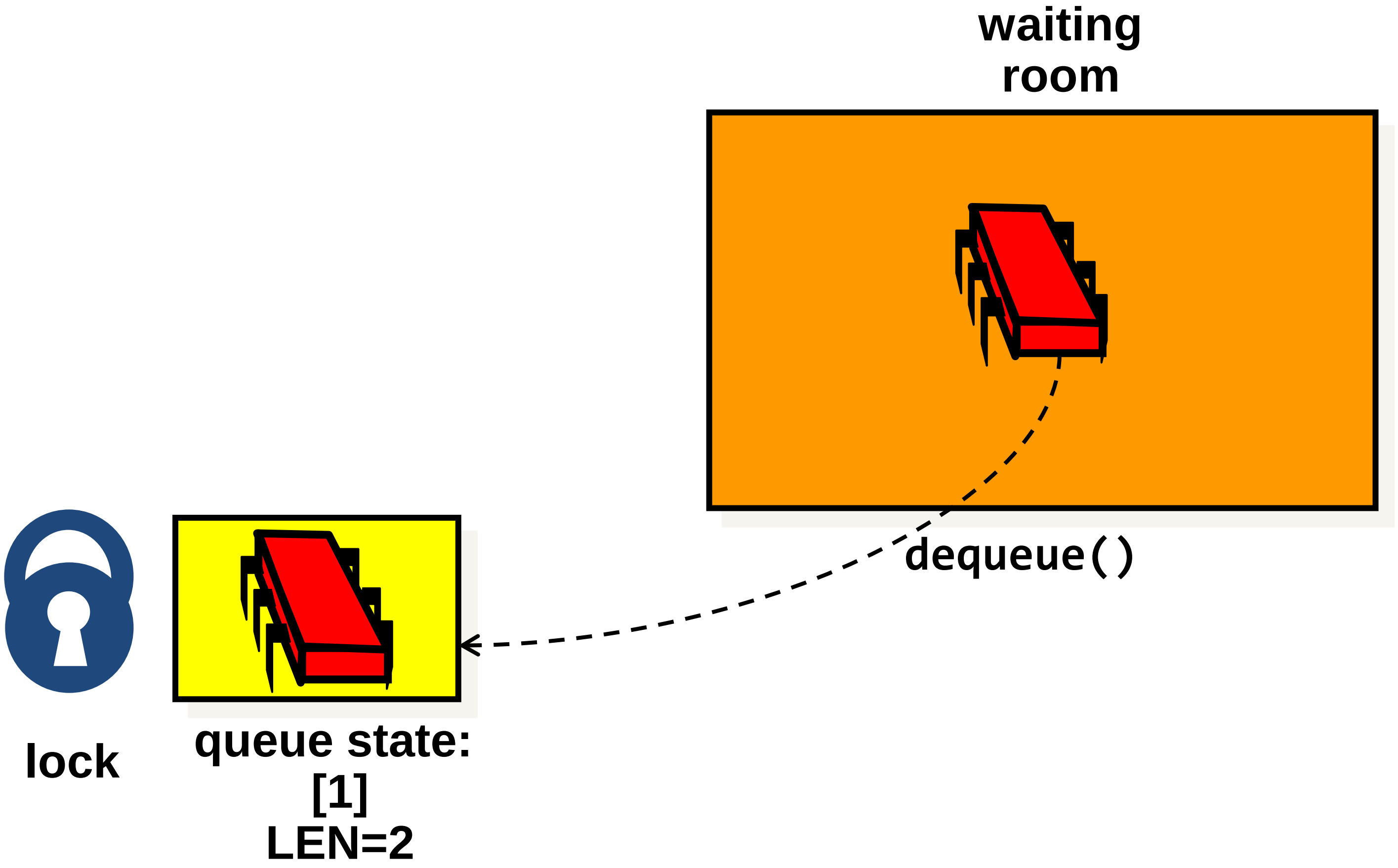
queue state:
[1,2]
LEN=2

no call to
`signal(notEmpty)!`

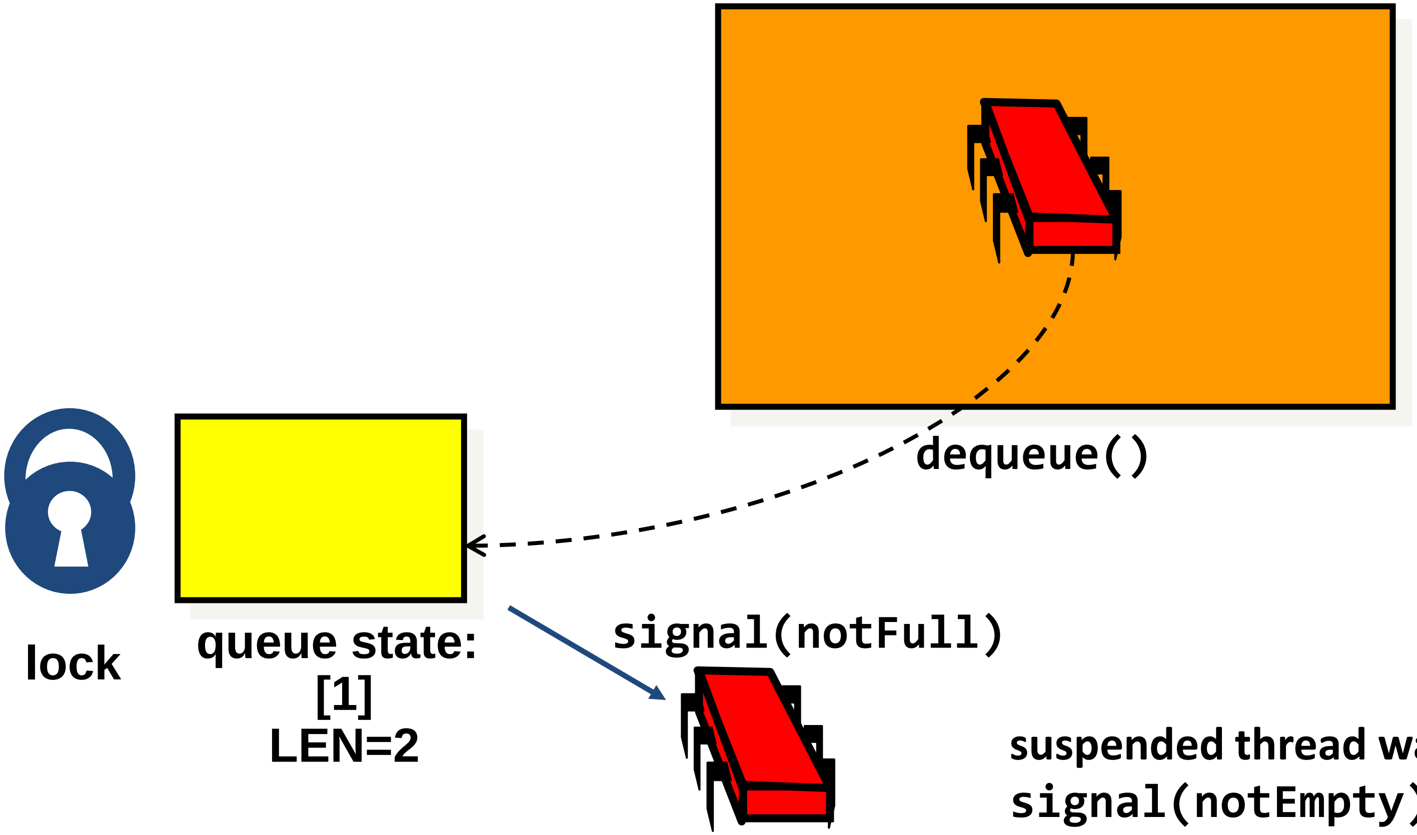
Lost Wakeup



Lost Wakeup



Lost Wakeup



waiting room

dequeue()

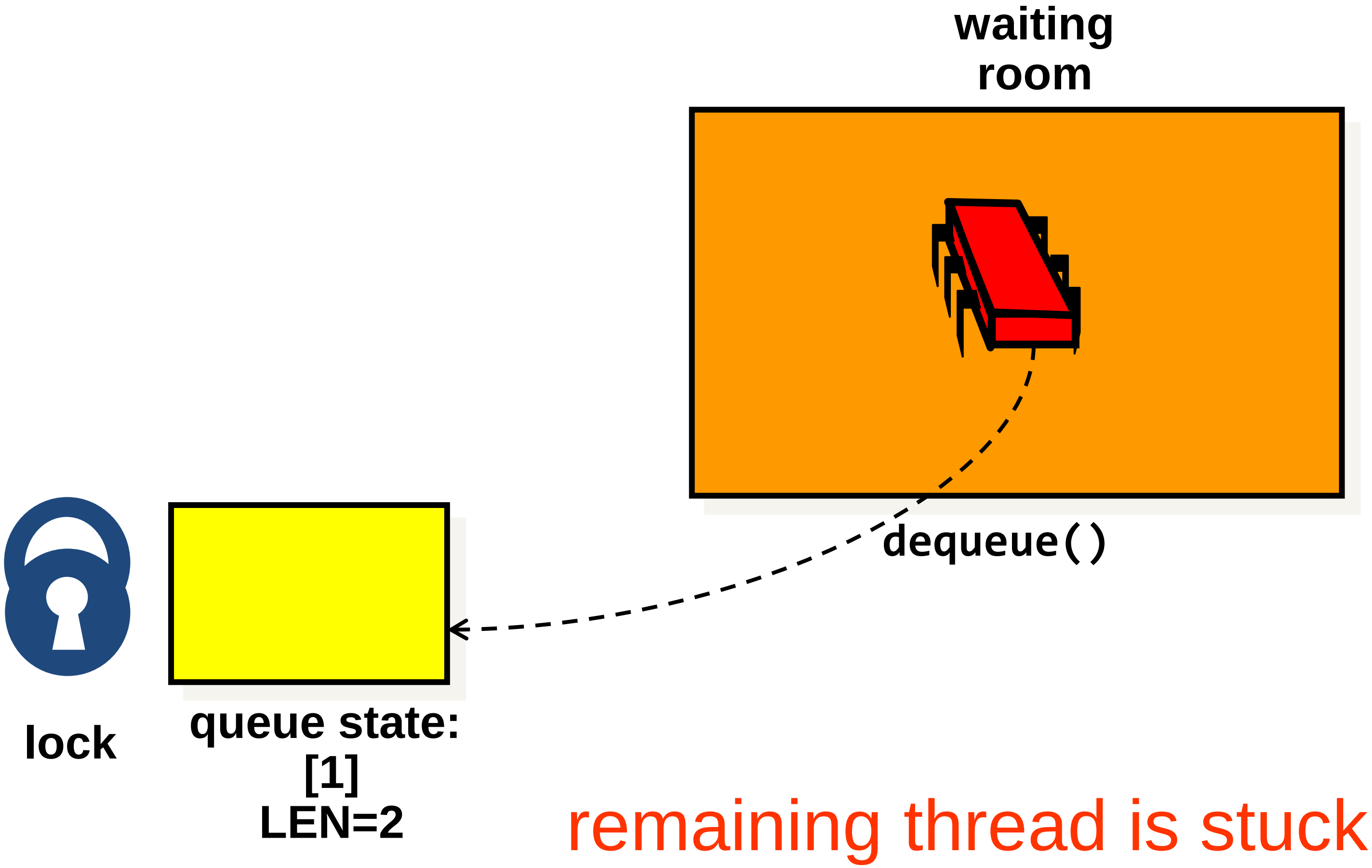
lock

queue state:
[1]
LEN=2

signal(notFull)

suspended thread waits for `signal(notEmpty)`!

Lost Wakeup



remaining thread is stuck!

The Lost-Wakeup Problem

- Condition variables are inherently vulnerable to lost wakeups
  - one thread waits forever without realizing that its waiting condition has become true
- Programming practices
  - if in doubt, broadcast to **all** waiting processes
  - specify a timeout when waiting

Simplified Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t cond;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

Simplified Blocking Queue

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t cond;
  int items[LEN];
  int tail, head, count;
} queue_t;
```

Simplified Blocking Queue: enqueue

```
void enqeue(queue_t *queue, int x) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == LEN)
    pthread_cond_wait(&queue->cond, &queue->mu);
  queue->items[queue->tail] = x;
  if (++queue->tail == LEN) queue->tail = 0;
  ++queue->count;
  pthread_cond_broadcast(&queue->cond, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
}
```

Simplified Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->cond, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_broadcast(&queue->cond, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

Simplified Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->cond, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->cond, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

Simplified Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->cond, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->cond, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

enough?

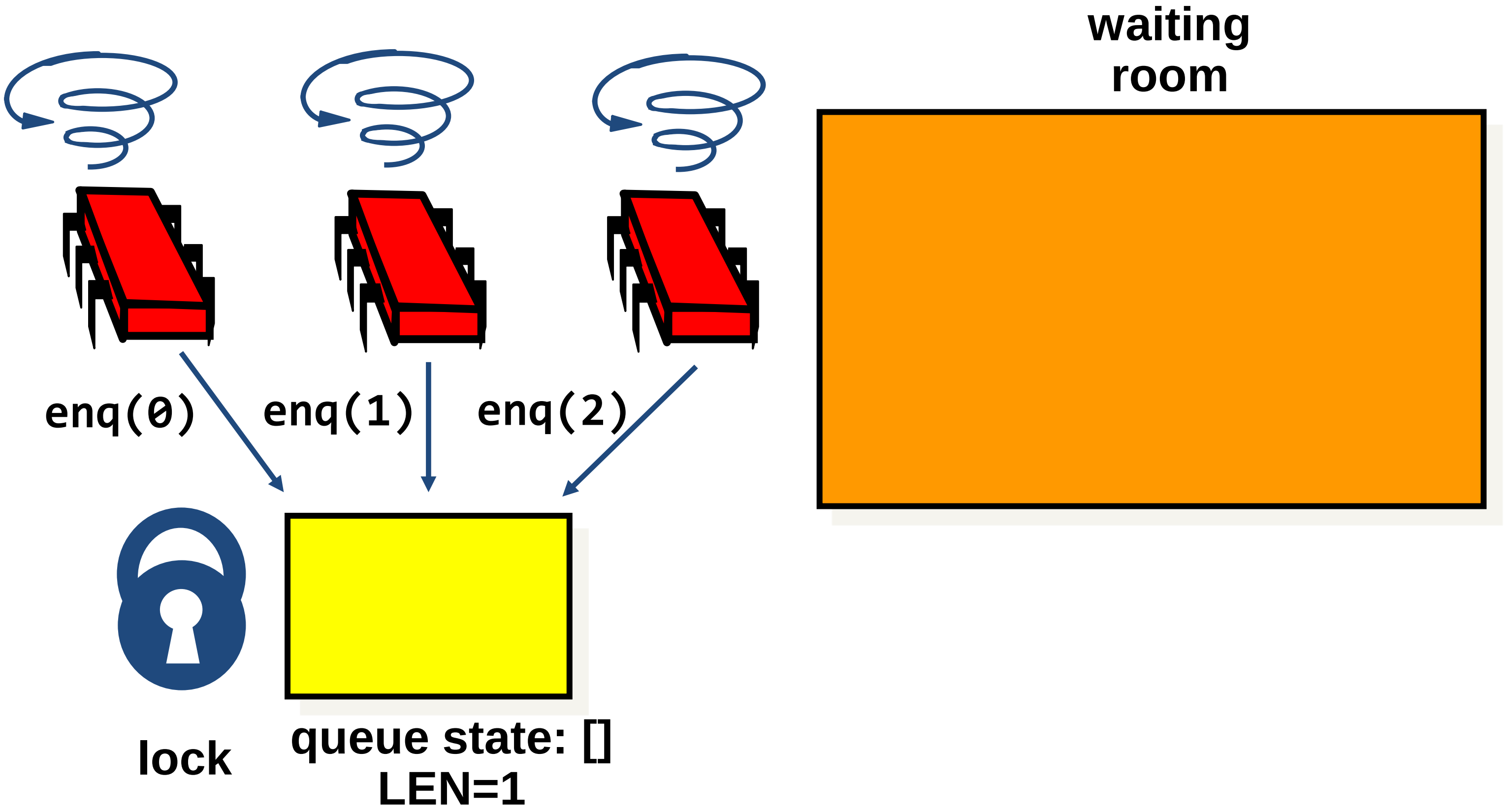
Simplified Blocking Queue: dequeue

```
int deqeue(queue_t *queue) {
  pthread_mutex_lock(&queue->mu);
  while (queue->count == 0)
    pthread_cond_wait(&queue->cond, &queue->mu);
  int x = queue->items[queue->head];
  if (++queue->head == LEN) queue->head = 0;
  --queue->count;
  pthread_cond_signal(&queue->cond, &queue->mu);
  pthread_mutex_unlock(&queue->mu);
  return x;
}
```

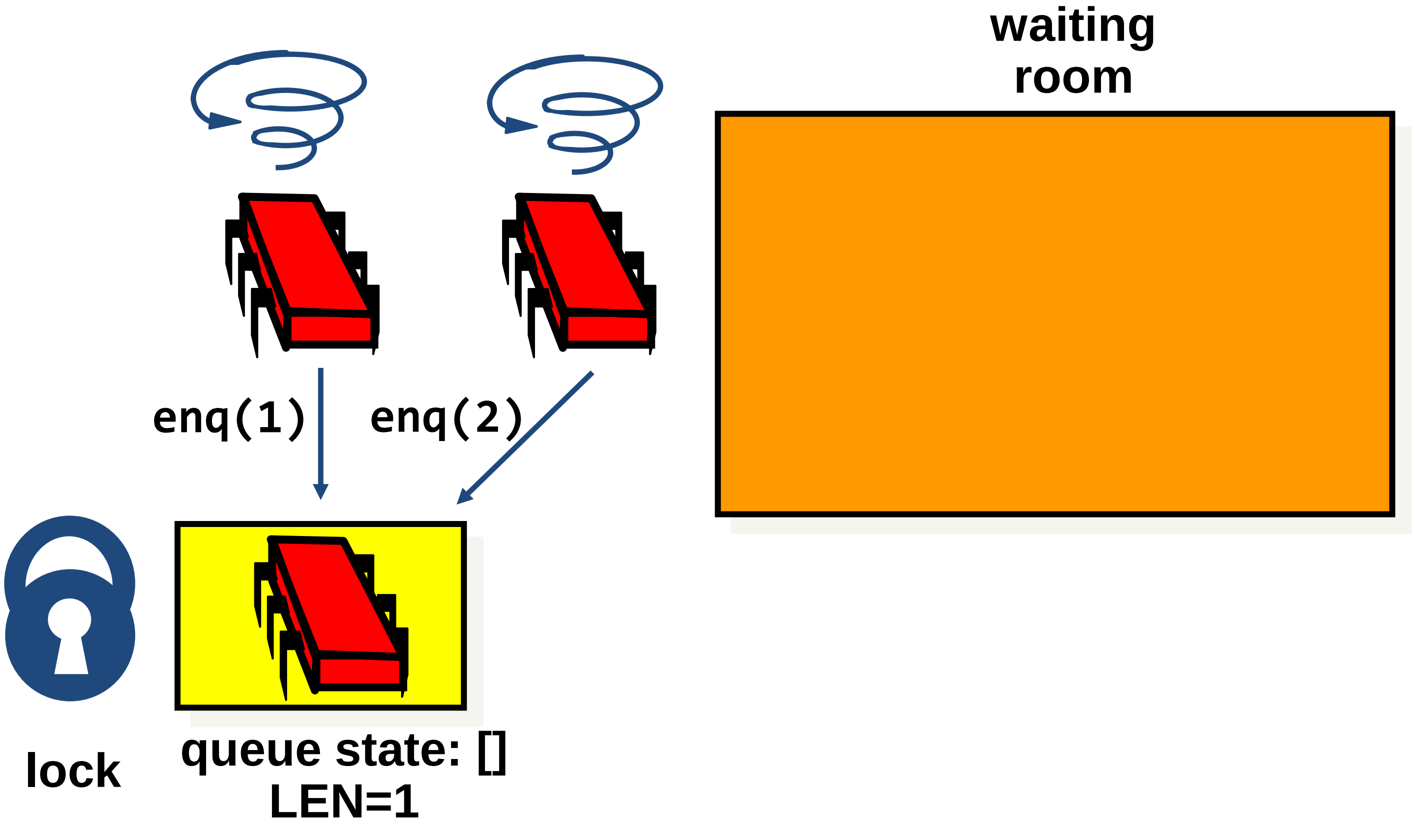
enough?

lost wakeups

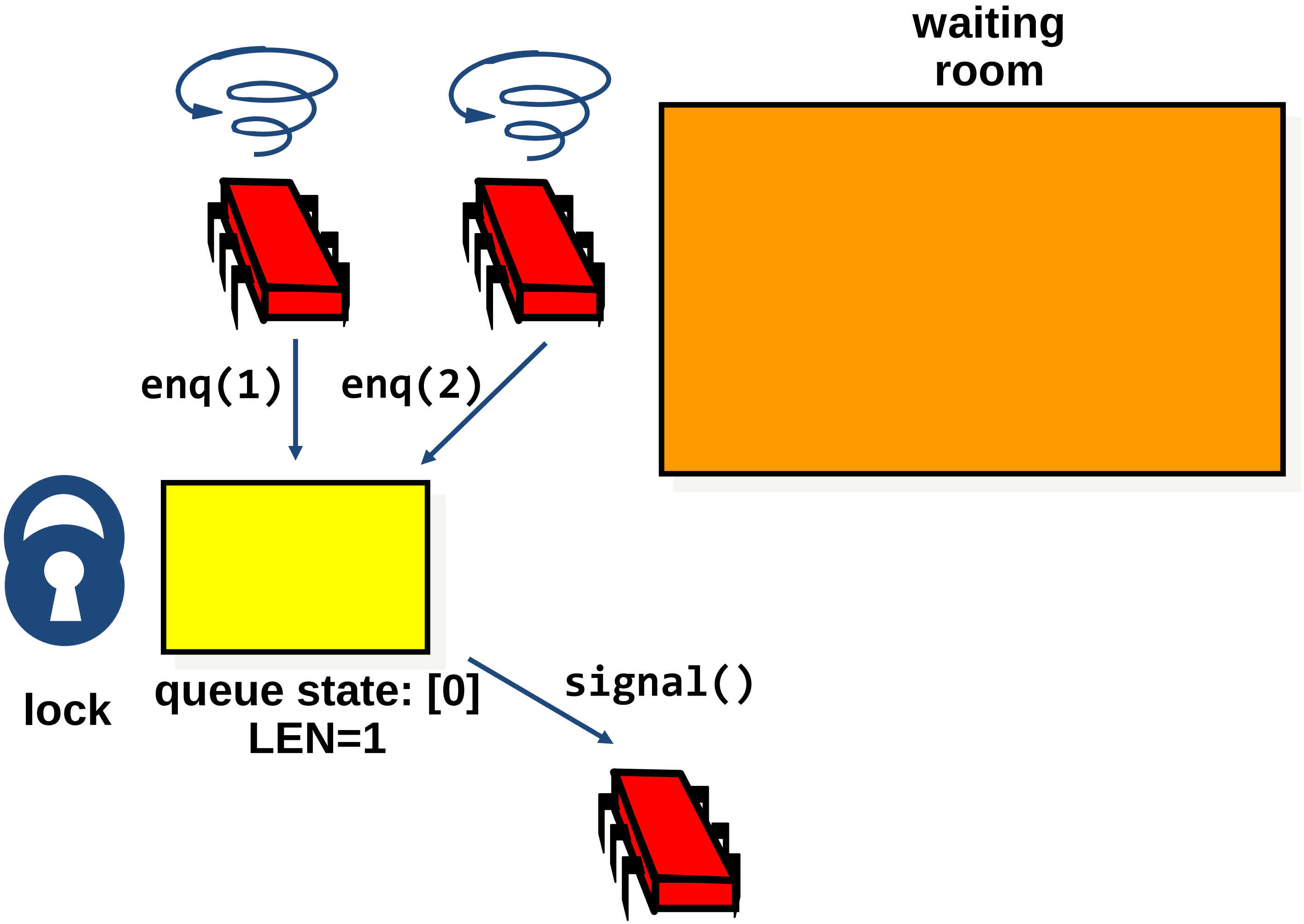
Lost Wakeup in Simplified Queue with `signal()`



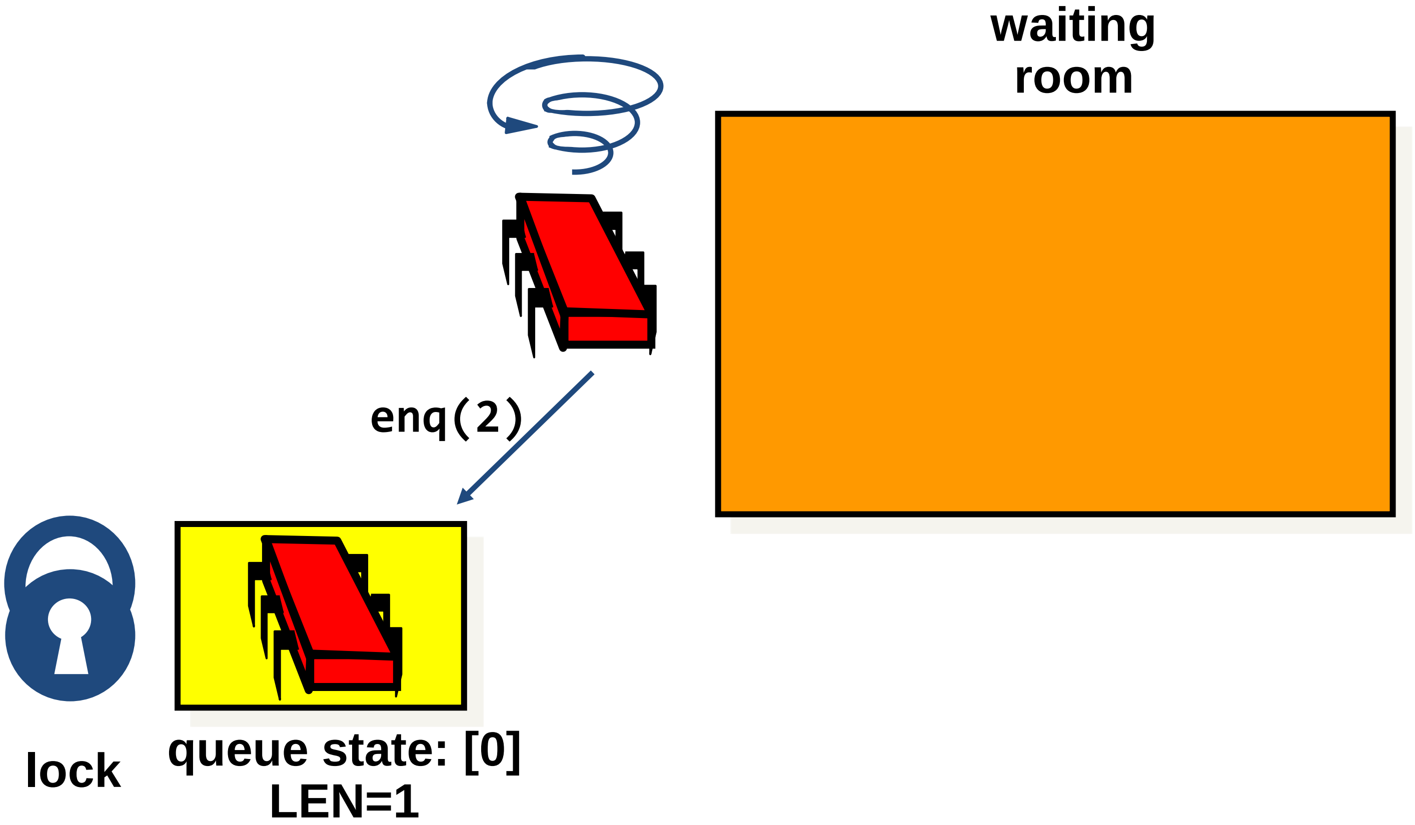
Lost Wakeup in Simplified Queue with `signal()`



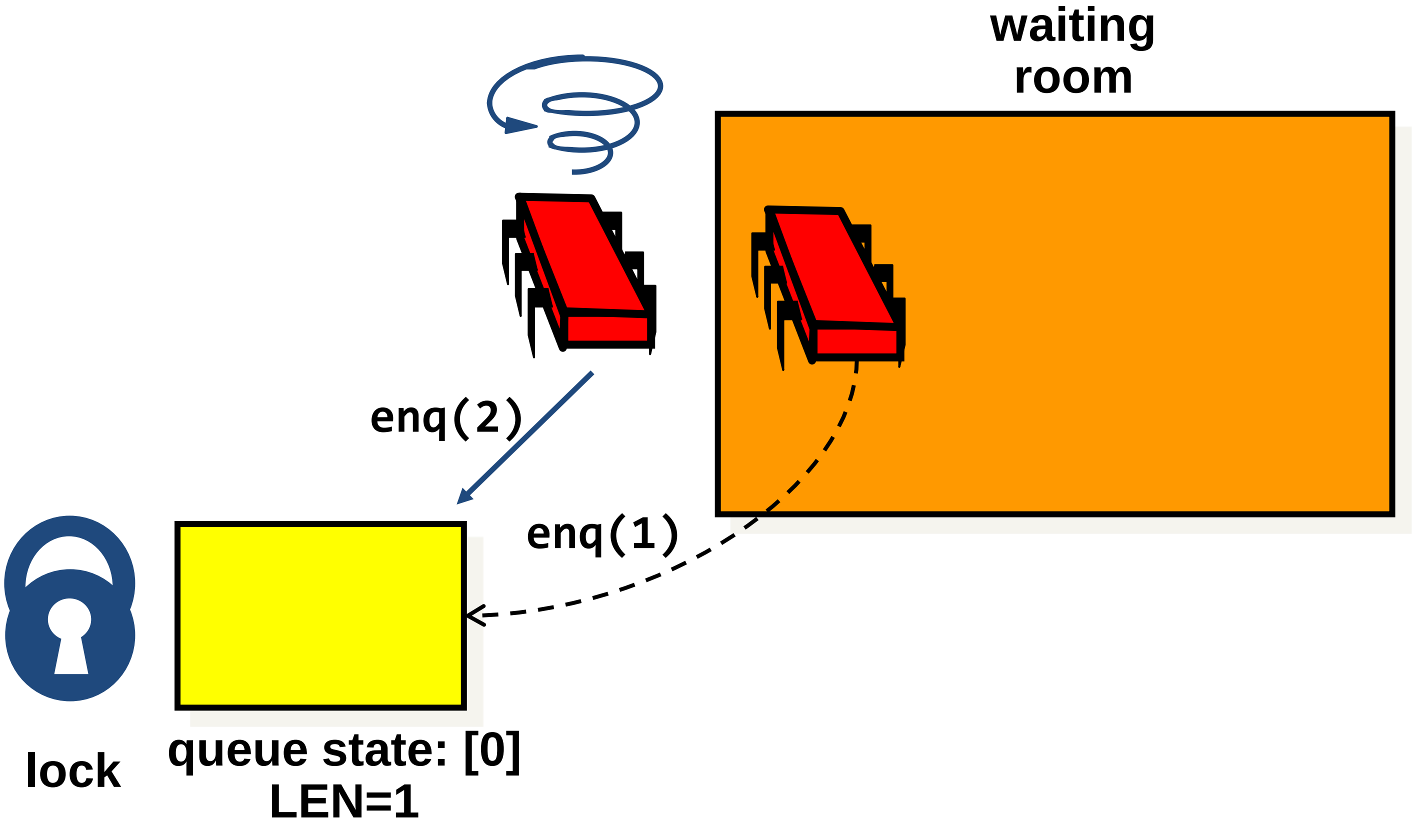
Lost Wakeup in Simplified Queue with `signal()`



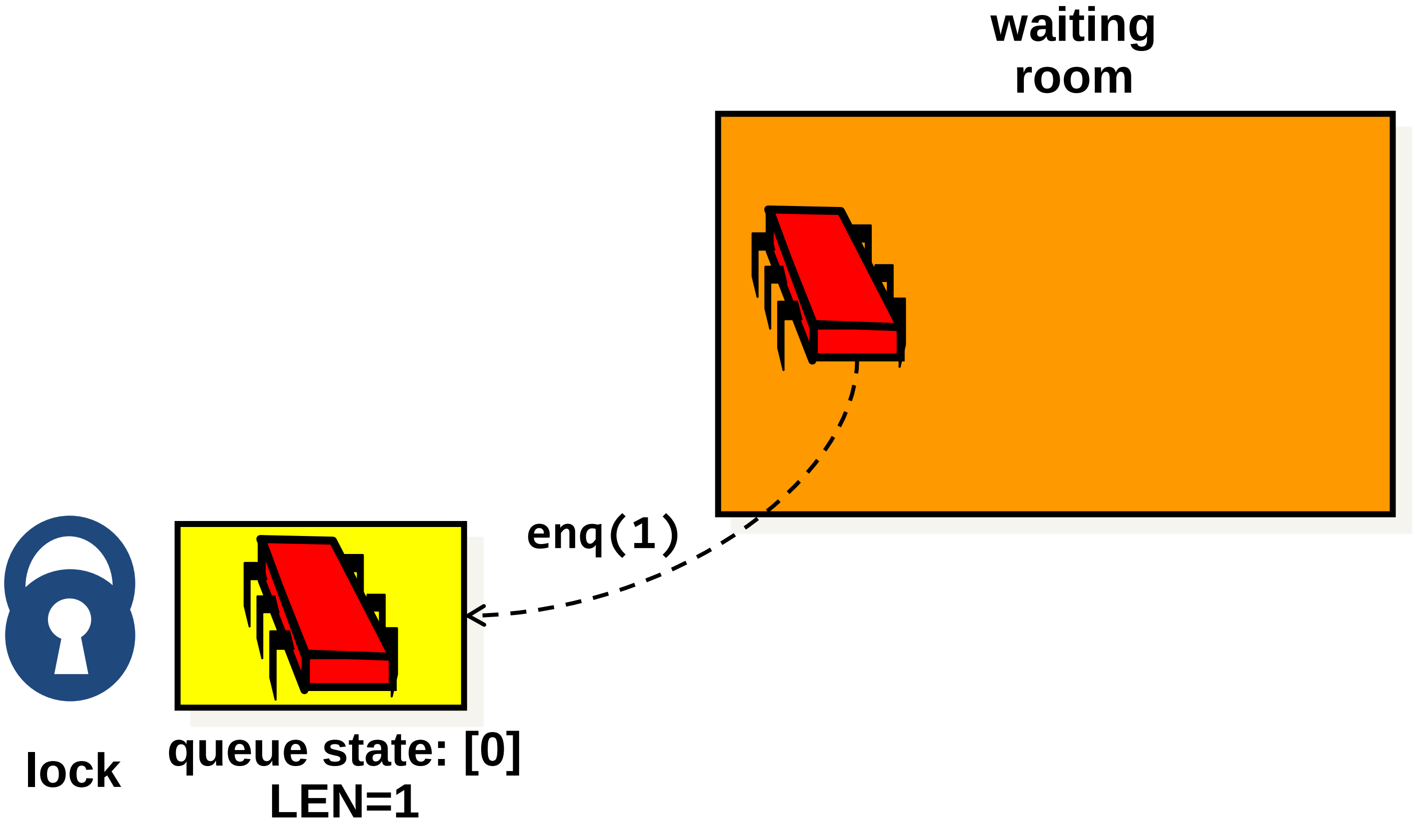
Lost Wakeup in Simplified Queue with `signal()`



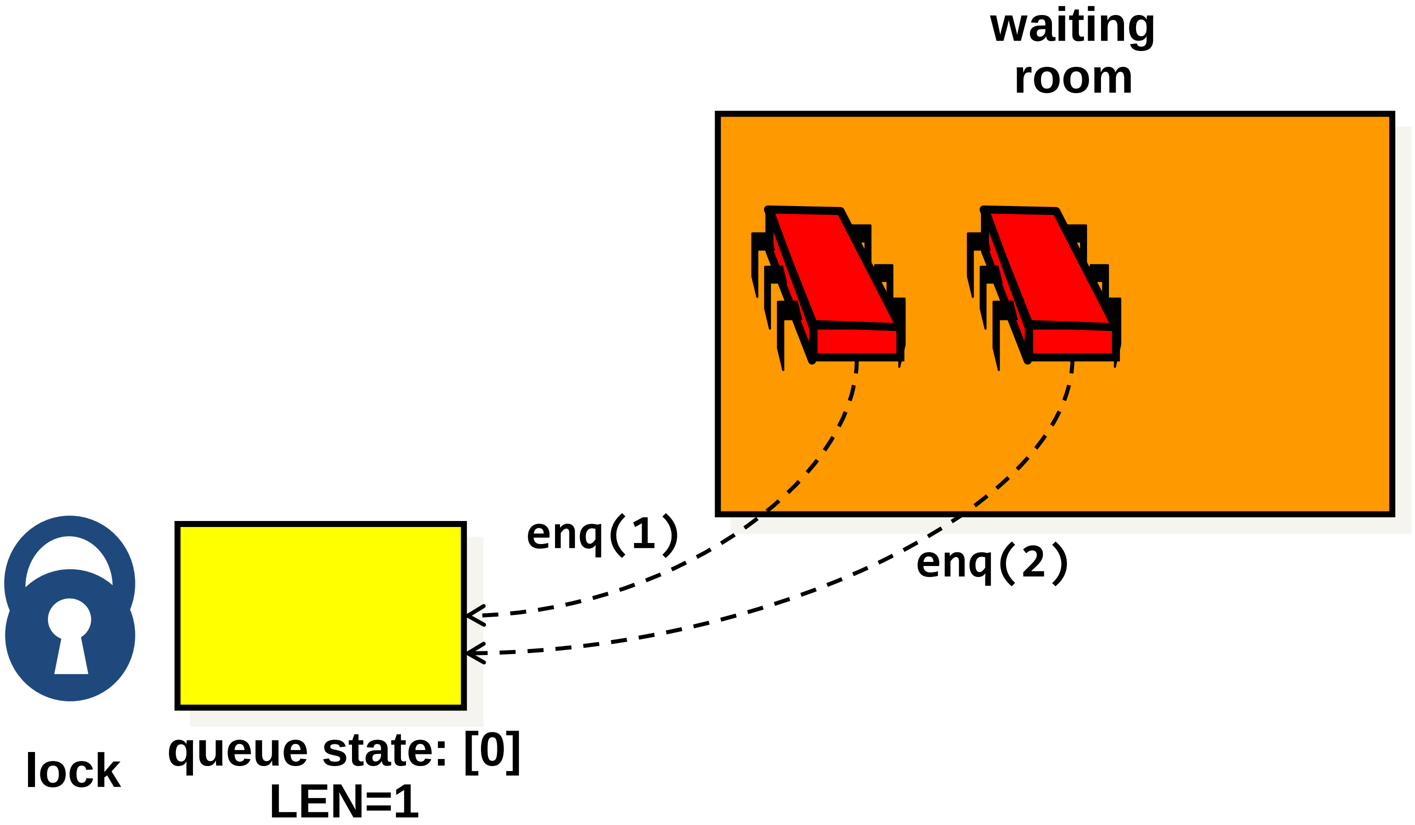
Lost Wakeup in Simplified Queue with `signal()`



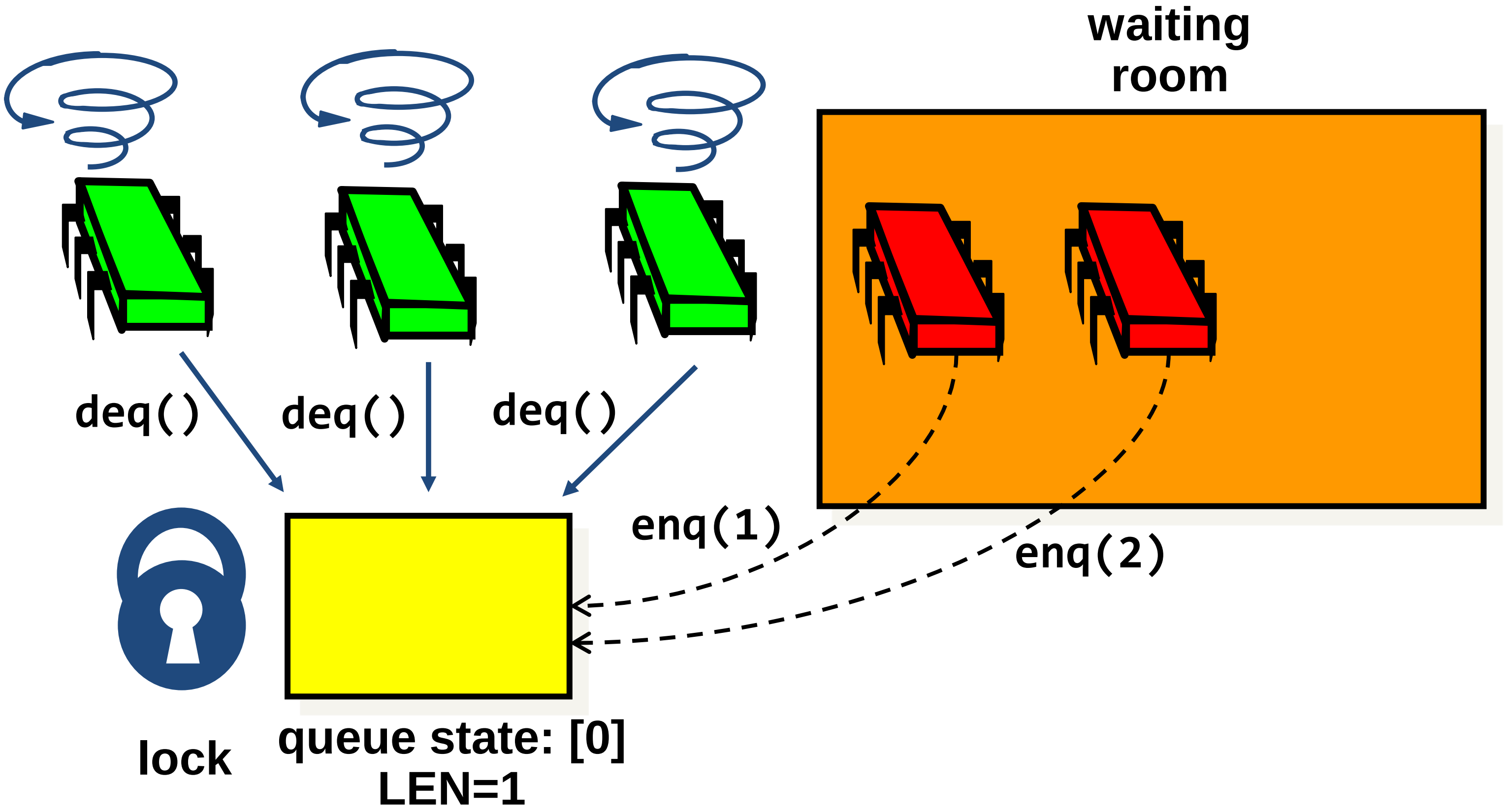
Lost Wakeup in Simplified Queue with `signal()`



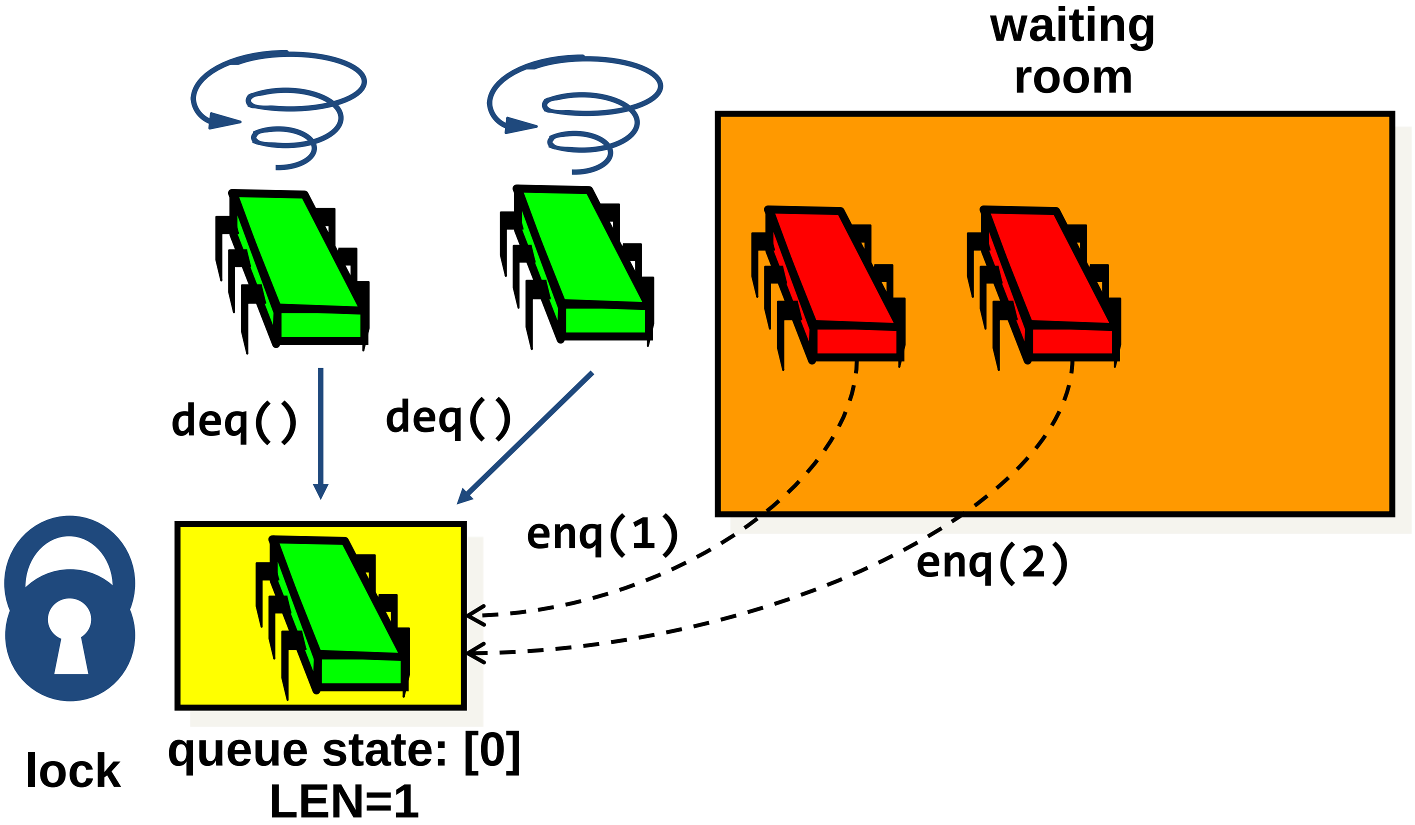
Lost Wakeup in Simplified Queue with `signal()`



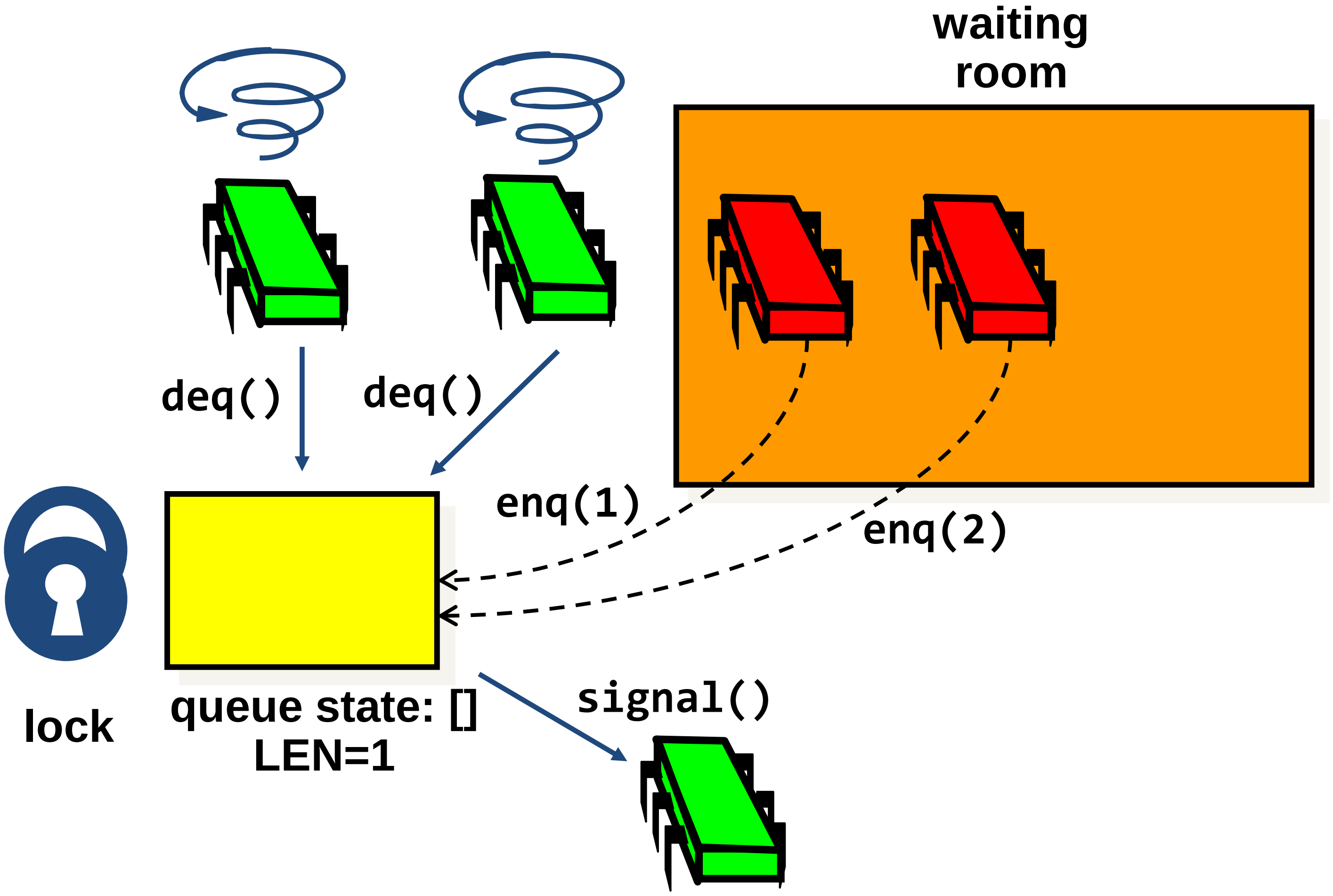
Lost Wakeup in Simplified Queue with `signal()`



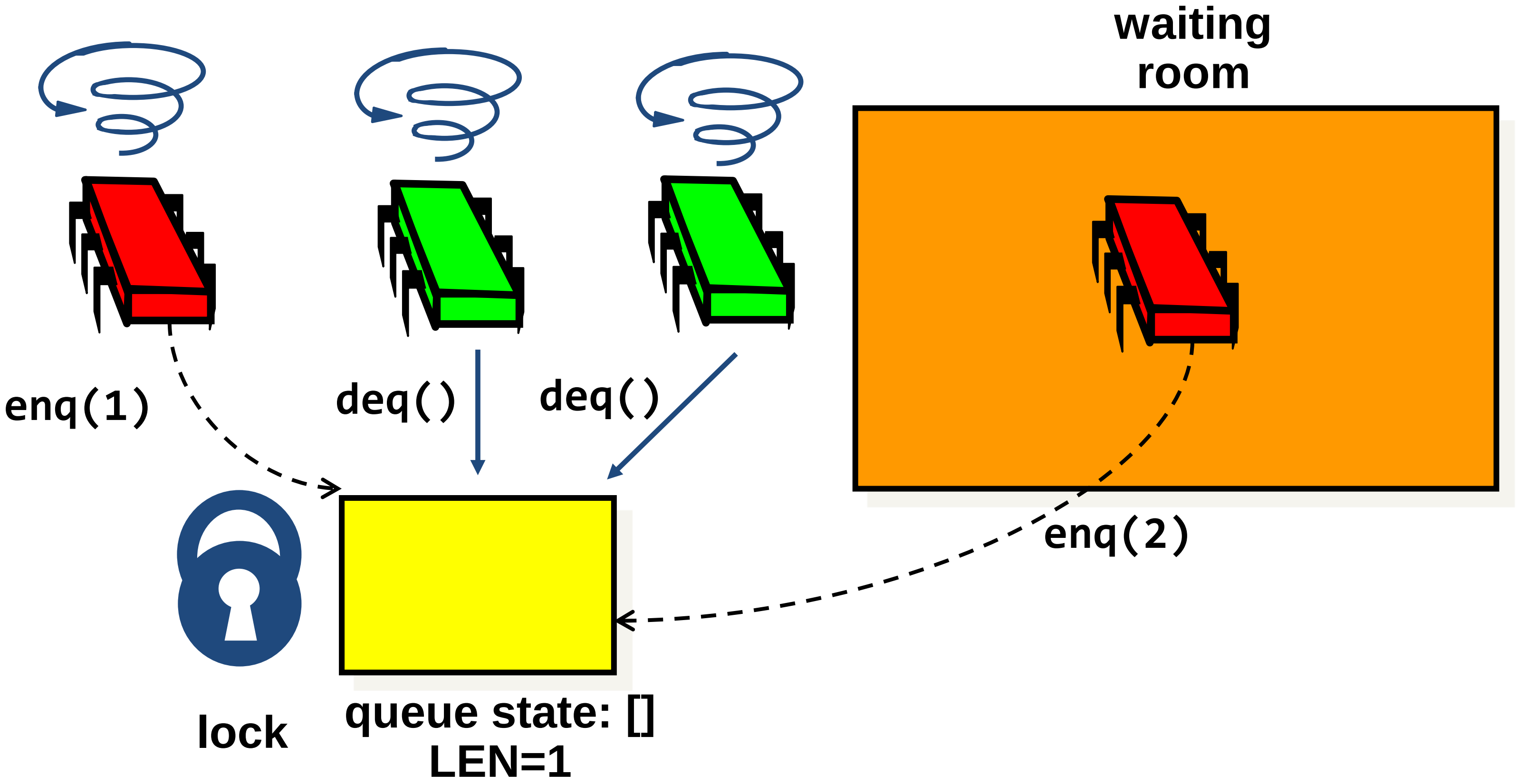
Lost Wakeup in Simplified Queue with `signal()`



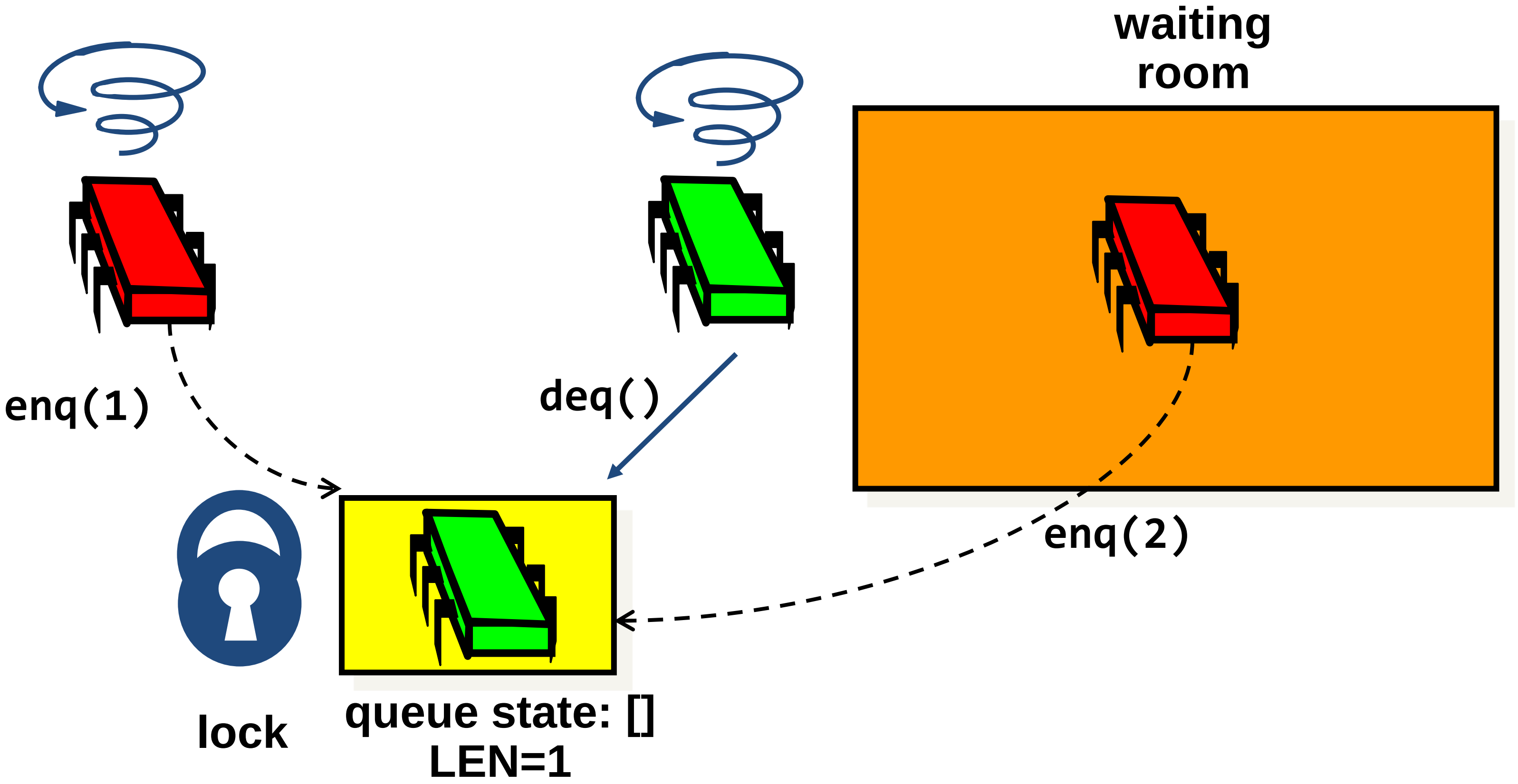
Lost Wakeup in Simplified Queue with `signal()`



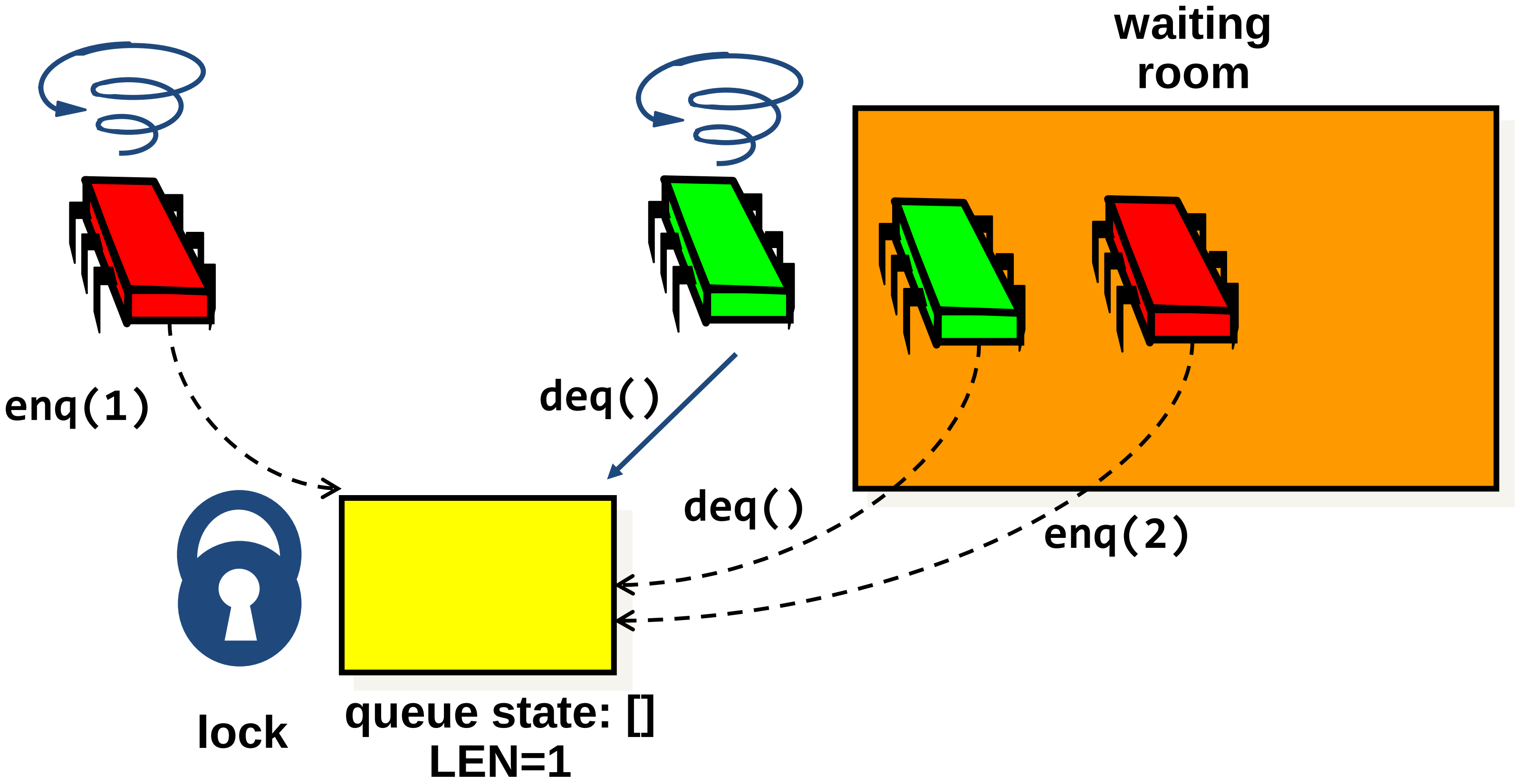
Lost Wakeup in Simplified Queue with `signal()`



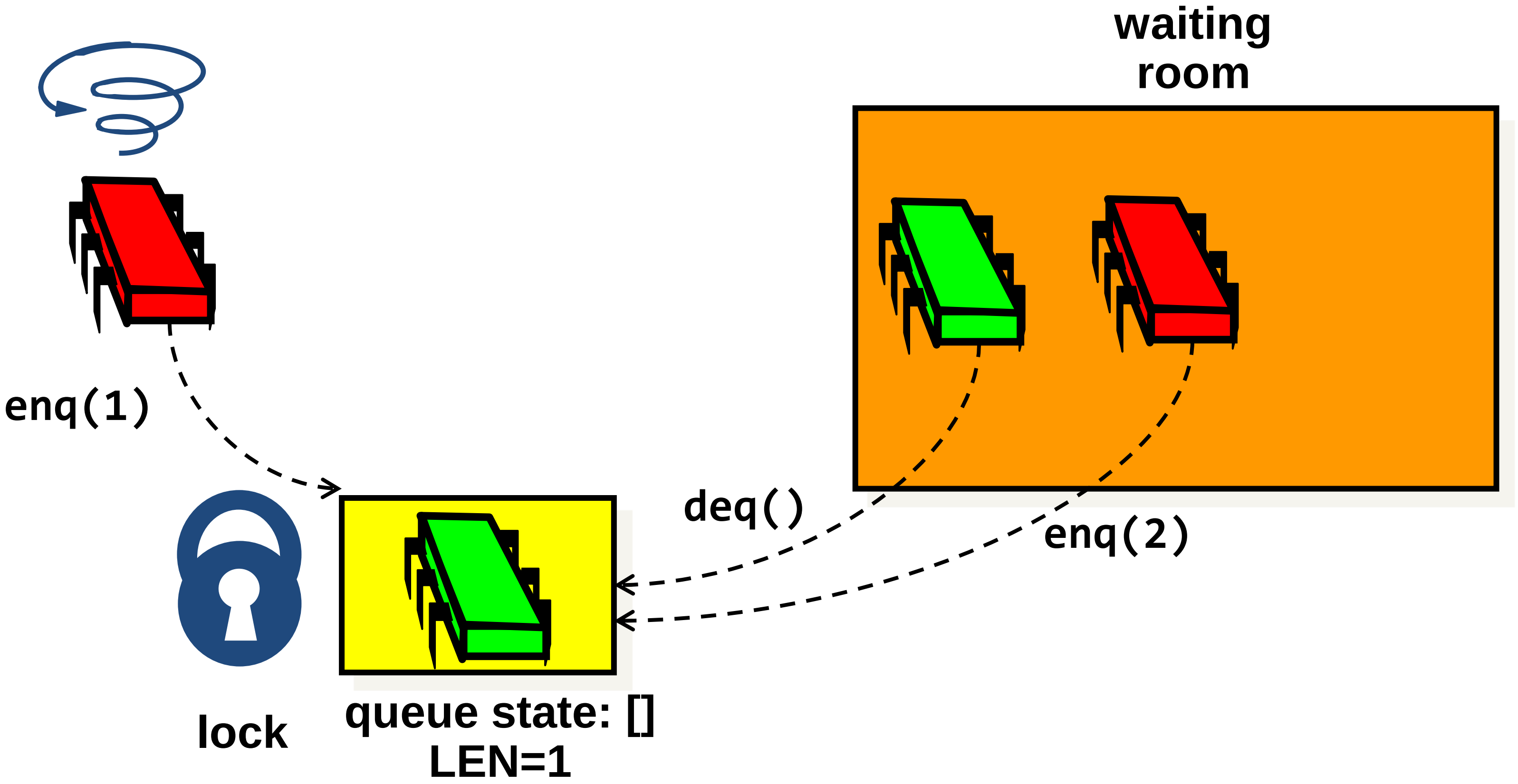
Lost Wakeup in Simplified Queue with `signal()`



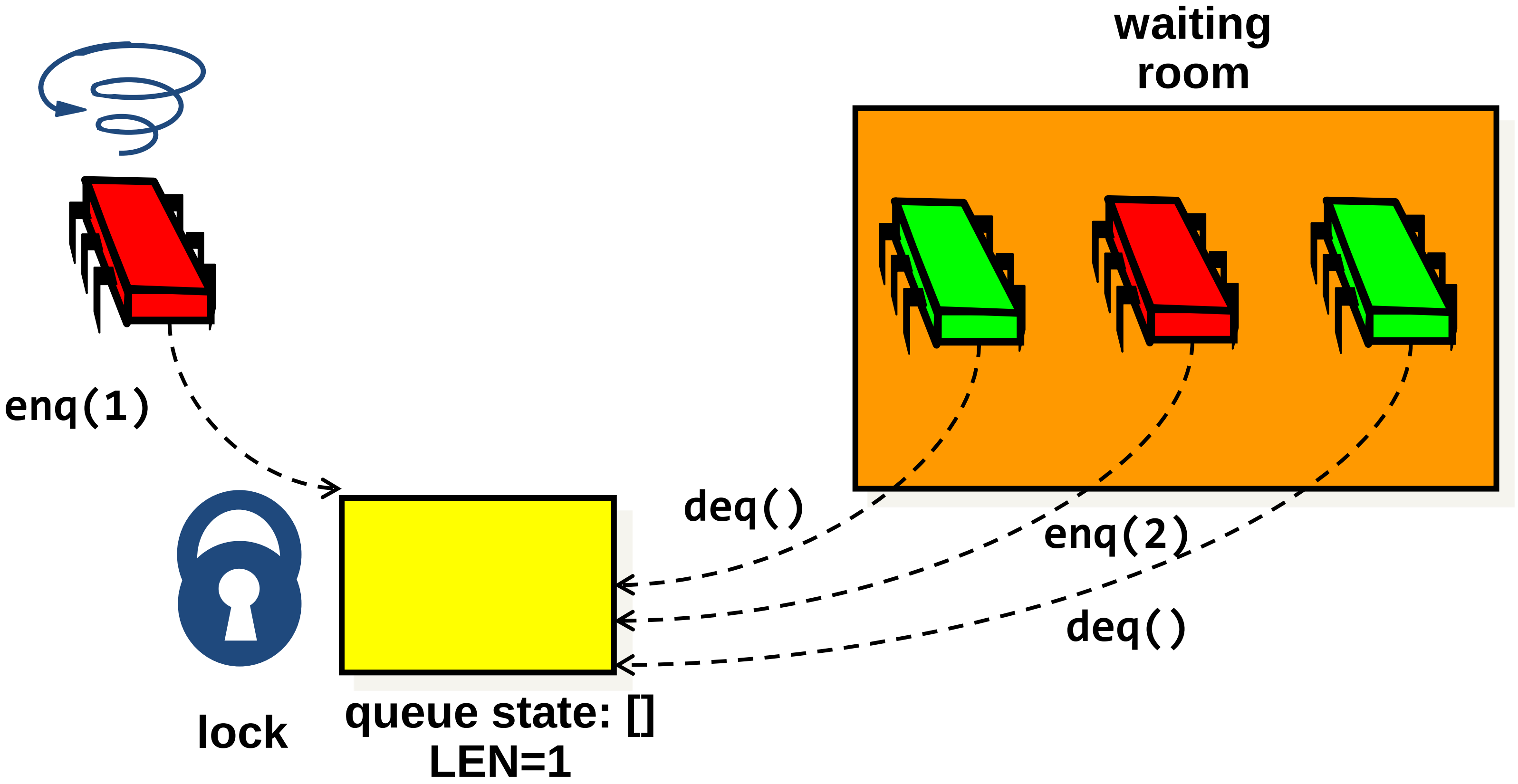
Lost Wakeup in Simplified Queue with `signal()`



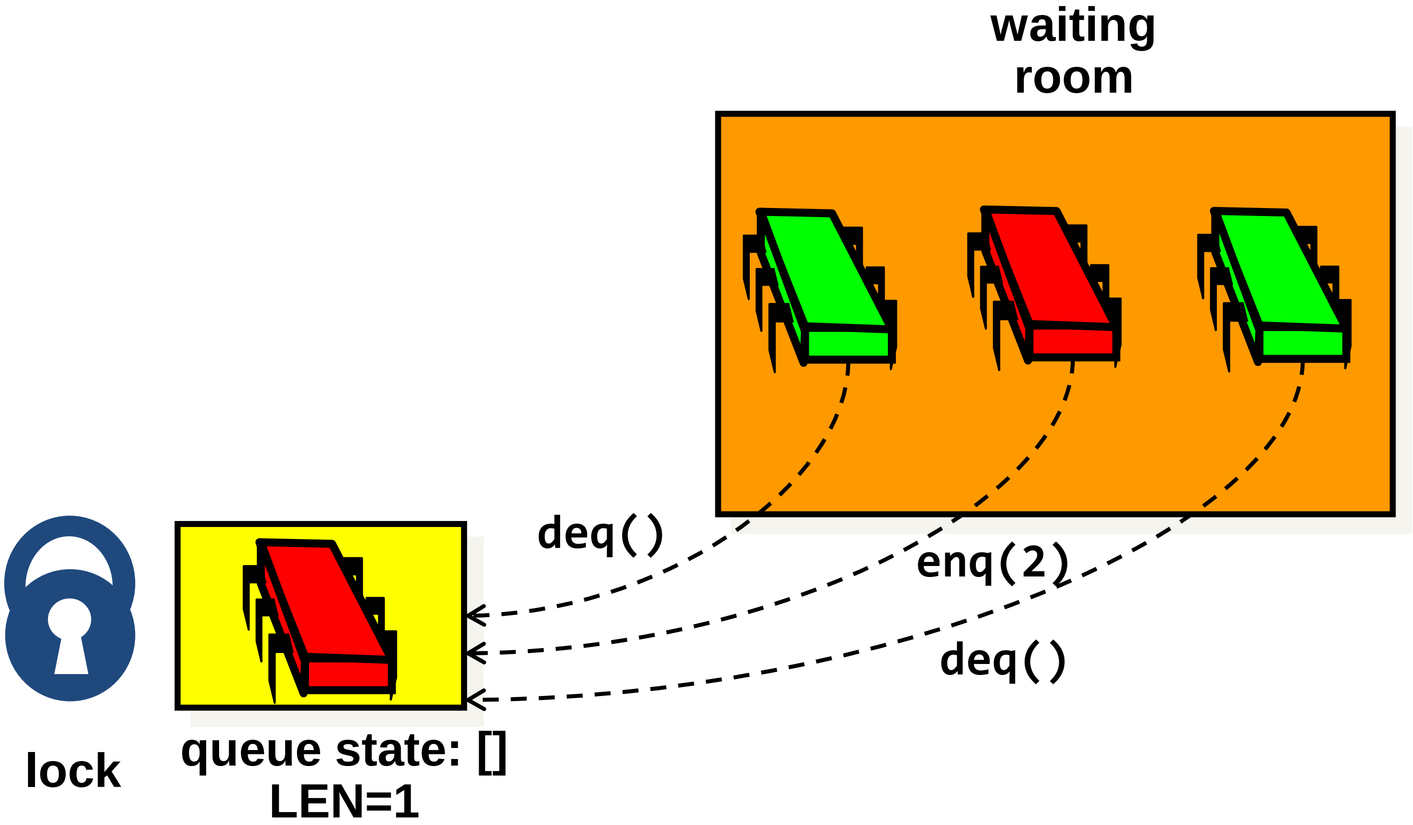
Lost Wakeup in Simplified Queue with `signal()`



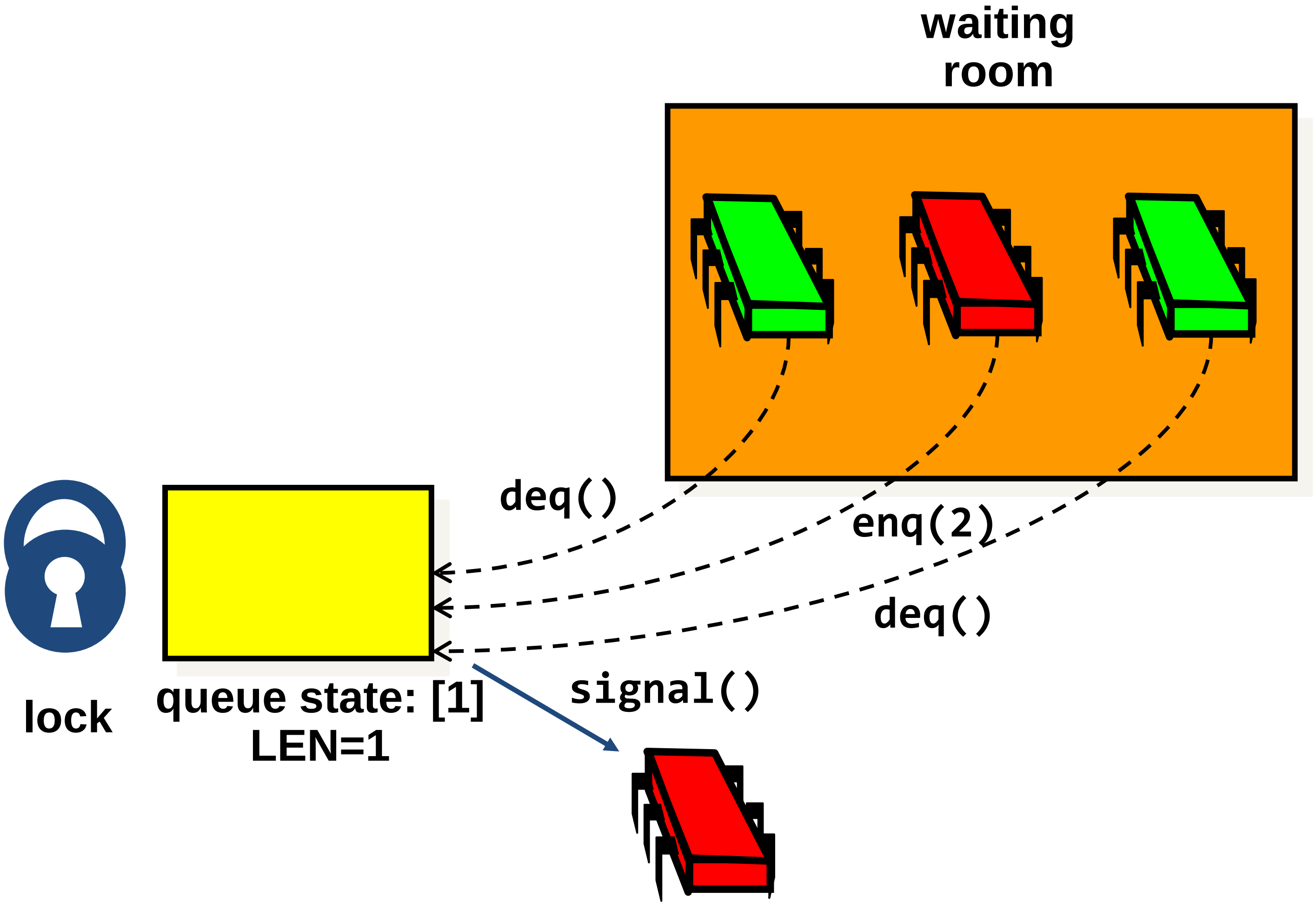
Lost Wakeup in Simplified Queue with `signal()`



Lost Wakeup in Simplified Queue with `signal()`



Lost Wakeup in Simplified Queue with `signal()`



waiting room

deq()

enq(2)

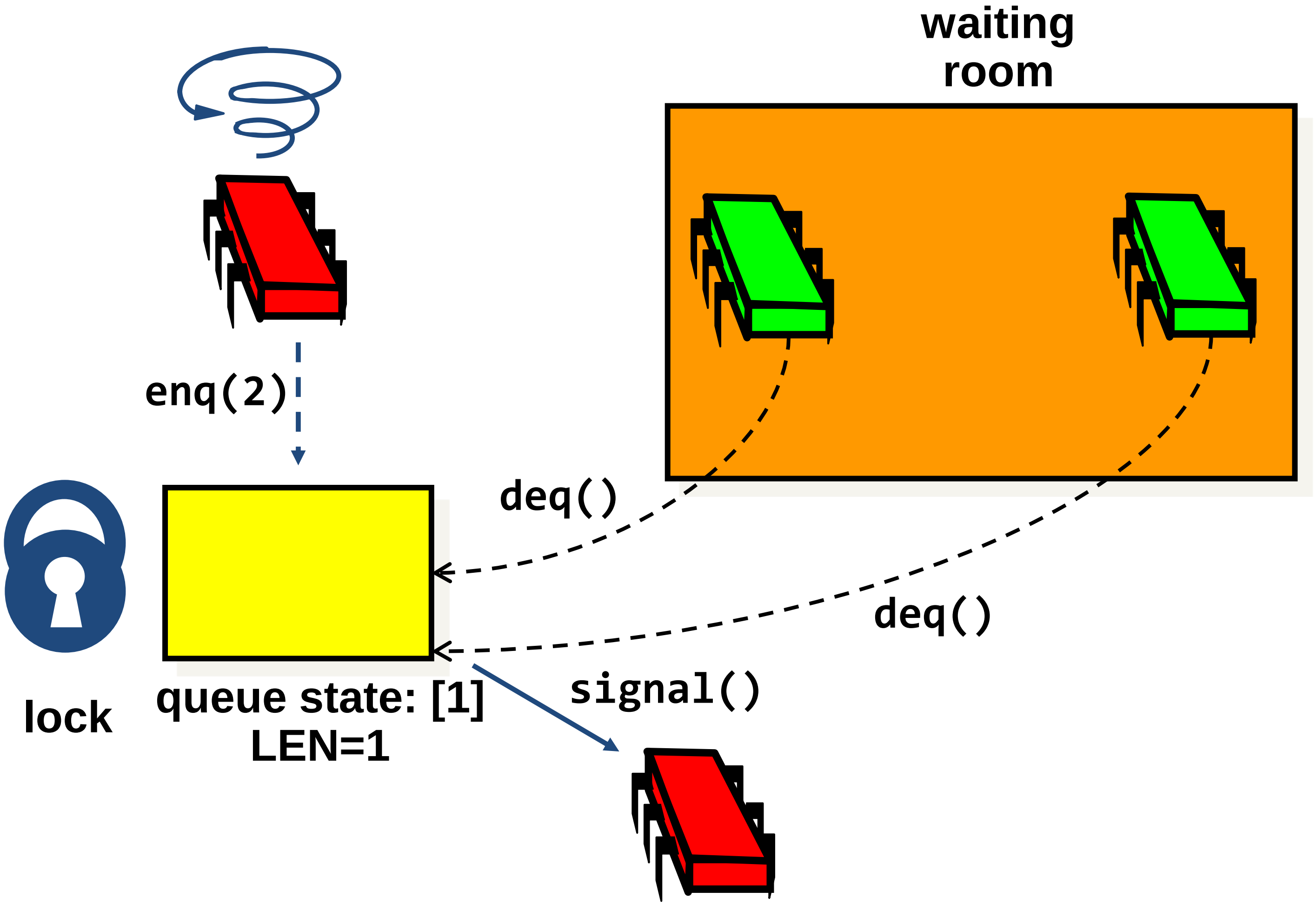
deq()

signal()

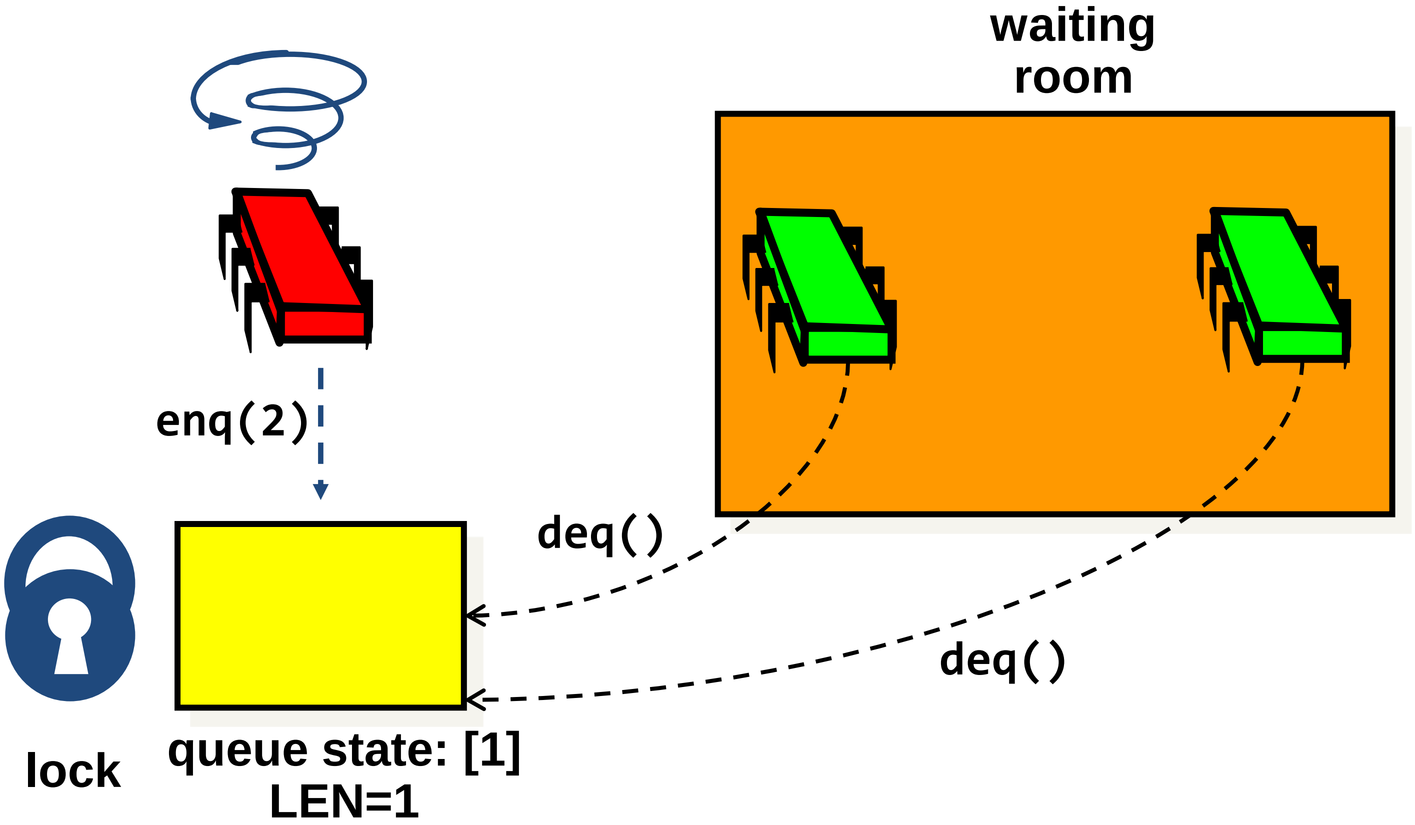
lock

queue state: [1]
LEN=1

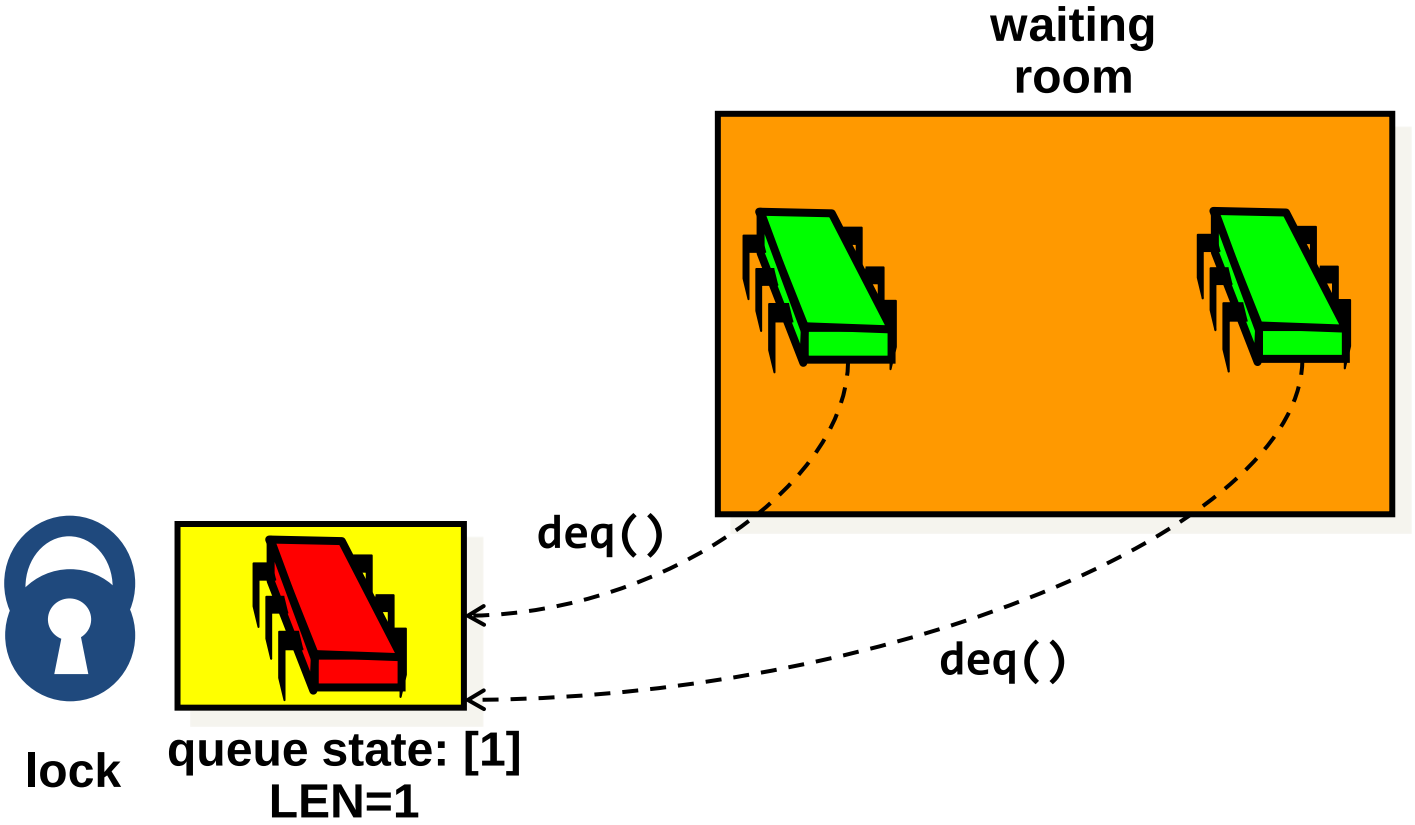
Lost Wakeup in Simplified Queue with `signal()`



Lost Wakeup in Simplified Queue with `signal()`



Lost Wakeup in Simplified Queue with `signal()`



waiting room

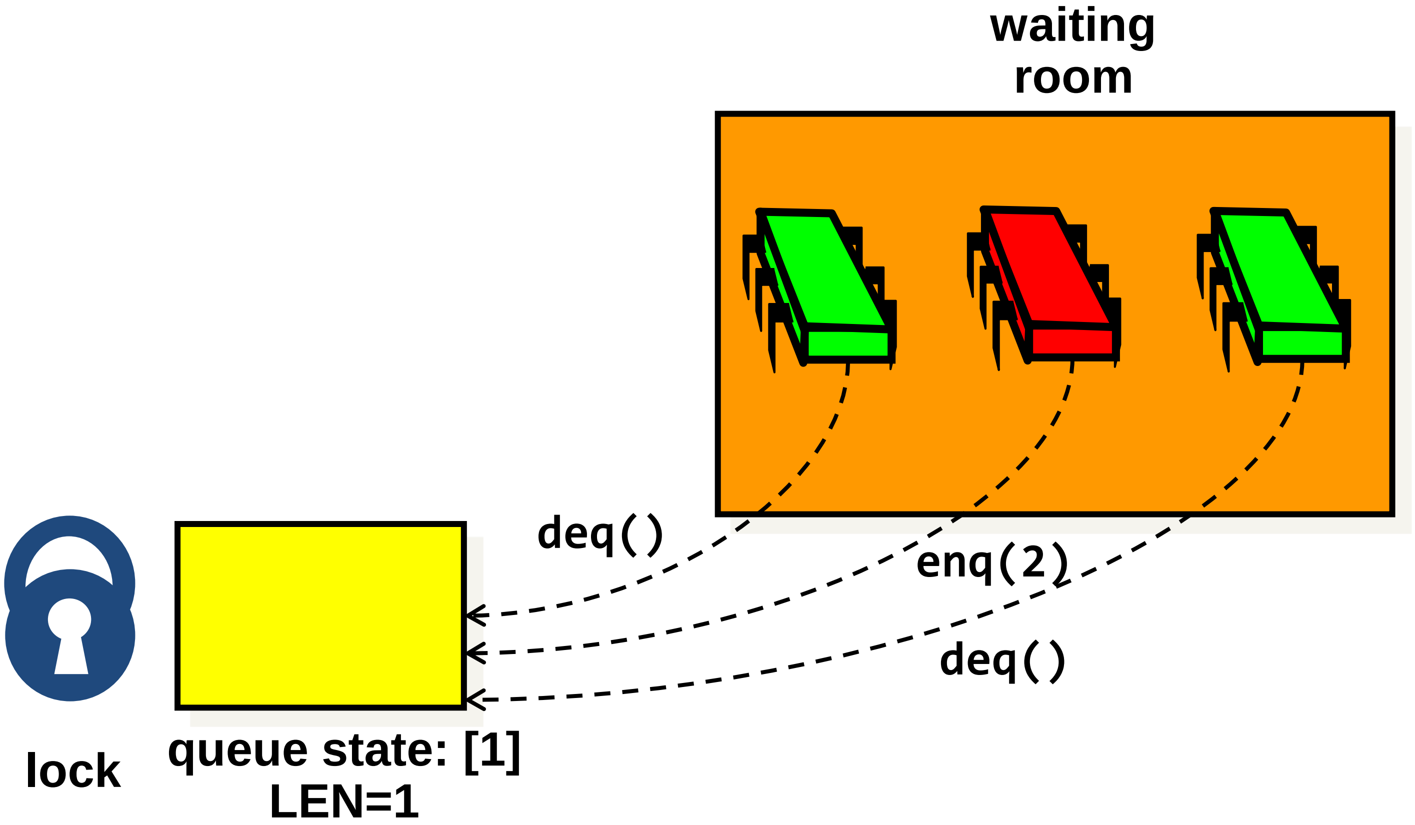
deq()

deq()

lock

queue state: [1]
LEN=1

Lost Wakeup in Simplified Queue with `signal()`



remaining threads are stuck!

Fairness and Starvation

pthread_mutex_lock does not guarantee fairness

Thread 0 Thread 1 Thread 2

```
pthread_mutex_lock(&mu);
                          pthread_mutex_lock(&mu);
                                                    pthread_mutex_lock(&mu);
```

Fairness and Starvation

pthread_mutex_lock does not guarantee fairness

Thread 0

```
pthread_mutex_lock(&mu);

processing...

pthread_mutex_unlock(&mu);
```

Thread 1

```
pthread_mutex_lock(&mu);
```

block and wait Z Z Z

Thread 2

```
pthread_mutex_lock(&mu);
```

block and wait Z Z Z

Fairness and Starvation

pthread_mutex_lock does not guarantee fairness

Thread 0

```
pthread_mutex_lock(&mu);

processing...

pthread_mutex_unlock(&mu);
pthread_mutex_lock(&mu);
```

block and wait

Thread 1

```
pthread_mutex_lock(&mu);
```

block and wait

```
processing...
```

Thread 2

```
pthread_mutex_lock(&mu);
```

block and wait

Fairness and Starvation

pthread_mutex_lock does not guarantee fairness

Thread 0

```
pthread_mutex_lock(&mu);

processing...

pthread_mutex_unlock(&mu);
pthread_mutex_lock(&mu);
```

block and wait

```
processing...
```

Thread 1

```
pthread_mutex_lock(&mu);
```

block and wait

```
processing...

pthread_mutex_unlock(&mu);
```

Thread 2

```
pthread_mutex_lock(&mu);
```

block and wait

Thread 2 is starving!

Ticket Lock

```
typedef struct {
  pthread_mutex_t mu;
  pthread_cond_t cond;
  unsigned long queue_head, queue_tail;
} ticket_lock_t;
```

Ticket lock: lock

```
void lock(ticket_lock_t *tlock) {
  unsigned long my_ticket;
  pthread_mutex_lock(&tlock->mu);
  my_ticket = tlock->queue_tail++;
  while (my_ticket != tlock->queue_head)
    pthread_cond_wait(&tlock->cond, &tlock->mu);
  pthread_mutex_unlock(&tlock->mu);
}
```

Ticket lock: unlock

```
void unlock(ticket_lock_t *tlock) {
  pthread_mutex_lock(&tlock->mu);
  tlock->queue_head++;
  pthread_broadcast(&tlock->cond, &tlock->mu);
  pthread_mutex_unlock(&tlock->mu);
}
```