

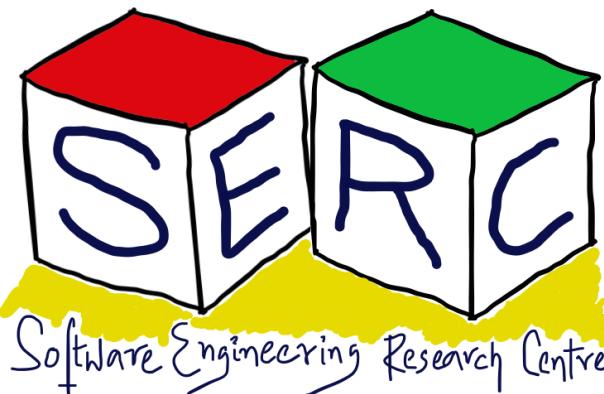
CS3.301 Operating Systems and Networks

Concurrency - Semaphores and Classical Concurrency Problems

Karthik Vaidhyanathan

<https://karthikvaidhyanathan.com>

1



Acknowledgement

The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidyanathan

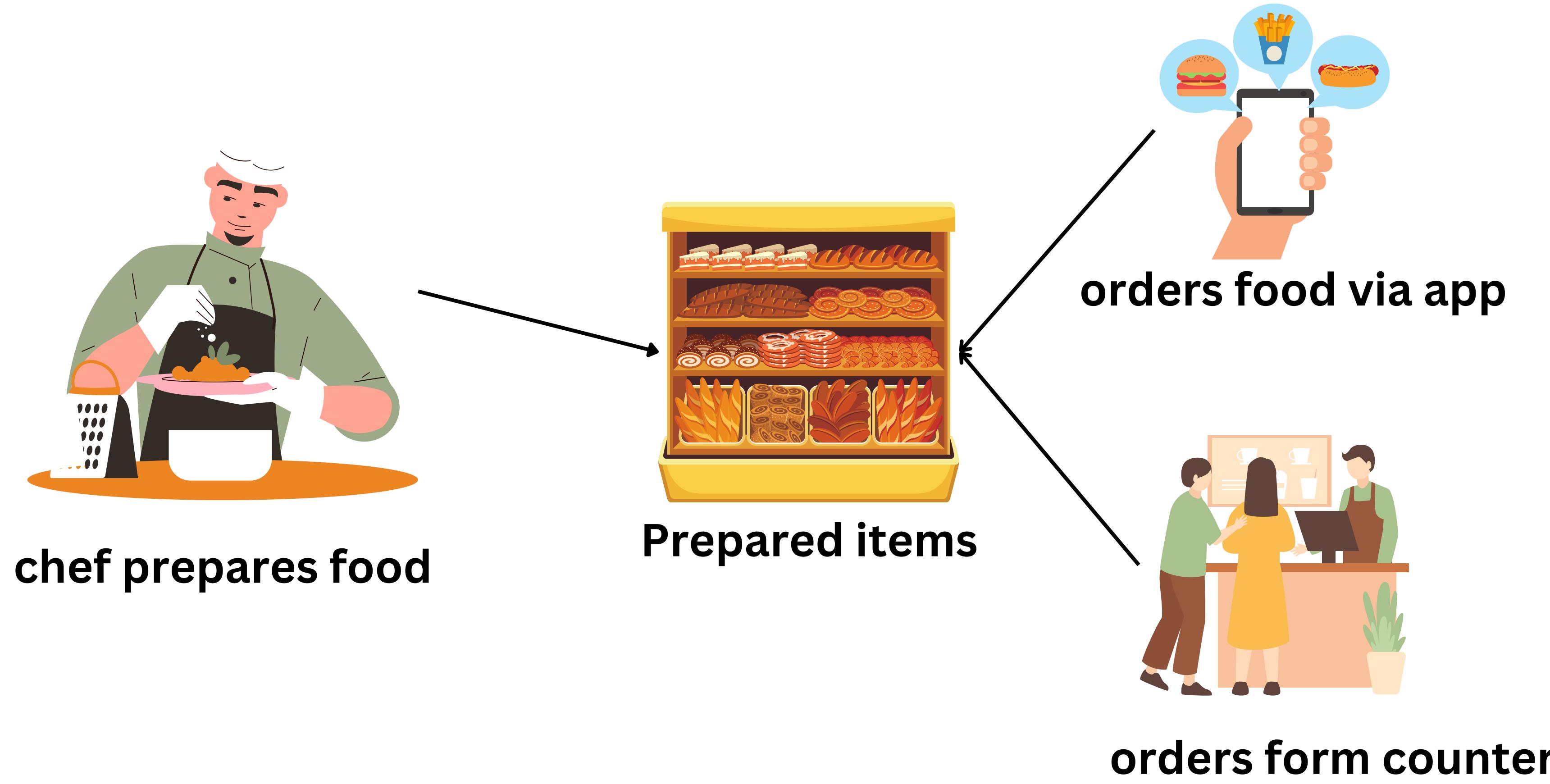
Sources:

- Operating Systems in three easy pieces by Remzi et al.



Producer-Consumer Problem

The Bounded Buffer Problem



One cannot get food items that are not yet ready!



Lets start simple

- Consider buffer can hold only one item, a single integer - **How to solve?**



Producer-Consumer-GetAndPut

```
int buffer = 0
int count = 0

int get()
{
    assert(count==1);
    count = 0;
    return buffer;
}

void put (int value)
{
    assert (count==0);
    buffer = value;
    count = 1;
}
```



Producer-Consumer

```
void *producer (void *arg)
{
    int i;
    int maxLoops = (int) arg;
    for (i=0; i<maxLoops; i++)
    {
        put(i);
    }
}

int *consumer(void *arg)
{
    int value;
    while (1)
    {
        value = get();
        printf("%d\n", value);
    }
}
```

Surround with Locks and Condition Variables

Only one producer and one consumer



Producer

```
cond_t cond;
mutex_t mutex;

void *producer(void *arg)
{
    int i;
    int maxLoops = (int)arg;
    for (i=0; i<maxLoops; i++)
    {
        pthread_mutex_lock(&mutex); //get the lock into CS
        if (count==1) // check if something exist
        {
            pthread_cond_wait(&cond,&mutex);
        }
        put (i);
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&unlock);
    }
}
```



Consumer

```
cond_t cond;
mutex_t mutex;

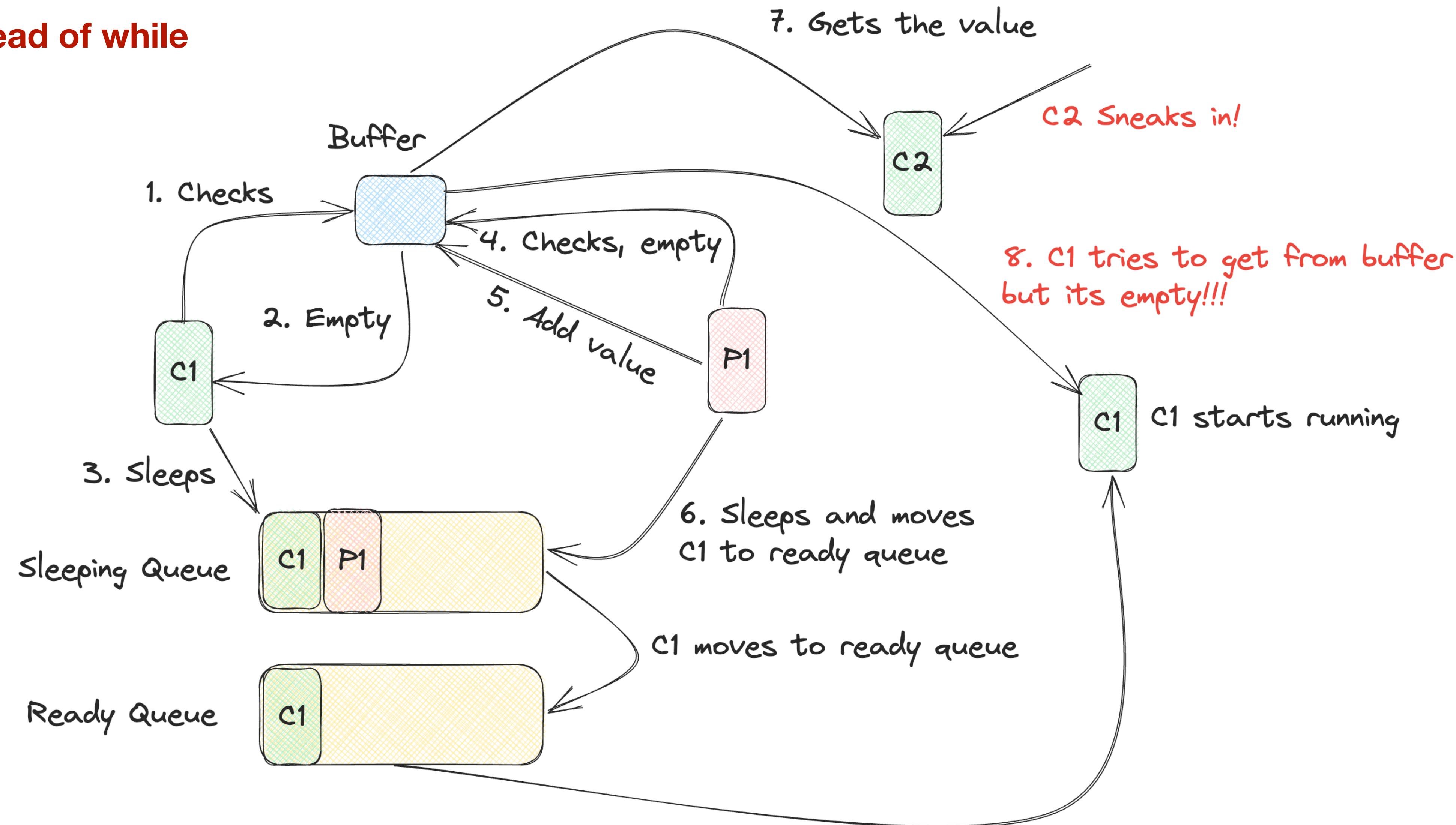
void *consumer(void *arg)
{
    int i;
    int maxLoops = (int)arg;
    for (i=0; i<maxLoops; i++)
    {
        pthread_mutex_lock(&mutex); //get the lock into CS
        if (count==0) // check if there is nothing
        {
            pthread_cond_wait(&cond,&mutex);
        }
        int temp = get();
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&unlock);
        printf ("%d\n", temp);
    }
}
```



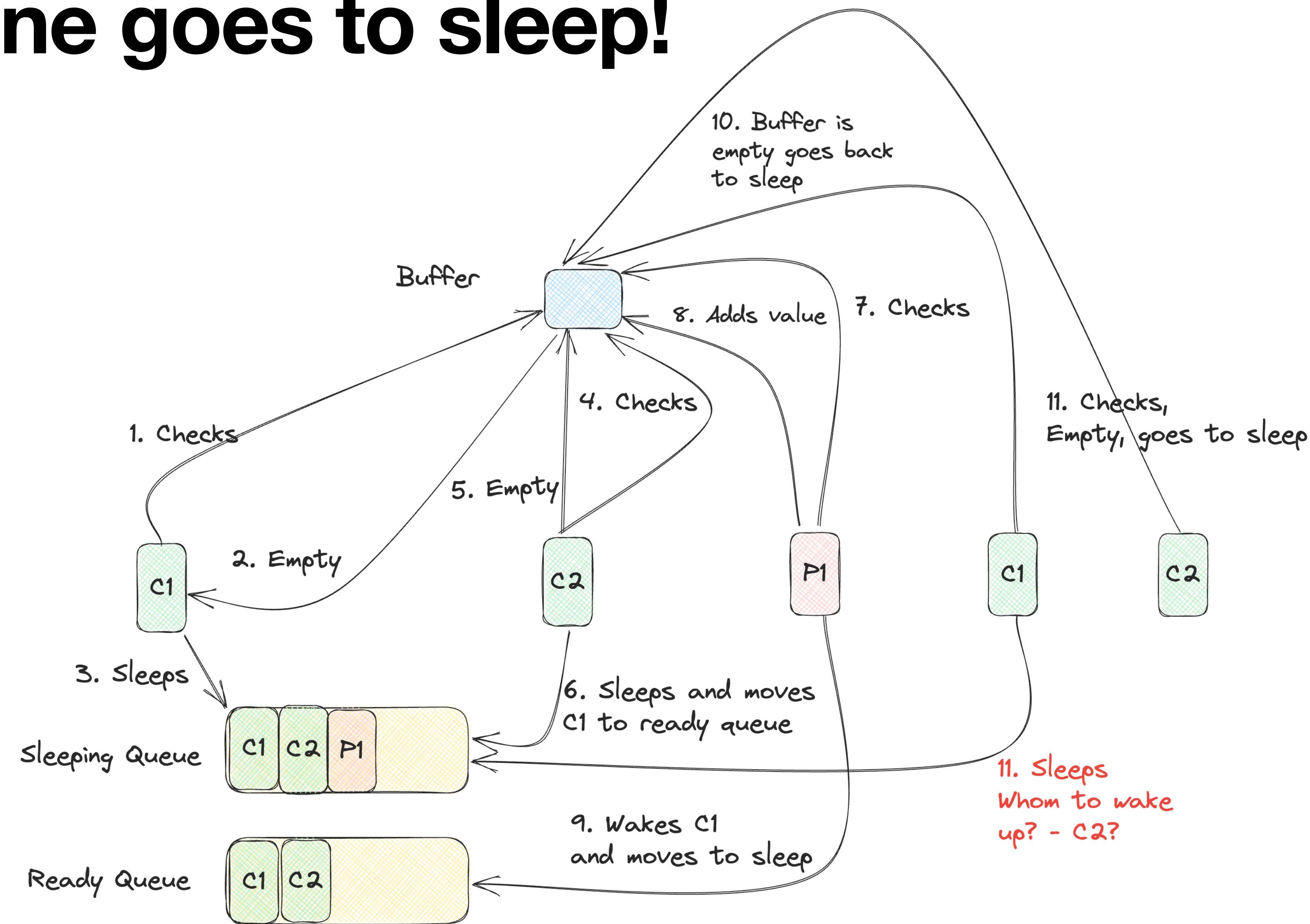
What if there are more producers and consumers

Two Key Challenges

If instead of while



Everyone goes to sleep!



Use Two Condition Variables

```
cond_t fill;  
cond_t empty;
```

- **Producer waits on empty condition** => waits for consumer to empty the buffer
 - Signals on fill => signals consumer that buffer is filled!
 - **Consumer waits on fill condition** => waits for producer to fill buffer
 - Signals on empty => signals producer that buffer is empty!
 - Producer cannot awaken producer and consumer cannot awaken consumer
- What about more than one in the buffer? - **Buffer can be an array of integers**



Producer Consumer Problem Solution



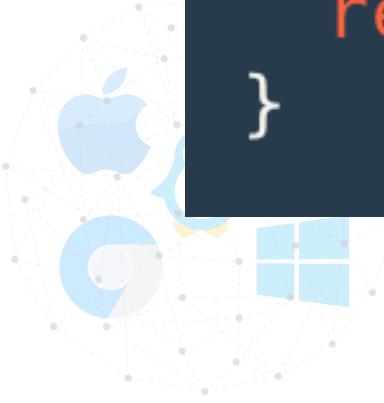
Get and Put for large sized buffer

```
int buffer[MAX];
int fill = 0;
int use = 0;
int count = 0;

void put (int value)
{
    buffer[fill] = value;
    fill = (fill + 1)%MAX;
    count++;
}

int get( )
{
    int tmp = buffer[use];
    use = (use + 1)%MAX;
    count--;
    return tmp;
}
```

- Buffer now can hold an array of integers
- Fill and use are used to manage indexing
- Producers can keep pushing data to the buffer
- Consumers can keep reading data from the buffer
- How to implement producer and consumer?



Producer Consumer Problem Solution



Producer with two condition variables

```
cond_t empty; //two condition variables
cond_t fill;
mutex_t mutex;

void *producer(void *arg)
{
    int i;
    int maxLoops = (int)arg;
    for (i=0; i<maxLoops; i++)
    {
        pthread_mutex_lock(&mutex); //get the lock into CS
        while (count==MAX) // check if its already full
        {
            pthread_cond_wait(&empty,&mutex);
        }
        put (i);
        pthread_cond_signal(&fill);
        pthread_mutex_unlock(&unlock);
    }
}
```



Consumer with two condition variables

```
cond_t empty; //two condition variables
cond_t fill;
mutex_t mutex;

void *consumer(void *arg)
{
    int i;
    int maxLoops = (int)arg;
    for (i=0; i<maxLoops; i++)
    {
        pthread_mutex_lock(&mutex); //get the lock into CS
        while (count==0) // check if there is nothing
        {
            pthread_cond_wait(&fill,&mutex);
        }
        int temp = get();
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&unlock);
        printf ("%d\n", temp);
    }
}
```

Is there a better way to do this?

- **Locks:** Provide atomic access to critical section
- **Condition Variables:** Allows signalling between threads or passing some information on condition between threads
- What if both can be done using a single mechanism?
 - Edsger W. Dijkstra did that through the concept of **Semaphores**



Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better

Semaphore: One structure which can act as both condition Variable and lock

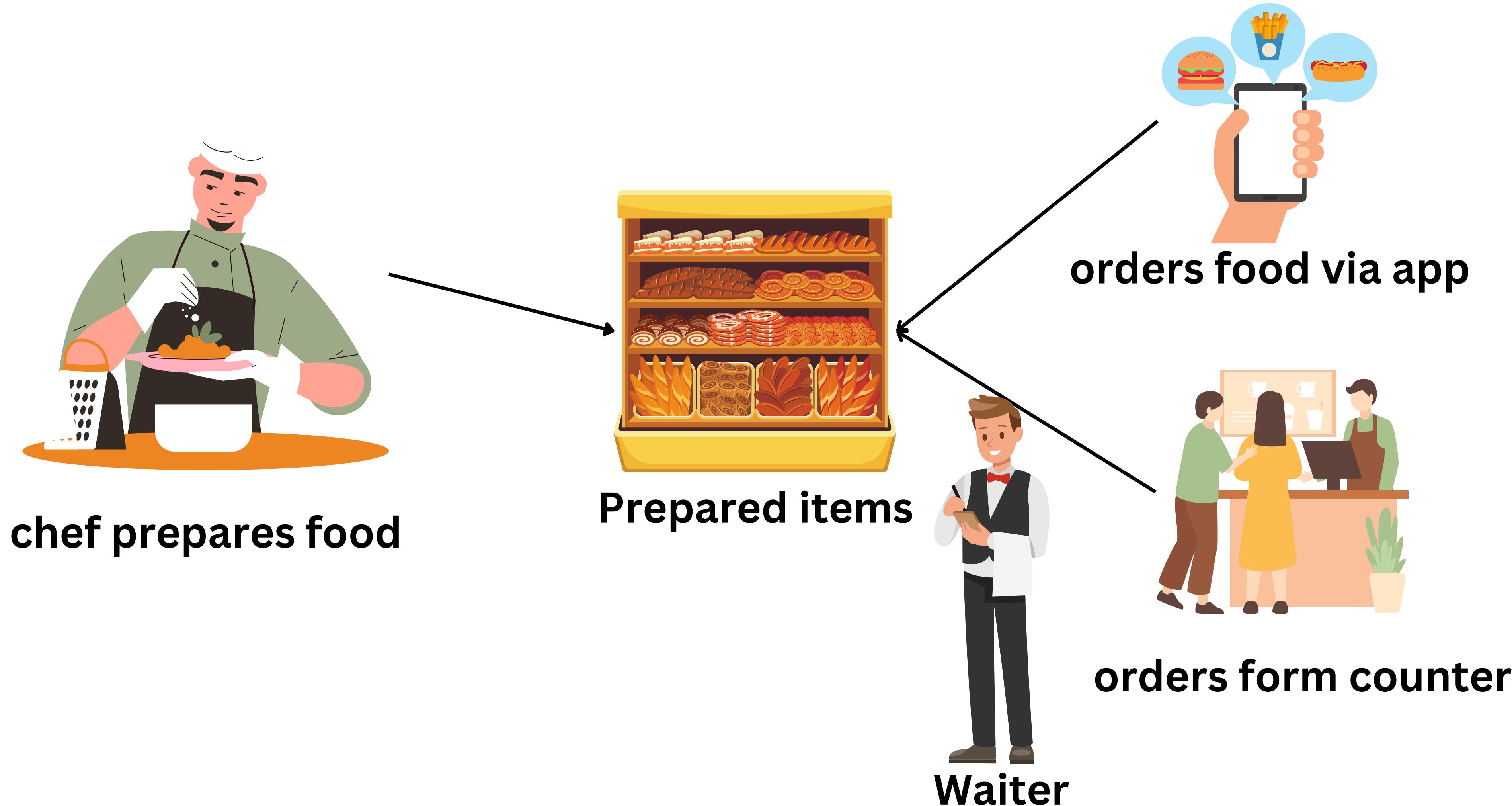


Edsger W. Dijkstra



An Analogy

May be a waiter can help better?



Semaphore

- An object with an integer value that we can manipulate with two routines: wait and post. As per original naming:
 - **P(): proberen** - Decrease the value, Check
 - **V(): Verhogen** - Increase the value
- In POSIX, there are two routines:
 - **sem_wait()**: decrease the semaphore, if negative block
 - **sem_post()**: increase the semaphore value



Semaphore

Semaphore

```
#include <semaphore.h>
sem_t s;
sem_init (&s, 0, 1);
```

Semaphore
shared between
threads in same process

Initial
value of semaphore

Semaphore - Wait

```
int sem_wait(sem_t *s)
{
    // decrement s by 1
    // wait if value of s is negative
}
```

Semaphore - Post

```
int sem_post(sem_t *s)
{
    // increment value of s by 1
    // if there are threads waiting,
    // wake one of them
}
```



Semaphore

- **sem_wait():**
 - Either, it will either return right away after decrementing the value
 - Or, it will cause the caller to suspend execution waiting for a subsequent post
 - When there are multiple threads, they can call wait and get queued
- **sem_post():**
 - Simply increments the value
 - If the thread is waiting, wakes one of them up
- Value of semaphore, when negative equals to number of waiting threads



Semaphores as Locks

Binary Semaphores - How to use Semaphores as locks?

- Always think about what should be the initial value of semaphore, **here it is 1**
- Assume there are two threads
 - Thread 0 calls `sem_wait()`
 - Decrements the value to 0
 - Thread 0 can enter CS
 - At this time if Thread 1 wants to enter CS -> calls `sem_wait()` -> -1, sleeps
 - Once thread 0 is done, calls `sem_post`
 - Increments value by 1, wakes thread 1



Semaphore - Locks

```
sem_t sem_var;
sem_init(&sem_var, 0, 1);
sem_wait (&sem_var);
//Critical section code here
sem_post (&sem_var);
```



Semaphores can also function as condition Variables



Semaphore - Condition variables

```
sem_t sem_var;

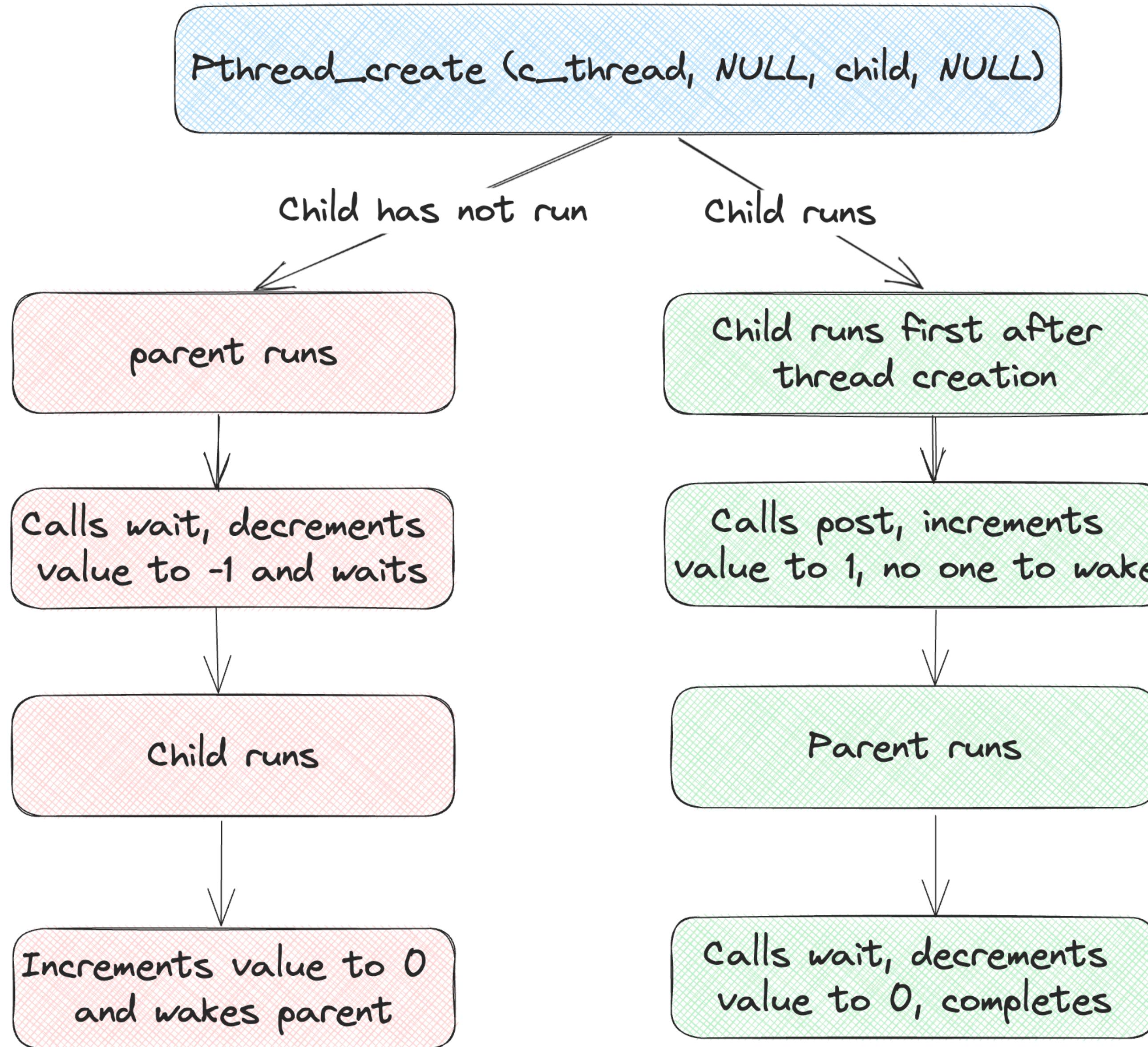
void *child(void *arg)
{
    printf("child\n");
    sem_post(&sem_var);
    return NULL;
}

int main (int argc, char *argv[])
{
    sem_init(&s, 0, 1);
    pthread_t c_thread;
    pthread_create(c_thread, NULL, child, NULL);
    sem_wait(&sem_var);
    printf("parent\n");
}
```

- There are two main possible execution
- Parent runs, create the child and the child has not run yet
- Parent runs, creates the child and the child immediately runs
- How does the semaphore help with both the above condition?
 - **What should be value of sem_var?**



Semaphores as condition variables



Producer Consumer Problem Using Semaphores

- Let us start with 2 semaphores: empty and wait, Buffer with MAX = 1

● ● ● Get and Put for large sized buffer

```
int buffer[MAX];
int fill = 0;
int use = 0;
int count = 0;

void put (int value)
{
    buffer[fill] = value;
    fill = (fill + 1)%MAX;
    count++;
}

int get()
{
    int tmp = buffer[use];
    use = (use + 1)%MAX;
    count--;
    return tmp;
}
```



● ● ● Producer-Consumer with buffer

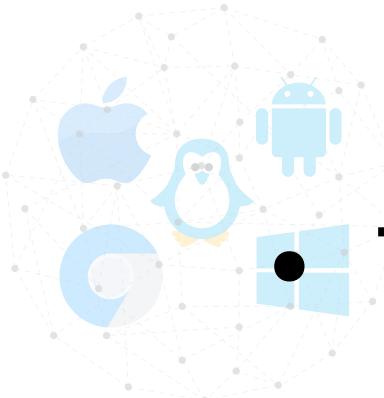
```
sem_t empty;
sem_t full;

void *producer(void *arg)
{
    int i;
    int maxLoops = (int)arg;
    for (i=0;i<maxLoops;i++)
    {
        sem_wait(&empty);
        put (i);
        sem_post(&full);
    }
}

void *consumer(void *arg)
{
    int i;
    int maxLoops = (int)arg;
    for (i=0;i<maxLoops;i++)
    {
        sem_wait(&full);
        int tmp = get();
        sem_post(&empty);
        printf("%d\n", tmp);
    }
}
```

Is our solution fine?

- Consider two threads (producer and consumer) on single thread
- Assume consume runs first `sem_wait(&full)`
 - Decrements **full (0) to -1** and waits for the thread to call post
 - Moves to a blocked state
- Producer runs, calls `sem_wait (&empty)`
 - **Empty (1) is decremented to 0** and proceeds to add value
 - Once done, calls post and moves consumer to ready
 - If producer runs again, it will keep looping, consumer when runs, can get the lock
- This can work for multiple producers and consumers but what **if MAX>1**



What about buffer with $\text{MAX} > 1$

- Assume two producers, **P1 and P2**
- P1 runs first, fills the buffer entry, before updated, interrupt happens
- P2 starts to run and overwrites the value written by P1
- The reason:
 - Two producers **calling put() at the same time!!**
 - **Race condition** is triggered!
 - Remember we have not locked get and put here. What can be done?



Add mutex to solve Producer-Consumer

Producer

```
sem_wait (&mutex);
sem_wait (&empty);
put(i);
sem_post (&full);
sem_post (&mutex);
```

Consumer

```
sem_wait (&mutex);
sem_wait (&full);
get();
sem_post (&empty);
sem_post (&mutex);
```

- Is there any issue with above code?
- C1 runs first gets mutex but waits on empty, P1 runs but waits for mutex - **Deadlock!!**



Deadlocks?

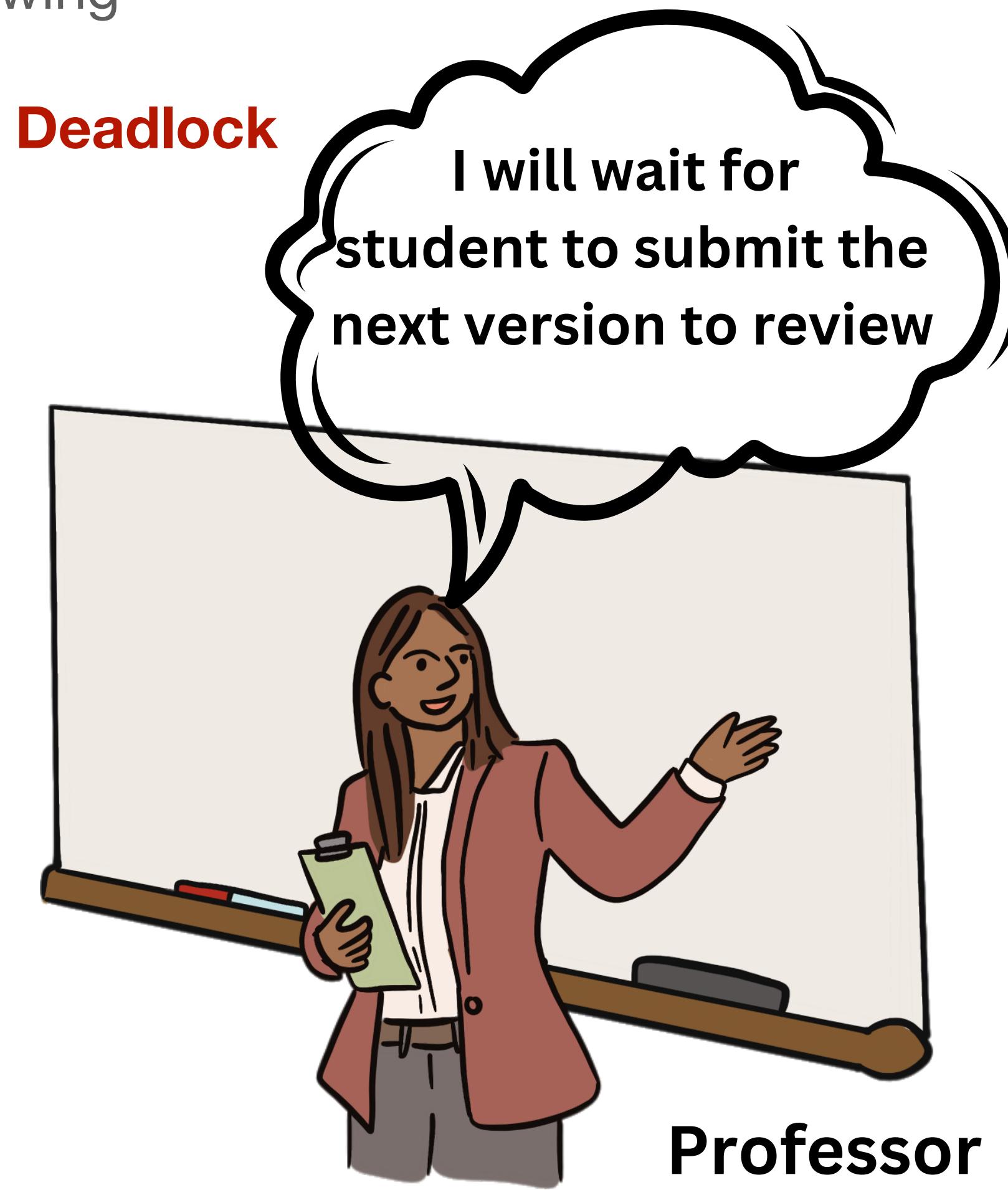
Student has submitted a draft and
Waits for review



Student

Professor is expecting student
To submit better version to start
Reviewing

Both wait - Deadlock



Professor



Producer Consumer Problem Using Semaphores

The Solution

Producer

```
sem_wait (&empty);
sem_wait (&mutex);
put(i);
sem_post (&mutex);
sem_post (&full);
```

Consumer

```
sem_wait (&full);
sem_wait (&mutex);
get();
sem_post (&mutex);
sem_post (&empty);
```

- Add mutex lock around put and get
- Let producer and consumer get the signal and then lock when entering CS



An Analogy

One Person writing



Many people reading at the same time



Online word processors

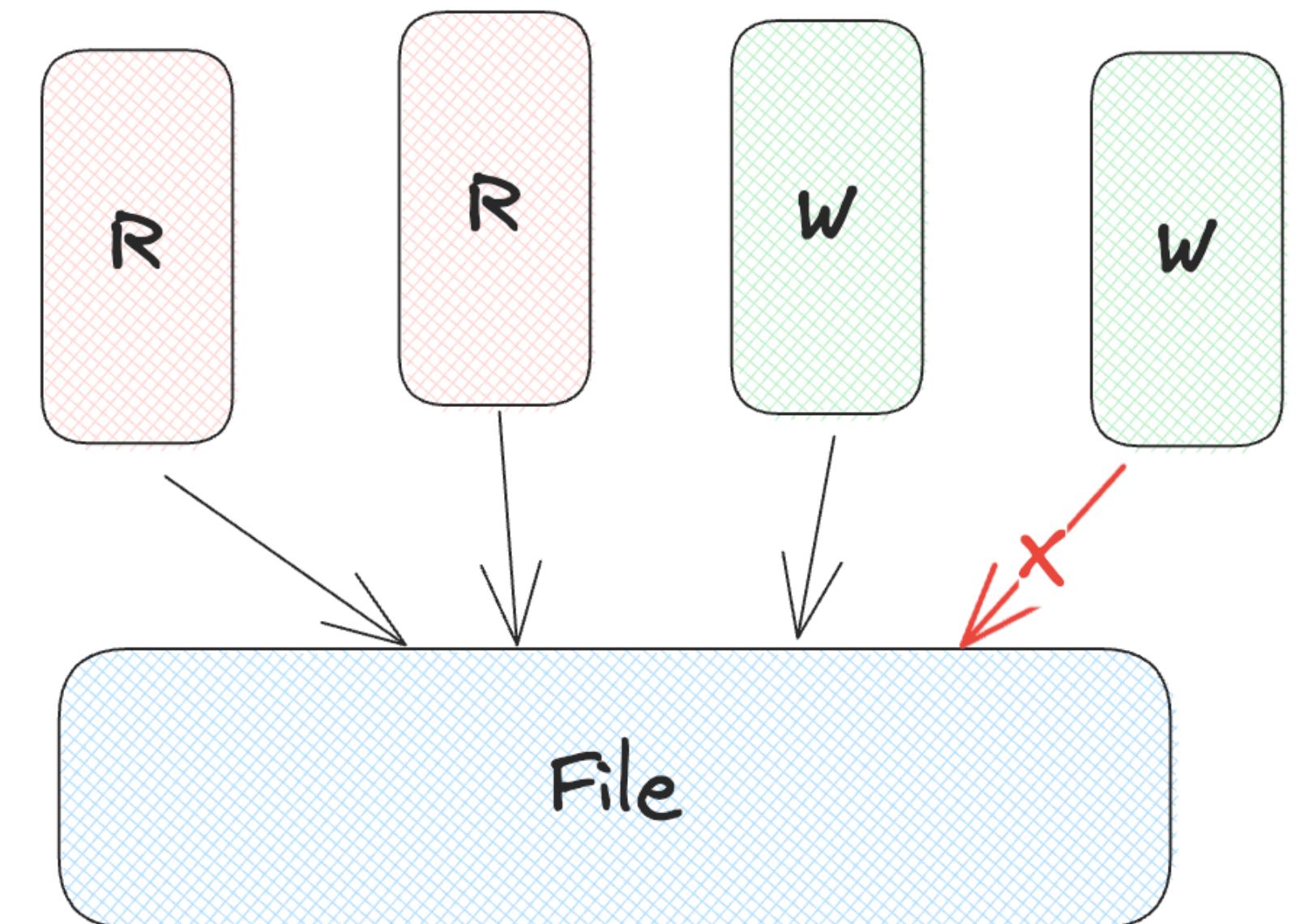


Databases



Readers/Writers Problem

- **Reader:** Process or thread that reads from memory
- **Writer:** Process or thread that writes on the memory
- Two readers can work on the same file at the same time
- Multiple writers cannot work on the same file at the same time



Readers/Writers Problem

Intuition

- **Only one writer can write at any point of time!**
- Reader thread can come in:
 - More readers come in, they can be allowed access
 - The moment writers come, it can be blocked
 - Once readers are done with reading, writers can start writing
- Can you think about writing a solution to this?
 - Do you foresee any challenges here?



Readers/Writers Problem



Readers/Writers Problem Solution

```
typedef struct _rwlock_t
{
    int readers;
    sem_t lock;
    sem_t writelock;
}rwlock_t;

void rwlock_init(rwlock_t *rw)
{
    rw -> readers = 0;
    sem_init(&rw->lock, 0, 1);
    sem_init(&rw->writelock, 0, 1);
}
```



Readers/writers Problem Solution



Readers/Writers Problem Solution

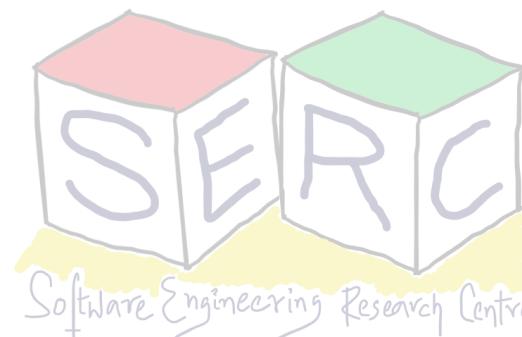
```
void acquire_readlock(rwlock_t *rw)
{
    sem_wait(&rw->lock);
    rw->readers++;
    if(rw->readers == 1)
        //disable writers to enter
        sem_wait(&rw->writelock);
    sem_post(&rw->lock);
}

void release_readlock(rwlock_t *rw)
{
    sem_wait(&rw->lock);
    rw->readers--;
    if(rw->readers == 0)
        //free the write lock
        sem_post(&rw->writelock);
    sem_post(&rw->lock);
}

void acquire_writelock(rwlock_t *rw)
{
    sem_wait(&rw->writelock)
}

void release_writelock(rwlock_t *rw)
{
    sem_post(&rw->writelock)
}
```

Writers Starve!!!



Readers/Writers Problem Solution

Add a lock that can act as priority common to both



Readers Writers - Better solution

```
sem_t serviceQueue;  
  
//writer code code  
  
sem_wait(&serviceQueue);  
sem_wait(&rw->writelock)  
  
.....  
  
sem_post(&rw-> writelock)  
sem_post (&serviceQueue);
```

Writer



Readers Writers - Better solution

```
sem_t serviceQueue;  
  
//reader code  
  
sem_wait(&serviceQueue);  
sem_wait(&rw->lock)  
  
.....  
  
sem_post(&rw-> lock)  
sem_post (&serviceQueue);
```

Reader





Thank you

Course site: karthikv1392.github.io/cs3301_osn

Email: karthik.vaidhyanathan@iiit.ac.in

Twitter: [@karthyishere](https://twitter.com/karthyishere)

