Synchronization

15-213: Introduction to Computer Systems
Recitation 14:December 24, 2014

Vinay Venkatesh
Section G

Topics

- Proxy lab
- Concurrency

Telnet/Curl Demo

Telnet

- Interactive remote shell – like ssh without security
- Must build HTTP request manually
  - This can be useful if you want to test response to malformed headers.

```
hartaj@ubuntu:~$ telnet www.cmu.edu 80
Trying 128.2.42.52...
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu.
Escape character is '^]'.
GET http://www.cmu.edu/ HTTP/1.0

HTTP/1.1 301 Moved Permanently
Date: Sun, 13 Apr 2014 22:21:11 GMT
Server: Apache/1.3.42 (Unix) mod_gzip/1.3.26.1a mod_pubcookie/3.3.4a mod_ssl/2.8.
31 OpenSSL/0.9.8e-fips-rhel5
Location: http://www.cmu.edu/index.shtml
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cm
u.edu</A> Port 80</ADDRESS>
</BODY></HTML>
```

Telnet/cURL Demo

cURL

- “URL transfer library” with a command line program
- Builds valid HTTP requests for you!

```
hartaj@ubuntu:~$ curl http://www.cmu.edu/
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cm
u.edu</A> Port 80</ADDRESS>
</BODY></HTML>
```

- Can also be used to generate HTTP proxy requests:

```
hartaj@ubuntu:~$ curl --proxy bambooshark.ics.cs.cmu.edu:47910 http://www.cmu.edu
/
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>301 Moved Permanently</TITLE>
</HEAD><BODY>
<H1>Moved Permanently</H1>
The document has moved <A HREF="http://www.cmu.edu/index.shtml">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.42 Server at <A HREF="mailto:webmaster@andrew.cmu.edu">www.cm
u.edu</A> Port 80</ADDRESS>
</BODY></HTML>
```

How the Web Really Works

- **Excerpt from www.cmu.edu/index.html:**

```
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  ...
  <link href="homecss/cmu.css" rel="stylesheet" type="text/css"/>
  <link href="homecss/cmu-new.css" rel="stylesheet" type="text/css"/>
  <link href="homecss/cmu-new-print.css" media="print" rel="stylesheet" type="text/css"/>
  <link href="http://www.cmu.edu/RSS/stories.rss" rel="alternate" title="Carnegie Mellon Homepage
Stories" type="application/rss+xml"/>
  ...
  <script language="JavaScript" src="js/dojo.js" type="text/javascript"></script>
  <script language="JavaScript" src="js/scripts.js" type="text/javascript"></script>
  <script language="javascript" src="js/jquery.js" type="text/javascript"></script>
  <script language="javascript" src="js/homepage.js" type="text/javascript"></script>
  <script language="javascript" src="js/app_ad.js" type="text/javascript"></script>
  ...
  <title>Carnegie Mellon University | CMU</title>
</head>
<body> ...
```

Proxy - Functionality

- **Should work on vast majority of sites**
  - Reddit, Vimeo, CNN, YouTube, NY Times, etc.
  - Some features of sites which require the POST operation (sending data to the website), will not work
    - Logging in to websites, sending Facebook message
  - HTTPS is not expected to work
    - Google (and some other popular websites) now try to push users to HTTPs by default; watch out for that
- **Cache previous requests**
  - Use LRU eviction policy
  - Must allow for concurrent reads while maintaining consistency
  - Details in write up

Proxy - Functionality

- **Why a multi-threaded cache?**
  - Sequential cache would bottleneck parallel proxy
  - Multiple threads can read cached content safely
    - Search cache for the right data and return it
    - Two threads can read from the same cache block
  - But what about writing content?
    - Overwrite block while another thread reading?
    - Two threads writing to same cache block?

Overview

- Parallelism
- Concurrency
- Mutexes/Semaphores
- Advance Topics in Concurrency

Parallelism

Let’s do some sums

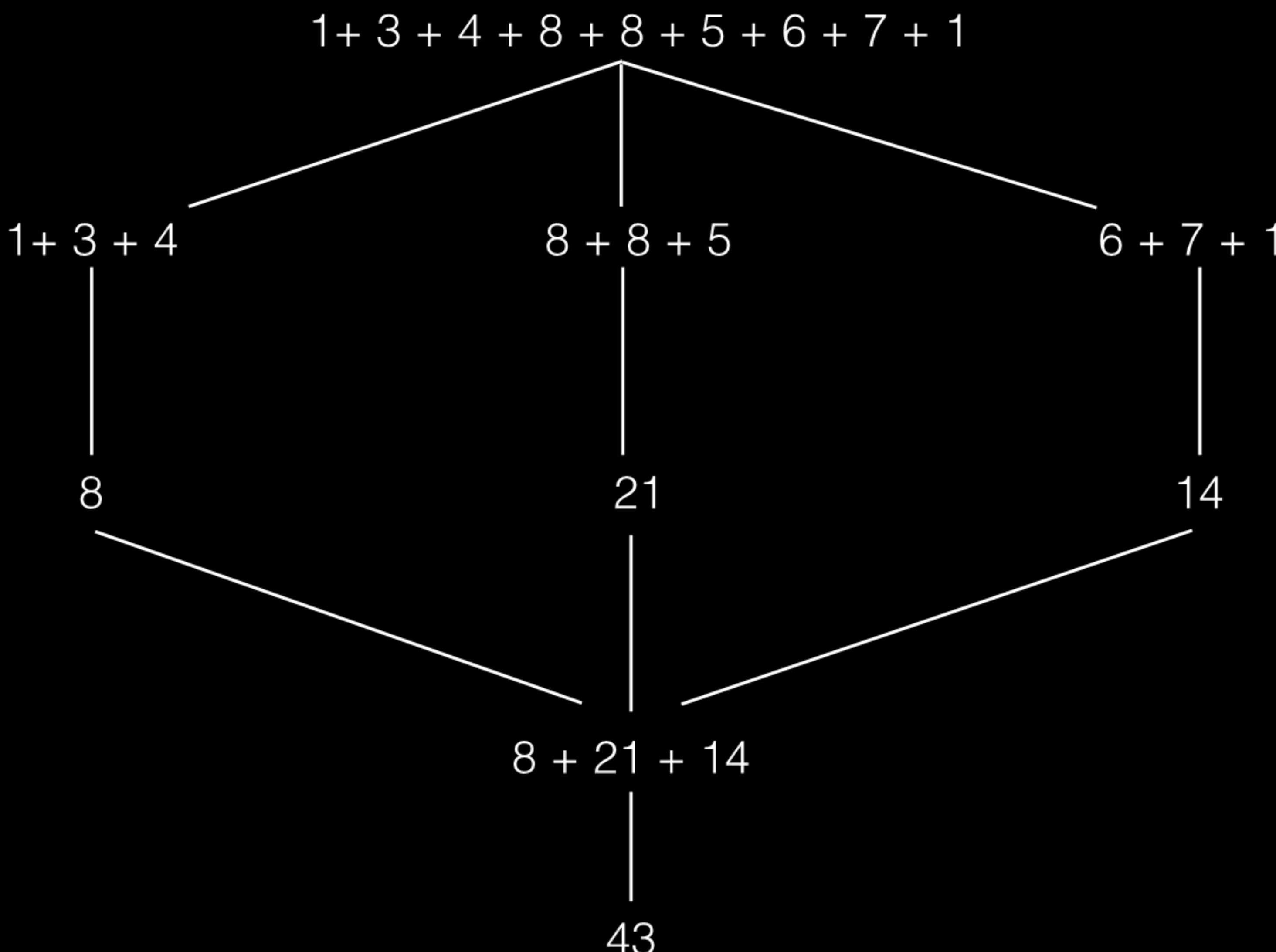
- Sum up elements of an *n*-size array
- i.e. 1 + 3 + 4 + 8 + 8 + 5 + 6 + 7 + 1

Non-Threaded

```
for (i = 0; i < nelems; i++) {
    result += psum[i];
}

return result
```

- i.e. 1+ 3 + 4 + 8 + 8 + 5 + 6 + 7 + 1
- 0 + 1 = 1
- 1 + 3 = 4
- 4 + 4 = 8
- …
- 42 + 1 = 43

Threaded



Let's do some sums

Non-Threaded

```
for (i = 0; i < nelems; i++) {
  result += psum[i];
}

return result
```

Let’s do some sums

Threaded

```
nelems_per_thread = nelems / nthreads;

/* Create threads and wait for them to finish */
for (i = 0; i < nthreads; i++) {
   myid[i] = i;
   Pthread_create(&tid[i], NULL, thread_fun, &myid[i]);
}

for (i = 0; i < nthreads; i++)
   Pthread_join(tid[i], NULL);

result = 0;

/* Add up the partial sums computed by each thread */
for (i = 0; i < nthreads; i++)
   result += psum[i*spacing];

/* Add leftover elements */
  for (e = nthreads * nelems_per_thread; e < nelems; e++)
      result += e;

return result;
```

Let's do some sums

Non-Threaded

```
for (i = 0; i < nelems; i++) {
    result += psum[i];
}

return result
```

Threaded

```
nelems_per_thread = nelems / nthreads;

/* Create threads and wait for them to finish */
for (i = 0; i < nthreads; i++) {
    myid[i] = i;
    Pthread_create(&tid[i], NULL, thread_fun, &myid[i]);
}

for (i = 0; i < nthreads; i++)
    Pthread_join(tid[i], NULL);

result = 0;

/* Add up the partial sums computed by each thread */
for (i = 0; i < nthreads; i++)
    result += psum[i*spacing];

/* Add leftover elements */
  for (e = nthreads * nelems_per_thread; e < nelems; e++)
      result += e;

return result;
```

Parallelizable Things

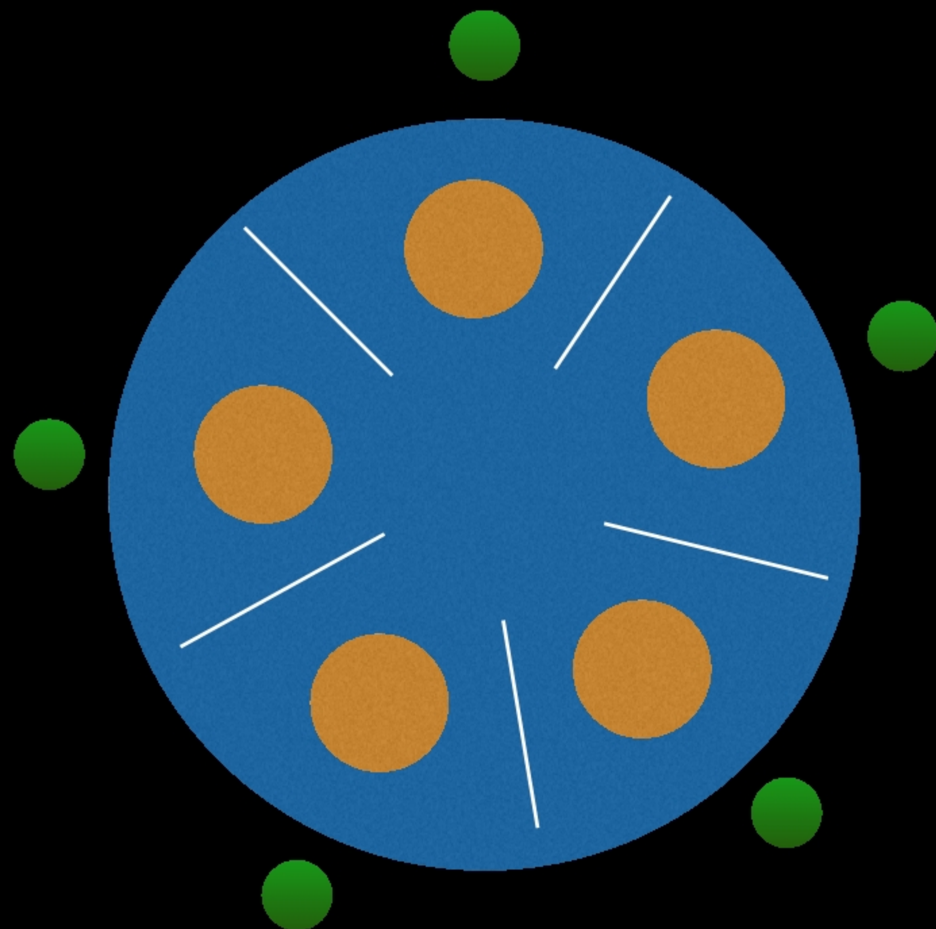
- Sorting - Merge/Quick Sort
- Search Problems - Dividing Search Space
- Commutative & Associative Operations (+,*)
- Serving multiple clients

How do I perform this sorcery?

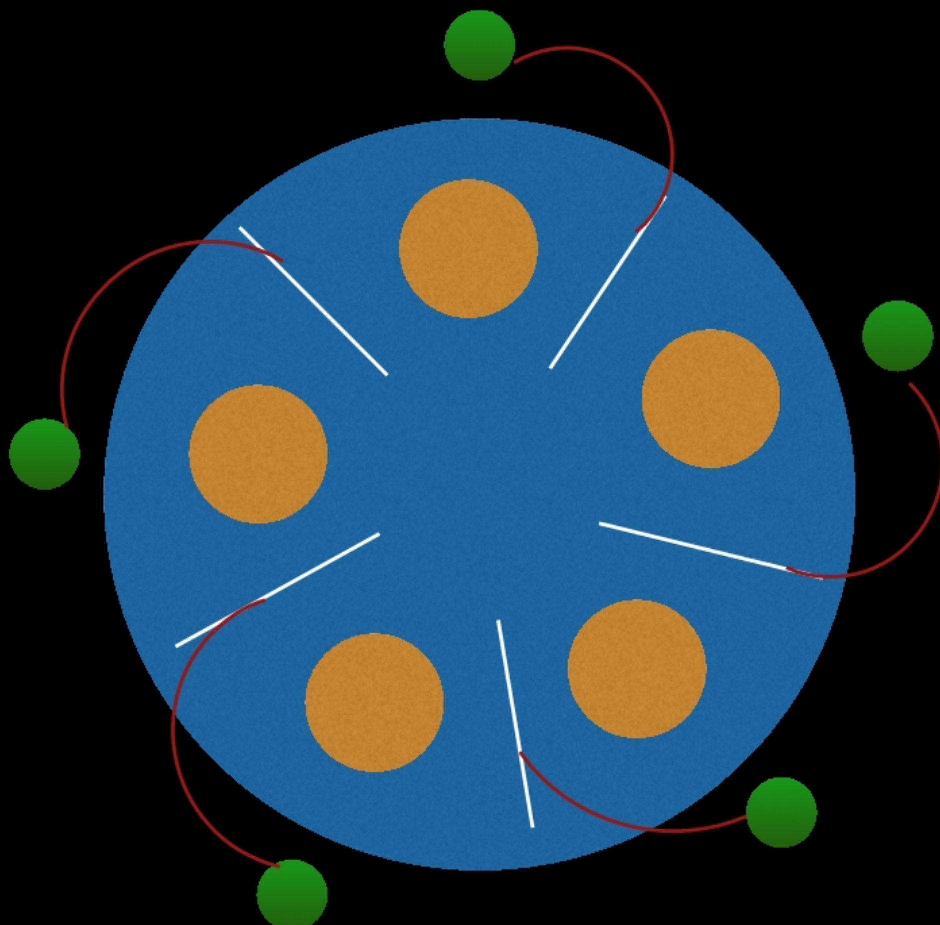
- Pthread Library in C (POSIX)
  - pthread_create (create thread)
  - pthread_join (like waited for threads)
  - pthread_detach (tell thread to kill itself if done)
- **<u>Check out the man pages!</u>**

Concurrency

Dining Philosophers

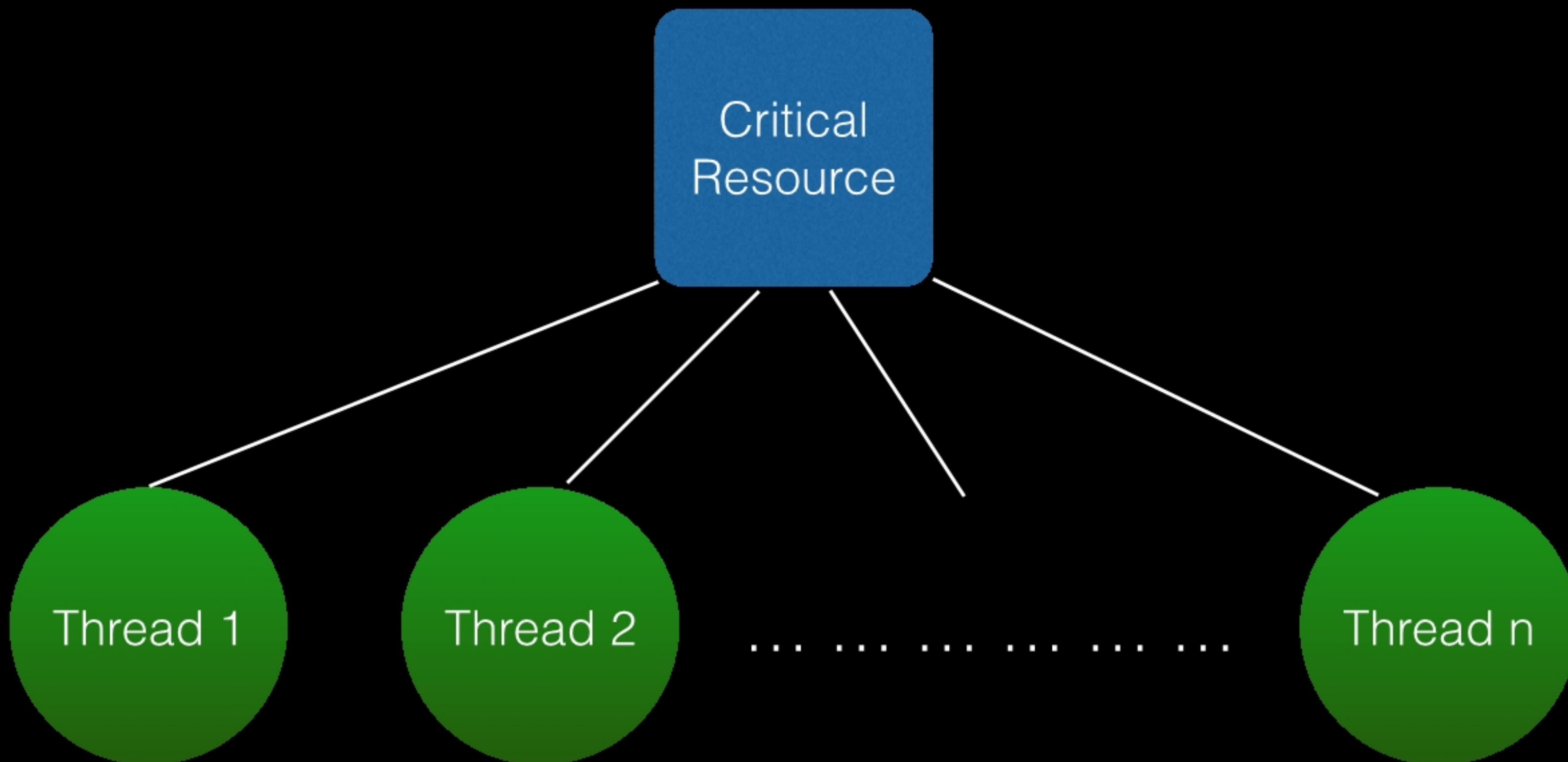


Dining Philosophers

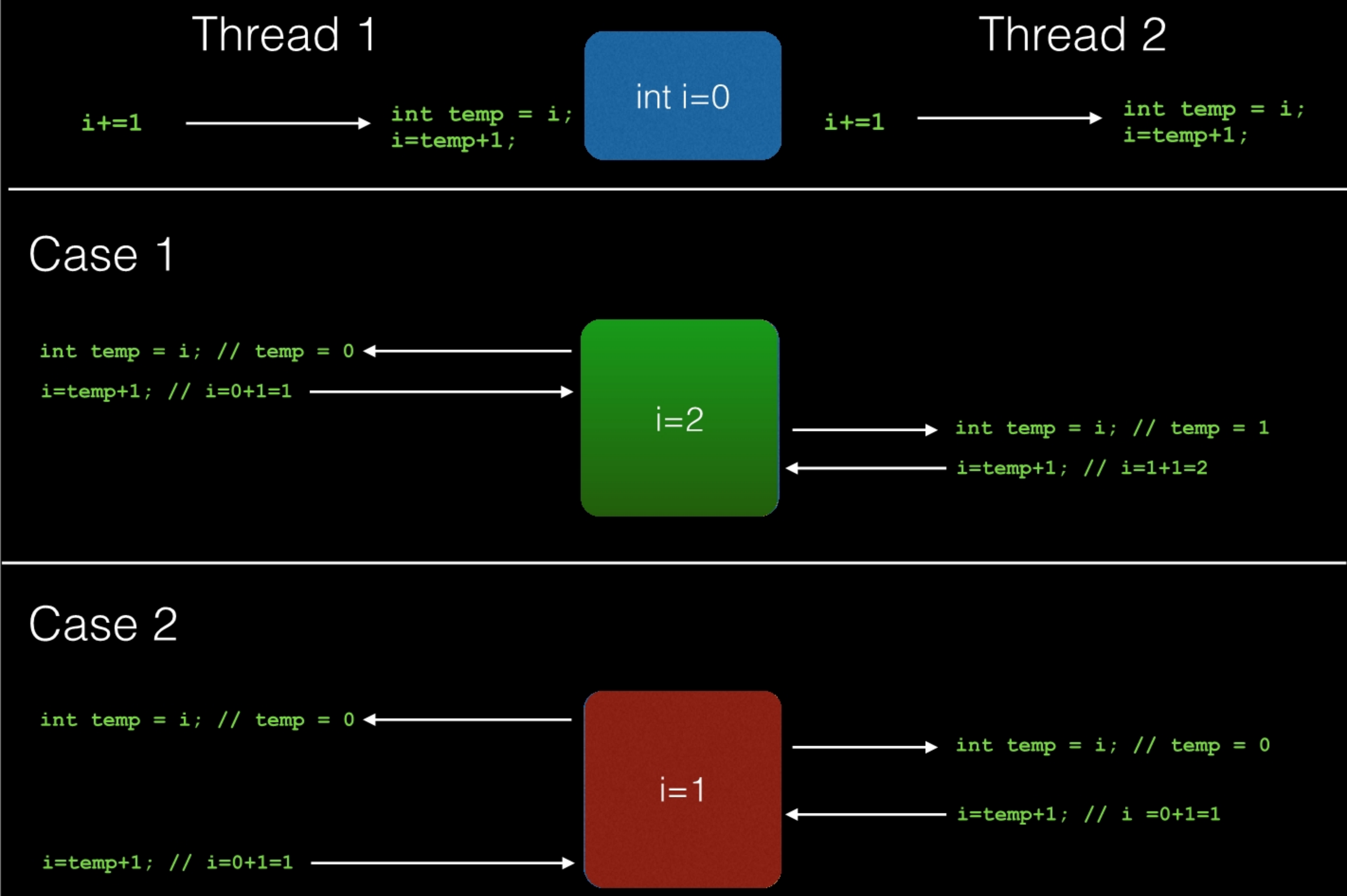


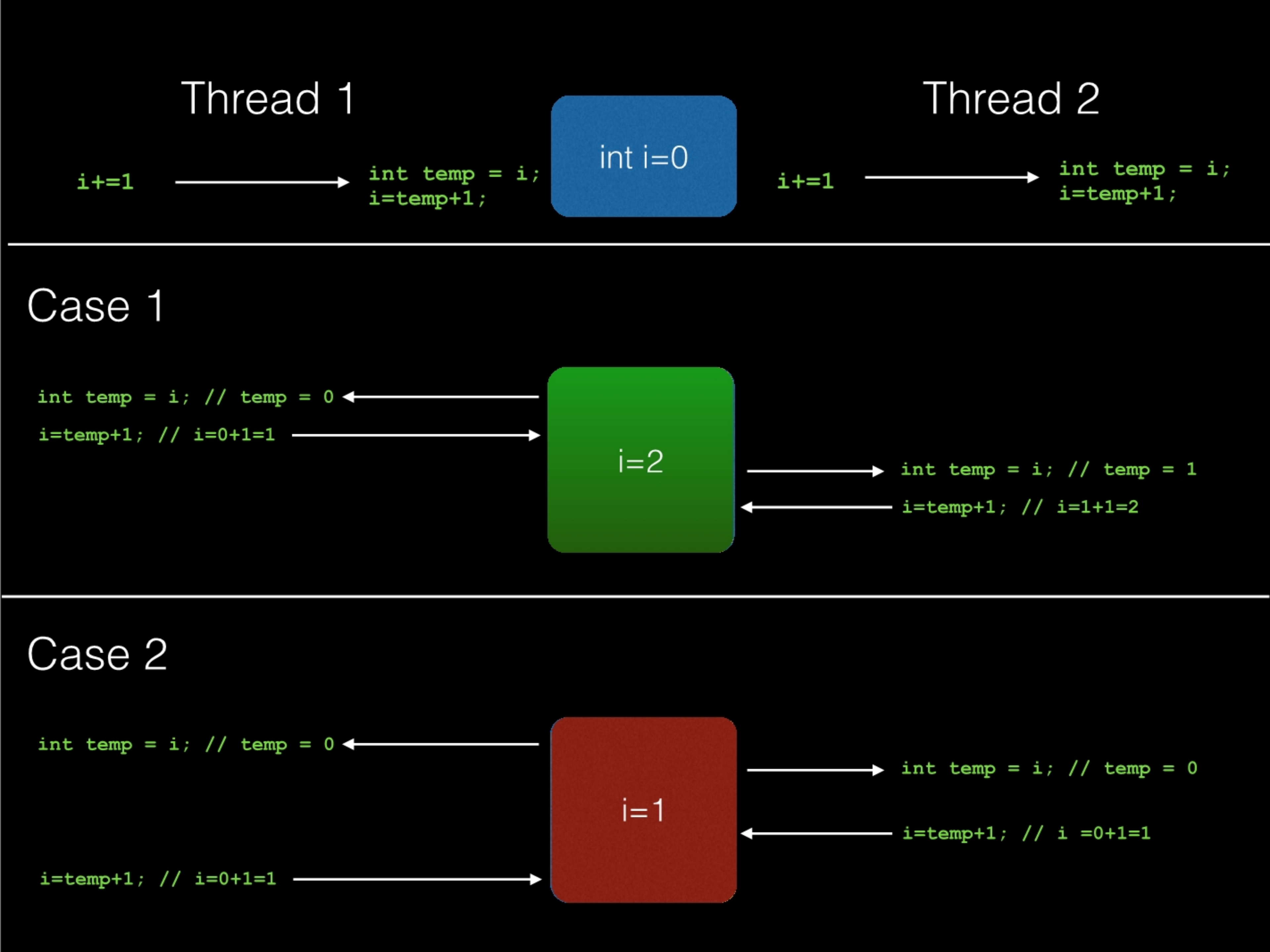
Concurrency Issues

- Race Conditions
- Deadlocks
- Starvation
- Unsafe Thread Functions



n Threads 1 CR

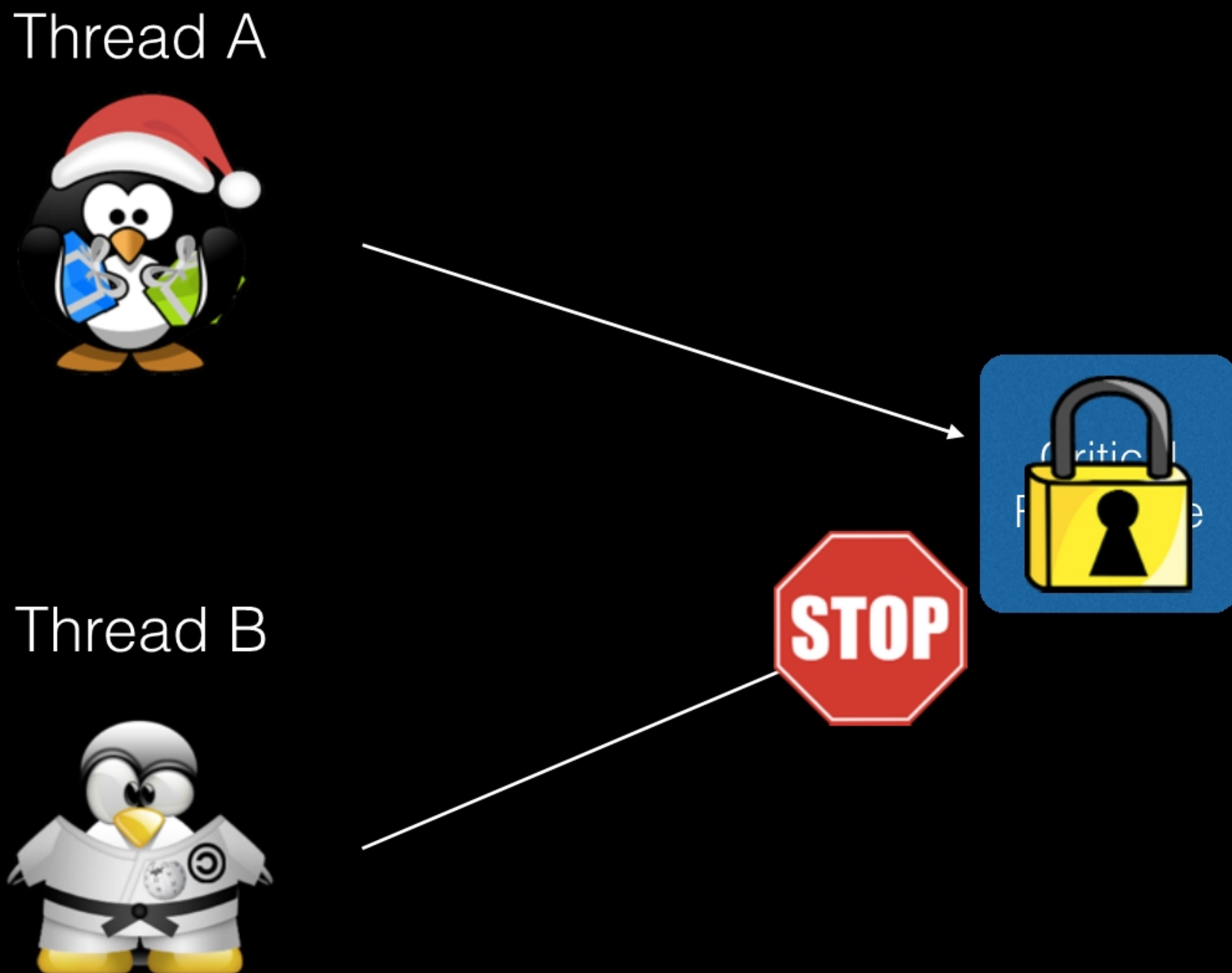

Thread 1
int i=0
Thread 2
i+=1
int temp = i;
i=temp+1;
i+=1
int temp = i;
i=temp+1;
Case 1
int temp = i; // temp = 0
i=temp+1; // i=0+1=1
i=2
int temp = i; // temp = 1
i=temp+1; // i=1+1=2
Case 2
int temp = i; // temp = 0
i=1
int temp = i; // temp = 0
i=temp+1; // i =0+1=1
i=temp+1; // i=0+1=1


Thread 1
Thread 2
int i=0
i+=1
int temp = i;
i=temp+1;
i+=1
int temp = i;
i=temp+1;
Case 1
int temp = i; // temp = 0
i=temp+1; // i=0+1=1
i=2
int temp = i; // temp = 1
i=temp+1; // i=1+1=2
Case 2
int temp = i; // temp = 0
i=1
int temp = i; // temp = 0
i=temp+1; // i =0+1=1
i=temp+1; // i=0+1=1

Code Example Incrementing Counter

```
volatile int ctr=0;

void* inc_counter(void* n) {
   for (i = 0; i < (int)n; i++) {
      ctr += 1;
   }
}


int main() {
   pthread_t pid1, pid2;
   pthread_create(&pid1,NULL, inc_counter, 100);
   pthread_create(&pid2,NULL, inc_counter, 100);

   pthread_join(pid1,NULL);
   pthread_join(pid2,NULL);

   printf("counter: %d\n", ctr);
}
```

What is the possible outputs of ctr?   2 - 200


Thread A
Thread B
STOP

Mutex

- Mutual Exclusion for a resource
- 1 use at a time
- pthread_mutex_init - Initialize Mutex
- P(&mutex) - Acquire Lock/Mutex
- V(&mutex) - Release Lock/Mutex

Thread 1

```
P(&mutex);
int temp = i;
i=temp+1;
V(&mutex);
```

int i=0
pthread_mutex_t mutex;
pthread_mutex_init
(&mutex,0)

Thread 2

```
P(&mutex);
int temp = i;
i=temp+1;
V(&mutex);
```

Case 2

```
P(&mutex);

int temp = i; // temp = 0

i=temp+1; // i=0+1=1

V(&mutex);
```


```
P(&mutex);
Waiting ...
Waiting ...
```

Thread 1

```
P(&mutex);
int temp = i;
i=temp+1;
V(&mutex);
```

int i=0
pthread_mutex_t mutex;
pthread_mutex_init
(&mutex,0)

Thread 2

```
P(&mutex);
int temp = i;
i=temp+1;
V(&mutex);
```

Case 2

```
P(&mutex);
int temp = i; // temp = 0
i=temp+1; // i=0+1=1
V(&mutex);
```


```
P(&mutex);
Waiting ...
Waiting ...
int temp = i; // temp = 1
i=temp+1; // i =1+1=2
V(&mutex);
```

Critical Sections

```
DepositUSD(int amt) {

    /* <<1>> */

    // Calculates the exchange rate of USD to CAD
    int cad_amt = USDToCAD(amt);

    /* <<2>> */

    // Deposit da monay
    account += cad_amt;

    /* <<3>> */

    // Print out amount deposited
    printf("Amount deposited: %d", cad_amt);

    /* <<4>> */
    return;
}
```

Critical Sections

```
DepositUSD(int amt) {

   /* <<1>> */

   // Calculates the exchange rate of USD to CAD
   int cad_amt = USDToCAD(amt);

   P(&mutex);

   // Deposit da monay
   account += cad_amt;

   V(&mutex);

   // Print out amount deposited
   printf("Amount deposited: %d", cad_amt);

   /* <<4>> */
   return;
}
```

Counting to 200

```
volatile int ctr=0;
pthread_mutex_t cnt_mutex;

void* inc_counter(void* n) {
   for (i = 0; i < (int)n; i++) {
      P(&cnt_mutex);
      ctr += 1;
      V(&cnt_mutex);
   }
}


int main() {
   pthread_init_mutex(cnt_mutex, 0);
   pthread_t pid1, pid2;
   pthread_create(&pid1,NULL, inc_counter, 100);
   pthread_create(&pid2,NULL, inc_counter, 100);

   pthread_join(pid1,NULL);
   pthread_join(pid2,NULL);

   printf("counter: %d\n", ctr);
}
```

What is the possible outputs of ctr? 200

Mutex

- Mutual Exclusion for a resource
- n use at a time
- sem_init - Initialize Mutex
- P(&semaphore) - Acquire Lock/Mutex
- V(&semaphore) - Release Lock/Mutex

Semaphores

- Mutexes, but allow *t* threads accessing at once
- Example scenario: We want to have 4 people using the service at once

Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
   P(&server_sem);
   doStuff();
   V(&server_sum);
}
```

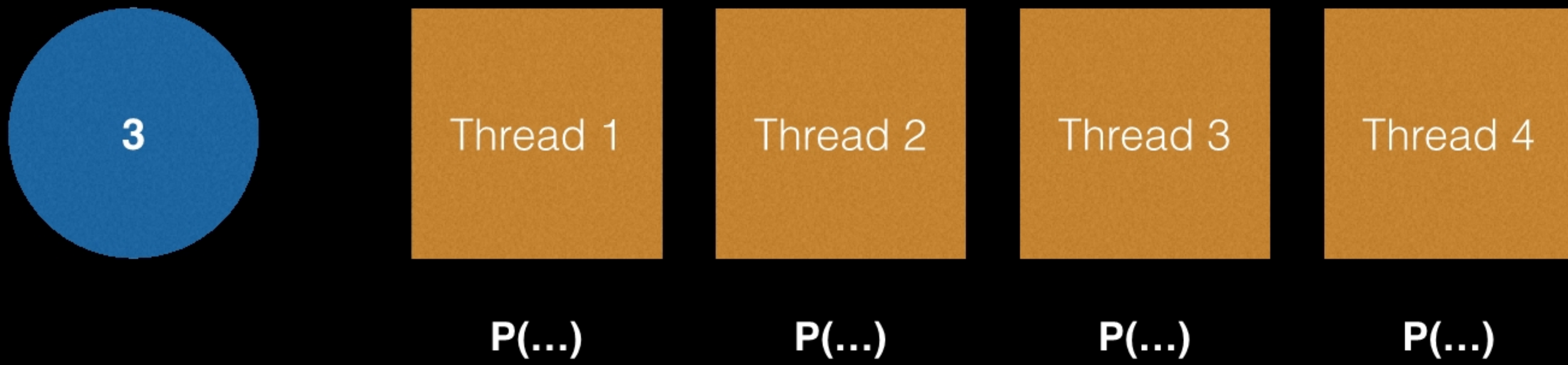
Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```

server_sem



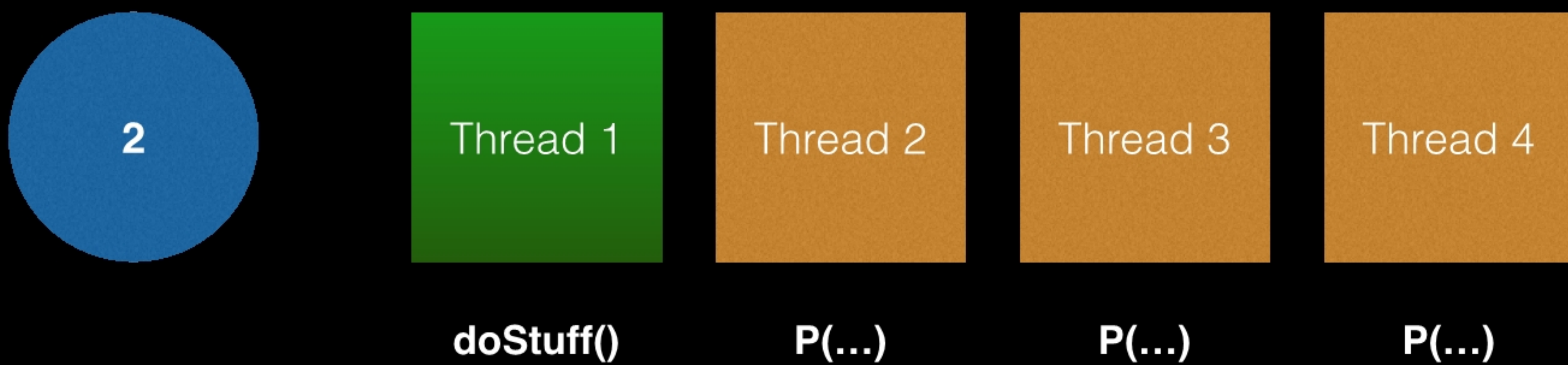
Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```

server_sem



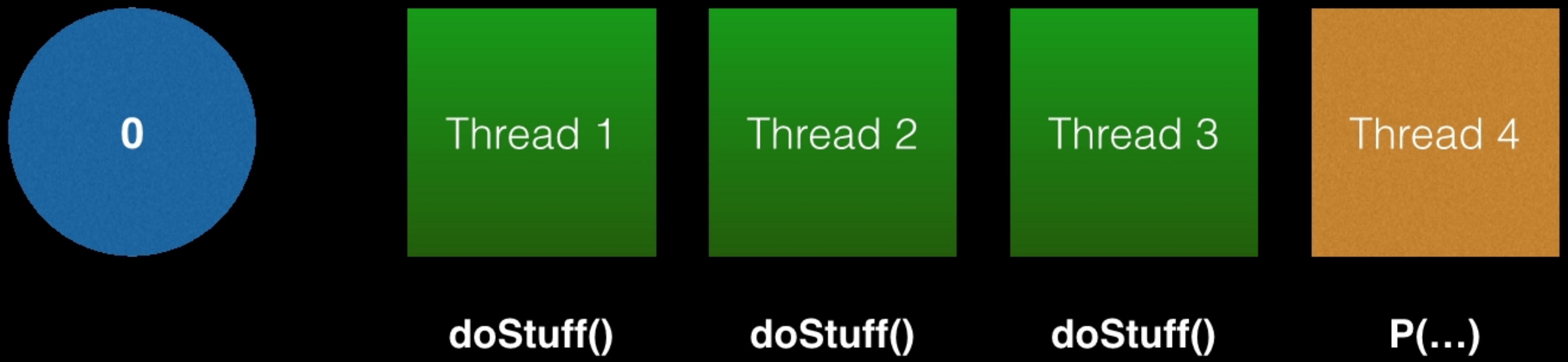
Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```

server_sem



Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```

server_sem

0

Thread 1

Thread 2

Thread 3

Thread 4

doStuff()

doStuff()

doStuff()

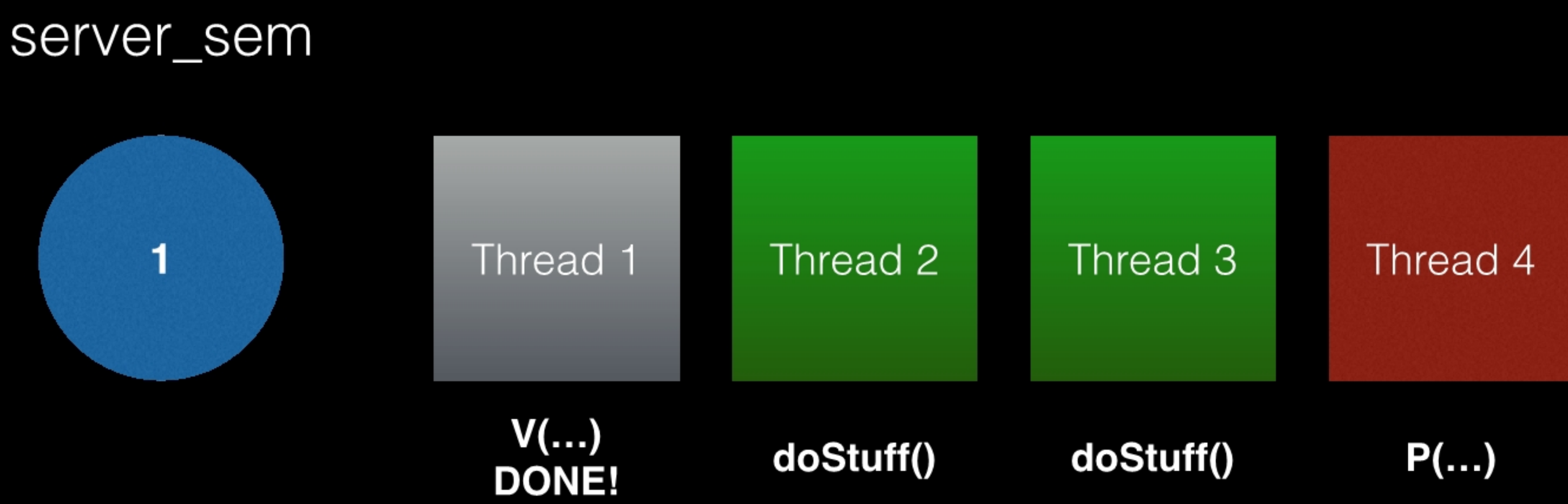
P(…)

Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```

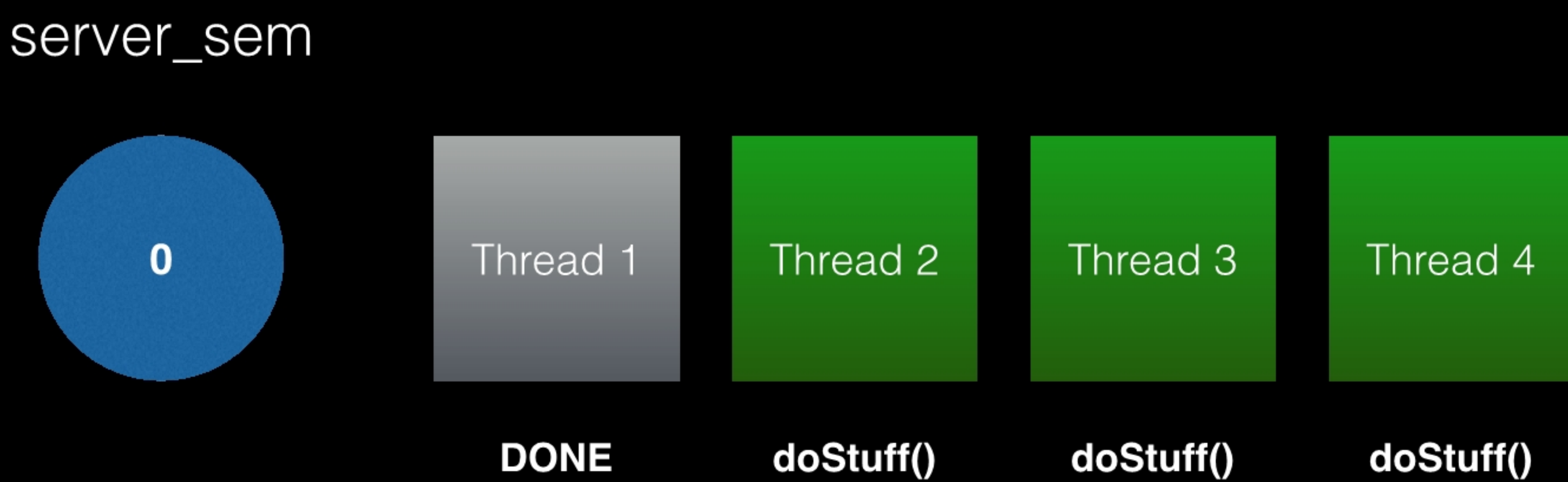


Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```



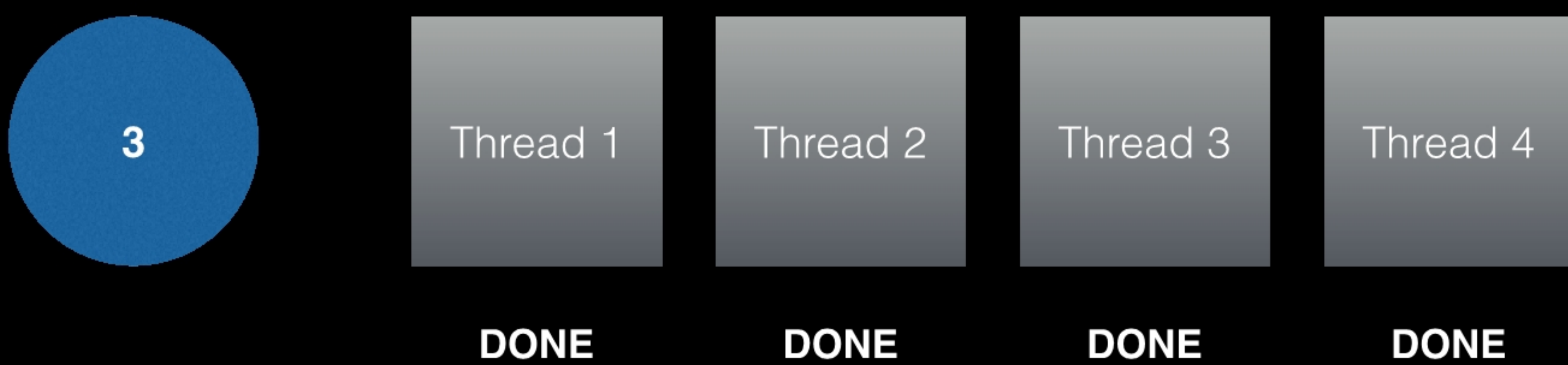
Init Semaphore

```
sem_t server_sem;
sem_init(&server_sum,0,3);
```

Thread Call

```
Connect() {
    P(&server_sem);
    doStuff();
    V(&server_sum);
}
```

server_sem



Mutexes vs Semaphores

- Mutex: Mutual Exclusion lock for a resource
- Semaphore: Generalized Mutex with n uses at once

Deadlocks

Thread 1

```
P(&A);
P(&B);
int c = a + b;
V(&A);
V(&B);
```

Thread 2

```
P(&B);
P(&A);
int c = a + b;
V(&A);
V(&B);
```

Deadlocks

Thread 1

```
P(&A);
P(&B);
int c = a + b;
V(&A);
V(&B);
```

Thread 2

```
P(&B);
P(&A);
int c = a + b;
V(&A);
V(&B);
```

Thread 1 locks A, Thread 2 locks B

Deadlocks

Thread 1

```
P(&A);
P(&B);
int c = a + b;
V(&A);
V(&B);
```

Thread 2

```
P(&B);
P(&A);
int c = a + b;
V(&A);
V(&B);
```

Thread 1 wants to acquire B, but B is locked by Thread 2

Thread 2 wants to acquire A, but A is locked by Thread1

Threads are waiting on each other forever - Deadlock!

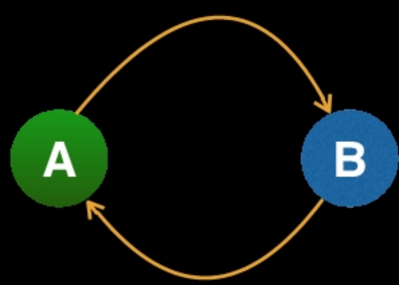
Deadlocks : Graph Cycles

Thread 1

```
P(&A);
P(&B);
int c = a + b;
V(&A);
V(&B);
```

Thread 2

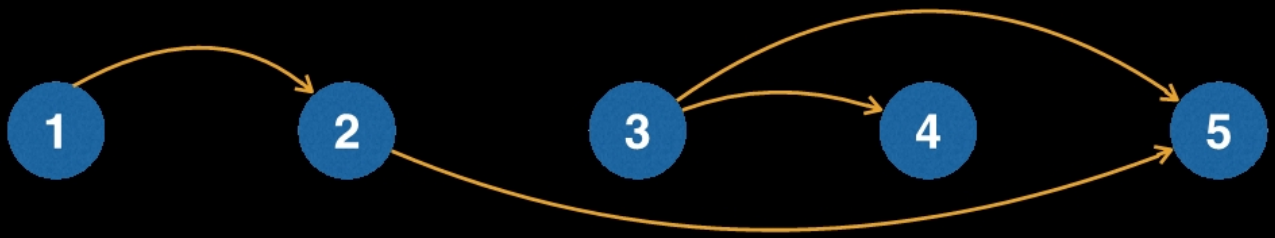
```
P(&B);
P(&A);
int c = a + b;
V(&A);
V(&B);
```



Cycle Detected!

How to prevent deadlocks

- Have a absolute ordering of mutexes and acquire them in the order for every critical section



- Write a deadlock detector! But, how if that deadlocks? Write another … and …

Reader/Writer Locks & Starvation

Reader

```
int readcnt;     /* Initially 0 */
sem_t mutex, w; /* Both initially 1 */

void reader(void)
{
  while (1) {
    P(&mutex);
    readcnt++;
    if (readcnt == 1) /* First in */
      P(&w);
    V(&mutex);

    /* Reading happens here */

    P(&mutex);
    readcnt--;
    if (readcnt == 0) /* Last out */
      V(&w);
    V(&mutex);
  }
}
```

Writer

```
void writer(void)
{
  while (1) {
    P(&w);

    /* Writing here */

    V(&w);
  }
}
```

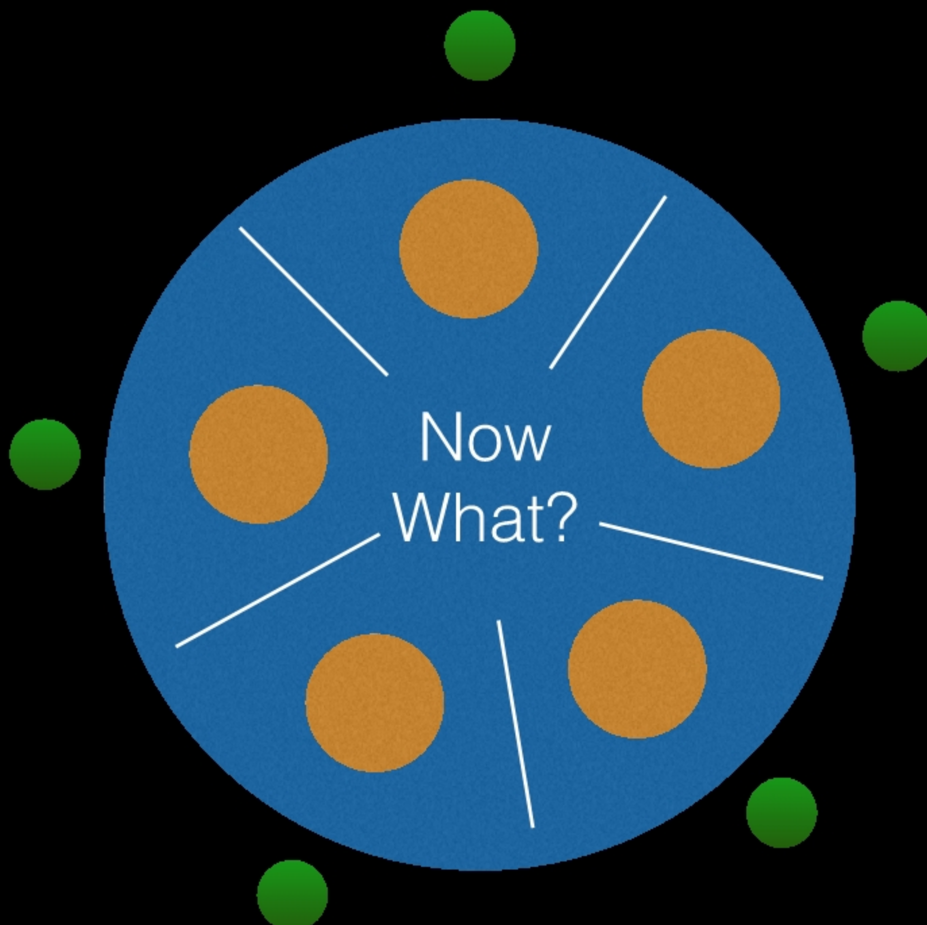
Many Reads, Single Write

Where is a possibility for starvation here?

Unsafe Thread Functions

- Class 1: Functions that do not protect shared variables
- Class 2: Functions that keep state across multiple invocations
- Class 3: Functions that return a pointer to a static variables
- Class 4: Functions that call Thread-unsafe functions

Dining Philosophers



Advance Topics in Concurrency

Slides from here onwards are additional text
for leisure reading :)

Conditional Variables

- Conditional Variables can be used to release a mutex until a condition is met.
- cond_wait : releases the mutex and sleeps/blocks until it is signaled to wake up
- cond_signal : signals one of the waiting conditional variables to wake and it tries to acquire the mutex
- cond_broadcast : signals all conditional variables waiting
- Each conditional is linked to a mutex (when it is slept, it releases the lock, and when it is woken up, it acquires the lock.

Example : Creating a Concurrent Queue

```
sem_t qmutex;
sem_init(&qmutex, 0, 1); // Initialize mutex
cond_init(&emptycond, 0); // Initialize Condvar
```

```
dequeue (){
   P(&qmutex);

   while (true) {
      if (len(Q.list)>0) {
         Q.list[0];
         Q.list.removeHead();
         break;
      }
      cond_wait(&emptycond, &qmutex);
   }

   V(&qmutex);
}
```

```
enqueue (elem e) {
   P(&qmutex);
   Q.list.append(e);
   cond_signal(&emptycond);
   V(&qmutex);
}
```

User Space Threads (Fibers)

- Usually OS responsible for context switching kernel threads (i.e. pthreads)
- User Space Threads are managed by a program/library
- Context switches are less expensive
- http://www.evanjones.ca/software/threading.html

Hybrid Threading Model
M:N