Previously, on CS4410...
### What are the implications?

<table>
<thead>
<tr>
<th>Hoare</th>
<th>Mesa</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤ Signaling is atomic with the resumption of waiting thread</td>
<td>✤ notify() and notifyAll() are hints</td>
</tr>
<tr>
<td>□ shared state cannot change before waiting thread is resumed!</td>
<td>□ adding them affects performance, never safety</td>
</tr>
<tr>
<td>□ safety requires to signal only when condition holds</td>
<td>□ Shared state must be checked in a loop (the condition could have changed since the thread was notified!)</td>
</tr>
<tr>
<td>✤ Shared state can be checked using an if statement</td>
<td>✤ Simple implementation</td>
</tr>
<tr>
<td>✤ Makes it easier to prove liveness</td>
<td>✤ Resilient to spurious wakeup</td>
</tr>
<tr>
<td>✤ Tricky to implement</td>
<td></td>
</tr>
</tbody>
</table>
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
How does Hoare pass the monitor lock?
Kid and Cook Threads

Kid

```cpp
close_read()
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```

Cook

```cpp
close_read()
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
        drive_to_home();
        watch_got();
        sleep();
}
```

Monitor BurgerKing

```cpp
Lock mlock;

int numburgers = 0;
condition hungrykid;

void kid_eat()
    mlock.acquire()
    while (numburgers==0)
        hungrykid.wait()
        numburgers -= 1
    mlock.release()

void makeburger()
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
```
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;

  int numburegers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numburegers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }

  void cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
      BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
  }
}
```c
Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        +numburger;
        hungrykid.notify();
        mlock.release()
    }
}
```
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Running

Kid and Cook Threads

cook gets the CPU

Ready
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;
  int numbburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0) 
      hungrykid.wait()
    numbburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(!not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}

cook executes
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numbburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numbburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

cook back on ready Q

Running

Ready

cook back on ready Q
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Running

Ready

boy gets the CPU
```c
 Kid
  and
  Cook
 Threads

 kid_main()
{
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

 cook_main()
{
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
  {
    BK.makeburger();
  }
  drive_to_home();
  watch_got();
  sleep();
}

 Monitor BurgerKing {
  Lock mlock;

  int numburgers = 0;
  condition hungrykid;

  void kid_eat()
  {
    mlock.acquire()  
    while (numburgers==0)
      hungrykid.wait()
    numburgers -= 1
    mlock.release()
  }

  void makeburger()
  {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}
```

Ready

0

Running

boy tries to enter monitor
Kid and Cook Threads

```c
Kid

kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Monitor BurgerKing {
    Lock mlock;
    int numb Burgers = 0;
    condition hungrykid;

    void kid_eat()
    {  
        mlock.acquire()
        while (num Burgers == 0)
            hungrykid.wait()
        numb Burgers -= 1
        mlock.release()
    }

    void makeburger()
    {  
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

Cook

cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```
Kid and Cook Threads

Running Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

boy back on ready Q

Ready
Kid and Cook Threads

```c
kid_main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing {
    Lock mlock;
    int numbigers = 0;
    condition hungrykid;

    void kid_eat()
    {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbigers -= 1
        mlock.release()
    }

    void makeburger()
    {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
Kid and Cook Threads

```c
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;

    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers==0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}```
Kid and Cook Threads

```
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

Monitor BurgerKing {
    Lock mlock;

    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

Q

Ready

Running

monitor has lock Q
Kid and Cook Threads

```c
int numburgers = 0;
condition hungrykid;

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}
```

```c
cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

```c
dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();

Kid and Cook Threads
```
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Monitor BurgerKing {
    lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0) {
            hungrykid.wait()
            numburgers -= 1
        }
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        numburgers += 1
        hungrykid.notify()
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm) {
        BK.makeburger();
        drive_to_home();
        watch_got();
        sleep();
    }
}

cook gets the CPU
Kid and Cook Threads

Kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
Kid and Cook Threads

*Monitor BurgerKing*

```c
int numburgers = 0;
condition hungrykid;

void kid_eat() {
    mlock.acquire()
    while (numburgers==0)
        hungrykid.wait()
    numburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}
```

Kid Main:

```c
dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();
}
```

Cook Main:

```c
cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
sleep();
}
```
Kid and Cook Threads

```c
Kid main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Cook main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
    
    BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```
Kid and Cook Threads

```c
#include <pthread.h>

int numburgers = 0;
condition hungrykid;

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}
```

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

Monitor BurgerKing {
    lock mlock;
    }
Monitor BurgerKing {
    lock mlock;
    int numburgers = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

cook_main()
{
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    int numburers = 0;
    condition hungrykid;
}

Monitor

cook_main() {

    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Kid and Cook Threads

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

void kid_eat() {
    mlock.acquire()
    while (numburesrs==0)
        hungrykid.wait()
    numburers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburer;
    hungrykid.notify();
    mlock.release()
}

boy releases monitor lock & waits for hungrykid signal
Kid and Cook Threads

```c
#include <thread>
#include <mutex>
#include <condition_variable>

void kid_eat() {
    std::lock_guard<std::mutex> lock(mlock);
    while (numburgers == 0)
        hungrykid.wait();
    numburgers -= 1;
    lock.unlock();
}

void makeburger() {
    mlock.acquire();
    ++numburger;
    hungrykid.notify();
    mlock.release();
}
```

```c
int numburgers = 0;
condition hungrykid;

void kid_eat() {
    BK.kid_eat();
    BK.kid_eat();
    BK.kid_eat();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    facetime_Gma();
    BK.kid_eat();
}

void makeburger() {
    BK.makeburger();
    BK.makeburger();
    BK.makeburger();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```

**Waiting**
- `kid_eat`
- `makeburger`
**Running**
- `cook_main`
- `cook_main`

*cook joins ready Q with monitor lock that boy released*
Kid

\[
\text{kid\_main()}\{ \\
\text{dig\_in\_mud();} \\
\text{BK.kid\_eat();} \\
\text{bathe();} \\
\text{draw\_on\_walls();} \\
\text{BK.kid\_eat();} \\
\text{facetime\_Karthik();} \\
\text{facetime\_Gma();} \\
\text{BK.kid\_eat();} \\
\}
\]

Cook

\[
\text{cook\_main()}\{ \\
\text{wake();} \\
\text{shower();} \\
\text{drive\_to\_work();} \\
\text{while\((!\text{not\_5pm})\)} \\
\text{\quad \text{BK.makeburger();}} \\
\text{\quad \text{drive\_to\_home();}} \\
\text{\quad \text{watch\_got();}} \\
\text{\quad \text{sleep();}} \\
\}
\]

Monitor BurgerKing

\[
\text{int\ numbburgers = 0;} \\
\text{condition\ hungrykid;} \\
\]

\[
\text{void\ kid\_eat()}\{ \\
\text{\quad mlock.acquire()} \\
\text{\quad while\((\text{numburgers}\leq 0)\)} \\
\text{\quad\quad hungrykid.wait()} \\
\text{\quad\quad numbburgers -= 1} \\
\text{\quad mlock.release()} \\
\}
\]

\[
\text{void\ makeburger()}\{ \\
\text{\quad mlock.acquire()} \\
\text{\quad ++numburger;} \\
\text{\quad hungrykid.notify();} \\
\text{\quad mlock.release()} \\
\}
\]
Monitor BurgerKing {
    lock mlock;
    int numbigers = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numbigers == 0)
        hungrykid.wait()
    numbigers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    numbigers += 1
    hungrykid.notify();
    mlock.release()
}

kid_main() {
dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();
}

cook_main() {
wake();
shower();
drive_to_work();
while(not_5pm)
    BK.makeburger();
drive_to_home();
watch_got();
sleep();
}
```c
Monitor BurgerKing {
    lock mlock;
    int numbburgers = 0;
    condition hungrykid;
}

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numbburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numbürger;
    hungrykid.notify();
    mlock.release()
}
```

```c
void kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```

```c
void cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```
Kid and Cook Threads

Monitor BurgerKing {
    lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

cook signals a hungry kid

Kid	and	Cook	Threads

Running

Waiting

Q:
Waiting
1

Kid	main()
{
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Kid and Cook Threads

Monitor BurgerKing {
Lock mlock;

int numbburgers = 0;
condition hungrykid;

void kid_eat() {
  mlock.acquire()
  while (numburgers==0)
    hungrykid.wait()
  numbburgers -= 1
  mlock.release()
}

void makeburger() {
  mlock.acquire()
  ++numburger;
  hungrykid.notify();
  mlock.release()
}
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();

Running

Running

cook releases monitor lock; girl goes back to ready Q

Ready
Kid and Cook Threads

**Monitor BurgerKing**

```c
int numbburgers = 0;
condition hungrykid;

void kid_eat()
{
mlock.acquire()
    while(numbburgers == 0)
        hungrykid.wait()
    numbburgers -= 1
mlock.release()
}

void makeburger()
{
mlock.acquire()
    ++numburger;
    hungrykid.notify();
mlock.release()
}
```

**cook_main()**

```c
wake();
shower();
drive_to_work();
while(not_5pm)
    BK.makeburger();
drive_to_home();
watch_got();
sleep();
```
**Kid and Cook Threads**

```
Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire() 
        while (numburgers == 0) 
            hungrykid.wait() 
        numbburgers -= 1 
        mlock.release() 
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;

    int numbhours = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numhours == 0)
            hungrykid.wait()
        numhours -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();
Kid and Cook Threads

Monitor BurgerKing {
  Lock mlock;
  int numbburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numbburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

Monitor BurgerKing {
    Lock mlock;
    int numbrellas = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numbrellas == 0)
            hungrykid.wait()
        numbrellas -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numbrellas;
        hungrykid.notify();
        mlock.release()
    }
}
Monitor BurgerKing {
  Lock mlock;

  int numbburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numbburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire();
        while (numburgers == 0) {
            hungrykid.wait();
            numburgers -= 1
        }
        mlock.release()
    }

    void makeburger() {
        mlock.acquire();
        ++numburger;
        hungrykid.notify();
        mlock.release();
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm) {
        BK.makeburger();
    }
    drive_to_home();
    watch_got();
    sleep();
}
**Kid and Cook Threads**

```c
Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}
```

```c
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```

```c
cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```
```c
Monitor BurgerKing {
Lock mlock;
int numbburgers = 0;
condition hungrykid;

void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
        hungrykid.wait()
    numbburgers -= 1
    mlock.release()
}

void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
}
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}
```
Monitor BurgerKing {
    Lock mlock;
    int numbigators = 0;
    condition hungrykid;
    void kid_eat() {
        mlock.acquire()
        while (numbigators == 0)
            hungrykid.wait()
        numbigators -= 1
        mlock.release()
    }
    void makeburger() {
        mlock.acquire()
        ++numbigators
        hungrykid.notify()
        mlock.release()
    }
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numbthers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numbthers == 0)
            hungrykid.wait()
        numbthers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numbthers;
        hungrykid.notify();
        mlock.release()
    }
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while (not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}

dig_in_mud();
BK.kid_eat();
bathe();
draw_on_walls();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();
facetime_Karthik();
facetime_Gma();
BK.kid_eat();

boy acquires monitor lock

Ready

Running
Monitor BurgerKing {
    Lock mlock;
    int numbhamburgers = 0;
    condition hungrykid;
    void kid_eat() {
        mlock.acquire()
        while (numbhamburgers == 0)
            hungrykid.wait()
        numbhamburgers -= 1
        mlock.release()
    }
    void makeburger() {
        mlock.acquire()
        ++numbhamburgers;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Kid and Cook Threads

Monitor BurgerKing {
    Lock mlock;
    int numbburgers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire()
        while (numburgers == 0)
            hungrykid.wait()
        numbburgers -= 1
        mlock.release()
    }

    void makeburger() {
        mlock.acquire()
        ++numburger;
        hungrykid.notify();
        mlock.release()
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
Monitor BurgerKing {
  Lock mlock;
  int numburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
Kid and Cook Threads

```
Monitor BurgerKing {
  Lock mlock;
  int numbburgers = 0;
  condition hungrykid;

  void kid_eat() {
    mlock.acquire()
    while (numburgers == 0)
      hungrykid.wait()
    numbburgers -= 1
    mlock.release()
  }

  void makeburger() {
    mlock.acquire()
    ++numburger;
    hungrykid.notify();
    mlock.release()
  }
}

kid_main() {
  dig_in_mud();
  BK.kid_eat();
  bathe();
  draw_on_walls();
  BK.kid_eat();
  facetime_Karthik();
  facetime_Gma();
  BK.kid_eat();
}

cook_main() {
  wake();
  shower();
  drive_to_work();
  while(not_5pm)
    BK.makeburger();
  drive_to_home();
  watch_got();
  sleep();
}
```

- boy releases monitor lock and waits for hungrykid signal.
- boy releases monitor lock and waits for hungrykid signal.

Q: Waiting
Q: Waiting

Running

Ready
Kid and Cook Threads

```
Monitor BurgerKing {
    Lock mlock;
    int numburers = 0;
    condition hungrykid;

    void kid_eat() {
        mlock.acquire();
        while (numburers == 0)
            hungrykid.wait();
        numburers -= 1
        mlock.release();
    }

    void makeburger() {
        mlock.acquire();
        ++numburger;
        hungrykid.notify();
        mlock.release();
    }
}

kid_main() {
    dig_in_mud();
    BK.kid_eat();
    bathe();
    draw_on_walls();
    BK.kid_eat();
    facetime_Karthik();
    facetime_Gma();
    BK.kid_eat();
}

cook_main() {
    wake();
    shower();
    drive_to_work();
    while(not_5pm)
        BK.makeburger();
    drive_to_home();
    watch_got();
    sleep();
}
```
and so forth...
Deadlock

Chapter 32 in “Three Easy Steps”
Chapter 19 in the Harmony Book
Dining Philosophers

\[ P_i: \text{do forever} \]
\[ \quad \text{acquire( left(i) );} \]
\[ \quad \text{acquire( right(i) );} \]
\[ \quad \text{eat;} \]
\[ \quad \text{release( left(i) );} \]
\[ \quad \text{release( right(i) );} \]
\[ \quad \text{end} \]

left(i): i

right(i): (i+1) mod 5
Dining Philosophers in Harmony

```python
from synch import Lock, acquire, release

c = const N = 5

forks = [Lock(),] * N

def diner(which):
    left, right = (which, (which + 1) % N):
    while choose({False, True}):
        acquire(forks[left])
        acquire(forks[right])
        # dine
        release(forks[left])
        release(forks[right])
        # think

for i in {0..N-1}:
    spawn diner(i)
```
Dining Philosophers in Harmony

**Final state** (all threads have terminated or are blocked):

* Threads:
  * T1: (blocked) diner(0) --> acquire(?forks[1])
    * about to execute atomic section in line synch/35
  * T2: (blocked) diner(1) --> acquire(?forks[2])
    * about to execute atomic section in line synch/35
  * T3: (blocked) diner(2) --> acquire(?forks[3])
    * about to execute atomic section in line synch/35
  * T4: (blocked) diner(3) --> acquire(?forks[4])
    * about to execute atomic section in line synch/35
  * T5: (blocked) diner(4) --> acquire(?forks[0])
    * about to execute atomic section in line synch/35

* Variables:
  * forks: [ True, True, True, True, True, True ]
Problematic Emergent Properties

- **Starvation**: Process waits forever

- **Deadlock**: a set of processes exist, where each is blocked and can become unblocked only by the action of another process in the same set
  - Deadlock implies Starvation (but not viceversa)
  - Starvation often tied to fairness — which requires that a process be not forever blocked on a condition that becomes (i) continuously true or (ii) infinitely-often true

Testing for starvation or deadlock is difficult in practice
More Examples of Deadlock

Example 1 (initially in1 = in2 = False):

\[
\begin{align*}
in1 &= \text{True}; \quad \text{await not in2}; \quad \text{in1} = \text{False} \\
&\quad // \\
in2 &= \text{True}; \quad \text{await not in1}; \quad \text{in2} = \text{False}
\end{align*}
\]

Example 2 (initially lk1 = lk2 = released):

\[
\begin{align*}
\text{acquire(lk1)}; \quad \text{acquire(lk2)}; \quad \text{release(lk2)}; \quad \text{release(lk1)} \\
&\quad // \\
\text{acquire(lk2)}; \quad \text{acquire(lk1)}; \quad \text{release(lk1)}; \quad \text{release(lk2)}
\end{align*}
\]
Set of resources requiring “exclusive” access

- Might be “k exclusive access” if k instances of resource are available
- Examples: buffers, packets, I/O devices, processors

Protocol to access a resource causes blocking

- If resource is free, access is granted and process proceeds
  - Uses resource
  - Releases resource
- If resource is in use, process blocks
A Graph Theoretic Model of Deadlock

Computer system modeled as a RAG, a directed graph $G(V, E)$

- $V = \{P_1, \ldots, P_n\} \cup \{R_1, \ldots, R_n\}$
- $E = \{\text{edges from a resource to a process}\} \cup \{\text{edges from a process to a resource}\}$
Necessary conditions for deadlock

Deadlock only if they all hold

1. Bounded resources
   Acquire can block invoker

2. No preemption
   the resource is mine, MINE! (until I release it)

3. Wait while holding
   holds one resource while waiting for another

4. Circular waiting
   \( P_i \) waits for \( P_{i+1} \) and holds a resource requested by \( P_{i-1} \)
   sufficient if one instance of each resource

Not sufficient in general

Diagram:
- \( P_0 \), \( P_1 \), \( P_2 \), \( P_3 \), \( P_4 \)
- Cycle: \( P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_0 \)
- Waiting for: \( P_0 \), \( P_1 \), \( P_2 \), \( P_3 \), \( P_4 \)
- Owned by: \( P_0 \), \( P_1 \), \( P_2 \), \( P_3 \), \( P_4 \)
Deadlock is Undesirable!

- **Deadlock prevention**: Ensure that a necessary condition cannot hold.
- **Deadlock avoidance**: System does not allocate resources that may lead to a deadlock.
- **Deadlock detection**: Allow system to deadlock; detect it; recover.
Testing for cycles

- **Reduction Algorithm**
  - Find a node with no outgoing edges
    - Erase any edges coming into it
    - Repeat until no such node

- **Intuition:** Node with no outgoing edges is not waiting on any resource
  - It will eventually finish and release its resources
  - Processes waiting for **those** resources will be able to acquire them and will no longer be waiting!

Erase all edges ⇔ Graph has no cycles
Edges remain ⇔ Deadlock
RAG Reduction

Deadlock?
NO! (no cycles)
Step 1: Satisfy P₃'s requests
Step 2: Satisfy P₂'s requests
Step 3: Satisfy P₁'s requests
Schedule [P₃ P₂ P₁] completely eliminates edges!
RAG Reduction

Deadlock?

NO! (no cycles)
Step 1: Satisfy $P_3$’s requests
Step 2: Satisfy $P_2$’s requests
Step 3: Satisfy $P_1$’s requests
Schedule $[P_3, P_2, P_1]$ completely eliminates edges!

Deadlock?

Yes!
RAG has a cycle
Every node has some outgoing edge
Cannot satisfy any of $P_1$, $P_2$, $P_3$ requests!

172
RAG Reduction

Deadlock?
NO!  (no cycles)
Step 1: Satisfy P_3's requests
Step 2: Satisfy P_2's requests
Step 3: Satisfy P_1's requests
Schedule [P_3 P_2 P_1] completely eliminates edges!

Deadlock?
Yes!
RAG has a cycle
Every node has some outgoing edge
Cannot satisfy any of P_1, P_2, P_3 requests!

Deadlock?
NO!
RAG has a cycle
Schedule [P_2 P_1 P_3 P_4] completely eliminates edges!
More Musings on Deadlock

- Does the order of RAG reduction matter?
  - No. If \( P_i \) and \( P_j \) can both be reduced, reducing \( P_i \) does not affect the reducibility of \( P_j \).

- Does a deadlock disappear on its own?
  - No. Unless a process is killed or forced to release a resource, we are stuck!

- If a system is not deadlocked at time \( T \), is it guaranteed to be deadlock-free at \( T+1 \)?
  - No. Just by requesting a resource (never mind being granted one) a process can create a circular wait!