Monitors (Continued)

Scheduling

Core Concepts and Classic Policies

Professor Natacha Crooks
https://cs162.org/

Slides based on prior slide decks from David Culler, Ion Stoica, John Kubiatowicz, Alison Norman and Lorenzo Alvisi
Recall: Monitors are better!

Use *locks* for mutual exclusion and *condition variables* for scheduling constraints.

**Monitor:** a *lock* and zero or more *condition variables* for managing concurrent access to shared data.

A monitor is a paradigm for concurrent programming
- Some languages like *Java* provide this natively
- Most others use actual locks and condition variables
Recall: Wait & Signal Pattern

... acquire(&buf_lock)
... cond_signal(&buf.CV);
... release(&buf_lock));

acquire(&buf_lock);
...
while (isEmpty(&queue)) {
    cond_wait(&buf.CV,&buf_lock);
}
...
lock.Release();
Hoare Semantics

Thread A

...  
acquire(&buf_lock)
...
cond_signal(&buf_CV);
...
release(&buf_lock));

Thread B

acquire(&buf_lock);
...
if (isEmpty(&queue)) {
    cond_wait(&buf_CV,&buf_lock);
}
...
lock.Release();

1. When call signal, handover buf_lock to thread B.
2. Thread B gets immediately scheduled (nothing can run in between).
3. Thread B eventually releases lock.
Thread A

... 
acquire(&buf_lock)
... 
cond_signal(&buf_CV);
... 
release(&buf_lock));

Thread B

acquire(&buf_lock);
...
while (isEmpty(&queue)) {
    cond_wait(&buf_CV,&buf_lock);
}
...
lock.Release();

1. When call signal, keep lock. Place Thread B on READY queue (no special priority)
2. Thread A eventually releases buf_lock.
3. Thread B eventually gets scheduled and acquires buf_lock.
   Thread C may have run in between.
4. Thread B eventually releases buf_lock.
Basic Structure of *Mesa* Monitor Program

Monitors represent the synchronization logic of the program

- Wait if necessary
- Signal when change something so any waiting threads can proceed

```java
lock
while (need to wait) {
    condvar.wait();
}
unlock

do something so no need to wait

lock
condvar.signal();
unlock
```

Check and/or update state variables
Wait if necessary

Check and/or update state variables
Motivation: Consider a shared database
  - Two classes of users:
    » Readers – never modify database
    » Writers – read and modify database
  - Is using a single lock on the whole database sufficient?
    » Like to have many readers at the same time
    » Only one writer at a time
Basic Readers/Writers Solution

Correctness Constraints:
- Readers can access database when no writers
- Writers can access database when no readers or writers
- Only one thread manipulates state variables at a time

Basic structure of a solution:

- Reader()
  Wait until no writers
  Access database
  Check out – wake up a waiting writer

- Writer()
  Wait until no active readers or writers
  Access database
  Check out – wake up waiting readers or writer
State variables (Protected by a lock called “lock”):
  » int \( AR \): Number of active readers; initially \( = 0 \)
  » int \( WR \): Number of waiting readers; initially \( = 0 \)
  » int \( AW \): Number of active writers; initially \( = 0 \)
  » int \( WW \): Number of waiting writers; initially \( = 0 \)
  » Condition \( okToRead = NIL \)
  » Condition \( okToWrite = NIL \)
Reader() {
    // First check self into system
    acquire(&lock);

    while ((AW + WW) > 0) { // Is it safe to read?
        WR++;                // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--;                 // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);

    // Perform actual read-only access
    AccessDatabase(ReadOnly);

    // Now, check out of system
    acquire(&lock);
    AR--;  // No longer active
    if (AR == 0 && WW > 0) // No other active readers
        cond_signal(&okToWrite);// Wake up one writer
    release(&lock);
}
Writer() {
    // First check self into system
    acquire(&lock);

    while ((AW + AR) > 0) { // Is it safe to write?
        WW++;            // No. Active users exist
        cond_wait(&okToWrite,&lock); // Sleep on cond var
        WW--;            // No longer waiting
    }

    AW++;            // Now we are active!
    release(&lock);

    // Perform actual read/write access
    AccessDatabase(ReadWrite);

    // Now, check out of system
    acquire(&lock);
    AW--;            // No longer active
    if (WW > 0) {    // Give priority to writers
        cond_signal(&okToWrite); // Wake up one writer
    } else if (WR > 0) { // Otherwise, wake reader
        cond_broadcast(&okToRead); // Wake all readers
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

Use an example to simulate the solution

Consider the following sequence of operators:
- R1, R2, W1, R3

Initially: AR = 0, WR = 0, AW = 0, WW = 0
Simulation of Readers/Writers Solution

R1 comes along (no waiting threads)
AR = 0, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock)
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++;
        // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--;        // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--;
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
R1 comes along (no waiting threads)
AR = 0, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R1 comes along (no waiting threads)
AR = 1, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R1 comes along (no waiting threads)
AR = 1, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);
}

AccessDBase(ReadOnly);

acquire(&lock);
AR--; // No waiting threads
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
Simulation of Readers/Writers Solution

R1 accessing dbase (no other threads)
AR = 1, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);
}

AccessDBase(ReadOnly)

acquire(&lock);
AR--; // No. Readers exist
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
Simulation of Readers/Writers Solution

R2 comes along (R1 accessing dbase)
AR = 1, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--;  // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);
    AccessDBase(ReadOnly);
    acquire(&lock);
    AR--;;
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R2 comes along (R1 accessing dbase)
AR = 1, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R2 comes along (R1 accessing dbase)

\[ AR = 2, \ WR = 0, \ AW = 0, \ WW = 0 \]

```c
Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);
    AccessDBase(ReadOnly);
    acquire(&lock);
    AR--;  
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
```
Simulation of Readers/Writers Solution

R2 comes along (R1 accessing dbase)
AR = 2, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);
}

acquire(&lock);
AR--;
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
Simulation of Readers/Writers Solution

R1 and R2 accessing database
AR = 2, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);
    acquire(&lock);
    AR--;
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}

Assume readers take a while to access database
Situation: Locks released, only AR is non-zero
Simulation of Readers/Writers Solution

W1 comes along (R1 and R2 are still accessing dbase)

AR = 2, WR = 0, AW = 0, WW = 0

Writer()

call acquire(&lock);

while ((AW + AR) > 0) { // Is it safe to write?
    WW++; // No. Active users exist
    cond_wait(&okToWrite,&lock); // Sleep on cond var
    WW--; // No longer waiting
}

AW++;
release(&lock);

AccessDBase(ReadWrite);

acquire(&lock);
AW--;
if (WW > 0){
    cond_signal(&okToWrite);
} else if (WR > 0) {
    cond_broadcast(&okToRead);
}
release(&lock);
Simulation of Readers/Writers Solution

W1 comes along (R1 and R2 are still accessing dbase)
AR = 2, WR = 0, AW = 0, WW = 0

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) {
        // Is it safe to write?
        WW++;  // No. Active users exist
        cond_wait(&okToWrite,&lock); // Sleep on cond var
        WW--;  // No longer waiting
    }
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    if (WW > 0) {
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

- W1 comes along (R1 and R2 are still accessing dbase)
- AR = 2, WR = 0, AW = 0, WW = 1

```c
Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) {  // Is it safe to write?
        WW++;                // No. Active users exist
        cond_wait(&okToWrite,&lock);// Sleep on cond var
        WW--;                // No longer waiting
    }
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);
}
```

```c
acquire(&lock);
AW--;
if (WW > 0){
    cond_signal(&okToWrite);
} else if (WR > 0) {
    cond_broadcast(&okToRead);
}
release(&lock);
```
Simulation of Readers/Writers Solution

R3 comes along (R1 and R2 accessing dbase, W1 waiting)

\[ AR = 2, \ WR = 0, \ AW = 0, \ WW = 1 \]

\[
\text{Reader()} \{ \\
\quad \text{acquire}(&\text{lock}); \\
\quad \text{while} \ ((\text{AW} + \text{WW}) > 0) \{ \ // \text{Is it safe to read?} \\
\quad \quad \text{WR}++; \ // \text{No. Writers exist} \\
\quad \quad \text{cond\_wait}(&\text{okToRead},&\text{lock}); // \text{Sleep on cond var} \\
\quad \quad \text{WR}--; \ // \text{No longer waiting} \\
\quad \} \\
\quad \text{AR}++; \ // \text{Now we are active!} \\
\quad \text{release}(&\text{lock}); \\
\}
\]

\[
\text{AccessDBase(ReadOnly);} \\
\]

\[
\text{acquire}(&\text{lock}); \\
\quad \text{AR}--; \\
\quad \text{if} \ (\text{AR} == 0 \ \&\& \ \text{WW} > 0) \\
\quad \quad \text{cond\_signal}(&\text{okToWrite}); \\
\quad \text{release}(&\text{lock}); \\
\}
\]
**Simulation of Readers/Writers Solution**

R3 comes along (R1 and R2 accessing dbase, W1 waiting)

AR = 2, WR = 0, AW = 0, WW = 1

```c
Reader() {
    acquire(&lock);
    while ((AR + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);
    AccessDBase(ReadOnly);
    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
```
Simulation of Readers/Writers Solution

R3 comes along (R1 and R2 accessing database, W1 waiting)

AR = 2, WR = 1, AW = 0, WW = 1

```c
Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    lock.release();

    AccessDBase(ReadOnly);
}
```

acquire(&lock);
AR--;
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
Simulation of Readers/Writers Solution

R3 comes along (R1, R2 accessing dbase, W1 waiting)
AR = 2, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--;
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R1 and R2 accessing dbase, W1 and R3 waiting
AR = 2, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++;                // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--;                // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--;
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}

Status:
• R1 and R2 still reading
• W1 and R3 waiting on okToWrite and okToRead, respectively
Simulation of Readers/Writers Solution

R2 finishes (R1 accessing dbase, W1 and R3 waiting)
AR = 2, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R2 finishes (R1 accessing dbase, W1 and R3 waiting)
AR = 1, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);
    AccessDBase(ReadOnly);
    acquire(&lock);
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
R2 finishes (R1 accessing dbase, W1 and R3 waiting)

\[
\begin{align*}
AR &= 1, \\
WR &= 1, \\
AW &= 0, \\
WW &= 1
\end{align*}
\]

```
Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++;  // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--;  // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);
    AccessDBase(ReadOnly);
    acquire(&lock);
    AR--;  
    if (AR == 0 && WW > 0) {
        cond_signal(&okToWrite);
        release(&lock);
    }
}
```
Simulation of Readers/Writers Solution

R2 finishes (R1 accessing dbase, W1 and R3 waiting)

AR = 1, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--;    
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R1 finishes (W1 and R3 waiting)

AR = 1, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);
}

AccessDBase(ReadOnly);

acquire(&lock);
AR--; // Now we are active!
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
Simulation of Readers/Writers Solution

R1 finishes (W1, R3 waiting)
AR = 0, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++;
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Not shown
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R1 finishes (W1, R3 waiting)
AR = 0, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R1 signals a writer (W1 and R3 waiting)
AR = 0, WR = 1, AW = 0, WW = 1

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock); // Sleep on cond var
        WR--;
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);
}

acquire(&lock);
AR--;
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
Simulation of Readers/Writers Solution

W1 gets signal (R3 still waiting)
AR = 0, WR = 1, AW = 0, WW = 1

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) {  // Is it safe to write?
        WW++;  // No. Active users exist
        cond_wait(&okToWrite,&lock);// Sleep on cond var
        WW--;  // No longer waiting
    }
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    AW--;
    if (WW > 0){
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

W1 gets signal (R3 still waiting)
AR = 0, WR = 1, AW = 0, WW = 0

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++;
        if (WW > 0) {
            cond_signal(&okToWrite, &lock); // Sleep on cond var
        }
        WW--;
    }
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    AW--;
    if (WW > 0) {
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

W1 gets signal (R3 still waiting)
AR = 0, WR = 1, AW = 1, WW = 0

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) { // Is it safe to write?
        WW++; // No. Active users exist
        cond_wait(&okToWrite,&lock); // Sleep on cond var
        WW--; // No longer waiting
    }
    AW++; // No longer waiting
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    AW--;
    if (WW > 0) {
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

W1 accessing dbase (R3 still waiting)
AR = 0, WR = 1, AW = 1, WW = 0

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) {
        // Is it safe to write?
        WW++;
        // No. Active users exist
        cond_wait(&okToWrite,&lock); // Sleep on cond var
        WW--;
        // No longer waiting
    }
    AW++;
    release(&lock);
}

AccessDBase(ReadWrite);

acquire(&lock);
AW--; if (WW > 0) {
    cond_signal(&okToWrite);
} else if (WR > 0) {
    cond_broadcast(&okToRead);
} release(&lock);
Simulation of Readers/Writers Solution

W1 finishes (R3 still waiting)
AR = 0, WR = 1, AW = 1, WW = 0

Writer() {
  acquire(&lock);
  while ((AW + AR) > 0) {
    // Is it safe to write?
    WW++;
    // No. Active users exist
    cond_wait(&okToWrite,&lock); // Sleep on cond var
    WW--;
    // No longer waiting
  }
  AW++;
  release(&lock);

  AccessDBase(ReadWrite);
}

acquire(&lock);
AW--
if (WW > 0){
  cond_signal(&okToWrite);
} else if (WR > 0) {
  cond_broadcast(&okToRead);
}
release(&lock);
Simulation of Readers/Writers Solution

W1 finishes (R3 still waiting)
AR = 0, WR = 1, AW = 0, WW = 0

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) {
        // Is it safe to write?
        WW++;
        // No. Active users exist
        cond_wait(&okToWrite,&lock); // Sleep on cond var
        WW--;
    } // No longer waiting
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    AW--;
    if (WW > 0) {
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

W1 finishes (R3 still waiting)

AR = 0, WR = 1, AW = 0, WW = 0

Writer() {
    acquire(&lock);
    while ((AW + AR) > 0) {  // Is it safe to write?
        WW++;  // No. Active users exist
        cond_wait(&okToWrite,&lock);// Sleep on cond var
        WW--;  // No longer waiting
    }
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    AW--;
    if (WW > 0){
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
Simulation of Readers/Writers Solution

$W_1$ signaling readers (R3 still waiting)
AR = 0, WR = 1, AW = 0, WW = 0

```c
Writer() {
    acquire(&lock);
    while (((AW + AR) > 0) {  // Is it safe to write?
        WW++;              // No. Active users exist
        cond_wait(&okToWrite,&lock); // Sleep on cond var
        WW--;              // No longer waiting
    }
    AW++;
    release(&lock);

    AccessDBase(ReadWrite);

    acquire(&lock);
    AW--;  
    if (WW > 0){
        cond_signal(&okToWrite);
    } else if (WR > 0) {
        cond_broadcast(&okToRead);
    }
    release(&lock);
}
```
Simulation of Readers/Writers Solution

R3 gets signal (no waiting threads)
AR = 0, WR = 1, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R3 gets signal (no waiting threads)
AR = 0, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDBase(ReadOnly);

    acquire(&lock);
    AR--; // No readers
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

**R3 accessing dbase (no waiting threads)**

**AR = 1, WR = 0, AW = 0, WW = 0**

```c
Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++;  // Now we are active!
    release(&lock);
}

AccessDBase(ReadOnly),

acquire(&lock);
AR--;
if (AR == 0 && WW > 0)
    cond_signal(&okToWrite);
release(&lock);
```
Simulation of Readers/Writers Solution

R3 finishes (no waiting threads)
AR = 1, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);
    AccessDBase(ReadOnly);
    acquire(&lock);
    AR--;  
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Simulation of Readers/Writers Solution

R3 finishes (no waiting threads)
AR = 0, WR = 0, AW = 0, WW = 0

Reader() {
    acquire(&lock);
    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }
    AR++; // Now we are active!
    release(&lock);

    AccessDbase(ReadOnly);
    acquire(&lock);
    AR--; // Now we are active!
    if (AR == 0 && WW > 0)
        cond_signal(&okToWrite);
    release(&lock);
}
Questions

Can readers starve? Consider Reader() entry code:

```c
while ((AW + WW) > 0) { // Is it safe to read?
    WR++; // No. Writers exist
    cond_wait(&okToRead,&lock); // Sleep on cond var
    WR--; // No longer waiting
}
AR++; // Now we are active!
```

What if we erase the condition check in Reader exit?

```c
AR--; // No longer active
if (AR == 0 && WW > 0) // No other active readers
    cond_signal(&okToWrite); // Wake up one writer
```
Further, what if we turn the `signal()` into `broadcast()`

```c
AR--; // No longer active
cond_broadcast(&okToWrite); // Wake up sleepers
```

Finally, what if we use only one condition variable (call it “okContinue”) instead of two separate ones?
- Both readers and writers sleep on this variable
- Must use `broadcast()` instead of `signal()`
Reader() {
    // First check self into system
    acquire(&lock);

    while ((AW + WW) > 0) { // Is it safe to read?
        WR++; // No. Writers exist
        cond_wait(&okToRead,&lock);// Sleep on cond var
        WR--; // No longer waiting
    }

    AR++; // Now we are active!
    release(&lock);

    // Perform actual read-only access
    AccessDatabase(ReadOnly);

    // Now, check out of system
    acquire(&lock);
    AR--; // No longer active
    if (AR == 0 && WW > 0) // No other active readers
        cond_signal(&okToWrite);// Wake up one writer
    release(&lock);
}
Writer() {
   // First check self into system
   acquire(&lock);

   while ((AW + AR) > 0) { // Is it safe to write?
      WW++; // No. Active users exist
      cond_wait(&okToWrite,&lock); // Sleep on cond var
      WW--; // No longer waiting
   }

   AW++; // Now we are active!
   release(&lock);

   // Perform actual read/write access
   AccessDatabase(ReadWrite);

   // Now, check out of system
   acquire(&lock);
   AW--; // No longer active
   if (WW > 0){ // Give priority to writers
      cond_signal(&okToWrite); // Wake up one writer
   } else { // Otherwise, wake reader
      cond_broadcast(&okToRead); // Wake all readers
   }
   release(&lock);
}
Monitors represent the synchronization logic of the program:
- Wait if necessary
- Signal when change something so any waiting threads can proceed

```java
lock
while (need to wait) {
    condvar.wait();
}
unlock

do something so no need to wait
lock
condvar.signal();
unlock
```

Check and/or update state variables
Wait if necessary

Check and/or update state variables
C-Language Support for Synchronization

C language: Pretty straightforward synchronization

Just make sure you know all the code paths out of a critical section

```c
int Rtn() {
    acquire(&lock);
    ...
    if (exception) {
        release(&lock);
        return errReturnCode;
    }
    ...
    release(&lock);
    return OK;
}
```
Concurrency and Synchronization in C

Harder with more locks

```c
void Rtn() {
    lock1.acquire();
    ...
    if (error) {
        lock1.release();
        return;
    }
    ...
    lock2.acquire();
    ...
    if (error) {
        lock2.release();
        lock1.release();
        return;
    }
    ...
    lock2.release();
    lock1.release();
}
```
C++ Language Support for Synchronization

Languages with exceptions like C++
– Languages that support exceptions are problematic (easy to make a non-local exit without releasing lock)

```cpp
void Rtn()
{
    lock.acquire();
    ...
    DoFoo();
    ...
    lock.release();
}
void DoFoo()
{
    ...
    if (exception) throw errException;
    ...
}

– Notice that an exception in DoFoo() will exit without releasing the lock!
```
Must catch all exceptions in critical sections
– Catch exceptions, release lock, and re-throw exception:

```cpp
void Rtn() {
    lock.acquire();
    try {
        ...
        DoFoo();
        ...
    } catch (...) { // catch exception
        lock.release(); // release lock
        throw; // re-throw the exception
    }
    lock.release();
}
void DoFoo() {
    ...
    if (exception) throw errException;
    ...
}
```
# Much better: C++ Lock Guards

```cpp
#include <mutex>

int global_i = 0;
std::mutex global_mutex;

void safe_increment() {
    std::lock_guard<std::mutex> lock(global_mutex);
    ...
    global_i++;
    // Mutex released when ‘lock’ goes out of scope
}
```
More versatile than we show here (can be used to close files, database connections, etc.)

```
lock = threading.Lock()
...
with lock:  # Automatically calls acquire()
    some_var += 1
...
# release() called however we leave block
```
Java synchronized Keyword

Every Java object has an associated lock:
- Lock is acquired on entry and released on exit from a synchronized method
- Lock is properly released if exception occurs inside a synchronized method
- Mutex execution of synchronized methods (beware deadlock)

```java
class Account {
    private int balance;

    // object constructor
    public Account (int initialBalance) {
        balance = initialBalance;
    }
    public synchronized int getBalance() {
        return balance;
    }
    public synchronized void deposit(int amount) {
        balance += amount;
    }
}
```
Java Support for Monitors

Along with a lock, every object has a single condition variable associated with it.

To wait inside a synchronized method:
- `void wait();`
- `void wait(long timeout);`

To signal while in a synchronized method:
- `void notify();`
- `void notifyAll();`
**Where are we going with synchronization?**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Load/Store</th>
<th>Disable Ints</th>
<th>Test&amp;Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programs</td>
<td>Shared Programs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Higher-level API</td>
<td>Locks</td>
<td>Semaphores</td>
<td>Monitors</td>
</tr>
</tbody>
</table>

Implement various higher-level synchronization primitives using atomic operations
Topic Breakdown

- Process Abstraction and API
- Threads and Concurrency
- Scheduling
- Virtual Memory
- Paging
- IO devices
- File Systems
- Challenges with distribution
- Data Processing & Storage

Virtualizing the CPU
Virtualizing Memory
Persistence
Distributed Systems
Goals for Today

- What is scheduling?

- What makes a good scheduling policy?

- What are existing schedulers and how do they perform?
The Scheduling Loop!

if (readyThreads(TCBs)) {
    nextTCB = selectThread(TCBs);
    run(nextTCB);
} else {
    run_idle_thread();
}

1. Which task to run next?

2. How frequently does this loop run?

3. What happens if run never returns?
Recall: Thread Life Cycle

Dying → Running

Scheduled

Running → Ready

Ready → Scheduled

Scheduled → Blocked

Blocked → Request I/O

Request I/O → Finish I/O

Finish I/O → Dying
Recall: What triggers a scheduling decision?

- IO Occurs
- Interrupt Occurs
- Fork a child / Yield
- Time Slice Expired
- IO Request
- Wait for an interrupt
What makes a good scheduling policy?

A hopeless Queue.
The Queue For the UK Queen
6 miles (10 KM) long.
Visible from Space.

A bad but more realistic queue.
The DMV
What makes a good scheduling policy?

What does the DMV care about?

What do individual users care about?
Important Performance Metrics

Response time (or latency).
User-perceived time to do some task

Throughput.
The rate at which tasks are completed

Scheduling overhead.
The time to switch from one task to another.

Predictability.
Variance in response times for repeated requests.
Important Performance Metrics

Fairness
Equality in the performance perceived by one task

Starvation
The lack of progress for one task, due to resources being allocated to different tasks
Sample Scheduling Policies

Assume DMV job A takes 1 second, job B takes 2 days

Policy Idea: Only ever schedule users with Job A

What is the metric we are optimizing?
A) Throughput  B) Latency  C) Predictability  D) Low-Overhead

Can the schedule lead to starvation?
A) Yes  B) No

Is the schedule fair?
A) Yes  B) No
Sample Scheduling Policies

Assume DMV consists only of jobs of type A.

Policy Idea: Schedule jobs randomly

What is the metric we are optimizing?
A) Throughput  B) Latency  C) Predictability  D) Low-Overhead

Can the schedule lead to starvation?
A) Yes  B) No

Is the schedule fair?
A) Yes  B) No
Sample Scheduling Policies

Assume DMV consists only of 100 different types of jobs. Some jobs need Clerk A, some Clerks A&B, others Clerk C.

Policy Idea: Every time schedule a job, compute all possible orderings of jobs, pick one that finishes quickest.

What is the metric we are optimizing?  
A) Throughput  B) Latency  C) Predictability  D) Low-Overhead

Can the schedule lead to starvation?  
A) Yes  B) No

Is the schedule fair?  
A) Yes  B) No
Scheduling Policy Goals/Criteria

Minimise Response Time

Maximise Throughput

While remaining fair and starvation-free
Useful metrics

Waiting time for P
Total Time spent waiting for CPU
Average waiting time
Average of all processes’ wait time

Response Time for P
Time to when process gets first scheduled

Completion time
Waiting time + Run time

Average completion time
Average of all processes’ completion time
Assumptions

Threads are independent!

One thread = One User

Unrealistic but simplify the problem so it can be solved

Only look at work-conserving scheduler

=> Never leave processor idle if work to do
Workload Assumptions

A workload is a set of tasks for some system to perform, including how long tasks last and when they arrive.

**Compute-Bound**
- Tasks that primarily perform compute
- Fully utilise CPU

**IO Bound**
- Mostly wait for IO, limited compute
- Often in the Blocked state
First-Come, First-Served (FCFS)

Run tasks in order of arrival.

Run task until completion (or blocks on IO).
No preemption

This is the DMV model.

Also called FIFO
**First-Come, First-Served (FCFS)**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>24</td>
</tr>
</tbody>
</table>

What is the average completion time? \( \frac{3+6+30}{3} = 13 \)

What is the average waiting time? \( \frac{0+3+6}{3} = 3 \)
First-Come, First-Served (FCFS)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_3$</td>
<td>24</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_1$</td>
<td>3</td>
</tr>
</tbody>
</table>

What is the average completion time? \( \left( \frac{24+27+30}{3} \right) = 27 \) 

What is the average waiting time? \( \left( \frac{0+24+27}{3} \right) = 17 \)
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

Convoy effect
Short process stuck behind long process
Lots of small tasks build up behind long tasks
FIFO is non-preemptible
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

Convoy effect
Short process stuck behind long process
Lots of small tasks build up behind long tasks
FIFO is non-preemptible
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

*Convoy effect*

Short process stuck behind long process
Lots of small tasks build up behind long tasks
FIFO is non-preemptible
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

Convoy effect
Short process stuck behind long process
Lots of small tasks build up behind long tasks
FIFO is non-preemptible
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

_convoy effect_

- Short process stuck behind long process
- Lots of small tasks build up behind long tasks
- FIFO is non-preemptible
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

Convoy effect
Short process stuck behind long process
Lots of small tasks build up behind long tasks
FIFO is non-preemptible
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

Convoy effect
- Short process stuck behind long process
- Lots of small tasks build up behind long tasks
- FIFO is non-preemptible

CPU

P3

P4

P5
The Convoy Effect

FIFO/FCFS very sensitive to arrival order

Convoy effect

Short process stuck behind long process
Lots of small tasks build up behind long tasks
FIFO is non-preemptible

CPU

P3

P4 P5 P6
FIFO/FCFS very sensitive to arrival order

**Convoy effect**

*Short process stuck behind long process*
*Lots of small tasks build up behind long tasks*
*FIFO is non-preemptible*

Can FIFO lead to starvation?
FCFS/FIFO Summary

The good
- Simple
- Low Overhead
- No Starvation

The bad
- Sensitive to arrival order (poor predictability)

The ugly
- Convoy Effect.
- Bad for Interactive Tasks
Shortest Job First

How can we minimise average completion time?

By scheduling jobs in order of estimated completion time

This is the “10 items or less” line at Safeway
**Shortest Job First**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
</tr>
<tr>
<td>$P_2$</td>
<td>6</td>
</tr>
<tr>
<td>$P_3$</td>
<td>24</td>
</tr>
<tr>
<td>$P_4$</td>
<td>1</td>
</tr>
</tbody>
</table>

What is the average completion time? \( \frac{1+4+10+34}{4} = 12.25 \)

Can prove that SJF generates optimal average completion time if all jobs arrive at the same time.
Are we done?

Can SJF lead to starvation?

Yes

Any scheduling policy that always favours a fixed property for scheduling leads to starvation.
Are we done?

Can SJF lead to starvation?

Yes

Any scheduling policy that always favours a fixed property for scheduling leads to starvation.
Can SJF lead to starvation?

Yes

Any scheduling policy that always favours a fixed property for scheduling leads to starvation.
Are we done?

Is SFJ subject to the convoy effect?

Yes

Any non-preemptible scheduling policy suffers from convoy effect
Are we done?

Is SFJ subject to the convoy effect?

Yes

Any non-preemptible scheduling policy suffers from convoy effect.
SJF Summary

The good

Optimal Average Completion Time when jobs arrive simultaneously

The bad

Sensitive to arrival order (poor predictability)

The ugly

Can lead to starvation!

Requires knowing duration of job
Shortest Time to Completion First (STCF)

Introduce the notion of preemption

A running task can be de-scheduled before completion.

STCF

Schedule the task with the least amount of time left
**Shortest Time to Completion First (STCF)**

STCF

Schedule the task with the least amount of time left

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>
**Shortest Time to Completion First (STCF)**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>16</td>
<td>18</td>
</tr>
</tbody>
</table>

P3
## Shortest Time to Completion First (STCF)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>16</td>
<td>18</td>
</tr>
</tbody>
</table>

![Diagram of process execution]

- **P3** finishes after **P2**
# Shortest Time to Completion First (STCF)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

**Diagram:**

```
0  1  7  10
P3  P2  P3
```
# Shortest Time to Completion First (STCF)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>16</td>
<td>18</td>
</tr>
</tbody>
</table>

![Diagram of process times]
**Shortest Time to Completion First (STCF)**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>P₂</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P₃</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>P₄</td>
<td>16</td>
<td>18</td>
</tr>
</tbody>
</table>

![Process Timeline Diagram]

0  1  7  10  13  18
## Shortest Time to Completion First (STCF)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>15</td>
<td>18</td>
</tr>
</tbody>
</table>

![Process timing diagram](image)
# Shortest Time to Completion First (STCF)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (left)</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>15</td>
<td>18</td>
</tr>
</tbody>
</table>
Are we done?

Can STCF lead to starvation?

Yes

Any scheduling policy that always favours a fixed property for scheduling leads starvation

No change!
Are we done?

Is STCF subject to the convoy effect?

No!

STCF is a *preemptible* policy
STCF Summary

The good

Optimal Average Completion Time Always

The bad

The ugly

Can lead to starvation!

Requires knowing duration of job
Taking a step back

<table>
<thead>
<tr>
<th>Property</th>
<th>FCFS</th>
<th>SJF</th>
<th>STCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimise Average Completion Time</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Prevent Starvation</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prevent Convoy Effect</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Psychic Skills Not Needed</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can we design a non-psychic, starvation-free scheduler with good response time?
**Round-Robin Scheduling**

RR runs a job for a **time slice** (a scheduling quantum)

Once time slice over, Switch to next job in ready queue. => Called **time-slicing**
<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>53</td>
</tr>
<tr>
<td>$P_2$</td>
<td>8</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

RR with Time Quantum = 20
RR with Time Quantum $= 20$

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$53 \Rightarrow 33$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>8</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

![Diagram](chart.png)

$P_1$ 
0 20
RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>33</td>
</tr>
<tr>
<td>$P_2$</td>
<td>8 =&gt; 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

- $P_1$: 0-33
- $P_2$: 28-28

Time quantum: 20
### RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>33</td>
</tr>
<tr>
<td>P₂</td>
<td>0</td>
</tr>
<tr>
<td>P₃</td>
<td>68 =&gt; 48</td>
</tr>
<tr>
<td>P₄</td>
<td>24</td>
</tr>
</tbody>
</table>

![Diagram showing the execution of processes with Burst Times]
## RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>33</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>0</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>48</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>24 =&gt; 4</td>
</tr>
</tbody>
</table>

**Diagram:**

```
0  20  28  48  68
P1  P2  P3  P4
```
**RR with Time Quantum = 20**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>33 =&gt; 13</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>0</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>48</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>4</td>
</tr>
</tbody>
</table>

**Diagram:**

- \( P_1 \) runs from 0 to 20
- \( P_2 \) runs from 20 to 28
- \( P_3 \) runs from 28 to 48
- \( P_4 \) runs from 48 to 68
- \( P_1 \) runs from 68 to 88
### RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>13</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>48 =&gt; 28</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
</tr>
</tbody>
</table>

The diagram illustrates the execution of processes with the time quantum set to 20. The sequence of execution is as follows:

- $P_1$ starts at 0 and runs for 13 units of time.
- $P_2$ starts at 20 and runs for 0 units of time.
- $P_3$ starts at 28 and runs for 48 units of time, then is rescheduled with a burst time of 28 units.
- $P_4$ starts at 48 and runs for 68 units of time.
- $P_1$ resumes execution starting at 88 and runs for 88 units of time.
- $P_3$ resumes execution starting at 108 and runs for 4 units of time.
### RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>13</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>28</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4 =&gt; 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_1$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>$P_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RR with Time Quantum = 20

Waiting time

- \( P_1 = 0 + (68-20)+(112-88)=72 \)
- \( P_2 = (20-0)=20 \)
- \( P_3 = (28-0)+(88-48)+(125-108)+0=85 \)
- \( P_4 = (48-0)+(108-68)=88 \)

Average waiting time

\[ \frac{72+20+85+88}{4} = 66.25 \]

Average completion time

\[ \frac{125+28+153+112}{4} = 104.25 \]
Decrease Completion Time

- $T_1$: Burst Length 10
- $T_2$: Burst Length 5
- $T_3$: Burst Length 10

For $Q = 10$:
Average Completion Time $= \frac{10 + 15 + 25}{3} = 16.7$

For $Q = 5$:
Average Completion Time $= \frac{20 + 10 + 25}{3} = 18.3$
Switching is not free!

Small scheduling quantas lead to frequent context switches
- Mode switch overhead
- Trash cache-state

$q$ must be large with respect to context switch, otherwise overhead is too high.
Are we done?

Can RR lead to starvation?

No

No process waits more than \((n-1)q\) time units
Are we done?

Can RR suffer from convoy effect?

No

Only run a time-slice at a time
RR Summary

The good

Bounded response time

The bad

Completion time can be high (stretches out long jobs)

The ugly

Overhead of context switching
# Taking a step back

<table>
<thead>
<tr>
<th>Property</th>
<th>FCFS</th>
<th>SJF</th>
<th>STCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimise Average Completion Time</td>
<td></td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>Prevent Starvation</td>
<td>✅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prevent Convoy Effect</td>
<td></td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>Psychic Skills Not Needed</td>
<td>✅</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Taking a step back

<table>
<thead>
<tr>
<th>Property</th>
<th>FCFS</th>
<th>SJF</th>
<th>STCF</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimise Average Completion Time</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Optimise Average Response Time</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Prevent Starvation</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Prevent Convoy Effect</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Psychic Skills Not Needed</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Assuming zero-cost context-switching time, is RR always better than FCFS?

10 jobs, each take 100s of CPU time
RR scheduler quantum of 1s
All jobs start at the same time

<table>
<thead>
<tr>
<th>Job #</th>
<th>FIFO</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>991</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>992</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>900</td>
<td>999</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>
## Earlier Example with Different Time Quantum

### Best FCFS:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>85</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>85</td>
<td>85</td>
<td>153</td>
<td>153</td>
<td>153</td>
</tr>
<tr>
<td>153</td>
<td>153</td>
<td>153</td>
<td>153</td>
<td>153</td>
</tr>
</tbody>
</table>

### Quantum

<table>
<thead>
<tr>
<th>Quantum</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best FCFS</td>
<td>85</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>69.5</td>
</tr>
<tr>
<td>Q=1</td>
<td>137</td>
<td>30</td>
<td>153</td>
<td>81</td>
<td>100.5</td>
</tr>
<tr>
<td>Q=5</td>
<td>135</td>
<td>28</td>
<td>153</td>
<td>82</td>
<td>99.5</td>
</tr>
<tr>
<td>Q=8</td>
<td>133</td>
<td>16</td>
<td>153</td>
<td>80</td>
<td>99.5</td>
</tr>
<tr>
<td>Q=10</td>
<td>135</td>
<td>18</td>
<td>153</td>
<td>92</td>
<td>104.5</td>
</tr>
<tr>
<td>Q=20</td>
<td>125</td>
<td>28</td>
<td>153</td>
<td>112</td>
<td>104.5</td>
</tr>
<tr>
<td>Worst FCFS</td>
<td>121</td>
<td>153</td>
<td>68</td>
<td>145</td>
<td>121.75</td>
</tr>
</tbody>
</table>

The table above shows the execution times for different quantum sizes, comparing to the best FCFS results.