Announcements

Midterm Exam

• Due tomorrow, Tuesday, 12 July, at 12:59pm – turn it in at the start of lecture

Monday, 18 July

• No lecture

HW2

• Will be made available Wednesday
• Similar in structure to HW1; will include book “questions” and a programming task.
17. Assume the following three threads with two instructions each:

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
<th>Thread C</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA1 : x = x+1</td>
<td>IB1 : x = x-2</td>
<td>IC1 : x = x+2</td>
</tr>
<tr>
<td>IA2 : print(x)</td>
<td>IB2 : print(x)</td>
<td>IC2 : print(x)</td>
</tr>
</tbody>
</table>

All three threads are **sharing** the integer variable x, and the instruction pipeline on the computer where these threads are **concurrently** executed contains fetch, execute, and write back stages. Each thread fetches the value of x into a private (non-shared) memory space, and the write back stages writes to the shared variable.

Under these conditions, which of the following outputs (output to the screen) is/are possible when all three threads run to completion?

Hint: As a first step, rewrite the instructions for the threads into a register/ALU view (as was shown in lecture). Assume that the print instruction of each thread cannot be decomposed further.

A. 222  
B. 121  
C. 434  
D. 789  
E. 432  
F. 936  
G. 557

Q: What is the initial value of x?  
Q: What is/are the possible initial value/s of x?
string line;
read a line of input from stdin into line;
while (!EOF) {
    co look for pattern in line;
    if (pattern is in line)
        write line;
    // read next line of input into line;
    oc;
}
From last time

string line;
read a line of input from stdin into line;
while (!EOF) {
    look for pattern in line;
    if (pattern is in line)
        write line;
    // read next line of input into line;
    oc;
}

These specify a region of code that is executed concurrently

This is the code enclosed by co and oc, which is being parallelized
From last time

```c
string line;
read a line of input from stdin into line;
while (!EOF) {
    // look for pattern in line;
    if (pattern is in line)
        write line;
    // read next line of input into line;
    oc;
}
```

What does // specify?
This specifies an arm of the program ... think of a program with multiple arms as being able to do several things at once (as many arms as there are available)

There are “two” arms, each of which is executed concurrently

Notice that the “first” arm is a sequence of statements, but the second arm is a single statement

Q: How many arms can a program have?
From last time

```c
string line1, line2;
read a line of input from stdin into line1;
while (! EOF) {
    co look for pattern in line1;
    if (pattern is in line1)
        write line1;
    // read next line of input into line2;
    oc;
    line1 = line2;
}
```

Q: What does this code accomplish?
From last time

string line1, line2;
read a line of input from `stdin` into `line1`;
while (! EOF) {
    co look for pattern in `line1`;
    if (pattern is in `line1`)
        write `line1`;
    // read next line of input into `line2`;
    oc;
    `line1` = `line2`;
}

At the end of each loop iteration, and after each arm has finished, copy the contents of `line2` into `line1`. Be sure that you can “see” what this is doing.

Reading from `stdin`, and saving into `line2`

Looking for the pattern

`CPU_0`

Arm 1

`CPU_1`

Arm 2
From last time

At the end of each loop iteration, and after each arm has finished, copy the contents of line2 into line1. Be sure that you can “see” what this is doing.

```cpp
string line1, line2;
read a line of input from stdin into line1;
while (! EOF) {
    co look for pattern in line1;
    if (pattern is in line1)
        write line1;
    // read next line of input into line2;
    oc;
    line1 = line2;
}
```

Reading from stdin, and saving into line2

Looking for the pattern

CPU_0
Arm 1

CPU_1
Arm 2
Although the above solution is “correct,” does it ensure correct coordination between \texttt{line1} and \texttt{line2}?
From last time

The pseudocode of the concurrent implementation of the “find pattern” code (shown above) is shown left.
From last time

```c
string buffer;  // contains one line of input
bool done = false;  // used to signal termination
co   # process 1: find patterns
    string line1;
    while (true) {
        wait for buffer to be full or done to be true;
        if (done) break;
        line1 = buffer;
        signal that buffer is empty;
        look for pattern in line1;
        if (pattern is in line1)
            write line1;
    }

    // # process 2: read new lines
    string line2;
    while (true) {
        read next line of input into line2;
        if (EOF) {done = true; break; }
        wait for buffer to be empty;
        buffer = line2;
        signal that buffer is full;
    }

    oc;
```

We need to accomplish these tasks ... how? ...
From last time

string buffer;  // contains one line of input
bool done = false;  // used to signal termination

co # process 1: find patterns
    string line1;
    while (true) {
        wait for buffer to be full or done to be true;
        if (done) break;
        line1 = buffer;
        signal that buffer is empty;
        look for pattern in line1;
        if (pattern is in line1)
            write line1;
    }

// # process 2: read new lines
string line2;
while (true) {
    read next line of input into line2;
    if (EOF) {done = true; break; }
    wait for buffer to be empty;
    buffer = line2;
    signal that buffer is full;
}

We need to accomplish these tasks ... how? ...

... which enforces that only one of these critical sections is being executed at any one time
From last time

Atomic action: ______________________________
Atomic action: Makes an **indivisible** state transformation. Any intermediate state that might exist in the implementation of the action must NOT be visible to other processes.
Atomic action: Makes an **indivisible** state transformation. Any intermediate state that might exist in the implementation of the action must NOT be visible to other processes.

```c
int y = 0, z = 0;
co x = y+z; // y = 1; z = 2; oc;
```

When executed concurrently, the assignment statements might be implemented by a sequence of fine-grained machine instructions, and depending on the execution history one of several (potentially many) different results may result. In this case the assignment statement when executed concurrently are NOT atomic actions.

**Q: What are the possible final values of x?**
From last time

**Atomic action**: Makes an **indivisible** state transformation. Any intermediate state that might exist in the implementation of the action must **NOT** be visible to other processes.

```c
int y = 0, z = 0;
const x = y + z; // y = 1; z = 2; oc;
```

Keep in mind that \( x = y + z \) is in itself a series of instructions:

- \( i_1 \): fetch \( y \)
- \( i_2 \): fetch \( z \)
- \( i_3 \): compute \( x = y + z \)
- \( i_4 \): write back \( x \)

Q: Assuming \( i_1 \)–\( i_6 \)

- \( i_5 \): \( y = 1 \)
- \( i_6 \): \( z = 2 \)

**Q**: How might \( x = 2 \) be the final state?

**Q**: What is an instruction history such that \( x \) has a final value of 2?

0, 1, \boxed{2} or 3
From last time

How do we synchronize two or more processes if we cannot guarantee that those actions are atomic?
From last time

How do we synchronize two or more processes if we cannot guarantee that those actions are atomic?

Atomic actions are specified by use of angle brackets, < and >

Atomic actions in pseudocode can be enforced via use of await statements, which can be implemented using while loops.
From last time

- `< await (B) S; >`
- B specifies a Delay condition
- S is a sequence of statements Guaranteed to terminate (a sequence of assignment operations)
- `<>` specify atomic action
- Therefore B is guaranteed to be true when execution of S begins
- No internal state of S is visible to other processes

`< await (s > 0) s = s - 1 >`

Q: What does the above `await` statement indicate?
From last time

Q: Task: Write “code” for delaying the execution process until `count > 0`

```csharp
<await (count > 0);>
```

Q: How do we implement in code an `await` statement?
From last time

Q: Task: Write “code” for delaying the execution process until \( \text{count} > 0 \)

\(<\text{await (count} > 0);>\>

Q: How do we implement in code an await statement?

\(\text{while (not B);}\)

And in this case \(<\text{await (count} > 0);>\> \) is
From last time

Q: Task: Write “code” for delaying the execution process until \texttt{count > 0}

\begin{lstlisting}
<await \texttt{(count > 0)};>
\end{lstlisting}

Q: How do we implement in code an \texttt{await} statement?

\begin{lstlisting}
while (not \texttt{B});
\end{lstlisting}

And in this case <\texttt{await (count > 0)};> is

\begin{lstlisting}
while (not \texttt{count > 0});
\end{lstlisting}
From last time

Q: If you have $n$ processors, how do you find the largest entry of a $n$-element array?
From last time

```
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

The naïve, non concurrent approach

Q: Why is this inefficient (especially on today’s modern computers)?
From last time

The naïve, non concurrent approach

```c
int m = 0;
for [i = 0 to n-1]
  if (a[i] > m)
    m = a[i];
```

Task: Explain in your own words what this accomplishes

Q: What are the shortcomings of this approach?
From last time

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

The naïve, non concurrent approach

```c
int m = 0;
co [i = 0 to n-1] {  
    if (a[i] > m)
        m = a[i];
}
```

Task: Explain in your own words what this accomplishes

```c
int m = 0;
co [i = 0 to n-1] {  
    if (a[i] > m)
    {  
        if (a[i] > m)
            m = a[i];
    }
}
```

Task: Explain in your own words what this accomplishes
Today

Detailed max entry of array example
Code optimization
Concurrent find largest entry

int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];

int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}

int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        if (a[i] > m)
            m = a[i];
}
Concurrent find largest entry

Q: For these three approaches for identifying the largest entry of an array, how does the array’s content affect performance?

Q: Is there a best, worst, average case array?

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

```c
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

```c
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        (if (a[i] > m)
            m = a[i];
    }
```
Concurrent find largest entry

Q: For these three approaches for identifying the largest entry of an array, how does the array’s content affect performance?

Q: Is there a best, worst, average case array?

```
int m = 0;
for [i = 0 to n-1]  
  if (a[i] > m)  
    m = a[i];
```

```
int m = 0;
co [i = 0 to n-1] {  
  if (a[i] > m)  
    m = a[i];
}
```

```
int m = 0;
co [i = 0 to n-1] {  
  if (a[i] > m)  
    if (a[i] > m)  
      m = a[i];
}
```

“worst” case

```
1 2 5 16 17
```

“best” case

```
17 16 5 2 1
```

“average” case

```
2 5 17 1 16
```
Concurrent find largest entry

Q: For these three approaches for identifying the largest entry of an array, how does the array’s content affect performance?

Q: Is there a best, worst, average case array?

Q: How do these three approaches perform on arrays that are fully sorted, somewhat sorted, and fully “un” sorted?
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
  if (a[i] > m)
    m = a[i];
```

"worst" case

| 1 2 5 16 17 |

"best" case

| 17 16 5 2 1 |

"average" case

| 2 5 17 1 16 |

Take a close look at the code.

Q: What actions does it perform?

Q: What are the “costs” of performing those actions?
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

Q: How many checks (whether $a[i]$ is greater than $m$) are performed in the worst, best, and average cases?

- "worst" case: 1 2 5 16 17
- "best" case: 17 16 5 2 1
- "average" case: 2 5 17 1 16
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

Q: How many checks (whether `a[i]` is greater than `m`) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>“worst” case</td>
<td>1</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>“best” case</td>
<td>17</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>“average” case</td>
<td>2</td>
<td>5</td>
<td>17</td>
</tr>
</tbody>
</table>
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

Worst : 5  
Best : 5   
Average : 5

Q: How many updates (to m) are performed in the worst, best, and average cases?

“worst” case
1 2 5 16 17

“best” case
17 16 5 2 1

“average” case
2 5 17 1 16
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

- **Worst**: 5
- **Best**: 5
- **Average**: 5

Q: How many updates (to m) are performed in the worst, best, and average cases?

- **Worst**: 5
- **Best**: 1
- **Average**: 3

“worst” case

```
1 2 5 16 17
```

“best” case

```
17 16 5 2 1
```

“average” case

```
2 5 17 1 16
```
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>“worst” case</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>“best” case</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>“average” case</td>
<td>2</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

Q: How many updates (to m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>“worst” case</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>“best” case</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>“average” case</td>
<td>2</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

Q: If each “check” takes 3 units of time, and each “update” takes 5 units of time, how much total time is required to process the array for the worst, best, and average cases?
Concurrent find largest entry

```
int m = 0;
for [i = 0 to n-1]
  if (a[i] > m)
    m = a[i];
```

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Time</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Q: How many updates (to m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Time</td>
<td>25</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>

Q: What is the total time needed to process each of the worst, best, and average case arrays?

- **“worst” case**
  - 1 2 5 16 17
- **“best” case**
  - 17 16 5 2 1
- **“average” case**
  - 2 5 17 1 16
Concurrent find largest entry

```c
int m = 0;
for [i = 0 to n-1]
    if (a[i] > m)
        m = a[i];
```

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th>Case</th>
<th>Checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>5</td>
</tr>
<tr>
<td>Best</td>
<td>5</td>
</tr>
<tr>
<td>Average</td>
<td>5</td>
</tr>
</tbody>
</table>

Q: How many updates (to m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th>Case</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>5</td>
</tr>
<tr>
<td>Best</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>3</td>
</tr>
</tbody>
</table>

Time

<table>
<thead>
<tr>
<th>Case</th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>“worst” case</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>“best” case</td>
<td>15</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>“average” case</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Q: What is the total time needed to process each of the worst, best, and average case arrays?

<table>
<thead>
<tr>
<th>Case</th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>“worst” case</td>
<td>25</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>“best” case</td>
<td>5</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>“average” case</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

```c
int m = 0;
co [i = 0 to n-1] {
  if (a[i] > m)
    m = a[i];
}
```

“worst” case

1 2 5 16 17

“best” case

17 16 5 2 1

“average” case

2 5 17 1 16
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

```
int m = 0;
co [i = 0 to n-1] {
  (if (a[i] > m)
    m = a[i];)
}
```

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

"worst" case

| 1 | 2 | 5 | 16 | 17 |

"best" case

| 17 | 16 | 5 | 2 | 1 |

"average" case

| 2 | 5 | 17 | 1 | 16 |
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

```java
int m = 0;
co [i = 0 to n-1] {
  if (a[i] > m)
    m = a[i];
}
```

Worst : 5
Best : 5
Average : 5

“worst” case
1 2 5 16 17

“best” case
17 16 5 2 1

“average” case
2 5 17 1 16

Q: How many updates (to m) are performed in the worst, best, and average cases?
Concurrent find largest entry

Q: How many checks (whether \( a[i] \) is greater than \( m \)) are performed in the worst, best, and average cases?

```c
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

Worst: 5
Best: 5
Average: 5

Q: How many updates (to \( m \)) are performed in the worst, best, and average cases?

Worst: 1-5
Best: 1-5
Average: 1-5
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

```java
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

<table>
<thead>
<tr>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Q: How many updates (to m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>1-5</td>
<td>1-5</td>
</tr>
</tbody>
</table>

Q: Under what circumstances is the 1 realized?
Concurrent find largest entry

Q: How many checks (whether \( a[i] \) is greater than \( m \)) are performed in the worst, best, and average cases?

```java
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
    m = a[i];
}
```

- Worst: 5
- Best: 5
- Average: 5

Q: How many updates (to \( m \)) are performed in the worst, best, and average cases?

- Worst: 1
- Best: 5
- Average: 5

Q: Under what circumstances is the 5 realized?

“worst” case: 1 2 5 16 17

“best” case: 17 16 5 2 1

“average” case: 2 5 17 1 16
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

```java
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

Worst: 5
Best: 5
Average: 5

Q: How many updates (to m) are performed in the worst, best, and average cases?

Worst: 1-5
Best: 1-5
Average: 1-5

“worst” case:
1 2 5 16 17

“best” case:
17 16 5 2 1

“average” case:
2 5 17 1 16

Q: If each “check” takes 3 units of time, and each “update” takes 5 units of time, how much total time is required to process the array for the worst, best, and average cases?
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

```c
int m = 0;
co [i = 0 to n-1] { 
  (if (a[i] > m) 
    m = a[i];)
}
```

Q: How many updates (to m) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th>Case</th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“worst” case</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 5 16 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“best” case</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 16 5 2 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“average” case</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 5 17 1 16</td>
</tr>
</tbody>
</table>

Q: What is the total time needed to process each of the worst, best, and average case arrays?

<table>
<thead>
<tr>
<th>Case</th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>5-25</td>
<td>5-25</td>
<td>5-25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“worst” case</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“best” case</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“average” case</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-25</td>
</tr>
</tbody>
</table>

Q: What is the total time needed to process each of the worst, best, and average case arrays?
Concurrent find largest entry

Q: How many checks (whether \(a[i]\) is greater than \(m\)) are performed in the worst, best, and average cases?

```c
int m = 0;
co [i = 0 to n-1] {
    < if (a[i] > m)
    m = a[i];>
}
```

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Q: How many updates (to (m)) are performed in the worst, best, and average cases?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Worst**: 1-5
- **Best**: 1-5
- **Average**: 1-5

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>5-25</td>
<td>5-25</td>
<td>5-25</td>
</tr>
</tbody>
</table>

Q: What is the total time needed to process each of the worst, best, and average case arrays?

- **Worst**: 20-40
- **Best**: 20-40
- **Average**: 20-40

- **“worst” case**
  - 1 2 5 16 17

- **“best” case**
  - 17 16 5 2 1

- **“average” case**
  - 2 5 17 1 16
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

Assume array entries are non-negative

```
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        (if (a[i] > m)
            m = a[i];)
}
```

“worst” case

```
1 2 5 16 17
```

“best” case

```
17 16 5 2 1
```

“average” case

```
2 5 17 1 16
```
Concurrent find largest entry

Q: How many checks (whether \(a[i]\) is greater than \(m\)) are performed in the worst, best, and average cases?

Worst: 5-10
Best: 5-10
Average: 5-10

```
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        if (a[i] > m)
            m = a[i];
}
```
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

Q: How many updates (to m) are performed in the worst, best, and average cases?

```
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        if (a[i] > m)
            m = a[i];
}
```

Worst: 5-10
Best: 5-10
Average: 5-10

“worst” case

```
1 2 5 16 17
```

“best” case

```
17 16 5 2 1
```

“average” case

```
2 5 17 1 16
```
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

Q: How many updates (to m) are performed in the worst, best, and average cases?

```
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        m = a[i];
}
```

Worst : 5-10
Best : 5-10
Average : 5-10

“worst” case
1 2 5 16 17

“best” case
17 16 5 2 1

“average” case
2 5 17 1 16
Concurrent find largest entry

Q: How many checks (whether \(a[i]\) is greater than \(m\)) are performed in the worst, best, and average cases?

```
int m = 0;
cr [i = 0 to n-1] {
    if (a[i] > m)
        ( if (a[i] > m)
            m = a[i];
    }
```

Worst : 5-10
Best : 5-10
Average : 5-10

Q: How many updates (to \(m\)) are performed in the worst, best, and average cases?

Worst : 1-5
Best : 1-5
Average : 1-5

Q: Under what circumstances is the 1 realized? The 5?
Concurrent find largest entry

Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?

Q: How many updates (to m) are performed in the worst, best, and average cases?

```
int m = 0;
co [i = 0 to n-1] { 
  if (a[i] > m) 
    ( if (a[i] > m) 
      m = a[i];
    )
}
```

Worst : 5-10
Best : 5-10
Average : 5-10

Worst : 1-5
Best : 1-5
Average : 1-5

Q: If each “check” takes 3 units of time, and each “update” takes 5 units of time, how much total time is required to process the array for the worst, best, and average cases?
Concurrent find largest entry

**Q:** How many checks (whether \( a[i] \) is greater than \( m \)) are performed in the worst, best, and average cases?

**Q:** How many updates (to \( m \)) are performed in the worst, best, and average cases?

<table>
<thead>
<tr>
<th>Case</th>
<th>Checks</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>5-10</td>
<td>5-10</td>
</tr>
<tr>
<td>Best</td>
<td>5-10</td>
<td>5-25</td>
</tr>
<tr>
<td>Average</td>
<td>5-10</td>
<td>5-25</td>
</tr>
</tbody>
</table>

If each “check” takes 3 units of time, and each “update” takes 5 units of time, how much total time is required to process the array for the worst, best, and average cases?

```
int m = 0;
c0 [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```
Concurrent find largest entry

**Q: How many checks (whether a[i] is greater than m) are performed in the worst, best, and average cases?**

<table>
<thead>
<tr>
<th>Case</th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks</td>
<td>5-10</td>
<td>5-10</td>
<td>5-10</td>
</tr>
</tbody>
</table>

**Q: How many updates (to m) are performed in the worst, best, and average cases?**

<table>
<thead>
<tr>
<th>Case</th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>5-10</td>
<td>5-10</td>
<td>5-10</td>
</tr>
</tbody>
</table>

```c
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        (if (a[i] > m)
            m = a[i];
}
```

**Q: What is the total time needed to process each of the worst, best, and average case arrays?**

<table>
<thead>
<tr>
<th>Case</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>15-30</td>
</tr>
<tr>
<td>Best</td>
<td>15-30</td>
</tr>
<tr>
<td>Average</td>
<td>15-30</td>
</tr>
</tbody>
</table>

**“worst” case**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>5</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
</table>

**“best” case**

<table>
<thead>
<tr>
<th>17</th>
<th>16</th>
<th>5</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

**“average” case**

<table>
<thead>
<tr>
<th>2</th>
<th>5</th>
<th>17</th>
<th>1</th>
<th>16</th>
</tr>
</thead>
</table>
Concurrent find largest entry

Q: How many checks (whether \( a[i] \) is greater than \( m \)) are performed in the worst, best, and average cases?

Q: How many updates (to \( m \)) are performed in the worst, best, and average cases?

```
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        m = a[i];
}
```

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>“worst” case</strong></td>
<td>1-5</td>
<td>5-10</td>
<td>5-10</td>
</tr>
<tr>
<td><strong>“best” case</strong></td>
<td>15-30</td>
<td>15-30</td>
<td>15-30</td>
</tr>
<tr>
<td><strong>“average” case</strong></td>
<td>2 5 17 1 16</td>
<td>5-25</td>
<td>20-55</td>
</tr>
</tbody>
</table>

Q: What is the total time needed to process each of the worst, best, and average case arrays?
Concurrent find largest entry

```java
int m = 0;
for [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

```java
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

```java
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        { if (a[i] > m)
            m = a[i];
        }
}
```

Q: Using the total time that is needed to process each array, which approach would you choose, and why?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>15</td>
<td>15</td>
<td>15-30</td>
</tr>
<tr>
<td>“worst” case</td>
<td>17</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>“best” case</td>
<td>16</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>“average” case</td>
<td>5</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>2</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>Time</td>
<td>20</td>
<td>40</td>
<td>16</td>
</tr>
</tbody>
</table>

Q: Using the total time that is needed to process each array, which approach would you choose, and why?
Concurrent find largest entry

Q: Using the total time that is needed to process each array, which approach would you choose, and why?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>15</td>
<td>15</td>
<td>15-30</td>
</tr>
<tr>
<td>“worst” case</td>
<td>17</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>“best” case</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>“average” case</td>
<td>2</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>15</td>
<td>15-30</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>17</td>
<td>1</td>
</tr>
</tbody>
</table>
Concurrent find largest entry

Q: Using the total time that is needed to process each array, which approach would you choose, and why?

```
int m = 0;
for [i = 0 to n-1] 
  if (a[i] > m) 
    m = a[i];
```

```
int m = 0;
co [i = 0 to n-1] { 
  if (a[i] > m) 
    m = a[i];
}
```

```
int m = 0;
co [i = 0 to n-1] { 
  if (a[i] > m) 
    m = a[i];
  
  if (a[i] > m) 
    m = a[i];
}
```

\begin{tabular}{|c|c|c|}
\hline
Worst & 15 & 15 \ 15-30 \ 
Best & 15 & 15 \ 15-30 \ 
Average & 15 & 15 \ 15-30 \ 
\hline
\end{tabular}

\begin{tabular}{|c|c|c|}
\hline
Worst & 25 & 5-25 \ 5-25 \ 
Best & 5 & 5-25 \ 5-25 \ 
Average & 15 & 5-25 \ 5-25 \ 
\hline
\end{tabular}

Q: Why is the green approach “bad”?

\begin{tabular}{|c|c|c|}
\hline
Worst & 40 & 20-40 \ 20-55 \ 
Best & 20 & 20-40 \ 20-55 \ 
Average & 30 & 20-40 \ 20-55 \ 
\hline
\end{tabular}

\begin{tabular}{|c|c|c|}
\hline
Worst & 17 & 16 \ 5 \ 2 \ 1 \ 
Best & 17 & 16 \ 5 \ 2 \ 1 \ 
Average & 17 & 16 \ 5 \ 2 \ 1 \ 
\hline
\end{tabular}

Q: Using the total time that is needed to process each array, which approach would you choose, and why?
Concurrent find largest entry

```java
int m = 0;
for [i = 0 to n-1] 
  if (a[i] > m) 
    m = a[i];
```
Concurrent find largest entry

Q: Using the total time that is needed to process each array, which approach would you choose, and why?

**“worst” case**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
</table>

**“best” case**

|   | 17 | 16 | 5 | 2 | 1 |

**“average” case**

|   | 2 | 5 | 17 | 1 | 16 |

Q: Why is gray approach preferred over the yellow approach?
Concurrent find largest entry

Q: Using the total time that is needed to process each array, which approach would you choose, and why?

<table>
<thead>
<tr>
<th></th>
<th>Worst</th>
<th>Best</th>
<th>Average</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Worst</strong></td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15-30</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Best</strong></td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15-30</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15-30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These times are “total” times, and do not take into account the speedup that is gained using concurrency.
Concurrent find largest entry

```c
int m = 0;
co [i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

Q: What is the effect on code performance due to the different placements of the atomic signs?
Q: For a 2 entry array, how many CPUs should we employ to find the maximum entry?
Concurrent find largest entry

```java
int m = 0;
co[i = 0 to n-1] {
    if (a[i] > m)
        m = a[i];
}
```

Q: If there are 2 CPUs, and CPU0 “goes” first, what is the order of events that occur when the 2-item array is processed by the code in yellow?

On the board explanation

m=0  5  3
Concurrent find largest entry

```java
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        m = a[i];
}
```

Q: If there are 2 CPUs, and CPU0 “goes” first, what is the order of events that occur when the 2-item array is processed by the code in yellow?

On the board explanation

CPU0
- Fetch a[0]
- Fetch m
- a[0] > m?
- Update m
- DONE

CPU1
- Fetch a[1]
- Fetch m
- a[1] > m?
- DONE

m=5

5

3
Q: If there are 2 CPUs, and CPU0 "goes" first, what is the order of events that occur when the 2-item array is processed by the code in grey?

On the board explanation
Concurrent find largest entry

```
int m = 0;
for (i = 0 to n-1) {
    if (a[i] > m)
        if (a[i] > m)
            m = a[i];
}
```

Q: If there are 2 CPUs, and CPU0 "goes" first, what is the order of events that occur when the 2-item array is processed by the code in grey?

On the board explanation
Concurrent find largest entry

Q: What is the difference between these two approaches?
Creating a concurrent threaded program doesn’t necessarily speed up the run-time during execution.

Task: List multiple reasons why that may be true.
Before we proceed to discussing critical section, locks, barriers, etc. ...

Q: If you have no control over the OS scheduler (how it sends threads to different CPUs and in which order they are invoked), what else can you do to ensure peak performance?
Before we proceed to discussing critical section, locks, barriers, etc. ...

Q: If you have no control over the OS scheduler (how it sends threads to different CPUs and in which order they are invoked), what else can you do to ensure peak performance?

Write better code!
First, some terminology
Q: What issues might arise when this code, with statements S1 and S2, is executed (a) concurrently, and (b) on a single CPU without threading (not concurrently)?

For the remaining examples today, assume that atomicity, semaphores, etc. are not in use.
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[ \begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad y = x
\end{align*} \]
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

Q: How is/are S1 and S2 dependent on each other?

S1: \( x = 3 + i \)
S2: \( y = x \)
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

```
S1: x = 3 + i
S2: y = x
```

Q: How is/are S1 and S2 dependent on each other?

- S2 is flow dependent on S1
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad y = x \\
S3 & \quad x = 3 + i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

Task: Identify all of the flow dependencies in the above code.
Code Optimization

Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

```
S1  x = 3+i
S2  y = x
S3  x = 3+i
S4  y = x
S5  z = y
```

Task: Identify all of the flow dependencies in the above code.

- S4 is flow dependent on S3

Q: Does there exist a second flow dependence?
Task: Identify all of the flow dependencies in the above code.

- S4 is flow dependent on S3
- S5 is flow dependent on S4
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

Task: Identify all of the flow dependencies in the above code.

- S4 is flow dependent on S3
- S5 is flow dependent on S4

Q: Does there exist a third flow dependence?
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3+i \\
S2 & \quad y = x \\
S3 & \quad x = 3+i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

Task: Identify all of the flow dependencies in the above code.

- S4 is flow dependent on S3
- S5 is flow dependent on S4 and also (indirectly) on S3
**Code Optimization**

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

- **S1**: \( x = 3 + i \)
- **S2**: \( y = x \)
- **S3**: \( x = 3 + i \)
- **S4**: \( y = x \)
- **S5**: \( z = y \)

Task: Identify all of the flow dependencies in the above code.

- S2 is flow dependent on S1
- S4 is flow dependent on S3
- S5 is flow dependent on S4 and also (indirectly) on S3

These flow dependencies are “easy” to see especially if we assume that programs are executed sequentially ...

But what if because of concurrency we **cannot** assume that program pieces are all executed sequentially?
Q: How would you analyze this code if (a) concurrency is enabled, and (b) if concurrency is not enabled?

Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3+i \\
S2 & \quad y = x \\
S3 & \quad x = 3+i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

\[
\begin{align*}
S1 & \quad i = x \\
S2 & \quad x = 324
\end{align*}
\]
Code Optimization

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

S1: $x = 3 + i$
S2: $y = x$
S3: $x = 3 + i$
S4: $y = x$
S5: $z = y$

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

S1: $i = x$
S2: $x = 324$
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

Q: How are these dependencies different?
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 &: x = 3 + i \\
S2 &: y = x \\
S3 &: x = 3 + i \\
S4 &: y = x \\
S5 &: z = y
\end{align*}
\]

Anti dependence, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 &: i = x \\
S2 &: x = 324
\end{align*}
\]

Q: How is/are S1 and S2 above dependent on each other?
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad y = x \\
S3 & \quad x = 3 + i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 & \quad i = x \\
S2 & \quad x = 324
\end{align*}
\]

Q: How is/are S1 and S2 above dependent on each other?

S2 is not flow dependent on S1, but re-ordering S2 and S1 would affect the final value of i.
Code Optimization

Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & \quad x = 3+i \\
S2 & \quad y = x
\end{align*}
\]

Anti dependence, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 & \quad i = x \\
S2 & \quad x = 324
\end{align*}
\]

\[
\begin{align*}
S3 & \quad x = 3+i \\
S4 & \quad y = x \\
S5 & \quad z = y
\end{align*}
\]

\[
\begin{align*}
S3 & \quad x = 42 \\
S4 & \quad y = x + 17 \\
S5 & \quad x = 6
\end{align*}
\]

Task: Identify all of the anti dependencies in the above code.
Code Optimization

**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S_1 & \quad x = 3 + i \\
S_2 & \quad y = x \\
S_3 & \quad x = 3 + i \\
S_4 & \quad y = x \\
S_5 & \quad z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S_1 & \quad i = x \\
S_2 & \quad x = 324 \\
S_3 & \quad x = 42 \\
S_4 & \quad y = x + 17 \\
S_5 & \quad x = 6
\end{align*}
\]

Task: Identify all of the anti dependencies in the above code.

S4 anti-depends on S5, because changing their ordering would affect the value of y.

Q: Any other anti-dependencies?
Code Optimization

Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

S1  \( x = 3 + i \)
S2  \( y = x \)

S3  \( x = 3 + i \)
S4  \( y = x \)
S5  \( z = y \)

Anti dependence, Write-after-read (WAR): an instruction requires a value that is later updated.

S1  \( i = x \)
S2  \( x = 324 \)

S3  \( x = 42 \)
S4  \( y = x + 17 \)
S5  \( x = 6 \)

Q: Why are flow and anti-dependencies of special concern in concurrent programming?
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S1 & : x = 3 + i \\
S2 & : y = x \\
S3 & : x = 3 + i \\
S4 & : y = x \\
S5 & : z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S1 & : i = x \\
S2 & : x = 324 \\
S3 & : x = 42 \\
S4 & : y = x + 17 \\
S5 & : x = 6
\end{align*}
\]

**Q:** How can we eliminate anti dependence (between S4 and S5) from this code? Assume S3 goes first, and you cannot modify S3

**Q:** How would you rewrite this code?

In class exercise
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

- **S1** \( x = 3 + i \)
- **S2** \( y = x \)

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

- **S1** \( i = x \)
- **S2** \( x = 324 \)

**Q:** Does it matter which of S4 or S5 goes last? (S3 still goes first)

- **S3** \( x = 42 \)
- **S4** \( y = x + 17 \)
- **S5** \( x = 6 \)

- **S3** \( x = 42 \)
- **S6** \( z = x \)
- **S4** \( y = z + 17 \)
- **S5** \( x = 6 \)
**Flow dependence** (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[
\begin{align*}
S_1 & \quad x = 3 + i \\
S_2 & \quad y = x \\
S_3 & \quad x = 3 + i \\
S_4 & \quad y = x \\
S_5 & \quad z = y
\end{align*}
\]

**Anti dependence**, Write-after-read (WAR): an instruction requires a value that is later updated.

\[
\begin{align*}
S_1 & \quad i = x \\
S_2 & \quad x = 324 \\
S_3 & \quad x = 42 \\
S_4 & \quad y = x + 17 \\
S_5 & \quad x = 6
\end{align*}
\]

**Q**: Does it matter which of S4 or S5 goes last? (S3 still goes first)

**Q**: What are the possible concurrent histories such that S3 goes first and S6 < S4?
Flow dependence (true dependence), Read-after-write (RAW): when an instruction depends on the results of a previous instruction.

\[ \text{S1: } x = 3 + i \]
\[ \text{S2: } y = x \]

Anti dependence, Write-after-read (WAR): an instruction requires a value that is later updated.

\[ \text{S1: } i = x \]
\[ \text{S2: } x = 324 \]

Q: Does it matter which of S4 or S5 goes last? (S3 still goes first)

Notice that “fixing” the anti dependence between S4 and S5 has introduced another one (S4 is now anti-dependent on S6)

\[ \text{S3: } x = 42 \]
\[ \text{S4: } y = x + 17 \]
\[ \text{S5: } z = y \]

\[ \text{S3: } x = 42 \]
\[ \text{S6: } z = x \]
\[ \text{S4: } y = z + 17 \]
\[ \text{S5: } x = 6 \]
Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

Q: How is/are S1 and S2 output dependent?
Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value

\[
\begin{align*}
S1 & \quad x = 3 + i \\
S2 & \quad x = x \times 2
\end{align*}
\]

Q: How is/are S1 and S2 output dependent?

S2 is output dependent on S1
(S2 is also flow dependent on S1)
**Output dependence**, Write-after-write (WAW): when the ordering of instructions affects the final output value.

```
S1  x = 3+i
S2  x = x*2
S3  x = 3+i
S4  y = x/4
S5  x = 32
```

**Task**: Identify the output dependency in the above code.
Task: Identify the output dependency in the above code

S4 is output dependent on S3 and S5, because if S3 and S5 are reordered, that may affect S4.
Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

Q: How can we eliminate the output dependence of S4 on S3 and S5?

Q: How would you rewrite this code?

In class exercise
Code Optimization

Output dependence, Write-after-write (WAW): when the ordering of instructions affects the final output value.

\[
\begin{align*}
S1 & : x = 3+i \\
S2 & : x = x*2 \\
S3 & : x = 3+i \\
S4 & : y = x/4 \\
S5 & : x = 32
\end{align*}
\]

Rename and/or copy variables

Q: Does it matter if S3 or S5 goes first?

Q: What are the possible concurrent histories when S3 < S4?
Code Optimization

**Control Dependence**: an instruction is control dependent on a preceding instruction if the output of the latter *can* determine if the former should be executed.
Code Optimization

**Control Dependence**: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

```
S6  if (x == y)
S7   x = x + y
S8   y = x + y
```

Task: Identify the control dependency in the above code.
Code Optimization

**Control Dependence**: an instruction is control dependent on a preceding instruction if the output of the latter *can* determine if the former should be executed.

```
S6: if (x == y)
S7: x = x + y
S8: y = x + y
```

Control dependence is a bit more subtle ... you must also inspect the “logic” of the code in addition to noticing the pattern.
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

```c
S6 if (x == y)
S7 x = x + y
S8 y = x + y
```

```c
c = [ 3, 4, 5, 7, 4, 3]
a = [ -2, -3, 3, 4, 5, 6 ]
for (i=0; i<n; i++){
    a[i] = c[i];
    if (a[i] < 0){
        a[i] = 1;
    }
}
```
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

```
c = [3, 4, 5, 7, 4, 3]
a = [-2, -3, 3, 4, 5, 6]
for (i=0; i<n; i++){
    a[i] = c[i];
    if (a[i] < 0){
        a[i] = 1;
    }
}
```

```
a = [-2, -3, 3, 4, 5, 6]
for (i=1; i<n; i++){
    if (a[i-1] < 0){
        a[i] = -1;
    }
}
```

To answer this question, “Execute” the code.
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

```c
for (i=0; i<n; i++){
    a[i] = c[i];
    if (a[i] < 0){
        a[i] = 1;
    }
}
```

To answer this question, “Execute” the code.

Q: Does the if’s code block affect the evaluation of the conditional?

```c
for (i=0; i<n; i++){
    a[i] = c[i];
    if (a[i] < 0){
        a[i] = 1;
    }
}
```
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Although here the a[i]’s value determines the evaluation of the conditional, because of the logic of the program, will the if’s code block EVER influence the conditional expression?

Q: Which of these is an example of control dependence?

To answer this question, “Execute” the code
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

\[
c = [3, 4, 5, 7, 4, 3] \\
a = [-2, -3, 3, 4, 5, 6] \\
\text{for } (i=0; i<n; i++)\{ \\
\quad a[i] = c[i]; \\
\quad \text{if } (a[i] < 0)\{ \\
\quad \quad a[i] = 1; \\
\quad \} \\
\}
\]

\[
a = [-2, -3, 3, 4, 5, 6] \\
\text{for } (i=1; i<n; i++)\{ \\
\quad \text{if } (a[i-1] < 0)\{ \\
\quad \quad a[i] = -1; \\
\quad \} \\
\}
\]

Q: Does the if’s code block affect the evaluation of the conditional?

To answer this question, “Execute” the code.
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

Here the updating of the \texttt{a[i]} WILL affect the evaluation of the Boolean expression in subsequent iterations of the for loop.
Control Dependence: an instruction is control dependent on a preceding instruction if the output of the latter can determine if the former should be executed.

Q: Which of these is an example of control dependence?

No control dependence:
```python
c = [3, 4, 5, 7, 4, 3]
a = [-2, -3, 3, 4, 5, 6]
for (i=0; i<n; i++){
a[i] = c[i];
if (a[i] < 0){
a[i] = 1;
}
}
```

Control dependence:
```python
c = [3, 4, 5, 7, 4, 3]
a = [-2, -3, 3, 4, 5, 6]
for (i=1; i<n; i++){
if (a[i-1] < 0){
a[i] = -1;
}
}
```
Take home question: For which ranges of $i$ sent to two threads is there control dependence when the threads are executed concurrently?

```c
a = [ -2, -3, 3, 4, 5, 6 ]
for (i=1; i<n; i++){
    if (a[i-1] < 0){
        a[i] = -1;
    }
}
```
Big picture: Reordering, copying of, and renaming of variables in “intro” courses is irrelevant because those programs are executed sequentially (top-down). But concurrently executed programs may interweave instructions in different orders, which DOES affect a program’s final state.

Q: Just because we CAN optimize code, should we?
Up Next

Code Optimization
Critical Sections