The UPC Memory Model

Examples
Memory model rules

- Re-ordering restrictions on a thread:
  - ProgramOrder: Two operations cannot be reordered if they operate on the same variable and one of them is a write.
  - All threads must observe all writes (to shared memory) before the end of the program.
  - ReadOrder: A read operation gets its value from the latest observed write.
Memory model rules

● UPC specific rules:
  – StrictOrder: Two operations on the same thread cannot be reordered if at least one of them is strict.
  – All threads must observe all strict operations.
  – All threads must agree on the order of all strict operations.
    ● All operations issued before a strict operation must be observed before that strict operation by all threads.
Example 1

- Behavior that is illegal in SC, but **legal** in UPC.
- “Who was here before me?”

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(x)</code></td>
<td><code>print(x)</code></td>
</tr>
<tr>
<td><code>x = 1</code></td>
<td><code>x = 2</code></td>
</tr>
</tbody>
</table>

**Program**

**Output**

Can they both be second?
Memory ops

- “print(x)” is converted to a relaxed read
- “x = ?” is converted to a relaxed write

Thread 1

- print(x)
- x = 1

Thread 2

- print(x)
- x = 2
Adjusting for execution trace

- Filling in the observed values for the reads

```
print(x)
x = 1
print(x)
x = 2
```
Strict global order

- There are no strict operations so no global strict order!
Thread 1 ordering

- RR(x,2)
- RW(x,1)
- RW(x,2)

ReadOrder
Thread 2 ordering

- RW(x,1)
- RR(x,1)
- RW(x,2)

ReadOrder
Example 1 - summary

- Both thread-orderings are legal, global ordering is legal, so execution trace is UPC MM compliant!
- Legal because threads can disagree on order of relaxed writes.
Example 1 - Fix

- Adding a fence

**Thread 1**
- print(x)
- upc_fence()
- x = 1

**Thread 2**
- print(x)
- x = 2

```
RR(x,2)
  ↓
fence()
  ↓
RW(x,1)
RR(x,1)
  ↓
RW(x,2)
```
Example 1 - Fix

- Adding a fence

Thread 1

- RR(x,2)
- fence()
- RW(x,1)

Thread 2

- RR(x,1)
- fence()
- RW(x,1)
- RW(x,2)
Example 2

- Strict reads cannot be reordered

**Thread 1**

\[
\begin{align*}
  x &= 1 \\
  x &= 2
\end{align*}
\]

**Thread 2**

\[
\begin{align*}
  \text{print}(x) \\
  \text{print}(x)
\end{align*}
\]

Thread 2 is using strict operations only.
Example 2

Thread 1

\( x = 1 \)
\( x = 2 \)

Thread 2

\( \text{print}(x) \)
\( \text{print}(x) \)

Output

ProgramOrder

StrictOrder

ProgramOrder

RW(\(x,1\))

SR(\(x,2\))

RW(\(x,2\))

SR(\(x,1\))
Example 2

- Both threads will have problems explaining the behavior.

<table>
<thead>
<tr>
<th>Program</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>1</td>
</tr>
<tr>
<td>Thread 2</td>
<td>2</td>
</tr>
</tbody>
</table>

```
x = 1
x = 2
```

```
print(x)
print(x)
```
Example 3

- But writes can be reordered

**Thread 1**
- `x = 1`
- `y = 1`

**Thread 2**
- `print(y)`
- `print(x)`

Thread 2 is using strict operations only.
Example 3

- But writes can be reordered

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1</td>
<td>print(y)</td>
</tr>
<tr>
<td>y = 1</td>
<td>print(x)</td>
</tr>
</tbody>
</table>

Program

Output

1
0

RW(x,1)
SR(y,1)
RW(y,1)
SR(x,0)

StrictOrder
Example 3

- But writes can be reordered

```
x = 1
y = 1
print(y)
print(x)
```

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1</td>
<td>print(y)</td>
</tr>
<tr>
<td>y = 1</td>
<td>print(x)</td>
</tr>
</tbody>
</table>

Program

Output

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

```
RW(y,1)
SR(x,0)
SR(y,1)
RW(x,1)
```