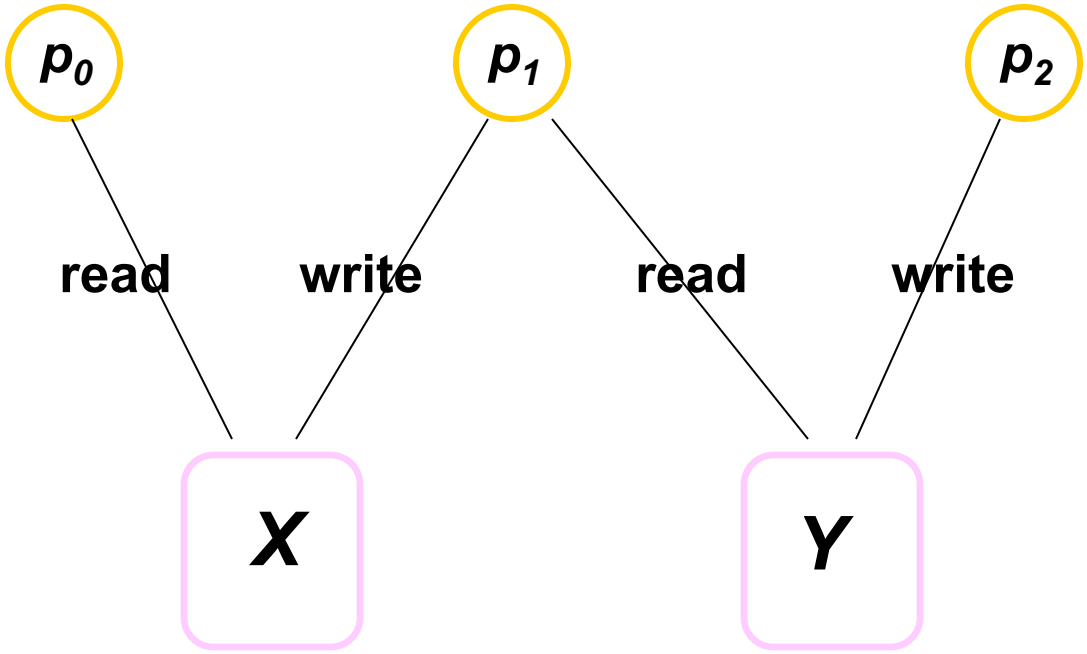
SHARED MEMORY MUTUAL EXCLUSION

Prof. Jennifer Welch

Shared Memory Model

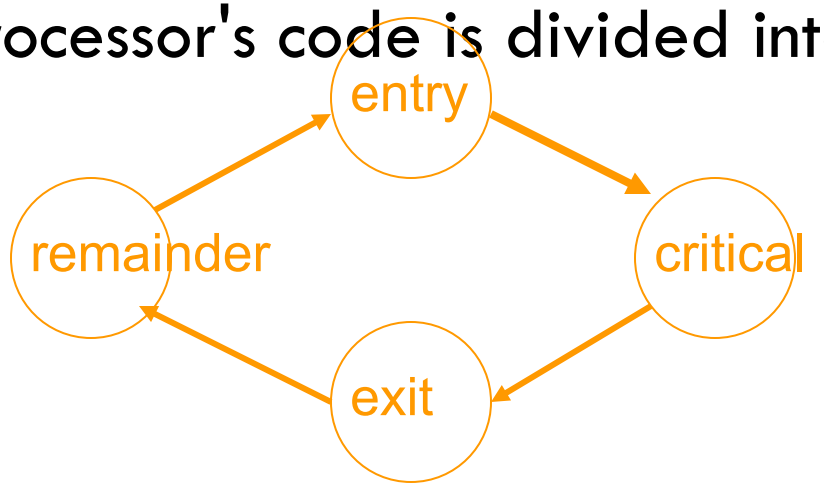
- Processors communicate via a set of shared variables, instead of passing messages.
- Each shared variable has a **type**, defining a set of operations that can be performed *atomically*.

Shared Memory Model Example



Mutual Exclusion (Mutex) Problem

- Each processor's code is divided into four sections:



  - **entry:** synchronize with others to ensure mutually exclusive access to the …
  - **critical:** use some resource; when done, enter the…
  - **exit:** clean up; when done, enter the…
  - **remainder:** not interested in using the resource

Read-Modify-Write Variables

Test-and-Set Shared Variable

- A **test-and-set** variable *V* holds two values, 0 or 1, and supports two (atomic) operations:
  - test&set(*V*):

```
temp := v
v := 1
return temp
```

  - reset(V):

```
v := 0
```

Mutex Algorithm Using Test&Set

- code for entry section:

```
repeat
     t := test&set(V)
until (t = 0)
```

An alternative syntactic construction is:

```
wait until test&set(V) = 0
```

- code for exit section:

```
reset(V)
```

Read-Write Variables

Read/Write Shared Variables

- In one atomic step a processor can
  - read the variable or
  - write the variable
  - but not both!

Bakery Algorithm

- An algorithm using *2n* shared read/write variables
  - booleans `Choosing[i]` : initially false, written by p_i and read by others
  - integers `Number[i]` : initially 0, written by p_i and read by others

Bakery Algorithm

Code for entry section:

```
Choosing[i] := true
Number[i] := max{Number[0], …, Number[n-1]} + 1
Choosing[i] := false
for j := 0 to n-1 (except i) do
    wait until Choosing[j] = false
    wait until Number[j] = 0 or
     (Number[j],j) > (Number[i],i)
endfor
```

Code for exit section:

```
Number[i] := 0
```

Space Complexity of Bakery Algorithm

- Number of shared variables is $2n$
- `Choosing` variables are boolean
- `Number` variables are unbounded
- Is it possible for an algorithm to use less shared space?

2-Processor ME Algorithm

Uses 3 binary shared read/write variables:

- `W[0]` : initially 0, written by p_0 and read by p_1
- `W[1]` : initially 0, written by p_1 and read by p_0
- `Priority` : initially 0, written and read by both

2-Processor Mutex Algorithm

- Start with a bounded algorithm for 2 processors with ND, then extend to NL, then extend to *n* processors.
- Some ideas used in 2-processor algorithm:
  - each processor has a shared boolean `W[i]` indicating if it wants the CS
  - p_0 always has priority over p_1 ; asymmetric code

2-Processor Mutex Algorithm

Code for p_0 's entry section:

```
1    .
2    .
3    W[0] := 1
4    .
5    .
6    wait until W[1] = 0
```

Code for p_0 's exit section:

```
7    .
8    W[0] := 0
```

2-Processor Mutex Algorithm

Code for p_1 's entry section:

```
1    W[1] := 0
2    wait until W[0] = 0
3    W[1] := 1
4    .
5        if (W[0] = 1) then goto Line 1
6    .
```

Code for p_1 's exit section:

```
7    .
8    W[1] := 0
```

Discussion of 2-Processor Algorithm

- Satisfies mutual exclusion: processors use `W` variables to make sure of this
- Does not deadlock, but may be unfair to one processor
- Fix by having the processors alternate having the priority:
  - shared variable `Priority`, read and written by both

2-Processor Mutex Algorithm

Code for entry section:

```
1   W[i] := 0
2   wait until W[1-i] = 0 or Priority = i
3   W[i] := 1
4   if (Priority = 1-i) then
5         if (W[1-i] = 1) then goto Line 1
6   else wait until (W[1-i] = 0)
```

Code for exit section:

```
7   Priority := 1-i
8   W[i] := 0
```