Synchronization: Implementing Barriers Promises + Futures

Chris Rossbach

CS378H

Today

- Questions?
- Administrivia
  - Lab 2 due sooner than you'd like
- Material for the day
  - Barrier implementation
  - Promises & Futures

- Acknowledgements
  - Thanks to Gadi Taubenfield: I borrowed from some of his slides on barriers

Faux Quiz (answer any N, 5 min)

- How are promises and futures related? Since there is disagreement on the nomenclature, don't worry about which is which—just describe what the different objects are and how they function.

Barriers

Barriers

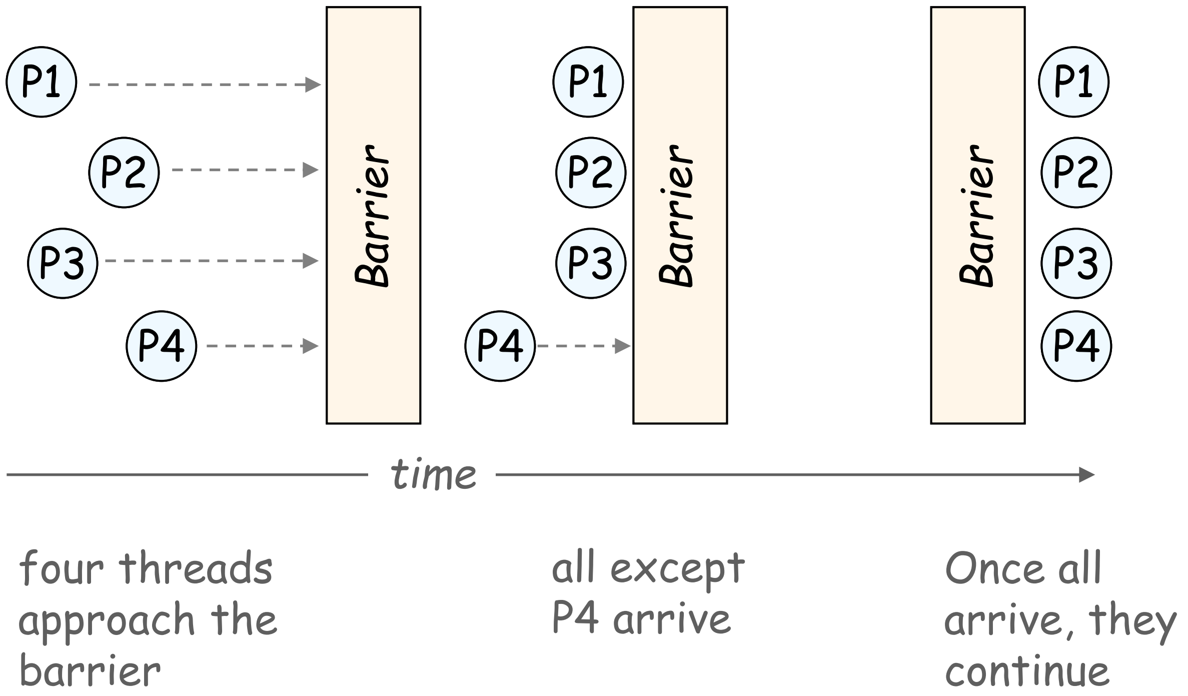
Review: Barrier Basics

Review: Barrier Basics

- *Coordination mechanism*
- *participants wait until all reach same point.*
- *Once all reach it, all can pass.*

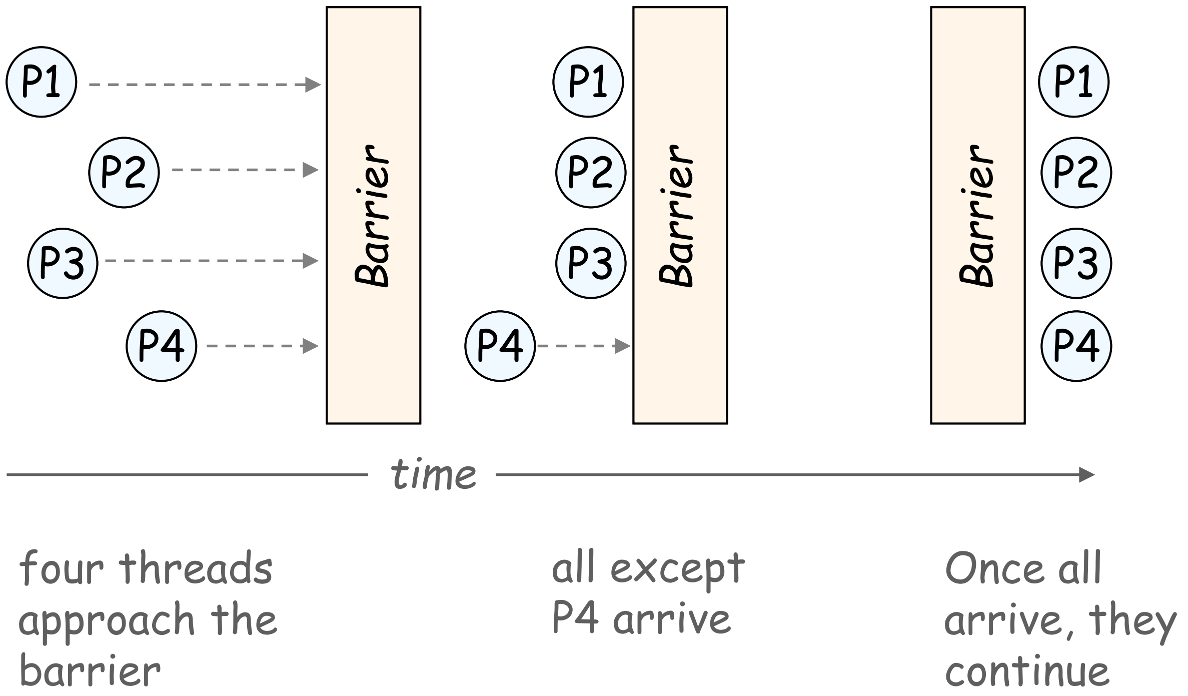
Review: Barrier Basics

- ➢ *Coordination mechanism*
- ➢ *participants wait until all reach same point.*
- ➢ *Once all reach it, all can pass.*



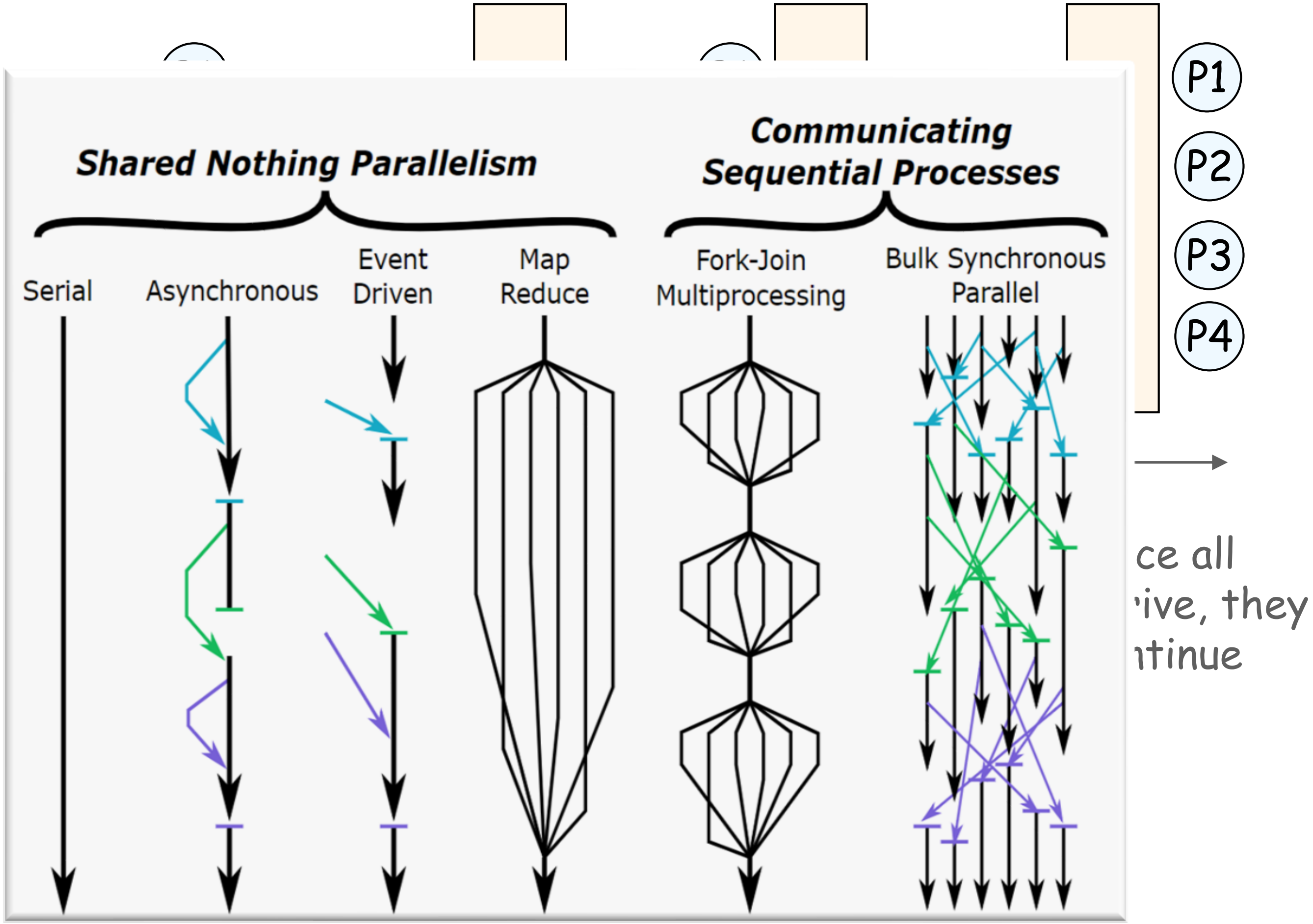
Review: Barrier Basics

- *Coordination mechanism*
- *participants wait until all reach same point.*
- *Once all reach it, all can pass.*
- ***Workhorse of BSP programming models***



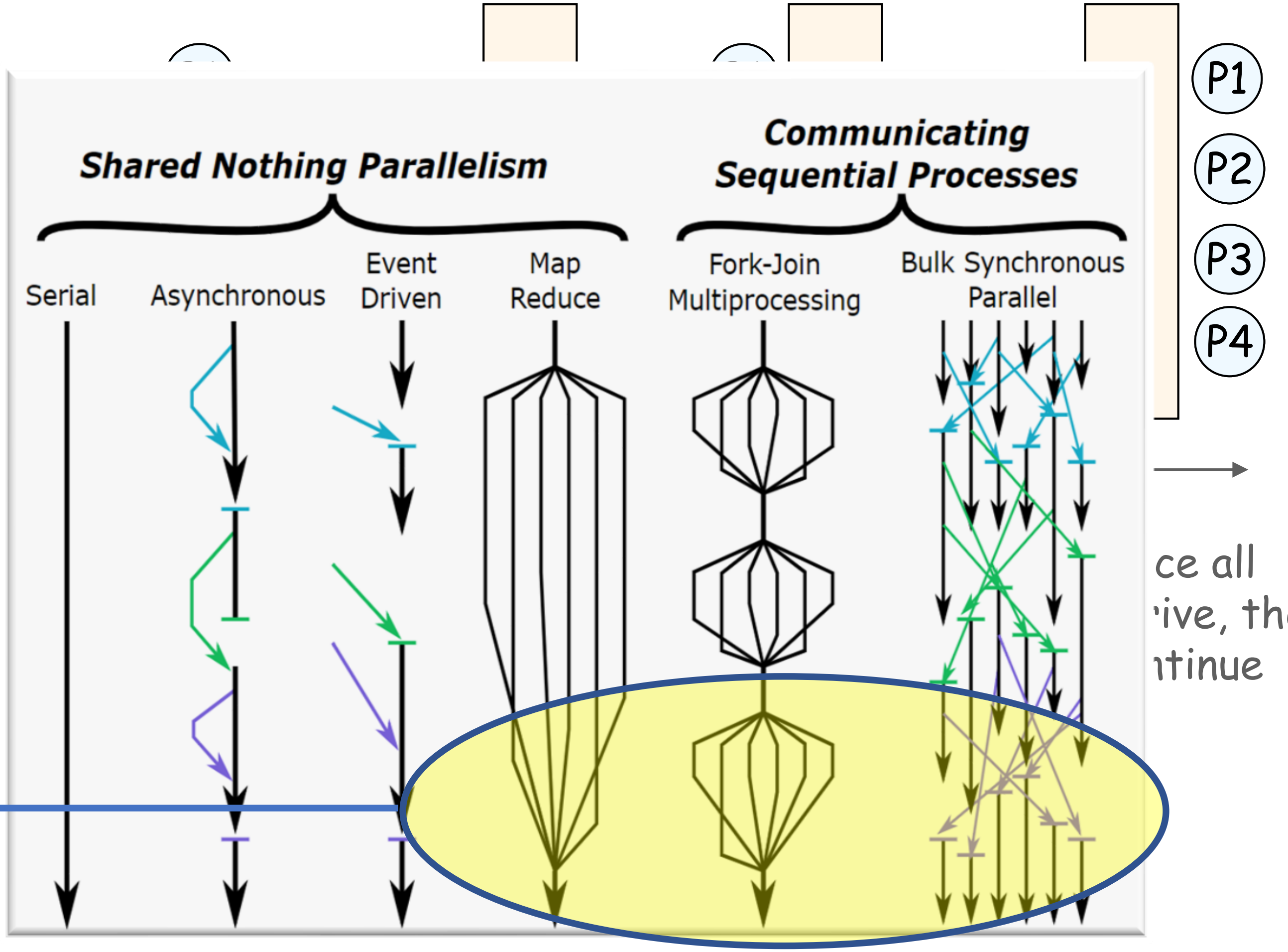
Review: Barrier Basics

- *Coordination mechanism*
- *participants wait until all reach same poin*
- *Once all reach it, all can pass.*
- ***Workhorse of BSP programming models***



Review: Barrier Basics

- *Coordination mechanism*
- *participants wait until all reach same poin*
- *Once all reach it, all can pass.*
- ***Workhorse of BSP programming models***



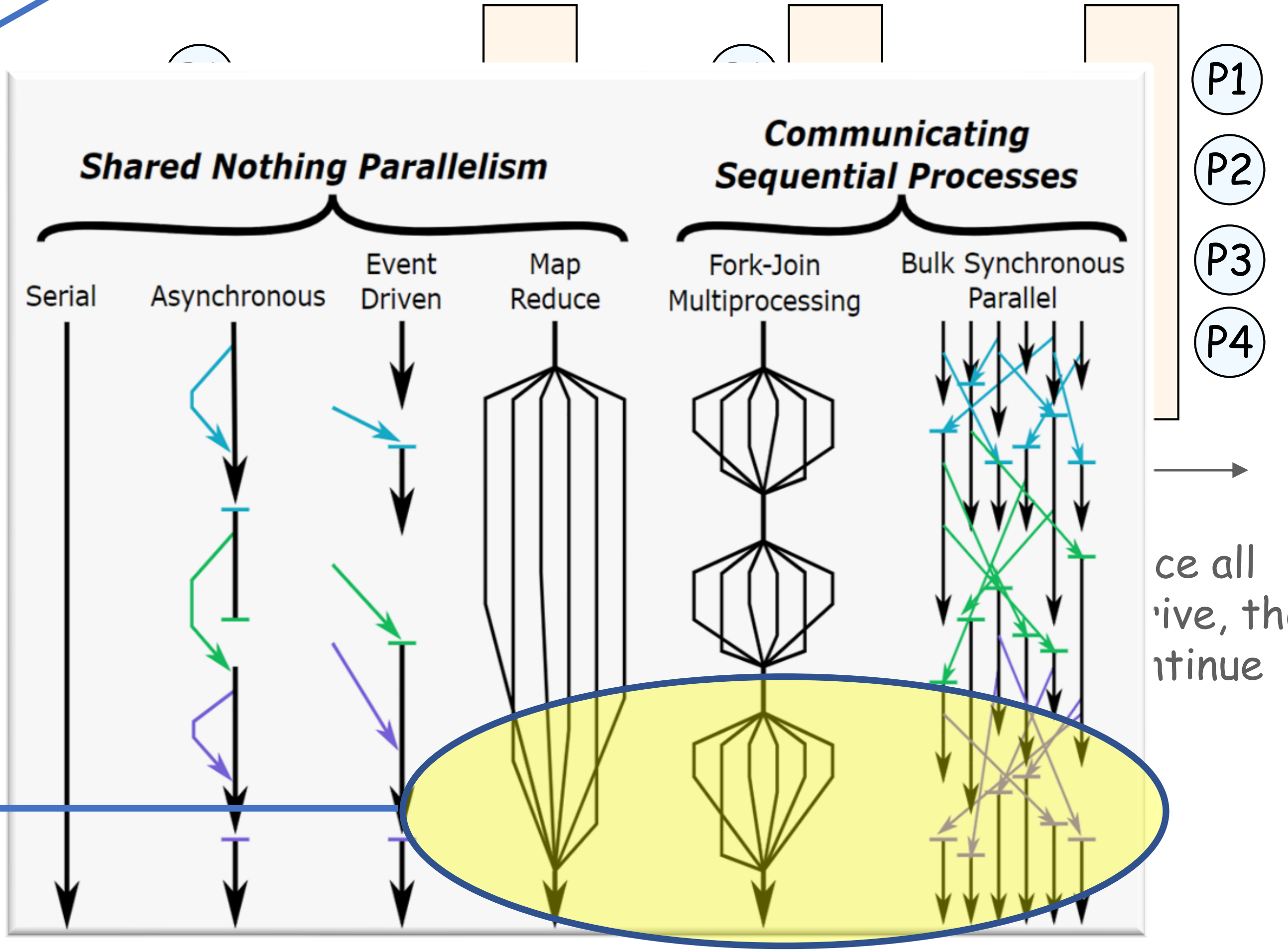
ce all
ive, they
itinue

Fundamental primitive in many parallel models

Review: Barrier Basics

Can you make a lock with a barrier?

- Coordination mechanism
- *participants wait until all reach same poin*
- *Once all reach it, all can pass.*
- ***Workhorse of BSP programming models***



ce all
ive, they
itinue

Fundamental primitive in many parallel models

Barriers based on counters

Counter Barrier Ingredients

Fetch-and-Increment register

- A shared register that supports a F&I operation:
- Input: register *r*
- Atomic operation:
  - *r* is incremented by 1
  - the old value of r is returned

```
function fetch-and-increment (r : register)
        orig_r := r;
        r:= r + 1;
        return (orig_r);
end-function
```

Await

- For brevity, we use the **await** macro
- Not an operation of an object
- This is just "spinning"

```
macro await (condition : boolean condition)
        repeat
                cond = eval(condition);
        until (cond)
end-macro
```

Simple Barrier Using an Atomic Counter

shared	counter: fetch and increment reg. – {0,..n}, initially = 0
	go: atomic bit, initial value does not matter
local	local.go: a bit, initial value does not matter
	local.counter: register

Simple Barrier Using an Atomic Counter

shared	counter: fetch and increment reg. – {0,..n}, initially = 0
	go: atomic bit, initial value does not matter
local	local.go: a bit, initial value does not matter
	local.counter: register

```
local.go := go
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        go := 1 - go
else await(local.go ≠ go)
```

Simple Barrier Using an Atomic Counter

shared	counter: fetch and increment reg. – {0,..n}, initially = 0
	go: atomic bit, initial value does not matter
local	local.go: a bit, initial value does not matter
	local.counter: register

```
local.go := go
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        go := 1 - go
else await(local.go ≠ go)
```

Pros/Cons?

Simple Barrier Using an Atomic Counter

shared	counter: fetch and increment reg. – {0,..n}, initially = 0
	go: atomic bit, initial value does not matter
local	local.go: a bit, initial value does not matter
	local.counter: register

```
1   local.go := go
2   local.counter := fetch-and-increment (counter)
3   if local.counter + 1 = n then
4           counter := 0
5           go := 1 - go
6   else await(local.go ≠ go)
```

Pros/Cons?

Simple Barrier Using an Atomic Counter

shared counter: fetch and increment reg. – {0,..n}, initially = 0
go: atomic bit, initial value does not matter

local local.go: a bit, initial value does not matter
local.counter: register

```
local.go := go
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
    counter := 0
    go := 1 - go
else await(local.go ≠ go)
```

Pros/Cons?

- There is high memory contention on *go* bit

Simple Barrier Using an Atomic Counter

shared counter: fetch and increment reg. – {0,..n}, initially = 0
go: atomic bit, initial value does not matter
local local.go: a bit, initial value does not matter
local.counter: register

local.go := go
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n **then**
    counter := 0
    go := 1 - go
else await(local.go ≠ go)

Pros/Cons?

- There is high memory contention on *go* bit
- Reducing the contention:
  - Replace the *go* bit with *n* bits: *go*[1],…,*go*[*n*]
  - Process p_i may spin only on the bit *go*[*i*]

A Local Spinning Counter Barrier

Program of a Thread i

shared	counter: fetch and increment reg. – {0,..n}, initially = 0
	go[1..n]: array of atomic bits, initial values are immaterial
local	local.go: a bit, initial value is immaterial
	local.counter: register

A Local Spinning Counter Barrier

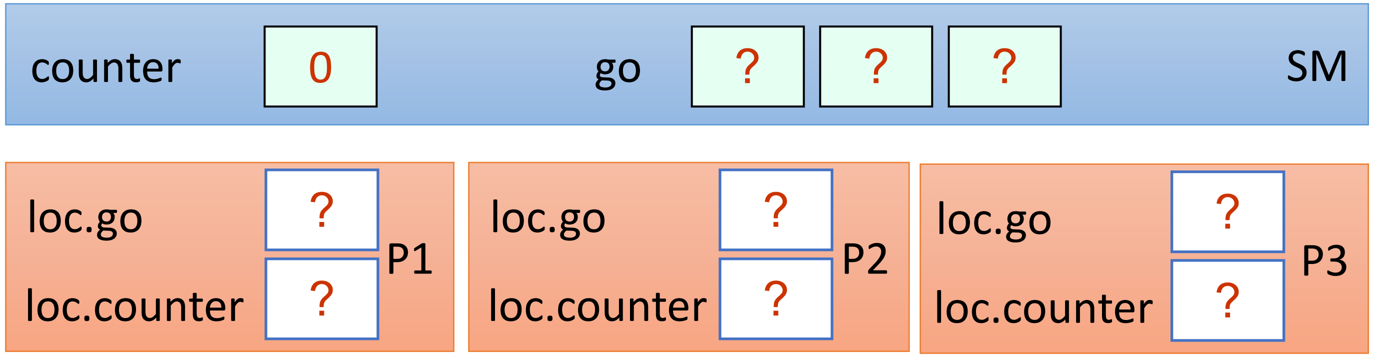
Program of a Thread i

shared	counter: fetch and increment reg. – {0,..n}, initially = 0
	go[1..n]: array of atomic bits, initial values are immaterial
local	local.go: a bit, initial value is immaterial
	local.counter: register

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

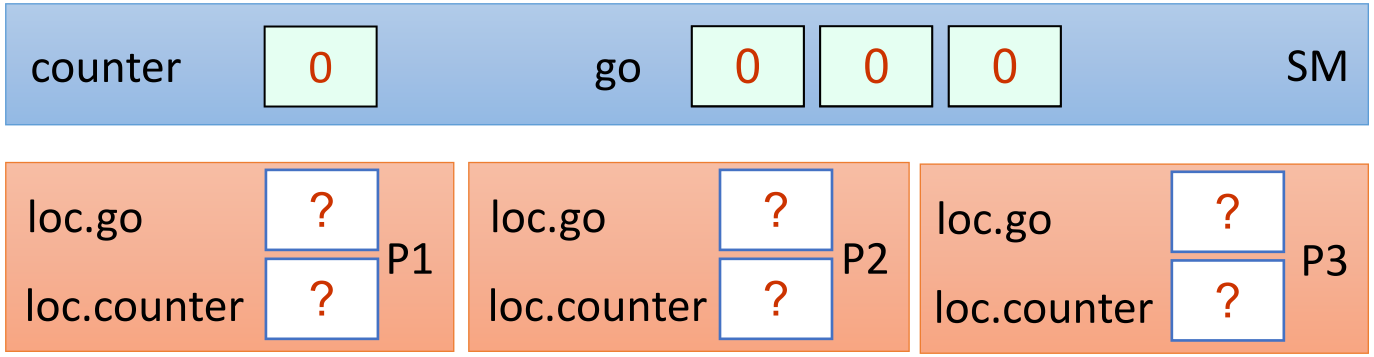
Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

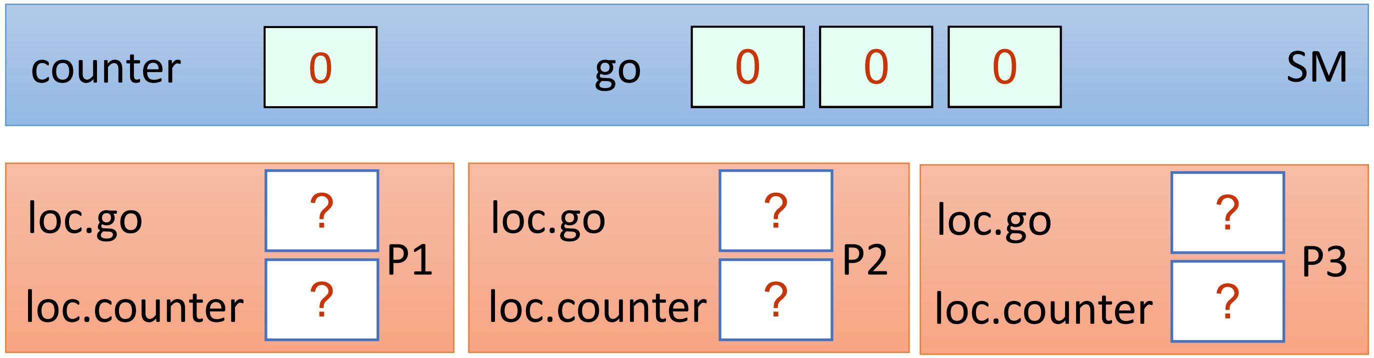
Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads

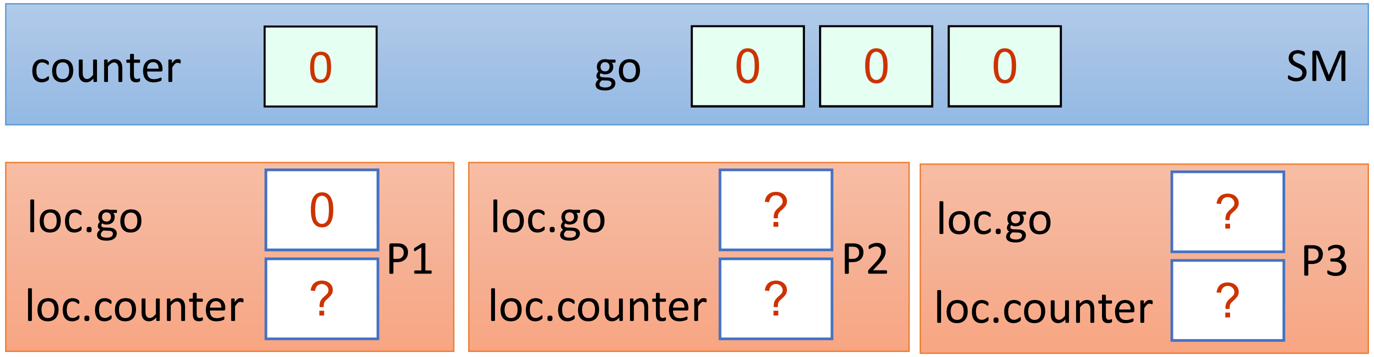


P1 →

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

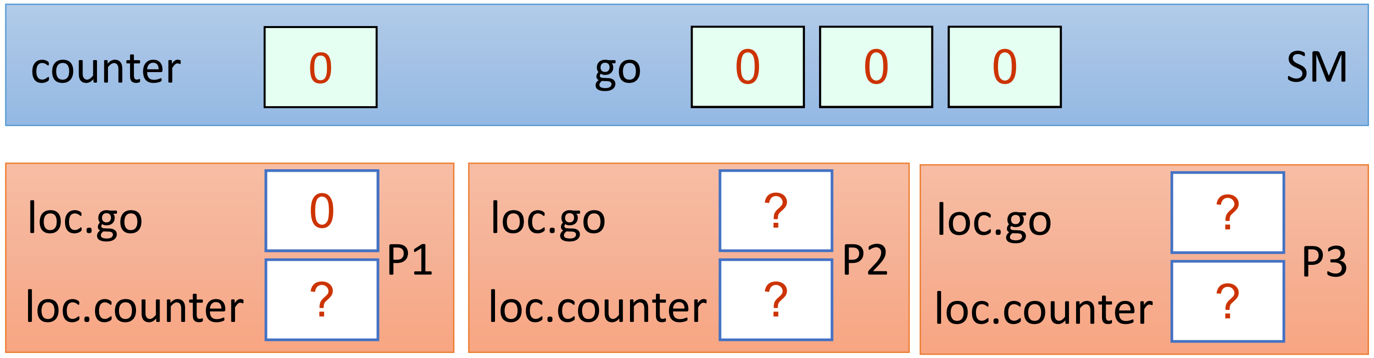
Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads

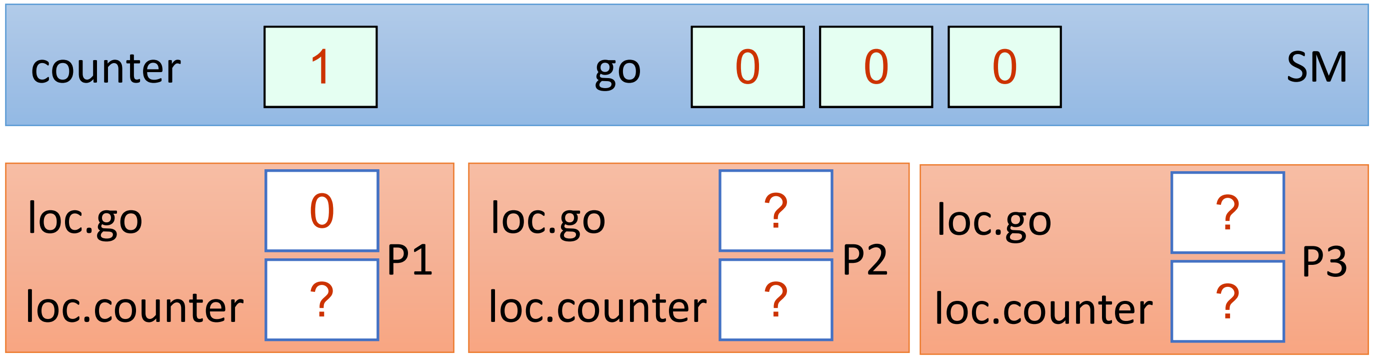


P1 →

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads

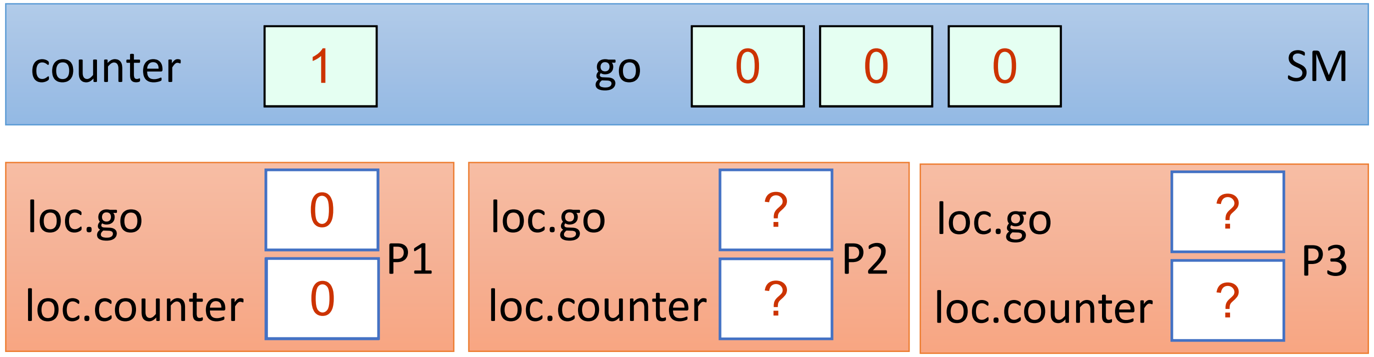


P1 →

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads

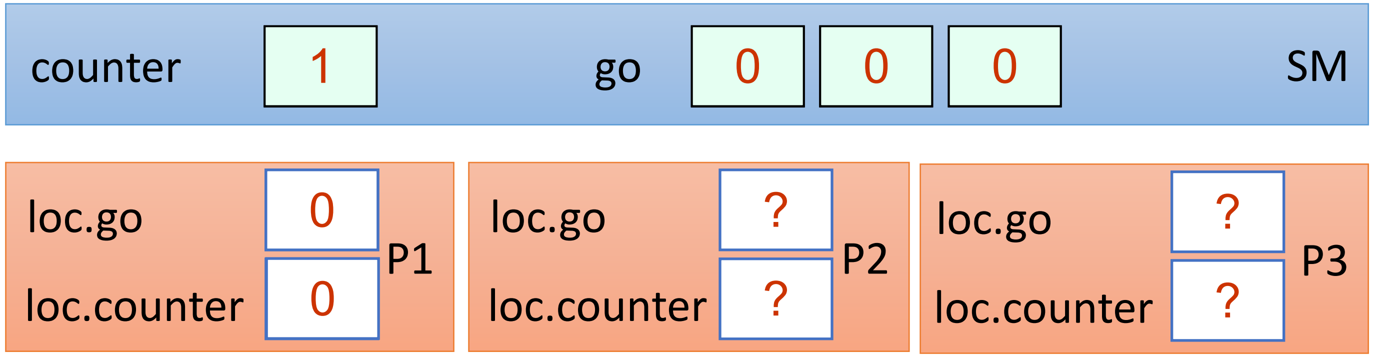


P1 →

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads

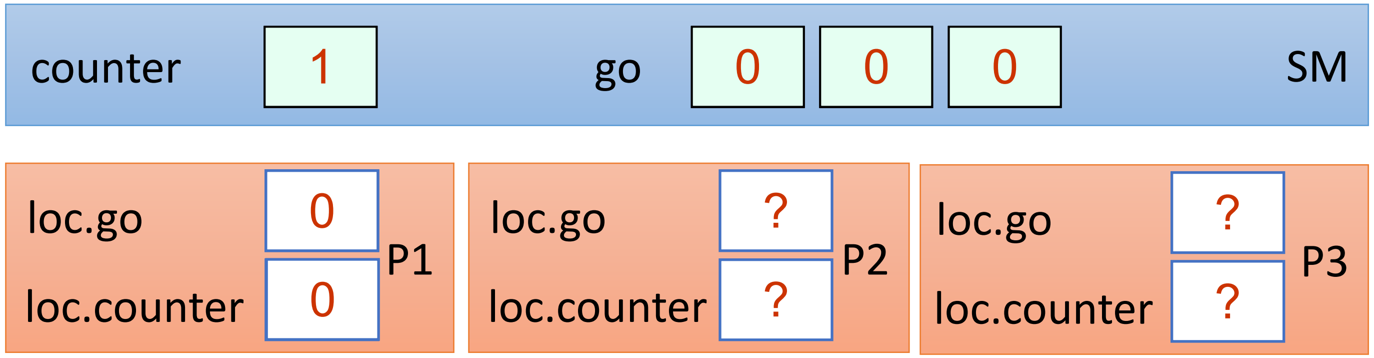


P1 → (line 3)

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads



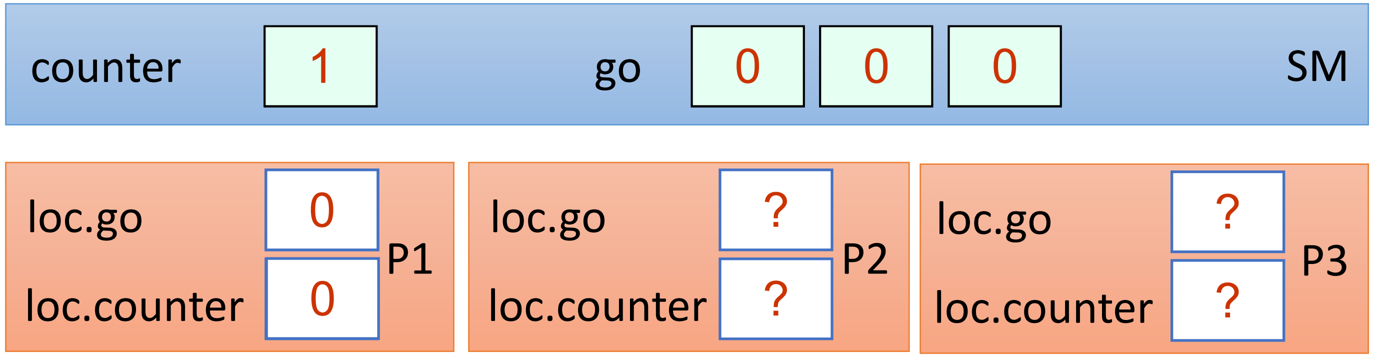

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P1 →

0+1≠3

A Local Spinning Counter Barrier

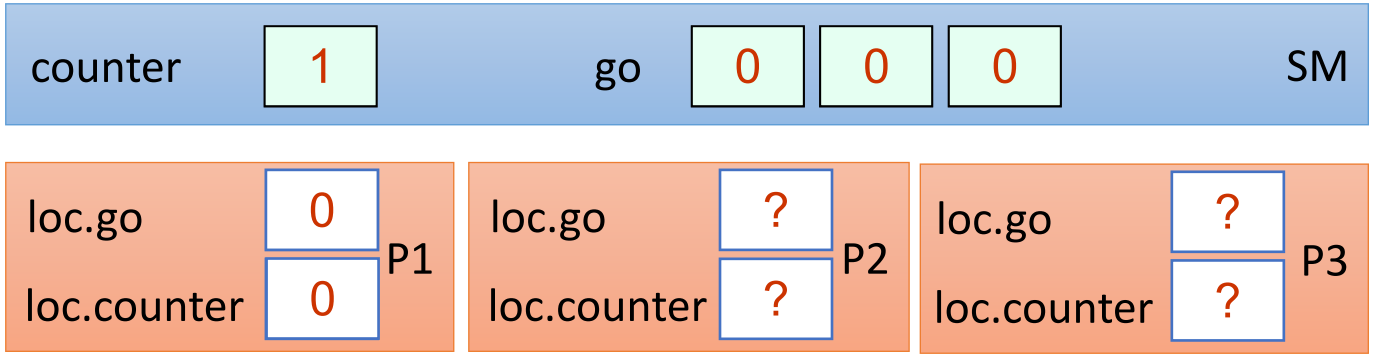
Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads



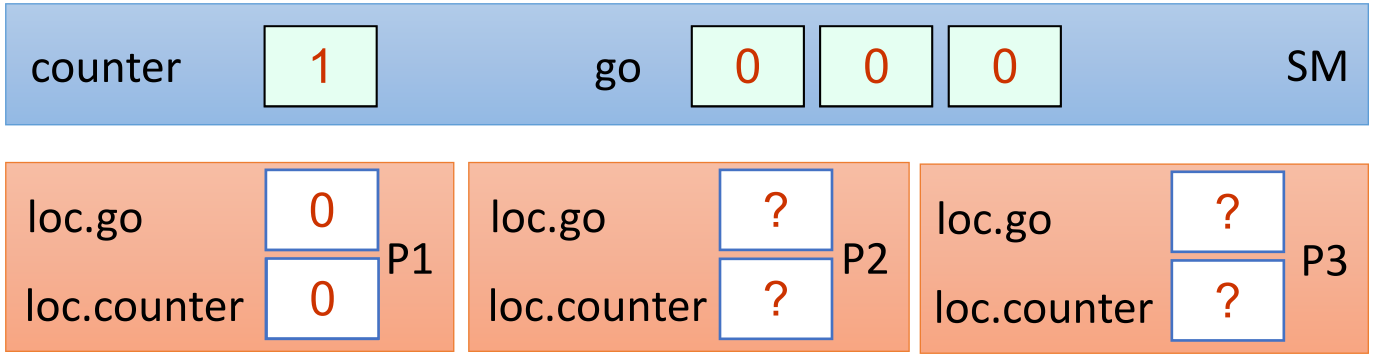

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P1 → (line 6)

P1 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads



P2 →

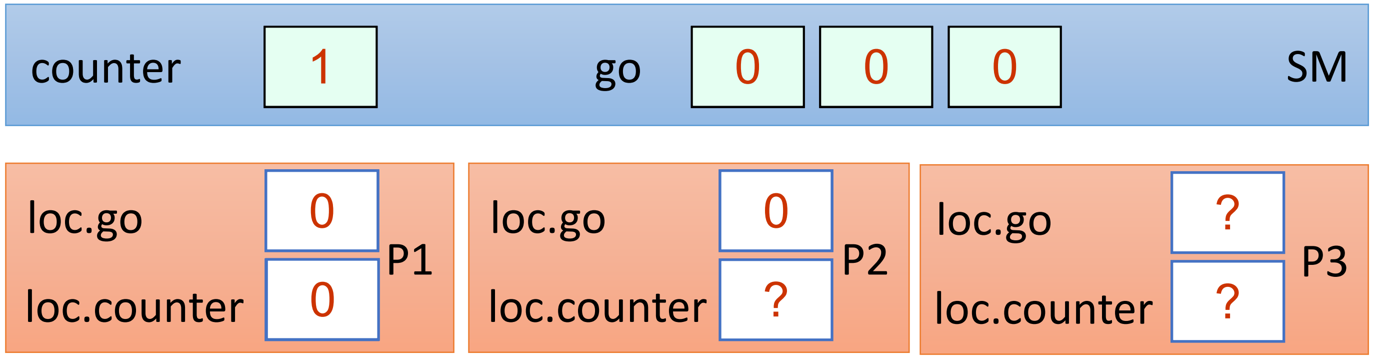
```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P1 →

P1 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

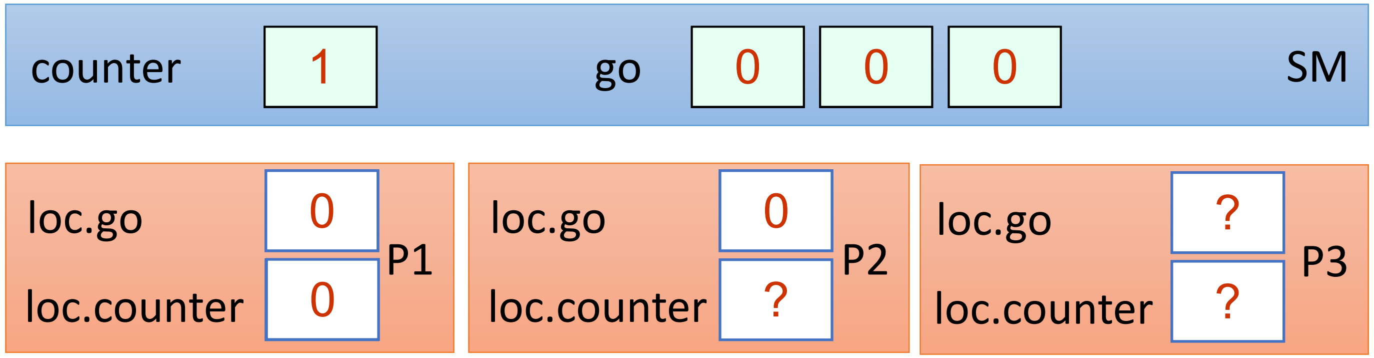
P2 → (line 1)

P1 → (line 6)

P1 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads

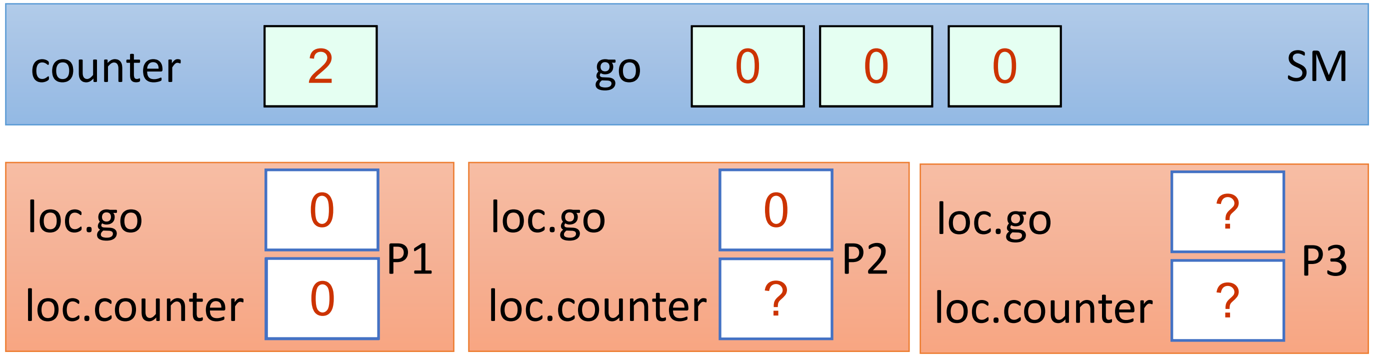



```
local.go := go[i]
local.counter := fetch-and-increment (counter)      ← P2
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])                         ← P1
```

P1 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads

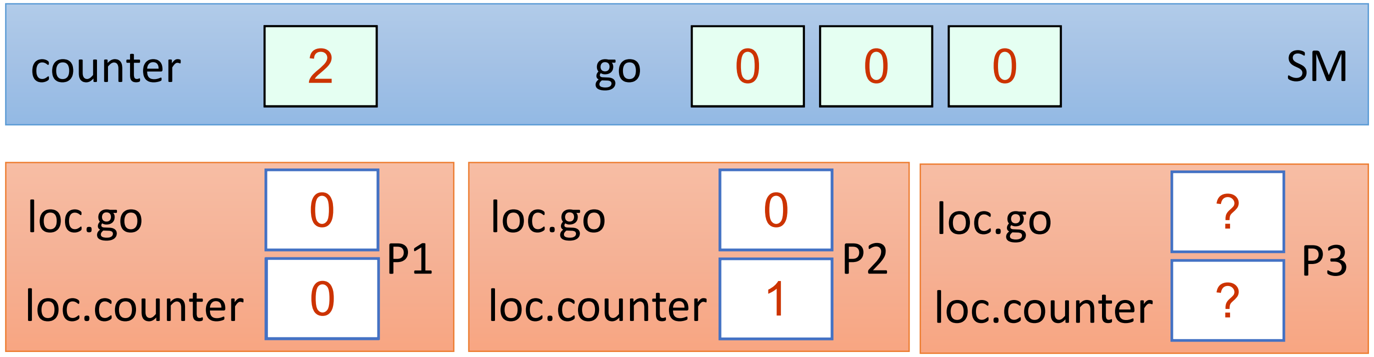



```
local.go := go[i]
local.counter := fetch-and-increment (counter)      <- P2
if local.counter + 1 = n then
    counter := 0
    for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])                        <- P1
```

P1 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads

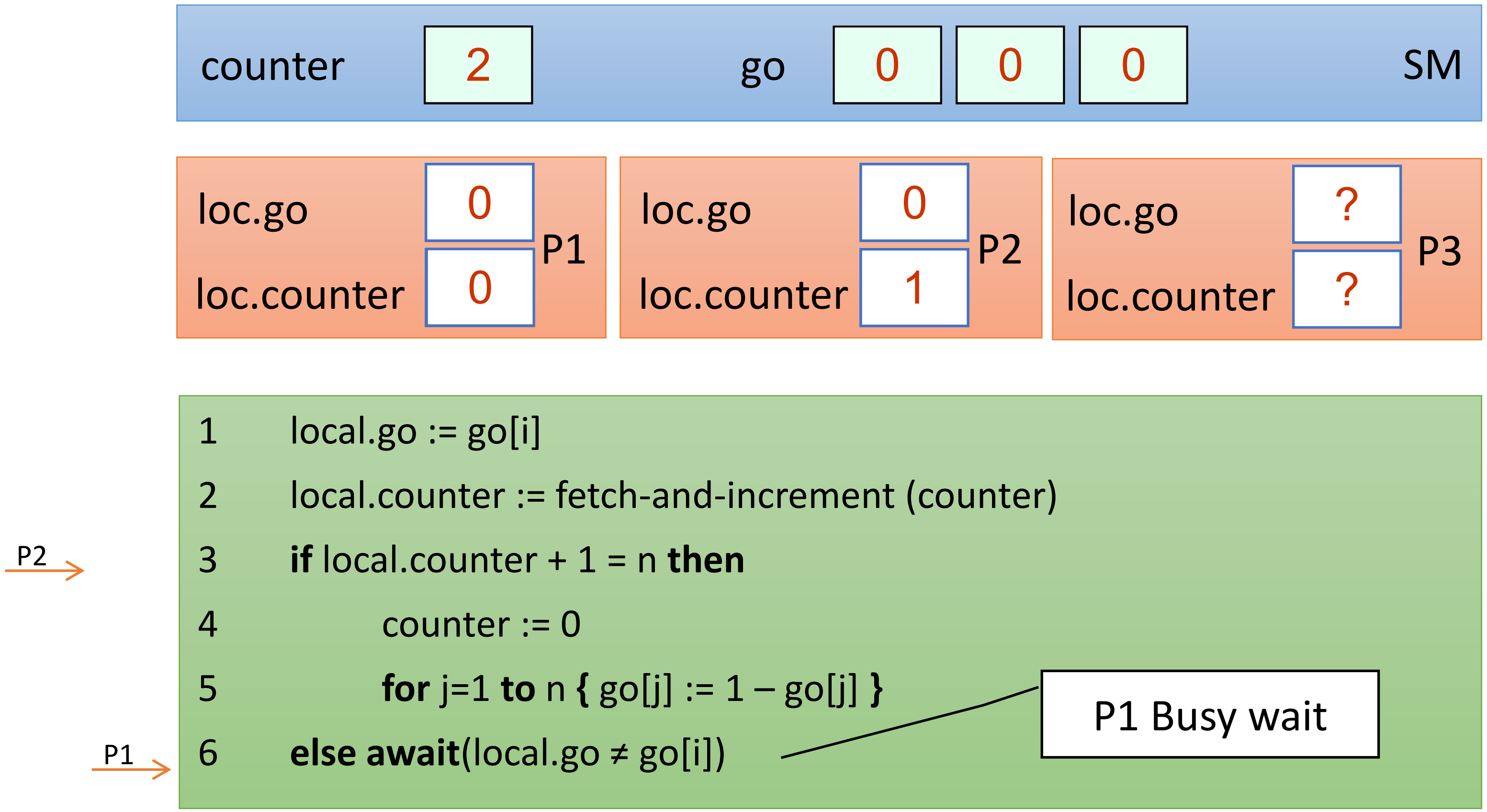



```
local.go := go[i]
local.counter := fetch-and-increment (counter)        <- P2
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])                           <- P1
```

P1 Busy wait

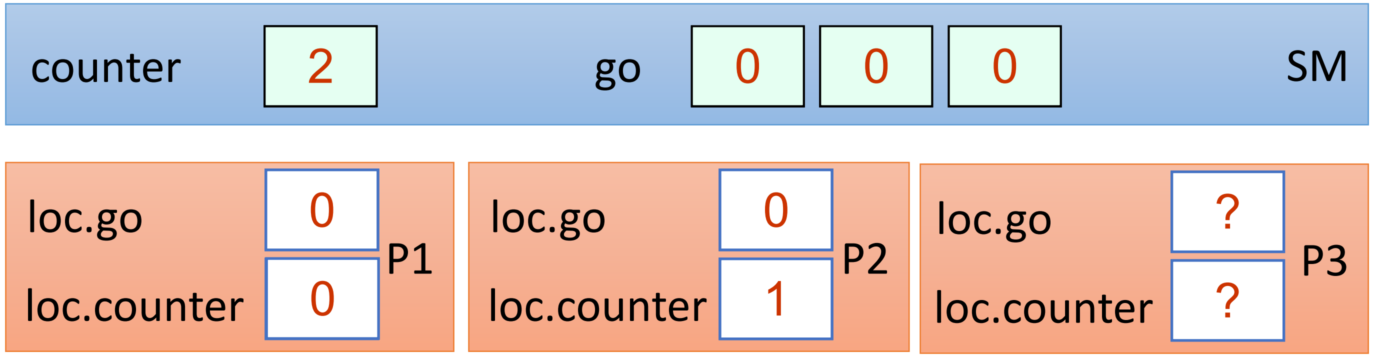
A Local Spinning Counter Barrier

Example Run for n=3 Threads



A Local Spinning Counter Barrier

Example Run for n=3 Threads



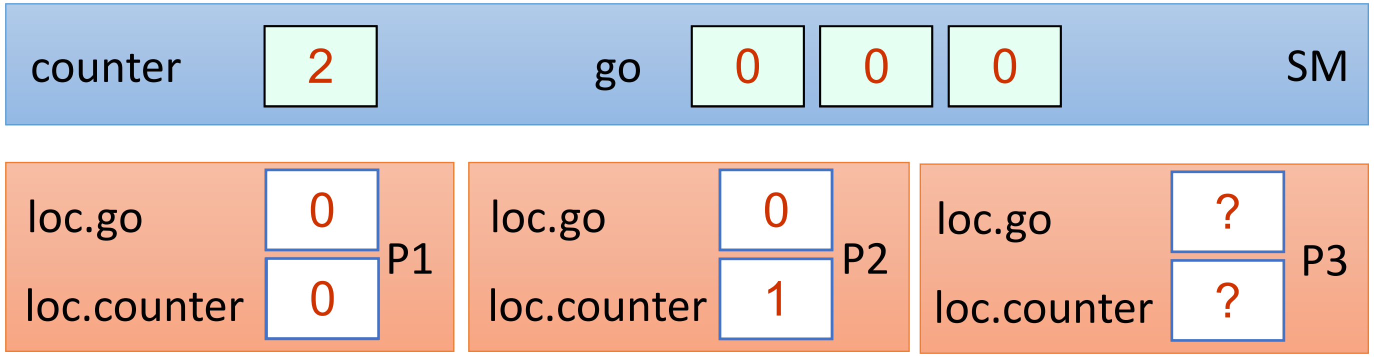

```
local.go := go[i]
local.counter := fetch-and-increment(counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P2 → line 3: 1+1≠3

P1 → line 6: P1 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads



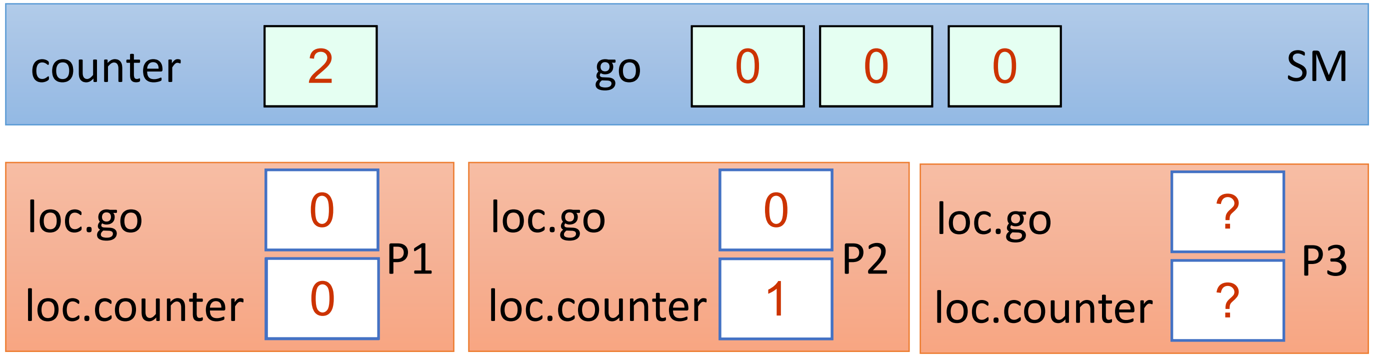

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P2 → P1 → (at line 6)

P1,P2 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

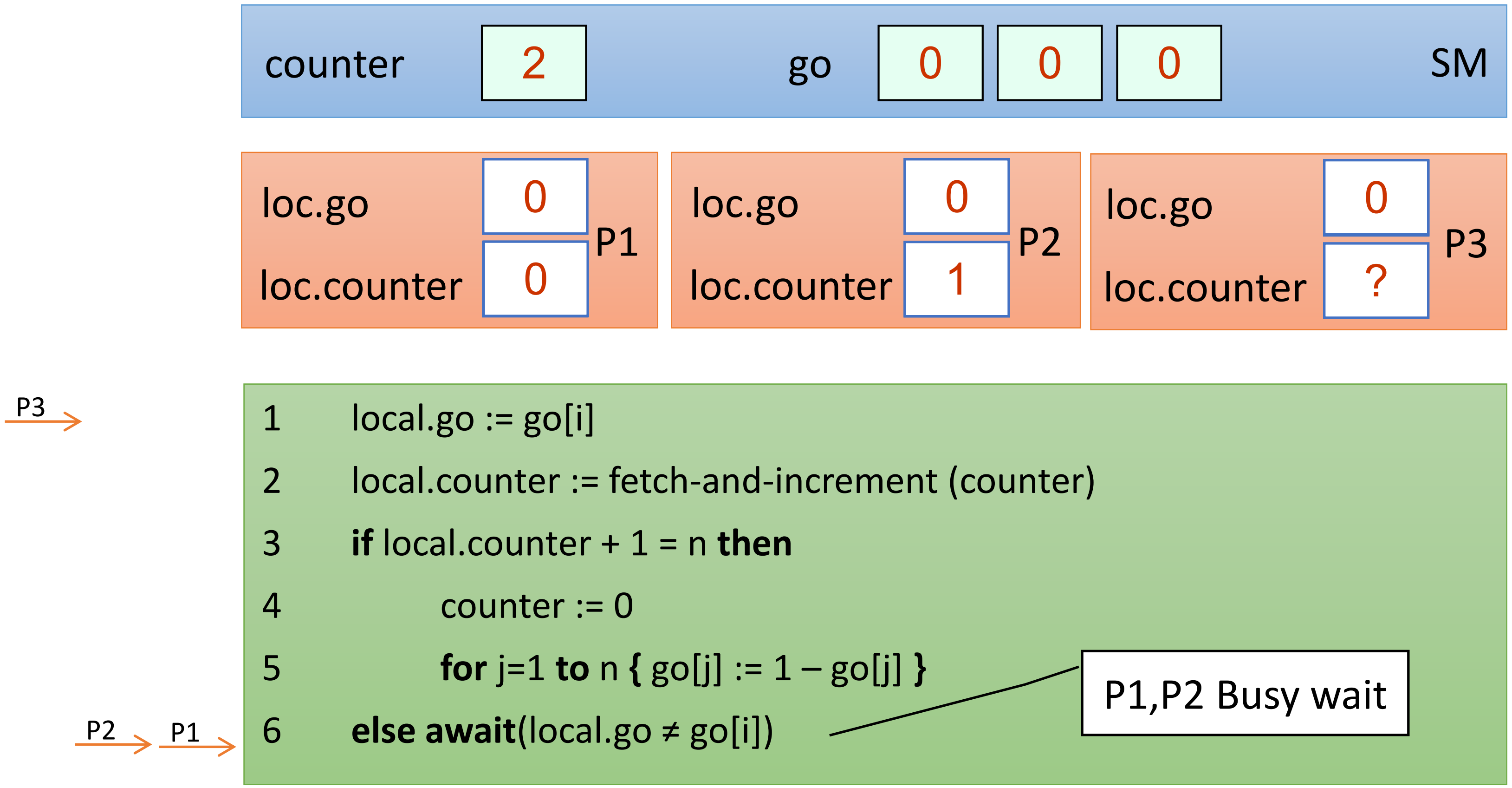
P3 → (at line 1)

P2 → P1 → (at line 6)

P1,P2 Busy wait

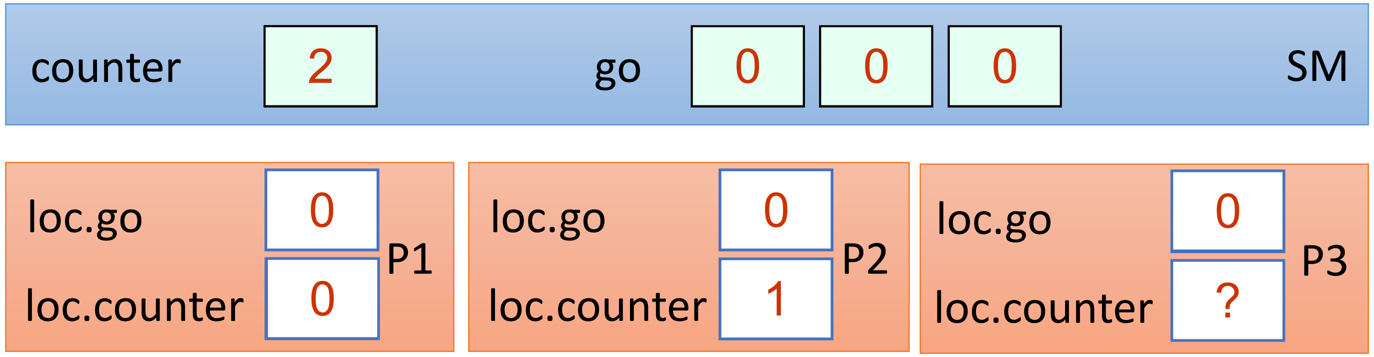
A Local Spinning Counter Barrier

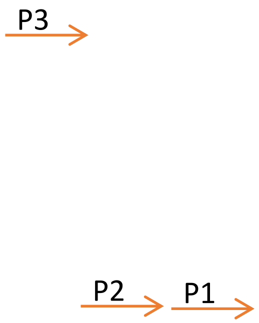
Example Run for n=3 Threads



A Local Spinning Counter Barrier

Example Run for n=3 Threads



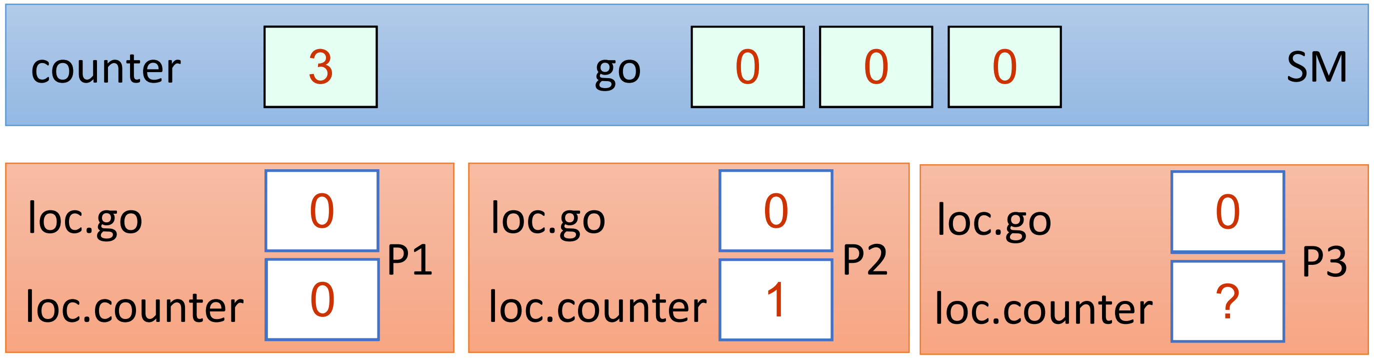



```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P1,P2 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads



P3

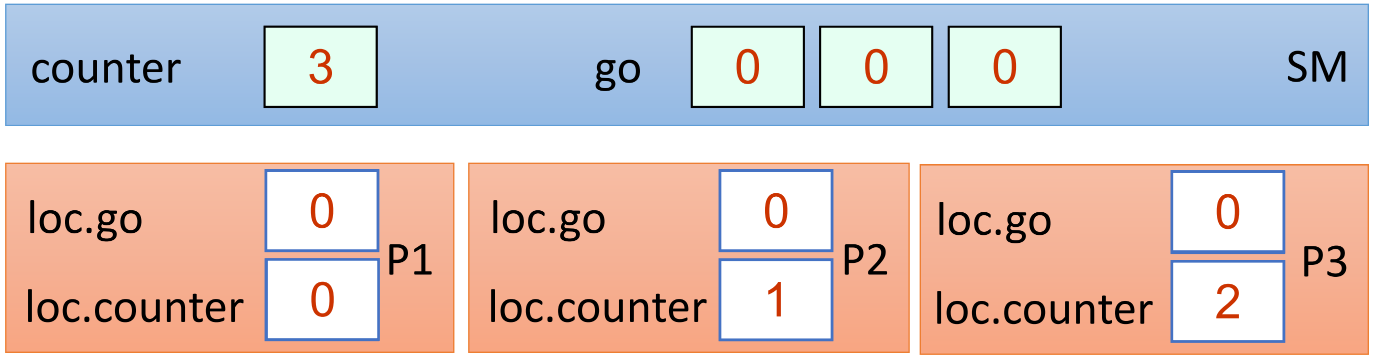
P2 P1

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P1,P2 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads



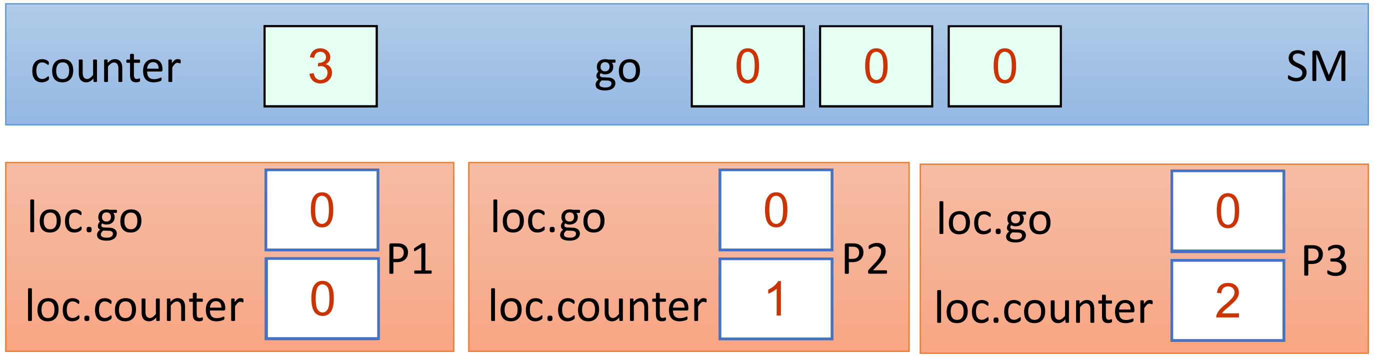

```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

P3 → (line 2); P2 → P1 → (line 6)

P1,P2 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads

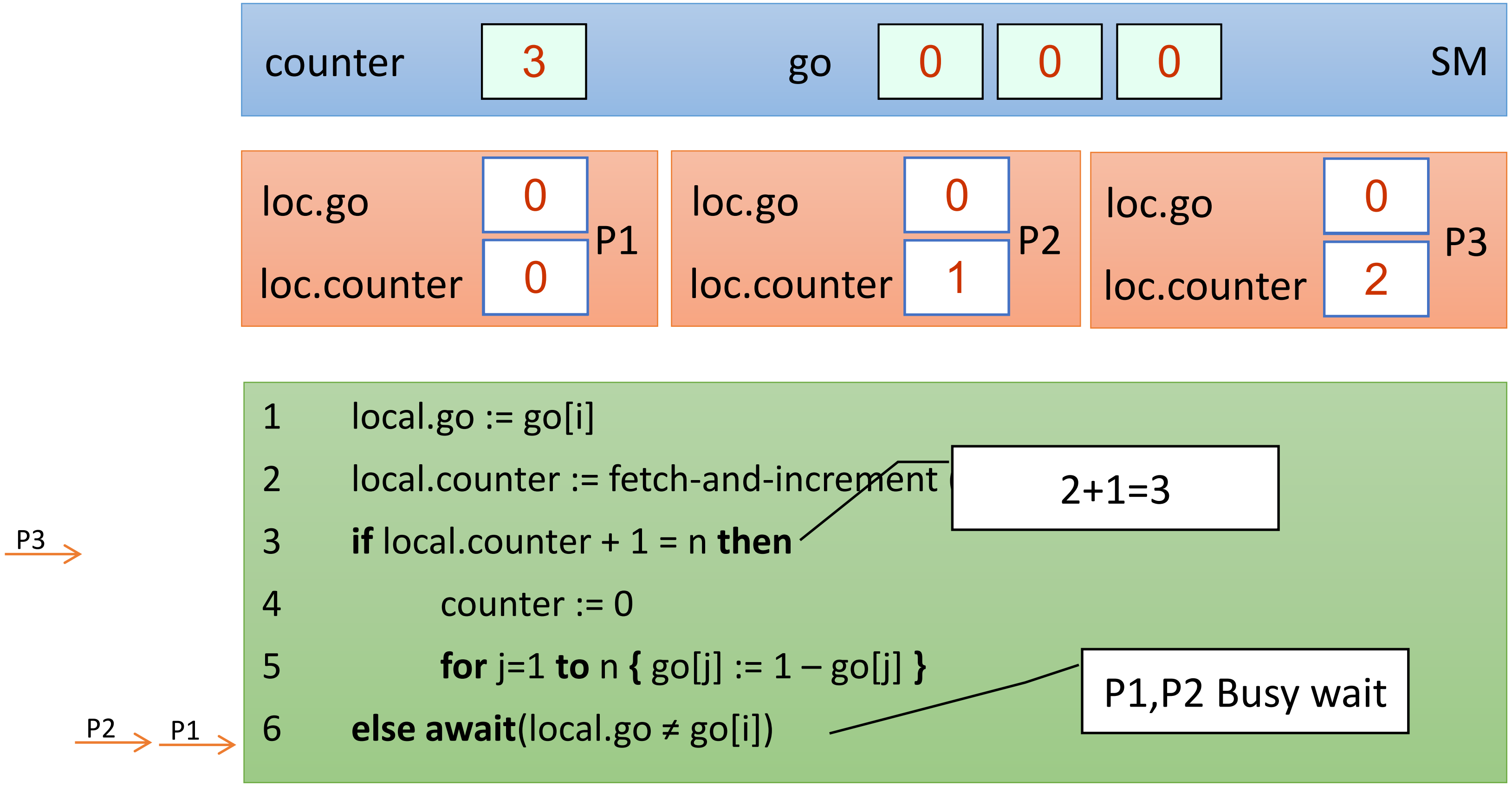



```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then          <- P3
    counter := 0
    for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])           <- P2, P1
```

P1,P2 Busy wait

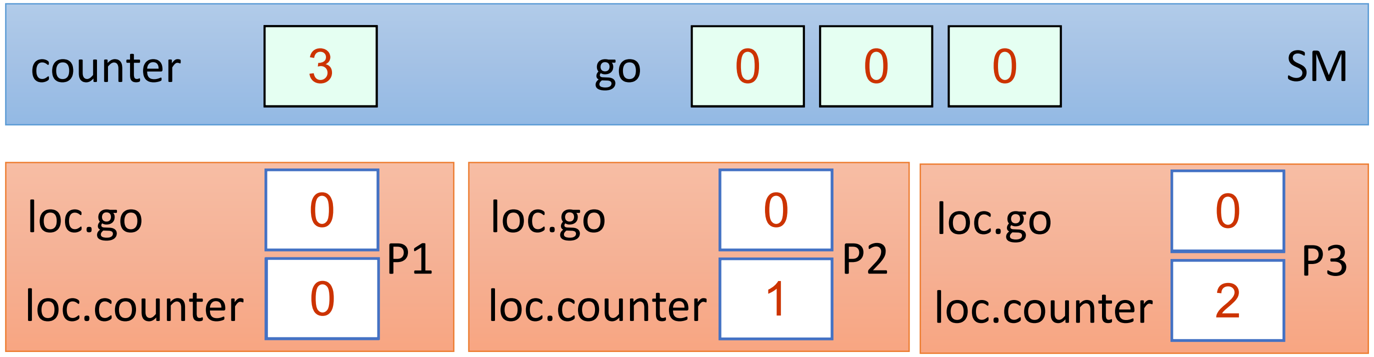
A Local Spinning Counter Barrier

Example Run for n=3 Threads



A Local Spinning Counter Barrier

Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

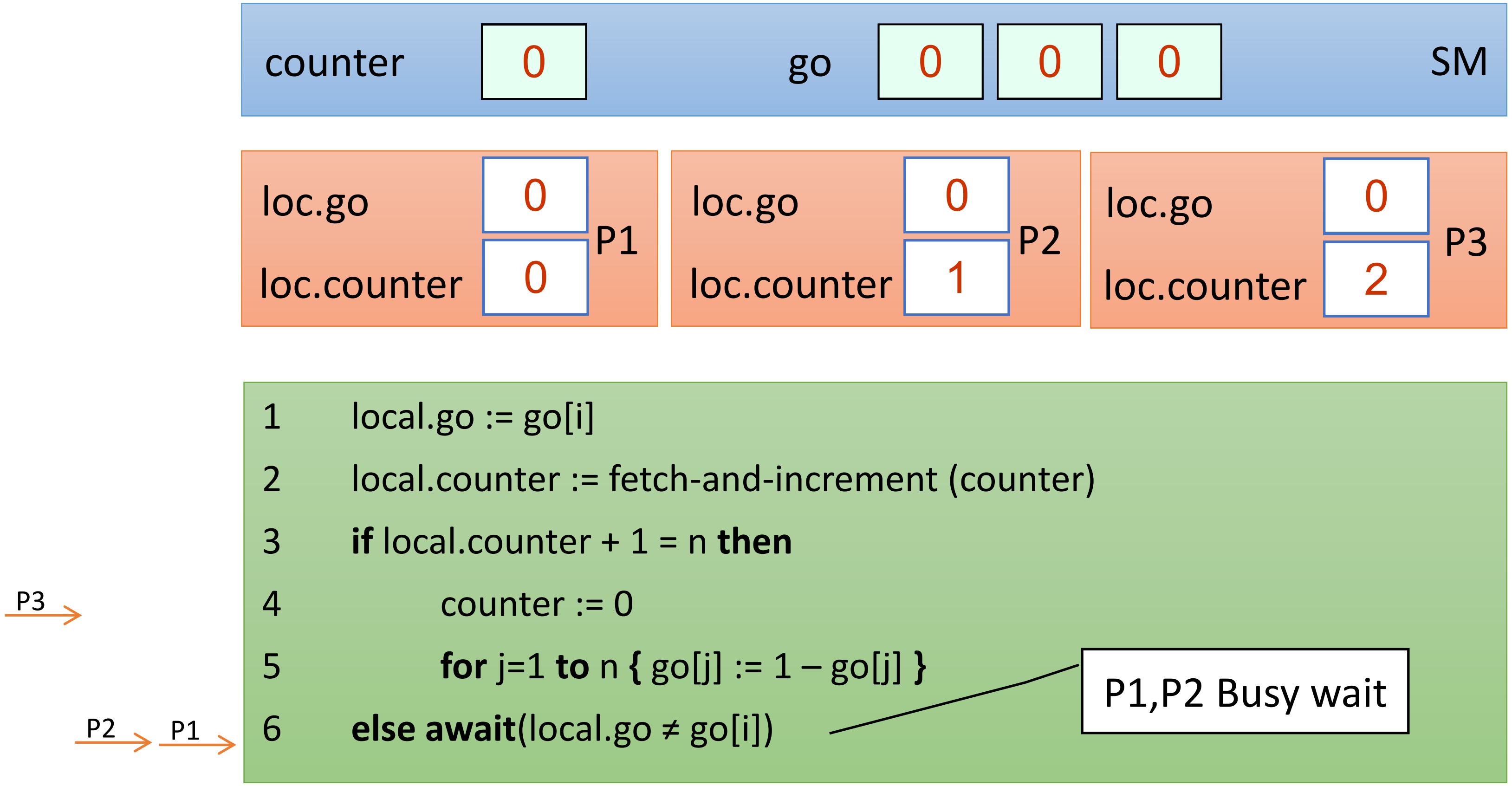
P3 → (line 4)

P2 → P1 → (line 6)

P1,P2 Busy wait

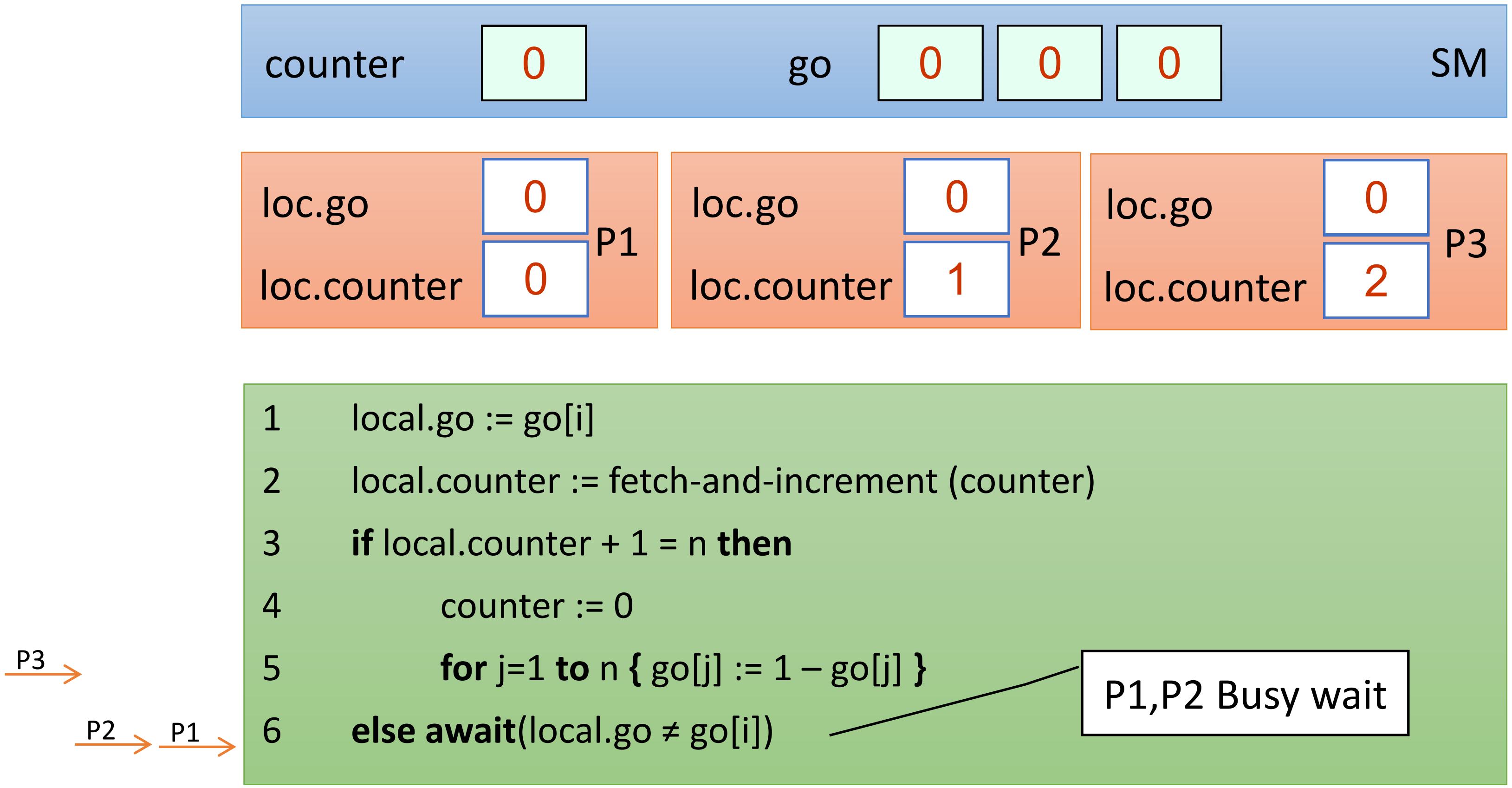
A Local Spinning Counter Barrier

Example Run for n=3 Threads



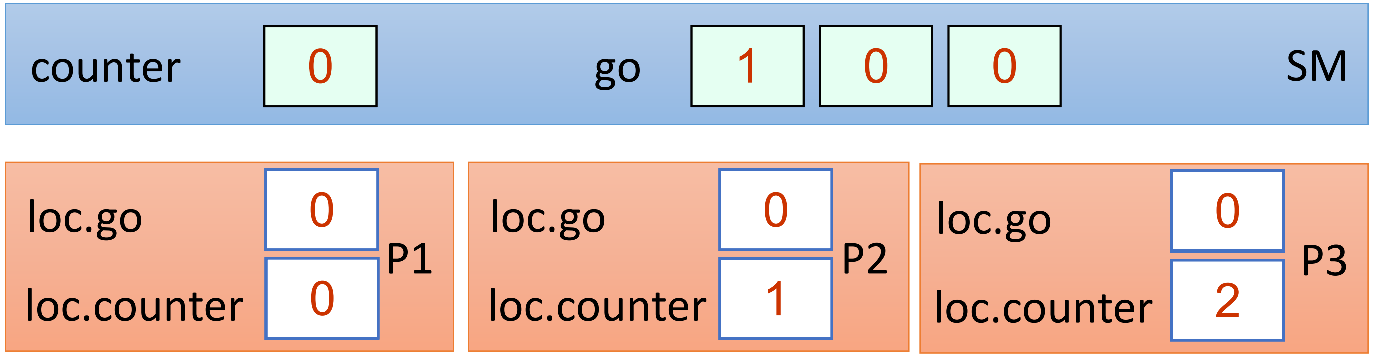
A Local Spinning Counter Barrier

Example Run for n=3 Threads



A Local Spinning Counter Barrier

Example Run for n=3 Threads

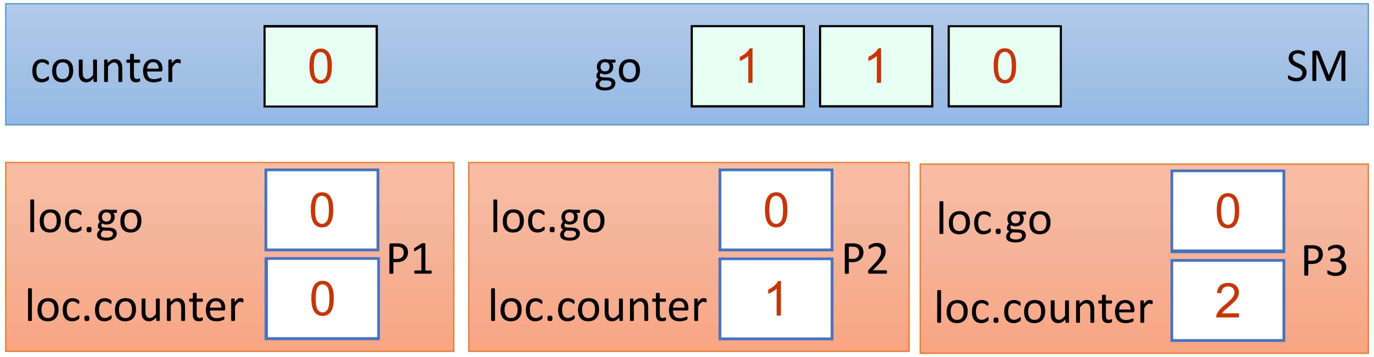



```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }      ← P3
else await(local.go ≠ go[i])                     ← P2, P1
```

P1,P2 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads




```
1    local.go := go[i]
2    local.counter := fetch-and-increment (counter)
3    if local.counter + 1 = n then
4         counter := 0
5         for j=1 to n { go[j] := 1 – go[j] }
6    else await(local.go ≠ go[i])
```

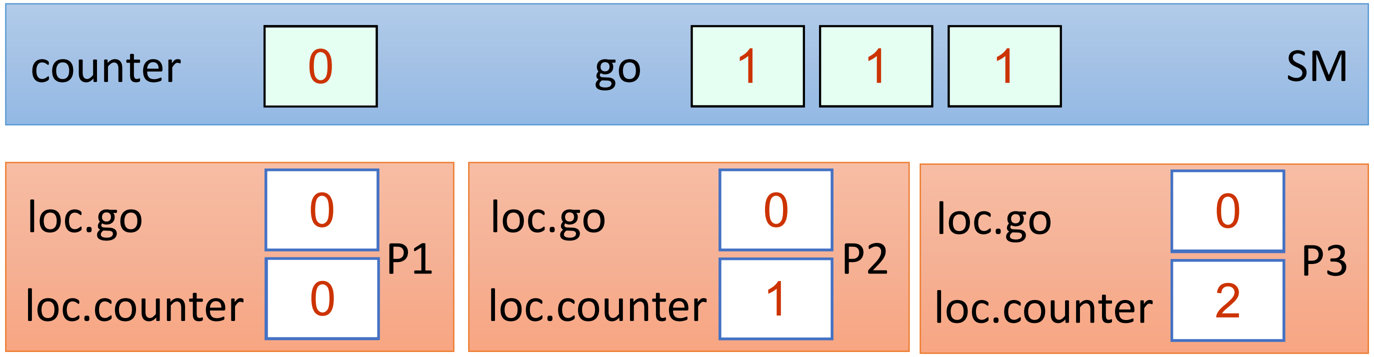
P3 → (line 5)

P2 → P1 → (line 6)

P1,P2 Busy wait

A Local Spinning Counter Barrier

Example Run for n=3 Threads

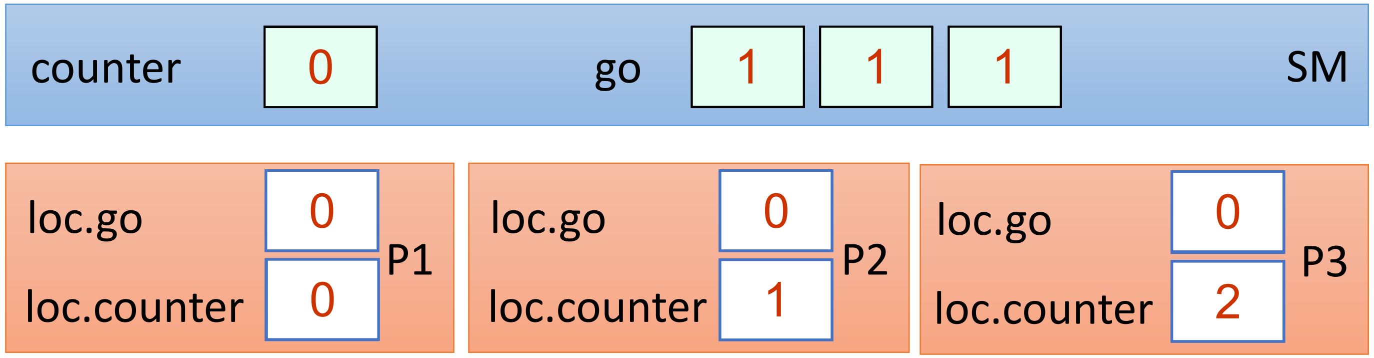



```
      local.go := go[i]
      local.counter := fetch-and-increment (counter)
      if local.counter + 1 = n then
            counter := 0
P3 →        for j=1 to n { go[j] := 1 – go[j] }
P2 → P1 → else await(local.go ≠ go[i])
```

P1,P2 Busy wait

A Local Spinning Counter Barrier

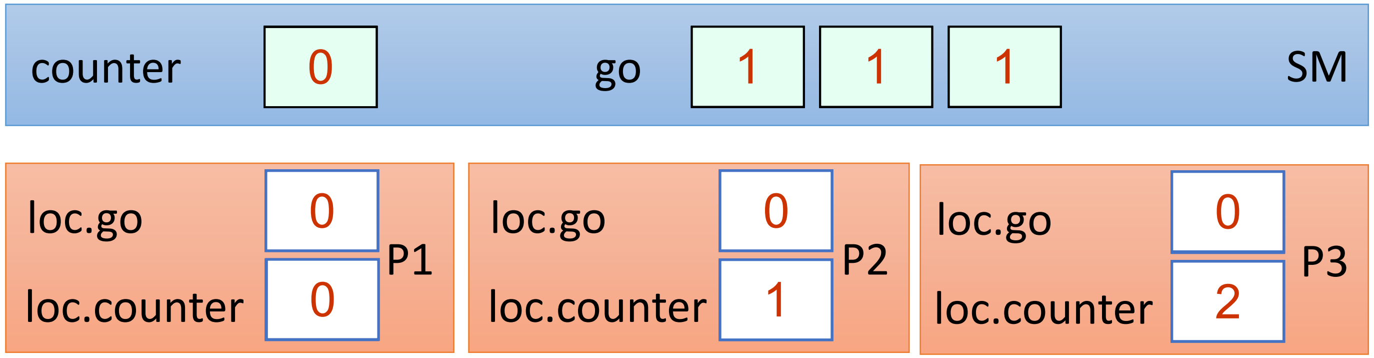
Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

A Local Spinning Counter Barrier

Example Run for n=3 Threads




```
local.go := go[i]
local.counter := fetch-and-increment (counter)
if local.counter + 1 = n then
        counter := 0
        for j=1 to n { go[j] := 1 – go[j] }
else await(local.go ≠ go[i])
```

Pros/Cons?

Does this actually reduce contention?

Comparison of counter-based Barriers

Simple Barrier

- Pros:
- Cons:

Simple Barrier with go array

- Pros:
- Cons:

Comparison of counter-based Barriers

Simple Barrier

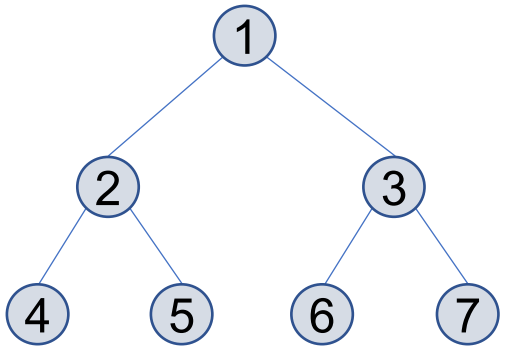
- Pros:
  - Very Simple
  - Shared memory: O(log n) ***bits***
  - Takes O(1) until last waiting p is awaken
- Cons:
  - High contention on the go bit
  - Contention on the counter register (*)

Simple Barrier with go array

- Pros:
  - Low contention on the go array
  - In some models:
    - spinning is done on local memory
    - remote mem. ref.: O(1)
- Cons:
  - Shared memory: O(n)
  - Still contention on the counter register (*)
  - Takes O(n) until last waiting p is awaken

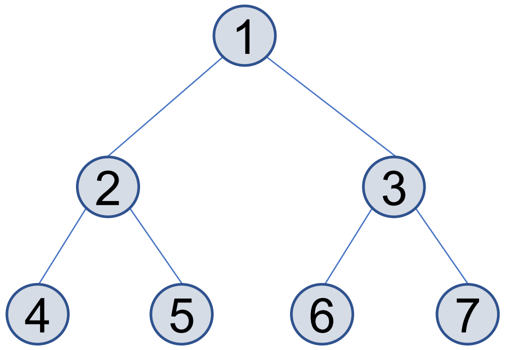
Tree Barriers

A Tree-based Barrier



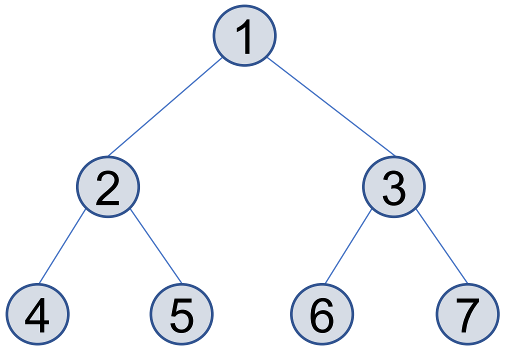
A Tree-based Barrier

- Threads are organized in a binary tree
- Each node is owned by a predetermined thread



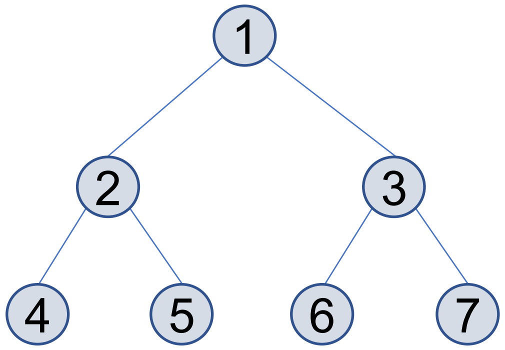
A Tree-based Barrier

- Threads are organized in a binary tree
- Each node is owned by a predetermined thread
- Each thread waits until its 2 children arrive
  - combines results
  - passes them on to its parent

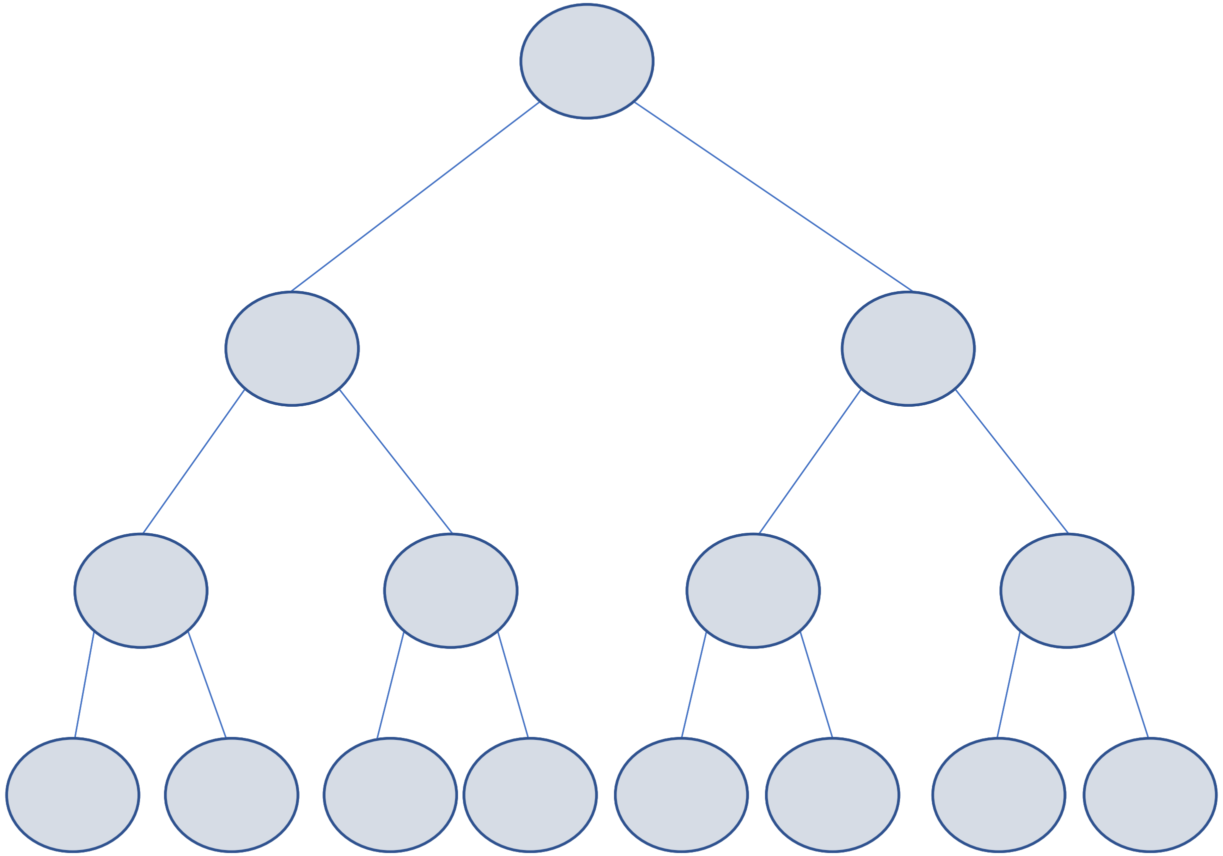


A Tree-based Barrier

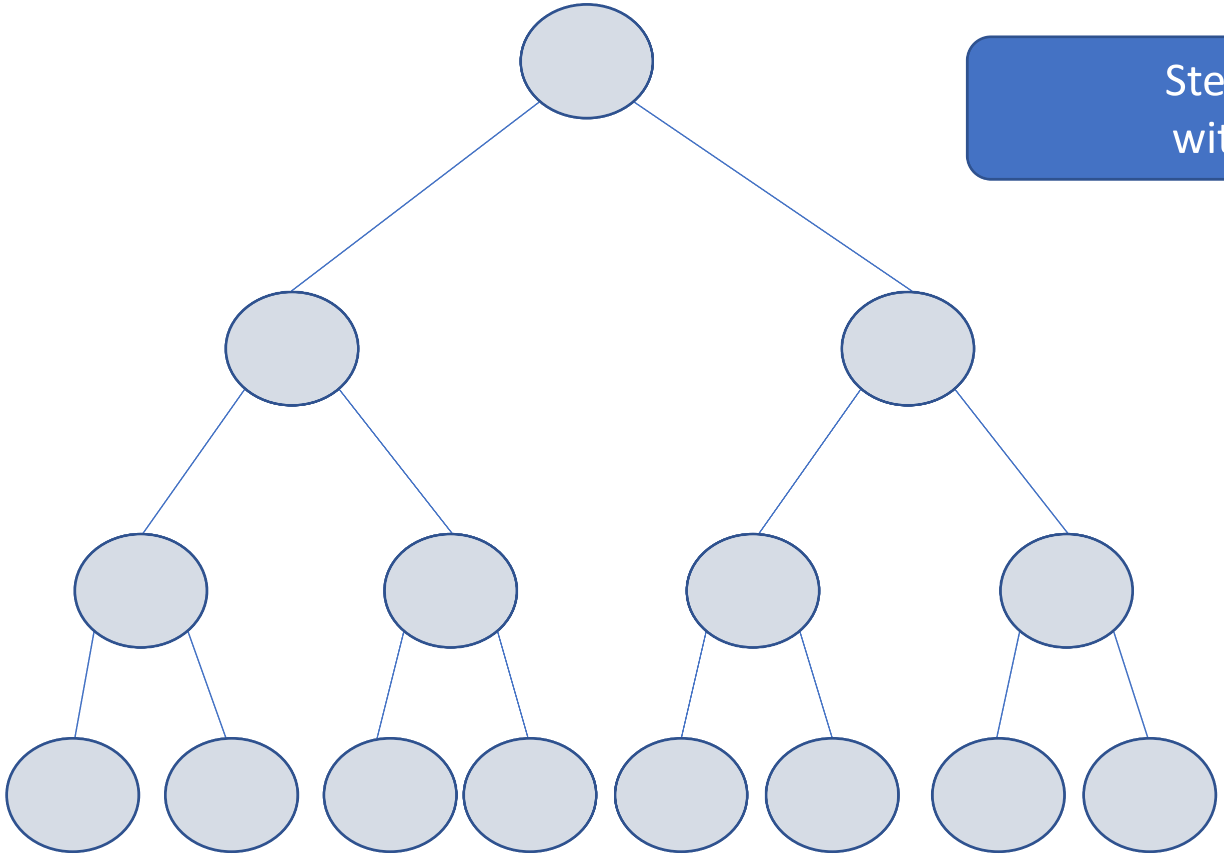
- Threads are organized in a binary tree
- Each node is owned by a predetermined thread
- Each thread waits until its 2 children arrive
  - combines results
  - passes them on to its parent
- Root learns that its 2 children have arrived→tells children they can go
- The signal propagates down the tree until all the threads get the message



A Tree-based Barrier: indexing

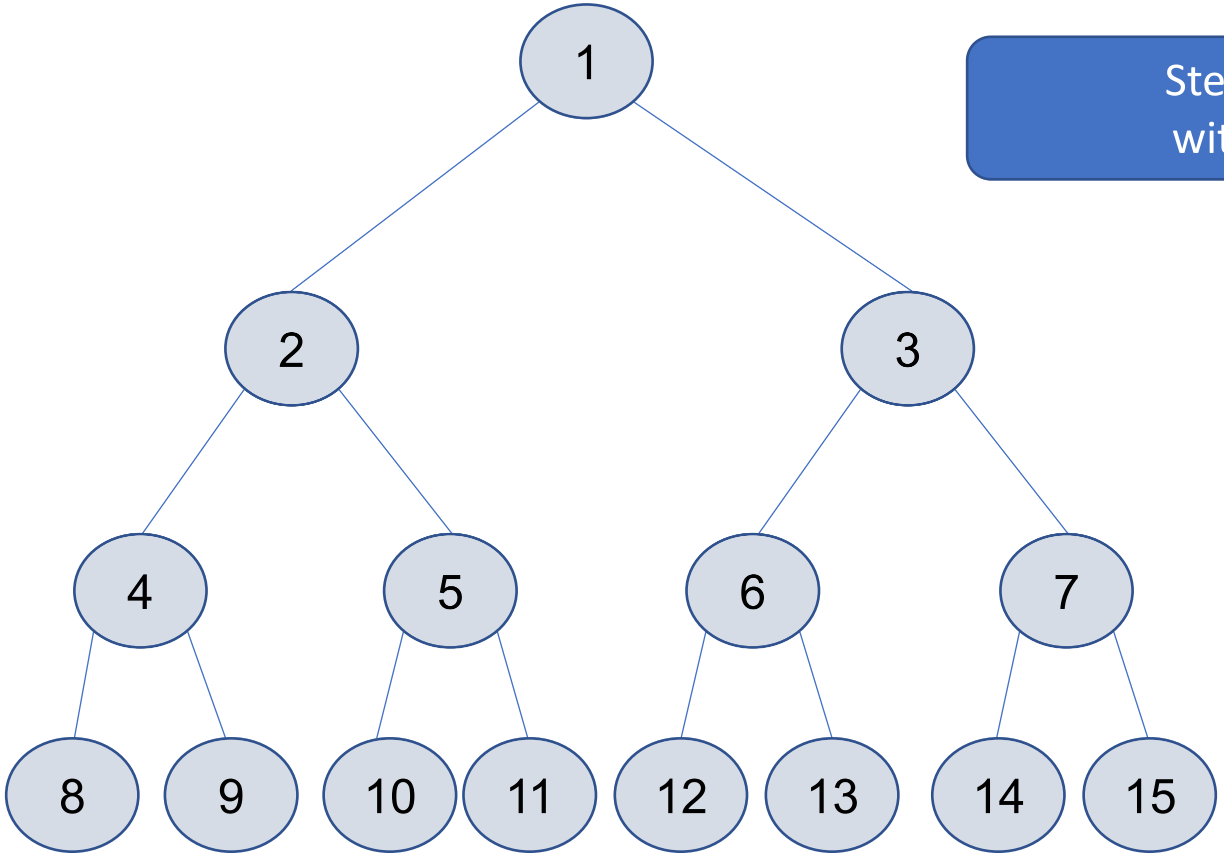


A Tree-based Barrier: indexing



Step 1: label numerically with depth-first traveral

A Tree-based Barrier: indexing

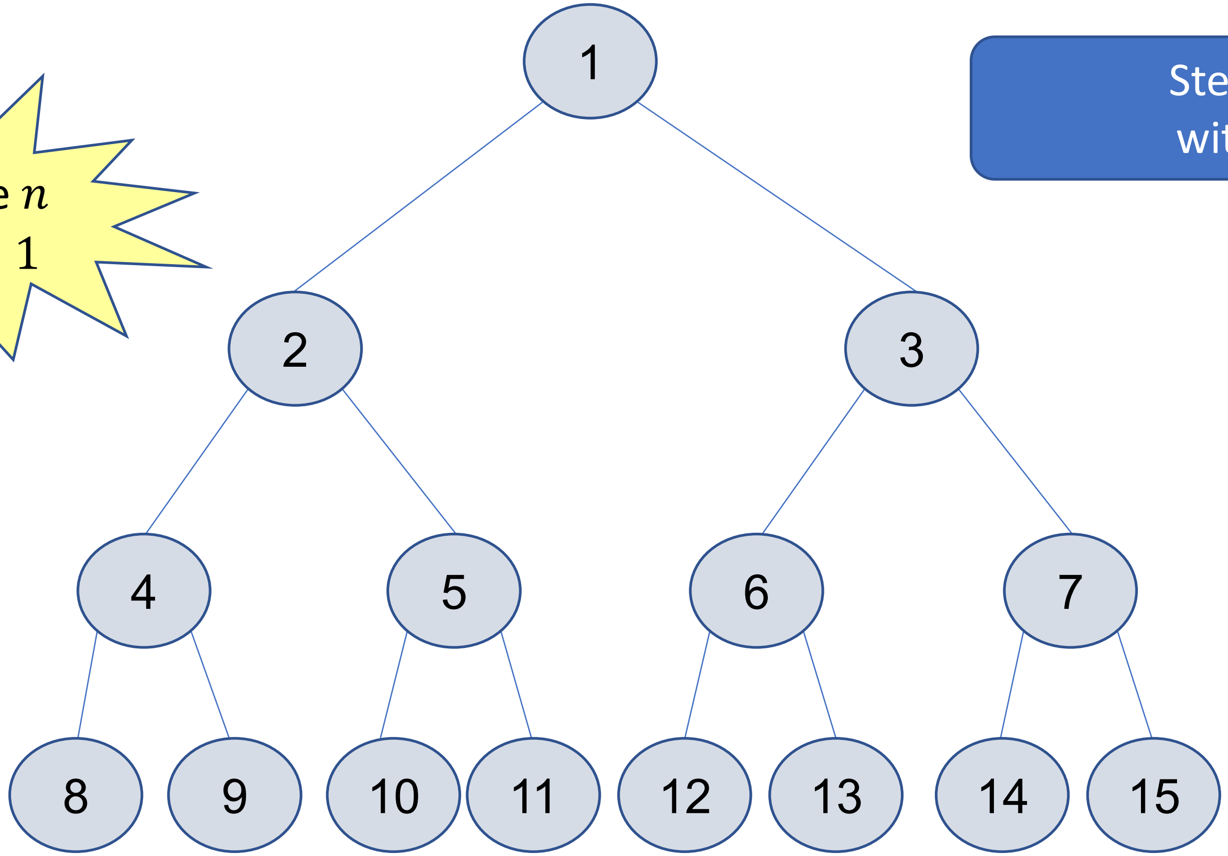


Step 1: label numerically
with depth-first traveral

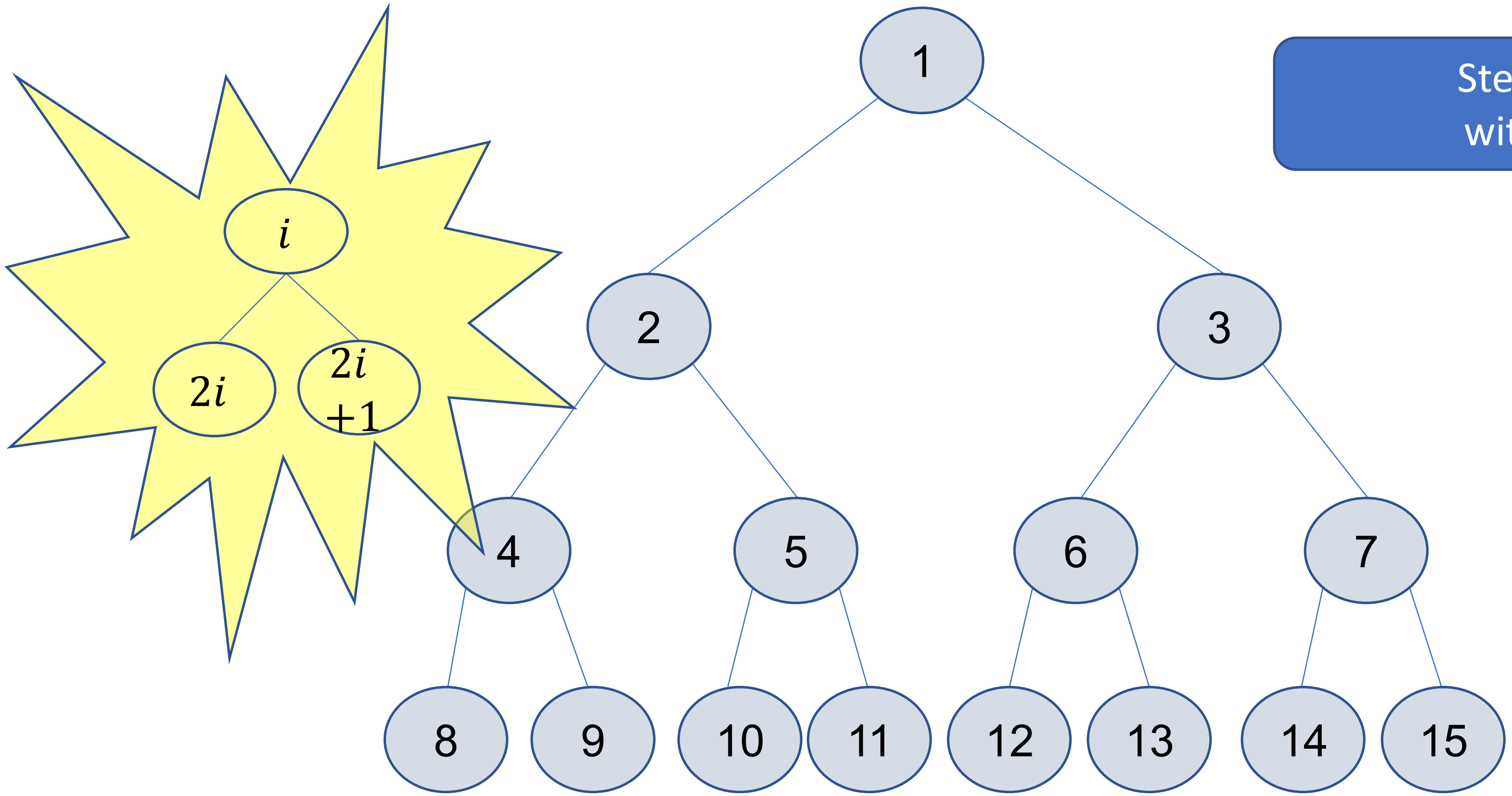
A Tree-based Barrier: indexing

Assume $n = 2^k - 1$

Step 1: label numerically with depth-first traveral



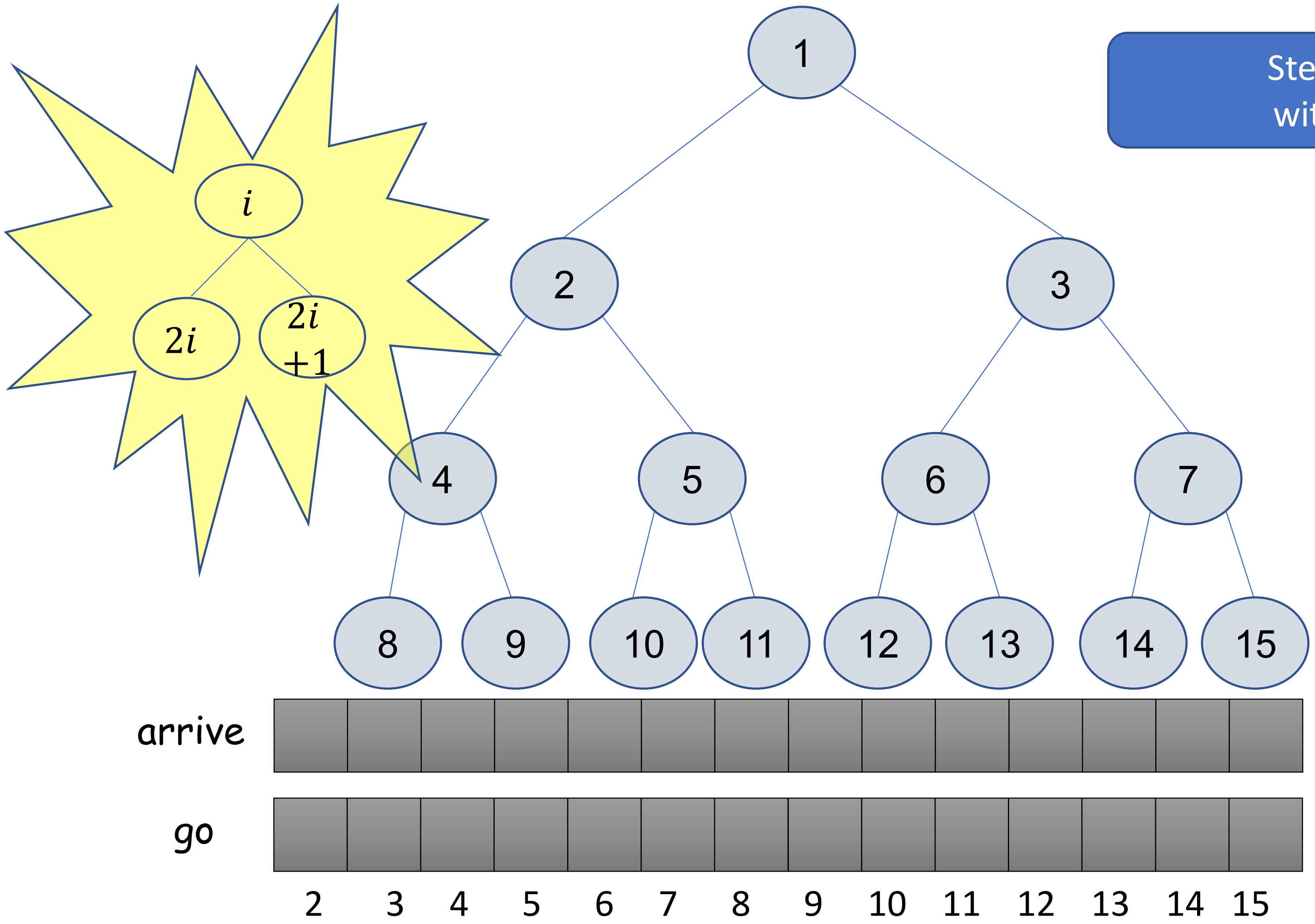
A Tree-based Barrier: indexing



Step 1: label numerically
with depth-first traveral

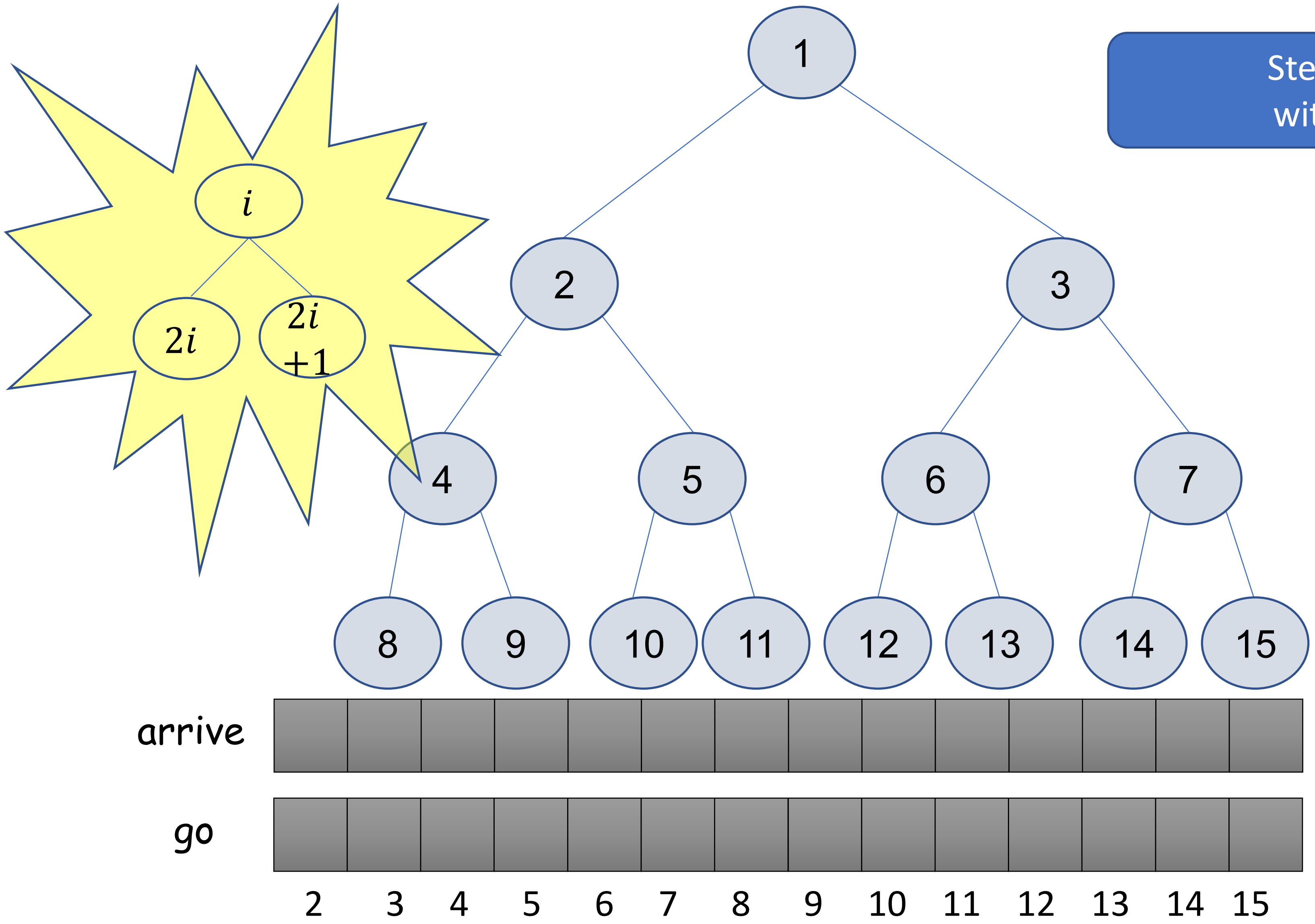
A Tree-based Barrier: indexing

Step 1: label numerically
with depth-first traveral



A Tree-based Barrier: indexing

Step 1: label numerically with depth-first traveral



Indexing starts from 2
Root → 1, doesn't need wait objects

A Tree-based Barrier
program of thread i

```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                          // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
program of thread i

shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

Root

```
if i=1 then                                        // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
```

Internal

```
else if i ≤ (n-1)/2 then                           // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
```

Leaf

```
else                                               // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
program of thread i

shared arrive[2..n]: array of atomic bits, initial values = 0
go[2..n]: array of atomic bits, initial values = 0

Root / Internal / Leaf

```
if i=1 then                                          // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

Root:
- Wait for arriving children
- Tell children to go

A Tree-based Barrier
program of thread i

```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                         // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

Root (lines 1–4)

Internal (lines 5–10)

Leaf (lines 11–14)

Root:
- Wait for arriving children
- Tell children to go

Internal:
- Wait for arriving children
- Tell parent about it
- Wait for parent go signal
- Tell children to go

A Tree-based Barrier

program of thread i

```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

Root:

```
if i=1 then                                          // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
```

Internal:

```
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
```

Leaf:

```
else                                                 // le
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

Root:
- Wait for arriving children
- Tell children to go

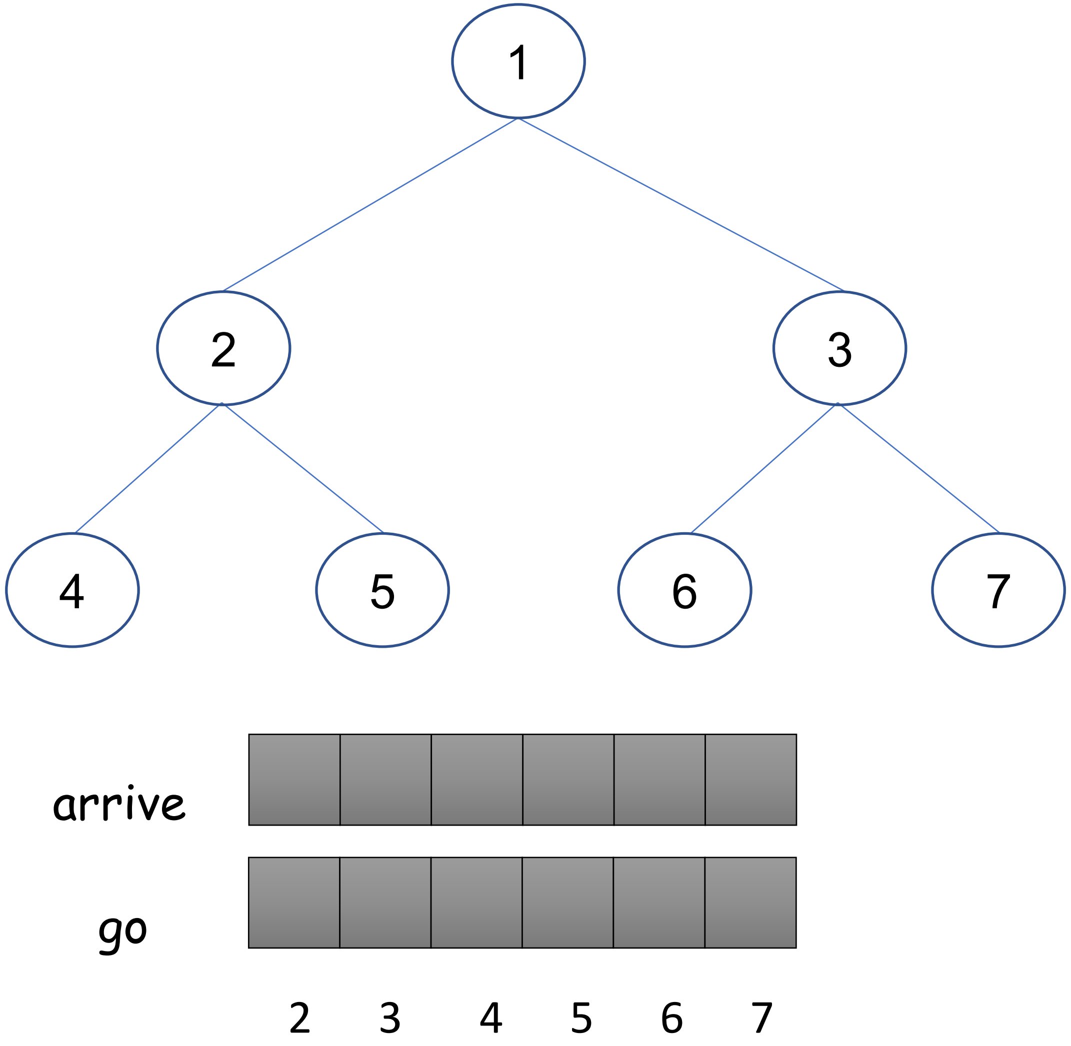
Internal:
- Wait for arriving children
- Tell parent about it
- Wait for parent go signal
- Tell children to go

Child:
- arrive
- Wait for parent go signal

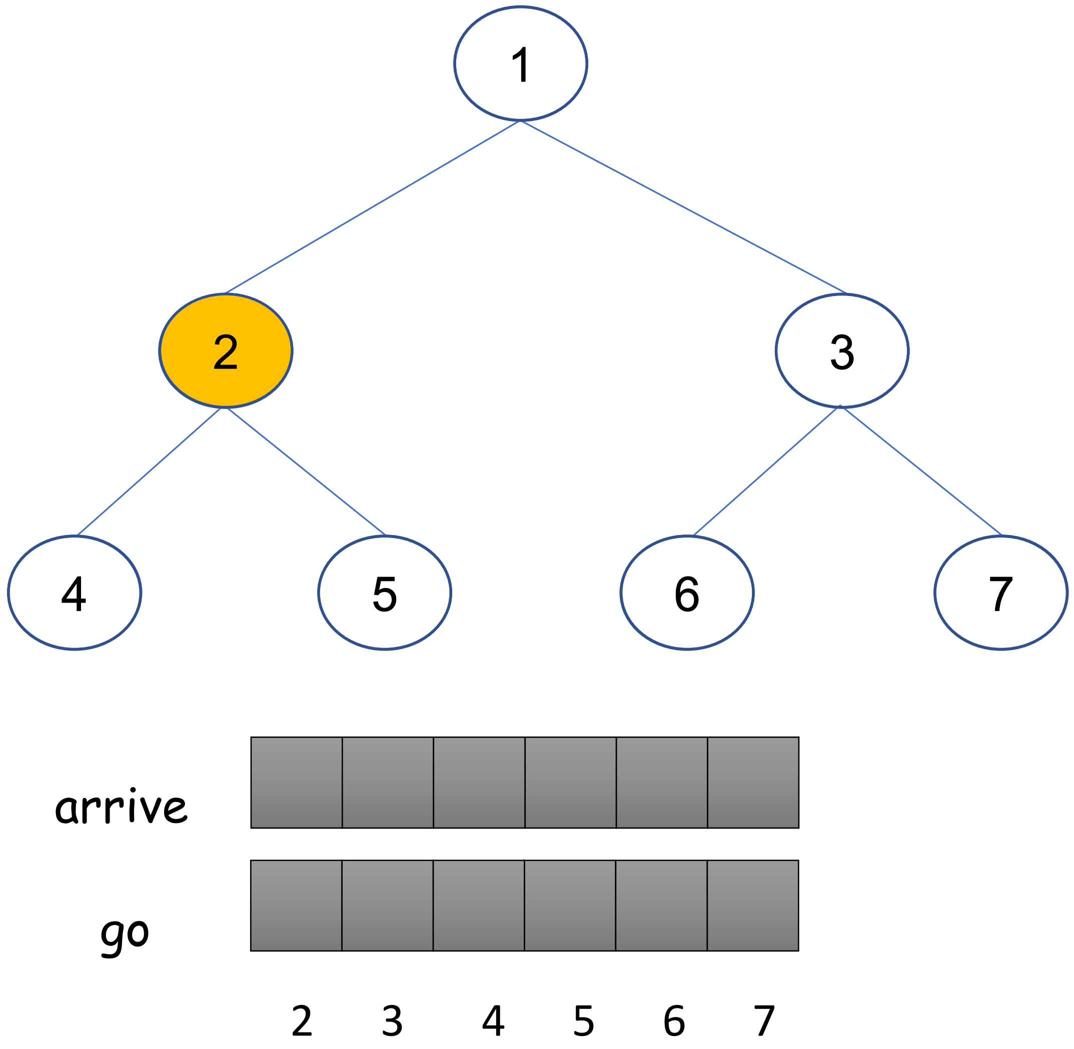
A Tree-based Barrier
Example Run for n=7 threads

```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                         // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```



A Tree-based Barrier
Example Run for n=7 threads

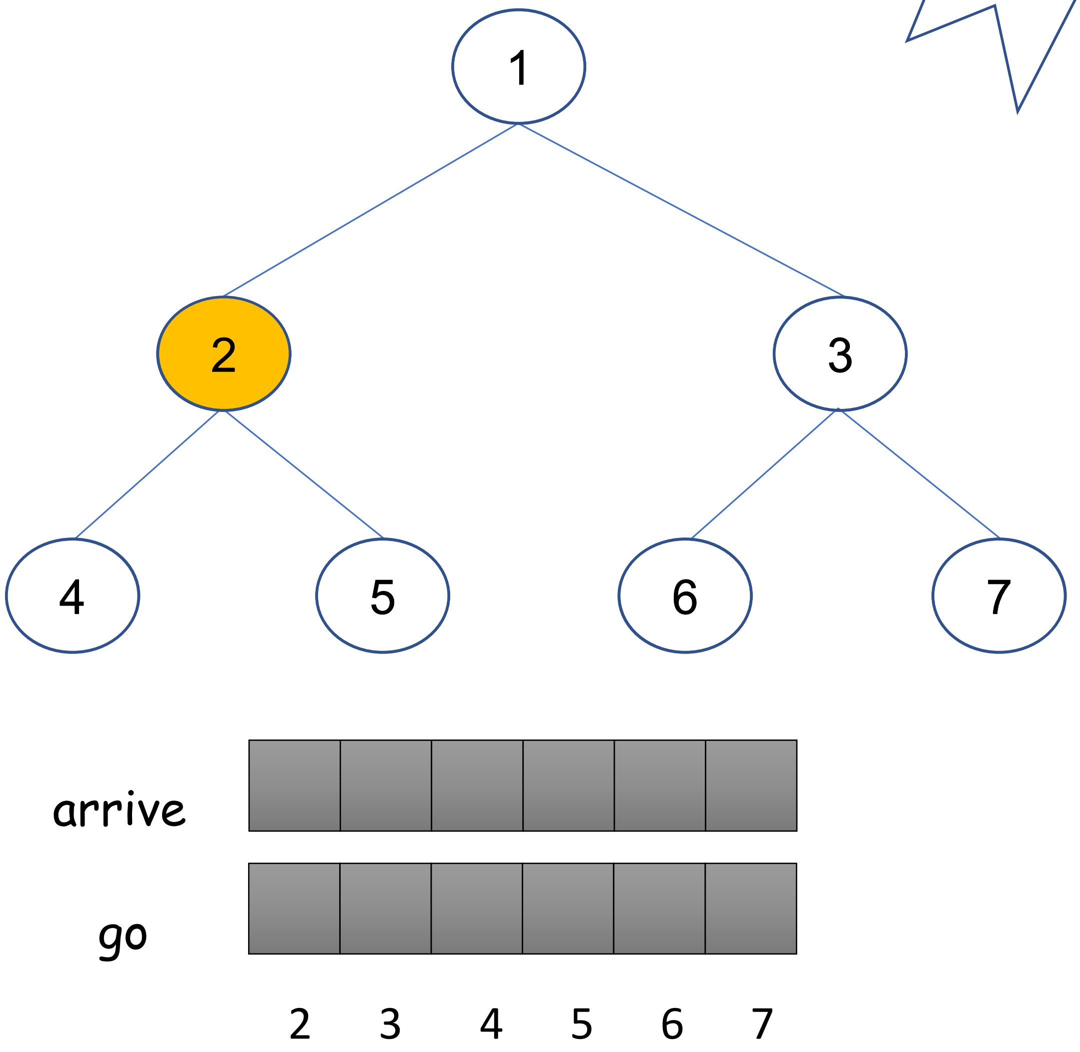


shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                         // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                                // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

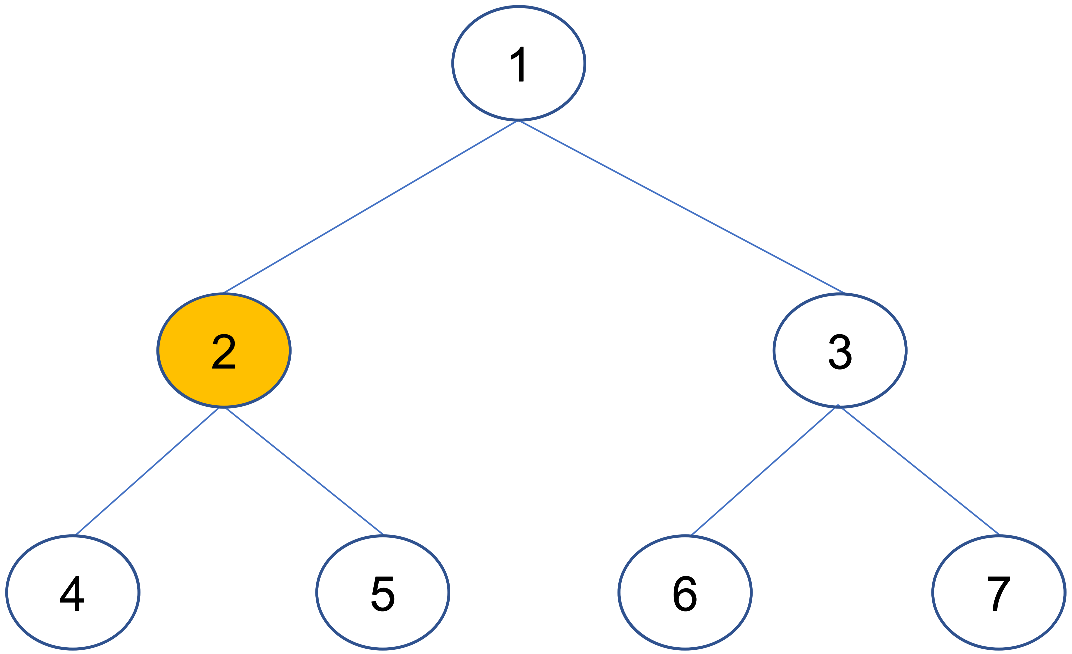
A Tree-based Barrier
Example Run for n=7 threads

arrive[2]=1 ?



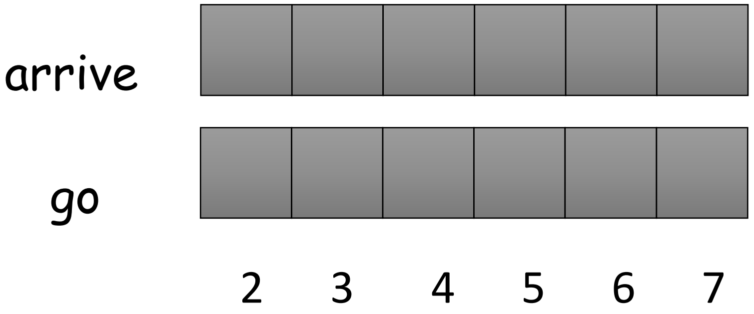

```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                        // root
     await(arrive[2] = 1); arrive[2] := 0
     await(arrive[3] = 1); arrive[3] := 0
     go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
     await(arrive[2i] = 1); arrive[2i] := 0
     await(arrive[2i+1] = 1); arrive[2i+1] := 0
     arrive[i] := 1
     await(go[i] = 1); go[i] := 0
     go[2i] = 1; go[2i+1] := 1
else                                               // leaf
     arrive[i] := 1
     await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

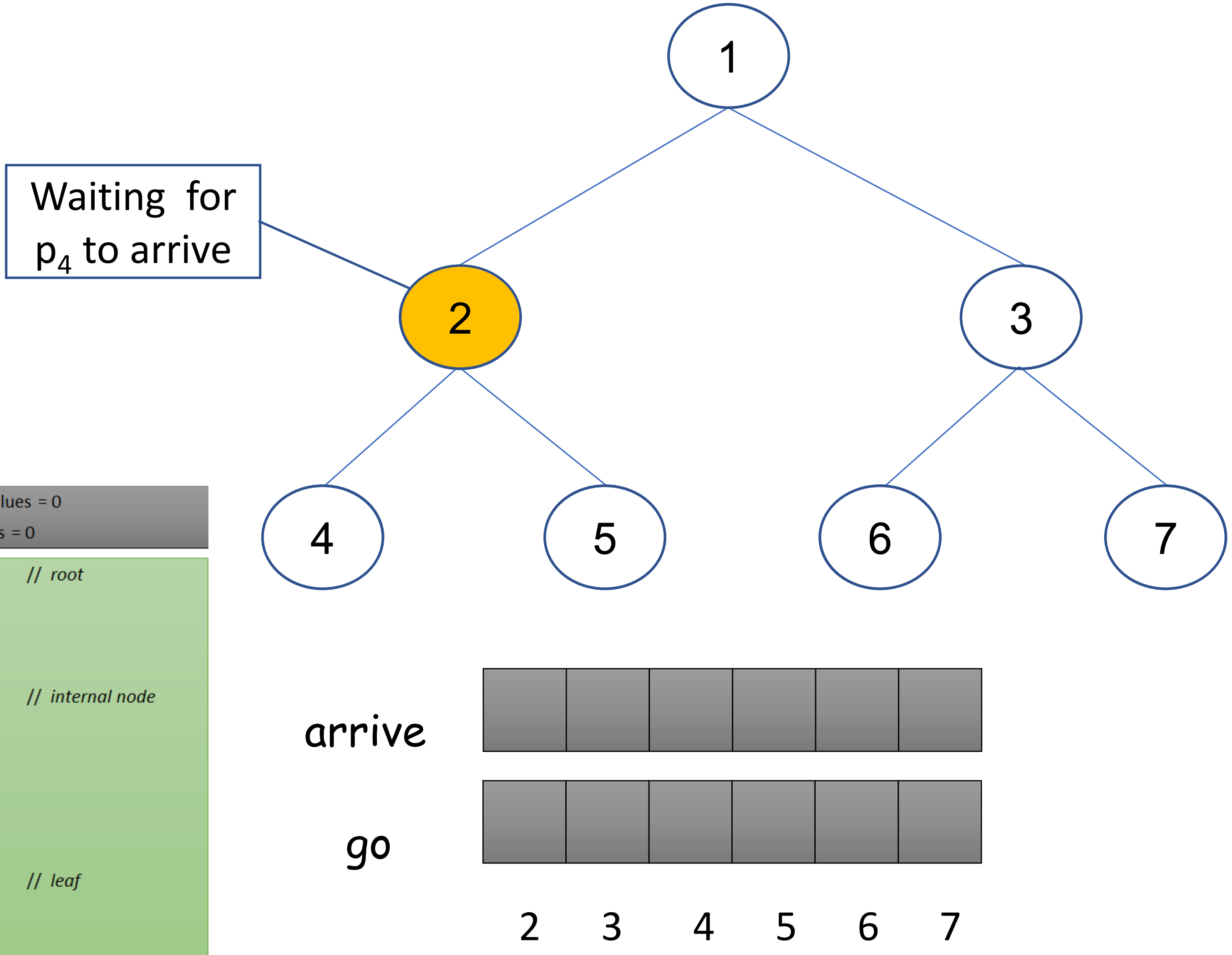



```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                        // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                               // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

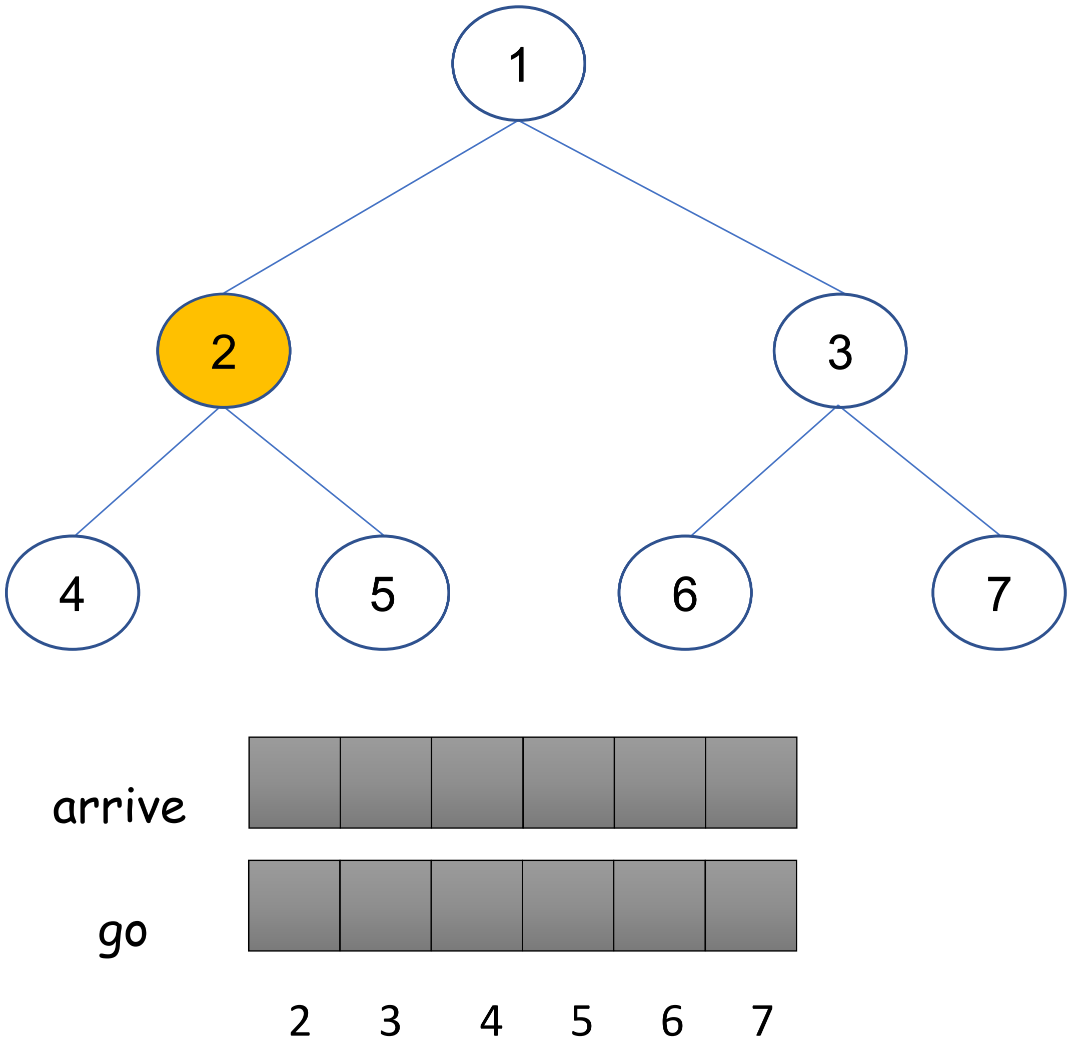


A Tree-based Barrier
Example Run for n=7 threads




```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                         // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                                // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

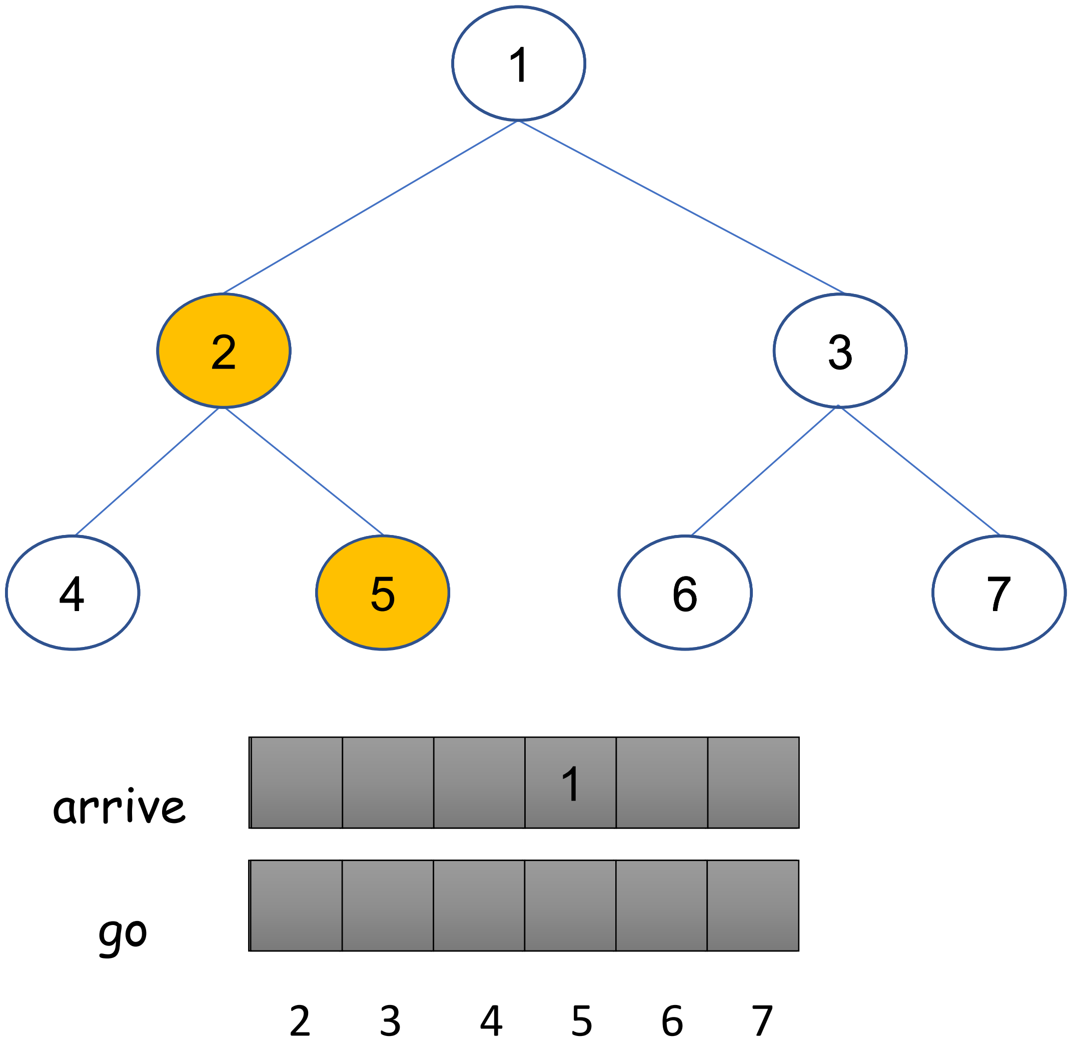
A Tree-based Barrier
Example Run for n=7 threads




```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                          // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

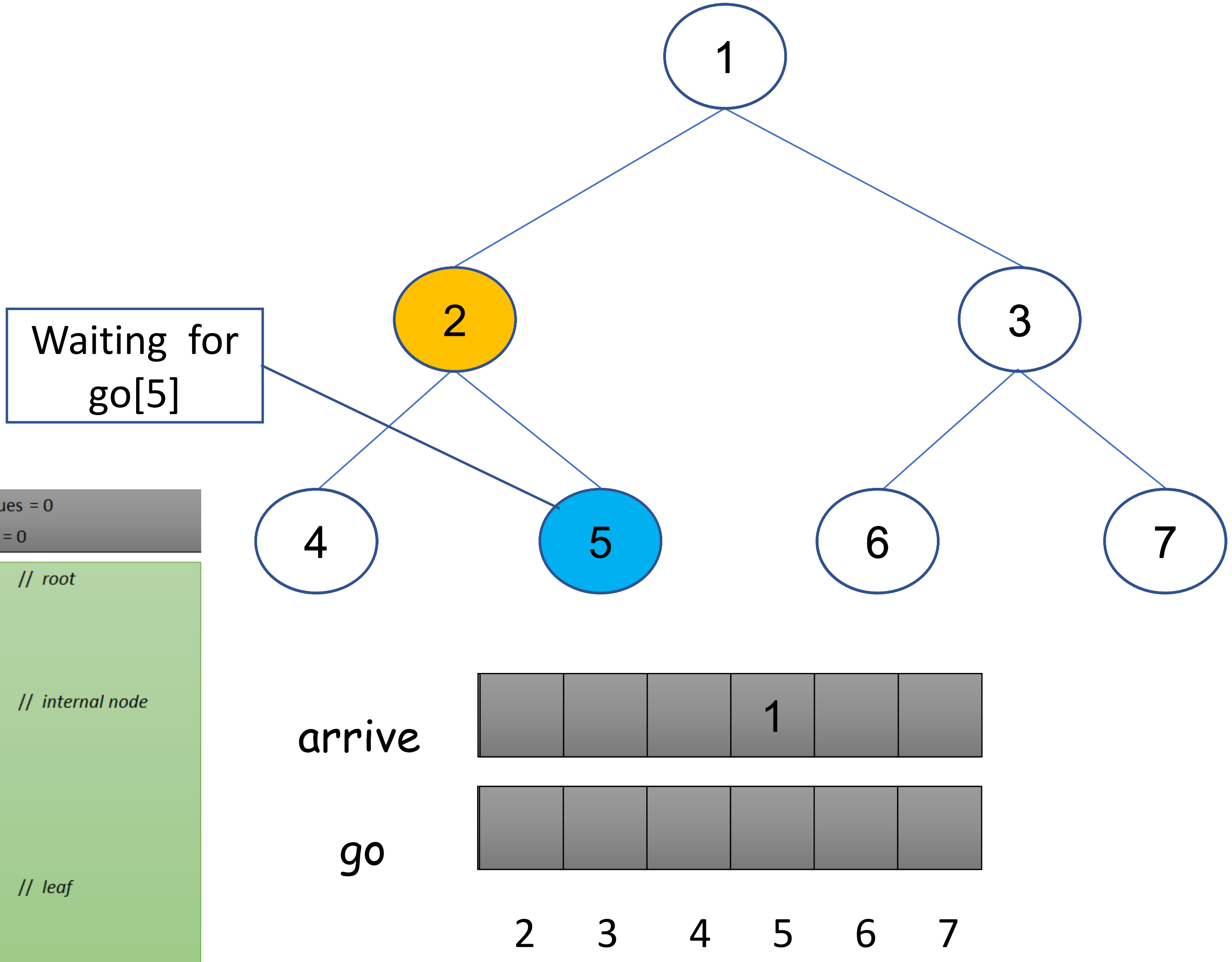
A Tree-based Barrier
Example Run for n=7 threads



shared arrive[2..n]: array of atomic bits, initial values = 0
go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                          // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads



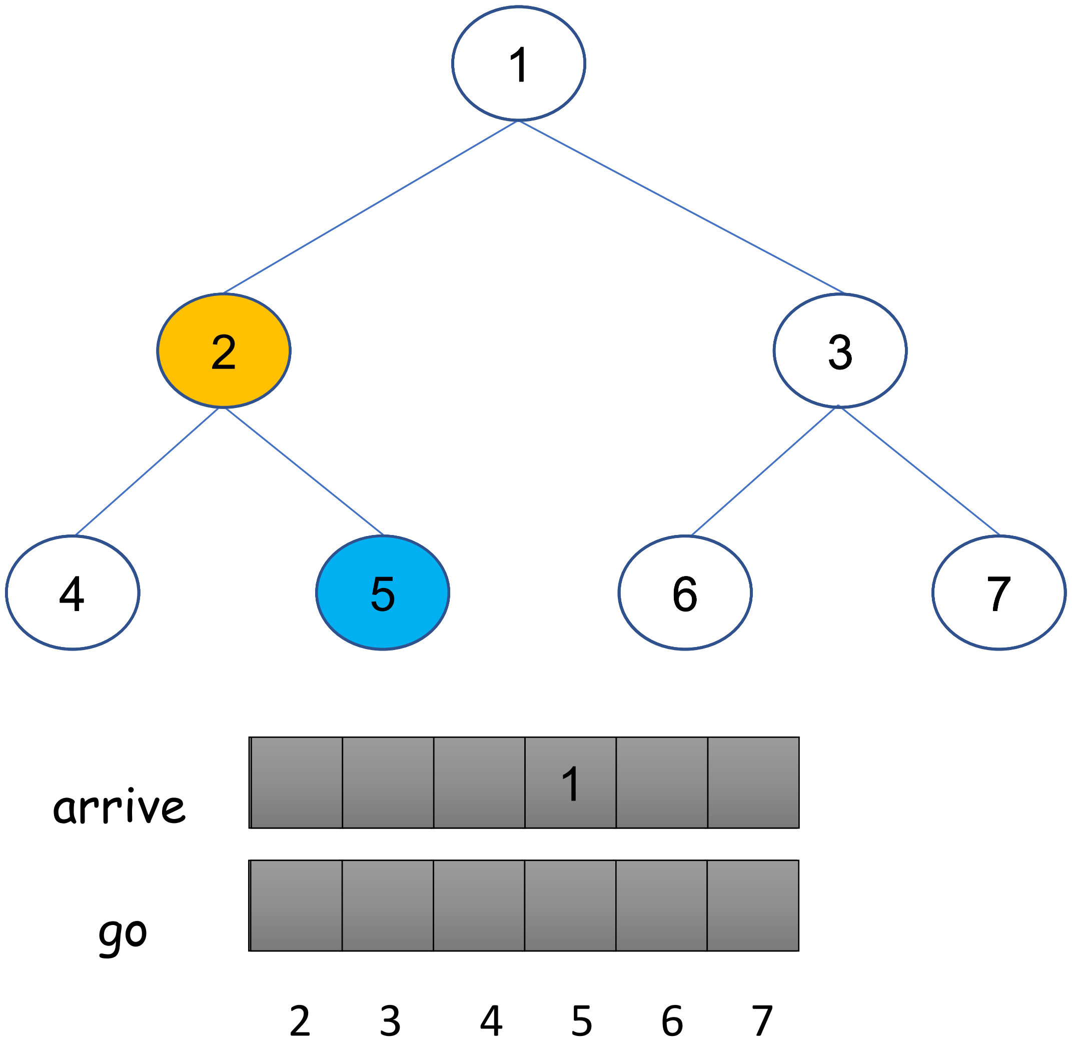

```
shared      arrive[2..n]: array of atomic bits, initial values = 0
            go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                              // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                                 // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                     // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

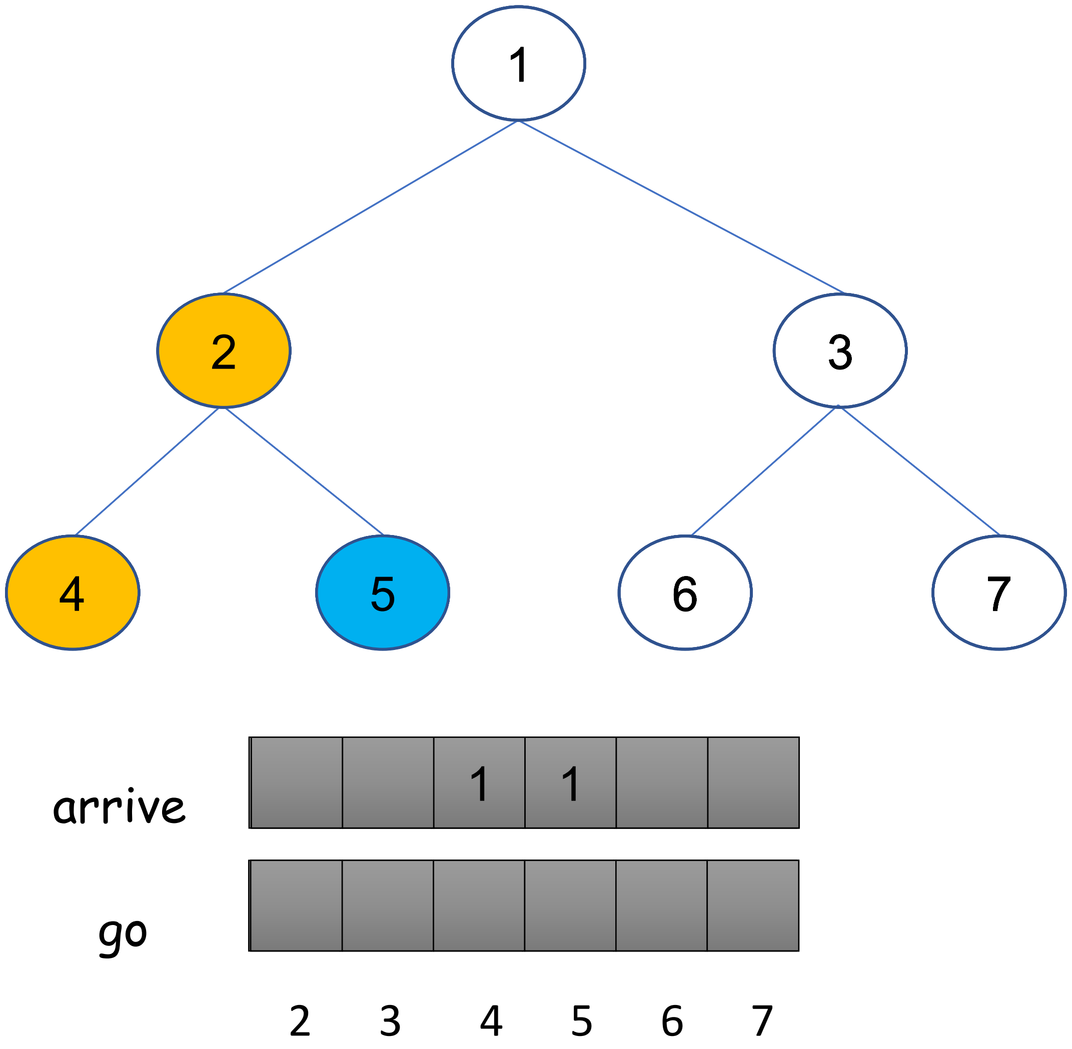
```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                        // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                               // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

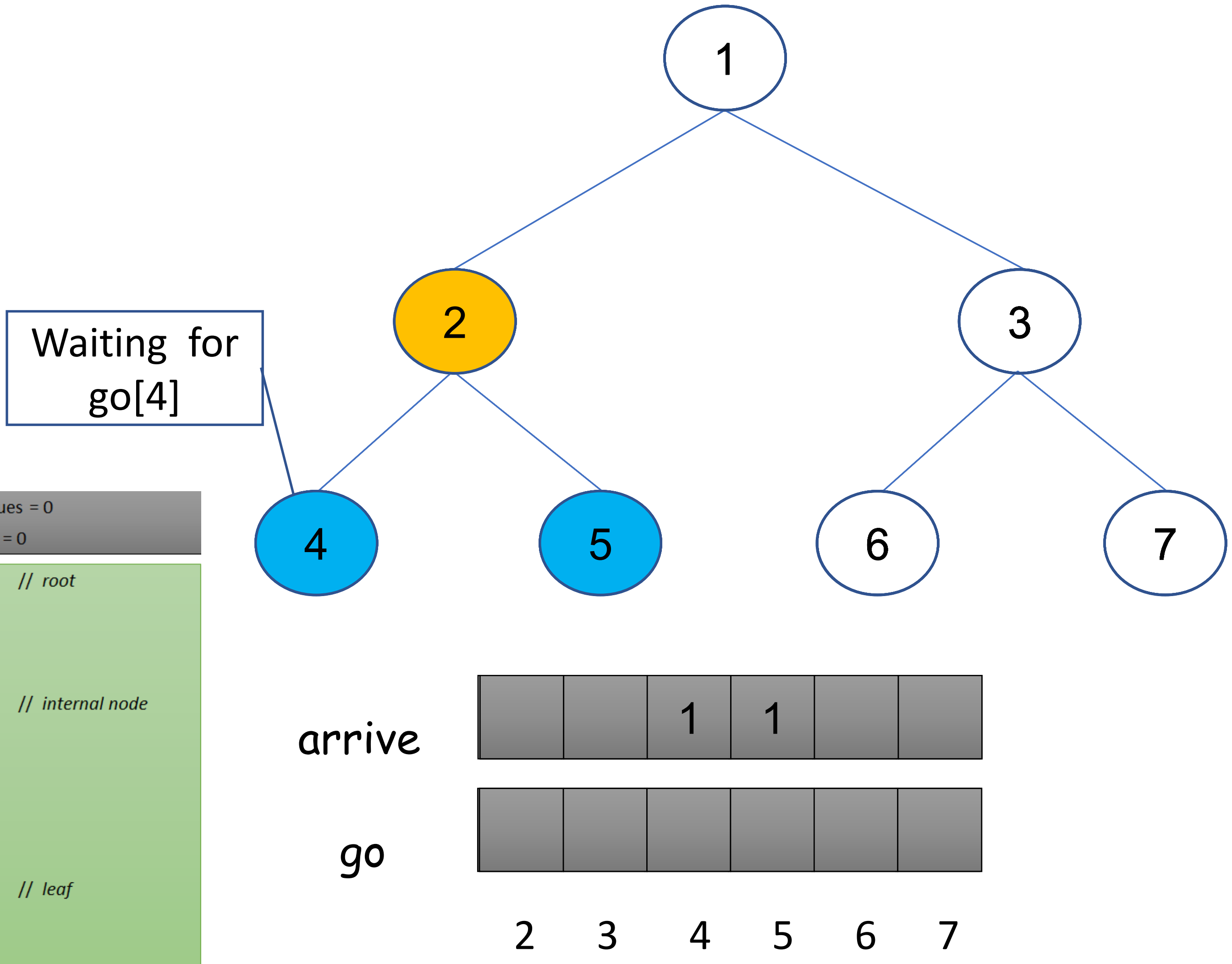


A Tree-based Barrier
Example Run for n=7 threads

```
shared   arrive[2..n]:  array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
      await(arrive[2]  = 1); arrive[2]  := 0
      await(arrive[3]  = 1); arrive[3]  := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i]  = 1); arrive[2i]  := 0
      await(arrive[2i+1]  = 1); arrive[2i+1]  := 0
      arrive[i]  := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i]  := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```



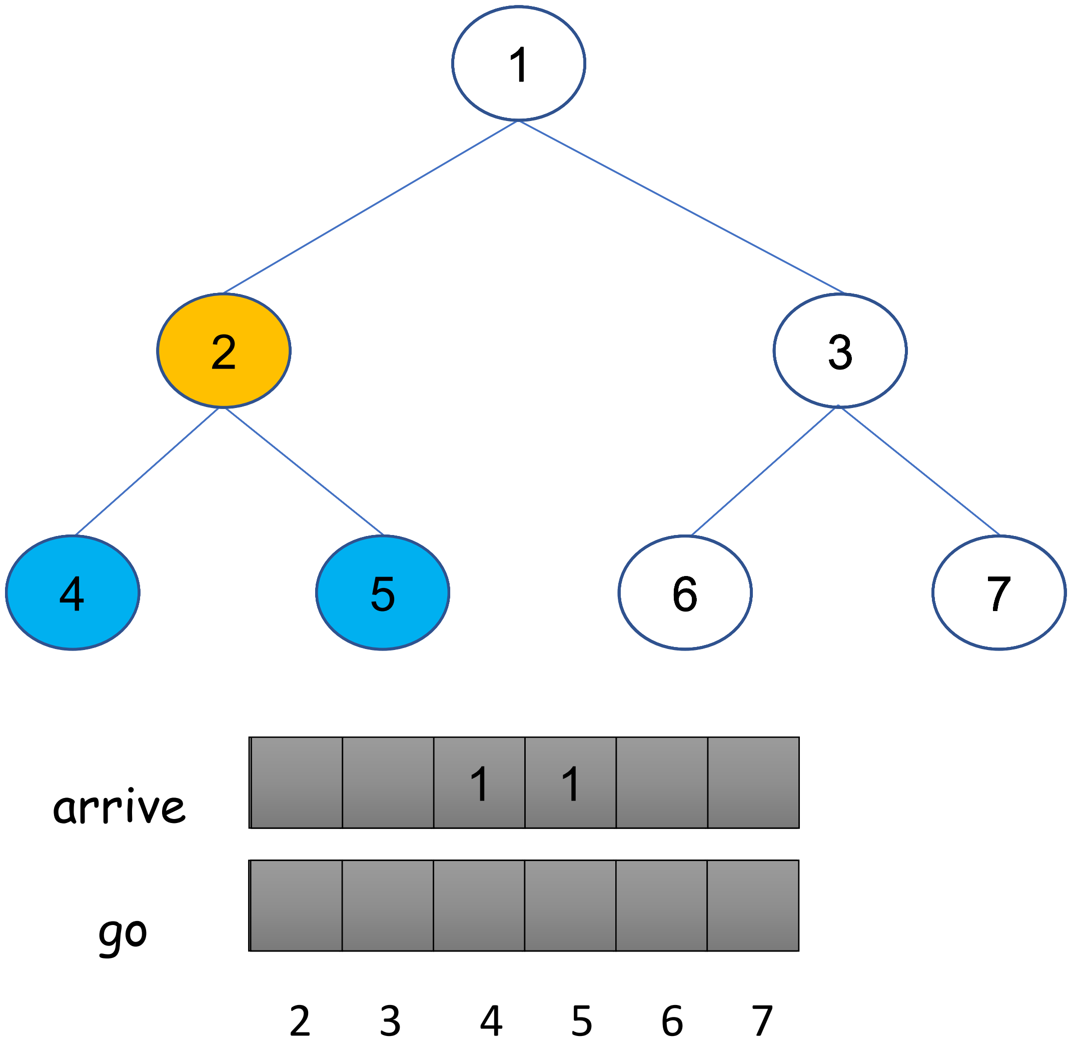
A Tree-based Barrier
Example Run for n=7 threads



shared arrive[2..n]: array of atomic bits, initial values = 0
go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                  // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                     // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                         // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

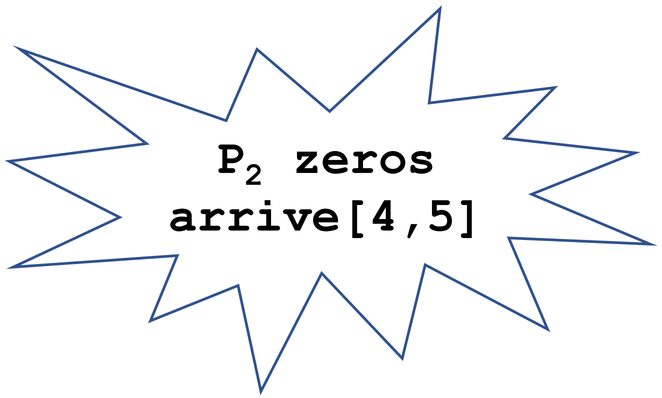
A Tree-based Barrier
Example Run for n=7 threads

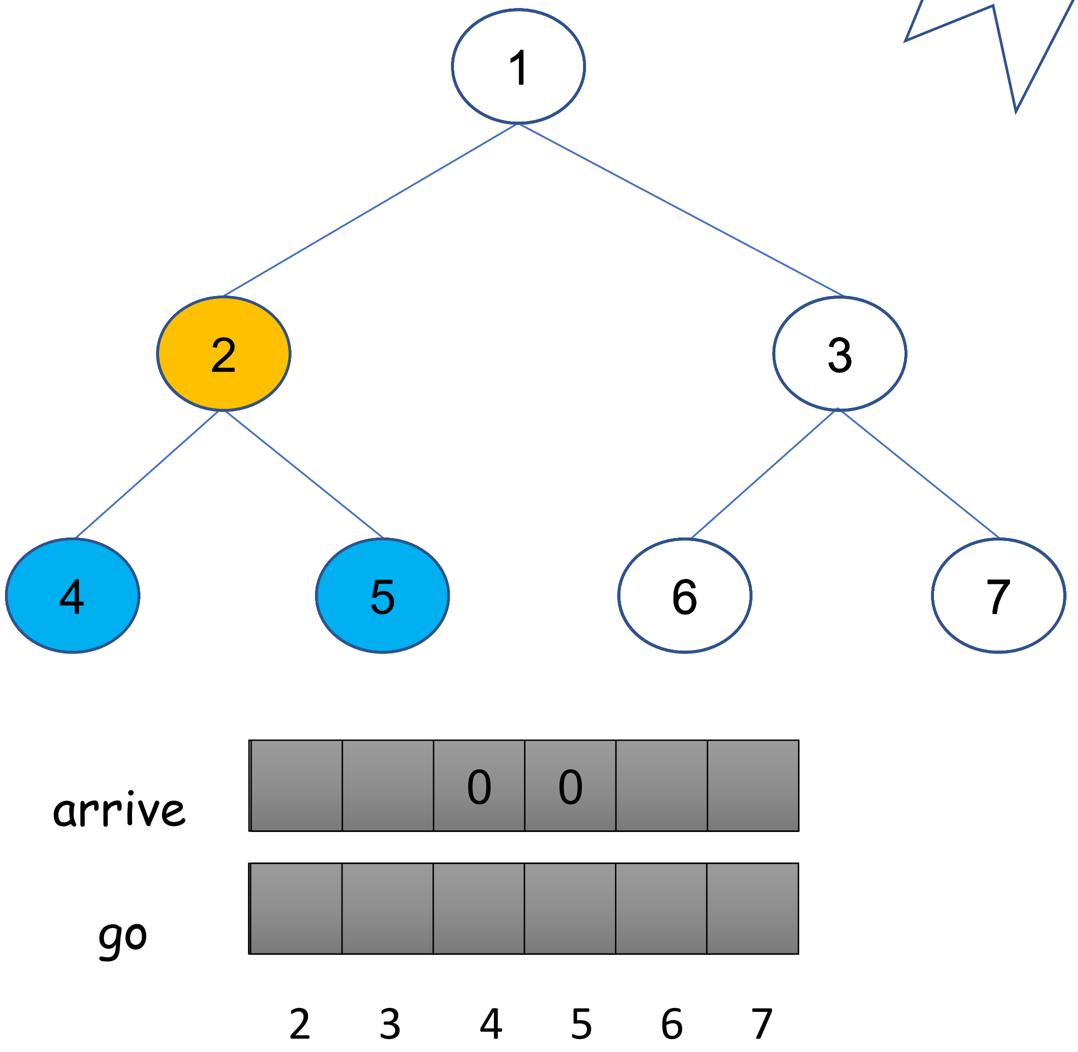



```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                          // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads





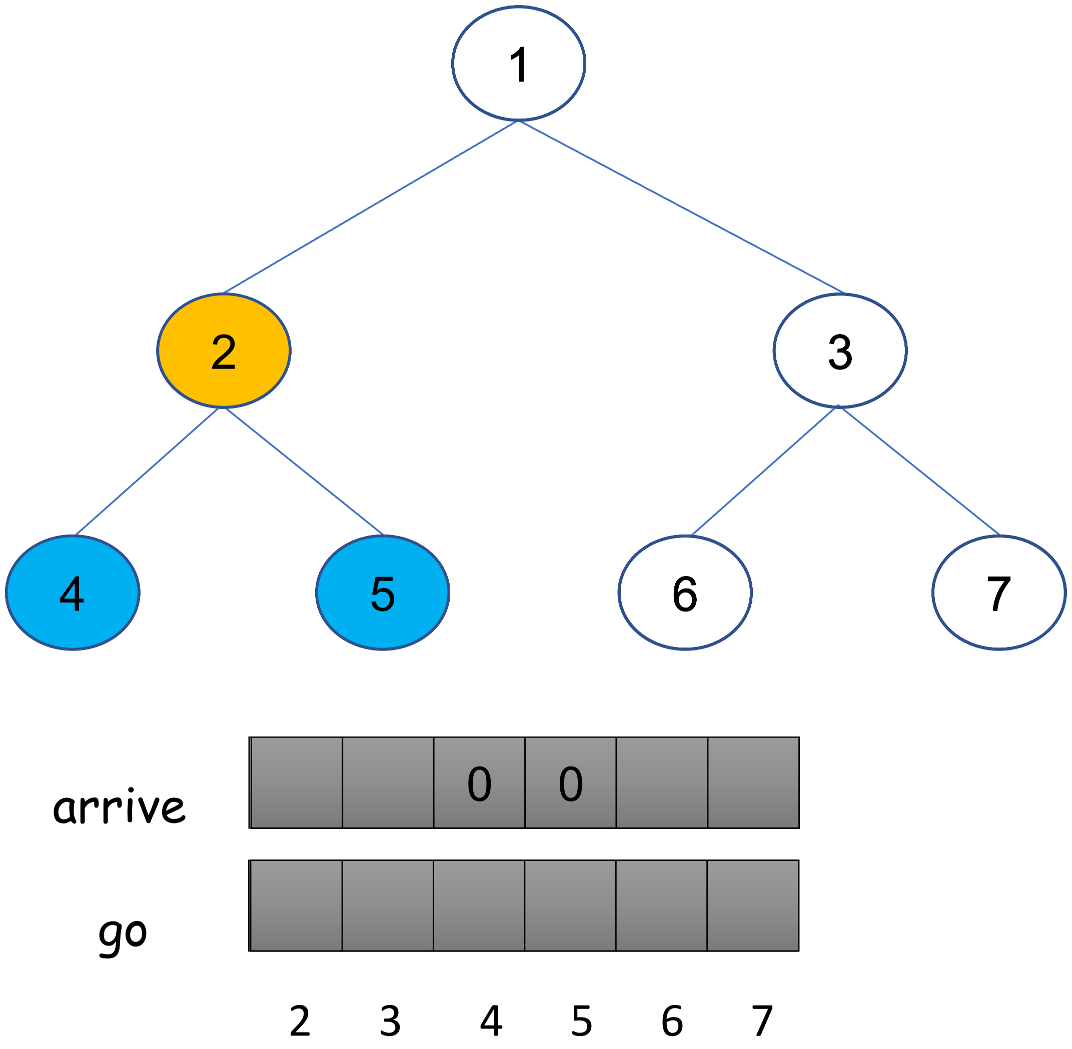
shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                      // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                         // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                             // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

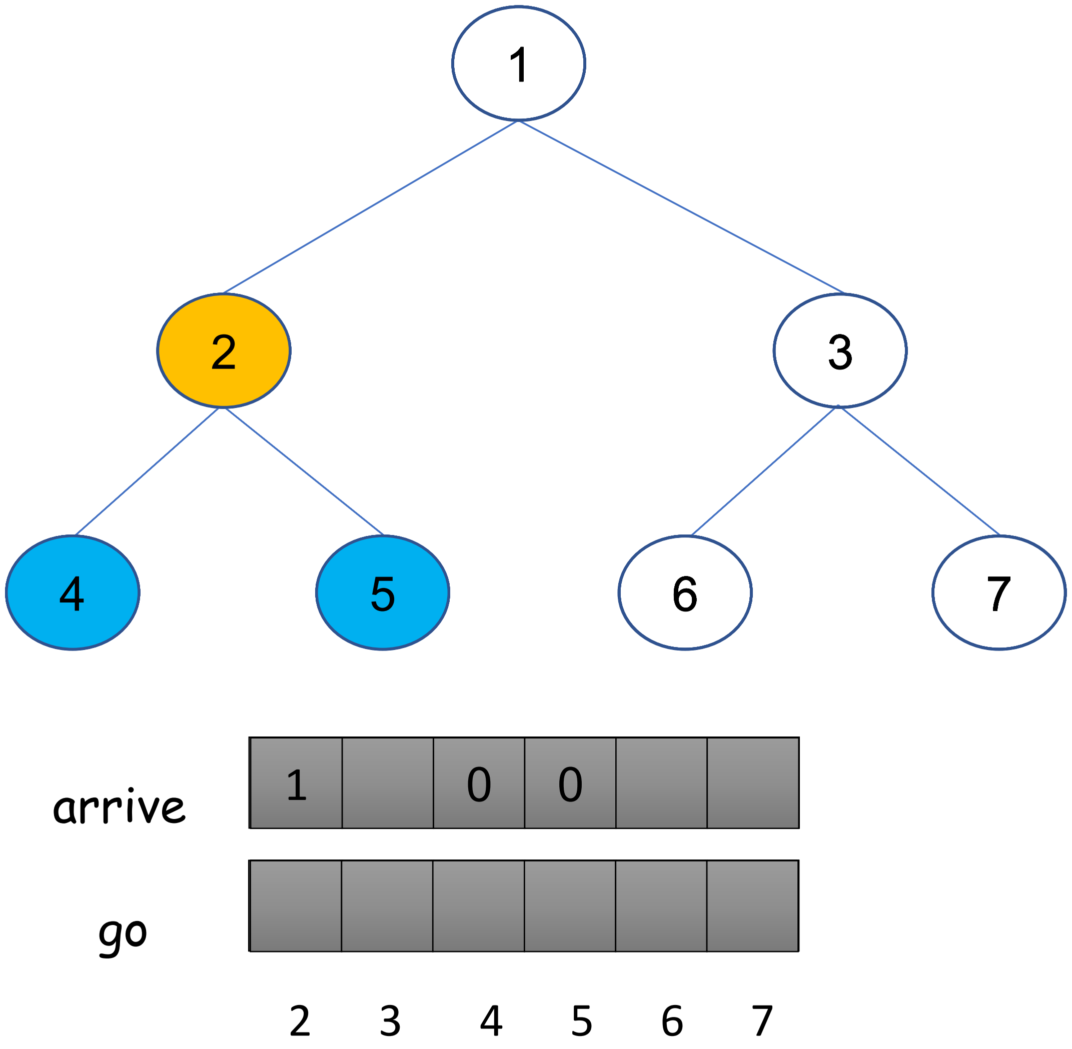
A Tree-based Barrier
Example Run for n=7 threads

```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                        // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                               // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```



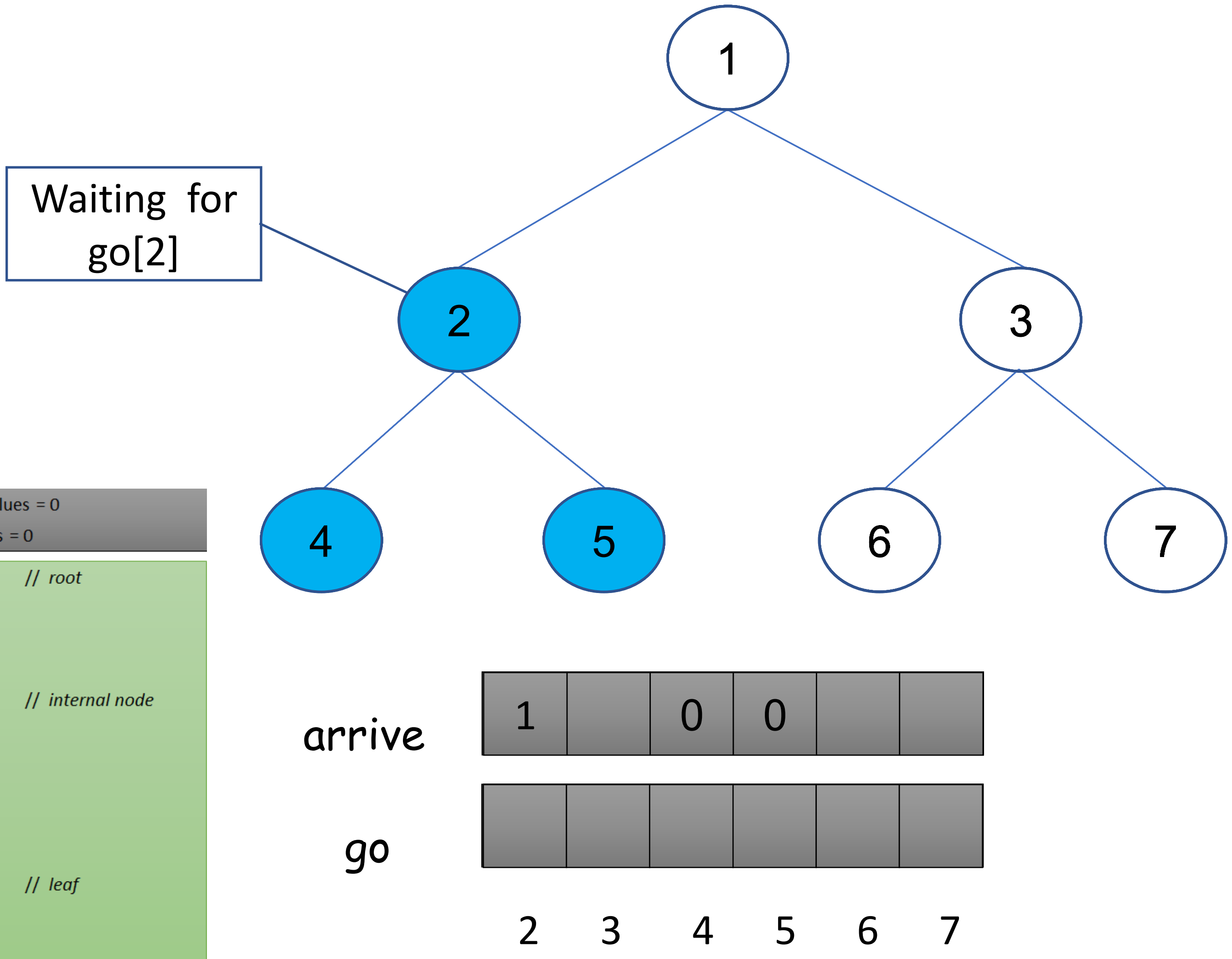
A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

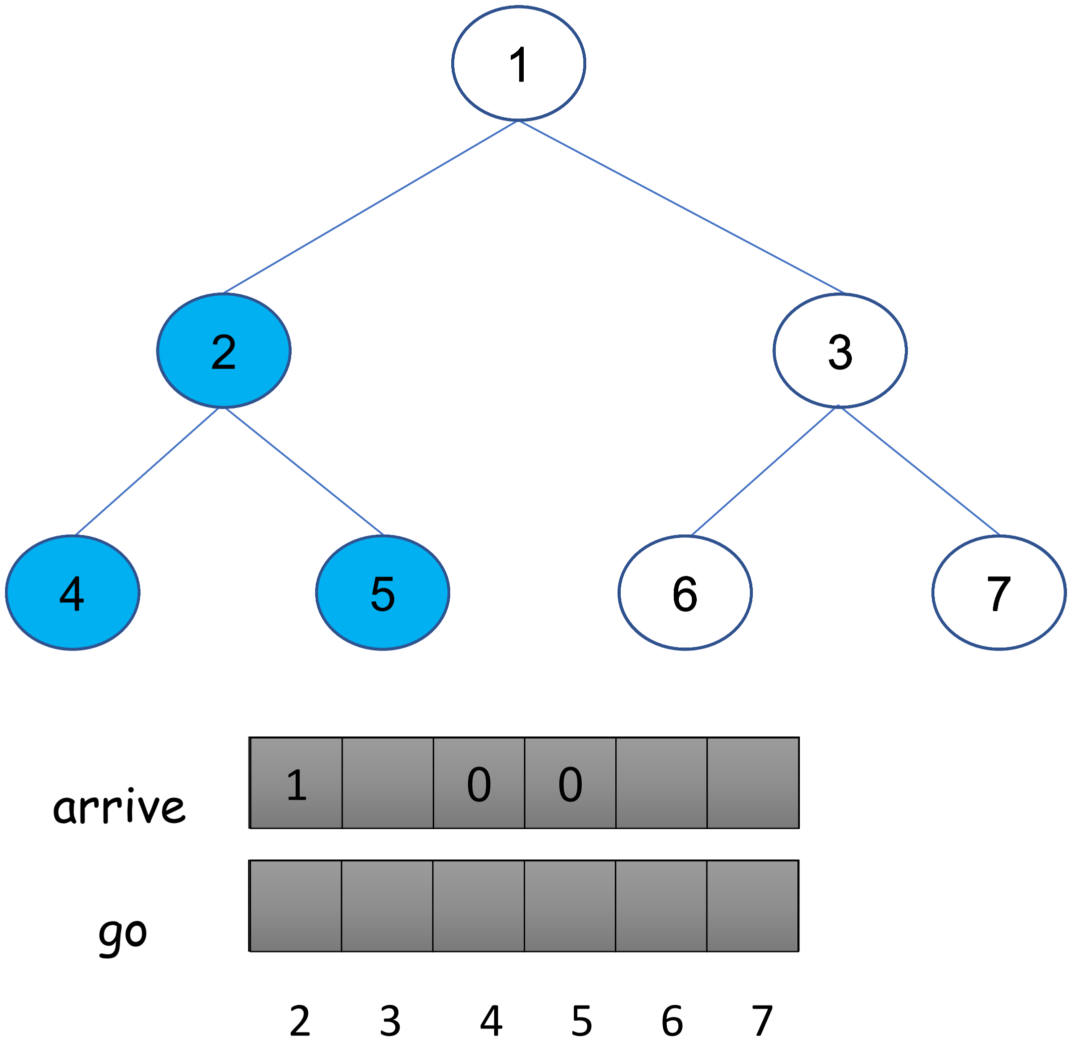
```
if i=1 then                                        // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                               // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads



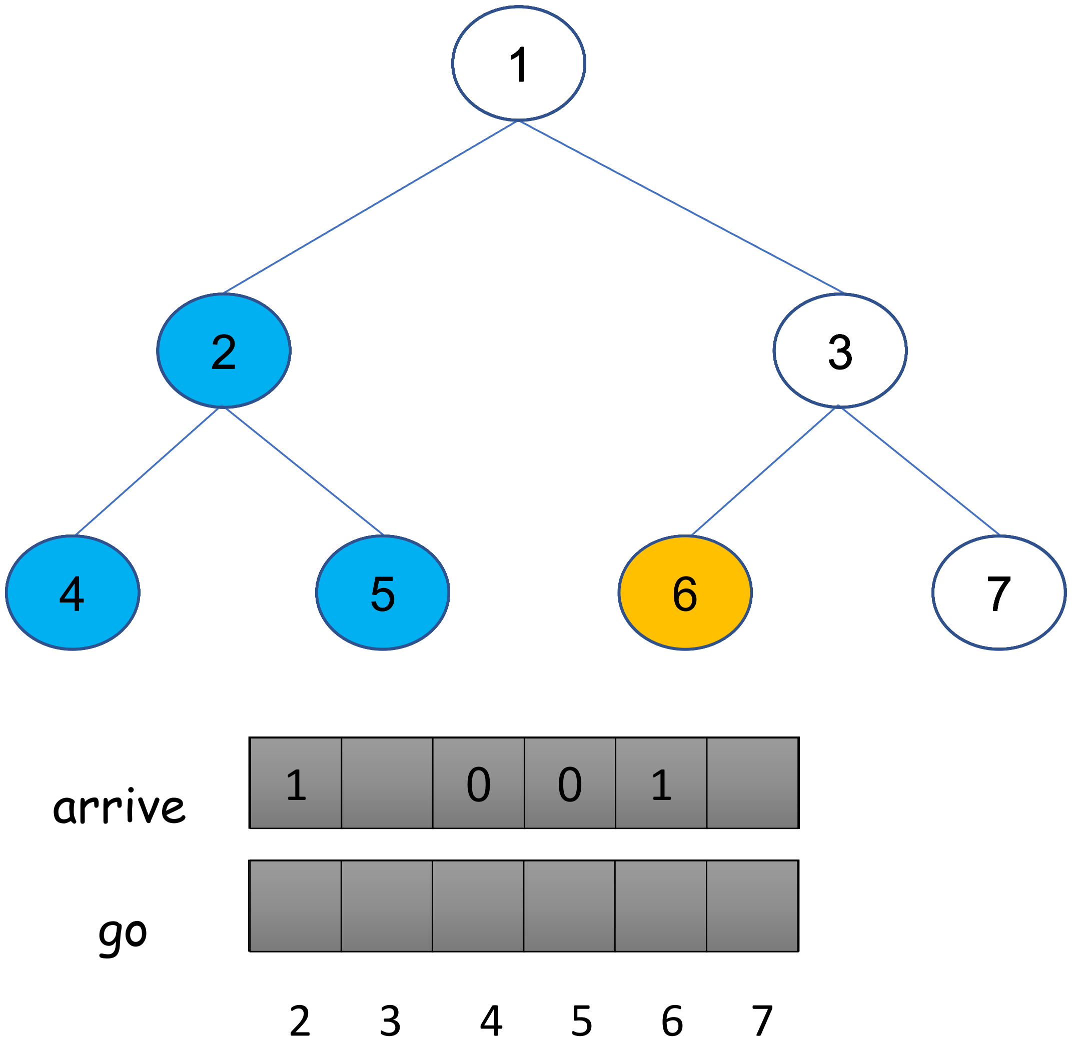

```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads



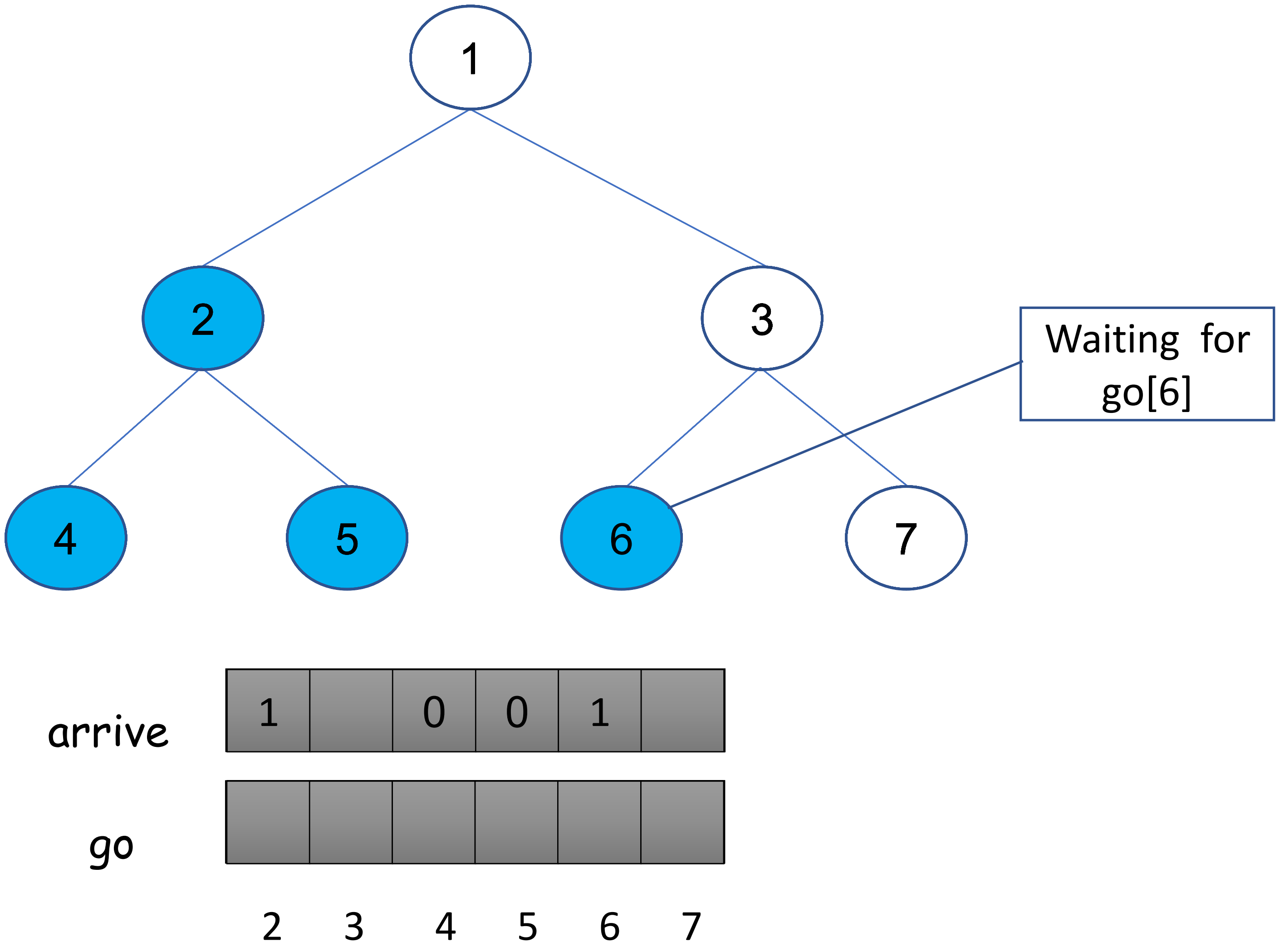

```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads



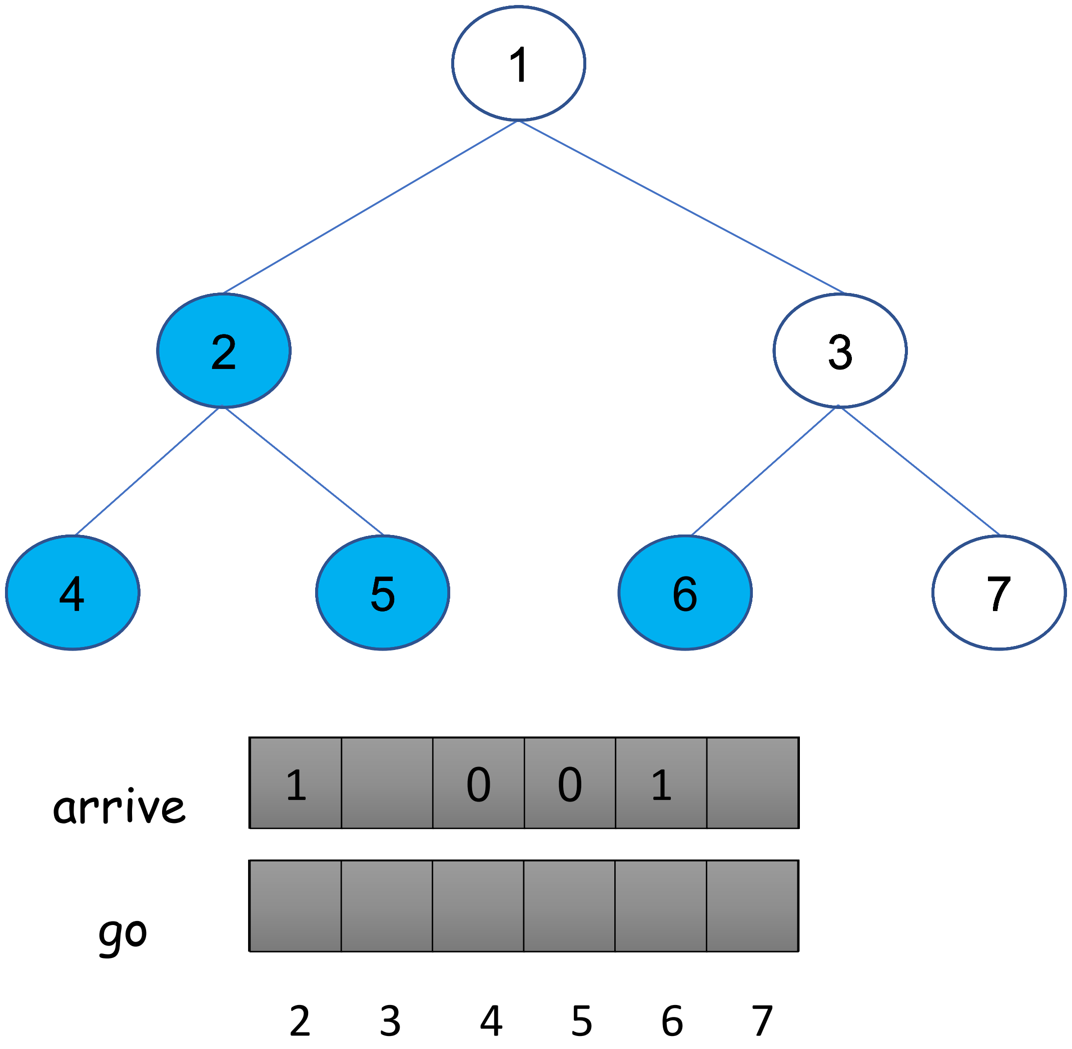

```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

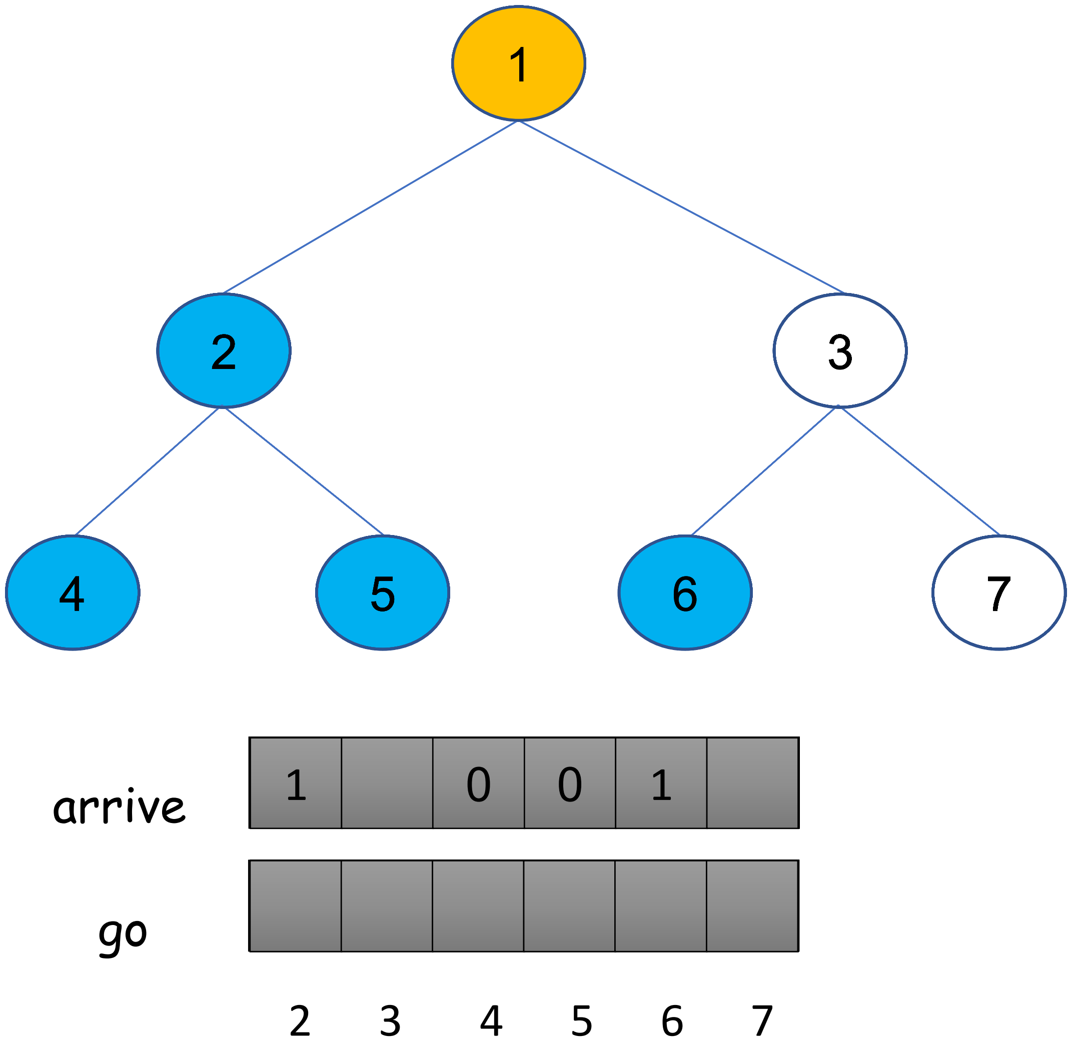
A Tree-based Barrier
Example Run for n=7 threads



shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                   // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                      // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                          // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

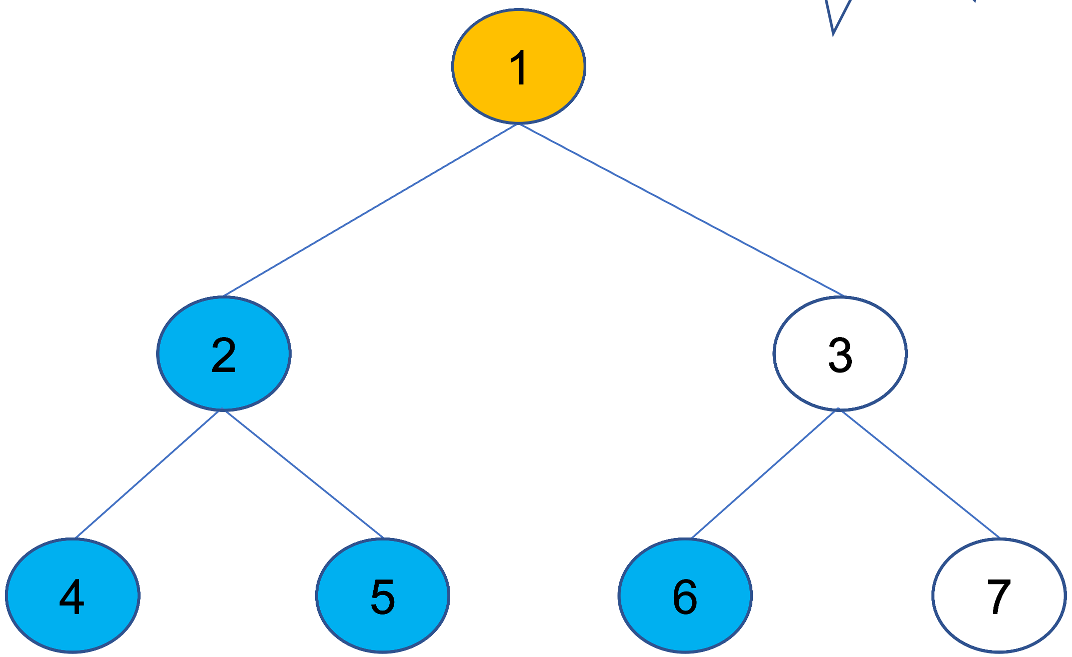
A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                         // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

Arrive[2]=1
?

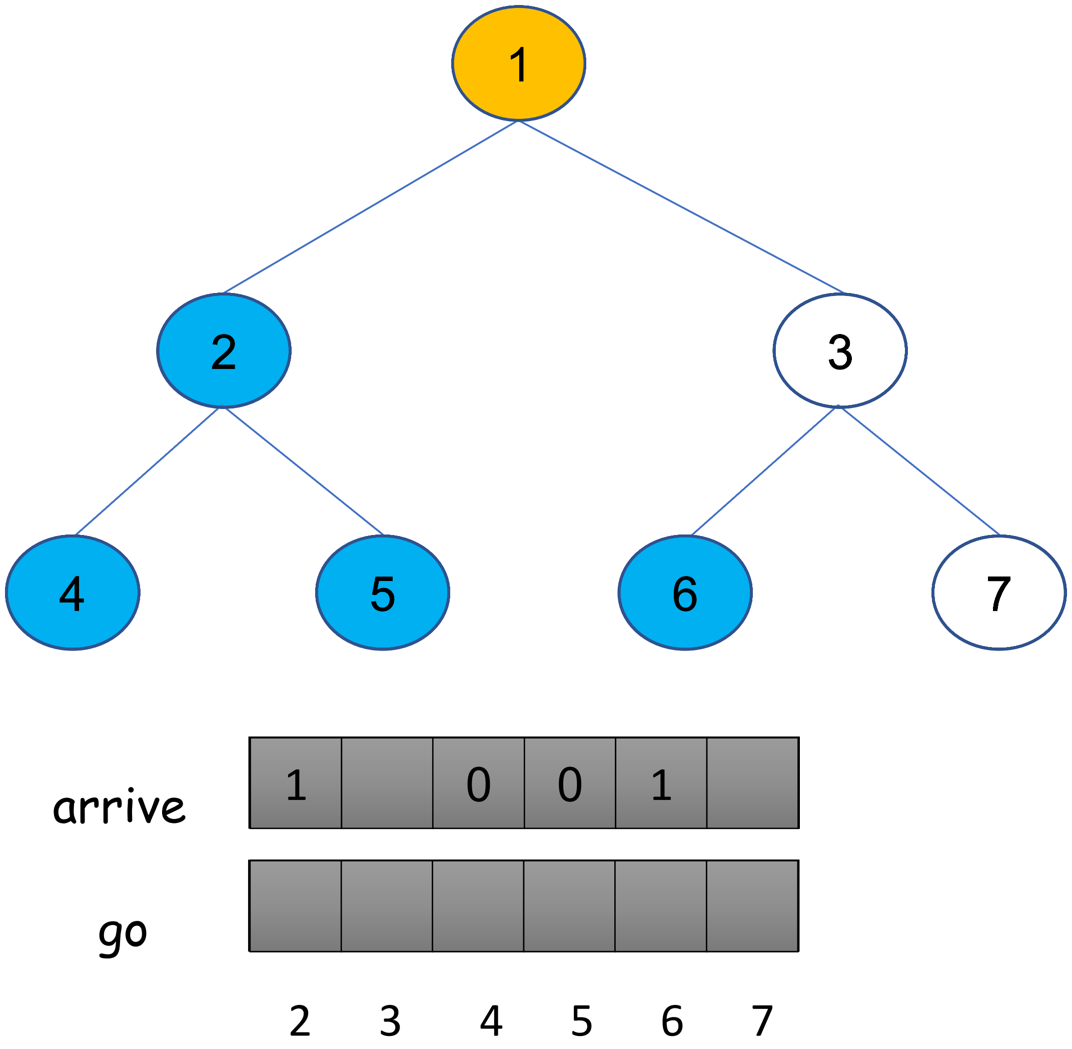


shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                   // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                      // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                          // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

	2	3	4	5	6	7
arrive	1		0	0	1	
go						

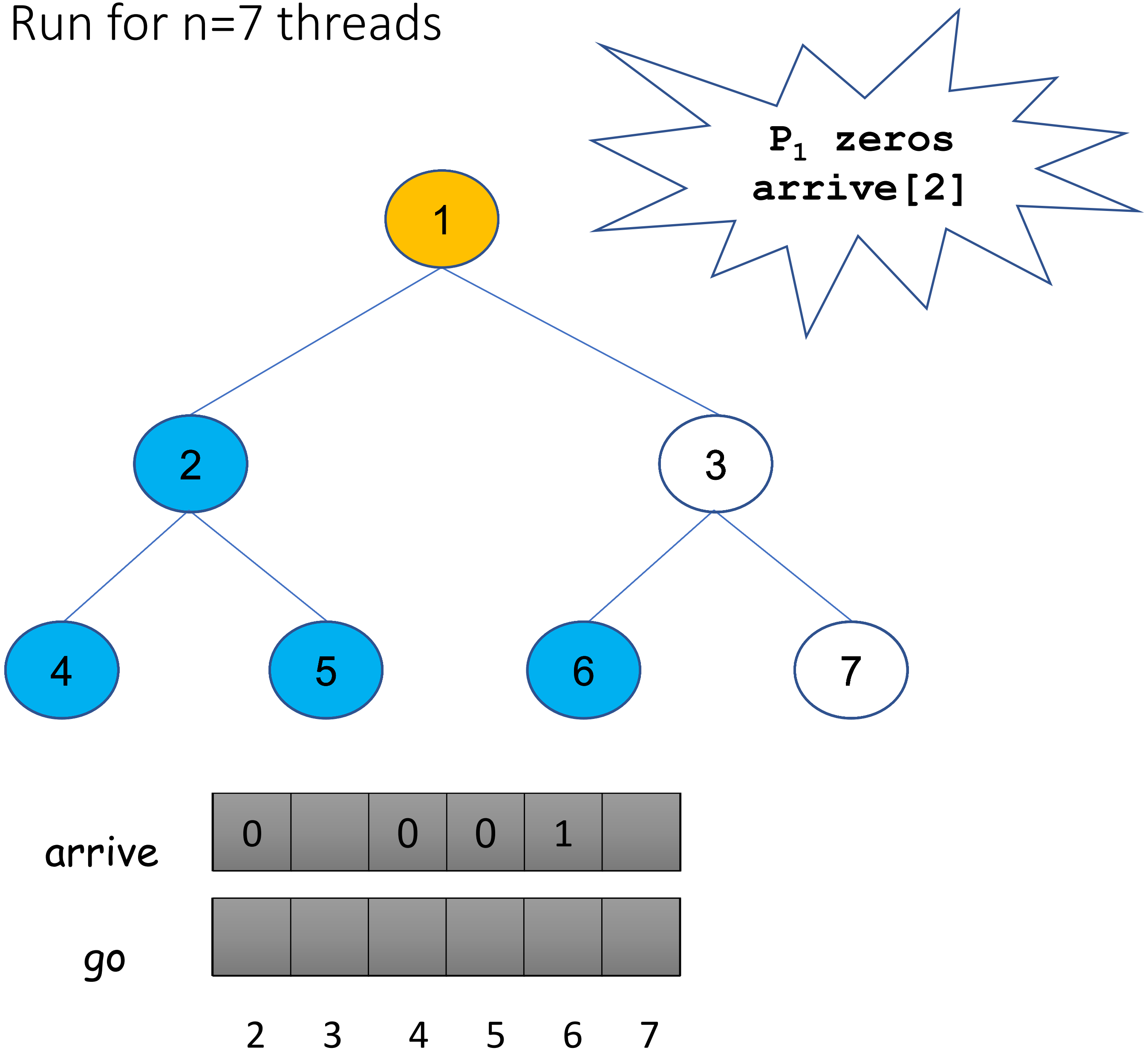
A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

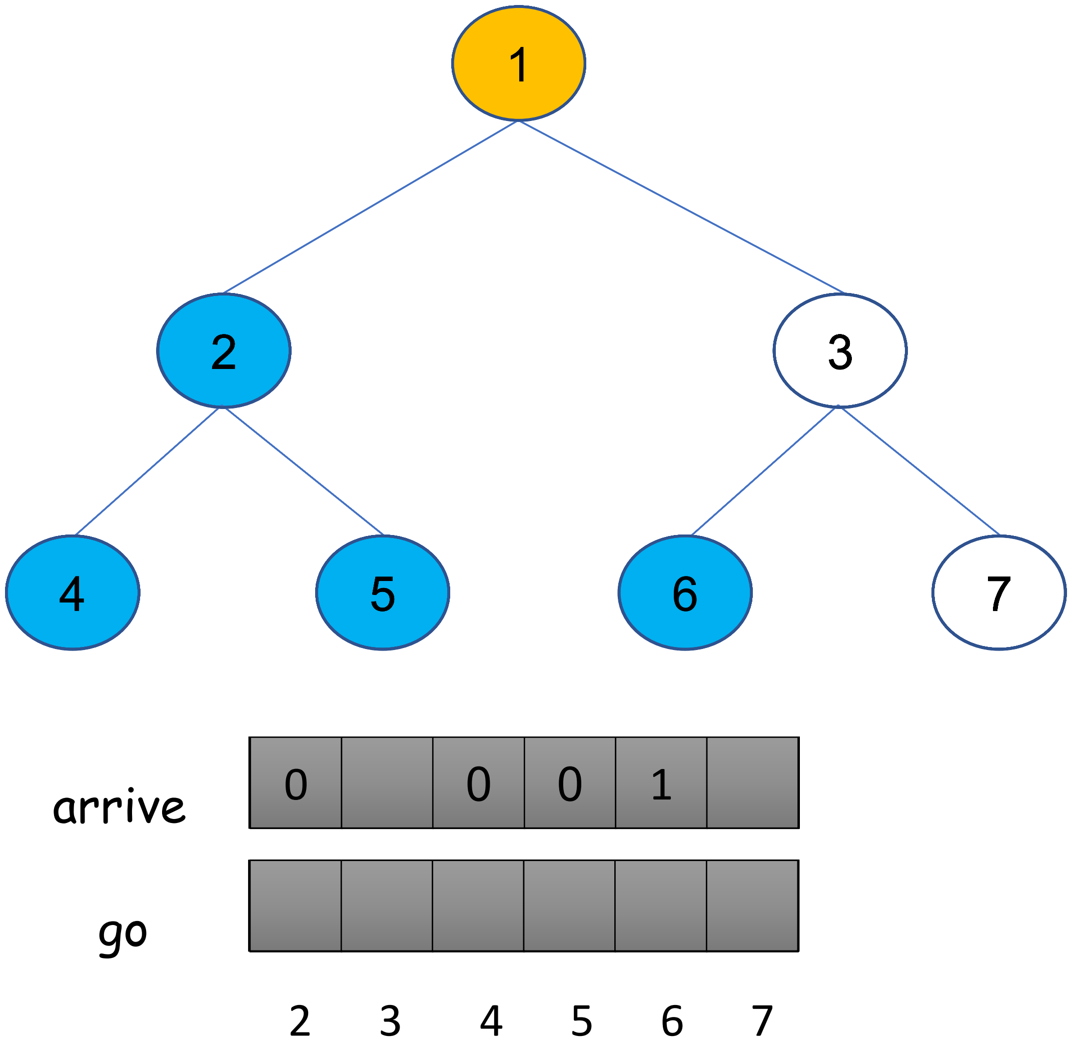
```
if i=1 then                                         // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                                // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads




```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                   // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                      // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                          // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

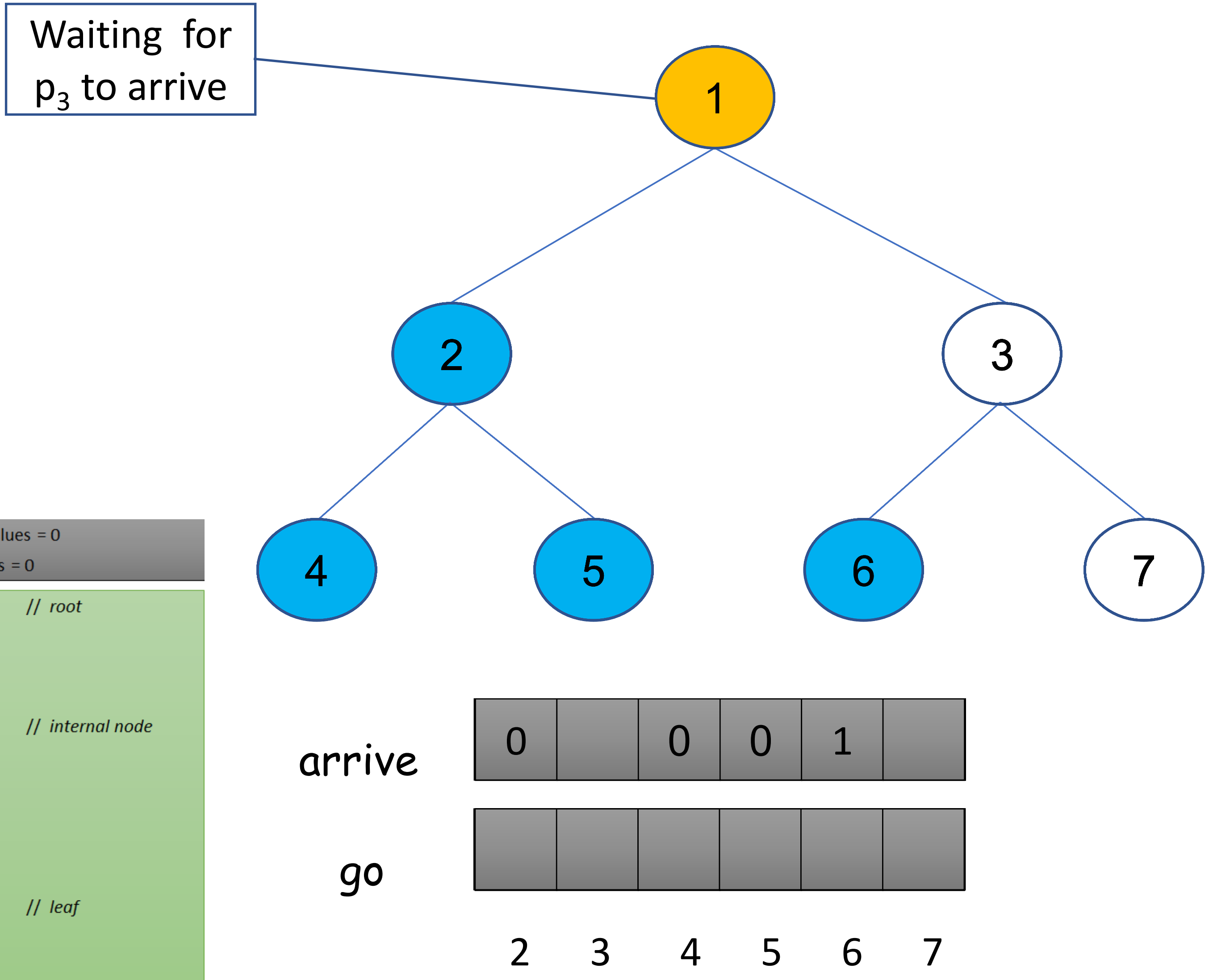
A Tree-based Barrier
Example Run for n=7 threads



shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

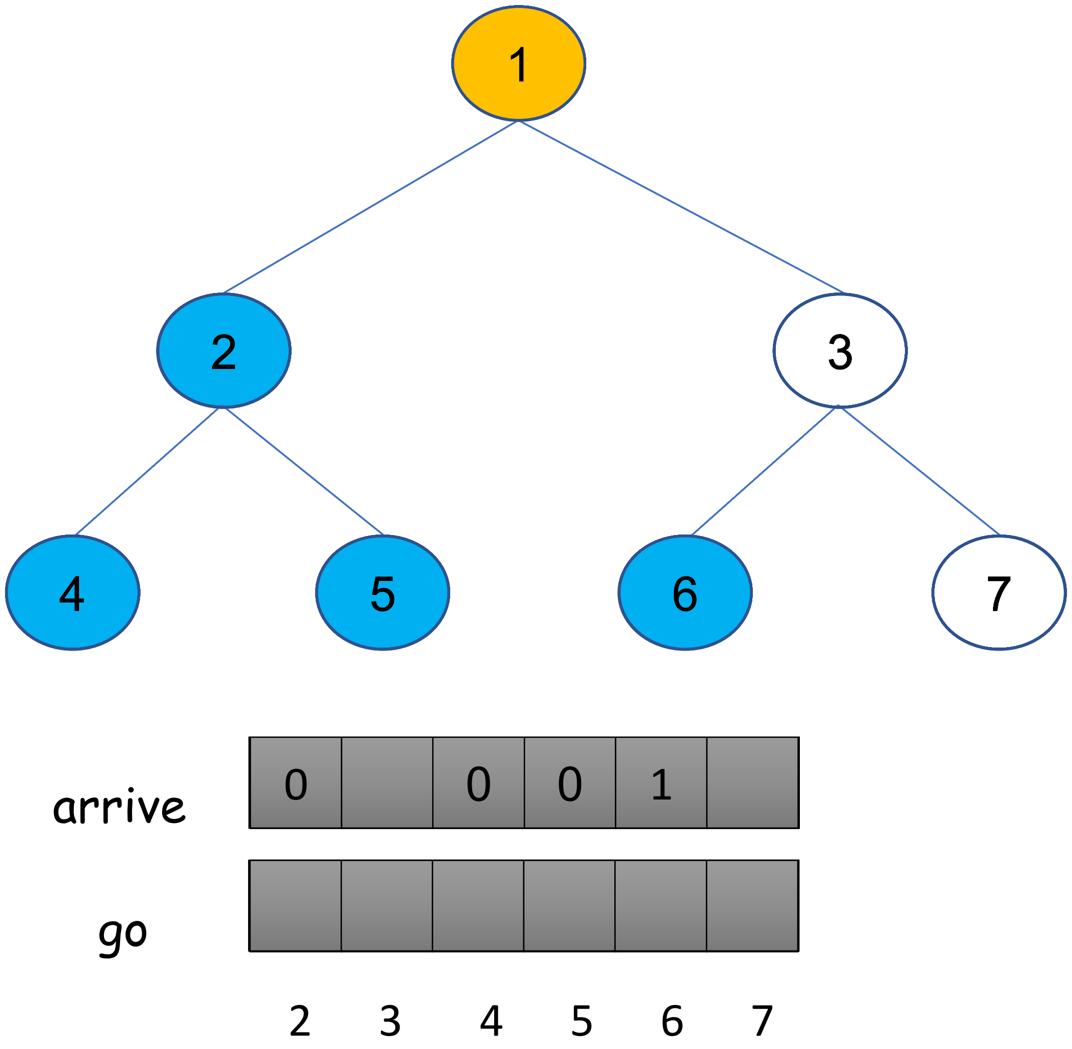
```
if i=1 then                                         // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                        // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                               // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

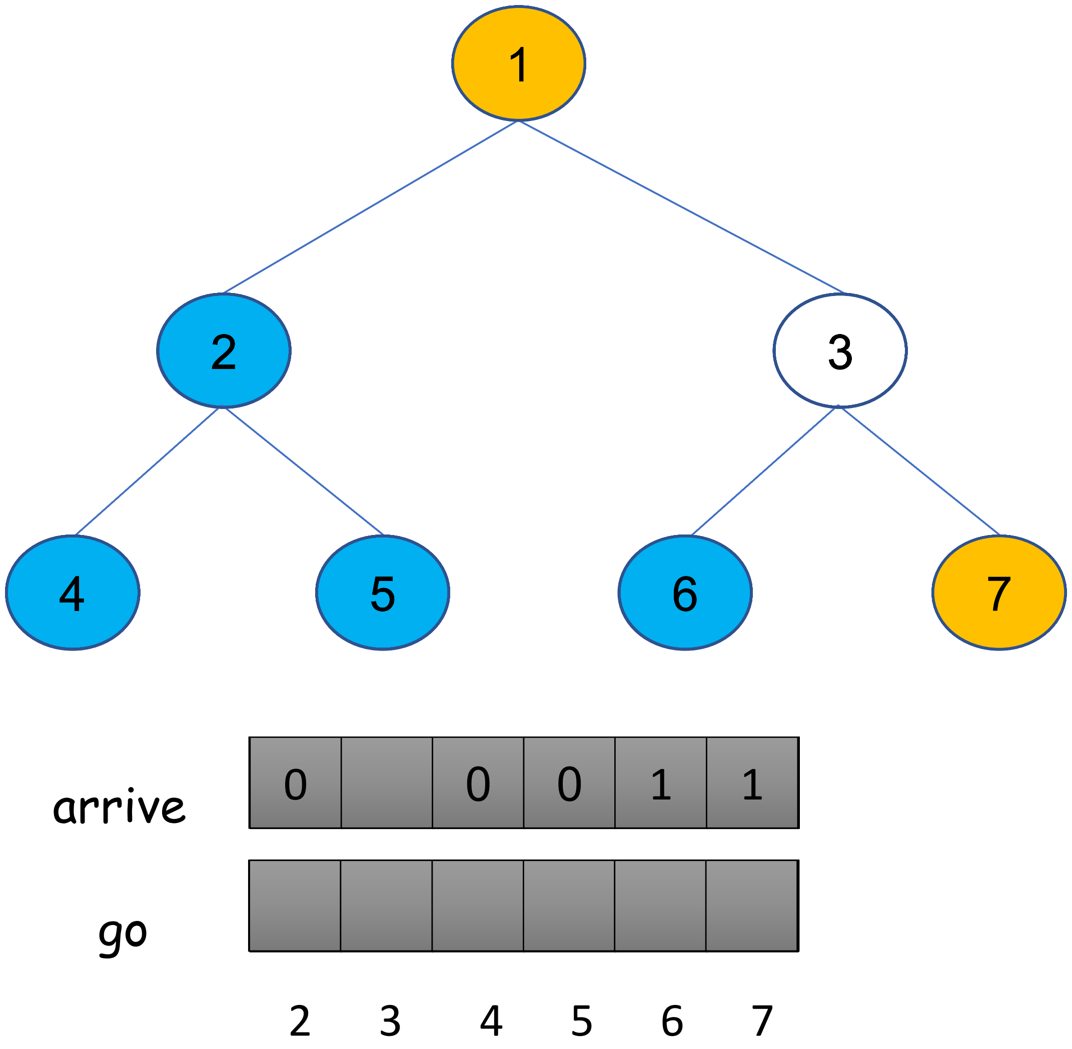
A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                    // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                       // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                           // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads



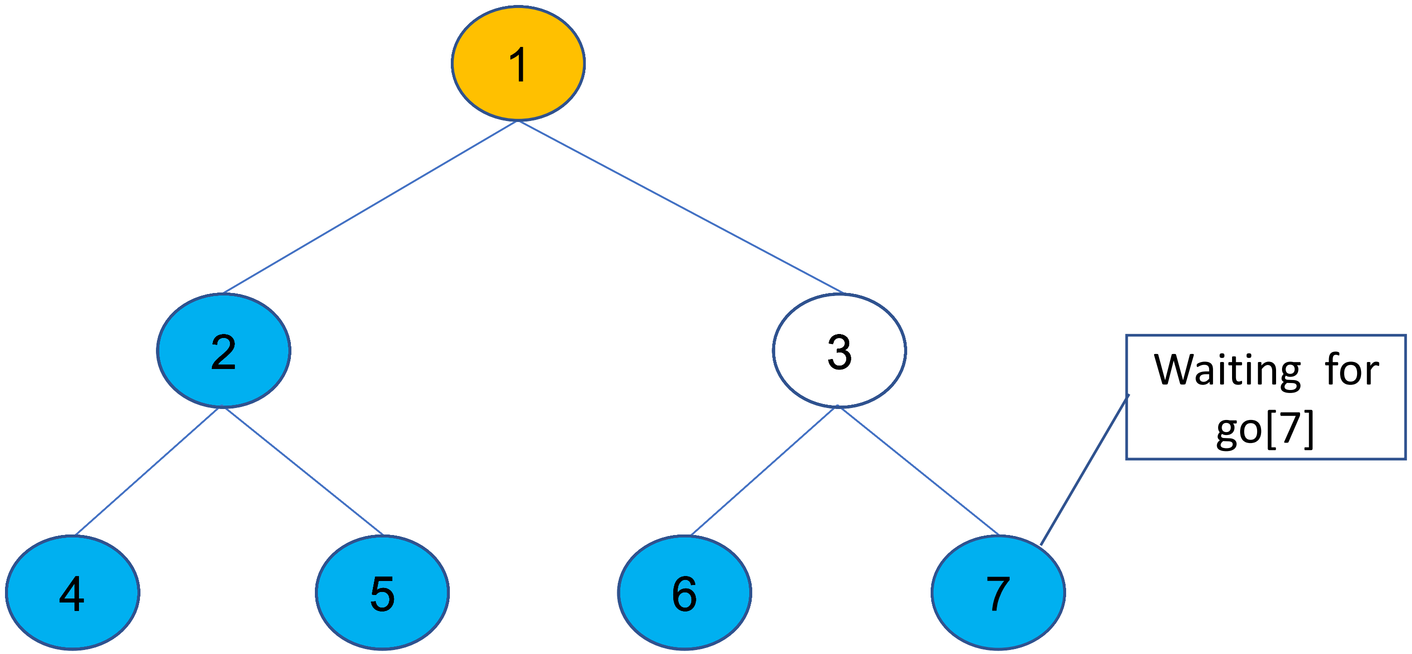
shared arrive[2..n]: array of atomic bits, initial values = 0
go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                        // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                               // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

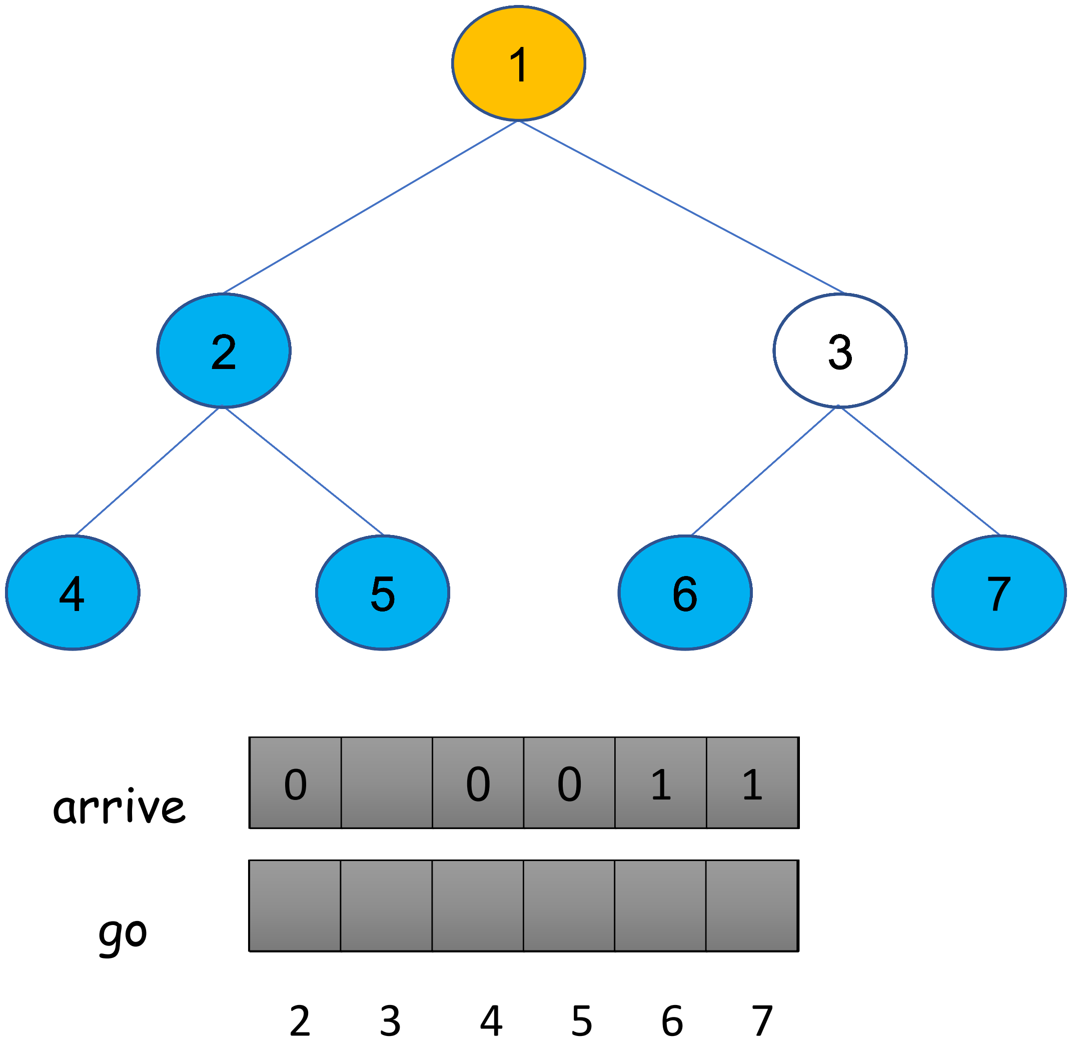
```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                    // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                       // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                           // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```



	2	3	4	5	6	7
arrive	0		0	0	1	1
go						

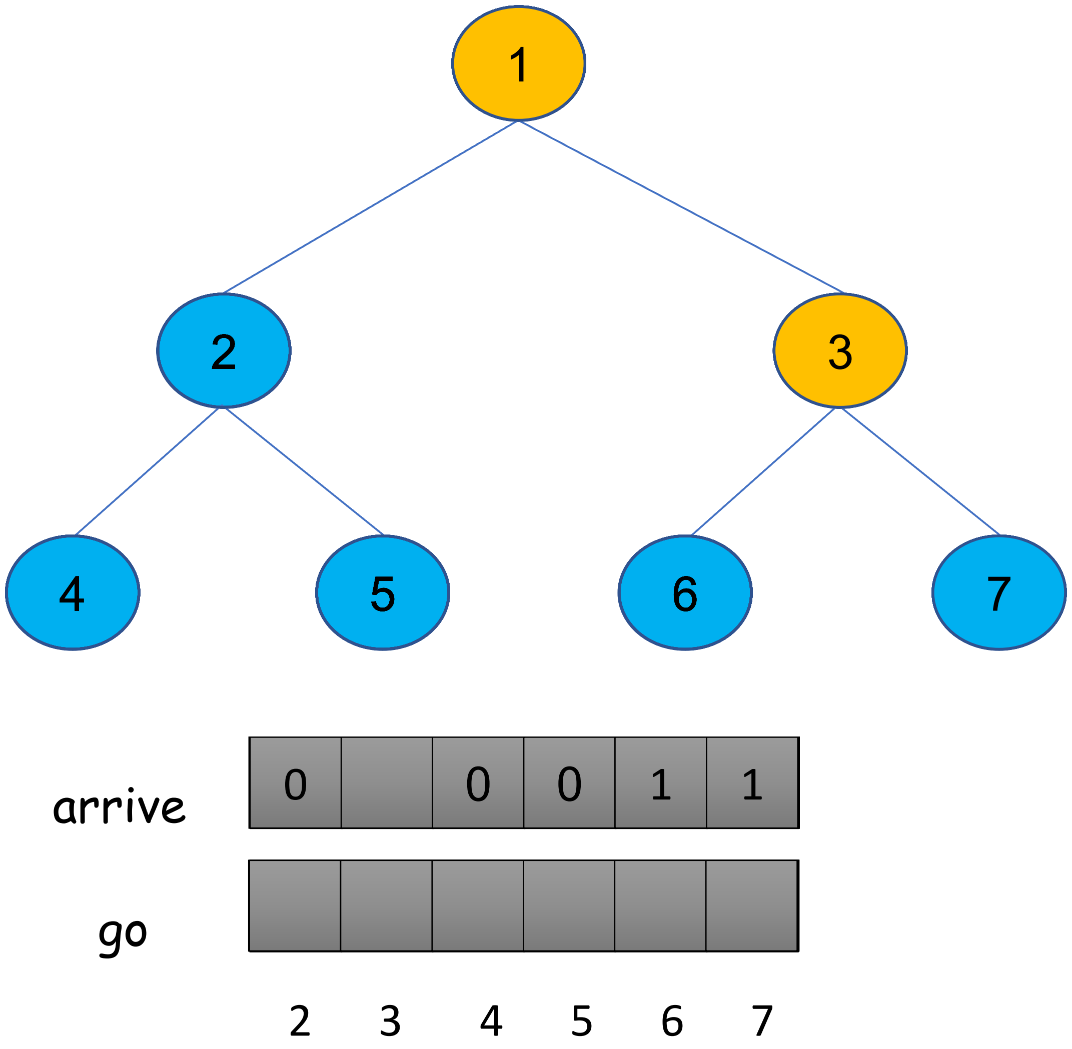
A Tree-based Barrier
Example Run for n=7 threads



shared	arrive[2..n]: array of atomic bits, initial values = 0
	go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                        // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                               // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

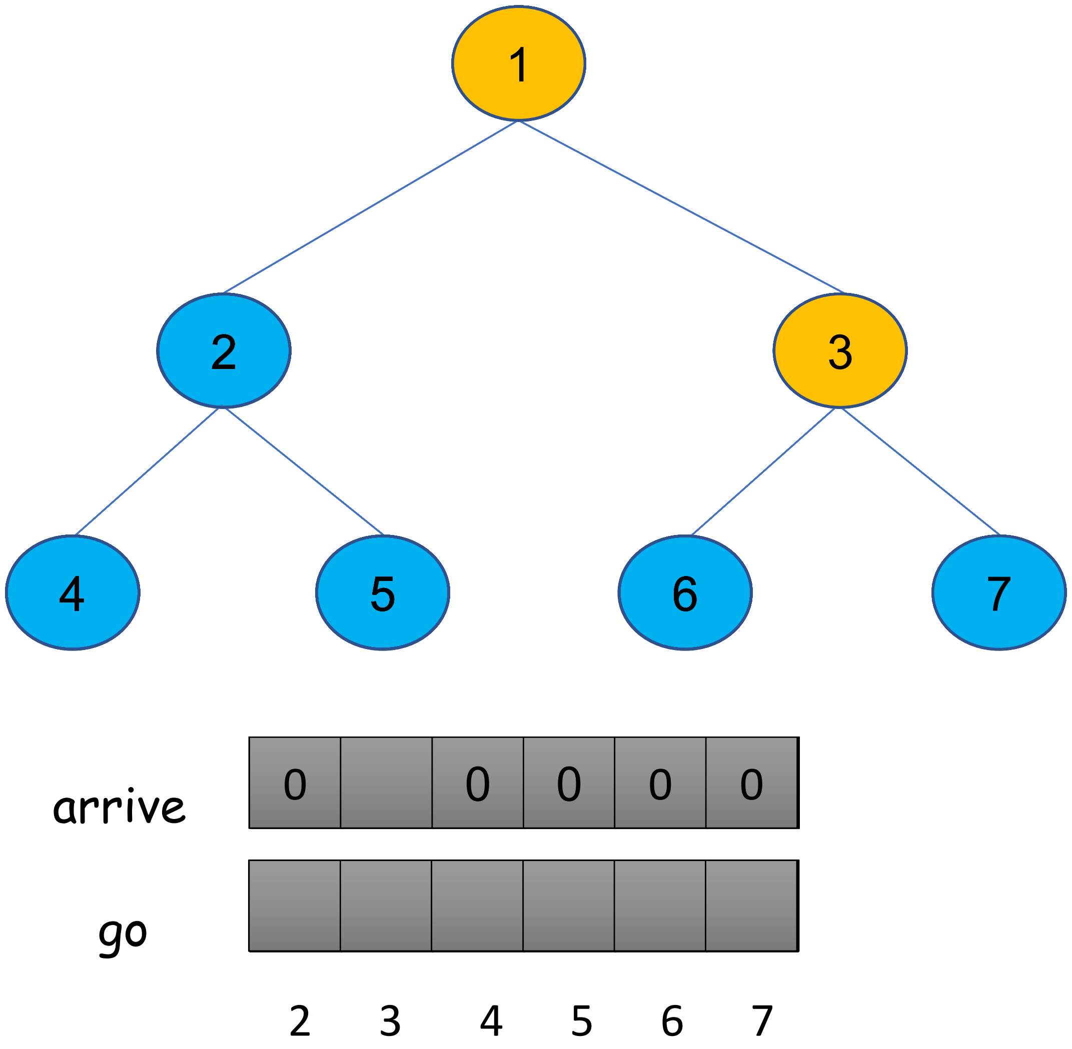
A Tree-based Barrier
Example Run for n=7 threads




```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                   // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                      // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                          // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

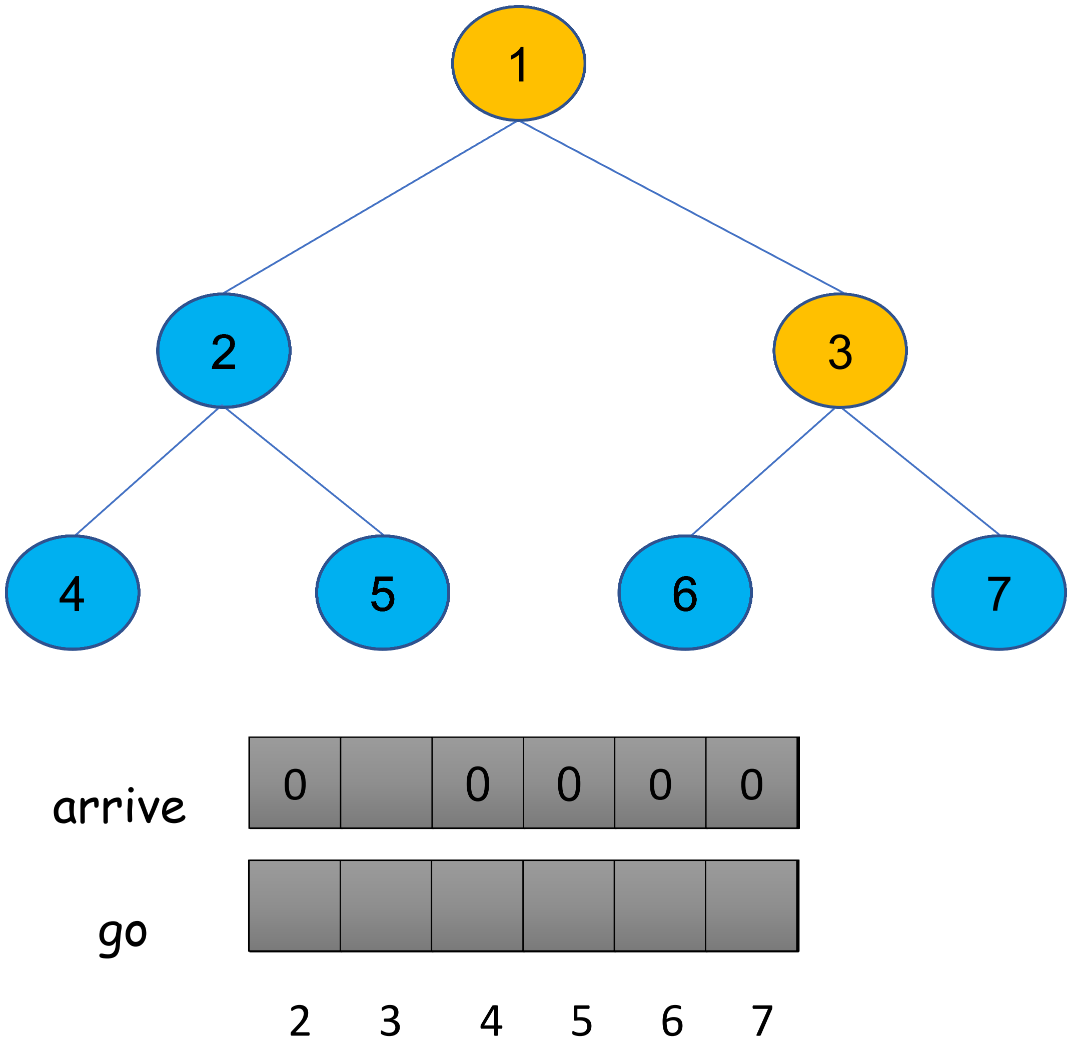
A Tree-based Barrier
Example Run for n=7 threads

P_3 zeros arrive[6,7]




```
shared  arrive[2..n]: array of atomic bits, initial values = 0
        go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                   // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                      // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                          // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

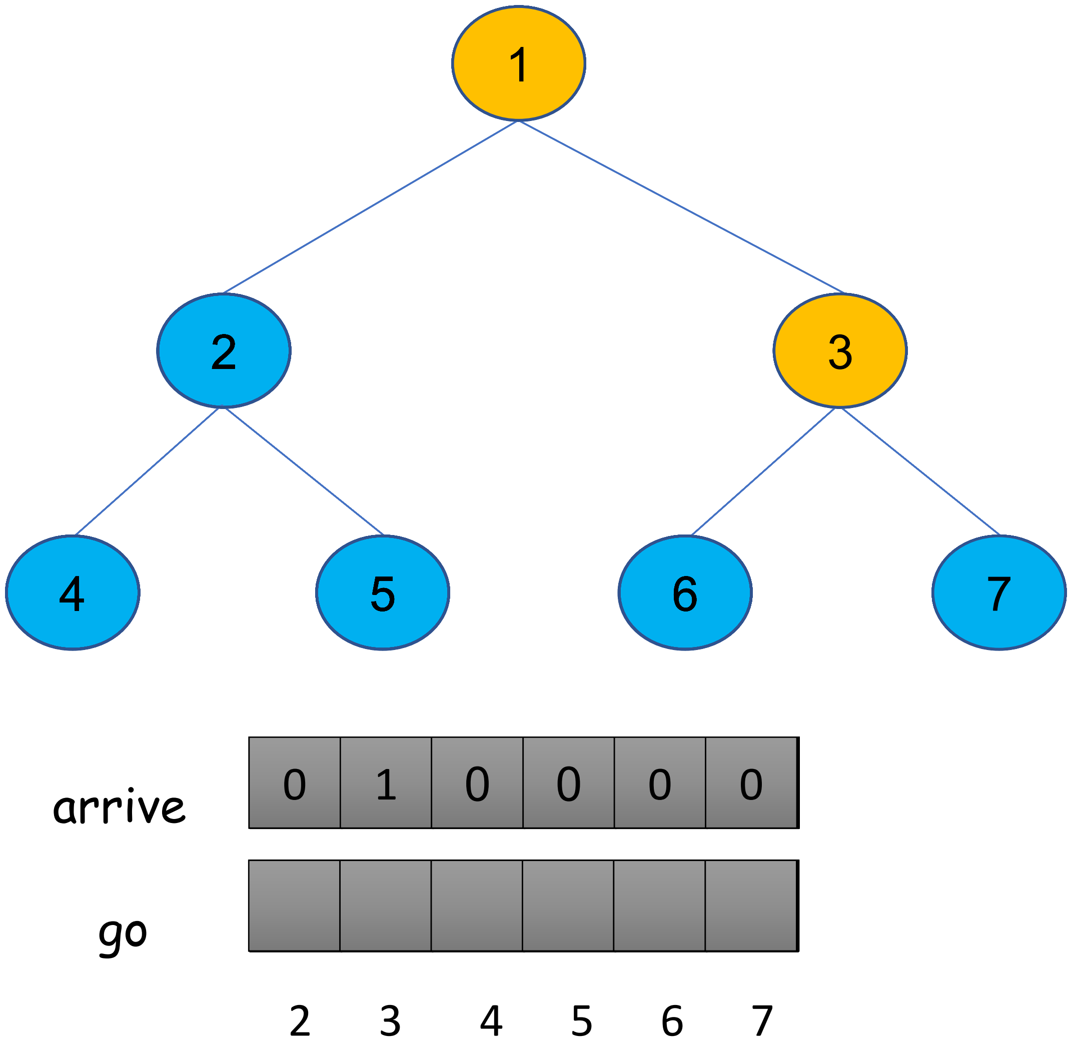
A Tree-based Barrier
Example Run for n=7 threads



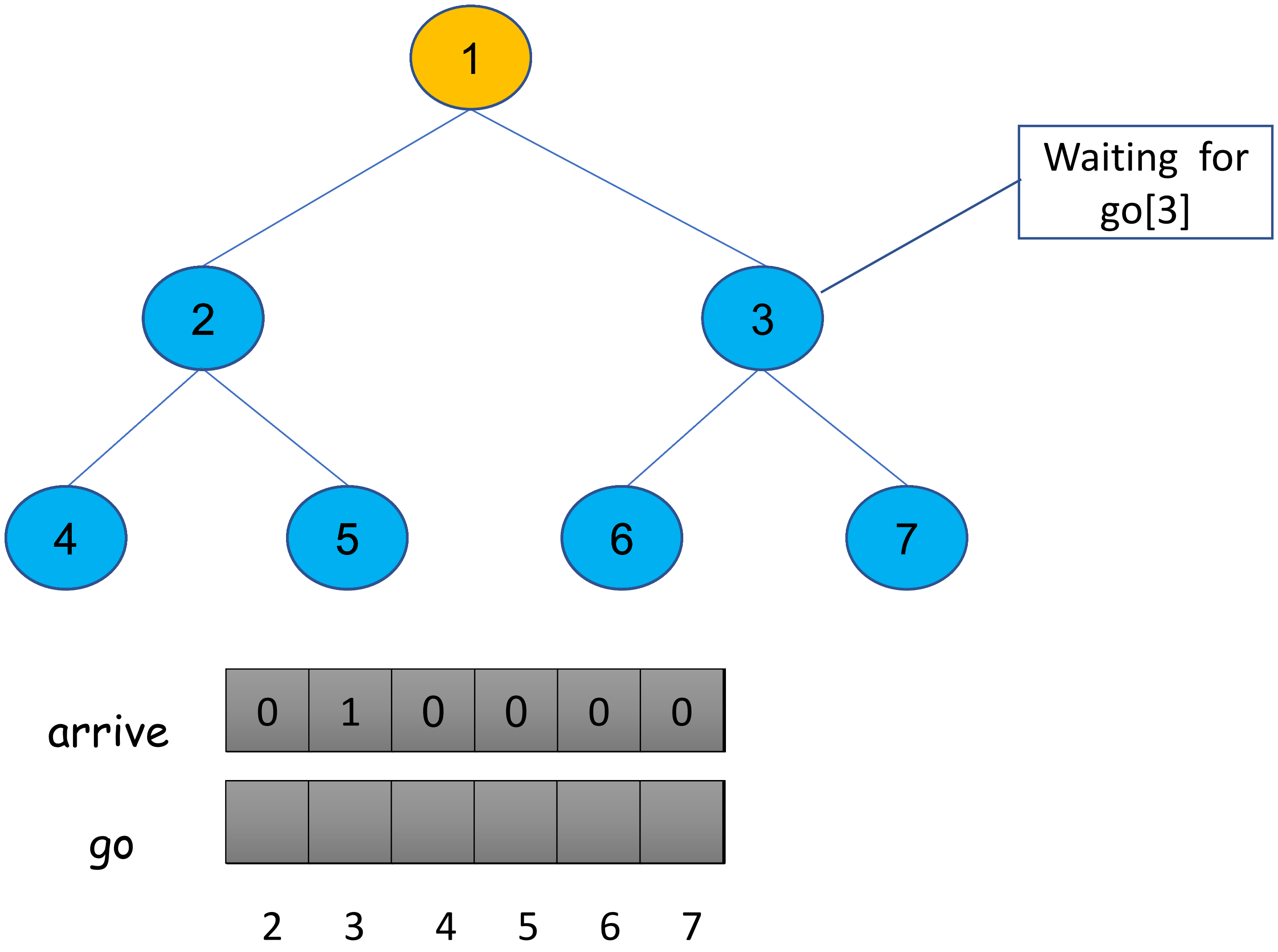

```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

```
shared    arrive[2..n]:  array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
      await(arrive[2]  = 1); arrive[2] := 0
      await(arrive[3]  = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i]  = 1); arrive[2i]  := 0
      await(arrive[2i+1]  = 1); arrive[2i+1]  := 0
      arrive[i]  := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i]  := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```



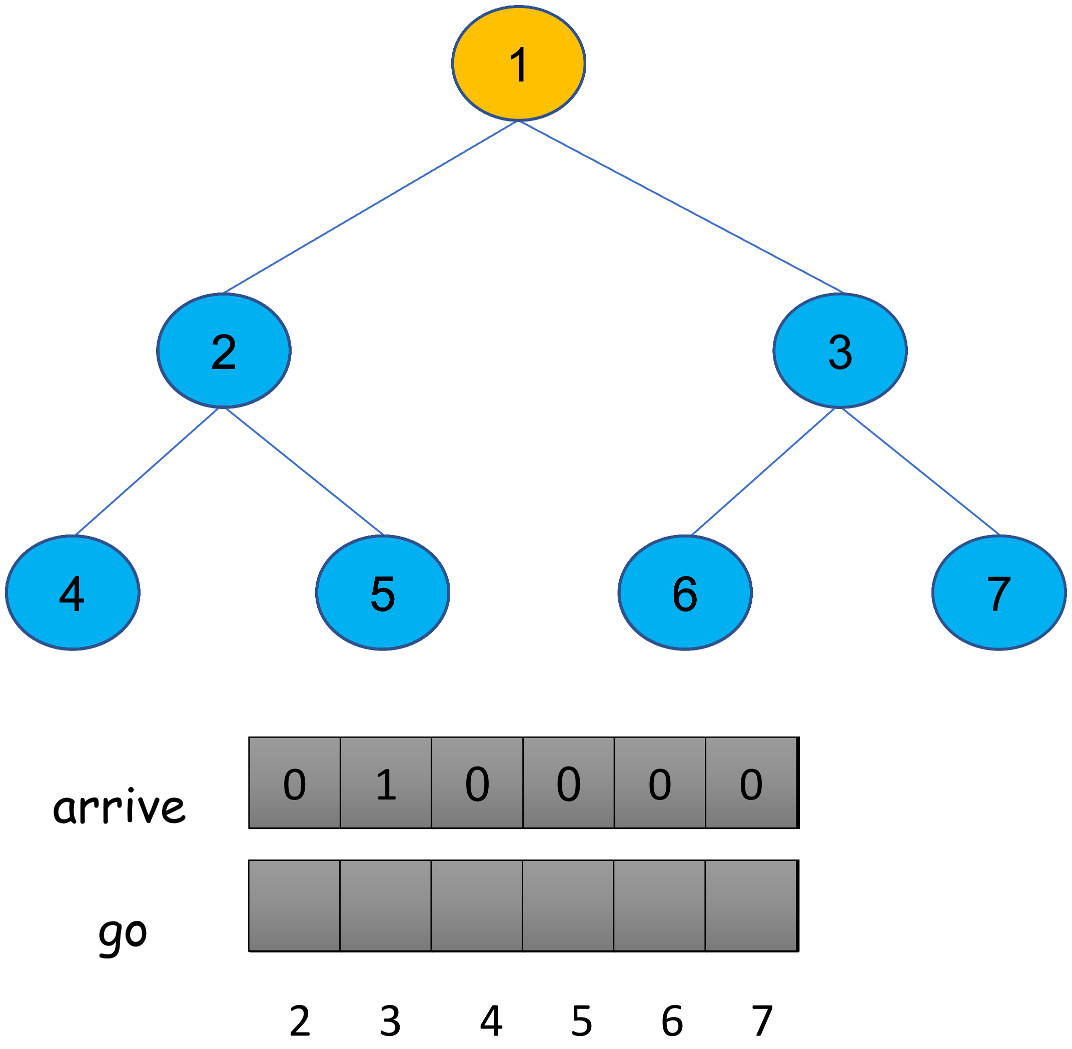
A Tree-based Barrier
Example Run for n=7 threads




```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                         // root
        await(arrive[2] = 1); arrive[2] := 0
        await(arrive[3] = 1); arrive[3] := 0
        go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
        await(arrive[2i] = 1); arrive[2i] := 0
        await(arrive[2i+1] = 1); arrive[2i+1] := 0
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0
        go[2i] = 1; go[2i+1] := 1
else                                                // leaf
        arrive[i] := 1
        await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

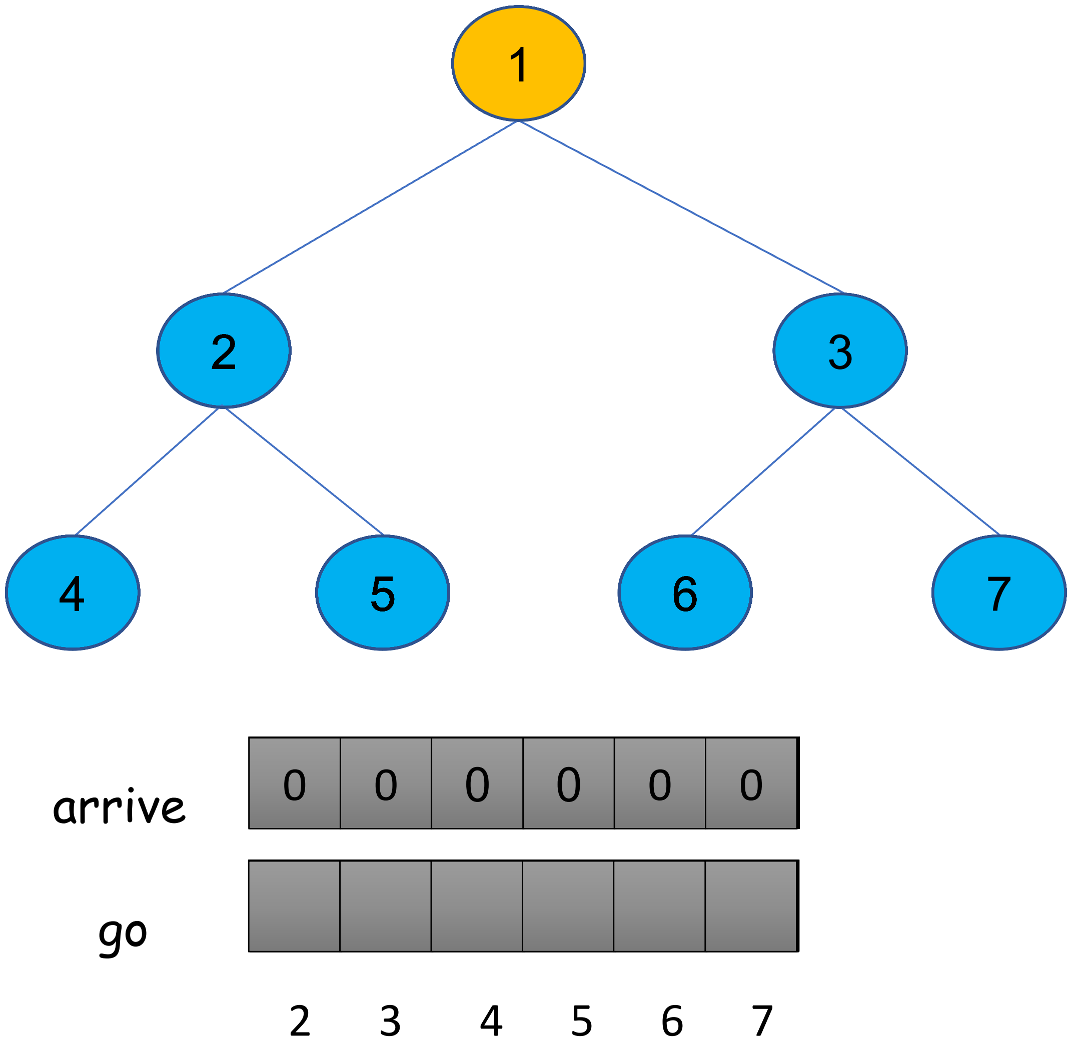



```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                      // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                         // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                             // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

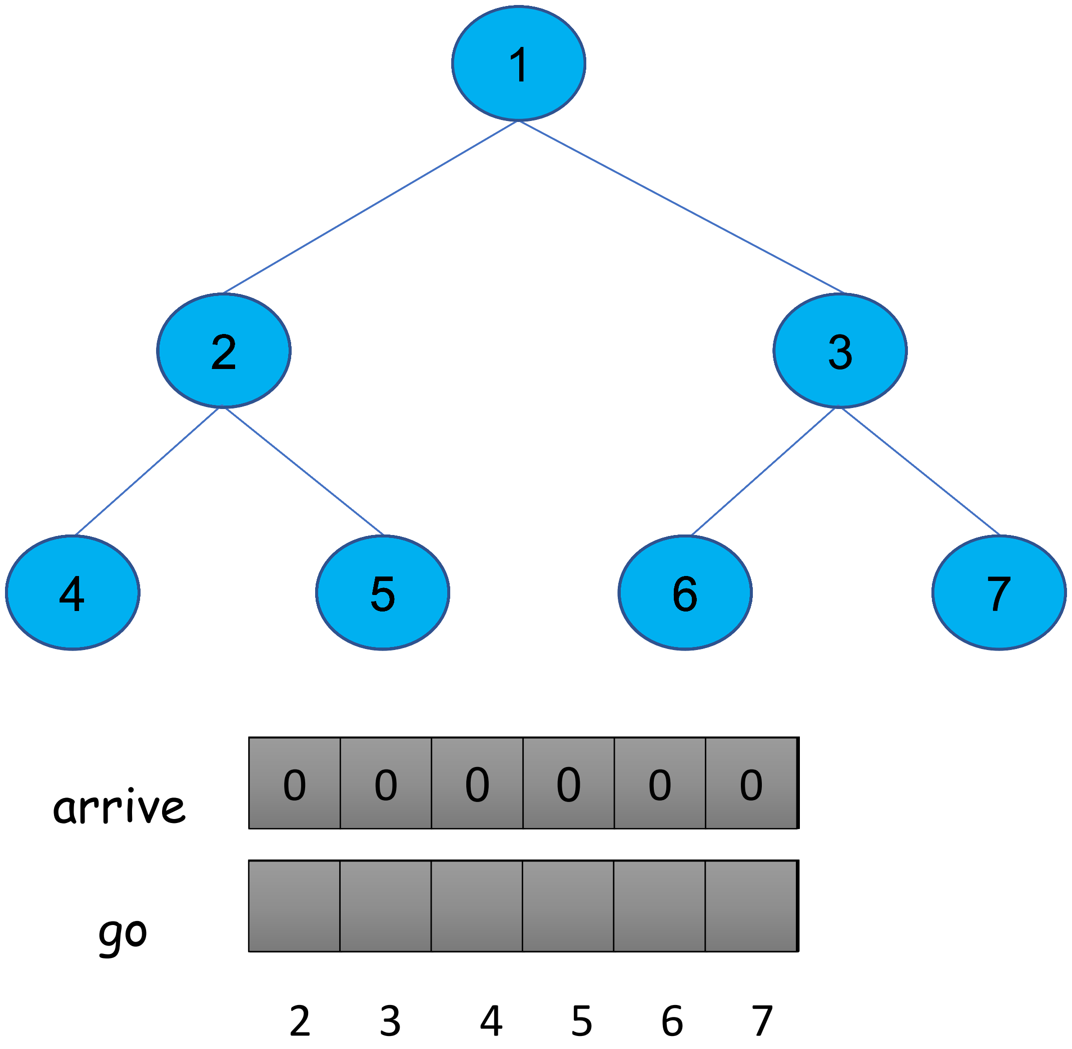
P_1 zeros arrive[3]




```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

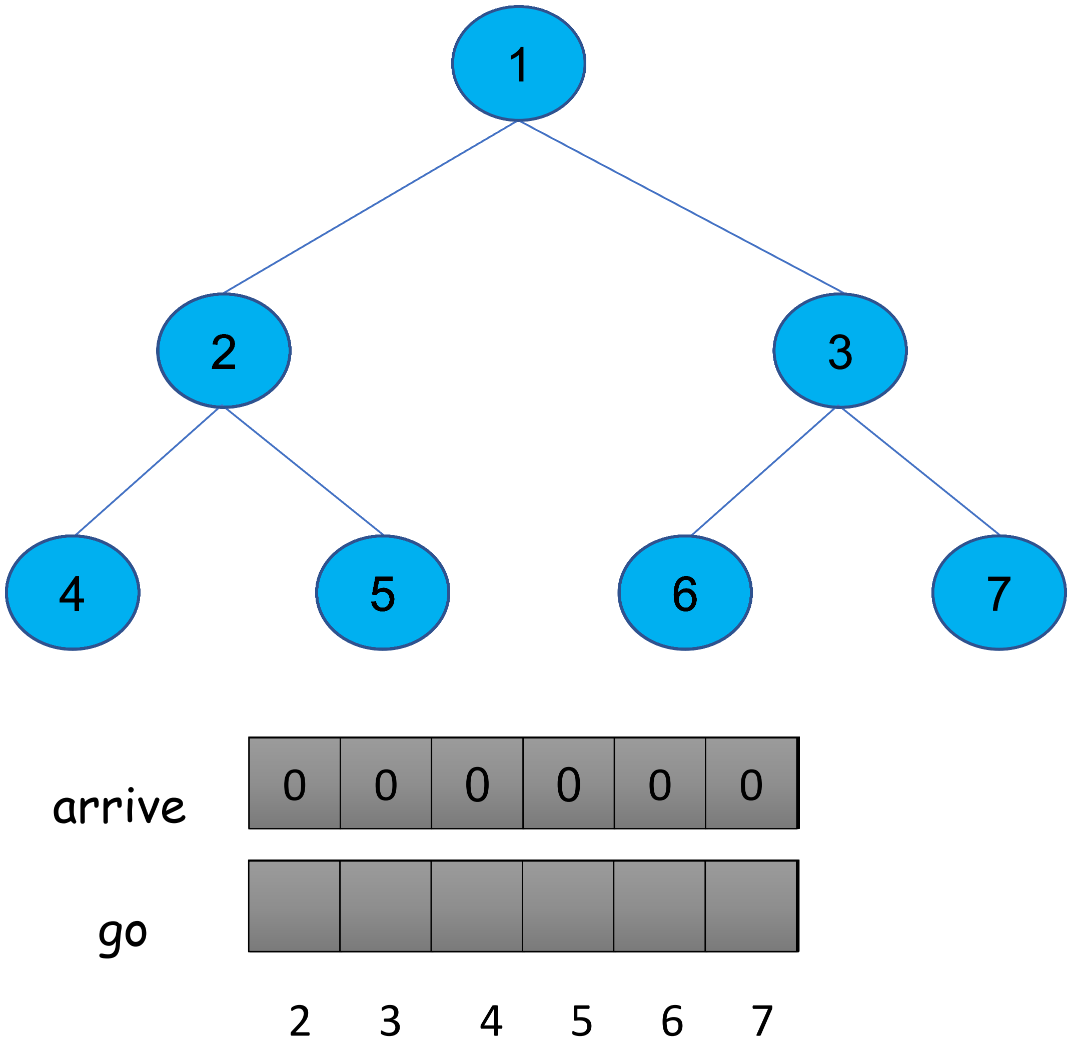
```
if i=1 then                                   // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                      // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                          // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

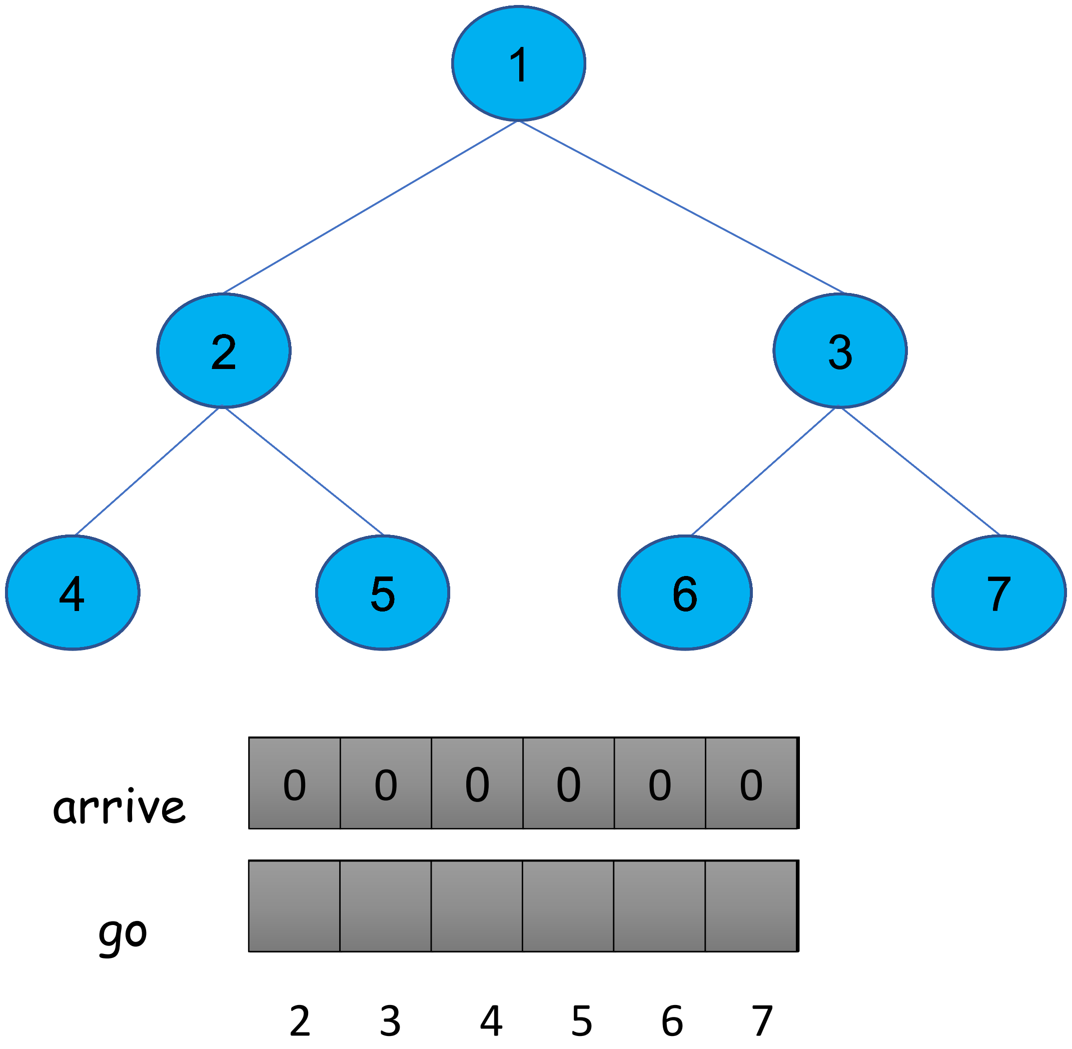
A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                         // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

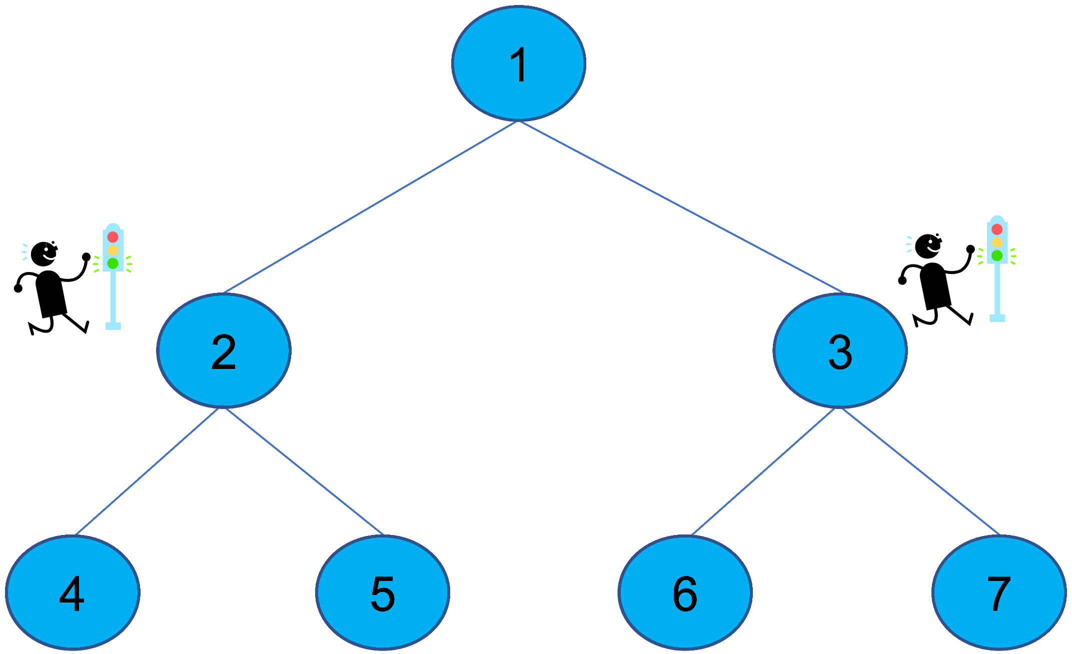
At this point all non-root threads in some await(go) case

A Tree-based Barrier
Example Run for n=7 threads



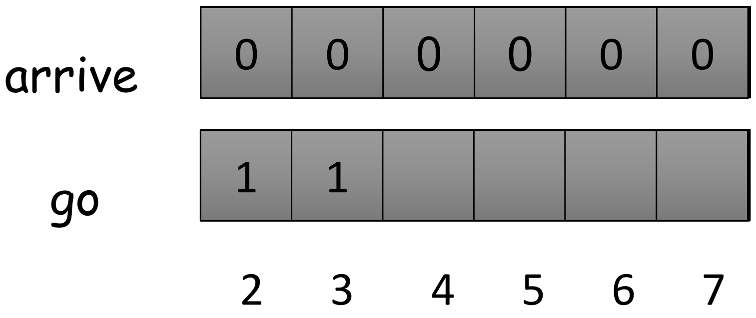

```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0

if i=1 then                                          // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                             // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                 // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads

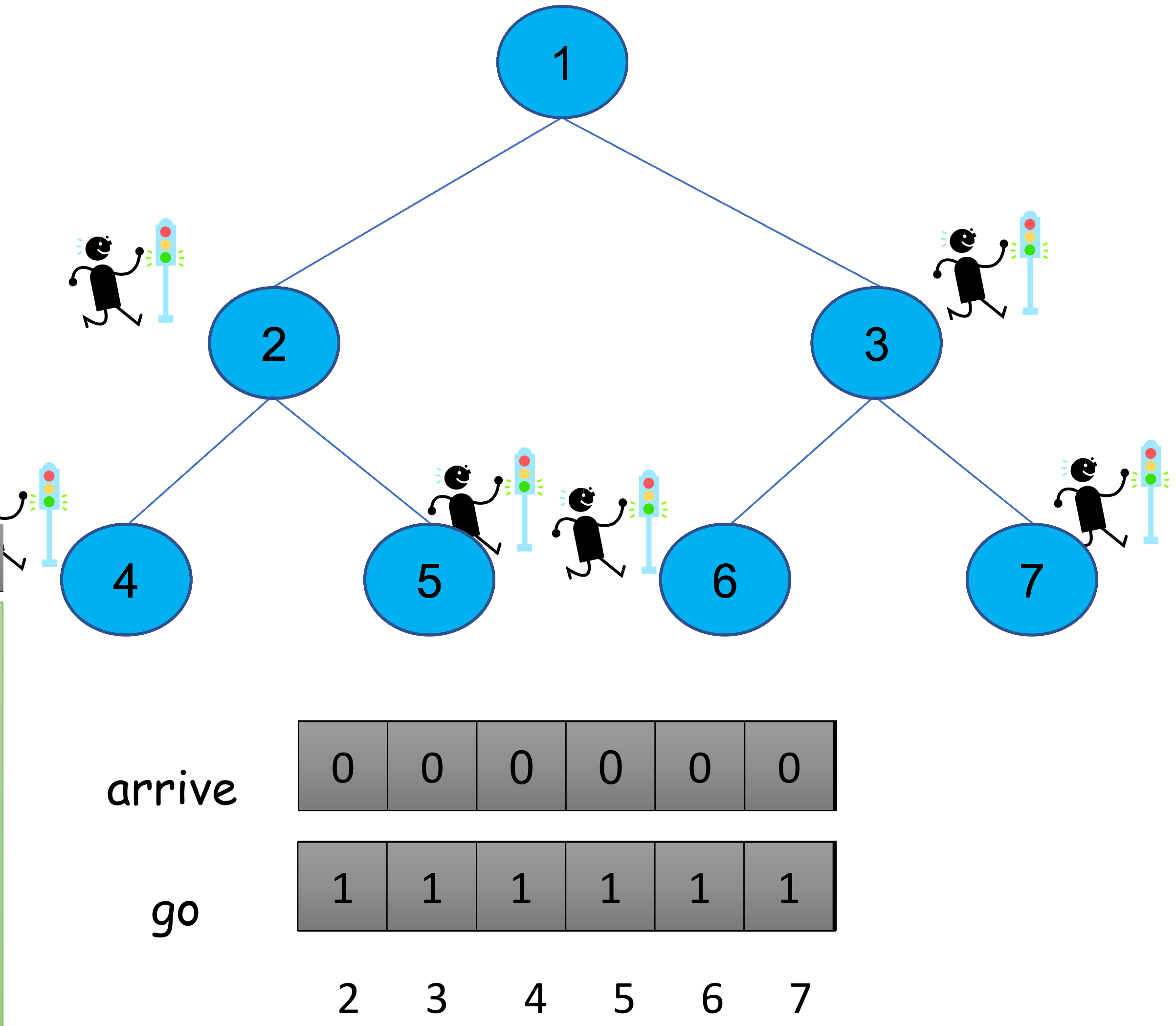



```
shared    arrive[2..n]: array of atomic bits, initial values = 0
          go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                              // root
      await(arrive[2] = 1); arrive[2] := 0
      await(arrive[3] = 1); arrive[3] := 0
      go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                                 // internal node
      await(arrive[2i] = 1); arrive[2i] := 0
      await(arrive[2i+1] = 1); arrive[2i+1] := 0
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0
      go[2i] = 1; go[2i+1] := 1
else                                                     // leaf
      arrive[i] := 1
      await(go[i] = 1); go[i] := 0 fi
fi
```



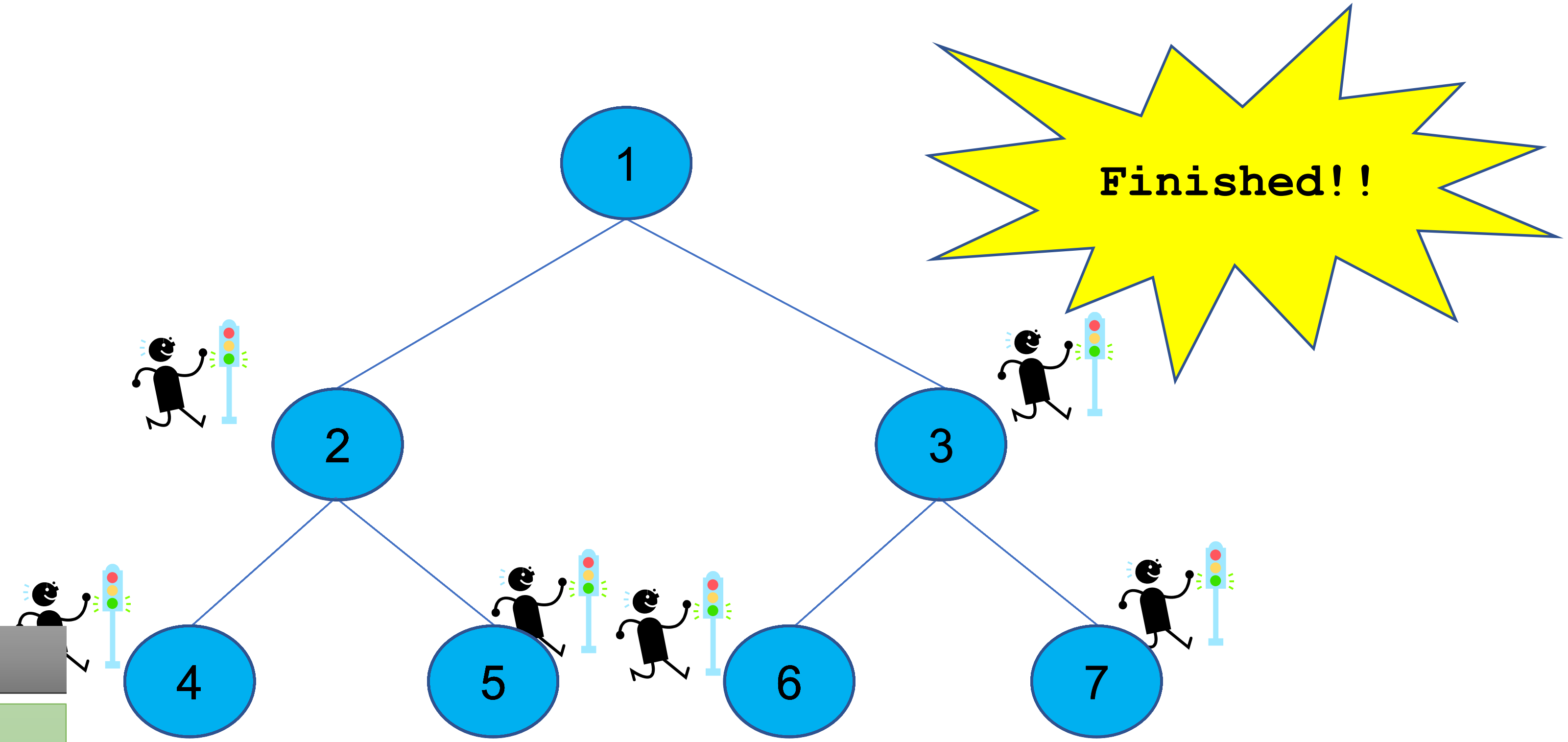
A Tree-based Barrier
Example Run for n=7 threads




```
shared   arrive[2..n]: array of atomic bits, initial values = 0
         go[2..n]: array of atomic bits, initial values = 0
```

```
if i=1 then                                        // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                           // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                               // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

A Tree-based Barrier
Example Run for n=7 threads



shared arrive[2..n]: array of atomic bits, initial values = 0
go[2..n]: array of atomic bits, initial values = 0

```
if i=1 then                                         // root
    await(arrive[2] = 1); arrive[2] := 0
    await(arrive[3] = 1); arrive[3] := 0
    go[2] = 1; go[3] = 1
else if i ≤ (n-1)/2 then                            // internal node
    await(arrive[2i] = 1); arrive[2i] := 0
    await(arrive[2i+1] = 1); arrive[2i+1] := 0
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0
    go[2i] = 1; go[2i+1] := 1
else                                                // leaf
    arrive[i] := 1
    await(go[i] = 1); go[i] := 0 fi
fi
```

arrive	0	0	0	0	0	0
go	1	1	1	1	1	1
	2	3	4	5	6	7

Tree Barrier Tradeoffs

- Pros:

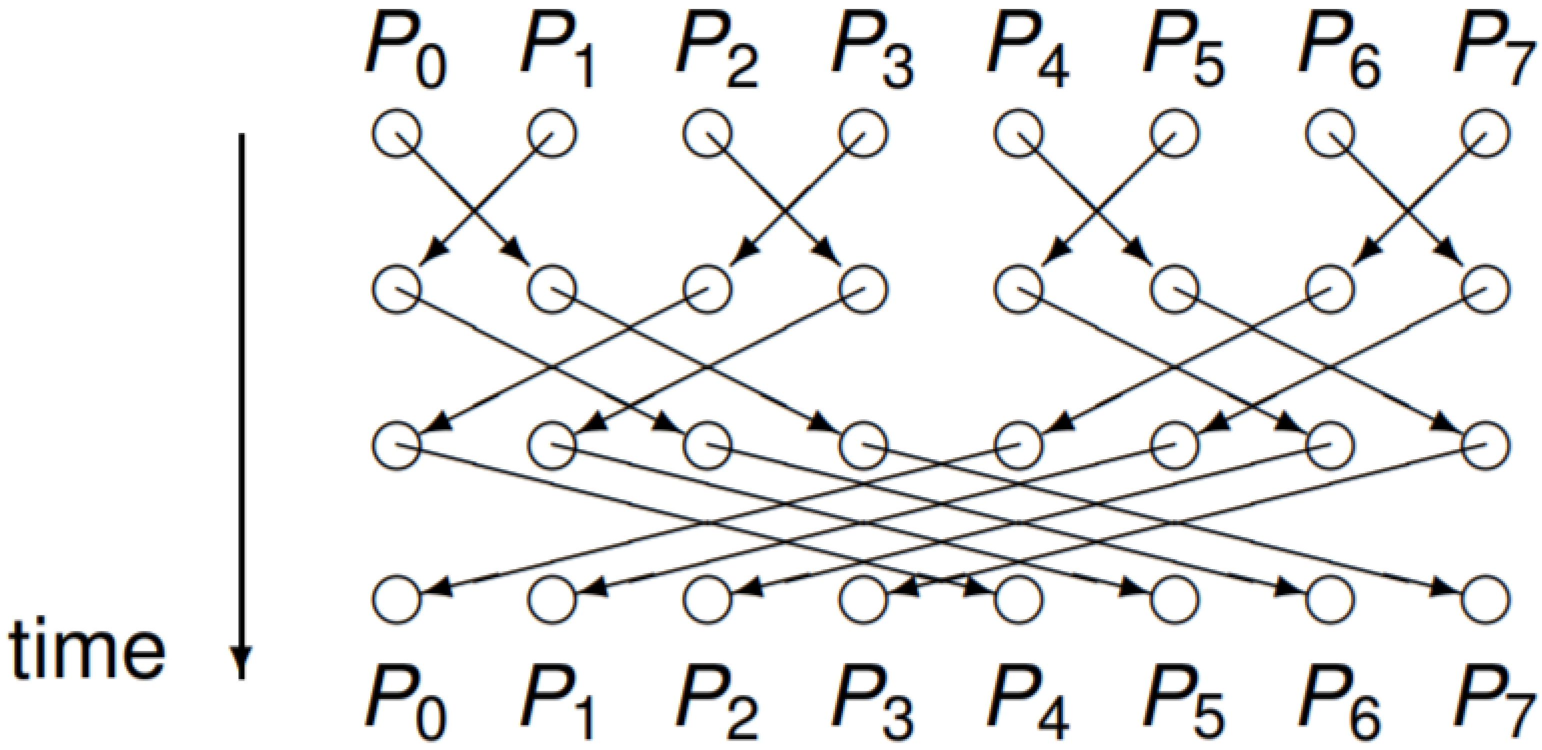
- Cons:

Tree Barrier Tradeoffs

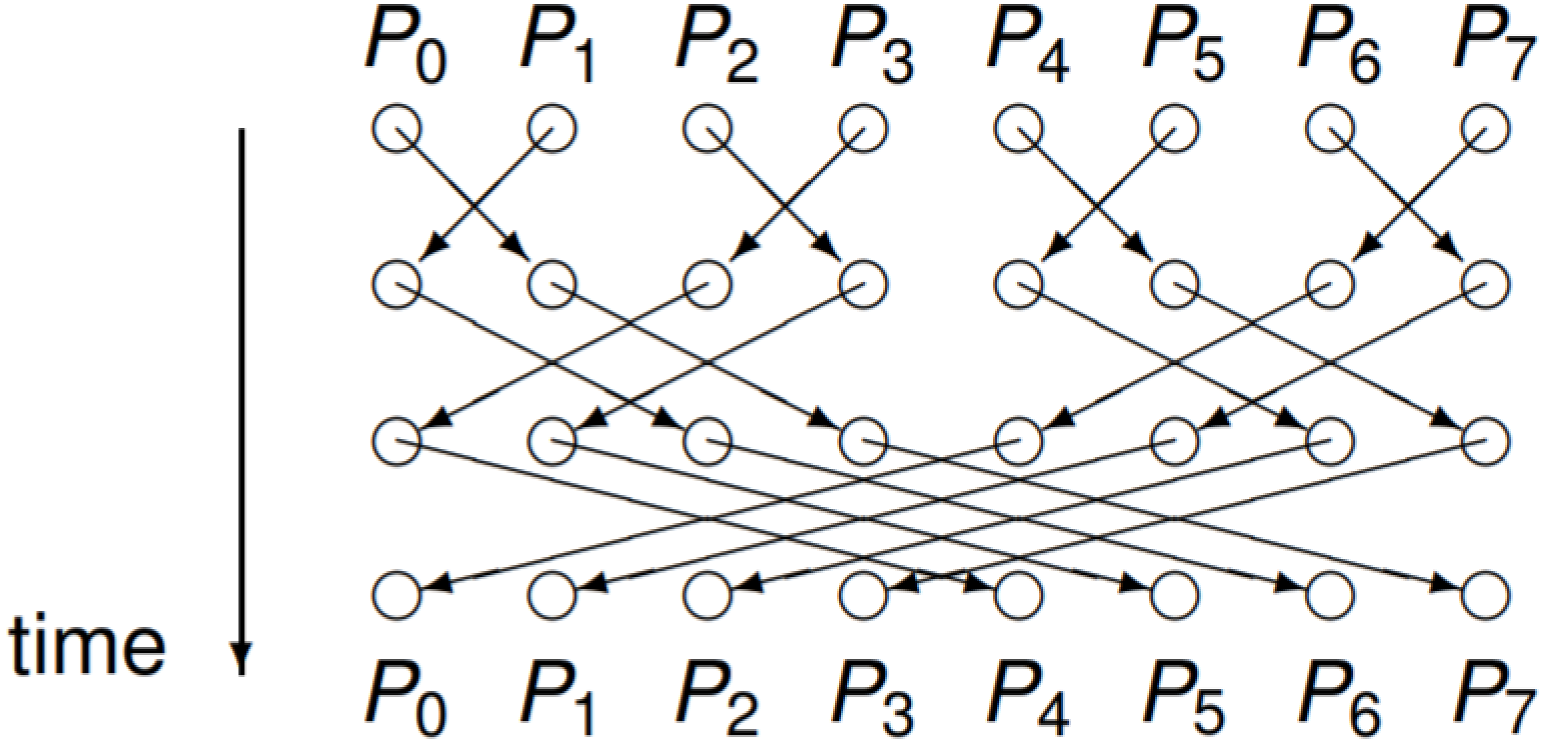
- Pros:
  - Low shared memory contention
    - No wait object is shared by more than 2 processes
    - Good for larger n
  - Fast – information from the root propagates after log(n) steps
  - Can use only atomic primitives (no special objects)
  - On some models:
    - each process spins on a locally accessible bit
    - # (remote memory ref.) = O(1) per process
- Cons:
  - Shared memory space complexity – O(n)
  - Asymmetric –all the processes don't the same amount of work
  - Corner cases for n != 2^k-1

Butterfly Barrier

Butterfly Barrier

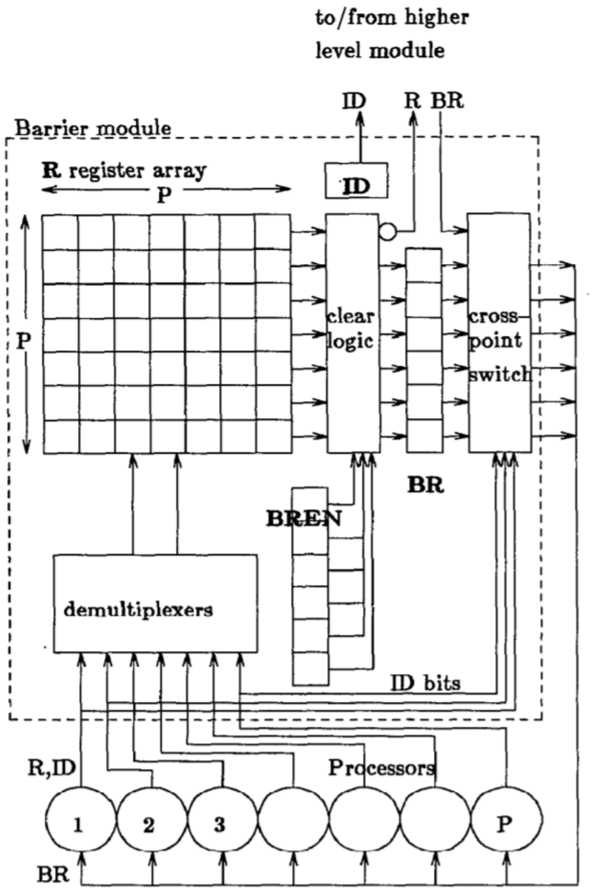


Butterfly Barrier



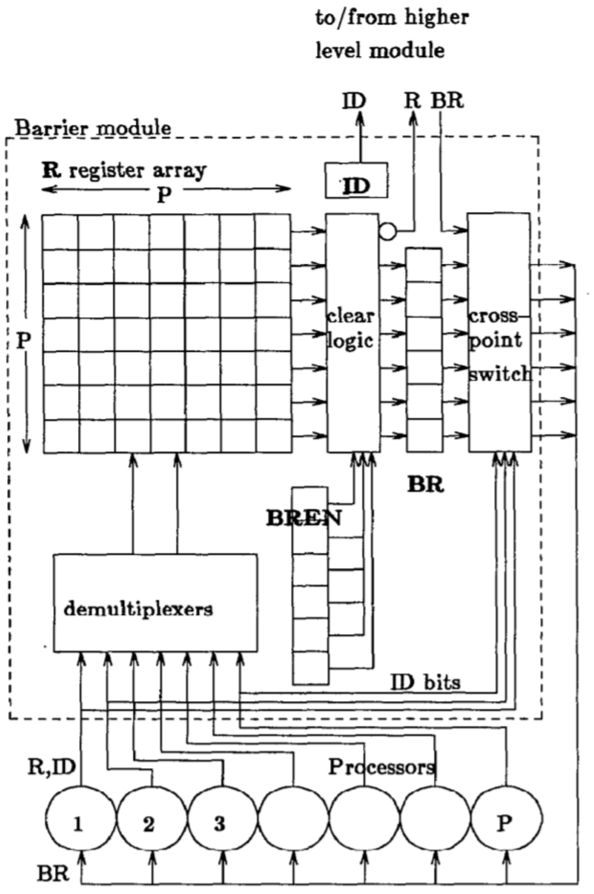
- When would this be preferable?

Hardware Supported Barriers



CPU

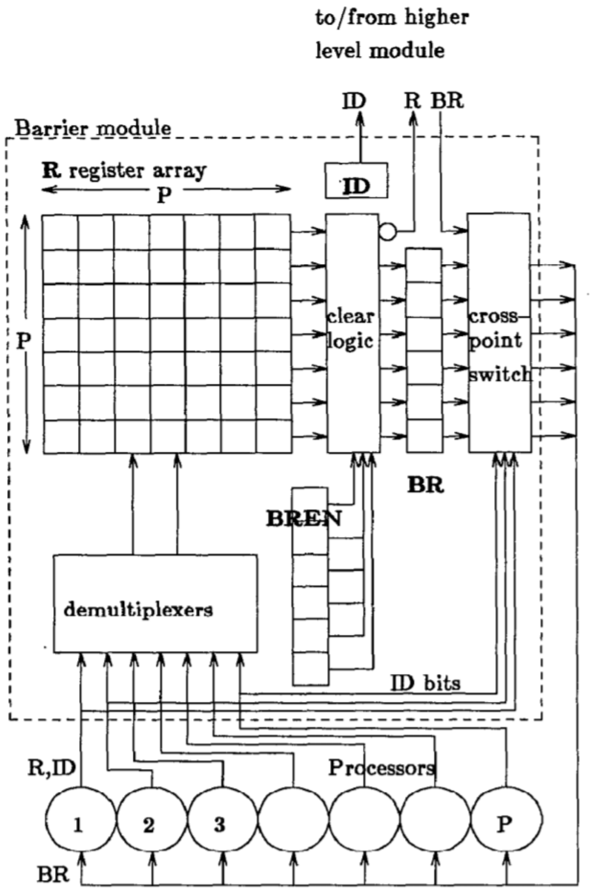
Hardware Supported Barriers



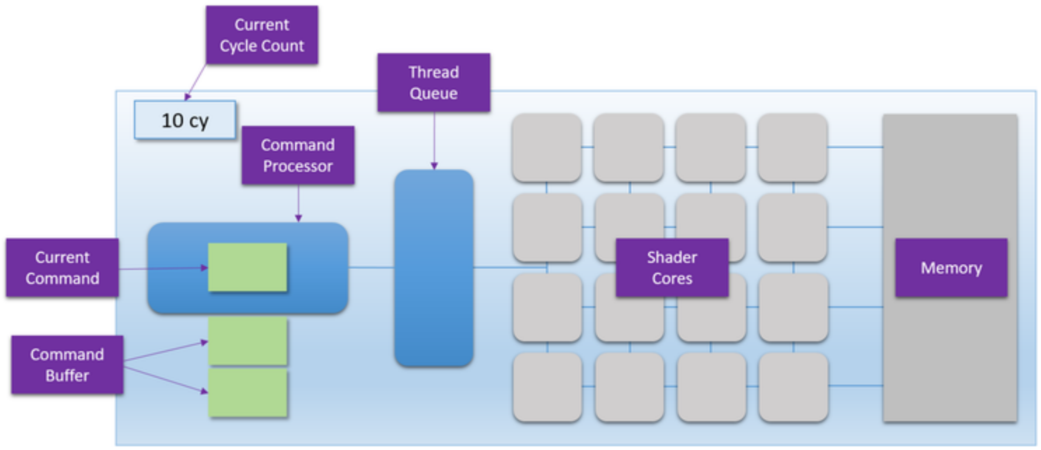
CPU

- When would this be useful?

Hardware Supported Barriers



CPU



GPU

- When would this be useful?

Barriers Summary

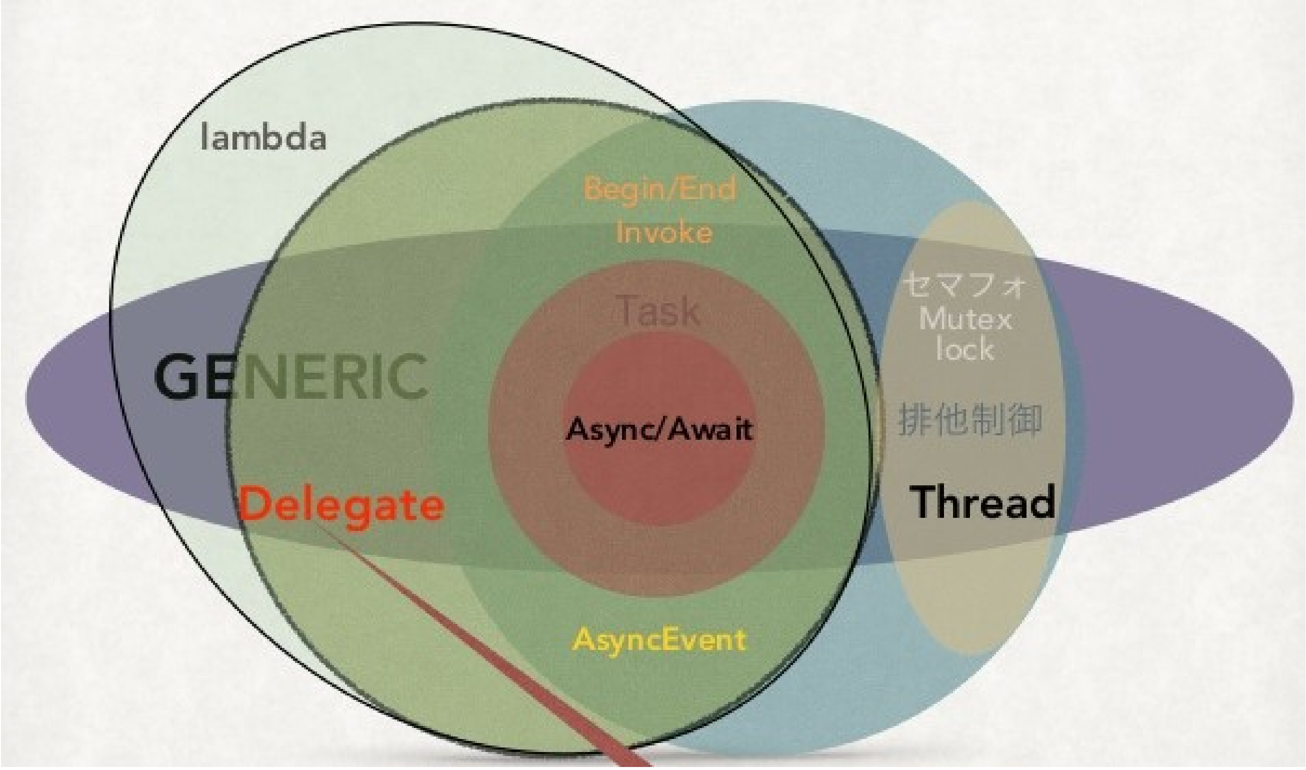
Seen:

- Semaphore-based barrier
- Simple barrier
  - Based on atomic fetch-and-increment counter
- Local spinning barrier
  - Based on atomic fetch-and-increment counter and go array
- Tree-based barrier

Not seen:

- Test-and-Set barriers
  - Based on test-and-test-and-set objects
  - One version without memory initialization
- See-Saw barrier
- Book has condition barriers

Asynchronous Programming
Events, Promises, and Futures



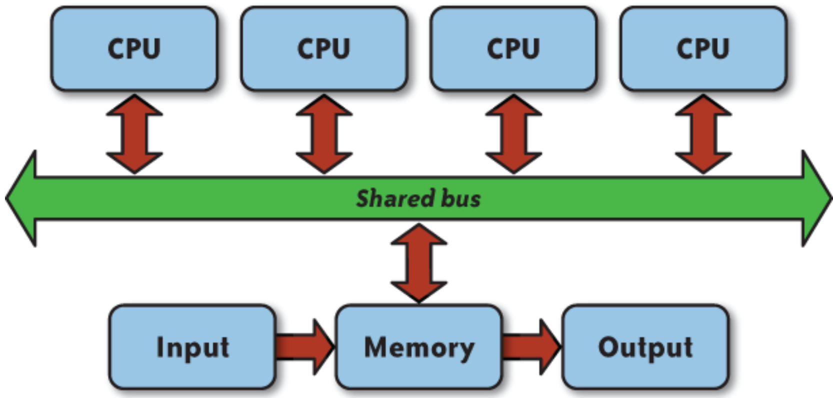
Programming Models for Concurrency

Programming Models for Concurrency

- Hardware execution model:

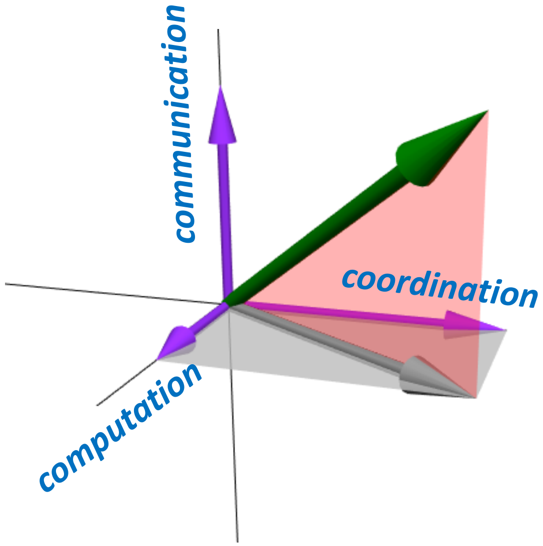
Programming Models for Concurrency

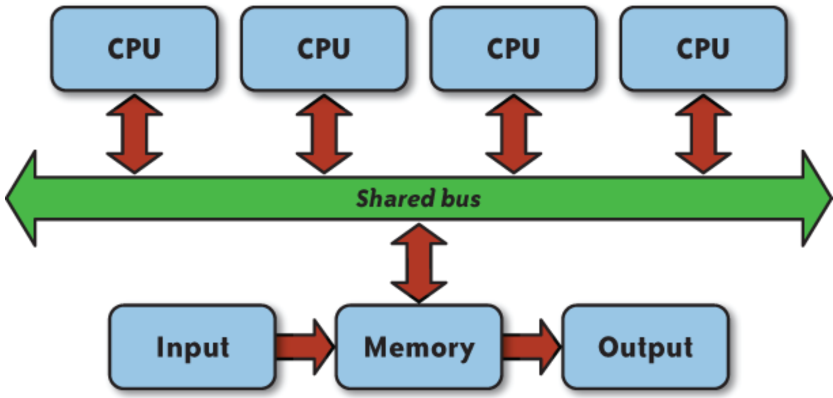
- Hardware execution model:
  - CPU(s) execute instructions sequentially



Programming Models for Concurrency

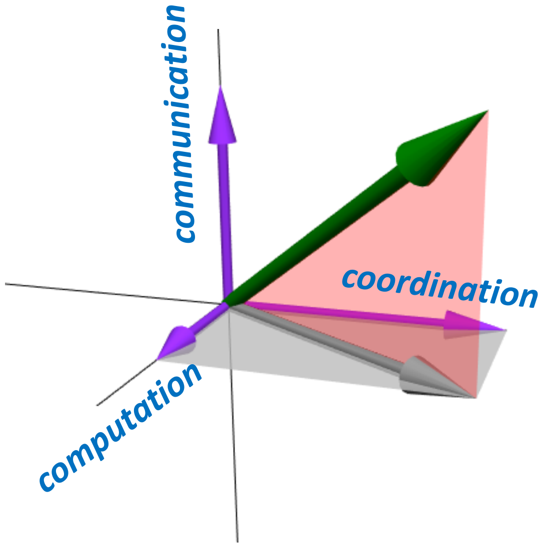
- Hardware execution model:
  - CPU(s) execute instructions sequentially
- Programming model dimensions:
  - How to specify computation
  - How to specify communication
  - How to specify coordination/control transfer

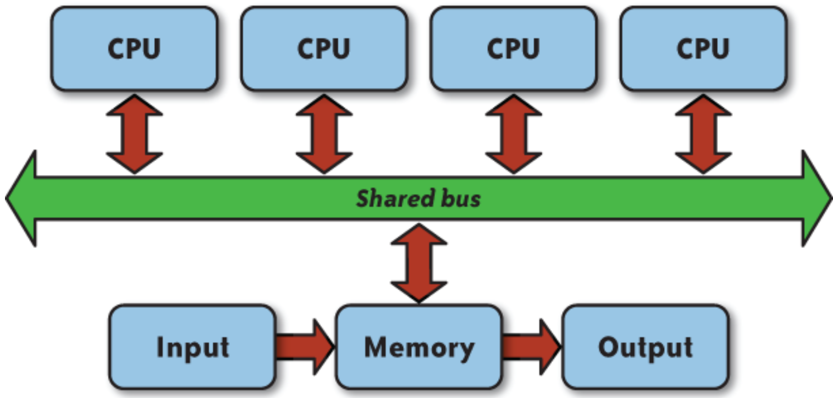




Programming Models for Concurrency

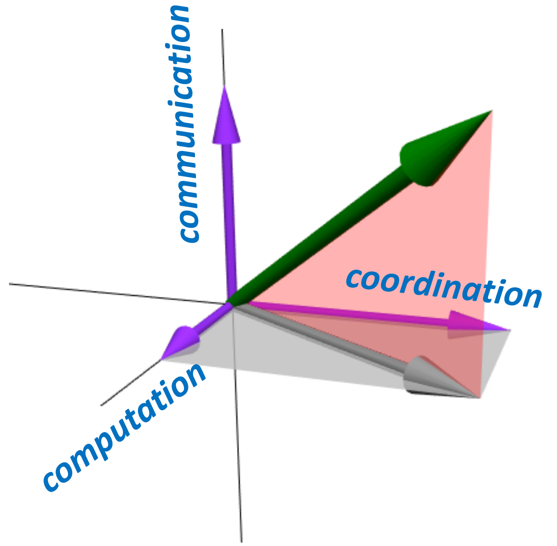
- Hardware execution model:
  - CPU(s) execute instructions sequentially
- Programming model dimensions:
  - How to specify computation
  - How to specify communication
  - How to specify coordination/control transfer
- Techniques/primitives
  - Message passing vs shared memory
  - Preemption vs Non-preemption

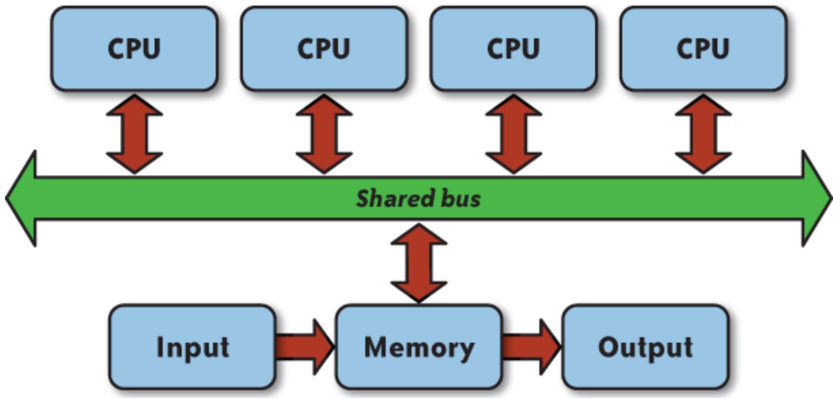




Programming Models for Concurrency

- Hardware execution model:
  - CPU(s) execute instructions sequentially
- Programming model dimensions:
  - How to specify computation
  - How to specify communication
  - How to specify coordination/control transfer
- Techniques/primitives
  - Message passing vs shared memory
  - Preemption vs Non-preemption
- Dimensions/techniques not always orthogonal

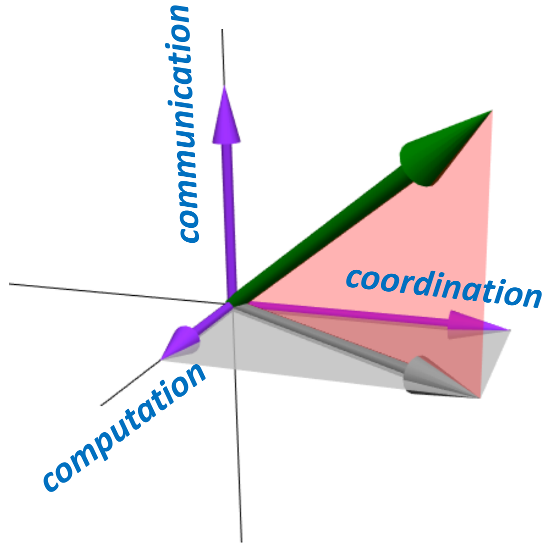


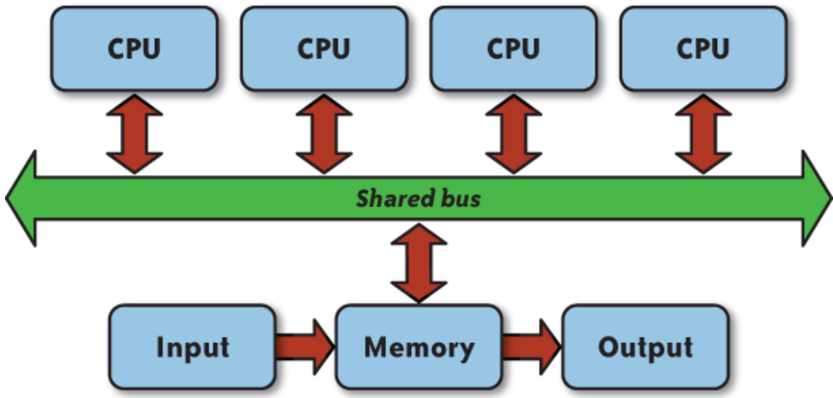


Programming Models for Concurrency

- Hardware execution model:
  - CPU(s) execute instructions sequentially
- Programming model dimensions:
  - How to specify computation
  - How to specify communication
  - How to specify coordination/control transfer
- Techniques/primitives
  - Message passing vs shared memory
  - Preemption vs Non-preemption
- Dimensions/techniques not always orthogonal

Futures & Promises touch all three dimension





Futures & Promises

Futures & Promises

- Values *that will eventually become available*

Futures & Promises

- Values *that will eventually become available*
- Time-dependent states:
  - **Completed/determined**
    - Computation complete, value concrete
  - **Incomplete/undetermined**
    - Computation not complete yet

Futures & Promises

- Values *that will eventually become available*
- Time-dependent states:
  - **Completed/determined**
    - Computation complete, value concrete
  - **Incomplete/undetermined**
    - Computation not complete yet
- Construct (future X)
  - immediately returns value
  - concurrently executes X

Java Example

```java
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

Java Example

```java
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

Java Example

```
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

- CompletableFuture is a container for Future object type

Java Example

```java
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

- CompletableFuture is a container for Future object type
- cf is an instance

Java Example

```java
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

- CompletableFuture is a container for Future object type
- cf is an instance
- runAsync() accepts
  - Lambda expression
  - Anonymous function
  - Functor

Java Example

```
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

- CompletableFuture is a container for Future object type
- cf is an instance
- runAsync() accepts
  - Lambda expression
  - Anonymous function
  - Functor
- runAsync() immediately returns a waitable object (cf)

Java Example

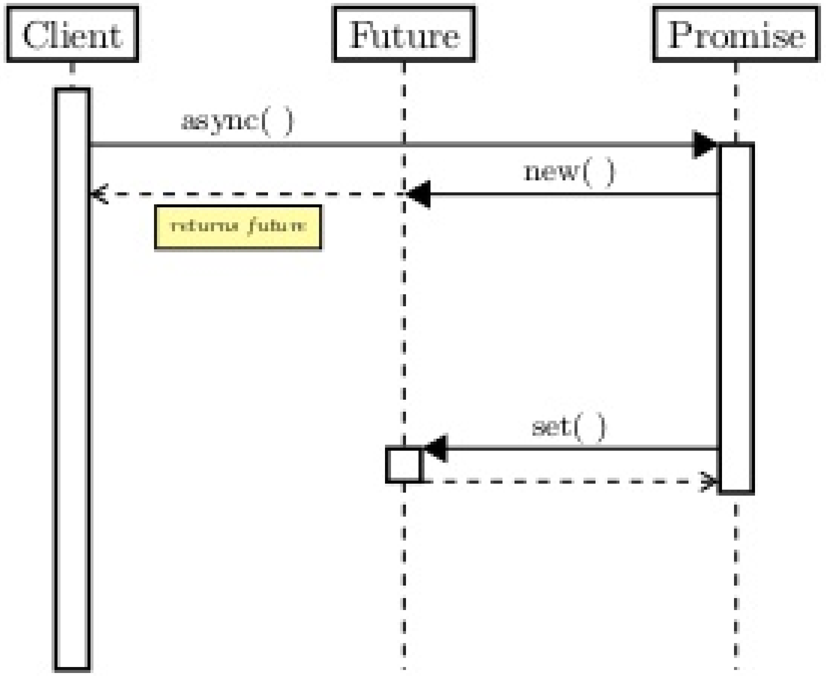
```java
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

- CompletableFuture is a container for Future object type
- cf is an instance
- runAsync() accepts
  - Lambda expression
  - Anonymous function
  - Functor
- runAsync() immediately returns a waitable object (cf)
- Where (on what thread) does the lambda expression run?

Futures and Promises:

Why two kinds of objects?

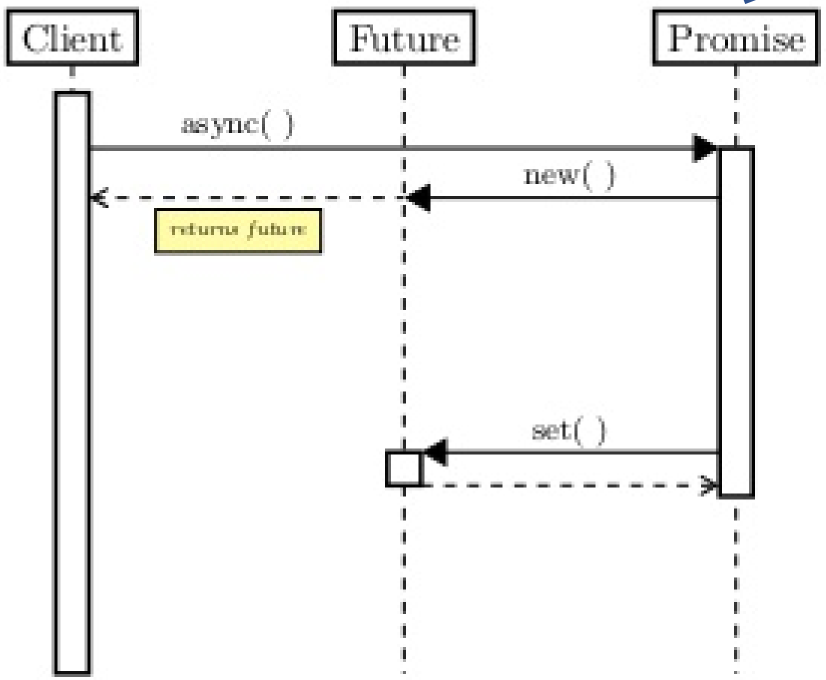
```
future<int> f1 = async(foo1);
...
int result = f1.get();
```



Futures and Promises:

Why two kinds of objects?

Promise: “thing to be done”

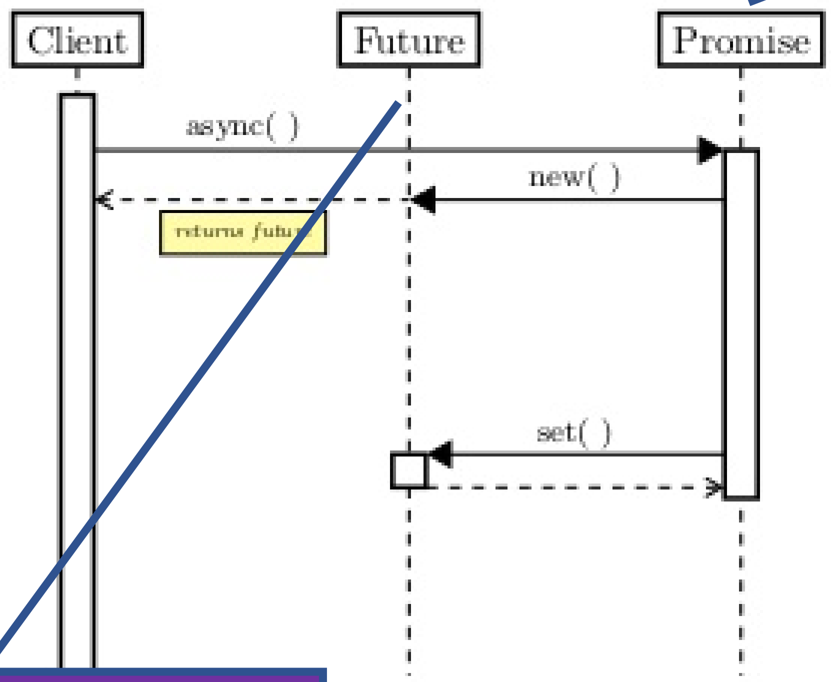



```
future<int> f1 = async(foo1);
...
int result = f1.get();
```

Futures and Promises:

Why two kinds of objects?

Promise: “thing to be done”



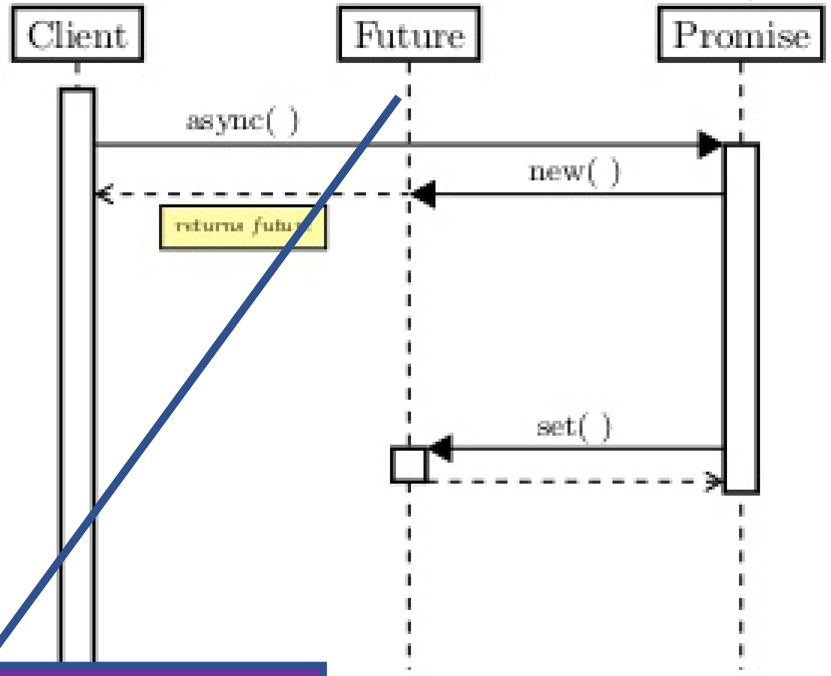

```
future<int> f1 = async(foo1);
...
int result = f1.get();
```

Future: encapsulation
(something to give caller)

Futures and Promises:

Why two kinds of objects?

Promise: “thing to be done”



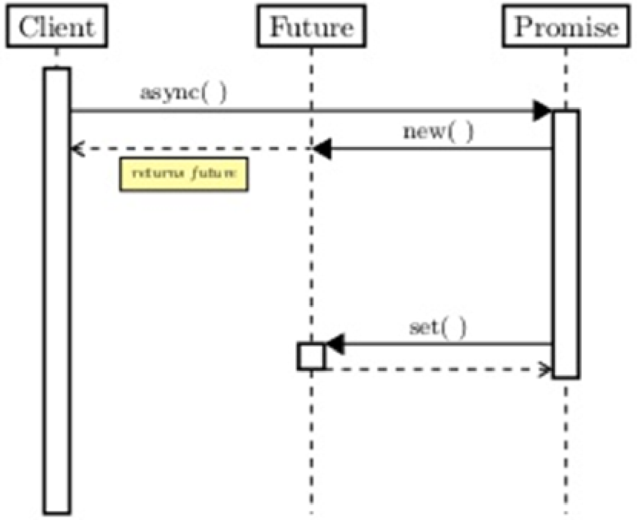

```
future<int> f1 = async(foo1);
...
int result = f1.get();
```

Future: encapsulation
(something to give caller)

Promise to do something in the future

Futures vs Promises

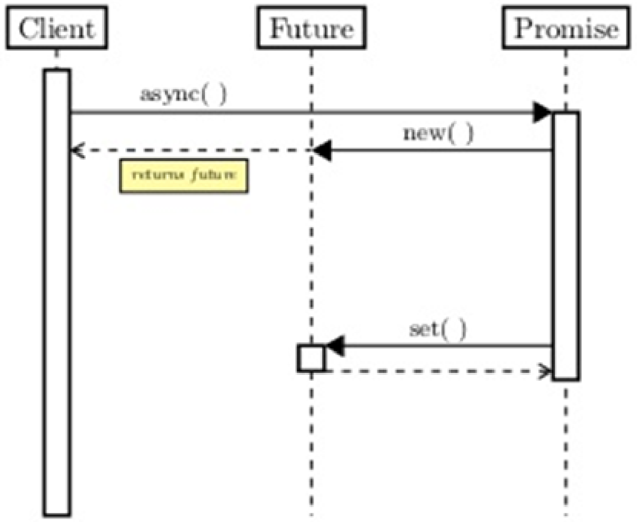
- **Future:** read-only reference to uncompleted value
- **Promise:** single-assignment variable that the future refers to
- Promises *complete* the future with:
  - Result with success/failure
  - Exception



Futures vs Promises

- **Future:** read-only reference to uncompleted value
- **Promise:** single-assignment variable that the future refers to
- Promises *complete* the future with:
  - Result with success/failure
  - Exception

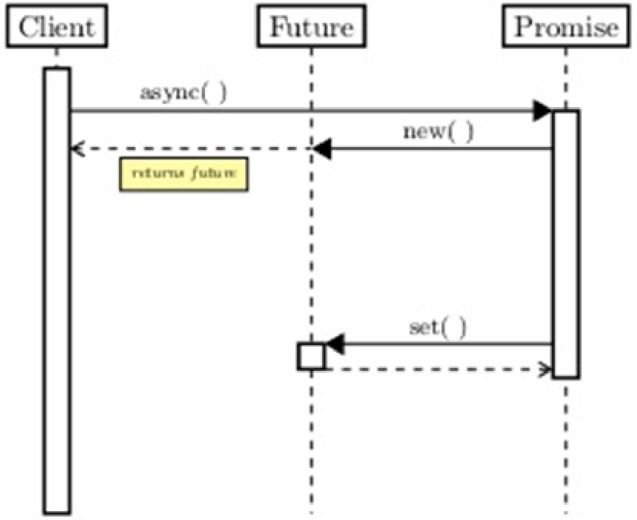
Language	Promise	Future
Algol	Thunk	Address of async result
Java	Future<T>	CompletableFuture<T>
C#/.NET	TaskCompletionSource<T>	Task<T>
JavaScript	Deferred	Promise
C++	std::promise	std::future



Futures vs Promises

- **Future:** read-only reference to uncompleted value
- **Promise:** single-assignment variable that the future refers to
- Promises *complete* the future with:
  - Result with success/failure
  - Exception

Language	Promise	Future
Algol	Thunk	Address of async result
Java	Future<T>	CompletableFuture<T>
C#/.NET	TaskCompletionSource<T>	Task<T>
JavaScript	Deferred	Promise
C++	std::promise	std::future

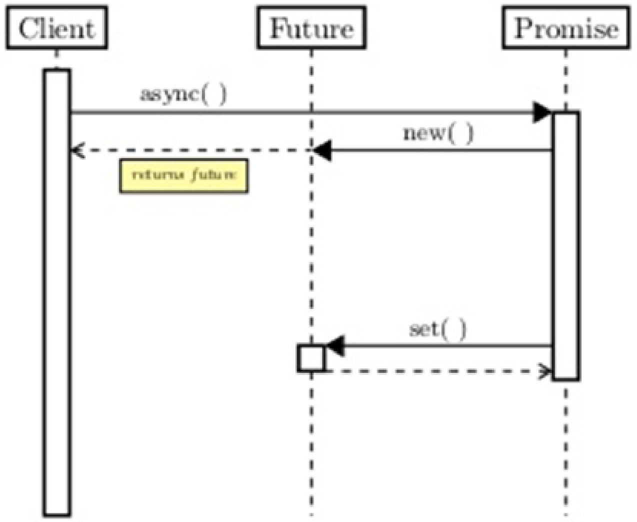


Futures vs Promises

Mnemonic:
Promise to ***do*** something
Make a promise ***for*** the future

- **Future:** read-only reference to uncompleted value
- **Promise:** single-assignment variable that the future refers to
- Promises *complete* the future with:
  - Result with success/failure
  - Exception

Language	Promise	Future
Algol	Thunk	Address of async result
Java	Future<T>	CompletableFuture<T>
C#/.NET	TaskCompletionSource<T>	Task<T>
JavaScript	Deferred	Promise
C++	std::promise	std::future



Putting Futures in Context

My unvarnished opinion

Putting Futures in Context
My unvarnished opinion

Futures:

Putting Futures in Context
My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***

Putting Futures in Context
My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

Putting Futures in Context

My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

```java
static void runAsyncExample() {
    CompletableFuture cf = CompletableFuture.runAsync(() -> {
        assertTrue(Thread.currentThread().isDaemon());
        randomSleep();
    });
    assertFalse(cf.isDone());
    sleepEnough();
    assertTrue(cf.isDone());
}
```

Putting Futures in Context

My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

Putting Futures in Context

My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

Compromise Programming Model between:

Putting Futures in Context

My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

Compromise Programming Model between:

- Event-based programming

Putting Futures in Context

My unvarnished opinion

Futures:

- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

Compromise Programming Model between:

- Event-based programming
- Thread-based programming

Putting Futures in Context

My unvarnished opinion

Futures:

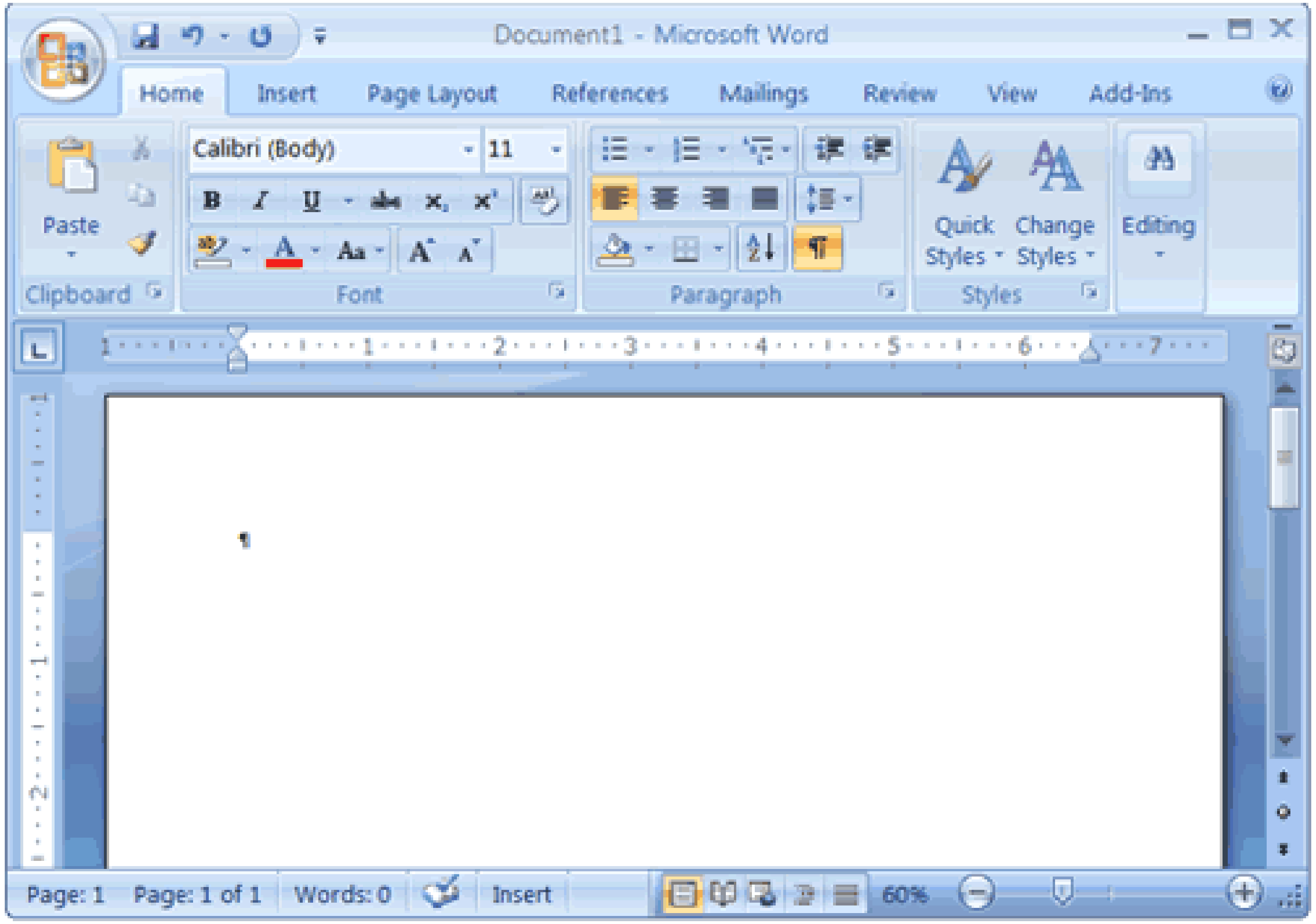
- *abstraction* for concurrent work supported by
  - Compiler: abstractions are ***language-level objects***
  - Runtime: scheduler, task queues, thread-pools are ***transparent***
- Programming remains **mostly** imperative/sequential
  - Threads of control peppered with asynchronous/concurrent tasks

Compromise Programming Model between:

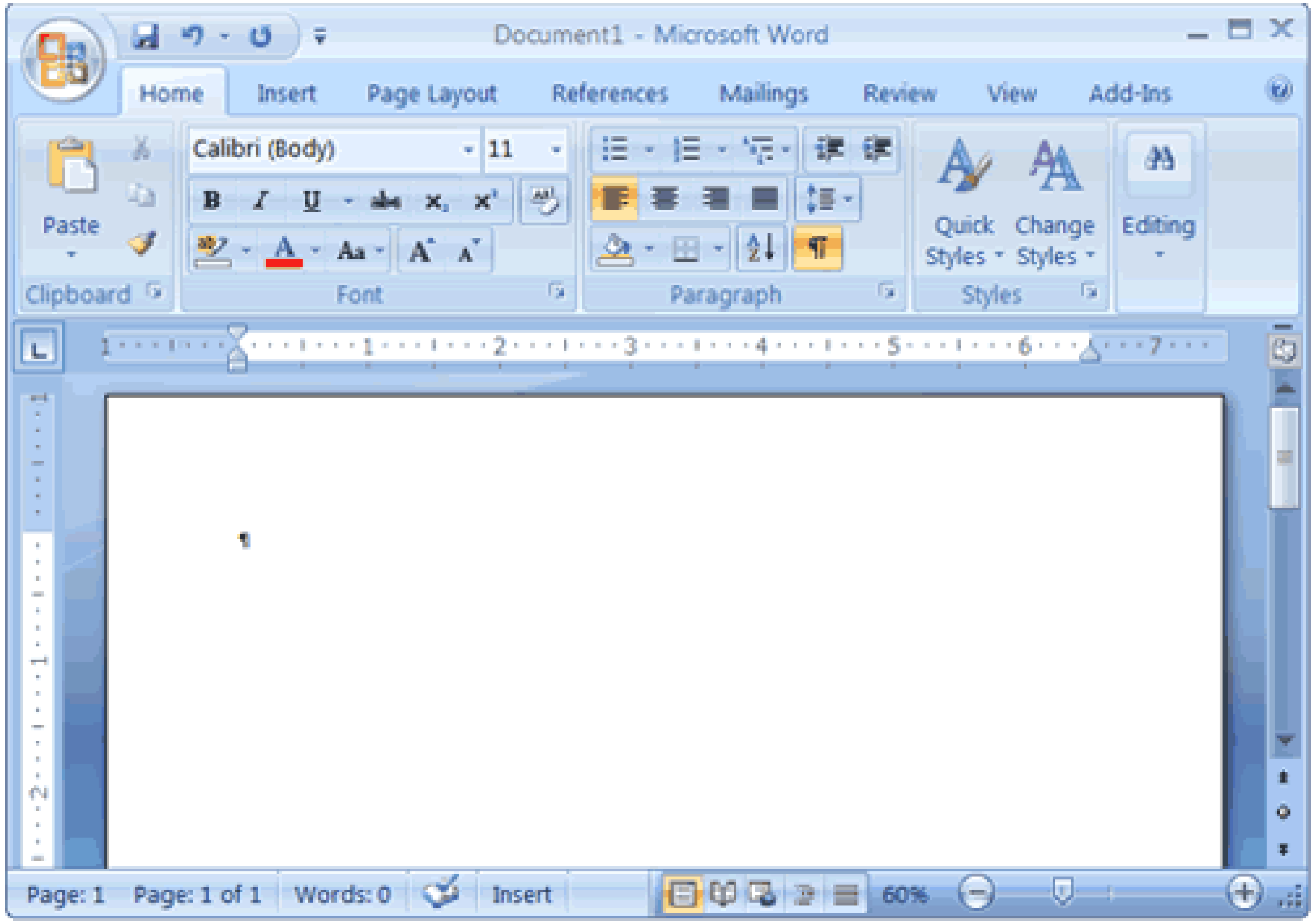
- Event-based programming
- Thread-based programming

Events vs. Threads!

GUI Programming

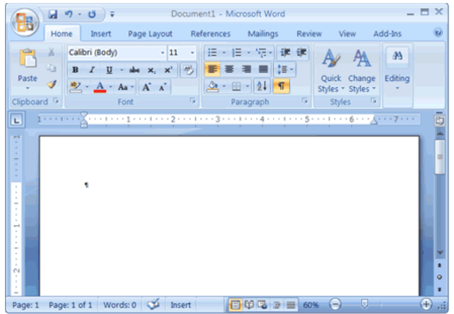


GUI Programming




```
do {
  WaitForSomething();
  RespondToThing();
} until(forever);
```

GUI Programming

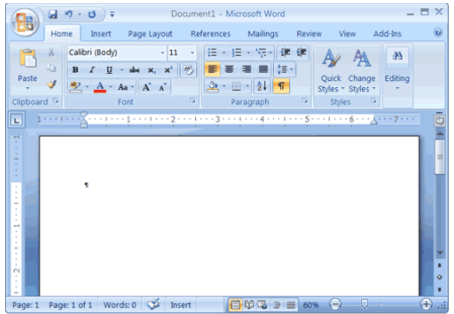



```c
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize        = sizeof(WNDCLASSEX);
    wc.style         = 0;
    wc.lpfnWndProc   = WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInstance;
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Step 2: Creating the Window
    hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);

    if(hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Step 3: The Message Loop
    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```

GUI Programming

```
// Step 2: Creating the Window
hwnd = CreateWindowEx(
    WS_EX_CLIENTEDGE,
    g_szClassName,
    "The title of my window",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
    NULL, NULL, hInstance, NULL);
```

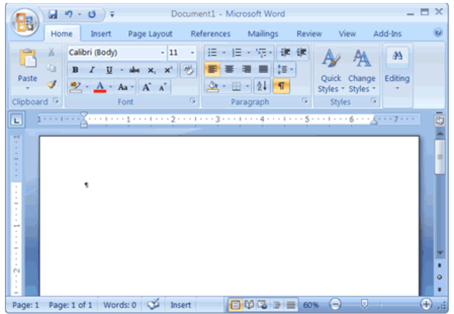
```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize        = sizeof(WNDCLASSEX);
    wc.style         = 0;
    wc.lpfnWndProc   = WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInstance;
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Step 2: Creating the Window
    hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);

    if(hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Step 3: The Message Loop
    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```



GUI Programming

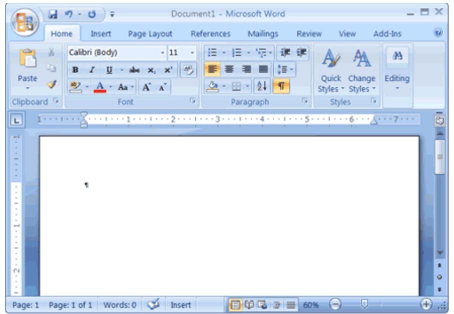
```
// Step 2: Creating the Window
hwnd = CreateWindowEx(
    WS_EX_CLIENTEDGE,
    g_szClassName,
    "The title of my window",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
    NULL, NULL, hInstance, NULL);
```

```
// Step 3: The Message Loop
while(GetMessage(&Msg, NULL, 0, 0) > 0)
{
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
}
```




```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize        = sizeof(WNDCLASSEX);
    wc.style         = 0;
    wc.lpfnWndProc   = WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInstance;
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Step 2: Creating the Window
    hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);

    if(hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Step 3: The Message Loop
    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```

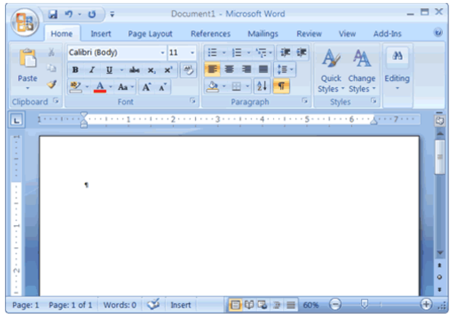
GUI Programming




```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize        = sizeof(WNDCLASSEX);
    wc.style         = 0;
    wc.lpfnWndProc   = WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInstance;
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Step 2: Creating the Window
    hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);

    if(hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Step 3: The Message Loop
    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```

GUI Programming

```
switch (message)
{
        //case WM_COMMAND:
          // handle menu selections etc.
        //break;
        //case WM_PAINT:
          // draw our window - note: you must paint something here or not trap it!
        //break;
        case WM_DESTROY:
             PostQuitMessage(0);
        break;
        default:
            // We do not want to handle this message so pass back to Windows
            // to handle it in a default way
            return DefWindowProc(hWnd, message, wParam, lParam);
}
```




```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize        = sizeof(WNDCLASSEX);
    wc.style         = 0;
    wc.lpfnWndProc   = WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInstance;
    wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    // Step 2: Creating the Window
    hwnd = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        g_szClassName,
        "The title of my window",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
        NULL, NULL, hInstance, NULL);

    if(hwnd == NULL)
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Step 3: The Message Loop
    while(GetMessage(&Msg, NULL, 0, 0) > 0)
    {
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```

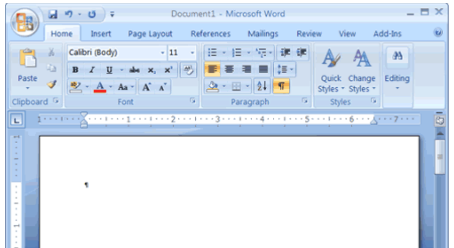
```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;
```

GUI programming

```
switch (message)
{
        //case WM_COMMAND:
          // handle menu selections etc.
        //break;
        //case WM_PAINT:
          // draw our window - note: you must paint something here or not
        //break;
        case WM_DESTROY:
             PostQuitMessage(0);
        break;
        default:
            // We do not want to handle this message so pass back to Winc
            // to handle it in a default way
            return DefWindowProc(hWnd, message, wParam, lParam);
}
```

Hex	Decimal	Symbolic
0000	0	WM_NULL
0001	1	WM_CREATE
0002	2	WM_DESTROY
0003	3	WM_MOVE
0005	5	WM_SIZE
0006	6	WM_ACTIVATE
0007	7	WM_SETFOCUS
0008	8	WM_KILLFOCUS
000a	10	WM_ENABLE
000b	11	WM_SETREDRAW
000c	12	WM_SETTEXT
000d	13	WM_GETTEXT
000e	14	WM_GETTEXTLENGTH
000f	15	WM_PAINT
0010	16	WM_CLOSE
0011	17	WM_QUERYENDSESSION
0012	18	WM_QUIT
0013	19	WM_QUERYOPEN
0014	20	WM_ERASEBKGND

```
ION);
ION);
ed!", "Error!",
, "Error!",
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```



```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;
```

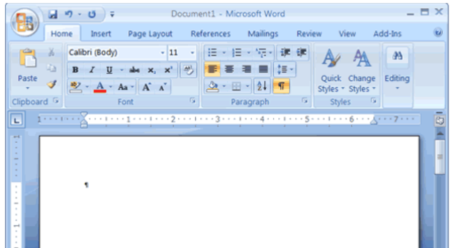
GUI programming

Over 1000 last time I checked!

```
switch (message)
{
        //case WM_COMMAND:
          // handle menu selections etc.
        //break;
        //case WM_PAINT:
          // draw our window - note: you must paint something here or not
        //break;
        case WM_DESTROY:
             PostQuitMessage(0);
        break;
        default:
            // We do not want to handle this message so pass back to Winc
            // to handle it in a default way
            return DefWindowProc(hWnd, message, wParam, lParam);
}
```

Hex	Decimal	Symbolic
0000	0	WM_NULL
0001	1	WM_CREATE
0002	2	WM_DESTROY
0003	3	WM_MOVE
0005	5	WM_SIZE
0006	6	WM_ACTIVATE
0007	7	WM_SETFOCUS
0008	8	WM_KILLFOCUS
000a	10	WM_ENABLE
000b	11	WM_SETREDRAW
000c	12	WM_SETTEXT
000d	13	WM_GETTEXT
000e	14	WM_GETTEXTLENGTH
000f	15	WM_PAINT
0010	16	WM_CLOSE
0011	17	WM_QUERYENDSESSION
0012	18	WM_QUIT
0013	19	WM_QUERYOPEN
0014	20	WM_ERASEBKGND

```
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```



GUI programming

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;
```

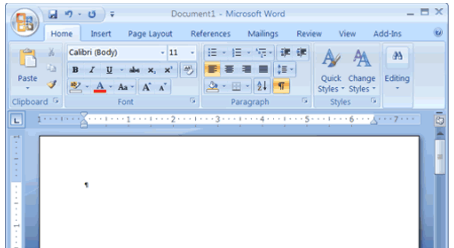
Over 1000 last time I checked!

```
void OnMove() { … }
void OnSize() { … }

void OnPaint() { … }
```

```
switch (message)
{
        //case WM_COMMAND:
          // handle menu select
        //break;
        //case WM_PAINT:
          // draw our window -                    r not
        //break;
        case WM_DESTROY:
             PostQuitMessage(0);
        break;
        default:
            // We do not want to handle this message so pass back to Winc
            // to handle it in a default way
            return DefWindowProc(hWnd, message, wParam, lParam);
}
```

Hex	Decimal	Symbolic
0000	0	WM_NULL
0001	1	WM_CREATE
0002	2	WM_DESTROY
0003	3	WM_MOVE
0005	5	WM_SIZE
0006	6	WM_ACTIVATE
0007	7	WM_SETFOCUS
0008	8	WM_KILLFOCUS
000a	10	WM_ENABLE
000b	11	WM_SETREDRAW
000c	12	WM_SETTEXT
000d	13	WM_GETTEXT
000e	14	WM_GETTEXTLENGTH
000f	15	WM_PAINT
0010	16	WM_CLOSE
0011	17	WM_QUERYENDSESSION
0012	18	WM_QUIT
0013	19	WM_QUERYOPEN
0014	20	WM_ERASEBKGND

```
ION);
ION);
ed!", "Error!",
, "Error!",
        TranslateMessage(&Msg);
        DispatchMessage(&Msg);
    }
    return Msg.wParam;
}
```



GUI Programming Distilled

```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis(); break;
      case WM_THAT: DoThat(); break;
      case WM_OTHERTHING: DoOtherThing(); break;
      case WM_DONE: return;
      }
  }
}
```

GUI Programming Distilled

```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis(); break;
      case WM_THAT: DoThat(); break;
      case WM_OTHERTHING: DoOtherThing(); break;
      case WM_DONE: return;
      }
  }
}
```

Pros

GUI Programming Distilled

```
winmain(...) {
    while(true) {
        message = GetMessage();
        switch(message) {
        case WM_THIS: DoThis(); break;
        case WM_THAT: DoThat(); break;
        case WM_OTHERTHING: DoOtherThing(); break;
        case WM_DONE: return;
        }
    }
}
```

Pros

- Simple imperative programming

GUI Programming Distilled

```
winmain(...) {
    while(true) {
        message = GetMessage();
        switch(message) {
        case WM_THIS: DoThis(); break;
        case WM_THAT: DoThat(); break;
        case WM_OTHERTHING: DoOtherThing(); break;
        case WM_DONE: return;
        }
    }
}
```

Pros

- Simple imperative programming
- Good fit for uni-processor

GUI Programming Distilled

```
winmain(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis(); break;
    case WM_THAT: DoThat(); break;
    case WM_OTHERTHING: DoOtherThing(); break;
    case WM_DONE: return;
    }
  }
}
```

Pros

- Simple imperative programming
- Good fit for uni-processor

Cons

GUI Programming Distilled

```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis(); break;
      case WM_THAT: DoThat(); break;
      case WM_OTHERTHING: DoOtherThing(); break;
      case WM_DONE: return;
      }
  }
}
```

Pros

- Simple imperative programming
- Good fit for uni-processor

Cons

- Awkward/verbose

GUI Programming Distilled

```
winmain(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis(); break;
    case WM_THAT: DoThat(); break;
    case WM_OTHERTHING: DoOtherThing(); break;
    case WM_DONE: return;
    }
  }
}
```

Pros

- Simple imperative programming
- Good fit for uni-processor

Cons

- Awkward/verbose
- **Obscures available parallelism**

GUI Programming Distilled

```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_LONGRUNNING_CPU_HOG: HogCPU(); break;
      case WM_HIGH_LATENCY_IO: BlockForALongTime(); break;
      case WM_DO_QUICK_IMPORTANT_THING: HopeForTheBest(); break;
      }
  }
}
}
```

Pros

- Simple imperative programming
- Good fit for uni-processor

Cons

- Awkward/verbose
- **Obscures available parallelism**

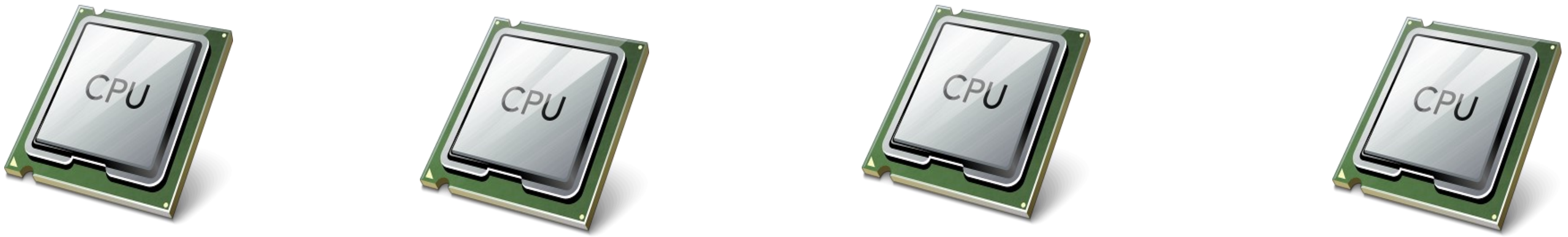
GUI Programming Distilled

```
winmain(...) {
    while(true) {
        message = GetMessage();
        switch(message) {
        case WM_THIS: DoThis(); break;
        case WM_THAT: DoThat(); break;
        case WM_OTHERTHING: DoOtherThing(); break;
        case WM_DONE: return;
        }
    }
}
```

GUI Programming Distilled

```
winmain(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis(); break;
    case WM_THAT: DoThat(); break;
    case WM_OTHERTHING: DoOtherThing(); break;
    case WM_DONE: return;
    }
  }
}
```

How can we parallelize this?



Parallel GUI Implementation 1

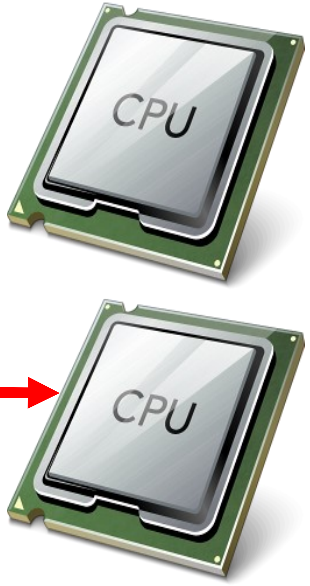
```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis(); break;
      case WM_THAT: DoThat(); break;
      case WM_OTHERTHING: DoOtherThing(); break;
      case WM_DONE: return;
      }
  }
}
```

Parallel GUI Implementation 1

```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis(); break;
      case WM_THAT: DoThat(); break;
      case WM_OTHERTHING: DoOtherThing(); break;
      case WM_DONE: return;
      }
  }
}
```

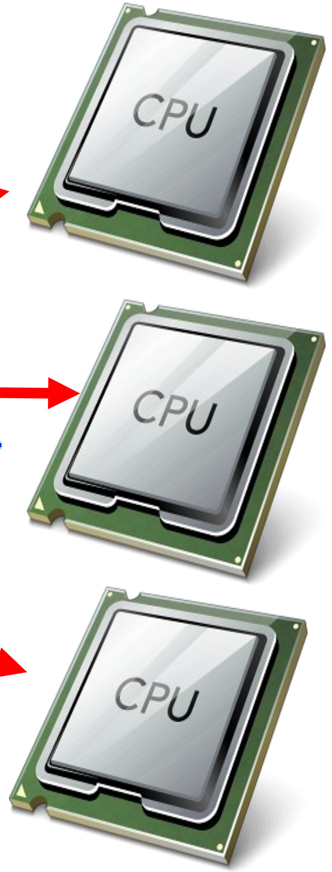
Parallel GUI Implementation 1

```
winmain(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis(); break;
      case WM_THAT: DoThat(); break;
      case WM_OTHERTHING: DoOtherThing(); break;
      case WM_DONE: return;
      }
  }
}
```



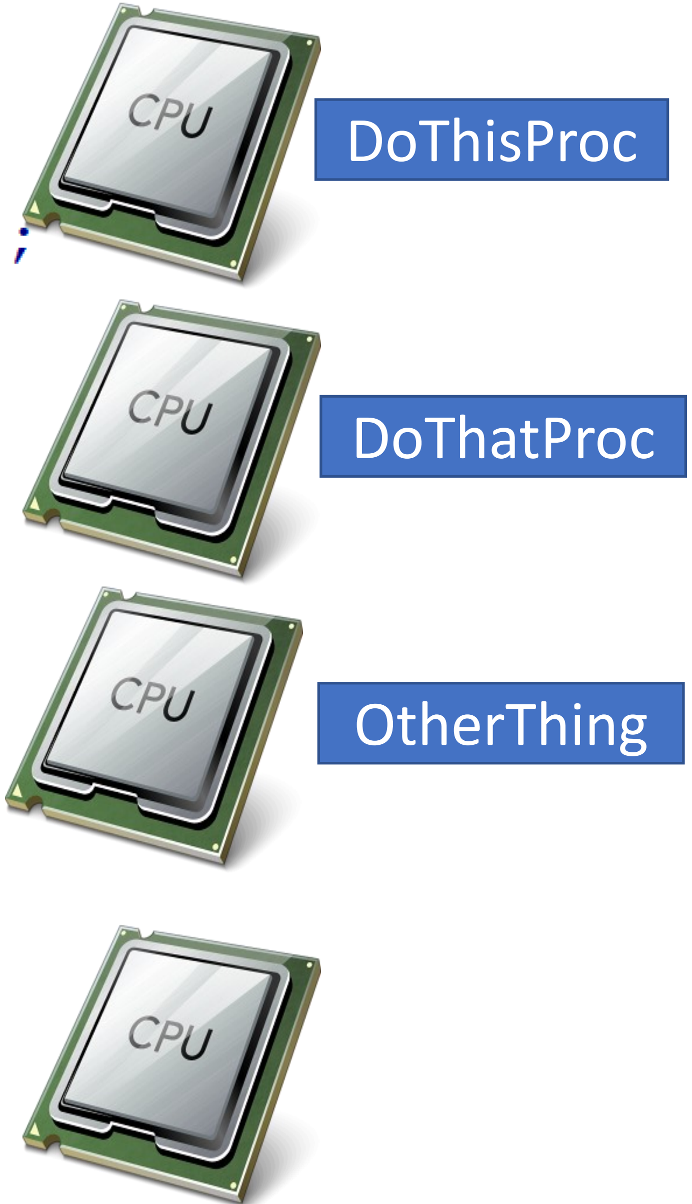
Parallel GUI Implementation 1

```
winmain(...) {
    while(true) {
        message = GetMessage();
        switch(message) {
        case WM_THIS: DoThis(); break;
        case WM_THAT: DoThat(); break;
        case WM_OTHERTHING: DoOtherThing(); break;
        case WM_DONE: return;
        }
    }
}
```



Parallel GUI Implementation 1

```
winmain() {
    pthread_create(&tids[i++], DoThisProc);
    pthread_create(&tids[i++], DoThatProc);
    pthread_create(&tids[i++], DoOtherThingProc);
    for(j=0; j<i; j++)
        pthread_join(&tids[j]);
}

DoThisProc() {
    while(true){
        if(ThisHasHappened)
            DoThis();
    }
}
```

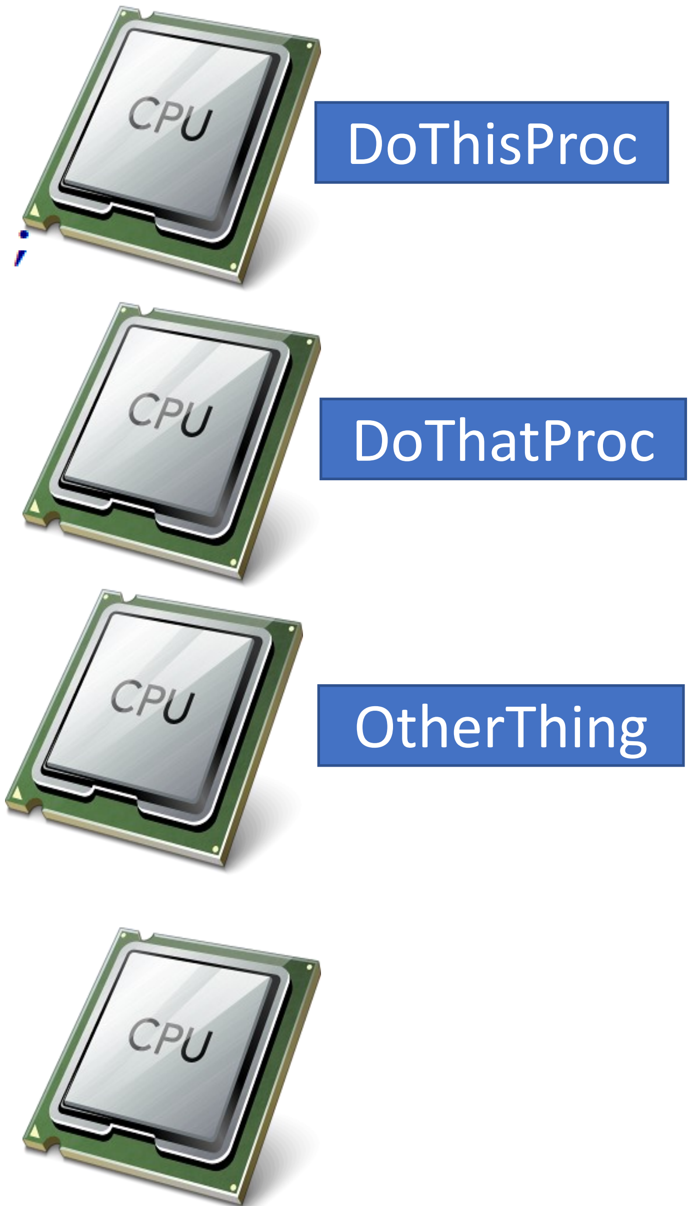


Parallel GUI Implementation 1

```
winmain() {
    pthread_create(&tids[i++], DoThisProc);
    pthread_create(&tids[i++], DoThatProc);
    pthread_create(&tids[i++], DoOtherThingProc);
    for(j=0; j<i; j++)
        pthread_join(&tids[j]);
}
```

Pros/cons?

```
DoThisProc() {
    while(true){
        if(ThisHasHappened)
            DoThis();
    }
}
```



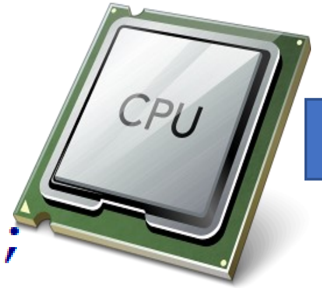
Parallel GUI Implementation 1

```
winmain() {
    pthread_create(&tids[i++], DoThisProc);
    pthread_create(&tids[i++], DoThatProc);
    pthread_create(&tids[i++], DoOtherThingProc);
    for(j=0; j<i; j++)
        pthread_join(&tids[j]);
}

DoThisProc() {
    while(true){
        if(ThisHasHap
            DoThis();
    }
}
```

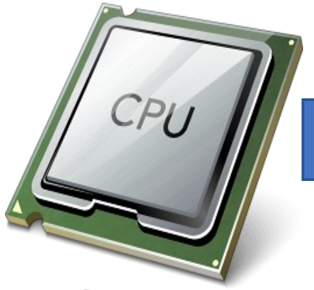
Pros/cons?

Pros:
- Encapsulates parallel work

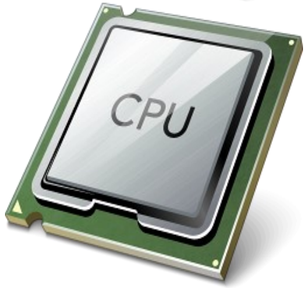
Cons:
- Obliterates original code structure
- How to assign handlers→CPUs?
- Load balance?!?
- Utilization



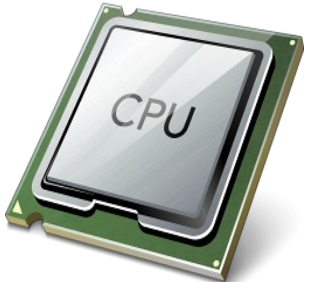
DoThisProc



DoThatProc



OtherThing



Parallel GUI Implementation 2

```
winmain() {
    for(i=0; i<NUMPROCS; i++)
        pthread_create(&tids[i], HandlerProc);
    for(i=0; i<NUMPROCS; i++)
        pthread_join(&tids[i]);
}
```

```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```


```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```

```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```

```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```


Parallel GUI Implementation 2

Pros/cons?

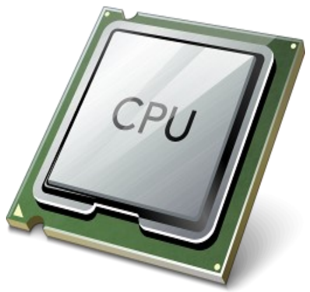
```
winmain() {
    for(i=0; i<NUMPROCS; i++)
        pthread_create(&tids[i], HandlerProc);
    for(i=0; i<NUMPROCS; i++)
        pthread_join(&tids[i]);
}
```

```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```

```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```

```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```

```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```



Parallel GUI Implementation 2

```
winmain() {
    for(i=0; i<NUMPROCS; i++)
        pthread_create(&tids[i], H
    for(i=0; i<NUMPROCS; i++)
        pthread_join(&tids[i]);
}
```

Pros/cons?

Pros:
- Preserves programming model
- Can recover some parallelism

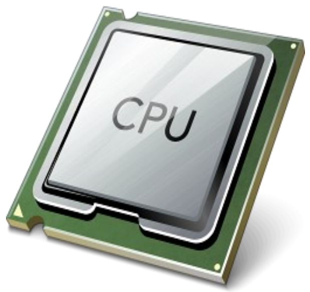
Cons:
- Workers still have same problem
- How to load balance?
- Shared mutable state a problem

```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```


```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```


```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```


```
threadproc(...) {
  while(true) {
      message = GetMessage();
      switch(message) {
      case WM_THIS: DoThis();
      case WM_THAT: DoThat();
      }
  }
}
```



Parallel GUI Implementation 2

```
winmain() {
    for(i=0; i<NUMPROCS; i++)
        pthread_create(&tids[i], H
    for(i=0; i<NUMPROCS; i++)
        pthread_join(&tids[i]);
}
```

Pros/cons?

Pros:
- Preserves programming model
- Can recover some parallelism

Cons:
- Workers still have same problem
- How to load balance?
- Shared mutable state a problem

```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```


```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```

```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```

```
threadproc(...) {
  while(true) {
    message = GetMessage();
    switch(message) {
    case WM_THIS: DoThis();
    case WM_THAT: DoThat();
    }
  }
}
```


Extremely difficult to solve without changing the whole programming model…so ***change it***

Questions?