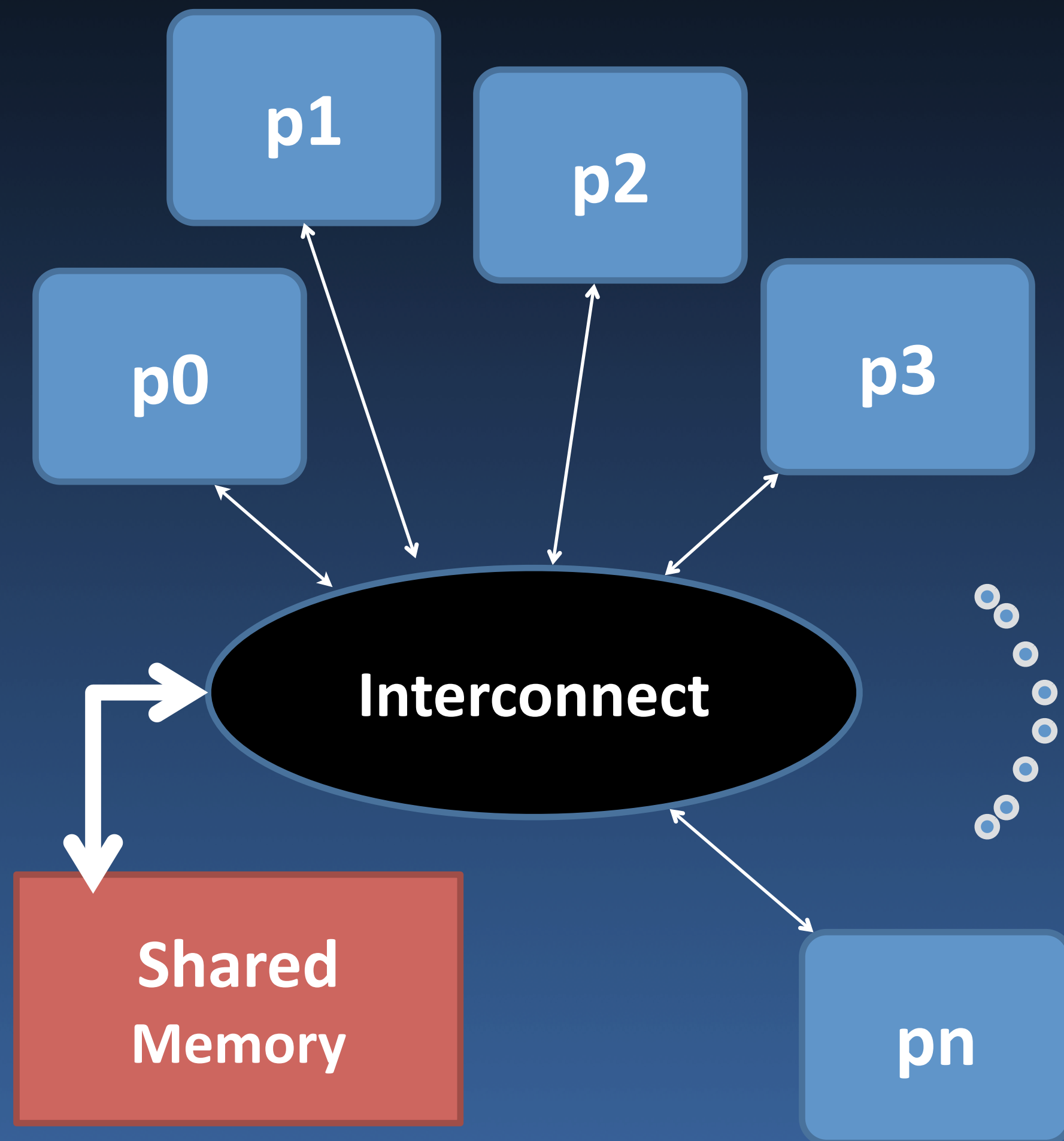


COL380

Introduction to  
Parallel & Distributed Programming

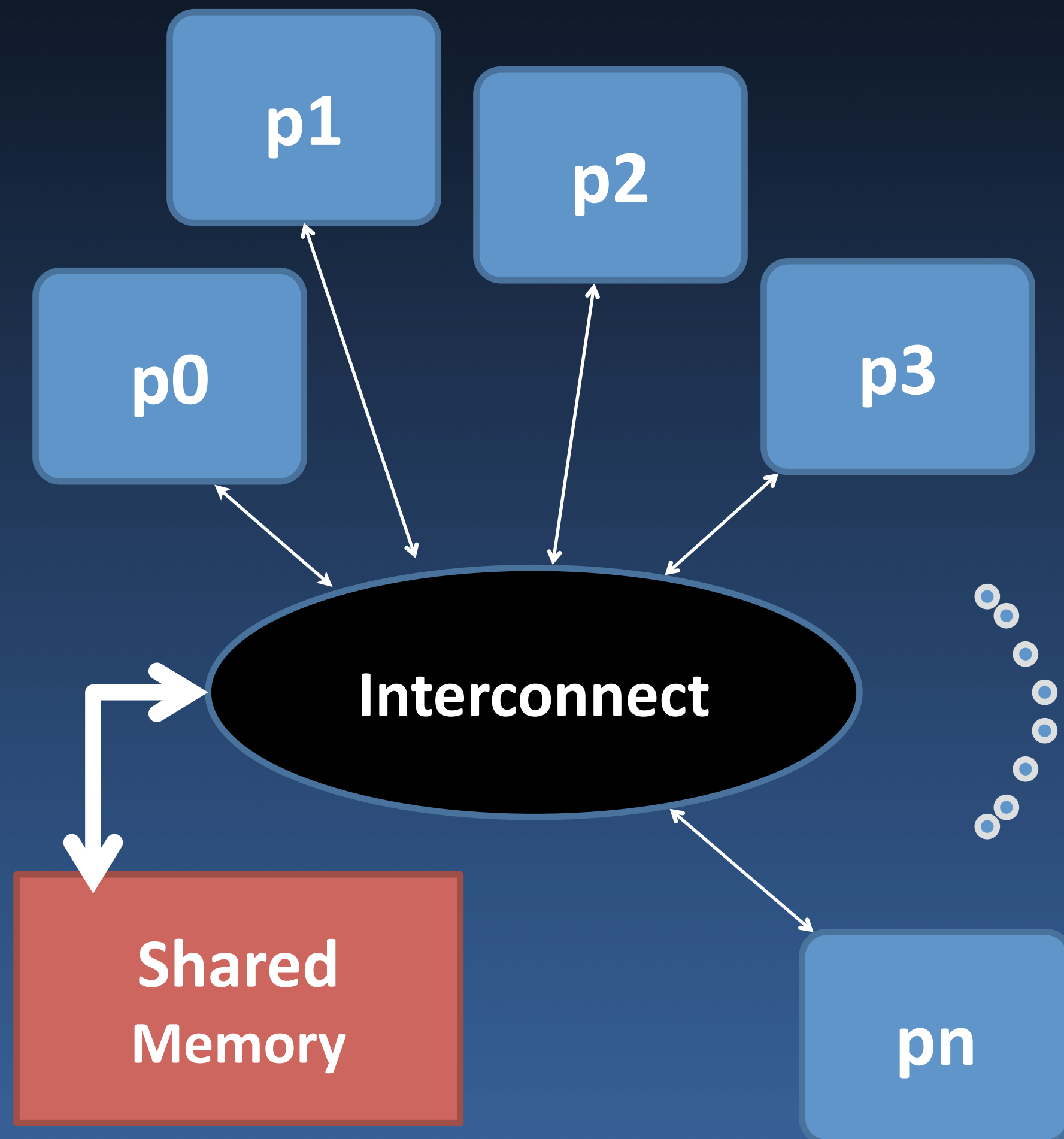
- Simplify specifying, reasoning, analyzing algorithms
- Must abstract away many details
  - ➔ Should predict computability
- Should track performance
- General classes
  - ➔ Shared Memory vs Distributed Memory
  - ➔ Synchronous vs Asynchronous

# PRAM Model



- Synchronous, Shared Mem
  - Arbitrary number of cells
- Arbitrary number of processors, each:
  - has local memory (Arbitrary number of cells)
  - knows its ID
  - can access a shared memory location in *constant* time

# PRAM Model

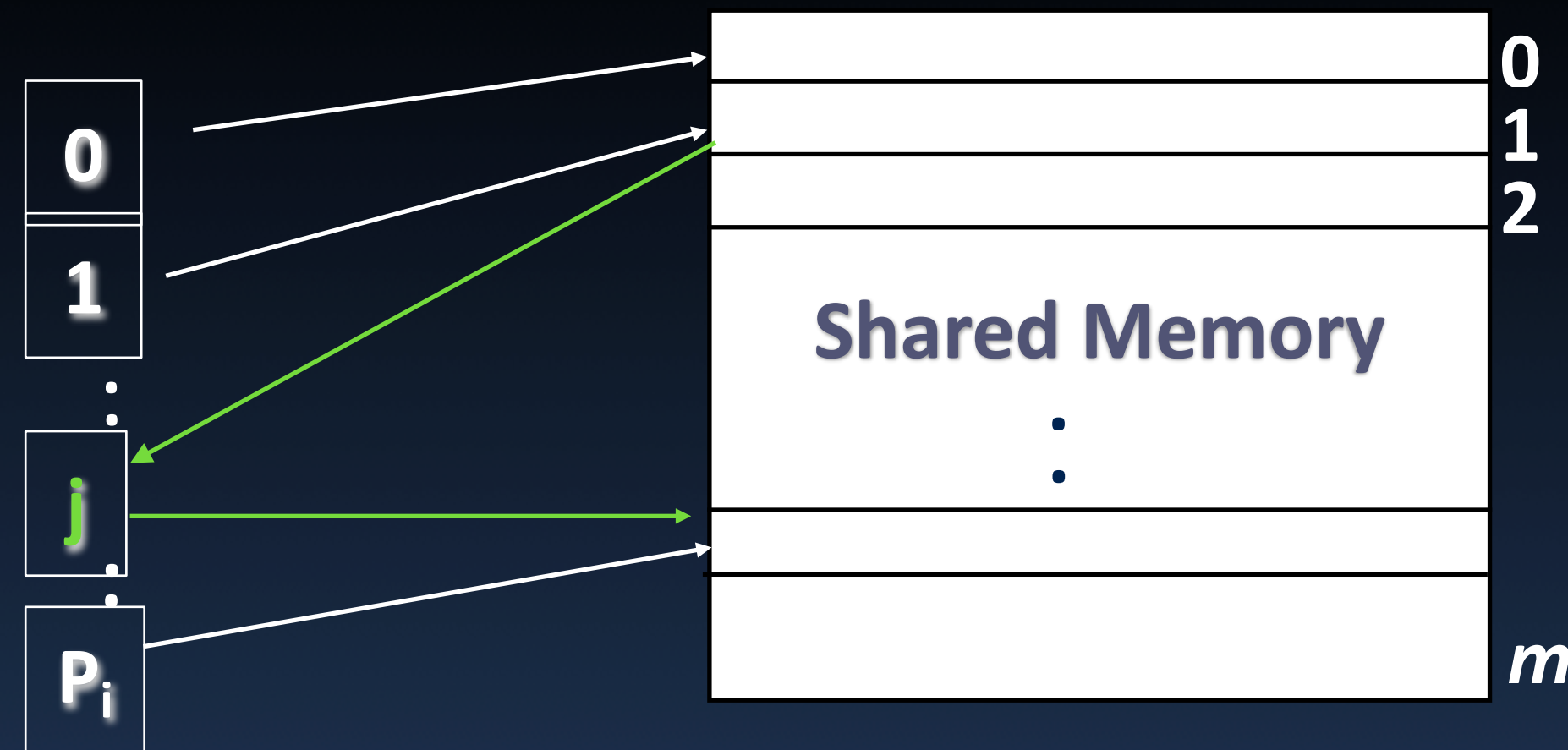


- Synchronous, Shared Mem
  - Arbitrary number of cells
- Arbitrary number of processors, each:
  - has local memory (Arbitrary number of cells)
  - knows its ID
  - can access a shared memory location in *constant* time

Unrealistic?

but can be simulated  
and provides useful asymptotic behavior


## PRAM Model Steps



- At each time-step  $j$ ,  $\forall i < P_j$ :
    1. read from memory cell  $m_{1ji}$
    2. perform a local computation step
    3. write to memory cell  $m_{2ji}$  (Read and write are in two phases)
      - Co-access may be restricted
- Takes  $O(1)$  time
- Thus, processors  $i_1$  and  $i_2$  can communicate in two steps =  $O(1)$
  - Synchronization is implicit and 'race' is restricted

- Each processor has unlimited local registers
- Shared memory has unique addresses
  - Address of Inputs/Outputs are known to the program (global to processors)
  - A step “ $\forall i < P_j: \dots$ ,” may invoke an arbitrary number of processors
    - ▶ Strictly, active processors at step  $j$  may activate the required ones at  $j+1$ 
      - With one processor active at the beginning of the program
- Processors are synchronous (lock step)
- Cost, Work, Time (taken by the longest running processor)
  - Also:  $\max(P_j)$  and number of unique shared memory memory addresses

(Asynchronous PRAM variants exist as well)

- EREW (Exclusive Read Exclusive Write)
  - ➔ Only one processors may read or write any given location in a step
- CREW (Concurrent Read Exclusive Write)
  - ➔ Many processors can simultaneously read a location, but only one may write
- CRCW (Concurrent Read Concurrent Write) 
  - ➔ Many processors can read/write the same memory location
- ~~ERCW (Exclusive Read Concurrent Write)~~
  - ➔ Not commonly used

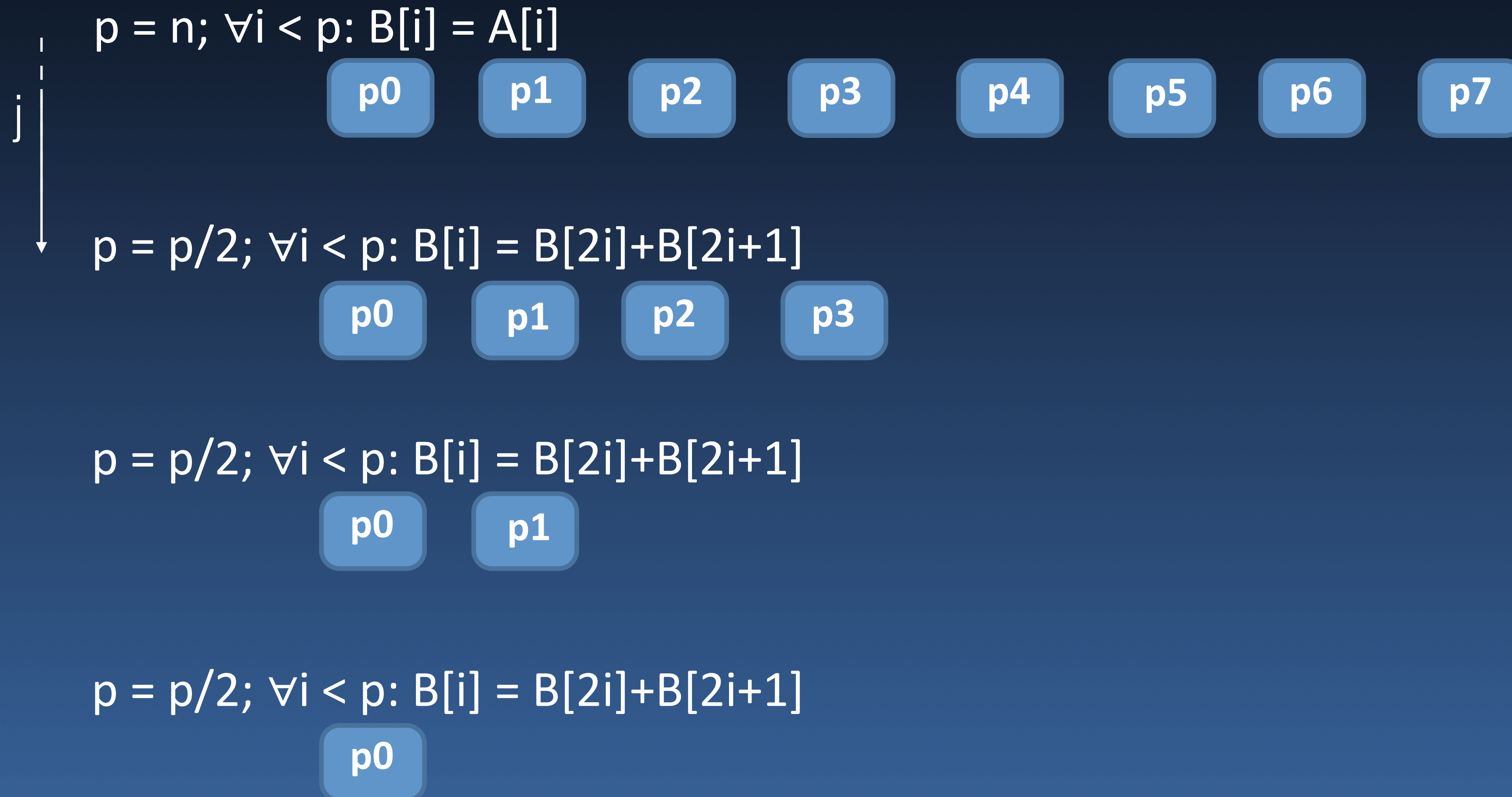
# Concurrent Write Models (CW)

- **Priority CW**
  - Higher priority processor (normally lower index) wins
- **Common CW**
  - Succeeds only if all writes have the same value
- **Arbitrary/Random CW**
  - One of the values is randomly chosen

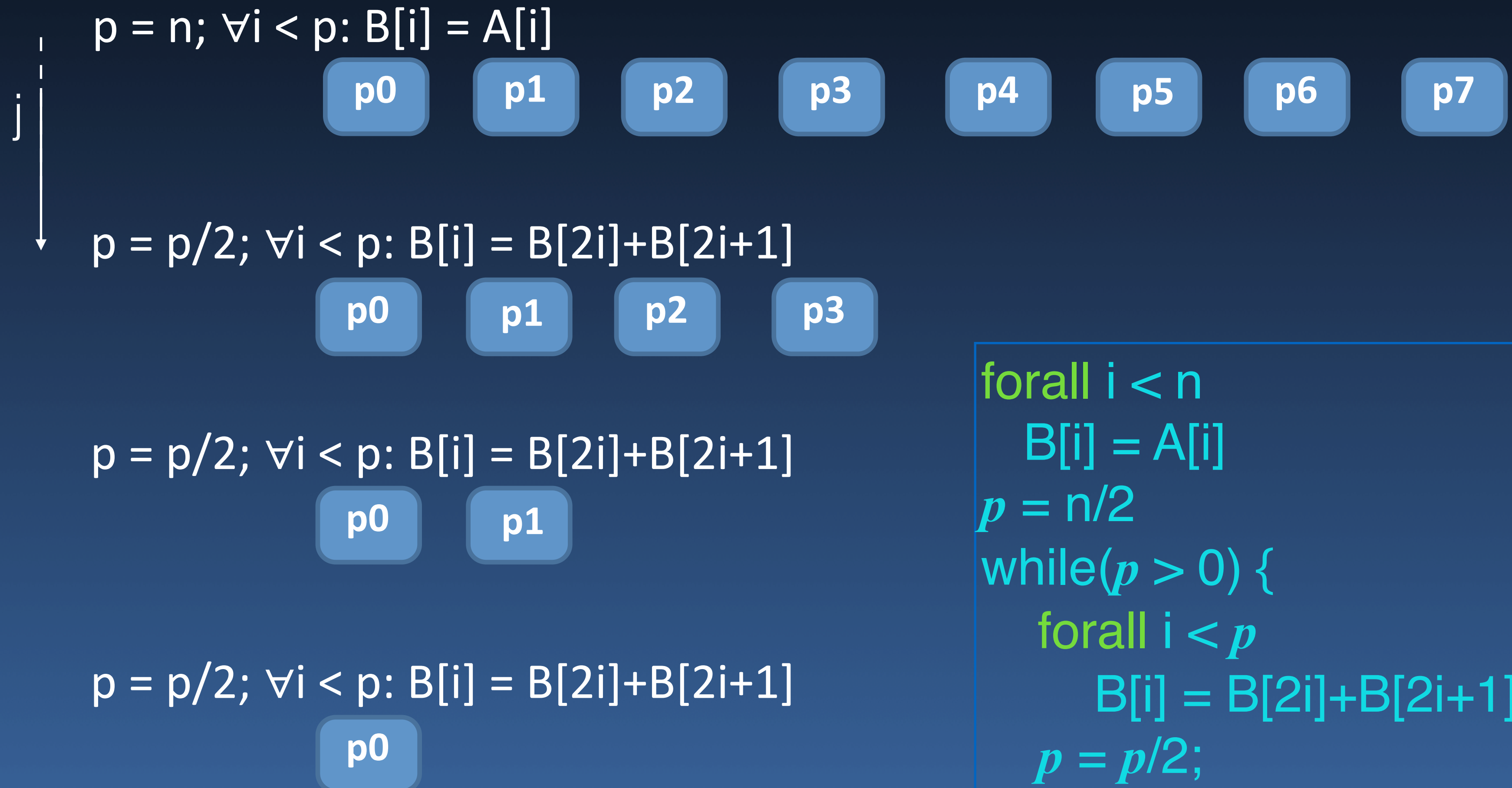
**EREW  $\leq$  CREW  $\leq$  Common CRCW  $\leq$  Arbitrary CRCW  $\leq$  Priority CRCW**

→ Increasing amount of computation per step →

# Parallel Addition



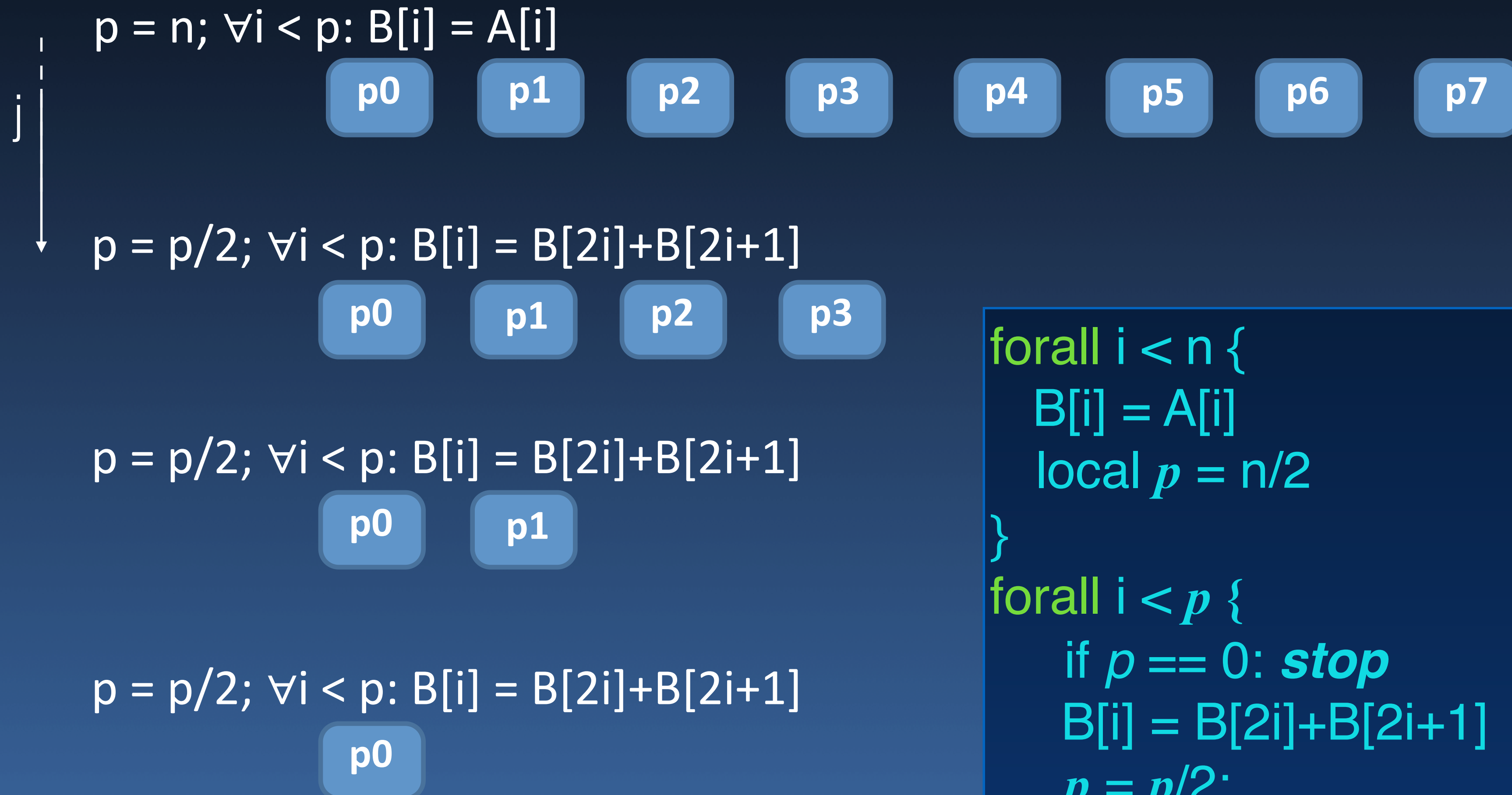
# Parallel Addition



```
forall i < n  
  B[i] = A[i]  
p = n/2  
while(p > 0) {  
  forall i < p  
    B[i] = B[2i] + B[2i+1]  
  p = p/2;  
}
```

(assumes n is a power of 2)

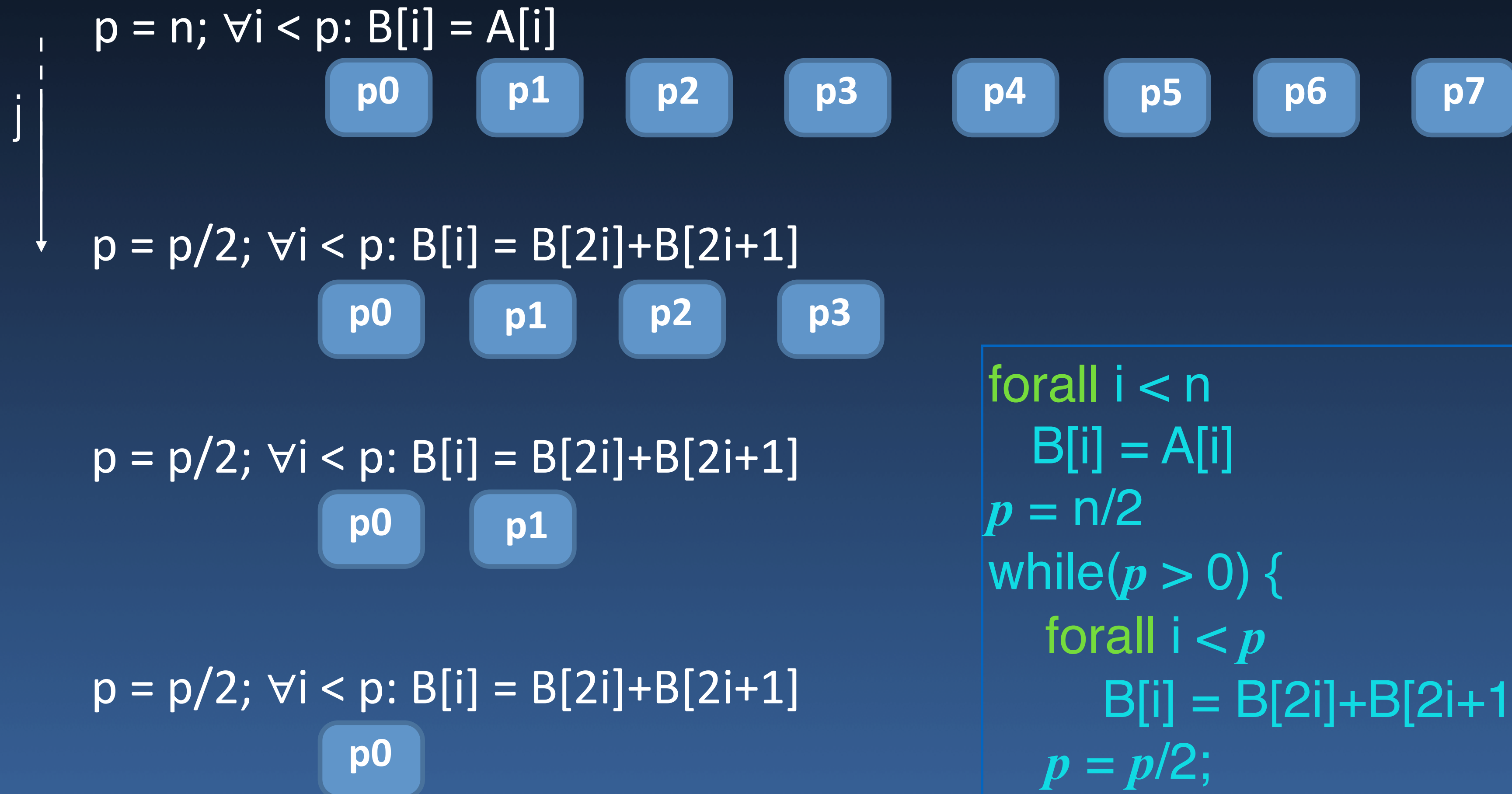
# Parallel Addition



```
forall i < n {  
  B[i] = A[i]  
  local p = n/2  
}  
forall i < p {  
  if p == 0: stop  
  B[i] = B[2i] + B[2i+1]  
  p = p/2;  
}: repeat
```

(assumes n is a power of 2)

# Parallel Addition



```
forall i < n
  B[i] = A[i]
p = n/2
while(p > 0) {
  forall i < p
    B[i] = B[2i] + B[2i+1]
  p = p/2;
}
```

(assumes n is a power of 2)

- processors: n
- time:  $O(\log n)$  CREW
- Speed-up:  $n/(\log n)$
- Efficiency:  $1/\log(n)$
- Cost:  $n \log n$
- Work: n

# Linear Search

$p < n$

- $n$  input integers in  $n$  memory cells
- Does  $x$  exist in the input?
  - $x$  is initially stored in shared memory

## Algorithm

step1: If  $p=0$ , broadcast  $x$

step2: Processor  $p_i$ : search in  $i$ th  $[n/p]$  block and {set flag  $f_i$ }

step3: If  $p=0$ , check if 'any' flag is 1 and print answer

EREW

•  $\log(p)$

•  $n/p$

•  $\log(p)$

CREW

• 1

•  $n/p$

•  $\log(p)$

CRCW

• 1

•  $n/p$

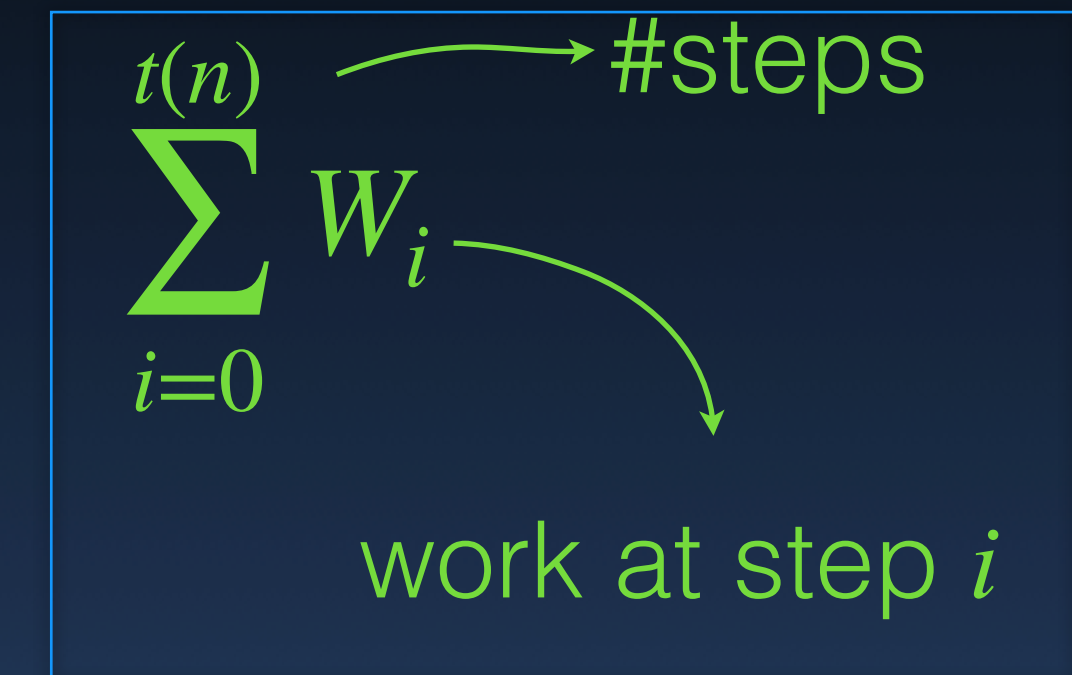
• 1

## PRAM Generality

- ➔ Any problem that can be solved on a  $p$ -processor PRAM in  $t$  steps can be solved on a  $p'$ -processor PRAM in  $t' = O(t \cdot p / p')$  steps (assuming the same size of shared memory).
- ➔ Any problem that can be solved on a  $p$ -processor and  $m$ -cell PRAM in  $t$  steps can be solved on a  $\max(p, m')$ -processor and  $m'$ -cell PRAM in  $O(tm / m')$  steps.
- ➔ Any problem that can be solved on a  $p$ -processor and  $m$ -cell CREW PRAM in  $t$  steps can be solved on a  $p$ -processor and  $m$ -cell EREW PRAM in  $O(tp)$  steps.
- ➔ ...

## Work Time Scheduling Principle

- Design algorithm in terms of
  - Total work done per 'time step':  $W_i(n)$
  - $t(n)$  steps
- Total work done  $W(n) = \sum W_i(n)$
- Given  $p$  processors, at step  $i$ :
  - divide the work  $W_i(n)$  among  $p$  processors
    - ▶  $t(n,p) \leq \sum \lceil W_i(n)/p \rceil = O(W(n)/p + t(n))$
- Cost =  $t(n,p) * p$



- **Work = Cost if:**
  - $p * t(n,p) = O(W(n))$
  - Or,  $p = O(W(n)/t(n,p))$

**Work  $\leq$  Cost:**  
Cost optimality is more stringent.

- If **sequentially** optimal algorithm is  $O(t'(n))$

→ Work done by **Work-optimal** parallel algorithm:

▶  $O(t'(n))$  ( with time  $t(n) = O(t'(n))$  ).

→ Work-scheduling on  $p$  processors takes time:

▶  $t(n,p) = O(t'(n)/p + t(n))$

→ Optimal speed-up:  $t'(n)/t(n,p) = \theta(p)$ , if

▶  $[p \cdot t'(n)] / [t'(n) + p \cdot t(n)] = \theta(p)$

- **Work-time** optimal if:

→  $t(n)$  cannot be improved (and work-optimal)

- Two naive parameters:

→  $p(n)$ ,  $t(n)$

- Generally, use work:  $W(n)$

- If  $W(n)$  is similar, use  $t(n)$

- Speedup/Scalability

→ Absolute: over best sequential algorithm

→ Relative: over the 1-processor implementation of the same algorithm

→  $p(n)$  is hidden in  $W(n)$

▶  $W_a(n) = O(n)$ ;  $t_a(n) = O(n)$

▶  $W_b(n) = O(n \log n)$  and  $t_b(n) = O(\log n)$

## Why PRAM?

- **Easy to design, specify, analyze algorithms**
  - Independent of machine details
- **Fidelity of predicted performance**
  - Not many surprises for shared-memory architecture
  - Only partially successful for distributed memory
  - Note that memory-access and message latency is often bounded
- **Strong model**
  - Possible to simulate on a wide variety of hardware
  - Poor PRAM solution often implies a hard problem

## Bulk Synchronous Parallel Model

- A set of virtual (processor, memory) pairs
  - ➔ No notion of “locality” in mapping to physical processor
- A point-to-point interconnect
- Barrier synchronization (all or subset)
- Repeat “super-step”:
  - ➔ Local computation
  - ➔ Communication
  - ➔ Barrier synchronization

Designed as a Bridging  
model for parallel  
programming

