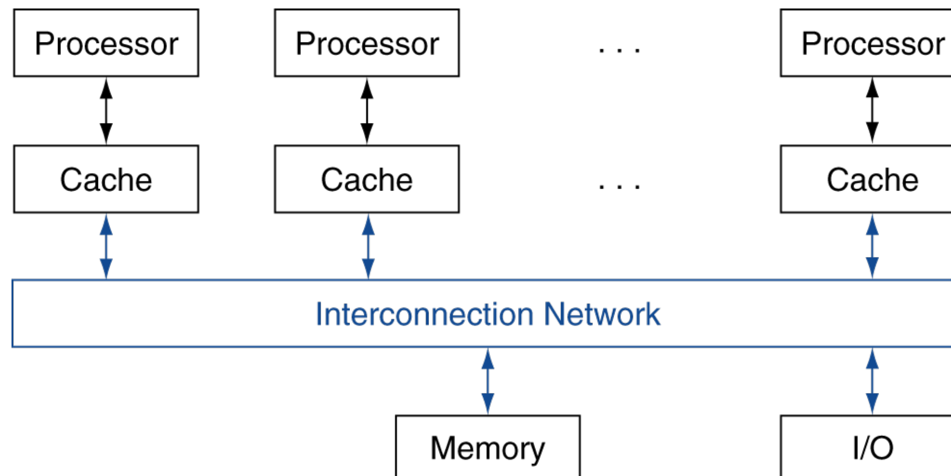
Computer Architecture

Multiprocessors

Shared Memory

- Shared memory multiprocessor
  - Hardware provides single physical address space for all processors



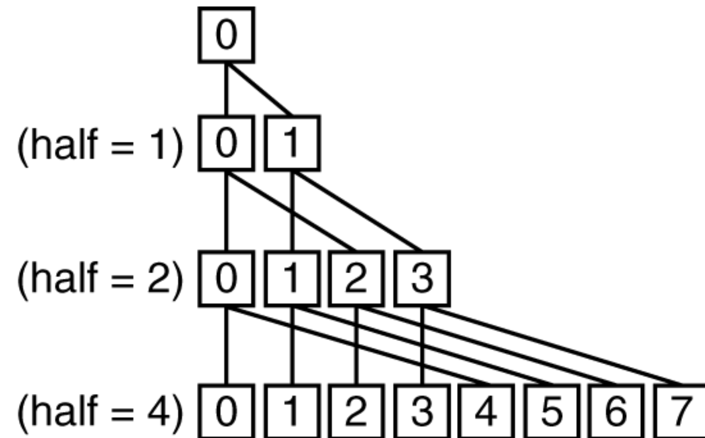
Example: Sum Reduction

- Sum 100,000 numbers on 100 processor UMA
  - Each processor has ID: $0 \le Pn \le 99$
  - Partition 1000 numbers per processor
  - Initial summation on each processor

```
sum[Pn] = 0;
  for (i = 1000*Pn;
      i < 1000*(Pn+1); i = i + 1)
    sum[Pn] = sum[Pn] + A[i];
```

- Now need to add these partial sums
  - Reduction: divide and conquer
  - Half the processors add pairs, then quarter, …
  - Need to synchronize between reduction steps

Example: Sum Reduction




```
half = 100;
repeat
  synch();
  if (half%2 != 0 && Pn == 0)
    sum[0] = sum[0] + sum[half-1];
    /* Conditional sum needed when half is odd;
       Processor0 gets missing element */
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1);
```

Synchronization in Shared Memory

- **Shared data**

type *item* = …;
var *buffer*. **Array** [0..n-1] **of** *item*;
in, out: 0..*n*-1;
counter. 0..*n*;
in, out, counter :=0;

- **Producer process**

repeat
    …
  produce an item in *nextp*
    …
  while *counter* = *n* **do** *no-op;*
  buffer [*in*] := *nextp*;
  in := *in* + 1 **mod** *n*;
  counter := *counter* + 1;
until *false*;

Bounded-Buffer Example

- **Consumer process**

```
repeat
    while counter = 0 do no-op;
    nextc := buffer [out];
    out := out + 1 mod n;
    counter := counter - 1;
       ...
    consume the item in nextc
       ...
until false;
```

More Detailed Picture

counter := counter + 1

register-a := counter;

register-a := register-a + 1;

counter := register-a;

counter := counter - 1

register-b := counter;

register-b := register-b -1;

counter := register-b;

Problem

```
producer  execute  register-a := counter
producer  execute  register-a := register-a + 1
consumer  execute  register-b := counter
consumer  execute  register-b := register-b -1
producer  execute  counter := register-a
consumer  execute  counter := register-b
```

- Assuming counter is initially 5, what will be the final value of counter?

The Critical-Section Problem

- n processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.
- Structure of process P_i

```
repeat
    entry section
        critical section
    exit section
        remainder section
until false;
```

Correctness Criteria for a Solution to the Critical-Section Problem

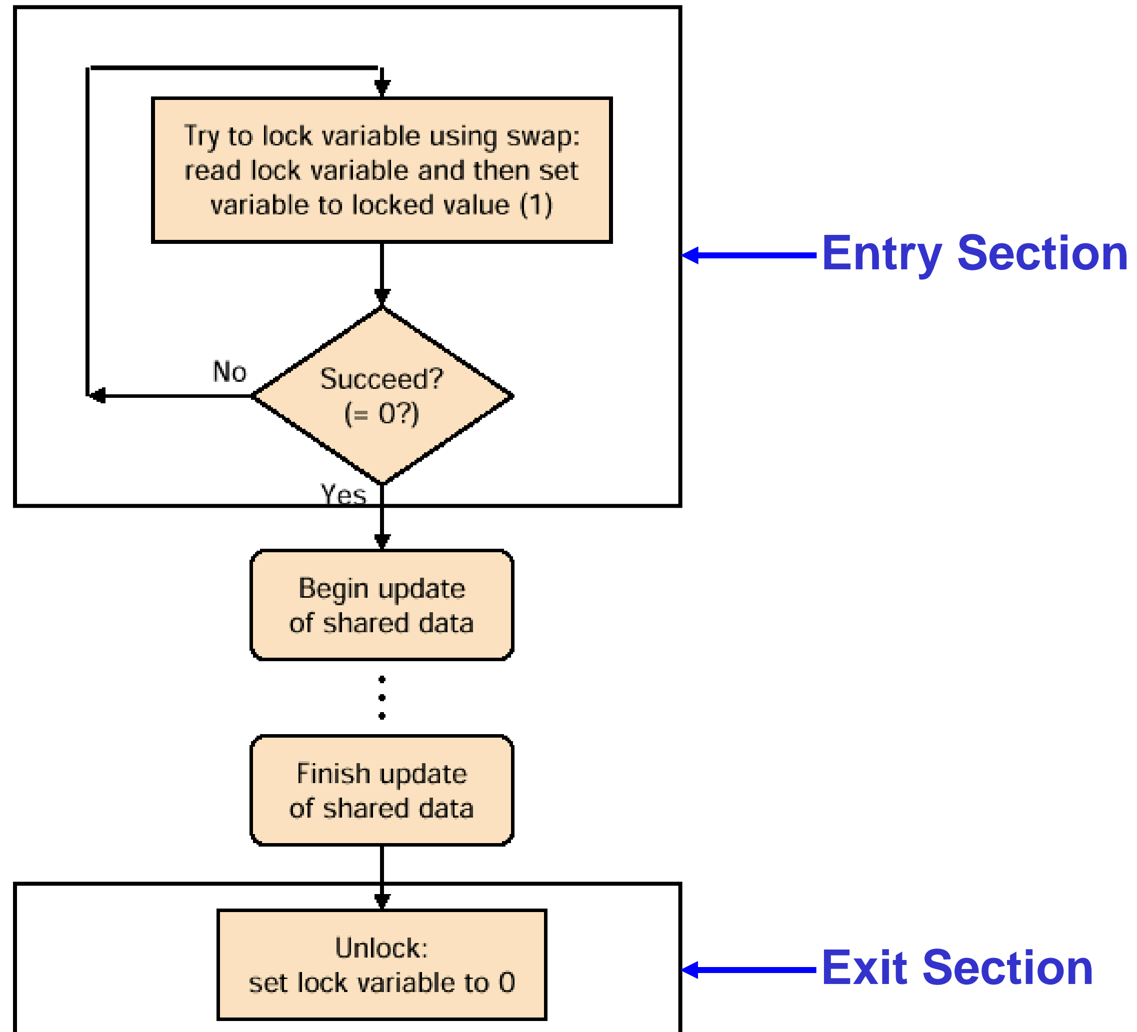
- **Mutual Exclusion.** If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
- **Progress.** If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
- **Bounded Waiting.** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Mutual Exclusion with Test-and-Set

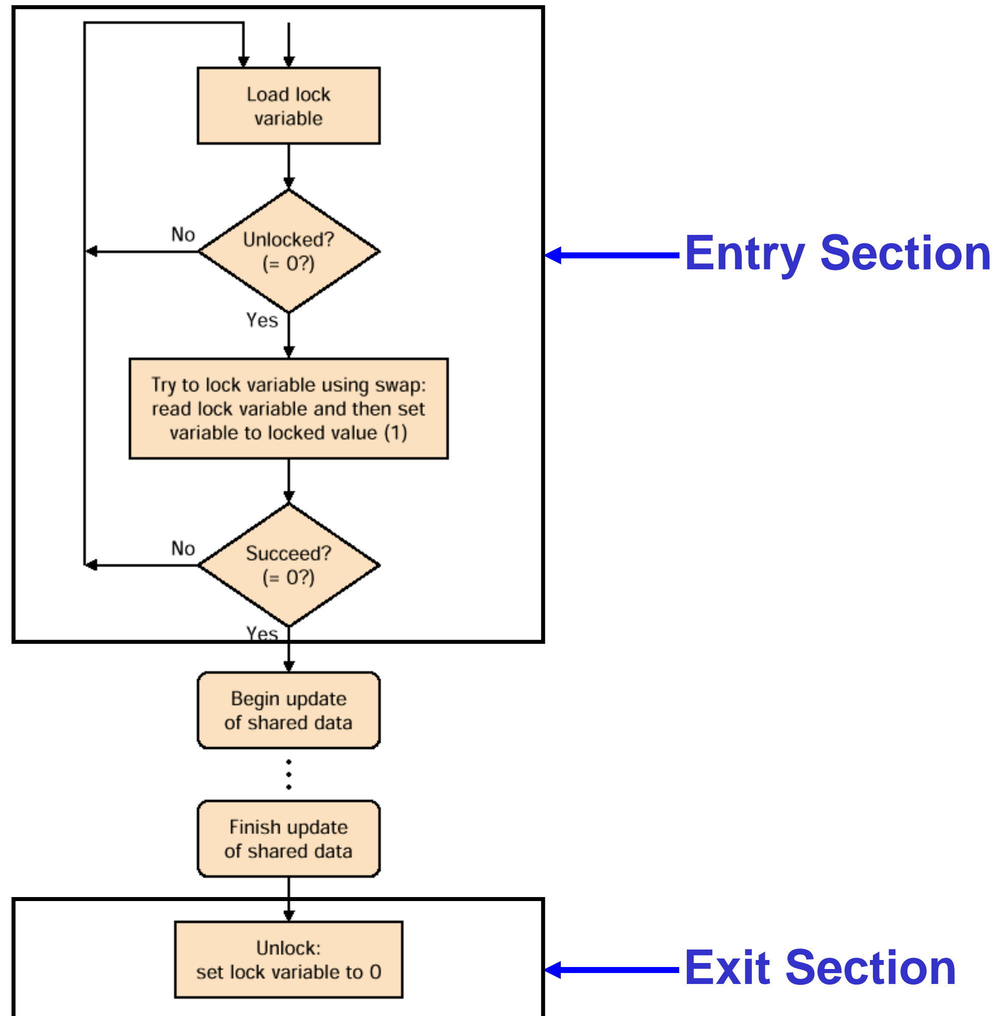
- Shared data: **var** lock: boolean (initially false)
- Process P_i

```
repeat
    while Test-and-Set (lock) do no-op;      <- Entry Section
            critical section
    lock := false;                           <- Exit Section
            remainder section
until false;
```

Naive Synchronization

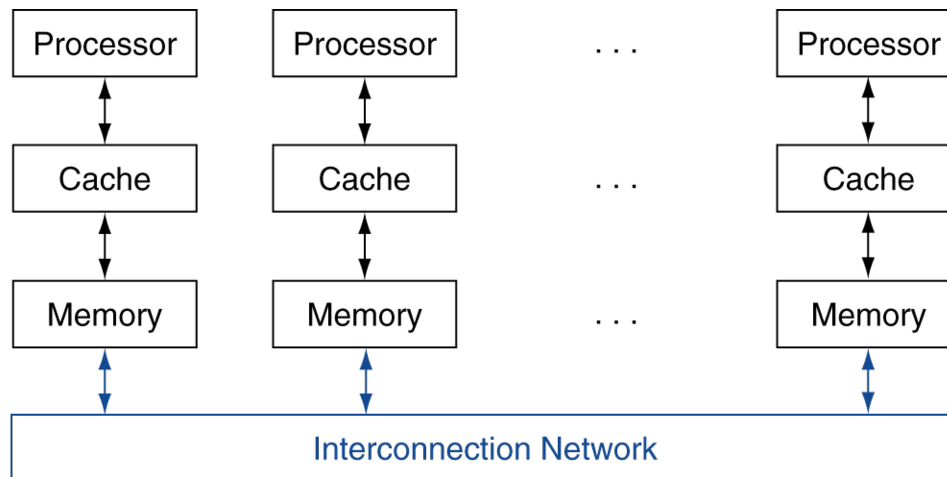


Optimized Synchronization



Message Passing

- Each processor has private physical address space
- Hardware sends/receives messages between processors



Loosely Coupled Clusters

- Network of independent computers
  - Each has private memory and OS
  - Connected using I/O system
    - E.g., Ethernet/switch, Internet
- Suitable for applications with independent tasks
  - Web servers, databases, simulations, …
- High availability, scalable, affordable
- Problems
  - Administration cost (prefer virtual machines)
  - Low interconnect bandwidth
    - c.f. processor/memory bandwidth on an SMP

Sum Reduction (Again)

- Sum 100,000 on 100 processors
- First distribute 1000 numbers to each
  - The do partial sums

```
sum = 0;
for (i = 0; i<1000; i = i + 1)
   sum = sum + AN[i];
```

- Reduction
  - Half the processors send, other half receive and add
  - The quarter send, quarter receive and add, …

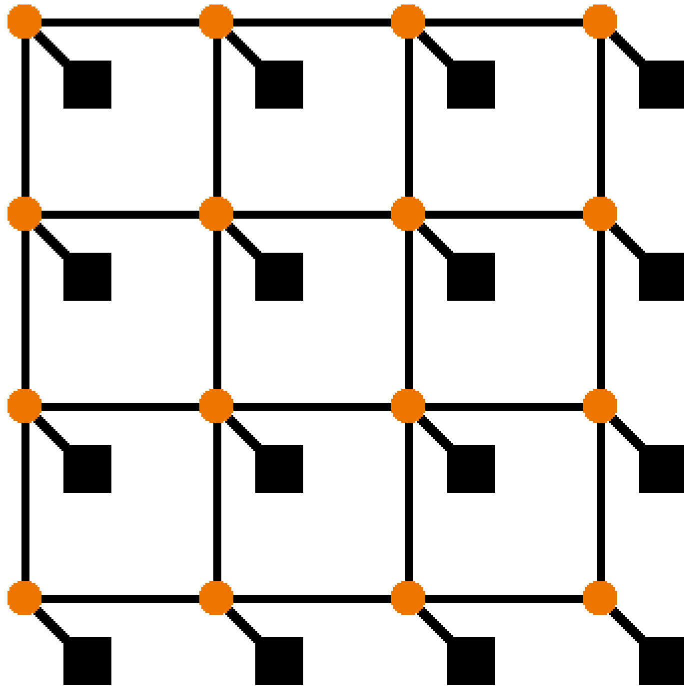
Sum Reduction (Again)

- Given send() and receive() operations

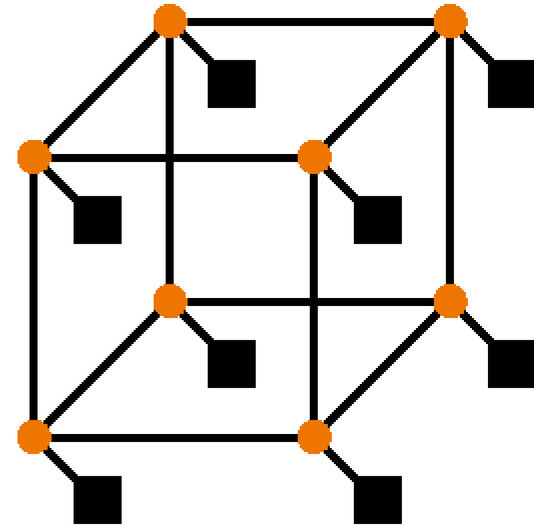
```
limit = 100; half = 100;/* 100 processors */
repeat
  half = (half+1)/2; /* send vs. receive
                        dividing line */
  if (Pn >= half && Pn < limit)
    send(Pn - half, sum);
  if (Pn < (limit/2))
    sum = sum + receive();
  limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```

- Send/receive also provide synchronization
- Assumes send/receive take similar time to addition

Network Topology

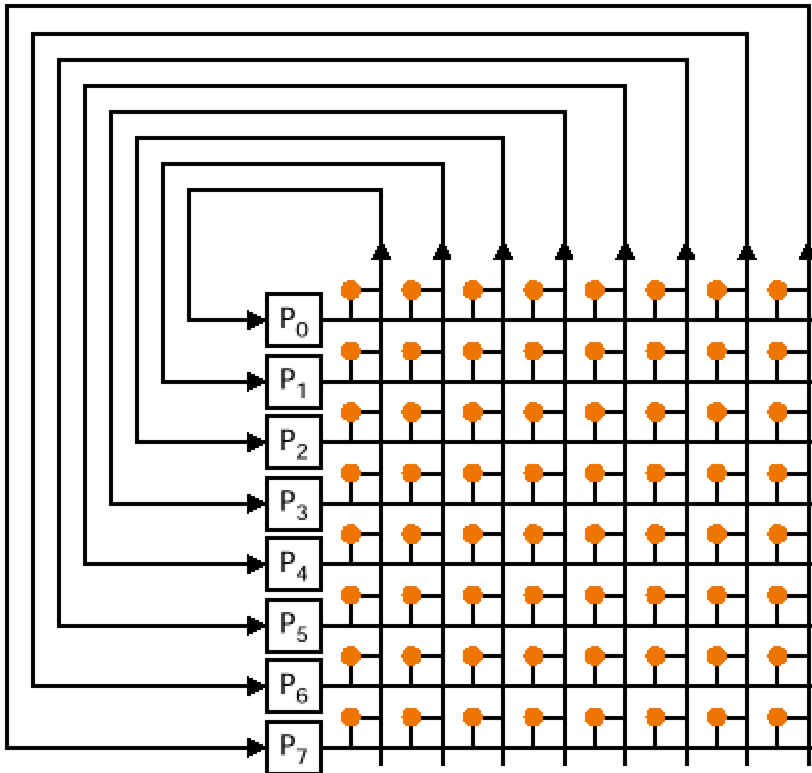


2-D grid or mesh

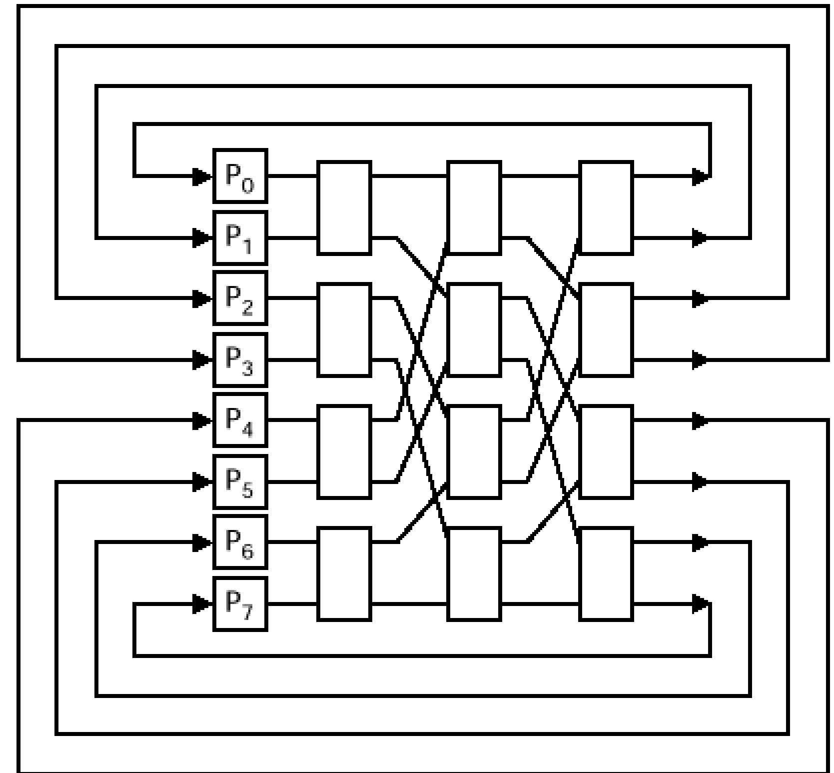


n-cube

Network Topology



Crossbar



Omega network

The Evolution-Revolution Spectrum of Computer Architecture