Physical Organization of Parallel Platforms
The PRAM Model

Alexandre David
B2-206

Today

- Introduction to Parallel Algorithms (Sven Skyum)
  - PRAM model
  - Optimality
  - Examples
- Physical Organization of Parallel Platforms (2.4)

Standard RAM Model

- Standard **R**andom **A**ccess **M**achine:
  - Each operation
    load, store, jump, add, etc …
  - takes one unit of time.
- Simple, generally one model.

Multi-processor Machines

- Numerous architectures
  $\rightarrow$ different models.
- Difference in communication
  - Synchronous
  - Asynchronous
- Difference in memory layout
  - NUMA
  - UMA

PRAM Model

- A PRAM consists of
  - a *global* access *memory* (i.e. shared)
  - a set of *processors* running the same program (though not always), with a private *stack*.
- A PRAM is **synchronous.**
- Unlimited resources.

Classes of PRAM

- How to resolve *contention*?
  - EREW PRAM – exclusive read, exclusive write
  - CREW PRAM – concurrent read, exclusive write
  - ERCW PRAM – exclusive read, concurrent write
  - CRCW PRAM – concurrent read, concurrent write

Example: Sequential Max

```
Function smax(A,n)
        m := -∞
        for i := 1 to n do
                m := max{m,A[i]}
        od
        smax := m
end
```

Time $O(n)$

Example: Sequential Max (bis)

Time $O(n)$

```
Function smax2(A,n)
        for i := 1 to n/2 do
                B[i] := max{A[2i-1],A[2i]}
        od
        if n = 2 then
                smax2 := B[1]
        else
                smax2 := smax2(B,n/2)
        fi
end
```

Example: Parallel Max

Time $O(\log n)$

```
Function smax2(A,n) [p1,p2,…,pn/2]
    for i := 1 to n/2 pardo
        pi: B[i] := max{A[2i-1],A[2i]}
    od
    if n = 2 then
        p1: smax2 := B[1]
    else
        smax2 := smax2(B,n/2) [p1,p2,…,pn/4]
    fi
end
```

Analysis of the Parallel Max

- Time: $O(\log n)$ for $n/2$ processors.
- *Work done?*
  - $p(n)=n/2$ number of processors.
  - $t(n)$ time to run the algorithm.
  - $w(n)=p(n)*t(n)$ work done.
    Here $w(n)=O(n \log n)$.

Optimality

Definition

If *w(n)* is of the **same order** as the time for the best known sequential algorithm, then the parallel algorithm is said to be **optimal**.

Design Principle

Construct optimal algorithms to run **as fast as possible.**

=

Construct optimal algorithms using **as many processors as possible**!

Brent's Scheduling Principle

Theorem

If a parallel computation consists of
***k* phases**
taking time $t_1, t_2, \ldots, t_k$
using $a_1, a_2, \ldots, a_k$ processors
in phases $1, 2, \ldots, k$
then the computation can be done in time ***O(a/p+t)*** using ***p* processors** where $t = \mathrm{sum}(t_i)$, $a = \mathrm{sum}(a_i t_i)$.

Previous Example

- k phases = $\log n$.
- t_i = constant time.
- $a_i = n/2, n/4, \ldots, 1$ processors.
- With p processors we can use time $O(\log n + n/p)$.
- Choose $p = O(n/\log n) \rightarrow$ time $O(\log n)$ and this is **optimal**!

Prefix Computations

Input: array A[1..n] of numbers.
Output: array B[1..n] such that B[k] = sum(i:1..k) A[i]
Sequential algorithm:

```
function prefix+(A,n)
    B[1] := A[1]
    for i = 2 to n do
        B[i] := B[i-1]+A[i]
    od
end
```

Time $O(n)$

Parallel Prefix Computation

```
function prefix+(A,n)[p1,…,pn]
        p1: B[1] := A[1]
        if n > 1 then
                for i = 1 to n/2 pardo
                        pi: C[i]:=A[2i-1]+A[2i]
                od
                D:=prefix+(C,n/2)[p1,…,pn/2]
                for i = 1 to n/2 pardo
                        pi: B[2i]:=D[i]
                od
                for i = 2 to n/2 pardo
                        pi: B[2i-1]:=D[i-1]+A[2i-1]
                od
        fi
        prefix+:=B
end
```

Prefix Computations

- The point of this algorithm:
  - It works because + is associative (i.e. the compression works).
  - It will work for *any* other associative operations.
  - Brent's scheduling principle:

For any associative operator computable in *O(1)*, its prefix is computable in *O(logn)* using *O(n/logn)* processors, which is optimal!

Merging (of Sorted Arrays)

- Rank function:
  - rank(x,A,n) = 0 if x < A[1]
  - rank(x,A,n) = max{i | A[i] ≤ x}
  - Computable in time $O(\log n)$ by binary search.
- Merge A[1..n] and B[1..m] into C[1..n+m].
- Sequential algorithm in time $O(n+m)$.

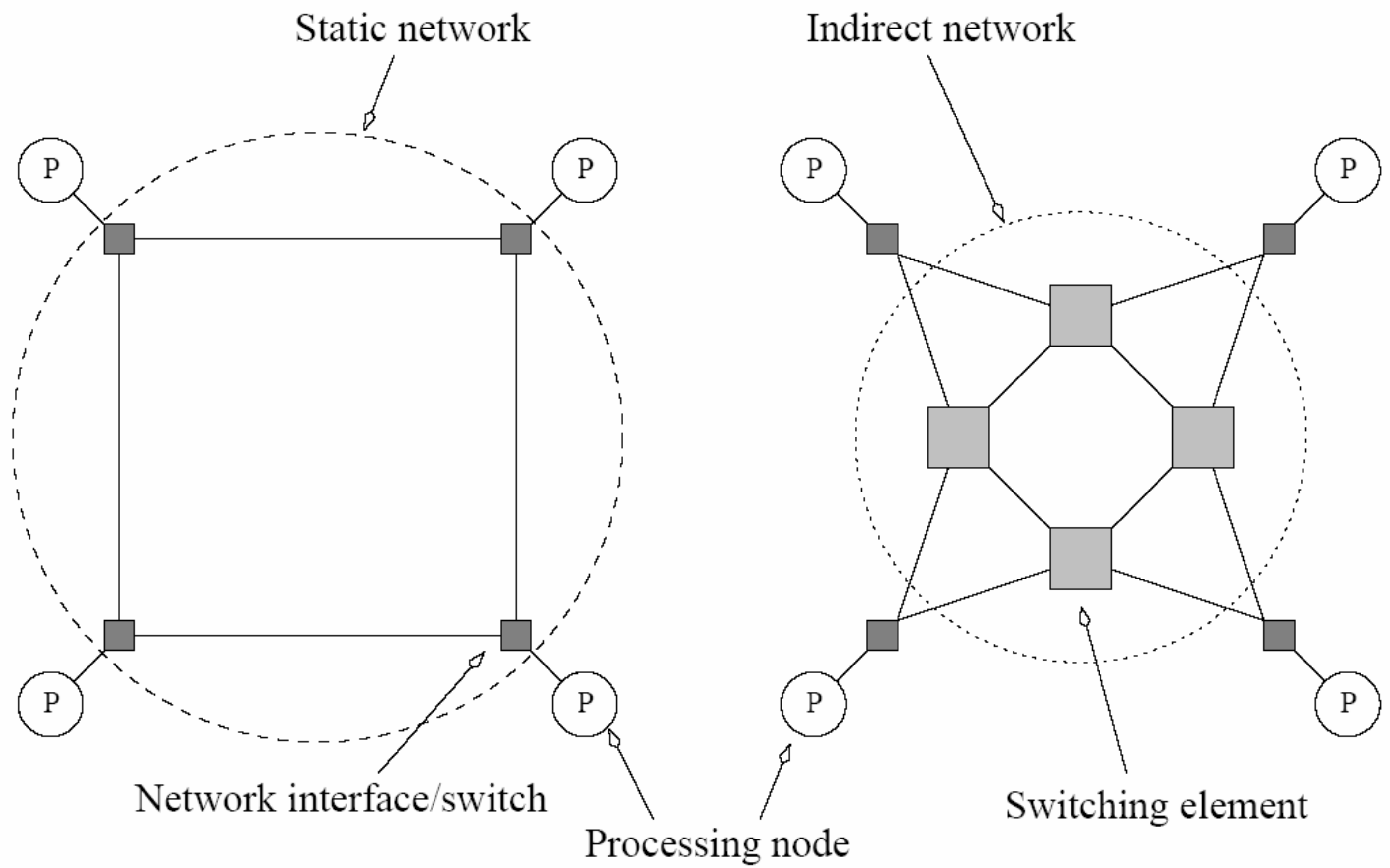
Parallel Merge

```
function merge1(A,B,n,m)[p1,…,pn+m]
        for i = 1 to n pardo pi:
                IA[i] := rank(A[i]-1,B,m)
                C[i+IA[i]] := A[i]
        od
        for i = 1 to m pardo pi:
                IB[i] := rank(B[i],A,n)
                C[i+IB[i]] := B[i]
        od
        merge1 := C
end
```

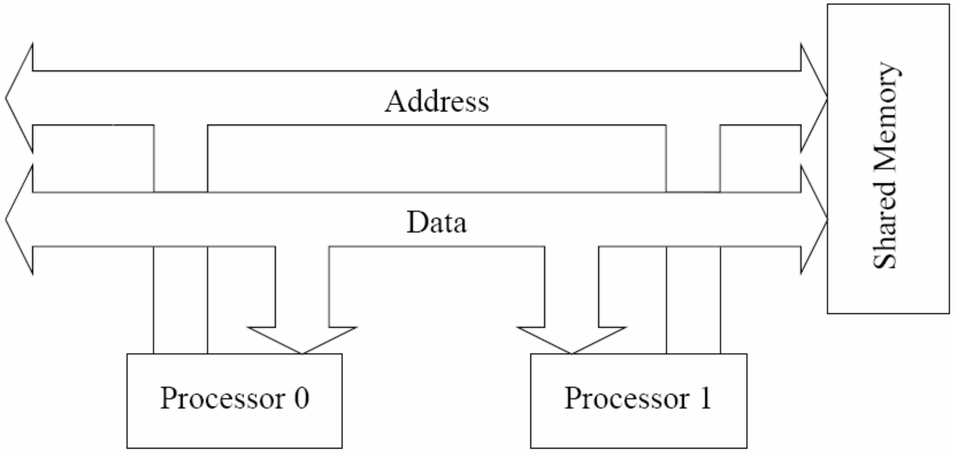
Simulating CRCW on EREW

- Assumption on addressed memory $p(n)^c$ for some constant c.
- Simulation algorithm idea:
  - Sort accesses.
  - Give priority to 1st.
  - Broadcast result for contentious accesses.
- Conclusion: Optimality can be kept with EREW-PRAM when simulating a CRCW algorithm.

Static vs. Dynamic Networks

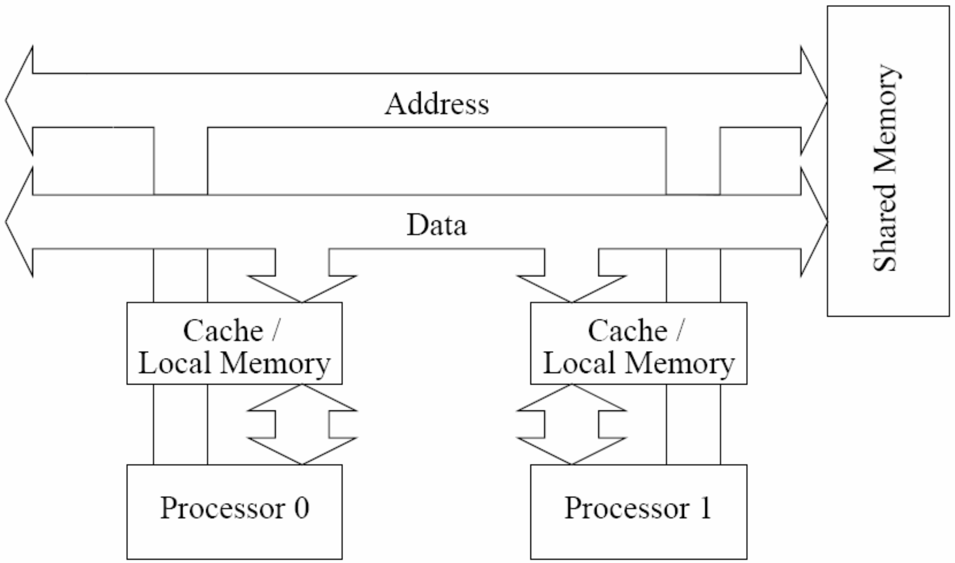


Bus Based Networks



(a)

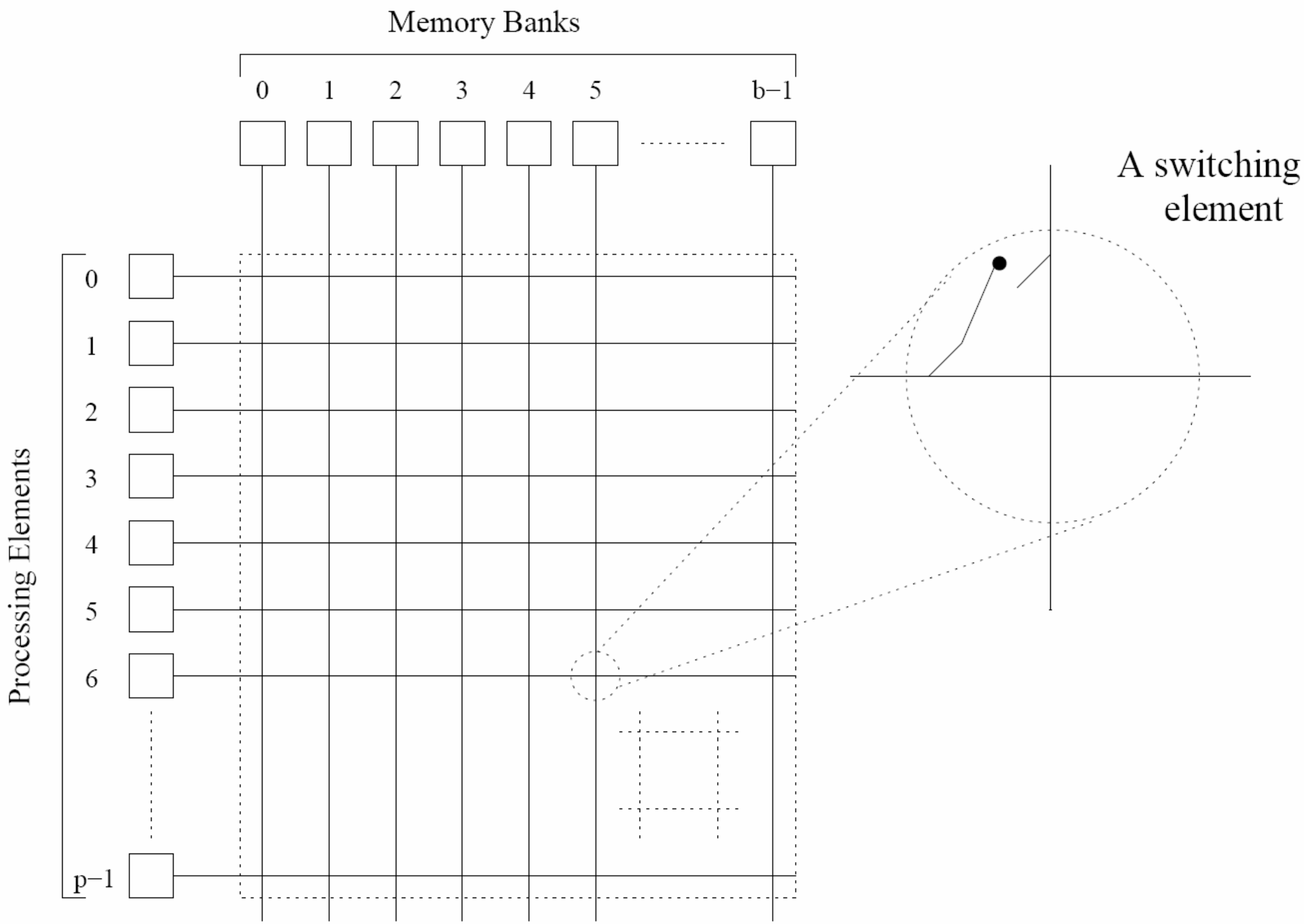
No local cache



(b)

Local cache

Crossbar Networks



Multistage Networks

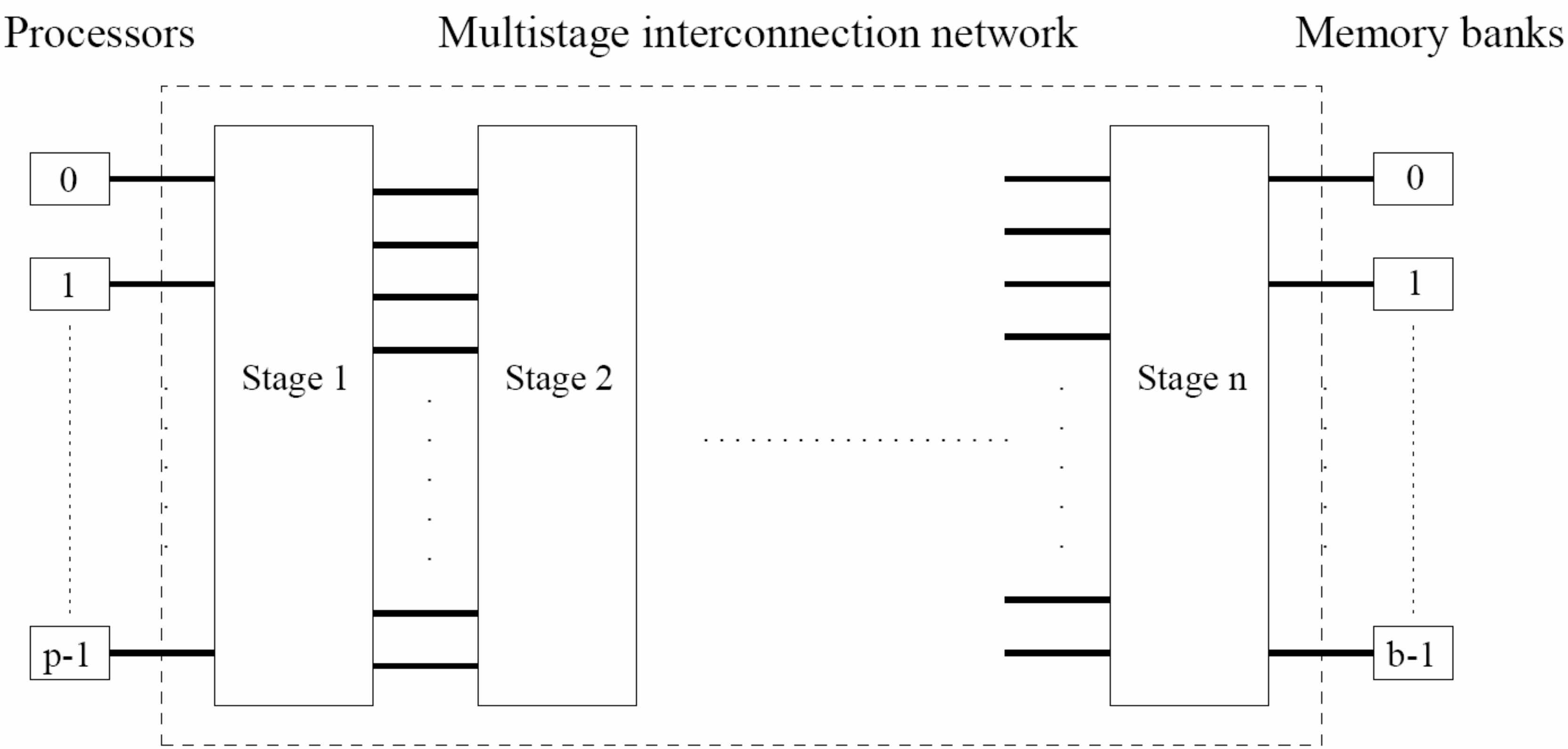
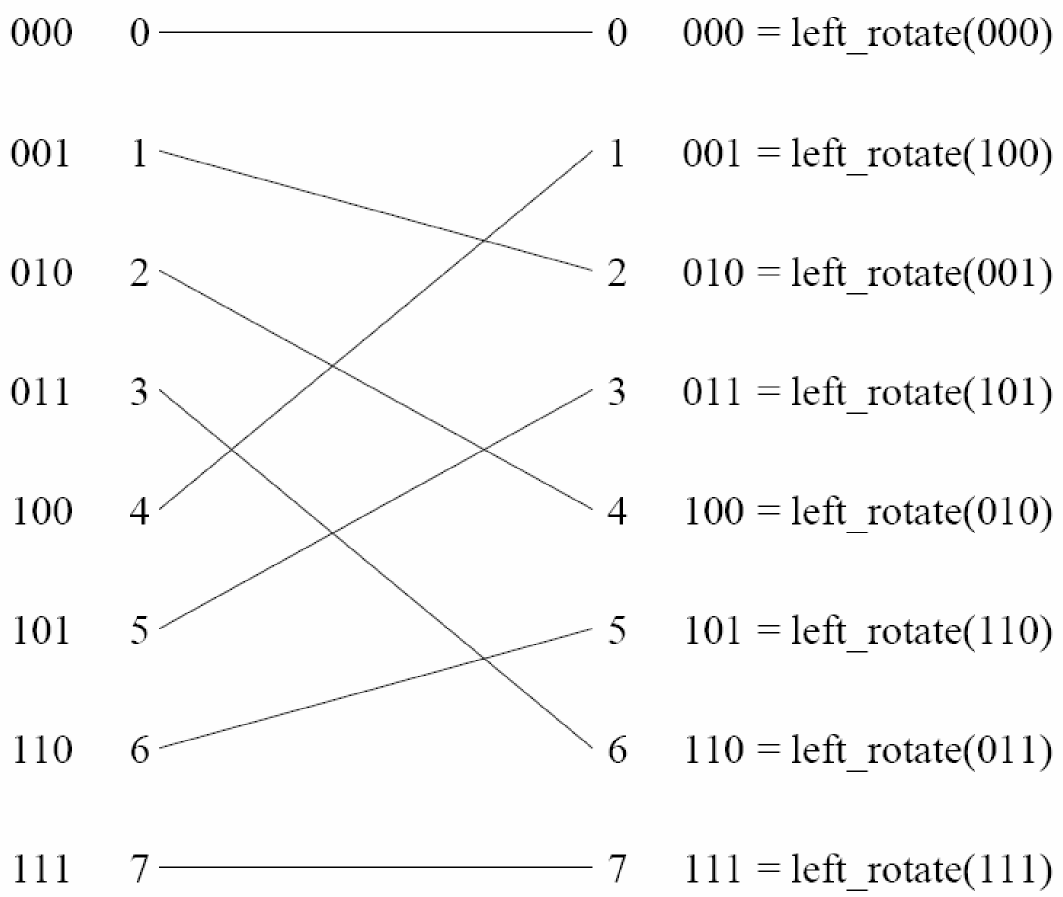
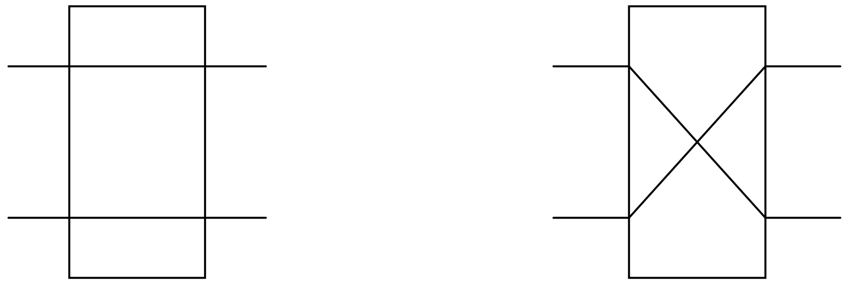


Figure 2.9 The schematic of a typical multistage interconnection network.

Perfect Shuffle Pattern



Switches in Omega Networks



Configurations: pass-through and cross-over.

p/2 * log p switching nodes:
log p stages, p/2 inputs & outputs.

Omega Network

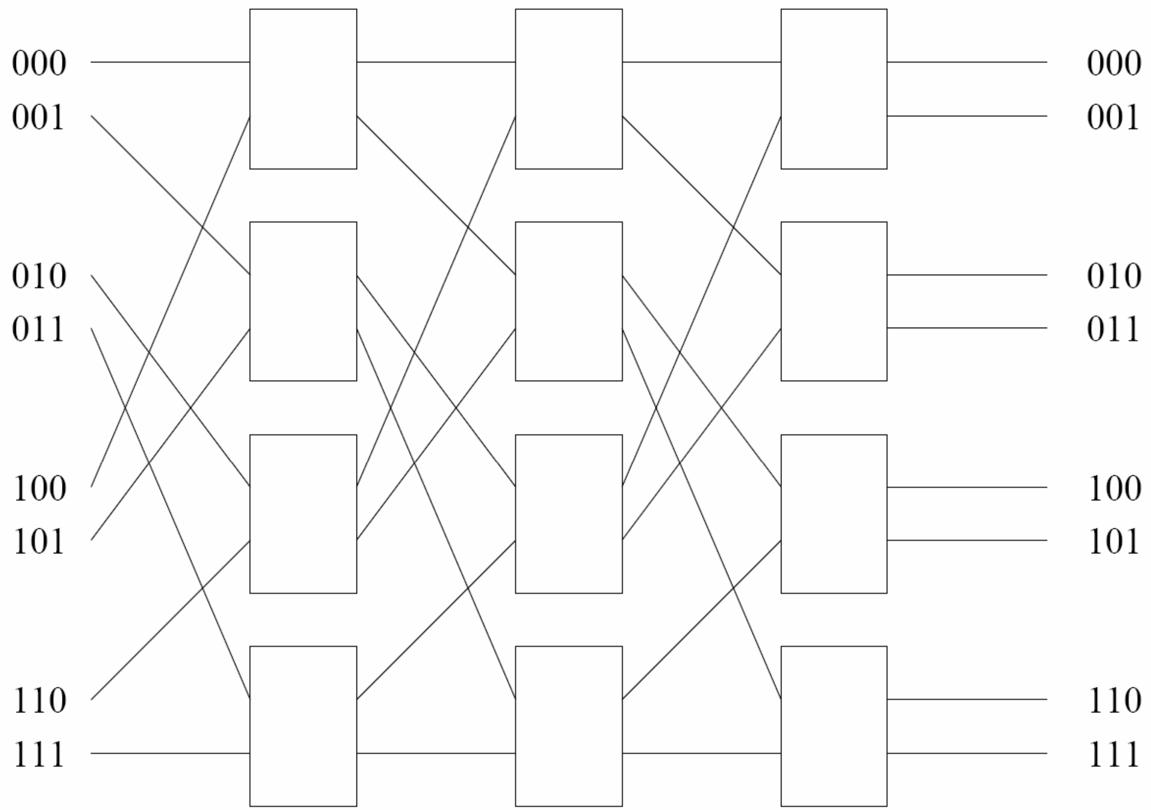


Figure 2.12 A complete omega network connecting eight inputs and eight outputs.

Blocking in Omega Networks

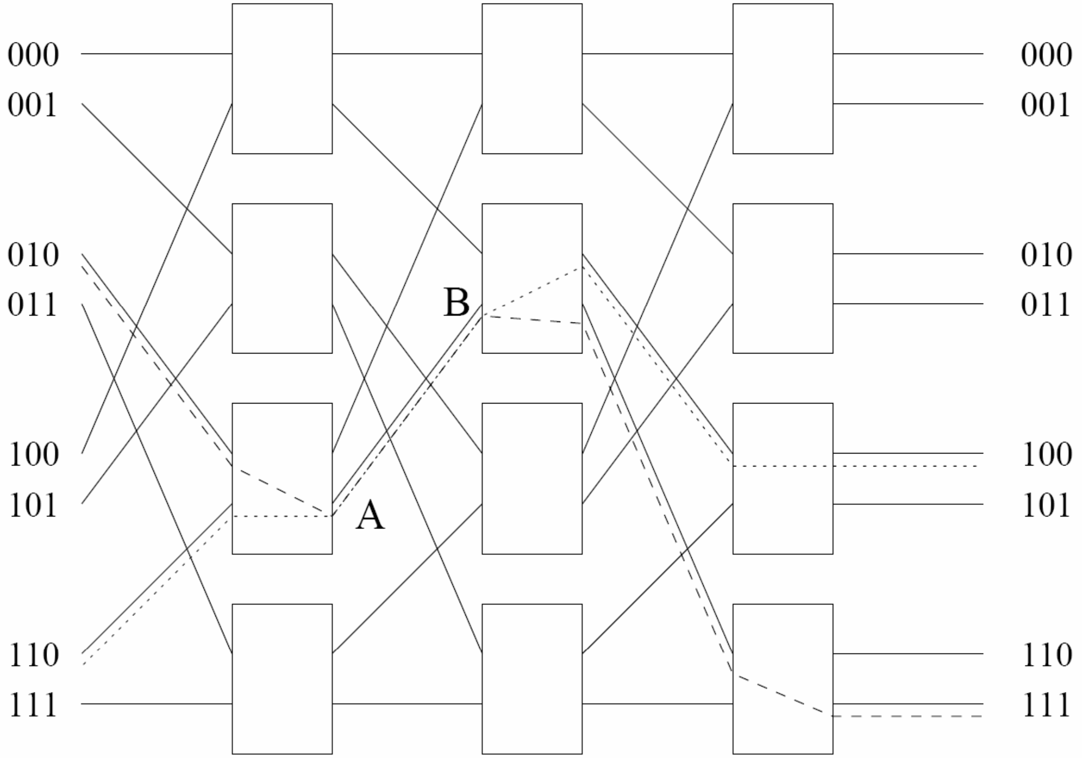


Figure 2.13 An example of blocking in omega network: one of the messages (010 to 111 or 110 to 100) is blocked at link AB.

Processors <-> Processors Networks

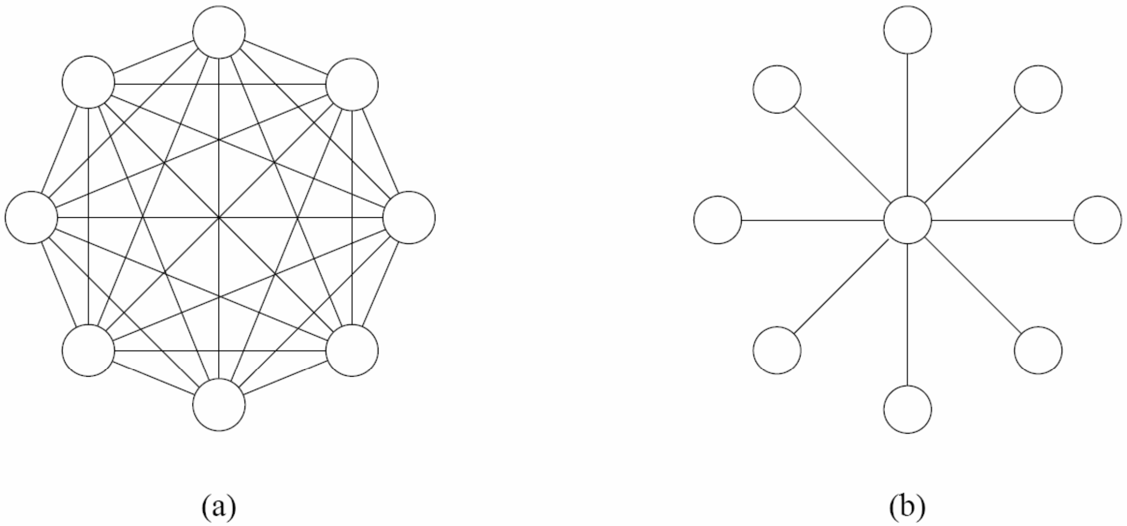
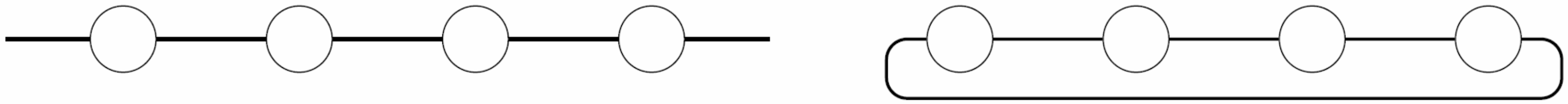


Figure 2.14 (a) A completely-connected network of eight nodes; (b) a Star connected network of nine nodes.

Performant, very expensive.

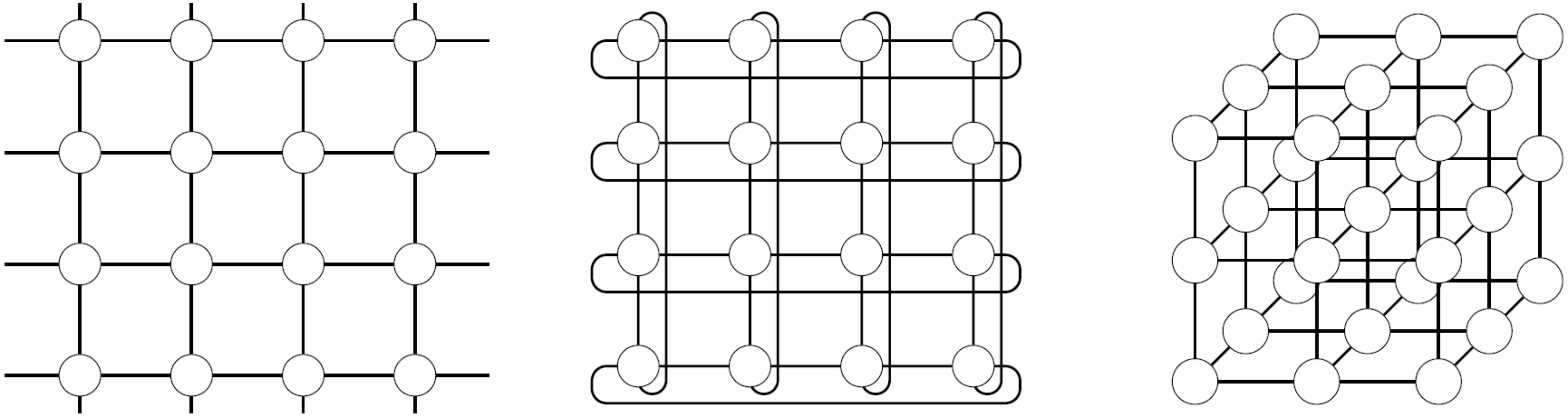
Bottleneck, cheaper.

Linear Arrays and Meshes



(a) (b)

Figure 2.15 Linear arrays: (a) with no wraparound links; (b) with wraparound link.



(a) (b) (c)

Figure 2.16 Two and three dimensional meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound.

Hypercubes

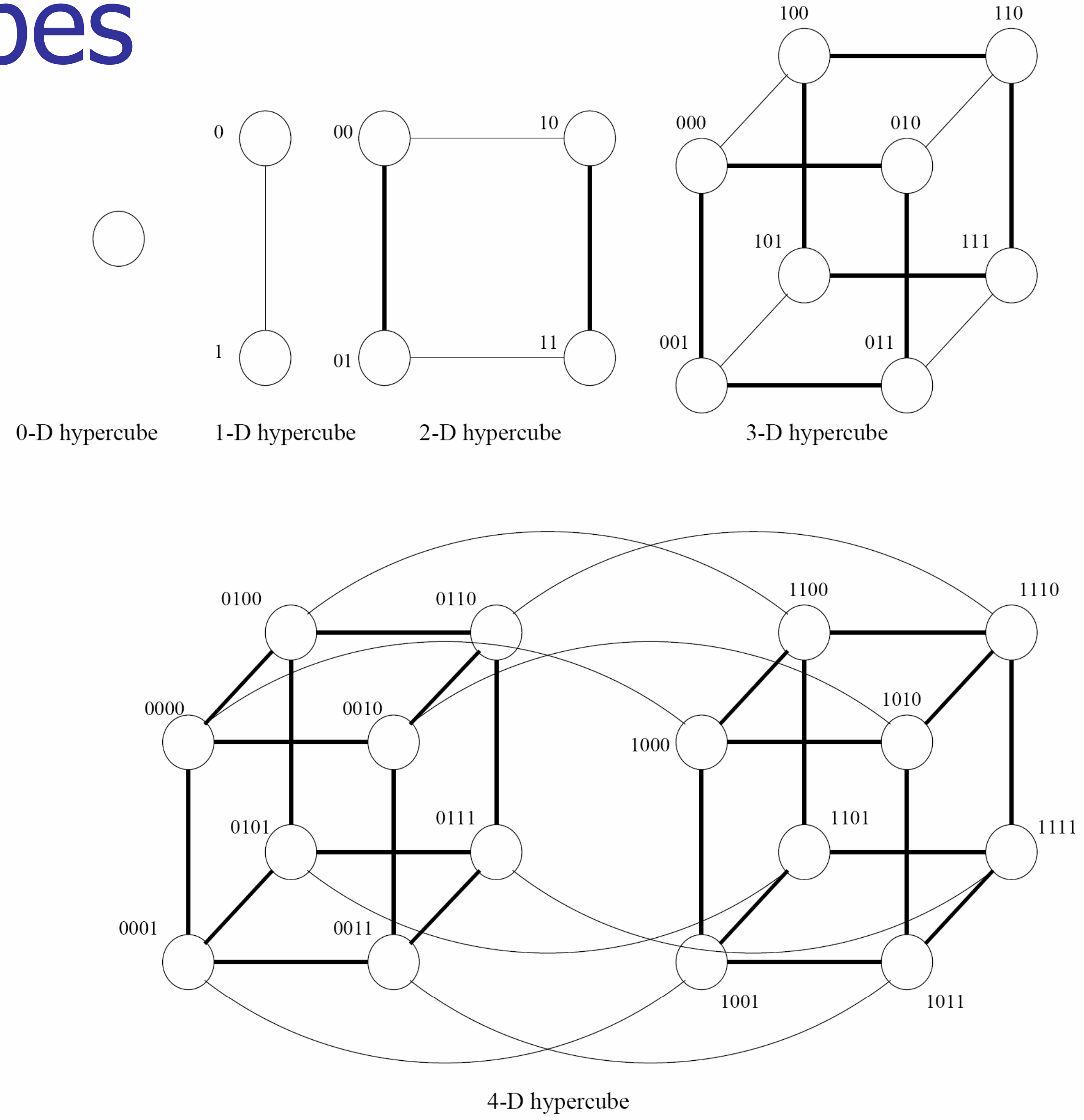


Figure 2.17 Construction of hypercubes from hypercubes of lower dimension.

Tree Based Networks

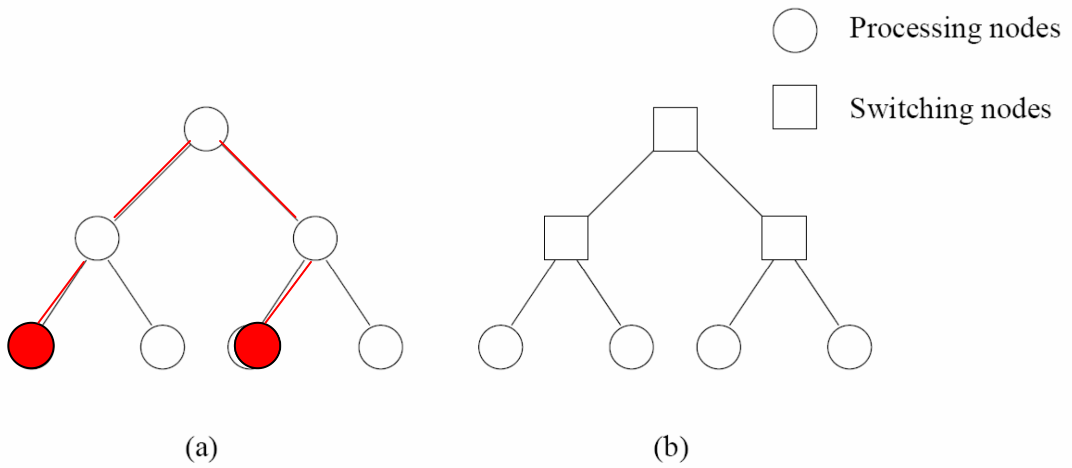


Figure 2.18 Complete binary tree networks: (a) a static tree network; and (b) a dynamic tree network.

Fat Trees

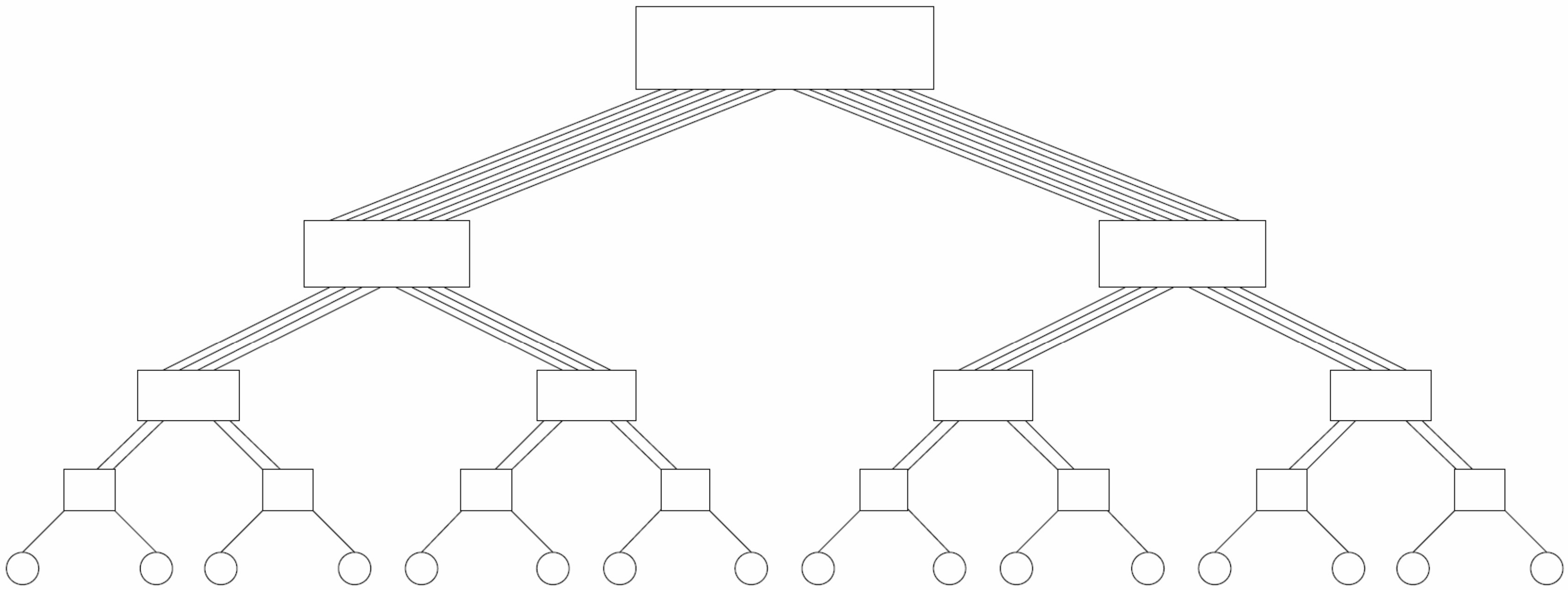


Figure 2.19 A fat tree network of 16 processing nodes.

Evaluating The Networks

- All the previous topologies have advantages and disadvantages.
- Important factors: cost and performance.
- Define criteria to characterize cost and performance.

Criteria

- Diameter: maximum distance $p_a \leftrightarrow p_b$.
- Connectivity.
- Bisection width.
- Bisection bandwidth.
- Cost.

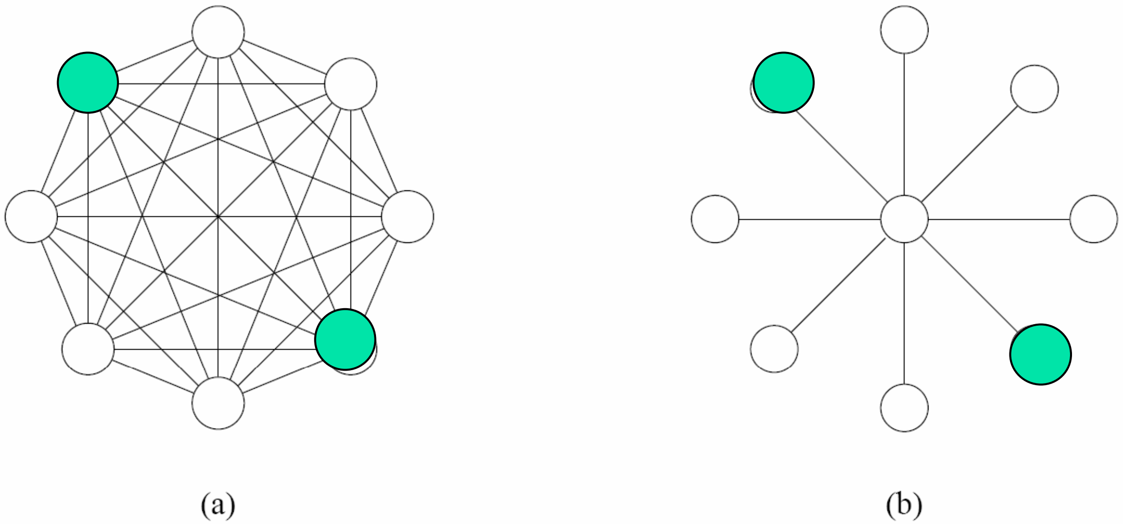
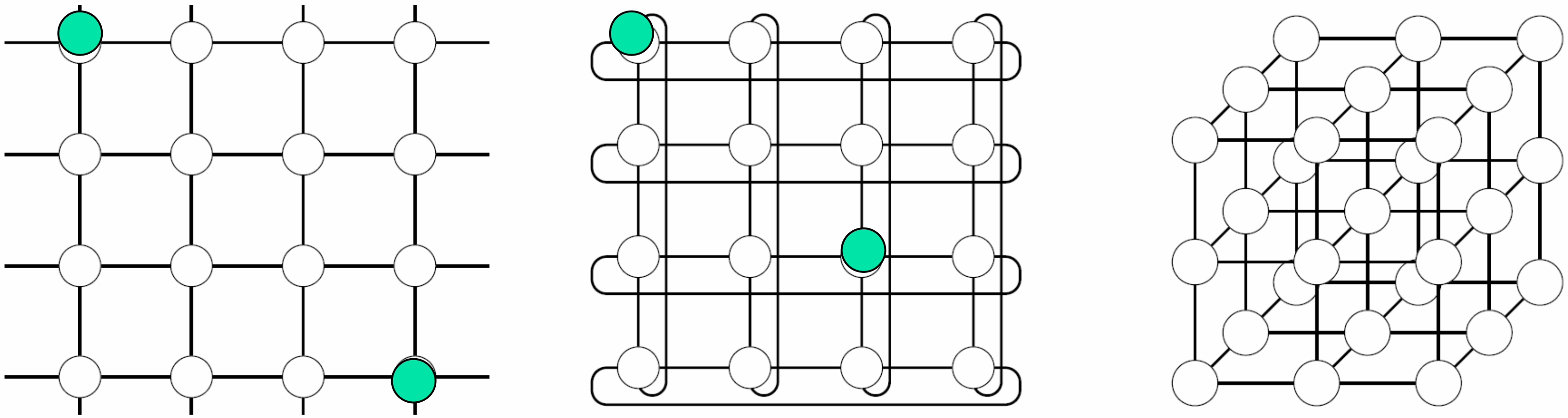


Figure 2.14 (a) A completely-connected network of eight nodes; (b) a Star connected network of nine nodes.

(a) (b)

Figure 2.15 Linear arrays: (a) with no wraparound links; (b) with wraparound link.



(a) (b) (c)

Figure 2.16 Two and three dimensional meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound.

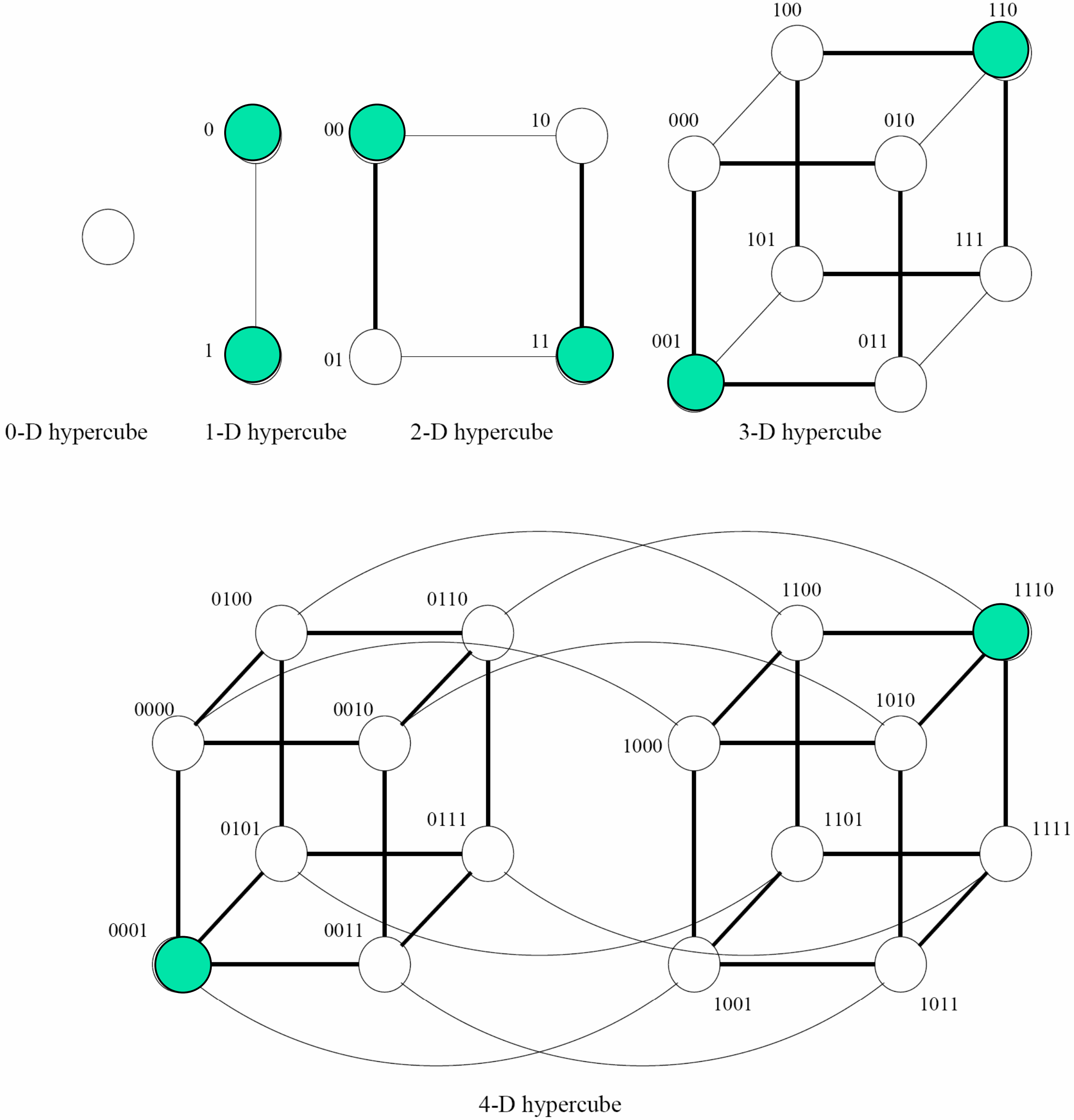


Figure 2.17 Construction of hypercubes from hypercubes of lower dimension.

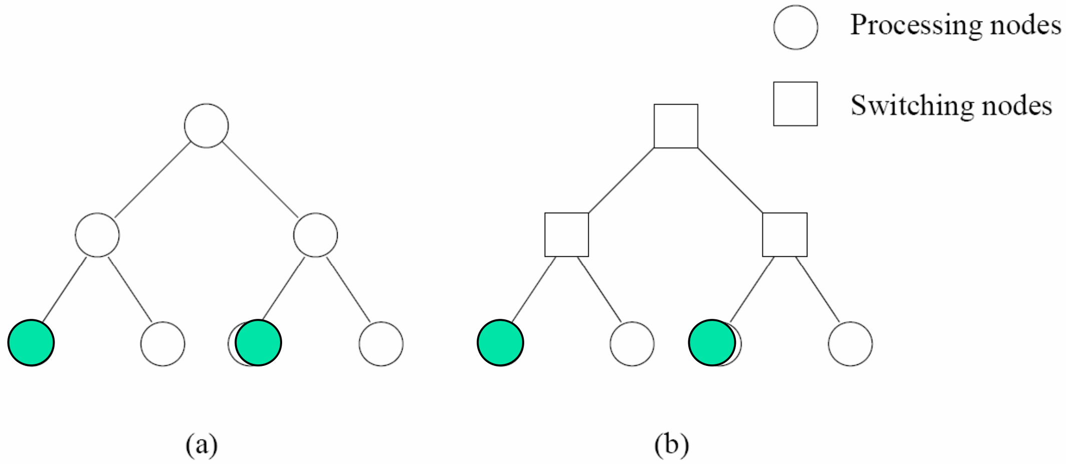


Figure 2.18 Complete binary tree networks: (a) a static tree network; and (b) a dynamic tree network.

Criteria

- Diameter.
- Connectivity: measure of multiplicity of paths.
- Bisection width.
- Bisection bandwidth.
- Cost.

Criteria

- Diameter.
- Connectivity.
- Bisection width: minimum number of links to cut in order to partition the network in 2 equal halves.
- Bisection bandwidth: minimum volume of communication allowed between 2 halves.
- Cost.

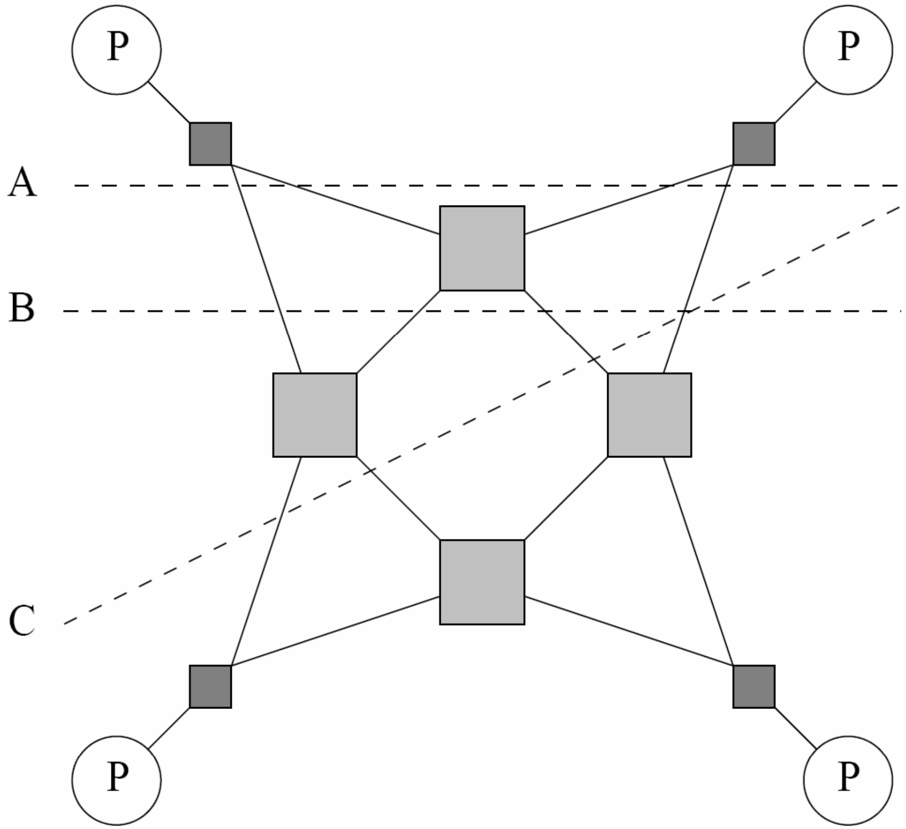


Figure 2.20 Bisection width of a dynamic network is computed by examining various equi-partitions of the processing nodes and selecting the minimum number of edges crossing the partition. In this case, each partition yields an edge cut of four. Therefore, the bisection width of this graph is four.

Criteria

- Diameter.
- Connectivity.
- Bisection width.
- Bisection bandwidth.
- Cost: number of communication links, i.e., wires.

Comparing The Topologies

Table 2.1 A summary of the characteristics of various static network topologies connecting p nodes.

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p-1$	$p(p-1)/2$
Star	2	1	1	$p-1$
Complete binary tree	$2\log((p+1)/2)$	1	1	$p-1$
Linear array	$p-1$	1	1	$p-1$
2-D mesh, no wraparound	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\lfloor\sqrt{p}/2\rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p\log p)/2$
Wraparound k-ary d-cube	$d\lfloor k/2\rfloor$	$2k^{d-1}$	$2d$	dp

Cache Coherence Protocols

- We need additional hardware to keep *multiple copies* of the same memory bank *consistent* with each other.
- We have seen that $$ is good but it does not come for free.
- Mechanism known as cache coherence protocol, usually described as state machines.

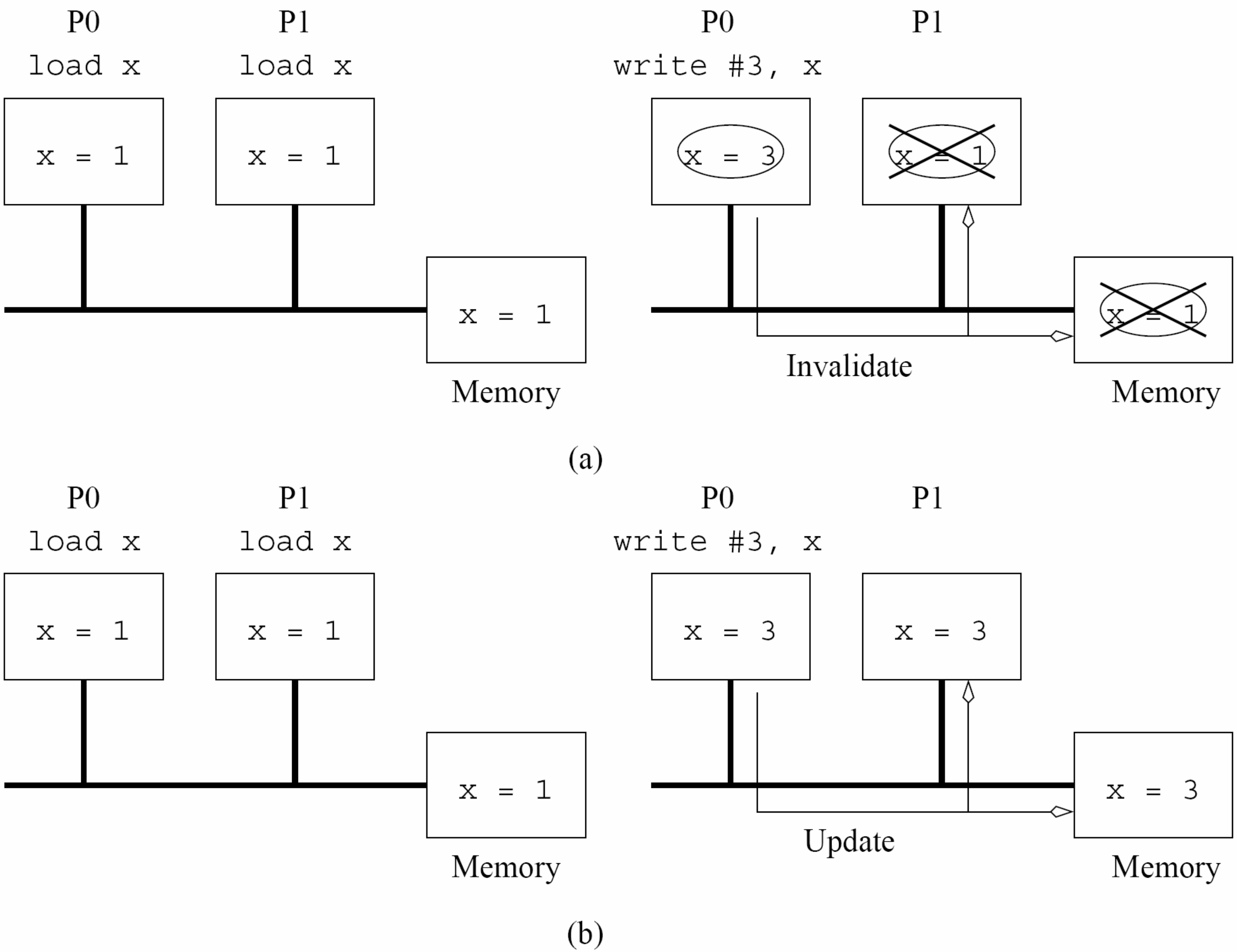


Figure 2.21 Cache coherence in multiprocessor systems: (a) Invalidate protocol; (b) Update protocol for shared variables.

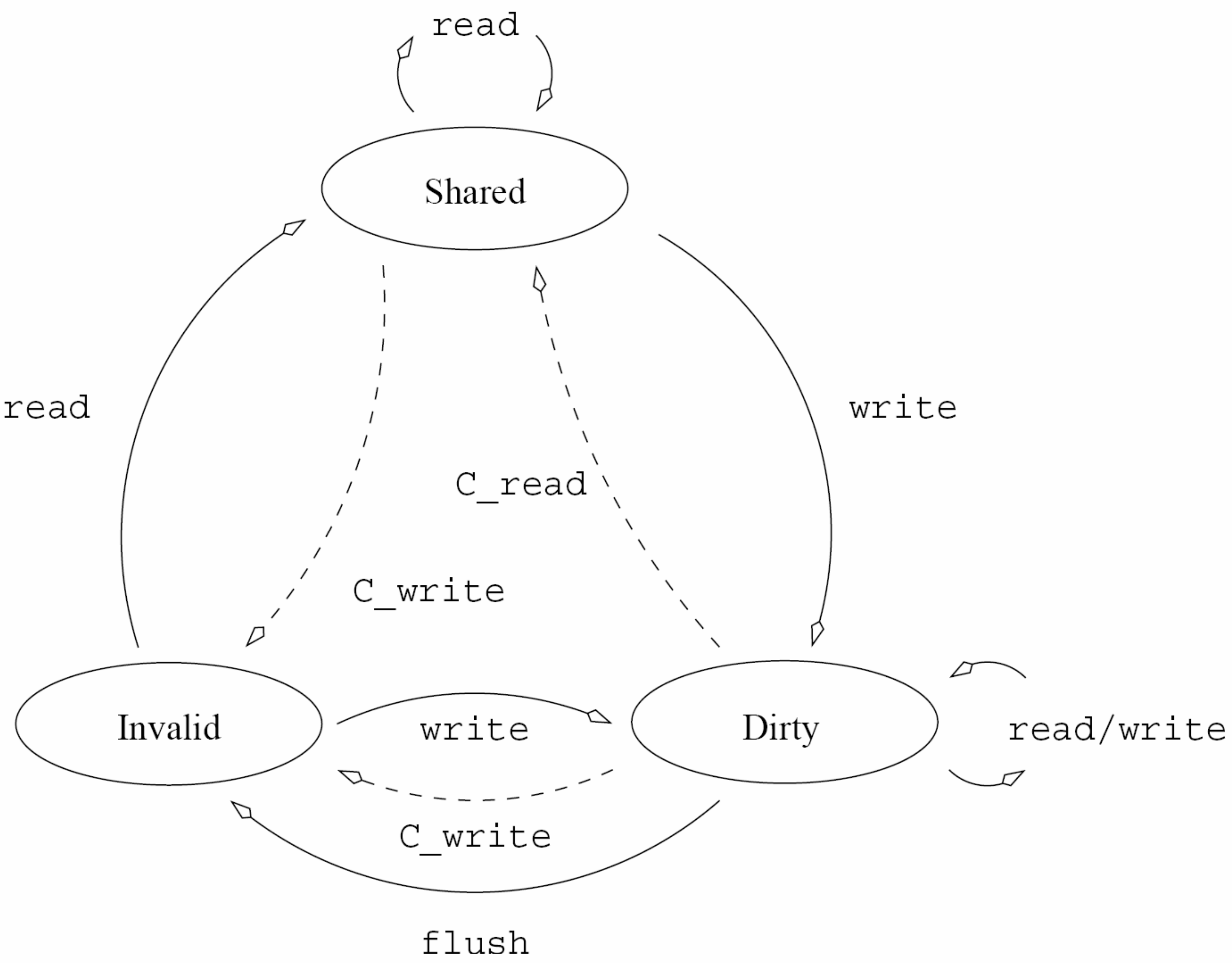


Figure 2.22 State diagram of a simple three-state coherence protocol.

Implementations of Cache Coherence Protocols

- Different ways to implement the protocol described by the state machine.
  - Snoopy cache: good on busses.
    Snoopy hardware that monitors states.
  - Directory based systems: states and presence bits for cache lines.
  - Distributed directory: physically distribute directory with memory.

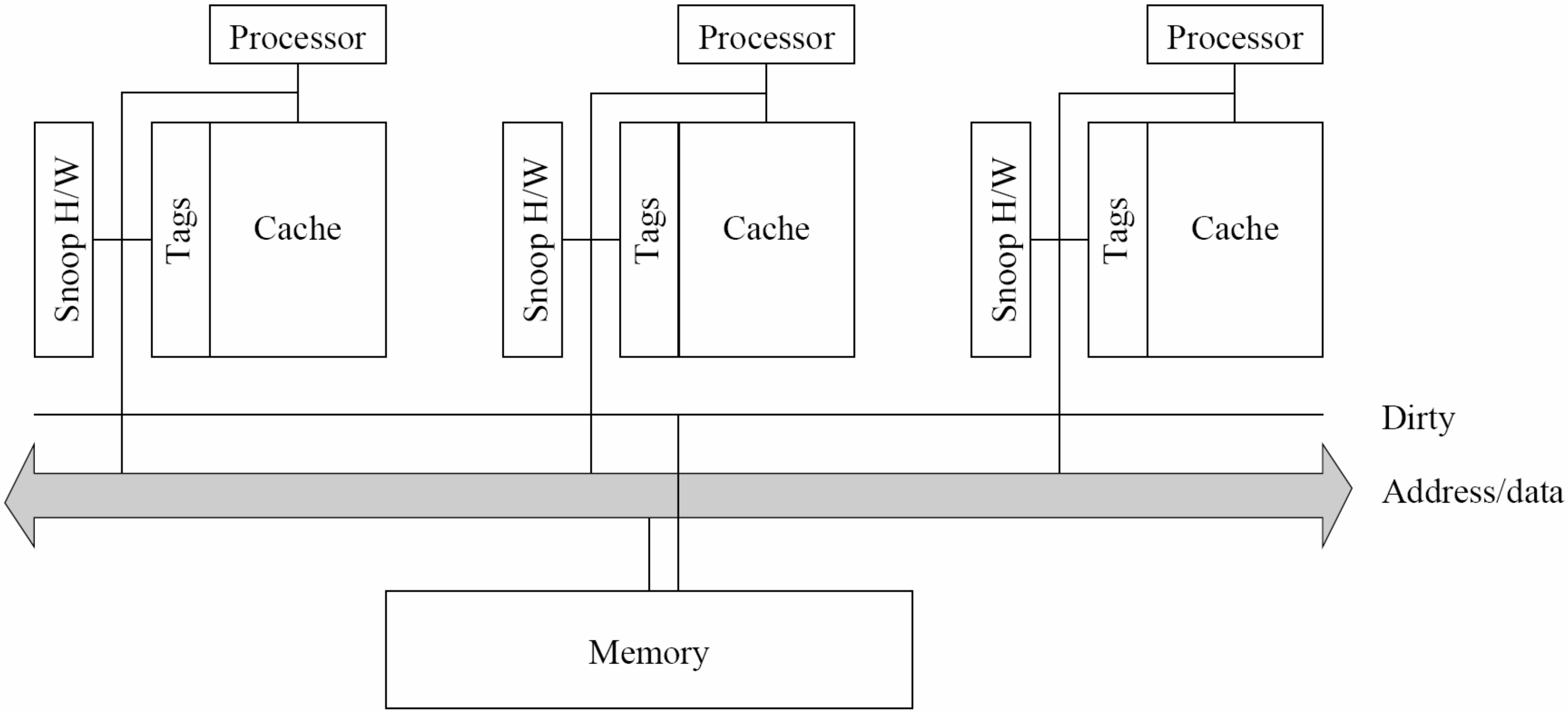


Figure 2.24 A simple snoopy bus based cache coherence system.

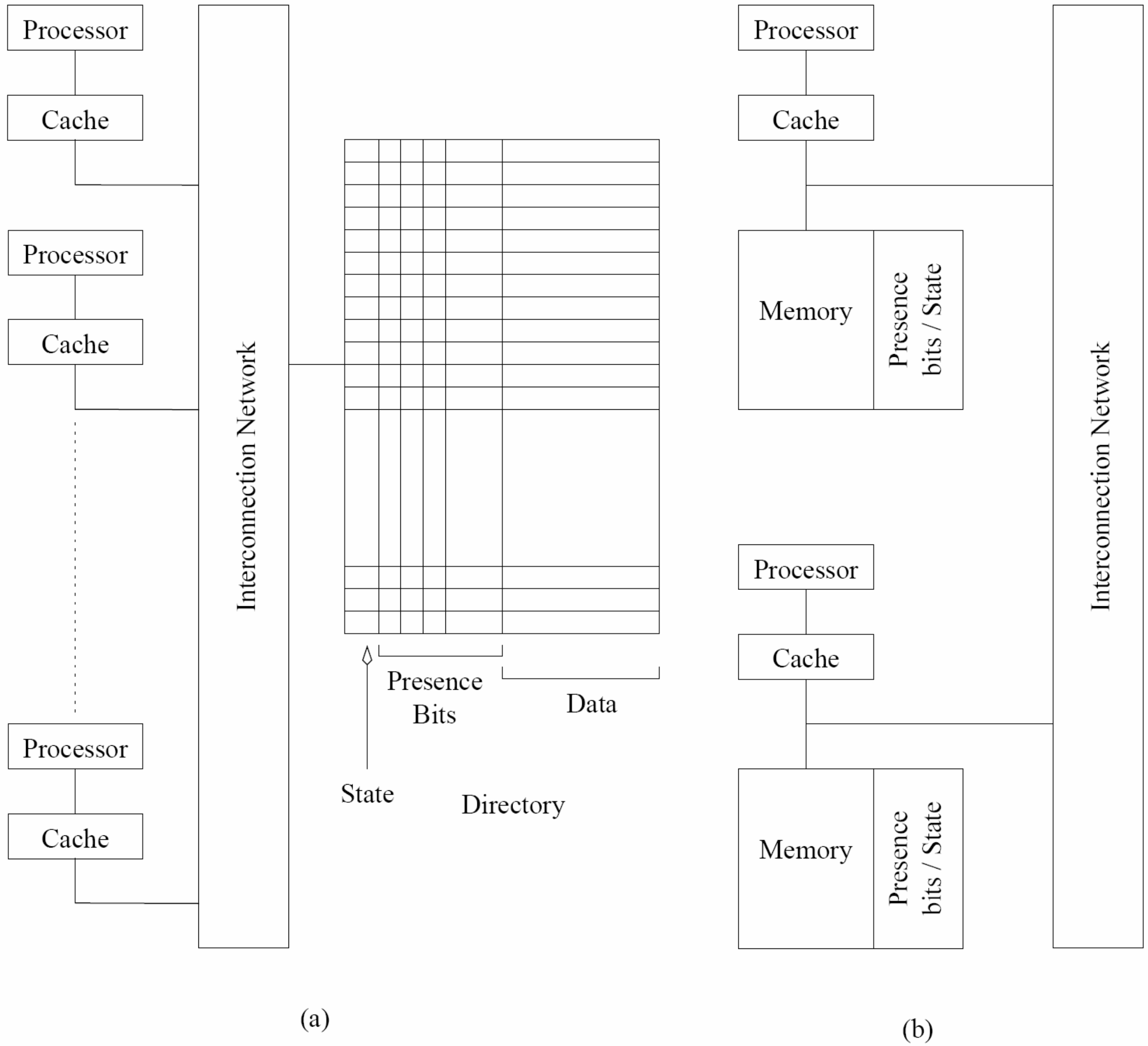


Figure 2.25 Architecture of typical directory based systems: (a) a centralized directory; and (b) a distributed directory.