8

PARALLEL COMPUTER ARCHITECTURES

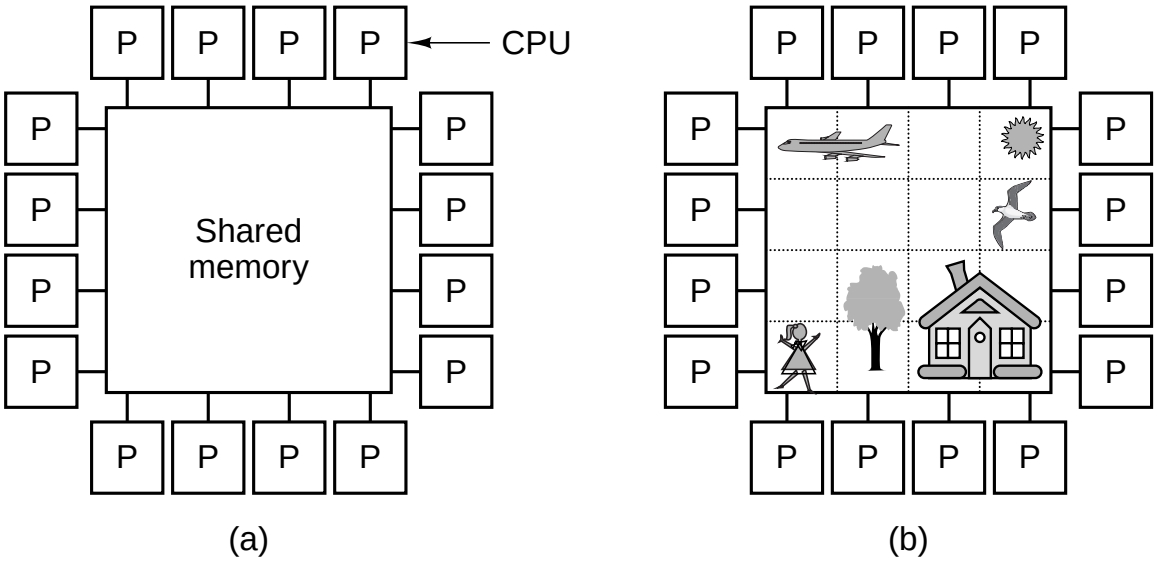

Figure 8-1. (a) A multiprocessor with 16 CPUs sharing a common memory. (b) An image partitioned into 16 sections, each being analyzed by a different CPU.

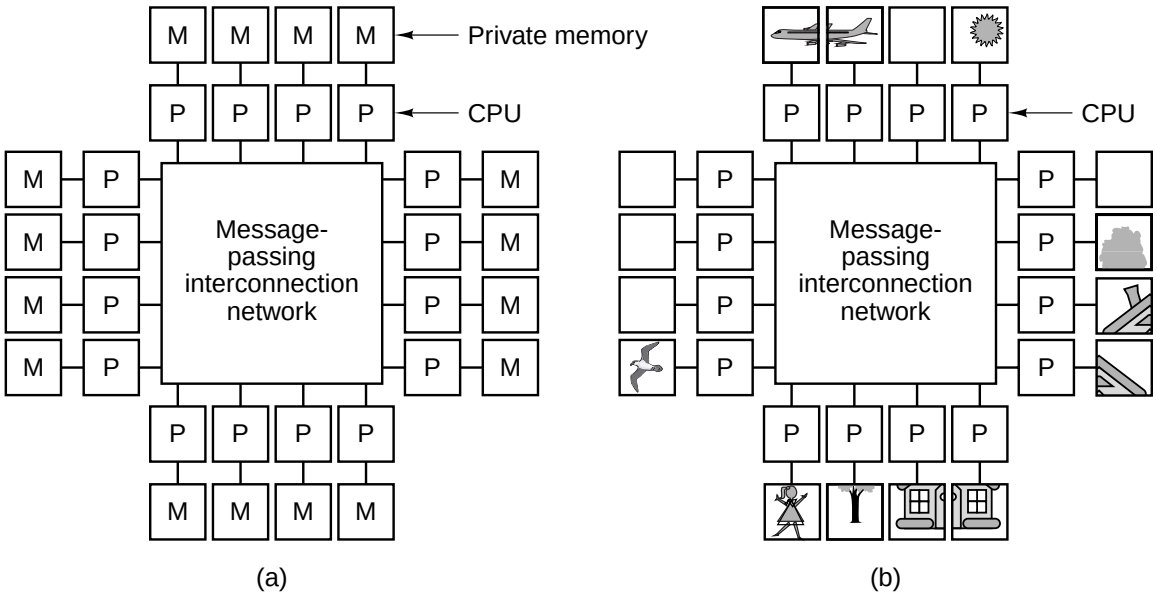

Figure 8-2. (a) A multicomputer with 16 CPUs, each with each own private memory. (b) The bit-map image of Fig. 8-1 split up among the 16 memories.

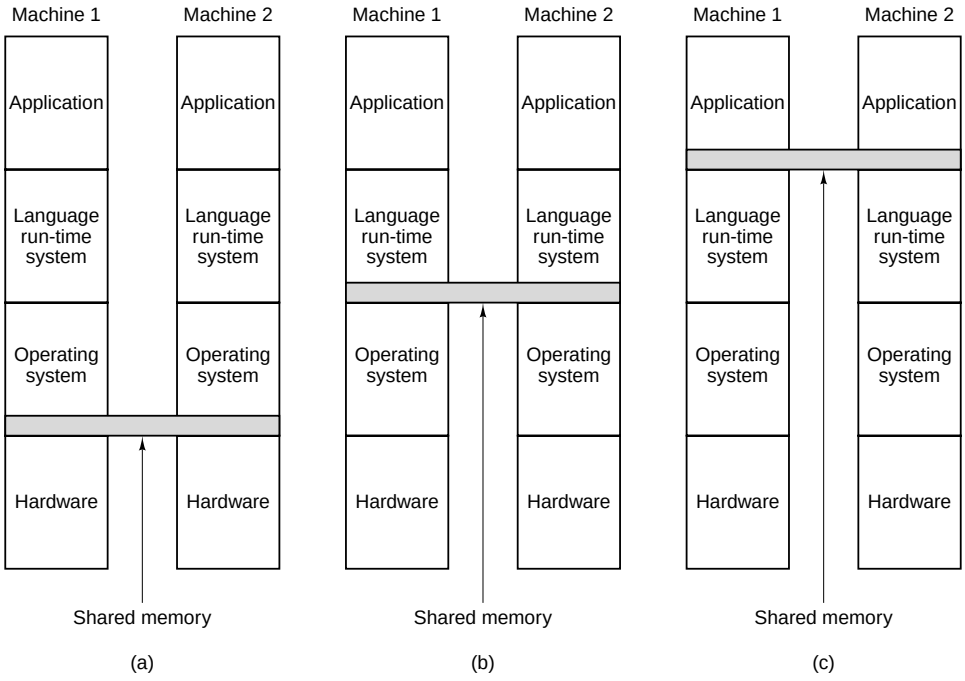

Figure 8-3. Various layers where shared memory can be implemented. (a) The hardware. (b) The operating system. (c) The language runtime system.

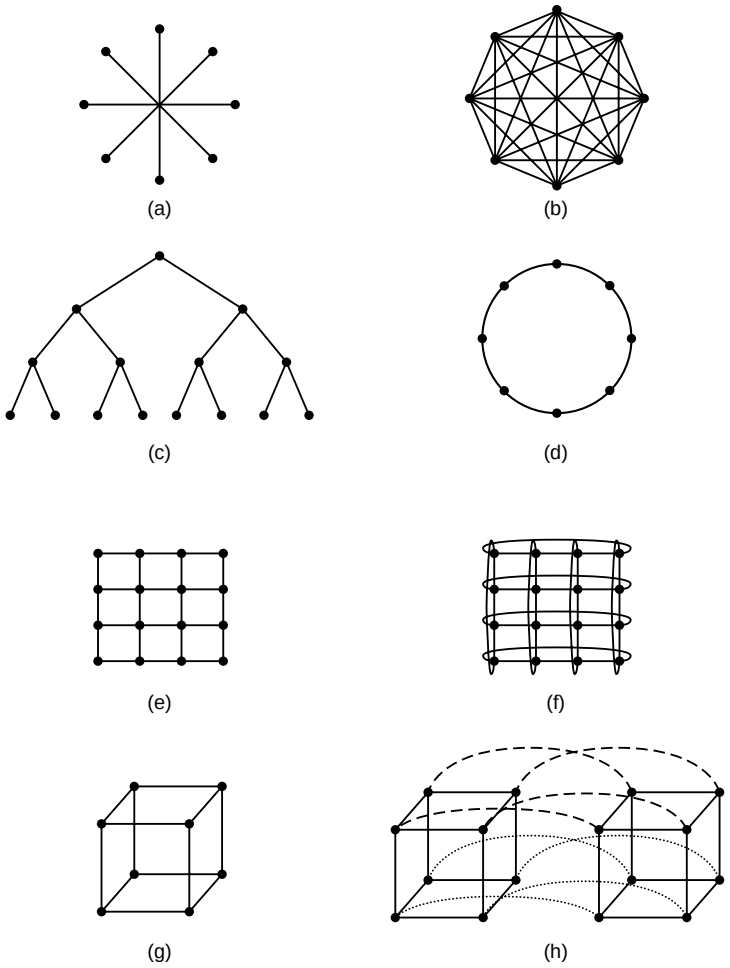

Figure 8-4. Various topologies. The heavy dots represent switches. The CPUs and memories are not shown. (a) A star. (b) A complete interconnect. (c) A tree. (d) A ring. (e) A grid. (f) A double torus. (g) A cube. (h) A 4D hypercube.

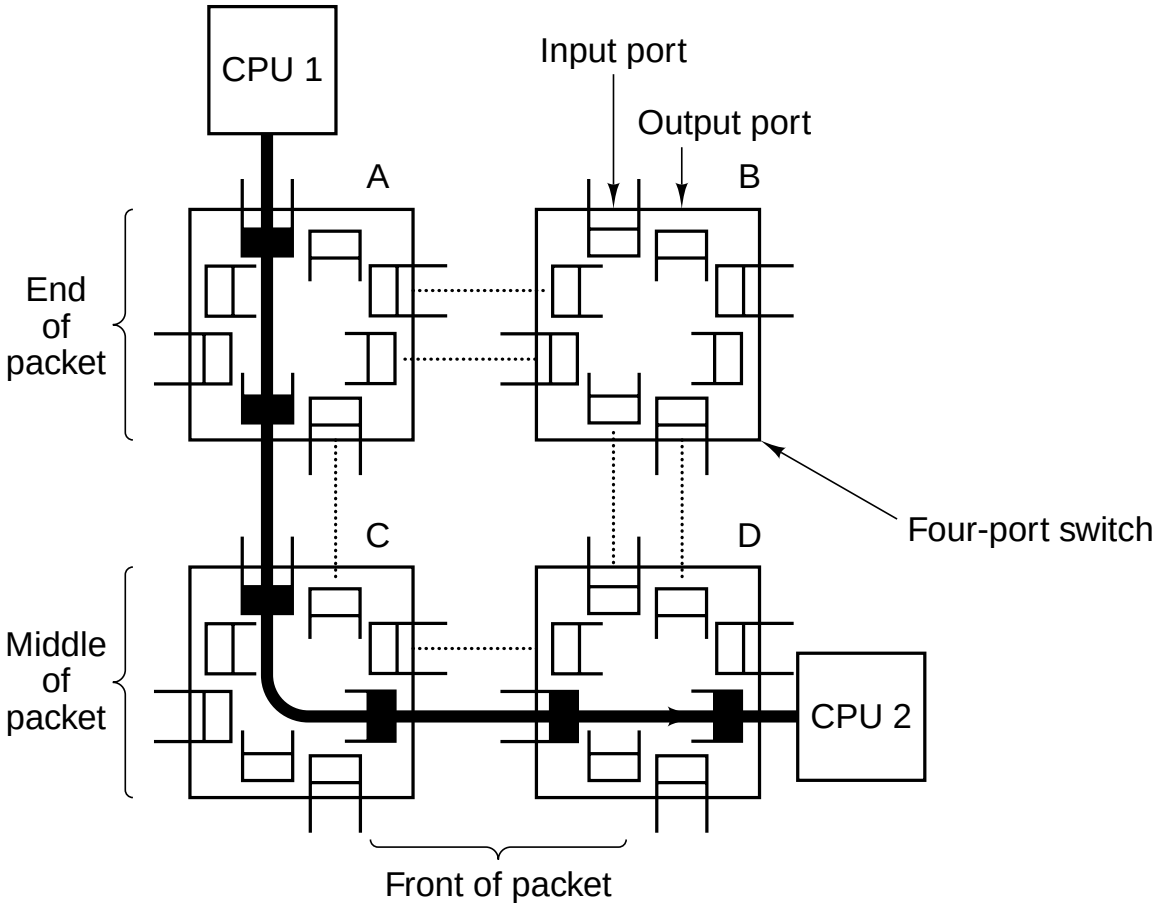

Figure 8-5. An interconnection network in the form of a four-switch square grid. Only two of the CPUs are shown.

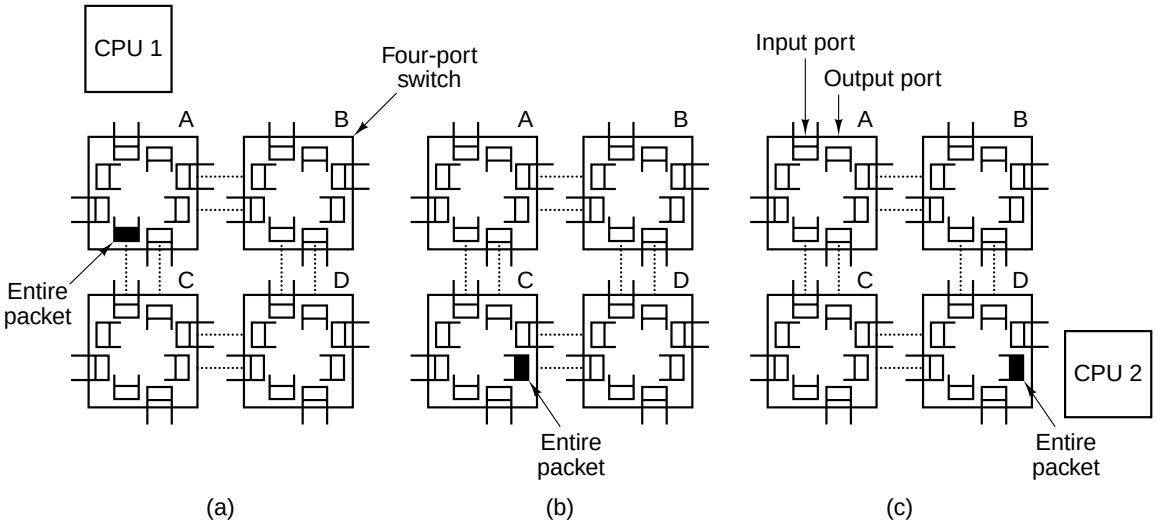

Figure 8-6. Store-and-forward packet switching.

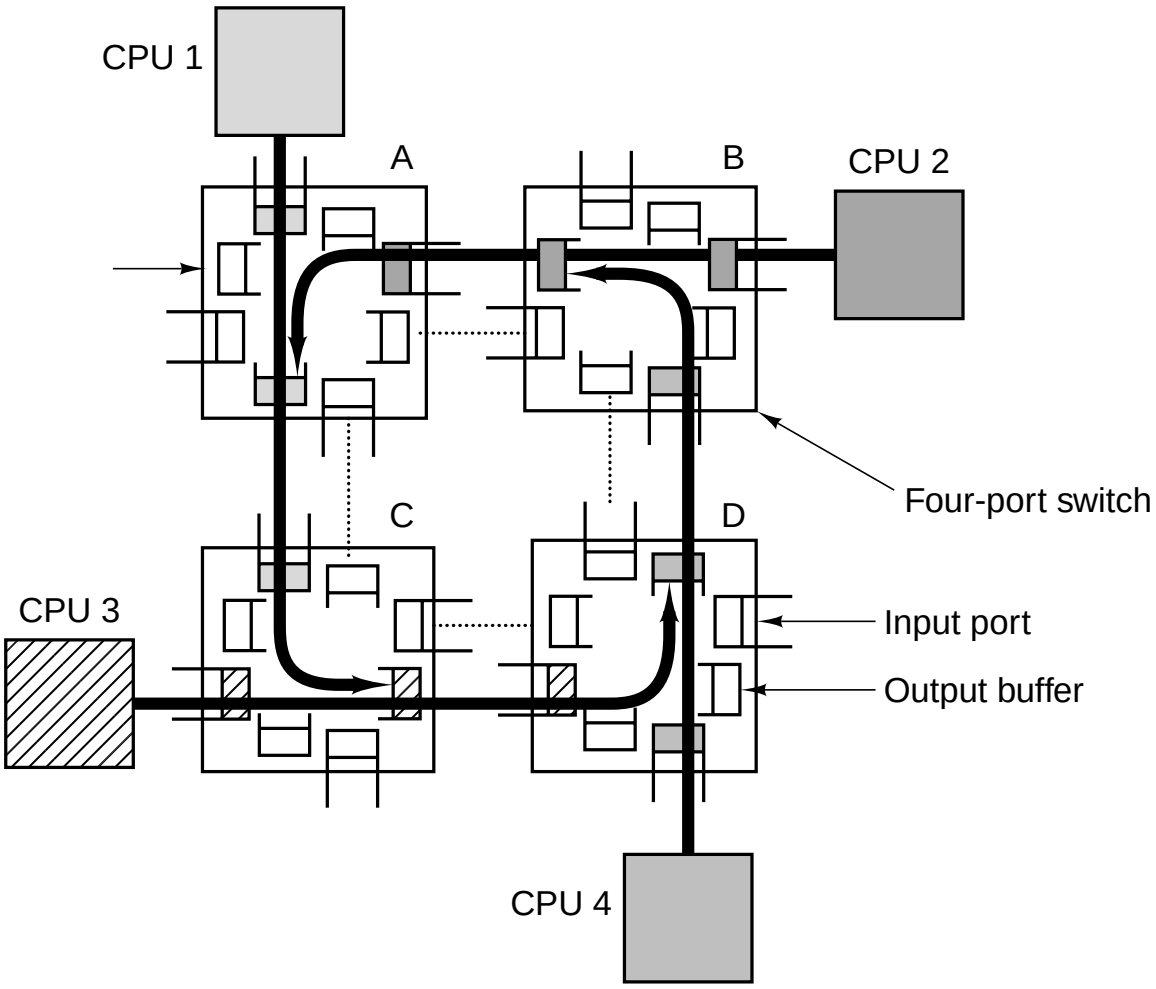

Figure 8-7. Deadlock in a circuit-switched interconnection network.

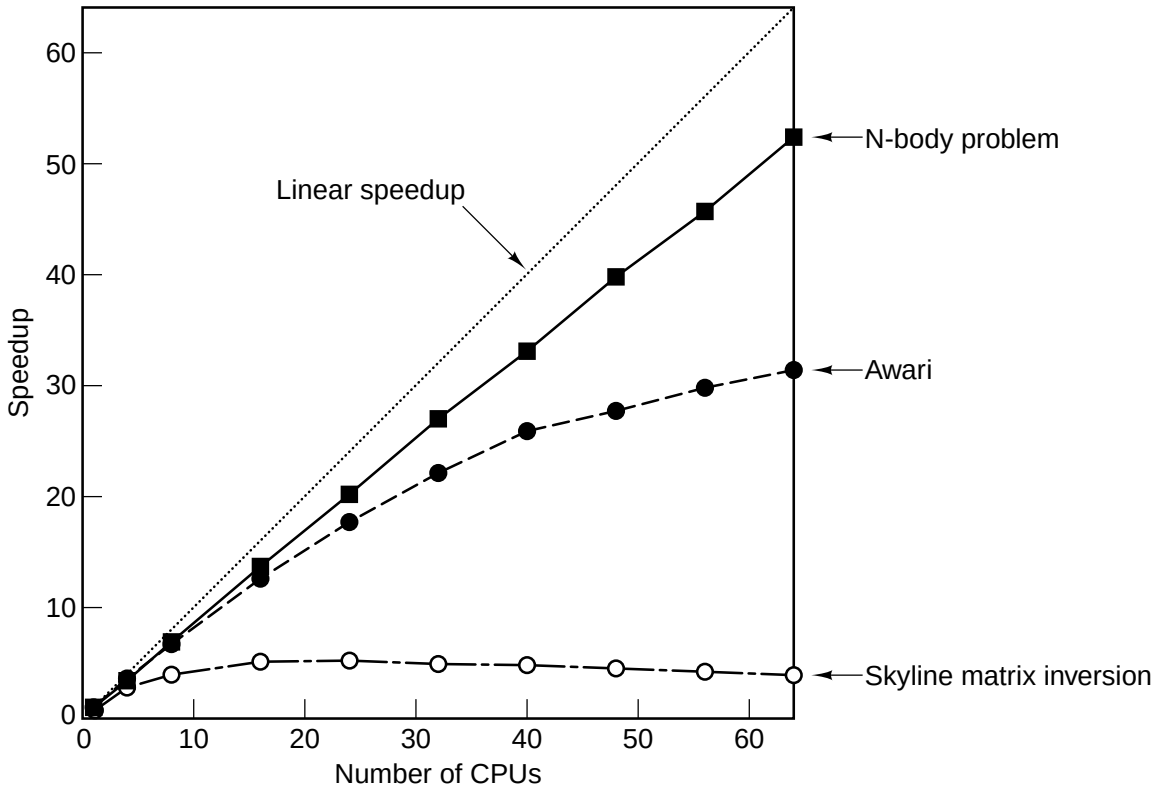

Figure 8-8. Real programs achieve less than the perfect speed-up indicated by the dotted line.

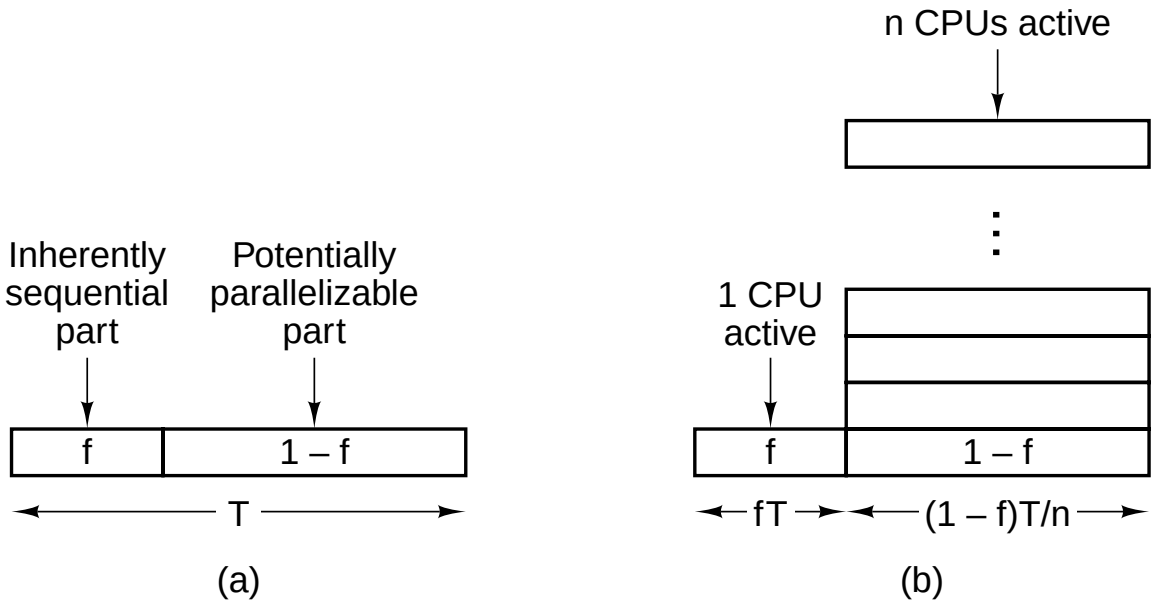

Figure 8-9. (a) A program has a sequential part and a parallelizable part. (b) Effect of running part of the program in parallel.

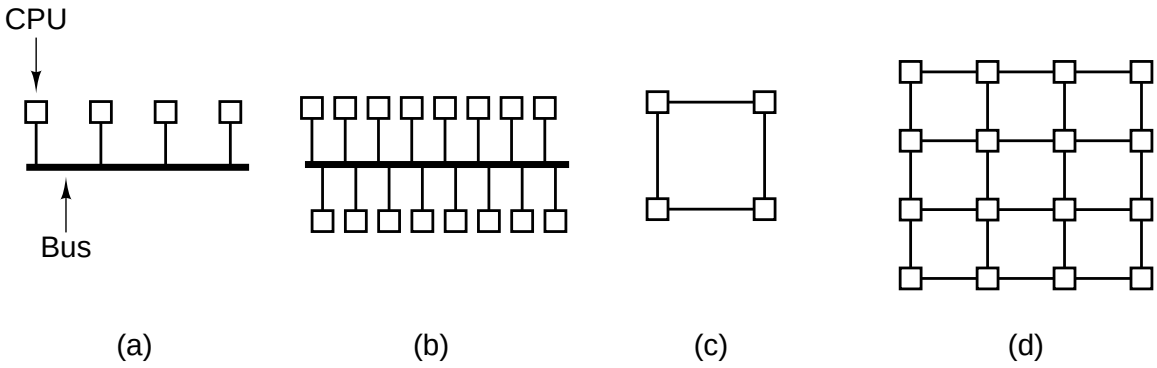

Figure 8-10. (a) A 4-CPU bus-based system. (b) A 16-CPU bus-based system. (c) A 4-CPU grid-based system. (d) A 16-CPU grid-based system.

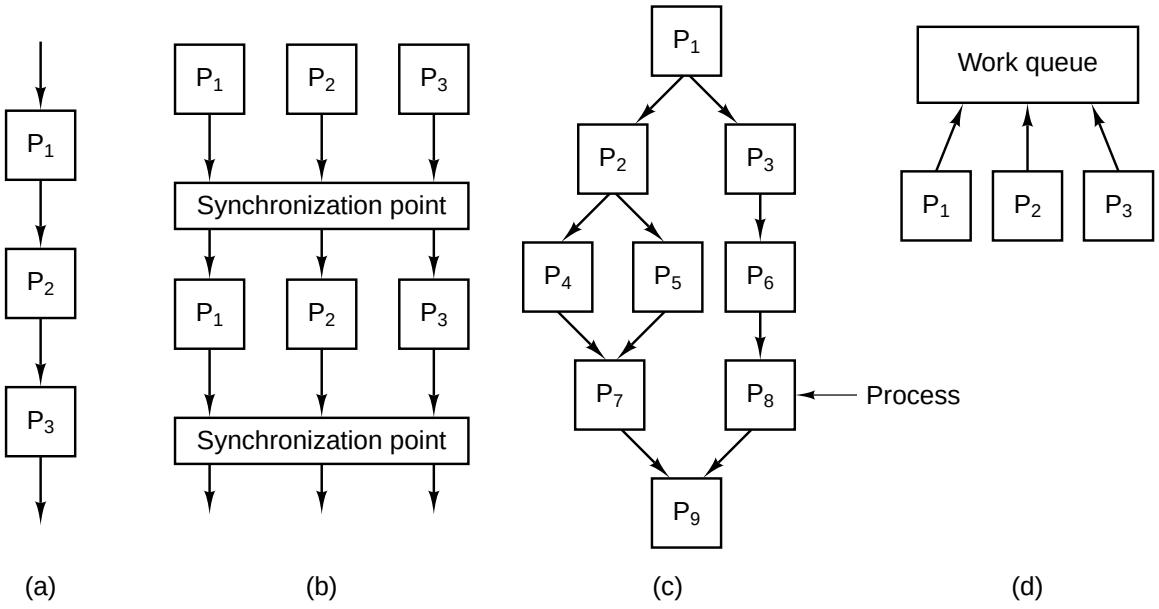

Figure 8-11. Computational paradigms. (a) Pipeline. (b) Phased computation. (c) Divide and conquer. (d) Replicated worker.

Physical (hardware)	Logical (software)	Examples
Multiprocessor	Shared variables	Image processing as in Fig. 8-1
Multiprocessor	Message passing	Message passing simulated with buffers in memory
Multicomputer	Shared variables	DSM, Linda, Orca, etc. on an SP/2 or a PC network
Multicomputer	Message passing	PVM or MPI on an SP/2 or a network of PCs

Figure 8-12. Combinations of physical and logical sharing.

Instruction streams	Data streams	Name	Examples
1	1	SISD	Classical Von Neumann machine
1	Multiple	SIMD	Vector supercomputer, array processor
Multiple	1	MISD	Arguably none
Multiple	Multiple	MIMD	Multiprocessor, multicomputer

Figure 8-13. Flynn's taxonomy of parallel computers.

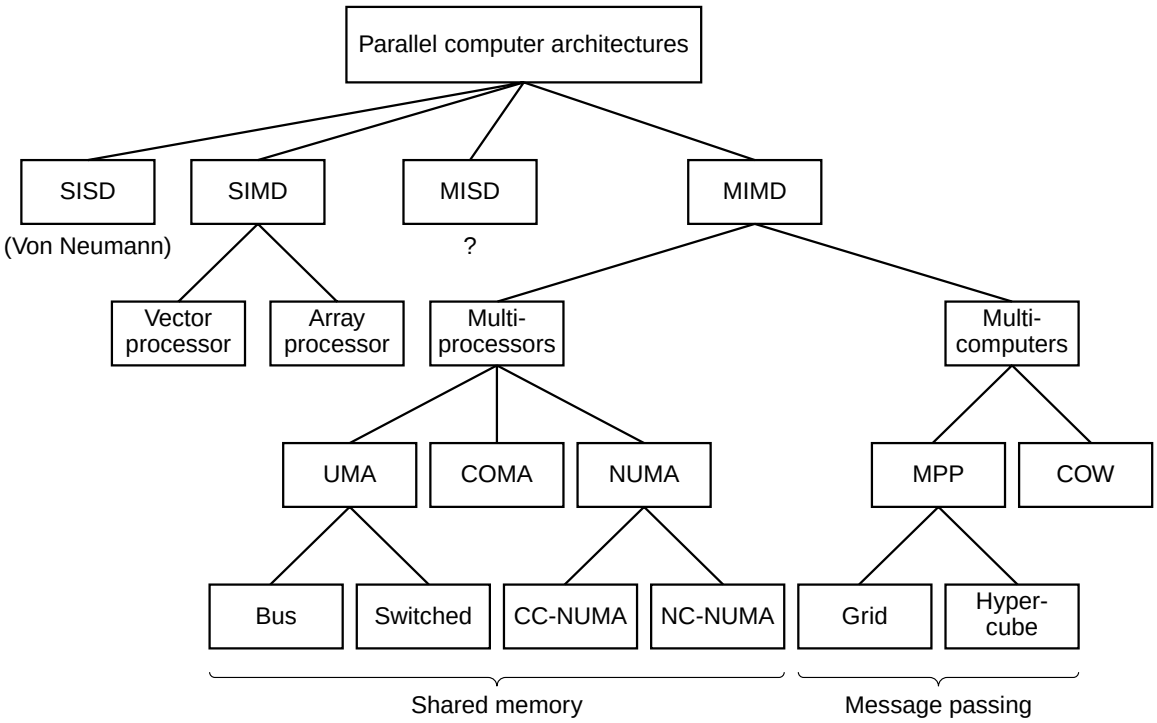

Figure 8-14. A taxonomy of parallel computers.

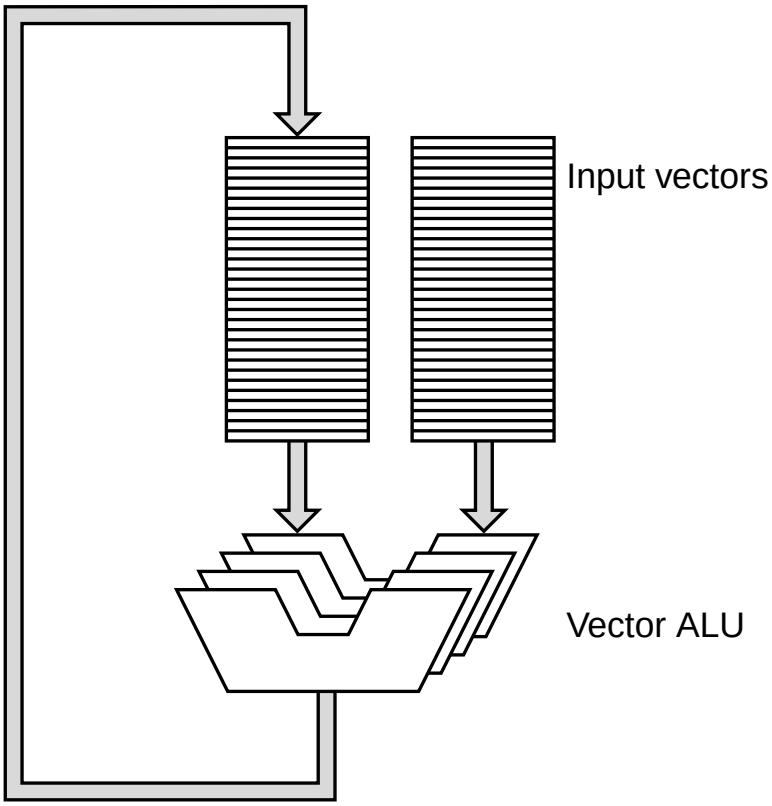

Figure 8-15. A vector ALU.

Operation	Examples
$A_i = f_1(B_i)$	f_1 = cosine, square root
Scalar = $f_2(A)$	f_2 = sum, minimum
$A_i = f_3(B_i, C_i)$	f_3 = add, subtract
$A_i = f_4(\text{scalar}, B_i)$	f_4 = multiply B_i by a constant

Figure 8-16. Various combinations of vector and scalar operations.

Step	Name	Values
1	Fetch operands	$1.082 \times 10^{12} - 9.212 \times 10^{11}$
2	Adjust exponent	$1.082 \times 10^{12} - 0.9212 \times 10^{12}$
3	Execute subtraction	0.1608×10^{12}
4	Normalize result	1.608×10^{11}

Figure 8-17. Steps in a floating-point subtraction.

	Cycle						
Step	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Fetch operands	B_1, C_1	B_2, C_2	B_3, C_3	B_4, C_4	B_5, C_5	B_6, C_6	B_7, C_7
Adjust exponent		B_1, C_1	B_2, C_2	B_3, C_3	B_4, C_4	B_5, C_5	B_6, C_6
Execute operation			$B_1 + C_1$	$B_2 + C_2$	$B_3 + C_3$	$B_4 + C_4$	$B_5 + C_5$
Normalize result				$B_1 + C_1$	$B_2 + C_2$	$B_3 + C_3$	$B_4 + C_4$

Figure 8-18. A pipelined floating-point adder.

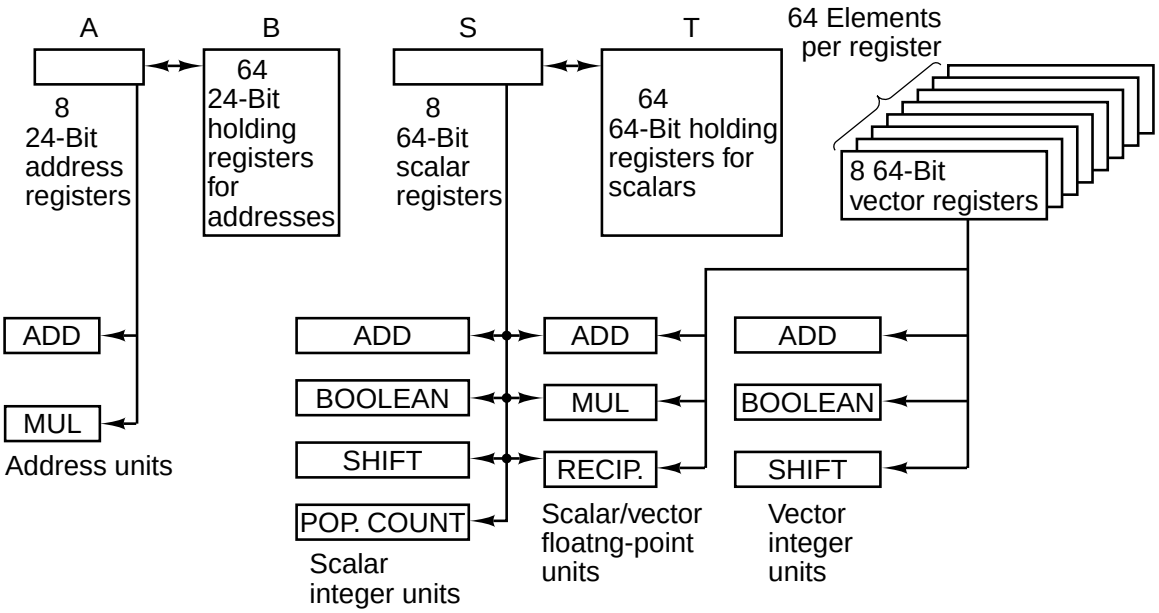

Figure 8-19. Registers and functional units of the Cray-1

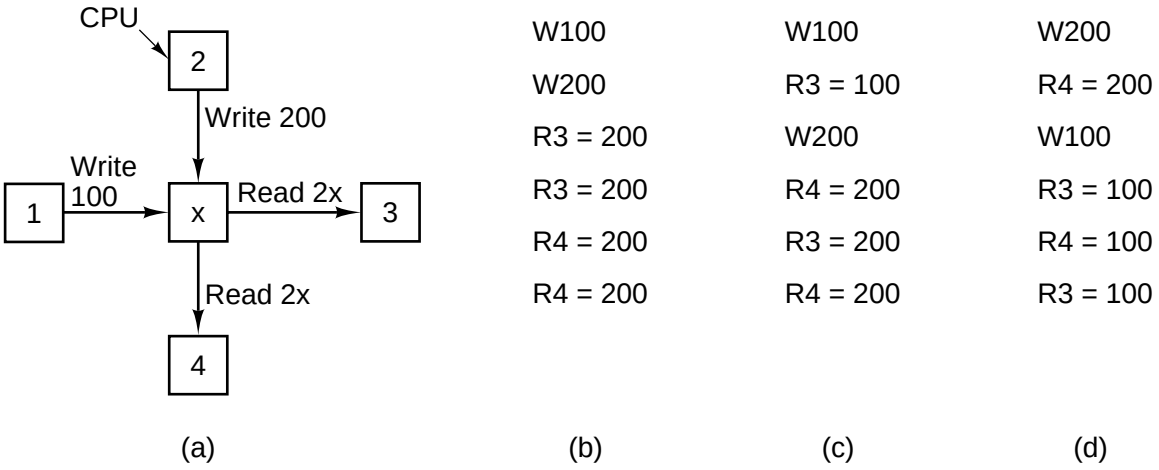

Figure 8-20. (a) Two CPUs writing and two CPUs reading a common memory word. (b) - (d) Three possible ways the two writes and four reads might be interleaved in time.

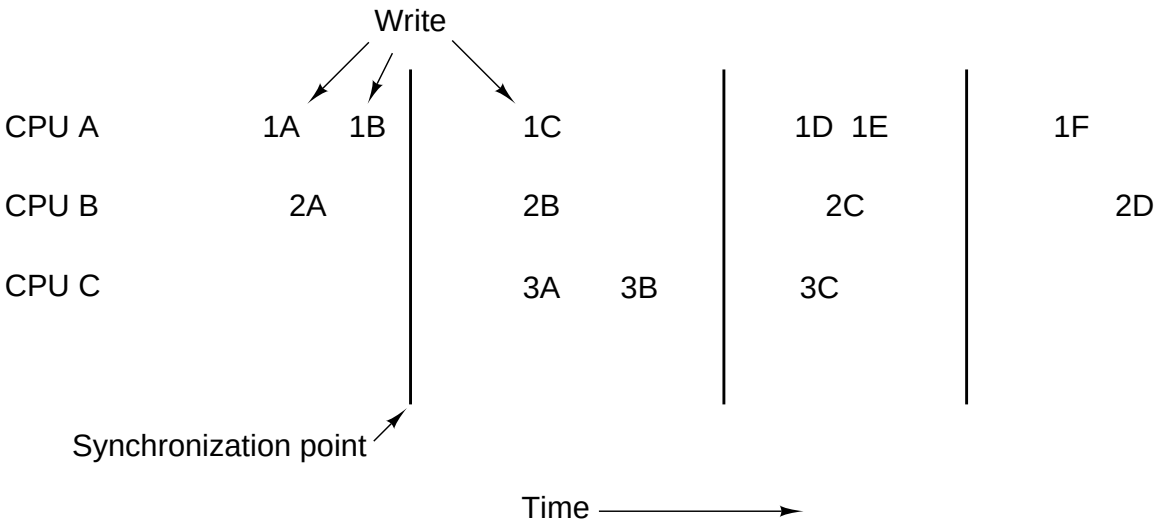

Figure 8-21. Weakly consistent memory uses synchronization operations to divide time into sequential epochs.

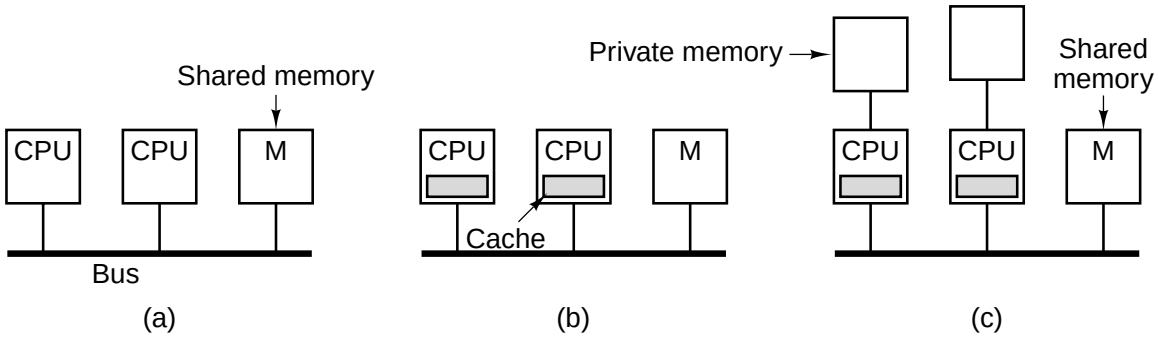

Figure 8-22. Three bus-based multiprocessors. (a) Without caching. (b) With caching. (c) With caching and private memories.

Action	Local request	Remote request
Read miss	Fetch data from memory	
Read hit	Use data from local cache	
Write miss	Update data in memory	
Write hit	Update cache and memory	Invalidate cache entry

Figure 8-23. The write through cache coherence protocol. The empty boxes indicate that no action is taken.

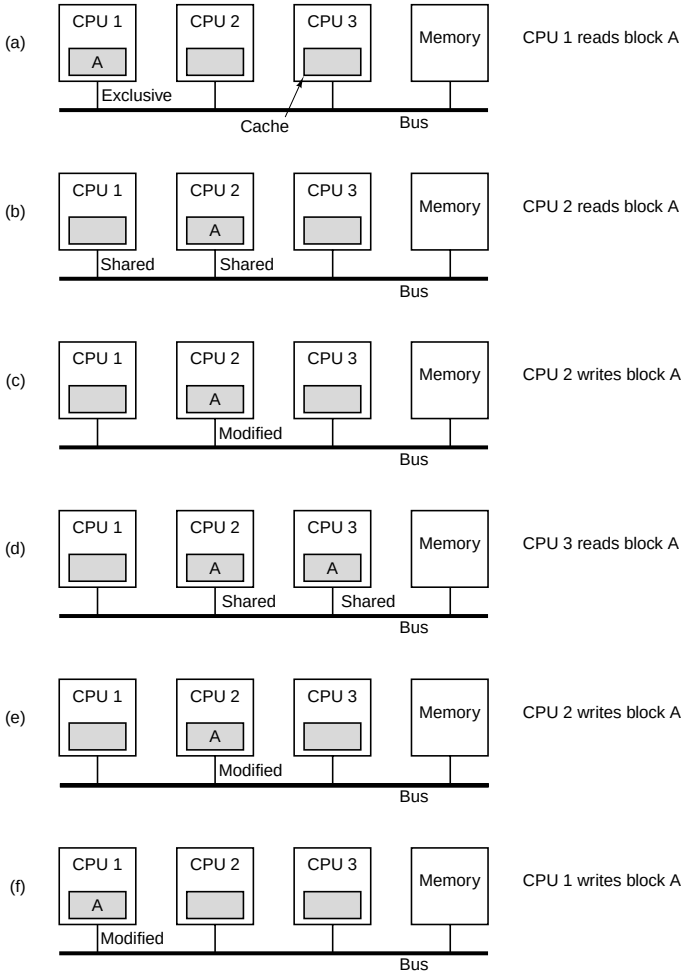

Figure 8-24. The MESI cache coherence protocol.

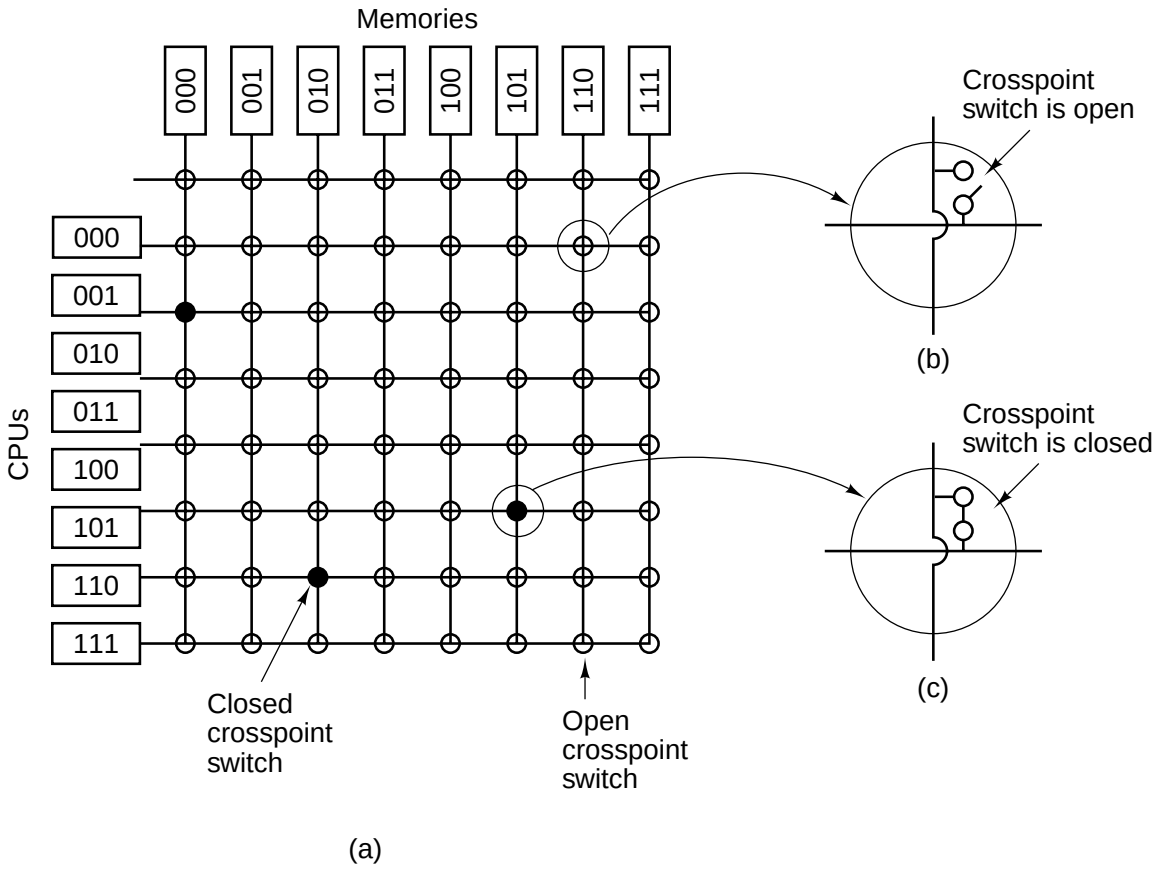

Figure 8-25. (a) An 8×8 crossbar switch. (b) An open crosspoint. (c) A closed crosspoint.

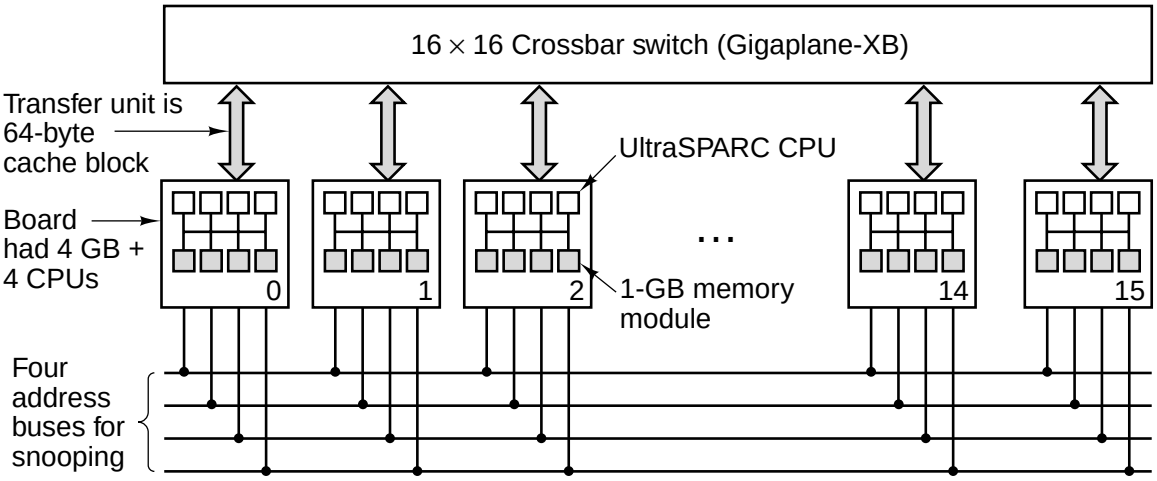

Figure 8-26. The Sun Enterprise 10000 symmetric multiprocessor.

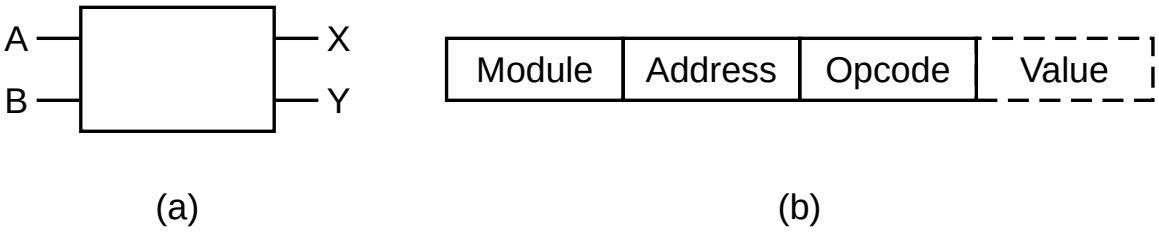

Figure 8-27. (a) A 2×2 switch. (b) A message format.

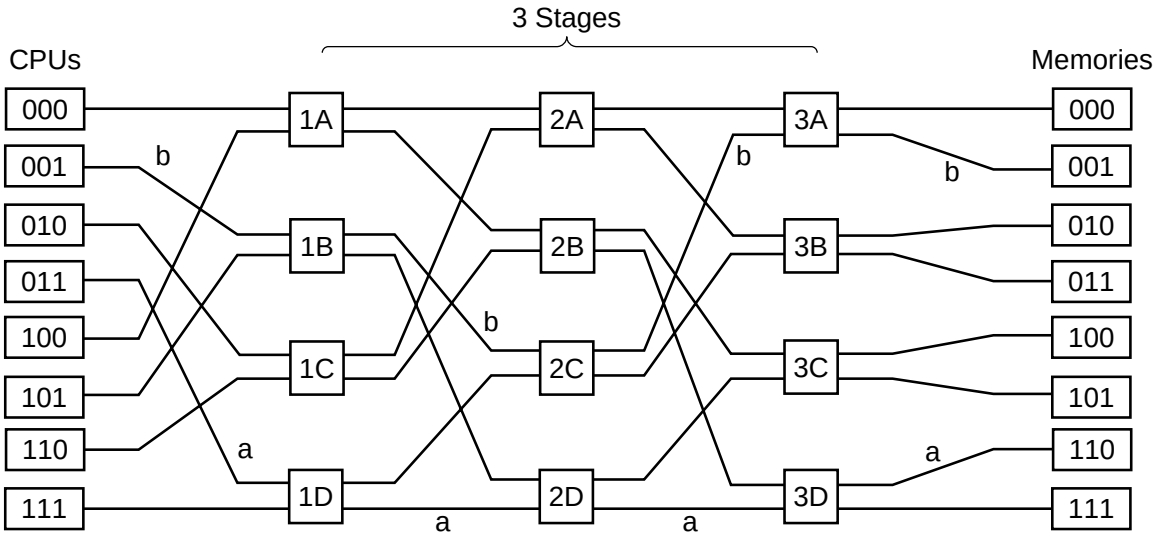

Figure 8-28. An omega switching network.

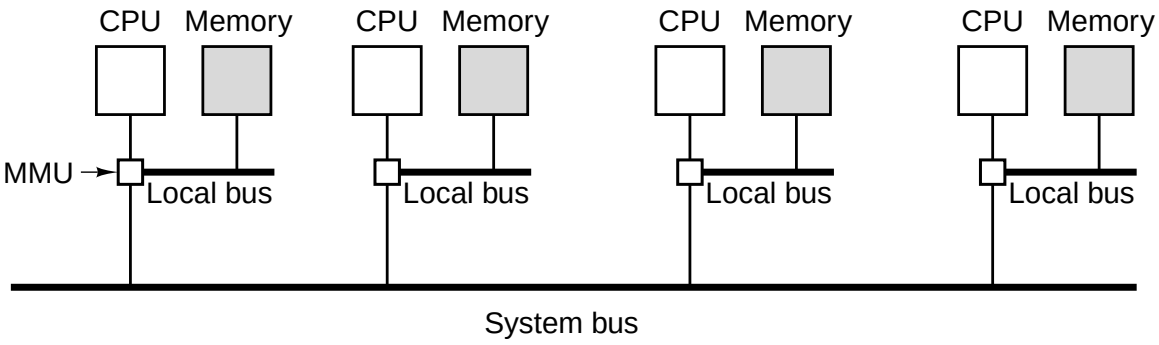

Figure 8-29. A NUMA machine based on two levels of buses. The Cm* was the first multiprocessor to use this design.

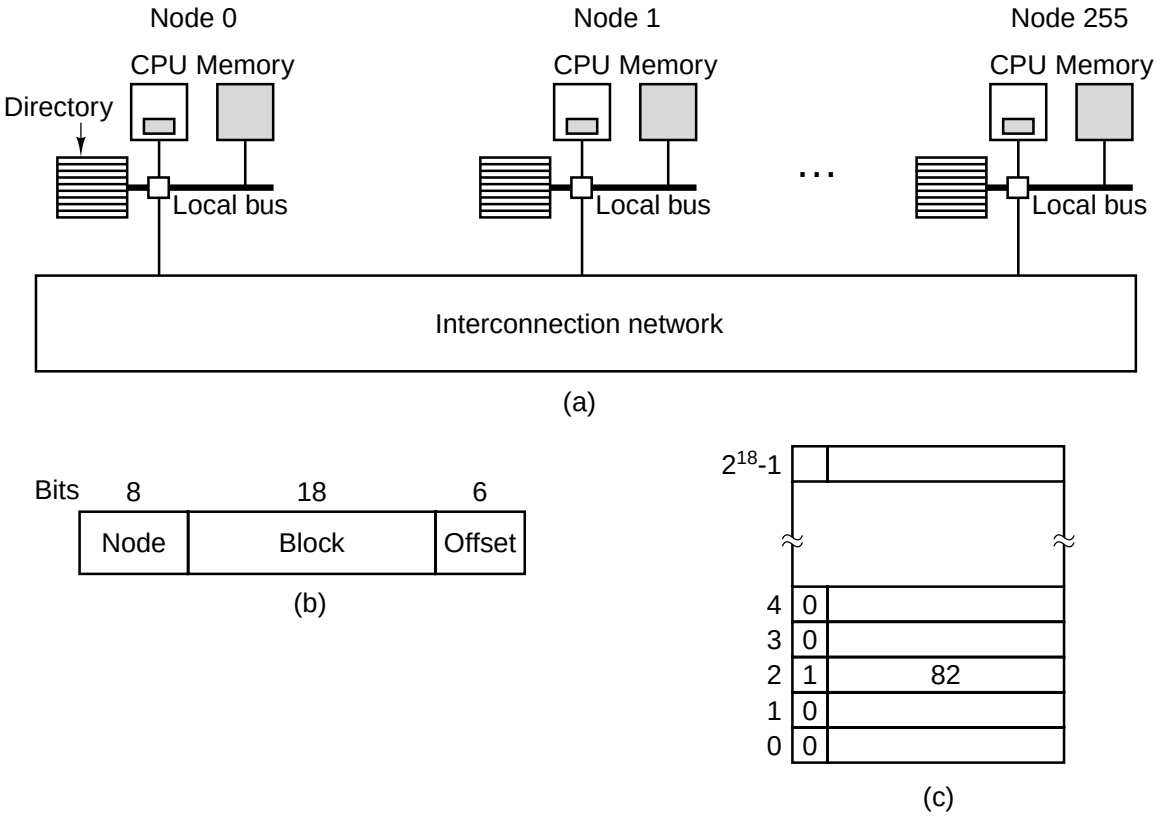

Figure 8-30. (a) A 256-node directory-based multiprocessor. (b) Division of a 32-bit memory address into fields. (c) The directory at node 36.

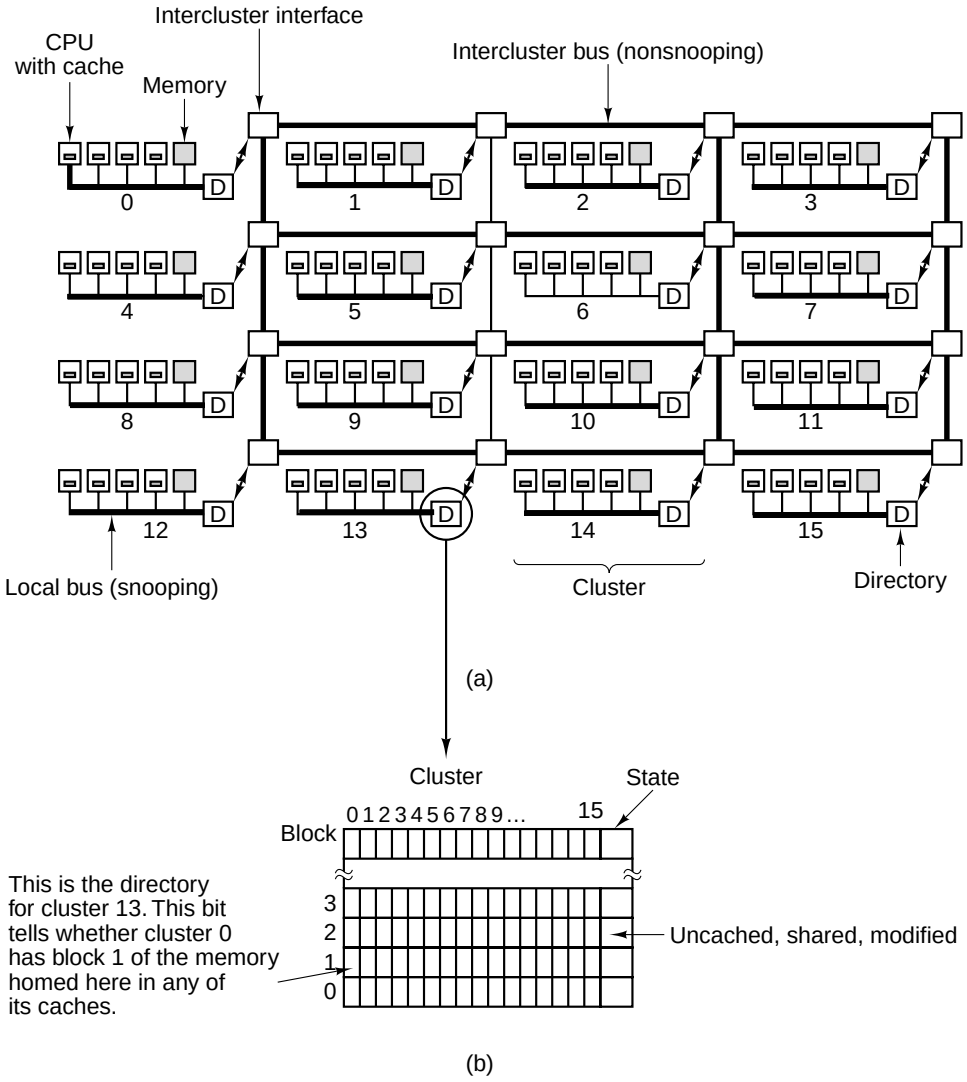

Figure 8-31. (a) The DASH architecture. (b) A DASH directory.

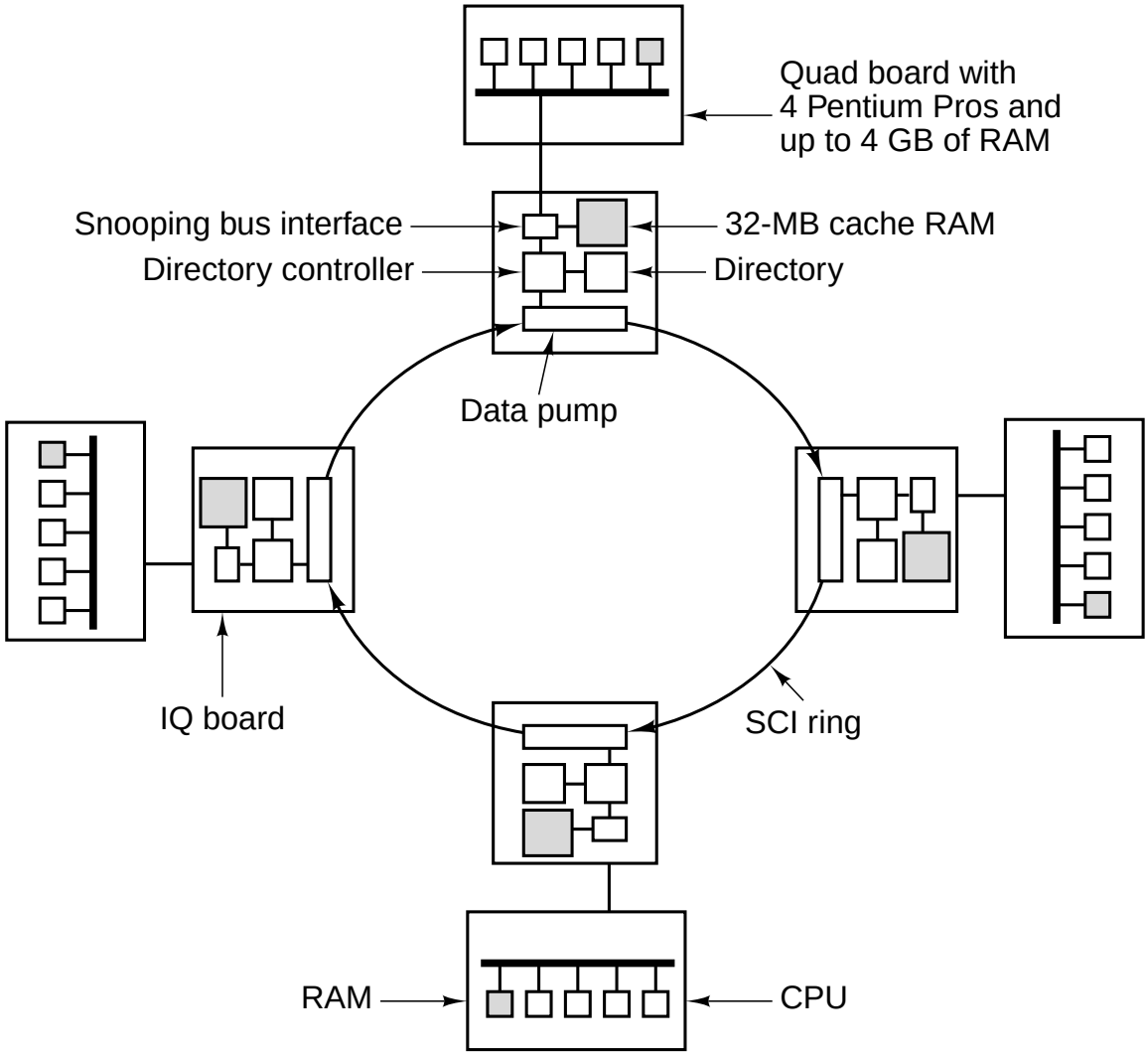

Figure 8-32. The NUMA-Q multiprocessor.

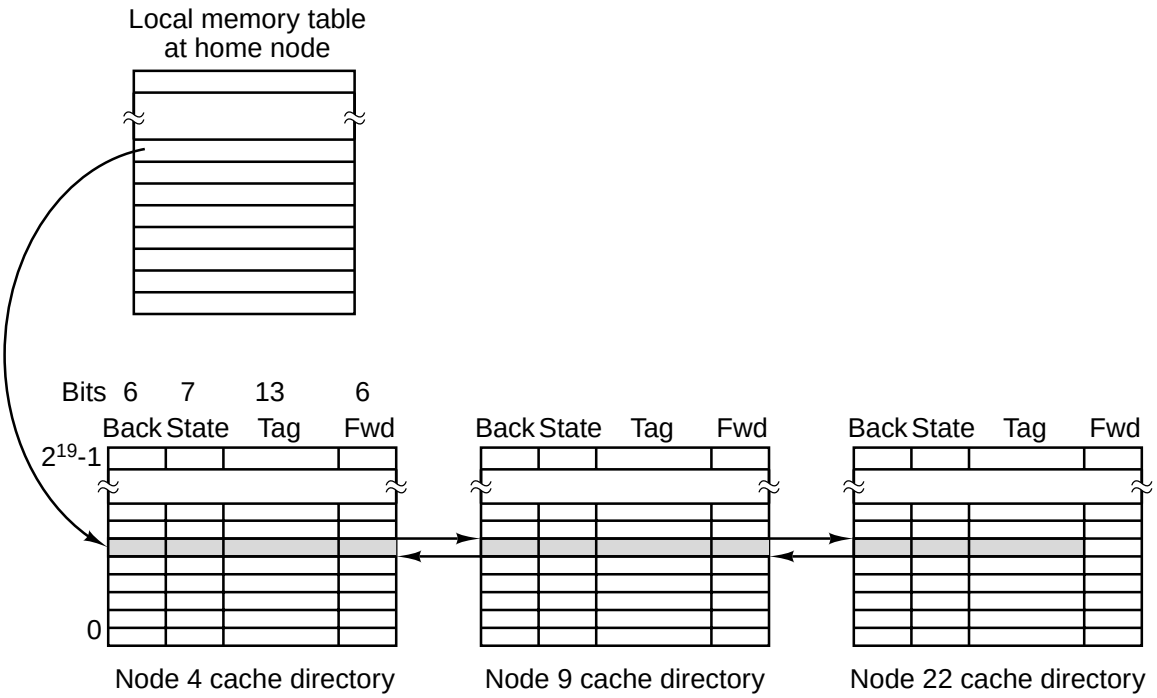

Figure 8-33. SCI chains all the holders of a given cache line together in a doubly-linked list. In this example, a line is shown cached at three nodes.

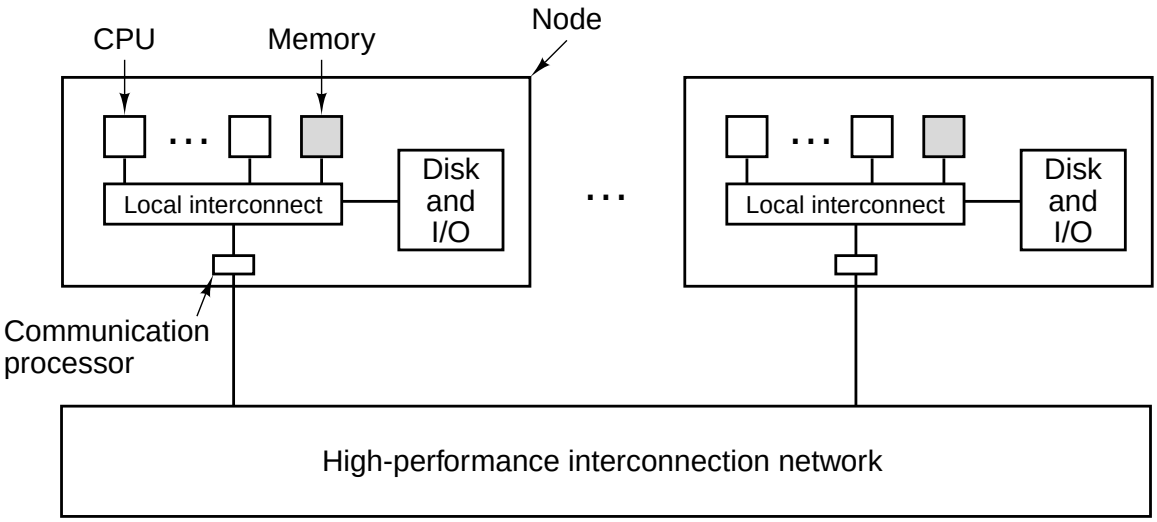

Figure 8-34. A generic multicomputer.

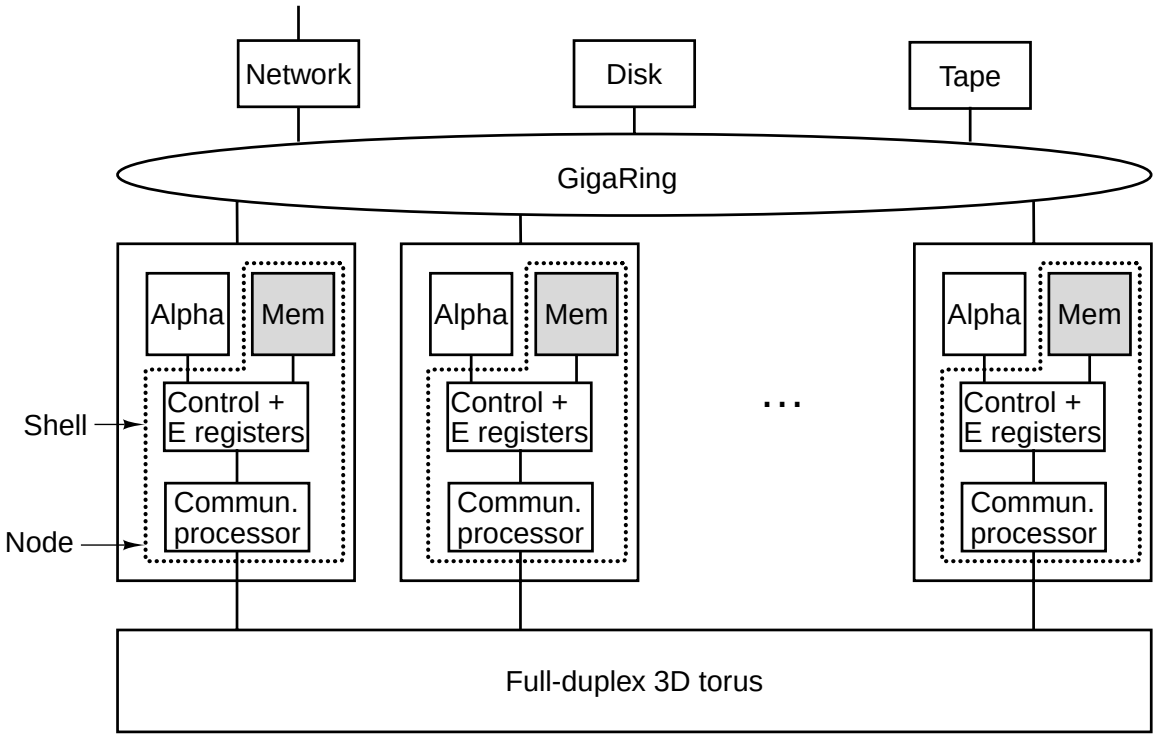

Figure 8-35. The Cray Research T3E.

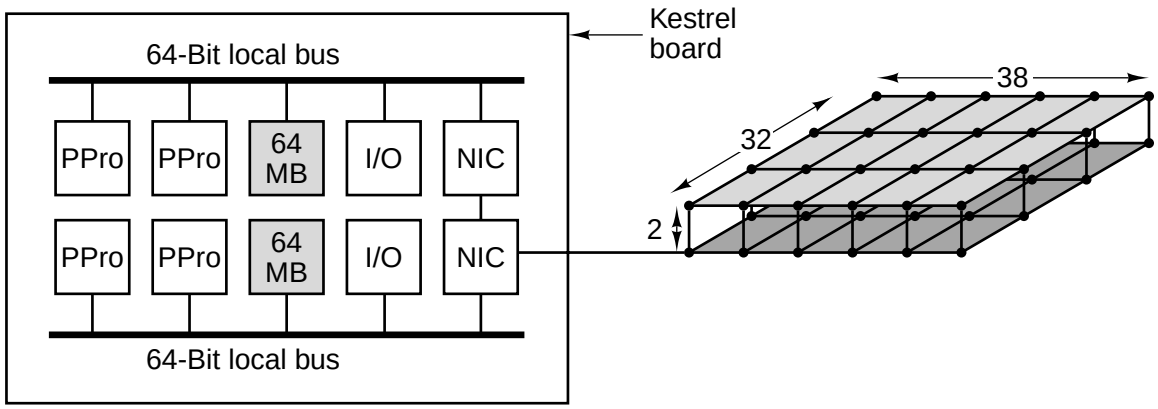

Figure 8-36. The Intel/Sandia Option Red system. (a) The kestrel board. (b) The interconnection network.

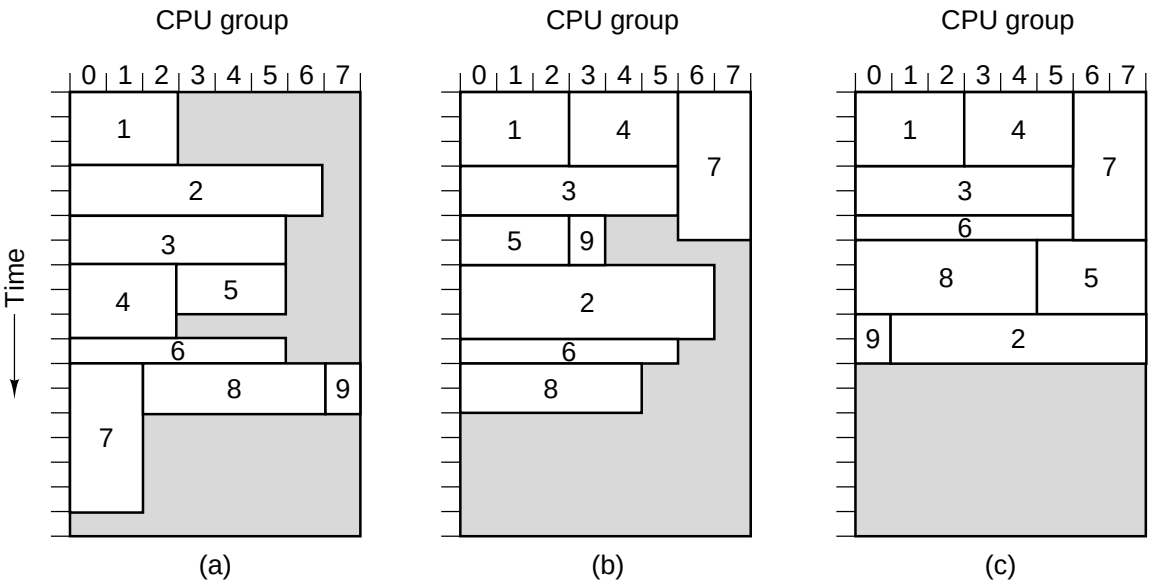

Figure 8-37. Scheduling a COW. (a) FIFO. (b) Without head-of-line blocking. (c) Tiling. The shaded areas indicate idle CPUs.

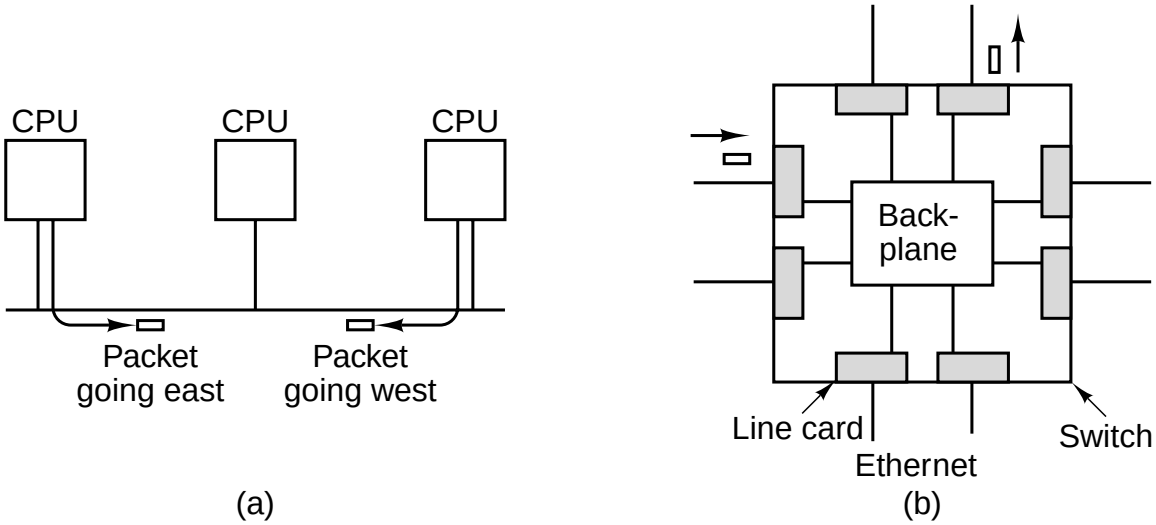

Figure 8-38. (a) Three computers on an Ethernet. (b) An Ethernet switch.

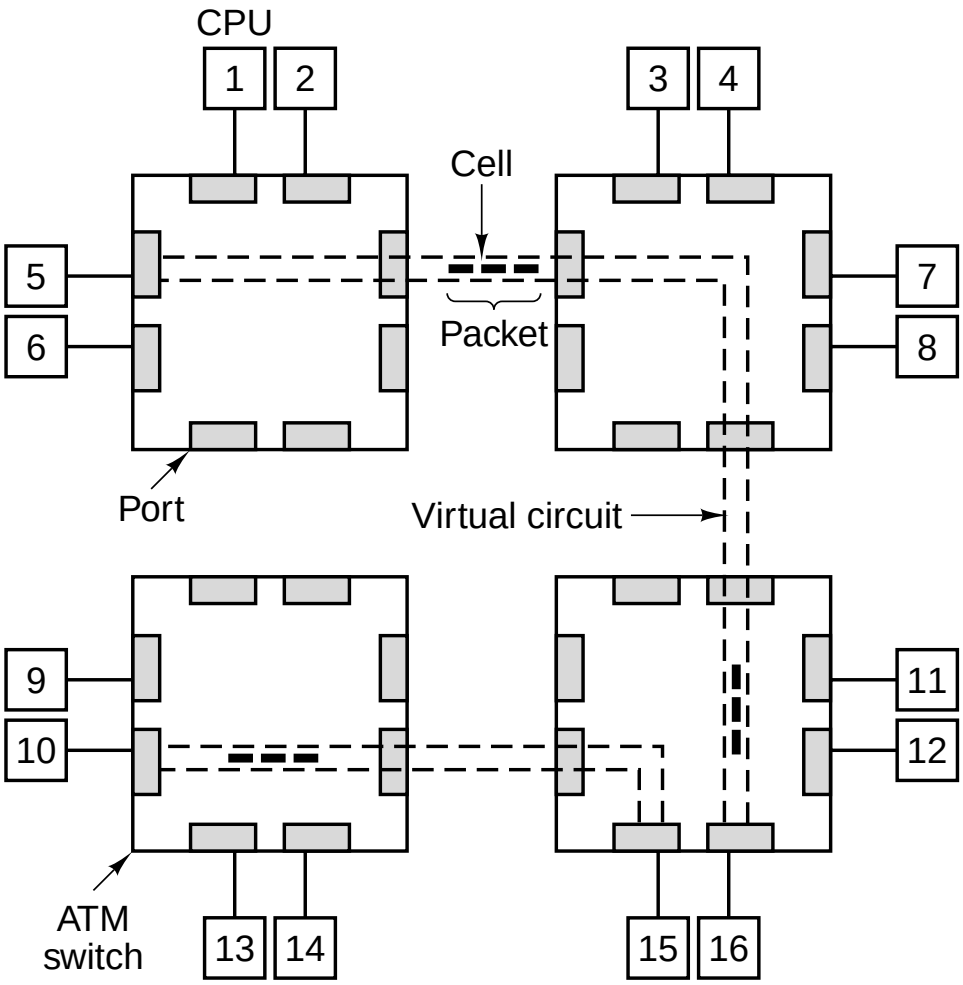

Figure 8-39. Sixteen CPUs connected by four ATM switches. Two virtual circuits are shown.

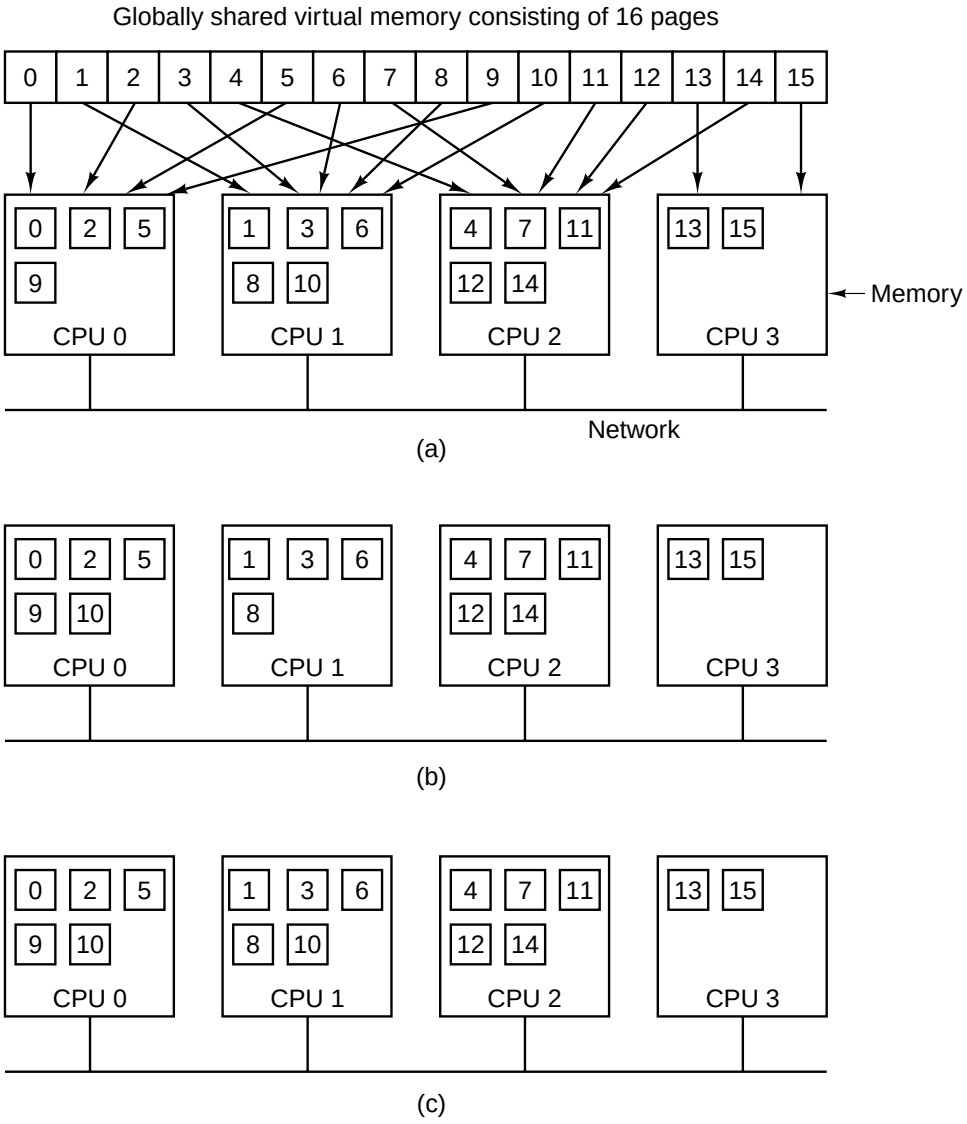

Figure 8-40. A virtual address space consisting of 16 pages spread over four nodes of a multicomputer. (a) The initial situation. (b) After CPU 0 references page 10. (c) After CPU 1 references page 10, here assumed to be a read-only page.

(″abc″, 2, 5)
(″matrix-1″, 1, 6, 3.14)
(″family″, ″is sister″, Carolyn, Elinor)

Figure 8-41. Three Linda tuples.

```
Object implementation stack;
   top:integer;                    # storage for the stack
   stack: array [integer 0..N-1] of integer;

   operation push(item: integer);  function returning nothing
   begin
      stack[top] := item;          push item onto the stack
      top := top + 1;              # increment the stack pointer
   end;

   operation pop( ): integer; # function returning an integer
   begin
      guard top > 0 do             # suspend if the stack is empty
         top := top - 1;           # decrement the stack pointer
         return stack[top];        # return the top item
      od;
   end;

begin
   top := 0;                       # initialization
end;
```

Figure 8-42. A simplified ORCA stack object, with internal data and two operations.