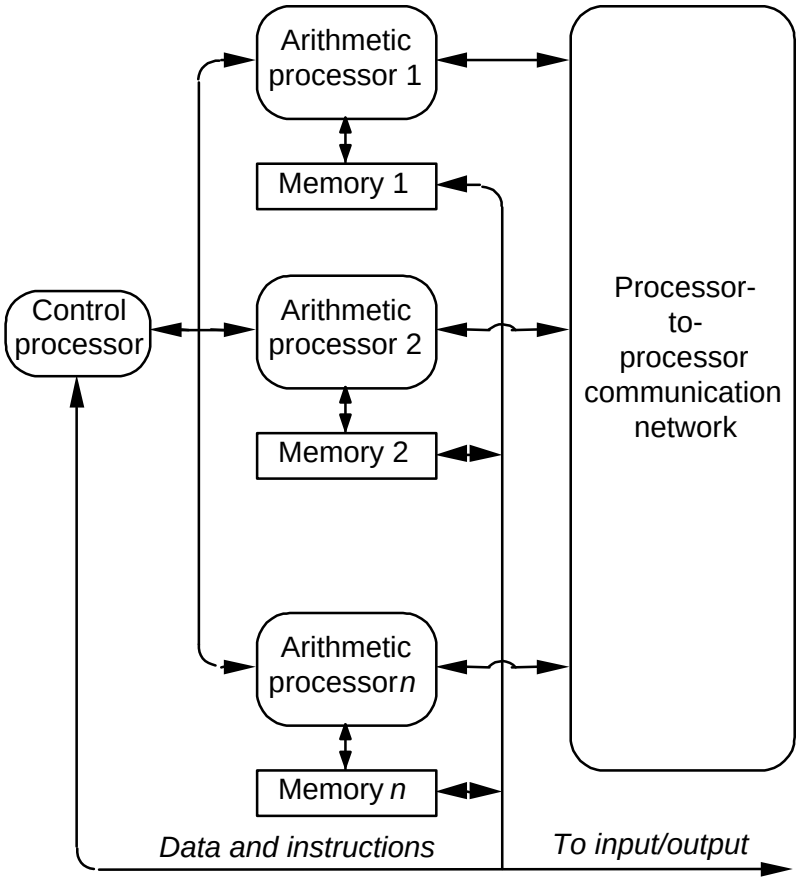
Multiprocessor Systems

A ***Multiprocessor*** system generally means ***that more than one instruction stream*** is being executed in parallel.

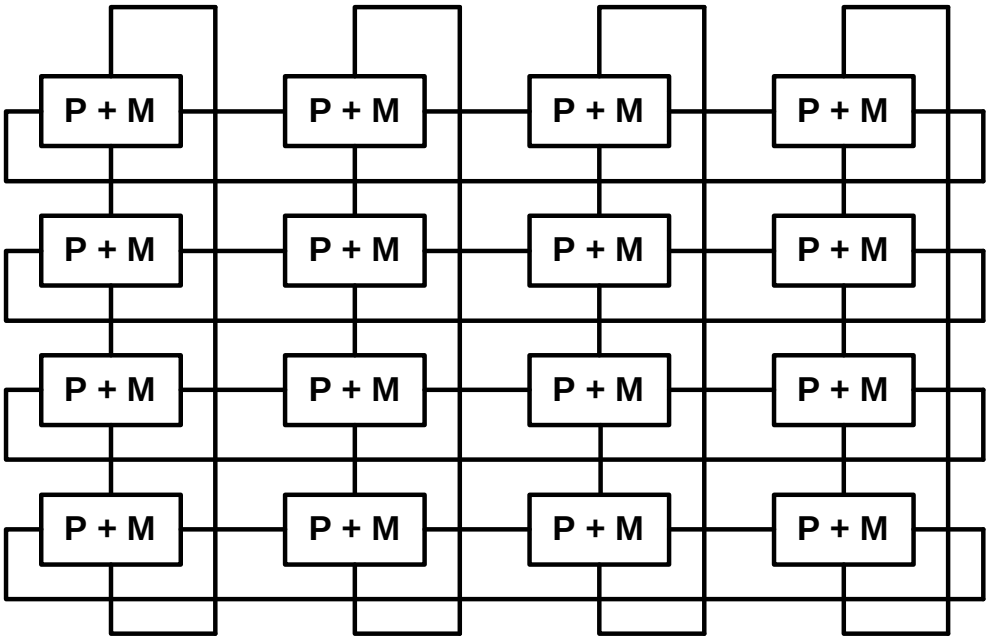
However, Flynn's ***SIMD*** machine classification, also called an ***array processor***, is a multiprocessor but uses only a ***single instruction stream*** operating against ***multiple execution units*** in parallel. This architecture is of ***academic interest*** (only). No commercially successful array processors have ever been built.

An example array processor:



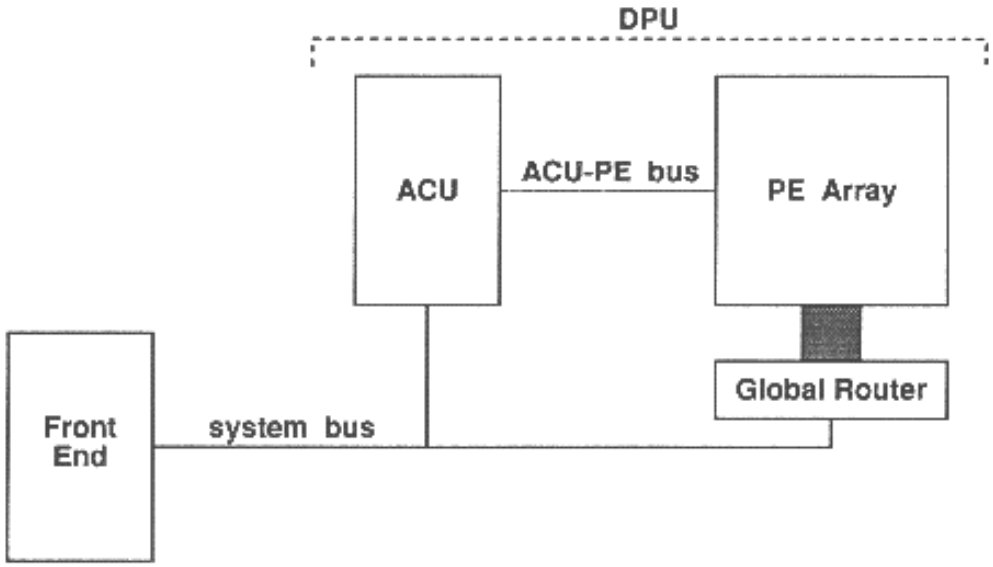
The single instruction stream goes to the control processor, which executes control instructions such as compare and branch sequentially, but also broadcasts instructions to the arithmetic processors to be performed on their local chunk of data in parallel.

Processors may be connected to four neighbors, eight neighbors, or more.



The MasPar MP-2/MP-1 Cluster – MasPar Computer Corporation

32K processor elements
MP-2 = 6.2 gigaFLOPS peak



MasPar front-end

Since the computational engine of the MasPar does not have an operating system of it's own, a UNIX based workstation is used to provide the programmer with a "friendly" interface to the MasPar.

DPU (Data Parallel Unit)

The DPU executes the parallel portions of a program and consists of two parts:

- ACU (Array Control Unit)
- PE Array (Processor Element Array)

The ACU has two tasks:

- Execute instructions that operate on singular data.
- Simultaneously feed instructions which operate on parallel data (known as "plural" data in MPL) to each PE (Processor Element).

The game of LIFE on a MasPar MP-2/MP-1

```
plural int now;     /* Define integer "now" on each processor  */

main()
{
    register plural int next, sums;
    register int config, maxloop,count;
    extern plural int init;

    for (config = 0; config < 5; config++ ) {
     now = next = sums = 0;
     init_life(config);    /* Initialize a Life configuration  */

   /* The plural variable "now == 1" represents "Live" points  */
     maxloop = 1000;
     count = 0;

     while (reduceOr32(now) != 0 ) {   /* Only if life exists  */
         init = now;
         sums =  xnetNE[1].now + xnetE[1].now
               + xnetSE[1].now + xnetS[1].now
               + xnetSW[1].now + xnetW[1].now
               + xnetNW[1].now + xnetN[1].now;

         /* Determine if a cell lives or dies
            according to the game of life.  */
         if (now == 1 ) {
          if ((sums > 3) || (sums < 2))
              next = 0;
          else
              next = 1;
         }
         if (!now ) {
          if ( sums == 3 )
              next = 1;
          else
              next = 0;
         }
         now = next;    /*  Update all cells to the new state  */
         count++;
         if (count >= maxloop )
          break;
     }
     printf("done with life configuration: %d; count: %d\n",
config, count);
    }
    printf("done with all configurations\n");
}
```

Commercially Successful SIMD Architecture

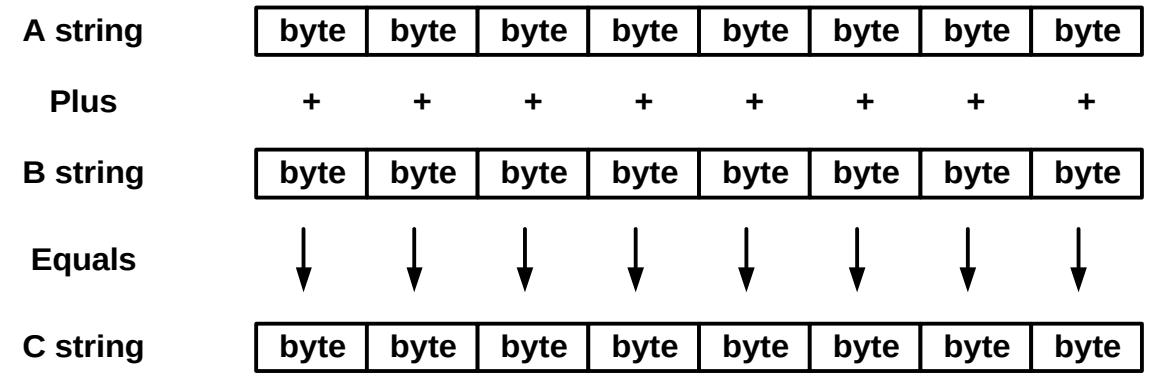
A variant of the SIMD architecture has been popularized by Intel as ***Multimedia Extensions*** (MMX), and now ***Streaming SIMD Extensions*** (SSE). MMX instructions operate on fixed-point data and SSE instructions operate on floating-point data.

MMX

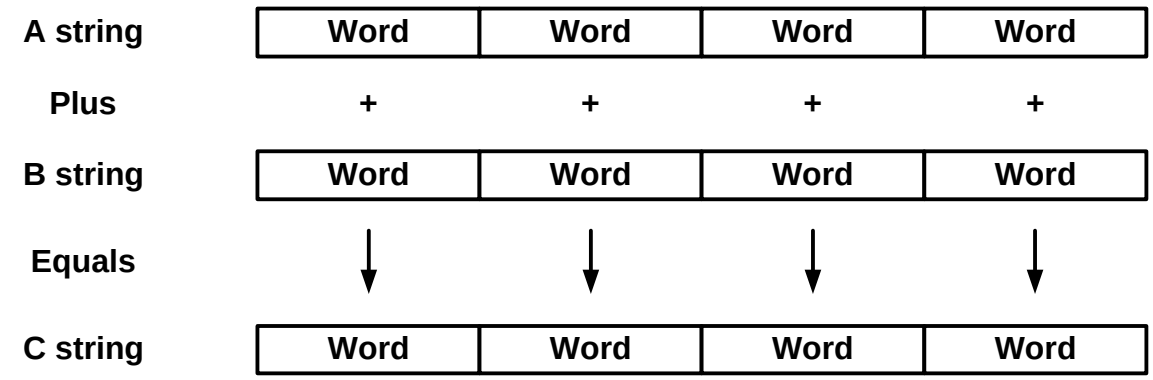
MMX Technology added new instructions to the x86 architecture that operate on multiple chunks of fixed-point data in parallel. Operations on these multiple data chunks are Add, Subtract, Multiply, Multiply and Add, Compare, And, And Not, Or, Exclusive Or, and Shift.

The data chunks are all 64 bits wide, and can be broken into bytes, words, and double words, where a word is defined as 16 bits.

Byte Operation



Word (16-bit) Operation



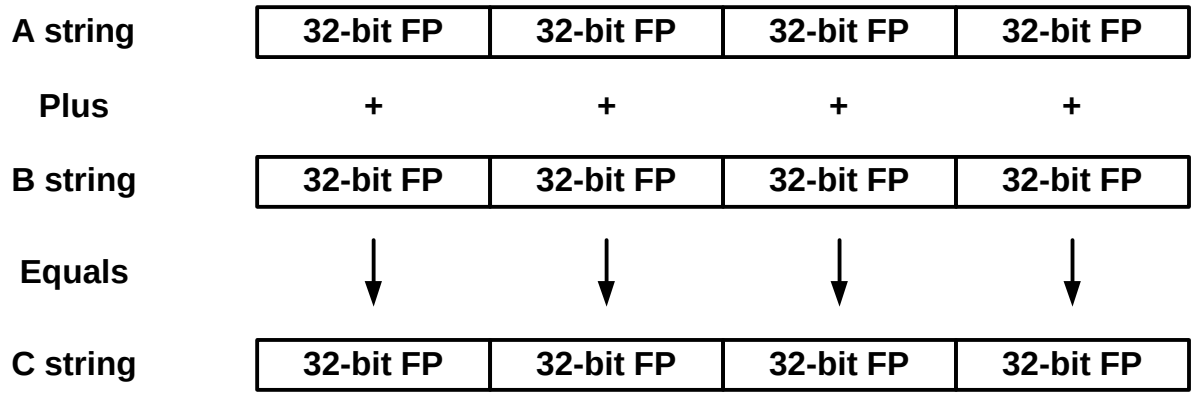
SSE

Streaming SIMD Extensions added new instructions to the x86 architecture that operate on multiple chunks of ***floating-point*** data in parallel. The SSE set consists of 70 instructions, grouped into the following categories:

- Data movement instructions
- Arithmetic instructions
- Comparison instructions
- Conversion instructions
- Logical instructions
- Additional SIMD integer instructions
- Shuffle instructions
- State management instructions
- Cacheability control instructions

The data chunks are all 128 bits wide, and can be viewed as four 32-bit packed floating-point numbers.

Single-precision Floating-point Operation



Cache Management

Instructions added for the Streaming SIMD Extensions allow the programmer to prefetch data before its actual use and also to load and store data bypassing the cache.

The PREFETCH (Load 32 or greater number of bytes) instructions load either non-temporal data or temporal data in the specified cache level.

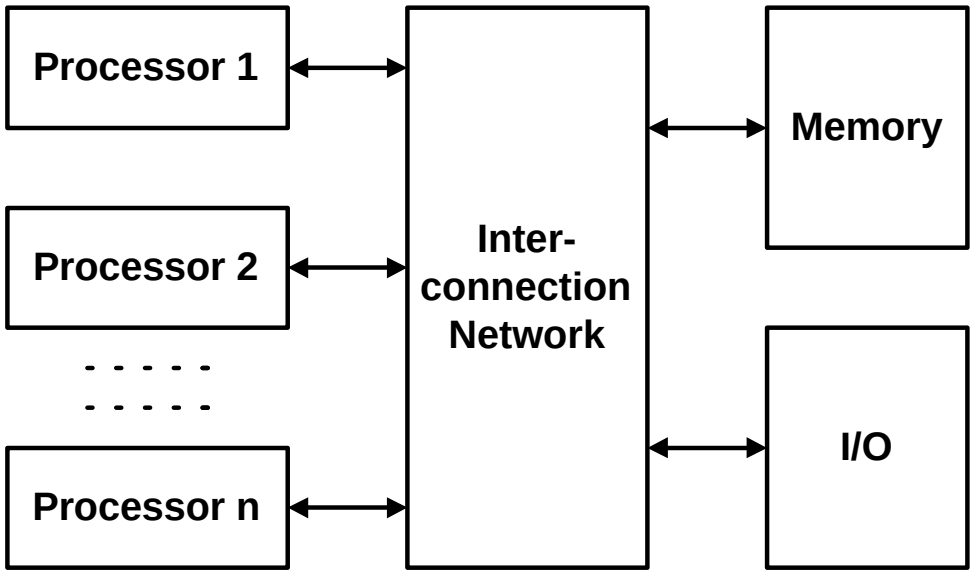
The following three instructions provide programmatic control for minimizing cache pollution when writing data to memory from either the MMX™ registers or the SIMD floating-point registers.

- The MASKMOVQ (Non-temporal byte mask store of packed integer in an MMX™ register) instruction stores data from an MMX™ register to the location specified by the (DS) EDI register. The instruction does not write-allocate (i.e., the processor will not fetch the corresponding cache line into the cache hierarchy, prior to performing the store).
- The MOVNTQ (Non-temporal store of packed integer in an MMX™ register) instruction stores data from an MMX™ register to memory. The instruction does not write-allocate, and minimizes cache pollution.

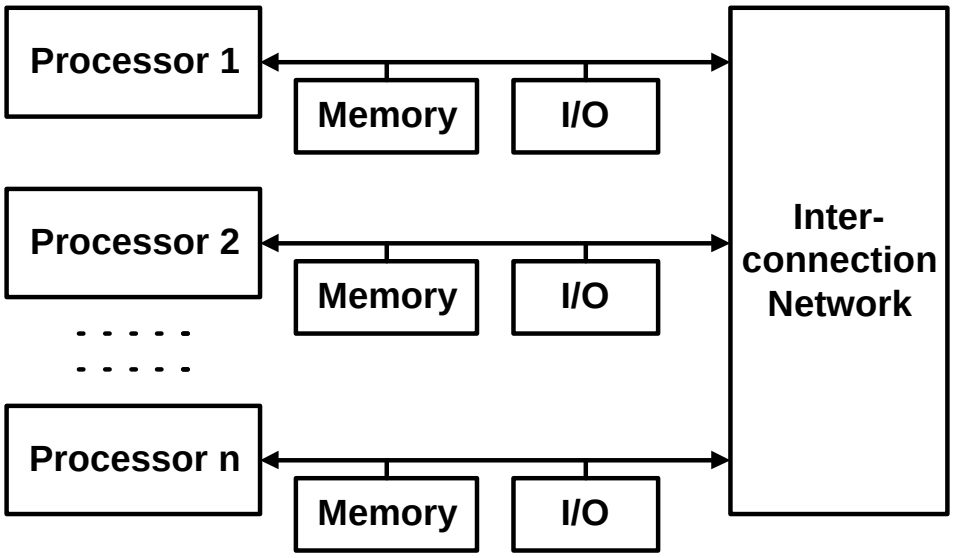
The MOVNTPS (Non-temporal store of packed, single-precision, floating-point) instruction stores data from a SIMD floating-point register to memory. The memory address must be aligned to a 16-byte boundary; if it is not aligned, a general protection exception will occur. The instruction does not write-allocate, and minimizes cache pollution.

Flynn's ***MIMD*** classification: multiple instruction streams, each operating against its own data stream. This is the ***predominant type*** of multiprocessor, and two major types exist:

Uniform Memory Access (UMA), also ***tightly coupled*** or ***symmetric multiprocessing*** (SMP) or ***shared-memory multiprocessing*** (SMP).



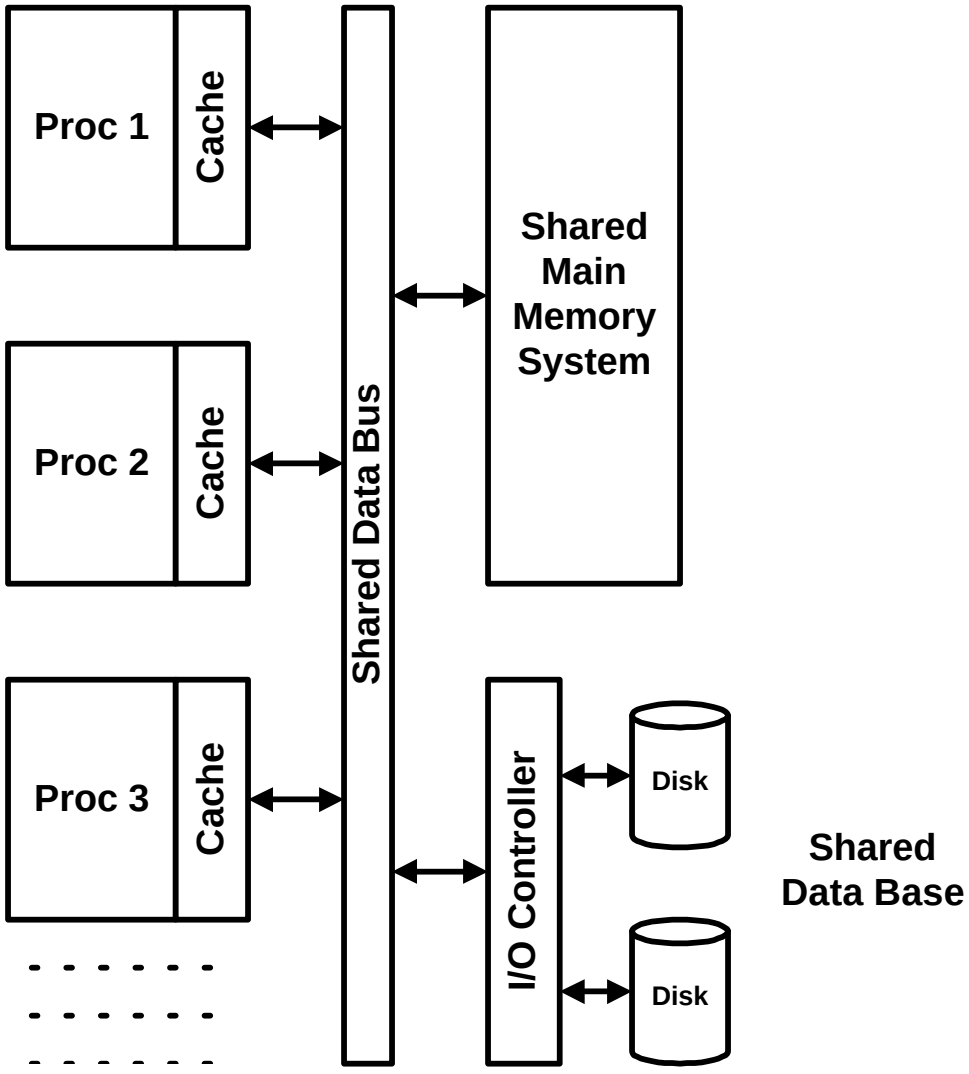
Non-uniform Memory Access (NUMA), also ***loosely-coupled*** or ***clustering***.



SMP Systems

When you hear the word ***multiprocessing***, it is generally referring to a ***Symmetric Multiprocessing*** (SMP) configuration.

With an SMP system, we have ***multiple processors***, each with has its own cache, but a ***single main memory system*** and I/O subsystem. In this configuration, we need to ***minimize the need to use the shared bus*** to access the shared memory system. The shared bus/memory system becomes the ***bottleneck*** that prevents scaling the SMP system up higher than 8 to 10 processors.

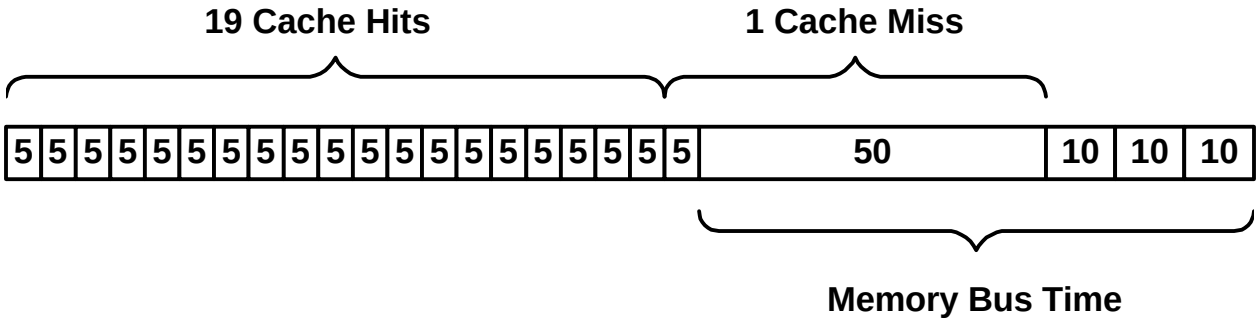


Memory Bus Contention

Consider a ***single processor*** running at 200 MHz with a CPI of 1, an average of 1 memory accesses per instruction, and a write-back cache with access time of 1 (processor) cycle. The cache line is 4 words of 64 bits each, hit ratio is 95%, the main memory is SDRAM clocked at 100 MHz, with a RAS access time is 5 clocks. Recall that the processor can proceed on a cache miss immediately after the missed word is loaded.

- What is the average utilization of the main memory bus and the memory system due to this single processor?

200 MHz / 1 CPI x 1 access/inst. = 200M attempted memory accesses per second from the processor. But with a hit rate of 0.95, we take a cache miss on the average of every 20 instructions. So we have:



Each set of 20 instructions takes a total of 150 ns, and 80 ns of that time we are using the main memory bus. So the ***memory/bus utilization is 53%.***

- Clearly, a ***dual-processor SMP system*** (with each processor operating under the same conditions) will ***saturate the bus/memory system***.

Other Considerations

- What would be the bus utilization if we used a write-through cache and we assume the processor issues a write an average of once every 10 instructions?
- What would be the bus utilization if we improved the cache hit rate to 98%?
- What would be the bus utilization if we improved the cache hit rate to 99%?
- How do we deal with multiple processors reading and writing shared data, and each possibly having a copy of the shared data in its private cache?

Other than improving the cache hit rate, how can we reduce the shared memory contention, and thus increase the number of processors we can effectively use in an SMP configuration?

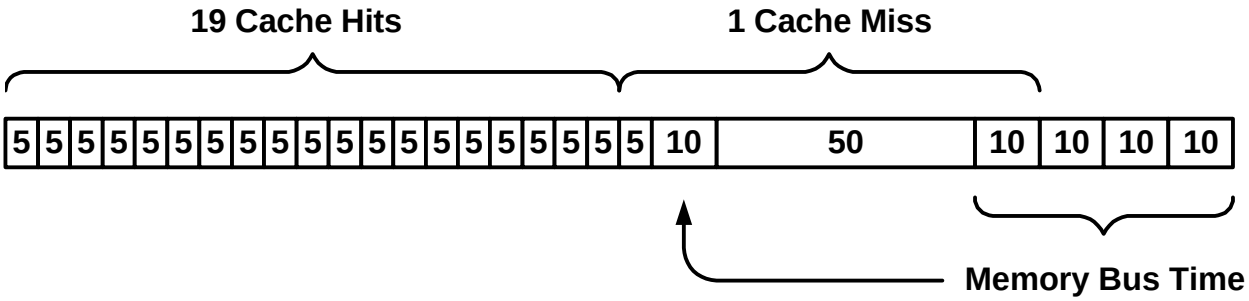
We are assuming that the shared bus is being used for the ***entire memory access time*** as well as the actual data transfer time, and that we have a monolithic memory system.

- We could change the bus into a ***transaction bus***.
- Use an ***interleaved memory*** system.

Transaction Bus

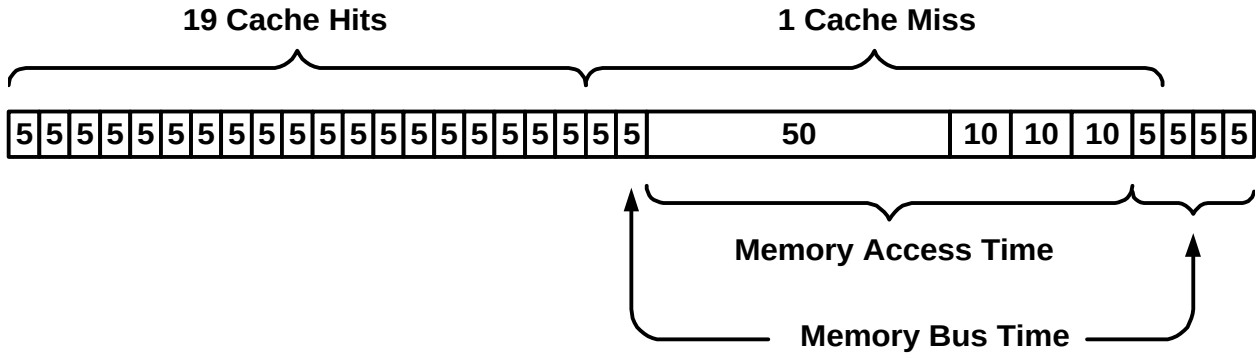
With a transaction bus, we can send a request for data over the bus in one cycle time, ***release the bus for use by other processes***, and then reattach to the bus and transfer the requested data ***when it is available***.

Using a bus clock rate of 100 MHz, and adding one bus clock time to send the request to the memory system, we get the following picture:



Assuming that we have contention for the bus only, and not for the memory, our bus utilization for the 95% hit rate is now 50 ns out of 170 ns for a ***utilization of 29%*** instead of the 53% where we held onto the bus for the entire memory access. Notice, however, that we also lengthened the memory access time by 20 ns on a miss, for a total of 75 ns instead of 55.

We may further improve this by speeding up the bus clock and transferring the data in a burst after the memory has it available after 80 ns.

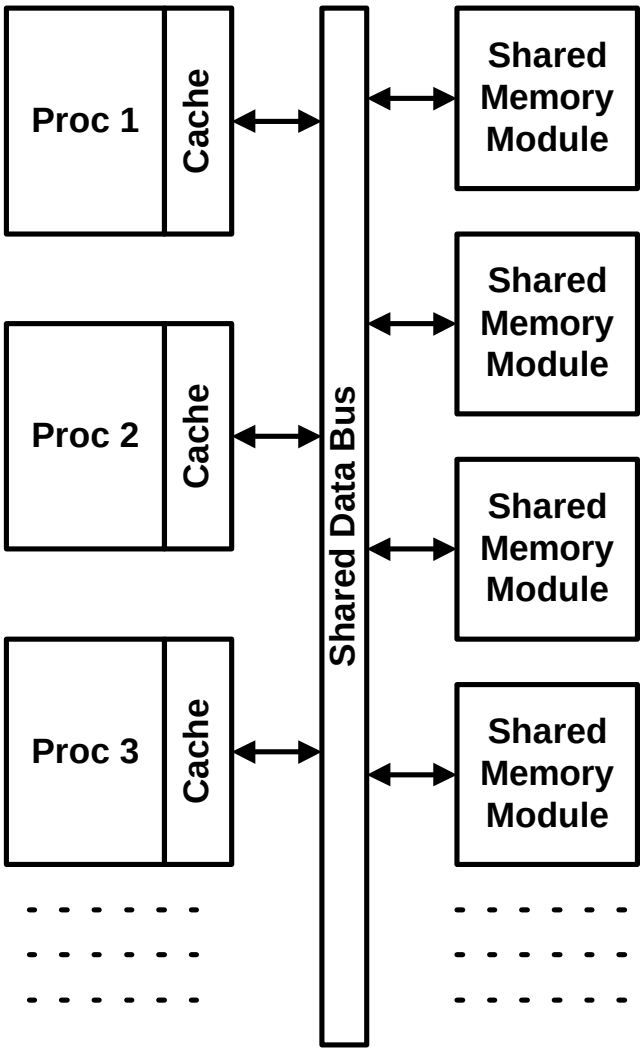


This drops our utilization to 25 ns out of 190 for ***13%***. But we again lengthened our cache miss time – now to 95 ns.

Interleaved Memory Again

The preceding discussion on use of a transaction bus assumed that the point of contention was the path to the memory – the bus – and not the actual memory. In reality, we need to organize the memory so multiple processors can be accessing the shared memory concurrently.

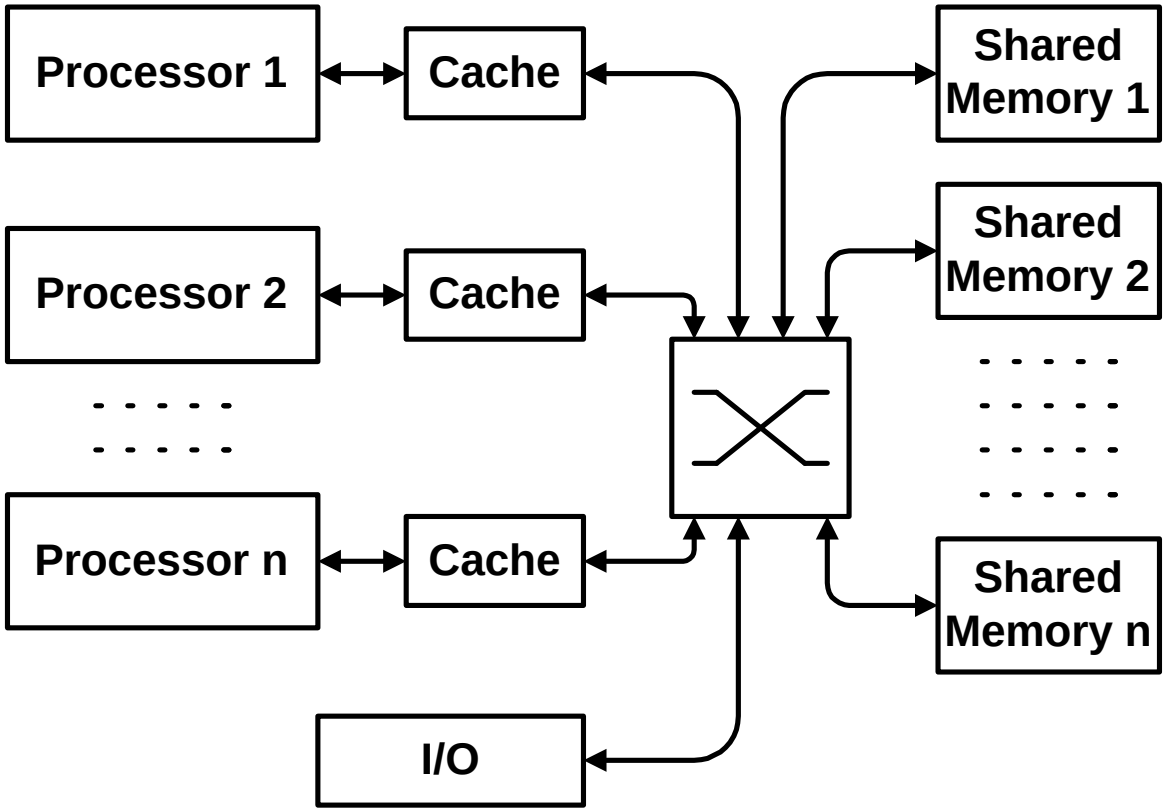
- We do this with interleaved memory



We need to be concerned with contention for the bus ***and*** contention for the memory module.

Switch Connection

One way to ***totally eliminate contention*** for the path to the memory is to use a non-blocking switch instead of a bus to access the memory.

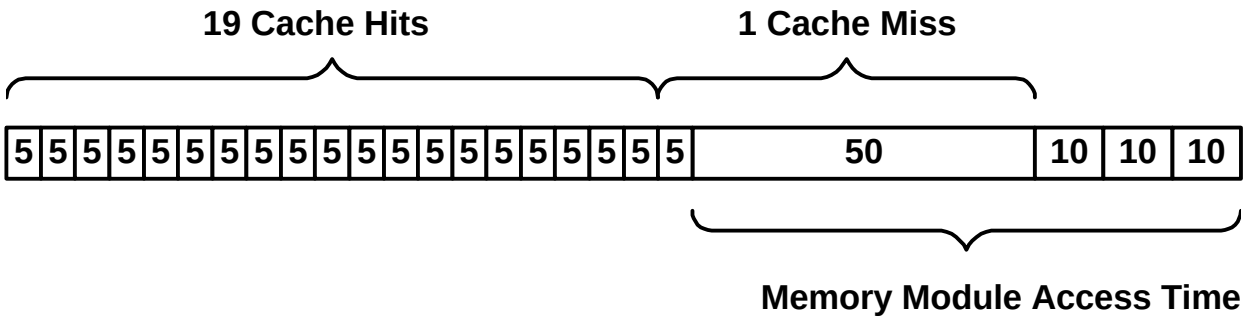


Now we are concerned ***only with contention for the memory modules*** themselves.

- The non-blocking switch can connect any processor to any memory module concurrently with any other processor being connected to any other memory module.

Memory Utilization With a Switch

Given our example processor with a cache hit rate of 95% and a ***suitably-arranged 8-way interleaved memory system***, the utilization of any one of the memory modules would be:



We are back to a cache miss taking only 55 ns, but we are dividing the access on a miss over the eight independent memory modules, so the average module utilization is 1/8 of 80/(100 + 50) = ***7% utilization***.

- Using a switch and interleaved memory makes a ***big difference*** with SMP multiprocessors.

If we assume a cache hit rate of 98% (reasonable), the utilization drops to 1/8 of 80/(250 + 50) = ***3.3% utilization per processor.***

- This is how we can scale up to about 16 processors with an SMP configuration.

However, a big drawback with switched systems is that bus snooping cannot be used to maintain cache coherency. Each cache cannot see addresses that other caches are placing on the bus, and so cannot alert others of dirty data it is holding or invalidate stale cache lines.

Multiprocessing with Clusters

SMP systems have a problem with scalability because of the need to access shared memory and the locking and cache coherency problems that go with it.

- Bus-based systems cannot go beyond 8 to 10 processors (with very large caches) because of the shared bus bottleneck.
- Switch-based systems can go up to 10 to 16 processors with more complex locking and cache coherency methods before they also succumb to communication overhead required to keep everything straight.

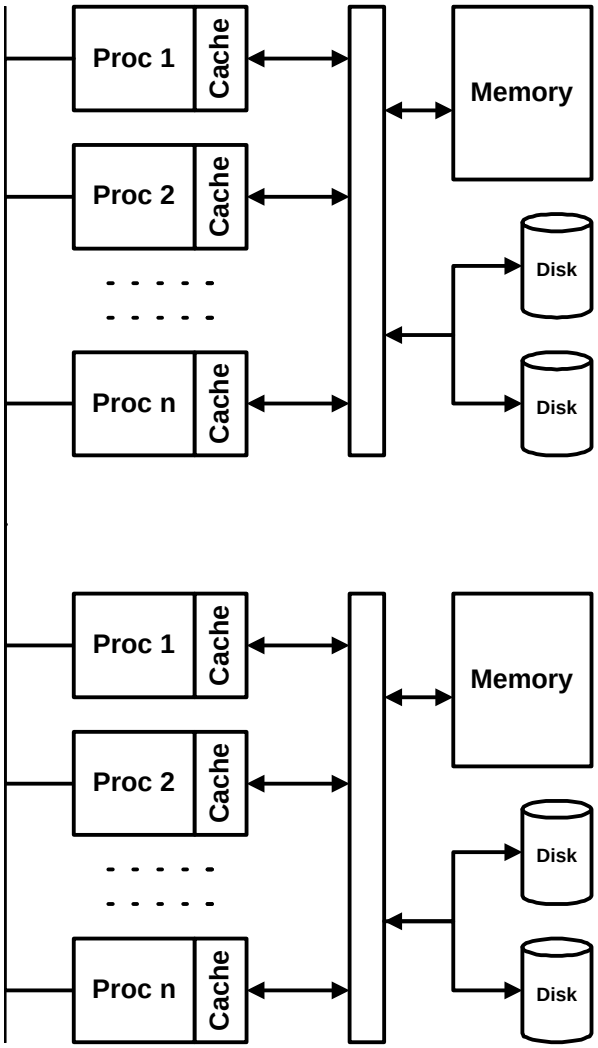
A solution that allows greater expandability is to ***cluster*** multiple computer nodes together. A cluster is a collection of interconnected ***whole computer systems*** that are used to ***work on a single problem***. Each computer system or "node" in the cluster can be a uniprocessor computer or an SMP multiprocessor.

Clusters – three models:

- Shared-nothing
- Shared-disk or shared-storage
- Shared-everything

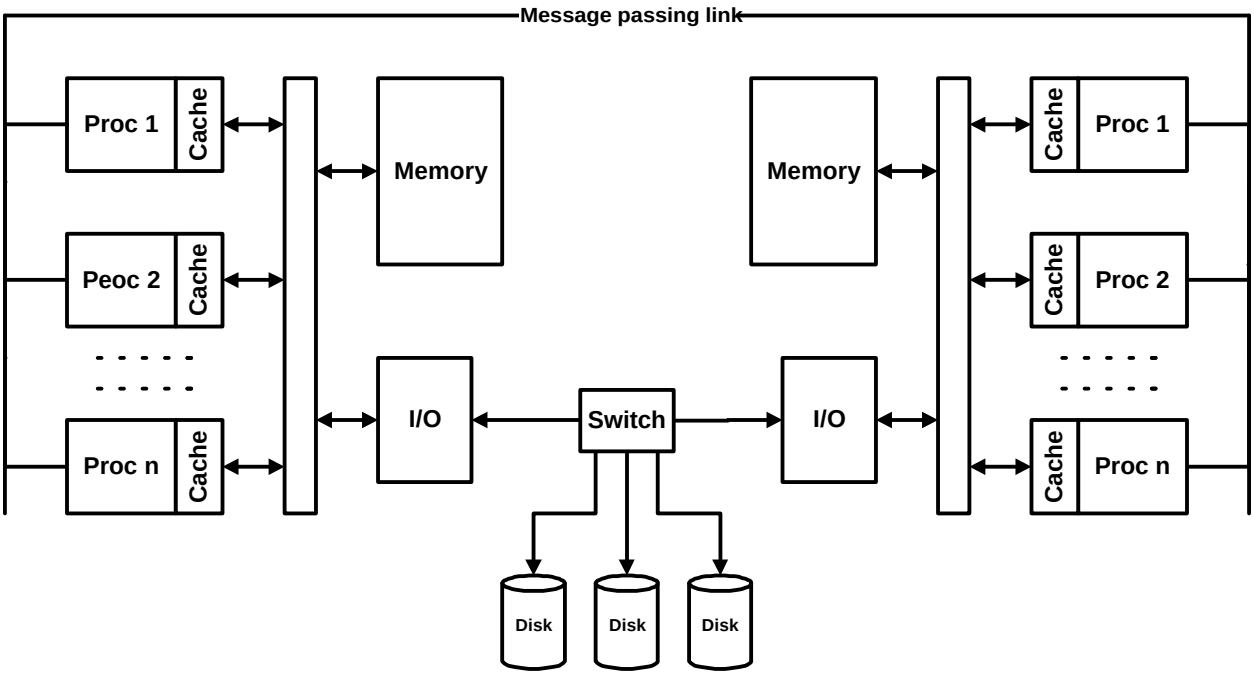
Actually, the shared-everything model is SMP, and not a cluster at all.

The Shared-nothing model:



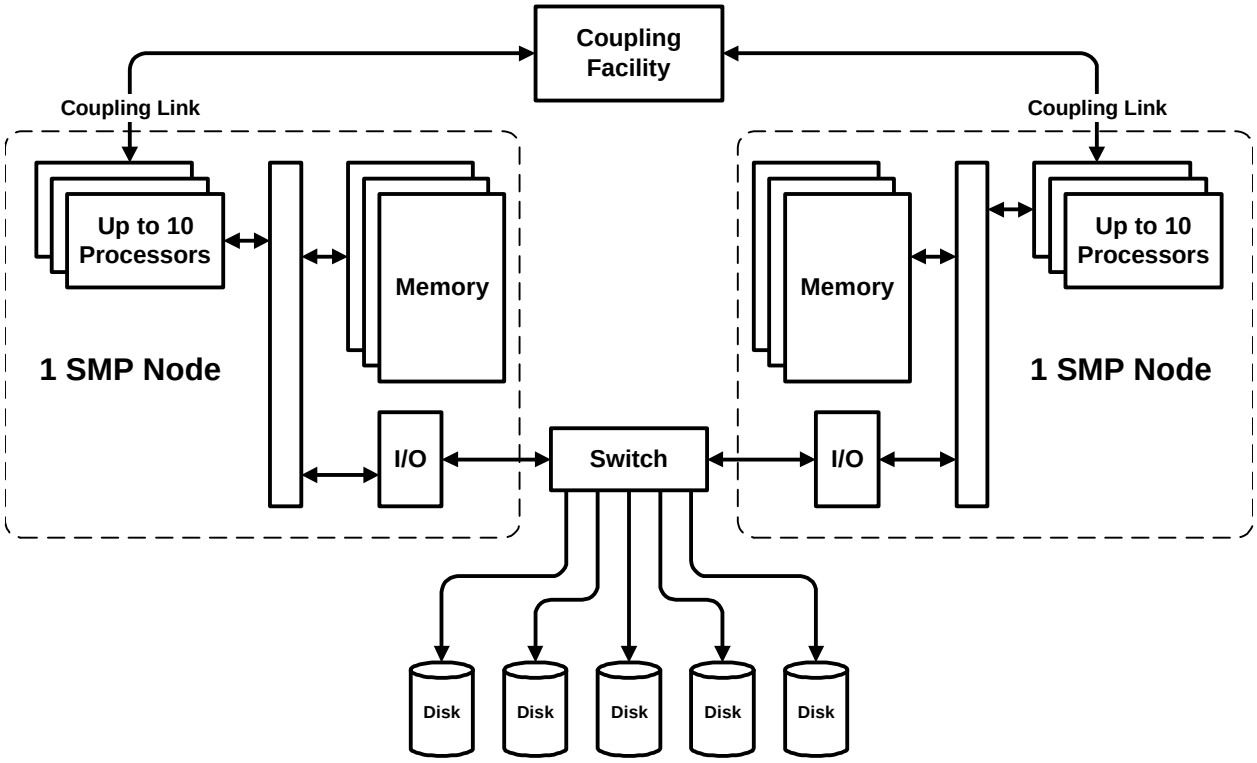
- Nodes are interconnected by some means to allow message passing.
- Need to partition the data, and then send a transaction to the node that has the relevant data.
- Each node has a separate instance of the operating system.
- No locking or disk cache coherency problems because each node can access only its local data.
- Excellent scalability.

The Shared-disk model:



- Nodes are partitioned and interconnected by some means to allow message passing.
- All disk storage is accessible from any node.
- Work can be dynamically balanced across all nodes.
- Each node has a separate instance of the operating system.
- Locking and disk cache coherency problems because of common access to shared data.
- Scalability problems because of common access to shared data.

IBM's S/390 Parallel Sysplex Cluster (shared-disk model)



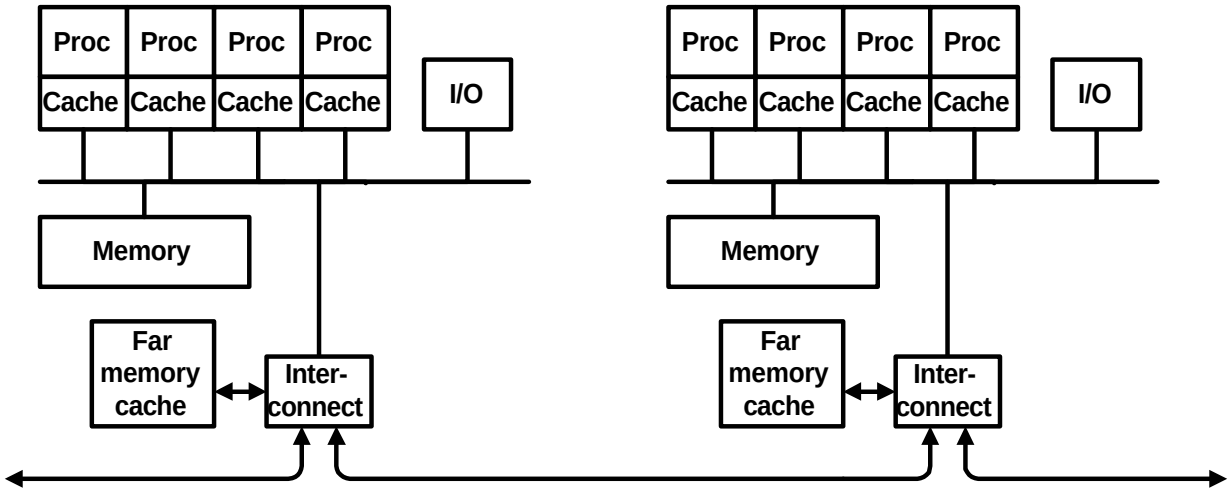
- Each node can be up to a 10-way SMP system.
- Up to 32 SMP nodes can be attached to the shared I/O and Coupling Facility, for a total of 320 processors.
- The Coupling Links are fiber-optic links running at 100 megabytes per second (800 megabits/sec).
- The Coupling Facility is implemented using a S/390 processor with control code to provide specific functions:

  ⇒ Locking facility
  ⇒ Cache management
  ⇒ Queue (message) management

- Special instructions are defined to access the coupling facility over the fiber-optic link. Synchronous access is 50 to 500 microseconds.

ccNUMA (cache coherent NUMA)

Cache coherent NUMA (ccNUMA) is a cross between UMA (SMP) and Clusters.



- Interconnect is the SCI (Scalable Coherent Interface). SCI is an ANSI/IEEE standard that handles cache coherency across multiple motherboards.
- All processors share the use of a single linear memory address space with a single copy of the operating system and application programs.
- Each SMP node provides memory for part of the total address space.
- The operating system must be aware of thread affinity to a particular processor or SMP node, and dispatch that work accordingly.
- Different from a cluster in that a cluster does not map memory into a single address space, and each node in a cluster has its own copy of the operating system and application programs.