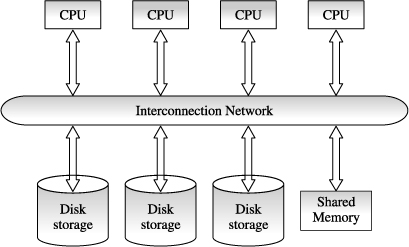
**Architecture of parallel database**

A **parallel**[**database**](https://en.wikipedia.org/wiki/Database) system seeks to improve performance through [parallelization](https://en.wikipedia.org/wiki/Parallelization) of various operations, such as loading data, building indexes and evaluating queries.[[1]](https://en.wikipedia.org/wiki/Parallel_database#cite_note-1)Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and [input/output](https://en.wikipedia.org/wiki/Input/output) speeds by using multiple [CPUs](https://en.wikipedia.org/wiki/CPU) and disks in parallel. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially. Parallel databases can be roughly divided into two groups, the first group of architecture is the multiprocessor architecture, the alternatives of which are the following:

**[Shared memory architecture](https://en.wikipedia.org/wiki/Shared_memory_architecture" \o "Shared memory architecture)**

In a shared-memory system, a computer has several (multiple) simultaneously active CPUs that are attached to an interconnection network and can share (or access) a single (or global) main memory and a common array of disk storage. Thus, in shared-memory architecture, a single copy of a multithreaded operating system and multithreaded DBMS can support multiple CPUs. Fig shows a schematic diagram of a shared-memory multiple CPU architecture. The shared-memory architecture of parallel database system is closest to the traditional single-CPU processor of centralised database systems, but much faster in performance as compared to the single-CPU of the same power. This structure is attractive for achieving moderate parallelism. Many commercial database systems have been ported to shared-memory platforms with relative ease.

Where multiple [processors](https://en.wikipedia.org/wiki/Central_processing_unit) share the [main memory (RAM)](https://en.wikipedia.org/wiki/Main_memory) space but each processor has its own disk (HDD). If many processes run simultaneously, the speed is reduced, the same as a computer when many parallel tasks run and the computer slows down.



**Benefits of shared-memory architecture**

* Communication between CPUs is extremely efficient. Data can be accessed by any CPU without being moved with software. A CPU can send messages to the other CPUs much faster by using memory writes, which usually takes less than a microsecond, than by sending a message through a communication mechanism.
* The communication overheads are low, because main memory can be used for this purpose and operating system services can be leveraged to utilise the additional CPUs.

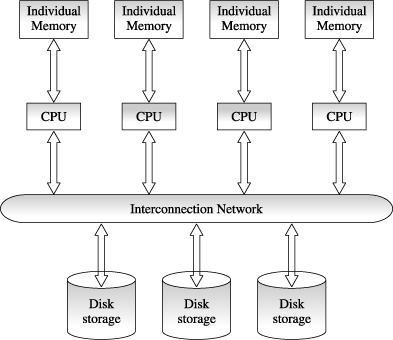
**Limitations of Shared-memory Architecture**

* Memory access uses a very high-speed mechanism that is difficult to partition without losing efficiency. Thus, the design must take special precautions that the different CPUs have equal access to the common memory. Also, the data retrieved by one CPU should not be unexpectedly modified by another CPU acting in parallel.
* Since the communication bus or interconnection network is shared by all the CPUs, the shared-memory architecture is not scalable beyond 80 or 100 CPUs in parallel. The bus or the interconnection network becomes a bottleneck as the number of CPUs increases.
* The addition of more CPUs causes CPUs to spend time waiting for their turn on the bus to access memory.

[**Shared disk architecture**](https://en.wikipedia.org/wiki/Shared_disk_architecture)

In a shared disk system, multiple CPUs are attached to an interconnection network and each CPU has its own memory but all of them have access to the same disk storage or, more commonly, to a shared array of disks. The scalability of the system is largely determined by the capacity and throughput of the interconnection network mechanism. Since memory is not shared among CPUs, each node has its own copy of the operating system and the DBMS. It is possible that, with the same data accessible to all nodes, two or more nodes may want to read or write the same data at the same time. Therefore, a kind of global (or distributed) locking scheme is required to ensure the preservance of data integrity. Sometimes, the shared-disk architecture is also referred to as a parallel database system. [Fig. 17.2](https://www.safaribooksonline.com/library/view/database-systems-concepts/9788177585674/9788177585674_ch17lev1sec3.html#ch17fig02)shows a schematic diagram of a shared-disk multiple CPU architecture.

Where each node has its own main memory, but all nodes share mass storage, usually a [storage area network](https://en.wikipedia.org/wiki/Storage_area_network). In practice, each node usually also has multiple processors.



**Benefits of Shared-disk Architecture**

* Shared-disk architecture is easy to load-balance, because data does not have to be permanently divided among available CPUs.
* Since each CPU has its own memory, the memory bus is not a bottleneck.
* It offers a low cost solution to provide a degree of fault tolerance. In case of a CPU or memory failure, the other CPUs take over its task; since the database is resident on disks that are accessible form all CPUs.
* It has found acceptance in wide applications.

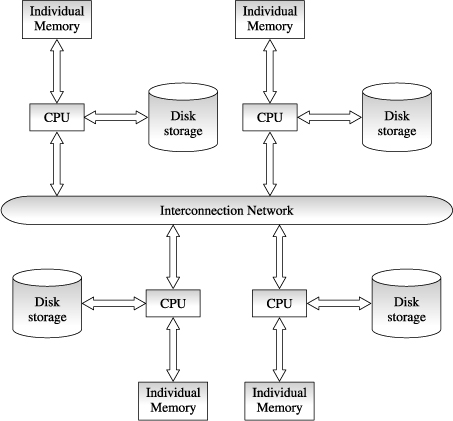
**Limitations of Shared-disk Architecture**

* Shared-disk architecture also faces similar problems of interference and memory contention bottleneck as the number of CPUs increases. As more CPUs are added, existing CPUs are slowed down because of the increased contention for memory accesses and network bandwidth.
* Shared-disk architecture also has a problem of scalability. The interconnection to the disk subsystem becomes bottleneck, particularly when the database makes a large number of accesses to the disks.

[**Shared nothing architecture**](https://en.wikipedia.org/wiki/Shared_nothing_architecture)

In a shared-nothing system, multiple CPUs are attached to an interconnection network through a node and each CPU has a local memory and disk storage, but no two CPUs can access the same disk storage area. All communication between CPUs is through a high-speed interconnection network. Node functions as the server for the data on the disk or disks that the node owns. Thus, shared-nothing environments involve no sharing of memory or disk resources. Each CPU has its own copy of operating system, its own copy of the DBMS, and its own copy of a portion of data managed by the DBMS. [Fig. 17.3](https://www.safaribooksonline.com/library/view/database-systems-concepts/9788177585674/9788177585674_ch17lev1sec3.html#ch17fig03) shows a schematic diagram of a shared-nothing multiple CPU architecture. In this type of architecture, CPUs sharing responsibility for database services usually split up the data among themselves. CPUs then perform transactions and queries by dividing up the work and communicating by message over the high-speed network (at the rate of megabits per second).

Where each node has its own mass storage as well as main memory.



**Benefits of Shared-nothing Architecture**

* Shared-nothing architectures minimise contention among CPUs by not sharing resources and therefore offer a high degree of scalability.
* Since local disk references are serviced by local disks at each CPU, the shared-nothing architecture overcomes the limitations of requiring all I/O to go through a single interconnection network. Only queries, accesses to non-local disks and result relations pass through the network.

##### Limitations of Shared-nothing Architecture

* Shared-nothing architectures are difficult to load-balance. In many multi CPU environments, it is necessary to split the system workload in some way so that all system resources are being used efficiently. Proper splitting or balancing this workload across a shared-nothing system requires an administrator to properly partition or divide the data across the various disks such that each CPU is kept roughly as busy as the others. In practice this is difficult to achieve.
* Adding new CPUs and disks to shared-nothing architecture means that the data may need to be redistributed in order to make advantage of the new hardware (resource) and thus requires more extensive reorganisation of the DBMS code.
* Shared-nothing architectures introduce a single point of failure to the system. Since each CPU manages its own disk(s), data stored on one or more of these disks become inaccessible if its CPU goes down.
* It requires an operating system that is capable of accommodating the heavy amount of messaging required to support inter-processor communications