

**12<sup>th</sup> International DEXA Workshop: Parallel and Distributive Databases**

**Munich, September 6, 2001**

# **Design and Evaluation of Database Multiprocessor Architecture with High Data Availability**

**Leonid B. Sokolinsky**

**Chelyabinsk State University, Russia**

[www.csu.ru/~sok](http://www.csu.ru/~sok)

sokol@csu.ru

---

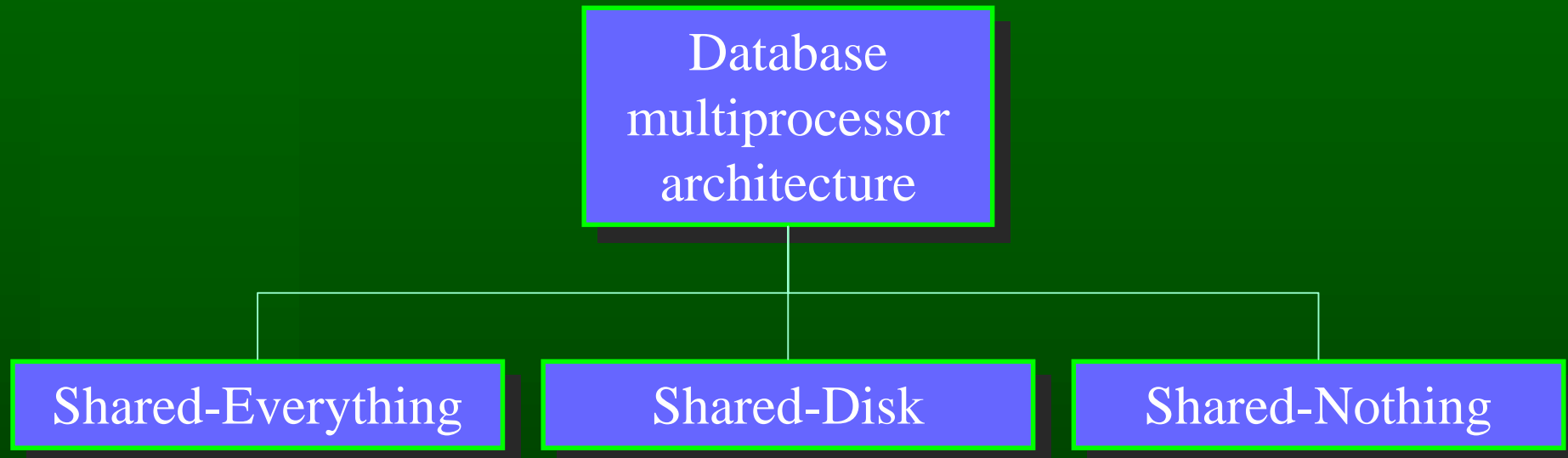
This work was supported by the Russian Foundation for Basic Research under Grant 00-07-90077

# Outline

---

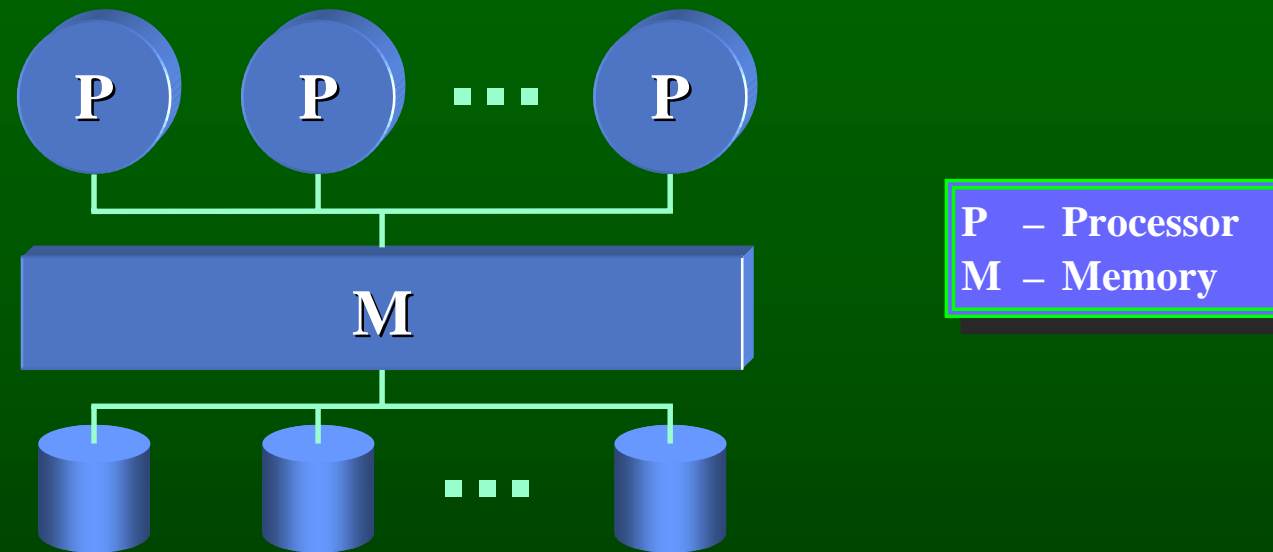
- **Introduction**
- **Omega System Architecture**
- **Design Solutions**
  - **Intracluster Data Shadowing**
  - **Algorithm of load balancing**
  - **Stream model of query parallelization**
- **Experimental Results**
- **Conclusion**

# Three classical architectures



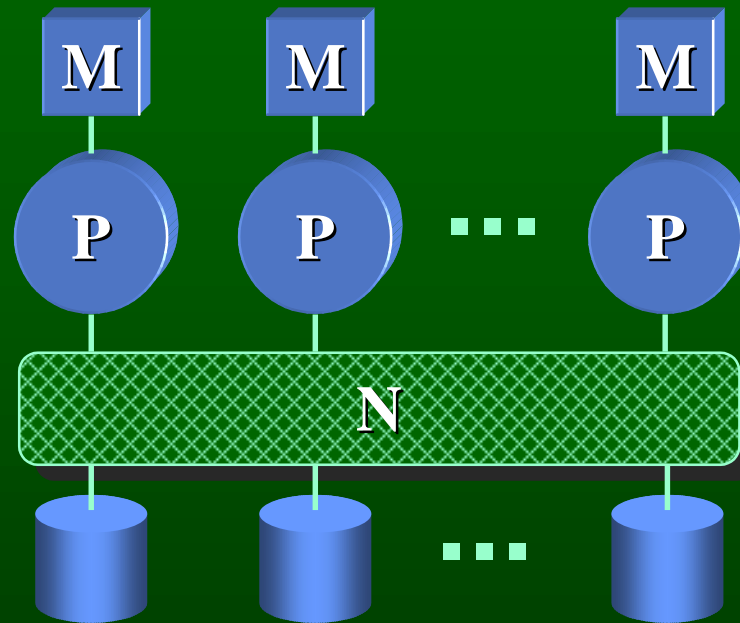
There are three well known classical architectures of parallel database systems. They are *shared-everything* (or *shared-memory*), *shared-disk* and *shared-nothing*. However each of these architectures suffers from different disadvantages.

# Shared-Everything (SE) Architecture



The *shared-everything* architecture suffers from low scalability and reliability. However it's very attractive from standpoint of performance and simplicity.

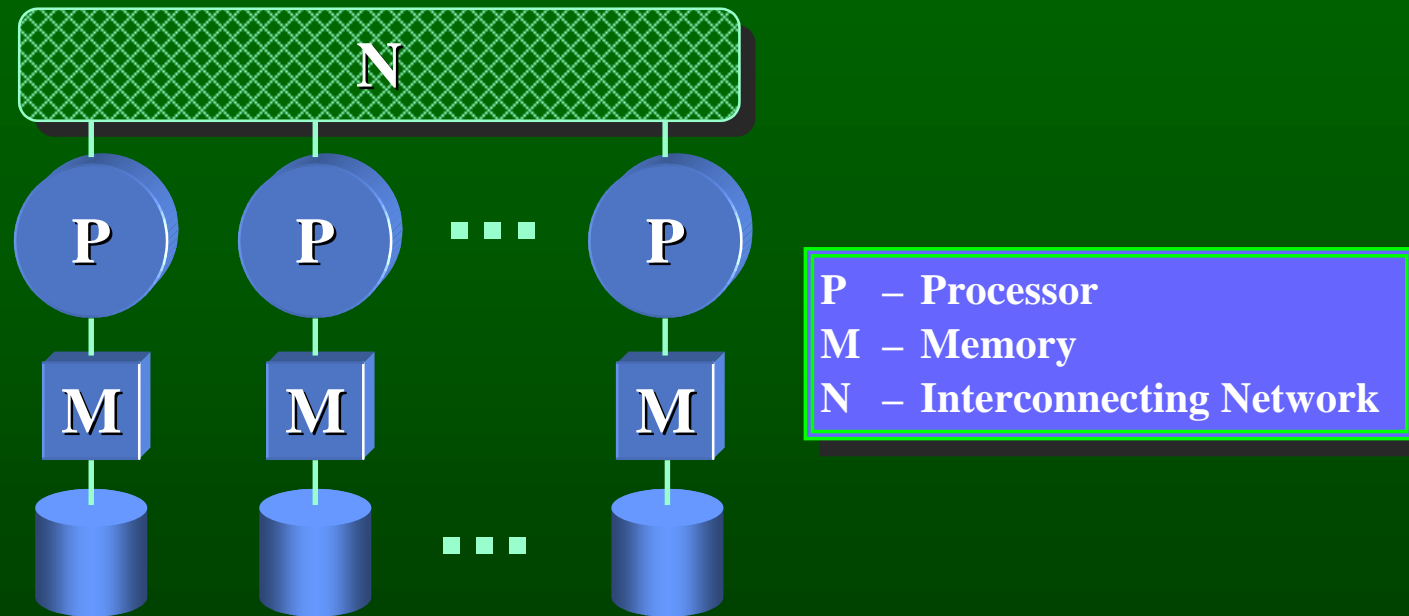
# Shared-Disk (SD) Architecture



P – Processor  
M – Memory  
N – Interconnecting Network

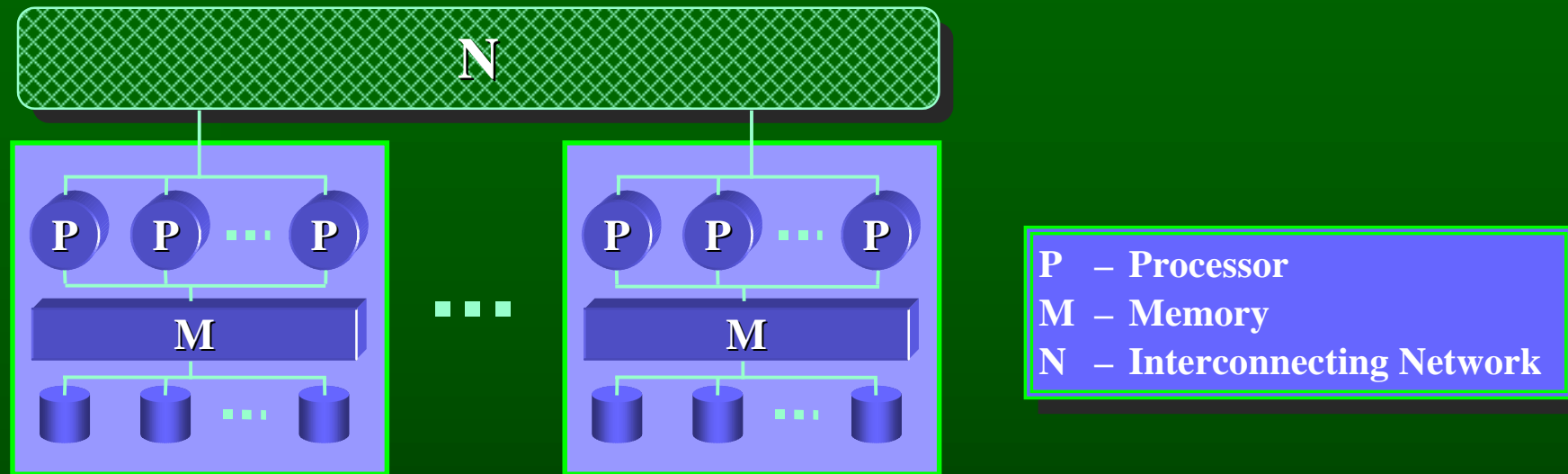
The *shared-disk* architecture introduces too many locking and buffer coherency problems, therefore it offers no particularly persuasive arguments for its support.

# Shared-Nothing (SN) Architecture



The *shared-nothing* architecture is attractive from the standpoint of scalability and reliability. However, load balancing is difficult to achieve, and it results in high sensitivity of performance to data skew.

# Clustered-Everything (CE) architecture



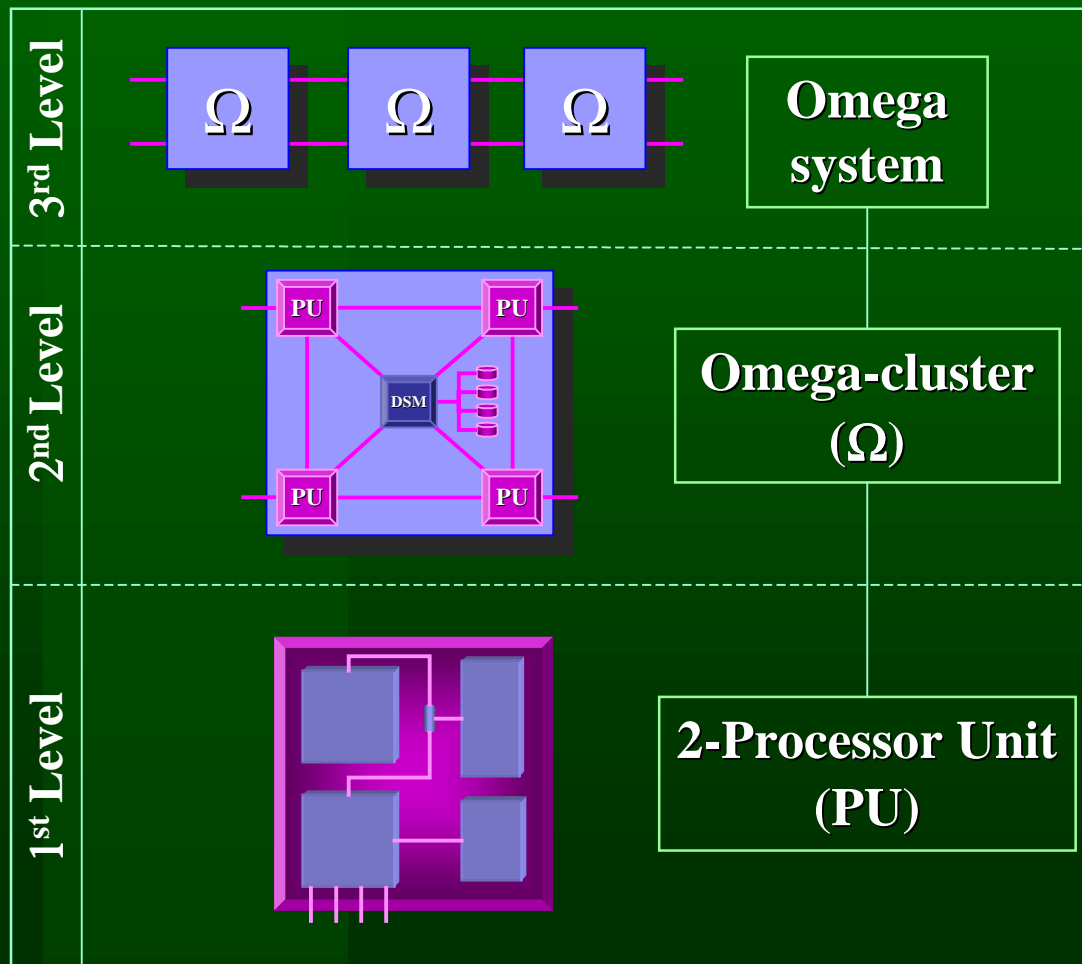
To incorporate the best features of shared-nothing and shared-everything architectures Bhide suggested to consider a hierarchical database multiprocessor architecture in which SE-clusters (inner level) are combined in SN-manner (outer level). We denote it as *Clustered-Everything* (CE) architecture.

CE-architecture demonstrates a good scalability. The load balancing can be arranged more easily than in SN-architecture because it can be addressed at two levels, locally among the processors of each cluster and globally among all clusters.

However, CE-architecture can't provide the high data availability because SE-cluster has low reliability. To overcome this difficulties, we suggested an alternative three-level hierarchical architecture, which is based on the notion of *omega-cluster*.

# Omega System Architecture

## Tree-Level Hierarchy



This tree-level hierarchical architecture had been designed for the Russian parallel database machine called *Omega System*.

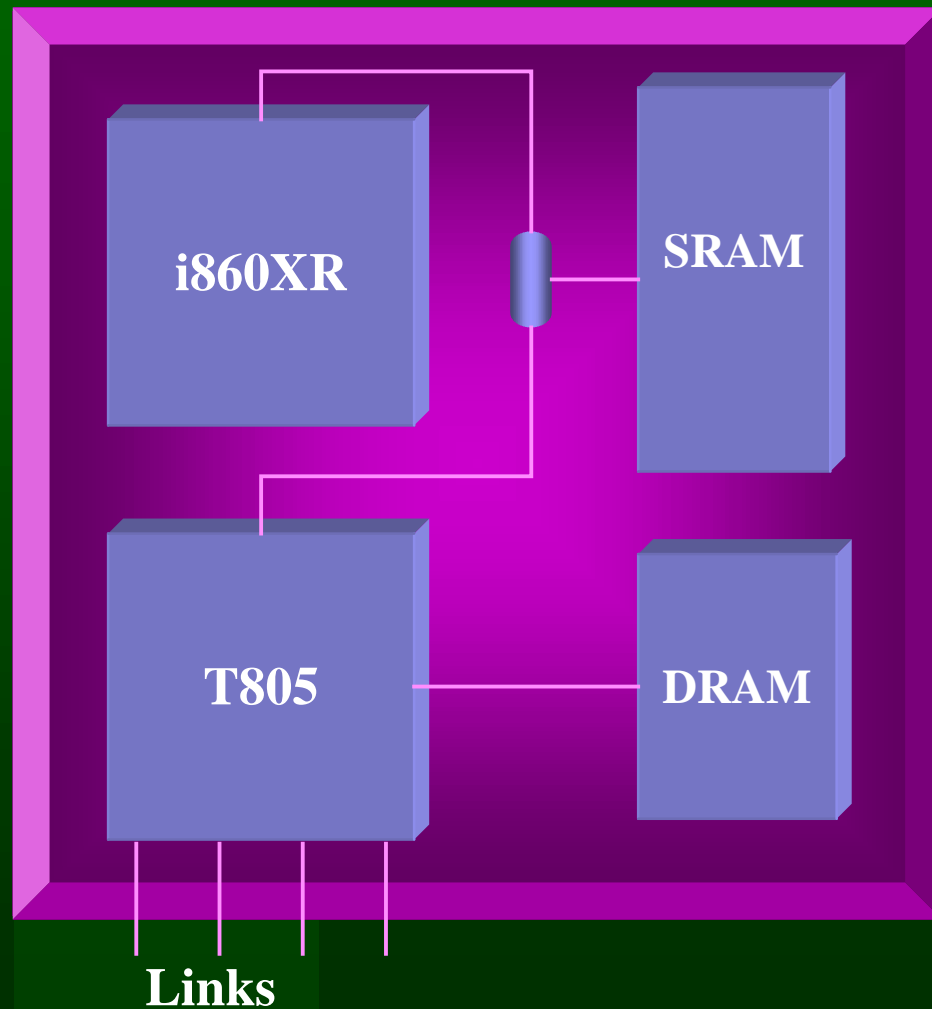
The *first level* of hierarchy is presented by the processor units.

The processor units are composed in *omega-clusters* that form the *second level* of the hierarchy.

On the *third level*, the omega-clusters are integrated in the whole system in shared-nothing manner.

Let's consider each level in details.

# Omega Processor Unit

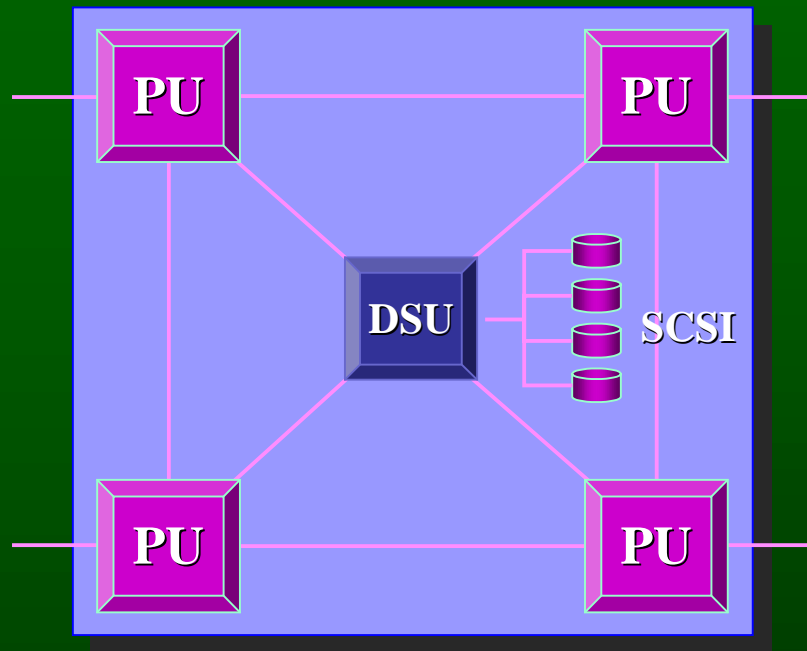



The Omega *processor unit* has a shared memory architecture with 16-128 Mbytes of static random access memory (SRAM) coupled to both a computational processor (i860XR), and a communicational processor (T805), which has four serial links to handle I/O and board-to-board communications.

The communicational processor has also 8-64 Mbytes of private dynamic random access memory (DRAM).

The two processors exchange data in shared memory, with interrupts and full bus locking for synchronization.

# Omega-Cluster Structure



 2-Processor Unit

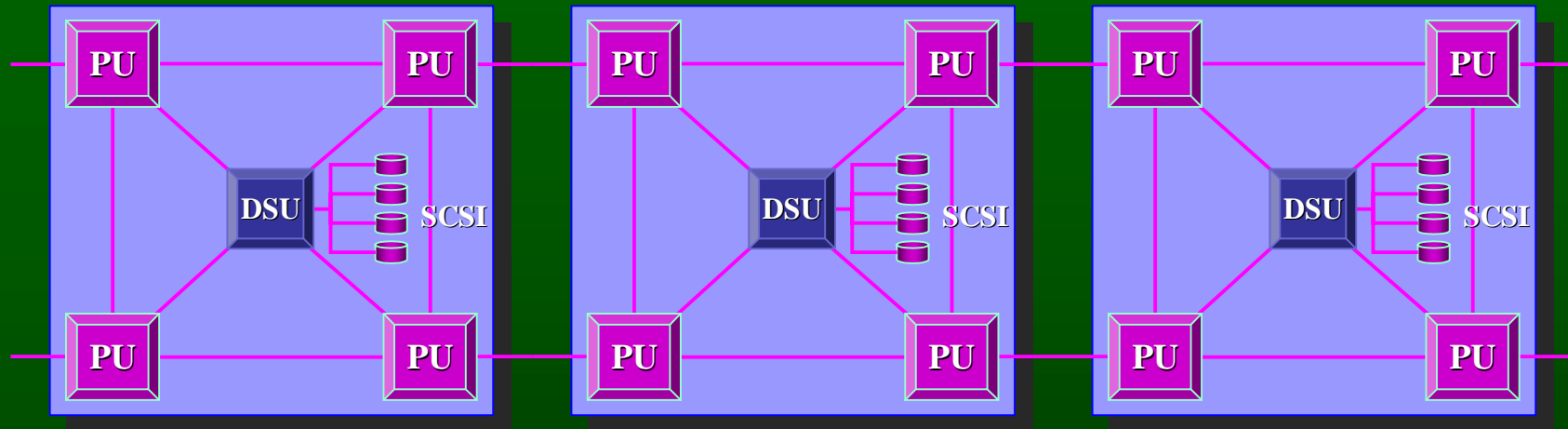
 Disk Subsystem Unit

The *omega-cluster* has shared-disk architecture and consists of four processor units connected by links.

Every processor unit is directly connected to the *disk subsystem unit* (DSU). DSU has its own communication processor with private memory connected to four disks by SCSI bus. One link of each processor unit remains free to connect the given omega-cluster to others omega-clusters.

The capacity of SCSI bus is enough to prevent DSU becoming a bottleneck.

# Structure of Omega System



At the third (outer) level of hardware hierarchy, omega-clusters are composed into the *whole Omega system* in a shared-nothing manner. One Omega system can be scaled up to several hundreds omega-clusters. There is no restriction for the interconnect topology in the Omega system. It can be varied from simple pipeline to hypercube.

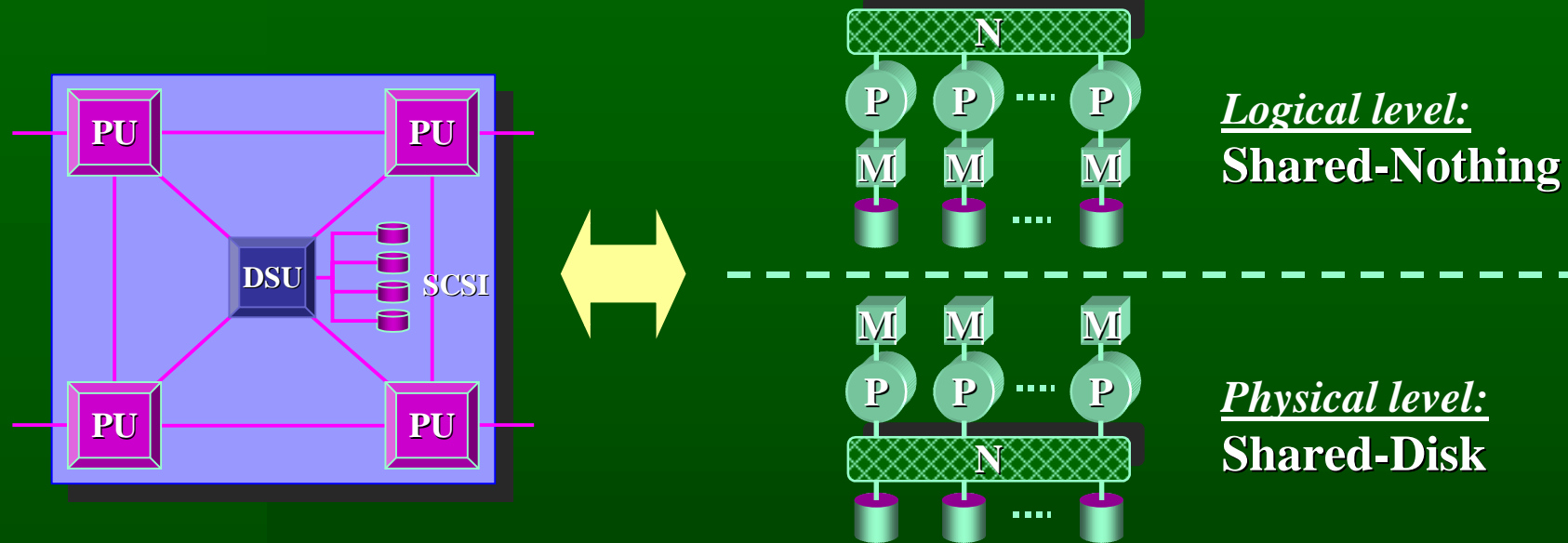
# Design Solutions

---

In the Omega System we use the following main design solutions:

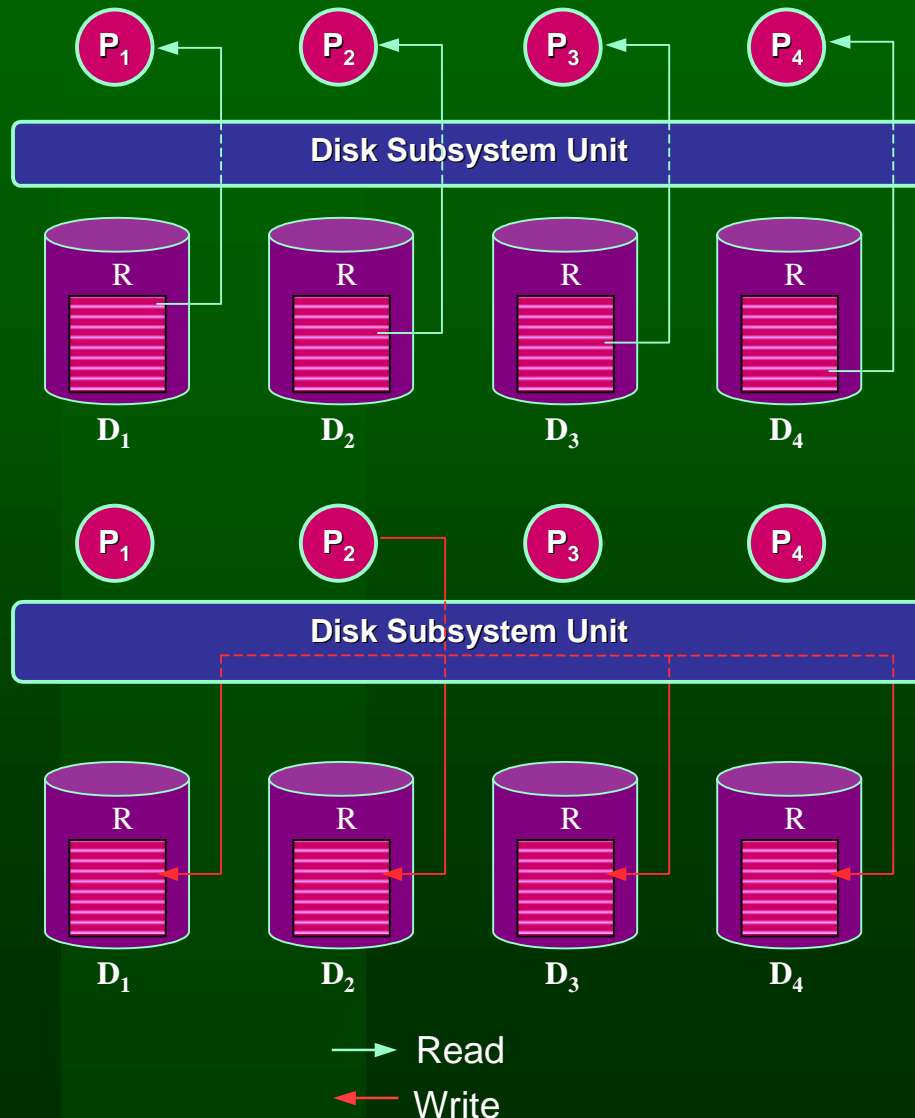
- **Dual nature of omega-cluster**
- **Intracluster data shadowing**
- **Stream model of query parallelization**

# Dual Nature of Omega-Cluster



In the omega-cluster, the disk pool is divided on the logical level into nonintersecting subsets of physical disks, each of which forms the so-called virtual disk. The number of virtual disks in an omega-cluster is a constant and equal to the number of the processor units. In the simplest case, one physical disk corresponds to one virtual disk. Each processor unit is associated with one private virtual disk. Thus, on the logical level, the omega-cluster may be viewed as a system with the SN architecture, whereas, on the physical level, it is a system with the SD architecture. This dual nature of omega-cluster is used in an original algorithm of load balancing, which is considered below.

# Intracuster Data Shadowing



The hierarchical architecture of the Omega system suggests two levels of fragmentation. Each relation can be divided into fragments placed in different omega-clusters (intercluster fragmentation). Each fragment, in its turn, is divided into smaller fragments (*partitions*) distributed among different nodes of the omega-cluster (intracuster fragmentation).

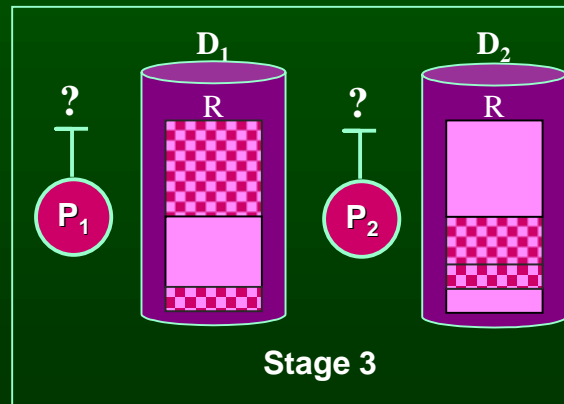
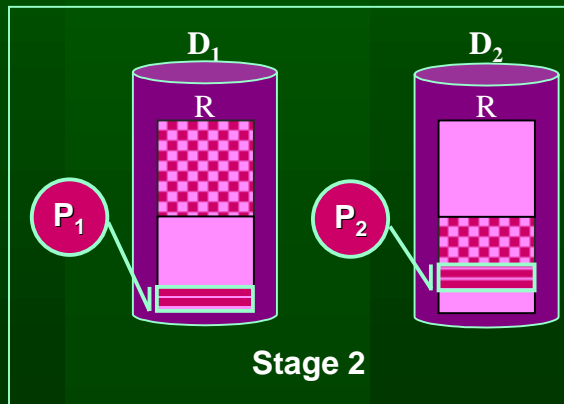
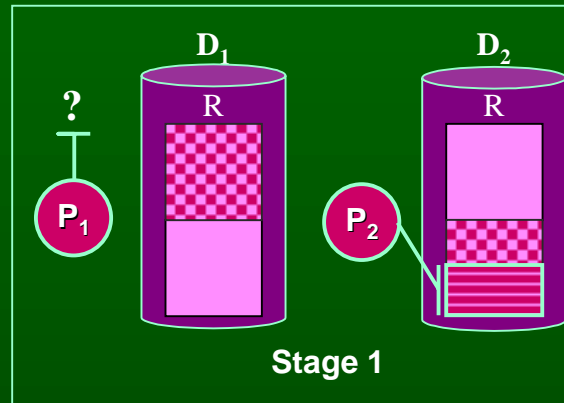
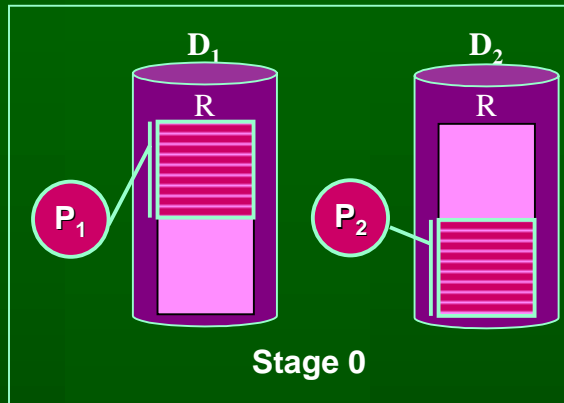
To handle load balancing, we use an original algorithm based on the method called *intracuster data shadowing*. It means replicating each partition across all cluster nodes.



Reading: The processors can't concurrently read the same partition of R. However, the processors can access different partitions of R in parallel mode.

Writing: If one processor is writing to R, Disk Subsystem automatically propagates the updates on all disks.

Due to special omega-cluster structure, the intracuster data shadowing can be arranged by the disk subsystem unit without excessive overhead.

# Algorithm of load balancing



-  - scheduled for processing
-  - processed

Let's consider the load balancing algorithm by using an omega-cluster with two nodes.

At stage 0, the partitions of the relation  $R$  are divided into two equal subsets. The first one is scheduled for processing by CPU  $P_1$  and the second one is scheduled for processing by  $P_2$ .

At stage 1, processor  $P_1$  has performed its own part of the work and has become idle. At the same time, the processor  $P_2$  has performed only half of its part of the work.

At stage 2, in order to handle data skew, we divide the unprocessed set of partitions into two subsets and schedule them to be processed by processor  $P_1$  and processor  $P_2$ , respectively.

At stage 3,  $P_1$  and  $P_2$  have performed their work approximately at the same time.

During all stages, each processor reads the data only from its own disk. Such technique allows us to avoid a huge interconnect traffic in order to manage data skew by redistributing the partitions between processor nodes.

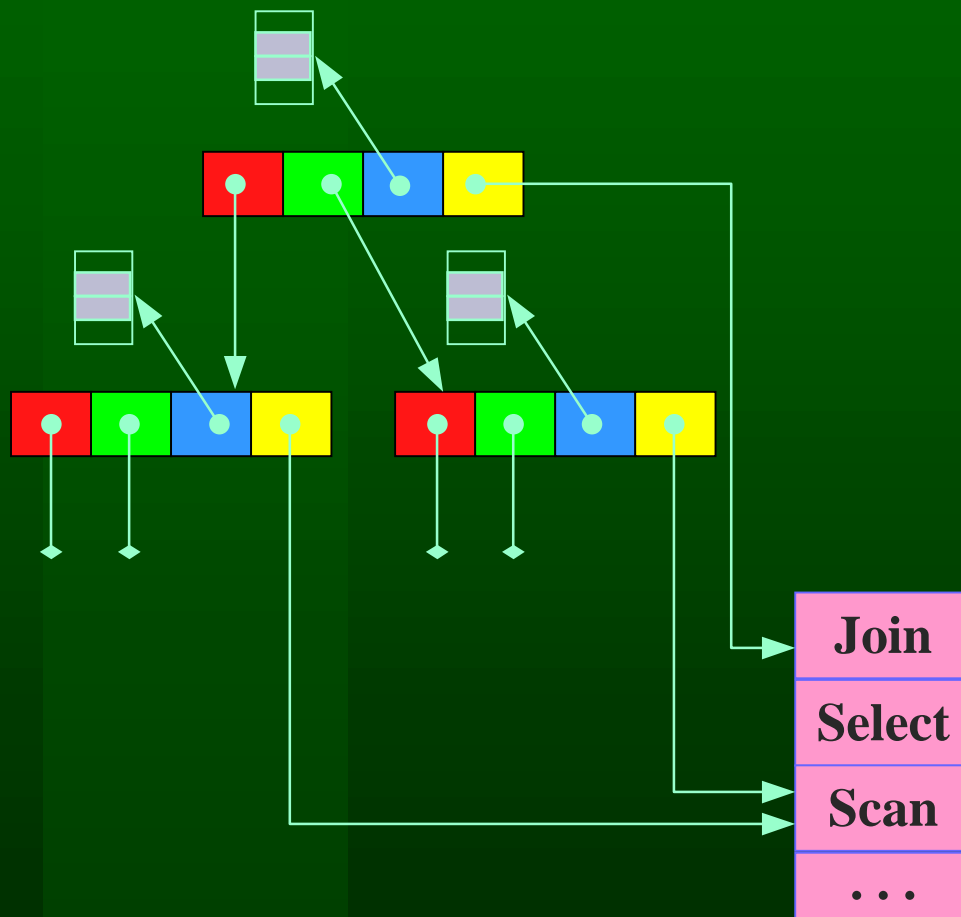
# Stream model of query parallelization

---

In the Omega system, we use an original mechanism for parallelizing query execution, which we called the *stream model*. This model utilizes the consumer/producer paradigm and data-driven dataflow mechanism for exchanging data between operators efficiently. The stream model incorporates advantages of both the bracket and operator models and use the following main tools:

- **Generic bracket template**
- **Stream objects**
- **Omega exchange operator**

# Generic bracket template

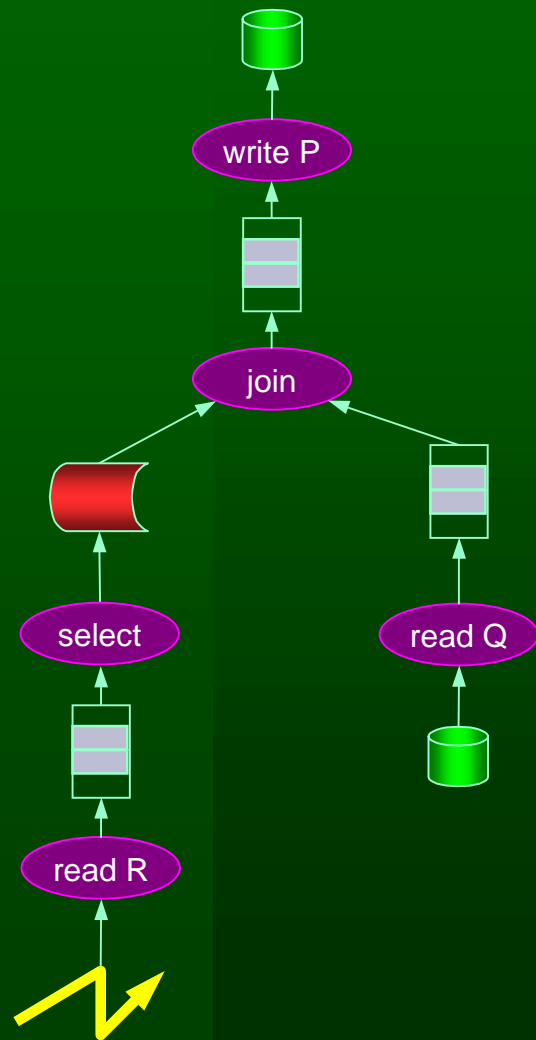


In the Omega system, query tree nodes are implemented by using the generic *bracket template* that can consume and produce data granules and can execute exactly one operator. Each query tree node is built on bracket template by assigning the following main attributes:

- **pointer to the function implementing the relational operation,**
- **pointer to the left son,**
- **pointer to the right son,**
- **pointer to the *output stream*.**

During the query tree execution, all bracket templates that compose the tree are executed concurrently as threads in one process.

# Stream objects

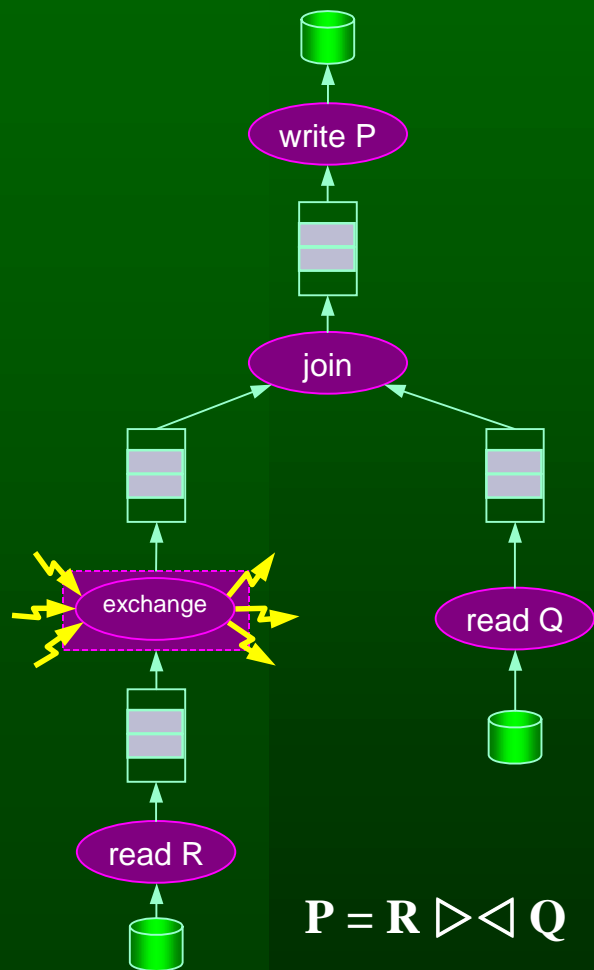


A *stream* is an object that operates as a virtual FIFO file. The Omega system supports the following four kinds of stream objects:

- **stored relation,**
- **temporary file,**
- **message passing channel,**
- **stock.**

The *stock* is a buffer, which is served as a queue. In the Omega system, the stocks are used for supporting pipeline parallelism. If stock length is equal to one, the stock model actually becomes equivalent to the classical iterator model. However, as it will be shown below, the stock model can provide the better system performance in the presence of data skew.

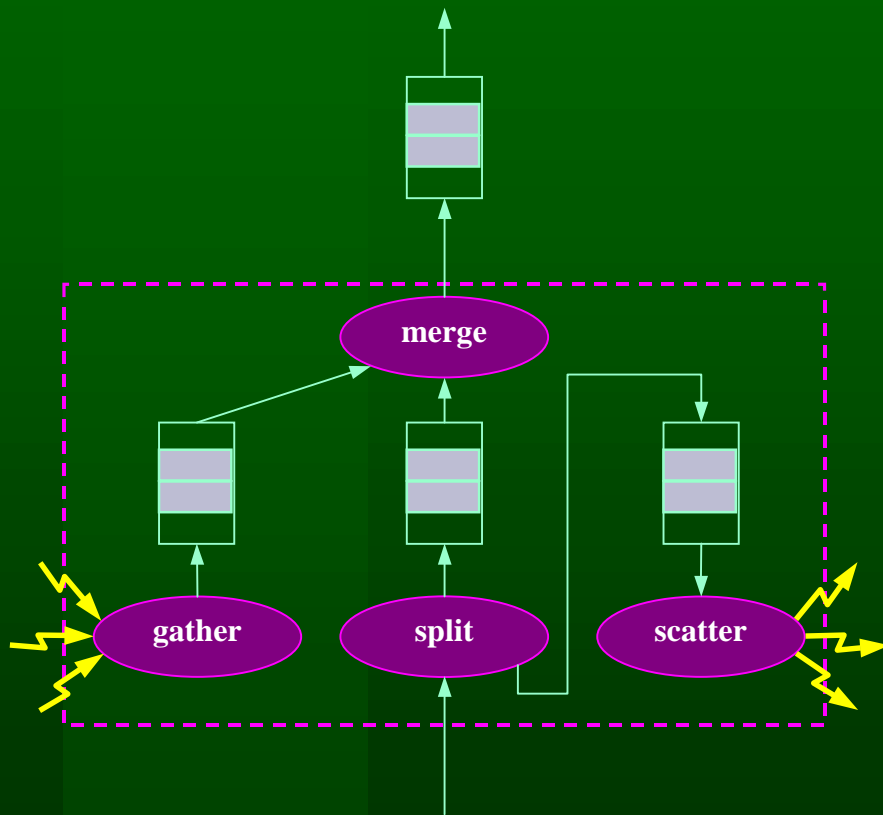
# Omega exchange operator



In order to support intraoperator parallelism on partitioned datasets, we introduced the *Omega exchange operator*. The figure shows a join plan parallelization by using the Omega exchange operator.

For the sake of simplicity we assume here that input Q is partitioned on the join attribute and input R is partitioned on another attribute. The join plan is executed simultaneously on all omega-cluster nodes. Each processor unit processes its own part of R and Q datasets. The exchange operator redistributes data granules of R dataset among all omega-cluster nodes using the Q partitioning function.

# Exchange Operator Structure



The Omega exchange operator consists of the following four suboperators: split operator, gather operator, scatter operator and merge operator. All these operators are implemented by using the generic bracket template.

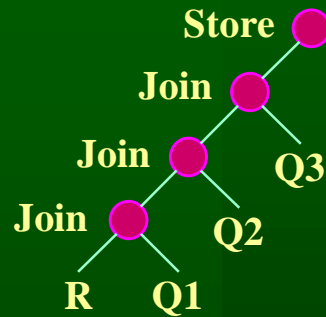
The *split operator* divides its input into two parts using the partitioning function. The first part includes just those data granules, which have to be processed inside the local node. These granules are directed to the output stock of the split operator. The second part consists of those data granules, which have to be processed on remote nodes. These granules are directed into the output stock of the scatter operator, which serves as an input stream.

The *scatter operator* consumes data granules from its own output stream and sends them to the corresponding omega-cluster nodes by using the partition function. In context of generic bracket template, the scatter operator is implemented as an operator whose output stream serves as input stock.

The *gather operator* permanently reads data granules from all omega-cluster nodes excepting its own node and puts these granules into its output stock.

The *merge operator* gets data granules from its input streams and merges them into its output stream.

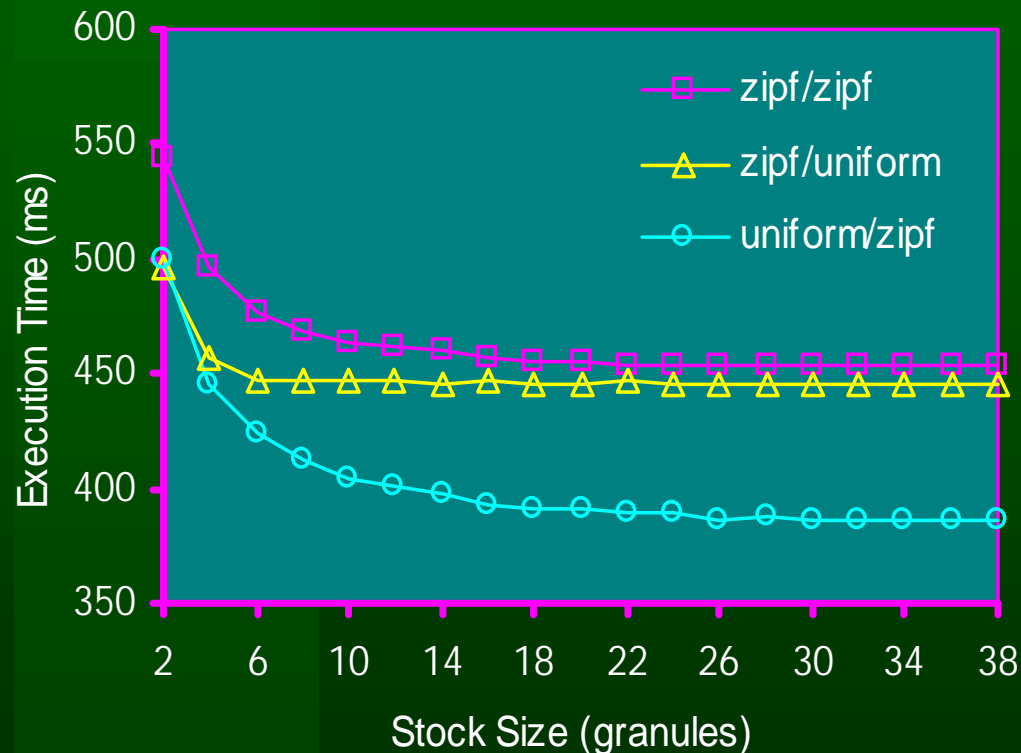
# Experimental Results



We implemented our execution model on a 8-processor MBC-100 computer. Using this implementation, we studied the model under various system and database conditions. In the experiments, we used left-deep join trees like the one shown in the figure. We assume that relation R consists of tree attributes: a1, a2, a3. The a1 is the join attribute for Q1, the a2 is the join attribute for Q2, and the a3 is the join attribute for Q3.

In our experiments we study the Zipfian and uniform value distributions for the join attributes.

# Execution Time versus Stock Size

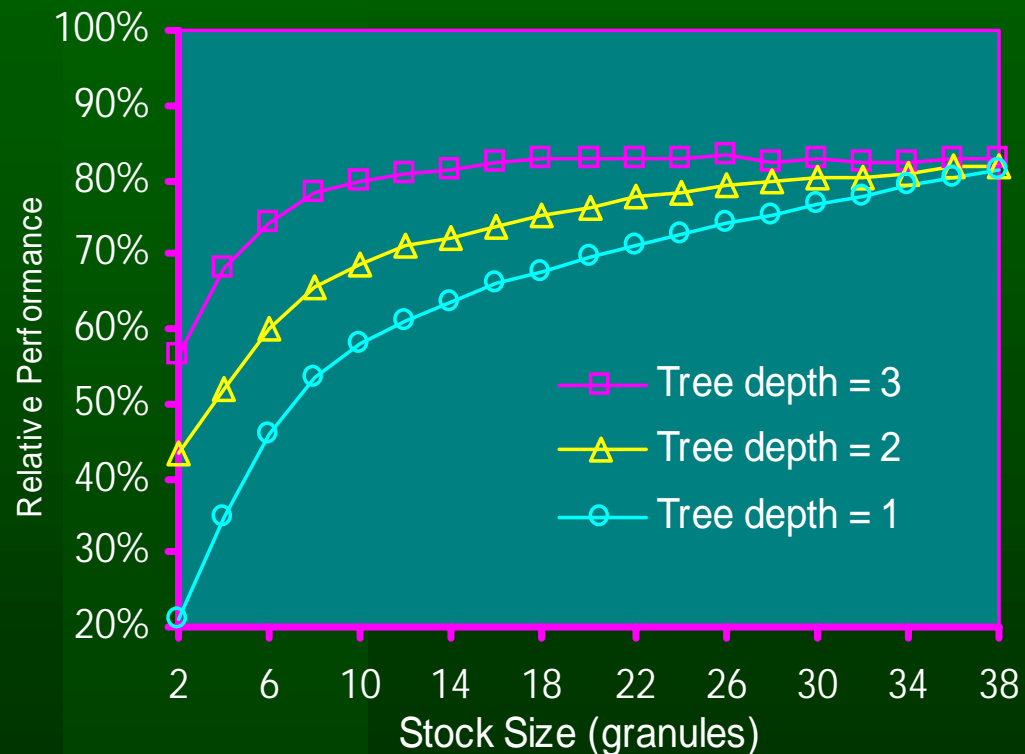


The first experiment examines the effect of increasing the stock size for various value distributions. The figure shows the variation in execution time with the size of the stock for the different distribution couples:

R distribution /  $Q_i$  distribution.

We can see in that the effect of stock size increase is insignificant if  $Q_i$  join attribute has the uniform distribution. In this case, the minimum of the curve is achieved at point 8, which provides approximately 10% of the response time reduction. However, if  $Q_i$  join attribute has Zipfian distribution, the minimum of the curve is achieved at point 26, which provides more than 20% of the response time reduction.

# Omega-cluster versus SE-cluster



In the second experiment, we studied the performance of the omega-cluster versus a shared-everything cluster. In this experiment, we simulated the 4-processor shared-everything cluster on the 4-processor omega-cluster by using an *ideal distribution* of join attribute values for  $R$  and  $Q_i$ . This ideal distribution prevents any delay to obtain the required data element at any point of time in all nodes during processing the multiple join.

The important conclusion from this experiment is that we can significantly increase the performance of the omega-cluster by using the stream model. Moreover, the proposed hierarchical architecture can provide the overall system performance, which can be comparable with one provided by the systems with shared-everything clusters, having better reliability in the same time.

# Conclusion

---

- We suggested a new hierarchical multiprocessor architecture, which can be used for construction of high-performance and reliable database systems. The distinctive features of the suggested architecture are fault tolerance, high degree of data availability, and the possibility of efficient load balancing under conditions of data skew.
- The model described has been implemented in a prototype of parallel DBMS Omega on the basis of eight-processor configuration of the MBC-100 system. This prototype has been used in computational experiments, which substantiated high efficiency of the approach suggested.

# URL

---

[www.csu.ru/~sok](http://www.csu.ru/~sok)