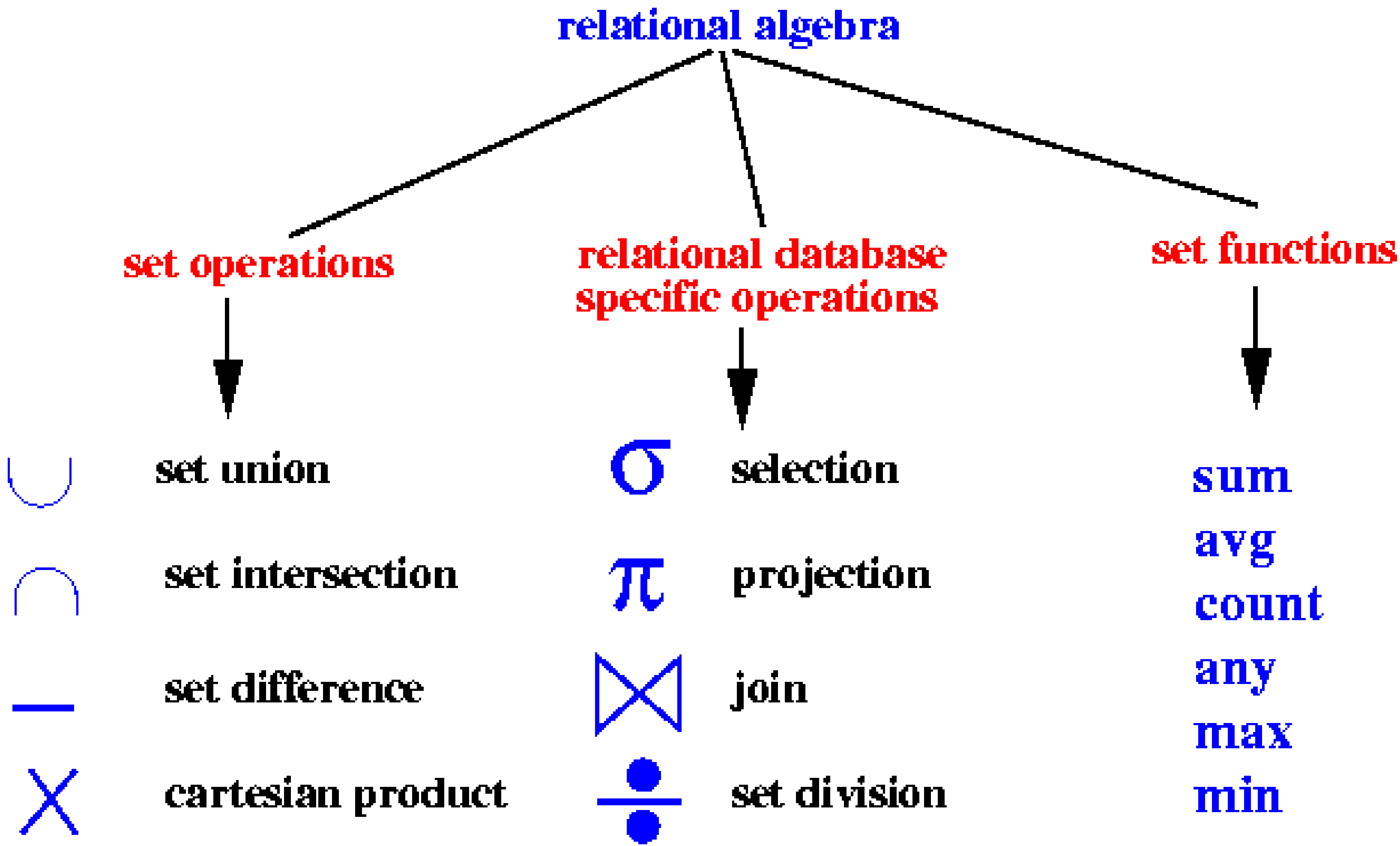
Fundamental Operators

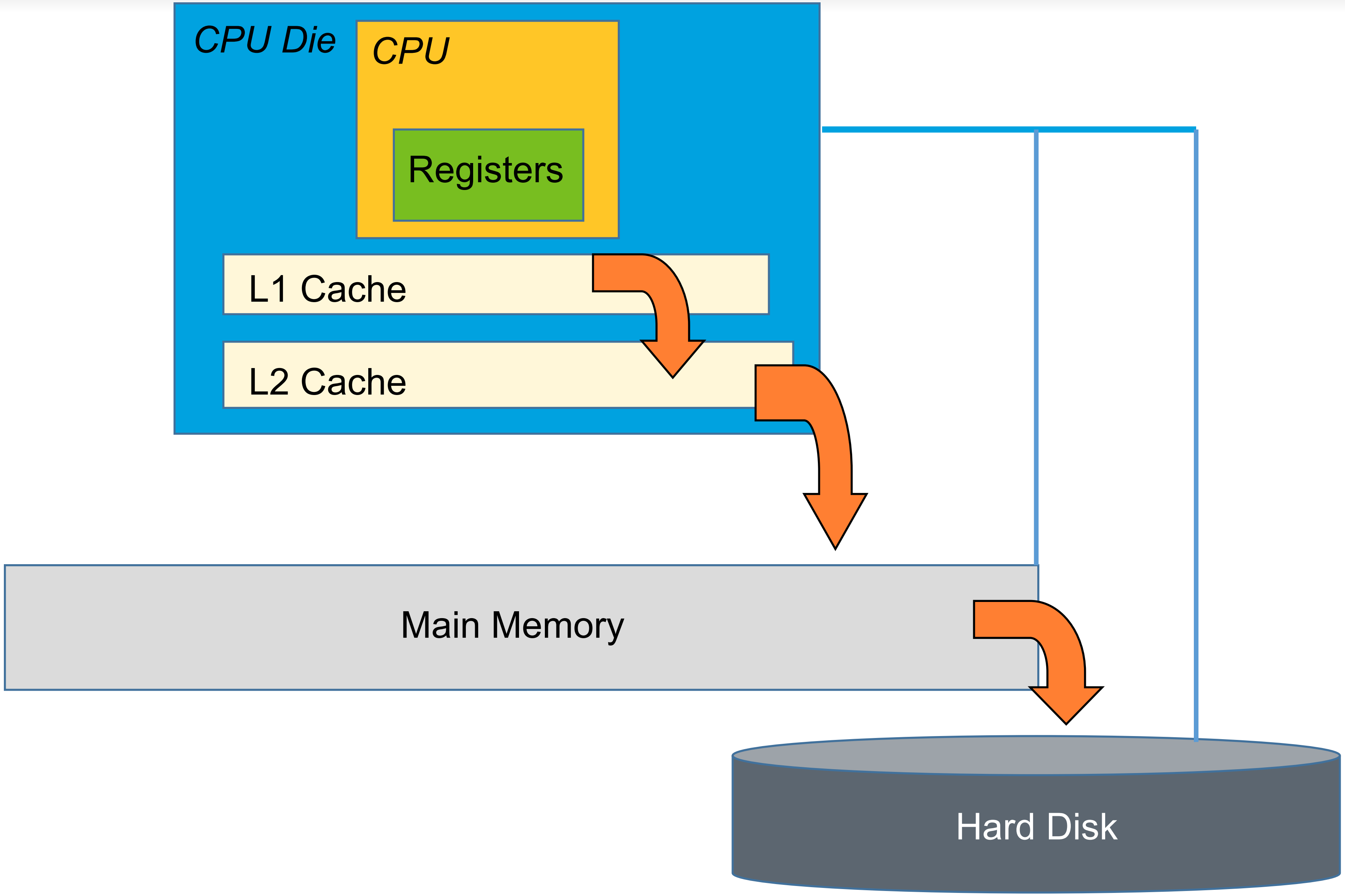
Let **r** and **s** be relations with **schemas R and S**

union	$r \cup s = \{ t \mid t \in r \lor t \in s \}$
difference	$r - s = \{ t \mid t \in r \land t \notin s \}$
cartesian_product	$r \times s = \{ t \mid t = t_r\, t_s \text{ where } t_r \in r \land t_s \in s \}$
selection	$\sigma_P(r)$
projection	$\pi_A(r)$


relational algebra
set operations
relational database specific operations
set functions
set union
set intersection
set difference
cartesian product
selection
projection
join
set division
sum
avg
count
any
max
min

The Memory Hierarchy

Example:
Intel PIII
CPU: 450MHz
Memory: 512MB



Internal Data Storage

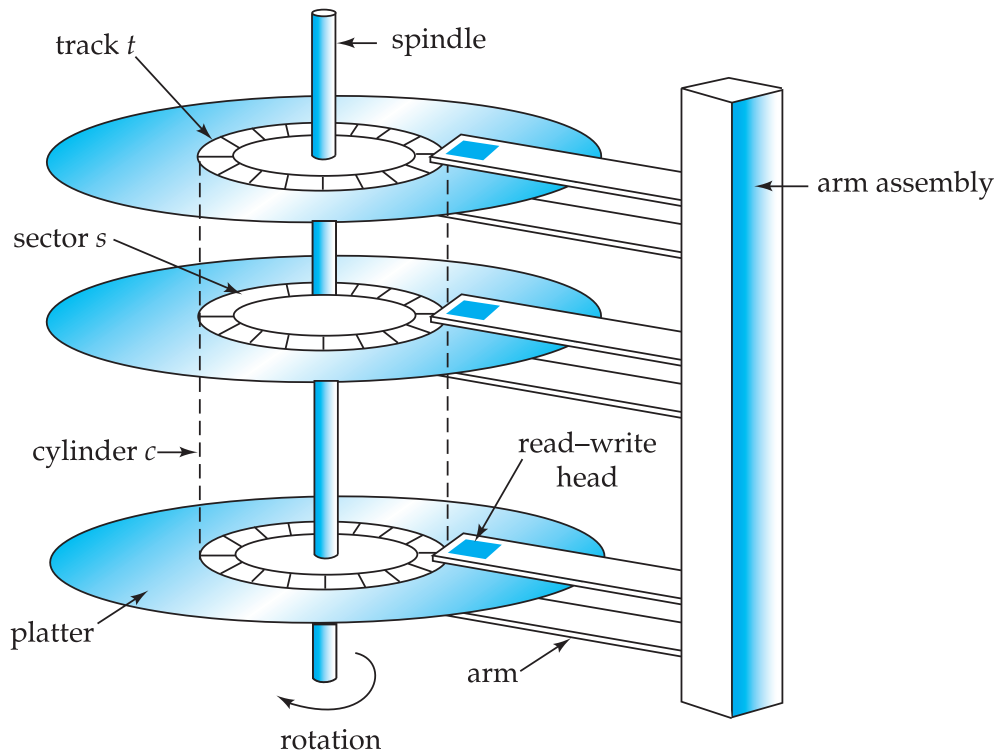
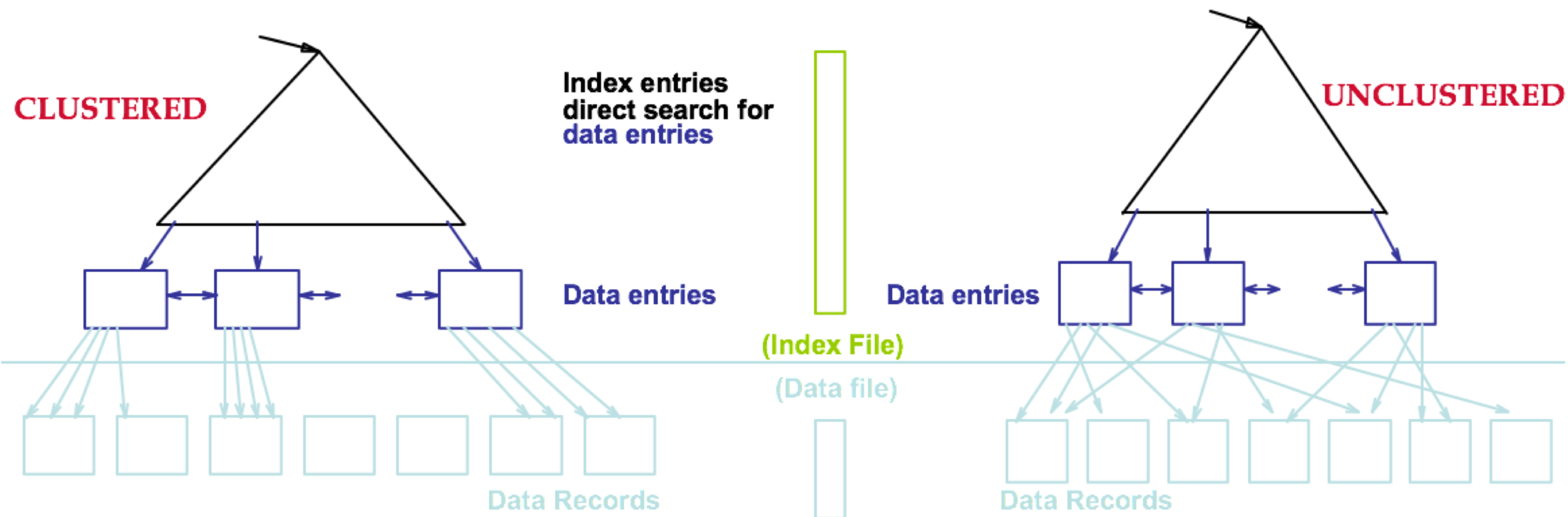
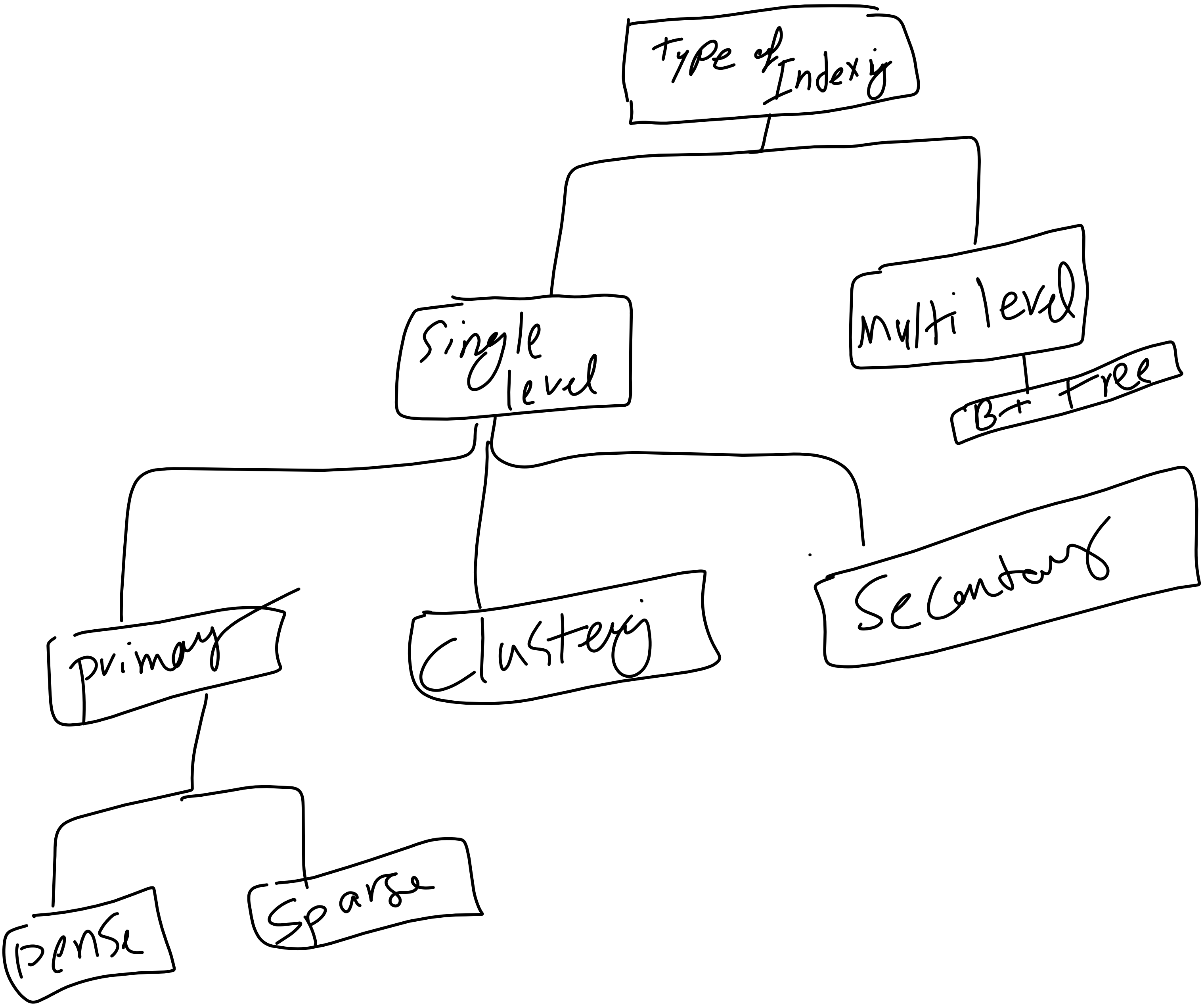


Image source: https://www.snia.org

Clustered vs. Unclustered Index



Types of Indexing



1-single level

a- Primary Indexing

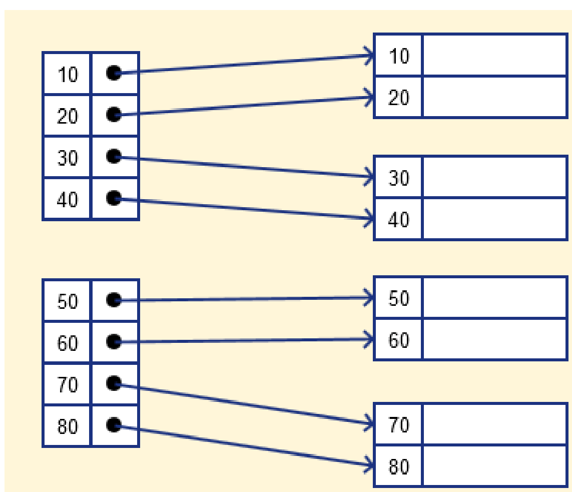
Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

The primary Indexing is also further divided into two types.

- Dense Index
- Sparse Index

a-1- Dense Index

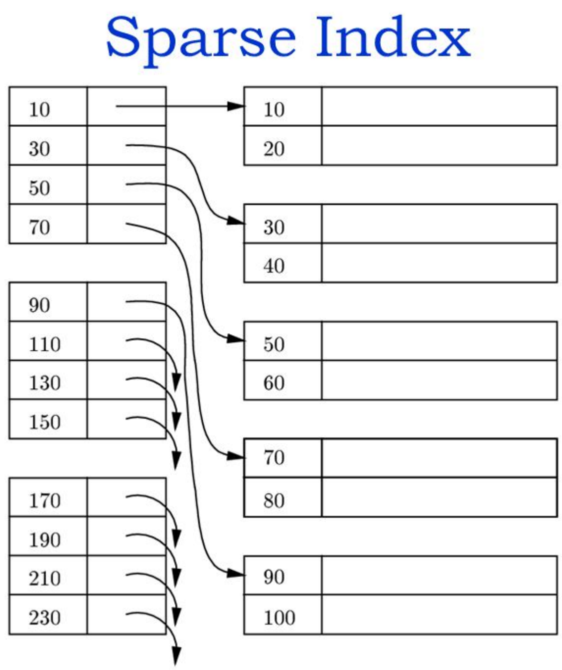
- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the disk.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



a-2-Sparse Index

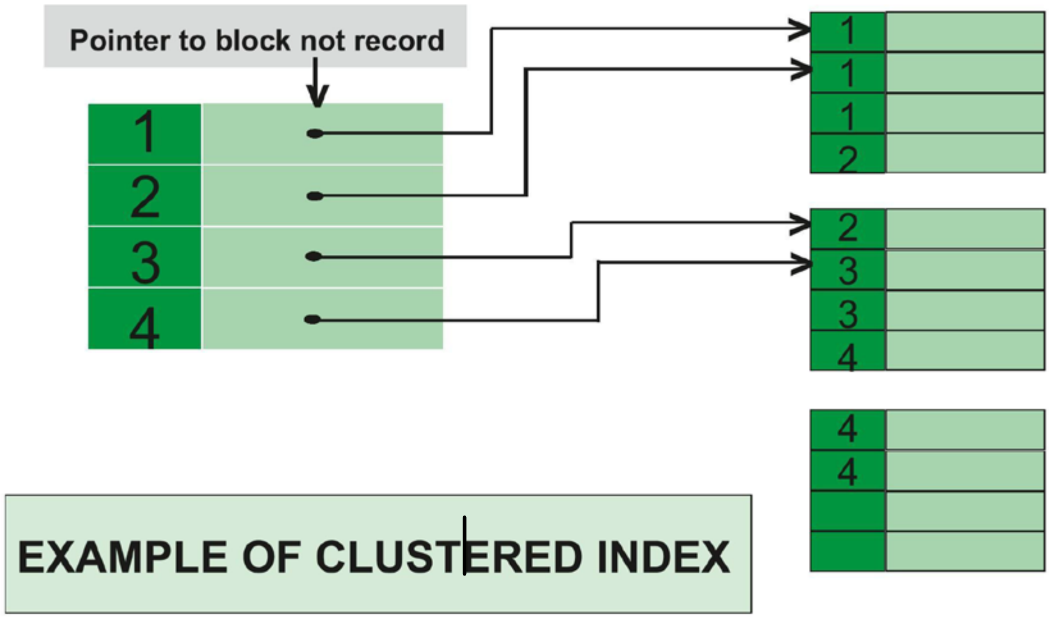
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.

Example of Sparse Index



b- Clustering Index

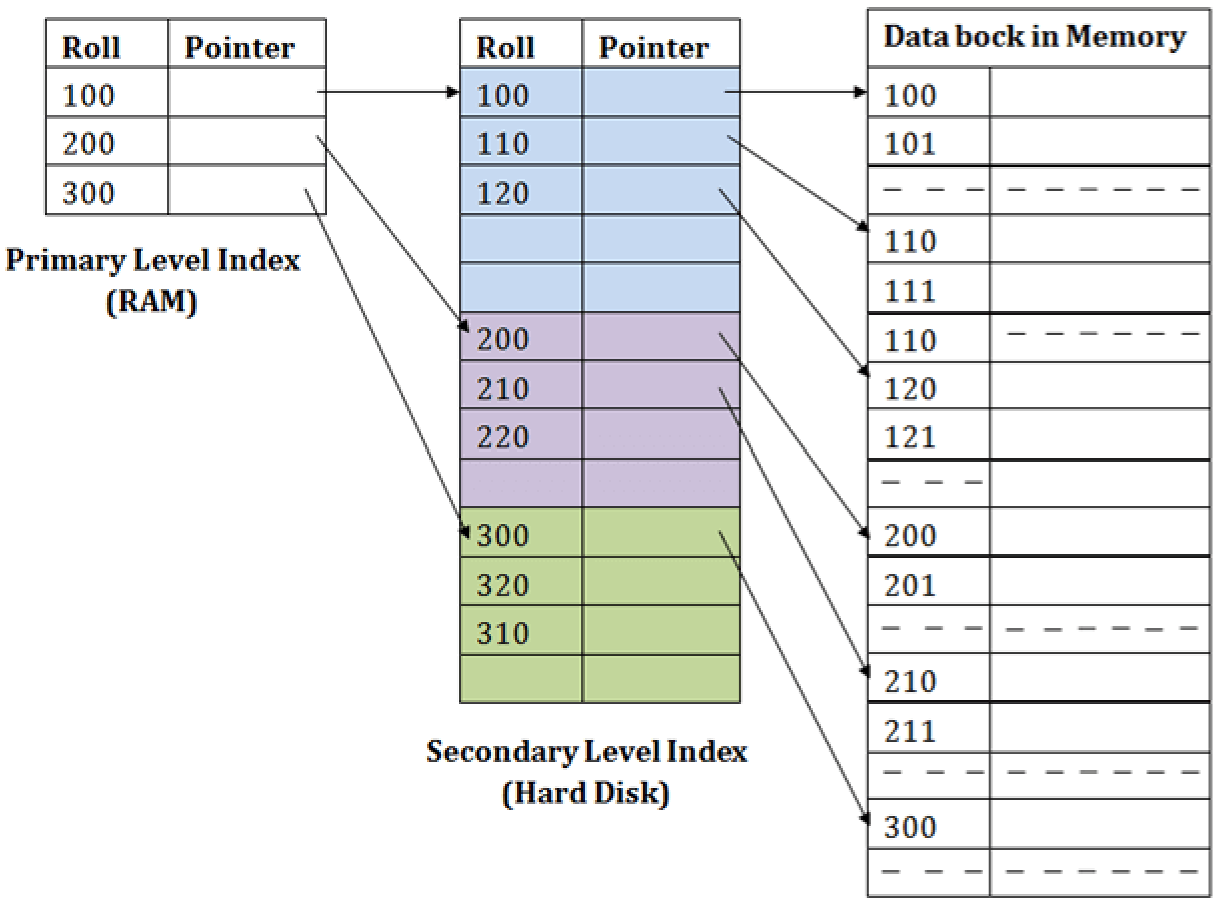
- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.



c- Secondary Index(non-clustering index)

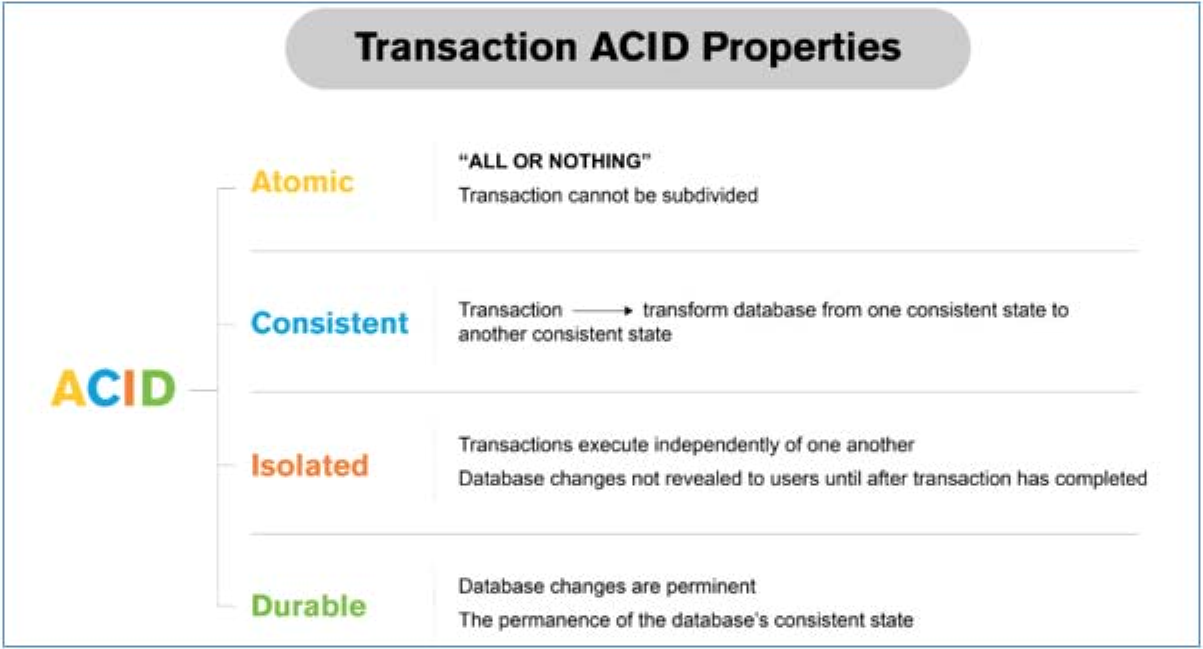
In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).

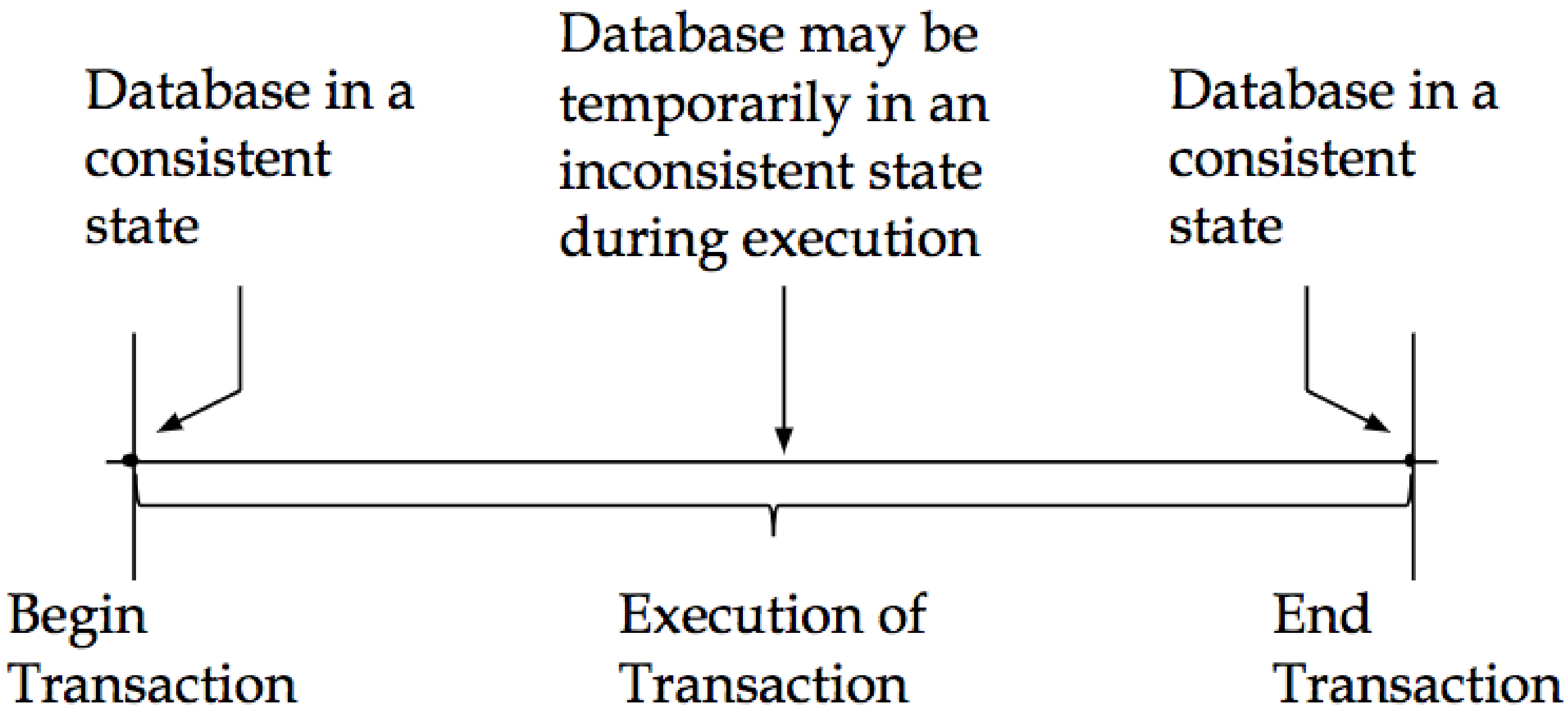


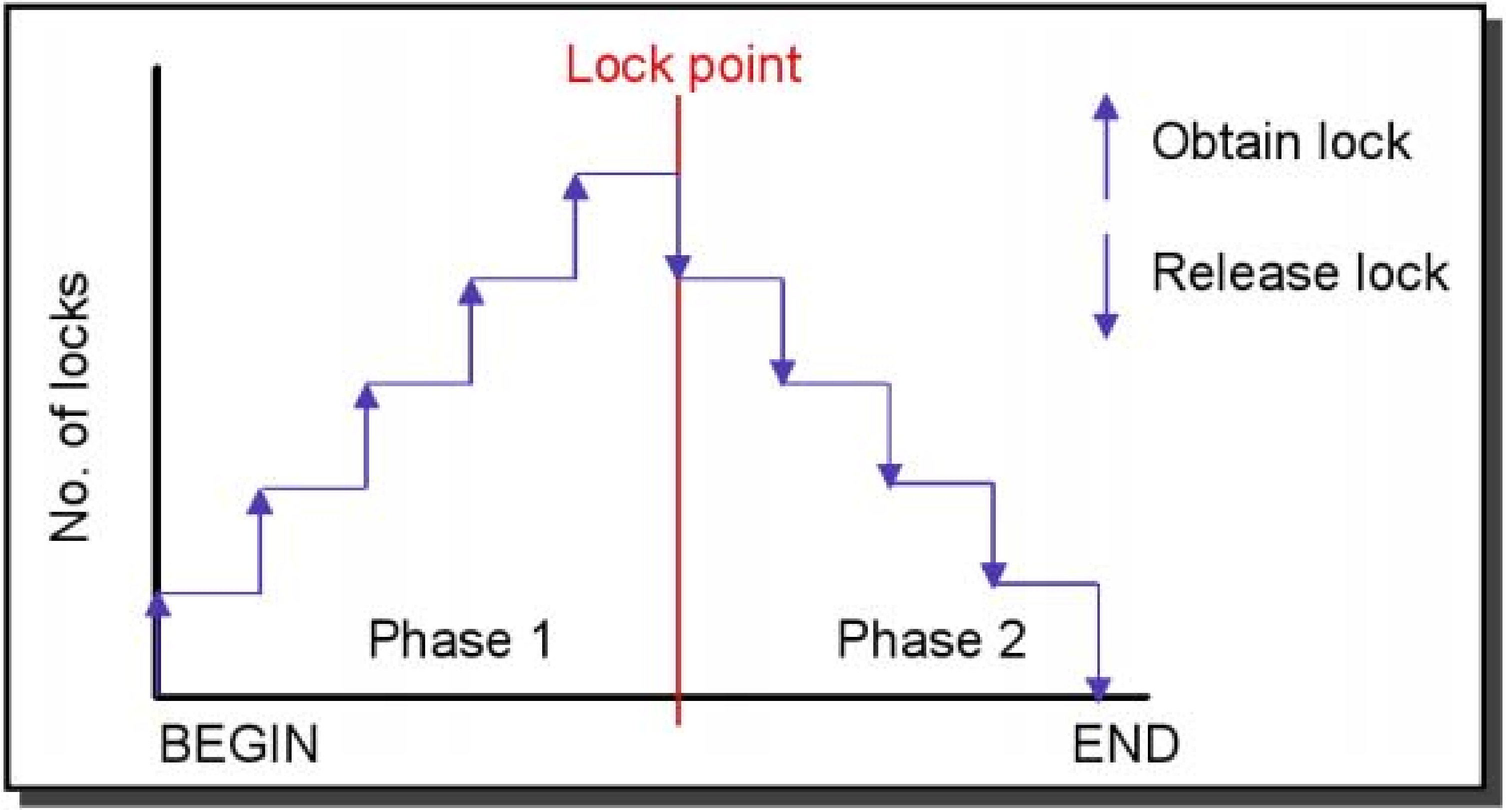
2- Multi-level:

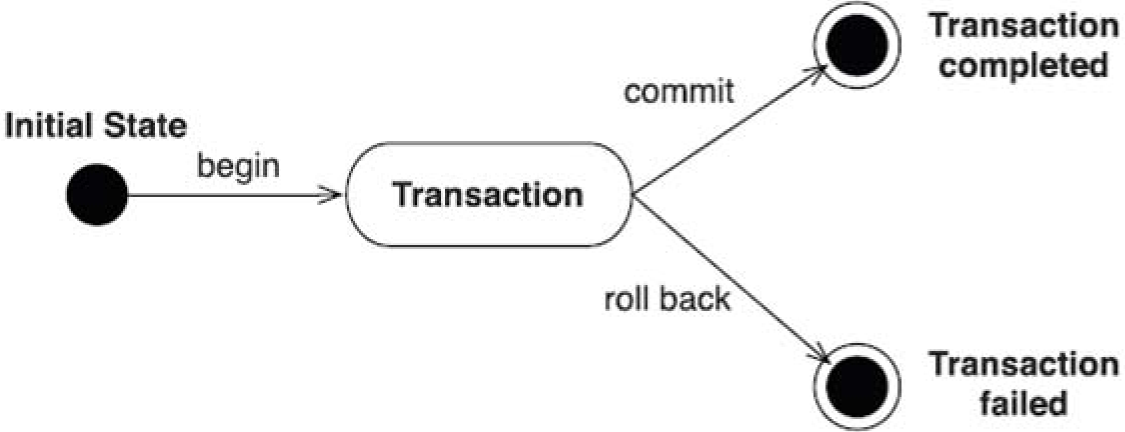
B+-tree


Transaction ACID Properties
ACID
Atomic
"ALL OR NOTHING"
Transaction cannot be subdivided
Consistent
Transaction → transform database from one consistent state to another consistent state
Isolated
Transactions execute independently of one another
Database changes not revealed to users until after transaction has completed
Durable
Database changes are perminent
The permanence of the database's consistent state

Transactions



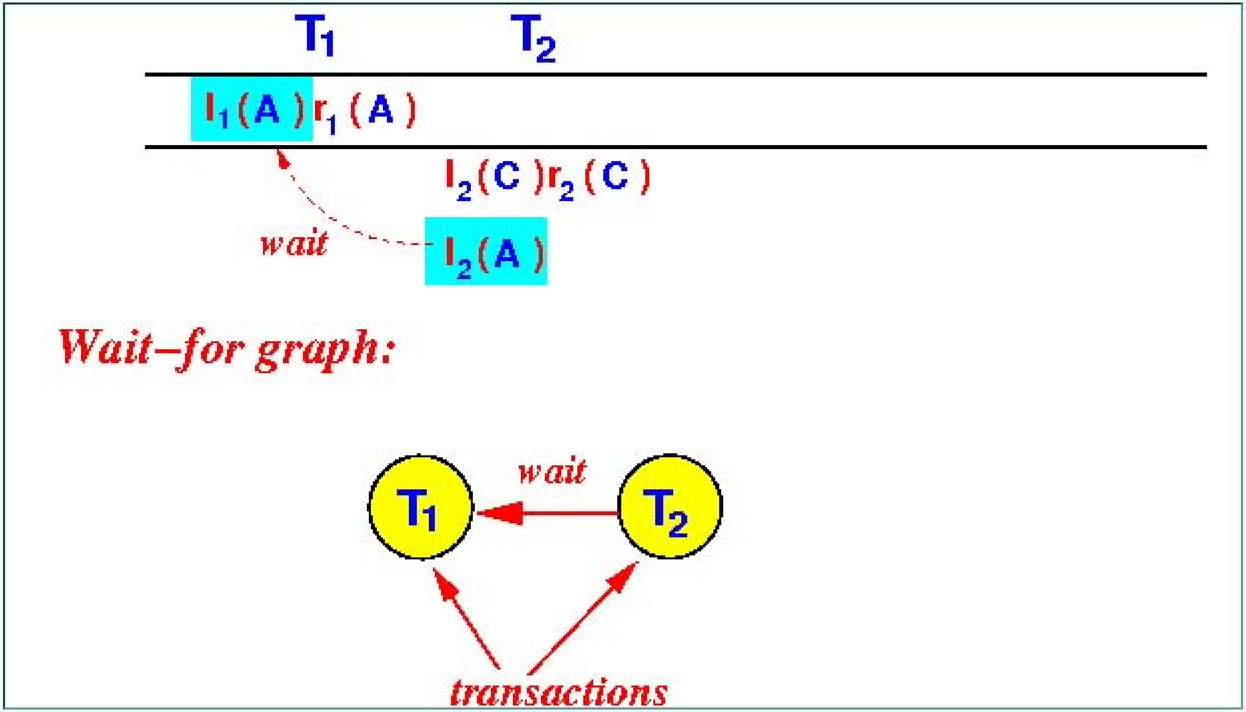

Lock point
Obtain lock
Release lock
No. of locks
Phase 1
Phase 2
BEGIN
END



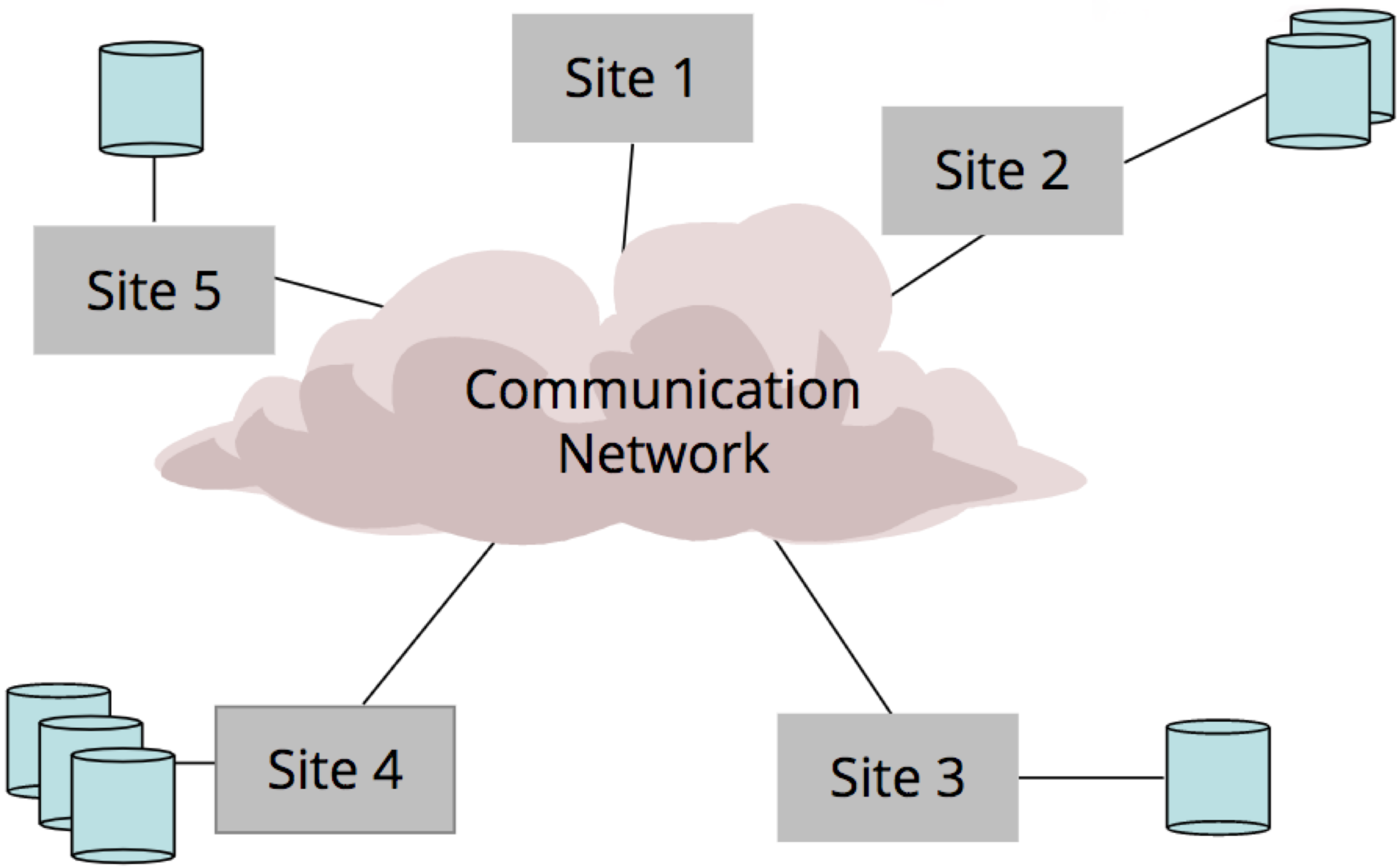
Initial State
begin
Transaction
commit
roll back
Transaction
completed
Transaction
failed

- **Wait-for graph** is a **graph** where:
    - **Node** represents a **transaction**
    - **Edge** $i \Rightarrow j$ represents the **fact** that:
        - The **transaction** i is **waiting** for a **lock** held by the **transaction** j

Example:



Distributed DBMS Environment



Definition

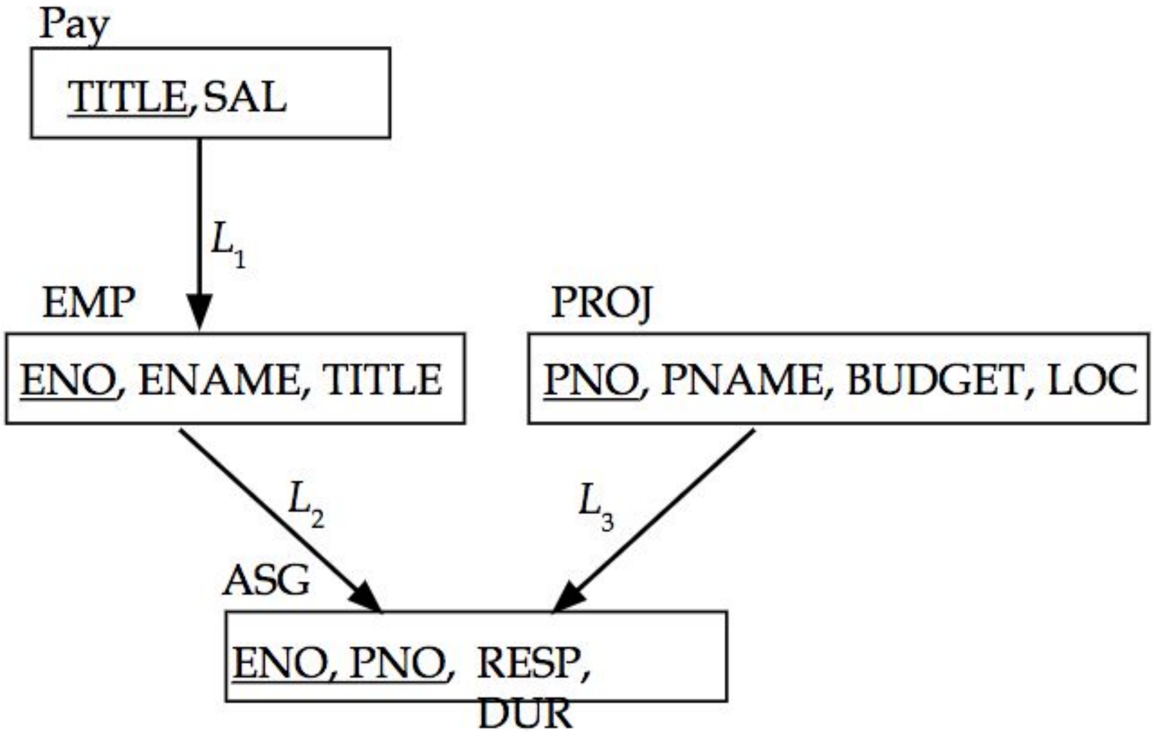
Primary Horizontal Fragmentation is:

$$R_j = \sigma_{F_j}(R),\quad 1 \le j \le w$$

A horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i.

Given a set of minterm predicates M, there are as many horizontal fragments of relation R as there are minterm predicates.

Derived Horizontal Fragmentation (DHF)



Definition

- Given a link L where $owner(L)=S$ and $member(L)=R$, the derived horizontal fragments of R are defined as

$$R_i = R \ltimes_F S_i,\ 1 \le i \le w$$

- where w is the maximum number of fragments that will be defined on R and

$$S_i = \sigma_{F_i}(S)$$

- where F_i is the formula according to which the primary horizontal fragment S_i is defined.

DHF Example

- Given link L_1 where owner(L_1)=PAY and member(L_1)=EMP
  - $EMP_1 = EMP \ltimes PAY_1$
  - $EMP_2 = EMP \ltimes PAY_2$
- Where
  - $PAY_1 = \sigma_{SAL \leq 30000}(PAY)$
  - $PAY_2 = \sigma_{SAL > 30000}(PAY)$

EMP_1

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

DHF Example

- Given link L_1 where owner(L_1)=PAY and member(L_1)=EMP
  - $EMP_1 = EMP \ltimes PAY_1$
  - $EMP_2 = EMP \ltimes PAY_2$
- Where
  - $PAY_1 = \sigma_{SAL \leq 30000}(PAY)$
  - $PAY_2 = \sigma_{SAL > 30000}(PAY)$

EMP_2

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

VF: Information Requirements

Application Information:

Attribute affinities

- a measure that indicates how closely related the attributes are
- This is obtained from more primitive usage data

Attribute usage values

- Given a set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ that will run on the relation: $R[A_1, A_2, \ldots, A_n]$,\

$$use(q_i, A_j) = \begin{cases} 1 \text{ if attribute } A_j \text{ is referenced by query } q_i \\ 0 \text{ otherwise} \end{cases}$$

$use(q_i, \bullet)$ can be defined accordingly

VF – Definition of $use(q_i, A_j)$

Consider the following 4 queries for relation PROJ:

q_1: SELECT BUDGET
FROM PROJ
WHERE PNO=Value

q_2: SELECT PNAME,BUDGET
FROM PROJ

q_3: SELECT PNAME
FROM PROJ
WHERE LOC=Value

q_4: SELECT SUM(BUDGET)
FROM PROJ
WHERE LOC=Value

Let A_1= PNO, A_2= PNAME, A_3= BUDGET, A_4= LOC

$$
\begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array}
\begin{array}{c}
\begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\
\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}
\end{array}
$$

Attribute Usage Matrix

VF – Affinity Measure $aff(A_i, A_j)$

- The attribute affinity measure between two attributes A_i and A_j of a relation $R[A_1, A_2, \ldots, A_n]$ with respect to the set of applications $Q = (q_1, q_2, \ldots, q_q)$ is defined as:

$$aff(A_i, A_j) = \sum_{\text{all queries that access } A_i \text{ and } A_j} \text{(query access)}$$

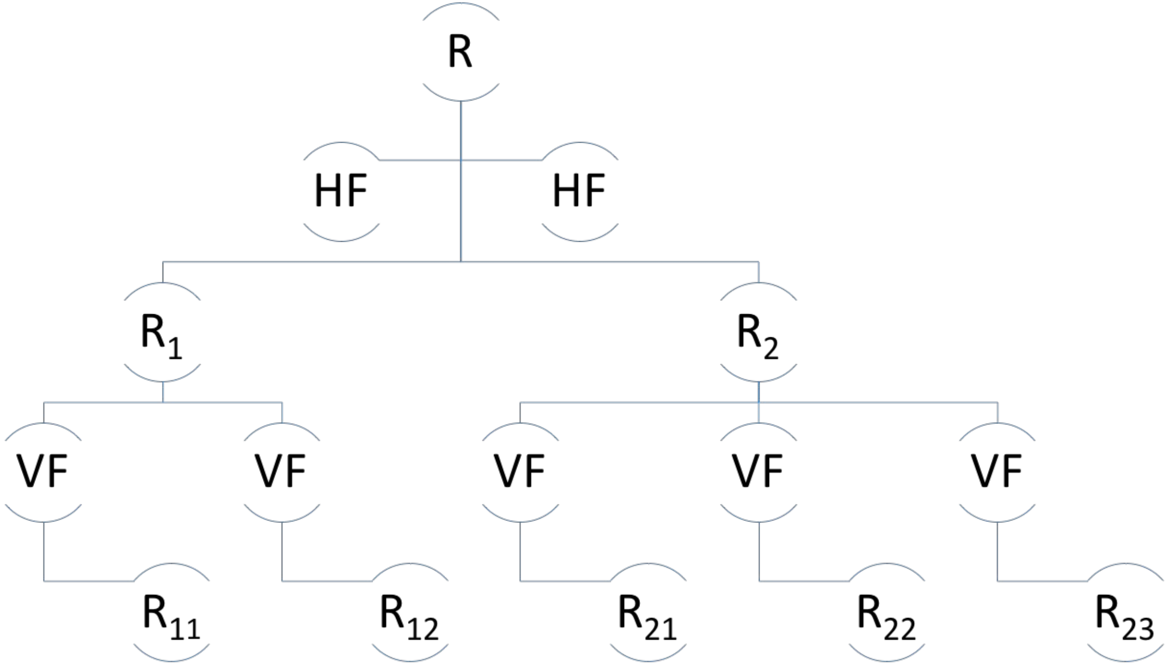
$$\text{query access} = \sum_{\text{all sites}} \text{access frequency of a query} * \frac{\text{access}}{\text{execution}}$$

Attribute Affinity Matrix

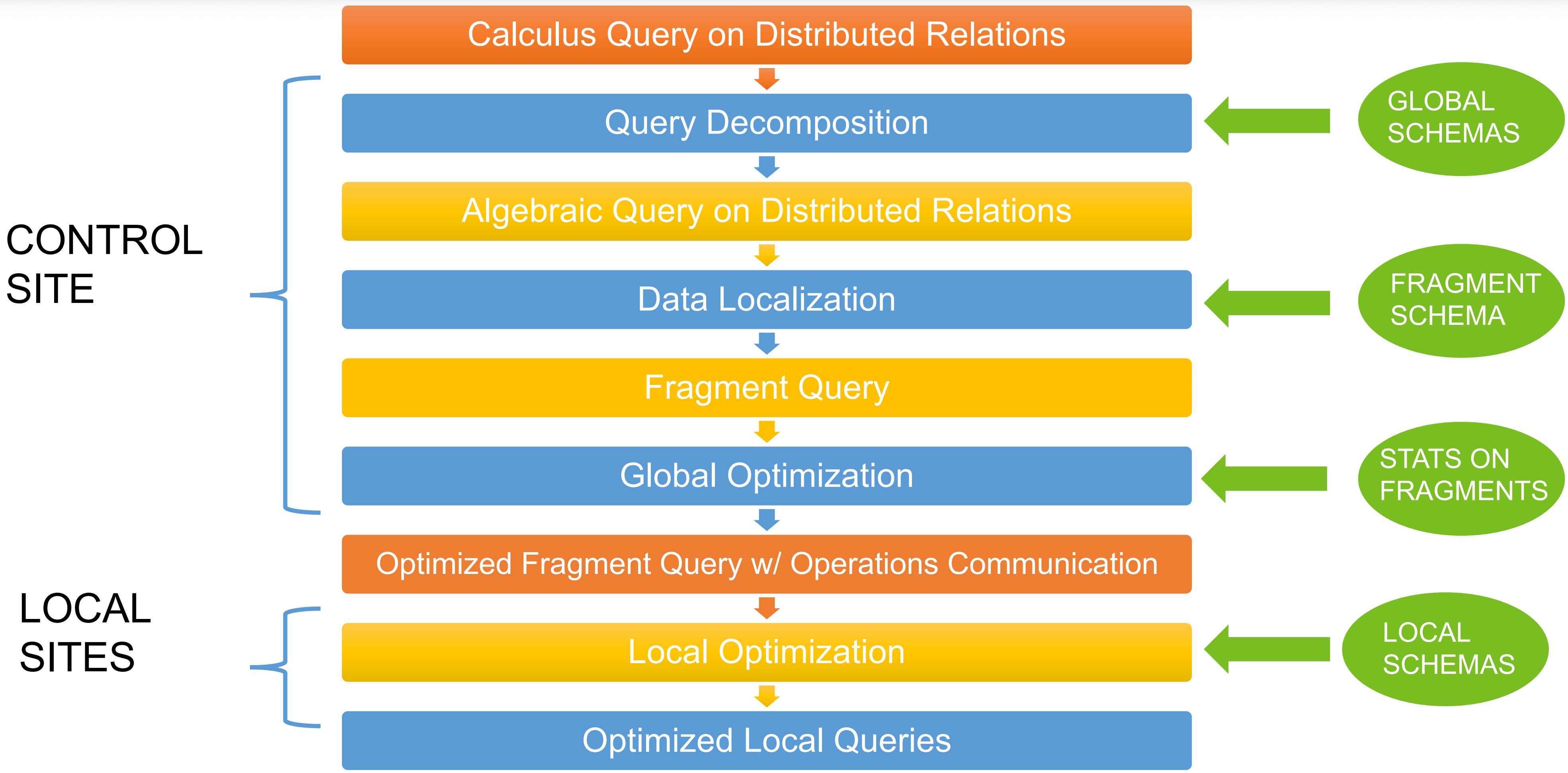
	A1	A2	A3	A4	A5
A1					
A2	50				
A3	45	48			
A4	1	1	0		
A5	0	0	4	75	

$R_1[K, A_1, A_2, A_3]$ $R_2[K, A_4, A_5]$

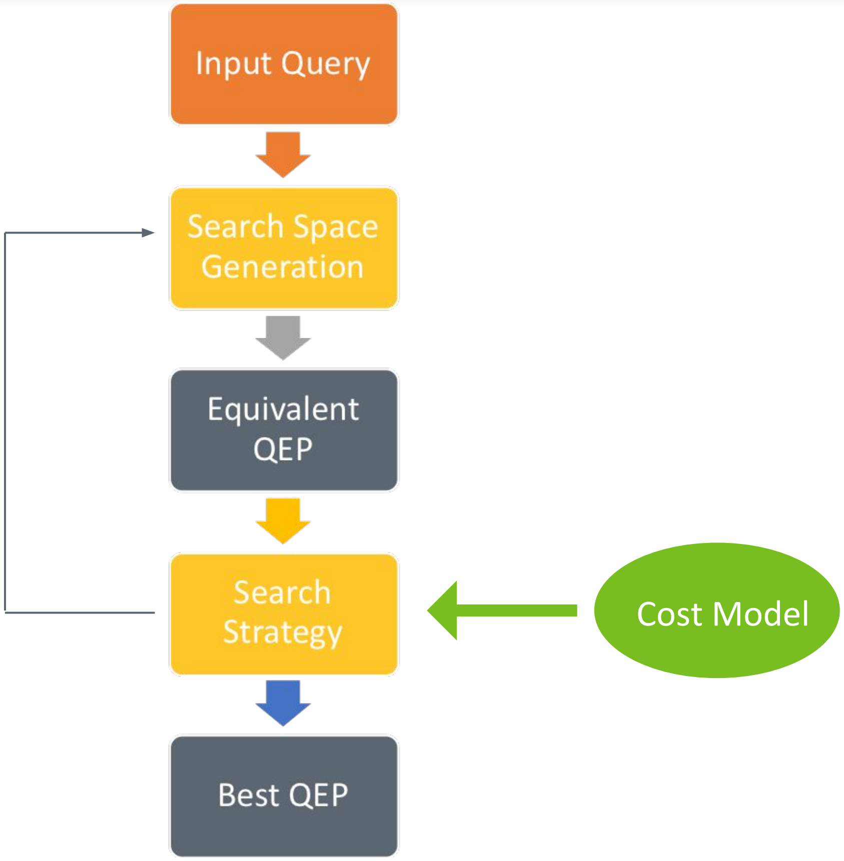
Hybrid Fragmentation



Distributed Query Processing



Query Optimization Process



Total Cost

Total cost= CPU cost + I/O cost + communication cost

CPU cost= unit instruction cost $*$ no. of instructions

I/O cost= unit disk I/O cost $*$ no. of disk I/Os

communication cost = message initiation + transmission

Response Time

Response time= CPU time + I/O time + communication time

CPU time= unit instruction time * no. of sequential instructions

I/O time= unit I/O time*no. of sequential I/Os

communication time= unit msg initiation time*no. of sequential msg

+ unit transmission time*no. of sequential bytes

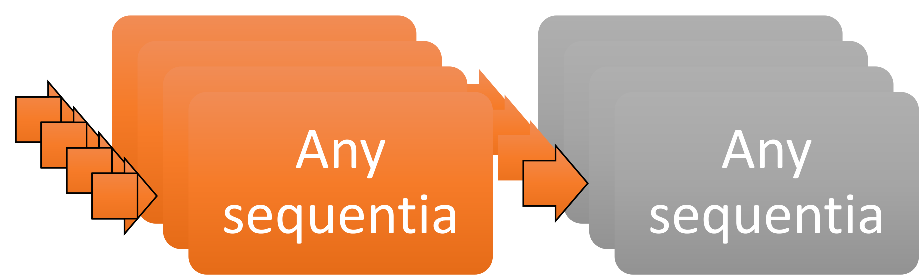
Parallel DBMS

Pipeline Parallelism

– many machines each doing one step in a multi-step process.

Partition Parallelism

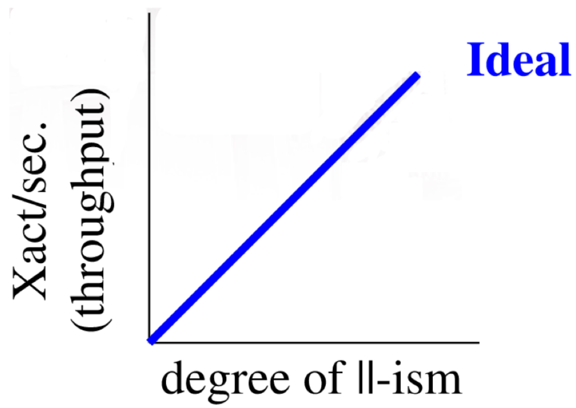
– many machines doing the same thing to different pieces of data.



Terminology

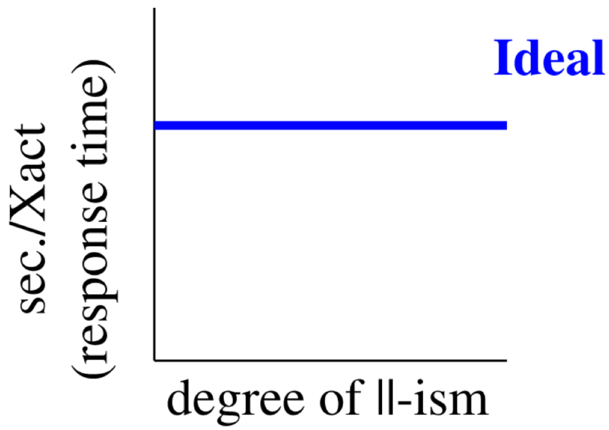
Speed-Up

- More resources means proportionally less time for given amount of data.



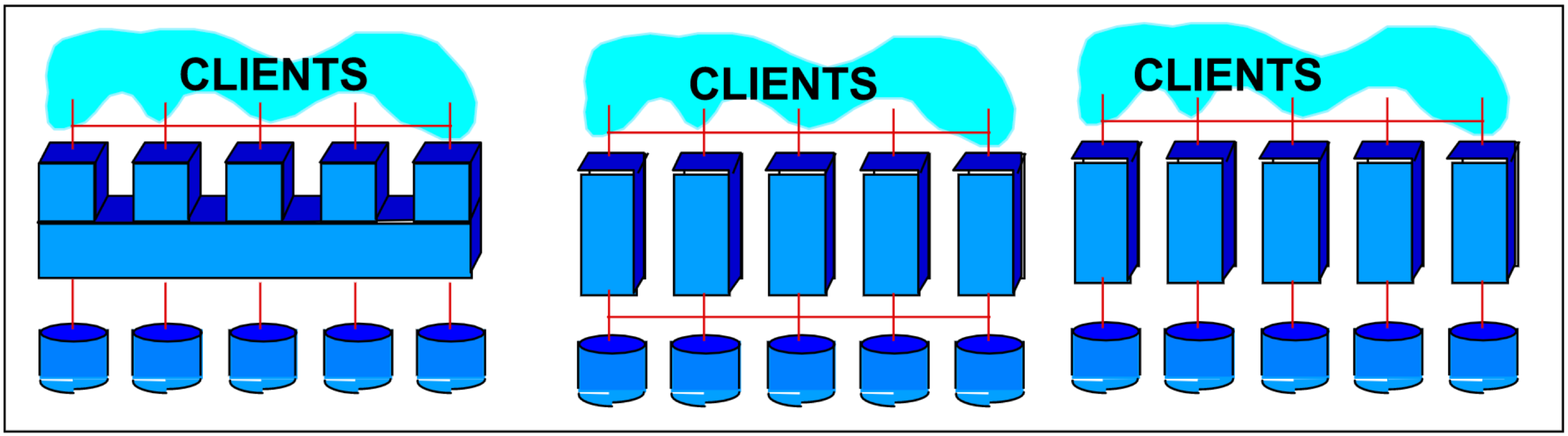
Scale-Up

- If resources increased in proportion to increase in data size, time is constant.



Architecture Issues: Shared What?

Shared Memory (SMP) | **Shared Disk** | **Shared Nothing (network)**



Shared Memory (SMP):
Easy to program
Expensive to build
Difficult to scaleup

Shared Nothing (network):
Hard to program
Cheap to build
Easy to scaleup