Parallel Databases

CS227, Spring 2011

Yang Lu

James Tavares

Overview

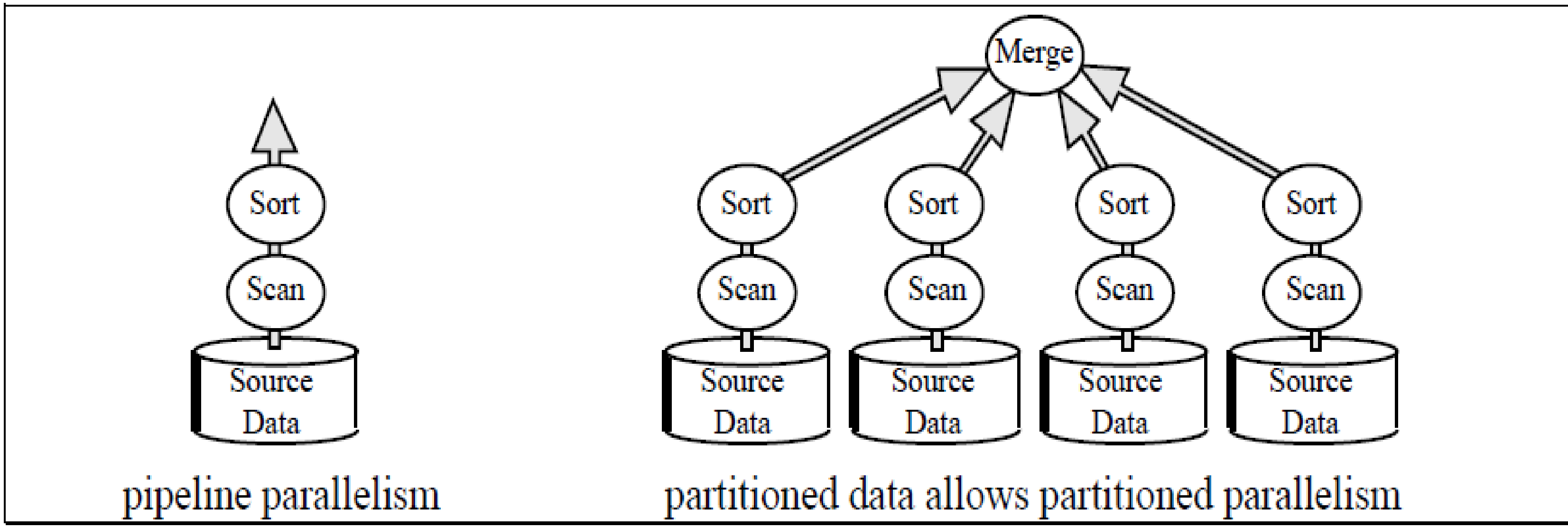
- Motivations
- Architectures
- Partitioning Schemes
- Relational Operator Parallelism
  - Parallel Sort, Join, Selection, etc.
- Gamma
  - Architecture, Performance Analysis
- XPRS Design

Why parallel database ?

- Driving force
  - Demand on storing and analyzing large volumes of data
  - Demand on high throughput for transaction processing

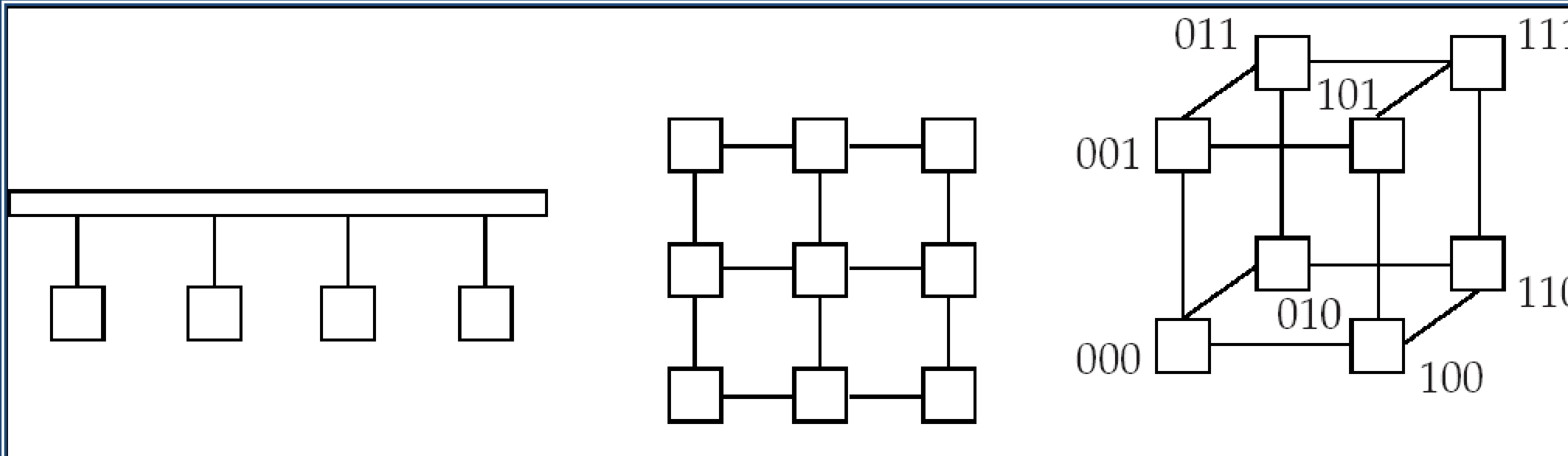
- Prices of microprocessors, memory and disks have dropped sharply
- Relational databases are ideally suited to parallelization.

Relation database parallelization

- 

- Operations can be executed in parallel
  - Pipelined parallelism

Interconnection Networks

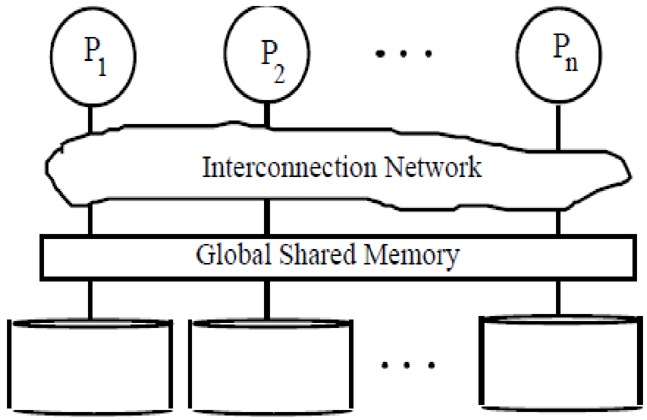


(a) bus (b) mesh (c) hypercube

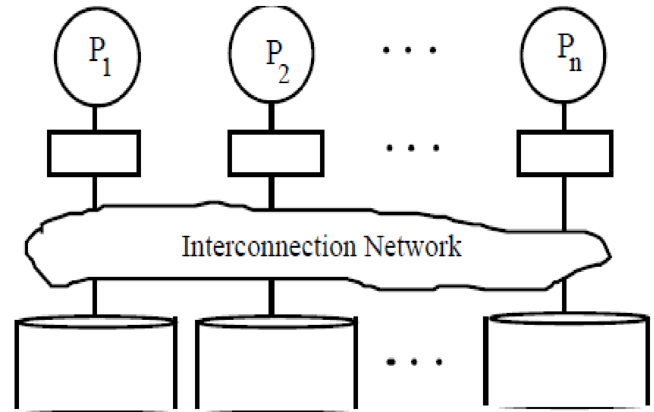
Architectures

- shared-memory:
  - share direct access to a common global.
- shared-disks
  - Each processor has direct access to all disks.
- shared-nothing:
  - The Teradata, Tandem, Gamma

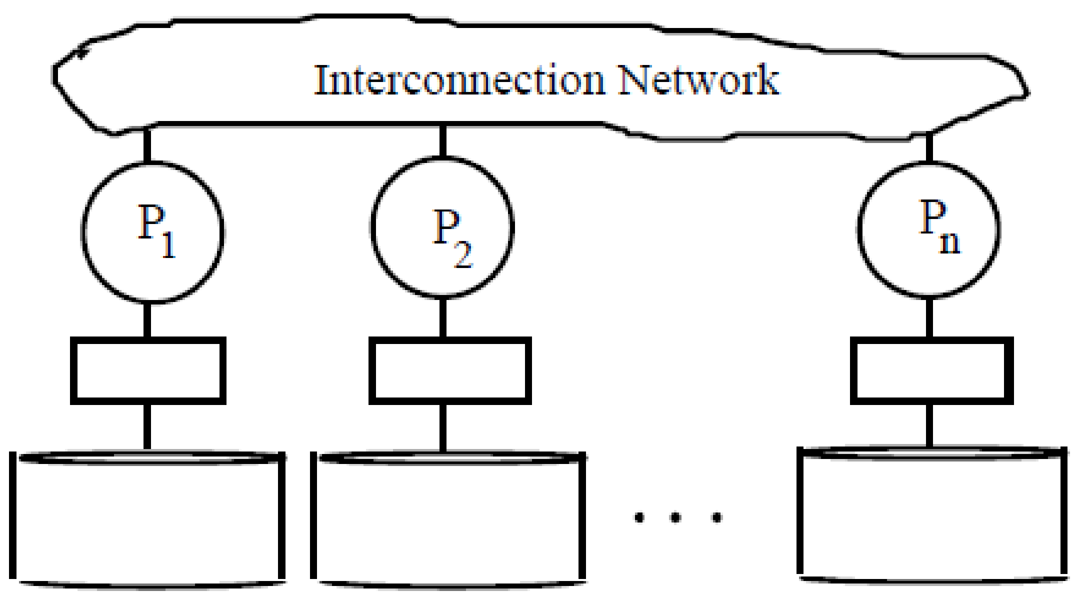
Architectures



Shared Memory Multiprocessor



Shared Disk Multiprocessor



Partitioning a Relation across Disks

- Principles
  - It is better to assign a small relation to a single disk.
  - Large relations are preferably partitioned across all the available disks
    - m disk blocks and n disks
    - should be allocated **min**(m,n) disks
- Techniques
  - Round-robin
  - Hash partitioning
  - Range partitioning

Partitioning Techniques

Round-robin:

Send the i^{th} tuple inserted in the relation to disk i mod *n*.

Hash partitioning:

– Choose one or more attributes as the partitioning attributes.

– Choose hash function *h* with range 0…*n* - 1

– Let *i* denote result of hash function *h* applied to the partitioning attribute value of a tuple. Send tuple to disk *i*.

Partitioning Techniques

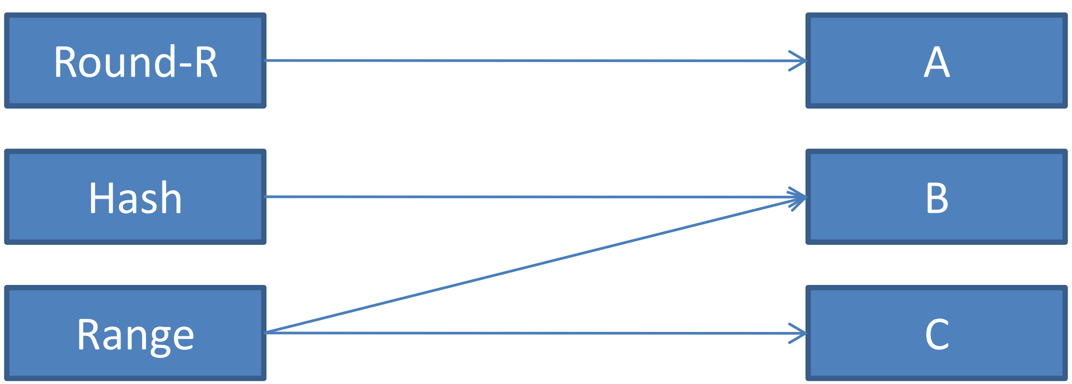
- **Range partitioning**:
  - Choose an attribute as the partitioning attribute.
  - A partitioning vector [v_o, v_1, ..., v_{n-2}] is chosen.
  - Let v be the partitioning attribute value of a tuple. Tuples such that $v_i \leq v_{i+1}$ go to disk i+ 1. Tuples with $v < v_0$ go to disk 0 and tuples with $v \geq v_{n-2}$ go to the last disk.

Comparison of Partitioning Techniques

- A. Sequential scan
- B. Point queries.

  E.g. employee-name=“Campbell”.
- C. Range queries.

  E.g. 10000<salary<20000



Parallelism Hierarchy

- Interquery
  - Queries/transactions execute in parallel with one another
    - Locking and logging must be coordinated by passing messages between processors.
    - Cache-coherency has to be maintained
- Intraquery
  - Execution of a single query in parallel on multiple processors

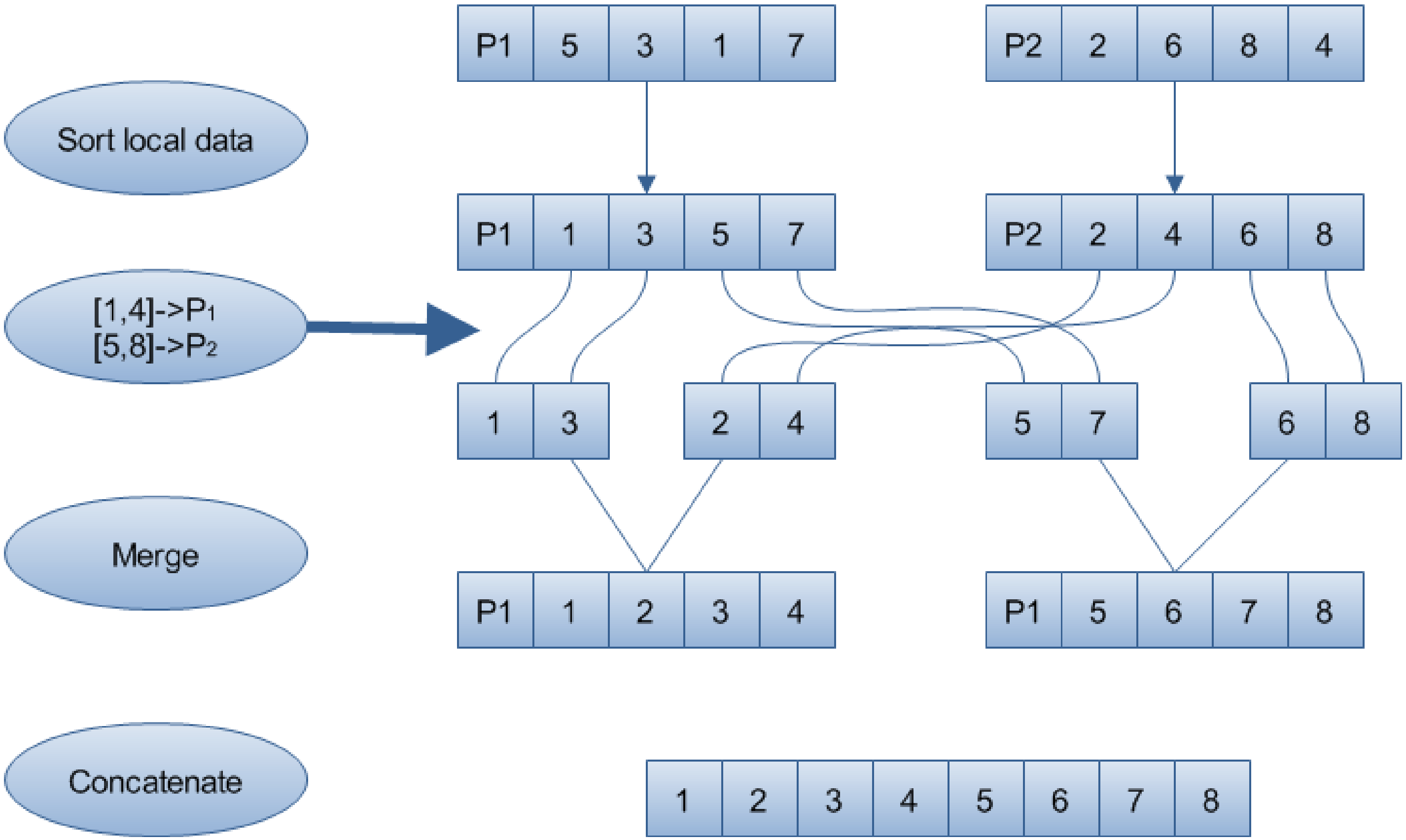
Parallelism Hierarchy

- Two complementary forms of intraquery parallelism:
  - **Intraoperation Parallelism** – parallelize the execution of each individual operation in the query.
  - **Interoperation Parallelism** – execute the different operations in a query expression in parallel.

Parallel Sort

- Range-Partitioning Sort
  - Redistribution using a range-partition strategy
  - Each processor sorts its partition locally
- Parallel External Merge-Sort
  - Each processor P_i locally sorts the data on disk D_i.
  - The sorted runs on each processor are then merged.

Parallel External Merge-Sort

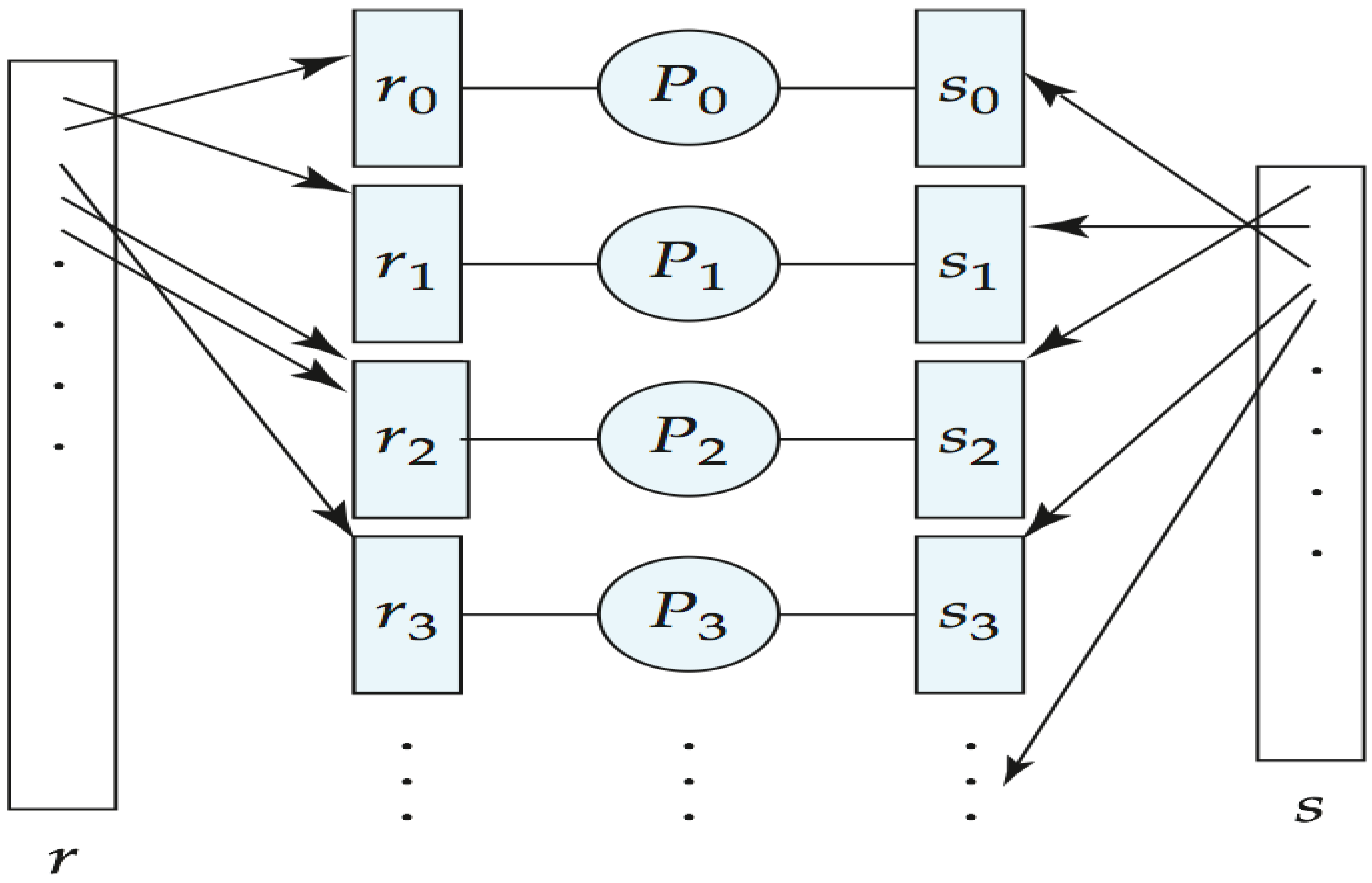


Parallel Join

- Partitioned Join
  - Use the same partitioning function on both relations
    - Range partitioning on the join attributes
    - Hash partitioning on the join attributes

  - Equi-joins and natural joins

Partitioned Join



Partitioned Parallel Hash-Join

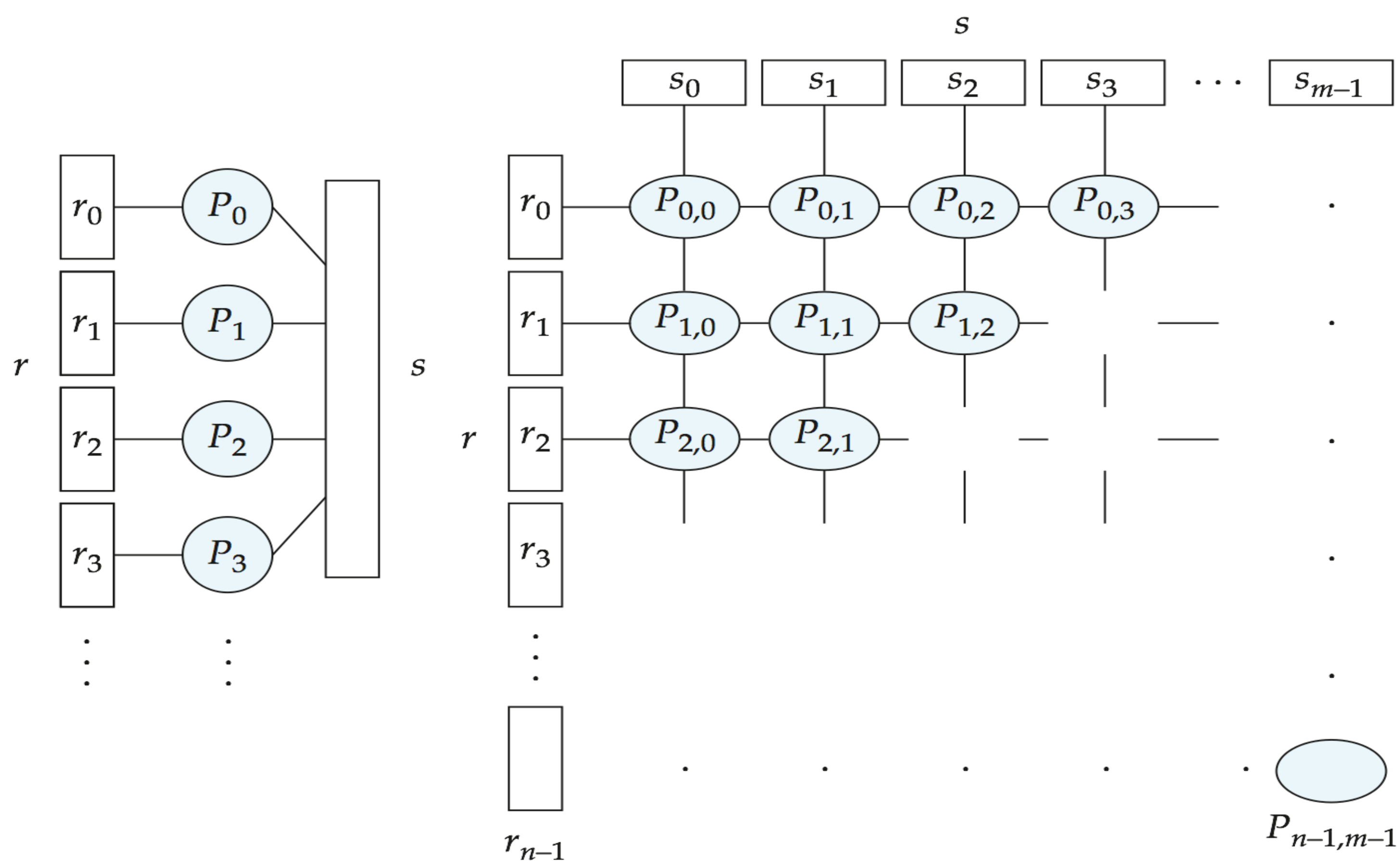
- Simple Hash-Join
  - Route tuples to their appropriate joining site.
  - The smaller joining relation staged in an in-memory hash(which is formed by hashing on the join attribute of each tuple).
  - Tuples of the larger joining relations probe the hash table for matches.
- Other optimization: Hybrid Hash-Join

Parallel Join

- Fragment-and-Replicate Join
  - Partitioning not possible for some join conditions
    - E.g., non-equijoin conditions, such as r.A > s.B.

  - **fragment and replicate** technique

Fragment-and-Replicate Join



(a) Asymmetric fragment and replicate

(b) Fragment and replicate

Interoperator Parallelism

- Pipelined Parallelism
  - The output tuples of one operation are consumed by a second operation.
  - No need to write any of the intermediate results to disk.

Pipelined parallelism

– Consider a join of four relations

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

- Let P1 be assigned the computation of temp1 = $r_1 \bowtie r_2$
- Let P2 be assigned the computation of

  temp2 = temp1 $\bowtie$ r_3
- And P3 be assigned the computation of temp2 $\bowtie$ r_4

Measuring DB Performance

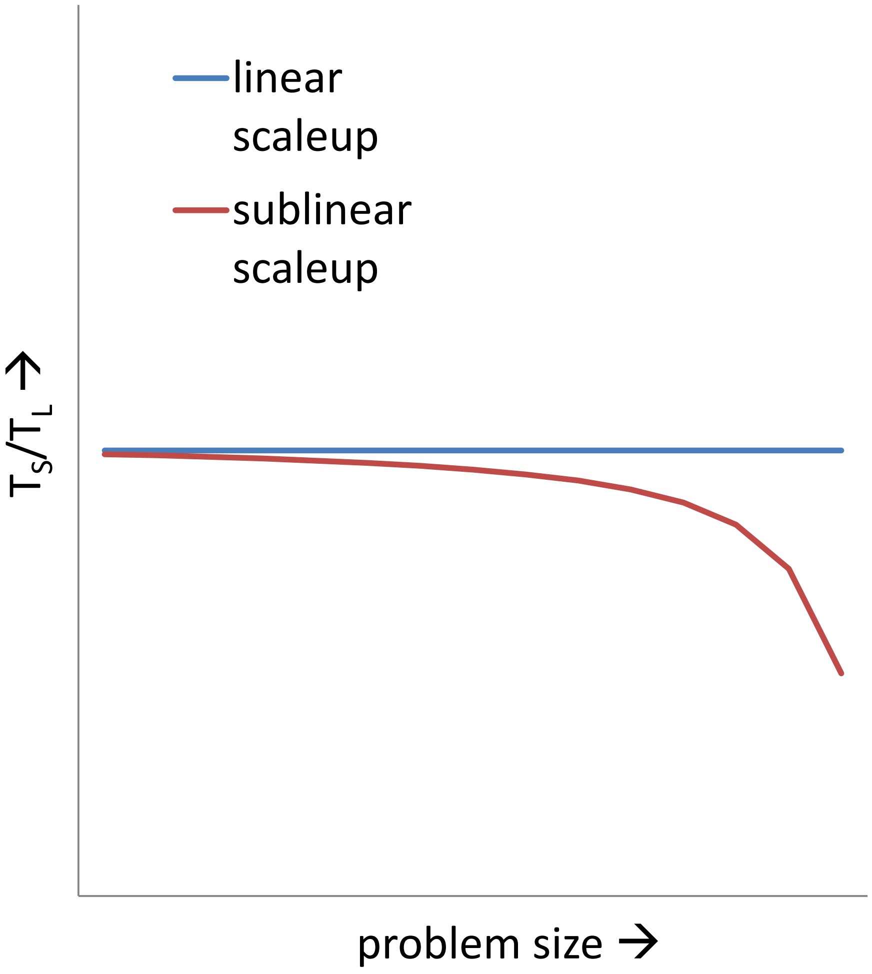
- Throughput
  - The number of tasks, or the size of task, that can be completed in a given time interval
- Response Time
  - The amount of time it takes to complete a single task from the time it is submitted
- Goal: improve both through parallelization

Absolute vs. Relativistic

- Absolute
  - Q: Does system meet my requirements?
  - Q: How does system compare with system Y?
- Relativistic
  - As some resource is varied, determine how system **scales** and how **speed** is affected
  - Q: Will increased resources let me process larger datasets?
  - Q: Can I speed up response time by adding resources?

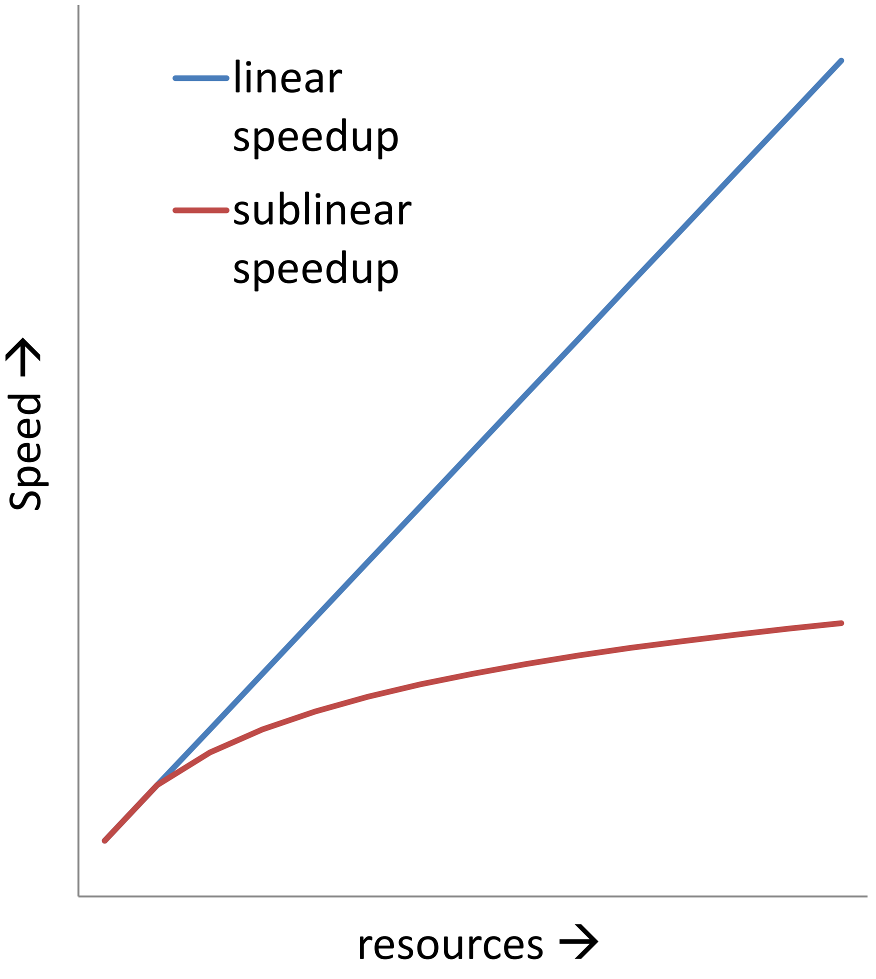
Scaleup

- Baseline: Task Q runs on M_S in T_S seconds
- Task Q_N runs on M_L in T_L seconds
- Q_N, M_L are N times larger than Q, M_S, respectively
- Scaleup = T_S/T_L
  - Linear: $T_S = T_L$
  - Sublinear: $T_L > T_S$



Speedup

- Task Q runs on M_S and responds in time T_S
- Same task Q runs on M_L and responds in time T_L
  - Goal: T_L should be time: $T_S * (S/L)$
- Speedup = T_S/T_L



Performance Factors

- Interference
  - Parallel processes compete for shared resources (e.g., system bus, network, or locks)
- Start-up costs
  - Associated with initiating a single process
  - Start-up time may overshadow processing time
- Skew
  - Difficult to subdivide tasks in to equal-sized parts
  - Most-skewed subtask governs response time

Gamma Overview

- First operational prototype 1985, U. of Wisconsin
- Shared-nothing architecture
  - Interconnected by communications network
  - Promotes commodity-based hardware, lots of processors
- Hash-based parallel algorithms to disburse load

Gamma Hardware

- Version 1.0
  - (18) VAX 11/750 machines, with 2MB RAM
  - 8 machines with 333 MB HD; balance is diskless
  - 80mbit/s token ring, 4mbit/s at each CPU
- Version 2.0
  - 32x Intel 386 iPSC/2 hypercube CPUs, with 8MB RAM
  - 330 MB HDD per CPU
  - 8 x 22.4Mbps/s serial hypercube channels

Gamma Storage Engine

- Horizontally Partitioned
  - Round robin, hashed, or range partitioned
  - For performance analysis:
    - Hashed for source relations
    - Round-robin for destination relations
- Clustered and non-clustered indexes offered *within each partition*
  - Clustered index allowed on non-partition attribute

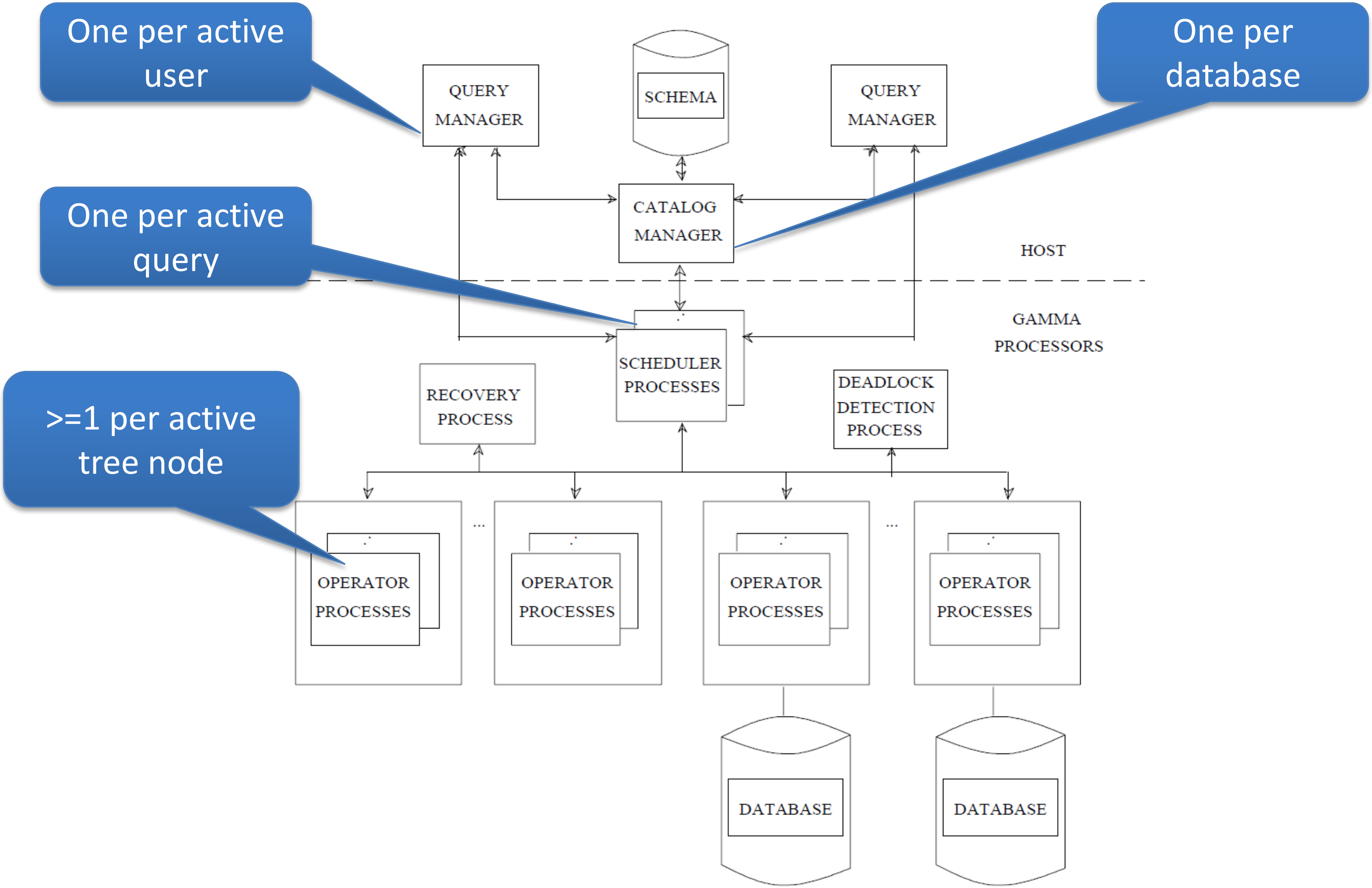
Recovery: Chained Declustering

- Assume N nodes, and N fragments of R, R_N
- Backup copy stored at node: (i+1) mod N
- On failure, nodes assumes 1/(N-1) of the load
- Multiple failures permitted as long as no two adjacent nodes fail together

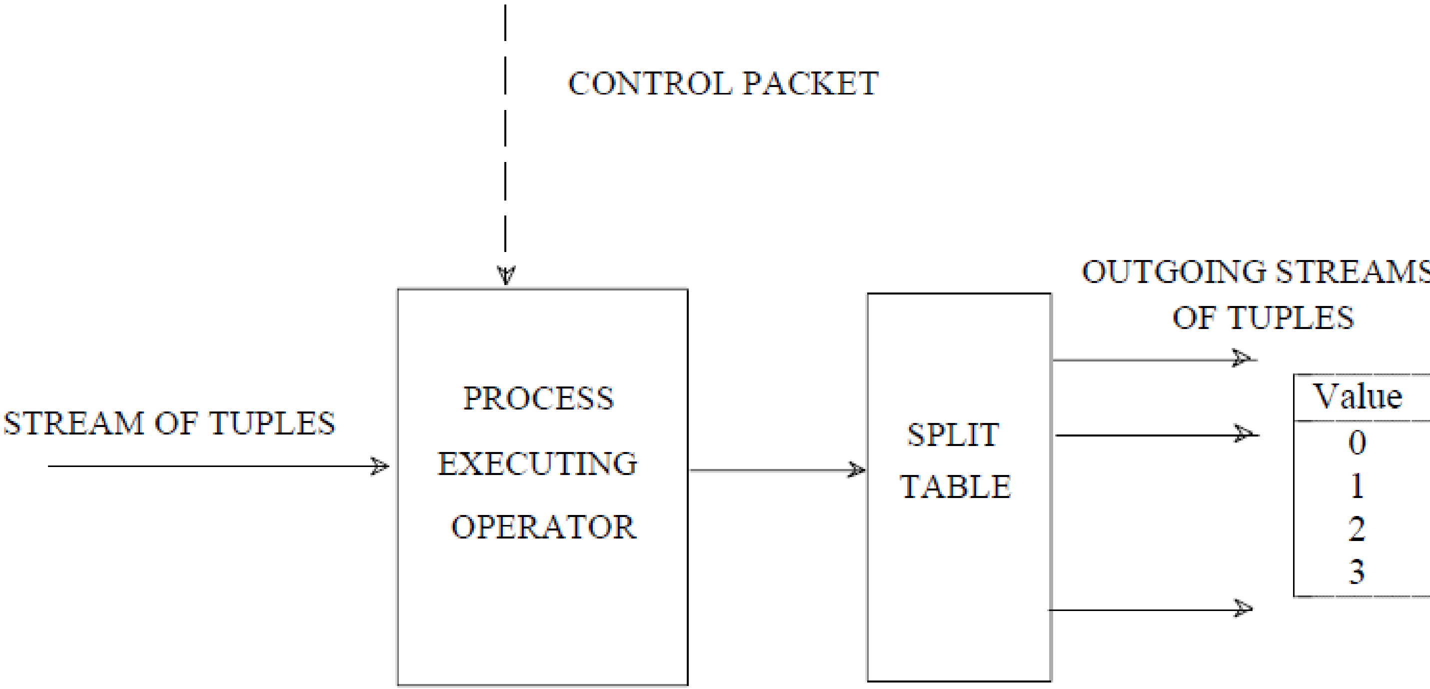
Node	0	1	2	3	4	5	6	7
Primary Copy	R0	R1	R2	R3	R4	R5	R6	R7
Backup Copy	r7	r0	r1	r2	r3	r4	r5	r6

Node	0	1	2	3	4	5	6	7
Primary Copy	R0	---	$\frac{1}{7}$R2	$\frac{2}{7}$R3	$\frac{3}{7}$R4	$\frac{4}{7}$R5	$\frac{5}{7}$R6	$\frac{6}{7}$R7
Backup Copy	$\frac{1}{7}$r7	---	r1	$\frac{6}{7}$r2	$\frac{5}{7}$r3	$\frac{4}{7}$r4	$\frac{3}{7}$r5	$\frac{2}{7}$r6

Gamma Architecture



Gamma Operator & Split Table

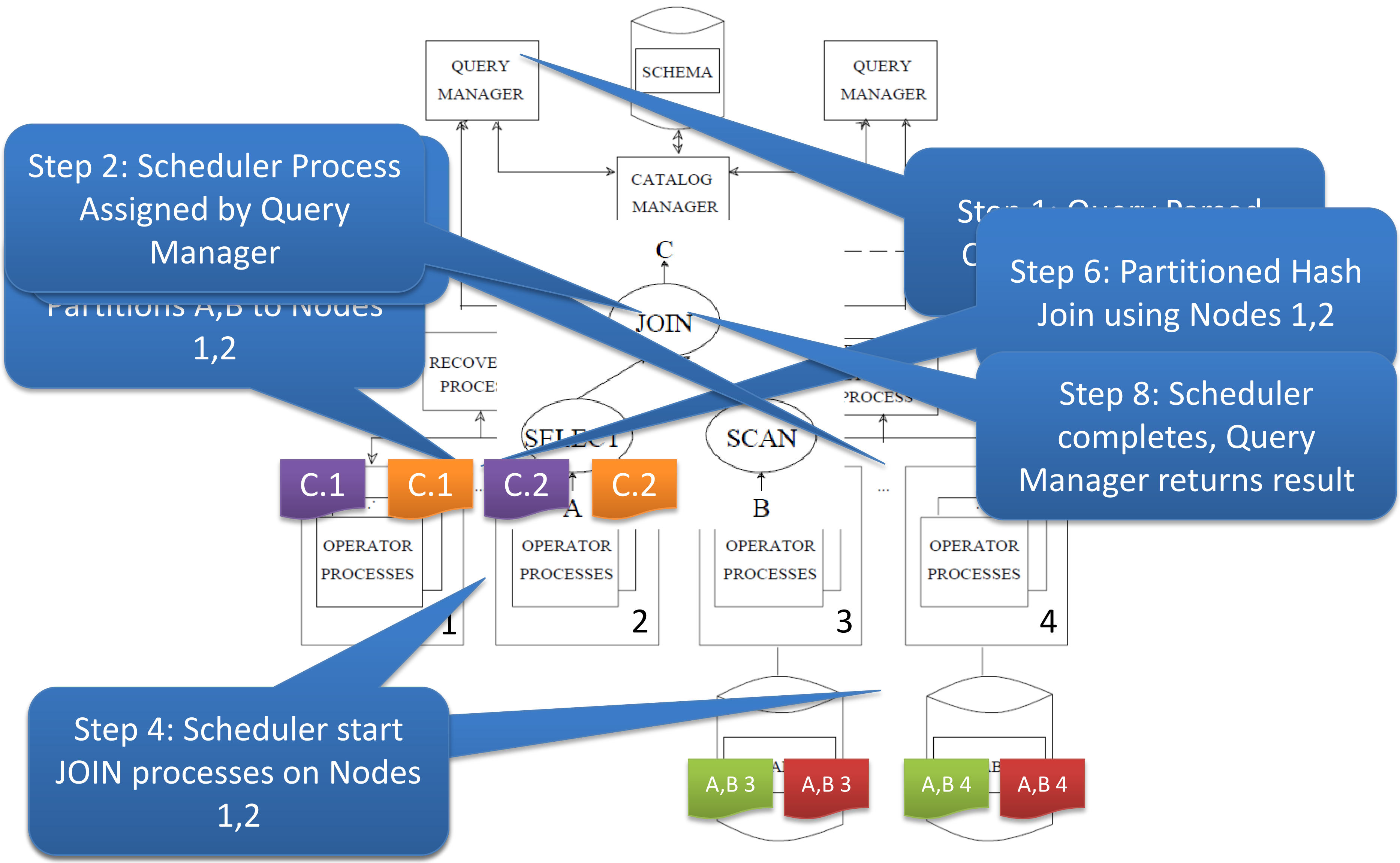


Value	Destination Process
0	(Processor #3, Port #5)
1	(Processor #2, Port #13)
2	(Processor #7, Port #6)
3	(Processor #9, Port #15)

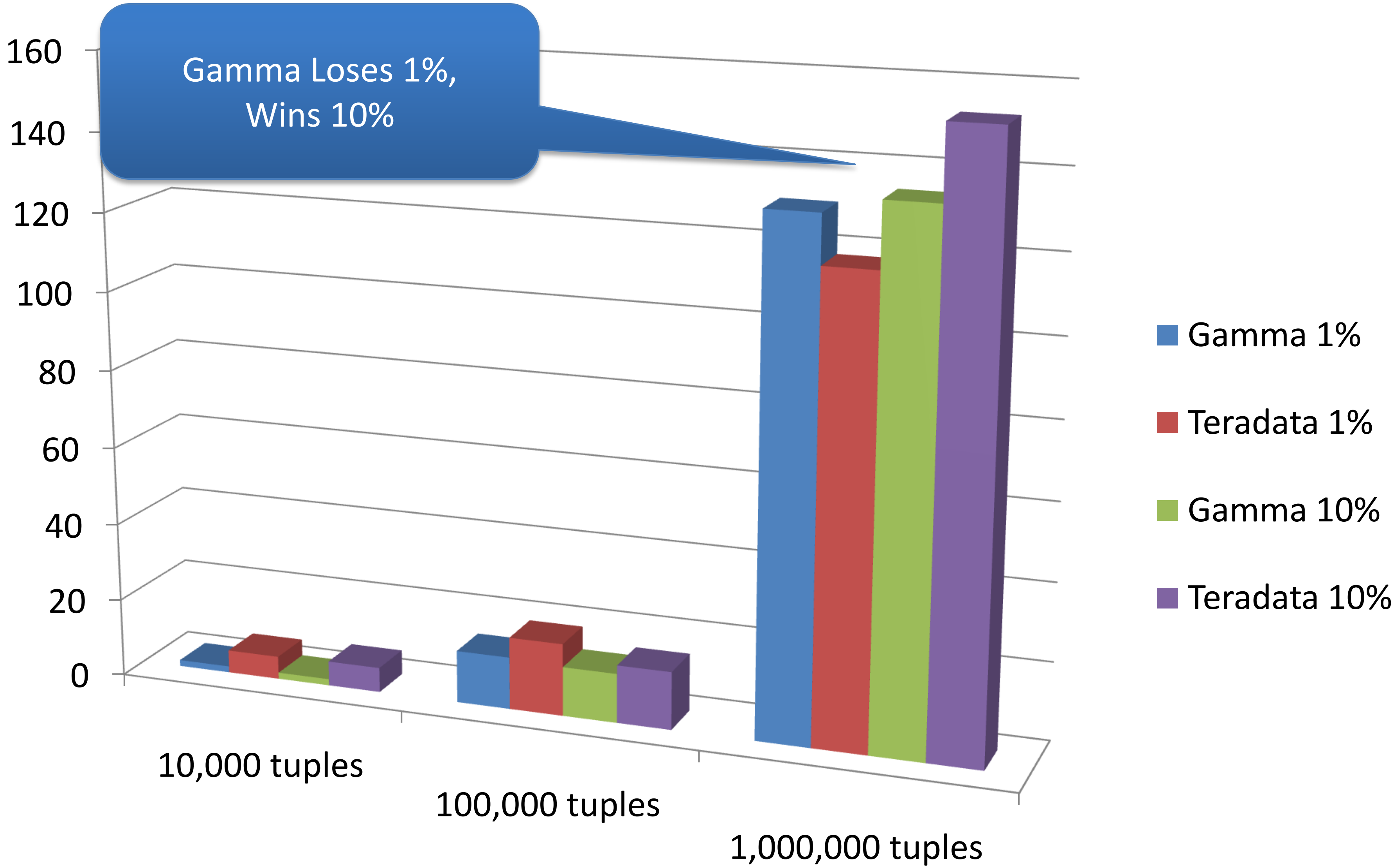
Operators Include:

SCAN, SELECT, JOIN, STORE, UPDATE, etc

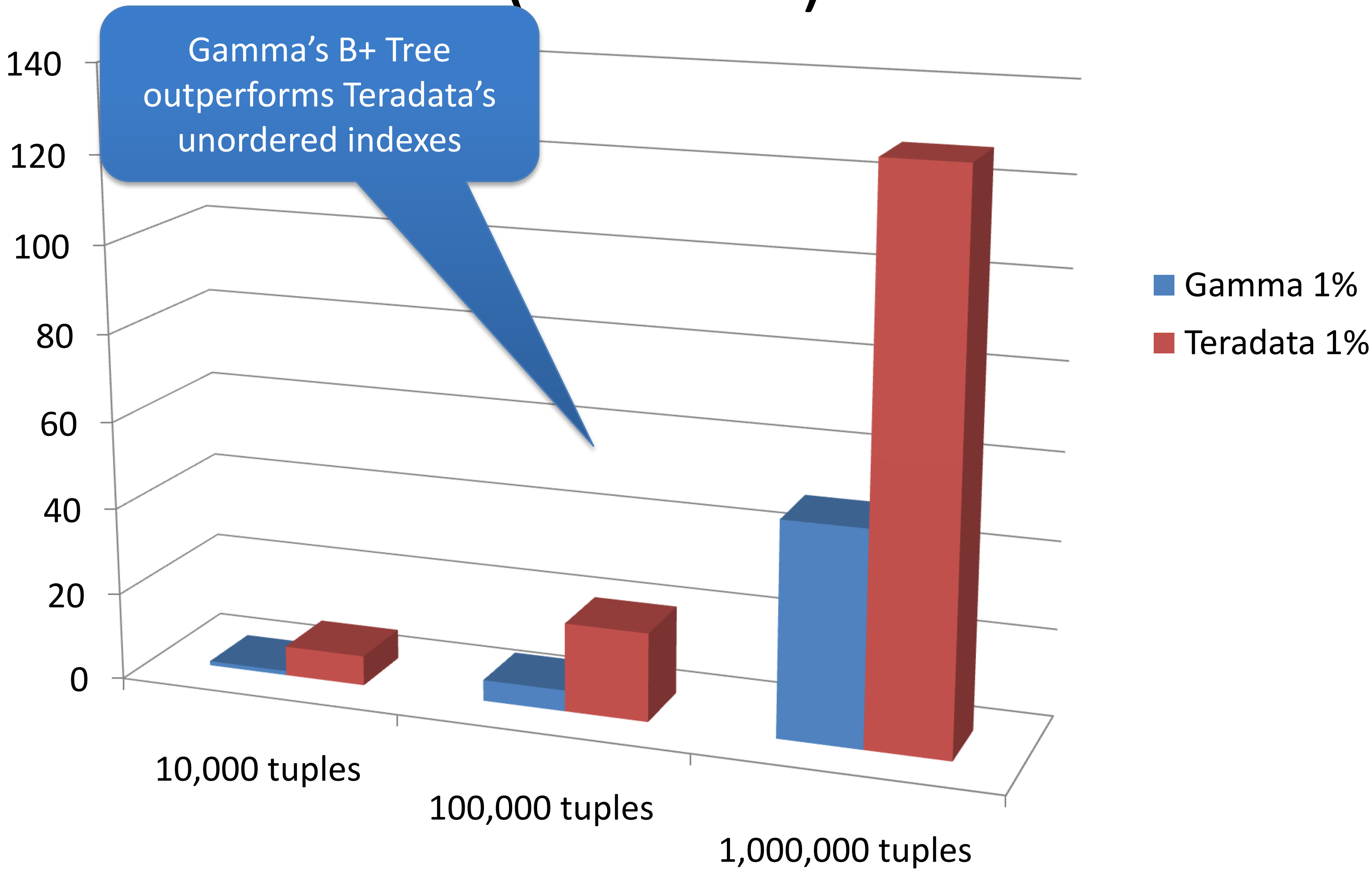
Example Query



Nonindexed Selections (seconds)

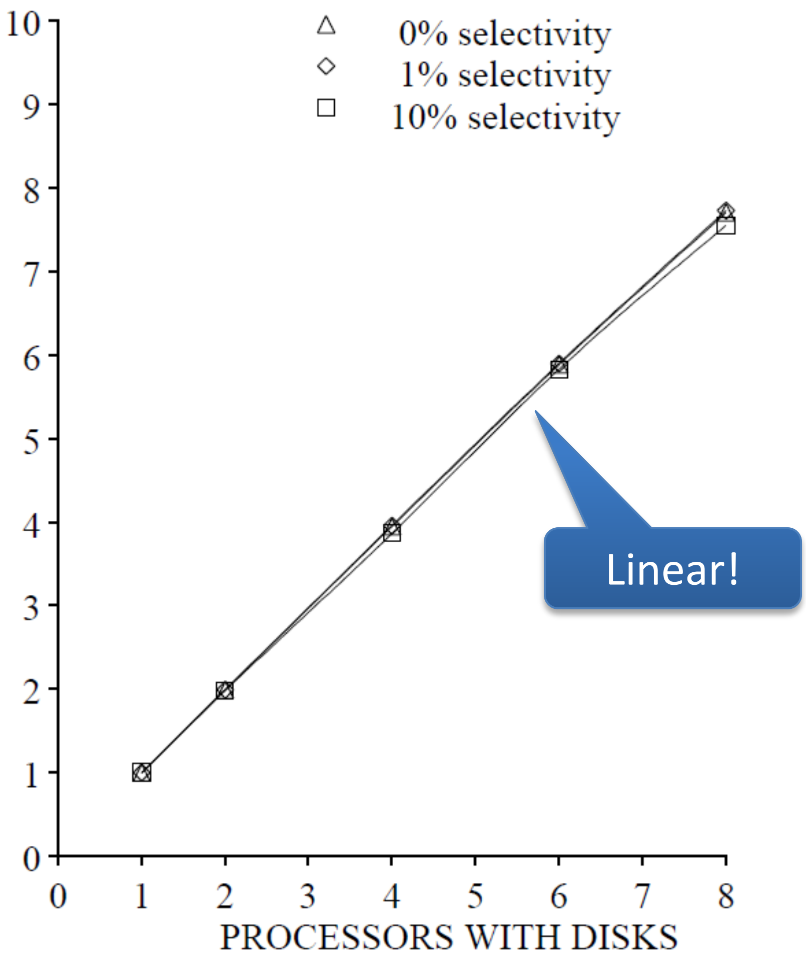


Non-clustered Indexed Selections (seconds)

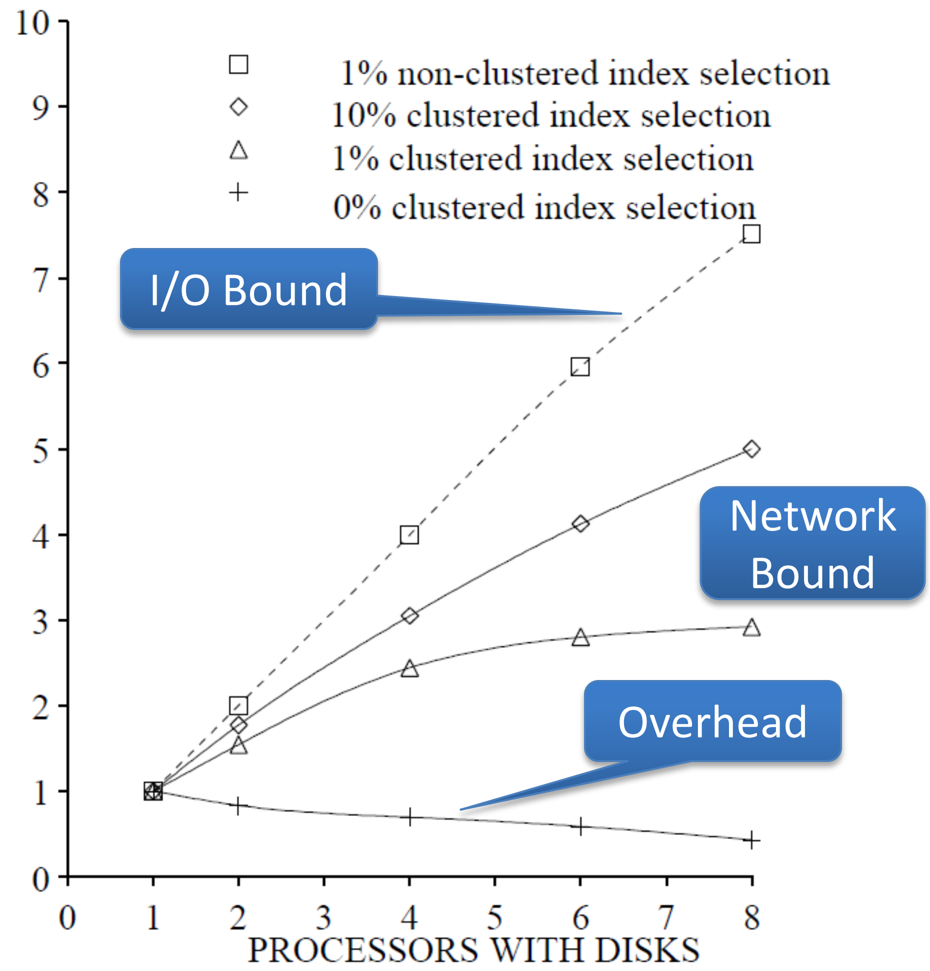


Selection Speedup

Nonindexed Selection



Indexed Selection

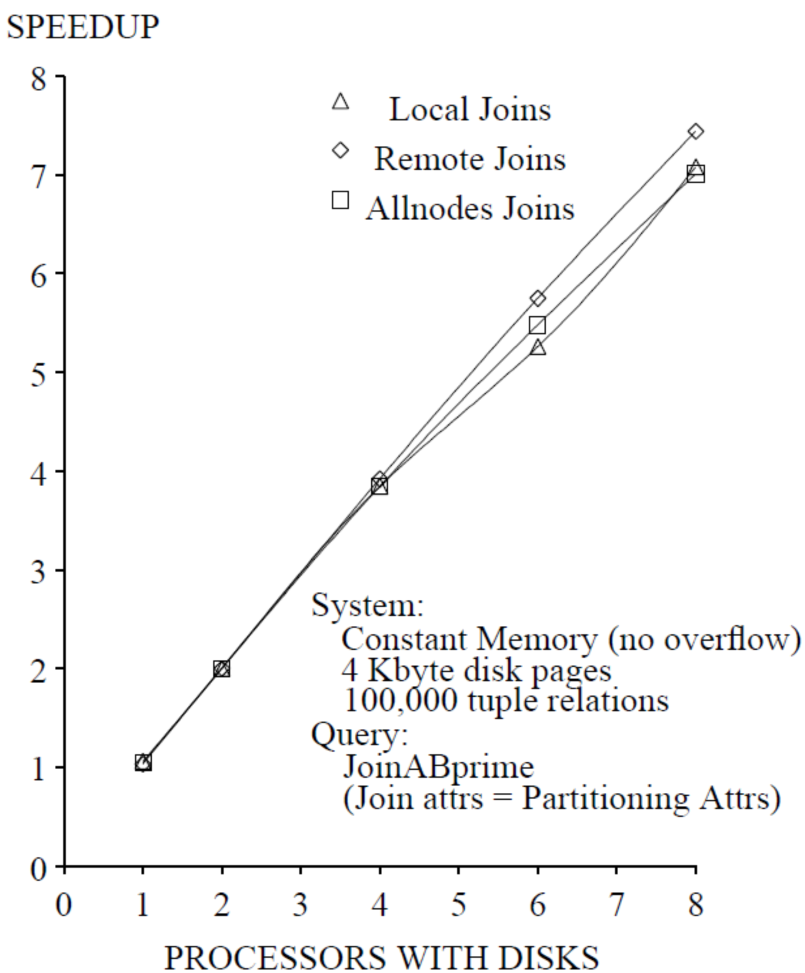


Gamma Join Performance

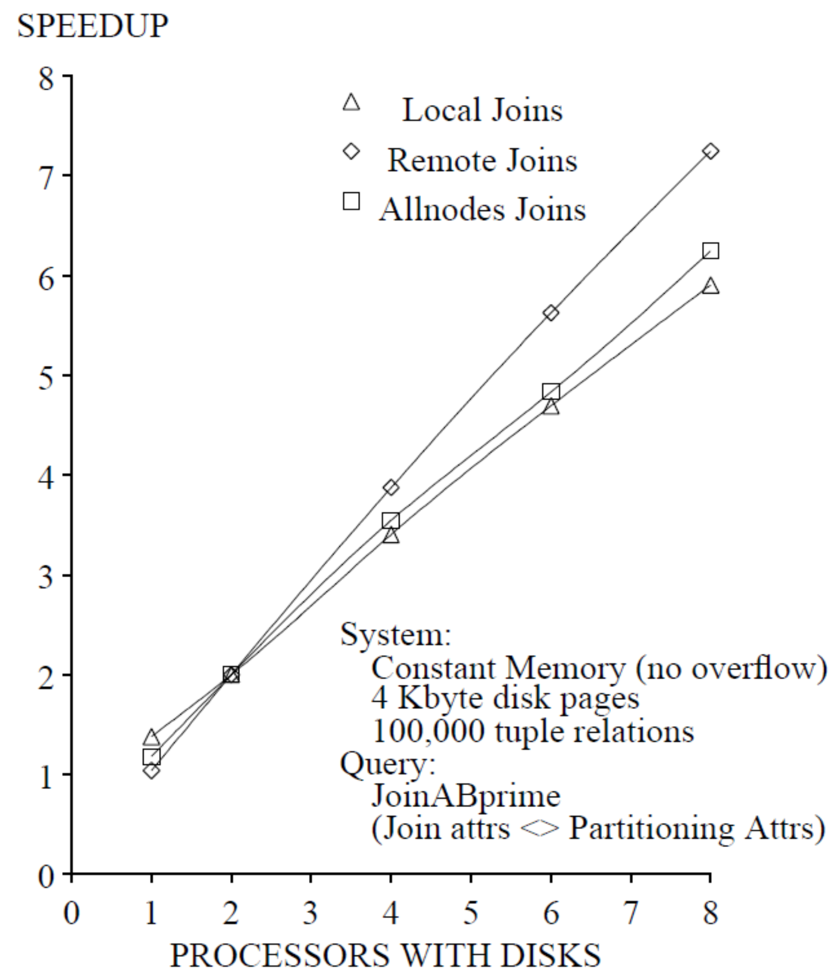
- Relations
  - A – 100,000 tuples
  - Bprime – 10,000 tuples
  - A $\bowtie$ Bprime – 10,000 tuples
- Join Types
  - Local
    - join occurs only on disk nodes
  - Remote
    - join occurs only on disk-less nodes
  - Allnodes
    - join occurs on both disk and disk-less nodes
  - Scans always run on respective disk node

Join A,Bprime Speedup

Join Attr = Partitioning Attr

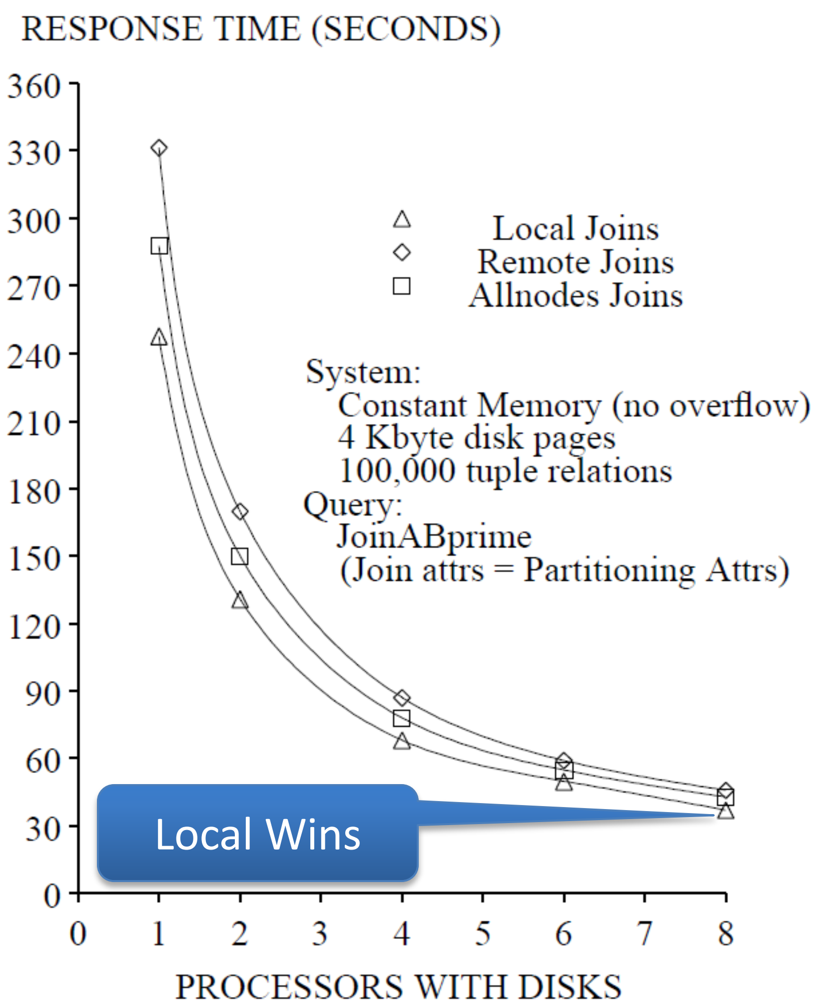


Join Attr != Partitioning Attr

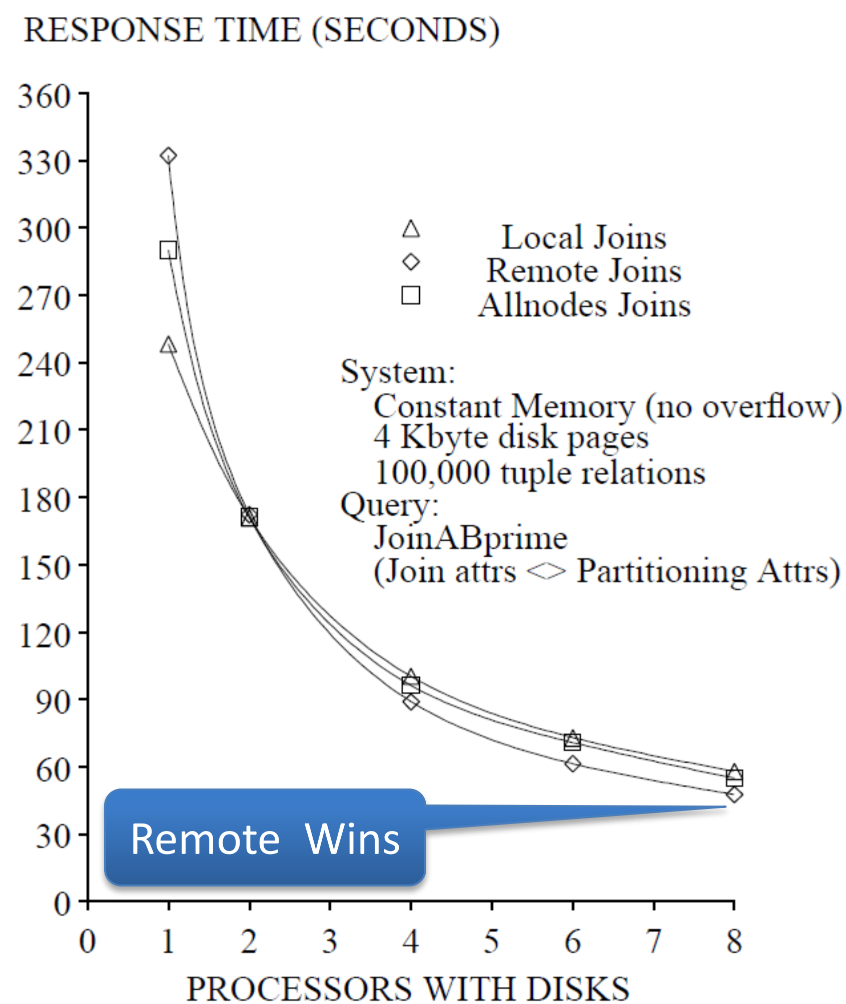


Join A,Bprime Response Time

Join Attr = Partitioning Attr

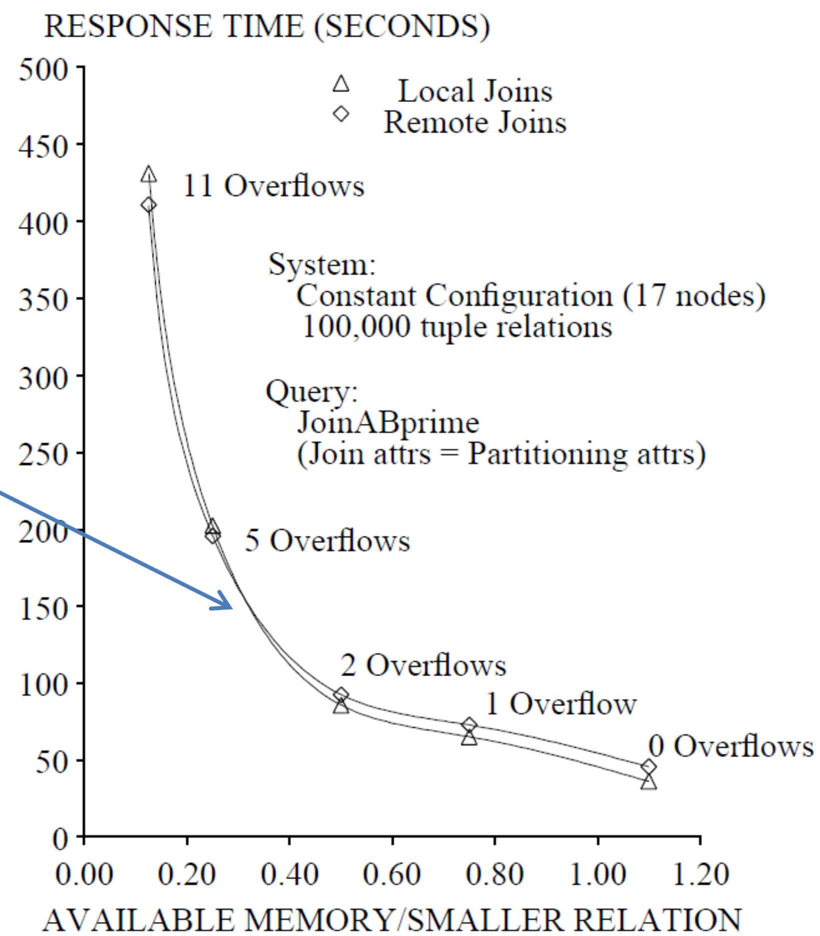


Join Attr != Partitioning Attr



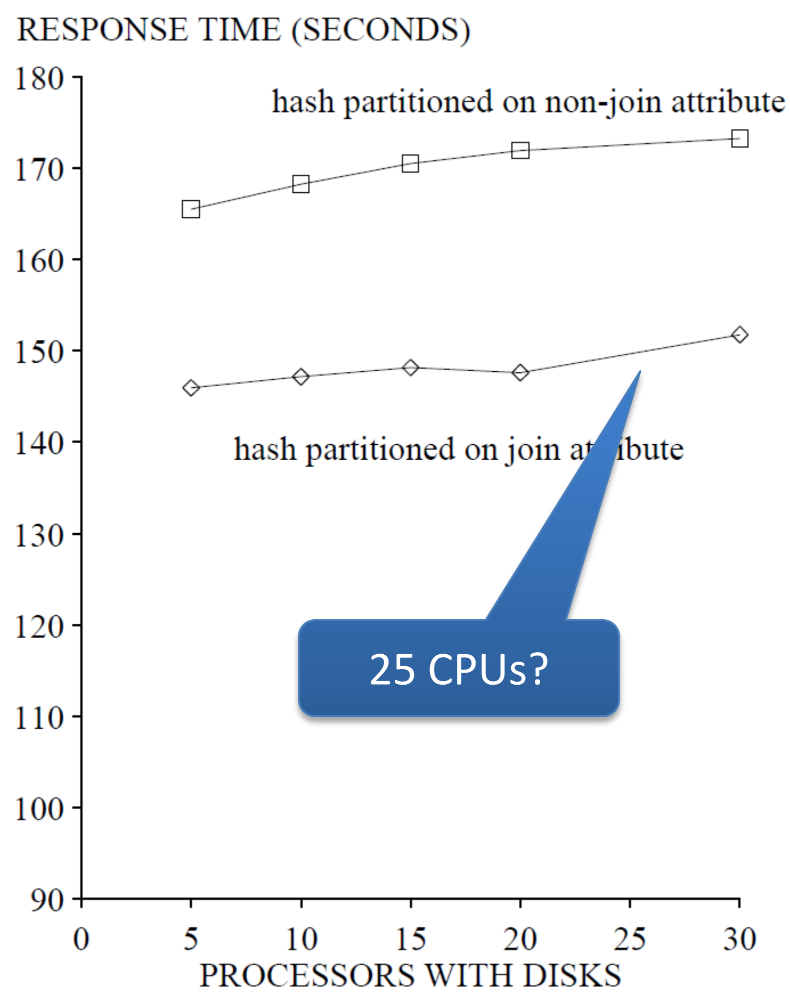
Gamma Join Overflow Performance

- Simple Hash Join w/ Join Attr. = Part. Attr
- Memory was incrementally reduced
- Performance crossover
- Why? Overflows handled by recursive joins
  - With new hash function!
  - New hash equiv. of:
    Join Attr. != Part. Attr



Gamma (V2) Scaleup – Join A,Bprime

- Intel Hypercube
- Ranges
  - CPUs: [5, 30]
  - "A" relation: [1M, 6M]
  - "Bprime" relation: [100k, 600k]
- Factors
  - Scheduler on single CPU
  - Diminished short-circuiting
  - Communications network

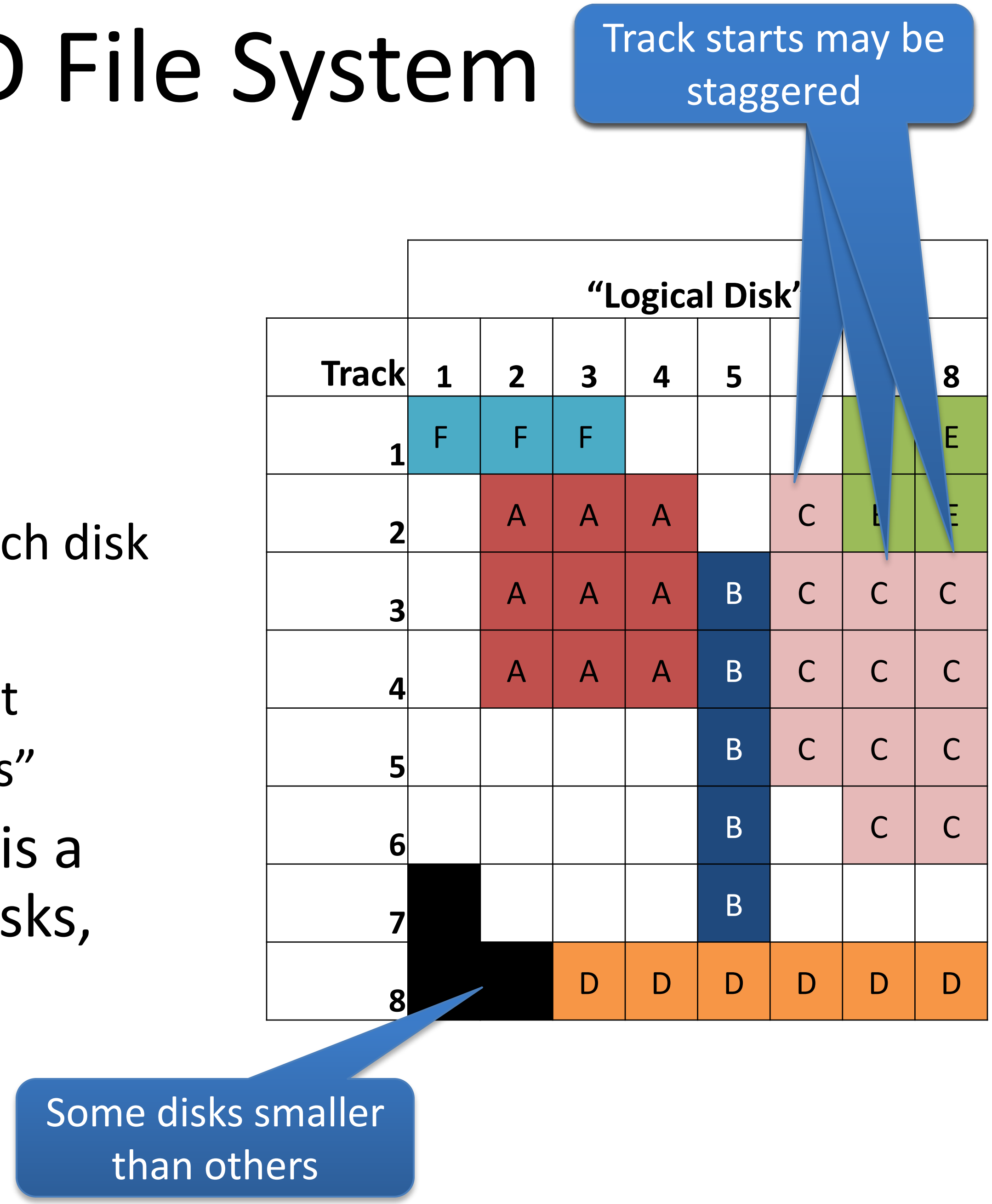


XPRS Overview

- Proposed extension to POSTGRES
  - 2-D file allocation, and RAID
  - Parallel queries, fast path, partial indexes
  - Special purpose concurrency
  - Parallel query plans
- Architecture
  - Shared-memory (faster than network)
  - General-purpose OS
  - Large Sun machine or a SEQENT Symmetry

2-D File System

- A file is defined by:
  - Starting disk
  - Width, in disks
  - Height, in tracks
  - Starting track on each disk
- Larger Widths
  - Increase throughput
  - Minimize “hot spots”
- Each “Logical Disk” is a group of physical disks, protected by RAID5



Changes to POSTQUEL

- **Parallel** keyword alerts DBMS to statements that can be executed in parallel (inter-query parallelism)
  - RETRIEVE… *PARALLEL* RETRIEVE… *PARALLEL* RETRIEVE…
- **Fast Path**
  - Allow users to define stored procedures, which run pre-compiled plans with given arguments
  - Bypass: Type checking, parsing, and query optimization
- **Partial Indexes**
  - E.g.: INDEX on EMP(salary) WHERE age < 20
  - Reduces index size, increases performance
- **Range-partitioned Relations**
  - E.g.: EMP where age < 20 TO file1
  - E.g.: EMP where age >= 20 TO file2

Special Purpose Concurrency

- Exploit transactions that *failure commute*
- E.g.: Given two bank withdrawals
  - Both will succeed if there are sufficient funds
  - The failure of one has no impact on the other
- Idea: Mark transaction in class "C1" or "C2"
  - Allow C1 transactions to run concurrently with each other, but not with C2 transactions
  - E.g.: Withdrawal as C1, Transfer as C2

Parallel Query Planner

- Find BIG = min(RAM_Needed, Total_RAM)
- Find *optimal* sequential plan for memory intervals:
  - [BIG, BIG/2], [BIG/2, BIG/4], …, [BIG/n, 0]
- Explore all possible parallel plans of each sequential plan
  - With a sprinkle of heuristics to limit plan space
- Use *optimal* parallel plan

Conclusions

- Parallel DBs important to meet future demands
- Historical context important
- Proved many can be made to perform the work of one, only better
- Horizontal partitioning effective
- Speedup and scaleup is possible, at least for sufficiently “small” node counts

Questions?