
Introduction to Parallel and Distributed Databases

25 April 2010
Lecture 6

Topics for Today

- Introduction to Parallel and Distributed Database
 - Motivations
- Architectures for Parallel-Databases
- Parallel Query Evaluation
- Source: R&G 22.1-22.4

Parallel and Distributed DBs

- So far we have talked about a database as sitting on one computer/server
 - Is that realistic?
 - Why would we want a different architecture?
- We'll talk the next 3 weeks about other options
 - Parallel databases
 - Distributed databases
- We'll discuss algorithms and techniques which can make them work
 - We'll try to bring in applications too
 - Distributed systems in general is a topic we'll learn about in Semester 1 of תשע"א

Parallel Databases: Definition

- A **parallel database** is one which involves multiple processors working together to achieve some goal
- The point is to increase
 - Throughput - scale
 - Response time - speed
- Parallel databases can improve by distributing data to multiple processors
 - We decide where to put the data based on speed and throughput considerations alone

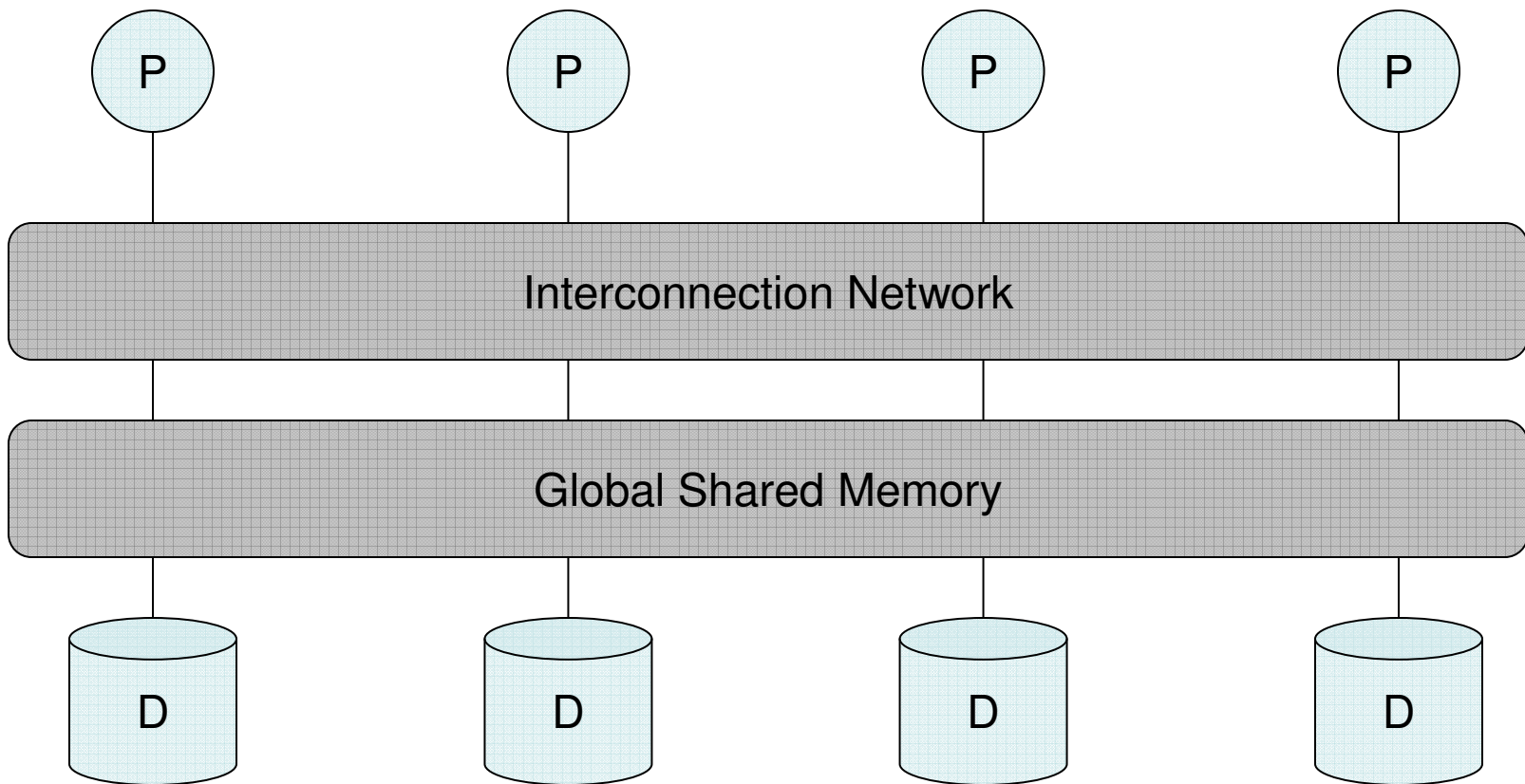
Distributed Database: Definition

- A **distributed database** places information in disparate or distant locations to achieve some goal.
 - Each location may be *independent of the others*
- Some common goals:
 - Exploit locales and regional needs
 - Provide content distribution
 - Provide resilience in the face of failures or crashes
 - Enable analysis of distributed data
- We could build a distributed parallel database too

Architectures for Parallel DBs

- Three basic ideas
 - Shared Memory
 - Shared Disk
 - Shared Nothing
- We could also build a combination: Hierarchical

Shared Memory



Shared Memory

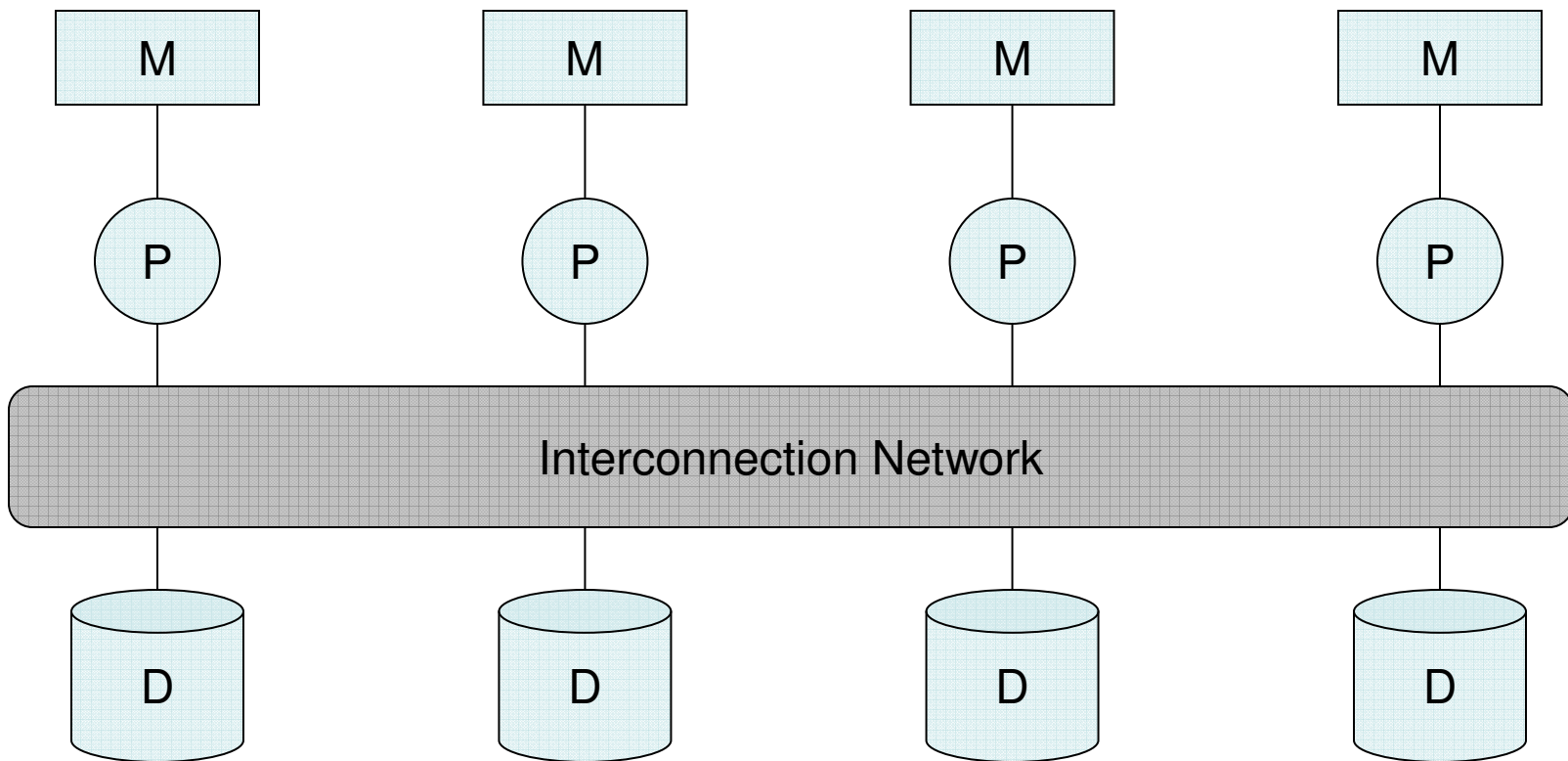
Shared Memory Properties

- Behaves like a single computer since multiple CPUs
- Each CPU can have an on board **cache**, but it shares main memory

- Advantage:
 - Message passing between processors is very fast via memory
 - Very easy to port to

- Disadvantage:
 - Cache coherence
 - Bus limitations and memory contentions
 - Maximum of about 64 processors
 - Once contention gets bad, adding more CPUs makes it worse

Shared Disk

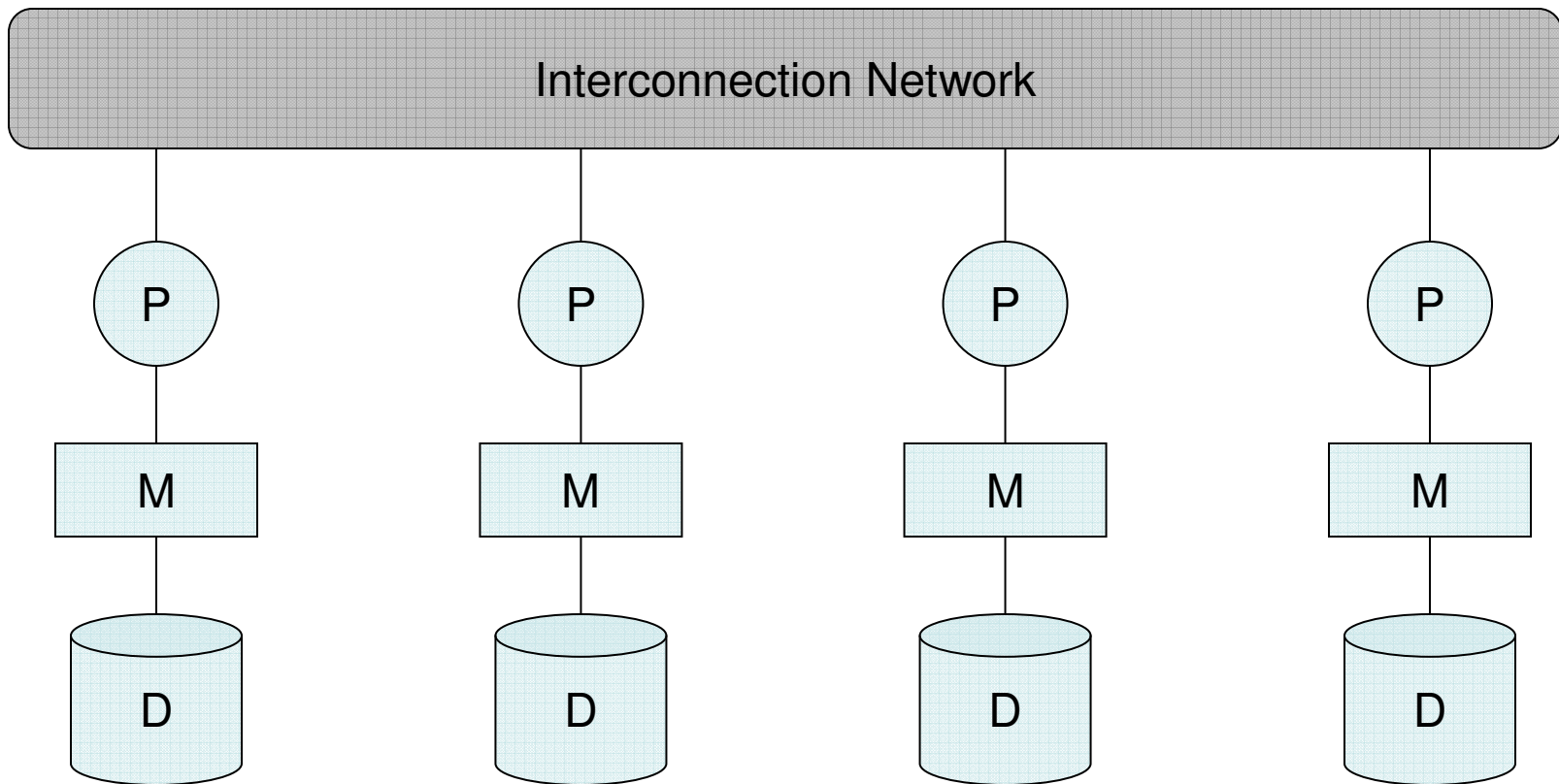


Shared Disk

Shared Disk Properties

- Can be more efficient than Shared Memory since each CPU has its own memory
 - Less contention if they can use it
- Advantages:
 - Much less chance for contention
 - Can introduce fault tolerance, similar to RAID
- Disadvantages:
 - Disk contention if there is lots of data to ship around
 - Cache coherence too
 - Communication slower than Shared Memory since it goes to disk

Shared Nothing



Shared Nothing

Shared Nothing Properties

- Each processor is its own independent computer
 - Each processor is in charge of the data which sits on its disk
 - If the processor doesn't want to own the records, it shouldn't accept them
- Advantages:
 - No contention for resources
 - Makes division of work rely of data distribution
- Disadvantages:
 - Communication is much slower
 - If lots of data needs to be shipped, this will be an issue

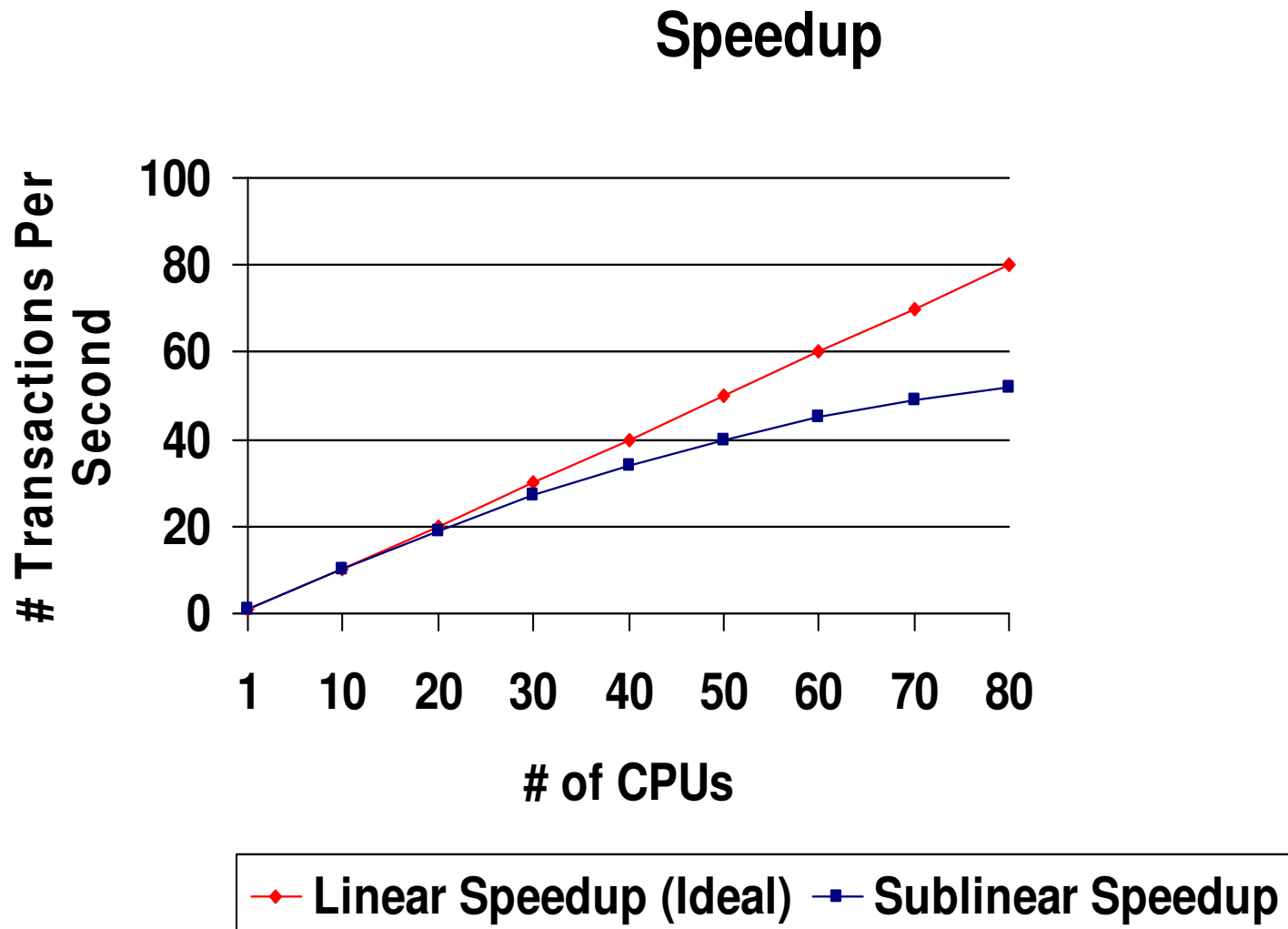
Hierarchical

- Combine the ideas
- Example:
 - 4 Shared Nothing Database Servers
 - Each Database Server is a Shared Memory system

Goals of Parallelization

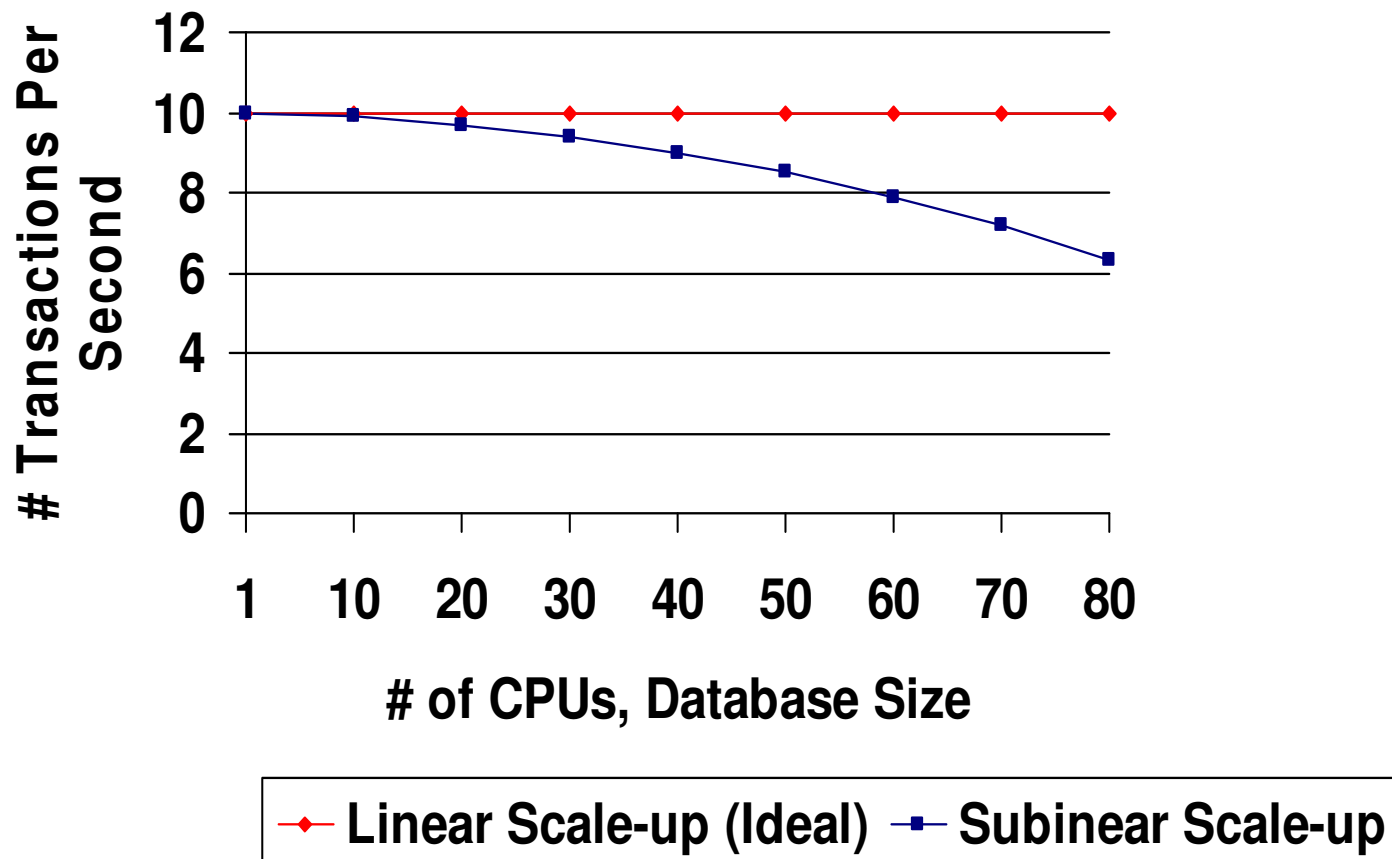
- Improve throughput or response time
 - Don't let adding a processor make the algorithm go slower
- Some graphs to show ideal and real with respect to transactions and processors:

Speedup



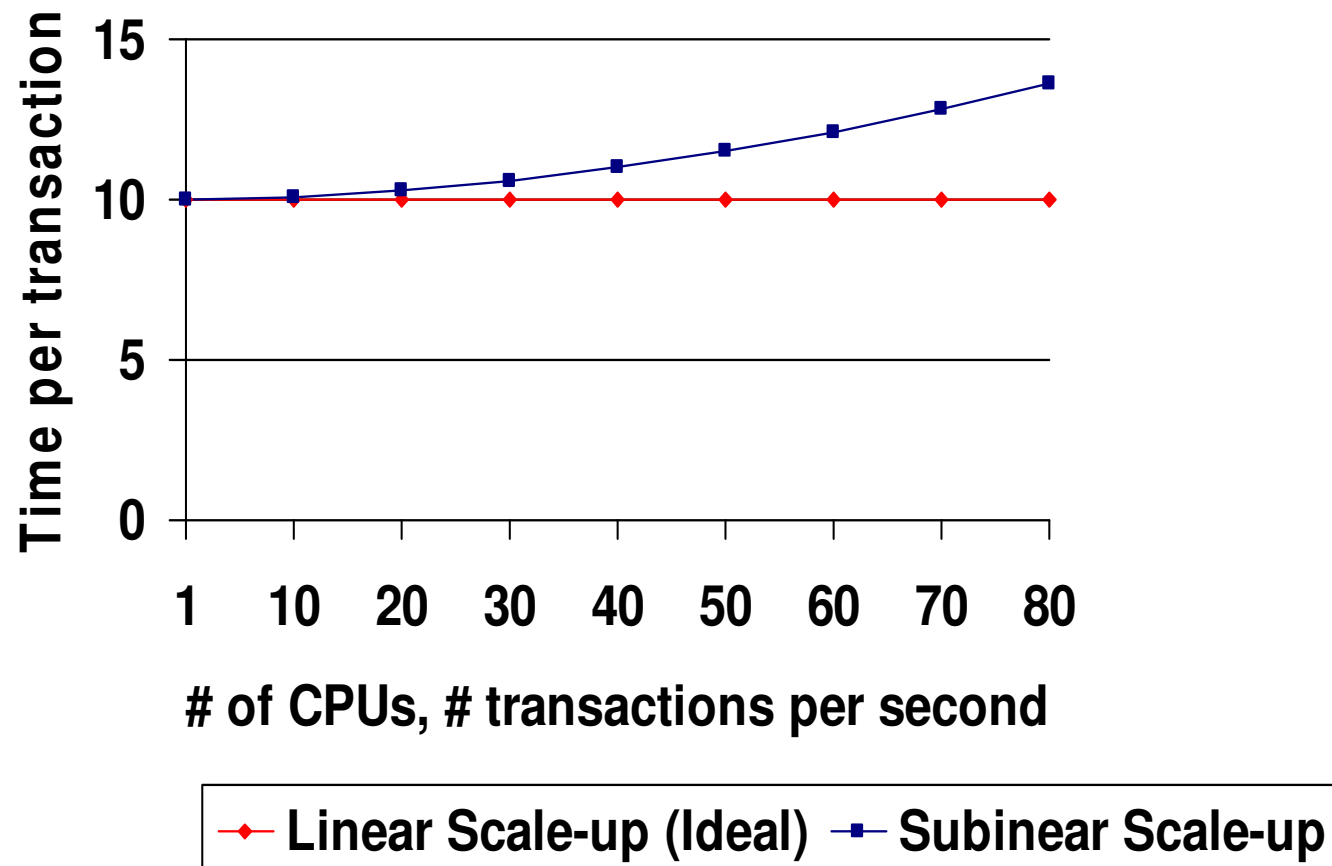
Scale-Up

Scale-Up with DB Size



Scale-up with Transactions

Scale-Up with #XACTS/Second



Graph Lessons

- Adding more processors, but keeping the number of transactions constant ought to lead to an increase in speedup
- Increasing the number of processors and size of the data in the database proportionately should give a flat line
- Increasing the processors in the database as well as the number of transactions proportionately should also be a flat line

So Far

- Introduction to Parallel and Distributed Database
 - Motivations
- Architectures for Parallel-Databases
- **Parallel Query Evaluation**

Parallel Query Evaluation

- We'll consider a shared nothing architecture
 - Other ones will work too with some adaptations
- How could we parallelize?
 - Multiple queries
 - A single query
- We'll focus on single query parallelization
 - Multiple queries are doable if they're all SELECTs, but if we modify data, it's much harder to predict

Introduction to Parallelization

- Serial Execution
 - Begin
 - Run
 - End
- Data Partitioning
- Skew
- Pipelining

What do we gain by it?

Algorithm a on data n

- Complete serial execution: $S_a(n)$
 - So working on *many* n_1, n_2, n_3, n_4 : We need to do $S_a(n_1) + S_a(n_2) + S_a(n_3) + S_a(n_4) + \dots$
- What if we could run a on n by breaking it into m pieces and running it on m computers?
 - Ideally: $S_a(n)/m$ runtime
- That's almost *never* the case, there is some setup and collecting time (may be relative to the size of n):
 - Setup time: $B_a(n)$
 - Completion time: $C_a(n)$
 - Our real run time: $B_a(n) + S_a(n)/m + C_a(n)$
- So we must check whether:

$$B_a(n) + S_a(n)/m + C_a(n) < S_a(n)$$

But that's not all... (1/2)

$$B_a(n) + S_a(n)/m + C_a(n) < S_a(n)$$

How do we divide up the job into m pieces?

- Do we ship the data to the m computers and then have them ship back the results?
 - Distribution time: $D_a(n)$
 - Recollection time: $R_a(n)$
 - So we must now check: $B_a(n) + D_a(n) + S_a(n)/m + R_a(n) + C_a(n) < S_a(n)$
-

- What if we don't do the division of work perfectly?
 - Say we made n_1, n_2, \dots, n_m pieces
 - Let $n_{max} = MAX(n_1, n_2, \dots, n_m)$
 - Then we really are checking:
$$B_a(n) + D_a(n) + S_a(n_{max}) + R_a(n) + C_a(n) < S_a(n)$$
 - That means we are benefitting only as much as we can reduce the size of the biggest piece of n
 - **Skew** refers to how fair the division is: $(n_{max} - n/m)$ (**less = better**)

But that's not all... (2/2)

$$B_a(n) + D_a(n) + S_a(n_{max}) + R_a(n) + C_a(n) < S_a(n)$$

- This still isn't realistic – most algorithms contain a portion which is not parallelizable
 - It may be a part which depends serially on its previous output
 - It may be some processing which must be done together
- Call the parallelizable segment: $P_a(n)$
- Call the rest $NP_a(n)$

So we really must check:

$$B_a(n) + D_a(n) + NP_a(n) + P_a(n_{max}) + R_a(n) + C_a(n) < S_a(n)$$

Conclusion

- Introduction to Parallel and Distributed Database
 - Motivations
- Architectures for Parallel-Databases
- Parallel Query Evaluation