CS 764: Topics in Database Management Systems

Lecture 7: Parallel Database

Xiangyao Yu

9/25/2025

Today's Paper: Parallel DBMSs

Parallel Database Systems:
The Future of High Performance Database Processing[1]

David J. DeWitt[2]
Computer Sciences Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI. 53706
dewitt @ cs.wisc.edu

Jim Gray
San Francisco Systems Center
Digital Equipment Corporation
455 Market St. 7th floor
San Francisco, CA. 94105-2403
Gray @ SFbay.enet.dec.com

January 1992

Abstract: Parallel database machine architectures have evolved from the use of exotic hardware to a software parallel dataflow architecture based on conventional shared-nothing hardware. These new designs provide impressive speedup and scaleup when processing relational database queries. This paper reviews the techniques used by such systems, and surveys current commercial and research systems.

1. Introduction

Highly parallel database systems are beginning to displace traditional mainframe computers for the largest database and transaction processing tasks. The success of these systems refutes a 1983 paper predicting the demise of database machines [BORA83]. Ten years ago the future of highly-parallel database machines seemed gloomy, even to their staunchest advocates. Most database machine research had focused on specialized, often trendy, hardware such as CCD memories, bubble memories, head-per-track disks, and optical disks. None of these technologies fulfilled their promises; so there was a sense that conventional cpus, electronic RAM, and moving-head magnetic disks would dominate the scene for many years to come. At that time, disk throughput was predicted to double while processor speeds were predicted to increase by much larger factors. Consequently, critics predicted that multi-processor systems would soon be I/O limited unless a solution to the I/O bottleneck were found.

While these predictions were fairly accurate about the future of hardware, the critics were certainly wrong about the overall future of parallel database systems. Over the last decade Teradata, Tandem, and a host of startup companies have successfully developed and marketed highly parallel database machines.

Communications of the ACM, 1992

Agenda

Parallelism metrics

Parallel architecture

Parallel OLAP operators

Cloud parallel database

Agenda

Parallelism metrics

Parallel architecture

Parallel OLAP operators

Cloud parallel database

Parallel Database History

1980's: database machines

- Specialized hardware to make databases run fast
- Special hardware cannot catch up with Moore's Law

1980's – 2010's: shared-nothing architecture

- Connecting machines using a network

2010's – future?

Scaling in Parallel Systems

Linear speedup

- Twice as much hardware can perform the task in half the elapsed time
- $\text{Speedup} = \frac{small\ system\ elapsed\ time}{big\ system\ elapsed\ time}$
- Linear speedup = N, where the big system is N times larger than the small system

Scaling in Parallel Systems

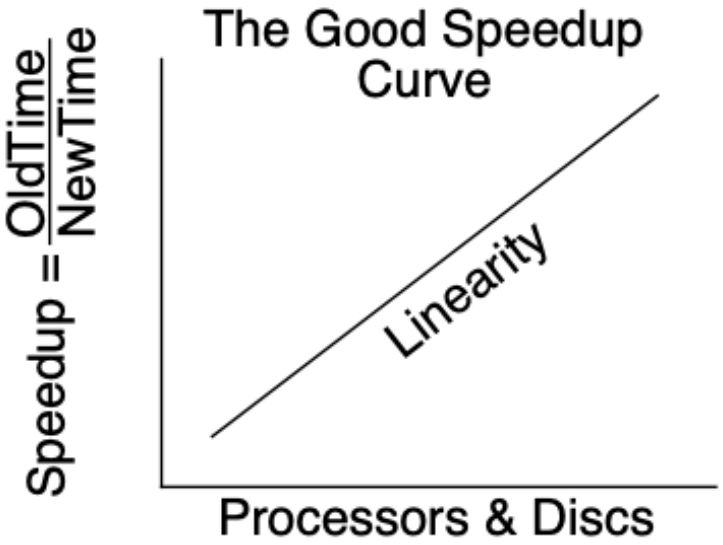
Linear speedup

- Twice as much hardware can perform the task in half the elapsed time
- $\text{Speedup} = \frac{small\ system\ elapsed\ time}{big\ system\ elapsed\ time}$
- Linear speedup = N, where the big system is N times larger than the small system

Linear scaleup

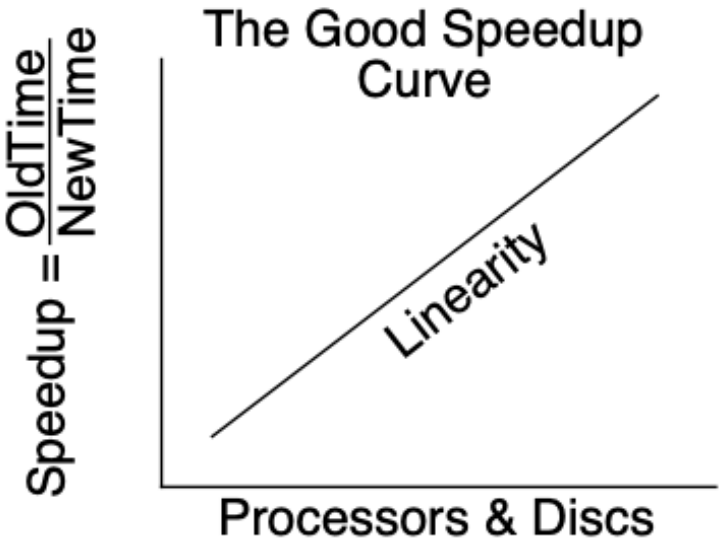
- Twice as much hardware can perform twice as large a task in the same elapsed time
- $\text{Scaleup} = \frac{small\ system\ elapsed\ time\ on\ small\ problem}{big\ system\ elapsed\ time\ on\ big\ problem}$
- Linear scaleup = 1

Scaling in Parallel Systems

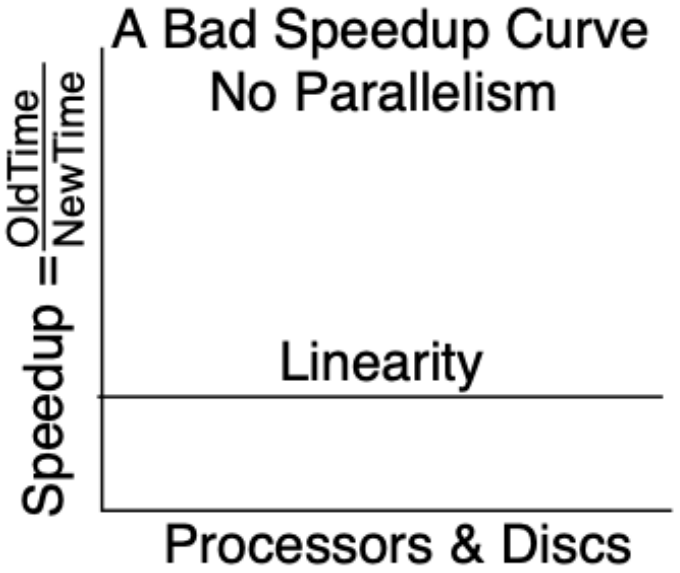


Ideal speedup

Scaling in Parallel Systems

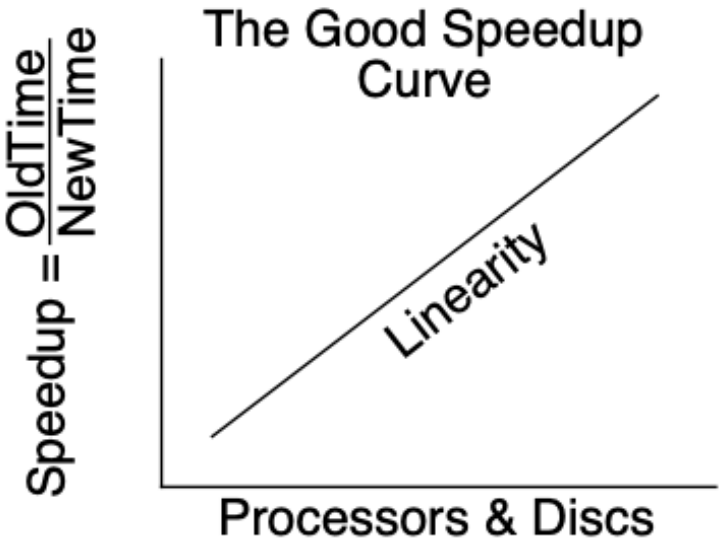


Ideal speedup

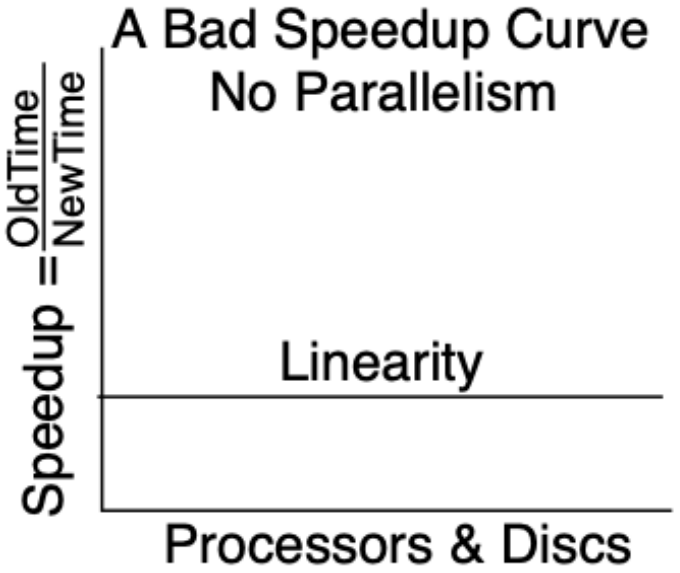


No speedup

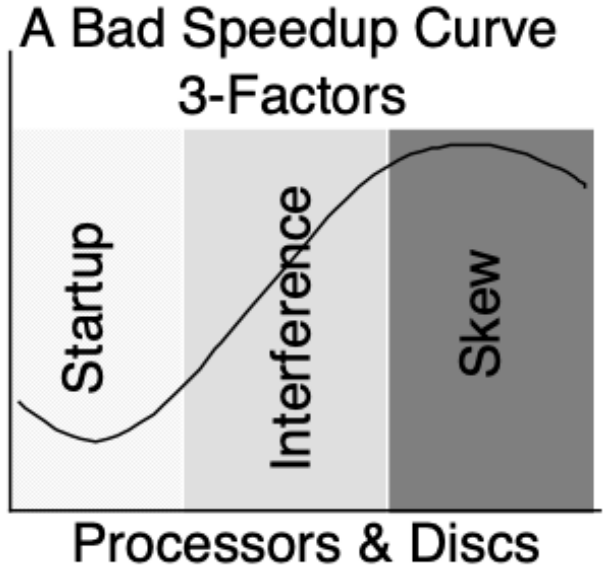
Scaling in Parallel Systems



Ideal speedup

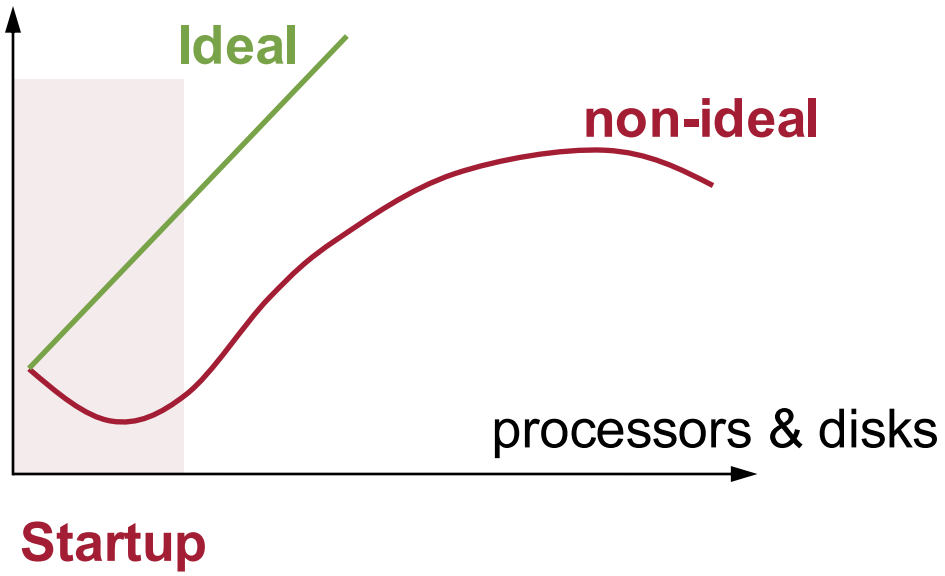


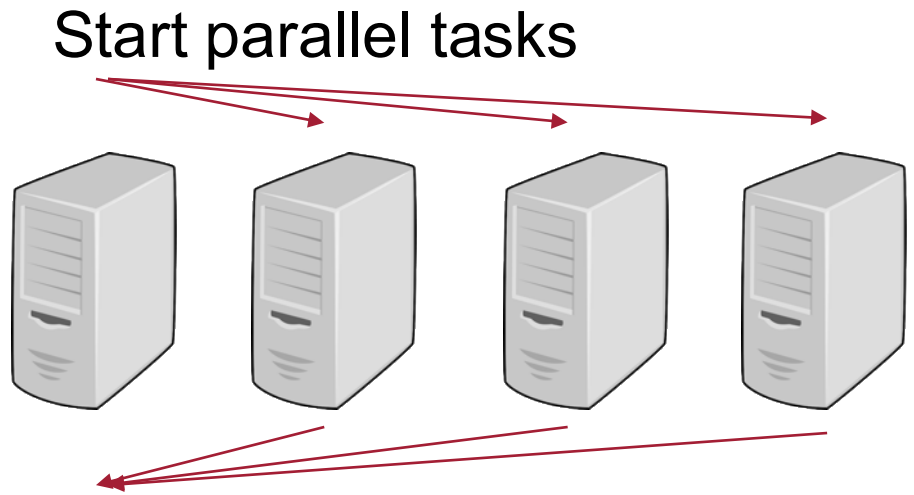
No speedup



In practice

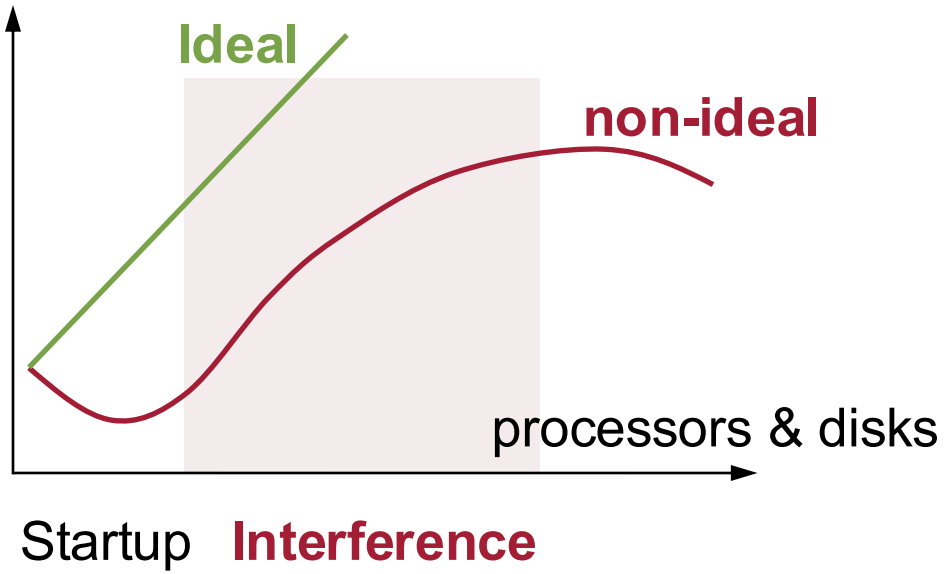
Threats to Parallelism

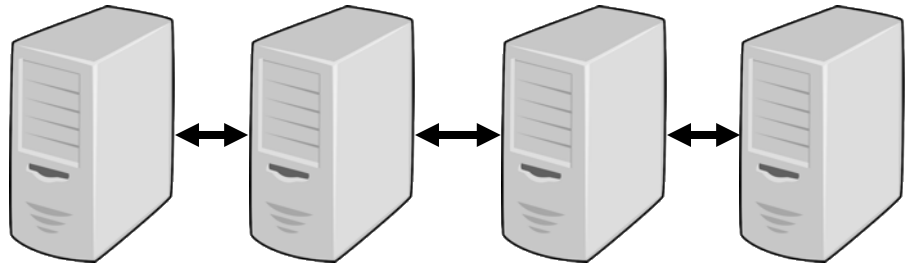




Starting remote tasks incurs performance overhead

Threats to Parallelism

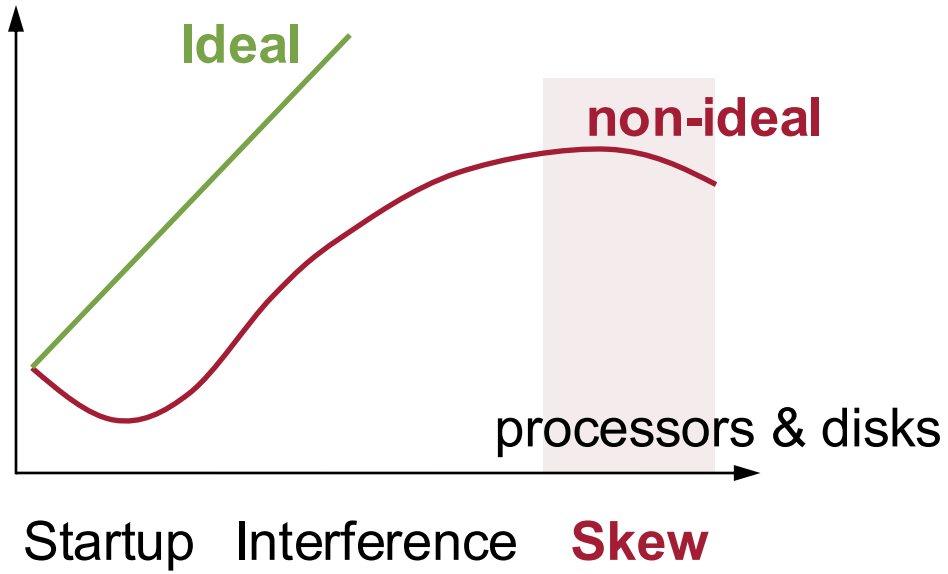


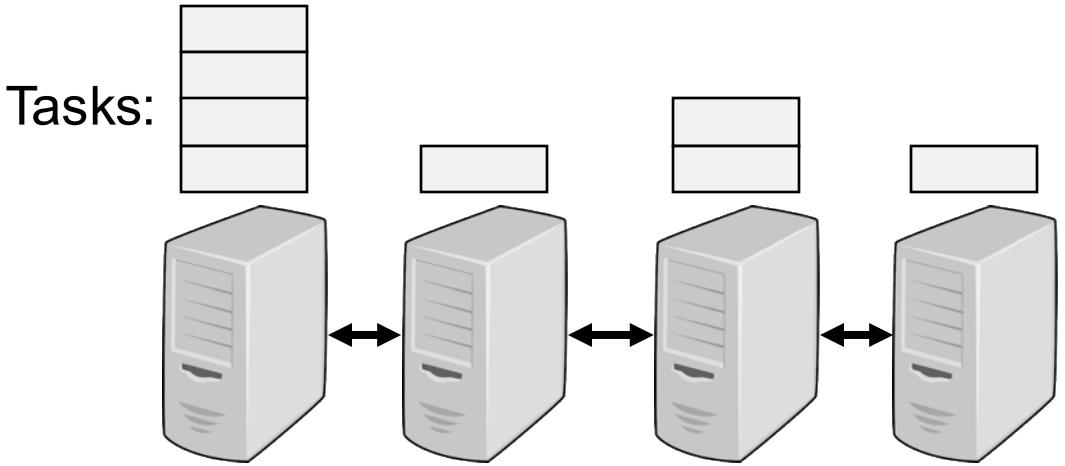


Examples of interference

- Shared hardware resources (e.g., memory, disk, network)
- Synchronization (e.g., locking)

Threats to Parallelism





Some nodes take more time to execute the assigned tasks, e.g.,

- More tasks assigned
- More computational intensive tasks assigned
- Node has slower hardware

Agenda

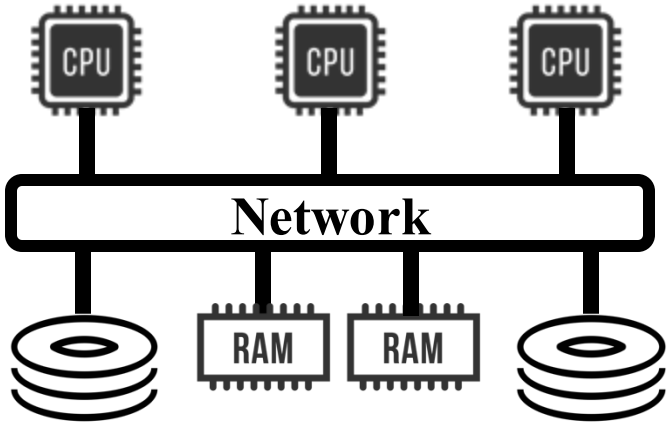
Parallelism metrics

Parallel architecture

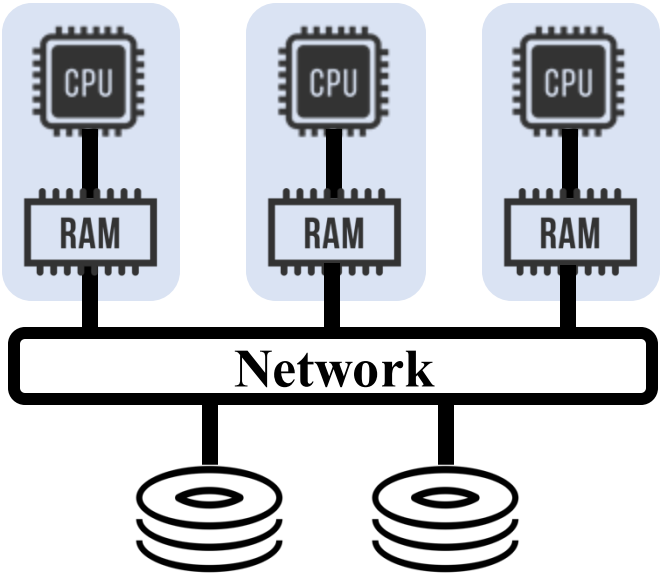
Parallel OLAP operators

Cloud parallel database

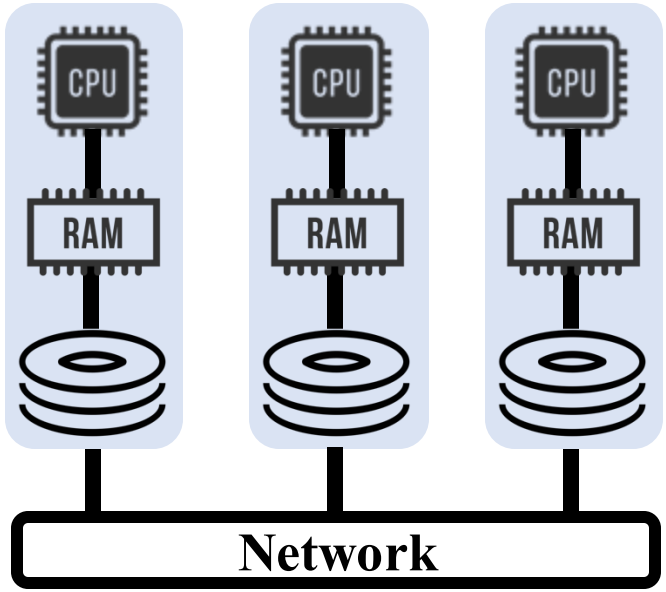
Design Spectrum



Shared Memory



Shared Disk



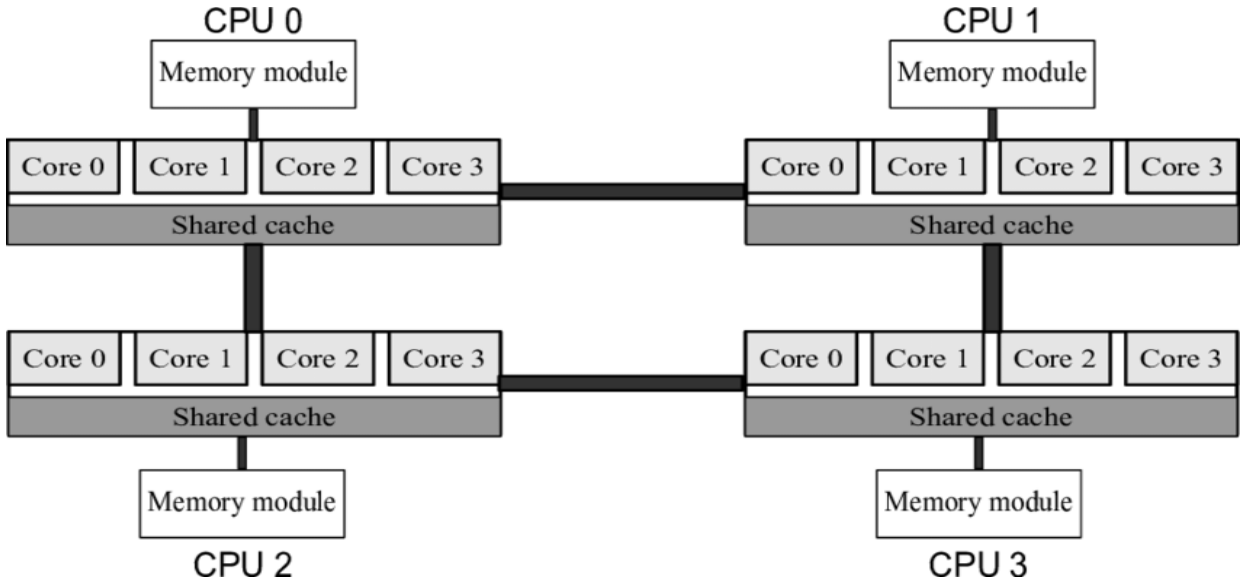
Shared Nothing

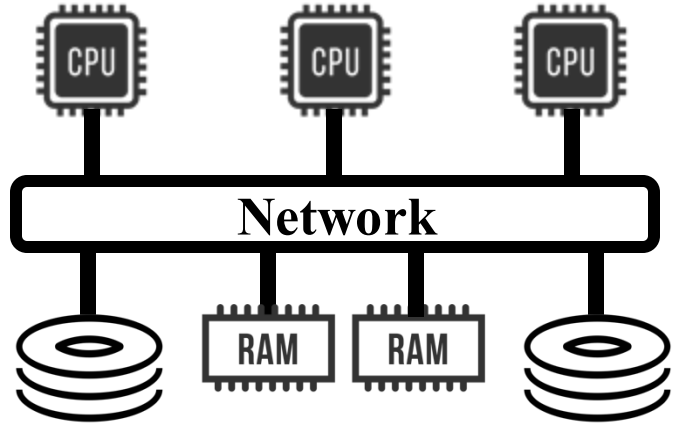
Design Spectrum – Shared Memory (SM)

All processors share direct access to a common global memory and to all disks

- Does not scale beyond **a single server**

Example: multicore processors





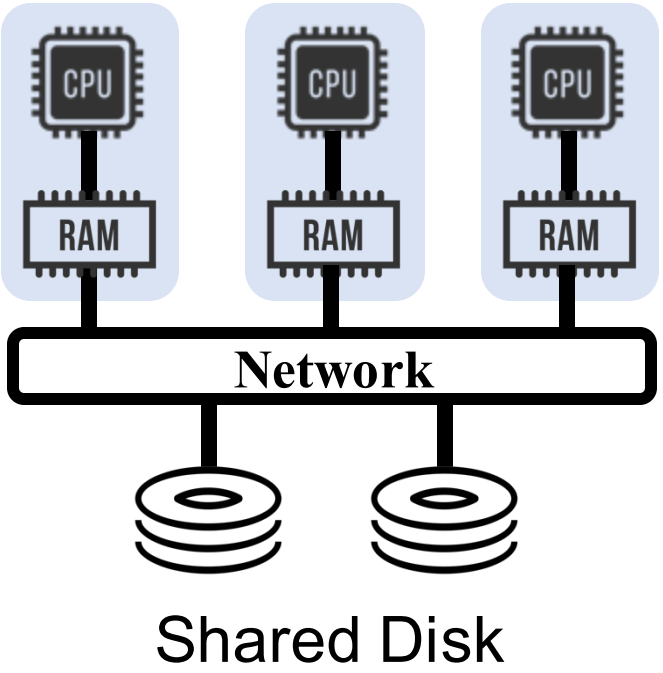
Shared Memory

Design Spectrum – Shared Disk (SD)

Each processor has a private memory but has direct access to all disks

- Does not scale beyond **tens of servers**

Example: Network attached storage (NAS) and storage area network (SAN)

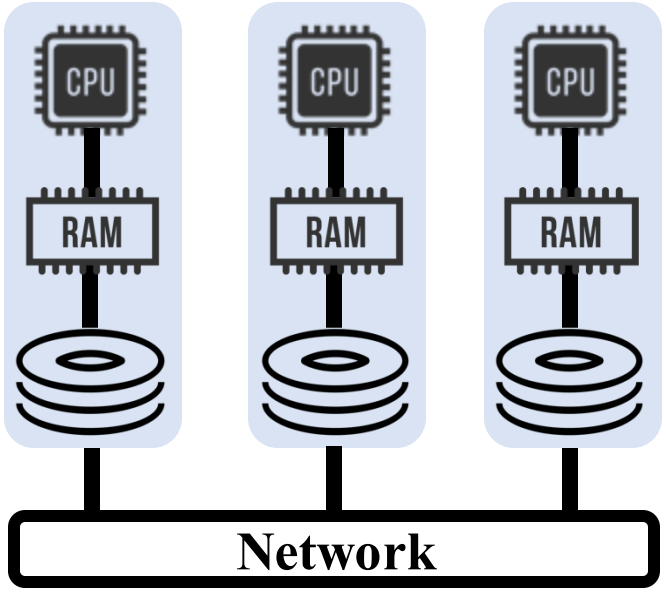


Design Spectrum – Shared Nothing (SN)

Each memory and disk is owned by some processor that acts as a server for that data

- Scales to **thousands of servers and beyond**

Important optimization goal: minimize network data transfer



Shared Nothing

Agenda

Parallelism metrics

Parallel architecture

Parallel OLAP operators

Cloud parallel database

How to Build Parallel Database?

Old uni-processor software must be rewritten to benefit from parallelism

Most database programs are written in relational language SQL

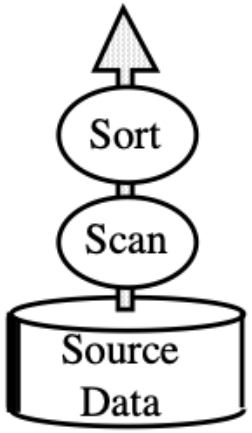
- **Can make SQL work on parallel hardware without rewriting**
- Benefits of a high-level programming interface

How to Build Parallel Database?

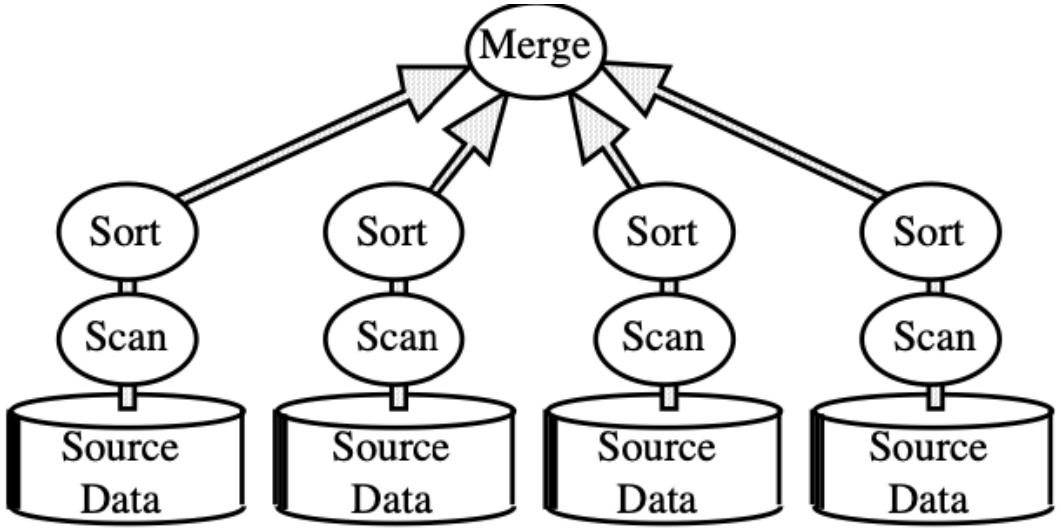
Old uni-processor software must be rewritten to benefit from parallelism

Most database programs are written in relational language SQL

- **Can make SQL work on parallel hardware without rewriting**
- Benefits of a high-level programming interface



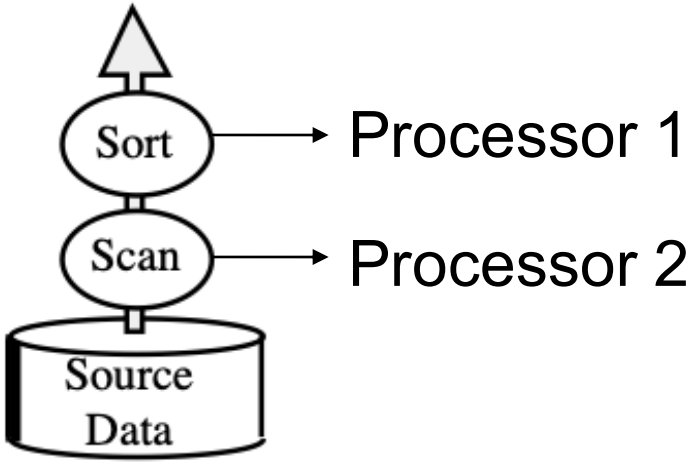
Pipelined Parallelism



Partitioned Parallelism

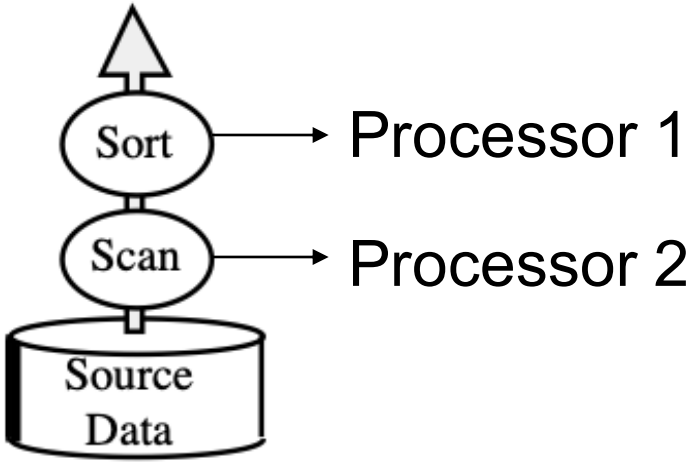
Pipelined Parallelism

Pipelined parallelism: pipeline of operators



Pipelined Parallelism

Pipelined parallelism: pipeline of operators

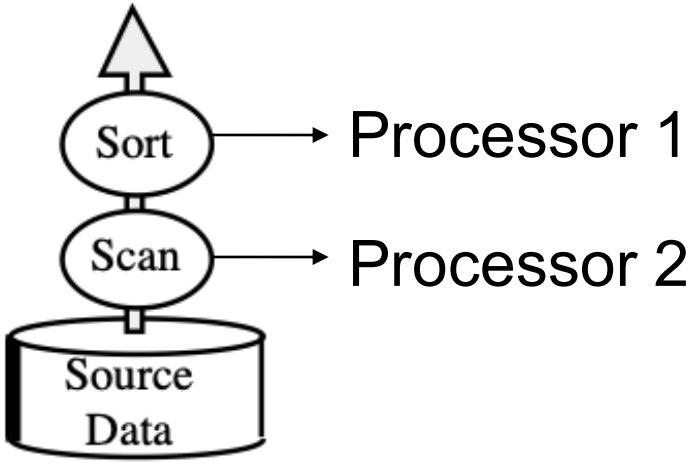


Advantages

- Avoid writing intermediate results back to disk

Pipelined Parallelism

Pipelined parallelism: pipeline of operators



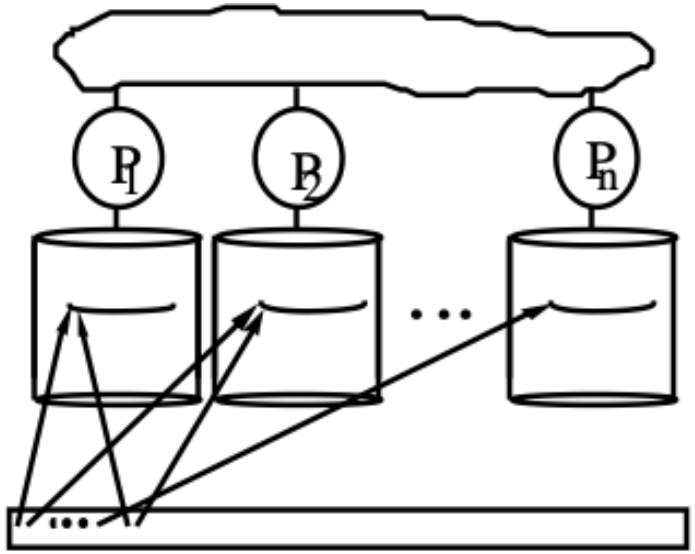
Advantages

- Avoid writing intermediate results back to disk

Disadvantages

- Small number of stages in a query
- Blocking operators: e.g., sort and aggregation
- Different speed: scan faster than join. Slowest operator becomes the bottleneck

Partitioned Parallelism

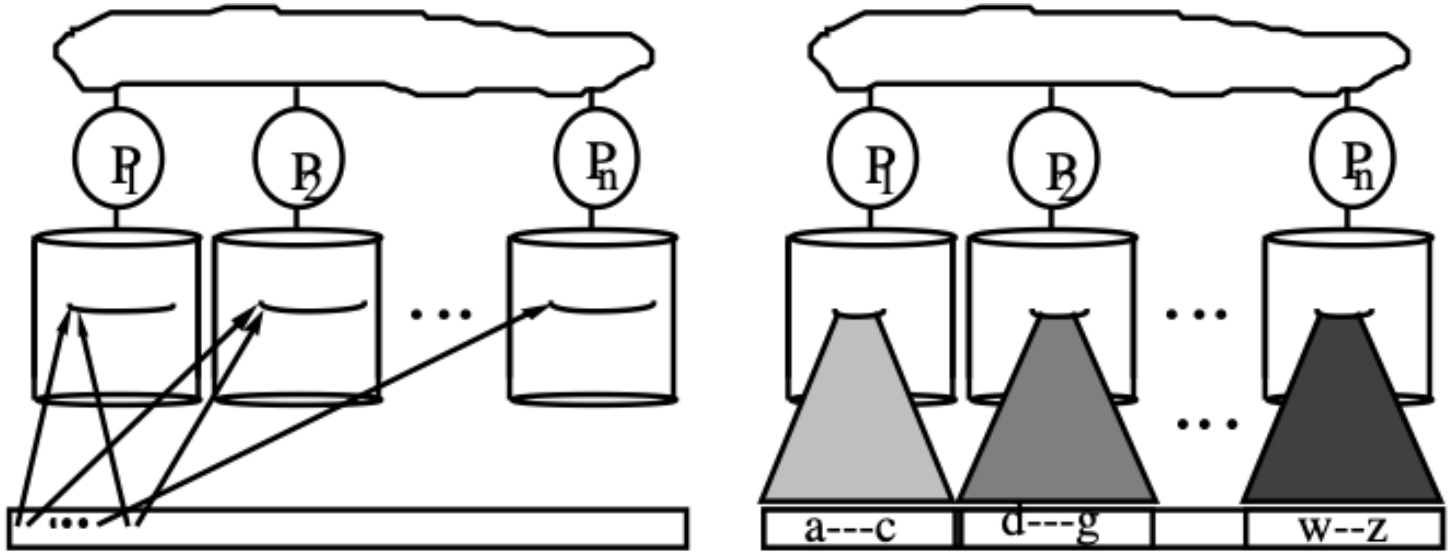


Round robin

Map tuple *i* to disk (*i mode n*)

- **Advantage**: Simplicity, good load balancing
- **Disadvantage**: Hard to identify the partition of a particular record

Partitioned Parallelism

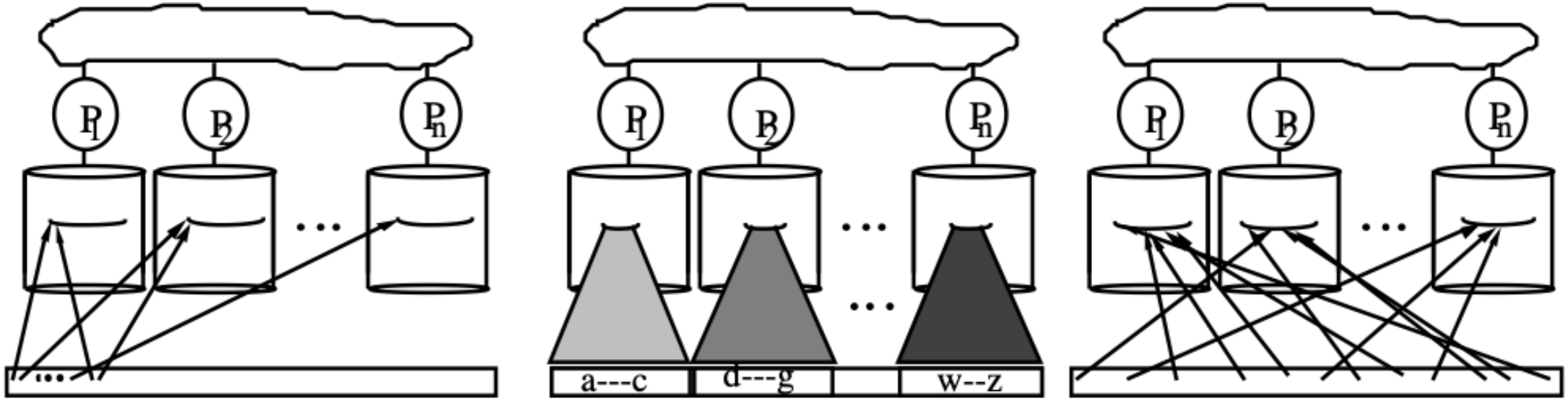


Round robin Range Partitioning

Map contiguous attribute ranges to partitions

- **Advantage**: Good locality due to clustering
- **Disadvantage**: May suffer from skewness

Partitioned Parallelism



Round robin | Range Partitioning | Hash Partitioning

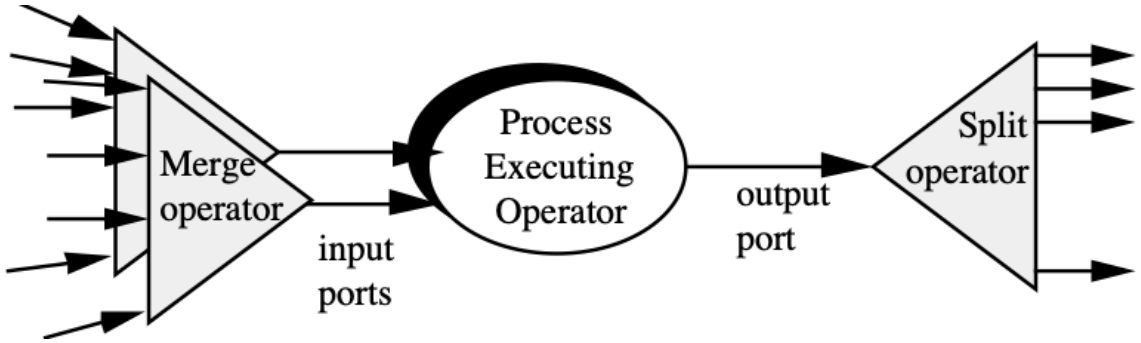
Map based on the hash value of tuple attributes

- **Advantage**: Good load balance, low skewness
- **Disadvantage**: Bad locality

Parallelism within Relational Operators

Parallel data streams so that sequential operator code is not modified

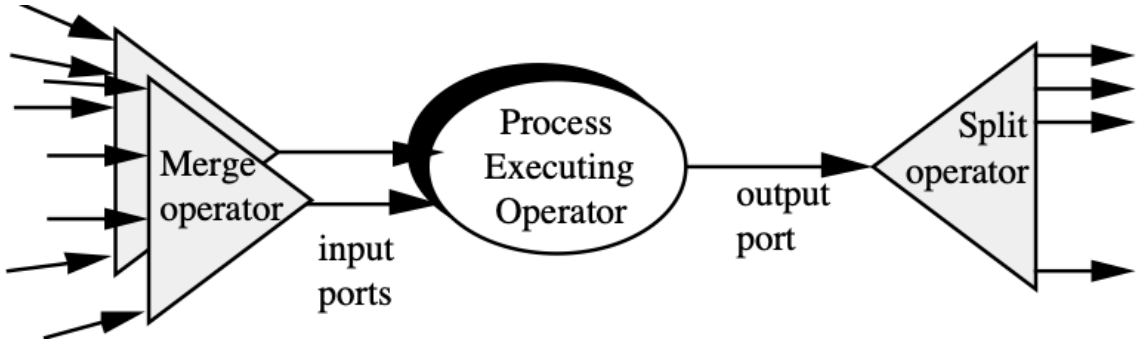
- Each operator has a set of input and output ports
- Partition and merge these ports to sequential ports so that **an operator is not aware of parallelism**

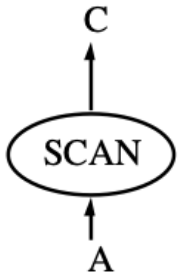


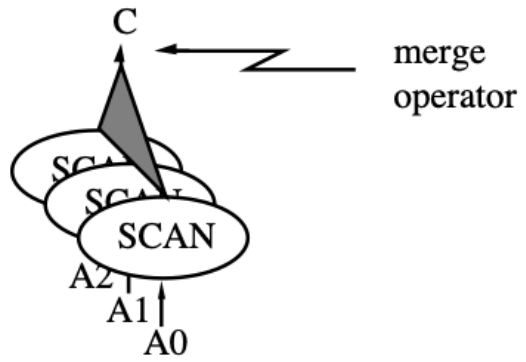
Parallelism within Relational Operators

Parallel data streams so that sequential operator code is not modified

- Each operator has a set of input and output ports
- Partition and merge these ports to sequential ports so that **an operator is not aware of parallelism**





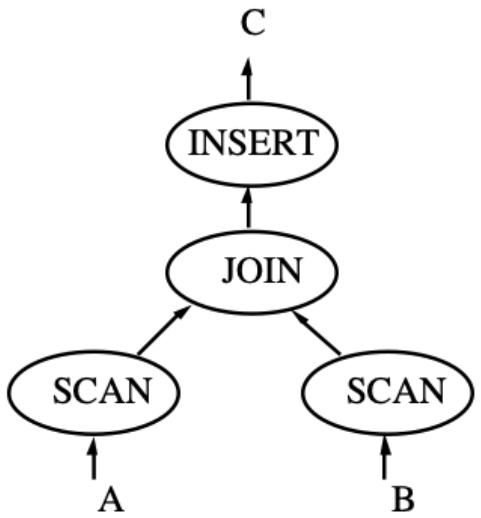


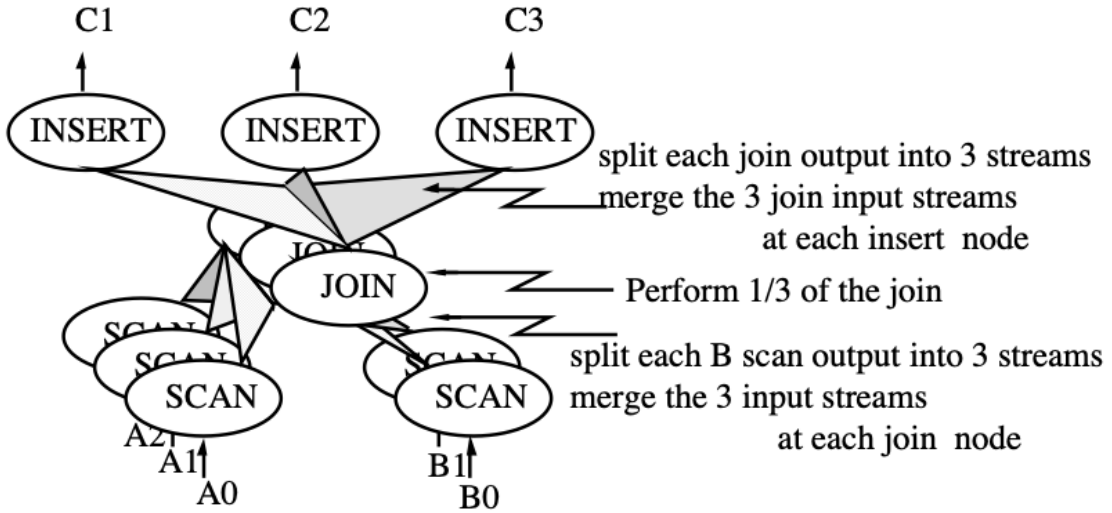
Parallelism within Relational Operators

Parallel data streams so that sequential operator code is not modified

- Each operator has a set of input and output ports
- Partition and merge these ports to sequential ports so that **an operator is not aware of parallelism**

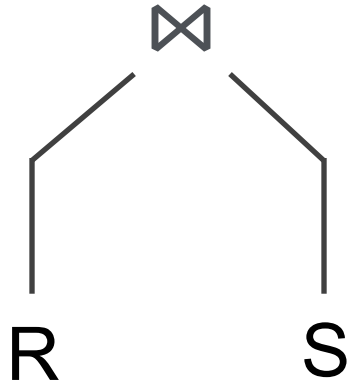
```
insert into C
  select     *
  from       A, B
  where      A.x = B.y;
```



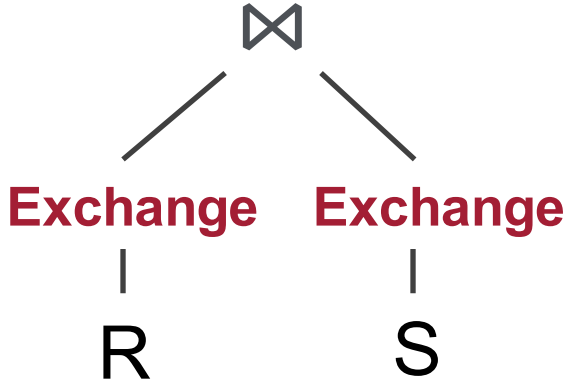


Data Shuffle

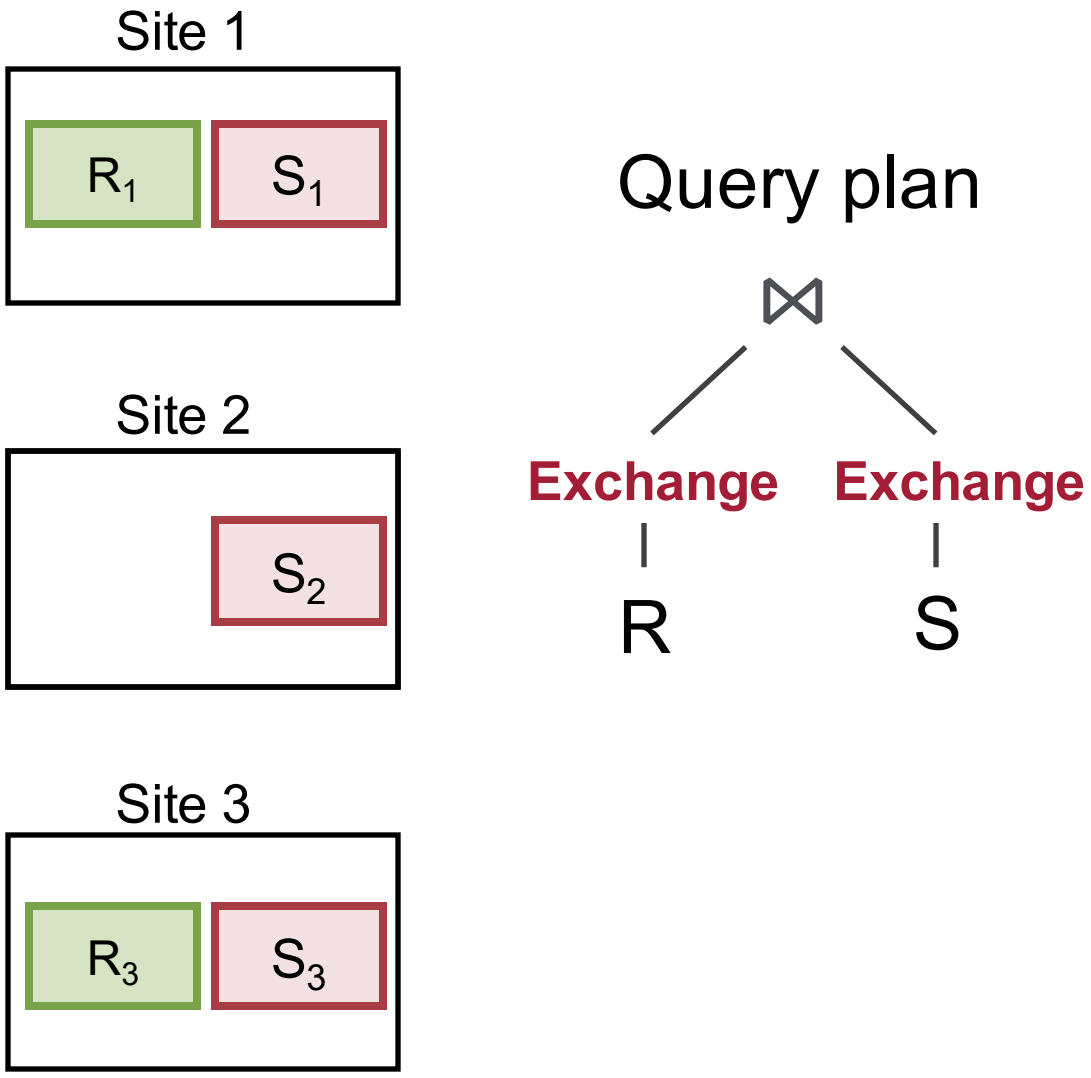
Single-node query plan



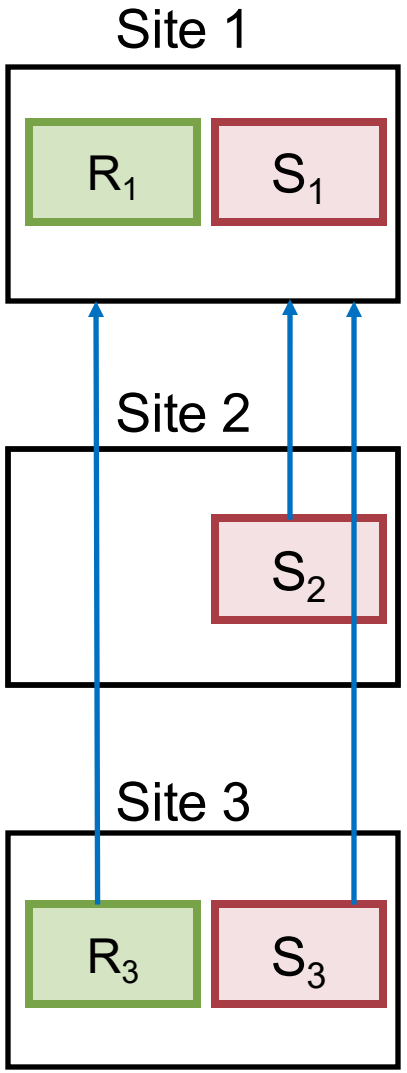
Distributed query plan

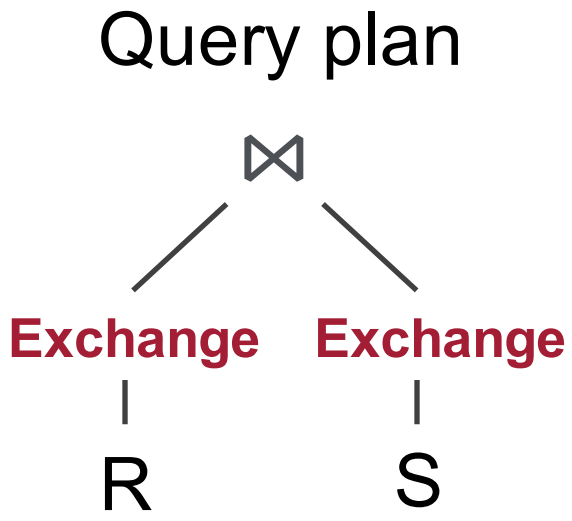


Data Shuffle – Example



Data Shuffle – Single-Site

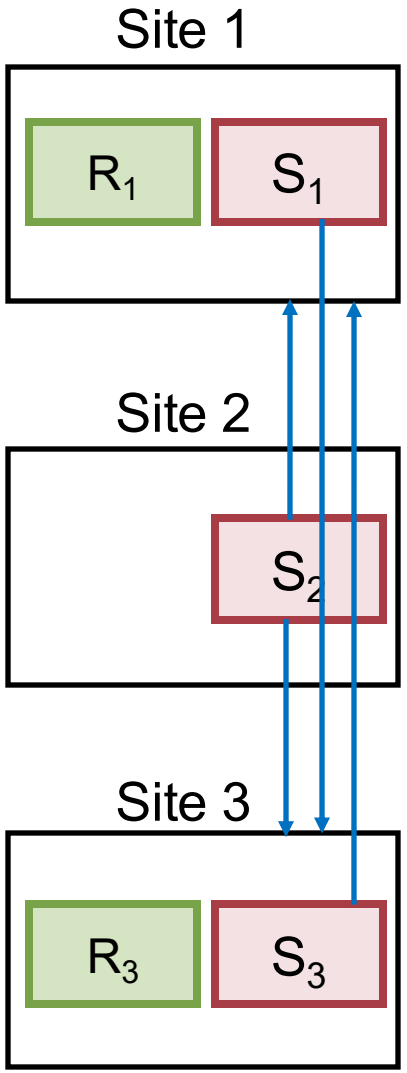




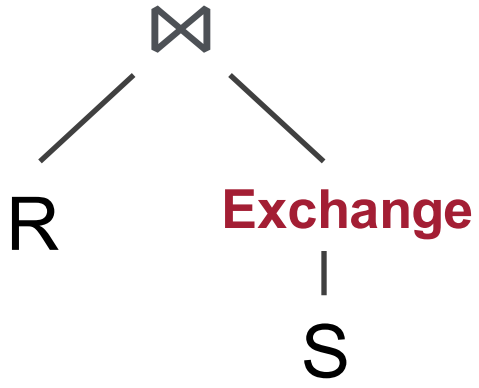
Solution 1: send all the involved tables to a single site

- **Advantage**: Single-site query execution is a solved problem
- **Disadvantage**: (1) Single site execution can be slow (2) Data may not fit in single site's memory or disk

Data Shuffle – Broadcast



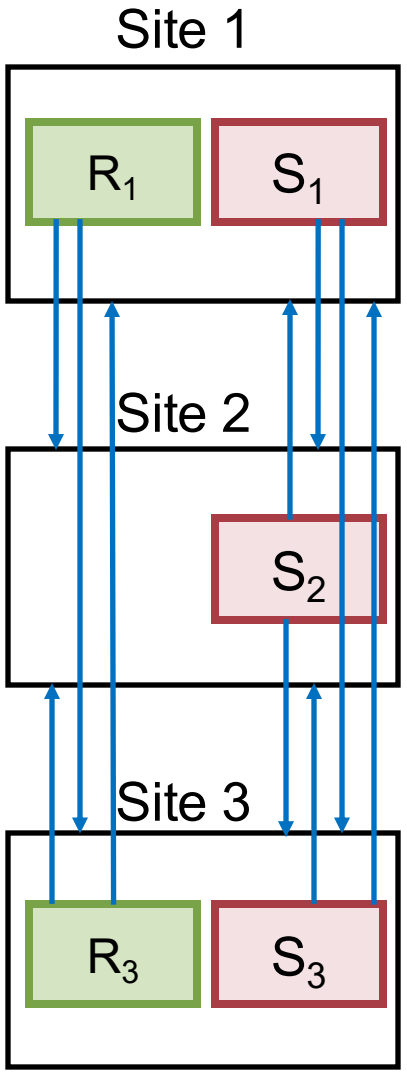
Query plan

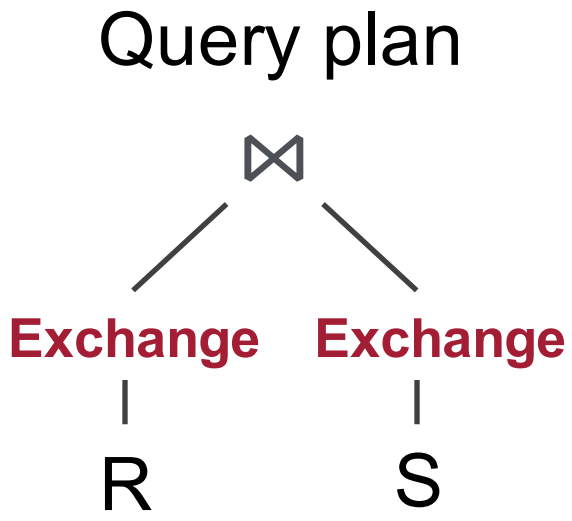


Solution 2: Keep one relation partitioned and broadcast the other relation to all sites

- **Advantage**: One relation does not need to move
- **Disadvantage**: Still need to broadcast the other relation to all sites

Data Shuffle – Co-partition





Solution 3: Partition both relations using the join key

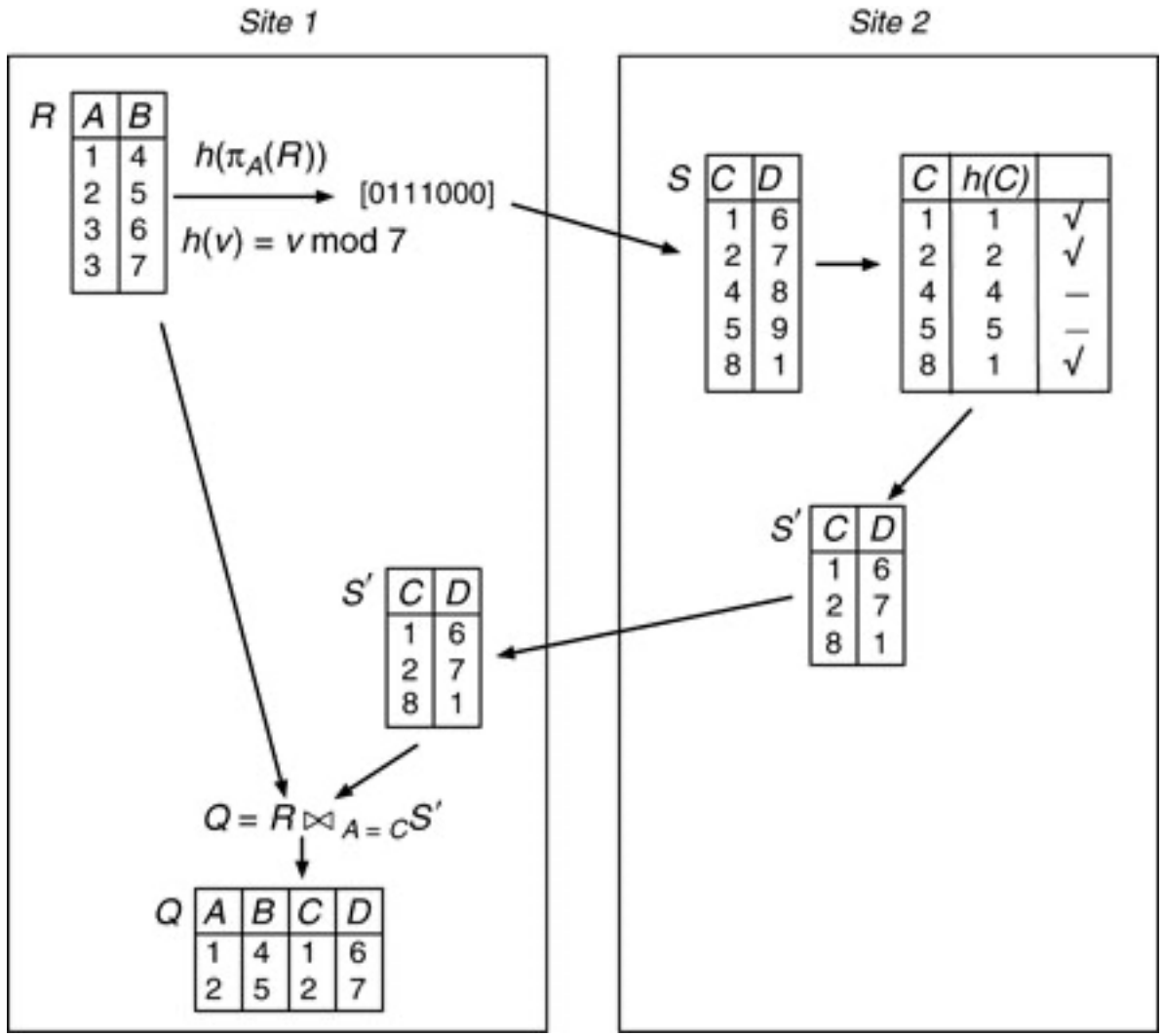
- **Advantage**: Each site has less data to process
- **Disadvantage**: Both relations are shuffled (if not already partitioned based on join key)

Specialized Parallel Operators

Semi-join

- Example:

```
SELECT *
FROM T1, T2
WHERE T1.A = T2.C
```



* Source: Sattler KU. (2009) Semijoin. Encyclopedia of Database Systems.

Agenda

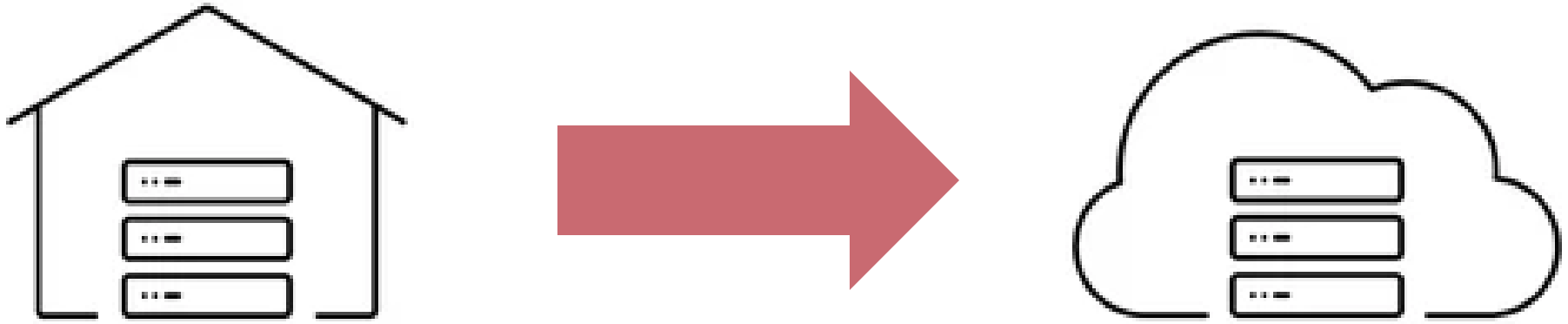
Parallelism metrics

Parallel architecture

Parallel OLAP operators

Cloud parallel database

Databases Moving to the Cloud



On-premises database

- Fixed capacity
- Scaling takes months/years

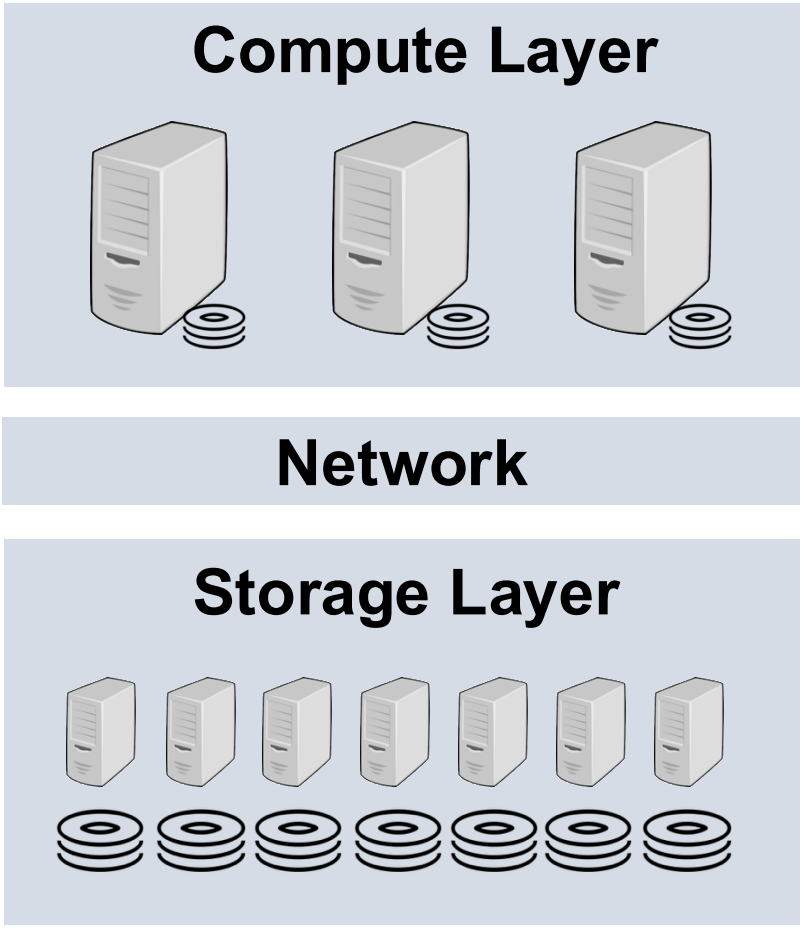
Cloud database

- **On-demand scalability**
- Scaling takes seconds/minutes

Traditional architectures fail to leverage cloud elasticity

Storage Disaggregation Architecture

Why disaggregate? Compute and storage behave differently

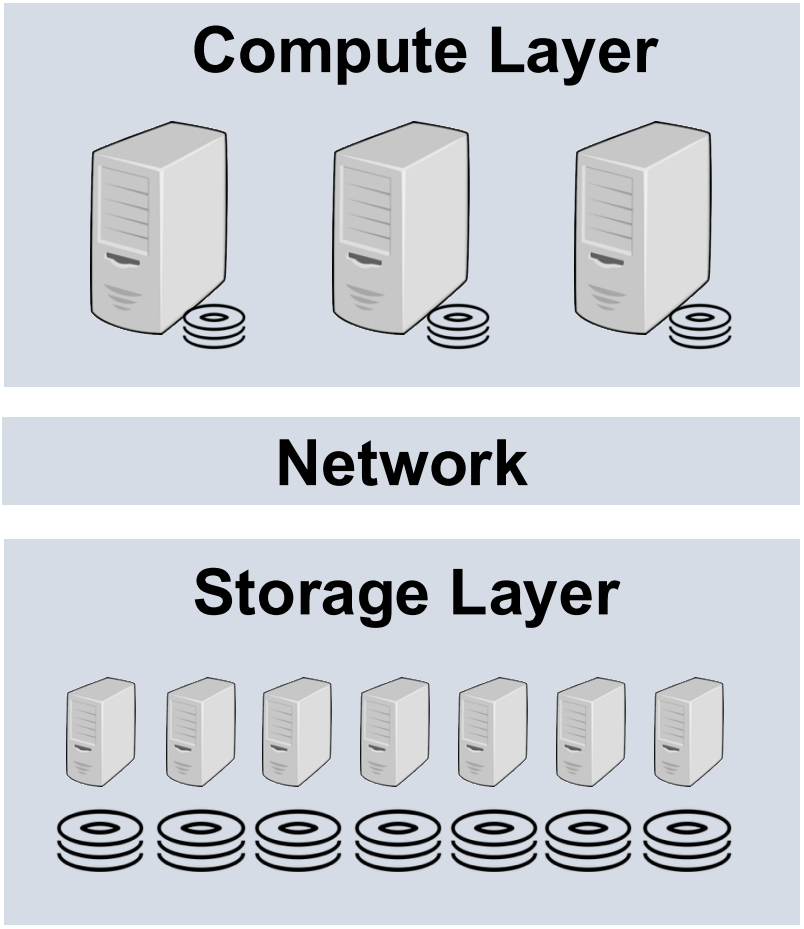


- **Stateless** → easy to scale
- **Expensive** → VMs cost **$0.1–10/hour**
- **Bursty** → demand changes rapidly

- **Stateful** → difficult to scale
- **Low-cost** → S3 costs **$0.02/GB/month**
- **Steady** → demand changes slowly

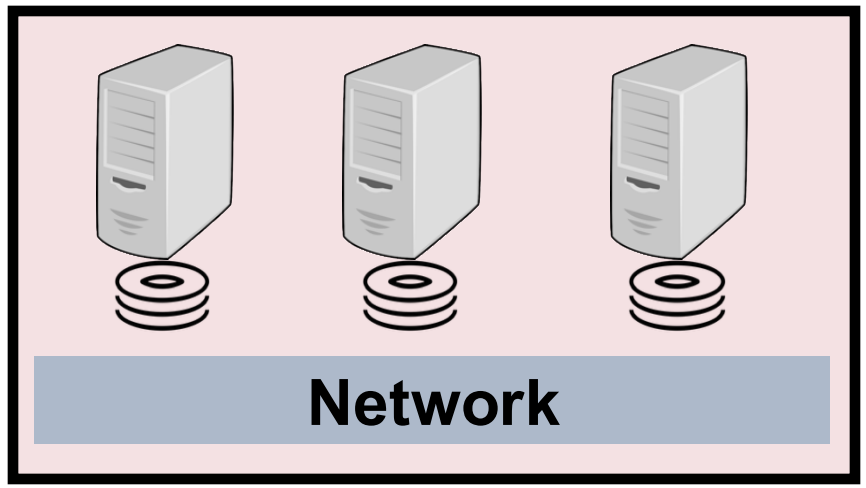
Storage Disaggregation Architecture

Why disaggregate? Compute and storage behave differently

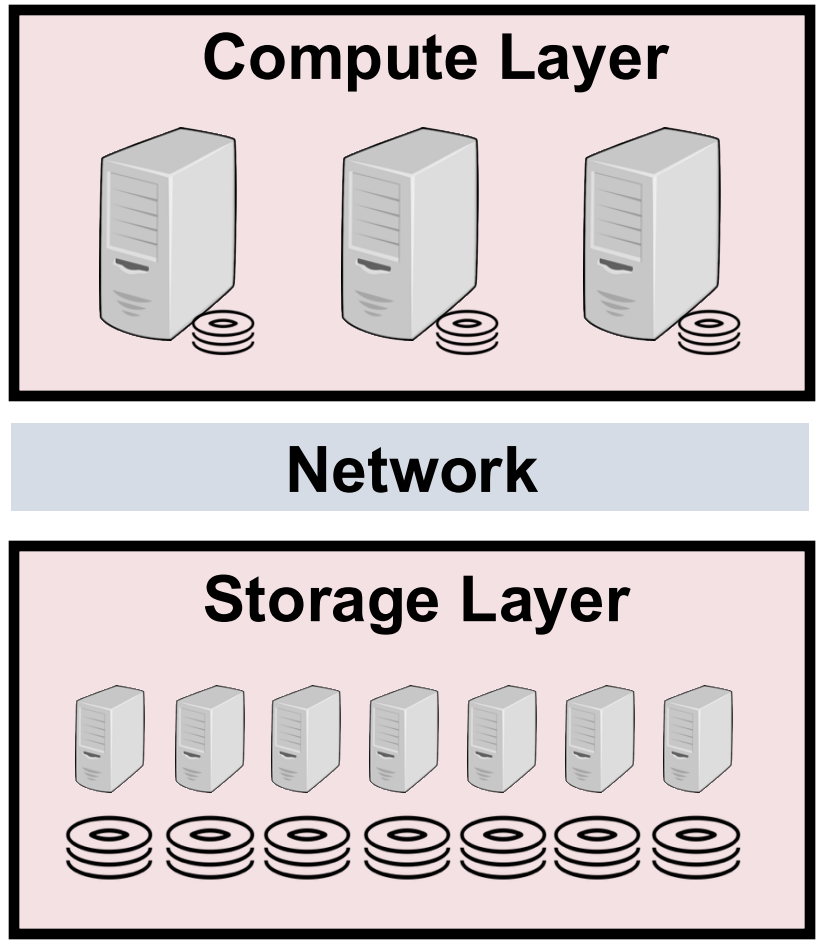


Storage Disaggregation vs. Shared Nothing?

Shared Nothing

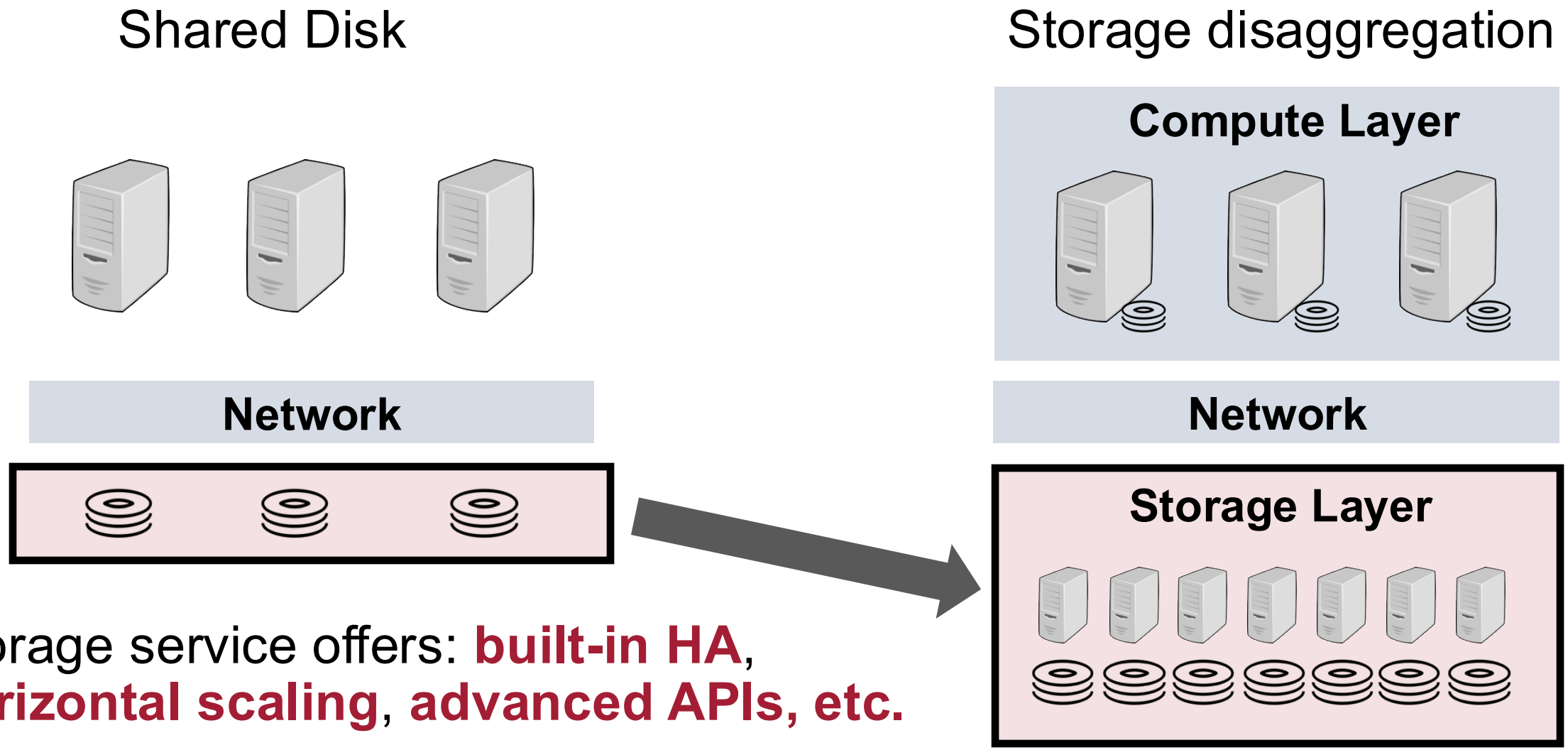


Storage disaggregation



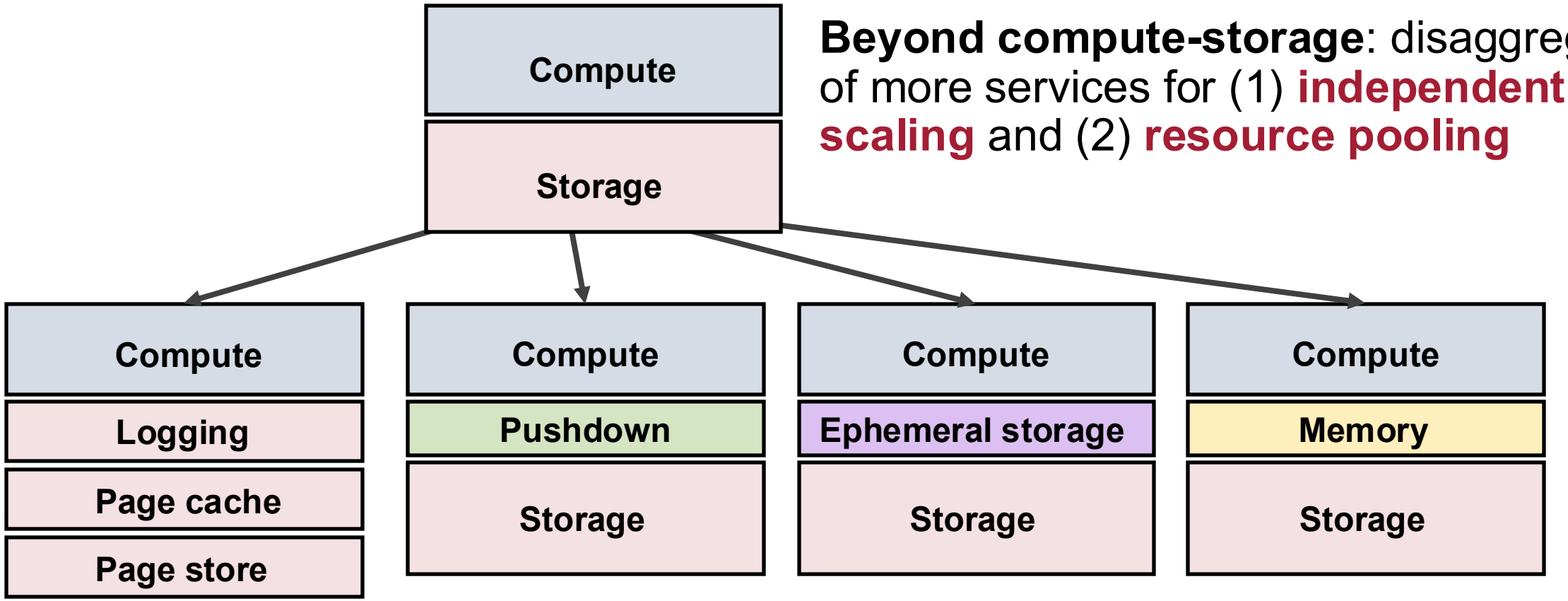
Storage disaggregation ≈ **A cluster of shared-nothing clusters**

Storage Disaggregation vs. Shared Disk?



Storage service offers: **built-in HA**, **horizontal scaling**, **advanced APIs, etc.**

Generalized Disaggregation



Beyond compute-storage: disaggregation of more services for (1) **independent scaling** and (2) **resource pooling**

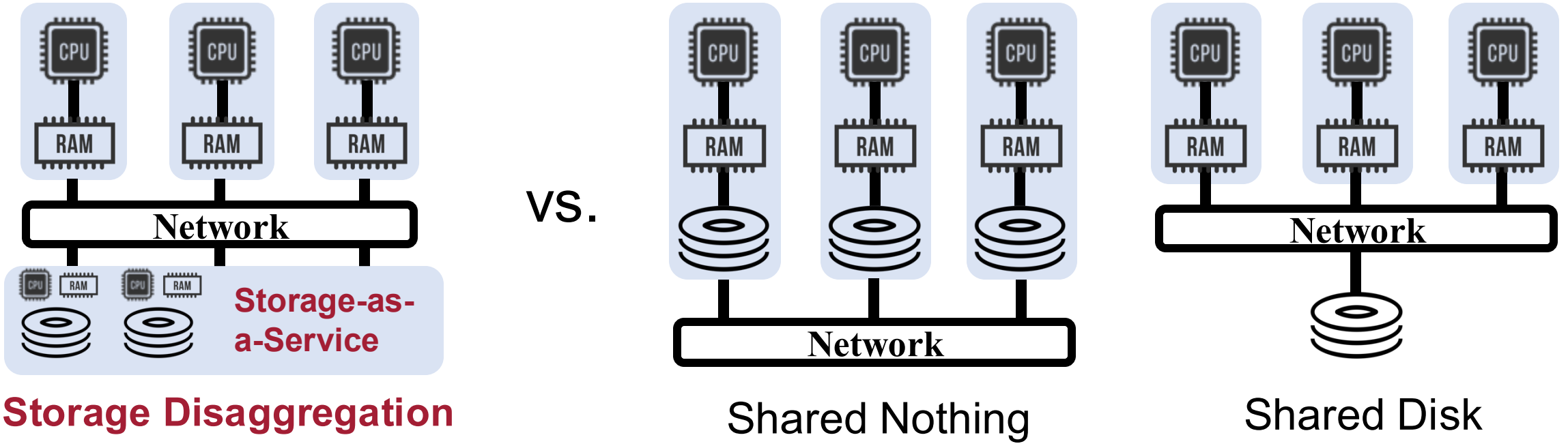
E.g., Socrates

E.g., Redshift spectrum, S3 select, pushdownDB

E.g., Snowflake

E.g., PolarDB

Network Bottleneck with Disaggregation



Storage Disaggregation | Shared Nothing | Shared Disk

Key challenge: **Network becomes a bottleneck**

– Performance of disaggregation can be 10x lower than shared-nothing [1]

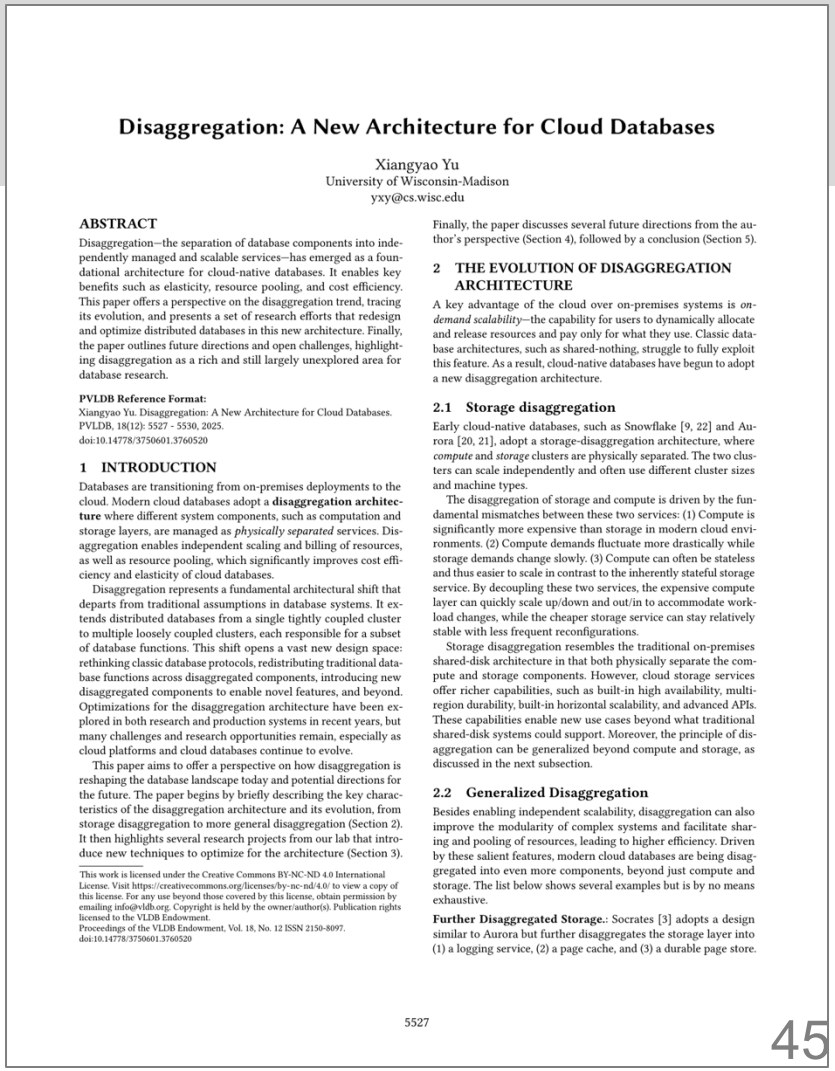
[1] Junjay Tan, et al. *Choosing A Cloud DBMS: Architectures and Tradeoffs,* VLDB'19

Disaggregation: A New Architecture for Cloud Databases

Disaggregation opens a vast, underexplored design space. Now is the time to rethink old assumptions and build new foundations for cloud databases

Xiangyao Yu

yxy@cs.wisc.edu



Disaggregation: A New Architecture for Cloud Databases

Xiangyao Yu
University of Wisconsin-Madison
yxy@cs.wisc.edu

ABSTRACT

Disaggregation—the separation of database components into independently managed and scalable services—has emerged as a foundational architecture for cloud-native databases. It enables key benefits such as elasticity, resource pooling, and cost efficiency. This paper offers a perspective on the disaggregation trend, tracing its evolution, and presents a set of research efforts that redesign and optimize distributed databases in this new architecture. Finally, the paper outlines future directions and open challenges, highlighting disaggregation as a rich and still largely unexplored area for database research.

PVLDB Reference Format:
Xiangyao Yu. Disaggregation: A New Architecture for Cloud Databases. PVLDB, 18(12): 5527 - 5530, 2025.
doi:10.14778/3750601.3760520

1 INTRODUCTION

Databases are transitioning from on-premises deployments to the cloud. Modern cloud databases adopt a **disaggregation architecture** where different system components, such as computation and storage layers, are managed as *physically separated* services. Disaggregation enables independent scaling and billing of resources, as well as resource pooling, which significantly improves cost efficiency and elasticity of cloud databases.

Disaggregation represents a fundamental architectural shift that departs from traditional assumptions in database systems. It extends distributed databases from a single tightly coupled cluster to multiple loosely coupled clusters, each responsible for a subset of database functions. This shift opens a vast new design space: rethinking classic database protocols, redistributing traditional database functions across disaggregated components, introducing new disaggregated components to enable novel features, and beyond. Optimizations for the disaggregation architecture have been explored in both research and production systems in recent years, but many challenges and research opportunities remain, especially as cloud platforms and cloud databases continue to evolve.

This paper aims to offer a perspective on how disaggregation is reshaping the database landscape today and potential directions for the future. The paper begins by briefly describing the key characteristics of the disaggregation architecture and its evolution, from storage disaggregation to more general disaggregation (Section 2). It then highlights several research projects from our lab that introduce new techniques to optimize for the architecture (Section 3). Finally, the paper discusses several future directions from the author's perspective (Section 4), followed by a conclusion (Section 5).

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3760520

2 THE EVOLUTION OF DISAGGREGATION ARCHITECTURE

A key advantage of the cloud over on-premises systems is *on-demand scalability*—the capability for users to dynamically allocate and release resources and pay only for what they use. Classic database architectures, such as shared-nothing, struggle to fully exploit this feature. As a result, cloud-native databases have begun to adopt a new disaggregation architecture.

2.1 Storage disaggregation

Early cloud-native databases, such as Snowflake [9, 22] and Aurora [20, 21], adopt a storage-disaggregation architecture, where *compute* and *storage* clusters are physically separated. The two clusters can scale independently and often use different cluster sizes and machine types.

The disaggregation of storage and compute is driven by the fundamental mismatches between these two services: (1) Compute is significantly more expensive than storage in modern cloud environments. (2) Compute demands fluctuate more drastically while storage demands change slowly. (3) Compute can often be stateless and thus easier to scale in contrast to the inherently stateful storage service. By decoupling these two services, the expensive compute layer can quickly scale up/down and out/in to accommodate workload changes, while the cheaper storage service can stay relatively stable with less frequent reconfigurations.

Storage disaggregation resembles the traditional on-premises shared-disk architecture in that both physically separate the compute and storage components. However, cloud storage services offer richer capabilities, such as built-in high availability, multi-region durability, built-in horizontal scalability, and advanced APIs. These capabilities enable new use cases beyond what traditional shared-disk systems could support. Moreover, the principle of disaggregation can be generalized beyond compute and storage, as discussed in the next subsection.

2.2 Generalized Disaggregation

Besides enabling independent scalability, disaggregation can also improve the modularity of complex systems and facilitate sharing and pooling of resources, leading to higher efficiency. Driven by these salient features, modern cloud databases are being disaggregated into even more components, beyond just compute and storage. The list below shows several examples but is by no means exhaustive.

Further Disaggregated Storage.: Socrates [3] adopts a design similar to Aurora but further disaggregates the storage layer into (1) a logging service, (2) a page cache, and (3) a durable page store.

5527

Q/A – Parallel Database

- Why shared-nothing succeed and shared-memory/disk failed?
- Main obstacles to parallel query execution optimization?
- How are split-operator routing decisions implemented at runtime?
- Which production systems deployed dynamic repartitioning?

Before Next Lecture

Submit review for

Bobbi Yogatama, et al., Rethinking Analytical Processing in the GPU Era. arXiv 2025