Carnegie Mellon University

13 | Query Execution – Part II

Intro to Database Systems
15-445/15-645
Fall 2019

AP Andy Pavlo
Computer Science
Carnegie Mellon University

ADMINISTRIVIA

Homework #3 is due Today @ 11:59pm

Mid-Term Exam is Wed Oct 16th @ 12:00pm

Project #2 is due Sun Oct 20^{th} @ 11:59pm

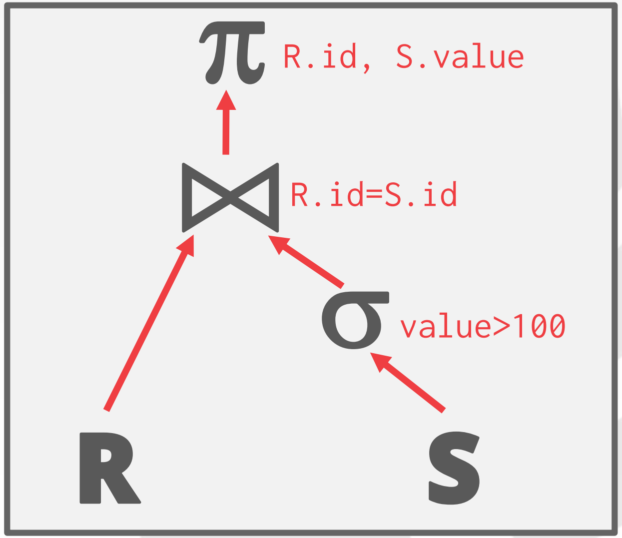
QUERY EXECUTION

We discussed last class how to compose operators together to execute a query plan.

We assumed that the queries execute with a single worker (e.g., thread).

We now need to talk about how to execute with multiple workers…

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```



WHY CARE ABOUT PARALLEL EXECUTION?

Increased performance.
→ Throughput
→ Latency

Increased responsiveness and availability.

Potentially lower ***total cost of ownership*** (TCO).

PARALLEL VS. DISTRIBUTED

Database is spread out across multiple **resources** to improve different aspects of the DBMS.

Appears as a single database instance to the application.

→ SQL query for a single-resource DBMS should generate same result on a parallel or distributed DBMS.

PARALLEL VS. DISTRIBUTED

Parallel DBMSs:

→ Resources are physically close to each other.
→ Resources communicate with high-speed interconnect.
→ Communication is assumed to cheap and reliable.

Distributed DBMSs:

→ Resources can be far from each other.
→ Resources communicate using slow(er) interconnect.
→ Communication cost and problems cannot be ignored.

TODAY'S AGENDA

Process Models

Execution Parallelism

I/O Parallelism

PROCESS MODEL

A DBMS's **process model** defines how the system is architected to support concurrent requests from a multi-user application.

A **worker** is the DBMS component that is responsible for executing tasks on behalf of the client and returning the results.

PROCESS MODELS

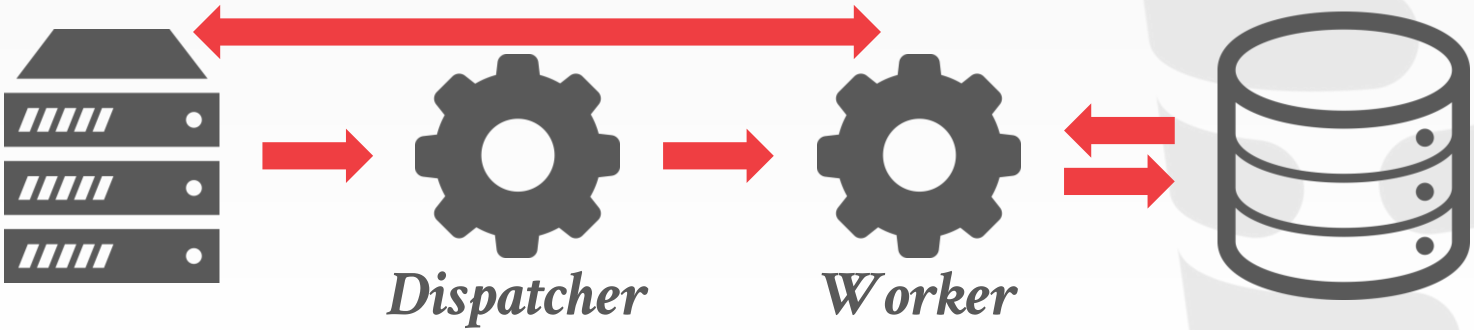
Approach #1: Process per DBMS Worker

Approach #2: Process Pool

Approach #3: Thread per DBMS Worker

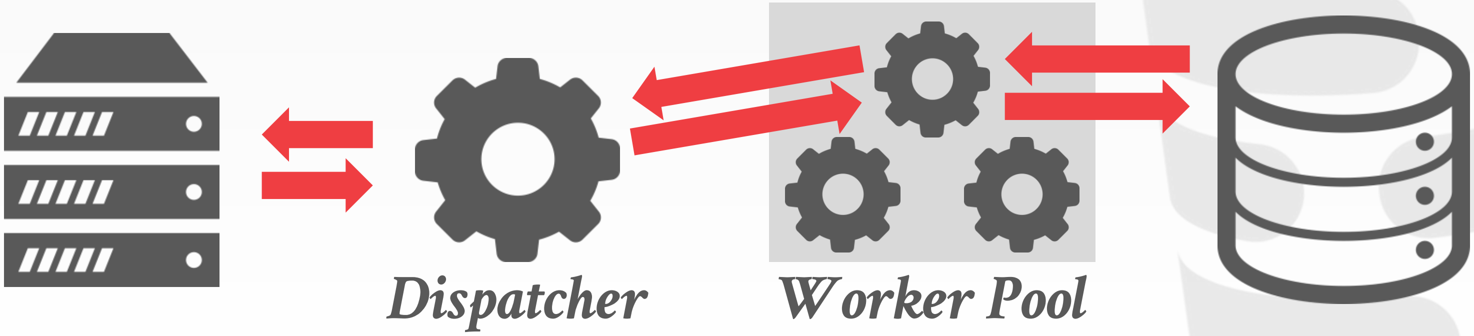
PROCESS PER WORKER

Each worker is a separate OS process.
→ Relies on OS scheduler.
→ Use shared-memory for global data structures.
→ A process crash doesn't take down entire system.
→ Examples: IBM DB2, Postgres, Oracle



PROCESS POOL

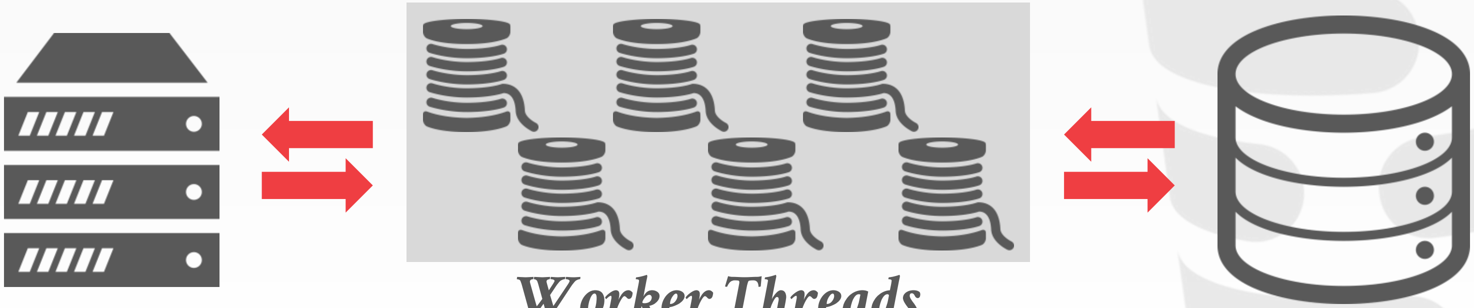
A worker uses any process that is free in a pool
→ Still relies on OS scheduler and shared memory.
→ Bad for CPU cache locality.
→ Examples: IBM DB2, Postgres (2015)



THREAD PER WORKER

Single process with multiple worker threads.

→ DBMS manages its own scheduling.
→ May or may not use a dispatcher thread.
→ Thread crash (may) kill the entire system.
→ Examples: IBM DB2, MSSQL, MySQL, Oracle (2014)



Worker Threads

PROCESS MODELS

Using a multi-threaded architecture has several advantages:

→ Less overhead per context switch.

→ Do not have to manage shared memory.

The thread per worker model does **not** mean that the DBMS supports intra-query parallelism.

Andy is not aware of any new DBMS from last 10 years that doesn't use threads unless they are Postgres forks.

SCHEDULING

For each query plan, the DBMS decides where, when, and how to execute it.

- → How many tasks should it use?
- → How many CPU cores should it use?
- → What CPU core should the tasks execute on?
- → Where should a task store its output?

The DBMS ***always*** knows more than the OS.

INTER- VS. INTRA-QUERY PARALLELISM

Inter-Query: Different queries are executed concurrently.

→ Increases throughput & reduces latency.

Intra-Query: Execute the operations of a single query in parallel.

→ Decreases latency for long-running queries.

INTER-QUERY PARALLELISM

Improve overall performance by allowing multiple queries to execute simultaneously.

If queries are read-only, then this requires little coordination between queries.

Lecture 16

If multiple queries are updating the database at the same time, then this is hard to do correctly…

INTRA-QUERY PARALLELISM

Improve the performance of a single query by executing its operators in parallel.

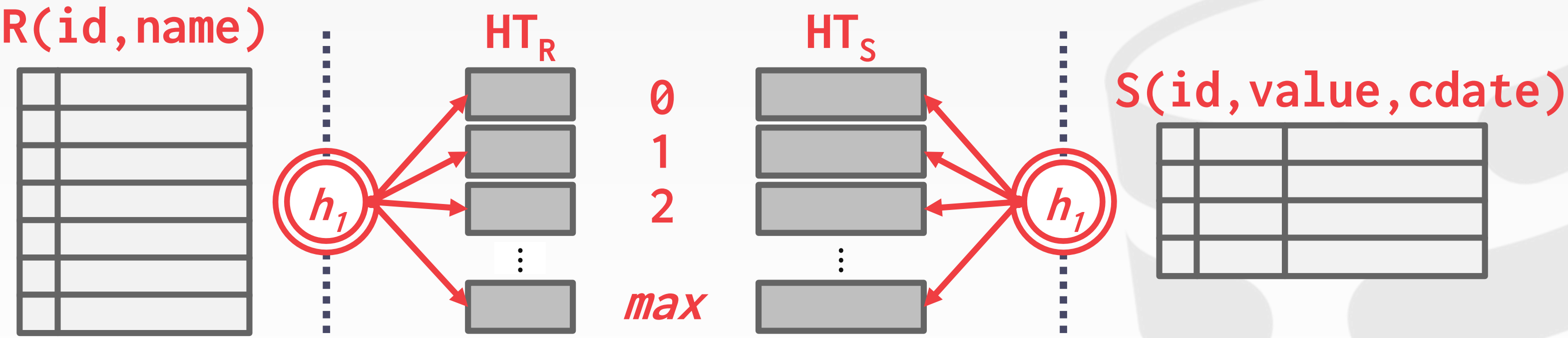
Think of organization of operators in terms of a ***producer/consumer*** paradigm.

There are parallel algorithms for every relational operator.

→ Can either have multiple threads access centralized data structures or use partitioning to divide work up.

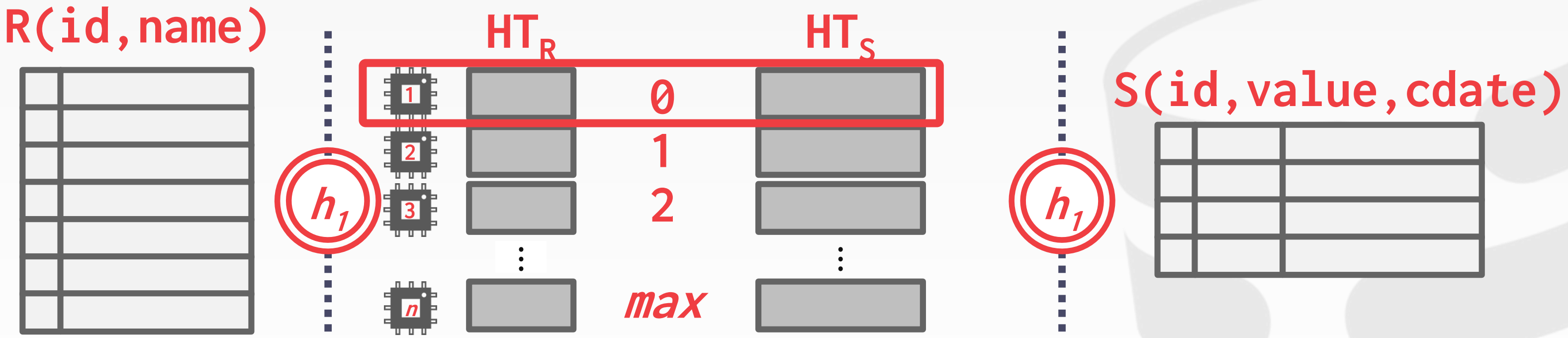
PARALLEL GRACE HASH JOIN

Use a separate worker to perform the join for each level of buckets for **R** and **S** after partitioning.



PARALLEL GRACE HASH JOIN

Use a separate worker to perform the join for each level of buckets for **R** and **S** after partitioning.



INTRA-QUERY PARALLELISM

Approach #1: Intra-Operator (Horizontal)

Approach #2: Inter-Operator (Vertical)

Approach #3: Bushy

INTRA-OPERATOR PARALLELISM

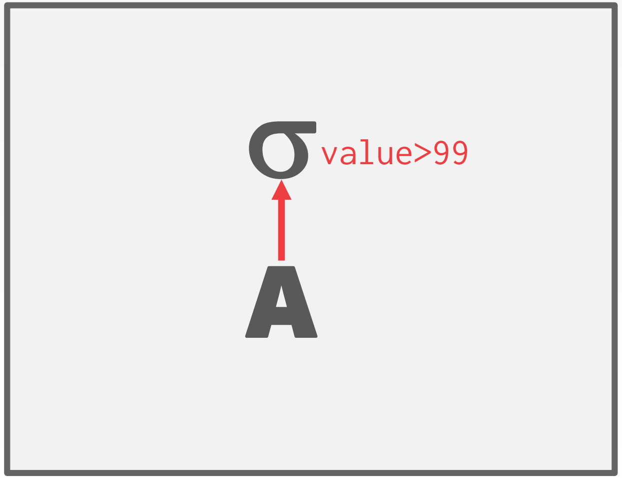
Approach #1: Intra-Operator (Horizontal)

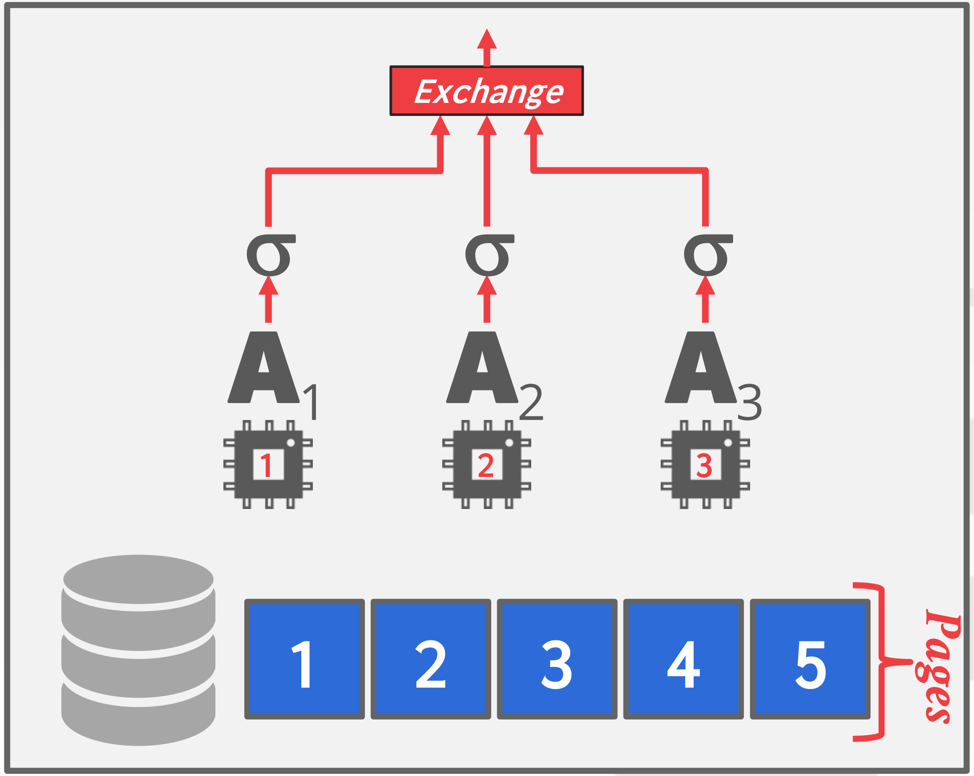
→ Decompose operators into independent **fragments** that perform the same function on different subsets of data.

The DBMS inserts an **exchange** operator into the query plan to coalesce results from children operators.

INTRA-OPERATOR PARALLELISM

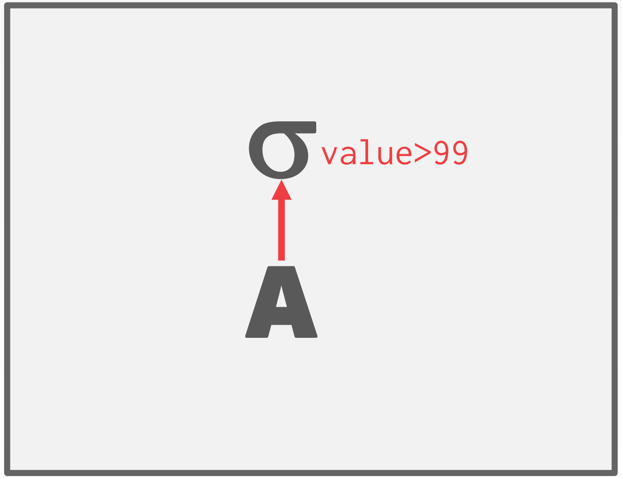
```
SELECT * FROM A
 WHERE A.value > 99
```

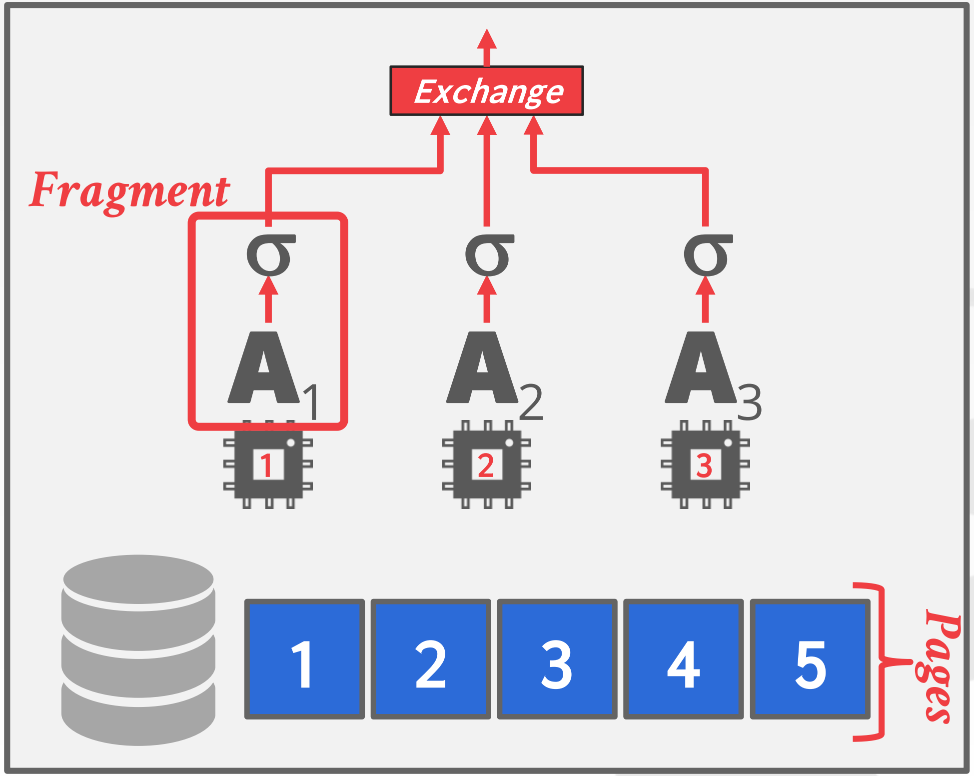




INTRA-OPERATOR PARALLELISM

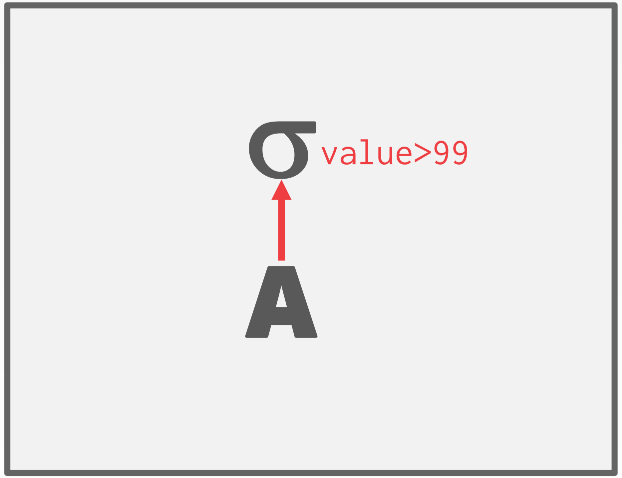
```
SELECT * FROM A
 WHERE A.value > 99
```

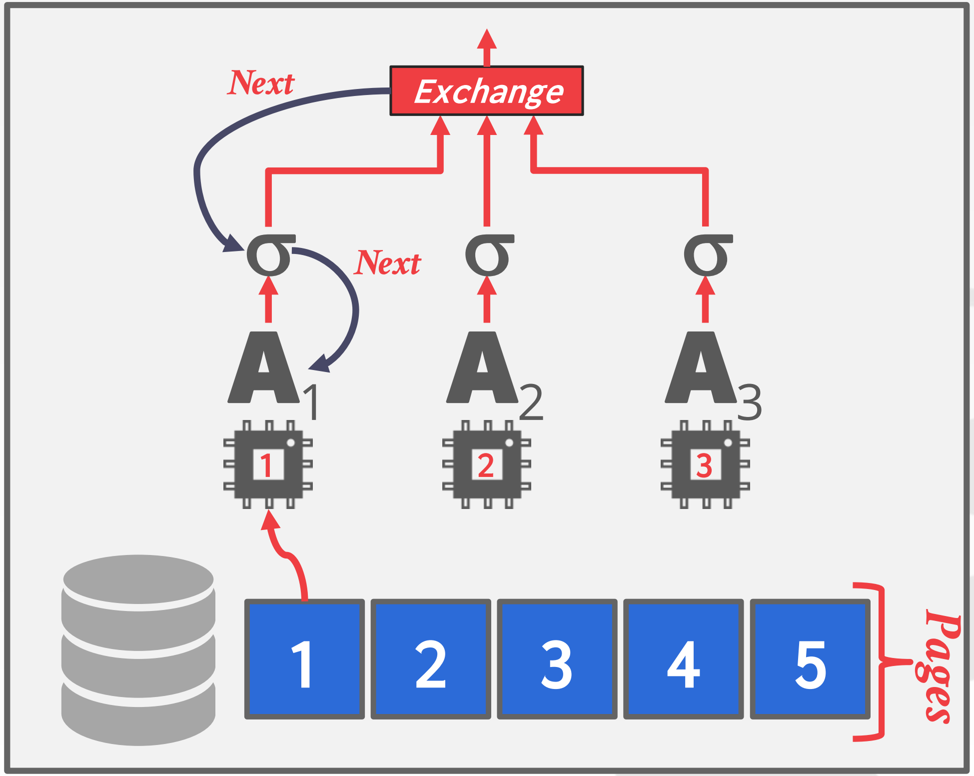




INTRA-OPERATOR PARALLELISM

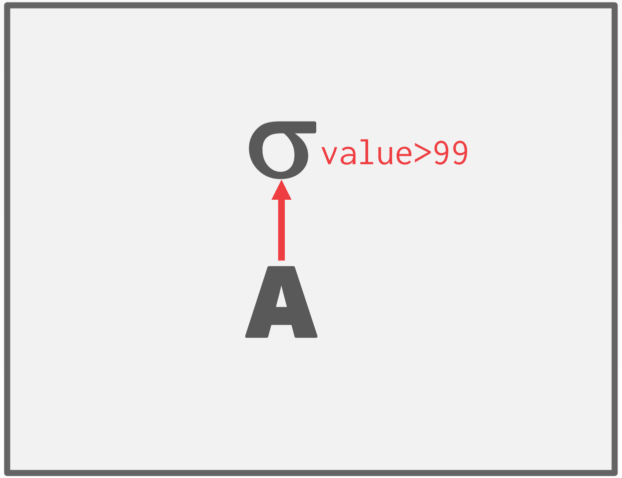
```
SELECT * FROM A
 WHERE A.value > 99
```

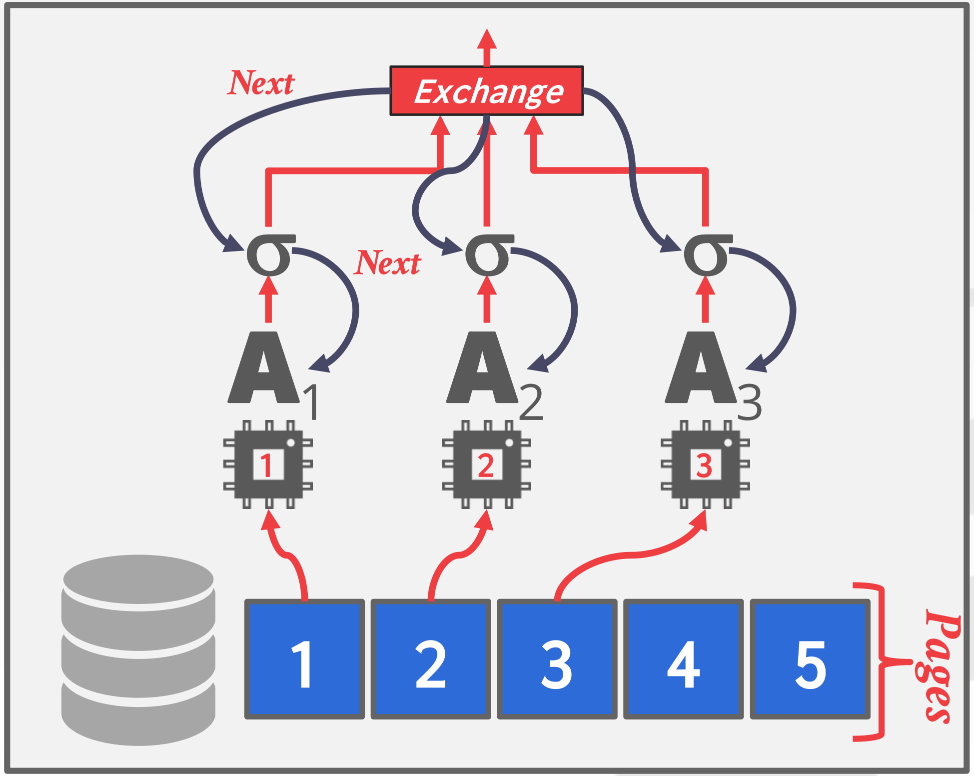




INTRA-OPERATOR PARALLELISM

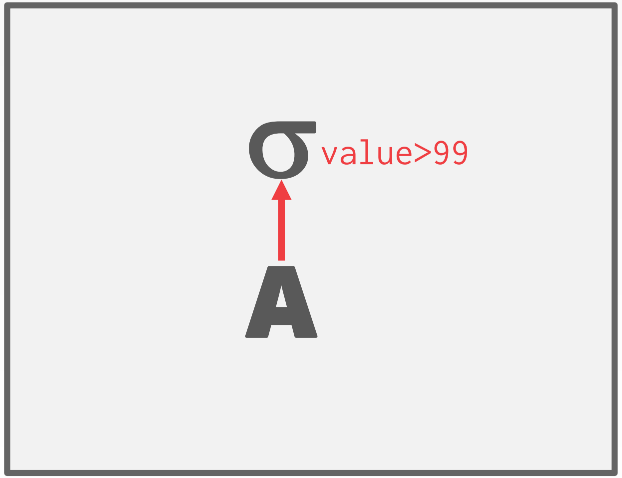
```
SELECT * FROM A
 WHERE A.value > 99
```

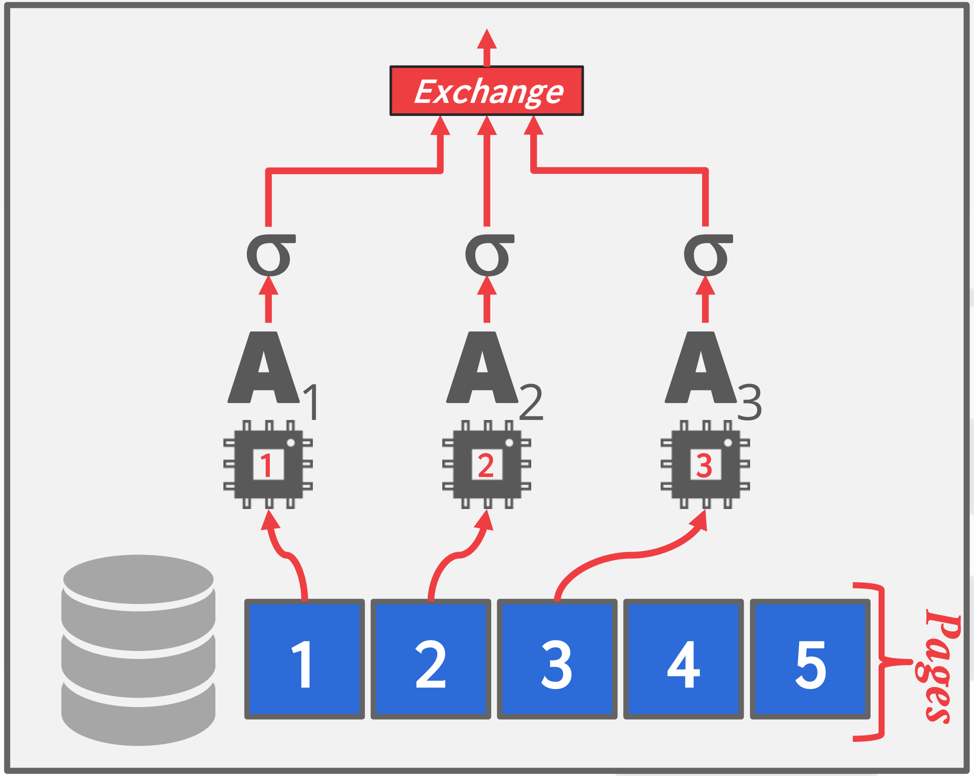




INTRA-OPERATOR PARALLELISM

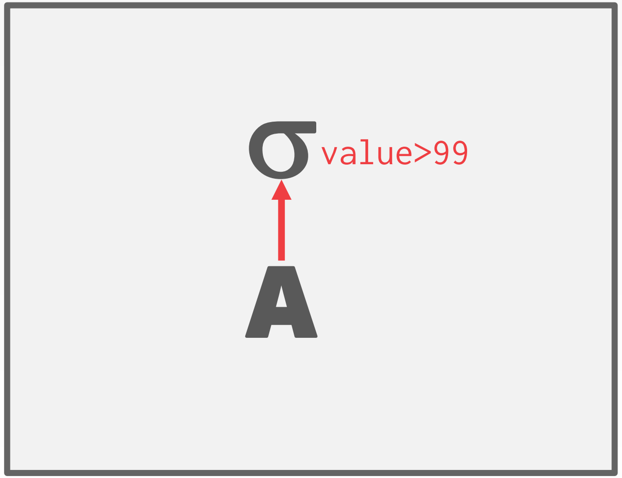
```
SELECT * FROM A
 WHERE A.value > 99
```

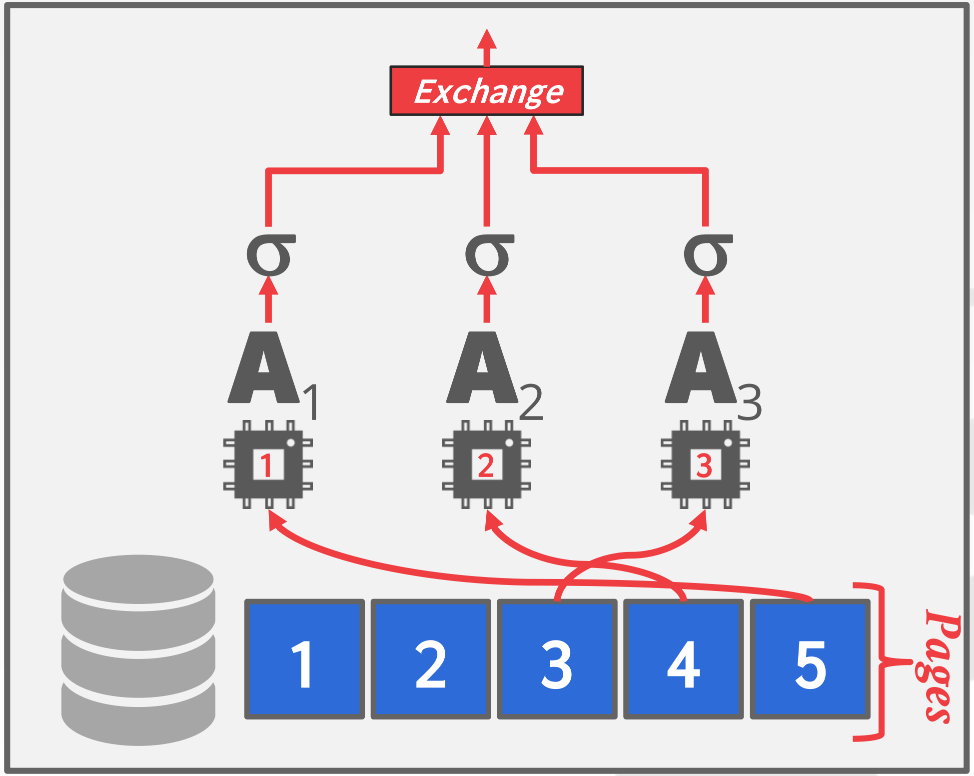




INTRA-OPERATOR PARALLELISM

```
SELECT * FROM A
 WHERE A.value > 99
```





EXCHANGE OPERATOR

Exchange Type #1 – Gather

→ Combine the results from multiple workers into a single output stream.

→ Query plan root must always be a gather exchange.

Exchange Type #2 – Repartition

→ Reorganize multiple input streams across multiple output streams.

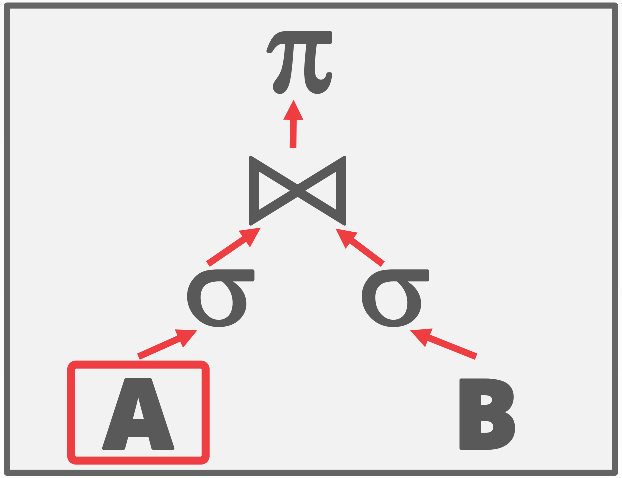
Exchange Type #3 – Distribute

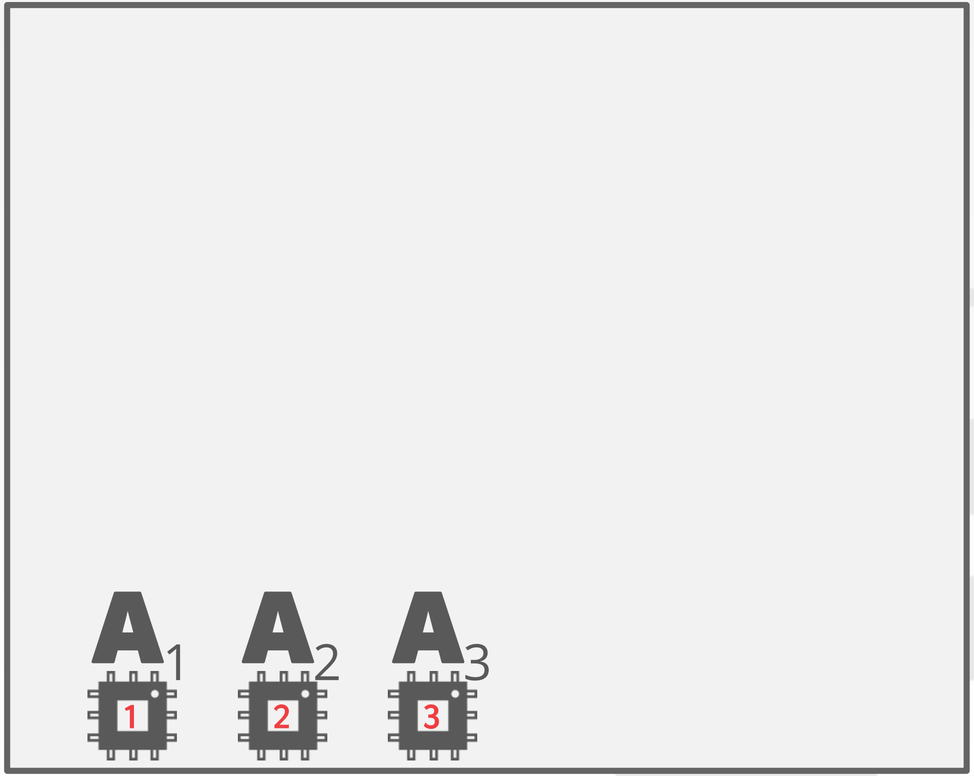
→ Split a single input stream into multiple output streams.

Source: Craig Freedman

INTRA-OPERATOR PARALLELISM

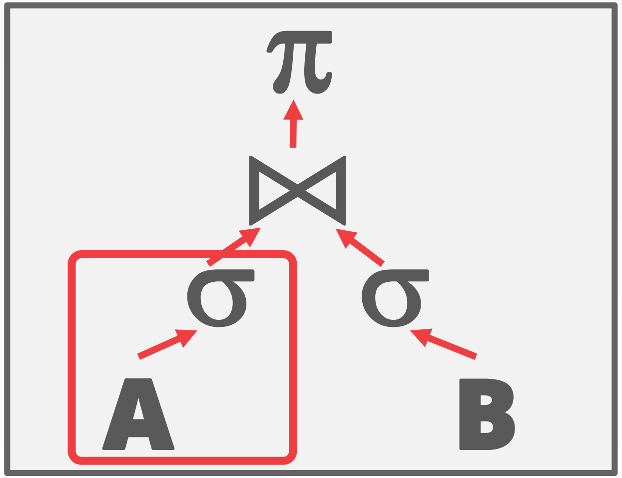
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

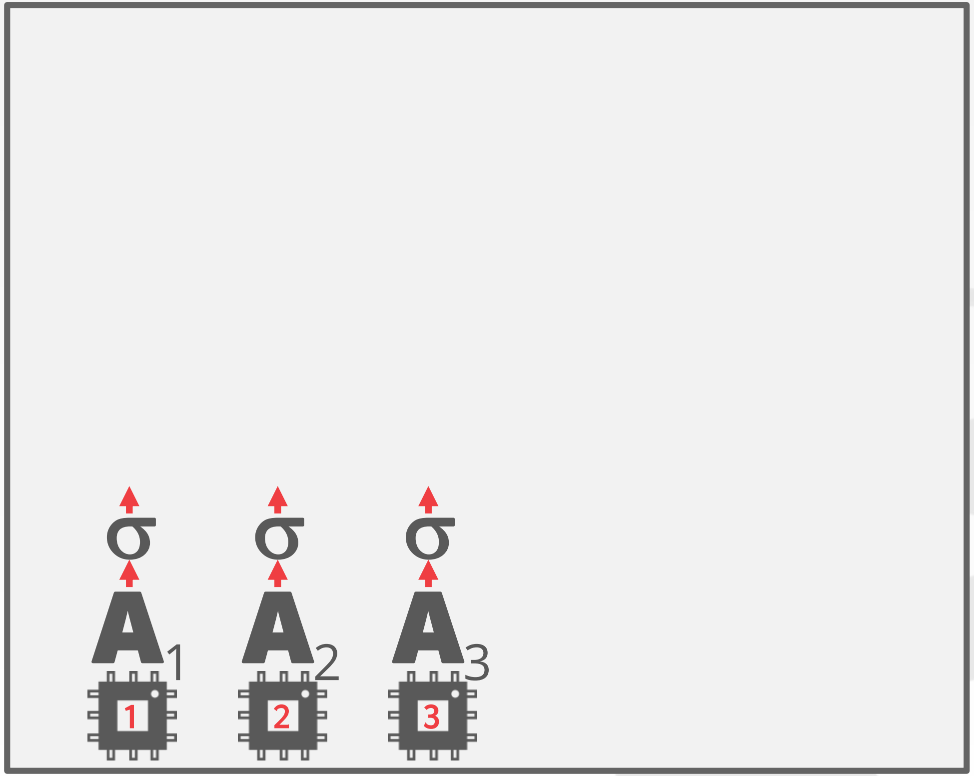




INTRA-OPERATOR PARALLELISM

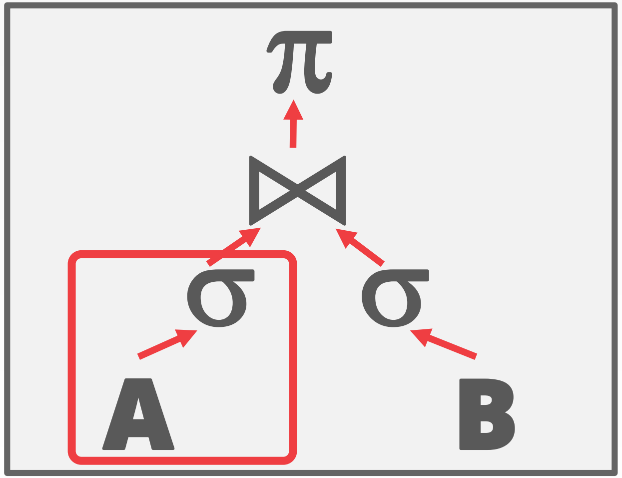
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

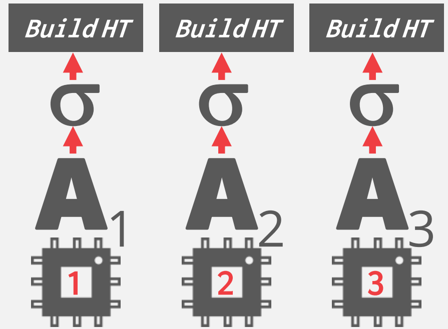




INTRA-OPERATOR PARALLELISM

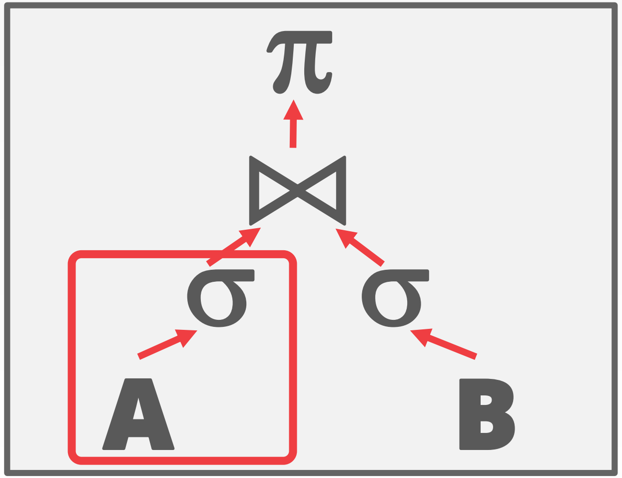
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

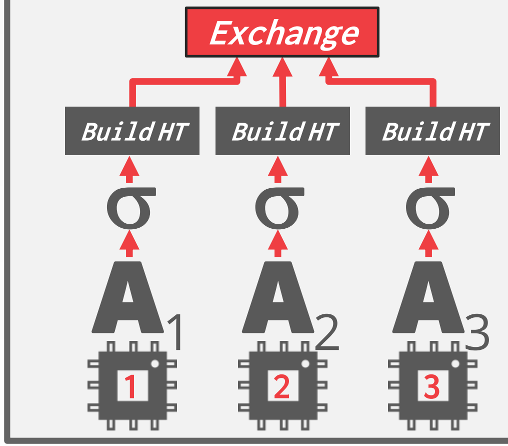




INTRA-OPERATOR PARALLELISM

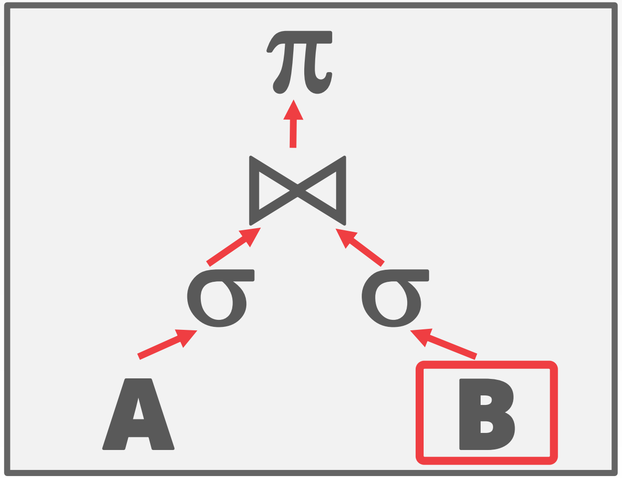
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

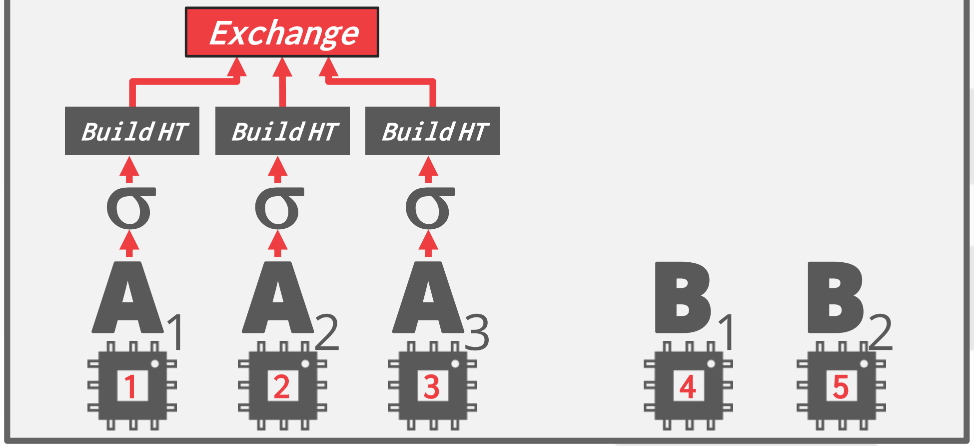




INTRA-OPERATOR PARALLELISM

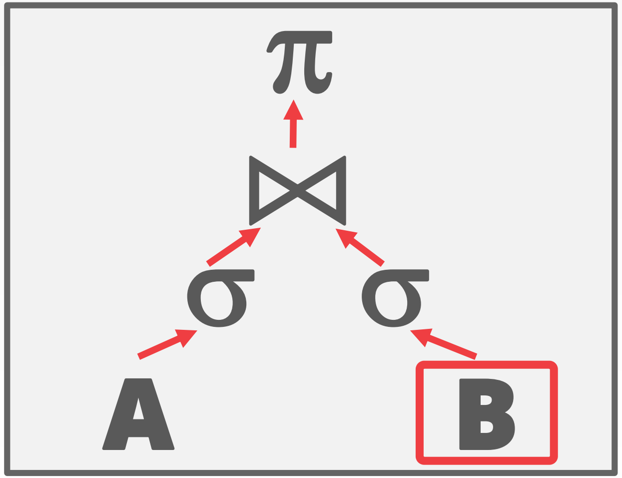
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

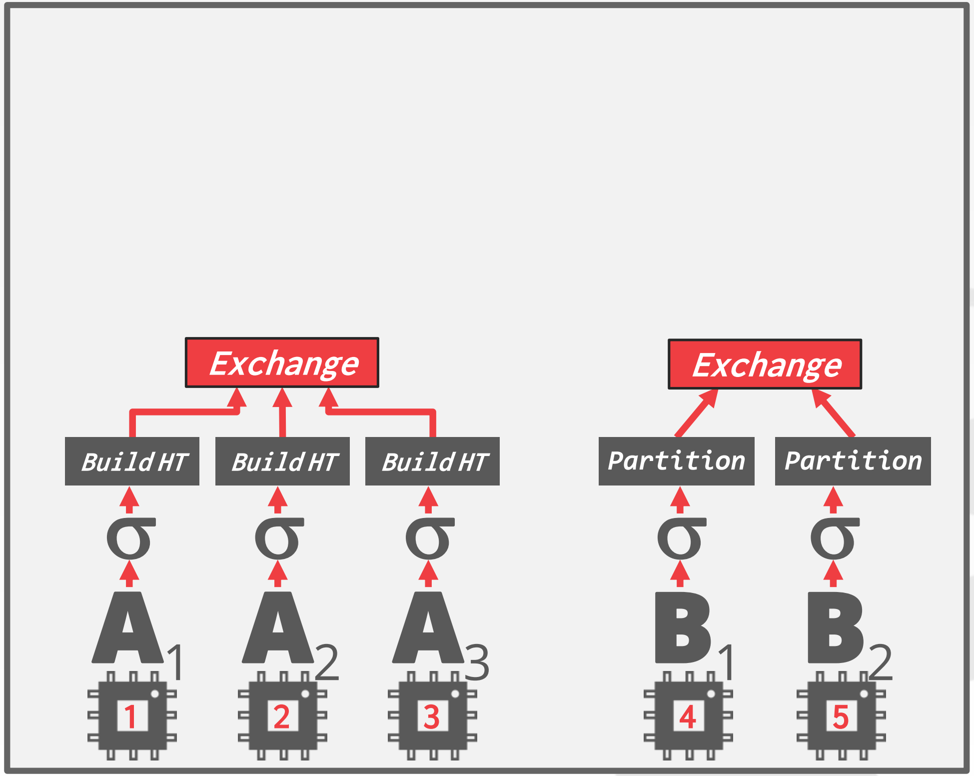




INTRA-OPERATOR PARALLELISM

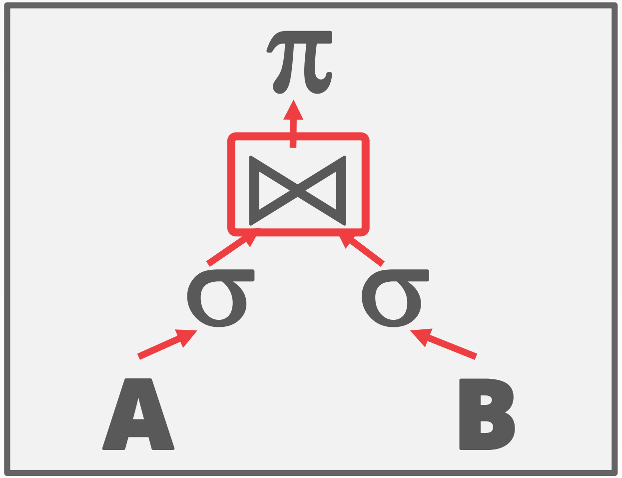
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

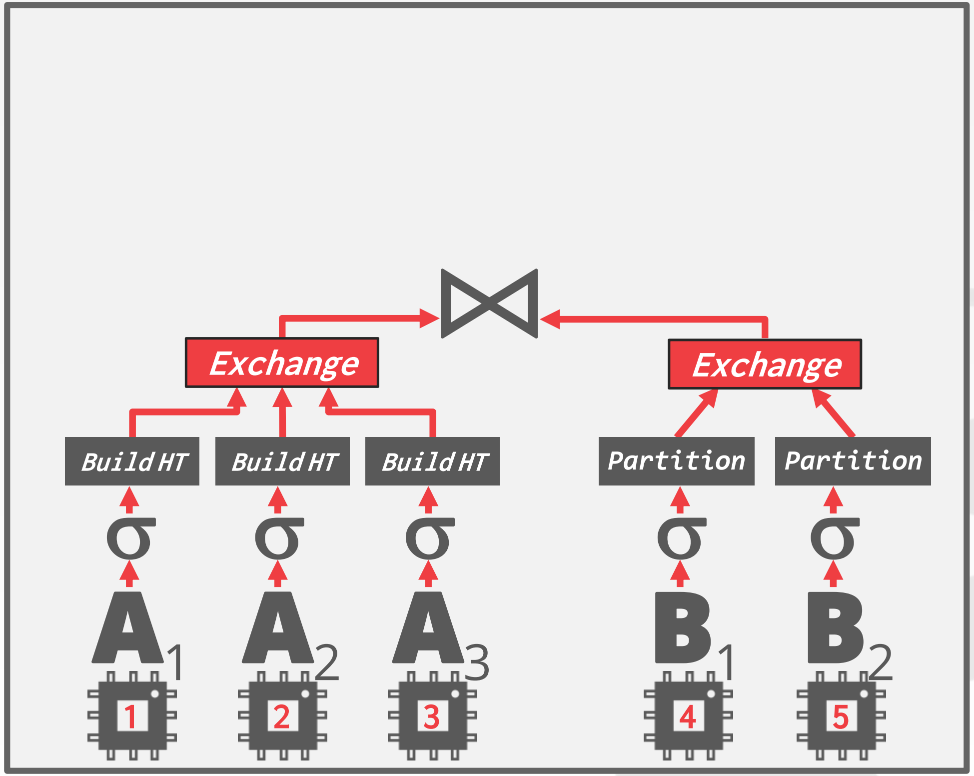




INTRA-OPERATOR PARALLELISM

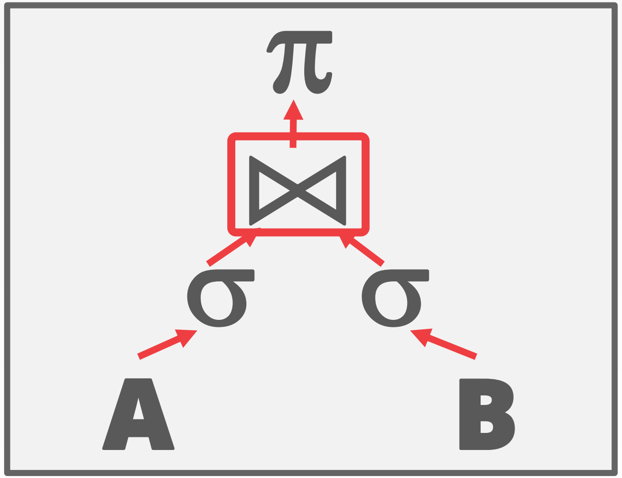
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

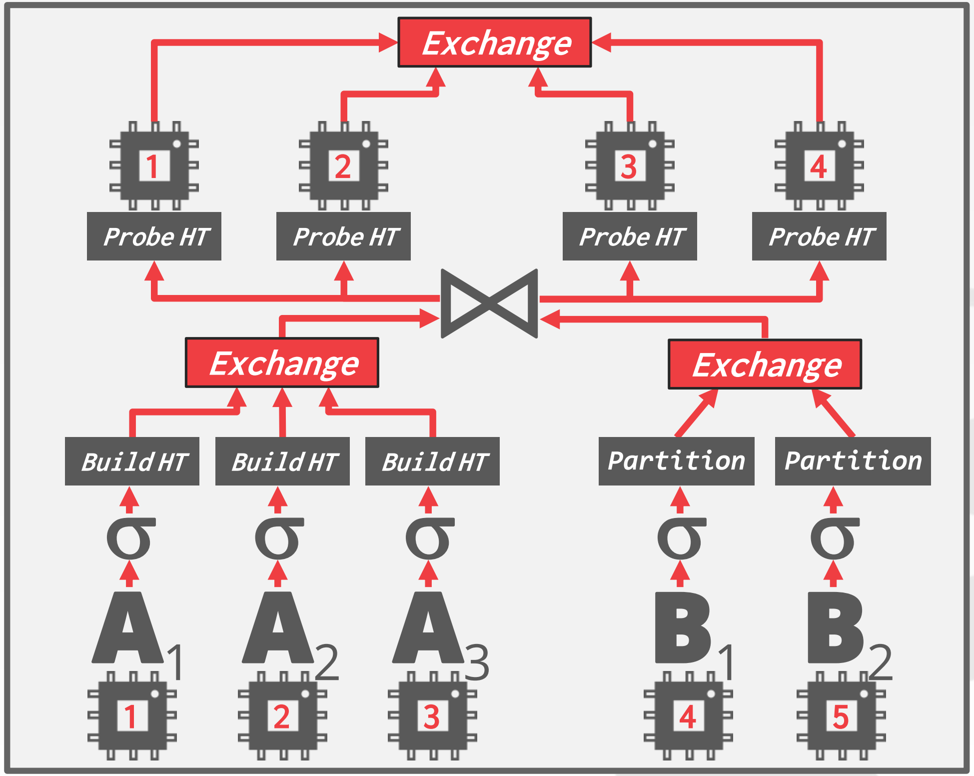




INTRA-OPERATOR PARALLELISM

```sql
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```





INTER-OPERATOR PARALLELISM

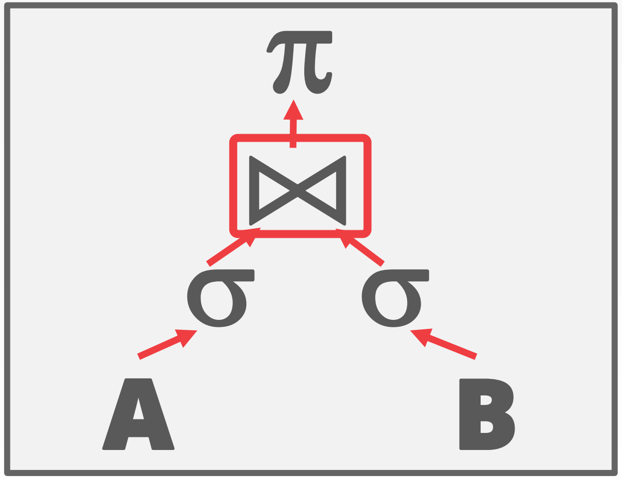
Approach #2: Inter-Operator (Vertical)

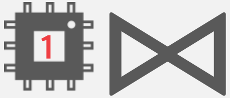
→ Operations are overlapped in order to pipeline data from one stage to the next without materialization.

Also called **pipelined parallelism**.

INTER-OPERATOR PARALLELISM

```sql
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

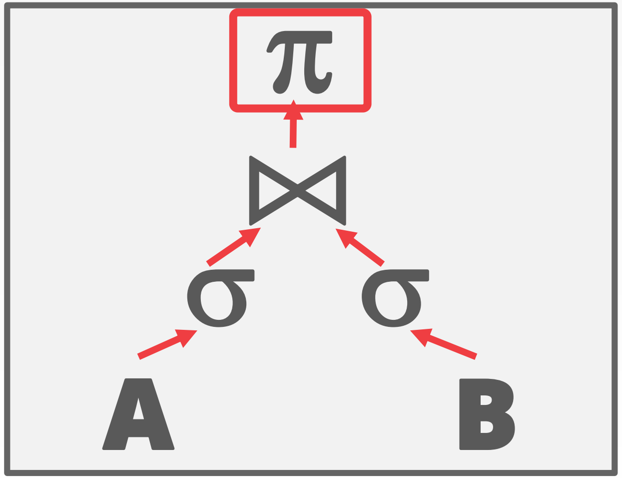


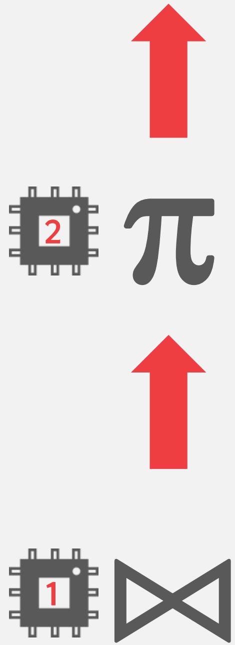



```
for r1 ∈ outer:
  for r2 ∈ inner:
    emit(r1⋈r2)
```

INTER-OPERATOR PARALLELISM

```sql
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```



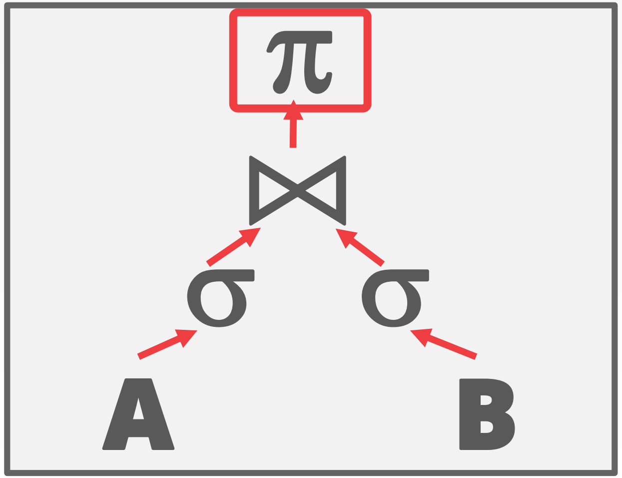


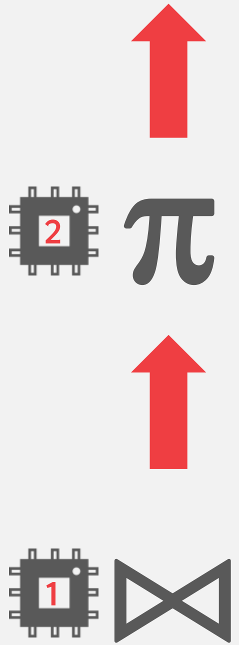

```
for r ∈ incoming:
  emit(πr)
```

```
for r1 ∈ outer:
  for r2 ∈ inner:
    emit(r1⋈r2)
```

INTER-OPERATOR PARALLELISM

```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```






```
for r ∈ incoming:
  emit(πr)
```

```
for r1 ∈ outer:
  for r2 ∈ inner:
    emit(r1⋈r2)
```

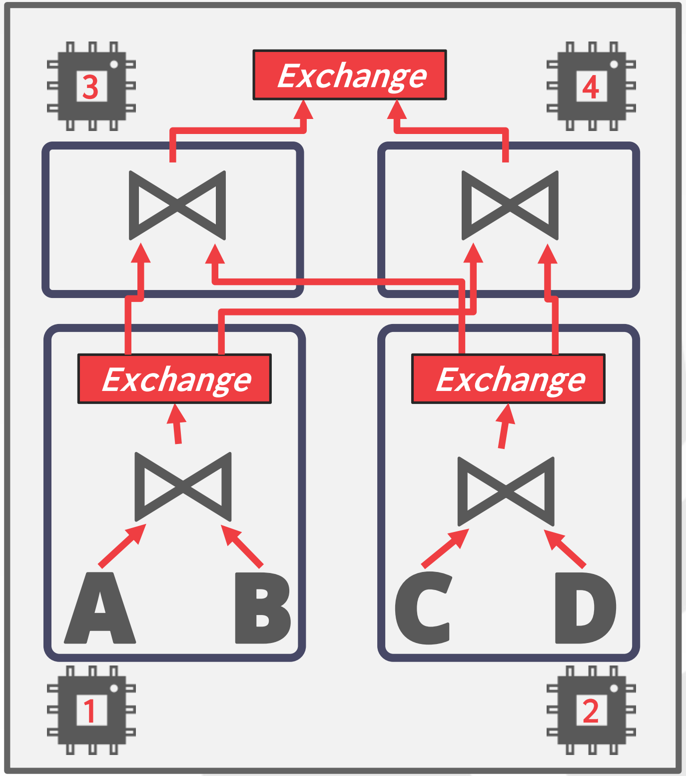
BUSHY PARALLELISM

Approach #3: Bushy Parallelism

→ Extension of inter-operator parallelism where workers execute multiple operators from different segments of a query plan at the same time.

→ Still need exchange operators to combine intermediate results from segments.

```
SELECT *
  FROM A JOIN B JOIN C JOIN D
```



OBSERVATION

Using additional processes/threads to execute queries in parallel won't help if the disk is always the main bottleneck.

→ Can make things worse if each worker is reading different segments of disk.

I/O PARALLELISM

Split the DBMS installation across multiple storage devices.

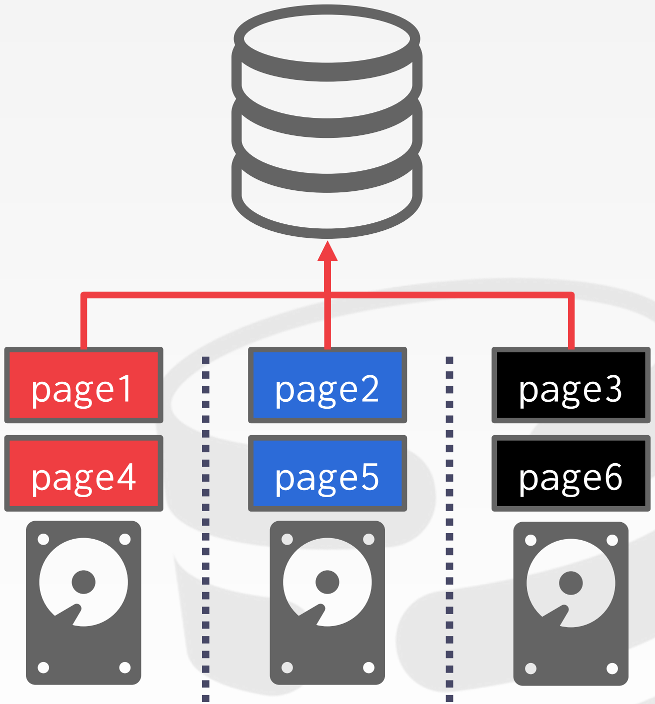
→ Multiple Disks per Database
→ One Database per Disk
→ One Relation per Disk
→ Split Relation across Multiple Disks

MULTI-DISK PARALLELISM

Configure OS/hardware to store the DBMS's files across multiple storage devices.

→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.



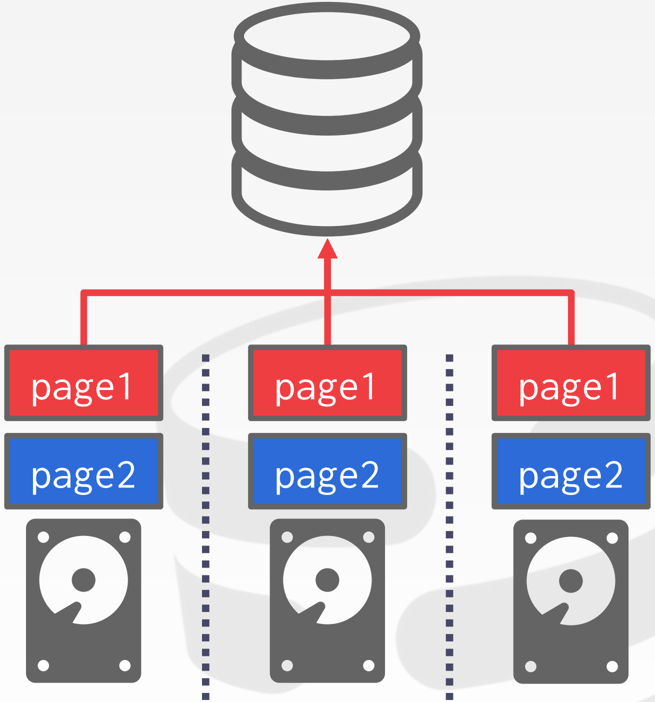
RAID 0 (Stripping)

MULTI-DISK PARALLELISM

Configure OS/hardware to store the DBMS's files across multiple storage devices.

→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.



RAID 1 (Mirroring)

DATABASE PARTITIONING

Some DBMSs allow you specify the disk location of each individual database.

→ The buffer pool manager maps a page to a disk location.

This is also easy to do at the filesystem level if the DBMS stores each database in a separate directory.

→ The log file might be shared though

PARTITIONING

Split single logical table into disjoint physical segments that are stored/managed separately.

Ideally partitioning is transparent to the application.

→ The application accesses logical tables and does not care how things are stored.
→ Not always true in distributed DBMSs.

VERTICAL PARTITIONING

Store a table's attributes in a separate location (e.g., file, disk volume).

Have to store tuple information to reconstruct the original record.

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Tuple#1	attr1	attr2	attr3	attr4
Tuple#2	attr1	attr2	attr3	attr4
Tuple#3	attr1	attr2	attr3	attr4
Tuple#4	attr1	attr2	attr3	attr4

VERTICAL PARTITIONING

Store a table's attributes in a separate location (e.g., file, disk volume).

Have to store tuple information to reconstruct the original record.

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Partition #1

Tuple#1	attr1	attr2	attr3
Tuple#2	attr1	attr2	attr3
Tuple#3	attr1	attr2	attr3
Tuple#4	attr1	attr2	attr3

Partition #2

Tuple#1	attr4
Tuple#2	attr4
Tuple#3	attr4
Tuple#4	attr4

HORIZONTAL PARTITIONING

Divide the tuples of a table up into disjoint segments based on some partitioning key.

→ Hash Partitioning
→ Range Partitioning
→ Predicate Partitioning

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Tuple#1	attr1	attr2	attr3	attr4
Tuple#2	attr1	attr2	attr3	attr4
Tuple#3	attr1	attr2	attr3	attr4
Tuple#4	attr1	attr2	attr3	attr4

HORIZONTAL PARTITIONING

Divide the tuples of a table up into disjoint segments based on some partitioning key.

→ Hash Partitioning
→ Range Partitioning
→ Predicate Partitioning

```
CREATE TABLE foo (
  attr1 INT,
  attr2 INT,
  attr3 INT,
  attr4 TEXT
);
```

Partition #1

Tuple#1	attr1	attr2	attr3	attr4
Tuple#2	attr1	attr2	attr3	attr4

Partition #2

Tuple#3	attr1	attr2	attr3	attr4
Tuple#4	attr1	attr2	attr3	attr4

CONCLUSION

Parallel execution is important.

(Almost) every DBMS support this.

This is really hard to get right.

→ Coordination Overhead
→ Scheduling
→ Concurrency Issues
→ Resource Contention

MIDTERM EXAM

Who: You

What: Midterm Exam

When: Wed Oct 16^{th} @ 12:00pm - 1:20pm

Where: MM 103

Why: https://youtu.be/GHPB1eCROSA

Covers up to Query Execution II (inclusive).

→ Please email Andy if you need special accommodations.

→ https://15445.courses.cs.cmu.edu/fall2019/midterm-guide.html

MIDTERM EXAM

What to bring:

→ CMU ID
→ Calculator
→ One 8.5x11" page of handwritten notes (double-sided)

What not to bring:

→ Live animals
→ Your wet laundry
→ Votive Candles (aka "Jennifer Lopez" Candles)

RELATIONAL MODEL

Integrity Constraints

Relation Algebra

SQL

Basic operations:
- → SELECT / INSERT / UPDATE / DELETE
- → WHERE predicates
- → Output control

More complex operations:
- → Joins
- → Aggregates
- → Common Table Expressions

STORAGE

Buffer Management Policies
→ LRU / MRU / CLOCK

On-Disk File Organization
→ Heaps
→ Linked Lists

Page Layout
→ Slotted Pages
→ Log-Structured

HASHING

Static Hashing
→ Linear Probing
→ Robin Hood
→ Cuckoo Hashing

Dynamic Hashing
→ Extendible Hashing
→ Linear Hashing

TREE INDEXES

B+Tree
- → Insertions / Deletions
- → Splits / Merges
- → Difference with B-Tree
- → Latch Crabbing / Coupling

Radix Trees

SORTING

Two-way External Merge Sort

General External Merge Sort

Cost to sort different data sets with different number of buffers.

JOINS

Nested Loop Variants

Sort-Merge

Hash

Execution costs under different conditions.

QUERY PROCESSING

Processing Models
→ Advantages / Disadvantages

Parallel Execution
→ Inter- vs. Intra-Operator Parallelism

NEXT CLASS

Query Planning & Optimization