Carnegie Mellon University

12 Query Execution – Part II

Intro to Database Systems
15-445/15-645
Fall 2021

Andrew Crotty
Computer Science
Carnegie Mellon University

ADMINISTRIVIA

Project #2 is due Sunday, Oct 17th @ 11:59pm

Homework #3 is due Sunday, Oct 24th @ 11:59pm

Midterm Exam is Wednesday, Oct 13th

→ During regular class time @ 3:05-4:25pm
→ Open book / open notes
→ Will include all material covered before midterm
→ See Piazza post for more details

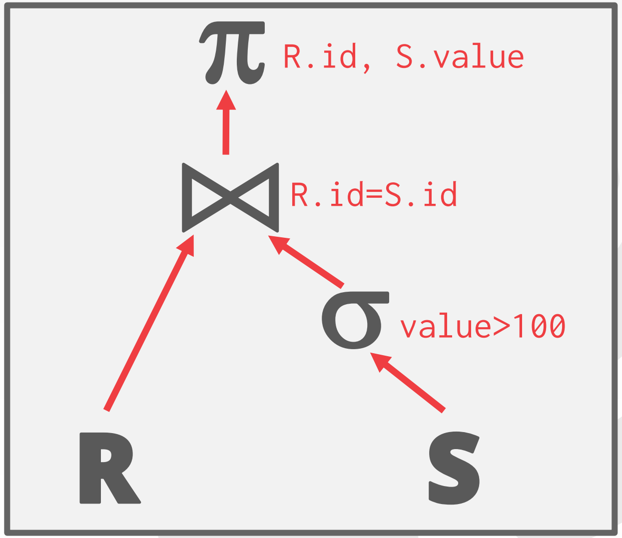
QUERY EXECUTION

We discussed in the last class how to compose operators together into a plan to execute an arbitrary query.

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

We will now discuss how to execute queries using 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?

WHY CARE ABOUT PARALLEL EXECUTION?

Increased performance

→ Throughput

→ Latency

WHY CARE ABOUT PARALLEL EXECUTION?

Increased performance
→ Throughput
→ Latency

Increased responsiveness and availability

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 logical database instance to the application, regardless of physical organization.
→ 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 over high-speed interconnect.
→ Communication is assumed to be 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 MODEL

Approach #1: Process per DBMS Worker

Approach #2: Process Pool

Approach #3: Thread per DBMS Worker

PROCESS PER WORKER

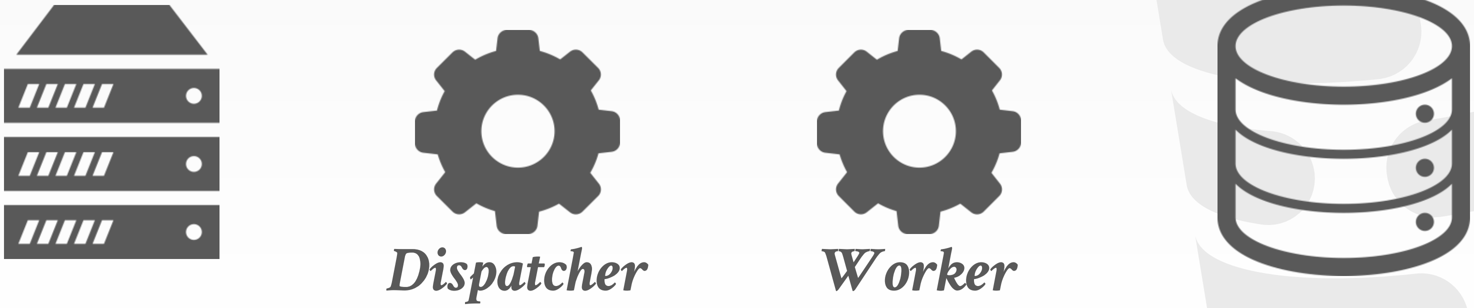
Each worker is a separate OS process.

→ 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 PER WORKER

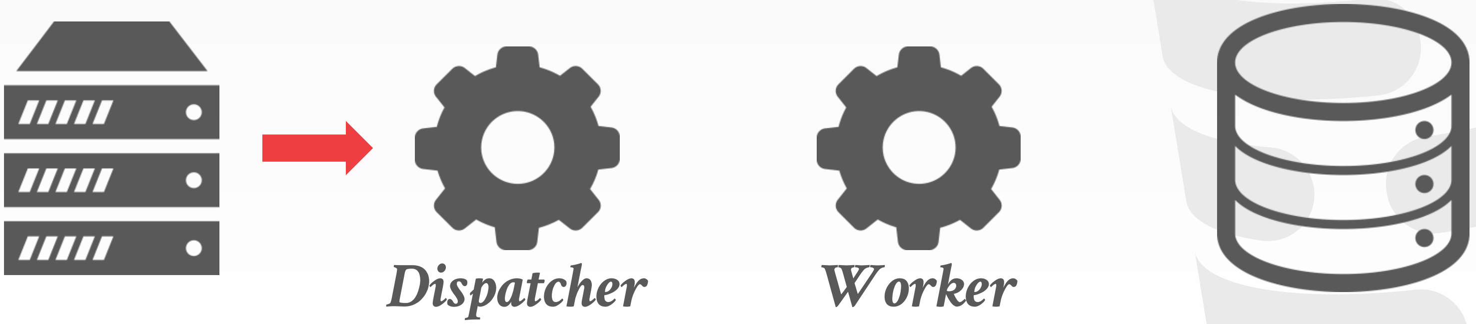
Each worker is a separate OS process.

→ 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 PER WORKER

IBM DB2

ORACLE

PostgreSQL

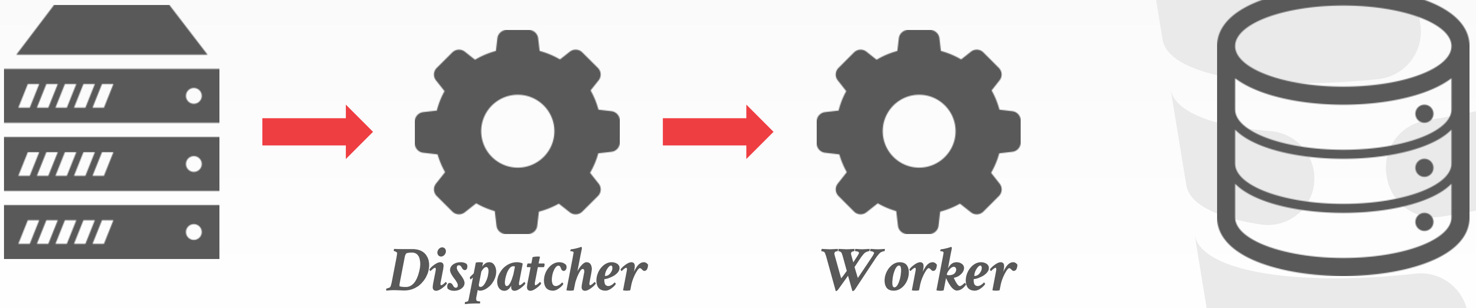
Each worker is a separate OS process.

→ 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 PER WORKER

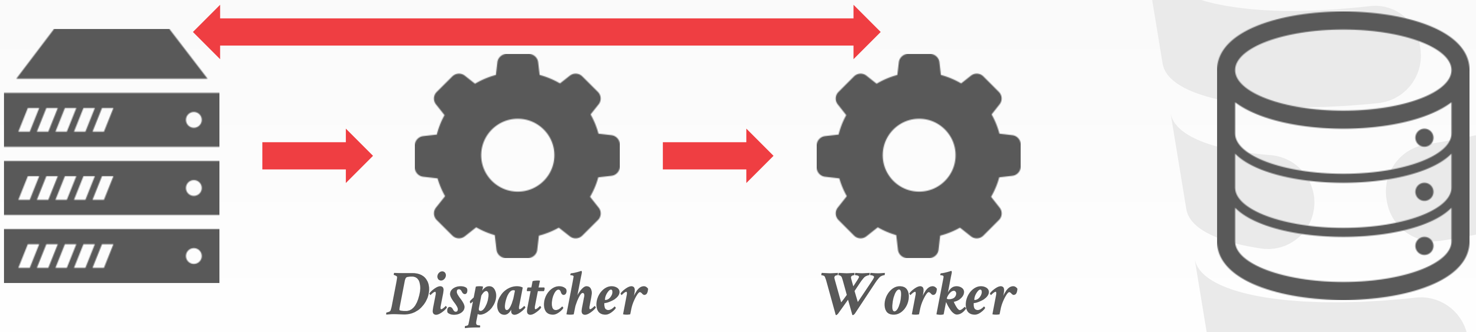
Each worker is a separate OS process.

→ 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 PER WORKER

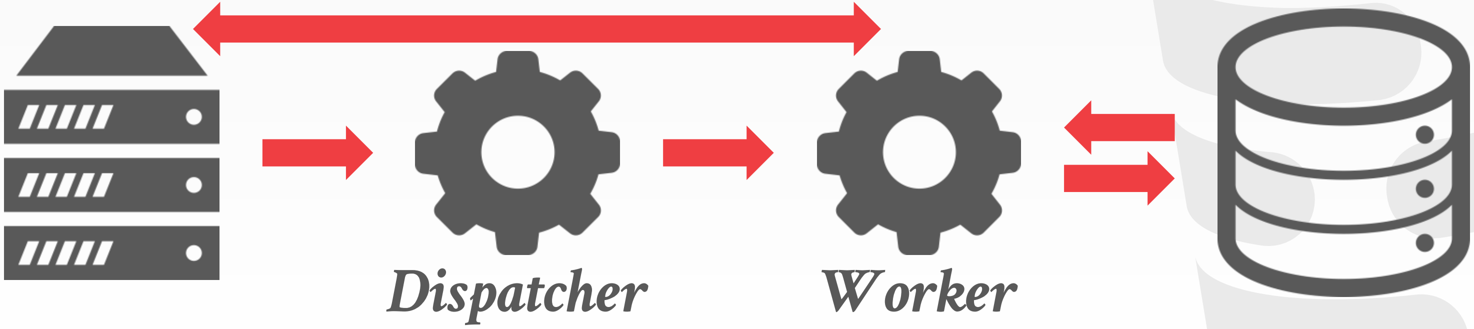
Each worker is a separate OS process.

→ 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



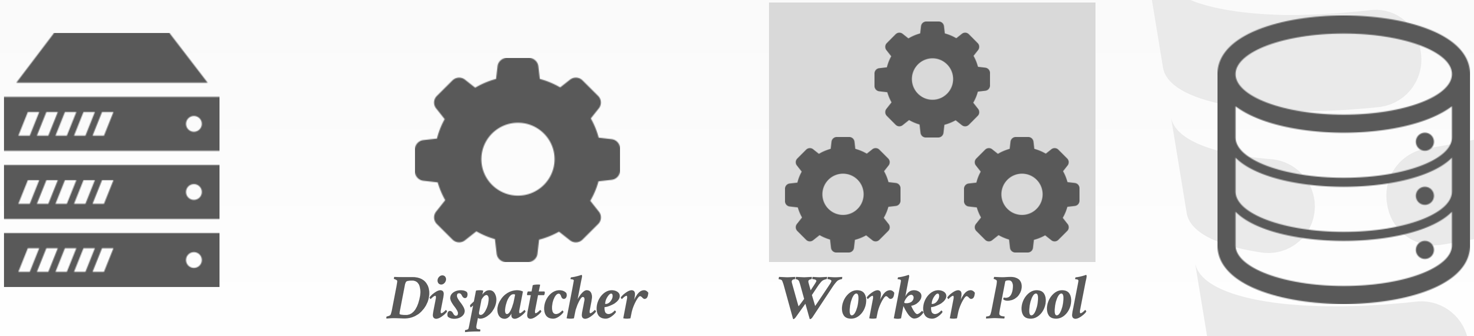
PROCESS POOL

A worker uses any free process from the pool.

→ Still relies on OS scheduler and shared memory.

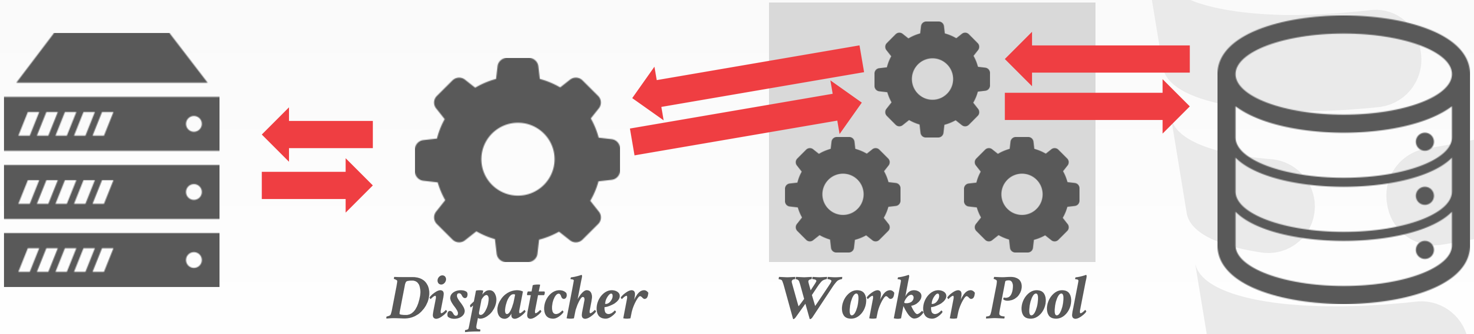
→ Bad for CPU cache locality.

→ Examples: IBM DB2, Postgres (2015)



PROCESS POOL

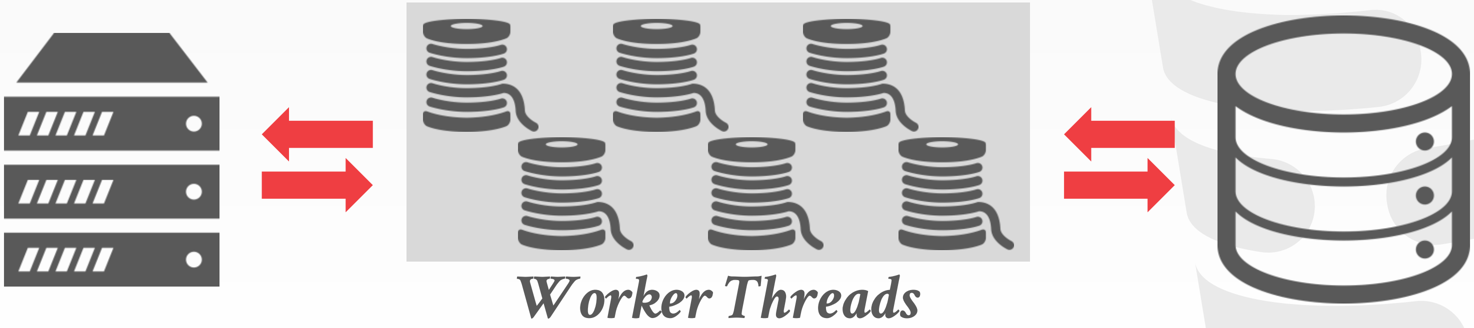
A worker uses any free process from the pool.
→ Still relies on OS scheduler and shared memory.
→ Bad for CPU cache locality.
→ Examples: IBM DB2, Postgres (2015)



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)



PROCESS MODELS

Advantages of a multi-threaded architecture:

→ 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.

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.

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

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.

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

Lecture 15

INTRA-QUERY PARALLELISM

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

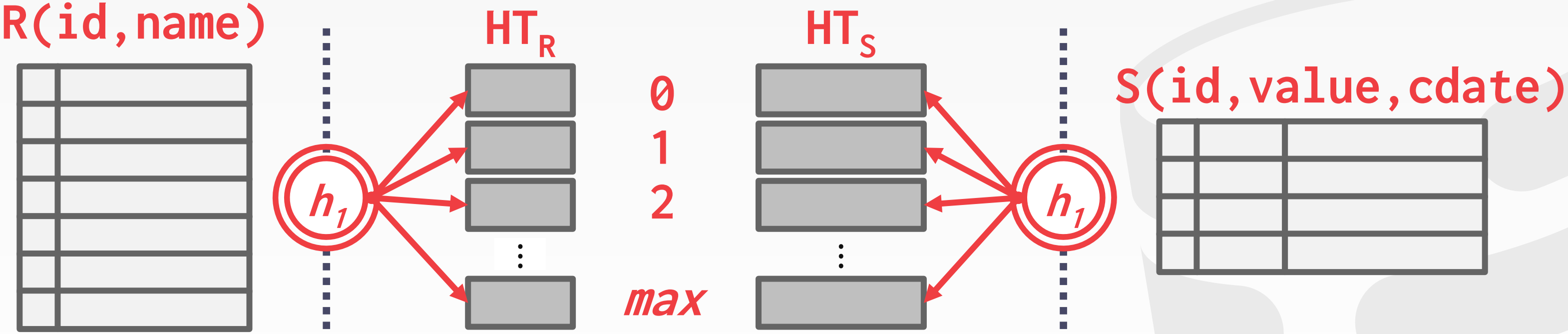
Think of organization of operators in terms of a ***producer/consumer*** paradigm.

There are parallel versions of every operator.

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

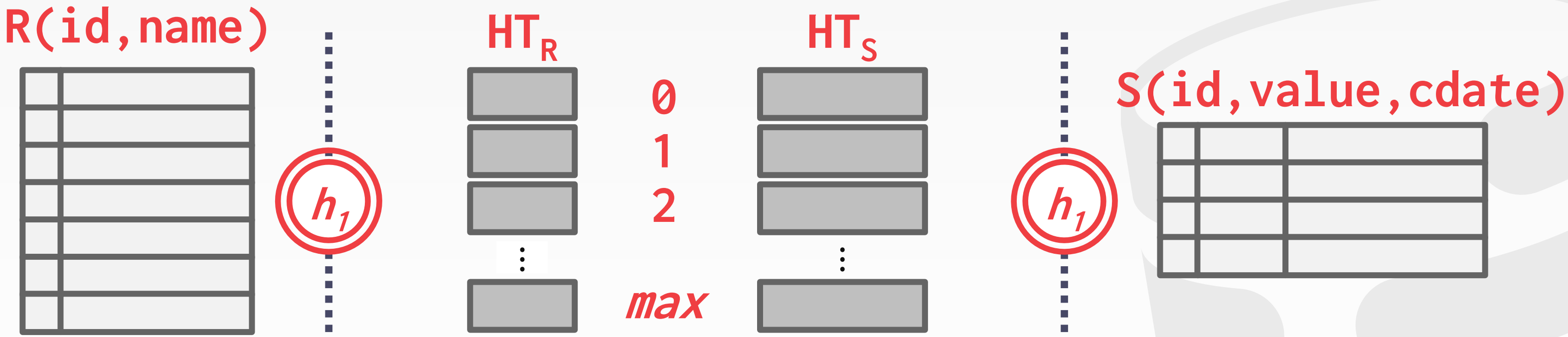
PARALLEL GRACE HASH JOIN

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



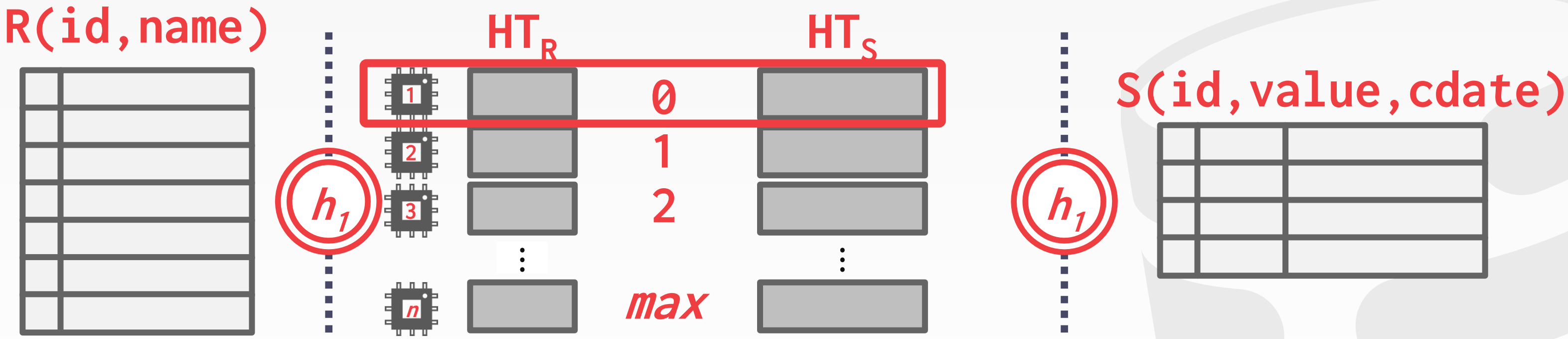
PARALLEL GRACE HASH JOIN

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



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

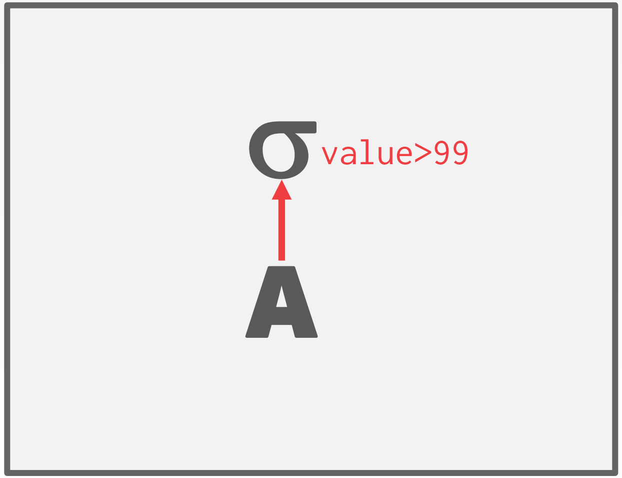
Approach #1: Intra-Operator (Horizontal)

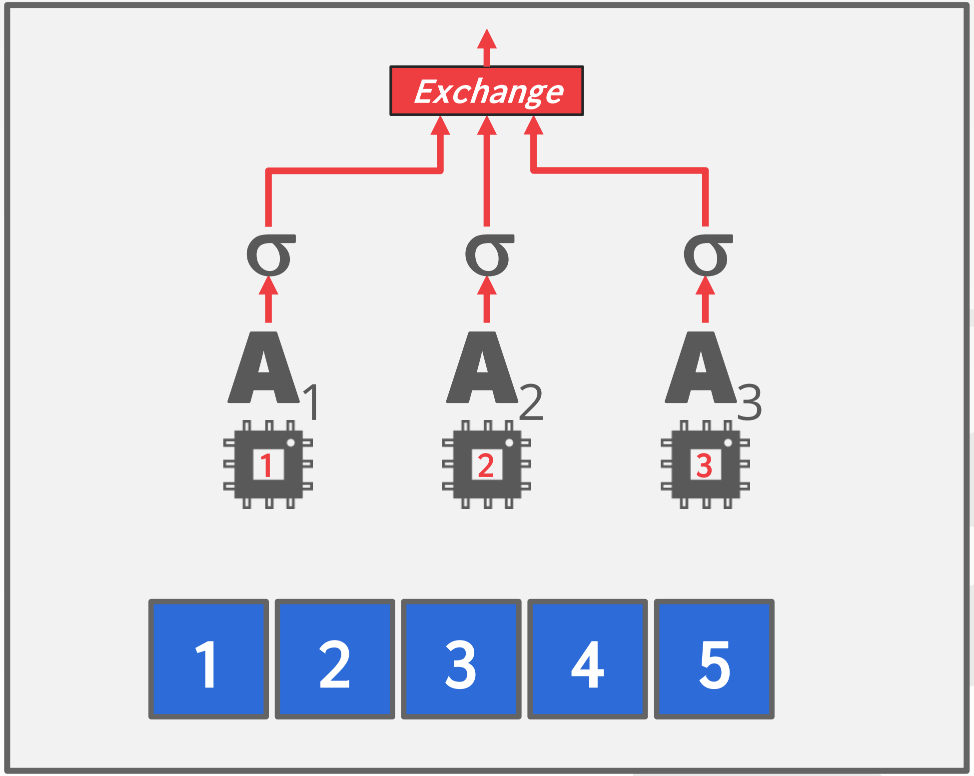
→ Decompose operators into independent **fragments** that perform the same function on different subsets of data.

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

INTRA-OPERATOR PARALLELISM

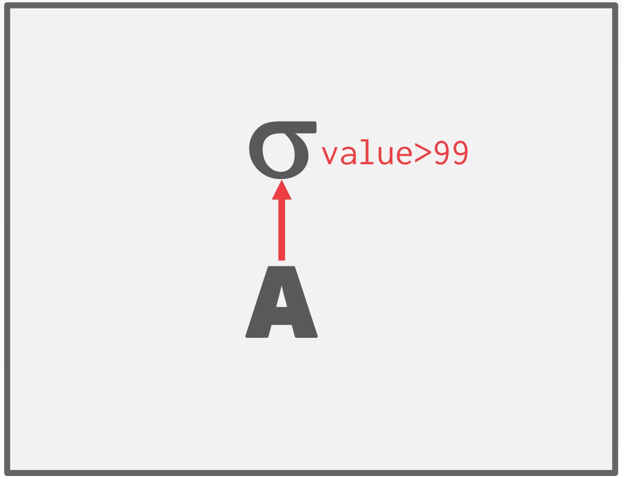
```
SELECT * FROM A
 WHERE A.value > 99
```

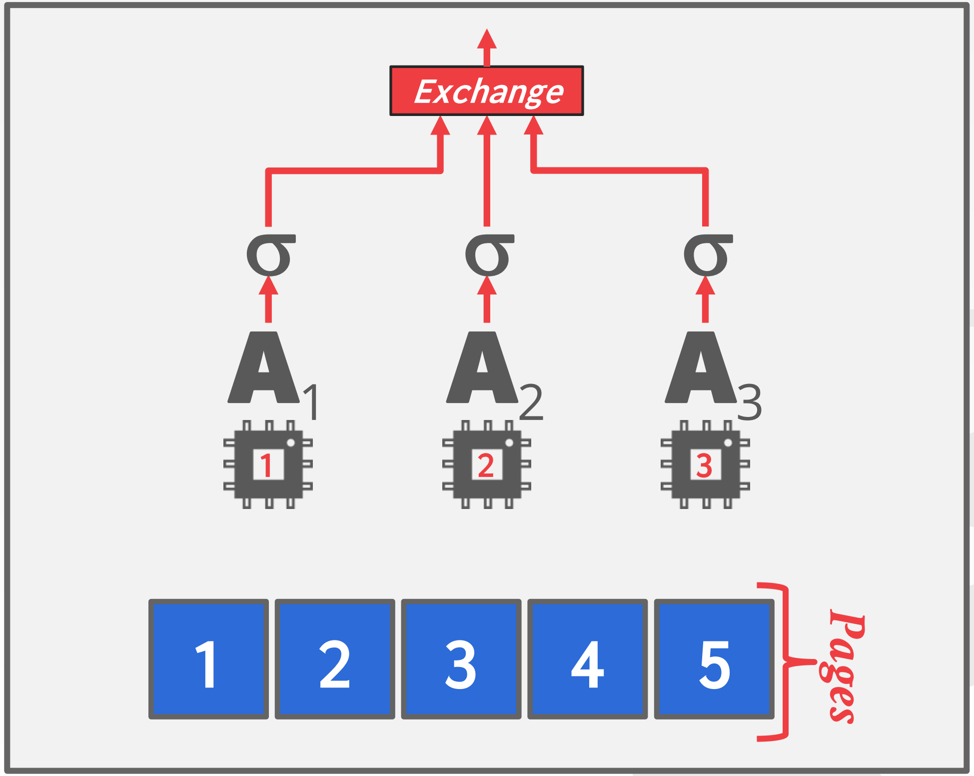




INTRA-OPERATOR PARALLELISM

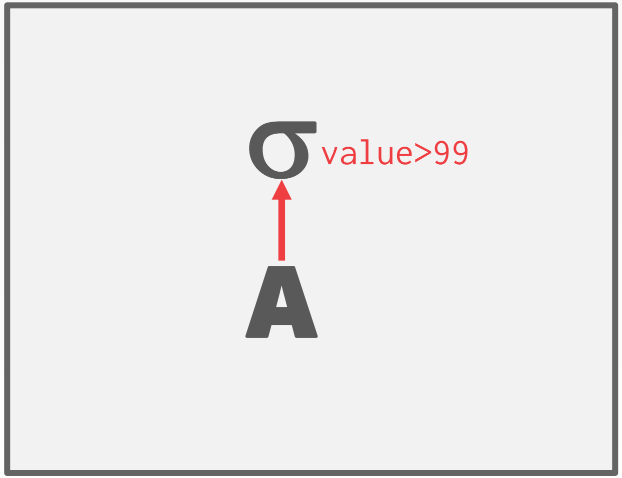
```
SELECT * FROM A
 WHERE A.value > 99
```

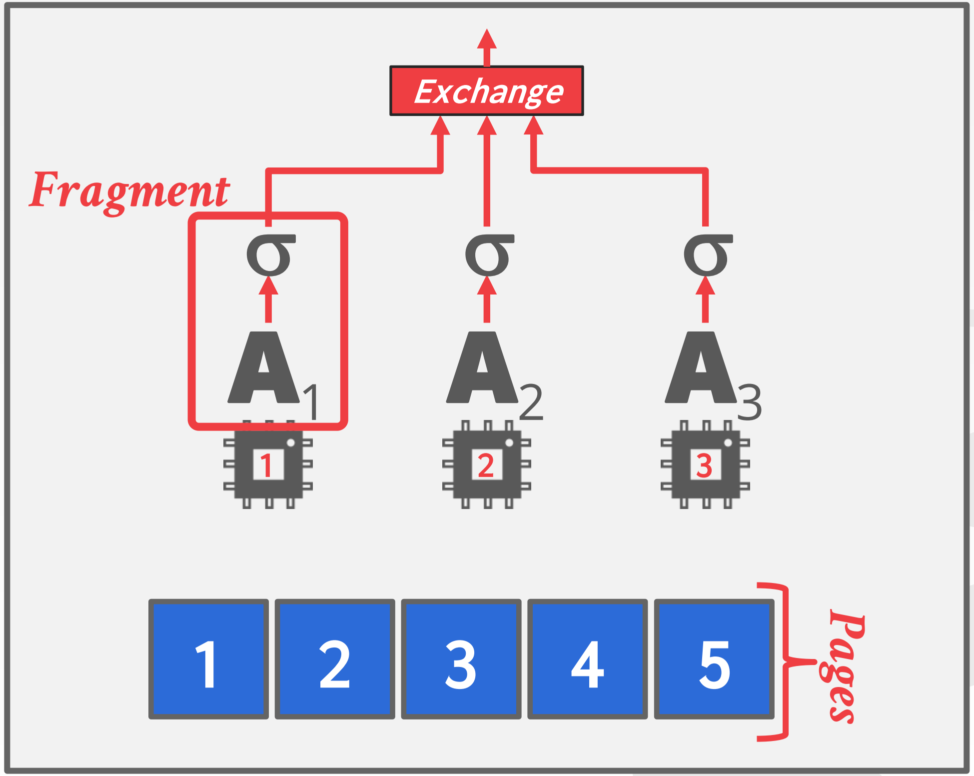




INTRA-OPERATOR PARALLELISM

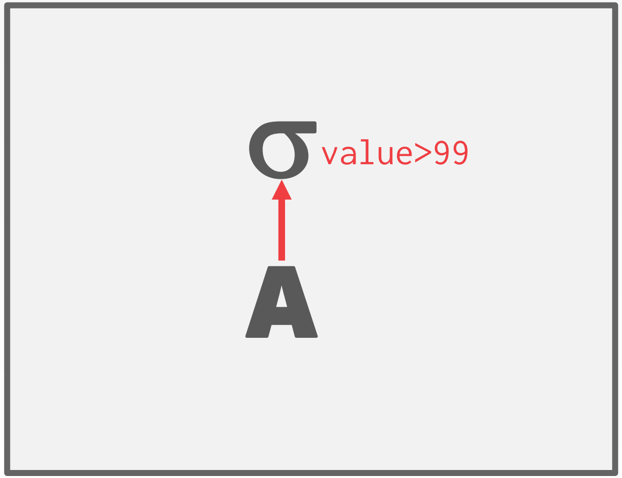
```
SELECT * FROM A
 WHERE A.value > 99
```

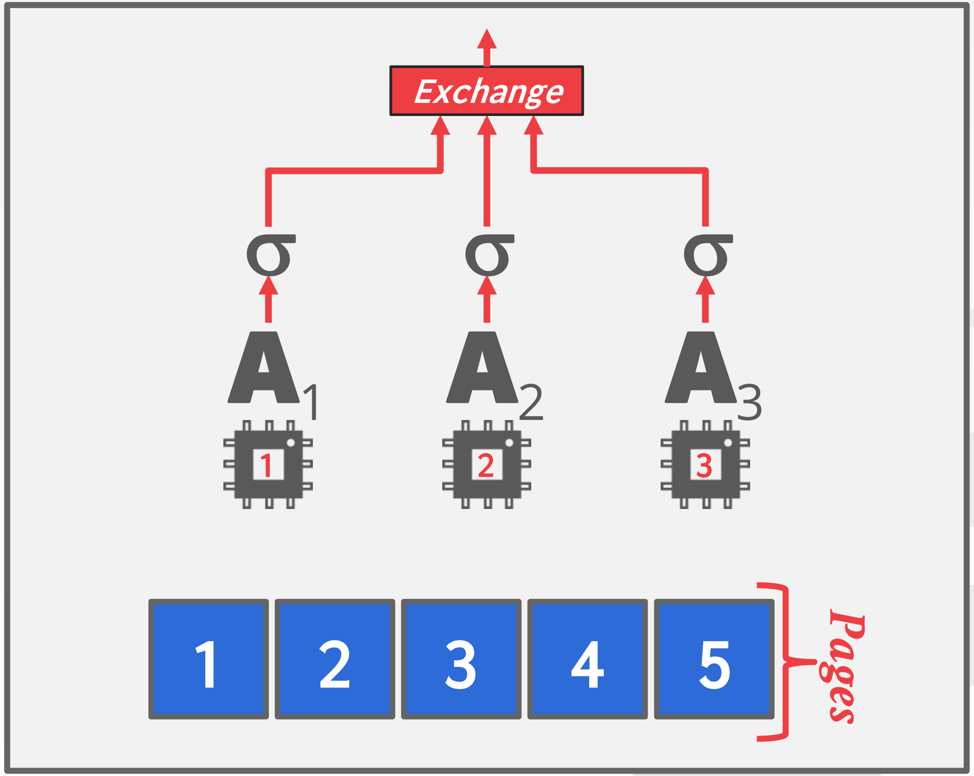




INTRA-OPERATOR PARALLELISM

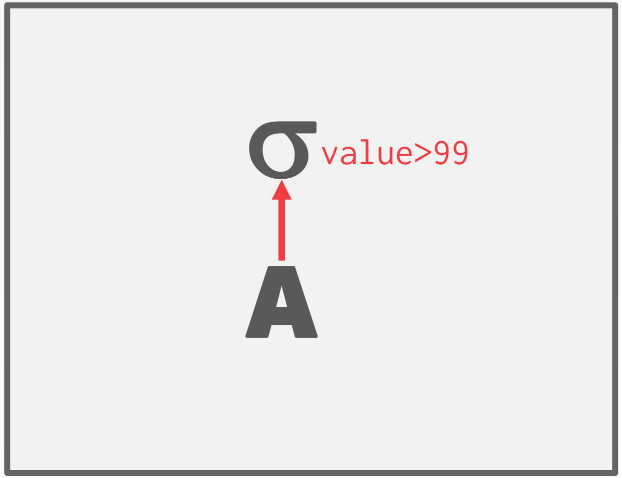
```
SELECT * FROM A
 WHERE A.value > 99
```

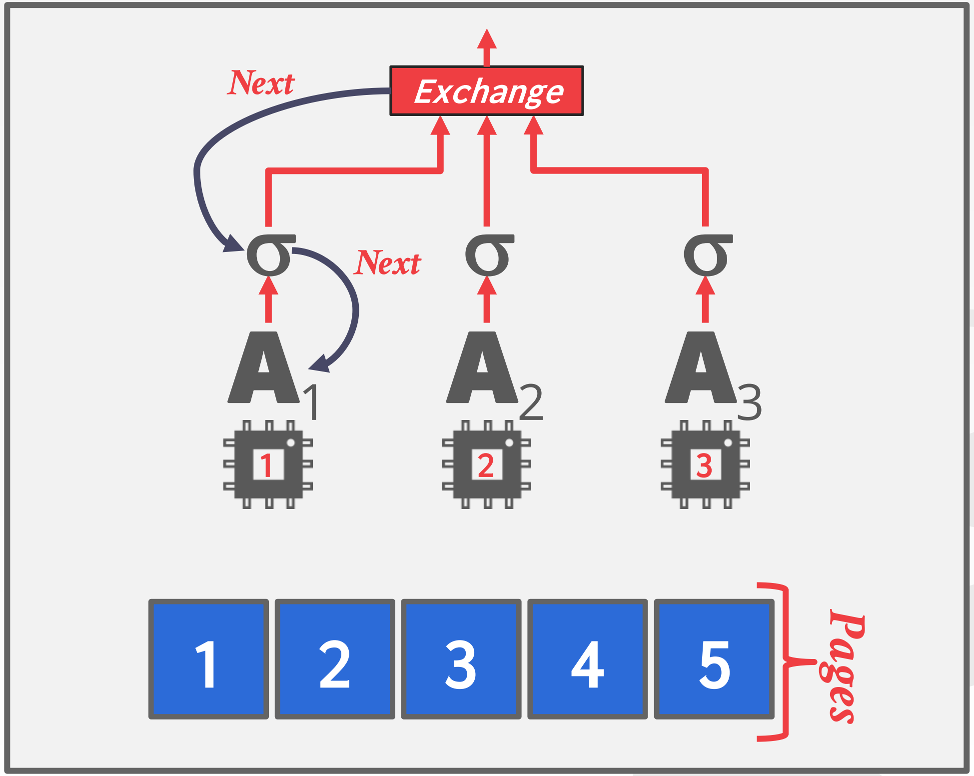




INTRA-OPERATOR PARALLELISM

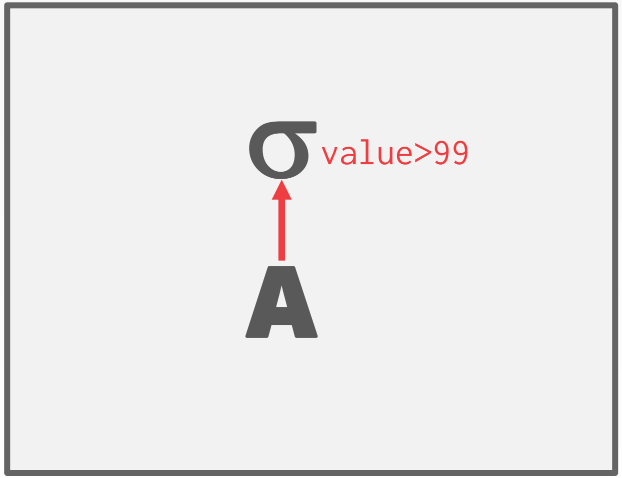
```
SELECT * FROM A
 WHERE A.value > 99
```

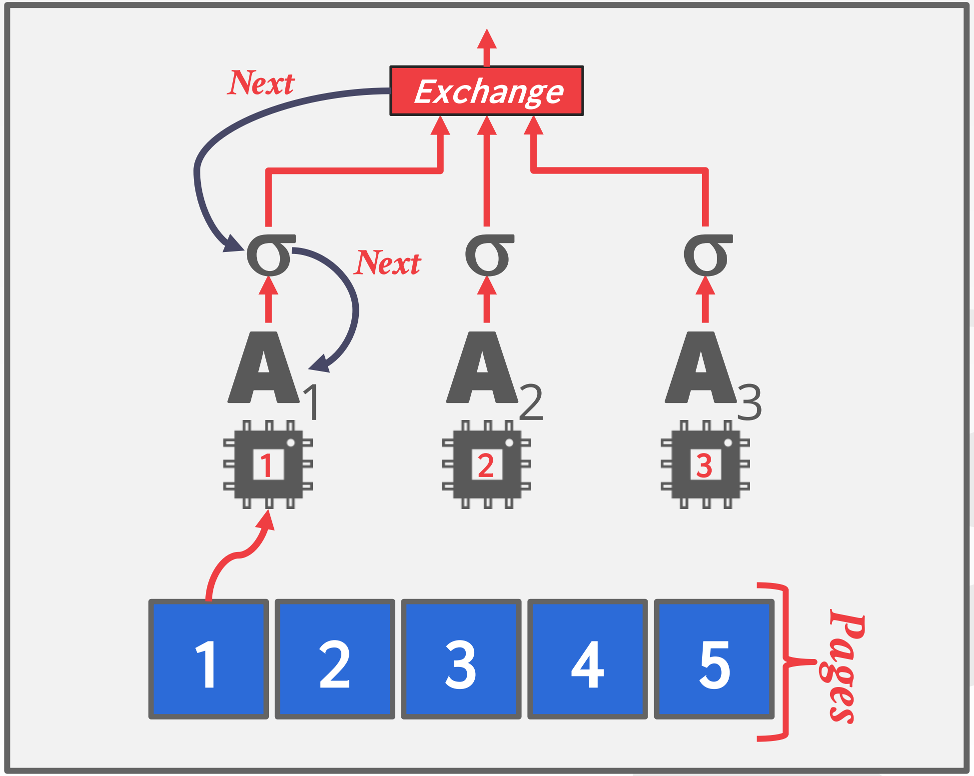




INTRA-OPERATOR PARALLELISM

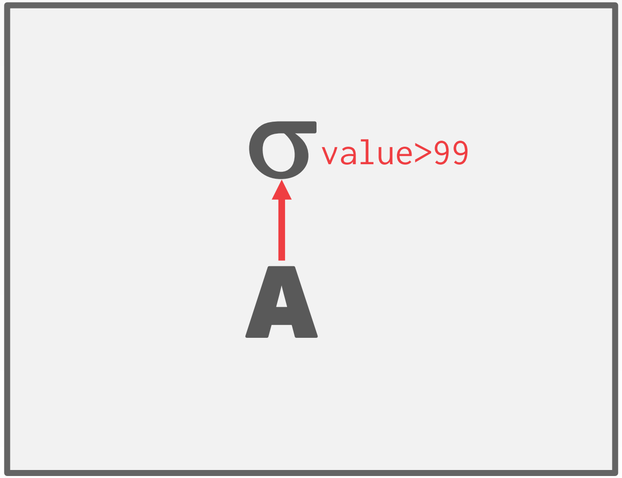
```
SELECT * FROM A
 WHERE A.value > 99
```

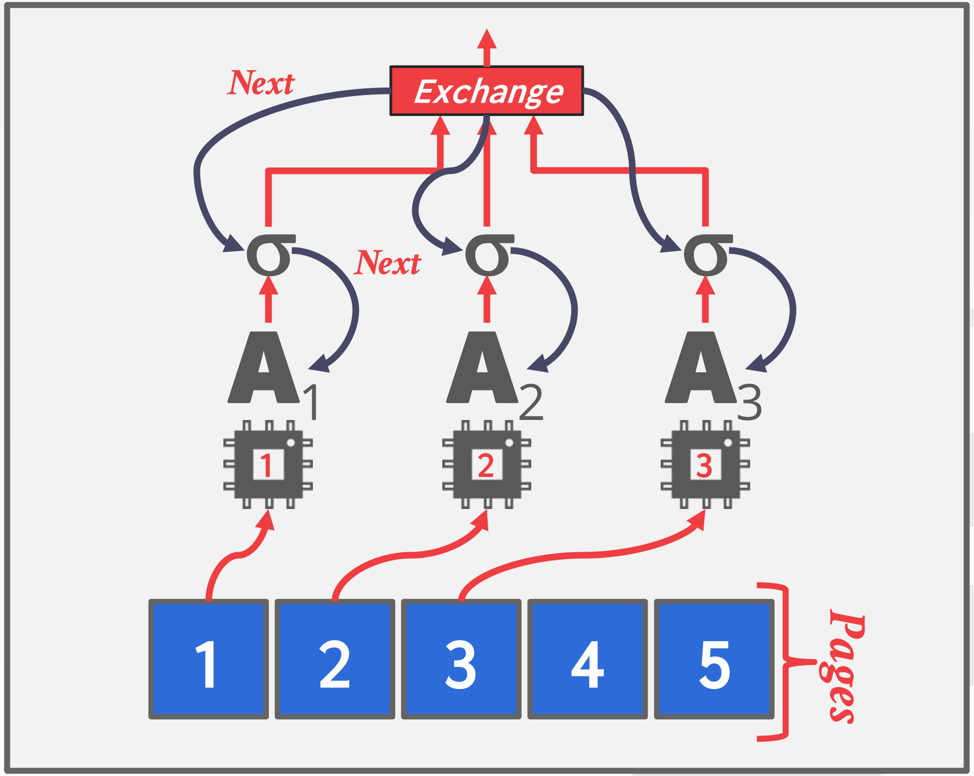




INTRA-OPERATOR PARALLELISM

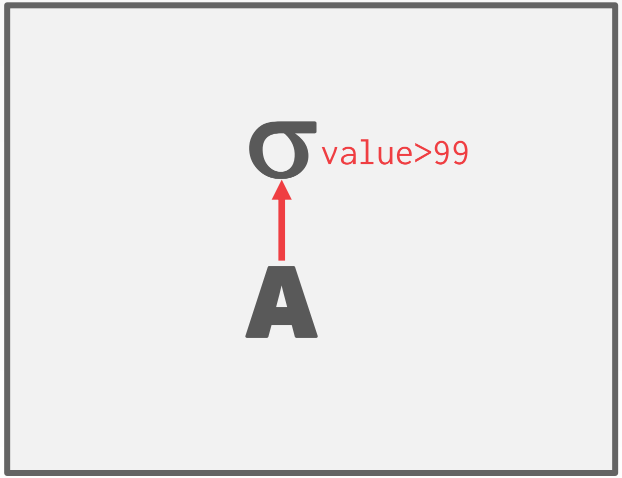
```
SELECT * FROM A
 WHERE A.value > 99
```

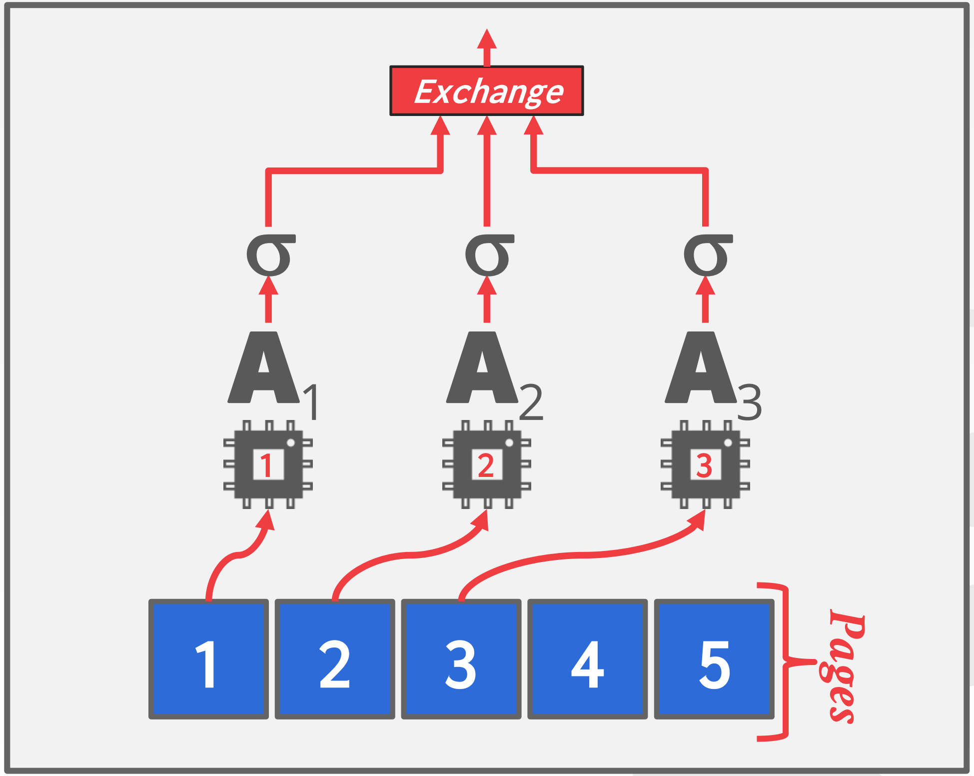




INTRA-OPERATOR PARALLELISM

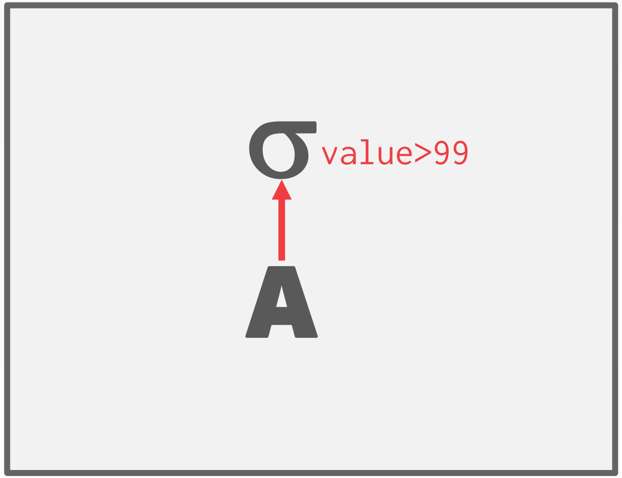
```
SELECT * FROM A
 WHERE A.value > 99
```

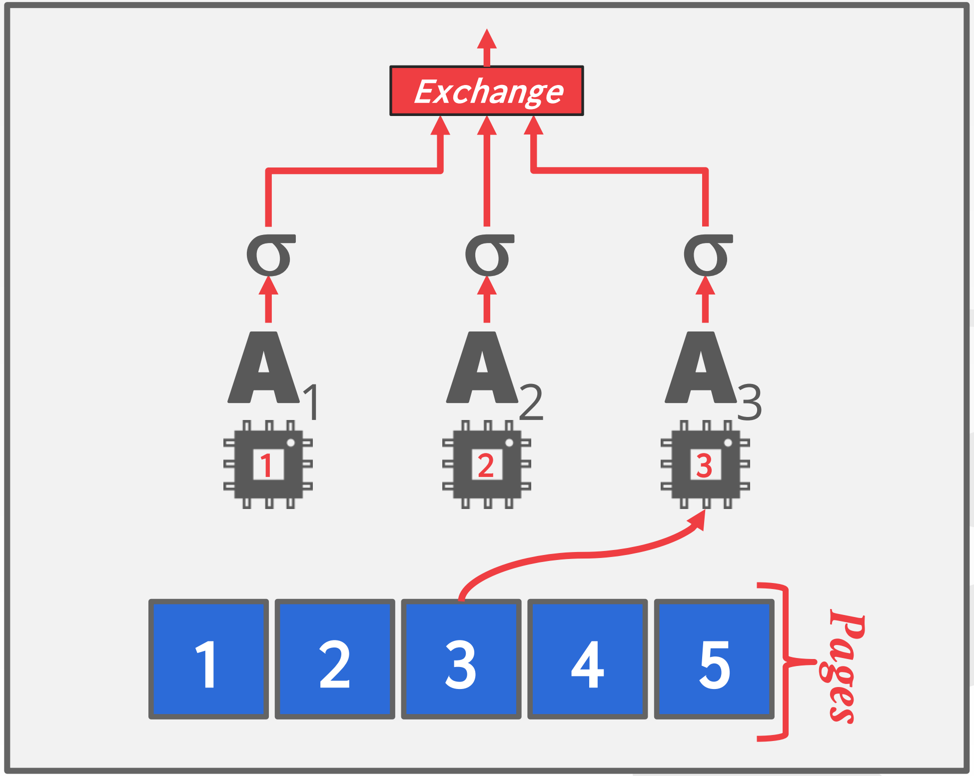




INTRA-OPERATOR PARALLELISM

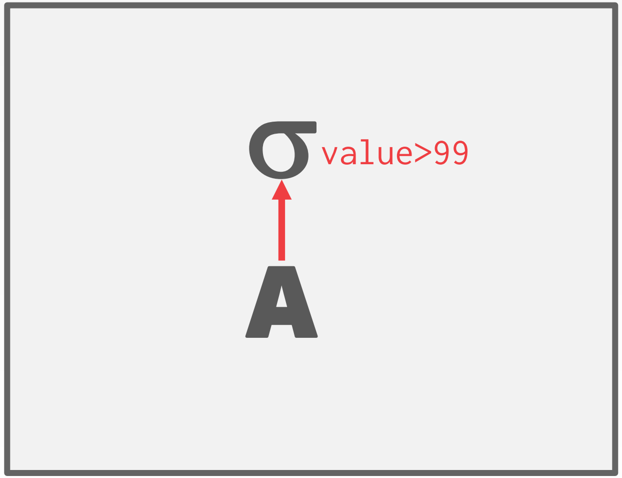
```
SELECT * FROM A
 WHERE A.value > 99
```

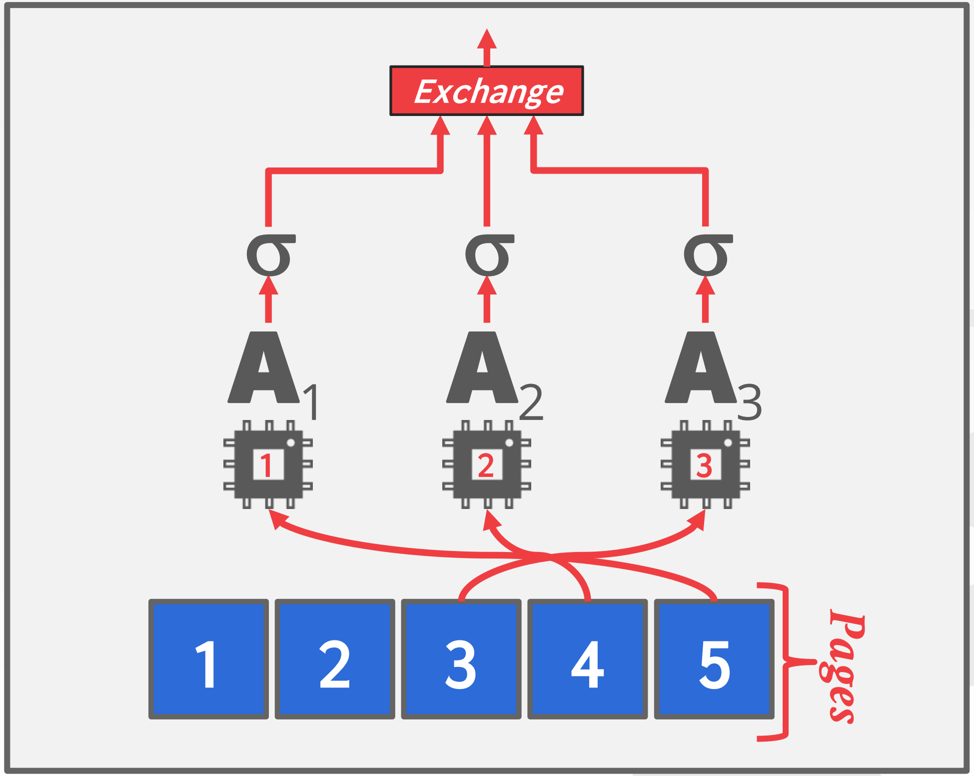




INTRA-OPERATOR PARALLELISM

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





EXCHANGE OPERATOR

Exchange Type #1 – Gather

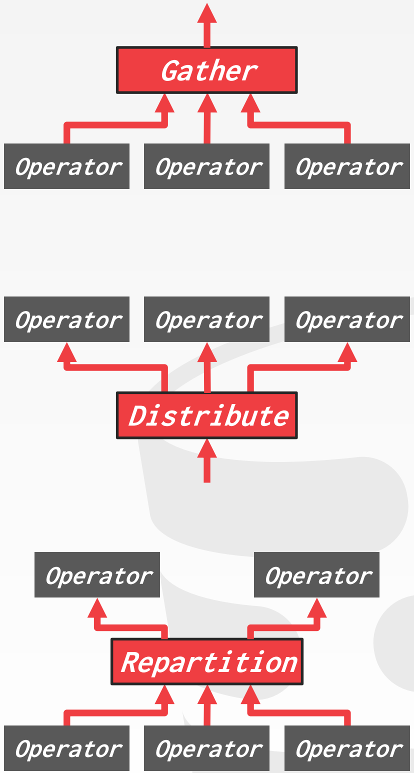
→ Combine the results from multiple workers into a single output stream.

Exchange Type #2 – Distribute

→ Split a single input stream into multiple output streams.

Exchange Type #3 – Repartition

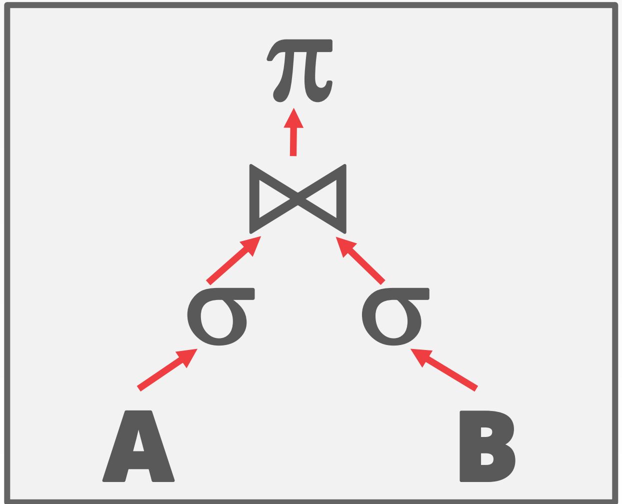
→ Shuffle multiple input streams across multiple output streams.



Source: Craig Freedman

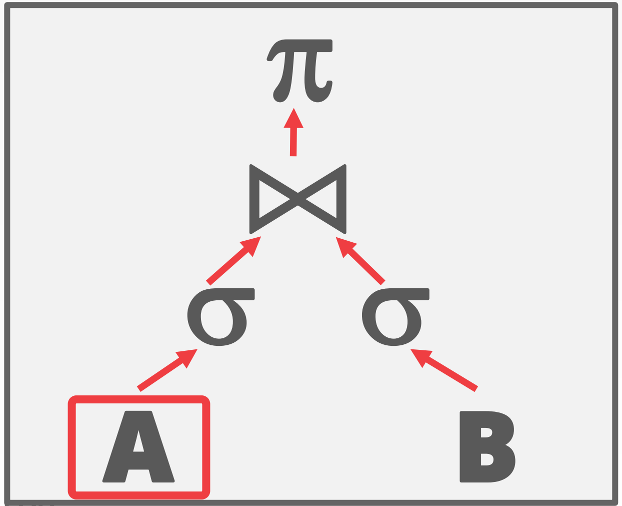
INTRA-OPERATOR PARALLELISM

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



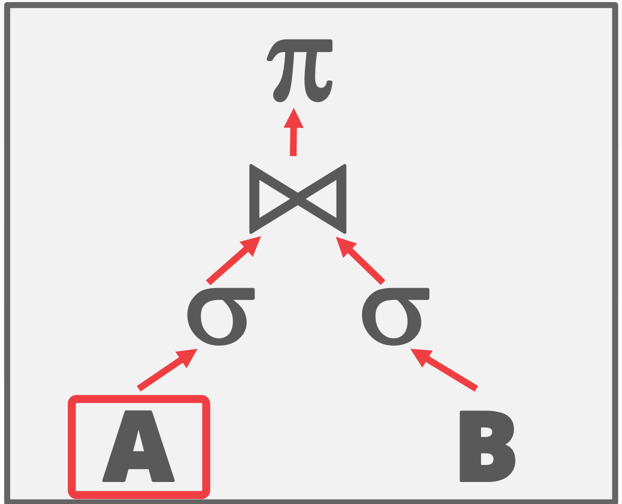
INTRA-OPERATOR PARALLELISM

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



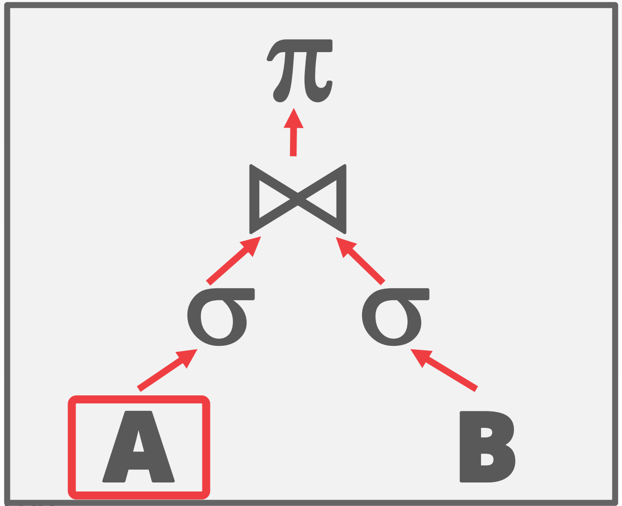
INTRA-OPERATOR PARALLELISM

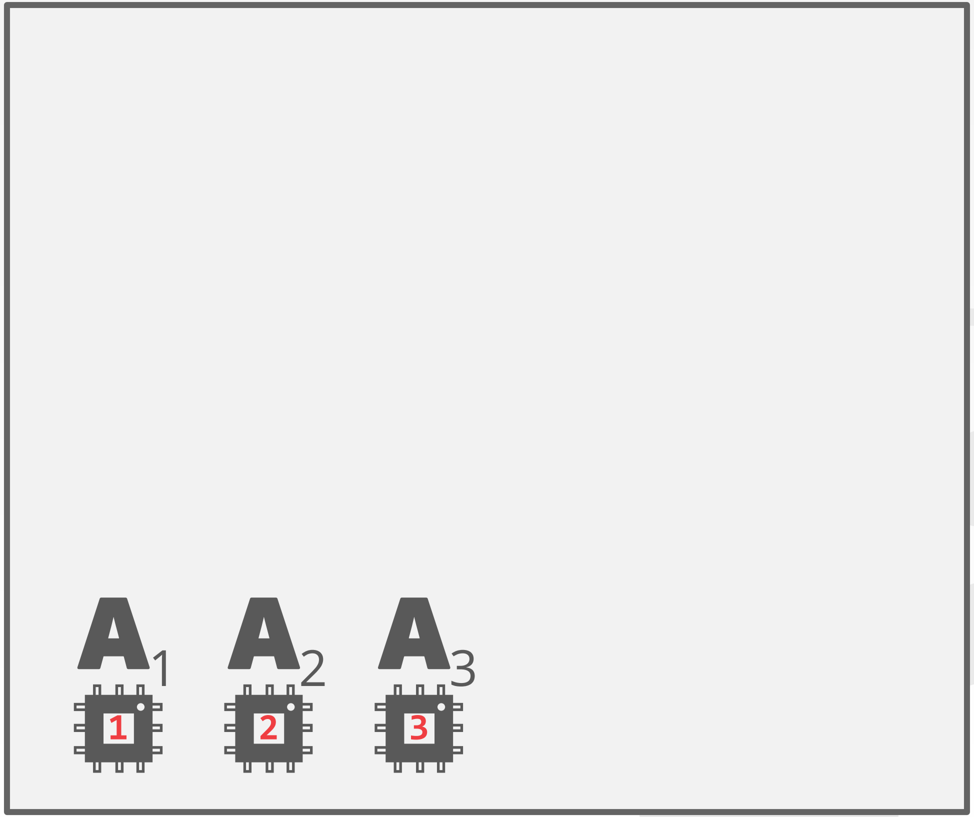
```sql
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```



INTRA-OPERATOR PARALLELISM

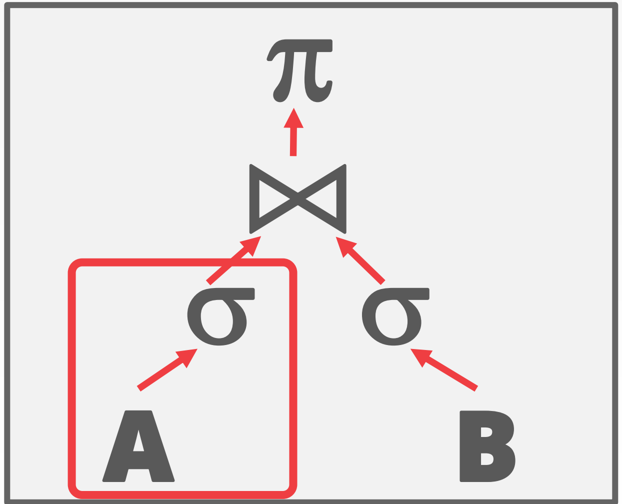
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

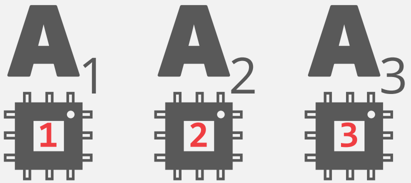




INTRA-OPERATOR PARALLELISM

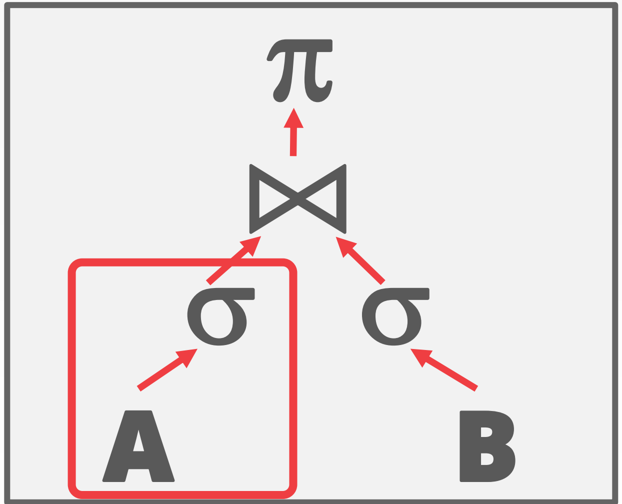
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

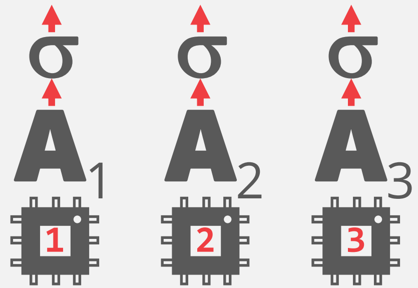




INTRA-OPERATOR PARALLELISM

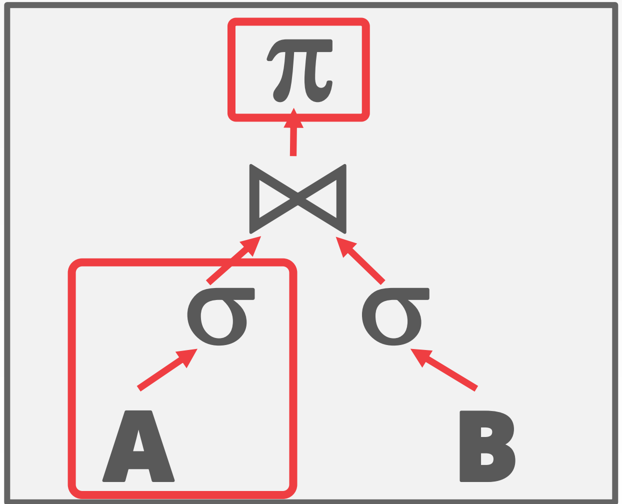
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

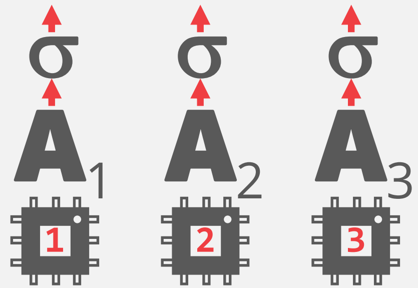




INTRA-OPERATOR PARALLELISM

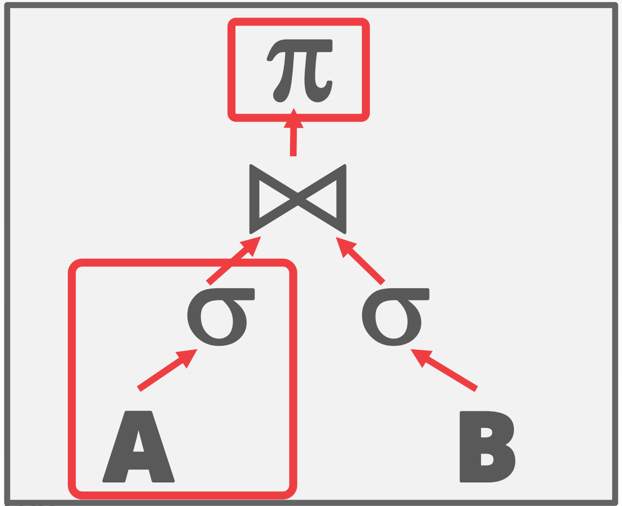
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

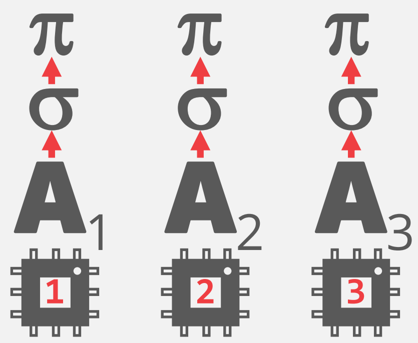




INTRA-OPERATOR PARALLELISM

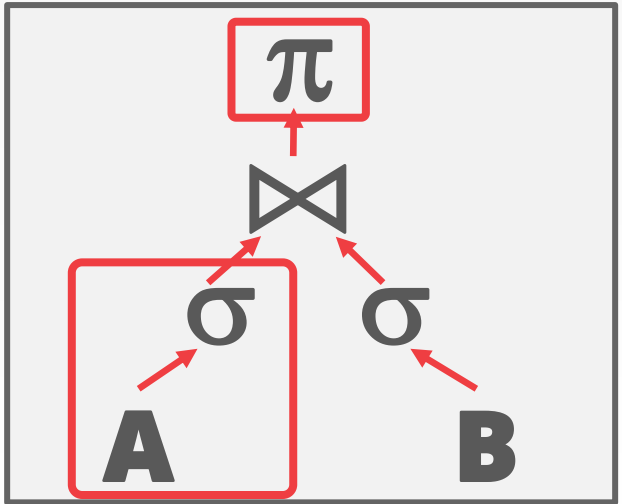
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

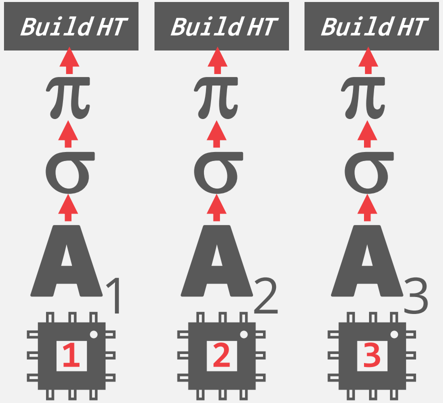




INTRA-OPERATOR PARALLELISM

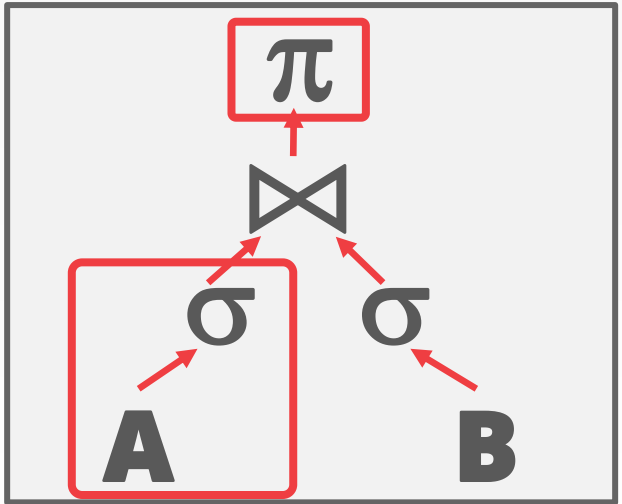
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

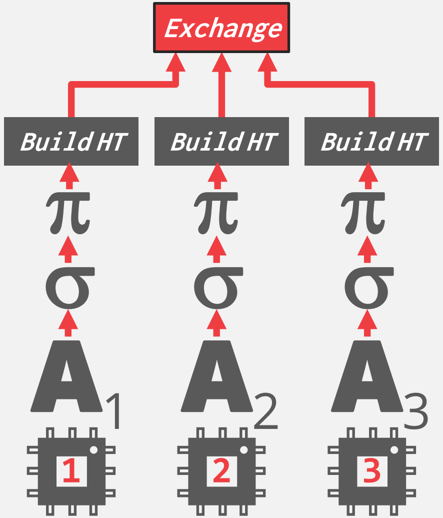




INTRA-OPERATOR PARALLELISM

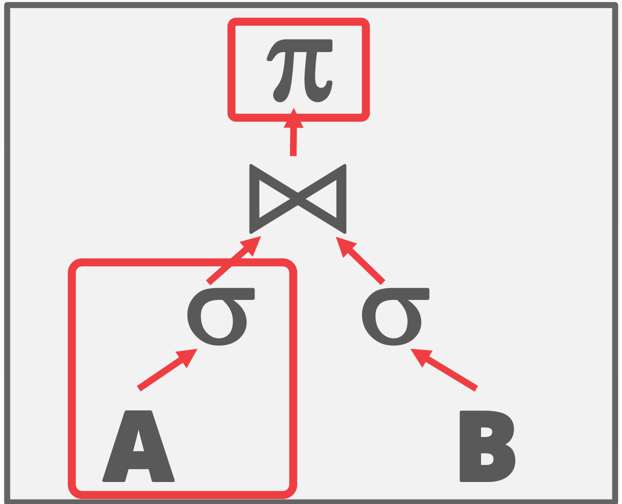
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

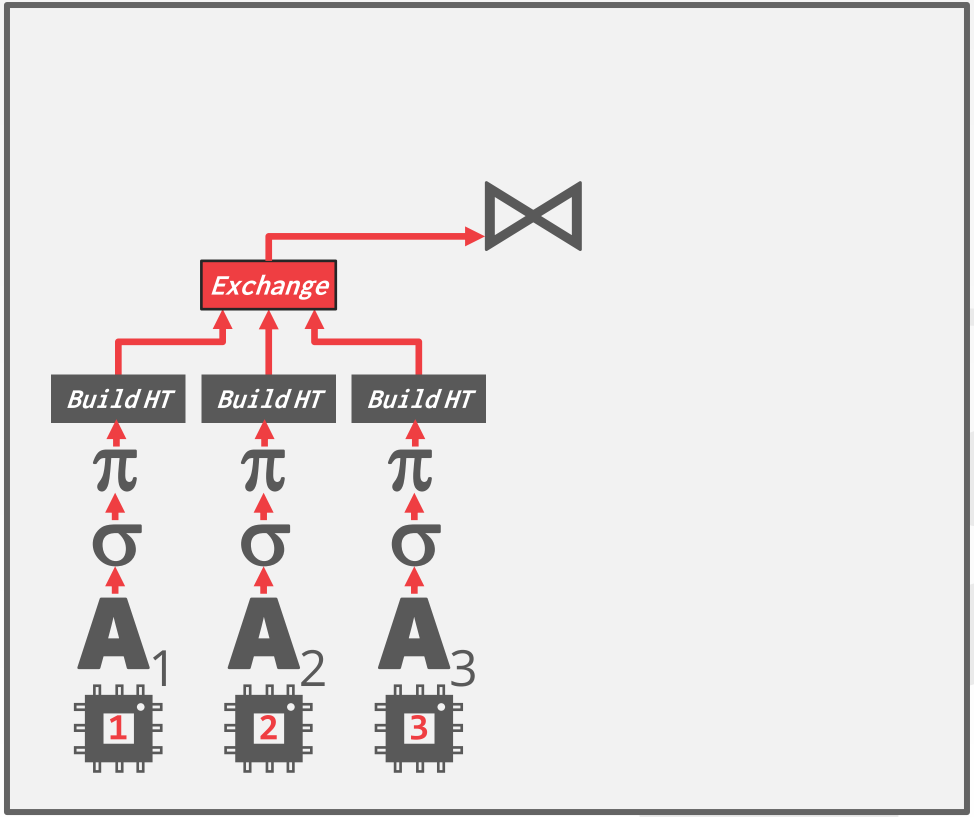




INTRA-OPERATOR PARALLELISM

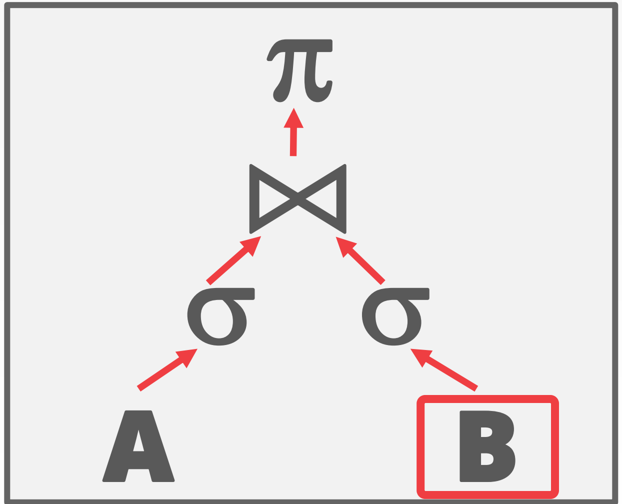
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

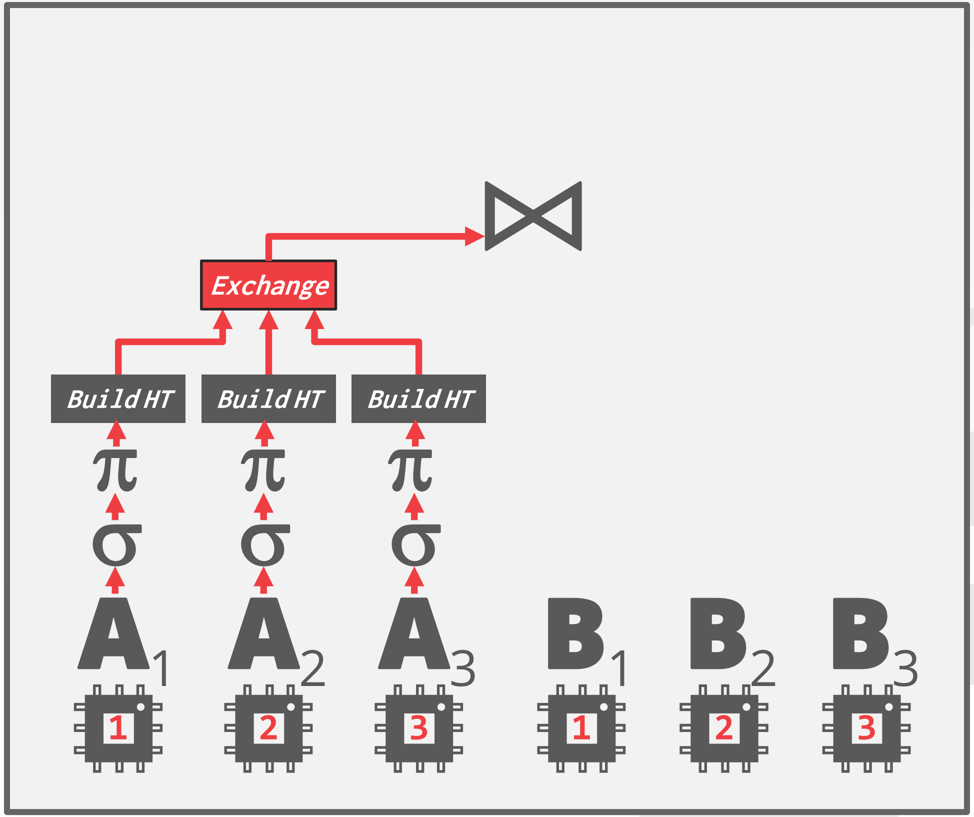




INTRA-OPERATOR PARALLELISM

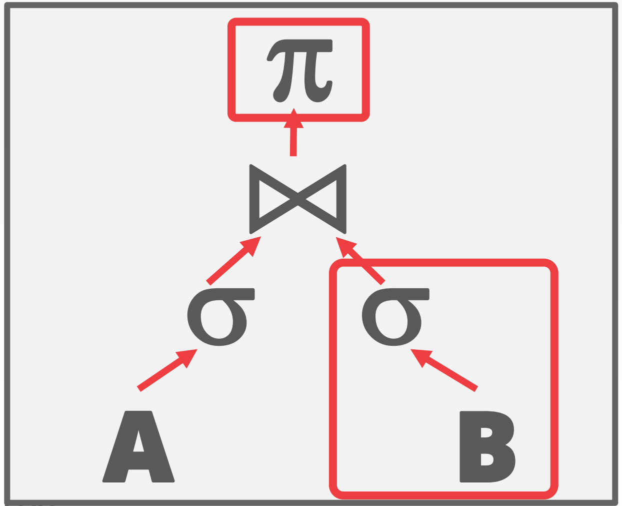
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

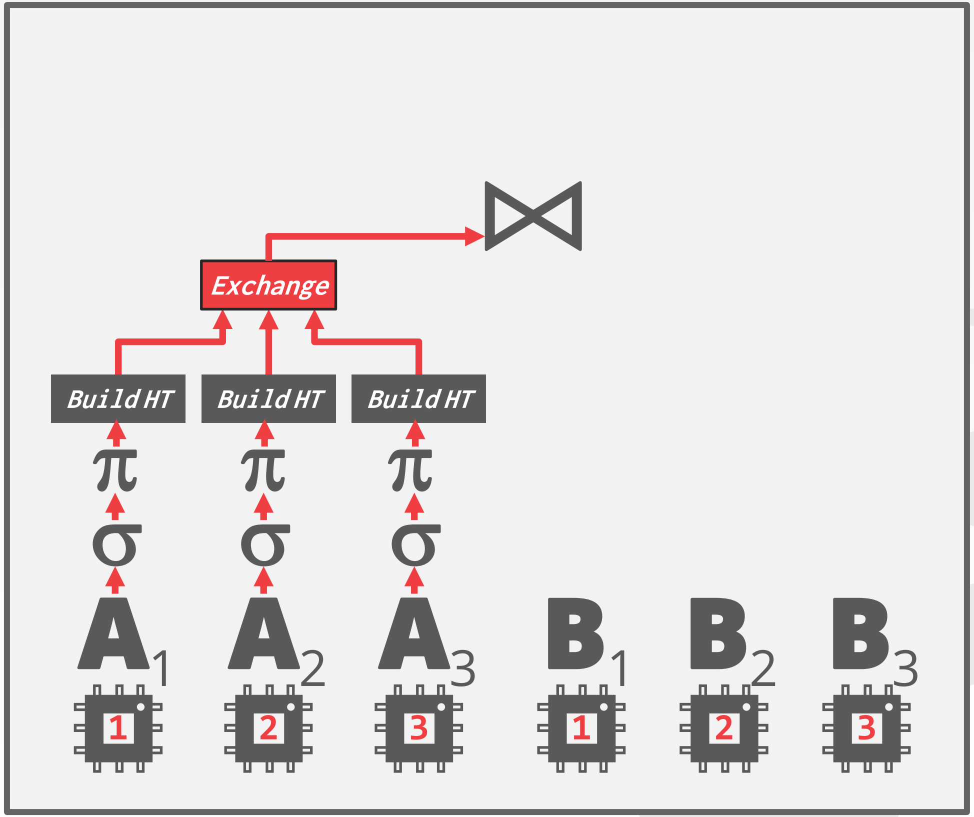




INTRA-OPERATOR PARALLELISM

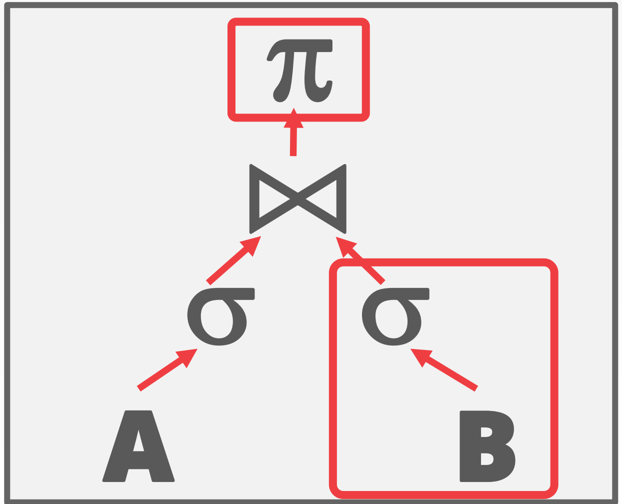
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

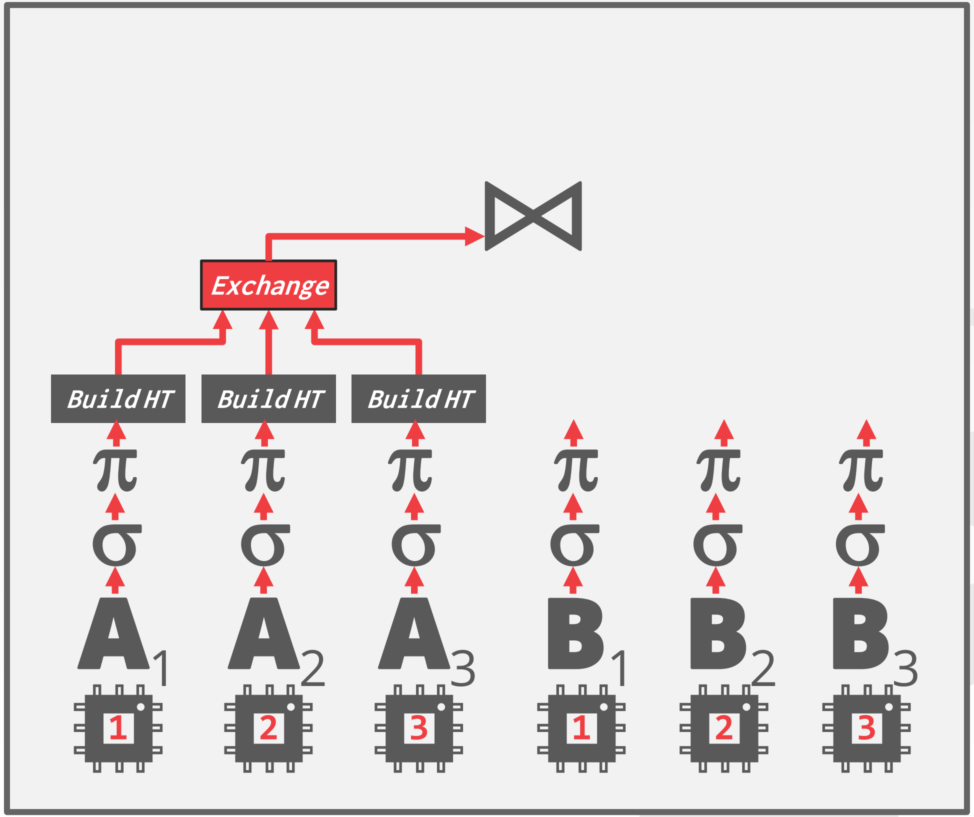




INTRA-OPERATOR PARALLELISM

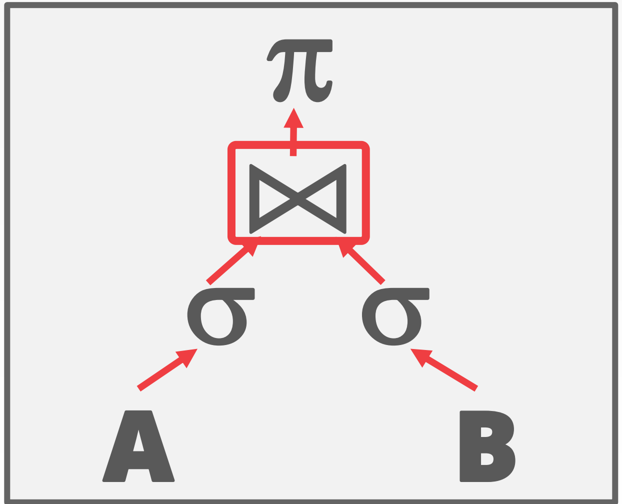
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

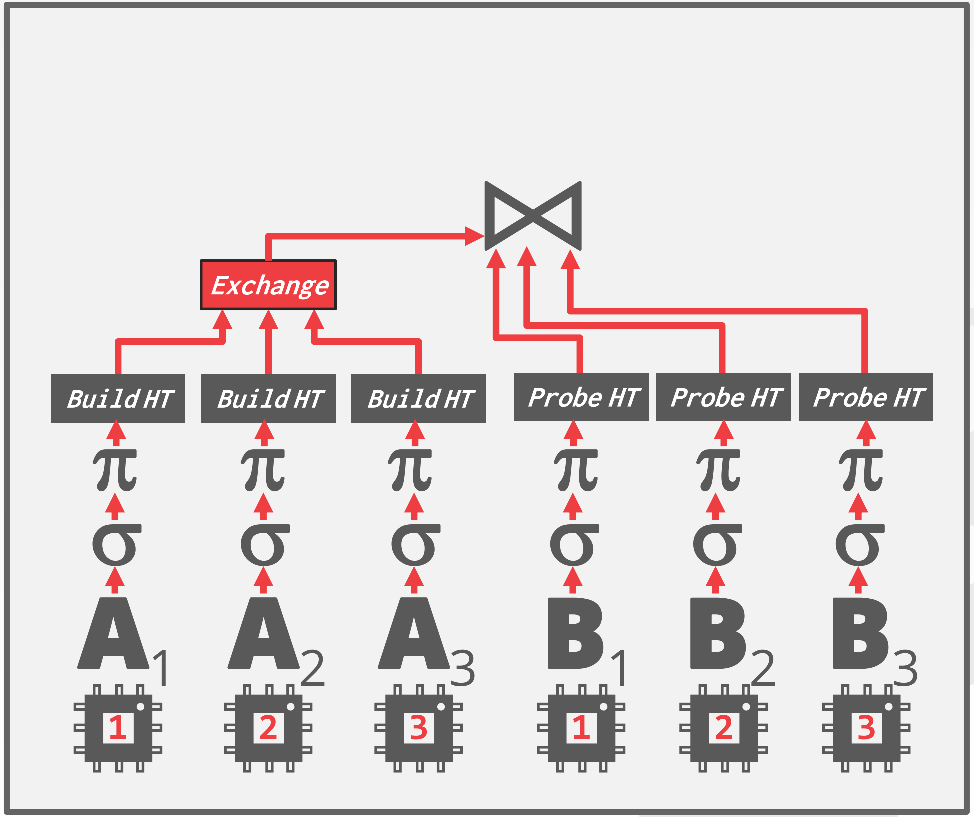




INTRA-OPERATOR PARALLELISM

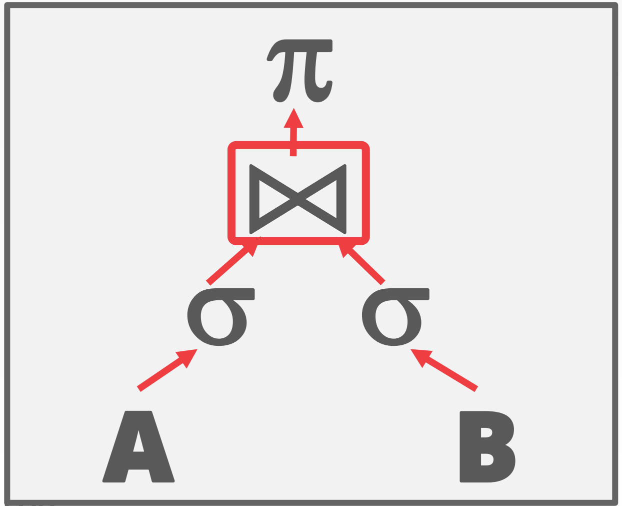
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```

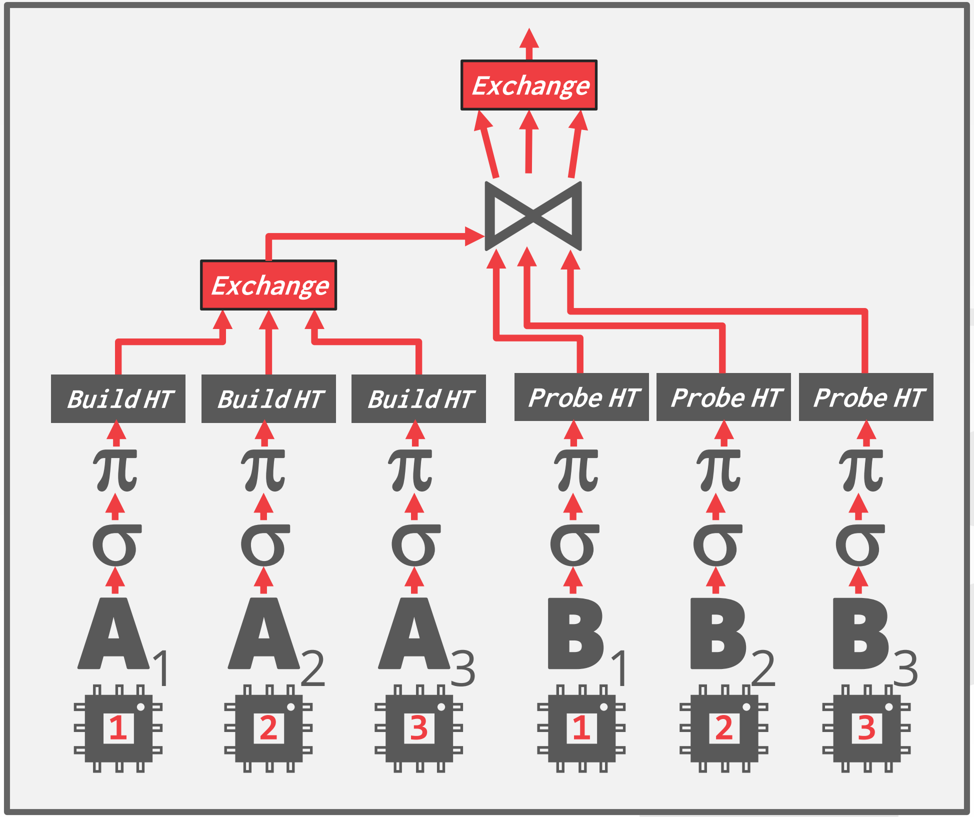




INTRA-OPERATOR PARALLELISM

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





INTER-OPERATOR PARALLELISM

Approach #2: Inter-Operator (Vertical)

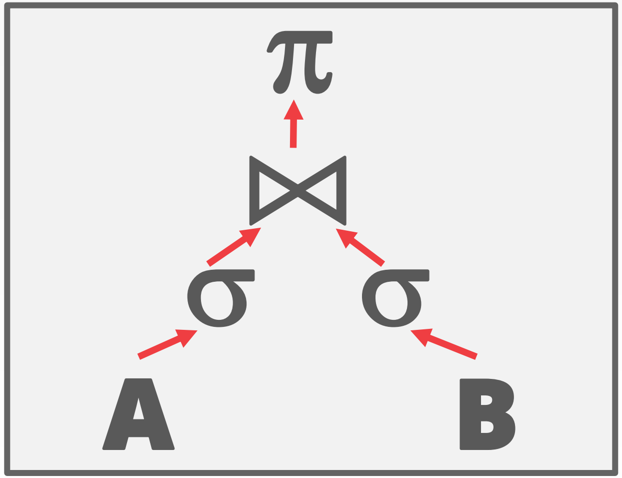
→ Operations are overlapped in order to pipeline data from one stage to the next without materialization.

→ Workers execute operators from different segments of a query plan at the same time.

Also called **pipeline parallelism**.

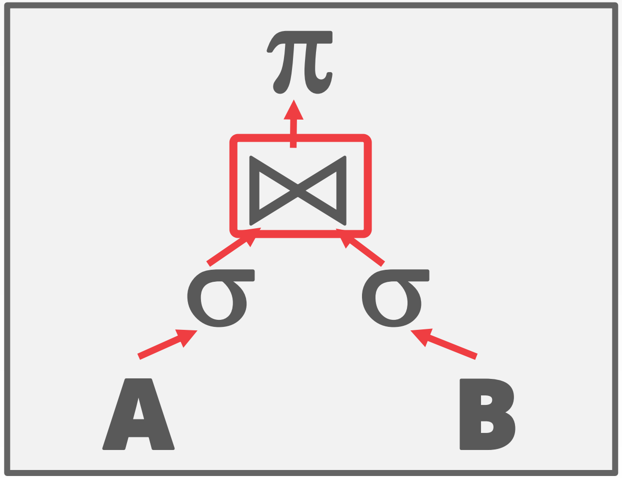
INTER-OPERATOR PARALLELISM

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



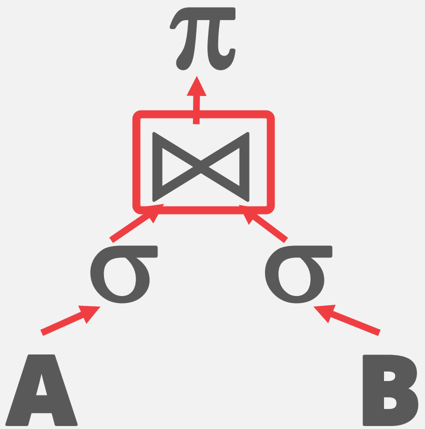
INTER-OPERATOR PARALLELISM

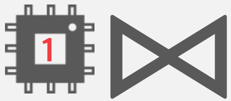
```
SELECT A.id, B.value
  FROM A JOIN B
    ON A.id = B.id
 WHERE A.value < 99
   AND B.value > 100
```



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
```

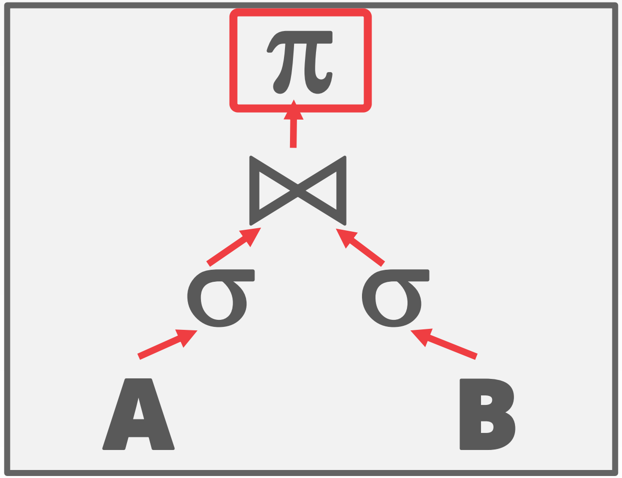


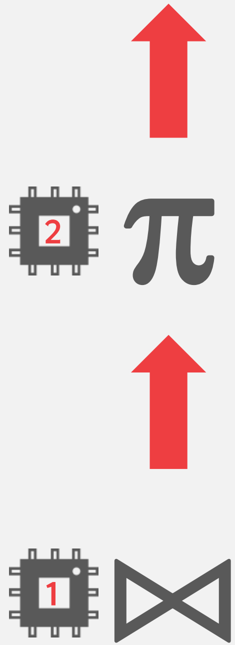


for r_1 ∊ outer:
  for r_2 ∊ inner:
    emit(r_1⋈r_2)

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
```



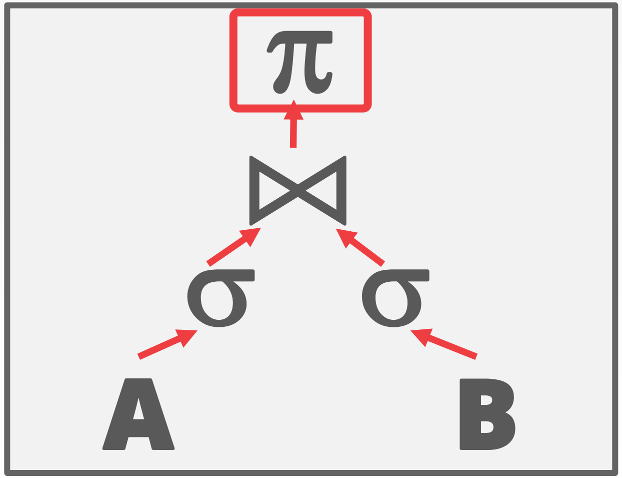


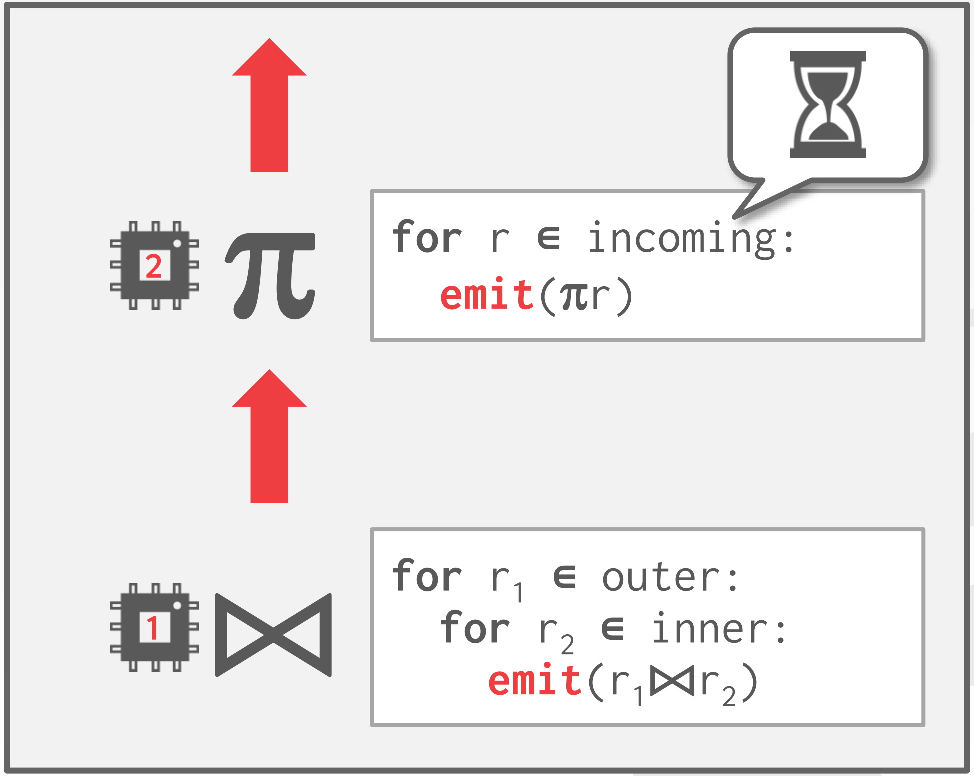

```
for r ∈ incoming:
  emit(πr)
```

```
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
```





BUSHY PARALLELISM

Approach #3: Bushy Parallelism

→ Hybrid of intra- and 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.

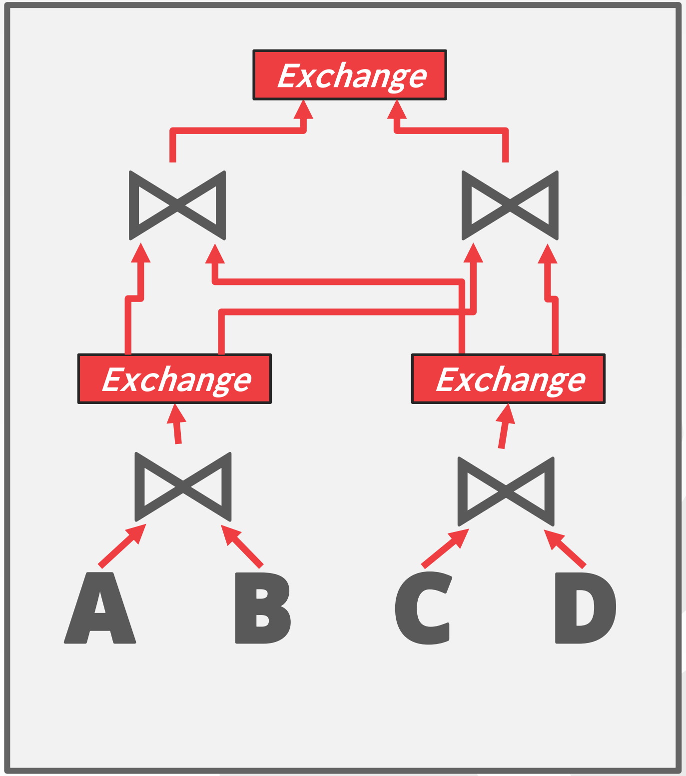
BUSHY PARALLELISM

Approach #3: Bushy Parallelism

→ Hybrid of intra- and 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
```



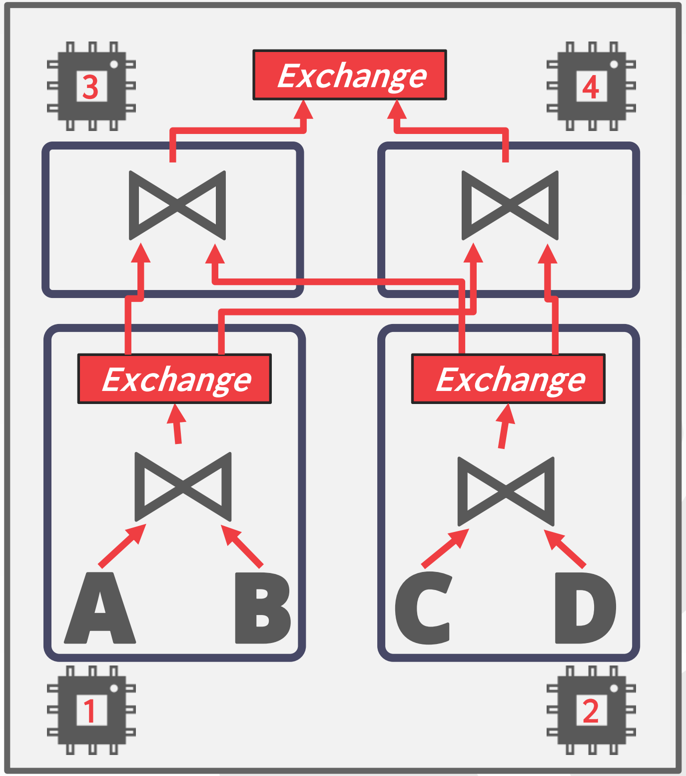
BUSHY PARALLELISM

Approach #3: Bushy Parallelism

→ Hybrid of intra- and 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.

→ In fact, it can make things worse if each worker is working on different segments of the disk.

I/O PARALLELISM

Split the DBMS 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

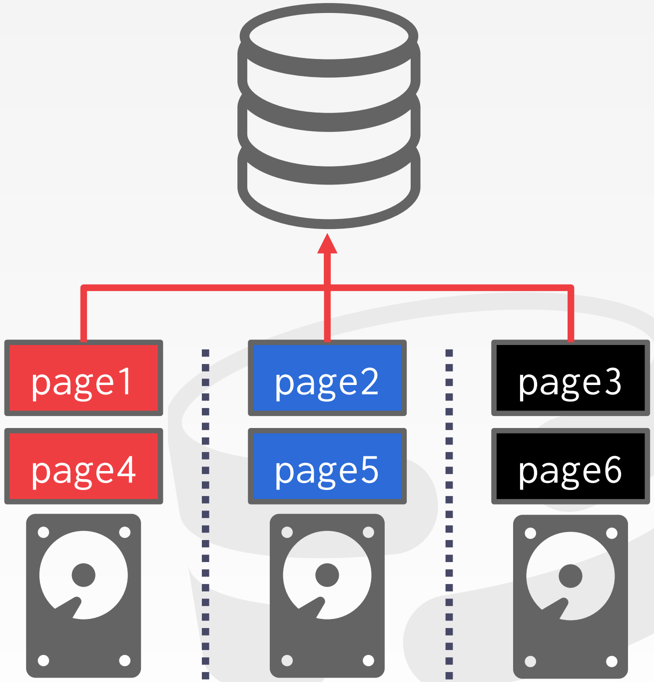
This is **transparent** to the DBMS.

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.



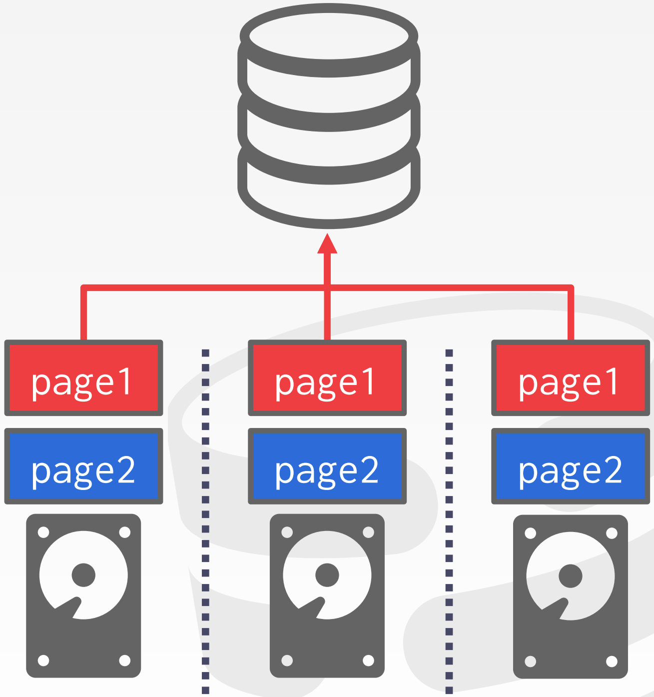
RAID 0 (Striping)

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 to 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 DBMS recovery log file might still be shared if transactions can update multiple databases.

PARTITIONING

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

Partitioning should (ideally) be transparent to the application.

→ The application should only access logical tables and not have to worry about how things are physically stored.

VERTICAL PARTITIONING

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

Must 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).

Must 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).

Must 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 table 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 table 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, which is why (almost) every major DBMS supports it.

However, it is hard to get right.
→ Coordination Overhead
→ Scheduling
→ Concurrency Issues
→ Resource Contention

MIDTERM EXAM

Who: You

What: Midterm Exam

Where: Here (McConomy Auditorium)

When: Wednesday, Oct 13th @ 3:05-4:25pm

Why: https://youtu.be/EDRsQQ6Onnw

https://15445.courses.cs.cmu.edu/fall2021/midterm-guide.html

MIDTERM EXAM

Exam will cover all lecture material up to and including today (**Lecture #12**).

Open book / open notes / calculator

What to bring:

→ CMU ID
→ Calculator
→ Pen or pencil (pencils recommended)