Carnegie Mellon University

Parallel Execution

Lecture #14

Database Systems
15-445/15-645
Fall 2017

Andy Pavlo
Computer Science Dept.
Carnegie Mellon Univ.

ADMINISTRIVIA

Mid-term Exam is on Wednesday October 18th (in class)

Project #2 is due Wednesday October 25th @ 11:59am

MIDTERM EXAM

Who: You

What: Midterm Exam

When: Wed Oct 18th 12:00pm - 1:20pm

Where: Scaife Hall 125

Why: https://youtu.be/xgMiaIPxSlc

http://cmudb.io/f17-midterm

WHY DO WE CARE ABOUT PARALLEL EXECUTION?

Increased performance.
→ Throughput
→ Latency

Increased availability.

Potentially lower TCO.

PARALLEL & DISTRIBUTED DATABASE SYSTEMS

Database is spread out across multiple resources to improve parallelism.

Appears as a single database instance to the application.

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

PARALLEL VS. DISTRIBUTED

Parallel DBMSs:

→ Nodes are physically close to each other.
→ Nodes connected with high-speed LAN.
→ Communication cost is assumed to be small.

Distributed DBMSs:

→ Nodes can be far from each other.
→ Nodes connected using public network.
→ Communication cost and problems cannot be ignored.

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.

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

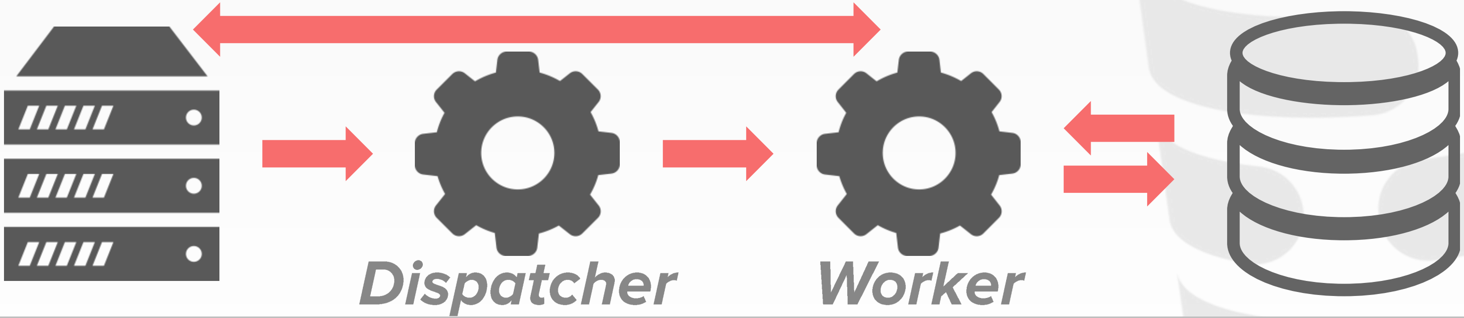
Approach #3: Thread per DBMS Worker

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.

IBM DB2
ORACLE
PostgreSQL

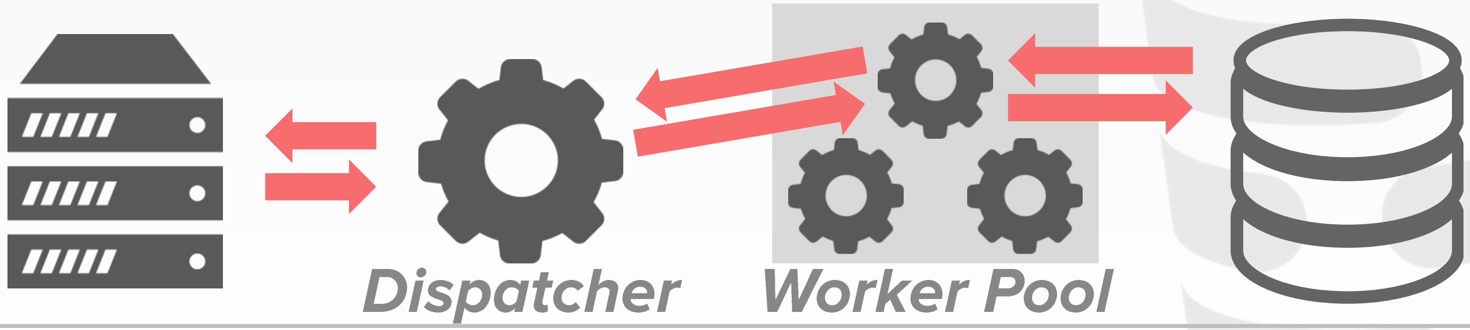


PROCESS POOL

A worker uses any process that is free in a pool

→ Still relies on OS scheduler and shared memory.

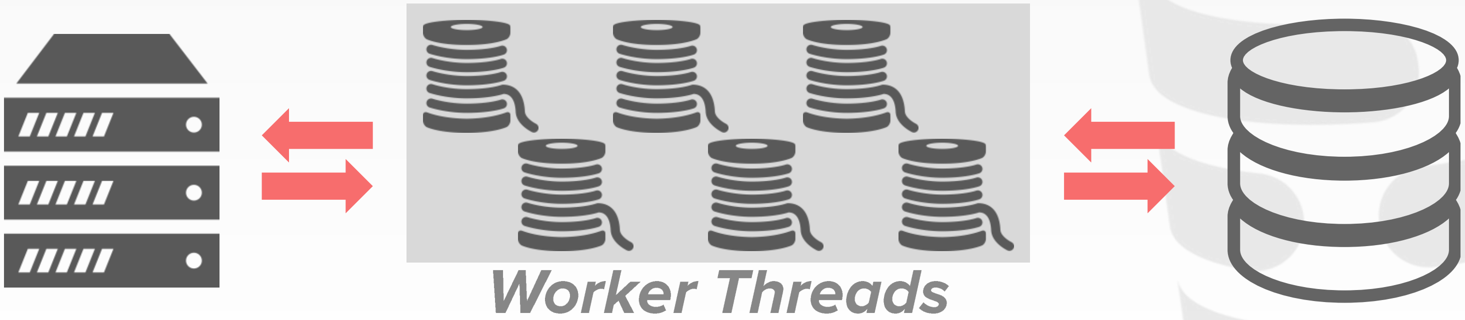
→ Bad for CPU cache locality.



THREAD PER WORKER

Single process with multiple worker threads.

→ DBMS has to manage its own scheduling.
→ May or may not use a dispatcher thread.
→ Thread crash (may) kill the entire system.



PROCESS MODELS

Using a multi-threaded architecture has several advantages:

→ Less overhead per context switch.
→ Don't have to manage shared memory.

The thread per worker model does not mean that you have intra-query parallelism.

SCHEDULING

For each query plan, the DBMS has to decide 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-QUERY PARALLELISM

Improve overall performance by allowing multiple queries to execute simultaneously.

→ Provide the illusion of isolation through **concurrency control** scheme.

This is really hard.

We will discuss more in 2 weeks.

INTRA-QUERY PARALLELISM

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

→ **Approach #1: Intra-Operator**

→ **Approach #2: Inter-Operator**

These techniques are <u>not</u> mutually exclusive.

There are parallel algorithms for every relational operator.

INTRA-OPERATOR PARALLELISM

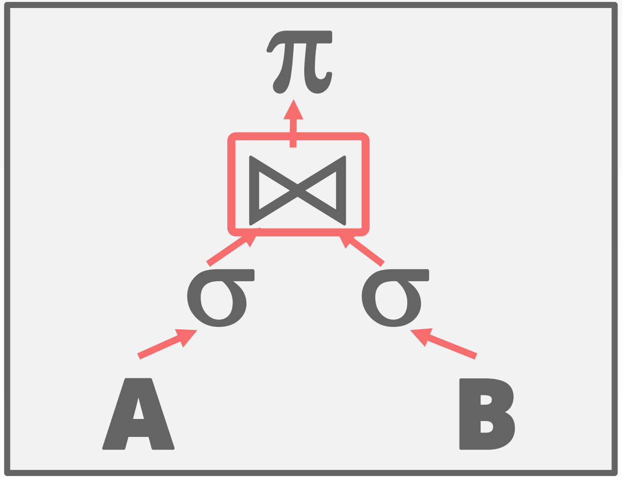
Approach #1: Intra-Operator (Horizontal)

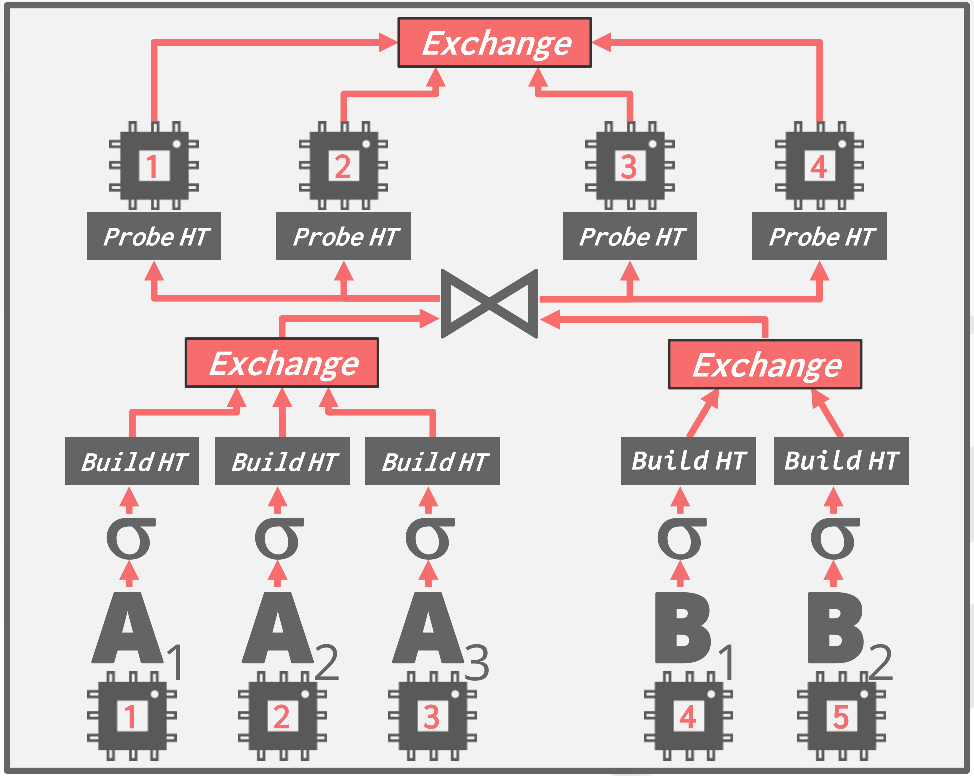
→ Operators are decomposed into independent instances that perform the same function on different subsets of data.

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

INTRA-OPERATOR PARALLELISM

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





INTER-OPERATOR PARALLELISM

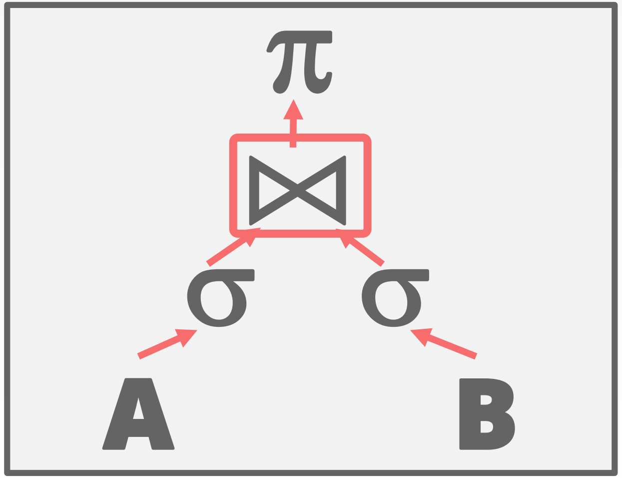
Approach #2: Inter-Operator (Vertical)

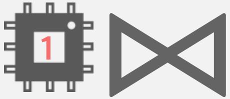
→ Operations are overlapped in order to pipeline data from one stage to the next without materialization.

Also called **pipelined parallelism**.

INTER-OPERATOR PARALLELISM

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

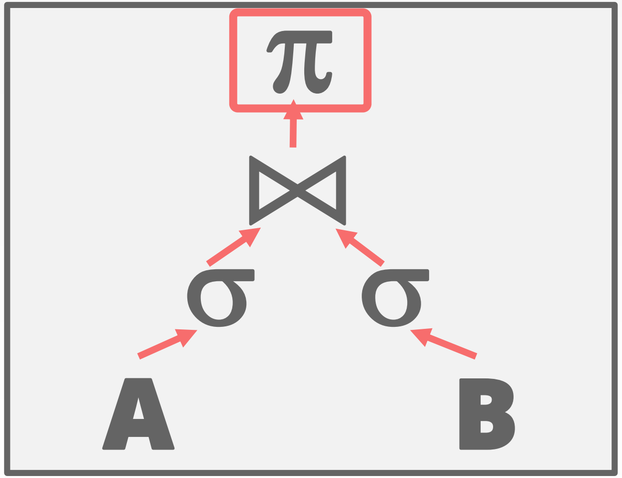


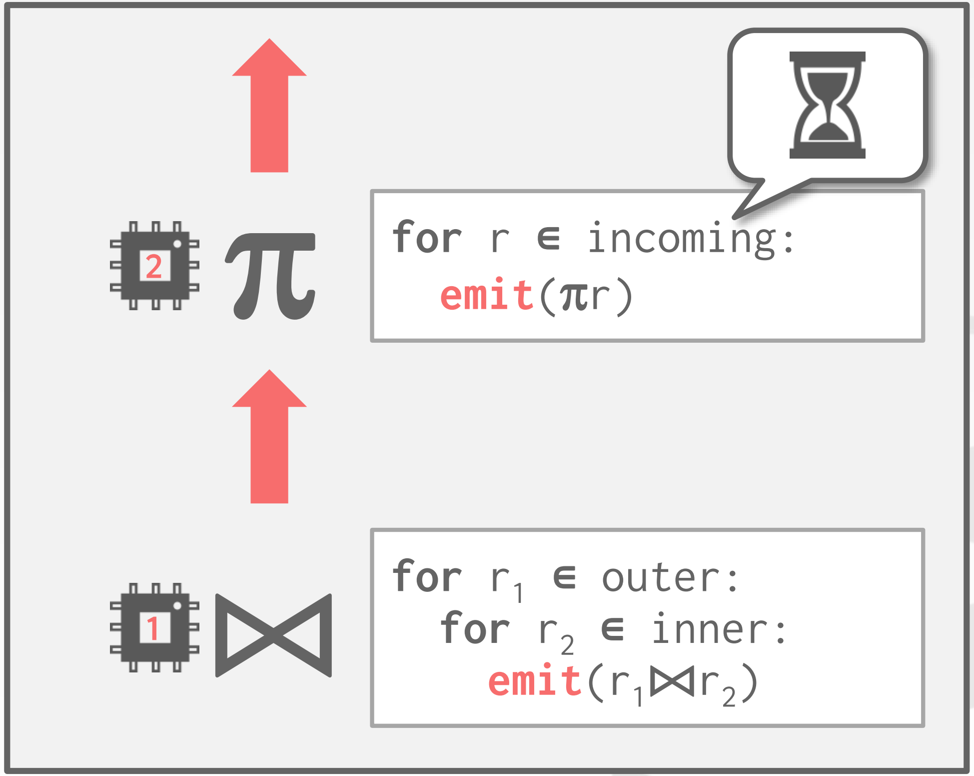



```
for r1 ∈ outer:
  for r2 ∈ inner:
    emit(r1⋈r2)
```

INTER-OPERATOR PARALLELISM

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





INTER-OPERATOR PARALLELISM

AFAIK, this approach is not widely used in traditional relational DBMSs.

→ Not all operators can emit output until they have seen all of the tuples from their children.

This is more common in **stream processing systems**.

APACHE Spark
APACHE nifi
kafka
APACHE STORM
Flink

OBSERVATION

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

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

I/O Parallelism

Split the DBMS installation across multiple storage devices.

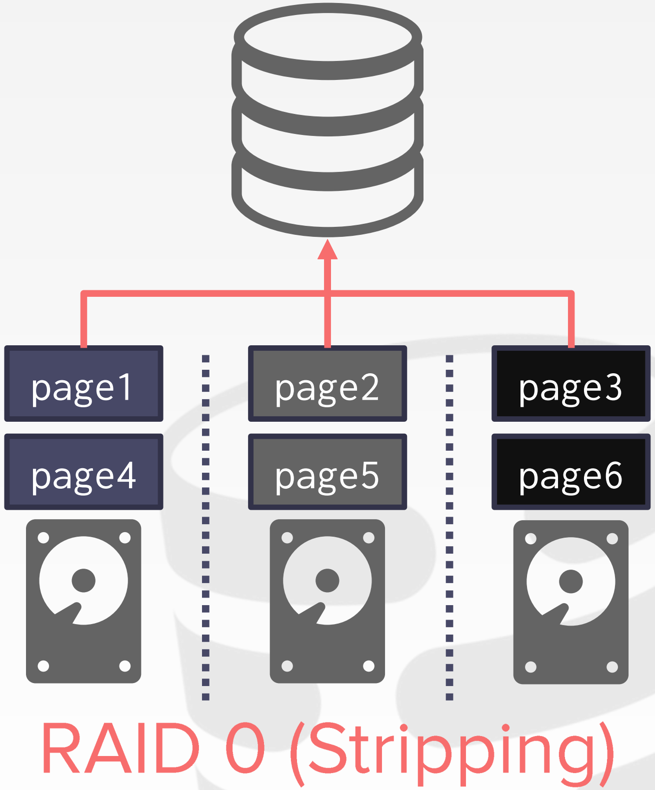
→ Multiple Disks per Database
→ One Database per Disk
→ One Relation per Disk
→ Split Relation across Multiple Disks

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.

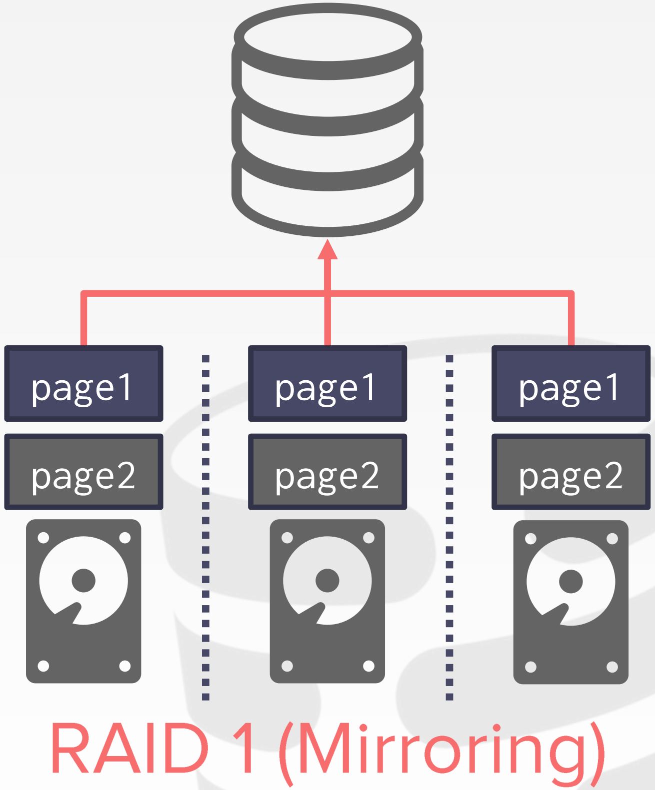


MULTI-DISK PARALLELISM

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

→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.



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 doesn't care how things are stored.
→ Not always true.

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

NEXT CLASS

MID-TERM EXAMINATION

After that we will have a "potpourri" lecture:
→ Stored Procedures
→ User-defined Functions
→ User-defined Types
→ Triggers
→ Views