Lecture #14

Carnegie Mellon University

# ADVANCED DATABASE SYSTEMS


## Scheduling

@Andy_Pavlo // 15-721 // Spring 2019

# CORRECTION

DPDK is available on Amazon EC2 since 2016.

**Ryan Worl** @ryanworl · Follow

@andy_pavlo The ENA in EC2 does support DPDK. See end of this article.

**Elastic Network Adapter – High Performance Network Interface for Amazon E...**
Many AWS customers create high-performance systems that run across multiple EC2 instances and make good use of all available network bandwidth. Over the years, we...
aws.amazon.com

11:17 AM - 28 Feb 2019

1

---

**Dor Laor** @DorLaor · Follow

Replying to @andy_pavlo @soumya_sd and 5 others

Cool! About dpdk, it does run on EC2 NICs and also para-virt NICs.

Besides our network stack, Scylla has lots on interesting mechanisms, from shard-per-cpu-core to sophisticate flow control:

**Worry-Free Ingestion: Flow Control of Writes in Scylla**
How Scylla ensures ingestion of data using a flow-control mechanism for tables with and without materialized views in Scylla Open Source 3.0 release.
scylladb.com

2:16 AM - 13 Dec 2018

1 Retweet 1 Like

1 1 1

# QUERY EXECUTION

A query plan is comprised of **operators**.

An **operator instance** is an invocation of an operator on some segment of data.

A **task** is the execution of a sequence of one or more operator instances.

# 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.

# TODAY'S AGENDA

Process Models
Data Placement
Scheduling

# 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.

ARCHITECTURE OF A DATABASE SYSTEM
FOUNDATIONS AND TRENDS IN DATABASES 2007

# PROCESS MODELS

**Approach #1: Process per DBMS Worker**

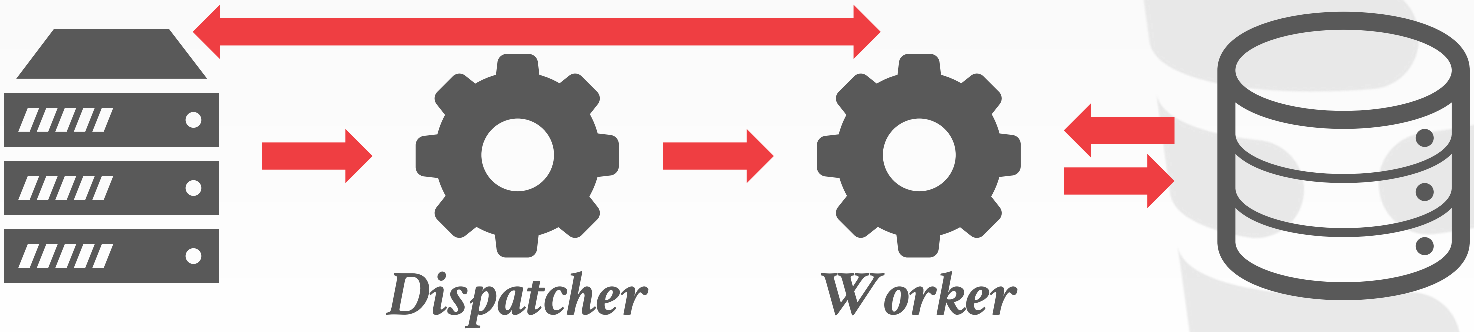
**Approach #2: Process Pool**

**Approach #3: Thread per DBMS Worker**

# PROCESS PER WORKER

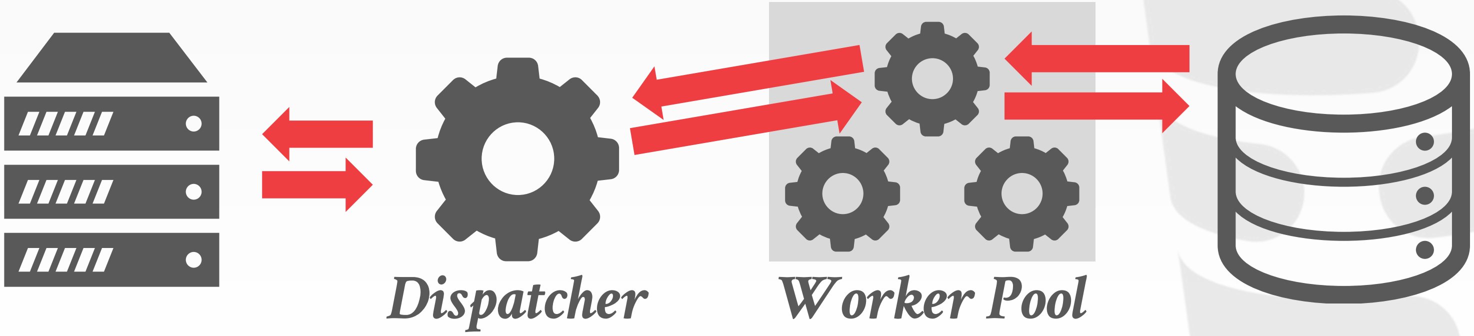
Each worker is a separate OS process.

→ 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

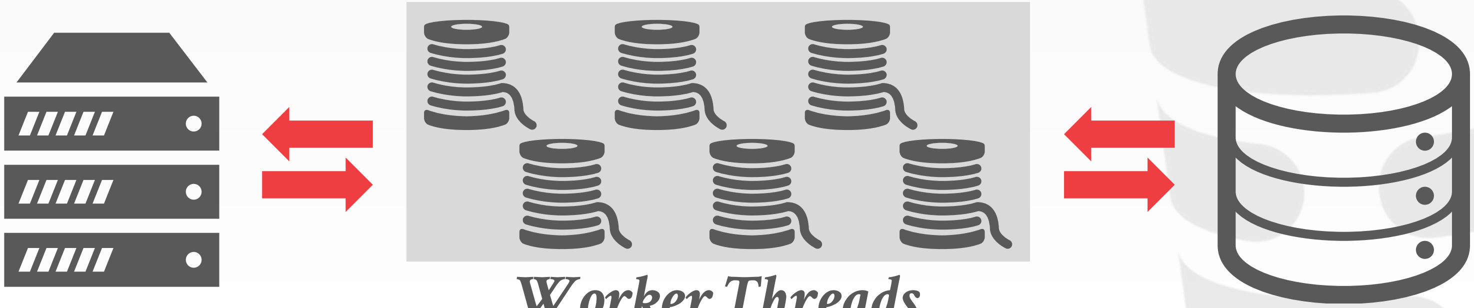
A worker uses any process that is free in a pool
→ Still relies on OS scheduler and shared memory.
→ Bad for CPU cache locality.
→ Examples: IBM DB2, Postgres (2015)



# THREAD PER WORKER

Single process with multiple worker threads.

→ DBMS has to manage 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.
→ Don't have to manage shared memory.

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

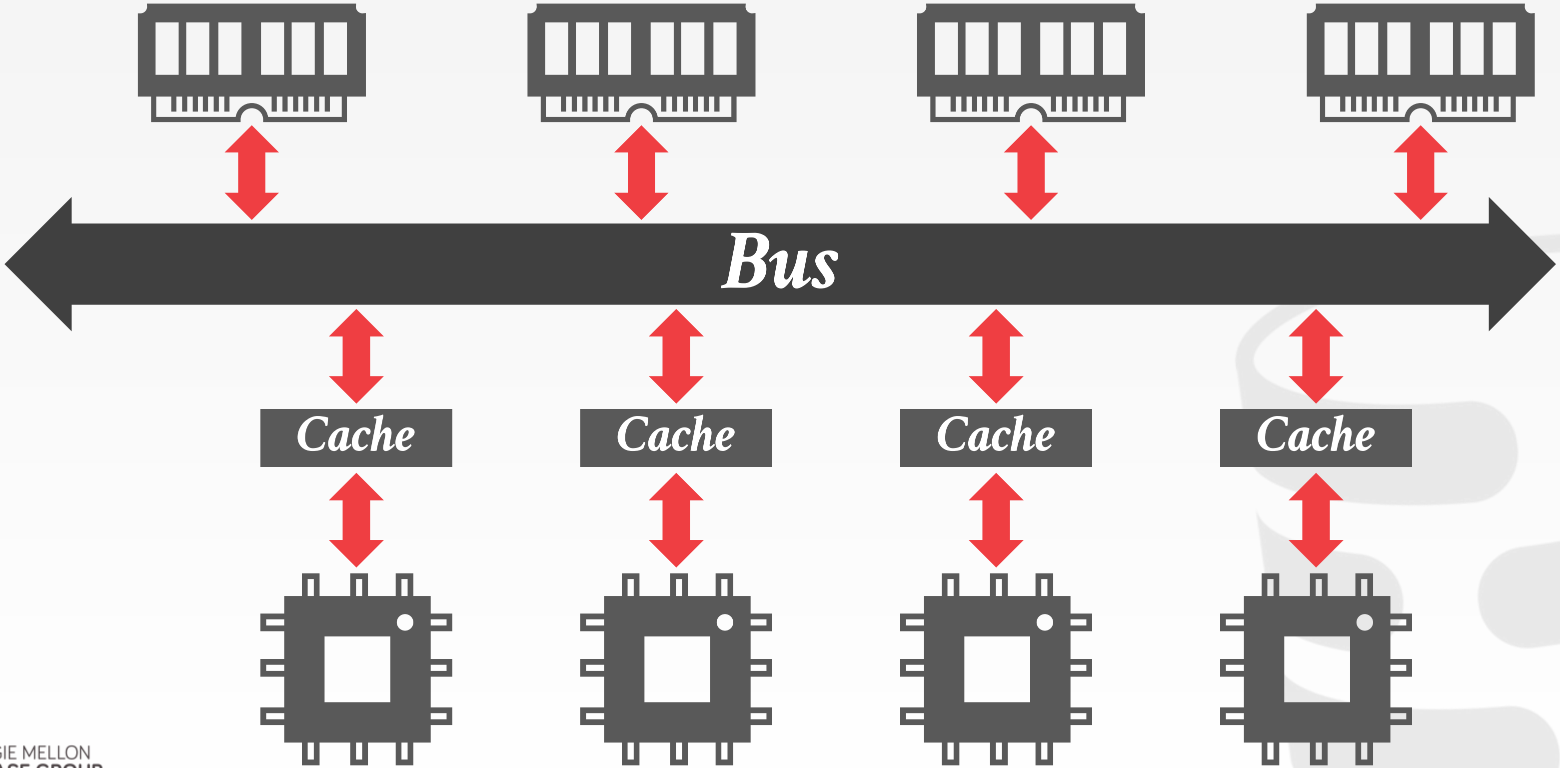
I am not aware of any new DBMS built in the last 10 years that doesn't use threads.

# OBSERVATION

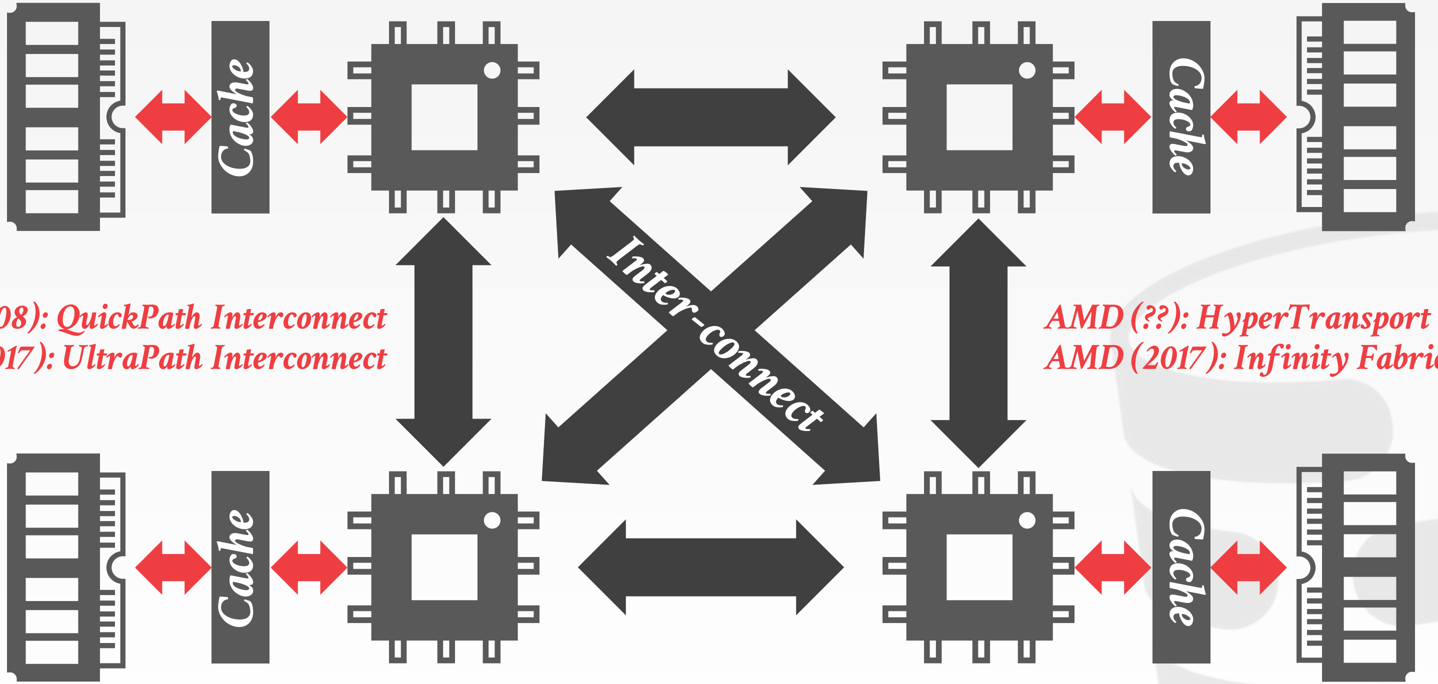
Regardless of what worker allocation or task assignment policy the DBMS uses, it's important that workers operate on local data.

The DBMS's scheduler has to be aware of it's underlying hardware's memory layout.
→ Uniform vs. Non-Uniform Memory Access

# UNIFORM MEMORY ACCESS



# NON-UNIFORM MEMORY ACCESS




*Intel (2008): QuickPath Interconnect*
*Intel (2017): UltraPath Interconnect*

*AMD (??): HyperTransport*
*AMD (2017): Infinity Fabric*

# DATA PLACEMENT

The DBMS can partition memory for a database and assign each partition to a CPU.

By controlling and tracking the location of partitions, it can schedule operators to execute on workers at the closest CPU core.

See Linux's **`move_pages`**

# MEMORY ALLOCATION

What happens when the DBMS calls **malloc**?

→ Assume that the allocator doesn't already have an chunk of memory that it can give out.

Actually, almost nothing:

→ The allocator will extend the process' data segment.

→ But this new virtual memory is not immediately backed by physical memory.

→ The OS only allocates physical memory when there is a page fault.

# MEMORY ALLOCATION LOCATION

Now after a page fault, where does the OS allocate physical memory in a NUMA system?

**Approach #1: Interleaving**

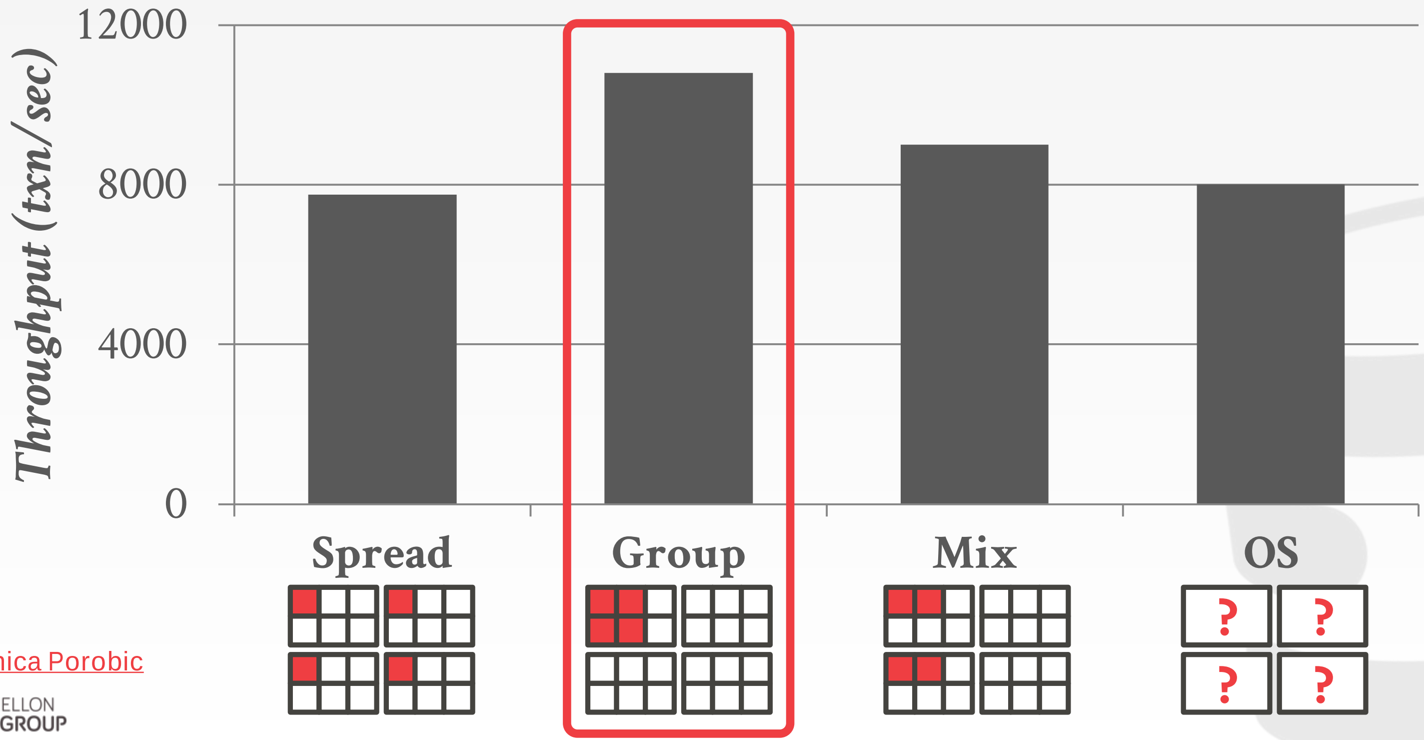
→ Distribute allocated memory uniformly across CPUs.

**Approach #2: First-Touch**

→ At the CPU of the thread that accessed the memory location that caused the page fault.

# DATA PLACEMENT – OLTP

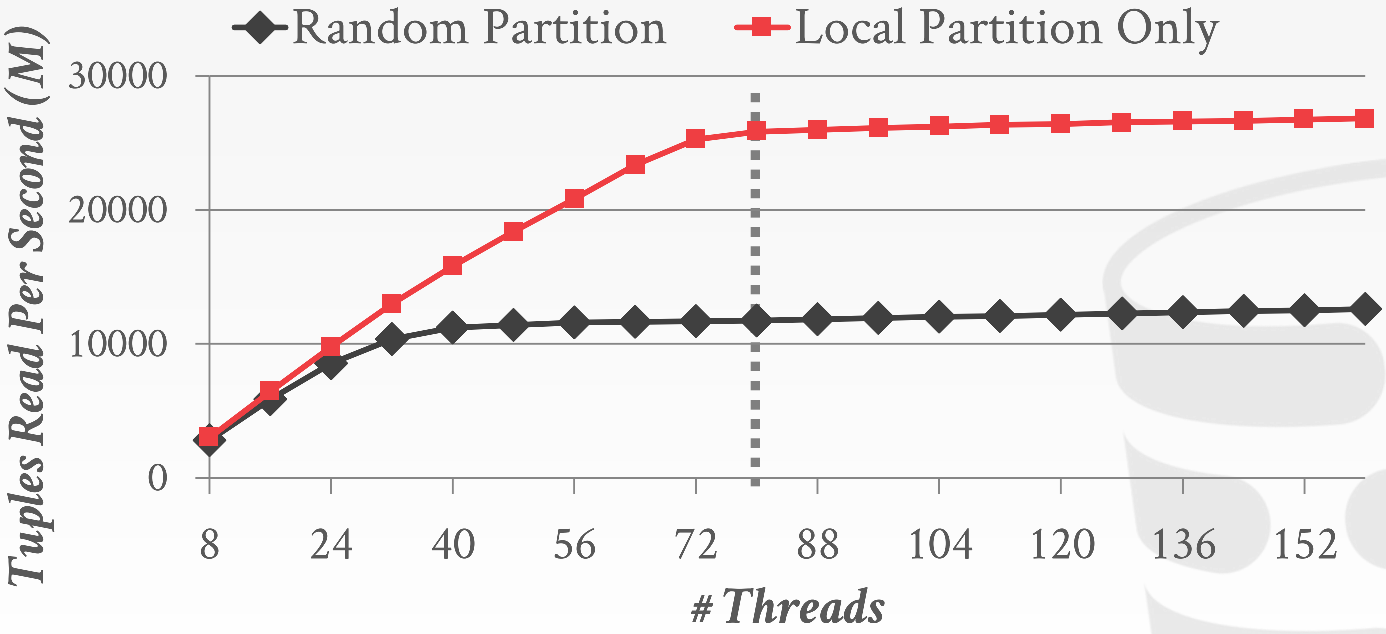
*Workload: TPC-C Payment using 4 Workers*
*Processor: NUMA with 4 sockets (6 cores each)*




Source: [Danica Porobic]

# DATA PLACEMENT – OLAP

***Sequential Scan on 10m tuples***
***Processor: 8 sockets, 10 cores per node (2x HT)***




Source: Haibin Lin

# PARTITIONING VS. PLACEMENT

A **partitioning** scheme is used to split the database based on some policy.

→ Round-robin
→ Attribute Ranges
→ Hashing
→ Partial/Full Replication

A **placement** scheme then tells the DBMS where to put those partitions.

→ Round-robin
→ Interleave across cores

# OBSERVATION

We have the following so far:

→ Process Model
→ Worker Allocation Model
→ Task Assignment Model
→ Data Placement Policy

But how do we decide how to create a set of tasks from a logical query plan?

→ This is relatively easy for OLTP queries.
→ Much harder for OLAP queries…

# STATIC SCHEDULING

The DBMS decides how many threads to use to execute the query when it generates the plan.

It does **not** change while the query executes.

→ The easiest approach is to just use the same # of tasks as the # of cores.

# MORSEL-DRIVEN SCHEDULING

Dynamic scheduling of tasks that operate over horizontal partitions called "morsels" that are distributed across cores.

→ One worker per core
→ Pull-based task assignment
→ Round-robin data placement

Supports parallel, NUMA-aware operator implementations.

MORSEL-DRIVEN PARALLELISM: A NUMA-AWARE QUERY EVALUATION FRAMEWORK FOR THE MANY-CORE AGE
SIGMOD 2014

# HYPER – ARCHITECTURE

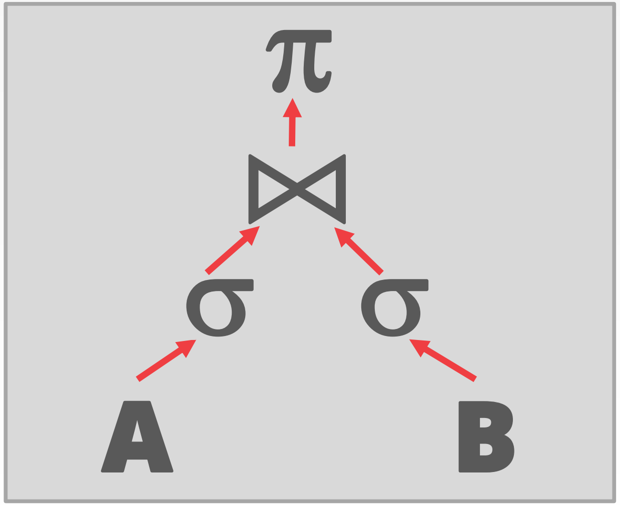
No separate dispatcher thread.

The threads perform cooperative scheduling for each query plan using a single task queue.

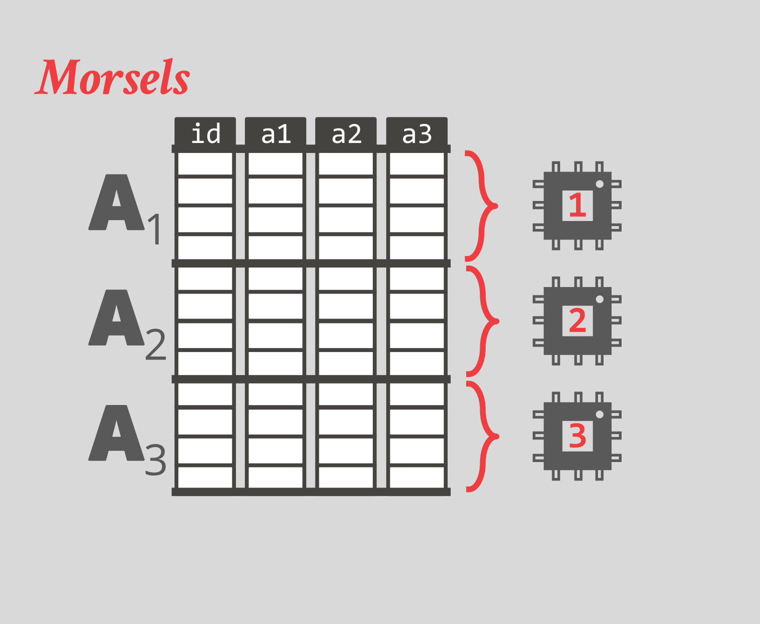
→ Each worker tries to select tasks that will execute on morsels that are local to it.
→ If there are no local tasks, then the worker just pulls the next task from the global work queue.

# HYPER – DATA PARTITIONING

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

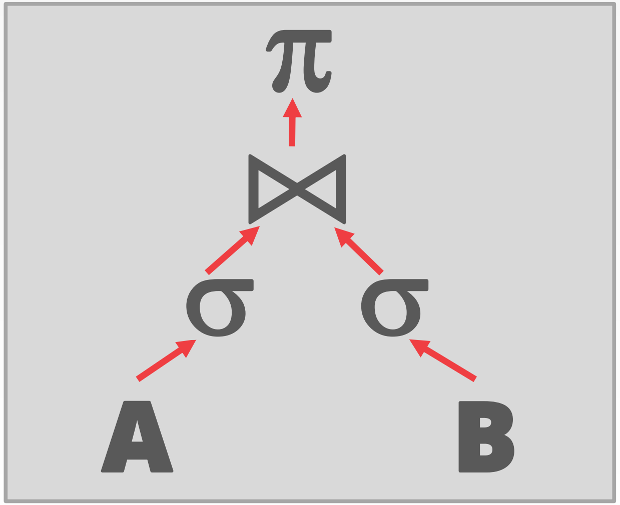



## *Data Table*

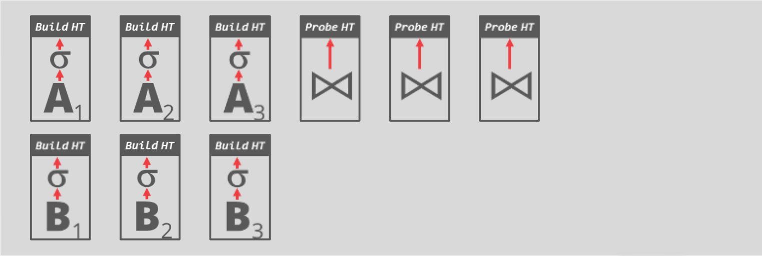


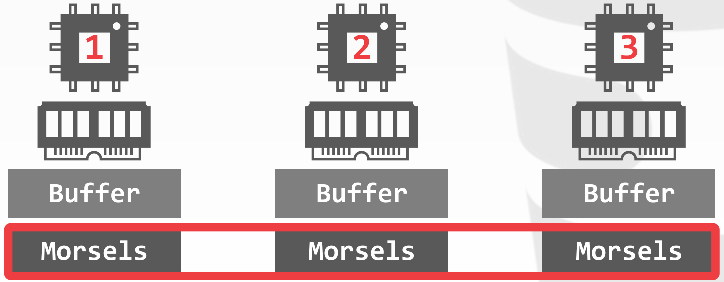
# HYPER – EXECUTION EXAMPLE

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



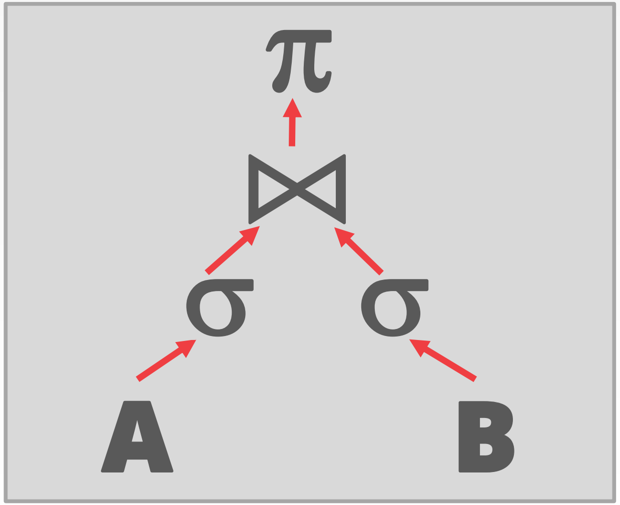

## *Global Task Queue*



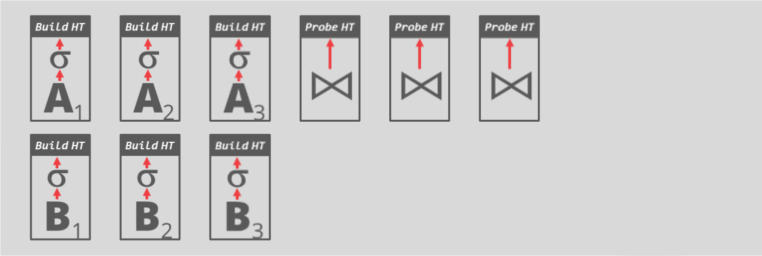



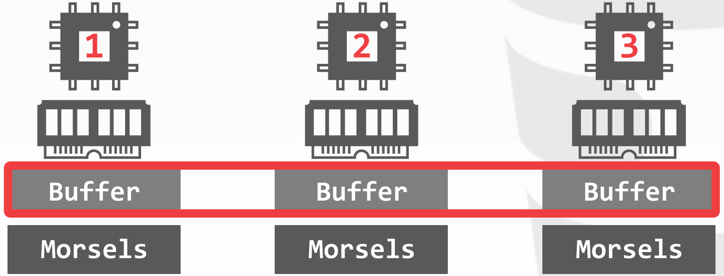
# HYPER – EXECUTION EXAMPLE

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



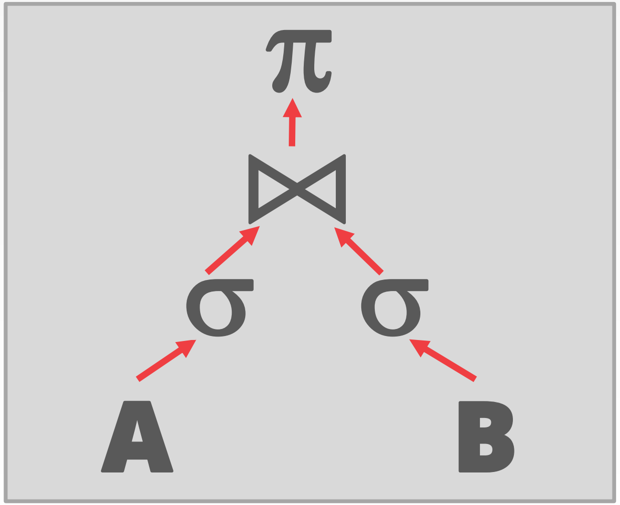

## *Global Task Queue*

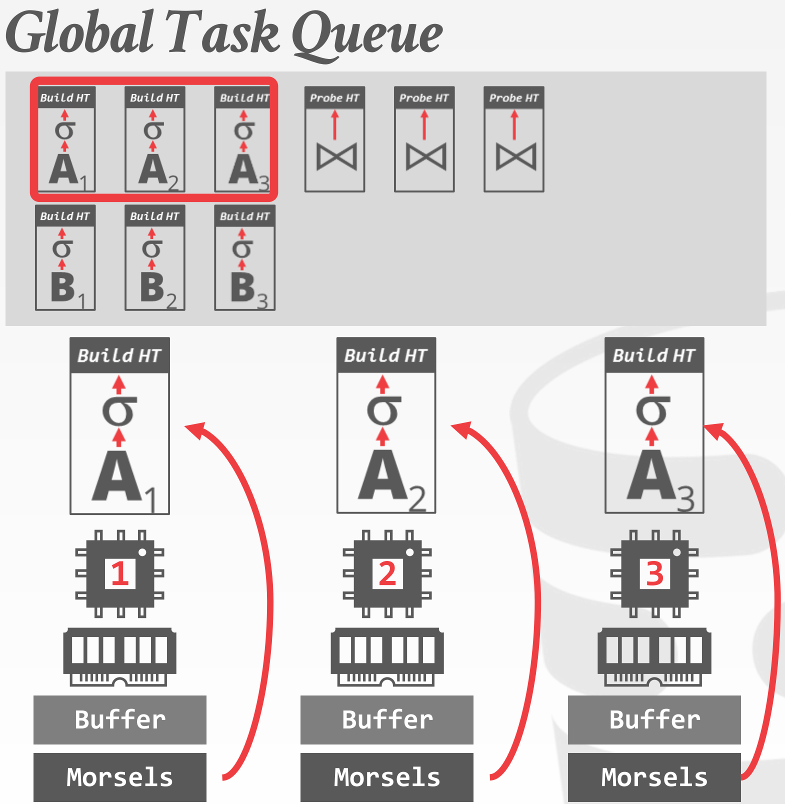


# HYPER – EXECUTION EXAMPLE

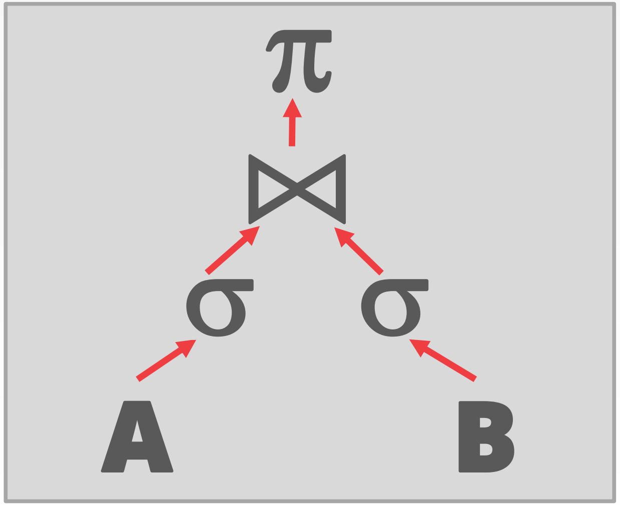
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

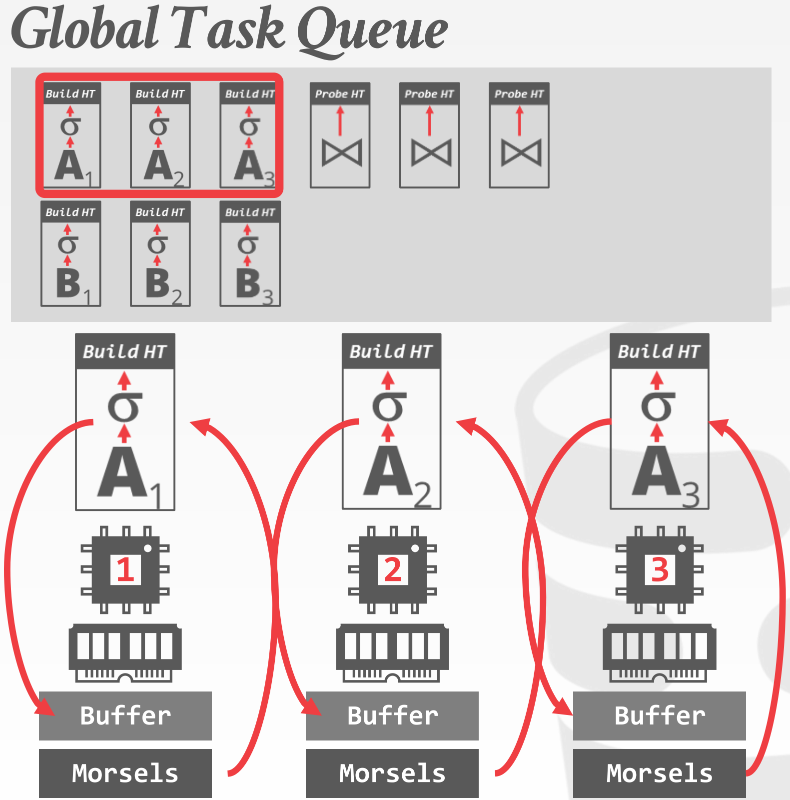


# HYPER – EXECUTION EXAMPLE

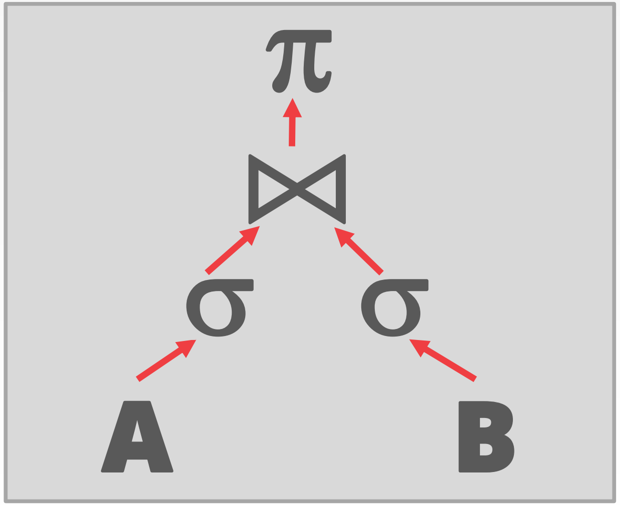
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```



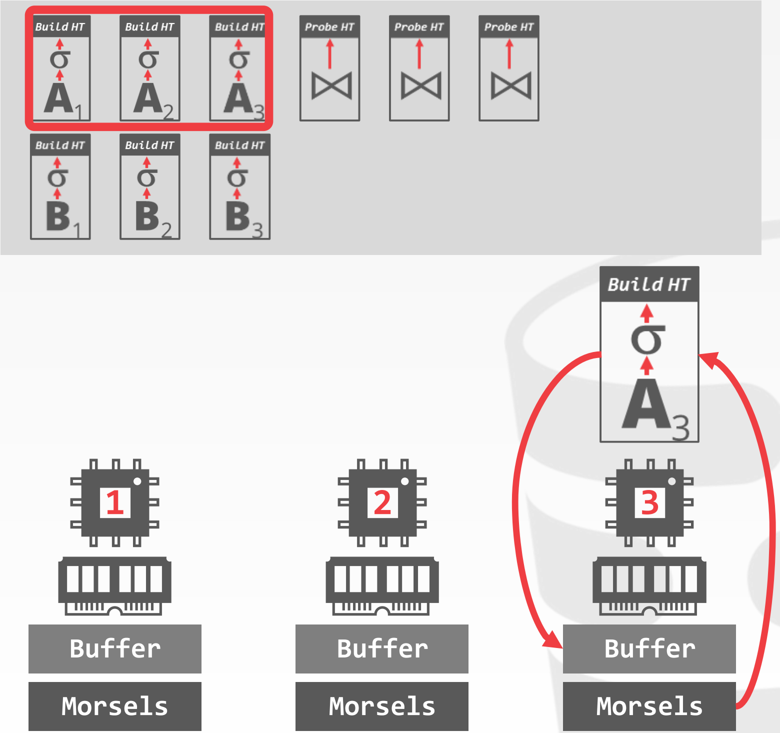



# HYPER – EXECUTION EXAMPLE

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

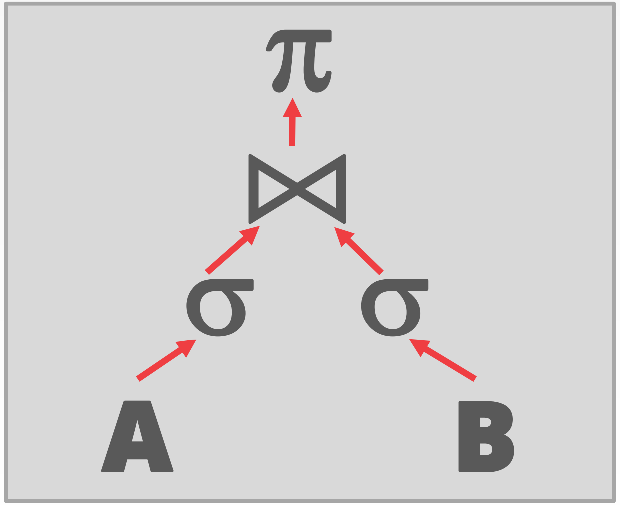



*Global Task Queue*

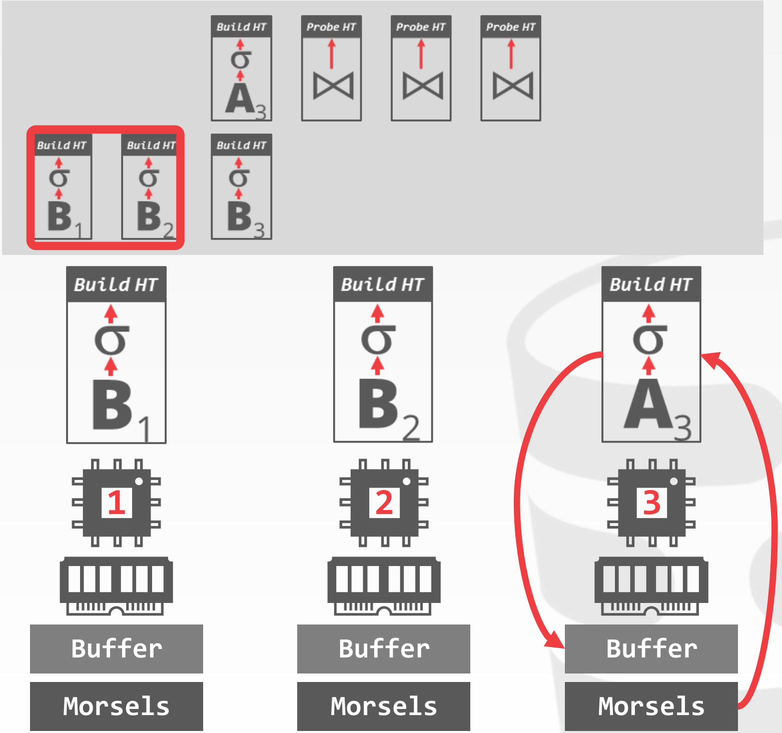


# HYPER – EXECUTION EXAMPLE

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

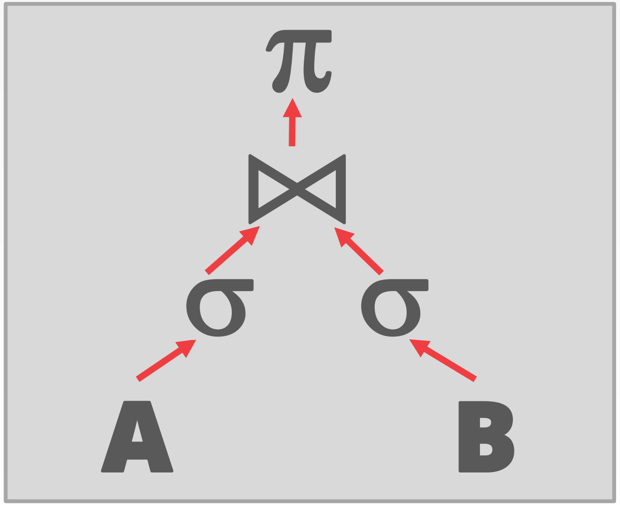


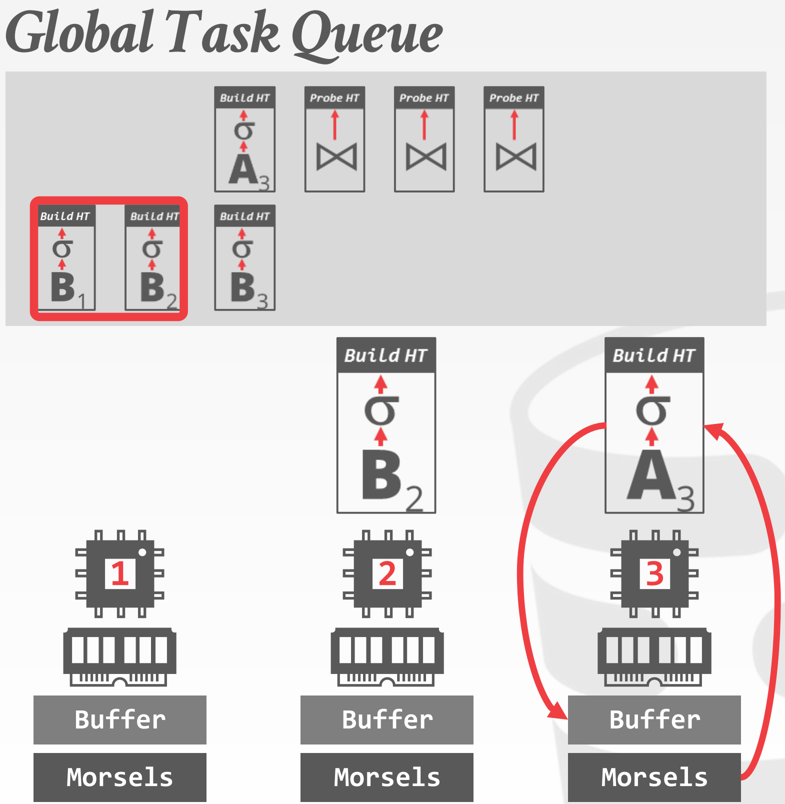

## *Global Task Queue*



# HYPER – EXECUTION EXAMPLE

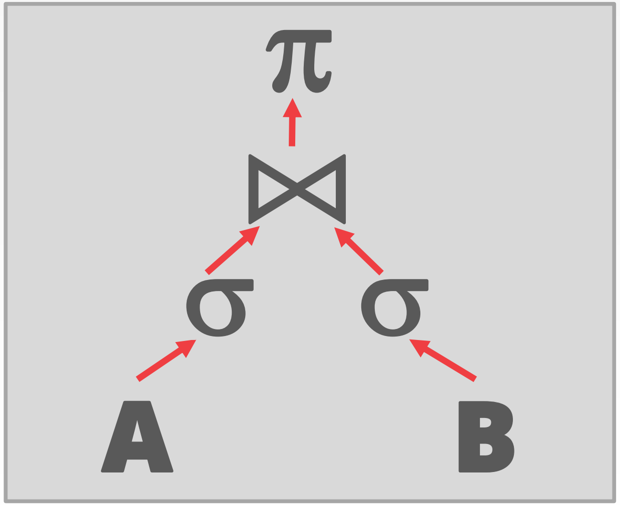
```sql
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```



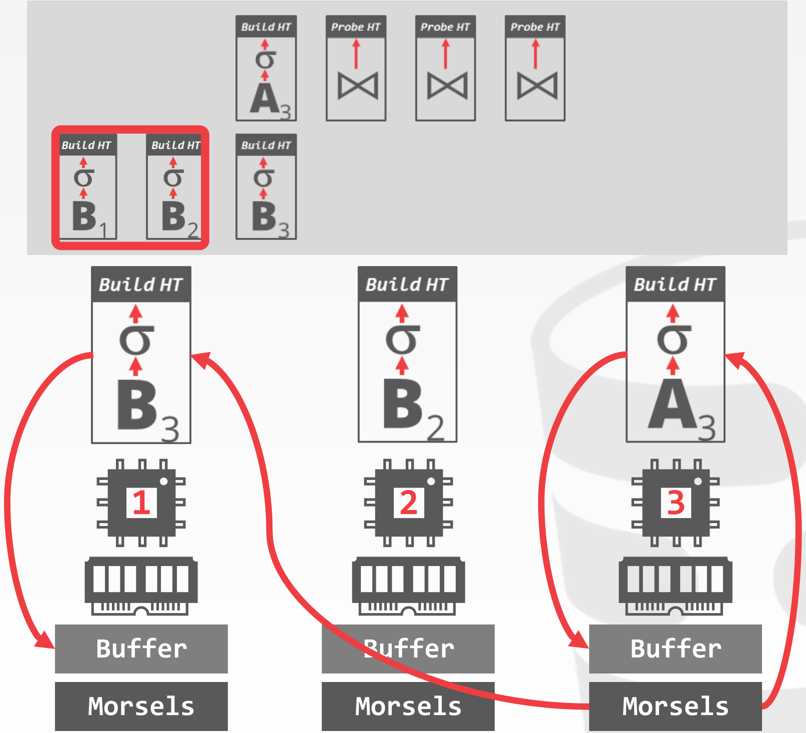



# HYPER – EXECUTION EXAMPLE

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




## *Global Task Queue*



# MORSEL-DRIVEN SCHEDULING

Because there is only one worker per core, they have to use work stealing because otherwise threads could sit idle waiting for stragglers.

Uses a lock-free hash table to maintain the global work queues.

→ We will discuss hash tables next class…

# SAP HANA – NUMA-AWARE SCHEDULER

Pull-based scheduling with multiple worker threads that are organized into groups (pools).

→ Each CPU can have multiple groups.
→ Each group has a soft and hard priority queue.

Uses a separate "watchdog" thread to check whether groups are saturated and can reassign tasks dynamically.

SCALING UP CONCURRENT MAIN-MEMORY COLUMN-STORE SCANS: TOWARDS ADAPTIVE NUMA-AWARE DATA AND TASK PLACEMENT
VLDB 2015

# SAP HANA – THREAD GROUPS

Each thread group has a soft and hard priority task queues.

→ Threads are allowed to steal tasks from other groups' soft queues.

Four different pools of thread per group:

→ **Working**: Actively executing a task.
→ **Inactive**: Blocked inside of the kernel due to a latch.
→ **Free**: Sleeps for a little, wake up to see whether there is a new task to execute.
→ **Parked**: Like free but doesn't wake up on its own.

# SAP HANA – NUMA-AWARE SCHEDULER

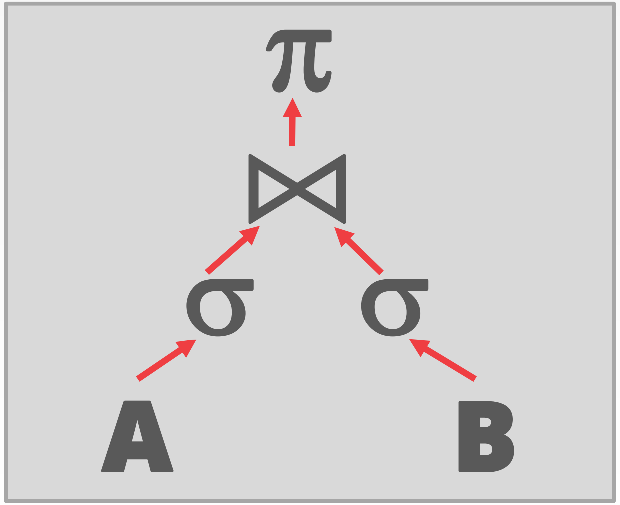
Can dynamically adjust thread pinning based on whether a task is CPU or memory bound.

Found that work stealing was not as beneficial for systems with a larger number of sockets.

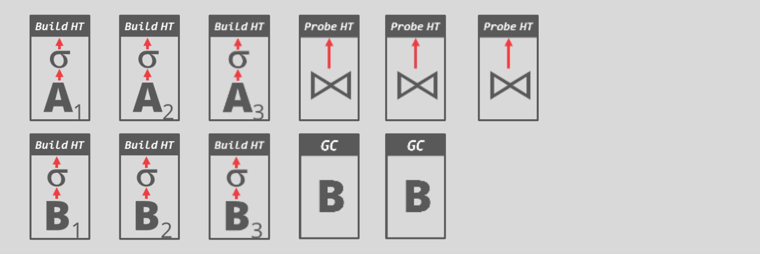
Using thread groups allows cores to execute other tasks instead of just only queries.

# HANA NUMA-AWARE SCHEDULER

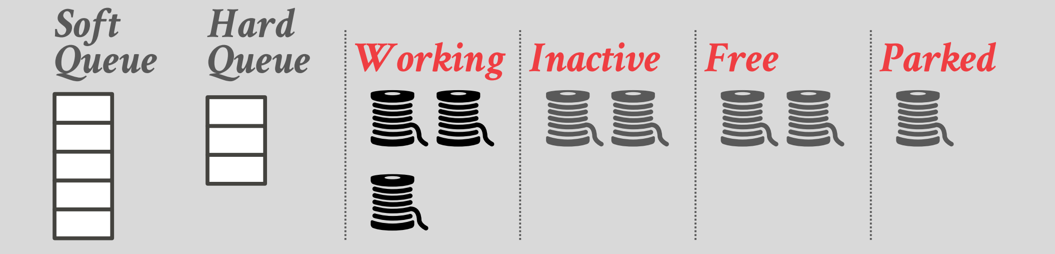
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```




## Tasks

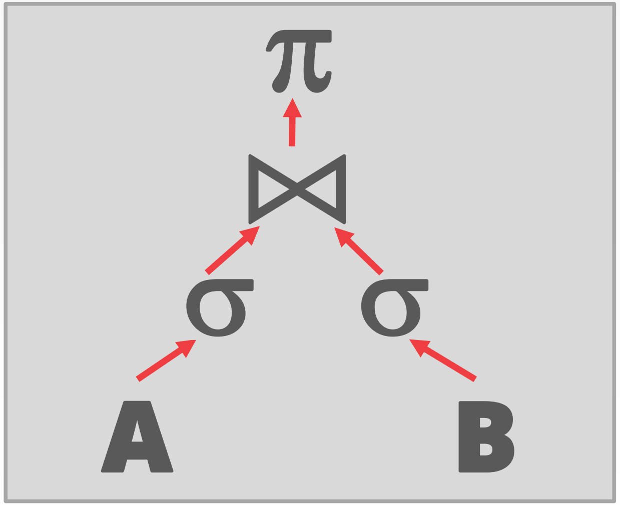



## Thread Group

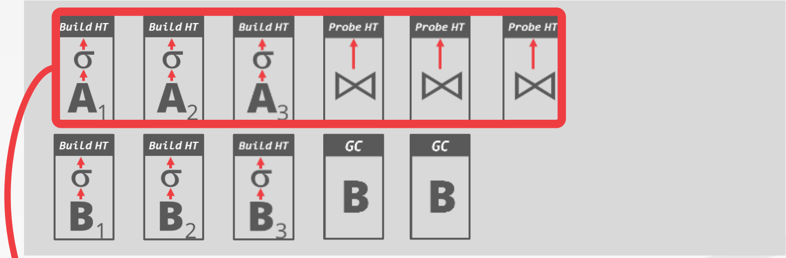


# HANA NUMA-AWARE SCHEDULER

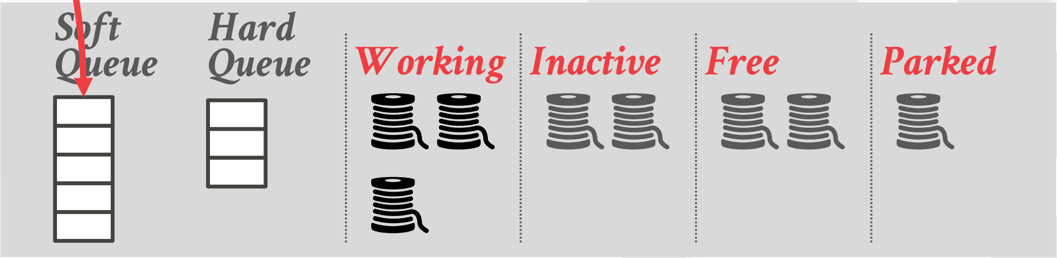
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```




## *Tasks*

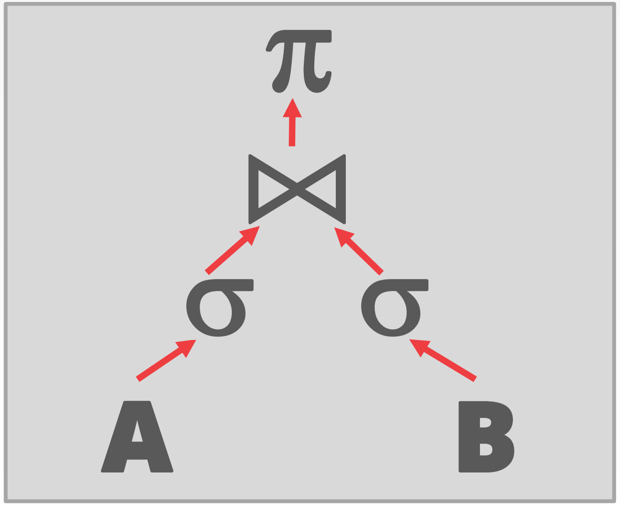



## *Thread Group*

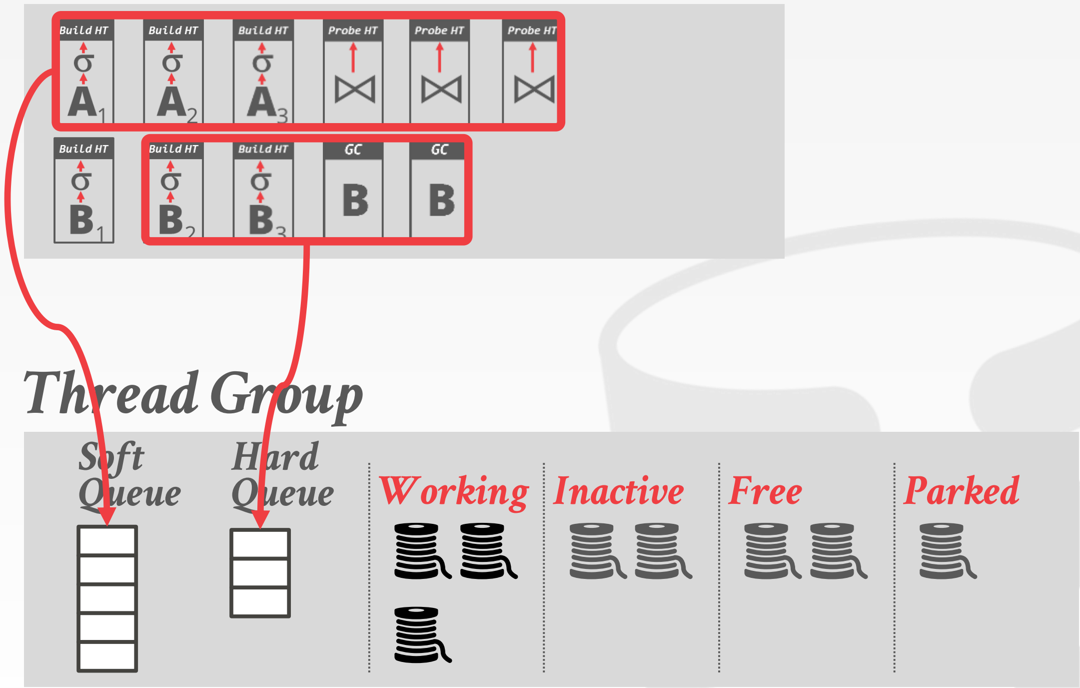


# HANA NUMA-AWARE SCHEDULER

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

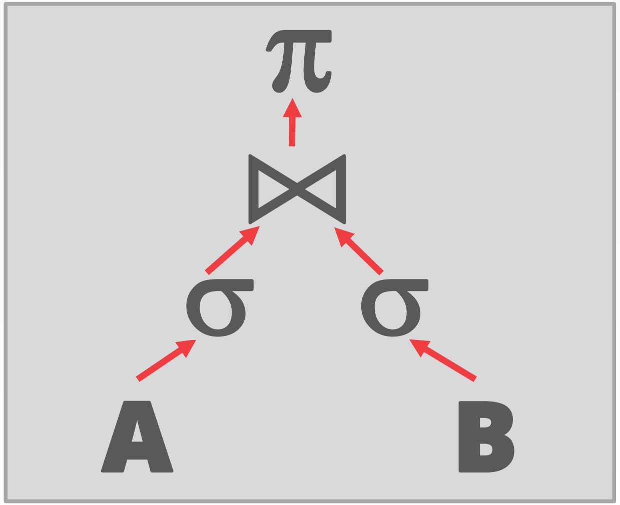


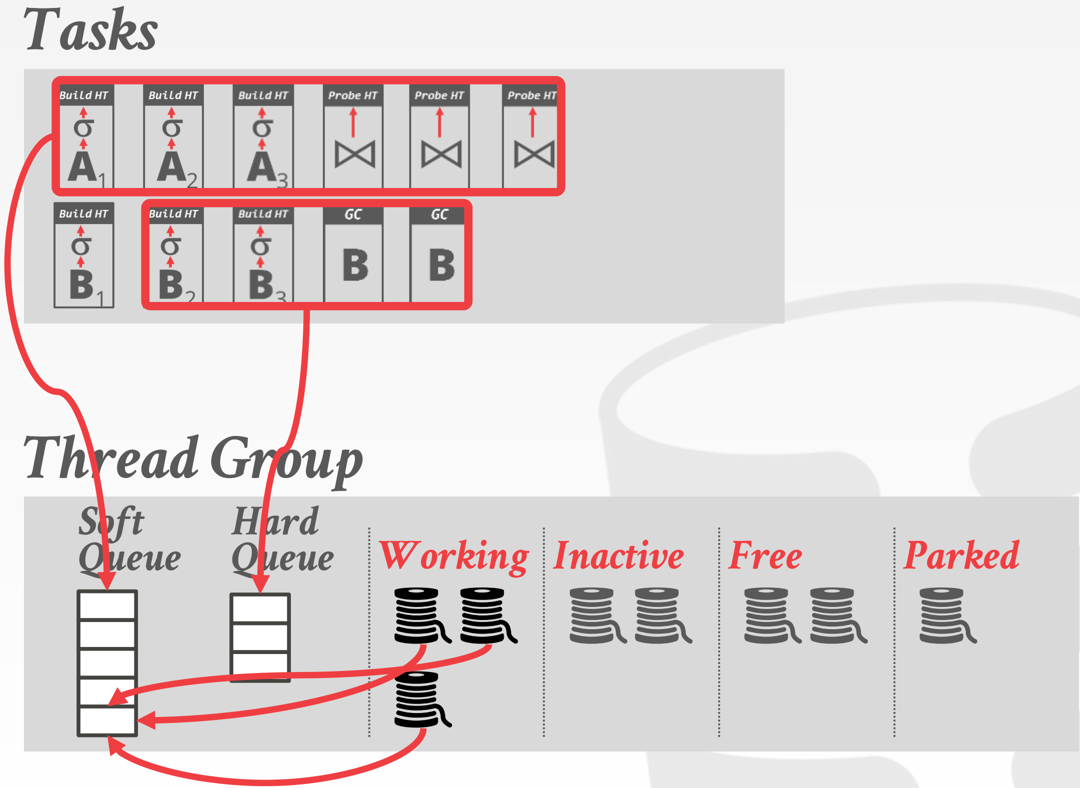

## Tasks



# HANA NUMA-AWARE SCHEDULER

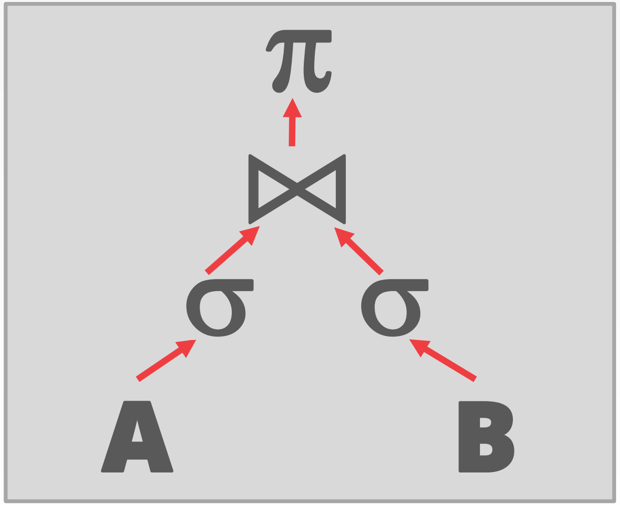
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```



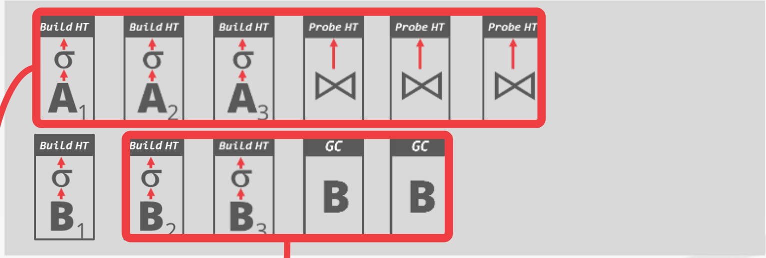



# HANA NUMA-AWARE SCHEDULER

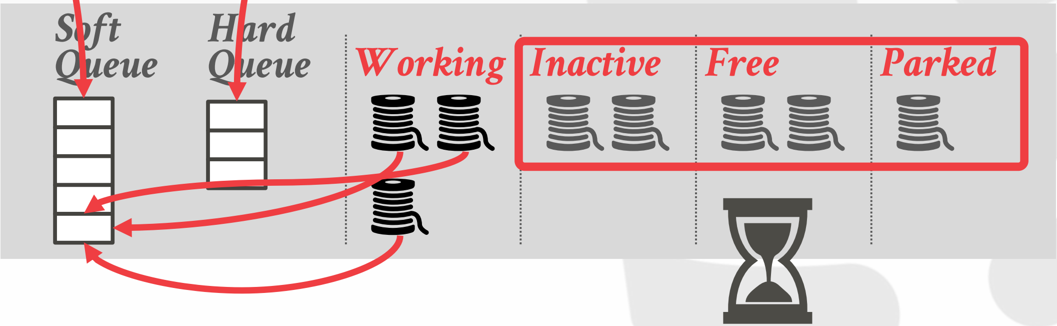
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```




## *Tasks*

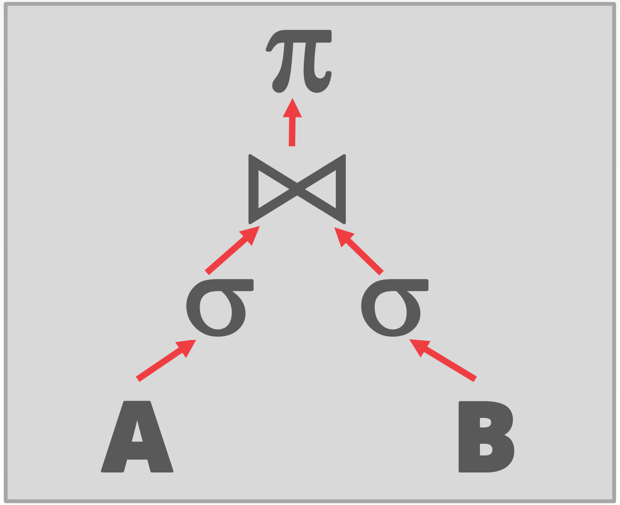


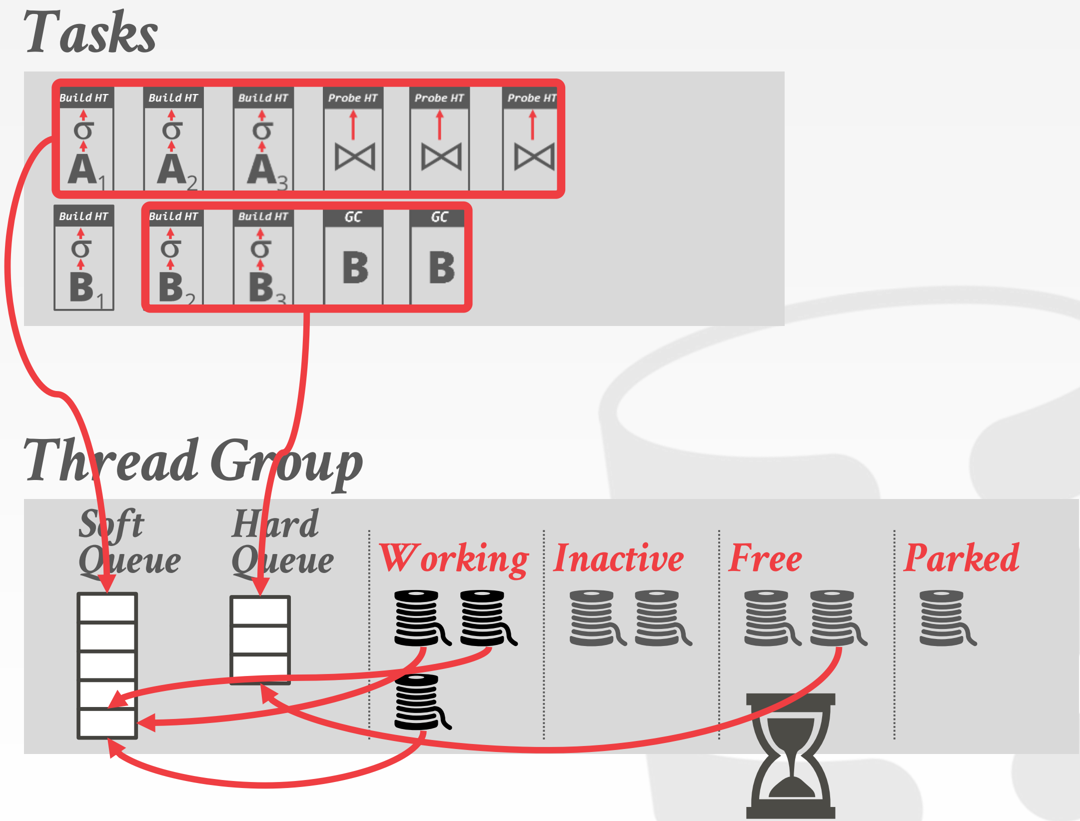

## *Thread Group*



# HANA NUMA-AWARE SCHEDULER

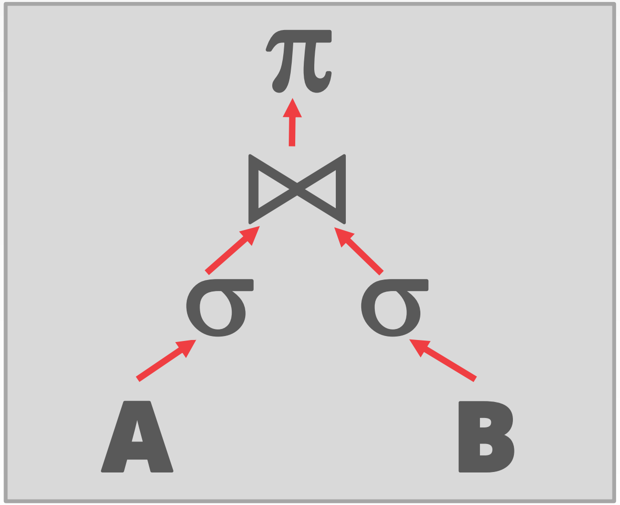
```
SELECT A.id, B.value
  FROM A, B
 WHERE A.id = B.id
   AND A.value < 99
   AND B.value > 100
```

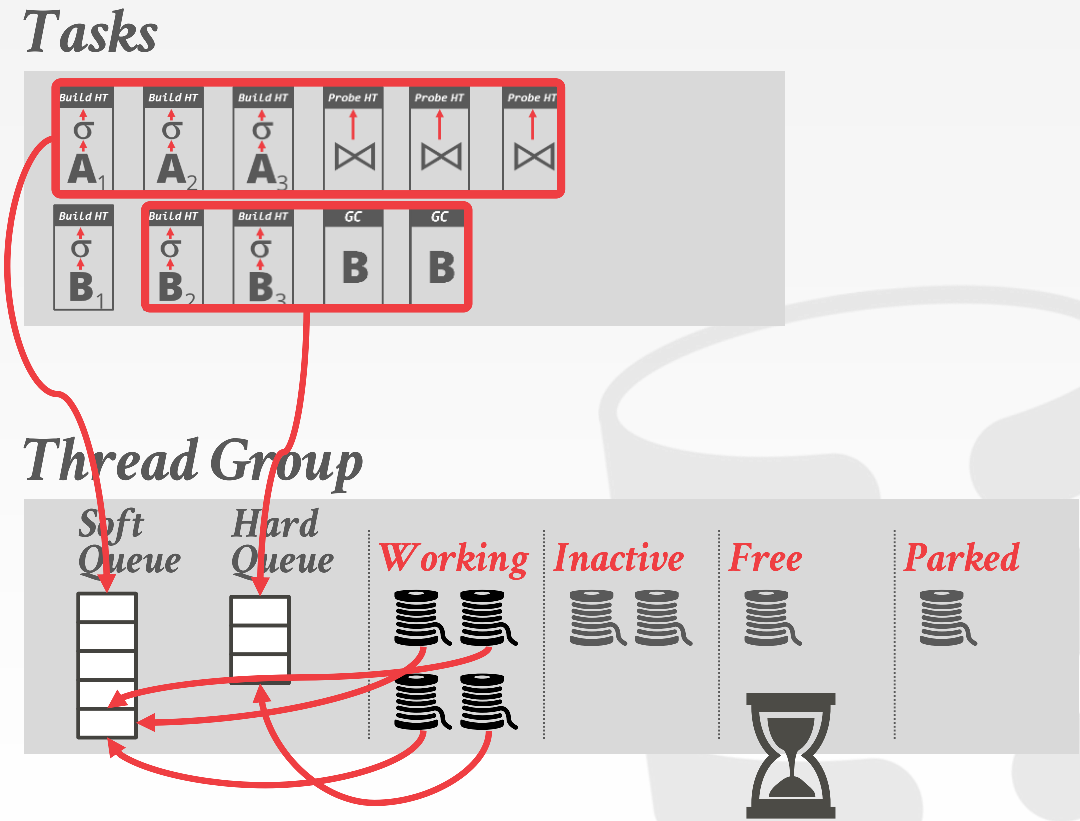


# HANA NUMA-AWARE SCHEDULER

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



# OBSERVATION

If requests arrive at the DBMS faster than it can execute them, then the system becomes overloaded.

The OS doesn't help us here:

→ CPU Bound: Do nothing
→ Memory Bound: OOM

Easiest DBMS Solution: Crash

# FLOW CONTROL

## Approach #2: Admission Control

→ Abort new requests when the system believes that it will not have enough resources to execute that request.

## Approach #1: Throttling

→ Delay the responses to clients to increase the amount of time between requests.
→ This assumes a synchronous submission scheme.

# PARTING THOUGHTS

A DBMS is a beautiful, strong-willed independent piece of software.

But it has to make sure that it uses its underlying hardware correctly.

→ Data location is an important aspect of this.
→ Tracking memory location in a single-node DBMS is the same as tracking shards in a distributed DBMS

Don't let the OS ruin your life.

# EXTRA CREDIT

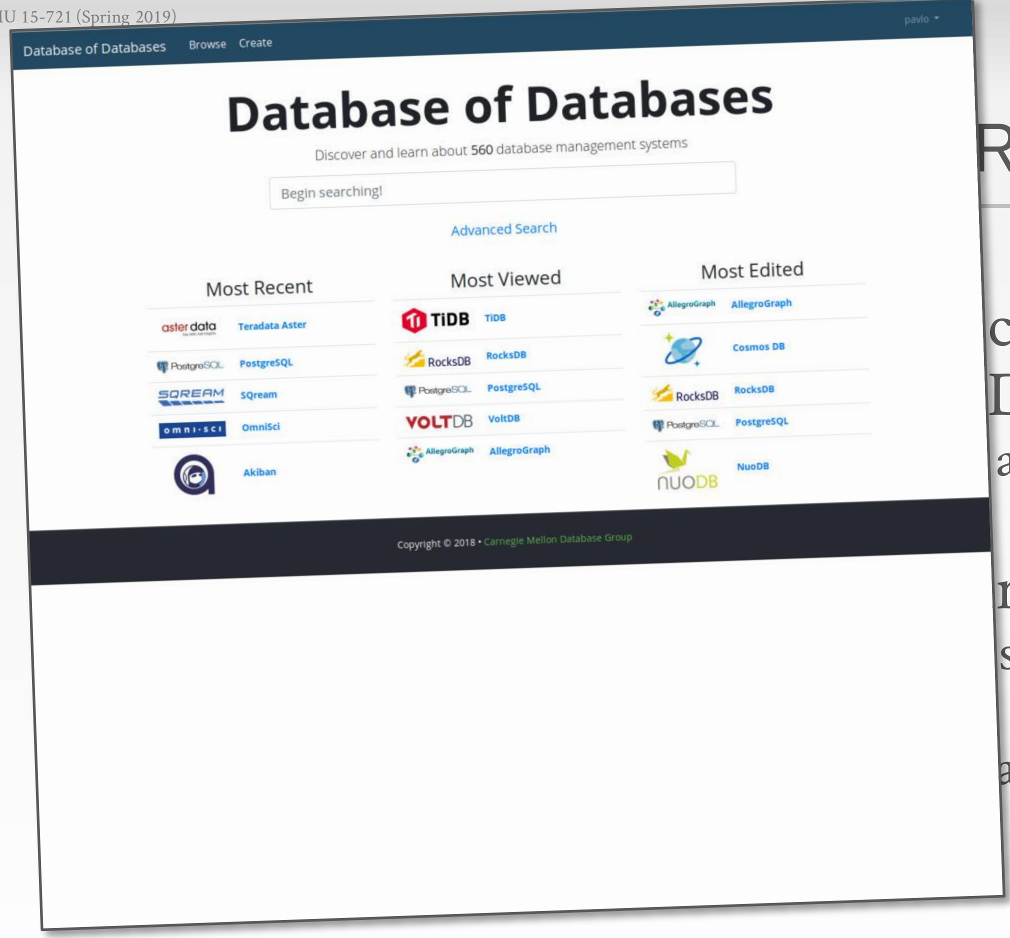
Each student can earn extra credit if they write a encyclopedia article about a DBMS.

→ Can be academic/commercial, active/historical.

Each article will use a standard taxonomy.

→ For each feature category, you select pre-defined options for your DBMS.

→ You will then need to provide a summary paragraph with citations for that category.




# REDIT

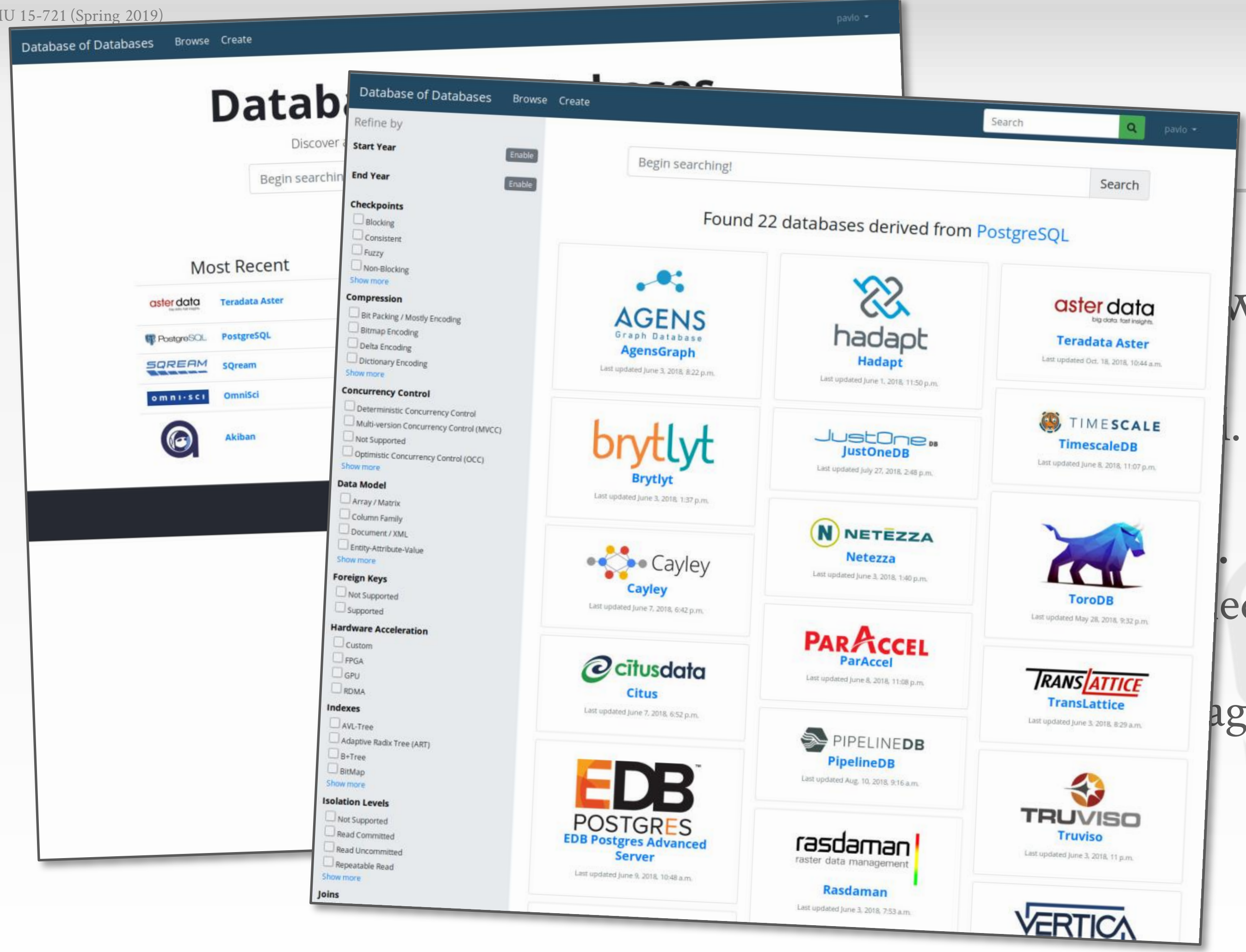
credit if they write a DBMS.

active/historical.

rd taxonomy.

select pre-defined options

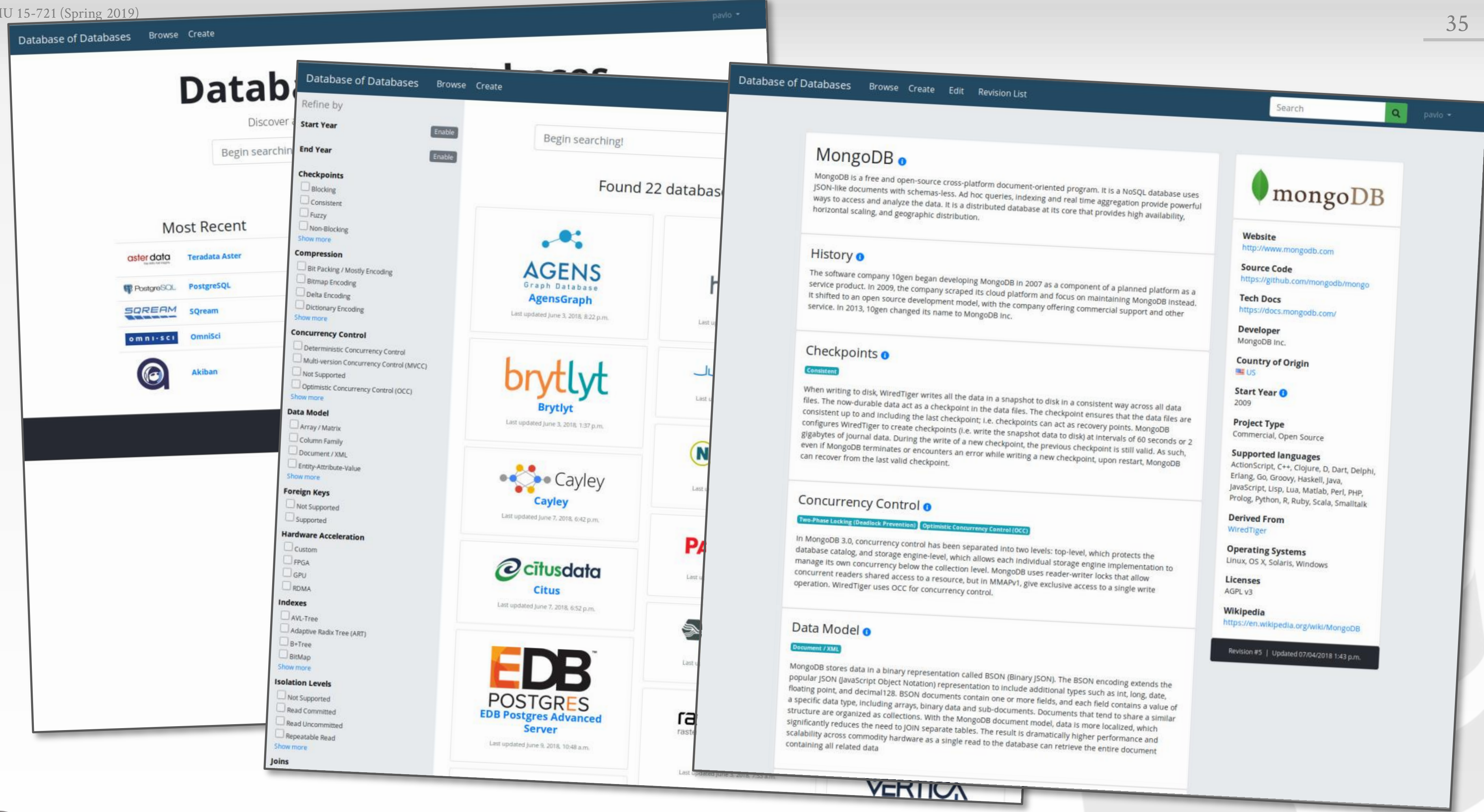
a summary paragraph with




write a

…ed options

…agraph with


Database of Databases
Browse
Create
Discover
Begin searching!
Most Recent
Teradata Aster
PostgreSQL
SQream
OmniSci
Akiban
Refine by
Start Year
End Year
Enable
Checkpoints
Blocking
Consistent
Fuzzy
Non-Blocking
Show more
Compression
Bit Packing / Mostly Encoding
Bitmap Encoding
Delta Encoding
Dictionary Encoding
Concurrency Control
Deterministic Concurrency Control
Multi-version Concurrency Control (MVCC)
Not Supported
Optimistic Concurrency Control (OCC)
Data Model
Array / Matrix
Column Family
Document / XML
Entity-Attribute-Value
Foreign Keys
Not Supported
Supported
Hardware Acceleration
Custom
FPGA
GPU
RDMA
Indexes
AVL-Tree
Adaptive Radix Tree (ART)
B+Tree
BitMap
Isolation Levels
Not Supported
Read Committed
Read Uncommitted
Repeatable Read
Joins
Found 22 databases
AGENS Graph Database
AgensGraph
Last updated June 3, 2018, 8:22 p.m.
brytlyt
Brytlyt
Last updated June 3, 2018, 1:37 p.m.
Cayley
Last updated June 7, 2018, 6:42 p.m.
citusdata
Citus
Last updated June 7, 2018, 6:52 p.m.
EDB POSTGRES
EDB Postgres Advanced Server
Last updated June 9, 2018, 10:48 a.m.
Edit
Revision List
Search
pavlo
MongoDB
MongoDB is a free and open-source cross-platform document-oriented program. It is a NoSQL database uses JSON-like documents with schemas-less. Ad hoc queries, indexing and real time aggregation provide powerful ways to access and analyze the data. It is a distributed database at its core that provides high availability, horizontal scaling, and geographic distribution.
History
The software company 10gen began developing MongoDB in 2007 as a component of a planned platform as a service product. In 2009, the company scraped its cloud platform and focus on maintaining MongoDB instead. It shifted to an open source development model, with the company offering commercial support and other service. In 2013, 10gen changed its name to MongoDB Inc.
Checkpoints
Consistent
When writing to disk, WiredTiger writes all the data in a snapshot to disk in a consistent way across all data files. The now-durable data act as a checkpoint in the data files. The checkpoint ensures that the data files are consistent up to and including the last checkpoint; i.e. checkpoints can act as recovery points. MongoDB configures WiredTiger to create checkpoints (i.e. write the snapshot data to disk) at intervals of 60 seconds or 2 gigabytes of journal data. During the write of a new checkpoint, the previous checkpoint is still valid. As such, even if MongoDB terminates or encounters an error while writing a new checkpoint, upon restart, MongoDB can recover from the last valid checkpoint.
Concurrency Control
Two-Phase Locking (Deadlock Prevention)
Optimistic Concurrency Control (OCC)
In MongoDB 3.0, concurrency control has been separated into two levels: top-level, which protects the database catalog, and storage engine-level, which allows each individual storage engine implementation to manage its own concurrency below the collection level. MongoDB uses reader-writer locks that allow concurrent readers shared access to a resource, but in MMAPv1, give exclusive access to a single write operation. WiredTiger uses OCC for concurrency control.
Data Model
Document / XML
MongoDB stores data in a binary representation called BSON (Binary JSON). The BSON encoding extends the popular JSON (JavaScript Object Notation) representation to include additional types such as int, long, date, floating point, and decimal128. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data and sub-documents. Documents that tend to share a similar structure are organized as collections. With the MongoDB document model, data is more localized, which significantly reduces the need to JOIN separate tables. The result is dramatically higher performance and scalability across commodity hardware as a single read to the database can retrieve the entire document containing all related data
mongoDB
Website
http://www.mongodb.com
Source Code
https://github.com/mongodb/mongo
Tech Docs
https://docs.mongodb.com/
Developer
MongoDB Inc.
Country of Origin
US
Start Year
2009
Project Type
Commercial, Open Source
Supported languages
ActionScript, C++, Clojure, D, Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, Matlab, Perl, PHP, Prolog, Python, R, Ruby, Scala, Smalltalk
Derived From
WiredTiger
Operating Systems
Linux, OS X, Solaris, Windows
Licenses
AGPL v3
Wikipedia
https://en.wikipedia.org/wiki/MongoDB
Revision #5 | Updated 07/04/2018 1:43 p.m.

# DBDB.IO

All the articles will be hosted on our new website.

I will post a sign-up sheet for you to pick what DBMS you want to write about.

- → If you choose a widely known DBMS, then the article will need to be comprehensive.
- → If you choose an obscure DBMS, then you will have do the best you can to find information.

# ☠ PLAGIARISM WARNING ☠

This article must be your own writing with your own images. You may **not** copy text/images directly from papers or other sources that you find on the web.

→ This includes both your submission for review and submission for your grade.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

# NEXT CLASS

Mid-Term