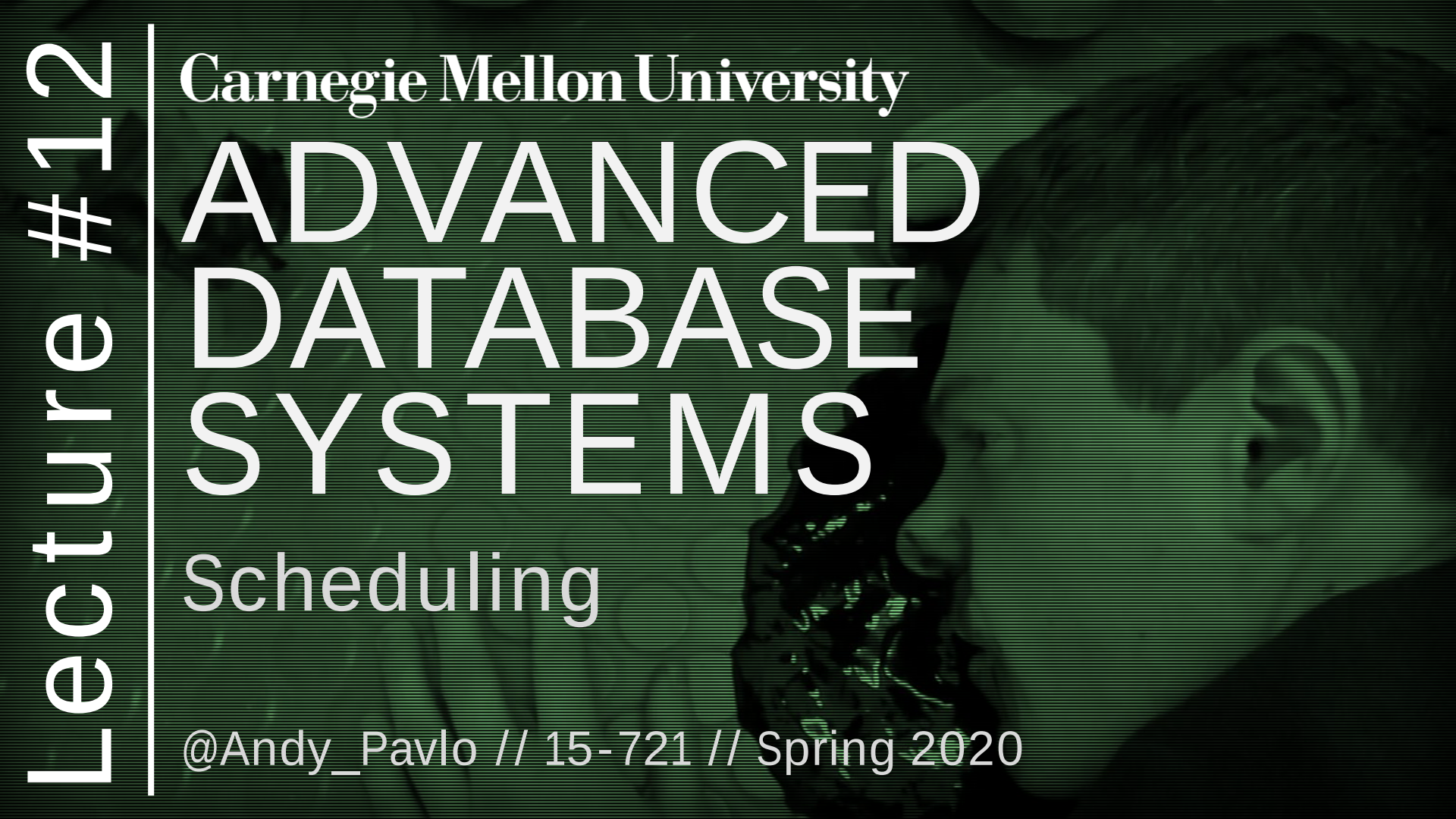

Lecture #12
Carnegie Mellon University
ADVANCED DATABASE SYSTEMS
Scheduling
@Andy_Pavlo // 15-721 // Spring 2020

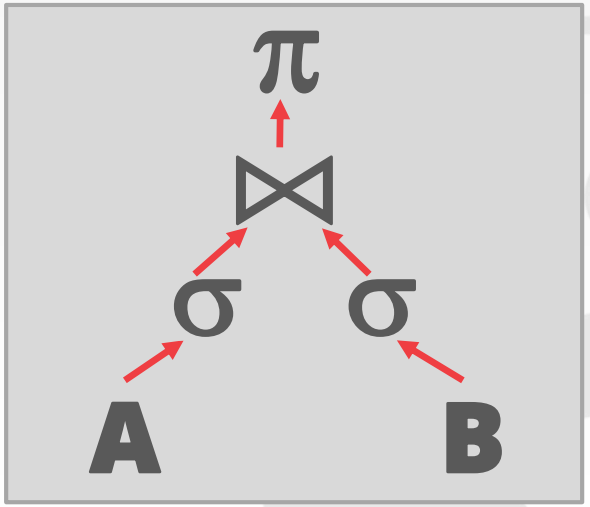
QUERY EXECUTION

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 (also sometimes referred to as a **pipeline**).

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



SCHEDULING

For each query plan, the DBMS must 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

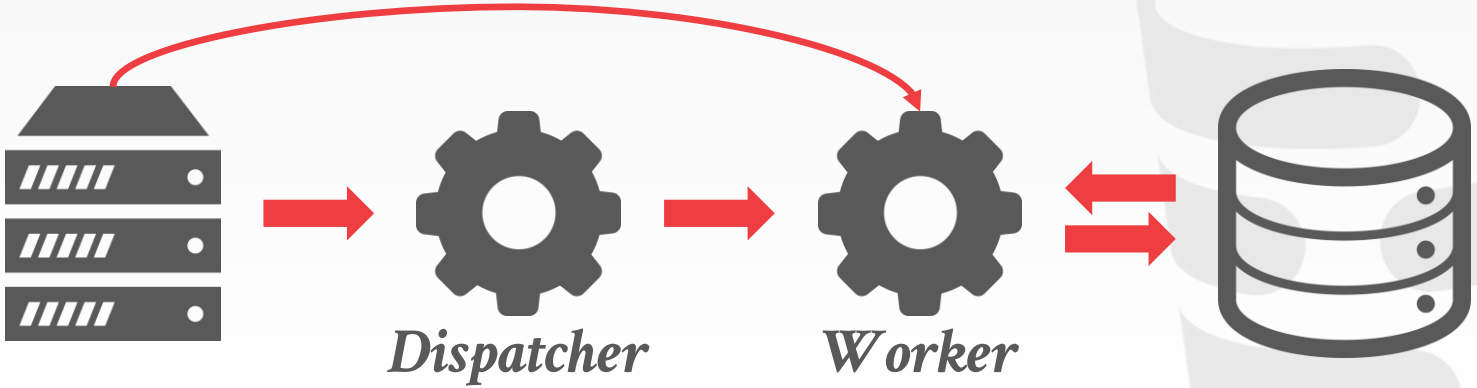
Approach #2: Process Pool

Approach #3: Thread per DBMS Worker

PROCESS PER WORKER

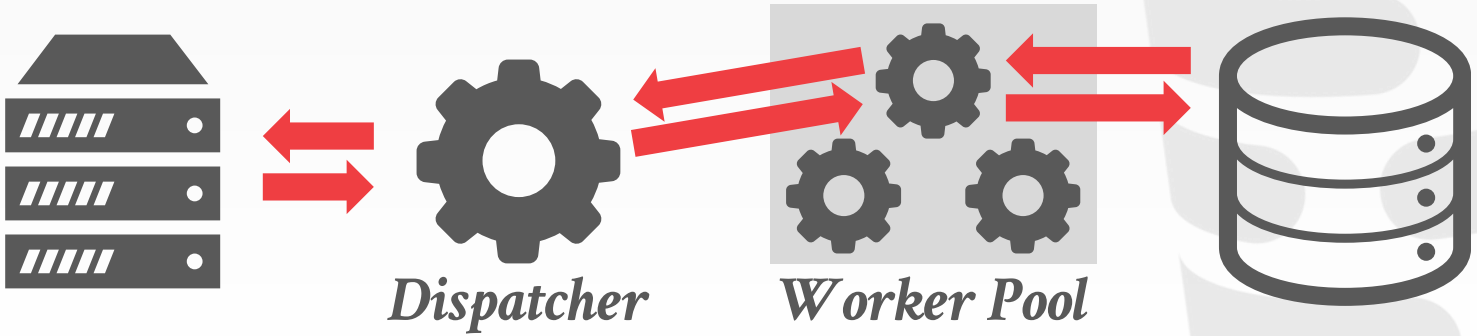
Each worker is a separate OS process.

→ 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



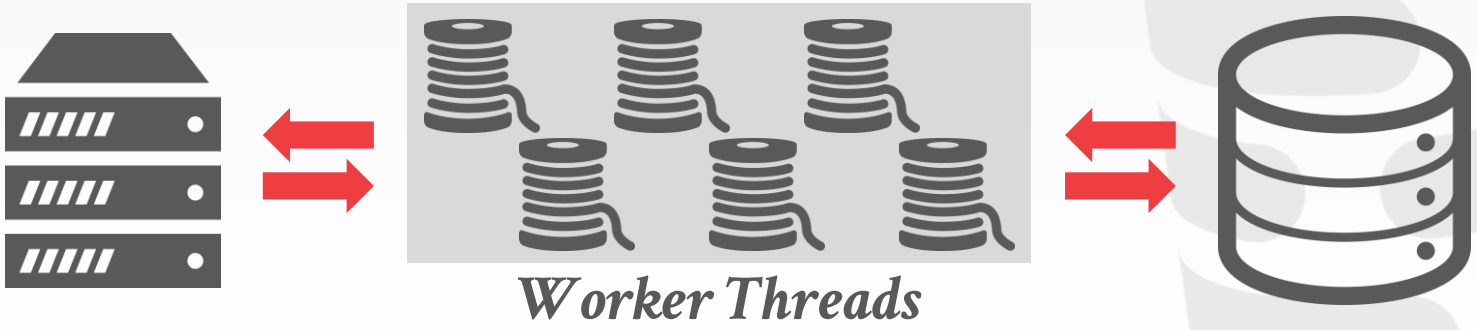
PROCESS POOL

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 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

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.

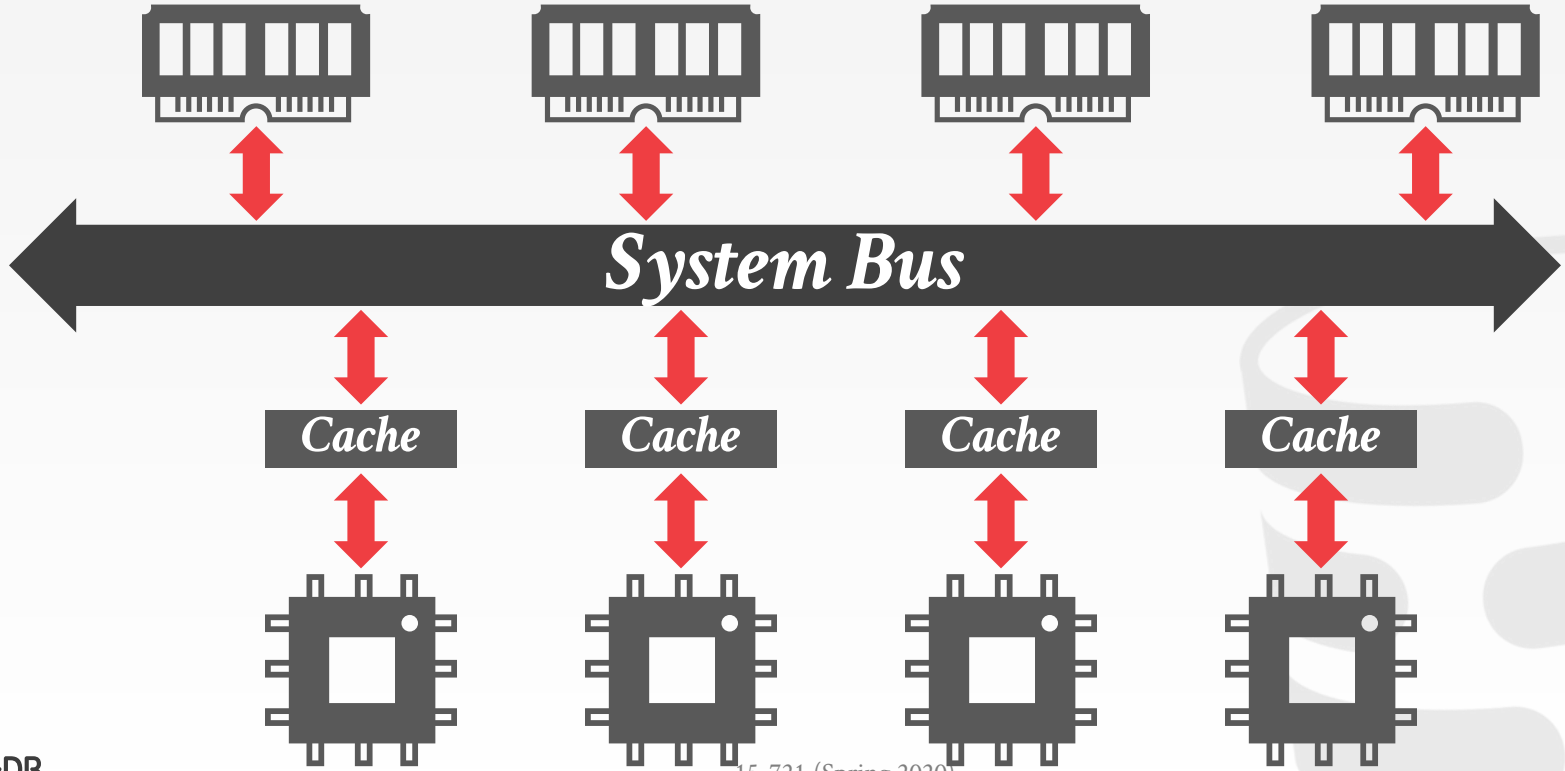
Andy is not aware of any new DBMS from last 10 years that doesn't use threads unless they are Postgres forks.

OBSERVATION

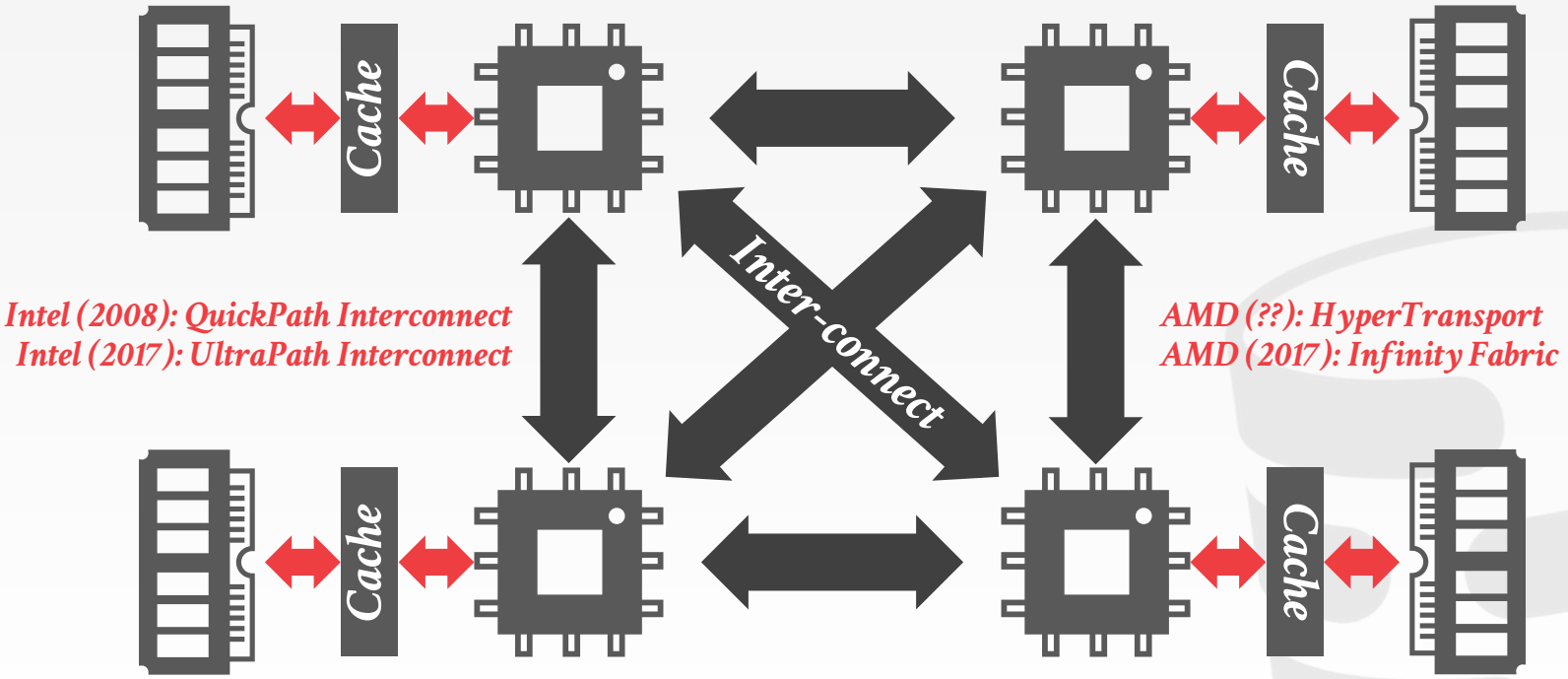
Regardless of what worker allocation or task assignment policy the DBMS uses, it's important that workers operate on local data.

The DBMS's scheduler must be aware of its hardware memory layout.
→ Uniform vs. Non-Uniform Memory Access

UNIFORM MEMORY ACCESS



NON-UNIFORM MEMORY ACCESS



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` and `numactl`

MEMORY ALLOCATION

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

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

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 on access.

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

MEMORY ALLOCATION LOCATION

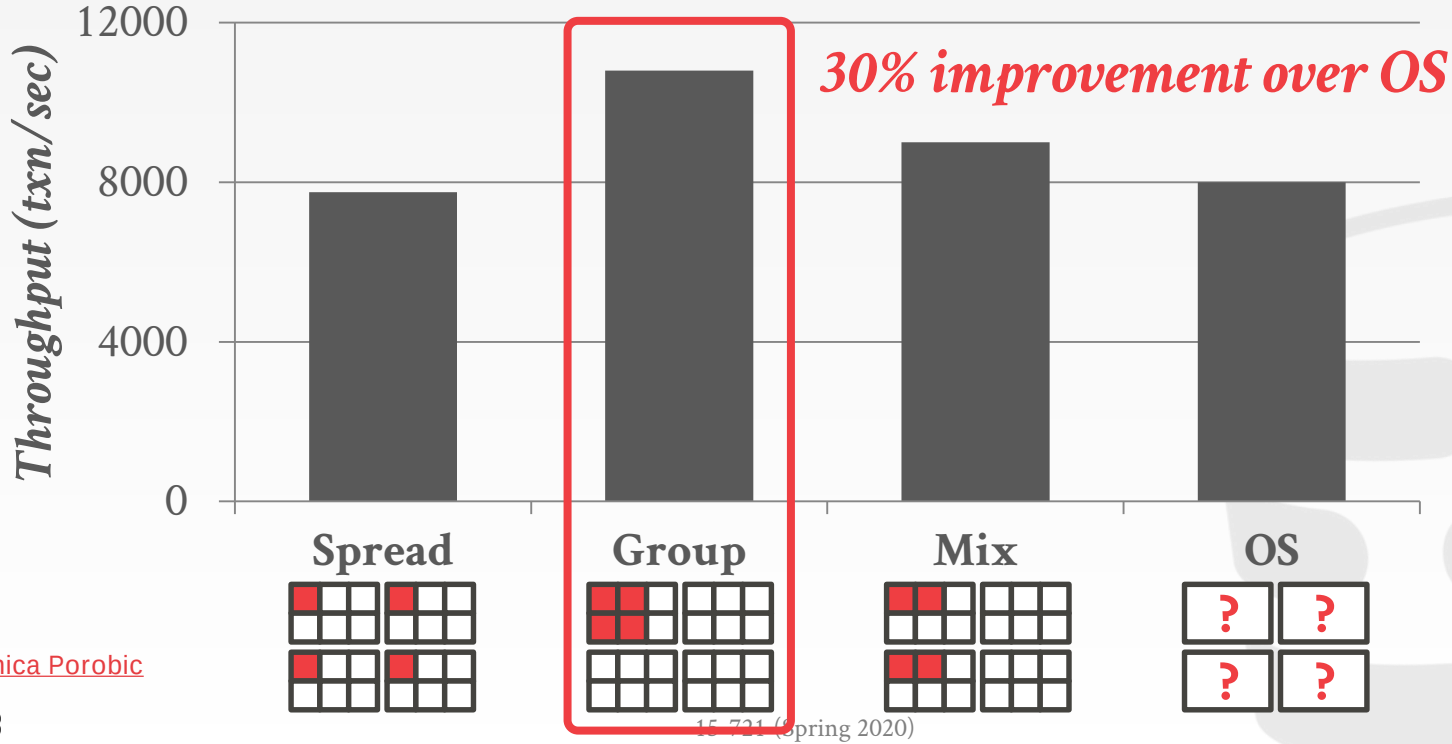
Approach #1: Interleaving
→ Distribute allocated memory uniformly across CPUs.

Approach #2: First-Touch
→ At the CPU of the thread that accessed the memory location that caused the page fault.

The OS can try to move memory to another NUMA region from observed access patterns.

DATA PLACEMENT – OLTP

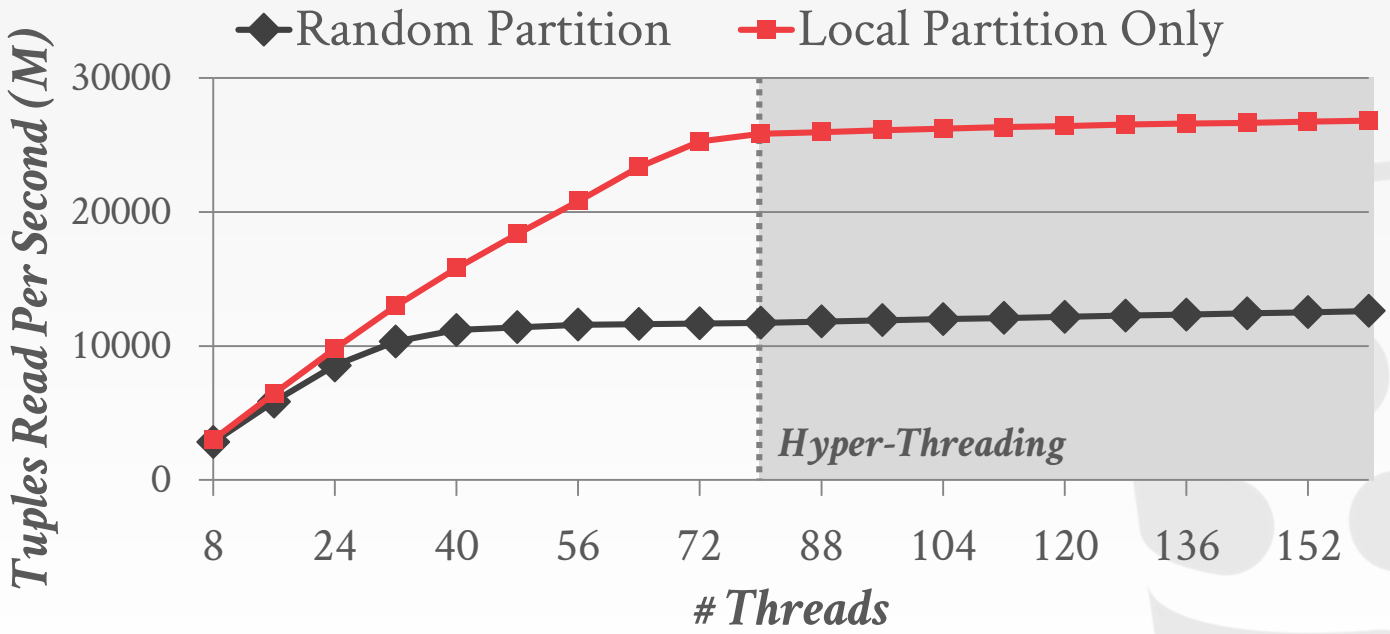
Workload: TPC-C Payment using 4 Workers
Processor: NUMA with 4 sockets (6 cores each)



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
→ 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.

→ Can still assign tasks to threads based on data location to maximize local data processing.

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.

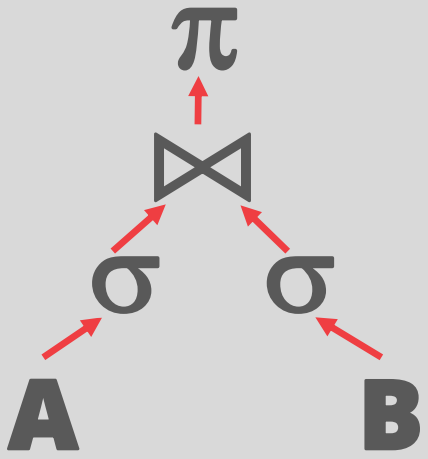
The workers perform cooperative scheduling for each query plan using a single task queue.

→ 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.

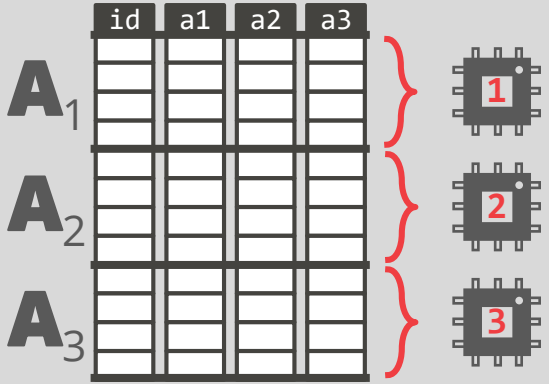
HYPER – DATA PARTITIONING

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



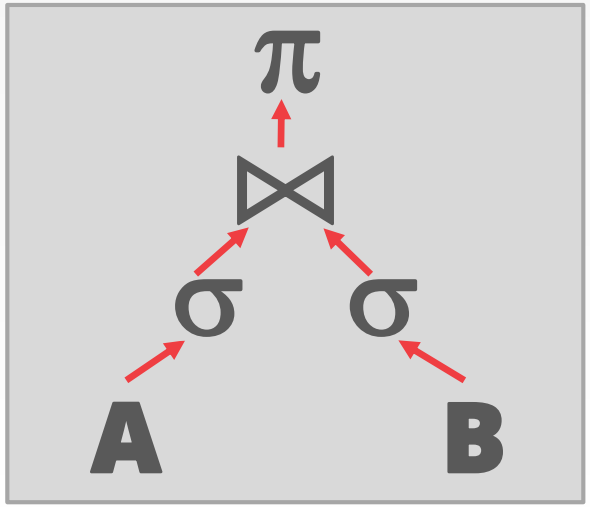
Data Table

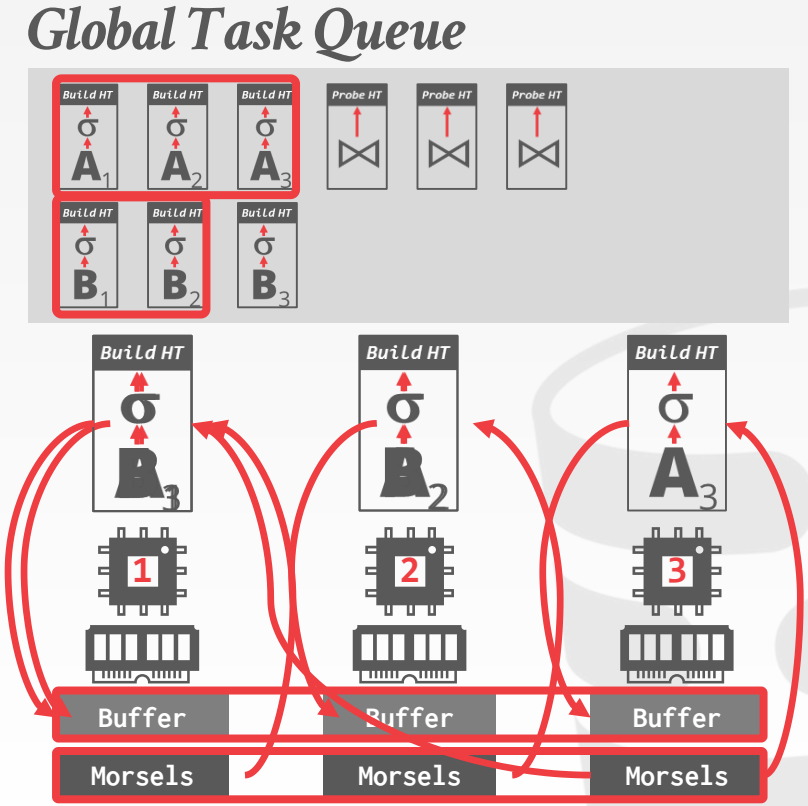
Morsels



HYPER – EXECUTION EXAMPLE

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





MORSEL-DRIVEN SCHEDULING

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

The DBMS uses a lock-free hash table to maintain the global work queues.

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 can 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

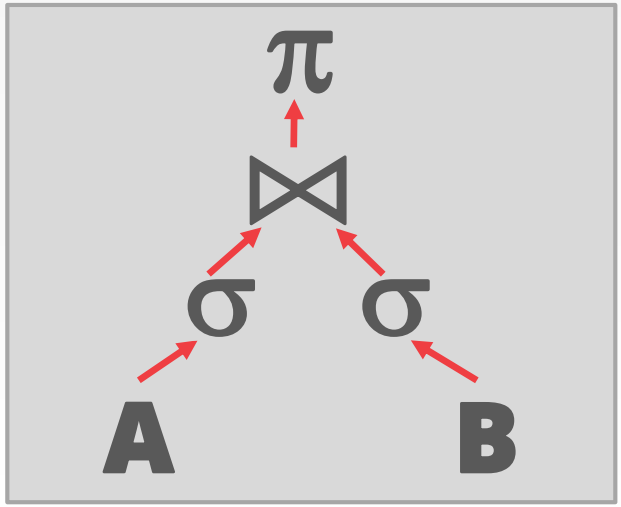
Dynamically adjust thread pinning based on whether a task is CPU or memory bound.

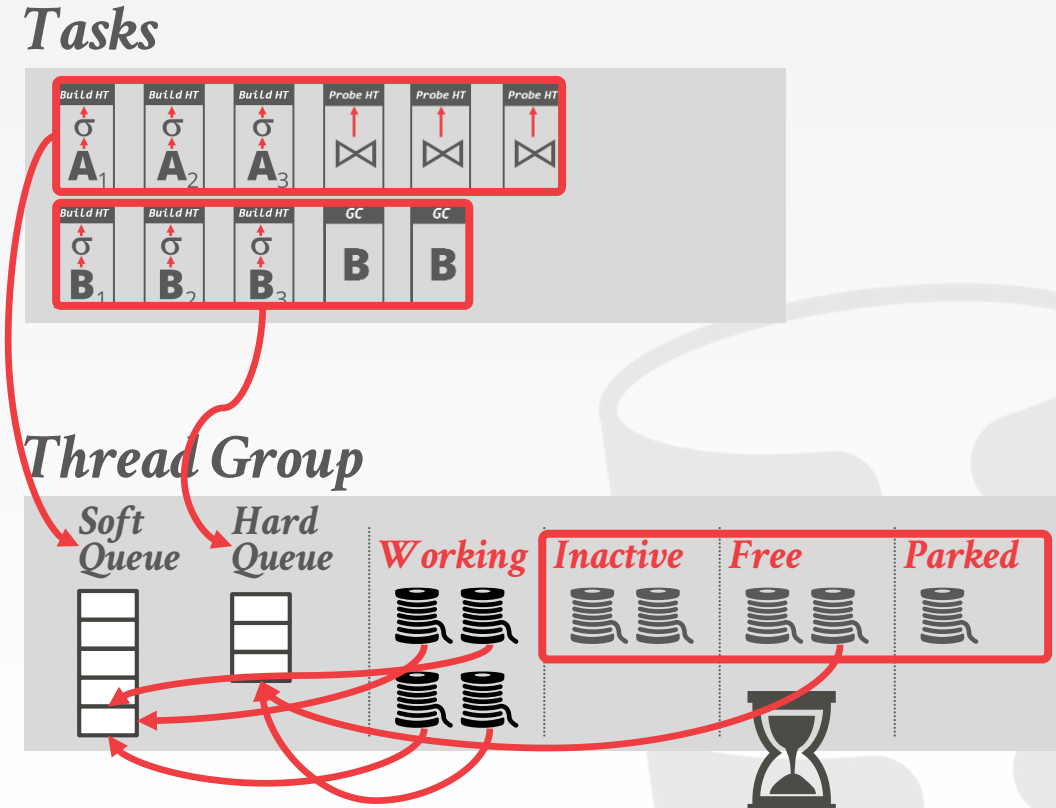
Found that work stealing was not as beneficial for systems with a larger number of sockets.

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

SAP 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
```





SQL

SQLOS is a us
that runs insi
provisioned h
→ Determines
threads.
→ Also manag
like logical

Non-preemp
instrumente

MICROSOFT SQL SERVER 2012 INTERNALS
PEARSON 2013

TC

Join Extra Crunch
Login

Search
Startups
Videos
Audio
Newsletters
Extra Crunch
Advertise
Events
—
More

Transportation
Apple
Tesla
Security

How Microsoft brought SQL Server to Linux

Frederic Lardinois @fredericl / 12:00 pm EDT • July 17, 2017

Comment

Back in 2016, when **Microsoft** announced that SQL Server would soon run on Linux, the news came as a major surprise to users and pundits alike. Over the course of the last year, Microsoft's support for Linux (and open source in general), has come into clearer focus and the company's mission now seems to be all about bringing its tools to wherever its users are.

The company today launched the first release candidate of SQL Server 2017, which will be the first version to run on Windows, Linux and in Docker containers. The Docker container alone has already seen more than 1 million pulls, so there can be no doubt that there is a lot of interest in this new version. And while there are plenty of new features and speed improvements in this new version, the fact that SQL Server 2017 supports Linux remains one of the most interesting aspects of this release.

Ahead of today's announcement, I talked to Rohan Kumar, the general manager of Microsoft's Database Systems group, to get a bit more info about the history of this project and how his team managed to bring an extremely complex piece of software like SQL Server to Linux. Kumar, who has been at Microsoft for more than 18 years, noted that his team noticed many enterprises were starting to use SQL Server for their mission-critical workloads. But at the same time, they were also working in mixed environments that included both Windows Server and Linux. For many of these businesses, not being able to run their database of choice on Linux became a friction point.

"Talking to enterprises, it became clear that doing this was necessary," Kumar said. "We were forcing customers to use Windows as their platform of choice." In another incarnation of Microsoft, that probably would've been seen as something positive, but the company's strategy today is quite different.

SQL SERVER – SQLOS

```
SELECT * FROM A WHERE A.val = ?
```

```
last = now()
for t in range(table.num_tuples):
  tuple = get_tuple(table, t)
  if eval(predicate, tuple, params):
    emit(tuple)
  if now() - last > 4ms:
     yield
     last = now()
```

SQLOS quantum is **4 ms** but the scheduler cannot enforce that.

DBMS developers must add explicit yield calls in various locations in the source code.

OBSERVATION

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

The OS cannot help us here because it does not know what threads are doing:
→ 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 must use hardware correctly.

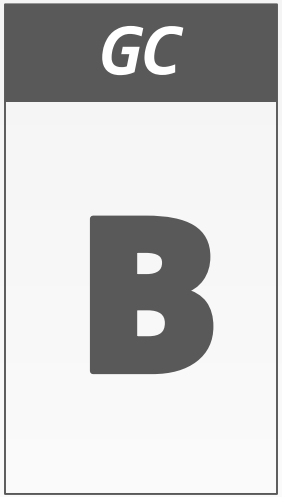
→ Data location is an important aspect of this.

→ 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.

NEXT CLASS

Query Execution


GC
B


Build HT
σ
B3