ADVANCED DATABASE SYSTEMS

Google BigQuery / Dremel

17

Andy Pavlo
CMU 15-721
Spring 2024

Carnegie Mellon University

OBSERVATION

In the 2000s, Google had the most influence in industry trends in development of database systems.

Whenever Google released a research paper describing an internal system, other tech companies would write open-source clones.

→ People assumed that whatever Google did was the way it should be done because they are successful.

DATA SYSTEMS AT GOOGLE

NoSQL
- MapReduce (2004)
- BigTable (2005)
- Chubby (2006)
- LevelDB (2011)

SQL
- Megastore (2010)
- Vitess (2010)
- Dremel (2011)
- Spanner (2011)
- F1 (2013)
- Mesa (2014)
- Napa (2021)

DATA SYSTEMS AT GOOGLE

NoSQL
- MapReduce (2
- BigTable (200
- Chubby (2006
- LevelDB (201

SQL
- Megastore (20
- Vitess (2010)
- Dremel (2011)
- Spanner (2011)
- F1 (2013)
- Mesa (2014)
- Napa (2021)

> The conventional wisdom at Google was "SQL doesn't scale", and with a few exceptions, Google had moved away from SQL almost completely. In solving for scalability, we had given up ease of use and ability to iterate quickly.
>
> Dremel was one of the first systems to reintroduce SQL for Big Data analysis. Dremel made it possible for the first time to write a simple SQL query to analyze web-scale datasets. Analysis jobs

DATA SYSTEMS AT GOOGLE

NoSQL

- MapReduce (2004) → *Hadoop, Spark*
- BigTable (2005) → *HBase, Accumulo, Hypertable*
- Chubby (2006) → *Zookeeper, etcd*
- LevelDB (2011) → *RocksDB*

SQL

- Megastore (2010)
- Vitess (2010) → *Vitess, Planetscale*
- Dremel (2011) → *Drill, PrestoDB, Impala, Dremio*
- Spanner (2011) → *CockroachDB*, TiDB**
- F1 (2013)
- Mesa (2014)
- Napa (2021)

(*) Only for marketing...

GOOGLE DREMEL

Originally developed in 2006 as a side-project for analyzing data artifacts generated from other tools.

→ The "interactive" goal means that they want to support ad hoc queries on **in-situ** data files.

→ Did not support joins in the first version.

Rewritten in the late 2010s to shared-disk architecture built on top of GFS.

Released as public commercial product (BigQuery) in 2012.

DREMEL: A DECADE OF INTERACTIVE SQL ANALYSIS AT WEB SCALE
VLDB 2020

GOOGLE DREMEL

Originally deve
analyzing data
→ The "interactive" goal means that they want to support ad-hoc queries on **in-situ** data fil
→ Did not support joins in the fi

Rewritten in the late 2010s
architecture built on top of

Released as public commer
in 2012.

[1]Dremel is a brand of power tools that primarily rely on their speed as opposed to torque. We use this name for an internal project only.

DREMEL: A DECADE OF INTERACTIVE SQL ANALYSIS AT WEB SCALE
VLDB 2020

IN-SITU DATA PROCESSING

Execute queries on data files residing in shared storage (e.g., object store) in their original format without first ingesting them into the DBMS (i.e., managed storage).

→ This is what people usually mean when they say **data lake**.
→ A **data lakehouse** is the DBMS that sits above all this.

The goal is to reduce the amount of prep time needed to start analyzing data.

→ Users are willing to sacrifice query performance to avoid having to re-encode / load data files.

GOOGLE DREMEL

Shared-Disk / Disaggregated Storage

Vectorized Query Processing

Shuffle-based Distributed Query Execution

Columnar Storage
→ Zone Maps / Filters
→ Dictionary + RLE Compression
→ Only Allows "Search" Inverted Indexes

Hash Joins Only

Heuristic Optimizer + Adaptive Optimizations

DREMEL: QUERY EXECUTION

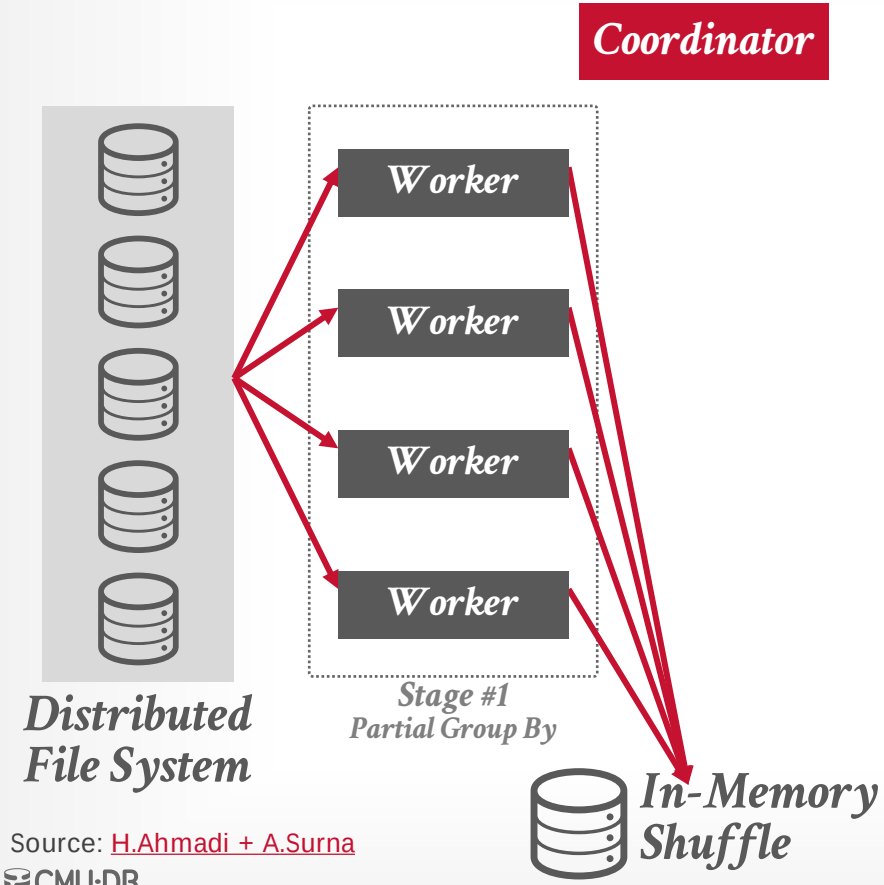
DBMS converts a logical plan into **stages** (pipelines) that contain multiple parallel **tasks**.

→ Each task must be deterministic and idempotent to support restarts.

Root node (Coordinator) retrieves all the meta-data for target files in a batch and then embeds it in the query plan.

Each worker node has its own local memory and can spill to local disk if needed.

DREMEL: QUERY EXECUTION

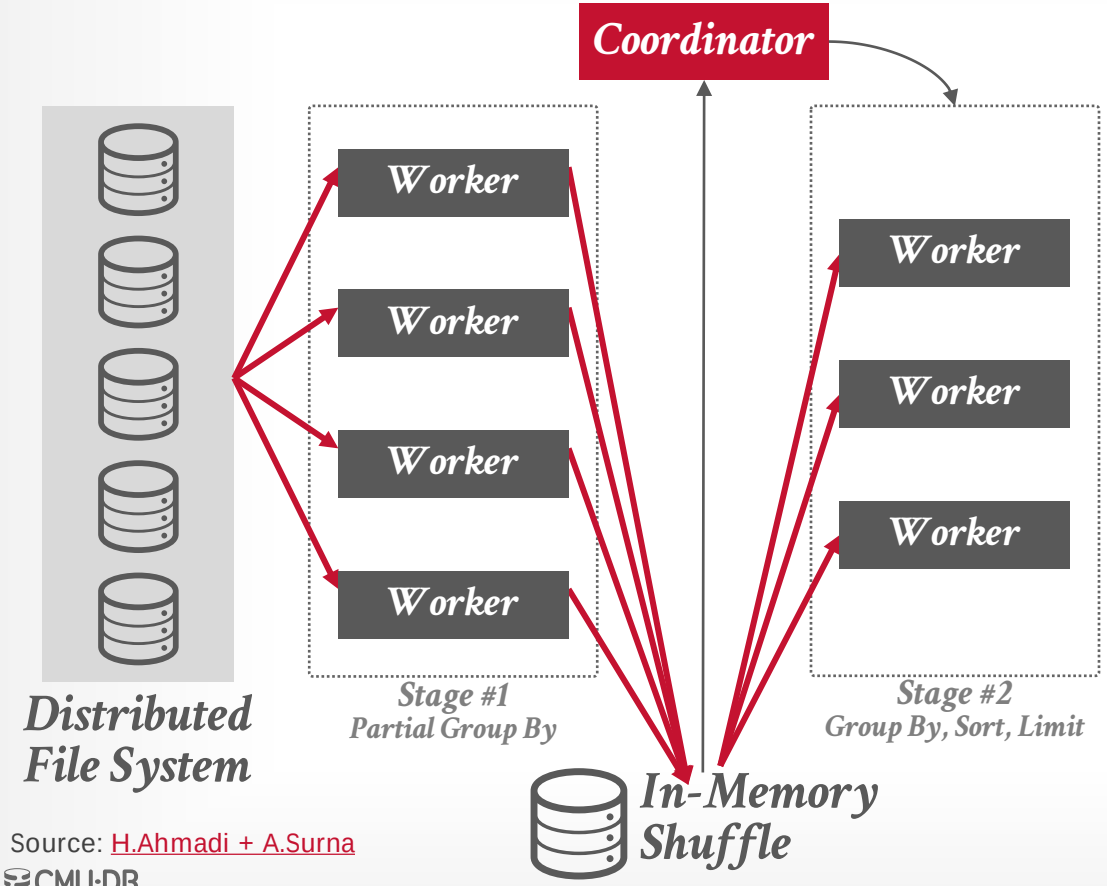



```
SELECT language, MAX(views)
  FROM wikipedia
 WHERE title LIKE "%Pavlo%"
 GROUP BY 1 ORDER BY 2 DESC
 LIMIT 100
```

Source: H.Ahmadi + A.Surna

DREMEL: QUERY EXECUTION

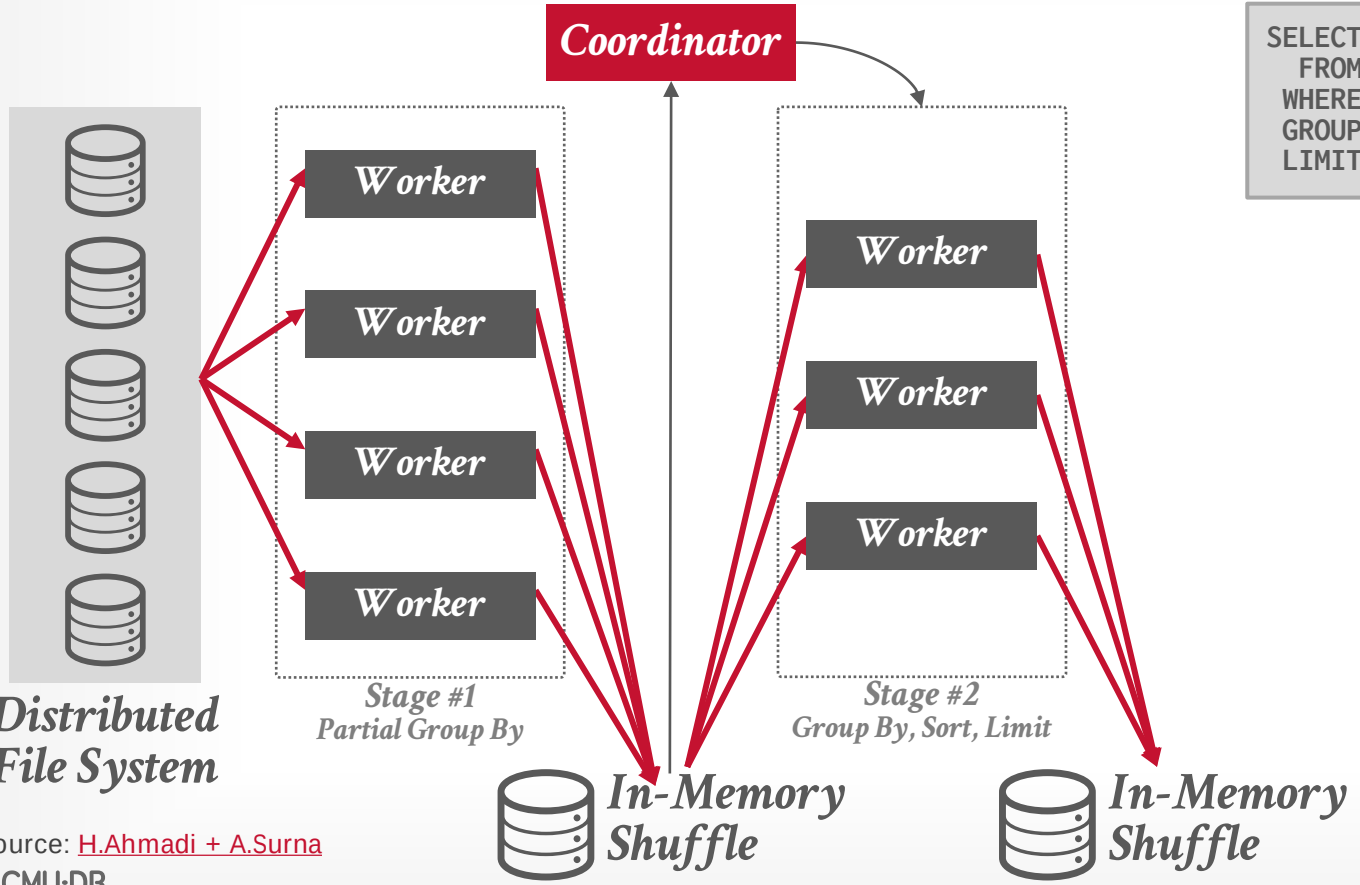
```
SELECT language, MAX(views)
  FROM wikipedia
 WHERE title LIKE "%Pavlo%"
 GROUP BY 1 ORDER BY 2 DESC
 LIMIT 100
```



Source: H.Ahmadi + A.Surna

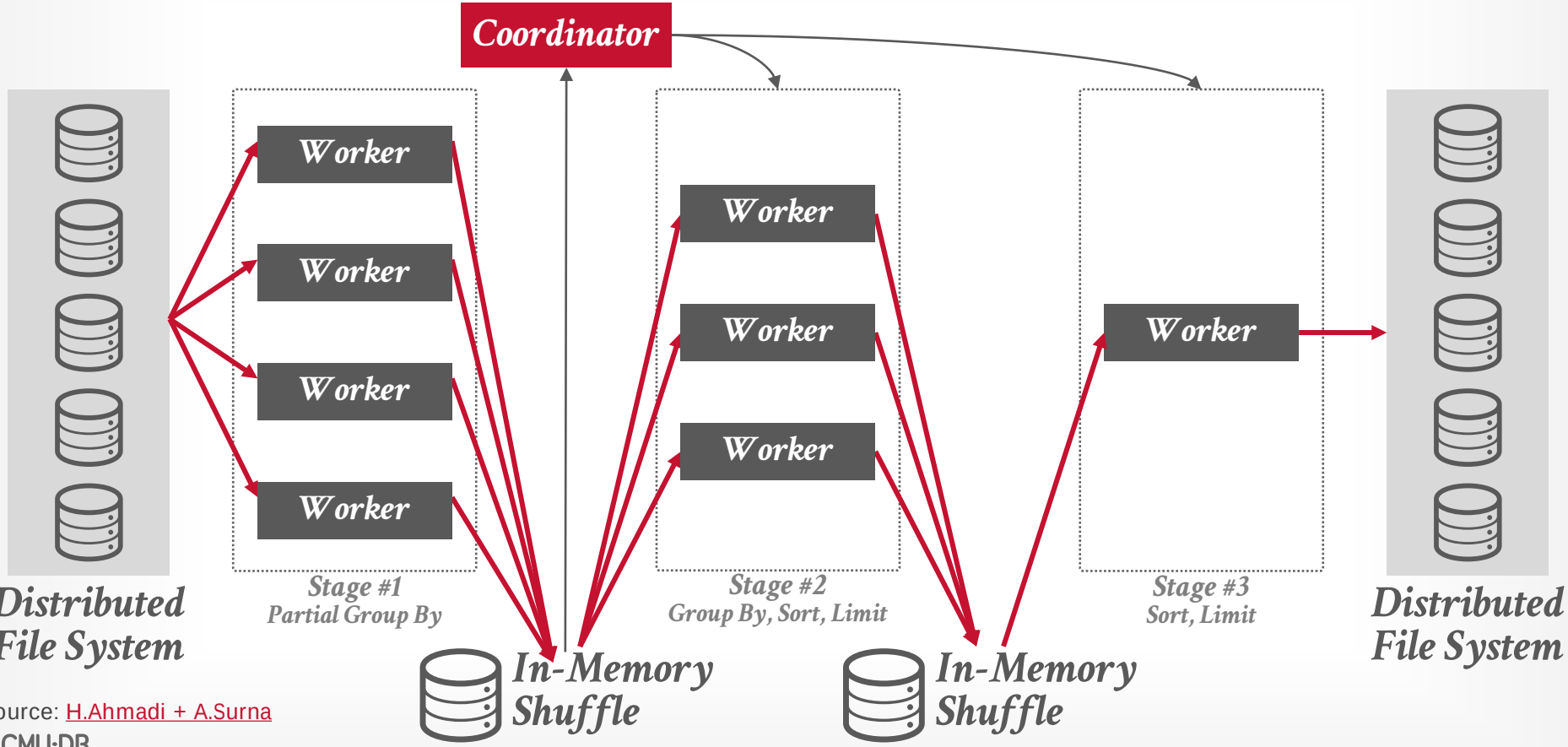
DREMEL: QUERY EXECUTION

```
SELECT language, MAX(views)
  FROM wikipedia
 WHERE title LIKE "%Pavlo%"
 GROUP BY 1 ORDER BY 2 DESC
 LIMIT 100
```



Source: H.Ahmadi + A.Surna

DREMEL: QUERY EXECUTION



Source: H.Ahmadi + A.Surna

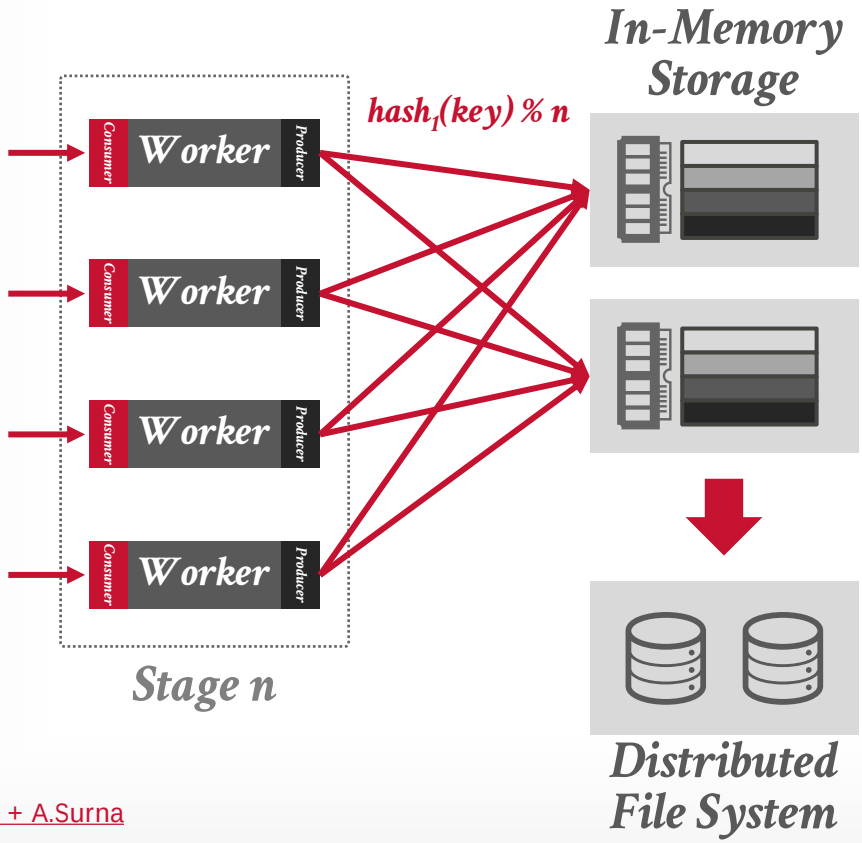
DREMEL: IN-MEMORY SHUFFLE

Producer/consumer model for transmitting intermediate results from each stage to the next using dedicated nodes.

→ Workers send output to shuffle nodes.
→ Shuffle nodes store data in memory in hashed partitions.
→ Workers at the next stage retrieve their inputs from the shuffle nodes.

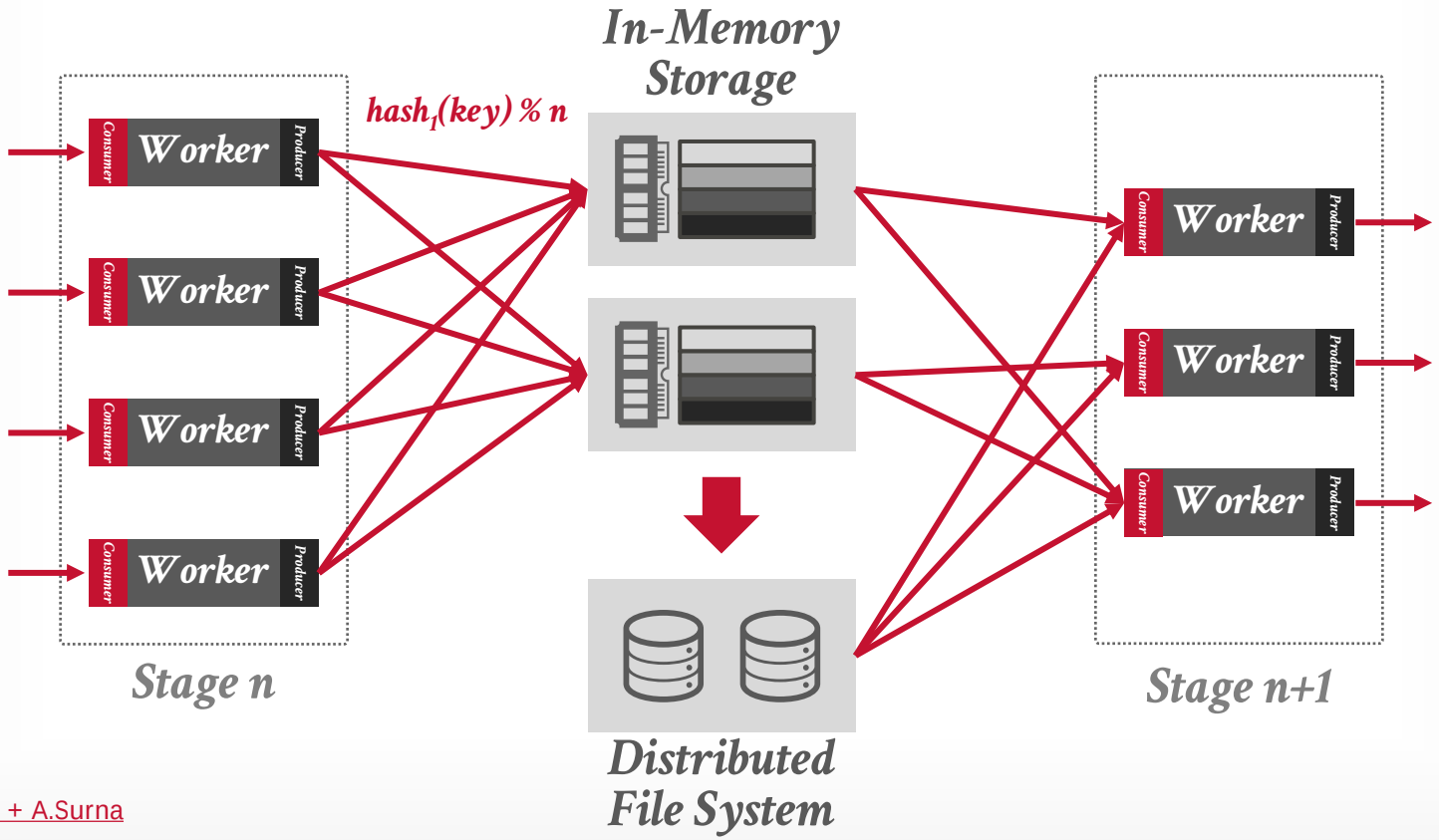
Shuffle nodes store this data in memory and only spill to disk storage if necessary.

DREMEL: IN-MEMORY SHUFFLE



Source: H.Ahmadi + A.Surna

DREMEL: IN-MEMORY SHUFFLE



Source: H.Ahmadi + A.Surna

DREMEL: IN-MEMORY SHUFFLE

The shuffle phases represent checkpoints in a query's lifecycle where that the coordinator makes sure that all tasks are completed.

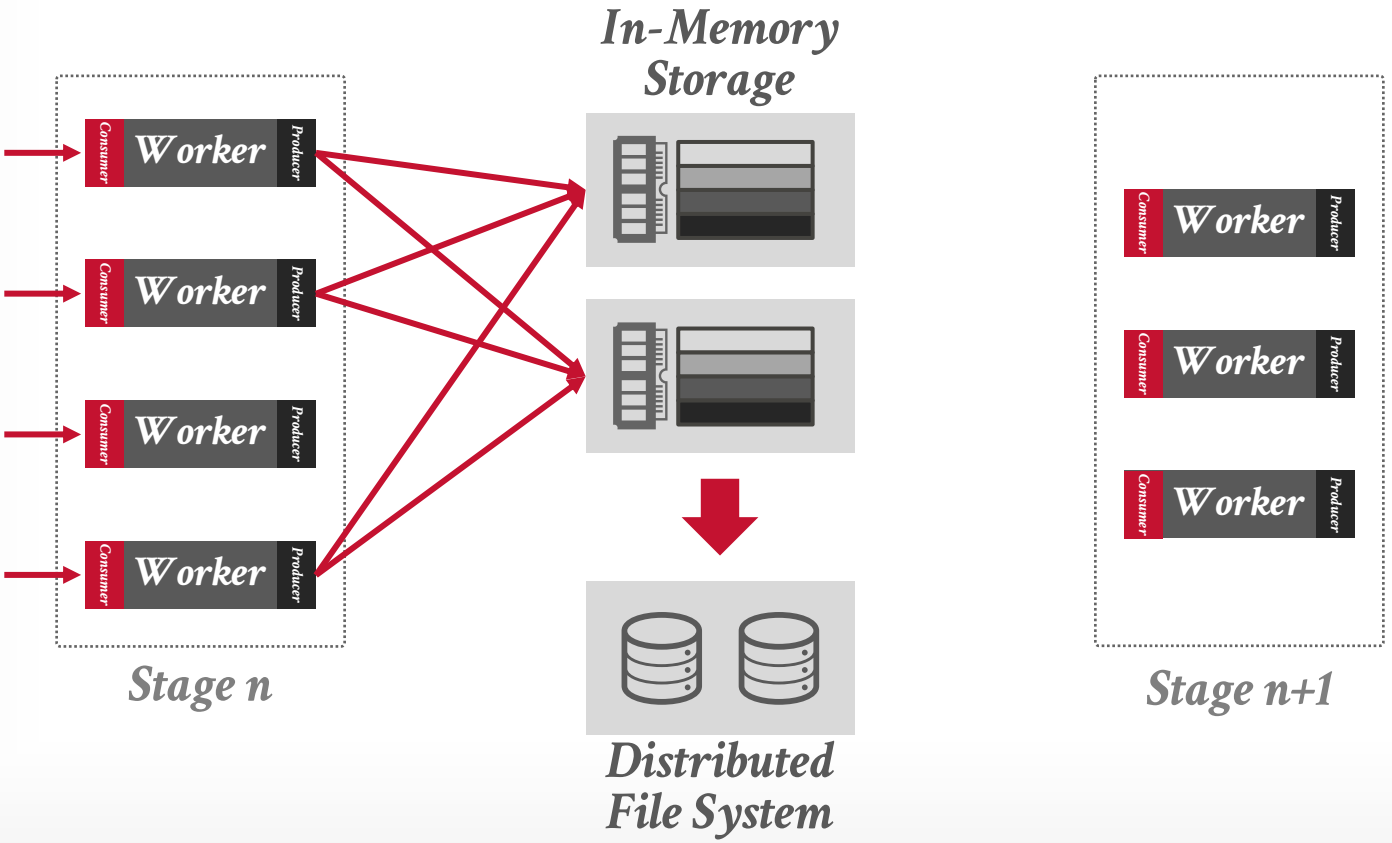
Fault Tolerance / Straggler Avoidance:

→ If a worker does not produce a task's results within a deadline, the coordinator speculatively executes a redundant task.

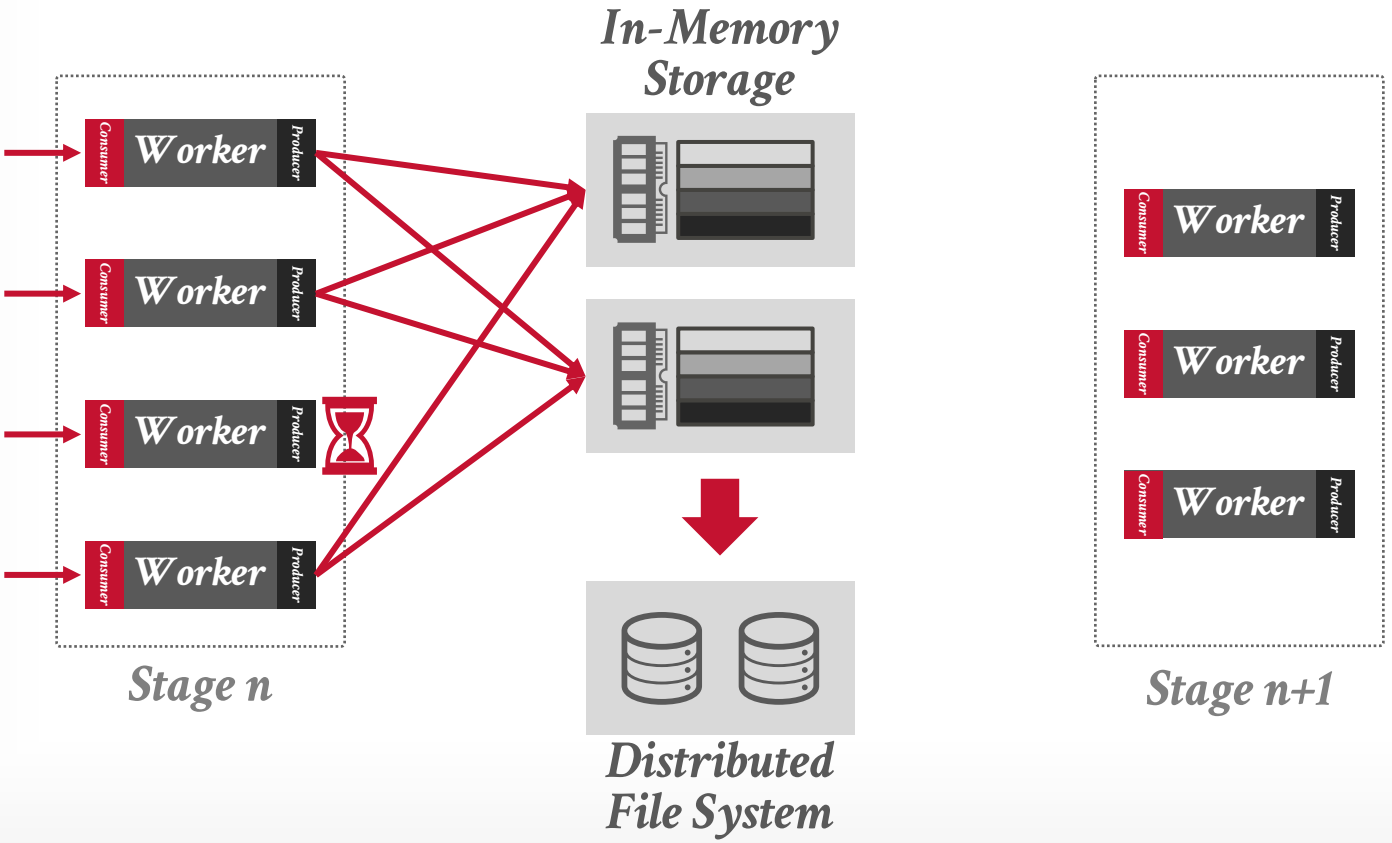
Dynamic Resource Allocation:

→ Scale up / down the number of workers for the next stage depending size of a stage's output.

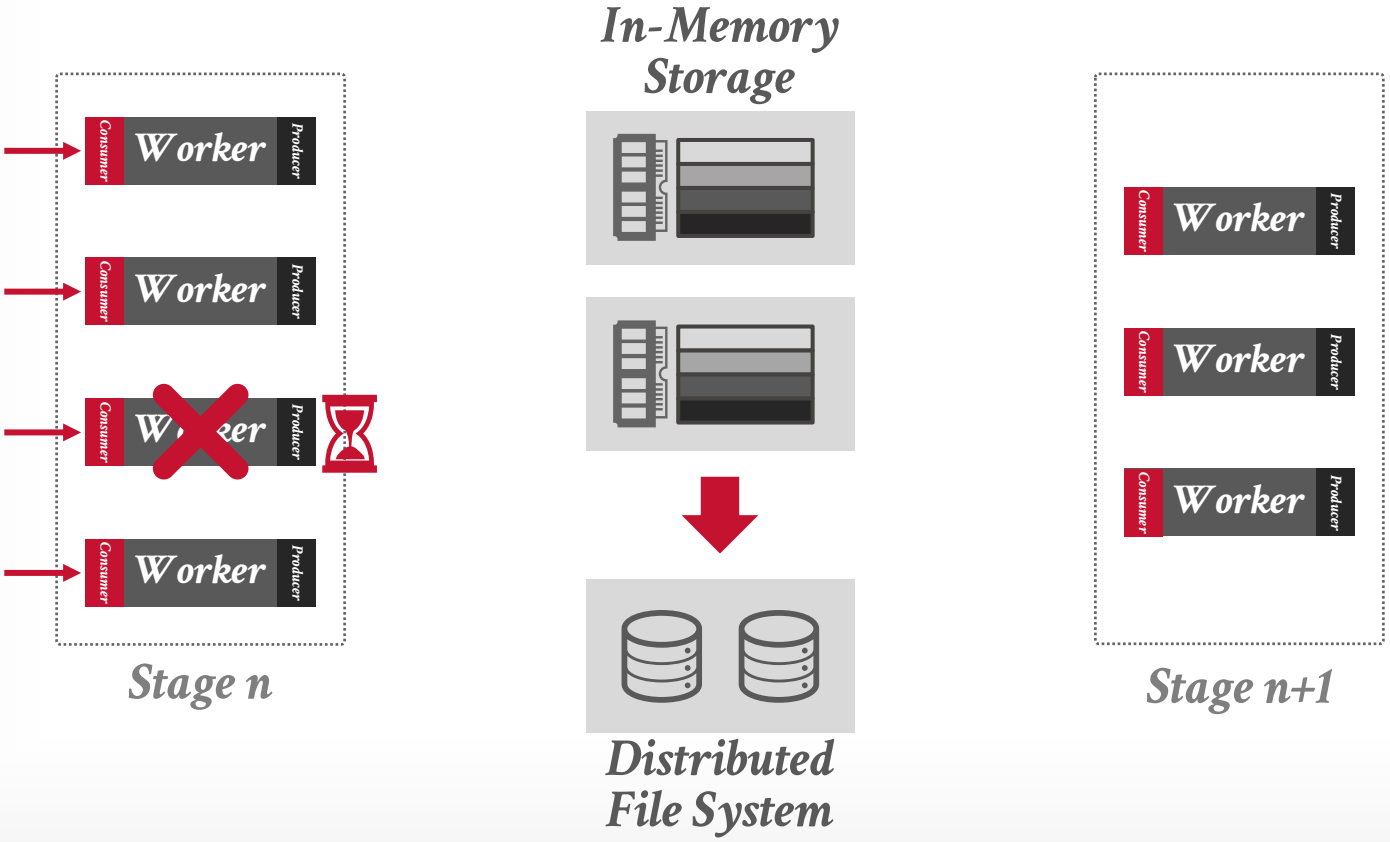
DREMEL: IN-MEMORY SHUFFLE



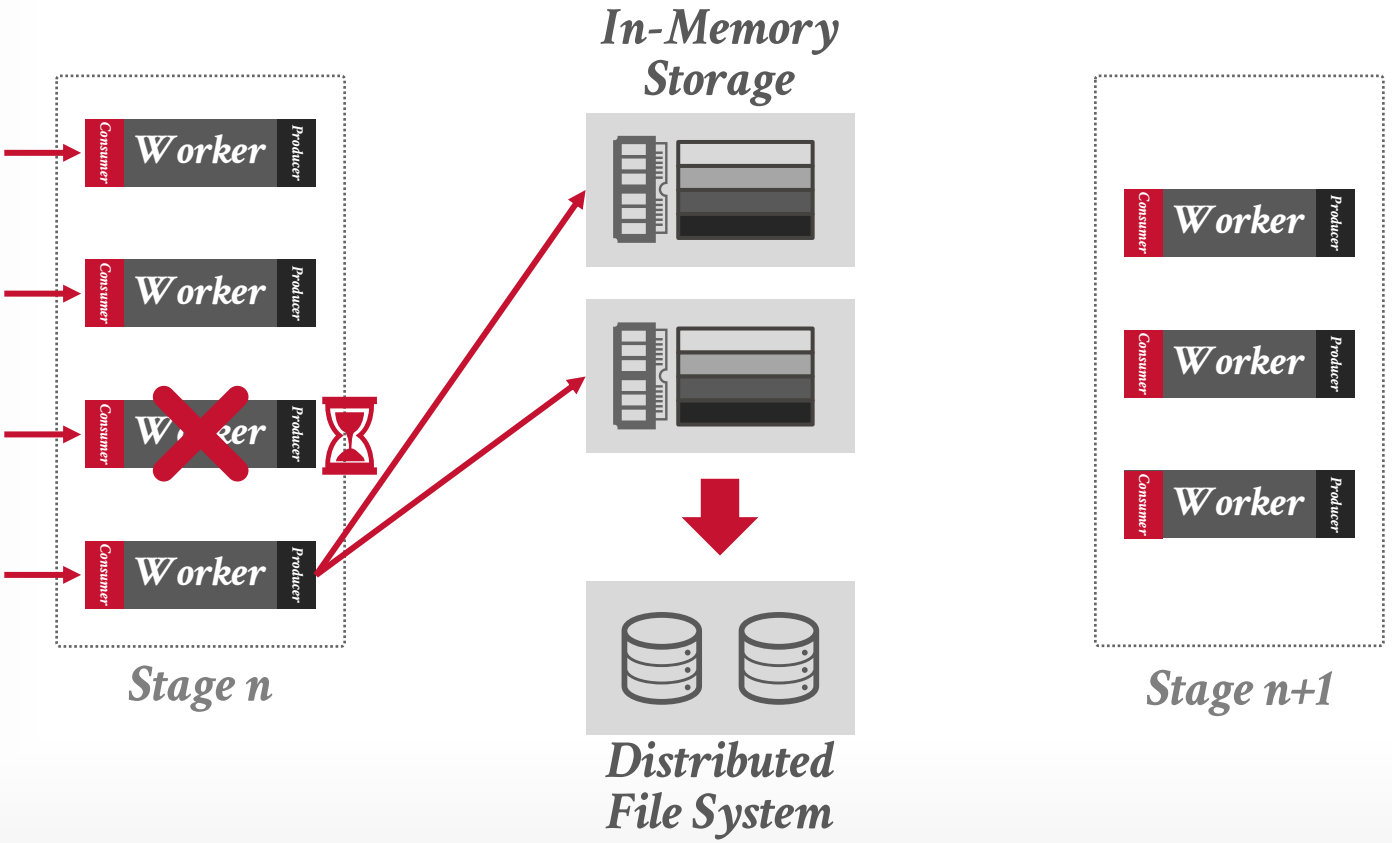
DREMEL: IN-MEMORY SHUFFLE



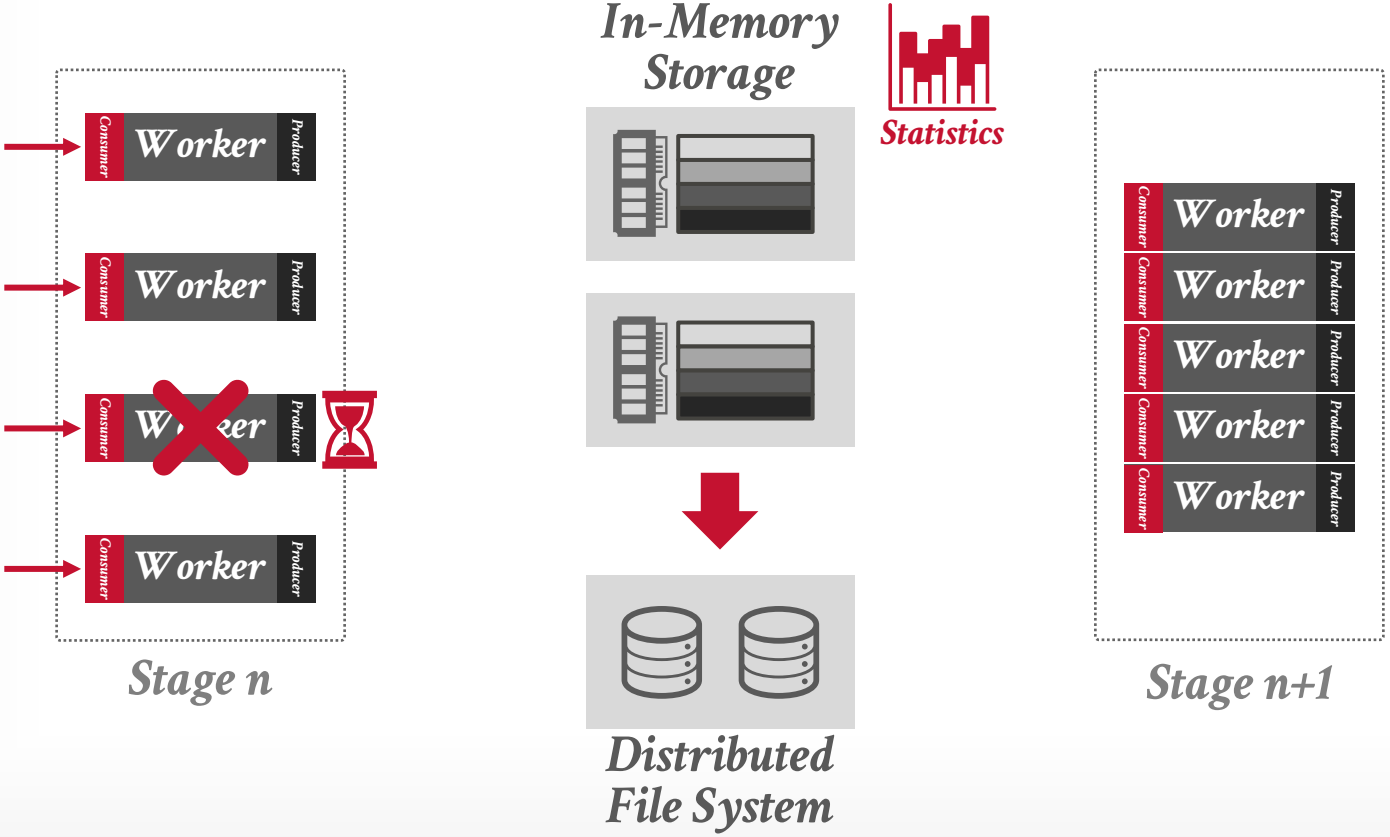
DREMEL: IN-MEMORY SHUFFLE



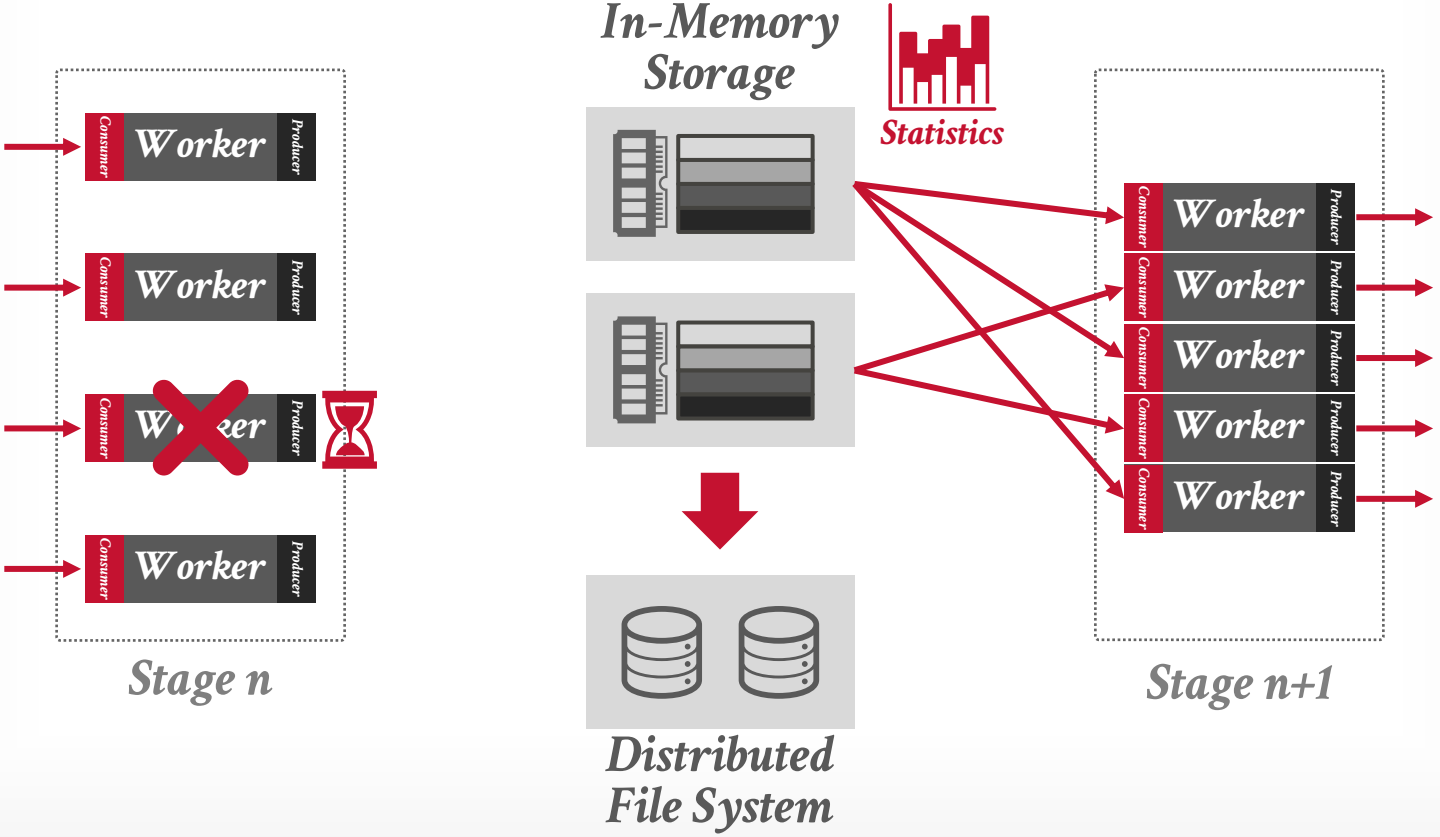
DREMEL: IN-MEMORY SHUFFLE



DREMEL: IN-MEMORY SHUFFLE



DREMEL: IN-MEMORY SHUFFLE



OBSERVATION

We discussed how query optimizers rely on cost models derived from statistics extracted from data.

But how can the DBMS optimize a query if there are no statistics?

→ Data files the DBMS has never seen before.
→ Query APIs from other DBMSs (connectors).

DREMEL: QUERY OPTIMIZATION

Dremel's optimizer uses a stratified approach with rule-based + cost-based optimizer passes to generate a preliminary physical plan to start execution.

→ Rules for predicate pushdown, star schema constraint propagation, primary/foreign key hints, join ordering.

→ Cost-based optimizations only on data that the DBMS has statistics available (e.g., materialized views).

To avoid the problems with bad cost model estimates, Dremel uses adaptive query optimization…

Source: H.Ahmadi + A.Surna

DREMEL: ADAPTIVE QUERY OPTIMIZATION

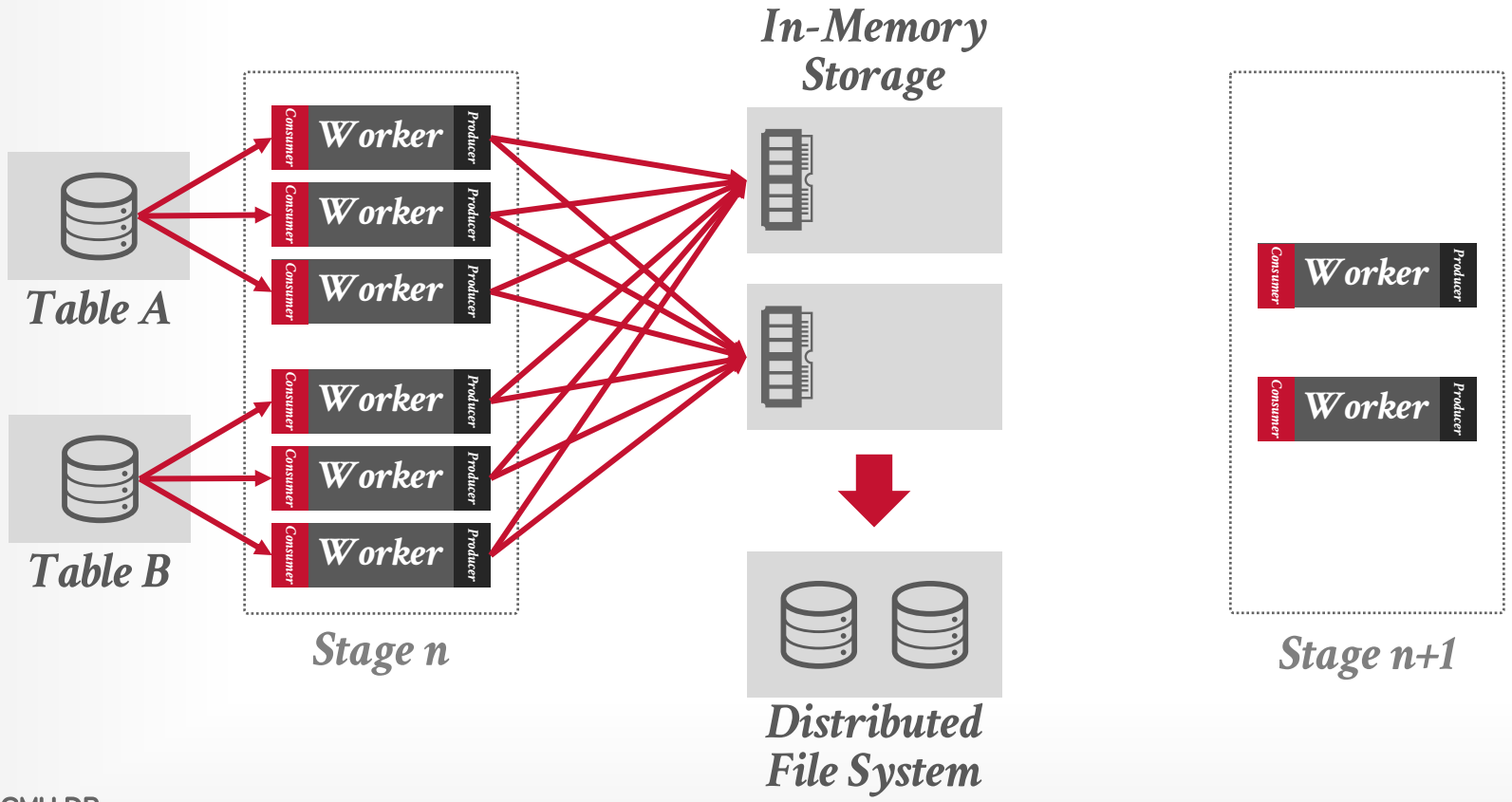
Dremel changes the query plan before a stages starts based on observations from the preceding stage.

→ Avoids the problem of optimizer making decisions with inaccurate (or non-existing) data statistics.

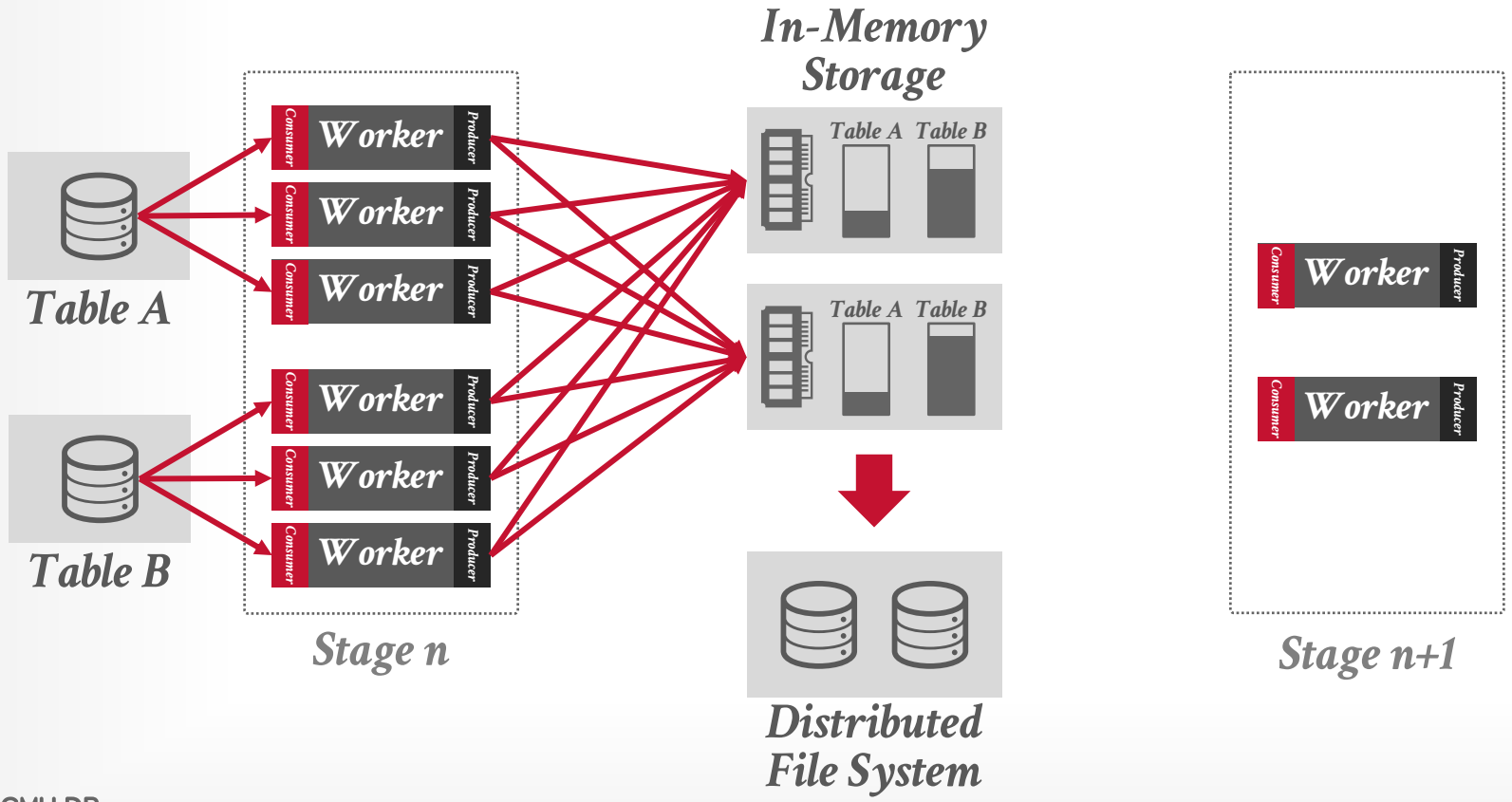
Optimization Examples:

→ Change the # of workers in a stage.
→ Switch between shuffle vs. broadcast join.
→ Change the physical operator implementation.
→ Dynamic repartitioning.

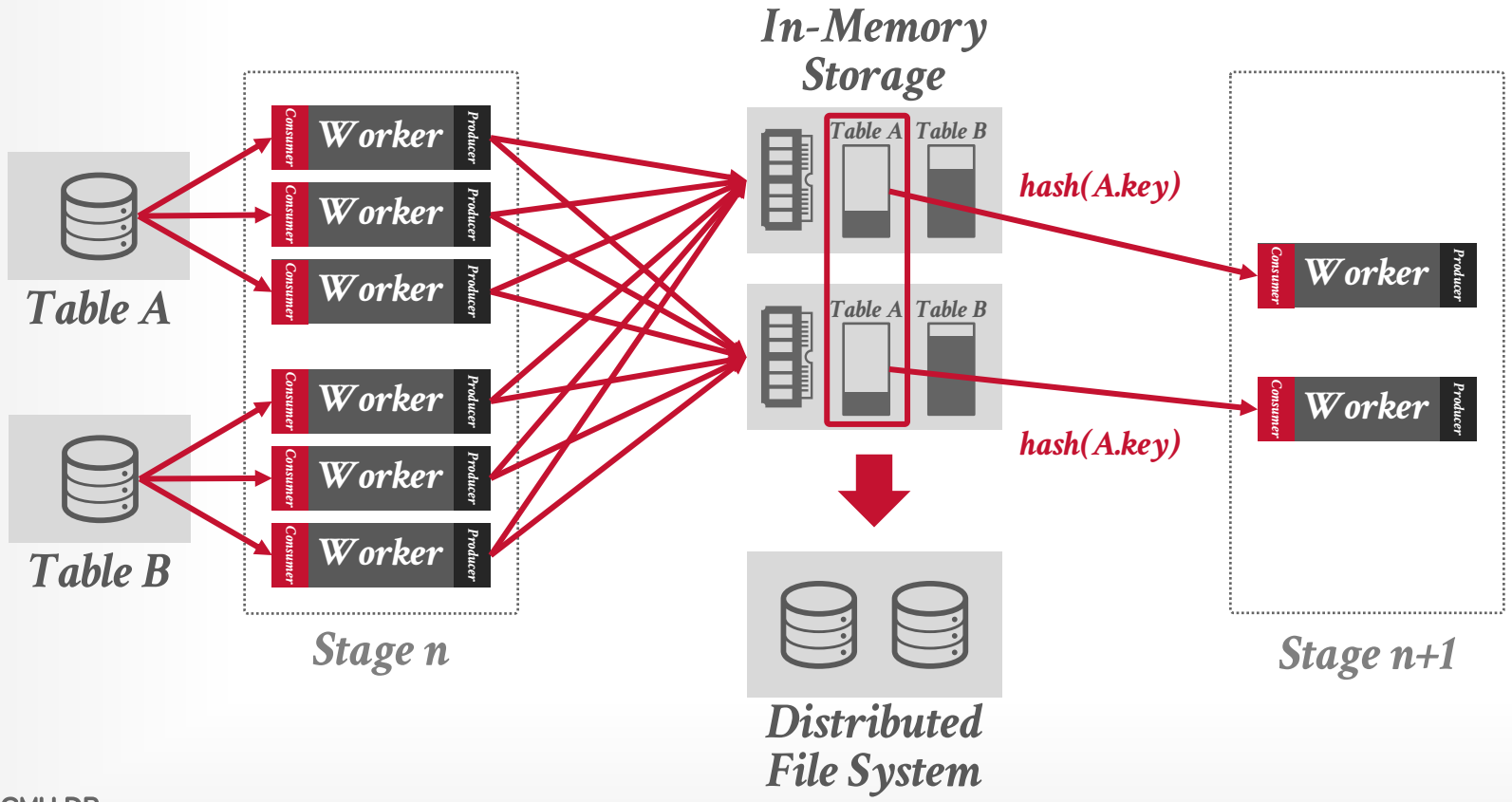
DREMEL: ADAPTIVE JOIN



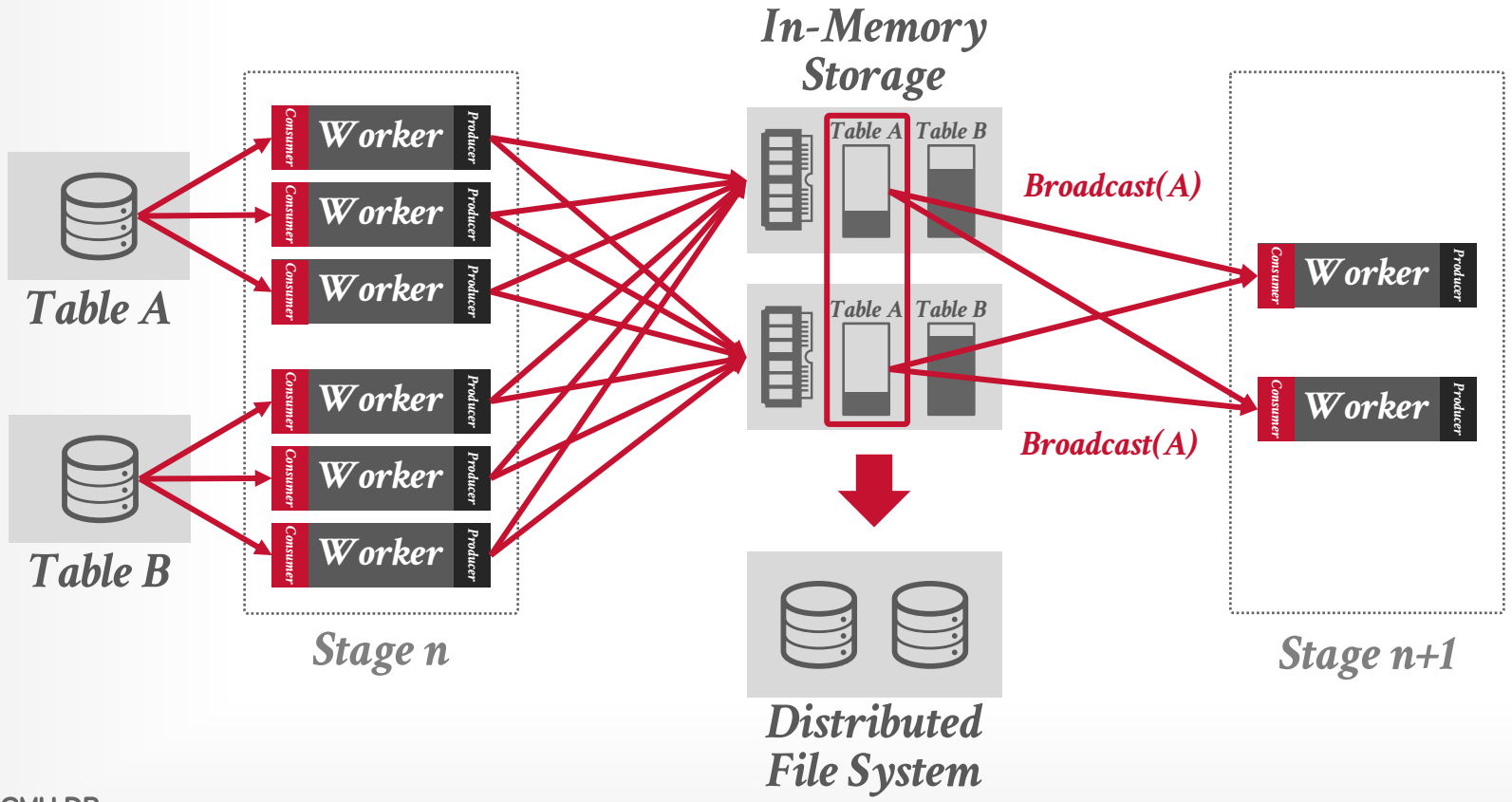
DREMEL: ADAPTIVE JOIN



DREMEL: ADAPTIVE JOIN



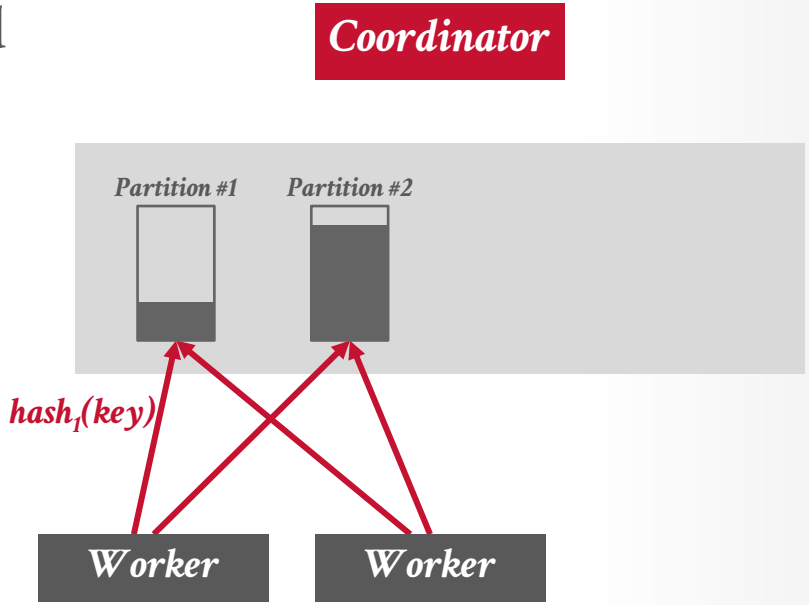
DREMEL: ADAPTIVE JOIN



DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

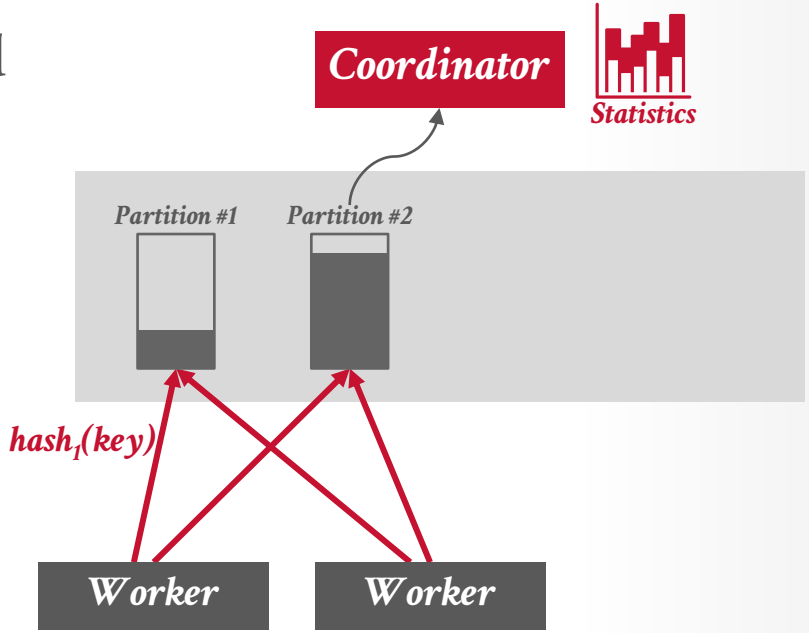


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

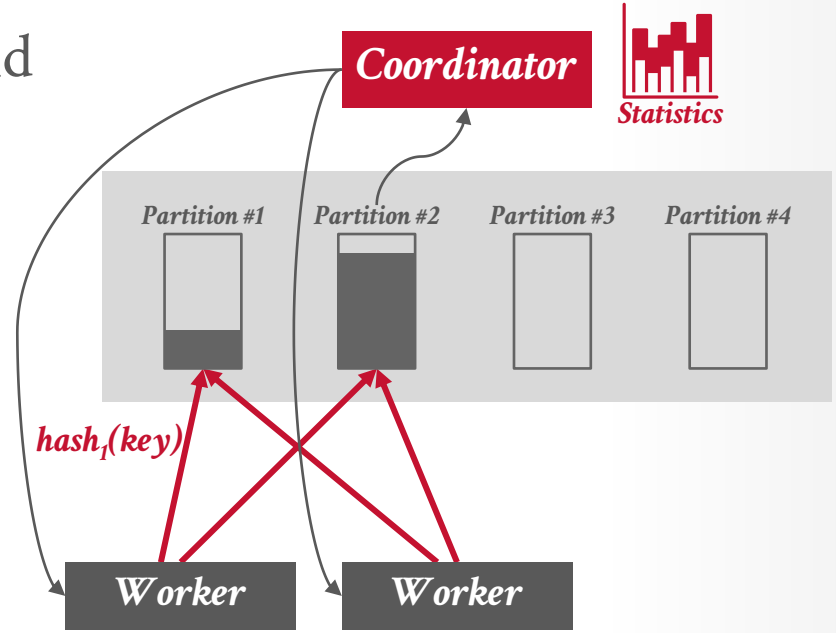


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

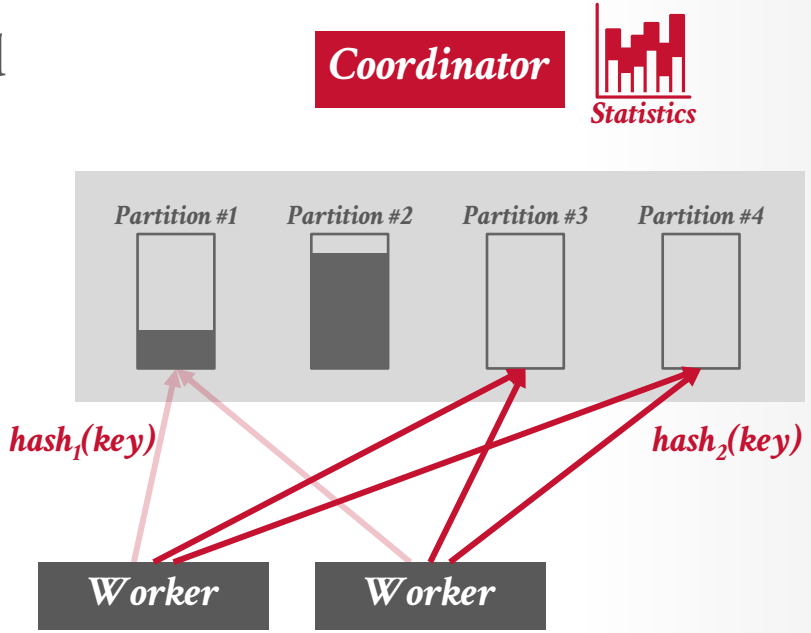


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

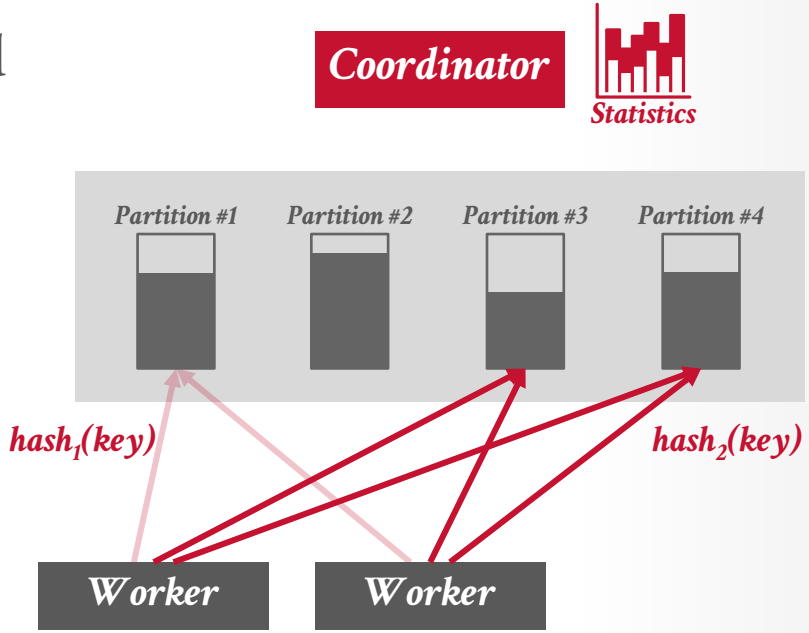


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

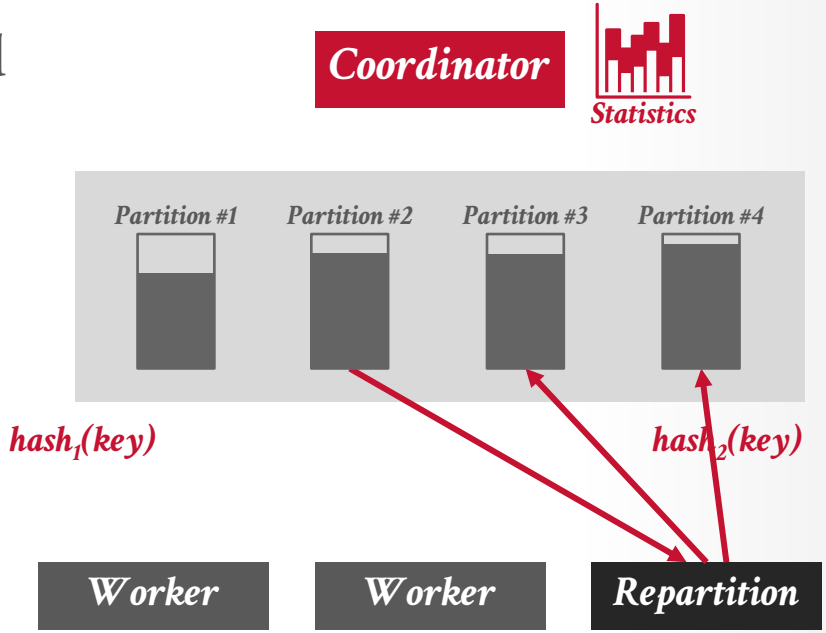


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

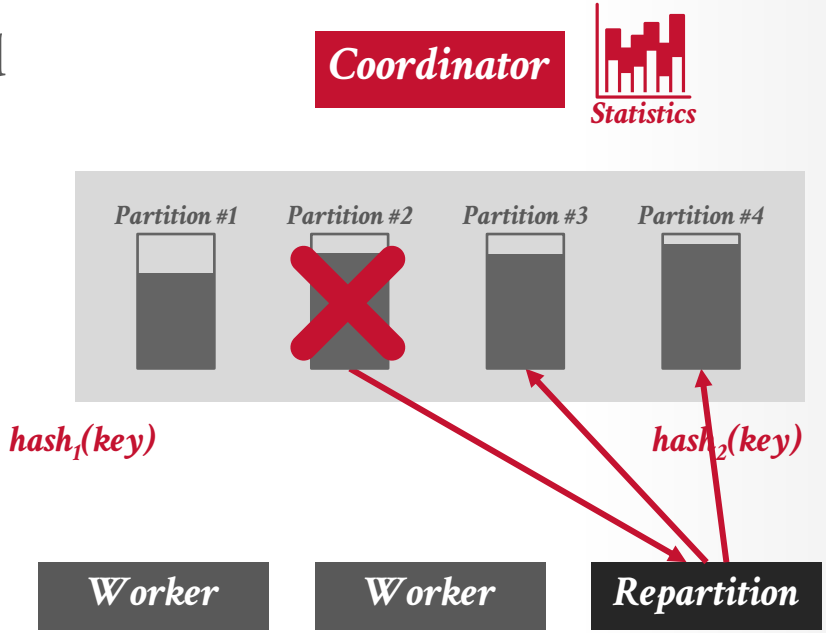


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.

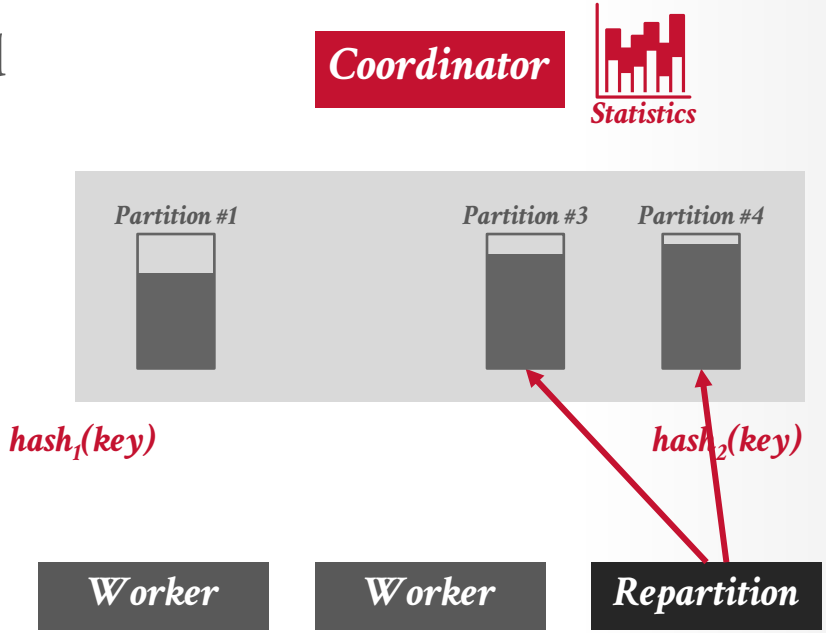


Source: H.Ahmadi + A.Surna

DREMEL: DYNAMIC REPARTITIONING

Dremel dynamically load balances and adjusts intermediate result partitioning to adapt to data skew.

DBMS detects whether shuffle partition gets too full and then instructs workers to adjust their partitioning scheme.



Source: H.Ahmadi + A.Surna

DREMEL: STORAGE

DBMS relies on Google's distributed file system (Colossus) to scale out storage capacity.

Relies on Capacitor's columnar encoding scheme for nested relational and semi-structured data.

→ Think of it as JSON/YAML without the slowness.
→ Capacitor also provides access libraries with basic filtering.
→ Similar to Parquet vs. ORC formats.

Repetition and definition fields are embedded in columns to avoid having to retrieve/access ancestor attributes.

DREMEL: SCHEMA REPRESENTATION

Dremel's internal storage format is self-describing

→ Everything the DBMS needs to understand what is in a file is contain within the file.

But the DBMS must parse a file's embedded schema whenever it wants to read that a file.

→ Tables can have thousands of attributes. Most queries only need a subset of attributes.

DBMS stores schemas in a columnar format to reduce overhead when retrieving meta-data.

DREMEL: SQL

In the early 2010s, many of Google's internal DBMS projects each had their own SQL dialect.

The GoogleSQL project unified these redundant efforts to build a data model, type system, syntax, semantics, and function library.

(Zombie?) Open-Source Version: ZetaSQL

SPANNER: BECOMING A SQL SYSTEM
SIGMOD 2017

OBSERVATION

Since the 2011 VLDB paper, there are DBMS projects that are copies or inspired by Dremel.

→ **Apache Drill** (MapR)
→ **Presto** (Meta)
→ **Apache Impala** (Cloudera)
→ **Dremio**

There are also shuffle-as-a-service systems:

→ **Apache Celeborn** (Alibaba)
→ **Apache Uniffle** (Tencent)
→ **Remote Shuffle Service** (Uber)

APACHE DRILL

Drill is an open-source implementation of Dremel built on top of Hadoop.
→ Project started in 2012 at MapR.

Supports query codegen via Janino embedded Java compiler.

HPE announced in 2020 that they will no longer support Drill development.

PRESTODB

Started at Facebook in 2012 to replace Apache Hive query engine based on Hadoop.

→ Java-based execution engine for data lakes.
→ Many connectors to different storage systems and DBMSs.
→ Replace PrestoDB's Java-based runtime engine with Velox-based engine called Prestissimo.

Hard-forked in 2019 by Starburst Data into Trino (formerly PrestoSQL) because Meta would not give up control of source code.

PRES

Started at Facebook in 201
Apache Hive query engine
→ Java-based execution engine
→ Many connectors to differer
→ Replace PrestoDB's Java-bas
based engine called Prestissi

Hard-forked in 2019 by St
Trino (formerly PrestoSQ
would not give up contro

Question of the episode: Will Trino be making a vectorized C++ version of Trino workers?

Full question from Trino Slack

Answer: Writing a C++ worker would require each plugin to be implemented in C++ as well. However, you don't need C++ for vectorization. Java already does a technique called auto-vectorization which we will demonstrate later in the show! Java 17 also introduces the new Vector API which unlocks complex usage patterns that we can invest in moving forward. However, there's so much more to making operations faster than just bare metal speed that we are going to focus on.

To demonstrate this, I'd like to use an analogy about how I think of this. Comparing C++ and Java implementation is like comparing the two fastest men in the world. Usain Bolt holds the most world records for mens track to this date, and teammate Yohan Blake holds many of the second place titles. Most of us know Usain Bolt is the fastest of the two, and you may not have known or remembered Yohan's name before. Want to hear something crazy, Yohan has beaten Usain Bolt in a few races. The two are so close in speed, it's seconds to milliseconds difference. The main difference in this analogy is that speed is the only thing that matters in an olymic race. Howver, programming languages and frameworks have a lot more tradeoffs.

The point is, Java is fast and more importantly, it removes a lot of burden maintaining and scaling out the code. This is conducive to a healthy open-source project, and lowers the barrier for collaboration. Rather than go against this and take on the feat of having to rewrite an entire system in C++, why not lean into the incredible innovation recent Java features have to offer to improve performance even more.

Another important aspect is rather than chasing the fastest bare metal speed, it's also incredibly important to dedicate time into ensuring that Trino's optimizer is producing the best possible plans to avoid doing unnecessary work. To continue with the analogy, in a 100m race on a 400m track, imagine we have Usain and Yohan go head to head. We may expect that Usain will likely win, given his track record. However, if Usain is given the wrong instructions and runs in the wrong direction (300m), my bets are that Yohan will win the race.

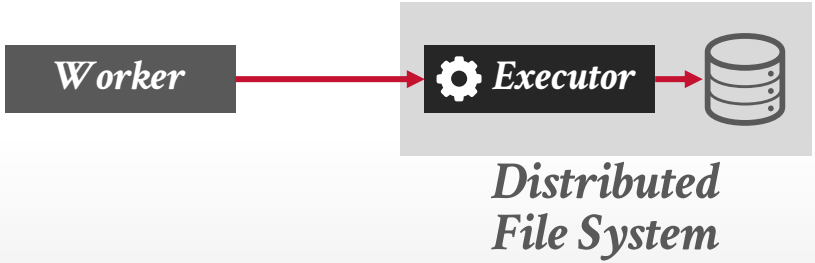
In essence, the direction of Trino while still including bare metal performance improvements in the JVM, will instead focus on not wasting time with suboptimal query plans before or during runtime. There are so many optimizations that are constantly being added to every release that ultimately makes for a work-smarter-not-harder query engine.

APACHE IMPALA

Impala is another Dremel inspired DBMS for executing queries on distributed filesystems.
→ Started in 2012 at Cloudera by ex-Google DB people.

Supports codegen of filters and parsing logic.

Co-locate executor component on each data node to provide parsing and predicate pushdown.



DREMIO

Open-source / commercial DBMS inspired by Dremel based on Apache Arrow.
→ Started in 2015 by CMU alum.

Leverages user-defined materialized views ("reflections") to speed up query execution on external data files.

Also relies on Java-based codegen and vectorization.

APACHE CELEBORN

Standalone shuffle-as-a-service system written in Java that replaces the built-in shuffle mechanisms of Spark + Flink.

→ Decouples shuffle operation from workers.

Maintains its own buffer pool with support for block compression, and spilling data to local disks and HDFS.

PARTING THOUGHTS

Dremel is an innovative DBMS that predates all other major cloud-native OLAP DBMSs.

The shuffle phase seems wasteful but it simplifies engineering and can improve performance.

It is also a good example of the benefit of decomposing a DBMS's components into individual services to abstract raw resources.

PARTING THOUGHTS

Dremel is an innovative DBMS that predates all other major cloud-native OLAP DBMSs.

The shuffle phase seems wasteful but it simplifies engineering and can improve performance.

It is also a good example of the benefit of decomposing a DBMS's components into individual services to abstract raw resources.