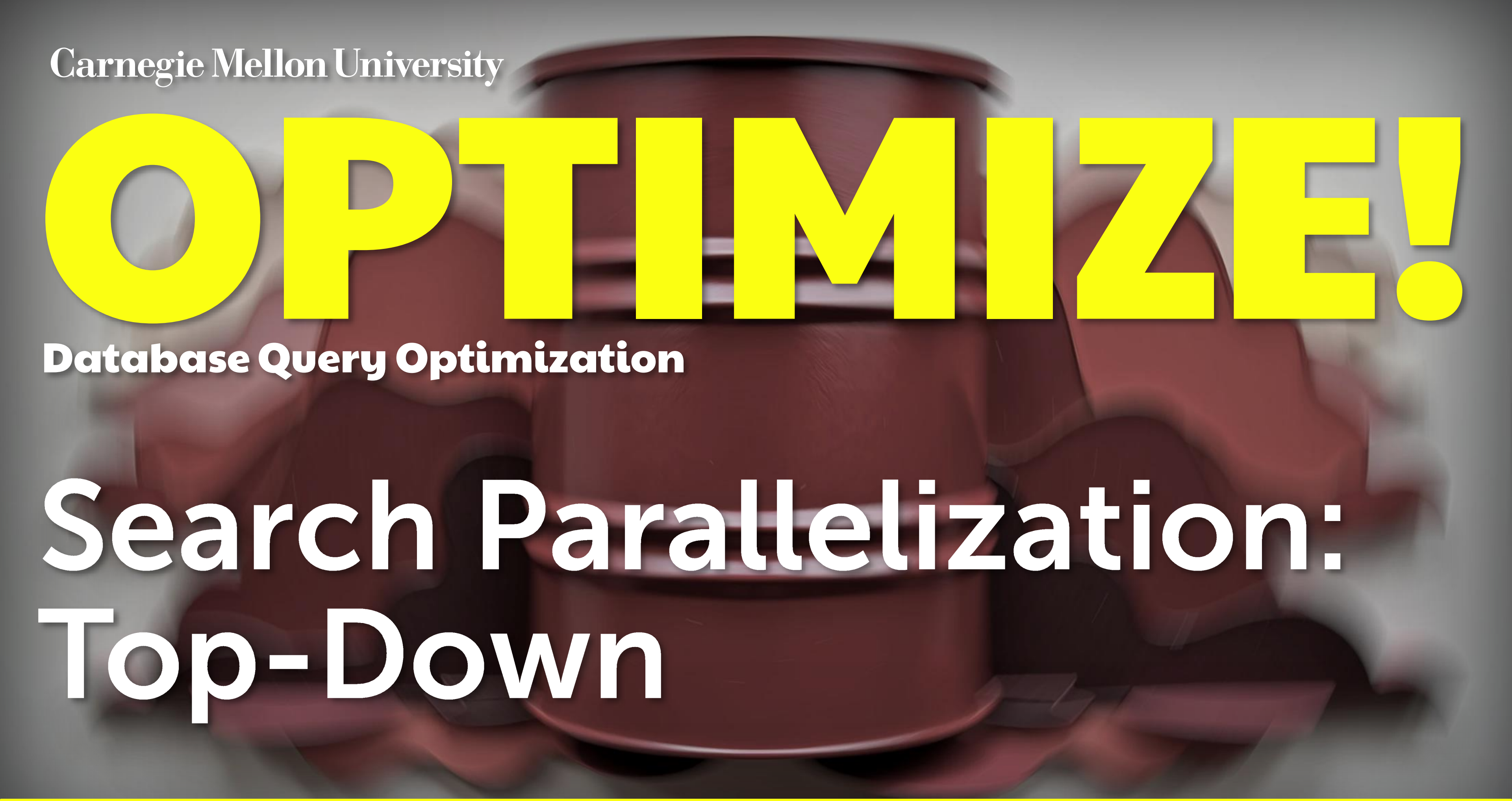
Carnegie Mellon University

OPTIMIZE!

Database Query Optimization



Search Parallelization: Top-Down

SPRING 2025 » SPECIAL TOPICS IN DATABASES » PROF. ANDY PAVLO

UPCOMING DATABASE TALKS

The Germans (DB Seminar)
→ Monday Feb 17th @ 4:30pm ET
→ Zoom

Pinot (DB Seminar)
→ Monday Feb 24th @ 4:30pm ET
→ Zoom

Malloy (DB Seminar)
→ Monday Mar 3rd @ 4:30pm ET
→ Zoom

LAST CLASS

We discussed a parallel join enumeration algorithm for bottom-up query optimization.

→ These apply rules / heuristics before switching to the join enumeration phase.

Key Idea: Partition a query's search space according to the join graph so that workers can process independent portions.

TRANSFORMATION SEARCH SPACE

Since all changes made to a plan are transformations in a top-down optimizer, the search space mostly contains alternatives not related to join ordering.

```
SELECT n_name,
    SUM(l_extendedprice * (1 - l_discount)) AS revenue
FROM
    customer, orders, lineitem, supplier, nation, region
WHERE
    c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND l_suppkey = s_suppkey
    AND c_nationkey = s_nationkey
    AND s_nationkey = n_nationkey
    AND n_regionkey = r_regionkey
    AND r_name = 'ASIA'
    AND o_orderdate >= date '1994-01-01'
    AND o_orderdate < date '1994-01-01' + interval '1' year
GROUP BY n_name
ORDER BY revenue DESC;
```

Example: TPC-H Query 6

→ Join Order Search Space: **<100,000**

→ Everything Else Search Space: **230,000,000**

Source: Venkatesh Raghaven

CASCADES: TASK-BASED SEARCH

Optimizer maintains a LIFO stack of tasks to perform actions on groups and expressions.

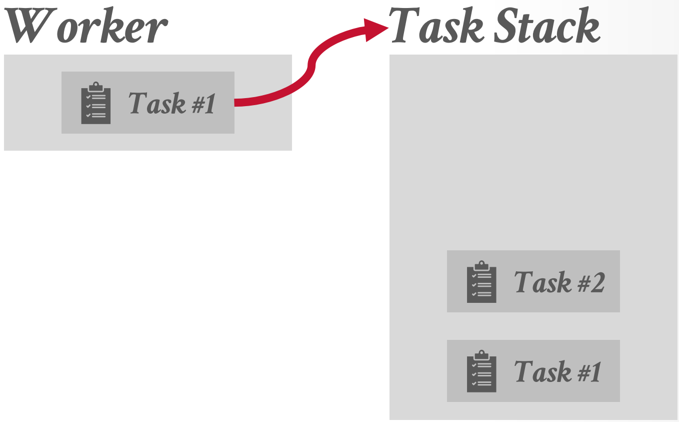
Stack ensures expressions are derived after the best plans of its input expressions are derived.

→ Tasks are stored in the heap rather than in the program stack to reduce OOM errors.

CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

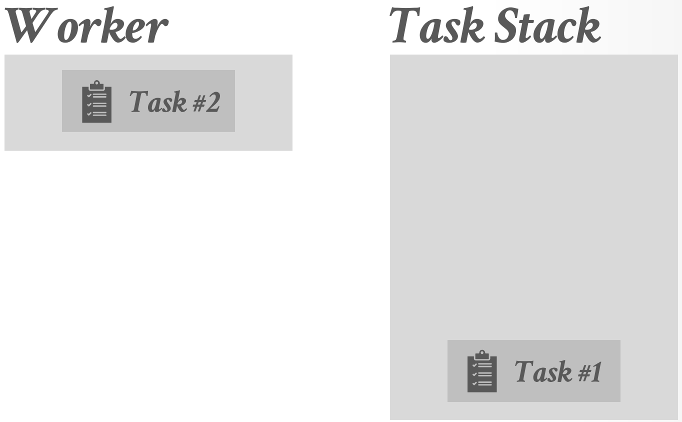
This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

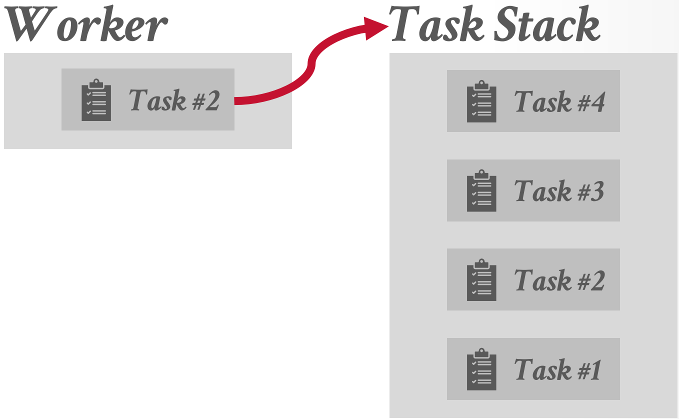
This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

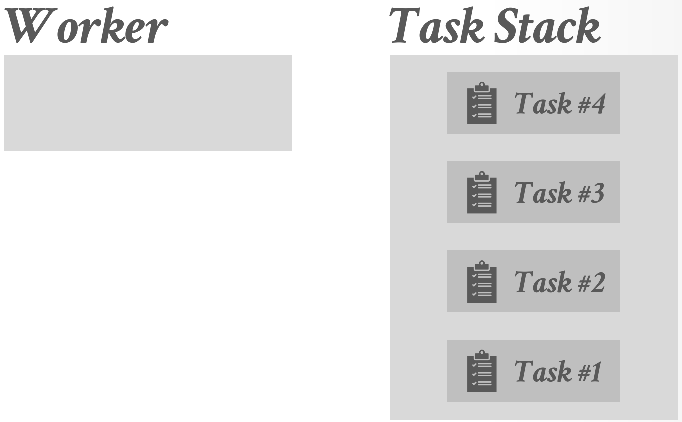
This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



Worker

Task Stack

Task #4

Task #3

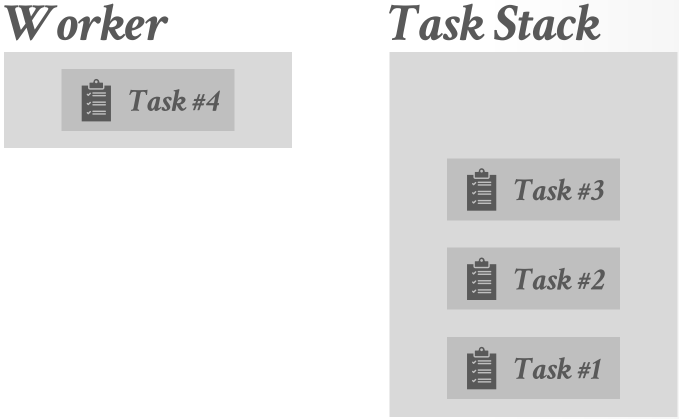
Task #2

Task #1

CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



Worker

Task #4

Task Stack

Task #3

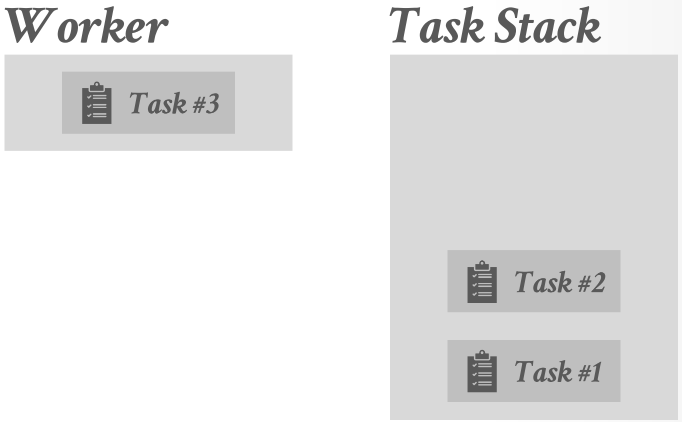
Task #2

Task #1

CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

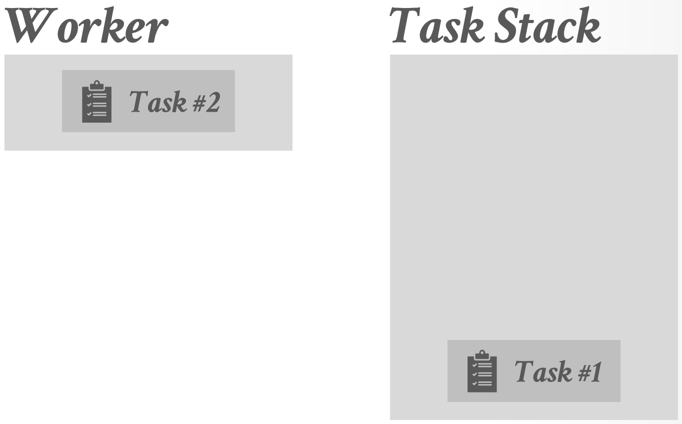
This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does <u>not</u> preserve dependencies between tasks.

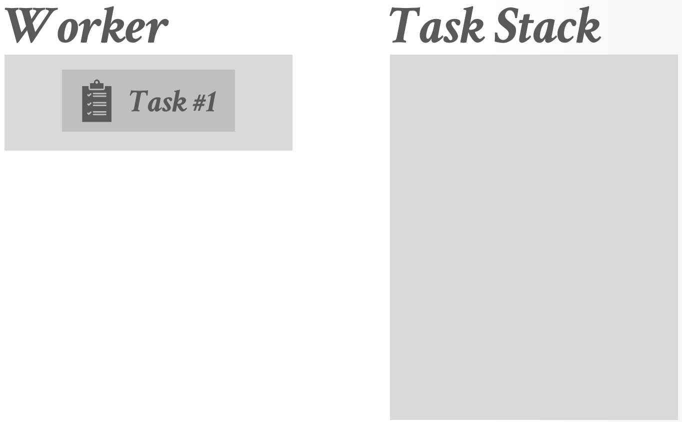
This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



CASCADES: TASK-BASED SEARCH

The original Cascades stack-based scheduling does not preserve dependencies between tasks.

This restricts execution to a single worker because the optimizer cannot guarantee a task's dependencies complete before it starts running.



TODAY'S AGENDA

Parallel Top-Down Search

Project #2

PARALLEL TOP-DOWN CASCADES

Replace the optimizer's task stack with a scheduler that tracks the state of each task and can execute any task once its runnable.

Encode dependencies between tasks as child-parent links in a dependency graph.

→ A parent task can start before its children start, but a parent task cannot finish before its children finish.

Precursor to the Greenplum Orca optimizer.

PARALLELIZING EXTENSIBLE QUERY OPTIMIZERS
SIGMOD 2009

TASKS

Explore(*g*):

→ Generate logically equivalent expressions of all group expressions in group ***g***.

Explore(*gexpr*):

→ Generate logically equivalent expressions of a group expression ***gexpr***.

Imp(*g*):

→ Generate implementations of all group expressions in group ***g***.

Imp(*gexpr*):

→ Generate implementation alternatives of a group expression ***gexpr***.

Opt(*g*, *req*):

→ Return the plan with the least estimated cost that is rooted by an operator in group ***g*** and satisfies optimization request ***req***.

Opt(*gexpr*, *req*):

→ Return the plan with the least estimated cost that is rooted by ***gexpr*** and satisfies optimization request ***req***.

Xform(*gexpr*, *t*):

→ Transform group expression ***gexpr*** using rule ***t***.

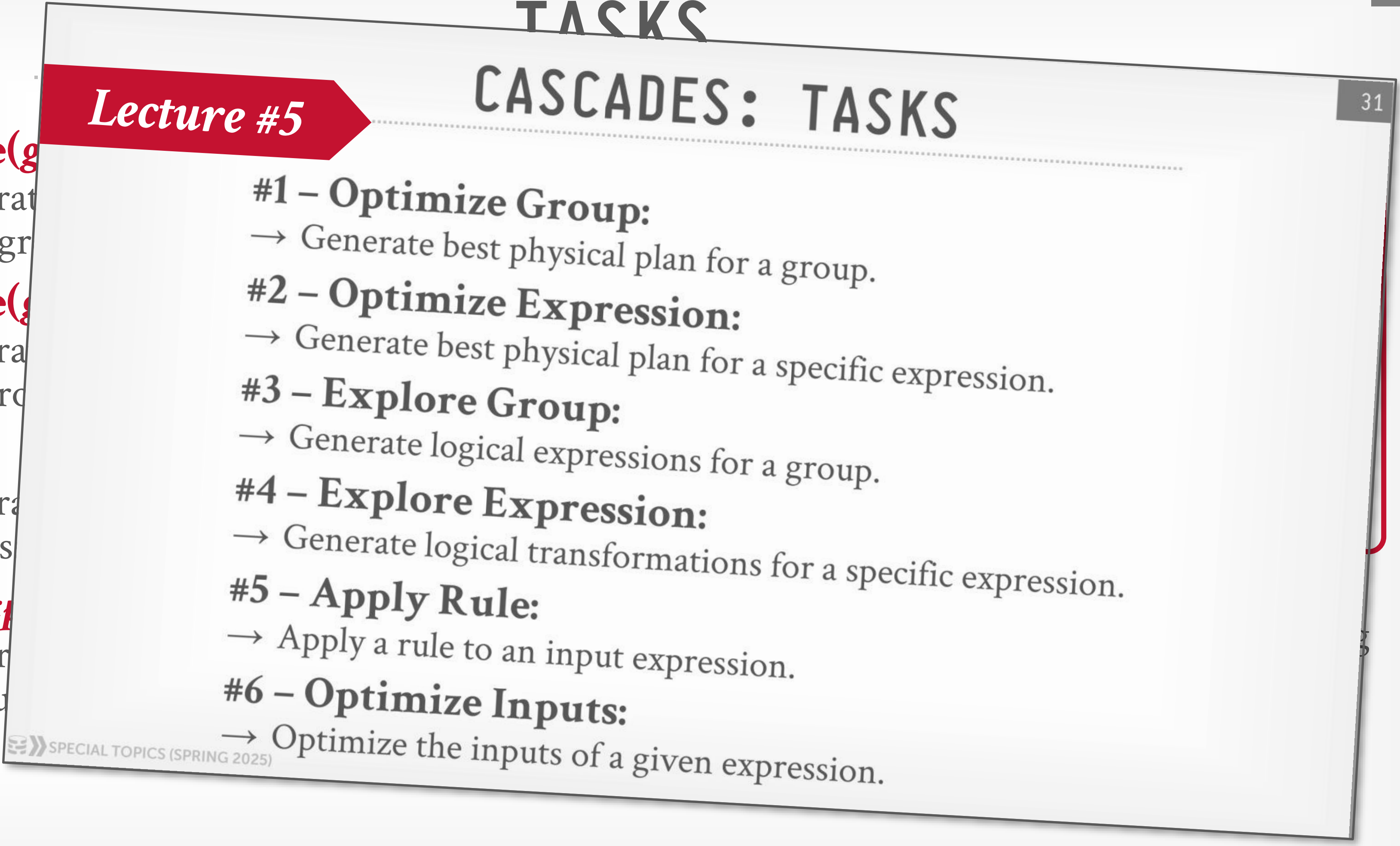
TASKS

Explore(g
→ Generat
of all gr

Explore(g
→ Genera
of a gro

Imp(*g*):
→ Genera
expres

Imp(*gex*
→ Gener
a grou



Lecture #5

CASCADES: TASKS

#1 – Optimize Group:
→ Generate best physical plan for a group.

#2 – Optimize Expression:
→ Generate best physical plan for a specific expression.

#3 – Explore Group:
→ Generate logical expressions for a group.

#4 – Explore Expression:
→ Generate logical transformations for a specific expression.

#5 – Apply Rule:
→ Apply a rule to an input expression.

#6 – Optimize Inputs:
→ Optimize the inputs of a given expression.

TASK SCHEDULING

As a task executes, it can generate additional tasks that either fan out from the current group or traverse down into the search tree.

Tasks are defined in terms of their **goal**.

- → Example: **Explore(g_0)**
- → When a task with a certain goal is running, all newly created tasks with that same goal are paused until the first task completes. Resumed tasks retrieve results from memo.

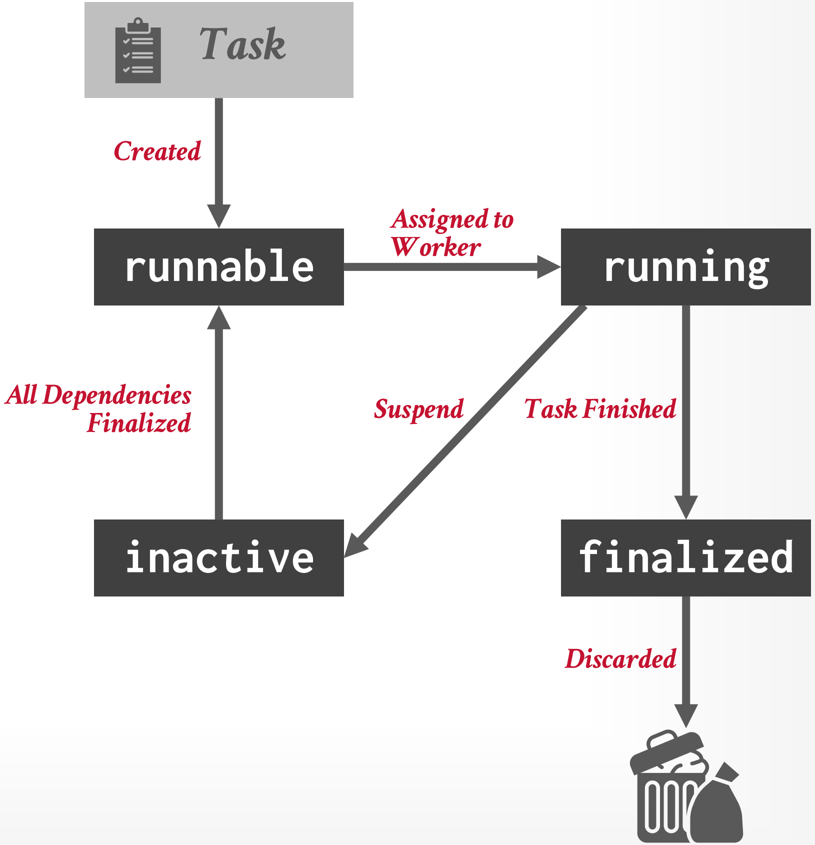
SEARCH STATE DEPENDENCY GRAPH

Runnable: The task can be assigned to a worker for execution.

Running: A worker is actively executing this task, and it cannot be assigned to another worker.

Inactive: The task is waiting for dependent tasks to complete.

Finalized: The task is complete and can be discarded.



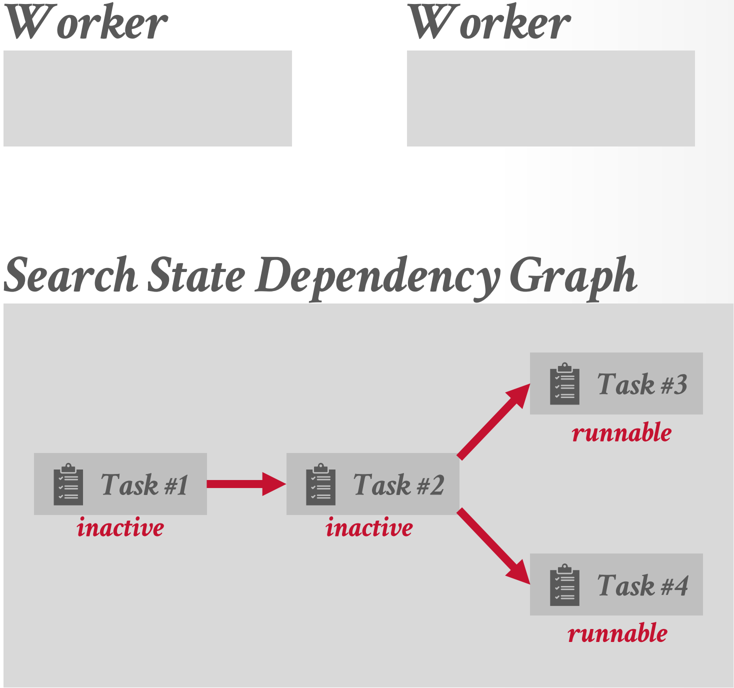
SEARCH STATE DEPENDENCY GRAPH

When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

All optimizer tasks are reentrant.

→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.



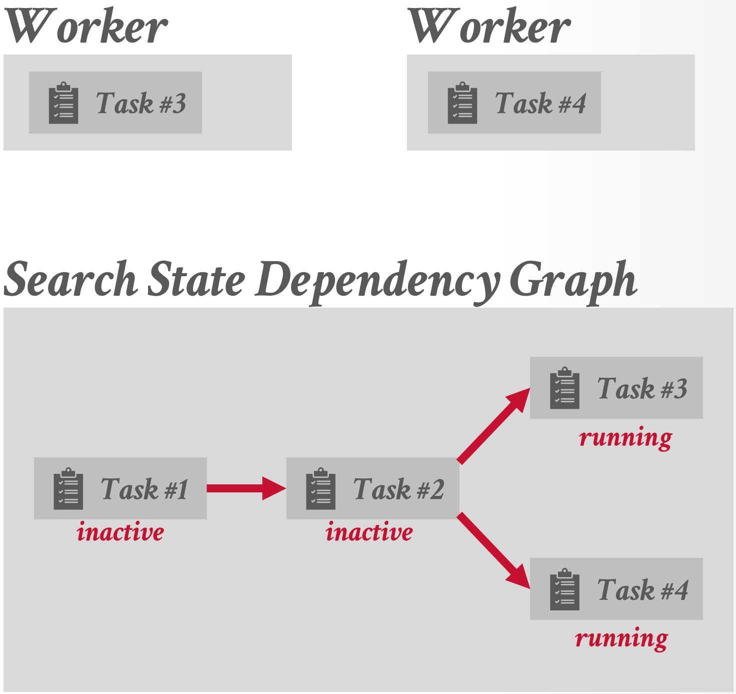
SEARCH STATE DEPENDENCY GRAPH

When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

All optimizer tasks are reentrant.

→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.



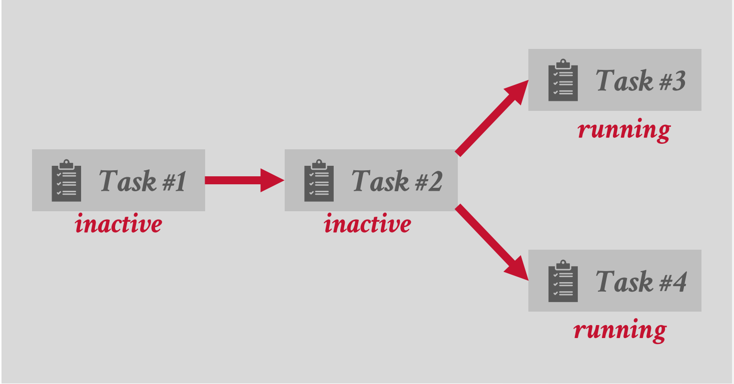
SEARCH STATE DEPENDENCY GRAPH

When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

All optimizer tasks are <u>reentrant</u>.
→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.

Search State Dependency Graph



SEARCH STATE DEPENDENCY GRAPH

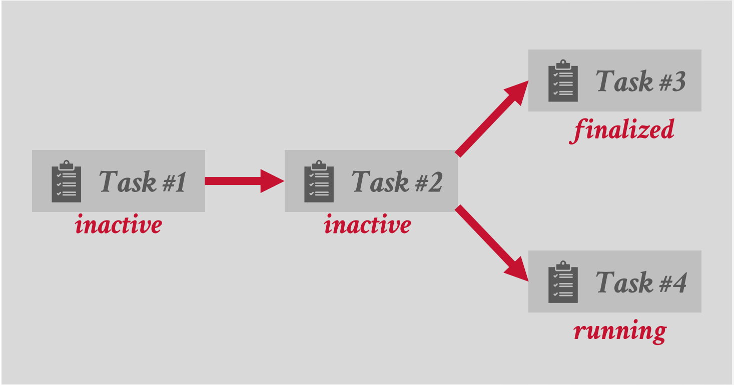
When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

All optimizer tasks are reentrant.

→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.

Search State Dependency Graph



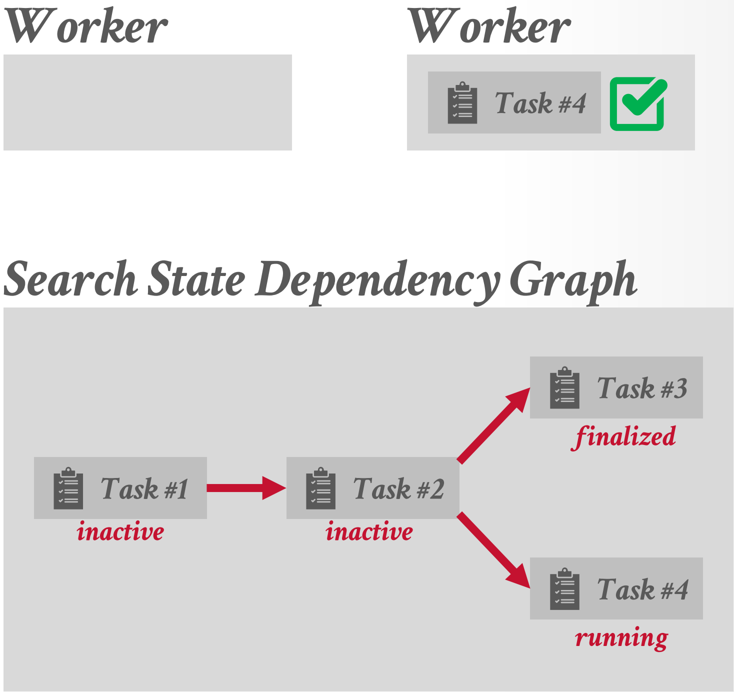
SEARCH STATE DEPENDENCY GRAPH

When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

All optimizer tasks are reentrant.

→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.



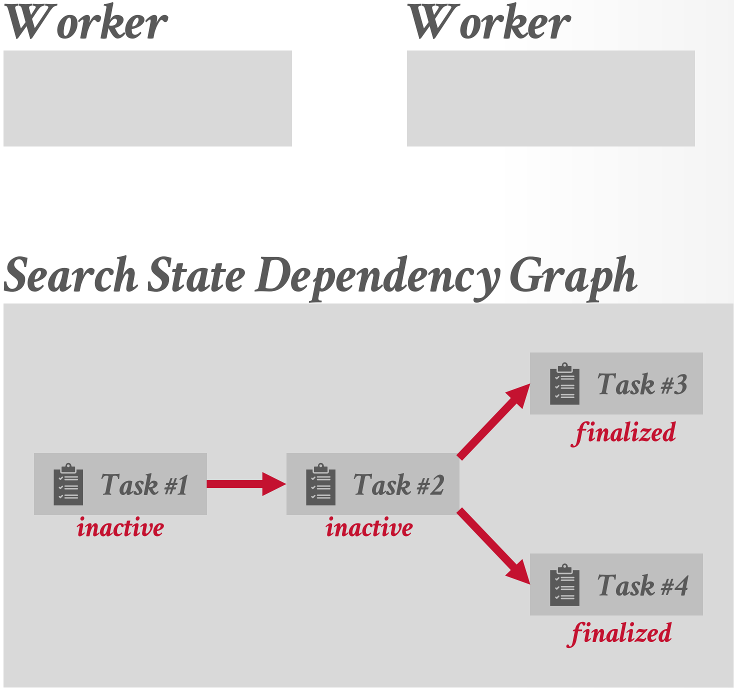
SEARCH STATE DEPENDENCY GRAPH

When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

All optimizer tasks are reentrant.

→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.



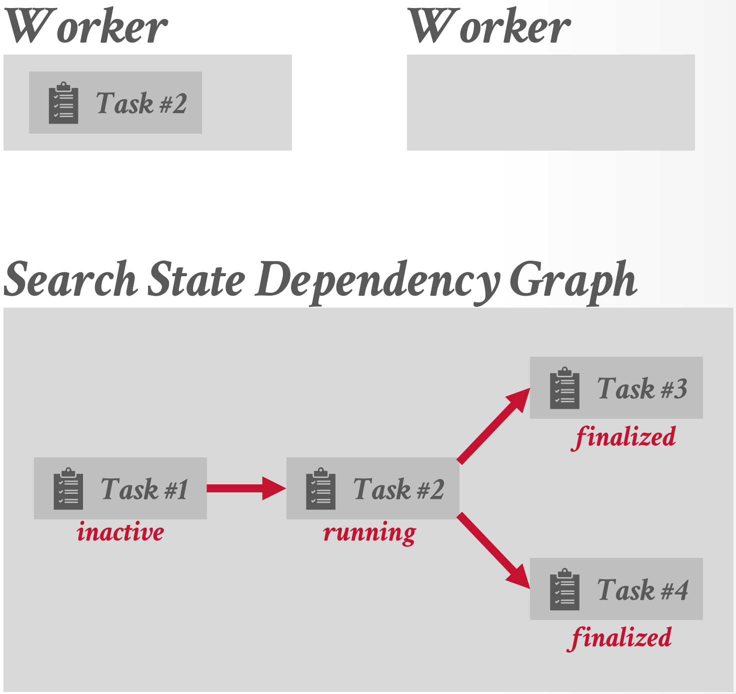
SEARCH STATE DEPENDENCY GRAPH

When a new task is added to the SSDG or when a task completes, the scheduler assigns any runnable task to an idle worker.

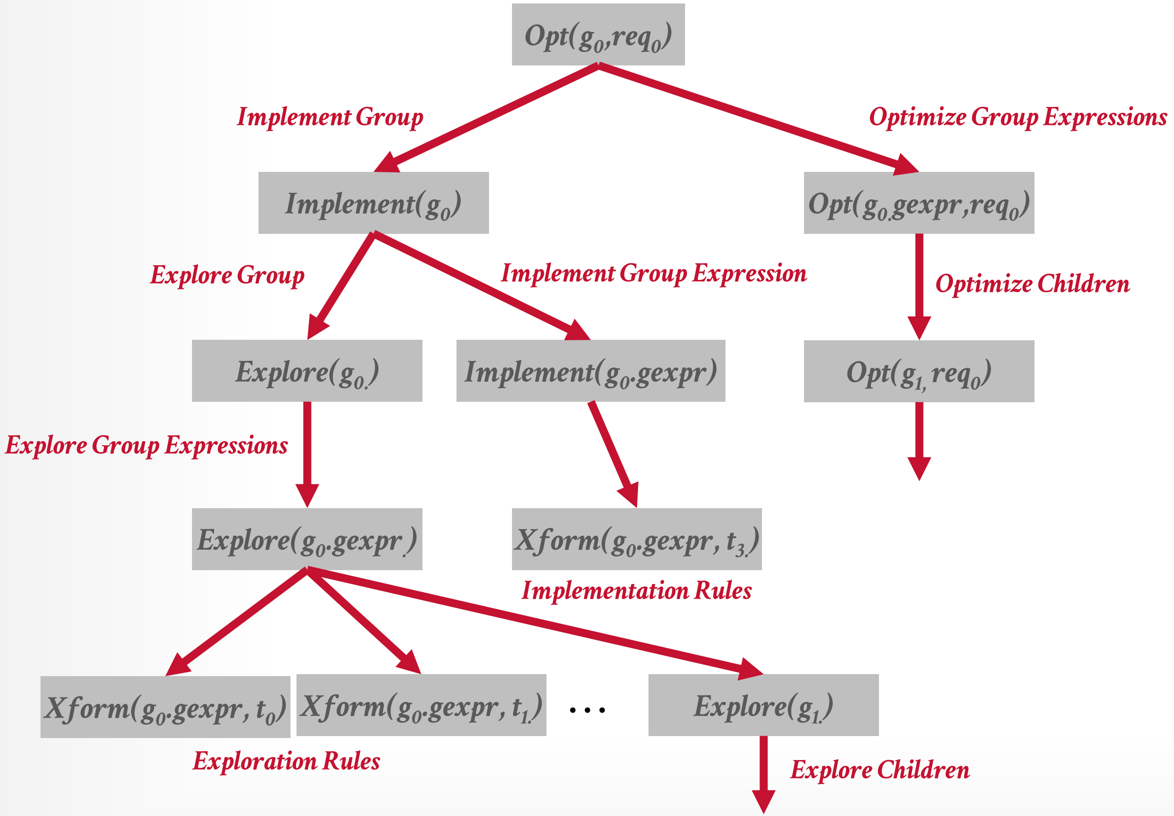
All optimizer tasks are reentrant.

→ The worker can pause a task, switch to another one, and resume the first task at the same point it was paused.

Priority is based on task promises.



ORCA: PARALLEL TASK EXAMPLE



$Opt(g_0,req_0)$

Implement Group

Optimize Group Expressions

$Implement(g_0)$

$Opt(g_0.gexpr,req_0)$

Explore Group

Implement Group Expression

Optimize Children

$Explore(g_0)$

$Implement(g_0.gexpr)$

$Opt(g_1, req_0)$

Explore Group Expressions

$Explore(g_0.gexpr)$

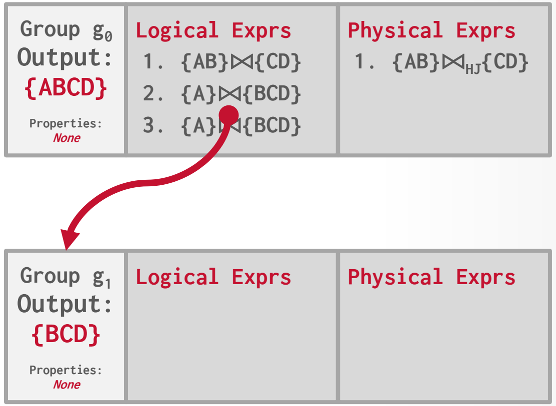
$Xform(g_0.gexpr, t_3)$

Implementation Rules

$Xform(g_0.gexpr, t_0)$ $Xform(g_0.gexpr, t_1)$ … $Explore(g_1)$

Exploration Rules

Explore Children



Group g_0 Output: {ABCD} Properties: *None*	Logical Exprs	Physical Exprs
	1. {AB}⋈{CD} 2. {A}⋈{BCD} 3. {A}⋈{BCD}	1. {AB}$\bowtie_{HJ}${CD}

Group g_1 Output: {BCD} Properties: *None*	Logical Exprs	Physical Exprs

OPPORTUNITY FOR PARALLELISM

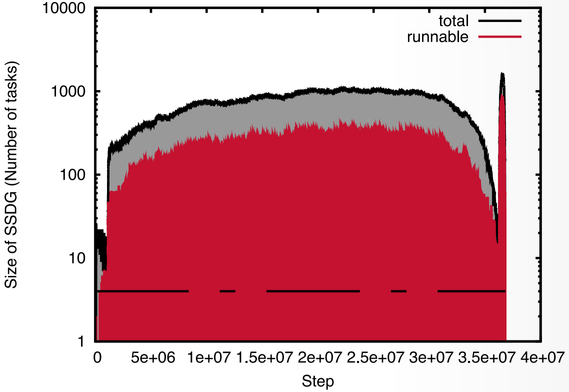
Intel Core 2 Quad Core Q6600

Measure the total number of tasks versus the number of runnable tasks in the SSDG over time.

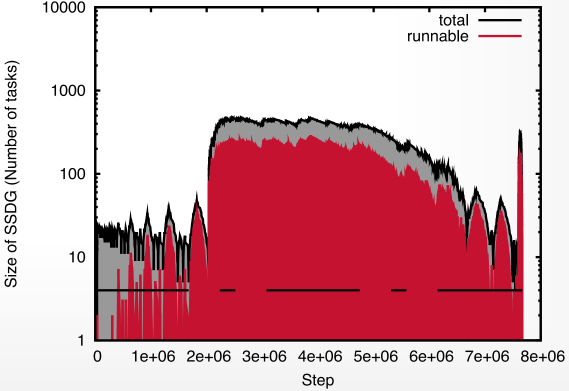
→ Query #1: 10-way join, star-shaped graph
→ Query #2: 10-way join, linear/chain graph

Both queries show an initial phase that requires sequential processing. But then search space opens to more parallelizable tasks.

Query #1: Star



Query #2: Chain



Source: Florian Waas

SPEED-UP

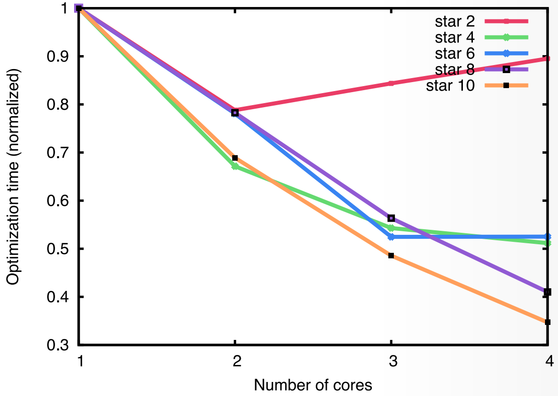
Intel Core 2 Quad Core Q6600

Measure the relative performance improvement as the optimizer is given more CPU cores for workers.

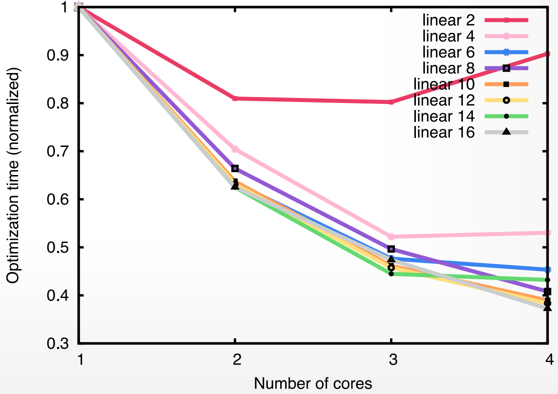
Opportunities for parallelism increases with query complexity.

Latch contention on memo table limits the scalability of the optimizer.

Query #1: Star



Query #2: Chain



Source: Florian Waas

ERLANG

Functional programming language from the 1980s based on message-passing actors.

The Erlang runtime supports sophisticated features out of the box:
→ Fault-tolerance
→ Hot swapping
→ Lightweight Green Threads

Erlang is the most used functional PL in DBMS implementations.

Couchbase CouchDB relax riak mnesia AntidoteDB

ERLANG

Functional programming language from th
based on message-passing actors.

The Erlang runtime supports sophisticated
out of the box:

→ Fault-tolerance
→ Hot swapping
→ Lightweight Green Threads

Erlang is the most used functional PL in D
implementations.

Implementation	
C++	184
Java	150
C	120
Go	94
Rust	68
Python	45
C#	34
JavaScript	31
Erlang	13
Scala	12

Couchbase CouchDB relax riak mnesia AntidoteDB

PARTING THOUGHTS

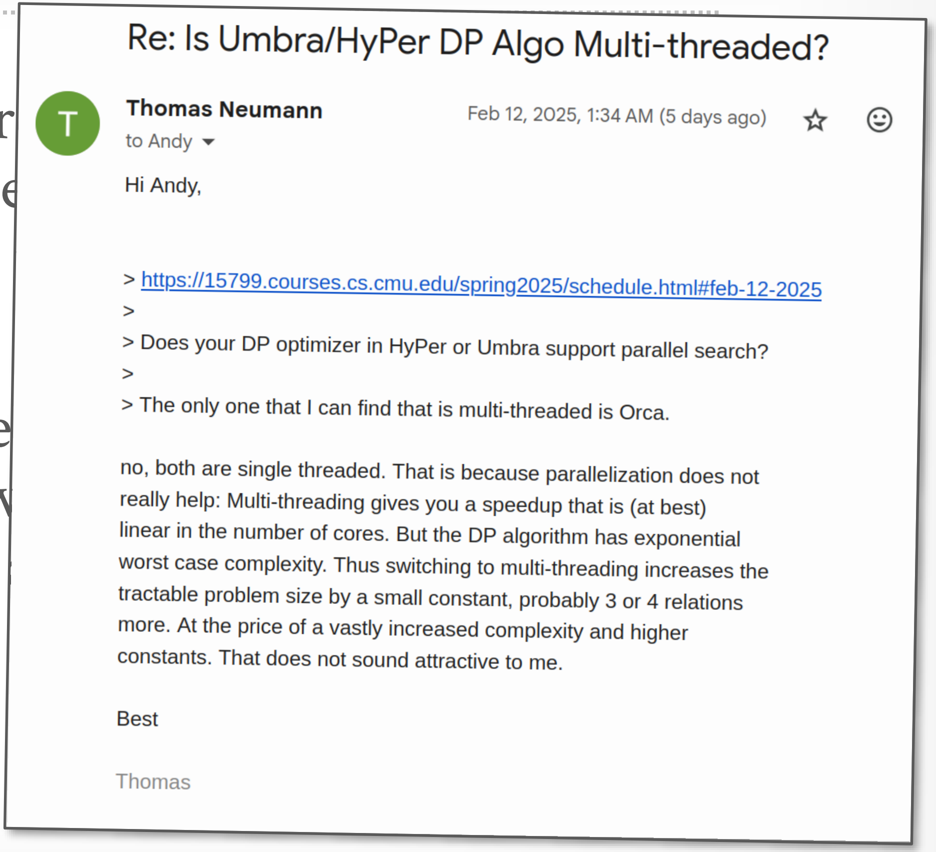
Top-down optimizers are more amenable to parallel implementations because they are considering multiple transformation types at the same time.

Counter Argument: The "throw it all in!" nature of exploration in a top-down optimizer is so wasteful that one is forced to use a parallel implementation.

PARTING THOUGHTS

Top-down optimizers ar
implementations becaus
multiple transformation

Counter Argument: The
exploration in a top-dow
that one is forced to use



Re: Is Umbra/HyPer DP Algo Multi-threaded?

Thomas Neumann
to Andy

Feb 12, 2025, 1:34 AM (5 days ago)

Hi Andy,

> https://15799.courses.cs.cmu.edu/spring2025/schedule.html#feb-12-2025
>
> Does your DP optimizer in HyPer or Umbra support parallel search?
>
> The only one that I can find that is multi-threaded is Orca.

no, both are single threaded. That is because parallelization does not really help: Multi-threading gives you a speedup that is (at best) linear in the number of cores. But the DP algorithm has exponential worst case complexity. Thus switching to multi-threading increases the tractable problem size by a small constant, probably 3 or 4 relations more. At the price of a vastly increased complexity and higher constants. That does not sound attractive to me.

Best

Thomas

NEXT CLASS

Unnesting Arbitrary Queries (The German Way)

PROJECT #2: FINAL PROJECT

Group project to implement some substantial component or feature in a query optimizer.

Projects should incorporate topics discussed in this course as well as from your own interests.

Each group must pick a project that is unique from their classmates.

https://15799.courses.cs.cmu.edu/spring2025/project2.html

PROJECT #2 – DELIVERABLES

Proposal Presentation: March 10th

Status Update Presentation: April 7th

Design Document: Final Exam Date (TBA)

Final Presentation: Final Exam Date (TBA)

PROJECT #2: PROPOSAL

Five-minute presentation to the class that discusses the high-level topic.

Each proposal must discuss:
→ Architecture and implementation overview of the project.
→ How you will test whether your implementation is correct.
→ What workloads you will use for your project.

PROJECT #2: STATUS UPDATE

Five-minute presentation to update the class about the current status of your project.

Each presentation should include:
→ Current development status.
→ Whether your plan has changed and why.
→ Anything that surprised you during coding.

PROJECT #2: DESIGN DOCUMENT

As part of the status update, you must provide a design document that describes your project implementation:

- → Architectural Design
- → Design Rationale
- → Testing Plan
- → Trade-offs and Potential Problems
- → Future Work

PROJECT #2: FINAL PRESENTATION

10-minute presentation on the final status of your project during the scheduled final exam.

You should include any performance measurements or benchmarking numbers for your implementation.

Demos are always hot too…

PROJECT #2: TOPICS

Our goal was to have all projects based on CMU-DB's optd project.

We are in the process of rewriting so unfortunately it is not ready for others to start contributing.

→ Some projects can start as standalone prototypes that we can work to integrate into the fall semester.

PROJECT #2: TOPICS

Learned Transformation Promises

German Arbitrary Unnesting via Transformations

Verified LLM SQL Rewriting

Transformation Rule Lingua Franca + Corpus

Injecting PostgreSQL Statistics

Testing / Benchmark Suite for Optimizers

Deparsing Physical Plans to PostgreSQL with Hints

Predicate Embeddings

HOW TO START

Form a team. Sign-up on class spreadsheet.

Meet with your team and discuss potential topics.

Plan a (rough) schedule on what you will need to implement.

I am around during Spring Break for additional discussion and clarification of the project idea.