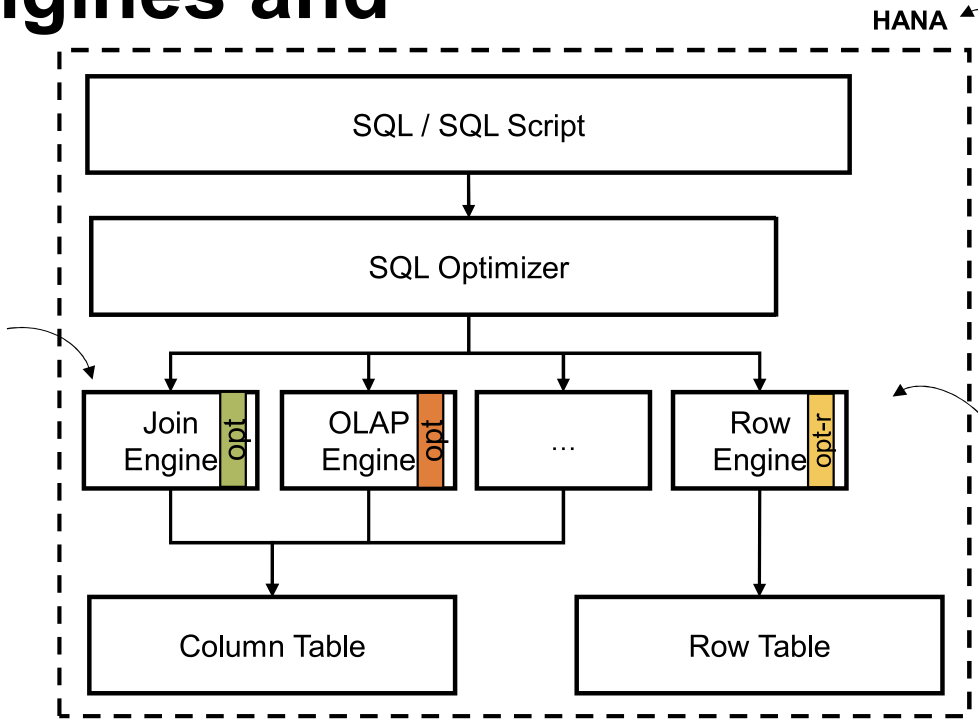
SAP SE

HEX : SAP’s new HANA Execution Engine

Daniel Ritter
Cloud Database Architect & Member of HANA Research Campus at SAP SE

Sponsor talk, Conference on Innovative Data Systems Research (CIDR), 1/2023

HANA Core Engines and Stores



In-memory + disk (cf. NSE), HTAP (cf. Translytical Data Platform)

Modular, disconnected

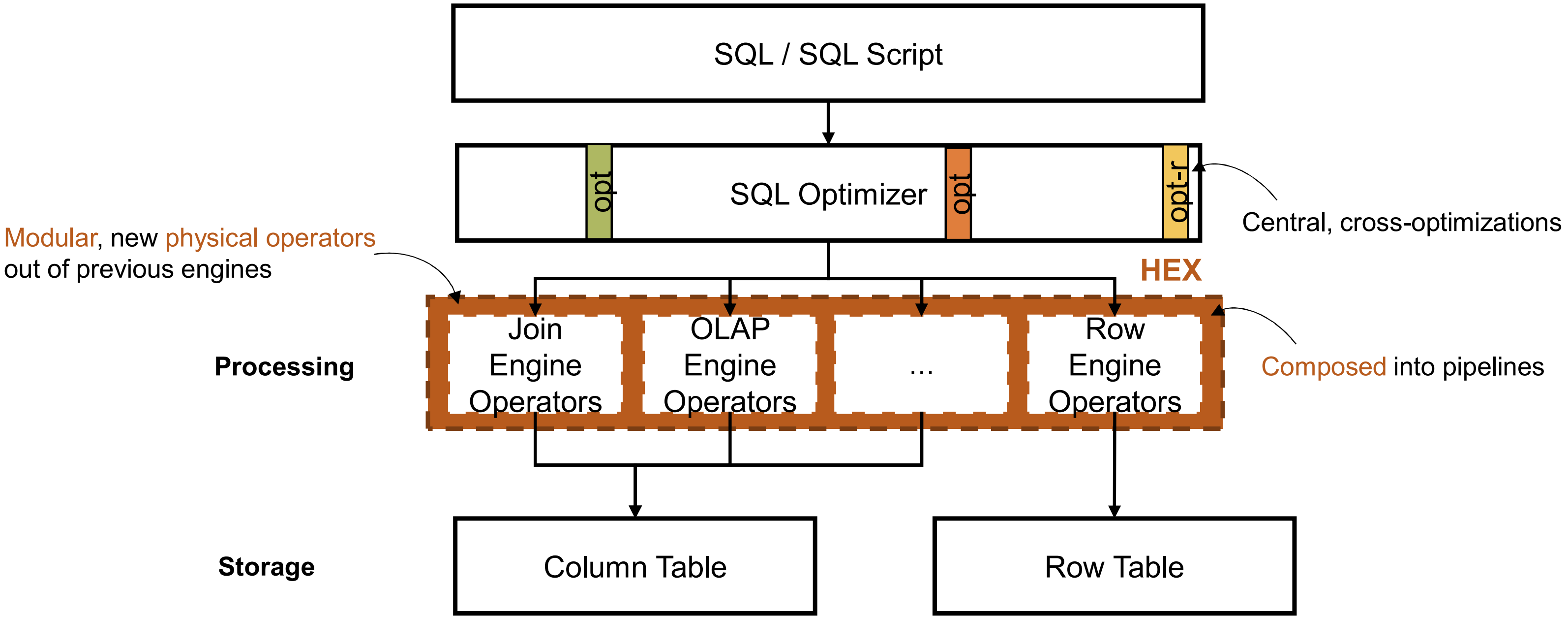
Processing

Storage

… and many more: e.g. Calculation engine (calculation views, star joins), MDS engine (multi-dimensional queries incl. aggregation, transformation, calculation), …

- Färber, Franz, et al. "The SAP HANA Database - An Architecture Overview." *IEEE Data Eng. Bull.* 35.1 (2012): 28-33.
- Sherkat, Reza, et al. “Native Store Extension for SAP HANA" *Proceedings of the VLDB Endowment* 12.12 (2019): 2047-2058.
- Translytical Data Platforms, Forrester, Q4/2022: https://news.sap.com/2022/12/translytical-data-platforms-forrester-wave-sap-a-leader/

HEX


SQL / SQL Script
SQL Optimizer
opt
opt
opt-r
Central, cross-optimizations
Modular, new physical operators
out of previous engines
HEX
Processing
Join Engine Operators
OLAP Engine Operators
…
Row Engine Operators
Composed into pipelines
Storage
Column Table
Row Table

Overview

Engine	Proc. Model	Data flow model	Level of Parallelism	Workload
DuckDB	Vectorized	Pull ("Vector Vulcano")	Intra (pipeline)	OLAP
HyPer / Umbra	**JIT-LLVM / Pipelined**	**Push**	**Intra (pipeline),** Inter?	**HTAP**
Hyrise	Materialized (lazy)	Push	Intra (pipeline)	OLAP
Redshift	**JIT-C++ / Pipelined +** Vectorized?	**Push**	**Intra (pipeline)**	**HTAP**
HANA / HEX	**JIT-L / Pipelined**	**Push**	**Intra (pipeline)**	**HTAP**

- → HEX State-of-the-art engine for HTAP (see table)
  - → Workloads: transactional applications (e.g., S4/HANA), analytical queries (e.g., Data Warehouse Cloud)
  - → Data chunks
  - → JIT-L pipelined
- → Data-centric code generation in L (LLVM convenience layer)
  - → L used also for, e.g., stored procedures
  - → Supportability: debugging, profiling L programs on tooling level; portability
- → Extensible: New physical operators can be added to HEX (e.g., application- / service-specific)
- → TCO, Price / Performance
  - → Reduce memory footprint: pipelining and streaming, fewer engines (reduce intermediate result materialization)
  - → More CPU-efficient due to JIT compilation
  - → Performance same or slightly better
- → Distributed query processing (send, receive)
- → Intra pipeline parallelization (dynamic)

- Code generation based on Neumann, Thomas. "Efficiently compiling efficient query plans for modern hardware." *Proceedings of the VLDB Endowment* 4.9 (2011): 539-550.
- Leis, Viktor, et al. "Morsel-driven parallelism: A NUMA-aware query evaluation framework for the many-core age." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014.
- Raasveldt, Mark, and Mühleisen, Hannes. "DuckDB: an embeddable analytical database." *Proceedings of the 2019 International Conference on Management of Data*. 2019.

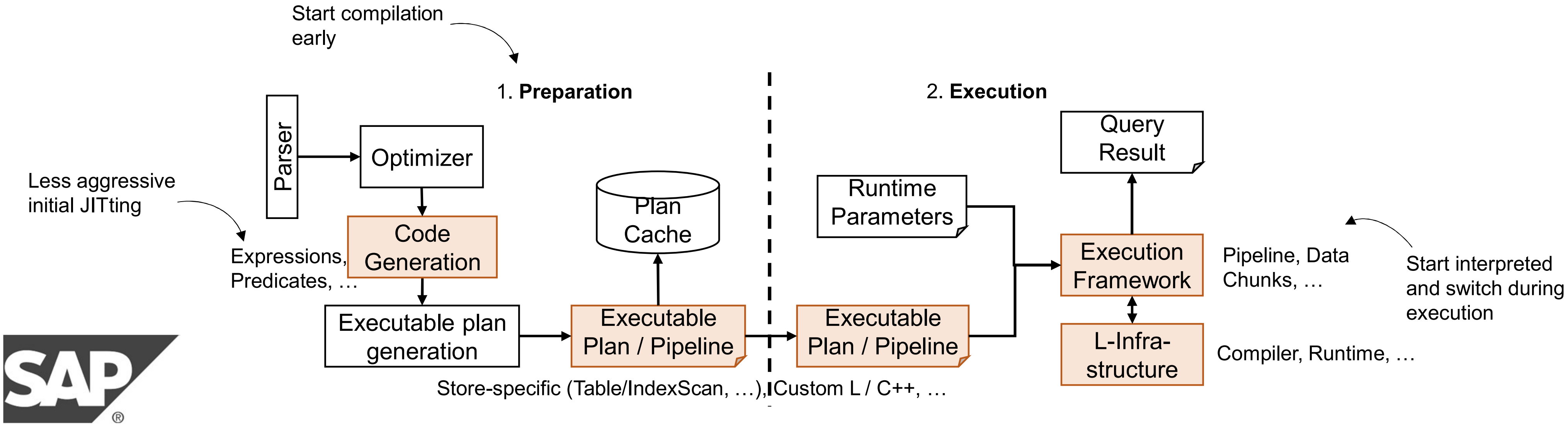
Execution Phases

→ In practice

- → Works well / no issues for OLTP queries with plan caching
- → JIT compilation times challenging for large and complex analytical queries during cold start

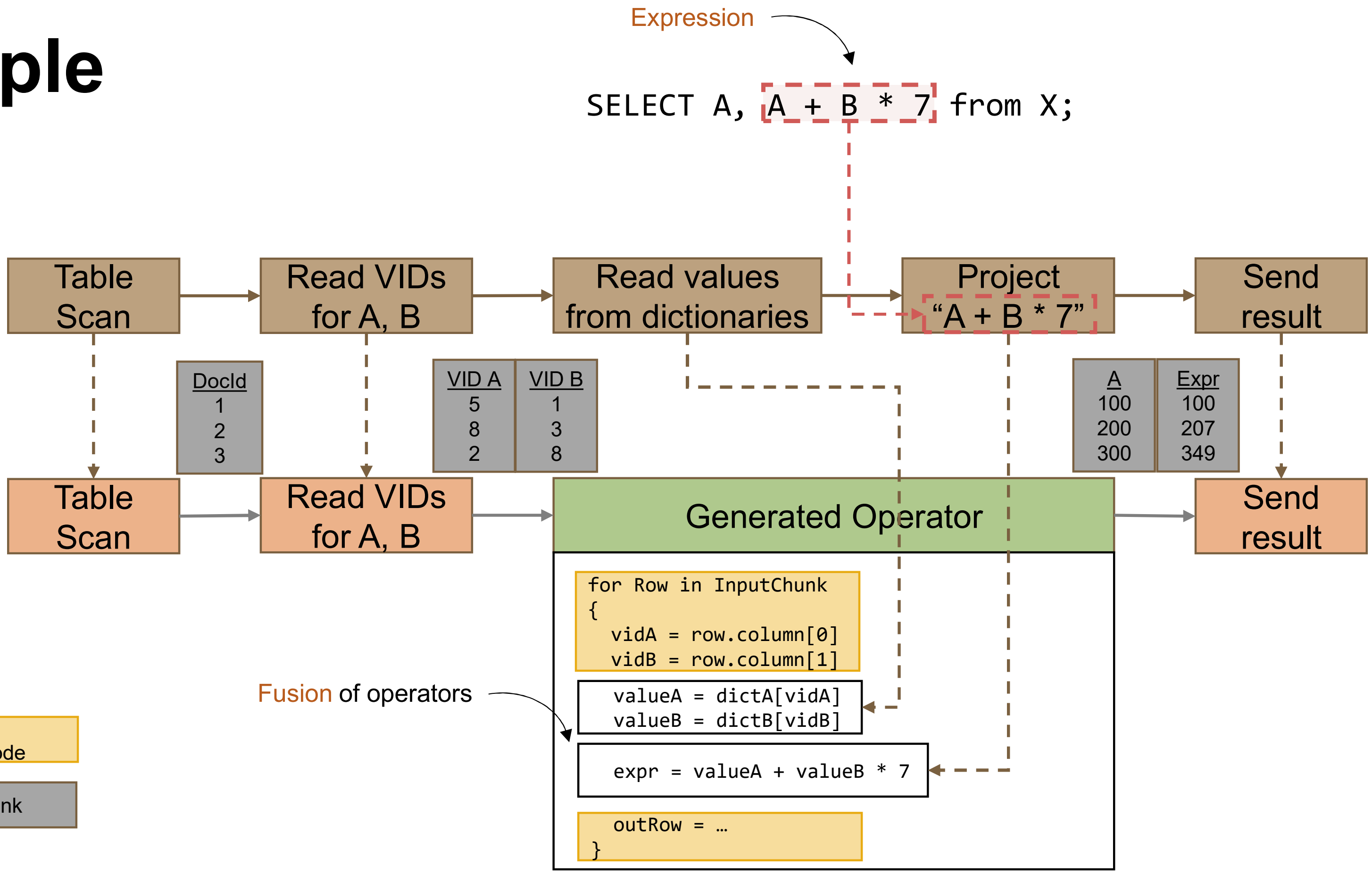
→ Mitigate JIT compilation times

- → Start interpreted / uncompiled, compile in background per query / L program (fragment)
- → Switch to compilation after third execution


Start compilation early
1. Preparation
2. Execution
Parser
Optimizer
Less aggressive initial JITting
Expressions, Predicates, …
Code Generation
Executable plan generation
Plan Cache
Executable Plan / Pipeline
Executable Plan / Pipeline
Store-specific (Table/IndexScan, …),
Custom L / C++, …
Runtime Parameters
Query Result
Execution Framework
L-Infra-structure
Pipeline, Data Chunks, …
Compiler, Runtime, …
Start interpreted and switch during execution

SAP SE

Example



```
SELECT A, A + B * 7 from X;
```

DocId
1
2
3

VID A	VID B
5	1
8	3
2	8

A	Expr
100	100
200	207
300	349

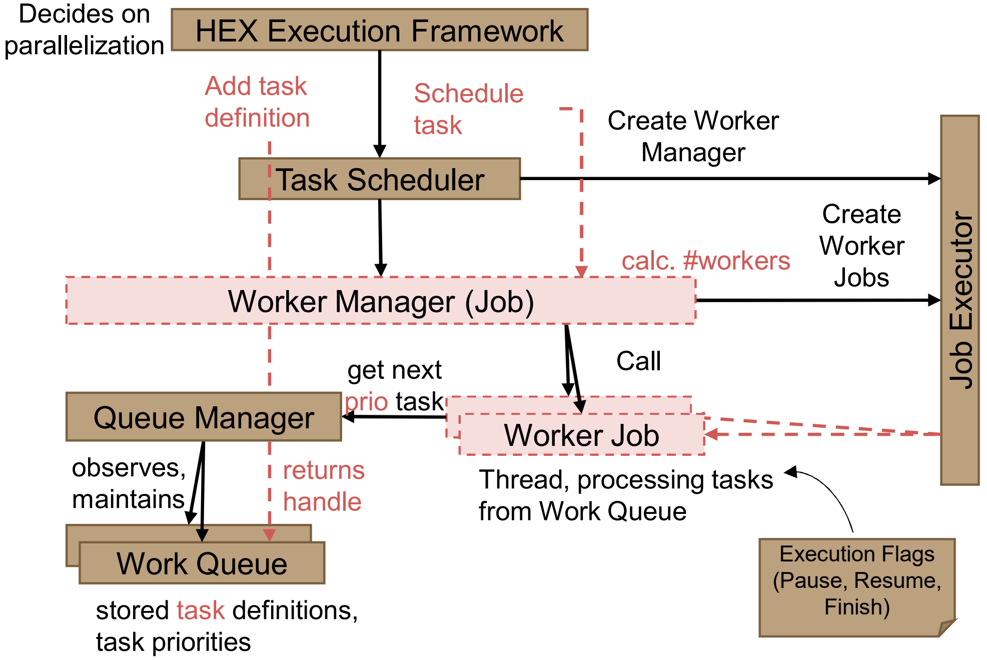
```
for Row in InputChunk
{
  vidA = row.column[0]
  vidB = row.column[1]
  valueA = dictA[vidA]
  valueB = dictB[vidB]

  expr = valueA + valueB * 7

  outRow = …
}
```

- 🟧 Precompiled Operator
- 🟨 Framework Generated Code
- 🟩 Generated Operator
- ⬜ Data Chunk

sap.com

#sap

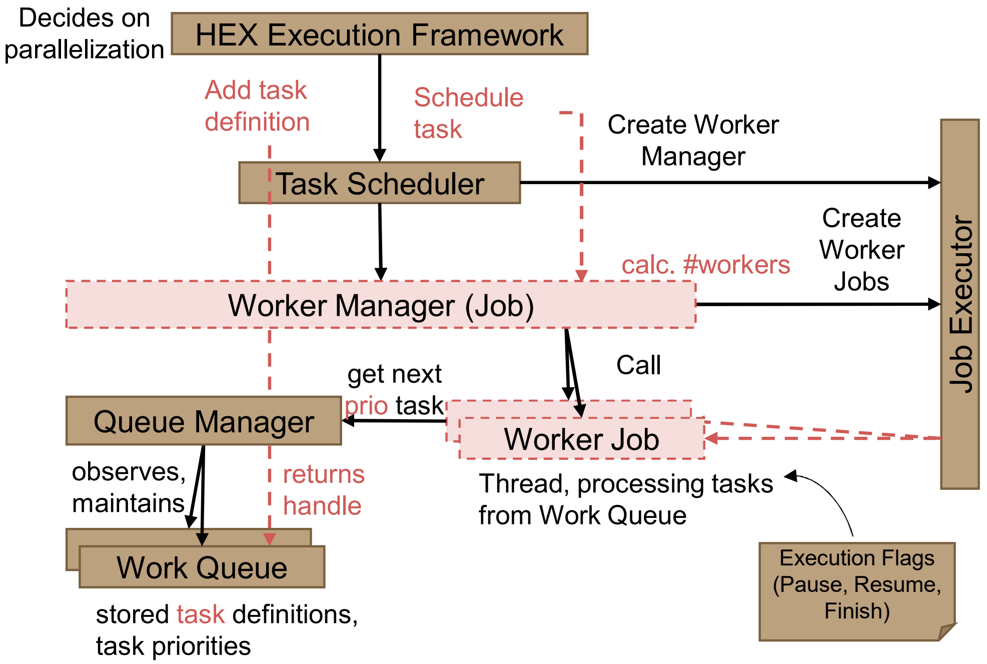
Intra Pipeline Parallelization



- → Pipelining: better memory access pattern (less cache misses) and no full materialization between operators (lower memory footprint)
- → Parallelization with pipelining more complicated
  - → Parallelize operators instead of data → determining task size complicated: fixed task size → skewed workload
  - → Parallelization requires (expensive) scheduling → bigger tasks sizes preferrable, BUT due to skewed workloads → fine-grained tasks

SAP SE

Intra Pipeline Parallelization



Decides on parallelization

HEX Execution Framework

Add task definition

Schedule task

Create Worker Manager

Task Scheduler

Create Worker Jobs

calc. #workers

Worker Manager (Job)

Job Executor

get next prio task

Call

Queue Manager

Worker Job

observes, maintains

returns handle

Thread, processing tasks from Work Queue

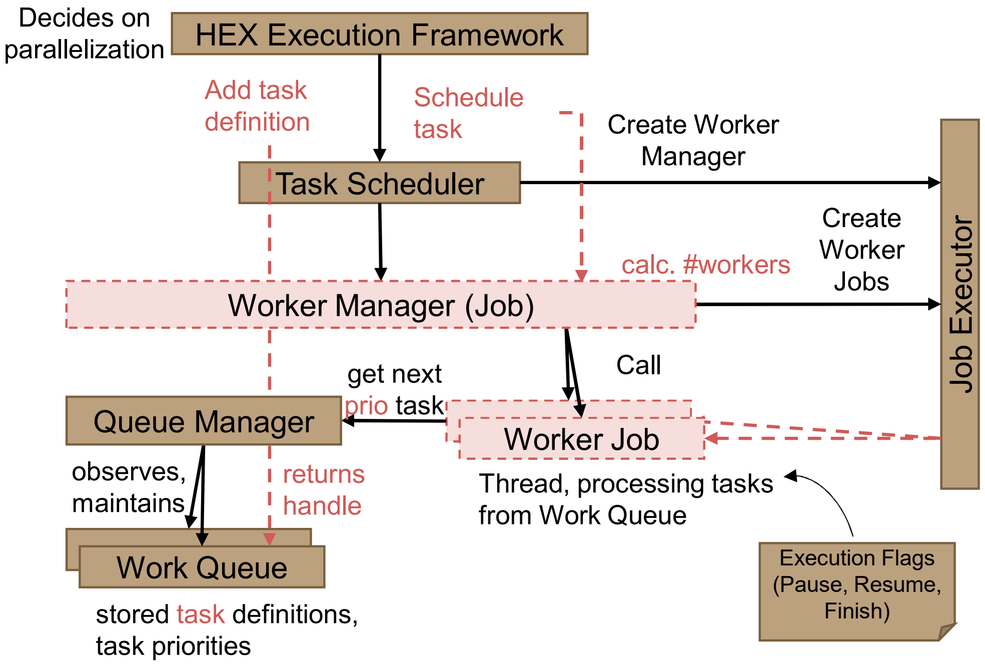
Work Queue

Execution Flags (Pause, Resume, Finish)

stored task definitions, task priorities

- → Pipelining: better memory access pattern (less cache misses) and no full materialization between operators (lower memory footprint)
- → Parallelization with pipelining more complicated
  - → Parallelize operators instead of data → determining task size complicated: fixed task size → skewed workload
  - → Parallelization requires (expensive) scheduling → bigger tasks sizes preferrable, BUT due to skewed workloads → fine-grained tasks
- → Reduce / tame job creation overhead / scheduling:
  - → HEX task scheduling integrated in HANA job scheduling
  - → Map several tasks (possibly of different kind) to one job (pooled)
  - → Job will live longer than task → less job creation overhead

Intra Pipeline Parallelization



Decides on parallelization

HEX Execution Framework

Add task definition

Schedule task

Create Worker Manager

Task Scheduler

Create Worker Jobs

calc. #workers

Worker Manager (Job)

Job Executor

get next prio task

Call

Queue Manager

Worker Job

observes, maintains

returns handle

Thread, processing tasks from Work Queue

Work Queue

Execution Flags (Pause, Resume, Finish)

stored task definitions, task priorities

- → Pipelining: better memory access pattern (less cache misses) and no full materialization between operators (lower memory footprint)
- → Parallelization with pipelining more complicated
  - → Parallelize operators instead of data → determining task size complicated: fixed task size → skewed workload
  - → Parallelization requires (expensive) scheduling → bigger tasks sizes preferrable, BUT due to skewed workloads → fine-grained tasks
- → Reduce / tame job creation overhead / scheduling:
  - → HEX task scheduling integrated in HANA job scheduling
  - → Map several tasks (possibly of different kind) to one job (pooled)
  - → Job will live longer than task → less job creation overhead
- → Address workload skew: sampling / re-parallelization
  - → `Worker Manager` checks the `Queue Manager` regularly to calculate progress and creates more workers, if needed > #workers dynamic
  - → Sampling phase decides if parallelization is needed + size of tasks
  - → Intermediate scheduling operators measure elapsed time to execute remaining pipeline (e.g., after selective / expanding joins, selective table scans) + find new, good task size
- → Sampling not for free due to scheduling points
  - → Are sync. points > too many lead to fluctuations between runs
  - → Break operator fusion

Challenges and Opportunities

…

Remove old engines "in-flight" without disruptions: no functional or performance regressions

State-of-the-art, compiled, pipelined query engine with extensible architecture

Multi-Model engines in HEX, nested file formats, …

Multimodel Data Platforms, Forrester, Q3 2021: https://www.sap.com/cmp/dg/forresterwave-mmdp/index.html

SAP SE

sap.com

#sap

Join us later at CIDR:

- Tuesday 4:50 pm: **Data Pipes: Declarative Control over Data Movement** Lukas Vogel (Technische Universität München); Daniel Ritter (SAP); Danica Porobic (Oracle); Pinar Tozun (IT University of Copenhagen)*; Tianzheng Wang (Simon Fraser University); Alberto Lerner (University of Fribourg)
- Wednesday 11:10 am: **DASH: Asynchronous Hardware Data Processing Services** Norman May (SAP SE)*; Daniel Ritter (SAP); Andre Dossinger (SAP SE); Christian Faerber (Intel Corporation); Suleyman Demirsoy (Intel Corporation)

Ph.D. position available!

Special thanks go to our academic and industrial collaboration partners as part of the SAP HANA Research Campus!

SAP SE

Thank you!

Contact information:

Daniel Ritter
E-Mail: daniel.ritter@sap.com
HEX-Blog: https://blogs.sap.com/2023/01/05/faster-query-execution-using-lesser-memory-in-sap-hana-cloud/

sap.com

#sap