An Adaptive Query Execution Engine for Data Integration

Zachary Ives, Daniela Florescu, Marc Friedman, Alon Levy, Daniel S. Weld

University of Washington

Slides by Peng Li, Modified by Rachel Pottinger

Presentation: Jonas Tai

Discussion: Dorna Dehghani

What makes Data Integration Systems (DIS) more challenging than traditional DB Systems?

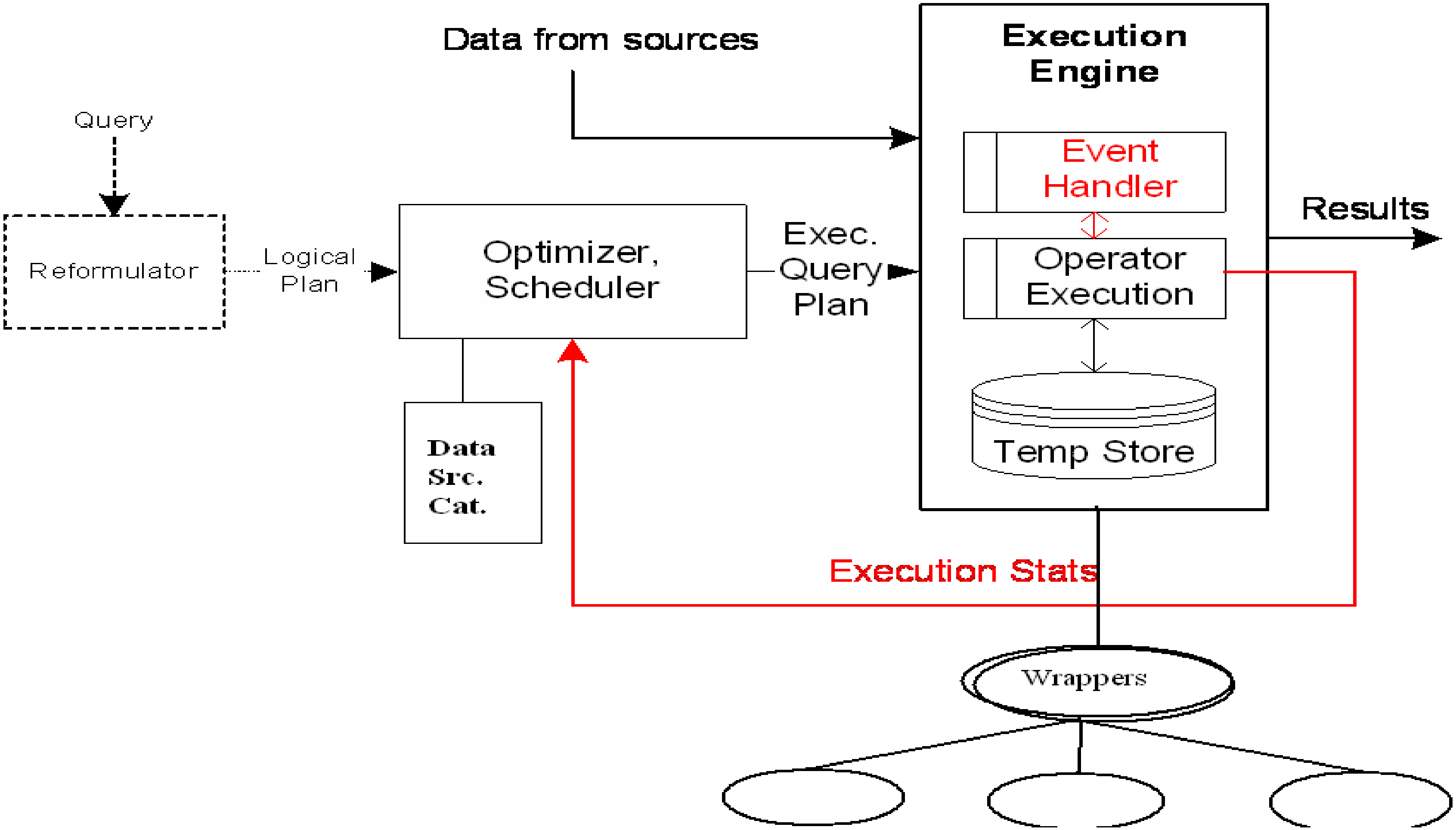
- Query Reformulation
- The construction of wrapper programs
- Query optimizers and efficient query execution engines

Characteristics of DISs

- Unreliable or missing data statistics
- Unpredictable data transfer rates
- Unreliable, overlapping sources
- Want initial results quickly
- Network bandwidth generally constrains the data sources to be “small”

System needs to be adaptive

Tukwila Architecture



Adaptivity through interleaving of planning and execution

Novel characteristics of Tukwila:

- The optimizer can create a partial plan if essential statistics are missing or uncertain
- The optimizer generates operator trees ***and*** appropriate event-condition-action rules
- Optimizer conserves the state of its search space when it calls the execution engine
- Dynamic collector operator to work with multiple sources

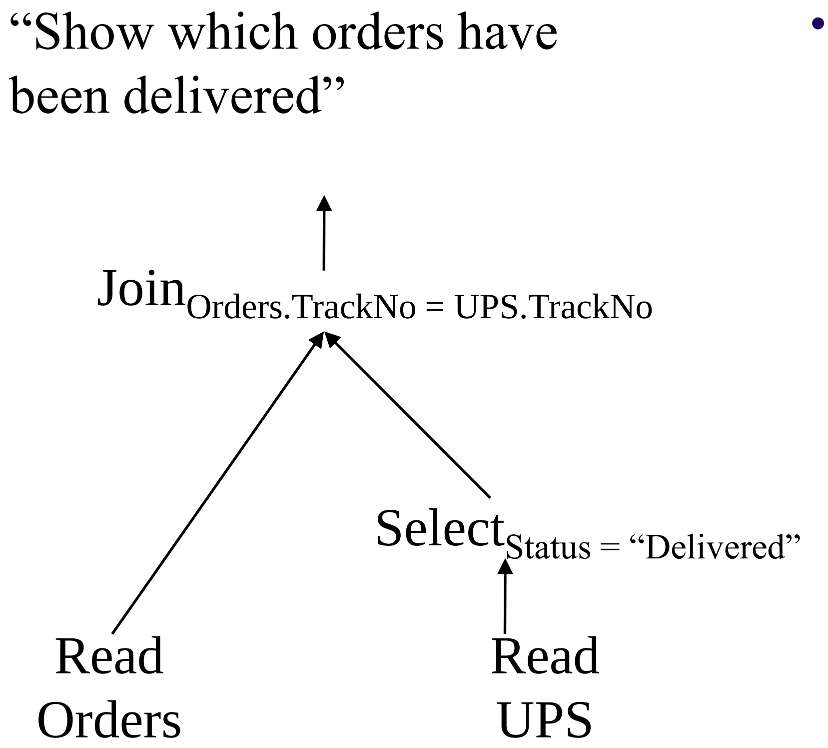
Discussion 1: Overlapping data sources

Tukwila handles data sources with overlaps, though the paper doesn't delve into the reasons for its significance. What challenges might arise due to overlapping data, and what are potential approaches to address them?

Discuss in groups of two.

Query Plan Execution

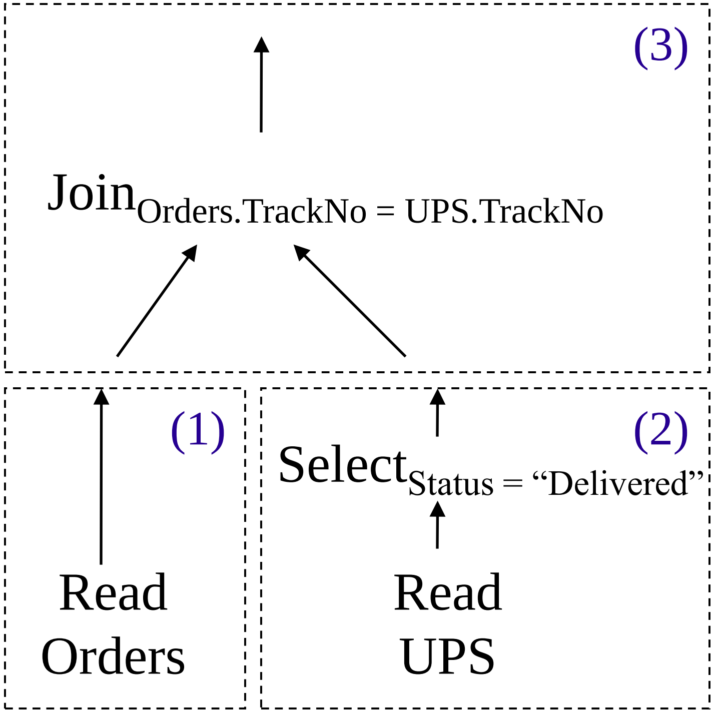
Query plan represented as data-flow tree:



- Control flow
  - **Iterator** (top-down)
    - Used by Tukwila
  - **Data-driven** (bottom-up)

Tukwila Query Plan Structure

- A *plan* includes a partially-ordered set of *fragments* and a set of *global rules*
- A *fragment* consists of a fully pipelined tree of *physical operators* and a *set of local rules*.
- Key mechanism for adaptivity: At the end of each fragment, the rest of the plan can be re-optimized or rescheduled



Implementation of Adaptivity: Rules

- **Reoptimization**
  - Reinvoke optimizer if cardinality estimates of results differ significantly
- **Contingent planning**
  - The execution engine checks properties of the result to select the next fragment
- **Rescheduling**
  - Reschedule if a source times out
- **Adaptive operators**

Tukwila Plans & Execution

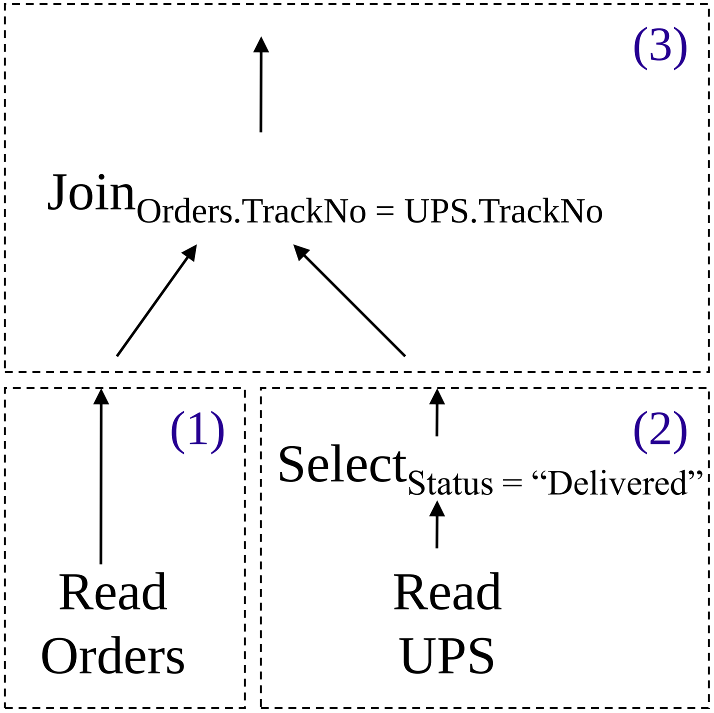
When *event* if *condition* then *actions*

When *closed(frag1)*

if *card(join1)>2*est_card(join1)*

then *replan*

On event trigger, the rules are checked



When(closed(1)):
if size_of(Orders) > 1000
then reoptimize {2, 3}

Discussion 2: Tukwila's motivations

Consider one of the following motivating situations of Tukwila:

1. Absence of statistics
2. Unpredictable data arrival characteristics
3. Overlap and redundancy among sources
4. Optimizing the time to initial answers

Answer these questions:

Q1: Can you give some examples where the chosen topic matters?

Q2: If you are a member of Tukwila team, what rules or policy would you have to deal with the problem?

Discuss in groups of 4, half system. half non-system.

Adaptive Query Operators
Why not conventional Joins?

Unpredictable Data Transfers

- Sort merge joins & indexed joins
  - Slow transfer blocks execution
- Nested loops joins and hash joins
  - Slow transfer of inner relationship, hard to choose inner relationship

Solution: Double Pipelined Hash Join

Double Pipelined Hash Join

- Proposed for parallel main-memory databases (Wilschut 1990)
  - One hash table per source
  - Add to hash table and probe opposite table as stream

Advantages:

- Data-driven: Results as soon as tuples are received
- Symmetric
- Can process faster data source while waiting for the slower source

Example

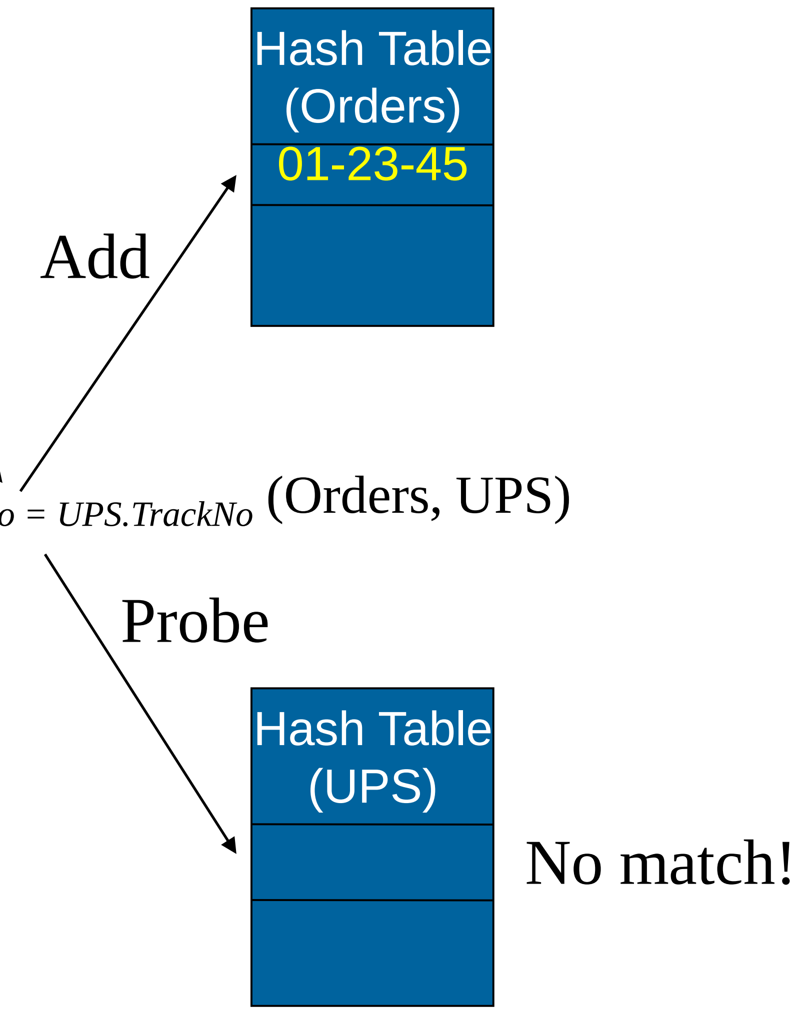
Orders

OrderNo	TrackNo
1234	01-23-45
1235	02-90-85
1399	02-90-85
……	……

$$Join_{Orders.TrackNo\ =\ UPS.TrackNo}\ \text{(Orders, UPS)}$$

UPS

TrackNo	Status
01-23-45	In Transit
02-90-85	Delivered
03-99-10	Delivered
……	……



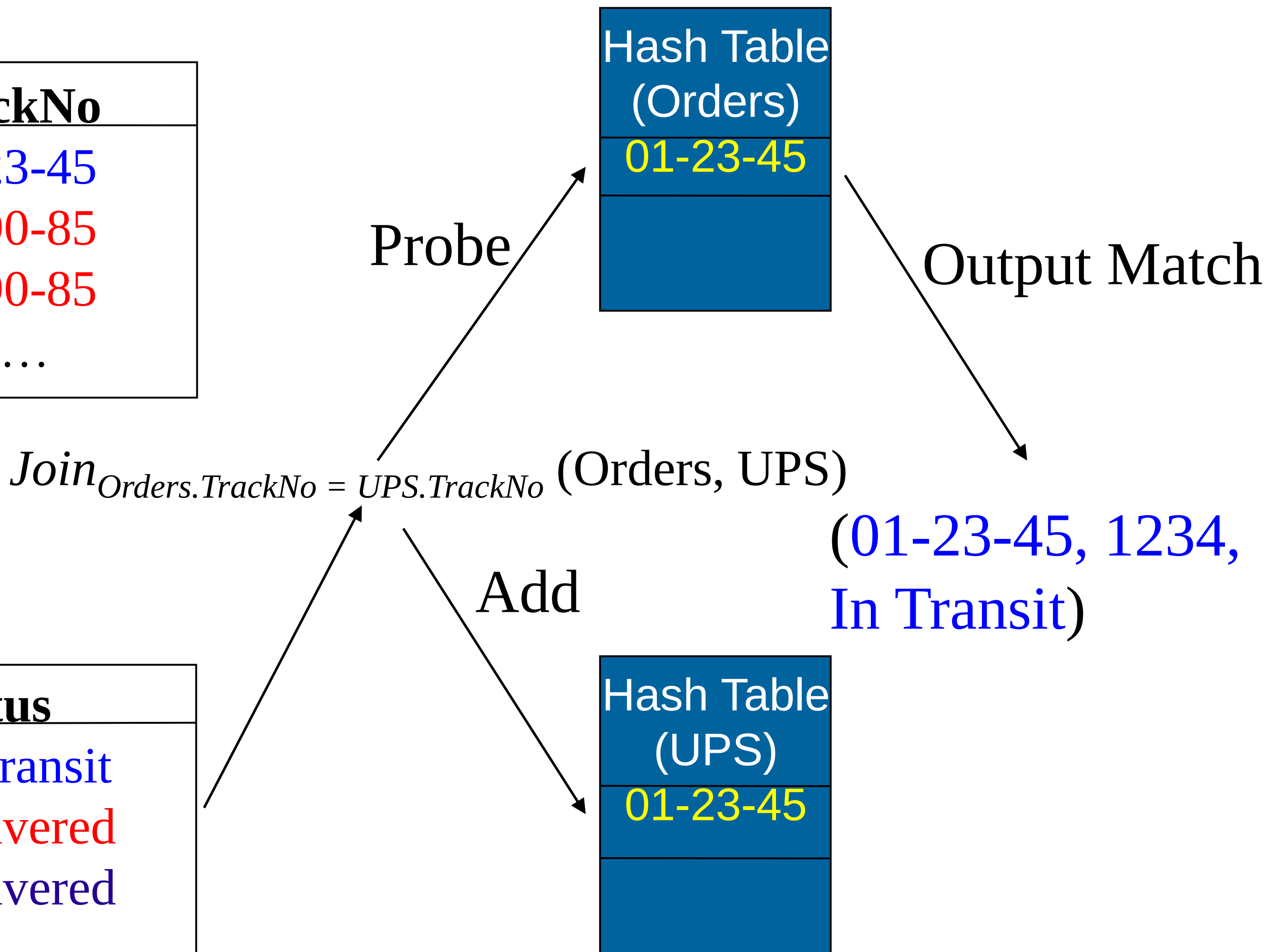
Example

Orders

OrderNo	TrackNo
1234	01-23-45
1235	02-90-85
1399	02-90-85
……	……

UPS

TrackNo	Status
01-23-45	In Transit
02-90-85	Delivered
03-99-10	Delivered
……	……



Memory Overflow

- **Memory heavy**: May not be able to fit both hash tables in RAM
- Strategies
  - **Incremental Left Flush:** Slowly degrade to hybrid hash join by flushing buckets of one source
  - **Incremental Symmetrical Flush:** Flush corresponding buckets of both sources
  - Set strategy through rules

Discussion 3: Tukwila

Would the adaptive behaviour of Tukwila be beneficial in general database systems? Would it boost efficiency?

What could be some advantages and disadvantages of applying the same methods to general database systems?

Discuss in groups of 2.

Summary

- Modified query optimizer to account for the unique characteristics of DISs
- Adaptivity through interleaving of optimization and execution
- Event-condition-action rule system
- Adaptive double pipelined hash join and collector