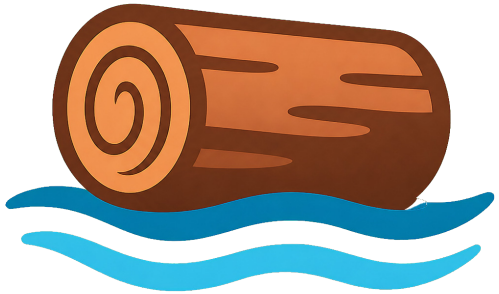


VLDB 2026

FlowLog

Efficient and Extensible Datalog via Incrementality

Hangdong Zhao[1] · Zhenghong Yu[2] · Srinag Rao[2] · Simon Frisk[2] · Zhiwei Fan[3] · Paraschos Koutris[2]

1 Microsoft Microsoft Azure Data Gray Systems Lab

2 WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

3 Meta

Talk Overview

01 Motivation

02 System Design

03 Evaluation

04 Beyond the Paper

01 Motivation

Datalog

A declarative logic programming language

ADVANTAGES

Declarative

Recursive

APPLICATIONS

Program analysis
Doop · Polonius (legacy) · DDISASM

Knowledge graphs
RDFox

Network control
OVN northd (legacy)

Distributed systems
Paxos · 2PC

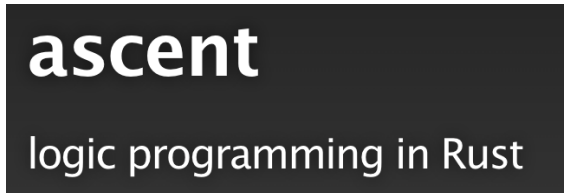
Existing Approach I: Build from Scratch

✓ **ADVANTAGES**

1. Fine-grained control over evaluation
2. Datalog-specific optimizations

× **LIMITATIONS**

1. New features require invasive changes
2. Incrementality and scale-out require major engineering



Existing Approach II: Reuse a Data System

✓ **ADVANTAGES**

1. Reuse mature execution primitives
2. Inherit parallelism or incrementality

× **LIMITATIONS**

1. Datalog-aware optimization becomes difficult
2. Runtime costs are dictated by the substrate

RecStep

DDlog

vmware® Research

FlowLog combines both.

Mature Differential Dataflow[1] primitives

+

Fine-grained, Datalog-aware optimization

[1] McSherry et al. "Differential Dataflow." CIDR 2013.

Differential Dataflow (DD)

A data-parallel streaming framework

Use off-the-shelf stream operators as primitives.

map	reorder or project record fields
filter	retain records matching a predicate
join	combine relations with the same key
reduce	group by key and aggregate values
…	

Efficiency

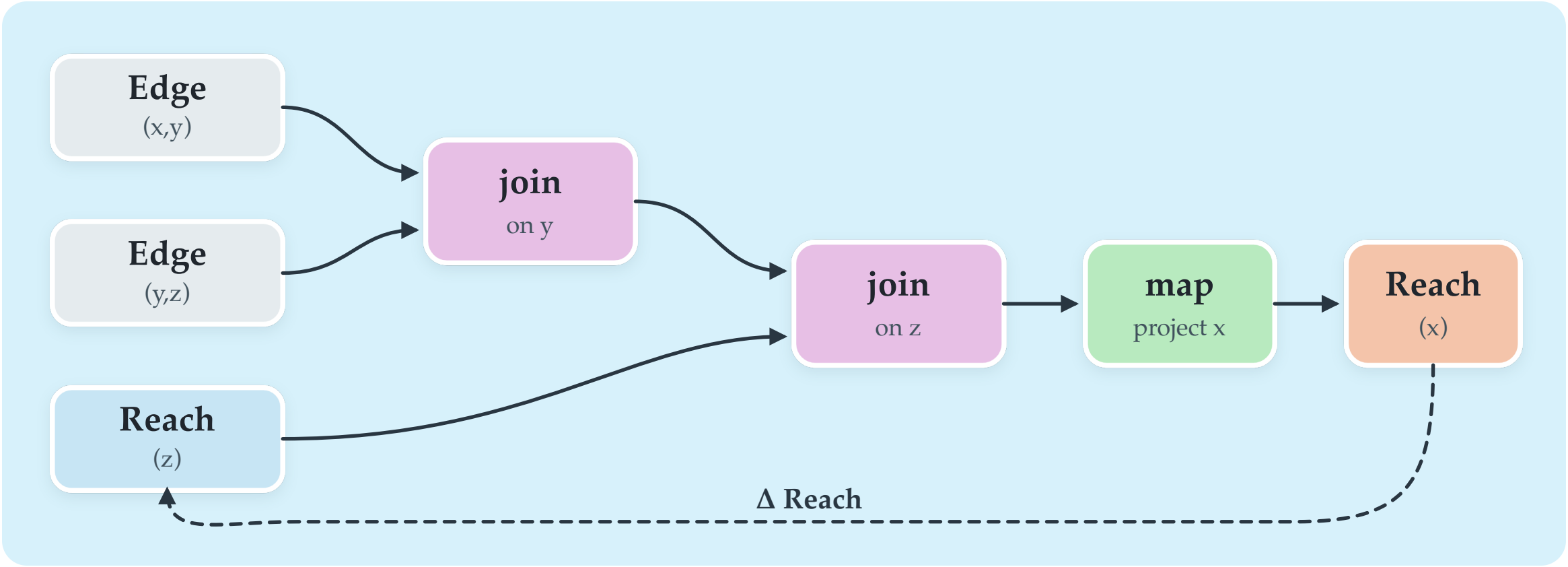
- Reuse work across recursive iterations
- Parallel execution across the dataflow
- Scale up and out

Flexibility

- Extensible operators, including recursive aggregation
- Incremental maintenance as inputs change

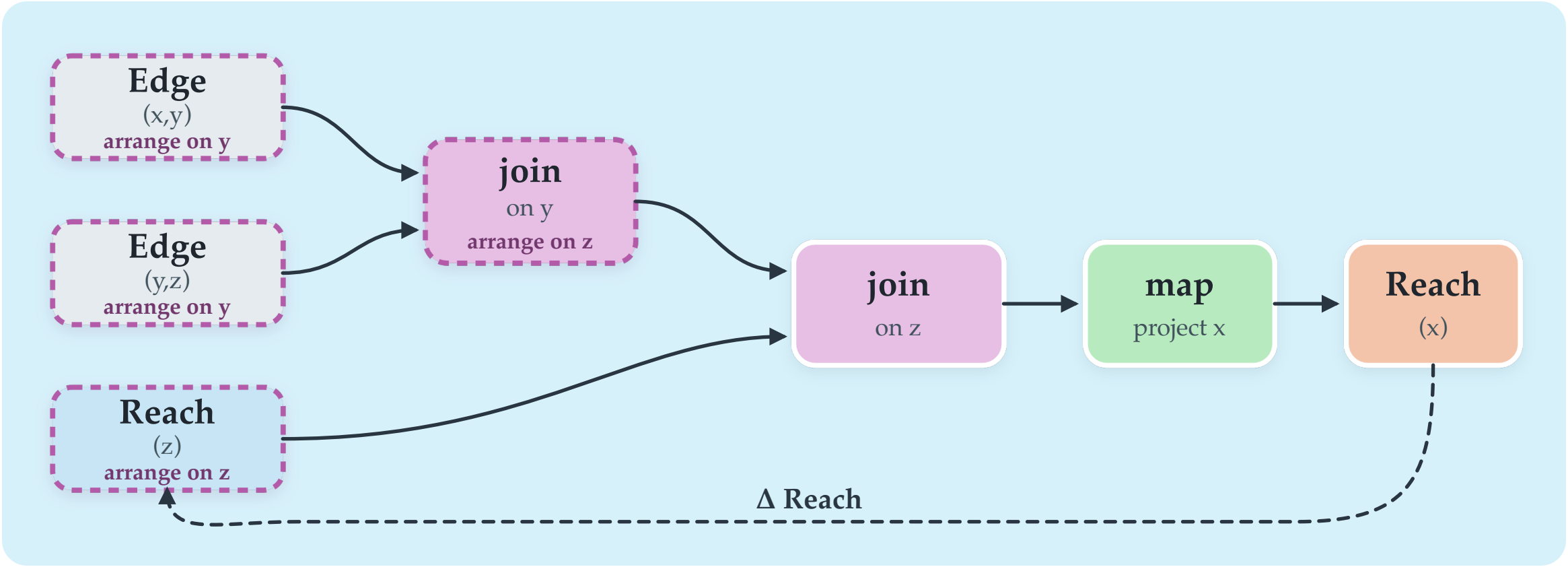
FlowLog on Differential Dataflow

reach(x) :- edge(x,y), edge(y,z), reach(z).



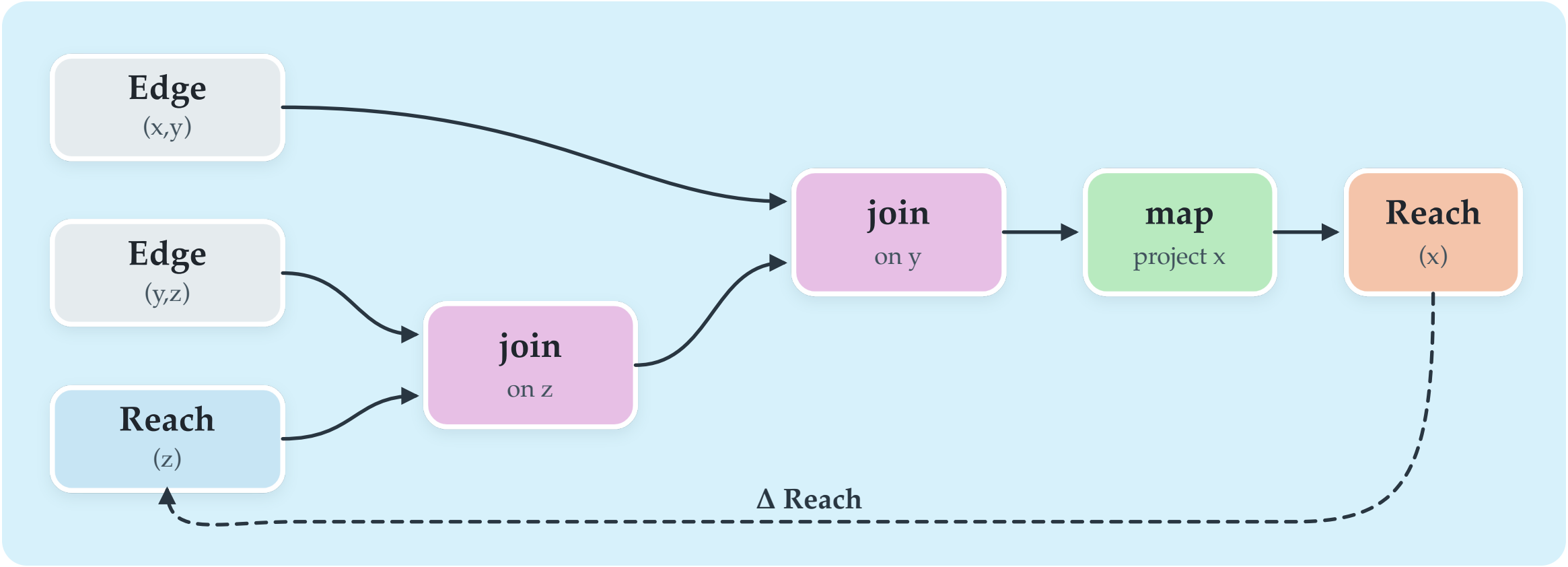
Problem 1: DD is Stateful

Every join input is maintained as a materialized arrangement.



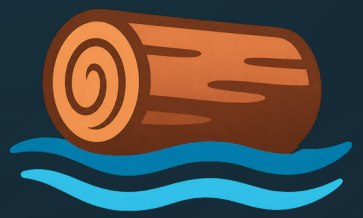
Problem 2: Join Ordering

reach(x) :- edge(x,y), edge(y,z), reach(z).



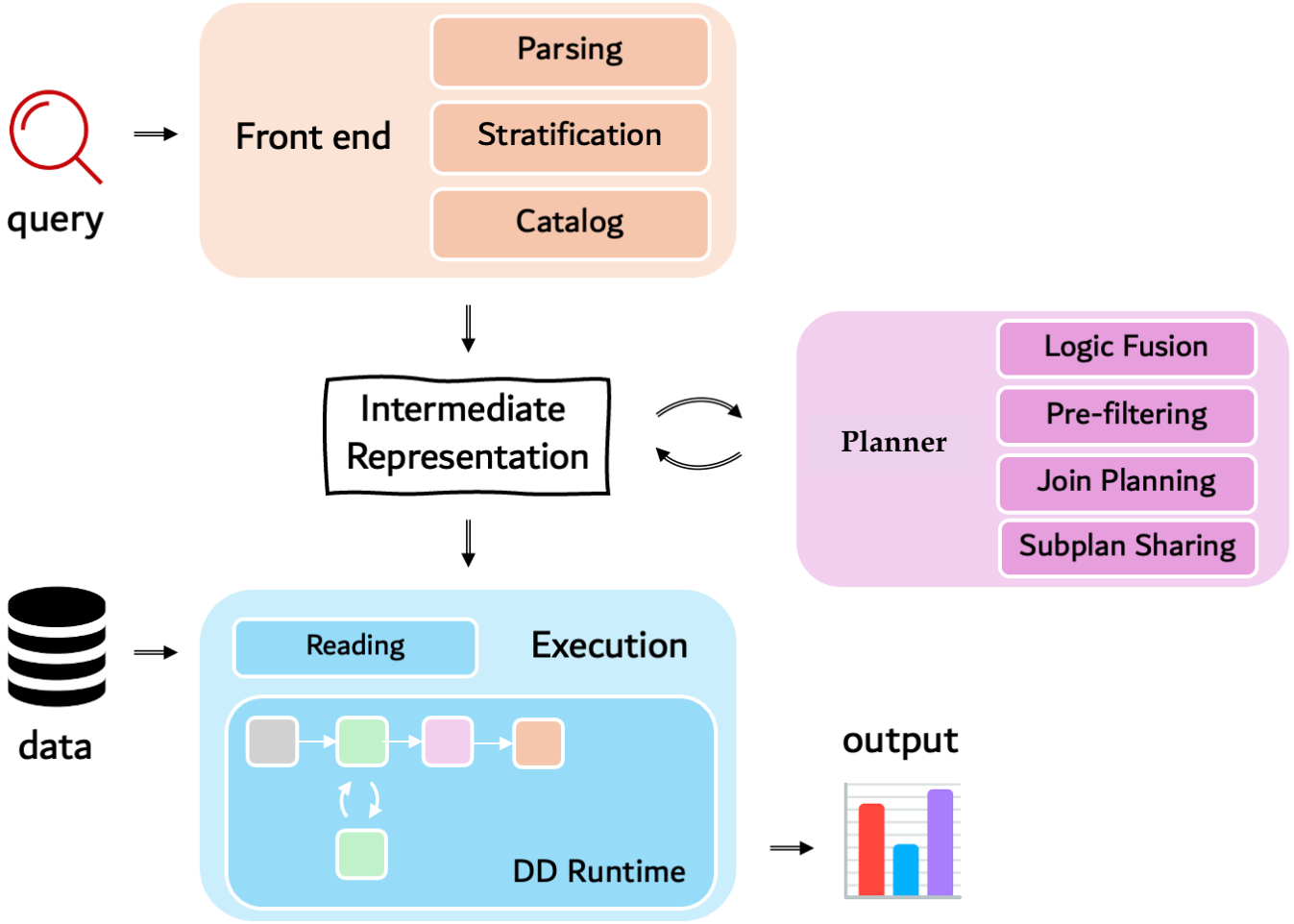
Problem 3: Workloads Need Different Semantics

Ordinary Datalog	**Set**	**reach(x) :- edge(x,y), edge(y,z), reach(z).**
Negation	**Non-monotonic**	**sink(x) :- node(x), !edge(x,_).**
Recursive Aggregation	**Aggregation**	**sssp(y,min(d+w)) :- sssp(x,d), edge(x,y,w).**



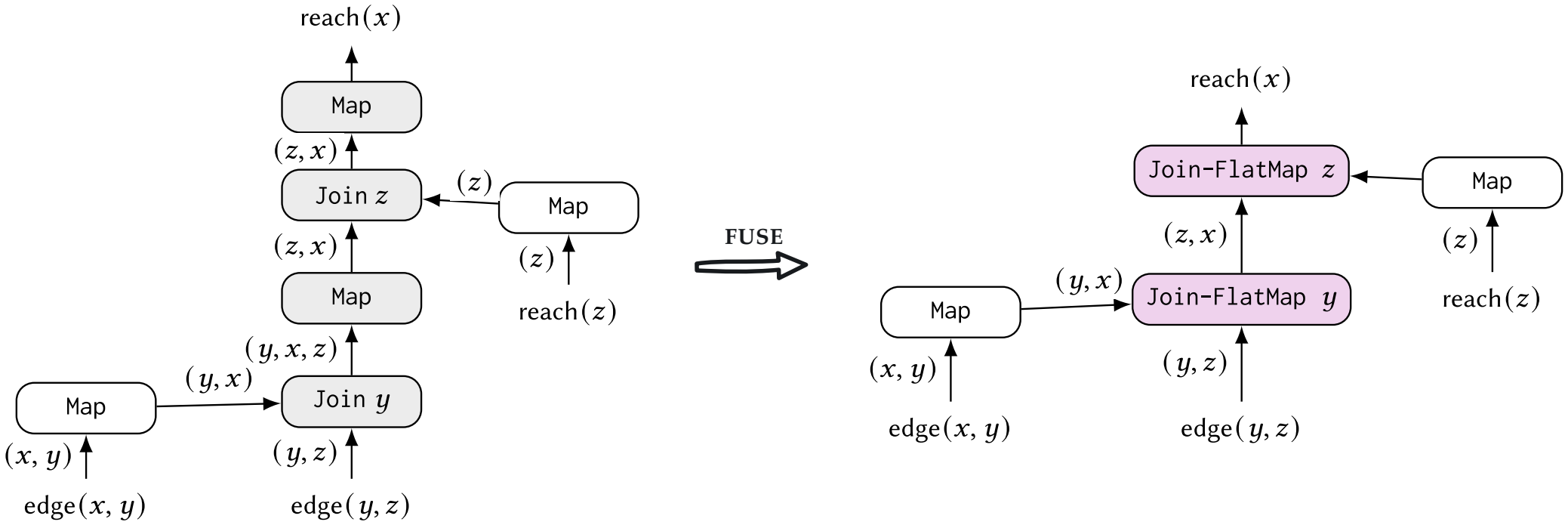
02 System Design

FlowLog System Architecture



Solving Problem 1: Logic Fusion

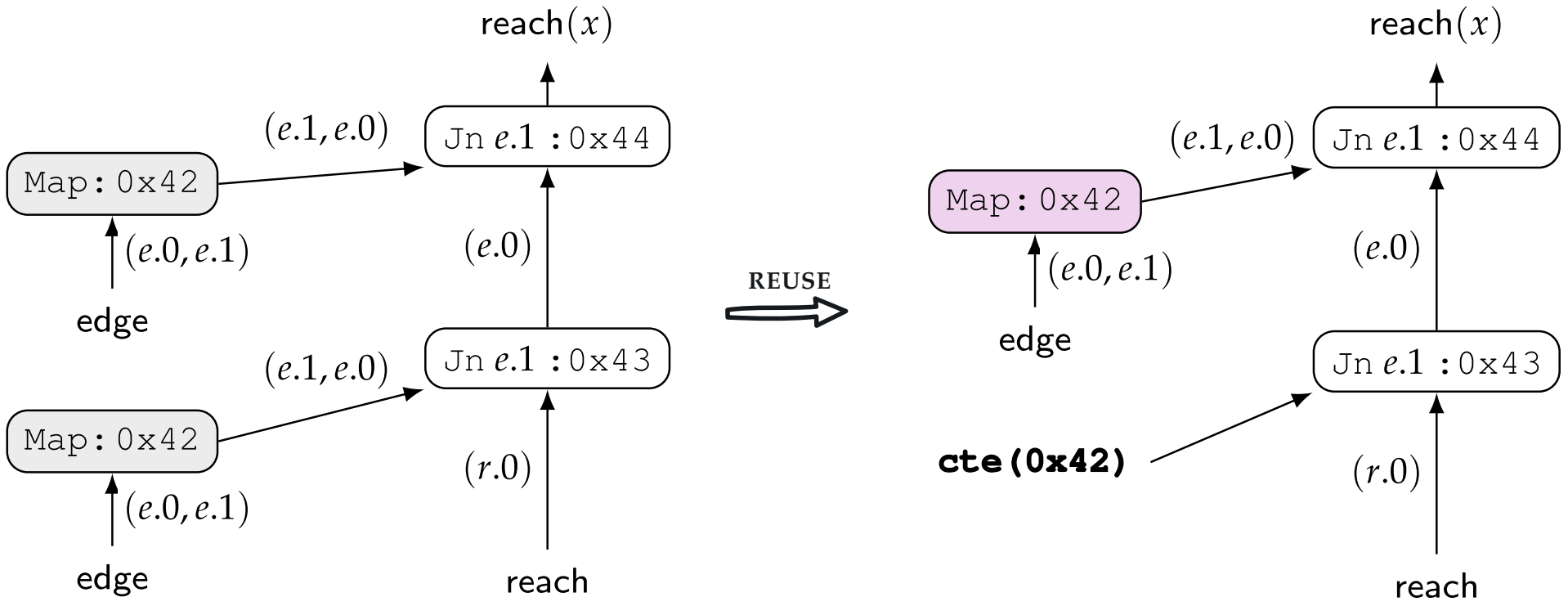
reach(x) :- edge(x,y), edge(y,z), reach(z).



FlowLog paper, Figure 2.

Solving Problem 1: Subplan Sharing

reach(x) :- edge(x,y), edge(y,z), reach(z).



FlowLog paper, Figure 5.

Solving Problem 2: Join Planning

> **DOOP** is a declarative framework for context-sensitive Java points-to analysis.[2]

```
VarPointsTo(heap, to) :-
  Reach(inm),
  LoadArrayIdx(base, to, inm),
  VarPointsTo(bh, base),
  ArrayIdxPointsTo(bh, heap),
  VarType(to, tp),
  HeapAllocationType(bh, bht),
  ComponentType(bht, bct),
  SupertypeOf(tp, bct).
```

Why not use cardinality estimates?

Missing or unstable

Intermediate relations grow during recursion; delta sizes and distributions change every iteration.

Complex search space

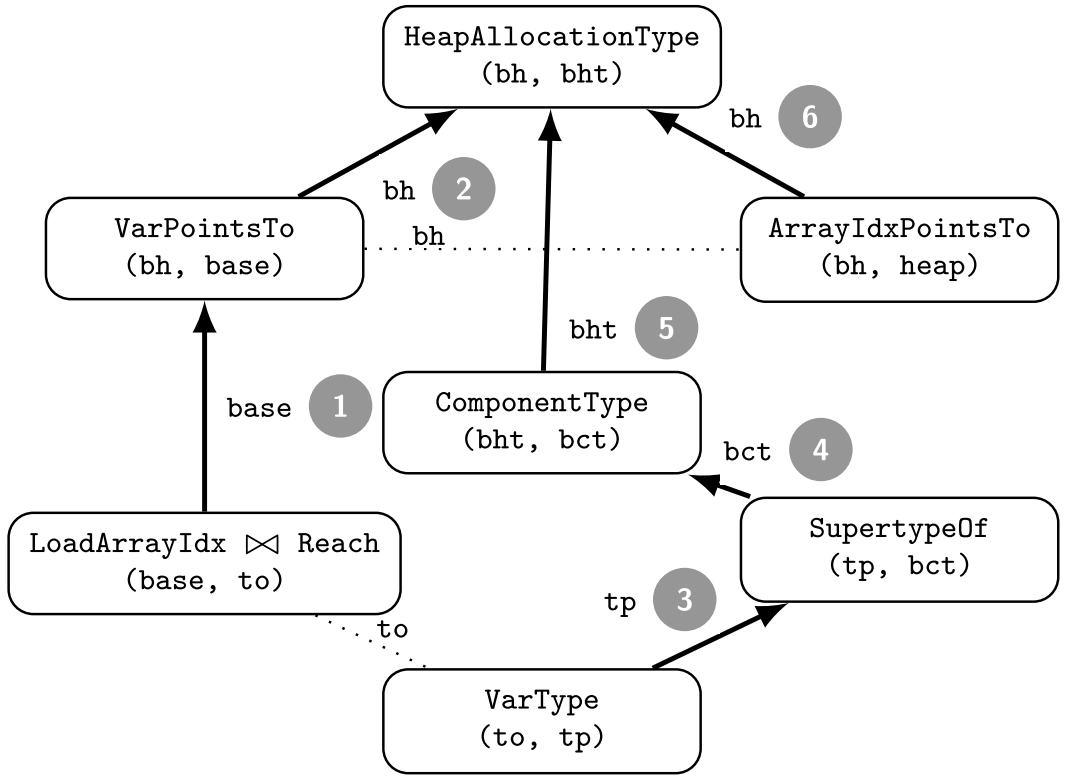
DOOP contains cyclic, multi-way joins—a harder search space than typical DBMS plans.

Runtime CE is costly

Fresh statistics and re-optimization add synchronization and catalog-maintenance overhead.

[2] Bravenboer & Smaragdakis. "Strictly Declarative Specification of Sophisticated Points-to Analyses." OOPSLA 2009.

Solving Problem 2: Join Planning



Every join touches at most 3 variables

$$P^\star = \arg\min_{P} \max_{\tau \in P} |\mathrm{vars}(\tau)|$$

Analyze	Build the weighted join graph.
Search	Enumerate candidate plans.
Select	Minimize worst-case intermediate width.

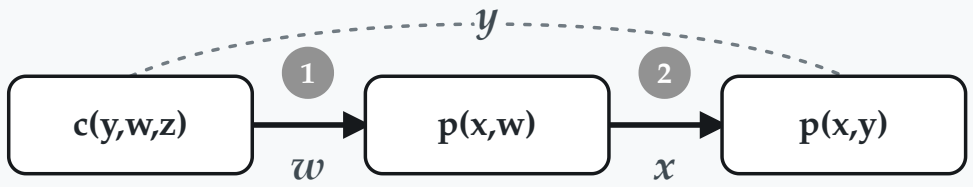
FlowLog paper, Example 5.1 and Figure 4.

Solving Problem 2: Pre-filtering (SIP)

> **Galen** is a medical ontology inference workload.

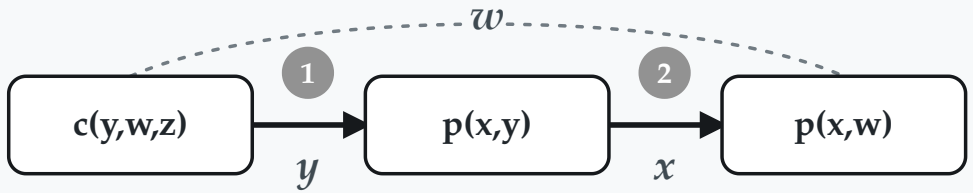
p(x, z) :- c(y, w, z), p(x, w), p(x, y).

JST A — cost 4



faster

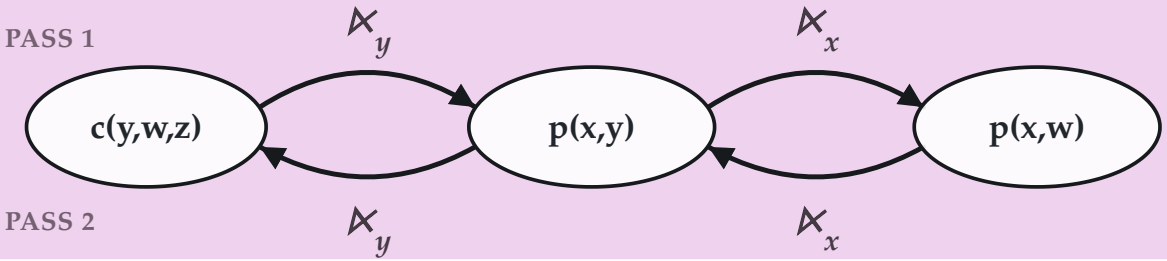
JST B — cost 4



large intermediate blow-up — ≈10× slower

Yannakakis-style pre-filtering[3]

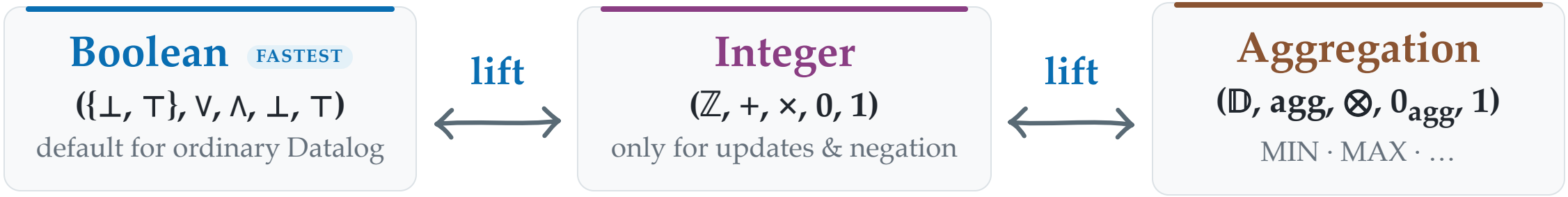
Sideways information passing (SIP) prunes dangling tuples before the actual joins.



FlowLog paper, Example 6.1 (Galen). [3] M. Yannakakis. "Algorithms for Acyclic Database Schemes." VLDB 1981.

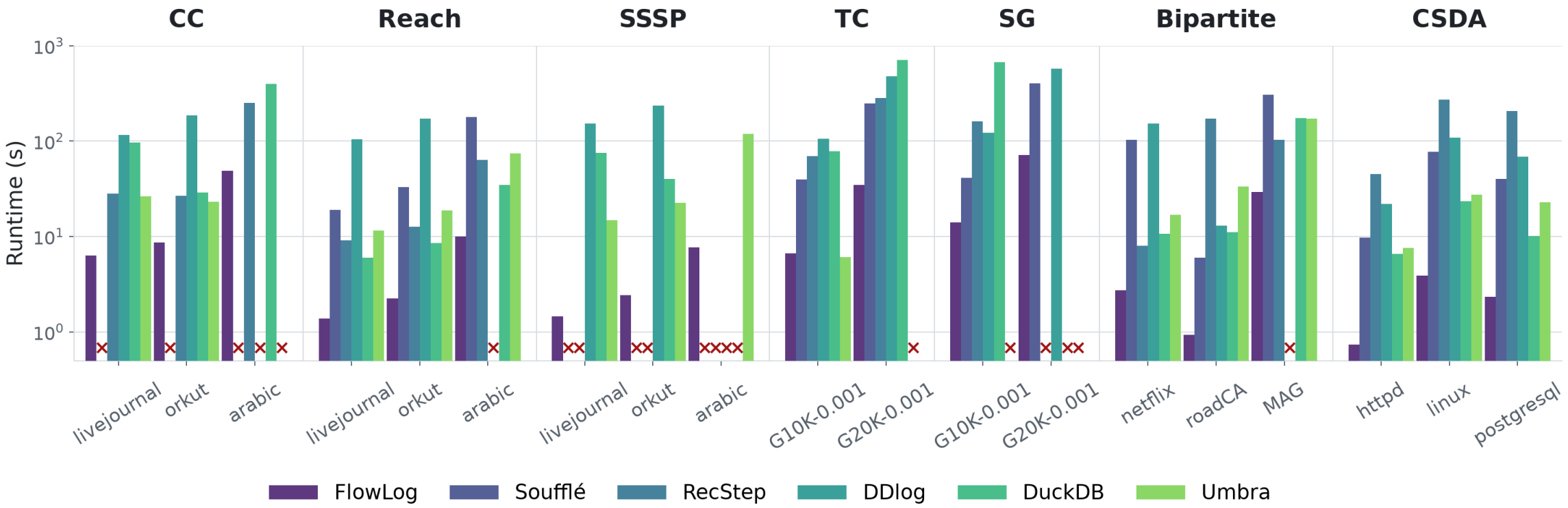
Solving Problem 3: Algebraic Specialization

FlowLog provides pluggable semirings.



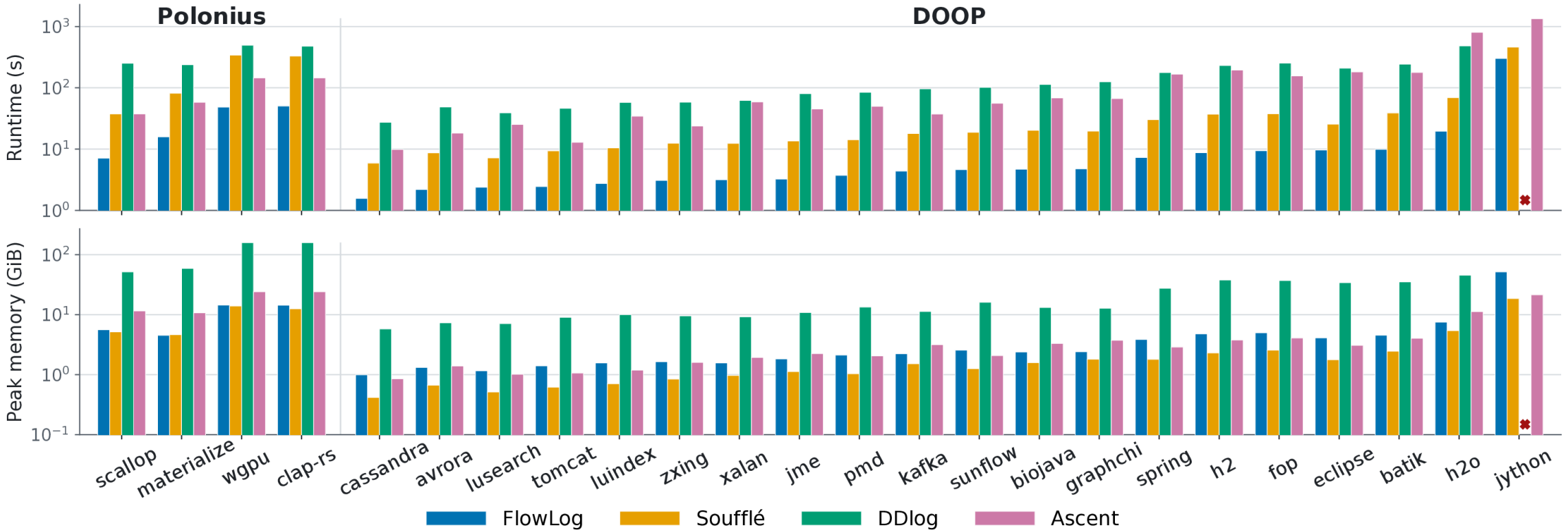
03 Evaluation

Graph & Reasoning



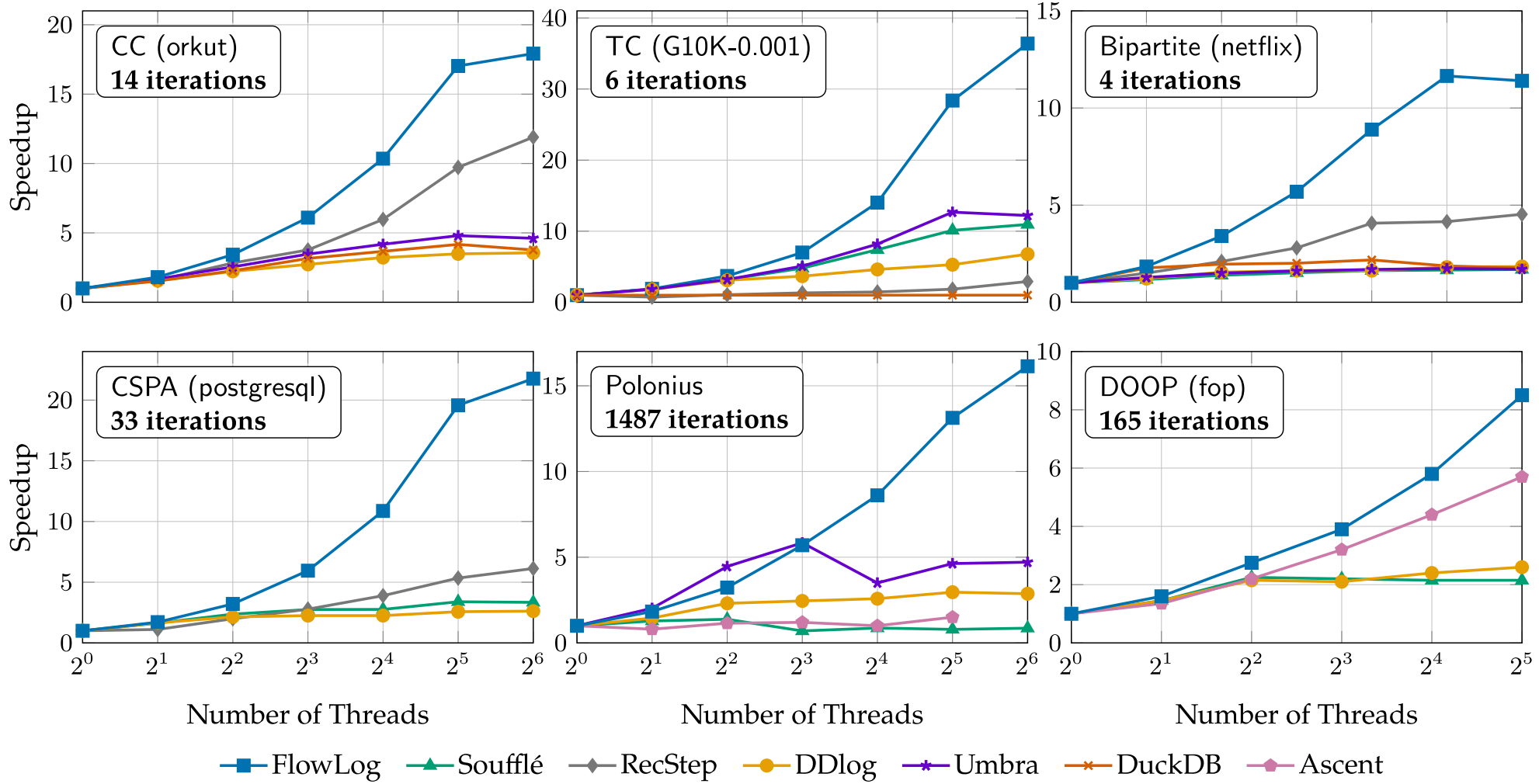
FlowLog: latest rerun medians. Baselines: FlowLog paper, Table 2. Lower is better.

Program Analysis



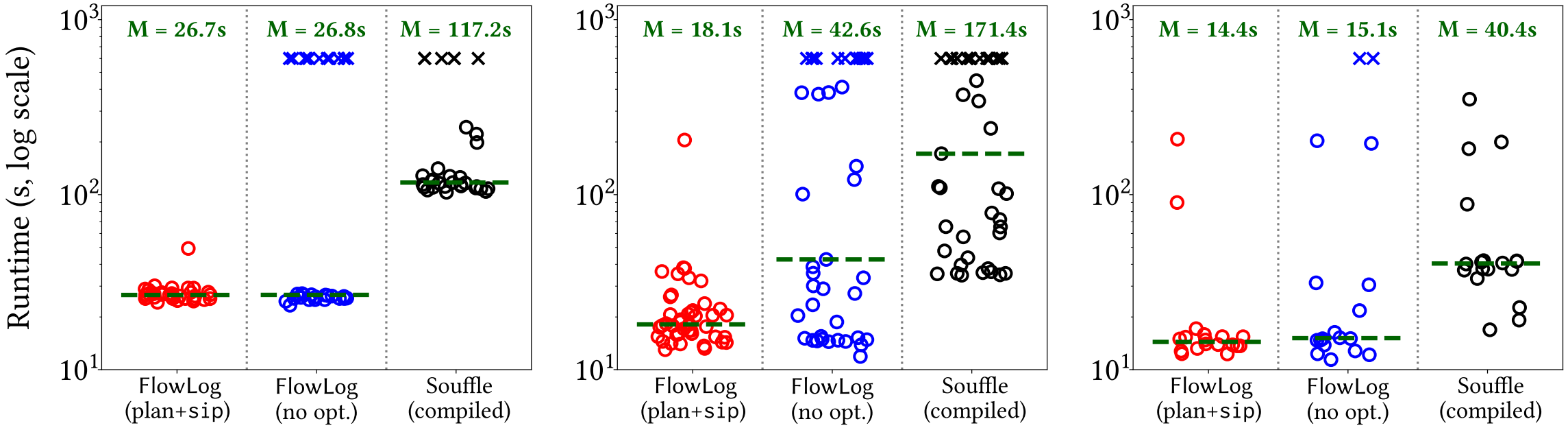
SPLASH evaluation artifact. Lower is better.

Parallel Scalability



FlowLog paper evaluation and SPLASH artifact; selected workloads.

Robustness



(a) DDISASM (z3) on 30 join orders **(b) DOOP (batik) on 43 join orders** **(c) Galen (galen) on 18 join orders**

FlowLog paper, Figure 9.

04 Beyond the Paper

Interactive Incrementality

DOOP/context-insensitive.dl

```
// t=0: load base facts and run the analysis
[t=0] » begin
[t=0] » file LoadInstanceField load.csv +1
// ... load other EDB relations
[t=0] » commit
// tuples omitted; report only the VarPointsTo size
[t=0] VarPointsTo size=4,986,481

// t=1: remove one load from ParameterParser.chars
[t=1] » begin
[t=1] » put LoadInstanceField ParameterParser.chars ->
          getToken/$stack44 -1
[t=1] » commit
[t=1] VarPointsTo data=(char[]@String.toCharArray,
      getToken/$stack44) diff=-1
```

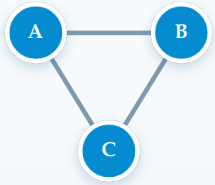
COMMAND	DESCRIPTION
`begin`	start an update round
`put rel tuple diff`	stage one tuple update
`file rel path diff`	stage a batch of updates
`commit`	apply staged changes at one logical time

RECOMPUTE AFTER RETRACTION	INCREMENTAL UPDATE
2.23 s	7 ms

https://arxiv.org/pdf/2607.23971

Richer Semantics

K-core k = 2



k-core.dl

```
fixpoint {
  .iterative active_edge
  .iterative degree

  active_edge(x,y) :- edge(x,y),
                      !removed(x), !removed(y).

  degree(x, count(y)) :- active_edge(x, y).
  removed(x) :- degree(x, d), d < 2.
}
```

https://www.flowlog-rs.com/tutorial/language/extended-semantics

FlowLog

Efficient
Fine-grained optimization

+

Extensible
Composable semantics

Mature Differential Dataflow primitives

FlowLog

Tutorial & Playground

flowlog-rs.com

GitHub Repo

github.com/flowlog-rs/flowlog

VLDB Paper