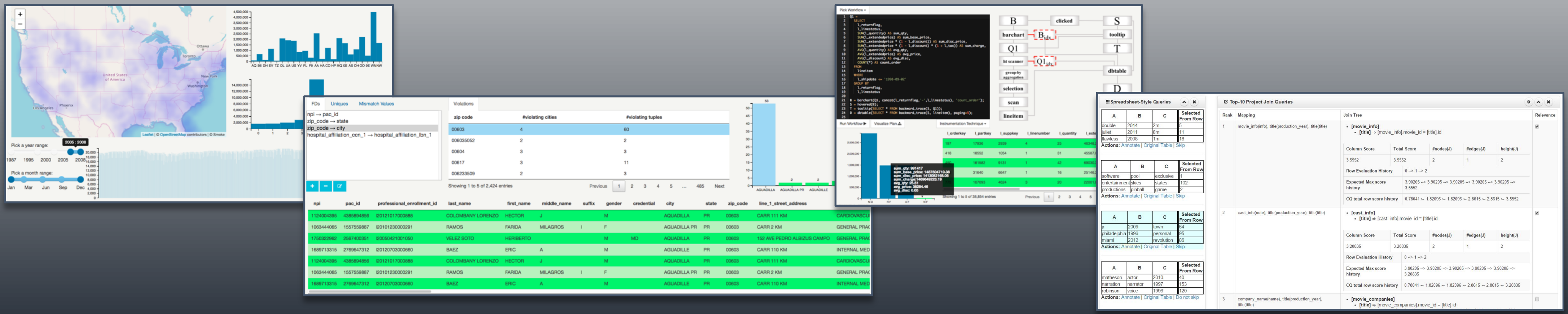
SMOKE: Fine-Grained Provenance Capture at Interactive Speed

Fotis Psallidas
fotis@cs.columbia.edu

Eugene Wu
ewu@cs.columbia.edu



Fine-Grained Provenance Primer

$$O = \gamma_{state,avg(delay)}(\texttt{Airports} \bowtie_{name = from} \texttt{Flights})$$

Airports

	name	state
a_1	LGA	NY
a_2	JFK	NY
a_3	MIA	FL

Flights

	from	delay
f_1	LGA	30
f_2	LGA	40
f_3	JFK	50
f_4	MIA	60

$\bowtie$ →

	from	delay	state
j_1	LGA	30	NY
j_2	LGA	40	NY
j_3	JFK	50	NY
j_4	MIA	60	FL

→ γ →

O

	state	avg(delay)
o_1	NY	40
o_2	FL	60

Fine-Grained Provenance Primer

Airports

	name	state
a_1	LGA	NY
a_2	JFK	NY
a_3	MIA	FL

Flights

	from	delay
f_1	LGA	30
f_2	LGA	40
f_3	JFK	50
f_4	MIA	60

	from	delay	state
j_1	LGA	30	NY
j_2	LGA	40	NY
j_3	JFK	50	NY
j_4	MIA	60	FL

O

	state	avg(delay)
o_1	NY	40
o_2	FL	60

$\{f_4\}$ = backward_trace($\{o_2\}$, Flights)

Fine-Grained Provenance Primer

Airports

	name	state
a_1	LGA	NY
a_2	JFK	NY
a_3	MIA	FL

Flights

	from	delay
f_1	LGA	30
f_2	LGA	40
f_3	JFK	50
f_4	MIA	60

	from	delay	state
j_1	LGA	30	NY
j_2	LGA	40	NY
j_3	JFK	50	NY
j_4	MIA	60	FL

O

	state	avg(delay)
o_1	NY	40
o_2	FL	60

$\{o_2\}$ = forward_trace($\{f_4\}$, O)

Fine-Grained Provenance Primer

Airports

	name	state
a_1	LGA	NY
a_2	JFK	NY
a_3	MIA	FL

Flights

	from	delay
f_1	LGA	30
f_2	LGA	40
f_3	JFK	50
f_4	MIA	60

	from	delay	state
j_1	LGA	30	NY
j_2	LGA	40	NY
j_3	JFK	50	NY
j_4	MIA	60	FL

O

	state	avg(delay)
o_1	NY	40
o_2	FL	60

Fine-Grained Provenance Primer

Navigation of the input-output connections

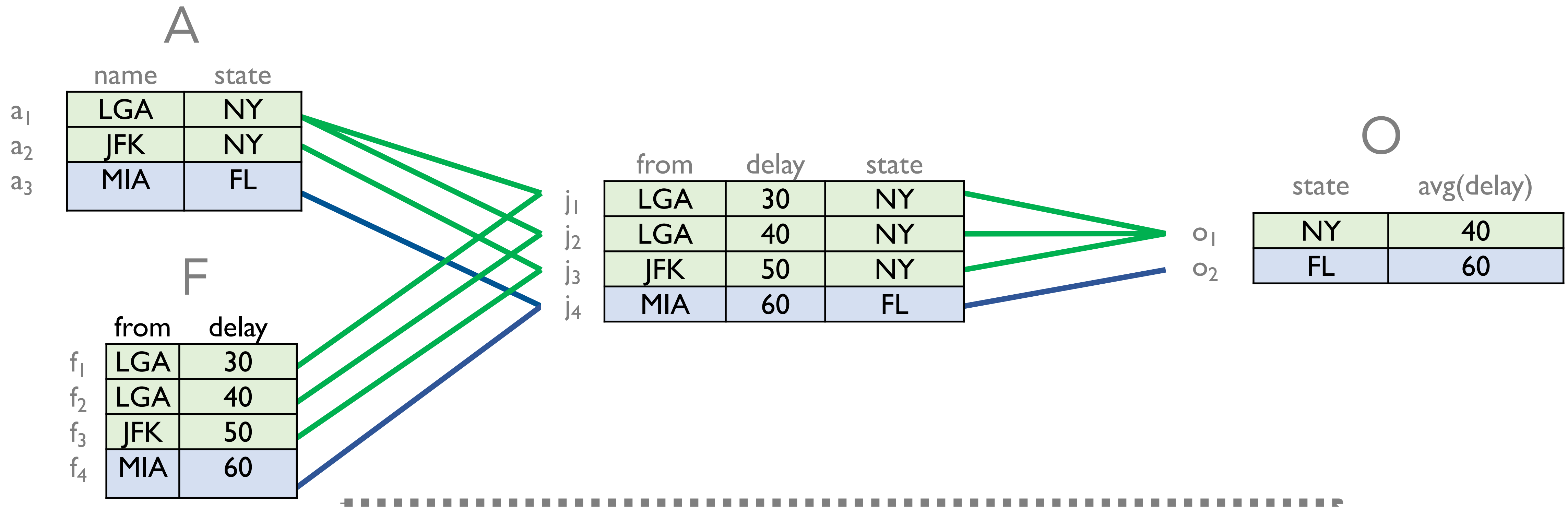
- {records} = backward_trace(...)
- {records} = forward_trace(...)

Provenance consuming queries

- SQL(backward_trace(...))
- SQL(forward_trace(...))

Fine-Grained Provenance Capture

Optimizing Provenance Constructs ⇒ Fine Grained Provenance Capture Problem



A

	name	state
a_1	LGA	NY
a_2	JFK	NY
a_3	MIA	FL

F

	from	delay
f_1	LGA	30
f_2	LGA	40
f_3	JFK	50
f_4	MIA	60

	from	delay	state
j_1	LGA	30	NY
j_2	LGA	40	NY
j_3	JFK	50	NY
j_4	MIA	60	FL

O

	state	avg(delay)
o_1	NY	40
o_2	FL	60

Fine-grained Provenance Capture Problem

Capture provenance graph w/ low-overhead to answer provenance queries efficiently

Previous Work

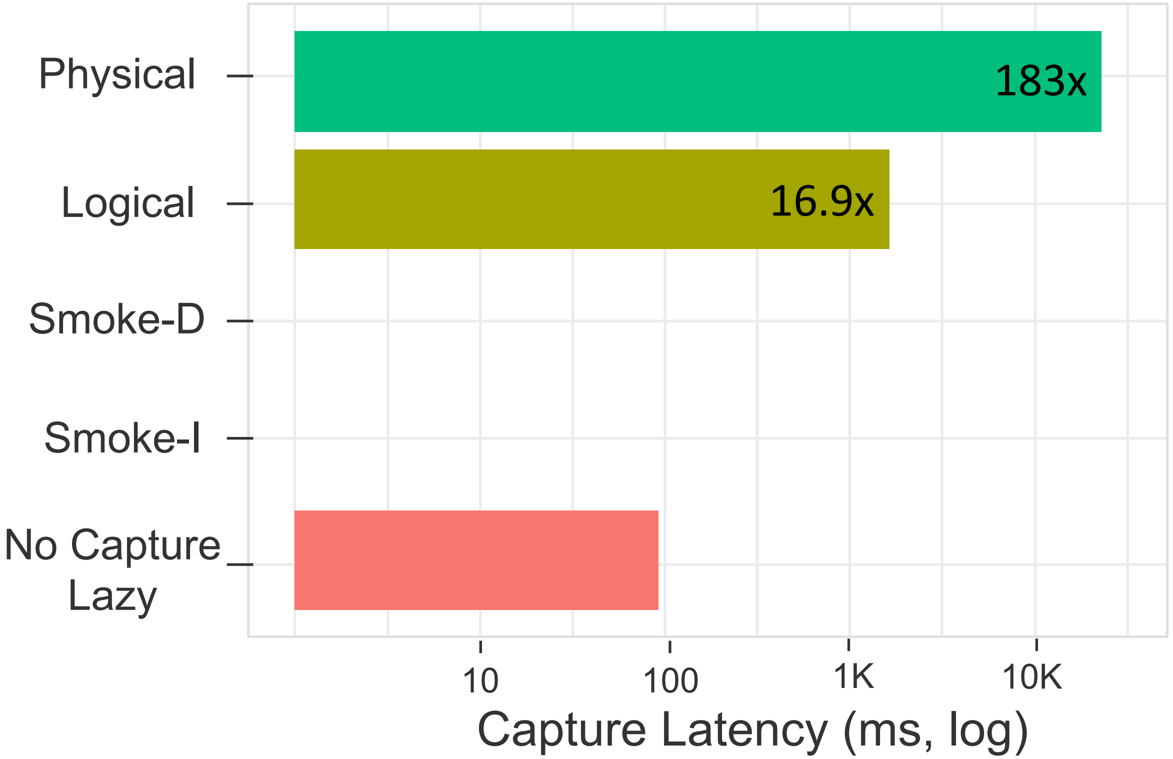
LAZY [Cui et al. and Ikeda et al.]

- Do not materialize the provenance graph
- Provenance query evaluation by rewriting to SQL queries

EAGER [Subzero, NewT, Ramp, Clothia et al., Titian, Trio, Perm, GProm, ProQL and DBNotes]

- Materialize the provenance graph
  - **Logical**: In the relational model
  - **Physical**: In a separate provenance subsystem
- Provenance query evaluation using the materialized graph

Previous Work – Provenance Capture



Main Problems

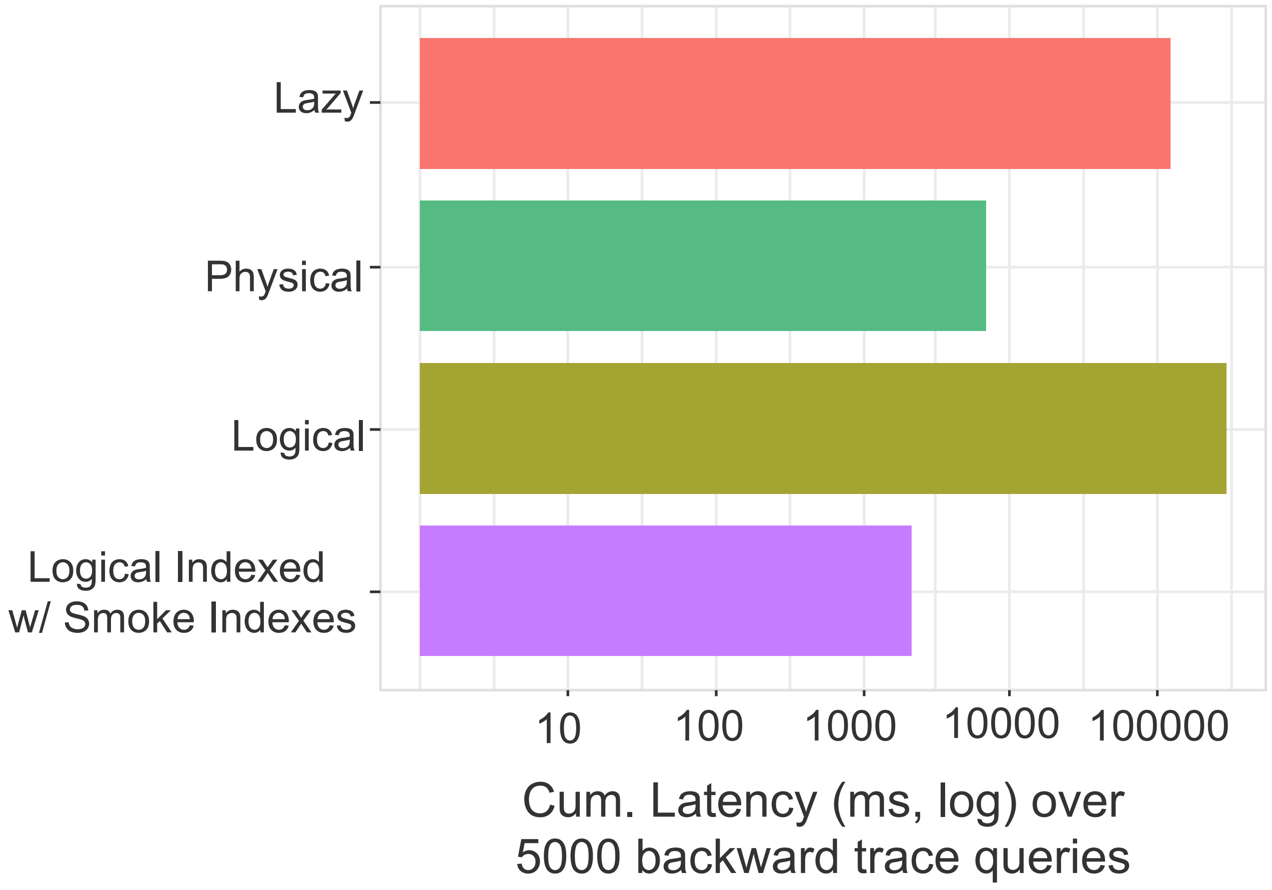
Physical

- Expensive virtual function calls
- Write-inefficient provenance storage
- No physical plan co-optimizations

Logical

- Storing graph under relational model
- Hard to optimize queries
- Expensive joins

Previous Work – Provenance Query



Main Problems

Lazy Approaches

- Equivalent SQL queries are slow

Physical Approaches

- Read-inefficient provenance storage

Logical Approaches

- Need extra indexing steps

Design Principles for Fast Provenance Systems

4 Design Principles

P1. Tight Integration

Integrate provenance capture within query plans
Index provenance in read- and write-efficient indexes

P2. Reuse work

Provenance indexes in hash tables

Intra-plan hash table reuse

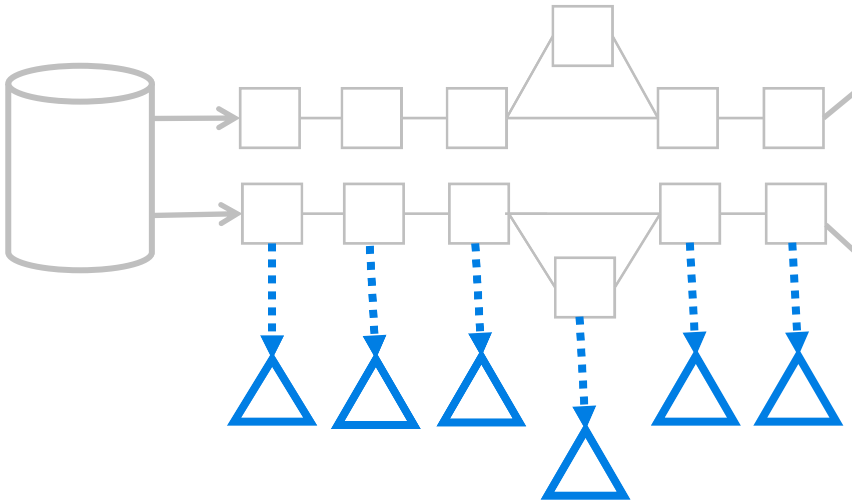
P3. Apriori Knowledge

Don't capture if not used

P4. Provenance Consumption

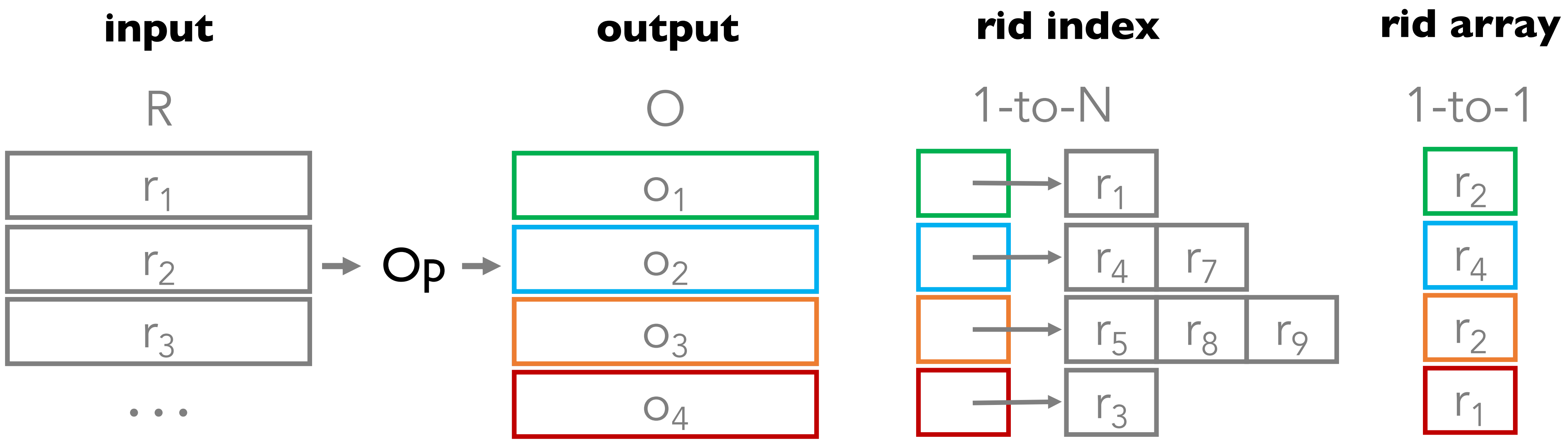
Push computation into provenance capture

Smoke Overview

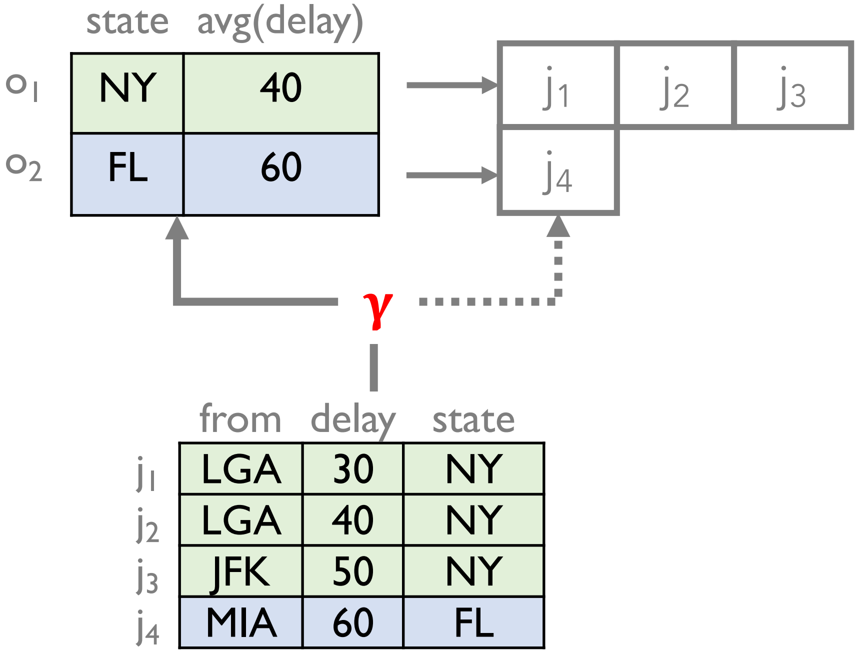


- In-Memory Query Compiled Engine
- Eager physical approach
- Write and read-efficient provenance indexes
- Two capture paradigms: Defer and Inject

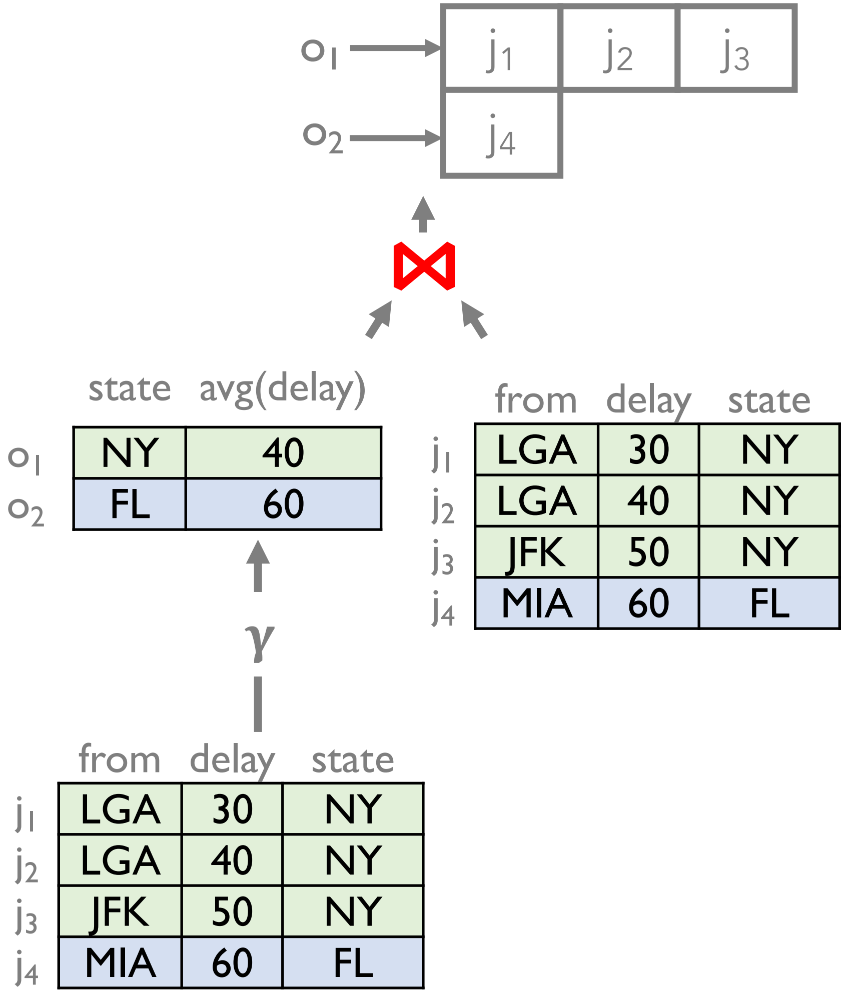
Provenance Index Representations



Two Capture Paradigms



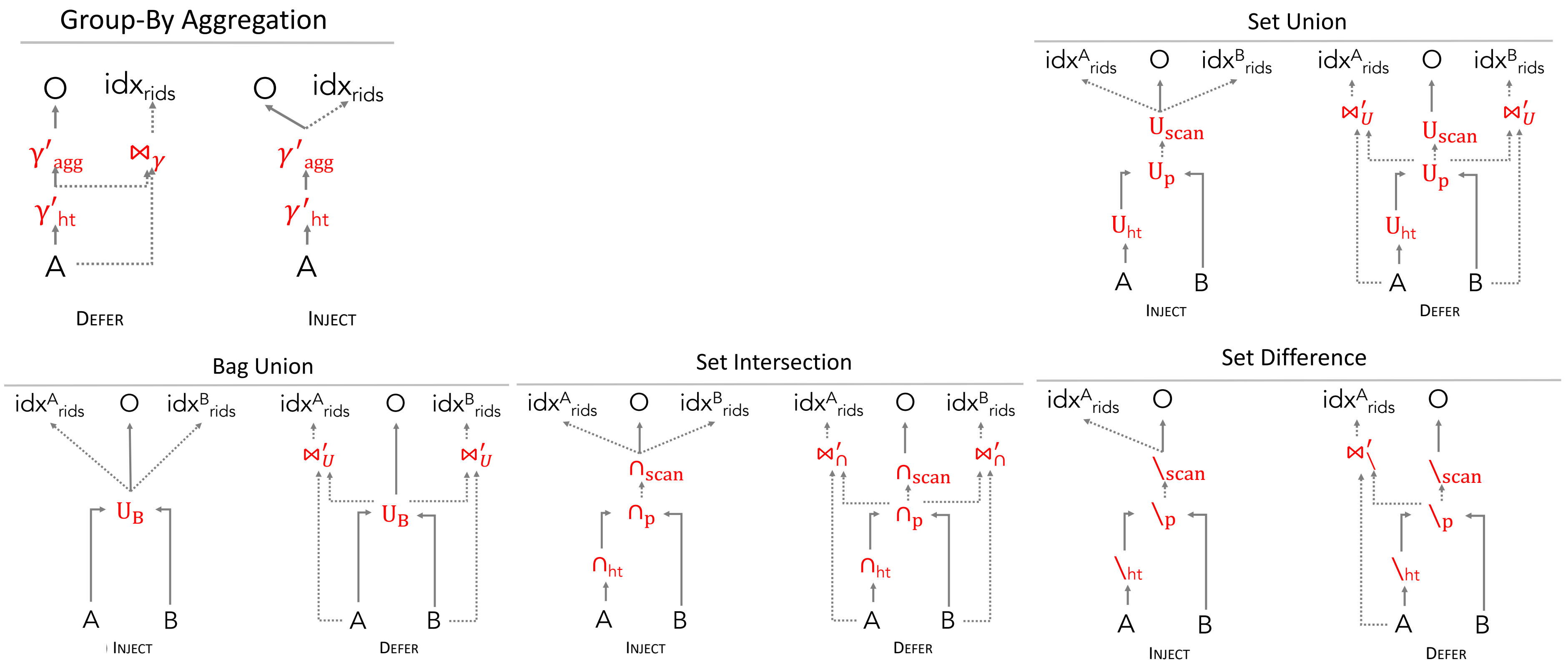
Inject

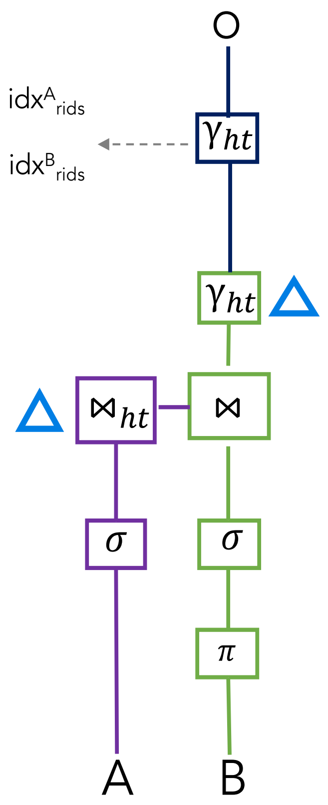


Defer

Physical Algebra

- Both paradigms for every relational operator
- Avoid materializing intermediate provenance indexes for multi-operator plans

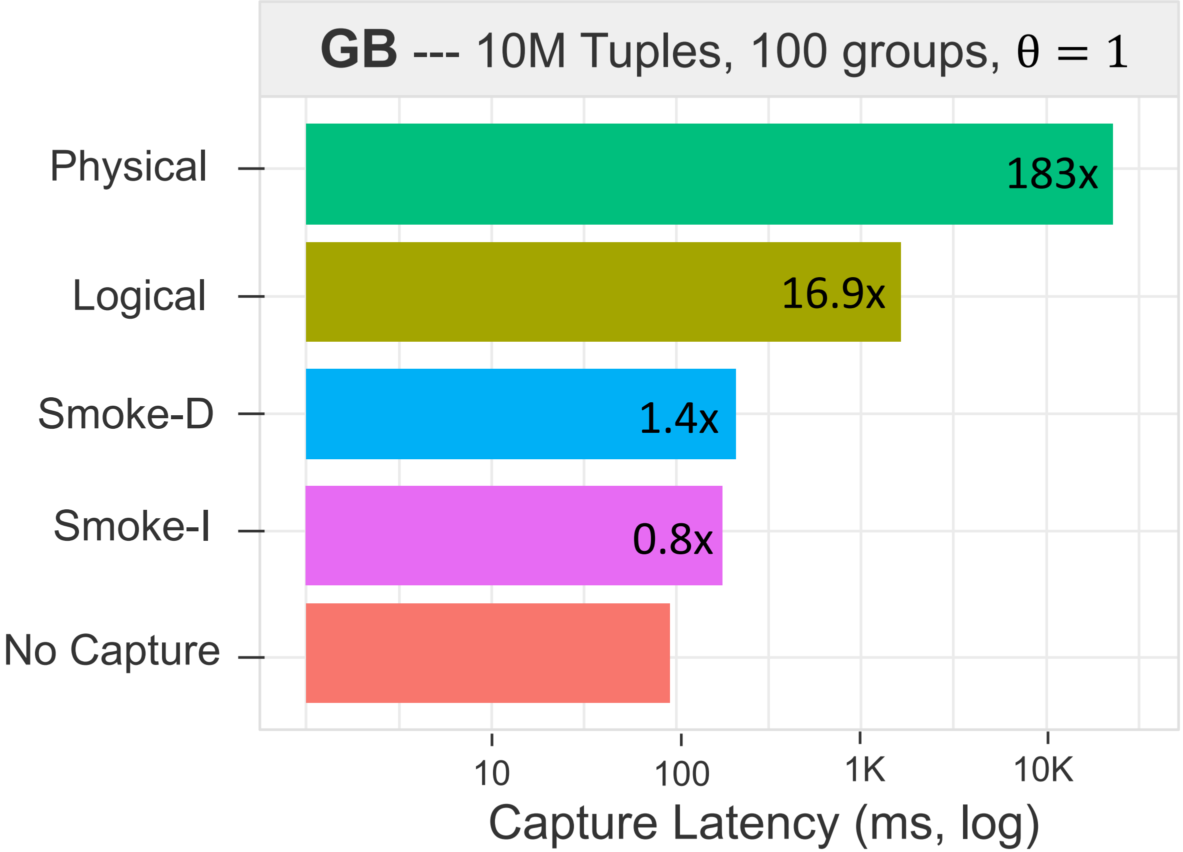




Experimental Setup

- **Provenance Capture and Querying**
  - Comparison with state-of-the-art Physical, Logical, and Lazy approaches
  - Single operator performance: microbenchmarks using zipfians
  - Multi-operator performance: TPC-H
- **Real-World Applications**
  - Comparison with state-of-the-art, hand-tuned implementations
  - Interactive Visualizations:
    - Overview first then details on demand
    - Crossfilter
  - Data Profiling: Functional Dependency Checks

Main Results - Provenance Capture



Smoke outperforms alternatives by embodying the design principles

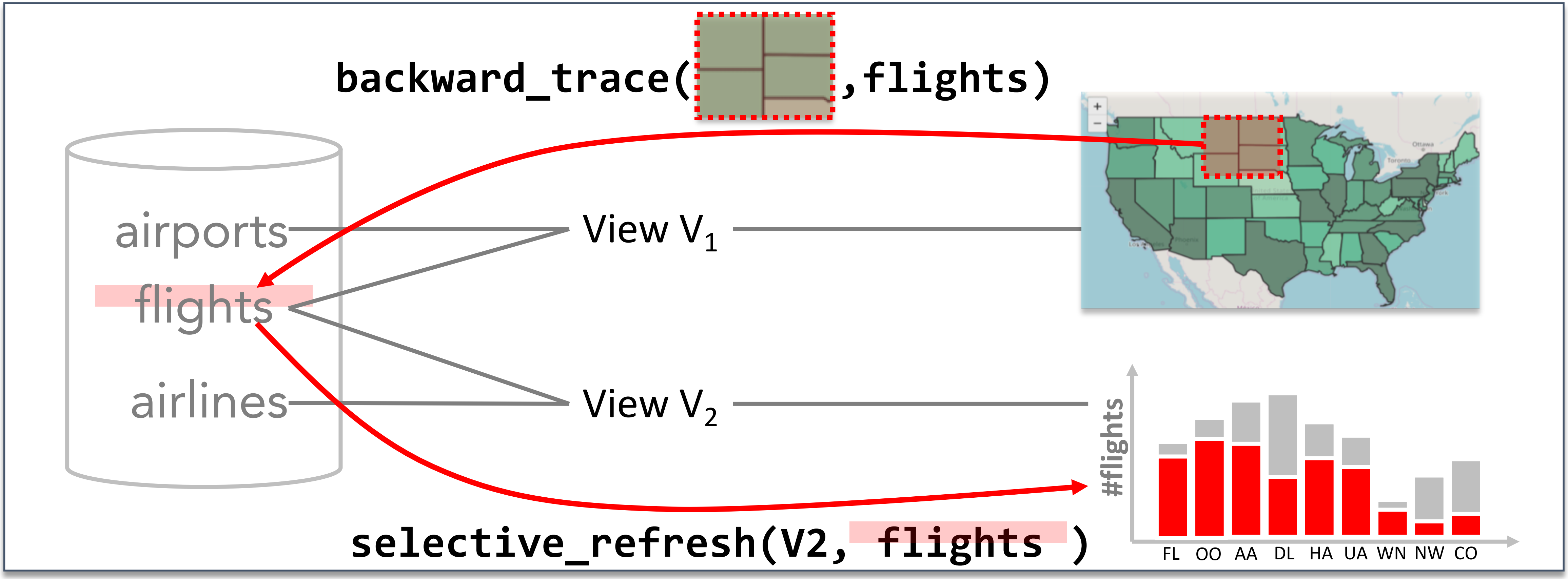
Negligible overhead over fast query engines

Can do better with opts (.8x →.4x)

Main Results - CROSSFILTER

Provenance can express important application logic

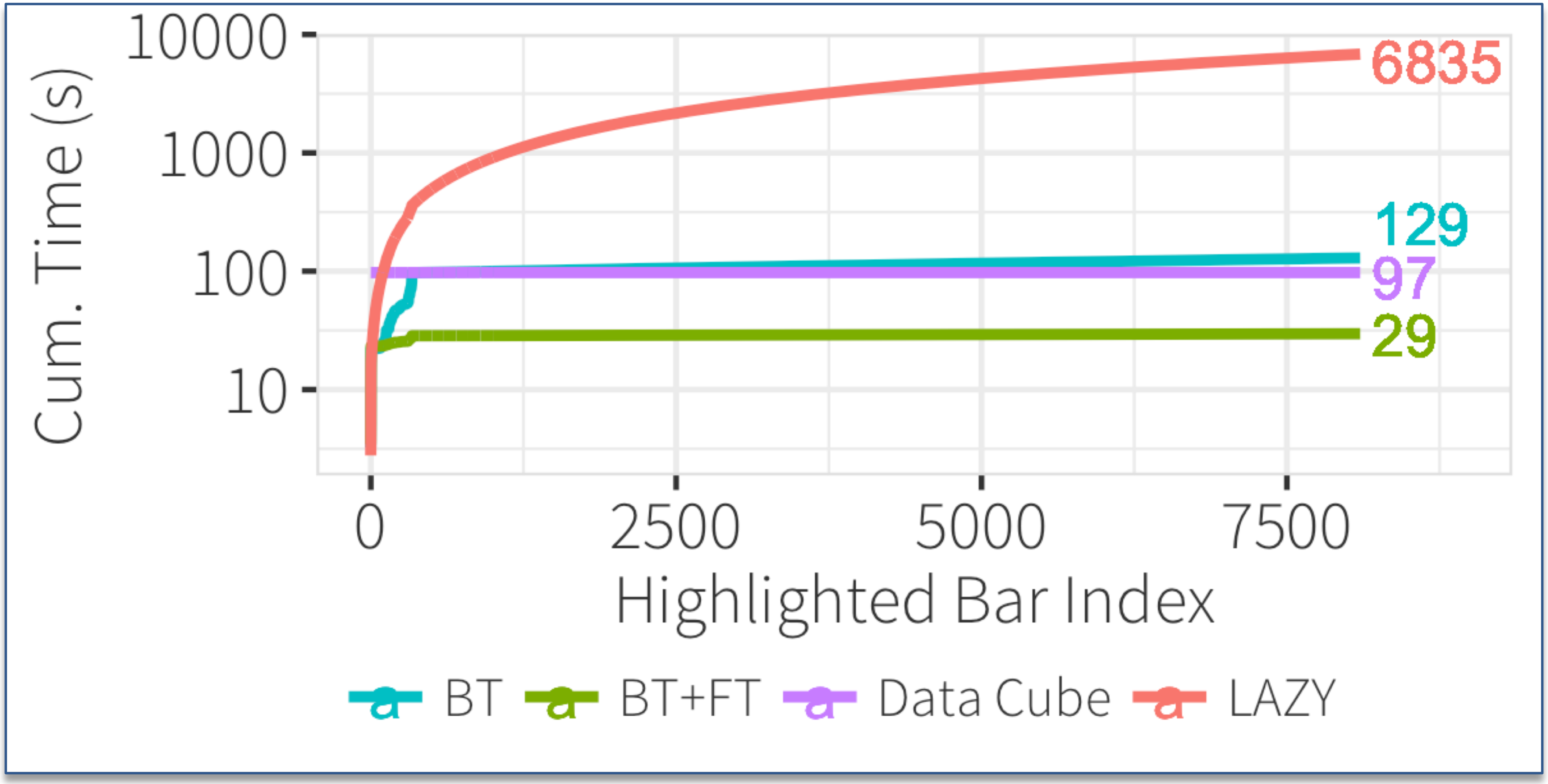
Example: Show the distribution of #flights per carrier only for selected states



Main Results - CROSSFILTER

Crossfilter on Ontime dataset: 5 initial views over 125 million delayed flights

Data cubes: Immens > 1 hour , Nanocubes > 30 minutes, Hashedcubes > 4 minutes



Takeaway

Interactive Visualizations	Interactive Data Profiling	Multi-Application Linking	Interactive Query Specification	What-if Provisioning
Query Explanations	Why-not Analytics	Iterative Analytics	Viz Workflow Debugging	Interactive Data Cleaning
Interaction Debugging	ML Interpretability	Visualization Deconstruction and Restyling	Interaction By Example	Application Design Search
Collaborative Communication	Action Recovery	Sense-Making	Meta-Analysis	Replication and Reproducibility
Data Integration	Debugging	Network Diagnostics	Dataset Versioning	Auditing

…

Provenance for Interactive Visualizations [HILDA '18]

Thank you // Q

Smoke: Fine-Grained Lineage At Interactive Speed **[VLDB18]**

A Deep breath of Data-Intensive Lineage Applications [SIGMOD18]

Provenance for Interactive Visualizations [HILDA18]

Combining Design and Performance in a DVMS [CIDR17]