An (Updated) Overview of Data Provenance: Concepts, Challenges and Opportunities

Chrysanthi Kosyfaki
HKUST

Paul Groth
UvA

SIGMOD 2026 – Tutorial
Bengaluru, India

Tutorial outline

- Provenance in the past
- Provenance now
- Provenance in the future

Tutorial outline

- Provenance in the past
  - Provenance history
  - Provenance models
  - Provenance in databases
  - Provenance and graph analytics
- Provenance now
- Provenance in the future

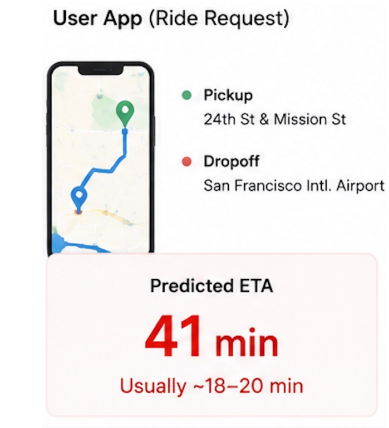
Modern systems became hard to understand

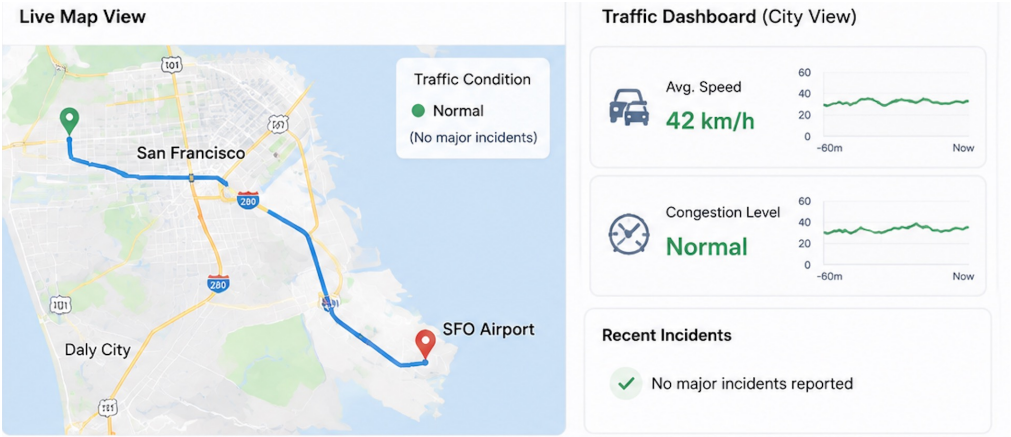
- Today’s data systems are highly interconnected
- A single output may depend on:
  - Multiple data sources
  - Streaming computations
  - Graph analytics
  - Feature pipelines
  - Machine learning models
  - External services

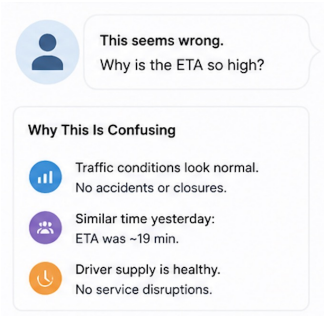
Outputs are generated through long chains of dependencies

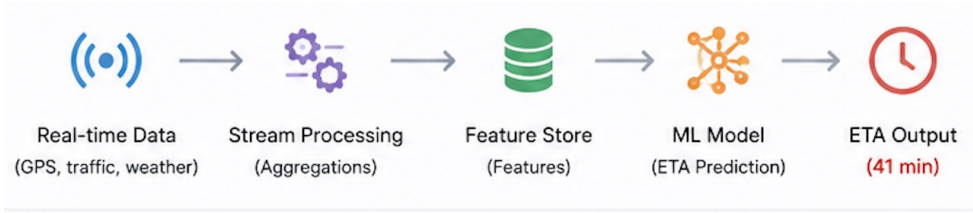
A simple example

- A user requests a ride. The system predicts a long ETA, but conditions look normal.









Why did the system predict 41 minutes? 😖

Why is this difficult to explain?

- The predictions depend on many components
- Possible causes:
  - Delayed GPS events
  - Outdated road graph
  - Incorrect stream aggregation
  - Corrupted features
  - Stale model inputs
  - Inconsistent external data

Takeaway: Understanding the output requires understanding dependencies across the pipeline.

Existing explainability is not enough!

- Explaining the model is different from explaining the data.
- Missing information:
  - Where did traffic data come from?
  - Which stream generated it?
  - Which transformations modified it?
  - Which events influenced it?
  - Was the source data correct?

Modern systems require explanations about both models and data

Why provenance became important again?

- Streaming systems:
  - Continuous updates, infinite event streams, stateful computations
- AI and ML pipelines:
  - Feature engineering, training data attribution, prediction debugging
- Distributed data ecosystems:
  - Multiple engines, cross-system workflows, complex dependencies

Modern systems became too complex for manual dependency tracking

Why do we need provenance today?

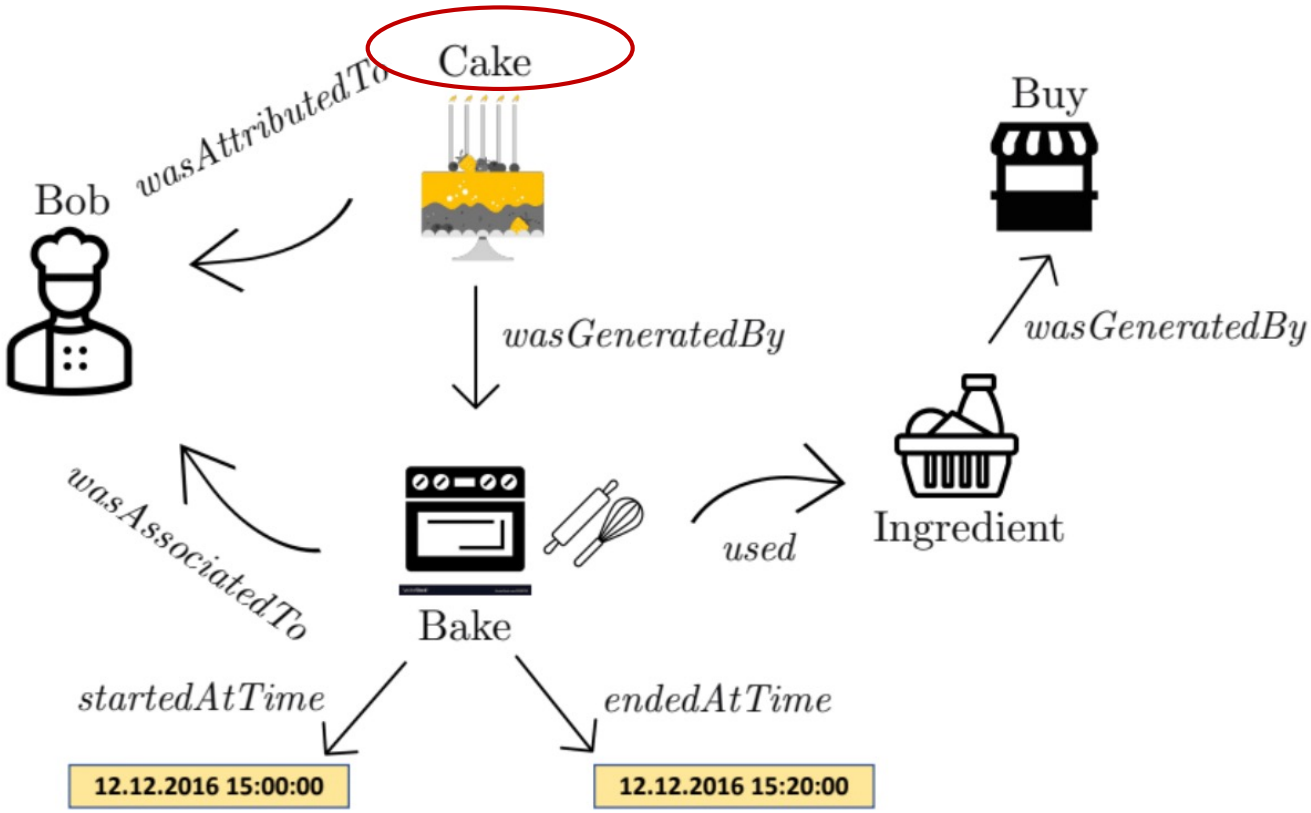
- Modern systems are:
  - Distributed
  - Streaming
  - ML-driven
  - Dependency-heavy
  - Difficult to debug
- Organizations need:
  - Explainability
  - Debugging
  - Reproducibility
  - Accountability
  - Trust

Modern data systems cannot be understood without dependency tracking

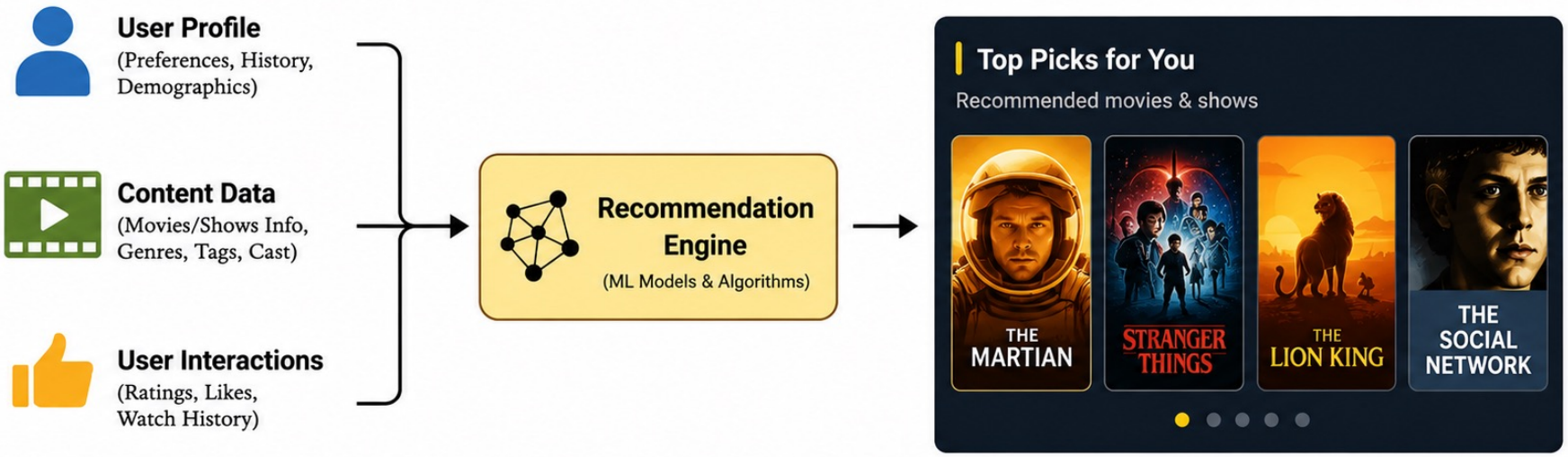
But before we start, what is data provenance really?

Data Provenance – an example

What's the origin of the cake?



Data Provenance – another example



Stars	Meaning
★	Dislike
★★	It's okay
★★★	Like
★★★★	Really like
★★★★★	Love it

Data provenance

- Data provenance
history of a piece

Why and Where:
A Characterization of Data Provenance*

Peter Buneman, Sanjeev Khanna**, and Wang-Chiew Tan

University of Pennsylvania
Department of Computer and Information Science
200 South 33rd Street, Philadelphia, PA 19104, USA
{peter,sanjeev,wctan}@saul.cis.upenn.edu

Abstract. With the proliferation of database views and curated databases, the issue of *data provenance* – where a piece of data came from and the process by which it arrived in the database – is becoming increasingly important, especially in scientific databases where understanding provenance is crucial to the accuracy and currency of data. In this paper we describe an approach to computing provenance when the data of interest has been created by a database query. We adopt a syntactic approach and present results for a general data model that applies to relational databases as well as to hierarchical data such as XML. A novel aspect of our work is a distinction between "why" provenance (refers to the source data that had some influence on the existence of the data) and "where" provenance (refers to the location(s) in the source databases from which the data was extracted).

1 Introduction

Data provenance — sometimes called "lineage" or "pedigree" — is the description of the origins of a piece of data and the process by which it arrived in a database. The field of molecular biology, for example, supports some 500 public databases [1], but only a handful of these are "source" data in the sense that they receive experimental data. All the other databases are in some sense *views* either of the source data or of other views. In fact, some of them are views of each other, which sounds nonsensical until one understands that the individual databases are not simply computed by queries, but also have added value in the form of corrections and annotations by experts (they are "curated"). A serious problem confronting the user of one of these databases is knowing the provenance of a given piece of data. This information is essential to anyone interested in the accuracy and timeliness of the data.

Understanding provenance and the process by which one records it is a complex issue. In this paper we address an important part of the general problem. Suppose a database (a view) $V = Q(D)$ is constructed by a query Q applied to

Data Provenance: Some Basic Issues

Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan

University of Pennsylvania

ase with which one can copy and transform data on
e it increasingly difficult to determine the origins of a
use the term *data provenance* to refer to the process
ording the origins of data and its movement between
ance is now an acute issue in scientific databases where
validation of data. In this paper we discuss some of
es that have emerged in an initial exploration of the

ıta on the Web, do you have any information about how
possible that it was copied from somewhere else on the
y have also been copied; and in this process it may well
and edited. Of course, when we are looking for a best
movie rating, we know that what we are getting may be
e learned not to put too much faith in what we extract
, if you are a scientist, or any kind of scholar, you would
in the accuracy and timeliness of the data that you are
ular, you would like to know how it got there.
, the Web has completely changed the way in which data
moved very rapidly from a world of paper documents
ocuments and databases. In particular, this is having a
scientific research is conducted. Let us list some aspects

is essentially unmodifiable. To "change" it one issues a
his is a costly and slow process. On-line documents, by
ıd often are) frequently updated.
are often databases, which means that they have explicit
elopment of XML has blurred the distinction between
abases.
/databases typically contain data extracted from other
es through the use of query languages or "screen-scrap-

, the field of Molecular Biology is possibly one of the
umers of modern database technology and has generated

Supporting Fine-Grained Data Lineage in a Database Visualization Environment

ıael Stonebraker
ECS
ifornia
20-1776*

ific cyclone track point in the processed data set to a
icular array element in the source data set is not feasible
g such an approach; the amount of metadata required
ıld be far too large.
This type of scenario is common in the computational
nces. In this paper, we present an approach for deriving
lineage of a datum dynamically and at a fine granularity.
ead of relying on metadata, our approach combines a
ted amount of knowledge about the processing opera-
with analysis of the base data. The approach works best
n integrated into a database server using user-defined
tions, but can be implemented outside of a database
ronment as well. In this paper, we describe our ap-
ach in terms of abstract properties and then in terms of
lementation.
n general, the **lineage** of a datum consists of its en-
processing history. This includes its origin (*e.g.*, the
ıtifier of the base data set, the recording instrument, the
rument's operating parameters) as well as all subsequent
essing steps (algorithms and respective parameters) ap-
d to it. Many applications of lineage become evident if
considers processing history as a dataflow graph. For
mple, lineage information allows the user to trace the im-
of faulty source data or buggy programs on derived data
. It also allows the user to investigate the source data
rograms that produced an anomalous data set. Such
stigations may be difficult or impossible without data
age: the user may not be familiar with processing steps
ch were written by an expert programmer. Further, trac-
back through a number of processing steps is tedious and
-consuming, particularly if large data sets are involved.
The perceived importance of data lineage has grown in
with the increased volume and widened dissemination
rocessed data sets. The amount of support for data lin-
has grown as well. For example, new scientific data
dards (*e.g.*, the Spatial Data Transfer Standard [9], the
tial Archive and Interchange Format [13], and the draft
ıtent Standard for Digital Spatial Metadata [5]) gener-

What was the need in the 1990s?

- A way to solve problems created by data warehouses and materialized views:
  - Fixing broken views:
    - Data warehouses combined data from dozens of different company databases
    - If a report showed a strange or corrupted number, database administrators had no way to know which source database or which specific row caused the error
  - Explaining visualization outputs
    - As early data visualization tools emerged, users looking at a chart needed to right-click a data point and ask, *"Where did this specific number come from?"* requiring a direct map back to the source
  - Efficient view maintenance
    - When data changed in a source database, the system needed to know exactly which parts of the warehouse data to update without recalculating the entire database from scratch

Need for a provenance tutorial

- The last tutorial related to data provenance in major database conferences was on SIGMOD 2016
  - "Provenance: On and behind the screens" by Herschel and Hlawatsch
    - Provenance on the screen: focused on visualization techniques
      - Without getting overwhelmed by metadata
    - Provenance behind the screen: explored the fundamental concepts, models, and types of provenance
      - required to track data dependencies, assess quality, and ensure reproducibility

- Why a tutorial is more important than ever 10 years later?
  - Data provenance is more relevant than ever especially with data explainability concept
  - Need to cover more topics where provenance has been involved
  - Update researchers and academics about what's new in the area

Provenance models

Data Provenance Models

- Provenance for existing results
  - Refers to metadata related to the production of the data in the query results
  - **Categories:**
    - Where-provenance
    - Why-provenance
    - How-provenance

Where-provenance*

Query Result

name	address
Bob	99 High St

Where is the address copied from?

Customers

name	address	phone no
Bob	99 High St	6173330043
Mary	125 Baker St	6175051234

```
SELECT DISTINCT C.name, C.address
FROM Customers C, Orders O
WHERE C.name = O.name
  AND O.type="Furniture"
```

Orders

name	type	price
Bob	Furniture	$1200
Bob	Furniture	$670
Mary	Clothing	$234

*CS 5999 L1: User-centric Systems for Data Science, CS Dept. Boston University

Why-provenance*

Query Result

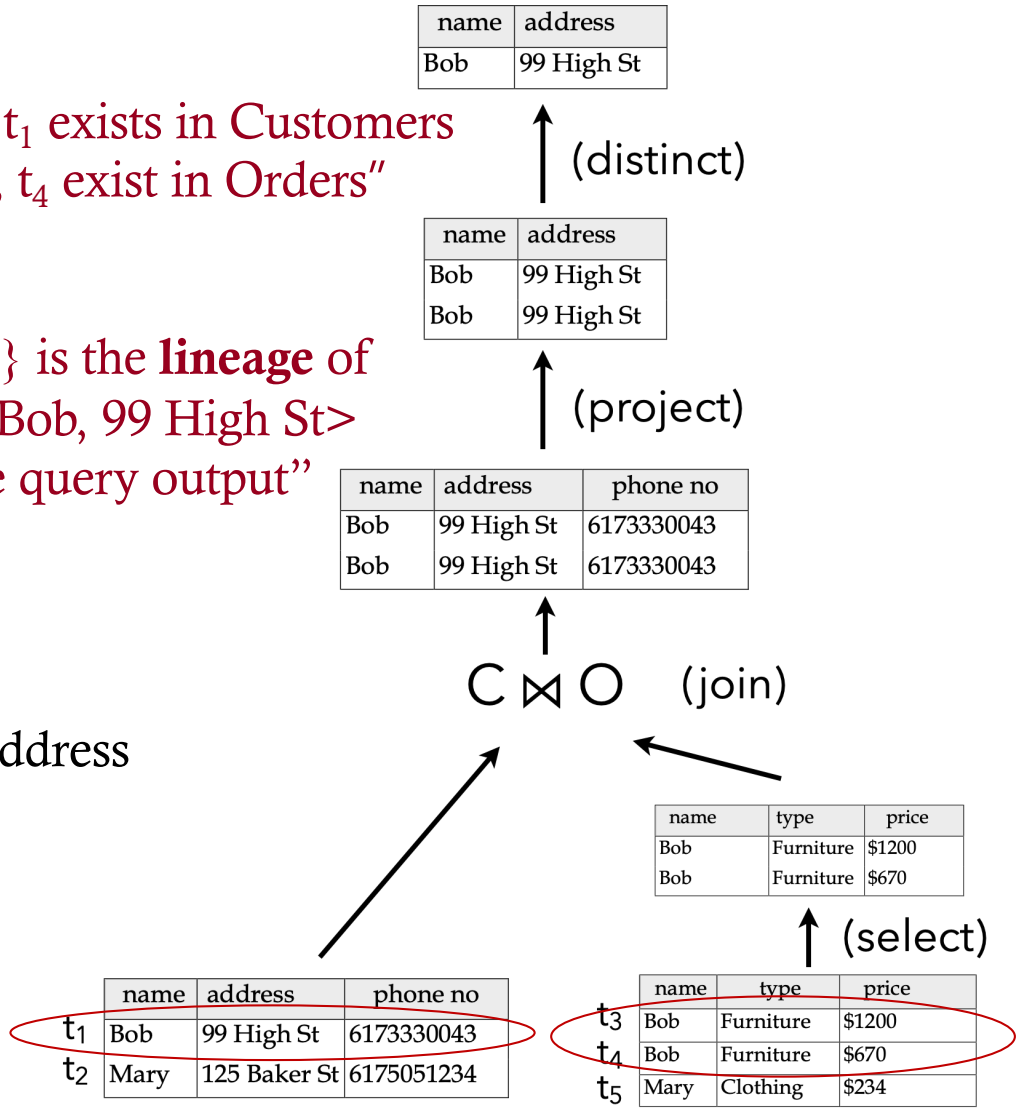
name	address
Bob	99 High St

Why is this tuple in the result?

"Because t_1 exists in Customers and t_3, t_4 exist in Orders"

$\{t_1, t_3, t_4\}$ is the **lineage** of tuple <Bob, 99 High St> in the query output"

```
SELECT DISTINCT C.name, C.address
FROM Customers C, Orders O
WHERE C.name = O.name
  AND O.type="Furniture"
```



*CS 5999 L1: User-centric Systems for Data Science, CS Dept. Boston University

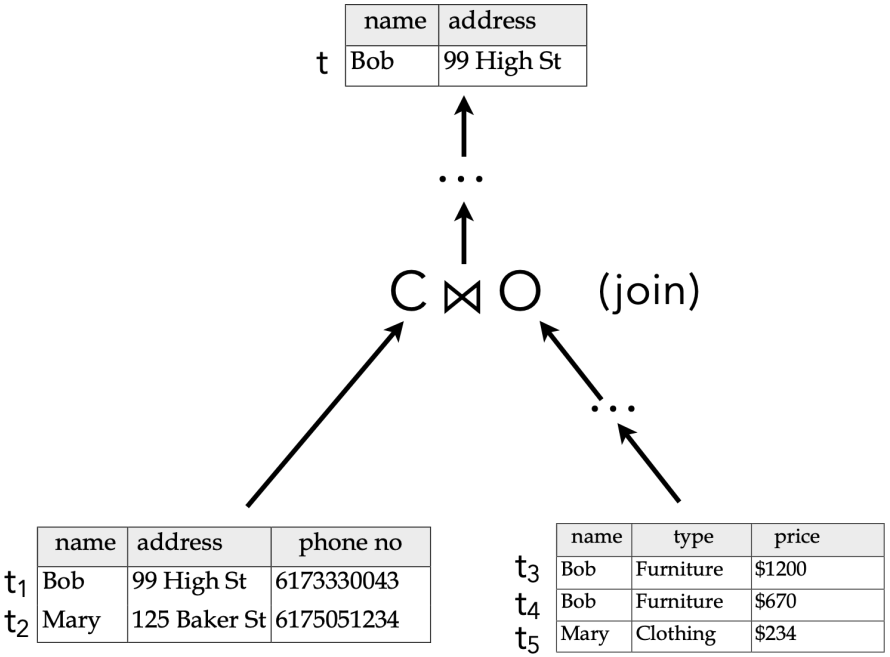
Witness*

- Given a database D and a SQL query q, a witness W for a tuple t in the query result q(D) is a set of input tuples sufficient to produce t

$W_1(t) = \{t_1, t_3, t_4\}$

$W_2(t) = \{t_1, t_3\}$ $W_3(t) = \{t_1, t_4\}$

- There can be more than one witnesses
- A witness is sufficient but not necessary to produce t
- W_2 and W_3 form the minimal witness basis



*CS 5999 L1: User-centric Systems for Data Science, CS Dept. Boston University

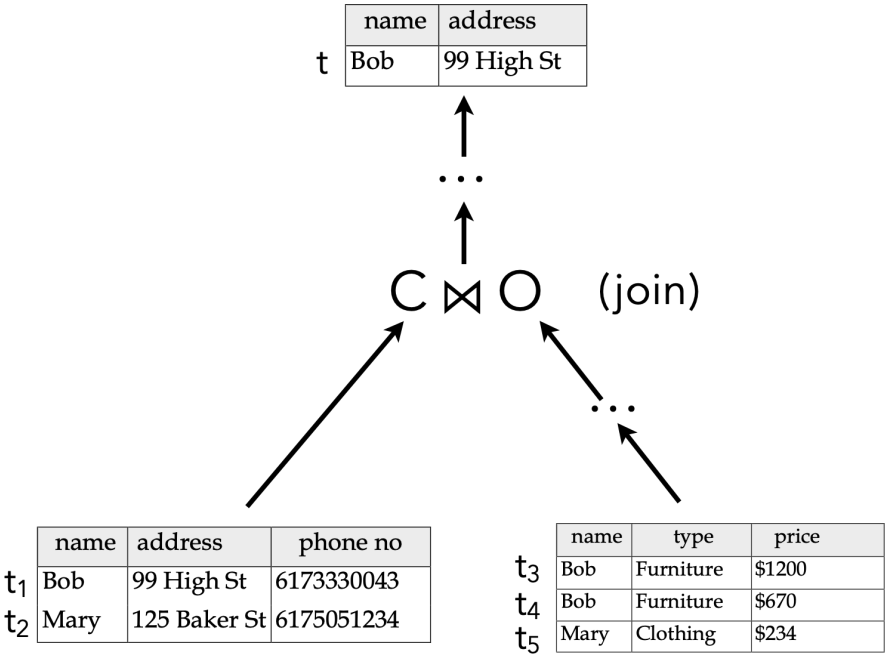
Why-provenance*

- Given a database *D* and a SQL query *q*, the Why-provenance of a tuple t in the query result *q(D)* is the set of witnesses of *t*

Why(t)= { {t1, t3}, {t1, t4} }

t's Why-provenance is not equivalent to t's lineage

t's lineage is the union of the witnesses in t's Why provenance



*CS 5999 L1: User-centric Systems for Data Science, CS Dept. Boston University

How-Provenance*

Query Result

name	address
Bob	99 High St

How was this tuple produced?

Customers

name	address	phone no
Bob	99 High St	6173330043
Mary	125 Baker St	6175051234

Orders

name	type	price
Bob	Furniture	$1200
Bob	Furniture	$670
Mary	Clothing	$234

```
SELECT DISTINCT C.name, C.address
FROM Customers C, Orders O
WHERE C.name = O.name
  AND O.type="Furniture"
```

*CS 5999 L1: User-centric Systems for Data Science, CS Dept. Boston University

Why-not Provenance

- Provenance for missing results
  - Refers to metadata that explains why some expected data is not present in the query result
  - **Categories:**
    - Instance-based provenance:
      - Identifies absent data in the source database
      - Explains which details are missing to obtain the intended tuple in the query output
    - Query-based provenance:
      - Detects one or more query operators that eliminate data which otherwise could produce the missing tuple
    - Refinement-based provenance:
      - Modifies the query, when feasible, to retrieve the missing result

Instance-based Provenance

- Example

Table: Employees

Tuple ID	EID	Name	Dept
e_1	7	David	Sales
e_2	8	Bob	Sales

Table: Performance

Tuple ID	EID	Revenue	Rating
p_1	7	\$15k	5
p_2	8	\$5k	3

- Query: Select the employees where their performance rating is larger than 4.
  - Result: Name: 'David'
  - The "Instance-based": e_1, p_1
    - This output exists because row e_1 and row p_1 were processed together

Query-based Provenance

- Example

Table: Employees

Tuple ID	EID	Name	Dept
e_1	7	David	Sales
e_2	8	Bob	Sales

Table: Performance

Tuple ID	EID	Revenue	Rating
p_1	7	$15k	5
p_2	8	$5k	3

- Query: Select the employees where their performance rating is larger than 4.
  - Result: Name: 'David'
  - The "Query-based": [join] → [Filter: Rating more than 4] → [Project: Name]
    - The system shows the logic path through the database operators rather than the rows

Refinement-based Provenance

- Example

Table: Employees

Tuple ID	EID	Name	Dept
e_1	7	David	Sales
e_2	8	Bob	Sales

Table: Performance

Tuple ID	EID	Revenue	Rating
p_1	7	$15k	5
p_2	8	$5k	3

- Query: Select the employees where their performance rating is larger than 4.
  - Result: Name: 'David'
  - The "Refinement-based":
    - Level 1 (Course - Database): result from the HR_Production database
    - Level 2 (Mid - Schema): Result generated by joining the Employees and Performance
    - Level 3 (Fine - Tuple): Result derived specifically from Employee ID 7 and Performance_Record p_1

Summary Comparison

- Instance-based:
  - **Primary focus** – data items
  - **Best use cases** – debugging incorrect values in a specific report
- Query-based:
  - **Primary focus** – query logic
  - **Best use cases** – understanding the complexity and cost of a transformation
- Refinement-based:
  - **Primary focus** – user needs
  - **Best use cases** – managing massive datasets where you only need details on demand

Provenance in database systems

Why do databases need provenance?

- Unexplained query results
  - A user sees a row in the output and asks “where did this come from”
    - Standard SQL has no answer
    - Without lineage, debugging requires manual reverse-engineering of joins and transformations
- Missing answers (why-not)
  - An expected tuple is absent. Was it filtered out? Did a join condition fail?
    - Without provenance, answering why a tuple is NOT in the result requires re-examining every operator.
- View update and data curation
- Probabilistic and uncertain data

Provenance inside database engines

- Provenance has traditionally been studied at the logical query level
- Modern analytical engines reopen provenance questions at the systems level
- Vectorized execution and late materialization affect lineage capture
- Provenance increasingly interacts with physical execution strategies

Classical provenance focus
- Relational algebra semantics
- Tuple derivations
- Logical operator dependencies

Modern engine realities
- Vectorized pipelines
- Compressed execution
- Adaptive operators
- Columnar storage
- In-memory processing

Modern provenance must coexist with high-performance analytical execution

Systems: from theory to practice

Perm [1]

Glavic & Alonso (ICDE 2009)

- Rewrites SQL into self-join-free queries that carry provenance as extra columns
- Same data model for data and provenance (no external store needed)
- Supports SPJU + aggregation
- Limitation: nested/correlated subqueries require complex rewrites

GProM [2]

Arab et al., (TaPP 2014)

- Algebraic graph model: query → rewrite → any SQL backend (Oracle, PostgreSQL, SQLite)
- Extends Perm: handles INSERT/UPDATE/DELETE and concurrent transactions
- First system to compute provenance of concurrent transactions via reenactment
- Cost-based optimizer for instrumented (rewritten) queries

ProvSQL

Senellart et al., (VLDB 2018)

- Hooks into the PostgreSQL executor (no query rewriting required)
- Computes semiring-annotated results natively (supports provenance polynomials)
- Integrated with probabilistic DBs via d-DNNF circuit compilation
- Open source

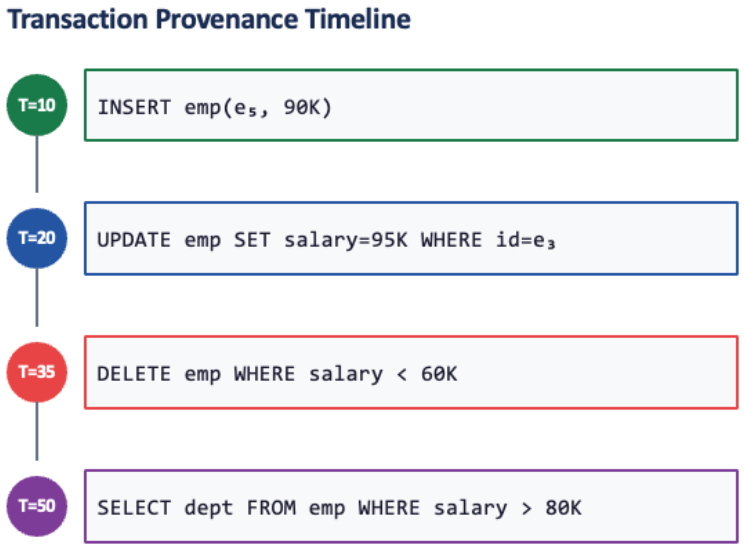
[1] Glavic and Alonso, "Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting", ICDE 2009
[2] Arab et al., "A Generic Provenance Middleware for Queries, Updates, and Transactions", TaPP 2014
[3] Senellart et al., "ProvSQL: provenance and probability management in PostgreSQL", VLDB 2018

Provenance for updates and transactions

- Problem
  - Classical provenance is defined only for SELECT queries (reads, not writes)
  - Real databases have INSERT/UPDATE/DELETE and concurrent transactions
  - Audit logs record that a row changed but not why
    - They don't link back to the query that caused it
  - Compliance (SOX, HIPPA)
    - Need to know which transaction T wrote this tuple and what T read to compute it

Provenance for updates and transactions

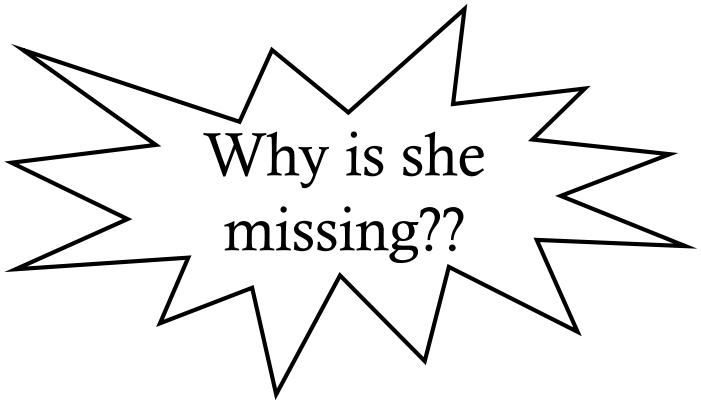
- GProM reenactment approach
  - Replay a past transaction using time-travel queries against a versioned snapshot
  - Provenance of updates = semiring annotations on the reenacted result
  - **Works retroactively:**
    - Only requires an audit log + DBMS time travel (Oracle flashback, PostgresSQL, temporal tables)



Reenactment replays T = 20 and T = 35 via time-travel to compute which INSERT (T = 10) is responsible for the query result at T = 50

The why-not problem

- Explaining missing answers

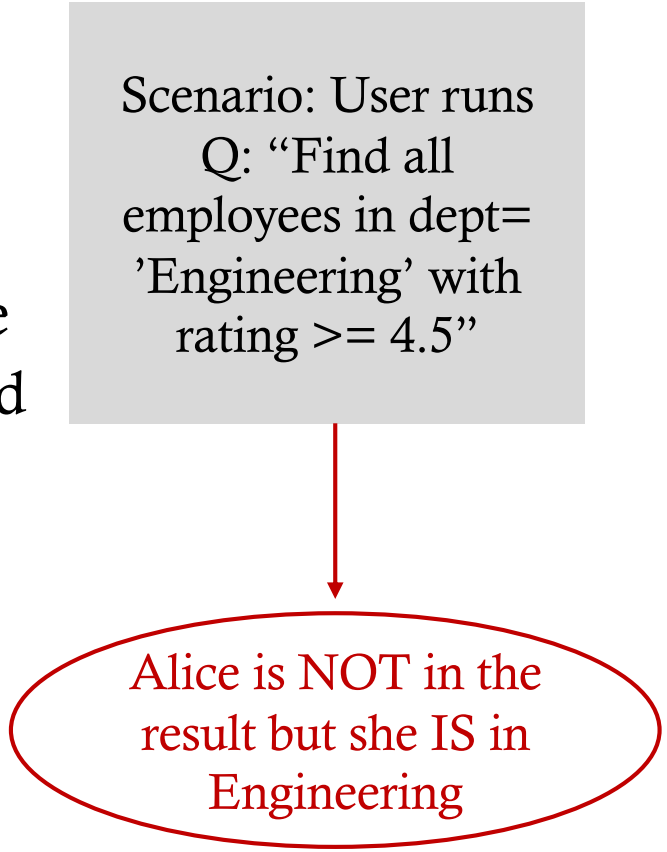


Scenario: User runs Q: “Find all employees in dept= ’Engineering’ with rating >= 4.5”

Alice is NOT in the result but she IS in Engineering

The why-not problem

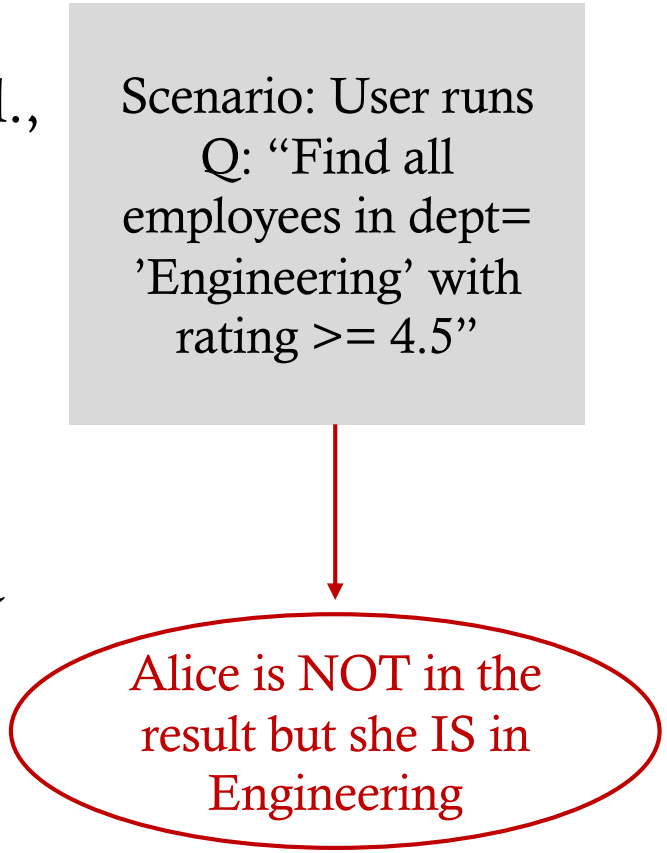
- Why-not concept
  - First introduced by Chapman and Jagadish (SIGMOD 2009)*
  - Identifies the first operator in the query plan that eliminated the tuple
  - Reports: "σ (rating ≥ 4.5) eliminated Alice because Alice.rating = 4.2"
  - Limitation: only explains the first filtering operator
    - Does not rank explanations or suggest fixes



*Chapman and Jagadish, "Why not?" SIGMOD 2009

The why-not problem

- PUG framework
  - Why and why-not unified (Lee et al., VLDBJ 2019)*
  - Provenance and unification graph
    - A single structure answers both positive (why) and negative (why-not) questions over conjunctive and SPJU queries
  - Limitation: handles aggregation via provenance abstraction
    - Complexity analyzed per query class



*Lee et al., “PUG: a framework and practical implementation for why and why-not provenance”, VLDBJ 2019

Storage explosion

- Fine-grained may exceed original data size
- Repeated workflows generate repeated derivations
- Compression becomes necessary for scalability
- Approximate provenance may become acceptable in practice

- How to solve this issue:
  - Factorized provenance, compressed lineage graphs, provenance summarization, lazy reconstruction, hierarchical lineage
- System implications:
  - Indexing complexity, provenance query optimization, storage hierarchy design, incremental maintenance

Scalable provenance requires compression-aware system design

Takeaways

Summary

- Provenance evolved from logical lineage to systems infrastructure
- Scalability remains a systems challenge
- Future analytical engines will increasingly require provenance-aware execution

Provenance and graph analytics

Why provenance matters in graphs

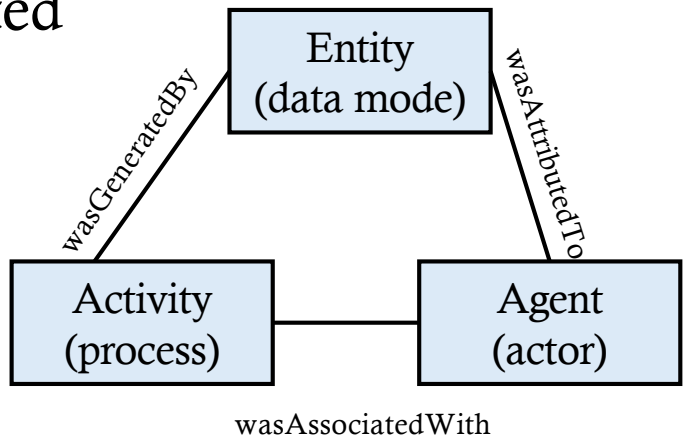
- Graphs are everywhere
- Queries over graphs produce derived facts
  - But where do they come from?
- Without provenance, graph-derived results are black boxes
- Provenance enables:
  - Trust and quality assessment, reproducibility, debugging, auditing, data integration
    - Example 1: A SPARQL query over DBpedia infers that ‘Marie Curie was born in Poland’
      - Which triples justify this?
    - Example 2: A graph neural network labels a node
      - Which edges and nodes influenced the prediction

Why provenance matters in graphs

- Graphs are everywhere
- Queries over graphs produce derived facts
  - But where do they come from?
- Without provenance, graph-derived results are black boxes
- Provenance enables:
  - Trust and quality assessment, reproducibility, debugging, auditing, data integration
- Use cases
  - Science – workflow lineage
  - Web of data – RDF triple trust
  - Fast-checking – trace claims
  - Compliance – audit trails
  - ML/GNN – explain predictions

RDF and knowledge graphs

- RDF graphs as provenance carriers
  - RDF data is naturally a labeled directed graph
    of triples <subject, predicate, object>
  - Named graphs (RDF datasets)
    - Each graph identified by a URL
    - Provenance metadata attached per-graph

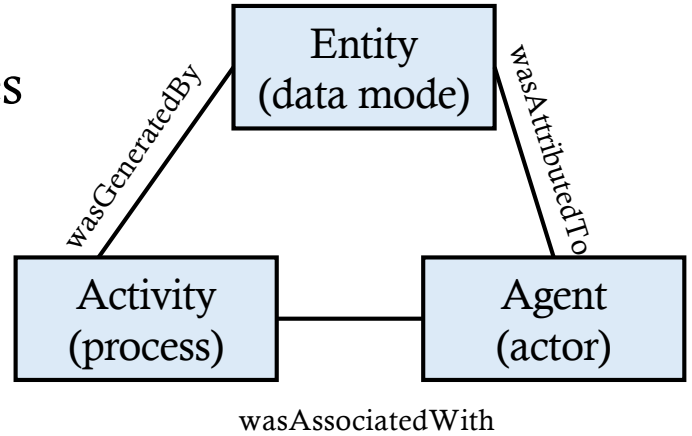


Named Graph Example

```
GRAPH <http://dbpedia.org/en> {
  :Inception :director :Nolan .
  :Nolan :birthPlace :London .
}
GRAPH <http://freebase.org> {
  :Nolan :birthDate "1970" .
}
```

RDF and knowledge graphs

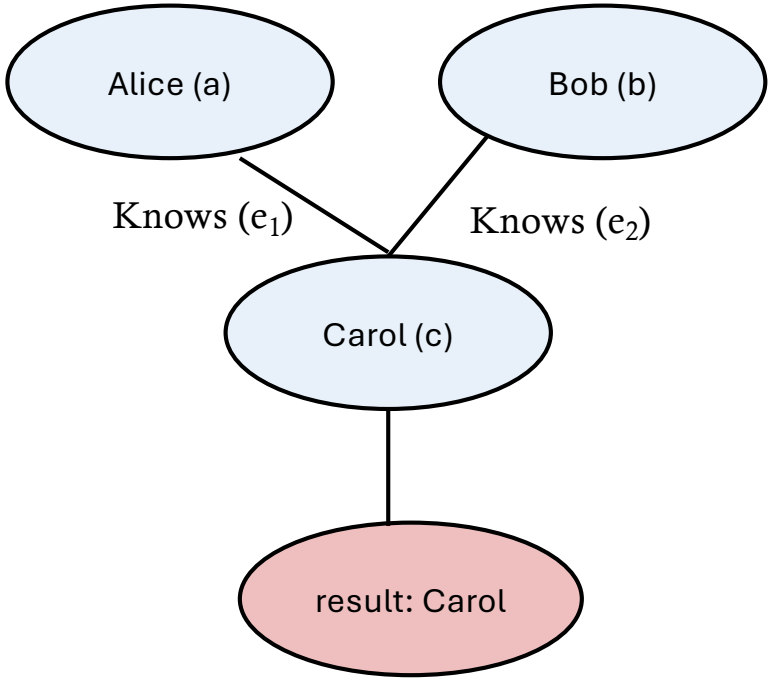
- RDF graphs as provenance carriers
  - RDF* / RDF 1.2
    - Reification allows triples about triples
    - Direct annotation of edges
  - SPARQL queries can track which named graph contributed each result binding
  - W3C PROV (2013)*
    - Standard ontology: entity, activity, agent
    - Maps cleanly to graph nodes/edges



*https://en.wikipedia.org/wiki/W3C_Prov

Provenance for graph queries

- Graph queries go beyond relational
  - Path queries, reachability, pattern matching, recursion
- Datalog on graphs
  - Recursive rules → provenance must handle cycles and infinite derivations
- Conjunctive graph queries
  - Each variable binding corresponds to selecting a subgraph
  - Provenance = selected edges + nodes
- Regular path queries
  - Provenance = all paths matching the regular expression
  - Can be exponential
- GQL (ISO 2024)
  - New standard for property graph queries
  - Provenance semantics TBD, active research area



SPARQL / Datalog equivalent:

$$\text{reachable}(X,Z) :- \text{edge}(X,Y,e_1), \text{edge}(Y,Z,e_2).$$

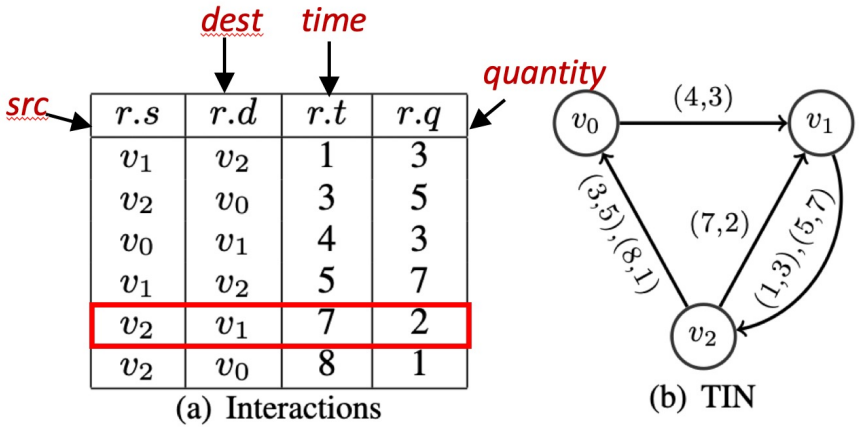
Provenance in graph analytics

- Graph explanations are rarely linear
- Multiple paths may contribute simultaneously
- Provenance often becomes a subgraph extraction problem
- Influence and dependency must be reconstructed

- Applications:
  - Fraud detection
  - Recommendation systems
  - Influence propagation
  - Community detection

Provenance in temporal networks

- Interactions evolve over time *
- Nodes and edges may appear or disappear
- Temporal ordering affects causality
- Provenance must preserve chronology



- In static graphs, provenance may ask:
  - Who is connected?
- In temporal graphs, a provenance question could be:
  - Who interacted with whom, and when?

Temporal ordering becomes part of provenance semantics

* Kosyfaki and Mamoulis, "Provenance in Temporal Interaction Networks", ICDE 2022

Capturing provenance in graph systems

- Architecture layers*
  - Interaction streams
  - Graph construction layer
  - Graph processing engine
  - Temporal provenance store
  - Query/replay
- Captured metadata
  - Edge timestamps, traversal history, propagation events, graph snapshots, update history
- Supported tasks
  - Explanation, anomaly investigation, replay, simulation

Temporal graph provenance becomes a continuously evolving dependency graph

* Interlandi et al., “Titian: Data Provenance Support in Spark” (VLDB 2015)

Storage explosion and summarization

- Provenance can grow faster than the graph itself
- Streaming interaction generate continuous provenance events
- Exact storage quickly becomes infeasible
- Compression ideas:
  - Path summarization
  - Temporal windowing
  - Snapshot compression
  - Hierarchical aggregation

Temporal provenance can grow faster than the graph itself

Takeaways

Summary

- Graphs are a natural substrate for provenance
- RDF and KGs have mature provenance tools
- Graph query provenance is harder than relational
- Temporal ordering changes causality
- Scalability and interpretability remain open problems
- Open directions
  - GQL semantics
  - GNN explainability
  - Scalable compression
  - Privacy-preserving provenance

Tutorial outline

- Provenance in the past
- Provenance now
  - Streaming systems
  - Datalakes
  - Explainability
- Provenance in the future

Provenance in streaming and distributed systems

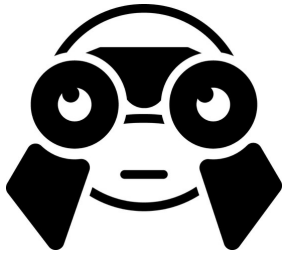
Streaming and distributed systems

- Provenance is not limited to static relational databases
- Modern data systems are:
  - Continuous
  - Distributed
  - Elastic
  - Event-driven
  - Stateful
- Provenance must operate under:
  - High throughput
  - Low latency
  - Fault tolerance
  - Dynamic scaling
  - Out-of-order events

How can we explain, trace, debug, reproduce, and audit continuously evolving computations across distributed infrastructures?

Provenance in streaming systems

- Why provenance becomes hard in streaming systems?
  - Input is unbounded
  - Computation is continuous
  - State evolves over time
  - Windows expire
  - Operators are distributed
  - Outputs may later be revised
- Consequences:
  - Provenance graph can become infinite
  - Need temporal semantics
  - Need compact provenance representations
  - Need incremental provenance maintenance
  - Need online provenance querying



Key observation:

Streaming provenance is fundamentally temporal!!

Provenance in streaming systems

- What does it even mean?
  - Streaming provenance captures:
    - Which events contributed to an output?
    - Which operators transformed the data?
    - Which state influenced the computation?
    - Which temporal windows were active?
    - Which execution nodes processed the data?

Provenance dimensions

- Data provenance
  - Input tuples/events responsible for outputs
- Workflow provenance
  - Operator sequence and execution pipeline
- Temporal provenance
  - Evolution of state and windows over time
- Infrastructure provenance
  - Machine/node/task-level execution metadata

Streaming provenance

- Static provenance:
  - Output *O* depends on input tuples T_1, T_2, T_3
- Streaming provenance:
  - Output *O* depends on:
    - Input events arriving before t
    - Active state at t
    - Window boundaries at t
    - Watermark at t
    - Operator placement at t
  - **We care about:**
    - Event time: timestamp attached to event generation
    - Processing time: timestamp when the system process the event
    - Watermarks: progress indicators for event completeness
    - Window lifecycle: open → accumulate → trigger → close → evict

Provenance for stateful operators

- Stateless operator: provenance is simple

Important insight:

in streaming systems, state becomes a first-class provenance entity

contents and trigger policy

Provenance must capture:

- State snapshots
- State transitions
- Incremental updates
- Window membership
- State expiration

Window-based provenance

- Window types:
  - Tumbling windows: non-overlapping intervals
  - Sliding windows: overlapping intervals
  - Session windows: dynamic user-based intervals

Research challenge:
How to compress repetitive provenance dependencies?

- Provenance explosion:
  - A single event:
    - participates in multiple windows
    - affects multiple aggregates
    - may generate many downstream outputs

Incremental provenance maintenance

- Naïve approach: store complete lineage for every output
  - Problem: impossible at streaming scale
- Incremental provenance: maintains provenance updates incrementally by:
  - Add dependencies for new events
  - Remove expired dependencies
  - Update aggregates incrementally
- **Example:**
  - Window count:

COUNT = previous_count + entering_event - leaving_event

Provenance is also evolves incrementally!!

Provenance and checkpointing

- Streaming systems use checkpoints
- Periodic snapshots of operator state, metadata, and offsets
- Example:
  - Consider Apache Flink
  - Using checkpoints, we can:
    - Save operator states
    - Barriers injected into streams
    - Exactly-once guarantees
  - Checkpoint metadata can support reproducibility, debugging, replay analysis, and rollback provenance

Replay-based provenance

- Instead of storing complete provenance:
  - Store execution logs
  - Replay computations when explanations are needed
- What do we need?
  - Event logs, checkpoints, operators metadata, deterministic replay engine

Replay-based provenance

- Example: A flight-booking application suddenly crashes for a user
  - Provenance log: the system records the sequence of clicks: Search flight → Select seat → Apply discount code
  - Replay: a developer feeds that exact 3-step sequence into a test environment to replicate the crash
  - Fix: the developer discovers the crash happens because the discount code step fails when a specific seat type is chosen

Instead of saving the entire system memory, the system only saved the small log of events to recreate the error

Fine-grained provenance

- Refers to the detailed tracking of data
- Tracing an end product back through every specific transformation, input, and processing step at the most granular level
  - E.g., cell, row, or file level
- How it works?
  - Systems record the “how”, “where”, and “why” by logging every action in a database or lineage graph
    - **Example:** How-provenance determines which query expressions evaluated selected pieces of data, while why-provenance maps exactly which input tuples contributed to a specific output row
- Limitations:
  - Overhead in metadata, computation, and storage

Coarse-grained provenance

- Refers to the high-level tracking of data and system histories at the level of entire files, whole datasets, or complete software updates
- How it works?
  - Instead of tracking individual data cells or lines of code, it records broad relationships between major inputs, processing steps, and outputs
- Limitations:
  - Lack of detail creates a major visibility gap
    - If a small error or malicious injection occurs inside a dataset, coarse-grained tracking can only tell you which file is corrupted
      - User must manually audit the entire file to find the root cause

Streaming provenance-based systems

GeneaLog: Fine-Grained Data Stre

Dimitris Palyvos-Giannas
Chalmers University of Technology
Gothenburg, Sweden
palyvos@chalmers.se

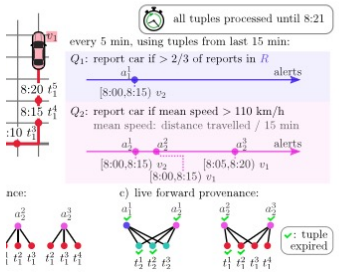
Vincenzo G
Chalmers University
Gothenburg,
vinmas@chal

Ananke: A Streaming Framework for Live Forward Provenance

Dimitris Palyvos-Giannas
Chalmers Univ. of Technology
Gothenburg, Sweden
palyvos@chalmers.se

Bastian Havers
Chalmers Univ. of Technology, Volvo Car Corporation
Gothenburg, Sweden
havers@chalmers.se

Vincenzo Gulisano
almers Univ. of Technology
vinmas@chalmers.se



o sample queries to monitor car location and ll tuples up to time 8:21 are processed), and rd and c) live forward provenance graphs.

ming applications, or *queries*, monitoring a vehic- spot cars visiting a specific area (Q_1) or speeding rts (timestamp, id, position), or *tuples*, arrive every e i-th tuple from car j, a_j^i the i-th alert from query is our running use-case throughout the paper.

nge. CPSs need continuous monitoring for emerg- ngerous events [32, 37], which may result in many ts are then left to prioritize [15, 34]. For streaming- *rovenance* techniques [19, 30], which connect re- ntributing data, are a practical way to inspect ies, since breakpoint-based inspection is not fit that run in a distributed manner and cannot be sting provenance tools for traditional databases *tracing*, to find which source tuples contribute , 12, 14, 19] (Figure 1b), and *forward tracing*, to ts originate from a source tuple [7, 12]; however, tools only exist for backward-provenance [19, 30]. live, streaming, forward provenance is multifold: ard tracing can give assurance on the trustwor- esults [11], forward tracing allows to identify all specific inputs [12], e.g. to mark all results linked sitive datapoint (e.g. a picture of a pedestrian, in hicular Networks) before such results are analyzed naintenance of the provenance graph avoids data allows to start the analysis of provenance data all sensitive results that could be connected to the

LPStream: Fine-grained Lazy Provenance for Stream Processing

MASAYA YAMADA*, University of Tsukuba, Japan and National Institute of Advanced Industrial Science and Technology, Japan
HIROYUKI KITAGAWA, University of Tsukuba, Japan and National Institute of Advanced Industrial Science and Technology, Japan
SALMAN AHMED SHAIKH, National Institute of Advanced Industrial Science and Technology, Japan
TOSHIYUKI AMAGASA, University of Tsukuba, Japan
AKIYOSHI MATONO, National Institute of Advanced Industrial Science and Technology, Japan

Stream processing enables real-time data analysis. Recent stream processing engines (SPEs) execute stream processing in a distributed manner for real-time analysis of massive amounts of data produced by IoT devices and sensors. It has been widely adopted in various applications that support critical decision making. To explain the results of stream processing, ensuring provenance is indispensable. Provenance clarifies the relationship between input data and output data in the processing. With provenance, we can understand what input data contributed to the output. Existing frameworks for providing provenance for stream processing generate provenance or additional information to construct provenance at runtime. However, these approaches impose substantial overhead in ordinary stream processing. In this paper, we propose a new framework, named LPStream, for fine-grained lazy provenance. LPStream is the first framework to support lazy provenance for stream processing. In the ordinary execution mode, LPStream executes stream processing with checkpointing but without provenance generation. If provenance is necessary for some target output tuples, it replays the processing from an appropriate checkpoint and generates the provenance for the target tuple. We explain the design and implementation of LPStream and evaluate its performance by comparing LPStream with stream processing without provenance and with eager provenance. The experimental results demonstrate the effectiveness of our proposal.

CCS Concepts: • **Information systems** → **Data provenance**; **Stream management**.

Additional Key Words and Phrases: Lazy Provenance, Stream processing, Checkpointing, Apache Flink

ACM Reference Format:
Masaya Yamada, Hiroyuki Kitagawa, Salman Ahmed Shaikh, Toshiyuki Amagasa, and Akiyoshi Matono. 2025. LPStream: Fine-grained Lazy Provenance for Stream Processing. *Proc. ACM Manag. Data* 3, 4 (SIGMOD), Article 254 (September 2025), 25 pages. https://doi.org/10.1145/3749172

ems was (DEBS

Provenance

scher
eiburg
er@
rg.de

Nesime Tatbul
ETH Zurich
Switzerland
tatbul@inf.ethz.ch

ng. These detected events are then used for as well as for notifying human supervisors. eed to understand why and how such events ble to assess their relevance and react appro- ws a simplified example of a continuous query ng. Two sensors feed timestamped tempera- ery. Each sensor stream is filtered to remove temperature t above 350°C). The stream is ng the temperature over a sliding window of 3 to further reduce the impact of sudden spikes. eps are applied to each sensor stream individ- dings from multiple sensors are combined for a union followed by a sort operator to glob- he final aggregation and selection ensure that raised if at least three different sensors show above 90°C within 2 time units. In this ex- d want to understand which sensor readings g" alarm event, i.e., determine the tuples that *ined* provenance of this event.

portunities: Tracking provenance to explore to a given query result has proven to be an ty in many domains such as scientific work- l relational databases [10]. However, provid- nance support over data streams introduces a allenges that are not well addressed by tradi- nagement techniques:

Data Arrival: Data streams can potentially no global view on all items is possible. As ethods that reconstruct provenance from the on request are not applicable.

l: In contrast to relational data, data streams l as ordered sequences. This ordering can be ptimized representations of provenance.

essing: In DSMSs, operators like *aggrega-* ically processed by grouping tuples from a Stream provenance must deal with window- to trace the outputs of such operators back to y and efficiently. The prevalence of aggrega- us amounts of provenance per result.

: Performance requirements in most stream- rict; in particular low latency should be main- generation has to be efficient enough to not n's latency constraints.

Mechanisms for coping with high input rates 22, 24]) and certain operator definitions such tem time result in outputs that are not deter nputs. Conventional provenance management

Takeaways

Summary

- Provenance in streaming systems is temporal
  - Outputs depend on evolving state and windows
- Scalability is the central challenge
  - Fine-grained provenance can become enormous
- Provenance is becoming foundational
  - Useful for:
    - Debugging
    - Explainability
    - Auditing
    - Reproducibility

Provenance and datalakes

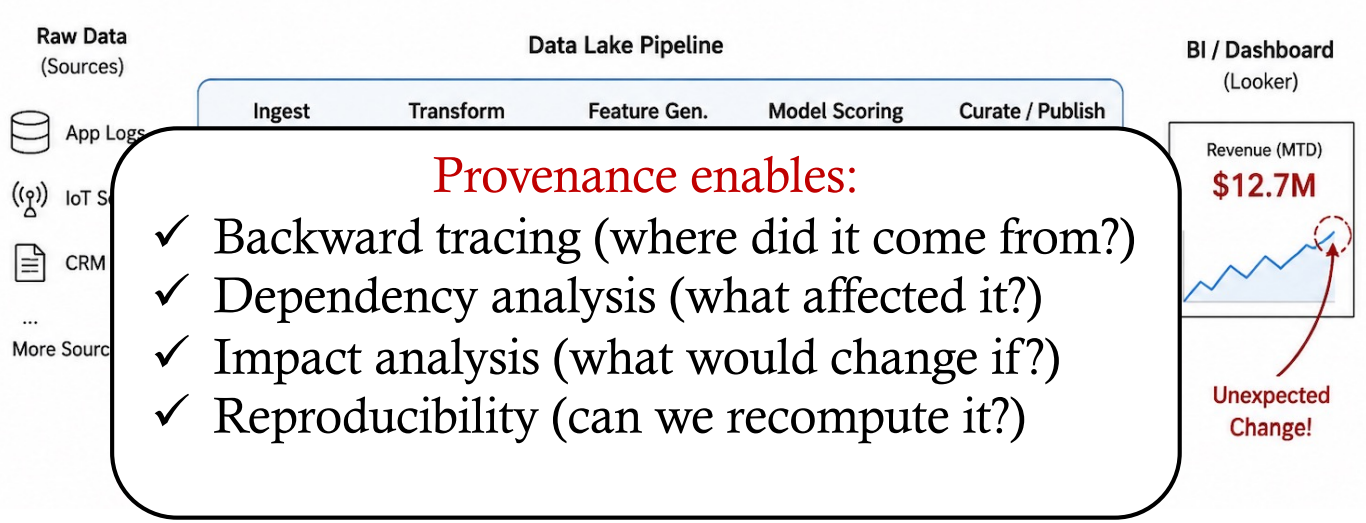
Data lakes – quick recap

- Data warehouse
  - Schema-on-write
  - Structured data only
  - ETL before loading
  - High governance
  - Slow to ingest
- Data lake
  - Schema-on-read
  - Any format
  - Load raw, transform later
  - Low upfront governance
  - Risk: becomes a data swamp

Without provenance, a data lake quickly turns into a data swamp (e.g., data of unknown origin, quality, and meaning)

Provenance and data lakes

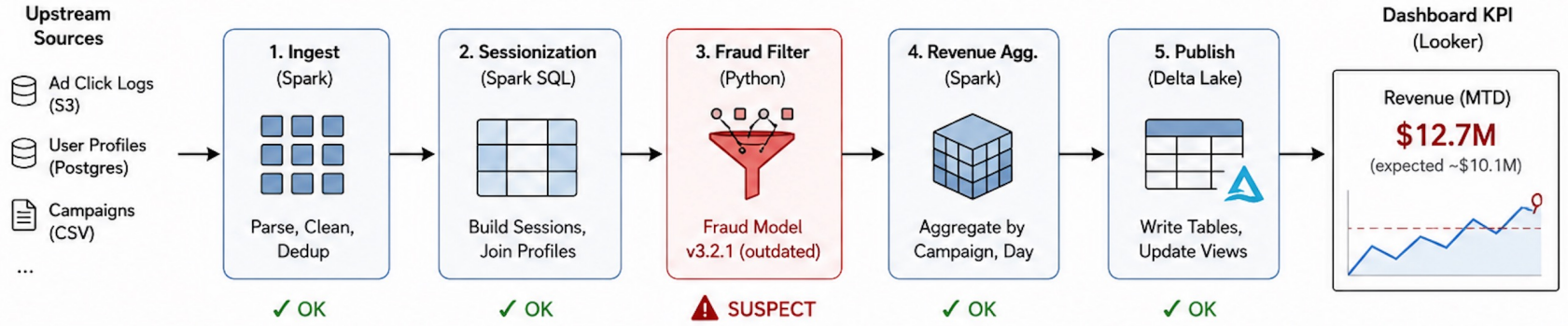
- Motivation: the wrong dashboard number
  - Answering "Why did this number change?" is a provenance problem

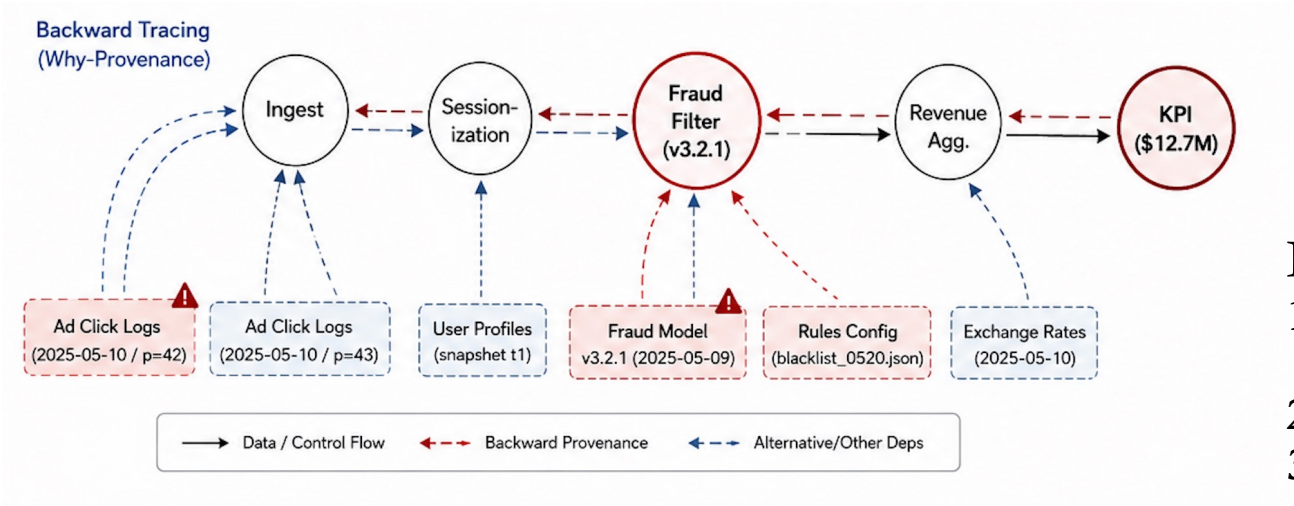


- Questions:
  - Which input data or upstream change caused this number to change?
  - Which transformations, code/logic, parameters, or model version were involved?
  - What is the impact if we roll back or fix a specific component?
  - Can we reproduce the old number exactly?

A realistic failure scenario

- Scenario: tracing the wrong KPI backwards
  - Provenance will help us find possible causes and their impact





Possible root causes:

1. Outdated fraud model (model changed on 2025-05-11)
2. Corrupted partition in click logs
3. Missing blacklist update

Debugging the lake requires reconstructing derivation paths

From classical provenance to data lakes

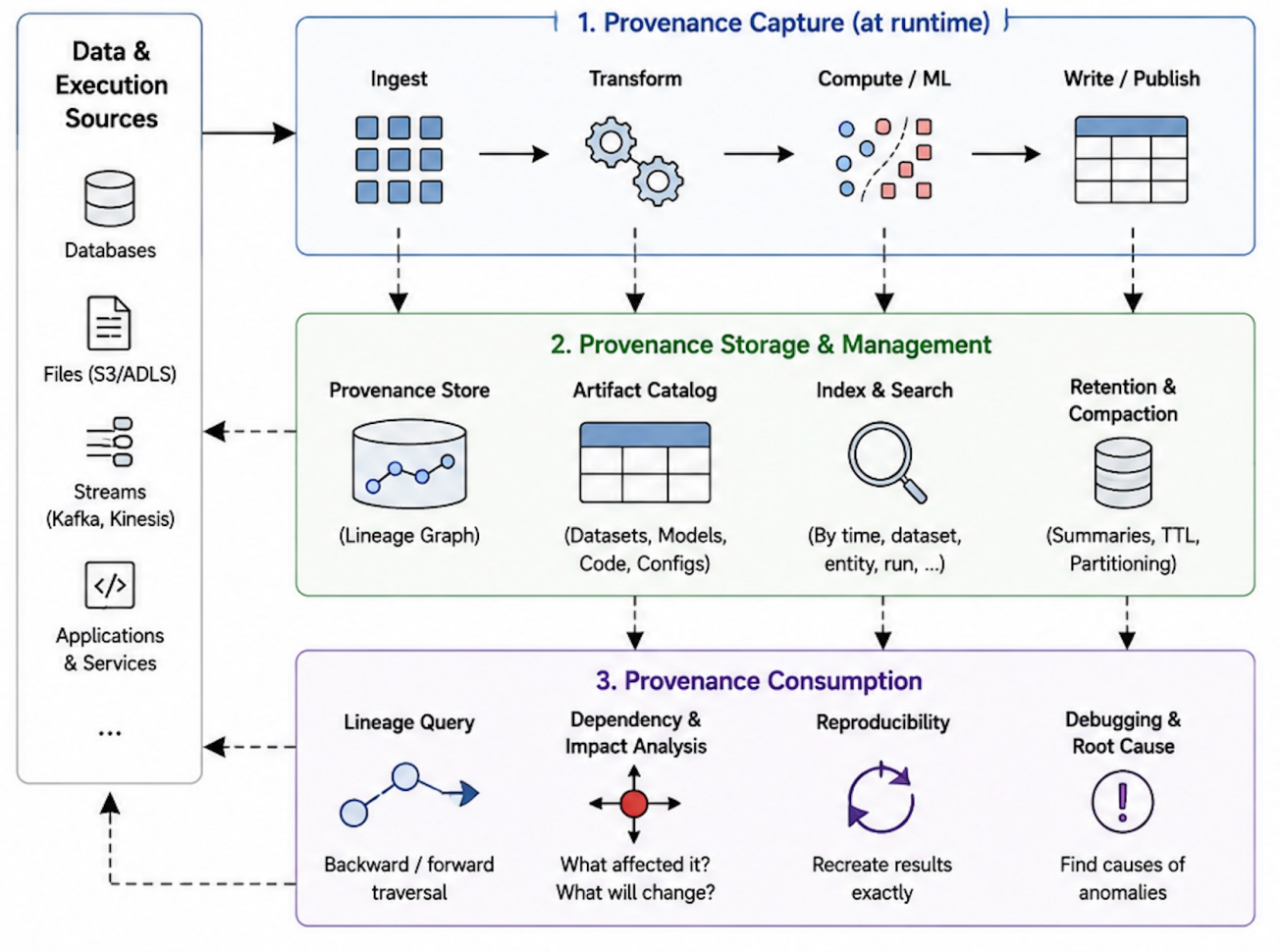
- Classical provenance focused on:
  - Why-provenance
  - Where-provenance
  - How-provenance
  - Fine-grained relational lineage
- Operators have well-defined semantics
  - Modern data lakes violate this assumption
  - Modern heterogeneous pipeline:
    - SQL engine
    - Spark
    - Notebooks
    - UDFs
    - ML pipelines
    - External services

Modern provenance must reason across partially understood transformations

Provenance and datalakes

Model	How it works	Fits lakes?
Where-provenance	Which source cell contributed to an output cell	Partial (row/column level only)
Why-provenance	Algebraic annotation tracking all derivations	Extends to semi-structured
How-provenance	Multiplicity and combination operators	Strong formal foundation
W3C PROV-DM	Entity-activity-agent graph OWL2 ontology	Widely adopted in practice
Lineage DAG	Directed acyclic graph of data assets and transforms	Apache Atlas, DataHub

Capturing provenance in a lake



- Operators / jobs
- Inputs & outputs
- Parameters and code
- Versions & configs
- Time, actor, location
- Execution context

- Scalable graph storage
- Versioned artifacts
- Efficient indexes
- Lifecycle management

- Ad-hoc exploration
- API/SQL interfaces
- Visualizations
- Alerts & explainability

- Treat provenance as first-class infrastructure
- Enables trust, debuggability, impact analysis, and reproducibility in data lake systems

An (Updated) Overview of Data Provenance: Concepts, Challenges and Opportunities

Provenance beyond relational pipelines

- Provenance questions:
  - Which data contributed to this prediction?
  - Which feature transformation changed?
  - Which model version consumed this dataset?
  - Which downstream predictions are affected?

Modern ML pipelines extend provenance beyond relational dataflow

Open research problems

- Scale:
  - Billion-edge lineage graphs
  - Streaming provenance
  - Online compression
- Semantics:
  - Black-box operators
  - Probabilistic provenance
  - Approximate lineage
- Usability:
  - Querying provenance
  - Visual debugging
  - Explanation interfaces

We still lack a scalable and universal provenance abstraction

Takeaways

Summary

- Lakes increase provenance complexity
- Granularity is the key trade-off
- Black-box transformations remain difficult
- ML pipelines expand provenance scope

Future data systems will treat provenance as a first-class data product

Provenance and explainability

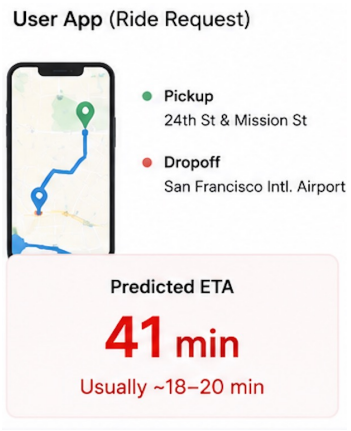
Explainability became a system problem

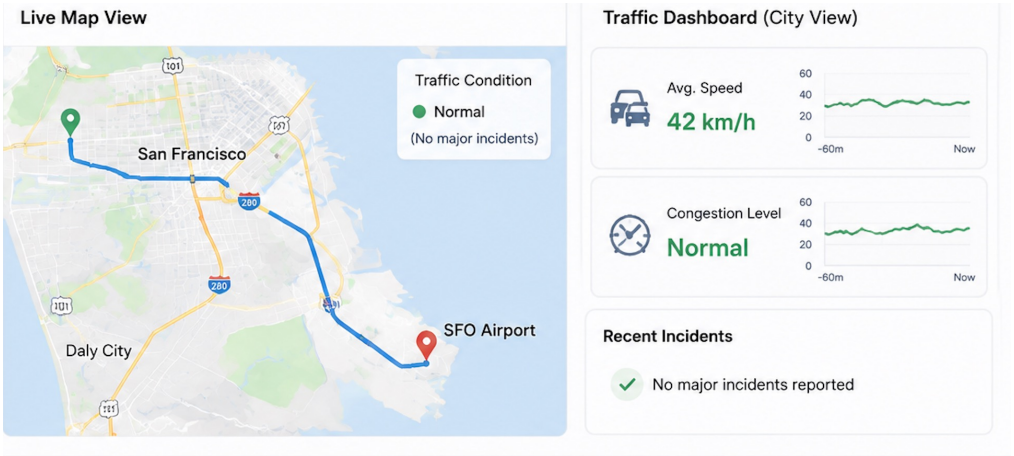
- Traditional ML
  - Static dataset → Model → Prediction
- Modern data systems
  - Streams
    - Feature stores, online aggregation, APIs, distributed state
    - External services → prediction

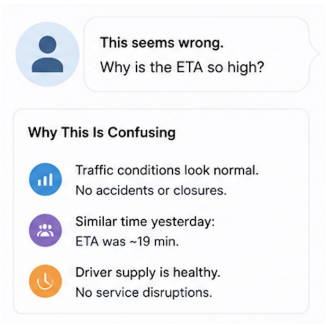
Modern predictions increasingly depend on dynamic computational workflows

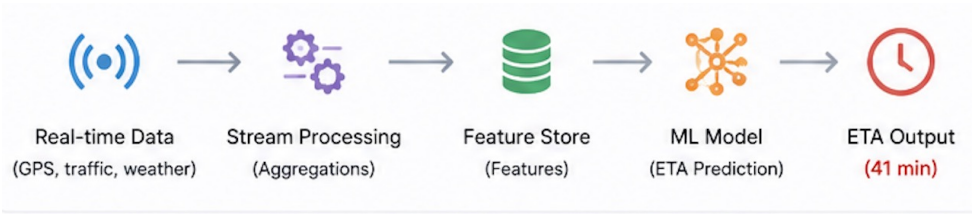
Explainability

- Example:









Why did the system predict 41 minutes? 😖

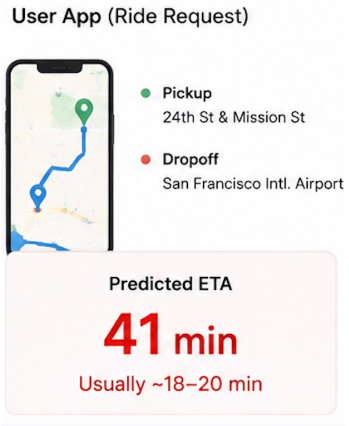
What exactly are we trying to explain?

Prediction explanation	System explanation	Causal explanation
• Why did the model output 41 minutes?	• Which component produced the incorrect estimate?	• Which event caused the ETA to increase?

Different explanations answer fundamentally different questions!

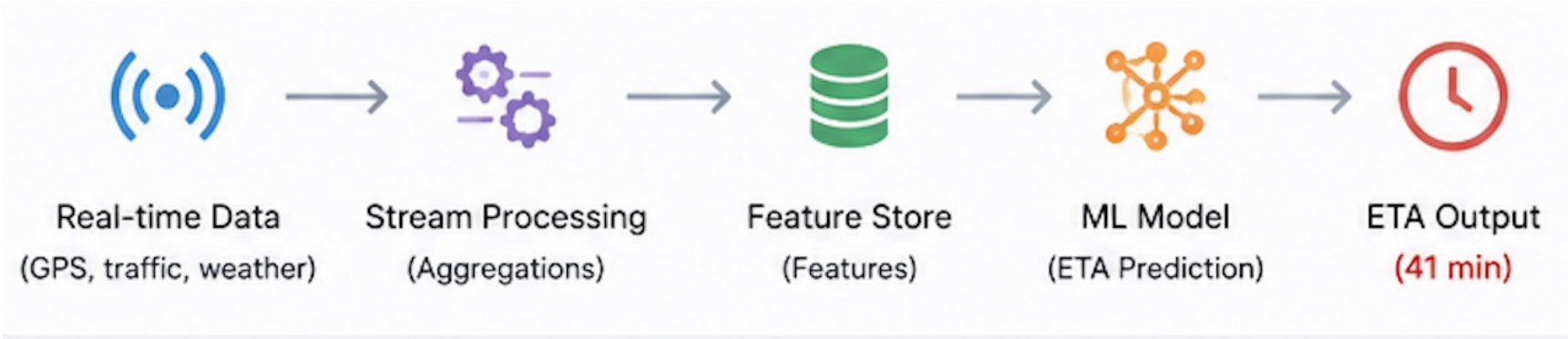
Explainability starts after the pipeline

- Important factors:
  - Congestion estimate
  - Route density
  - Average speed
- Typical explanation:
  - “High congestion increased the ETA”
  - But who explained the congestion feature itself?



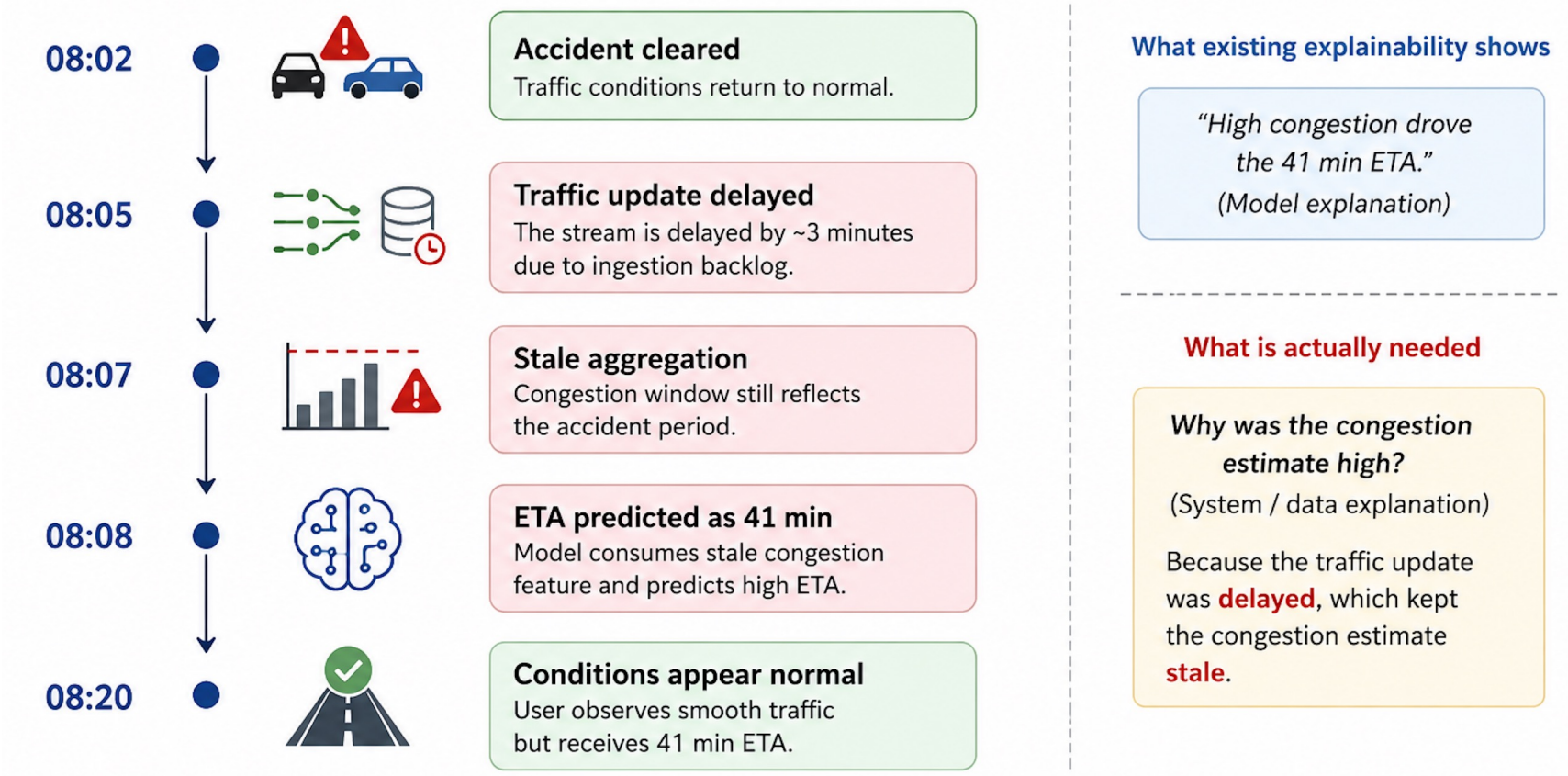
- Most explainability assumes the inputs are already correct!

Modern predictions come from pipelines



- The prediction depends on the entire computational workflow.

Reconstructing the failure



The model behaved consistently. The pipeline state did not!

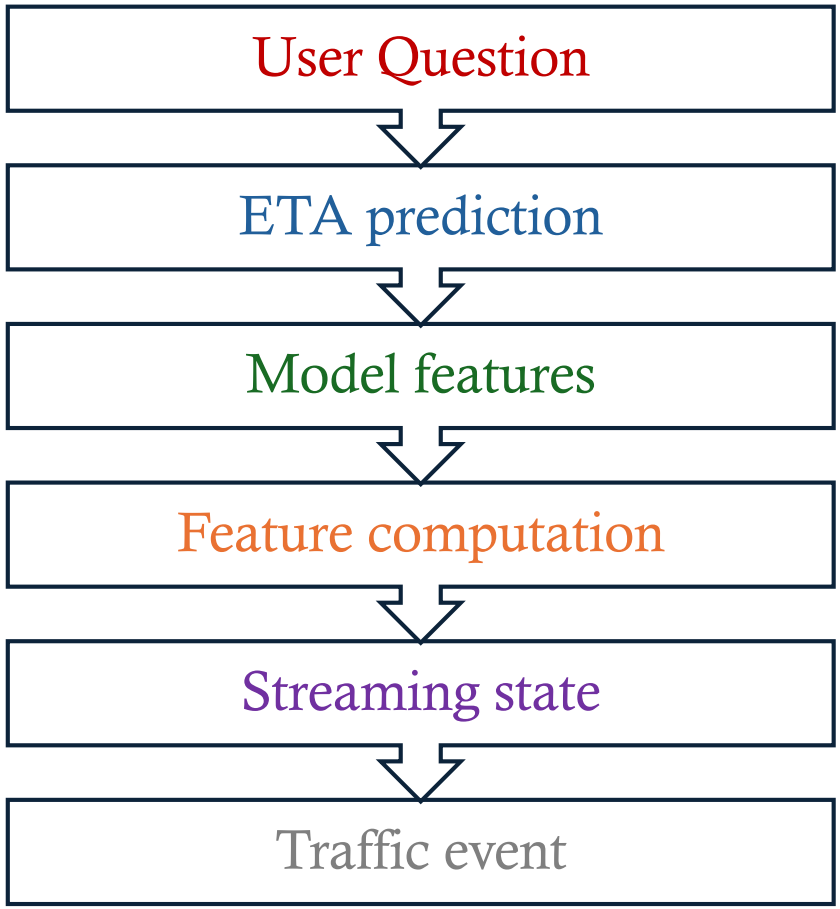
Existing explanations become incomplete

- Model explanation: high congestion increased the ETA
- Missing information:
  - Why was congestion high?
  - Which data source caused it?
  - Which computation propagated it?
  - Was the evidence stale?
  - Which system component introduced the error?

- The explanation sounds reasonable. But it cannot reconstruct the actual failure

Provenance for the rescue

- Provenance reconstructs
  - Derivation
  - Propagation
  - Dependency
  - Temporal evolution



Provenance links user-visible outcomes
to hidden system computations

Provenance and explainability

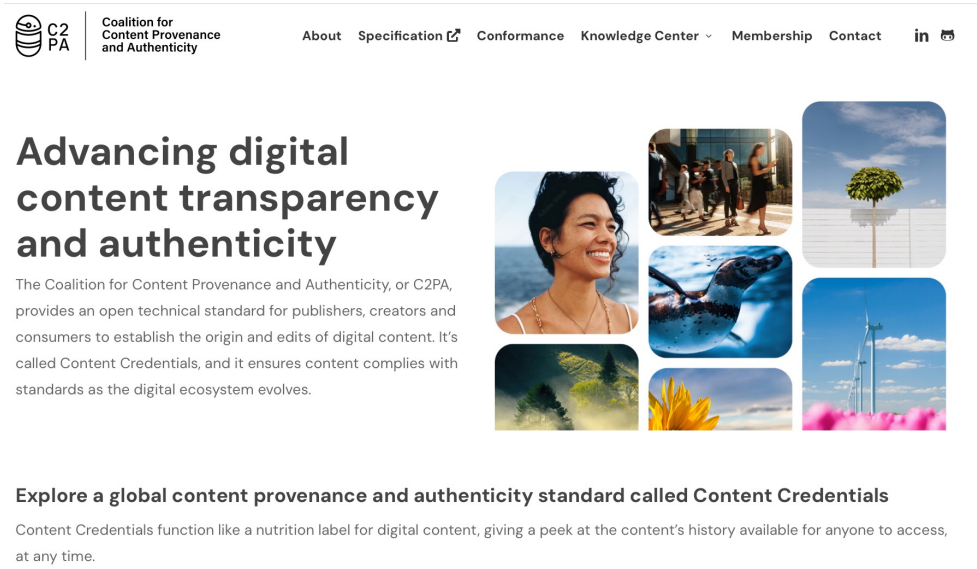
Explainability	Provenance
• Interpretation • Trust • Justification • Human understanding	• Dependency • Derivation • Evidence propagation • Causal reconstruction

Provenance → faithful explanations

Explainability describes decisions
Provenance explains how those decisions came to exist

Provenance is needed for GenAI

- Emerging need for communicating provenance in GenAI
- C2PA new standard for describing provenance on the web.
- Adopted by many organizations



Takeaways

Summary

- As systems become more autonomous, explanations increasingly require provenance
- Modern explainability often explains predictions
- Provenance explains how those predictions came to exist
- Trustworthy explanations require computational evidence

Outline

- Provenance in the past
- Provenance now
- Provenance in the future
  - Open challenges and future directions

Summary of Open Challenges

- Provenance in complex data ecosystems with agents
  - Systems for explaining large complex systems with many dependencies
  - Integrating provenance capturing and querying both inside and outside the database engine
  - Supporting emerging GenAI provenance standards
  - Benchmarks that represent complex data systems
- Fundamentals
  - Systems that support provenance as a first class citizen
  - Provenance-aware analytical engines
  - Richer theoretical understanding of provenance in graph queries
- Scalability

Thank you ☺