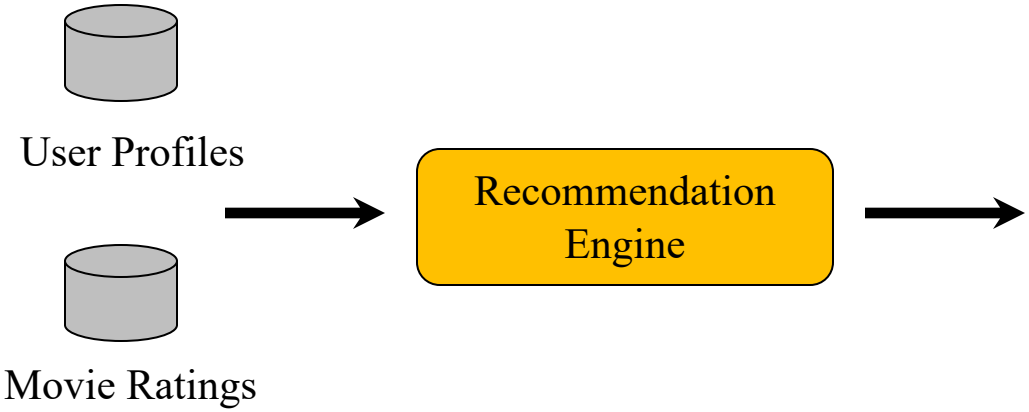
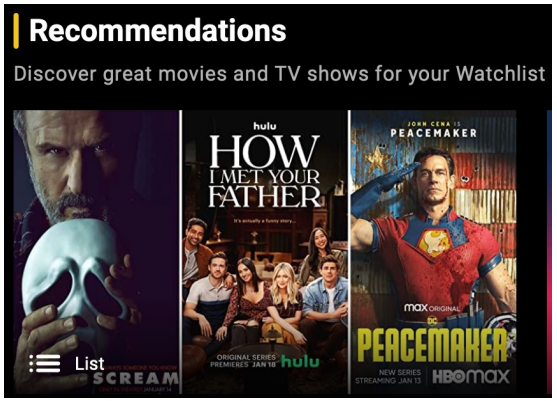
Data Provenance

- Definitions and Applications
- Core concepts:
  - Data lineage and data provenance
  - Data causality
- Techniques to generate ‘explanations’:
  - Backward tracing
  - Metadata propagation

Data Provenance
Motivation & Applications

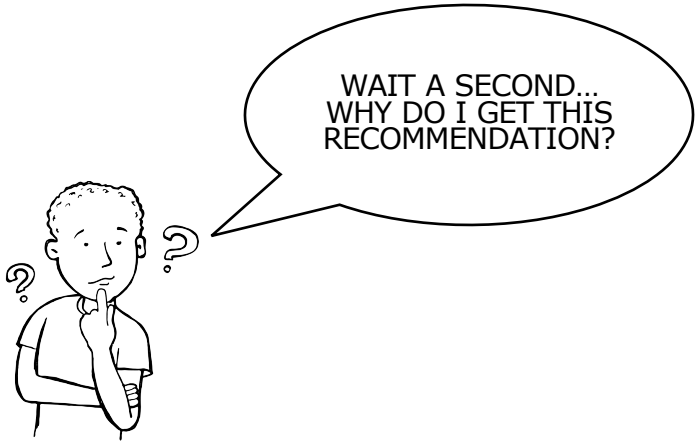
A Recommender System Example



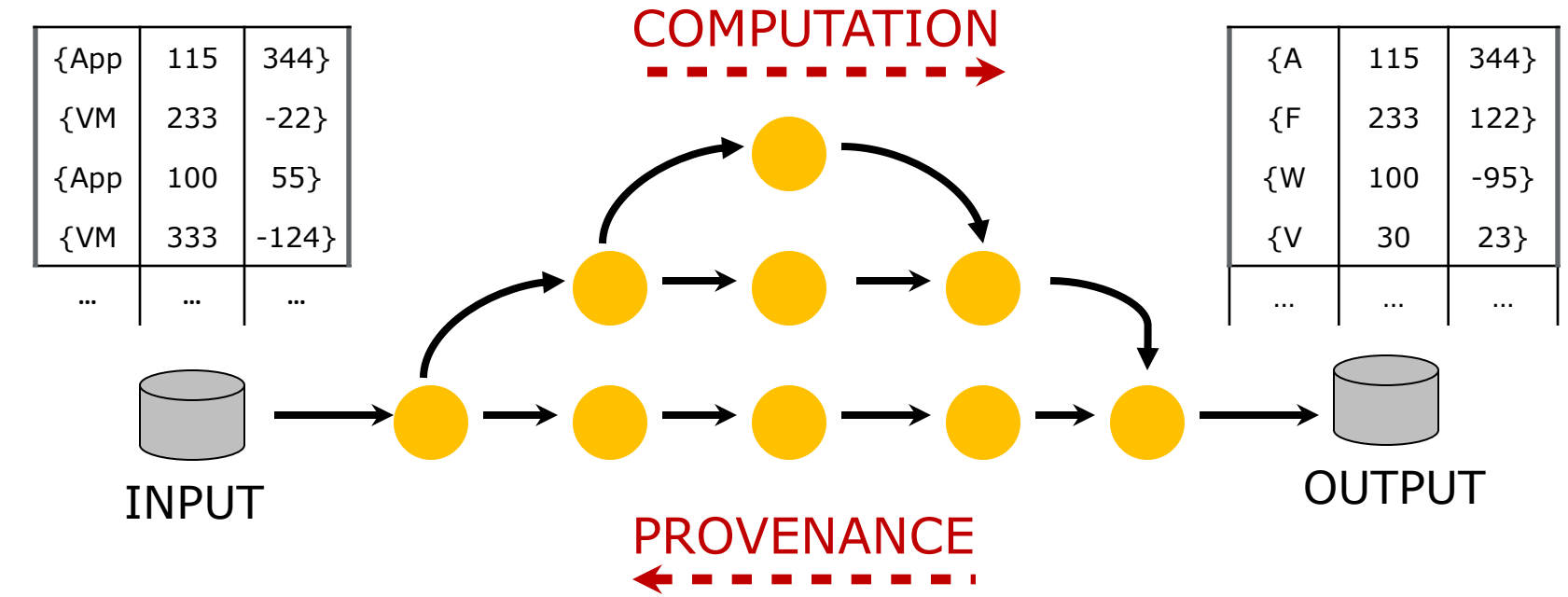


Stars	#Friends
*	0
**	2
***	7
****	23
*****	8

Explanation



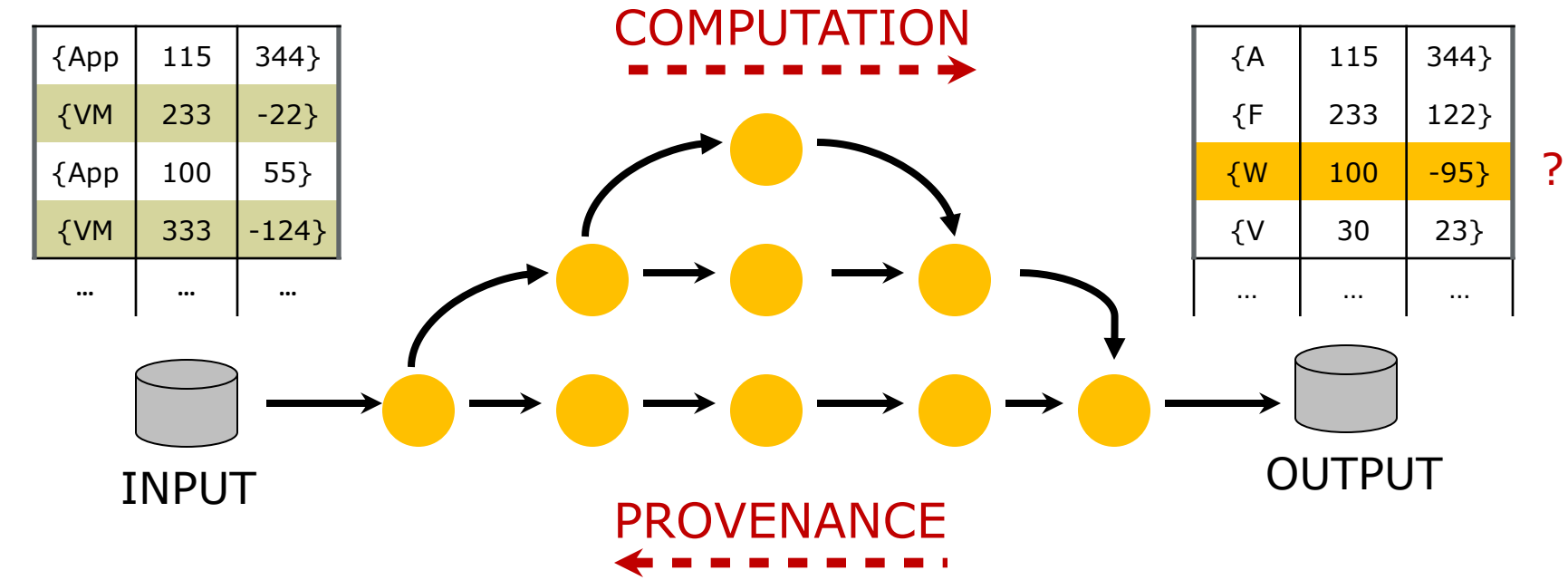
A Database System Example



Data operator

Flow of data

A Database System Example



Data operator

Flow of data

Provenance Questions in DBs

- Where does a particular output value come from?
- What computation steps took place in producing an output?
- Why is a particular tuple not included in the result?
- How much does a tuple contribute to producing a certain output?
- …

Impact of Provenance

- Explanations build trust and increase user acceptance
- Part of user rights in GDPR

GDPR's Right to Explanation: the pros and the cons

Corporate · algorithm · GDPR · PredPol · right to explanation

Under the General Data Protection Regulation's Right to Explanation a user can ask for an explanation about an algorithmic decision made about them – but with it comes positives and negatives.

22/05/2017 BY: BILL BRENNER

Provenance Application Areas

AUDITING

SECURITY

DEBUGGING

REPRODUCIBILITY

DATA VISUALIZATION

EXPLANATIONS

Provenance Application Examples

- Supply chains: In 2013, EU beef products were discovered to contain horsemeat. Labeling the origin of food and documenting its processing and supply chain can find the reasons behind and prevent such scandals.
- Scientific experiments: In the ATLAS experiment (CERN), the first run of data acquisition gathered 100 PB of data. The ATLAS collaboration used provenance mechanisms to ensuring the reproducibility and accessibility of the scientific results, as experiments of this scale are not repeatable.
- Complex data processing: Developers write code, test it, analyze the result, refine the code (or clean the data), test again… The collection and analysis of provenance in this context can lead to faster and higher-quality analysis, debugging, and refinement of data processing.

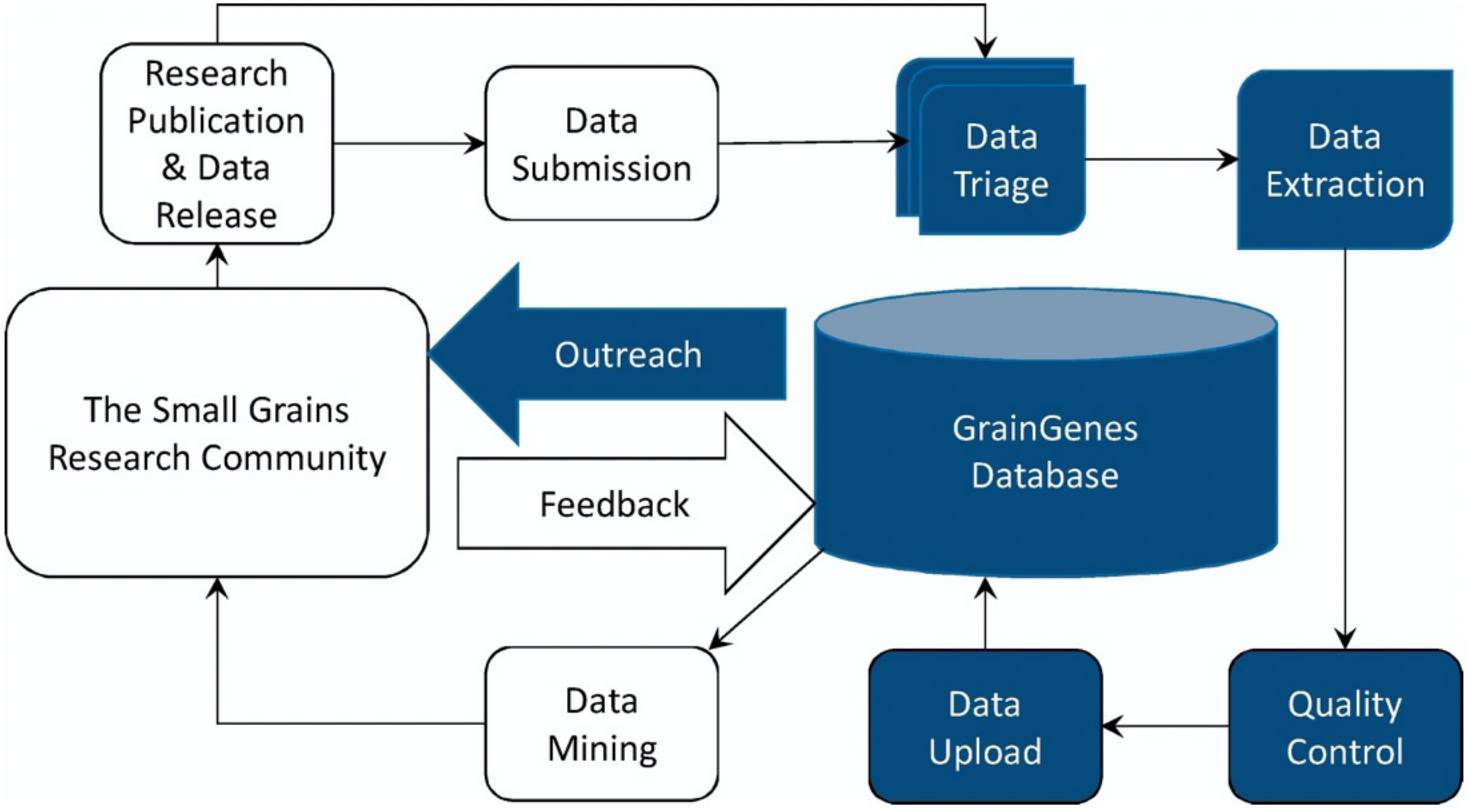
Data Provenance Concepts

Data Lineage vs Data Provenance

- **Data lineage** includes the data origin, what happens to it and where it moves over time.
  - gives visibility and simplifies the ability to trace errors back to the root cause in a data analytics process.
  - enables replaying specific portions or inputs of the data flow […] Database systems use such information, called data provenance, to address similar validation and debugging challenges.
- **Data provenance** refers to records of the inputs […] that influence data of interest […], providing a historical record of the data and its origins.
  - The generated evidence supports data-dependency analysis, error/compromise detection and recovery, auditing, and compliance analysis.
- "Lineage is a simple type of why-provenance."

Source: Wikipedia

Use case: Curated Databases



[S. G. Odell et al. *The art of curation at a biological database: Principles and application*, Current Plant Biology, Vol 11-12, pages 2-11, 2017.]

Data Curation

- The organization and integration of data collected from various sources.
  - annotation, publication and presentation
  - the value of the data is maintained over time
  - the data remains available for reuse and preservation
- Includes "all the processes needed for principled and controlled data creation, maintenance, and management, together with the capacity to add value to data".
- In science, data curation may indicate the process of extracting important information from scientific texts, such as research articles by experts, to be converted into an electronic format, such as an entry of a biological database.

Source: Wikipedia

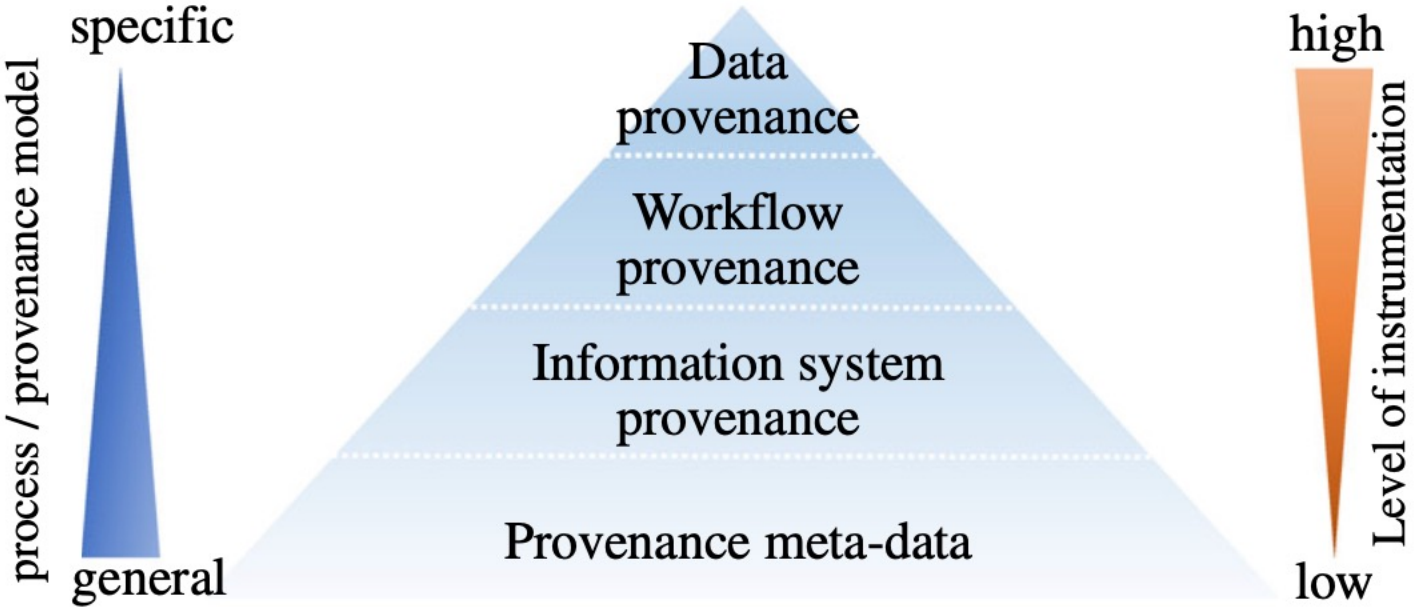
Data Curation (cont'd)

- A core task in data-driven scientific workflows
  - Helps maintaining high-quality data
  - Enables reproducibility
- Data and metadata organization
  - Annotations about data sources, changes, owners, etc.
  - Code, workflow specification, experiment parameters
- A continuous process
  - Data and metadata change
- A collaborative task
  - Researchers, lab personnel, developers, database admins, …

Provenance Questions in Curated DBs

- What is the original source of this dataset?
- Who changed these data values and when?
- How did the database look like on 09.17.2002?
- Where can I find the datasets and the code used in experiment X?
- What changes have been made in the database over the last week?
- How was this dataset produced?

Provenance Hierarchy

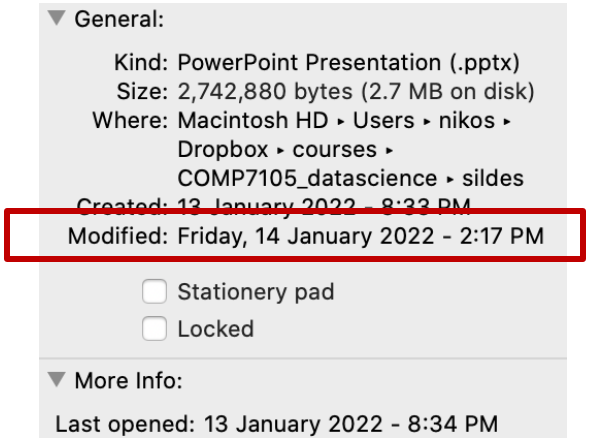


M. Herschel et al. *A survey on provenance: What for? What form? What from?*, VLDBJ 26:881–906, 2017

Provenance meta-data

- The simplest and most general form of provenance about a production process
- Typically unstructured or semi-structured
- Easy to generate and maintain
- Often coarse-grained, so not very informative

- Example: meta-data about a file
  - When was it last modified
    - Not very helpful if you want to track who changed a particular slide in the presentation and when



Information System provenance

- Meta-data about information-disseminating processes producing digital data within information systems
- Provenance collection may exploit the input, the output, and parameters of a process
- Computed based on the input, the output, and the parameters of the process.

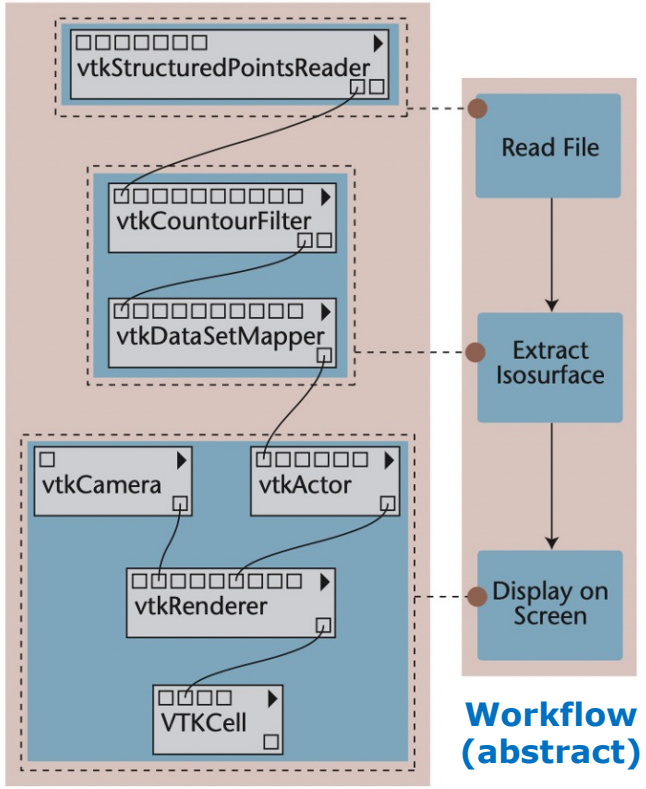
Workflow provenance

- Workflow provenance = Info. Sys. provenance restricted to production processes in the category of workflows
- Workflow is a directed graph
  - nodes represent arbitrary functions/modules
  - nodes have some input, output, and parameters
  - edges model a predefined dataflow or control flow between modules

Workflow Provenance (example 1)

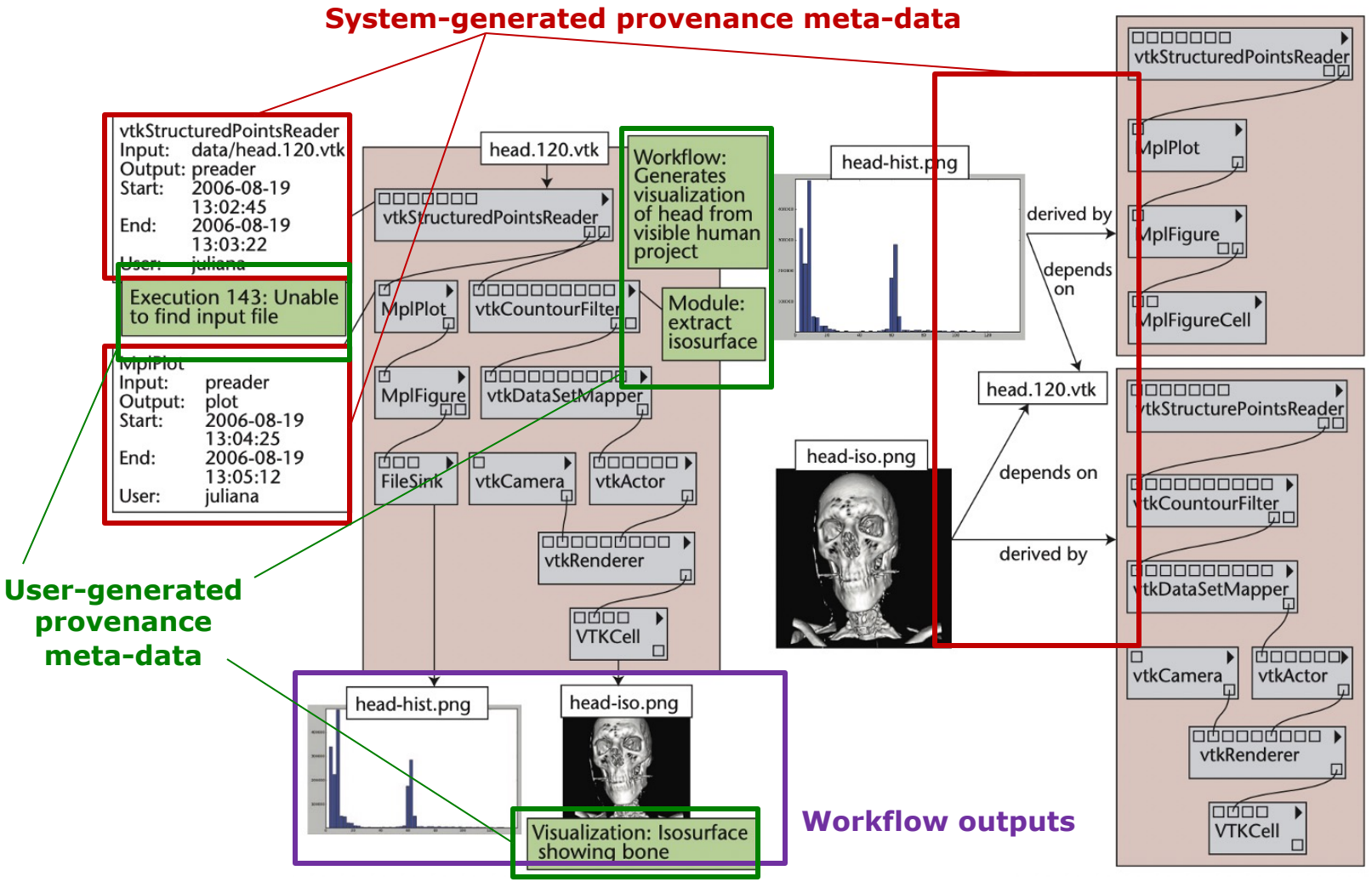
```
1  import vtk
2
3  data = vtk.vtkStructuredPointsReader()
4  data.setFileName("../../../examples/data/head.120.vtk")
5
6  contour = vtk.vtkContourFilter()
7  contour.SetInput(0, data.GetOutput())
8  contour.SetValue(0, 67)
9
10 mapper = vtk.vtkPolyDataMapper()
11 mapper.SetInput(contour.GetOutput())
12 mapper.ScalarVisibilityOff()
13
14 actor = vtk.vtkActor()
15 actor.SetMapper(mapper)
16
17 cam = vtk.vtkCamera()
18 cam.SetViewUp(0,0,-1)
19 cam.SetPosition(745,-453,369)
20 cam.SetFocalPoint(135,135,150)
21 cam.ComputeViewPlaneNormal()
22
23 ren = vtk.vtkRenderer()
24 ren.AddActor(actor)
25 ren.SetActiveCamera(cam)
26 ren.ResetCamera()
27
28 renwin = vtk.vtkRenderWindow()
29 renwin.AddRenderer(ren)
30
31 style = vtk.vtkInteractorStyleTrackballCamera()
32 iren = vtk.vtkRenderWindowIneractor()
33 iren.SetRenderWindow(renwin)
34 iren.SetInteractorStyle(style)
35 iren.Initialize()
36 iren.Start()
```

Python program



J. Freire et al. Provenance for Computational Tasks: A Survey, Comput. Sci. Eng. 10(3), 11–21 (2008)

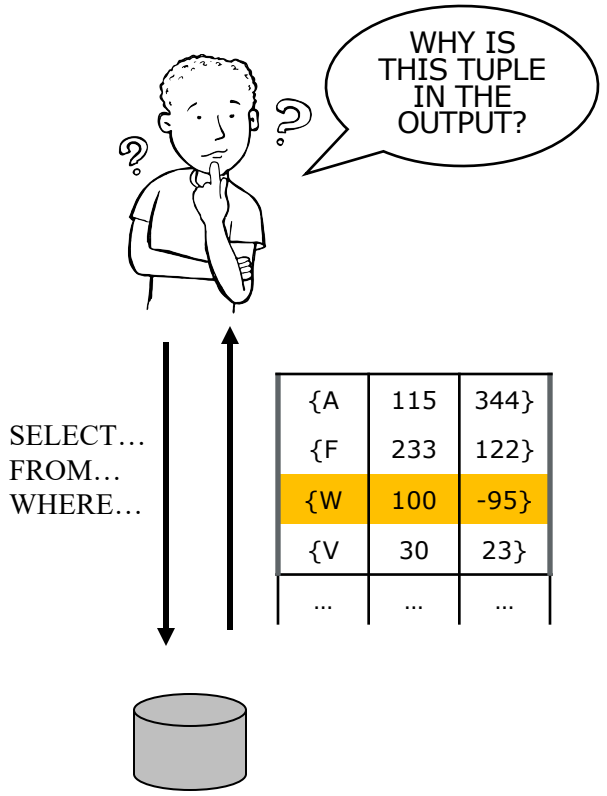
Workflow Provenance (example 2)



J. Freire et al. Provenance for Computational Tasks: A Survey, Comput. Sci. Eng. 10(3), 11–21 (2008)

Data(base) Provenance

- Tracks individual data items (e.g., tuples) at the "highest resolution"
- Well defined semantics
- Fine-grained
- Database provenance:
  - Information about individual tuples or values in the output of SQL queries



An analogy: program slicing

```
...
int i;
int sum = 0;
int product = 1;
int w = 7;
for(i = 1; i < N; ++i) {
        sum = sum + i + w;
        product = product * i;
}
...
write(sum);
write(product);
...
```

An analogy: program slicing

Where does the value of **product** come from and which part of the program affects it?

```
...
int i;
int sum = 0;
int product = 1;
int w = 7;
for(i = 1; i < N; ++i) {
        sum = sum + i + w;
        product = product * i;
}
...
write(sum);
write(product);
...
```

An analogy: program slicing

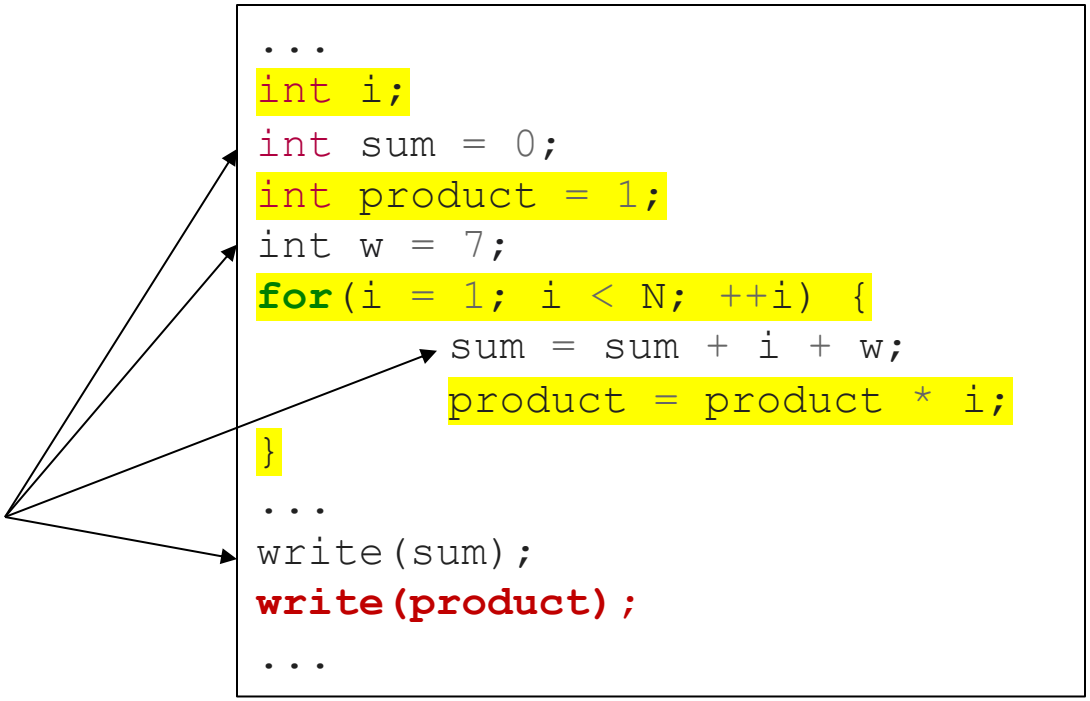
Where does the value of **product** come from and which part of the program affects it?

```
...
int i;
int sum = 0;
int product = 1;
int w = 7;
for(i = 1; i < N; ++i) {
        sum = sum + i + w;
        product = product * i;
}
...
write(sum);
write(product);
...
```

An analogy: program slicing

Where does the value of **product** come from and which part of the program affects it?

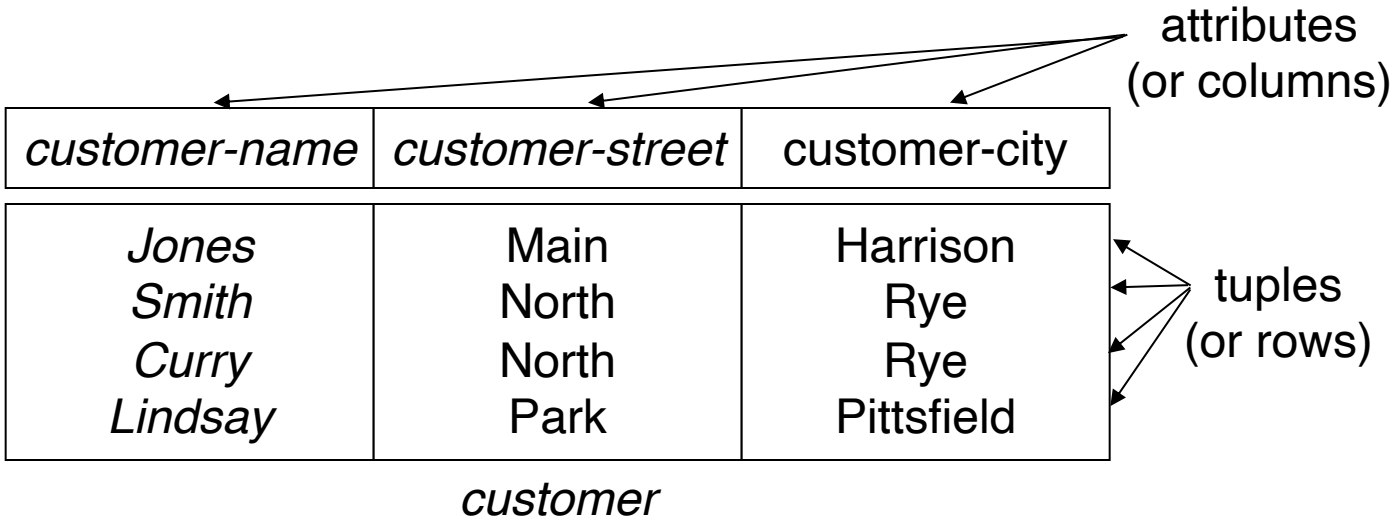
product's value does not depend on these lines



Some Background in Database Systems

Basic Structure of Relational Databases

- Formally, given sets (domains) D_1, D_2, …. D_n a **relation** r is a subset of D_1 x D_2 x … x D_n
  Thus a relation is a **set** of n-tuples (a_1, a_2, …, a_n) where each $a_i \in D_i$
- The current values (*relation instance*) of a relation are specified by a table
- An element t of r is a *tuple*, represented by a *row* in a table

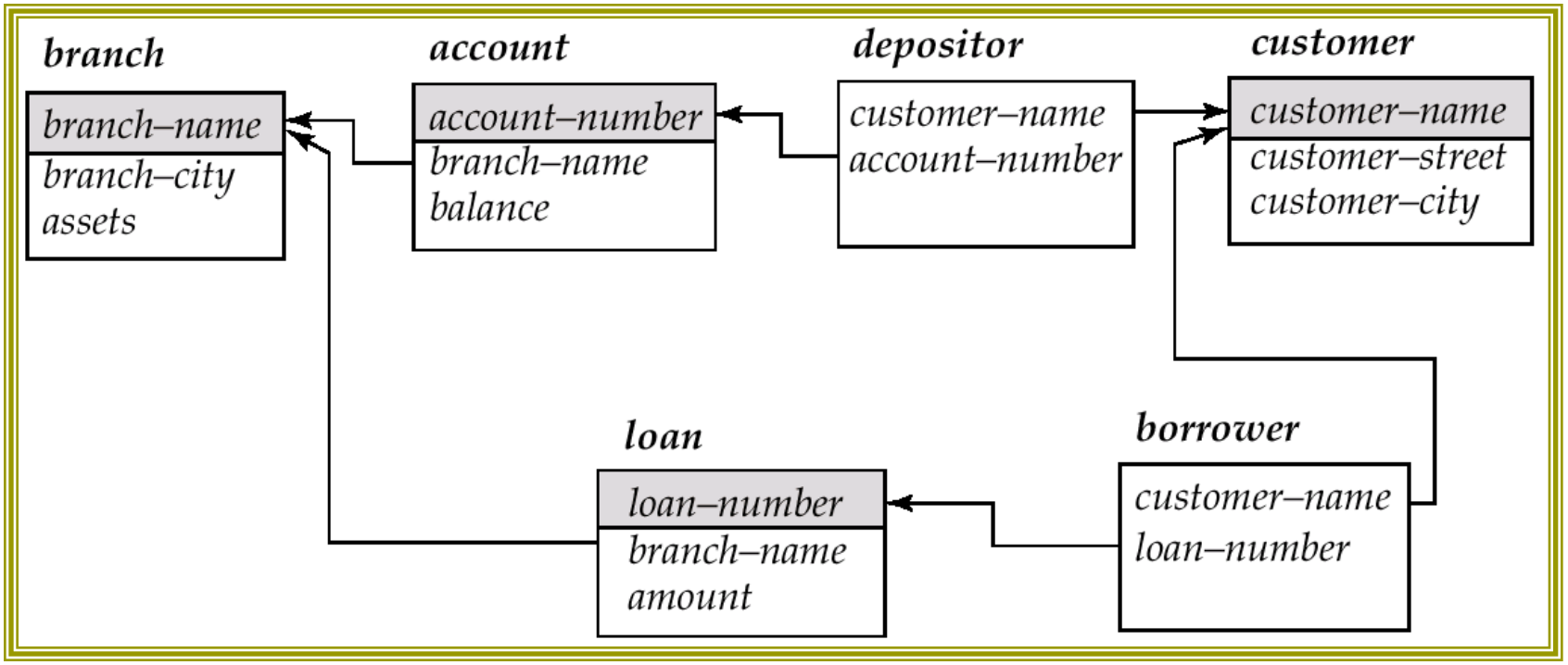


Database

- A database consists of multiple relations which are inter-related
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

  E.g.:
  - *account* : stores information about accounts
  - *depositor* : stores information about which customer owns which account
  - *customer* : stores information about customers

Schema Diagram for a Banking Enterprise



Query Languages: Relational Algebra

- Procedural language
- Six basic operators
  - select, project, union, set difference, Cartesian product, rename
- Additional operations
  - set intersection, natural join, θ-join, division, generalized projection, assignment, aggregation, outer joins, ...
- The operators take one or more relations as inputs and give a new relation as a result.
- In Relational Algebra all inputs and outputs are relations. They are sets of tuples.
  - In SQL the output of an operator is a multiset of tuples

Select Operation – Example

- Relation *r*

A	*B*	*C*	*D*
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

A	*B*	*C*	*D*
α	α	1	7
β	β	23	10

Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Cartesian-Product Operation-Example

Relations *r, s*:

A	*B*
α	1
β	2

r

C	*D*	*E*
α	10	a
β	10	a
β	20	b
γ	10	b

s

r x *s*:

A	*B*	*C*	*D*	*E*
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Natural Join Operation – Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural Join Operation – Example

- Relation *loan*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

- *Relation loan* $\bowtie$ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Aggregate Functions and Operations

- **Aggregate operation** in relational algebra
  - $G1, G2, \ldots, Gn\ \mathcal{g}\ F1(A1), F2(A2), \ldots, Fn(An)\ (E)$
- E is any relational-algebra expression
- $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
- Each F_i is an aggregate function
  - e.g., sum, min, max, count, average, …
- Each A_i is an attribute name

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$branch\text{-}name\ \mathcal{G}\ sum(balance)\ (account)$$

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

SQL: Basic Structure

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  - A_is represent attributes
  - r_is represent relations
  - P is a predicate.
- This query is equivalent to the relational algebra expression.

$$\Pi_{A1, A2, \ldots, An}(\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

- The result of an SQL query is a multiset of tuples.

Aggregate Functions – Group By

- Find the number of depositors for each branch.

select *branch-name,* **count (distinct** *customer-name)*
from *depositor, account*
where *depositor.account-number = account.account-number*
group by *branch-name*

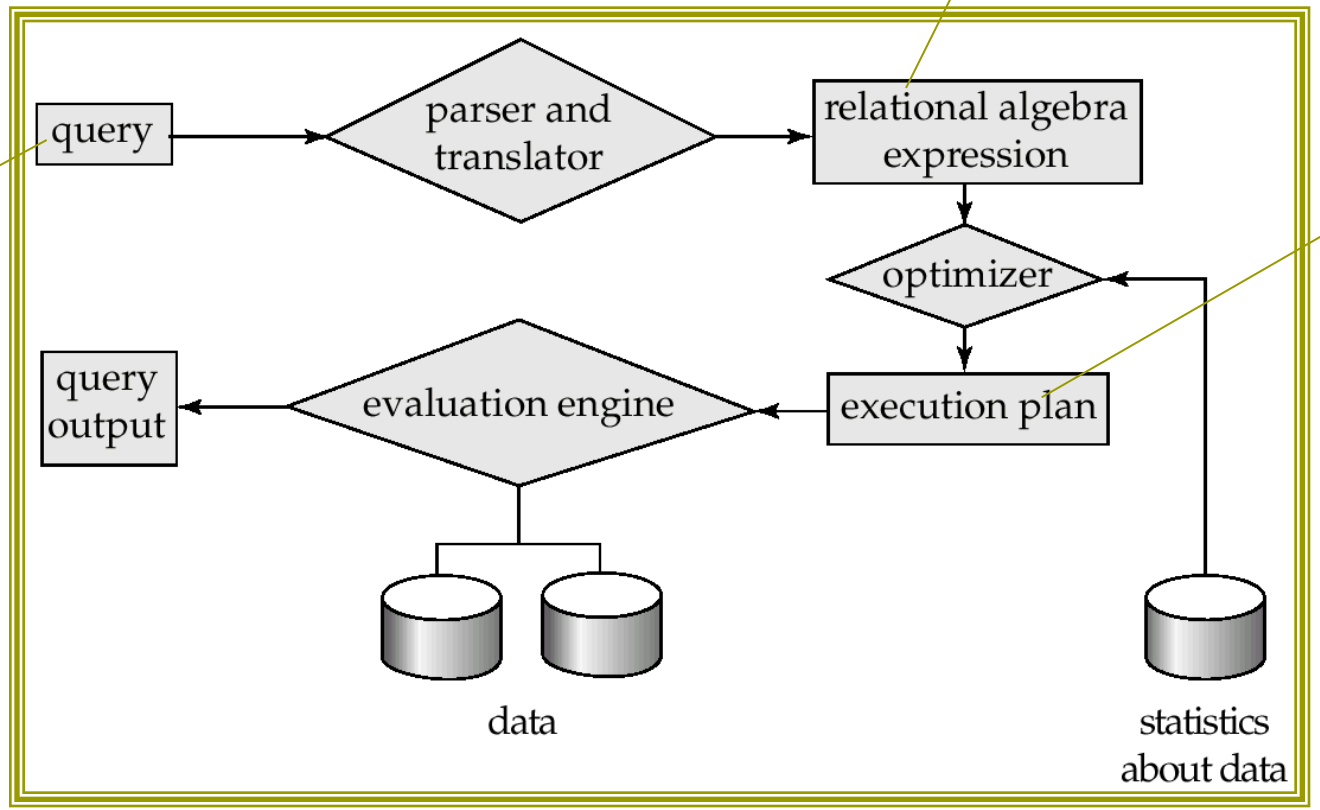
Note: In the **select** clause outside of aggregate functions we must have:

- attributes that appear in the **group by** list and/or
- aggregate functions on attributes of each group

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation

relational algebra is procedural: it describes how the query will be processed



e.g., in SQL

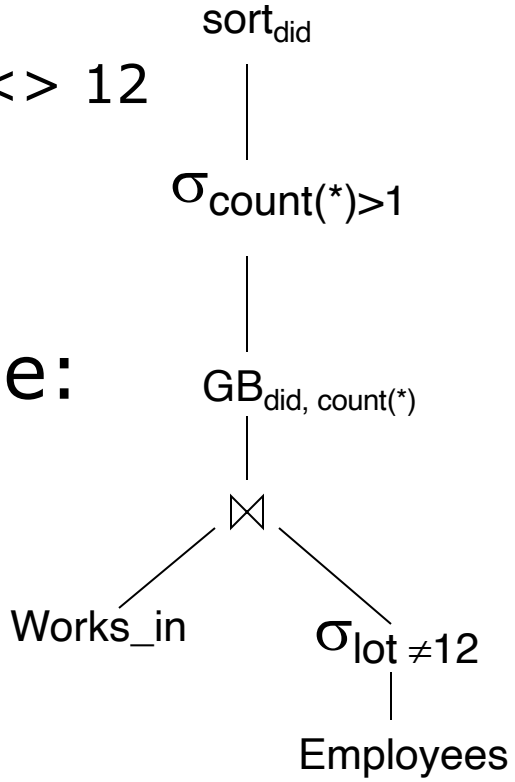
an expression for evaluating the query

Query Parsing and Translation (Example)

- Query in SQL:

```
SELECT W.did, COUNT(*) AS Numemp
FROM Works_in W, Employees E
WHERE W.ssn = E.ssn AND E.lot <> 12
GROUP BY W.did
HAVING COUNT(*)>1
ORDER BY W.did
```

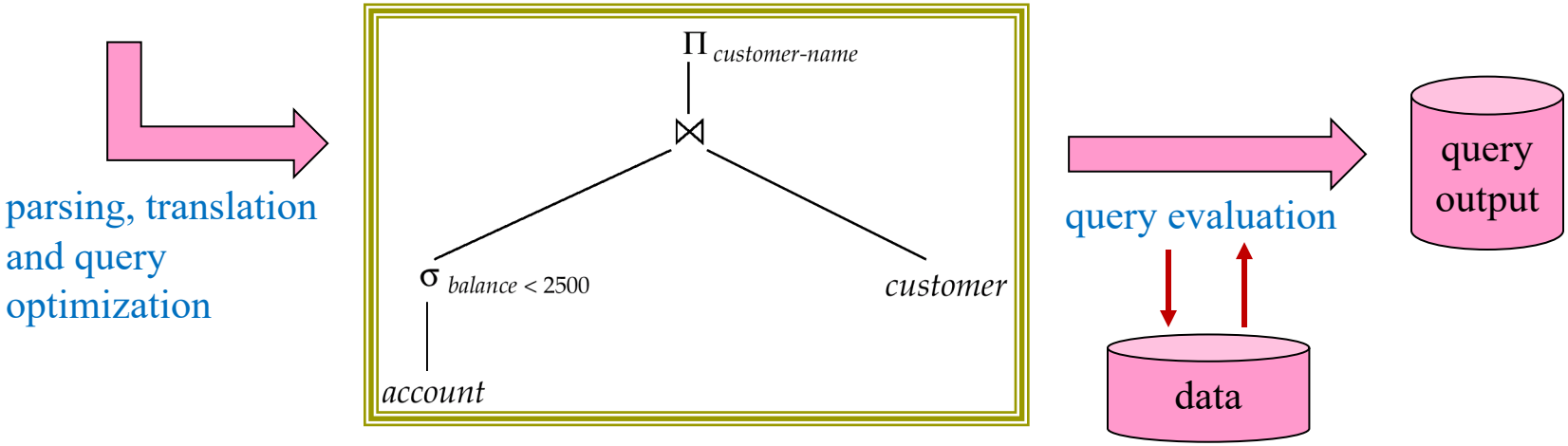
- Extended RA expression tree:



Query Evaluation

- Each query is transformed to an evaluation plan
  - Tree of operations (expression tree)
  - Algorithm used for each operation

```
SELECT C.customer-name
FROM customer C, account A
WHERE C.accountno = A.accountno
  AND A.balance<2500
```



$$\Pi_{\text{customer-name}}(\sigma_{\text{balance} < 2\,500}(\textit{account}) \bowtie \textit{customer})$$

Database Provenance

Reading: "Provenance in Databases: Why, How, and Where," Foundations and Trends in Databases Volume 1, Issue 4, April 2009 pp 379–474, https://doi.org/10.1561/1900000006

Concepts of Database Provenance

- Where: *Where does an attribute value come from?*
- Why: *Why is a tuple produced?*
- How: *How was a tuple produced?*
- Why-not: *Why tuple X doesn't show up in the result set?*

Example of Database Provenance

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"
```

Query Result

name	address
Bob	99 High St

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

Why-Provenance

Query Result

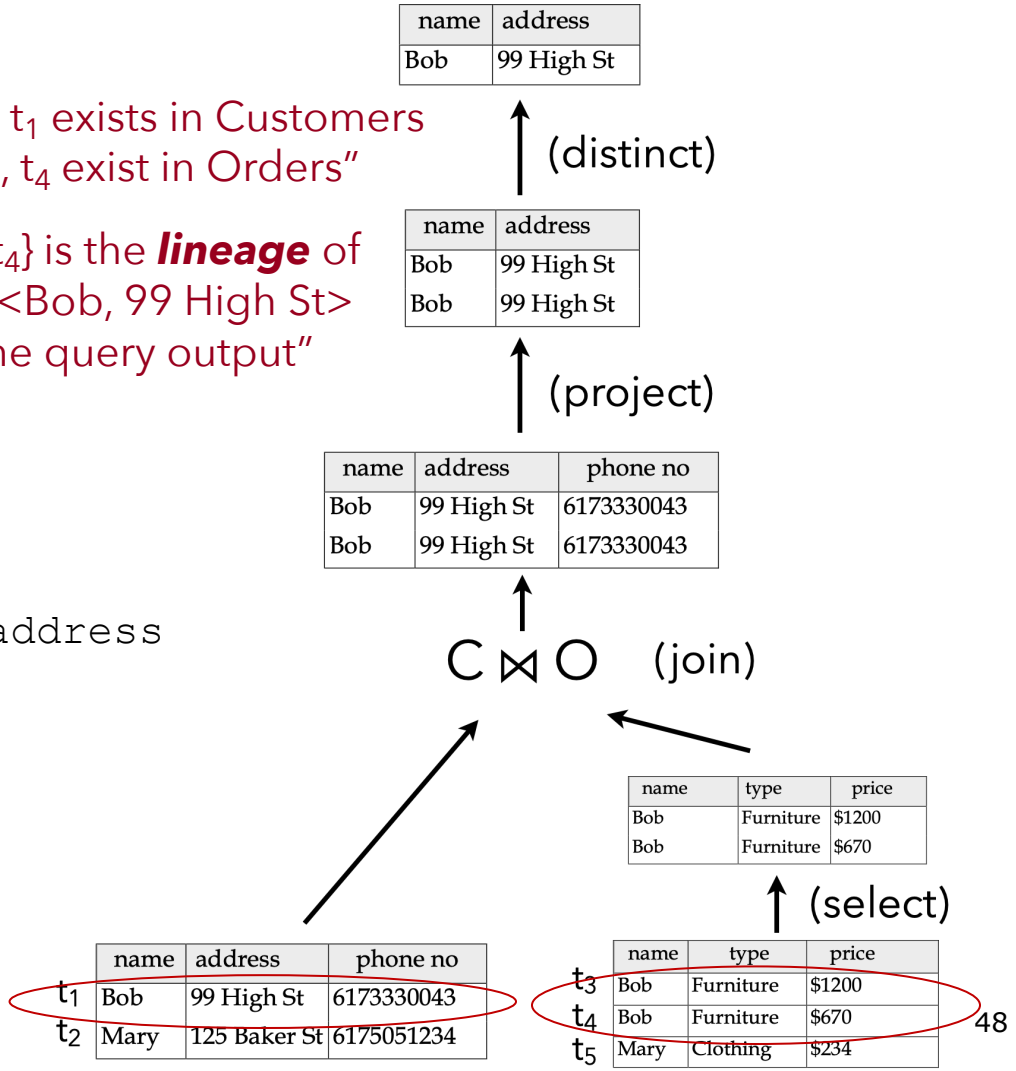
name	address
Bob	99 High St

"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"

Why is this tuple in the result?

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

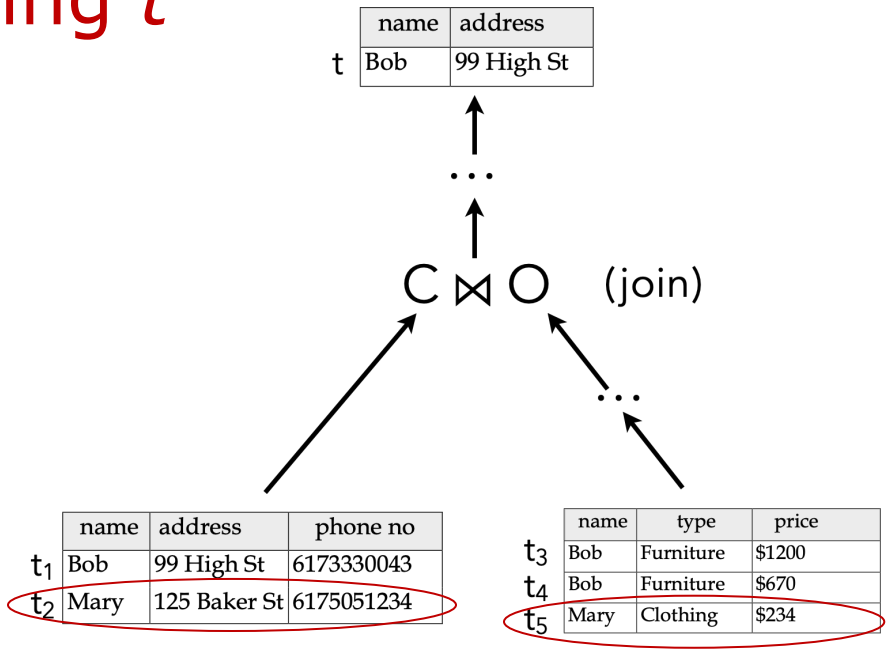


Lineage in Databases

- Given a database *D* and a SQL query *q*, the lineage of a tuple *t* in the query result *q(D)* is the set of input tuples that contributed to producing *t*

Lineage(t) = {t_1, t_3, t_4}

Tuples t_2 and t_5 did not contribute to having *t*=<Bob, 99 High St> in the result, so they are not part of *t*'s lineage



Why-Provenance

Query Result

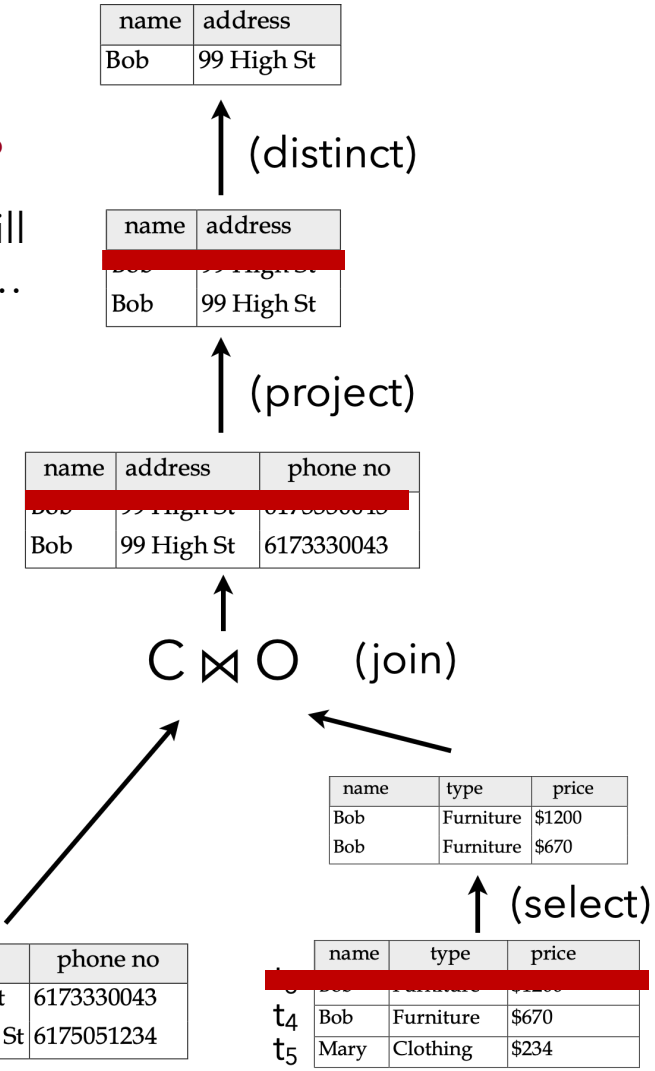
name	address
Bob	99 High St

Why is this tuple in the result?

What if t_3 was not included in Orders?

<Bob, 99 High St> still appears in the result…

<Bob, 99 High St> is produced when t_1 appears in Customers and at least one of t_3 and t_4 appears in Orders




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

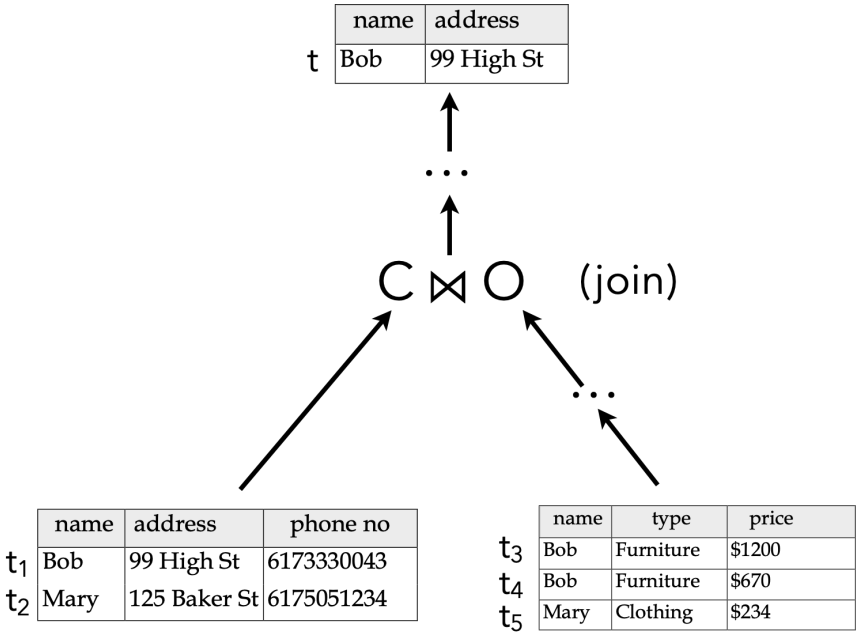
Witness

- 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



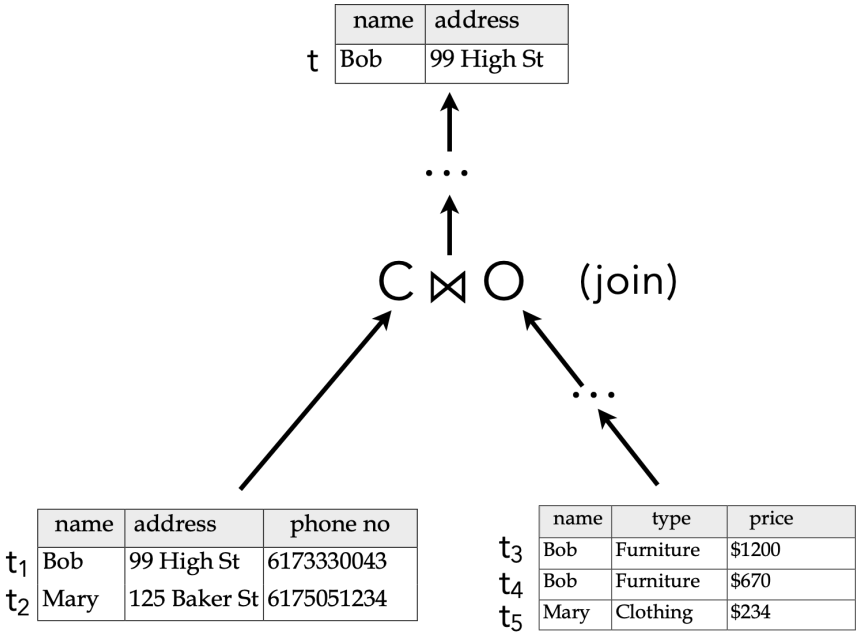
Why-Provenance

- 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



Why-Provenance ≠ Minimal Witness Basis ⊇

- Let R(A,B) be a relation with three tuples:
  - R = {t_1, t_2, t_3}, where t_1={1,2}, t_2={1,3}, t_3={4,2}
- Let q_1, q_2 be two queries:
  - q_1: Ans(x,y) := R(x,y)
  - q_2: Ans(x,y) := R(x,y), R(x,z)

Why({1,2}, q_1(R)) = { {t_1} }

Why({1,2}, q_2(R)) = { {t_1}, {t_1,t_2}}
Is this a minimal witness basis?
No! The minimal witness basis is { {t_1} }

- Assuming that Ans is a *set* of tuples (i.e., there is an implicit DISTINCT), both queries generate the same output when applied to R:
  - q_1(R) = { {1,2}, {1,3}, {4,2}}
  - q_2(R) = { {1,2}, {1,3}, {4,2}}

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"
```

EXPLAIN command in SQL

```
EXPLAIN SELECT *
FROM foo
```

Operator type (select)

Input relation

Estimated costs

Useful but not really what we want…

The result of EXPLAIN does not include the actual tuples or tuple IDs

```
EXPLAIN (FORMAT JSON) SELECT *
FROM foo;
            QUERY PLAN
--------------------------------
 [                              +
   {                            +
     "Plan": {                  +
       "Node Type": "Seq Scan",+
       "Relation Name": "foo", +
       "Alias": "foo",          +
       "Startup Cost": 0.00,    +
       "Total Cost": 155.00,    +
       "Plan Rows": 10000,      +
       "Plan Width": 4          +
     }                          +
   }                            +
 ]
(1 row)
```

Provenance Semirings

- A *commutative semiring* is a quintuple $(K, 0, 1, \oplus, \otimes)$:
  - K is a set of elements containing the distinguished elements 0 and 1
  - 0 and 1 are the identities of $\oplus$ and $\otimes$ respectively
    - $t \oplus 0 = t$ and $t \otimes 1 = t$
  - $\oplus$ and $\otimes$ are two binary operators that are both commutative and associative
    - $t \oplus t' = t' \oplus t$ and $t \otimes t' = t' \otimes t$
    - $t \oplus (t' \oplus t'') = (t \oplus t') \oplus t''$ and $t \otimes (t' \otimes t'') = (t \otimes t') \otimes t''$
  - $\otimes$ is distributive over $\oplus$
    - $t \otimes (t' \oplus t'') = (t \otimes t') \oplus (t \otimes t'')$
  - $t \otimes 0 = 0 \otimes t = 0$

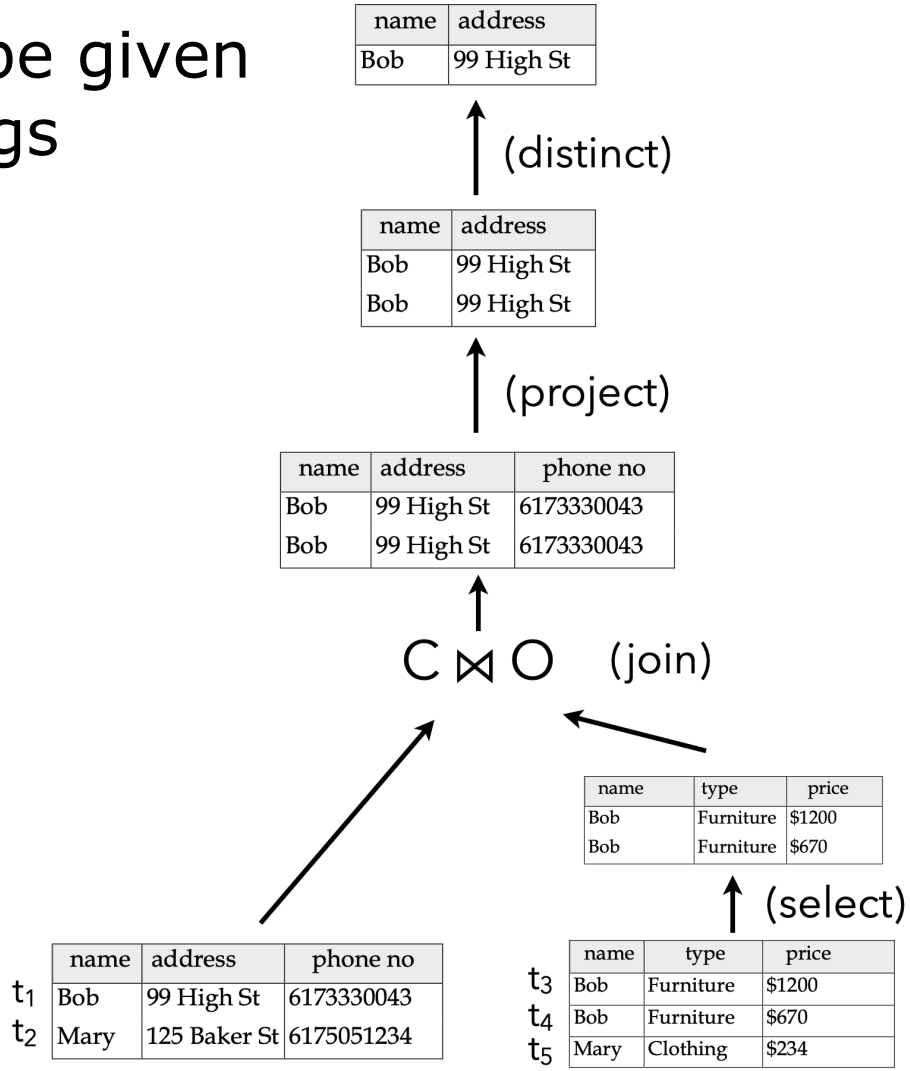
Provenance Semirings

- How provenance can be given in the form of semirings

$$How(t) = (t_1 \otimes t_3) \oplus (t_1 \otimes t_4)$$

"t_1 and t_3" "or" "t_1 and t_4"

lineage and why-provenance can be derived from how-provenance

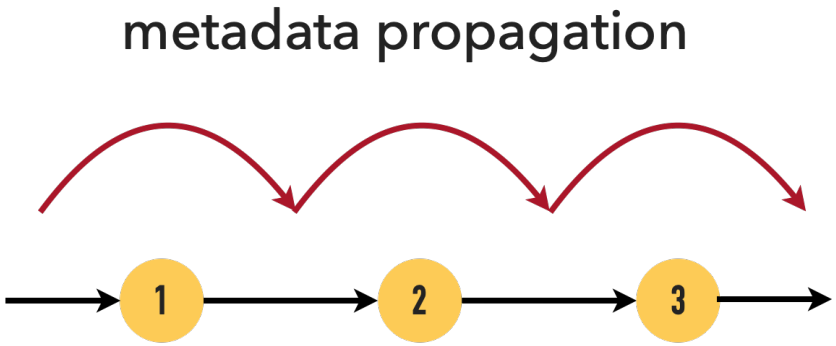


Provenance Techniques

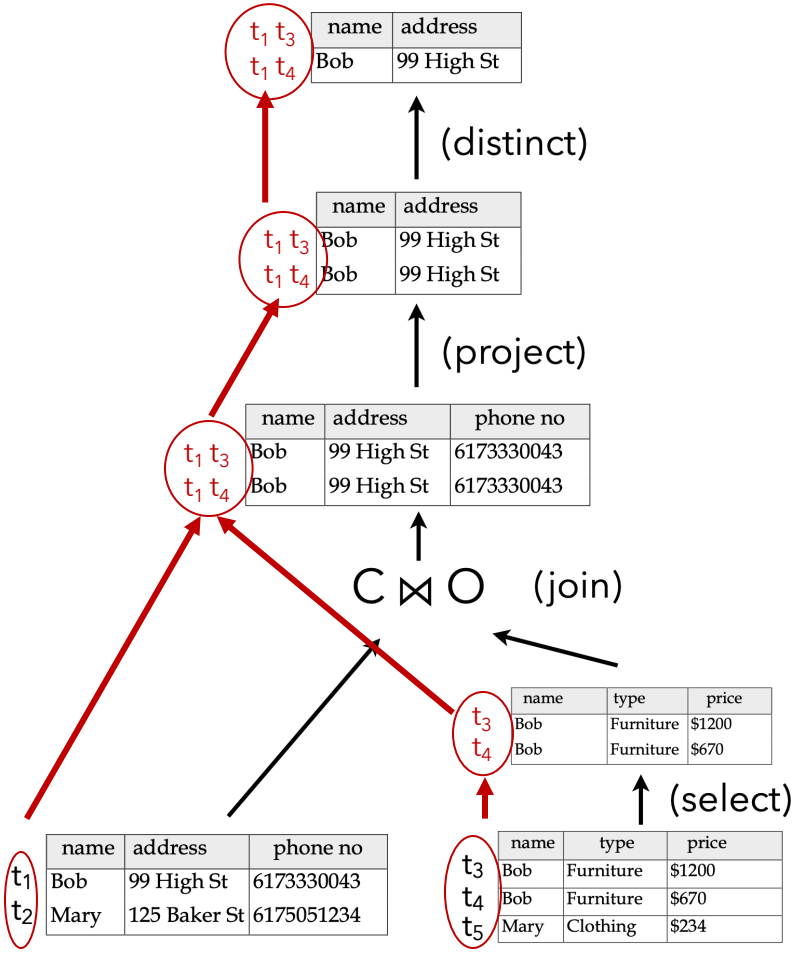
- Annotation propagation
- Operator inversion
- Backward tracing

Metadata Propagation

- Fast
- Explodes in size



Metadata Propagation



Annotation propagation for where-provenance

R_1

	A	B
t_1:	1^{a_1}	2^{a_2}
t_2:	1^{a_3}	3^{a_4}

R_2

	A	C
t_3:	1^{a_5}	2^{a_6}
t_4:	4^{a_7}	5^{a_8}

$\sigma_{B=2}(R_1)$

A	B
1^{a_1}	2^{a_2}

$\pi_A(R_1)$

A
$1^{a_1,a_3}$

$\rho_{B \mapsto C}(R_1)$

A	C
1^{a_1}	2^{a_2}
1^{a_3}	3^{a_4}

$R_1 \bowtie R_2$

A	B	C
$1^{a_1,a_5}$	2^{a_2}	2^{a_6}
$1^{a_3,a_5}$	3^{a_4}	2^{a_6}

annotation-union

Operator Inversion

- Small memory footprint
- Not generally applicable

Operator Inversion

Operation

```
SELECT address, name, phone_no
FROM Customers_A
```

Customers_A

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

Inverse operation

```
SELECT name, address, phone_no
FROM Customers_B
```

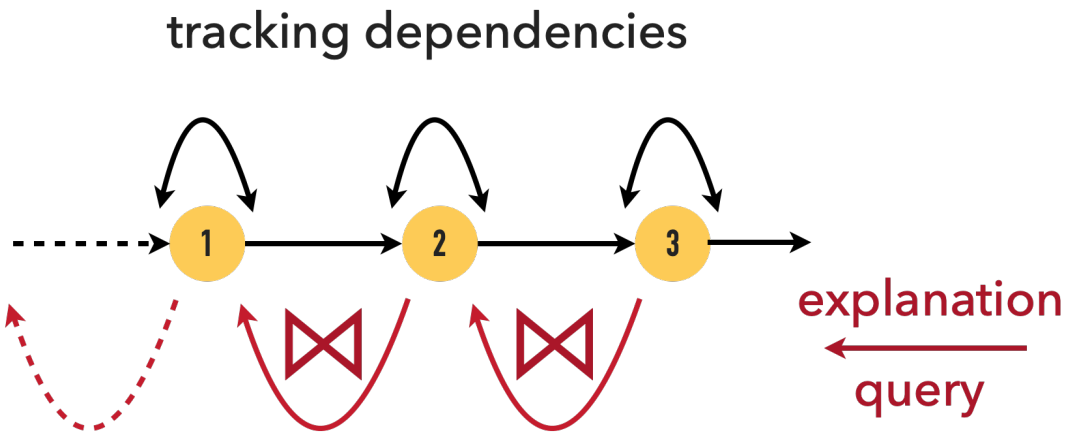
Customers_B

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

Inversion is not always applicable

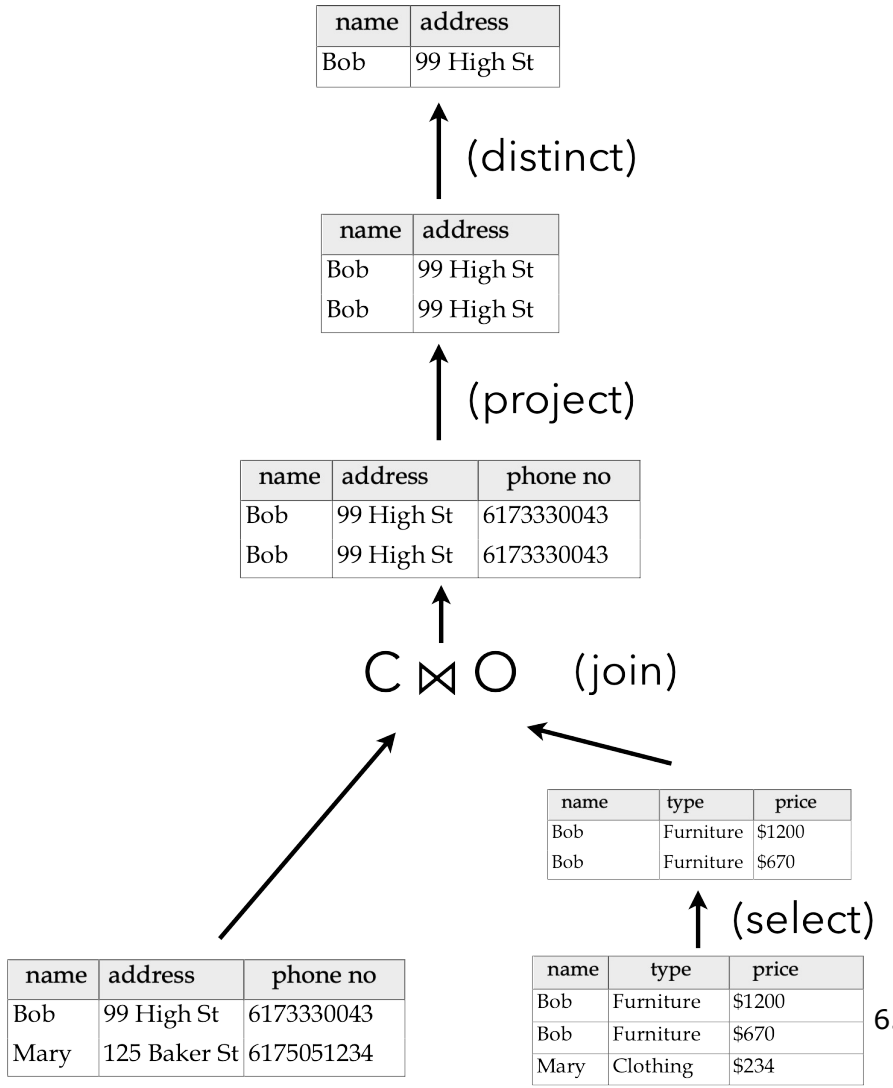
Backward Tracing

- Moderate memory footprint
- Generally applicable
- Fast



Backward Tracing

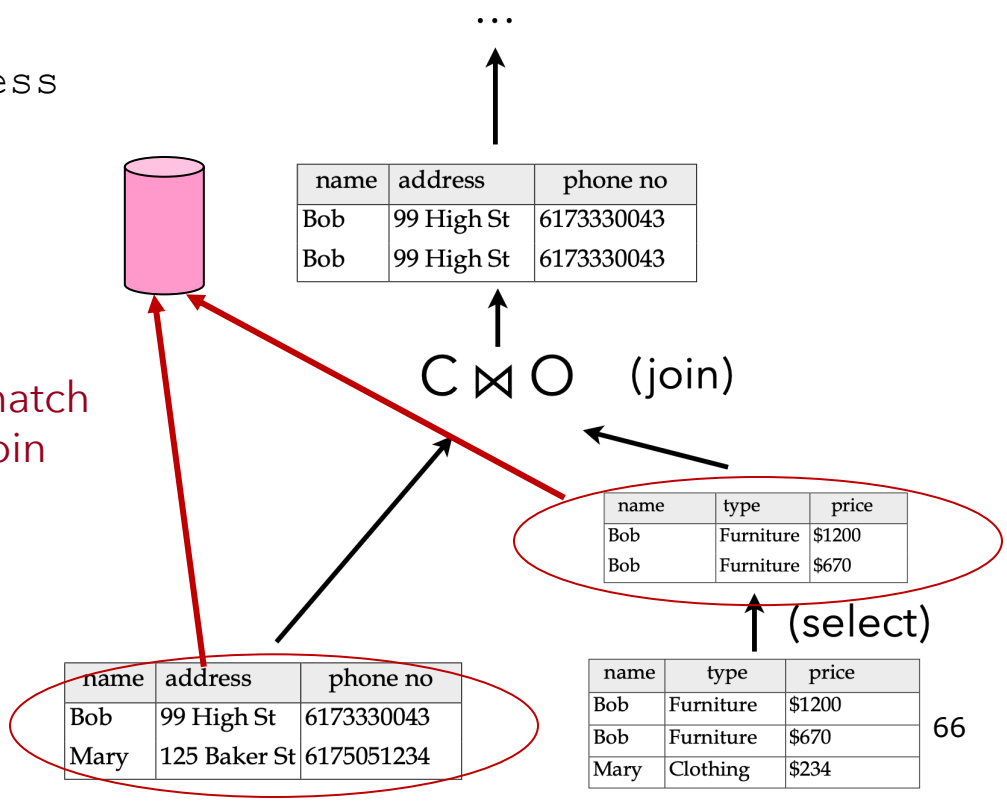
```
SELECT DISTINCT C.name, C.address
FROM Customers C, Orders O
WHERE C.name = O.name
  AND O.type="Furniture"
```



Backward Tracing

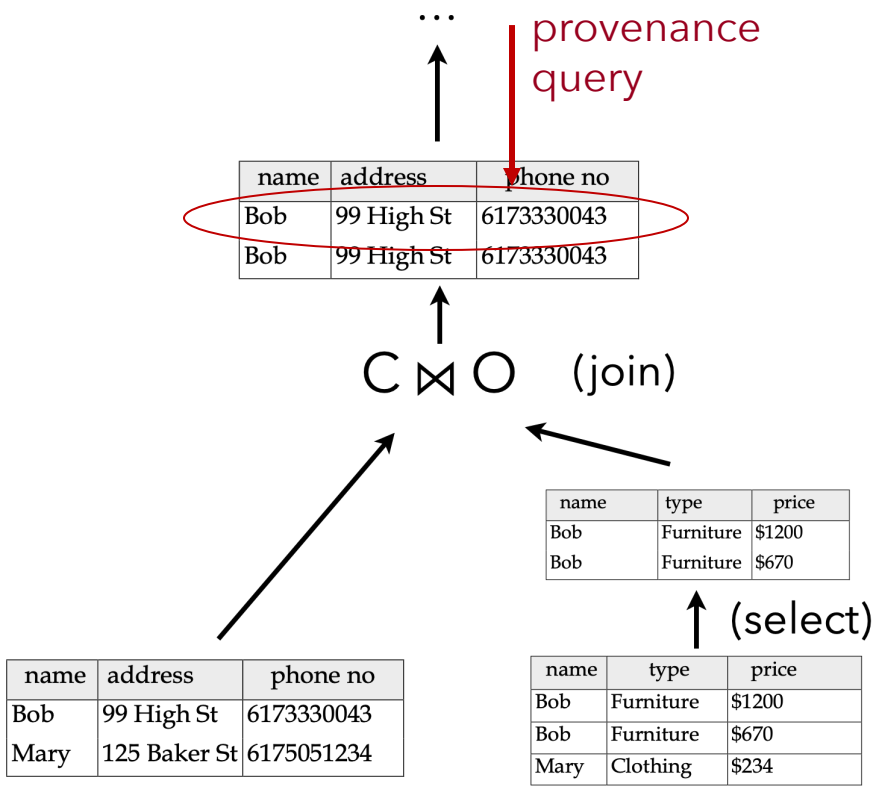
```
SELECT DISTINCT C.name, C.address
FROM Customers C, Orders O
WHERE C.name = O.name
  AND O.type="Furniture"
```

Cache input records that match and index them on the join attribute, i.e., name



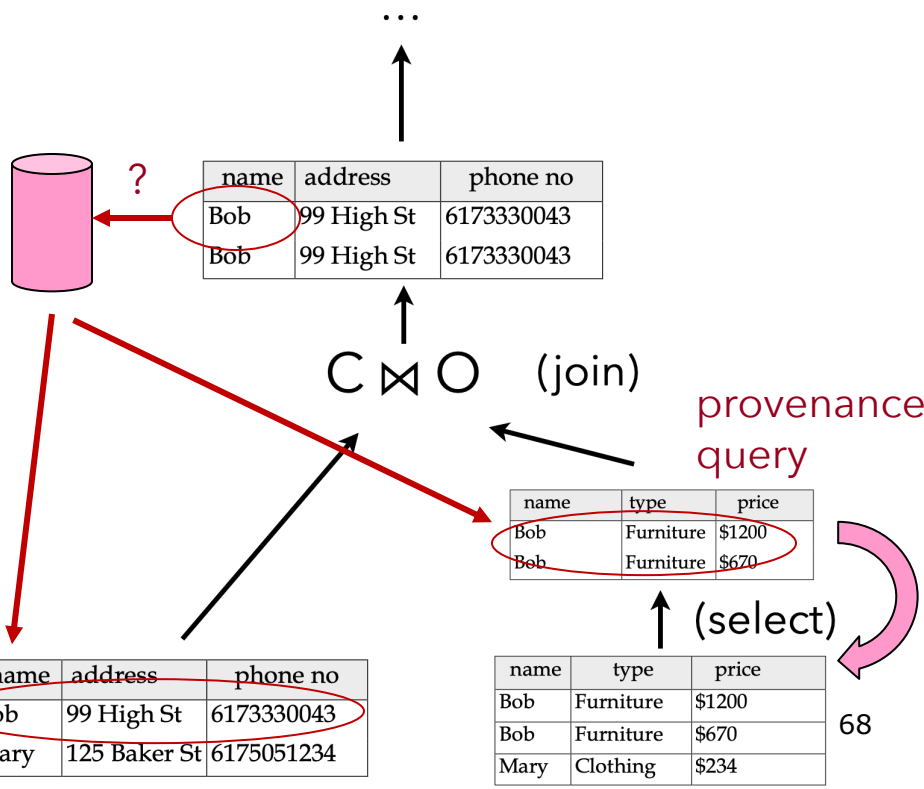
Backward Tracing

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



Backward Tracing

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



Continue backwards until you reach the input tables

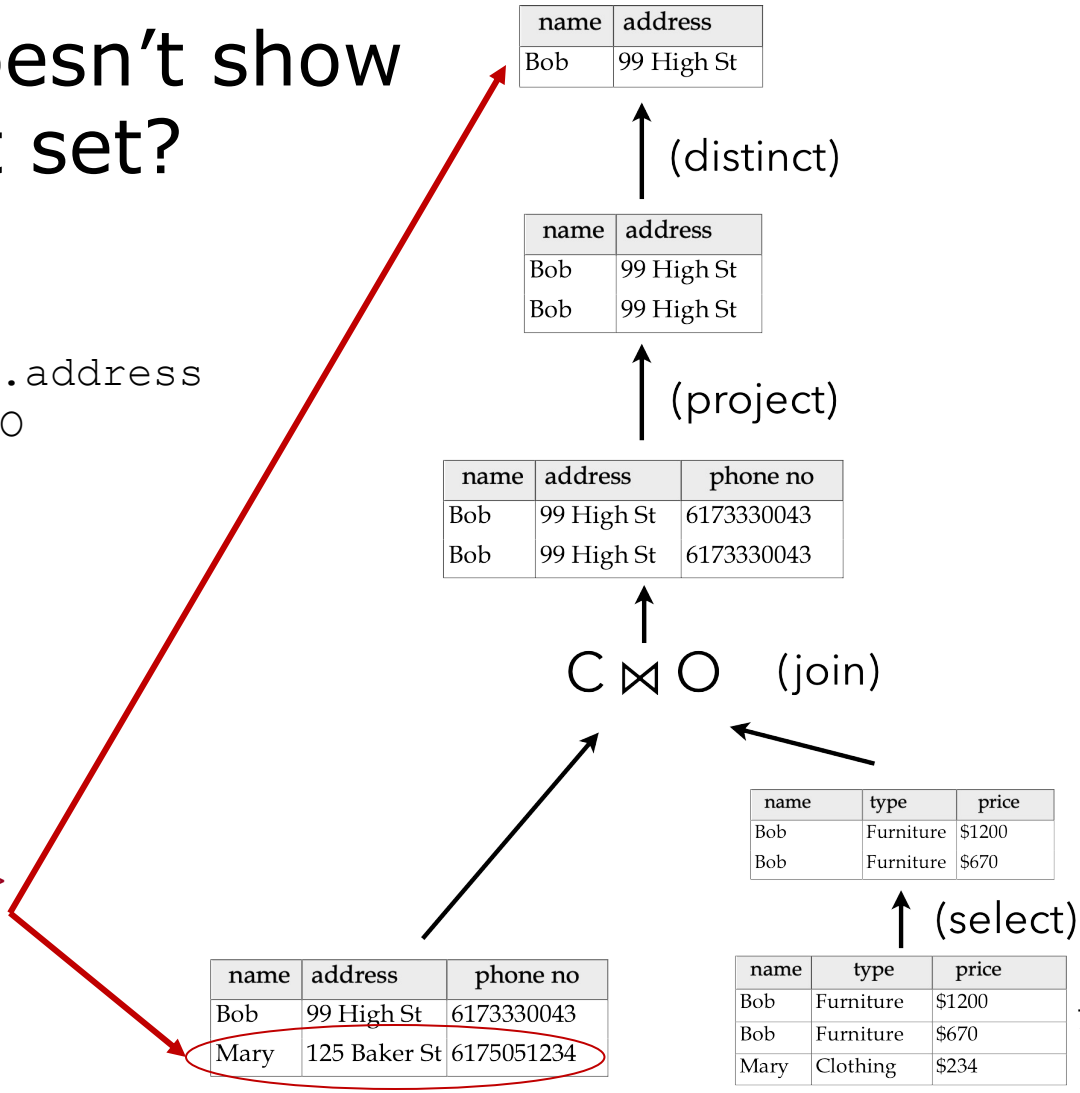
Why-not Provenance

Reading: A. Chapman, H.V. Jagadish. "Why not?" SIGMOD, 2009.

Why-not Provenance

- Why tuple X doesn't show up in the result set?

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



Why is <Mary, 125 Baker St> not included in the result?

Why-not Provenance

- Why tuple X doesn't show up in the result set?

```
SELECT B.Title, B.Price
FROM Books as B, Window_Books as W
WHERE B.Author = W.Author
```

"Show me all window books and their price"

Books

Author	Title	Price	Publisher
Euripides	Medea	$16	Free Press
Homer	Odyssey	$49	Penguin
Hrotsvit	Basilius	$20	Harper
Shakespeare	Coriolanus	$70	Penguin

Window_Books

Author	Title	Discount
Euripides	Medea	10%
Hrotvsit	Basilius	20%
Shakespeare	Coriolanus	5%

Result

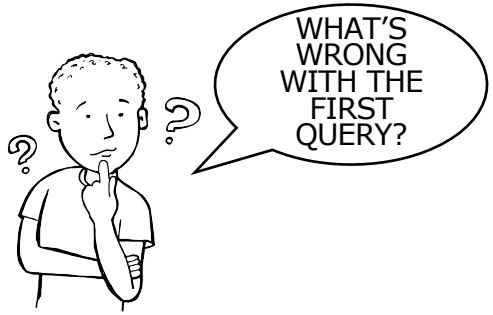
Title	Price
Medea	$16
Coriolanus	$70

Why-not Provenance

- Let's try something different

```
SELECT B.Title, B.Price
FROM Books as B, Window_Books as W
WHERE B.Author = W.Author
WHERE B.Title = W.Title
```

"Show me all window books and their price"



Books

Author	Title	Price	Publisher
Euripides	Medea	$16	Free Press
Homer	Odyssey	$49	Penguin
Hrotsvit	Basilius	$20	Harper
Shakespeare	Coriolanus	$70	Penguin

Window_Books

Author	Title	Discount
Euripides	Medea	10%
Hrotvsit	Basilius	20%
Shakespeare	Coriolanus	5%

Result

Title	Price
Medea	$16
Basilius	$20
Coriolanus	$70

Why-not Provenance

Leverages:

- Lineage/provenance
- SQL query
- Why-not question

to identify "*manipulations*" that contribute to not having an expected tuple in the result

Manipulation

- A basic unit of processing
- Defined at different levels of granularity:
  - A whole (black-box) workflow
  - A SQL query
  - An operator in a SQL query
  - …

Books

Author	Title	Price	Publisher
Euripides	Medea	$16	Free Press
Homer	Odyssey	$49	Penguin
Hrotsvit	Basilius	$20	Harper
Shakespeare	Coriolanus	$70	Penguin

Example:

```
SELECT Title
FROM Books
WHERE Price>$20
```

- Notation: $M(D^{I1}, D^{I2}, D^{I3}, \ldots D^{In}) = D^{O}$

manipulation

input datasets (e.g. tables)

output

Predicate satisfaction

- Given a database *D* and a query *q* that produces a result *q(D)*
  - Why does *q(D)* not satisfy predicate *S?*
  - *S is defined over attributes in D*
  - *S* is a logical formula with atoms combined with AND and OR
  - Example predicates:
    - Price<=$33 OR Publisher='Penguin'
    - Author=Homer AND Publisher='Harper'
    - ~~Author!=Shakespeare~~ AND Price<=$100

Books

Author	Title	Price	Publisher
Euripides	Medea	$16	Free Press
Homer	Odyssey	$49	Penguin
Hrotsvit	Basilius	$20	Harper
Shakespeare	Coriolanus	$70	Penguin

We do not consider negation

Only 'positive' predicates

Predicate satisfaction

- A tuple *t* satisfies predicate *S* *iff S(t)* evaluates to TRUE

Books

Author	Title	Price	Publisher
Euripides	Medea	$16	Free Press
Homer	Odyssey	$49	Penguin
Hrotsvit	Basilius	$20	Harper
Shakespeare	Coriolanus	$70	Penguin

  - Example 1:
    - *S* := Price<=$33 OR Publisher='Penguin'
    - *t*=<Homer, Odyssey, $49, Penguin>
      - *t* satisfies *S*
  - Example 2:

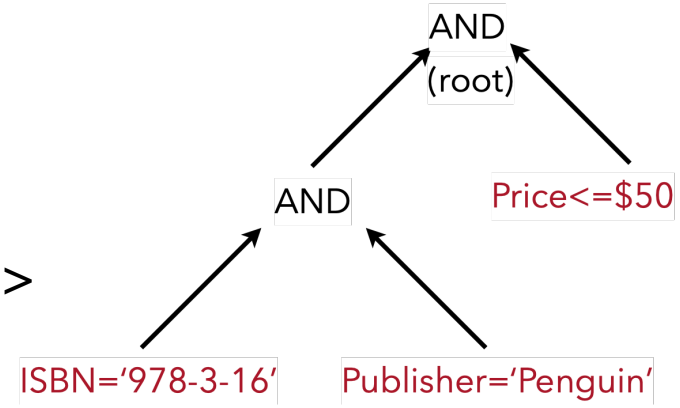
undefined for tuple t

evaluates to TRUE for tuple t

    - *S* := ISBN='978-3-16' AND Publisher='Penguin'
    - *t*=<Homer, Odyssey, $49, Penguin>
      - *t* is *satisfaction-compatible* with *S*
  - If *t* satisfies *S* then it is also satisfaction-compatible with *S*

Satisfaction compatibility

- A tuple *t* is *satisfaction-compatible* with a predicate *S* if:
  - There is no internal AND node with both inputs UNDEFINED
  - Setting UNDEFINED inputs to AND nodes to TRUE makes the root of the predicate tree TRUE for *t*
  - Example:
    - *S* := ISBN='978-3-16' AND Price<=$50 AND Publisher='Penguin'
    - *t*=<Homer, Odyssey, $49, Penguin>
    - If ISBN='978-3-16' is TRUE, then S(t) TRUE



Unpicked data item

- Given a query result *q(D)* and a predicate *S*, *a* tuple *t* is *unpicked* if:
  - There exists an attribute *a* that is both associated with *t* and appears in *S*
  - *t* is satisfaction-compatible with *S*
  - *t* is not in the lineage of any tuple in *q(D)*

```
SELECT Title
FROM Books
WHERE Price>$20
```

Result

	Title
t	Odyssey
t′	Coriolanus

Books

	Author	Title	Price	Publisher
t_1	Euripides	Medea	$16	Free Press
t_2	Homer	Odyssey	$49	Penguin
t_3	Hrotsvit	Basilius	$20	Harper
t_4	Shakespeare	Coriolanus	$70	Penguin

Why is 'Medea' not included in the result?

Question expressed as S := Title='Medea'

Tuple t_1 is an **unpicked item**

Lineage(t) = $\{t_2\}$
Lineage(t′) = $\{t_4\}$

Picky manipulation

- A manipulation *m* is *picky* w.r.t. an unpicked tuple *t* if:
  - *t* or a *successor* of *t* is in the input of *m*
  - there is no successor of *t* in the output of *m*
- A successor of a tuple *t* is a tuple *t'* such that *t* belongs to the lineage of *t'*
- Intuition: A picky manipulation is a manipulation that filters out *t*

Books

	Author	Title	Price	Publisher
t_1	Euripides	Medea	$16	Free Press
t_2	Homer	Odyssey	$49	Penguin
t_3	Hrotsvit	Basilius	$20	Harper
t_4	Shakespeare	Coriolanus	$70	Penguin

```
SELECT Title
FROM Books
WHERE Price>$20
```

The selection `Price>$20` is a picky manipulation w.r.t. t_1

Example

```
SELECT b2.Title, b2.Price
FROM Books as b1, Books as b2
WHERE b1.Title = 'Odyssey'
 AND b2.Price < b1.Price
```

self-join

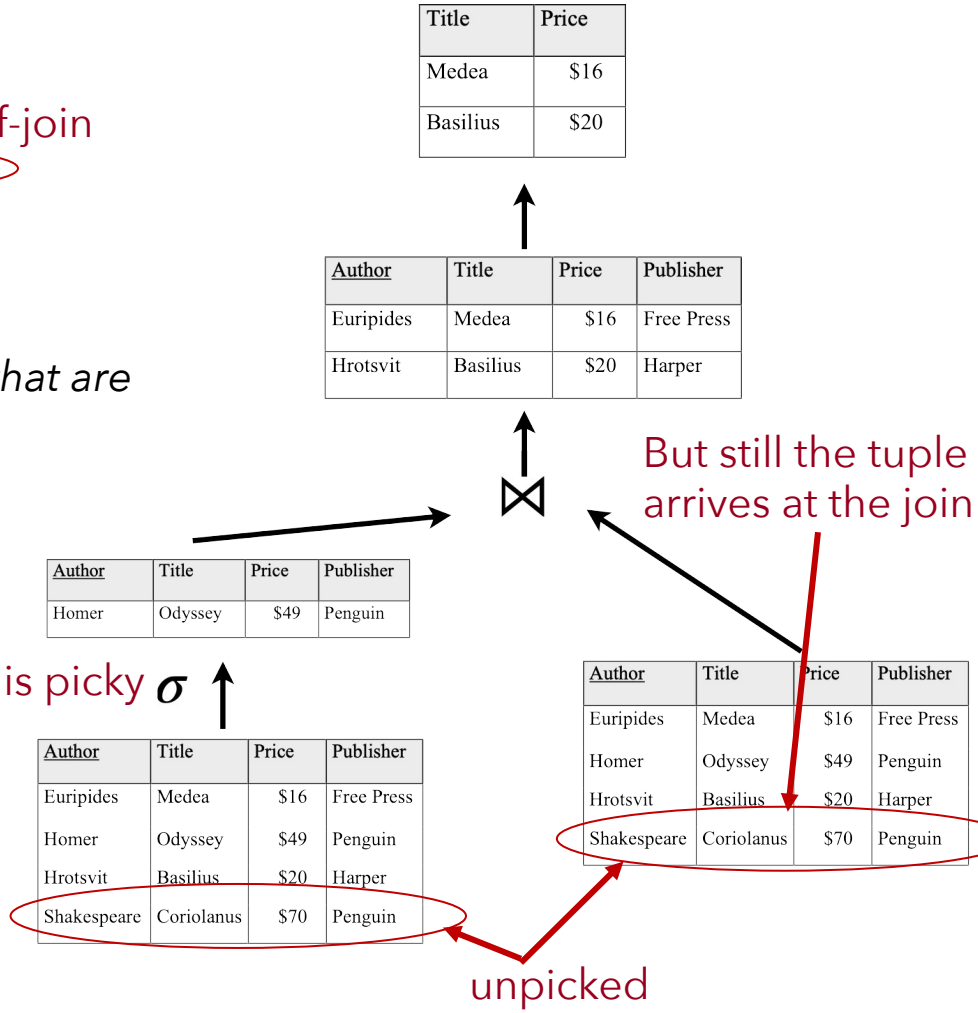
"Show me all titles and prices of books that are priced less than the Odyssey"

Books

Author	Title	Price	Publisher
Euripides	Medea	$16	Free Press
Homer	Odyssey	$49	Penguin
Hrotsvit	Basilius	$20	Harper
Shakespeare	Coriolanus	$70	Penguin

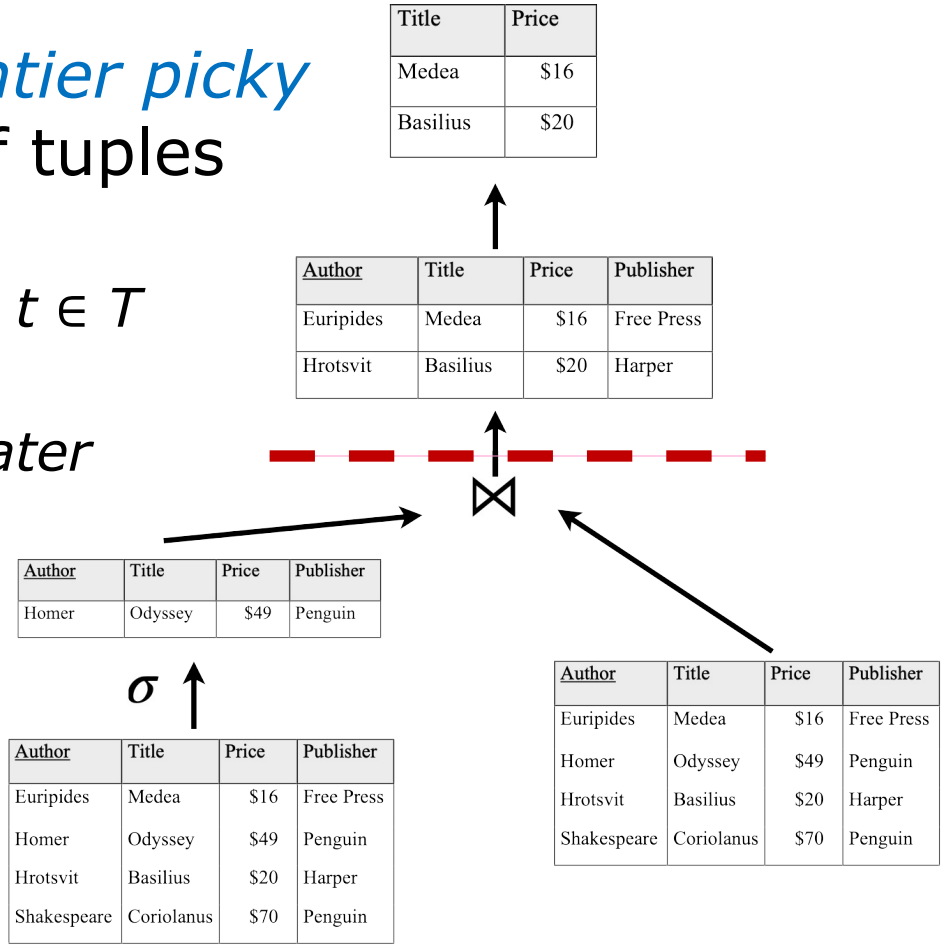
Why there are no books authored by Shakespeare in the result?

S := Author='Shakespeare'



Frontier picky manipulation

- A manipulation *m* is *frontier picky* w.r.t. an unpicked set of tuples *T* if:
  - *m* is picky for at least one $t \in T$
  - $\nexists\ t \in T$ such that *t* or a successor of *t appears later in the computation*



The join is a frontier picky manipulation w.r.t. T = $\{t_4\}$

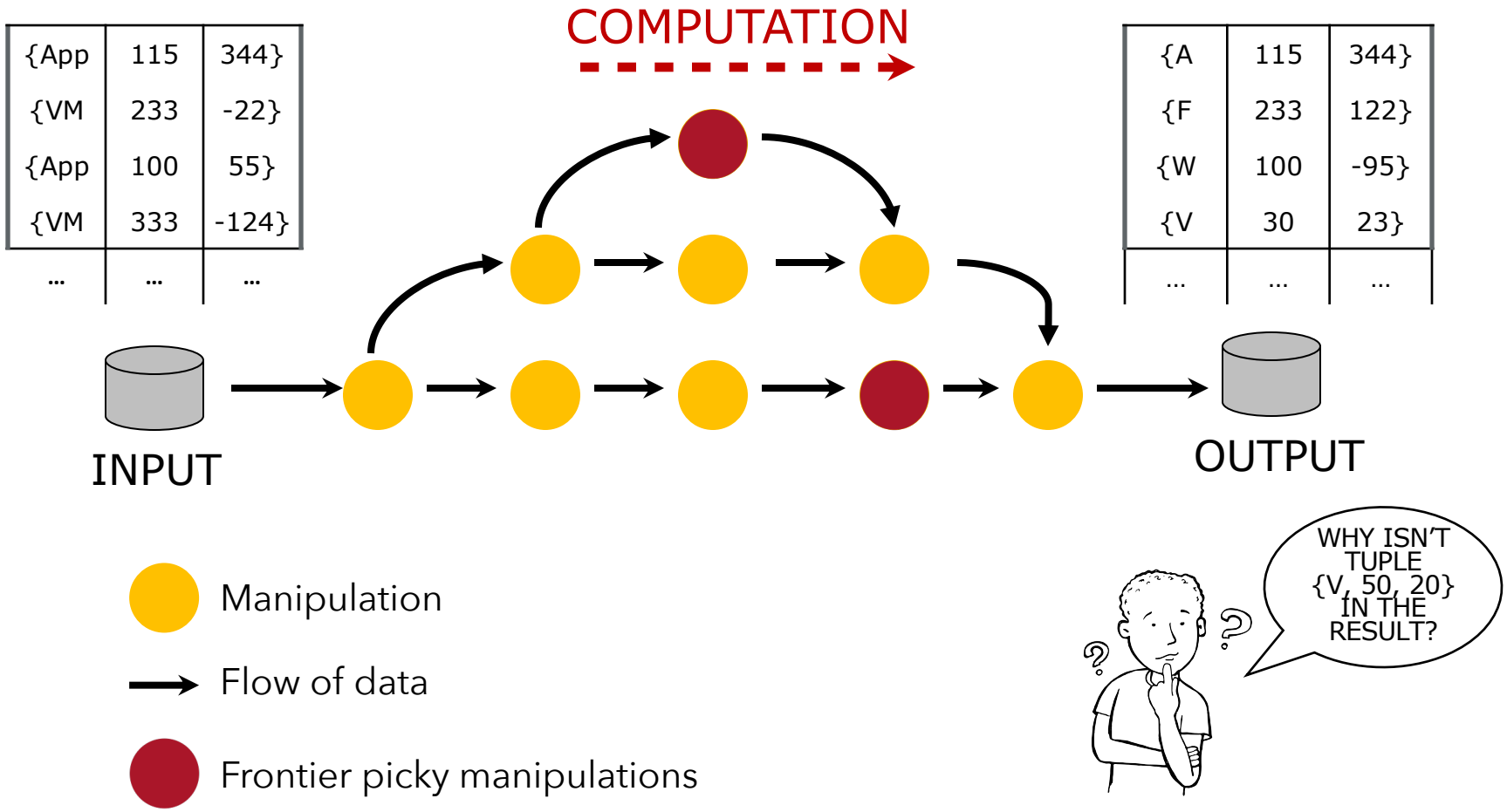
Why-not answer

- Given:
  - A database *D*
  - A workflow *W* that consists of *M* manipulations and produces a result *R*
  - A predicate *S* (why-not question)

- The answer to the why-not question is the set of frontier picky manipulations w.r.t. the tuples in *D* identified as *unpicked* according to *W* and *S*

A workflow can be anything, e.g. a SQL query, a series of SQL queries, a scientific workflow, etc.

Why-not answer



Computing why-not answers

- Alternatives:
  - Bottom-up (from INPUT to OUTPUT)
    - Re-executes query
  - Top-down (from OUTPUT to INPUT)
    - Uses intermediate data

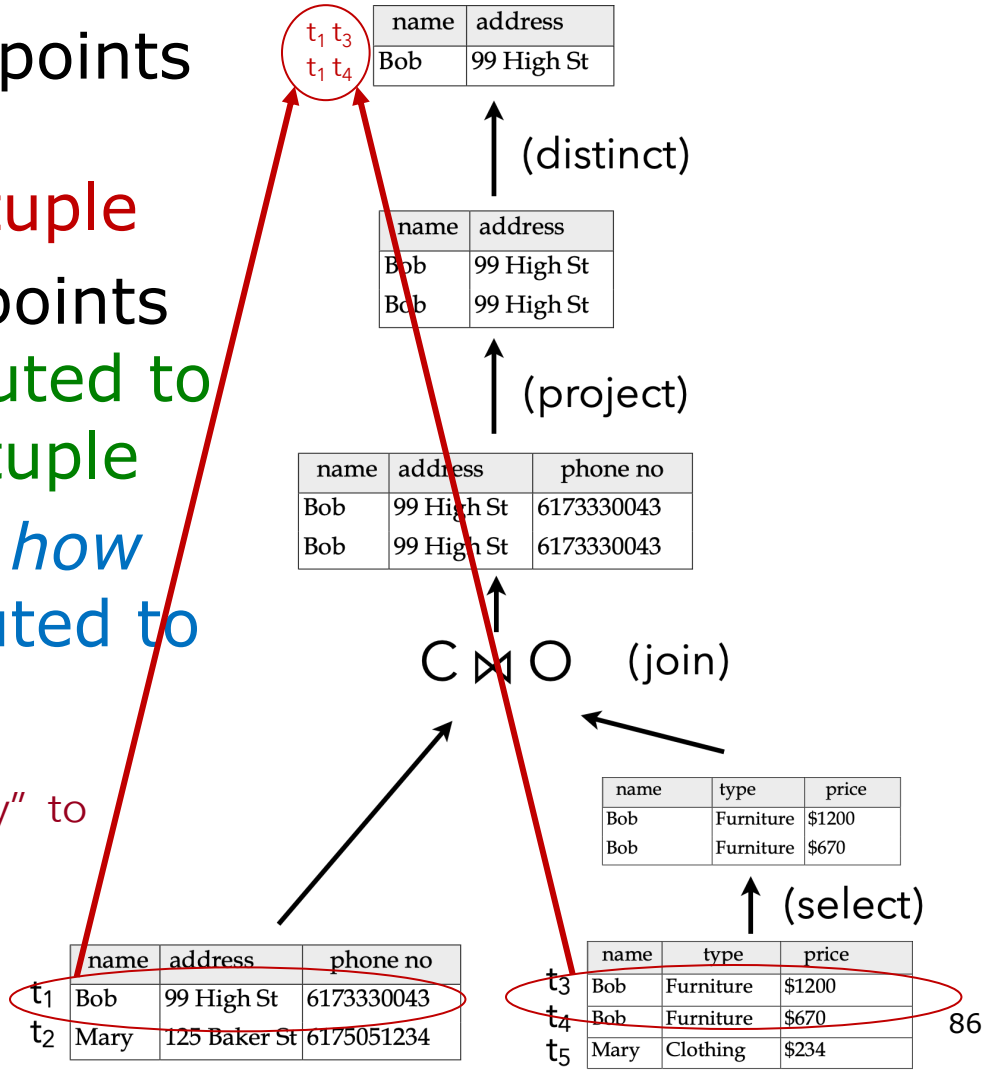
- High-level idea:
  - Start a BFS (Breadth-First Search) from the input or the output
  - At each step, identify picky manipulations
  - Maintain frontier picky manipulation along the way

Data Causality

What have we seen so far

- Why-not provenance points to operators that suppressed an input tuple
- Lineage/Provenance points to tuples that contributed to producing an output tuple
- No information about *how much* a tuple contributed to producing an output

Did tuples t_1, t_3, and t_4 contribute "equally" to having <Bob, 99 High St> in the output?



Data Causality

- Allows us to quantify the contribution of individual tuples
- Allows us to rank tuples according to their contribution in producing an output
- New concepts:
  - Counterfactual tuples
  - Actual causes of an output tuple
  - Tuple responsibility

Motivation:
Lineage can grow very large

```
SELECT R.MID
FROM (
    SELECT R.MID, AVG(R.Rating) as score
    FROM Friends as F, Ratings as R
    WHERE F.UID2 = R.UID
       AND F.UID1 = 'Alice'
    GROUPBY R.MID
    ORDERBY score DESC
    LIMIT 1 )
```

Result

MID
Star Wars

"Recommend Alice the movie with the highest average rating as given by their friends"

Friends

UID1	UID2
Alice	Dustin
Alice	Mary
Alice	Bob
…	…

>100 friends

Ratings

UID	MID	Rating
Dustin	Star Wars	5
Bob	Star Wars	4
Mary	Star Wars	2
…	…	…

>100 ratings

Lineage includes more than 200 input tuples

Motivation: Lineage can grow very large

- Lineage/Provenance based explanations may overwhelm the user
- Explanations must be as concise as possible
- Data causality helps us identify those input tuples in the lineage that serve as 'good' explanations

Data Causality

- Based on definition by Halpern and Pearl[1]
- Database $D = D^x \cup D^n$
  - D^x: Exogenous tuples
  - D^n: Endogenous tuples
- Endogenous tuples are candidates for being causes of an output
- Exogenous tuples are the rest of the tuples in D

[1] J. Y. Halpern, J. Pearl. *Causes and Explanations: A Structural Model Approach – Part I: Causes,* UAI, 2001

Counterfactual cause

- Given a database $D = D^x \cup D^n$, a tuple $t \in D^n$, and a query q that produces an output R, we say that t is a counterfactual cause for $r \in R$ if:
  - $q(D) \models r$ and
  - $q(D - \{t\}) \not\models r$

"entails"

"does not entail"

- Removing t from the database also removes r from the result of the query q

Actual cause

- Given a database $D = D^x \cup D^n$, a tuple $t \in D^n$, and a query q that produces an output R, we say that t is an actual cause (or simply cause) for $r \in R$ if there exists a set $\Gamma \subseteq D^n$ such that:
  - $q(D - \Gamma) \models r$ and
  - $q(D - \Gamma \cup \{t\}) \not\models r$

- The set of tuples Γ is called a contingency for tuple t

If t is a counterfactual cause then it is also an actual cause

Monotone vs non-monotone queries

- A query *q* is mononote if ∀ D'⊆D we have q(D')⊆q(D)
- Evaluating the query *q* on any subset of its input will produce a subset of its output
- Examples of monotone queries:
  - R(X) := Friends(X, Y)
  - S(Y) := Friends('Alice', Y)
  - W(X,Z) := Friends(X, Y), Friends(Y, Z)

```
SELECT UID1
FROM Friends
```

```
SELECT UID1
FROM Friends
WHERE UID1='Alice'
```

```
SELECT F1.UID1, F2.UID2
FROM Friends as F1, Friends as F2
WHERE F1.UID2=F2.UID1
```

A non-monotone query

```
SELECT MID, AVG(Rating) as AVG_Rating
FROM Friends as F, Ratings as R
WHERE F.UID2 = R.UID  AND F.UID1 = 'Alice'
GROUPBY MID
```

"Return the average rating per movie as given by Alice's friends"

Friends' ⊆ Friends

Friends

UID1	UID2
Alice	Dustin
Alice	Mary
~~Alice~~	~~Bob~~
…	…

Result

MID	AVG_Rating
Star Wars	~~4.5~~ 5
Blade Runner	2
…	…

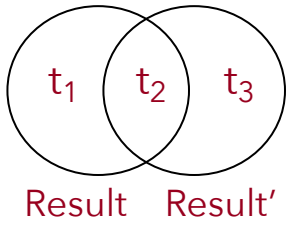
Ratings

UID	MID	Rating
Dustin	Star Wars	5
Bob	Star Wars	4
Mary	Blade Runner	2
…	…	…

Result = {t_1=(Star Wars, 4.5), t_2=(Blade Runner, 2)}

Result' = {t_3=(Star Wars, 5), t_2=(Blade Runner, 2)}

Result' ⊈ Result



Actual cause (full definition)

- Given a database $D = D^x \cup D^n$, a tuple $t \in D^n$, and a query q that produces an output R, we say that t is an actual cause (or simply cause) for $r \in R$ if there exists a set $\Gamma \subseteq D^n$ such that:
  - $q(D - \Gamma) \models r$ and
  - $q(D - \Gamma \cup \{t\}) \not\models r$
  - $q(D - \Gamma') \models r$ for all $\Gamma' \subseteq \Gamma$ (in case q is non-monotone)

$D\text{-}\Gamma \subseteq D\text{-}\Gamma'$ which means that, for a monotone query q, we have $q(D\text{-}\Gamma) \subseteq q(D\text{-}\Gamma')$ and the third condition is always satisfied

Responsibility

- Given a database $D = D = D^x \cup D^n$, a tuple $t \in D^n$, and a query q that produces an output R, and a tuple t that is a cause for $r \in R$, we define:

$$\rho = \frac{1}{1+\min|\Gamma|}$$

- ρ is called the responsibility of tuple t for r
- $\min|\Gamma|$ is the minimum size amongst all possible contingencies for t
- If t is a counterfactual cause, then $\Gamma=\emptyset$ and $\rho=1$

H. Chockler, J. Y. Halpern. *Responsibility and blame: A structural-model approach,* JAIR, 22:93-115, 2004

Back to our example

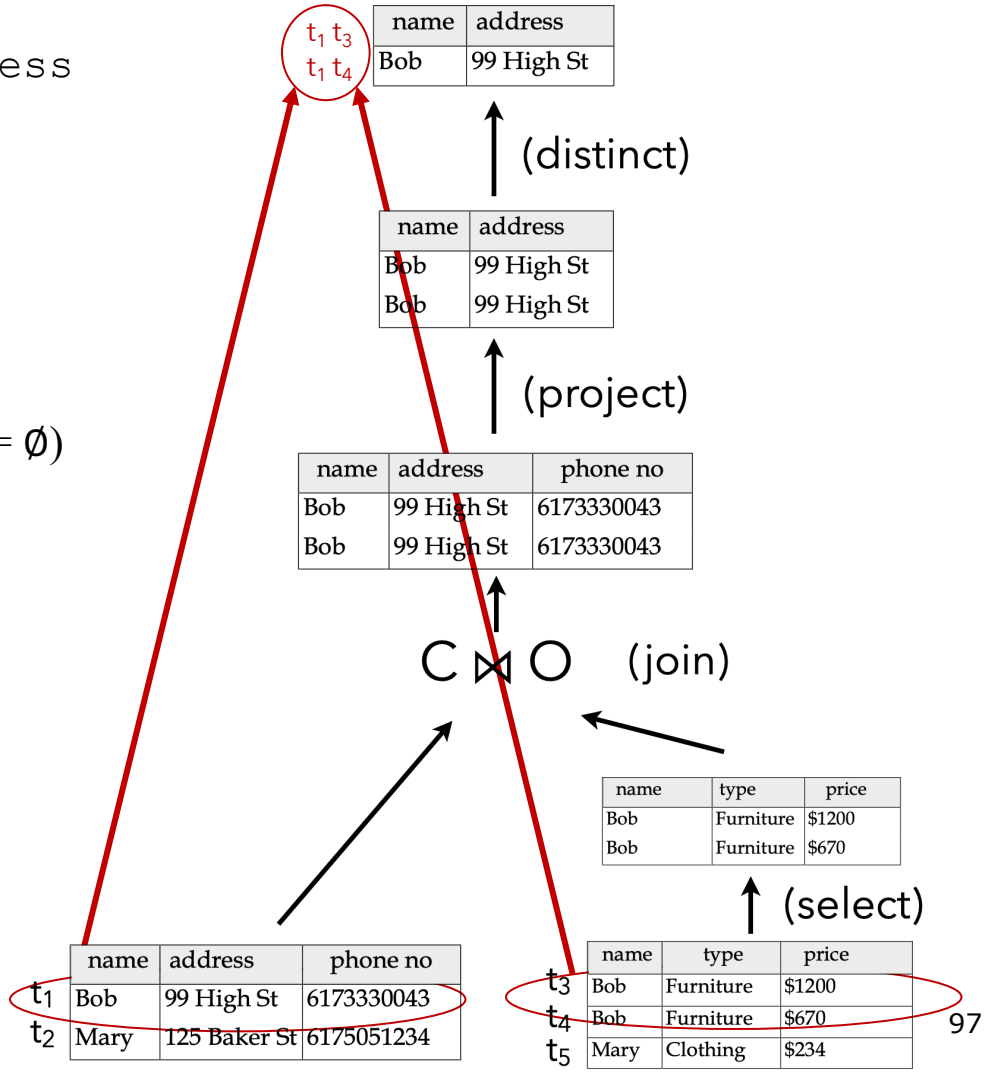
```
SELECT DISTINCT C.name, C.address
FROM Customers C, Orders O
WHERE C.name = O.name
  AND O.type="Furniture"
```

$\rho(t_1) = ?, \rho(t_3) = ?, \rho(t_4) = ?$

$\rho(t_1) = 1$ (t_1 is a counterfactual cause, i.e., $\Gamma = \emptyset$)

$\rho(t_3) = 0.5$ ($\Gamma = \{t_4\}$)

$\rho(t_4) = 0.5$ ($\Gamma = \{t_3\}$)



Another example (with aggregation)

```
SELECT R.MID
FROM (
    SELECT R.MID, AVG(R.Rating) as score
    FROM Friends as F, Ratings as R
    WHERE F.UID2 = R.UID
       AND F.UID1 = 'Alice'
    GROUPBY R.MID
    ORDERBY score DESC
    LIMIT 1 )
```

Friends

	UID1	UID2
t_1	Alice	Dustin
t_2	Alice	Mary
t_3	Alice	Bob
	…	…

r =(Star Wars)

$\rho(t_1) = ?, \rho(t_2) = ?, \rho(t_3) = ?$

Ratings

	UID	MID	Rating
t_4	Dustin	Star Wars	5
t_5	Bob	Star Wars	4
t_6	Mary	Blade Runner	2
	…	…	…

$\rho(t_1) = 0.5$ ($\Gamma=\{t_3\}$ or $\Gamma=\{t_5\}$)

$\rho(t_2) = 0$ (not an actual cause)

$\rho(t_3) = 0.5$ ($\Gamma=\{t_1\}$ or $\Gamma=\{t_4\}$)

Contingencies are not necessarily unique

There is no contingency that makes t_2 counterfactual

Responsibility vs Provenance

- Given:
  - A database *D*
  - A query *q* that produces a result *q(D)*
  - A tuple *r* in the result *q(D)*
- The set of actual causes for *r* is the union of tuples in the minimal witness basis of *r*
- The minimal witness basis includes those witnesses in the why-provenance of *r* that are not subsumed by any other witness
- The set of actual causes for *r* is a subset of *r*'s lineage

Back to our example

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

r =(Bob, 99 High St)

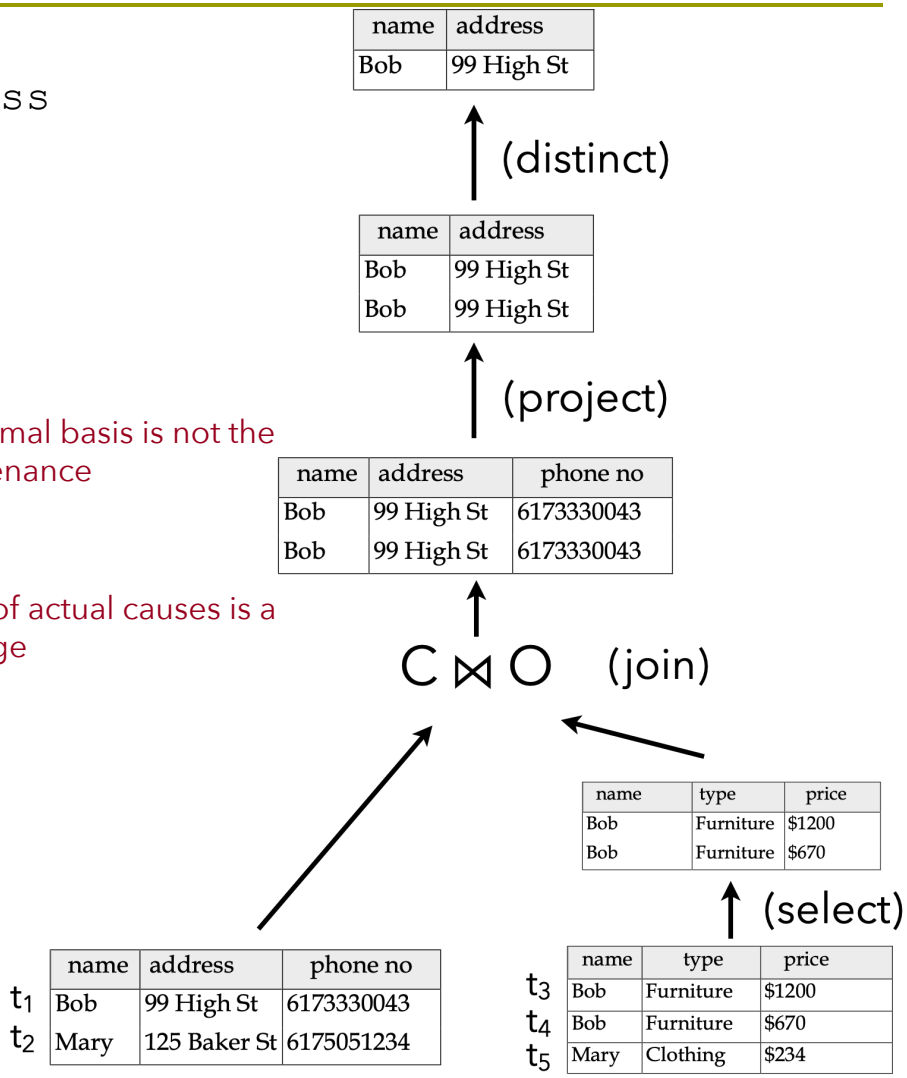
Why(r) = {{t_1, t_3}, {t_1,t_4}}

In general, the minimal basis is not the same as Why-provenance

Minimal Basis = {{t_1, t_3}, {t_1,t_4}}

Actual causes = {t_1, t_3, t_4}

In general, the set of actual causes is a subset of the lineage



Computing responsibility

- Given:
  - A database *D*
  - A query *q* that produces a result *q(D)*
  - A tuple *r* in the result *q(D)*
  - A tuple *t* in the input
- Compute $\rho(t)$
  - To do so, we need to find the contingency set with the minimum size amongst all possible contingencies that make *t* a counterfactual cause for *r*
- In general, this is an NP-hard problem
  - But for some classes of SQL queries can be done in PTIME[1]

[1] A. Meliou et al. *The Complexity of Causality and Responsibility for Query Answers and non-Answers,* VLDB, 2011

Computing responsibility: Baseline algorithm

- Let $s = 1$ be the size of Γ in the current step
- For each set of tuples Γ of size *s:*
  - Check if *t* is a counterfactual cause for *r* in *D*—Γ
  - If found such a contingency Γ:
  - Return $\rho(t) = \frac{1}{1+s}$
- Else: set s = s +1 and continue with the next step

- Each step requires $O(\frac{|D^n|}{s})$ checks to try all possible contingencies Γ of size *s*

Summary

- Data provenance is a well-studied problem with many useful applications
- Lineage was an initial, simple definition of provenance
- Provenance has widely been studied in database systems
  - where-, why-, how-, why-not provenance
- Data causality/responsibility finds how much each input tuple contributes to a query output