TreeToaster: Enabling Declarative Compilers

Darshana Balakrishnan[*†], Oliver Kennedy[†], Lukasz Ziarek[†],
Johannes Luong[*], Hinnerk Gildhoff[*], Gaurav Saxena[*]

Amazon[*], *University At Buffalo*[†]

Dec 12, 2024

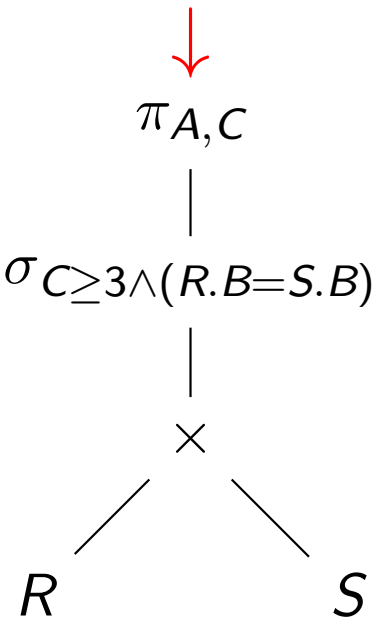
Compilers

```
SELECT A, C
FROM R, S
WHERE C >= 3 AND R.B = S.B
```

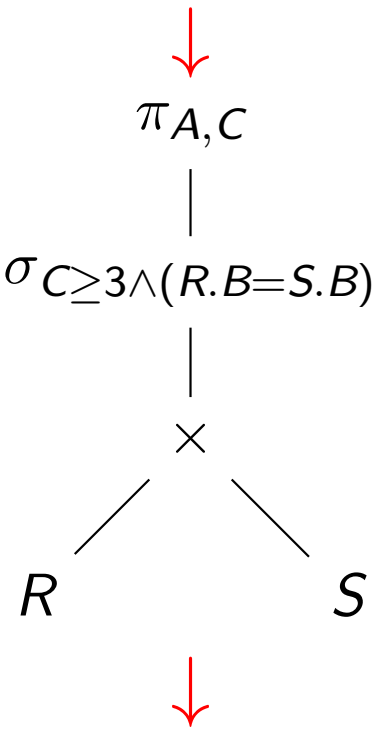
Compilers

```
SELECT A, C
FROM R, S
WHERE C >= 3 AND R.B = S.B
```

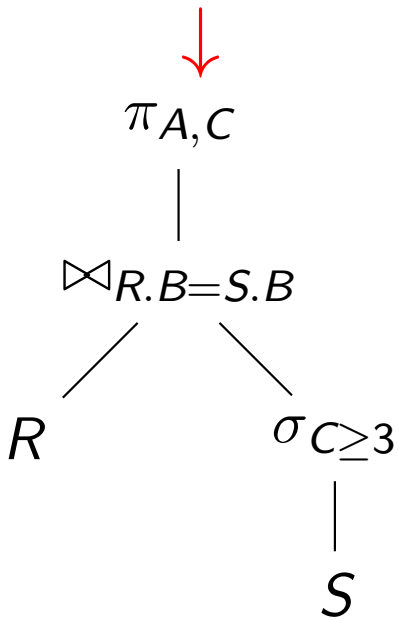
Parsing



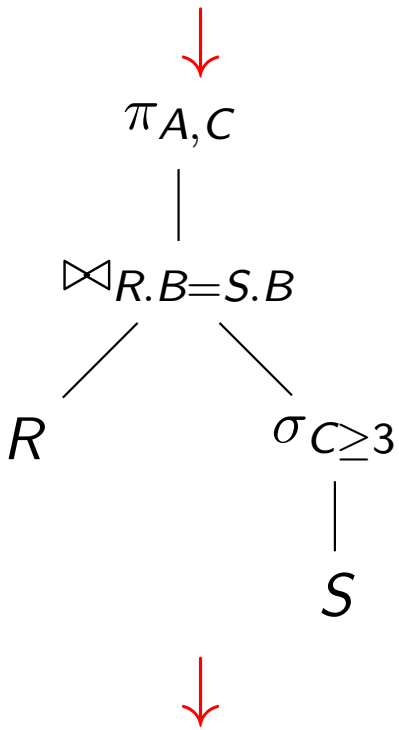
Parsing



Optimization



Optimization



Translation

```
data = {}
for r in R:
  data[r.B] = r
for s in S:
  if s.C >= 3:
    r = data[s.B]
    print(r.A, s.C)
```

Translation

```
data = {}
for r in R:
  data[r.B] = r
for s in S:
  if s.C >= 3:
    r = data[s.B]
    print(r.A, s.C)
```

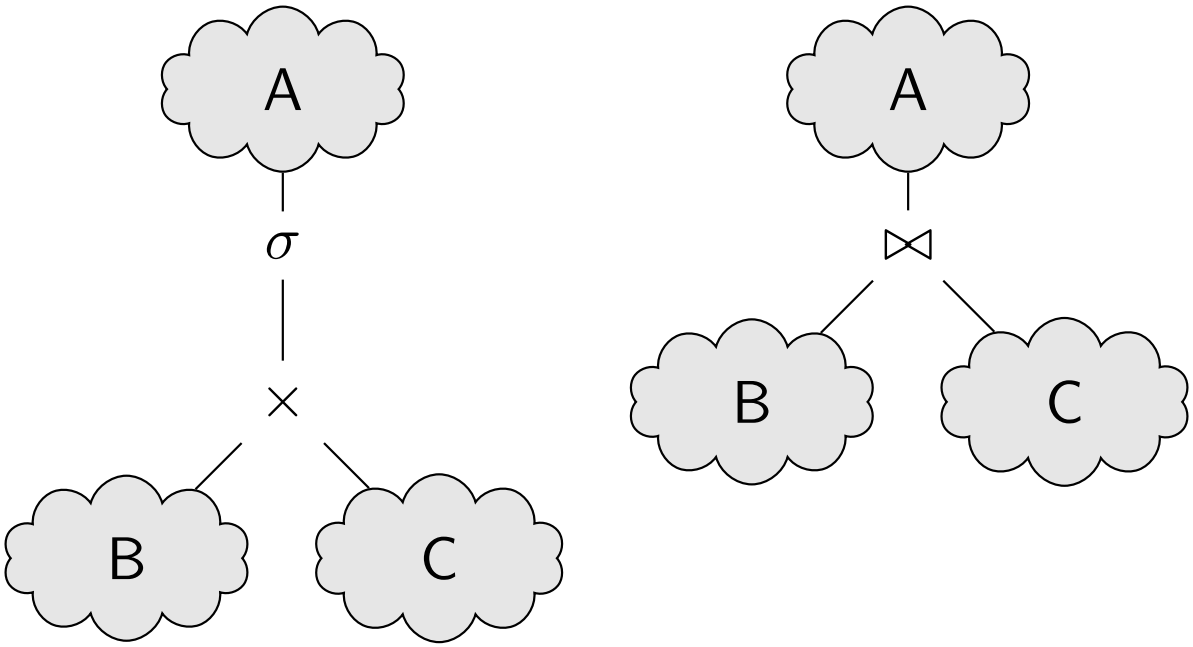
Analysis

```
data = {}
for r in R:                 # 10^6 rows
  data[r.B] = r
for s in S:                 # 10 rows
  if s.C >= 3:              # 50% selectivity
    r = data[s.B]           # 100% selectivity
    print(r.A, s.C)
```

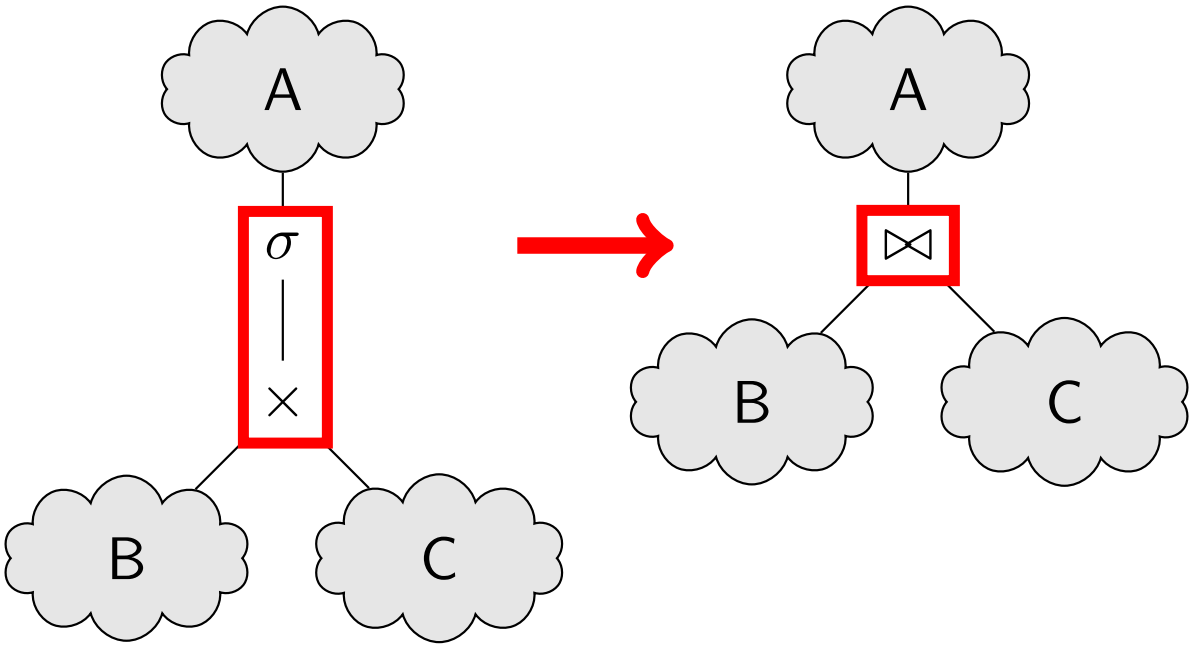
Match/Case Expressions

```
Q = Q match {
  case Filter(Equals(A, B), Cross(R, S))
    if A in R.sch && B in S.sch =>
      Join(R, S, A, B)
  case x => x
}
```

Local Reasoning



Local Reasoning



Optimization

$$\texttt{rule}_1(Q) \rightarrow \text{better} Q$$

Optimization

$$\texttt{rule}_1(Q) \rightarrow \text{better}Q$$

$$\texttt{rule}_2(\text{better}Q) \rightarrow \text{better}^2 Q$$

$$\texttt{rule}_3(\text{better}^2 Q) \rightarrow \text{better}^3 Q$$

$$\ldots$$

Optimization

$$\texttt{rule}_1(Q) \rightarrow \text{better}Q$$

$$\texttt{rule}_2(\text{better}Q) \rightarrow \text{better}^2 Q$$

$$\texttt{rule}_3(\text{better}^2 Q) \rightarrow \text{better}^3 Q$$

$$\ldots$$

$$\texttt{rule}_1(\text{better}^N Q) \rightarrow \text{better}^{N+1} Q$$

Optimization

$$\texttt{rule}_1(Q) \rightarrow \text{better}Q$$

$$\texttt{rule}_2(\text{better}Q) \rightarrow \text{better}^2 Q$$

$$\texttt{rule}_3(\text{better}^2 Q) \rightarrow \text{better}^3 Q$$

$$\ldots$$

$$\texttt{rule}_1(\text{better}^N Q) \rightarrow \text{better}^{N+1} Q$$

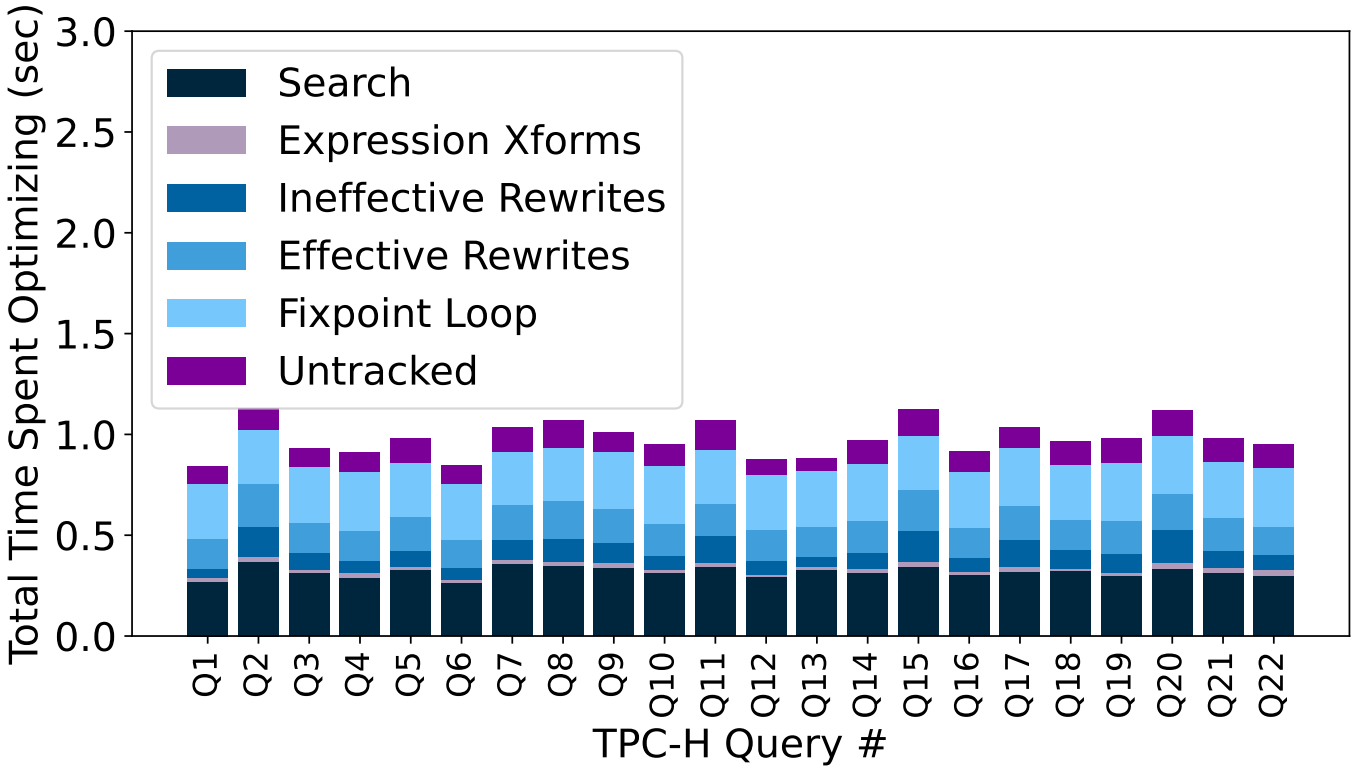
$$\texttt{rule}_2(\text{better}^{N+1} Q) \rightarrow \text{better}^{N+2} Q$$

$$\ldots$$

Fixed Point Loops

```
while AST is being changed:
  for rule in rules:
    for node in AST:
      if rule matches node:
        replace node with rule(node)
```

Apache Spark / Catalyst



Fixed Point Loops

```
while AST is being changed:
  for rule in rules:
    for node in AST:
      if rule matches node:
        /* ... */
```

The Fixed Point Loop Abstraction

Pro

- Simple
- Easy to Reason About

Con

- Slow
- Limited Expressiveness

A Broader Perspective

We've been talking about queries...

...but the same ideas show up in compilers in general.

Compilers are Databases

```
case Filter(Equals(A, B), Cross(R, S)) if ... =>
  Join(R, S, A, B)
```

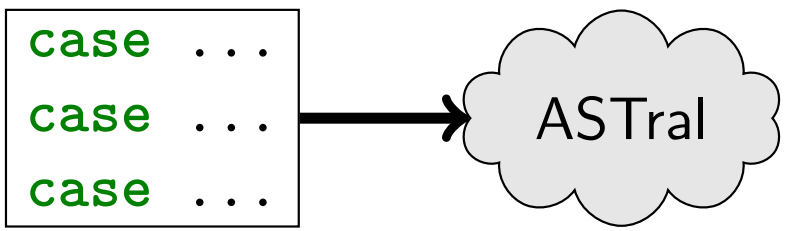
vs

```
UPDATE ast SET node = Join(R, S, A, B)
   WHERE node LIKE Filter(Equals(A, B), ....
```

The ASTral Compiler

```
case ...
case ...
case ...
```

The ASTral Compiler



The ASTral Compiler

Overview

- Optimization Rules as Queries
- Evaluating ASTral
- Indexing & Incremental View Maintenance
- State Machines for Multiquery Optimization

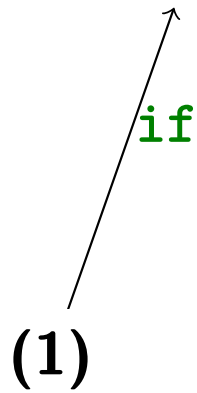
Optimization Rules as Queries

Breaking down a Pattern

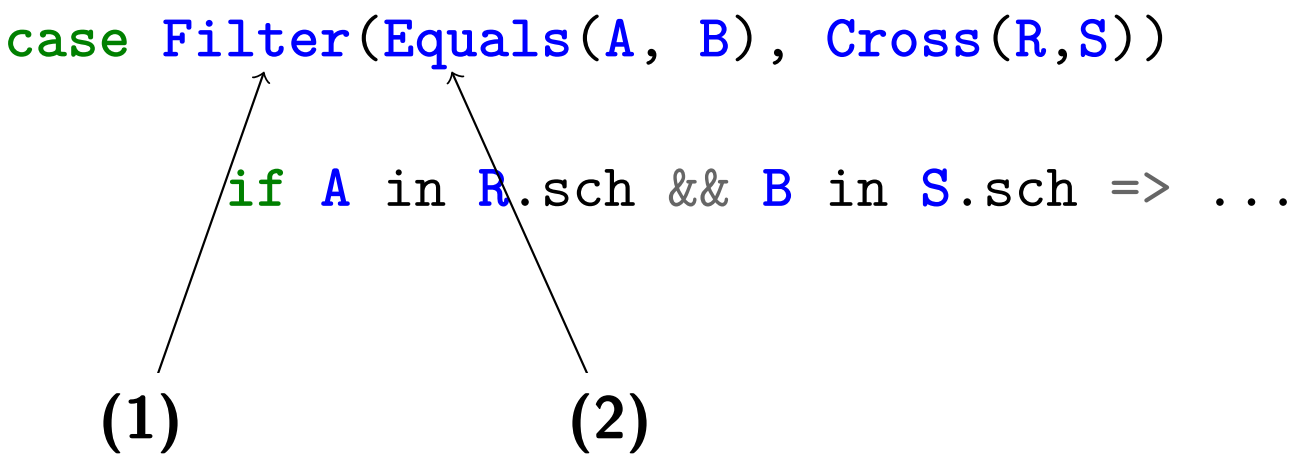
```
case Filter(Equals(A, B), Cross(R,S))

        if A in R.sch && B in S.sch => ...
```

Breaking down a Pattern

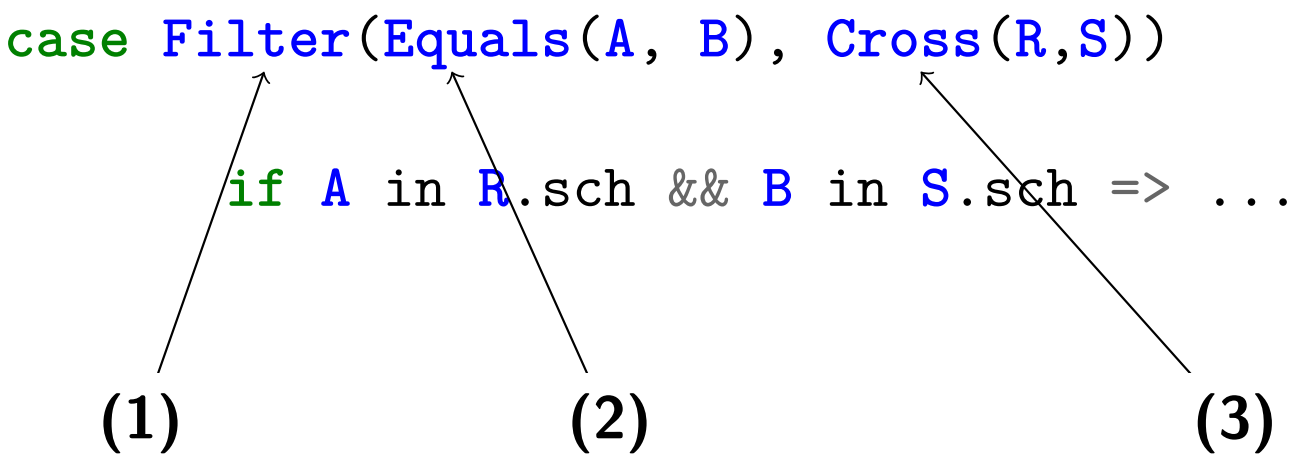
```
case Filter(Equals(A, B), Cross(R,S))
       if A in R.sch && B in S.sch => ...
```



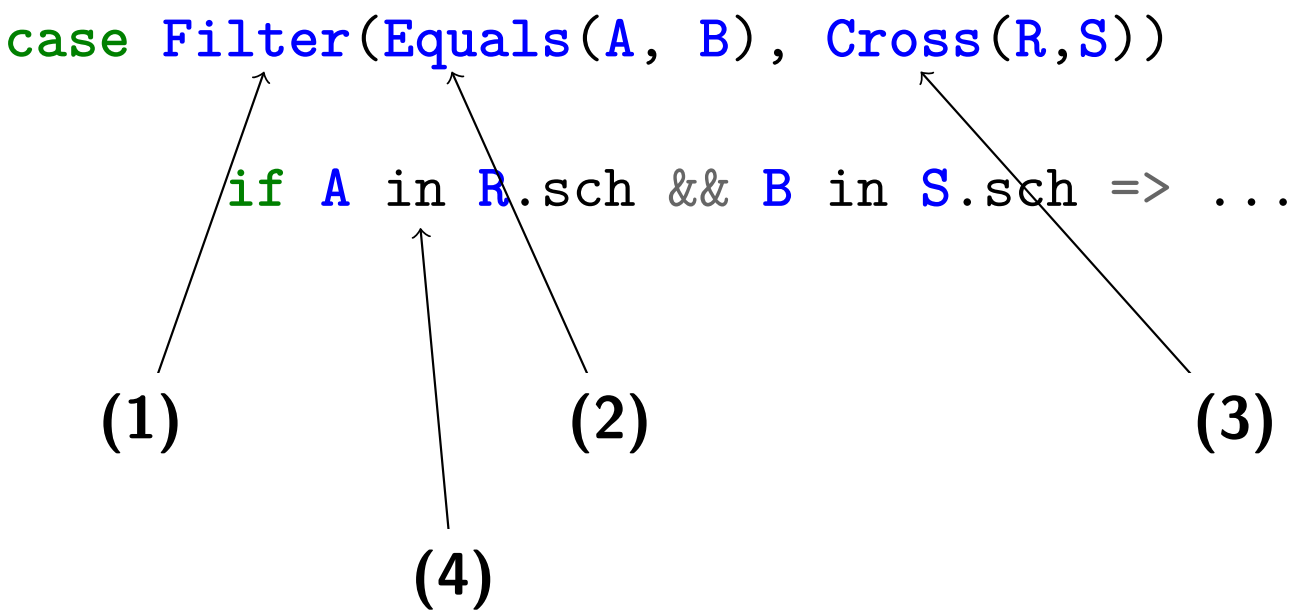
Breaking down a Pattern



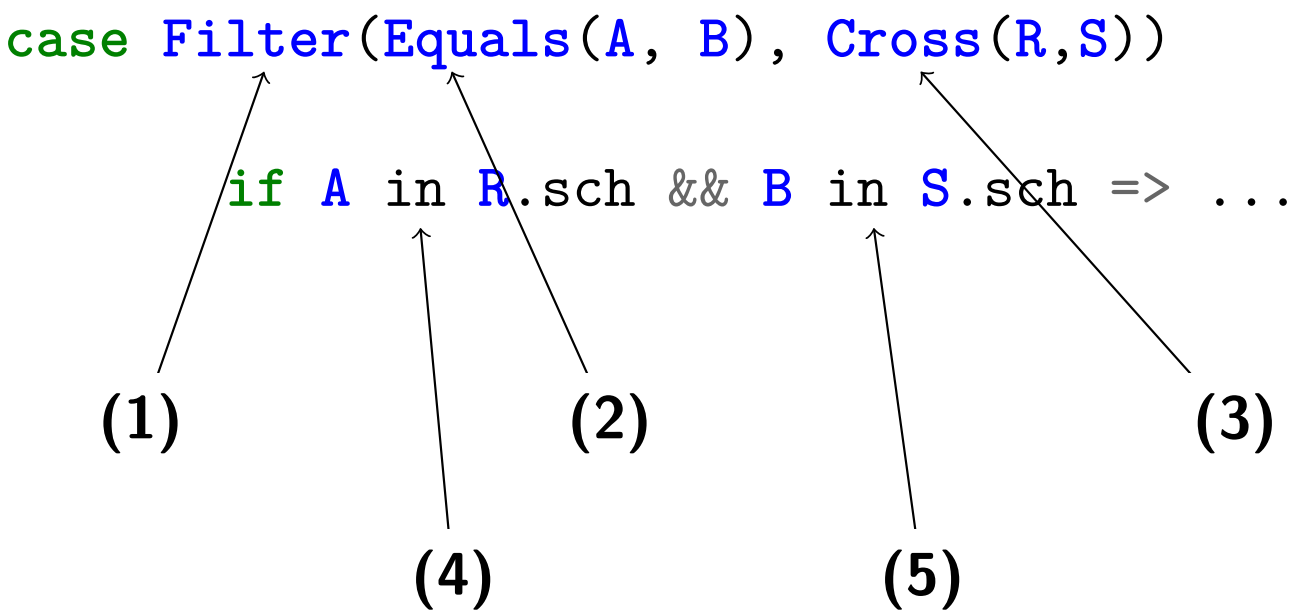
Breaking down a Pattern



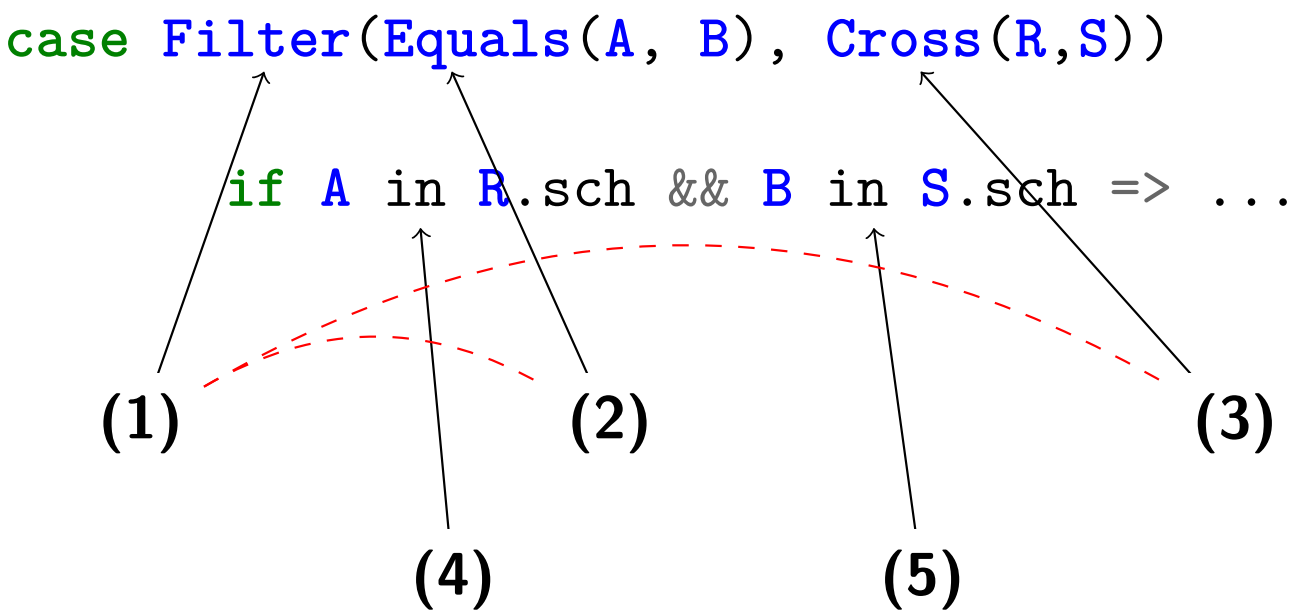
Breaking down a Pattern



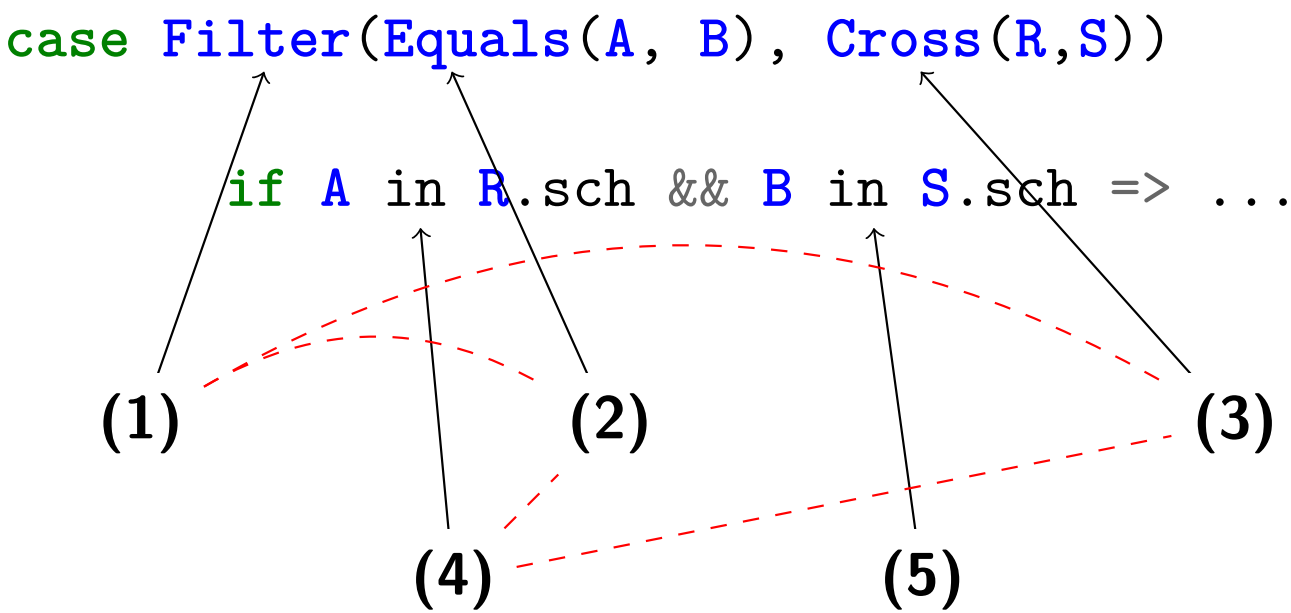
Breaking down a Pattern



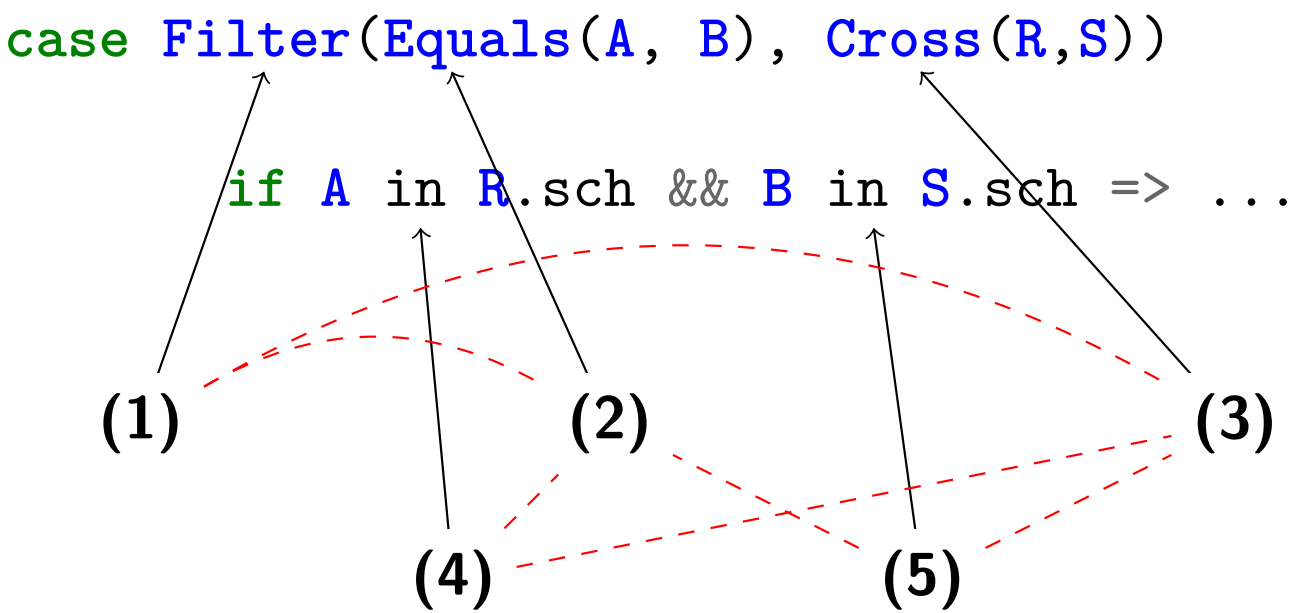
Breaking down a Pattern



Breaking down a Pattern



Breaking down a Pattern



ASTral (The AST-Relational Algebra)

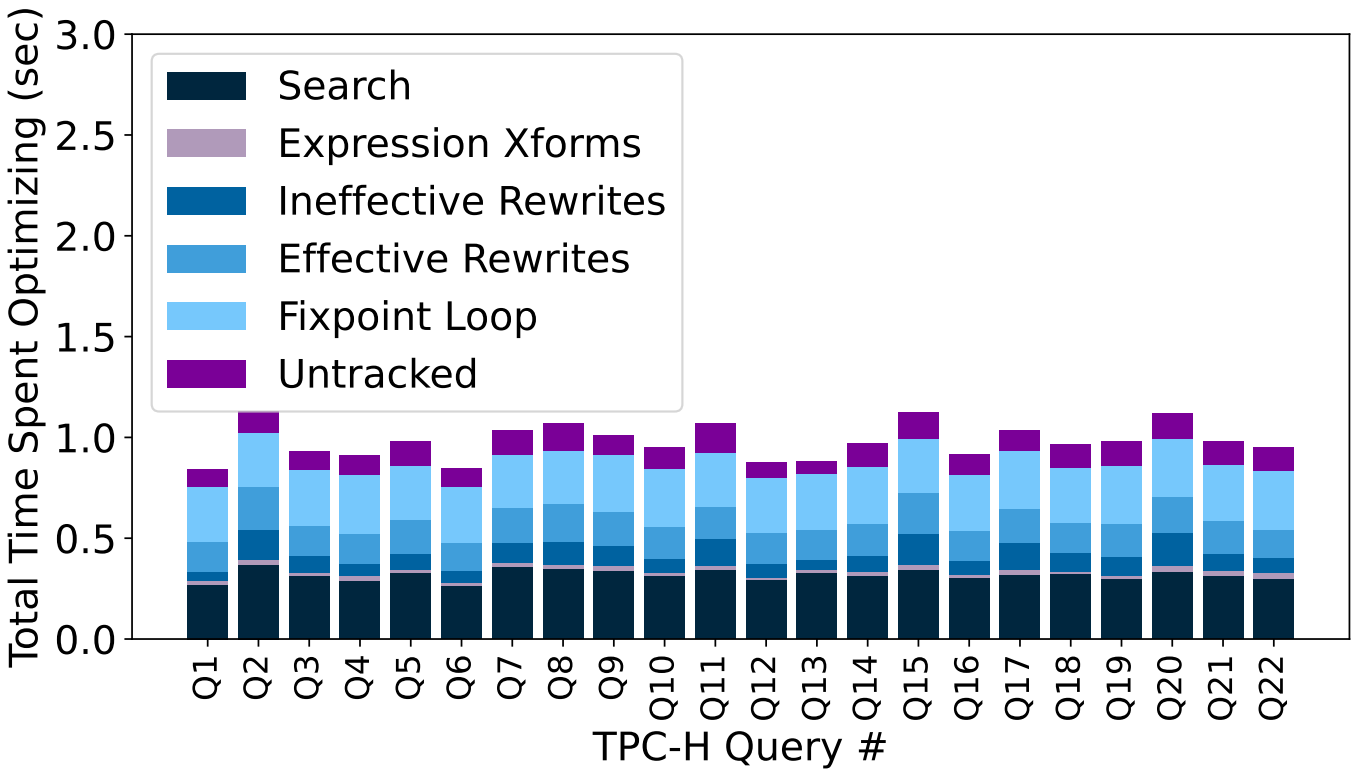
1. $\mathbf{Node} \sim \texttt{Filter}(\mathbf{X}, \mathbf{Y})$
2. $\wedge\ \mathbf{X} \sim \texttt{Equals}(\mathbf{A}, \mathbf{B})$
3. $\wedge\ \mathbf{Y} \sim \texttt{Cross}(\mathbf{R}, \mathbf{S})$
4. $\wedge\ \mathbf{A} \in \mathbf{R}.\texttt{sch}$
5. $\wedge\ \mathbf{B} \in \mathbf{S}.\texttt{sch}$

Evaluating ASTral

Compiled

```
for X in descendants(ROOT):
  if X is Filter:
    Y = X.condition
    Z = X.child
    if Y is Equality;
      A = Y.lhs
      B = Y.rhs
      if Z is Cross;
        R = Z.lhs
        S = Z.rhs
        if A in R.sch:
          if B in S.sch:
            replace X with Join(A, B, R, S)
```

Optimizer Performance



Performance Opportunities

```
for X in descendants(ROOT):
  if X is Filter:
    Y = X.condition
    Z = X.child
    if Y is Equality;
      A = Y.lhs
      B = Y.rhs
      if Z is Cross;
        R = Z.lhs
        S = Z.rhs
        if A in R.sch:
          if B in S.sch:
            replace X with Join(A, B, R, S)
```

So now what?

```
for X in descendants(ROOT):
  if X is Filter:
    ...
```

So now what?

```
for X in descendants(ROOT):
  if X is Filter:
    ...
```

Build an index on $\mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \wedge \mathbf{X} \sim \texttt{Filter}(\_, \_)$

Indexing

Indexing

```
for X in descendants(ROOT):
  if X is Filter:
    Y = X.condition
    Z = X.child
    ...
```

vs

```
for X in Index:
  Y = X.condition
  Z = X.child
  ...
```

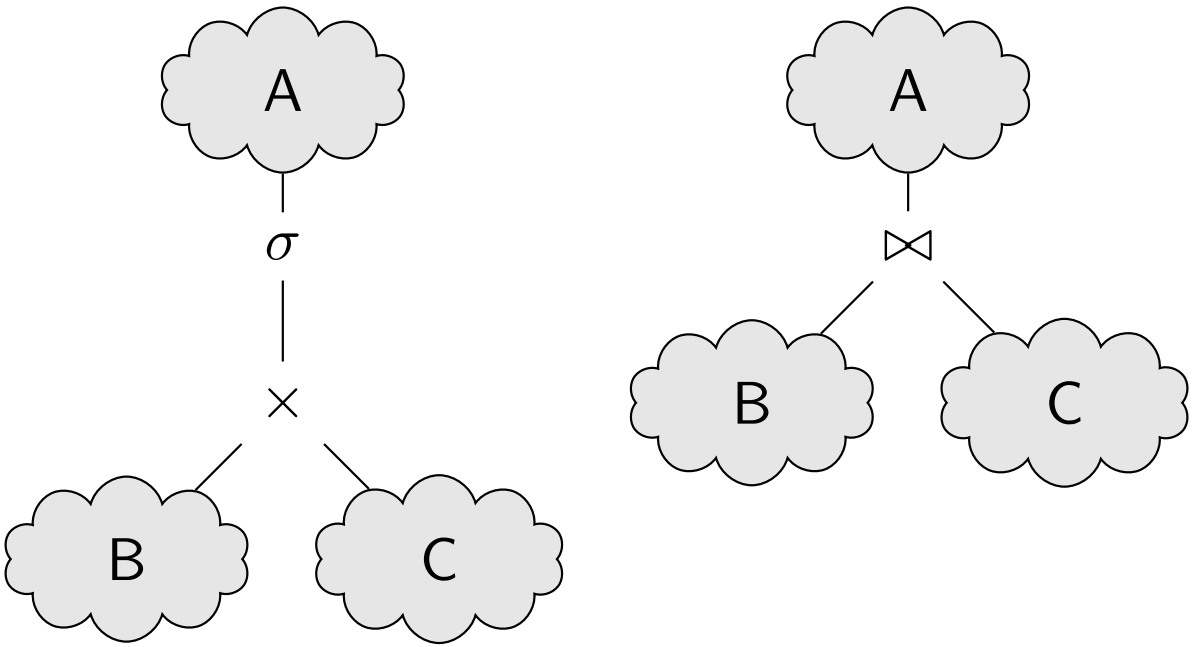
Index Maintenance

$$\mathit{Index}(X) :- \ \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \wedge \mathbf{X} \sim \texttt{Filter}(\_, \_)$$

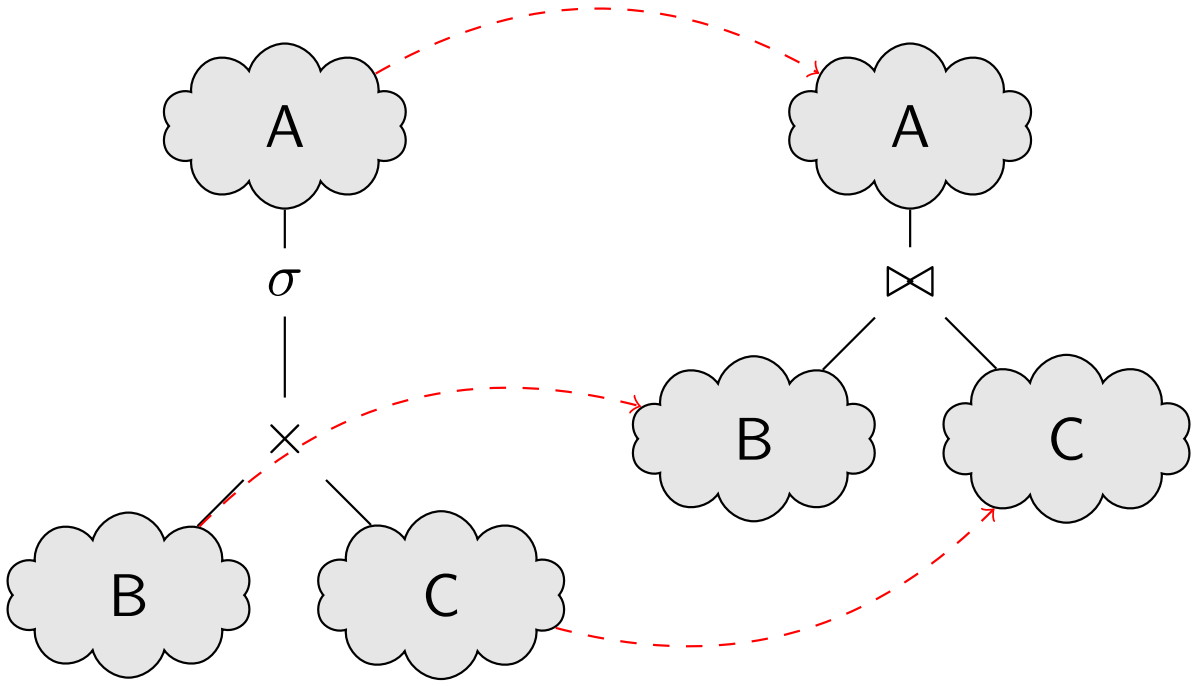
Say we replace $\sigma(\times(R, S))$ in the tree with $\bowtie(R, S)$.

How does $\mathit{Index}(X)$ change?

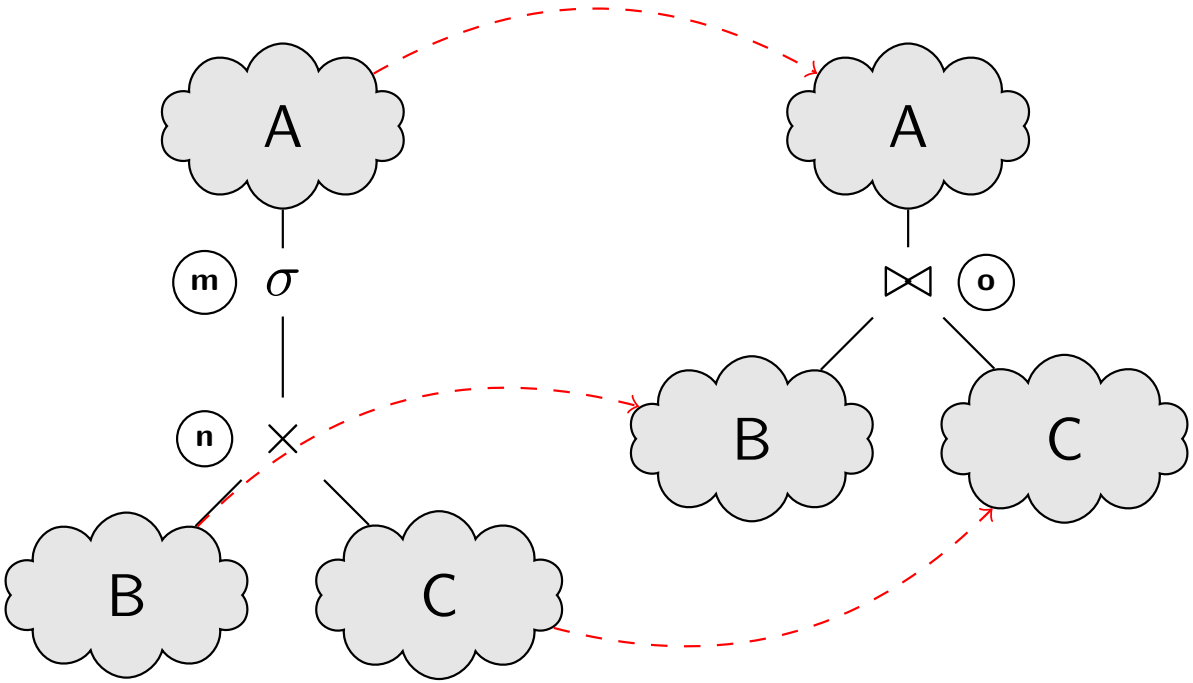
Index Maintenance



Index Maintenance



Index Maintenance



Why stop at indexing?

Why not just compute:

$$
\begin{aligned}
\mathit{Index}_1(X) :- \quad & \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \\
& \wedge\ \mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z}) \\
& \wedge\ \mathbf{Y} \sim \texttt{Equals}(\mathbf{A}, \mathbf{B}) \\
& \wedge\ \mathbf{Z} \sim \texttt{Cross}(\mathbf{R}, \mathbf{S}) \\
& \wedge\ \mathbf{A} \in \mathbf{R}.\texttt{sch} \wedge \mathbf{B} \in \mathbf{S}.\texttt{sch}
\end{aligned}
$$

Why stop at indexing?

Why not just compute:

$$
\begin{aligned}
\mathit{Index}_1(X) :- \quad & \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \\
& \wedge\ \mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z}) \\
& \wedge\ \mathbf{Y} \sim \texttt{Equals}(\mathbf{A}, \mathbf{B}) \\
& \wedge\ \mathbf{Z} \sim \texttt{Cross}(\mathbf{R}, \mathbf{S}) \\
& \wedge\ \mathbf{A} \in \mathbf{R}.\texttt{sch} \wedge \mathbf{B} \in \mathbf{S}.\texttt{sch} \\
\mathit{Index}_2(X) :- \quad & \ldots \\
\mathit{Index}_3(X) :- \quad & \ldots
\end{aligned}
$$

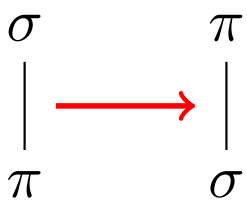
Incremental View Maintenance

Rewrites Under IVM

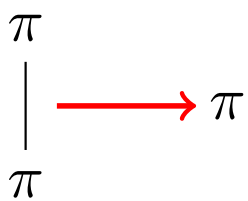
```
while AST is being changed:
  for rule in RULES:
    while Index[rule] is not empty:
      rewrite Index[rule][0] with rule
      update Indexes
```

But...

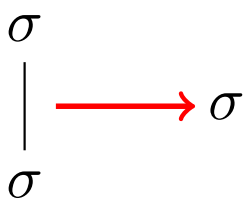
Rule 1



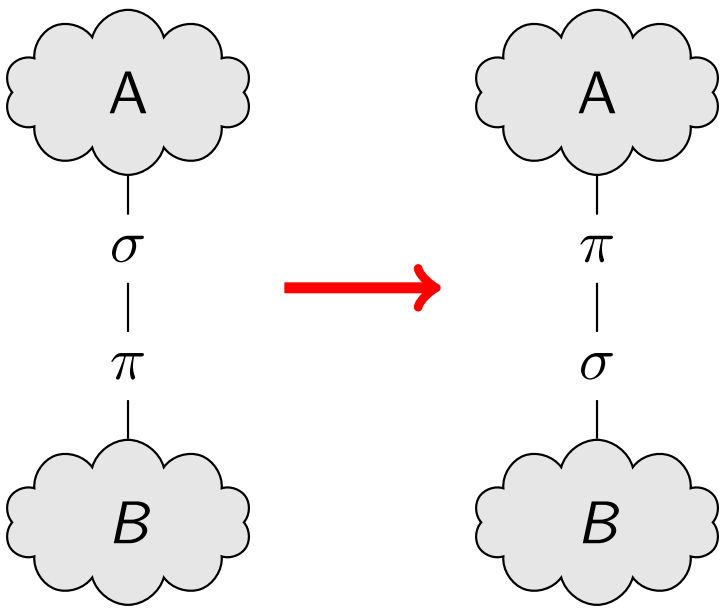
Rule 2



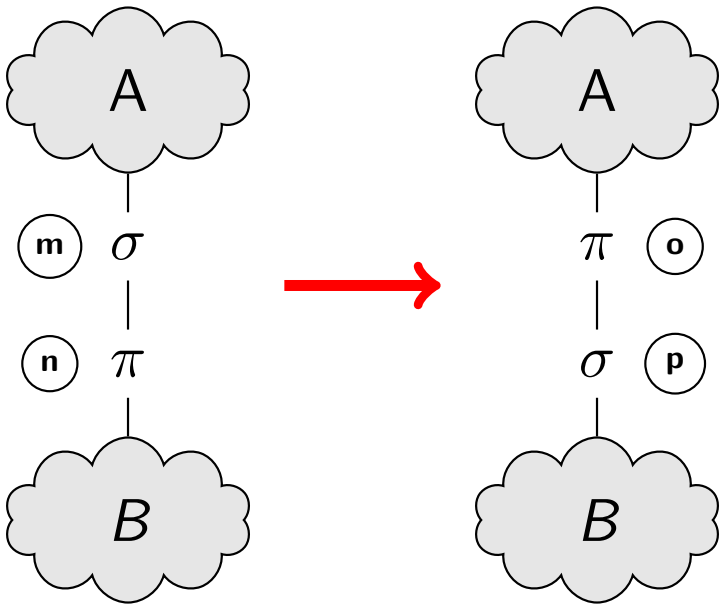
Rule 3



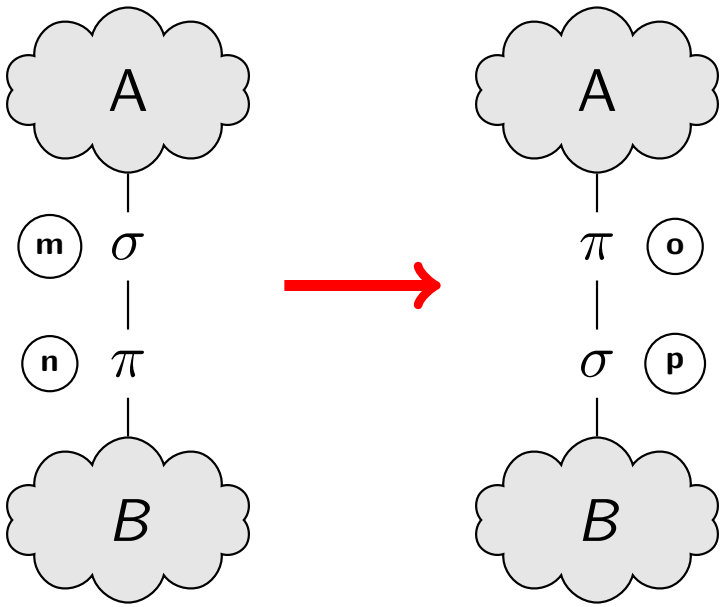
Updating Materialized Views



Updating Materialized Views

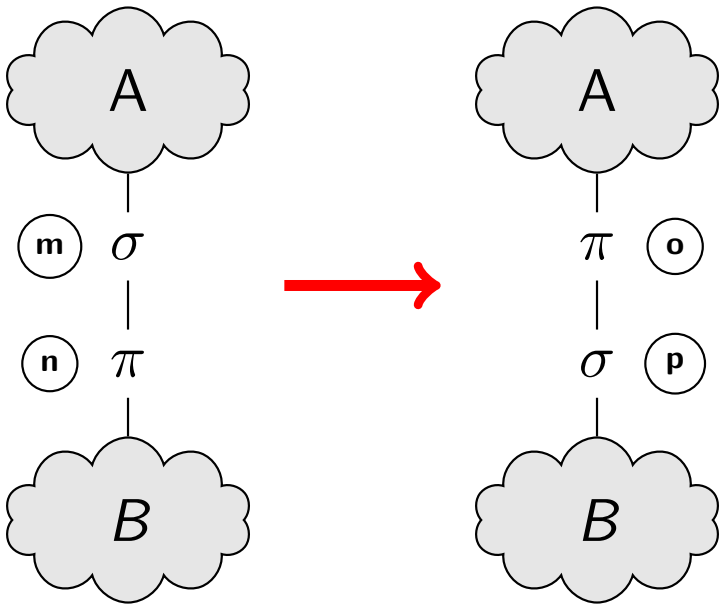


Updating Materialized Views



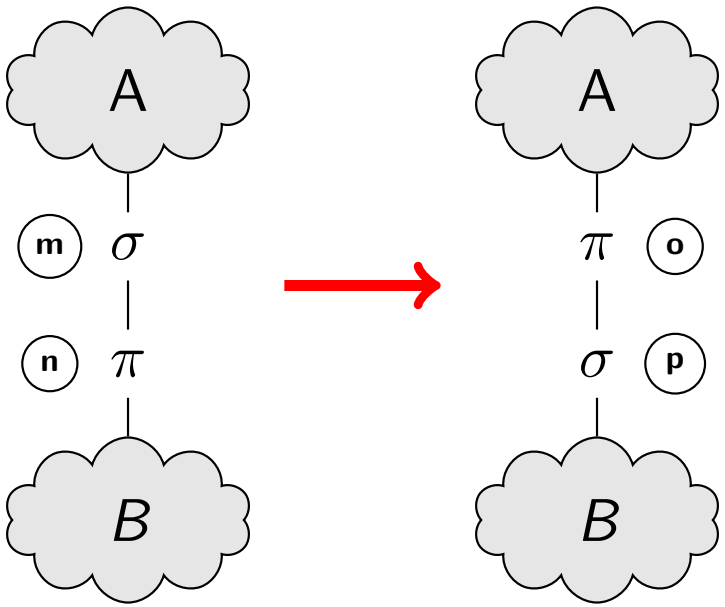
- Remove *n*, *m* from indices.

Updating Materialized Views



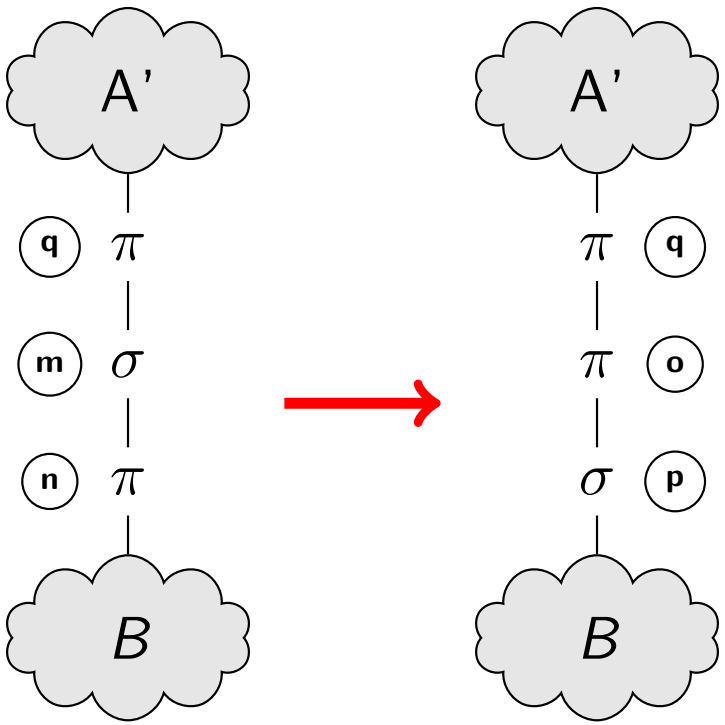
- Remove n, m from indices.
- Check o, p for matches.

Updating Materialized Views

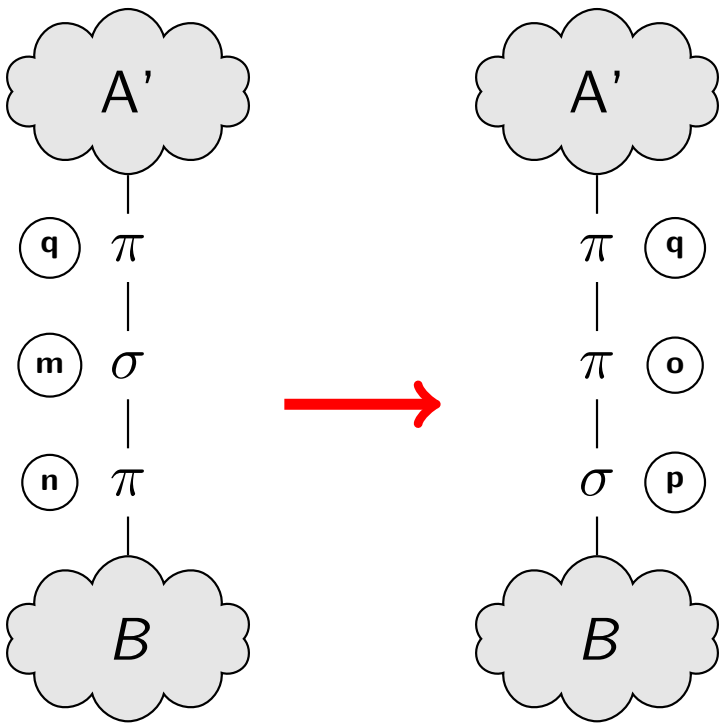


- Remove *n*, *m* from indices.
- Check *o*, *p* for matches.
- Done?

Updating Materialized Views



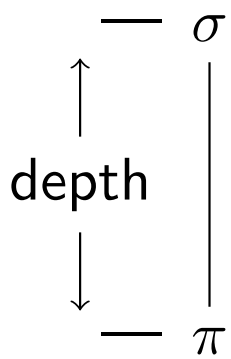
Updating Materialized Views



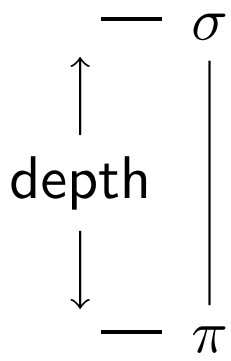
Problem: (q) now matches on Rule 2.

Depth-Bounded Search

Depth-Bounded Search



Depth-Bounded Search



A rule of depth d needs to check d ancestors for a match.

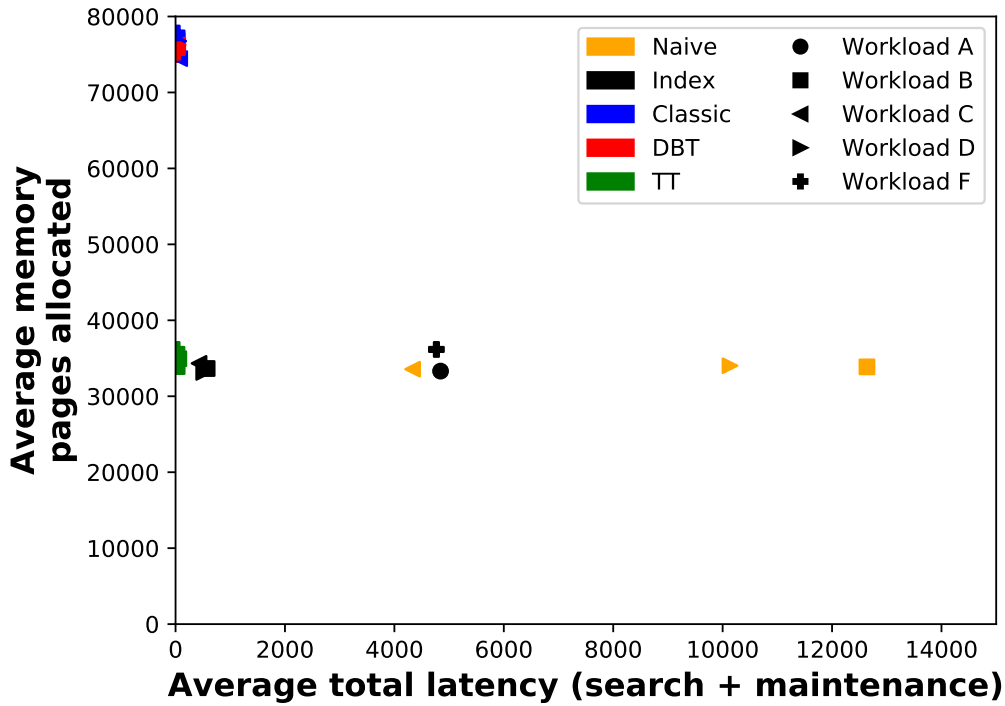
And also...

- In-Situ parallel updates on Trees
- Aggressive Code In-lining

"Fluid Data Structures"; Balakrishnan, Ziarek, Kennedy (DBPL 2019)

"Tree Toaster: Towards an IVM-Optimized Compiler", Balakrishnan et. al. (SIGMOD 2020)

Faster



"Tree Toaster: Towards an IVM-Optimized Compiler", Balakrishnan et. al. (SIGMOD 2020)

State Machines

Back to the fixed point loop...

```
while AST is being changed:
  for rule in rules:
    for node in AST:
      if node matches rule:
        rewrite node
```

Spatial Locality

```
while AST is being changed:
  for node in AST:
    for rule in rules:
      if node matches rule:
        rewrite node
```

Eliminate Redundancy

```
while AST is being changed:
  for node in AST:
    rule = match node in RuleIndex:
    rewrite node
```

De-duplicating Atoms

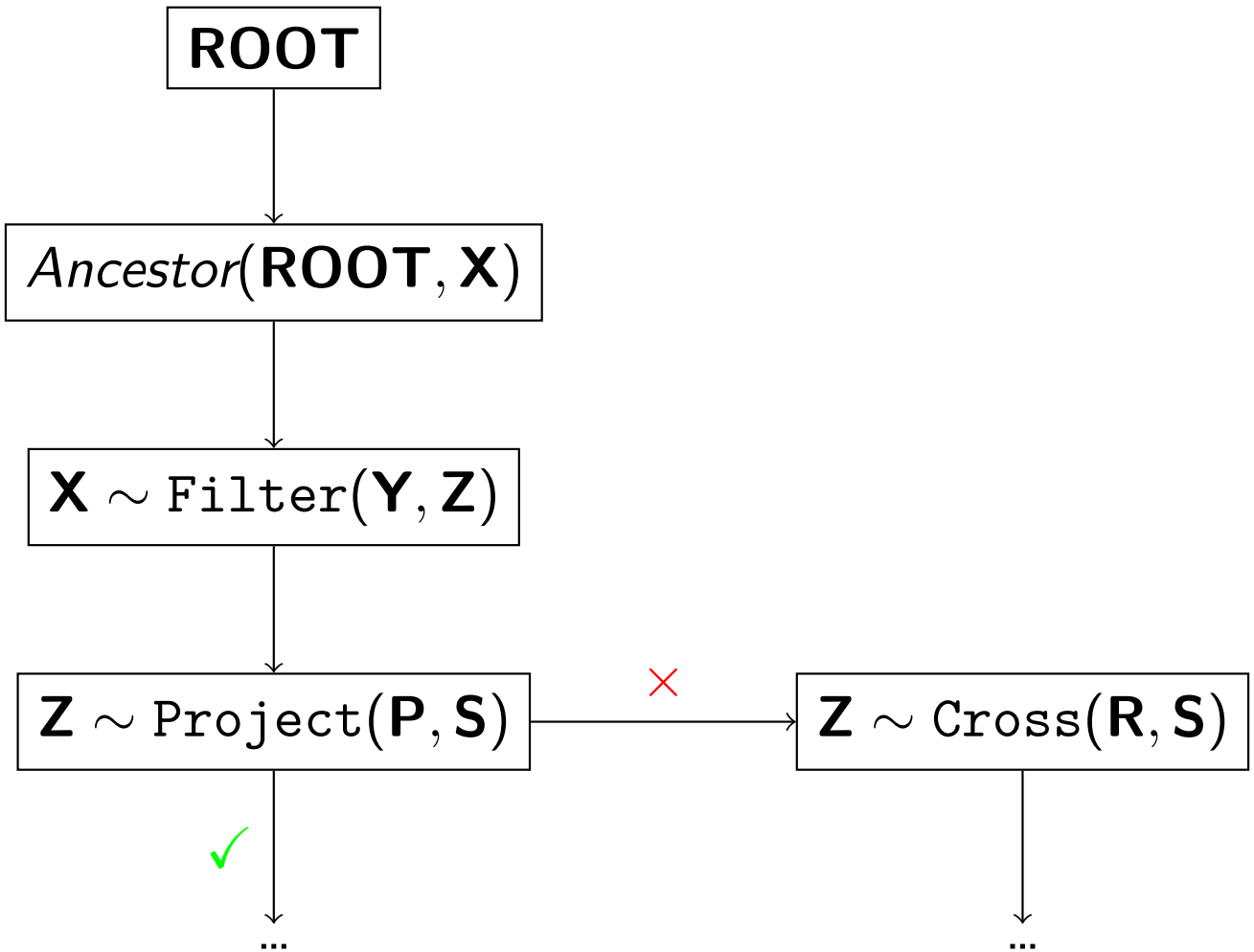
$$
\begin{aligned}
Q_1(\mathbf{X}) :- \quad & \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \\
& \wedge\ \mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z}) \\
& \wedge\ \mathbf{Y} \sim \texttt{Equals}(\mathbf{A}, \mathbf{B}) \\
& \wedge\ \mathbf{Z} \sim \texttt{Cross}(\mathbf{R}, \mathbf{S}) \\
& \wedge\ \mathbf{A} \in \mathbf{R}.\texttt{sch} \wedge \mathbf{B} \in \mathbf{S}.\texttt{sch} \\
Q_2(\mathbf{X}) :- \quad & \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \\
& \wedge\ \mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z}) \\
& \wedge\ \mathbf{Z} \sim \texttt{Project}(\mathbf{P}, \mathbf{S})
\end{aligned}
$$

De-duplicating Atoms

$$
\begin{aligned}
Q_1(\mathbf{X}) :- \quad & \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \\
& \wedge\ \mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z}) \\
& \wedge\ \mathbf{Y} \sim \texttt{Equals}(\mathbf{A}, \mathbf{B}) \\
& \wedge\ \mathbf{Z} \sim \texttt{Cross}(\mathbf{R}, \mathbf{S}) \\
& \wedge\ \mathbf{A} \in \mathbf{R}.\texttt{sch} \wedge \mathbf{B} \in \mathbf{S}.\texttt{sch} \\
Q_2(\mathbf{X}) :- \quad & \mathit{Ancestor}(\mathbf{ROOT}, \mathbf{X}) \\
& \wedge\ \mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z}) \\
& \wedge\ \mathbf{Z} \sim \texttt{Project}(\mathbf{P}, \mathbf{S})
\end{aligned}
$$

Q_1 and Q_2 share a prefix; Only check it once

State Machines



Expensive Predicates

Simple tests (e.g., $\mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z})$) are cheap.

Expensive Predicates

Simple tests (e.g., $\mathbf{X} \sim \texttt{Filter}(\mathbf{Y}, \mathbf{Z})$) are cheap.

...but some tests (does there exist a descendant with a columnar storage model?) present substantial opportunities for work sharing.

Conclusions

Databases for Compilers

Compilers are an exciting database workload!

The PL Community

- **Equality Saturation: A New Approach to Optimization**
  Tate et. al.
- **Better Together: Unifying Datalog and Equality Saturation**
  Zhang et. al.
- **Soufflé**
  `https://souffle-lang.github.io/`
- **Higher-Order, Data-Parallel Structured Deduction**
  Gilray et. al.

Visit Us!

Conclusions

Revisiting optimization (and translation and analysis) rules as queries creates opportunities for automatic optimization.

Decoupling compiler rewrite logic from performance optimizations makes each easier to reason about.

Find **Nick** to talk about program analysis on disk, and **Victoria** to talk about distributed IVM.
(and **Pratik** to talk about schema management for longitudinal surveys)