

Noria: Partially-Stateful Data-flow

Ashvin Goel

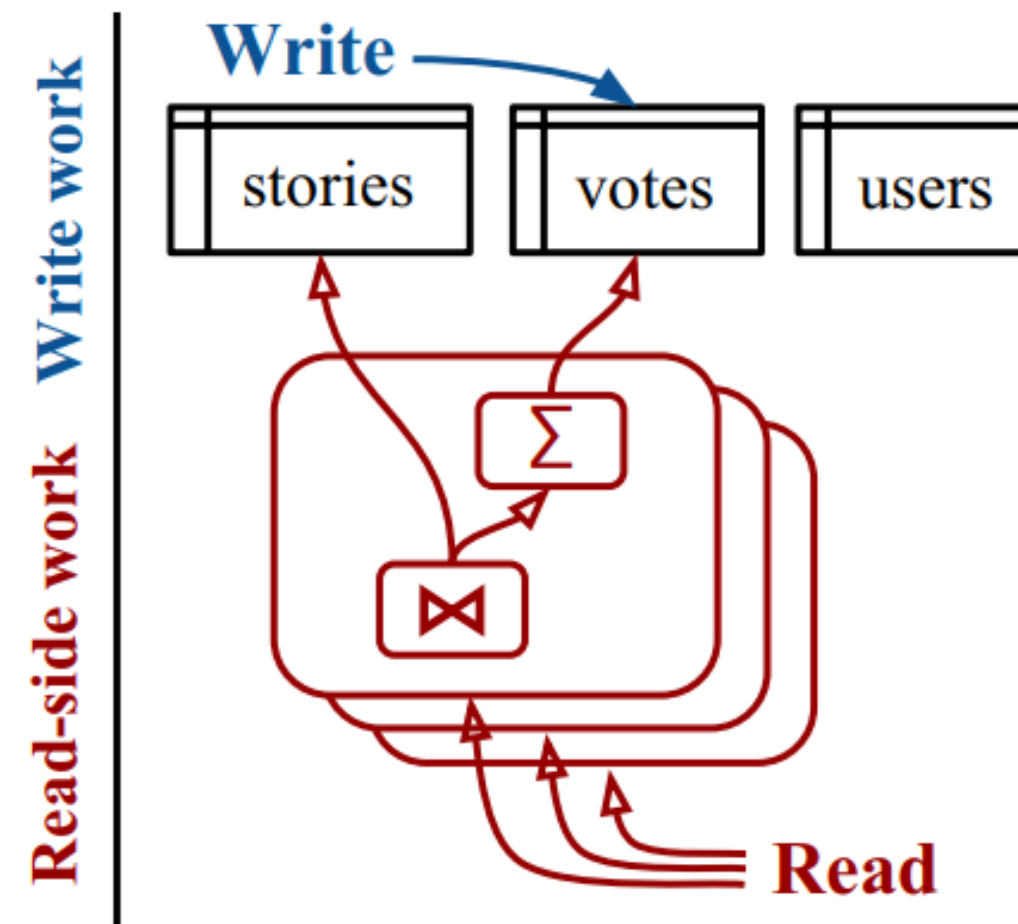
Electrical and Computer Engineering
University of Toronto

ECE1724

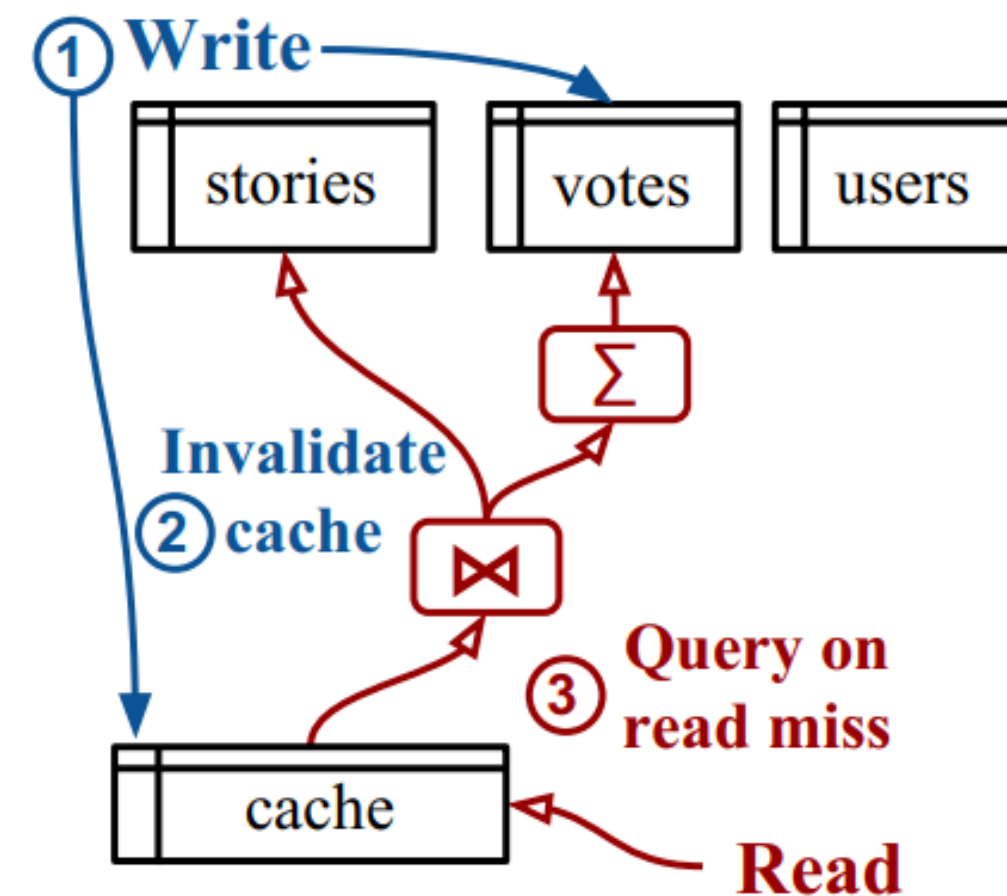
Authors: Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, and Lara Timbó Araújo, Martin Ek,
Eddie Kohler, M. Frans Kaashoek and Robert Morris

Motivation

Modern web apps add an in-memory cache in front of traditional databases for high performance



Classic database
reads are expensive

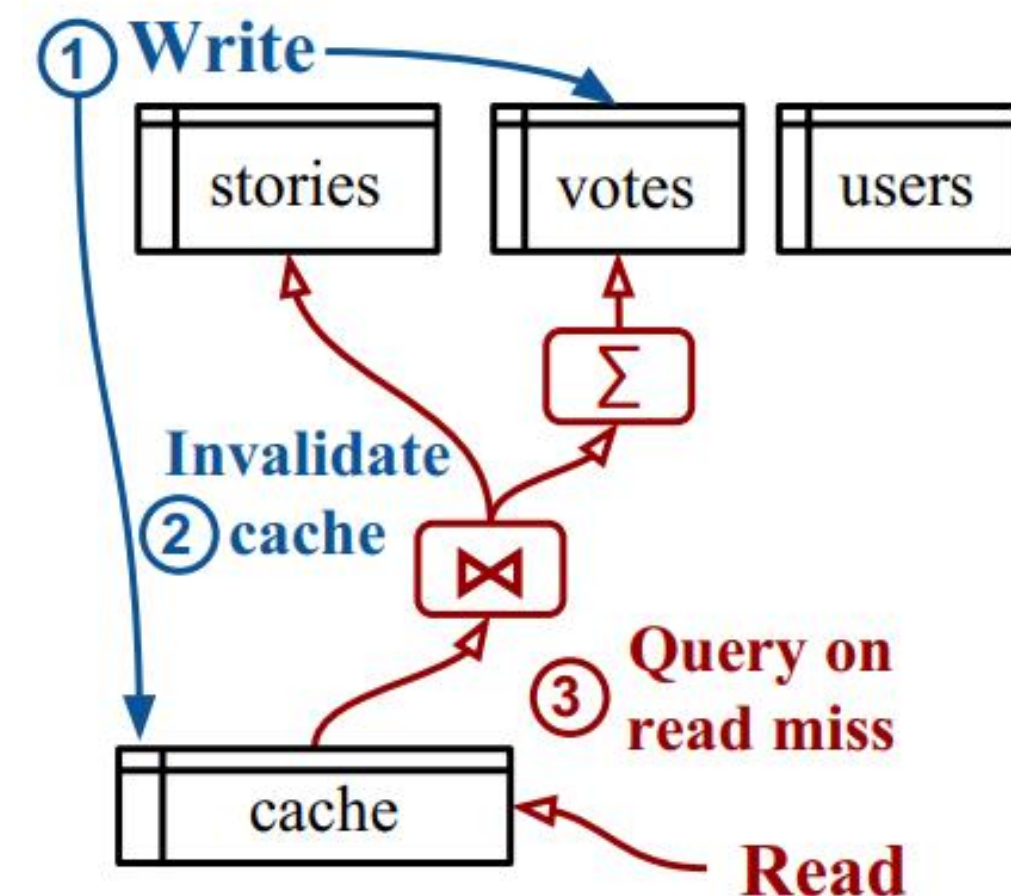


Database + in-memory cache
reads are fast

Motivation

In-memory cache work well, but programmer needs to handle invalidations correctly

Incorrect design may lead to stale data in caches forever



Database + in-memory cache
reads are fast

Motivation

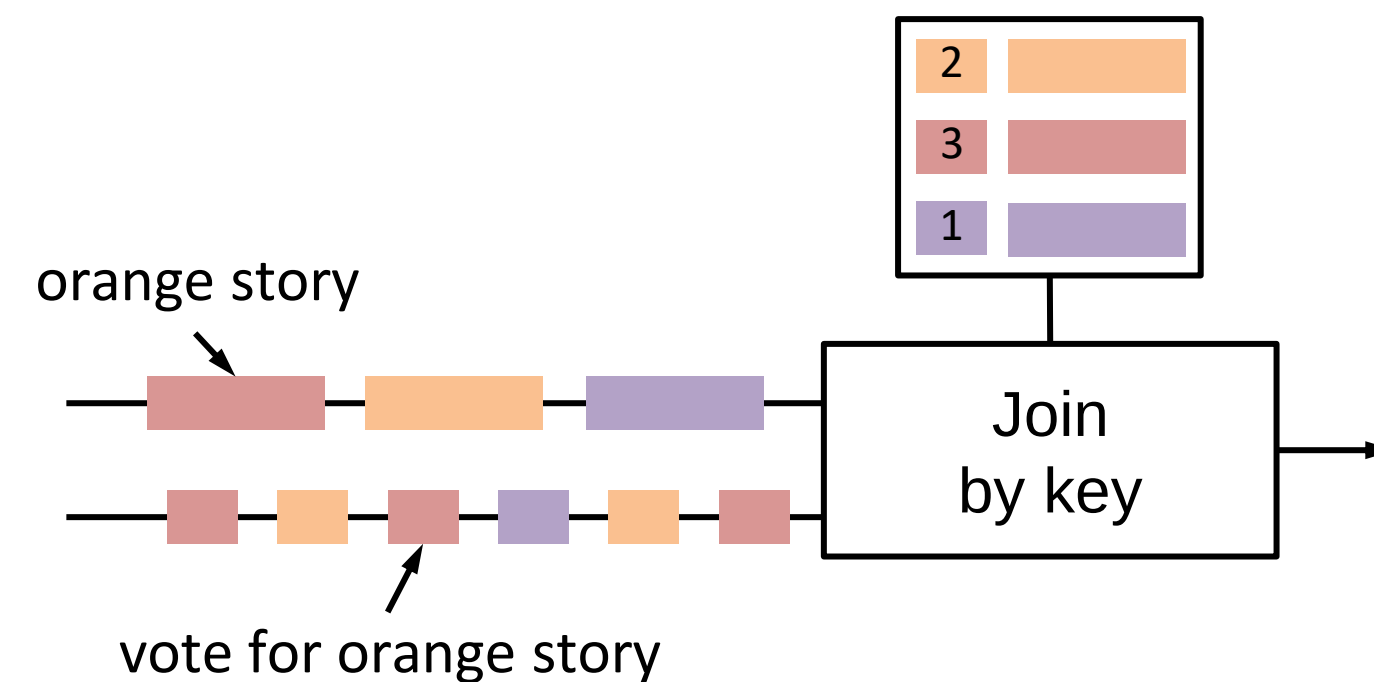
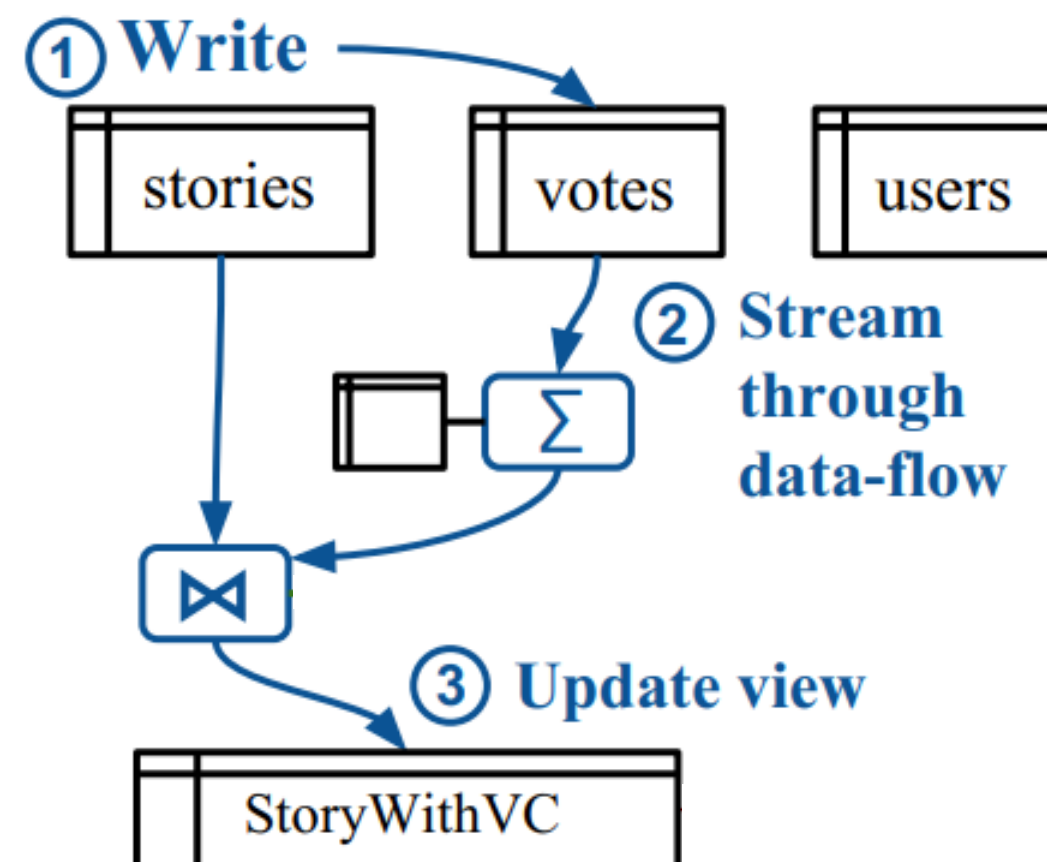
What about using a dataflow/streaming database?

Write path is expensive

Primarily support window operations

Generate data that may never be read

Require significant memory for caching



Key Idea in Noria

Use a partial-state dataflow model

Cache state on reads: Like traditional cache

Update cached state on writes: Like dataflow databases

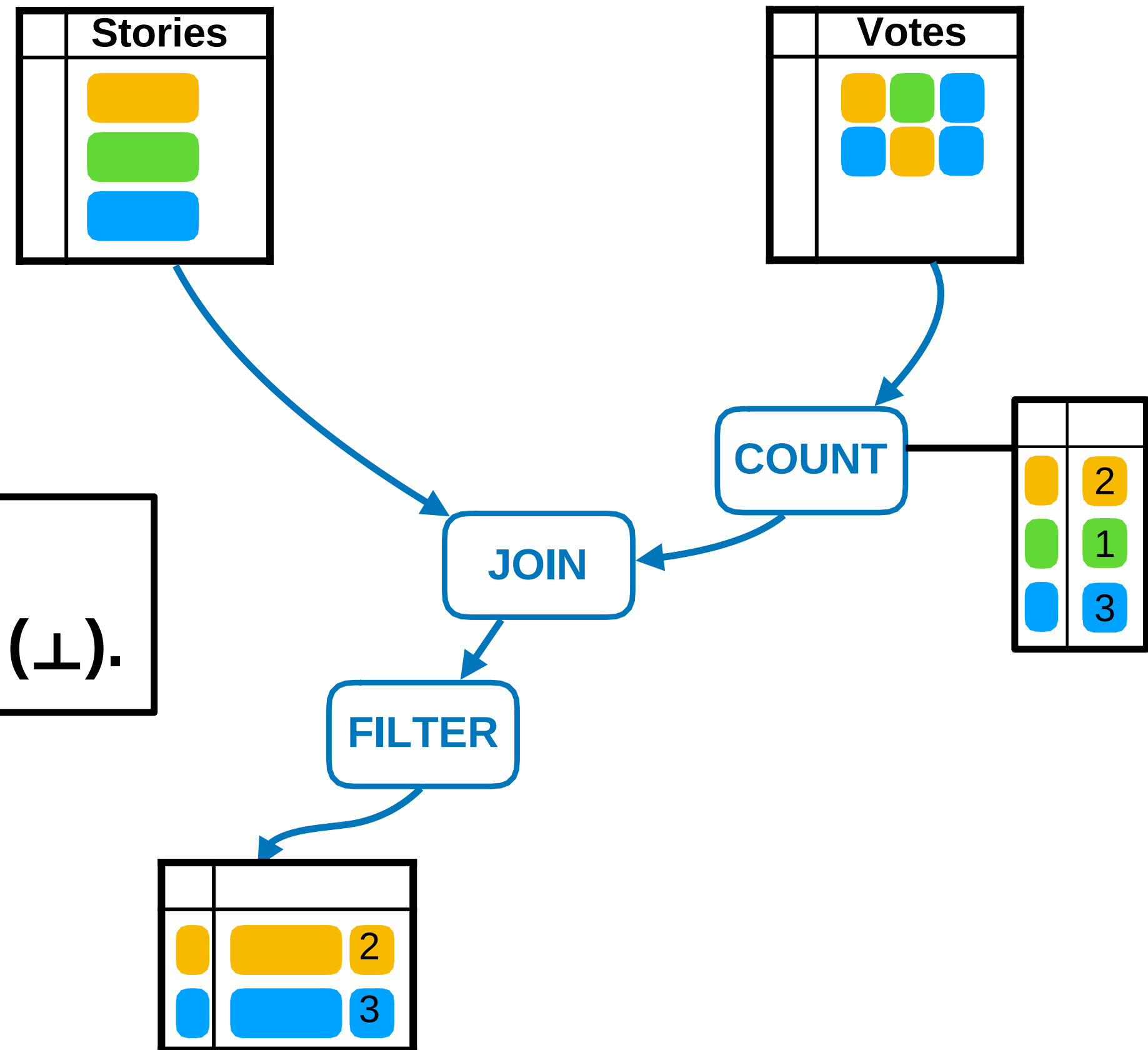
Evict cache state if needed: Limits memory requirements

No need to update evicted state: Reduces cost of writes

Adapts dataflow dynamically: Simplifies adding and removing queries that need caching

Partially-stateful data-flow

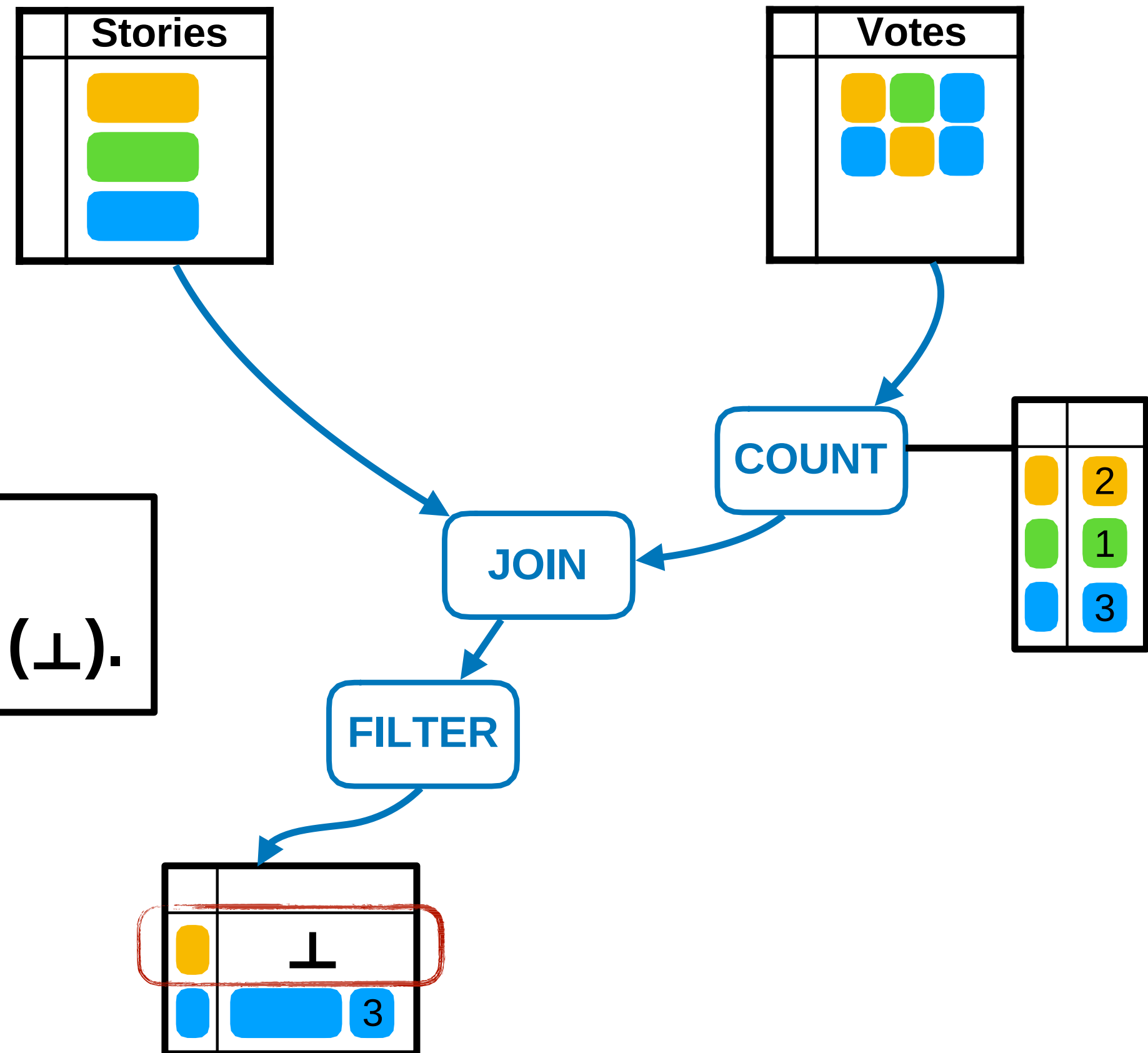
Data-flow state is *partial*:
entries for some keys are absent ($\perp$).



Frontend

Partially-stateful data-flow

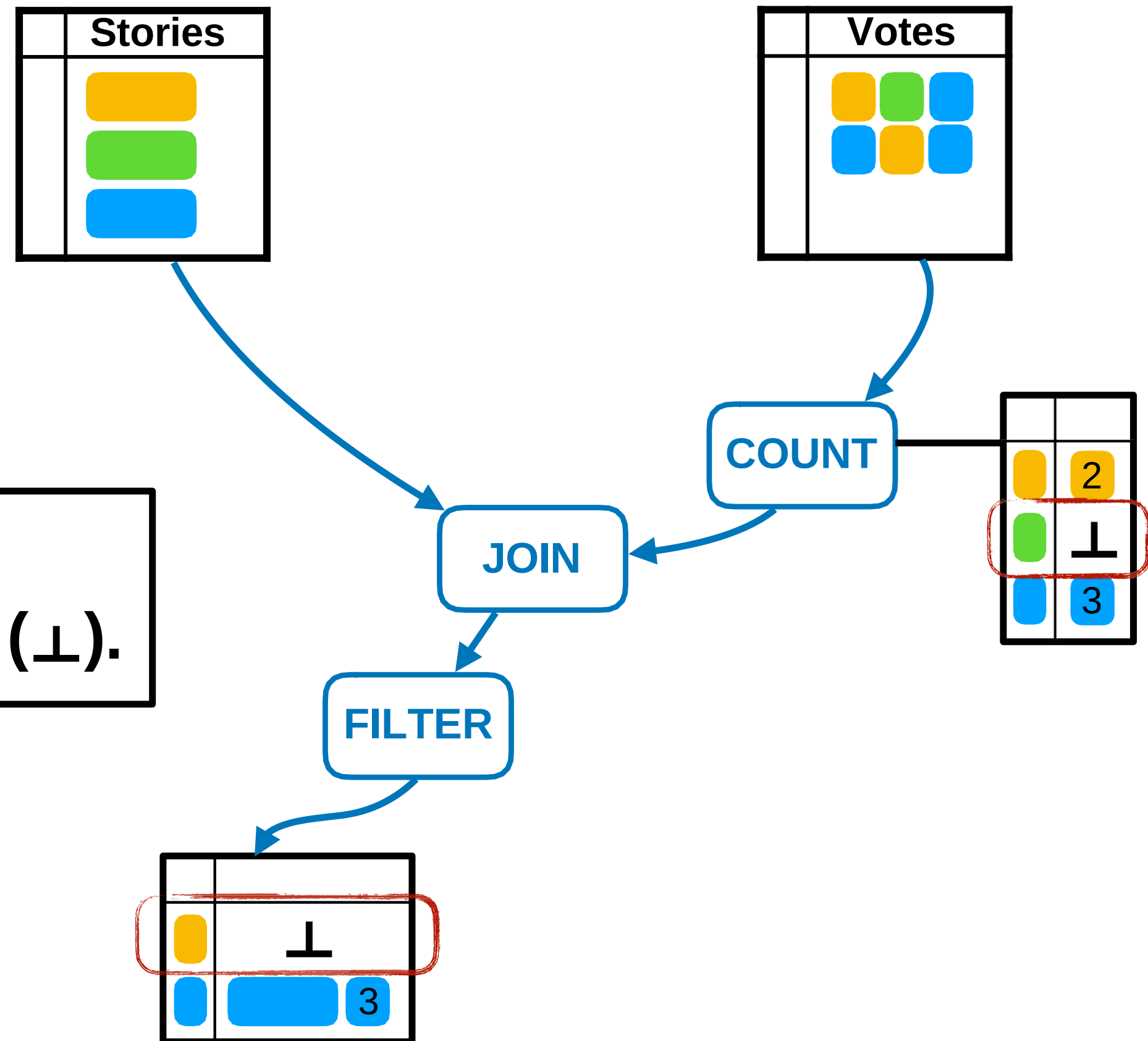
Data-flow state is *partial*:
entries for some keys are absent ($\perp$).



Frontend

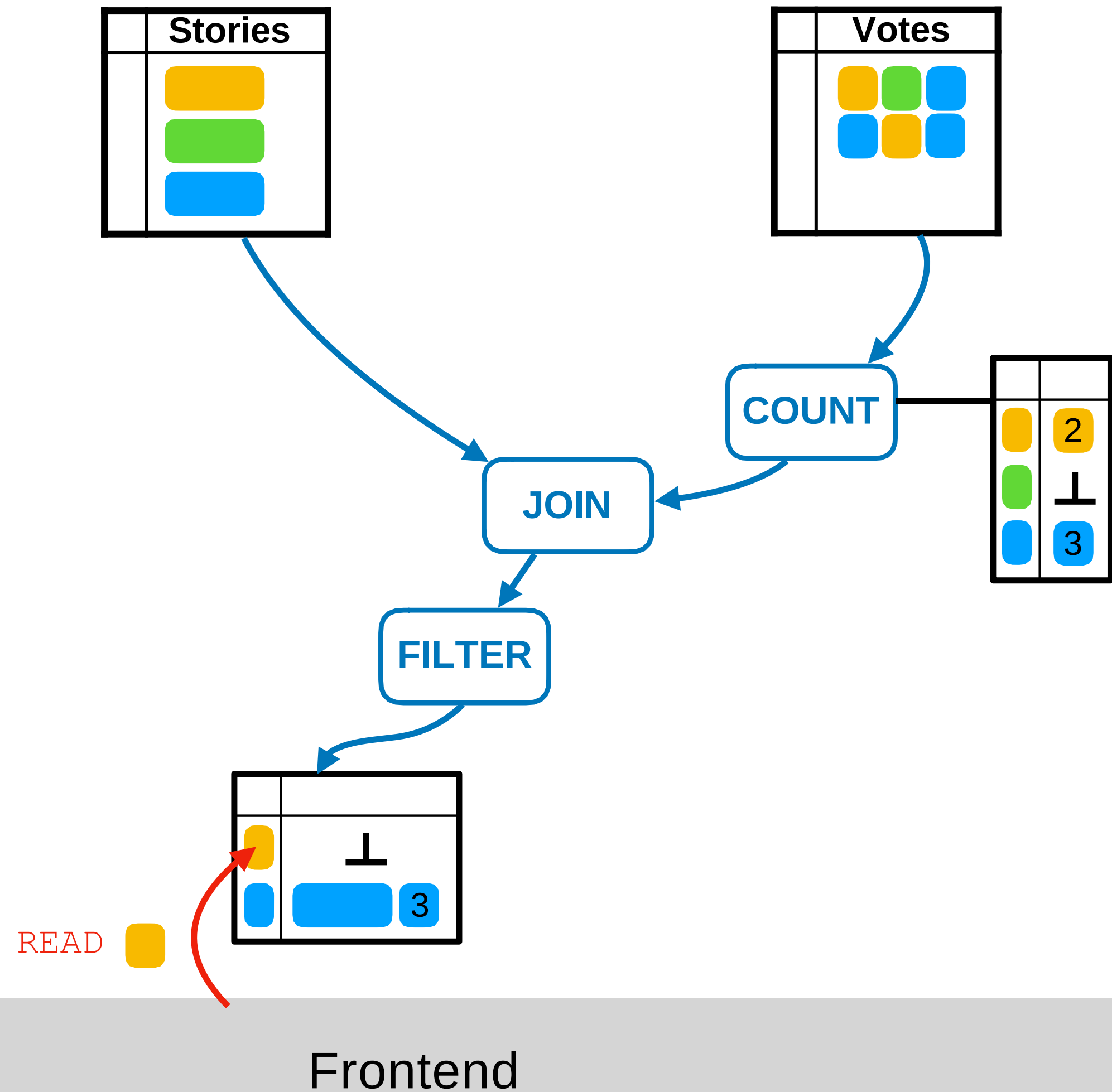
Partially-stateful data-flow

Data-flow state is *partial*:
entries for some keys are absent ($\perp$).

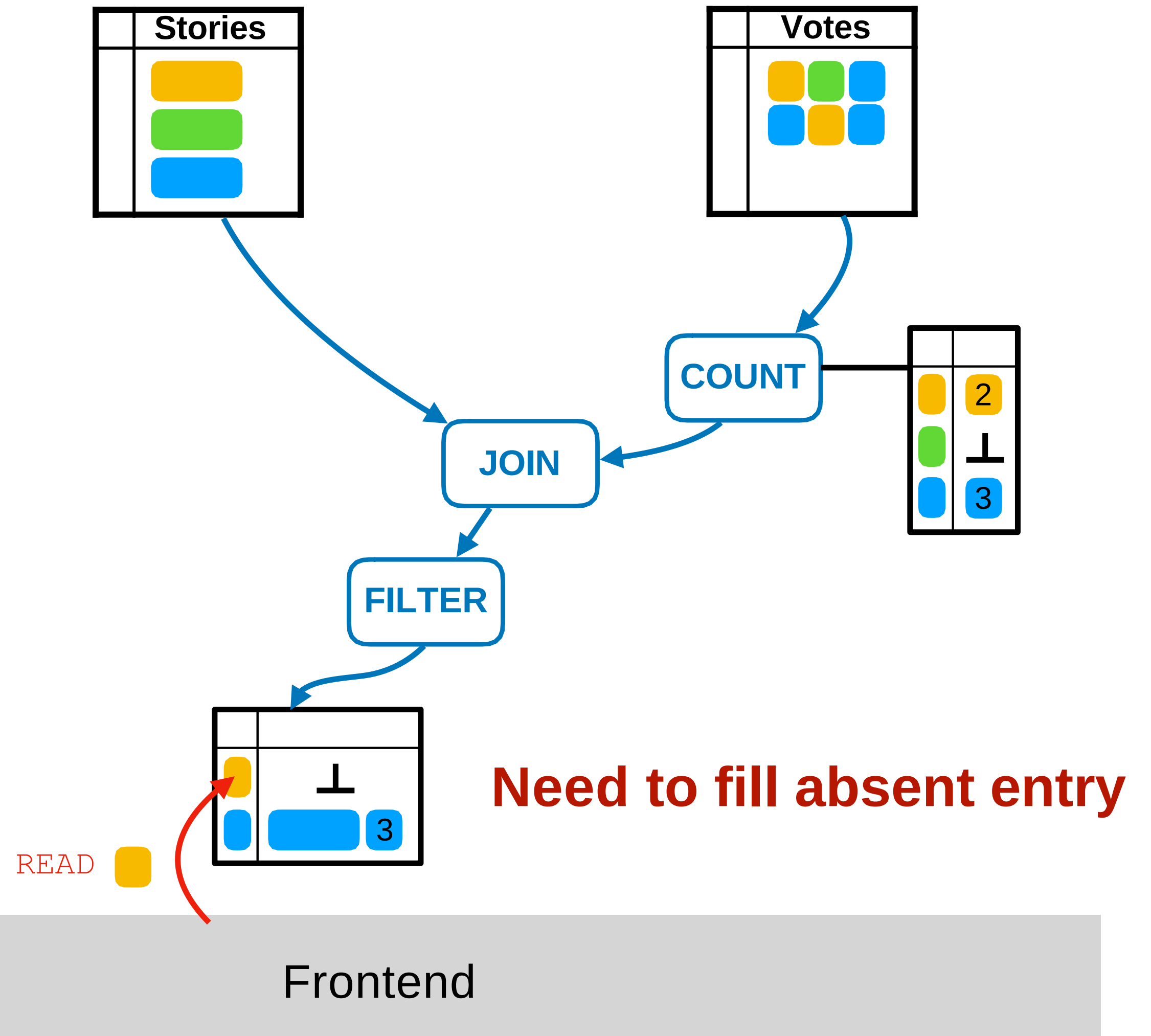


Frontend

Partially-stateful data-flow: upqueries



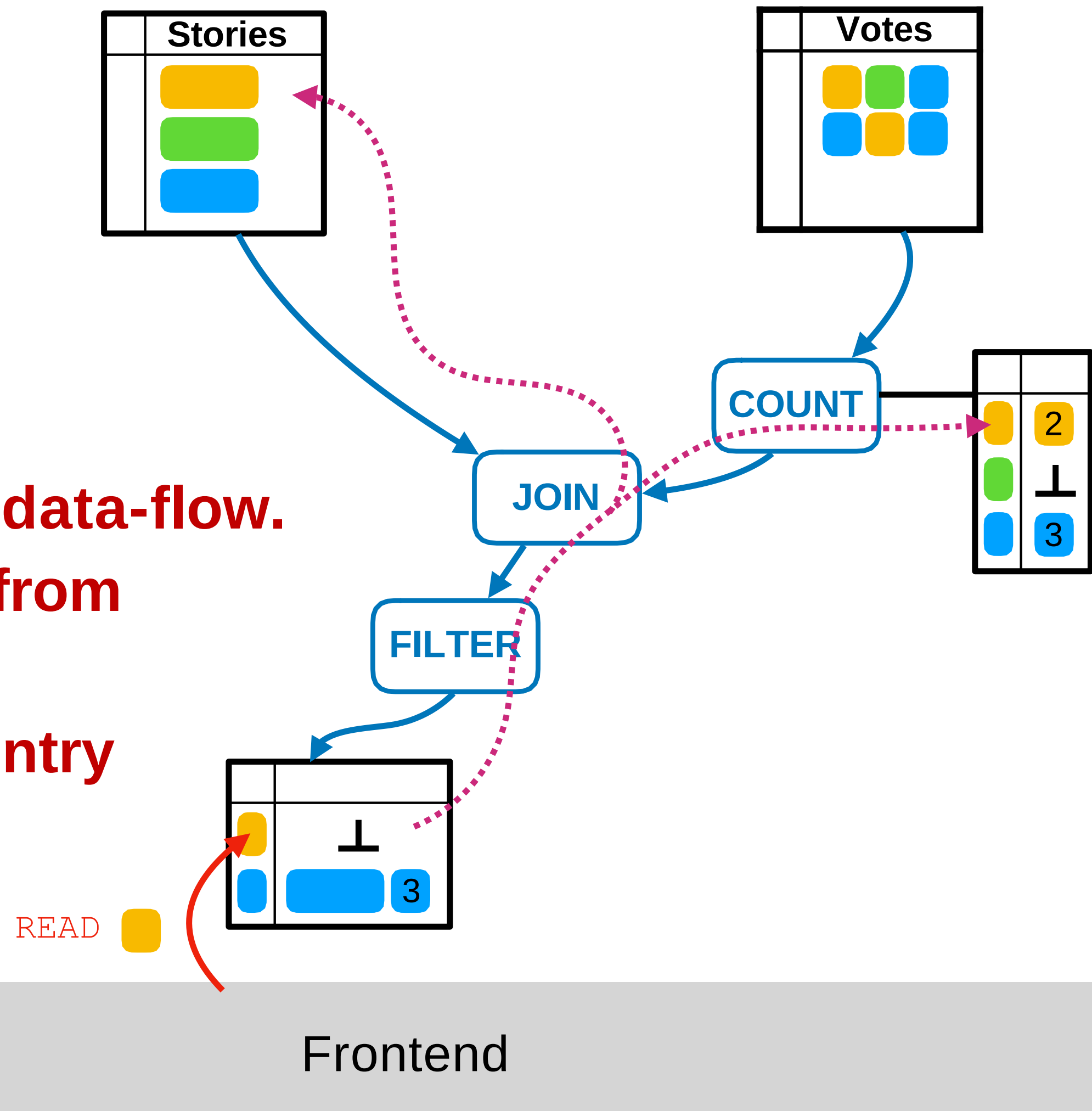
Partially-stateful data-flow: upqueries



Partially-stateful data-flow: upqueries

Solution: *upquery* through data-flow.

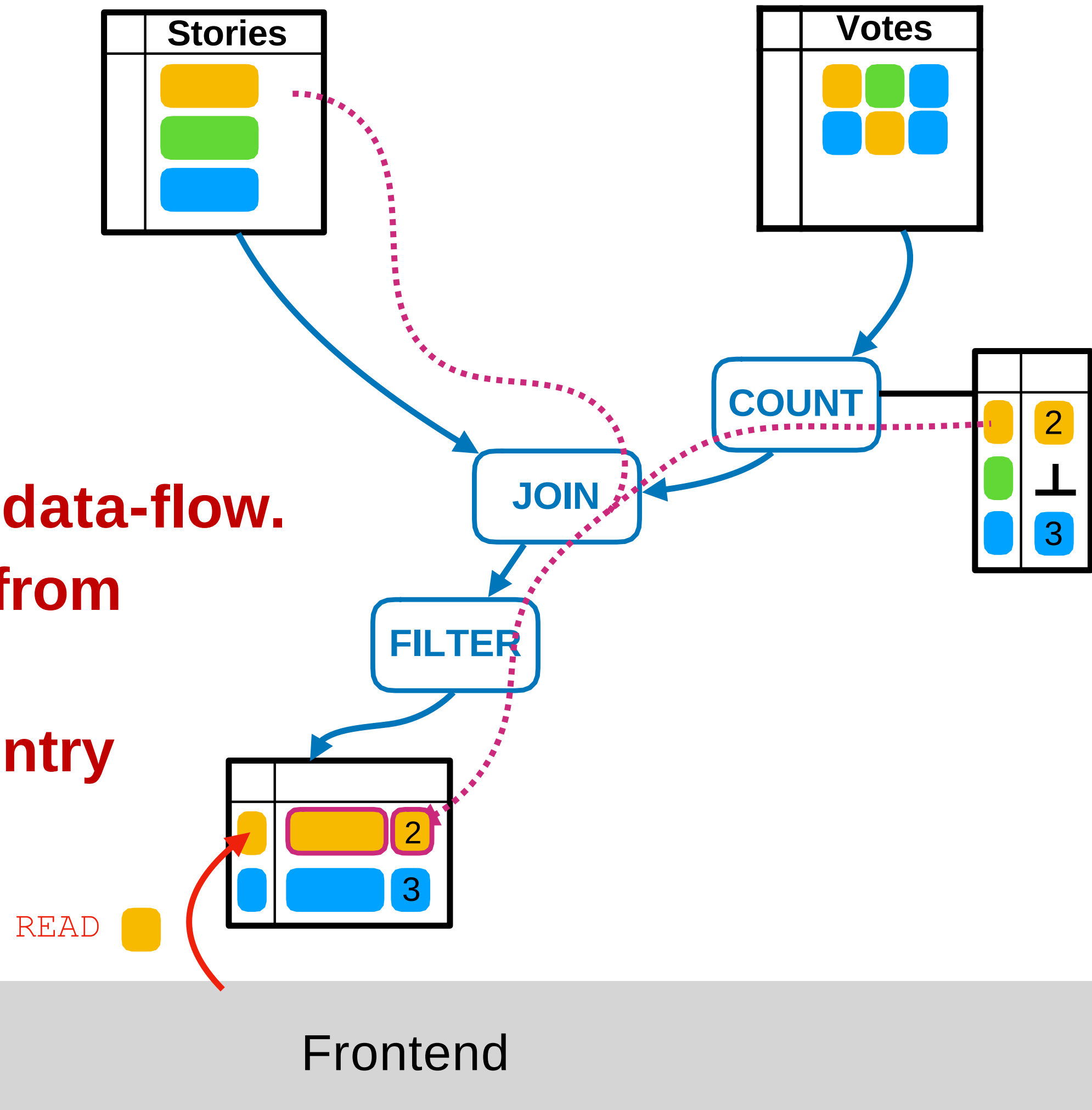
- Compute missing entry from upstream state
- Response fills missing entry



Partially-stateful data-flow: upqueries

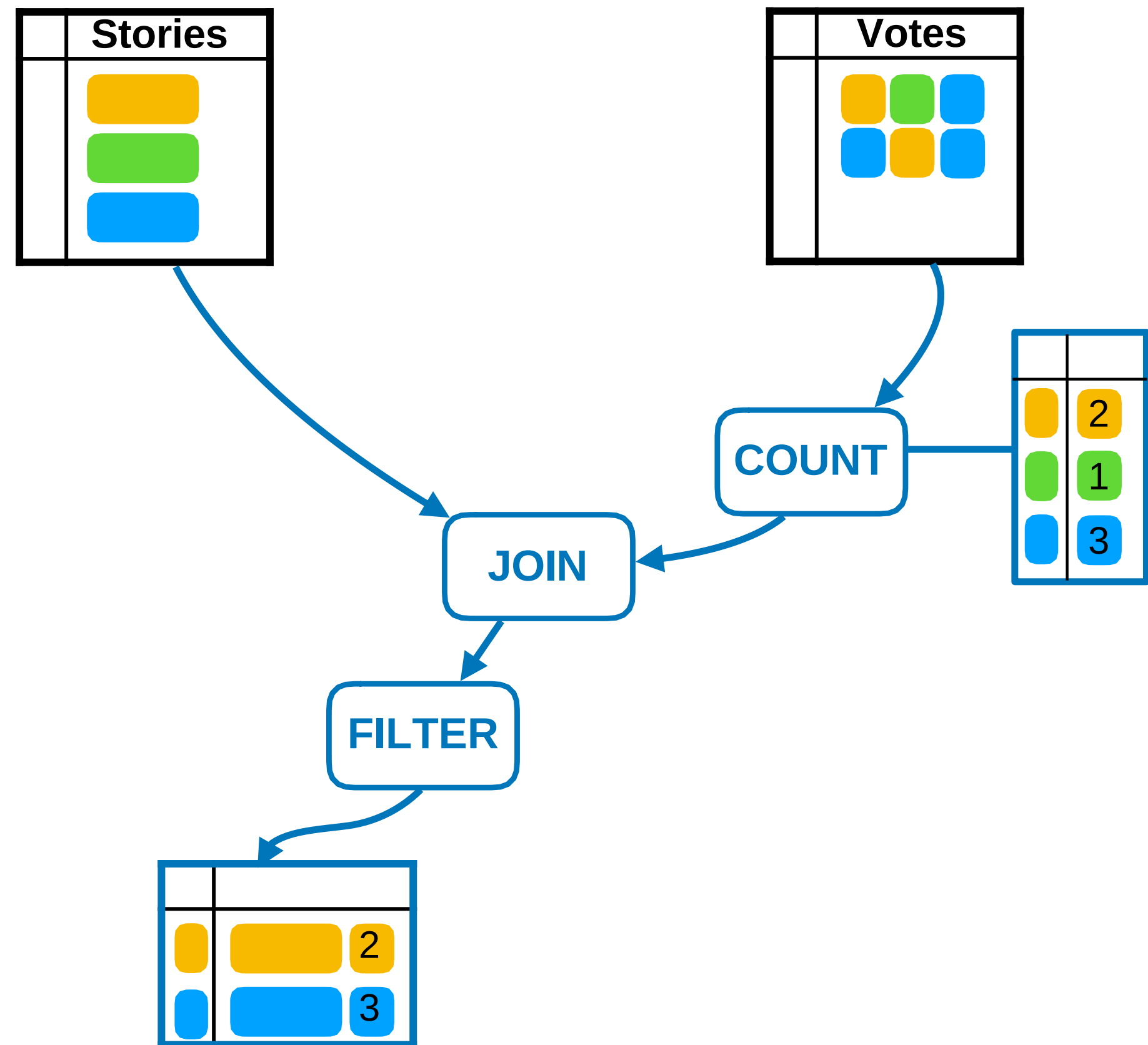
Solution: *upquery* through data-flow.

- Compute missing entry from upstream state
- Response fills missing entry



Partial state enables live data-flow changes

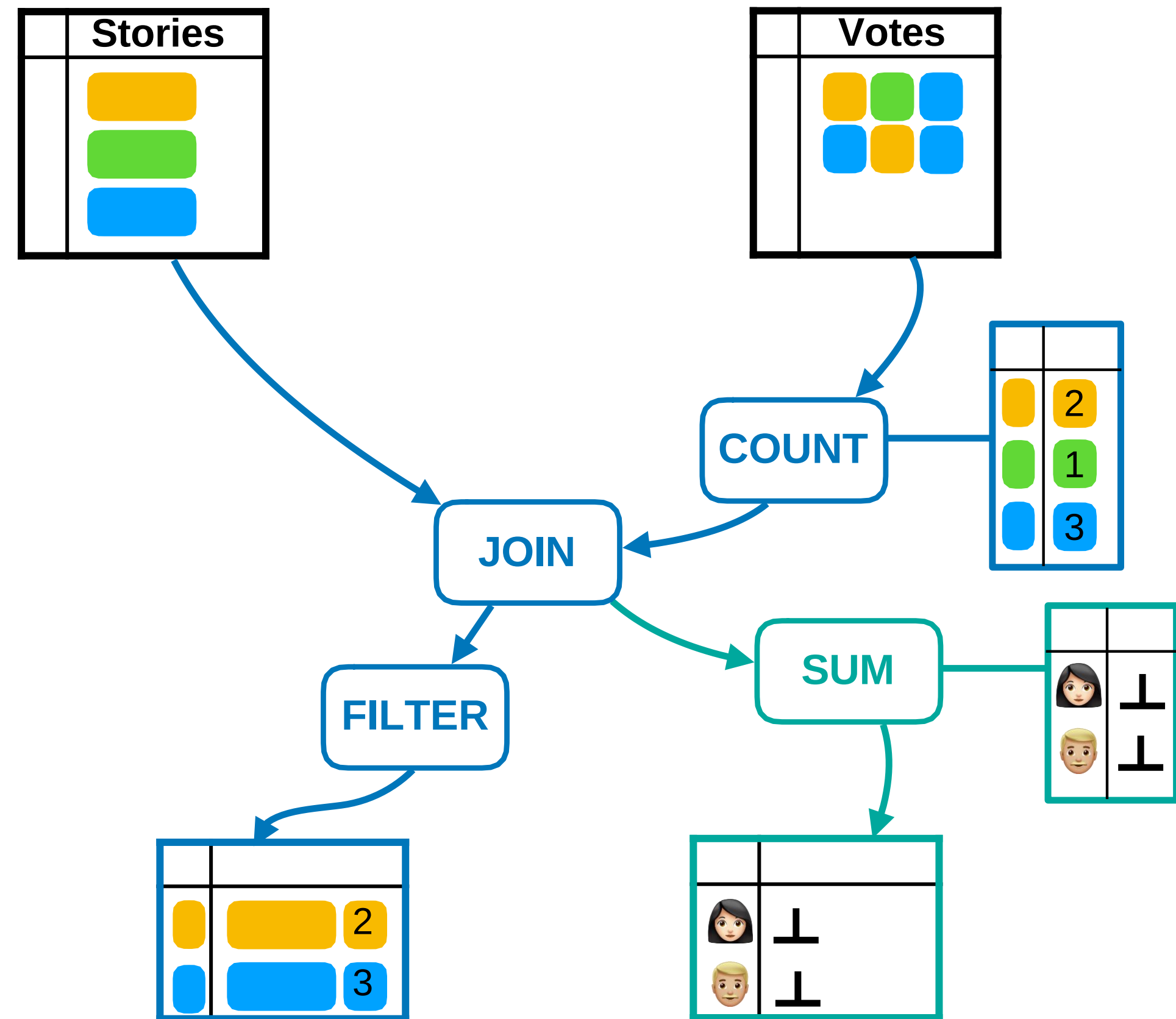
Start new views and operator
state **empty**, fill via **upqueries**.



Frontend

Partial state enables live data-flow changes

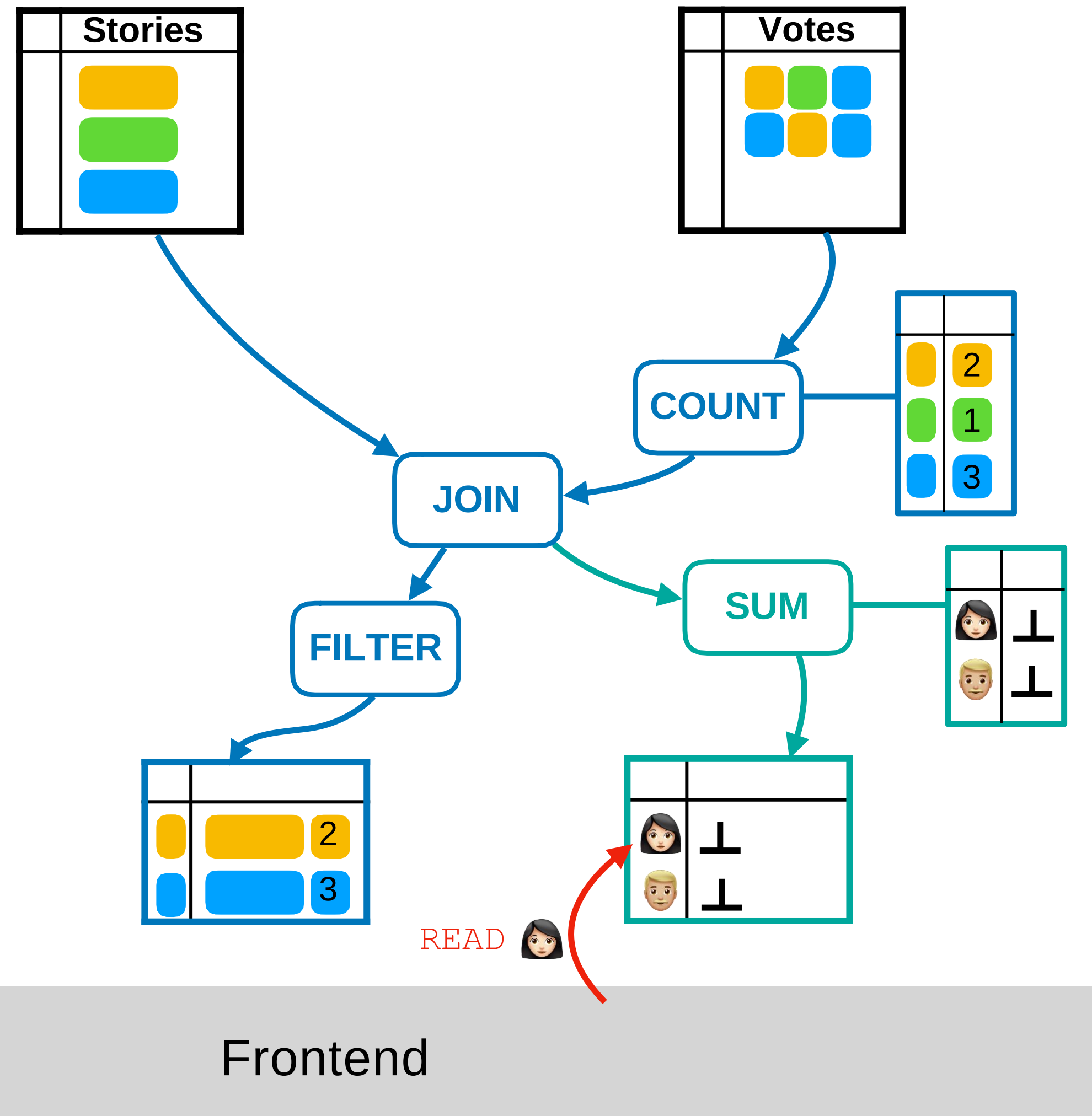
Start new views and operator
state **empty**, fill via **upqueries**.



Frontend

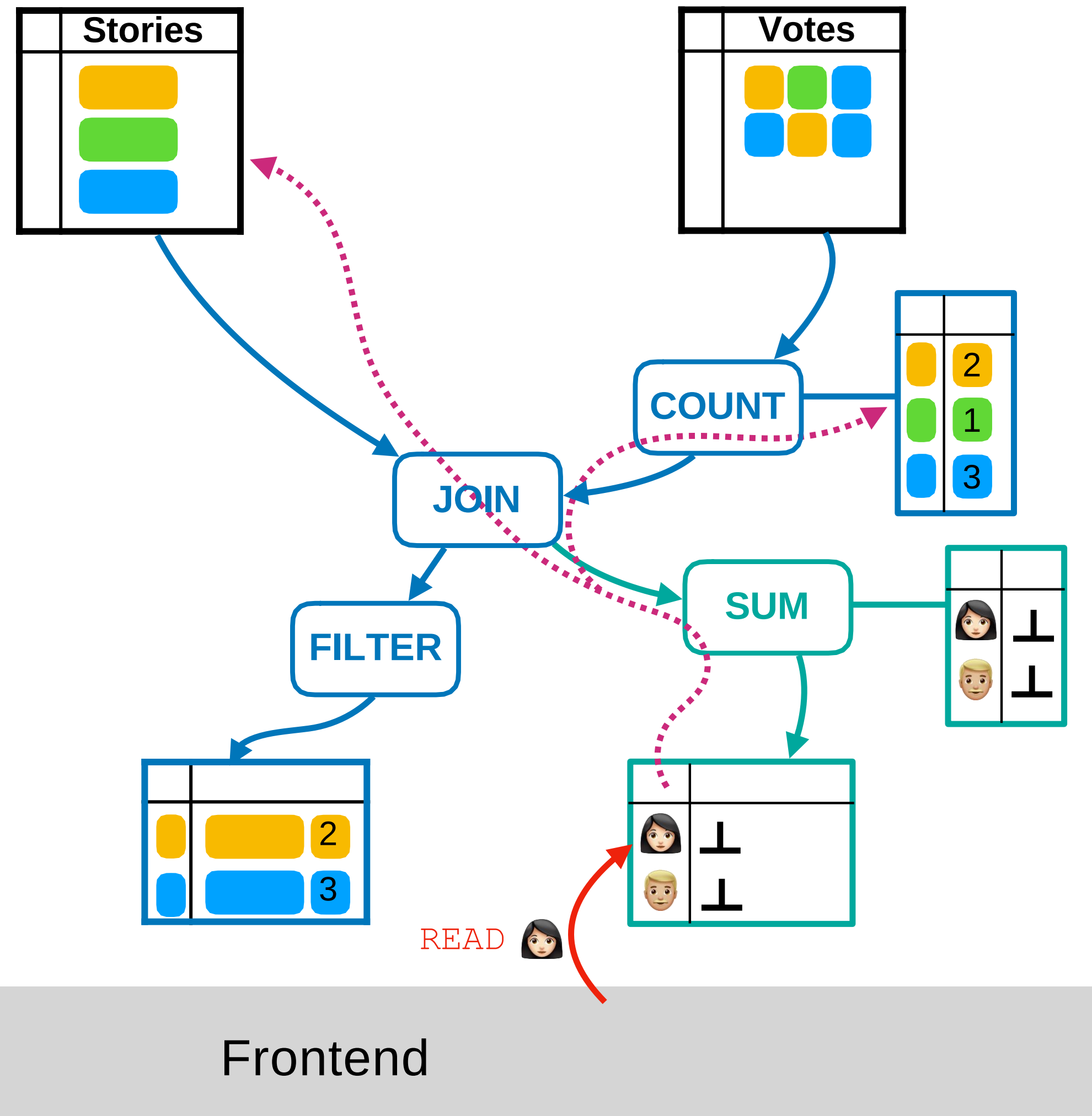
Partial state enables live data-flow changes

Start new views and operator
state **empty**, fill via **upqueries**.



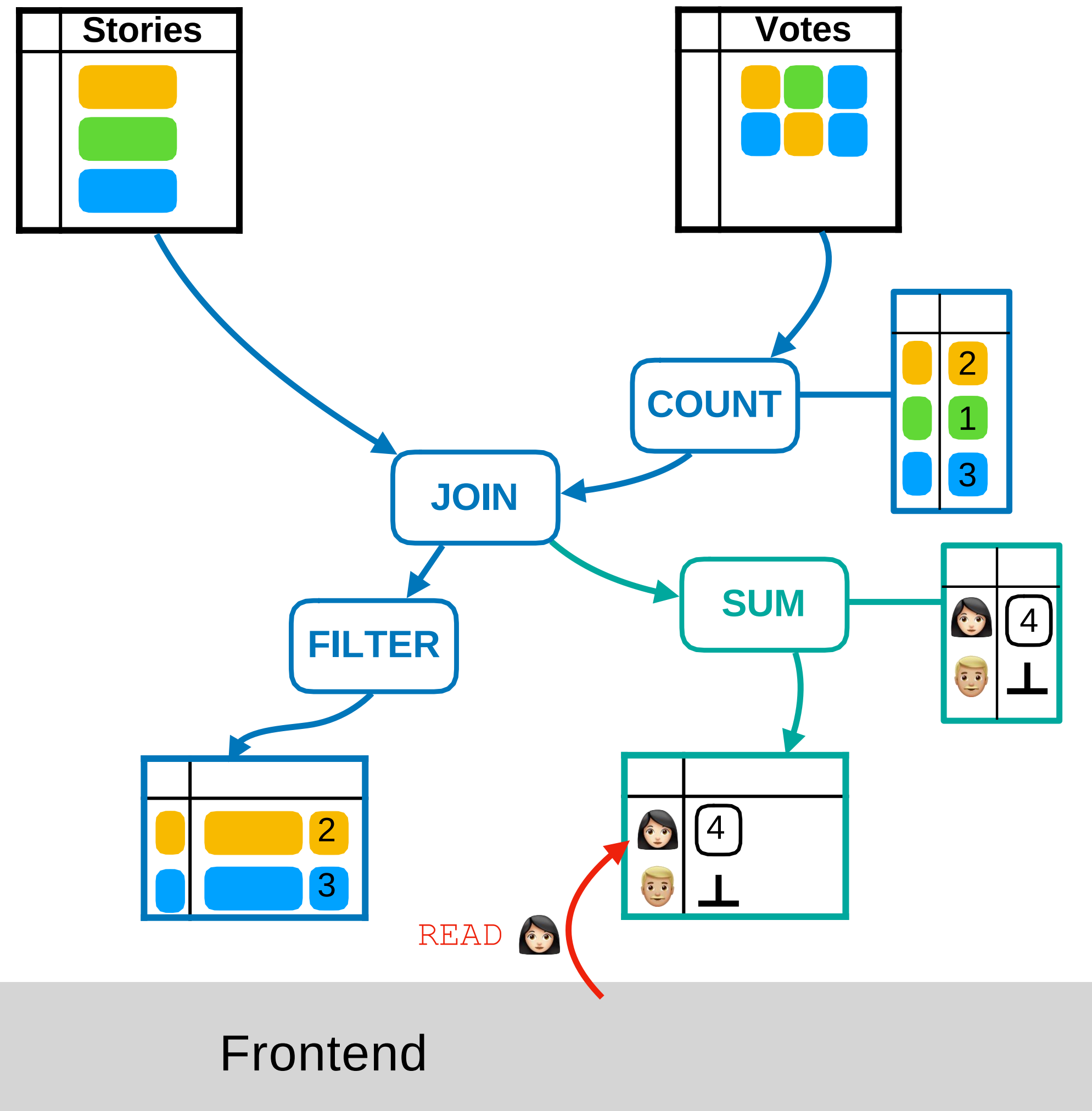
Partial state enables live data-flow changes

Start new views and operator
state **empty**, fill via **upqueries**.

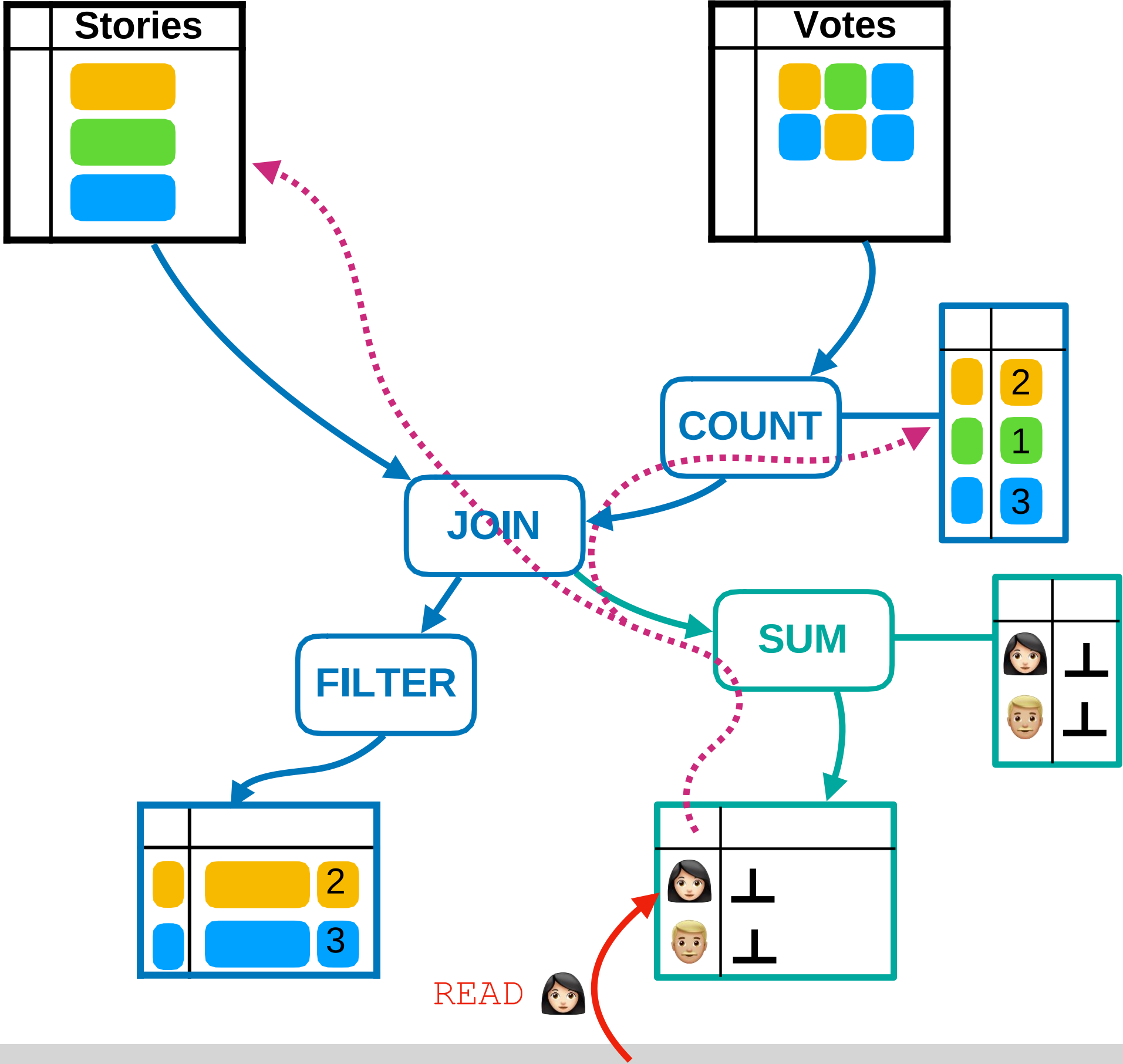
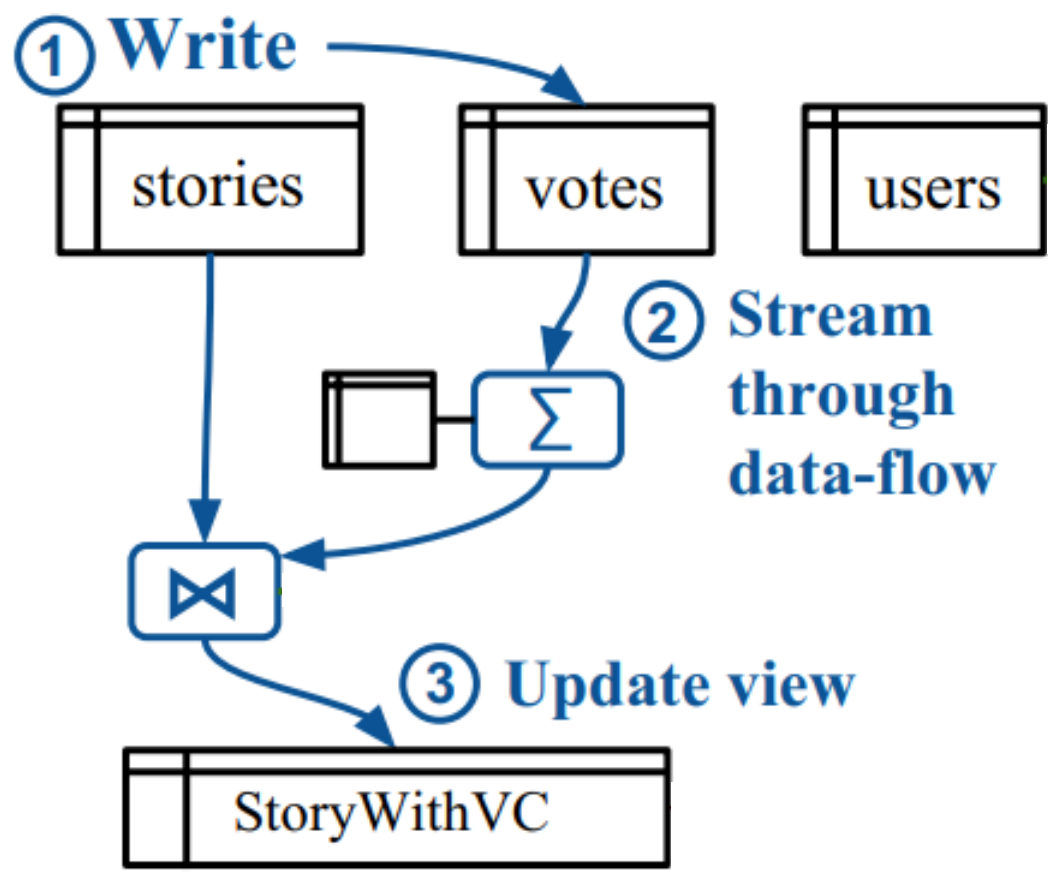


Partial state enables live data-flow changes

Start new views and operator
state **empty**, fill via **upqueries**.



Updates and upqueries



Challenges implementing partially-stateful data-flow

1. Concurrent upqueries and update processing — races!

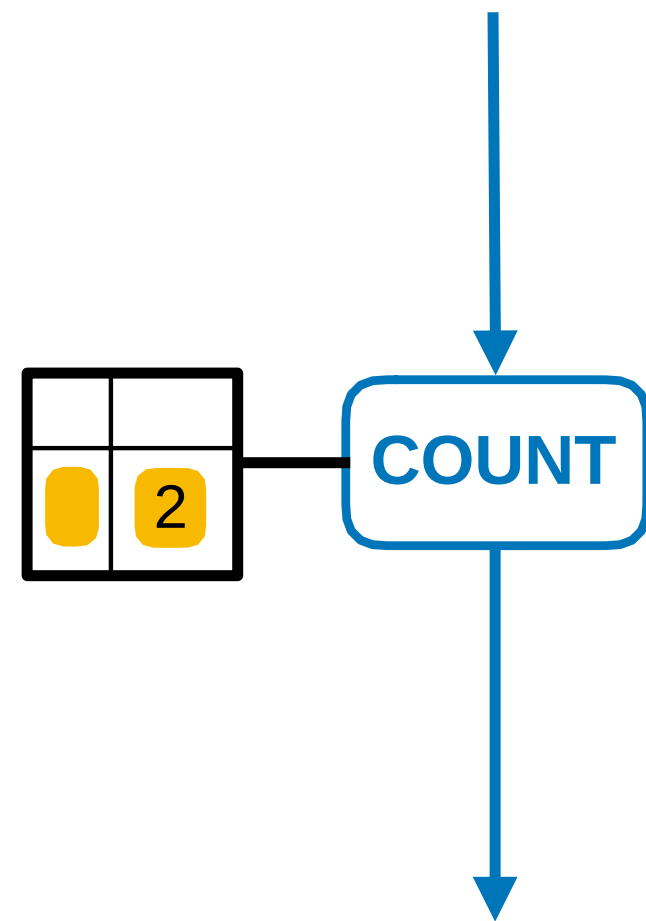
Must maintain **correctness** under concurrency!

Correctness under concurrency

Goal: upquery restores state as if present all along.

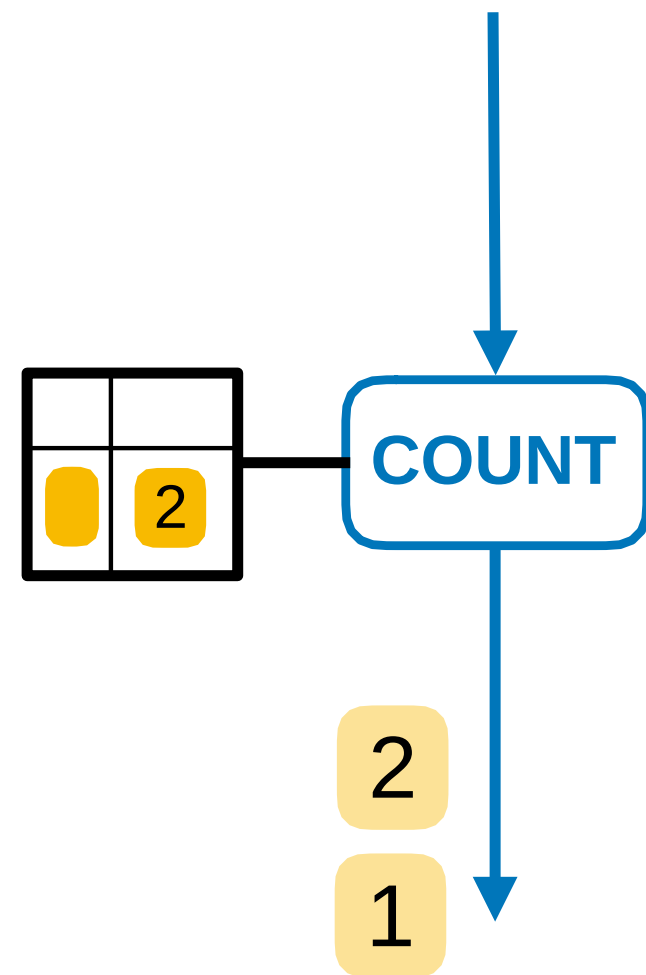
Correctness under concurrency

Goal: upquery restores state as if present all along.



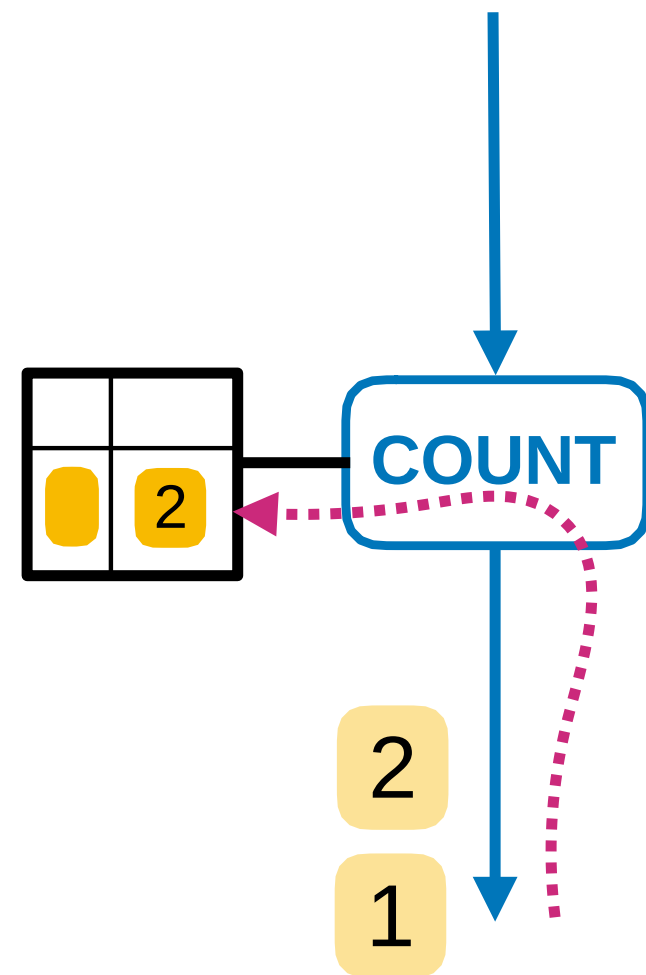
Correctness under concurrency

Goal: upquery restores state as if present all along.



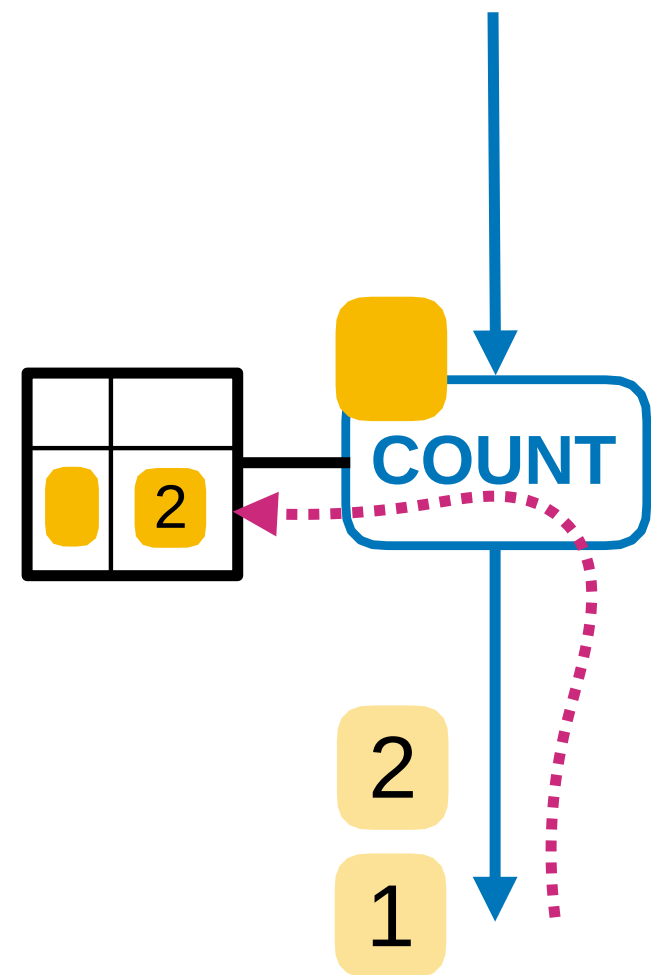
Correctness under concurrency

Goal: upquery restores state as if present all along.



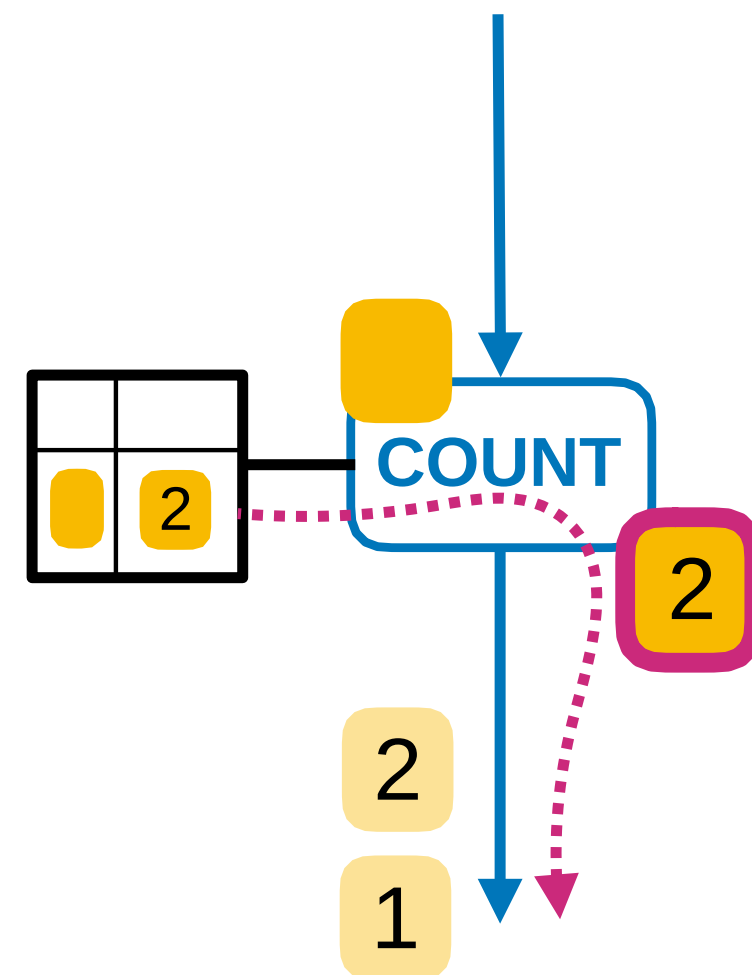
Correctness under concurrency

Goal: upquery restores state as if present all along.



Correctness under concurrency

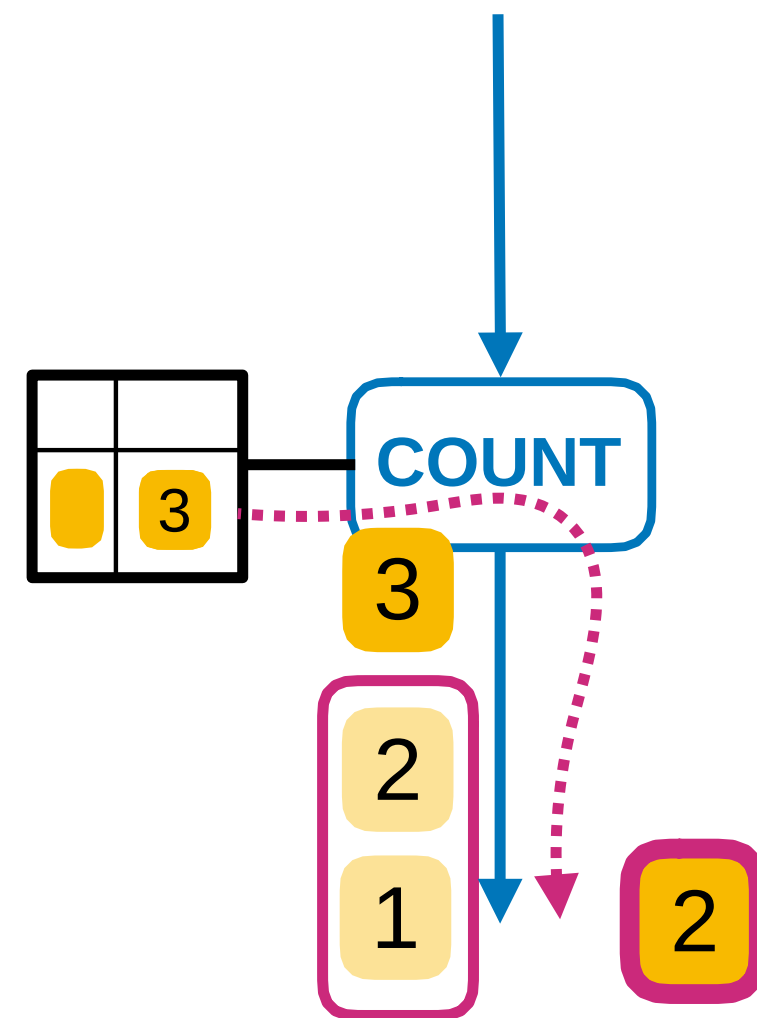
Goal: upquery restores state as if present all along.



Upquery response is a **snapshot** of state

Correctness under concurrency

Goal: upquery restores state as if present all along.



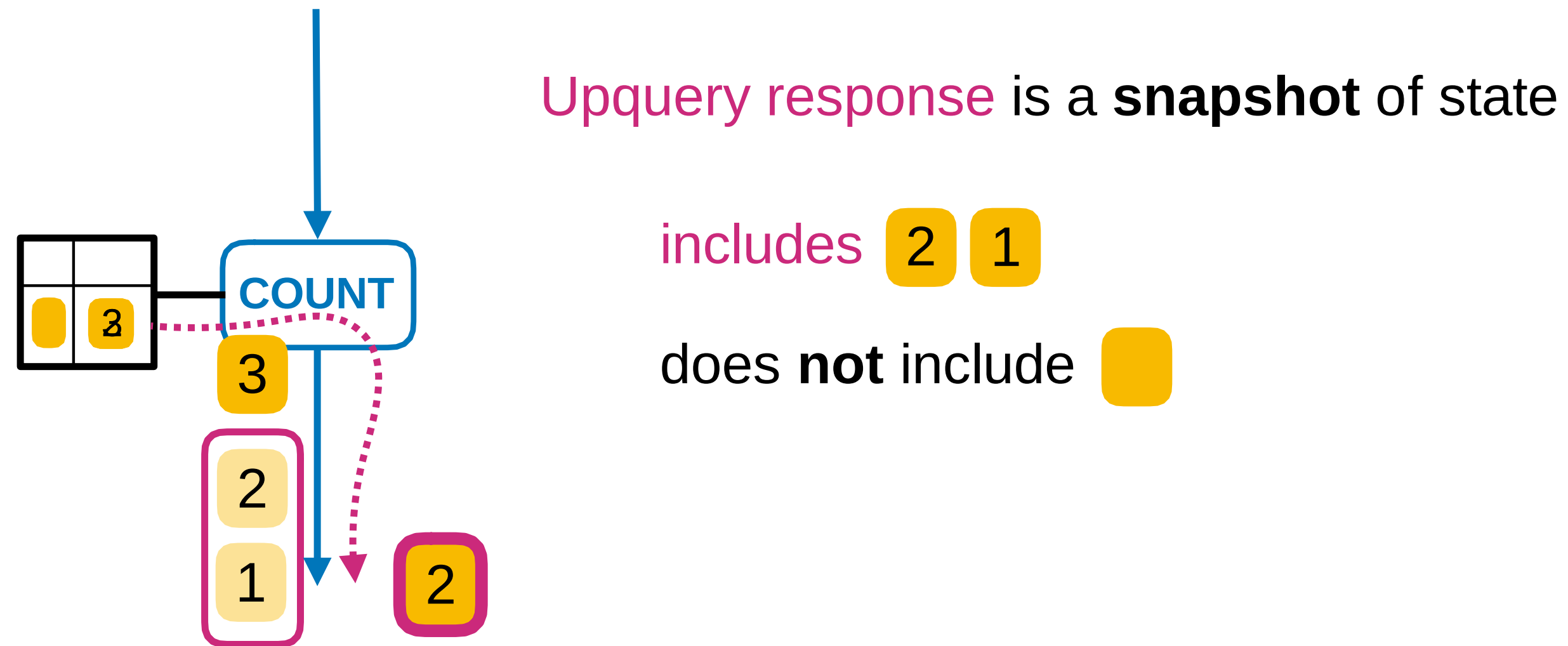
Upquery response is a **snapshot** of state

includes 2 1

does **not** include

Correctness under concurrency

Goal: upquery restores state as if present all along.



Solution: Maintain **order** of upquery response and surrounding updates, despite lack of global coordination.

Challenges implementing partially-stateful data-flow

1. Concurrent upqueries and forward processing — races!

Must maintain **correctness** under concurrency!

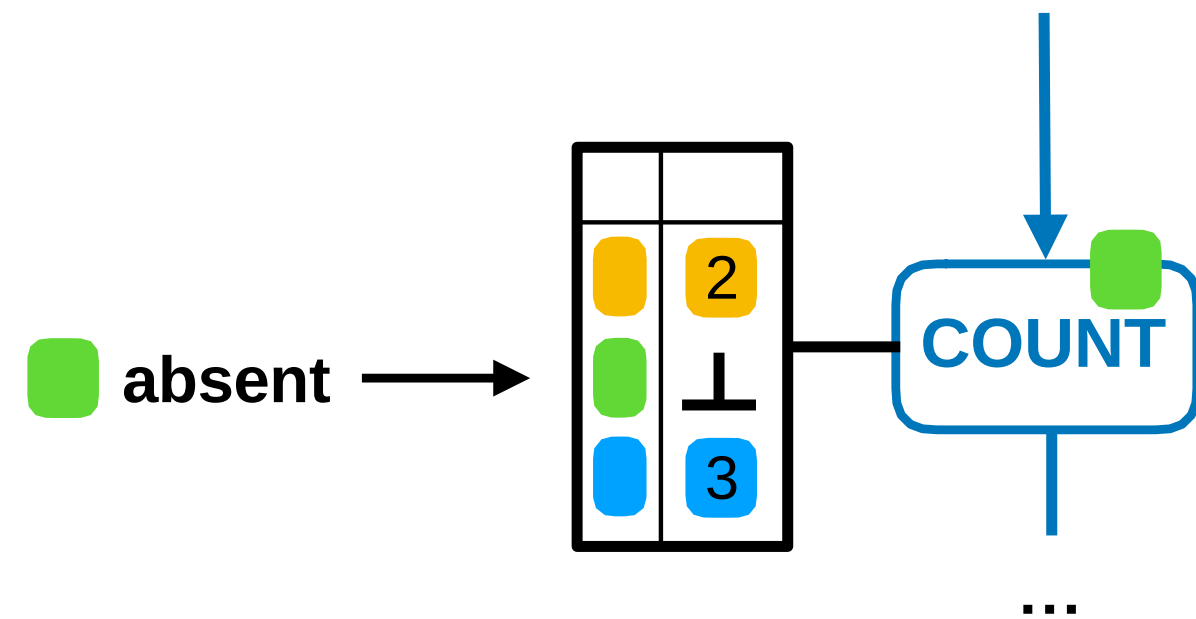
2. Update processing may require absent state

Challenges implementing partially-stateful data-flow

1. Concurrent upqueries and forward processing — races!

Must maintain **correctness** under concurrency!

2. Update processing may require absent state

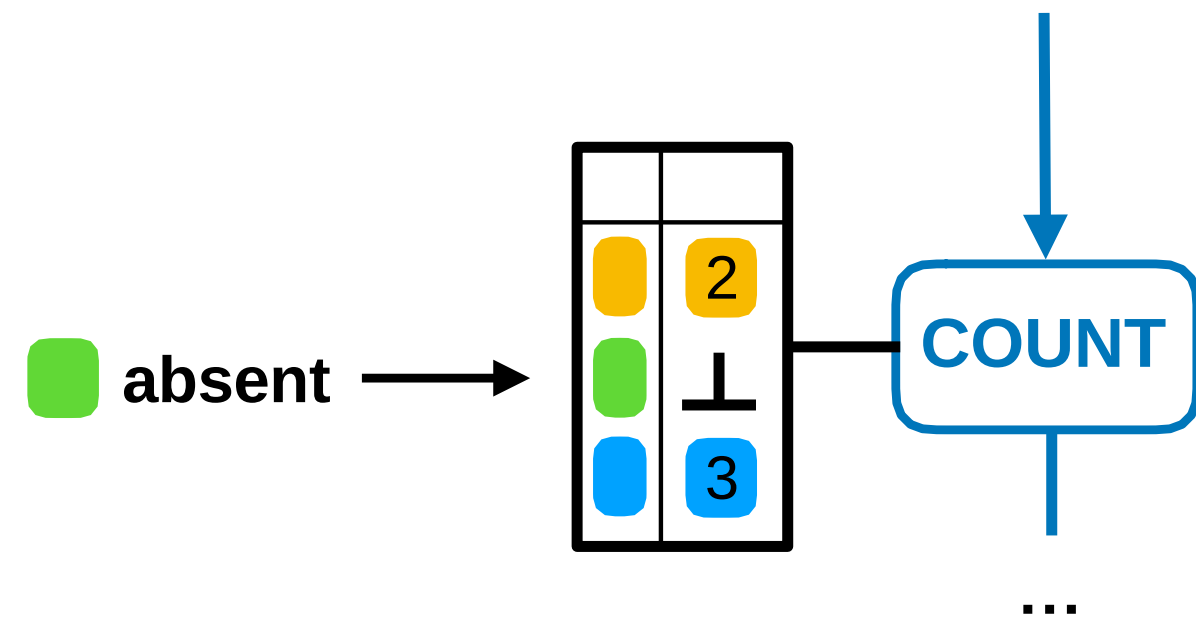


Challenges implementing partially-stateful data-flow

1. Concurrent upqueries and forward processing — races!

Must maintain **correctness** under concurrency!

2. Update processing may require absent state



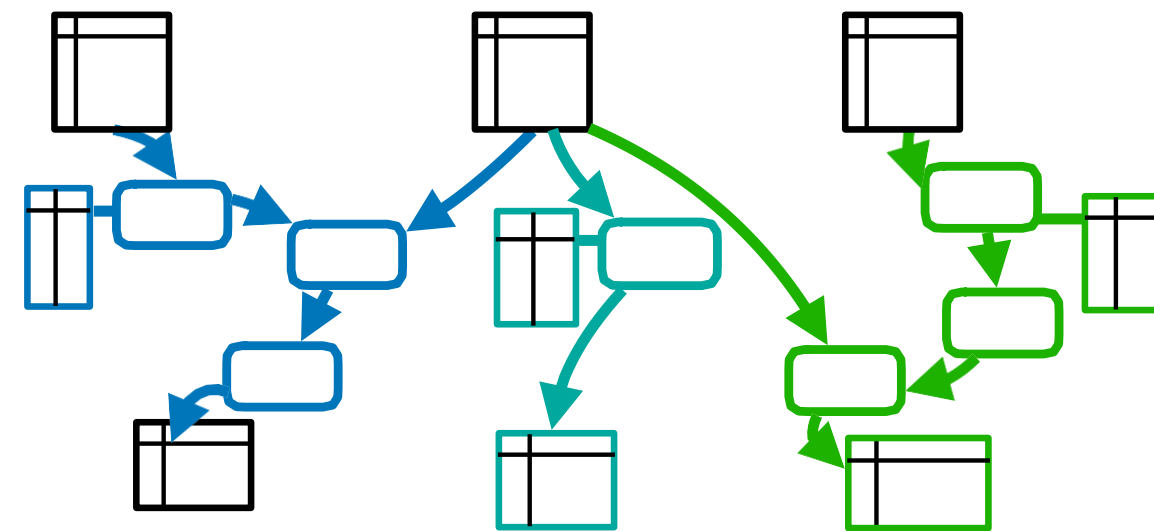
Drop updates that touch absent state, future upquery repeats them.

Noria implementation

```
1  /* base tables */
2  CREATE TABLE stories
3    (id int, author int, title text, url text);
4  CREATE TABLE votes (user int, story_id int);
5  CREATE TABLE users (id int, username text);
6  /* internal view: vote count per story */
7  CREATE INTERNAL VIEW VoteCount AS
8    SELECT story_id, COUNT(*) AS vcount
9      FROM votes GROUP BY story_id;
10 /* external view: story details */
11 CREATE VIEW StoriesWithVC AS
12   SELECT id, author, title, url, vcount
13     FROM stories
14     JOIN VoteCount ON VoteCount.story_id = stories.id
15   WHERE stories.id = ?;
```

Transform

Data-flow graph



MySQL adapter

- 45k lines of Rust + 15k libraries
- RocksDB for base table storage
- ZooKeeper for leader election

Evaluation

1. Can Noria improve a real web application's performance?

Case study: Lobsters (<http://lobste.rs>)

L Lobsters Recent Comments Search

Login

▲

Falling in love with Rust

programming rust dtrace.org

40

via blake 11 hours ago | cached | 7 comments

▲

FreeBSD Desktop - Part 16 - Configuration - Pause Any Application

freebsd illumos linux vermaden.wordpress.com

6

authored by vermaden 6 hours ago | cached | 4 comments

▲

You Think the Visual Studio Code binary you use is a Free Software? Think again

law privacy carlchenet.com

61

authored by chaica 31 hours ago | cached | 30 comments

▲

Using Make — writing less Makefile

programming text.causal.agency

27

authored by causal_agent 23 hours ago | cached | 11 comments

▲

LLVM 7.0.0 Release

release lists.llvm.org

4

via colin 43 minutes ago | cached | no comments

▲

Kit programming language

programming kitlang.org

27

via btbytes 25 hours ago | cached | 21 comments

▲

Why Aren't More Users More Happy With Our VMs? Part 2

performance tratt.net

5

via edd 4 hours ago | cached | no comments

▲

Spleen - Monospaced bitmap fonts

design cambus.net

1

authored by fcambus 3 minutes ago | cached | no comments

▲

The Singleton module in Ruby - Part I

ruby medium.com

1

authored by mehdi-farsi 14 minutes ago | cached | no comments

▲

You Can't Always Hash Pointers in C

c nullprogram.com

3

via calvin 4 hours ago | cached | 1 comment

▲

Learn Go with Seam Carving and Rockets

go getstream.io

1

authored by tschellenbach 18 minutes ago | cached | no comments

▲

Times Newer Roman is a sneaky font designed to make your essays look longer

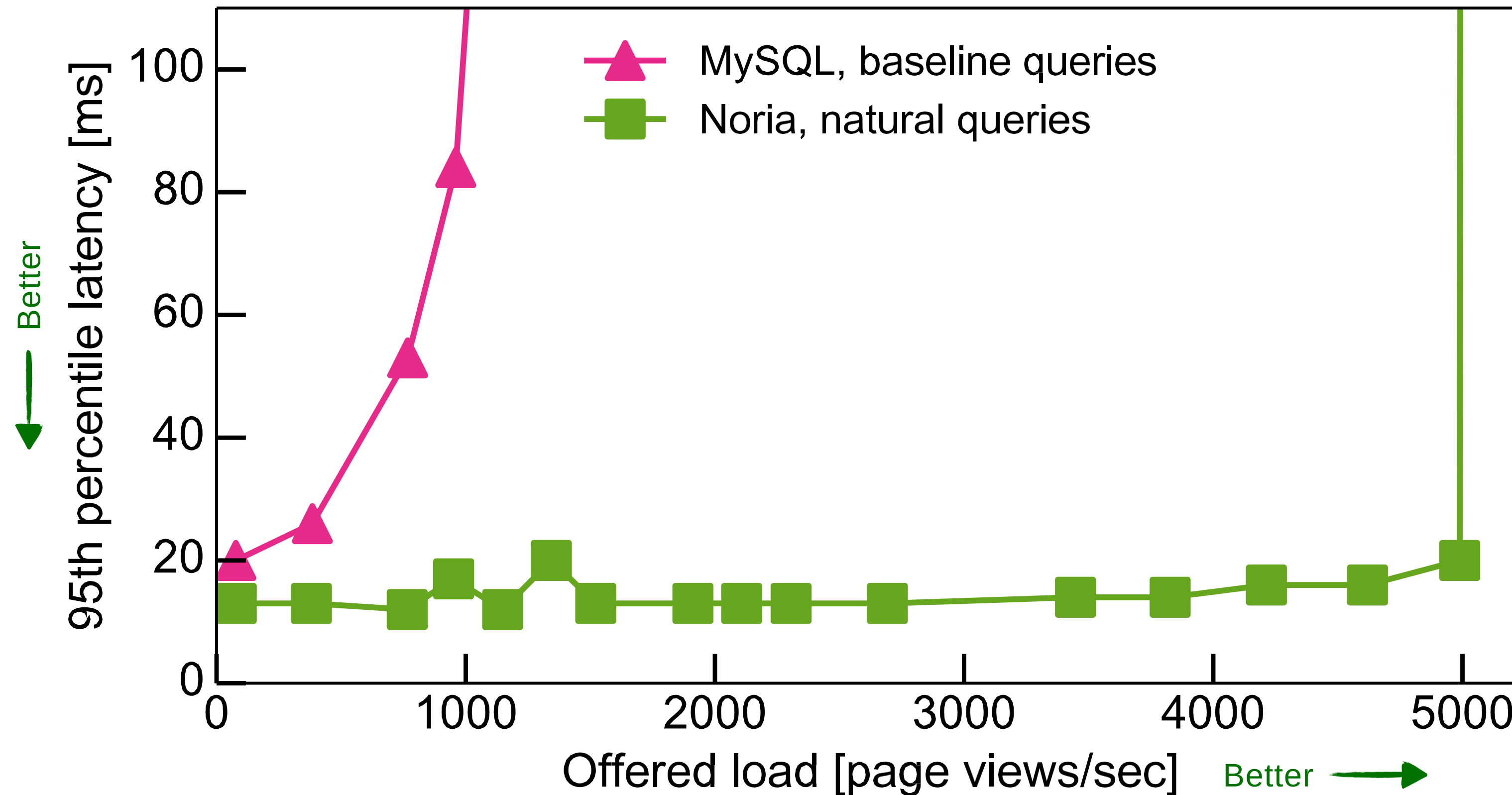
education graphics theverge.com

9

via Ricardus 15 hours ago | cached | 5 comments

- ▶ Ruby-on-Rails application with MySQL backend
- ▶ Hand-optimized by developers to pre-compute aggregations
- ▶ Noria data-flow with 235 operators, 35 views
- ▶ Emulate production load

Can Noria improve Lobsters' performance?



Noria with **natural queries** supports **5x** MySQL's throughput.

Noria — Summary

- New partially-stateful data-flow model
- Noria: new web application backend based on data-flow
- Partial state saves space and allows live change
- Supports high throughput on one or more machines
- Open source, try it out!

Q1

Noria ensures that clients read eventually consistent data.
What problem are they trying to solve? Why is eventual consistency okay?

Q2

How does Noria ensure eventual consistency?

Consider the following example: say there are two updates u_1 , u_2 that update two views, and a upquery uq issued by the user also updates these views.

base table:

view1 (upstream) op1:

view2 (downstream) op2:

Q3

What complicates Join processing in Noria?

Consider the following example:

