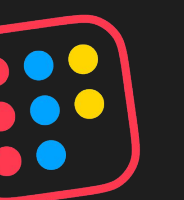


How Do We Sleep at Night?

Building Reliable Distributed Systems at Startup Speed

SRDS 44th International Symposium on Reliable Distributed Systems
01.10.25



Ciao, I'm Jacopo!

| Co-founder and CTO at **Bauplan**.

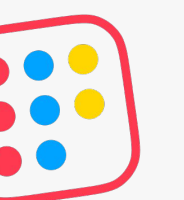
Backed by IE, SPC, Wes McKinney, Spencer Kimball, Chris Re et al.

| Started the “Reasonable Scale” movement.

Co-founder at Tooso and lead AI at TSX:CVO after the acquisition.

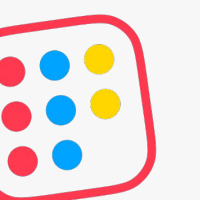
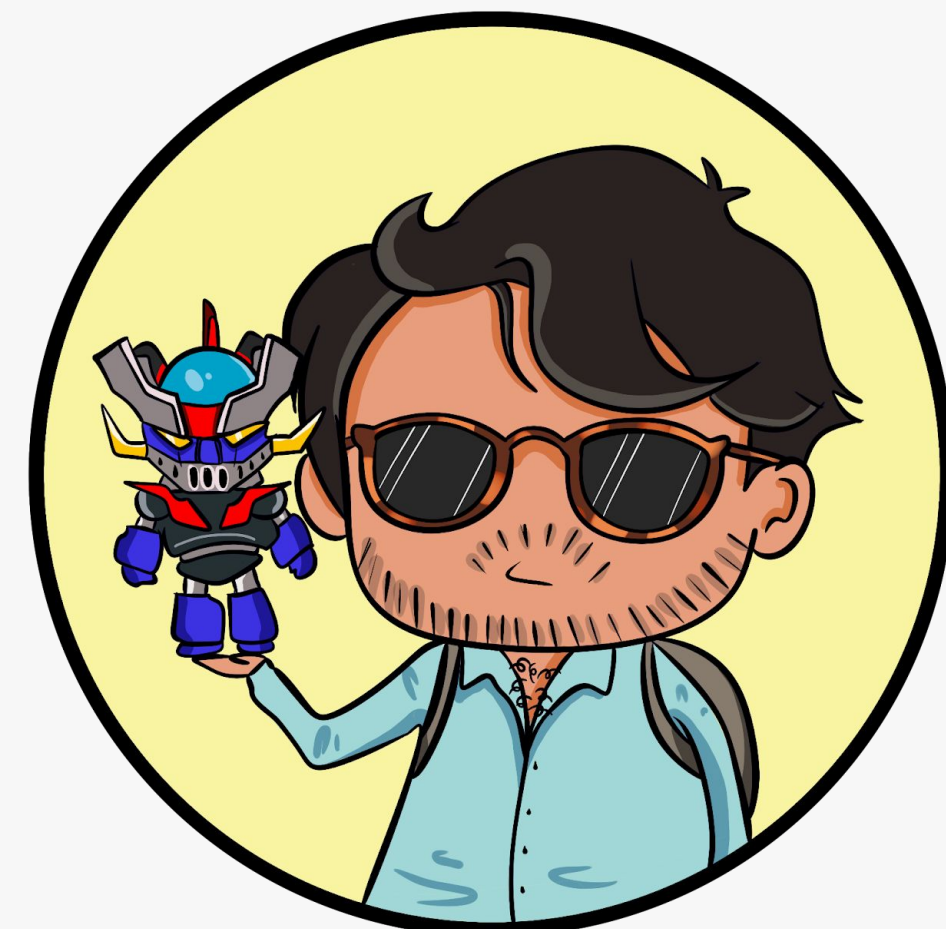
| 10 years up and down the stack in R&D, product, open source

ICML, KDD, VLDB, NAACL, SIGIR, WWW et al., >2k stars, >50M+ downloads.



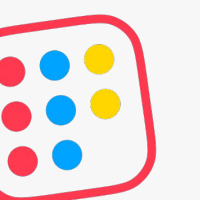
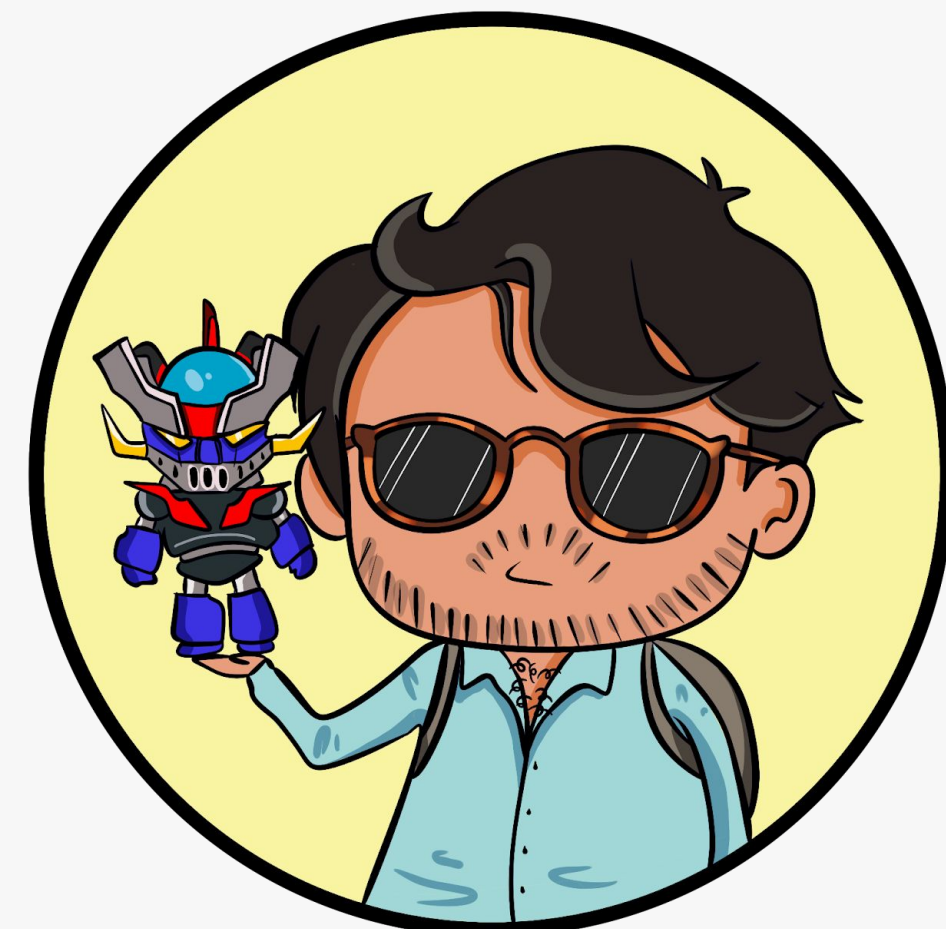
It takes a (distributed) village

| Matt, Ciro, Luca, Nate, Vlad (and others, unfortunately without a chibi) share with me the credit for whatever value these ideas may have.



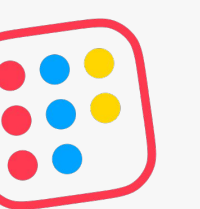
It takes a (distributed) village

- | Matt, Ciro, Luca, Nate, Vlad (and others, unfortunately without a chibi) share with me the credit for whatever value these ideas may have.
- | Obviously, all the remaining mistakes are theirs 😁



Bauplan is the easiest way to build reliable, fast cloud data pipelines.

- No infrastructure
- Just Python and SQL
- Like Git, but for data

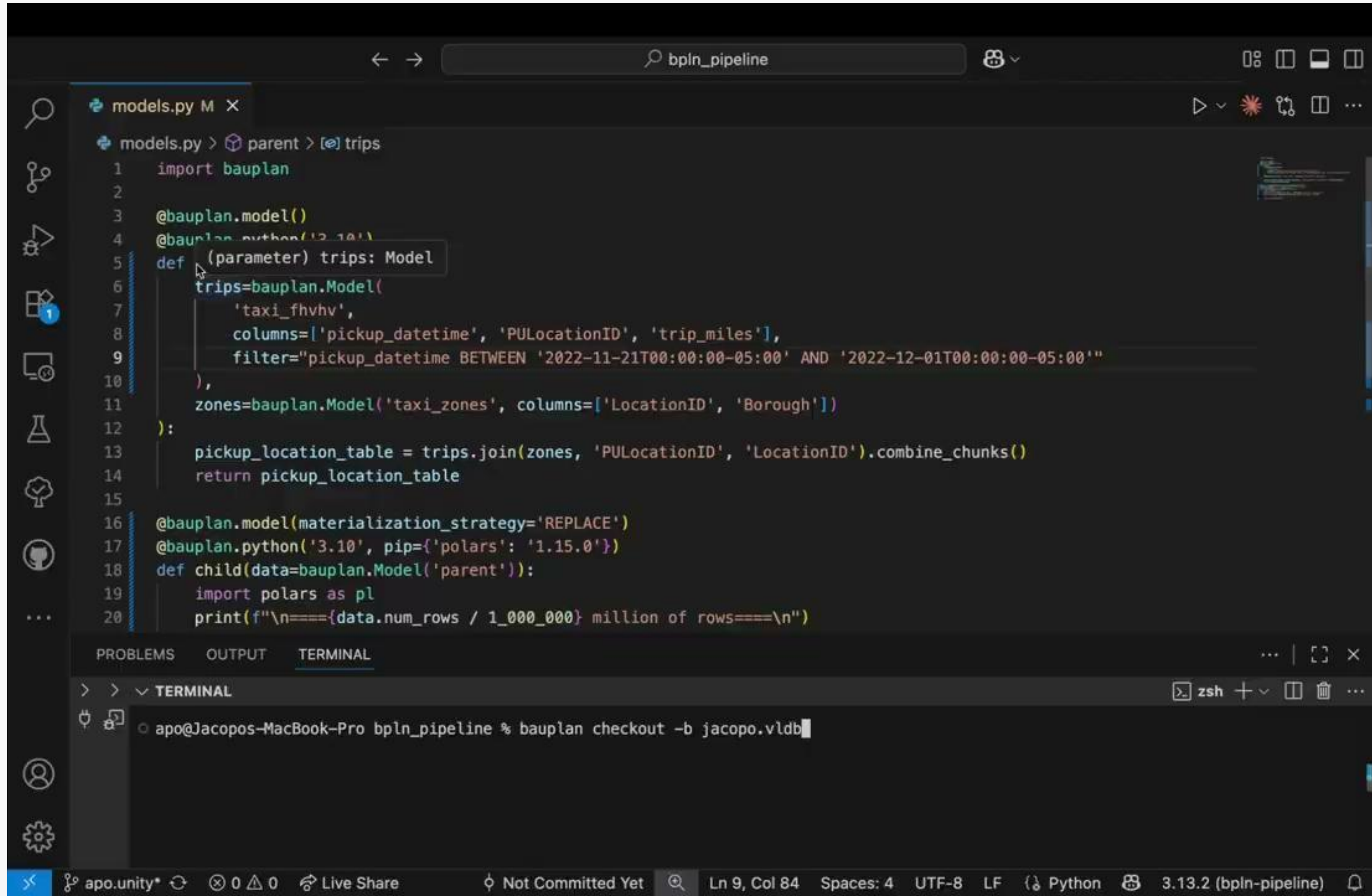




```
pip install bauplan  
bauplan checkout my-branch  
bauplan run
```



Seeing is believing



```
models.py M X
models.py > parent > [e] trips
1 import bauplan
2
3 @bauplan.model()
4 @bauplan.python('3.10')
5 def (parameter) trips: Model
6     trips=bauplan.Model(
7         'taxi_fhvvhv',
8         columns=['pickup_datetime', 'PULocationID', 'trip_miles'],
9         filter="pickup_datetime BETWEEN '2022-11-21T00:00:00-05:00' AND '2022-12-01T00:00:00-05:00'"
10    ),
11    zones=bauplan.Model('taxi_zones', columns=['LocationID', 'Borough'])
12 ):
13     pickup_location_table = trips.join(zones, 'PULocationID', 'LocationID').combine_chunks()
14     return pickup_location_table
15
16 @bauplan.model(materialization_strategy='REPLACE')
17 @bauplan.python('3.10', pip={'polars': '1.15.0'})
18 def child(data=bauplan.Model('parent')):
19     import polars as pl
20     print(f"\n===={data.num_rows / 1_000_000} million of rows====\n")

PROBLEMS OUTPUT TERMINAL
> > v TERMINAL
apo@Jacopos-MacBook-Pro bpln_pipeline % bauplan checkout -b jacopo.vldb
```

apo.unity* 0 0 Live Share Not Committed Yet Ln 9, Col 84 Spaces: 4 UTF-8 LF Python 3.13.2 (bpln-pipeline)

Everybody wants to ship “at scale” ...



Nearform

<https://nearform.com> › digital-community › vibe-codin... ⋮

Vibe coding is fun — until you have to ship at scale

14 May 2025 — Vibe coding is fun — until you have to **ship at scale**Vibe coding is f ... This concept, born from the increasing proficiency of large language ...



First Round Review

<https://review.firstround.com> › podcast › from-product-... ⋮

How to ship software at scale — Snir Kodesh

From product roadmapping to sprint planning: How to **ship software at scale** — Snir Kodesh. Snir Kodesh is the Head of Engineering at Retool and former Senior ...



HubSpot

<https://product.hubspot.com> › blog › how-we-built-our-s... ⋮

How We Built Our Stack For Shipping at Scale

How We Built Our Stack For **Shipping at Scale** ... We've designed our team structure, development processes, and technical architecture to promote strong team ...



Acceldata

<https://www.acceldata.io> › guide › data-pipelines-how-t... ⋮

Data Pipeline Optimization at Scale

The following provides an in-depth understanding of how to optimize **data pipelines at scale** with data observability.

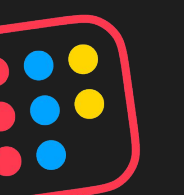


Medium · Marvich

2 months ago ⋮

How we solved Databricks Pipeline observability at scale ...

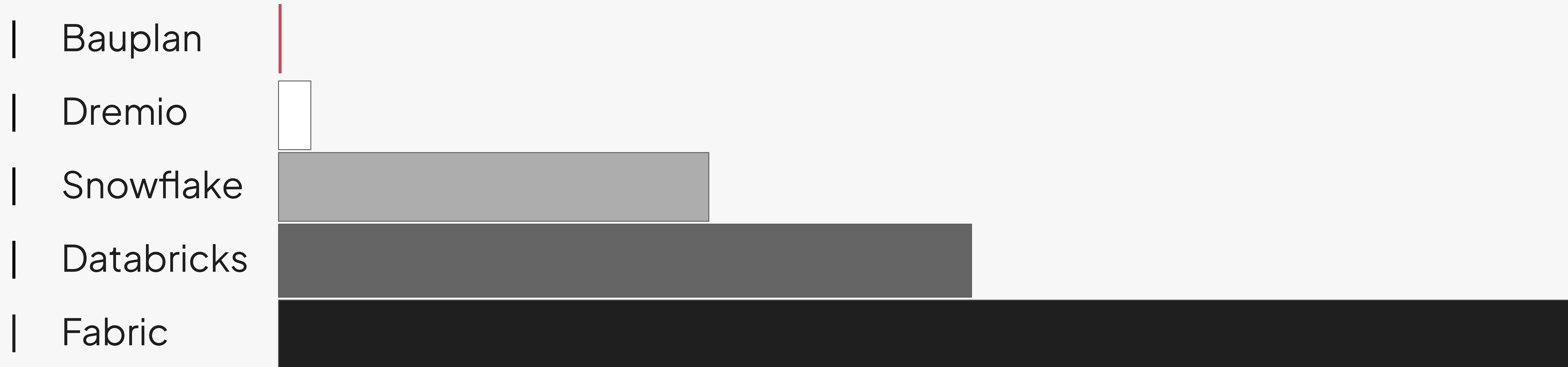
Here's how we tackled **pipeline** observability in Databricks **at scale**, and why it was far more challenging (and expensive) than expected.

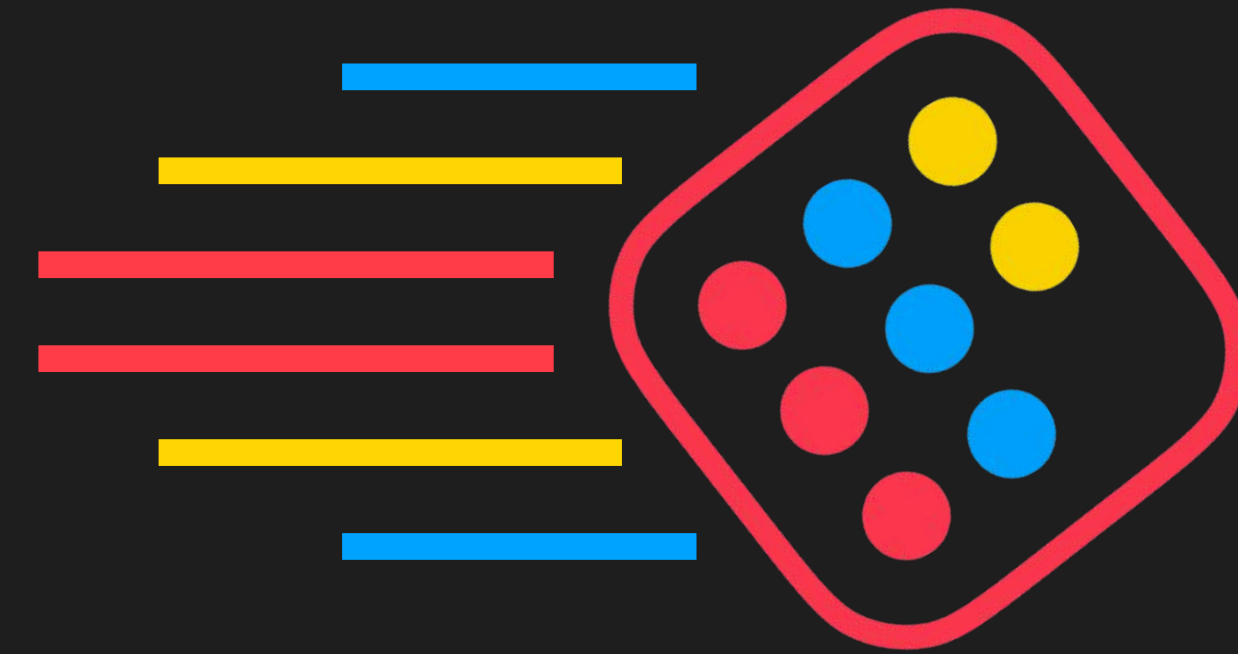


Building a Data Platform



Building a Data Platform





...but **startups** make you
ship “at speed”



Building Speedrunning a Data Platform

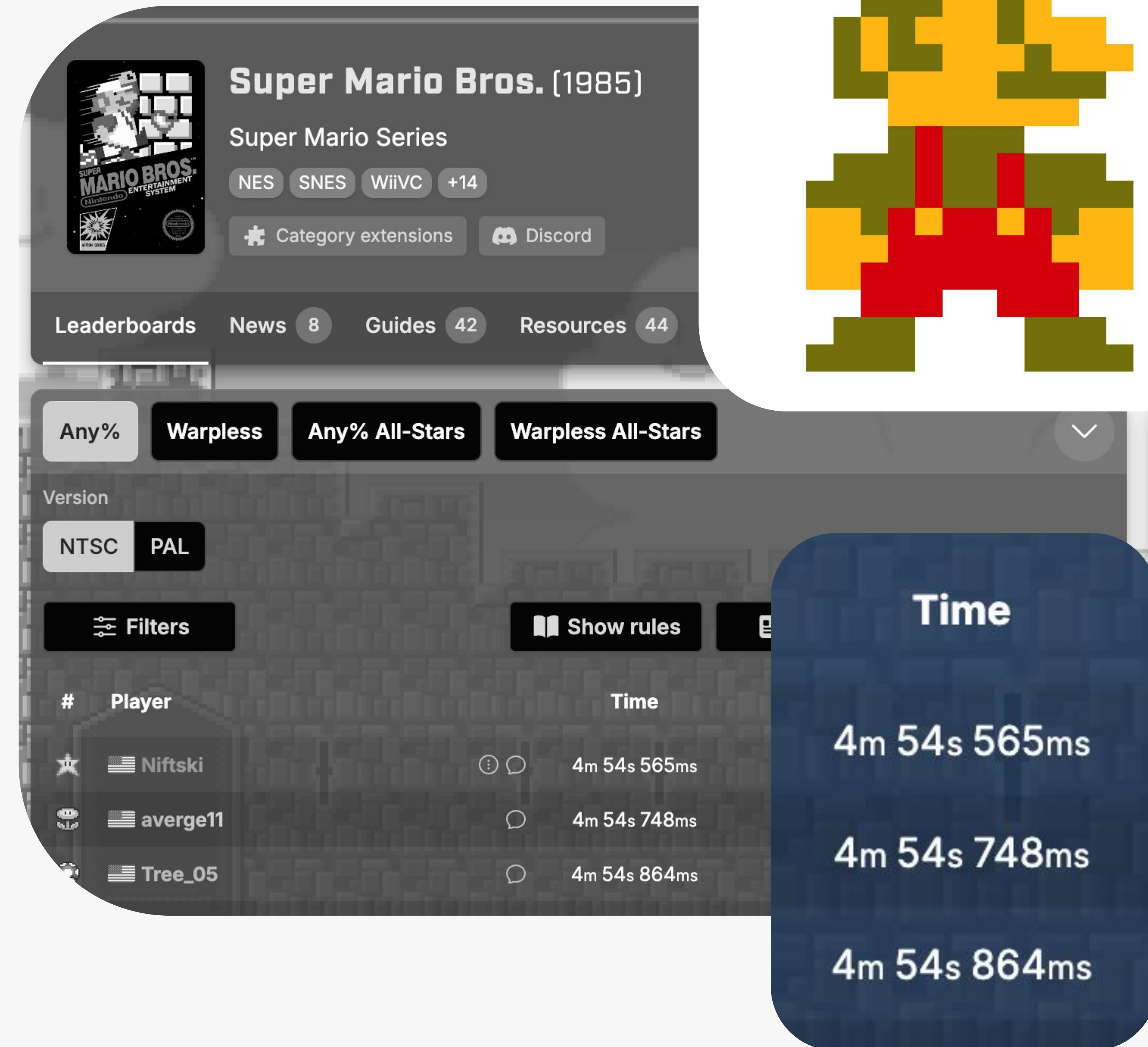
 **speed·run**
/'spēd,rən/

verb

gerund or present participle: **speedrunning**

complete (a video game, or level of a game) as fast as possible.

"I used to be able to speedrun this game in less than 20 minutes"






Super Mario Bros. (1985)
Super Mario Series
NES SNES WiiVC +14
Category extensions Discord

Leaderboards News 8 Guides 42 Resources 44

Any% Warpless Any% All-Stars Warpless All-Stars

Version
NTSC PAL

Filters Show rules

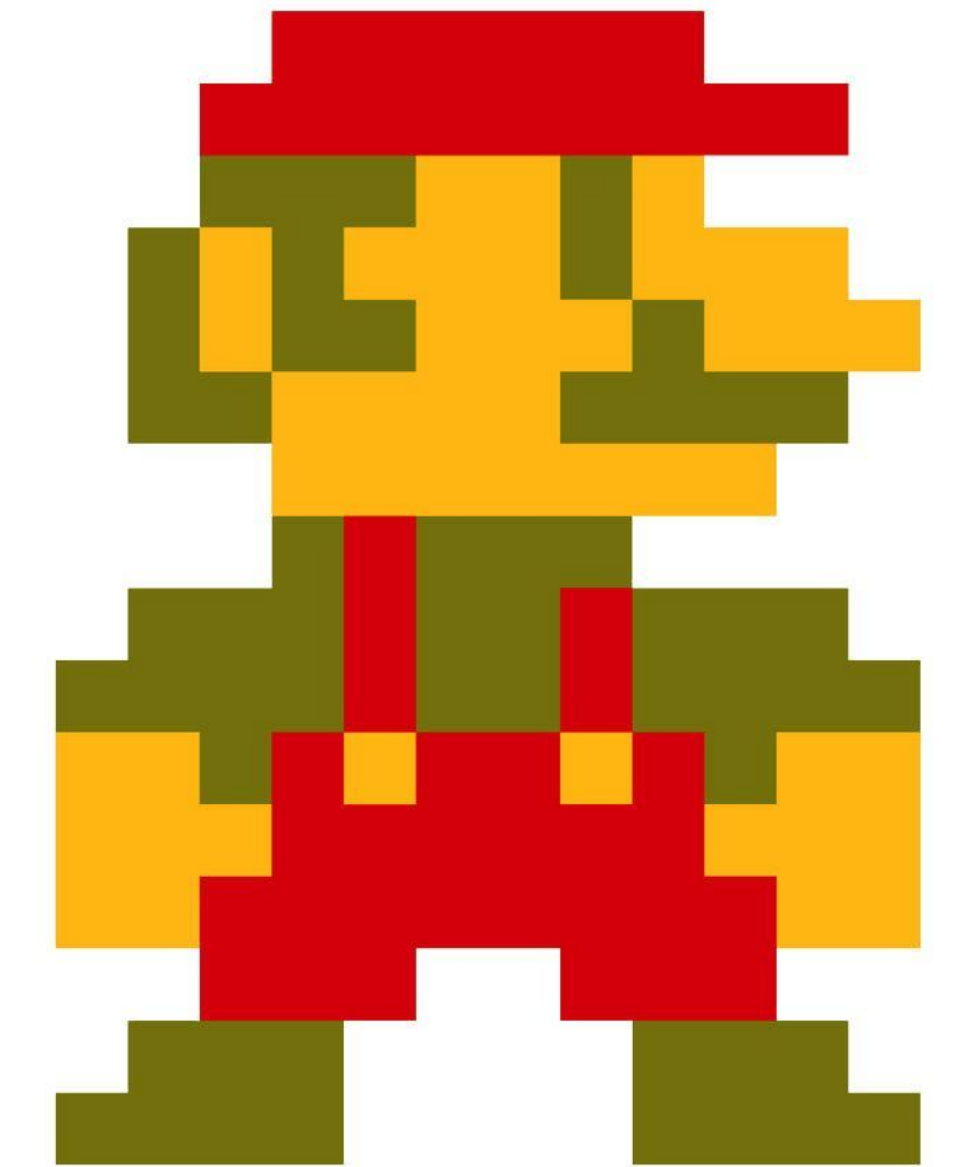
#	Player	Time
★	 Niftski	4m 54s 565ms
	 averge11	4m 54s 748ms
	 Tree_05	4m 54s 864ms

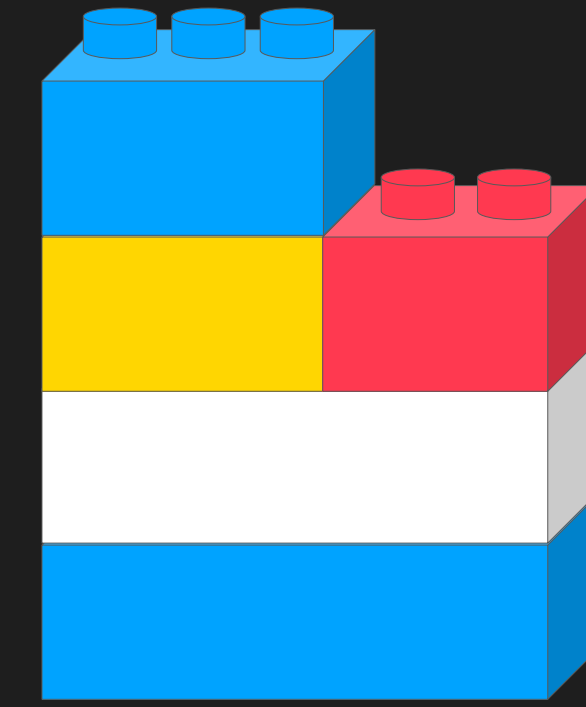
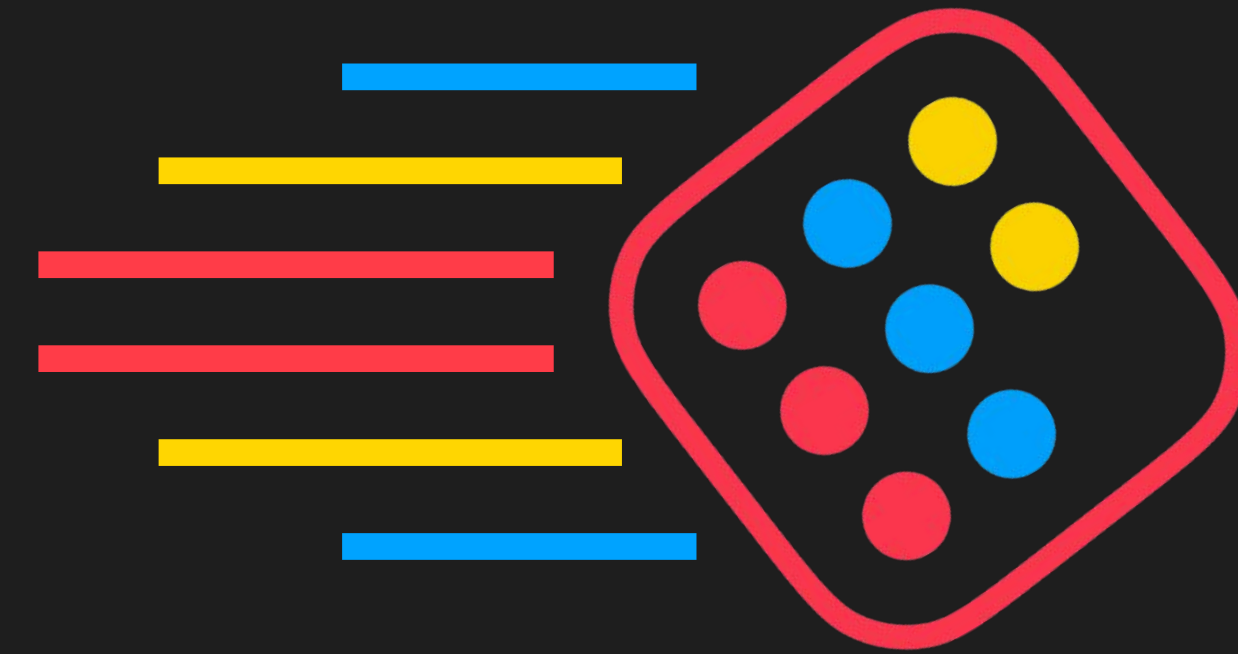
Time

4m 54s 565ms

4m 54s 748ms

4m 54s 864ms





Speedrunning > speed > **composability**





The Composable Data Platform

"bauplan is [a system] fully built using composable principles (...). It is refreshing to learn about a real-life system built using such architectural principles"

Reviewer 2

The Composable Data Management System Manifesto

Pedro Pedreira Meta Platforms Inc. pedroerp@meta.com	Orri Erling Meta Platforms Inc. oerling@meta.com	Konstantinos Karanasos Meta Platforms Inc. kkaranasos@meta.com	Scott Schneider Meta Platforms Inc. scottas@meta.com
Wes McKinney Voltron Data wes@voltrondata.com	Satya R Valluri Databricks Inc. satya.valluri@databricks.com	Mohamed Zait Databricks Inc. mohamed.zait@databricks.com	Jacques Nadeau Sundeck jacques@sundeck.io

ABSTRACT

The requirement for specialization in data management systems has evolved faster than our software development practices. After decades of organic growth, this situation has created a siloed landscape composed of hundreds of products developed and maintained as monoliths, with limited reuse between systems. This fragmentation has resulted in developers often reinventing the wheel, increased maintenance costs, and slowed down innovation. It has also affected the end users, who are often required to learn the idiosyncrasies of dozens of incompatible SQL and non-SQL API dialects, and settle for systems with incomplete functionality and inconsistent semantics. In this vision paper, considering the recent popularity of open source projects aimed at standardizing different aspects of the data stack, we advocate for a paradigm shift in how data management systems are designed. We believe that by decomposing these into a modular stack of reusable components, development can be streamlined while creating a more consistent experience for users. Towards that goal, we describe the state-of-the-art, principal open source technologies, and highlight open questions and areas where additional research is needed. We hope this work will foster collaboration, motivate further research, and promote a more composable future for data management.

VLDB Reference Format:

the first databases were developed, our software development practices have not; data management systems continue to be, by and large, developed and distributed as vertically integrated monoliths.

While modern specialized data systems may seem distinct at first, at the core, they are all composed of a similar set of logical components: (a) a **language frontend**, responsible for interpreting user input into an internal format; (b) an **intermediate representation** (IR), usually in the form of a logical and/or physical query plan; (c) a **query optimizer**, responsible for transforming the IR into a more efficient IR ready for execution; (d) an **execution engine**, able to locally execute query fragments (also sometimes referred to as the *eval engine*); and (e) an **execution runtime**, responsible for providing the (often distributed) environment in which query fragments can be executed. Beyond having the same logical components, the data structures and algorithms used to implement these layers are also largely consistent across systems. For example, there is nothing *fundamentally different* between the SQL frontend of an operational database system and that of a data warehouse; or between the expression evaluation engines of a traditional columnar DBMS and that of a stream processing engine; or between the string, date, array, or json manipulation functions across database systems.

However, this fragmentation and consequent lack of reuse across systems has slowed us down. It has forced developers to reinvent the wheel, duplicating work and hurting our ability to quickly ad-

Building a serverless Data Lakehouse from spare parts*

Jacopo Tagliabue^{1,2,*}, [Ciro Greco](#)¹ and [Luca Bigon](#)^{1,†}

¹*Bauplan, New York City, United States*

²*Tandon School of Engineering, NYU, New York City, United States*

Abstract

The recently proposed Data Lakehouse architecture is built on open file formats, performance, and first-class support for data transformation, BI and data science: while the vision stresses the importance of lowering the barrier for data work, existing implementations often struggle to live up to user expectations. At *Bauplan*, we decided to build a new serverless platform to fulfill the Lakehouse vision. Since building from scratch is a challenge unfit for a startup, we started by re-using (sometimes unconventionally) existing projects, and then investing in improving the areas that would give us the highest marginal gains for the developer experience. In this work, we review user experience, high-level architecture and tooling decisions, and conclude by sharing plans for future development.

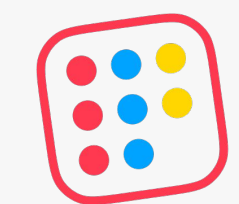
Keywords

data lakehouse, data pipelines, serverless, reasonable scale, containerized execution

1. Introduction

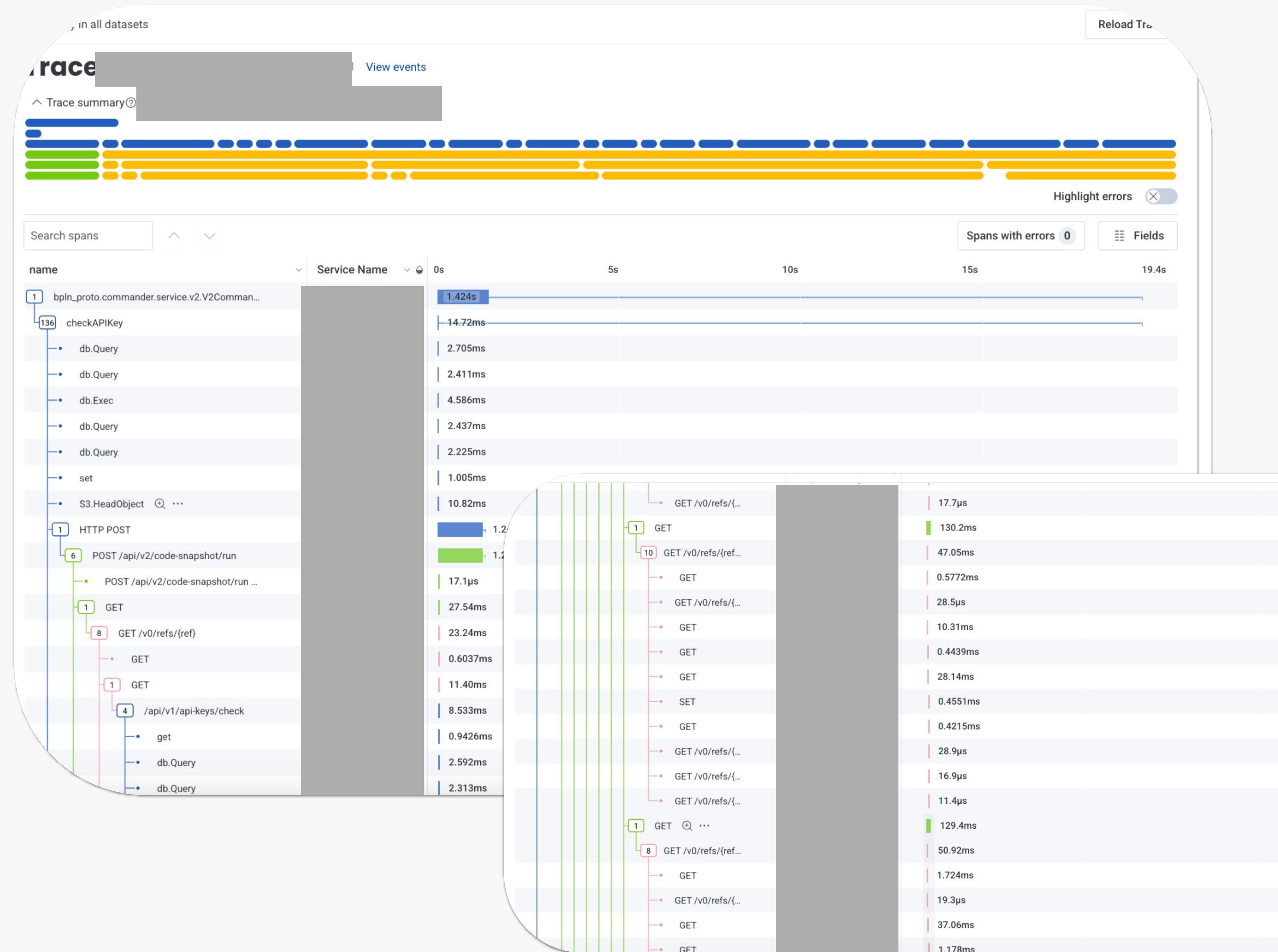
[2] argues that the popular data warehouse architecture will soon be replaced by a new architectural pattern, the Data Lakehouse (DLH). A DLH is built on open file

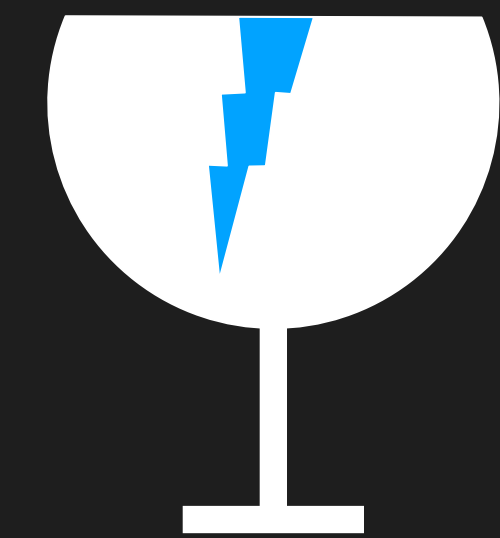
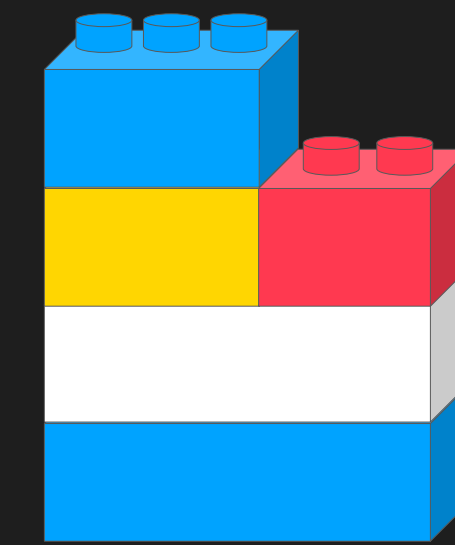
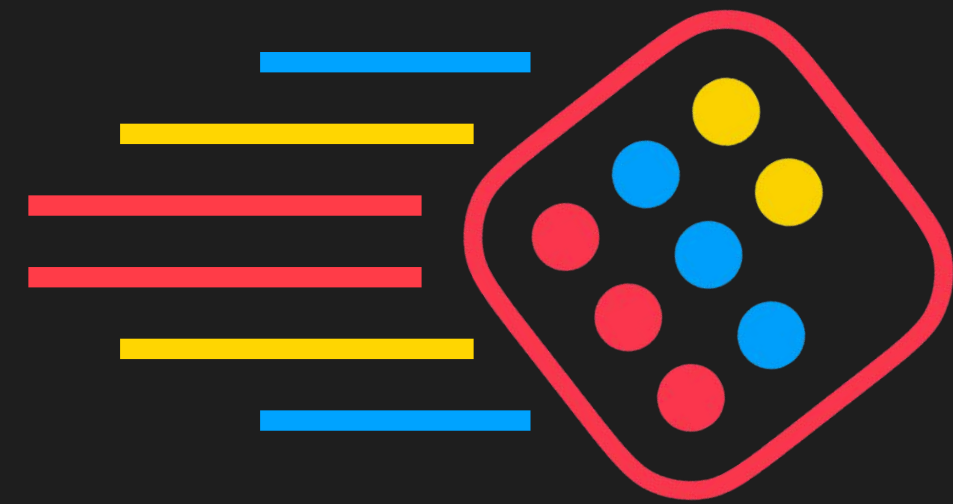
There are two primary approaches to realize the DLH vision. The first is improving the usability and flexibility of existing Big Data technologies: e.g., one could start by adding automated cluster configurations to Apache Spark. Although everyone will stand behind easier devel-



Inside a bauplan run

A single *run* spans
hundreds of traces,
across dozens of
services, ranging from
hyper-scaler PaaS to
obscure open-source
libraries.

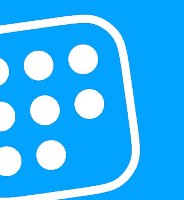




Speedrunning > speed > composability > **fragility?**



Move fast AND do not break things



Bauplan “at scale”

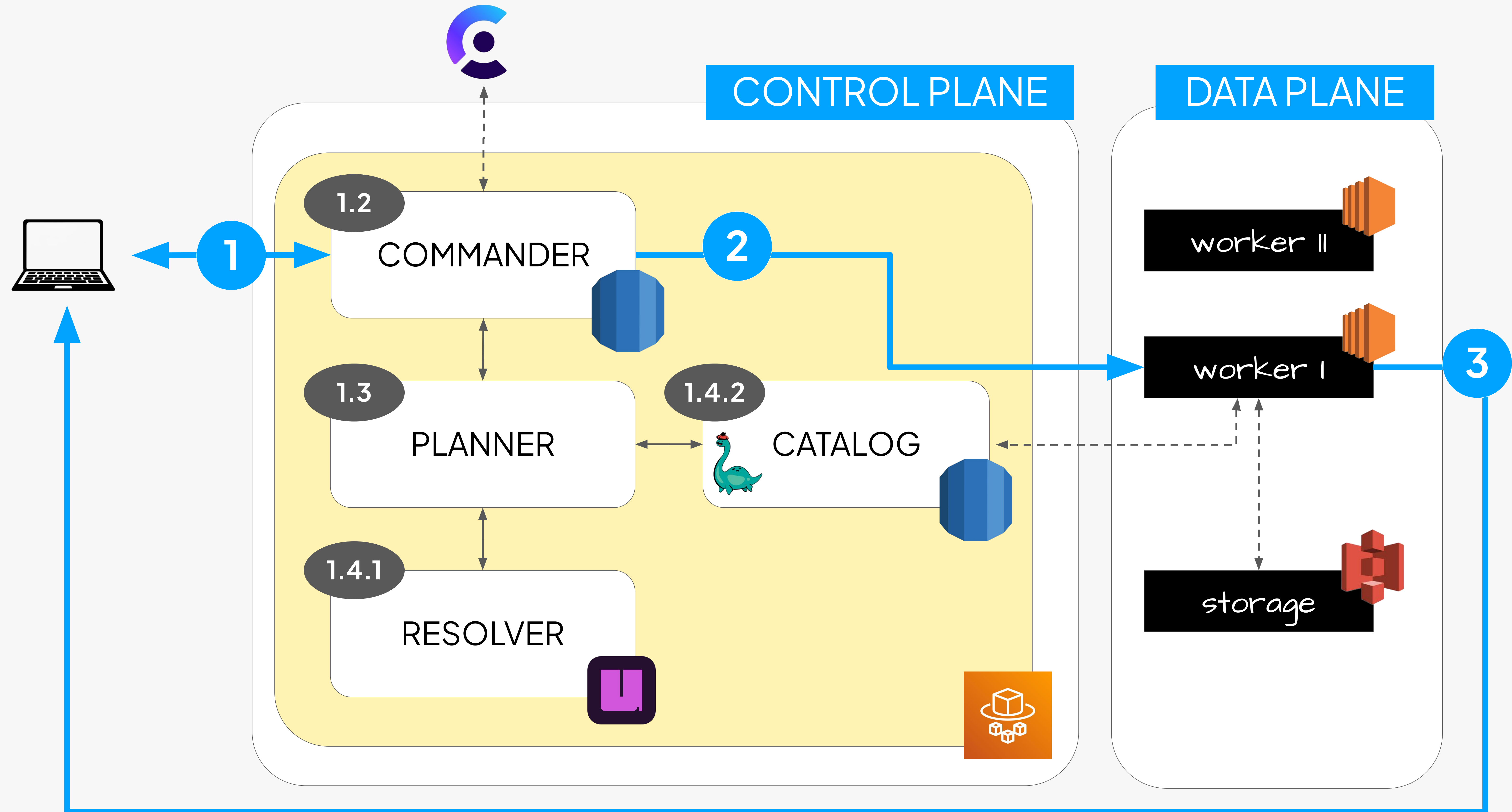
 ~150k `bauplan import`

 ~180k `bauplan run`

 ~250k `bauplan branch`



The interest rate on composability



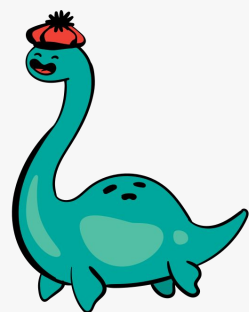
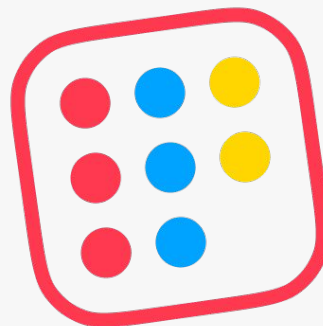


The interest rate on composability

Composability is NOT a get-out-of-jail card:

- | You need to work with the community and learn the ropes (technically, culturally).
- | You need to adapt, improve, maintain components.
- | You contribute back whenever possible.

25x
faster!



The Deconstructed Warehouse: An Ephemeral Query Engine Design for Apache Iceberg

Ryan Curtin
Independent Researcher
ryan@ratml.org

Jacopo Tagliabue
Bauplan Labs
jacopo.tagliabue@bauplanlabs.com

ABSTRACT

The rise of open formats (e.g. Apache Iceberg, Delta Lake) and single-node vectorized engines (e.g. DuckDB, DataFusion) have been important recent trends in data systems. Perhaps surprisingly, these trends did not interact to produce a credible, cloud-first OLAP experience *yet*: while open formats are a staple of large scale lake-houses [5], and proprietary offerings emerged over the DuckDB format [1], managed Iceberg-native experience are still lacking.

In the spirit of the ‘Composable Data Manifesto’ [2], we show how to achieve warehouse-like capabilities (with very modest resources) through ephemeral functions and disaggregated components—storage, data catalog, planning, caching, execution. We summarize our contributions as follows:

- (1) we present a novel design for integrating data catalogs, open formats and single-node engines into a “deconstructed warehouse”;
- (2) we motivate and detail a new command we implemented in our DuckDB fork, EXPLAIN SCANS, which sits in between the logical and the physical plan as an intermediate optimization;
- (3) we present preliminary results when benchmarking our I/O approach against sensible alternatives.

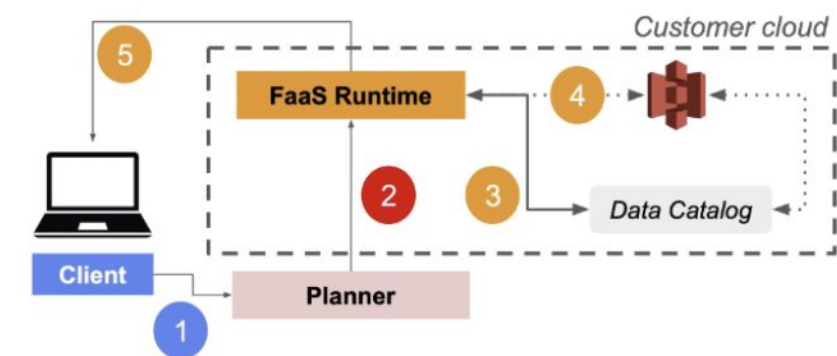


Figure 1: A FaaS-based architecture with storage-compute decoupling, ephemeral serverless compute and modular planning.

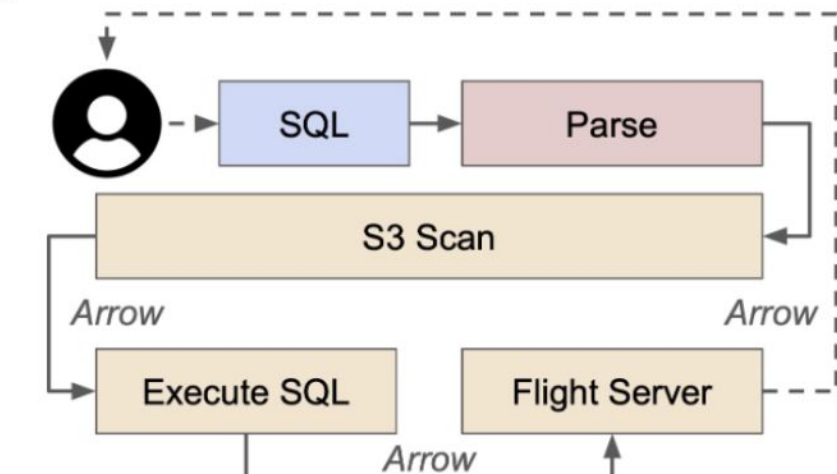


Figure 2: Query execution as a sequence of functions exchanging Arrow streams

DAG lakehouse planning with an ephemeral and embedded graph database

Luca Bigon
Bauplan Labs
luca.bigon@bauplanlabs.com

Jacopo Tagliabue
Bauplan Labs
jacopo.tagliabue@bauplanlabs.com

Semih Salihoğlu
Kuzu Inc.
semih@kuzudb.com

```
@bauplan.model()
@bauplan.python("3.10", pip={"pandas": "2.0"})
def cleaned_data(
    # reference to its parent DAG node
    data=bauplan.Model(
        "raw_data",
        columns=["c1", "c2", "c3"],
        filter="eventTime BETWEEN 2023-01-01 AND 2023-02-01"
    )
):
    # the body returns a dataframe after transformations
    return data.do_something()

@bauplan.model()
@bauplan.python("3.11", pip={"pandas": "1.5"})
def final_data(
    data=bauplan.Model("cleaned_data")
):
    return data.do_something()
```

Figure 1: A two nodes DAG in Bauplan.

ABSTRACT

Bauplan is a code-first lakehouse built by vertically integrating through APIs modular data components – catalog, I/O, runtime, Flight server etc. [5]. To abstract the underlying complexity away from users, Bauplan provides a declarative functional framework

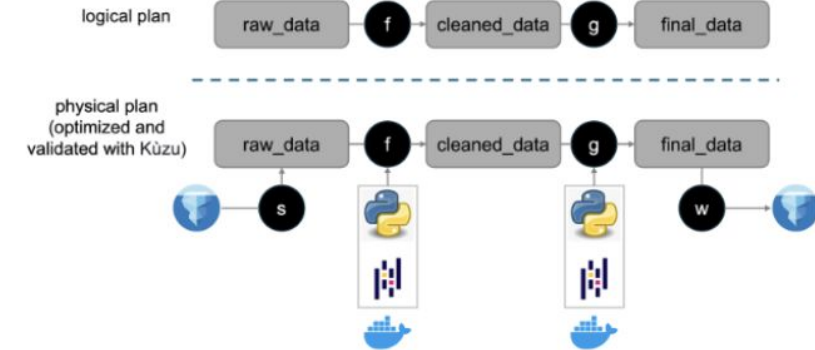
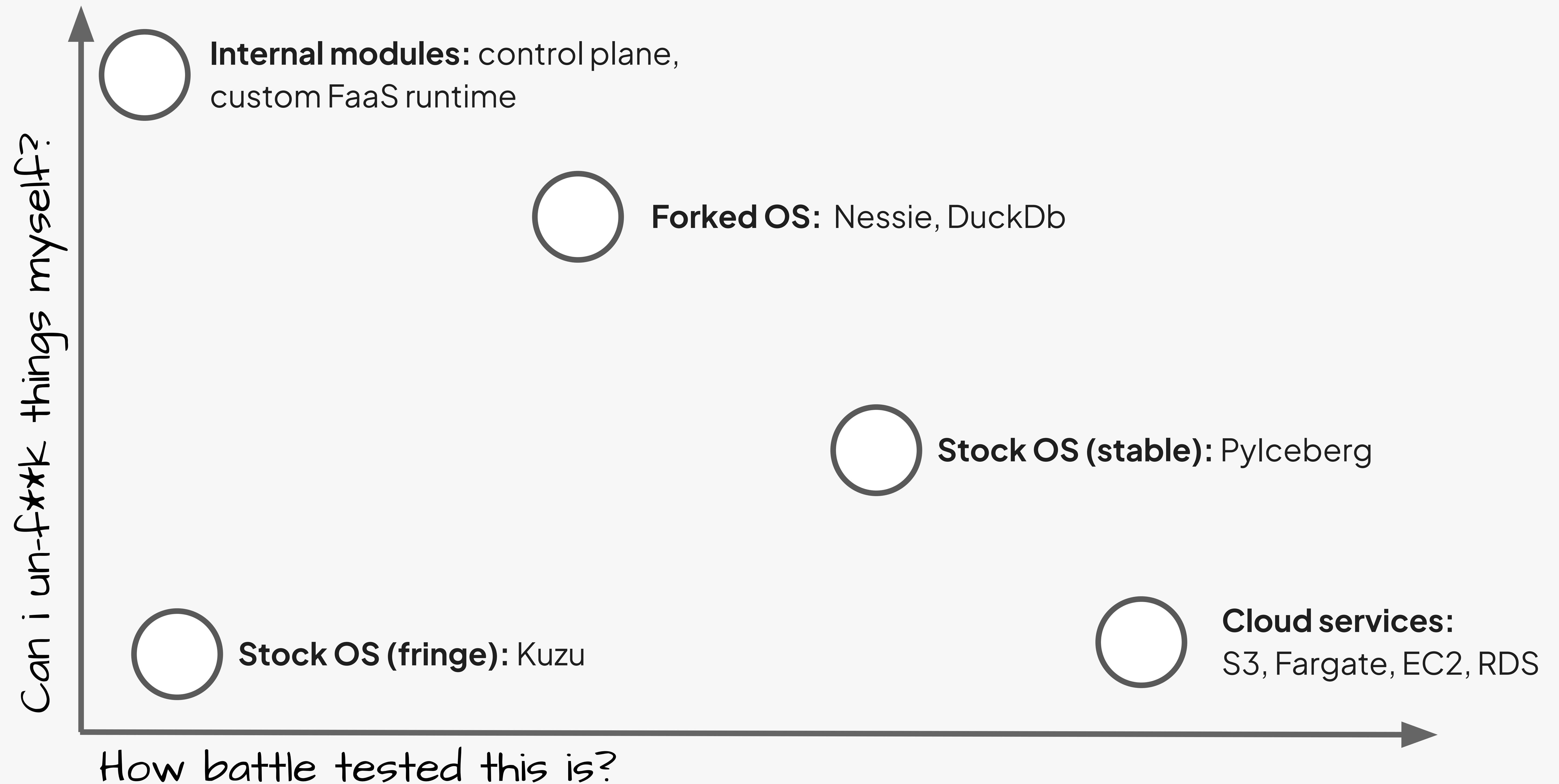


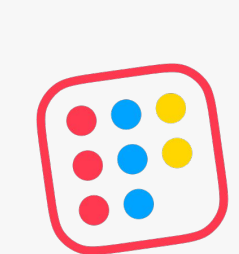
Figure 2: The logical plan is created by parsing user code, the physical plan is obtained running Cypher on Kuzu.

inference through recursive queries; (ii) optimized query execution with leveraging multi-core hardware.

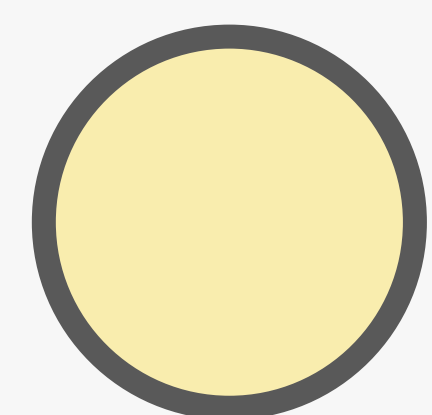
Given the on-demand nature of our workloads, we wish to move the embedded GDBMS in memory, further simplifying our infrastructure life-cycle and speeding up queries. In collaboration with the Kuzu team, we developed an in-memory version of their database, so that we could leverage a new, ephemeral graph at every run: as a result, we currently create tens of thousands of ephemeral graph databases on-the-fly per day (and growing). The in-memory version provided optimized inference without infrastructure depen-

The dependency plane

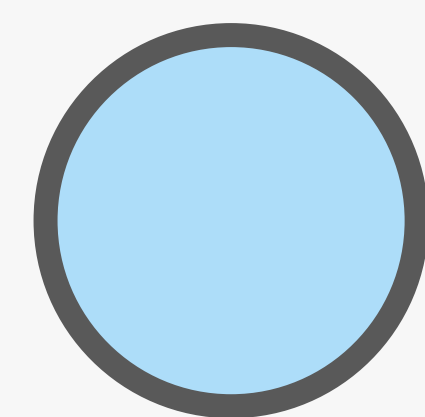




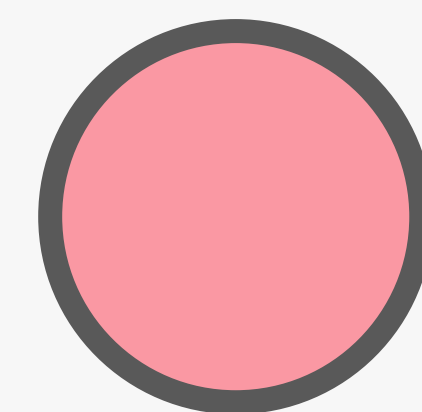
The testing line



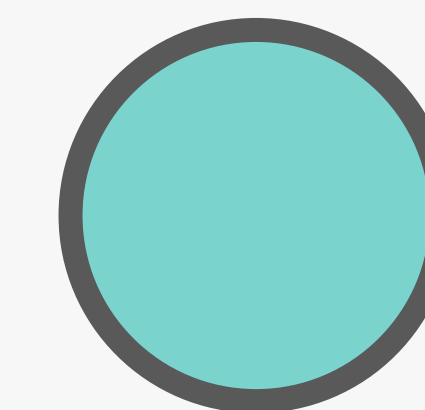
- Unit tests
- Small integration tests



End-to-End tests

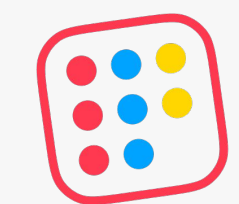


Simulations

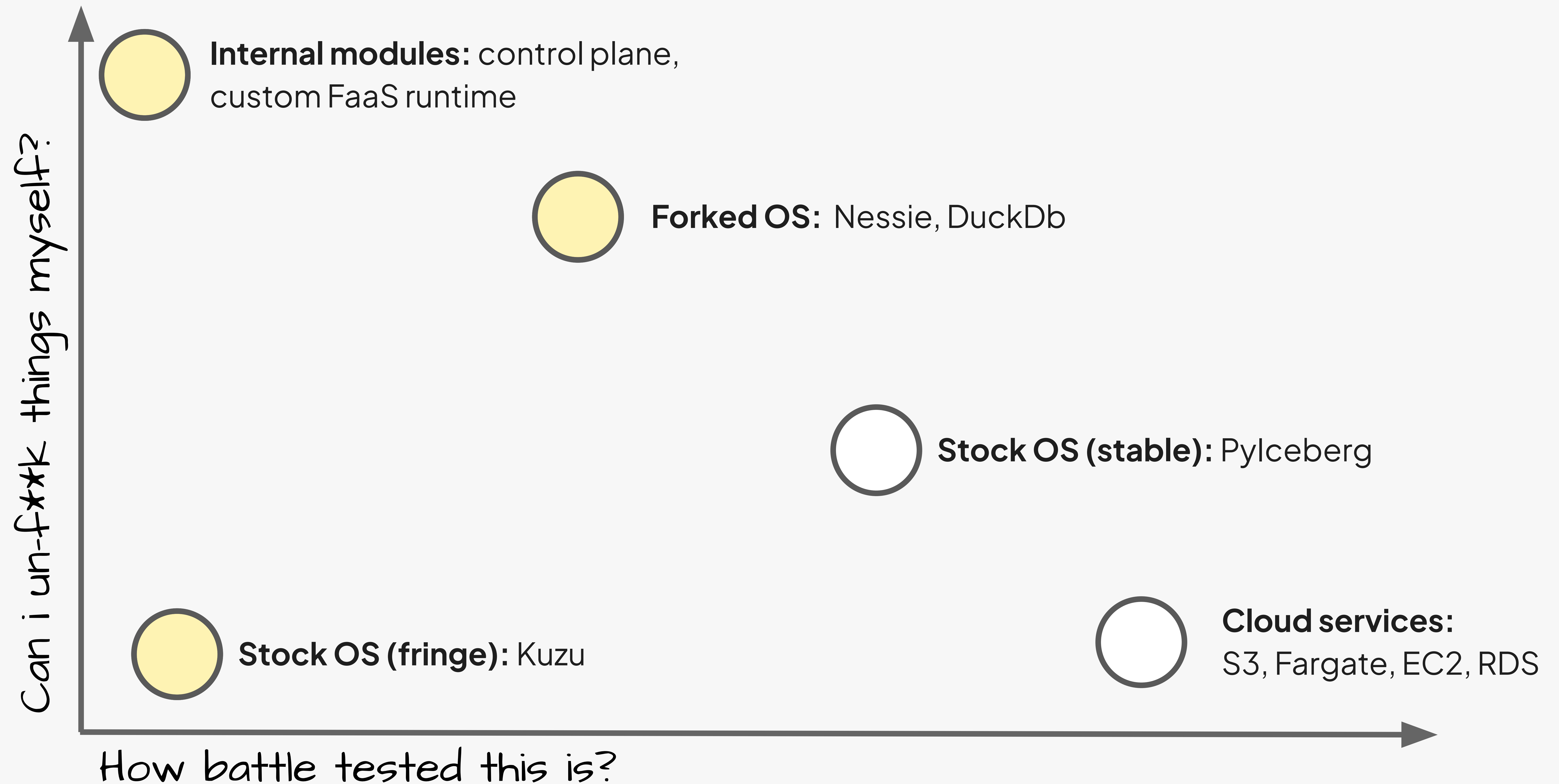


Formal proof



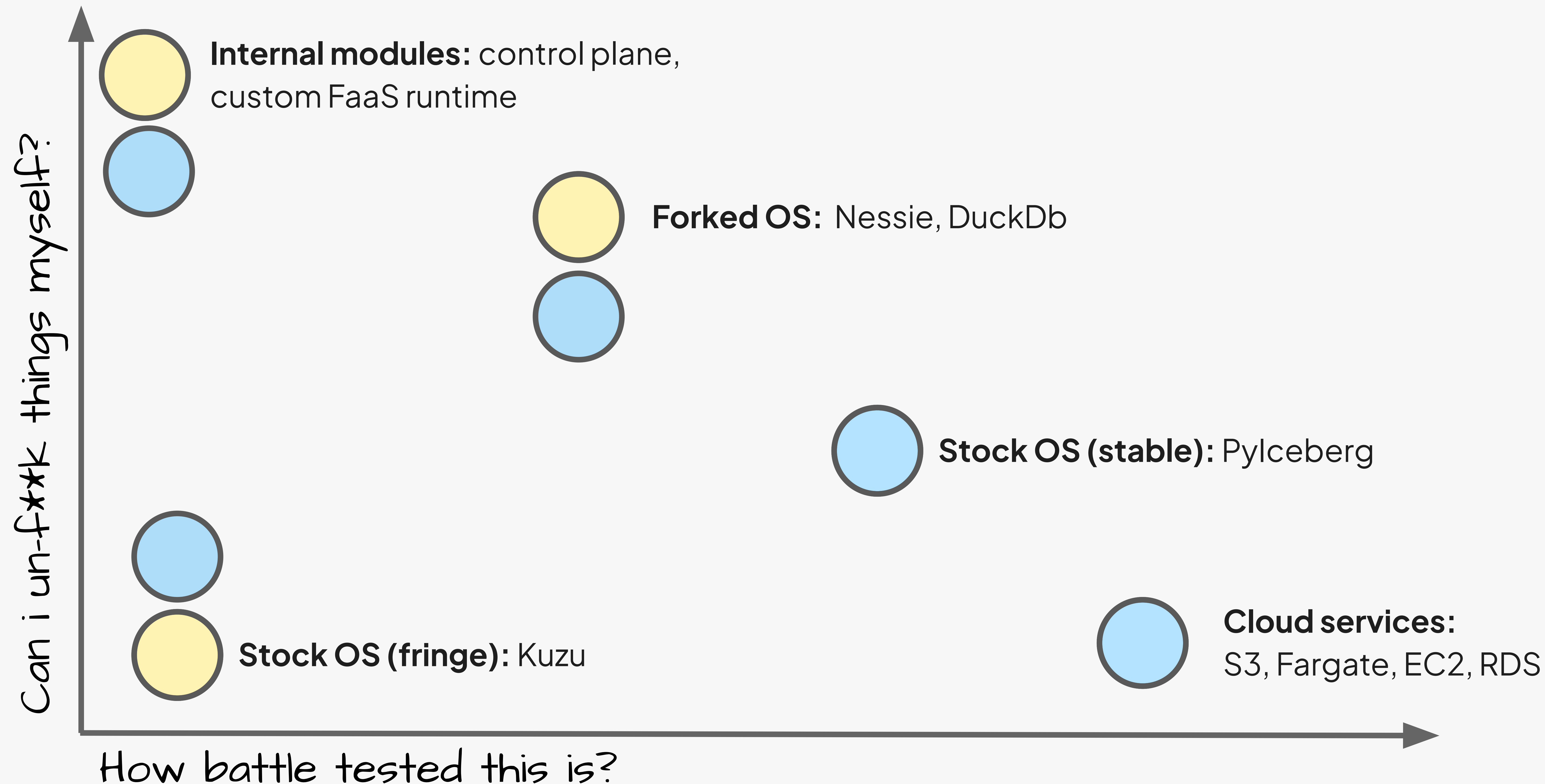


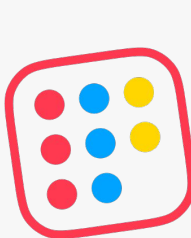
Unit Tests



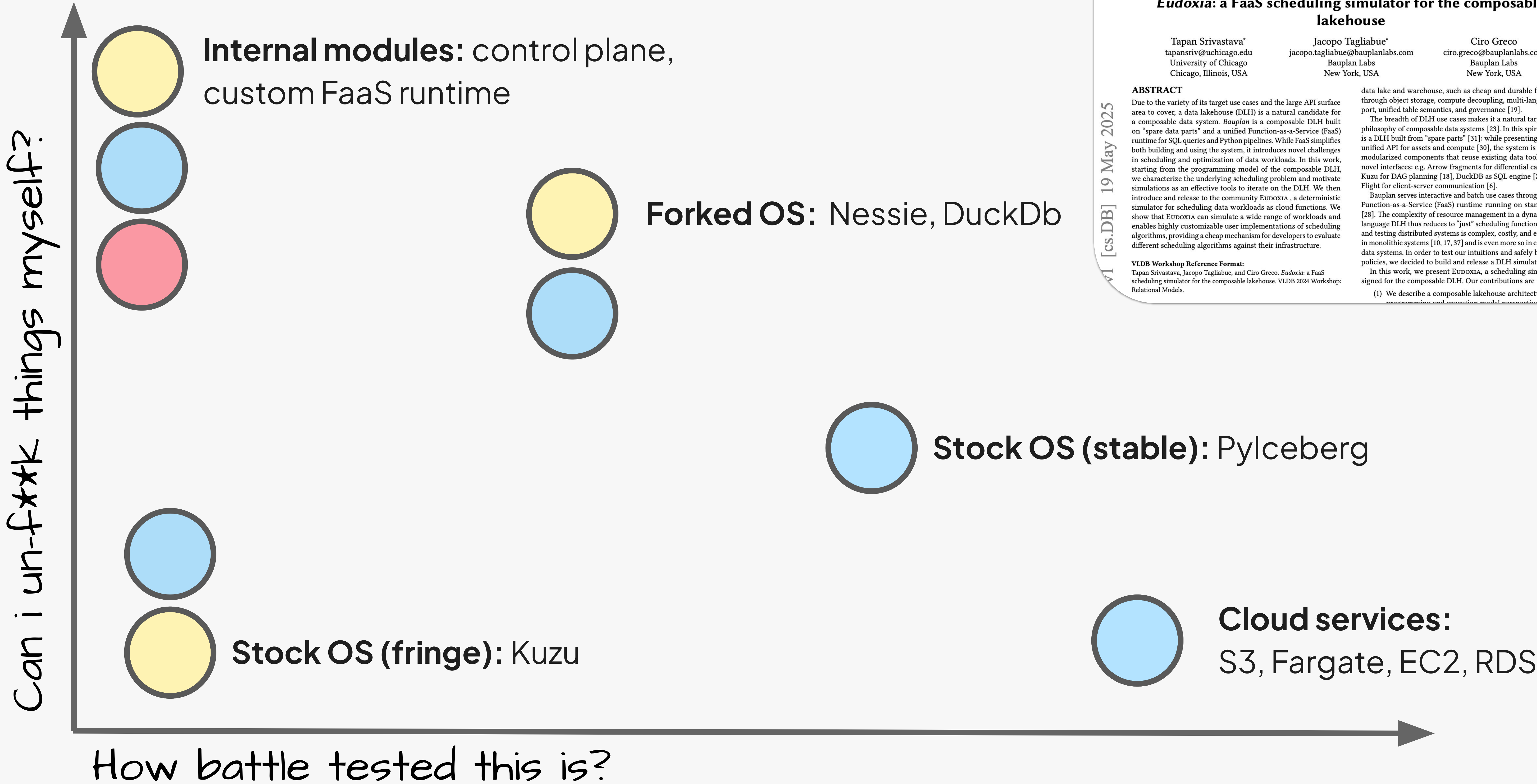


End-to-end tests





Simulations



Eudoxia: a FaaS scheduling simulator for the composable lakehouse

Tapan Srivastava*
tapansriv@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Jacopo Tagliabue*
jacopo.tagliabue@bauplanlabs.com
Bauplan Labs
New York, USA

Ciro Greco
ciro.greco@bauplanlabs.com
Bauplan Labs
New York, USA

ABSTRACT

Due to the variety of its target use cases and the large API surface area to cover, a data lakehouse (DLH) is a natural candidate for a composable data system. *Bauplan* is a composable DLH built on “spare data parts” and a unified Function-as-a-Service (FaaS) runtime for SQL queries and Python pipelines. While FaaS simplifies both building and using the system, it introduces novel challenges in scheduling and optimization of data workloads. In this work, starting from the programming model of the composable DLH, we characterize the underlying scheduling problem and motivate simulations as an effective tools to iterate on the DLH. We then introduce and release to the community EUDOXIA, a deterministic simulator for scheduling data workloads as cloud functions. We show that EUDOXIA can simulate a wide range of workloads and enables highly customizable user implementations of scheduling algorithms, providing a cheap mechanism for developers to evaluate different scheduling algorithms against their infrastructure.

VLDB Workshop Reference Format:
Tapan Srivastava, Jacopo Tagliabue, and Ciro Greco. *Eudoxia: a FaaS scheduling simulator for the composable lakehouse*. VLDB 2024 Workshop: Relational Models.

data lake and warehouse, such as cheap and durable foundation through object storage, compute decoupling, multi-language support, unified table semantics, and governance [19].

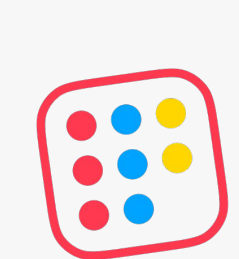
The breadth of DLH use cases makes it a natural target for the philosophy of composable data systems [23]. In this spirit, *Bauplan* is a DLH built from “spare parts” [31]: while presenting to users a unified API for assets and compute [30], the system is built from modularized components that reuse existing data tools through novel interfaces: e.g. Arrow fragments for differential caching [29], Kuzu for DAG planning [18], DuckDB as SQL engine [24], Arrow Flight for client-server communication [6].

Bauplan serves interactive and batch use cases through a unified Function-as-a-Service (FaaS) runtime running on standard VMs [28]. The complexity of resource management in a dynamic, multi-language DLH thus reduces to “just” scheduling functions. Building and testing distributed systems is complex, costly, and error-prone in monolithic systems [10, 17, 37] and is even more so in composable data systems. In order to test our intuitions and safely benchmark policies, we decided to build and release a DLH simulator.

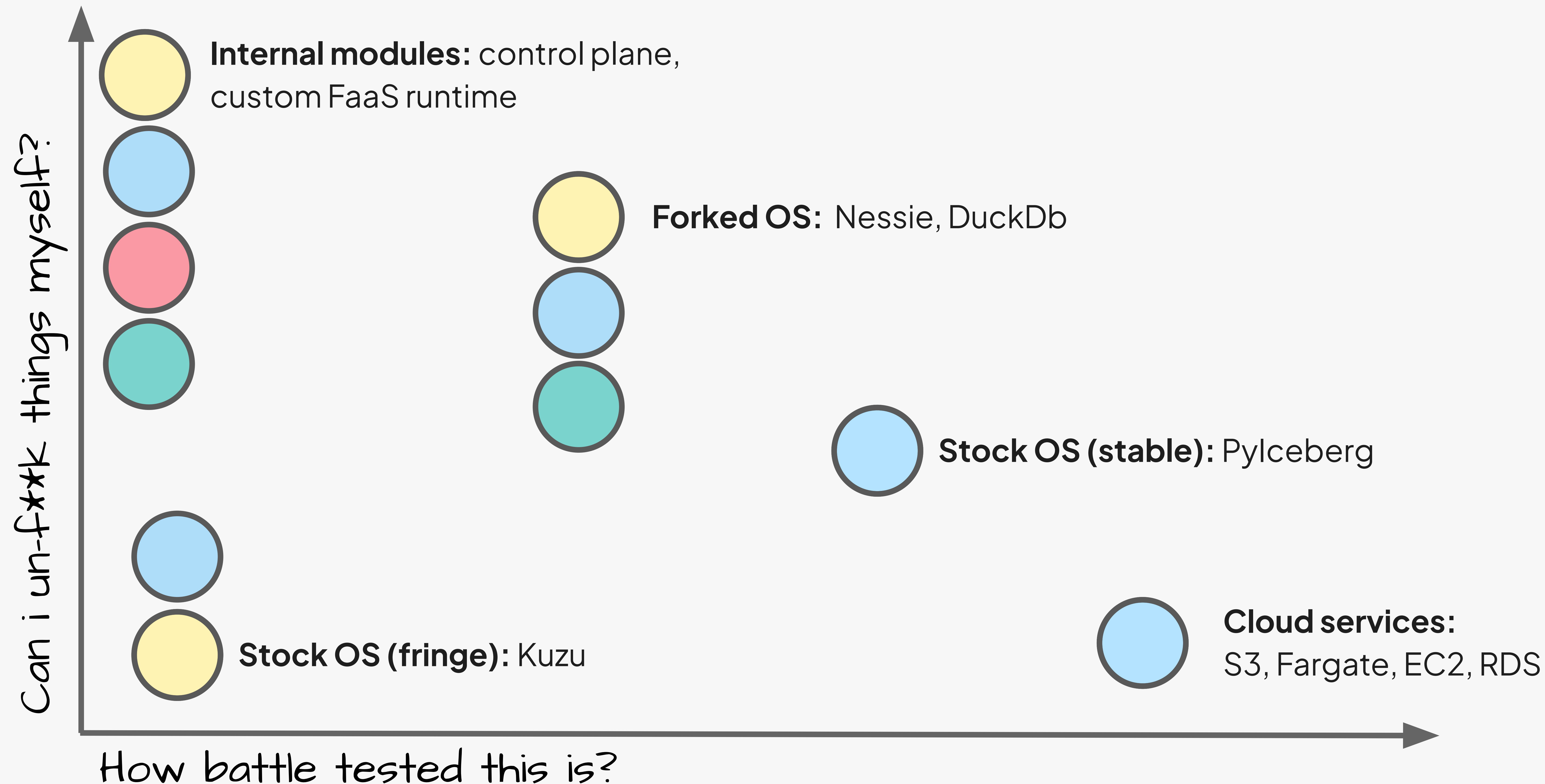
In this work, we present EUDOXIA, a scheduling simulator designed for the composable DLH. Our contributions are threefold:

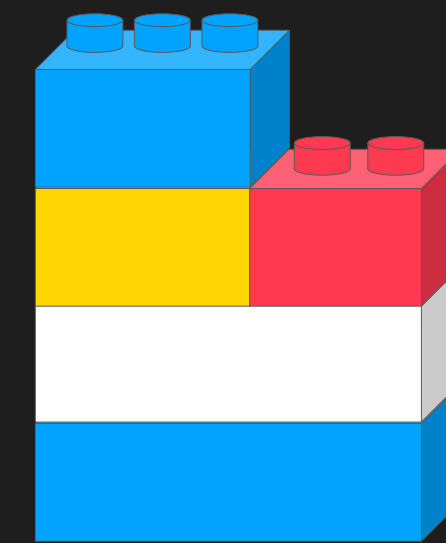
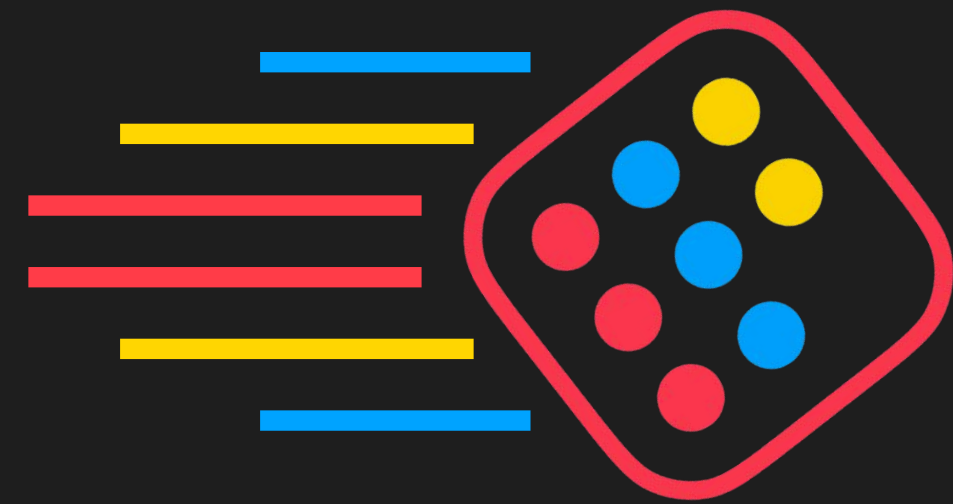
(1) We describe a composable lakehouse architecture from a programming and execution model perspective, showing

VI [cs.DB] 19 May 2025

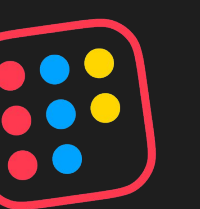


Formal proofs

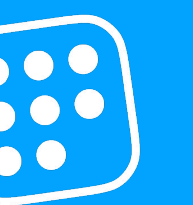




Speedrunning > speed > composability > ~~fragility~~
testability



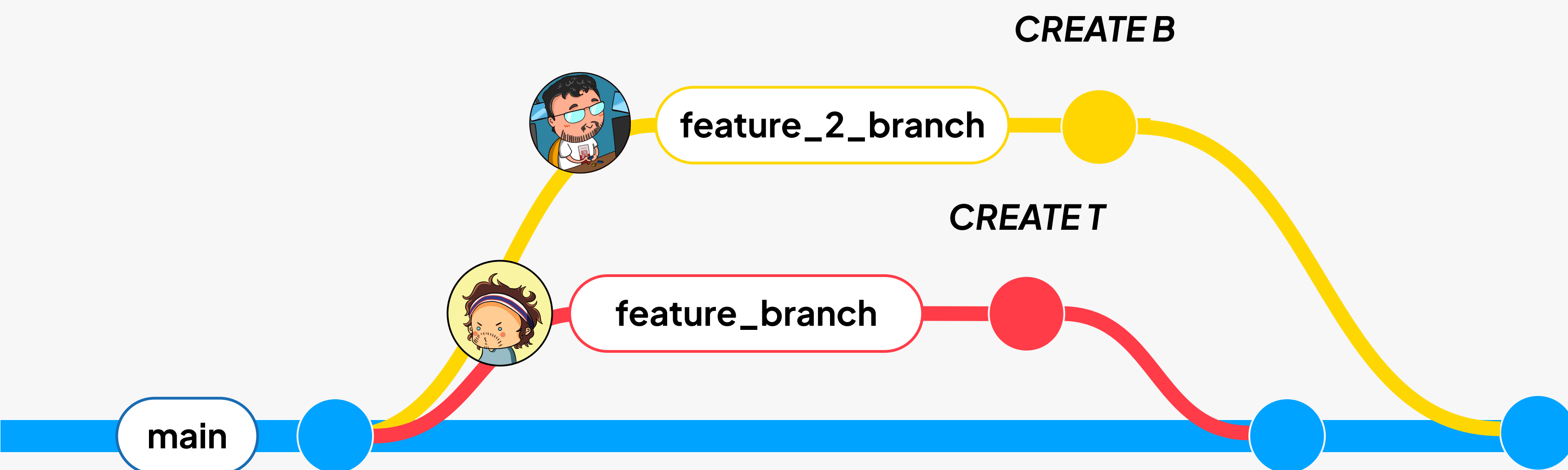
A tale of two systems: Git and MVCC



How much “Git” is in Git-for-data?

Git-for-data:

- | Multi-player mode (branch, merge)
- | Auditability + Time-travel (history, revert)



engineering Jul 17, 2025 Written by [Ciro Greco](#) and [Jacopo Tagliabue](#)

Git for Data: Formal Semantics of Branching, Merging, and Rollbacks (Part 1)

How formal methods help ensure safe, reproducible workflows in data lakehouses

Tables are for boys, pipeline are for men

| Apache Iceberg tables are an **abstraction over data files in object storage**, allowing atomic table writes (through a table-level optimistic lock).

The Mechanics of Apache Iceberg

Like all table formats, Iceberg is both a specification and a set of supporting libraries. The specification standardizes how to represent a table as a set of metadata and data files. Iceberg also defines a protocol for how to manipulate those files while safeguarding data consistency - this protocol is not fully documented in the specification but exists in the Iceberg code itself.

An Iceberg table's files are split between a metadata layer and a data layer, both stored in an object store such as S3. The difference with Iceberg is that commits are performed against a catalog component.

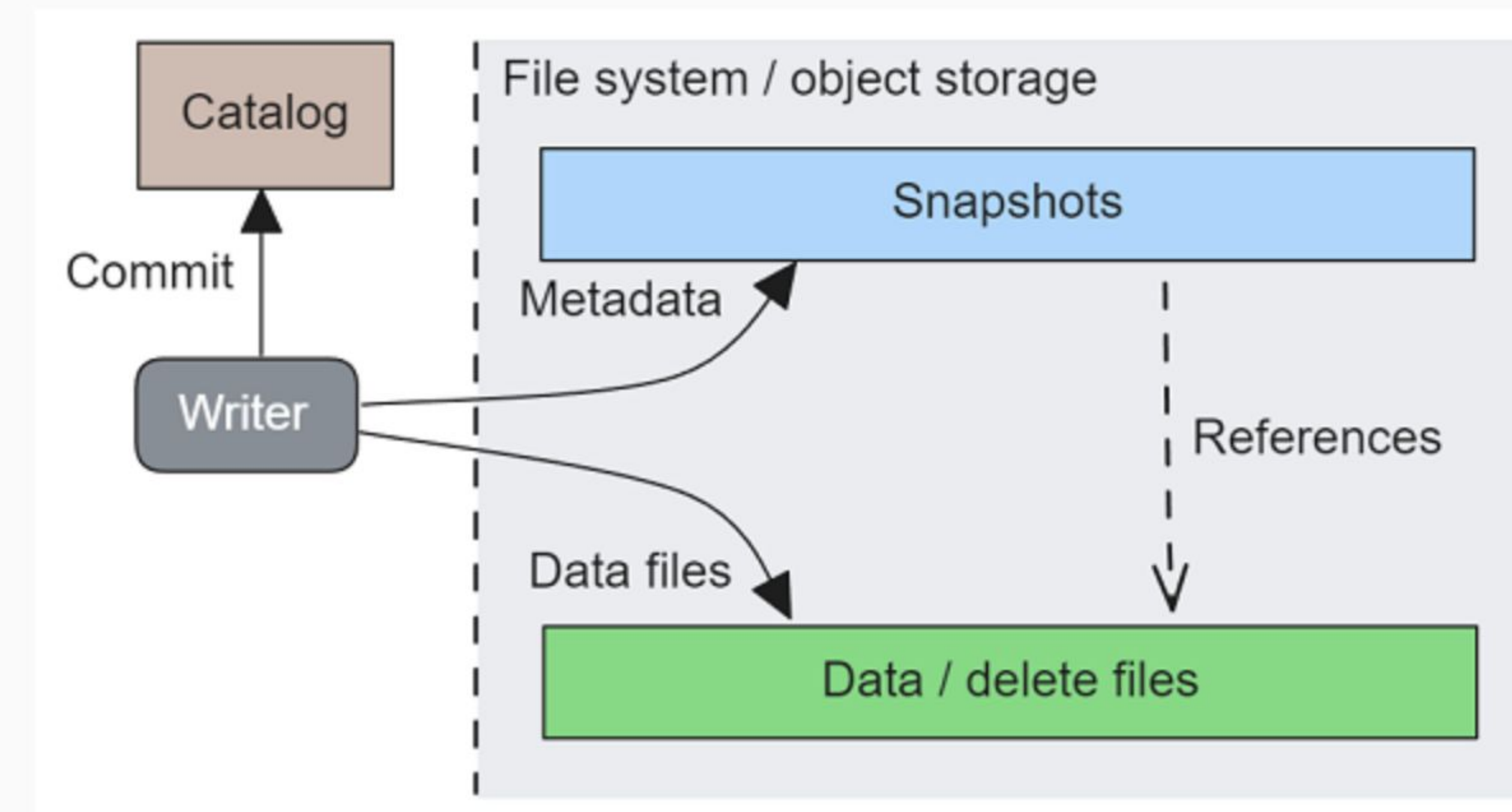
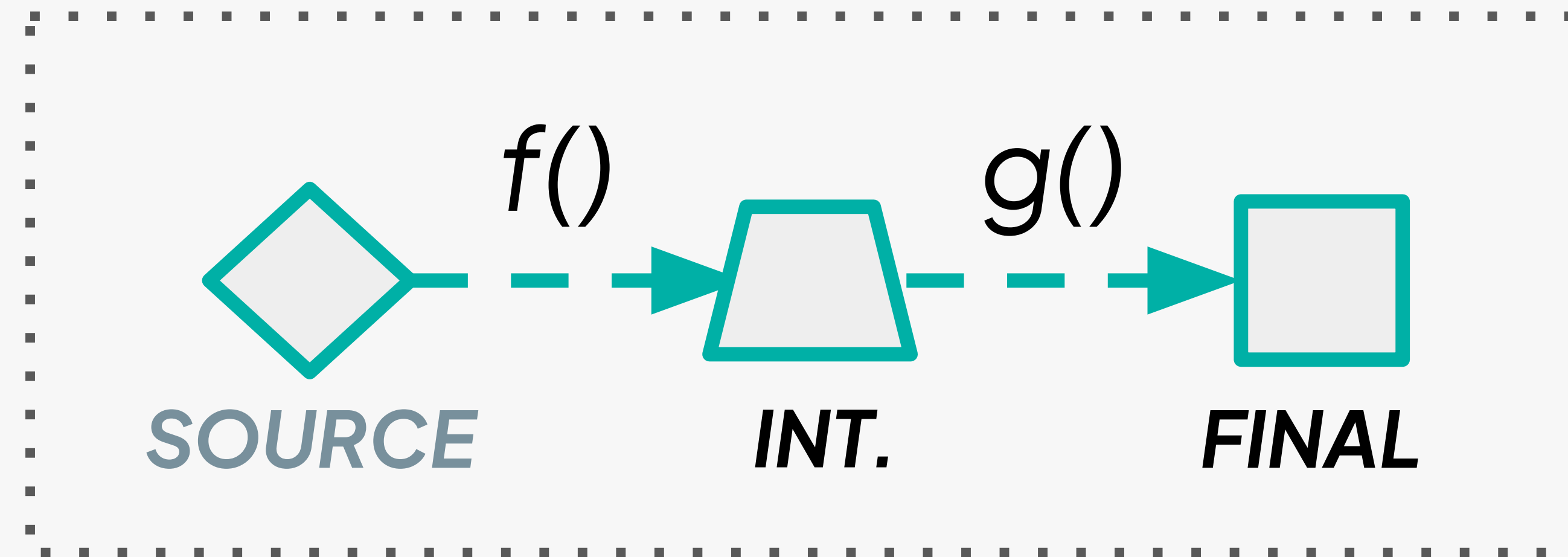


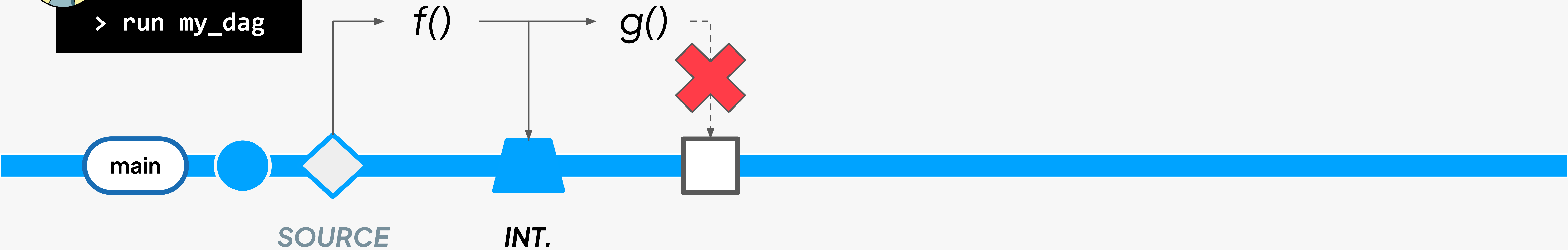
Fig 1. A writer writes metadata that references data files and commits the metadata's location to a catalog.

The worst data pipeline is the one half-done

my_dag (Definition)



> run my_dag



Data abstractions are not enough

CONCURRENCY CONTROL AND RECOVERY IN DATABASE SYSTEMS

- | Vertically integrated (monolithic) databases control both data (storage) and compute (read and writes).
- | Shouldn't **we** also combine data with compute abstractions?

Philip A. Bernstein

Wang Institute of Graduate Studies

Vassos Hadzilacos

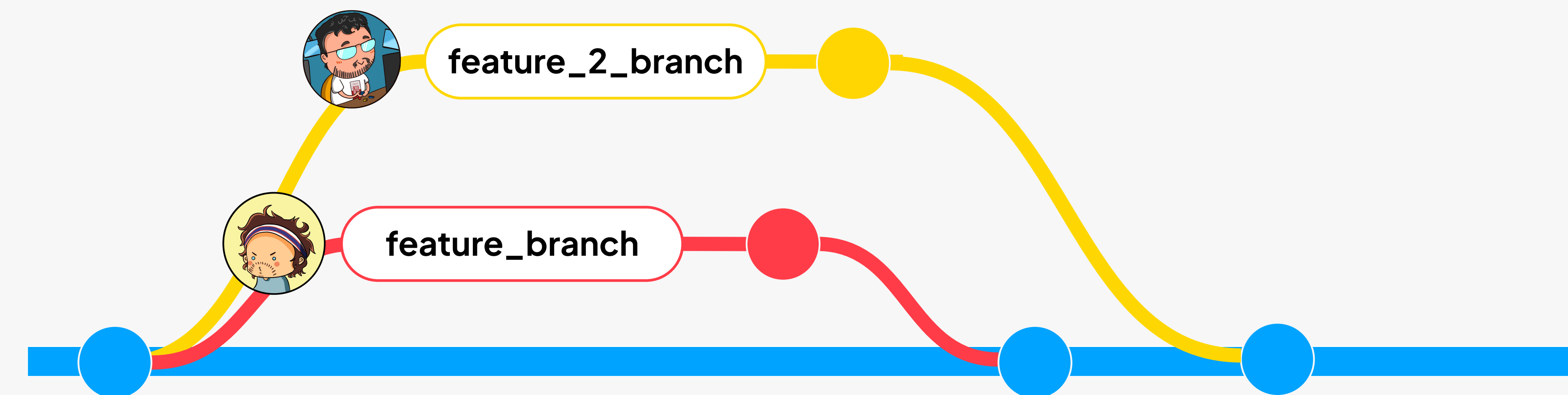
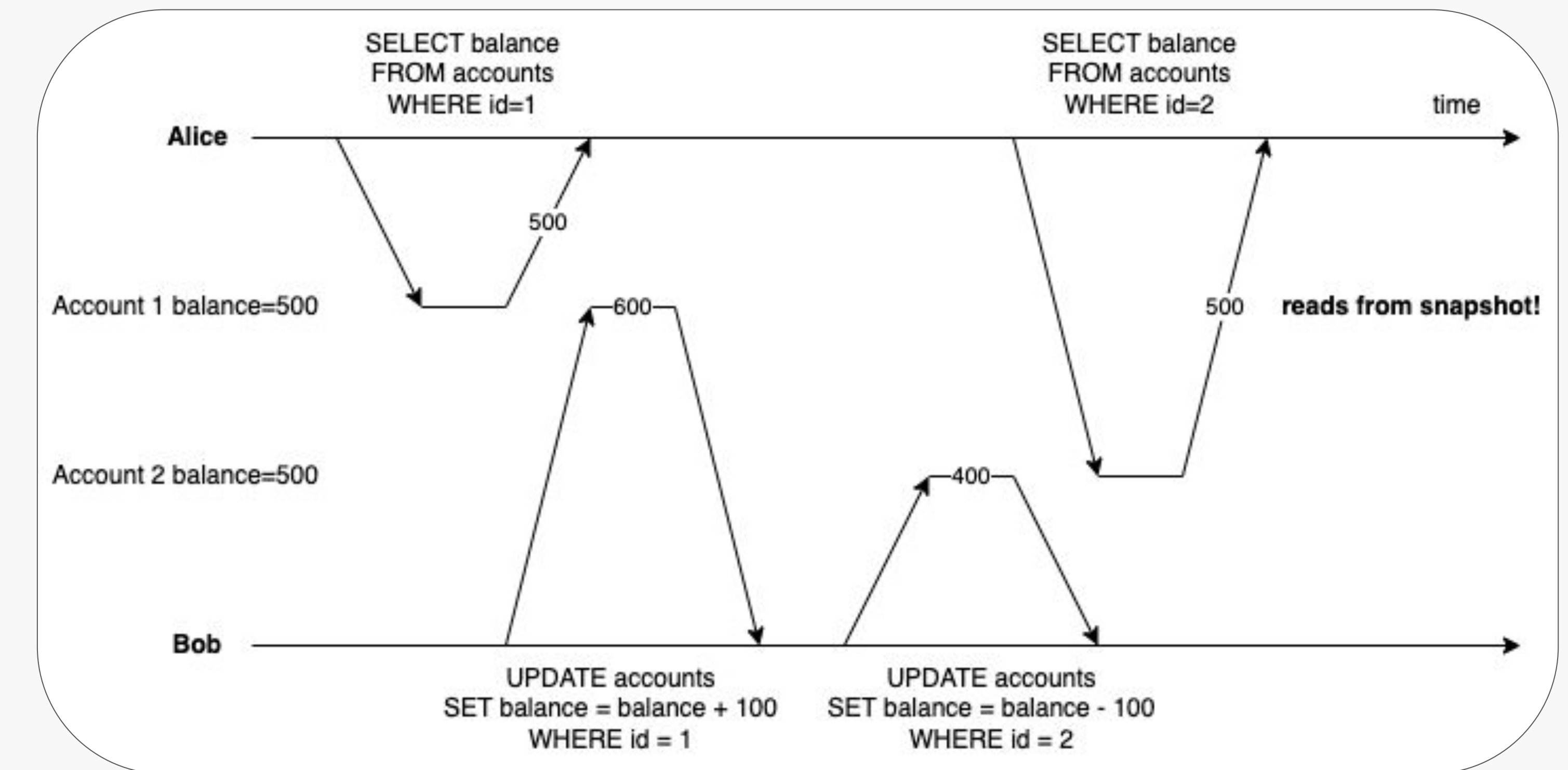
University of Toronto

Nathan Goodman

Kendall Square Research Corporation

Data abstractions are not enough

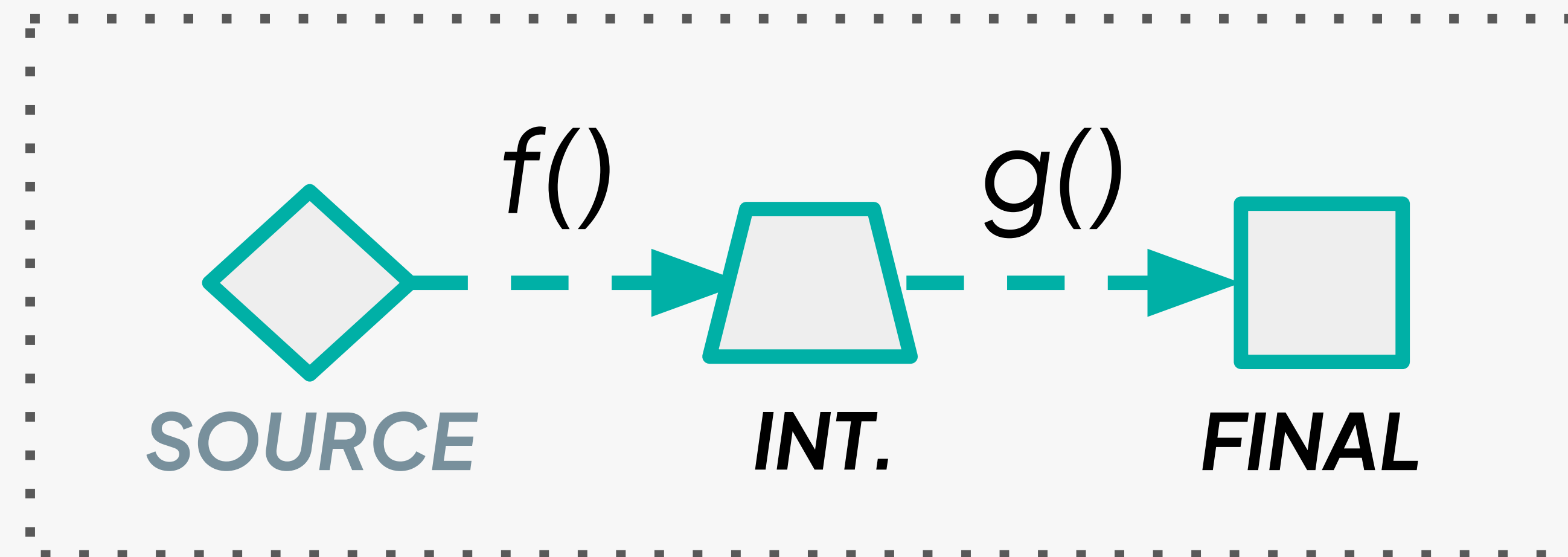
- | Vertically integrated (monolithic) databases control both data (storage) and compute (read and writes).
- | Shouldn't **we** also combine data with compute abstractions?



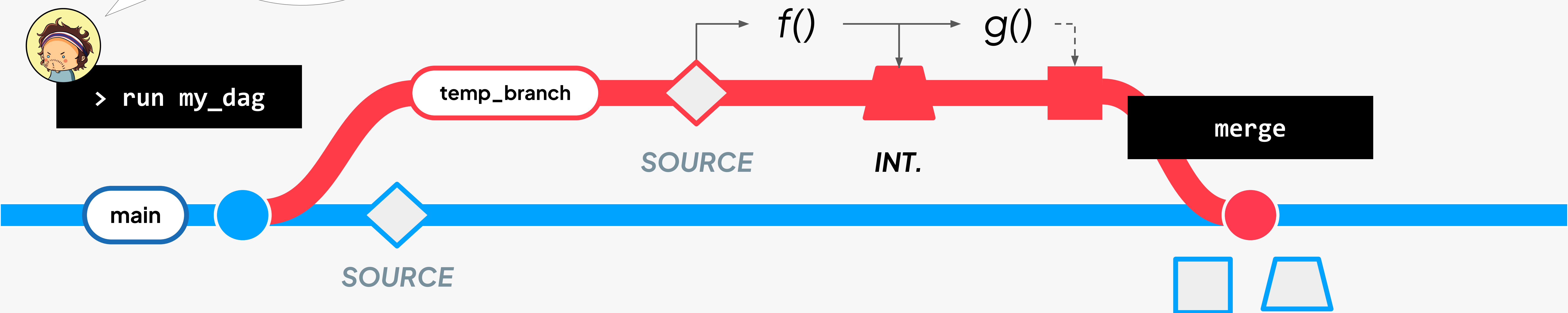
Run = branch + writes

Success:
atomic
updates

my_dag (Definition)



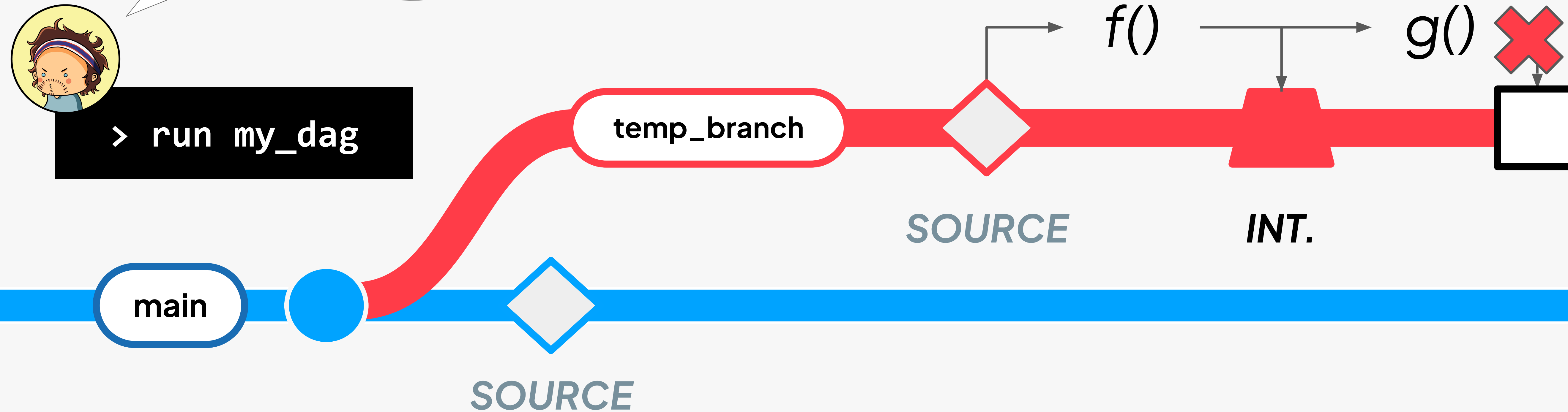
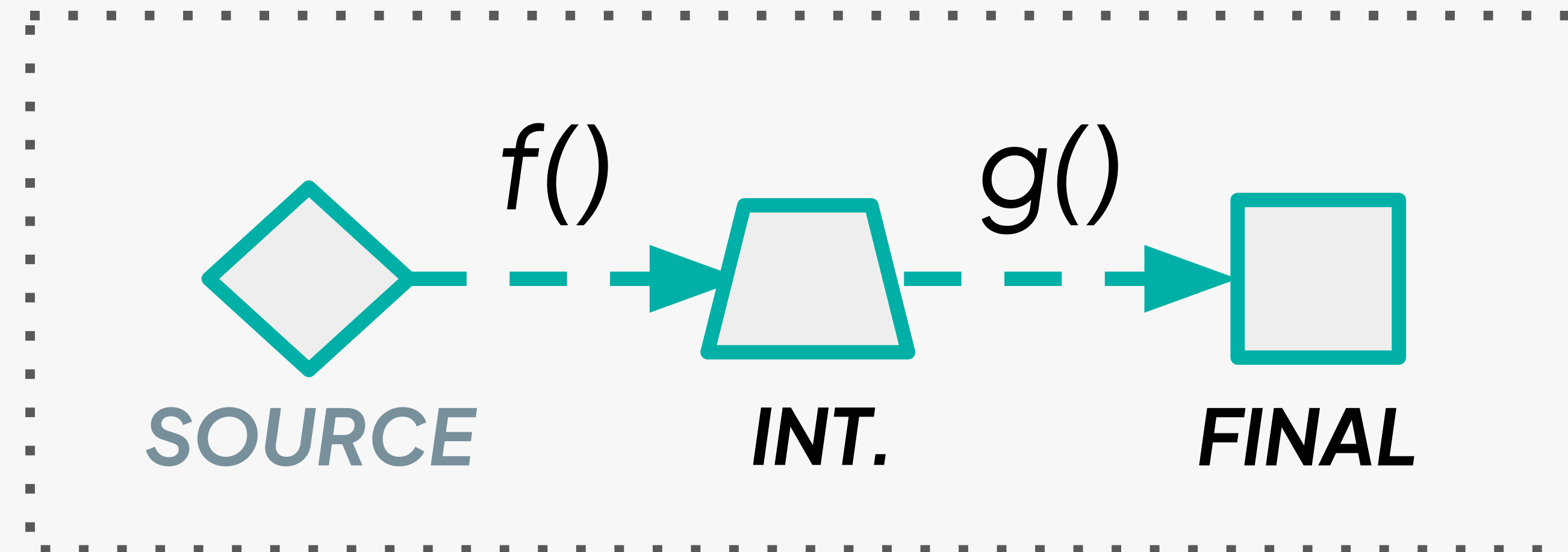
> run my_dag



Run = branch + writes

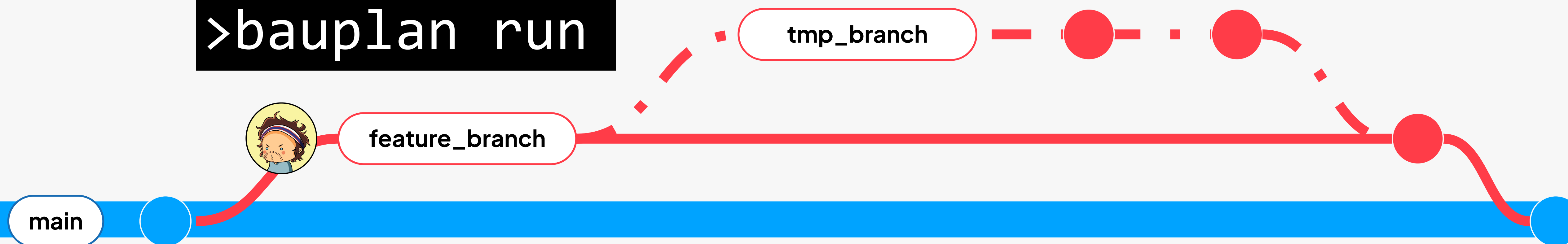
Failure: *main* is untouched

my_dag (Definition)



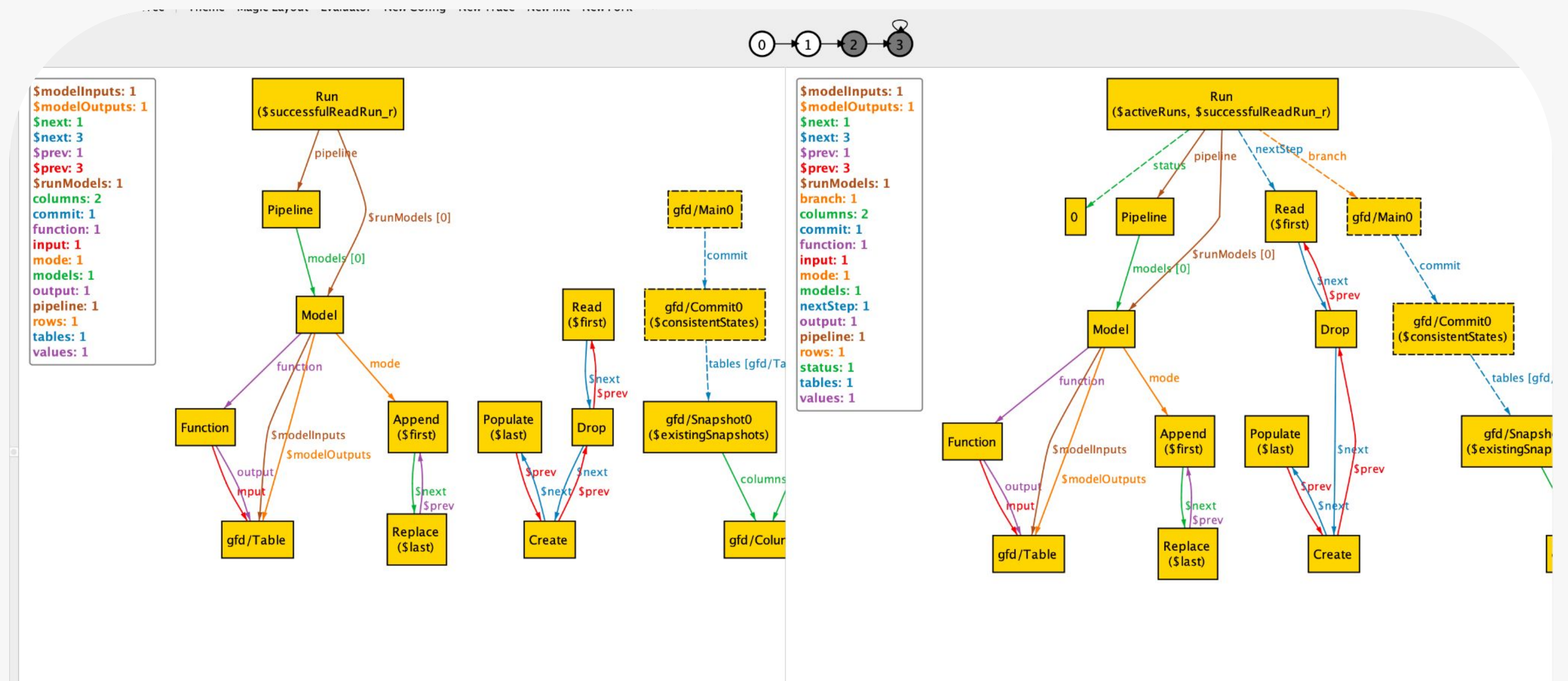
How much “database” is in Git-for-data?

```
>bauplan run
```



“We have discovered a truly marvelous proof of this, which this slide is too narrow to contain”

Lightweight formal models to verify data consistency in the face of failure, and run automated checks as we add new primitives!



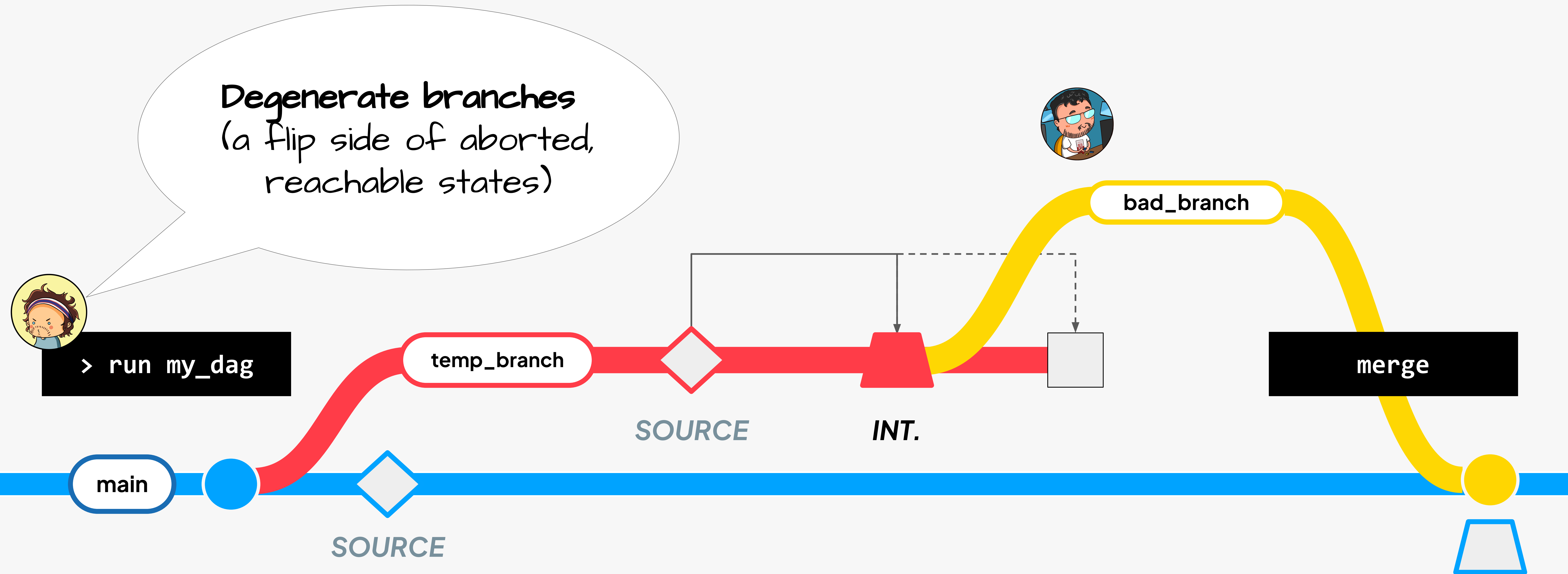
"It is one of the chief merits of proofs that they instill a certain skepticism as to the result proved."

B. Russell

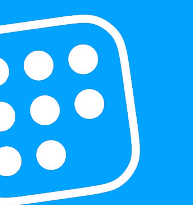


How much “database” is in Git-for-data?

Degenerate branches
(a flip side of aborted,
reachable states)



We barely scratched the surface!



Want to know more?

2023

- [CDMS@VLDB 2023](#)

2024

- [SIGMOD 2024](#)
- [MIDDLEWARE 2024](#) (with UMadison–Wisconsin)
- [BIG DATA 2024](#)

2025

- [UNDER REVIEW 2025](#) (with UMadison–Wisconsin)
- [CDMS@VLDB 2025](#) (with UChicago)

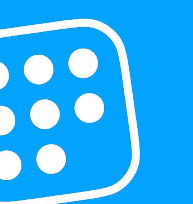


**Shipping at scale *and* speed mostly
means knowing which corners to cut**





Composability accelerate
time-to-market, but requires a
nuanced testing culture



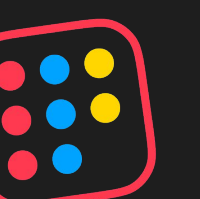


Formal methods are great, but still “too expensive”: AI to the rescue or something else?



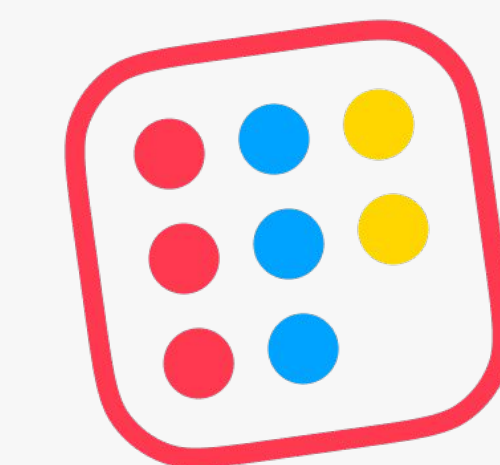
"It is not worth an intelligent man's time to be in the majority. By definition, there are already enough people to do that."

G.H. Hardy



| jacopo.tagliabue@bauplanlabs.com

| We are hiring!



bauplan