



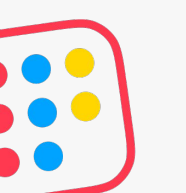
Building Scalable Data Pipelines

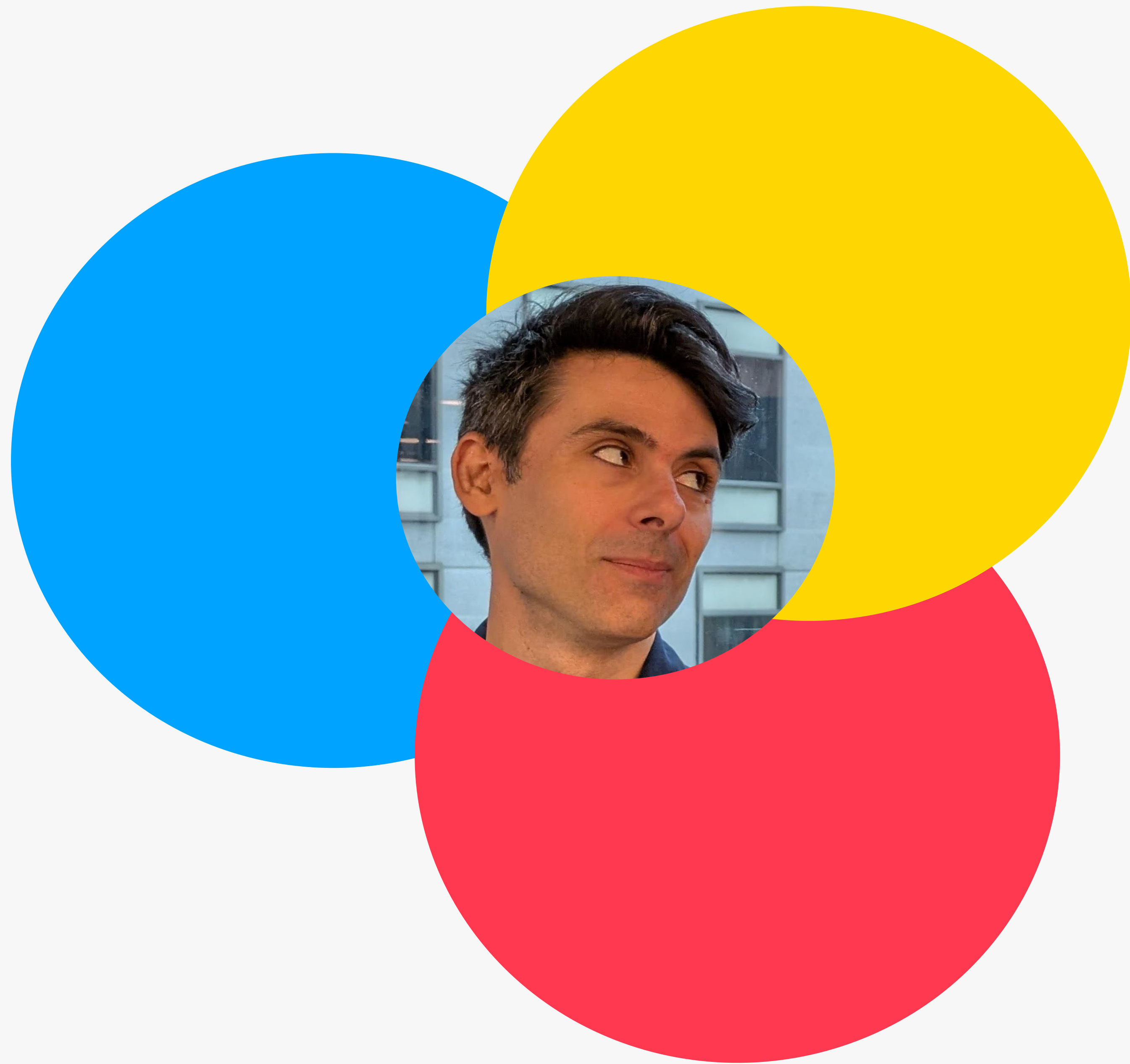
Ciro Greco

Co-founder and CEO
@bauplan

Tuesday, August 19, 11 AM ET

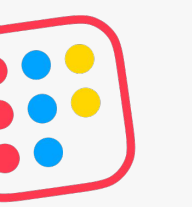
Reliable data pipelines are the backbone of every data-driven organization—but building them to scale is easier said than done. From scheduling and versioning to deploying in production, it takes more than just writing scripts to create pipelines that are robust, maintainable, and ready for growth.

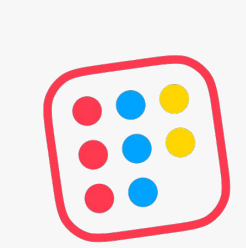




Ciro Greco

- | Co-founder and CEO at **Bauplan**.
- | Previously co-founder at Tooso (acquired by Coveo [TSX:CVO](#)).
- | VP AI at Coveo from scale-up to IPO.
- | Postdoc in computational stuff.

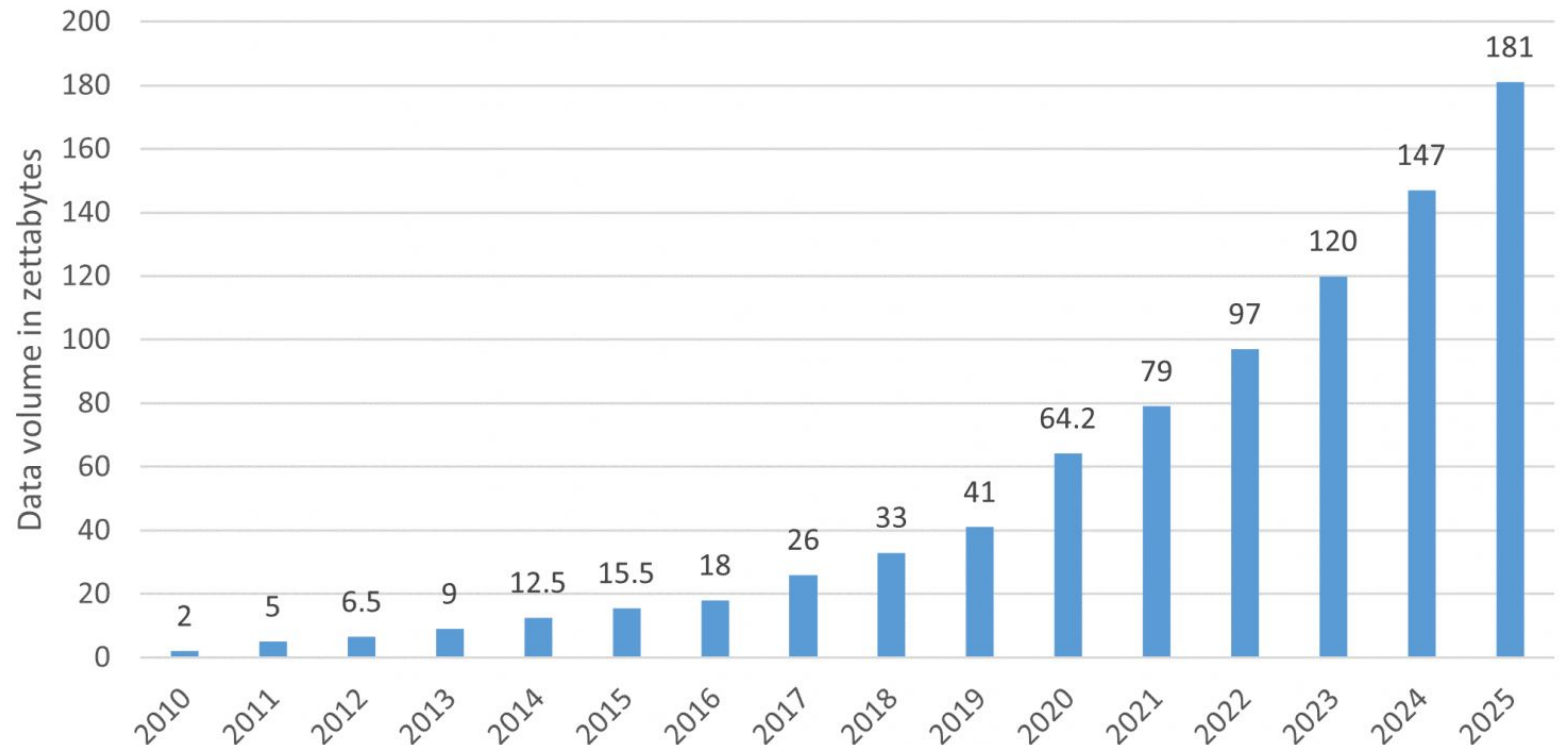




Data is exploding and so are the tools to manage It

- **Data generated in 2025:** 181 ZB (29 TB/sec).*
- **Big Data analytics market:** \$348B today, \$924B by 2032.*
- **Cloud analytics market:** from \$33B in 2023 to \$147B by 2032.**
- **AI infrastructure investment:** \$340B (2025) + \$3T buildout. ***

Volume of data created and replicated worldwide (source: IDC)



* <https://www.demandsage.com/big-data-statistics>

** <https://www.fortunebusinessinsights.com/cloud-analytics-market-102248>

*** <https://www.barrons.com/articles/ai-spending-economy-microsoft-amazon-meta-alphabet-3e2e5fda>

The applications is where you drive the value...

...but the data is actually more important

- As many as 87% of AI projects fail before production—poor data quality is a leading cause.*
- Data engineers spend, on average, 80% of their time fixing and maintaining data pipelines.**

*<https://www.akaike.ai/resources/the-hidden-cost-of-poor-data-quality-why-your-ai-initiative-might-be-set-up-for-failure>

**<https://www.collibra.com/blog/the-7-most-common-data-quality-issues>



jack morris
@jxmnop

one of the most important things I know about deep learning I learned from this paper: "Pretraining Without Attention"

this what I found so surprising:
these people developed an architecture very different from what was called BiGS, spent months and months optimizing it and then they tried different configurations, only to discover that **at the same time, with the same count, a wildly different architecture produces identical results to transformers**

What About the Data? A Mapping Study on Data Engineering for AI Systems

Petra Heck
p.heck@fontys.nl
Fontys University of Applied Sciences
Eindhoven, Netherlands

Google Research

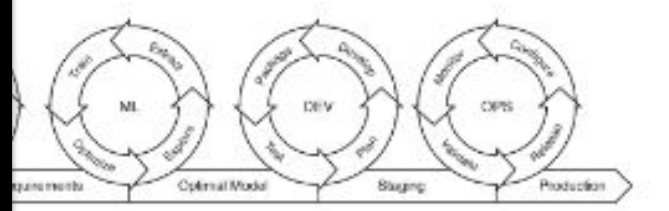
Who we are ▾ Research areas ▾ Our work ▾ Programs & events ▾ Careers Blog

[Publications](#) > "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI

"Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI

Nithya Sambasivan · Shivani Kapania · [Hannah Highfill](#) · [Diana Akron](#) · Praveen Kumar Paritosh · [Lora Moys Aroyo](#) · SIGCHI, ACM (2021)

...eds to be prepared to send it to the model and is. This data can be structured tabular data or such as images, sound, text or video. In 2021, the term "data-centric AI" for "the discipline of engineering the data used to build an AI system"¹, *ring*.



The AI engineering life cycle [12]

...y [36] define data engineering as "the development, and maintenance, of systems and processes that produce high-quality, consistent information for downstream use cases, such as analysis and machine learning. Data engineering is the intersection of security, data science, and software engineering."

FORBES > INNOVATION

AI Needs Data More Than Data Needs AI

 Rohit Sehgal Former Forbes Councils Member
Forbes Technology Council COUNCIL POST | Membership (Fee)

 Oct 5, 2023

 Rohit Sehgal, Founder and CEO of Vincilium.



Data-centric Artificial Intelligence: A Survey

DAOCHEN ZHA, Rice University, United States
ZAID PERVAIZ BHAT, Texas A&M University, United States
KWEI-HERNG LAI, Rice University, United States
FAN YANG, Rice University, United States
ZHIMENG JIANG, Texas A&M University, United States
SHAOCHEN ZHONG, Rice University, United States
XIA HU, Rice University, United States

Artificial Intelligence (AI) is making a profound impact in almost every domain. A vital enabler of its great success is the availability of abundant and high-quality data for building machine learning models. Recently, the role of data in AI has been significantly magnified, giving rise to the emerging concept of *data-centric AI*. The attention of researchers and practitioners has gradually shifted from advancing model design to enhancing the quality and quantity of the data. In this survey, we discuss the necessity of data-centric AI.

11 Jun 2023

Today

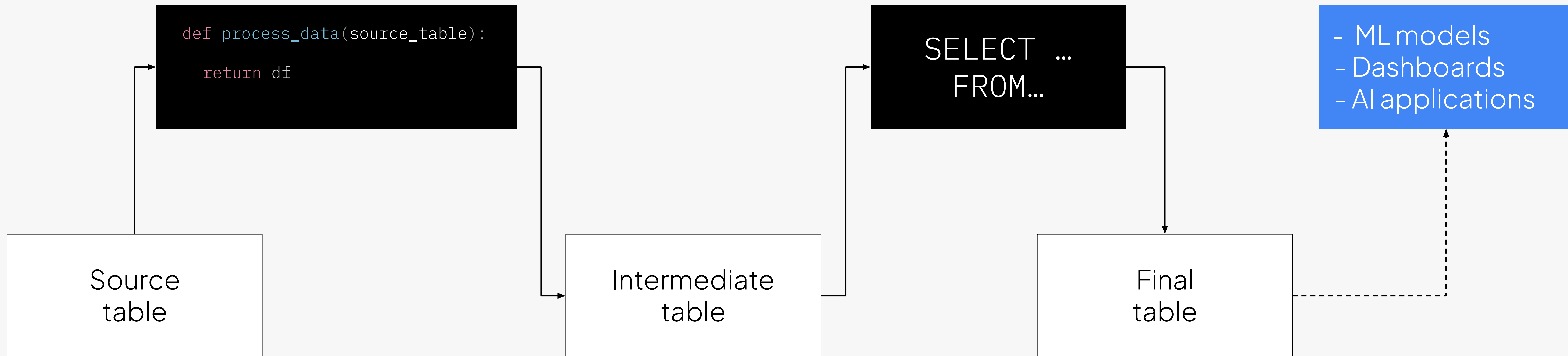
1. From raw data to reliable pipelines
2. Inside the data lakehouse
3. Live walk-through of a pipeline
4. Q&A

1. Data pipelines

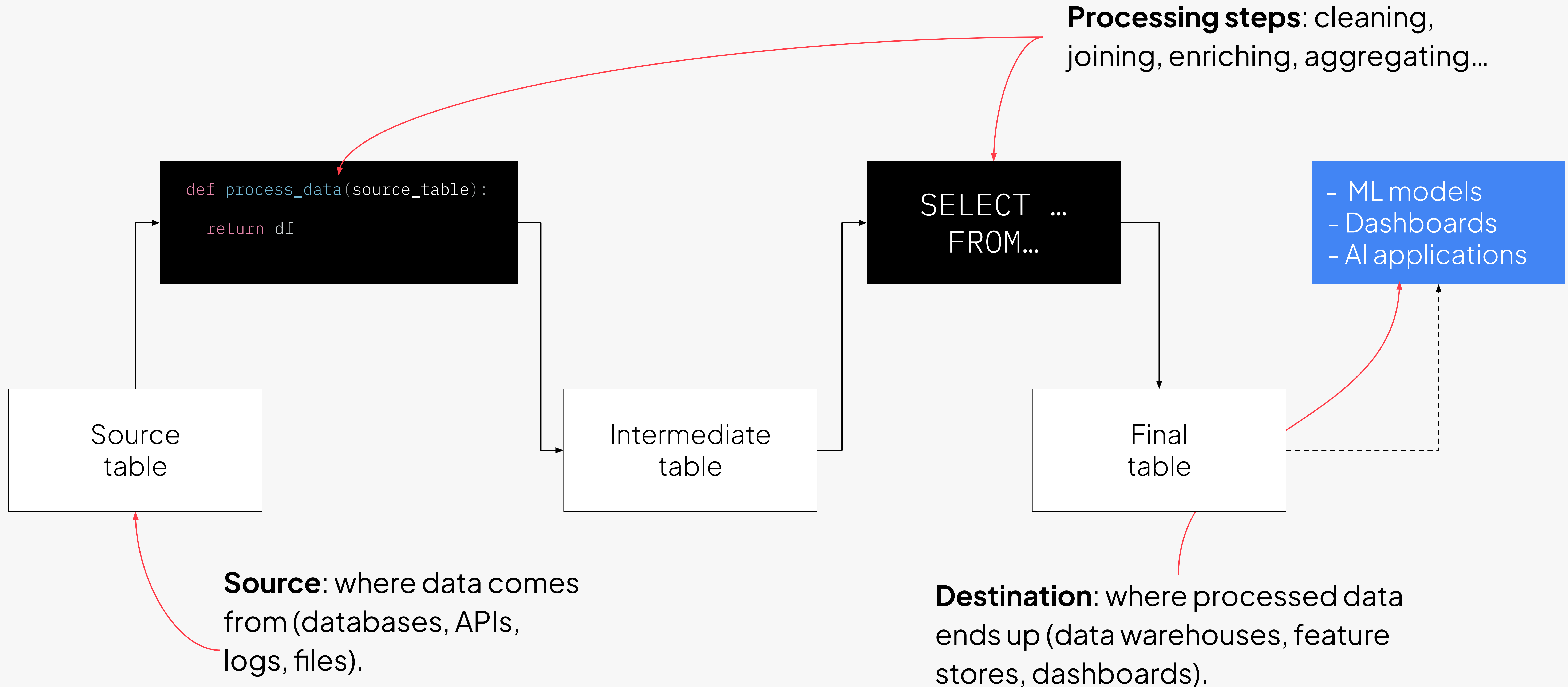


Data pipelines

A data pipeline is a sequence of processes that move, transform, and prepare data so it can be used by applications, analytics, or machine learning models.

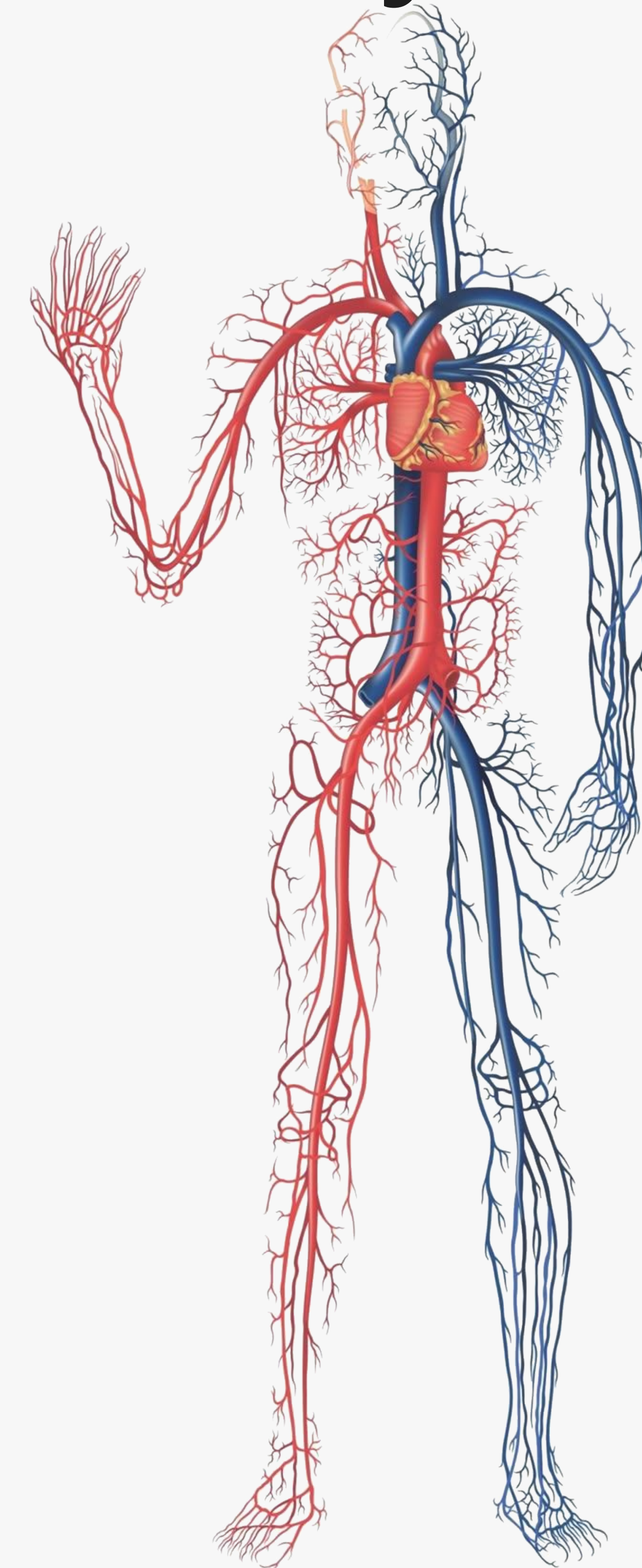


Core elements of a data pipeline

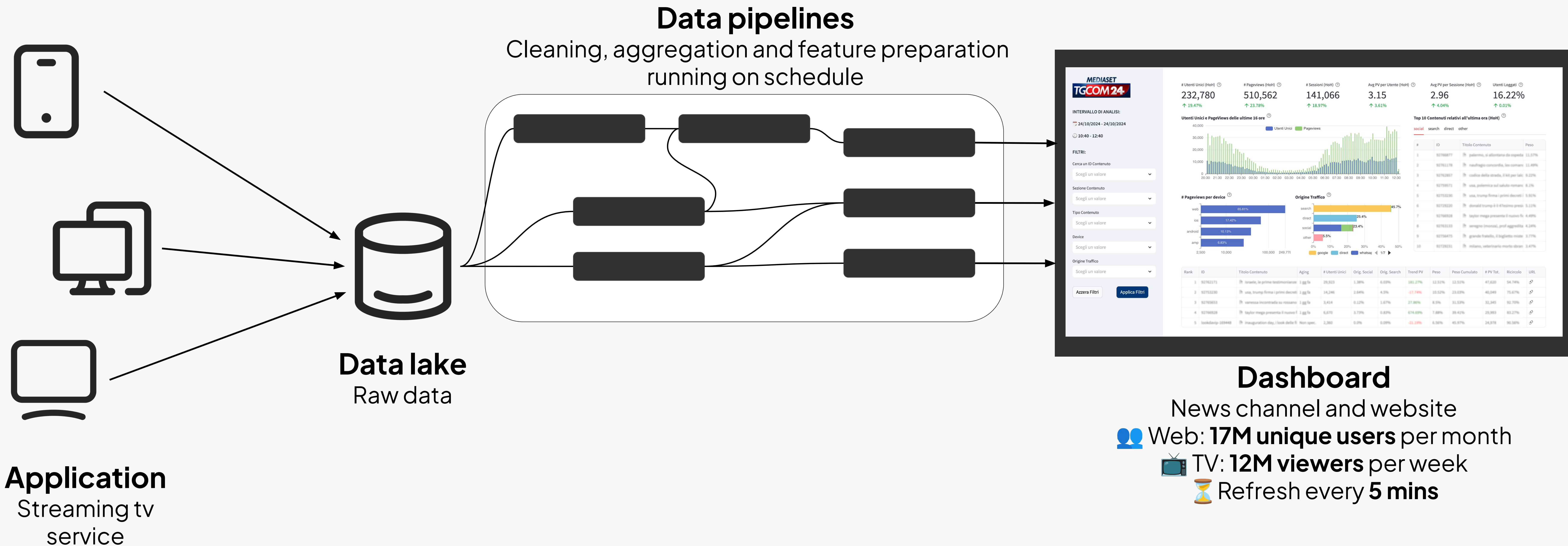


Data pipelines are the circulatory system of your data

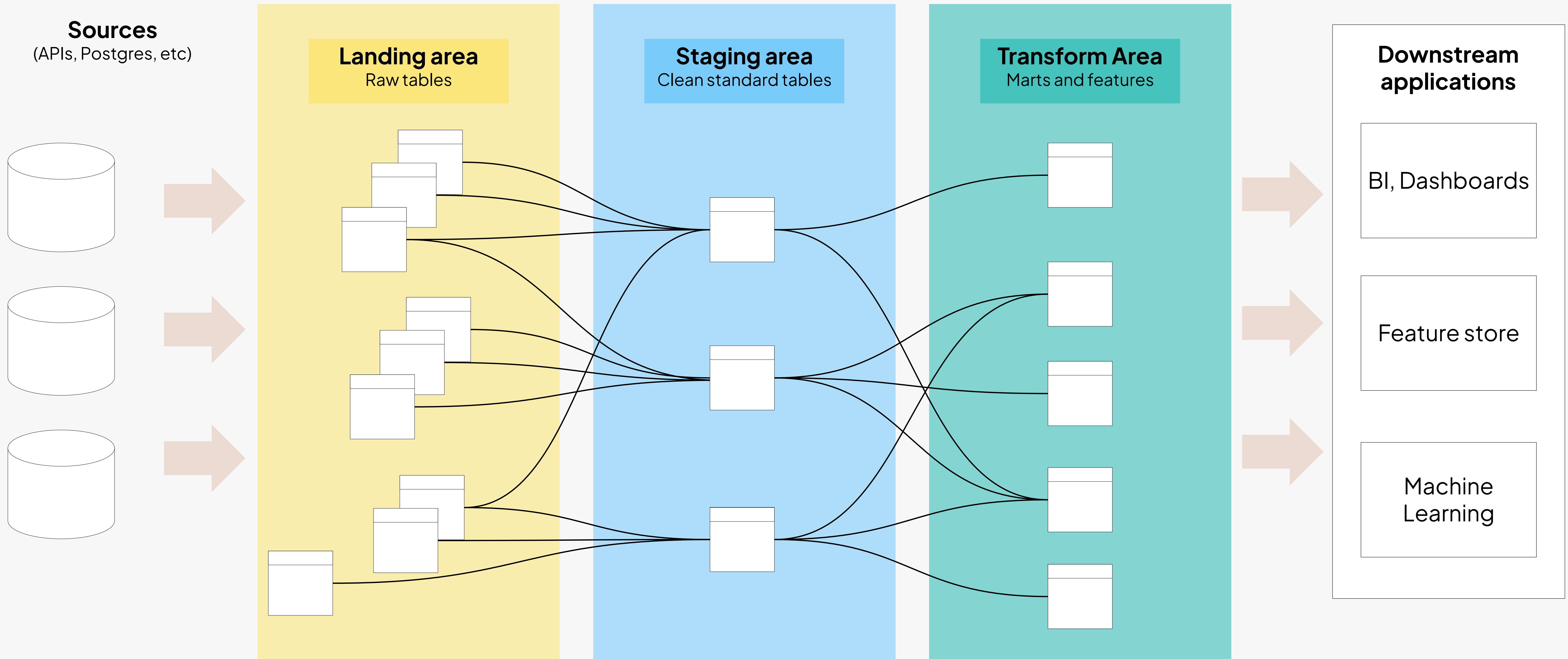
- **Move** and **transform** data from where it's generated to where it's needed.
- **Validate** and **standardize** data so it's accurate, consistent, and usable.
- **Automate** the flow so data is always **available** in the right form at the right time.
- Enable faster decisions and reliable applications by keeping data **fresh** and **dependable**.



Real world example: Dashboard for TGcom24



Landing, staging, transform architecture



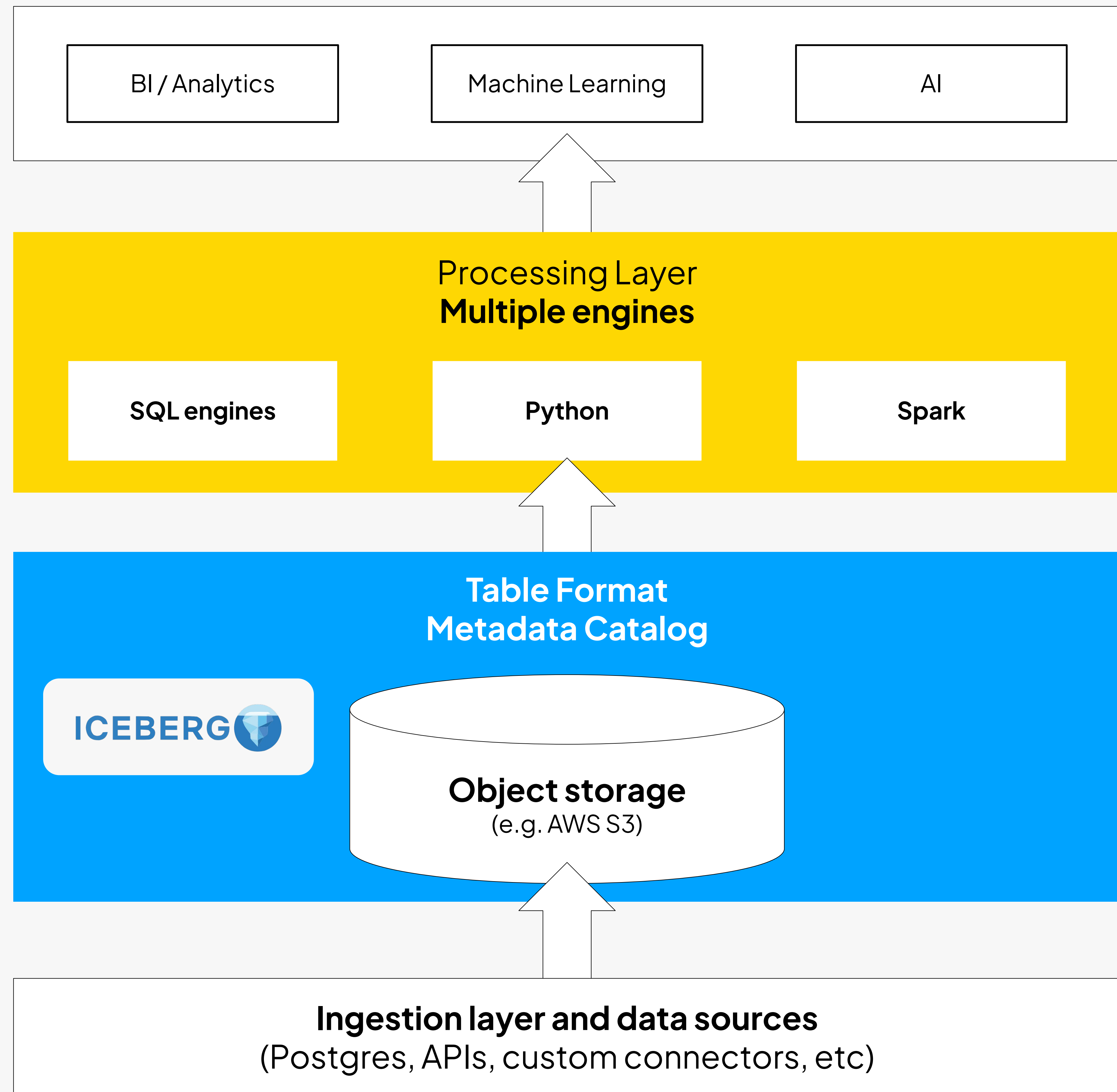
2. Data infrastructure



The data lakehouse

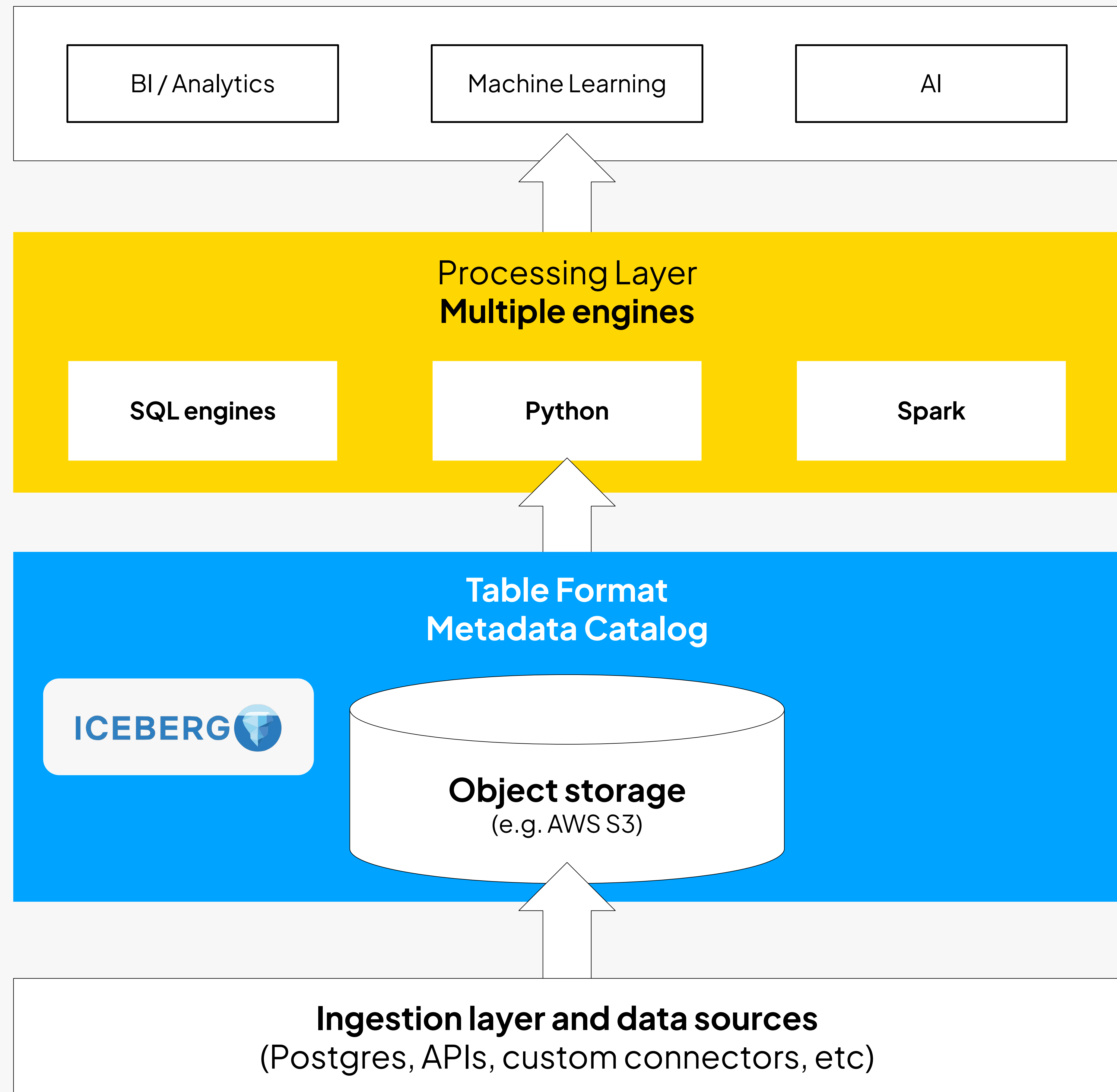
The scalability of a **data lake** + the management features of a **warehouse**

- Store data in **open formats** (Iceberg) on **object storage**.
- Supports **multiple compute** engines for **different use cases**.



Advantages

- **Separation of storage and compute:** store once, scale compute as needed.
- **Unification, not silos:** same data powers analytics, pipelines, ML models.
- **Tables, not files:** get ACID guarantees, versions and schema evolution.
- **Open:** use Iceberg to speak with all kinds of engines.



Still pipelines remain hard



Fragmentation across the stack

1. Runtime problems

- Environment drift.
- Scaling pain.
- Cold starts & resource contention.
- Debugging is painful.

2. Storage and catalog problems

- Inconsistent state.
- Lack of data versioning.
- Schema drift.

3. Orchestration problems

- Waterfall errors.
- Over-engineered DAGs.
- Coupling logic to orchestration

```
at org.apache.spark.deploy.yarn.ApplicationMaster$.main(ApplicationMaster.scala:912)
at org.apache.spark.deploy.yarn.ExecutorLauncher$.main(ApplicationMaster.scala:944)
at org.apache.spark.deploy.yarn.ExecutorLauncher.main(ApplicationMaster.scala)
Caused by: java.io.IOException: Failed to connect to emr-header-1.cluster-...:38803
at org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory.java:288)
at org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory.java:218)
at org.apache.spark.network.client.TransportClientFactory.createClient(TransportClientFactory.java:230)
at org.apache.spark.rpc.netty.NettyRpcEnv.createClient(NettyRpcEnv.scala:204)
at org.apache.spark.rpc.netty.Outbox$$anon$1.call(Outbox.scala:202)
at org.apache.spark.rpc.netty.Outbox$$anon$1.call(Outbox.scala:198)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
at java.lang.Thread.run(Thread.java:748)
Caused by: java.net.UnknownHostException: emr-header-1.cluster-...
at java.net.InetAddress.getAllByName0(InetAddress.java:1281)
at java.net.InetAddress.getAllByName(InetAddress.java:1193)
at java.net.InetAddress.getAllByName(InetAddress.java:1127)
at java.net.InetAddress.getByName(InetAddress.java:1077)
at io.netty.util.internal.SocketUtils$8.run(SocketUtils.java:156)
at io.netty.util.internal.SocketUtils$8.run(SocketUtils.java:153)
at java.security.AccessController.doPrivileged(Native Method)
at io.netty.util.internal.SocketUtils.addressByName(SocketUtils.java:153)
at io.netty.resolver.DefaultNameResolver.doResolve(DefaultNameResolver.java:41)
at io.netty.resolver.SimpleNameResolver.resolve(SimpleNameResolver.java:61)
at io.netty.resolver.SimpleNameResolver.resolve(SimpleNameResolver.java:53)
at io.netty.resolver.InetSocketAddressResolver.doResolve(InetSocketAddressResolver.java:55)
at io.netty.resolver.InetSocketAddressResolver.doResolve(InetSocketAddressResolver.java:31)
at io.netty.resolver.AbstractAddressResolver.resolve(AbstractAddressResolver.java:106)
at io.netty.bootstrap.Bootstrap.doResolveAndConnect0(Bootstrap.java:206)
at io.netty.bootstrap.Bootstrap.access$000(Bootstrap.java:46)
at io.netty.bootstrap.Bootstrap$1.operationComplete(Bootstrap.java:180)
at io.netty.bootstrap.Bootstrap$1.operationComplete(Bootstrap.java:166)
at io.netty.util.concurrent.DefaultPromise.notifyListener0(DefaultPromise.java:578)
at io.netty.util.concurrent.DefaultPromise.notifyListenersNow(DefaultPromise.java:552)
at io.netty.util.concurrent.DefaultPromise.notifyListeners(DefaultPromise.java:491)
at io.netty.util.concurrent.DefaultPromise.setValue0(DefaultPromise.java:616)
at io.netty.util.concurrent.DefaultPromise.setSuccess0(DefaultPromise.java:605)
at io.netty.util.concurrent.DefaultPromise.trySuccess(DefaultPromise.java:104)
at io.netty.channel.DefaultChannelPromise.trySuccess(DefaultChannelPromise.java:84)
at io.netty.channel.AbstractChannel$AbstractUnsafe.safeSetSuccess(AbstractChannel.java:1008)
at io.netty.channel.AbstractChannel$AbstractUnsafe.register0(AbstractChannel.java:516)
at io.netty.channel.AbstractChannel$AbstractUnsafe.access$200(AbstractChannel.java:429)
at io.netty.channel.AbstractChannel$AbstractUnsafe$1.run(AbstractChannel.java:486)
at io.netty.util.concurrent.AbstractEventExecutor.safeExecute(AbstractEventExecutor.java:164)
at io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(SingleThreadEventExecutor.java:469)
at io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:500)
at io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:986)
at io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
... 1 more
```





4:17:01 (!?!)

YouTube

Search

+

Create

i

1TB

Driver program

SparkContext

Cluster Manager

Worker Node

Executor

Cache

Task

Task

Worker Node

Executor

Cache

Task

Task

W1

W2

19:38 / 4:17:01

Ansh Lamba

46.6K subscribers

Join

Subscribe

4.8K

Share

Download

All

From Ansh Lamba

Big Data

Cloud cor

>

niniaOne

All In One RMM Solution

EnglishContact usSupportMy account

 Agentic AI | Discover AWS | Products | Solutions | Pricing | Resources

SearchSign in to consoleCreate Account

AWS BlogsHomeBlogsEditions

AWS Big Data Blog

Accelerate lightweight analytics using Pylceberg with AWS Lambda and an AWS Glue Iceberg REST endpoint

by Sotaro Hikita and Shuhei Fukami | on 09 MAY 2025 | in Analytics, AWS Big Data, AWS Glue, AWS Lambda | Permalink

Comments | Share

For modern organizations built on data insights, effective data management is crucial for powering advanced analytics and machine learning (ML) activities. As data use cases become more complex, data engineering teams require sophisticated tooling to handle versioning, increasing data volumes, and schema changes across multiple data sources and applications.

Apache Iceberg has emerged as a popular choice for data lakes, offering ACID (Atomicity, Consistency, Isolation, Durability) transactions, schema evolution, and time travel capabilities. Iceberg tables can be accessed from various distributed data processing frameworks like Apache Spark and Trino, making it a flexible solution for diverse data processing needs. Among the available tools for working with Iceberg, [Pylceberg](#) stands out as a Python implementation that enables table access and management without requiring distributed compute resources.

In this post, we demonstrate how Pylceberg, integrated with the [AWS Glue Data Catalog](#) and [AWS Lambda](#), provides a lightweight approach to harness Iceberg’s powerful features through intuitive Python interfaces. We show how this integration enables teams to start working with Iceberg tables with minimal setup and infrastructure dependencies.

Pylceberg's key capabilities and advantages

One of Pylceberg's primary advantages is its lightweight nature. Without requiring distributed computing frameworks, teams can perform table operations directly from Python applications, making it suitable for small to medium-scale data exploration and analysis with minimal learning curve. In addition, Pylceberg is integrated with Python data analysis libraries like Pandas and Polars, so data users can use their existing skills and workflows.

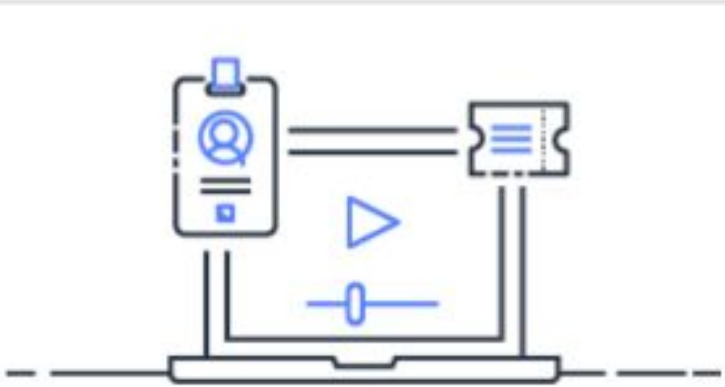
When using Pylceberg with the Data Catalog and [Amazon Simple Storage Service](#) (Amazon S3), data teams can store and manage their tables in a completely serverless environment. This means data teams can focus on analysis and insights rather than infrastructure management.

Resources

[Amazon Athena](#)[Amazon EMR](#)[Amazon Kinesis](#)[Amazon MSK](#)[Amazon QuickSight](#)[Amazon Redshift](#)[AWS Glue](#)

Follow

[Twitter](#)[Facebook](#)[LinkedIn](#)[Twitch](#)[Email Updates](#)



AWS Events

Amazon EventBridge is a fully managed event bus that enables you to connect and orchestrate multiple AWS services and applications. It allows you to create event rules that filter and route events to event targets.

17 pages!!!

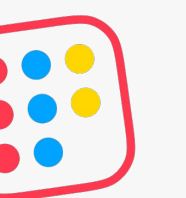
Today

Landing – Staging – Transform

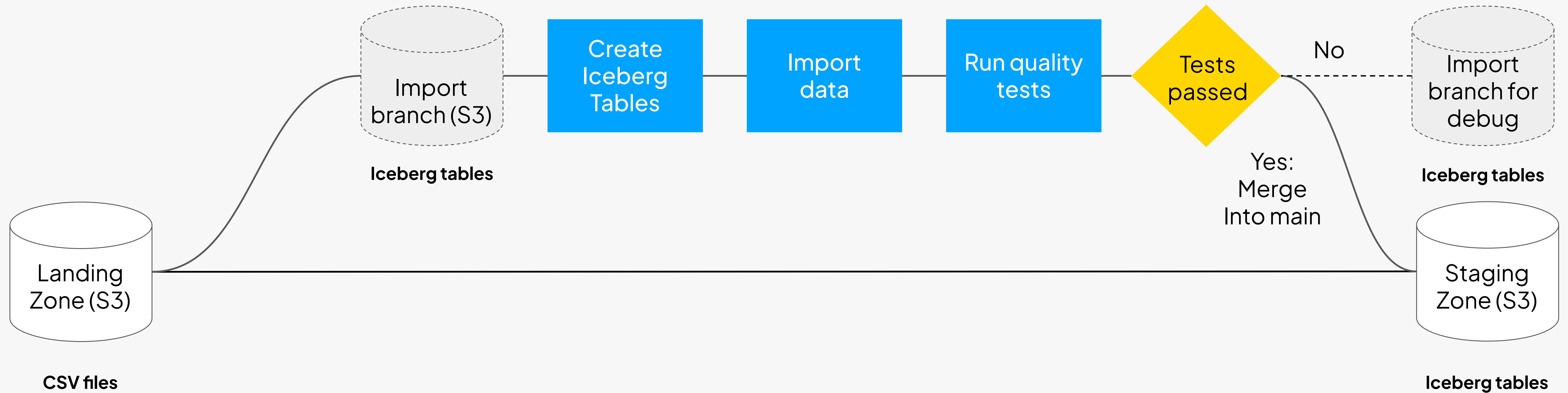


Bauplan is a lakehouse platform for engineering teams who treat data like software.

- No infrastructure
- Just Python and SQL
- Like Git for data
- Built on Apache Iceberg

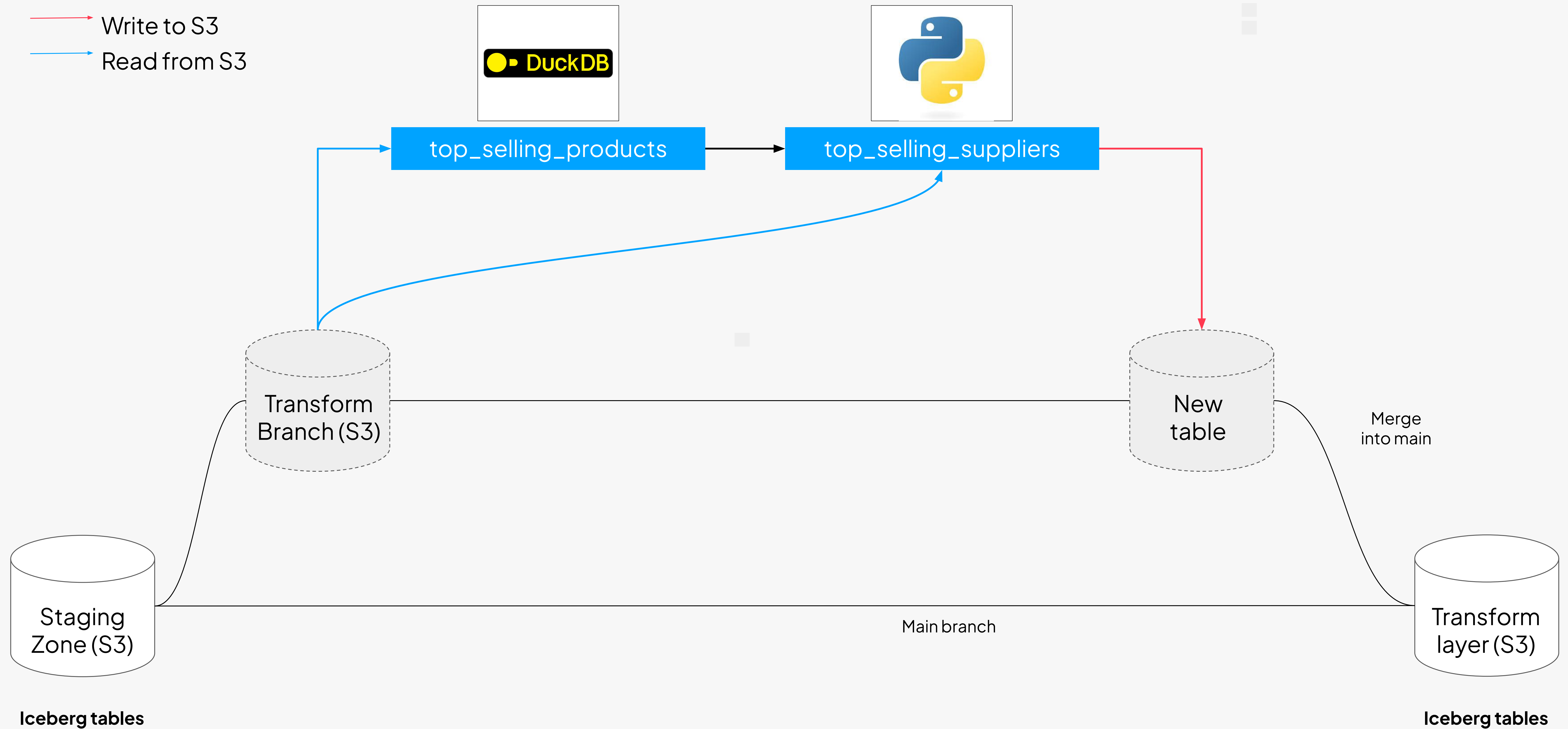


From Landing to Staging



From Staging to Transform

→ Write to S3
→ Read from S3



Data pipelines done well

- **Version your data.** Track changes to data, code, and config together for reproducibility.
- **Isolate workloads.** Develop and run changes in separate environments before merging to production.
- **Test data quality.** Validate schemas, values, and expectations before publishing.
- **Minimize infrastructure.** Fewer moving parts means lower cost, easier ops, and less to break.



