bauplan

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.

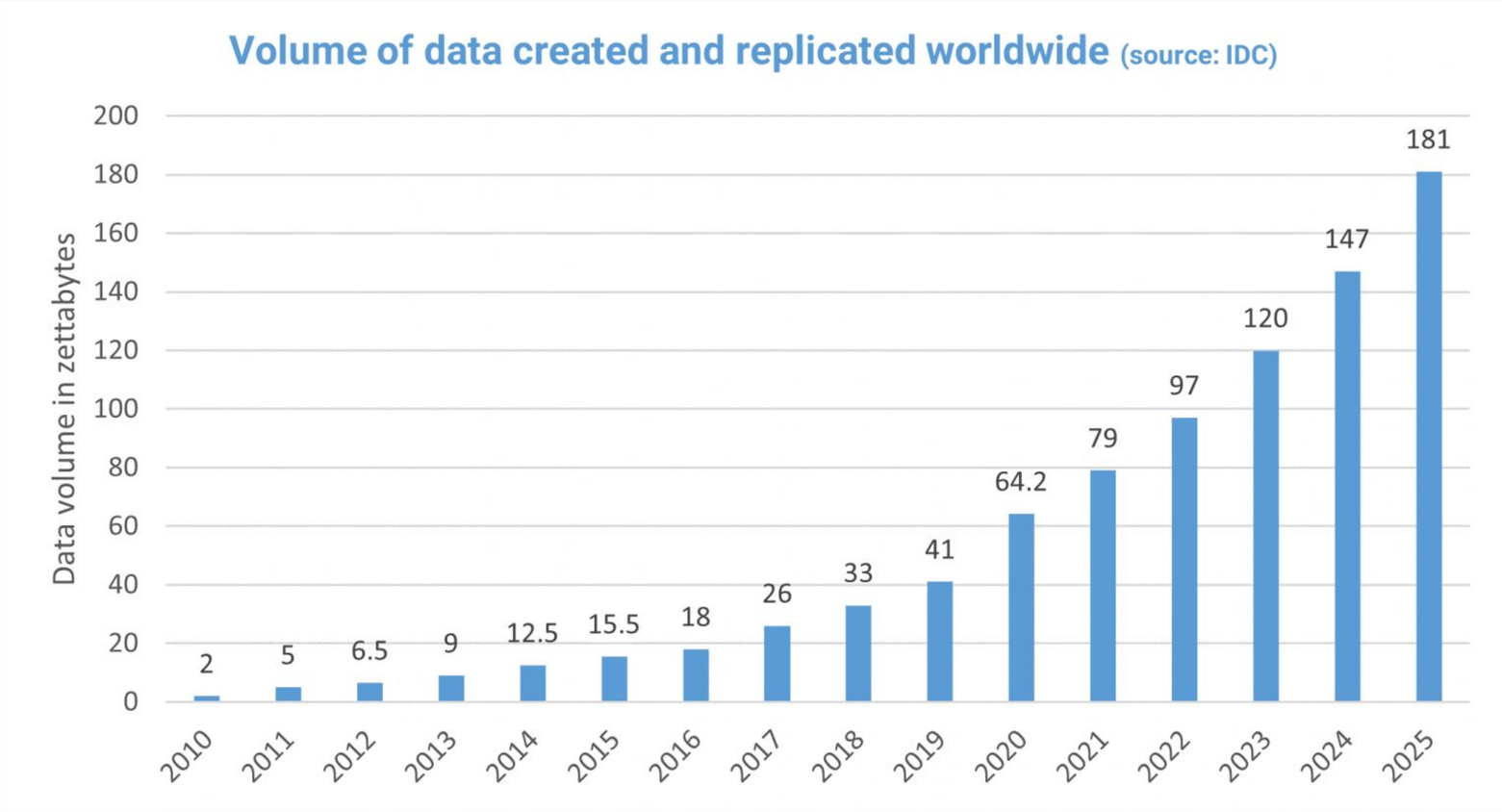
datacamp

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. ***

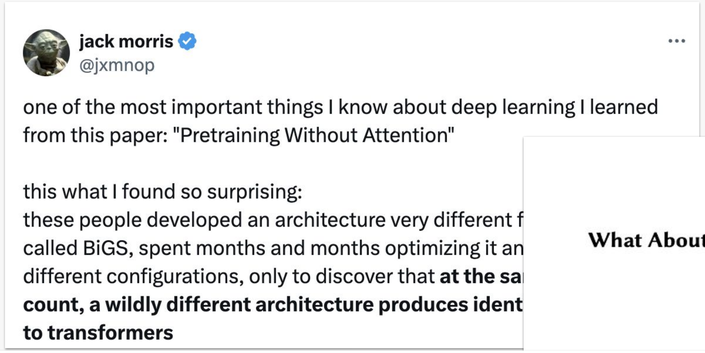


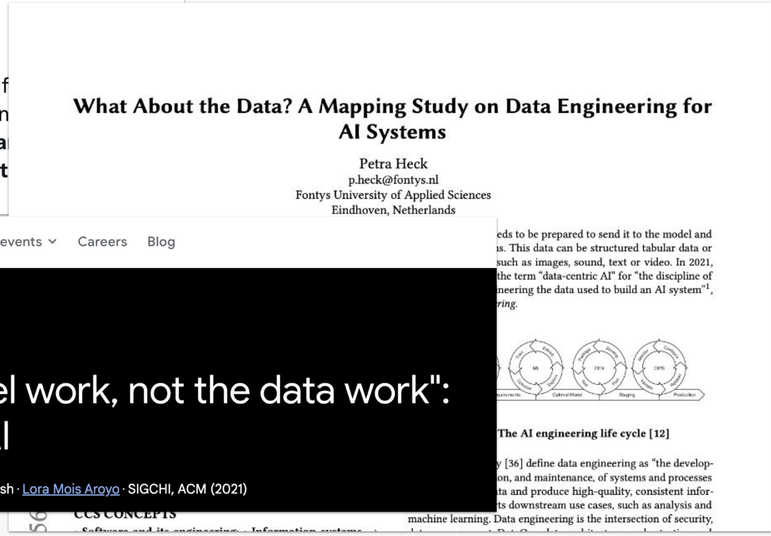
* https://www.demandsage.com/big-data-statistics
** https://www.fortunebusinessinsights.com/cloud-analytics-market-102248
***https://www.barrons.com/articles/ai-spending-economy-microsoft-amazon-meta-alphabet-3e2e5fda

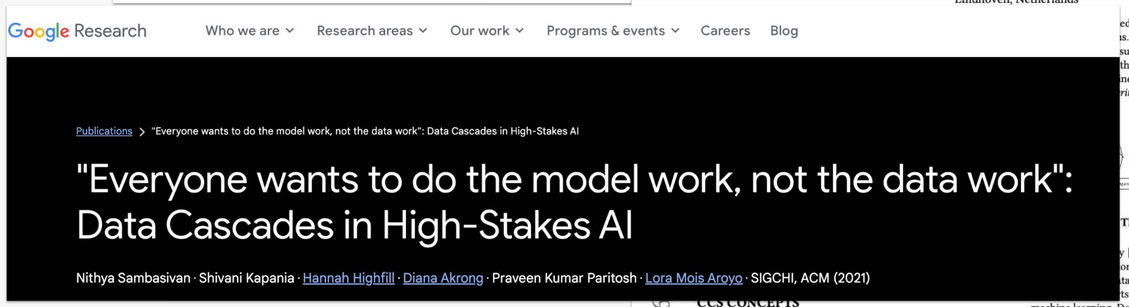
The applications is where you drive the value…

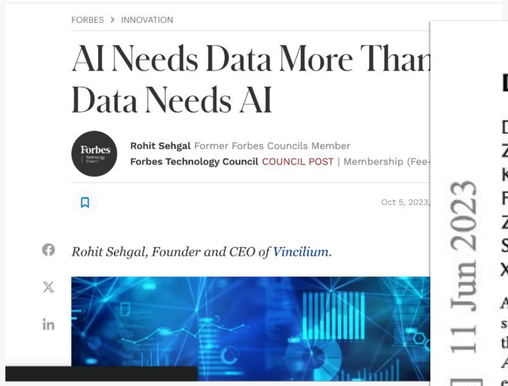
…but the data is actually more important

- 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

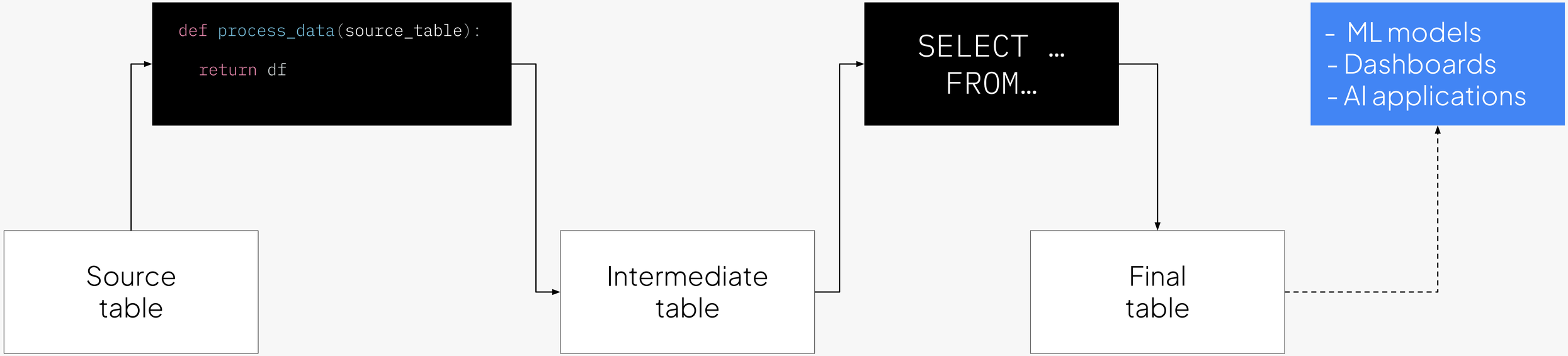
Today

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

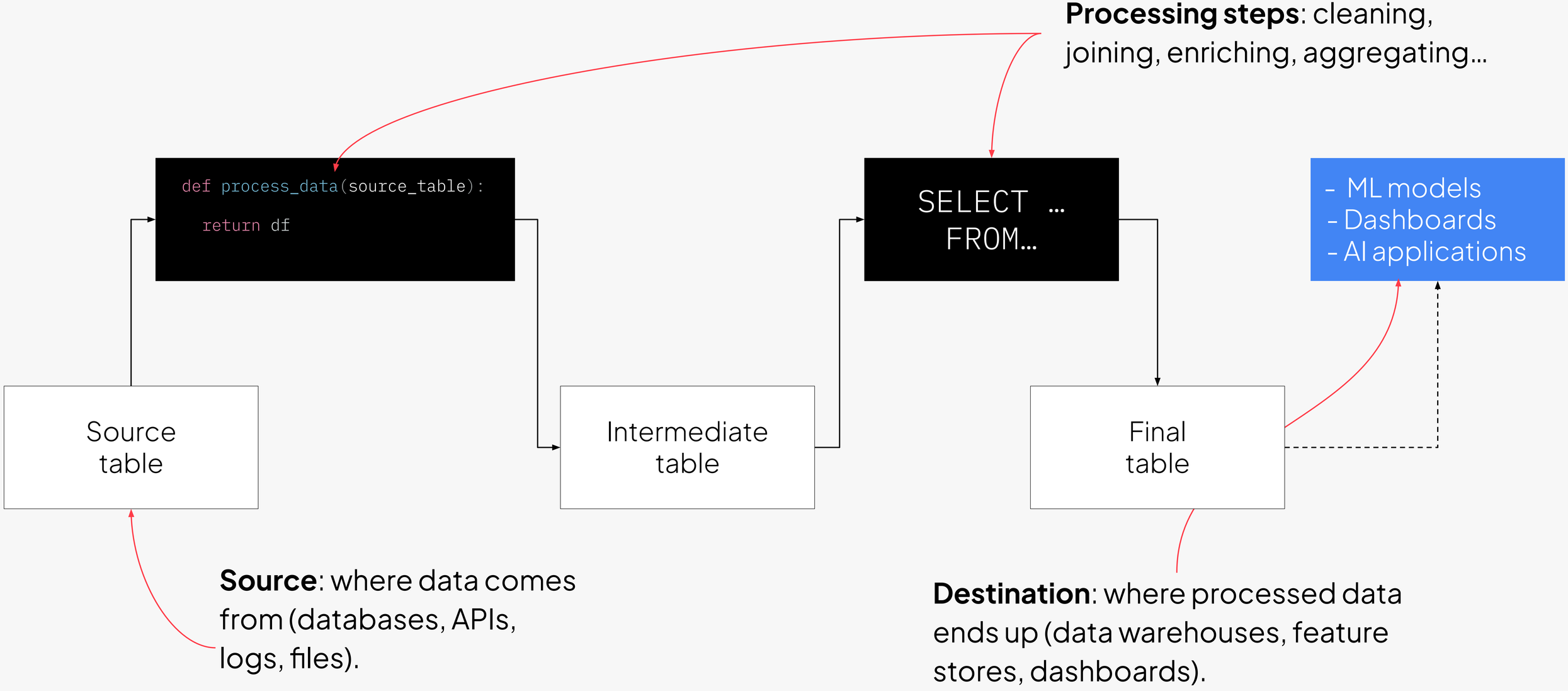
1. Data pipelines

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.

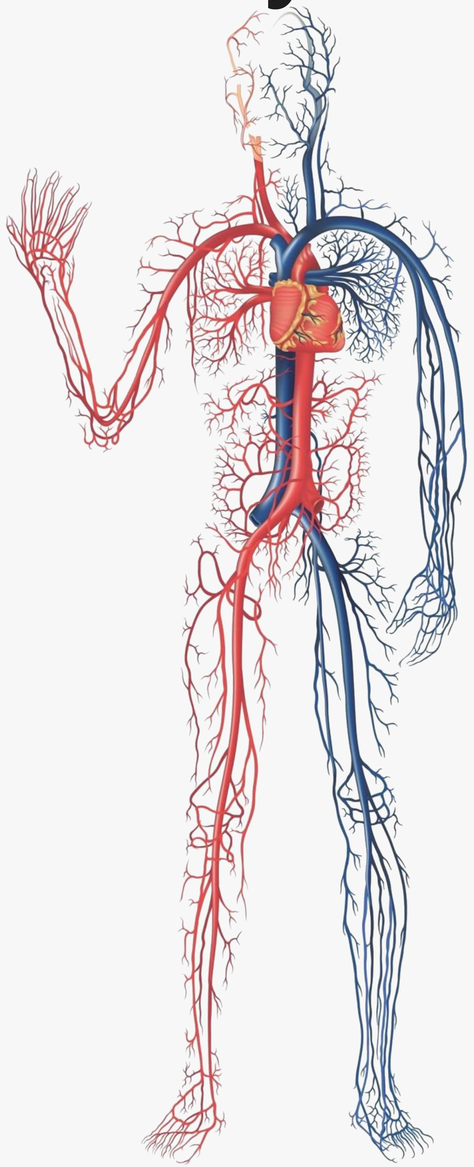


Core elements of a data pipeline

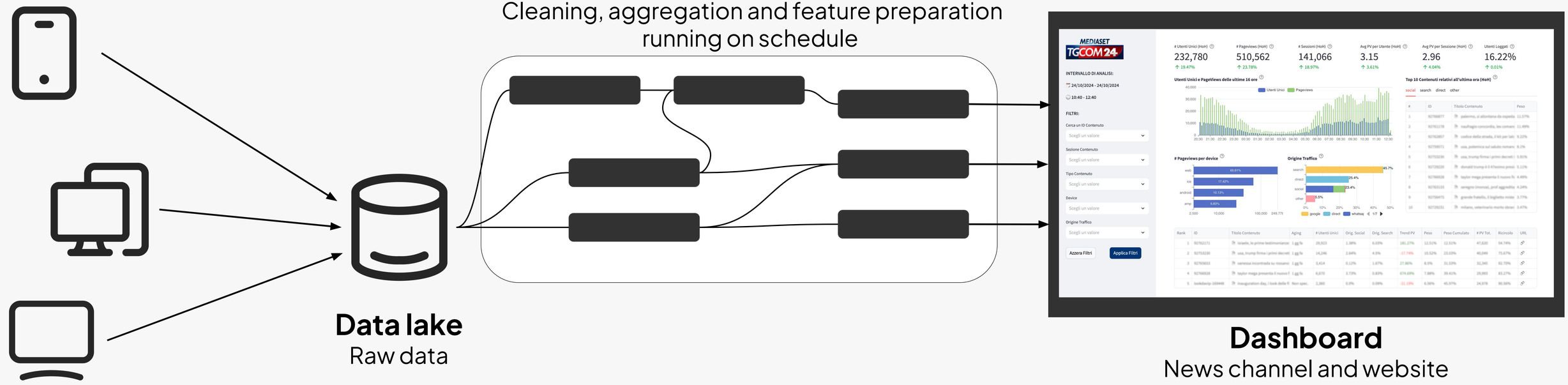


Data pipelines are the circulatory system of your data

- **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



Dashboard

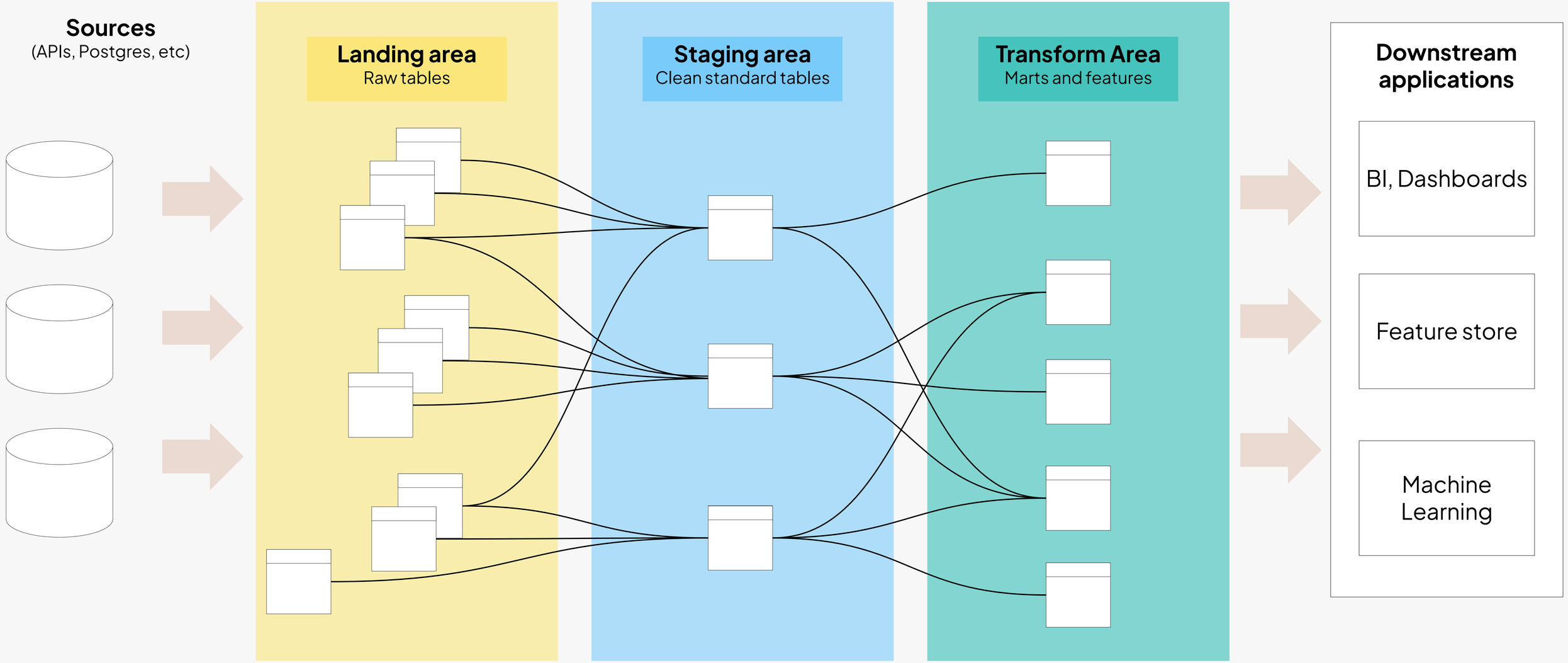
News channel and website

👥 Web: **17M unique users** per month

📺 TV: **12M viewers** per week

⏳ Refresh every **5 mins**

Landing, staging, transform architecture

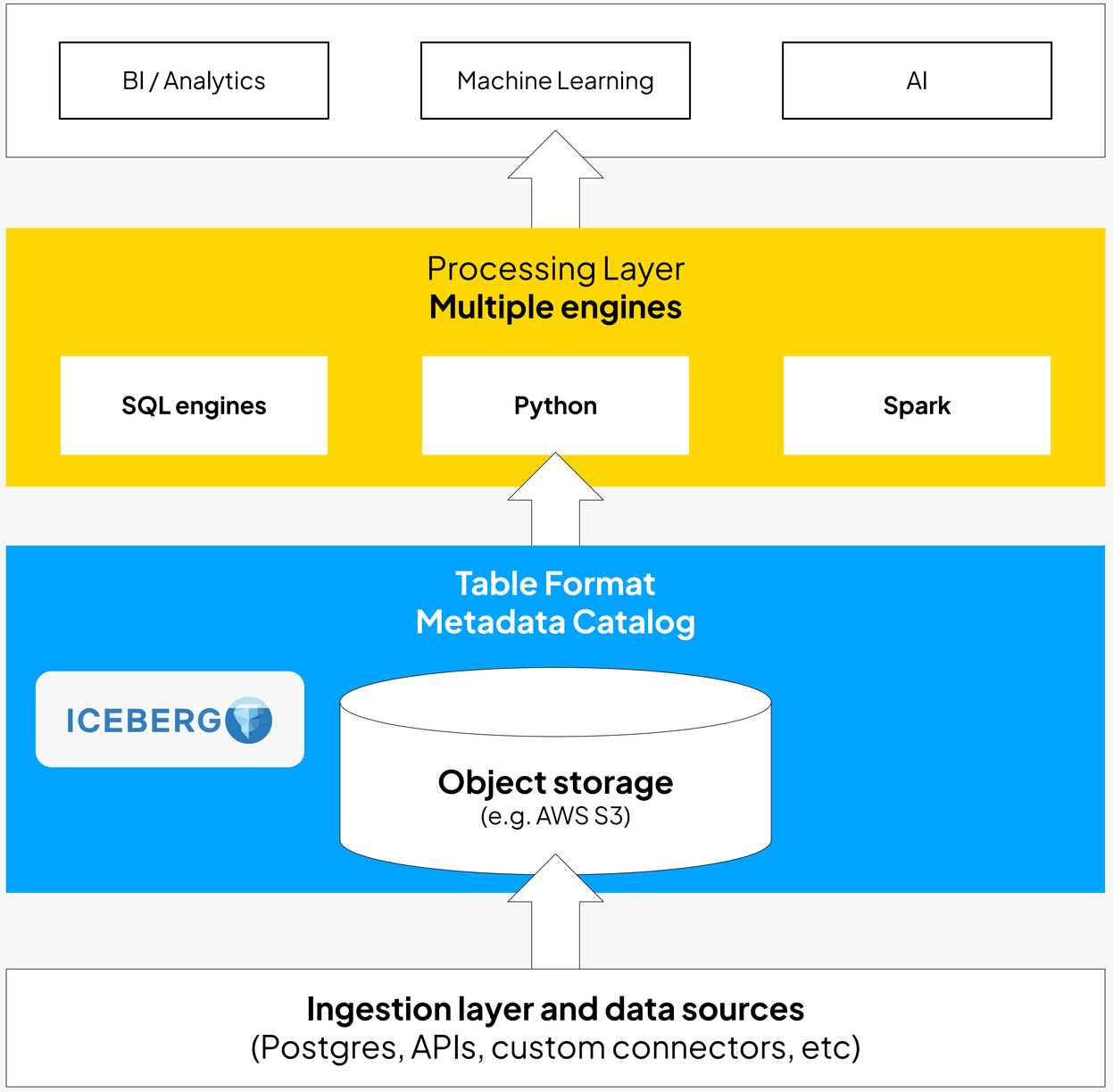


2. Data infrastructure

The data lakehouse

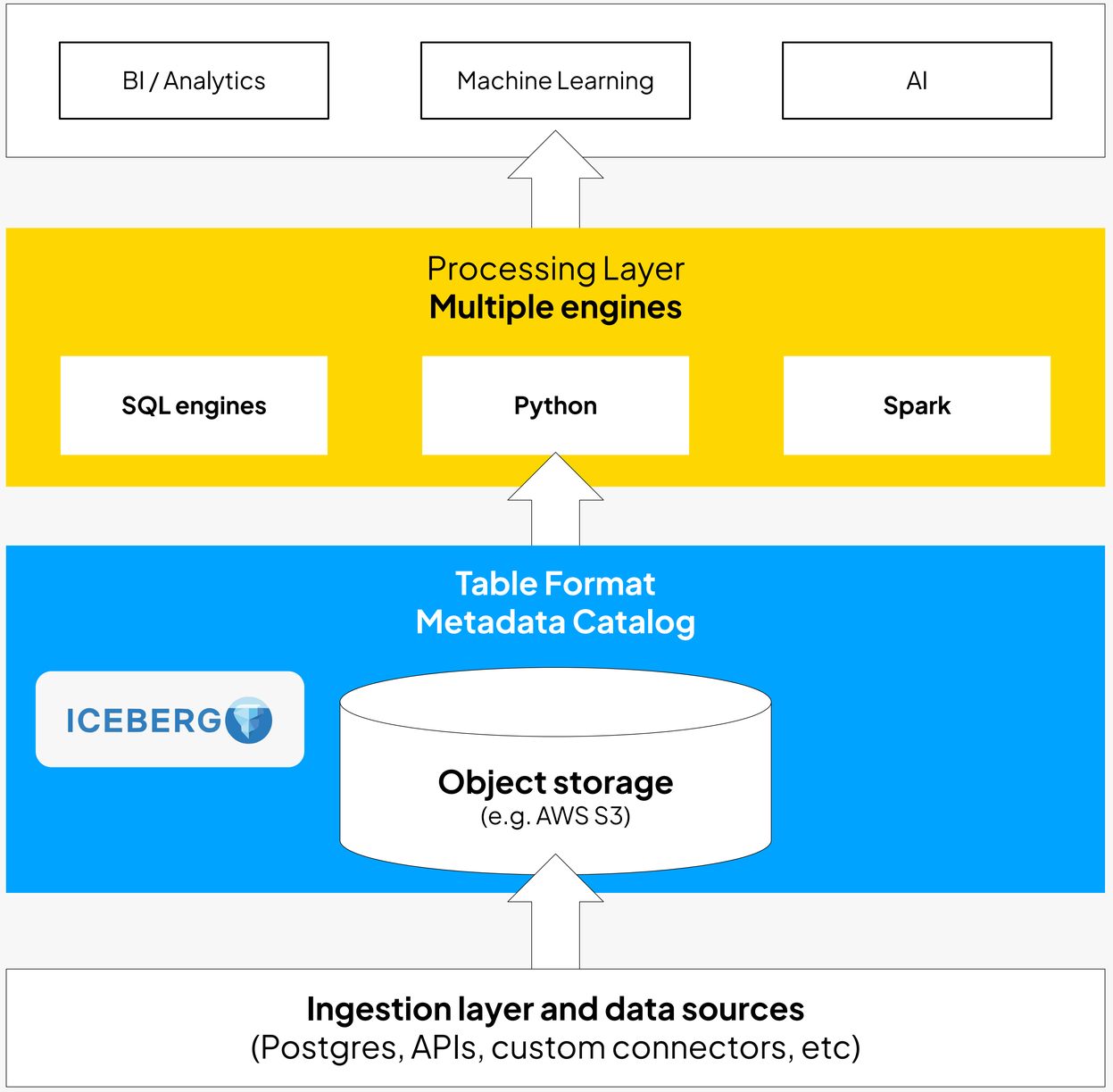
The scalability of a **data lake** + the management features of a **warehouse**

- 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.

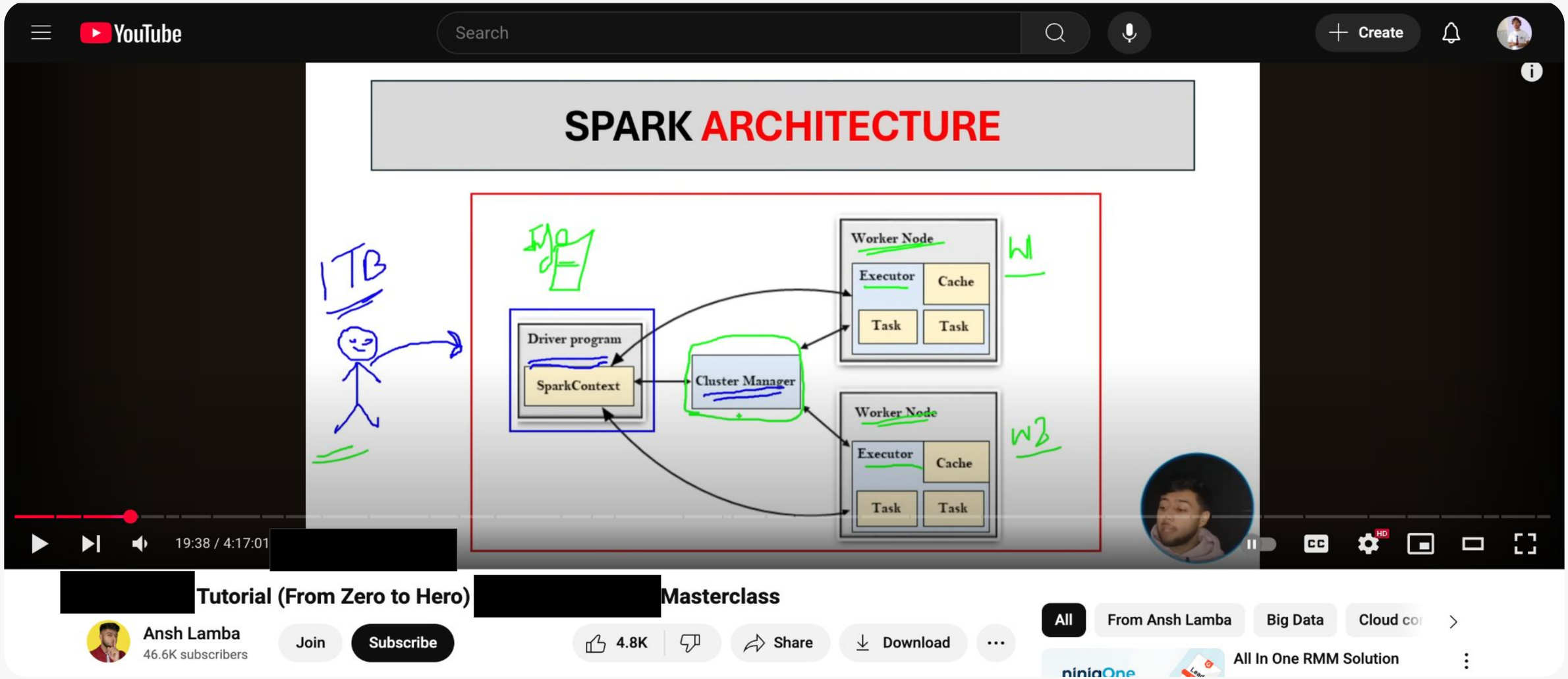
2. **Storage and catalog problems**
   - Inconsistent state.
   - Lack of data versioning.
   - Schema drift.

3. **Orchestration problems**
   - Waterfall errors.
   - Over-engineered DAGs.
   - Coupling logic to orchestration

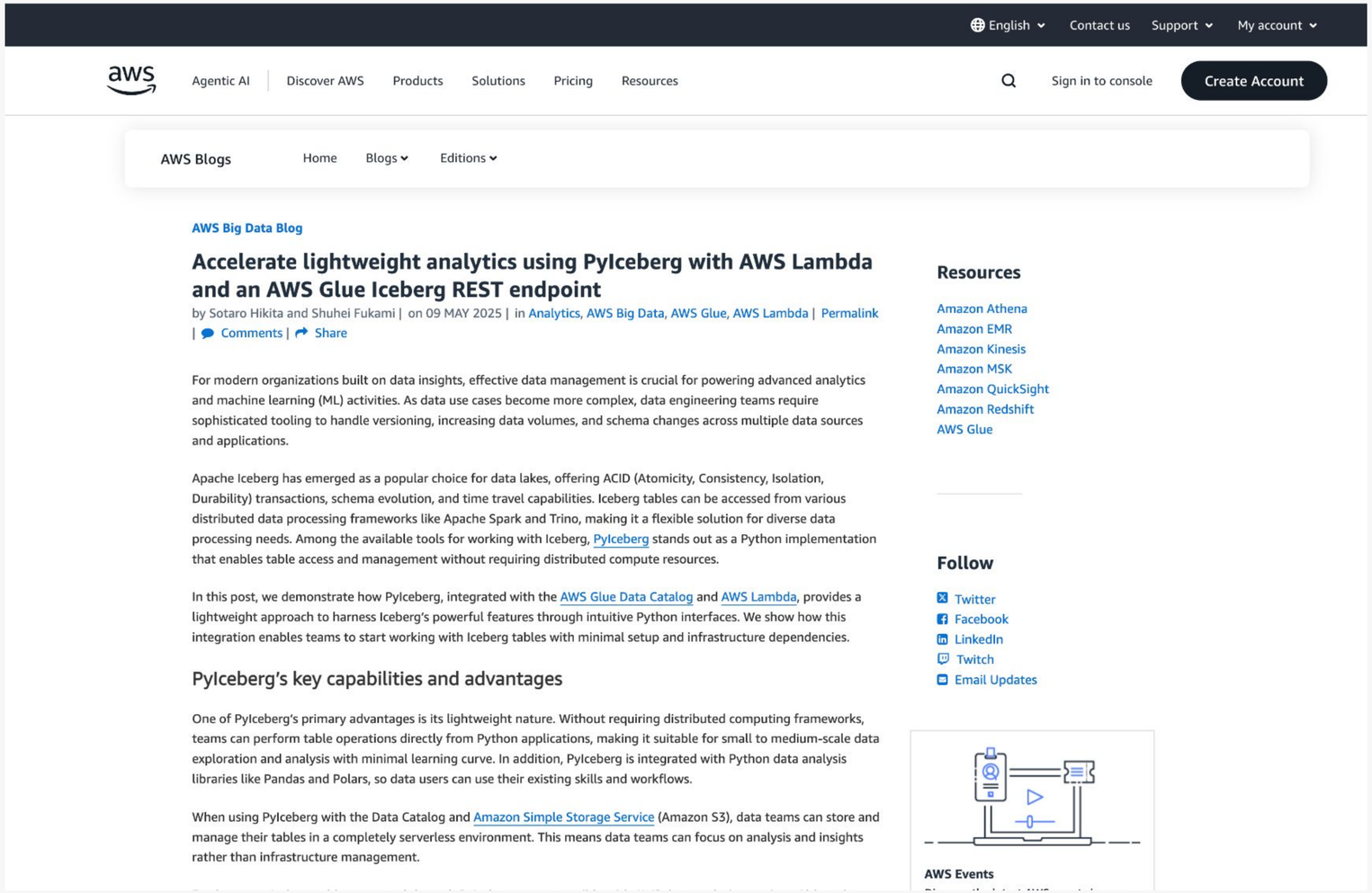
```
        at org.apache.spark.deploy.yarn.ApplicationMaster$.main(ApplicationMaster.scala:912)
        at org.apache.spark.deploy.yarn.ExecutorLauncher$.main(ApplicationMaster.scala:944)
        at org.apache.spark.deploy.yarn.ExecutorLauncher.main(ApplicationMaster.scala)
Caused by: java.io.IOException: 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
```


BLAH BLAH BLAH BLAH BLAH

4:17:01 (!?!)


YouTube
Search
Create
SPARK ARCHITECTURE
1TB
Driver program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task
Worker Node
Executor
Cache
Task
Task
19:38 / 4:17:01
[redacted] Tutorial (From Zero to Hero) [redacted] Masterclass
Ansh Lamba
46.6K subscribers
Join
Subscribe
4.8K
Share
Download
All
From Ansh Lamba
Big Data
All In One RMM Solution

Easy tutorials and DIY



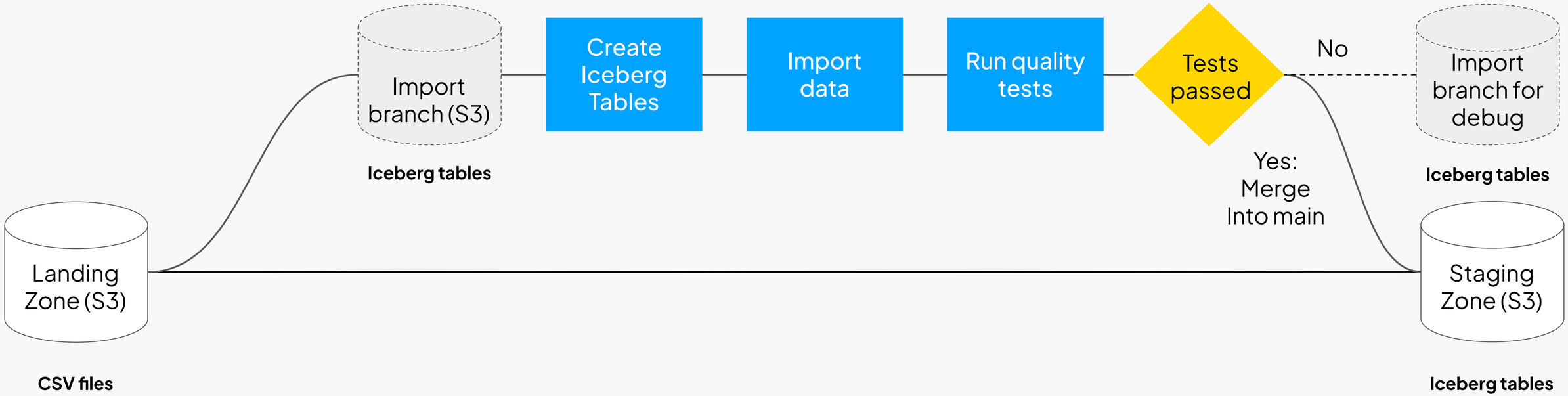
17 pages!!!

Today
Landing – Staging – Transform

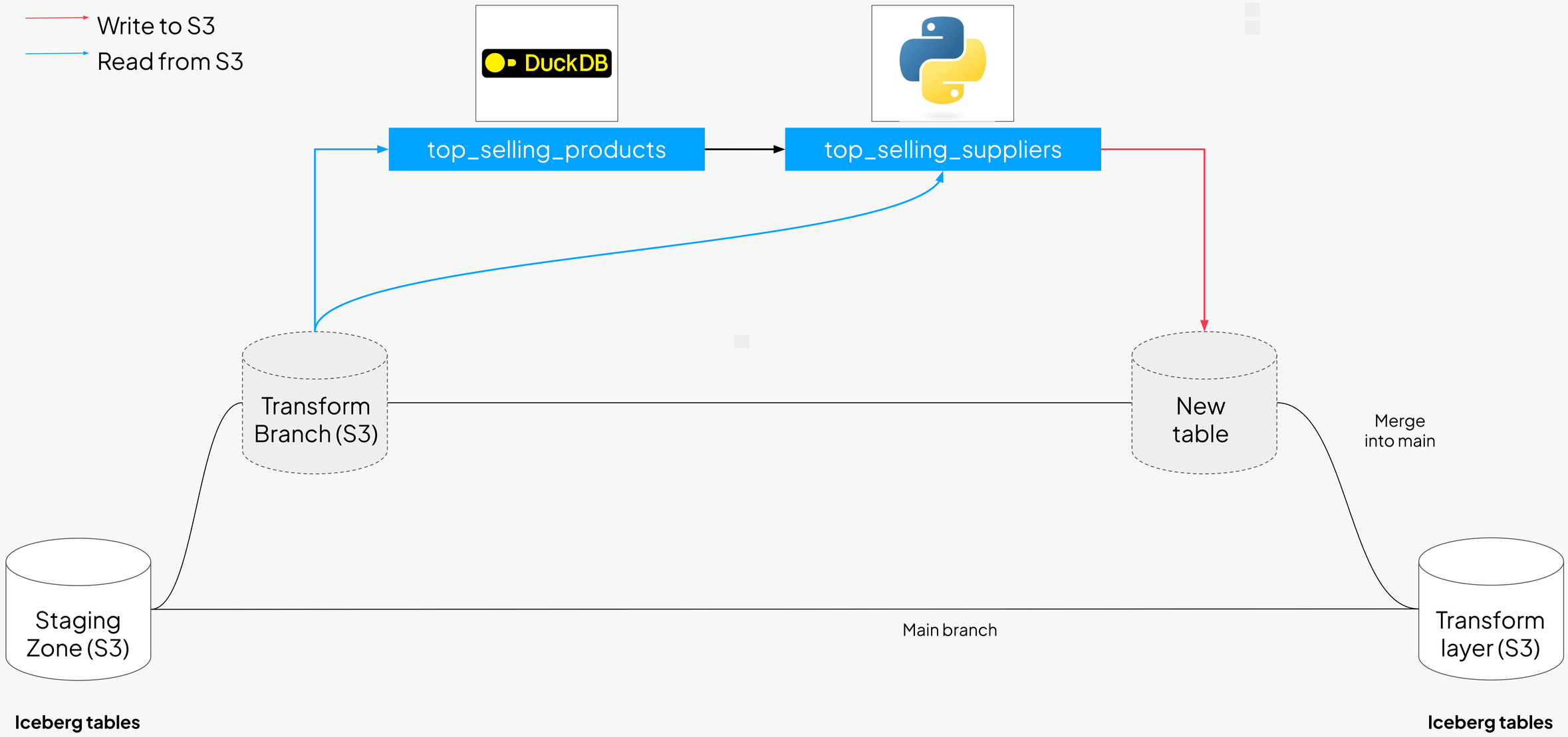
Bauplan is a lakehouse platform for engineering teams who treat data like software.

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

From Landing to Staging



From Staging to Transform



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.