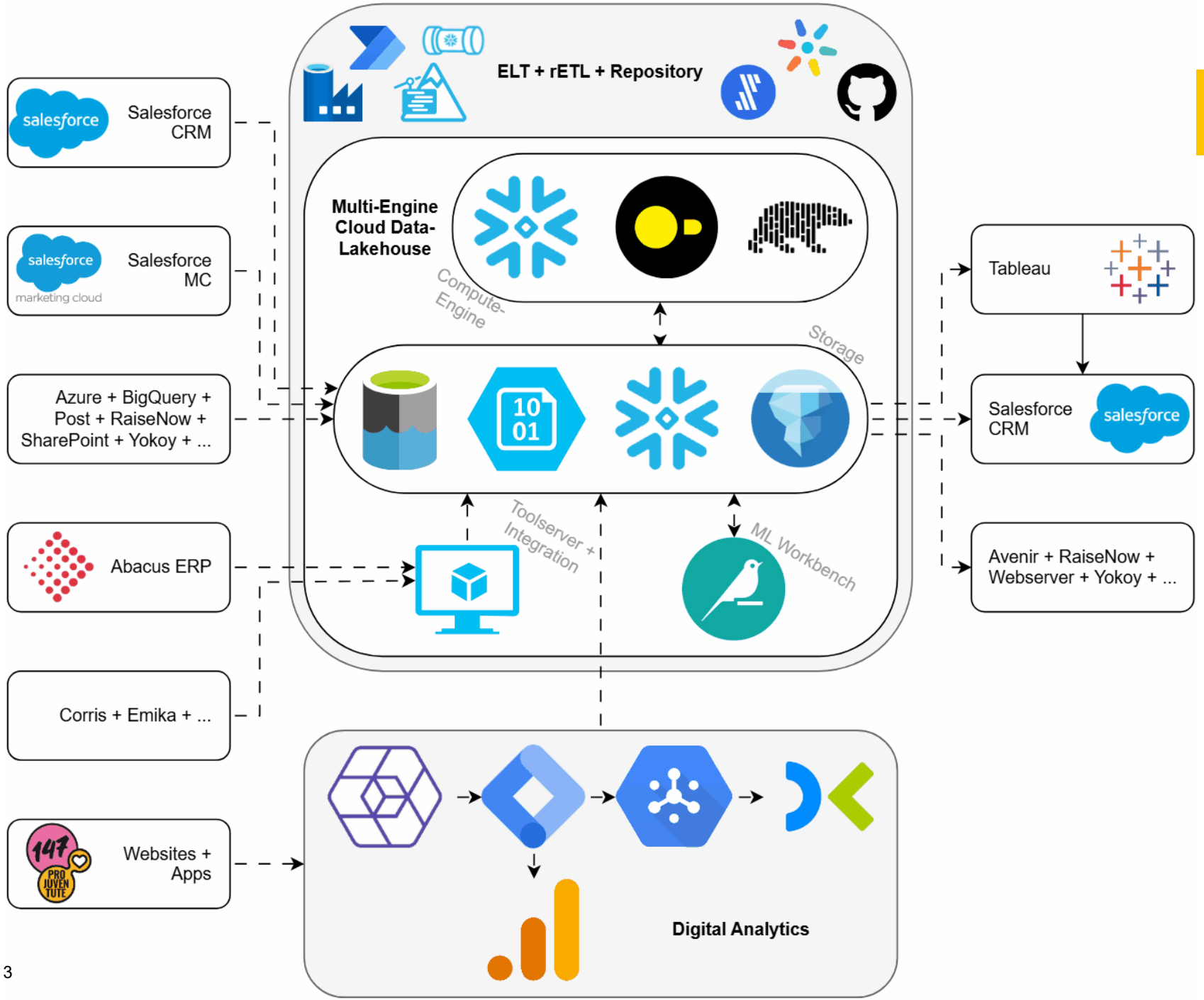
Multi-Engine Cloud Data-Lakehouse

Is it feasible (yet)?

About Pro Juventute

What do we do

- 147: national emergency hotline for children and young adults
- Letters to parents
- Media competency trainings
- Application trainings
- Lobbying on behalf of children
- Fundraising, IT, Finance, HR, ...
- ... and we have a hotel
- ... and, of course: data engineering and analytics



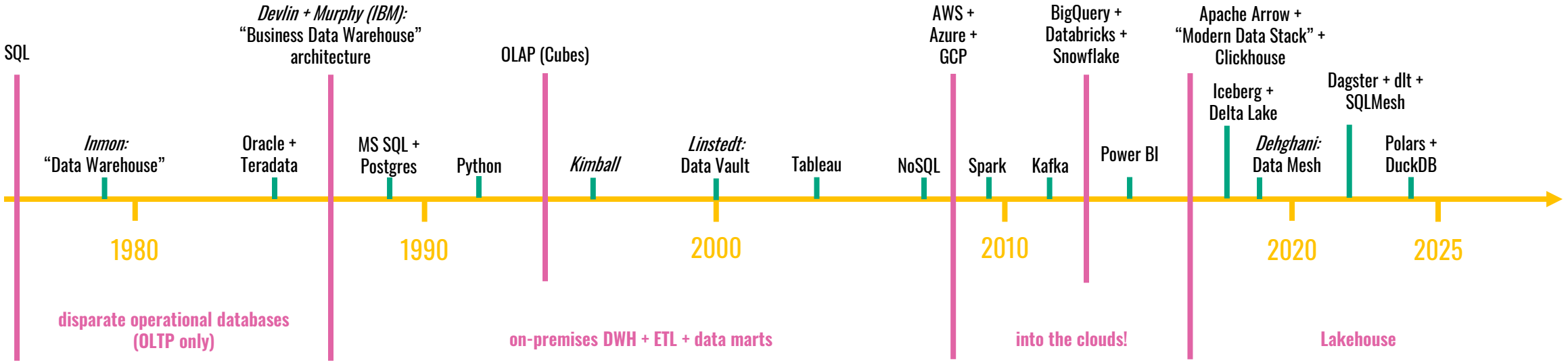
Looks fancy! How do we get there?

A brief history

How did we get here?

A brief history

How did we get here?



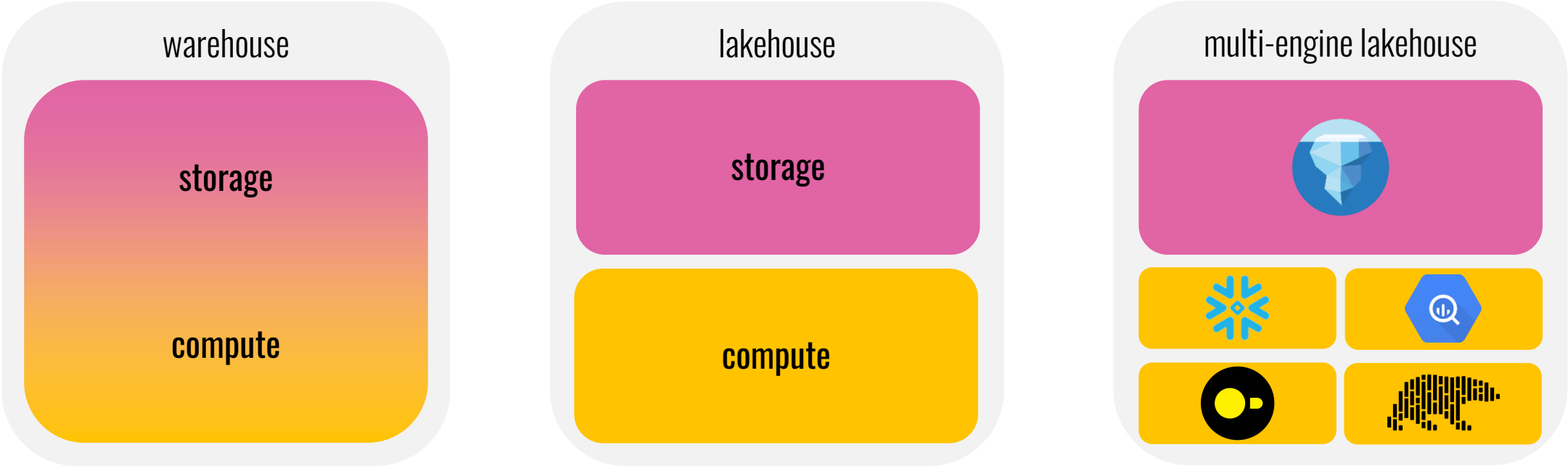
Components

... of a Lakehouse*

1. Integration / ingestion (bi-directional)
2. Storage (structured + semi-structured + unstructured)
3. Processing (compute)
4. Modeling
5. Orchestration
6. Visualization
7. Semantics (documentation + MCP)
8. Other meta-data (logs + lineage + ...)
9. Governance (ownership + access + security + ...)

* What else did I forget?

And what does “multi-engine” mean?



Storage and compute
are **really** separated

What’s wrong with the “Modern Data Stack”?

What's wrong

... with the "Modern Data Stack"?

- Hyper-modular → fragmentation → overhead
- Vendor lock-in
- Operational cost

TCO

The cost paradigm

Build
Bring your own

Buy
XaaS

How do break out

Enjoy listening to the **Foo Fighters “Breakout”** in your head...

How to break out

... of the “Modern Data Stack”?

1. Prioritize agnostic storage for flexibility

   → Iceberg (storage + catalog) on S3 / ADLS2 / GCS / Cloudflare R2 / ...

2. Migrate first use cases:

   - Identify use cases you want the results of to be accessible by other systems than your lakehouse
   - Identify use cases better suited for alternative compute engines

What if the same pipeline should sometimes use engine X and other times engine Y?

Some pipelines always use engine X, others use engine Y or any combination

→ easy enough!

How to break out

... of the “Modern Data Stack”?

1. Truly separate storage from compute

   → Iceberg
2. Build agnostic procedures (using plain SQL or Python - whichever is more performant),

   nothing else → dbt / SQLMesh / Coalesce?
3. Pick compute engines - remember: 90% of use cases fit in memory
4. Dump results in agnostic storage (Iceberg) again
5. Orchestrate

Drawbacks

How to decide which engine is best for which process?

→ Test and compare/benchmark the options

- SQL pipeline in Snowflake vs. DuckDB
- Python pipeline in Snowflake Snowpark vs. Polars

What if an engine is not available (i.e., my laptop is offline)?

→ Have an always-available variant of your pipeline ready (e.g. a second version of everything in Snowflake)

But then I need to maintain 2 (or more) variants/versions of each pipeline?

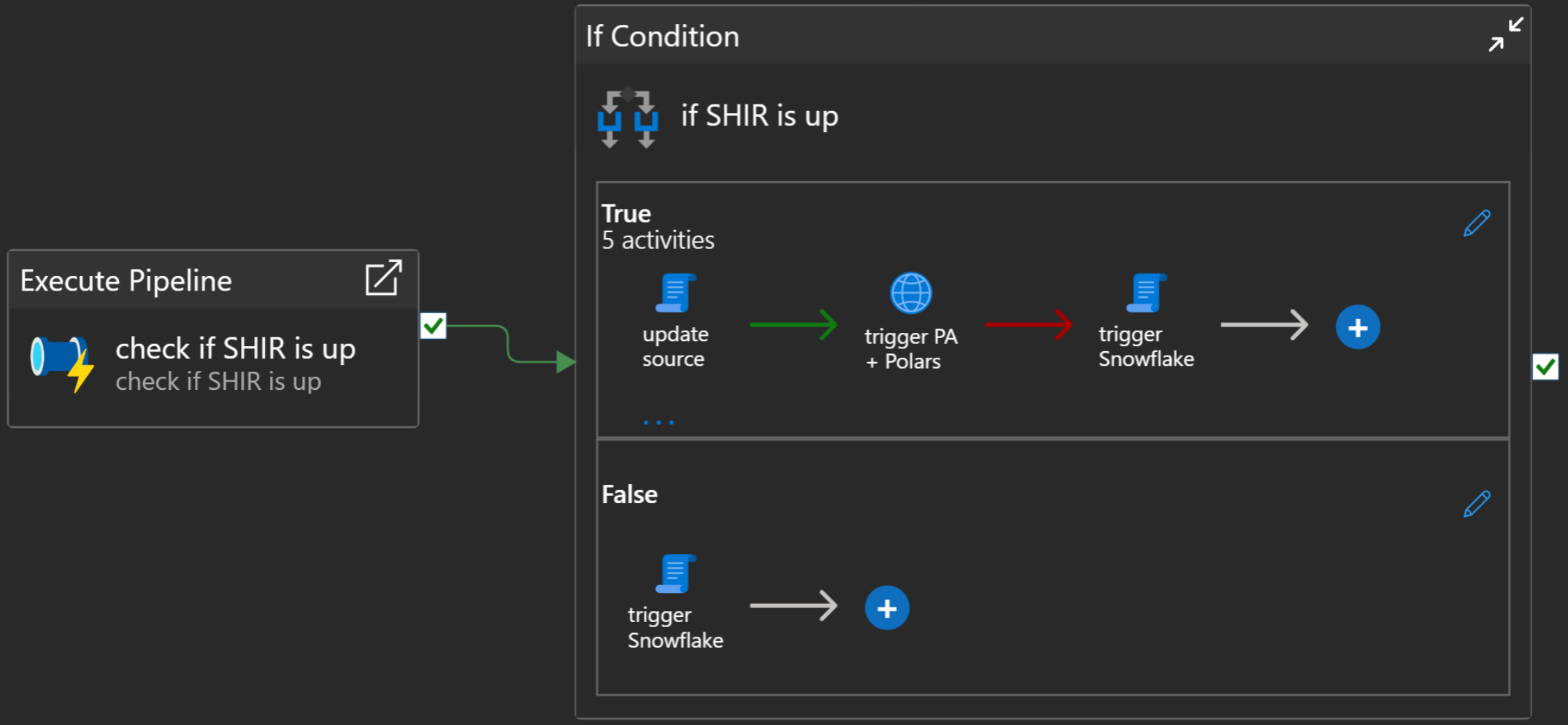
- That's why we use plain SQL/Python → Use 1 tool capable of modeling both variants to reduce friction
- Automatic orchestration options are on the horizon (though not quite here, yet): blog.greybeam.ai

Wait, another third-party tool to automatically pick (orchestrate) the engine?

Currently, yes – buuut...

Snowflake selects an available compute node already, if they get enough pressure to enable BYO-engine, **maybe some day**...

PRO JUVEN TUTE


Execute Pipeline
check if SHIR is up
check if SHIR is up
If Condition
if SHIR is up
True
5 activities
update source
trigger PA + Polars
trigger Snowflake
False
trigger Snowflake

Read more

juhache.substack.com

www.ssp.sh

blog.greybeam.ai

Thank you!

www.projuventute.ch/spenden

martin.seifert@projuventute.ch

www.seifert.link

www.linkedin.com/in/martinseifert