MACHINE LEARNING ON BIG DATA

Yury Zhuk

When your data doesn't fit in your laptop's memory

Shoutout to Rahul M.

- Organizer of this event + invited me ☺

- Academy Member @ Lumos
- FH Technikum MS, Data Science – 2027
- Data Engineer @ ÖAMTC |Data Scientist | ML Engineer | Python Dev

About Me – Yury Zhuk

- **Freelance Consultant**
  - Fractional CTO @ Shoelessly
  - Lead Eng @ Agent Alpha

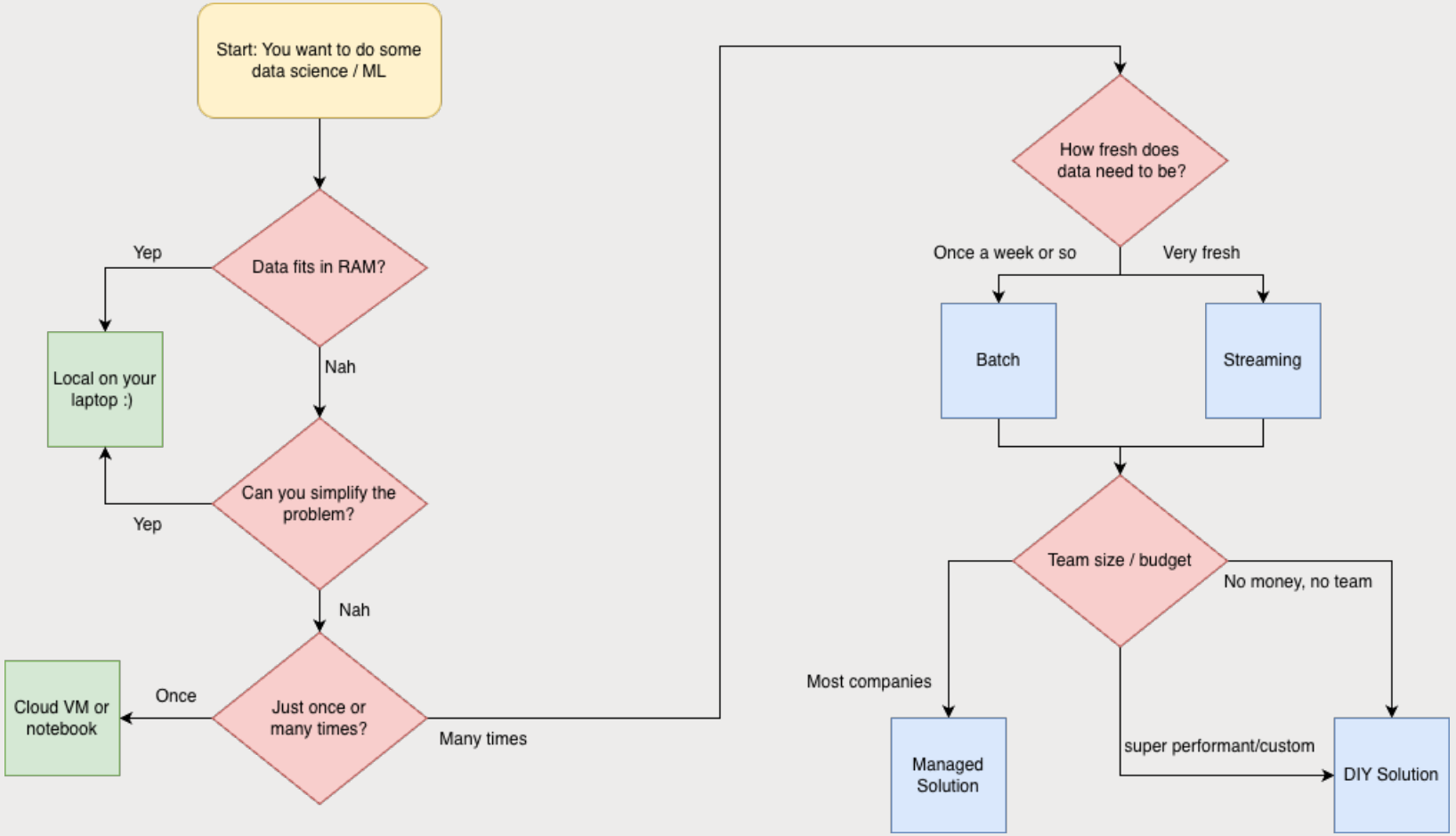
- **10+ years of building ML Products / AI Systems**
  - Computer Vision, Time Series FC, Transformers
  - GenAI, RAG, LLMs
  - Startups and large enterprise

- **Co-founded and scaled Balun Energy**

What we'll cover today

- Approaches to **manage massive data** that won't fit in memory (or disk, or GPU)
- The **"lay of the land"** in data tools: what to use, when
- Useful **data concepts**
- **Applications** in real-world (classic ML and GenAI/LLM)

By the end, we'll cover this chart



Session Format

- 15 min — Intro to the problem
- 15-20 min — Hands-on workshop
- 30 min — Advanced strategies

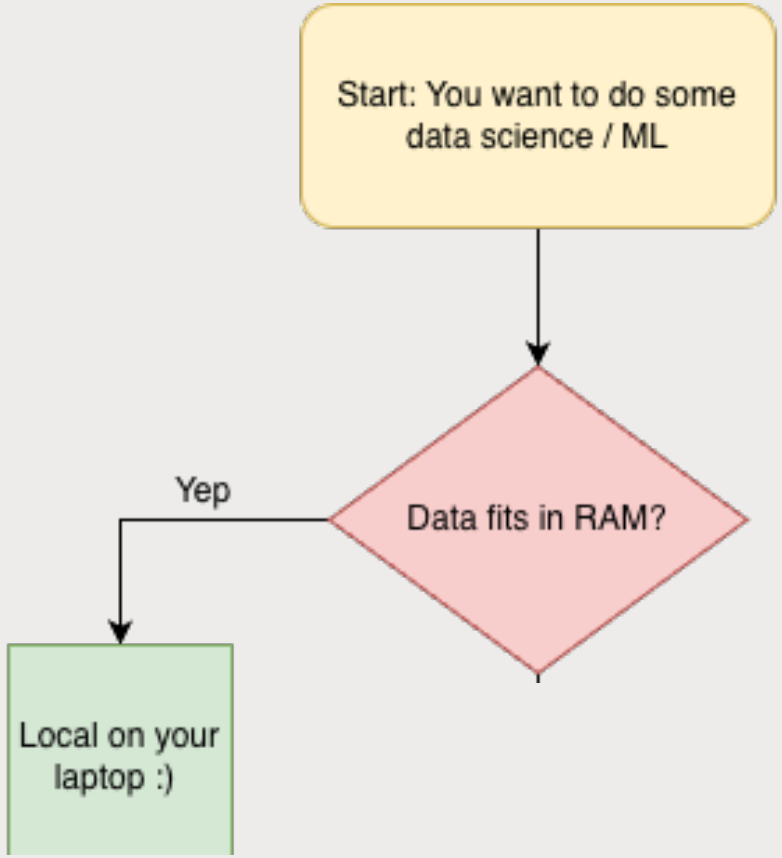
I'll keep going until time runs out ☺

? **Ask questions at any time!**

Quick Pulse Check

Show of hands 🙋

- Who can code?
- Knows a bit of SQL?
- Apache Spark? ⚡


Start: You want to do some data science / ML
Data fits in RAM?
Yep
Local on your laptop :)

Intro: problem

```
>>> import pandas as pd
>>> df = pd.read_csv("big_dataset.csv")
MemoryError: Unable to allocate 47.2 GiB
```

Now what?

Why Does This Happen?

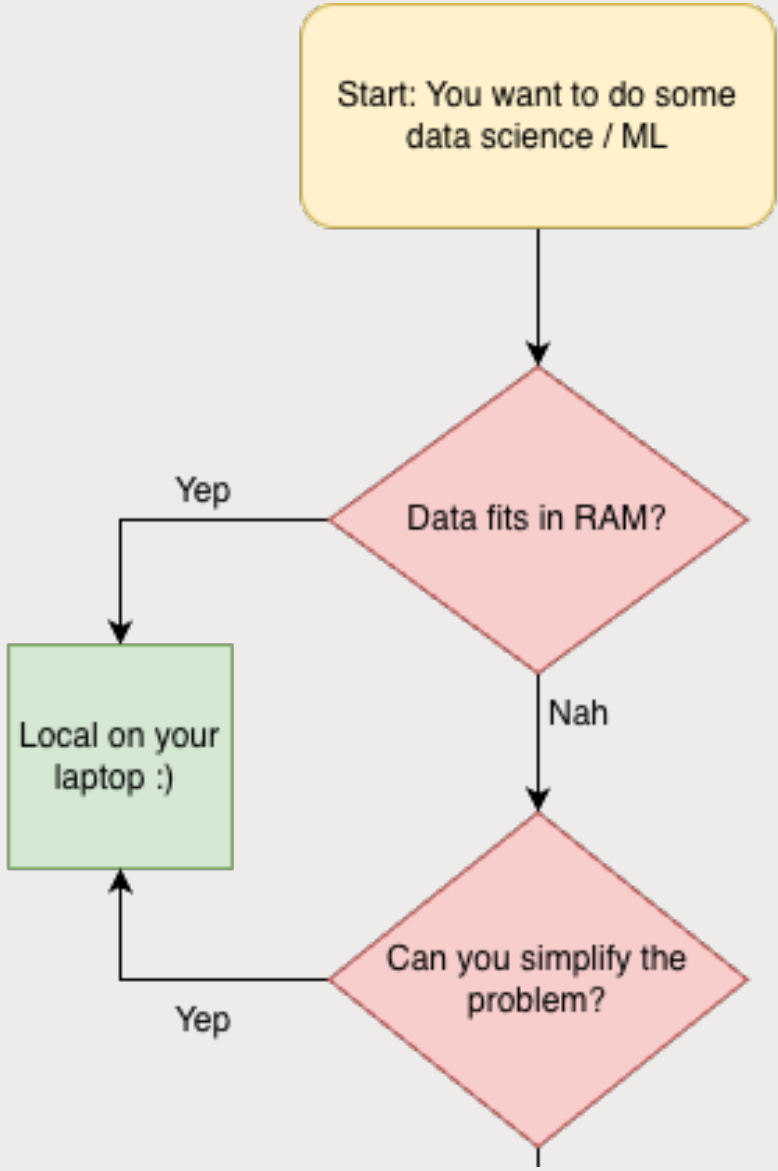
Rule of thumb for dataset size

rows × cols × 15 bytes = memory needed

Example: 50M rows × 7 cols × 15 bytes = ~15 GB

Pandas needs ~2-10x the data size

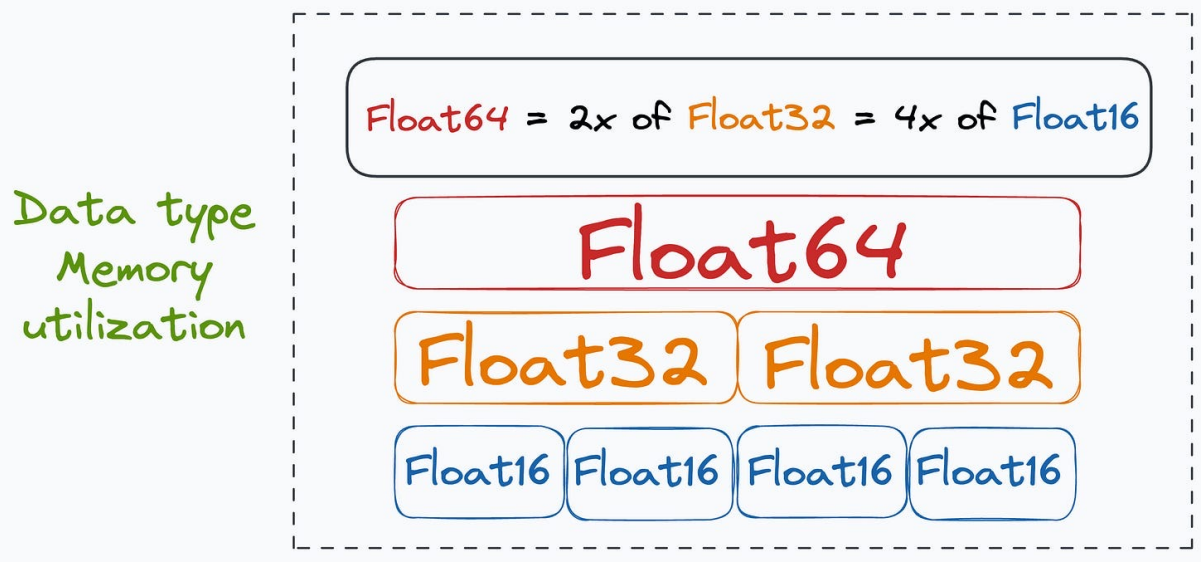
Even a 5 GB CSV can crash a 16 GB machine.


Start: You want to do some data science / ML
Data fits in RAM?
Yep
Nah
Local on your laptop :)
Can you simplify the problem?
Yep

Strategy 0: Reduce the Problem

Before reaching for new tools, **change the problem:**

- Sample, Filter early — do you really need all 500M rows?
- Subset columns — load only the 3 cols you need
- Downcast dtypes — float64 → float32 = half the memory



Example: 12345.67

Format	Approx Stored Value	Rounding Error
Float64	12345.67	~1e-12
Float32	12345.66992187	~5e-4
Float16	12344.0	±4

Img source: https://blog.dailydoseofds.com/p/mixed-precision-training

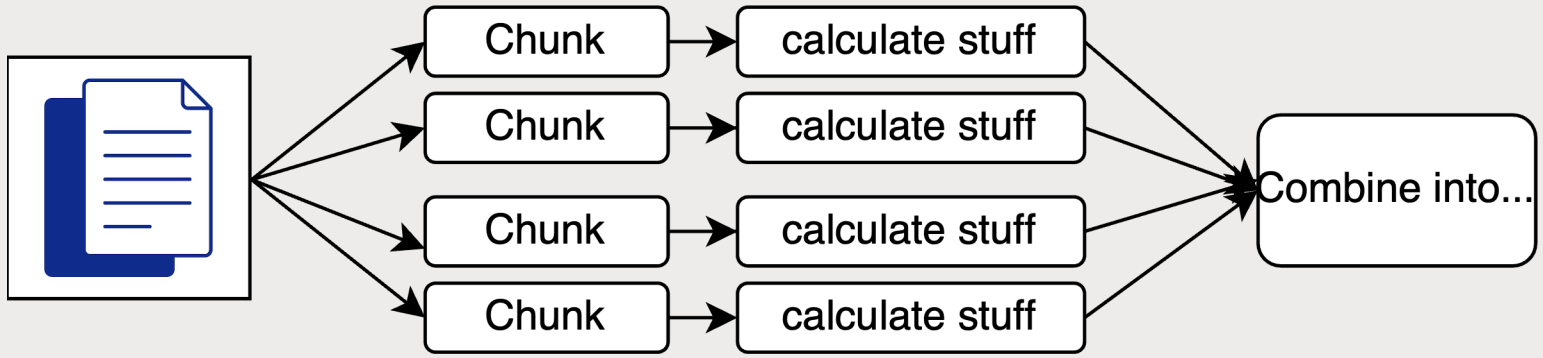
Strategy 1: Chunking

Works great for:

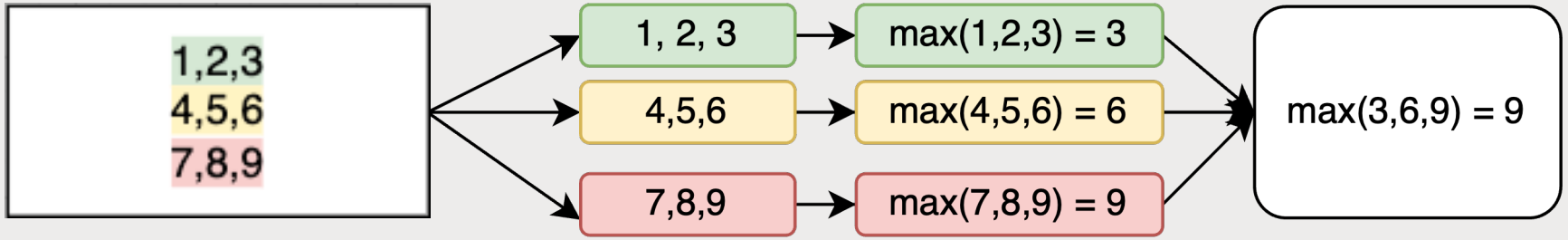
- Sum, count, max, min
- Mean (running sum / count)
- Row-level filters

More difficult work for:

- Median / percentiles
- Joins across chunks
- Window functions



Chunking - examples



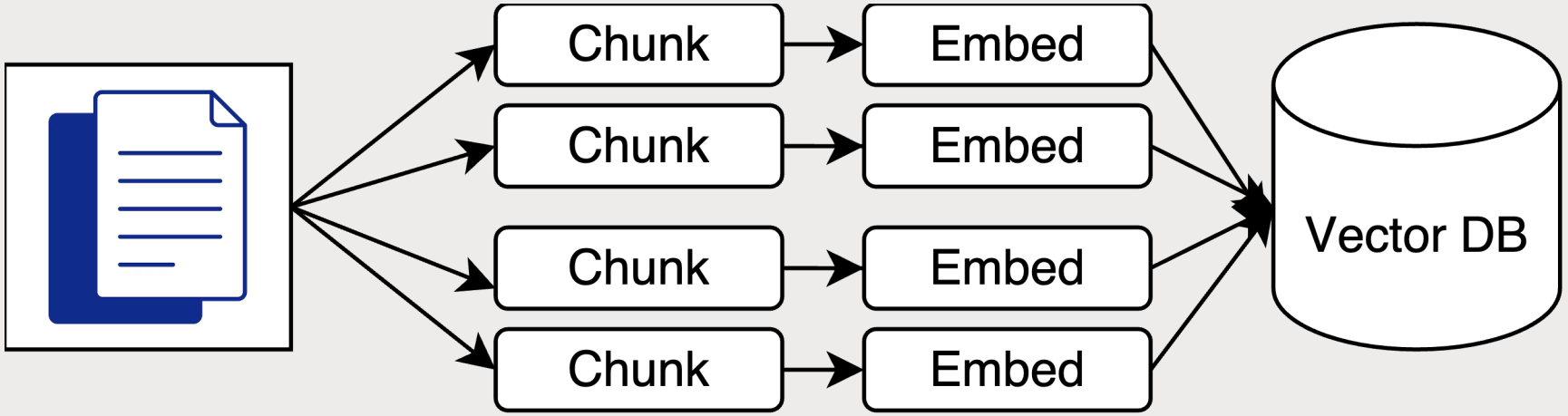
pyspark.sql.functions.approx_percentile

```
pyspark.sql.functions.approx_percentile(col, percentage,
accuracy=10000)
```
[source]

Returns the approximate *percentile* of the numeric column *col* which is the smallest value in the ordered *col* values (sorted from least to greatest) such that no more than *percentage* of *col* values is less than the value or equal to that value.

- Read More: https://www.techbrothersit.com/2025/05/how-to-use-approxquantile-in-pyspark.html

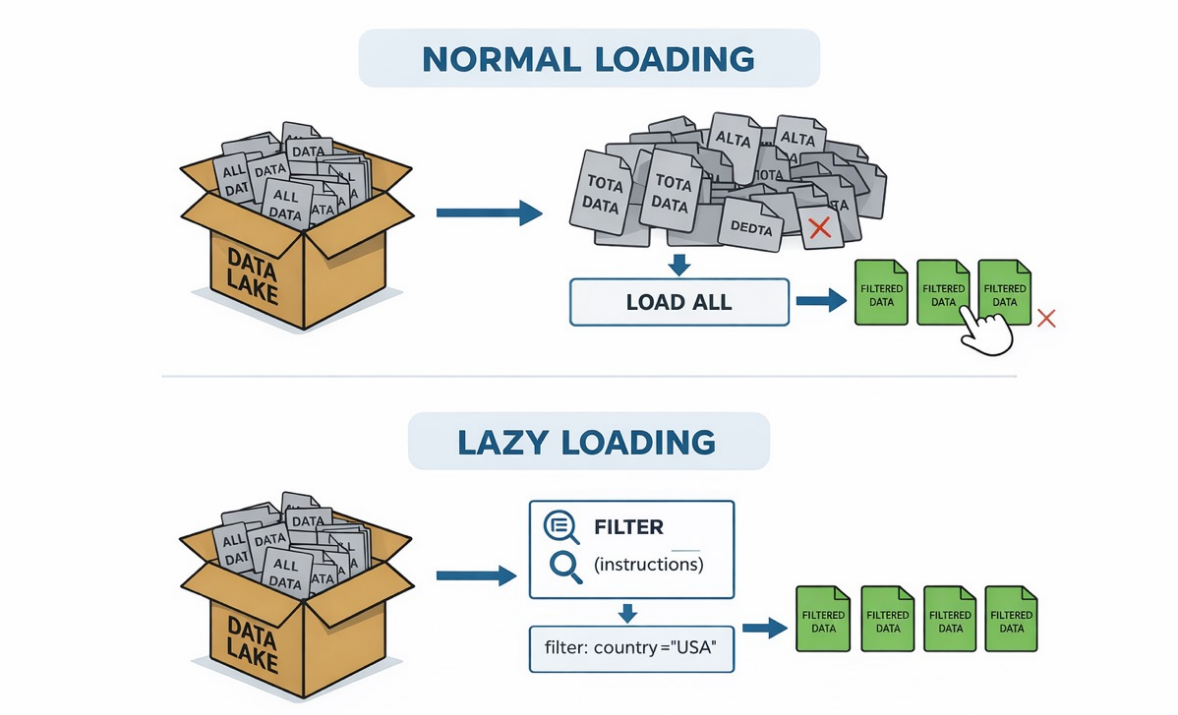
Aside: Chunking for RAG



- Read More: https://freedium-mirror.cfd/https://pub.towardsai.net/rag-part-1-chunking-strategies-90a05154f29c

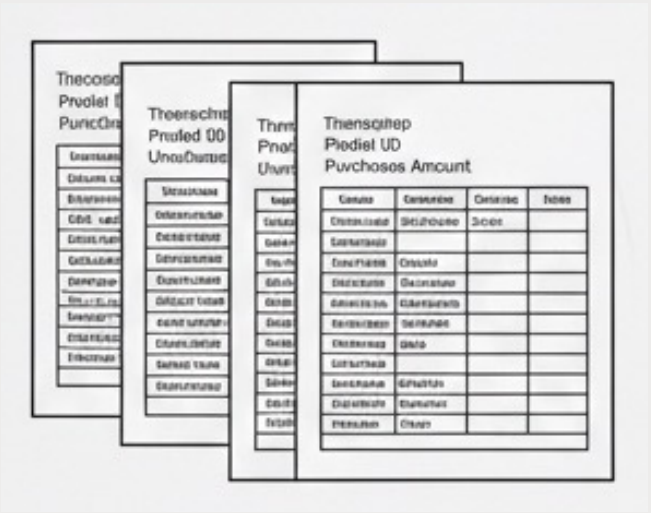
Strategy 3: Lazy Loading

- Build a **query plan** first, only materialize at the end
- The engine **optimizes** — pushes filters, skips unused columns
- Tools: Polars, DuckDB, Spark (all lazy by default or by API)



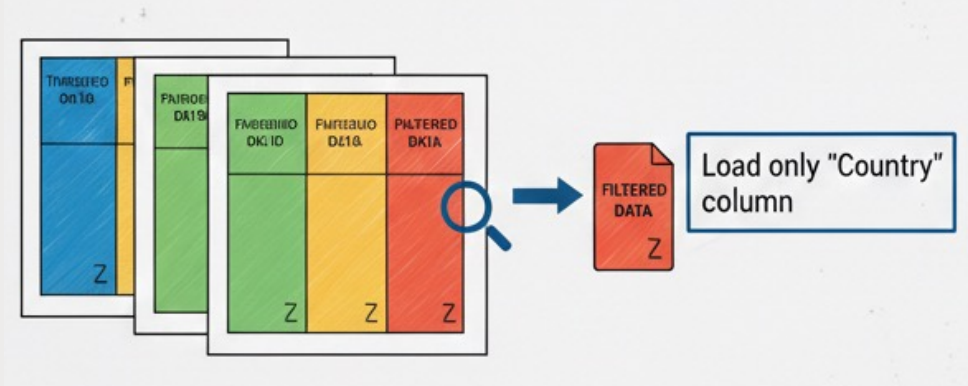
Strategy 4: Better File Formats

CSV, JSON
(Row-Based, uncompressed text)

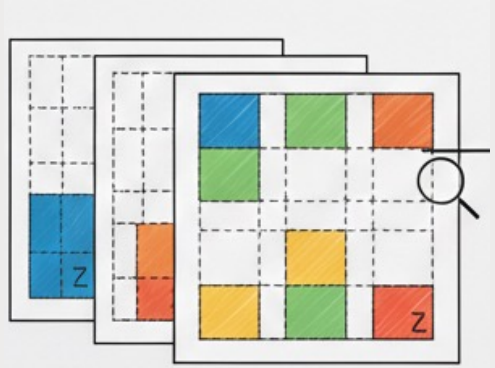


vs

Parquet
Column-based, Compressed binary



Xarray/Zarr
Chunk-based, compressed N-d Array



Tool Landscape

Feature/Tool	pandas	DuckDB	Polars	Apache Spark
SQL-Friendly	✗	✓	✓	✓
DataFrame API	✓	✗	✓	✓
Embedded / Zero-Setup	✓	✓	✓	✗
Lazy Loading	✗	✓	✓	✓
Distributed	✗	✗	✗	✓
Best For	• Quick exploration • Small datasets • **Prototyping**	• Analytical **SQL** queries • Local OLAP workloads • File-based analysis	• Fast local transforms • **Memory** efficiency • Streaming pipelines	• Large-scale ETL • Cluster computing • **Production** data pipelines

Every tool has trade-offs. The right choice depends on your data size, query patterns, and team skills.

Even these tools are not bulletproof…

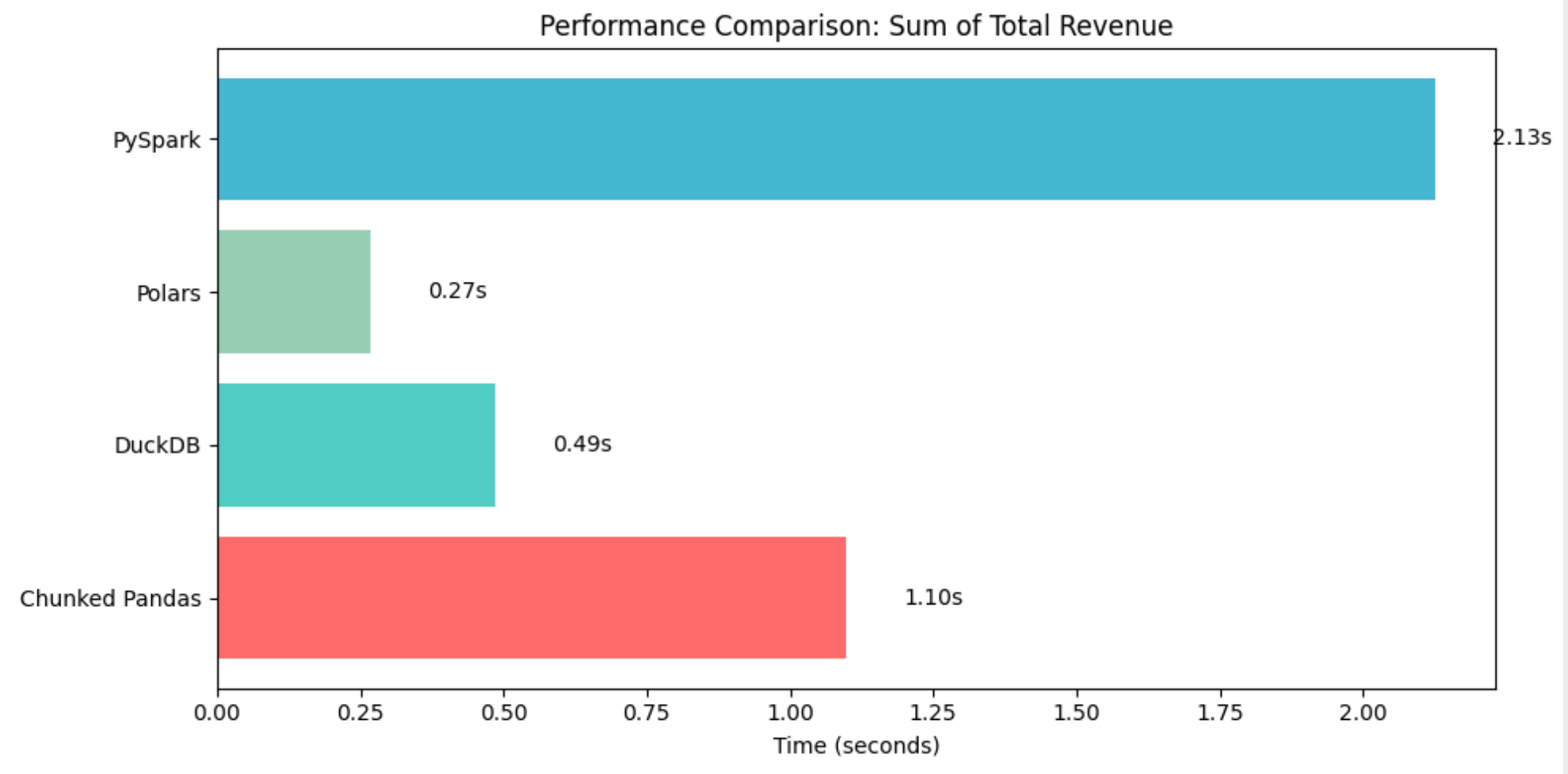
```
  at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:1605)
  at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:1594)
  at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:48)
  at org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:628)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:1928)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:1991)
  at org.apache.spark.rdd.RDD$$anonfun$reduce$1.apply(RDD.scala:1026)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:362)
  at org.apache.spark.rdd.RDD.reduce(RDD.scala:1008)
  at org.apache.spark.sql.Dataset.reduce(Dataset.scala:1434)
  ... 48 elided
Caused by: java.lang.OutOfMemoryError: Java heap space

scala> 20/06/09 08:28:58 ERROR TaskContextImpl: Error in TaskCompletionListener
java.lang.IllegalStateException: Block broadcast_24 not found
        at org.apache.spark.storage.BlockInfoManager$$anonfun$2.apply(BlockInfoManager.scala:293)
        at org.apache.spark.storage.BlockInfoManager$$anonfun$2.apply(BlockInfoManager.scala:293)
        at scala.Option.getOrElse(Option.scala:121)
        at org.apache.spark.storage.BlockInfoManager.unlock(BlockInfoManager.scala:292)
        at org.apache.spark.storage.BlockManager.releaseLock(BlockManager.scala:671)
        at org.apache.spark.broadcast.TorrentBroadcast$$anonfun$org$apache$spark$broadcast$TorrentBroadcast$$relea
        at org.apache.spark.broadcast.TorrentBroadcast$$anonfun$org$apache$spark$broadcast$TorrentBroadcast$$relea
        at org.apache.spark.TaskContext$$anon$1.onTaskCompletion(TaskContext.scala:123)
        at org.apache.spark.TaskContextImpl$$anonfun$markTaskCompleted$1.apply(TaskContextImpl.scala:97)
        at org.apache.spark.TaskContextImpl$$anonfun$markTaskCompleted$1.apply(TaskContextImpl.scala:95)
        at scala.collection.mutable.ResizableArray$class.foreach(ResizableArray.scala:59)
        at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:48)
```

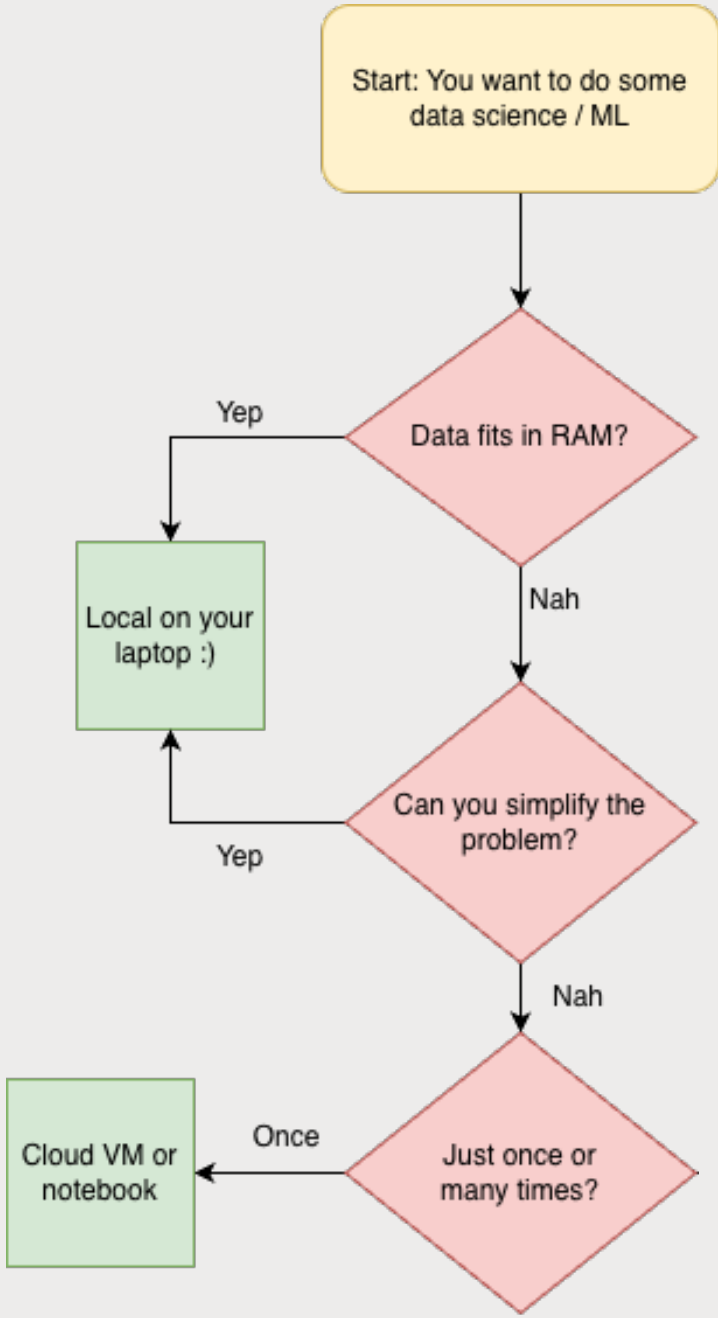
WORKSHOP TIME!

Workshop Setup

1. Go to the link
2. Login with your Google account
3. (Optional) Click "Copy to Drive" to save your changes
4. Click "Connect" in top right, then check RAM/Disk button
5. Everyone: run **Sections 0 and 1**
6. Advanced: continue as far as you get ☺

Query Times of Tools




Start: You want to do some data science / ML
Data fits in RAM?
Yep
Local on your laptop :)
Nah
Can you simplify the problem?
Yep
Nah
Just once or many times?
Once
Cloud VM or notebook

The Underrated Cloud VM

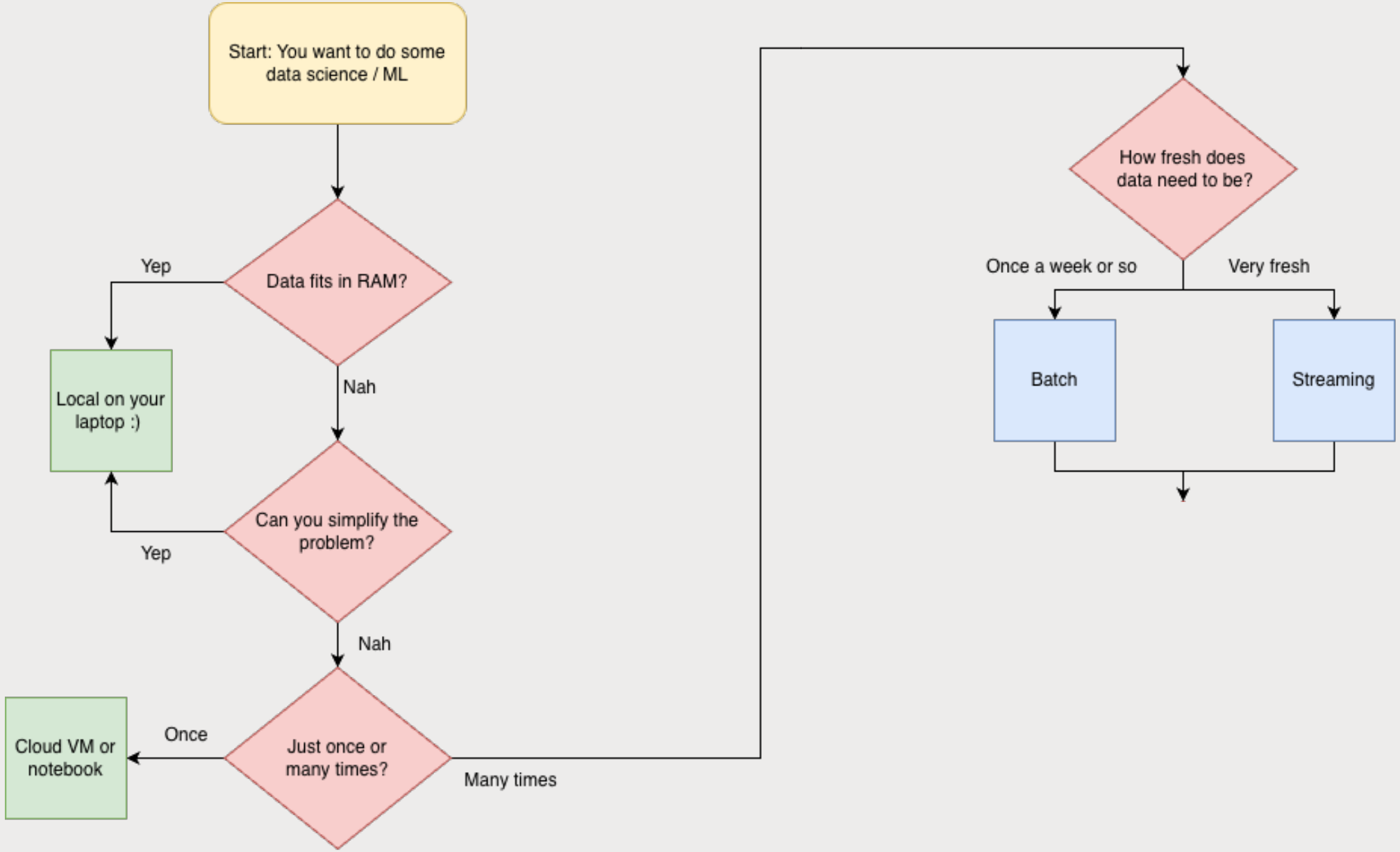
If it's a **one-off analysis**, don't overthink it:

- Rent a cloud VM with 256 GB RAM for a few $/hour
- Spin it up, run your script, shut it down
- No infrastructure, no pipeline, no Spark cluster

Underrated!

Even easier options for students:

- Databricks free edition
- Google Colab
- AWS Sagemaker Studio Lab
- Azure ML Studio

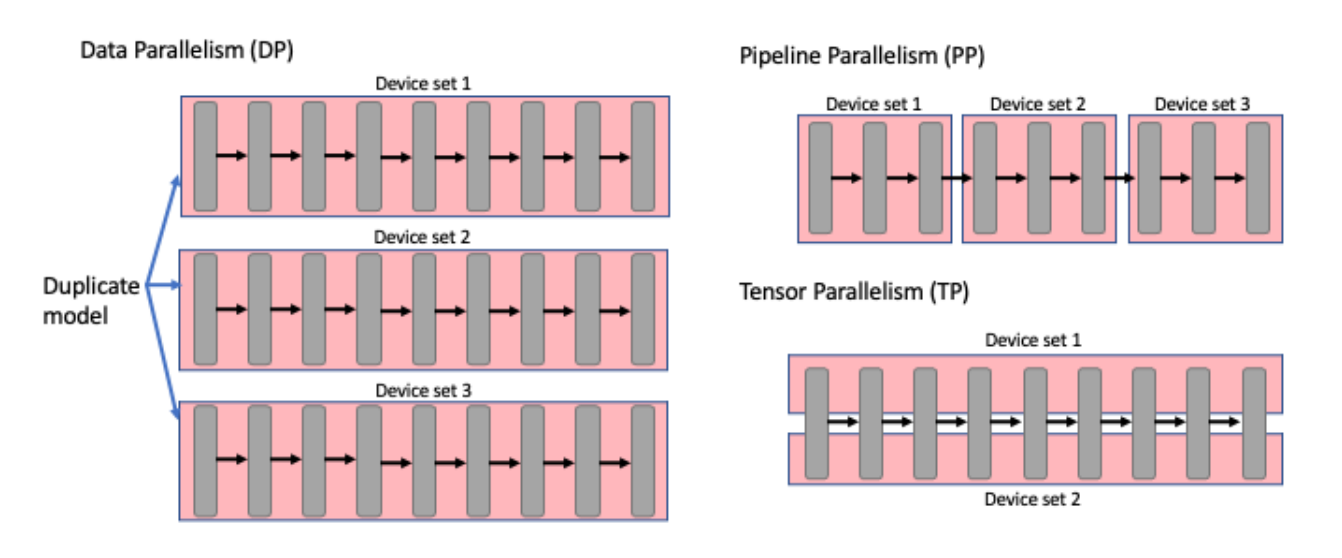

Start: You want to do some data science / ML
Data fits in RAM?
Yep
Local on your laptop :)
Nah
Can you simplify the problem?
Yep
Nah
Just once or many times?
Once
Cloud VM or notebook
Many times
How fresh does data need to be?
Once a week or so
Very fresh
Batch
Streaming

Aside: What If the Model Is the Bottleneck?

Everything so far assumed **data** is the bottleneck. Sometimes the **model** is.

- Data parallelism — same model, split data across GPUs
- Model parallelism — split model layers across GPUs

Industry tools used for large-scale Transformer / GenAI / LLM training: **DeepSpeed**, **Megatron-LM**

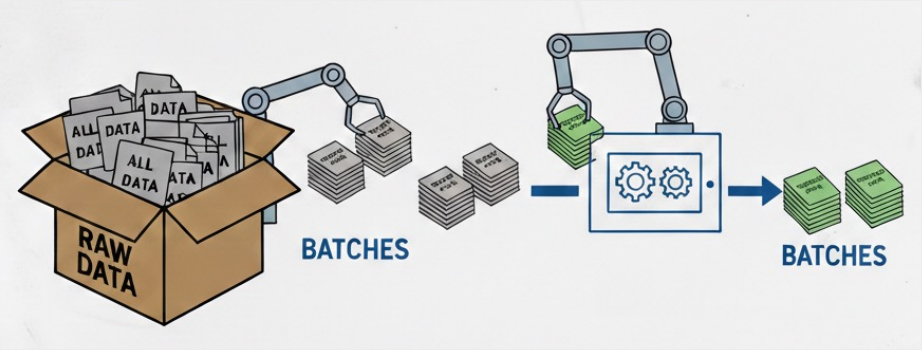


https://robotchinwag.com/posts/demystifying-tensor-parallelism/

Batch vs Streaming

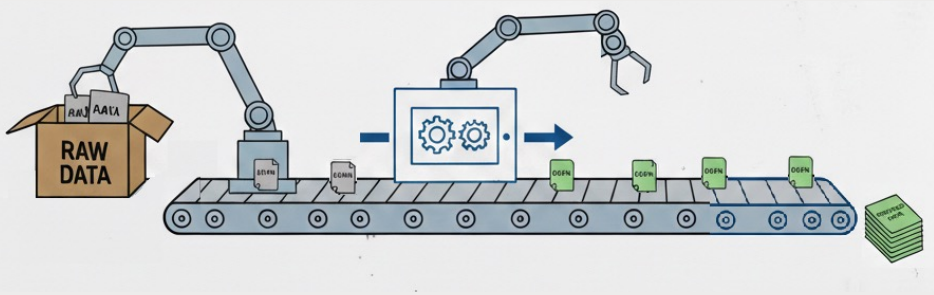
Batch (hours to days)

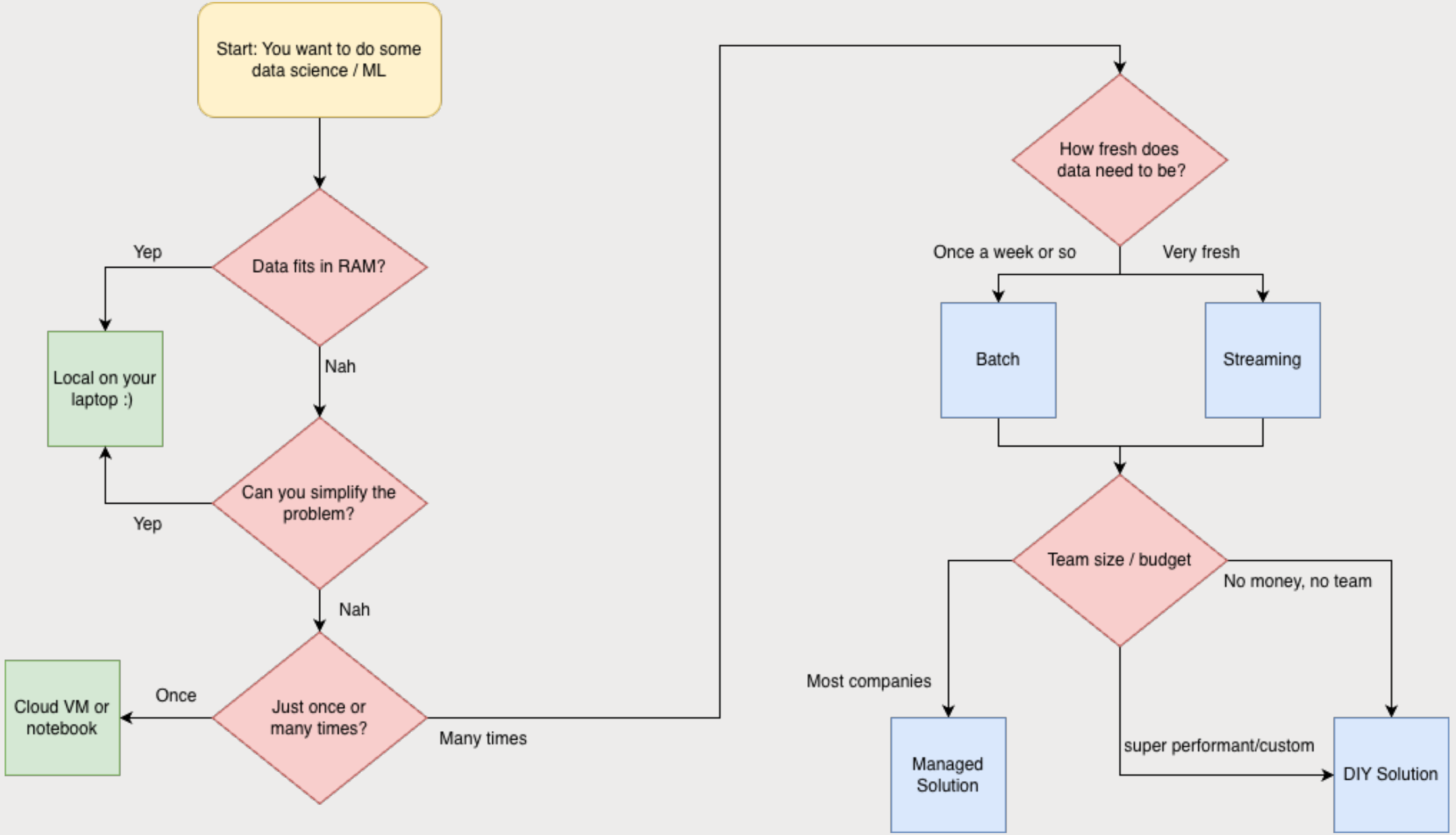
- Run on a schedule, store result
- Most ML pipelines live here
- **Example: nightly churn prediction**



Streaming (seconds to minutes)

- Data never stops, consumer can't wait
- Think event streams, not datasets
- **Example: fraud detection on each txn**




Start: You want to do some data science / ML
Data fits in RAM?
Yep
Local on your laptop :)
Nah
Can you simplify the problem?
Yep
Nah
Just once or many times?
Once
Cloud VM or notebook
Many times
How fresh does data need to be?
Once a week or so
Very fresh
Batch
Streaming
Team size / budget
No money, no team
Most companies
Managed Solution
super performant/custom
DIY Solution

Self-Managed Stack

Spark + Airflow on a VM

- Full control over every component
- More ops burden — you manage updates, failures, scaling
- Lower cost at small scale

Right for: small team that enjoys infra + has specific needs

Are Your Tables Interdependent?

Independent Jobs

- 5 tables, each recomputed on a schedule
- Simple cron job handles it
- No dependency worries

Cascading Dependencies

- A → B → C and D
- If A fails, everything is stale
- Need DAGs + lineage tracking

Orchestration Tools

When dependencies matter, you need orchestration:

- dbt — SQL transformations with built-in DAG
- Delta Live Tables — Databricks' declarative pipelines
- Airflow — general-purpose workflow orchestrator

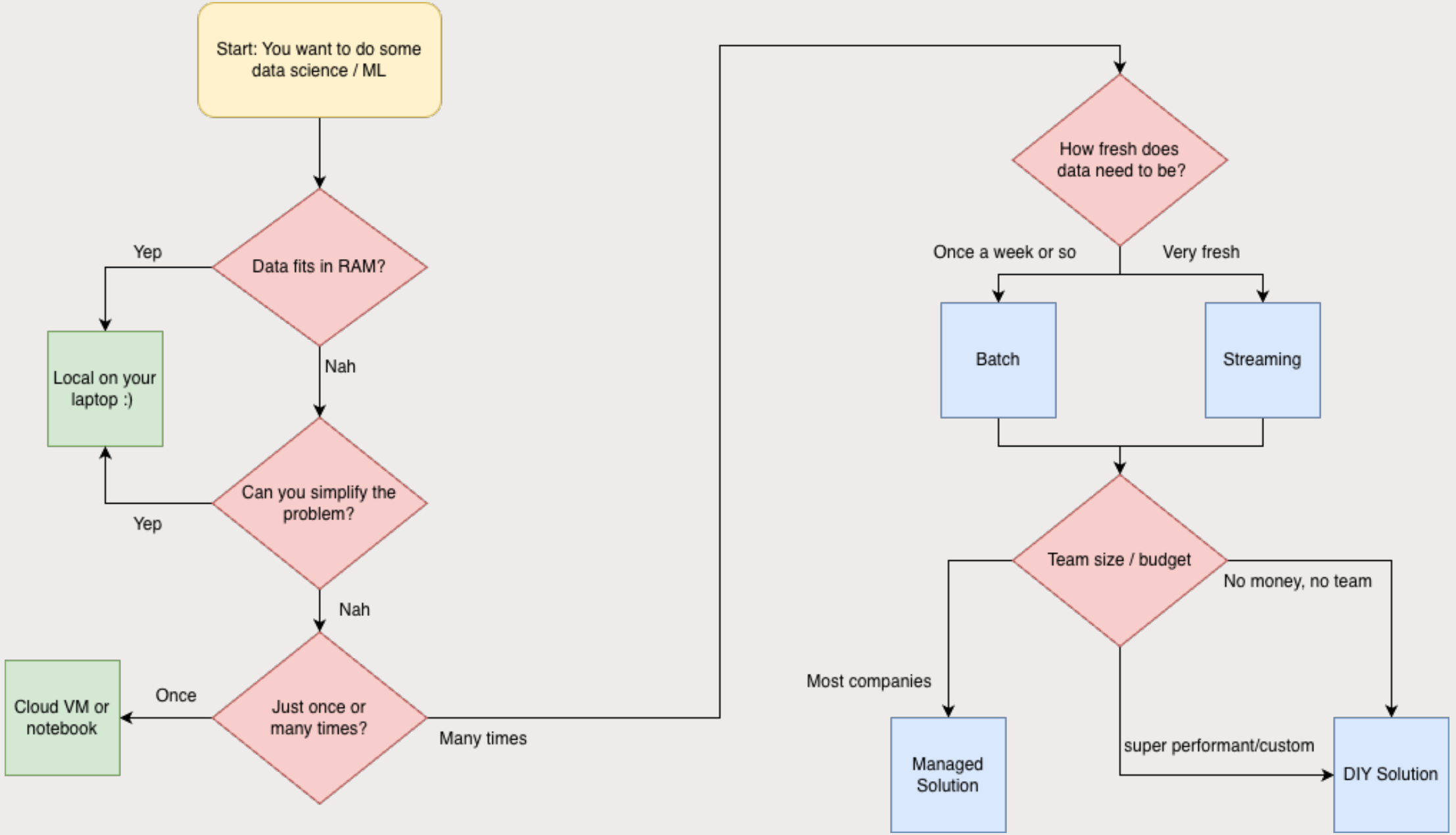
Example: At Balun, weather data → feature tables → ML models → forecasts. Every step depended on the last.

Managed Platforms

(AKA Data Warehouse / Datalake)

- Cluster management, monitoring, access control — handled
- Data catalog + scheduling included
- You pay in cost and lock-in

Right for: multiple teams, compliance, pipeline runs the business


Start: You want to do some data science / ML
Data fits in RAM?
Yep
Local on your laptop :)
Nah
Can you simplify the problem?
Yep
Nah
Just once or many times?
Once
Cloud VM or notebook
Many times
How fresh does data need to be?
Once a week or so
Very fresh
Batch
Streaming
Team size / budget
No money, no team
Most companies
Managed Solution
super performant/custom
DIY Solution

Live Walkthrough: 50M User Churn

Scenario: Build daily churn prediction for 50M users

1. Fits in RAM? No.
2. Can I reduce? Yes — active users only → 2M rows → back to local
3. Is it recurring? Yes.
4. How fresh? **Nightly batch is fine.**
5. Tables interdependent? Yes — features from raw events
6. Team size? 3 people → Just use Databricks or Snowflake

Lifecycle: local prototype → single-machine exploration → set up for everyone

THANK YOU!

Questions? Let's connect!

yury.tech