

M O

Welcome to Modern Data Systems: CS145

What do you learn?

The grey line is the systems technique you will learn.

● DATA STACKS FOR AI AGENTS

- 1 One text prompt, many valid SQL queries. Your LLM picks one. How do you know it's correct?
query ambiguity · unit tests · eval sets
- 2 Anthropic's text-to-SQL: 21% to 95% correct, no new model. What changed?
grounding · knowledge model · governed metrics
- 3 An agent forgets every session. What do you store so it remembers?
durable memory · SQLite · BM25 + vector retrieval
- 4 How do you find the nearest of a billion vectors, without a billion comparisons?
vector index · HNSW · graph search
- 5 An agent re-reads the same long prompt every turn. How do you stop paying for it?
KV cache · prefix reuse · paging

● READS AT SCALE · OLAP & ANALYTICS

- 6 How does Spotify search 100 million tracks without a scan?
inverted index · term dictionary · B+Tree
- 7 What takes the same query from \$550,000 a year to \$400?
index selection · I/O cost model · big-data economics
- 8 How does 'scan a petabyte' become 'scan three columns'?
columnar storage · Dremel · compression
- 9 How does Spotify Wrapped serve 700 million people in one day?
precomputation · clustered index
- 10 How does Chrome check every URL you open without shipping the list?
Bloom filter · hashing · false positives

● WRITES AT SCALE · OLTP & TRANSACTIONS

- 11 How does one Stripe charge become four writes, all or none?
ACID · atomicity · two-phase locking
- 12 How do you absorb a billion writes a day without random seeks?
LSM tree · MemTable · SSTables · compaction
- 13 How does one primary serve 800 million ChatGPT users?
read replicas · replication lag · fan-out
- 14 When do you actually need to shard? Later than you think.
sharding · one Postgres first · replicas
- 15 A transfer spans two shards. How do both commit, or neither?
two-phase commit · coordinator failure

● TRUST, SECURITY & STREAMING

- 16 Every nightly job ran green and the numbers were wrong. How do you catch it?
data quality · assertions · observability
- 17 Netflix released 'anonymous' ratings. How were the names put back?
linkage attack · re-identification · differential privacy
- 18 One concatenated string lets an attacker run their own DROP. What stops it?
SQL injection · parameterised queries · zero trust
- 19 A trillion messages a day, and a consumer can join years later. How?
Kafka · append-only log · consumer offsets
- 20 How do you build storage out of drives you expect to fail?
replication · commodity hardware · GFS

Four eras of data systems, compared on the same four questions

	Banks and retail 1970s TO 1980s	The web 1995 TO 2010	The cloud 2004 ONWARDS	AI agents 2025 ONWARDS
PRODUCTS	Ledgers, inventory, airline seats	Google, Amazon	Gmail, YouTube, Instagram	Copilots, agents, assistants
WHO IS ON THE OTHER END	Hundreds of clerks	Millions reading	Billions writing, at human pace	Machines, at machine pace
THE HARD QUESTION	Is it correct?	Can everyone read it at once?	Can everyone write and store forever?	Machines read and write nonstop. Is all of it correct, and what did it cost?
INVESTMENTS	Transactions, ACID, the relational model, locking, write-ahead logs	Replicas, caches, NoSQL, inverted indexes, the cluster, MapReduce	LSM trees, sharding, consistent hashing, object storage, quorums, columnar scans	Vector indexes, KV caches, eval sets, lineage, Iceberg, lakehouses

Each era stacks on the one before it. This course is built for the column on the right.

What you need on day one, and what helps a lot

WHAT YOU NEED

REQUIRED

Python and Colab

Read Python, run a notebook,
open a Colab and execute a cell.

```
df.head() → [10 rows]
```

HELPS A LOT

CS 161 · Algorithms

Sort, hash, complexity.
How an operation scales when
the input grows.

Modules 2 and 3 lean on it.

HELPS A LOT

CS 111 · Systems

Filesystems, the memory
hierarchy, scheduling.
Where the bytes actually live.

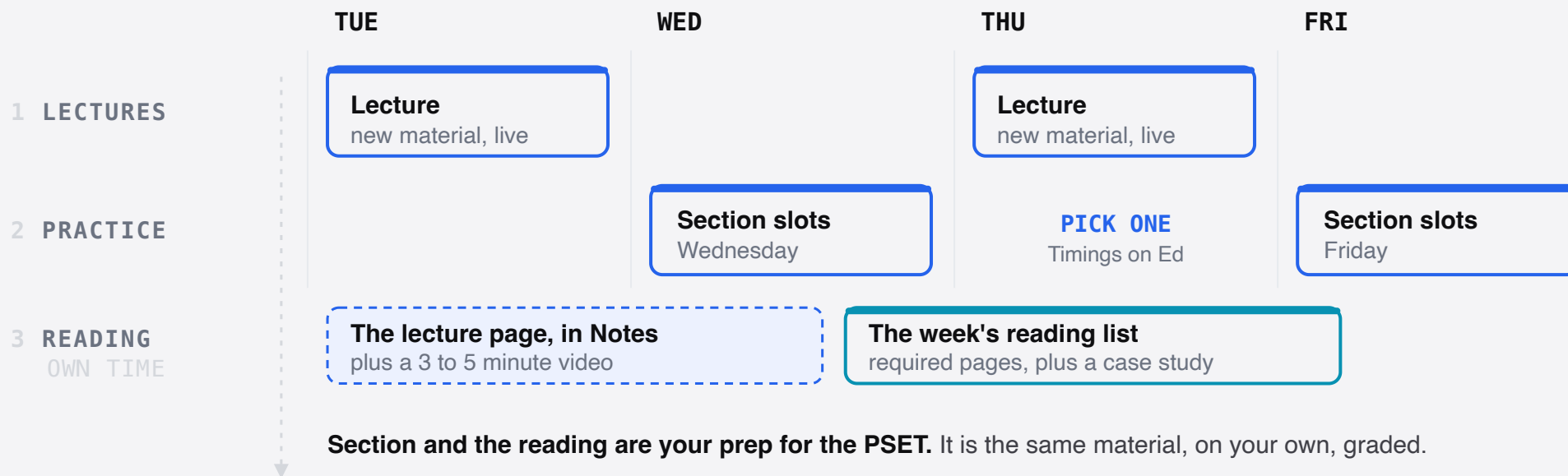
Module 2 leans on it.

CS145 takes those same concepts and runs them on data 100x to 1,000,000x bigger.

PSET 0 opens with ten questions on exactly these concepts. Start there, and scan ahead when one finds a gap.

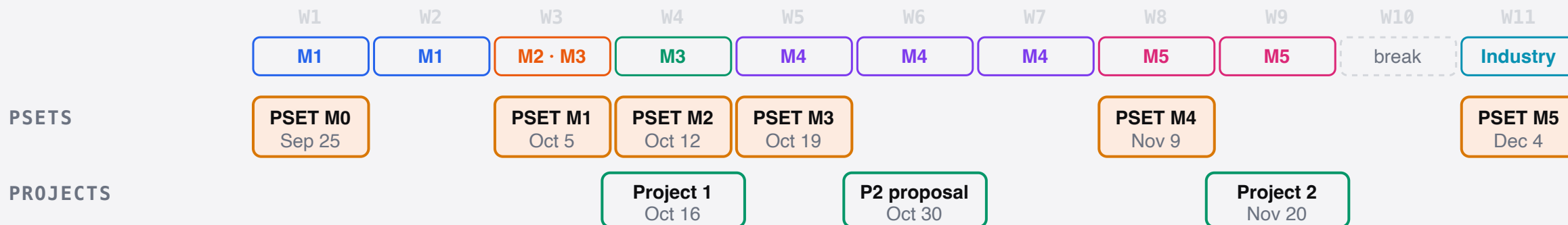
One week in CS145, and the eleven weeks it repeats across

ONE WEEK



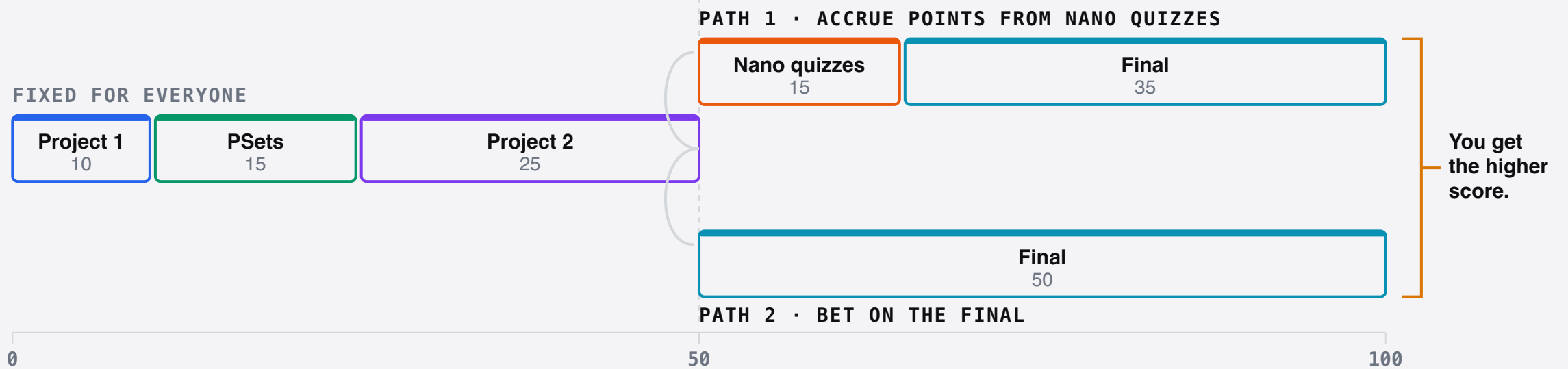
ELEVEN WEEKS

The module underneath changes. The rhythm stays the same.



One hundred points: fifty fixed, and fifty you can earn two different ways

YOUR GRADE, OUT OF 100



Teaching Team



Shiva
Instructor · CS145 '94 · Teaching since 2018



Jason Hu
Head CA · CS145 '24



Ananya Ganapathi



Aryan Katiyar



Ganesh Venu
CS145 '25



Jason Shin



Owen Jung
CS145 '25



Qirui Zheng



Yixiao Zhang
CS145 '25

M O

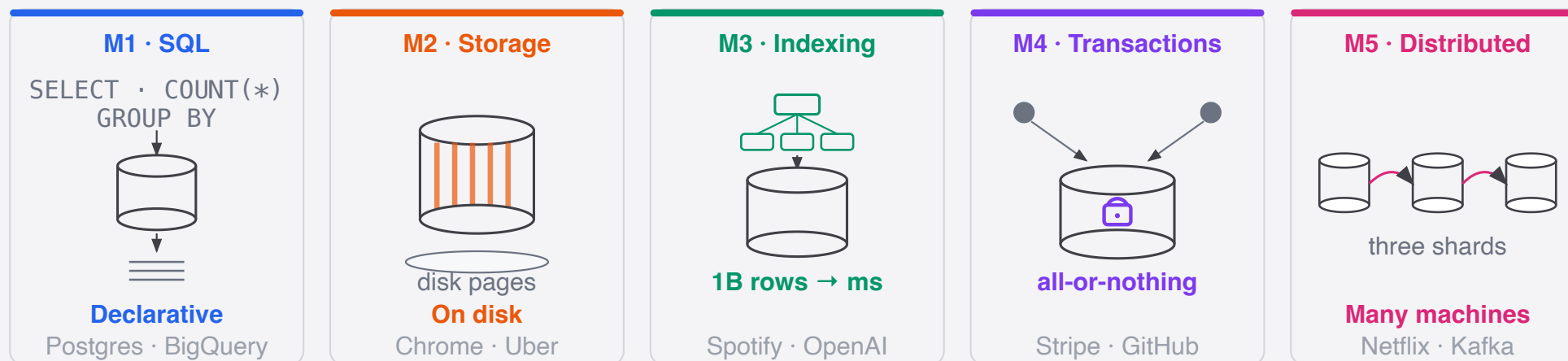
Why Memory Matters

Databases give applications durable memory: data survives
after a computation ends.

Every system is the same few ideas. You can build new ones.

The five you learn, then the modern systems you build with them.

ONE MACHINE → MANY MACHINES



Three systems, reconstructed

Working state, persistent state, then scale.

0.1 · Small analytics

Pandas starts in memory.
Polars can stream supported work.

one machine

the on-ramp

0.2 · Agent memory

Working context plus
persistent memory.

local → global

the whole course in one example

0.3 · A big service

Shared transactional state,
plus the disconnected edge.

billions of database transactions

planet scale and the edge

one machine

many machines

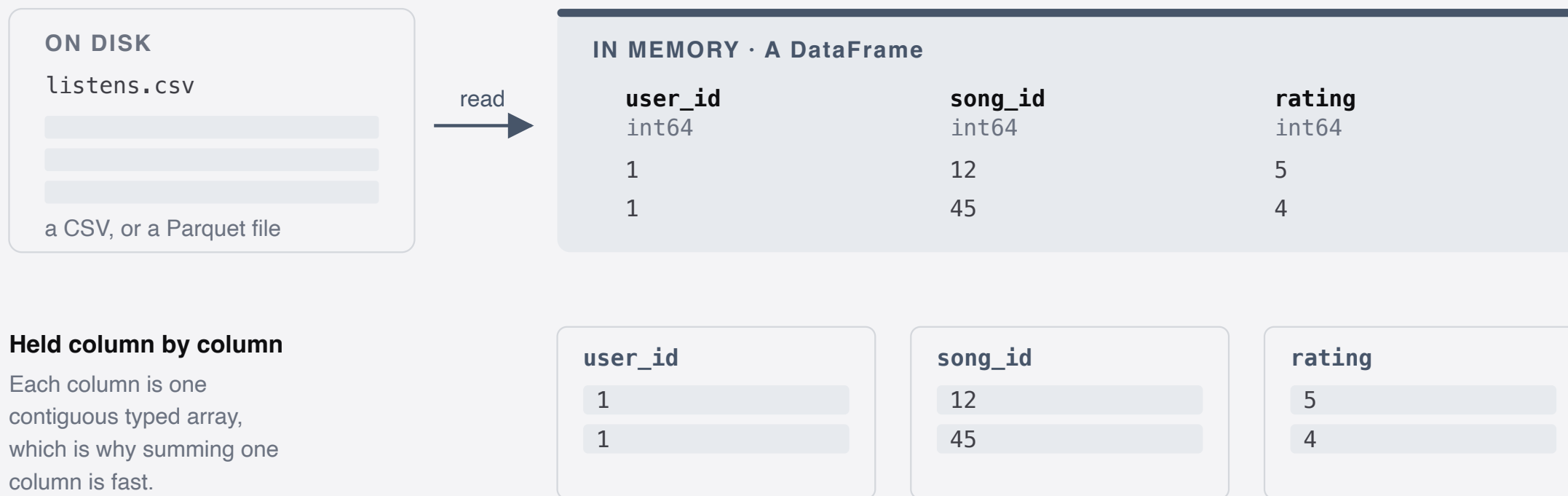
MO

Pandas: Deconstructing Small Analytics

Pandas normally materializes a DataFrame in memory.

How Pandas holds a DataFrame

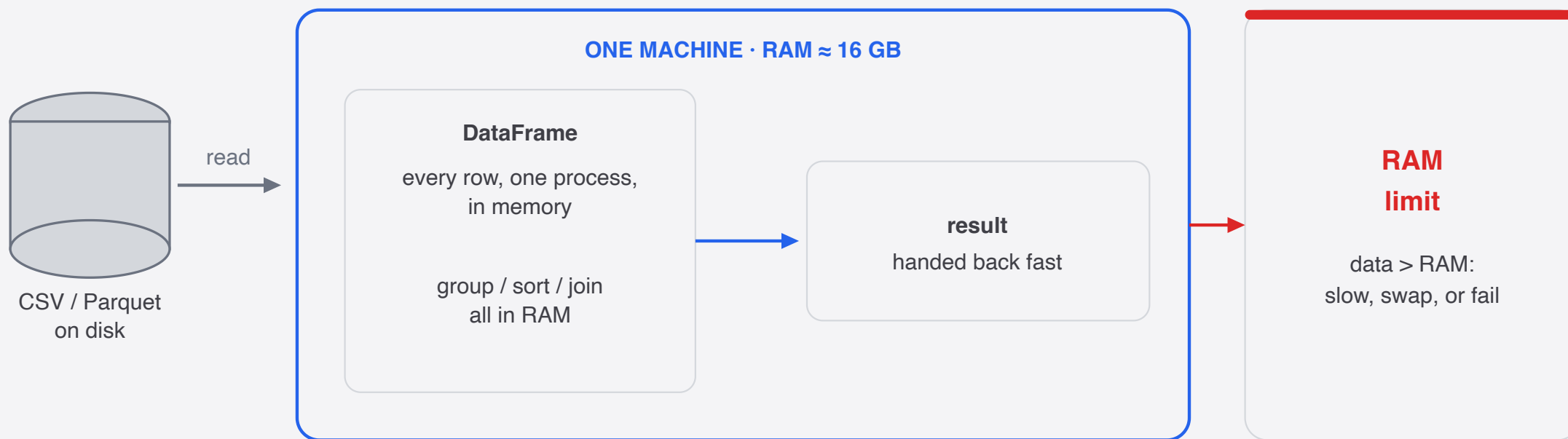
One file, read into memory, then held one column at a time.



Pandas materializes the DataFrame in memory before these operations run.

Pandas: the whole dataset has to fit in RAM

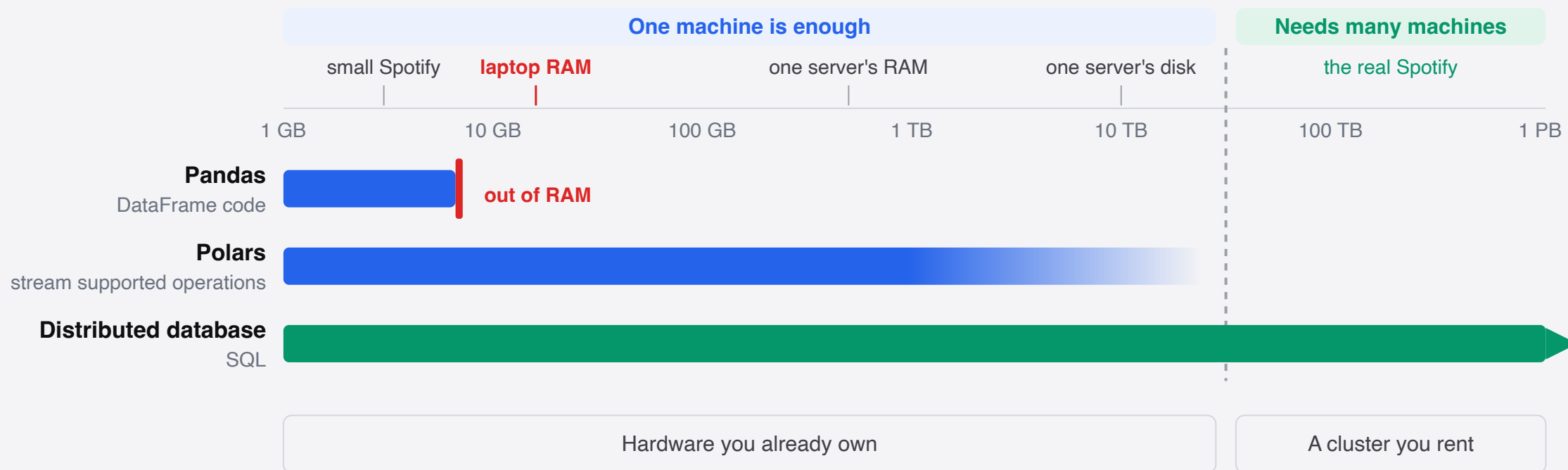
One process and one machine bound the available working memory.



Color key blue = the one machine in focus · red = memory pressure · grey = out of focus

Same query. The data keeps growing.

Illustrative ranges. The actual limit depends on the query and hardware.



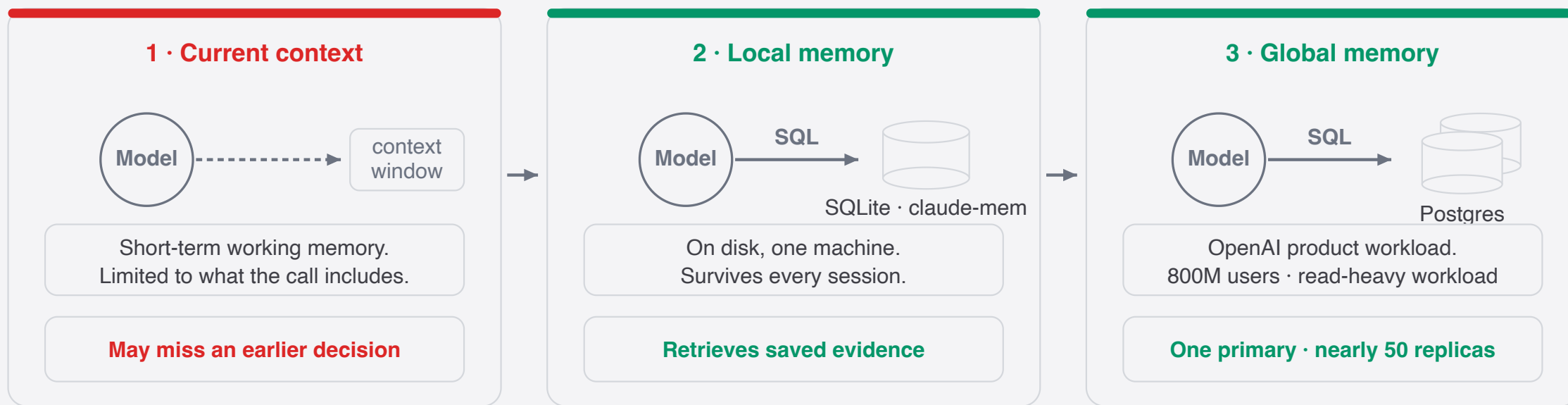
M O

Claude & OpenAI: Deconstructing Agent Memory

A model sees only the context supplied for its current call.

Agent memory: local retrieval to a global service

Applications persist memory outside the model and retrieve it when needed.



Context is bounded · memory persists · retrieval reconnects them

Color key **red** = no memory, the ephemeral context window · **grey** = the model, a separate topic, out of focus · **green** = persistent memory

A plausible answer is still wrong

Without retrieved evidence, the agent predicts.

Tuesday: "extend the auth module we built on Monday."

THE AGENT GUESSES

The earlier decision is absent.

```
verifyJWTWithScope()
```

A name that sounds exactly right.

IN YOUR CODEBASE

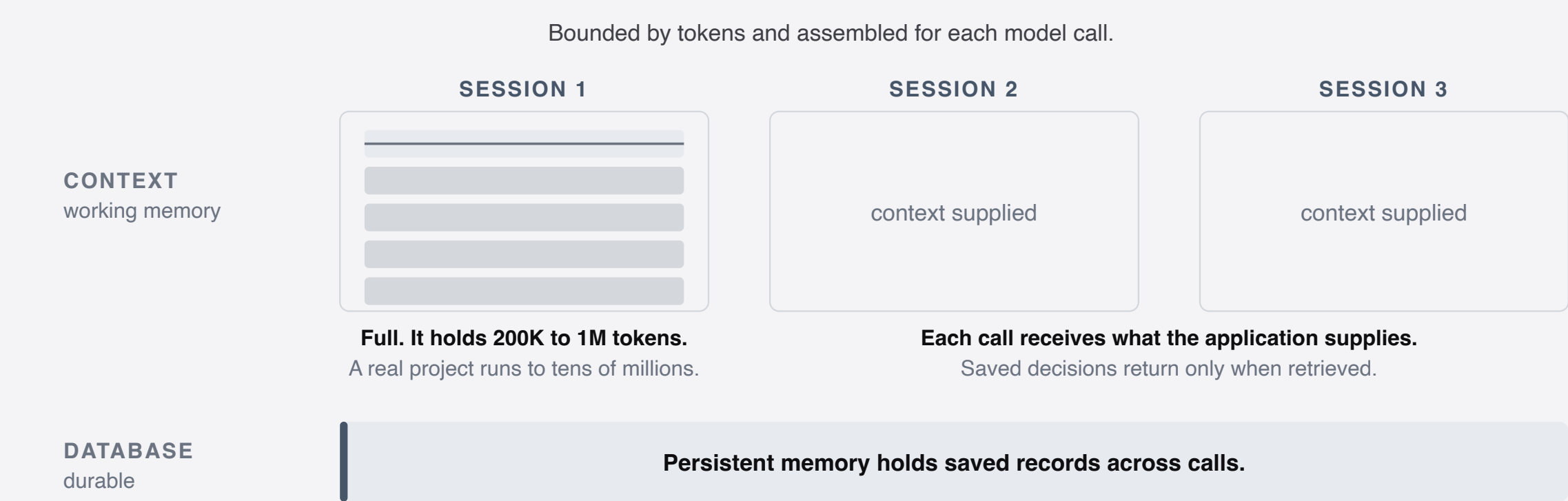
Every function you actually wrote:

```
no such function
```

It never existed anywhere.

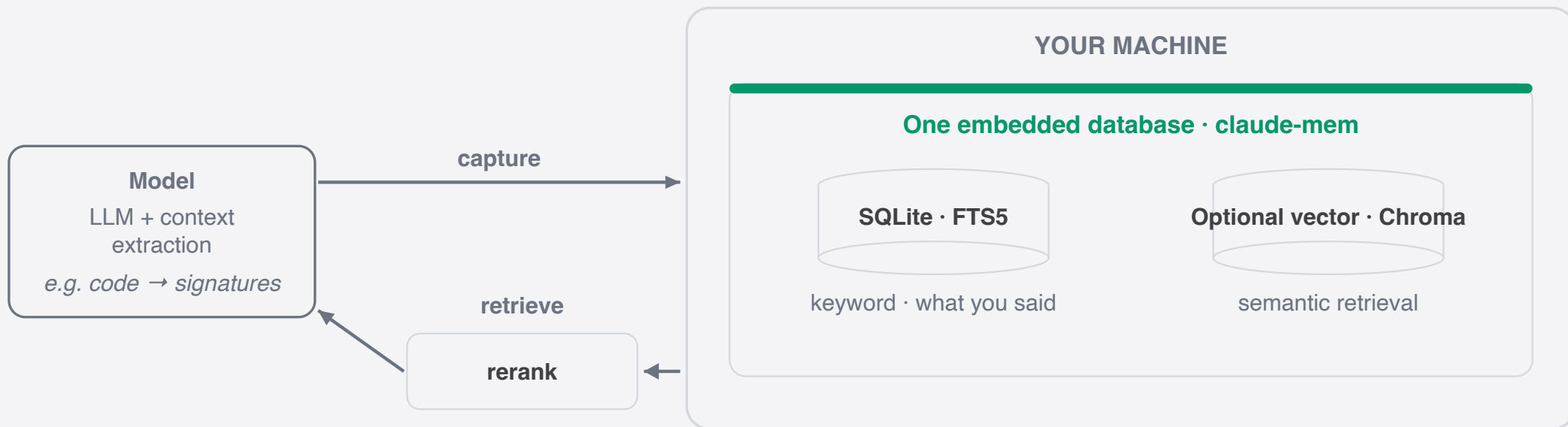
Persistent memory gives a later call evidence it can retrieve.

The context window is working memory



Local memory: capture, store, retrieve

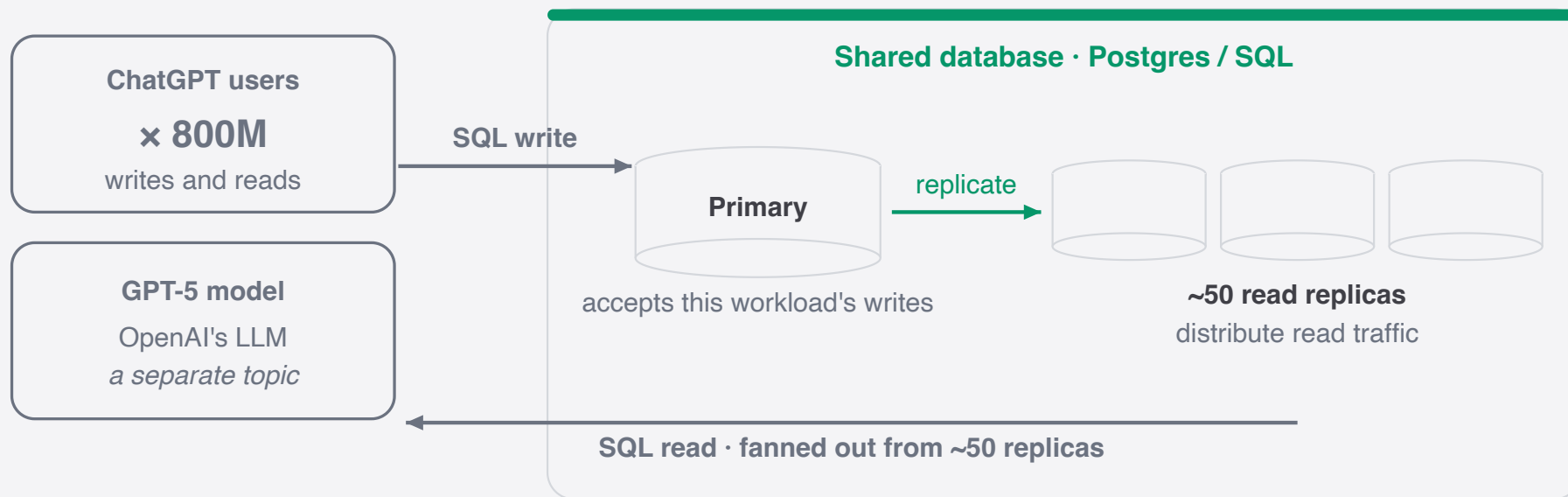
claude-mem stores records in SQLite and retrieves selected context later.



Color key grey = the model and your machine, out of focus · green = the persistent database

Global memory: one logical database, many machines

One primary and nearly 50 replicas support a read-heavy workload for 800 million users.



Color key grey = the users and the GPT-5 model, out of focus · green = the shared Postgres database

Three signals for selecting context

Text, meaning, and code structure answer different retrieval questions.

1 · WHAT YOU SAID

BM25

Keyword match on the text,
the words you actually typed.

```
search "auth"  
rows containing auth
```

2 · WHAT YOU MEANT

vector similarity

Embeds the question, finds
rows pointing the same way.

```
search "login"  
also finds auth
```

3 · WHAT MATTERS NOW

graph ranking

An importance signal over
your code: what points at it.

```
auth.ts  
40 files import it: ranks high
```

A retrieval pipeline can combine these scores.

The highest-ranked evidence enters the next model context.

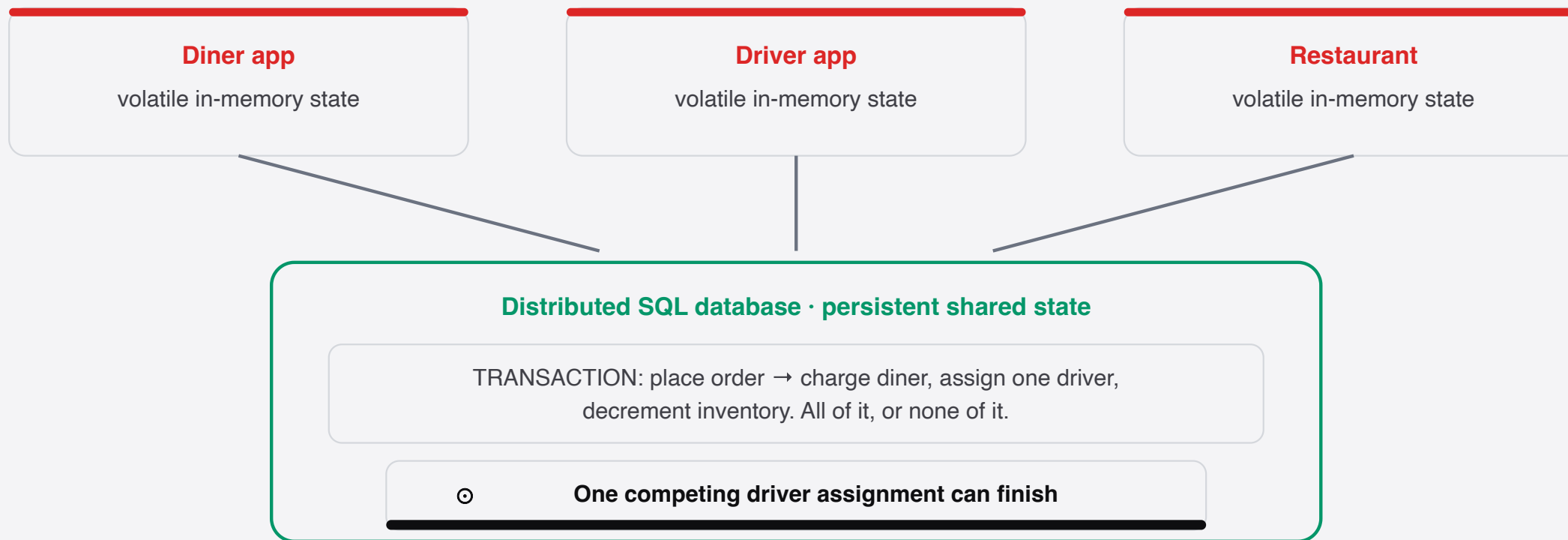
M O

UberEats: Deconstructing a Big Service

UberEats must keep the diner, driver, and restaurant in agreement while the service runs across many machines.

UberEats: shared state for three sides

A burrito order has three sides. They all have to agree the moment it is placed.



Color key **red** = temporary client state · **green** = persistent shared state

One order. Three records. All or none.

Atomicity, the A in ACID.

One diner places one order

ONE TRANSACTION

DINER

balance charged

DRIVER

exactly one assigned

RESTAURANT

inventory decremented

COMMIT

All three apply together.

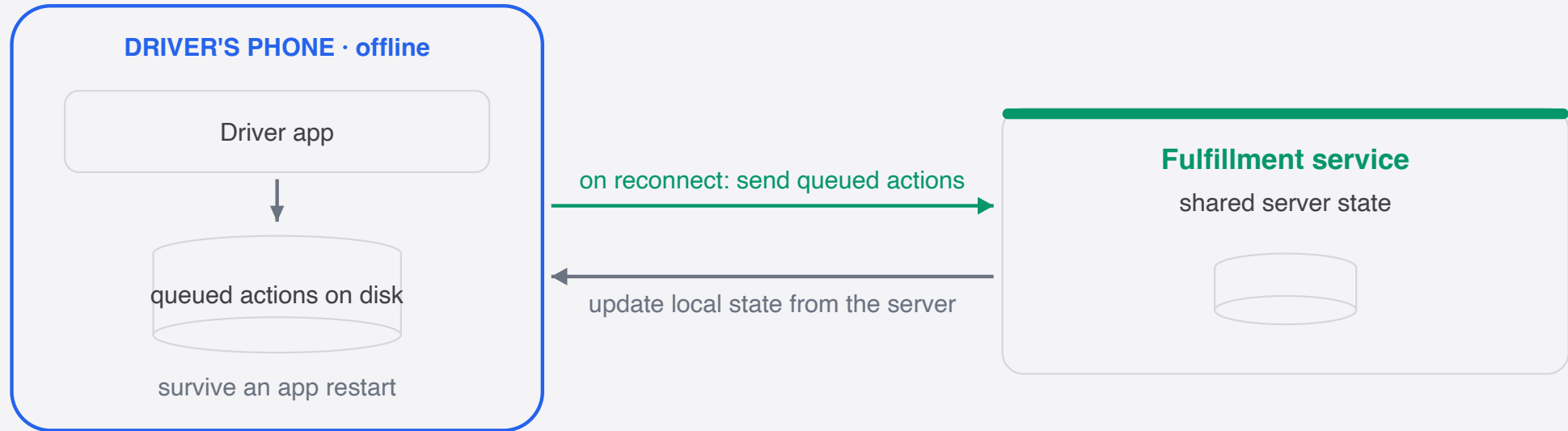
ANY ONE FAILS

None of them do.

Related changes apply together. A conflicting driver assignment cannot also finish.

The disconnected edge: save now, send later

The driver app keeps moving through a short network failure.

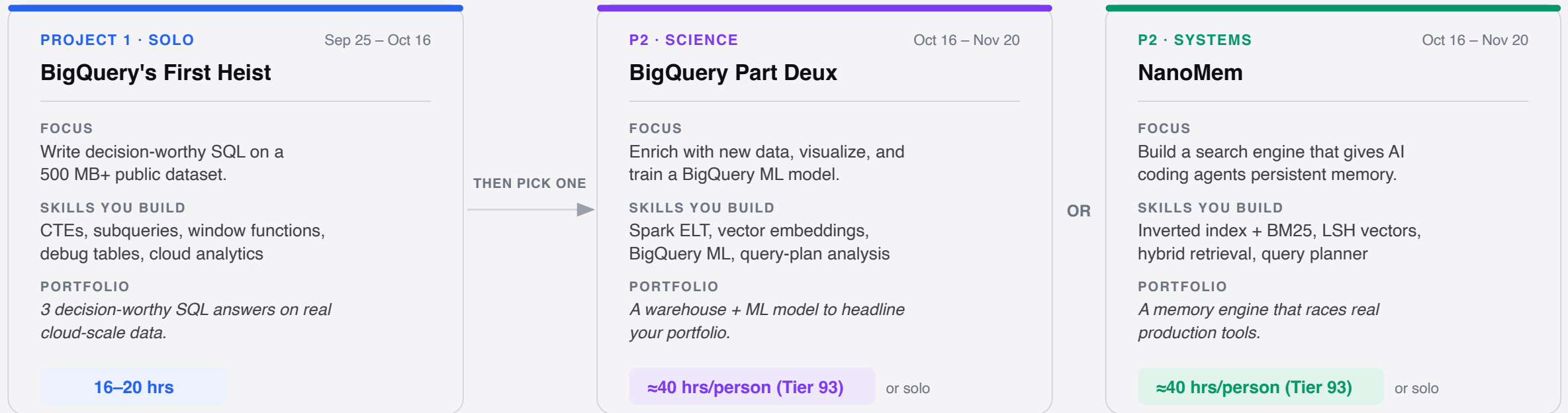


Color key blue = the phone, the system in focus · green = shared server state

MO

Projects

The project path: Project 1, then one of two Project 2 options



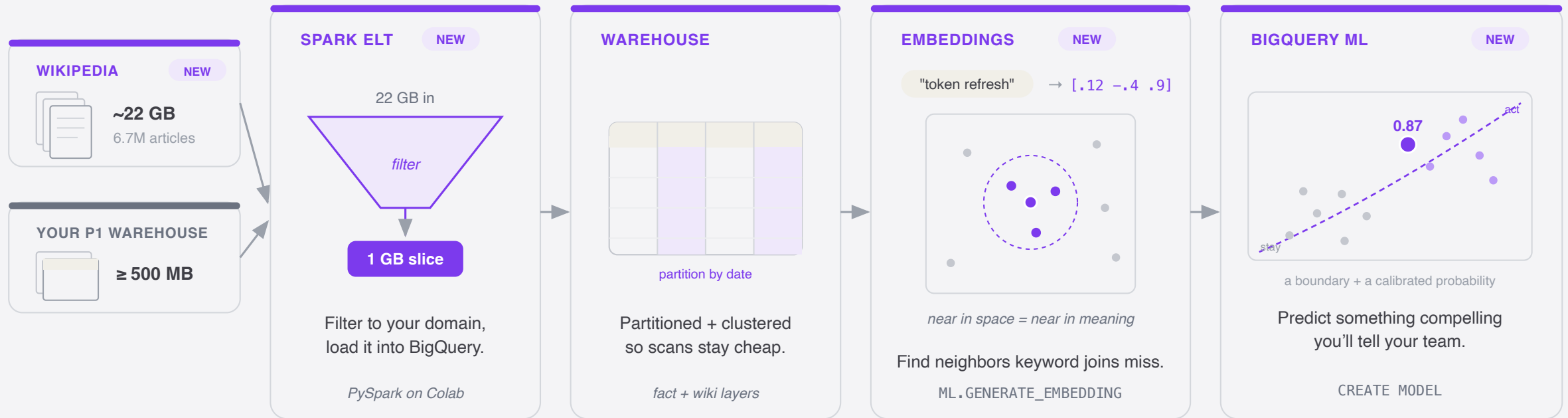
Project 1 architecture: query a public dataset with SQL to answer questions

A read-only public dataset, real SQL, one insight worth acting on.



Project 2 Science architecture: a production data and ML pipeline

Ingest outside data, embed it, model it, ship a calibrated decision.



violet = new in Project 2 (Project 1 was just SQL) · grey = the data you start from

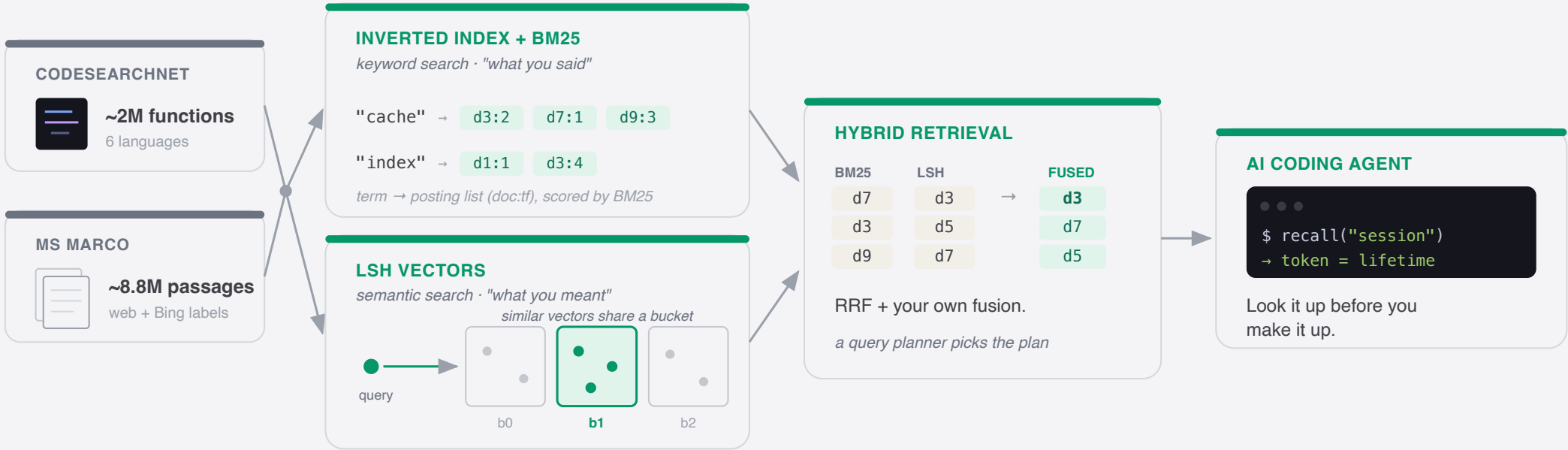


TIER 100 · THE CROWN

The Analyst's Agent · a governed text-to-SQL agent

Project 2 Systems architecture: a hybrid search engine for AI-agent memory

Two data sources, two kinds of search, fused, then raced against production tools.



THE BAR race the production tools for parity, same data, same eval

metric	your engine	reference tool	parity
R@5 keyword	0.71	FTS5 0.72	✓
R@5 vector	0.68	sqlite-vec 0.69	✓

+ R@1, MRR, p95 latency, peak memory

green = what you build · grey = the data you index



TIER 100 · THE CROWN

Web-scale memory · 22 GB Wikipedia, 1 GB working set

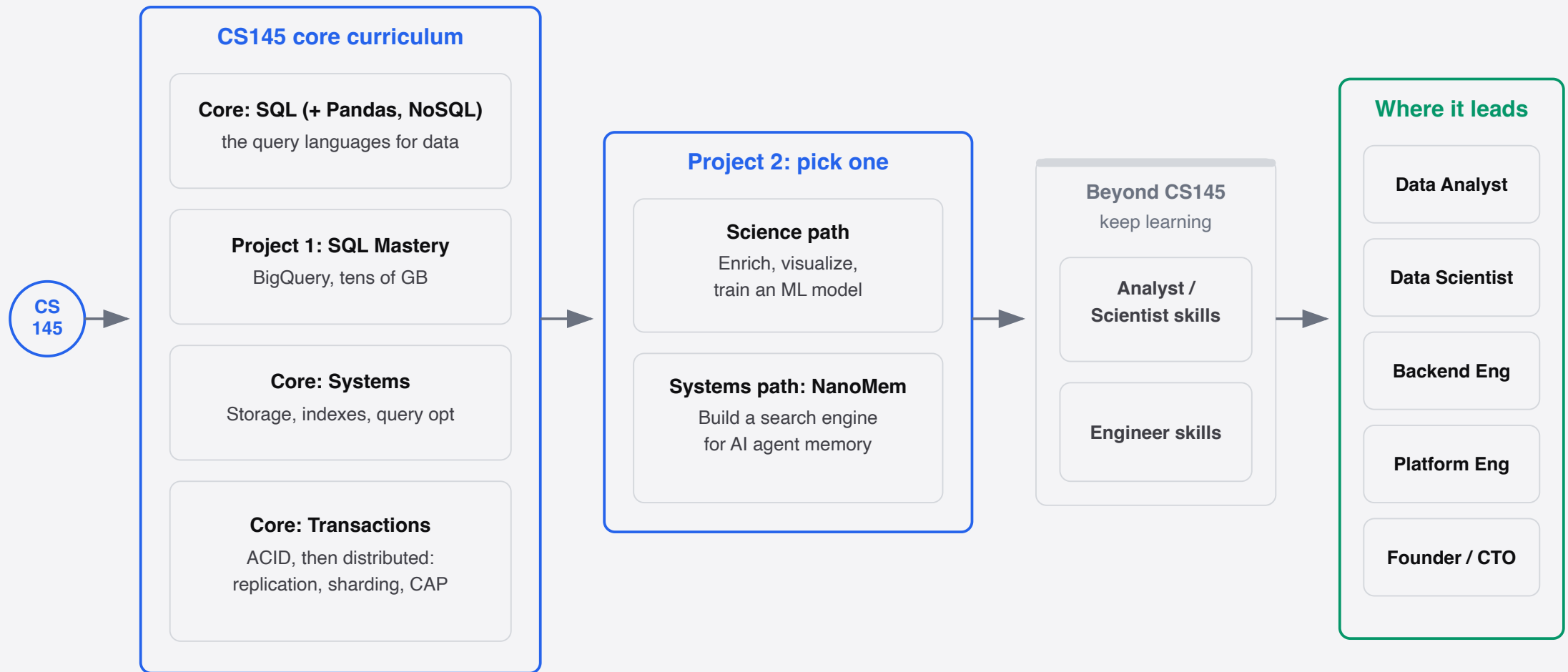
MO

CS145: Your Data Systems Journey

CS145 connects SQL to the storage, indexes, transactions,
and distributed systems that execute it.

CS145: Your Data Journey

Master the core, pick a technical path, build a career.

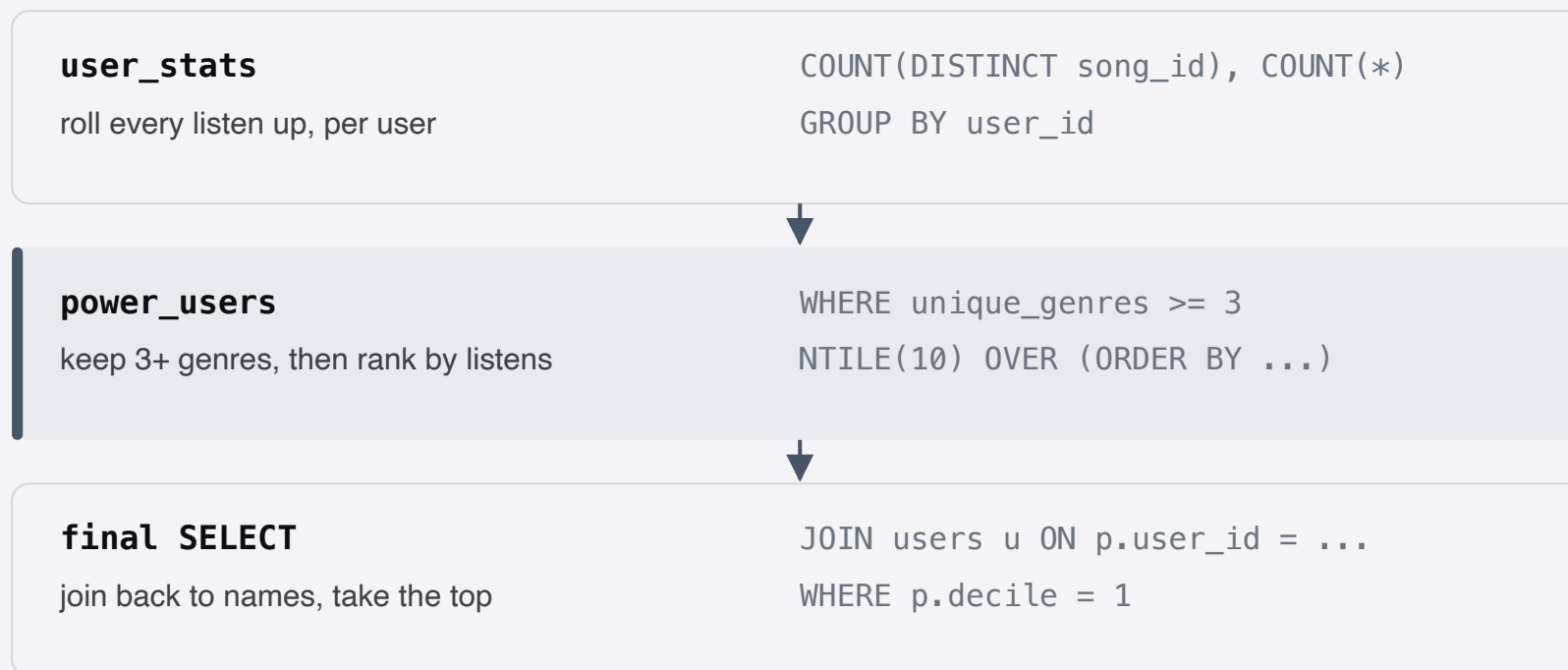


Color key **blue** = CS145 content · **green** = where it leads · **grey** = beyond CS145

You'll read, debug, and write queries like this

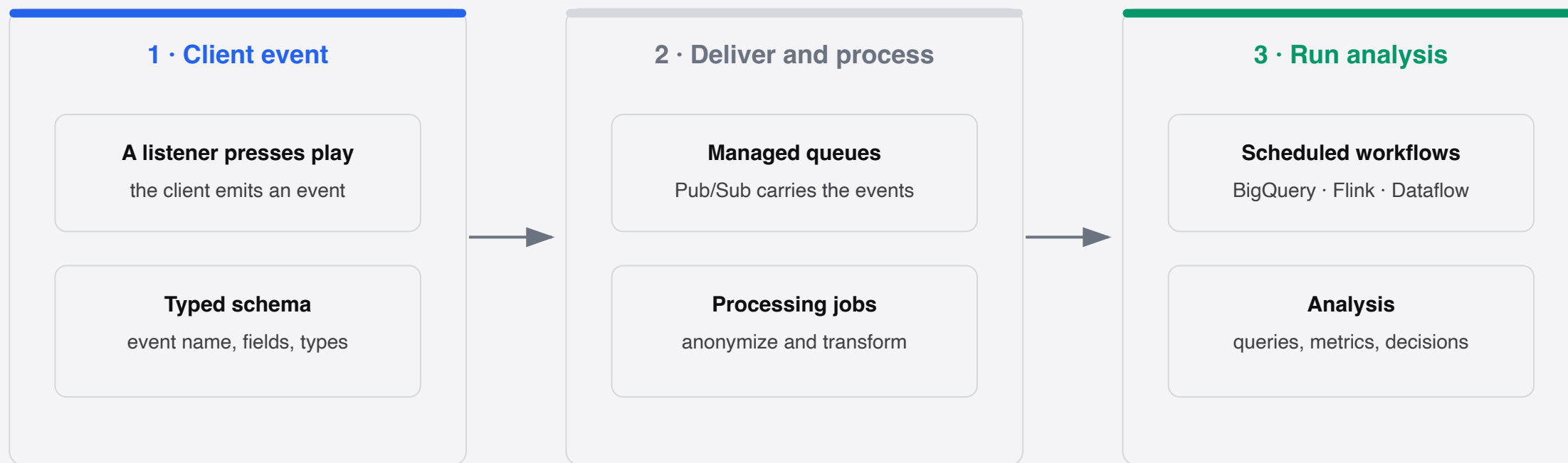
One query, three parts

The shape you will read, debug, and write.



Spotify: from a client event to analysis

Collect typed events. Process them reliably. Make the results available for analysis.



Color key **blue** = collection · **grey** = delivery and processing · **green** = analysis