MotherDuck

(v1.1)

DATA WAREHOUSES ARE GILDED CAGES.

WHAT COMES NEXT ?

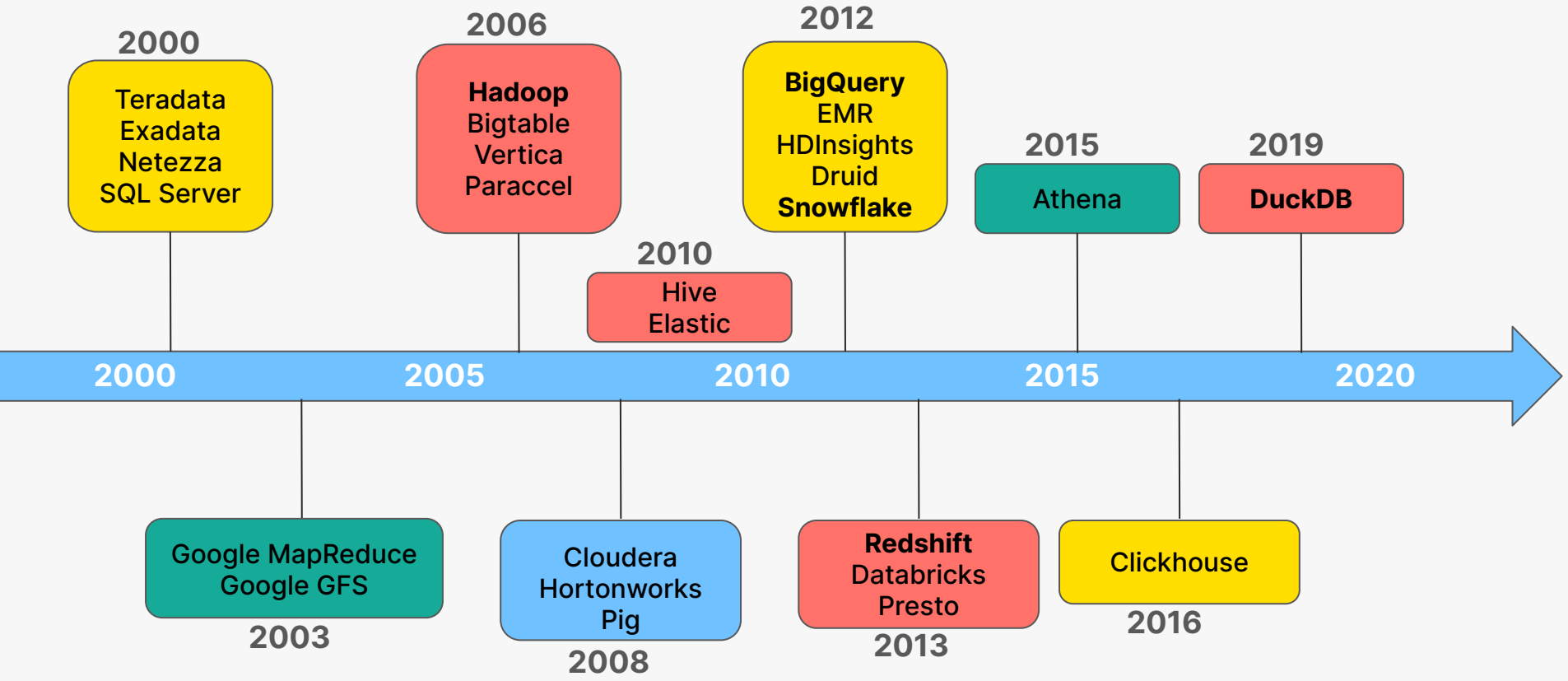
Tino Tereshko (in lieu of Nick Ursa)

Co-Founder & Head of Produck, MotherDuck

Agenda

1. My experience of the last 15 years of analytics tech developments
2. Some luke warm takes. Please no one sue me.
3. We note a shift in hardware
4. A possible different future

End of History for OLAP ?


2000
Teradata
Exadata
Netezza
SQL Server
2006
Hadoop
Bigtable
Vertica
Paraccel
2012
BigQuery
EMR
HDInsights
Druid
Snowflake
2015
Athena
2019
DuckDB
2010
Hive
Elastic
2000
2005
2010
2015
2020
Google MapReduce
Google GFS
2003
Cloudera
Hortonworks
Pig
2008
Redshift
Databricks
Presto
2013
Clickhouse
2016

The declining value of your average row

Nominal Value	***A Row that denotes***	***Human cost***	***Storage System***
$500,000	House Purchase	1/2 lifetime of effort	The Legal System
$50,000	Broker Stock Purchase	Years of savings	Mainframes in New Jersey protected by armed guards
$500	Airline Flight	A Paycheck	Blue Chip OLTP
$50	Marketing Lead	An hour of thinking about a product	OLTP
$5	Retail Purchase line item	1 minute decision	OLTP
$0.10	Ad Click	A few seconds	OLAP / NoSQL
$0.001	Ad View	The fifth of a second until you realize you can ignore it. Input into P(action)	Big Data
$0.000010	Real Time Bid for that Ad view	Input into P(View)	Big Data.
$0.000001 ??	Telemetry from a free phone game	Input into P(Keep_playing)	Big Data.

Data Warehouses made Distributed OLAP usable

Big Tech: had the resources to produce with open source Big Data. They could contribute patches, whole teams dedicated to what you, as a mid size co data engineer, had to learn on the fly.

Smaller data orgs hat to do their best. Needed better reliability and usability than those system. So the invisible hand kicked into gear.

Data Warehouses are becoming everything we asked for

- They took scale limits away
- Easy to use distributed systems. No OOMs. (Athena .. Presto)
- Scale up and down elastically (to varying degrees)
- Handle unstructured and semistructured data
- Data movement behind the scenes for us for locality
- Caching as needed, and cache expiry (the hard part)
- They have industry standard interfaces: JDBC, ODBC, postgres wire
- Some have facilities for easy streaming ingest
- They can also interoperate with your lake, external tables in S3
- Optimize your workloads behind the scenes

How will they grow ?

Are Data Warehouses... Mainframes ?

DWs as Mainframes

- You send text -> they send back results
- Everyone competes for central resources
- moving all your compute to a walled garden
- SaaS ecosystem reinforces this dynamic, cheaper for them to integrate once w/ cloud vendor

Let's compare to app developers

Data teams are kind of unique culture. Batch vs transaction, centralized state.

- App devs often can spin up whole stack locally
- They can run their database in docker
- Even AWS Lambdas have emulators...
- Local testing that's fast

Meanwhile...

- Can't test truly locally. can't even verify syntax locally !
- Fully Saas = mishmash of cloud based editors
- Dev DW environments often end up with stale schema


This is Your
Career on Data Warehouses

Vendorization of data engineering

2010 data engineer

- Java SWE / SRE-ish
- Understood how the systems worked, kinda
- Creating needed DB abstractions (sometimes reinventing the wheel)

2022 MDS data engineer

- Build v. buy often going to buy
- Glue together SaaS offerings
- Looks a lot like a cloud-ish DBAdmin from 2000s. Vendor jails.

But this is just the way automation goes, right ? Technology gets better, we move up the abstraction layer, more people can participate.

Bills.

Federal Funds Effective Rate (FEDFUNDS)

DOWNLOAD

Observation:
Feb 2023: **4.57**
(+ more)
Updated: Mar 1, 2023

Units:
Percent,
Not Seasonally Adjusted

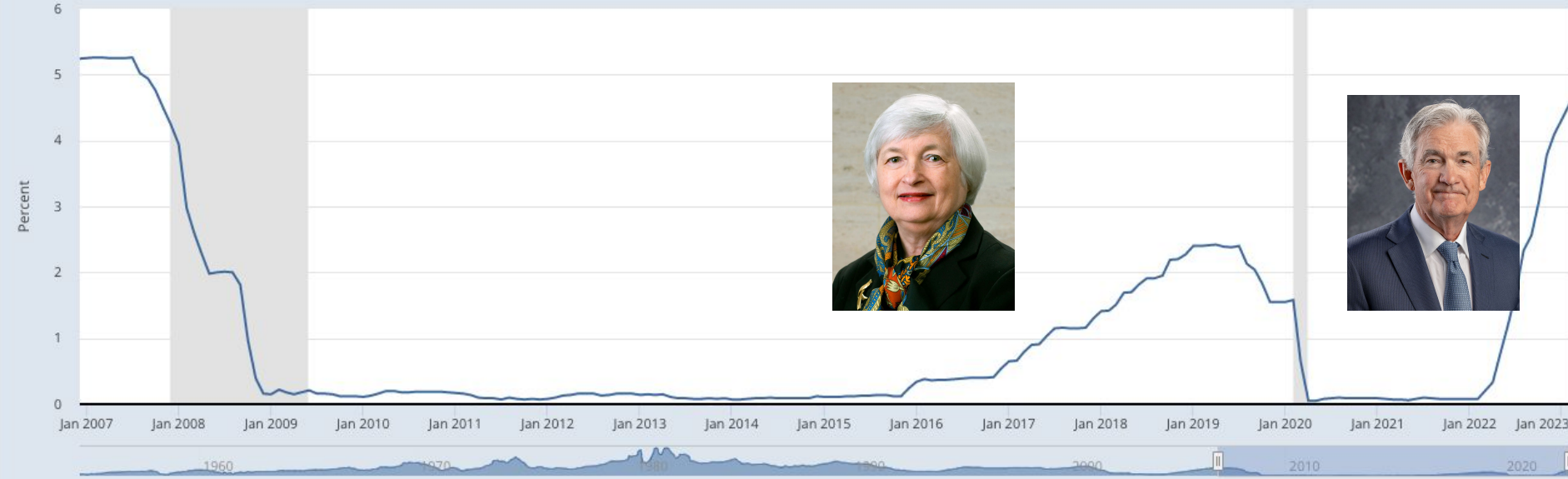
Frequency:
Monthly

1Y | 5Y | 10Y | Max

2006-12-05 to 2023-02-01

EDIT GRAPH

FRED — Federal Funds Effective Rate



Shaded areas indicate U.S. recessions.

Source: Board of Governors of the Federal Reserve System (US)

fred.stlouisfed.org

SELF SERVICE BI WITH ACCESS TO RAW DATA IS A **LOW INTEREST RATE PHENOMENON**

WHY ARE BALLOONING COSTS SO COMMON NOW ?

Why are ballooning costs so common now ?

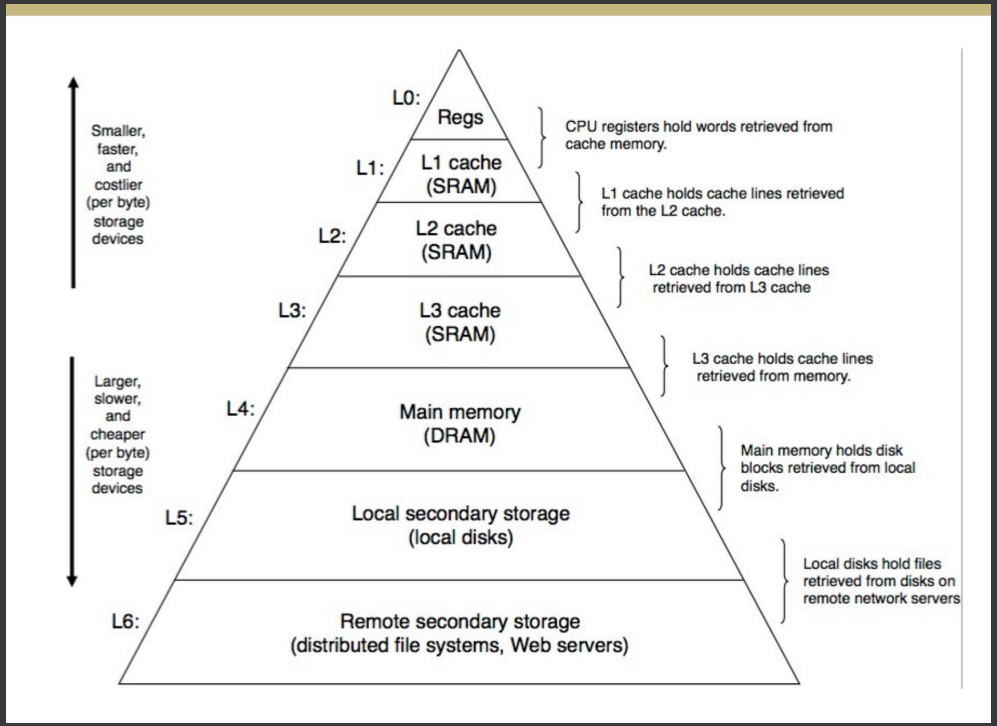
- Easy cloud scale up = kick the can down the road
- UX in tools favor accretion, not deprecation
- ELT
- Linear increase in data = Exponential cost in joins
- Self-service
- dbt: bias to whole table creation, ease of model proliferation
- Political economy of analytics teams
- Costs not exposed

THE MDS IS TURNING US ALL INTO DATA CONSUMERISTS

An Ode To Moore's Law *

(and how it relates to analytics)

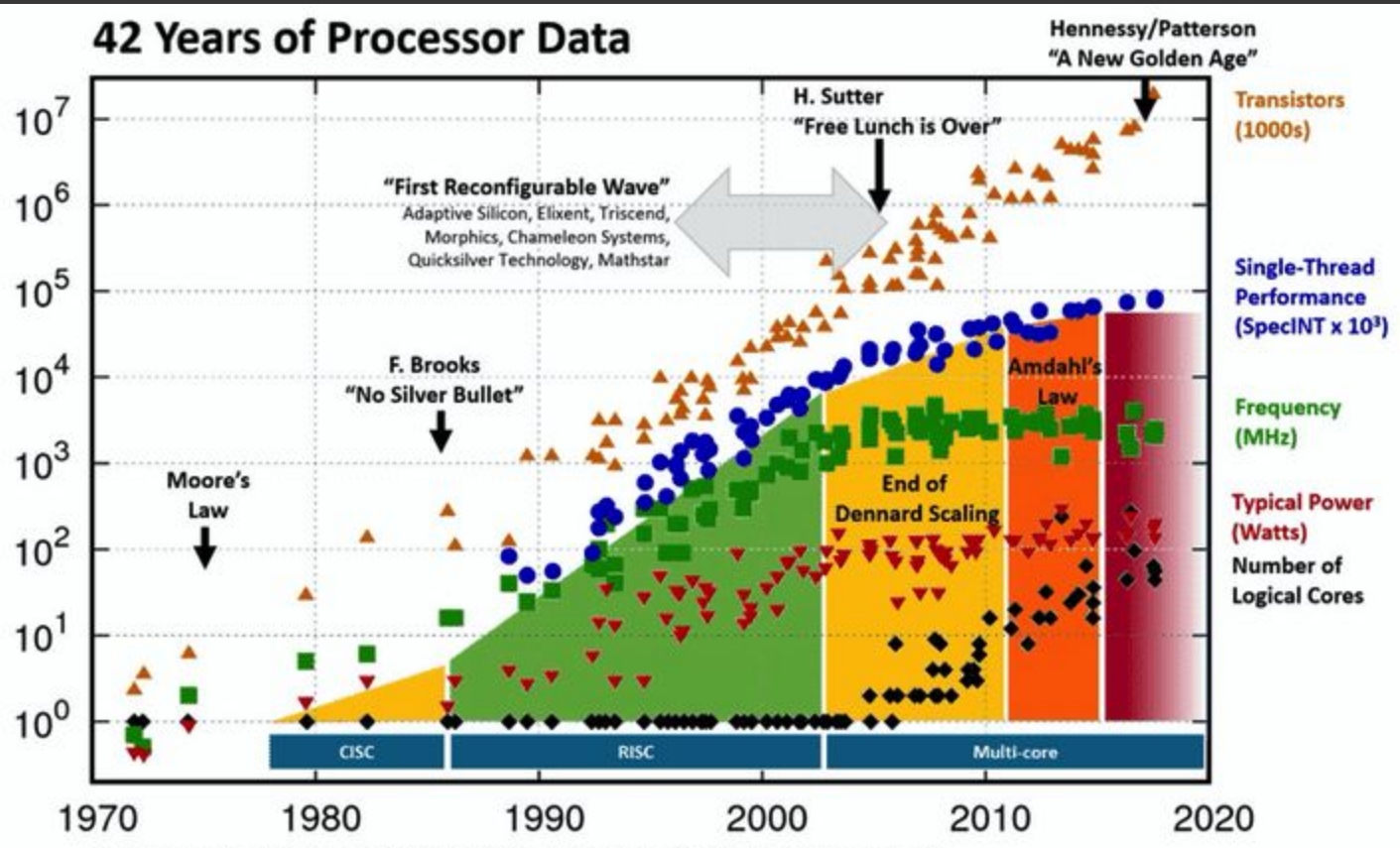
OLAP is primarily I/O bound,
Sequential access
Read heavy



What were the original motivations for distributed systems for data ?

- avoid Licencing costs for commercial DBs
- aggregate scan bandwidth over all nodes (If S3, NICs, if HDFS, disks)
- aggregate compute over all nodes when parallel possible
- cheap commodity hardware

But distribution didn't come for free. Method calls become RPC network calls, pointers become serialization. Most non trivial plans require shuffles.



Hennessy and Patterson, Turing Lecture 2018, overlaid over "42 Years of Processors Data"
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/;"First Wave" added by Les Wilson, Frank Schirrmeister
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

What *has* improved ?

Parallel execution:

- more cores w lower power each
- deeper pipelines, multiple operations at once
- branch prediction
- automatic vectorization

Caching:

- L1/L2/L3 caches bigger
- preloading data into caches

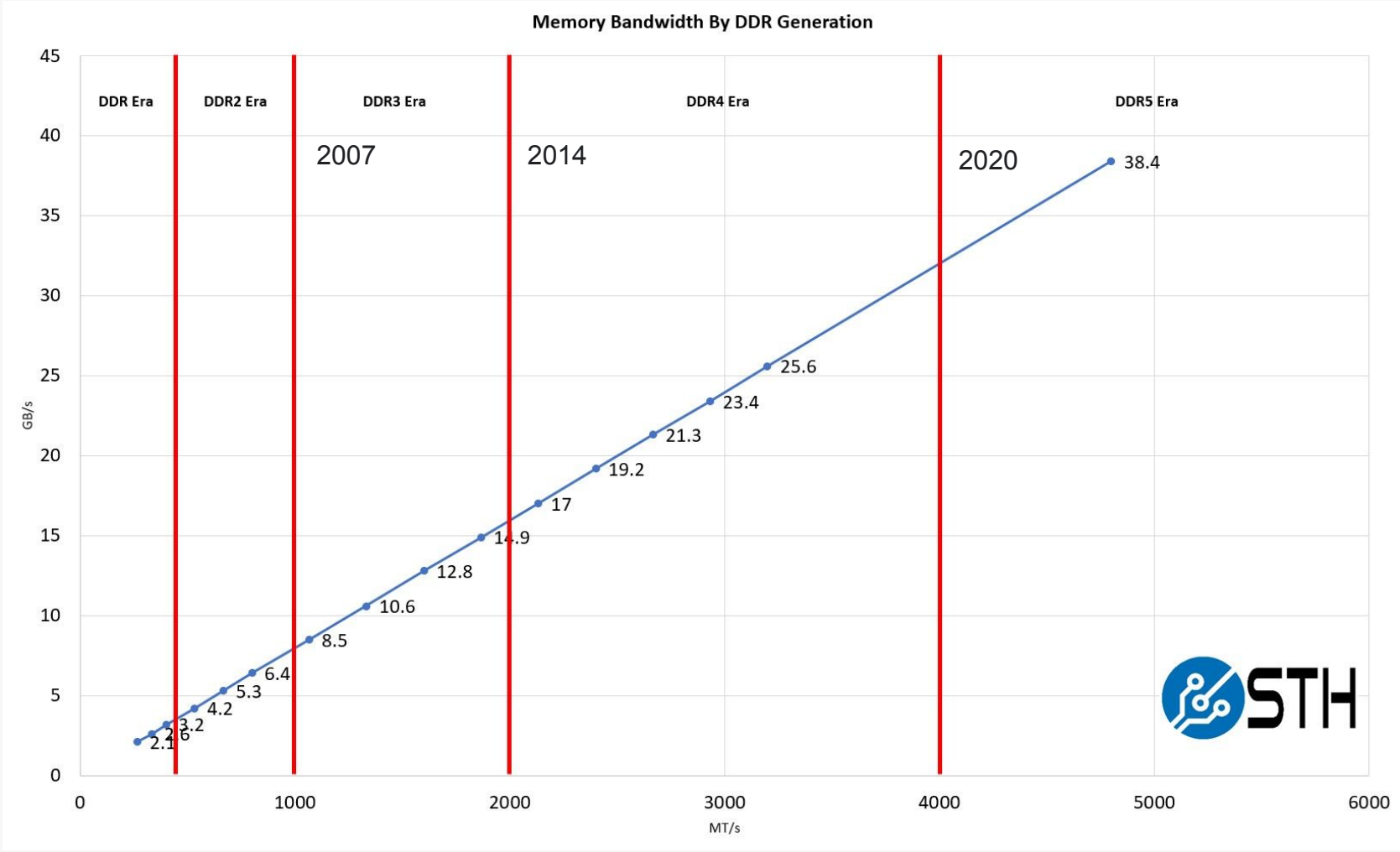
I/O

- memory bandwidth (L1-3 and DRAM)
- SSD cost
- DRAM Speeds
- cost of high performance networking. 10Gbit common, 100Gbit possible

OLAP is primarily I/O bound, Sequential access

```
for (idx_t i = 0; i < count; i++) {
    auto lentry : LEFT_TYPE  = ldata[LEFT_CONSTANT ? 0 : i];
    auto rentry : RIGHT_TYPE  = rdata[RIGHT_CONSTANT ? 0 : i];
    result_data[i] = OPWRAPPER::template Operation<FUNC, OP, LEFT_TYPE, RIGHT_TYPE, RESULT_TYPE>
        fun, lentry, rentry, mask, i);
```

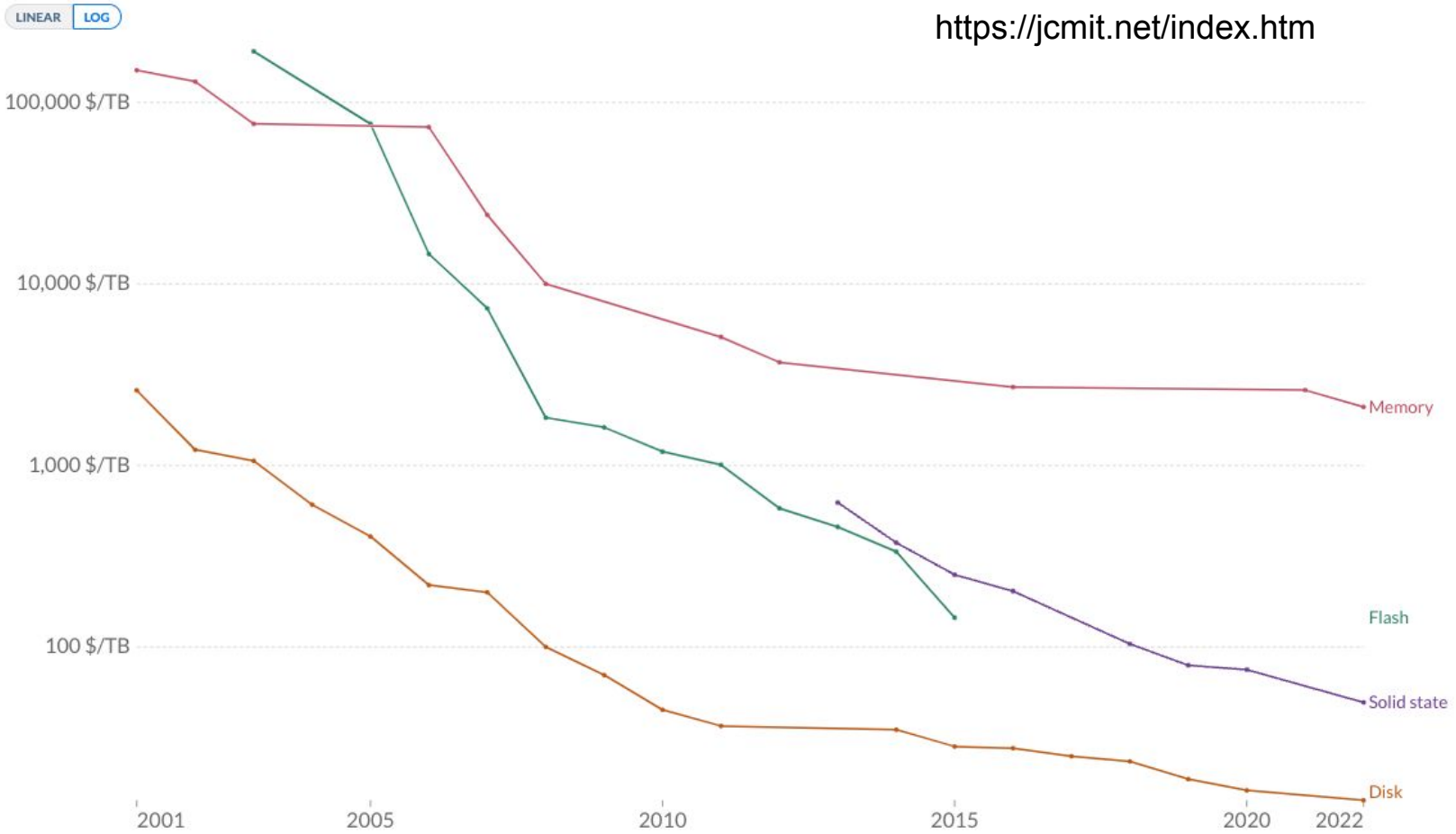
```
#define STANDARD_VECTOR_SIZE 2048
```

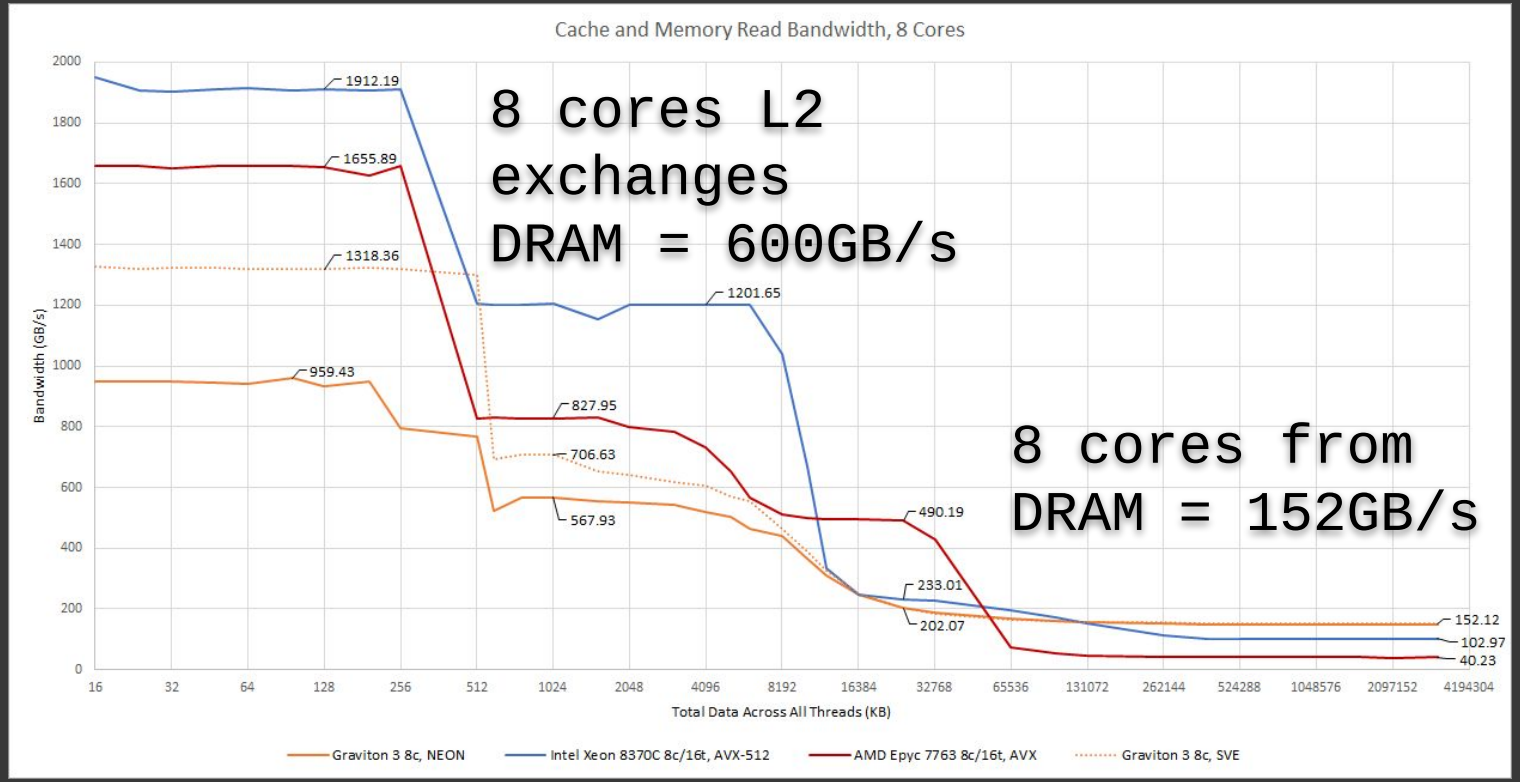

Memory Bandwidth By DDR Generation
DDR Era
DDR2 Era
DDR3 Era
DDR4 Era
DDR5 Era
2007
2014
2020
2.1
2.6
3.2
4.2
5.3
6.4
8.5
10.6
12.8
14.9
17
19.2
21.3
23.4
25.6
38.4
GB/s
MT/s
0
5
10
15
20
25
30
35
40
45
0
1000
2000
3000
4000
5000
6000
STH

Historical cost of computer memory and storage

This data is expressed in US dollars per terabyte (TB). It is not adjusted for inflation.

Our World in Data





https://chipsandcheese.com/2022/05/29/graviton-3-first-impressions/

AWS then and now

m7g.8xlarge
$1.30/hr in 2023 $

- Graviton3
- 32 cores
- L1 64k x 32,
- L2 1M x 32,
- L3 32M (shared)
- 128GB RAM
- 20 GB/s core mem
- 300 GB/s mem
- 2GB (15Gb) net

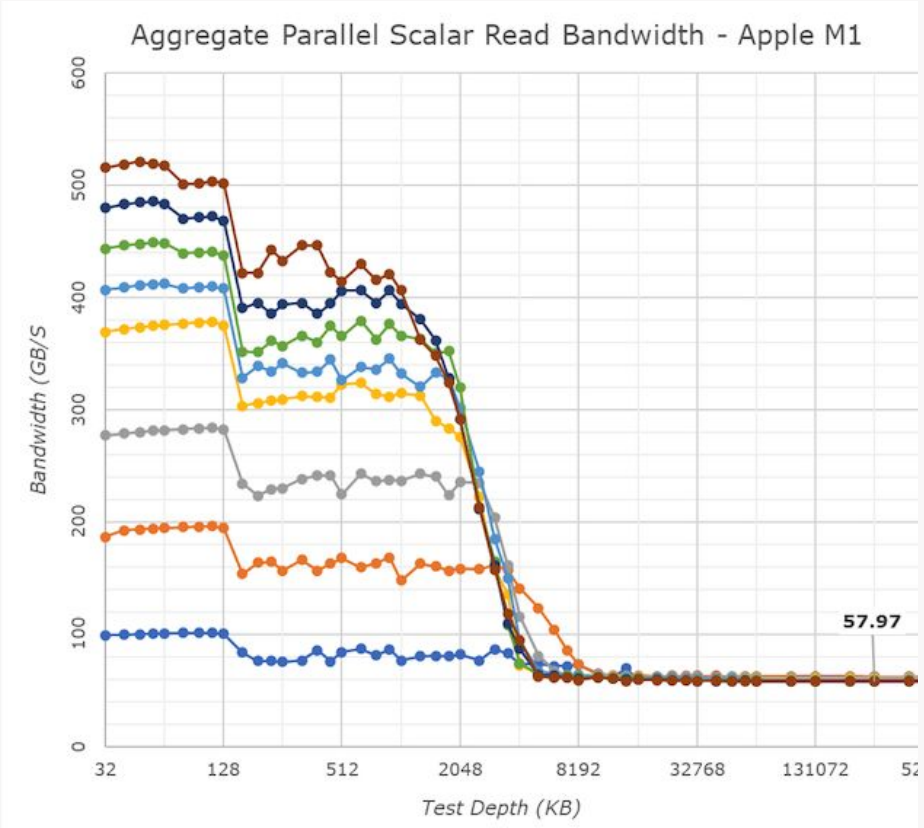
m2.4xl (2014)
$1.25/hr in 2023 $

- Intel Xeon E5-2665
- 8 cores
- L1: 32k x 8
- L2: 256k x 8
- L3: 20MB (shared)
- 68GB RAM
- ~7 GB/s core mem
- 51 MB/s agg mem
- "high" net

Apple's M1 I/O

Unified (M1) 60GB/s
M1 Pro 200 Gb/s
M1 Max 400 GB/s
M1 Ultra 800 Gb/s

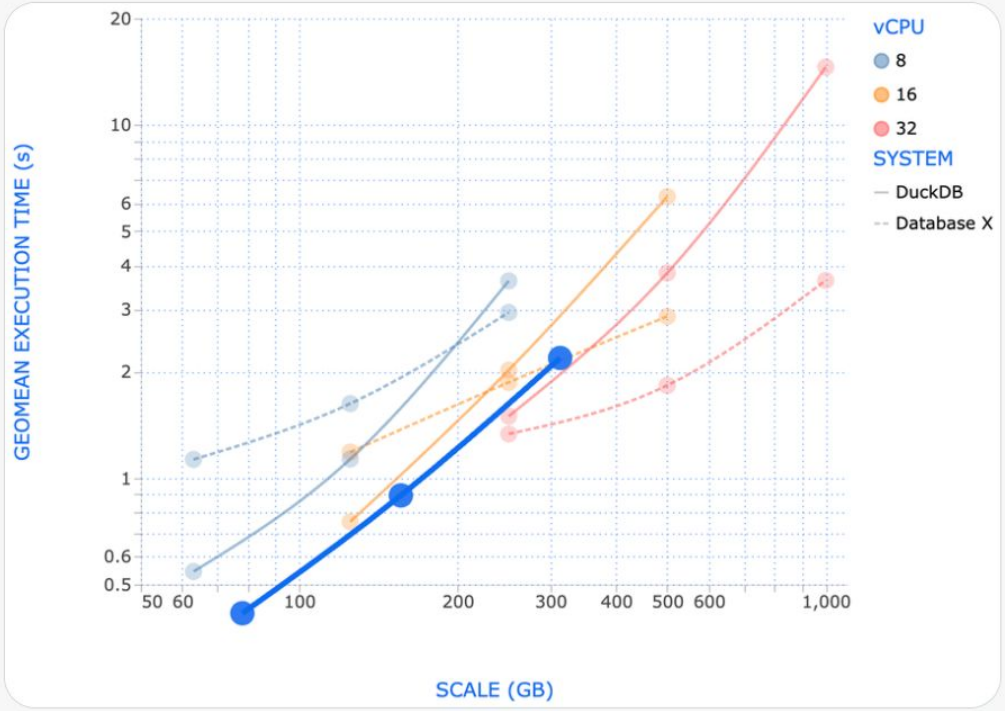
M1 SSD
2.0 GB/s SSD read
2.5 GB SSD read



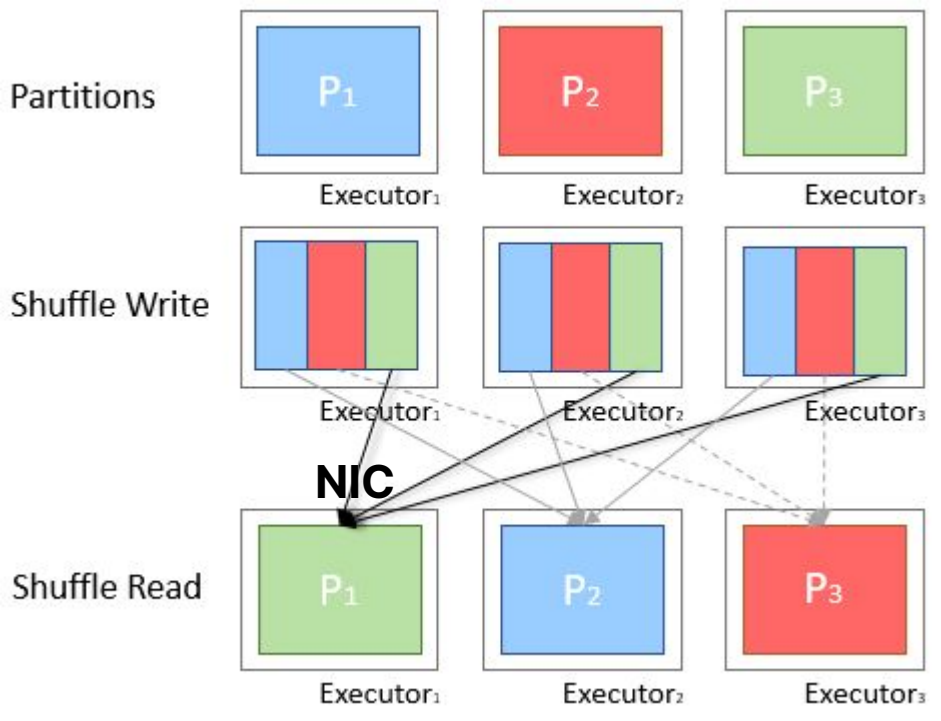
George Fraser @frasergeorgew · Mar 21

Replying to @frasergeorgew

But my most astonishing finding: the M1 Pro is an absolute beast. The laptop in front of me is faster than most of the systems I benchmarked!



Shuffles: Distributed



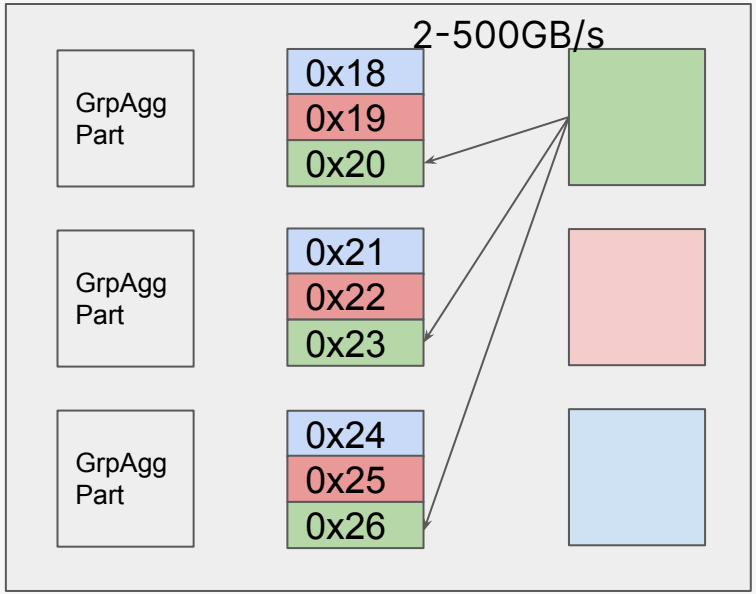
Consider what each of these lines actually *are* ?

Sender:
1. serialize the data from it's in memory representation
2. copy it into a network buffer

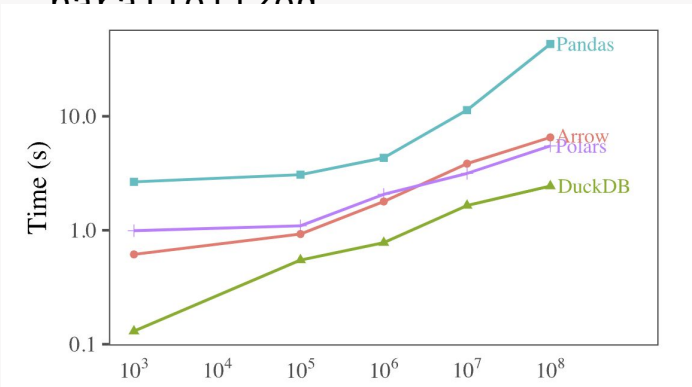
Receiver:
1. copy from network buffer when complete
2. deserialize into actual data structure

https://blog.scottlogic.com/2018/03/22/apache-spark-performance.html

... vs in memory



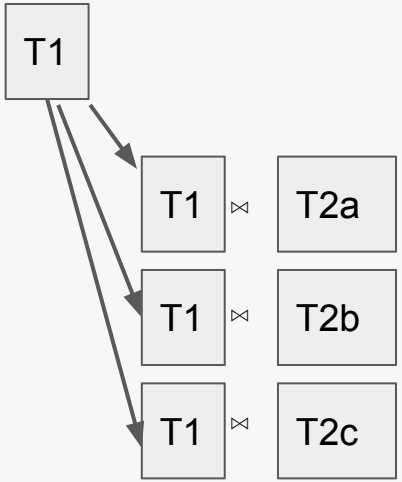
DuckDB avoids a shared hashmap. Each producer writes out pre-partitioned subsets of keys, so the next step can also be parallelized



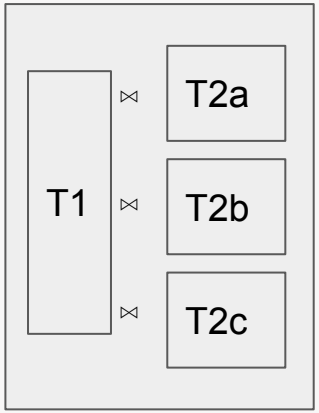
https://duckdb.org/2022/03/07/aggregate-hashtable.html

Broadcast: shared mem advantage

Same block transferred to all nodes via network



All threads just read the same block of mem.



The missing middle

When do you move to a data warehouse ?

You start with postgres. It handles your traffic as you grow. It's hosted, so you can add cores. You want to get a handle on your operations. You make some dashboard queries.

The business grows. You dashboard queries, and some exports you now do, are hurting prod. So you make a replica.

For many people this is fine. But let's say you outgrow this what is your next step ?

On GCP, BigQuery. Pricing is good.
On AWS, Redshift. Snowflake ?
It seems a big leap in cost, complexity, and fragility to go to a big warehouse.

It's like there is a step missing in the middle.

DuckDB: A playground

So:

We have a distant relationship with our Data Warehouse Processing engines. We can't run them locally. We can't even check syntax without calling them.

Meanwhile, the original motivations for using distributed systems in the first place are eroding. Our hot set is much smaller than our full data footprint. Single machines can have huge core counts, which we are now able to feed from memory. Our compilers are steadily getting better at harnessing parallelism.

Our local computers have similar increases in bandwidth, both internally. Meanwhile, the last mile connection to our local machines are beefy. SSD is getting cheaper than HDD.

And along comes a database that gives us bleeding edge DB research with a brew or pip update, that has near zero startup time.

And it can even compile to WASM.

What makes DuckDB fun ?

- Small binary matters
- Fast startup matters: 15 milliseconds
- Native code matters. JVM startup, hotspot overhead amortises as larger scale
- SQL Syntax that is postgres-like, and functions you can usually guess
- performant: query engine based on most recent innovations
- improving all the time. very short academic -> production pipeline
- embed in whatever you want

As tool of convenience

Jared Lander
@jaredlander

Just used @duckdb take 20 million rows out of Postgres into a parquet file in 18 seconds. So much faster than anything I have done before. This tool is getting better and better all the time.

9:27 AM · Feb 22, 2023 · **25.4K** View

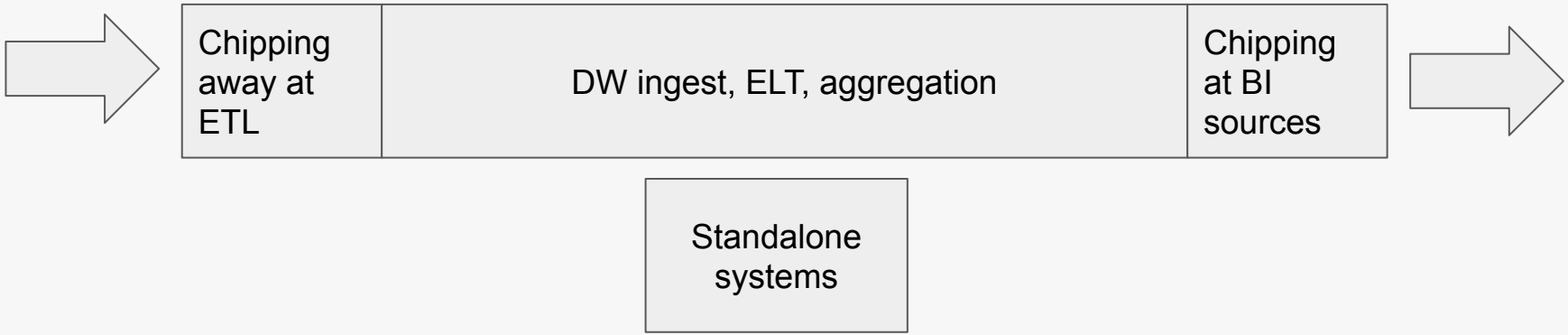
John Murray
@MurrayData

Originally in 2015, it took 3 days to run using Python & MariaDB on a 32GB box. Today it ran for the 1st time using @ApacheArrow @geopandas & @duckdb on the Kubernetes cluster node with 256GB. It took 2 hrs 43 mins 25 secs. Sorry can't share detail as commercially sensitive. 2/2

10:47 AM · Mar 7, 2023 · **313** Views

Data Infra Experiments with DuckDB

- Extract networks from twitter to supply leads
- Semantic similarity: HuggingFace model to get embeddings, and then did a cosine similarity over array type
- "ETL Agents". Summarize at edge.
- App vendor w some analytics. Some clients outgrow pg for BI. Export to parquet, DuckDB engine against S3
- Researcher wanted to use a real DB to test a lineage method. DuckDB made it available.

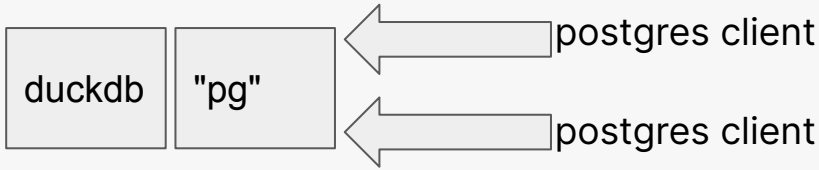


Add a postgres front-end...

Buena Vista: A Programmable Postgres Proxy Server

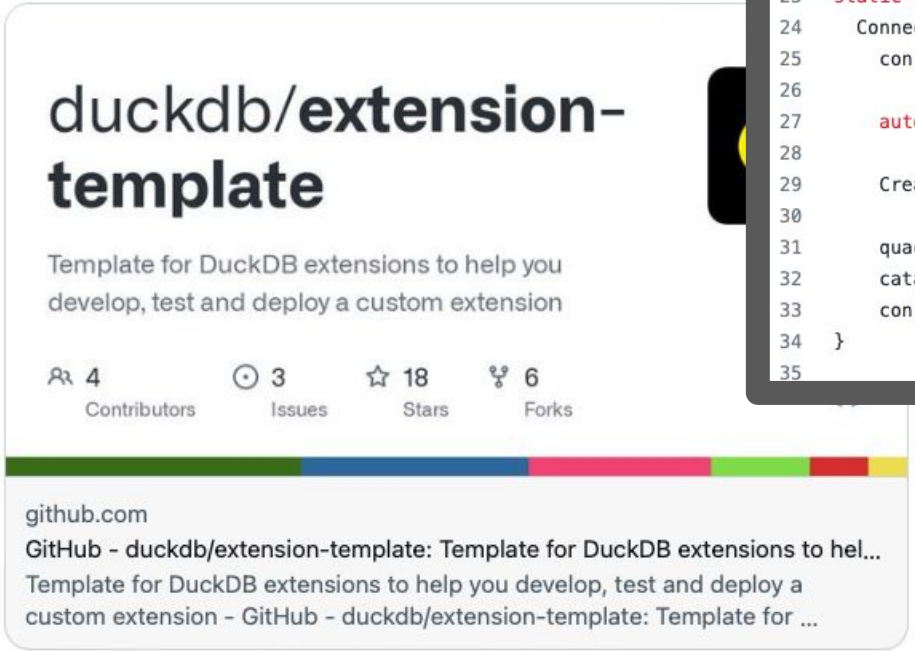
Buena Vista is a Python library that provides a socketserver-based implementation of the Postgres wire protocol (PDF).

I started working on this project in order to address a common issue that people had when they were using another one of my Python projects, dbt-duckdb: when a long-running Python process is operating on a DuckDB database, you cannot connect to the DuckDB file using the CLI or with a database query tool like DBeaver to examine the state of the database, because each DuckDB file may only be open by a single process at a time. The Buena Vista library makes it possible to work with a DuckDB database from multiple different processes over the Postgres wire protocol, and the library makes it simple enough to run an example that illustrates the idea locally:



MotherDuck
@motherduck

DuckDB Extension is a powerful way to add functionality to D
JSON support, to Full-text-search, there are already a couple
extensions.
@duckdblabs team has also worked on a template that you ca
create your own extension!

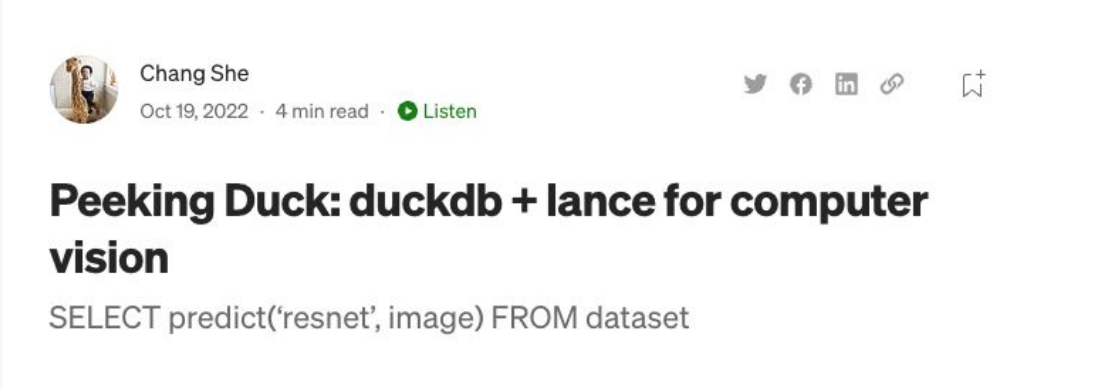



```
14  inline void QuackScalarFun(DataChunk &args, ExpressionState &state, Vector &result) {
15      auto &name_vector = args.data[0];
16      UnaryExecutor::Execute<string_t, string_t>(
17        name_vector, result, args.size(),
18        [&](string_t name) {
19        return StringVector::AddString(result, "Quack "+name.GetString()+" 🐥");;
20          });
21  }
22
23  static void LoadInternal(DatabaseInstance &instance) {
24    Connection con(instance);
25      con.BeginTransaction();
26
27      auto &catalog = Catalog::GetSystemCatalog(*con.context);
28
29      CreateScalarFunctionInfo quack_fun_info(
30              ScalarFunction("quack", {LogicalType::VARCHAR}, LogicalType::VARCHAR, QuackSc
31      quack_fun_info.on_conflict = OnCreateConflict::ALTER_ON_CONFLICT;
32      catalog.CreateFunction(*con.context, &quack_fun_info);
33      con.Commit();
34  }
35
```

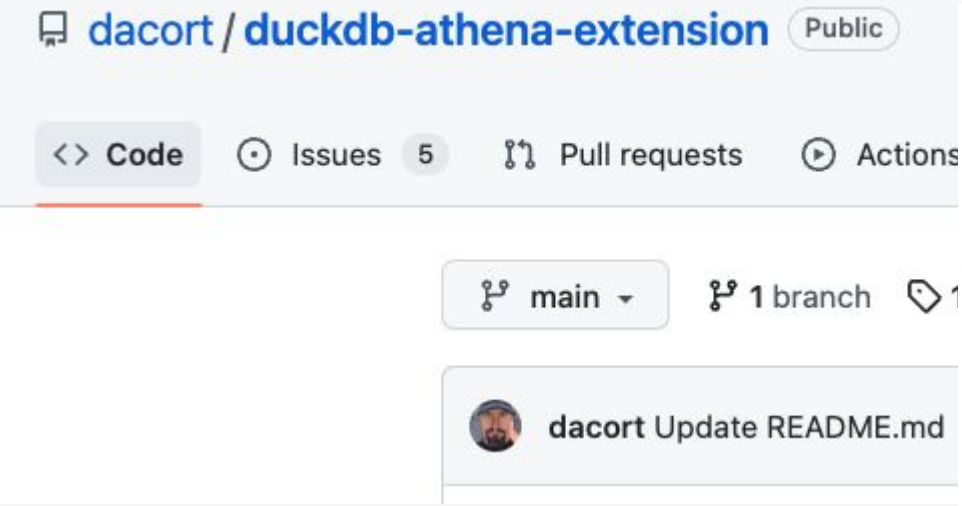
Lance: ML Data format

Lance is a columnar data format with better random access properties than parquet, making it a better choice for vector search, and inference.

"Compatible with Pandas, DuckDB, Polars, Pyarrow"
One of these is a Database, why ?
DuckDB is a good choice to showcase a db integration.



Extension to query Athena



- Load the extension

```
load 'build/debug/extension/duckdb-athena-extension/athena.duckdb_extension';
```

- Query a single table, also providing where S3 results are written to

```
select * from athena_scan('table_name', 's3://<bucket>/athena-results/);
```

⚠ Warning: 10,000 results will be returned by default! Use `maxrows=-1` to return the enti

```
D select * from athena_scan("amazon_reviews_parquet");
Running Athena query, execution id: 152a20c7-ff32-4a19-bb71-ae0135373ca6
State: Queued, sleep 5 secs ...
Total execution time: 1307 millis
100% ████████████████████████████████████████
```

marketplace varchar	customer_id varchar	review_id varchar	product_id varchar	product_parent varchar	…
US	37441986	R2H287L0BUP89U	B00CT780C2	473048287	…
US	20676035	R1222MJHP5QWXE	B004LLILFA	361255549	…
US	45090731	R32ECJRNTB61K8	B004LLIL4G	307223063	…
US	2207141	RLTEU3JZ1IJAA	B004LLILDM	87389551	…
US	15258	R1ZAX1TN66QOU6	B004LLIKVU	473048287	…
·	·	·	·	·	·

999 rows (40 shown)

Scan postgres faster than psql

README.md

DuckDB postgresscanner extension

The postgresscanner extension allows DuckDB to directly read data from a running Postgres instance. The data can be queried directly from the underlying Postgres tables, or read into DuckDB tables.

Usage

To make a Postgres database accessible to DuckDB, use the `POSTGRES_ATTACH` command:

```
CALL POSTGRES_ATTACH('');
```

`POSTGRES_ATTACH` takes a single required string parameter, which is the `libpq` connection string. For example you can pass `'dbname=postgresscanner'` to select a different database name. In the simplest case, the parameter is just `''`. There are three additional named parameters:

- `source_schema` the name of a non-standard schema name in Postgres to get tables from. Default is `public`.
- `sink_schema` the schema name in DuckDB to create views. Default is `main`.
- `overwrite` whether we should overwrite existing views in the target schema, default is `false`

WHERE TRUE
Technologies

Sign In / Create Account

WTT-01

WTT-01 can help you and your team reduce the time from bioinformatic data output to meaningful insight.

See Demo & Purchase Options

We think WTT-01 can help the full spectrum of bioinformatic practitioners from Bench Scientist to Engineer.

Bench Scientists can eschew complex python and R scripts and use the lingua franca of data analysis, SQL.

For example, lets say you had used a FASTA of contigs, a GFF with gene predictions, and a CSV with PFAM annotations. With WTT-01, you can quickly create a FASTA with contigs that have a gene with a certain PFAM domain.

```
SELECT id, sequence
FROM read_fasta('contigs.fasta') fa
  JOIN read_gff('genes.gff') gff
    ON fa.id = gff.reference_sequence_name
  JOIN read_csv('pfam.csv') pfam
    ON gff.attributes['gene_id'][1] = pfam.gene_id
```

Bioinformaticians & Data Scientists can move reduce the complexity of their pipelines. Go from cobbling together a pipeline in bash to a SQL queries.

For example, a Bioinformatician can assess the read quality from an NGS Run using analytical SQL queries to recapitulate and extend common read quality tools. Here, we compute the average quality score by base position directly from the FASTQ

```
WITH qc_by_base AS (
  SELECT name,
    qc_score,
    ROW_NUMBER() OVER (PARTITION BY name) AS position
  FROM (
    SELECT name,
      UNNEST(
        quality_score_string_to_list(quality_scores)
```

https://www.wheretrue.com/wtt-01

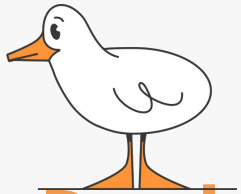
```
brew install duckdb
apt/yum install duckdb

pip install duckdb

duckdb.org -> Installations
```

MotherDuck

What is MotherDuck?



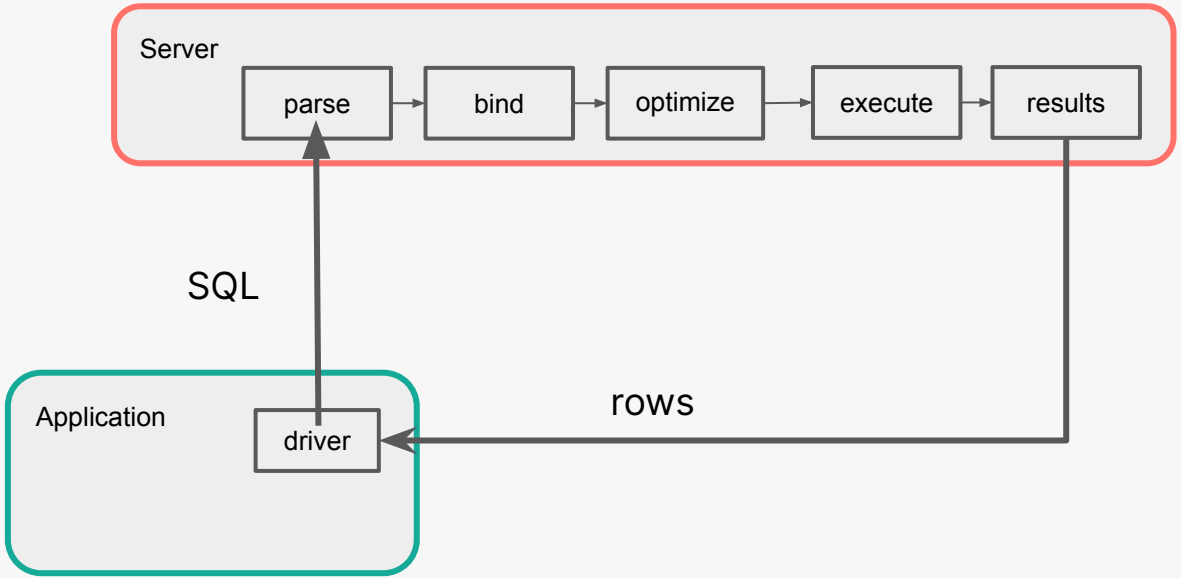
Serverless DuckDB

Client is DuckDB in all its forms:
python, cli, WASM, JS etc.

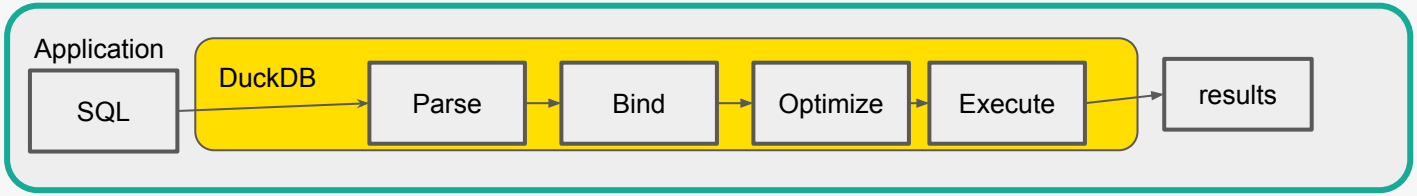
Typical relationship with DB

I send you SQL. I wait. You send me back rows.

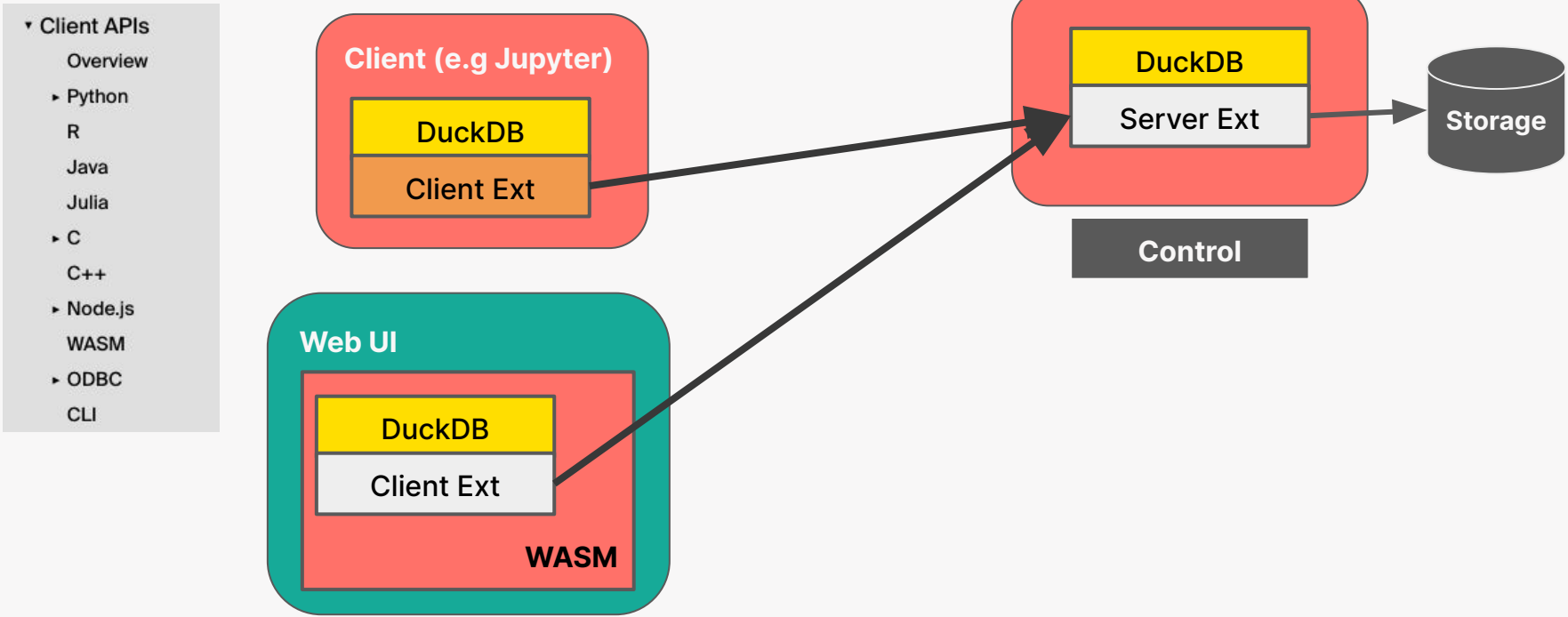
Mainframe relationship.



DuckDB In-Process

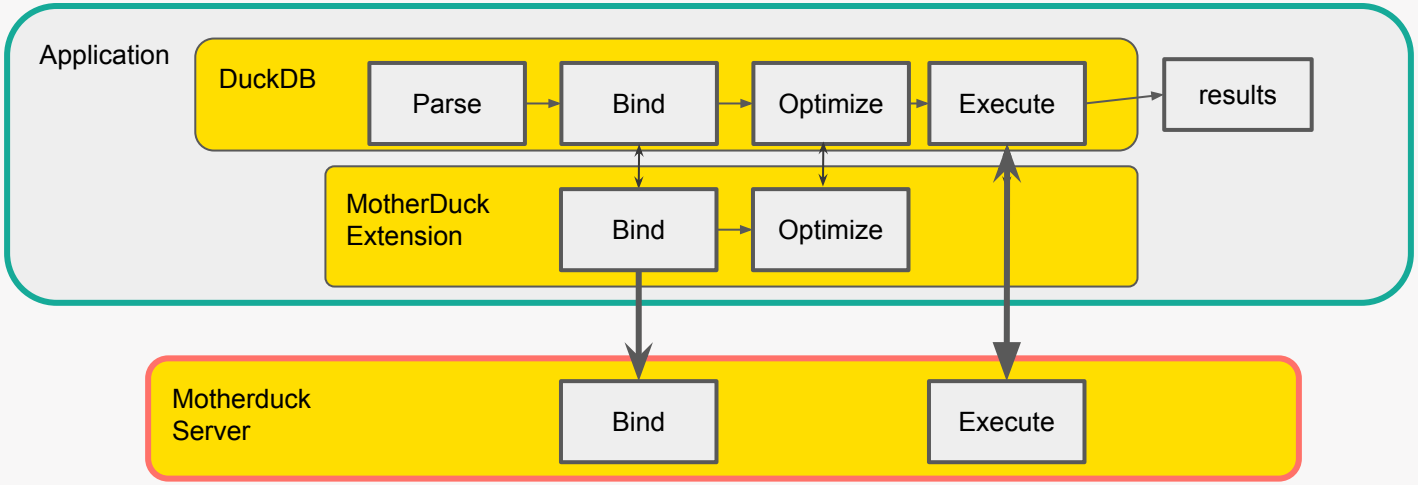


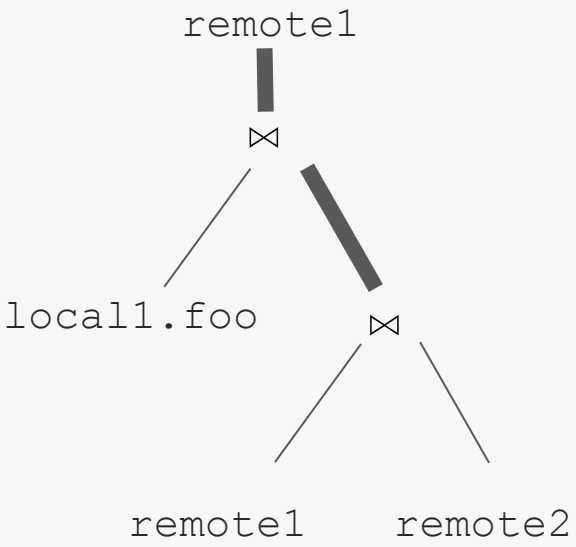
MotherDuck Architecture: DuckDB+Extensions

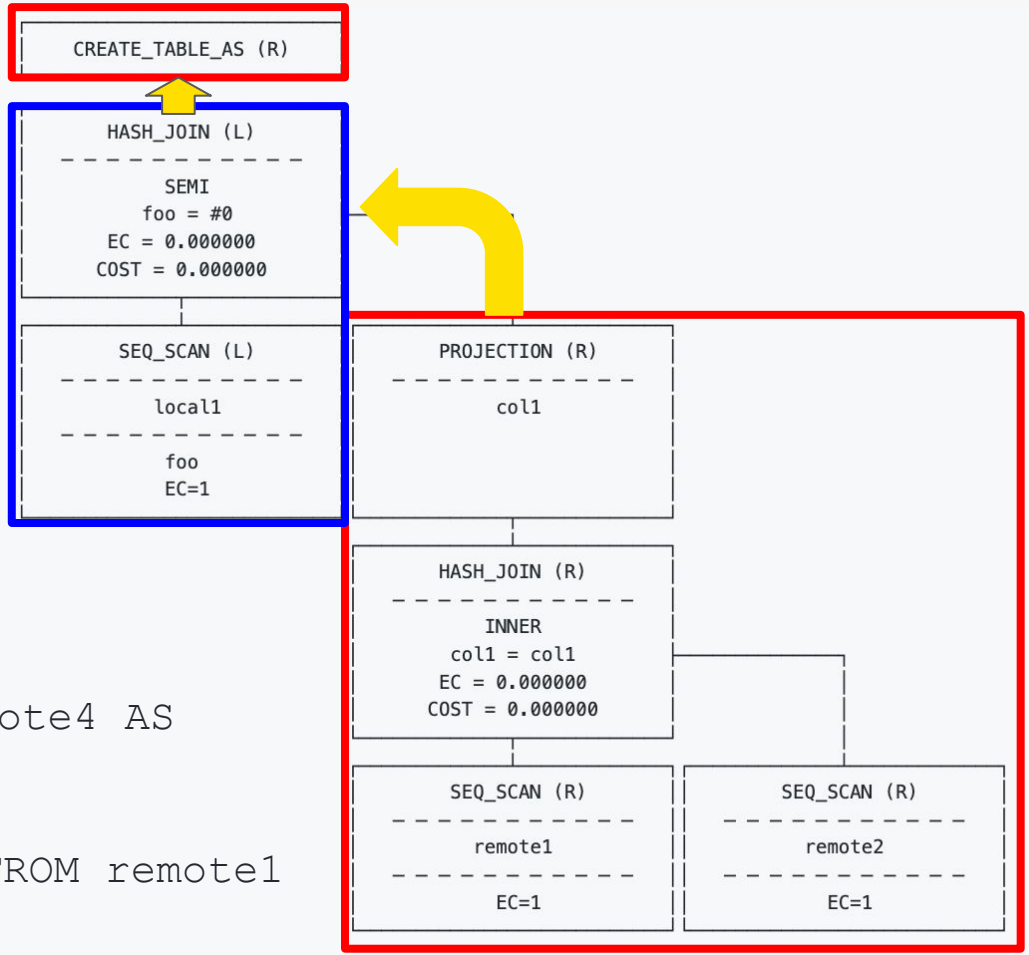


```
duckdb-0.7.1
D .open "md:tino"
-- OAuth Loop for credentials
D CREATE DATABASE db1
D SELECT name FROM md_databases()
  db1
D CREATE TABLE t1 as select 'abc' as x
D ATTACH local.db as L
D SELECT * from db1.t1
    JOIN L.t1 on (id)
```

MotherDuck Execution

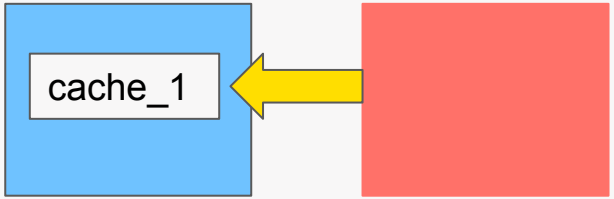







```
CREATE OR REPLACE TABLE remote4 AS
  SELECT * FROM local1
    where local1.foo IN
      (SELECT remote1.col1 FROM remote1
       JOIN remote2 ON
            remote1.col1=remote2.col1);
```

Trivial local result reuse




```
create table local.cache_1
as (select .... from remote)

select from cache_1 where...
-- 1ms

select a, sum(b) from
cache_1 group by a
-- 1ms
```

Thank you.

tino@motherduck.com
nick@motherduck.com