Rage DB

Building a Graph Database in Anger

Sixteenth TUC Meeting - June 24, 2023

RAGE DB Home Posts HTTP API Timeline

Tweet Slack Source Code Docker Download

Lose seconds off your graph queries with this one weird trick[1]

Doctors[2] hate him!
Developer builds a property graph server from home.

- Faster than a speeding bullet train
- Connect from anywhere via HTTP
- Use a programing language to query
- Apache license, version 2.0

Try it right now using Docker!

RAGE DB

(1) Put everything in memory (2) Computer Science PhDs.

main ▾ ragedb / LICENSE.txt

ragedb/ragedb is licensed under the
Apache License 2.0

A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- ✓ Commercial use
- ✓ Modification
- ✓ Distribution
- ✓ Patent use
- ✓ Private use

Limitations

- ✕ Trademark use
- ✕ Liability
- ✕ Warranty

This is not legal advice. Learn more about repository licenses.

maxdemarzi first commit Latest commit

1 contributor

177 lines (150 sloc) | 9.93 KB

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/
```

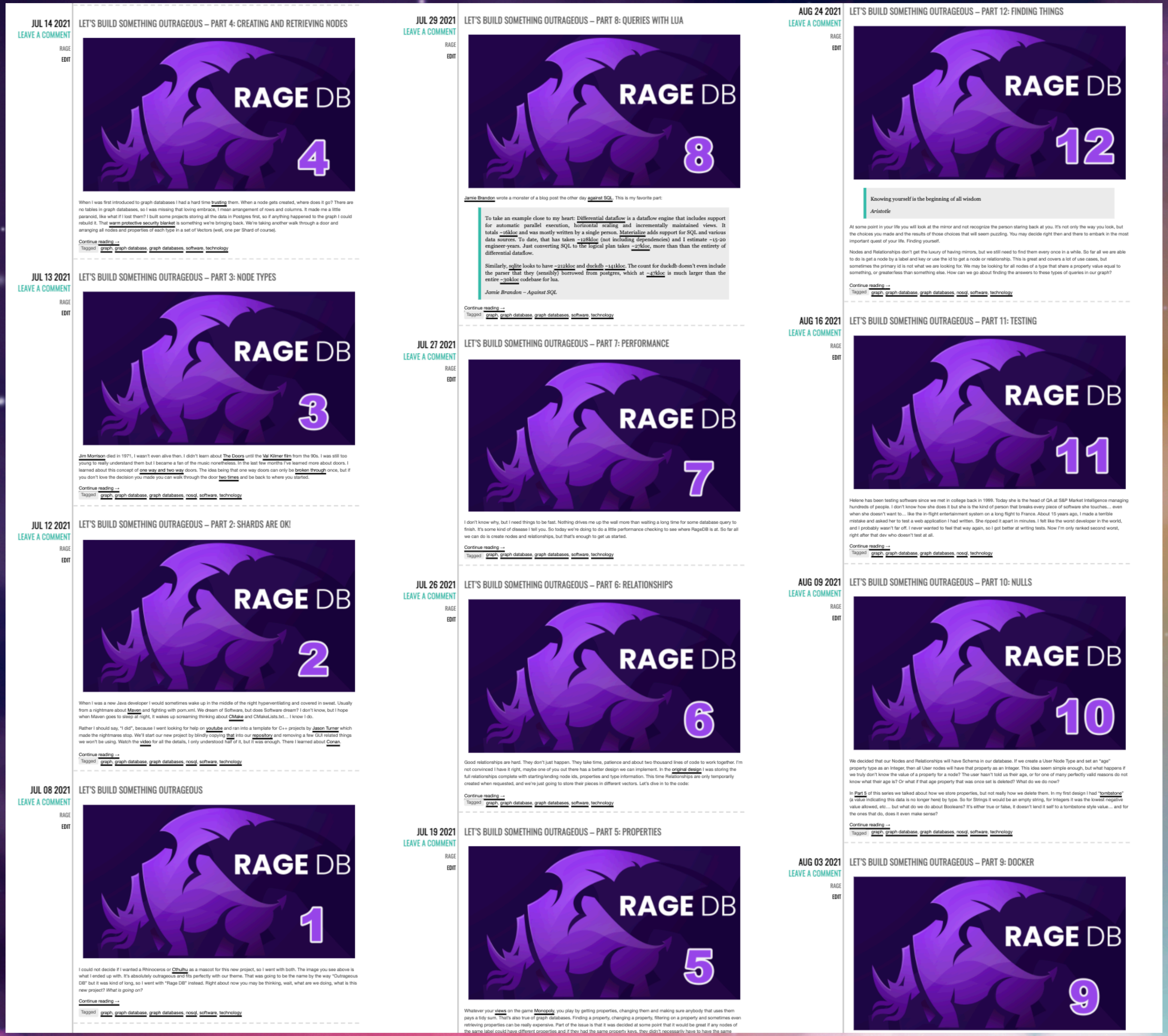
Blog Posts

maxdemarzi.com


APR 12 2022
LEAVE A COMMENT
RAGE
EDIT

LET'S BUILD SOMETHING OUTRAGEOUS – PART 25: DATES IN C++ AND FASTER IMPORTS

Back in February, we added the ability to load a CSV file and alter the contents while importing it. We also added Date support to RageDB using a Lua library. This was a masterful job of copy and paste and got us lots of functionality very quickly. When we timed the import for LDBC SNB SF10 it came in at 28 minutes. Which wasn't bad, but wasn't great. Let's try to speed that up today.



Reggie the RageDB Cthulhuceros

Please allow me to

INTRODUCE

MYSELF

I'M Here

because Neo4j

never went Public

Max De Marzi

@maxdemarzi
maxdemarzi.com
GitHub.com/maxdemarzi

JAN 04 2012
30 COMMENTS
NEOGRAPHY
EDIT

GETTING STARTED WITH RUBY AND NEO4J

Getting started with Ruby and Neo4j is very easy.

Follow these steps and you'll be up and running in no time.

First we install the neography gem:

Using Bundler:

```
echo "source 'http://rubygems.org'
gem 'neography' " > Gemfile
bundle install
```

Without Bundler:

```
gem install neography
```

Then we'll add our tasks to a Rakefile, download Neo4j and start it:

```
echo "require 'neography/tasks'" > Rakefile
rake neo4j:install
rake neo4j:start
```

Eleven Years

In the Graph game


NOT an Academic

Work the Field


THING
1
THING
2


Scale


1 Horse Sized Duck
VS
100 Duck Sized Horses

intel. XEON

Intel® Xeon® Processor codenamed Sierra Forest

First Xeon processor with Efficient-core (E-Core)

Sampling today, shipping 1st half of 2024

Excellent silicon health

Silicon power-on;

Operating systems booted in <18 hours

Lead vehicle for Intel 3

144 processor cores

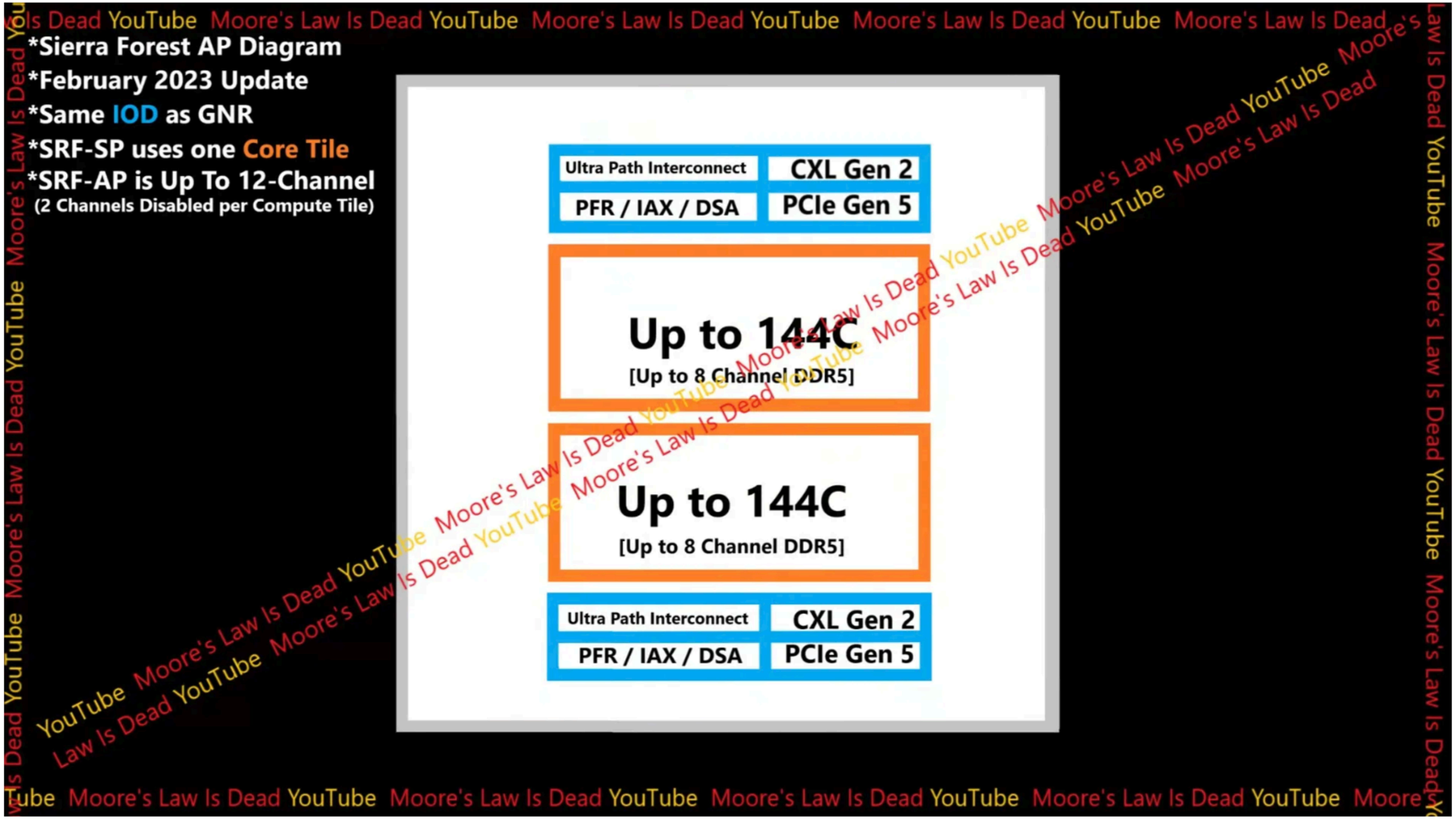
144

New class of Xeon

Built for cloud-optimized workloads

288 Cores

Below is the diagram for Sierra Forest-AP, featuring **2** tiles for a total of **288** cores. The customer in question wants the same CPU configuration, with just 1 tile disabled (144-cores). And that 144-core CPU is probably going to be marketed by Intel as Sierra Forest-SP.



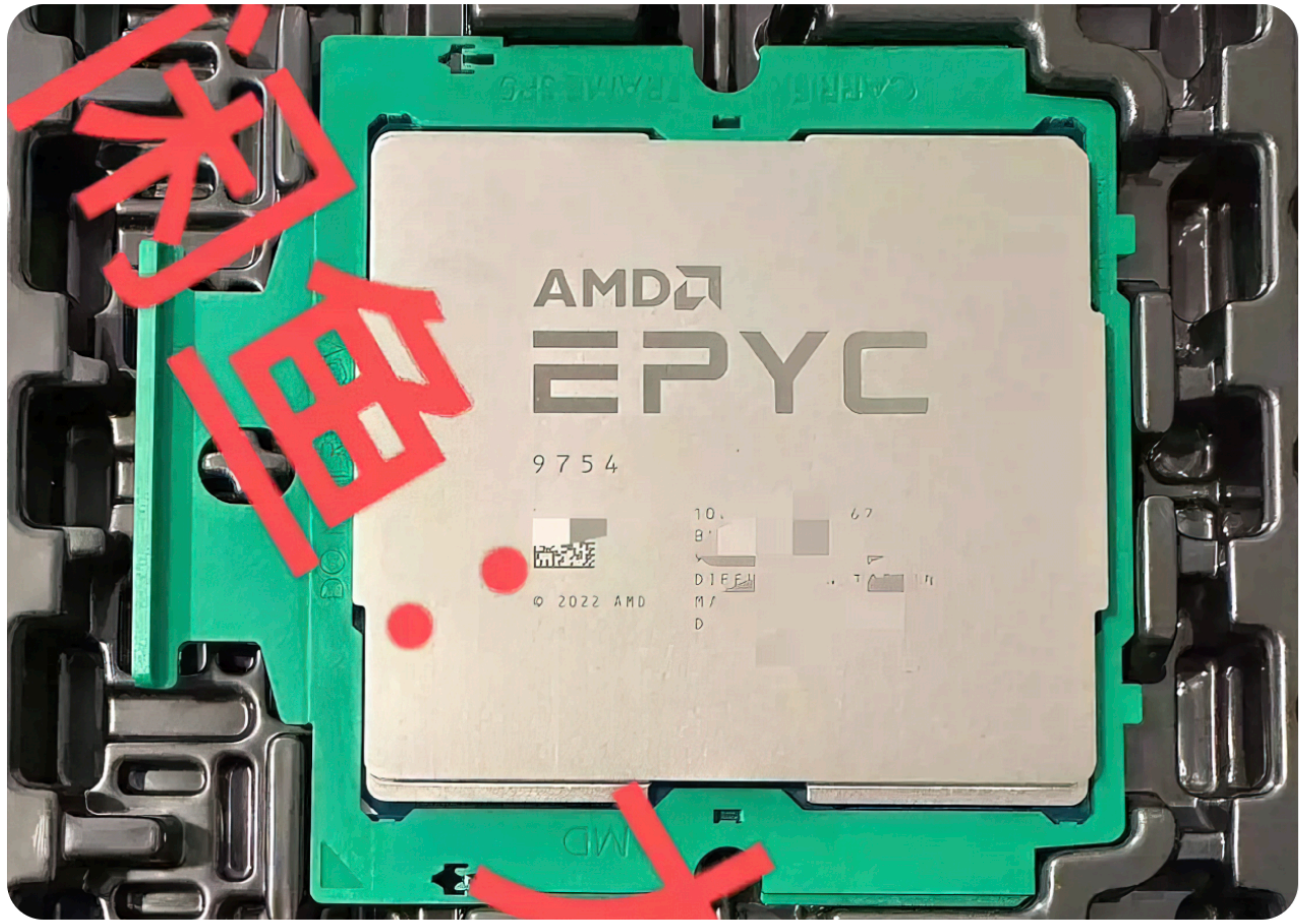
Sierra Forest-AP Configuration | MLID

128 Cores For $5500

Hardware

AMD EPYC 9754 "Bergamo" CPU With 128 Zen 4C Cores Being Sold For $5500 US, Less Than Half Its Official Price

Hassan Mujtaba • Jun 20, 2023 11:11 AM EDT • Copy Shortlink   96



AMD recently introduced its brand new Bergamo CPU lineup which features the flagship EPYC 9754 chip with 128 Zen 4C Cores. This chip is now being sold at the Chinese 3rd party seller, Goofish.

AMD's Flagship Bergamo Chip, The EPYC 9754 128-Core CPU, Is Being Sold For $5500 US

Moar core blimey: 384-core AMD EPYC Venice server chip with Zen 6 architecture enters the rumor mill

384 !!!

A well-known leaker has made some extraordinary claims about the future of AMD's server chips, EPYC. Apparently, there is a generation coming within a few years, codenamed "Venice", that will offer well over 200 cores per part and may even include an astonishing 384-core SKU. The Zen 6-based chips will also apparently feature custom memory upgrades.

AMD's EPYC lineup for 2024/25 will purportedly be codenamed "Venice" and will use Zen 6 microarchitecture. (Image source: AMD/Unsplash - edited)

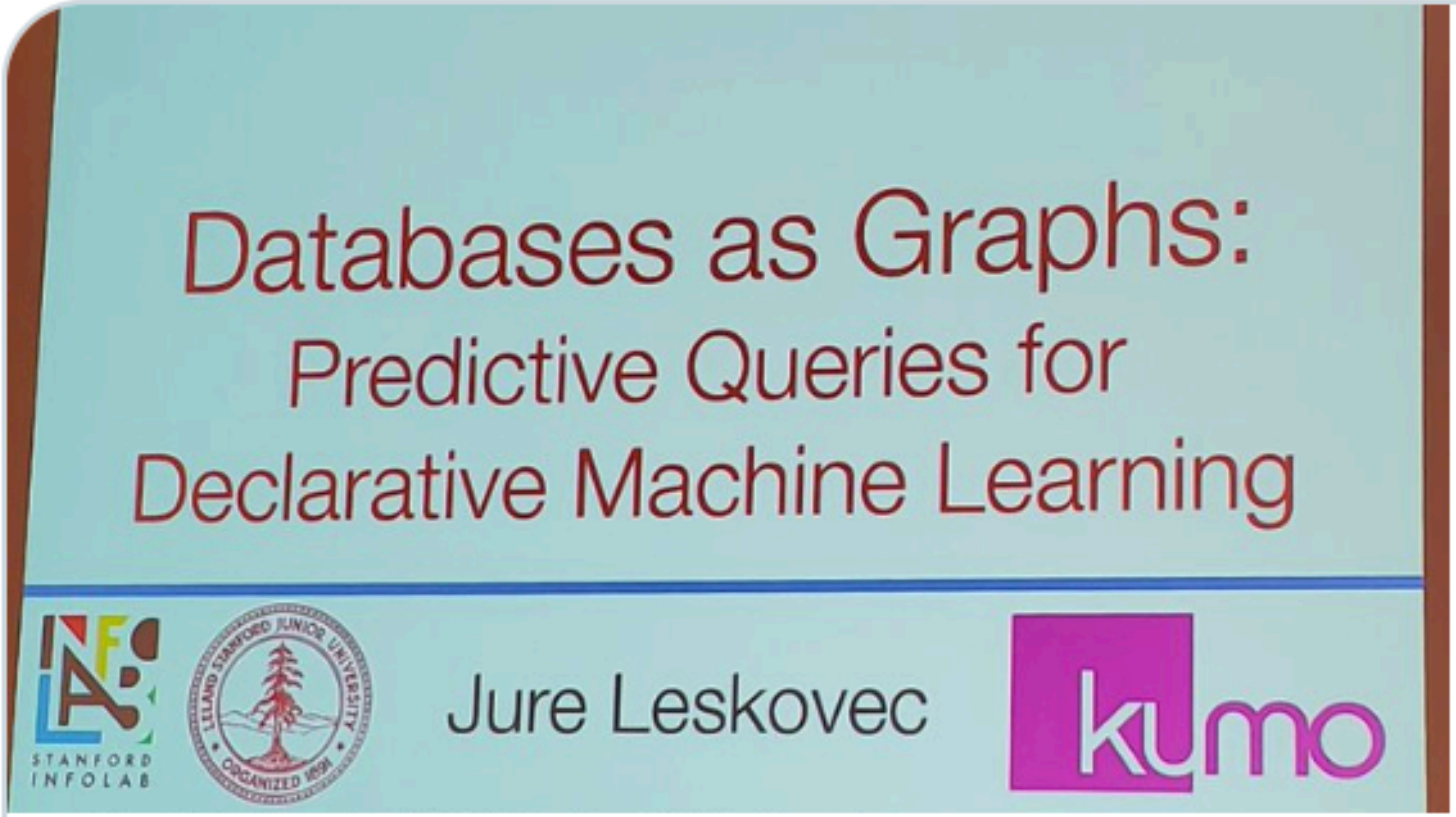
Daniel R Deakin, *05/01/2022* 🇫🇷 🇪🇸 ... Zen AMD Leaks / Rumors Workstation Business

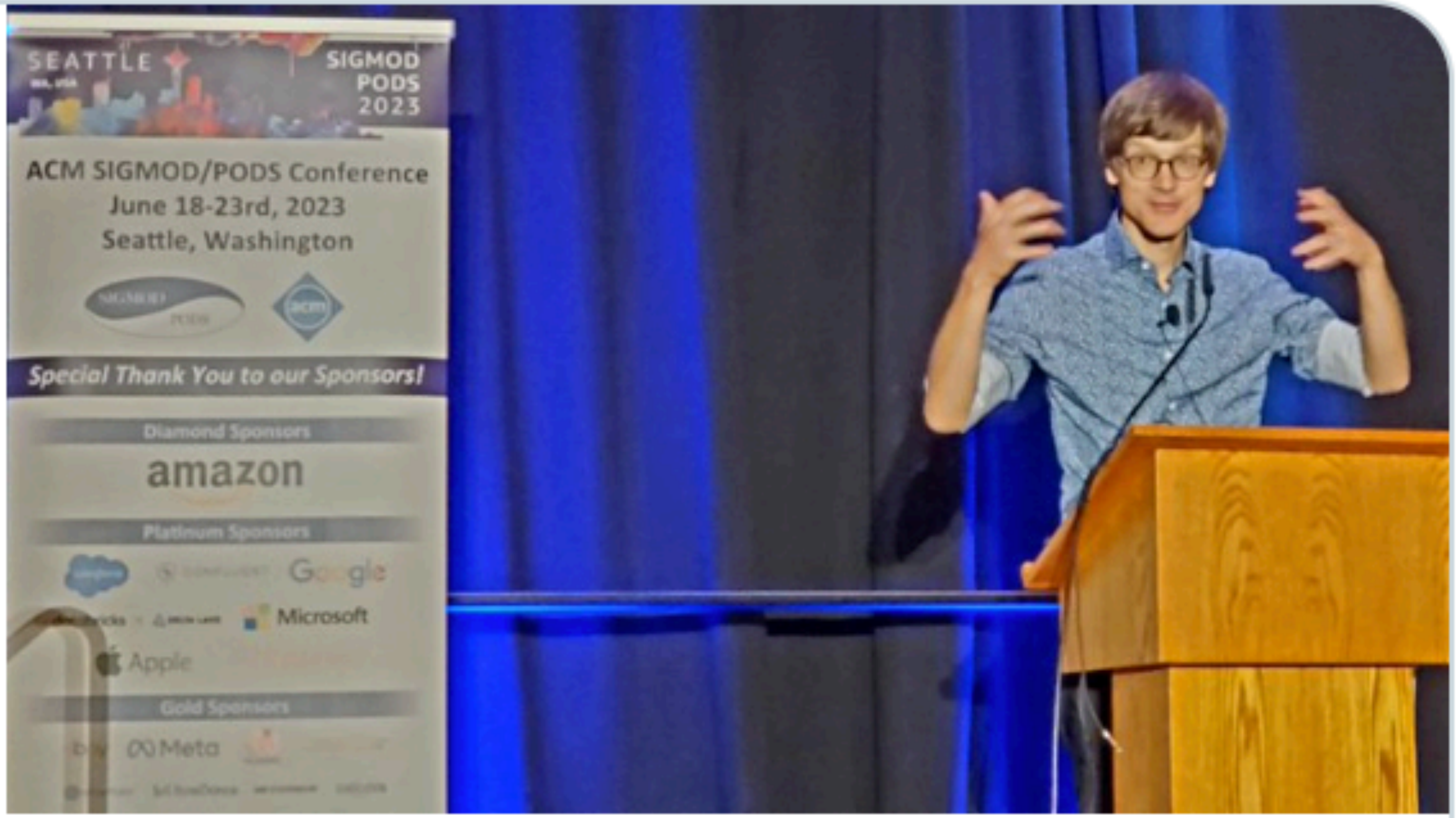
The YouTube channel Moore's Law Is Dead has been a frequent source of leaks and rumors in the past but this latest one from show host Tom will definitely require a pinch of salt, especially as it discusses naming schemes and configurations that are potentially

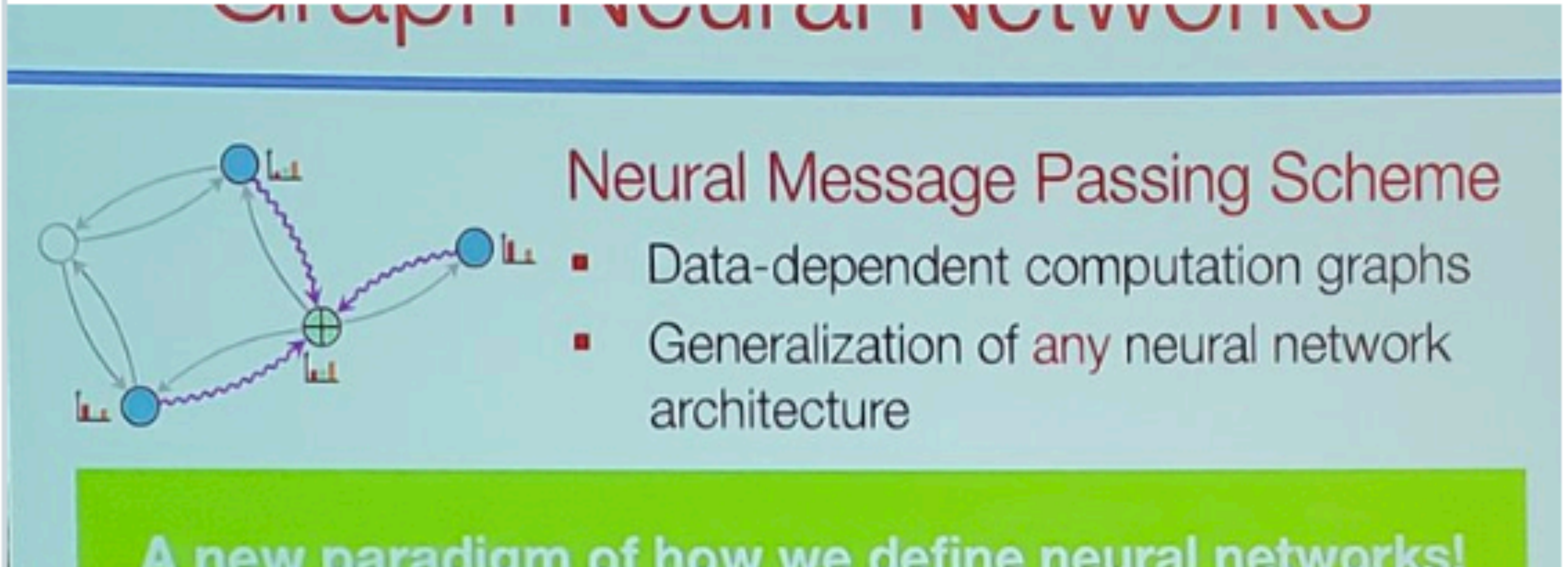

RAM
6TB RAM

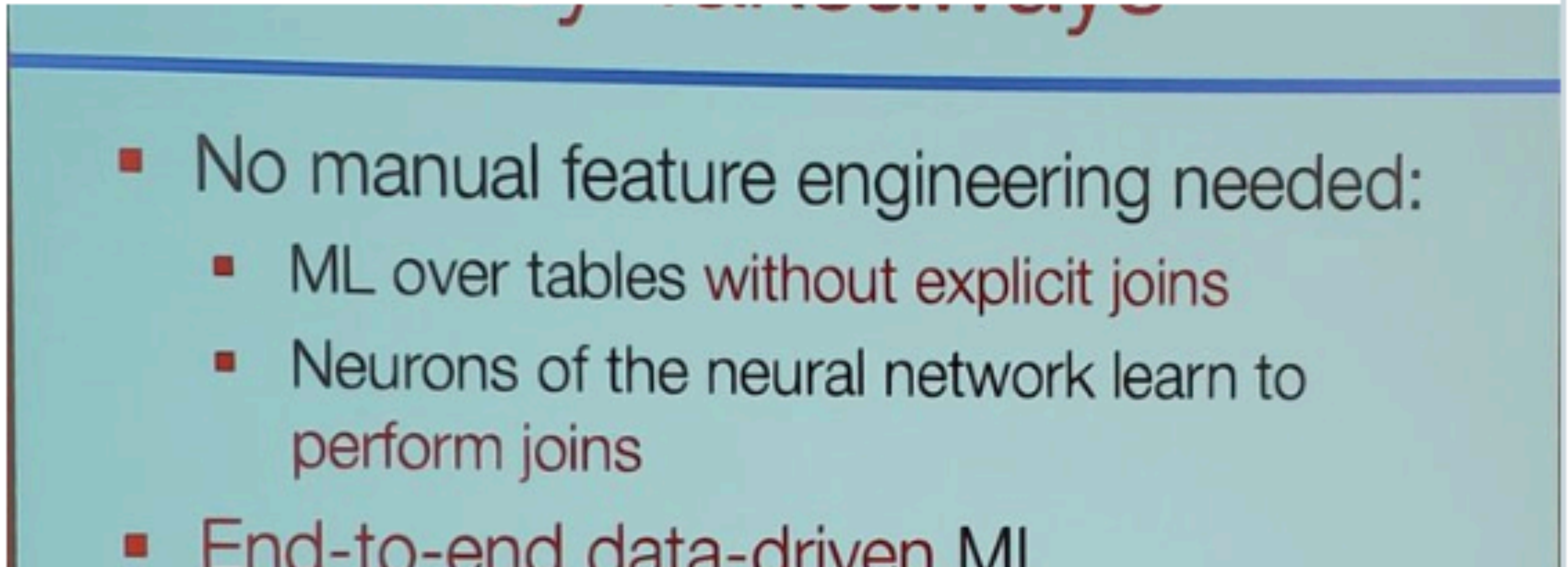

STOP
Distributed Systems

At @SIGMODConf #SIGMOD2023 #PODS2023 starts with excellent keynote by @jure of @StanfordEng

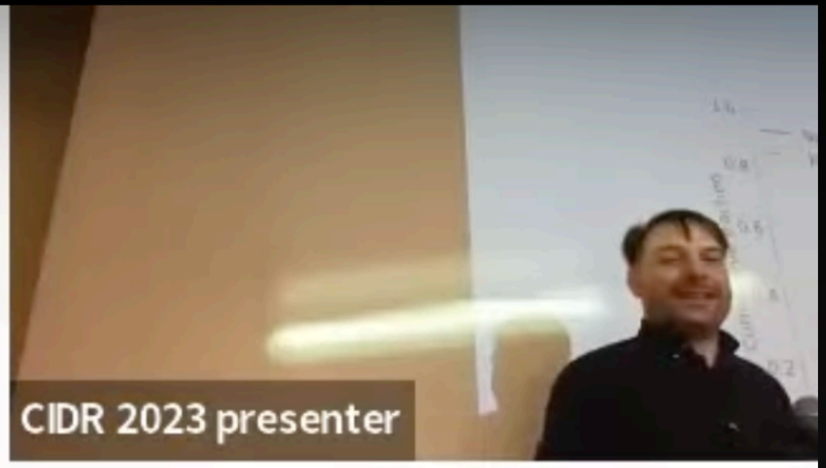




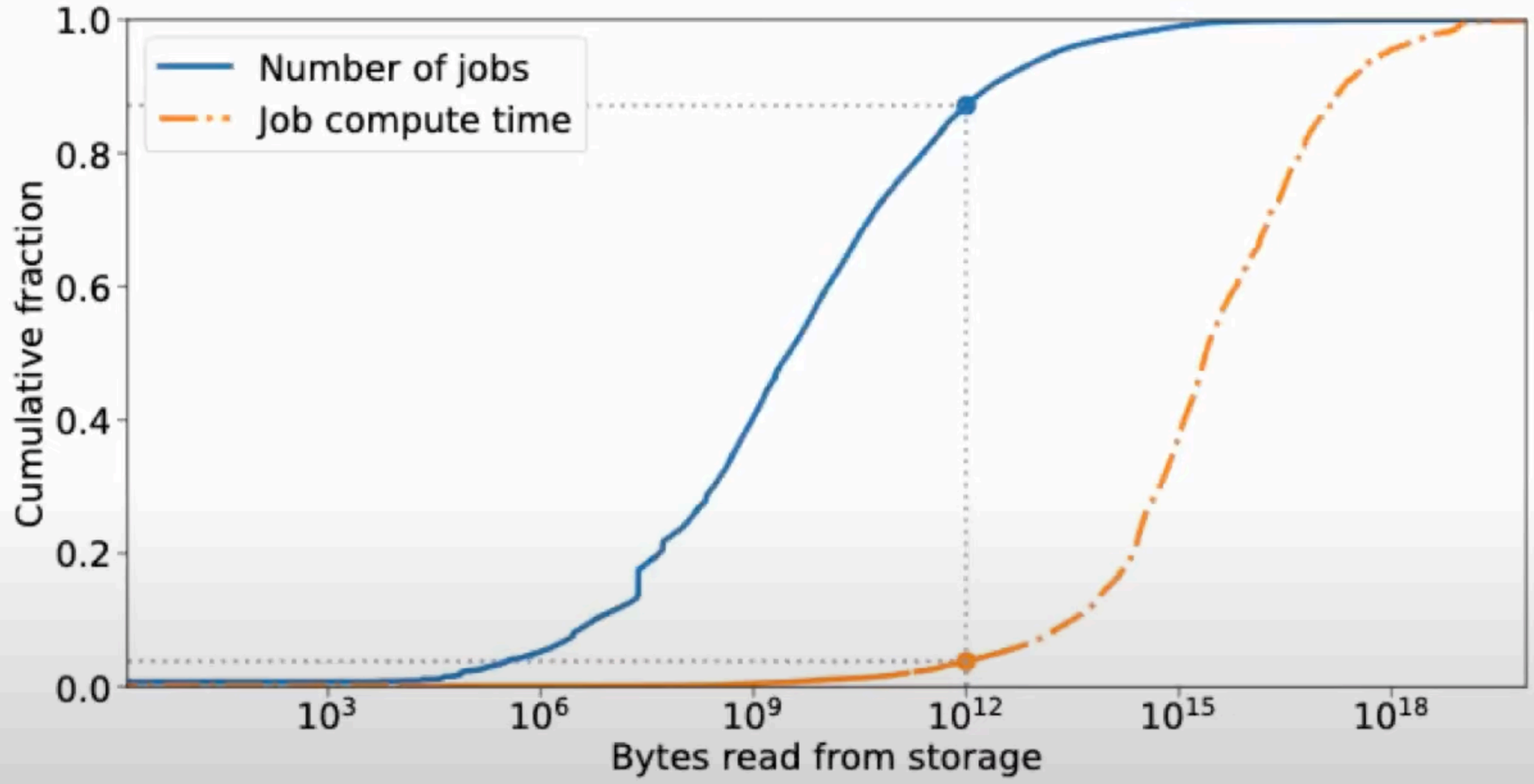




CIDR 2023 keynote – Developing Systems in Academia: The Good, the Bad, and the not-so-Ugly Duckling



At Google, 90% of all analytics workloads operate on less than 1 TB of data.

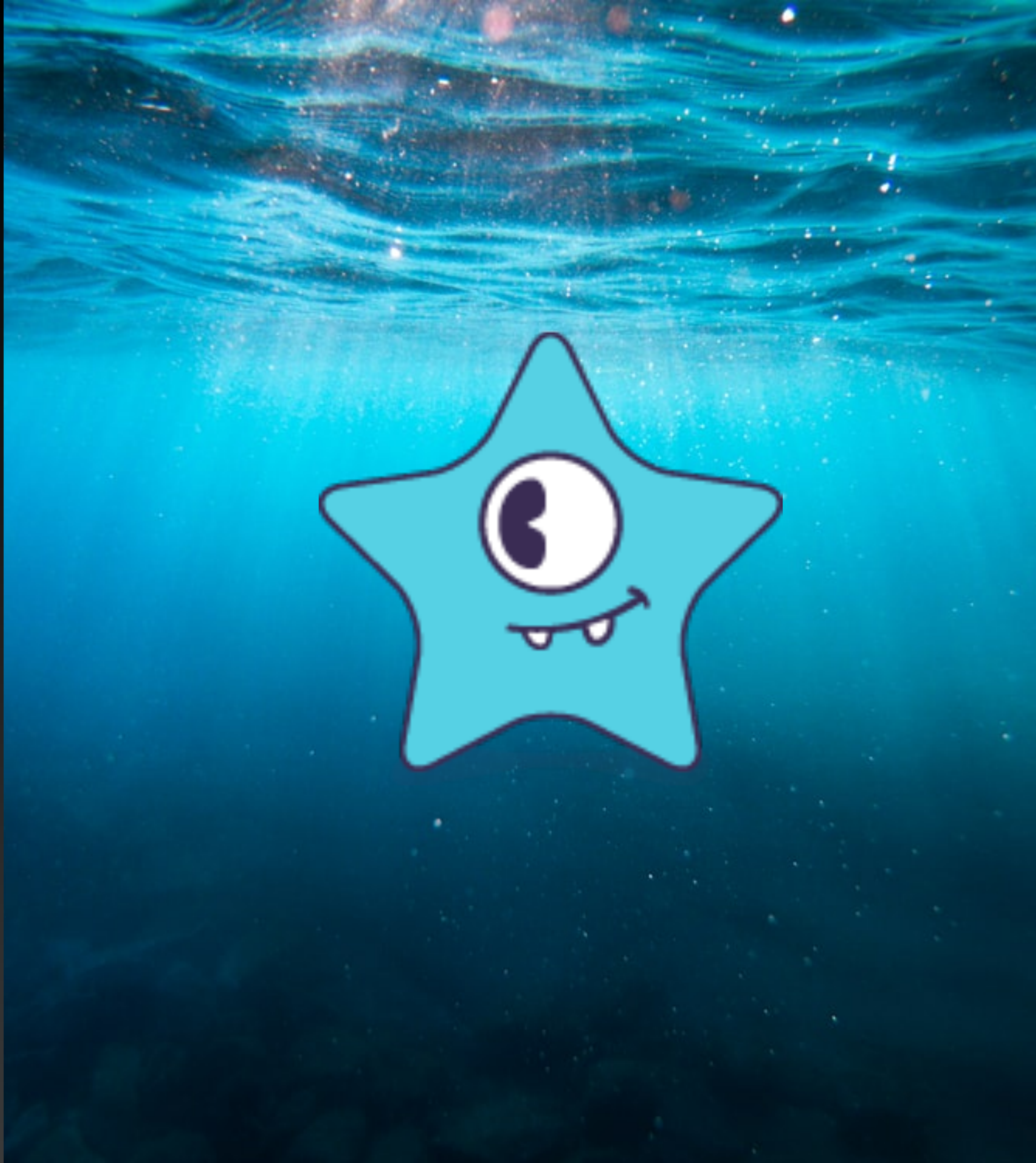


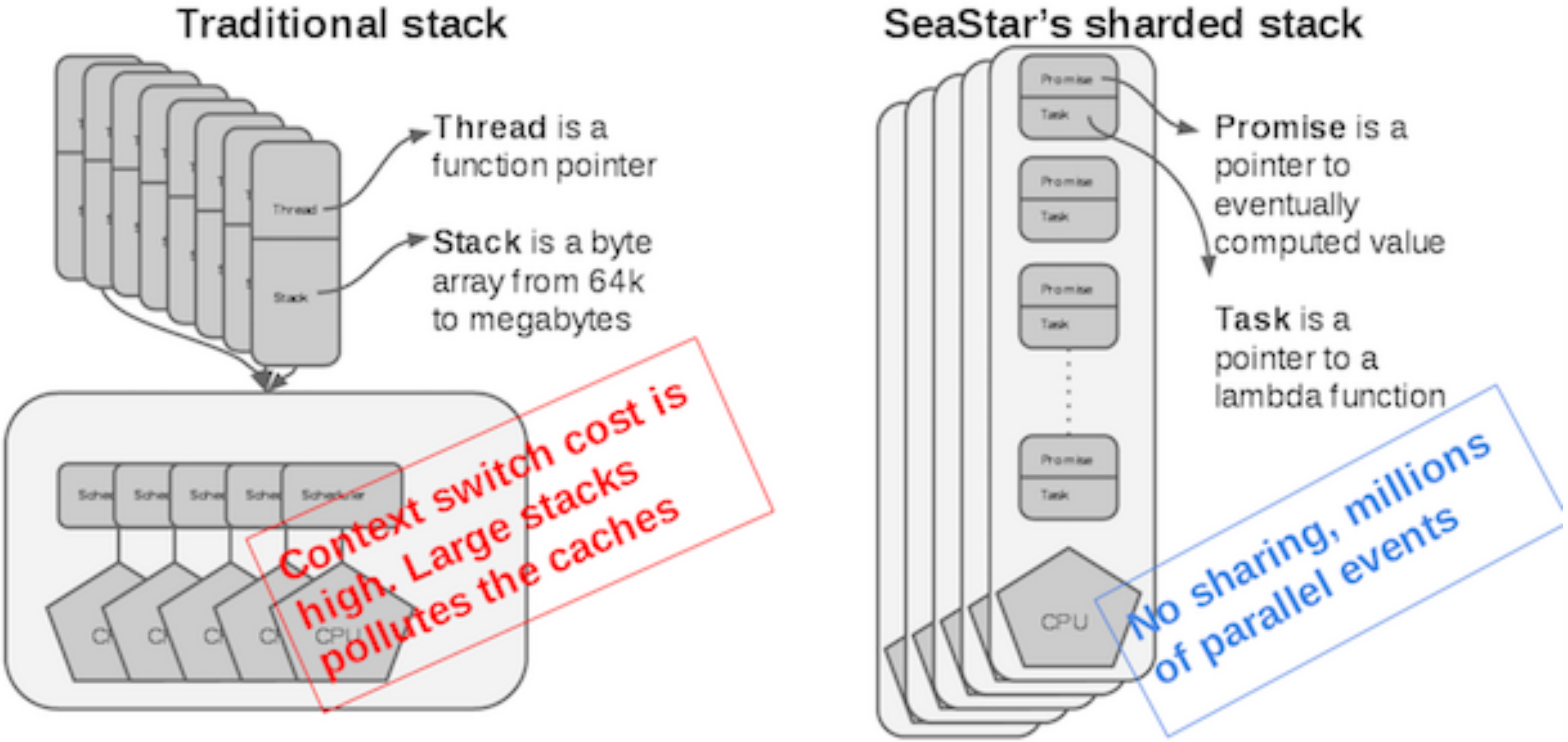
Source: tf.data: A Machine Learning Data Processing Framework, VLDB 2021

Distribute On Cores not on Servers

Seastar
Framework

- Shared Nothing
- Message Passing
- Futures and Promises
- High Performance Networking




Traditional stack
Thread is a function pointer
Stack is a byte array from 64k to megabytes
Thread
Stack
Scheduler
CPU
Context switch cost is high. Large stacks pollutes the caches
SeaStar's sharded stack
Promise
Task
CPU
Promise is a pointer to eventually computed value
Task is a pointer to a lambda function
No sharing, millions of parallel events

Queries

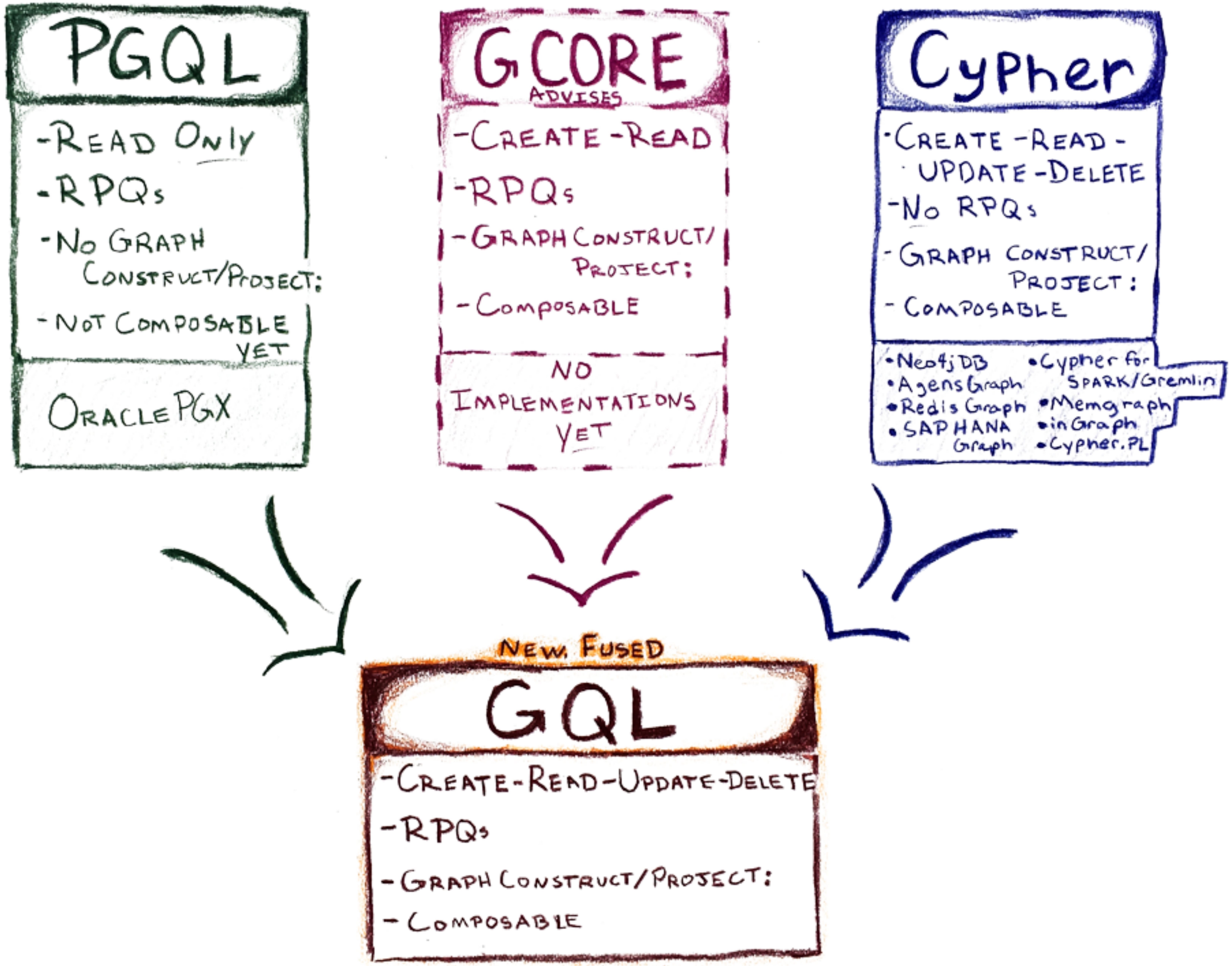
A Dozen to a Hundred

1- 3 Money Queries

Monster Queries

- Complex Business Logic
- Bidirectional Traversals
- Involve relationship properties
- Weighted Shortest Paths
- Fan out Exploration Queries
- Queries through Dense Nodes


Queries that make
your Query Optimizer Cry



PGQL
-READ Only
-RPQs
-No GRAPH CONSTRUCT/PROJECT:
-NOT COMPOSABLE YET
ORACLE PGX
GCORE
ADVISES
-CREATE -READ
-RPQs
-GRAPH CONSTRUCT/PROJECT:
-COMPOSABLE
NO IMPLEMENTATIONS YET
Cypher
-CREATE -READ - UPDATE -DELETE
-No RPQs
-GRAPH CONSTRUCT/PROJECT:
-COMPOSABLE
•Neo4j DB
•AgensGraph
•Redis Graph
•SAP HANA Graph
•Cypher for SPARK/Gremlin
•Memgraph
•inGraph
•Cypher.PL
NEW, FUSED
GQL
-CREATE-READ-UPDATE-DELETE
-RPQs
-GRAPH CONSTRUCT/PROJECT:
-COMPOSABLE

MAY 25 2020

1 COMMENT

DATABASE, RANDOM

EDIT

DECLARATIVE QUERY LANGUAGES ARE THE IRAQ WAR OF COMPUTER SCIENCE

44 Different
Queries

- There are at least 44 different ways to write:

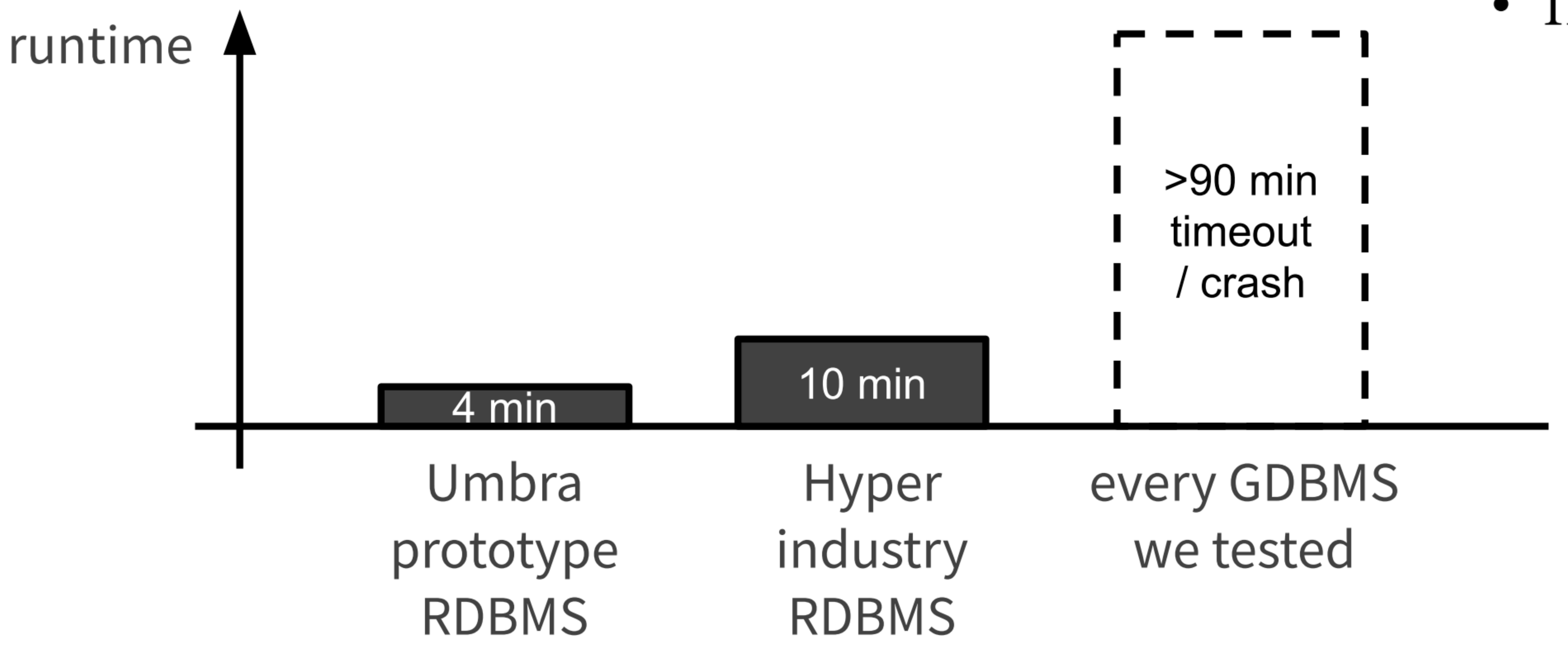
  "Find the customers who decreased their purchase amounts on their most recent order"
- 30 Unique Timings
- At least 30 ways for the Query Planner and Optimizer to execute

1. **46ms Peso's GianLuca-copy**
2. **46ms Gianluca Again**
3. **46ms mByther 2**
4. 50ms Gianluca Sartori
5. 60ms KevRiley
6. 60ms Paul Ireland
7. 60ms Kevriley 4
8. **60ms Celko**
9. 63ms Peso 3
10. 63ms KevRiley 5
11. 63ms giancula 3
12. 106ms Eric Pratley
13. 110ms Quan 2
14. 173ms girish Bhat
15. 280ms RobertFolkerts
16. 283ms Wm Brewer
17. 296ms Herman 2
18. 313ms AndyM
19. 330ms Peso
20. 330ms ivan.yong
21. 330ms Peso 4
22. 340ms andriy.zabavskyy
23. **343ms maxdemarzi**
24. 360ms Mark Marinovic
25. 360ms mByther 1
26. 376ms John McVay
27. 390ms Peso 2
28. 390ms Peter Brinkhaus
29. 390ms plamen
30. 423ms Alex Kuznetsov
31. 436ms Gustavo 2
32. 453ms back_of_the_fleet
33. **453ms maxdemarzi**
34. 560ms v_paronov
35. 610ms Steve Rowland
36. 610ms eemore-571761
37. 643ms Chris Howarth
38. 690ms Herman van Midden
39. 703ms jfortuny
40. 736ms Quan_L_Nguyen
41. 750ms Chris Howarth 2
42. 763ms Gustavo
43. 1030ms GSquared
44. **10766ms Ramesh**

CWI
Centrum Wiskunde & Informatica

GDBMS performance for subgraph queries

- Load the data: 100M vertices, 650M edges
- Run all 9 queries one-by-one (count number of matches)
- Environment: cloud VM, 370GB RAM, 48 vCPU cores



LDBC BI SQL Queries

- The queries changed over time
  - Over 10x improvement gained by rewriting queries
  - *The optimizer should have been doing what we had to do by hand!*
    - Remove redundant joins with redundant relations
    - Common subquery elimination

10x faster

JAN 28 2019

2 COMMENTS

CYPHER, JAVA, PROBLEMS, RANDOM

EDIT

NEO4J STORED PROCEDURES FOR DEVS THAT DON'T KNOW JAVA (YET)


Stored Procedure
Look at me,
I am the query optimizer now

Not Only Declarative

Add a “real time” Procedural Query Interface

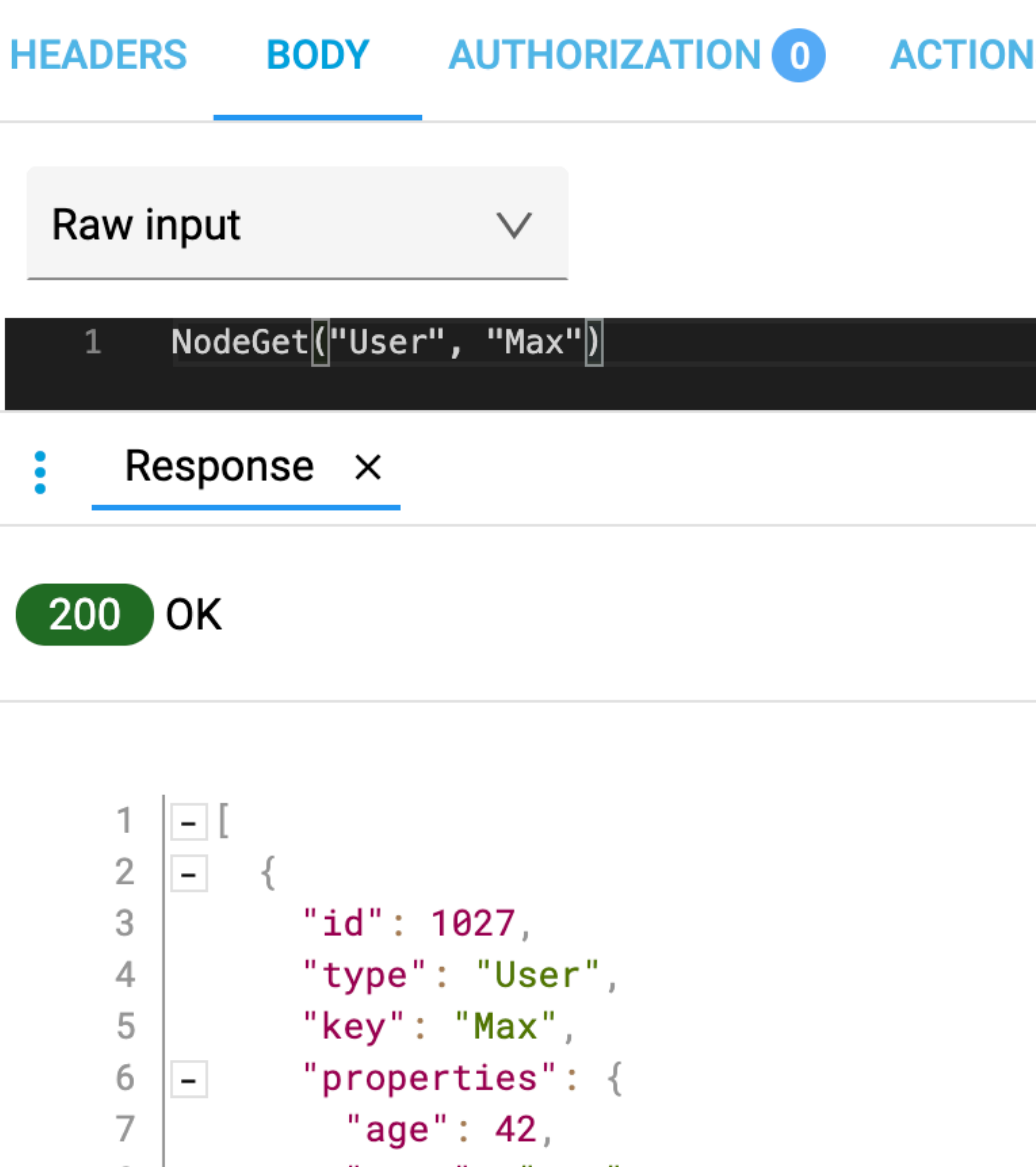
Lua
“Moon” in Portuguese

- Proven
  - Used in embedded systems and games
- Fast
  - Fastest scripting language I know of, and using LuaJIT
- Powerful, small and free (MIT)

Lua

As a Query Language

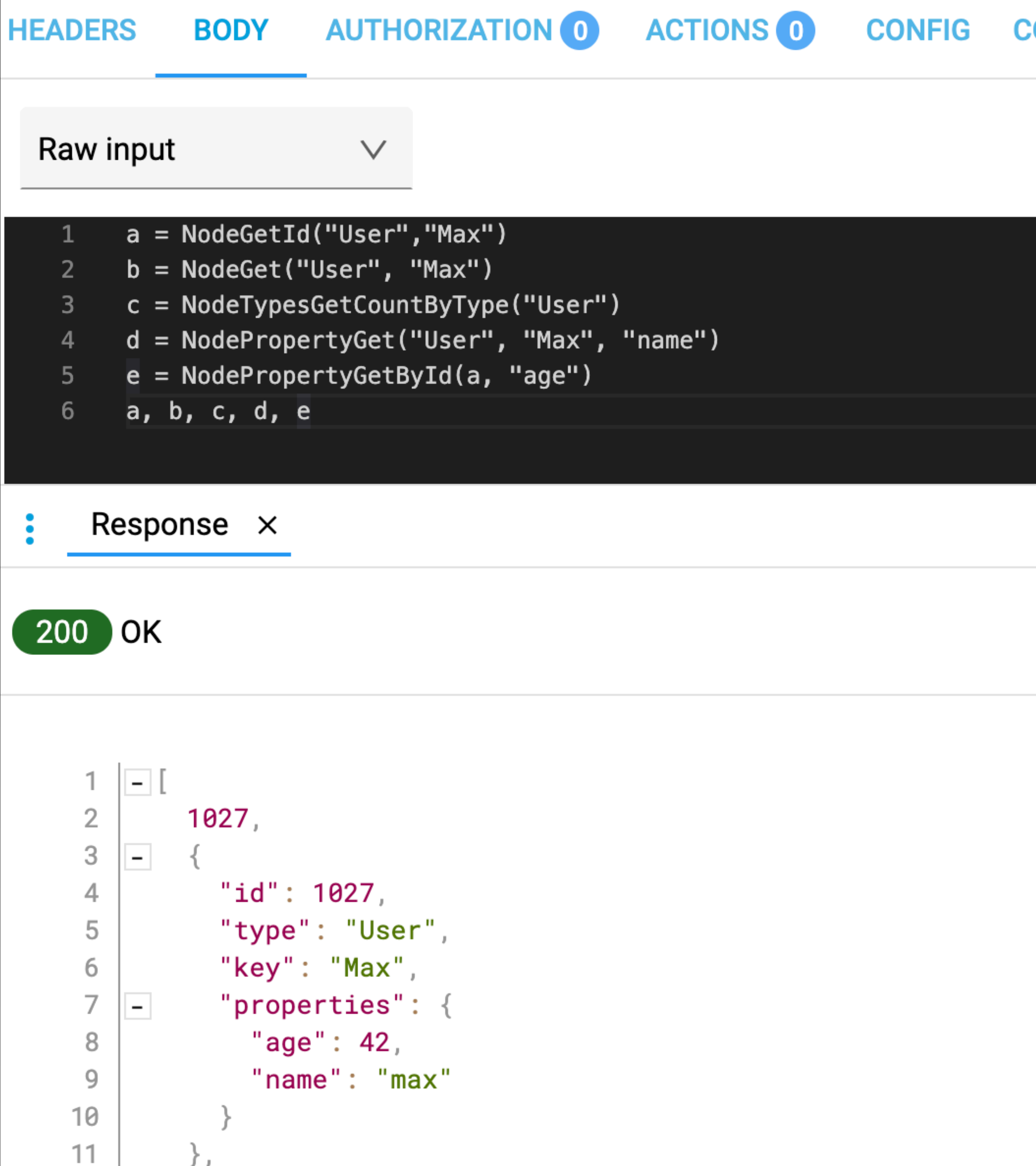
- Simple Queries



Lua

As a Query Language

- Simple Queries
- Pipelined Queries



Lua

As a Query Language

- Simple Queries
- Pipelined Queries
- Complex Queries

POST http://localhost:7243/db/rage/lua

HEADERS BODY AUTHORIZATION 0 ACTIONS 0 CONFIG

Raw input

```lua
names = {}
ids = NodeGetRelationshipsIds("User", "Max")
for k=1,#ids do
    v = ids[k]
    table.insert(names, NodePropertyGetById(v.node_id, "name"))
end
names
```

Response

200 OK

```json
[
    [
        "helene"
    ]
]
```

```
--ALL
get_city = function(person_id)
    return NodeGetNeighbors("Person", person_id, Direction.OUT, "IS_LOCATED_IN")[1]
end
```

Then in our queries we can just reuse it:

```
--ALL
ldbc_snb_is01 = function(person_id)
    local properties = NodeGetProperties("Person", person_id)
    local city = get_city(person_id)
    ...
```

RAGE DB

Community

Let's build something outrageous!

Follow RageDB on Twitter
Join us on Slack
Help out on GitHub

Script

Run | Clear | Favorite | h: 0 m: 0 s: 2 ms: 211

```
count = 0
person = "person"..math.random(NodeTypesGetCountByType("Person"))
person_id = NodeGetId("Person", person)
liked_items = NodeGetRelationshipsIdsByIdForDirectionForType(person_id, Direction.OUT, "LIKES")
item_likes = LinksGetRelationshipsIds(liked_items, Direction.IN, "LIKES")

for item, likes in pairs(item_likes) do
  other_person_likes = LinksGetRelationshipsIdsForDirectionForType(likes, Direction.OUT, "LIKES")
  for other_person, likes2 in pairs(other_person_likes) do
    for iter = 1,#likes2 do
        count = count + 1
    end
 end
end
count
```

Bulk API

Response

[50131000]

On 4 Cores

```
./wrk -c 192 -t 32 -d 70s -s rage_get_node.lua --latency -R 190000
Running 1m test @ http://x.x.x.x:7243
  32 threads and 192 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.31ms  565.03us  11.08ms   74.35%
    Req/Sec     6.26k   400.79     8.67k    73.88%
  Latency Distribution (HdrHistogram - Recorded Latency)
 50.000%    1.23ms
 75.000%    1.60ms
 90.000%    1.99ms
 99.000%    2.92ms
 99.900%    5.84ms
 99.990%    8.65ms
 99.999%    9.88ms
100.000%   11.09ms
  13297120 requests in 1.17m, 2.79GB read
Requests/sec: 189969.52
Transfer/sec:     40.88MB
```

190k Requests / Second


HELP
WANTED

Rage DB

An outrageous graph database

@rage_database

ragedb.com

GitHub.com/ragedb

hub.docker.com/u/ragedb