The Future of Data Infrastructure in the Agentic AI Era

(Technology Challenges and Research Opportunities in Data Systems)

Yannis Ioannidis

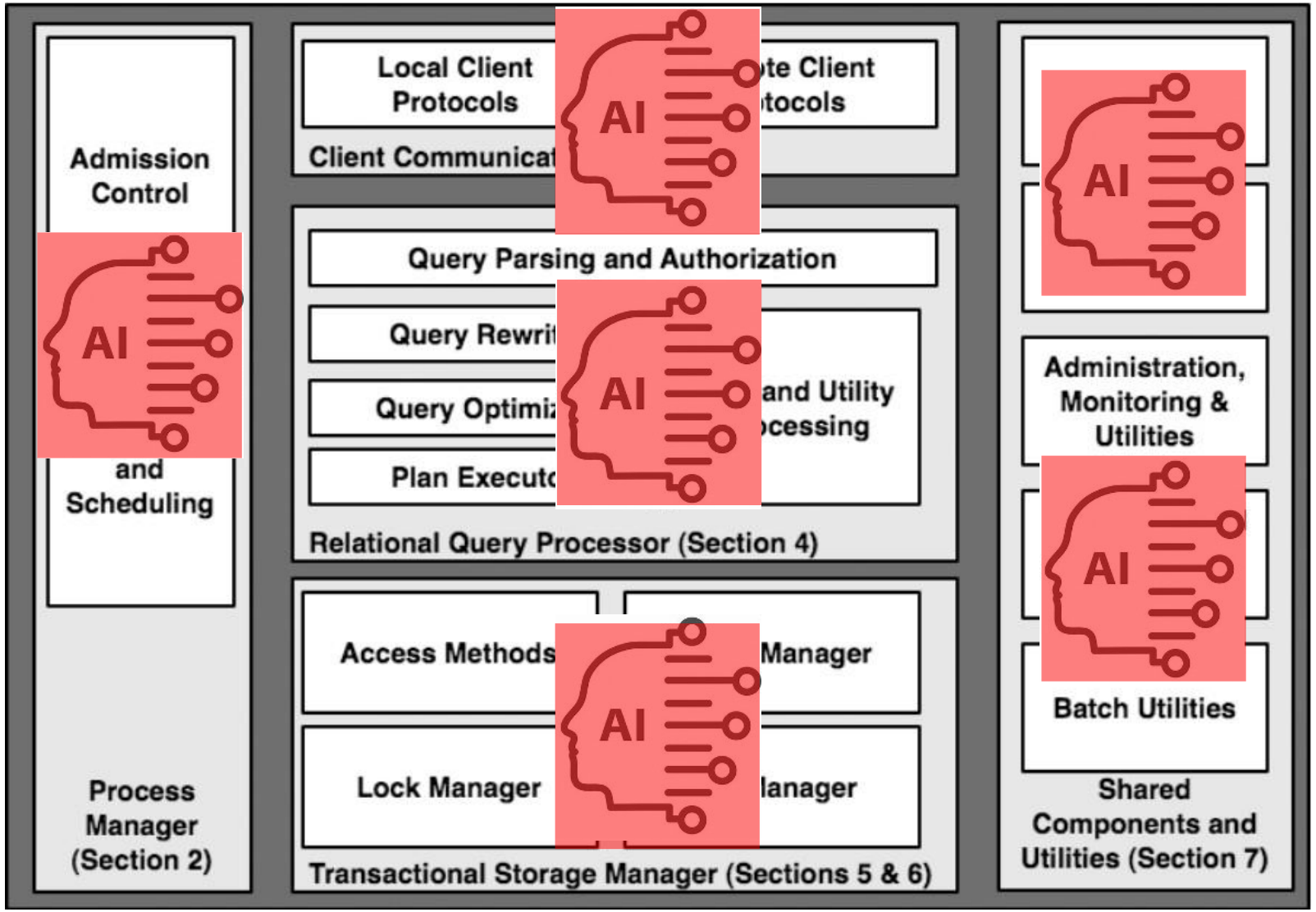


University of Athens **Athena Research Ce**

Past President of ACM

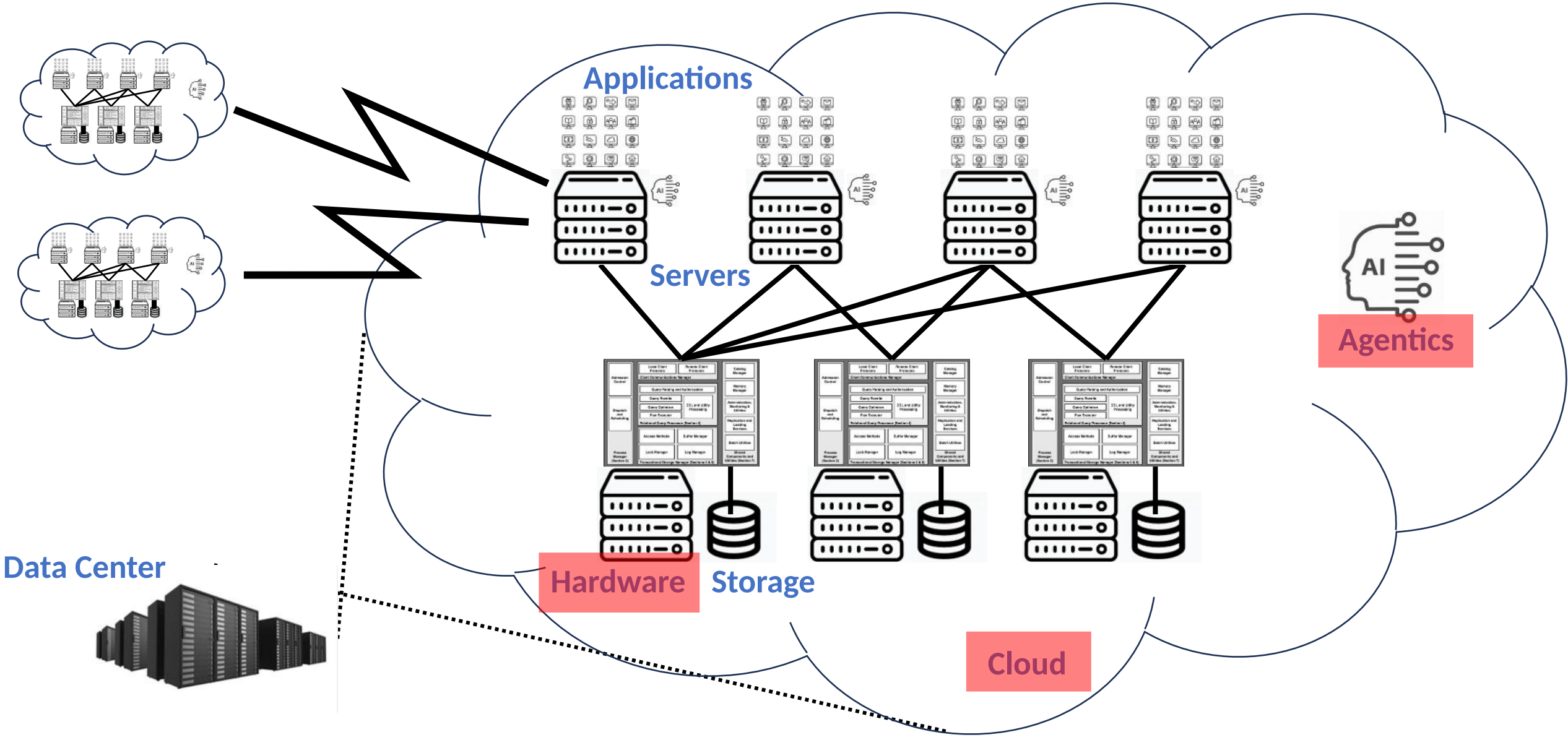
OLYMPIA AI Data Infrastructure Summit, Heraklion, Crete, Greece, 4-5 November 2026

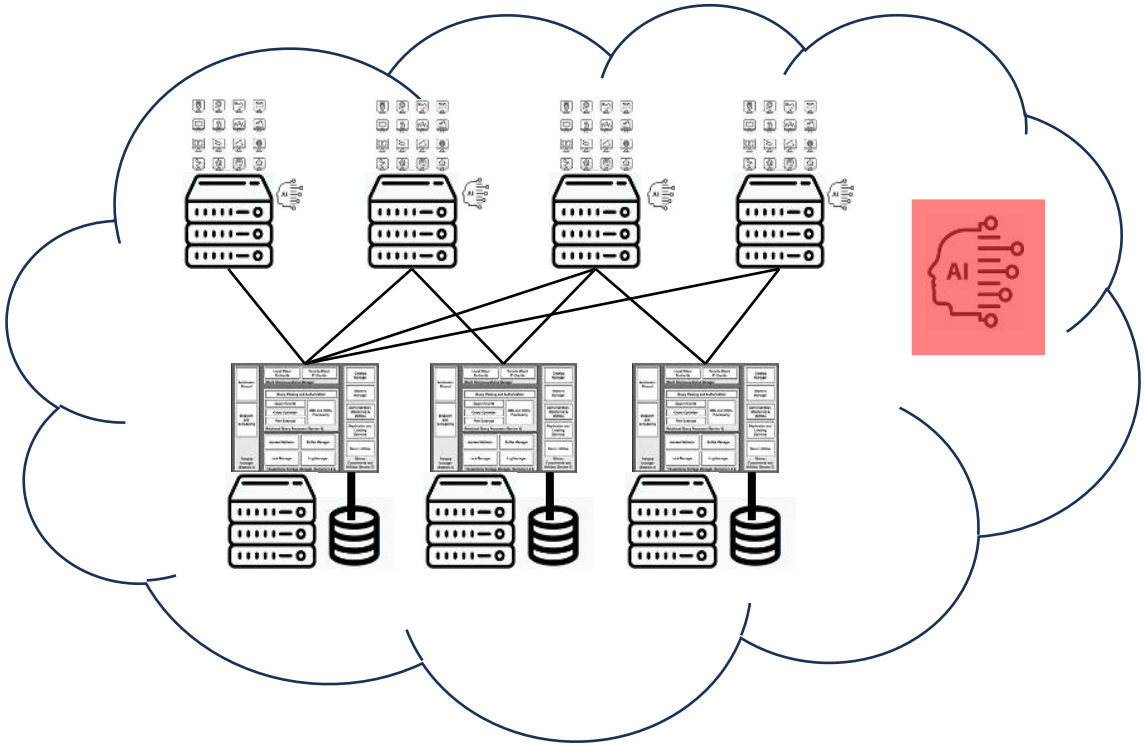
Basic Architecture of a Database Management System



Foundations and Trends in Databases (2007)
© Hellerstein, Stonebraker, Hamilton

Environment of a Database Management System

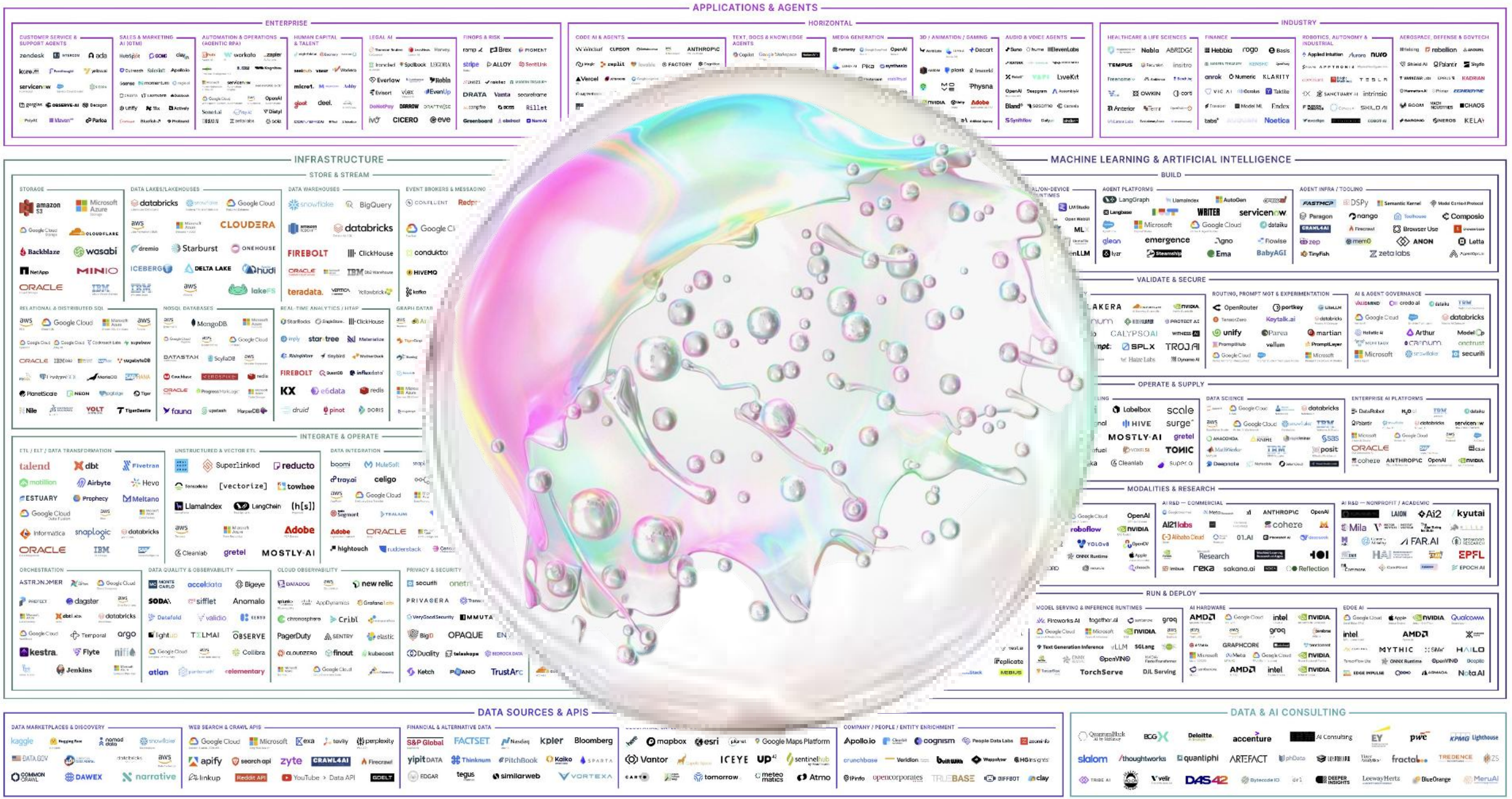




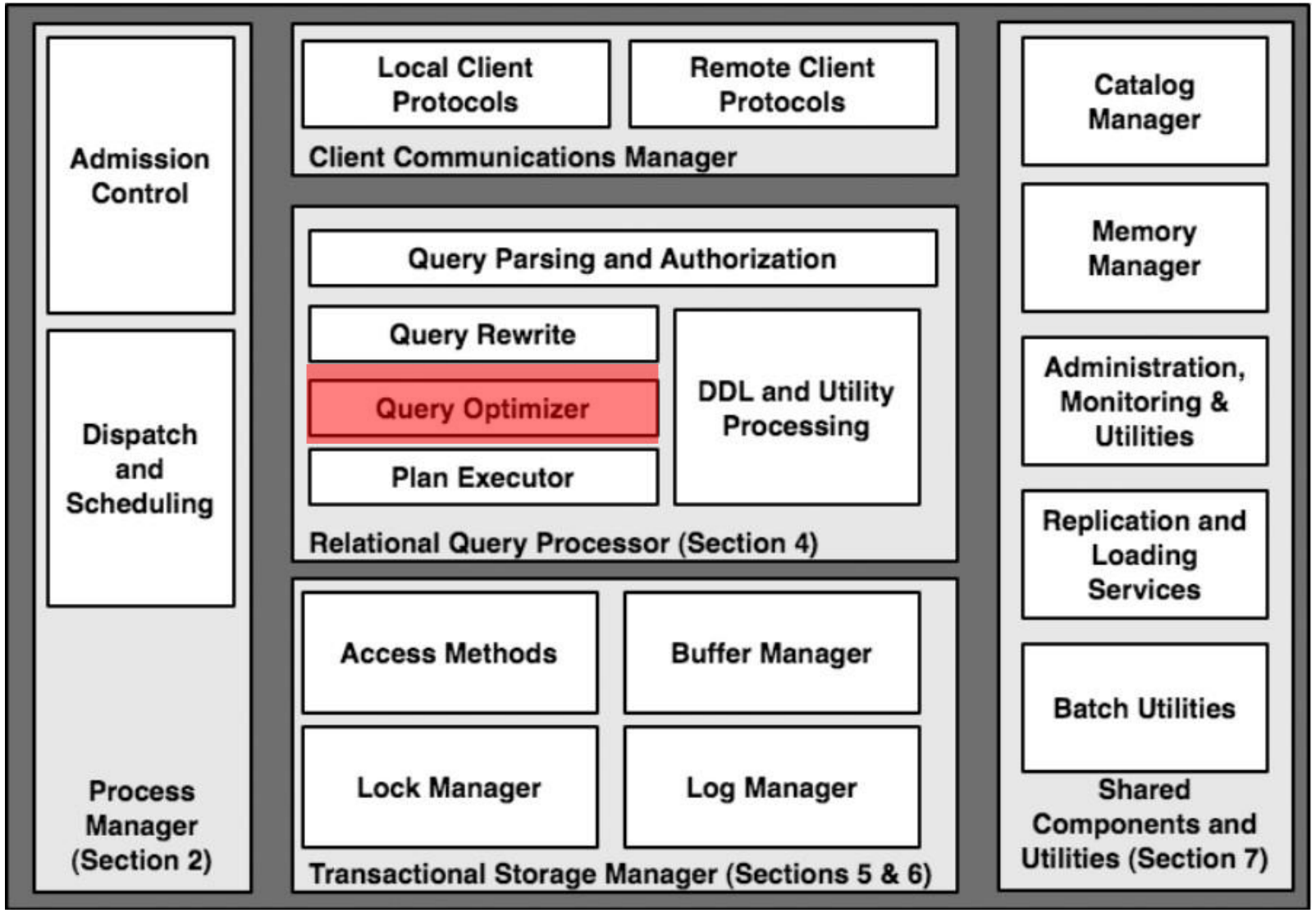
Technological Advances – AI (Agentic and other)

THE 2025 MAD (MACHINE LEARNING, ARTIFICIAL INTELLIGENCE & DATA) LANDSCAPE

MAD.FIRSTMARK.COM

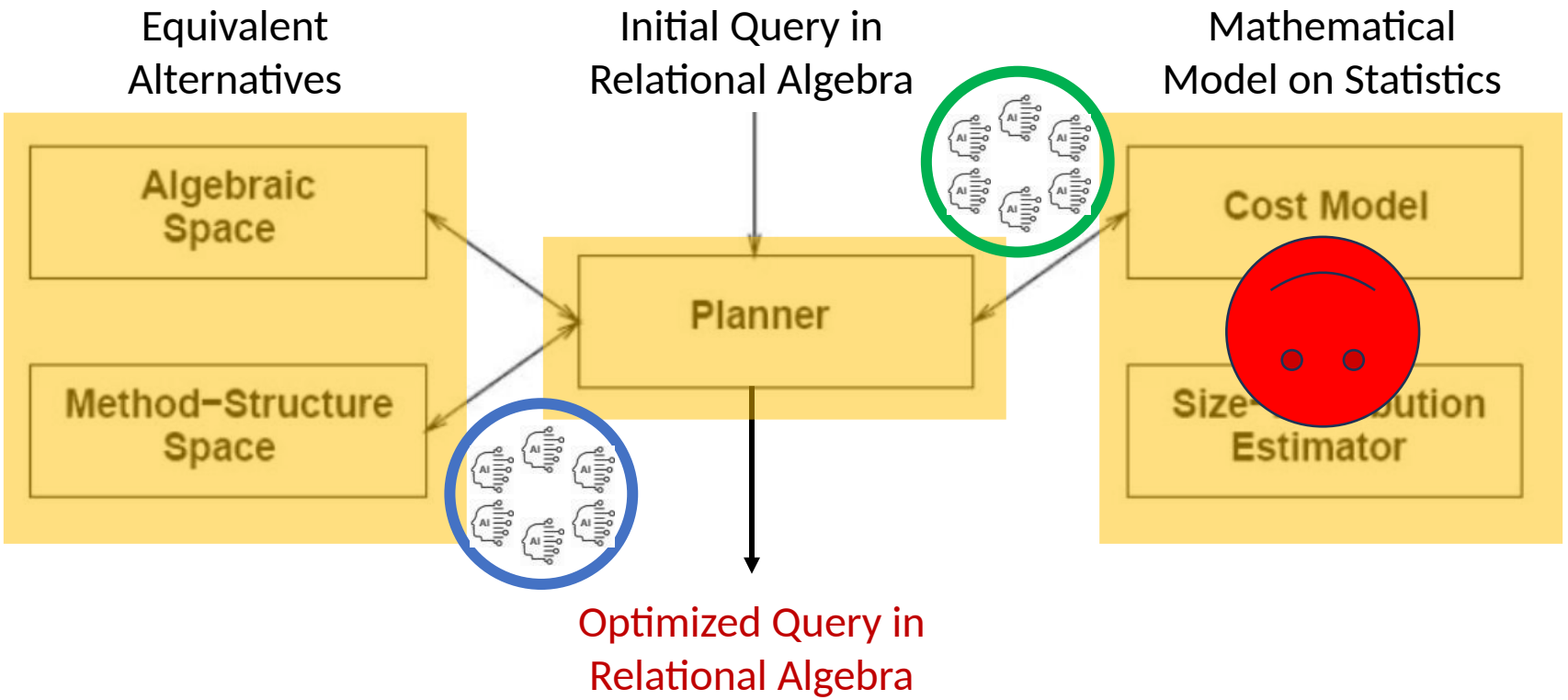


(AI-A) Agents for Query Optimization

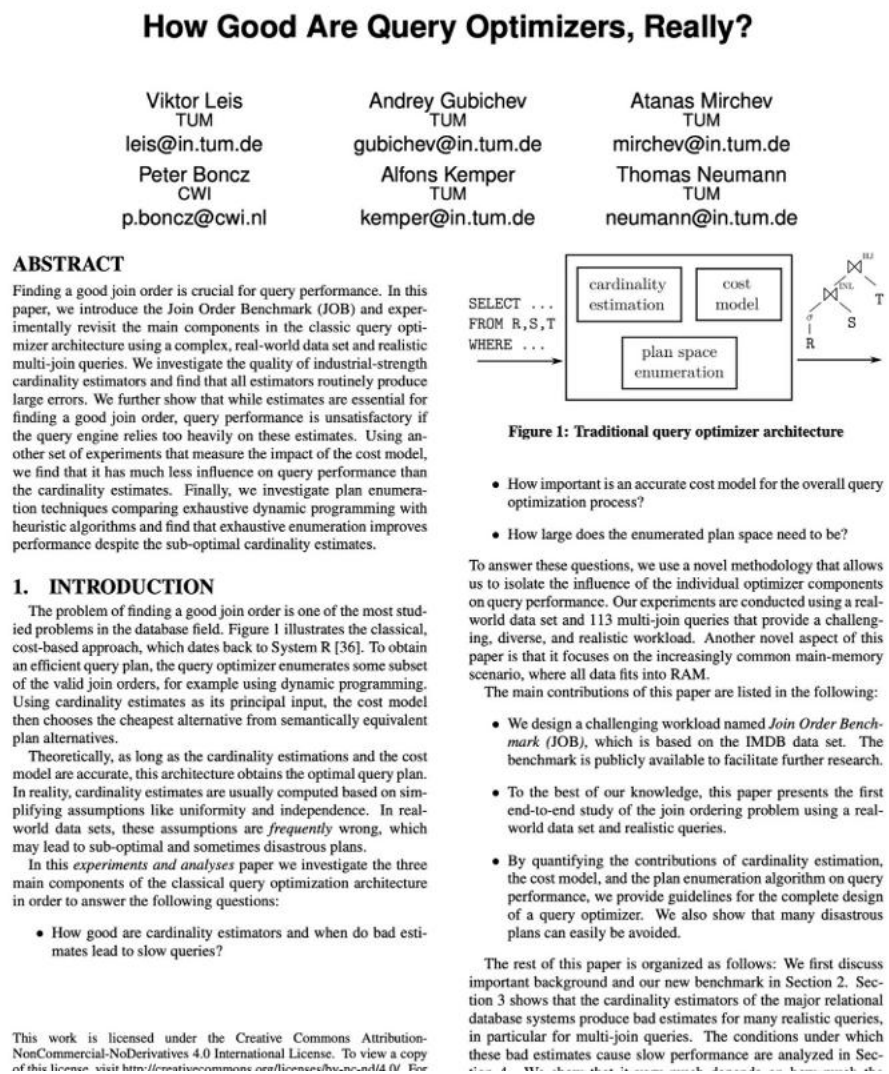


Foundations and Trends in Databases (2007)
© Hellerstein, Stonebraker, Hamilton

Query Optimization



Traditional Estimation of Query Cost and Result Sizes



How Good Are Query Optimizers, Really?

Viktor Leis, TUM, leis@in.tum.de
Andrey Gubichev, TUM, gubichev@in.tum.de
Atanas Mirchev, TUM, mirchev@in.tum.de
Peter Boncz, CWI, p.boncz@cwi.nl
Alfons Kemper, TUM, kemper@in.tum.de
Thomas Neumann, TUM, neumann@in.tum.de

ABSTRACT

Finding a good join order is crucial for query performance. In this paper, we introduce the Join Order Benchmark (JOB) and experimentally revisit the main components in the classic query optimizer architecture using a complex, real-world data set and realistic multi-join queries. We investigate the quality of industrial-strength cardinality estimators and find that all estimators routinely produce large errors. We further show that while estimates are essential for finding a good join order, query performance is unsatisfactory if the query engine relies too heavily on these estimates. Using another set of experiments that measure the impact of the cost model, we find that it has much less influence on query performance than the cardinality estimates. Finally, we investigate plan enumeration techniques comparing exhaustive dynamic programming with heuristic algorithms and find that exhaustive enumeration improves performance despite the sub-optimal cardinality estimates.

1. INTRODUCTION

The problem of finding a good join order is one of the most studied problems in the database field. Figure 1 illustrates the classical, cost-based approach, which dates back to System R [36]. To obtain an efficient query plan, the query optimizer enumerates some subset of the valid join orders, for example using dynamic programming. Using cardinality estimates as its principal input, the cost model then chooses the cheapest alternative from semantically equivalent plan alternatives.

Theoretically, as long as the cardinality estimations and the cost model are accurate, this architecture obtains the optimal query plan. In reality, cardinality estimates are usually computed based on simplifying assumptions like uniformity and independence. In real-world data sets, these assumptions are *frequently* wrong, which may lead to sub-optimal and sometimes disastrous plans.

In this *experiments and analyses* paper we investigate the three main components of the classical query optimization architecture in order to answer the following questions:

- How good are cardinality estimators and when do bad estimates lead to slow queries?

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For

Figure 1: Traditional query optimizer architecture

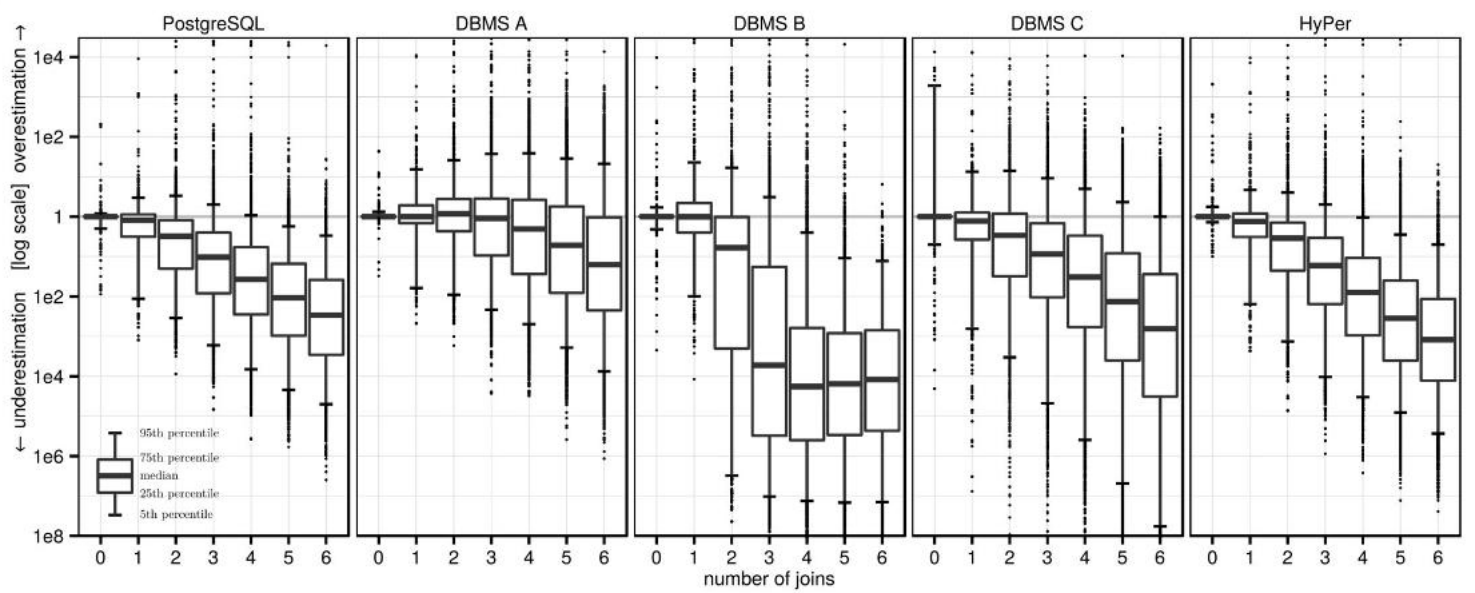
- How important is an accurate cost model for the overall query optimization process?
- How large does the enumerated plan space need to be?

To answer these questions, we use a novel methodology that allows us to isolate the influence of the individual optimizer components on query performance. Our experiments are conducted using a real-world data set and 113 multi-join queries that provide a challenging, diverse, and realistic workload. Another novel aspect of this paper is that it focuses on the increasingly common main-memory scenario, where all data fits into RAM.

The main contributions of this paper are listed in the following:

- We design a challenging workload named *Join Order Benchmark (JOB)*, which is based on the IMDB data set. The benchmark is publicly available to facilitate further research.
- To the best of our knowledge, this paper presents the first end-to-end study of the join ordering problem using a real-world data set and realistic queries.
- By quantifying the contributions of cardinality estimation, the cost model, and the plan enumeration algorithm on query performance, we provide guidelines for the complete design of a query optimizer. We also show that many disastrous plans can easily be avoided.

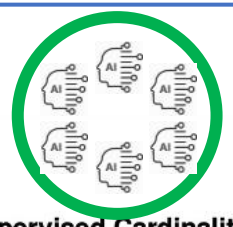
The rest of this paper is organized as follows: We first discuss important background and our new benchmark in Section 2. Section 3 shows that the cardinality estimators of the major relational database systems produce bad estimates for many realistic queries, in particular for multi-join queries. The conditions under which these bad estimates cause slow performance are analyzed in Sec-



- Poor estimates of both open-source and commercial systems
- Up to 4 orders of magnitude off

Courtesy of Matei Zaharia
(Databricks and UC Berkeley)

AI-based Estimations – Winning



The Case for Learned Index Structures

Tim Kraska* MIT kraska@mit.edu
Alex Beutel Google, Inc. abeutel@google.com
Ed H. Chi Google, Inc. edchi@google.com
Jeffrey Dean Google, Inc. jeff@google.com
Neoklis Pol Google, npoly@goog

ABSTRACT

Indexes are models: a B-Tree-Index can be seen as a model to map a key to the position of a record within a sorted array, a Hash-Index as a model to map a key to a position of a record within an unsorted array, and a BitMap-Index as a model to indicate if a data record exists or not. In this exploratory research paper, we start from this premise and posit that all existing index structures can be replaced with other types of models, including deep-learning models, which we term *learned indexes*. We theoretically analyze under which conditions learned indexes outperform traditional index structures and describe the main challenges in designing learned index structures. Our initial results show that our learned indexes can have significant advantages over traditional indexes. More importantly, we believe that the idea of replacing core components of a data management system through learned models has far reaching implications for future systems designs and that this work provides just a glimpse of what might be possible.

ACM Reference Format:
Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* , 16 pages. https://doi.org/10.1145/3183713.3196909

1 INTRODUCTION

Whenever efficient data access is needed, index structures are the answer, and a wide variety of choices exist to address the different needs of various access patterns. For example, B-Trees are the best choice for range requests (e.g., retrieve all records in a certain time frame); Hash-maps are hard to beat in performance for single key look-ups; and Bloom filters are typically used to check for record existence. Because of their importance for database systems and many other applications, indexes have been extensively tuned over the past decades to be more memory, cache and/or CPU efficient [11, 29, 36, 59].

Yet, all of those indexes remain general purpose data structures; they assume nothing about the data distribution and do not take advantage of more common patterns prevalent in real world data. For example, if the goal is to build a highly-tuned system to store and query ranges of fixed-length records over

a set of continuous integer keys (e.g., the keys 1 to 1 would not use a conventional B-Tree index over the the key itself can be used as an offset, making it an (than $O(\log n)$ operation to look-up any key or the of a range of keys. Similarly, the index memory size reduced from $O(n)$ to $O(1)$. Maybe surprisingly, si mizations are possible for other data patterns. In ot knowing the exact data distribution enables highly almost any index structure.

Of course, in most real-world use cases the da perfectly follow a known pattern and the enginee to build specialized solutions for every use case is u high. However, we argue that machine learning (up the opportunity to learn a model that reflects th in the data and thus to enable the automatic synth cialized index structures, termed **learned indexes** engineering cost.

In this paper, we explore the extent to which learn including neural networks, can be used to enhanc replace, traditional index structures from B-Trees filters. This may seem counterintuitive because N provide the semantic guarantees we traditionally with these indexes, and because the most powerful N neural networks, are traditionally thought of as b compute expensive. Yet, we argue that none of thes obstacles are as problematic as they might seem. In proposal to use learned models has the potential for benefits, especially on the next generation of hardv

In terms of semantic guarantees, indexes are al large extent learned models making it surprisingl forward to replace them with other types of ML m example, a B-Tree can be considered as a model wh key as an input and predicts the position of a data r sorted set (the data has to be sorted to enable effic requests). A Bloom filter is a binary classifier, whic a key predicts if a key exists in a set or not. Obvio exists subtle but important differences. For exampl filter can have false positives but not false negatives as we will show in this paper, it is possible to add differences through novel learning techniques and auxiliary data structures.

In terms of performance, we observe that ever ready has powerful SIMD capabilities and we spec many laptops and mobile phones will soon have a Processing Unit (GPU) or Tensor Processing Unit (also reasonable to speculate that CPU-SIMD/GPU/ be increasingly powerful as it is much easier to sc stricted set of (parallel) math operations used by n than a general purpose instruction set. As a resul

*Work done while the author was affiliated with Google.

2016 work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

SIGMOD'18, June 10–15, 2018, Houston, TX, USA
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4703-7/18/06.
https://doi.org/10.1145/3183713.3196909

Deep Unsupervised Cardinality Estimation

Zongheng Yang[1], Eric Liang[1], Amog Kamsetty[1], Chenggang Wu[1], Yan Duan[1]
Xi Chen[1,3], Pieter Abbeel[1,3], Joseph M. Hellerstein[1], Sanjay Krishnan[2], Ion Stoi
[1]UC Berkeley [2]University of Chicago [3]covariant.ai
[1]{zongheng,ericliang,amogkamsetty,cgwu,pabbeel,hellerstein,istoica}@berkeley.edu
[2]skr@uchicago.edu [3]{rocky,peter}@covariant.ai

ABSTRACT

Cardinality estimation has long been grounded in statistical tools for density estimation. To capture the rich multivariate distributions of relational tables, we propose the use of a new type of high-capacity statistical model: deep autoregressive models. However, direct application of these models leads to a limited estimator that is prohibitively expensive to evaluate for range or wildcard predicates. To produce a truly usable estimator, we develop a Monte Carlo integration scheme on top of autoregressive models that can efficiently handle range queries with dozens of dimensions or more.

Like classical synopses, our estimator summarizes the data without supervision. Unlike previous solutions, we approximate the joint data distribution without any independence assumptions. Evaluated on real-world datasets and compared against real systems and dominant families of techniques, our estimator achieves single-digit multiplicative error at tail, an up to 90× accuracy improvement over the second best method, and is space- and runtime-efficient.

PVLDB Reference Format:
Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. Deep Unsupervised Cardinality Estimation. *PVLDB*, 13(3): 279-292, 2019.
DOI: https://doi.org/10.14778/3368289.3368294

1. INTRODUCTION

Cardinality estimation is a core primitive in query optimization [42]. One of its main tasks is to accurately estimate the *selectivity* of a SQL predicate—the fraction of a relation selected by the predicate—without actual execution. Despite its importance, there is wide agreement that the problem is still unsolved [26,28,36]. Open-source and commercial DBMSes routinely produce up to $10^4-10^8\times$ estimation errors on queries over a large number of attributes [26].

The fundamental difficulty of selectivity estimation comes from condensing information about data into summaries [18]. The predominant approach in database systems today is to collect single-column summaries (e.g., histograms and

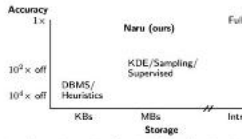


Figure 1: Approximating the joint data distributic enjoys high estimation accuracy and space efficienc

sketches), and to combine these coarse-graine suming column independence. This represent the spectrum, where the summaries are fast and cheap to store, but compounding errors the coarse information and over-simplifying assumptions. On the other end of the spectrum the *joint data distribution* of a relation (the freq unique tuple normalized by the relation's card fect selectivity "estimates" can be read off or c integration over the distribution. However, th tractable to compute or store for all but the tin Thus, traditional selectivity estimators face the off between the amount of information captured to construct, store, and query the summary.

An accurate and compact *joint approximatio* better design points in this tradeoff space (Figu advances in deep unsupervised learning have off ing tools in this regard. While it was previo intractable to approximate the data distributi tion in its full form [7, 14], *deep autoregressi* type of density estimator, have succeeded in m dimensional data such as images, text, and a 50]. However, these models only estimate *poin* in query processing terms, they only handle eq cates (e.g., "what is the fraction of tuples with $100?"). Full-featured selectivity estimation dling not only equality but also range predicates fraction of tuples have price less than $100 and than 10 lbs?"). Naive estimation of the rang integrating over the query region requires sun enormous number of points. In an 11-dimensic consider, a challenging range query has 10^{10} query region, which would take more than 1,000 hours to sum over by a naive enumeration scheme. A full-featured selectivity estimator, therefore, requires new techniques beyond the state of the art.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 13, No. 3
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3368289.3368294

Bao: Making Learned Query Optimization Practical

Ryan Marcus MIT & Intel Labs ryanmarcus@csail.mit.edu
Parimarjan Negi MIT pnegi@csail.mit.edu
Hongzi Mao MIT hongzi@csail.mit.edu
Nesime Tatbul MIT & Intel Labs tatbul@csail.mit.edu
Mohammad Alizadeh MIT alizadeh@csail.mit.edu
Tim Kraska MIT kraska@csail.mit.edu

ABSTRACT

Recent efforts applying machine learning techniques to query optimization have shown few practical gains due to substantive training overhead, inability to adapt to changes, and poor tail performance. Motivated by these difficulties, we introduce Bao (the Bandit optimizer). Bao takes advantage of the wisdom built into existing query optimizers by providing per-query optimization hints. Bao combines modern tree convolutional neural networks with Thompson sampling, a well-studied reinforcement learning algorithm. As a result, Bao automatically learns from its mistakes and adapts to changes in query workloads, data, and schema. Experimentally, we demonstrate that Bao can quickly learn strategies that improve end-to-end query execution performance, including tail latency, for several workloads containing long-running queries. In cloud environments, we show that Bao can offer both reduced costs and better performance compared with a commercial system.

CCS CONCEPTS

• **Information systems → Query optimization**.

KEYWORDS

Query optimization; machine learning; reinforcement learning

ACM Reference Format:
Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3452838

1 INTRODUCTION

Query optimization is an important task for database management systems. Despite decades of study [70], the most important elements of query optimization – cardinality estimation and cost modeling – have proven difficult to crack [45]. Several works have applied machine learning techniques to these stubborn problems [37, 40, 44, 51, 53, 59, 72, 73, 76]. While all of these new solutions demonstrate remarkable results, we argue that none of the techniques are yet practical, as they suffer from several fundamental problems:

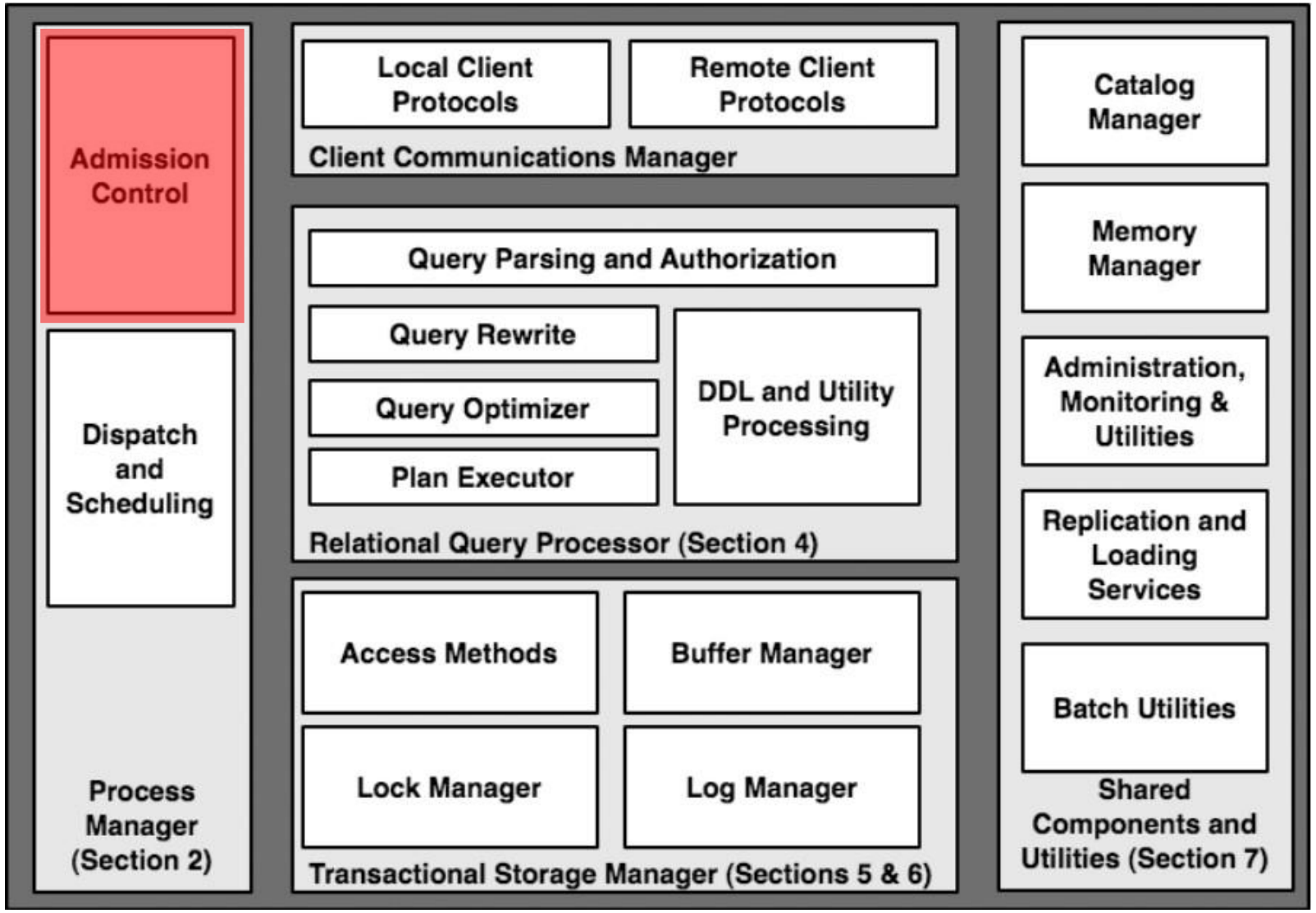
CC BY

(1) **Long training time.** Most proposed machine learning techniques require an impractical amount of training data before they have a positive impact on query performance. For example, ML-powered cardinality estimators based on supervised learning require gathering precise cardinalities from the underlying data, a prohibitively expensive operation in practice (this is why we wish to estimate cardinalities in the first place). Reinforcement learning techniques must process thousands of queries before outperforming traditional optimizers, which (when accounting for data collection and model training) can take on the order of days [51].
(2) **Inability to adjust to data and workload changes.** While performing expensive training operations once may already be impractical, changes in query workload, data, or schema can make matters worse. Cardinality estimators based on supervised learning must be retrained when data changes, or risk becoming stale. Several proposed reinforcement learning techniques assume that both the workload and the schema remain constant, and require complete retraining when this is not the case [40, 51, 53, 59].
(3) **Tail catastrophe.** Recent work has shown that learning techniques can outperform traditional optimizers *on average*, but often perform catastrophically (e.g., 100x regression in query performance) in the tail [27, 51, 58, 60]. This is especially true when training data is sparse. While some approaches offer statistical guarantees of their dominance in the average case [76], such failures, even if rare, are unacceptable in many real world applications.
(4) **Black-box decisions.** While traditional cost-based optimizers are already complex, understanding query optimization is even harder when black-box deep learning approaches are used. Moreover, in contrast to traditional optimizers, current learned optimizers do not provide a way for database administrators to influence or understand the learned component's query planning.
(5) **Integration cost.** To the best of our knowledge, all previous learned optimizers are still research prototypes, offering little to no integration with a real DBMS. None even supports all features of standard SQL, not to mention vendor specific features. Hence, fully integrating any learned optimizer into a commercial or open-source database system is not a trivial undertaking.

To the best of our knowledge, Bao (Bandit optimizer) is the first learned optimizer which overcomes the aforementioned problems. Bao is fully integrated into PostgreSQL as an extension, and can be easily installed without the need to recompile PostgreSQL. The database administrator (DBA) just needs to download our open-

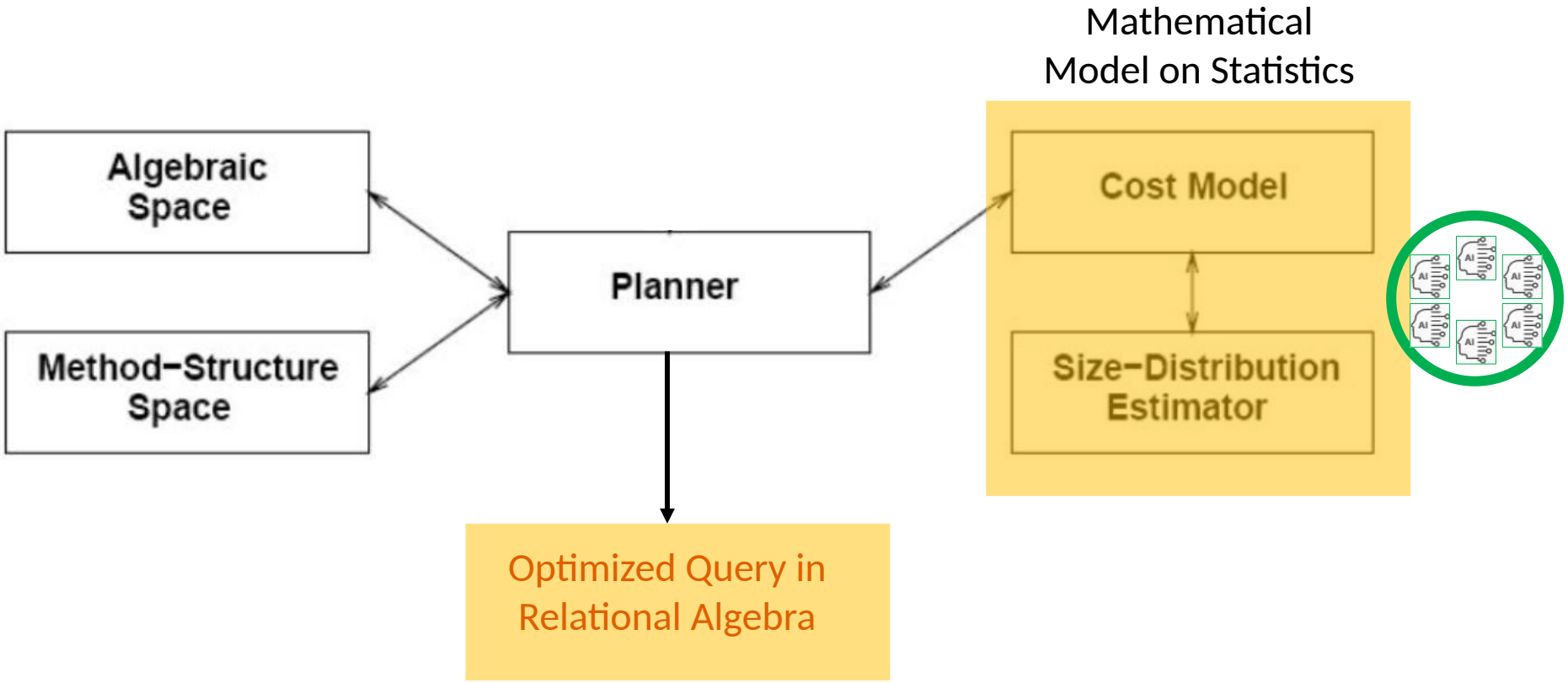
Courtesy of Matei Zaharia
(Databricks and UC Berkeley)

(AI-B) Agents for Admission Control



Foundations and Trends in Databases (2007)
© Hellerstein, Stonebraker, Hamilton

Cost (Estimation)-Based Admission Control



Cost (Estimation)-Based Admission Control

- Goal: High interactivity
- Prioritize short/quick queries over long/slow queries
- Reject/queue up long/slow queries for when machine load is low



```
select * from events where event_id = 123
```



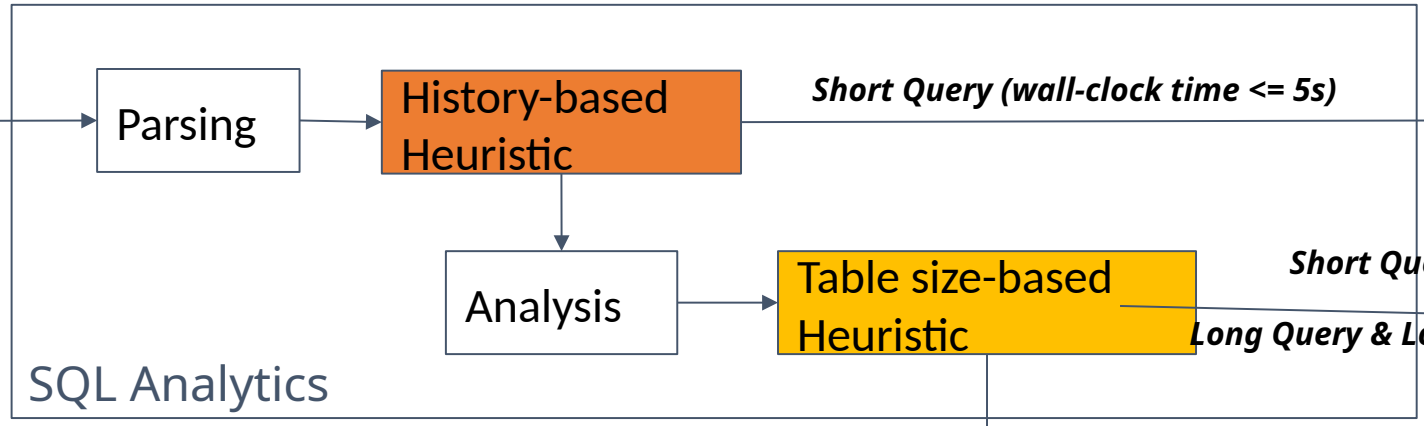
```
select date, COUNT(DISTINCT(user))
from events group by date
```



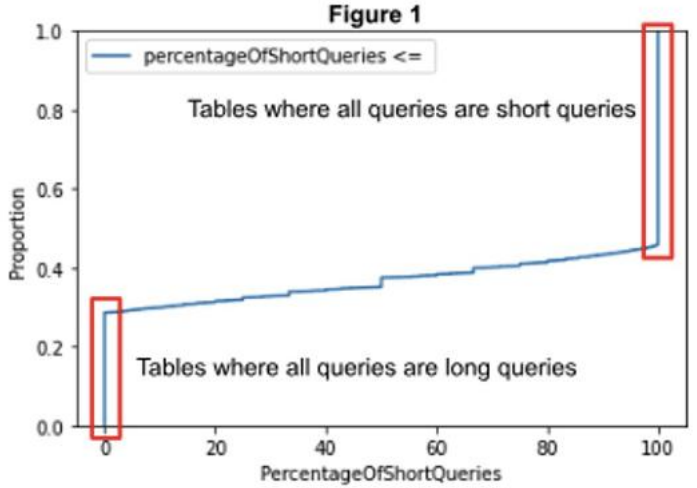
Courtesy of Matei Zaharia
(Databricks and UC Berkeley)

Heuristics-based Model

```
select * from
events where
event_id = 123
```

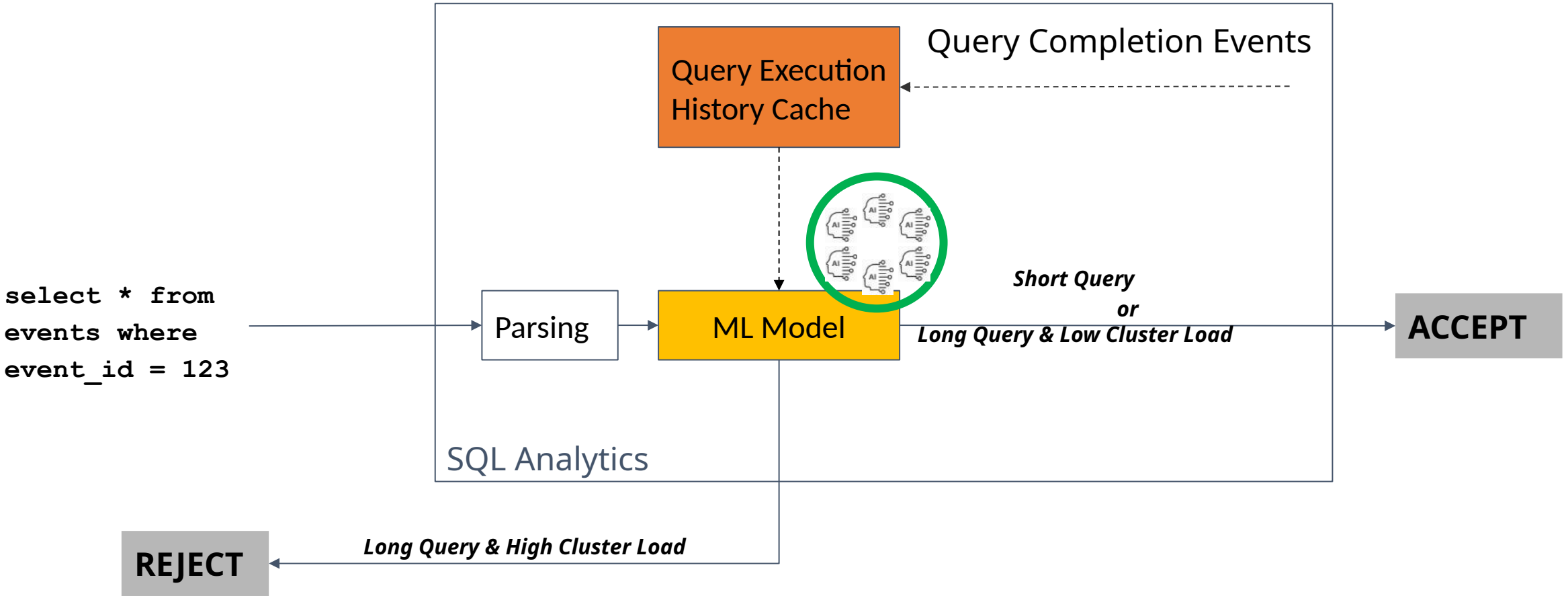


Ambigious / indecisive historical data



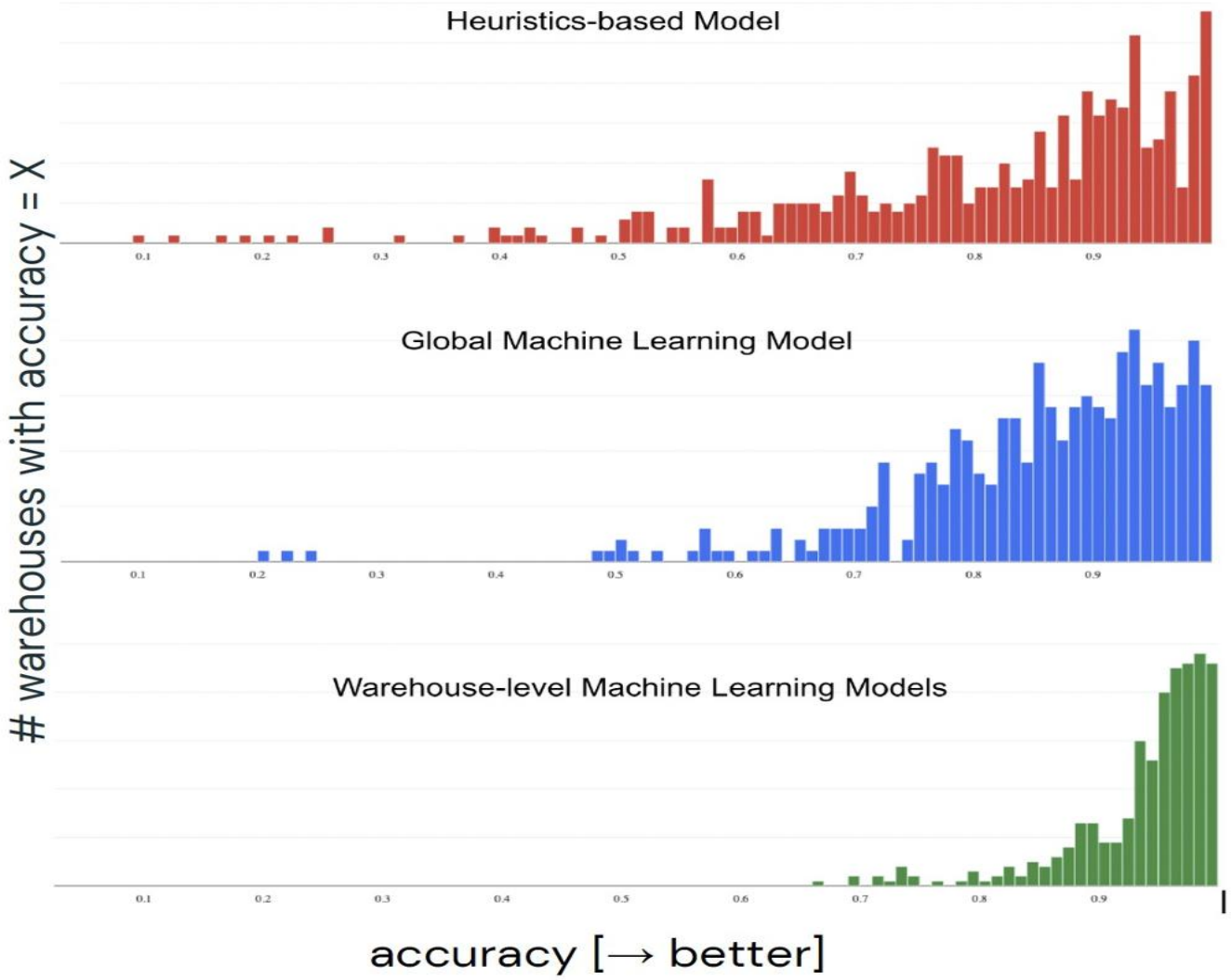
Courtesy of Matei Zaharia (Databricks and UC Berkeley)

ML-based Model



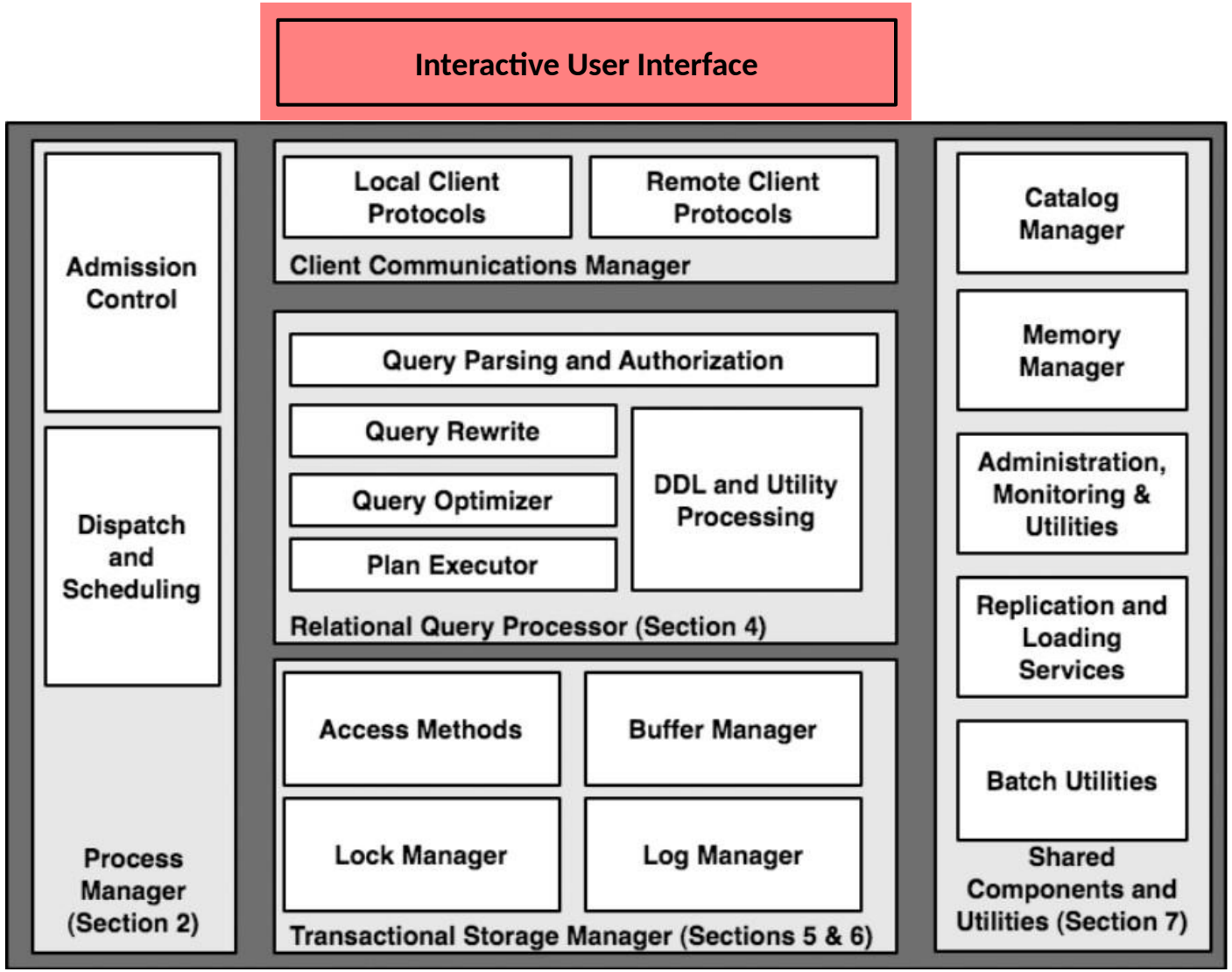
Courtesy of Matei Zaharia
(Databricks and UC Berkeley)

Heuristics vs. ML at Different Granularities



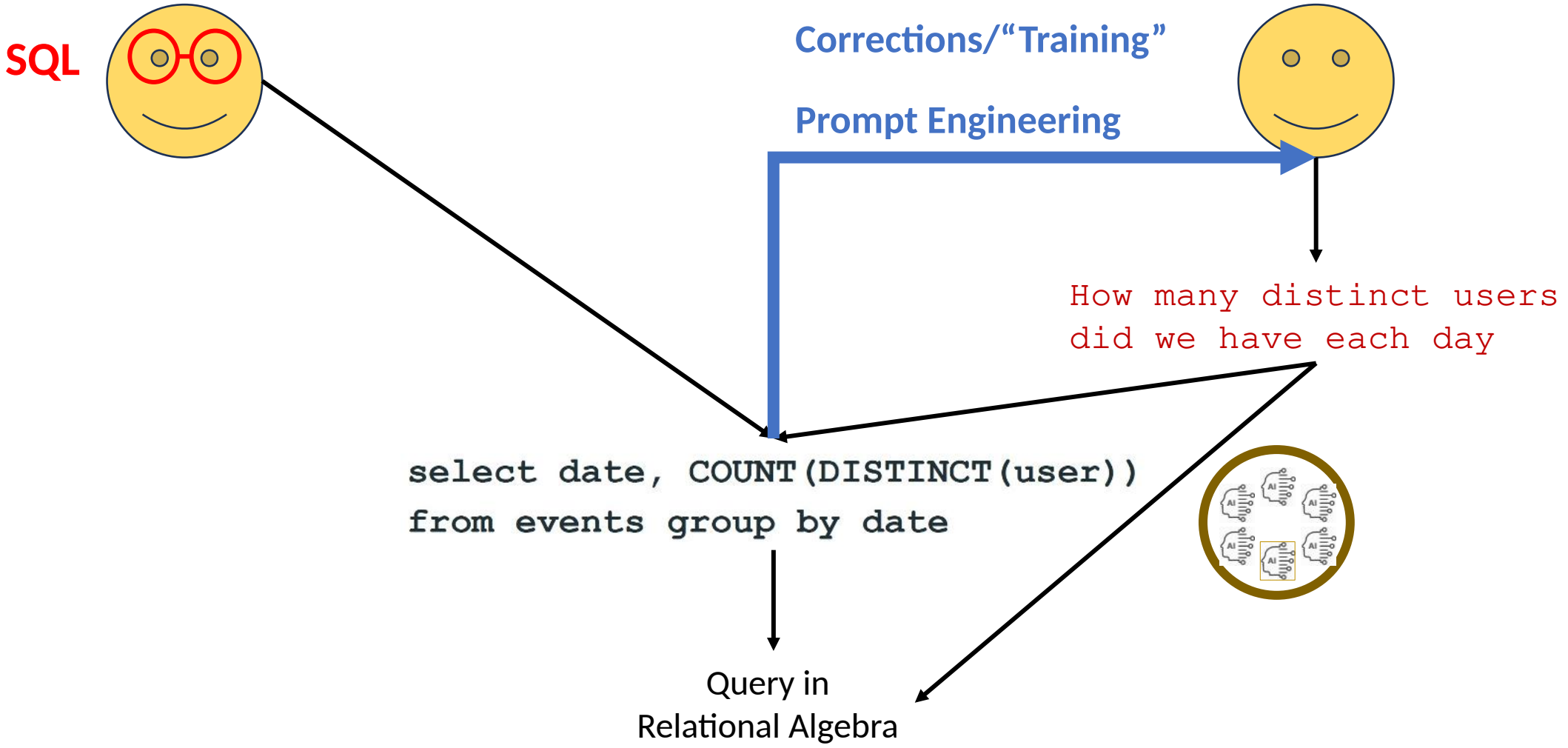
Courtesy of Matei Zaharia
(Databricks and UC Berkeley)

(AI-C) LLM for User Interface Enrichment



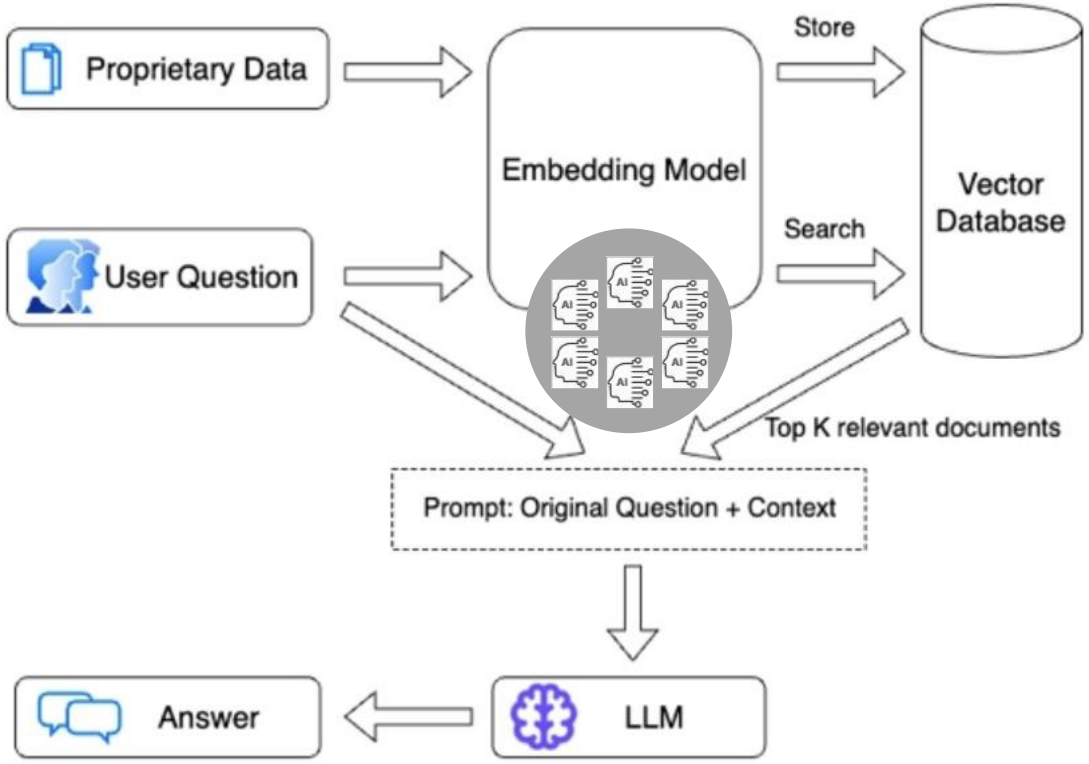
Foundations and Trends in Databases (2007)
© Hellerstein, Stonebraker, Hamilton

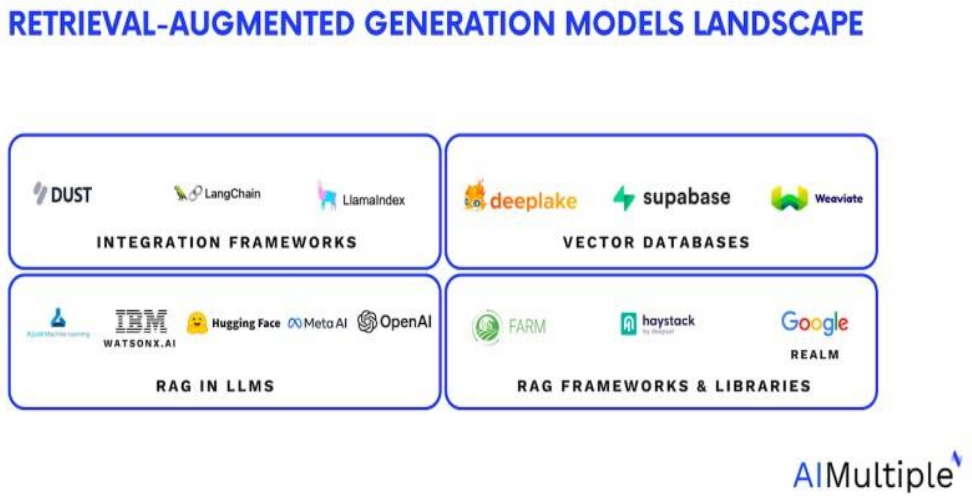
Natural Language as Query Language



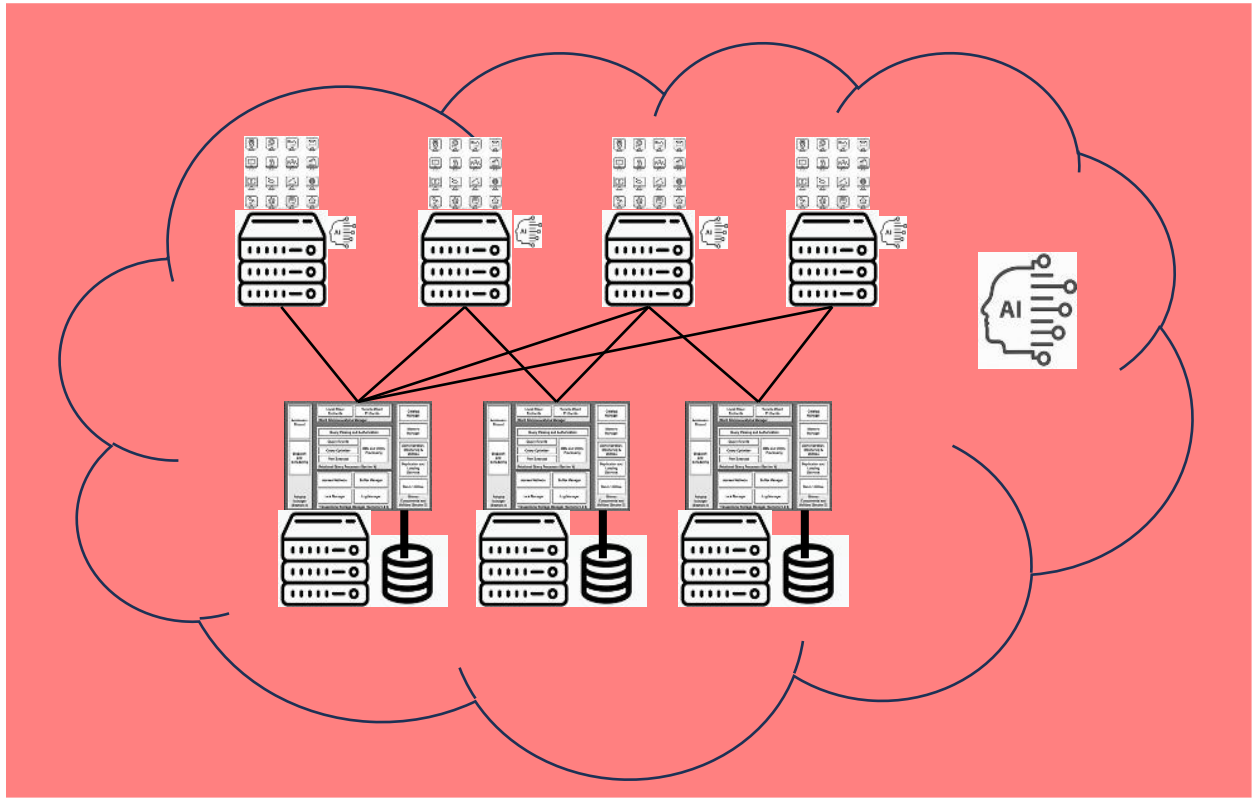
(AI-D) Database Retrieval Enhancing LLMs

- RAG: Retrieval-Augmented Generation
- Optimizing LLM output with targeted information …
  … retrieved from databases without modifying



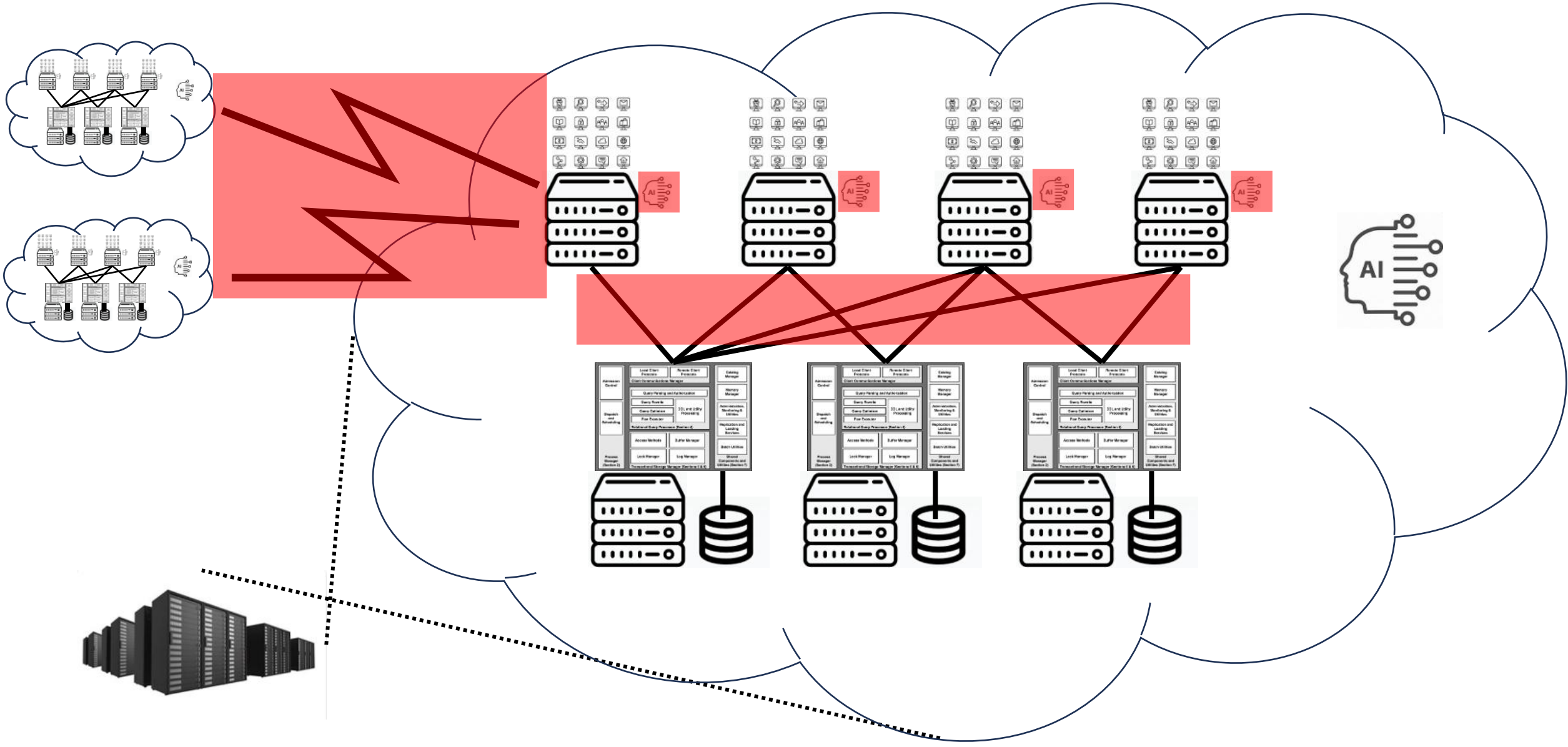


Original Pictures by Hazal Simsek (AIMultiple)



Technological Advances – AI + Cloud

(CL-A) Privacy-Based AI Processing of Shared Sensitive Data



Sensitive Data

- Organizations want to exchange/share/integrate (extremely large) datasets …

  … without actually revealing the details of the data itself

- Business intelligence (industrial competition)
- Personal privacy (national laws and regulations)
- Community identity (protected traditions)

- Often all data stored on same cloud data center
- Need Trusted Experimentation Facilities (TEFs) or other TFs

Example Use Cases

- Companies exchanging data with suppliers & partners
- Data vendors selling large datasets
- Retailers, banks, advertisers, etc., finding joint customers
- Hospitals sharing rare-disease data for medical discoveries



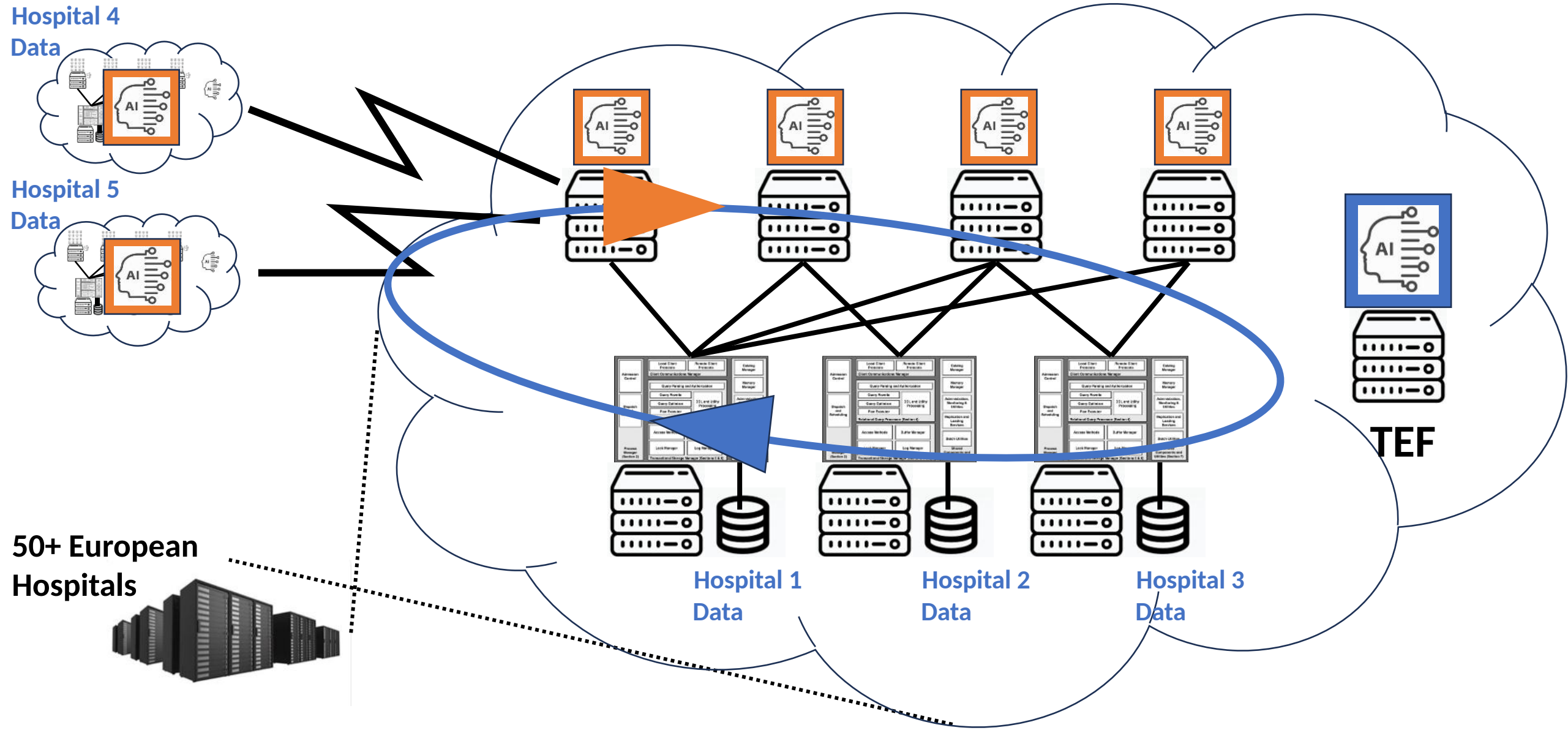






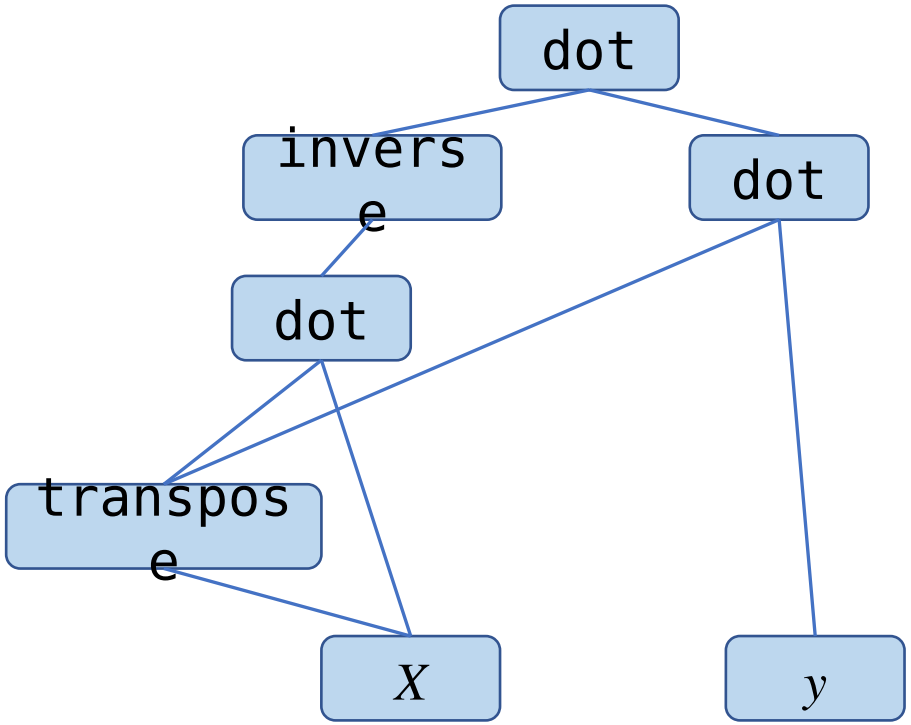
(parts) Courtesy of Matei Zaharia
(Databricks and UC Berkeley)

Federated Learning



Realistic Example: Linear Regression

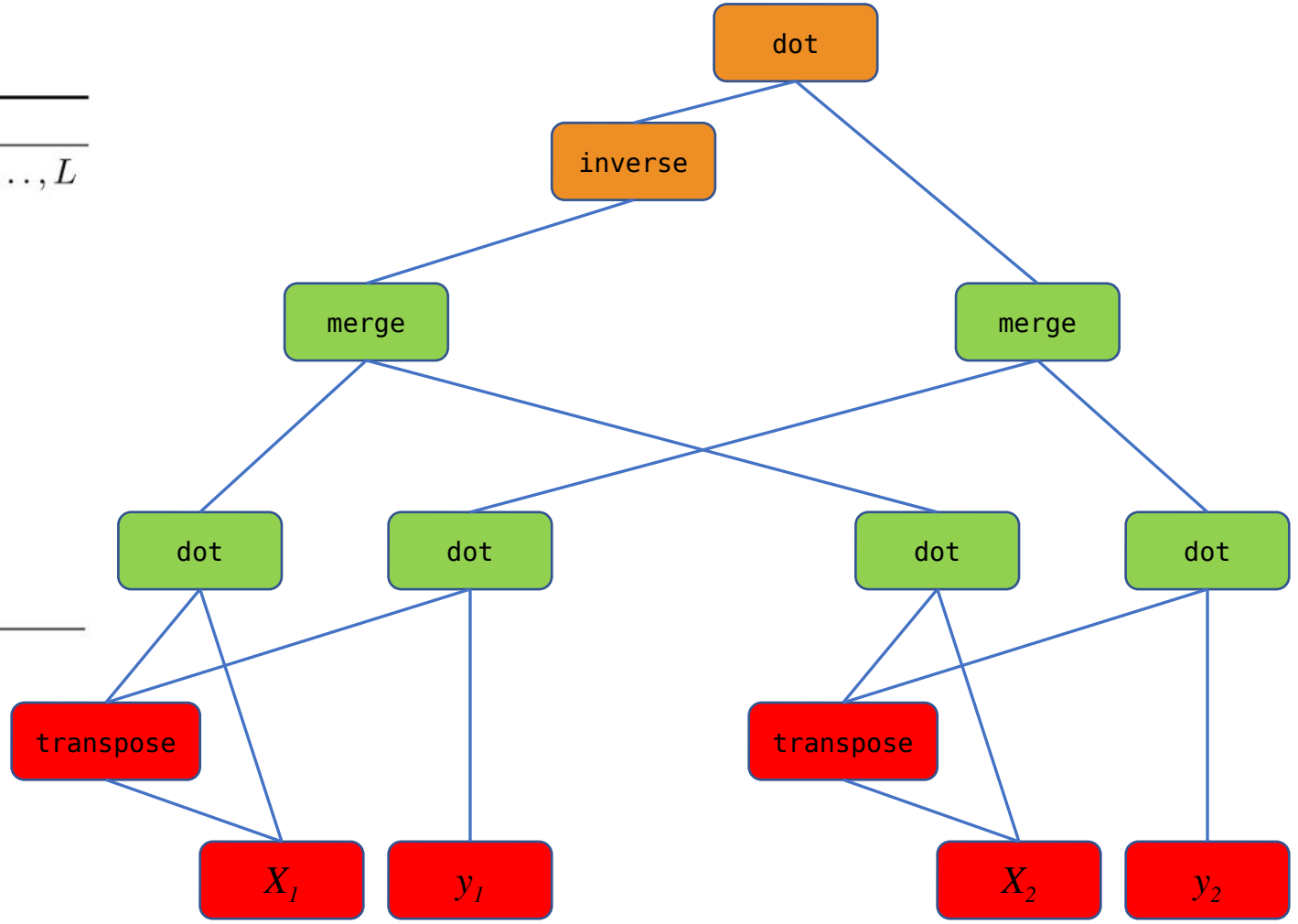
$$\beta = (X^\top X)^{-1} X^\top y$$



Federated Linear Regression

Algorithm LINEAR REGRESSION

procedure LOCAL($X^{(l)}$) ▷ run for $l = 1, \ldots, L$
  $\mathbf{A}^{(l)} \leftarrow X^{(l)\top} X^{(l)}$
  $\mathbf{b}^{(l)} \leftarrow X^{(l)\top} y^{(l)}$
  return $\mathbf{A}^{(l)}$, $\mathbf{b}^{(l)}$
end procedure
procedure GLOBAL($\{\mathbf{A}^{(l)}, \mathbf{b}^{(l)}\}$)
  $\mathbf{A} \leftarrow \sum_l \mathbf{A}^{(l)}$
  $\mathbf{b} \leftarrow \sum_l \mathbf{b}^{(l)}$
  $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$
  return $\mathbf{w}$
end procedure

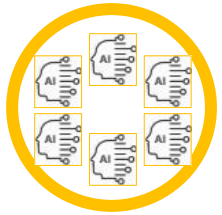


MIP Federated Learning

- Dozens of federated algorithms equivalent to centralized versions

  Linear regression
  Logistic regression
  k-means clustering
  Naïve Bayes classifier
  Principal component analysis
  Decision trees (ID3, CART)
  Descriptive statistics
  Histograms
  T-tests
  ANOVA
  Kaplan-Meier estimator
  Pearson correlation coefficients
  Calibration belt estimator

- Which ML algorithms have an equivalent federated version?
- In what context a simpler approximate model is adequate?

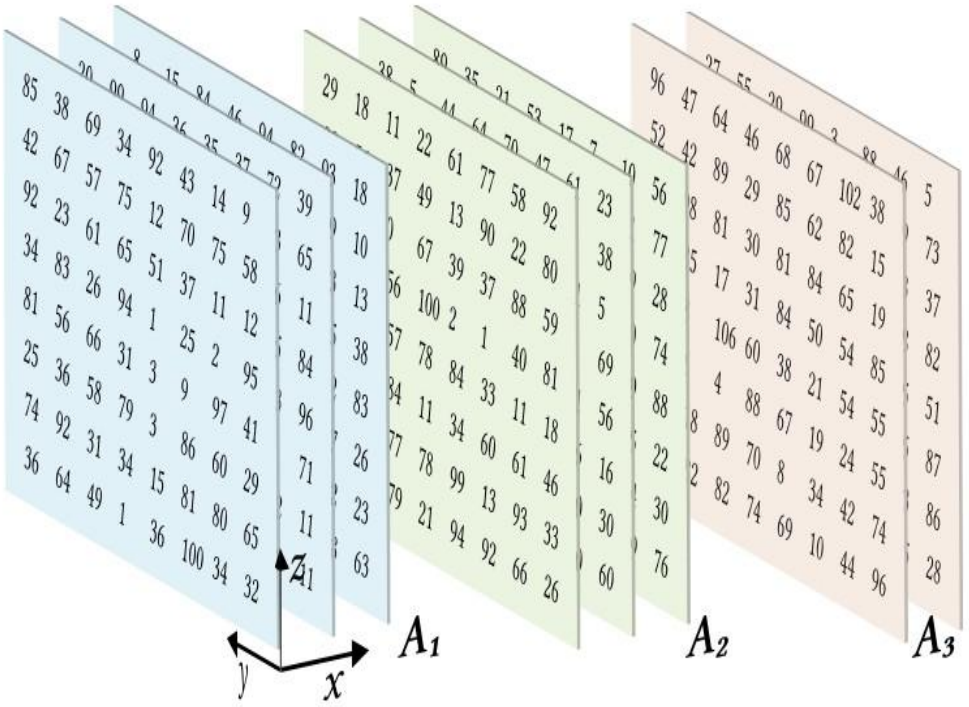


- Taking advantage of some datasets being on same cloud?

Mathematical Foundations: Typed Tensor Algebra / Language

Fundamental tensor types based on privacy properties:

- **Federated Tensors ($\mathcal{A}_F$):** Inherently private data (e.g., patient records) restricted to ensure no raw record-level data is communicated.
  - **Record Dimension (RD):** Identifies unique axis of privacy that *distinguishes individual records*.
- **Shared Tensors ($\mathcal{A}_S$):** Shared information (e.g., model parameters, aggregated statistics) that contains no private data and can be freely communicated.
- **Global Tensors ($\mathcal{A}_G$):** A virtual, conceptual concatenation of all client data along the record dimension; used for theoretical proofs but never stored.



Courtesy of Theofilos Mailis (Athena RC)

Federated Tensor Algebra (FeTA): A New Programming Paradigm for Federated Analytics and Learning

1. **A realization of the Typed Tensor Language**
2. **New FL programming paradigm:** Users write centralized tensor programs as user-defined functions, the algebra automatically derives encoders/merge/decoders
3. **Decouples algorithmic intent from internal system components:** Analogous to logical and physical separation in traditional DBMSes - Room for dataflow optimization
4. **Code evaluation via two-layer enforcement:** Accepts only code that secures the core algebraic properties of the language and enforces data exposure disciplines
5. **Pluggable controller-executor architecture:** Targets different federated backends without rewriting user code
6. **Code-complexity evaluation:** Shorter, more maintainable than SOTA FL frameworks

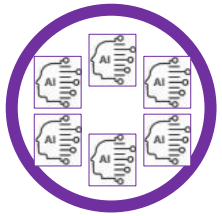
Code evaluation with Static Analysis

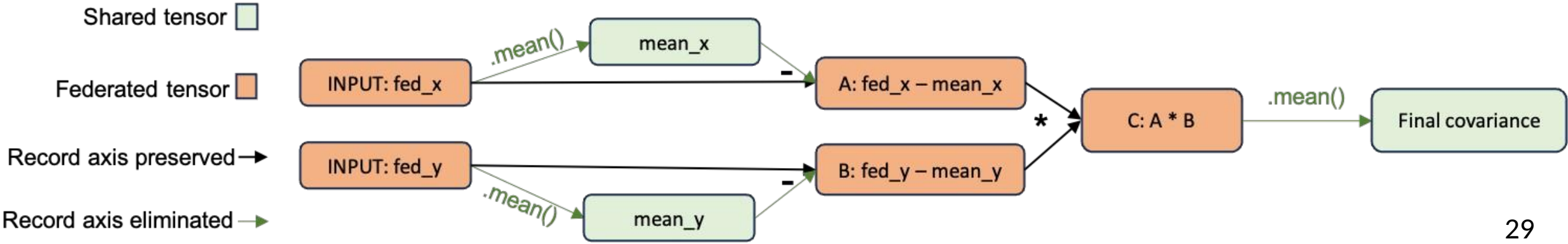
```
def cov(fed_x: FederatedTensor, fed_y: FederatedTensor):
    """Covariance between two federated arrays."""
    mean_x = fed_x.mean()
    mean_y = fed_y.mean()
    return ((fed_x - mean_x) * (fed_y - mean_y)).mean()
```

Shared Tensors

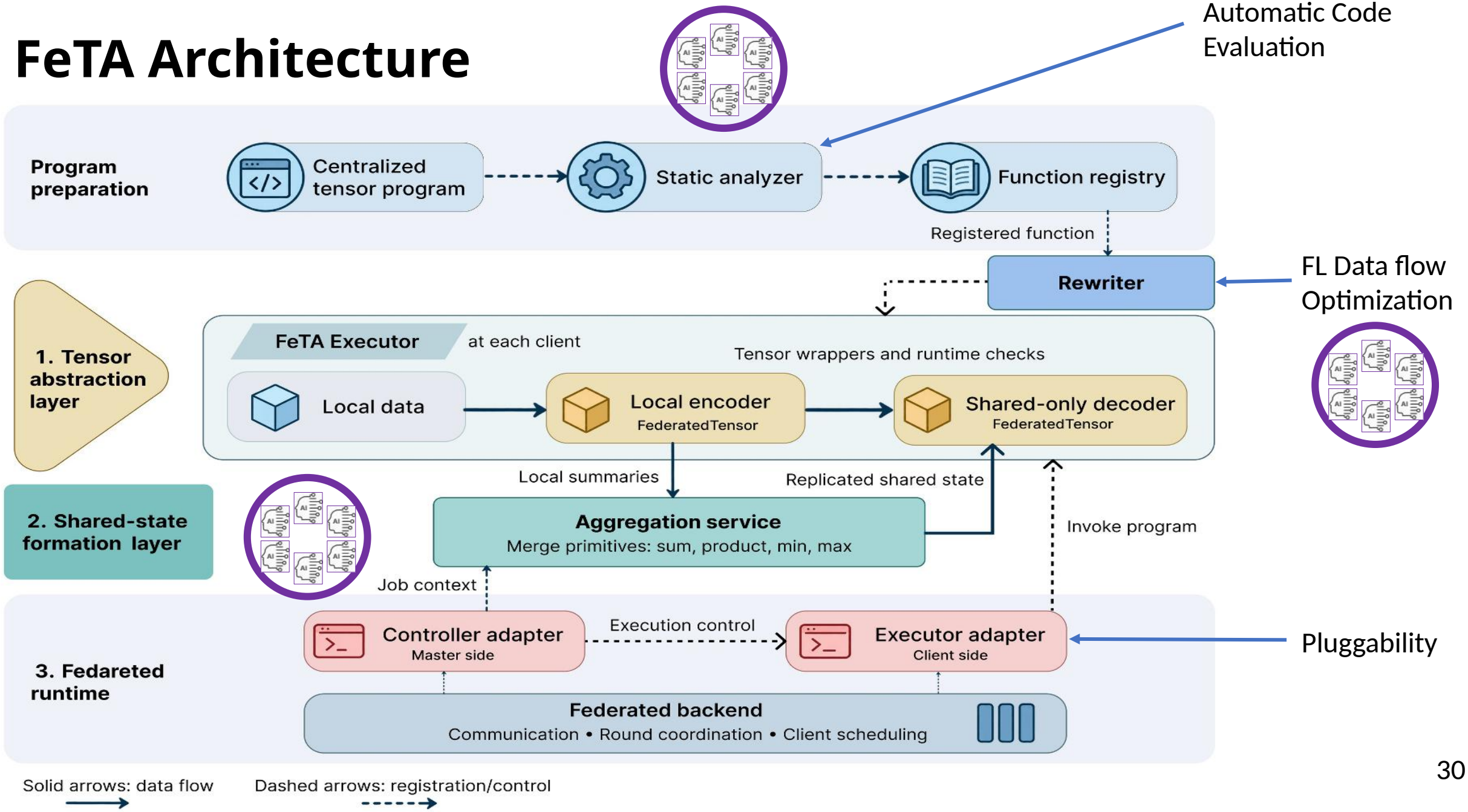
Record Axis Elimination

Data flow analysis

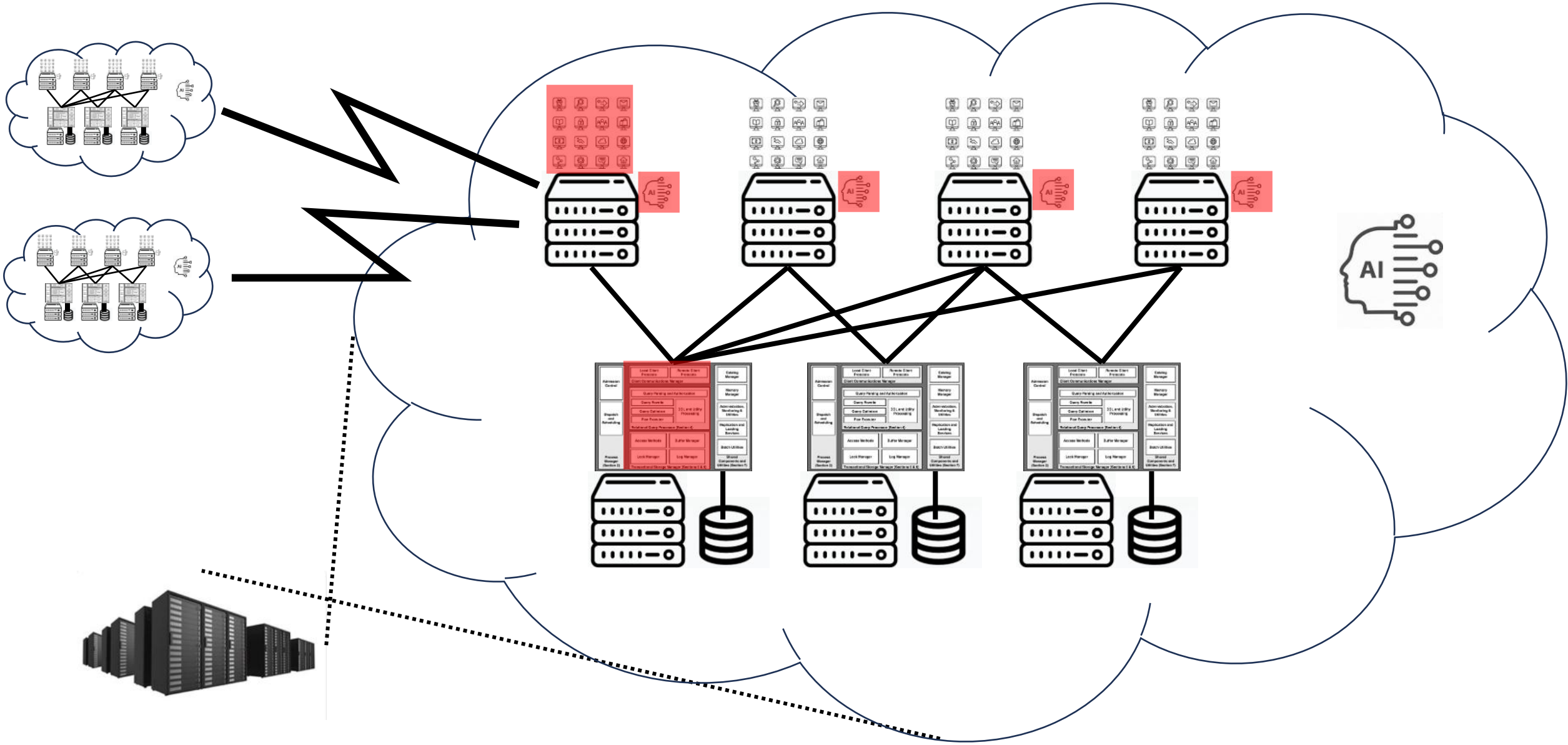




FeTA Architecture



(CL-B) New Architectures for User Defined (AI) Functions



Production query in OpenAIRE (snippet abridged)

- Mine full text of publications in OpenAIRE to extract links to projects from various funders

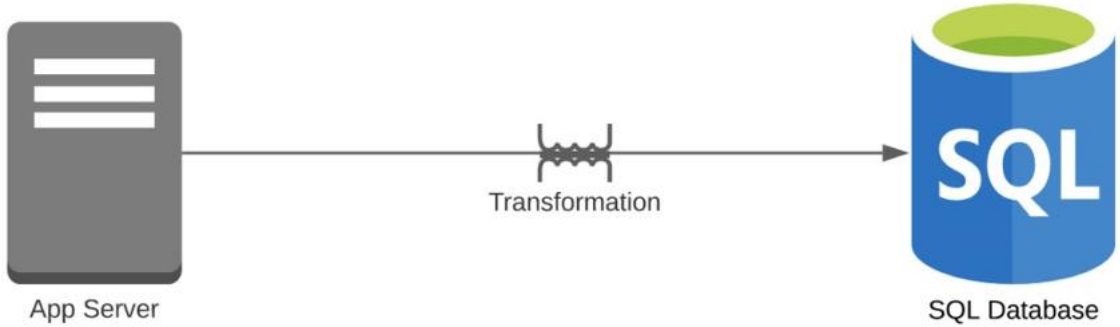
```
select docid, id, fundingclass1, grantid,
    jdict('documentId', docid, 'projectId', id, 'confidenceLevel',
    sqroot(min(1.49,confidence)/1.5), 'textsnippet', context) as C1, context
  from (
       select docid,id,confidence, docid, id,  fundingclass1, grantid, context
       from (
          select 0.8 as confidence, docid, id, fundingclass1, grantid, context from (select docid, regexpr("(\d+)",middle) as middle,
               comprspaces ( j2s(prev1,prev2,prev3,prev4,prev5,prev6,prev7,prev8,prev9,prev10,prev11,prev12,prev13,"<<<",middle,">>>",next)) as context
          from (
              setschema 'docid,prev1,prev2,prev3,prev4,prev5,prev6,prev7,prev8,prev9,prev10,prev11,prev12,prev13,middle,next'
                  select c1, textwindow(lower(c2),-13,0,1, '\b\d{5,6}\b') from pubs where c2 is not null
           )
           where CAST(regexpr("(\d+)",middle) AS INT)>70000
       ), grants
       where fundingclass1="AKA" and  (regexprmatches("[\b\s]academy of finland[\b\s]", context) or  regexprmatches("[\b\s]finnish (?:(?:programme for
          )?cent(?:re|er)s? of excellence|national research council|funding agency|research program)[\b\s]", context)
          or regexprmatches("[\b\s]finnish academy[\b\s]", context))
          and grantid=middle
  )
  group by docid,id
);dueio5k[5ktfmcl65eqey wb
```

The complete query can be found here:
https://github.com/openaire/iis/blob/34895ac6d9eec71050537224100d944576a38e23/iis-wf/iis-wf-referenceextraction/src/main/resources/eu/dnetlib/iis/wf/referenceextraction/project/main_sqlite/oozie_app/lib/scripts/projects.sql



Challenges: Performance

- Impedance mismatch between declarative (SQL) and procedural (e.g., Python) operation



- Frequent context switches
- Data conversions and copies
- Excessive function calls
- Inefficient compilation
- Limited query optimization
- Long UDF pipelines
- Materialization of intermediate results (some engines)

The YeSQL approach

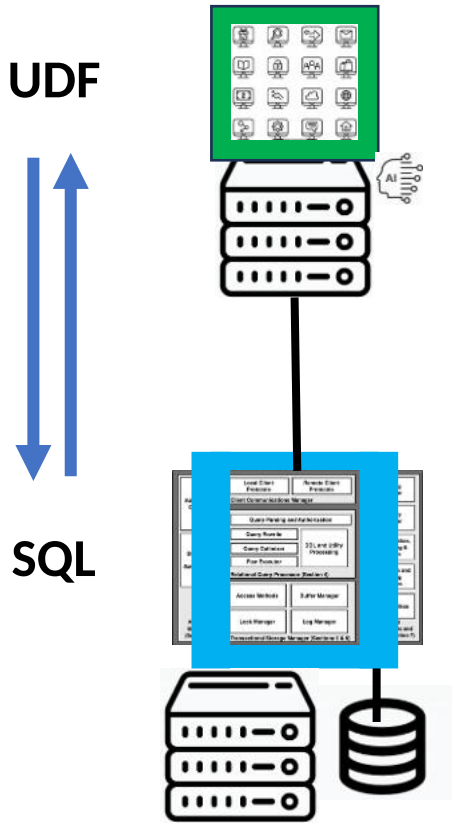
YeSQL name

You extend SQL

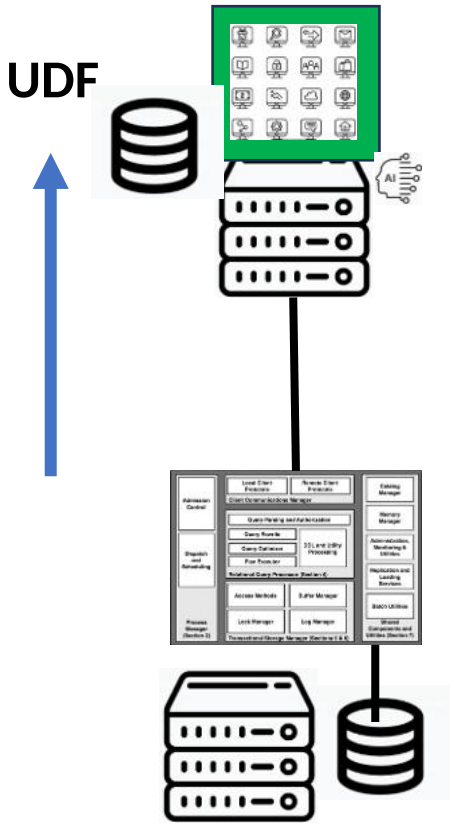
Enough with NoSQL!

Pronounced ‘yes’ ‘q’ ‘l’

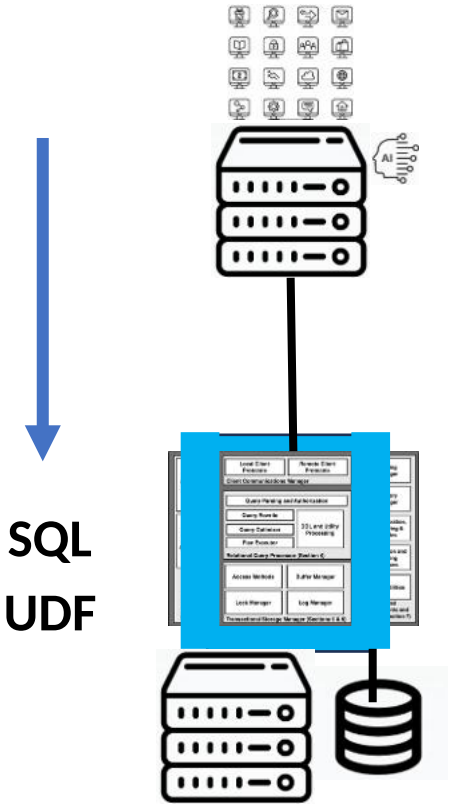
Architectural Alternatives for UDF Processing



Impedence Mismatch



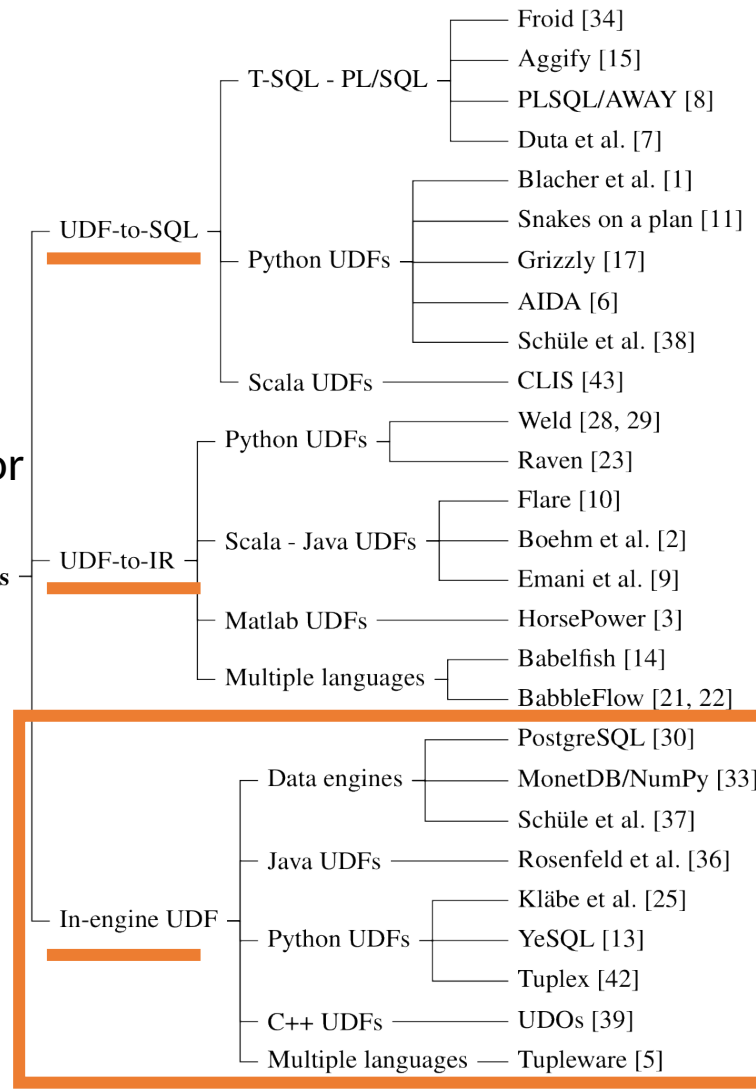
NoSQL: Conventional App Wisdom



YeSQL: Superior Performance

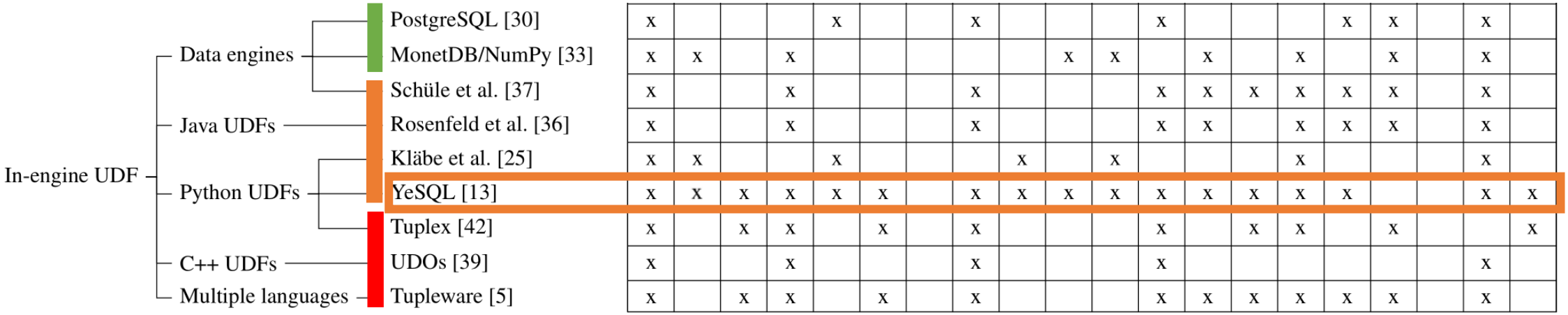
A taxonomy of solutions

- UDF optimization
  - Parallelization
  - Vectorization
  - Function inlining
  - In/Out-process
  - Tracing/method JIT
- Execution model
  - Tuple/Vector/Operator
  - Column/Row based
- Query optimization
  - Reordering
  - Fusion
  - Rules, Cost
- Usability / Expressivity
  - Engine/Lib specific
  - Static/Dynamic



UDF optimization							Execution model					Qry optimization				Usability & expr.			
parallelization	vectorization	func inlining	in-process	out-process	tracing JIT	method JIT	tuple-at-a-time	vector-at-a-time	oper-at-a-time	column-based	row-based	reordering	fusion	rules	cost model	engine spec	lib spec	static	dynamic
x			x				x				x	x	x	x	x			x	
x			x				x				x	x	x	x	x			x	
x			x				x					x	x	x	x	x		x	
x			x				x						x		x			x	
		x					x	x	x	x	x	x		x	x		x		x
		x					x	x	x	x	x	x		x	x		x		x
		x					x	x	x	x	x	x		x	x		x	x	
x		x							x	x				x					
	x	x				x	x				x				x	x	x	x	
x			x										x						
x	x		x						x	x			x			x	x	x	
x			x				x				x	x	x	x	x	x	x	x	
	x	x				x	x			x					x	x	x	x	
	x	x						x		x			x	x	x	x	x	x	
		x					x				x				x	x	x	x	
x		x	x						x	x			x	x		x	x		
x			x			x	x						x	x		x			
x		x	x				x		x		x	x	x	x	x	x		x	
x				x			x				x				x	x		x	
x			x				x					x		x		x			
x			x				x				x	x		x		x		x	
x			x				x				x	x		x		x		x	
x	x			x				x		x				x				x	
x		x	x	x	x		x	x	x	x	x	x	x	x	x			x	x
x		x	x		x		x				x		x	x		x			x
x			x				x				x							x	
x		x	x		x		x				x	x	x	x	x	x		x	

A taxonomy of solutions



PostgreSQL [30]	x				x			x				x				x	x		x	
MonetDB/NumPy [33]	x	x		x						x	x		x		x		x		x	
Schüle et al. [37]	x			x				x				x	x	x	x	x	x		x	
Rosenfeld et al. [36]	x			x				x				x	x		x	x	x		x	
Kläbe et al. [25]	x	x			x				x		x				x				x	
YeSQL [13]	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x			x	x
Tuplex [42]	x		x	x		x		x				x		x	x		x			x
UDOs [39]	x			x				x				x							x	
Tupleware [5]	x		x	x		x		x				x	x	x	x	x	x		x	

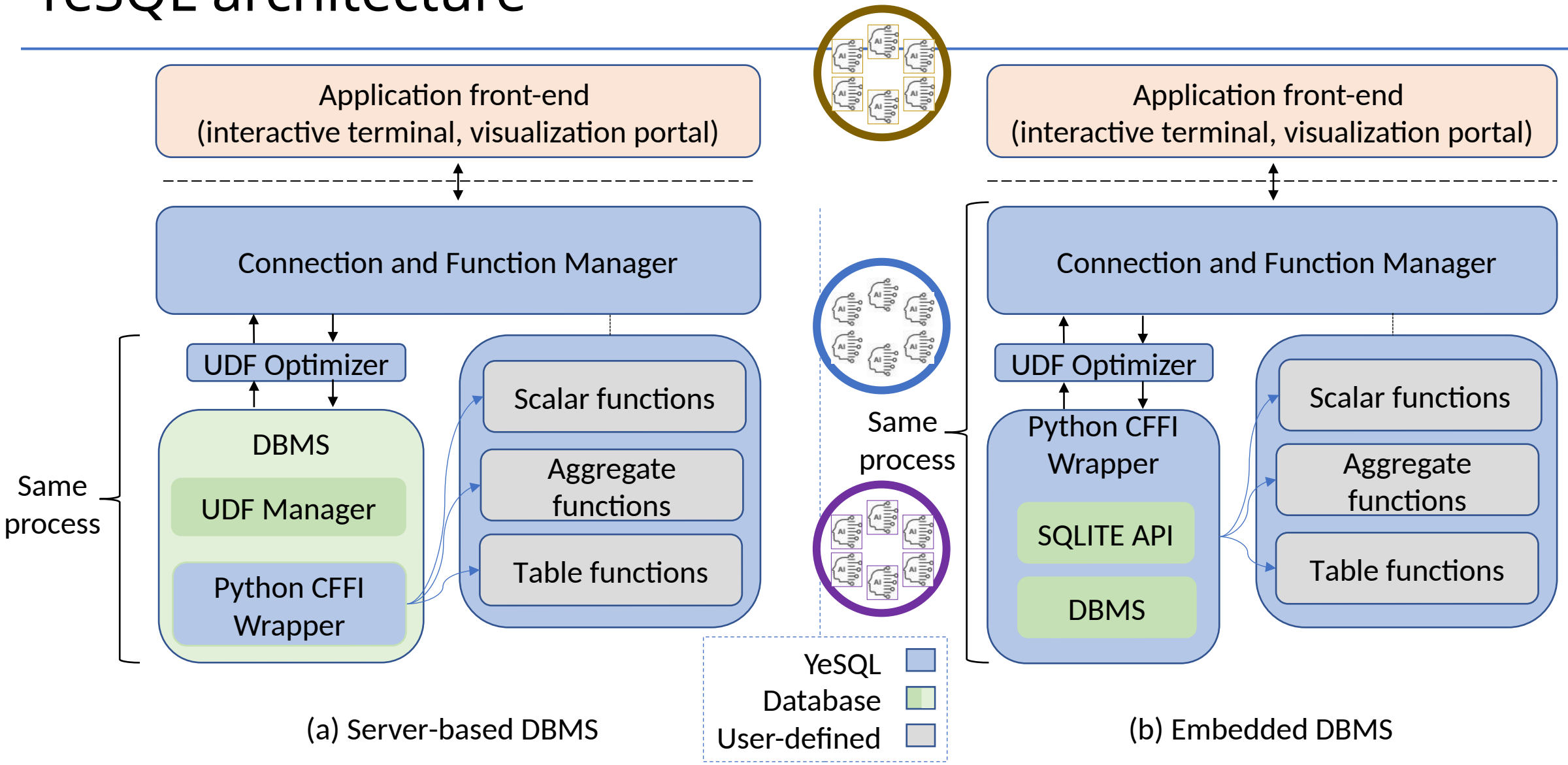
[Y. Foufoulas, A. Simitsis – tutorials @IEEE ICDE'23, @PVLDB'23]

Standard SQL interpretation (non-JIT)
[interpreted plan w/ UDFs]

JIT-compiled UDFs
[interpreted plan w/ JIT-cmp'd UDFs]

JIT-compiled queries
[JIT-cmp'd plan w/ UDFs]

YeSQL architecture



YeSQL characteristics

- SQL extension to more usable, more expressive, and more performant Python UDFs
- Expressiveness
  - Stateful UDFs
  - Dynamically typed UDFs
  - Scalar, aggregate, and table UDFs
- Performance
  - JIT-compilation (with seamless DBMS/UDF exchange)
  - Parallelization
  - Statefulness
  - Fusion for optimization
- Support for both server-based and embedded data engines
- Usability
  - Parametric polymorphic UDFs
  - Functional syntax for UDFs

Performance: UDF and relational operator fusion

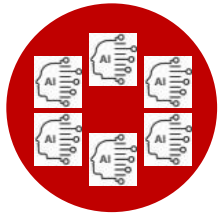
(1) udf → udf ⇨ fused_udf

(2) udf → rel ⇨ fused_udf or rel → udf ⇨ fused_udf

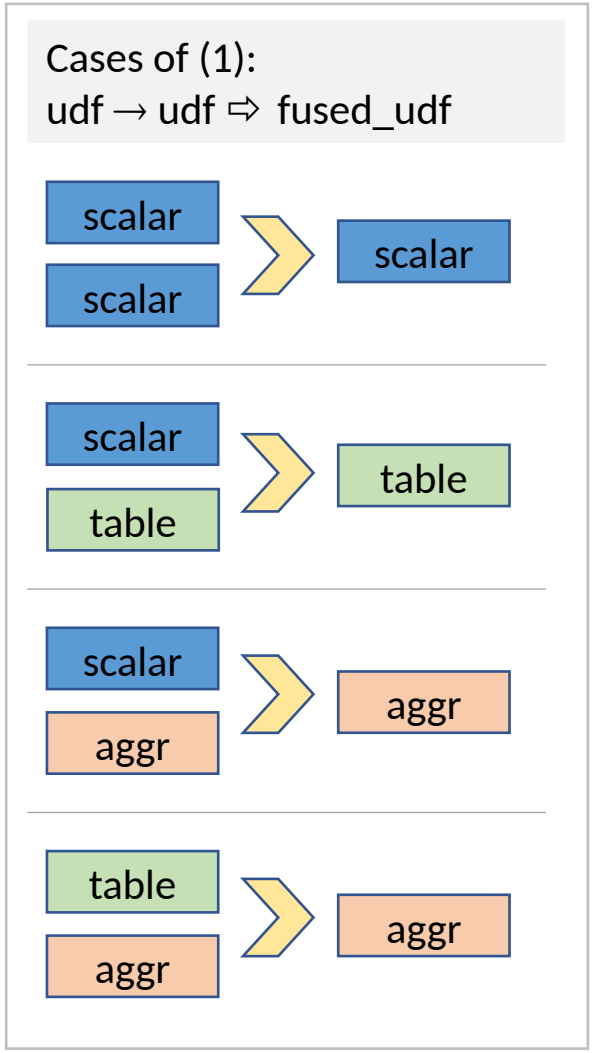
- Remove optimization barriers caused by black-box treatment of UDFs
- Result of fused_rel also possible [Blacher et al.@CIDR22]

(3) udf → rel → udf ↬ rel → udf → udf ⇨ rel → fused_udf

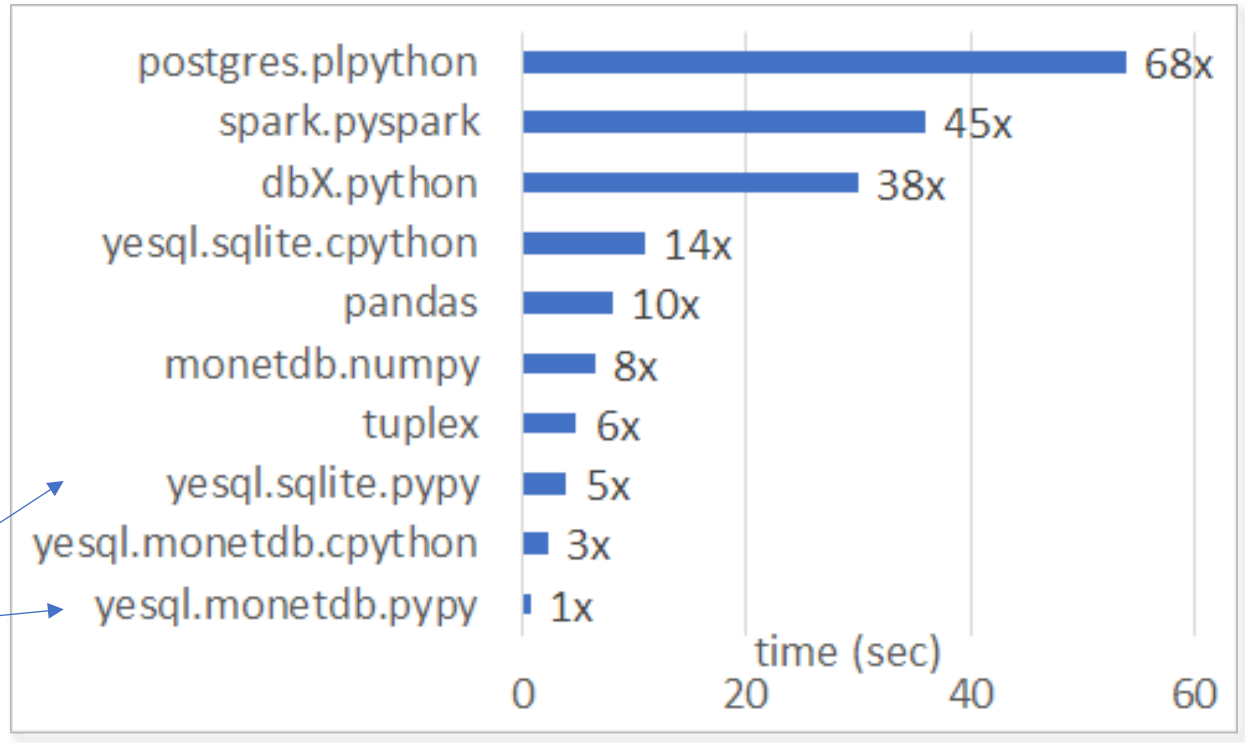
- Additional fusion and optimization opportunities



[→ , ⇨ , ↬ : data pipeline, operator fusion, and operator reordering, respectively

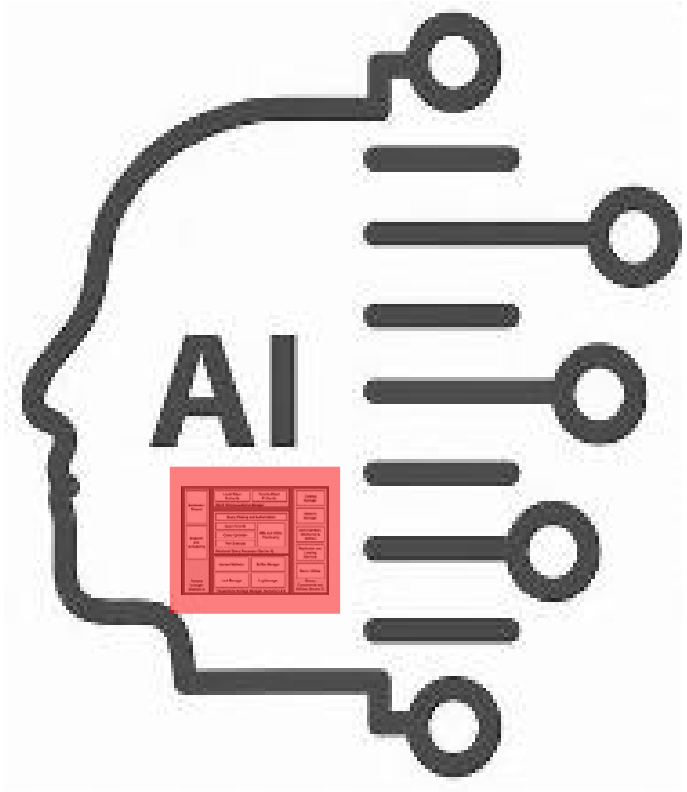


Evaluation: YeSQL vs. Data engines



JIT-compiled YeSQL

execution time [less is better]



Technological Advances – Reversing the Impact

YeSQL for Heterogeneous Federated Learning

- In synergy with Federated Tensor Algebra
- Introduces a new UDF type: Federated Learning Iterative User Defined Aggregate Function
  - Integration of stateful federated learning/AI pipelines into relational query plans
  - Portability on heterogeneous remote databases through YeSQL's pluggable architecture
  - Pluggability on different federated backends (e.g., NVIDIA FLARE) through FeTA's adapter interfaces
  - Enables automatic recovery of long running learning workflows

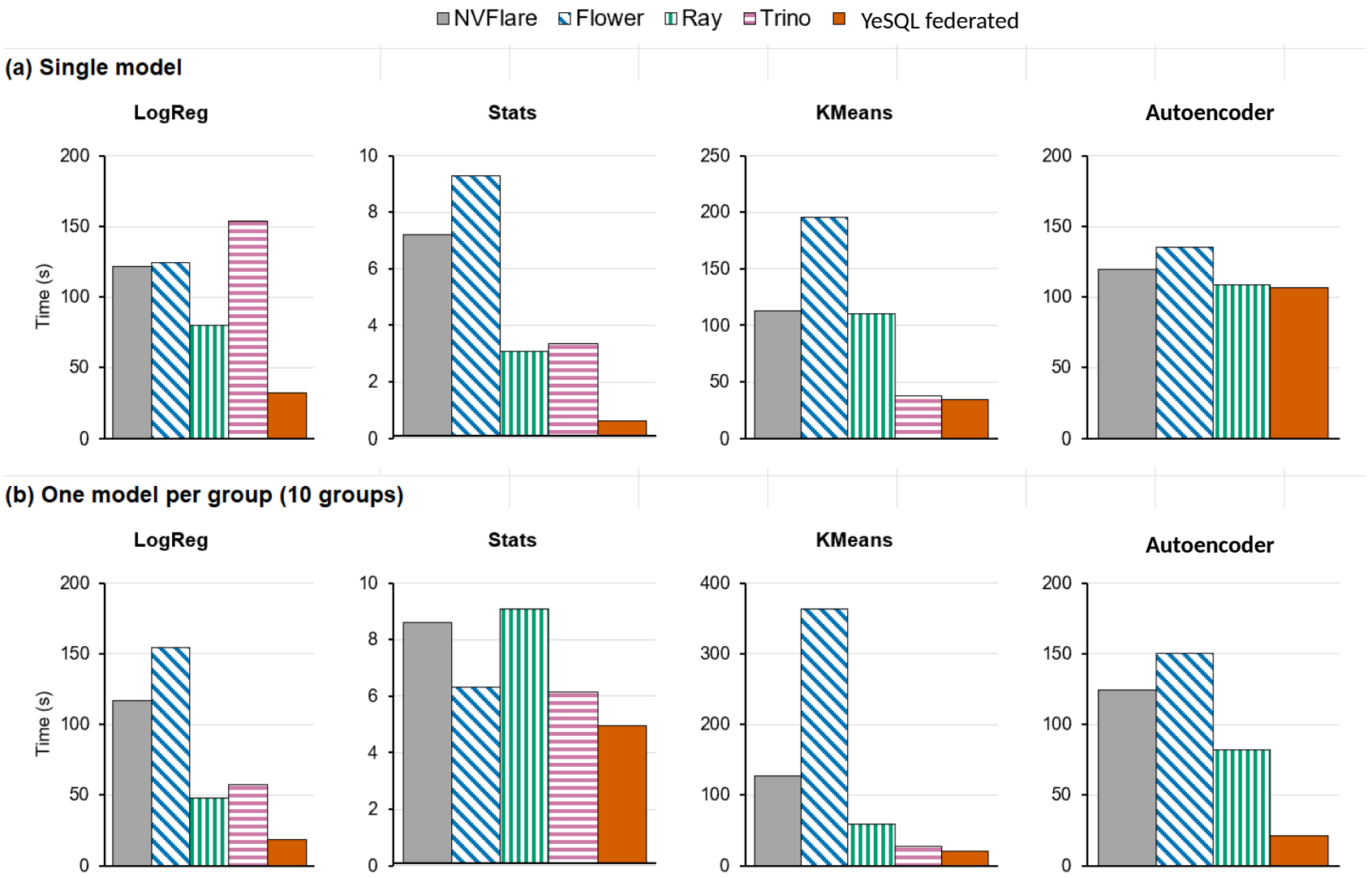
- Federated Learning Optimization and Scheduling
  - Faster FL via JIT compilation and/or GPU acceleration
  - Scheduling on heterogeneous hardware
    - Not only for performance but for tradeoff between performance and energy efficiency
  - Query Optimization and Fusion for FL pipelines

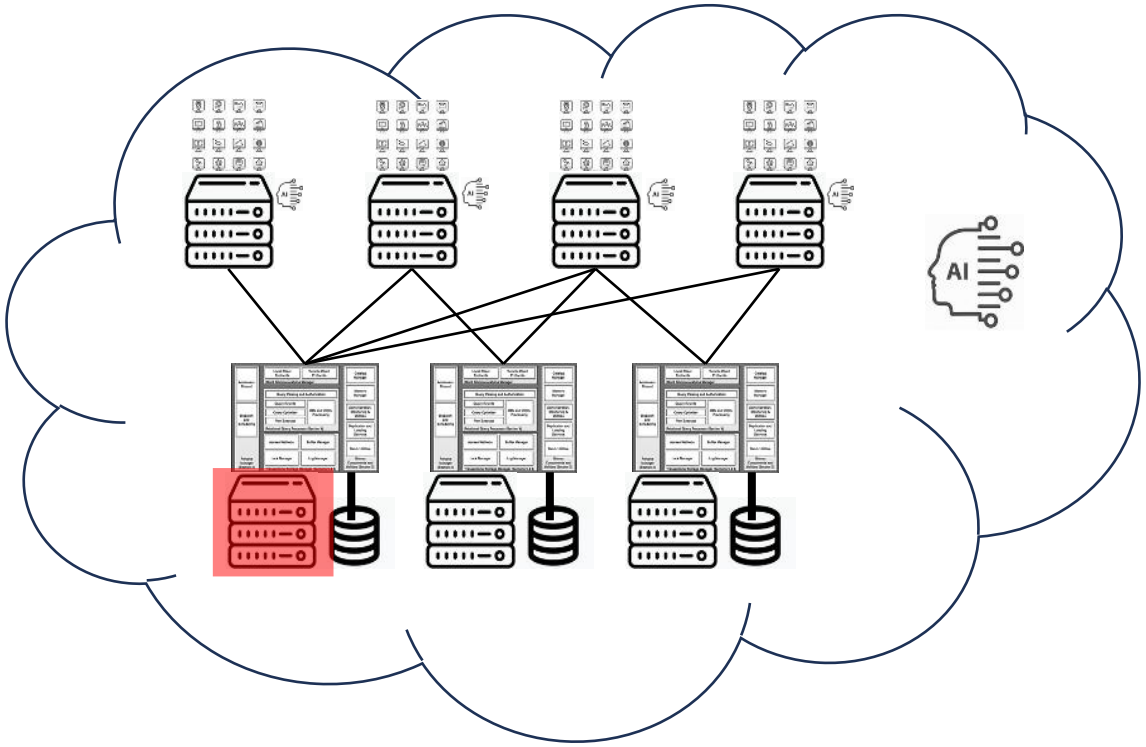
YeSQL for Heterogeneous Federated Learning

… and promising results:

- Outperforms alternatives in multiple workflows
- Gains are larger when FL requires integration with a relational operator (GROUP BY)

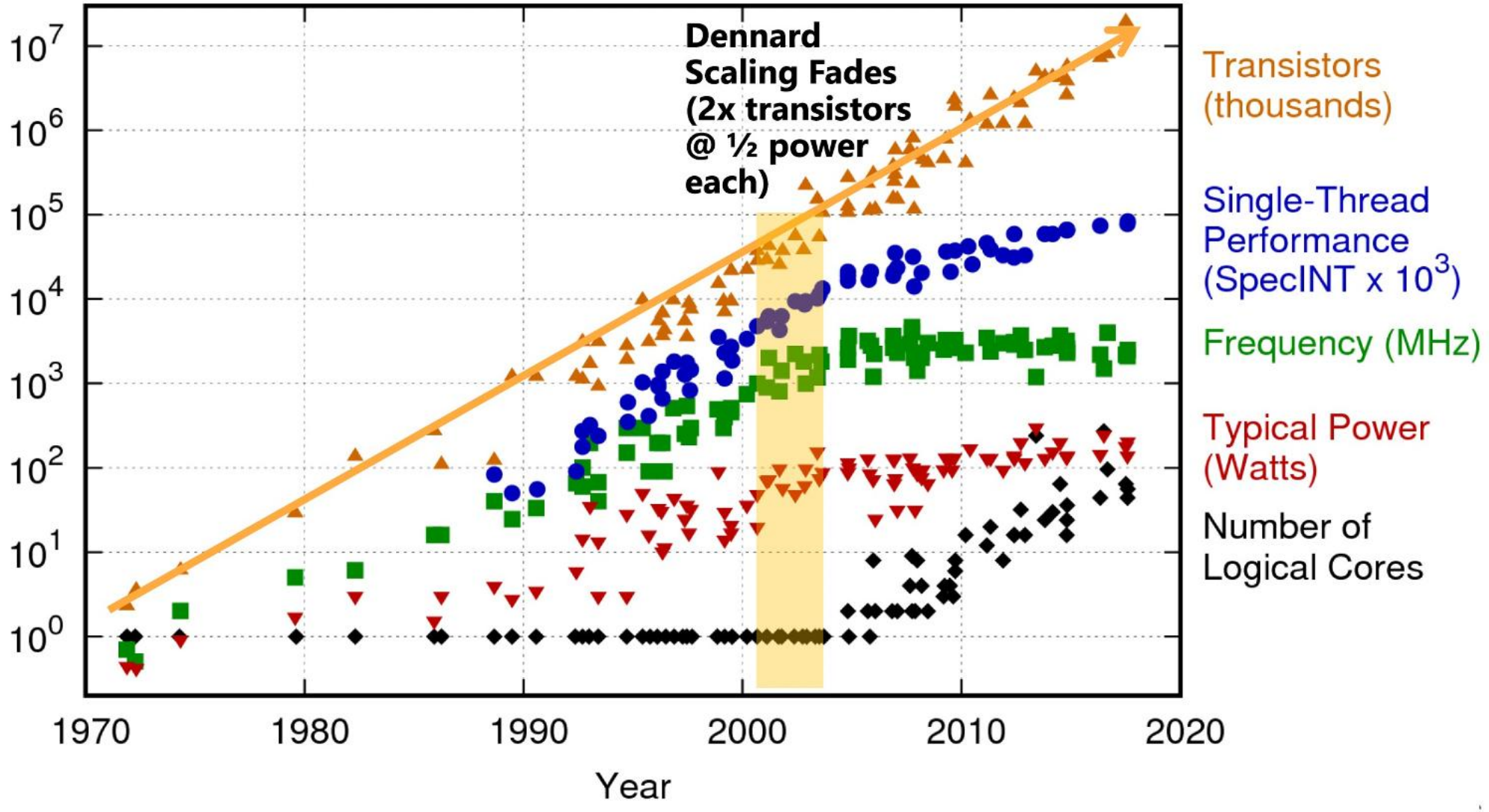
Results indicate clear need to integrate custom federated ML/AI to DBMS architectures





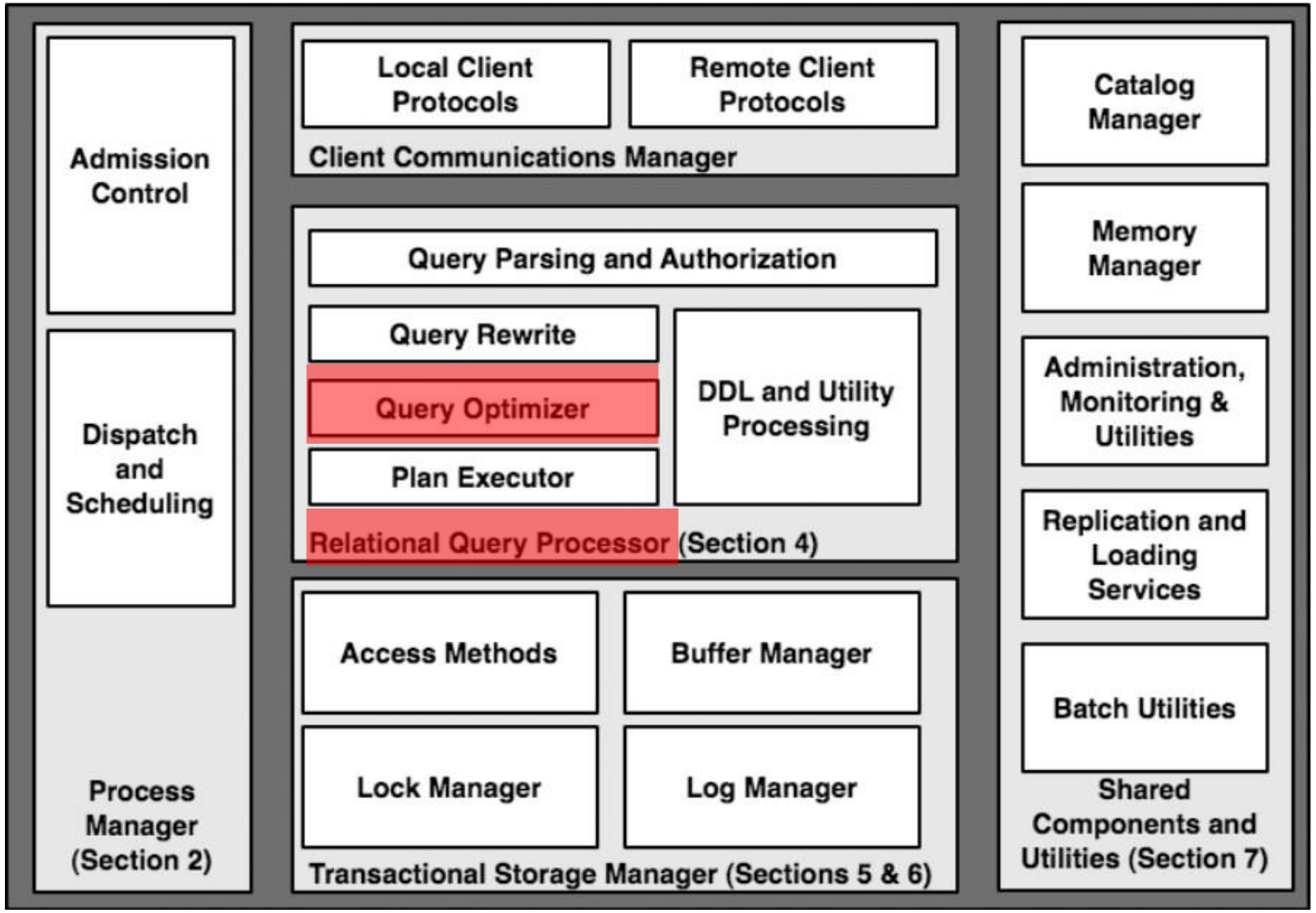
Technological Advances - Hardware

Microprocessor Trends



Courtesy of Mark Hill
(Microsoft Azure and U. Wisconsin)

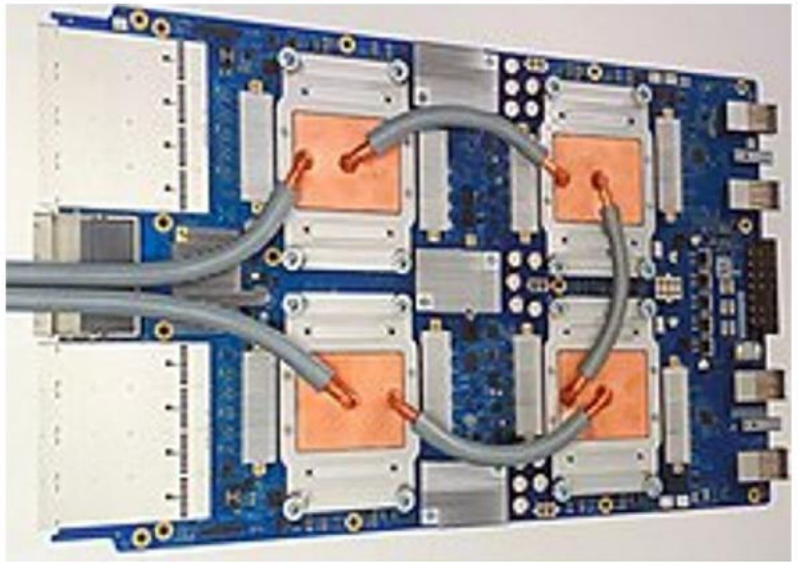
(HW-A) Hardware Accelerators for Query Processing / Optimization



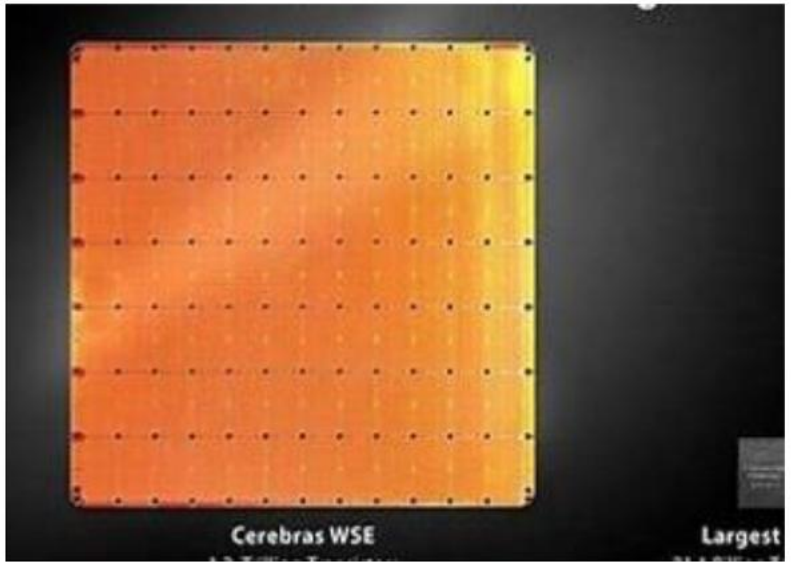
Foundations and Trends in Databases (2007)
© Hellerstein, Stonebraker, Hamilton

Hardware Accelerators

- Hardware component executing targeted computation class faster
  - Usually with (much) less energy
- Primary motivation: Deep Neural Network Machine Learning



Google Tensor Processing Unit



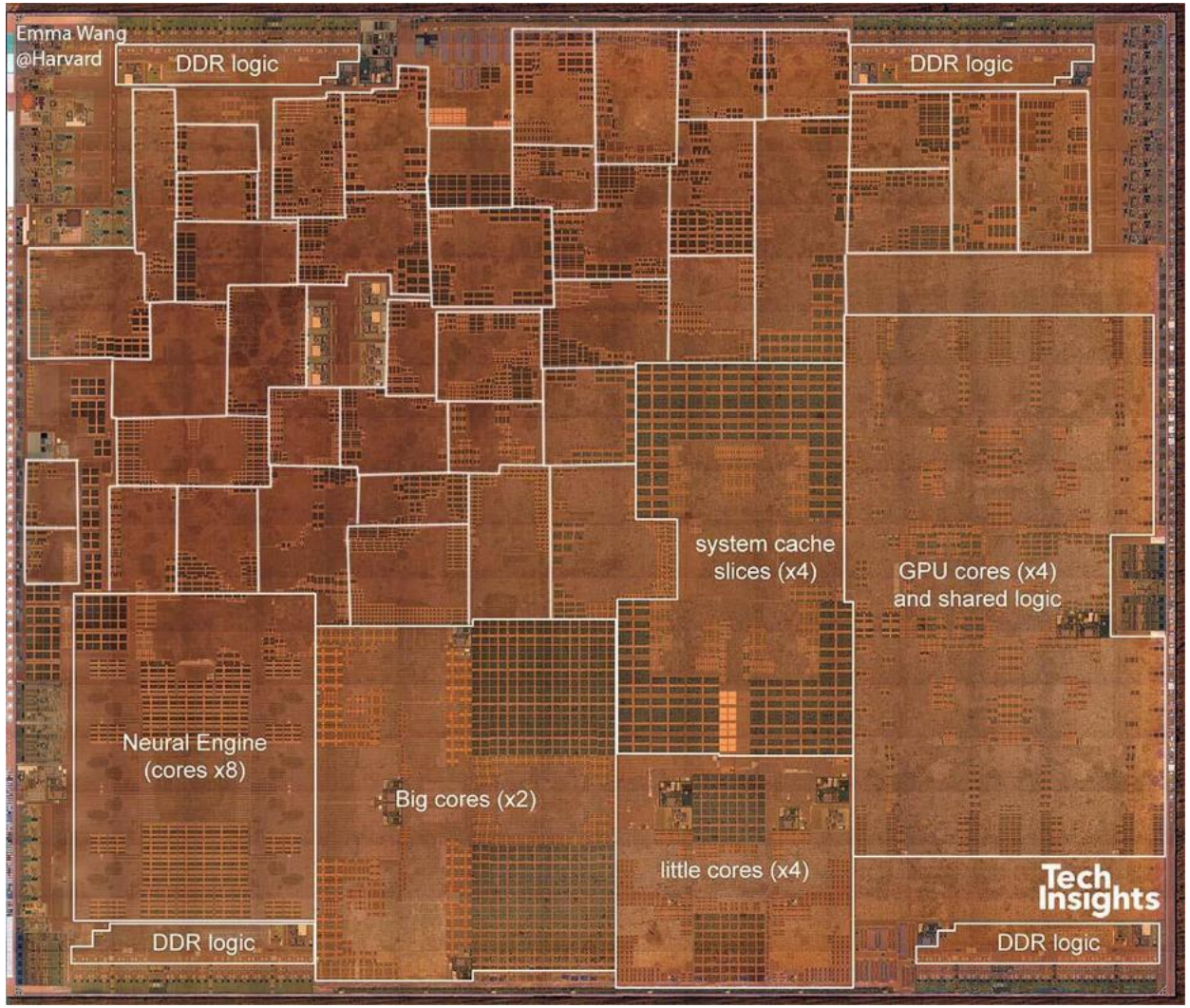
Cerebras Wafer Scale Engine



Nvidia Grace-Hopper

Courtesy of Mark Hill
(Microsoft Azure and U. Wisconsin)

Accelerator-Level Parallelism



2019 Apple A12: 42 Accelerators

- Deploy many
- Use concurrently

- CPU: Control plane
- Accelerators: Data plane

- Query optimization ~~nightmare~~ paradise

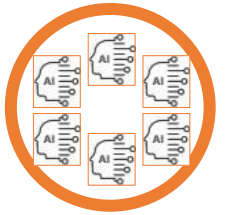
Courtesy of Mark Hill
(Microsoft Azure and U. Wisconsin)

Accerator-Driven Challenges/Opportunities for Database Systems

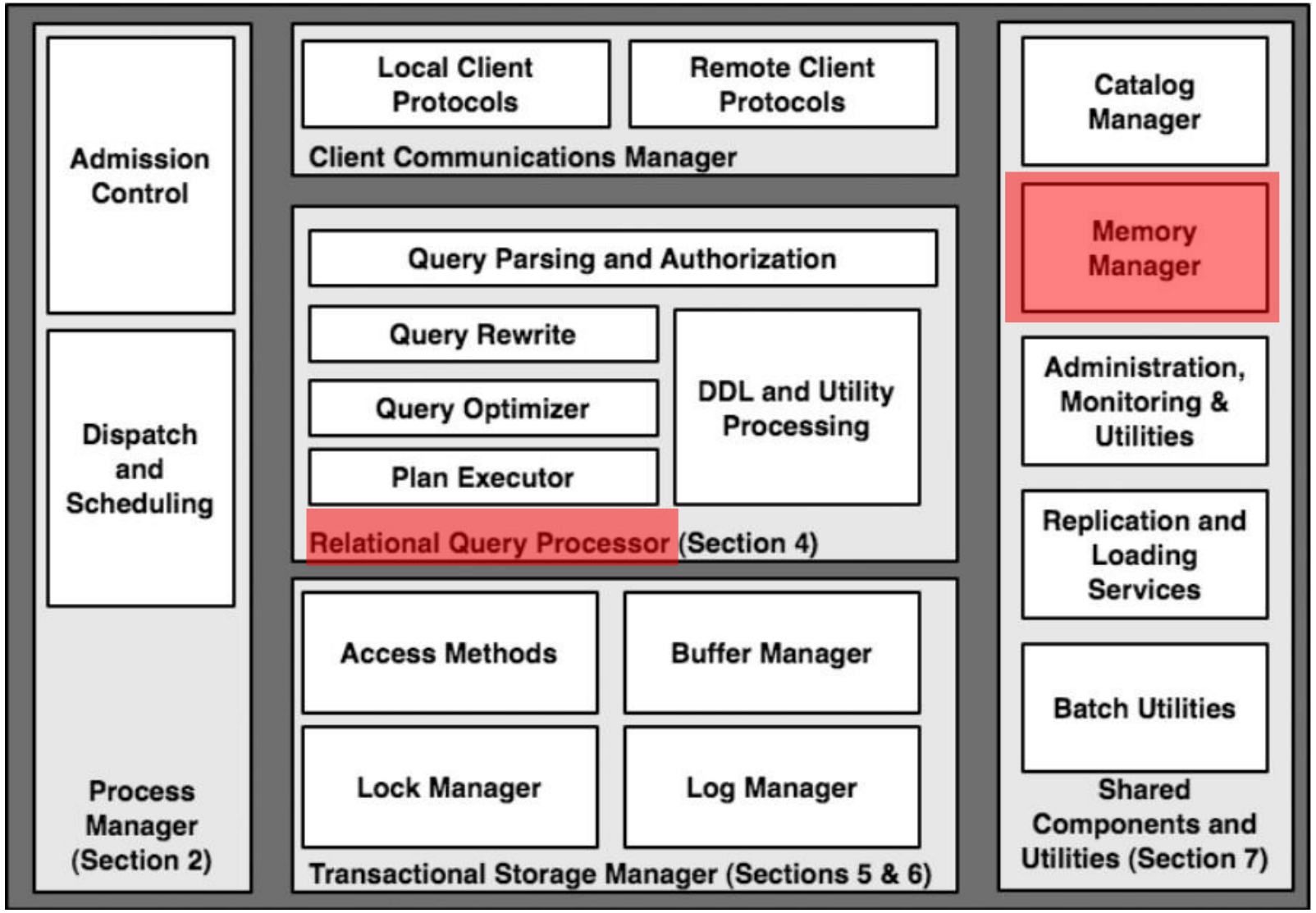
- Unaccelerated → Accelerated query execution
- Query optimization for accelerated execution

- Most promising data operations for acceleration?
  - (de)compression, (de)encryption, joins, …
- Most promising accelerators?
  - CPU SIMD/Matrix, GPU, FPGA, TPU, special-built for DB ops, …



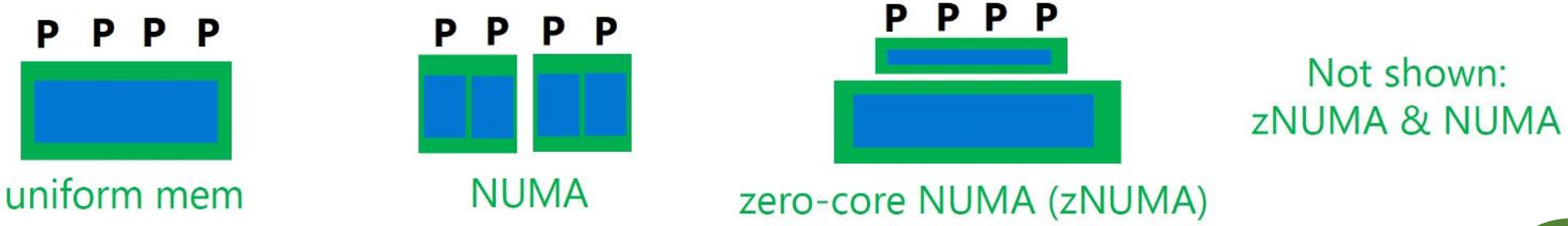


(HW-B) New Memory for Buffer Management and Query Processing

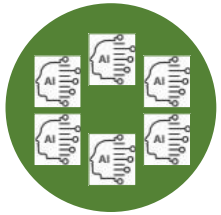


Foundations and Trends in Databases (2007)
© Hellerstein, Stonebraker, Hamilton

New Memory: Two-Tier Memory

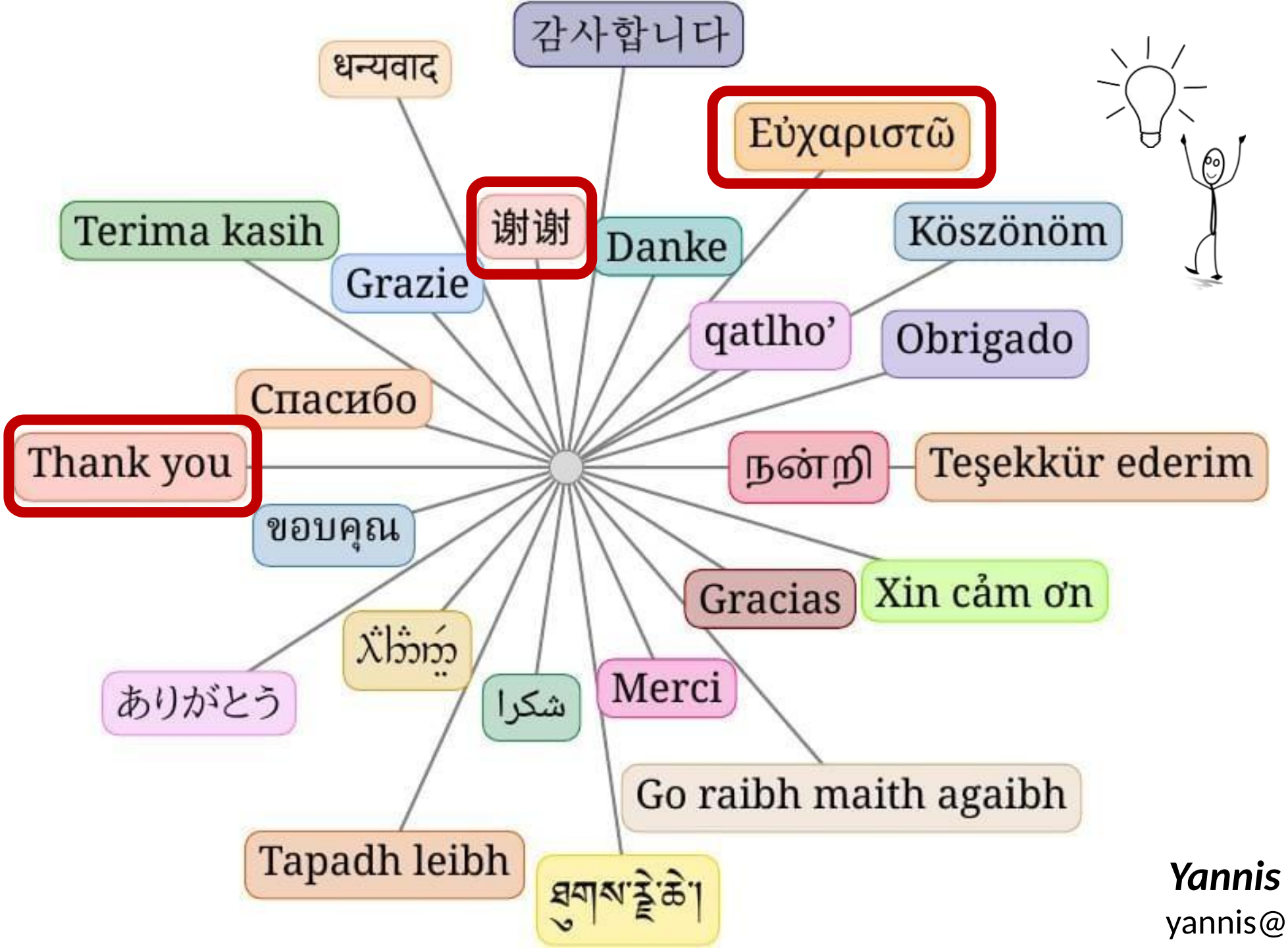


- Benefits for data applications, with new buffer pool management
- Balance point between DRAM and SSD



DRAM	<<	Capacity	<<	SSD
	<	Latency	<<	
	<	Energy/Bit	<<	

Courtesy of Mark Hill
(Microsoft Azure and U. Wisconsin)



Yannis Ioannidis
yannis@athenarc.gr