DO WE STILL NEED PEOPLE TO WRITE DATABASE SYSTEMS?

MARCH 2022

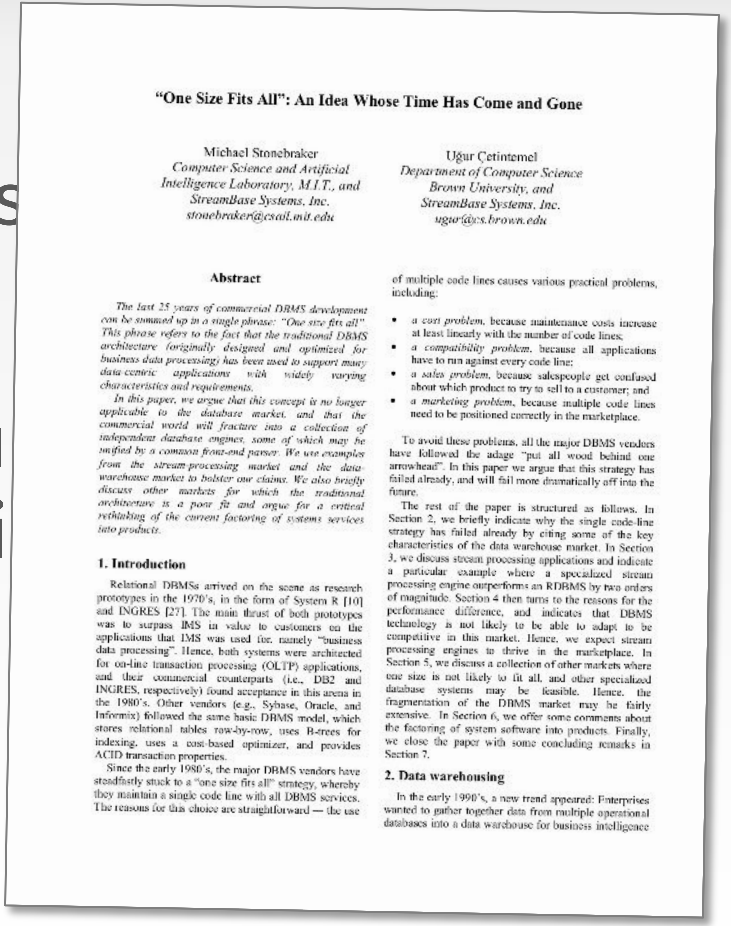
Carnegie Mellon University

OTTER TUNE

@andy_pavlo

Specialized DBMSs for analytics
around since the 1970s.

The OLAP DBMS landscape flou
2000s because more organizati
data sets than ever before.



"One Size Fits All": An Idea Whose Time Has Come and Gone

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Abstract

The last 25 years of commercial DBMS development can be summed up in a single phrase: "One size fits all". This phrase refers to the fact that the traditional DBMS architecture (originally designed and optimized for business data processing) has been used to support many data-centric applications with widely varying characteristics and requirements.

In this paper, we argue that this concept is no longer applicable to the database market, and that the commercial world will fracture into a collection of independent database engines, some of which may be unified by a common front-end parser. We use examples from the stream-processing market and the data-warehouse market to bolster our claims. We also briefly discuss other markets for which the traditional architecture is a poor fit and argue for a critical rethinking of the current factoring of systems services into products.

1. Introduction

Relational DBMSs arrived on the scene as research prototypes in the 1970's, in the form of System R [10] and INGRES [27]. The main thrust of both prototypes was to surpass IMS in value to customers on the applications that IMS was used for, namely "business data processing". Hence, both systems were architected for on-line transaction processing (OLTP) applications, and their commercial counterparts (i.e., DB2 and INGRES, respectively) found acceptance in this arena in the 1980's. Other vendors (e.g., Sybase, Oracle, and Informix) followed the same basic DBMS model, which stores relational tables row-by-row, uses B-trees for indexing, uses a cost-based optimizer, and provides ACID transaction properties.

Since the early 1980's, the major DBMS vendors have steadfastly stuck to a "one size fits all" strategy, whereby they maintain a single code line with all DBMS services. The reasons for this choice are straightforward — the use of multiple code lines causes various practical problems, including:

- a *cost problem*, because maintenance costs increase at least linearly with the number of code lines;
- a *compatibility problem*, because all applications have to run against every code line;
- a *sales problem*, because salespeople get confused about which product to try to sell to a customer; and
- a *marketing problem*, because multiple code lines need to be positioned correctly in the marketplace.

To avoid these problems, all the major DBMS vendors have followed the adage "put all wood behind one arrowhead". In this paper we argue that this strategy has failed already, and will fail more dramatically off into the future.

The rest of the paper is structured as follows. In Section 2, we briefly indicate why the single code-line strategy has failed already by citing some of the key characteristics of the data warehouse market. In Section 3, we discuss stream processing applications and indicate a particular example where a specialized stream processing engine outperforms an RDBMS by two orders of magnitude. Section 4 then turns to the reasons for the performance difference, and indicates that DBMS technology is not likely to be able to adapt to be competitive in this market. Hence, we expect stream processing engines to thrive in the marketplace. In Section 5, we discuss a collection of other markets where one size is not likely to fit all, and other specialized database systems may be feasible. Hence, the fragmentation of the DBMS market may be fairly extensive. In Section 6, we offer some comments about the factoring of system software into products. Finally, we close the paper with some concluding remarks in Section 7.

2. Data warehousing

In the early 1990's, a new trend appeared: Enterprises wanted to gather together data from multiple operational databases into a data warehouse for business intelligence

Columnar Data Storage ➡ **5 Years**
– *C-Store (VLDB 2005)*

Vectorized Query Execution ➡ **10 Years**
– *MonetDB/X100 (CIDR 2005)*

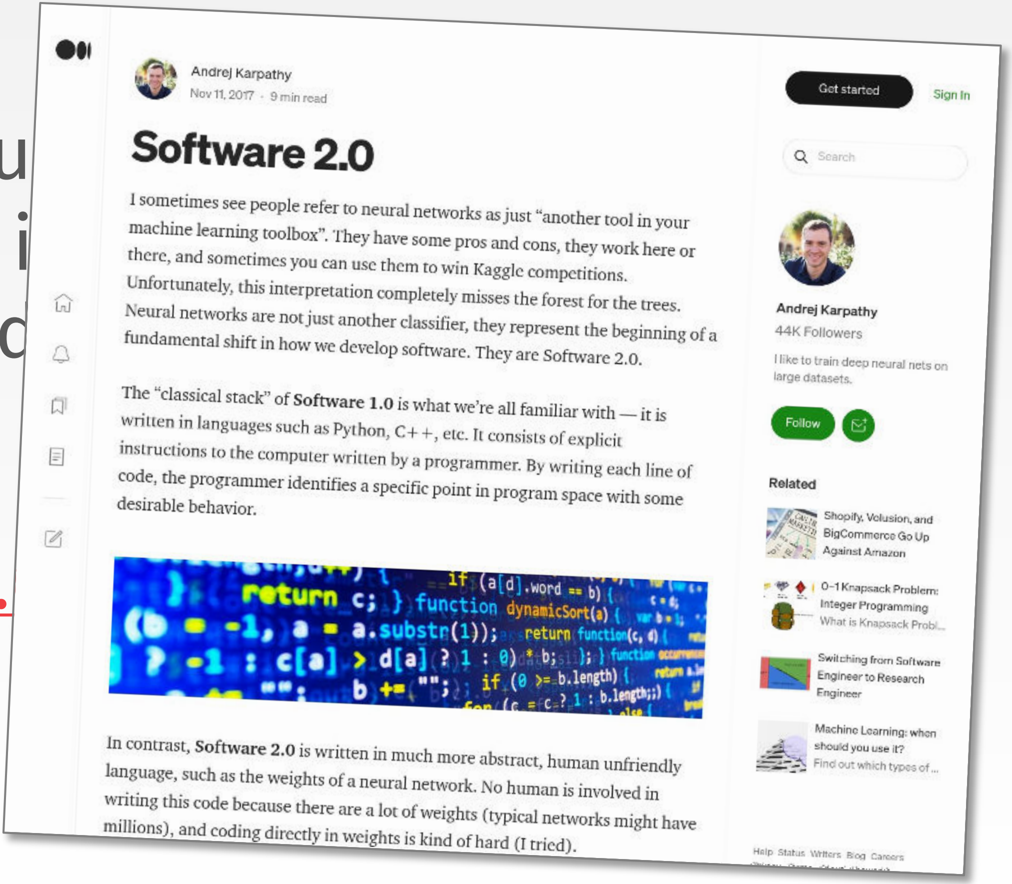
Query Codegen / JIT Compilation ➡ **8 Years**
– *HIQUE (ICDE 2010)*

Writing a DBMS is hard...

Can we instead use *machine learning*?

A learned component u
previous observations i
future behavior instead
devised strategy.

Example of Software 2.


Andrej Karpathy
Nov 11, 2017 · 9 min read

Get started
Sign In

Search

Software 2.0

I sometimes see people refer to neural networks as just "another tool in your machine learning toolbox". They have some pros and cons, they work here or there, and sometimes you can use them to win Kaggle competitions. Unfortunately, this interpretation completely misses the forest for the trees. Neural networks are not just another classifier, they represent the beginning of a fundamental shift in how we develop software. They are Software 2.0.

The "classical stack" of **Software 1.0** is what we're all familiar with — it is written in languages such as Python, C++, etc. It consists of explicit instructions to the computer written by a programmer. By writing each line of code, the programmer identifies a specific point in program space with some desirable behavior.

In contrast, **Software 2.0** is written in much more abstract, human unfriendly language, such as the weights of a neural network. No human is involved in writing this code because there are a lot of weights (typical networks might have millions), and coding directly in weights is kind of hard (I tried).

Andrej Karpathy
44K Followers
I like to train deep neural nets on large datasets.
Follow

Related
Shopify, Volusion, and BigCommerce Go Up Against Amazon
0-1 Knapsack Problem: Integer Programming
What is Knapsack Probl...
Switching from Software Engineer to Research Engineer
Machine Learning: when should you use it?
Find out which types of ...

Help Status Writers Blog Careers

Execution

- *Indexes*
- *Filters*
- *Sorting Algorithms*
- *Hashing Algorithms*
- *Scheduling*

Query Planning

- *Cardinality Estimation*
- *Cost Models*
- *Join Ordering Search*
- *SQL Rewriting*
- *Predicate Inference*

Data Storage

- *Compression*
- *Sampling*
- *Caching*

Traditional Index

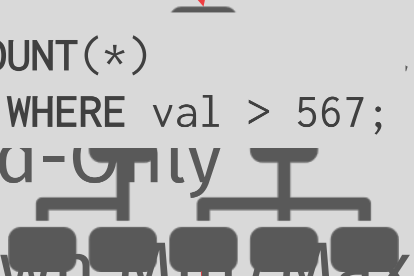
GET(val>567)

Assumptions:

```
SELECT COUNT(*)
  FROM X WHERE val > 567;
```

Read-Only

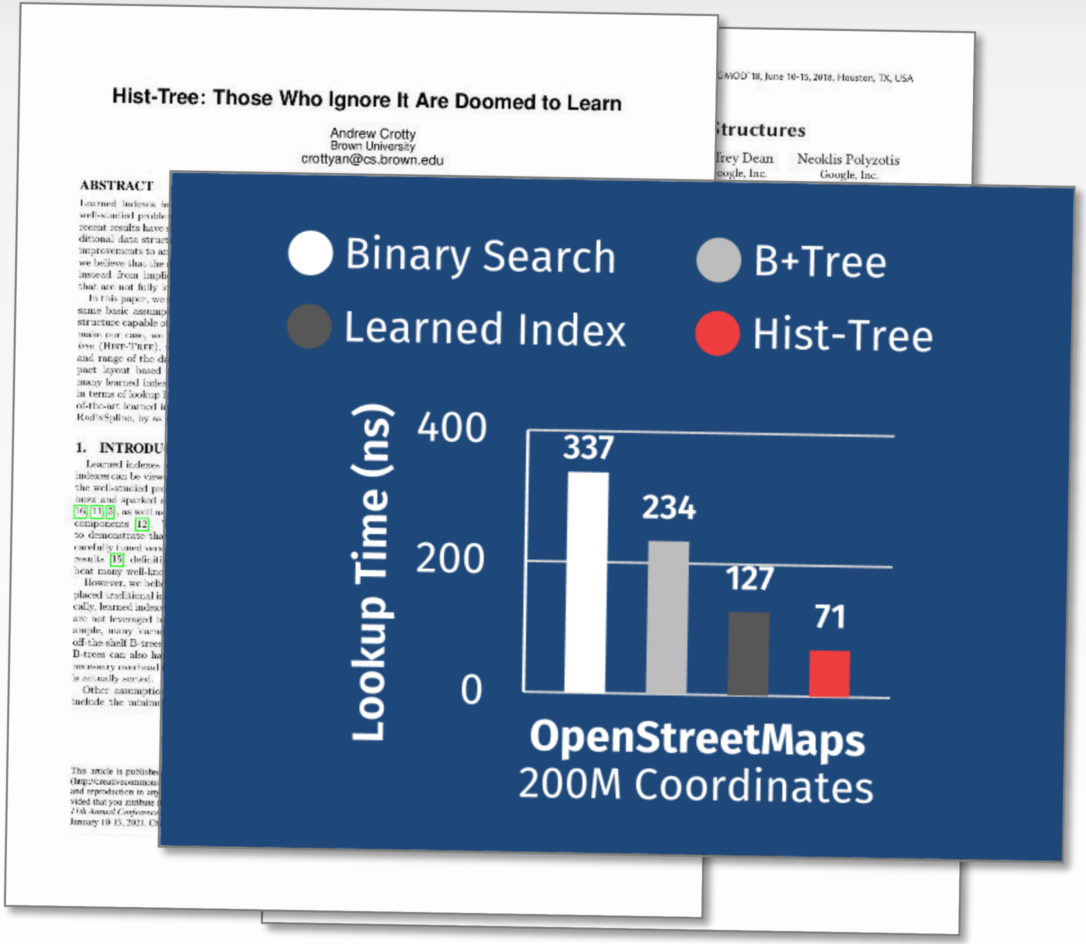
Known Min/Max



559	561	563	564	565	567	568	569

➡ *Sorted Data*

➡



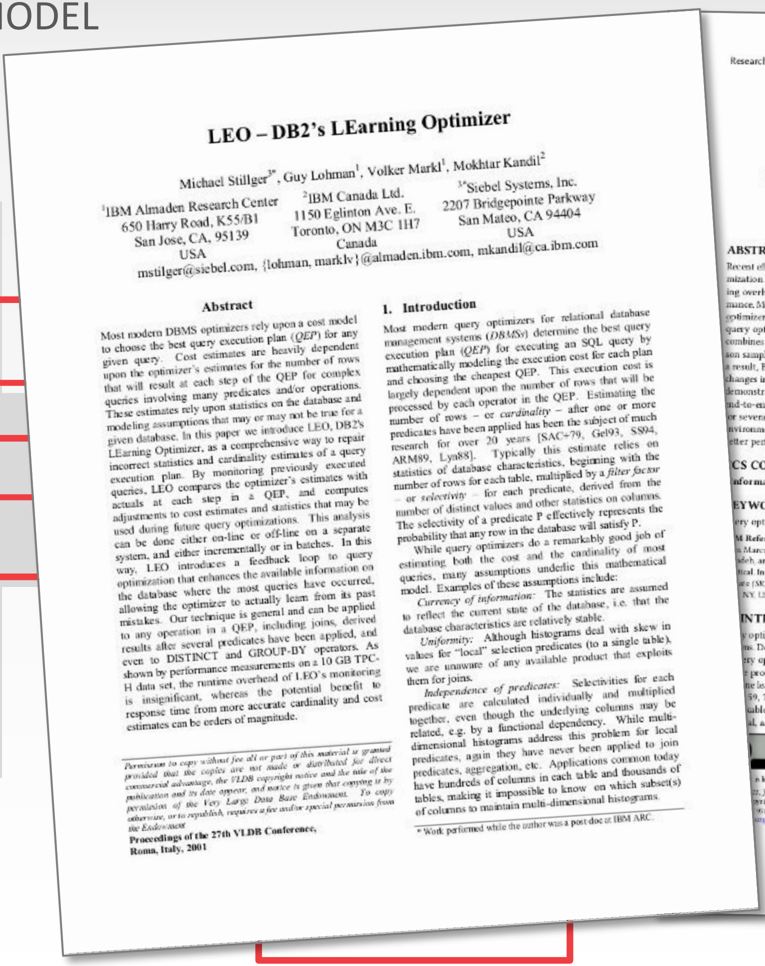
Source: Andrew Crotty

Query Optimizer

Nested Loop Join

```
SELECT *
  FROM X JOIN Y
    ON X.id = Y.id;
```

Alternative Query Plans



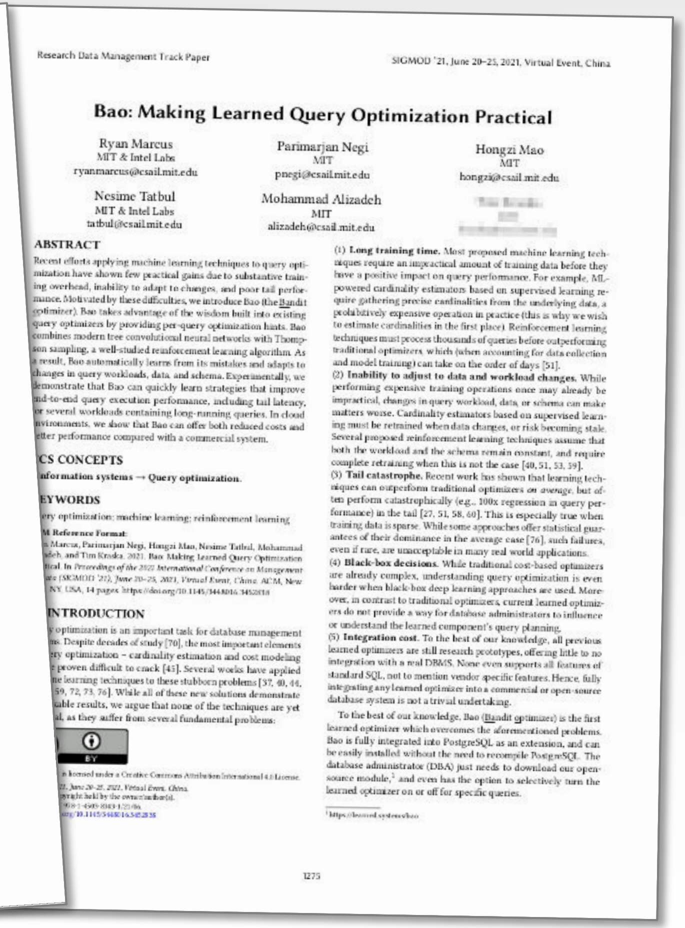
LEO – DB2's LEarning Optimizer

Michael Stillger[3*], Guy Lohman[1], Volker Markl[1], Mokhtar Kandil[2]
[1]IBM Almaden Research Center, 650 Harry Road, K55/B1, San Jose, CA, 95139, USA
[2]IBM Canada Ltd., 1150 Eglinton Ave. E., Toronto, ON M3C 1H7, Canada
[3*]Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404, USA
mstilger@siebel.com, {lohman, marklv}@almaden.ibm.com, mkandil@ca.ibm.com

Abstract

1. Introduction

Proceedings of the 27th VLDB Conference, Roma, Italy, 2001



Research Data Management Track Paper
SIGMOD '21, June 20–25, 2021, Virtual Event, China

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

ABSTRACT

INTRODUCTION

Source: Ryan Marcus

Failsafe Mechanisms

– *What do you do when models are horribly wrong?*

Explainability

– *How to tell humans why DBMS made certain choices?*

Human Feedback / Overrides

– *Can a human provide hints? What if they're wrong?*

Transferability

– *Can we reuse knowledge gained from one database and apply it to another?*

Can we instead use *machine learning*?

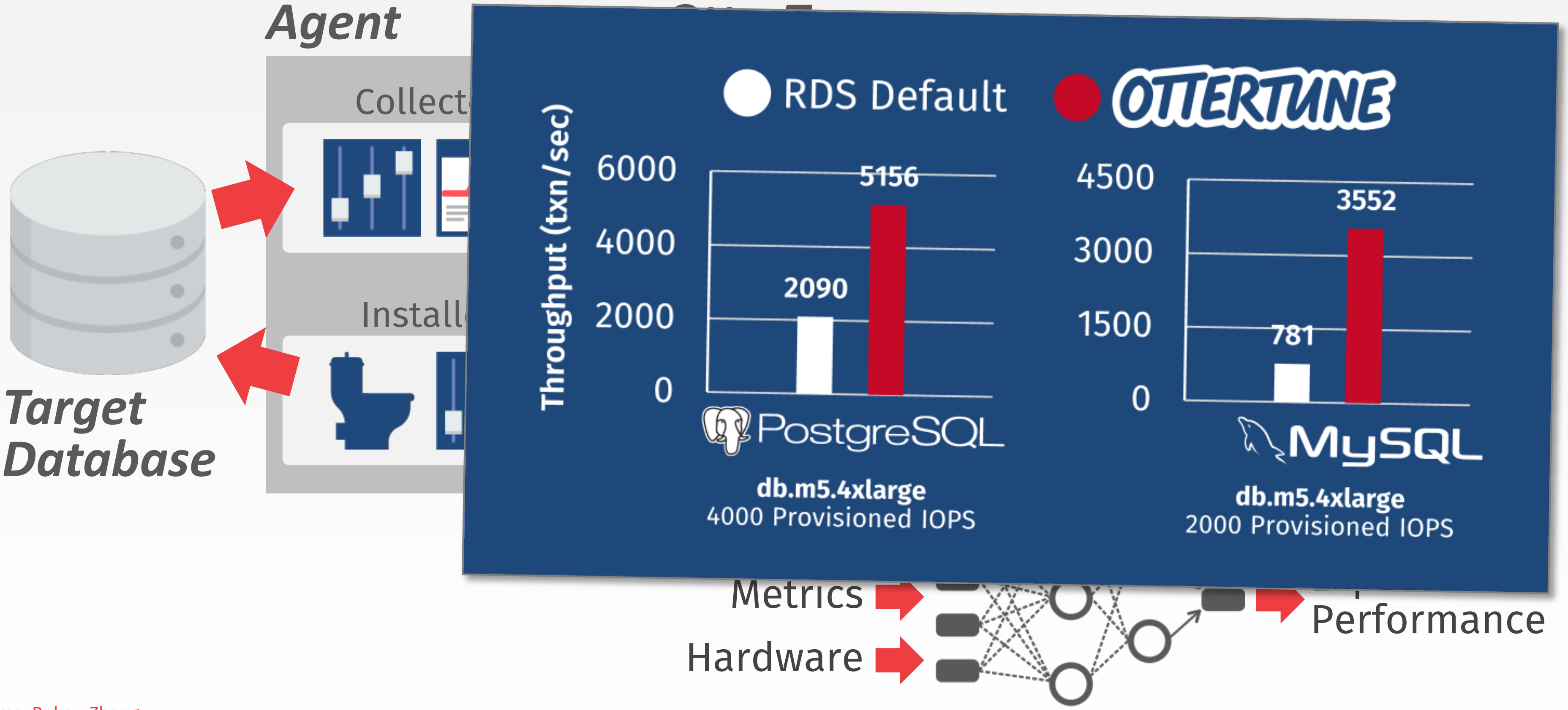
Yes, but...

There are already ML-powered tools to optimize database instances.

- *Leverage existing APIs to extract telemetry and apply changes to DBMS.*

Classic Database Administration

- *Physical Database Design (Indexes, Partitioning)*
- *Knob Configuration*
- *SQL Tuning*



Source: Bohan Zhang

What Does the Next 20 Years Look Like?

Challenge #1:

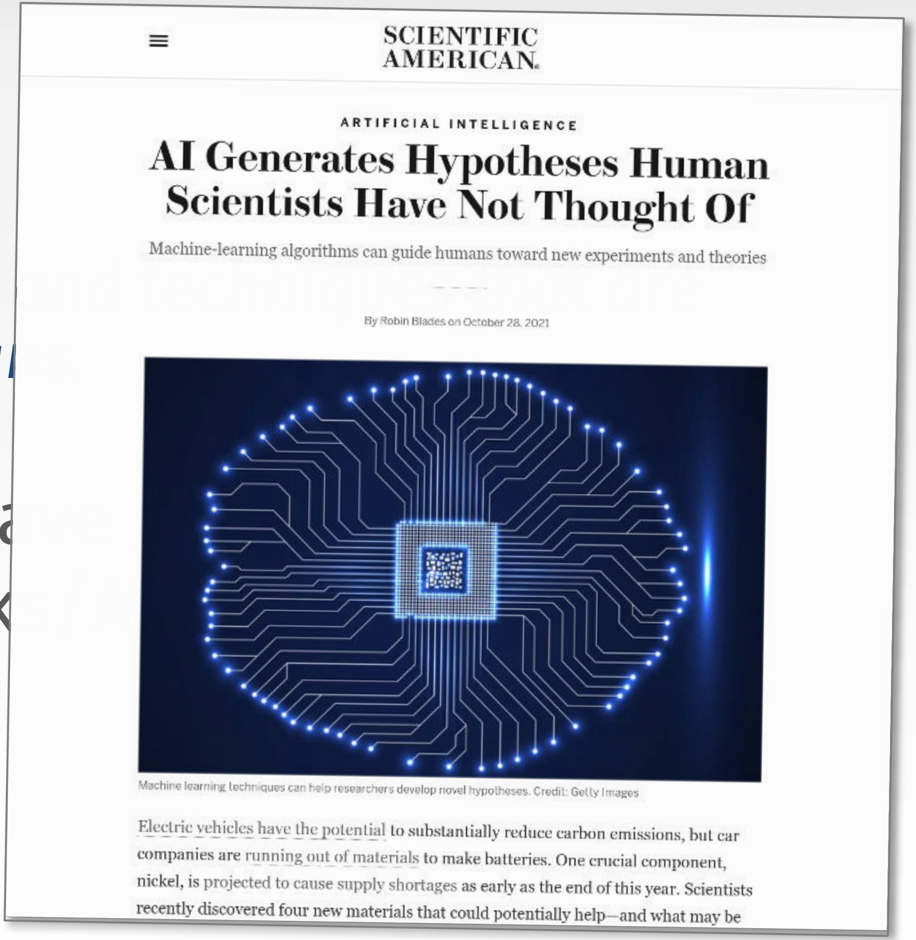
– *Remove the need for humans to perform any administrative task that does not require a human value judgement on externalities.*

Existing automation methods are reactive.
Humans are also proactive.

Challenge #2:

– *Discover new optimizations*
currently unknown to huma

This requires a DBMS to ha
and instrumentation hook


SCIENTIFIC AMERICAN.

ARTIFICIAL INTELLIGENCE

AI Generates Hypotheses Human Scientists Have Not Thought Of

Machine-learning algorithms can guide humans toward new experiments and theories

By Robin Blades on October 28, 2021

Machine learning techniques can help researchers develop novel hypotheses. Credit: Getty Images

Electric vehicles have the potential to substantially reduce carbon emissions, but car companies are running out of materials to make batteries. One crucial component, nickel, is projected to cause supply shortages as early as the end of this year. Scientists recently discovered four new materials that could potentially help—and what may be

Current ML methods are trying to create better versions of existing DBMS components.

- *Still require human experts to understand how leverage ML properly in the system.*

The next challenge is how to use ML to find things beyond human thoughts.

END

@andy_pavlo