Lecture #25

Carnegie Mellon University

ADVANCED DATABASE SYSTEMS

Self-Driving Database Management Systems

@Andy_Pavlo // 15-721 // Spring 2019

TODAY'S AGENDA

Autonomous DBMS History
Self-Driving DBMSs
Learned Components

MOTIVATION

Personnel is **~50%** of the TOC of a DBMS.

Average DBA Salary (2017): **$89,050**

The scale and complexity of DBMS installations have surpassed humans.

Source: https://www.bls.gov/oes/current/oes151141.htm

Source: https://www.highbeam.com/doc/1P3-1149052351.html

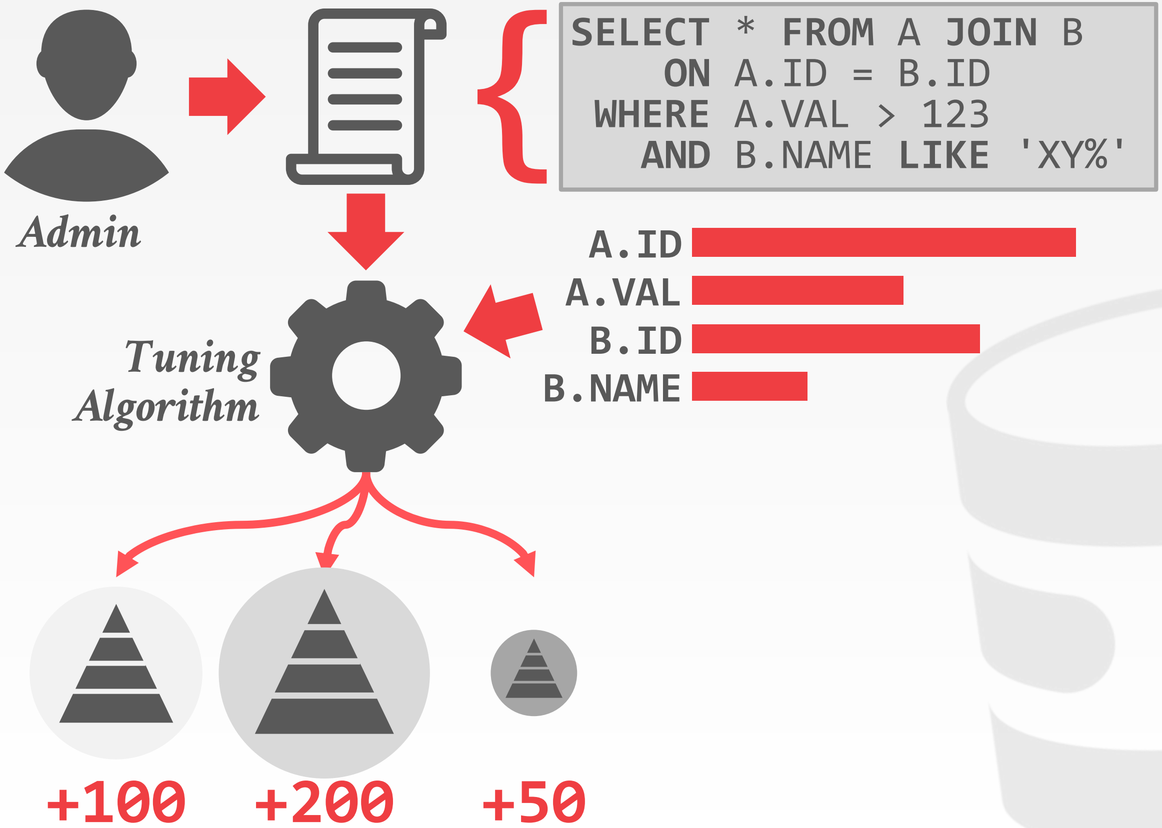
SELF-ADAPTIVE DATABASES (1970s-1990s)

Index Selection

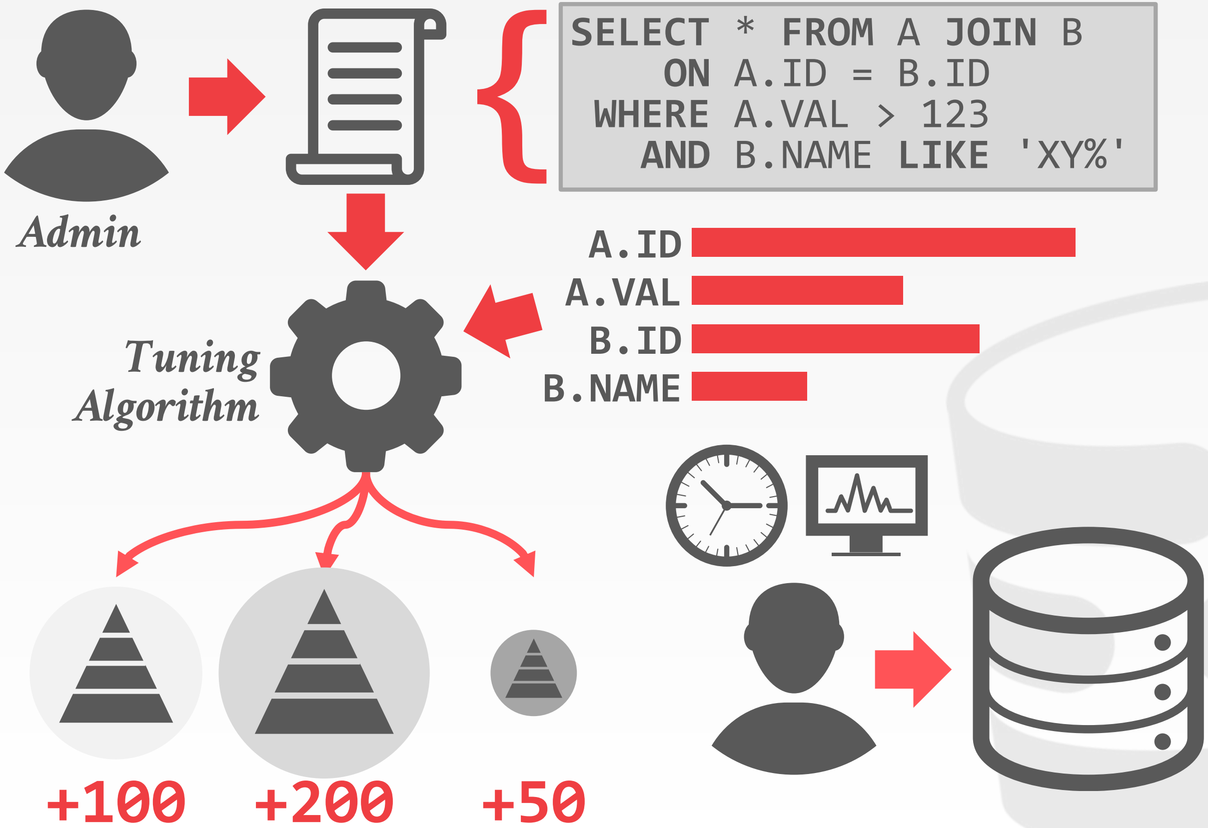
Partitioning / Sharding Keys

Data Placement

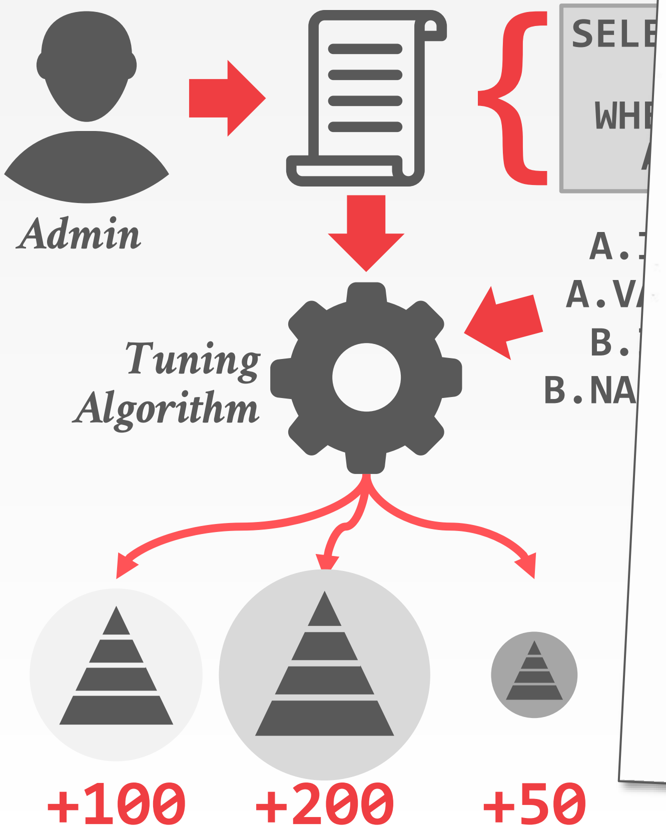
SELF-ADAPTIVE DATABASES (1970s-1990s)

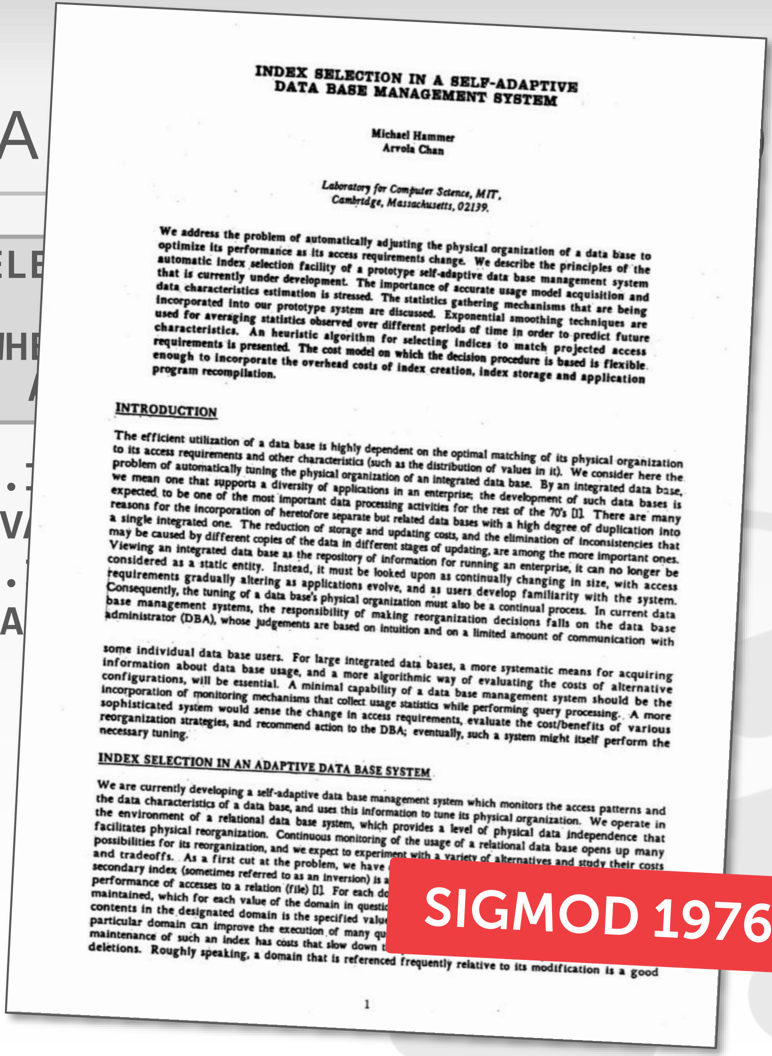


SELF-ADAPTIVE DATABASES (1970s-1990s)



SELF-ADAPTIVE DATABA





INDEX SELECTION IN A SELF-ADAPTIVE
DATA BASE MANAGEMENT SYSTEM

Michael Hammer
Arvola Chan

Laboratory for Computer Science, MIT,
Cambridge, Massachusetts, 02139.

We address the problem of automatically adjusting the physical organization of a data base to
optimize its performance as its access requirements change. We describe the principles of the
automatic index selection facility of a prototype self-adaptive data base management system
that is currently under development. The importance of accurate usage model acquisition and
data characteristics estimation is stressed. The statistics gathering mechanisms that are being
incorporated into our prototype system are discussed. Exponential smoothing techniques are
used for averaging statistics observed over different periods of time in order to predict future
characteristics. An heuristic algorithm for selecting indices to match projected access
requirements is presented. The cost model on which the decision procedure is based is flexible
enough to incorporate the overhead costs of index creation, index storage and application
program recompilation.

INTRODUCTION

The efficient utilization of a data base is highly dependent on the optimal matching of its physical organization
to its access requirements and other characteristics (such as the distribution of values in it). We consider here the
problem of automatically tuning the physical organization of an integrated data base. By an integrated data base,
we mean one that supports a diversity of applications in an enterprise; the development of such data bases is
expected to be one of the most important data processing activities for the rest of the 70's [1]. There are many
reasons for the incorporation of heretofore separate but related data bases with a high degree of duplication into
a single integrated one. The reduction of storage and updating costs, and the elimination of inconsistencies that
may be caused by different copies of the data in different stages of updating, are among the more important ones.
Viewing an integrated data base as the repository of information for running an enterprise, it can no longer be
considered as a static entity. Instead, it must be looked upon as continually changing in size, with access
requirements gradually altering as applications evolve, and as users develop familiarity with the system.
Consequently, the tuning of a data base's physical organization must also be a continual process. In current data
base management systems, the responsibility of making reorganization decisions falls on the data base
administrator (DBA), whose judgements are based on intuition and on a limited amount of communication with

some individual data base users. For large integrated data bases, a more systematic means for acquiring
information about data base usage, and a more algorithmic way of evaluating the costs of alternative
configurations, will be essential. A minimal capability of a data base management system should be the
incorporation of monitoring mechanisms that collect usage statistics while performing query processing. A more
sophisticated system would sense the change in access requirements, evaluate the cost/benefits of various
reorganization strategies, and recommend action to the DBA; eventually, such a system might itself perform the
necessary tuning.

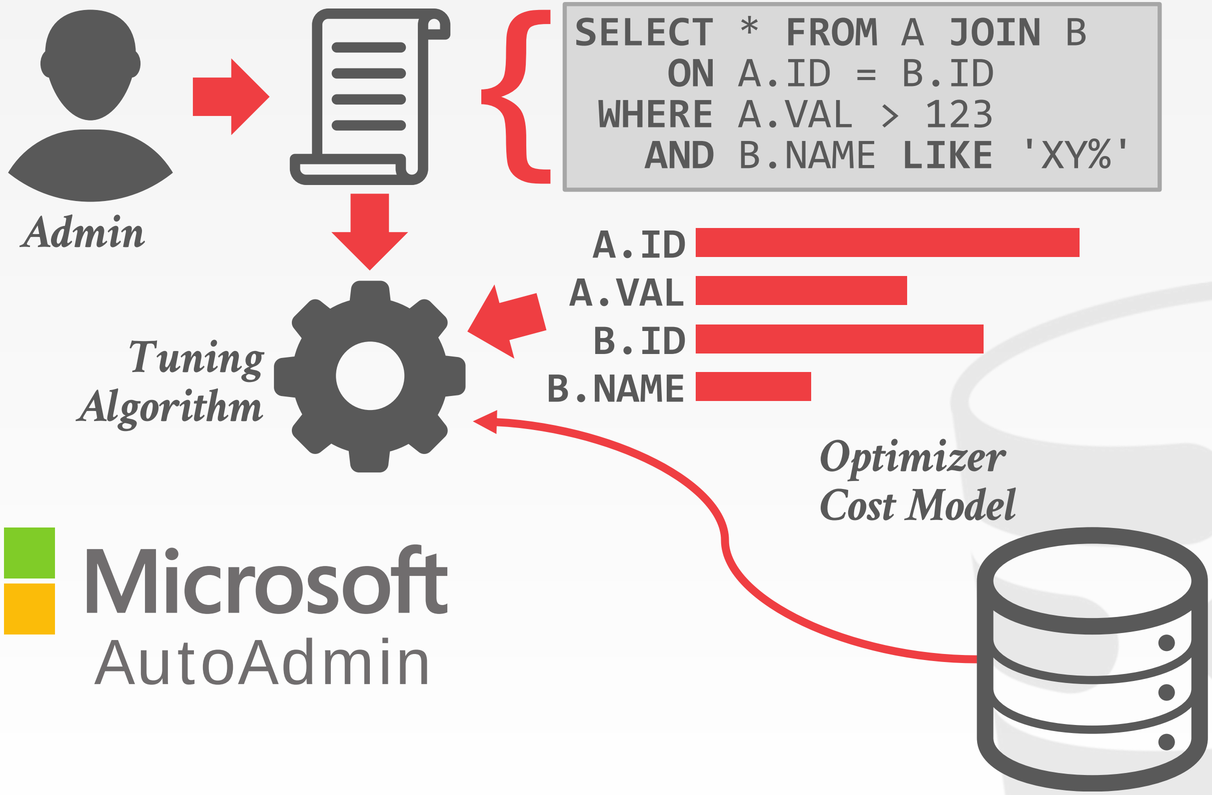
INDEX SELECTION IN AN ADAPTIVE DATA BASE SYSTEM

We are currently developing a self-adaptive data base management system which monitors the access patterns and
the data characteristics of a data base, and uses this information to tune its physical organization. We operate in
the environment of a relational data base system, which provides a level of physical data independence that
facilitates physical reorganization. Continuous monitoring of the usage of a relational data base opens up many
possibilities for its reorganization, and we expect to experiment with a variety of alternatives and study their costs
and tradeoffs. As a first cut at the problem, we have
secondary index (sometimes referred to as an inversion) is a
performance of accesses to a relation (file) [1]. For each do
maintained, which for each value of the domain in questio
contents in the designated domain is the specified valu
particular domain can improve the execution of many qu
maintenance of such an index has costs that slow down t
deletions. Roughly speaking, a domain that is referenced frequently relative to its modification is a good

1

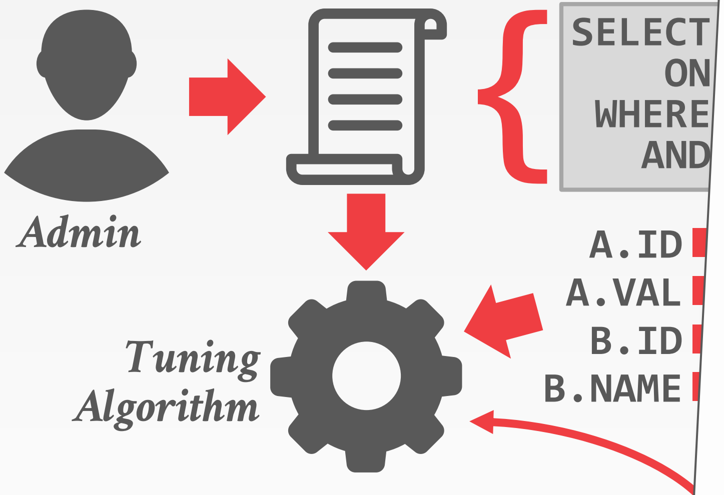
SIGMOD 1976

SELF-TUNING DATABASES (1990s-2000s)



Microsoft
AutoAdmin

SELF-TUNING DATABASES



Microsoft
AutoAdmin

Self-Tuning Database Systems: A Decade of Progress

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Vivek Narasayya
Microsoft Research
viveknar@microsoft.com

ABSTRACT
In this paper we discuss advances in self-tuning database systems over the past decade, based on our experience in the AutoAdmin project at Microsoft Research. This paper primarily focuses on the problem of automated physical database design. We also highlight other areas where research on self-tuning database technology has made significant progress. We conclude with our thoughts on opportunities and open issues.

1. HISTORY OF AUTOADMIN PROJECT

Our VLDB 1997 paper [26] reported our first technical results from the AutoAdmin project that was started in Microsoft Research in the summer of 1996. The SQL Server product group at that time had taken on the ambitious task of redesigning the SQL Server code for their next release (SQL Server 7.0). Ease of use and elimination of knobs was a driving force for their design of SQL Server 7.0. At the same time, in the database research world, data analysis and mining techniques had become popular. In starting the AutoAdmin project, we hoped to leverage some of the data analysis and mining techniques to automate difficult tuning and administrative tasks for database systems. As our first goal in AutoAdmin, we decided to focus on physical database design. This was by no means a new problem, but it was still an open problem. Moreover, it was clearly a problem that impacted performance tuning. The decision to focus on physical database design was somewhat ad-hoc. Its close relationship to query processing was an implicit driving function as the latter was our area of past work. Thus, the paper in VLDB 1997 [26] described our first solution to automating physical database design.

In this paper, we take a look back on the last decade and review some of the work on Self-Tuning Database systems. A complete survey of the field is beyond the scope of this paper. Our discussions are influenced by our experiences with the specific problems we addressed in the AutoAdmin project. Since our VLDB 1997 paper was on physical database design, a large part of this paper is also devoted to providing details of the progress in that specific sub-topic (Sections 2-6). In Section 7, we discuss briefly a few of the other important areas where self-tuning database technology have made advances over the last decade. We reflect on future directions in Section 8 and conclude in Section 9.

VLDB '07, September 23-28, 2007, Vienna, Austria.

2. AN INTRODUCTION TO PHYSICAL DATABASE DESIGN

2.1 Importance of Physical Design

A crucial property of a relational DBMS is that it provides physical data independence. This allows physical structures such as indexes to change seamlessly without affecting the output of the query; but such changes do impact efficiency. Thus, together with the capabilities of the execution engine and the optimizer, the physical database design determines how efficiently a query is executed on a DBMS.

The first generation of relational execution engines were relatively simple, targeted at OLTP, making index selection less of a problem. The importance of physical design was amplified as query optimizers became sophisticated to cope with complex decision support queries. Since query execution and optimization techniques were far more advanced, DBAs could no longer rely on a simplistic model of the engine. But, the choice of right index structures was crucial for efficient query execution over large databases.

2.2 State of the Art in 1997

The role of the workload, including queries and updates, in physical design was widely recognized. Therefore, at a high level, the problem of physical database design was - for a given workload, find a *configuration*, i.e. a set of indexes that minimize the cost. However, early approaches did not always agree on what constitutes a *workload*, or what should be measured as *cost* for a given query and configuration.

Papers on physical design of databases started appearing as early as 1974. Early work such as by Stonebraker [63] assumed a parametric model of the workload and work by Hammer and Chan [44] used a predictive model to derive the parameters. Later papers increasingly started using an explicit workload [40],[41],[56]. An explicit workload can be collected using the tracing capabilities of the DBMS. Moreover, some papers restricted the class of workloads, whether explicit or parametric, to single table queries. Sometimes such restrictions were necessary for their proposed index selection techniques to even apply and in some cases they could justify the goodness of their solution only for the restricted class of queries.

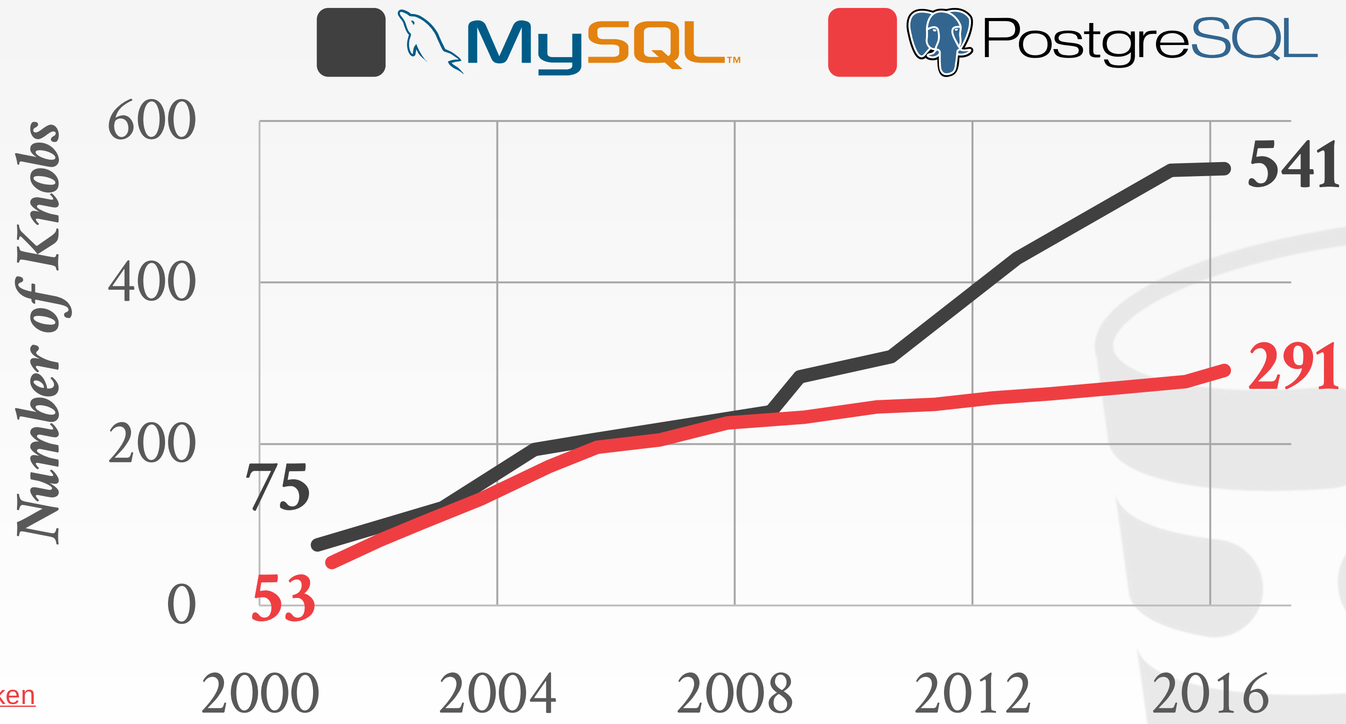
All papers recognized that it is not feasible to estimate goodness of a physical design for a workload by actual creation of indexes and then executing the queries and updates in the workload. Nonetheless, there was a lot of variance on what would be the model of cost. Some of the papers took the approach of doing the comparison among the alternatives by building their own cost model. For columns on which no indexes are present, they built

3

VLDB 2007

SELF-TUNING DATABASES (1990s-2000s)

Number of Configuration Knobs Per Release



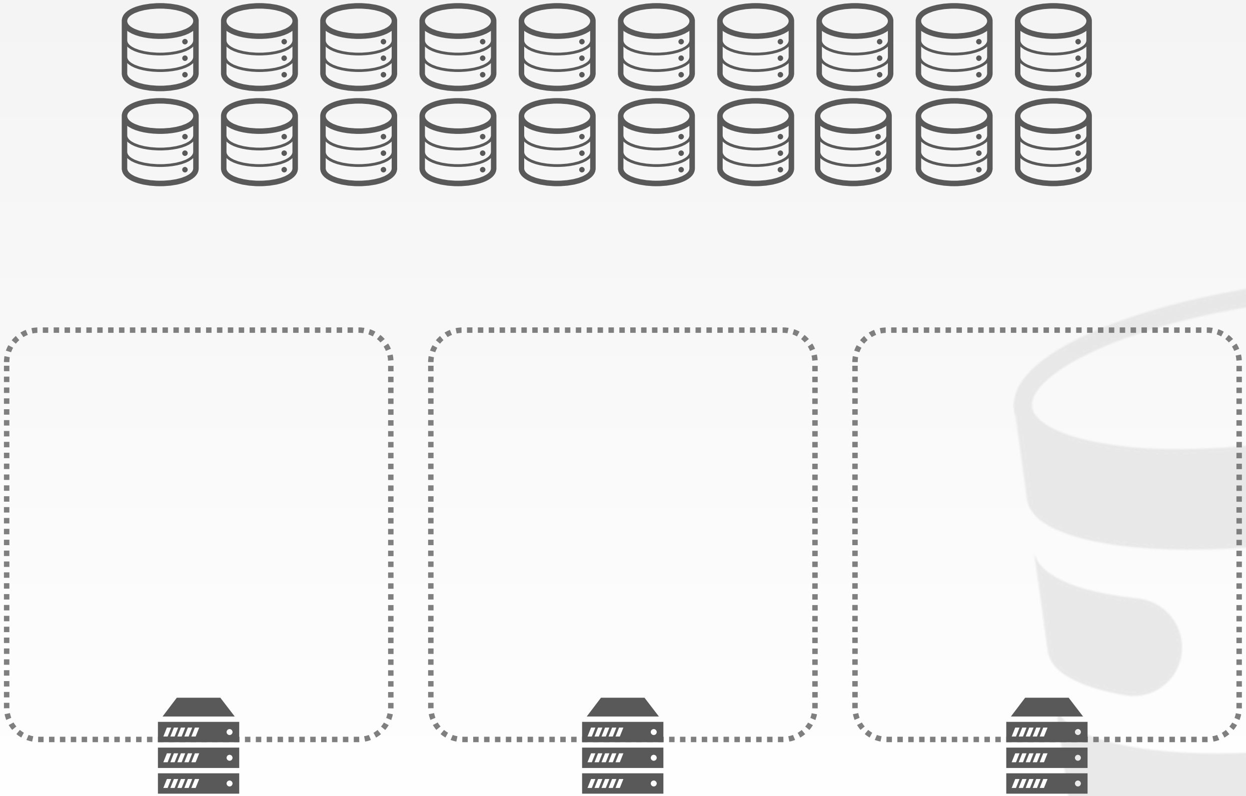
Source: Dana Van Aken

CLOUD-MANAGED DATABASES (2010S)

Initial Placement

Tenant Migration

CLOUD-MANAGED DATABASES (2010S)



CLOUD-MANAGED DATABASES (2010S)

OBSERVATION

People have been working on autonomous database systems for 45 years.

Why is this previous work insufficient?

PREVIOUS WORK

Problem #1: Human Judgements

→ User has to make final decision on whether to apply recommendations.

Problem #2: Reactionary Measures

→ Can only solve previous problems. Cannot anticipate upcoming usage trends / issues.

Problem #3: No Transfer Learning

→ Tunes each DBMS instance in isolation. Cannot apply knowledge learned about one DBMS to another.

OBSERVATION

Just like there are different levels of autonomy in cars, there are different levels for databases.

→ SAE (J3016) Automation Levels

We need to reason about the autonomous systems to understand their capabilities and limitations.

→ This will help us reason about how much a human needs to be involved in its administration.

AUTONOMOUS DBMS TAXONOMY

System only does what humans tell it to do. } **Level #0: Manual**

AUTONOMOUS DBMS TAXONOMY

Recommendation tools that suggest improvements. Human makes final decisions.

Level #0: Manual

Level #1: Assistant

AUTONOMOUS DBMS TAXONOMY

DBMS and humans work together to mange the system.

Human guides the process.

Level #0: Manual

Level #1: Assistant

Level #2: Mixed Management

AUTONOMOUS DBMS TAXONOMY

Level #0: Manual

Level #1: Assistant

Level #2: Mixed Management

Level #3: Local Optimizations

Subsystems can adapt without human guidance.

No higher-level coordination.

AUTONOMOUS DBMS TAXONOMY

Level #0: Manual

Level #1: Assistant

Level #2: Mixed Management

Level #3: Local Optimizations

Level #4: Direct Optimizations

Human only provides high-level direction + hints.

System can identify when it needs to ask humans for help.

AUTONOMOUS DBMS TAXONOMY

Level #0: Manual

Level #1: Assistant

Level #2: Mixed Management

Level #3: Local Optimizations

Level #4: Direct Optimizations

Level #5: Self-Driving

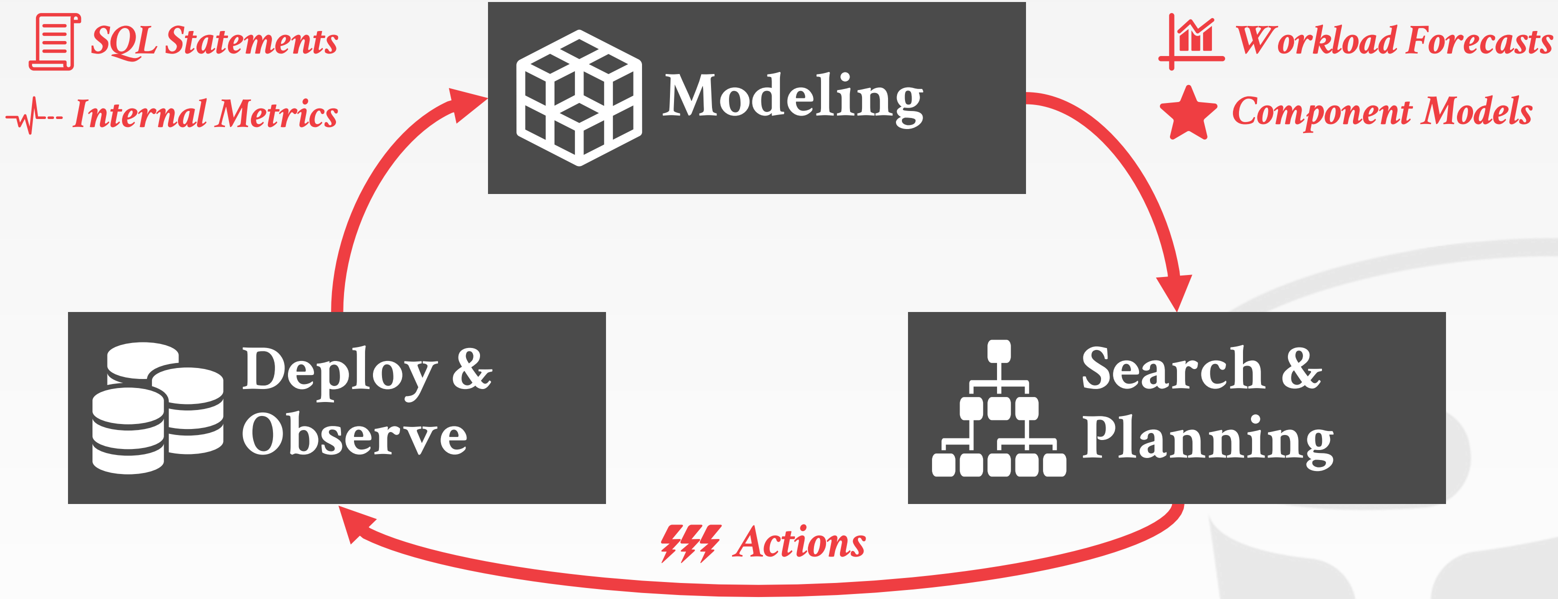
SELF-DRIVING DATABASE

A DBMS that can deploy, configure, and tune itself automatically <u>without</u> any human intervention.

→ Select actions to improve some objective function (e.g., throughput, latency, cost).
→ Choose when to apply an action.
→ Learn from these actions and refine future decision making processes.

SELF-DRIVING DATABASE MANAGEMENT SYSTEMS
CIDR 2017

ARCHITECTURE OVERVIEW



Where to Deploy?

When to Deploy?

How to Deploy?

Why?

SELF-DRIVING ENGINEERING

Environment Observations

→ How the DBMS collects training data.

Action Meta-Data

→ How the DBMS implements and exposes methods for controlling and modifying the system's configuration.

Action Engineering

→ How the DBMS deploys actions either for training or optimization.

MAKE YOUR DATABASE DREAM OF ELECTRIC SHEEP: ENGINEERING FOR SELF-DRIVING OPERATION
UNPUBLISHED MANUSCRIPT 2019

ENVIRONMENT OBSERVATIONS

Logical Workload History
→ SQL queries with their execution context.
→ Need to compress to reduce storage size.

Runtime Metrics
→ Internal measurements about the DBMS's runtime behavior.

Database Contents
→ Succinct representation/encoding of the database tables.

SUB-COMPONENT METRICS

If the DBMS has sub-components that are tunable, then it must expose separate metrics for those components.

Bad Example: RocksDB

SUB-COMPONENT METRICS

RocksDB Column Family Knobs

```
rocksdb_override_cf_options=\
     cf_link_pk={prefix_extractor=capped:20}
```

SUB-COMPONENT METRICS

RocksDB Column Family Knobs

```
rocksdb_override_cf_options=\
    cf_link_pk={prefix_extractor=capped:20}
```

Column Family Metrics

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ROCKSDB_CFSTAT;
+----------+-------------------------------------+-----------+
|  CF_NAME | METRIC_NAME                         | VALUE     |
+----------+-------------------------------------+-----------+
|  default | COMPACTION_PENDING                  | 1         |
|  default | CUR_SIZE_ACTIVE_MEM_TABLE           | 21672     |
|  default | CUR_SIZE_ALL_MEM_TABLES             | 21672     |
|  default | MEM_TABLE_FLUSH_PENDING             | 0         |
|  default | NON_BLOCK_CACHE_SST_MEM_USAGE       | 0         |
|  default | NUM_ENTRIES_ACTIVE_MEM_TABLE        | 18        |
|  default | NUM_ENTRIES_IMM_MEM_TABLES          | 0         |
|  default | NUM_IMMUTABLE_MEM_TABLE             | 0         |
|  default | NUM_LIVE_VERSIONS                   | 2         |
+----------+-------------------------------------+-----------+
```

Missing:
Reads
Writes

SUB-COMPONENT METRICS

RocksDB Column Family Knobs

```
rocksdb_override_cf_options=\
     cf_link_pk={prefix_extractor=capped:20}
```

Global Metrics

```
mysql> SHOW GLOBAL STATUS;
+--------------------------------------+------------+
| METRIC_NAME                          | VALUE      |
+--------------------------------------+------------+
| ABORTED_CLIENTS                      | 0          |
•••
| ROCKSDB_BLOCK_CACHE_BYTES_READ       | 295700537  |
| ROCKSDB_BLOCK_CACHE_BYTES_WRITE      | 709562185  |
| ROCKSDB_BLOCK_CACHE_DATA_HIT         | 64184      |
| ROCKSDB_BLOCK_CACHE_DATA_MISS        | 1001083    |
| ROCKSDB_BYTES_READ                   | 5573794    |
| ROCKSDB_BYTES_WRITTEN                | 5817440    |
| ROCKSDB_FLUSH_WRITE_BYTES            | 2906847    |
•••
| UPTIME_SINCE_FLUSH_STATUS            | 5996       |
+--------------------------------------+------------+
```

Aggregated Metrics

ACTION META-DATA

Configuration Knobs

→ Untunable flags
→ Value ranges

Dependencies

→ No hidden dependencies
→ Dynamic actions (i.e., an action creates new actions).

UNTUNABLE KNOBS

Anything that requires a human value judgement should be marked as off-limits to autonomous components.

→ File Paths
→ Network Addresses
→ Durability / Isolation Levels

KNOB HINTS

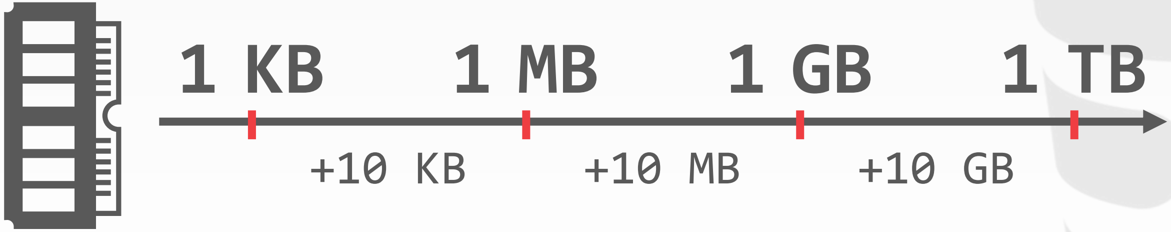
The autonomous components need hints about how to change a knob.

→ Min/max ranges.
→ Separate knobs to enable/disable a feature.
→ Non-uniform deltas.

KNOB HINTS

The autonomous components need hints about how to change a knob.

→ Min/max ranges.
→ Separate knobs to enable/disable a feature.
→ Non-uniform deltas.



ACTION ENGINEERING

No Downtime

Notifications

Replicated Training

NO DOWNTIME

The DBMS must be able to deploy any action without incurring downtime.
→ Restart vs. Unavailability

Without this, the system has to include the downtime in its cost model estimations.
→ Bad Example: MySQL Log File Size

NOTIFICATIONS

Provide a notification to indicate when an action starts and when it completes.

→ Need to know whether degradation is due to deployment or bad decision.

Harder for changes that can be used before the action completes.

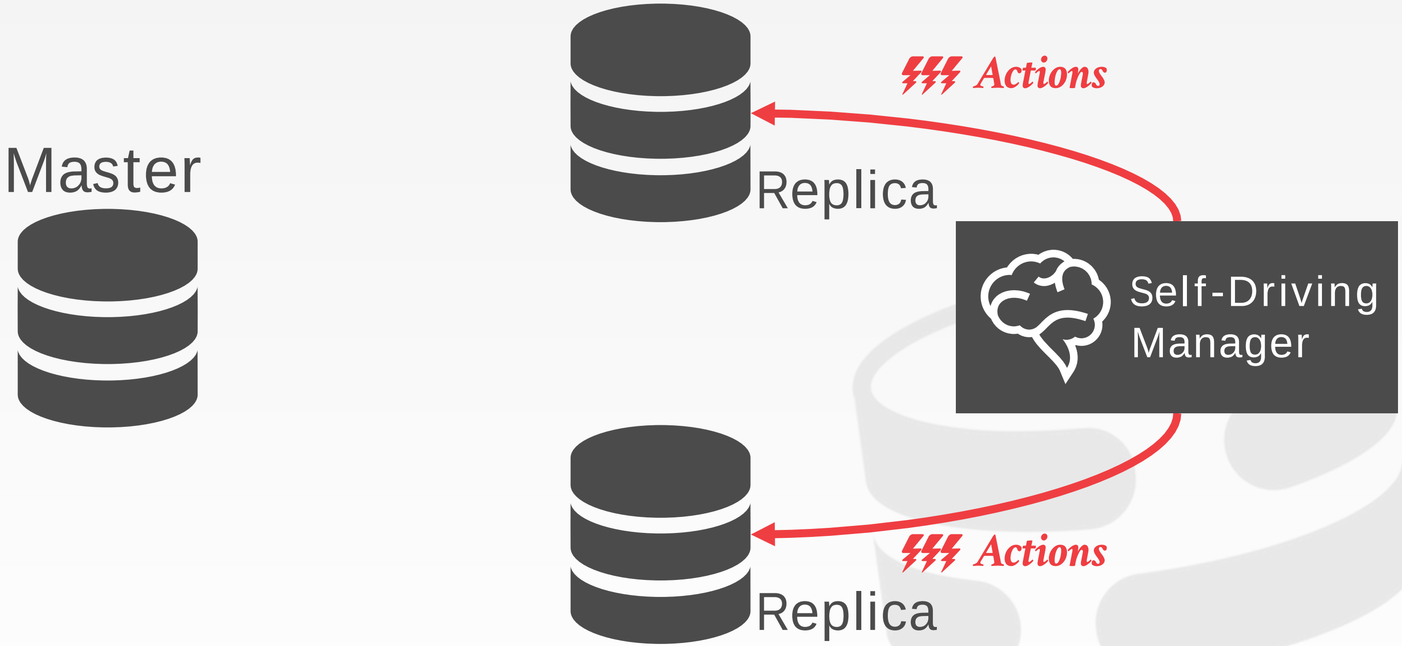
REPLICATED TRAINING

ML models need lots of training data. But getting this data is expensive in a DBMS.

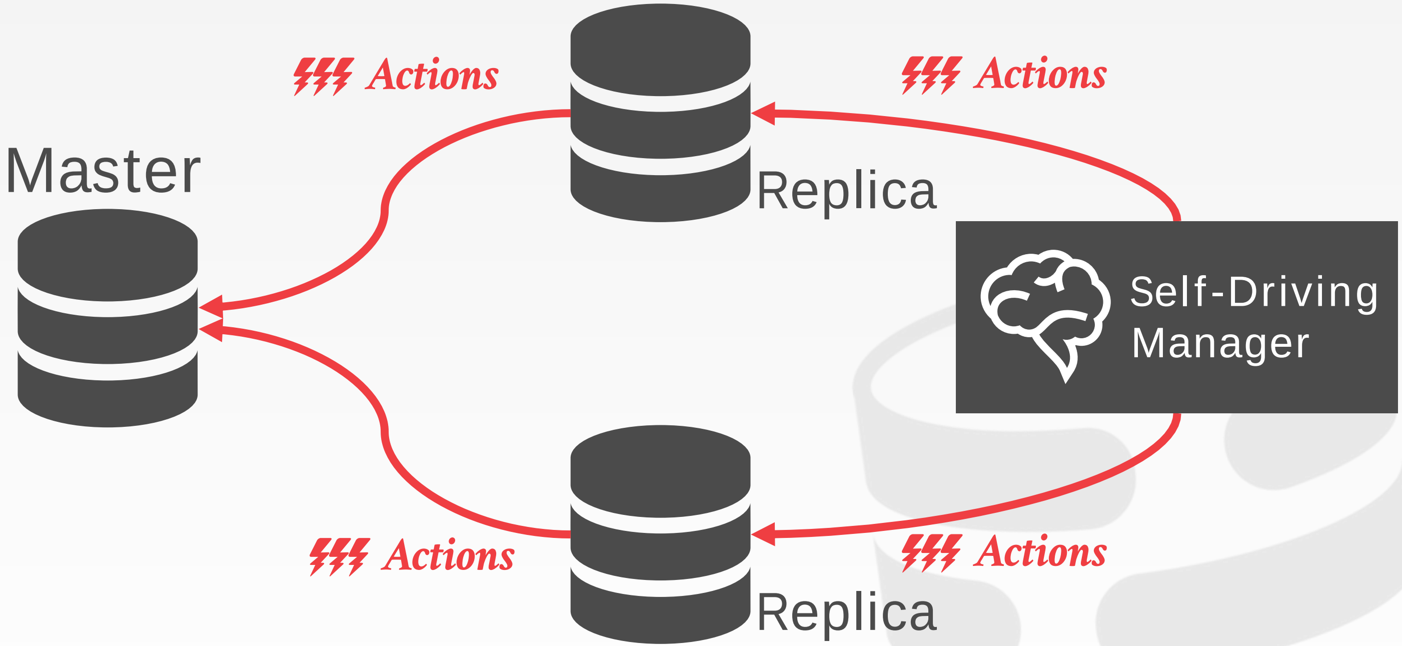
→ We don't want to slow down a production DBMS.
→ Building a simulator for the DBMS is too hard.

Ongoing Research: How to use the DBMS's replicas to explore configurations and train its models.

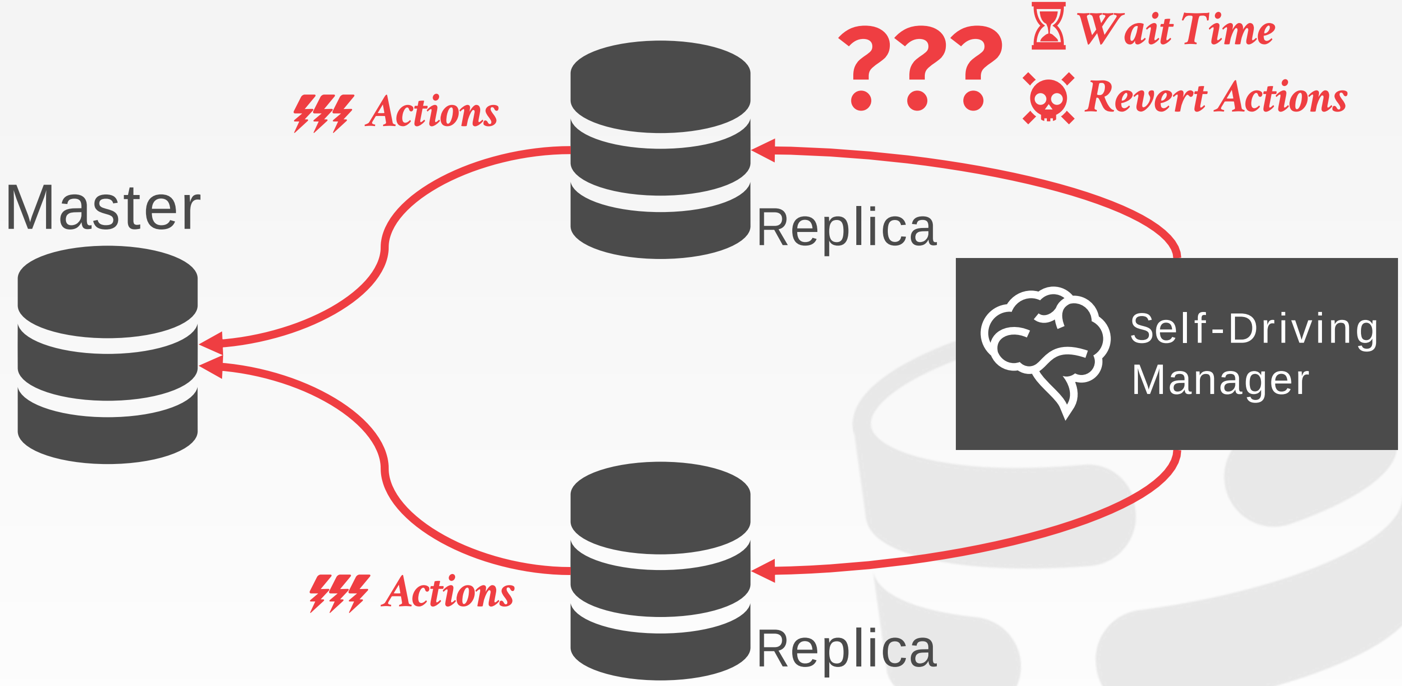
REPLICATED TRAINING



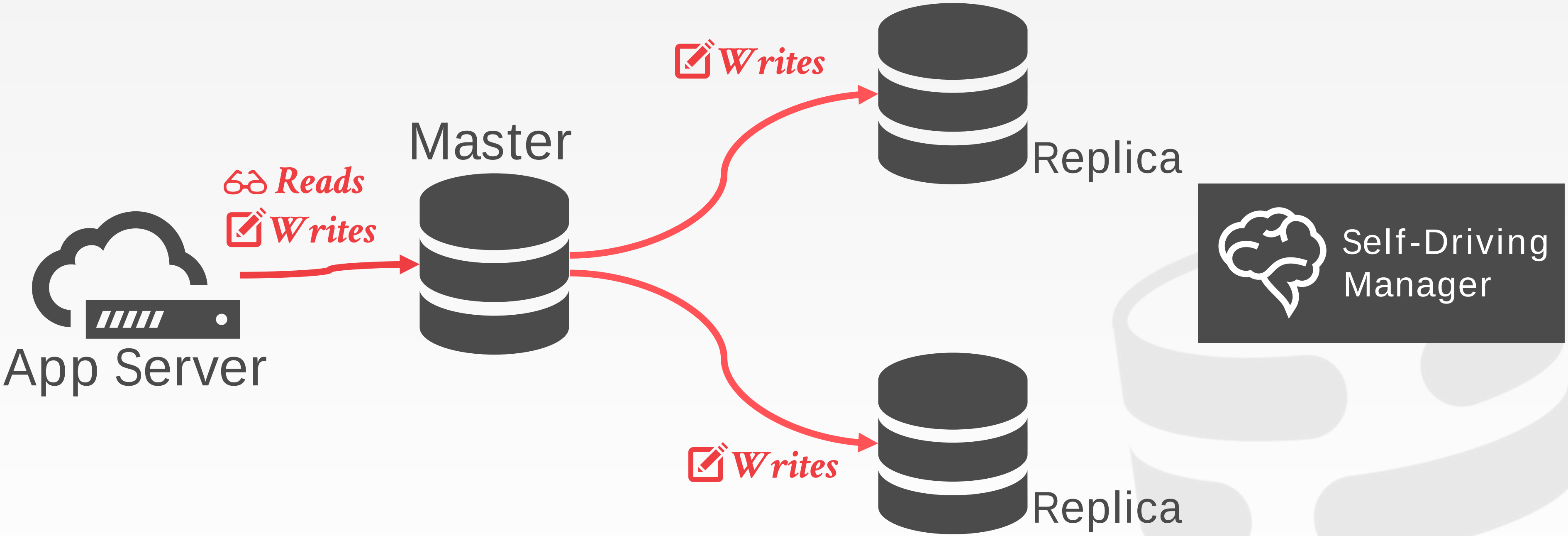
REPLICATED TRAINING



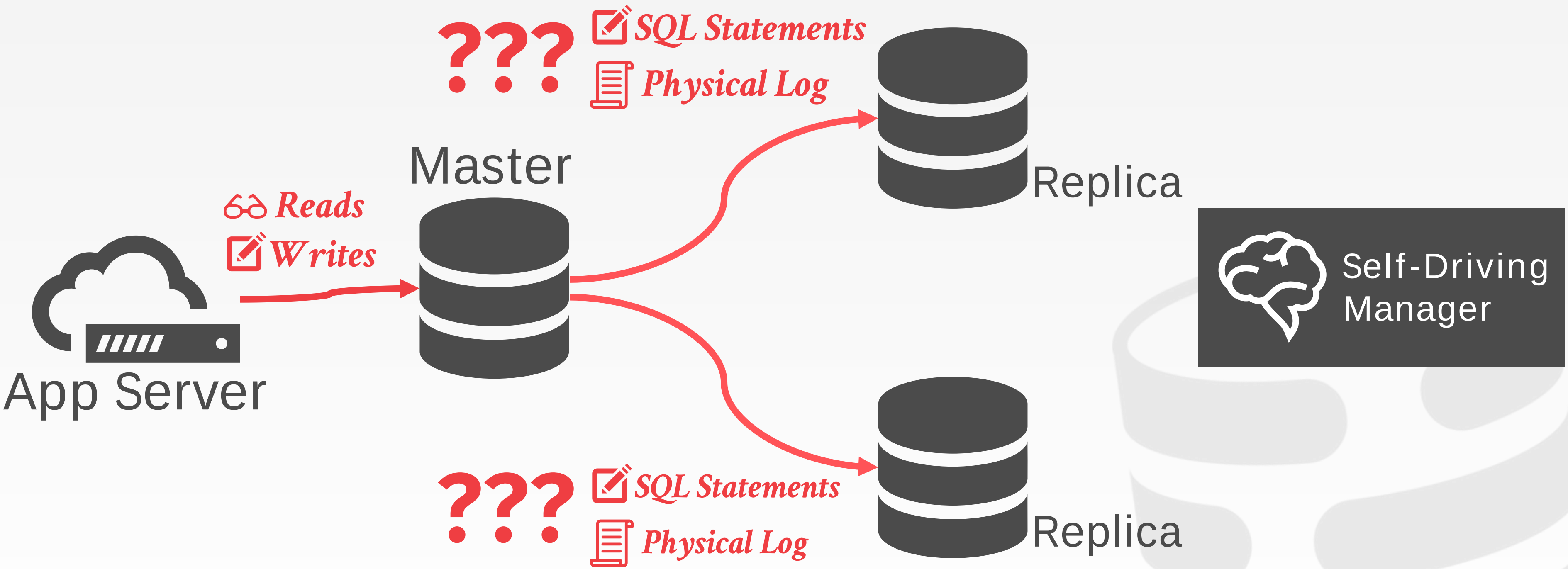
REPLICATED TRAINING



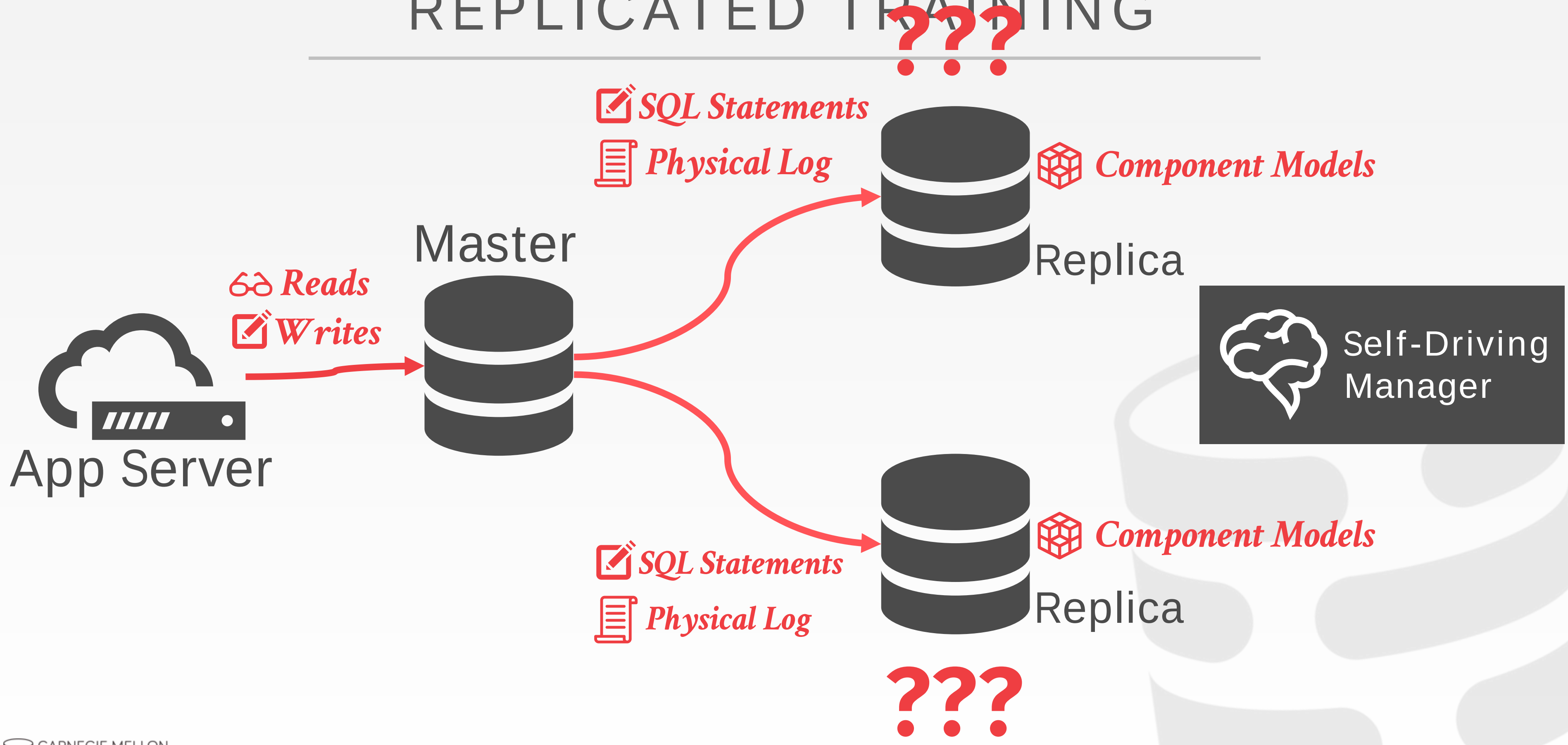
REPLICATED TRAINING



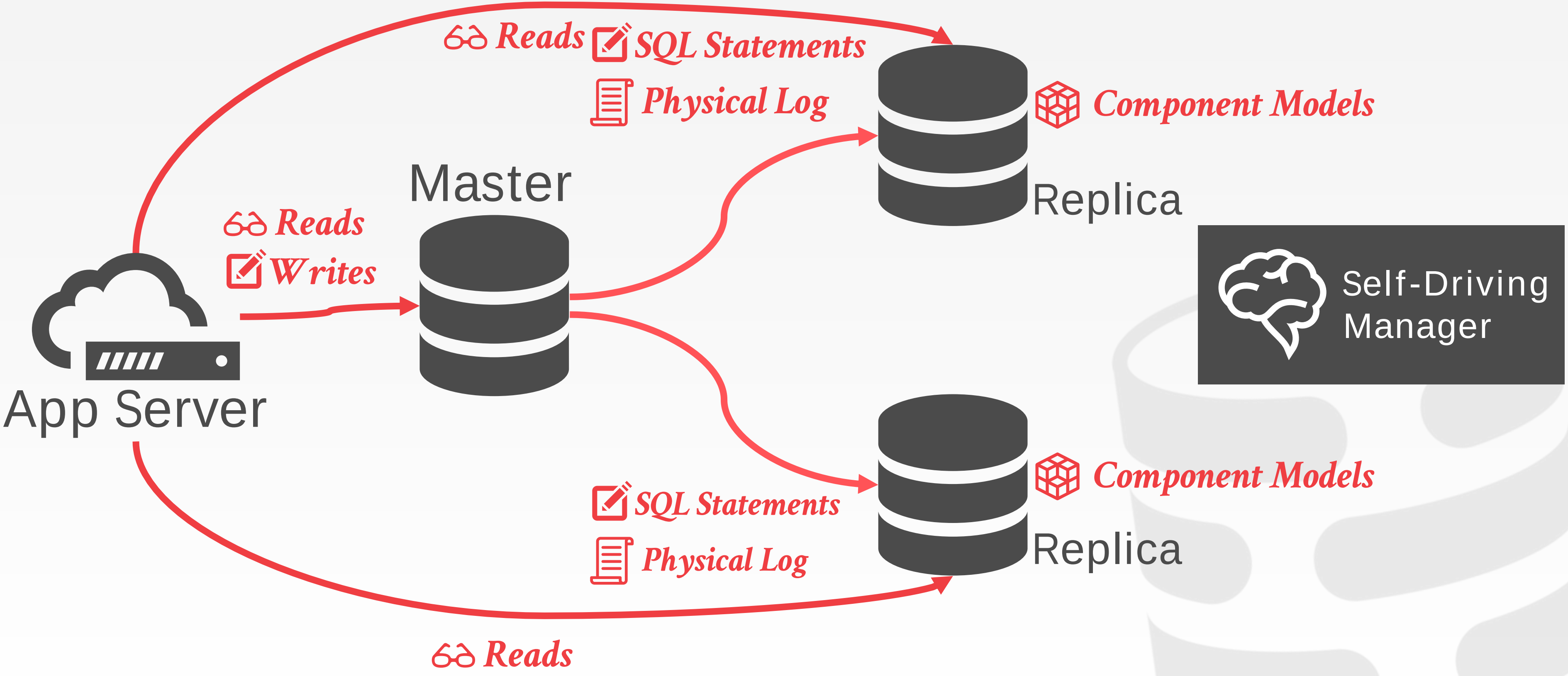
REPLICATED TRAINING



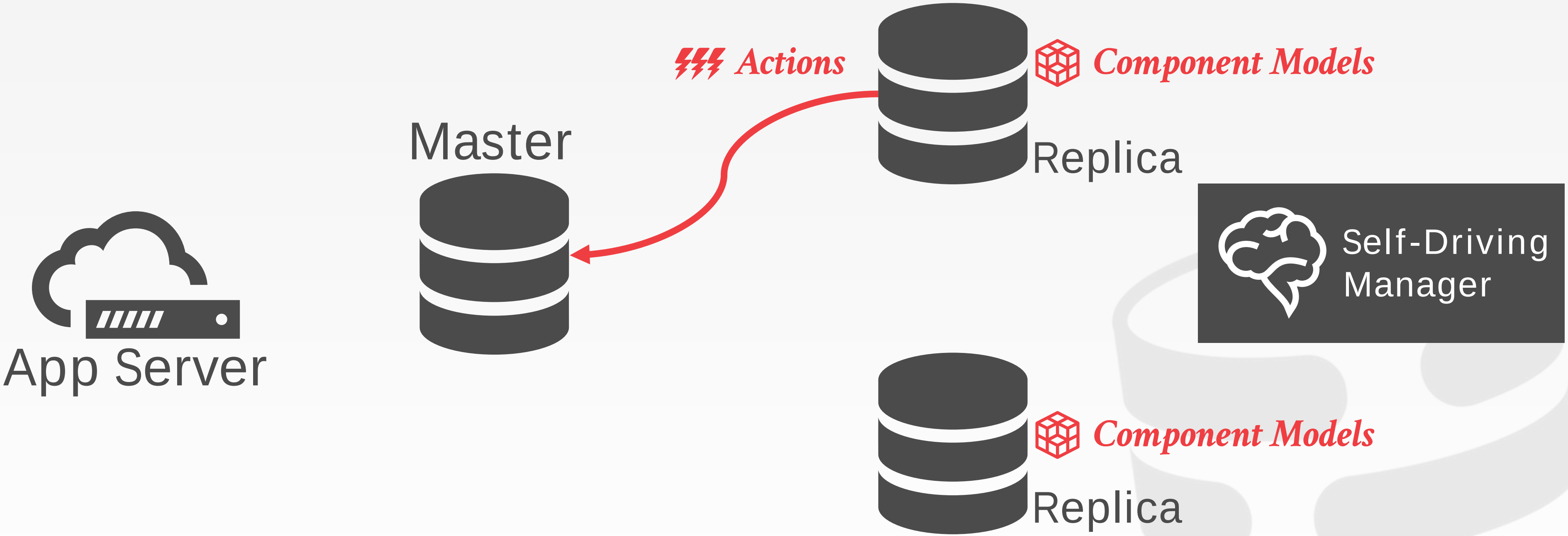
REPLICATED TRAINING



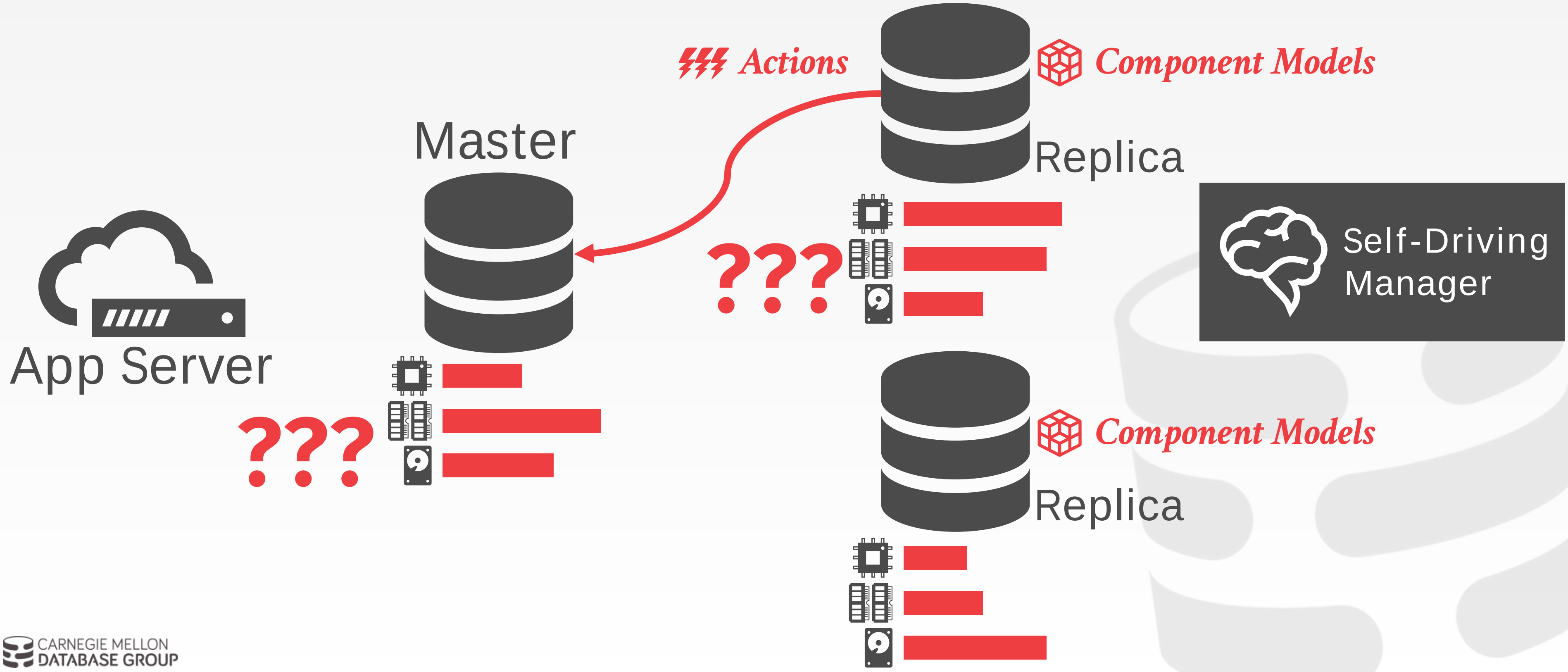
REPLICATED TRAINING



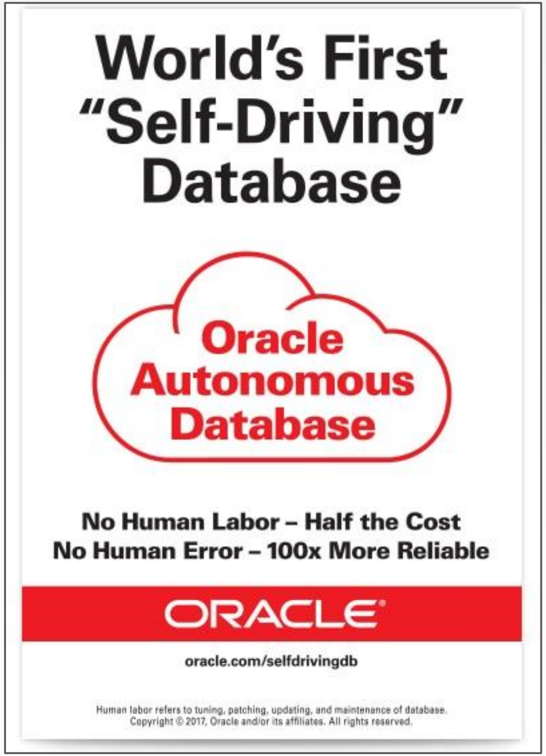
REPLICATED TRAINING



REPLICATED TRAINING



ORACLE SELF-DRIVING DBMS



September 2017

Self-Driving Database Management Systems

Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon
Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic
Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu*, Ran Xian, Tieying Zhang
Carnegie Mellon University, *National University of Singapore

ABSTRACT

In the last two decades, both researchers and vendors have built advisory tools to assist database administrators (DBAs) in various aspects of system tuning and physical design. Most of this previous work, however, is incomplete because they still require humans to make the final decisions about any changes to the database and are reactionary measures that fix problems after they occur.

What is needed for a truly "self-driving" database management system (DBMS) is a new architecture that is designed for autonomous operation. This is different than earlier attempts because all aspects of the system are controlled by an integrated planning component that not only optimizes the system for the current workload, but also predicts future workload trends so that the system can prepare itself accordingly. With this, the DBMS can support all of the previous tuning techniques without requiring a human to determine the right way and proper time to deploy them. It also enables new optimizations that are important for modern high-performance DBMSs, but which are not possible today because the complexity of managing these systems has surpassed the abilities of human experts.

This paper presents the architecture of Peloton, the first self-driving DBMS. Peloton's autonomic capabilities are now possible due to algorithmic advancements in deep learning, as well as improvements in hardware and adaptive database architectures.

1. INTRODUCTION

The idea of using a DBMS to remove the burden of data management was one of the original selling points of the relational model and declarative query languages from the 1970s [31]. With this approach, a developer only writes a query that specifies what data they want to access. The DBMS then finds the most efficient way to store and retrieve data, and to safely interleave operations.

Over four decades later, DBMSs are now the critical part of every data-intensive application in all facets of society, business, and science. These systems are also more complicated now with a long and growing list of functionalities. But using existing automated tuning tools is an onerous task, as they require laborious preparation of workload samples, spare hardware to test proposed updates, and above all else intuition into the DBMS's internals. If the DBMS could do these things automatically, then it would remove many of the complications and costs involved with deploying a database [40].

Much of the previous work on self-tuning systems is focused on standalone tools that target only a single aspect of the database. For example, some tools are able to choose the best logical or physical design of a database [16], such as indexes [30, 17, 58], partitioning schemes [6, 44], data organization [7], or materialized views [5]. Other tools are able to select the tuning parameters for an application [56, 22]. Most of these tools operate in the same way: the DBA provides it with a sample database and workload trace that guides a search process to find an optimal or near-optimal configuration. All of the major DBMS vendors' tools, including Oracle [23, 38], Microsoft [16, 42], and IBM [55, 57], operate in this manner. There is a recent push for integrated components that support adaptive architectures [36], but these again only focus on solving one problem. Likewise, cloud-based systems employ dynamic resource allocation at the service-level [20], but do not tune individual databases.

All of these are insufficient for a completely autonomous system because they are (1) external to the DBMS, (2) reactionary, or (3) not able to take a holistic view that considers more than one problem at a time. That is, they observe the DBMS's behavior from outside of the system and advise the DBA on how to make corrections to fix only one aspect of the problem after it occurs. The tuning tools assume that the human operating them is knowledgeable enough to update the DBMS during a time window when it will have the least impact on applications. The database landscape, however, has changed significantly in the last decade and one cannot assume that a DBMS is deployed by an expert that understands the intricacies of database optimization. But even if these tools were automated such that they could deploy the optimizations on their own, existing DBMS architectures are not designed to support major changes without stressing the system further nor are they able to adapt in anticipation of future bottlenecks.

In this paper, we make the case that self-driving database systems are now achievable. We begin by discussing the key challenges with such a system. We then present the architecture of **Peloton** [1], the first DBMS that is designed for autonomous operation. We conclude with some initial results on using Peloton's integrated deep learning framework for workload forecasting and action deployment.

2. PROBLEM OVERVIEW

The first challenge in a self-driving DBMS is to understand an application's workload. The most basic level is to characterize queries as being for either an OLTP or OLAP application [26]. If the DBMS identifies which of these two workload classes the application belongs to, then it can make decisions about how to optimize the database. For example, if it is OLTP, then the DBMS should store tuples in a row-oriented layout that is optimized for writes. If it is OLAP, then the DBMS should use a column-oriented

8th Biennial Conference on Innovative Data Systems Research (CIDR '17)
January 8-11, 2017 , Chaminade, California, USA.

January 2017

ORACLE SELF-DRIVING DBMS

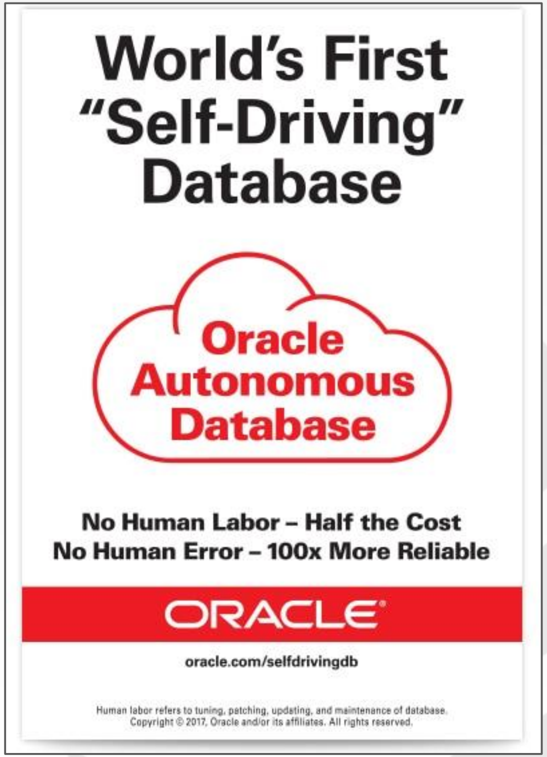
Automatic Patching

Automatic Indexing

Automatic Recovery

Automatic Scaling

Automatic Query Tuning



September 2017

ORACLE SELF-DRIVING DBMS

Automatic Patching

Automatic Indexing

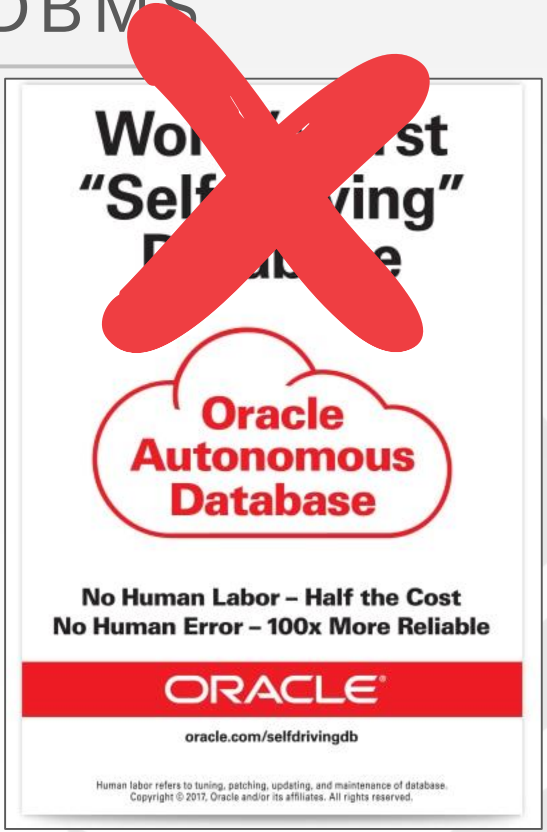
Automatic Recovery

Automatic Scaling

Automatic Query Tuning

Reactionary Measures

No Transfer Learning



September 2017

ORACLE SELF-DRIVING DBMS

Automatic Patching

Automatic Indexing

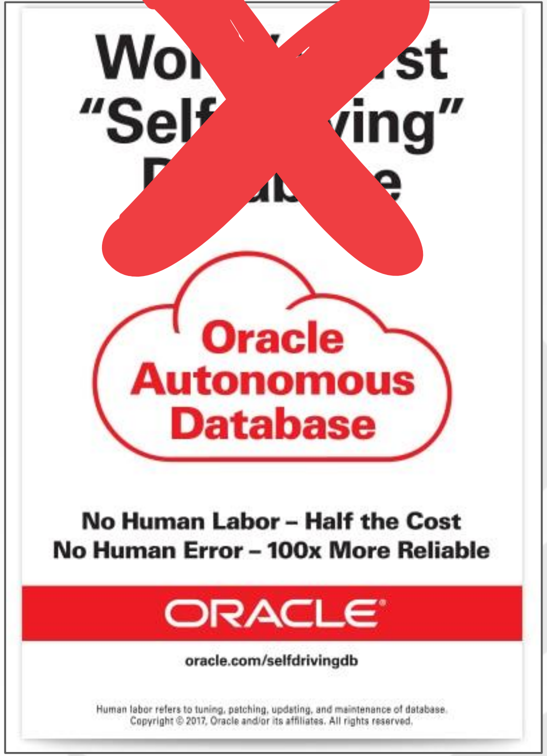
Automatic Recovery

Automatic Scaling

Automatic Query Tuning

Reactionary Measures

No Transfer Learning



September 2017

Microsoft SQL Server

IBM DB2

INGRES

OBSERVATION

There are many places in the DBMS that use human-engineered components to make decisions about the behavior of the system.

→ Optimizer Cost Models
→ Compression Algorithms
→ Data Structures
→ Scheduling Policies

What if the DBMS could "learn" these policies based on the data.

LEARNED COMPONENTS

Replace DBMS components with ML models trained at runtime.

PARTING THOUGHTS

True autonomous DBMSs are achievable in the next decade.

You should think about how each new feature can be controlled by a machine.

NEXT CLASS

SAP HANA Guest Lecture