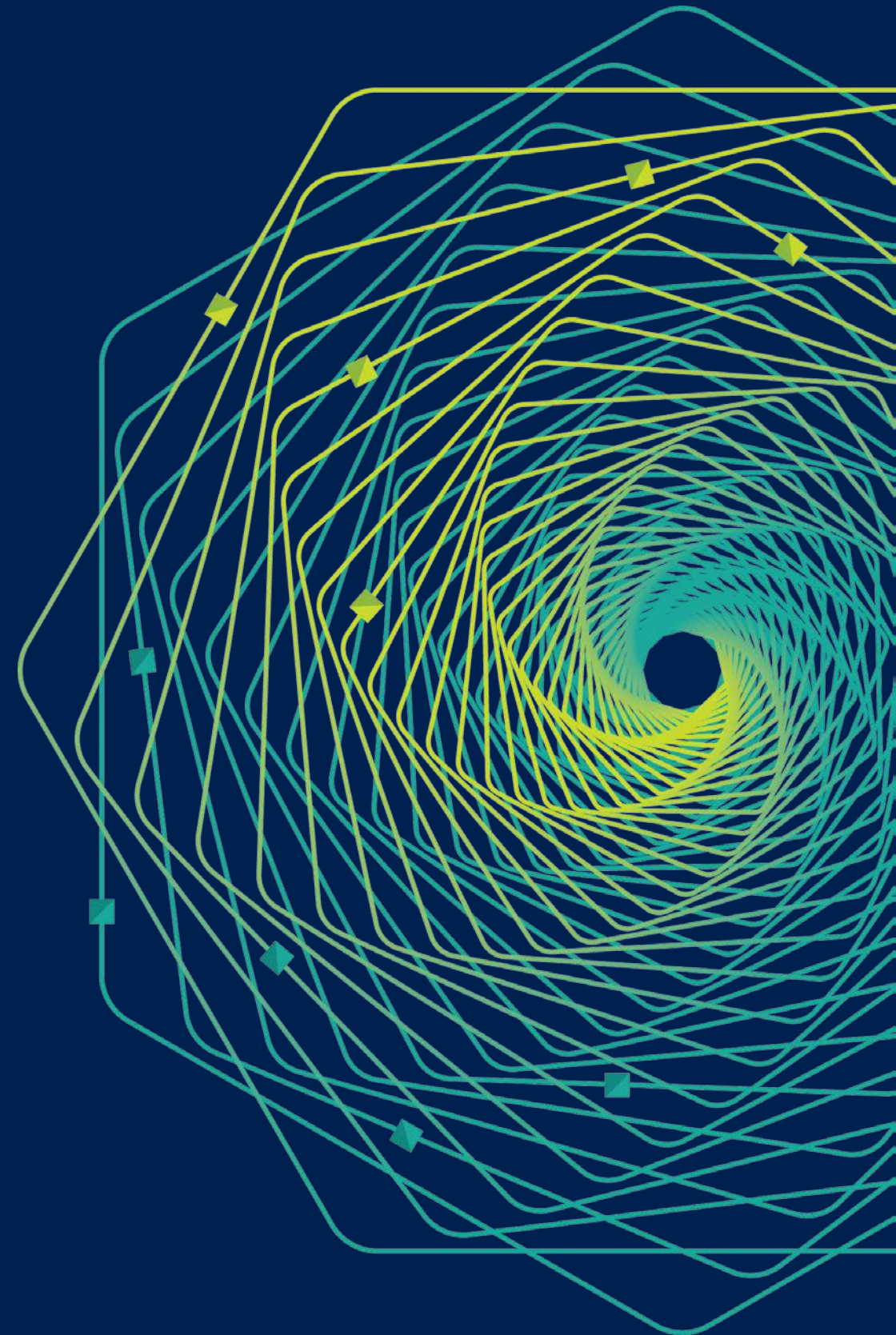


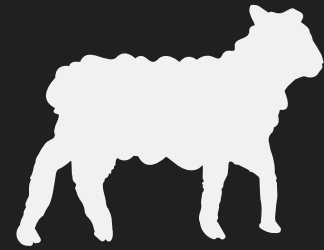
Research Faculty Summit 2018

Systems | Fueling future disruptions



MAKE YOUR
DATABASE
DREAM OF
ELECTRIC
SHEEP
DESIGNING
FOR
AUTONOMOUS
OPERATION





Part #1 – Background

Part #2 – Engineering

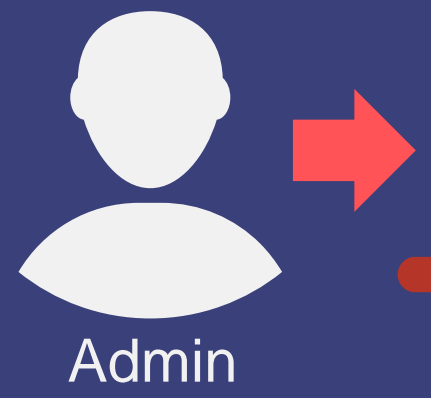
Part #3 – Oracle Rant



AUTONOMOUS DBMSs SELF-ADAPTIVE DATABASES

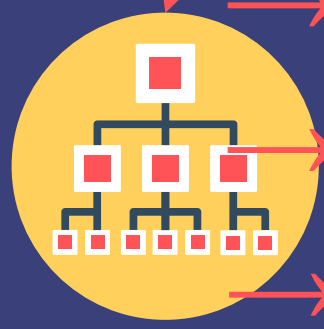


1970-1990s Self-Adaptive Databases



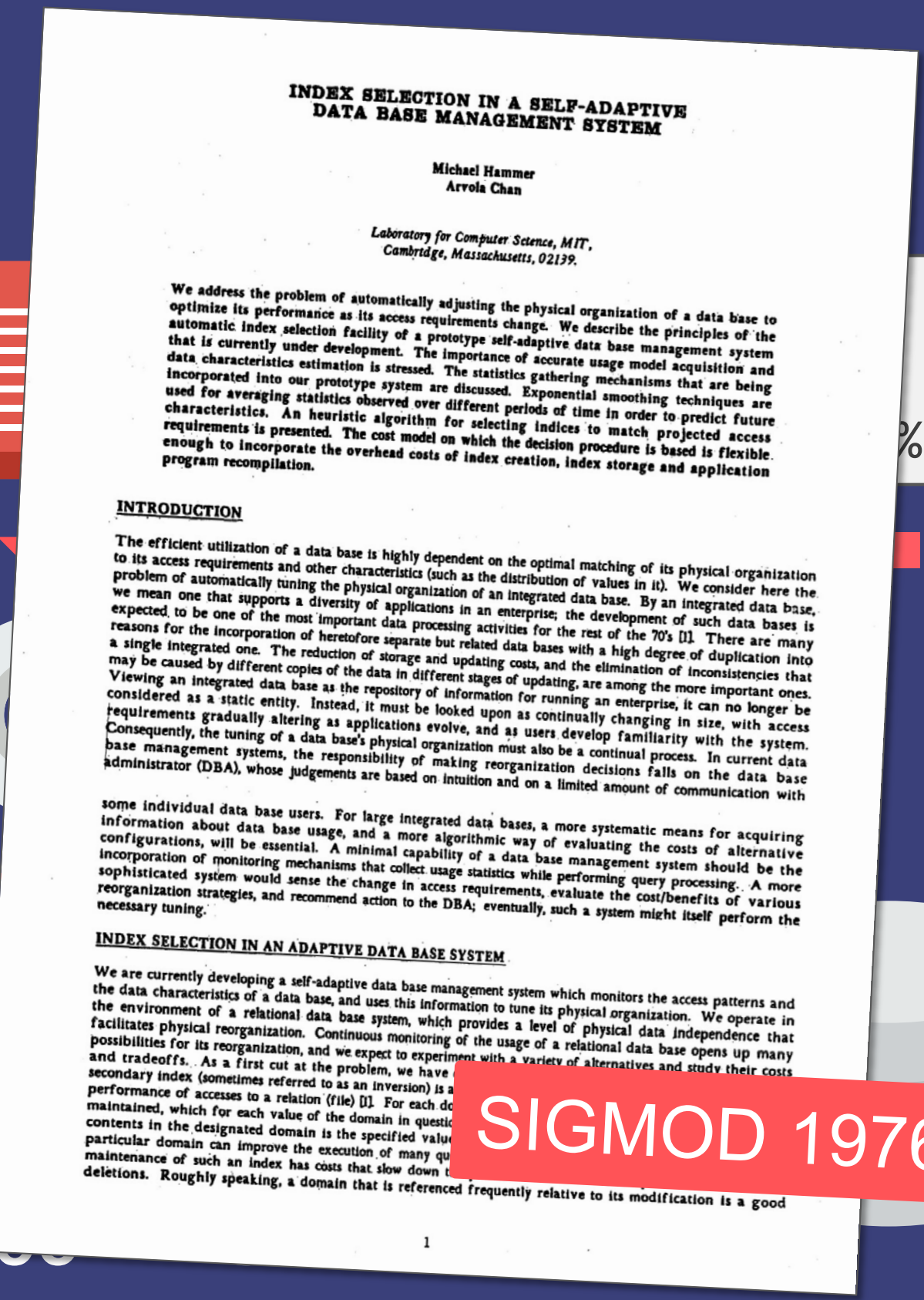
Admin

Tuning
Algorithm



+100

Index
Part
Data
+200



INDEX SELECTION IN A SELF-ADAPTIVE DATA BASE MANAGEMENT SYSTEM

Michael Hammer
Arvoia 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 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 a secondary index (sometimes referred to as an inversion) is a performance of accesses to a relation (file) [1]. For each domain maintained, which for each value of the domain in question contents in the designated domain is the specified value particular domain can improve the execution of many query maintenance of such an index has costs that slow down deletions. Roughly speaking, a domain that is referenced frequently relative to its modification is a good

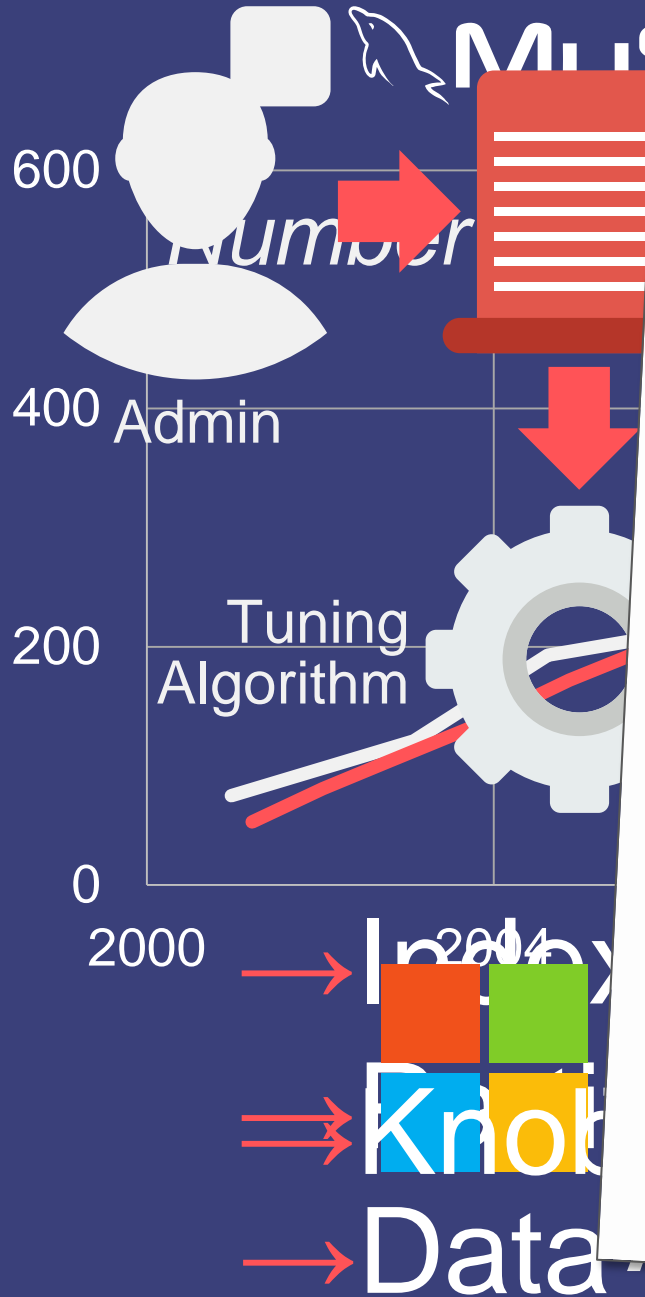
SIGMOD 1976



AUTONOMOUS DBMSs SELF-TUNING DATABASES



1990-2000s Self-Tuning Databases



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.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Database Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM.
 VLDB '07, September 23-28, 2007, Vienna, Austria.
 Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

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.

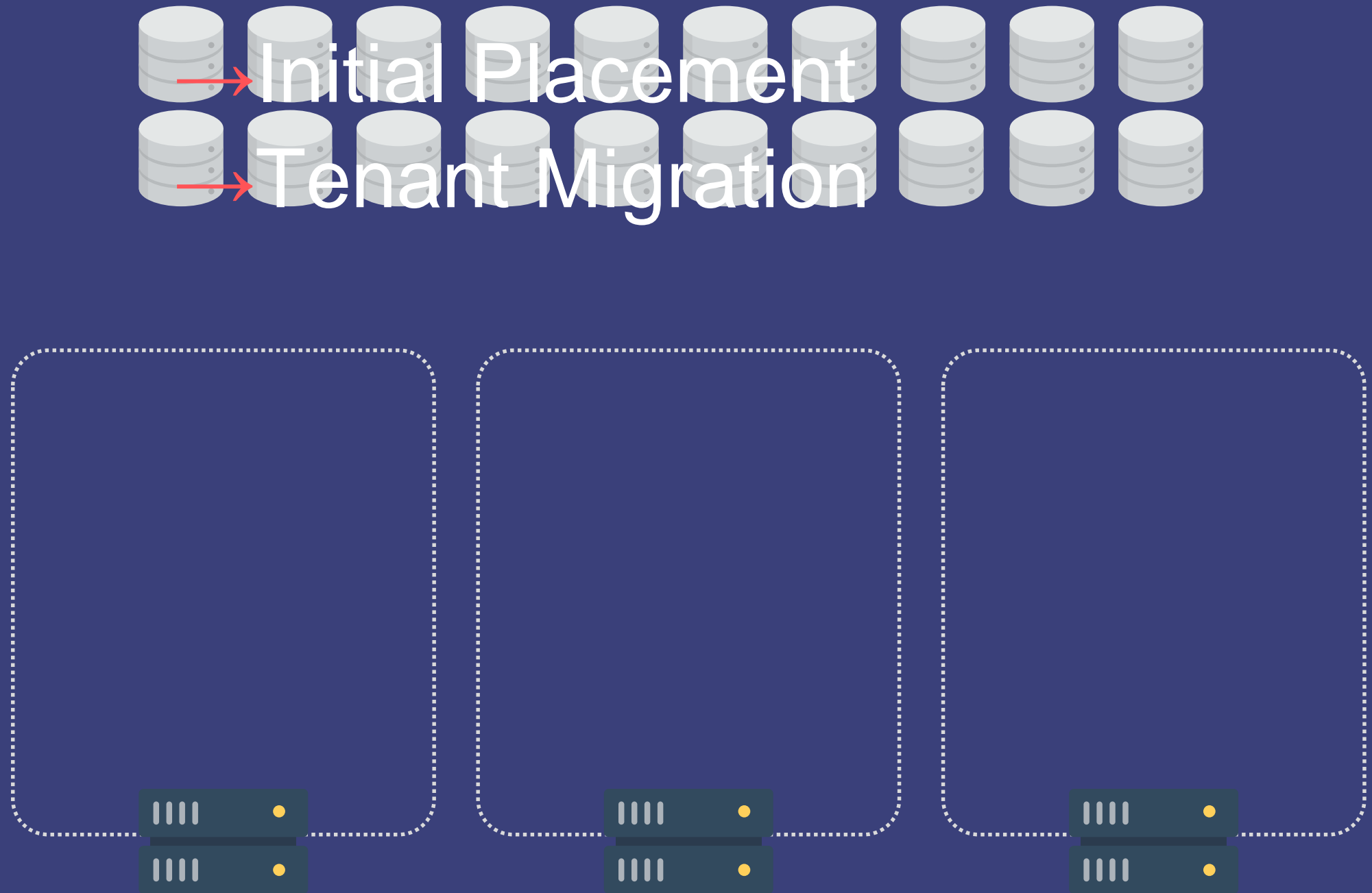
All papers recognized that it is not feasible to estimate goodness of a physical design for a workload by actual creation of indexes and then executing the queries and updates in the workload. Nonetheless, there was a lot of variance on what would be the model of cost. Some of the papers took the approach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

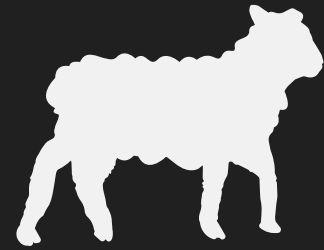
3

VLDB 2007



2010s
Cloud
Databases





Why is this Previous Work
Insufficient?



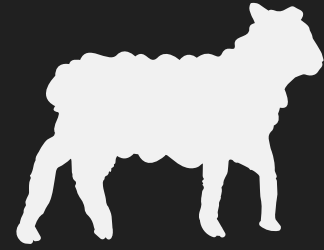
Problem #1
Human
Judgements



Problem #2
Reactionary
Measures



Problem #3
No Transfer
Learning



What is **Different** this Time?



OtterTune
Existing
Systems



Peloton
New
System



Database Tuning-as-a-Service

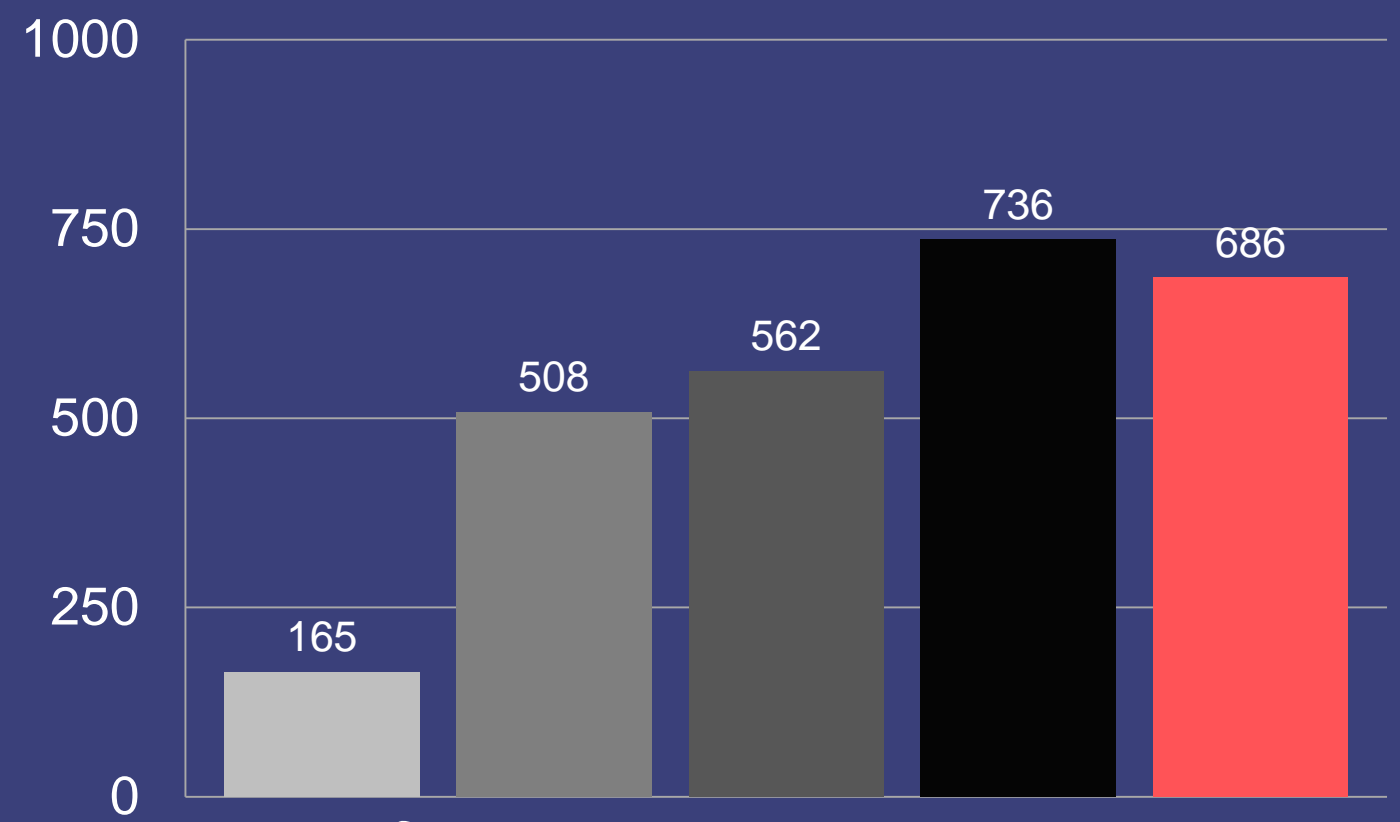
- Automatically generate DBMS knob configurations.
- Reuse data from previous tuning sessions.

OtterTune

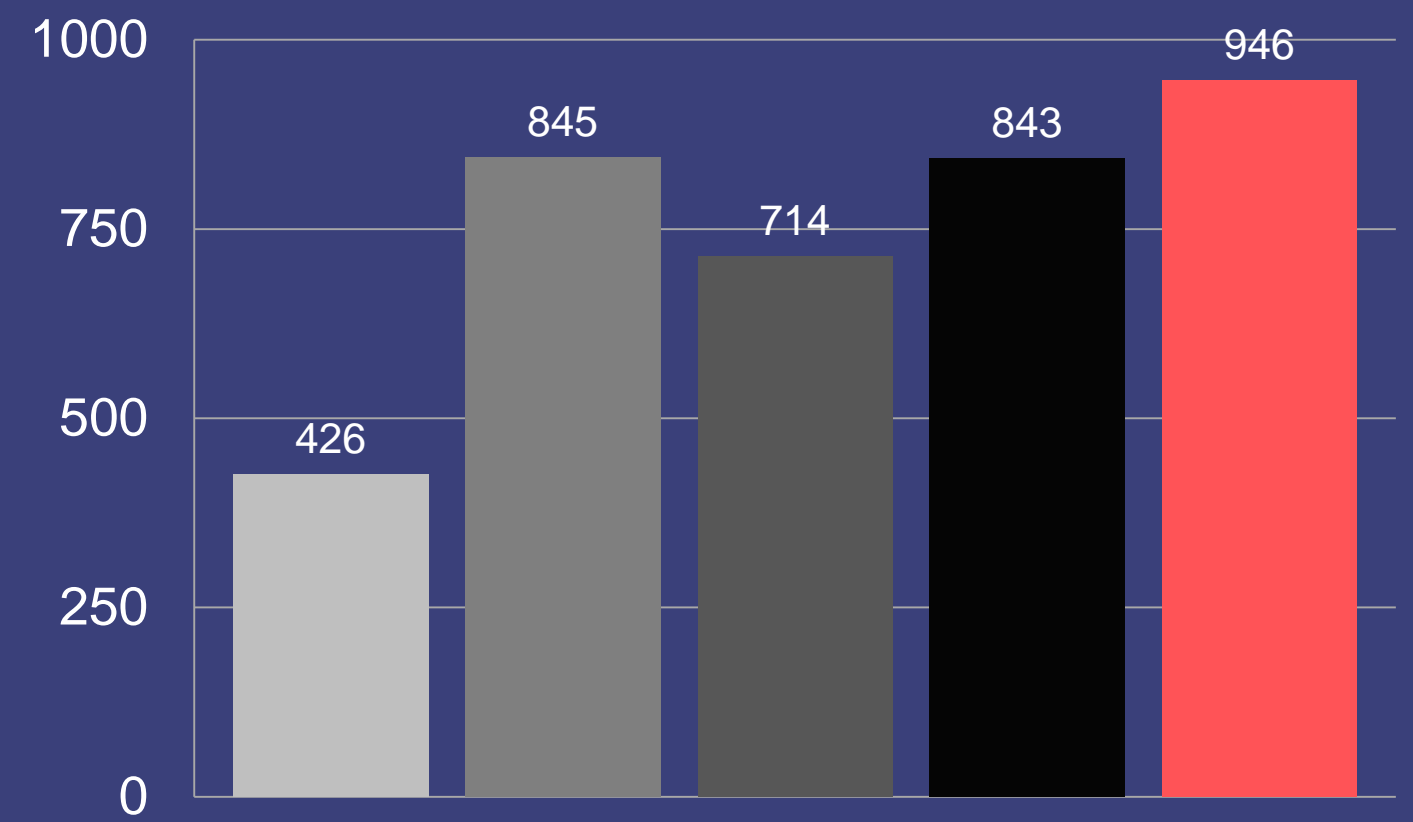
ottertune.cs.cmu.edu

Default Scripts RDS DBA OtterTune

Throughput (txn/sec)



MySQL



PostgreSQL

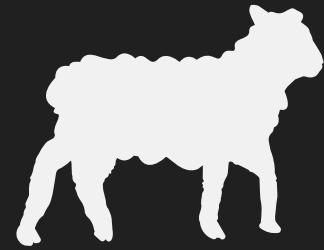


Peloton
pelotondb.io

Self-Driving Database System

→ In-memory DBMS with integrated planning framework.

→ Designed for autonomous operations.



Design Considerations for **Autonomous** Operation



Configuration
Knobs



Internal
Metrics



Action
Engineering



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

- *File Paths / Network Info*
- *Durability / Isolation Levels*
- *Hardware Usage*
- *Recovery Time*



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

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





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

Bad Example:



RocksDB



RocksDB Column Family Knobs

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

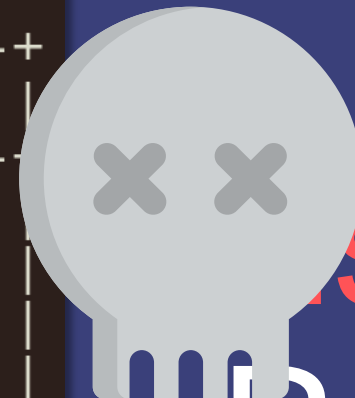
Global Metrics

```
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

```
STAT;
```

VALUE
1
21672
21672
0
0
18
0
0
2



Aggregated Metrics
Missing: Reads Writes

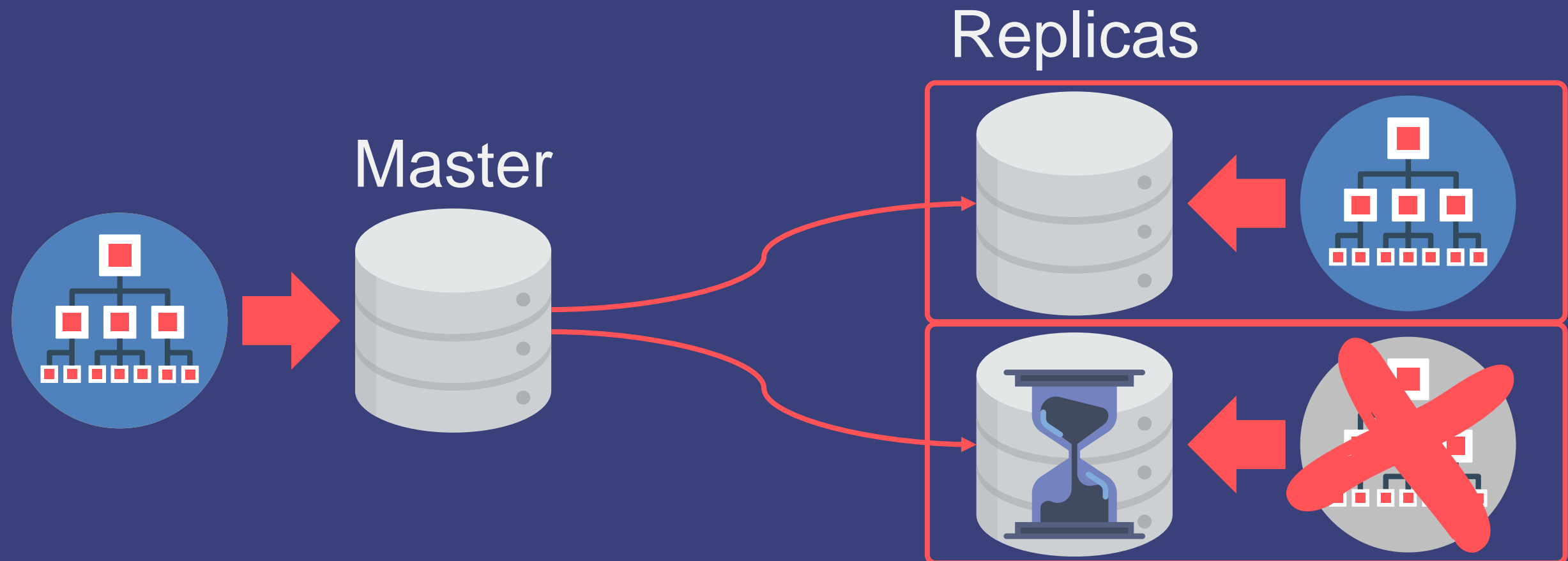


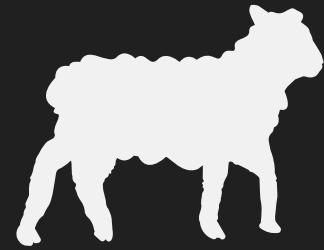
No action should ever require the DBMS to restart in order for it to take affect.

The commercial systems are much better than this than the open-source systems.



Allow replica configurations to diverge from each other.





What About **Oracle's**
Self-Driving DBMS?

Self-Driving DBMS?

From: Andy Pavlo <pavlo@cs.cmu.edu>
To: Larry Ellison <larry.ellison@oracle.com>
Date: 9/15/17 11:17 PM

LE - I saw your announcement about Oracle putting out the "self-driving" DBMS. What's up with that? You know that my squad been working on our self-driving DBMS for the last two years:

<http://pelotondb.io>

How can you do this to me after all that we've been through together? This is like that time we were together in Guatemala back in 1999. Do you remember when you asked me whether I had a spare condom? I gave my last one to you and then I found out the next night that you had a whole box of them in your suitcase. You told me that

Management Systems

bin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Siddharth Santurkar, Anthony Tomasic, Yijun Wu*, Ran Xian, Tieying Zhang
 National University of Singapore

Much of the previous work on self-driving databases focuses on standalone tools that target only a single aspect of database management. For example, some tools are able to automatically generate the physical design of a database [16], while others focus on partitioning schemes [6, 44], data distribution [17], or query optimization [5]. Other tools are able to select optimal 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) unable to take a holistic view that considers more than one problem at a time. That is, they observe the DBMS's behavior from outside the system and advise the DBA on how to make corrections to 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 a self-driving system. We then present the architecture of **Peloton** [1], the 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.

PROBLEM OVERVIEW

The first challenge in a self-driving DBMS is to understand an application's workload. The most basic level is to characterize the workload 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 point queries. If it is OLAP, then the DBMS should use a column-oriented

2017



True autonomous DBMSs are achievable in the next decade.

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

EN

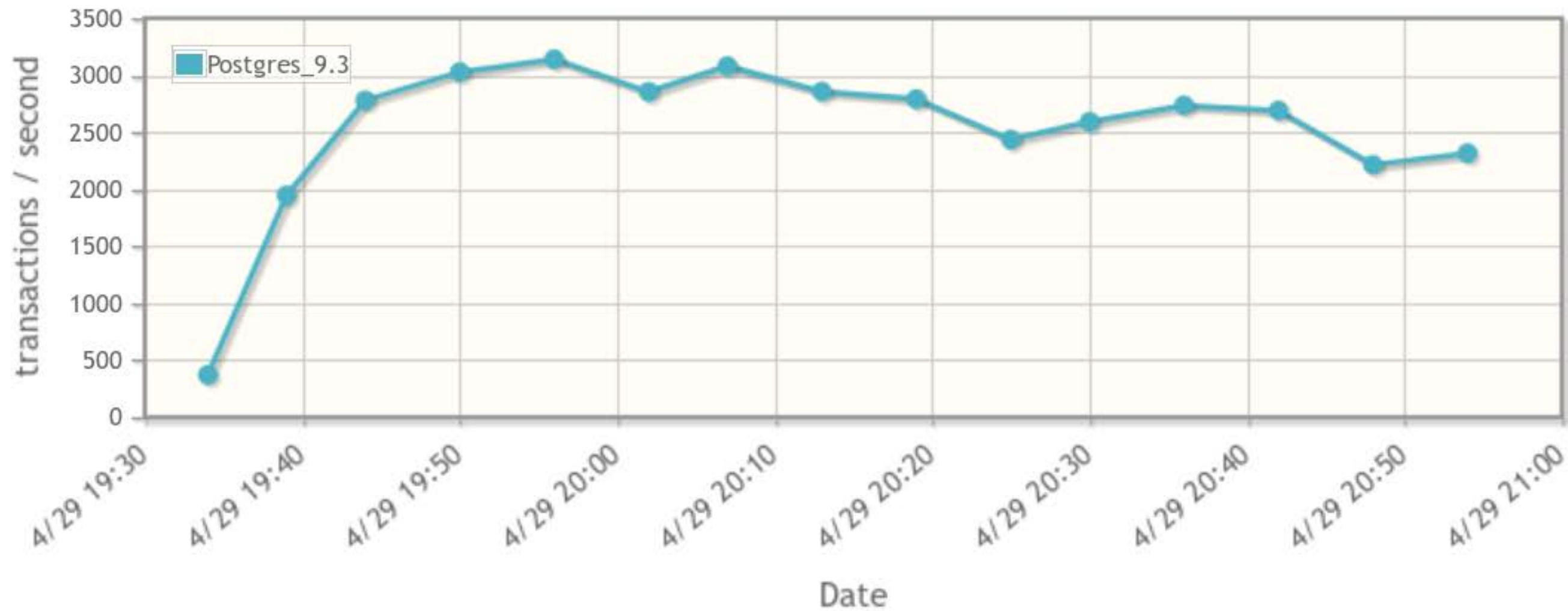
@andy_pavlo

D

Show the last 100 results

Eq

Throughput (more is better)



_fsync



CONTROLLER

COLLECTOR

INSTALL AGENT



TARGET DATABASE

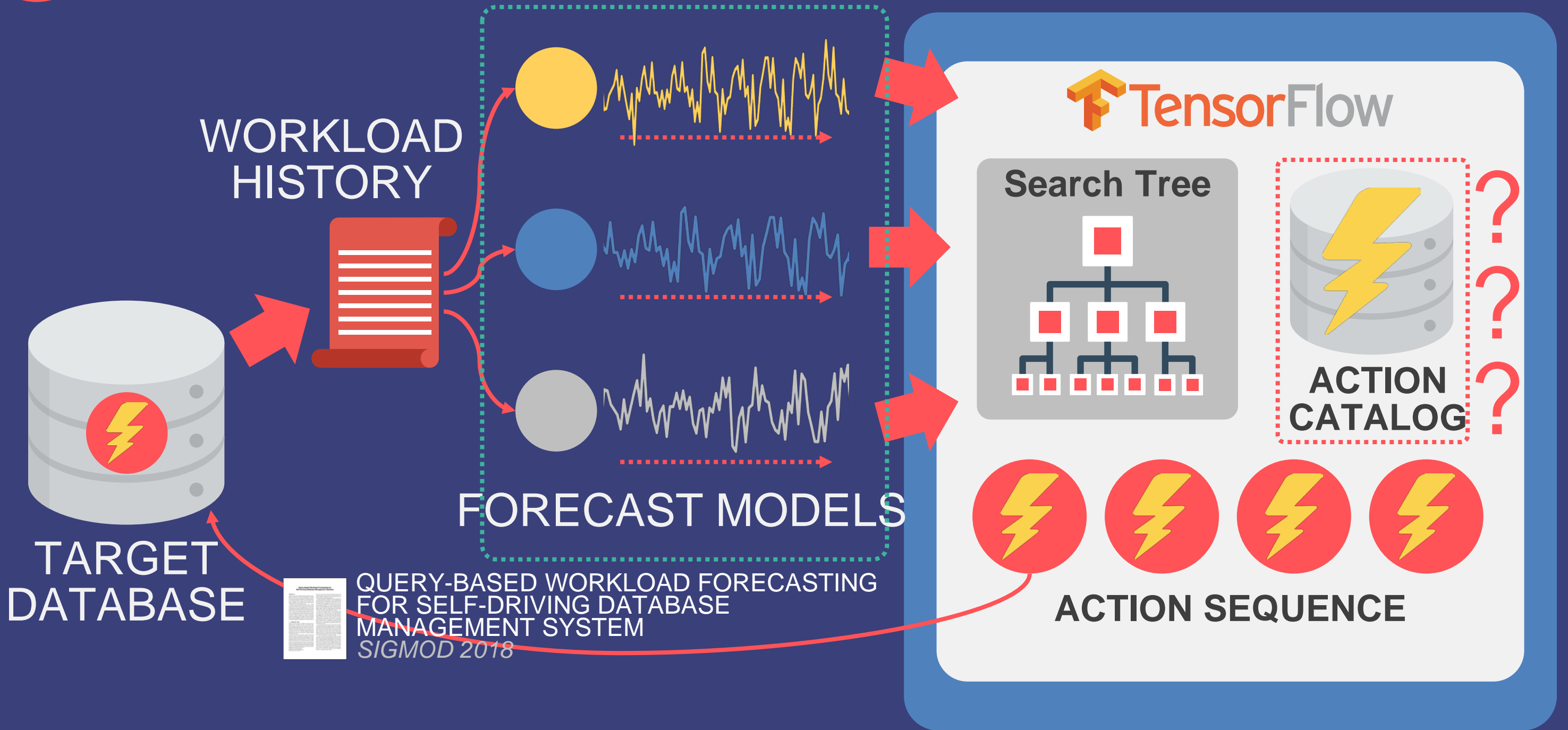


```
mysql> SHOW GLOBAL STATUS;
```

METRIC_NAME	VALUE
ABORTED_CLIENTS	0
ABORTED_CONNECTS	0
...	
INNODB_BUFFER_POOL_BYTES_DATA	129499136
INNODB_BUFFER_POOL_BYTES_DIRTY	76070912
INNODB_BUFFER_POOL_PAGES_DATA	7904
INNODB_BUFFER_POOL_PAGES_DIRTY	4643
INNODB_BUFFER_POOL_PAGES_FLUSHED	25246
INNODB_BUFFER_POOL_PAGES_FREE	0
INNODB_BUFFER_POOL_PAGES_MISC	288
INNODB_BUFFER_POOL_PAGES_TOTAL	8192
INNODB_BUFFER_POOL_READS	15327
INNODB_BUFFER_POOL_READ_AHEAD	0
INNODB_BUFFER_POOL_READ_AHEAD_EVICT	0
INNODB_BUFFER_POOL_READ_AHEAD_RND	0
INNODB_BUFFER_POOL_READ_REQUESTS	2604302
INNODB_BUFFER_POOL_WAIT_FREE	0
INNODB_BUFFER_POOL_WRITE_REQUESTS	562763
INNODB_DATA_FSYNCS	2836
INNODB_DATA_PENDING_FSYNCS	1
INNODB_DATA_WRITES	28026
...	
UPTIME	5996
UPTIME_SINCE_FLUSH_STATUS	5996



"THE BRAIN"

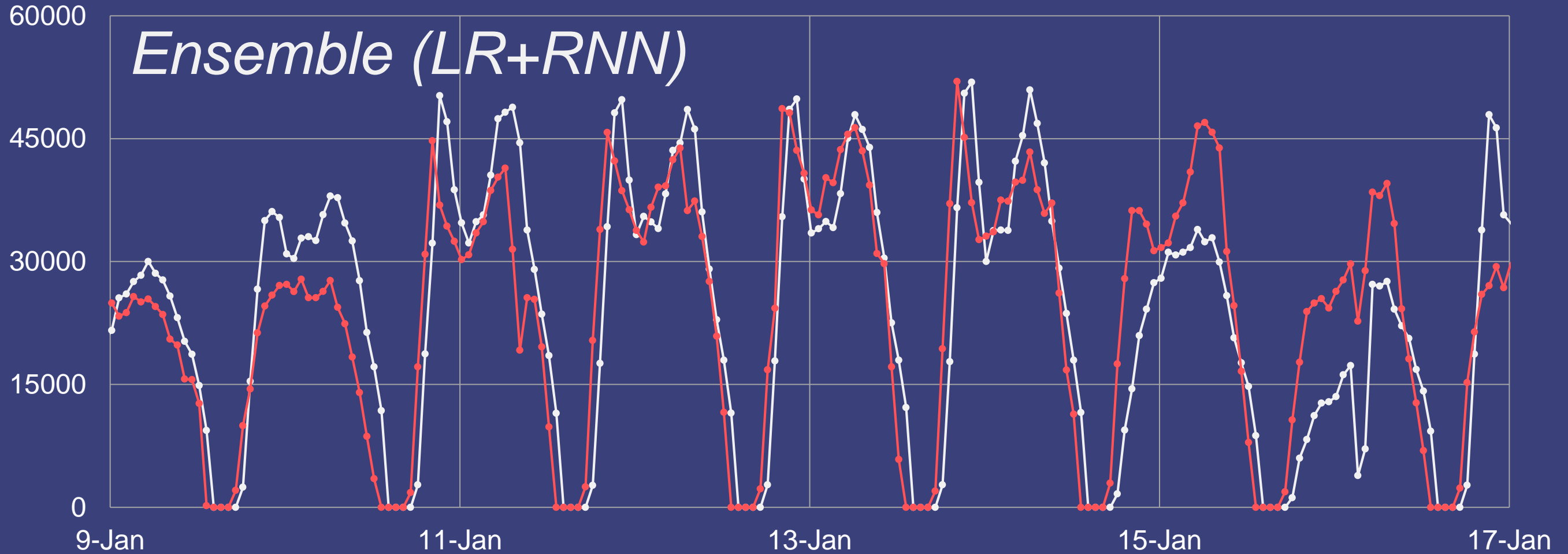


QUERY-BASED WORKLOAD FORECASTING
FOR SELF-DRIVING DATABASE
MANAGEMENT SYSTEM
SIGMOD 2018



Actual Predicted

Queries Per Hour





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

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

Thank you!

