CS 6530: Advanced Database Systems Fall 2022

Lecture 16
ML for Databases

Prashant Pandey

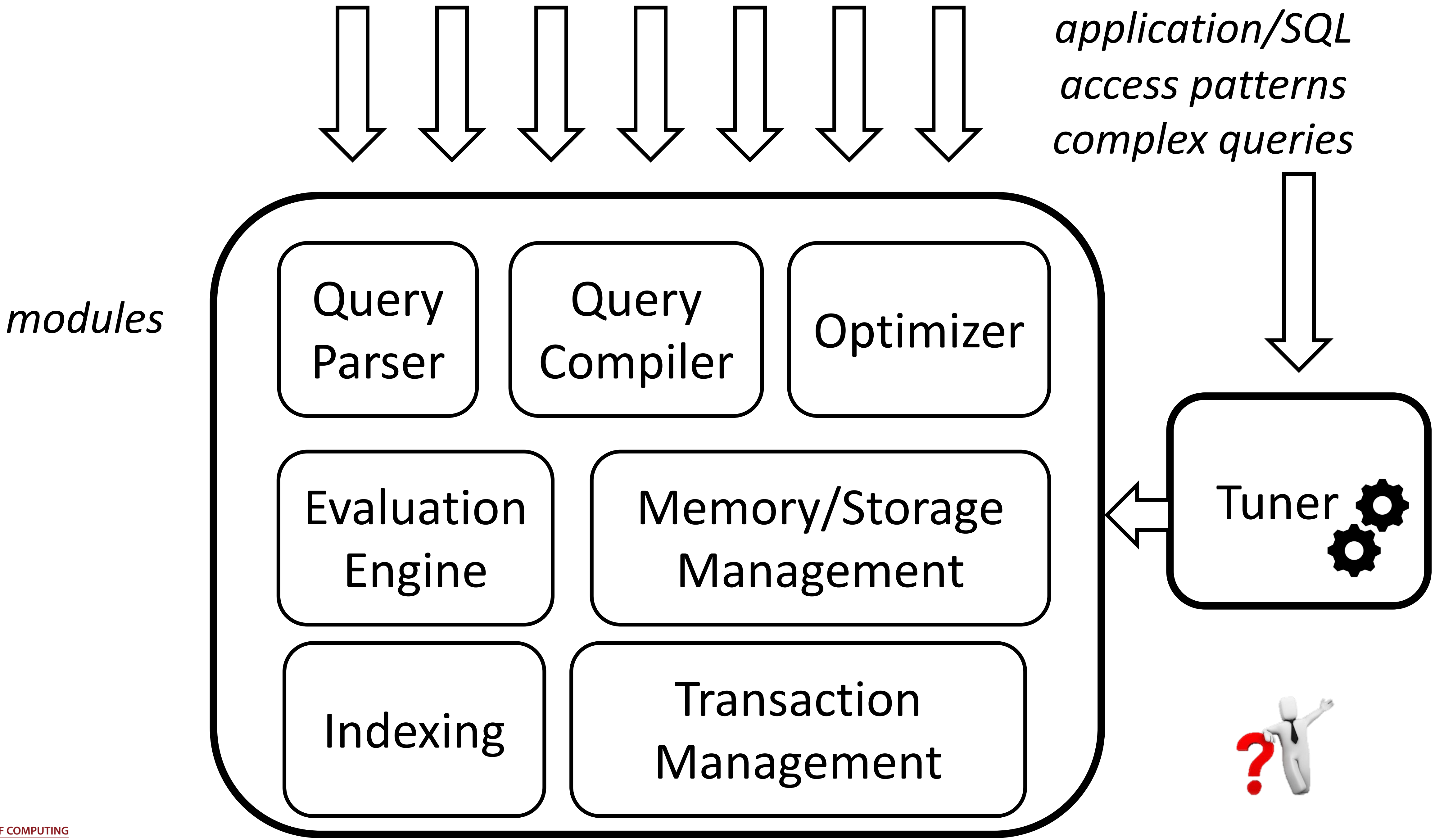
prashant.pandey@utah.edu

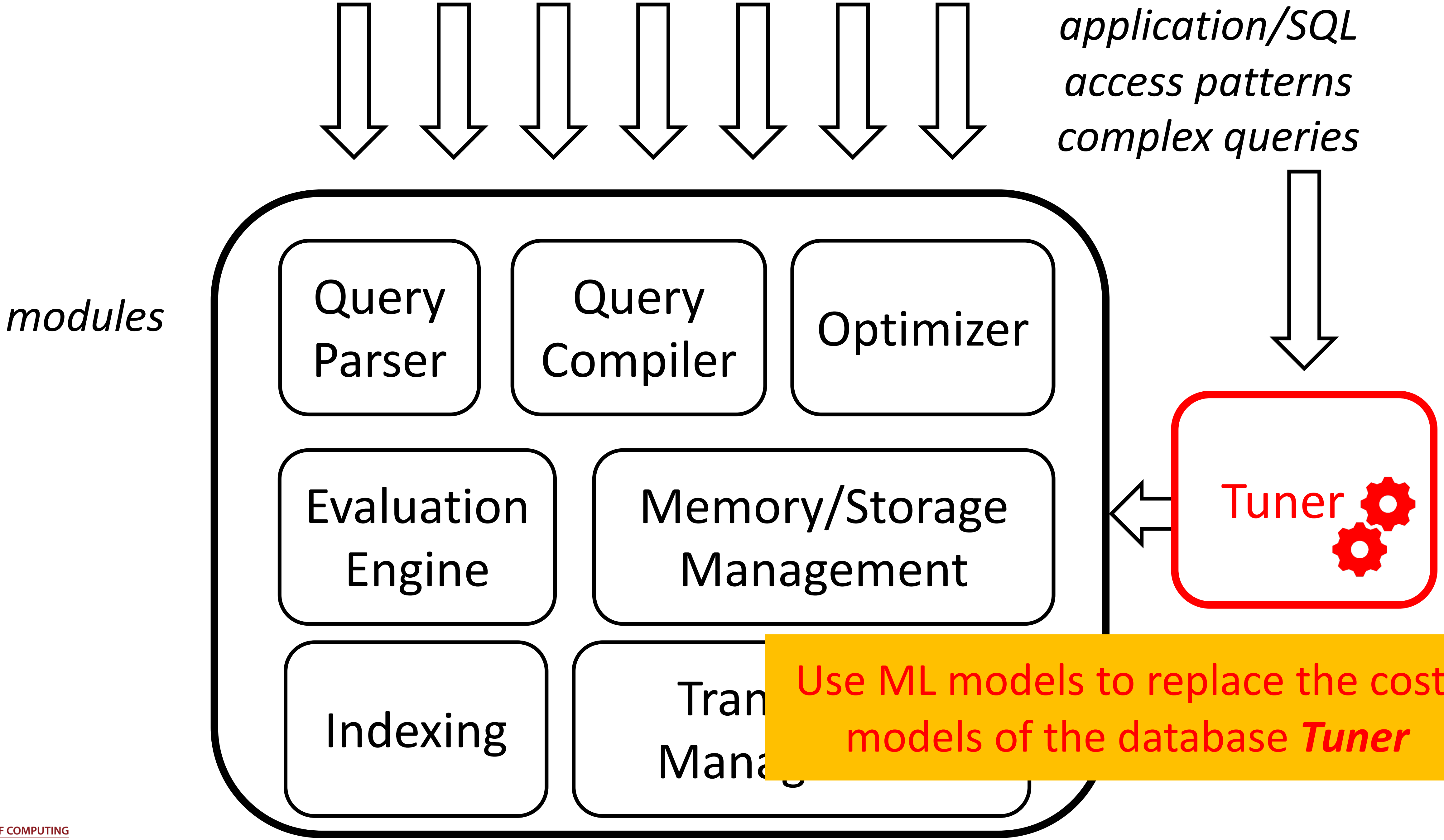
Acknowledgement: Slides taken from Prof. Manos Athanassoulis, BU

Machine learning algorithms improve *automatically* through *experience* and by the use of ***data***.

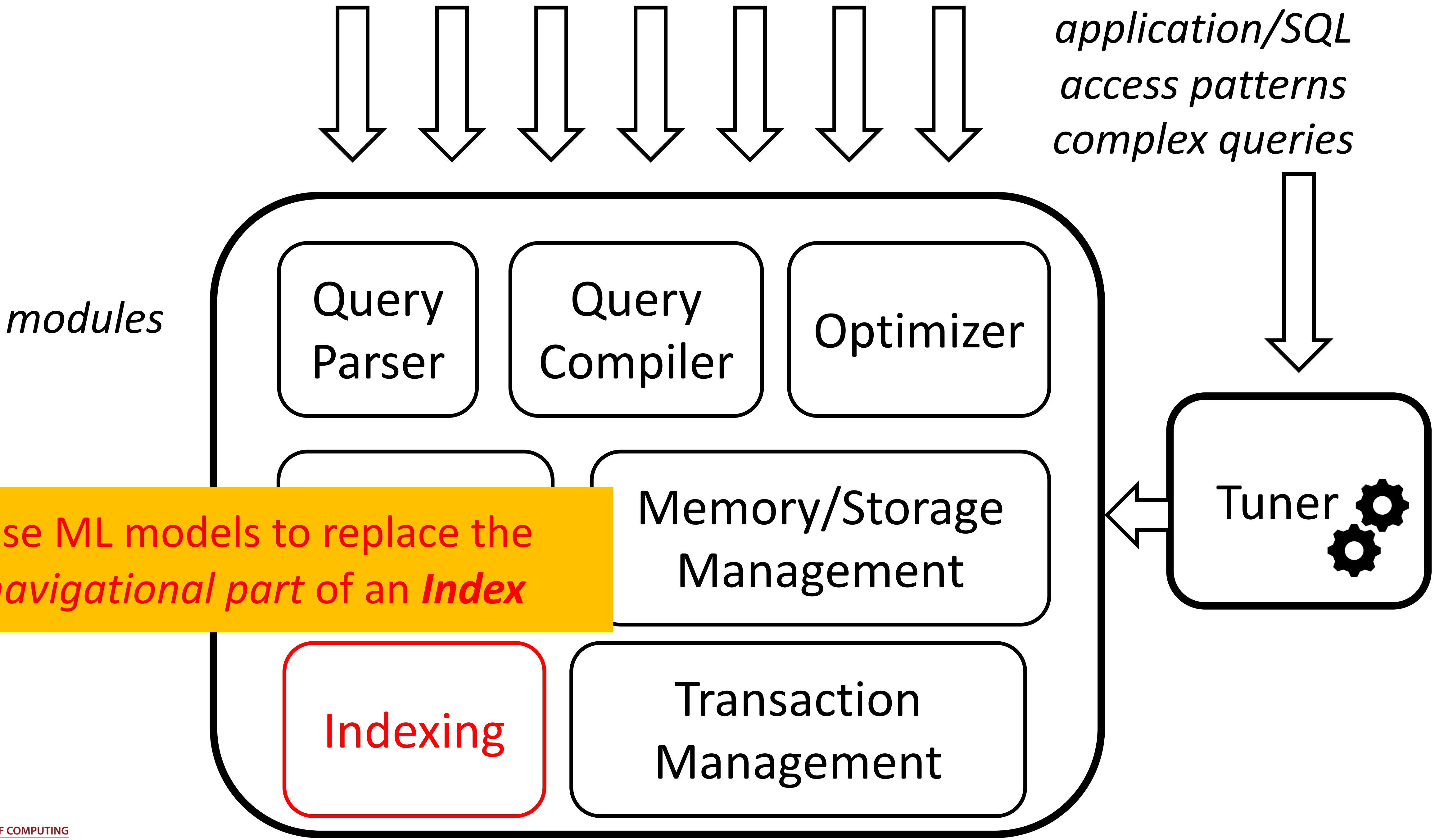
Machine learning algorithms build a model based on ***training data***, in order to make ***predictions*** or ***decisions*** without being explicitly programmed to do so.

Which database systems components can benefit/be replaced by ML algorithms?

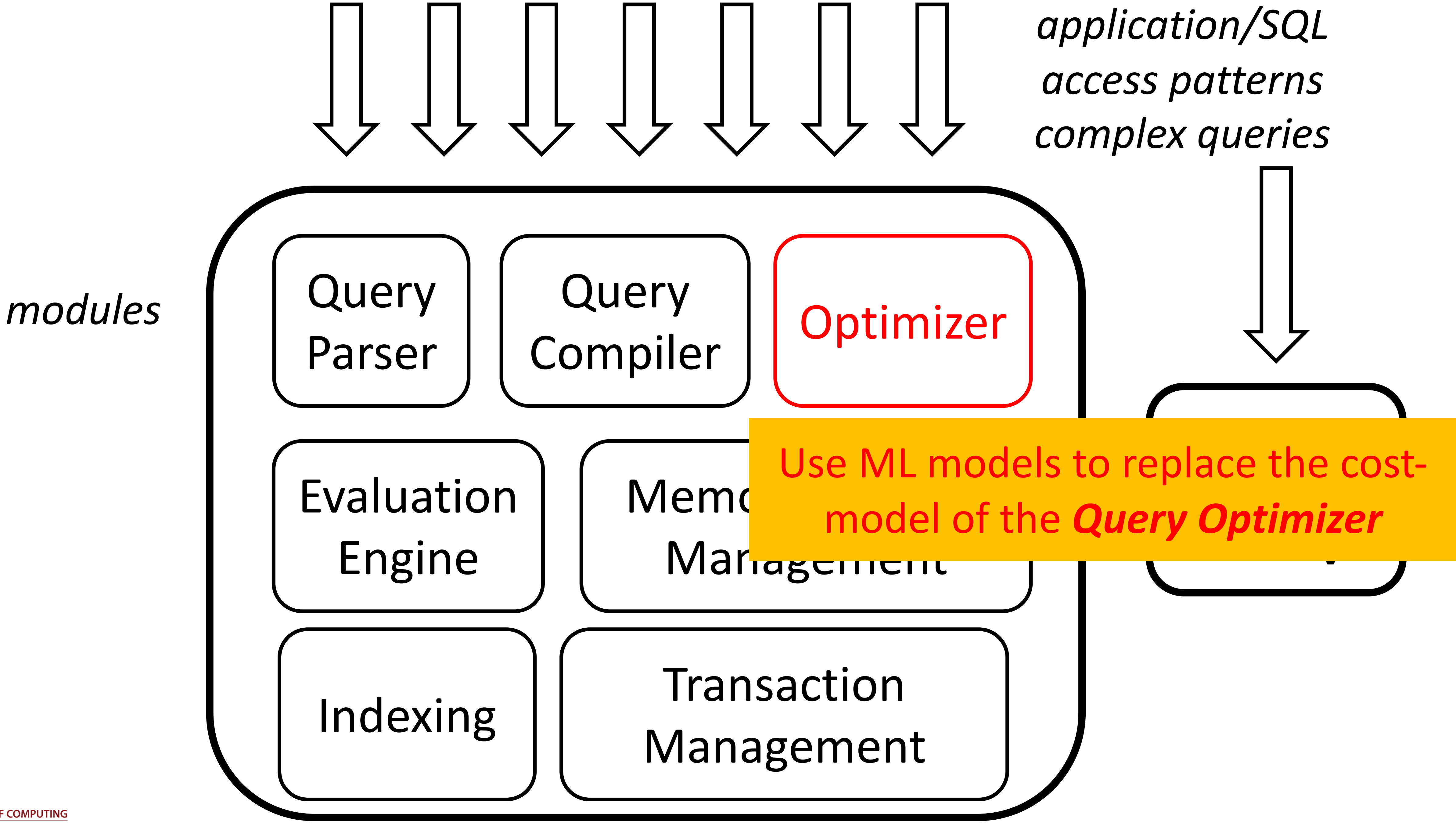

application/SQL
access patterns
complex queries
modules
Query Parser
Query Compiler
Optimizer
Evaluation Engine
Memory/Storage Management
Indexing
Transaction Management
Tuner

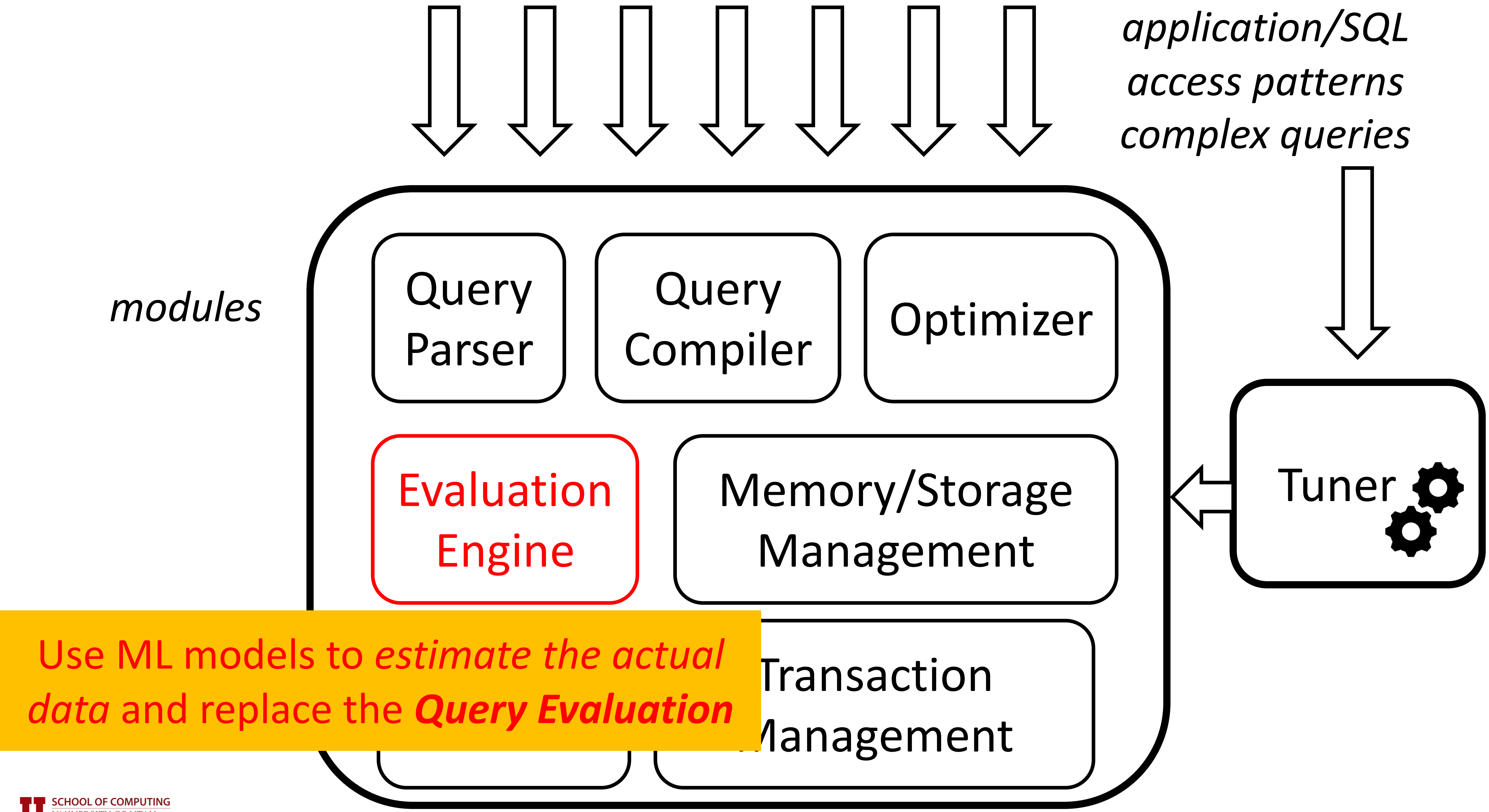


Use ML models to replace the cost-models of the database ***Tuner***



Use ML models to replace the *navigational part* of an ***Index***


application/SQL
access patterns
complex queries
modules
Query Parser
Query Compiler
Optimizer
Evaluation Engine
Memory Management
Indexing
Transaction Management
Use ML models to replace the cost-model of the *Query Optimizer*


application/SQL
access patterns
complex queries
modules
Query
Parser
Query
Compiler
Optimizer
Evaluation
Engine
Memory/Storage
Management
Transaction
Management
Tuner
Use ML models to *estimate the actual data* and replace the ***Query Evaluation***

Self-driving Data systems

Types of actions that a self-driving system needs to take ***automatically***

	Types	Actions
PHYSICAL	Indexes	AddIndex, DropIndex, Rebuild, Convert
	Materialized Views	AddMatView, DropMatView
	Storage Layout	Row→Columnar, Columnar→Row, Compress
DATA	Location	MoveUpTier, MoveDownTier, Migrate
	Partitioning	RepartitionTable, ReplicateTable
RUNTIME	Resources	AddNode, RemoveNode
	Configuration Tuning	IncrementKnob, DecrementKnob, SetKnob
	Query Optimizations	CostModelTune, Compilation, Prefetch

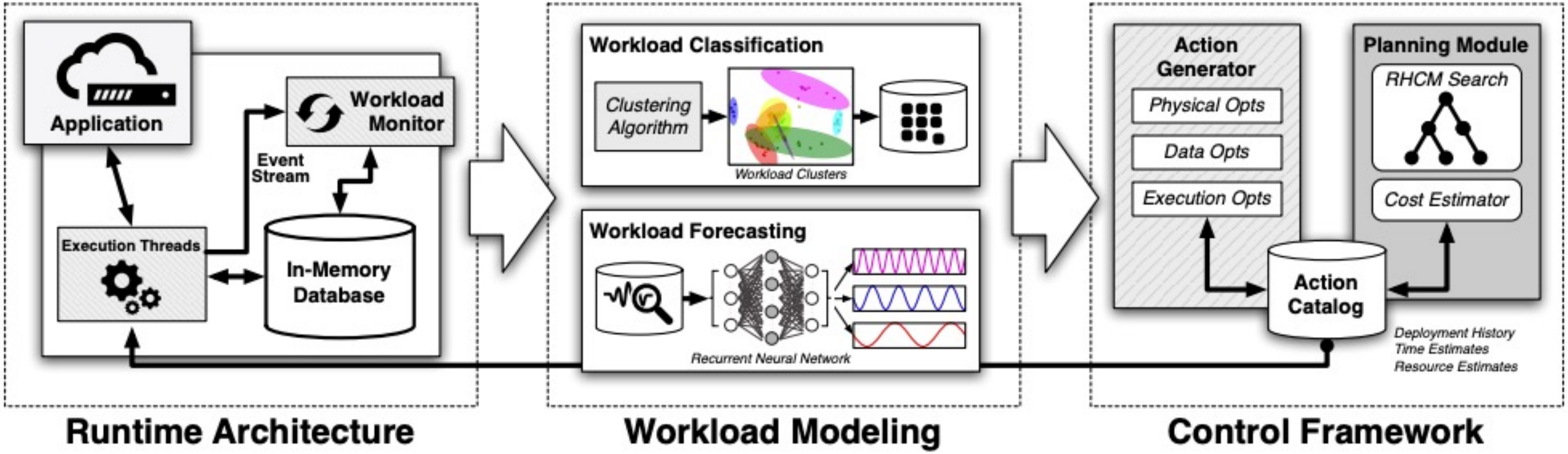
Use-case: Peloton Self-Driving Architecture

(A) Application

(B) Workload Monitoring

(C) Workload Classification
[unsupervised learning to group similar queries]

(E) Action Planning
[use tools like *receding-horizon control model* to select actions that might lead to better performance in the future]

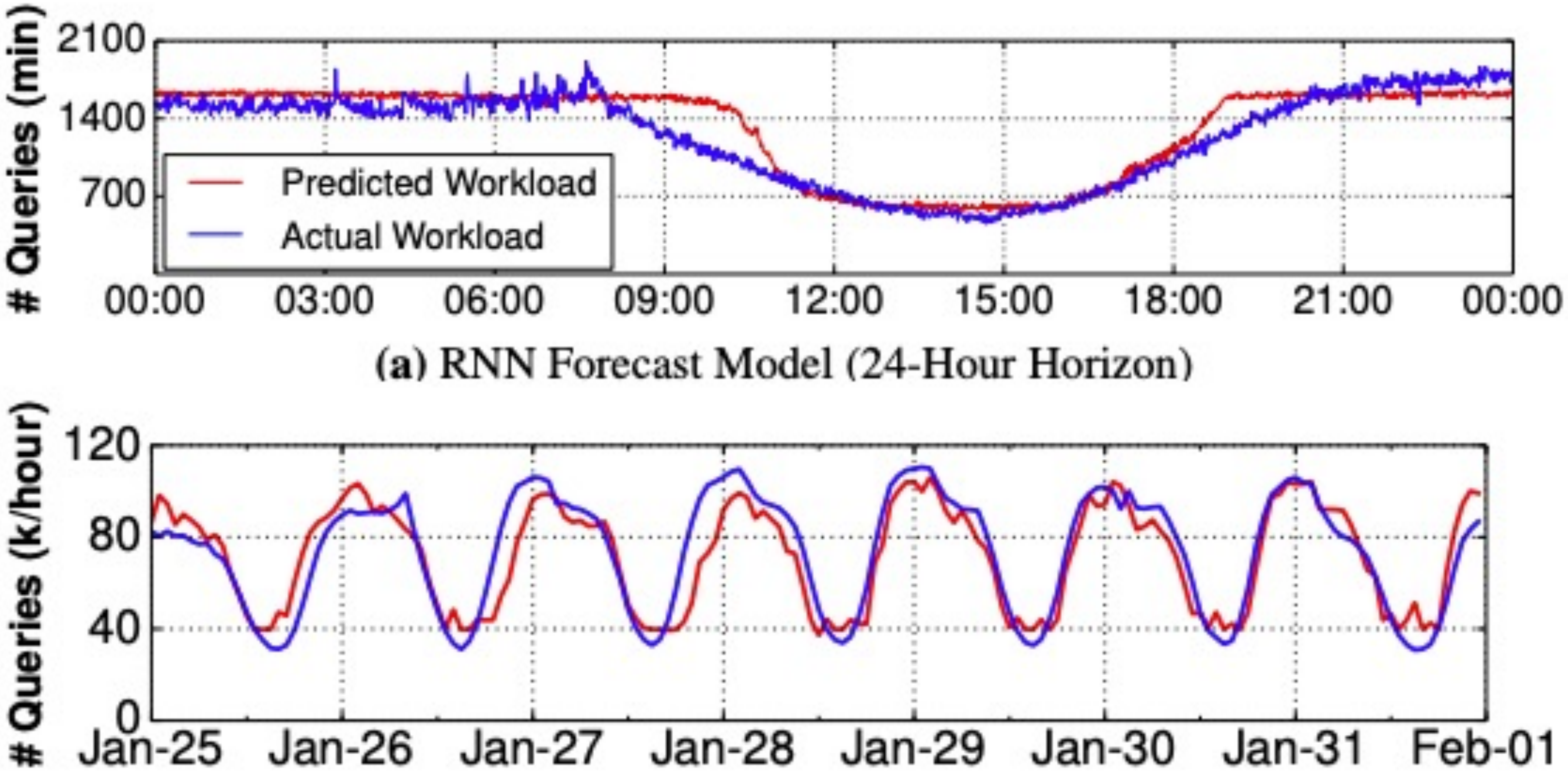


(D) Workload Forecasting
[predict future workload to autoscale cloud instances]

(F) Action Generator
[select action and log them, reversals may also happen]

Workload forecasting

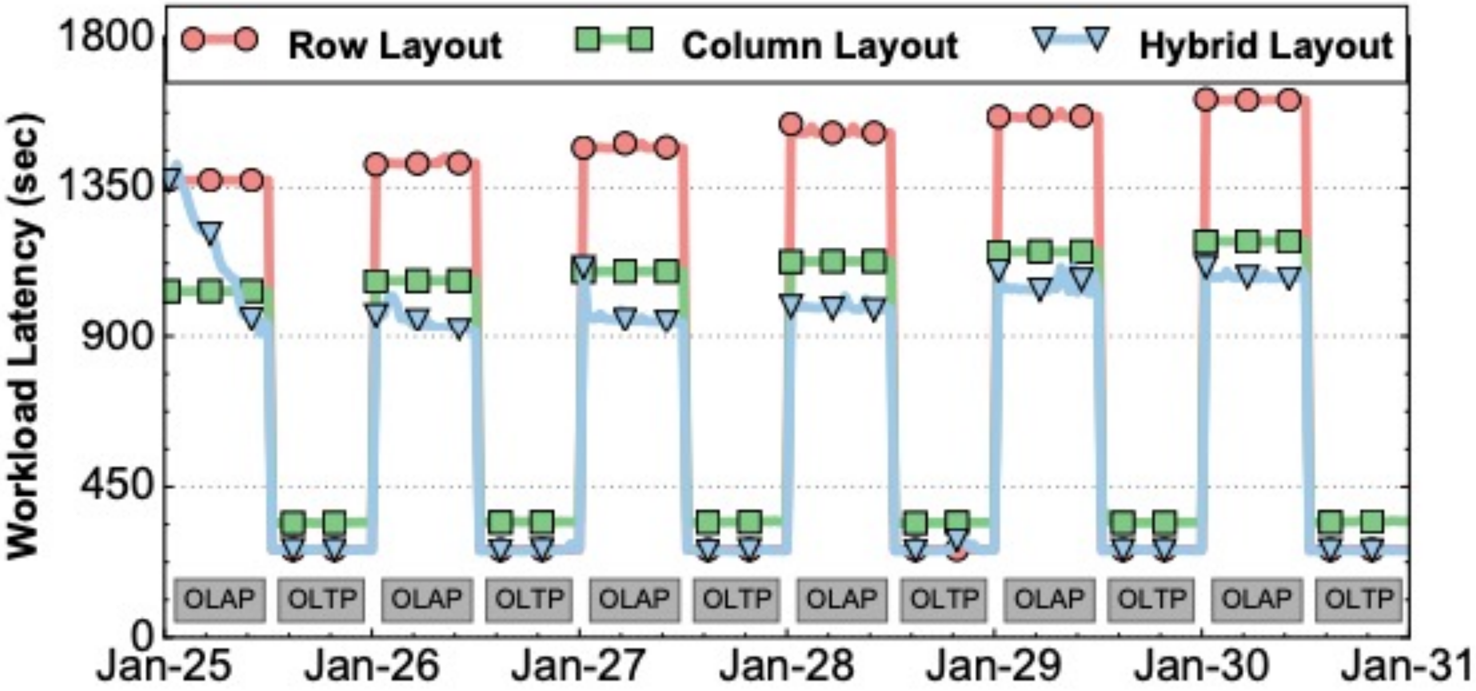
Using Recurrent Neural Networks (RNN) the model learns patterns and adapts to changes



(a) RNN Forecast Model (24-Hour Horizon)

(b) RNN Forecast Model (7-Day Horizon)

Action example: adapting the storage layout

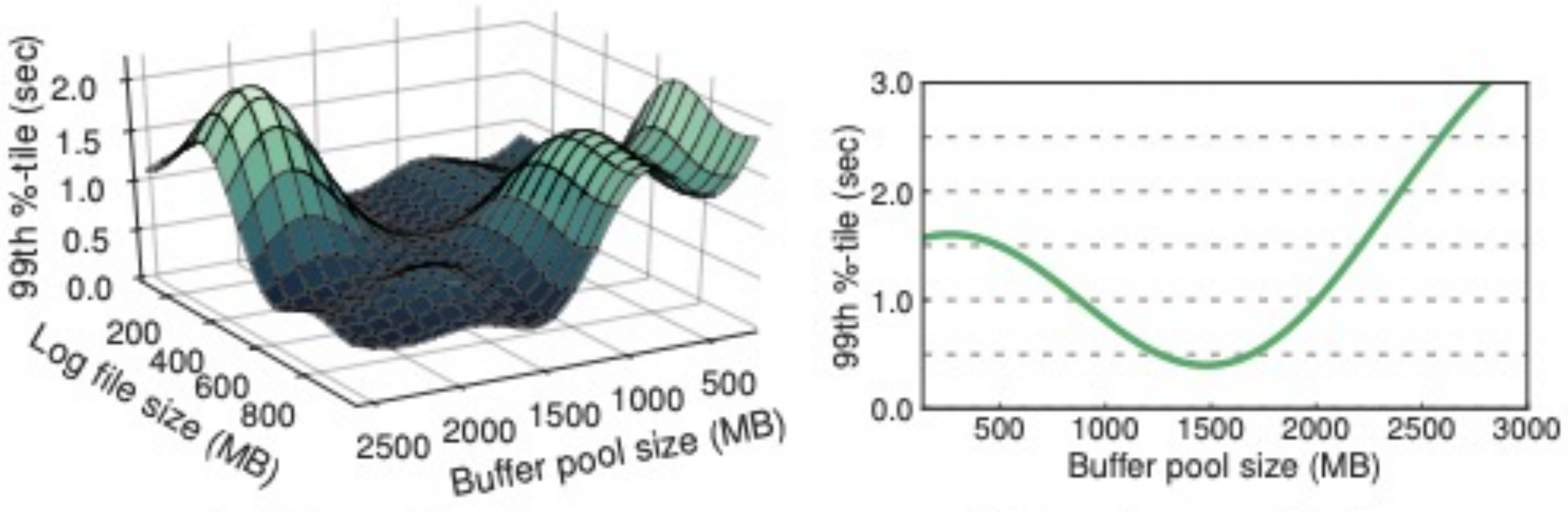


Columns are better for OLAP

Rows are better for OLTP

Hybrid matches the best when workload alternates

Why automatic tuning is hard? (1/2)



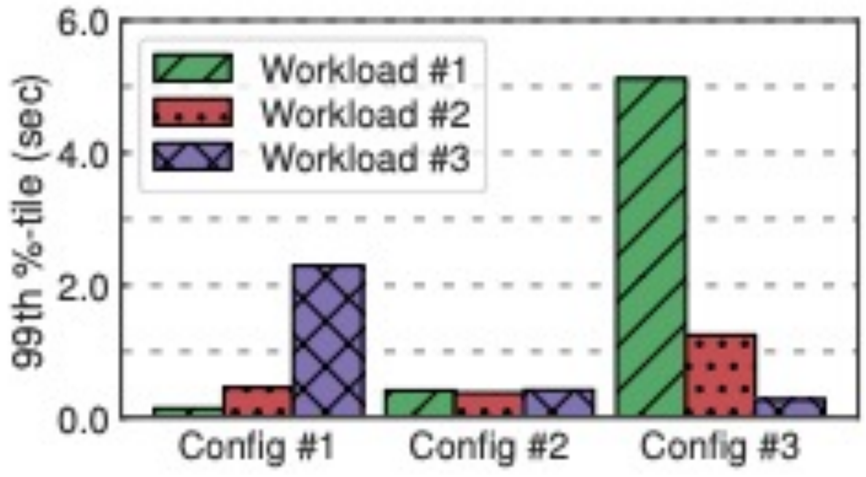
(a) Dependencies

(b) Continuous Settings

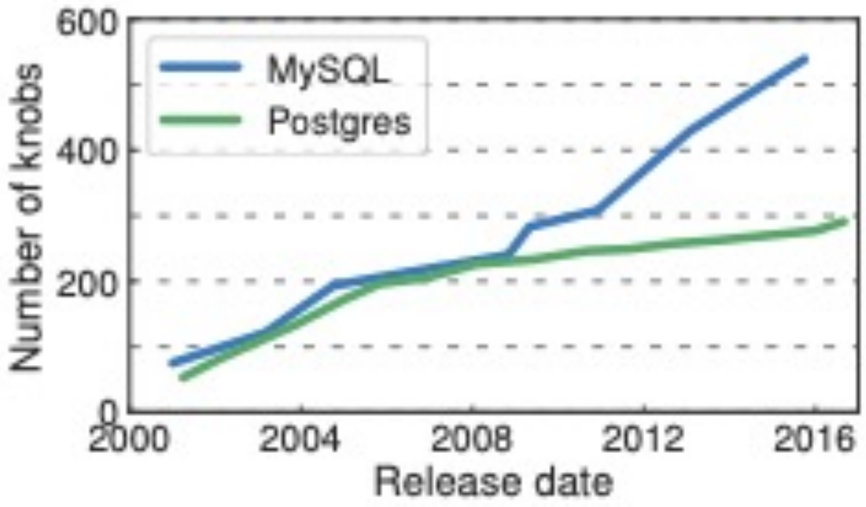
Complex interdependencies between different tuning knobs!

Continuous domain ("too many" knob options) with irregular benefits

Why automatic tuning is hard? (2/2)

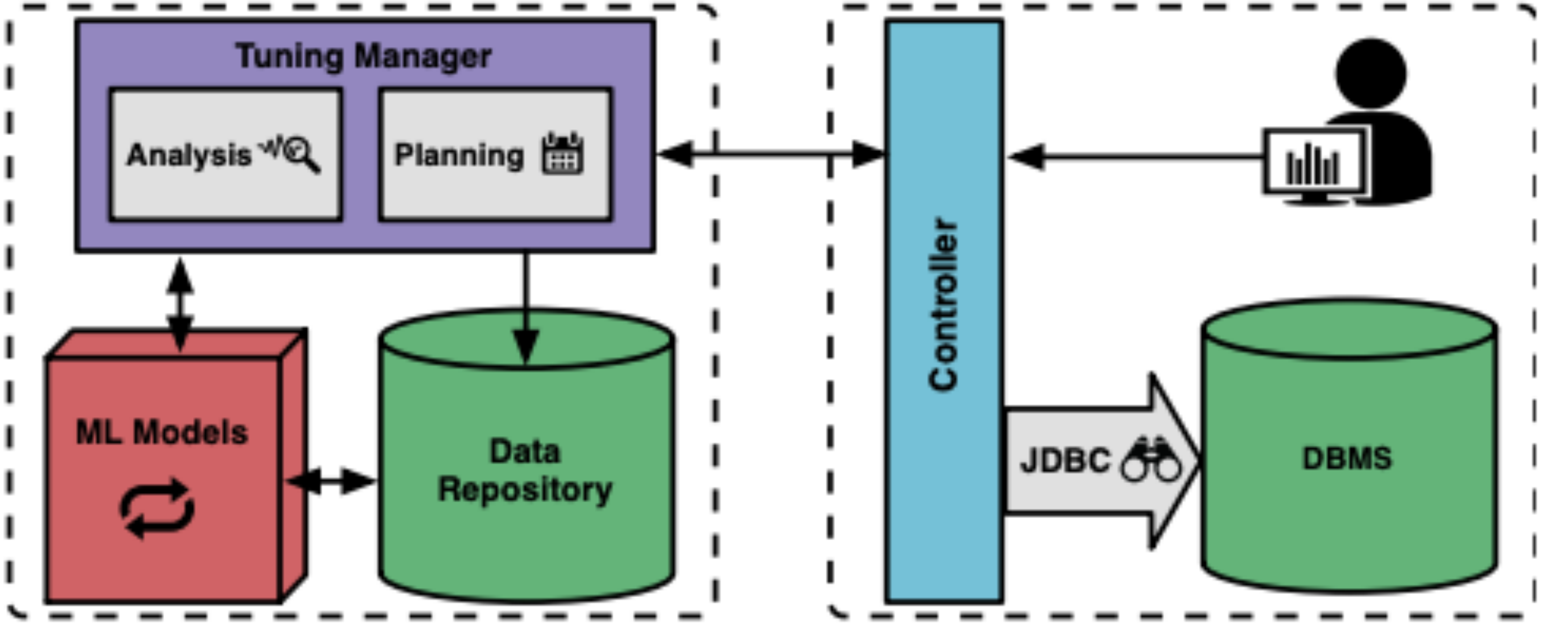


Non-reusable configurations!



Increasing tuning complexity

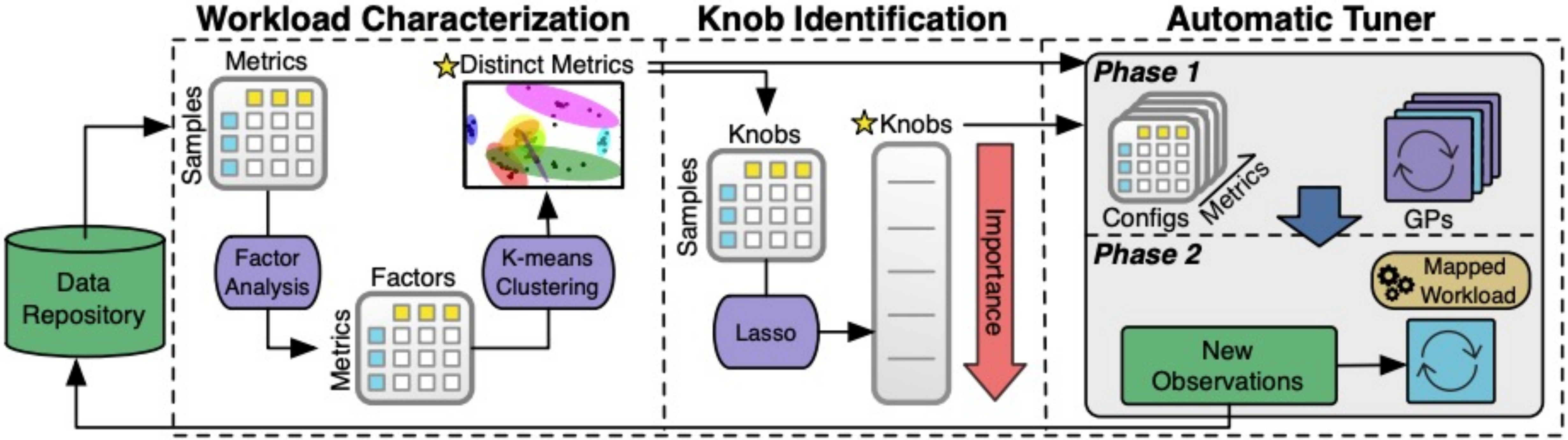
Use case: Ottertune



Two distinct components: the tuning manager **does not have access to data**, only to **performance metrics** and the values of the **tuning knobs**

All performance data are organized per system and per major version to ensure that no wrong, deprecated, or non-existing knobs are tuned.

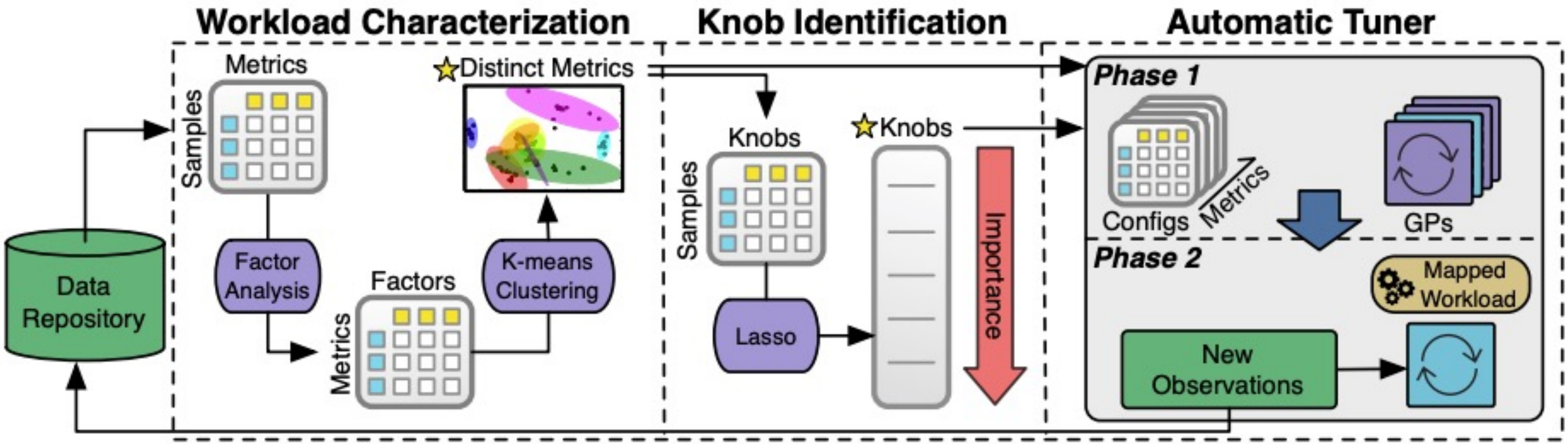
OtterTune Machine Learning Pipeline



How to classify/characterize a workload?

A workload is characterized based on the system metrics when it is executed (e.g., #pages reads/writes, cache utilization, locking overhead)

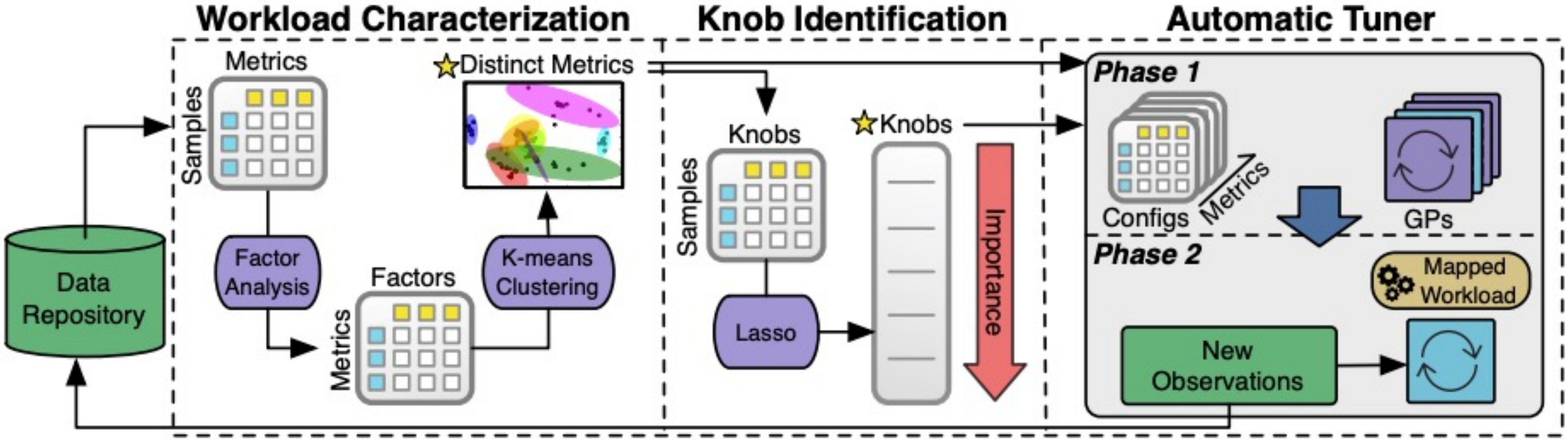
OtterTune Machine Learning Pipeline



Collect statistics at the global level (system-wide), per table proves to be challenging for various systems

Prune redundant metrics (e.g., data read and pages read are directly linked) via factor analysis and k-means clustering

OtterTune Machine Learning Pipeline



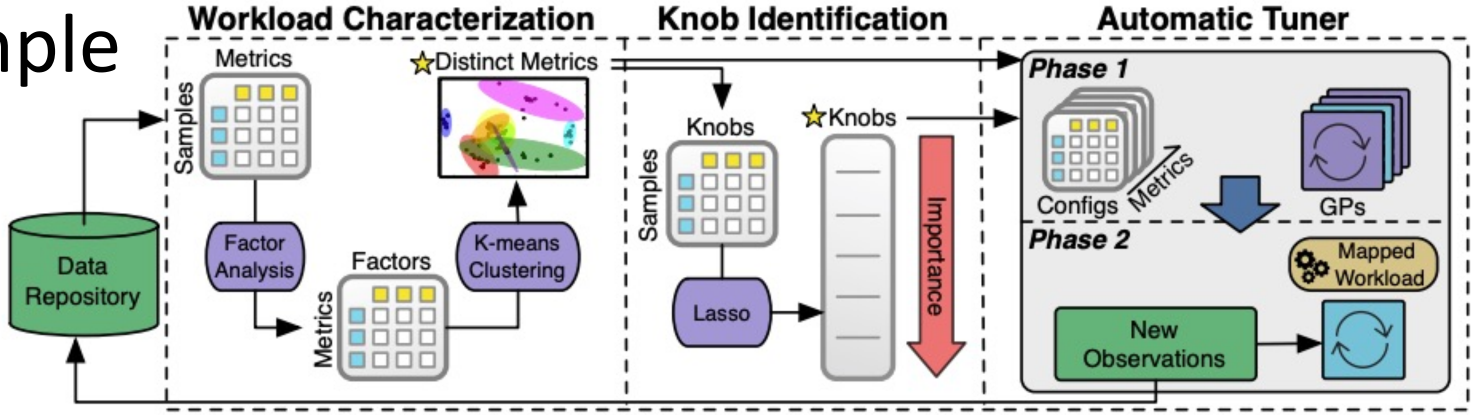
Identify important knobs

Order the knobs based on their significance on the system's performance (and identify knobs interdependencies)

Store in a repository observations

OtterTune Machine Learning Pipeline

Automated Tuning: an Example



Use the systems metrics to identify (classify) the workload

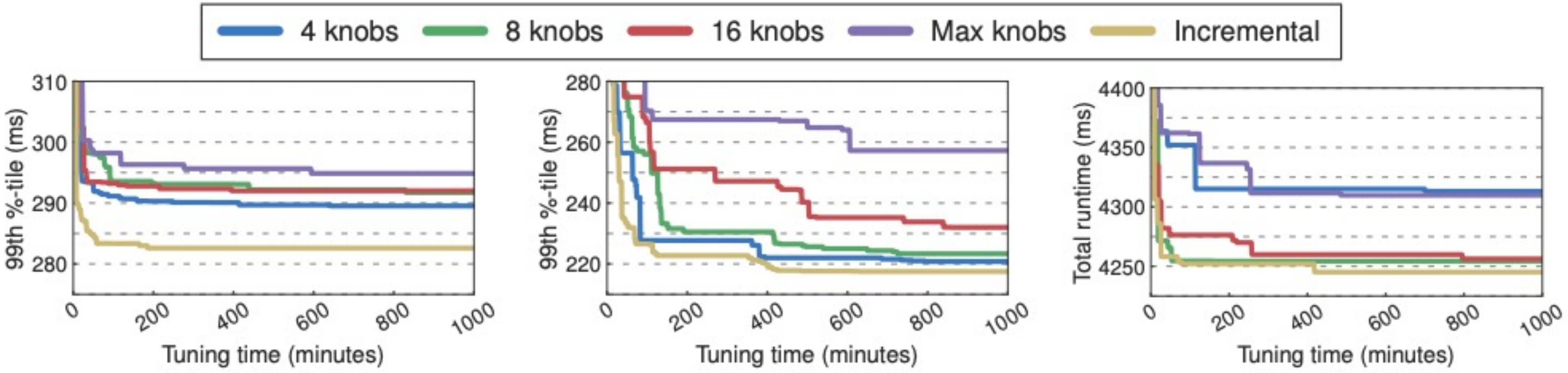
Iterative configuration recommendation balancing ***exploration*** vs. ***exploitation***

Exploration: try out a configuration for which there is not enough data in the repository
this is done when (i) there is not enough data for this workload (so more data are needed), or
(ii) the system decides to try out new configurations that help collect more data in general

Exploitation: the system uses small variations of a configuration that is close to optimal using the existing data

OtterTune in Action

Start by sweeping values of knobs to collect "training data"



(a) MySQL (TPC-C) (b) Postgres (TPC-C) (c) Vector (TPC-H)

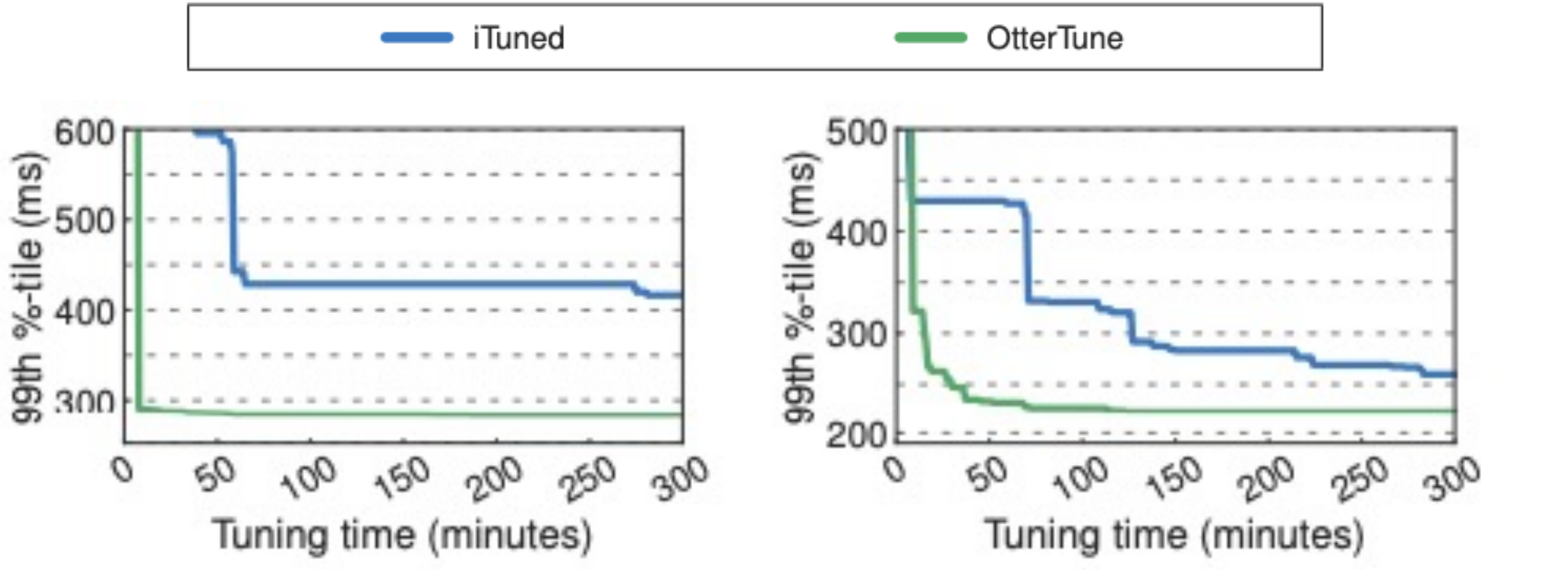
The optimal number of knobs varies per *DBMS* **and** *workload*!

Increasing the number of knobs gradually is the best approach, because it balances complexity and performance.

OtterTune tunes MySQL and Postgres that have few impactful knobs, and Actian Vector that requires more knobs to be tuned in order to achieve good performance.

OtterTune vs iTuned on TPCC

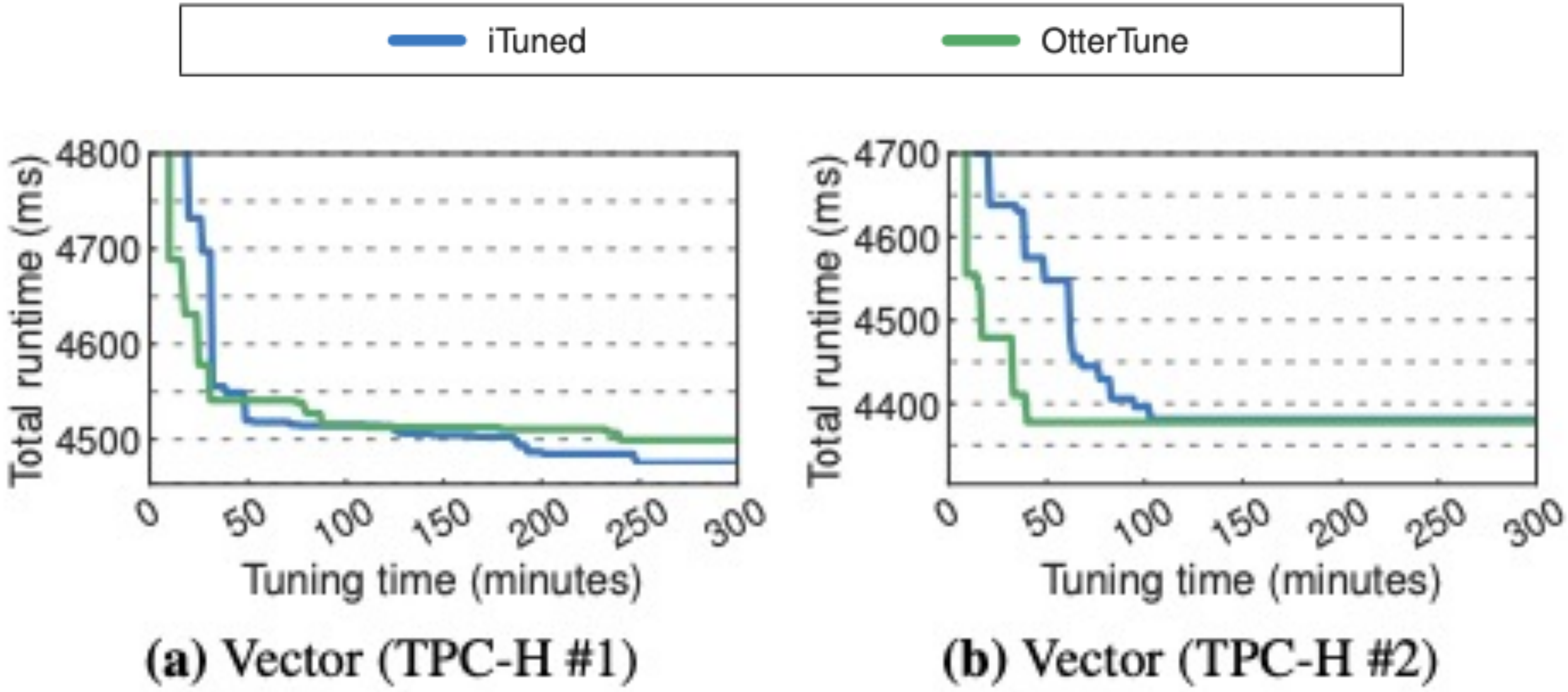
iTuned uses an initial set of 10 DBMS configurations at the beginning of the tuning session.



(a) MySQL (b) Postgres

OtterTune is trained with more data, so it can achieve a better end result!

OtterTune vs iTuned on TPCH

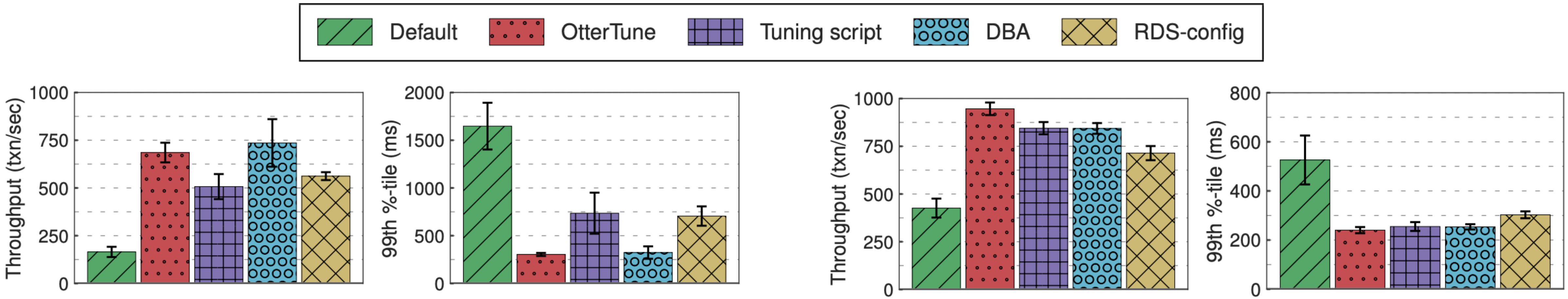


Actian Vector allows fewer “bad” options, so the training is easier.

“A tuning knob is a database engineer not knowing what do”

take this with a grain of salt!

OtterTune Efficacy Comparison



(a) TPC-C (Throughput) **(b)** TPC-C (99%-tile Latency)

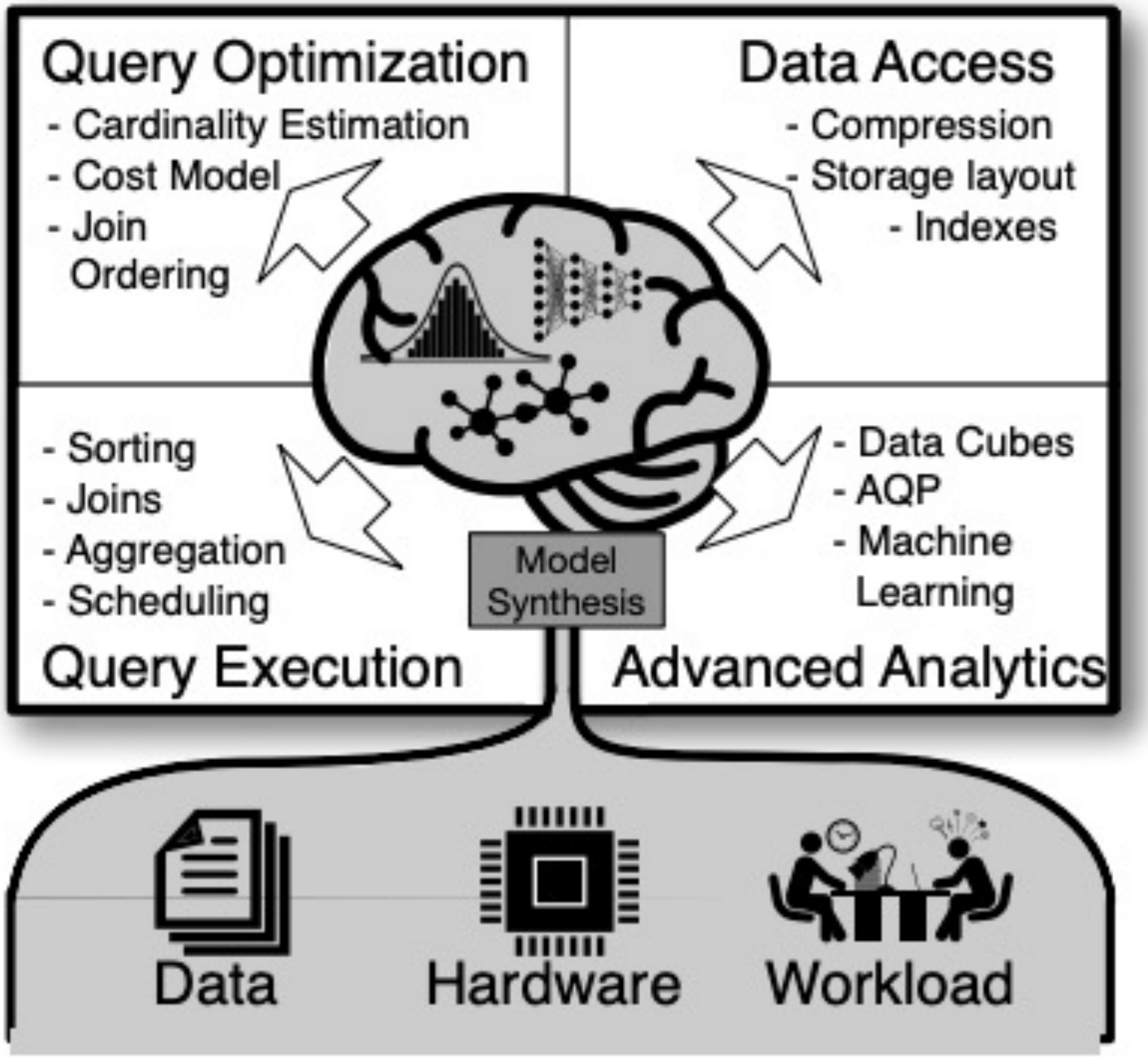
MySQL

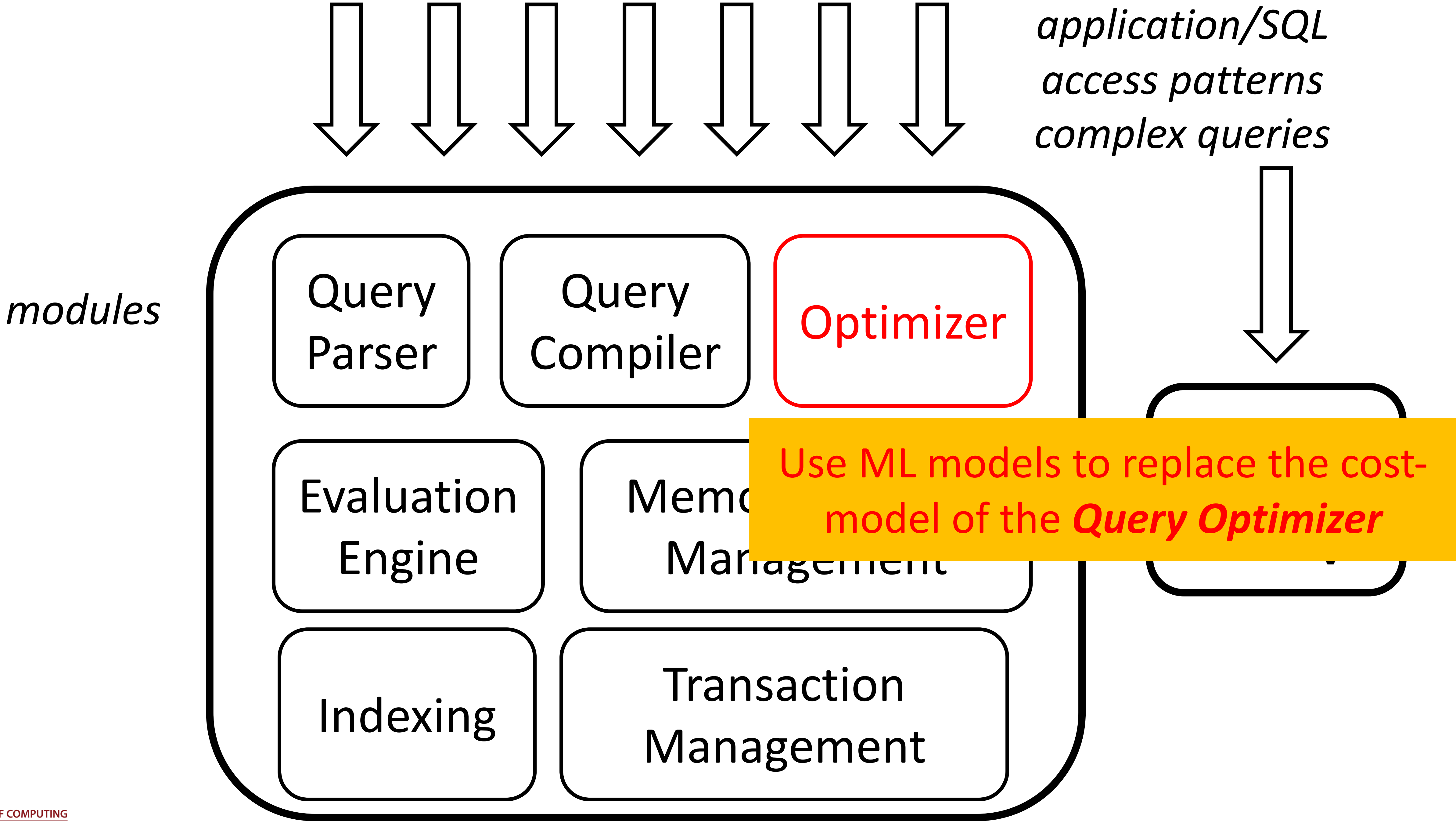
(a) TPC-C (Throughput) **(b)** TPC-C (99%-tile Latency)

PostgreSQL

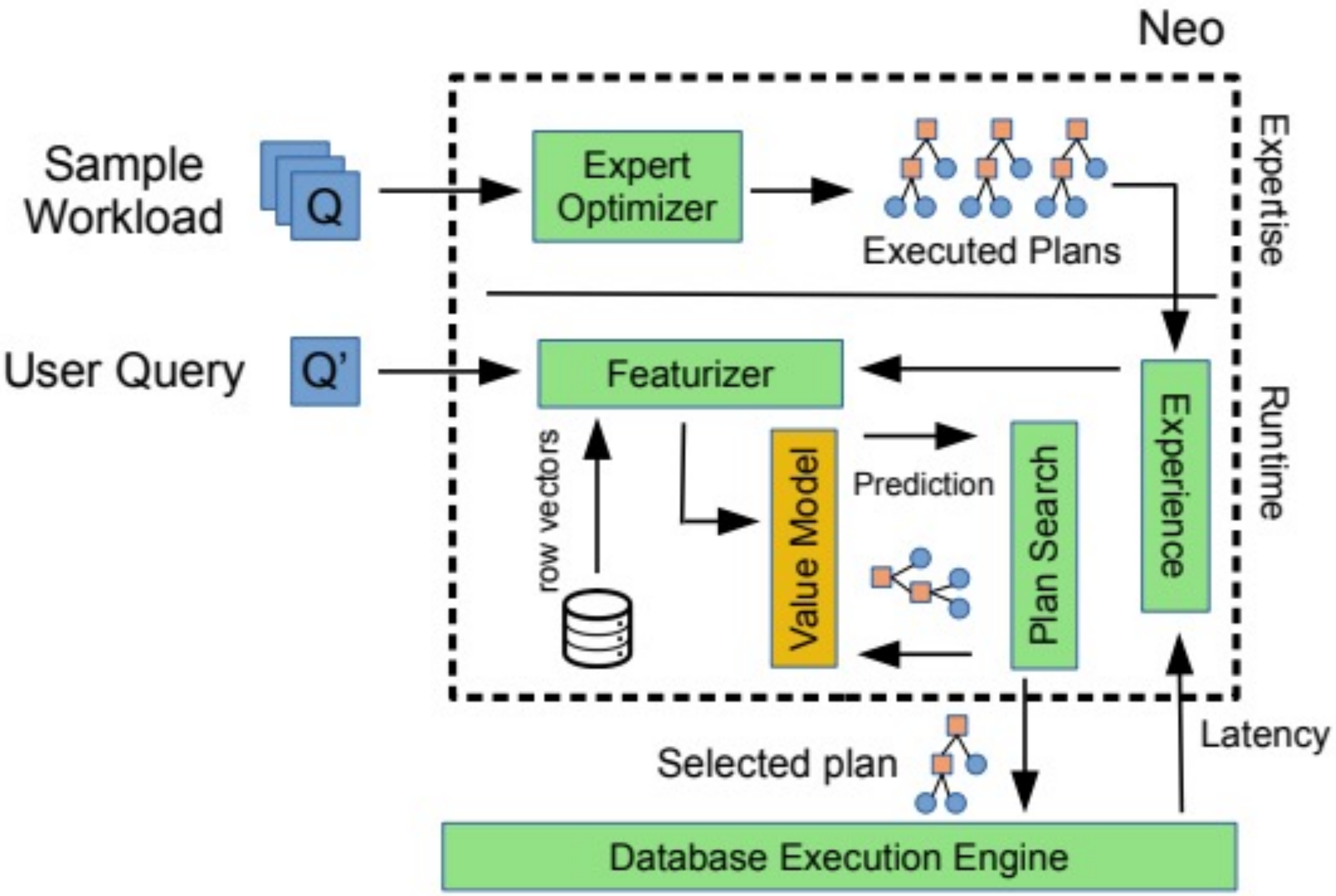
It is hard (but not impossible) to beat an expert DBA!

A Learned Database System

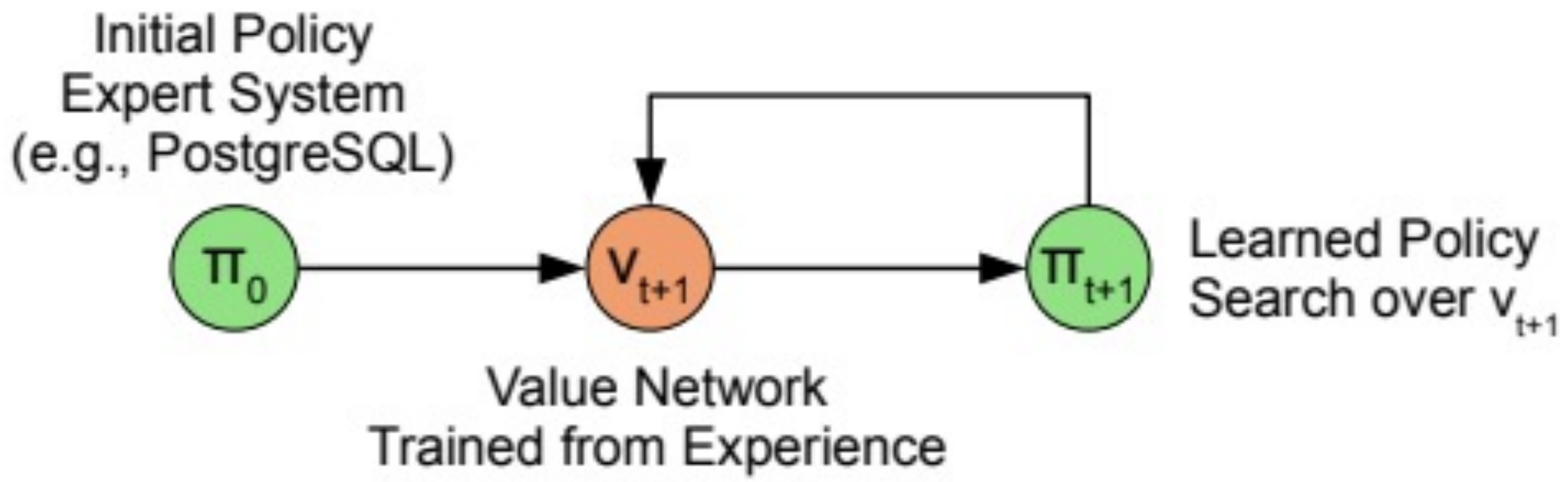


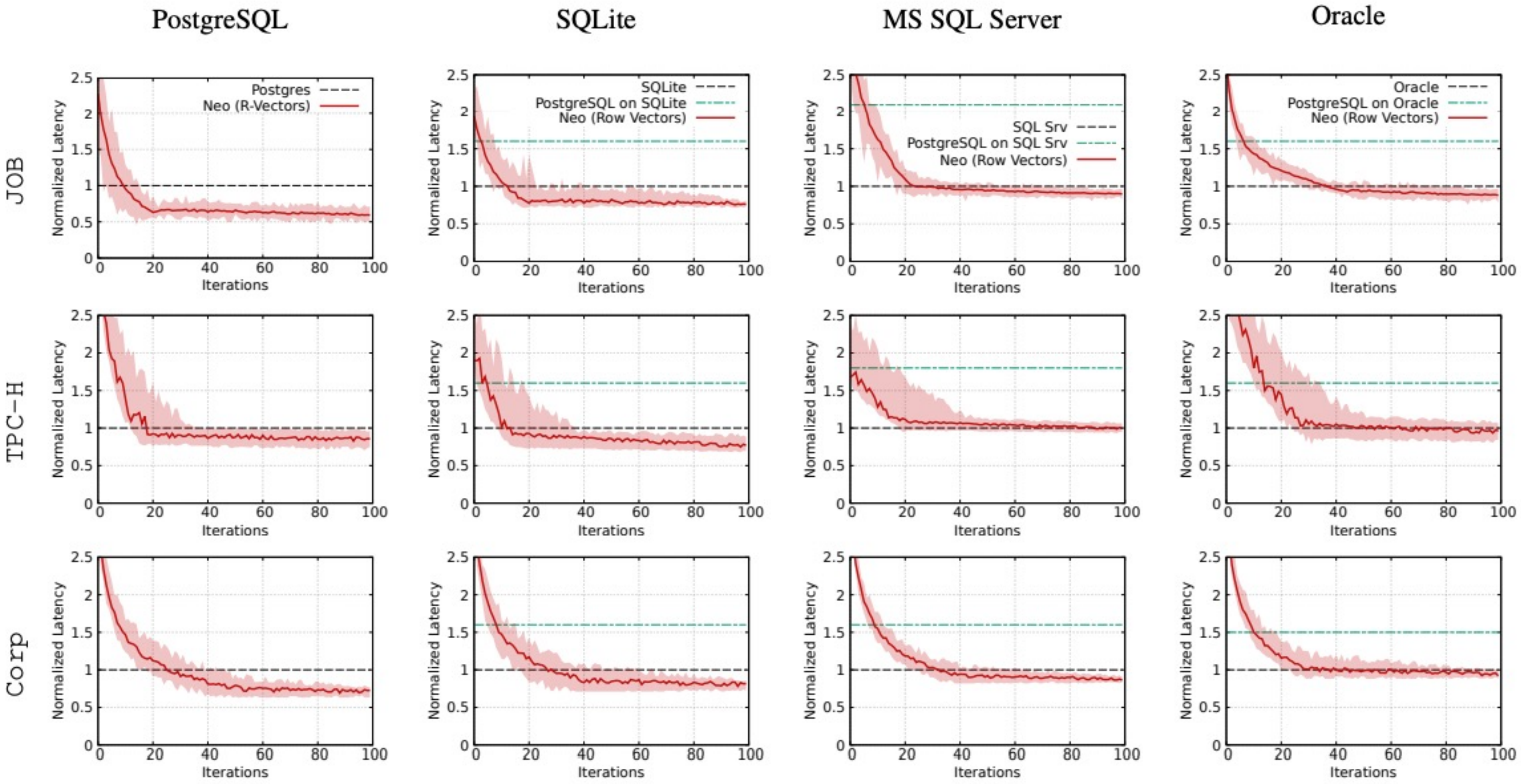

application/SQL
access patterns
complex queries
modules
Query Parser
Query Compiler
Optimizer
Evaluation Engine
Indexing
Transaction Management
Use ML models to replace the cost-model of the *Query Optimizer*

Learned Query Optimization



Learned Query Optimization




PostgreSQL
SQLite
MS SQL Server
Oracle
JOB
TPC-H
Corp
Postgres
Neo (R-Vectors)
SQLite
PostgreSQL on SQLite
Neo (Row Vectors)
SQL Srv
PostgreSQL on SQL Srv
Neo (Row Vectors)
Oracle
PostgreSQL on Oracle
Neo (Row Vectors)
Normalized Latency
Iterations
0
0.5
1
1.5
2
2.5
0
20
40
60
80
100

A perspective on ML in Database Systems

from: ML-In-Databases: Assessment and Prognosis, IEEE Data Engineering Bulletin

New *Forces*

(1) End-users want to

democratize data (all business units to have access to all data)

make data-driven decisions (often in real time)

(2) New applications

structured query processing (SQL) + natural language processing (NLP) + Complex Analytics (exploratory + predictive ML)

New *Forces*

(3) Data integration

diverse and inconsistent datasets are combined in common data repositories (data lakes)

(2) New hardware + the move to the cloud

moving from full ownership to pay-as-you-go

self-tuning systems *en masse* in the cloud (as we discussed today)

Consequences and New Directions

Storage ***hierarchy*** is still relevant, but the layers are elastic (in the cloud)

ML models can be deployed at-will as “functions”

New push for ***serverless computing***

use only services and not rent an entire server