BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 20

In-Situ Data Processing

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/

with slides from Dr. Ioannis Alagiannis and Dr. Matthaios Olma

NoDB: Efficient Query Execution on Raw Data Files

Ioannis Alagiannis[*] Renata Borovica[*] Miguel Branco[*] Stratos Idreos[‡] Anastasia Ailamaki[*]

[*]EPFL, Switzerland
{ioannis.alagiannis, renata.borovica, miguel.branco, anastasia.ailamaki}@epfl.ch

[‡]CWI, Amsterdam
stratos.idreos@cwi.nl

ABSTRACT

As data collections become larger and larger, data loading evolves to a major bottleneck. Many applications already avoid using database systems, e.g., scientific data analysis and social networks, due to the complexity and the increased *data-to-query* time. For such applications data collections keep growing fast, even on a daily basis, and we are already in the era of *data deluge* where we have much more data than what we can move, store, let alone analyze.

Our contribution in this paper is the design and roadmap of a new paradigm in database systems, called NoDB, which *do not require data loading while still maintaining the whole feature set of a modern database system*. In particular, we show how to make raw data files a first-class citizen, fully integrated with the query engine. Through our design and lessons learned by implementing the NoDB philosophy over a modern DBMS, we discuss the fundamental limitations as well as the strong opportunities that such a research path brings. We identify performance bottlenecks specific for in situ processing, namely the repeated parsing and tokenizing overhead and the expensive data type conversion costs. To address these problems, we introduce an adaptive indexing mechanism that maintains positional information to provide efficient access to raw data files, together with a flexible caching structure.

Our implementation over PostgreSQL, called PostgresRaw, is able to avoid the loading cost completely, while matching the query performance of plain PostgreSQL and even outperforming it in many cases. We conclude that NoDB systems are feasible to design and implement over modern database architectures, bringing an unprecedented positive effect in usability and performance.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - Query Processing; H.2.8 [**Database Applications**]: Scientific Databases

General Terms

Algorithms, Design, Performance

Keywords

Adaptive loading, In situ querying, Positional map

1. INTRODUCTION

We are now entering the era of data deluge, where the amount of data outgrows the capabilities of query processing technology. Many emerging applications, from social networks to scientific experiments, are representative examples of this deluge, where the rate at which data is produced exceeds any past experience. Scientific analysis such as astronomy is soon expected to collect multiple Terabytes of data on a daily basis, while web-based businesses such as social networks or web log analysis are already confronted with a growing stream of large data inputs. Therefore, there is a clear need for efficient big data processing to enable the evolution of businesses and sciences to the new era of data deluge.

Motivation. Although Database Management Systems (DBMS) remain overall the predominant data analysis technology, they are rarely used for emerging applications such as scientific analysis and social networks. This is largely due to the complexity involved; there is a significant initialization cost in loading data and preparing the database system for queries. For example, a scientist needs to quickly examine a few Terabytes of new data in search of certain properties. Even though only few attributes might be relevant for the task, the entire data must first be loaded inside the database. For large amounts of data, this means a few hours of delay, even with parallel loading across multiple machines. Besides being a significant time investment, it is also important to consider the extra computing resources required for a full load and its side-effects with respect to energy consumption and economical sustainability.

Instead of using database systems, emerging applications rely on custom solutions that usually miss important database features. For instance, declarative queries, schema evolution and complete isolation from the internal representation of data are rarely present. The problem with the situation today is in many ways similar to the past, before the first relational systems were introduced; there are a wide variety of competing approaches but users remain exposed to many low-level details and must work close to the physical level to obtain adequate performance and scalability.

The lessons learned in the past four decades indicate that in order to efficiently cope with the data deluge era in the long run, we will need to rely on the fundamental principles adopted by database management technology. That is, we will need to build extensible systems with declarative query processing and self-managing optimization techniques that will be tailored for the data deluge. A growing part of the database community recognizes this need for

Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing

Matthaios Olma[†] Manos Karpathiotakis[†] Ioannis Alagiannis[‡] Manos Athanassoulis[*] Anastasia Ailamaki[†]

[†]EPFL
{firstname.lastname}@epfl.ch

[‡]Microsoft
ioalagia@microsoft.com

[*]Harvard University
manos@seas.harvard.edu

ABSTRACT

The constant flux of data and queries alike has been pushing the boundaries of data analysis systems. The increasing size of raw data files has made data loading an expensive operation that delays the data-to-insight time. Hence, recent in-situ query processing systems operate directly over raw data, alleviating the loading cost. At the same time, analytical workloads have increasing number of queries. Typically, each query focuses on a constantly shifting – yet small – range. Minimizing the workload latency, now, requires the benefits of indexing in in-situ query processing.

In this paper, we present Slalom, an in-situ query engine that accommodates workload shifts by monitoring user access patterns. Slalom makes on-the-fly partitioning and indexing decisions, based on information collected by lightweight monitoring. Slalom has two key components: (i) an online partitioning and indexing scheme, and (ii) a partitioning and indexing tuner tailored for in-situ query engines. When compared to the state of the art, Slalom offers performance benefits by taking into account user query patterns to (a) *logically* partition raw data files and (b) build for each partition lightweight *partition-specific* indexes. Due to its lightweight and adaptive nature, Slalom achieves efficient accesses to raw data with minimal memory consumption. Our experimentation with both micro-benchmarks and real-life workloads shows that Slalom outperforms state-of-the-art in-situ engines ($3-10\times$), and achieves comparable query response times with fully indexed DBMS, offering much lower ($\sim 3\times$) cumulative query execution times for query workloads with increasing size and unpredictable access patterns.

1. INTRODUCTION

Nowadays, an increasing number of applications generate and collect massive amounts of data at a rapid pace. New research fields and applications (e.g., network monitoring, sensor data management, clinical studies, etc.) emerge and require broader data analysis functionality to rapidly gain deeper insights from the available data. In practice, analyzing such datasets becomes a costly task due to the data explosion of the last decade.

Big Data, Small Queries. The trend of exponential data growth due to intense data generation and data collection is expected to

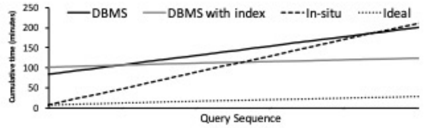


Figure 1: Ideally, in-situ data analysis should be able to retrieve only the relevant data for each query after the initial table scan (ideal - dotted line). In practice today, in-situ query processing avoids the costly phase of data loading (dashed line), however, as the number of the queries increases, the initial investment for full index on a DBMS pays off (the dashed line meets the grey line).

persist, however, recent studies of the data analysis workloads show that typically only a small subset of the data is relevant and ultimately used by analytical and/or exploratory workloads [1, 18]. In addition, modern businesses and scientific applications require interactive data access, which is characterized by *no or little a priori workload knowledge* and constant *workload shifting* both in terms of projected attributes and selected ranges of the dataset.

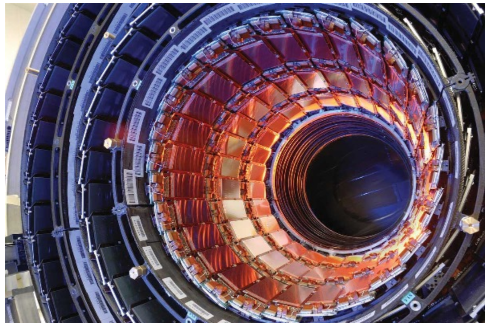
The Cost of Loading, Indexing, and Tuning. Traditional data management systems (DBMS) require the costly steps of *data loading*, *physical design decisions*, and then *index building* in order to offer interactive access over large datasets. Given the data sizes involved, any transformation, copying, and preparation steps over the data introduce substantial delays before the data can be queried, and provide useful insights [2, 5, 34]. The lack of a priori knowledge of the workload makes the physical design decisions virtually impossible because cost-based advisors rely heavily on past or sample workload knowledge [3, 17, 22, 29, 58]. The workload shifts observed in the interactive setting of exploratory workloads can nullify investments towards indexing and other auxiliary data structures (e.g., views), since frequently, they depend on the actual data values and the knowledge generated by the ongoing analysis.

Querying Raw Data Files Is Not Enough. Recent efforts opt to query directly raw files [2, 5, 13, 19, 30, 40] to reduce the data-to-insight cost. These *in-situ* systems avoid the costly initial data loading step, and allow the execution of declarative queries over external files without duplicating or "locking" data in a proprietary database format. Further, they concentrate on reducing costs as-

Extracting knowledge from data

"Most firms estimate that they are only analyzing 12% of the data that they already have" [Forrester 2014]

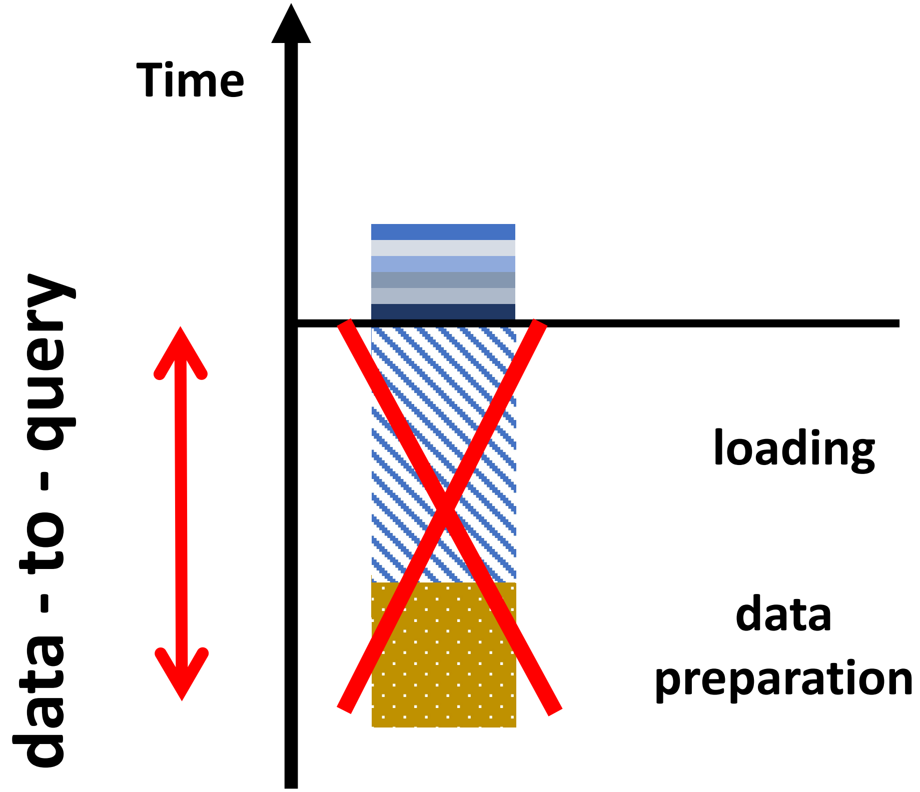
- Growing data collections
- No a priori knowledge about data
- Ad hoc queries



Need for efficient data exploration

BOSTON UNIVERSITY

From data to results



Increases data-to-query time

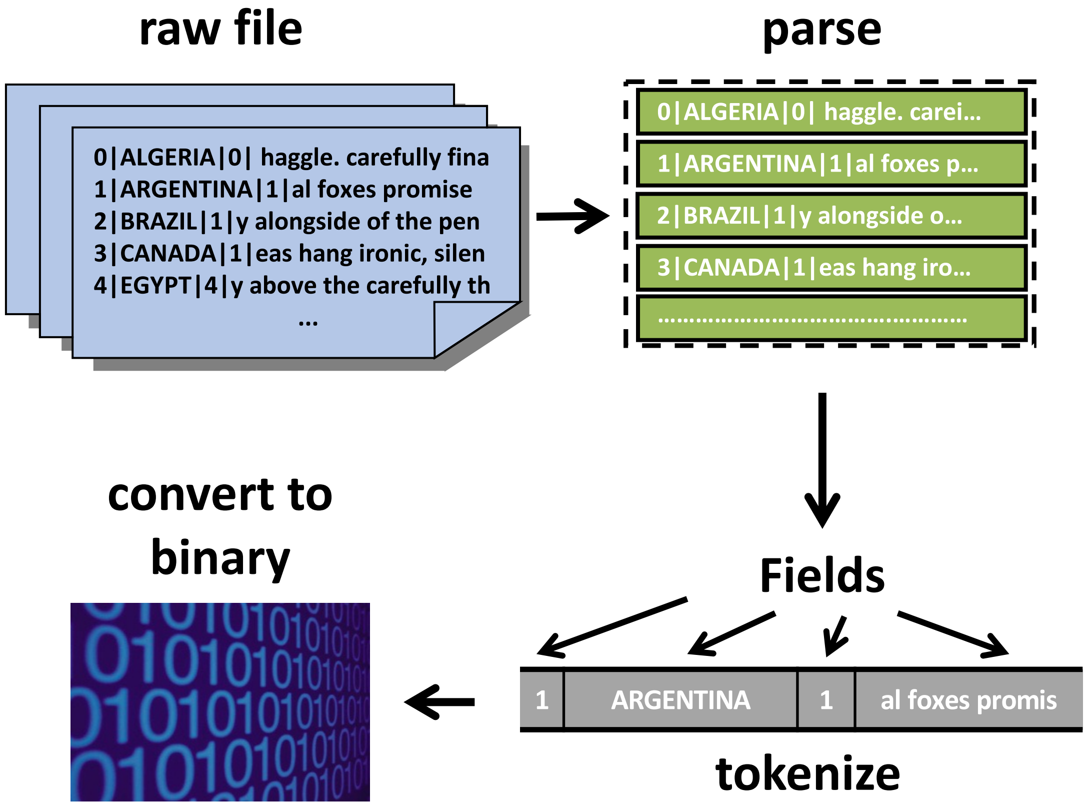
Requires workload knowledge

Data loading

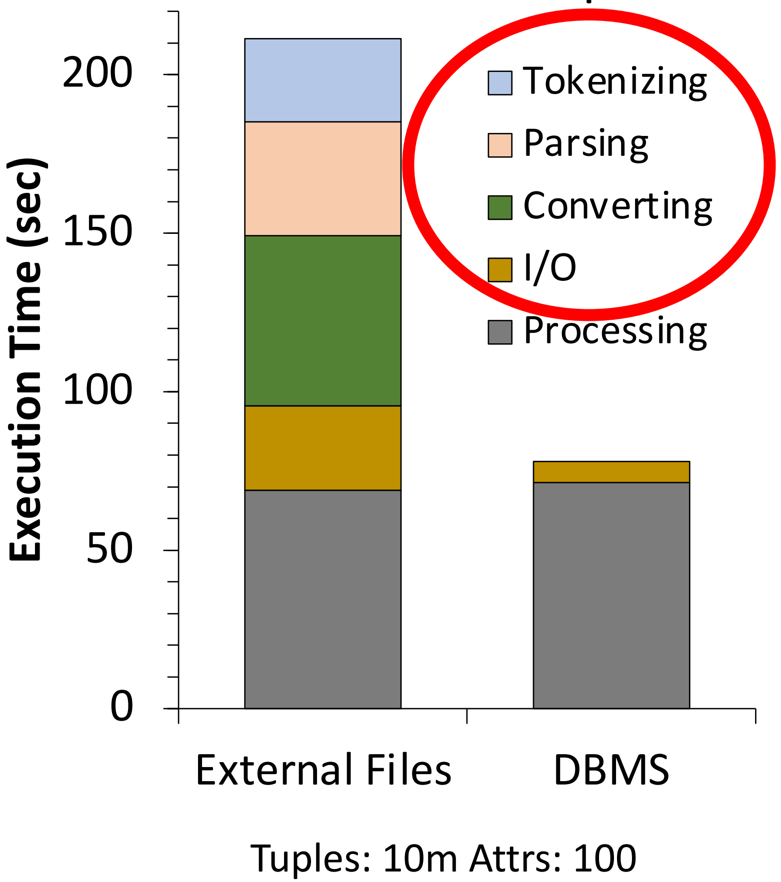
- Part of the first query
  - Both for row-stores and column-stores
- In practice:
  - Cost increases linearly with the dataset size
  - CPU and I/O intensive

Data analysis cost should depend on the data we need to process

Querying data *in situ**



no indexing
no statistics
+



…straw-man approach is slow

* DBMS-X External tables, CSV engine MySQL

Why *in-situ* query processing?

Quick data-to-query time

Why not DBMS?

Partial/no data ownership → cannot transform and load

NoDB: Technology

Efficient *in situ* querying

- **selective tokenizing**
- **positional indexing**
- **adaptive caching**
- **statistics**
- **vertical indexing**

Minimal changes to the query engine

PostgreSQL **+ NoDB = PostgresRaw**

PostgresRaw

Adaptive indexing mechanism

positional map

raw files

scan operator

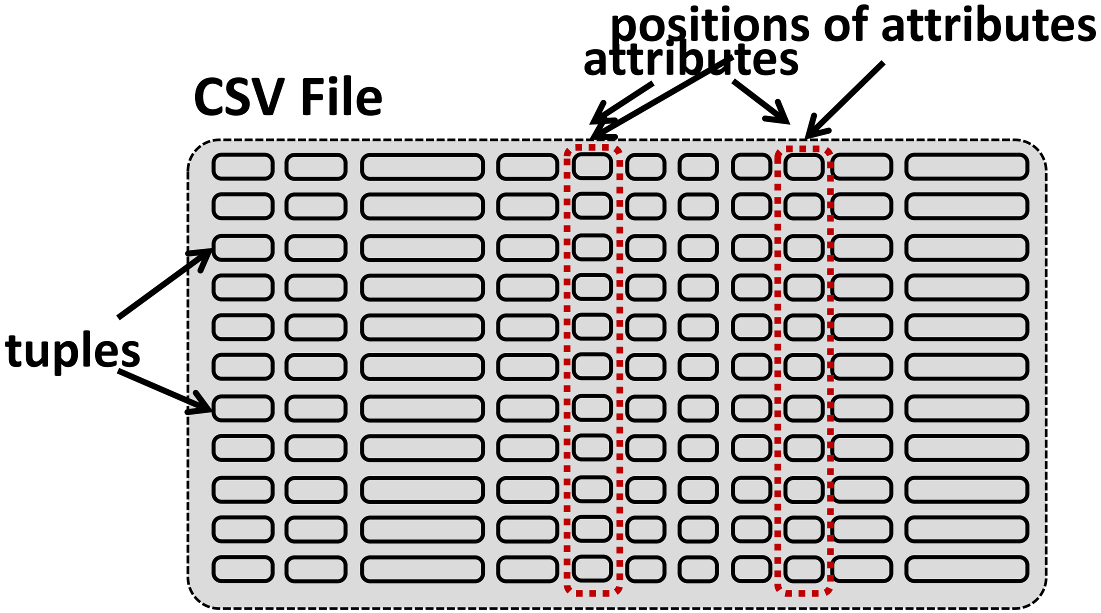
cache

statistics

PostgreSQL query engine

BOSTON UNIVERSITY

Positional map

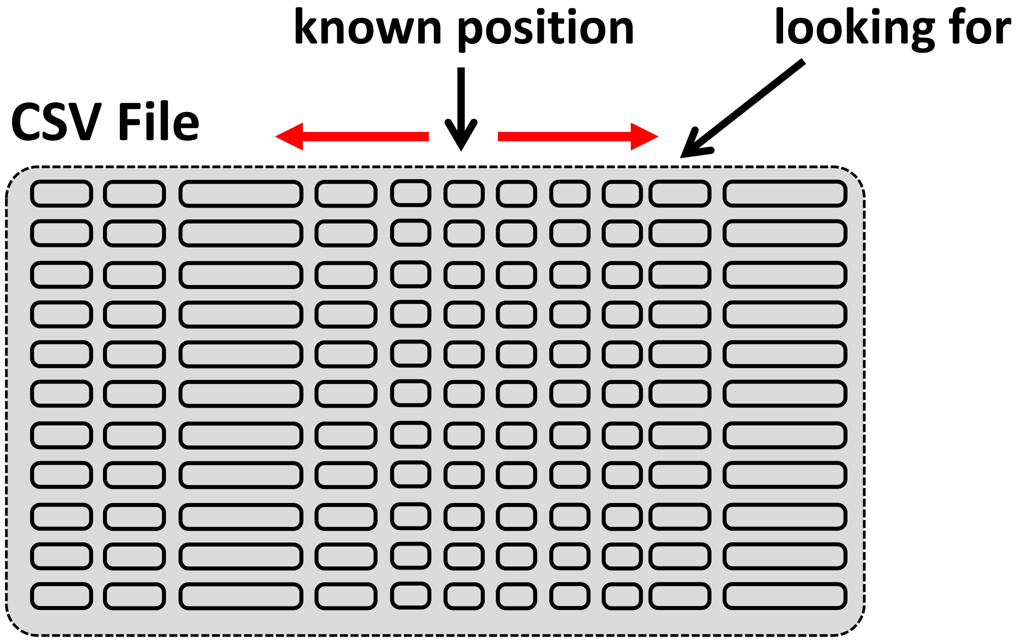


Reduce parsing

Reduce tokenizing

Created on-the-fly

Positional map



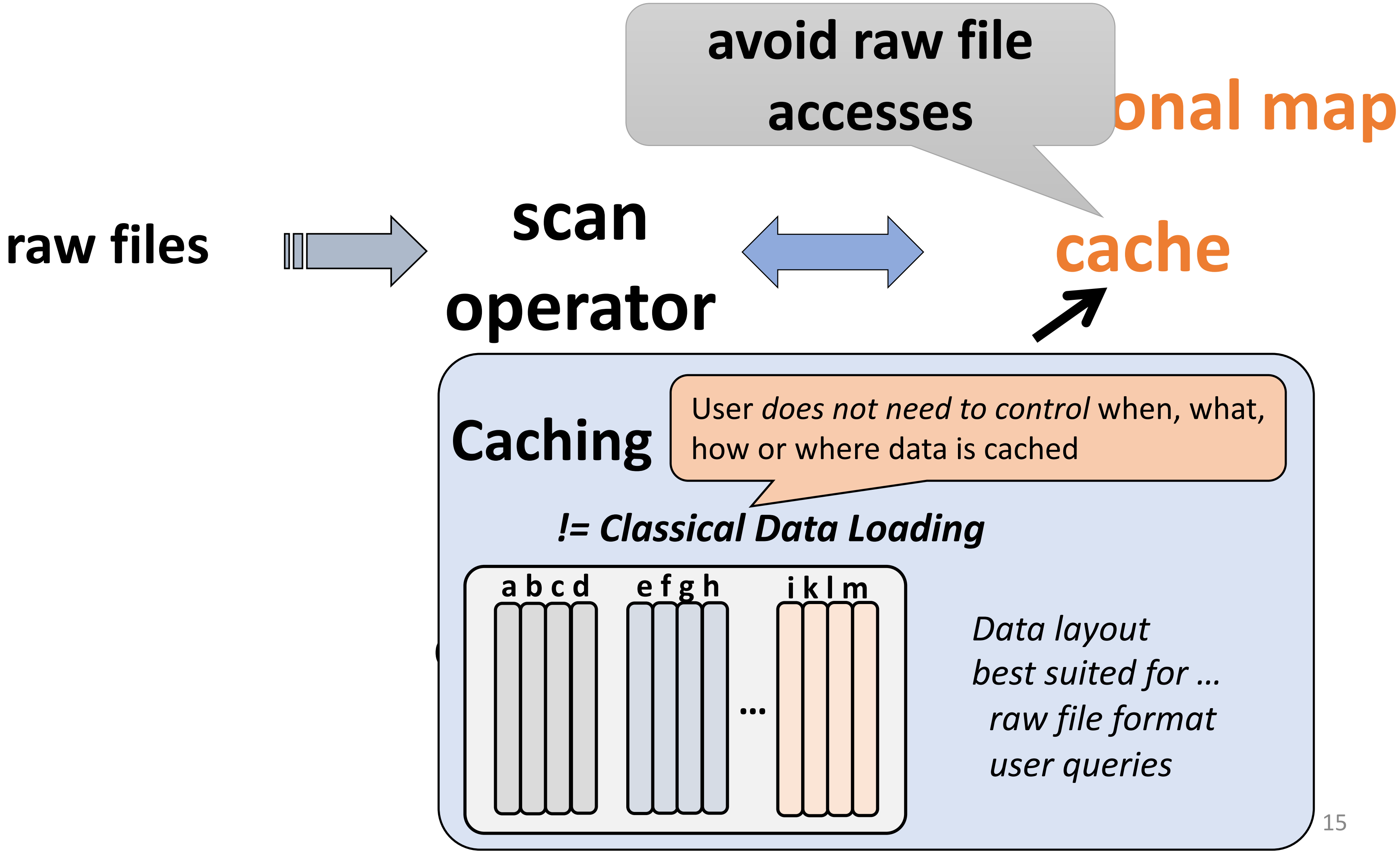
Reduce parsing

Reduce tokenizing

Created on-the-fly

Make raw data access progressively cheaper

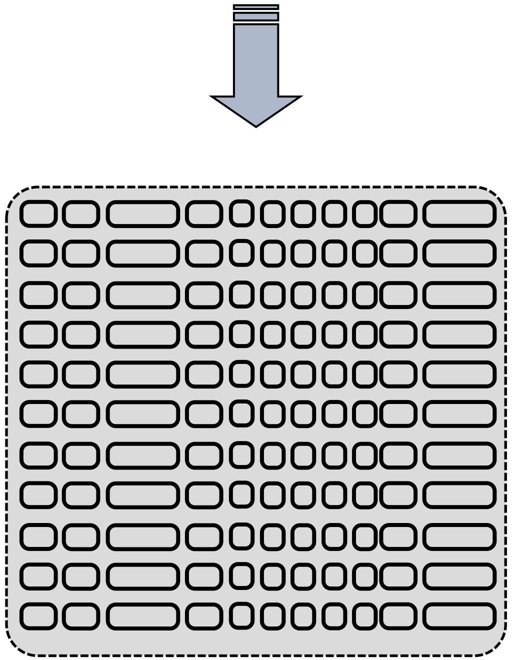
PostgresRaw



BOSTON UNIVERSITY

PostgresRaw: access paths

scan operator

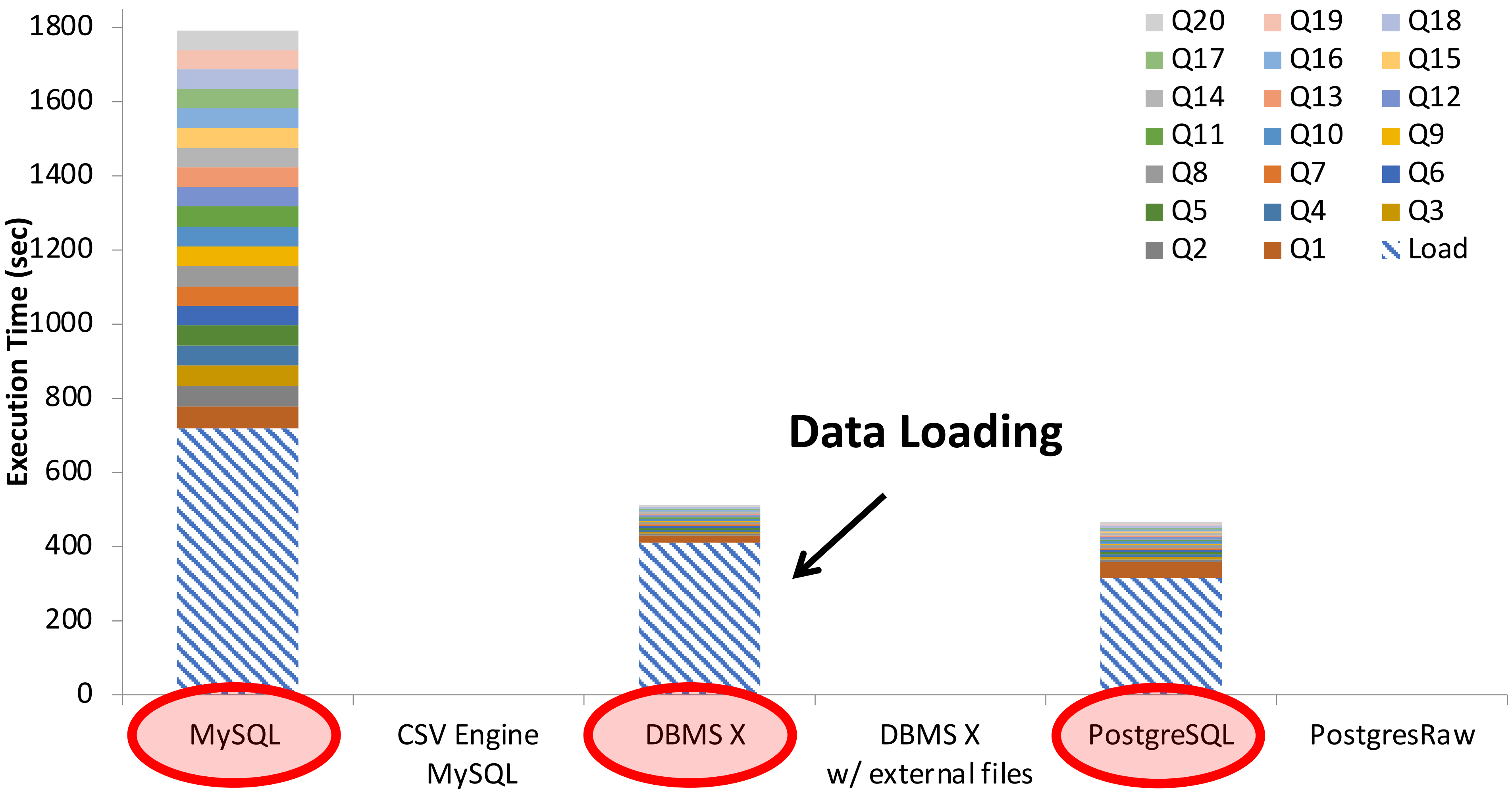


- Vertical index
- Caching
- Positional map
- Direct access

BOSTON UNIVERSITY

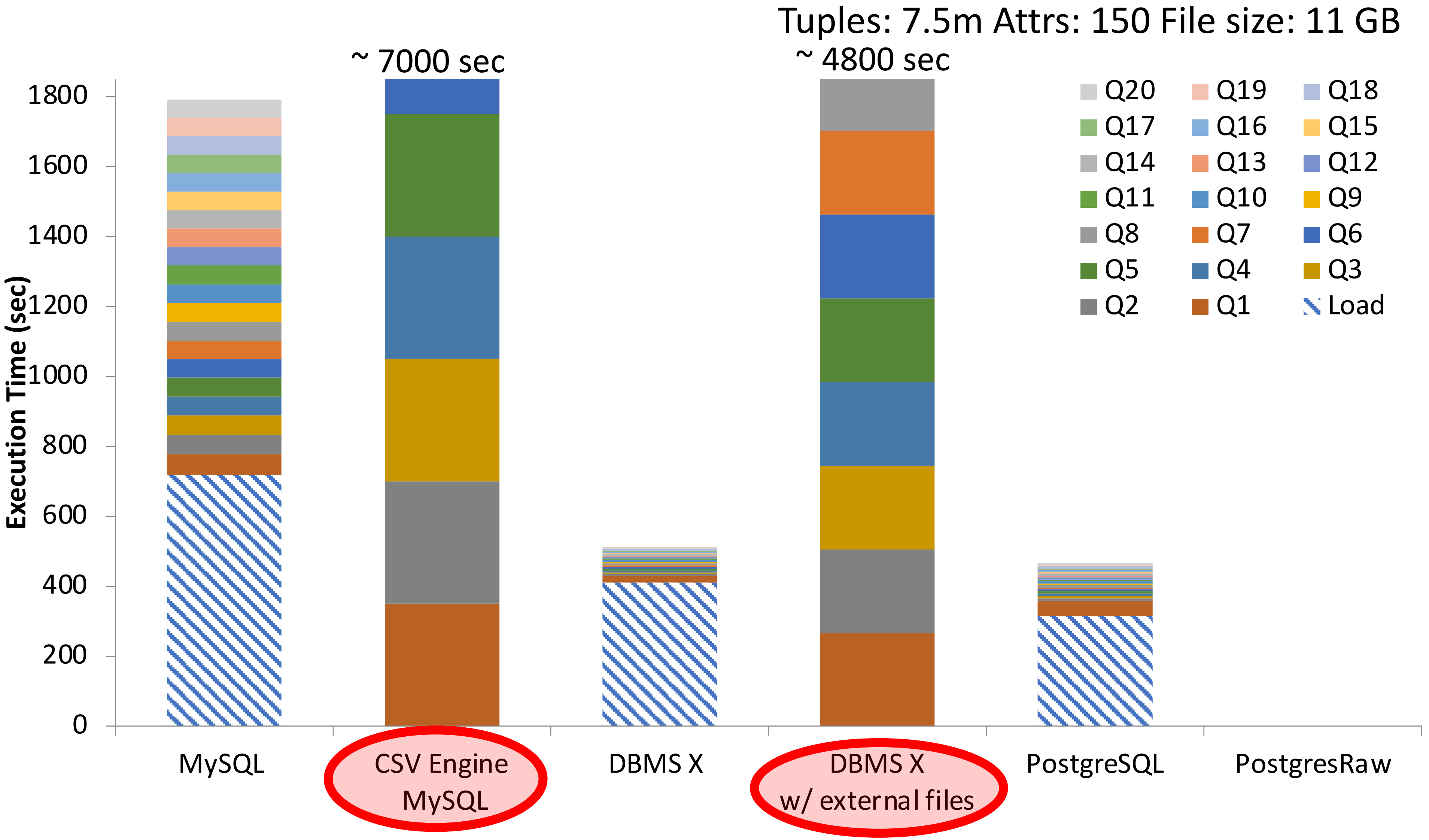
PostgresRaw vs. other DBMS

Tuples: 7.5m Attrs: 150 File size: 11 GB



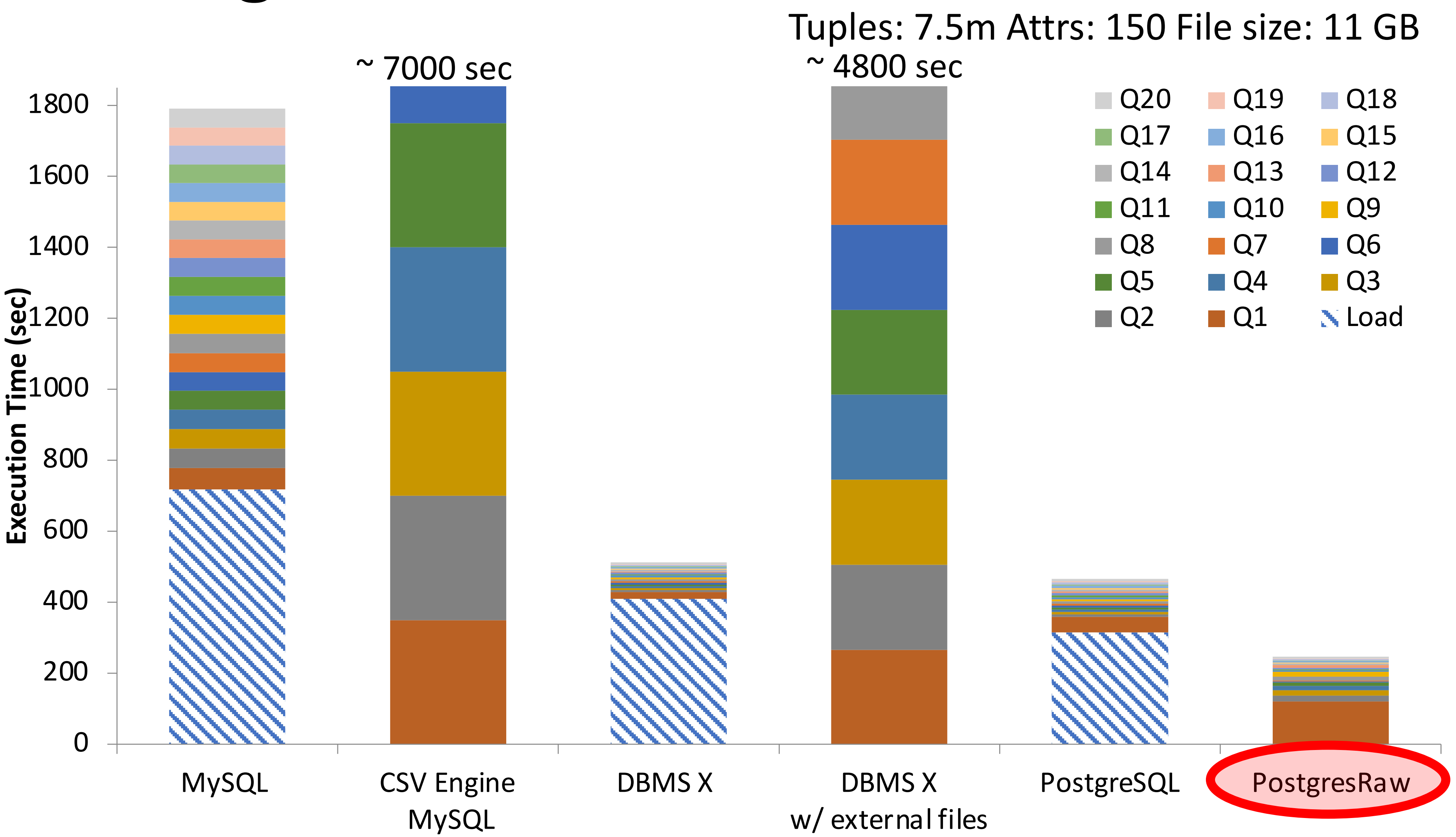
BOSTON UNIVERSITY

PostgresRaw vs. other DBMS



BOSTON UNIVERSITY

PostgresRaw vs. other DBMS

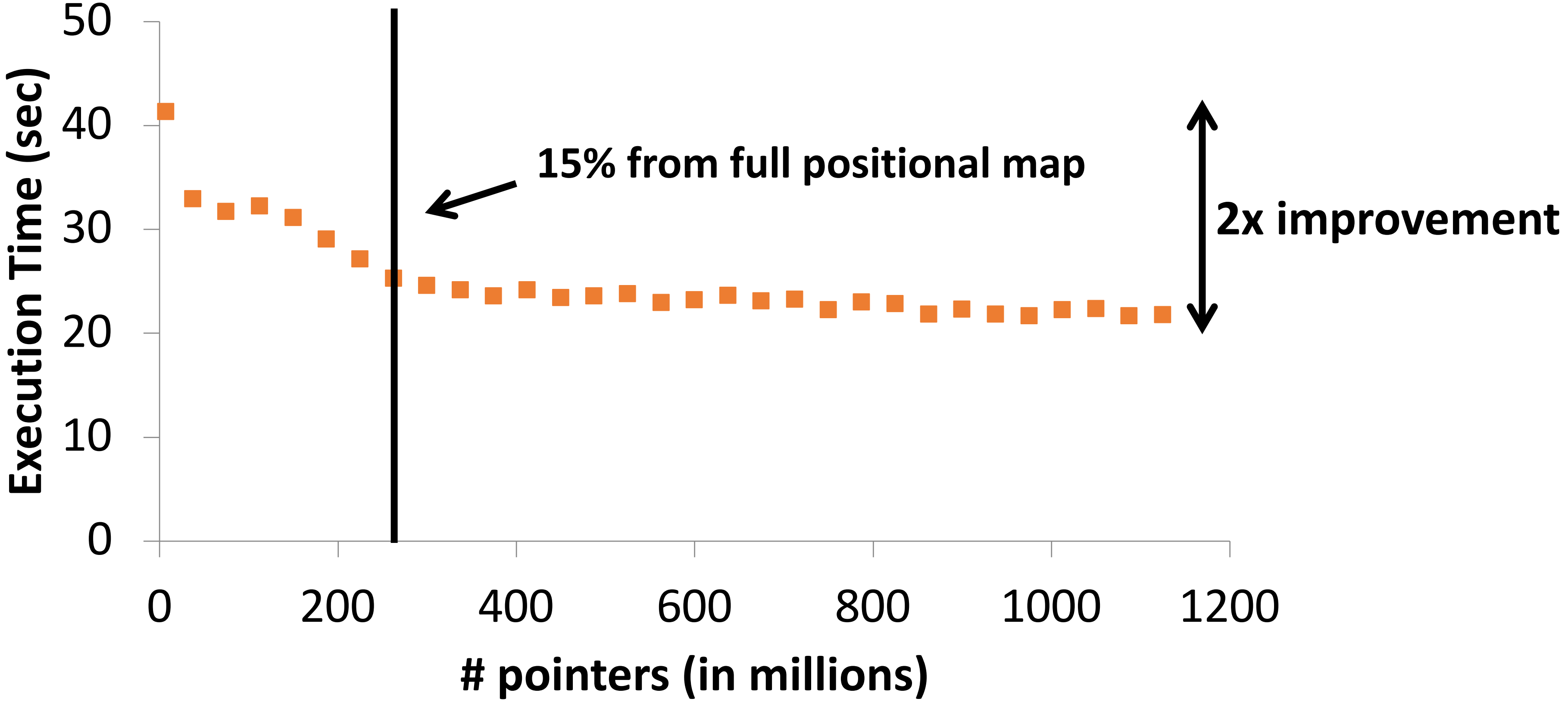


BOSTON UNIVERSITY

Comparable/competitive performance

Impact of positional map

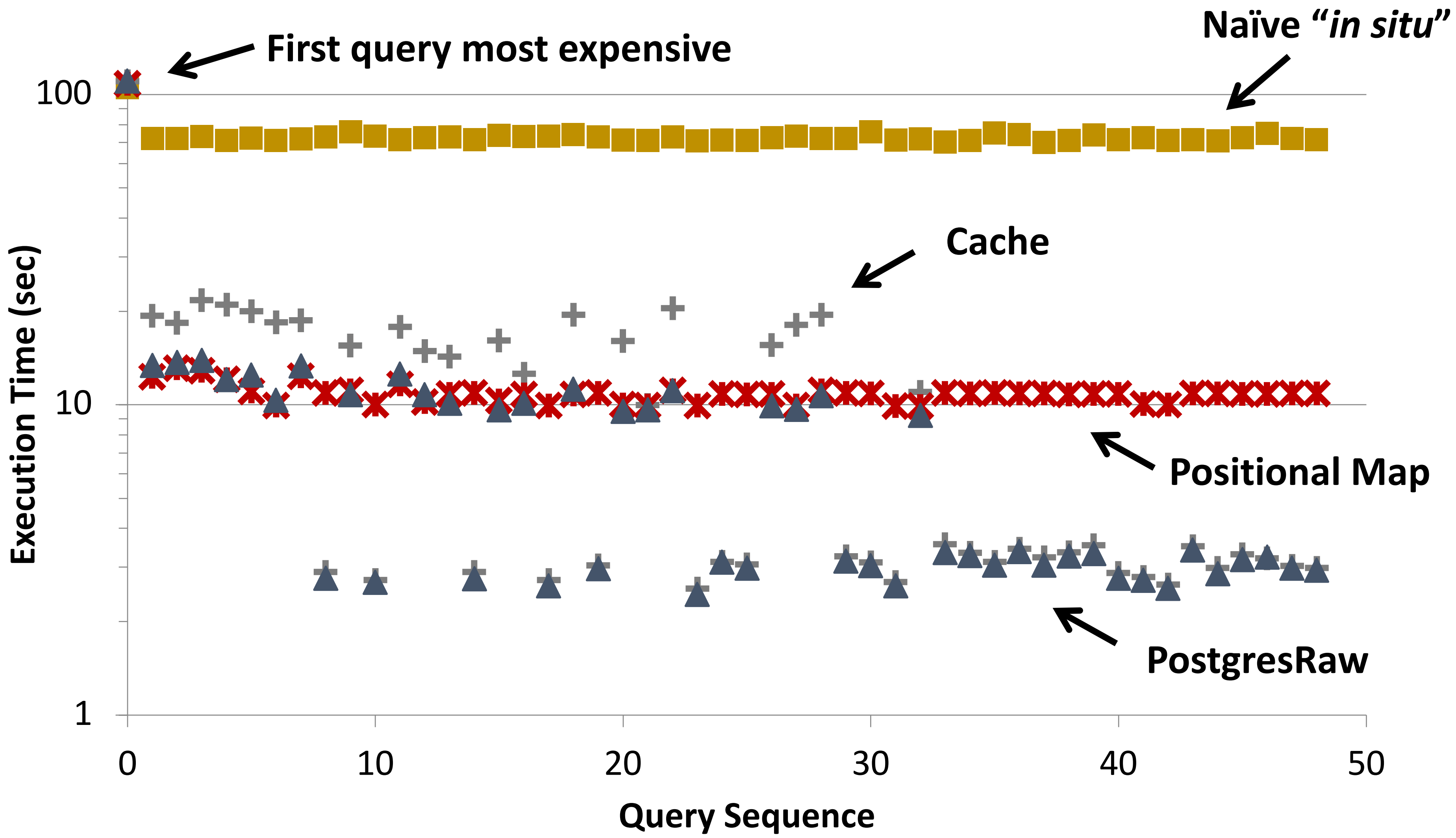
Random queries on 10 attributes
Tuples: 7.5m Attrs: 150 File size: 11 GB
Vary storage capacity (15MB-2GB)



No need for the whole positional map

BOSTON UNIVERSITY

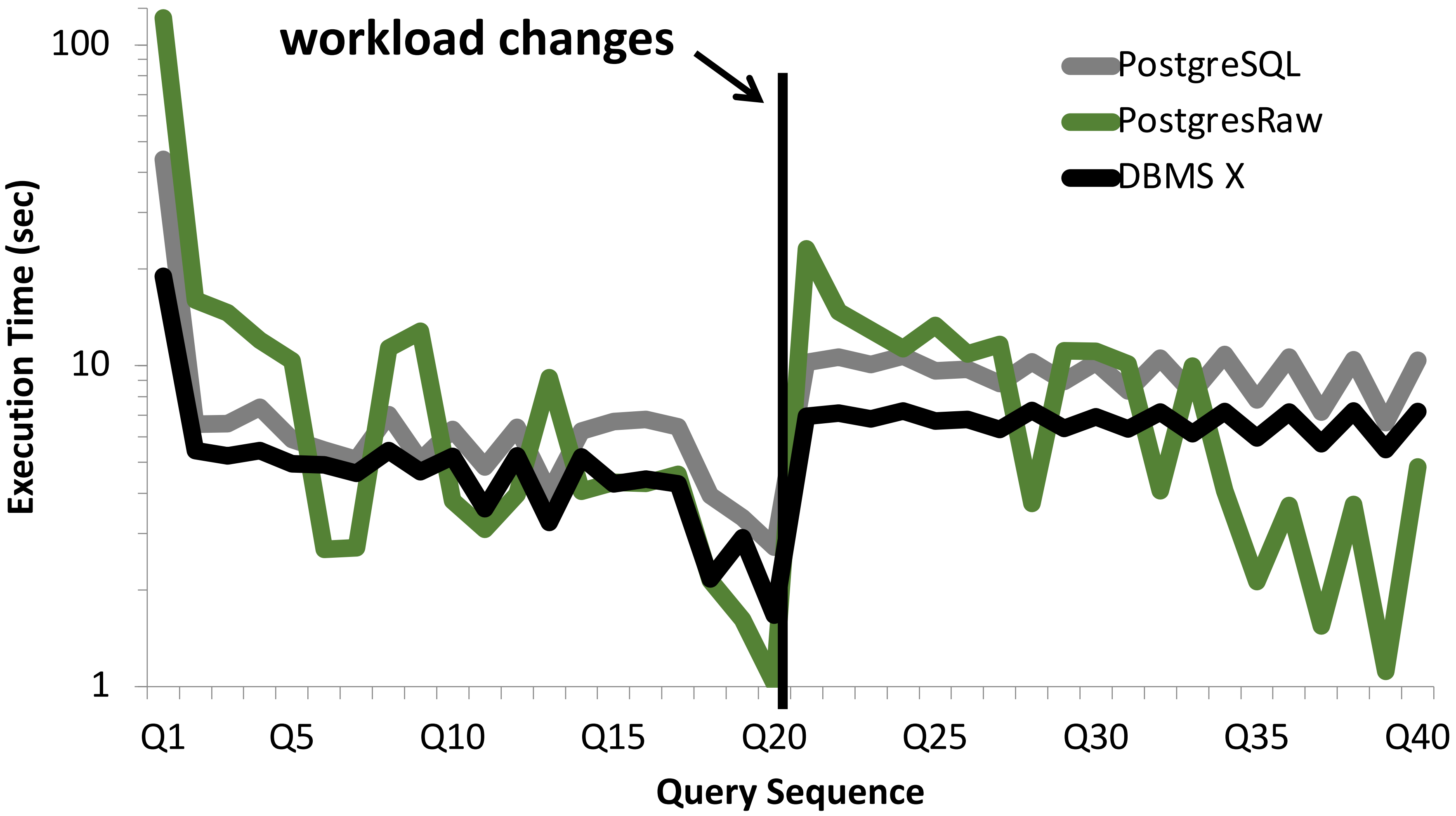
Is caching enough?



Best choice: Combine positional map and cache

BOSTON UNIVERSITY

Adapting to changes

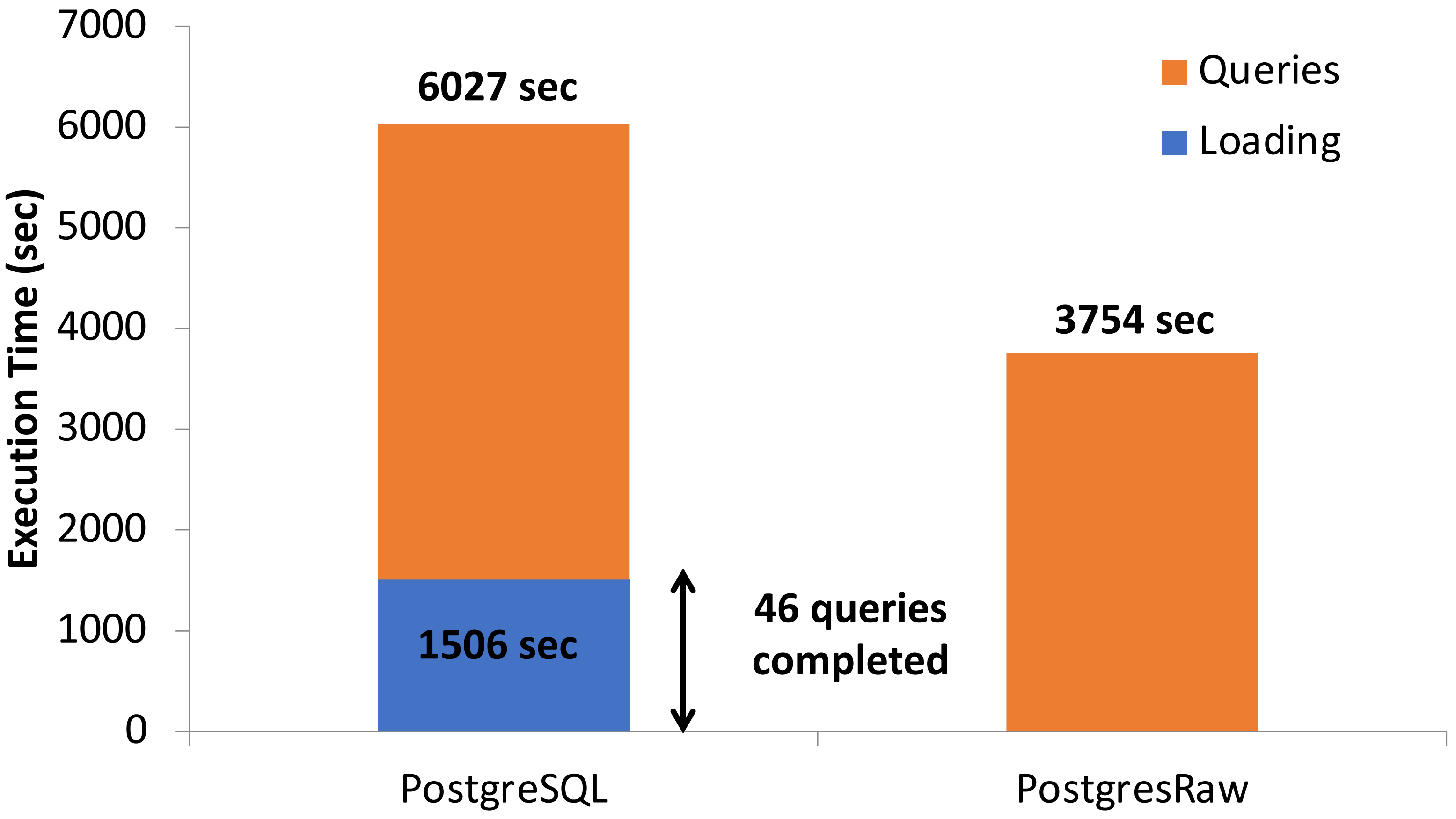


Graceful adaptation to workload changes

BOSTON UNIVERSITY

PostgreSQL vs. PostgresRaw

Tuples: 50m Attrs: 150
File size: **73 GB** DB size: 29 GB
150 queries each accessing 5 attrs



BOSTON UNIVERSITY

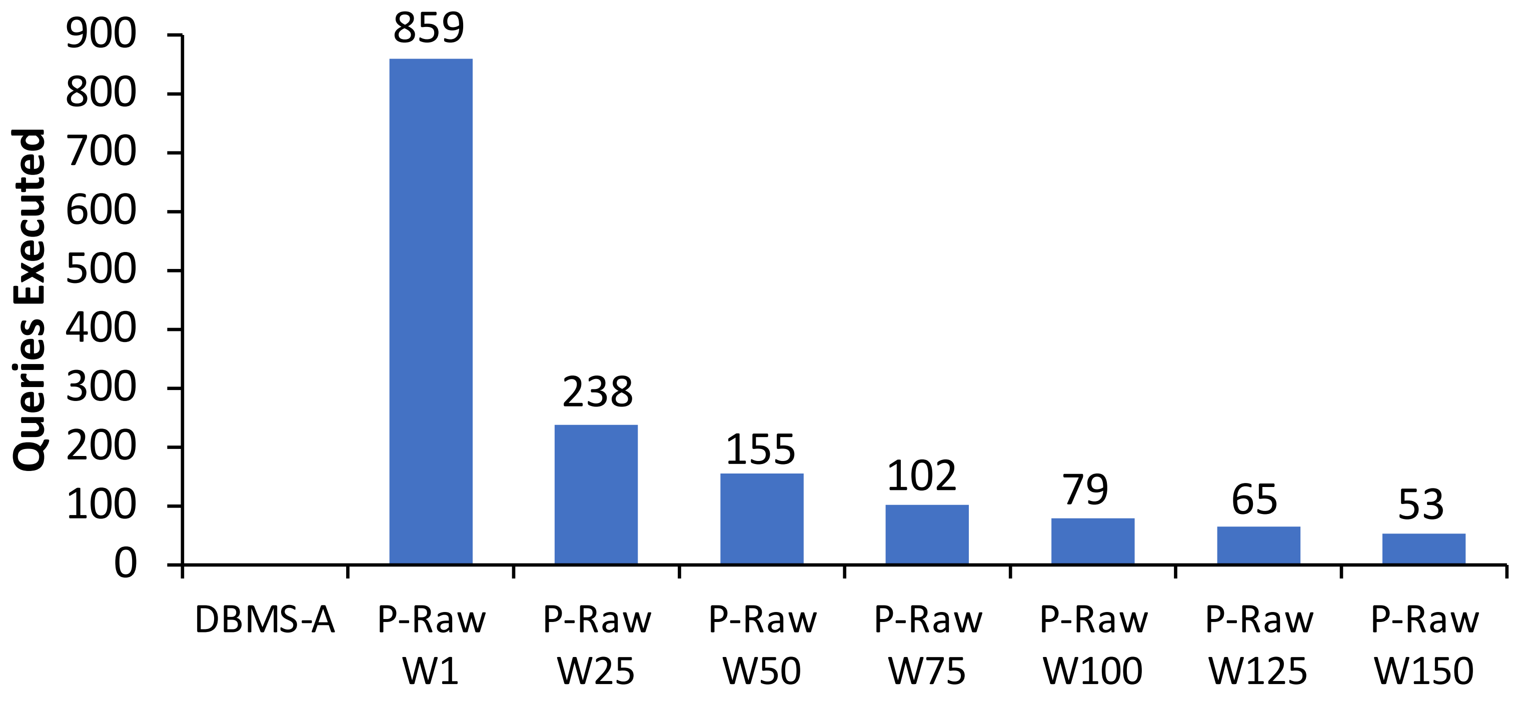
Break-even point

Parallel data loading vs. Parallel raw access (16 threads)

DBMS-A data loading: **925 sec**

Tuples: 40m Attrs: 150 File size: 56 GB

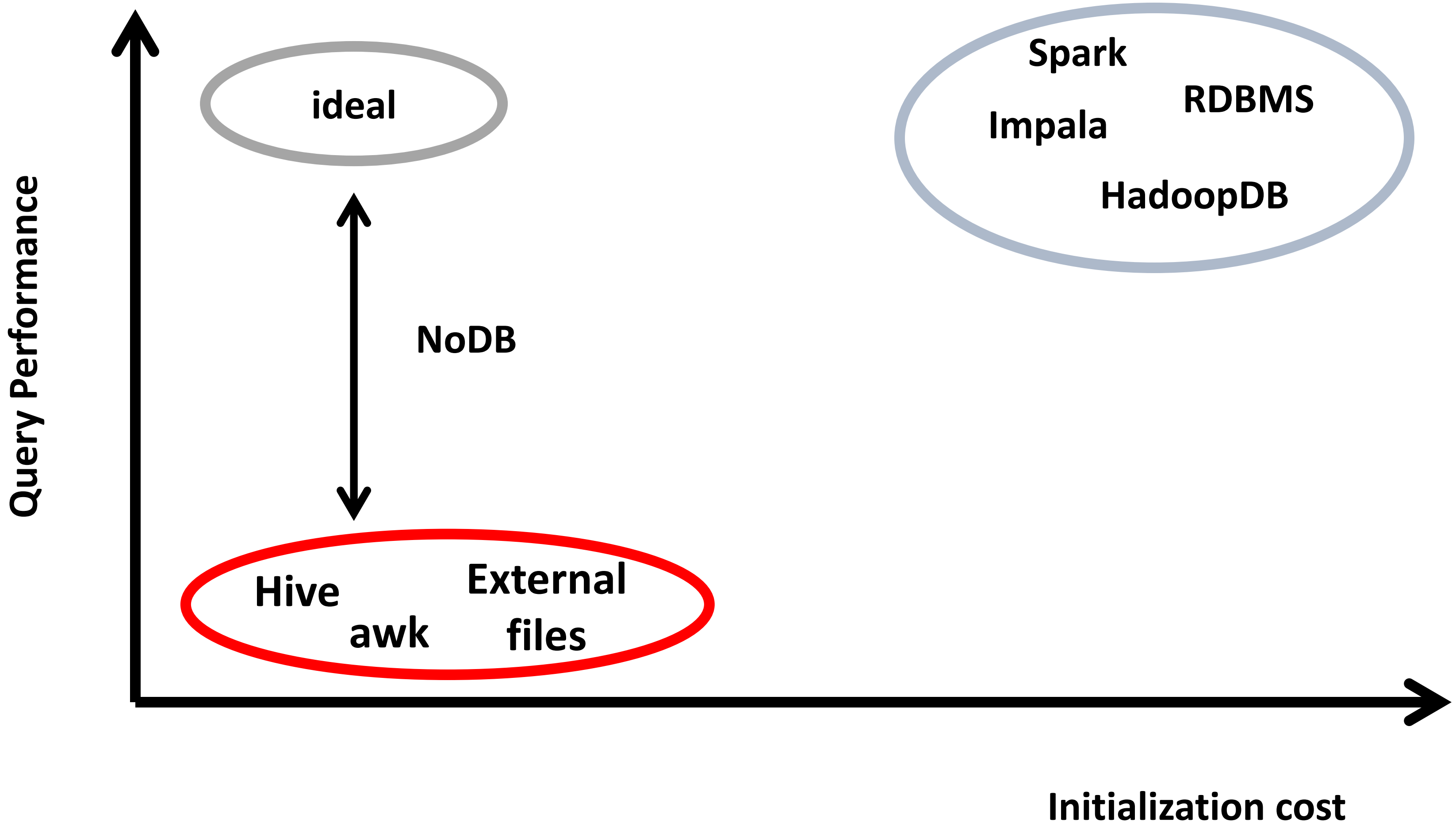
Query Template: select max(X), …, max(Y) from R;



Querying data files is a viable alternative even for long sequences of queries

BOSTON UNIVERSITY

NoDB in the research space



BOSTON UNIVERSITY

What is missing from the NoDB approach?

Indexing!

How to index?

What to index?

Updates!

NoDB: Efficient Query Execution on Raw Data Files

Ioannis Alagiannis[*] Renata Borovica[*] Miguel Branco[*] Stratos Idreos[‡] Anastasia Ailamaki[*]

[*]EPFL, Switzerland
{ioannis.alagiannis, renata.borovica, miguel.branco, anastasia.ailamaki}@epfl.ch

[‡]CWI, Amsterdam
stratos.idreos@cwi.nl

ABSTRACT

As data collections become larger and larger, data loading evolves to a major bottleneck. Many applications already avoid using database systems, e.g., scientific data analysis and social networks, due to the complexity and the increased *data-to-query* time. For such applications data collections keep growing fast, even on a daily basis, and we are already in the era of *data deluge* where we have much more data than what we can move, store, let alone analyze.

Our contribution in this paper is the design and roadmap of a new paradigm in database systems, called NoDB, which *do not require data loading while still maintaining the whole feature set of a modern database system.* In particular, we show how to make raw data files a first-class citizen, fully integrated with the query engine. Through our design and lessons learned by implementing the NoDB philosophy over a modern DBMS, we discuss the fundamental limitations as well as the strong opportunities that such a research path brings. We identify performance bottlenecks specific for in situ processing, namely the repeated parsing and tokenizing overhead and the expensive data type conversion costs. To address these problems, we introduce an adaptive indexing mechanism that maintains positional information to provide efficient access to raw data files, together with a flexible caching structure.

Our implementation over PostgreSQL, called PostgresRaw, is able to avoid the loading cost completely, while matching the query performance of plain PostgreSQL and even outperforming it in many cases. We conclude that NoDB systems are feasible to design and implement over modern database architectures, bringing an unprecedented positive effect in usability and performance.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems - Query Processing; H.2.8 [**Database Applications**]: Scientific Databases

General Terms

Algorithms, Design, Performance

Keywords

Adaptive loading, In situ querying, Positional map

1. INTRODUCTION

We are now entering the era of data deluge, where the amount of data outgrows the capabilities of query processing technology. Many emerging applications, from social networks to scientific experiments, are representative examples of this deluge, where the rate at which data is produced exceeds any past experience. Scientific analysis such as astronomy is soon expected to collect multiple Terabytes of data on a daily basis, while web-based businesses such as social networks or web log analysis are already confronted with a growing stream of large data inputs. Therefore, there is a clear need for efficient big data processing to enable the evolution of businesses and sciences to the new era of data deluge.

Motivation. Although Database Management Systems (DBMS) remain overall the predominant data analysis technology, they are rarely used for emerging applications such as scientific analysis and social networks. This is largely due to the complexity involved; there is a significant initialization cost in loading data and preparing the database system for queries. For example, a scientist needs to quickly examine a few Terabytes of new data in search of certain properties. Even though only few attributes might be relevant for the task, the entire data must first be loaded inside the database. For large amounts of data, this means a few hours of delay, even with parallel loading across multiple machines. Besides being a significant time investment, it is also important to consider the extra computing resources required for a full load and its side-effects with respect to energy consumption and economical sustainability.

Instead of using database systems, emerging applications rely on custom solutions that usually miss important database features. For instance, declarative queries, schema evolution and complete isolation from the internal representation of data are rarely present. The problem with the situation today is in many ways similar to the past, before the first relational systems were introduced; there are a wide variety of competing approaches but users remain exposed to many low-level details and must work close to the physical level to obtain adequate performance and scalability.

The lessons learned in the past four decades indicate that in order to efficiently cope with the data deluge era in the long run, we will need to rely on the fundamental principles adopted by database management technology. That is, we will need to build extensible systems with declarative query processing and self-managing optimization techniques that will be tailored for the data deluge. A growing part of the database community recognizes this need for

Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing

Matthaios Olma[†] Manos Karpathiotakis[†] Ioannis Alagiannis[‡] Manos Athanassoulis[*] Anastasia Ailamaki[†]

[†]EPFL
{firstname.lastname}@epfl.ch

[‡]Microsoft
ioalagia@microsoft.com

[*]Harvard University
manos@seas.harvard.edu

ABSTRACT

The constant flux of data and queries alike has been pushing the boundaries of data analysis systems. The increasing size of raw data files has made data loading an expensive operation that delays the data-to-insight time. Hence, recent in-situ query processing systems operate directly over raw data, alleviating the loading cost. At the same time, analytical workloads have increasing number of queries. Typically, each query focuses on a constantly shifting – yet small – range. Minimizing the workload latency, now, requires the benefits of indexing in in-situ query processing.

In this paper, we present Slalom, an in-situ query engine that accommodates workload shifts by monitoring user access patterns. Slalom makes on-the-fly partitioning and indexing decisions, based on information collected by lightweight monitoring. Slalom has two key components: (i) an online partitioning and indexing scheme, and (ii) a partitioning and indexing tuner tailored for in-situ query engines. When compared to the state of the art, Slalom offers performance benefits by taking into account user query patterns to (a) *logically* partition raw data files and (b) build for each partition lightweight *partition-specific* indexes. Due to its lightweight and adaptive nature, Slalom achieves efficient accesses to raw data with minimal memory consumption. Our experimentation with both micro-benchmarks and real-life workloads shows that Slalom outperforms state-of-the-art in-situ engines ($3-10\times$), and achieves comparable query response times with fully indexed DBMS, offering much lower ($\sim 3\times$) cumulative query execution times for query workloads with increasing size and unpredictable access patterns.

1. INTRODUCTION

Nowadays, an increasing number of applications generate and collect massive amounts of data at a rapid pace. New research fields and applications (e.g., network monitoring, sensor data management, clinical studies, etc.) emerge and require broader data analysis functionality to rapidly gain deeper insights from the available data. In practice, analyzing such datasets becomes a costly task due to the data explosion of the last decade.

Big Data, Small Queries. The trend of exponential data growth due to intense data generation and data collection is expected to

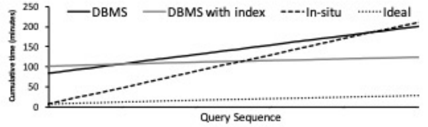


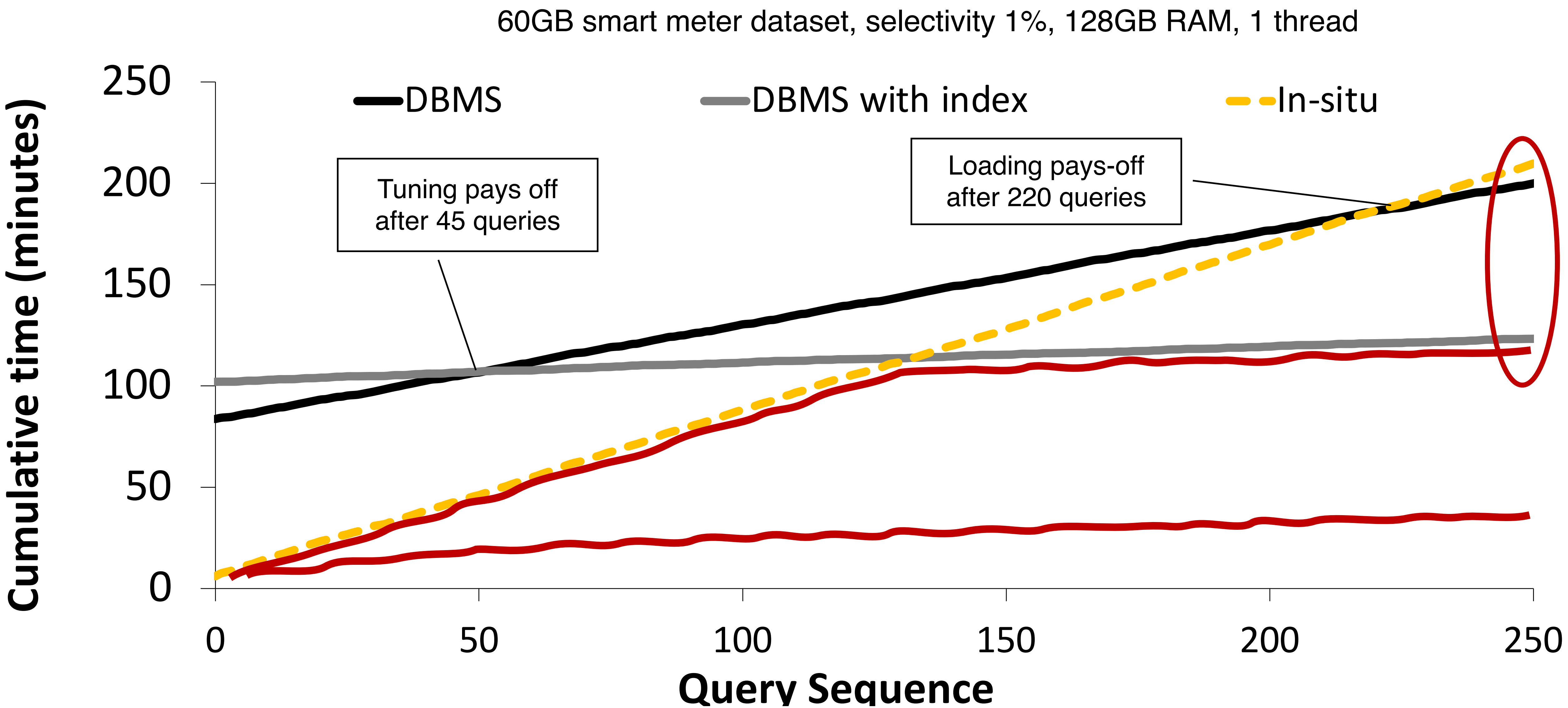
Figure 1: Ideally, in-situ data analysis should be able to retrieve only the relevant data for each query after the initial table scan (ideal - dotted line). In practice today, in-situ query processing avoids the costly phase of data loading (dashed line), however, as the number of the queries increases, the initial investment for full index on a DBMS pays off (the dashed line meets the grey line).

persist, however, recent studies of the data analysis workloads show that typically only a small subset of the data is relevant and ultimately used by analytical and/or exploratory workloads [1, 18]. In addition, modern businesses and scientific applications require interactive data access, which is characterized by *no or little a priori workload knowledge* and constant *workload shifting* both in terms of projected attributes and selected ranges of the dataset.

The Cost of Loading, Indexing, and Tuning. Traditional data management systems (DBMS) require the costly steps of *data loading*, *physical design decisions*, and then *index building* in order to offer interactive access over large datasets. Given the data sizes involved, any transformation, copying, and preparation steps over the data introduce substantial delays before the data can be queried, and provide useful insights [2, 5, 34]. The lack of a priori knowledge of the workload makes the physical design decisions virtually impossible because cost-based advisors rely heavily on past or sample workload knowledge [3, 17, 22, 29, 58]. The workload shifts observed in the interactive setting of exploratory workloads can nullify investments towards indexing and other auxiliary data structures (e.g., views), since frequently, they depend on the actual data values and the knowledge generated by the ongoing analysis.

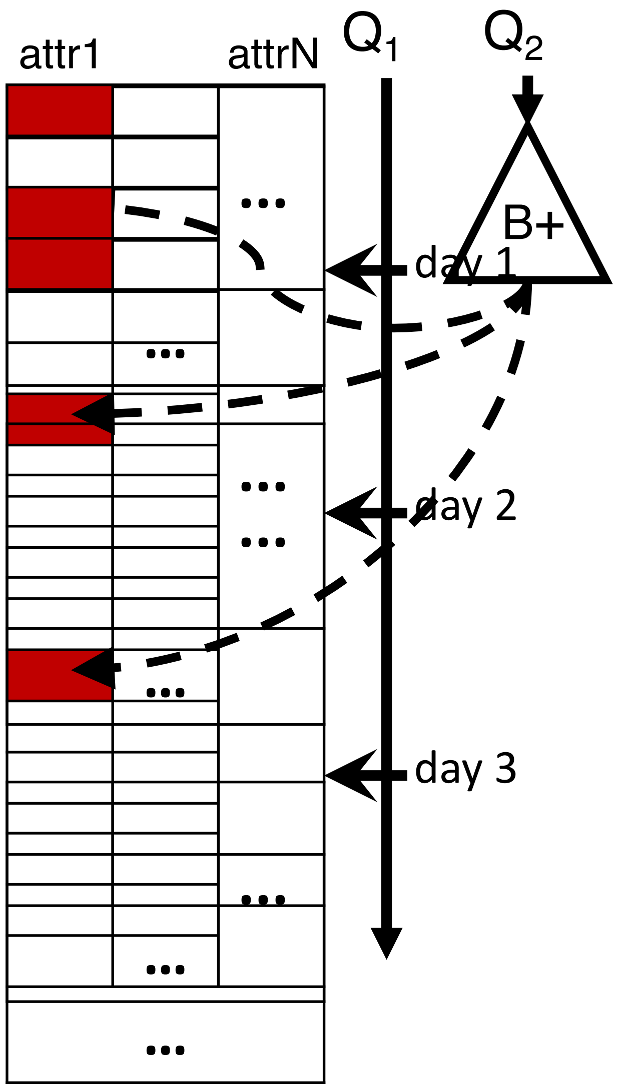
Querying Raw Data Files Is Not Enough. Recent efforts opt to query directly raw files [2, 5, 13, 19, 30, 40] to reduce the data-to-insight cost. These *in-situ* systems avoid the costly initial data loading step, and allow the execution of declarative queries over external files without duplicating or "locking" data in a proprietary database format. Further, they concentrate on reducing costs as-

Reducing data to query time



Ideal: instant access to data & interactive response to queries

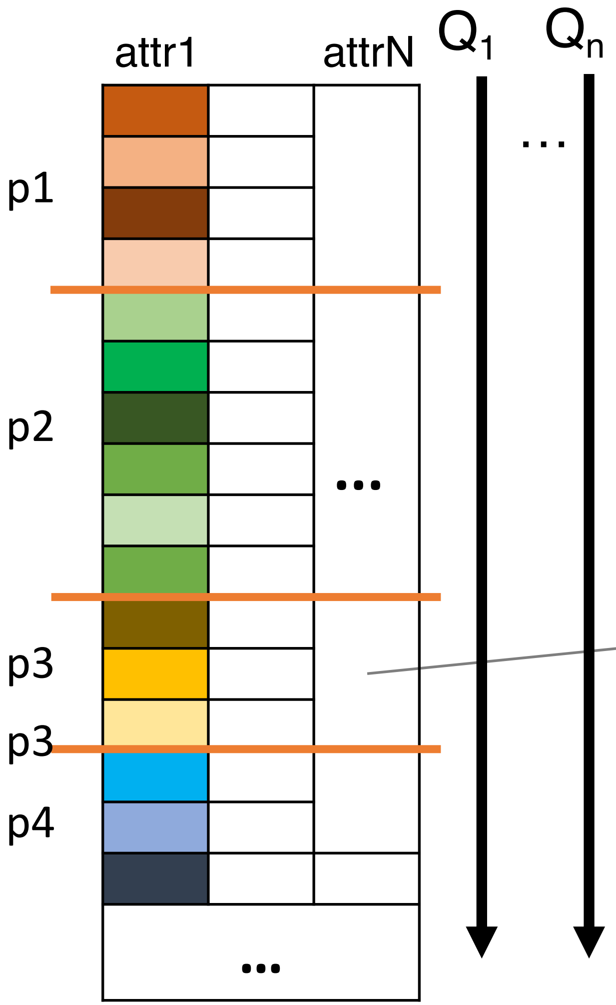
Interactive in situ query processing



- **Partitioning**: Shared data ownership
  - Physical restructuring prohibited
- **Indexing**: Depends on workload
  - A priori index tuning is impossible for exploratory workloads
- **Updates** in file interrupt in situ query processing

Incrementally tune only useful data

Adaptive logical partitioning



Enable data skipping

Fine-grained access path selection

Iteratively partition dataset → Query-based / Homogeneous

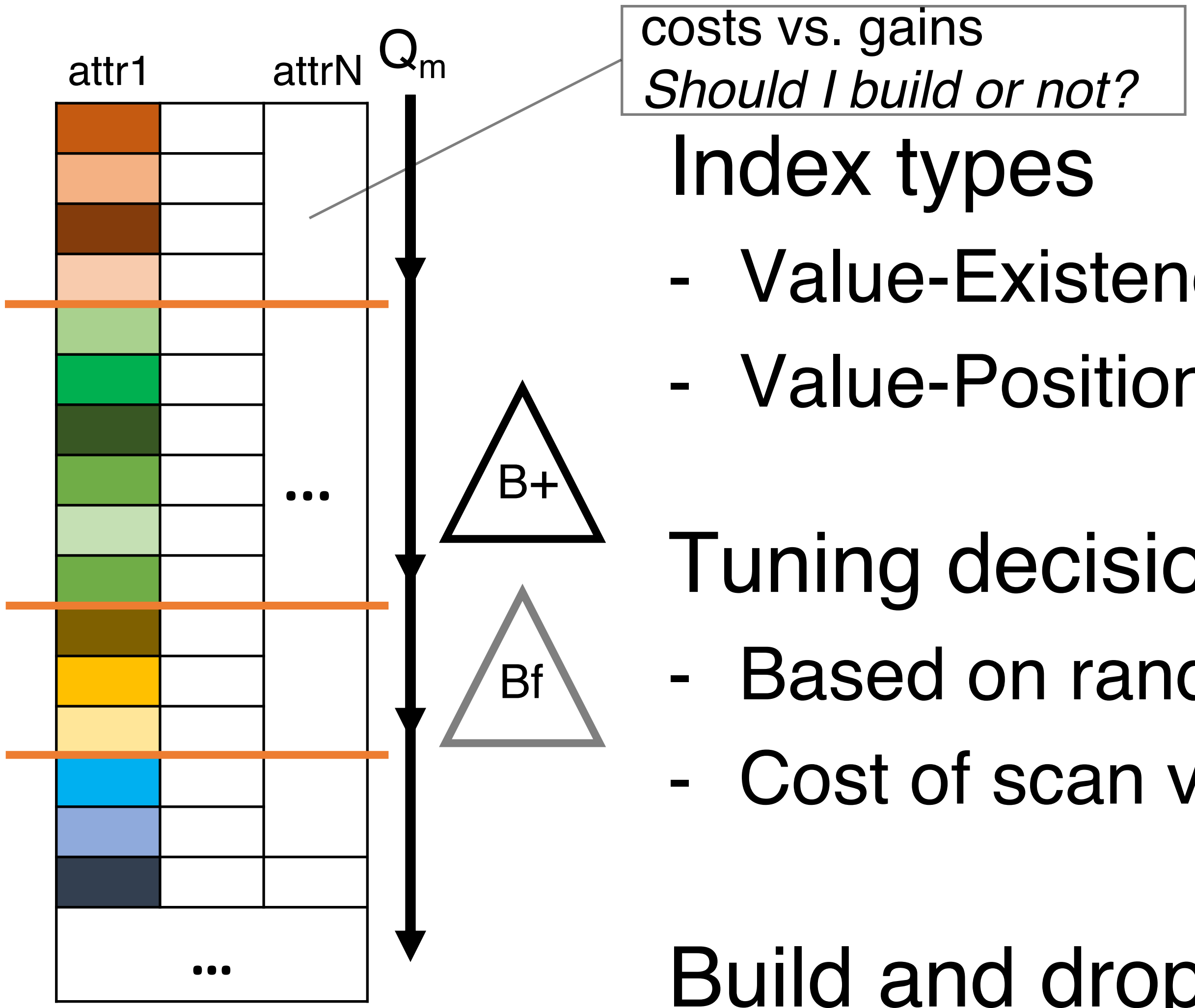
1) Collect data statistics at runtime
2) Calculate number of sub-partitions

Increase disjointness: Reduce distinct values

Remove tails: Reduce excess kurtosis

Set the ground for reducing data access

Online index tuning



Index types

- Value-Existence (i.e., Bloom filters)
- Value-Position (i.e., B+ Trees)

Tuning decision

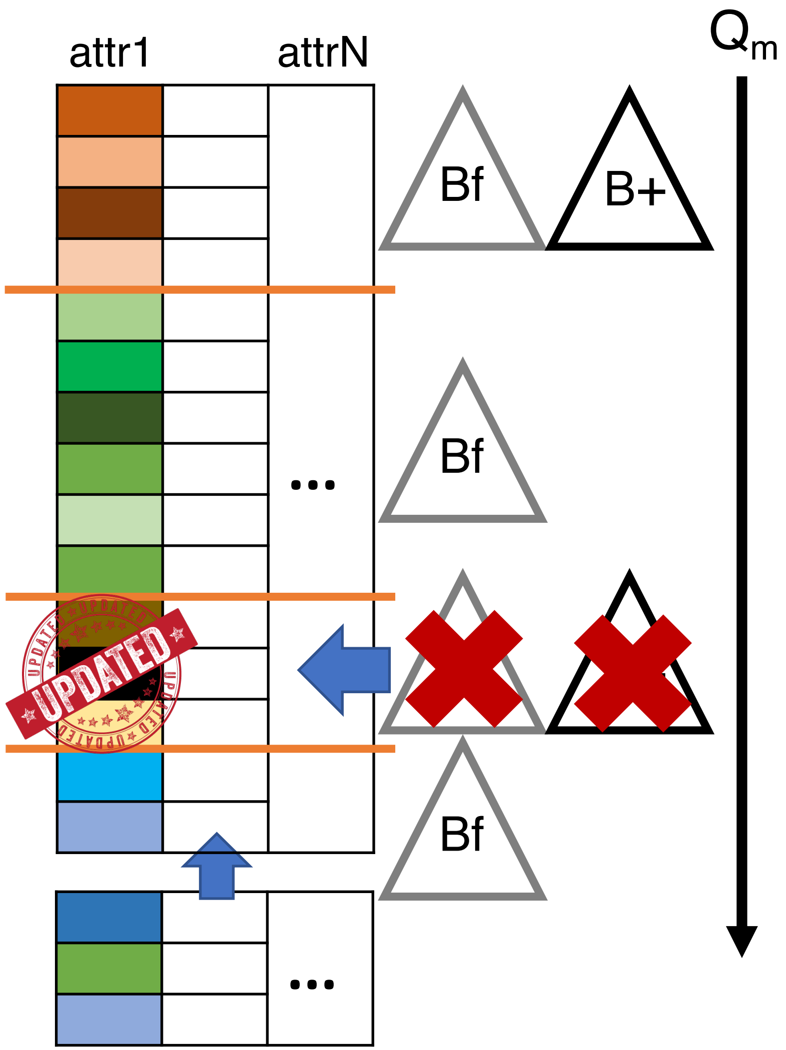
- Based on randomized algorithm
- Cost of scan vs. cost of build + gain

Build and drop based on budget

BOSTON UNIVERSITY

Maximize gain: build cost vs performance

Append and in place updates



Store partition state

- Calculate hash value (MD5)

Monitor file for modifications

Recognize updated partitions

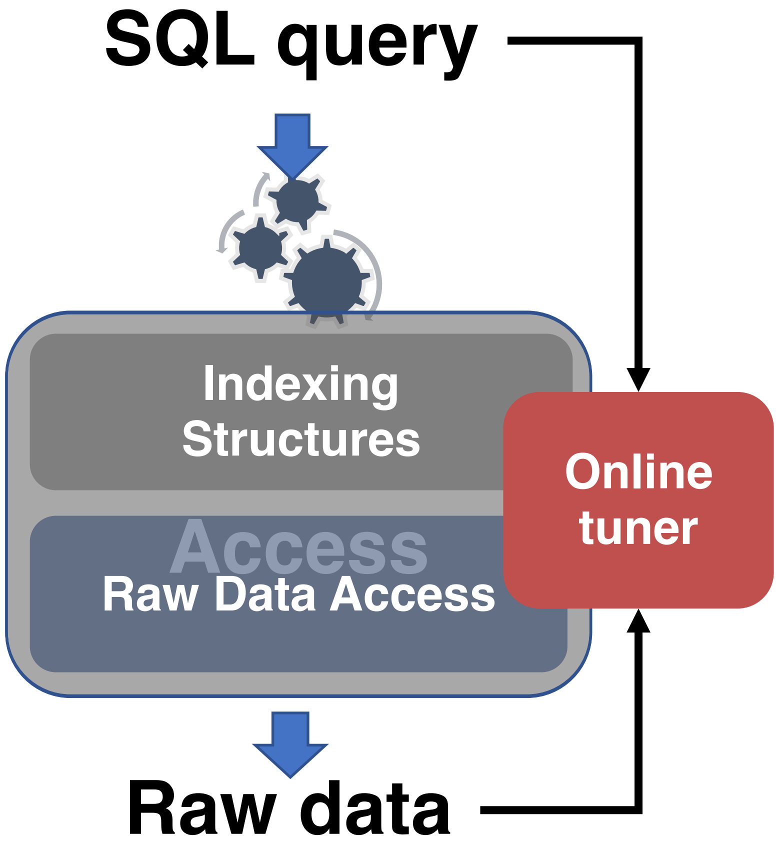
Fix modified partitions

- Drop/Re-build cache/index

BOSTON UNIVERSITY

Minimize overhead of updates

Slalom architecture



Incremental logical partitioning

- Based on data distribution

Adaptive partition indexing

- Based on access patterns

Monitors data for updates

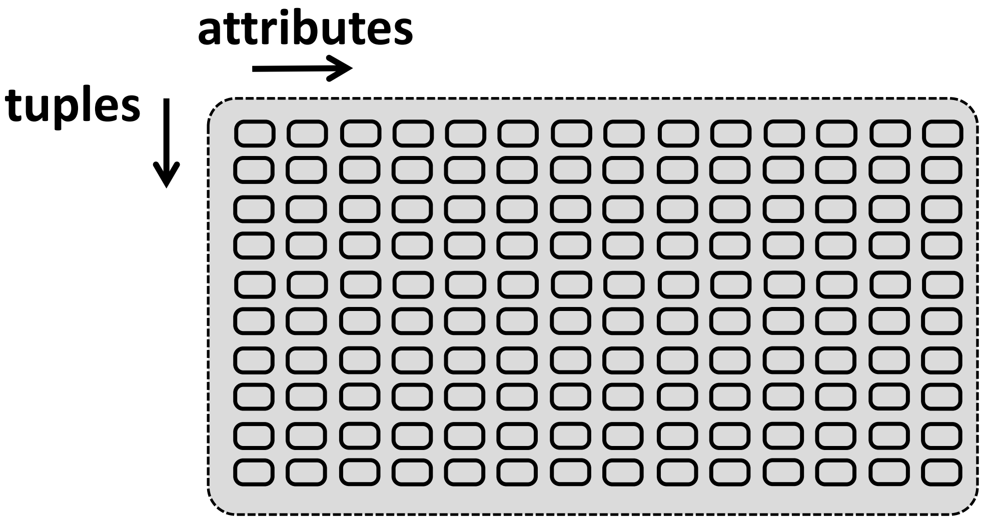
- Updates data structures

Combining Online Tuning with Adaptive Indexing

Adapt data access to queries and data at runtime

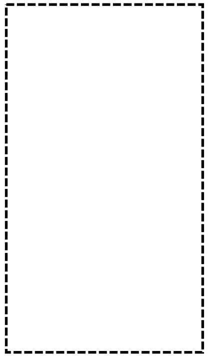
BOSTON UNIVERSITY

CSV Positional Index

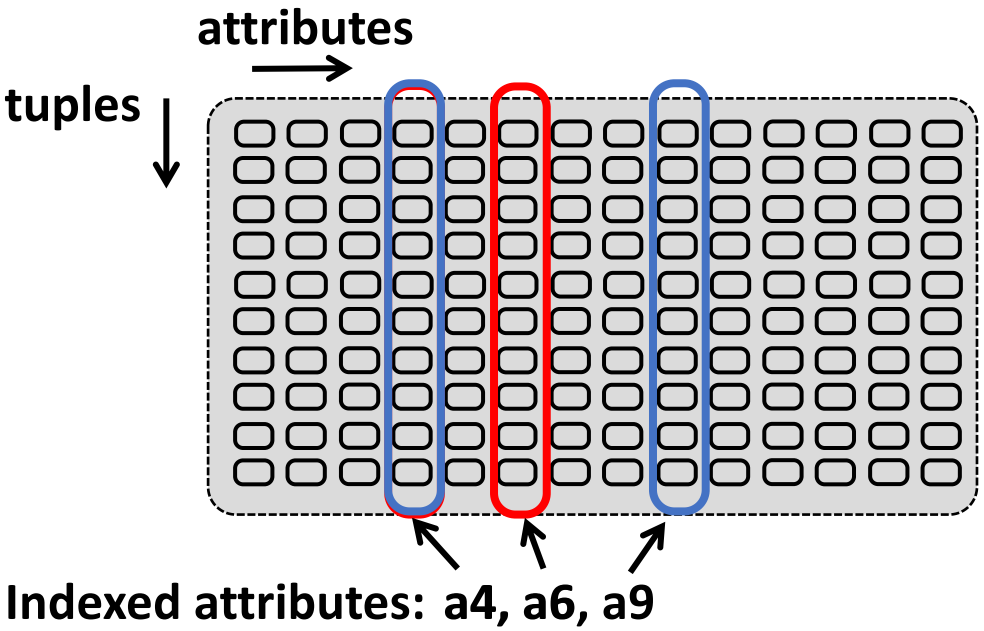


1. Positional index is empty

Indexed attributes:



CSV Positional Index



p4, p6
p4, p6
p4, p6
p4, p6

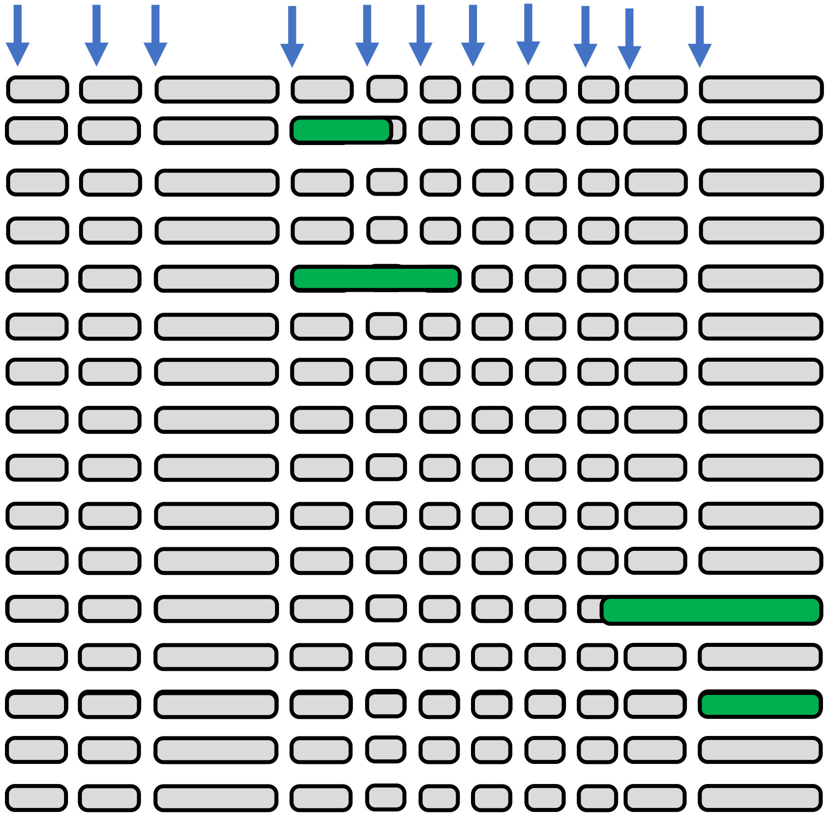
→

p4, p6, p9
p4, p6, p9
p4, p6, p9
p4, p6, p9

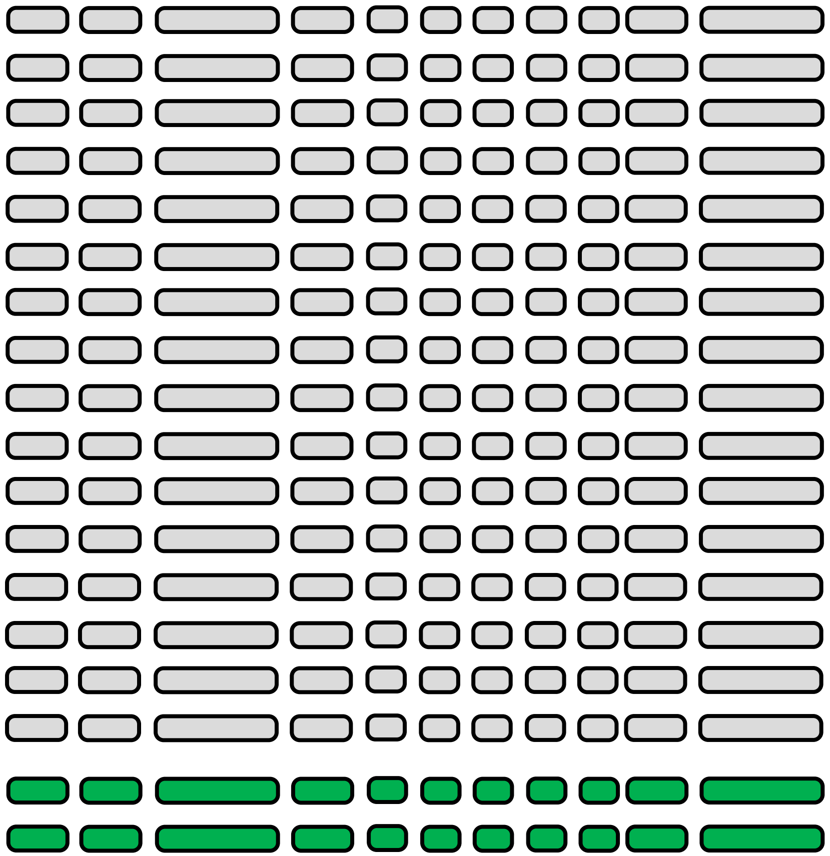
1. Positional index is empty
2. Q1 accesses *a4* and *a6*
3. Q2 accesses *a4* and *a9*

BOSTON UNIVERSITY

Types of updates



In-place update



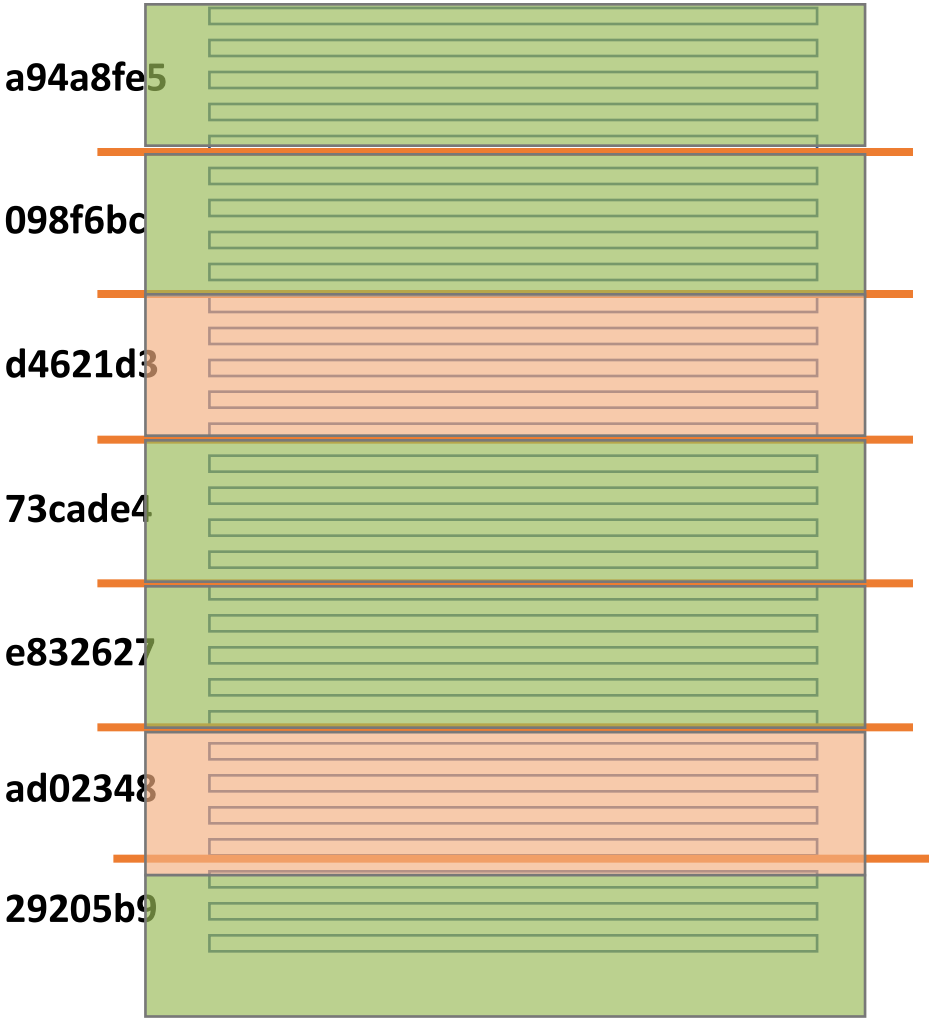
Append

Goal: Efficiently correct the auxiliary structures

BOSTON UNIVERSITY

Identifying in-place updates

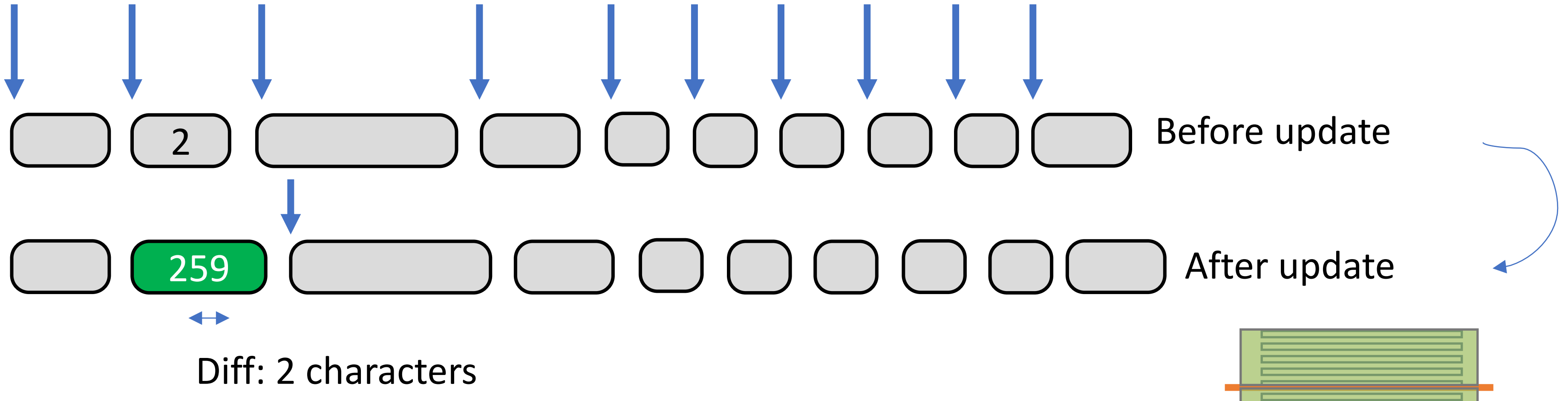
- Store partition state
  - Calculate MD5 hash
- Monitor file
  - Using OS support (iNotify)
- Find updated partitions
  - Calculate new MD5 hashes
  - Compare with previous state



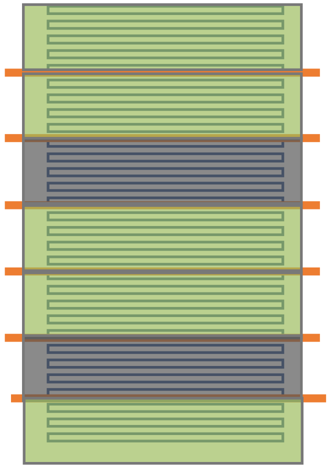
BOSTON UNIVERSITY

Fixing the touched partitions

- We find the diff offsets using the PM
- We store this diff in a separate array
  - Using it when fetching records from file

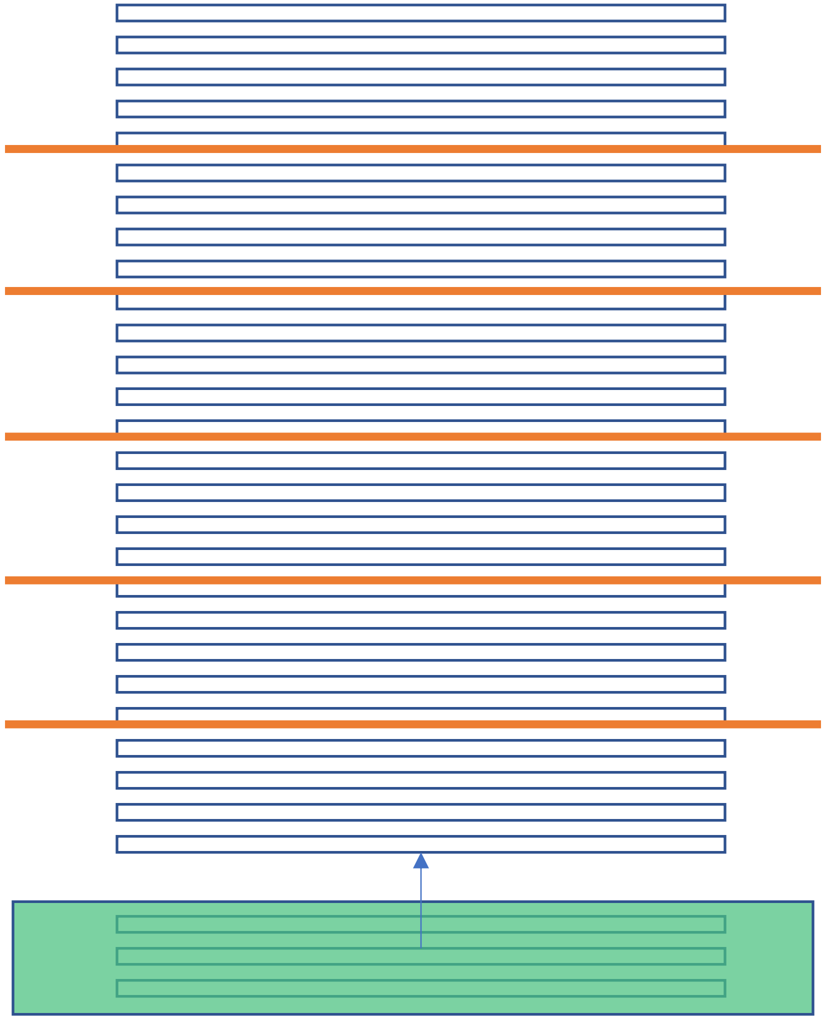


- Auxiliary structures are dropped for the touched partitions.



Identifying append-like updates

- Add new rows to a new partition
- Further split new partitions
  - Statically
  - Dynamically



BOSTON UNIVERSITY

Experimental setup

Hardware:

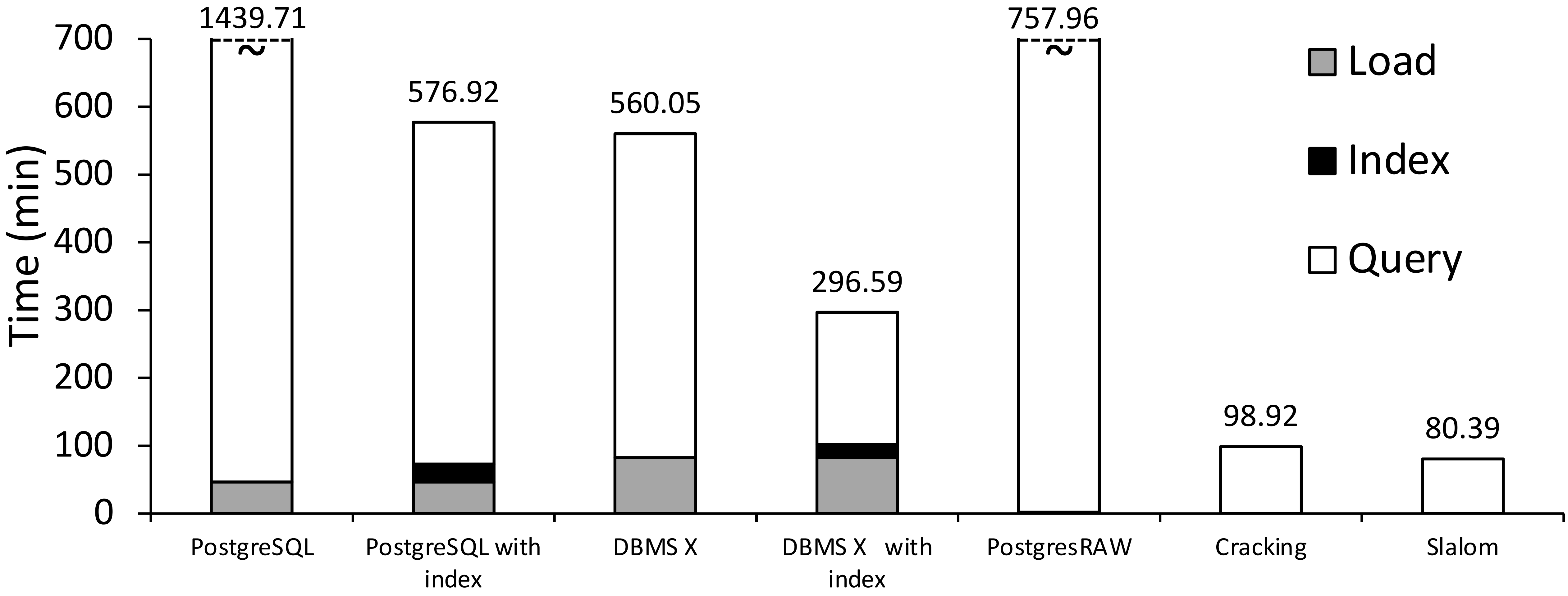
- Xeon CPU E5-2660 @ 2.20GHz, 2TB HDD - 7200RPM, 128GB RAM

Systems:

- Disk-based: PostgreSQL
- In-Memory: DBMS X
- In situ: PostgresRAW, Slalom with Stochastic Cracking

From raw data to results

59GB uniform dataset, 128GB RAM, cold caches,1000 point & range queries, selectivity: 0.5%-5%

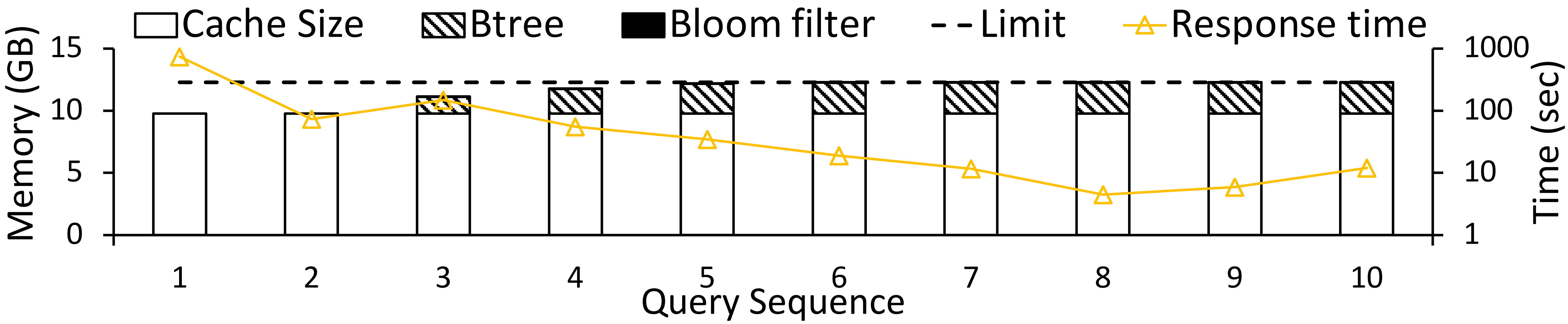


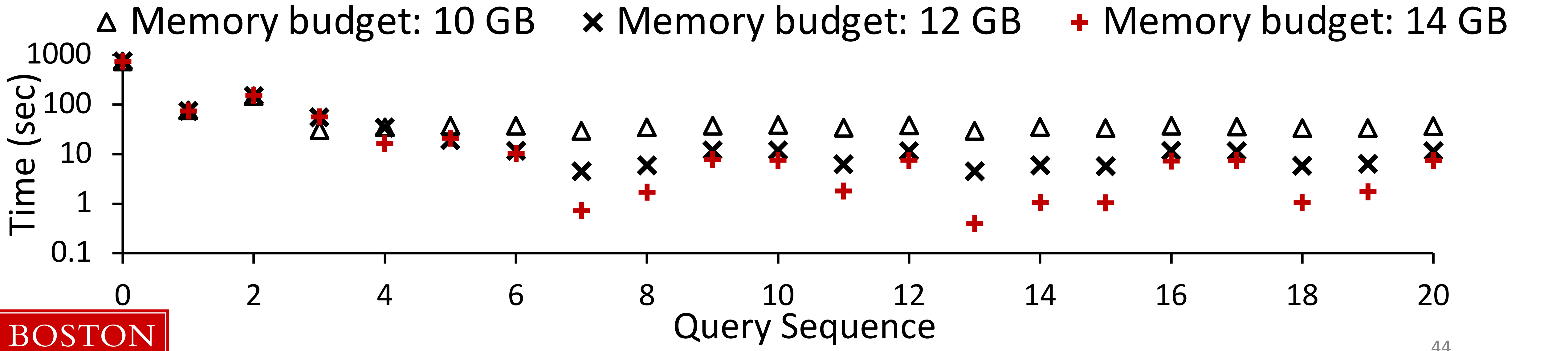
BOSTON UNIVERSITY

In-situ adaptive indexing achieves interactive access

Working under memory constraints

55GB uniform dataset, 128GB RAM, cold caches, selectivity: 0.1% (select 10 consecutive values)

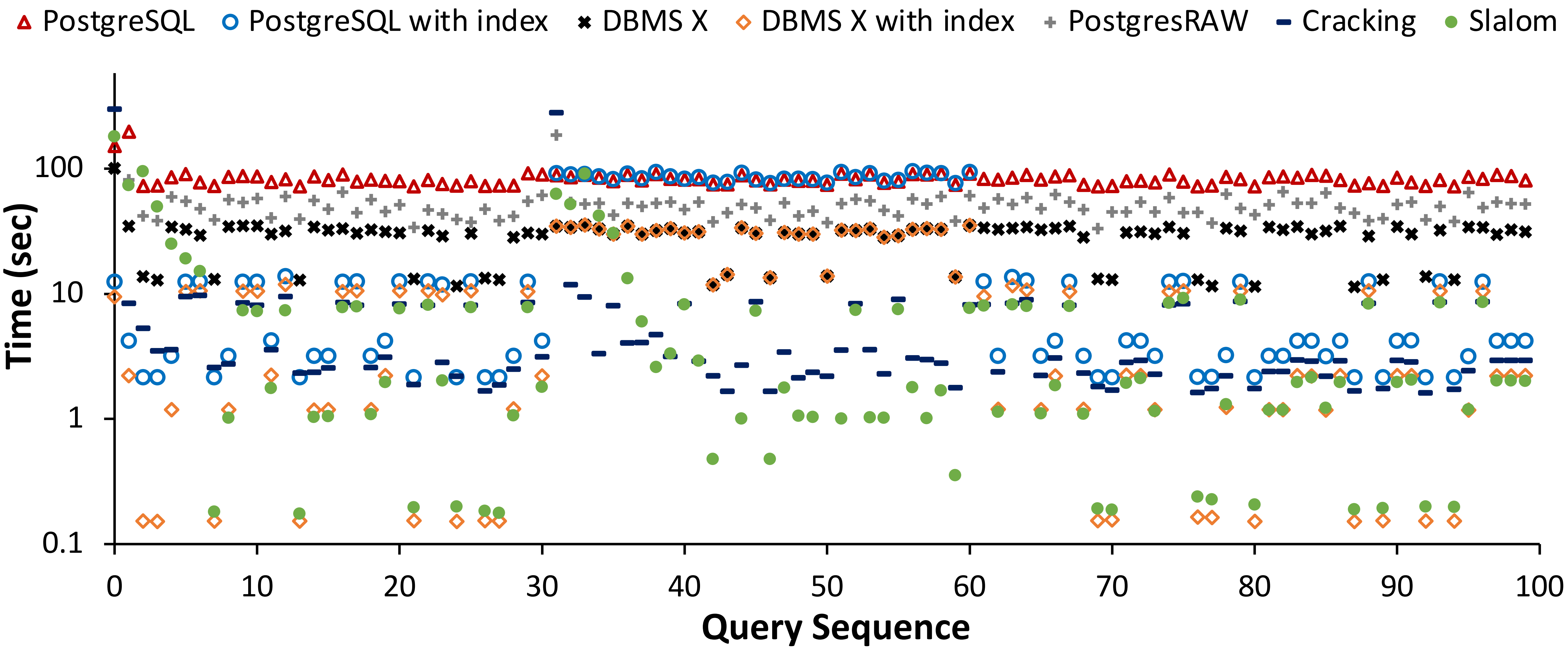




BOSTON UNIVERSITY

Uniform data query sequence

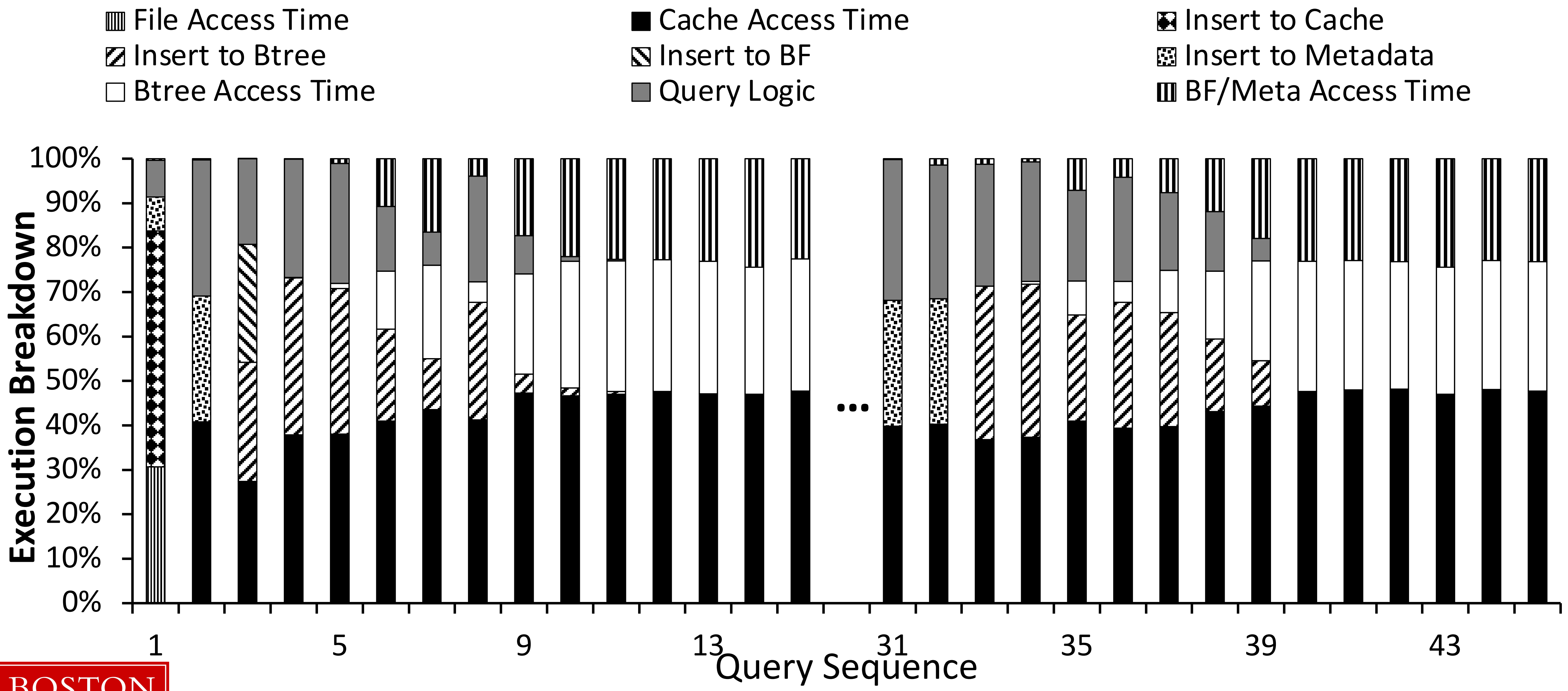
59GB uniform dataset, 128GB RAM, cold caches,100 point & range queries, selectivity: 0.5%-5%



BOSTON UNIVERSITY

Execution breakdown

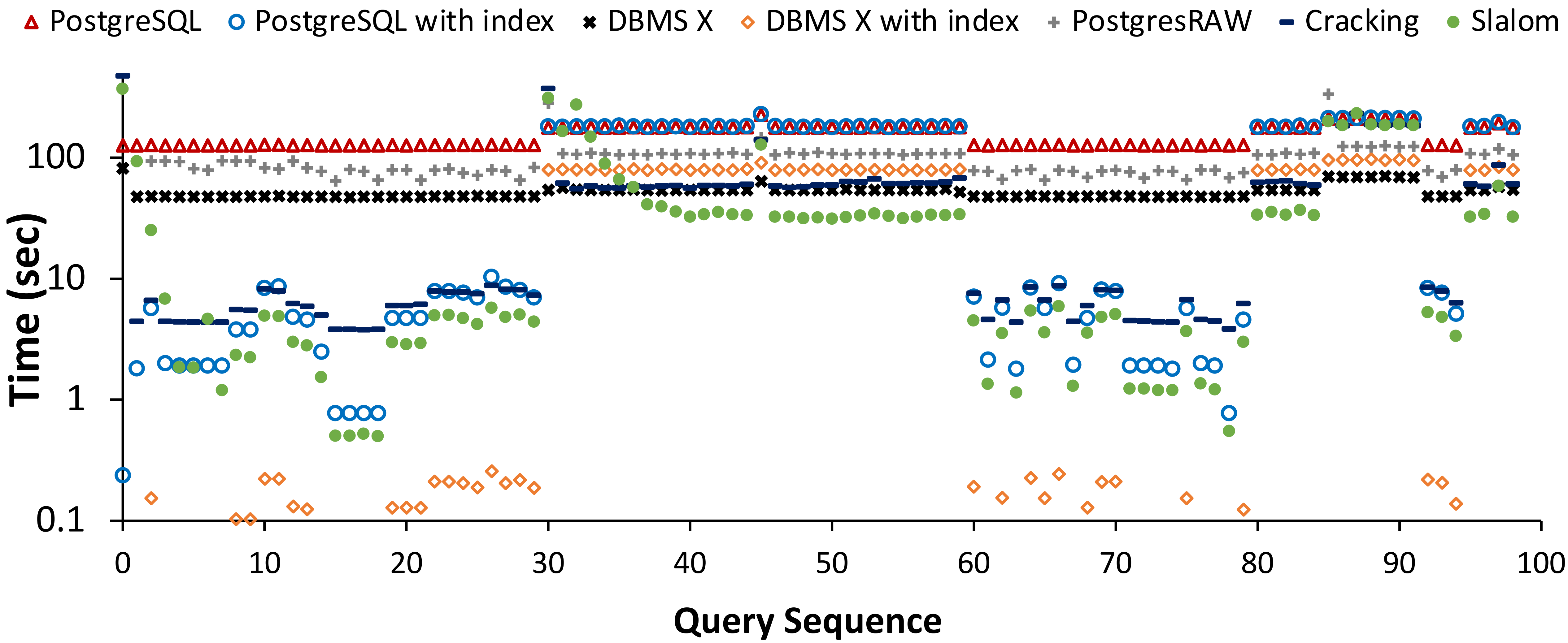
59GB uniform dataset, 128GB RAM, cold caches,1000 point & range queries, selectivity: 0.5%-5%



BOSTON UNIVERSITY

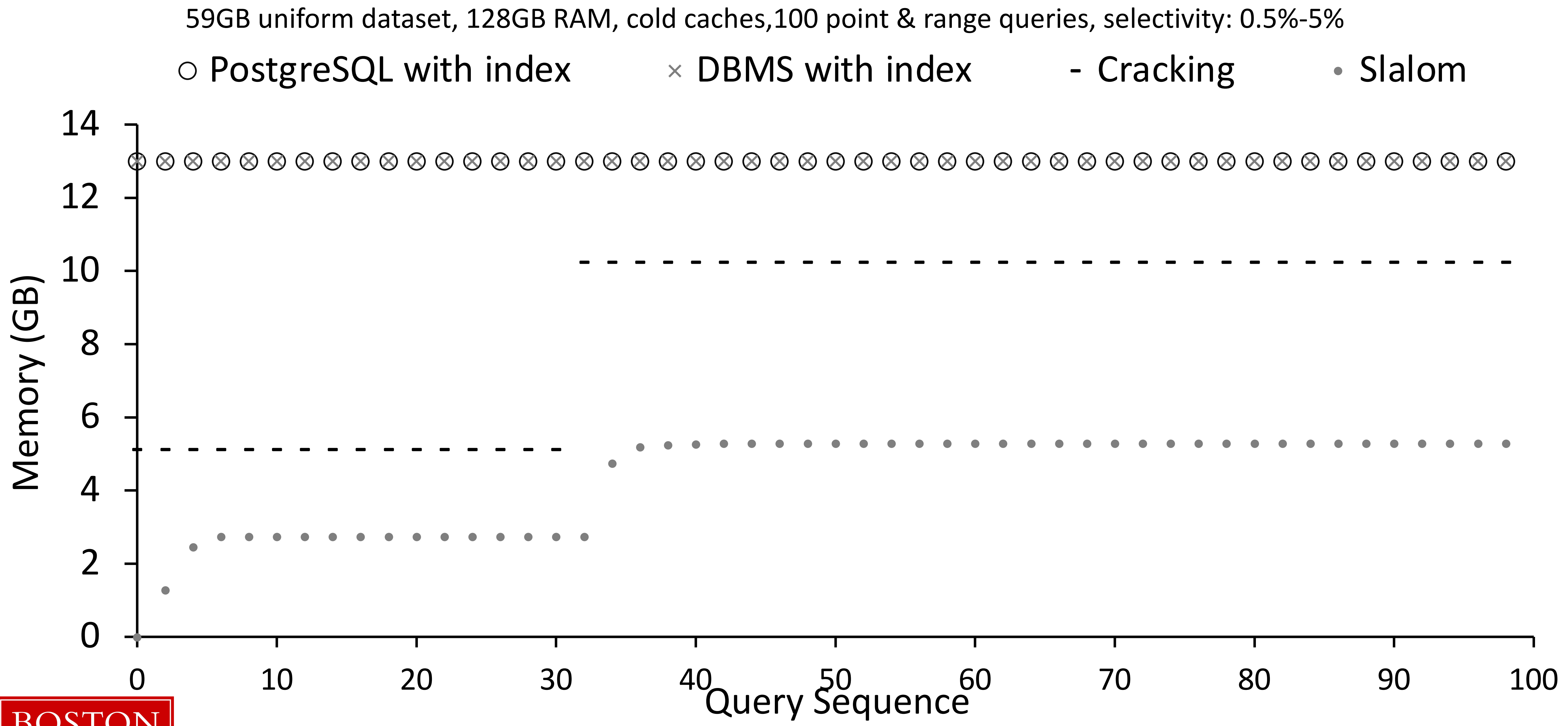
Smart meter workload query sequence

59GB uniform dataset, 128GB RAM, cold caches,100 point & range queries, selectivity: 0.5%-5%



BOSTON UNIVERSITY

Memory consumption

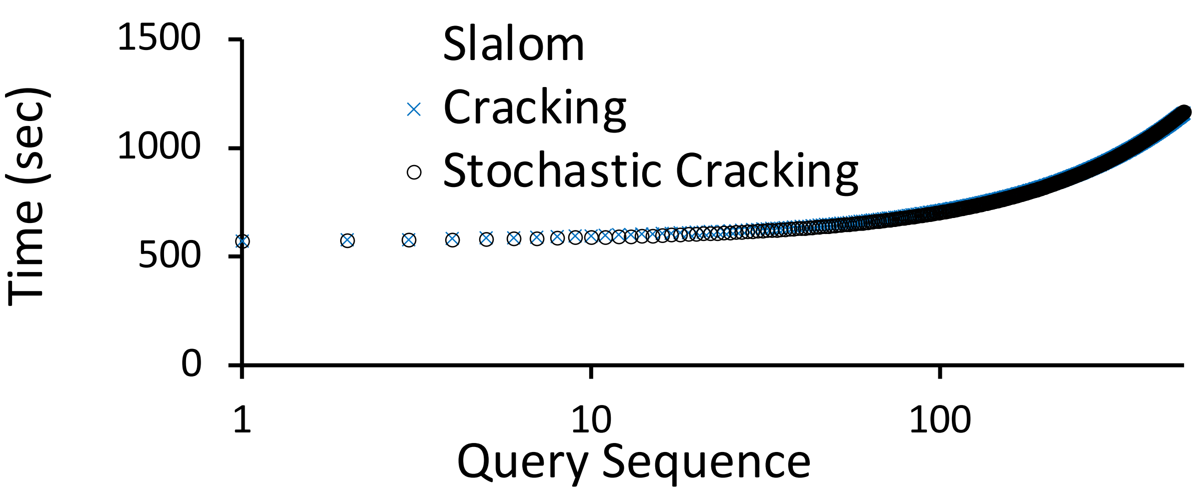


BOSTON UNIVERSITY

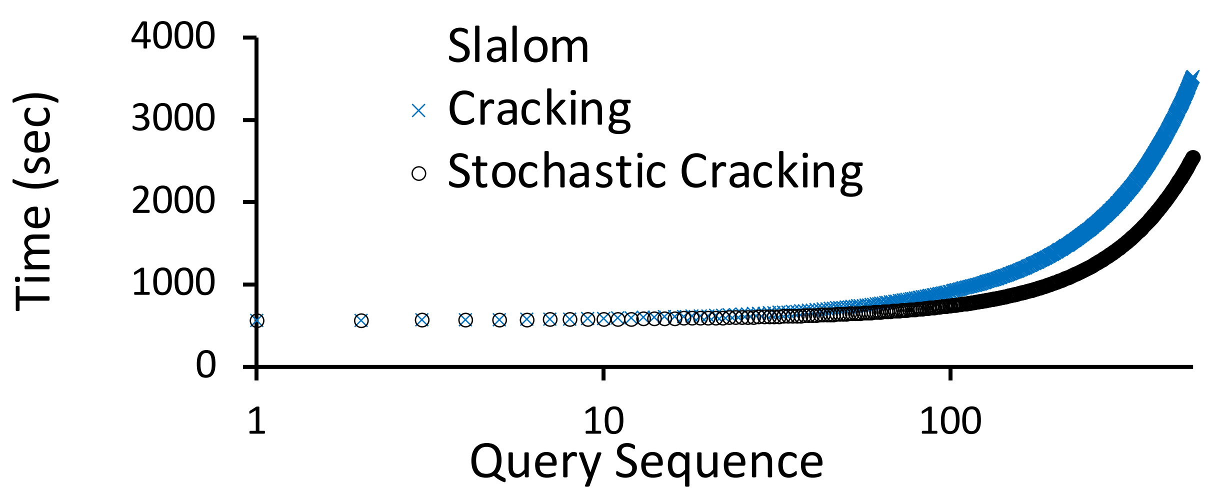
Comparing Cracking to Slalom

59GB uniform dataset, 128GB RAM, cold caches,1000 point & range queries, selectivity: 0.5%-5%

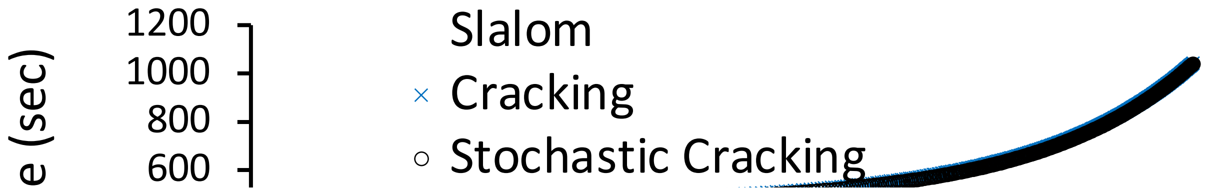
Random access/Uniform data



Sequential access/Uniform data



Random access/Clustered data



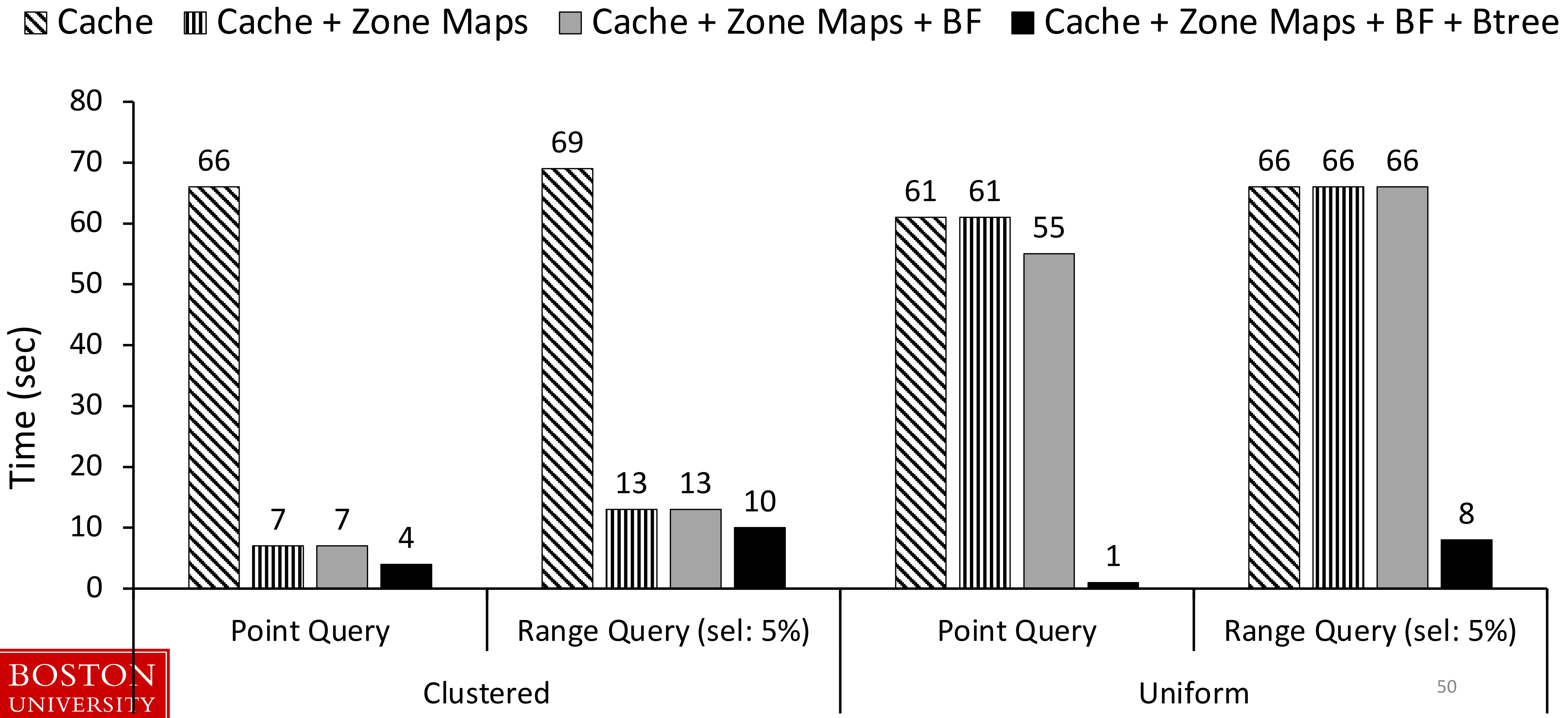
Memory Footprint

Slalom takes advantage of the underlying data

Cracking converges faster to final state

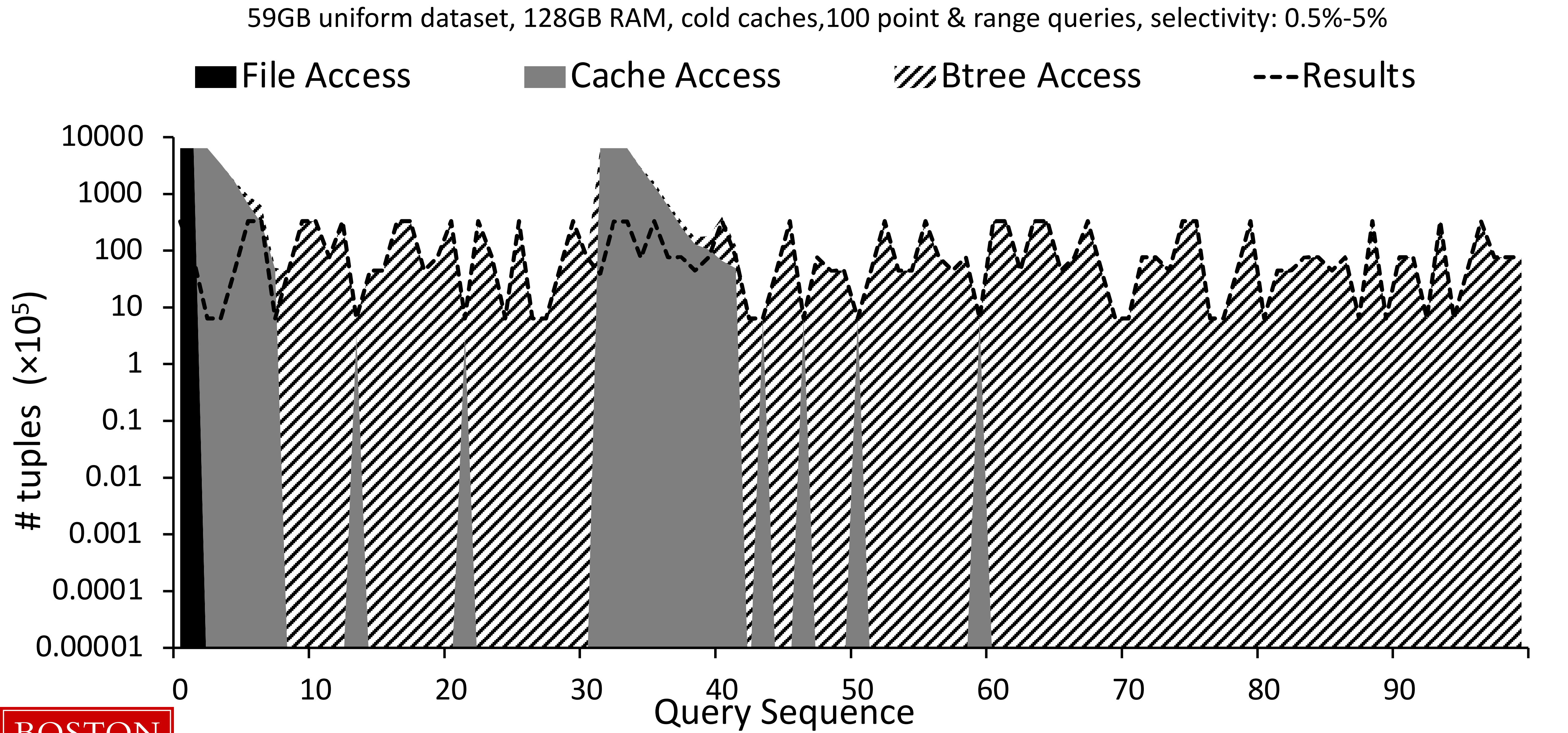
Minimizing data access

59GB uniform dataset, 128GB RAM, cold caches, selectivity: 0.5%-5%



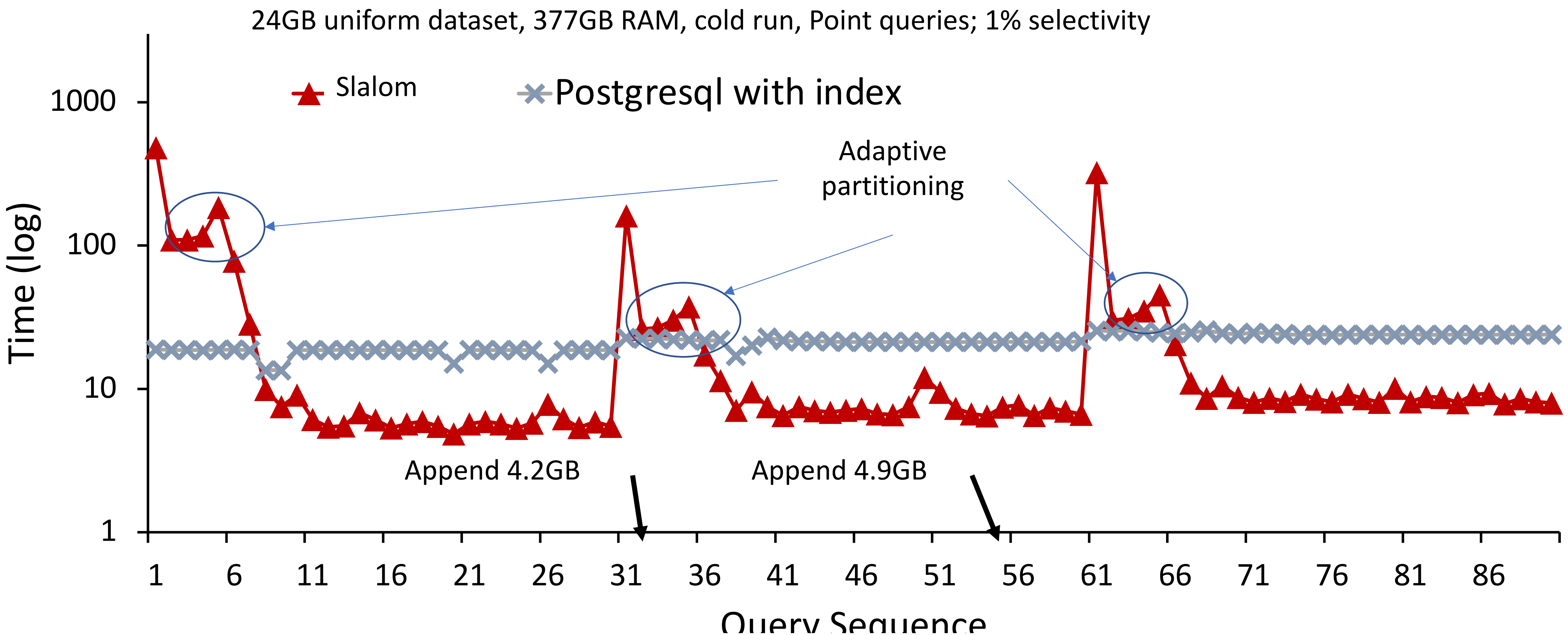
BOSTON UNIVERSITY

Access path used to access tuples



BOSTON UNIVERSITY

Append-like updates



Slalom adapts partitioning after an append

It offers competitive performance to a loaded system

Takeaways from Slalom

Speed-up in situ query processing

- Take advantage of data distribution when tuning databases

Online logical partitioning algorithm

- Extract logical clustering within the data

Low-overhead online fine-grained index selection

- Using a randomized algorithm

Performance comparable to in-memory DBMS

- 3x lower cumulative exec. time

CS 561: Data Systems Architectures

class 20

In-Situ Data Processing

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/