BoDS: A Benchmark on Data Sortedness

Aneesh Raman
aneeshr@bu.edu

Konstantinos Karatsenidis
karatse@bu.edu

Subhadeep Sarkar
ssarkar1@bu.edu

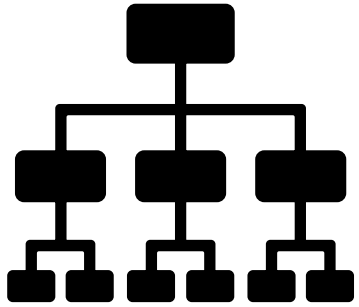
Matthaios Olma
maolma@microsoft.com

Manos Athanassoulis
mathan@bu.edu

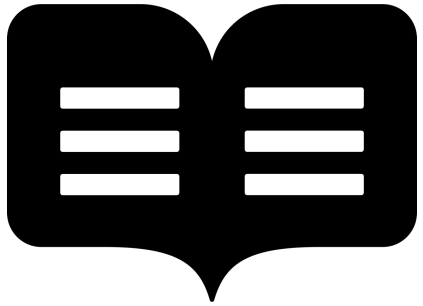
BOSTON UNIVERSITY

DiSC lab

Indexes in Databases

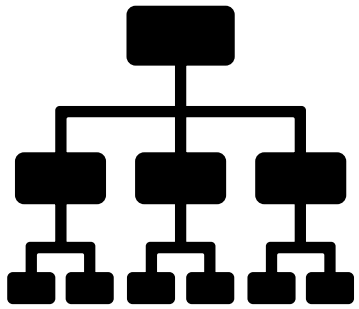


organize data

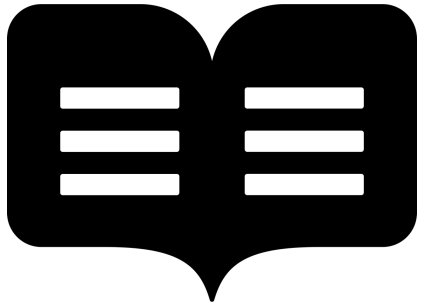


efficient queries

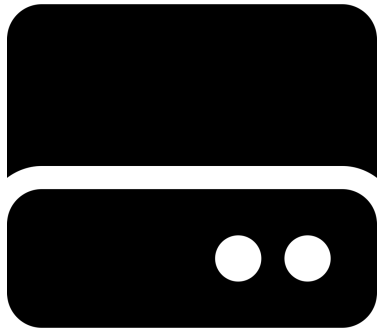
Indexes in Databases



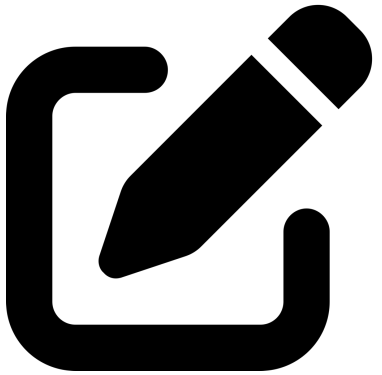
organize data



efficient queries



space amplification

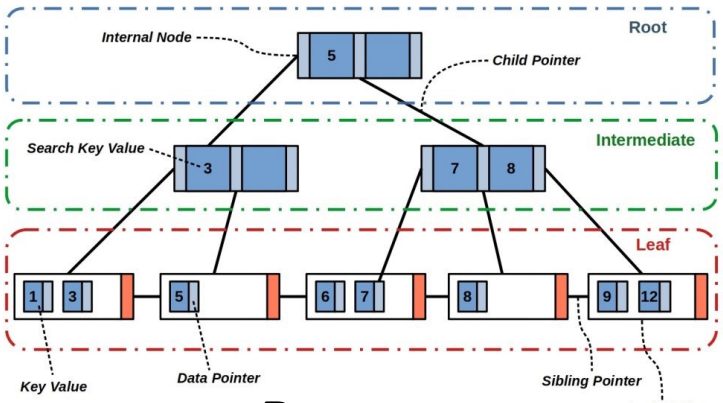


write amplification

What is Indexing?

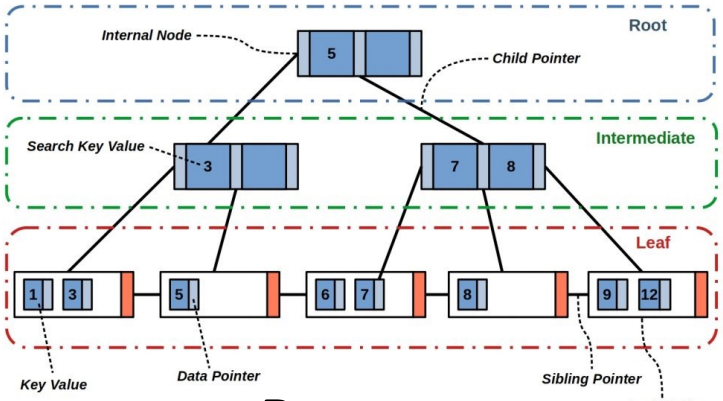
The process of inducing “sortedness” to an otherwise unsorted data collection

Indexes in Databases

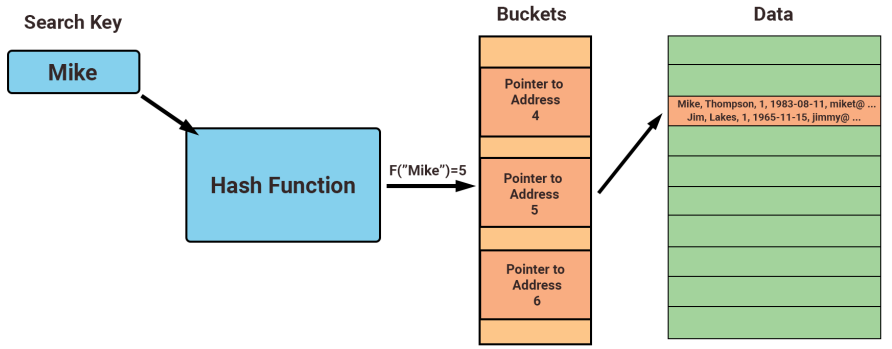


B-trees

Indexes in Databases

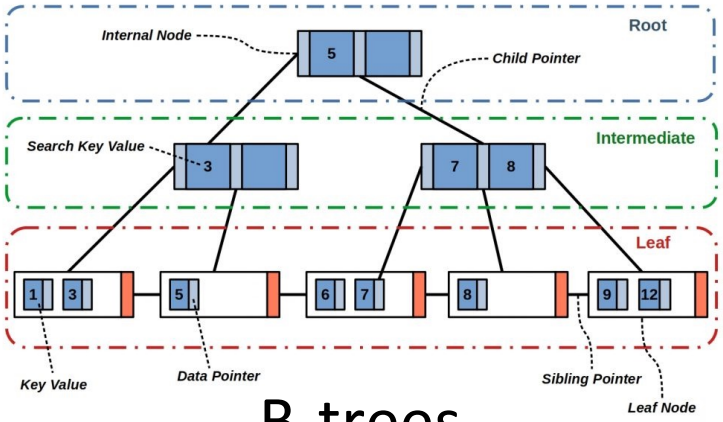


B-trees

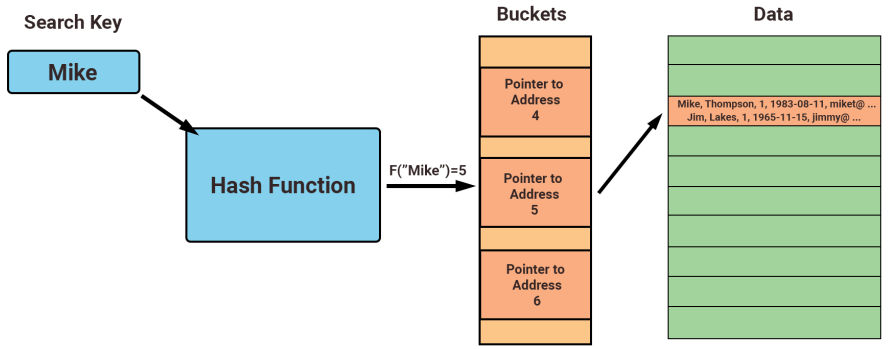


Hash maps

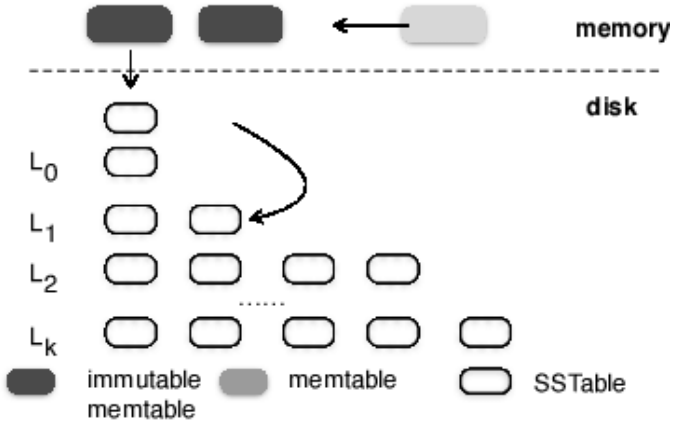
Indexes in Databases



B-trees

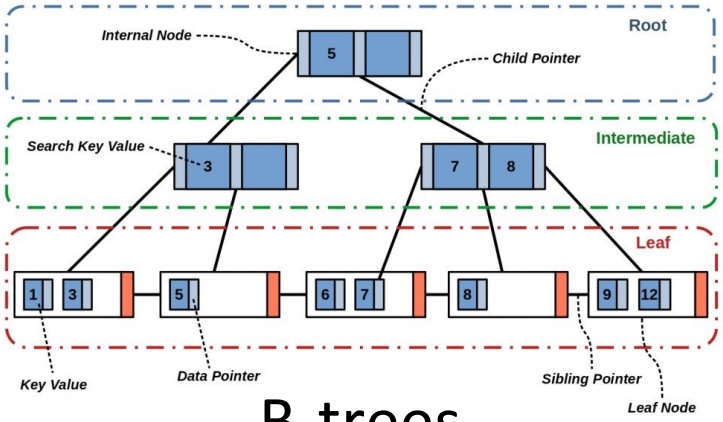


Hash maps

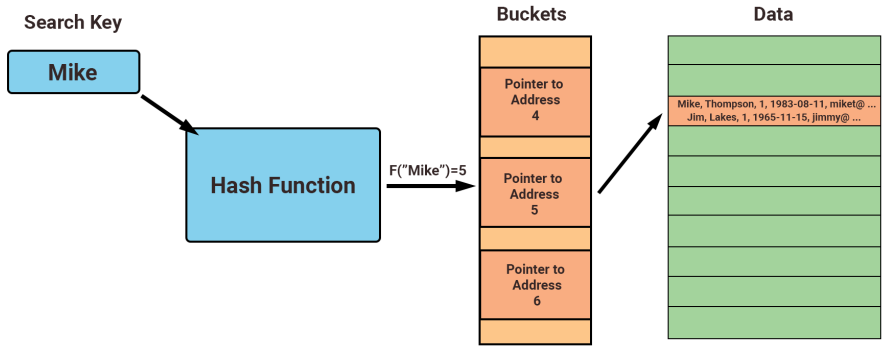


LSM-trees

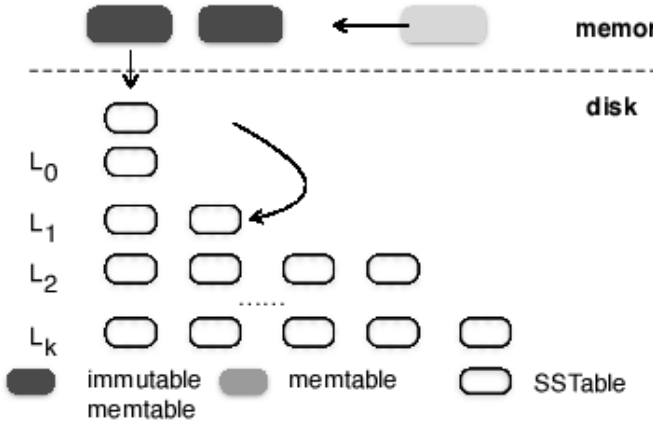
Indexes in Databases



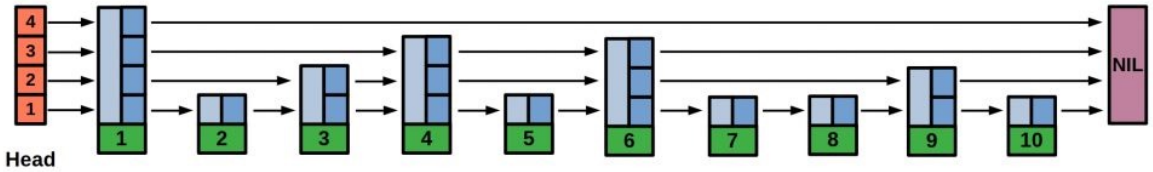
B-trees



Hash maps



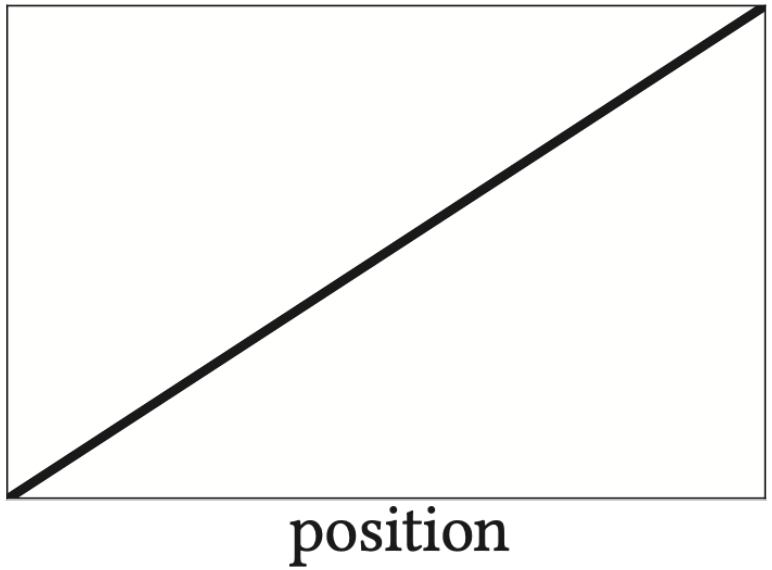
LSM-trees



Skip lists

Index Construction

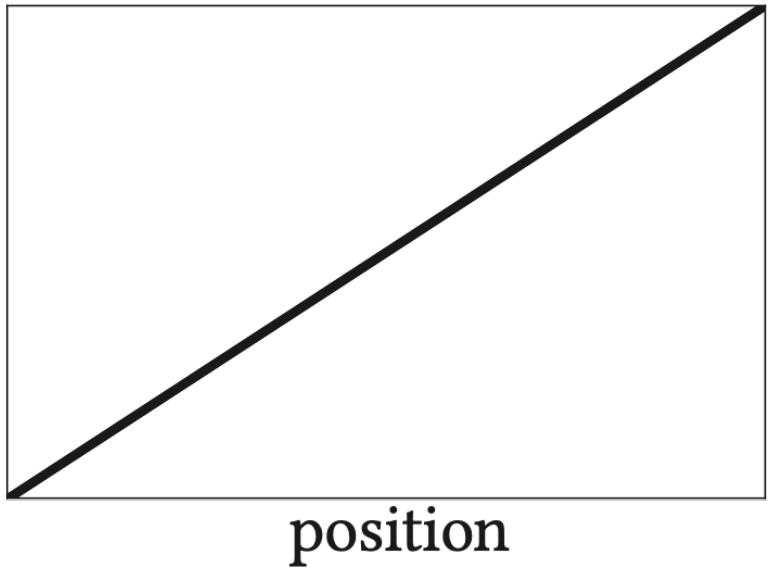
Sorted Data



Faster ingestion
(e.g., bulk loading)

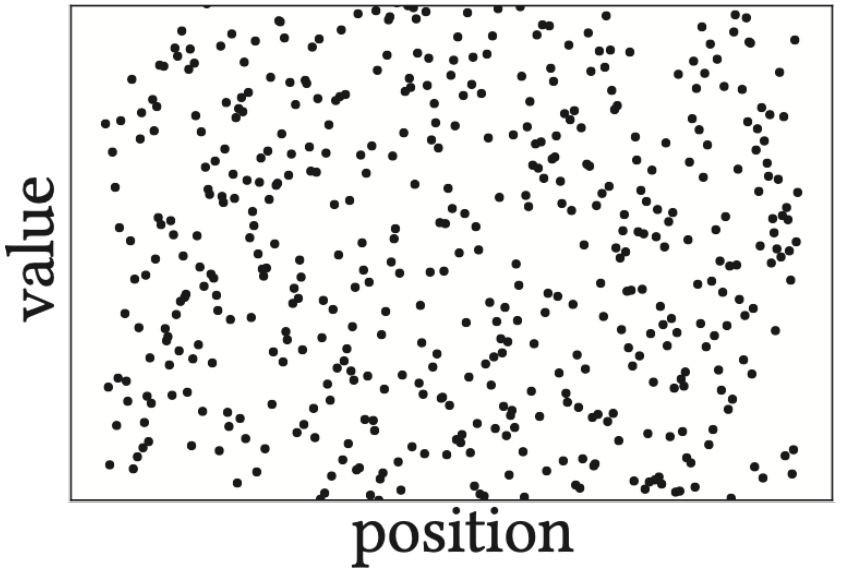
Index Construction

Sorted Data



Faster ingestion
(e.g., bulk loading)

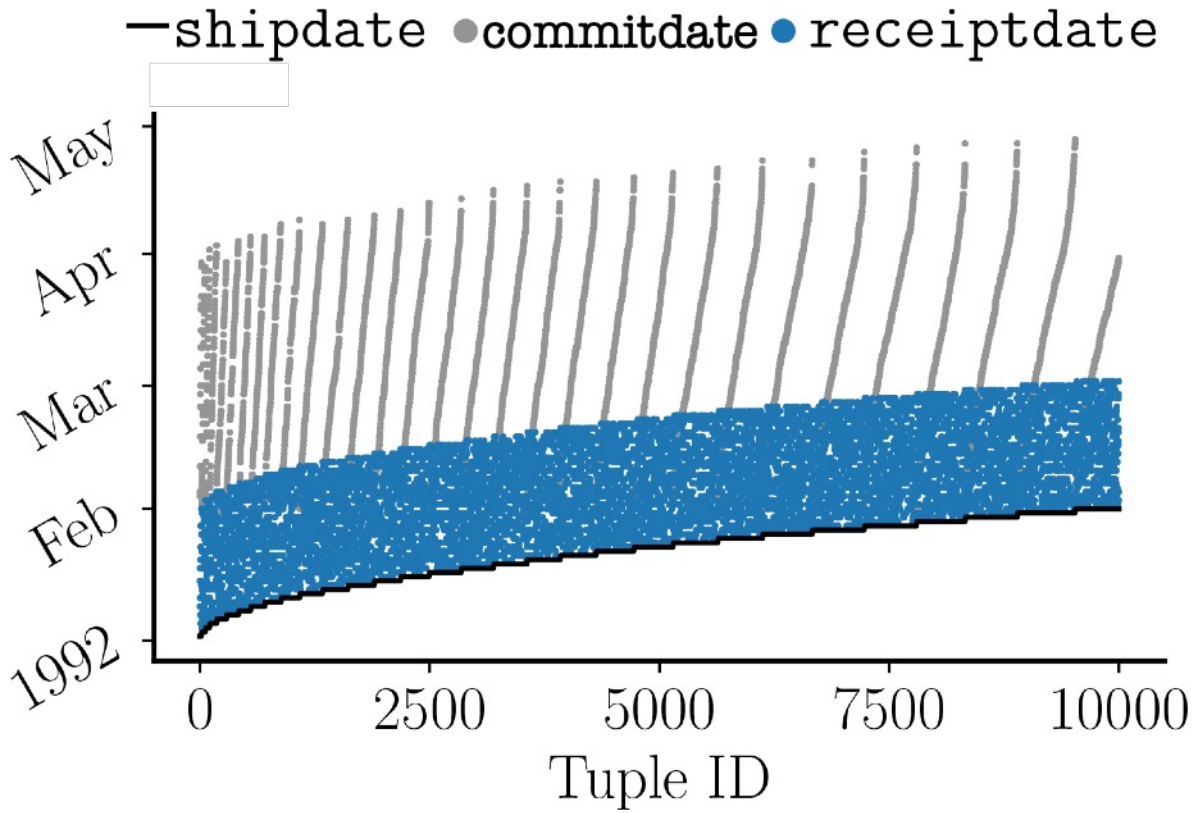
Scrambled Data



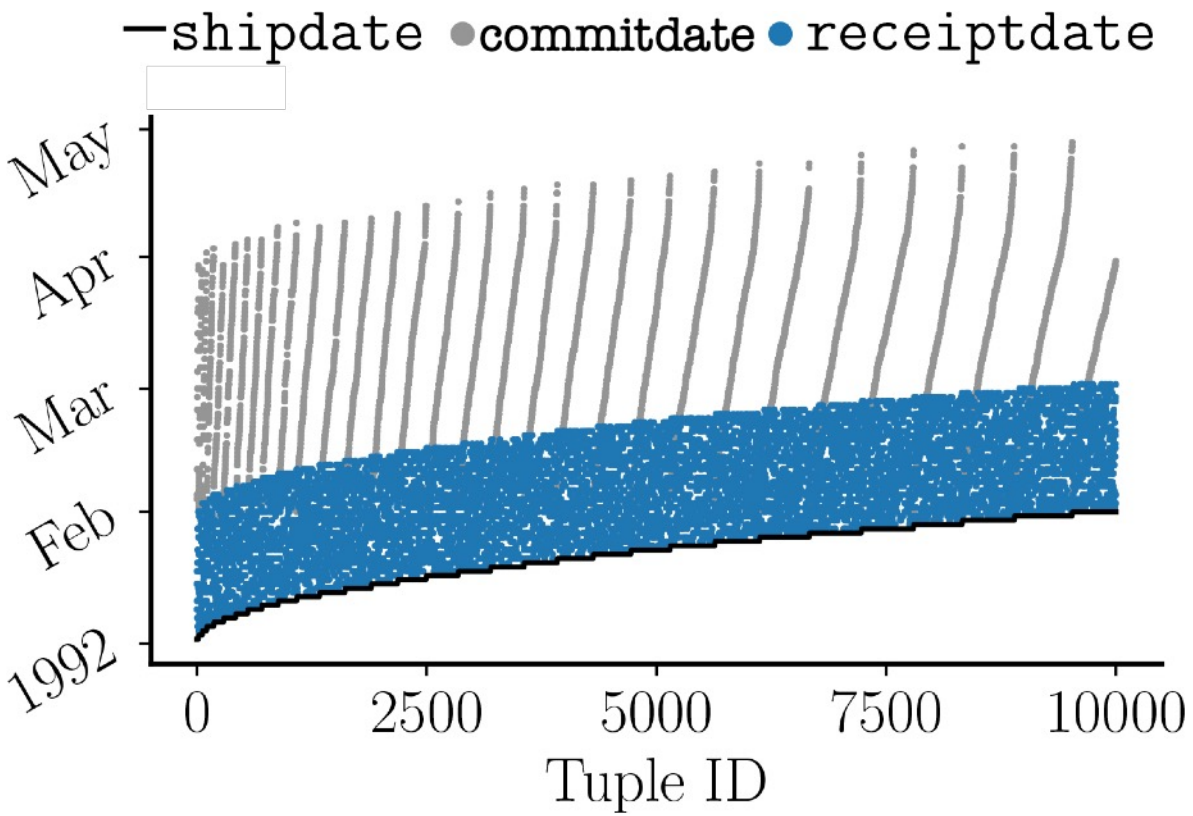
Standard
ingestion

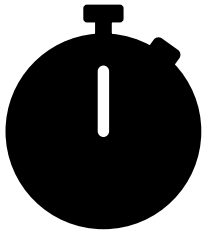
What if the data already has some degree of sortedness?

Intermediate-Sortedness in Practice



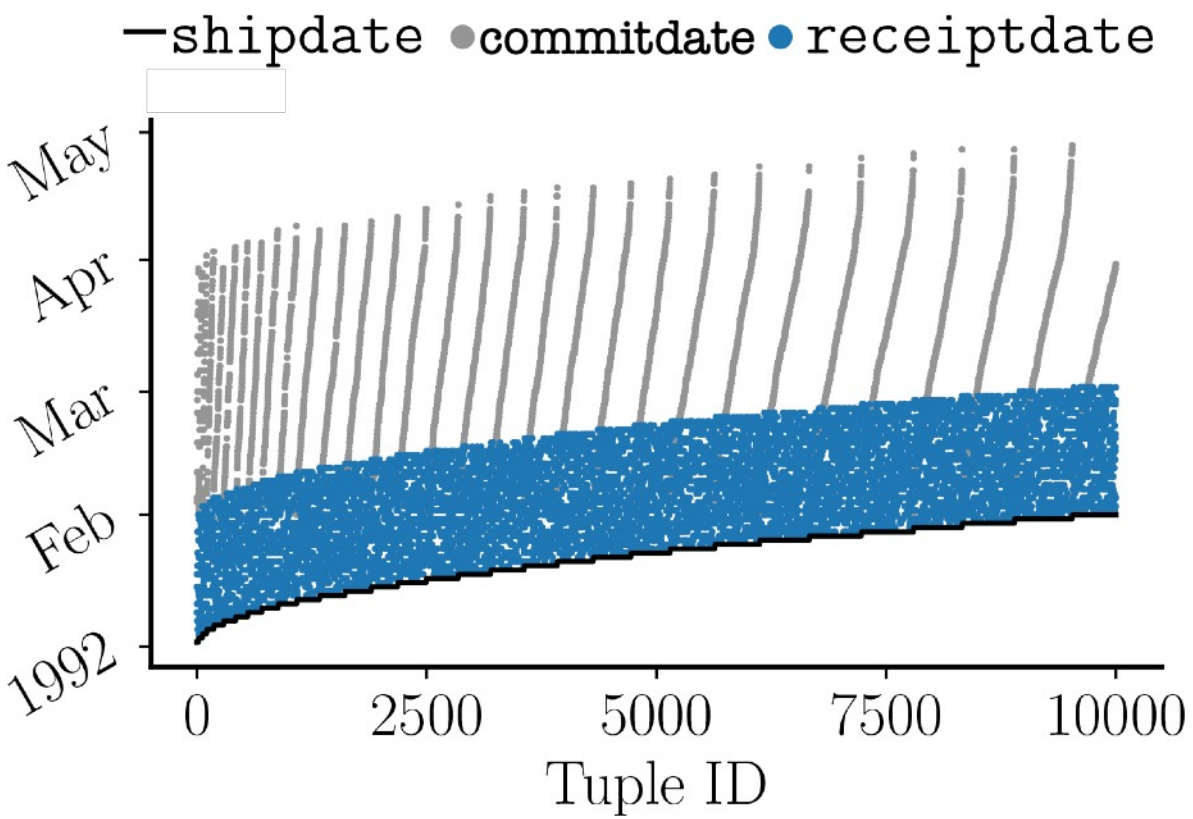
Intermediate-Sortedness in Practice

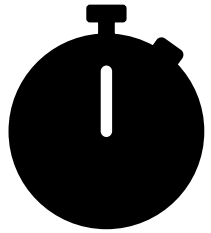




Time Series

Intermediate-Sortedness in Practice



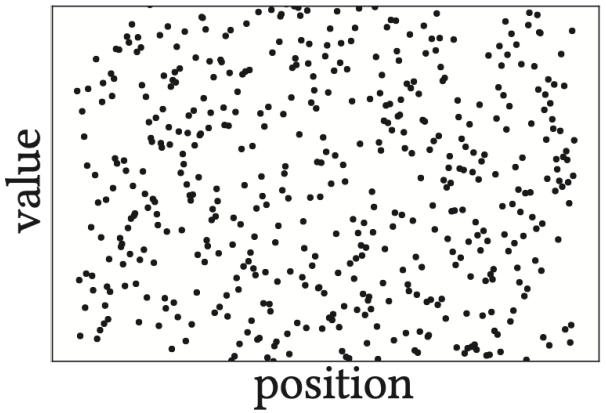


Time Series

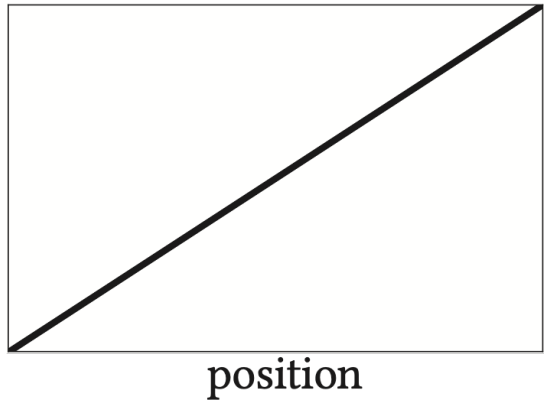
Stock market

Problem - Data Sortedness

Standard ingestion



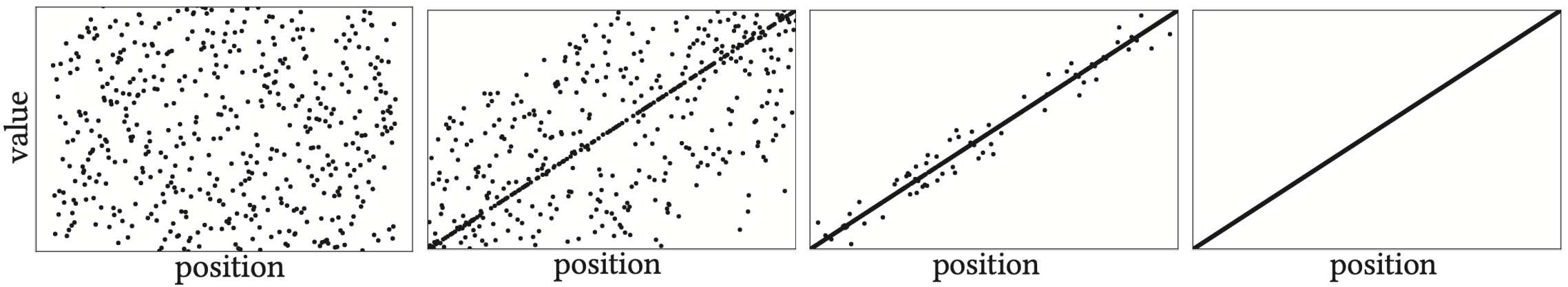
Faster ingestion (e.g., bulk loading)



Problem - Data Sortedness

Standard ingestion

Faster ingestion (e.g., bulk loading)

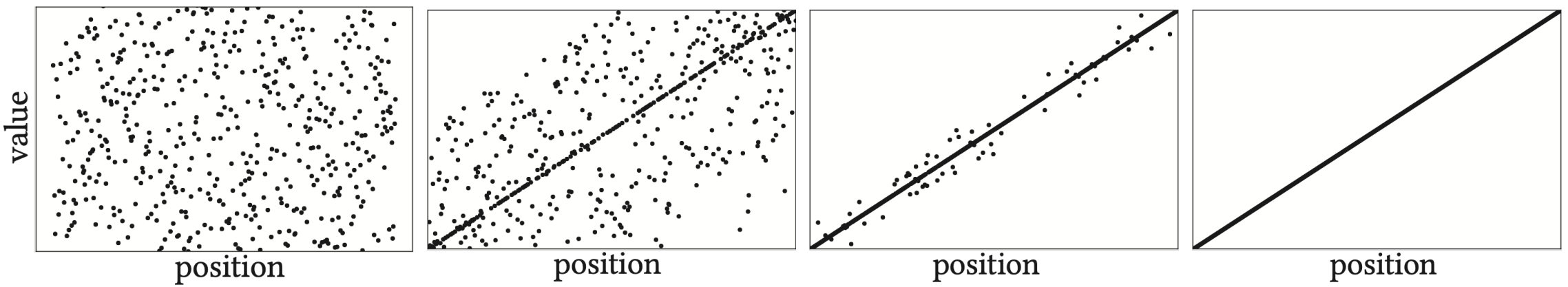


What happens when data is pre-sorted to some degree?

Problem - Data Sortedness

Standard ingestion

Faster ingestion (e.g., bulk loading)



What happens when data is pre-sorted to some degree?

Intermediately-sorted data treated as scrambled data!

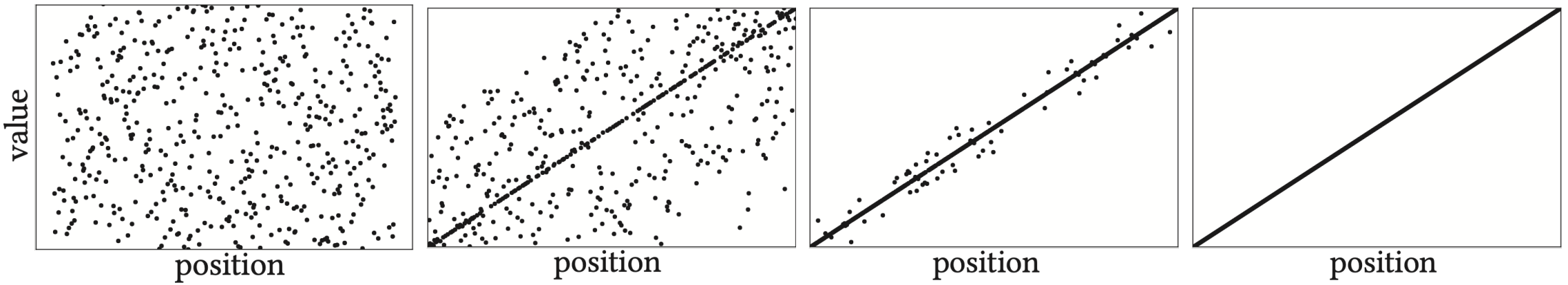
Problem - Data Sortedness

Standard ingestion

Standard ingestion

Standard ingestion

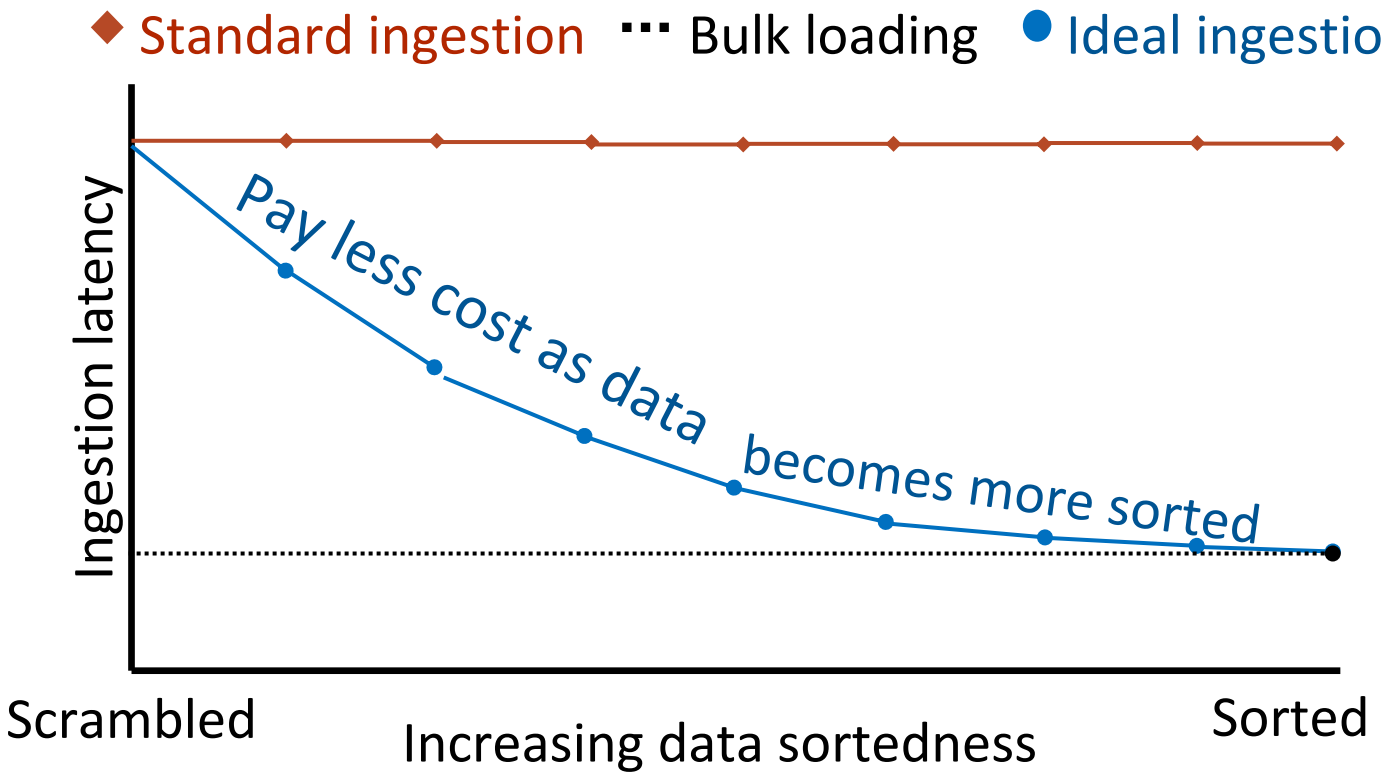
Faster ingestion (e.g., bulk loading)



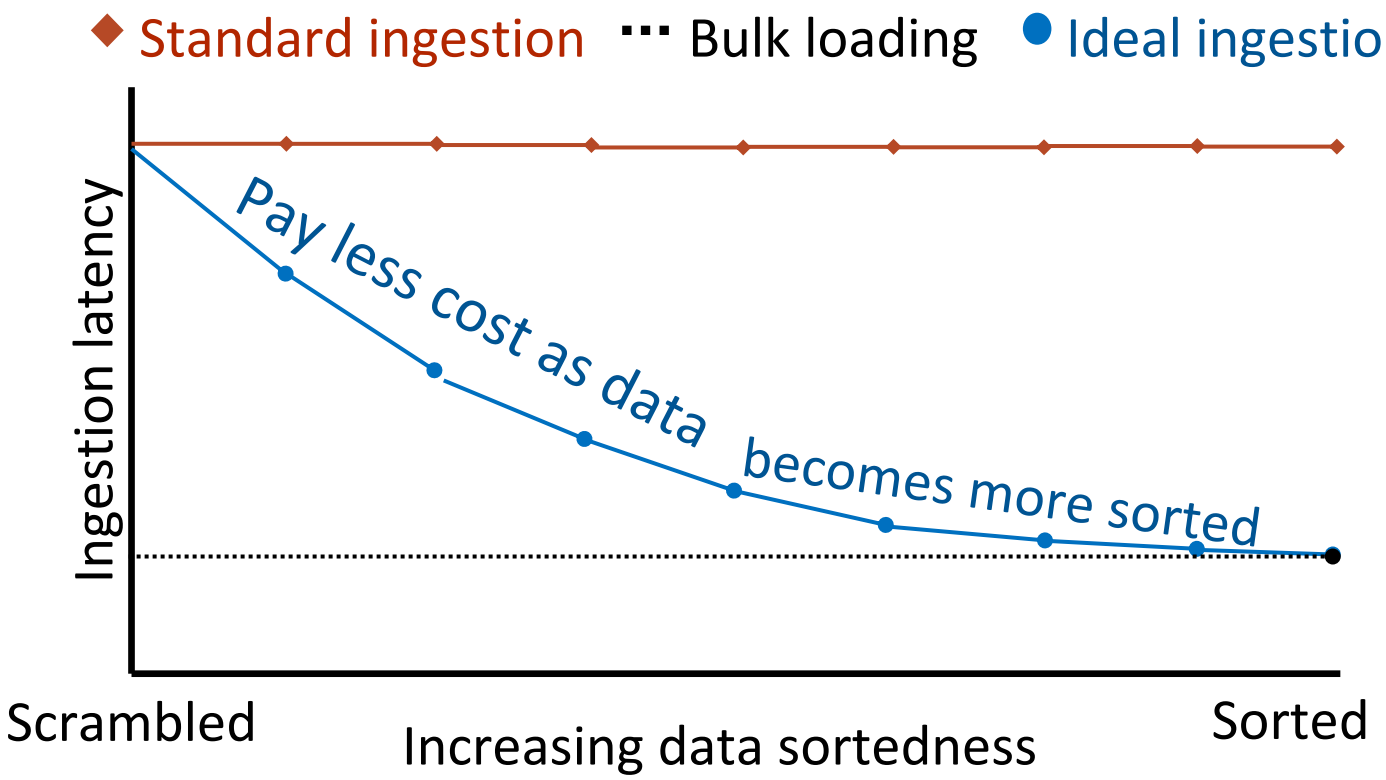
What happens when data is pre-sorted to some degree?

Intermediately-sorted data treated as scrambled data!

Ingestion with Increasing Sortedness

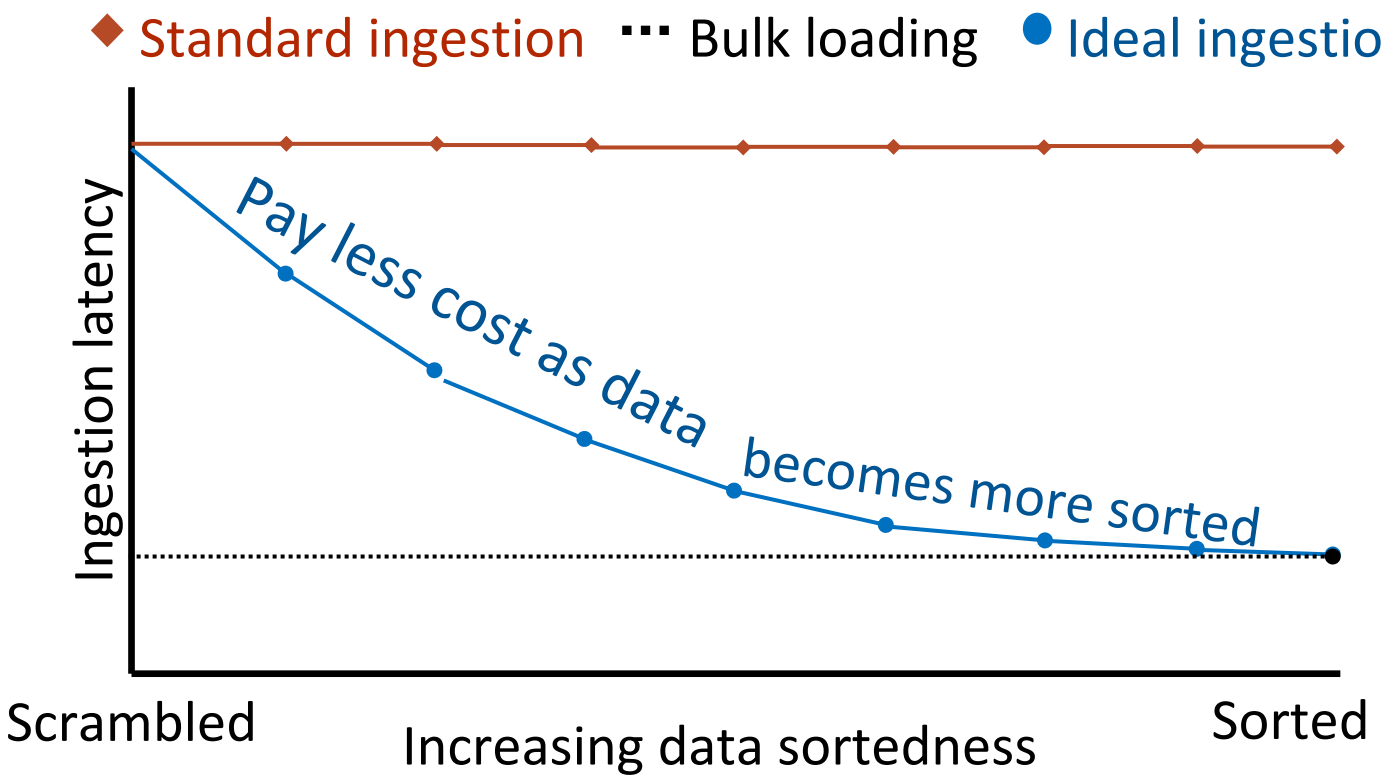


Ingestion with Increasing Sortedness



Is this possible?

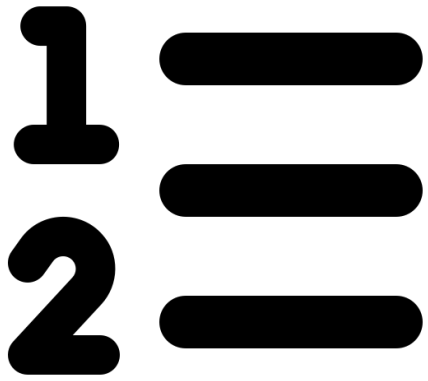
Ingestion with Increasing Sortedness



Is this possible?

1. Unexplored study with sortedness
2. Lack of testing framework

Benchmark on Data Sortedness (BoDS)

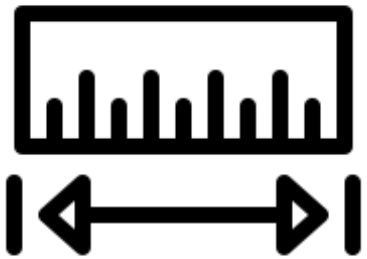


Variable Sortedness
Data Generator

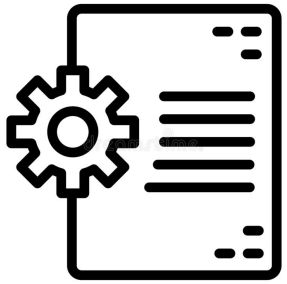
+

Benchmarking
Suite

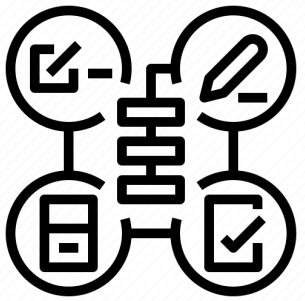
Agenda



Quantifying sortedness



Generating data



Benchmarking framework

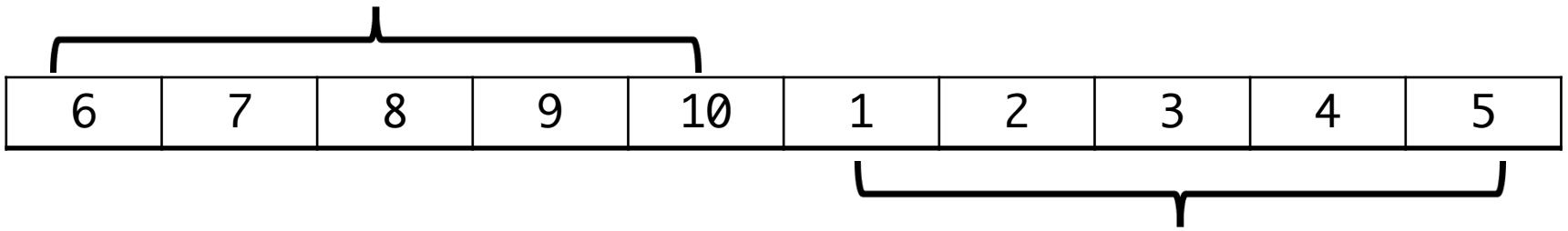
Evaluation

Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order

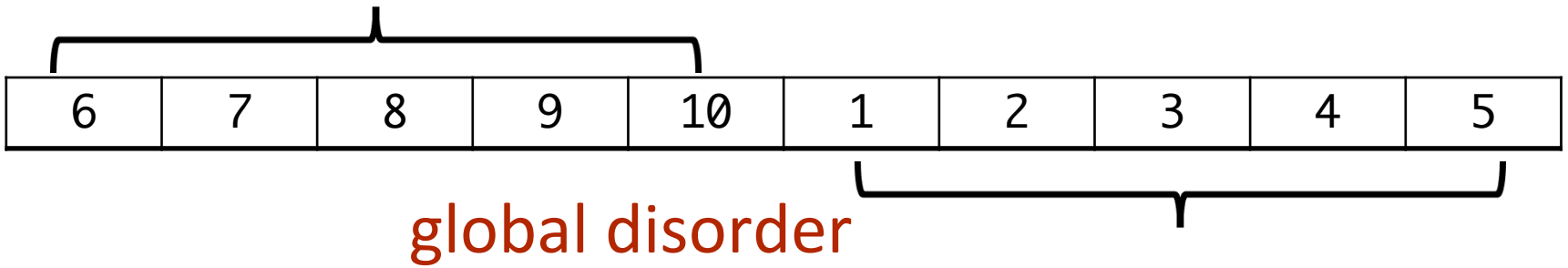
Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order



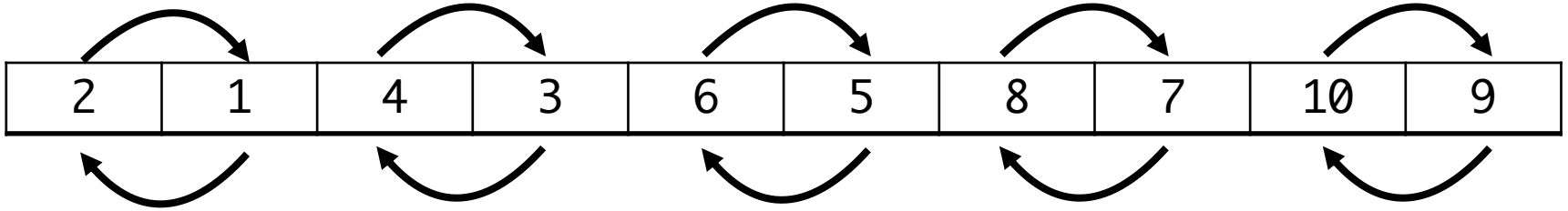
Quantifying Data Sortedness

Metric	Description
Inversions	**# pairs in incorrect order**
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order



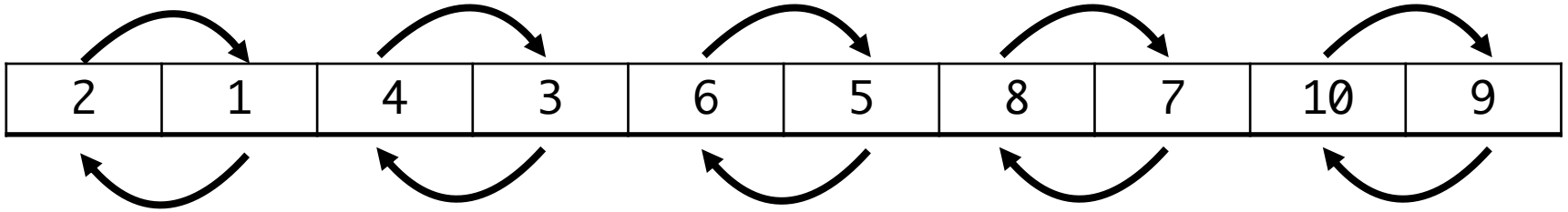
Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	# increasing contiguous subsequences
Exchanges	least # swaps needed to establish total order



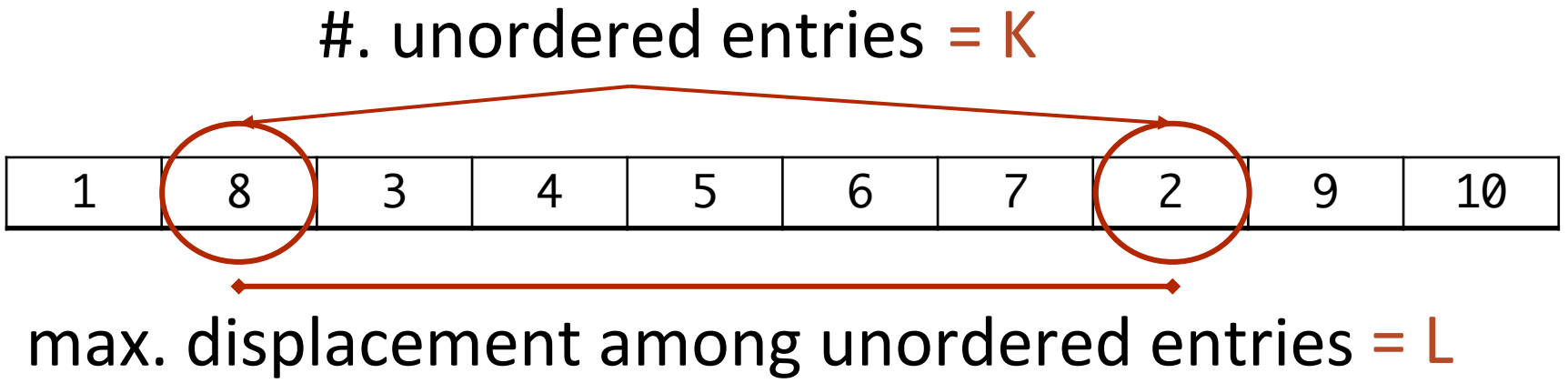
Quantifying Data Sortedness

Metric	Description
Inversions	# pairs in incorrect order
Runs	**# increasing contiguous subsequences**
Exchanges	least # swaps needed to establish total order



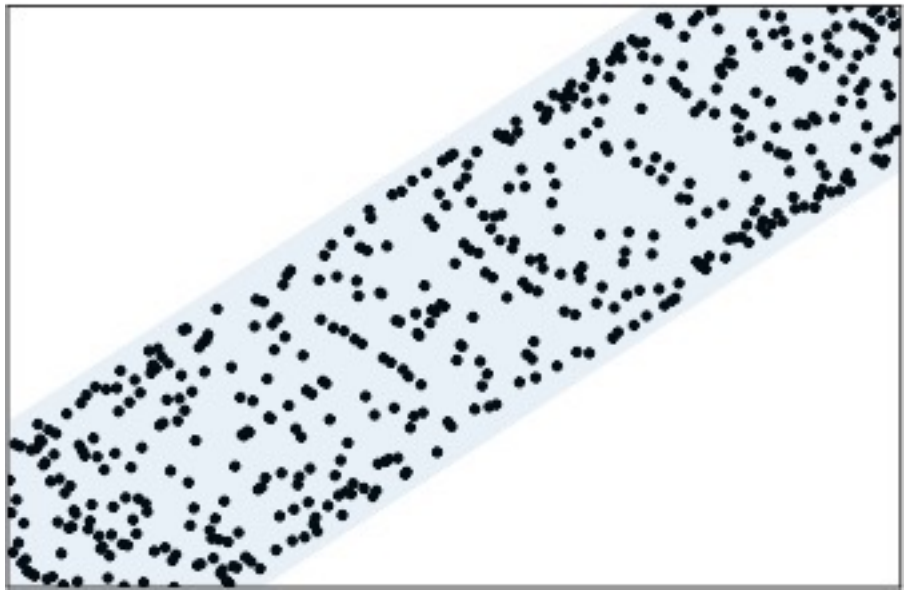
local disorder

(K, L)-Sortedness Metric

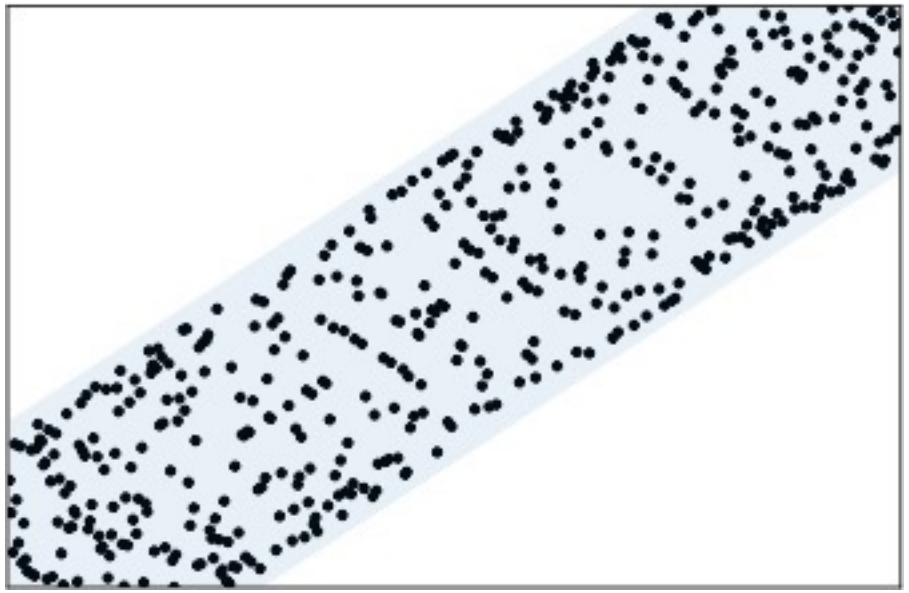


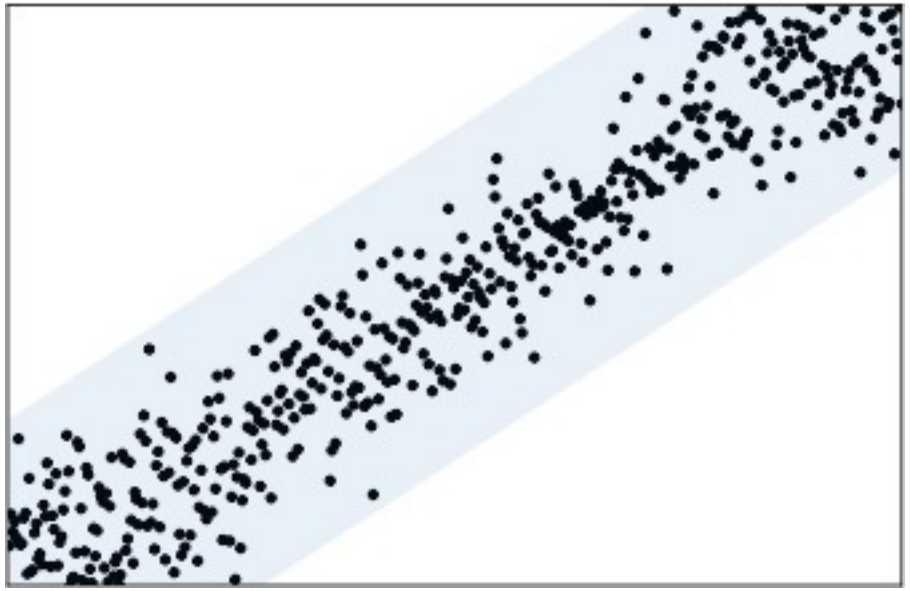
[BenMoshe, ICDT 2011]

Is (K, L) Enough?



Is (K, L) Enough?





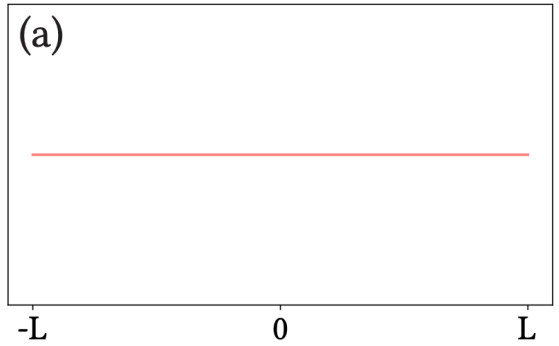
What if we want to distribute the unordered entries differently?

Displacement (L) Distribution

$B(\alpha, \beta)$ **bounded between** $[-L, L]$

Displacement (L) Distribution

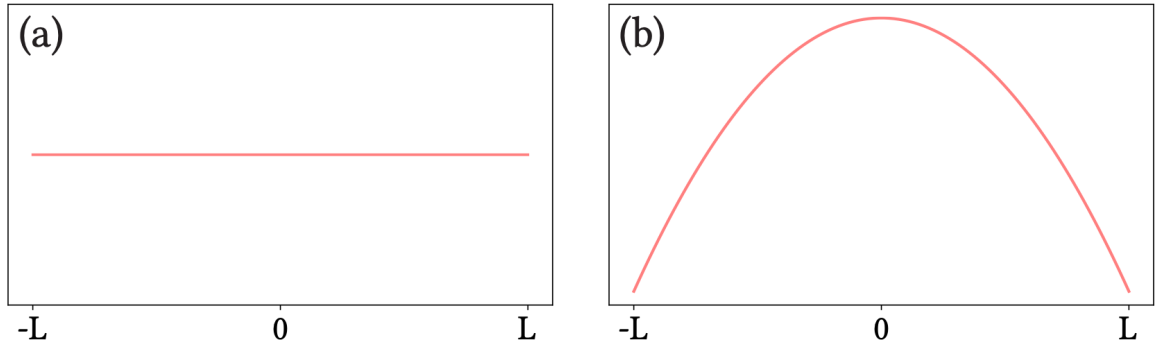
$B(\alpha, \beta)$ **bounded between** $[-L, L]$



$\alpha = \beta = 1$

Displacement (L) Distribution

$B(\alpha, \beta)$ **bounded between** $[-L, L]$

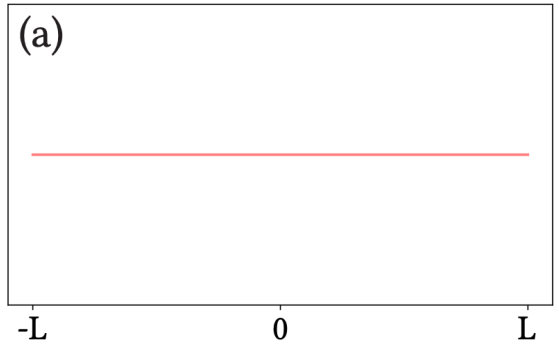


$\alpha = \beta = 1$

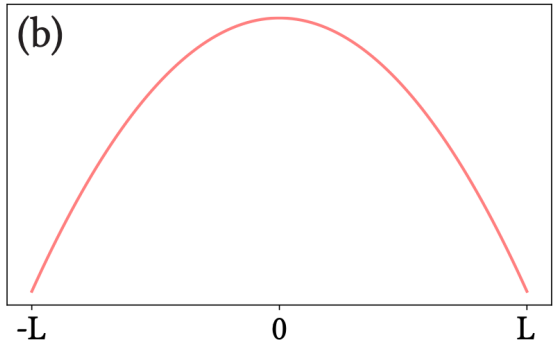
$\alpha = \beta = 2$

Displacement (L) Distribution

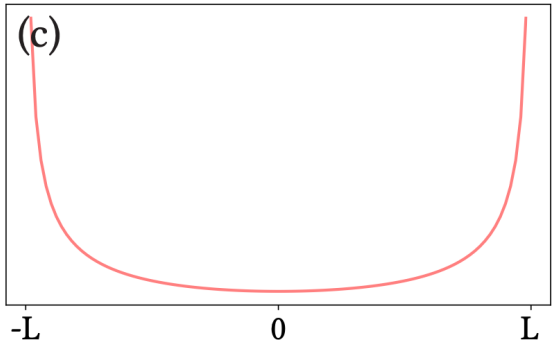
$B(\alpha, \beta)$ **bounded between** $[-L, L]$



$\alpha = \beta = 1$



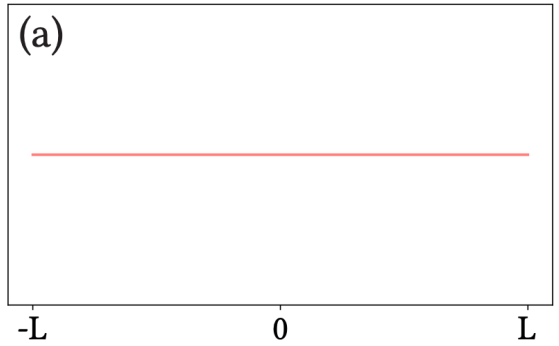
$\alpha = \beta = 2$



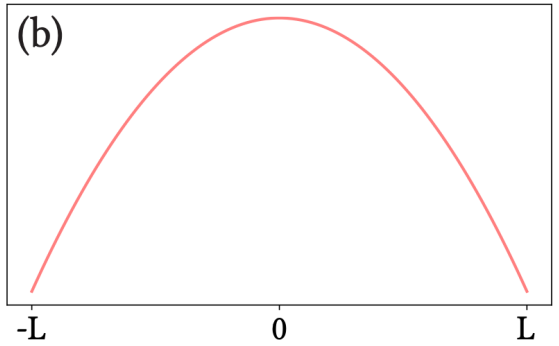
$\alpha = \beta = 0.5$

Displacement (L) Distribution

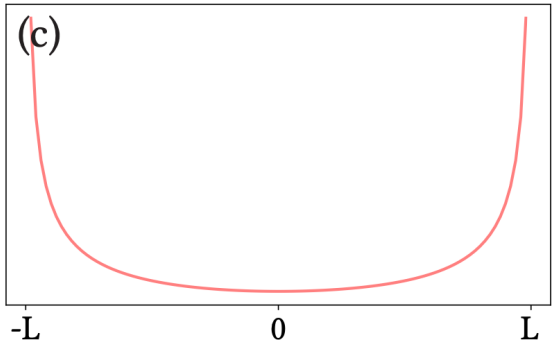
$B(\alpha, \beta)$ **bounded between** $[-L, L]$



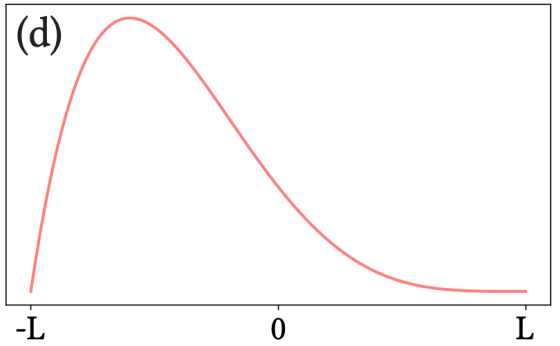
$\alpha = \beta = 1$



$\alpha = \beta = 2$



$\alpha = \beta = 0.5$



$\alpha = 2, \beta = 5$

Generating Differently-Sorted Data

Generating Data

Step 1: generate a fully-sorted data collection

1	2	3	4	5	6	7	8	9	10

Generating Data

Step 1: generate a fully-sorted data collection

1	2	3	4	5	6	7	8	9	10

Step 2: “induce” unsortedness

1	8	3	4	5	6	7	2	9	10

Generating Data

Step 1: generate a fully-sorted data collection

1	2	3	4	5	6	7	8	9	10

Step 2: “induce” unsortedness

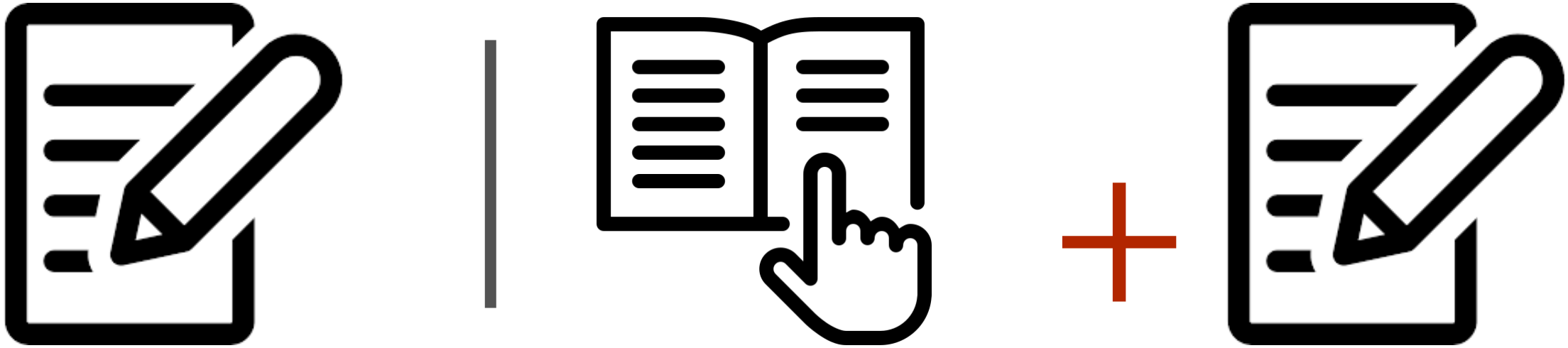
swap entries separated by at most L places

1	8	3	4	5	6	7	2	9	10

The Benchmark

Supported Workloads & Benchmarking Architecture

Supported Workloads



Insert Only

Reads

Writes

Mixed Workloads (interleaved reads and writes)

Supported Workloads

Type	Workload	Data Loading		Operations	
		Method	% of data	R-W ratio	% of data
Insert only	A	Bulk loading	100%	-	-
Mixed reads & writes					

Supported Workloads

Type	Workload	Data Loading		Operations	
		Method	% of data	R-W ratio	% of data
Insert only	A	Bulk loading	100%	-	-
	B	Individual inserts	100%	-	-
Mixed reads & writes					

Supported Workloads

Type	Workload	Data Loading		Operations	
		Method	% of data	R-W ratio	% of data
Insert only	A	Bulk loading	100%	-	-
	B	Individual inserts	100%	-	-
Mixed reads & writes	C	-	0%	17%-83%	100%

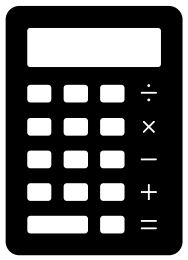
Supported Workloads

Type	Workload	Data Loading		Operations	
		Method	% of data	R-W ratio	% of data
Insert only	A	Bulk loading	100%	-	-
	B	Individual inserts	100%	-	-
Mixed reads & writes	C	-	0%	17%-83%	100%
	D	Bulk loading	80%	50%-50%	20%

Supported Workloads

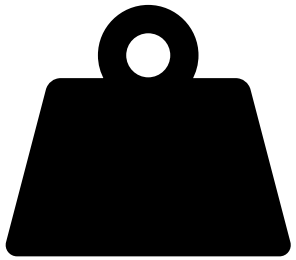
Type	Workload	Data Loading		Operations	
		Method	% of data	R-W ratio	% of data
Insert only	A	Bulk loading	100%	-	-
	B	Individual inserts	100%	-	-
Mixed reads & writes	C	-	0%	17%-83%	100%
	D	Bulk loading	80%	50%-50%	20%
	E	Individual inserts	80%	50%-50%	20%

Inputs to the Benchmark



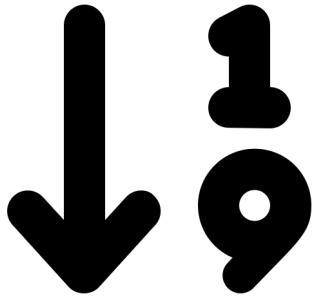
#.entries

N



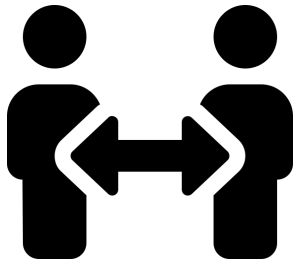
Payload Size

P



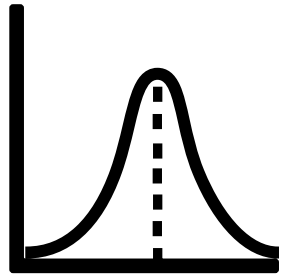
#.unordered entries

K



Maximum Displacement

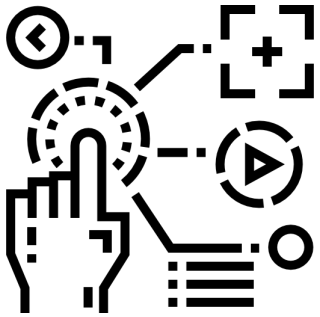
L



Displacement Distribution

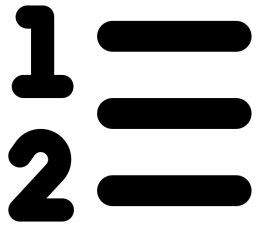
$B(\alpha, \beta)$

The Benchmark Architecture

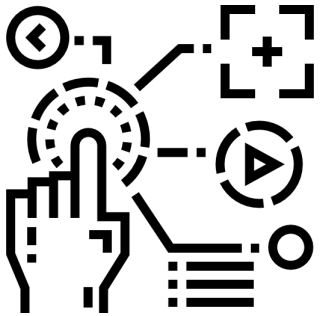


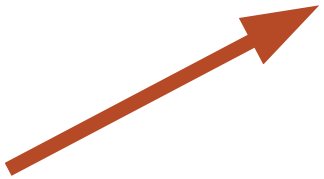
Coordinator

The Benchmark Architecture



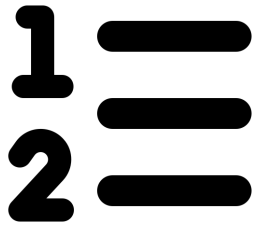
Data Generator



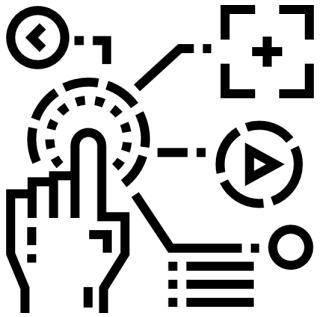


Coordinator

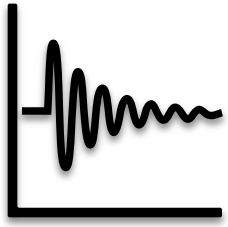
The Benchmark Architecture



Data Generator

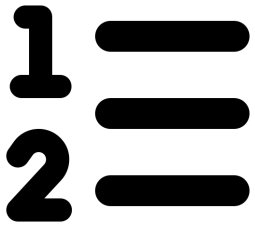


Coordinator

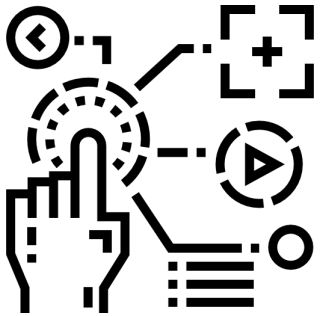


Workload Generator

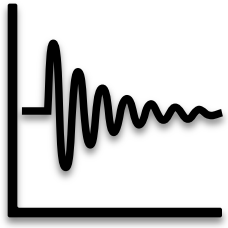
The Benchmark Architecture



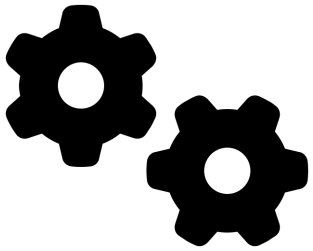
Data Generator



Coordinator

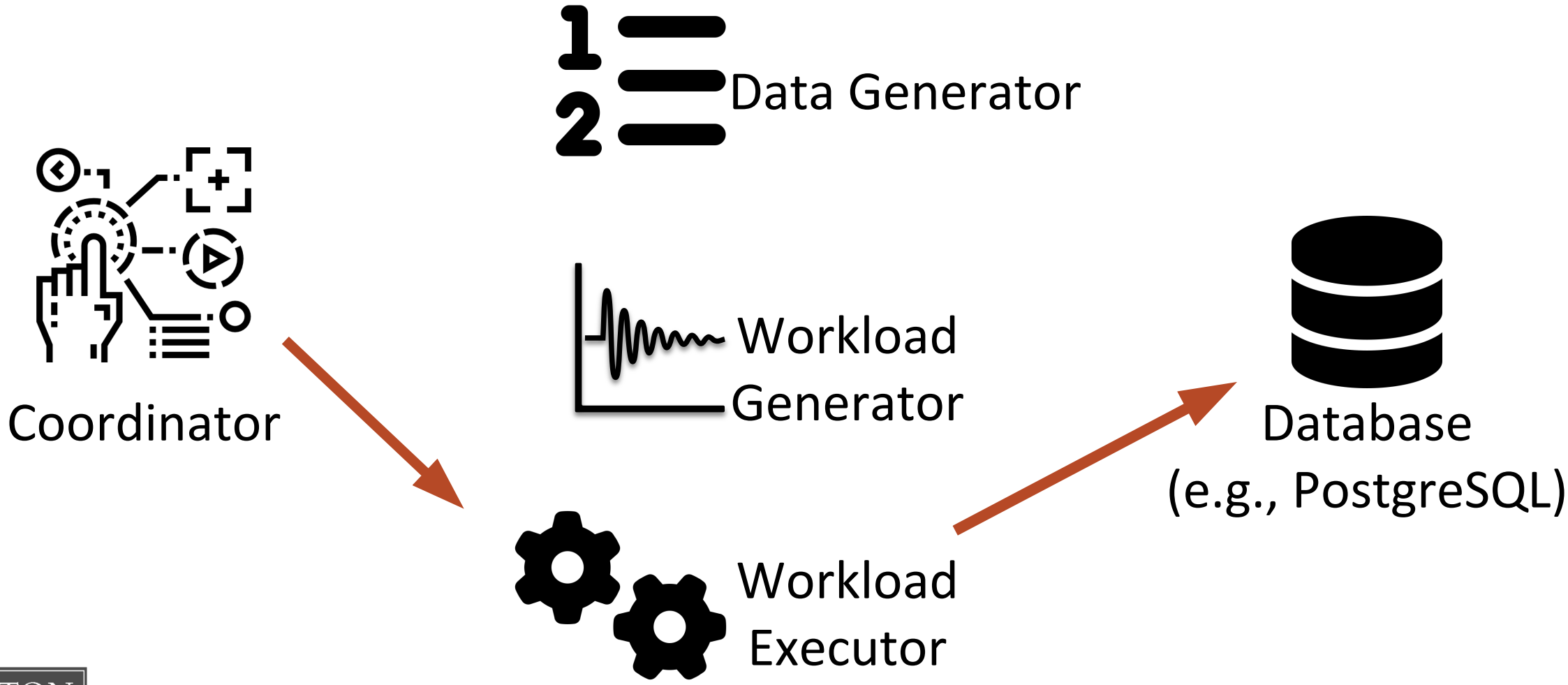


Workload Generator



Workload Executor

The Benchmark Architecture



Benchmarking Action

Metrics:

- Ingestion latency
- Overall operational latency

Benchmarking Action

Metrics:

- Ingestion latency
- Overall operational latency

Data Setup:

- 16M K-V pairs (~ 4GB)
- Key = 4B, Payload = 252B

Benchmarking Action

Metrics:

- Ingestion latency
- Overall operational latency

System Setup:

- AWS EC2 instance (t2.medium)
- 2 Intel Xeon CPU v4 @2.3GHz
- 4GB RAM, 40GB SSD

Data Setup:

- 16M K-V pairs (~ 4GB)
- Key = 4B, Payload = 252B

Benchmarking Action

Metrics:

- Ingestion latency
- Overall operational latency

System Setup:

- AWS EC2 instance (t2.medium)
- 2 Intel Xeon CPU v4 @2.3GHz
- 4GB RAM, 40GB SSD

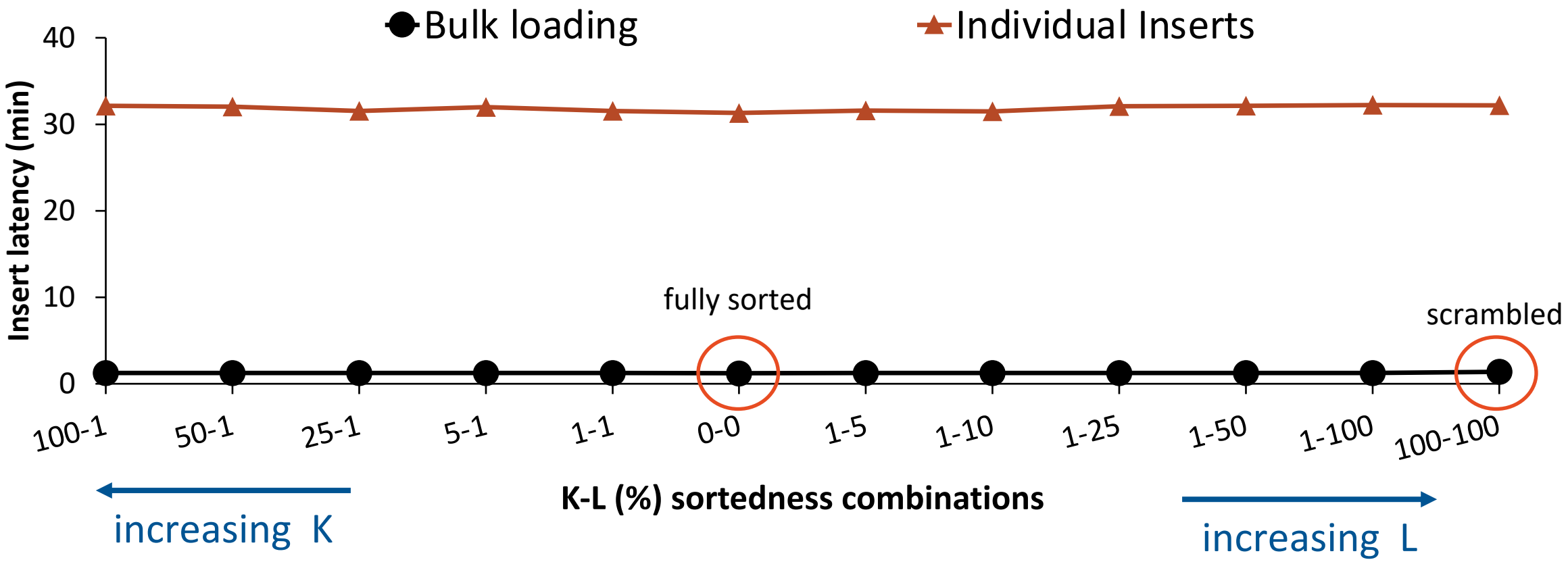
Data Setup:

- 16M K-V pairs (~ 4GB)
- Key = 4B, Payload = 252B

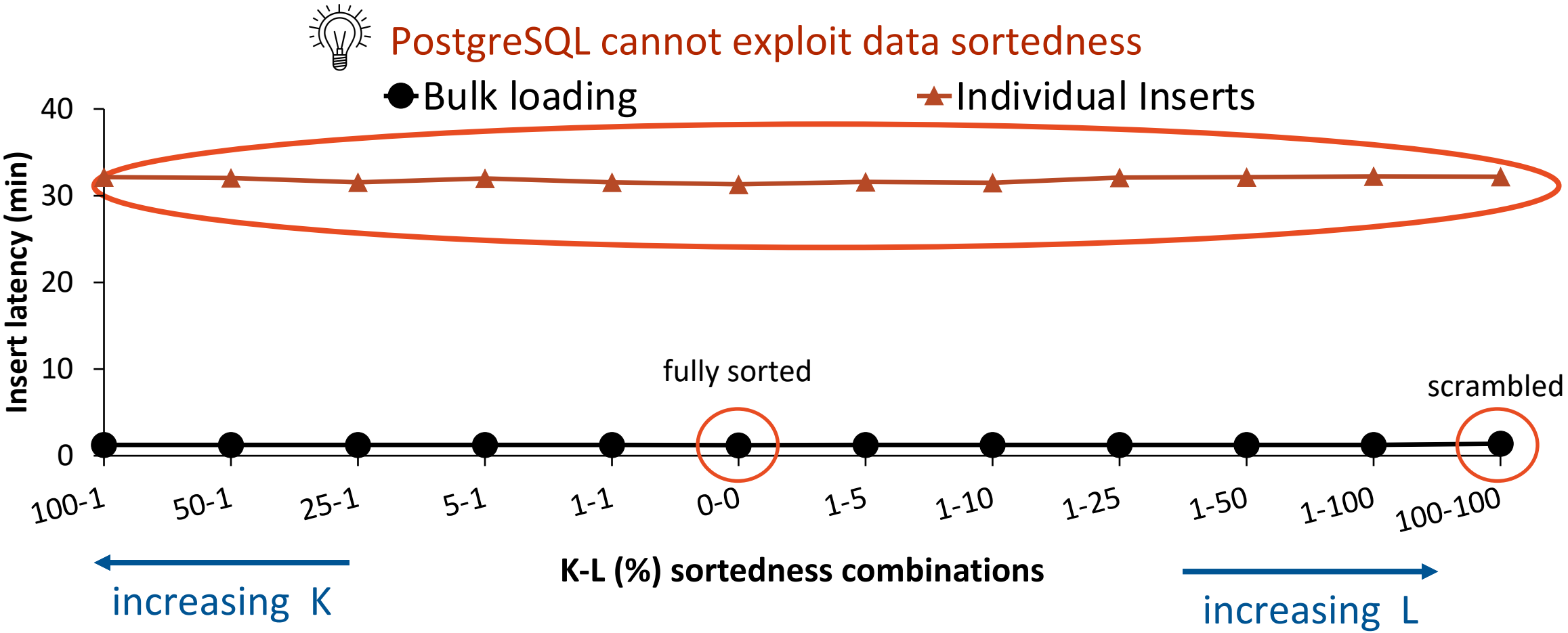
Default Index Setup:

- PostgreSQL (Unlogged tables)
- B-tree on key (id_col)

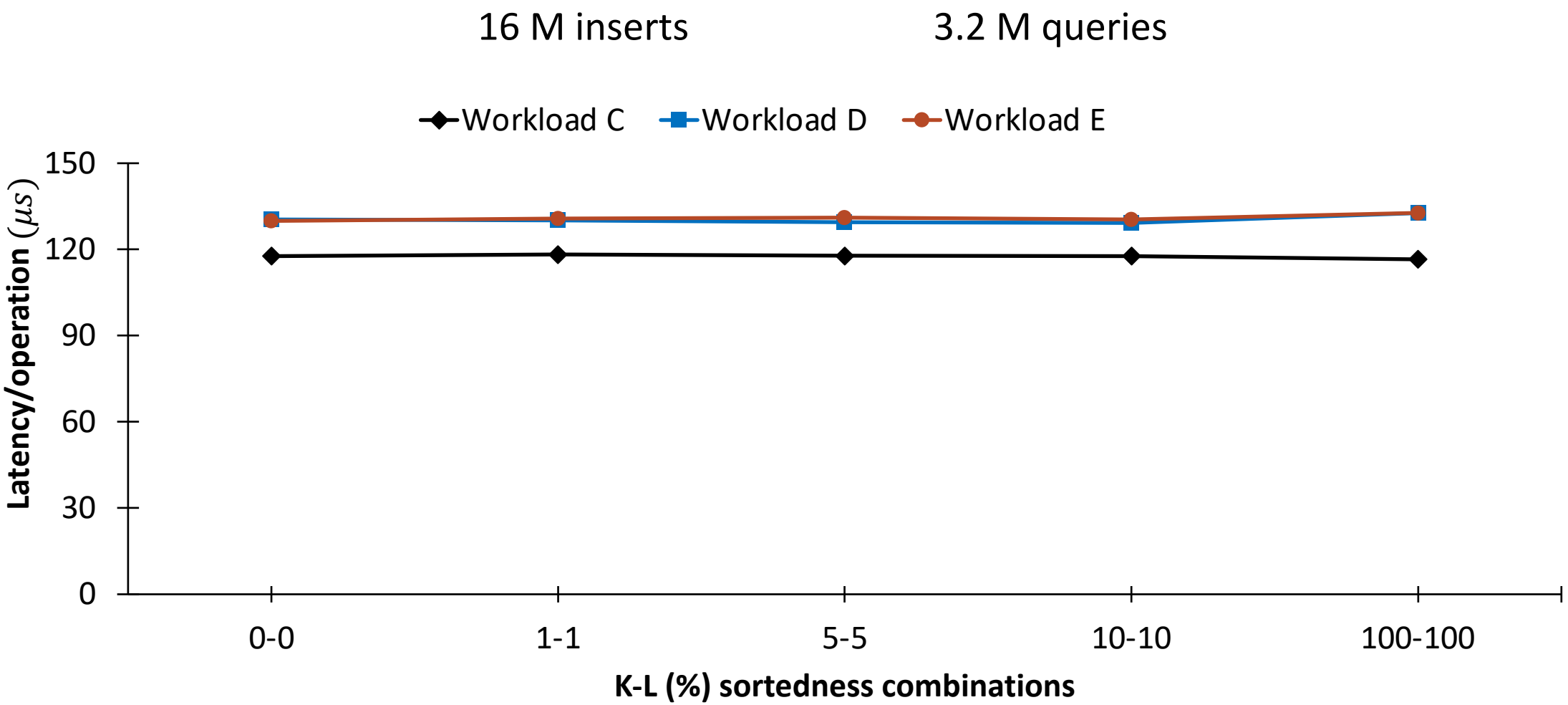
Raw Ingestion Performance



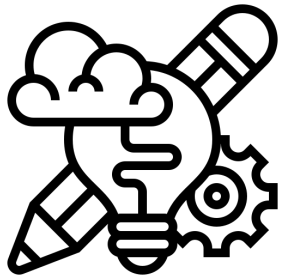
Raw Ingestion Performance



Mixed Workload Performance

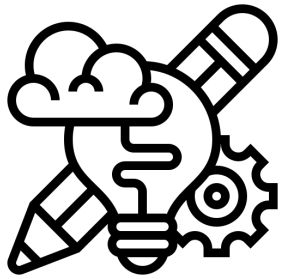


Column-Store Systems



Fundamental design changes

Column-Store Systems

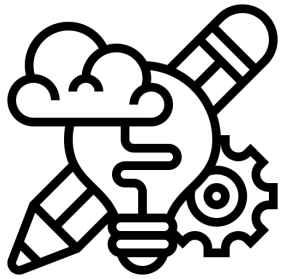


Fundamental design changes

1. MonetDB

   - ✕ invalidated by updates

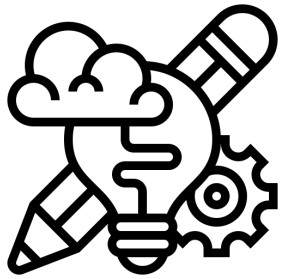
Column-Store Systems



Fundamental design changes

1. MonetDB
   - ✗ invalidated by updates
2. Vertica
   - ✗ no live updates to sorted column

Column-Store Systems

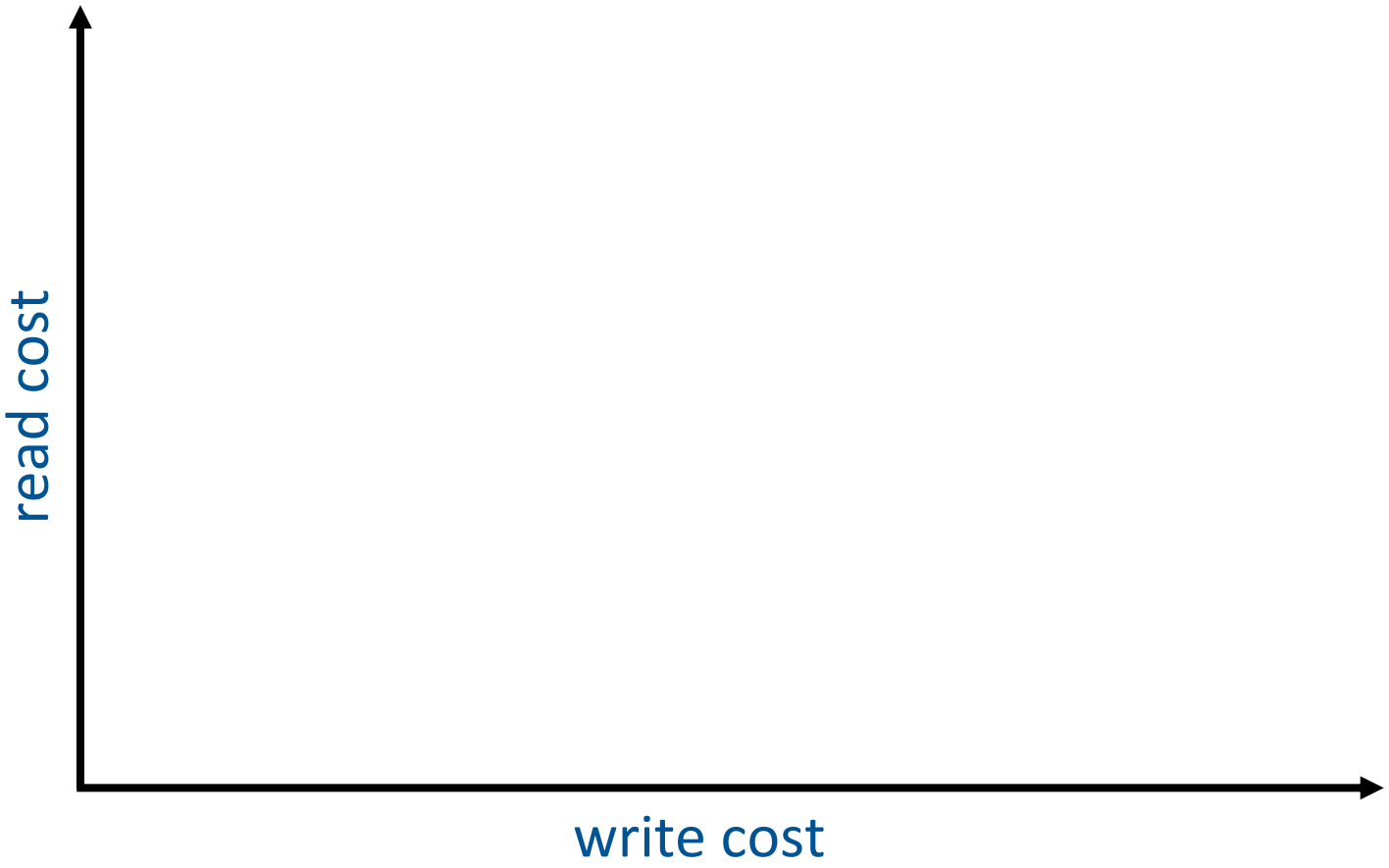


Fundamental design changes

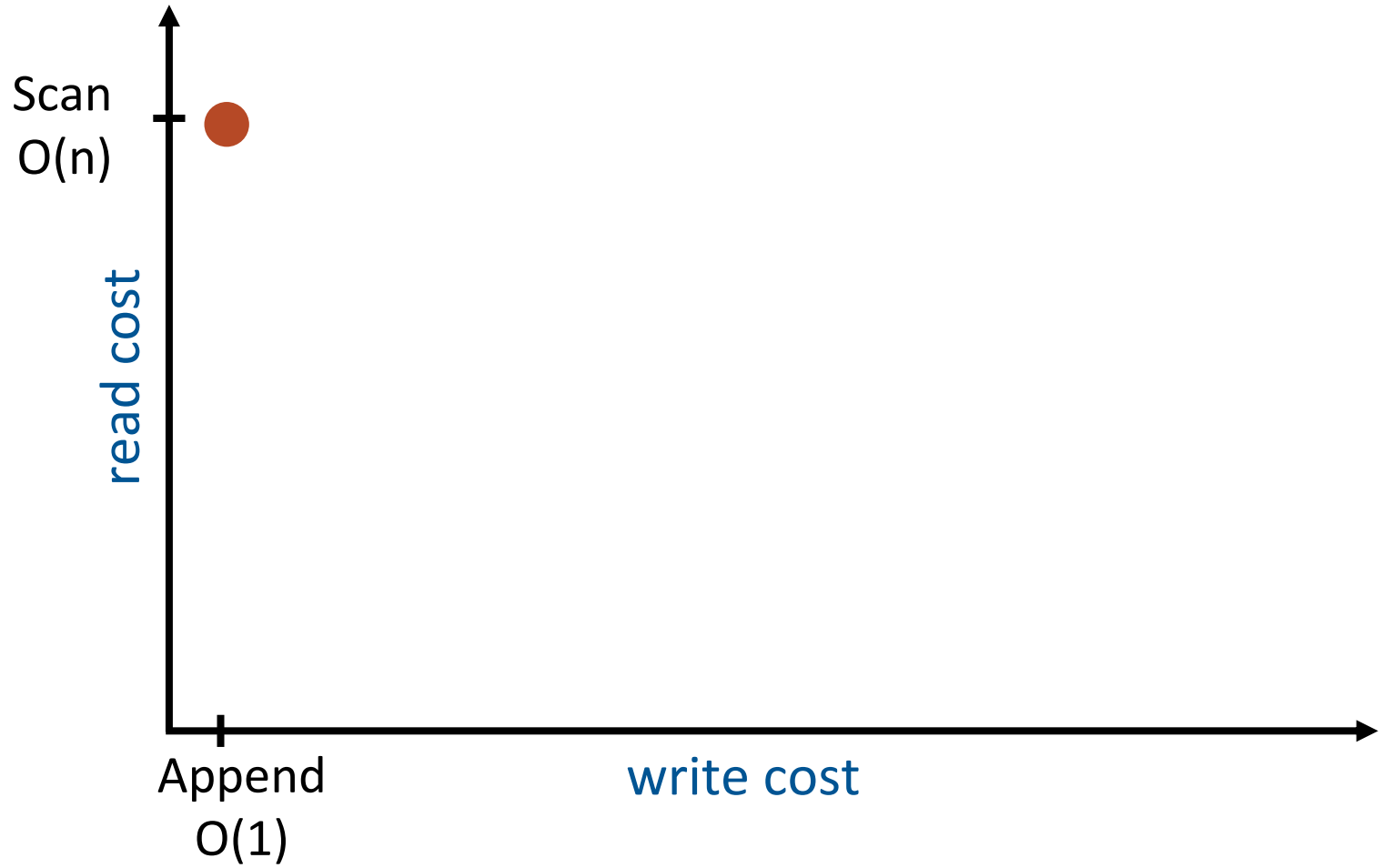
1. MonetDB
   - ✗ invalidated by updates
2. Vertica
   - ✗ no live updates to sorted column
3. Actian Vector
   - ⚠ PDT similar to b-trees

Towards Sortedness-Aware Indexes

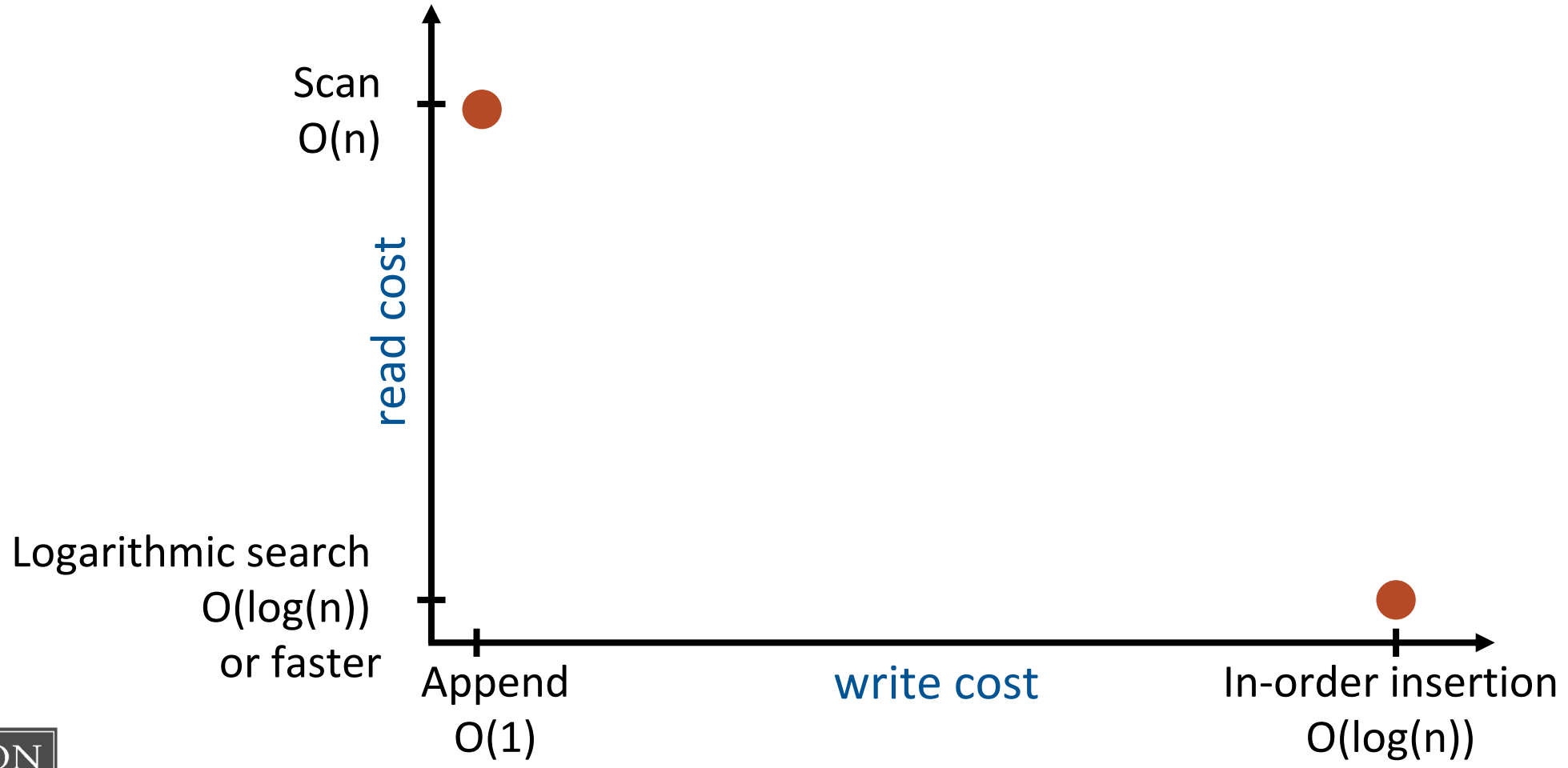
Read-Write Tradeoff



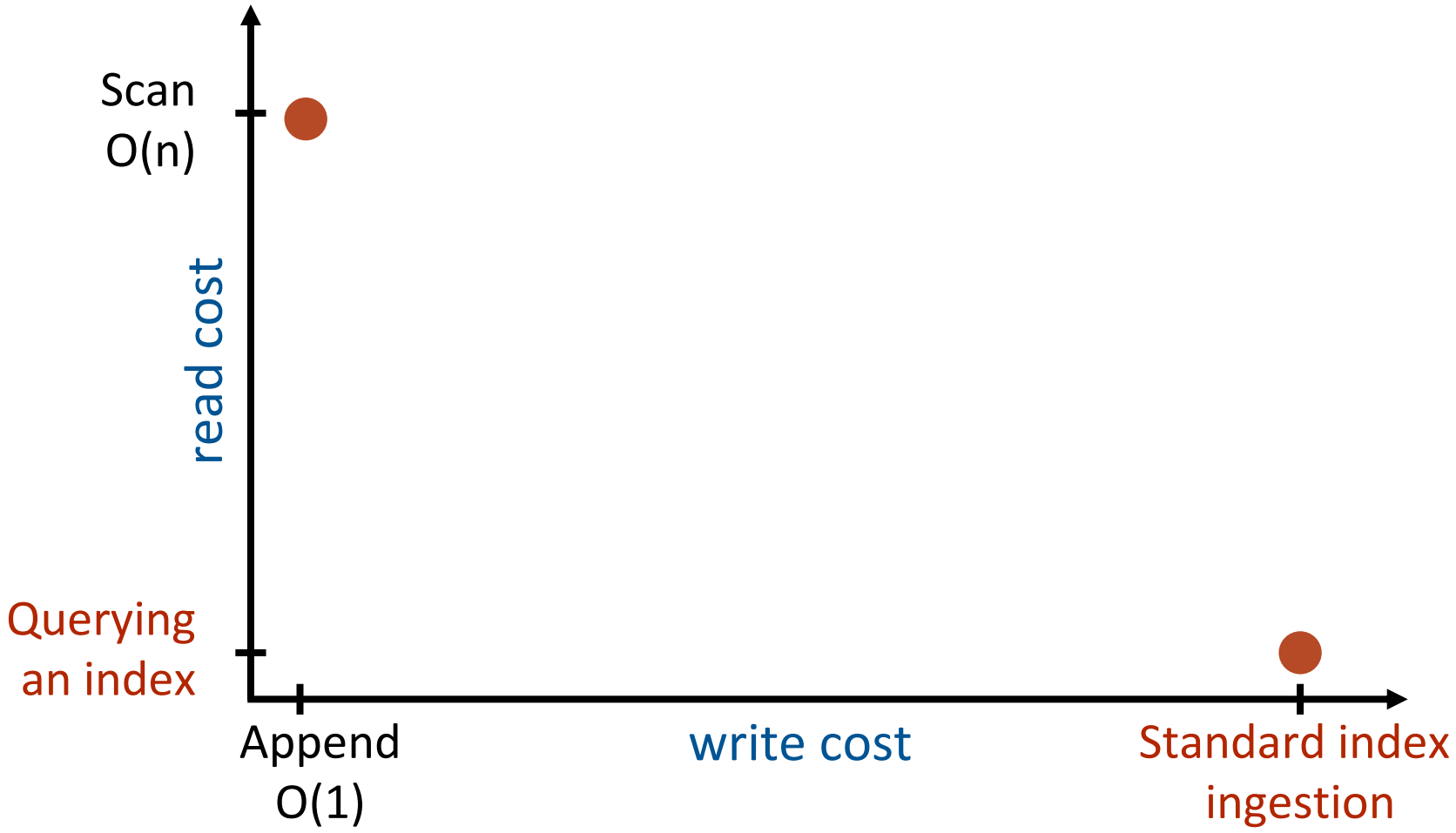
Read-Write Tradeoff



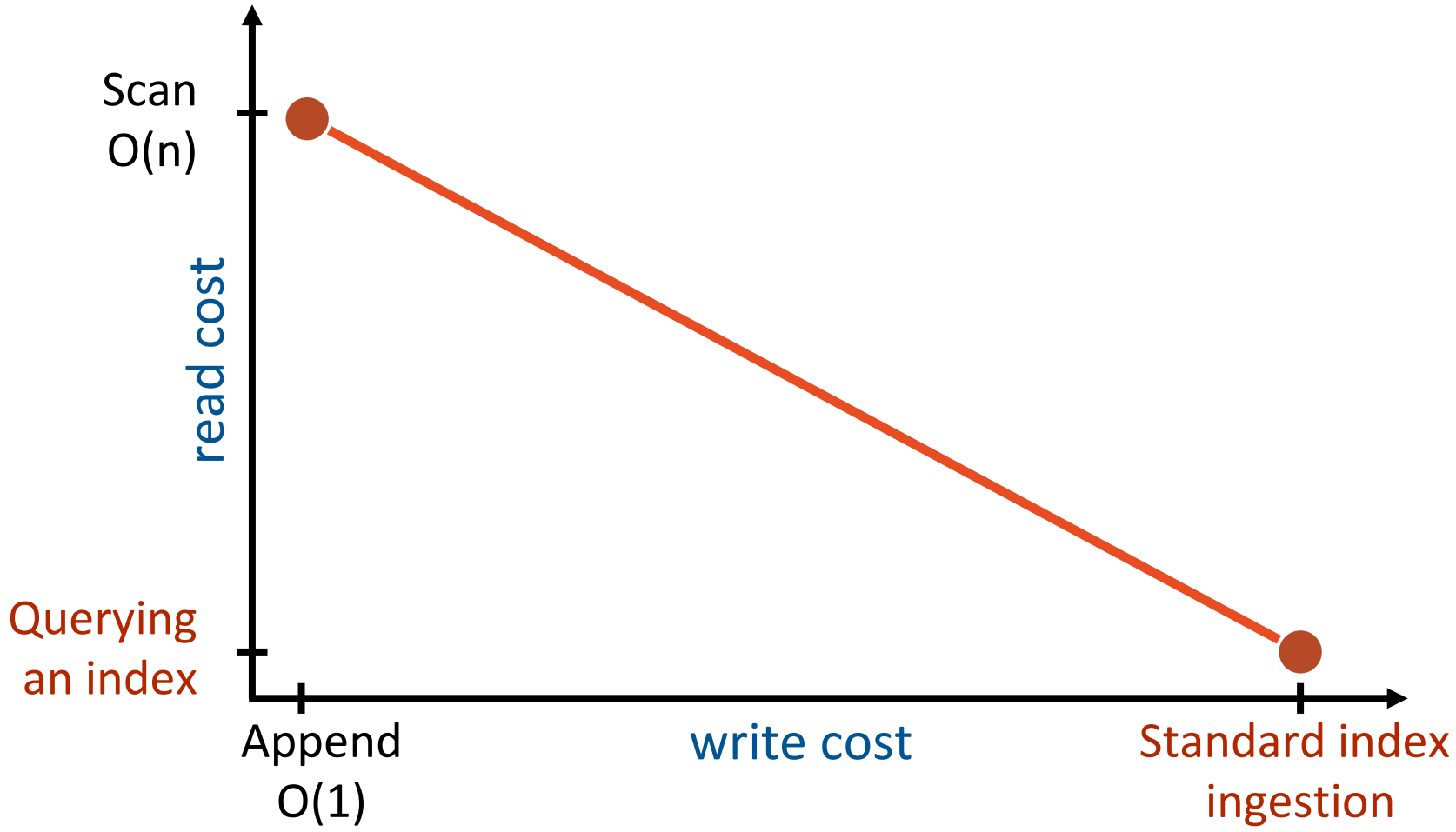
Read-Write Tradeoff



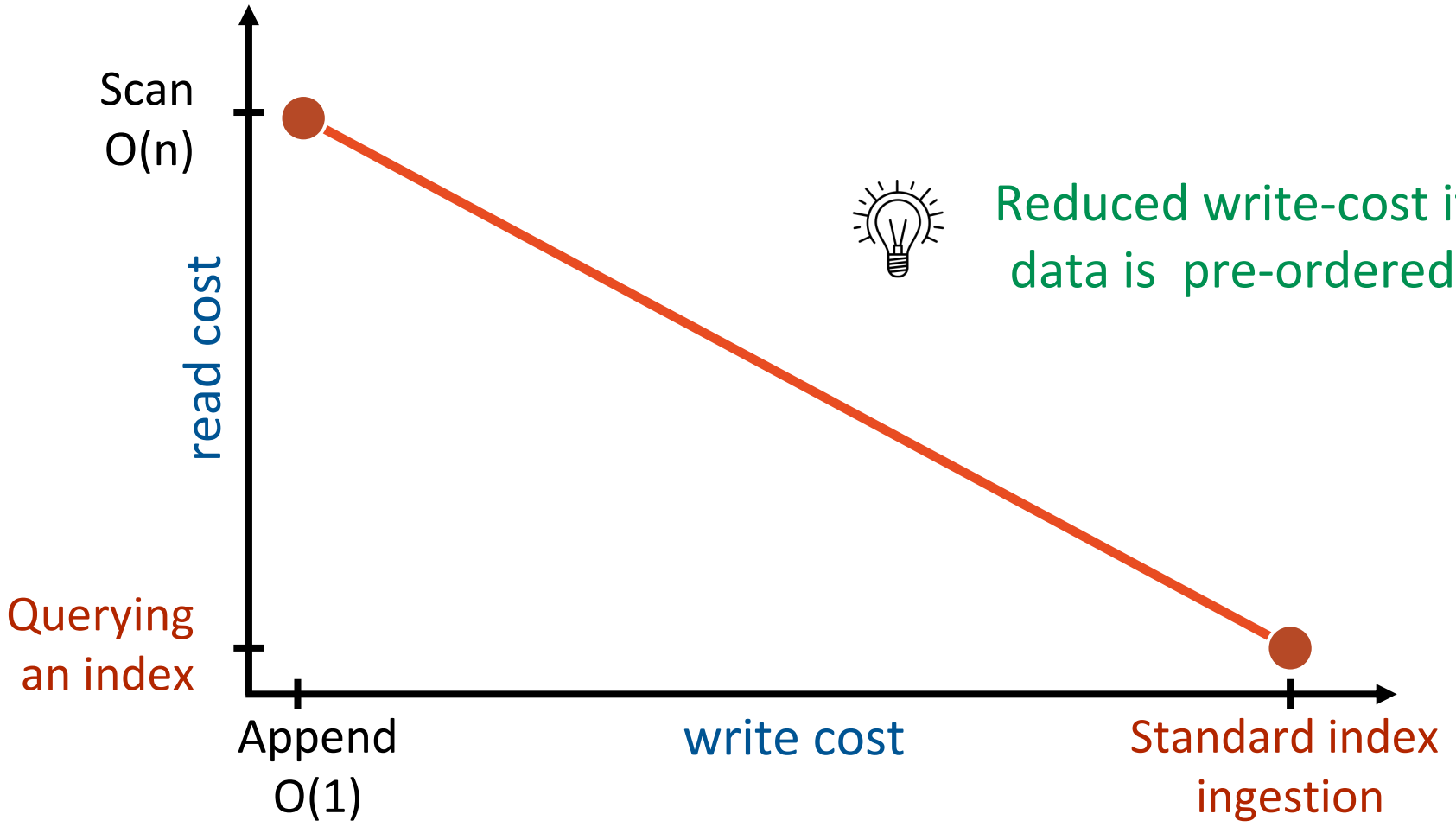
Read-Write Tradeoff



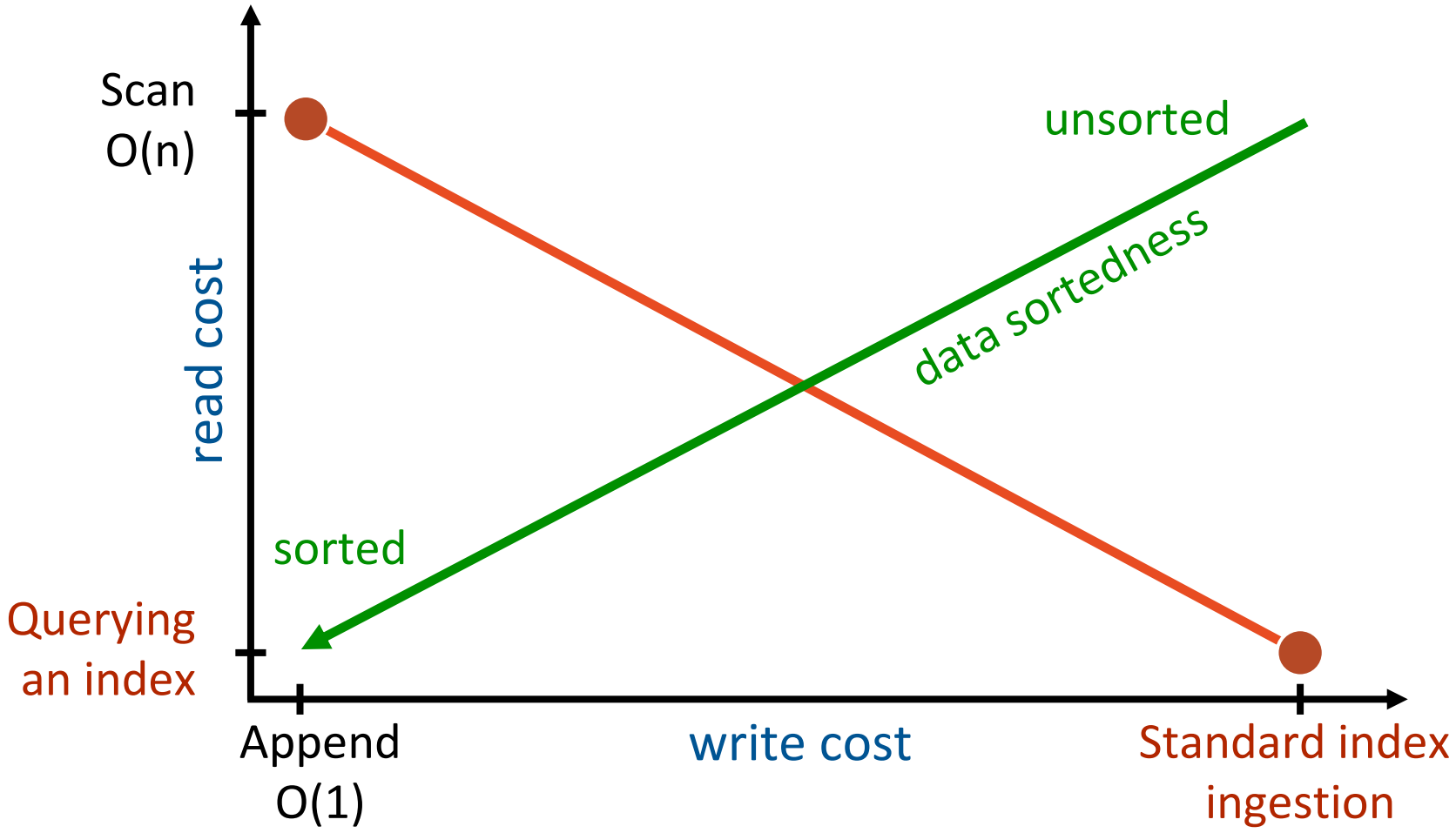
Read-Write Tradeoff



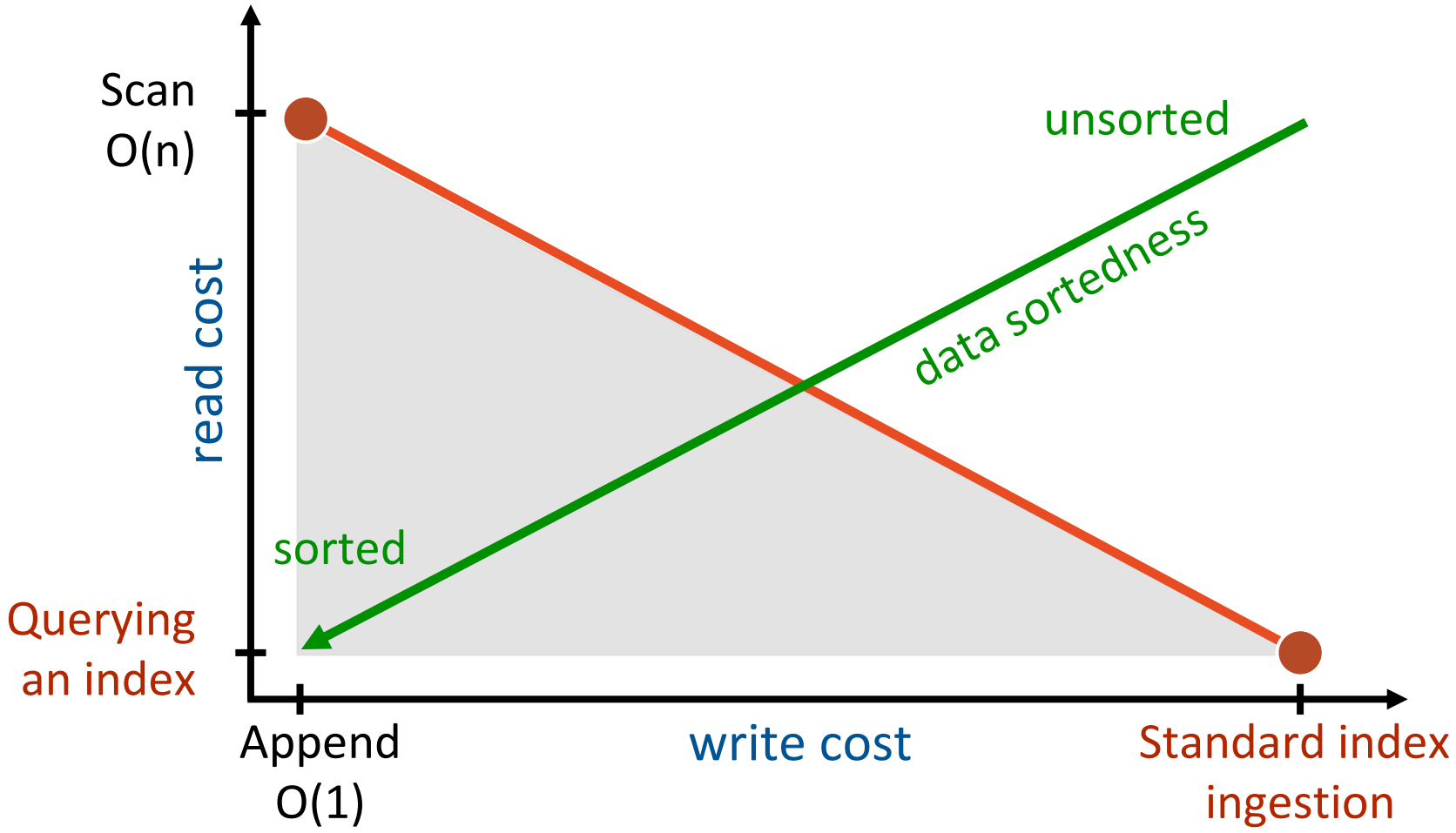
Read-Write Tradeoff



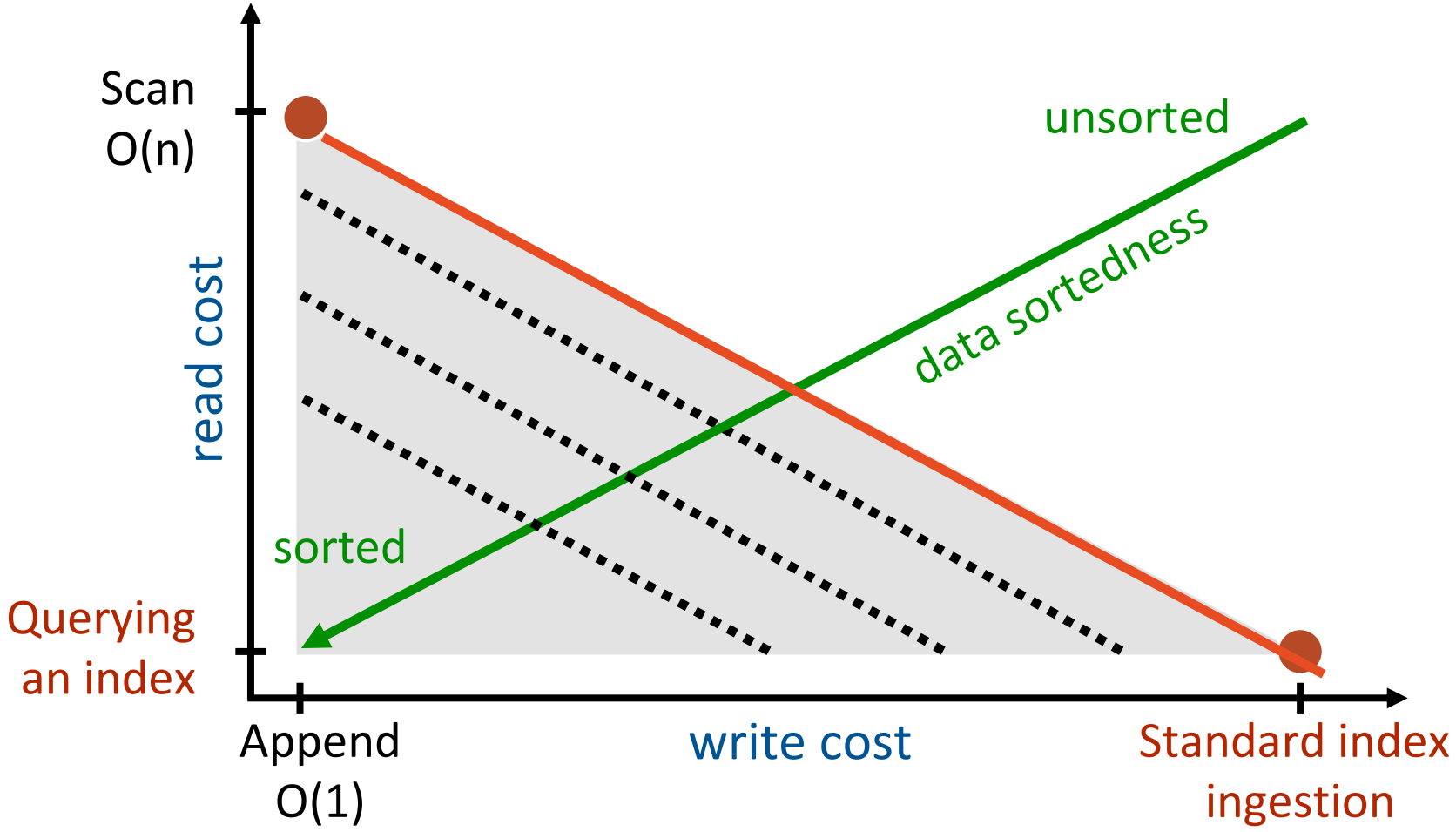
Vision: Sortedness-Aware Indexes



Vision: Sortedness-Aware Indexes



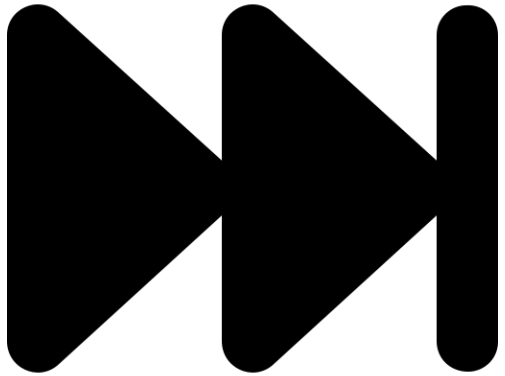
Vision: Sortedness-Aware Indexes



Augmenting the B-tree

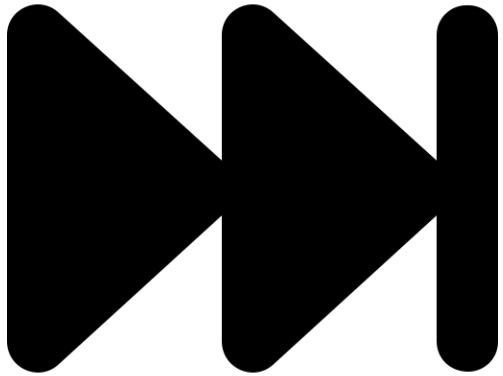
Design principles & brief results

Sortedness-Aware Design Elements

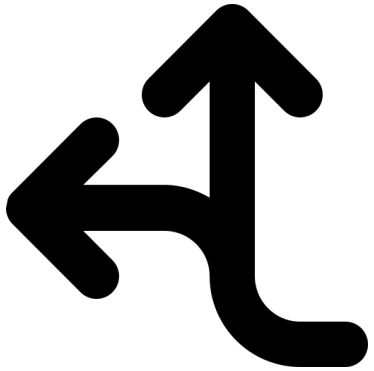


right-most
leaf appends

Sortedness-Aware Design Elements

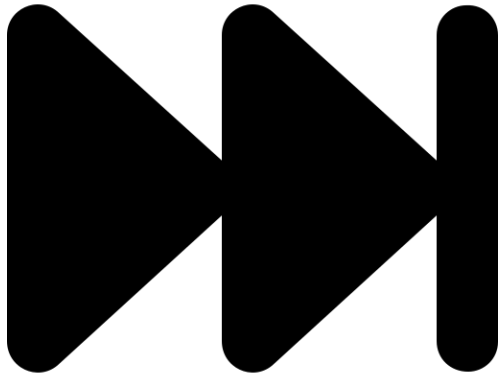


right-most
leaf appends

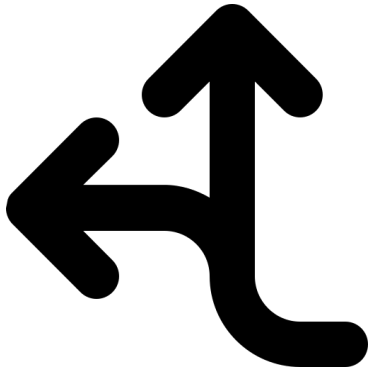


variable
split factor

Sortedness-Aware Design Elements



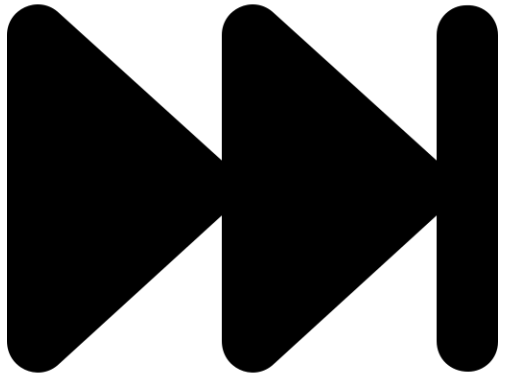
right-most
leaf appends



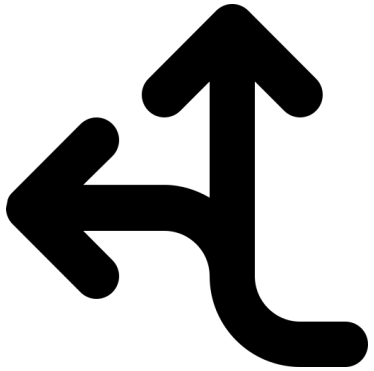
variable
split factor

work well with
fully sorted data

Sortedness-Aware Design Elements



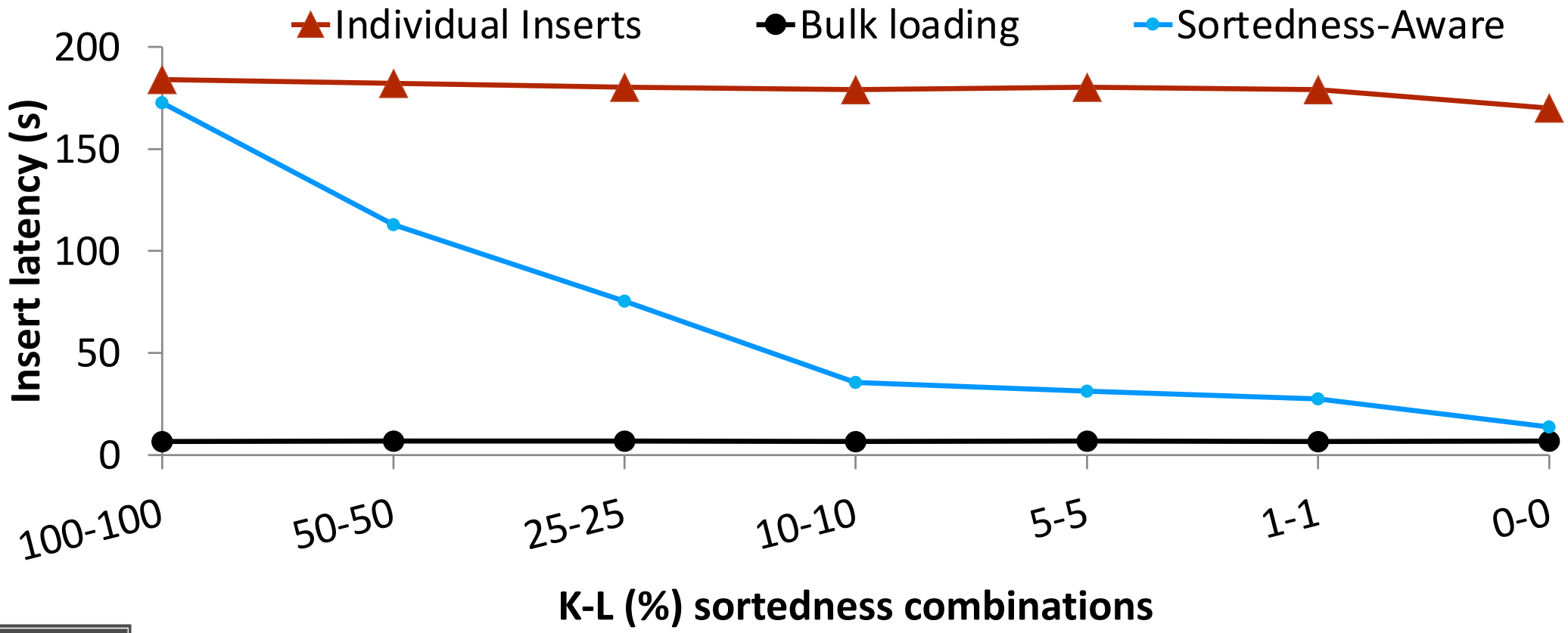
right-most
leaf appends



variable
split factor

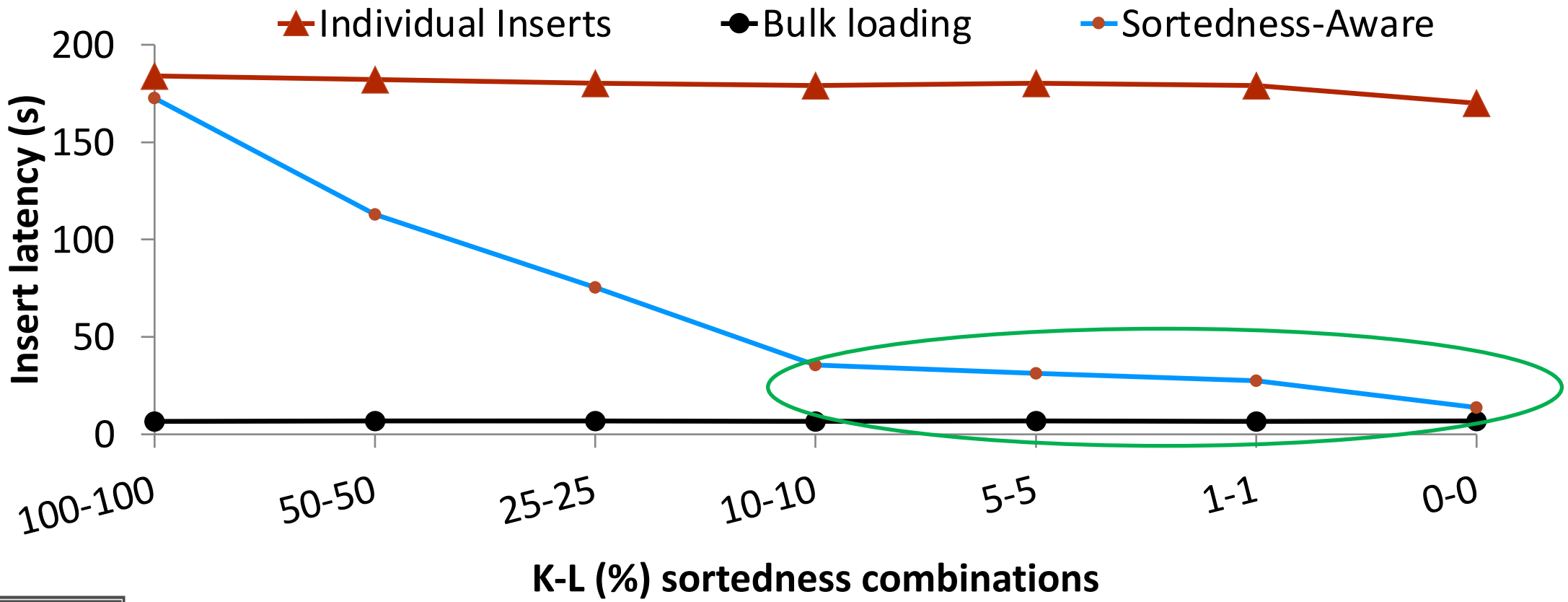
intelligent
buffering

Sortedness-Aware Performance



Sortedness-Aware Performance

Close-to-bulkloading performance with increasing data sortedness



Summary

BoDS lays groundwork for sortedness-aware testing

Classical indexes do not exploit sortedness

Navigate r-w tradeoff using the sortedness-axis

Summary

Thank you!

BoDS lays groundwork for sortedness-aware testing

Classical indexes do not exploit sortedness

Navigate r-w tradeoff using the sortedness-axis