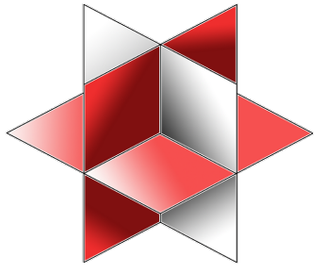
Succinct

Enabling Queries on Compressed Data

amplab

Distributed data stores

Fundamental building blocks for many cloud services



MICA

RAMCloud

MemC3

Records, primary key, secondary keys

Data stored as collection of records

USER	POSITION	BOSS	BOSS' Description
Anurag	Student	Rachit	Serious, Pushy
.....			
Rachit	Postdoc	Ion	Curious, funny

Records, primary key, secondary keys

Data stored as collection of records

Primary key

USER	POSITION	BOSS	BOSS' Description
Anurag	Student	Rachit	Serious, Pushy
.....			
Rachit	Postdoc	Ion	Curious, funny

Records, primary key, secondary keys

Data stored as collection of records

Secondary keys

USER	POSITION	BOSS	BOSS' Description
Anurag	Student	Rachit	Serious, Pushy
.....			
Rachit	Postdoc	Ion	Curious, funny

Queries in data stores

Queries on primary keys

- GET, PUT, DELETE
- Very efficient [MICA, Redis, RAMCloud]

Queries in data stores

Queries on primary keys

- GET, PUT, DELETE
- Very efficient [MICA, Redis, RAMCloud]

Queries on secondary keys

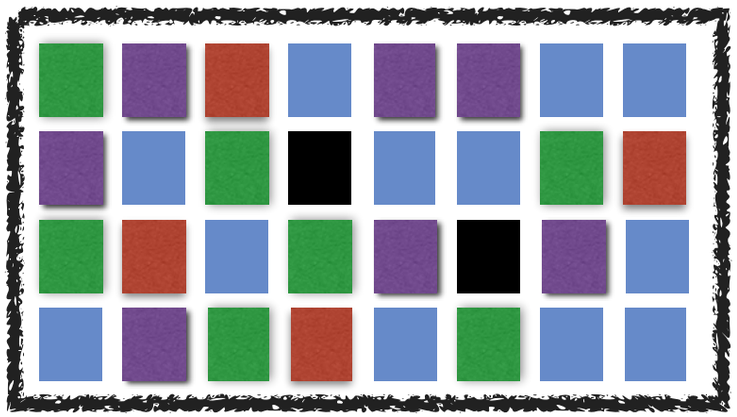
- E.g., search
- Executed via
  - Data scans [columnar stores]
  - Indexes [Cassandra, MongoDB]
- Either slow or large storage overhead

Flat (unstructured) files

For now, focus on:

- Queries on flat (unstructured) files
- Very powerful primitive
    - Allows queries on semi-structured data
    - Discussed later

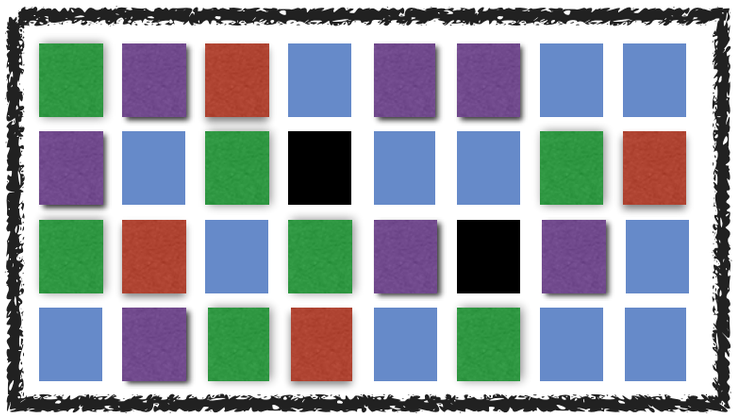
Example: Search(■ , file)

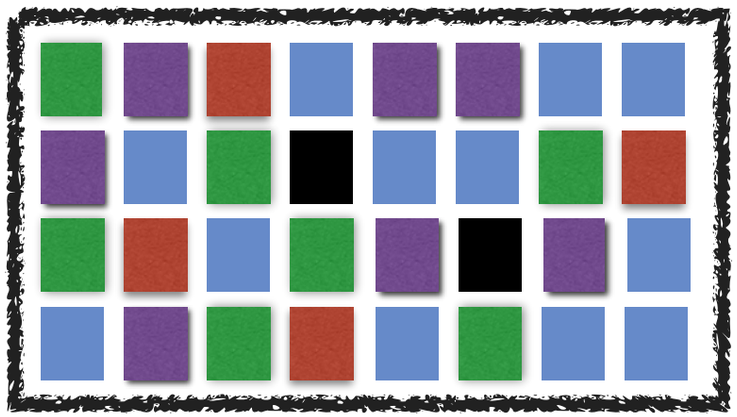


Search(■)

Example: Search(■ , file)

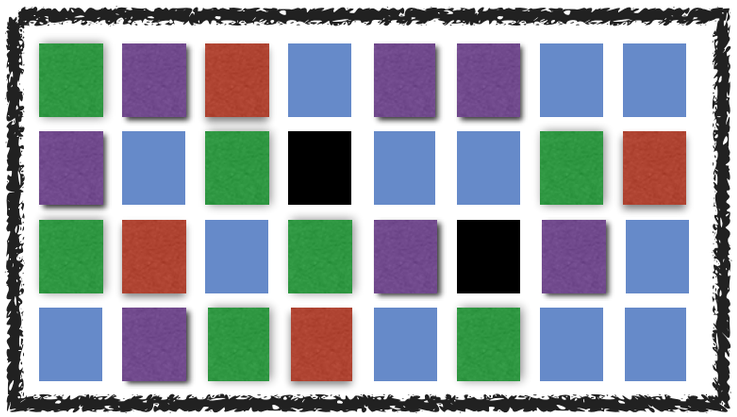
Data Scans





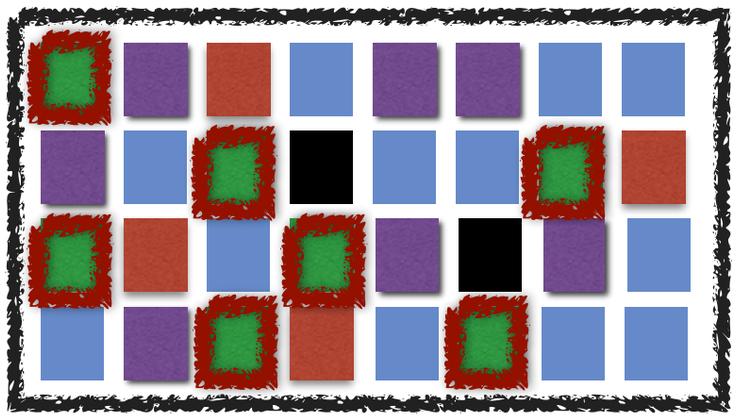
Search(■)

Example: Search(■ , file)

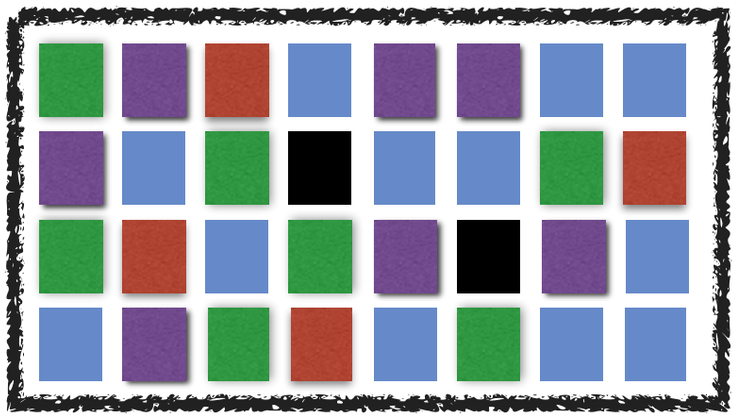


Search(■)

Data Scans

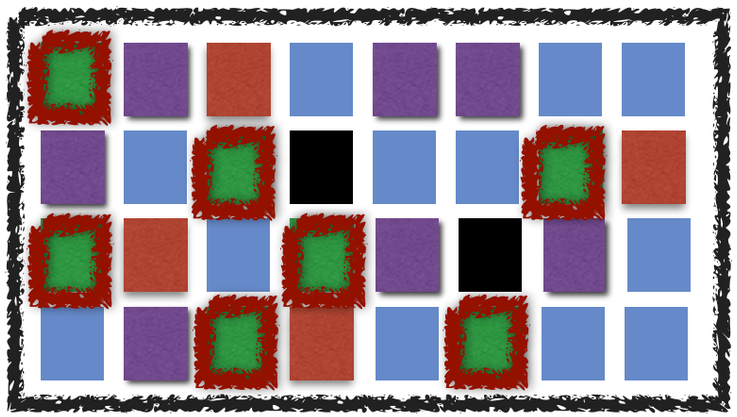


Example: Search(■ , file)



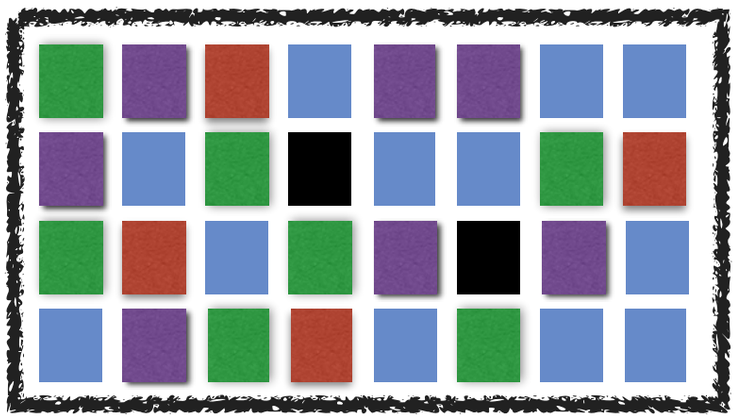
Search(■)

Data Scans



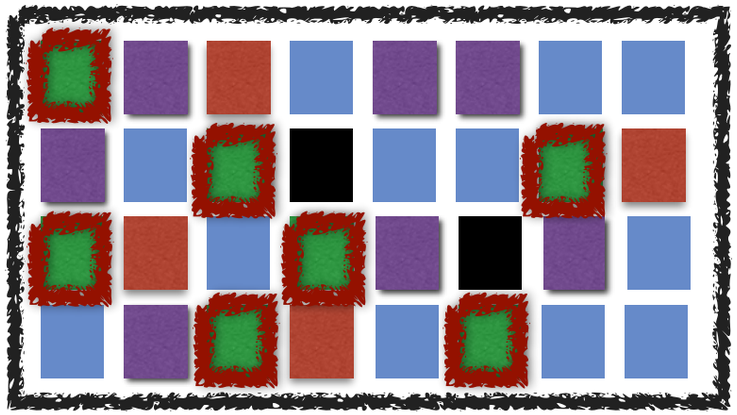
Low storage
High Latency

Example: Search(■ , file)



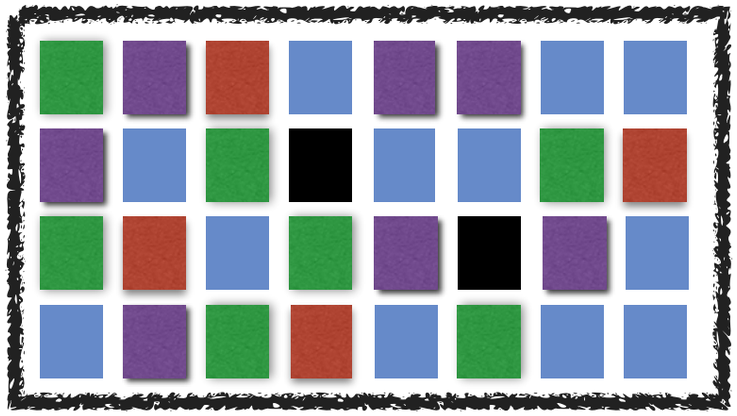
Search(■)

Data Scans

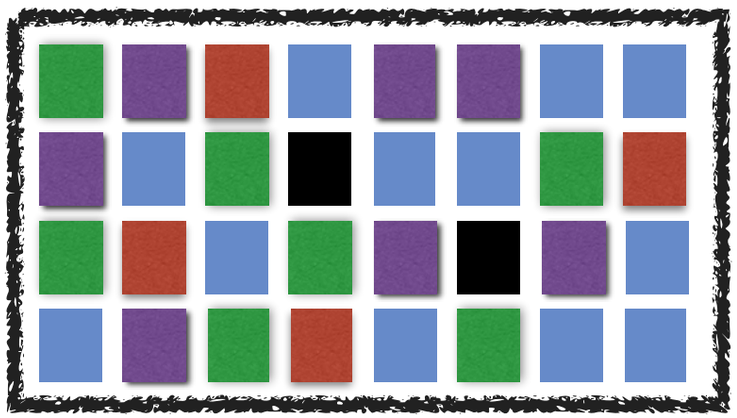


Low storage
High Latency

Indexes

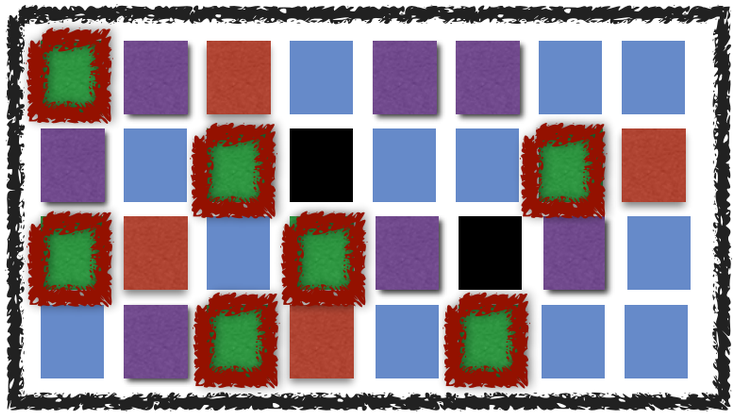


Example: Search(■ , file)



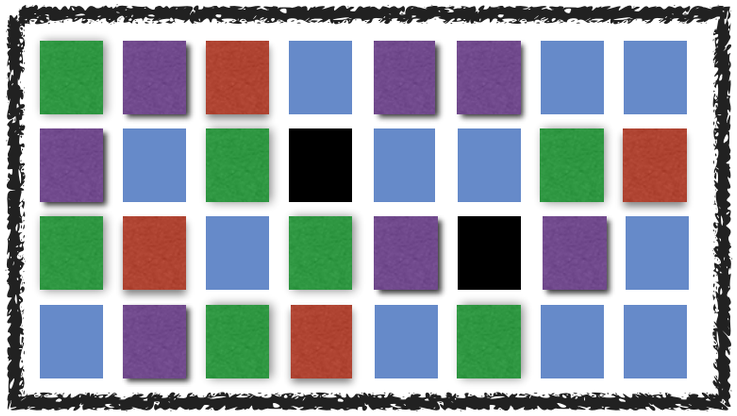
Search(■)

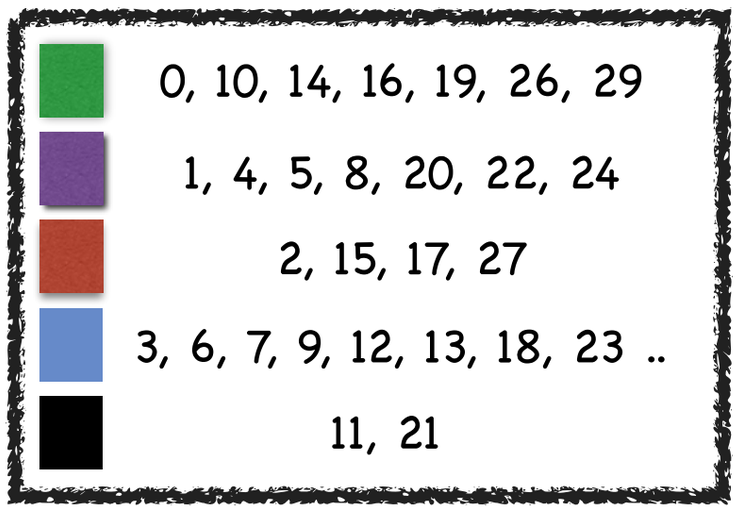
Data Scans



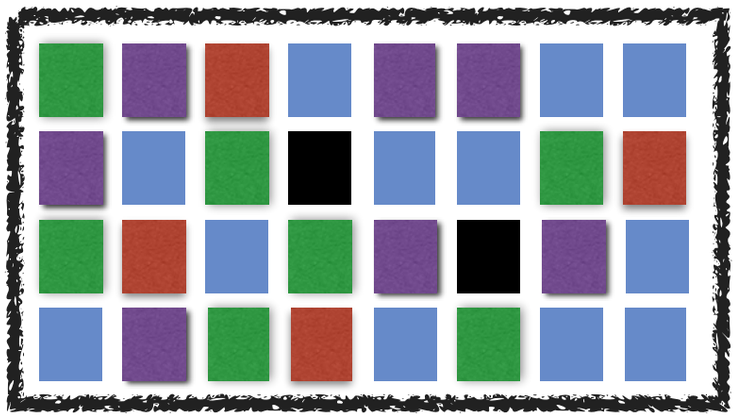
Low storage
High Latency

Indexes



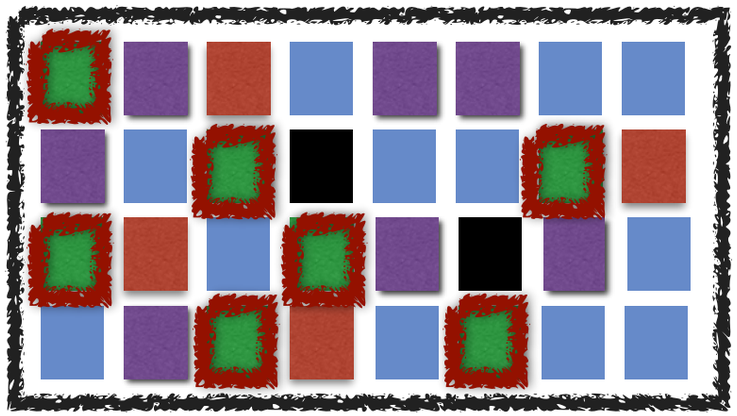


Example: Search(■ , file)



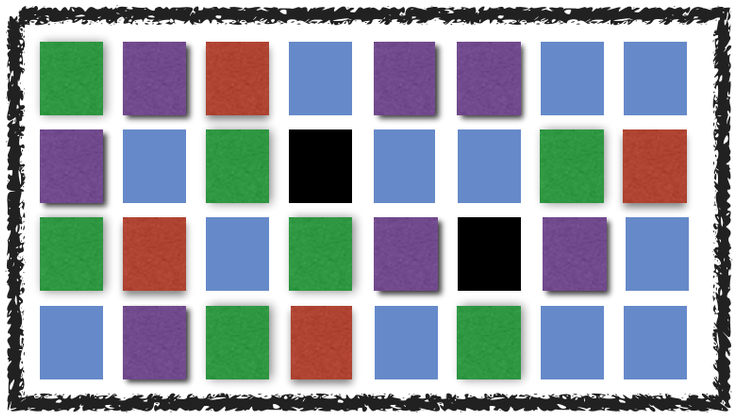
Search(■)

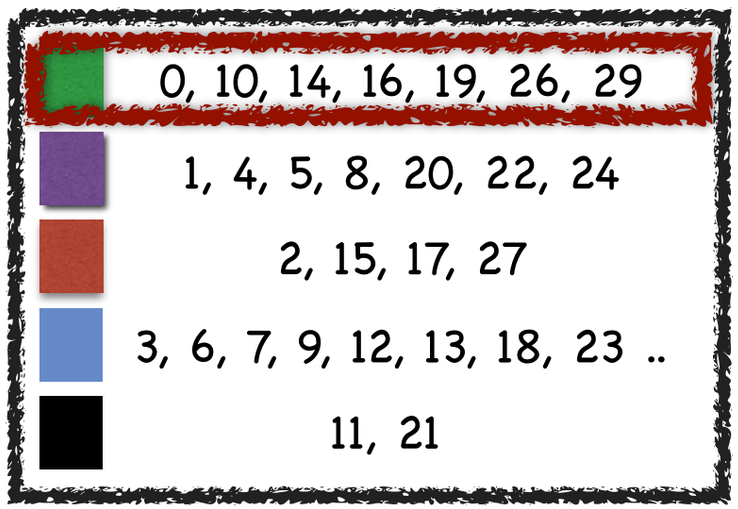
Data Scans



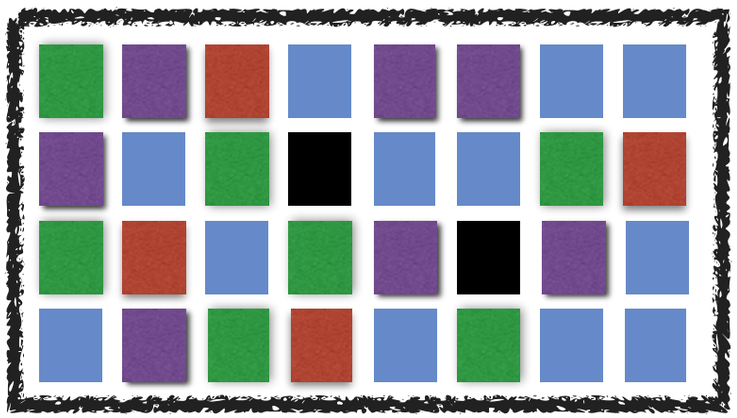
Low storage
High Latency

Indexes



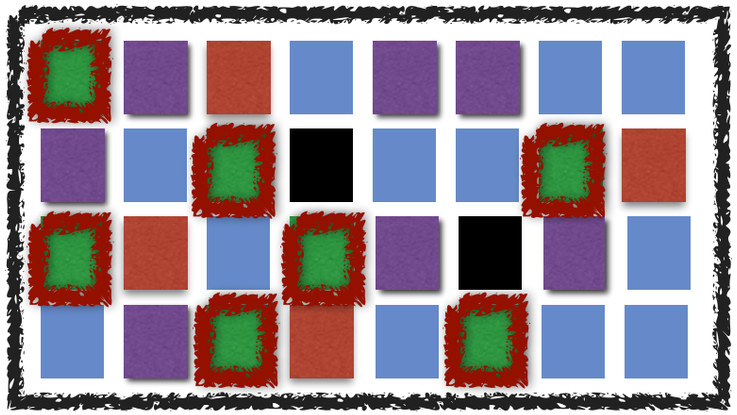


Example: Search(■ , file)



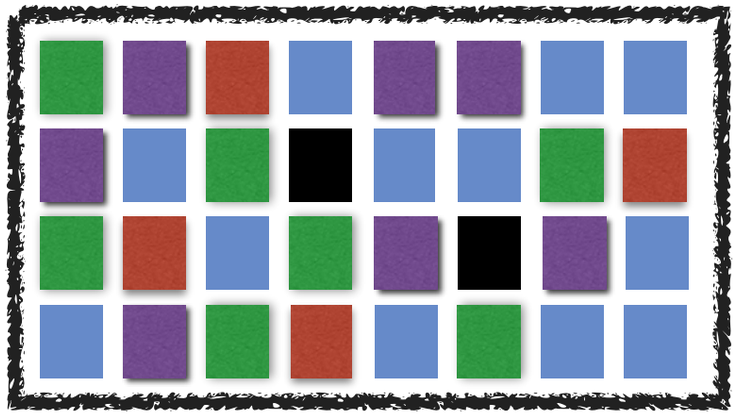
Search(■)

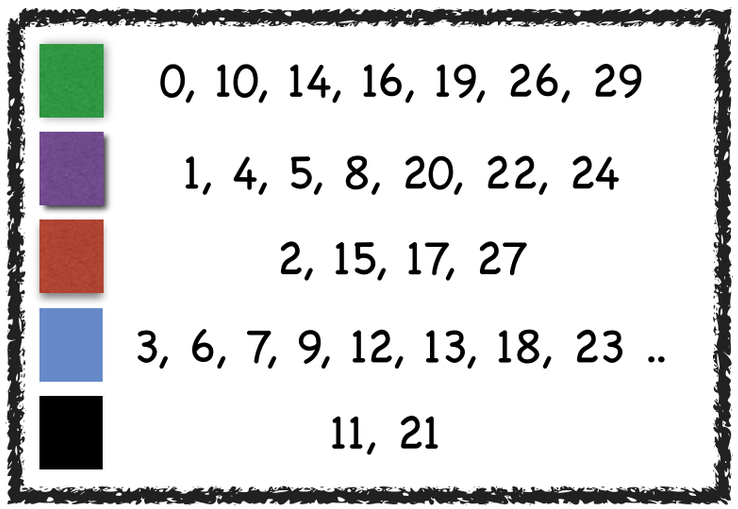
Data Scans



Low storage
High Latency

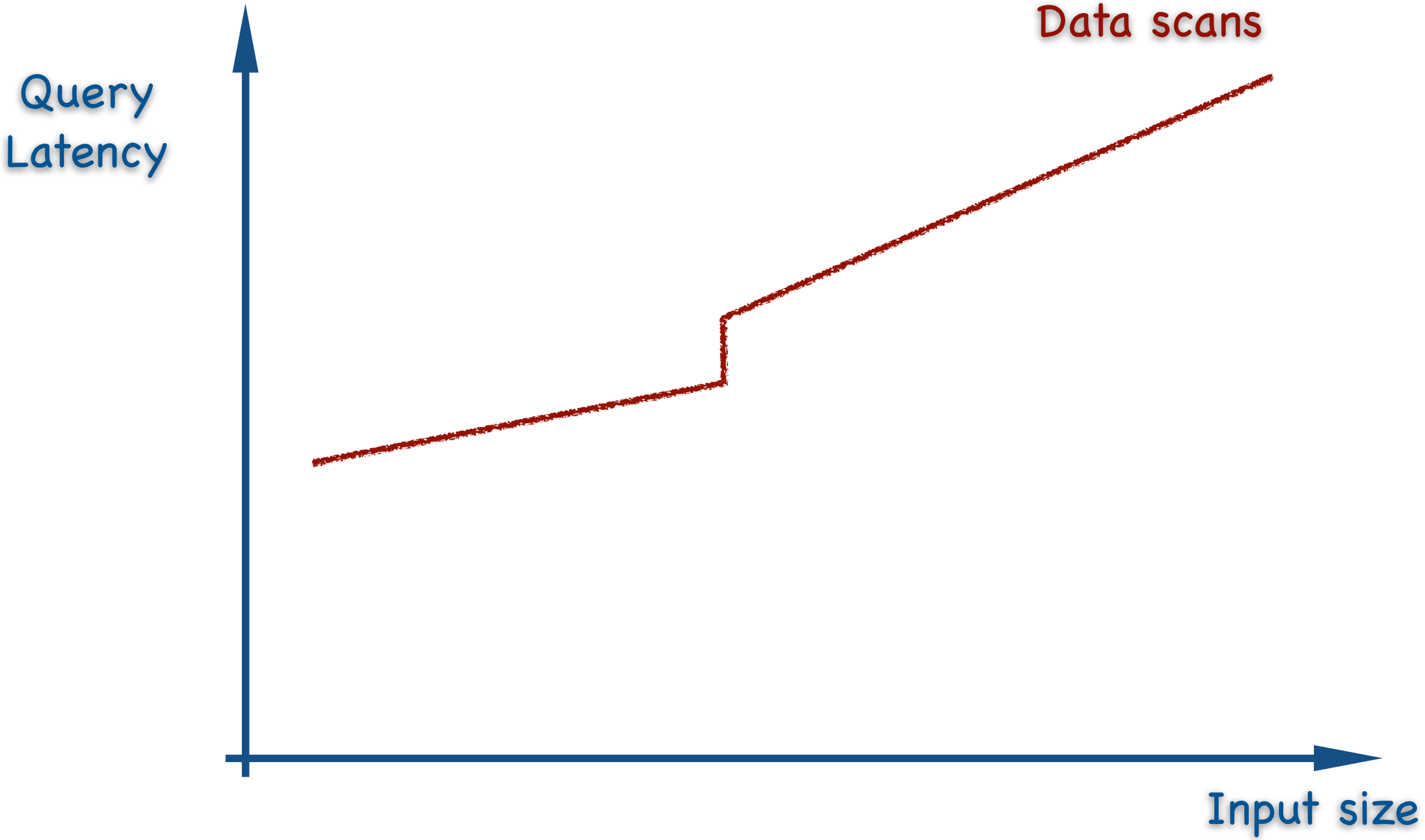
Indexes



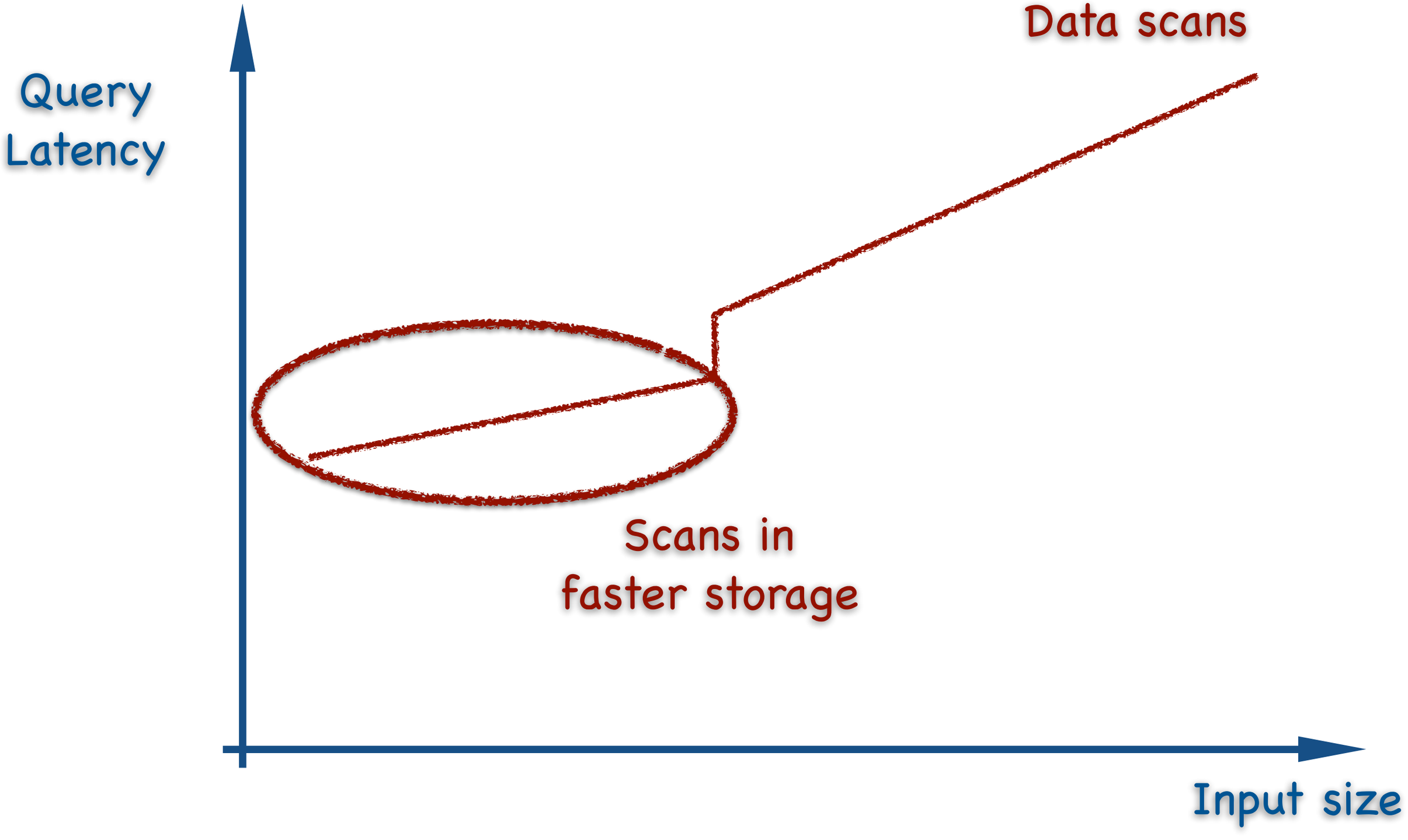


High storage
Low Latency

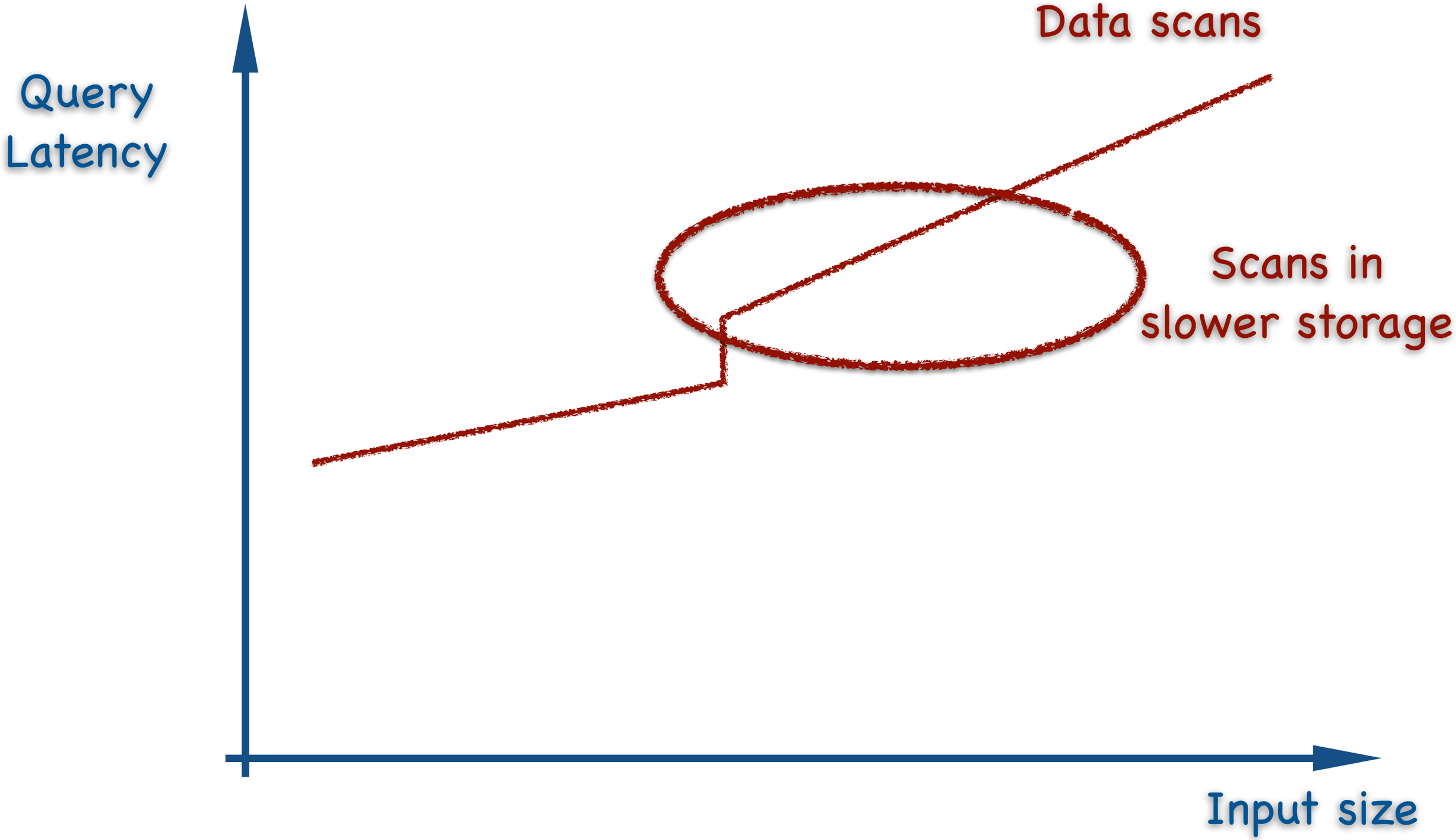
Latency with increasing input size



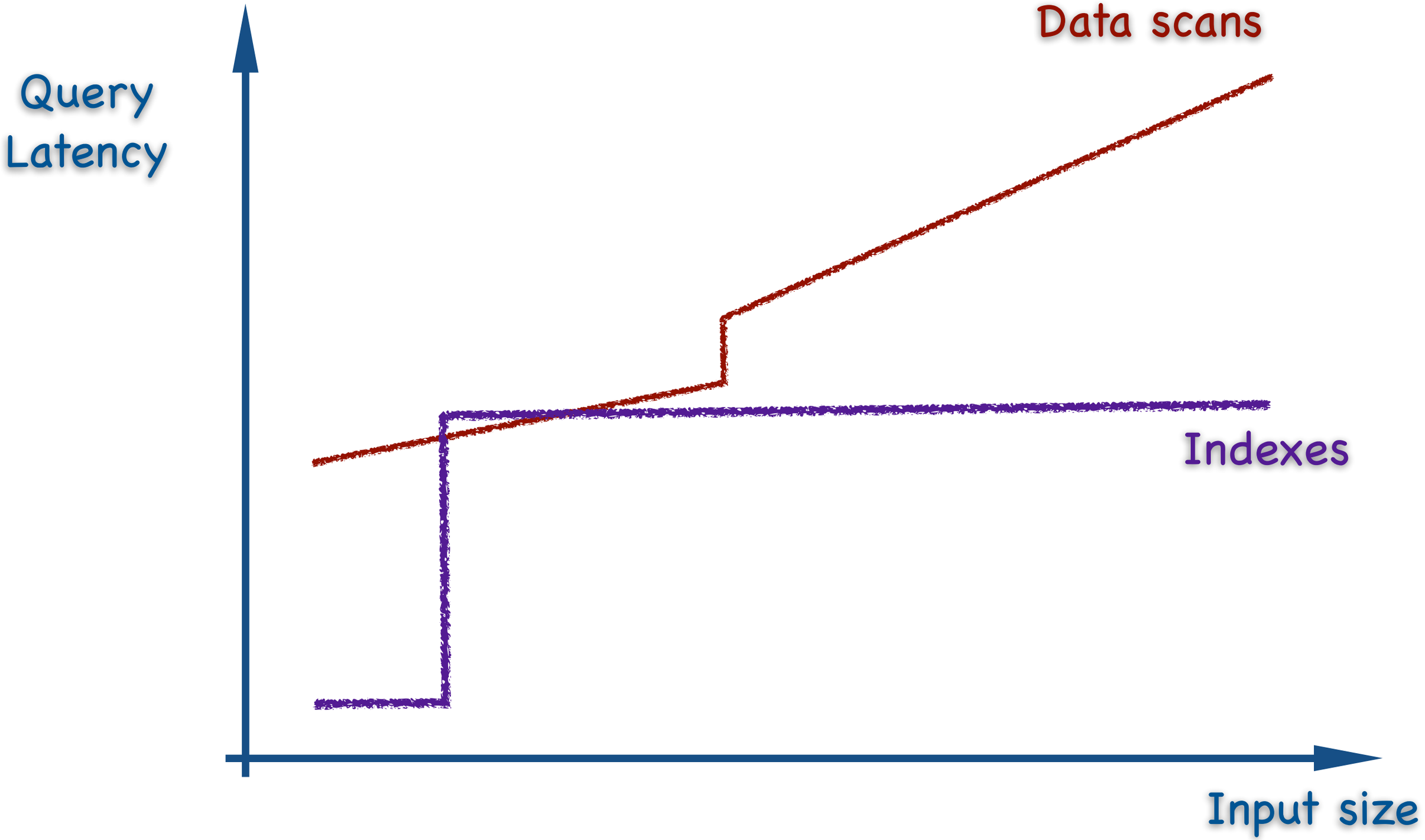
Latency with increasing input size



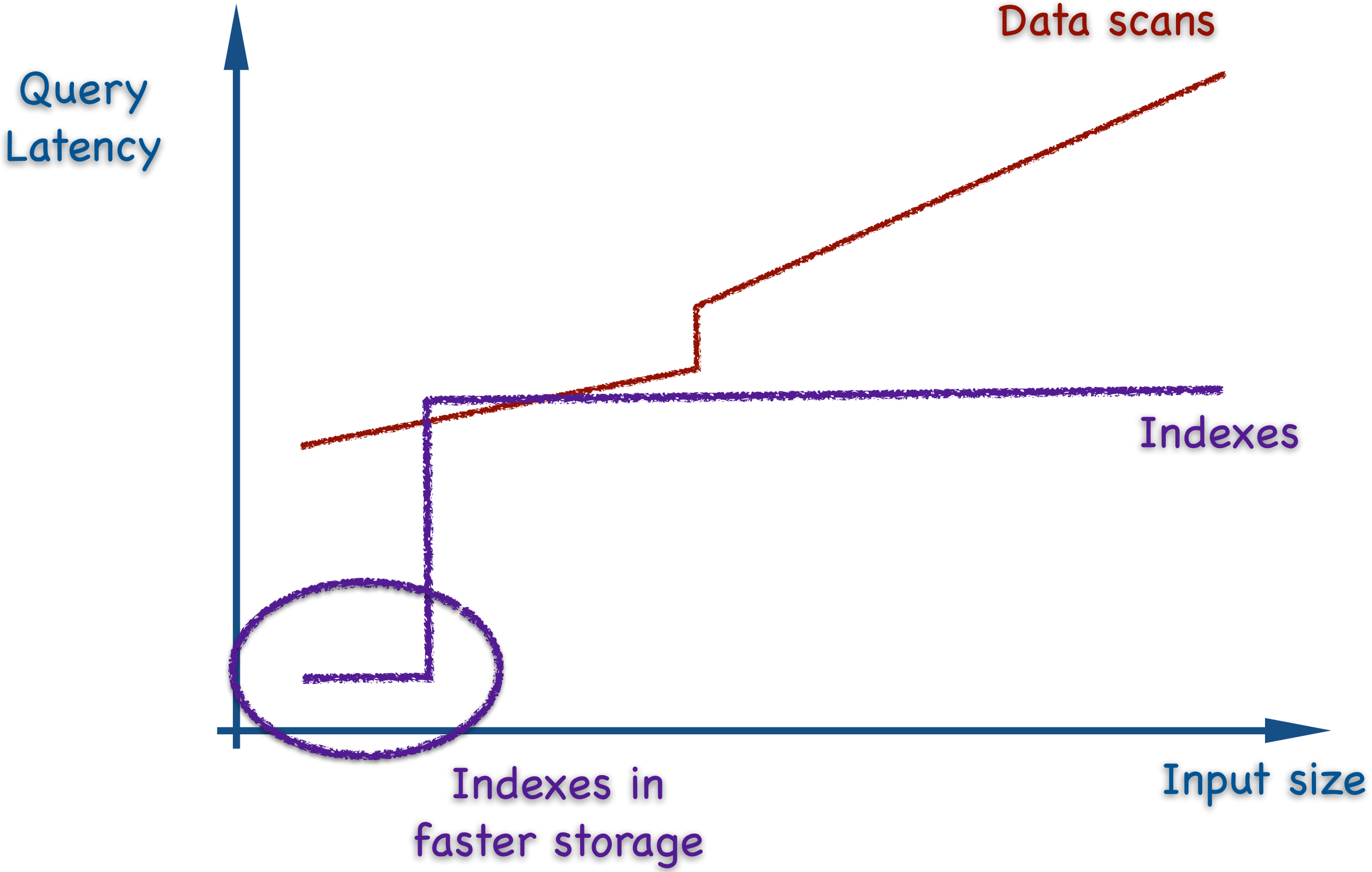
Latency with increasing input size



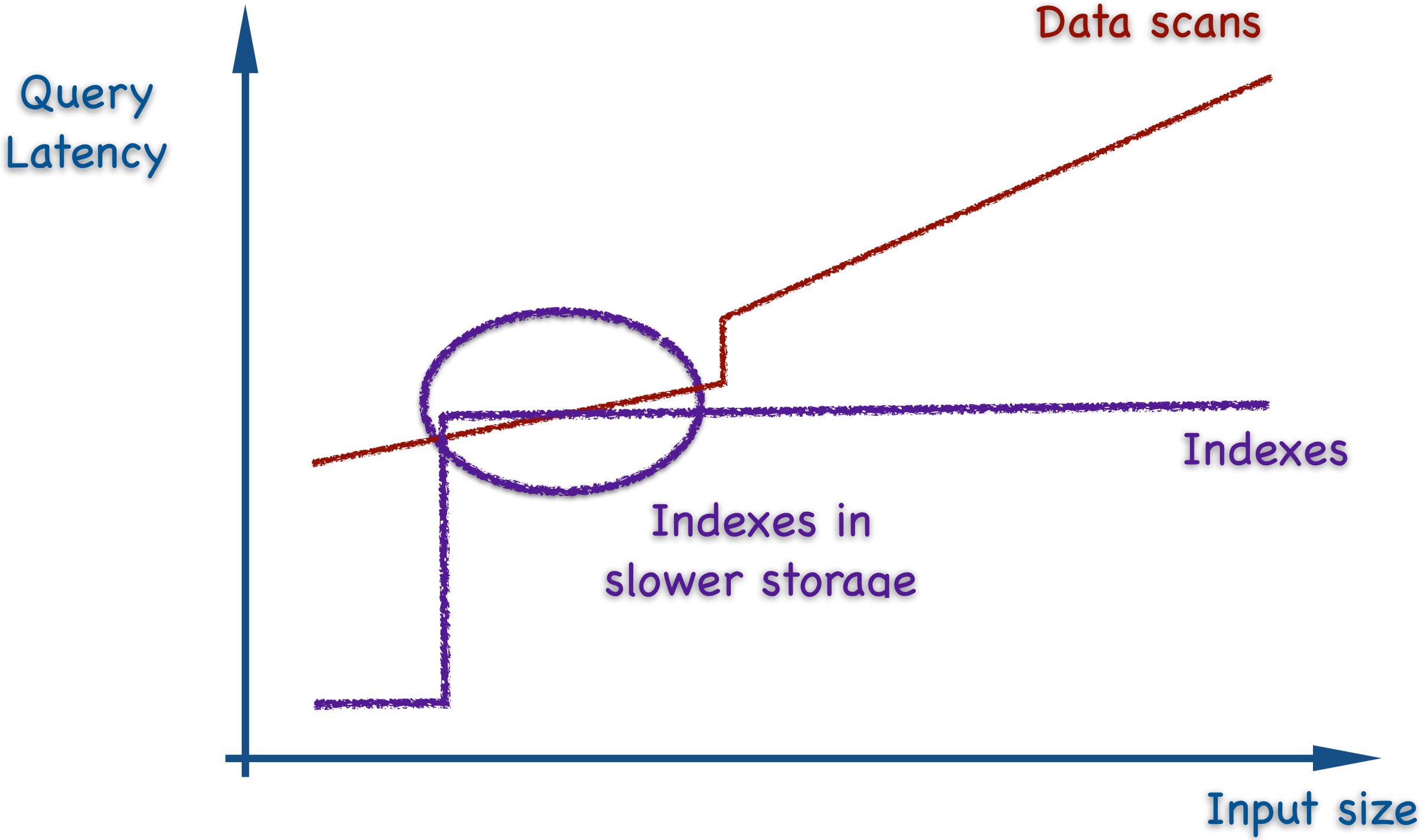
Latency with increasing input size



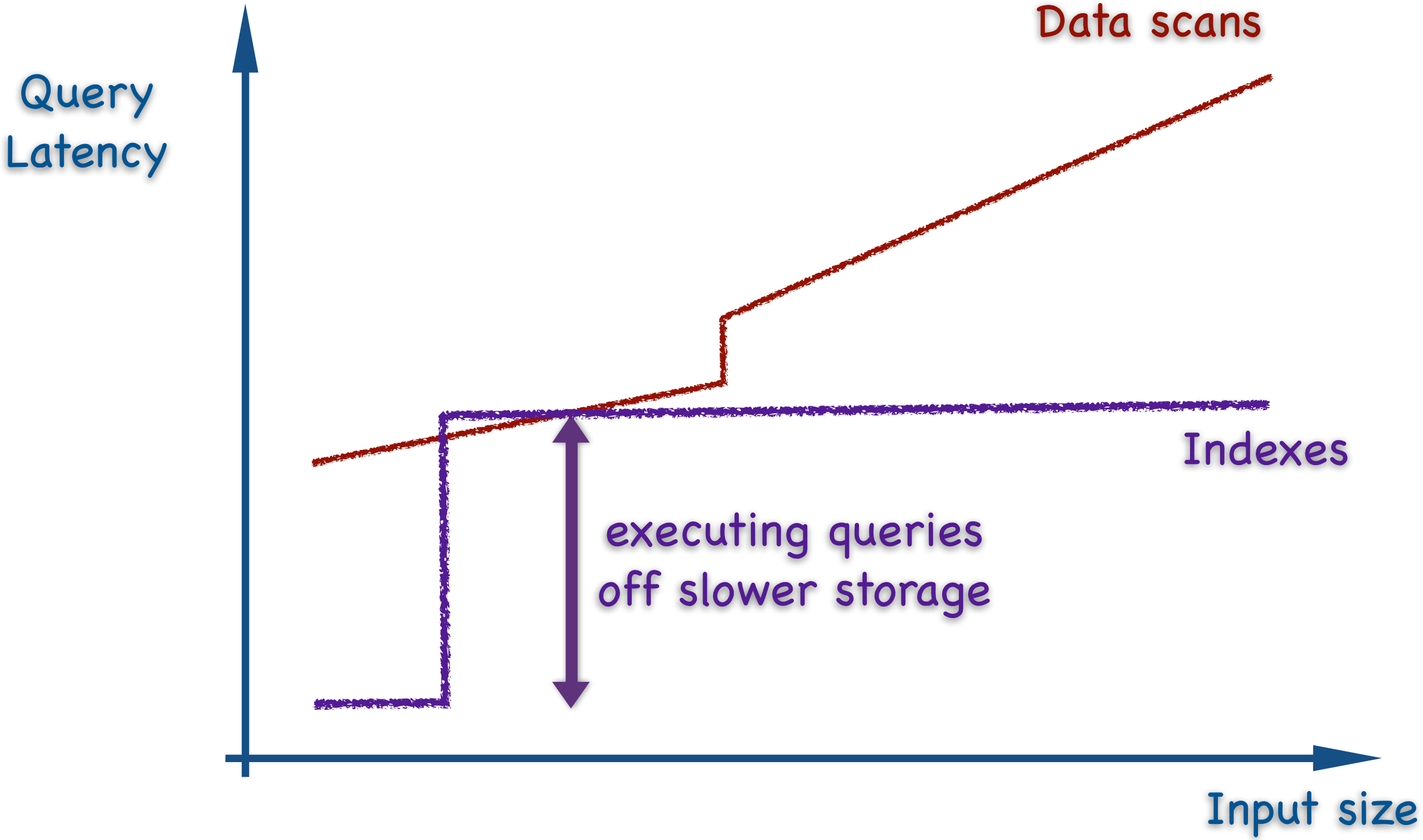
Latency with increasing input size



Latency with increasing input size



Latency with increasing input size



Succinct

Push more data in faster storage

- Storage < Input data size

Succinct

Push more data in faster storage

- Storage < Input data size

Execute queries directly on compressed data

- No data scans
- No data decompression

Succinct

Push more data in faster storage

- Storage < Input data size

Execute queries directly on compressed data

- No data scans
- No data decompression

Target: read-heavy append-only workloads

Succinct

Push more data in faster storage

- Storage < Input data size

Execute queries directly on compressed data

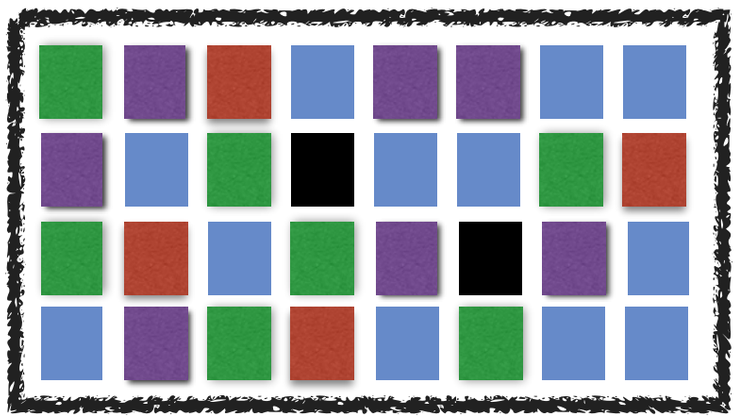
- No data scans
- No data decompression

Target: read-heavy append-only workloads

Non-goals: Fault tolerance, data consistency

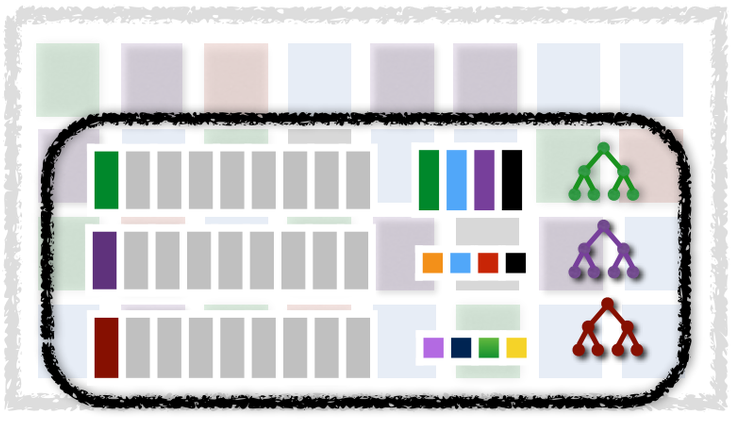
Example: Search(■ , file)

Succinct



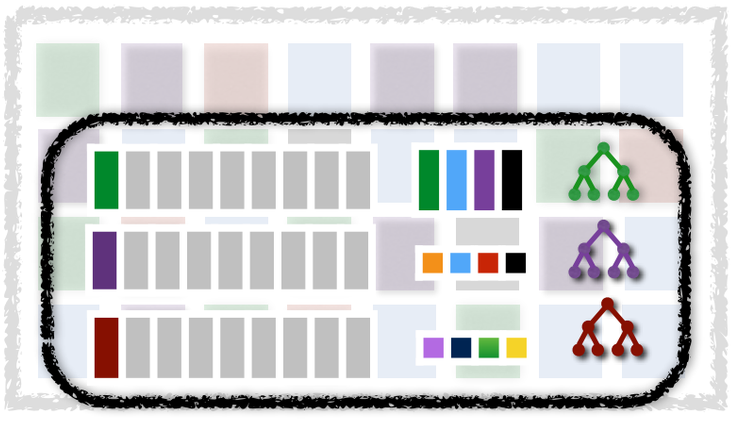
Example: Search(■ , file)

Succinct



Example: Search(■ , file)

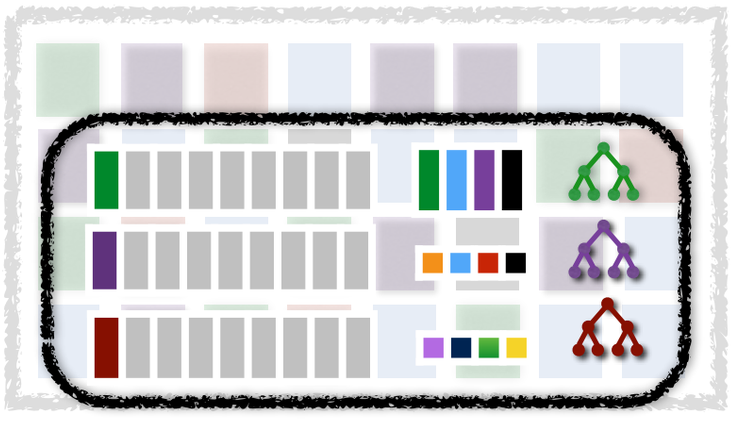
Succinct



Queries executed directly on the compressed representation

Example: Search(■ , file)

Succinct

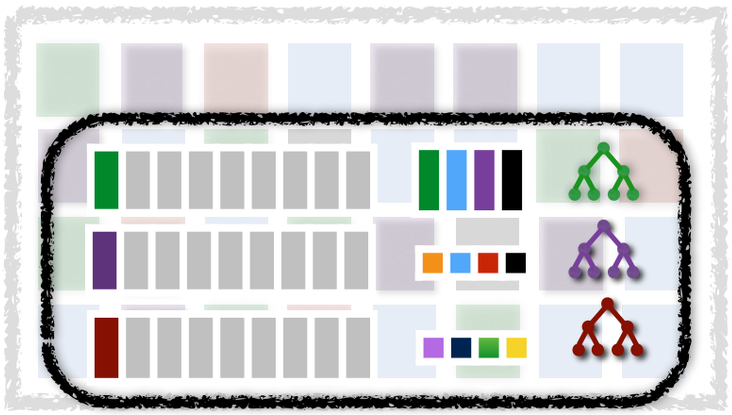


Queries executed directly on the compressed representation

Low storage
Low Latency

Example: Search(■ , file)

Succinct



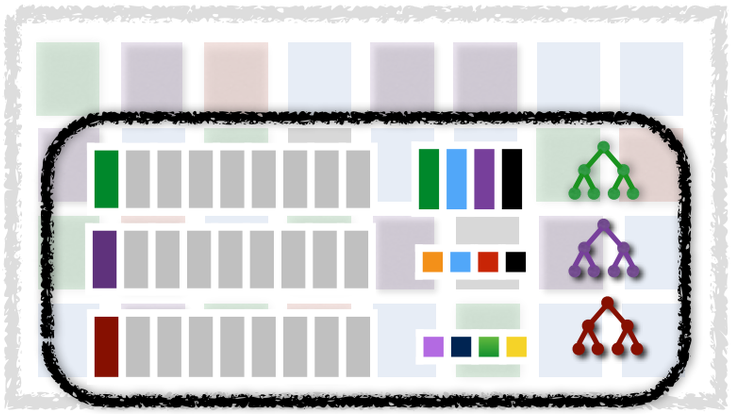
Queries executed directly on the compressed representation

Low storage
Low Latency

What makes Succinct unique

Example: Search(■ , file)

Succinct



Queries executed directly on the compressed representation

Low storage
Low Latency

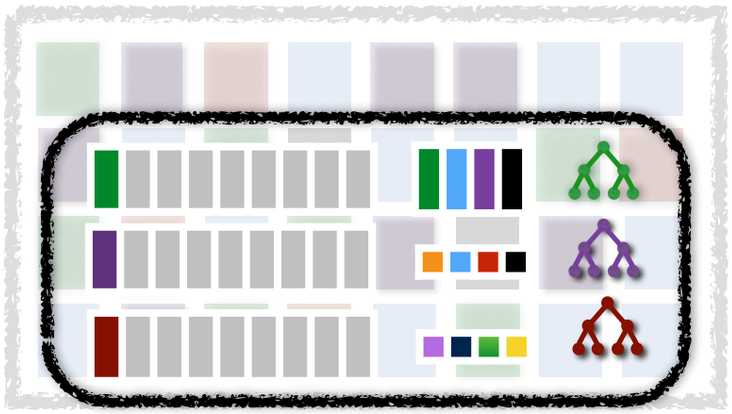
What makes Succinct unique

No additional indexes

Query responses embedded within the compressed representation

Example: Search(■ , file)

Succinct



Queries executed directly on the compressed representation

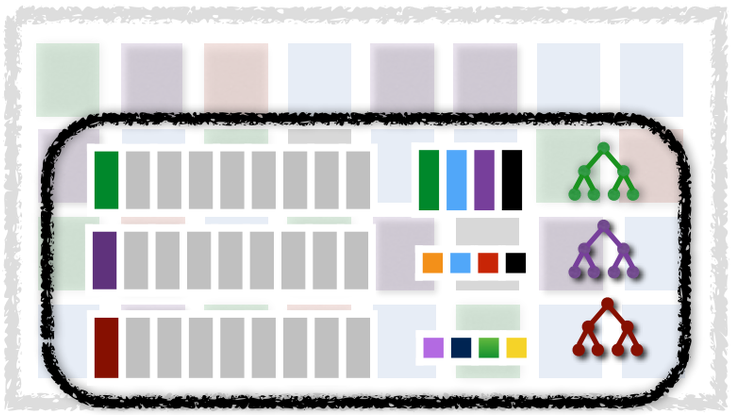
Low storage
Low Latency

What makes Succinct unique

No additional indexes	Query responses embedded within the compressed representation
No data scans	Functionality of indexes

Example: Search(■ , file)

Succinct



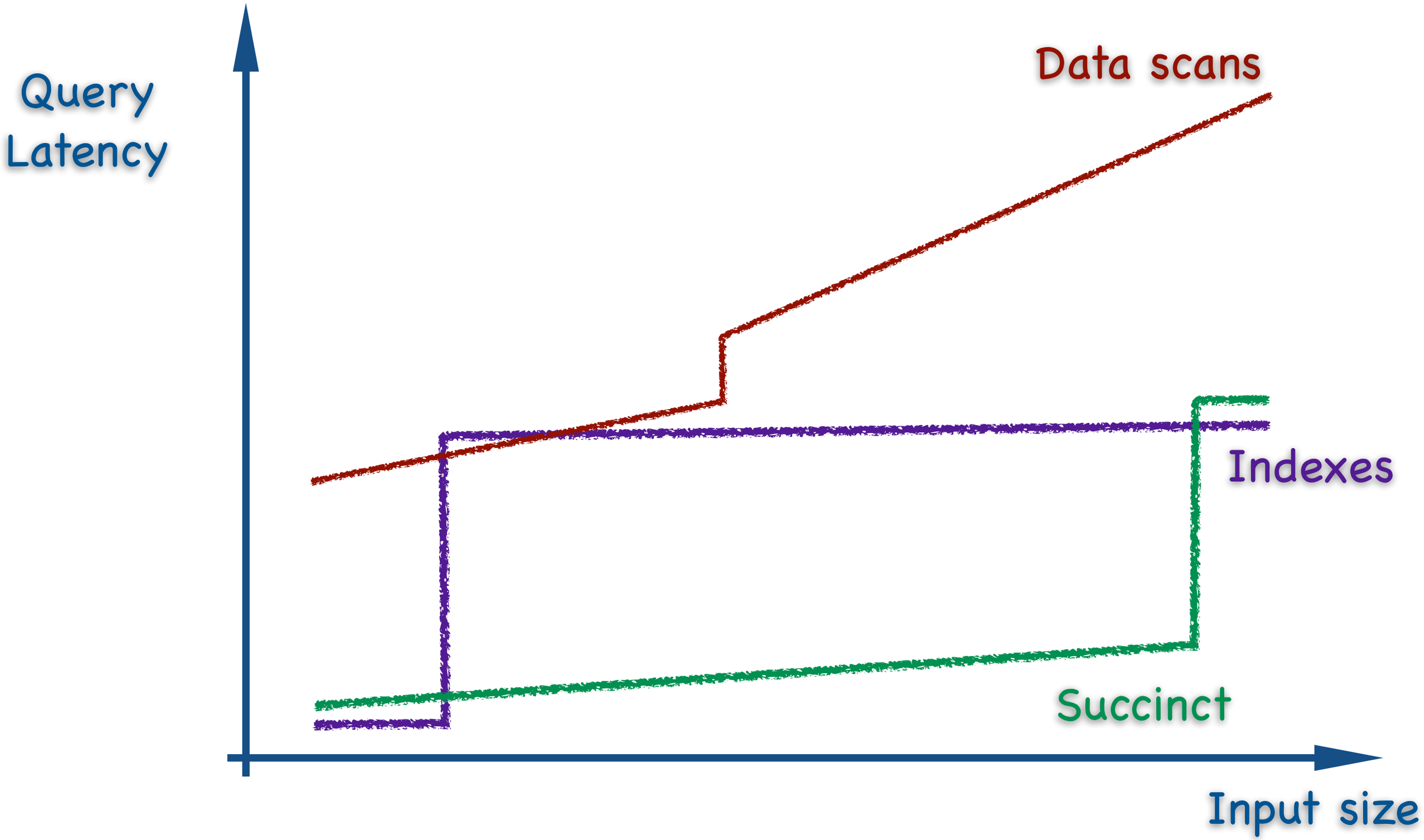
Queries executed directly on the compressed representation

Low storage
Low Latency

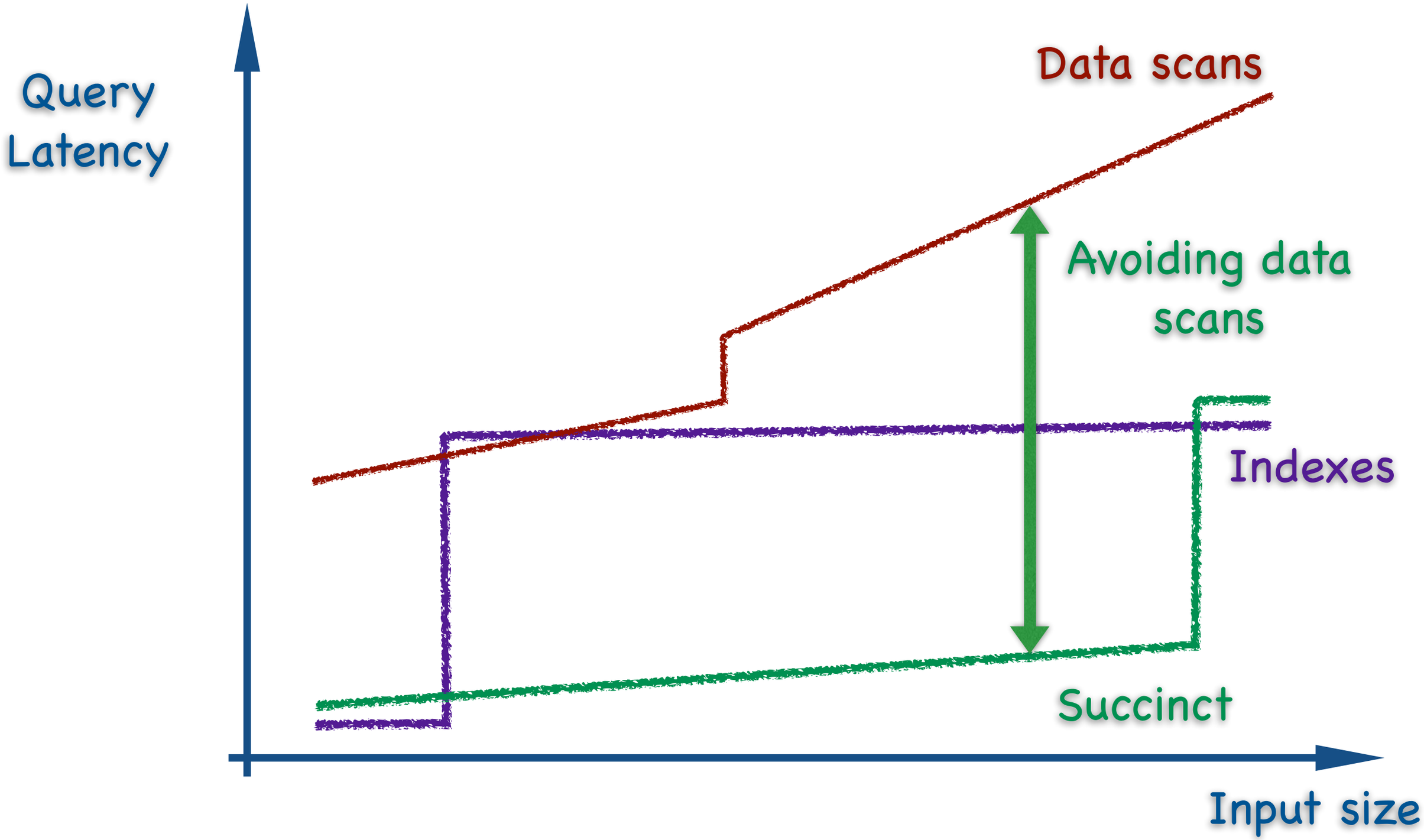
What makes Succinct unique

No additional indexes	Query responses embedded within the compressed representation
No data scans	Functionality of indexes
No decompression	Queries directly on the compressed representation (except for data access queries)

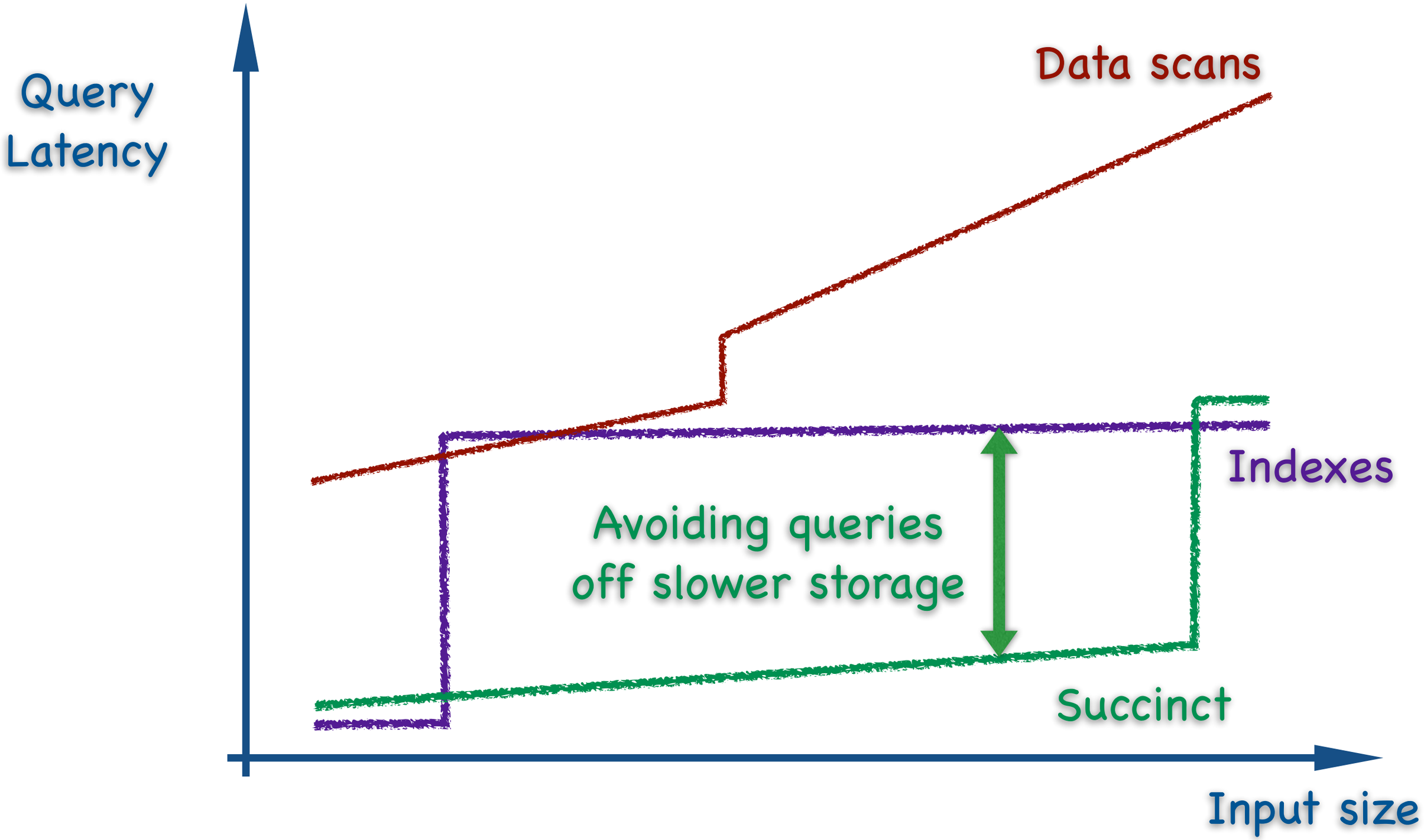
Search latency, qualitatively



Search latency, qualitatively



Search latency, qualitatively



What do we lose?

Succinct

- No secondary indexes
- No data scans, no data decompression

What do we lose?

Succinct

- No secondary indexes
- No data scans, no data decompression

What do we lose?

What do we lose?

Succinct

- No secondary indexes
- No data scans, no data decompression

What do we lose?

- Preprocessing expensive (4GB/hour/core)
- CPU (for data access)
- Sequential scan throughput (13Mbps/thread)
- In-place updates

Succinct Data Representation

Builds upon a large body of theory results

Succinct Data Representation

Builds upon a large body of theory results

- FM-index
    - BWT-based technique
- Compressed Suffix arrays
    - Suffix array based technique

Succinct Data Representation

Builds upon a large body of theory results

- FM-index
  - BWT-based technique
- Compressed Suffix arrays
  - Suffix array based technique

Succinct

- Builds upon latter due to simplicity
- Improved data structures (1.25—3x more space efficient)
- New query algorithm (2.3x faster on average)

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

Step 1:
Construct set of suffixes

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

Step 1:
Construct set of suffixes

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

Step 1:
Construct set of suffixes

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

Step 1:
Construct set of suffixes

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

a	p	p	y	p	u	p	p	y

Step 1:
Construct set of suffixes

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

a	p	p	y	p	u	p	p	y

p	p	y	p	u	p	p	y

p	y	p	u	p	p	y

y	p	u	p	p	y

p	u	p	p	y

u	p	p	y

p	p	y

p	y

y

Step 1:
Construct set of suffixes

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

a	p	p	y	p	u	p	p	y

p	p	y	p	u	p	p	y

p	y	p	u	p	p	y

y	p	u	p	p	y

p	u	p	p	y

u	p	p	y

p	p	y

p	y

y

Step 2:
Sort suffixes (lexicographically)

Array of suffixes (AoS)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

p	p	y

Array of suffixes (AoS)

p	p	y	p	u	p	p	y

p	u	p	p	y

p	y

p	y	p	u	p	p	y

u	p	p	y

y

y	p	u	p	p	y

Suffix array (AoS2Input)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

Step 3:
Mark location of each sorted suffix (in input file)

a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

p	p	y

p	p	y	p	u	p	p	y

p	u	p	p	y

p	y

p	y	p	u	p	p	y

u	p	p	y

y

y	p	u	p	p	y

Suffix array (AoS2Input)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

Step 3:
Mark location of each sorted suffix (in input file)

a	p	p	y	p	u	p	p	y
h	a	p	p	y	p	u	p	p	y
p	p	y						
p	p	y	p	u	p	p	y	
p	u	p	p	y				
p	y							
p	y	p	u	p	p	y		
u	p	p	y					
y								
y	p	u	p	p	y			

Suffix array (AoS2Input)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1	a p p y p u p p y
	h a p p y p u p p y
	p p y
	p p y p u p p y
	p u p p y
	p y
	p y p u p p y
	u p p y
	y
	y p u p p y

Step 3:
Mark location of each sorted suffix (in input file)

Suffix array (AoS2Input)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y

p	p	y

p	p	y	p	u	p	p	y

p	u	p	p	y

p	y

p	y	p	u	p	p	y

u	p	p	y

y

y	p	u	p	p	y

Step 3:
Mark location of each sorted suffix (in input file)

Suffix array (AoS2Input)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1	a p p y p u p p y
0	h a p p y p u p p y
	p p y
	p p y p u p p y
	p u p p y
	p y
	p y p u p p y
	u p p y
	y
	y p u p p y

Step 3:
Mark location of each sorted suffix (in input file)

Suffix array (AoS2Input)

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

AoS2Input

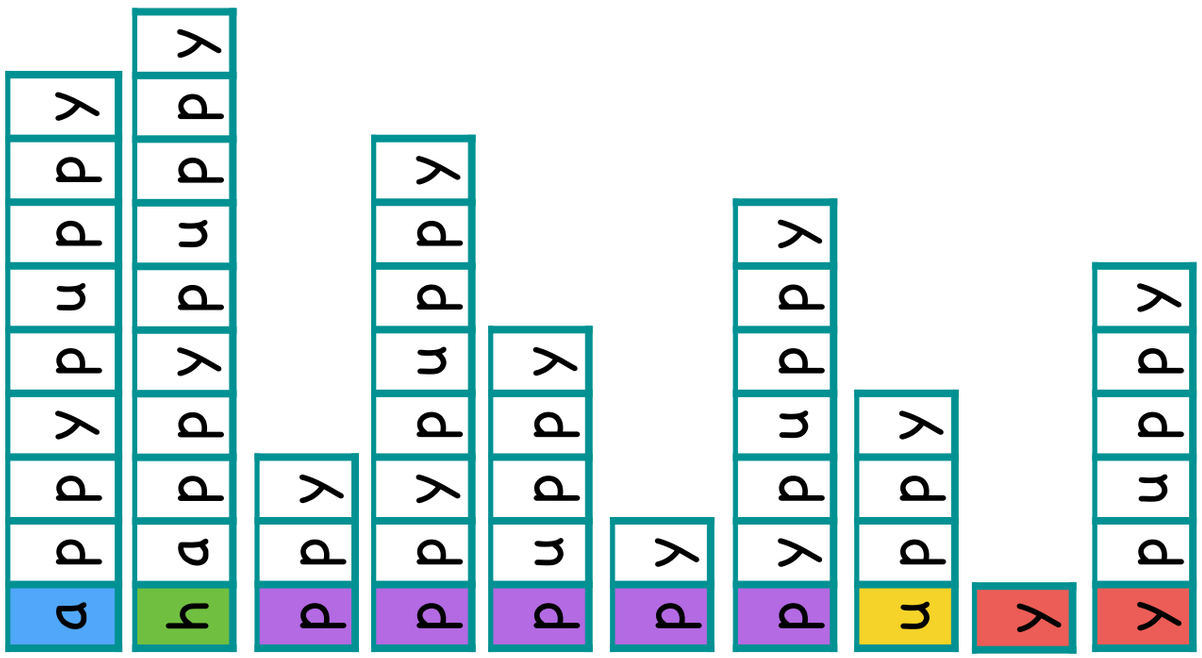
1	a p p y p u p p y
0	h a p p y p u p p y
7	p p y
2	p p y p u p p y
5	p u p p y
8	p y
3	p y p u p p y
6	u p p y
9	y
4	y p u p p y

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

Succinct Data Representation

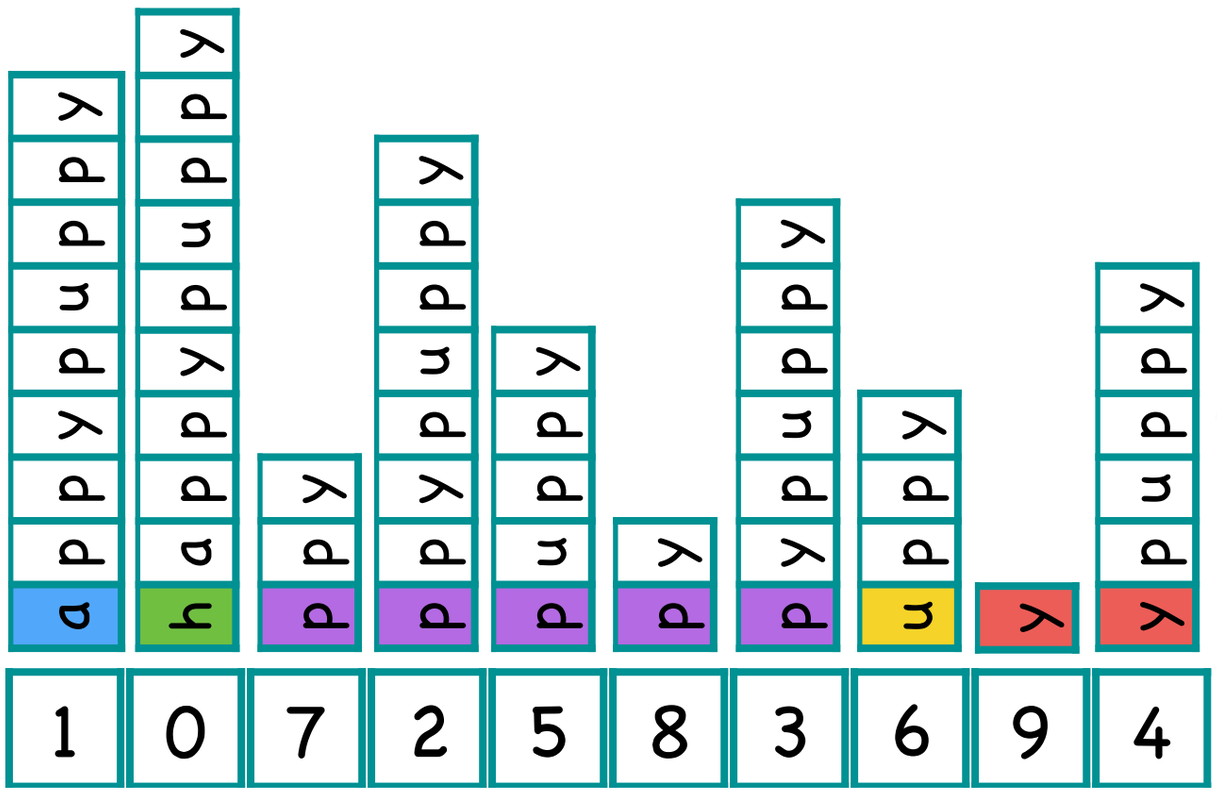
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



Array of Suffixes (AoS)
(suffixes in sorted order)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



Array of Suffixes (AoS)

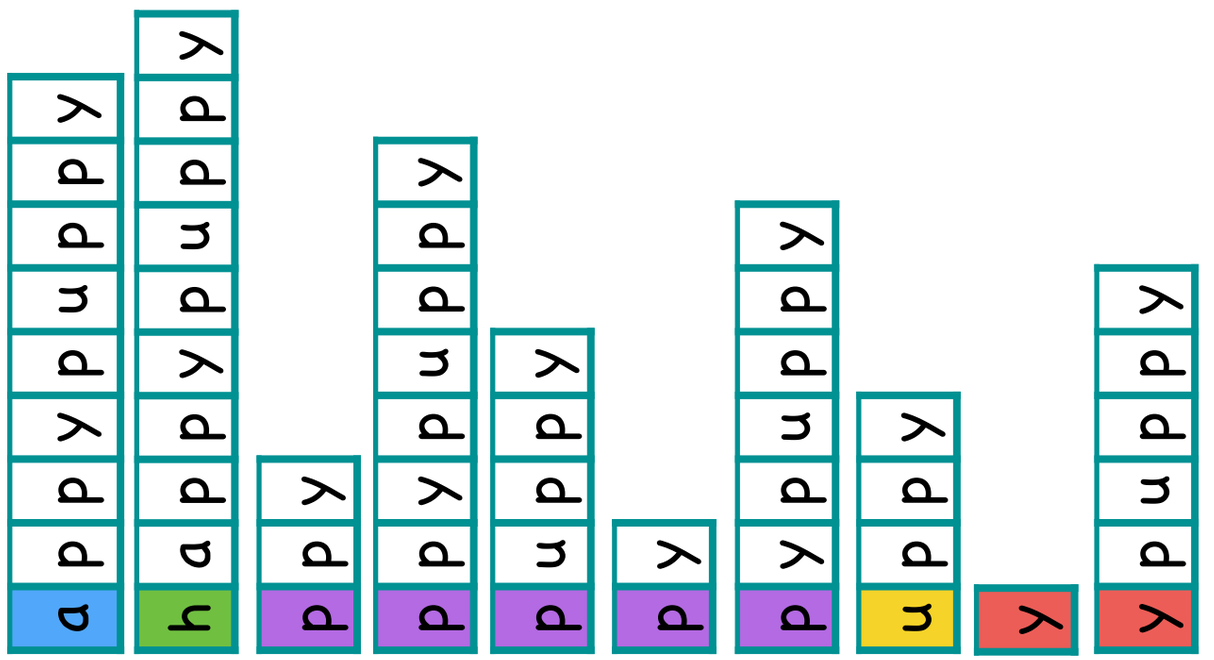
(suffixes in sorted order)

AoS2Input

(location of suffix in file)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



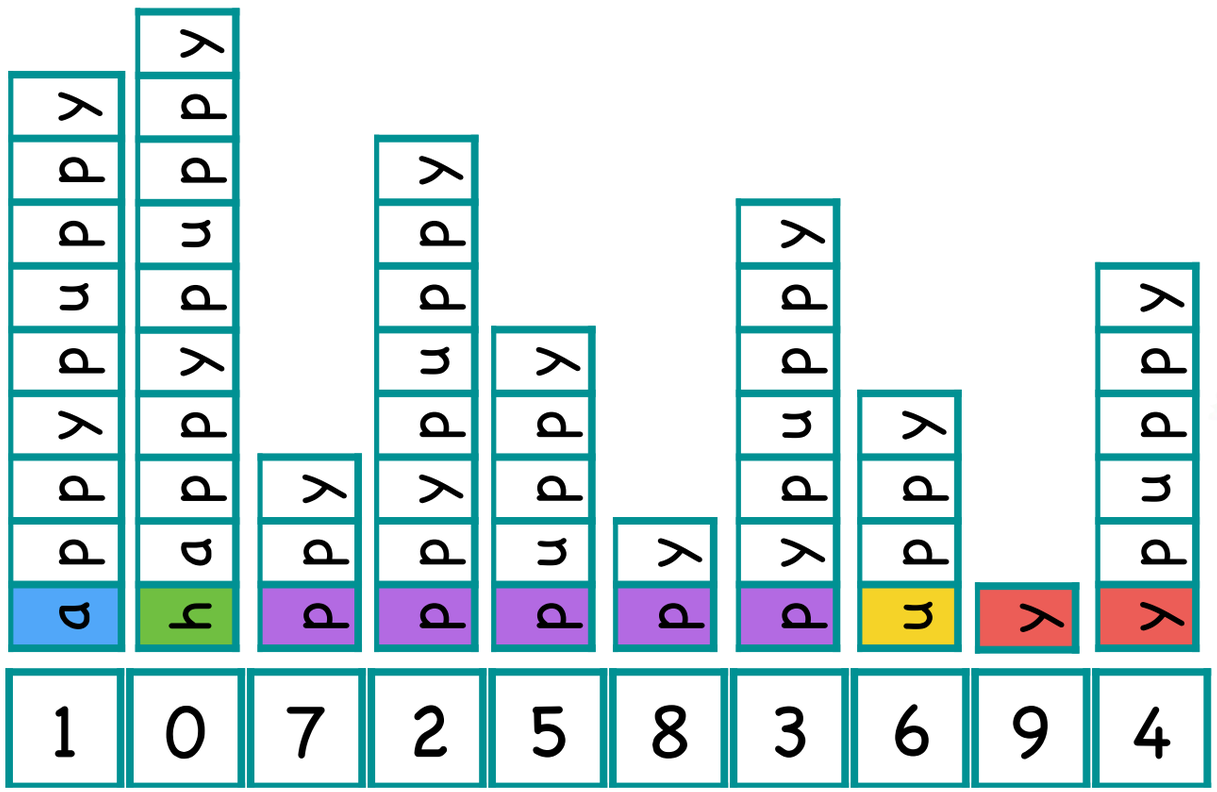
1	0	7	2	5	8	3	6	9	4

Arbitrary substring search

via binary search

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



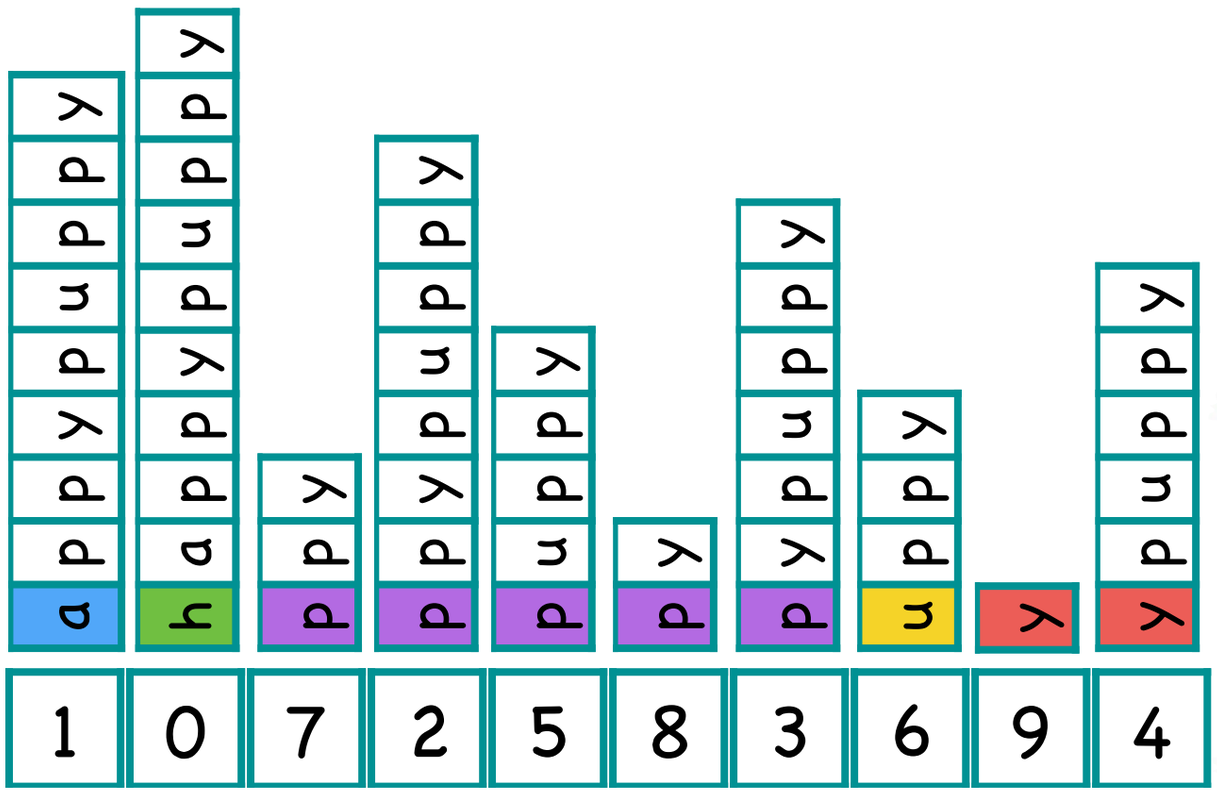
1	0	7	2	5	8	3	6	9	4

Problem1: Size of AoS

O(n*n) bits

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



Problem1: Size of AoS

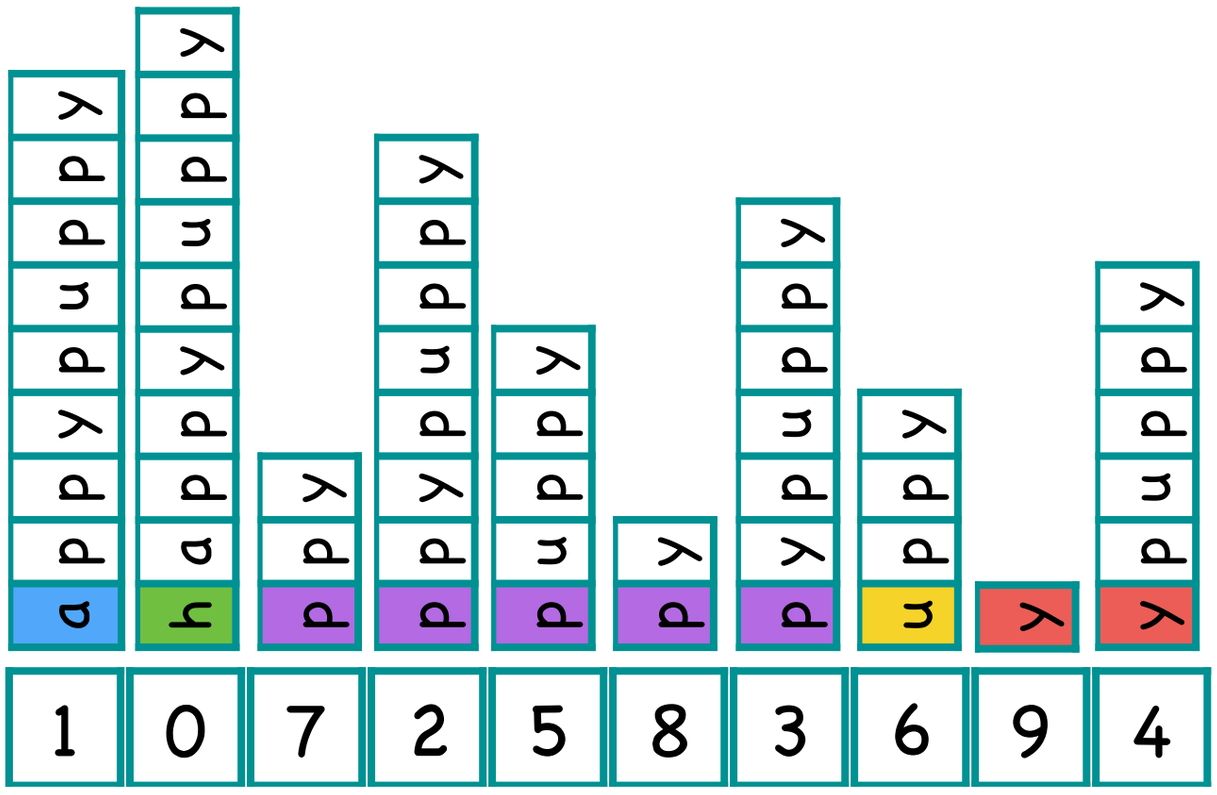
O(n*n) bits

Problem2: Size of AoS2Input

each entry takes log(n) bits

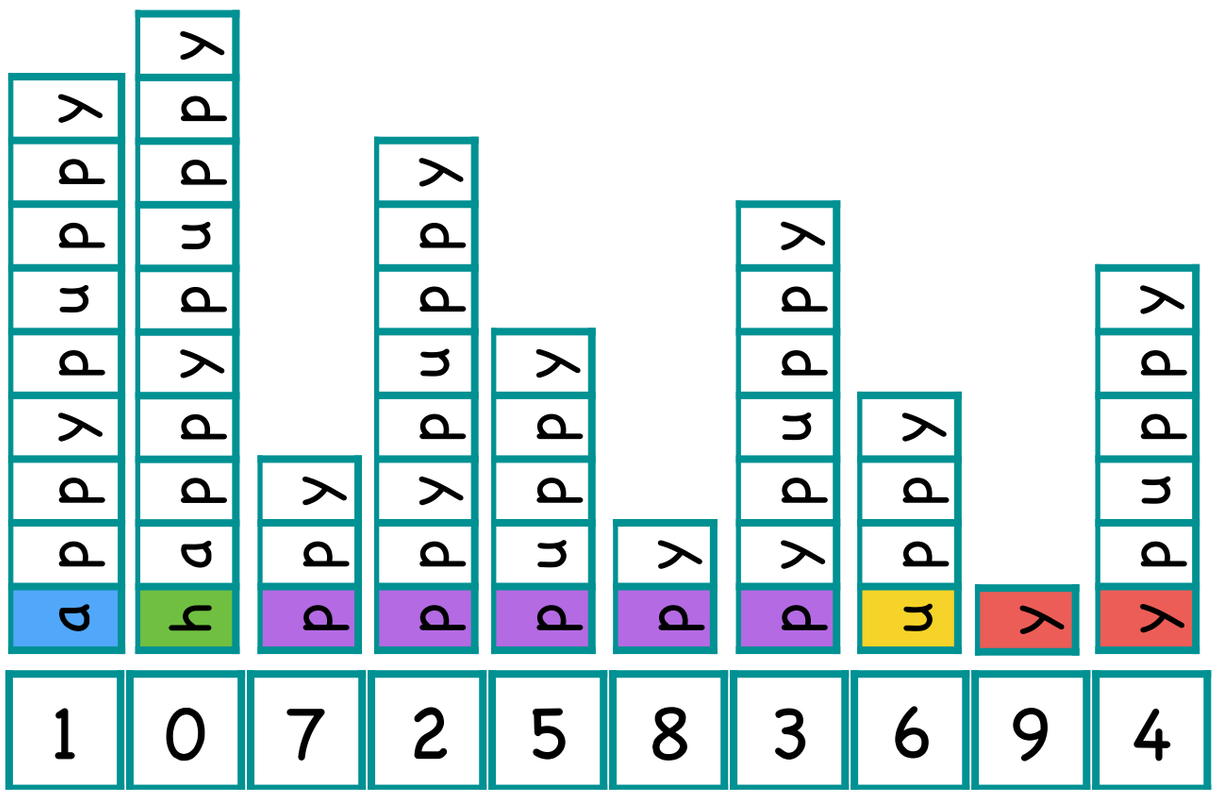
Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



Succinct Data Representation

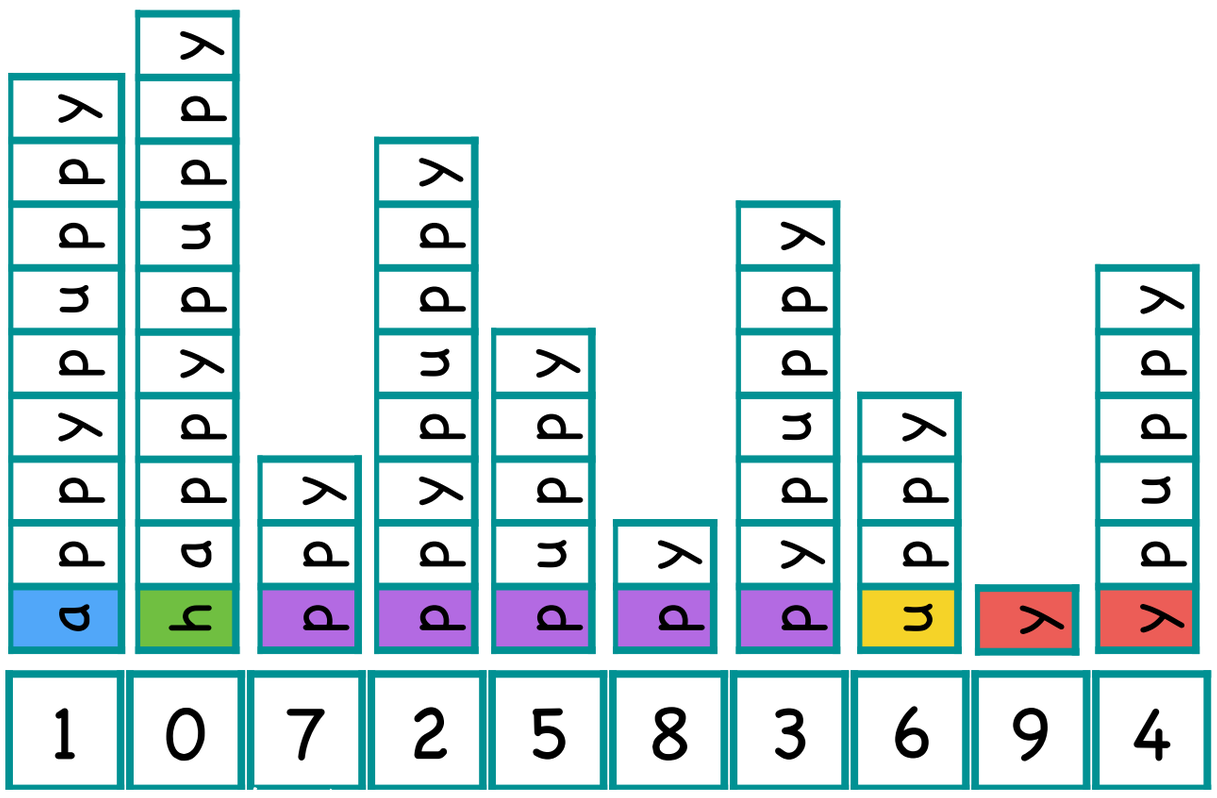
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



0

Succinct Data Representation

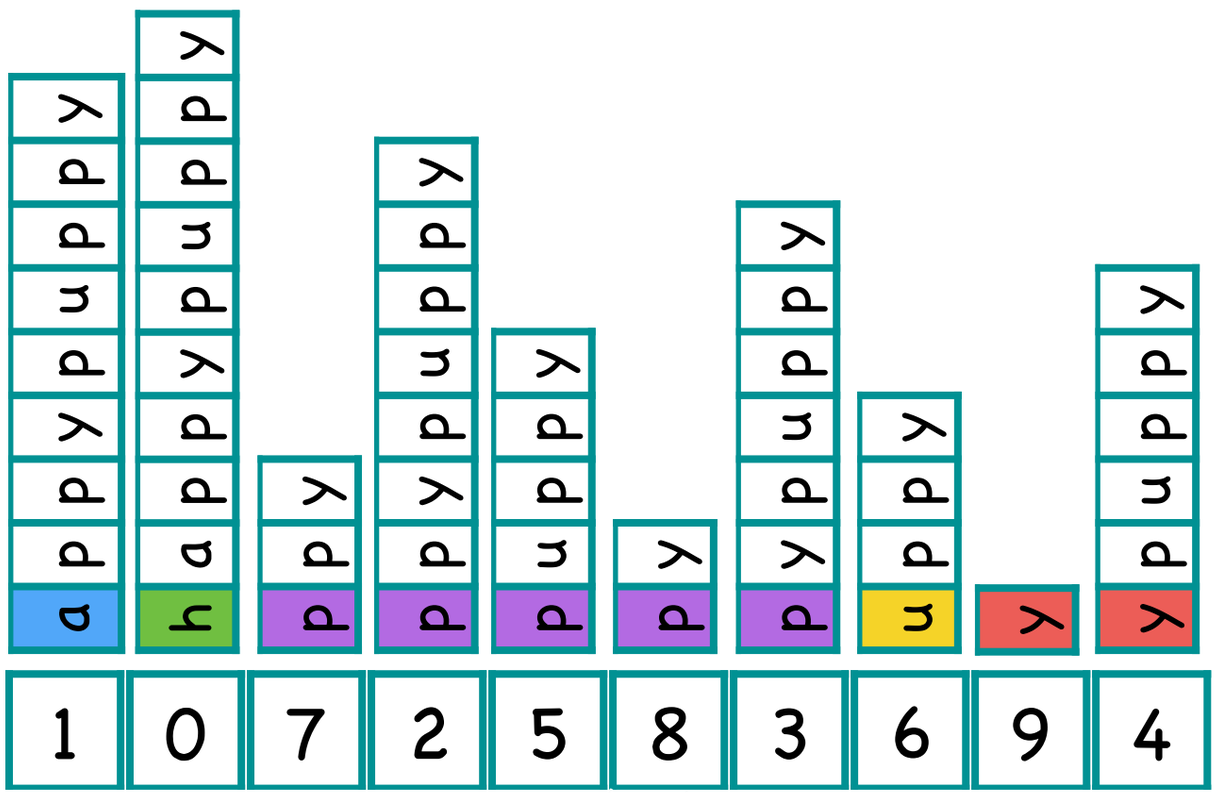
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



3	0

Succinct Data Representation

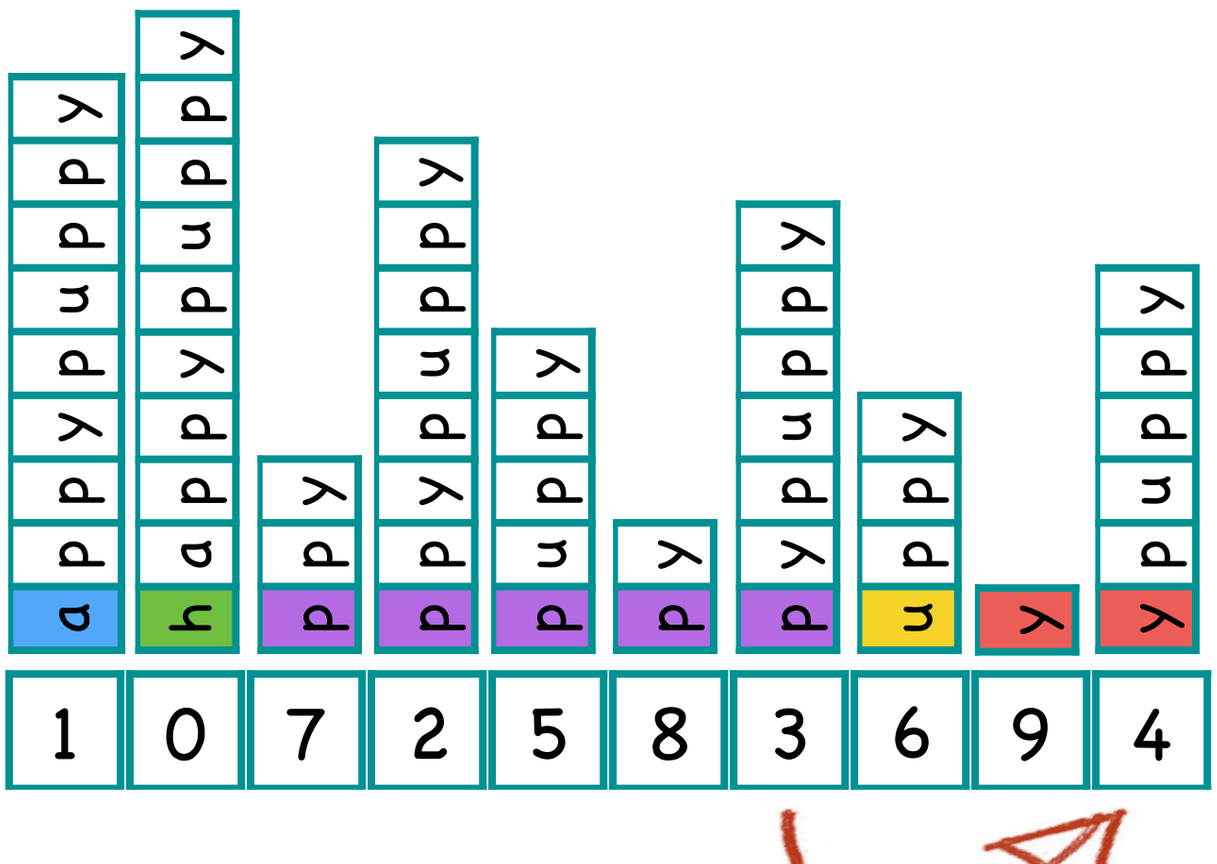
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



3	0		6

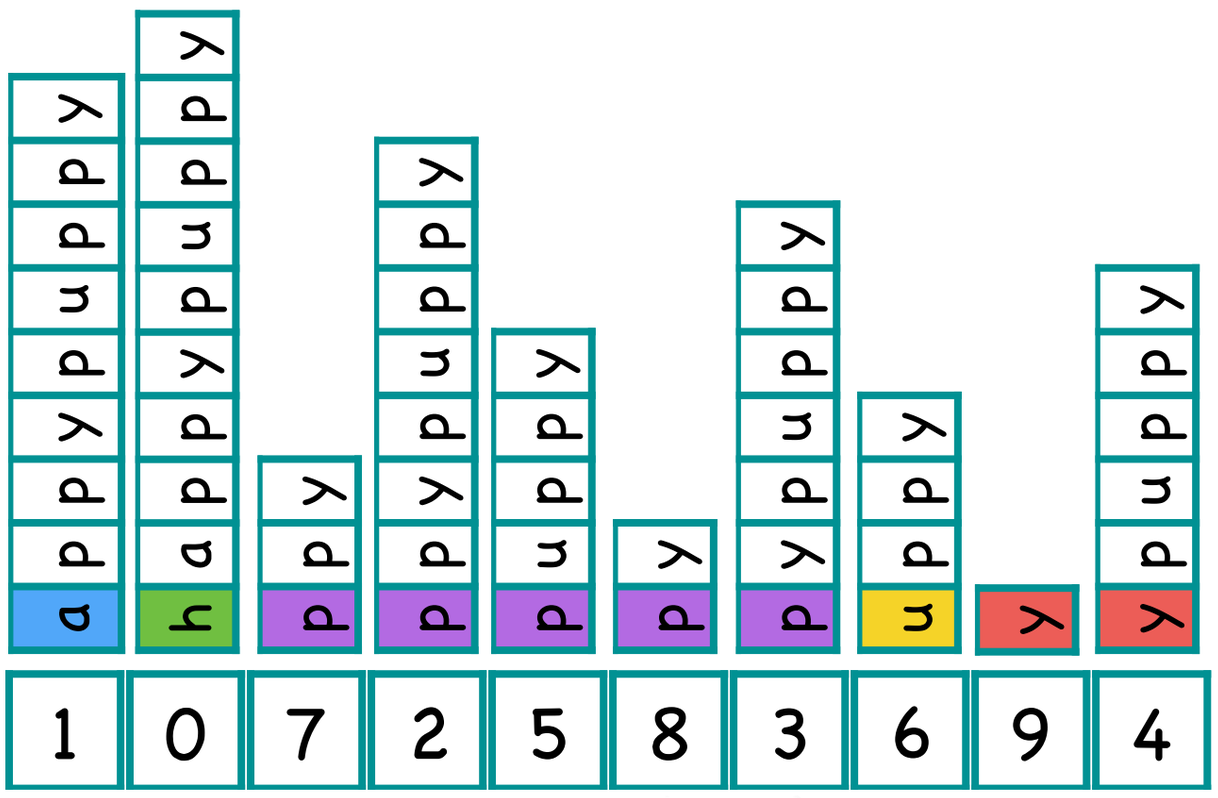
Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



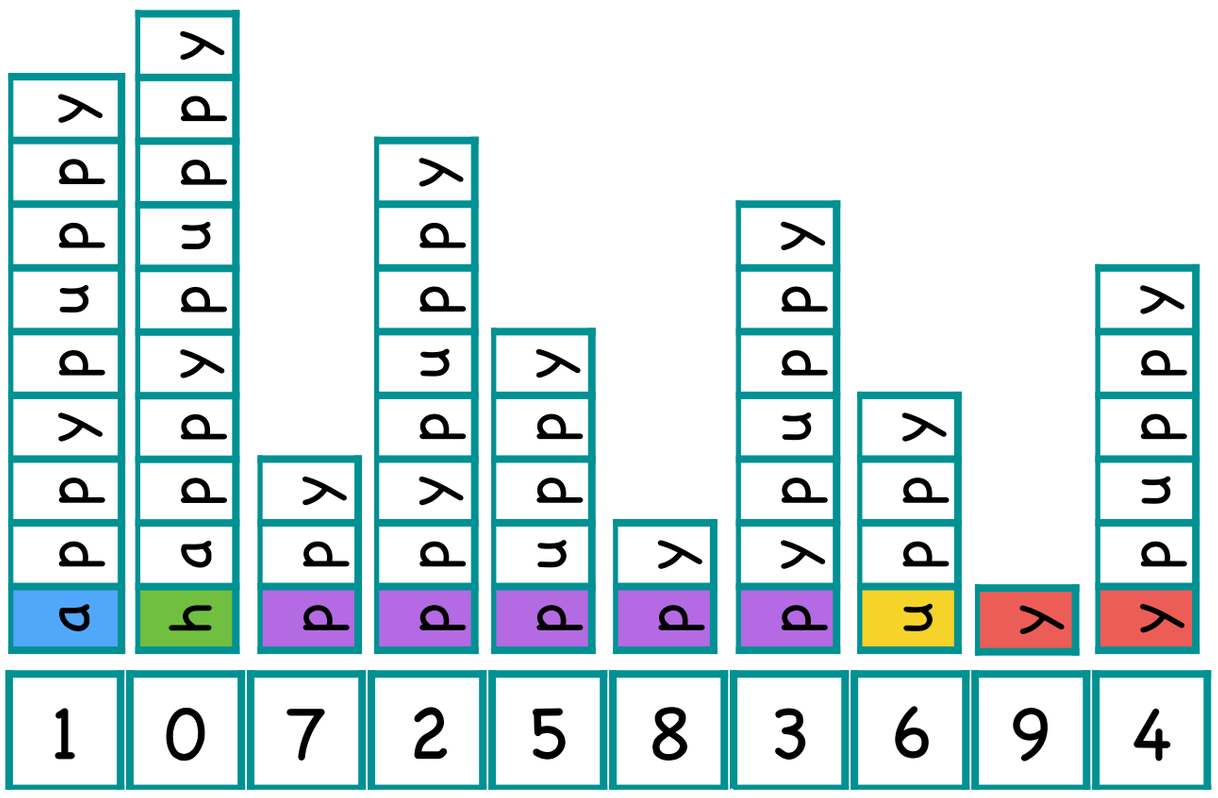
Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



Succinct Data Representation

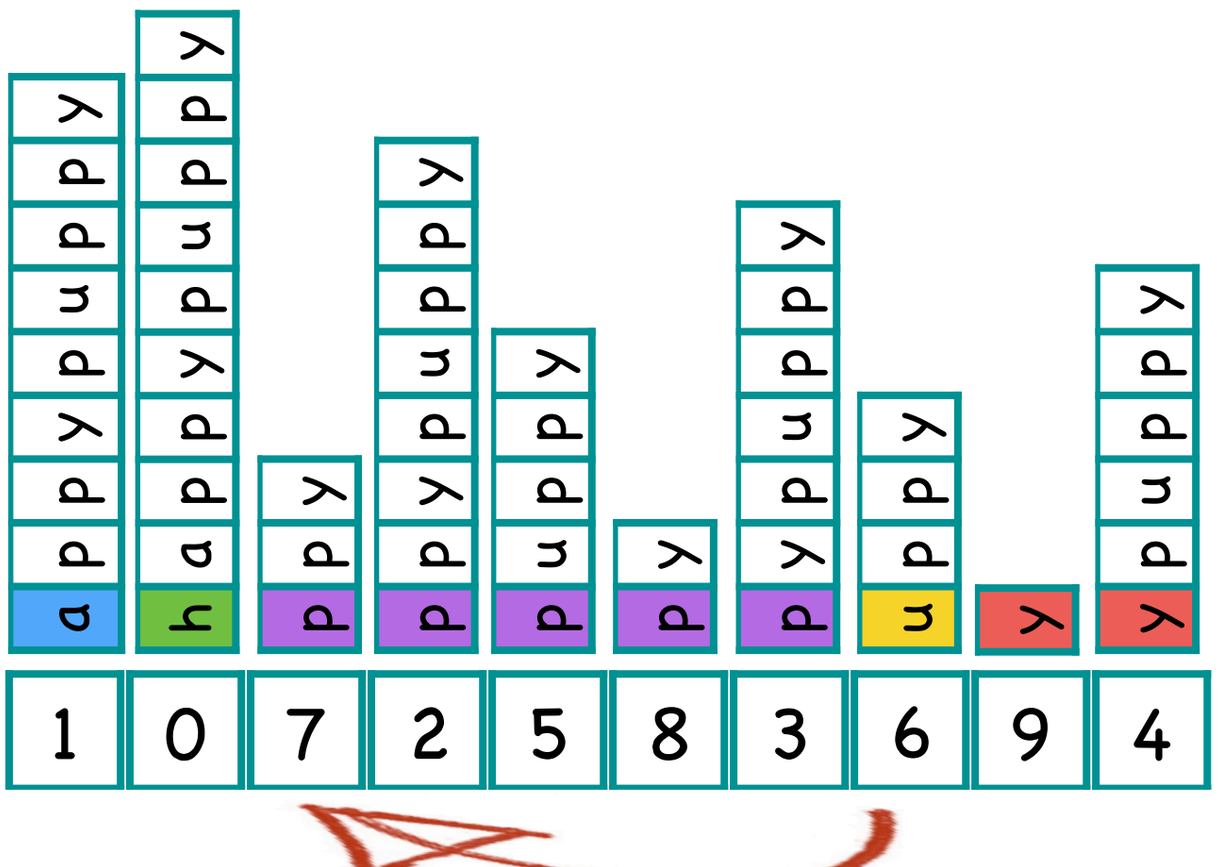
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



3	0		6	7		9			4

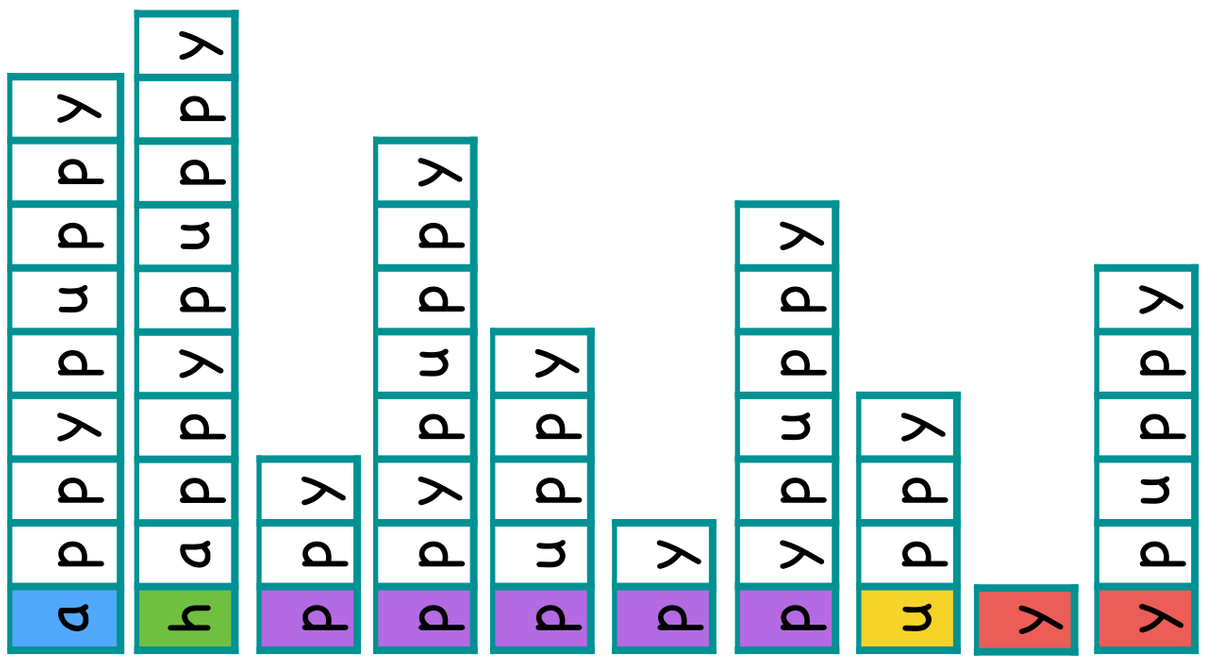
Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

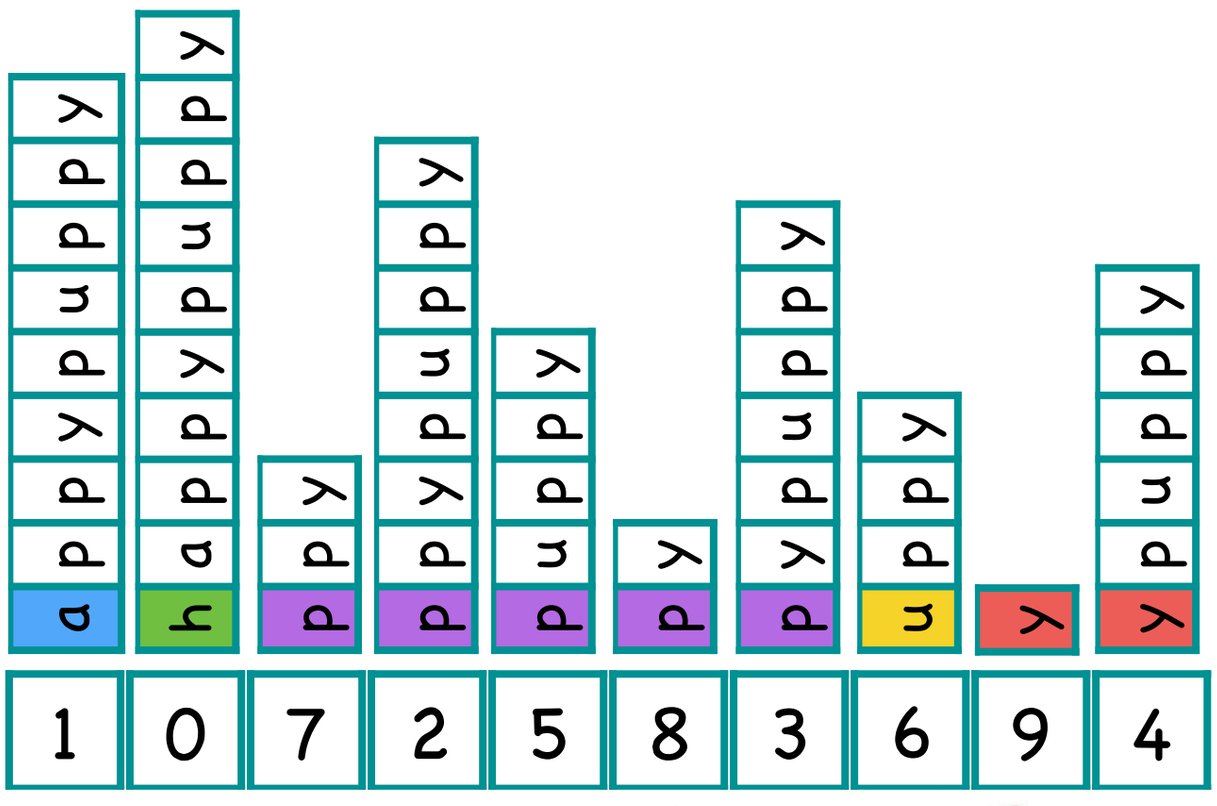


1	0	7	2	5	8	3	6	9	4

3	0	5	6	7	9	2	4

Succinct Data Representation

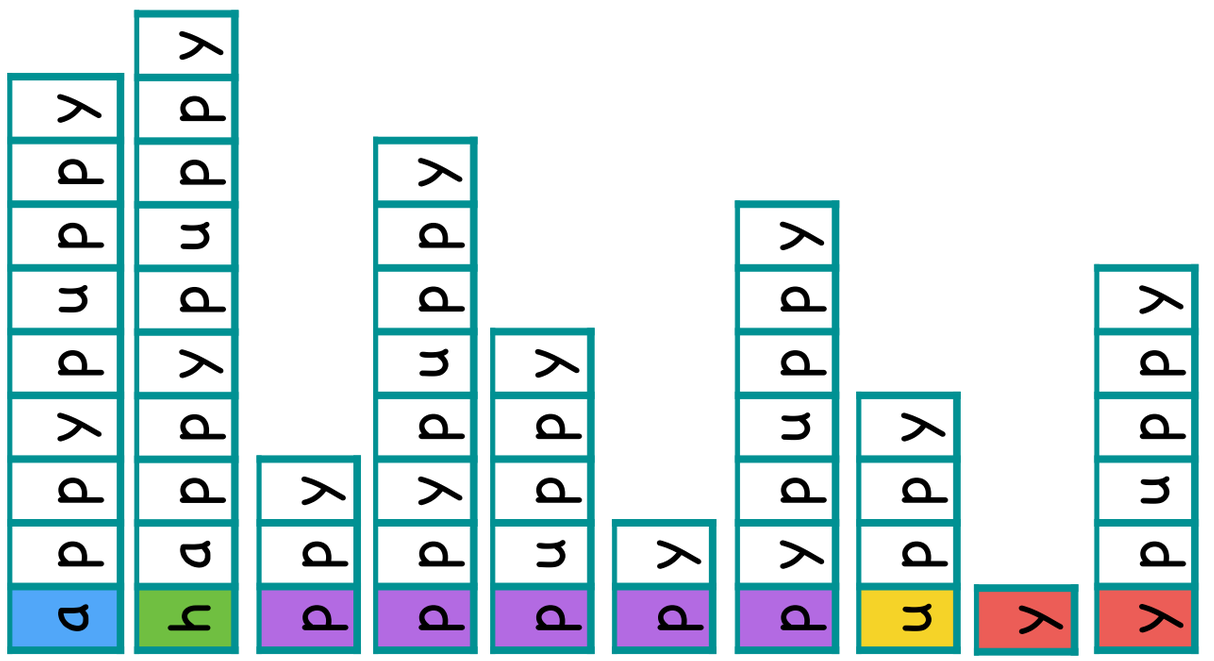
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



3	0	5	6	7	8	9	2		4

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

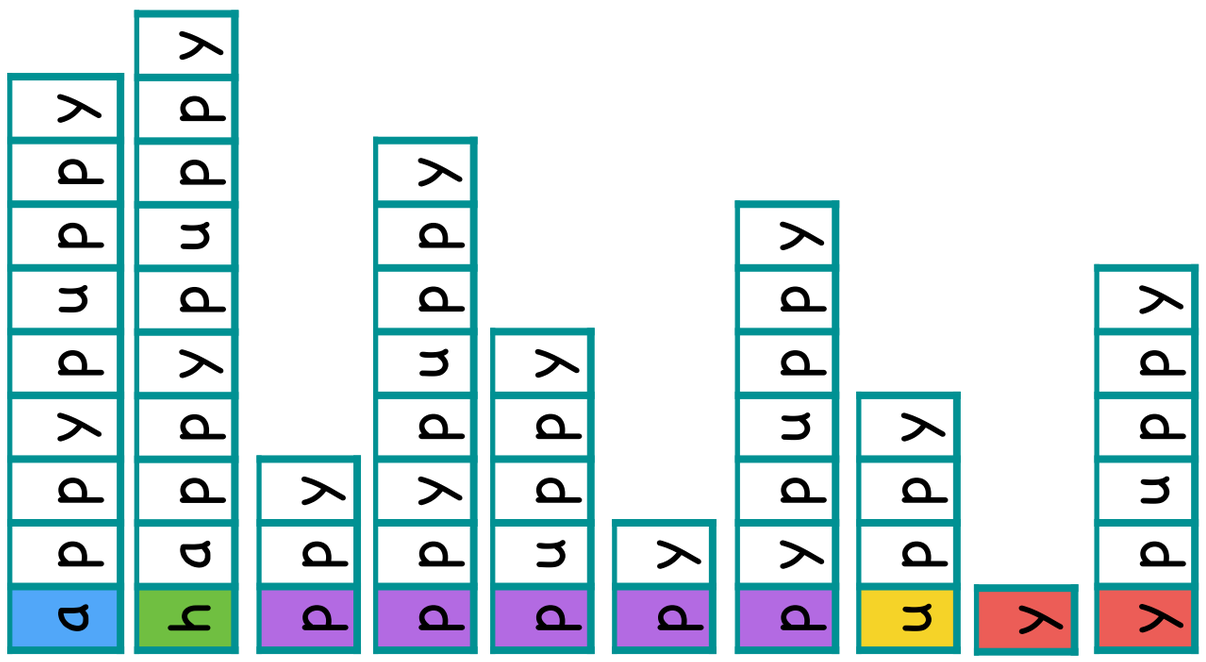


1	0	7	2	5	8	3	6	9	4

3	0	5	6	7	8	9	2	1	4

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

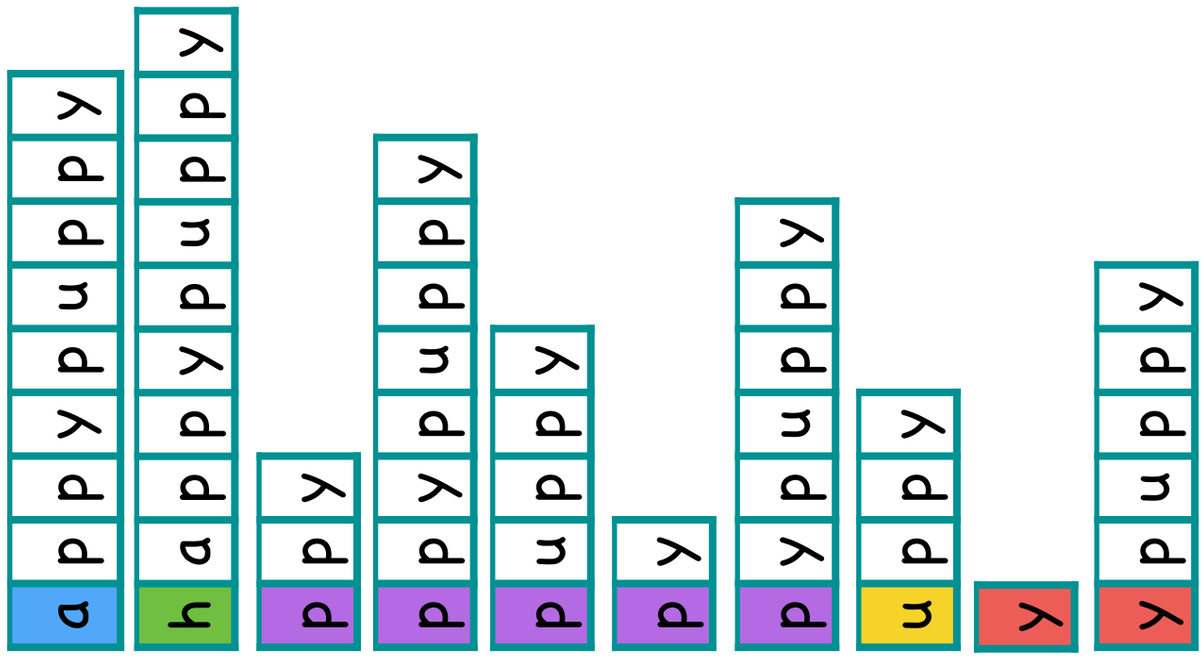


1	0	7	2	5	8	3	6	9	4

3	0	5	6	7	8	9	2	1	4

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



1	0	7	2	5	8	3	6	9	4

NextCharIdx

(index of suffix after removing the first character)

3	0	5	6	7	8	9	2	1	4

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

h	a	p	p	y	p	u	p	p	y
1	0	7	2	5	8	3	6	9	4

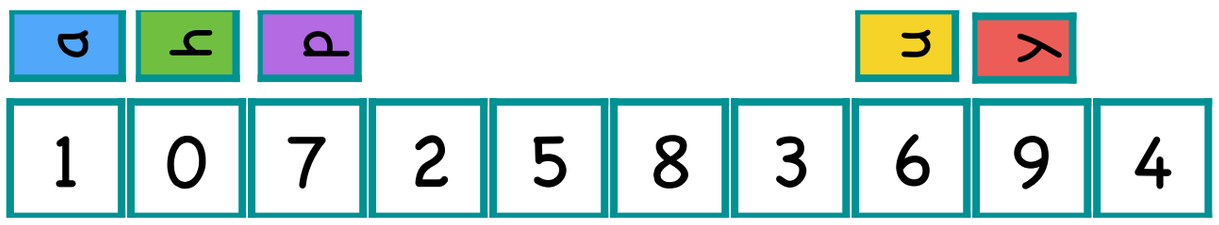
3	0	5	6	7	8	9	2	1	4

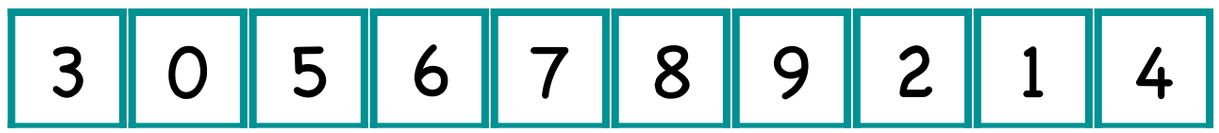
NextCharIdx

(index of suffix after removing the first character)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y





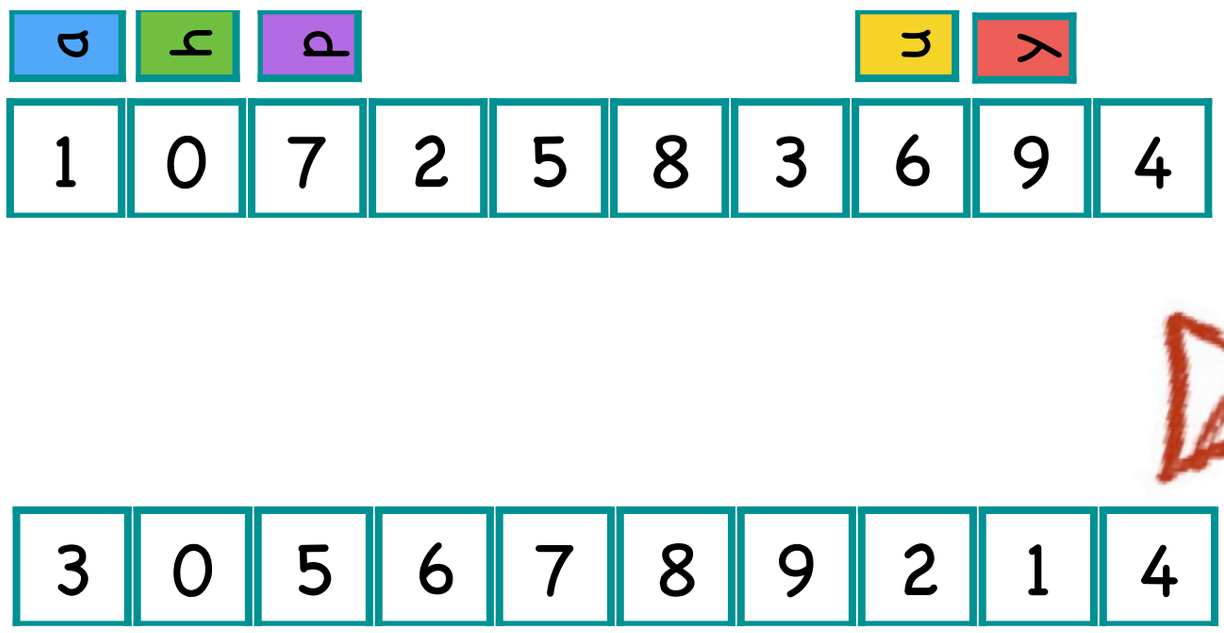
NextCharIdx

(index of suffix after removing the first character)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

No redundancy in AoS2Input

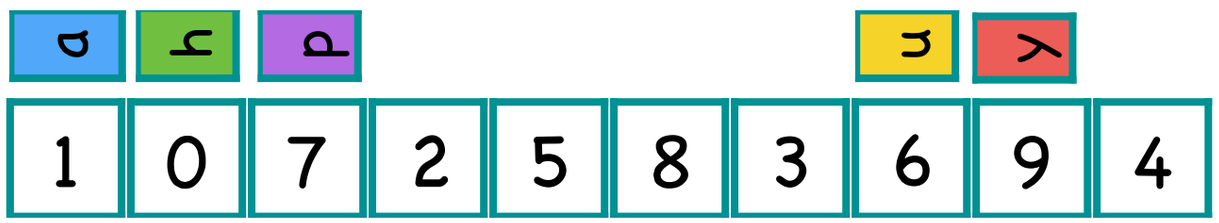


NextCharIdx

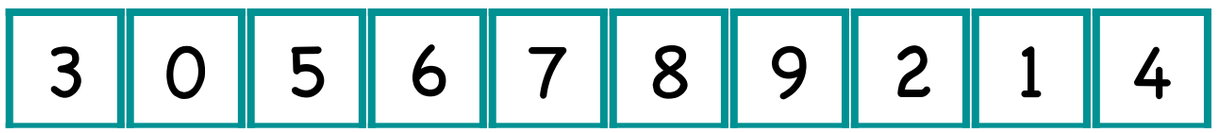
(index of suffix after removing the first character)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



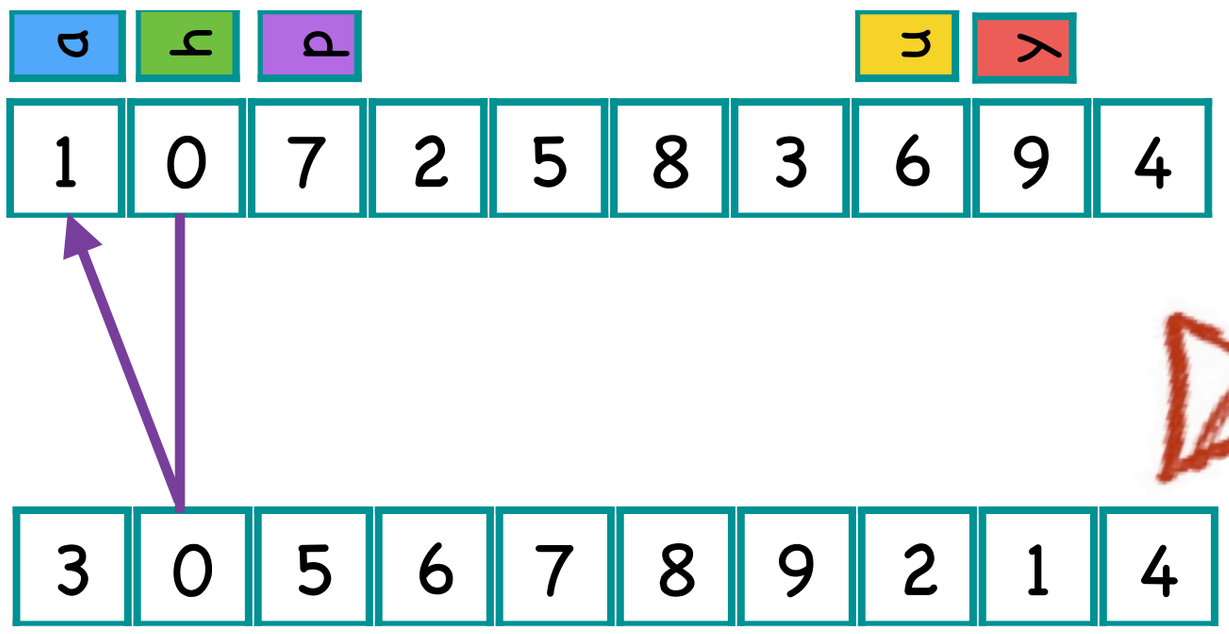
NextCharIdx
(index of next larger value in AoS2Input)



NextCharIdx
(index of suffix after removing the first character)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

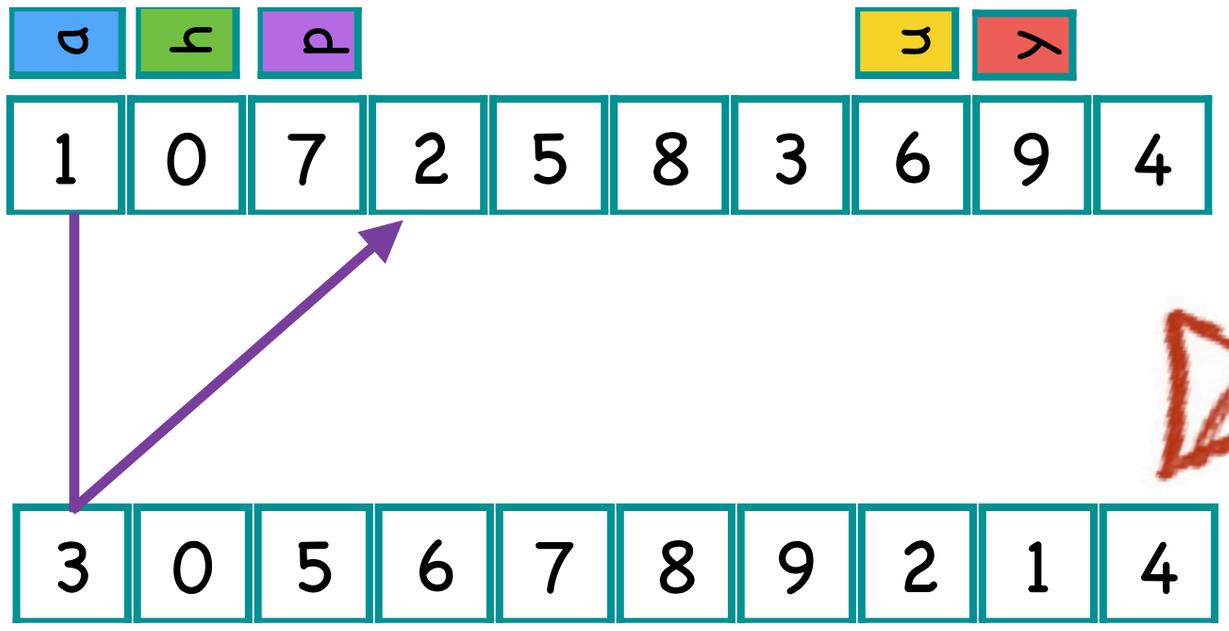


NextCharIdx
(index of next larger value in AoS2Input)

NextCharIdx
(index of suffix after removing the first character)

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y



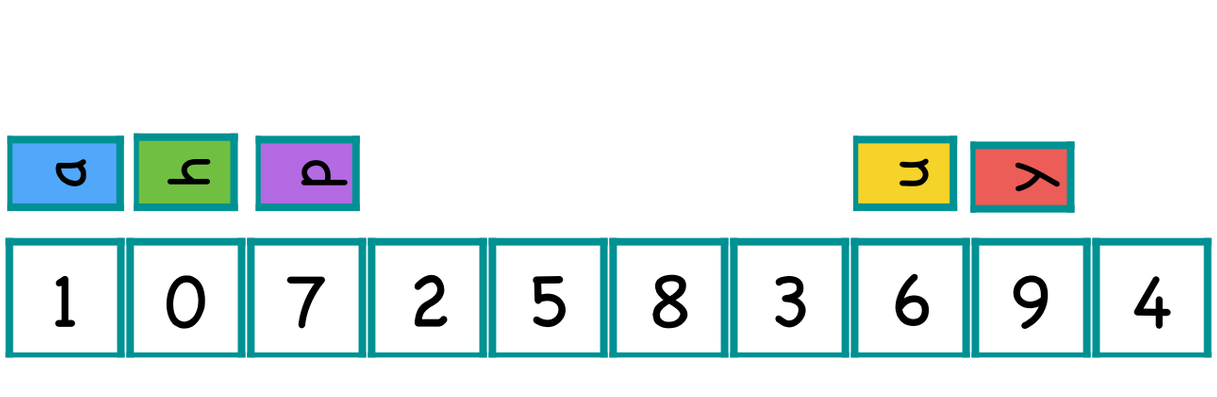
NextCharIdx
(index of next larger value in AoS2Input)

NextCharIdx
(index of suffix after removing the first character)

Succinct Data Representation

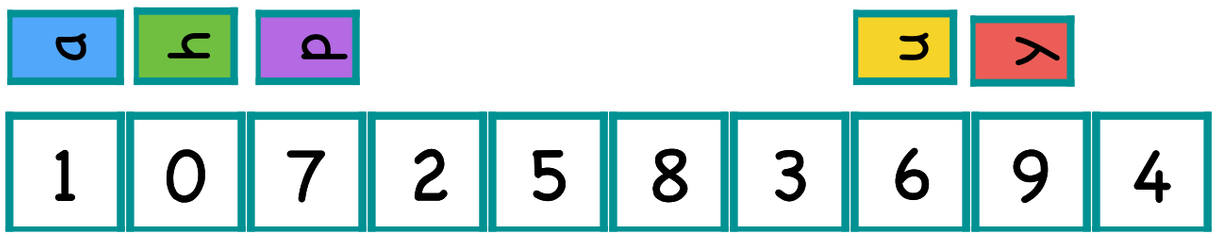
0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

No redundancy in AoS2Input



Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

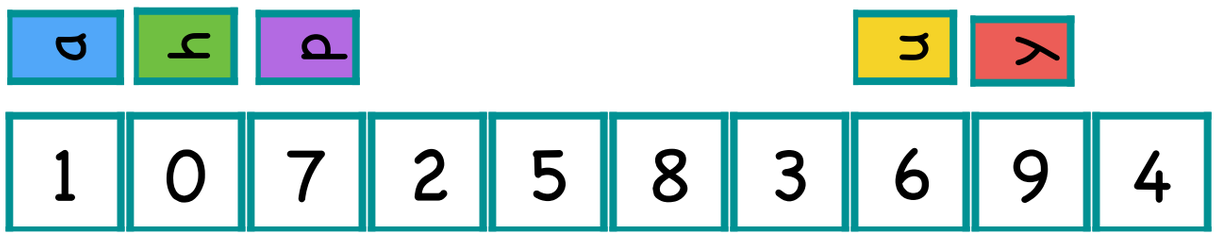


Solution: Sample!

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

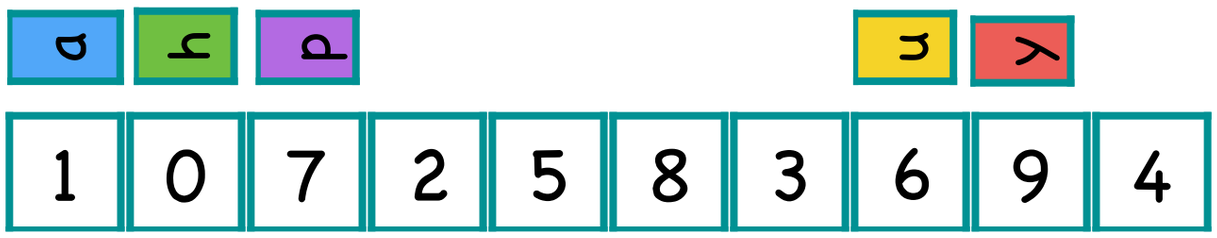
1		7		5		3		9	



Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	



Problem: How to find unsampled values?

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	

a	h	p	y	x

1	0	7	2	5	8	3	6	9	4

Problem: How to find unsampled values?

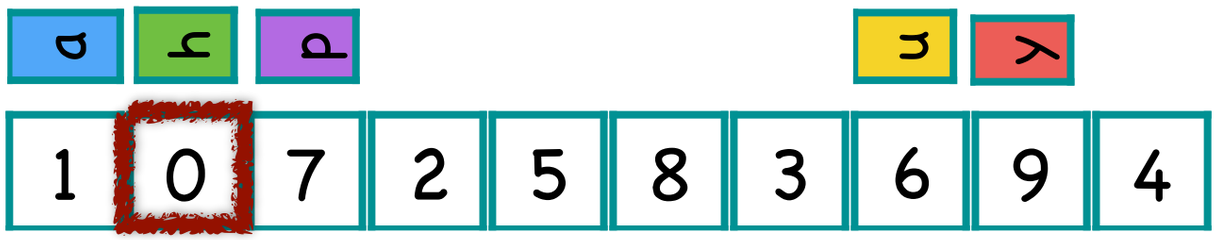
3	0	5	6	7	8	9	2	1	4

Solution:Follow NextCharIdx pointers until you hit a sampled value

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	



Problem: How to find unsampled values?

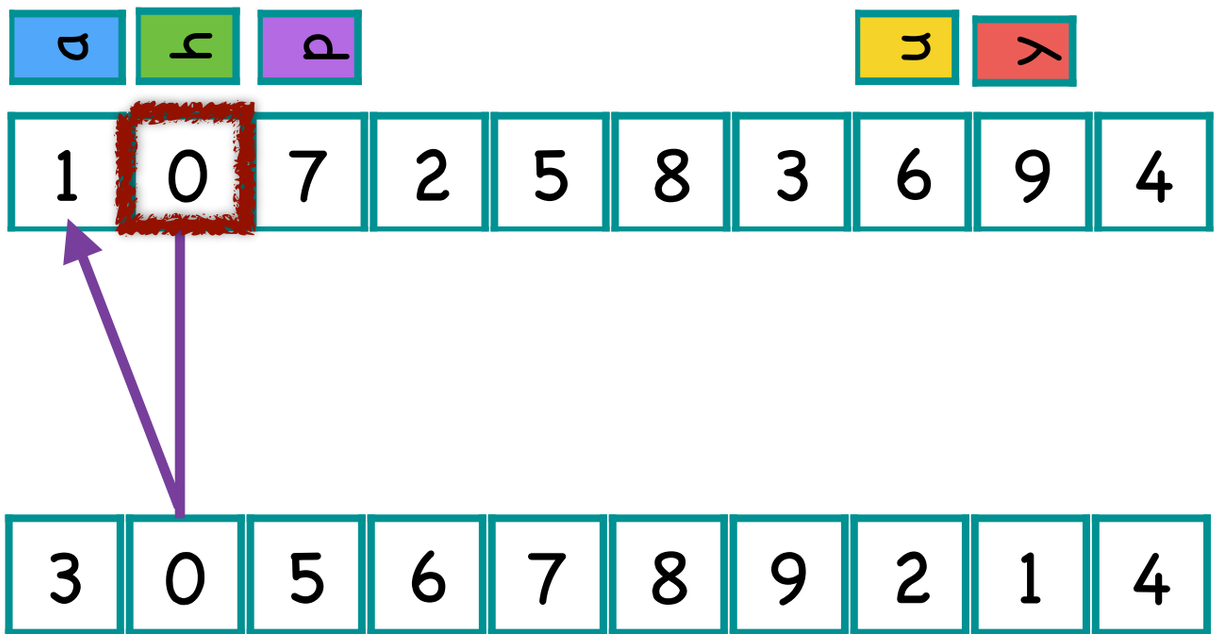
3	0	5	6	7	8	9	2	1	4

Solution:Follow NextCharIdx pointers until you hit a sampled value

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	



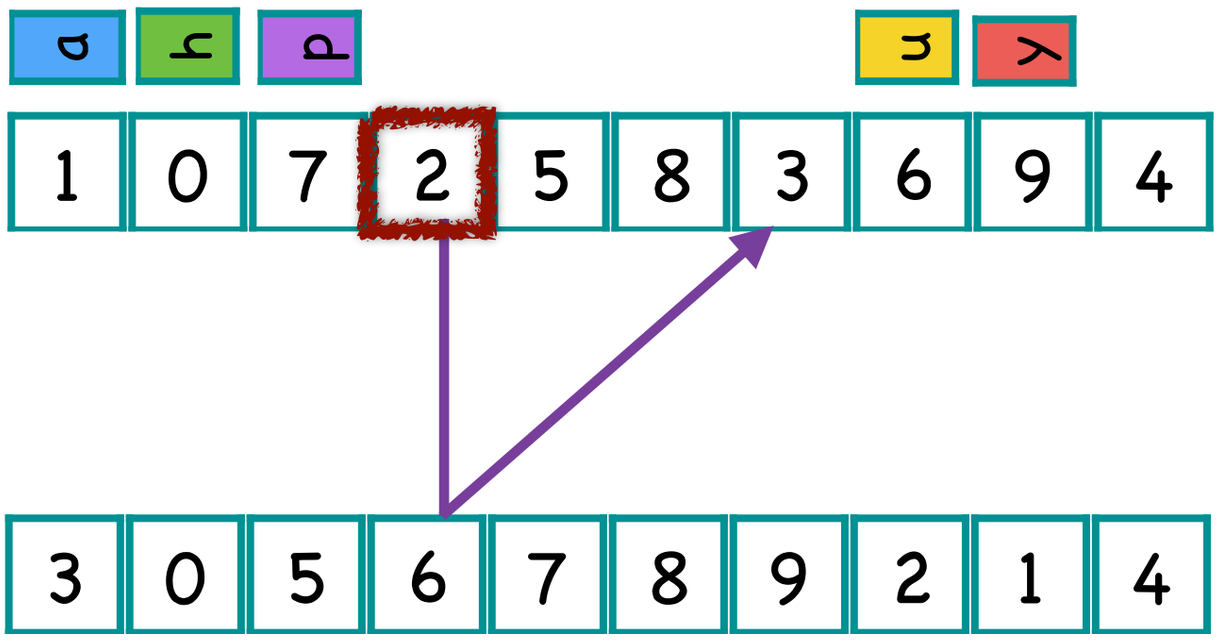
Problem: How to find unsampled values?

Solution:Follow NextCharIdx pointers until you hit a sampled value

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	



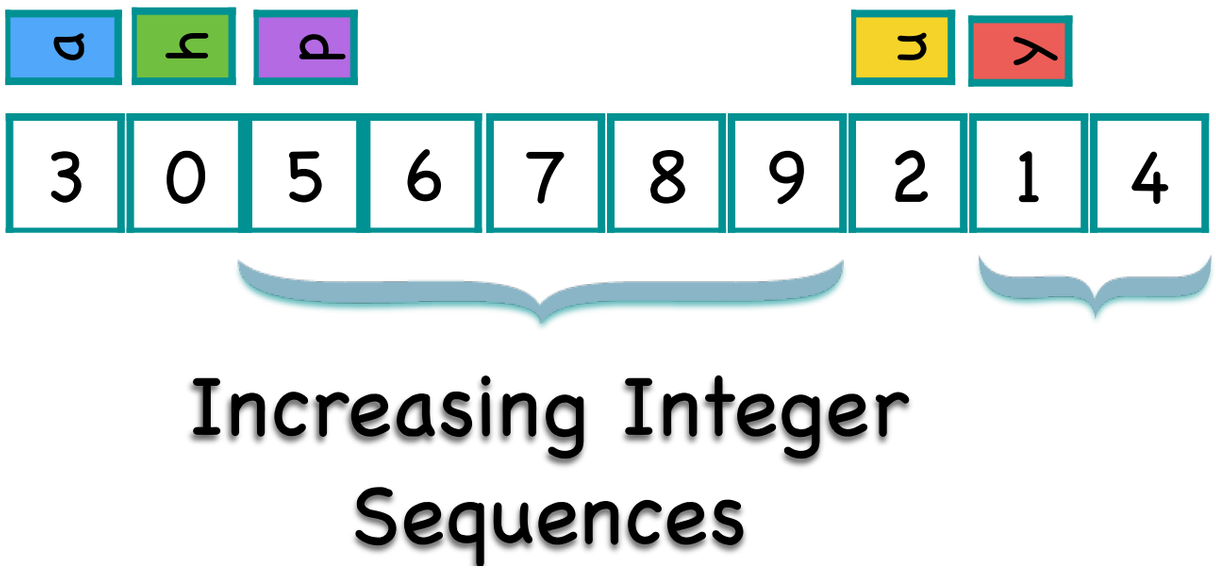
Problem: How to find unsampled values?

Solution:Follow NextCharIdx pointers until you hit a sampled value

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	

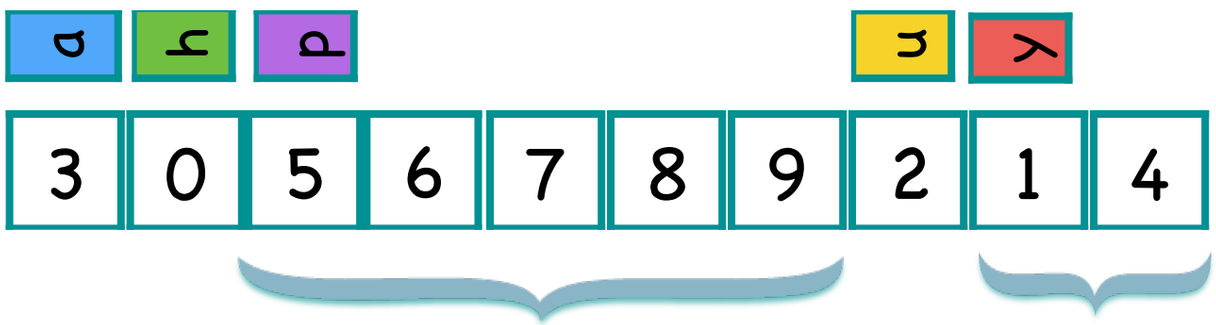


Increasing Integer Sequences

Succinct Data Representation

0	1	2	3	4	5	6	7	8	9
h	a	p	p	y	p	u	p	p	y

1		7		5		3		9	



Increasing Integer Sequences

Can be compressed!

Data Model and API

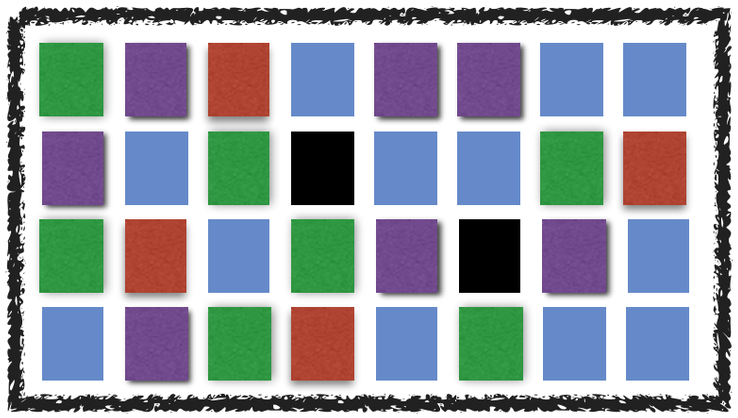
Data Model and API

Input: flat (unstructured) files

Data Model and API

Input: flat (unstructured) files

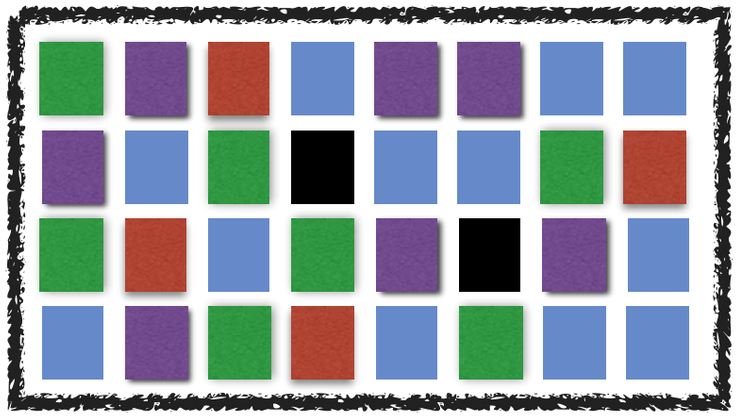
Original Input



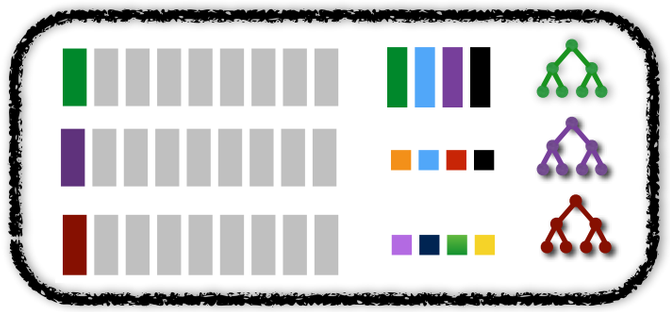
Data Model and API

Input: flat (unstructured) files

Original Input



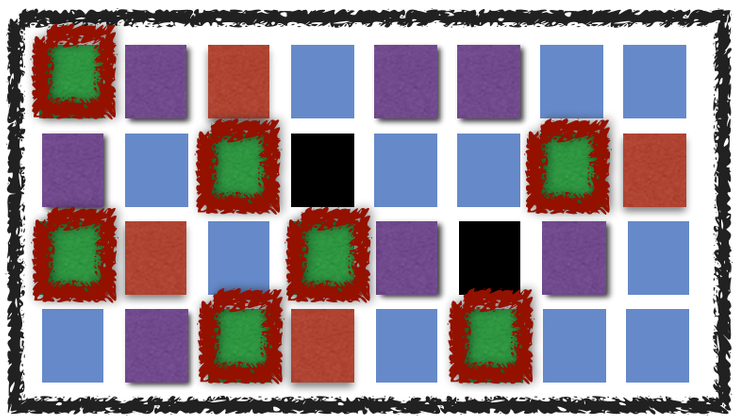
Succinct



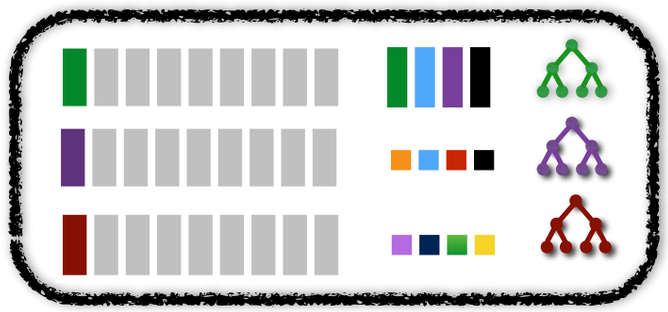
Data Model and API

Input: flat (unstructured) files

Original Input



Succinct



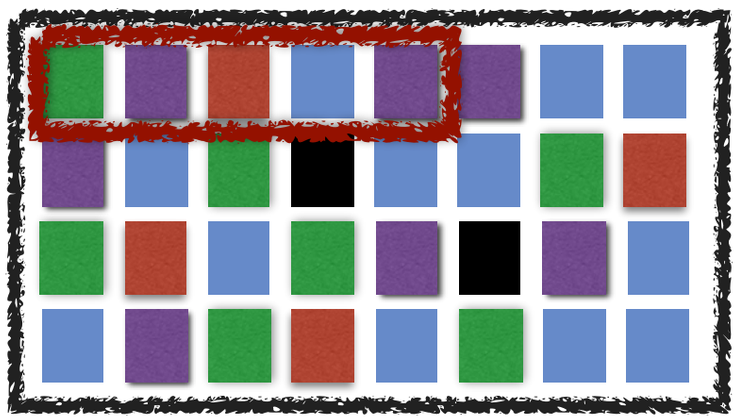
Search( ) = {0, 10, 14, 16, 19, 26, 29}

Search: returns offsets of arbitrary strings in uncompressed file

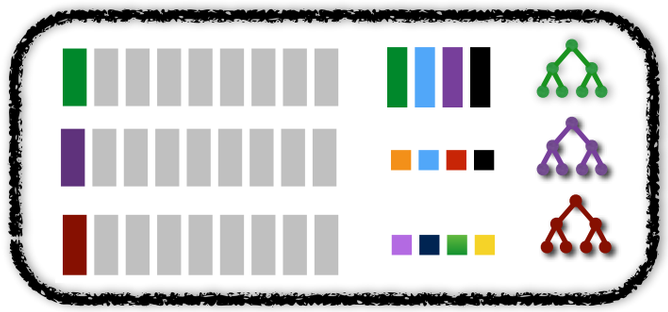
Data Model and API

Input: flat (unstructured) files

Original Input



Succinct



Search(■) = {0, 10, 14, 16, 19, 26, 29}

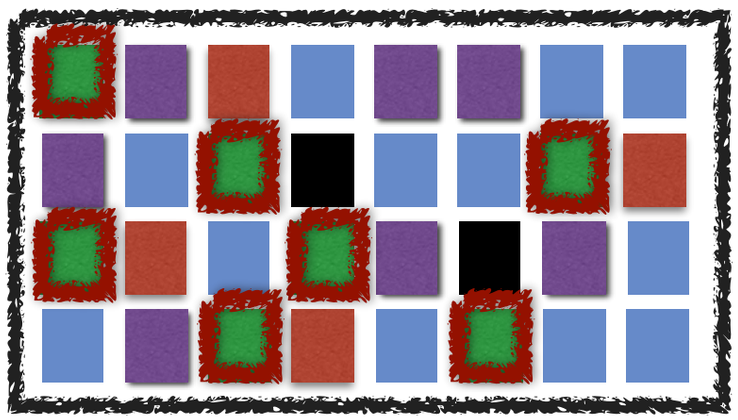
Extract(0, 5) = { ■ , ■ , ■ , ■ , ■ }

Extract: returns data at arbitrary offsets in uncompressed file

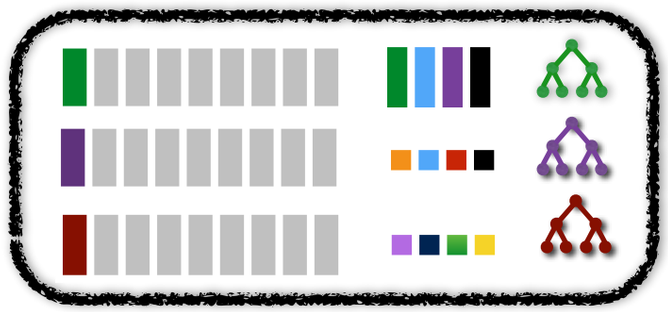
Data Model and API

Input: flat (unstructured) files

Original Input



Succinct



Search(■) = {0, 10, 14, 16, 19, 26, 29}

Extract(0, 5) = { ■ , ■ , ■ , ■ , ■ }

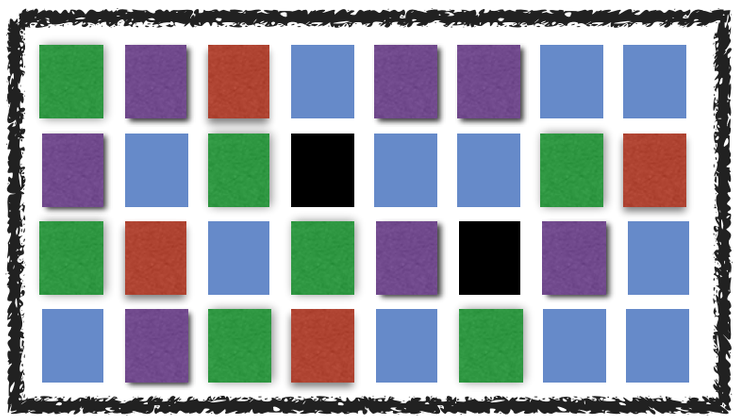
Count(■) = 7

Count: returns count of arbitrary strings in uncompressed file

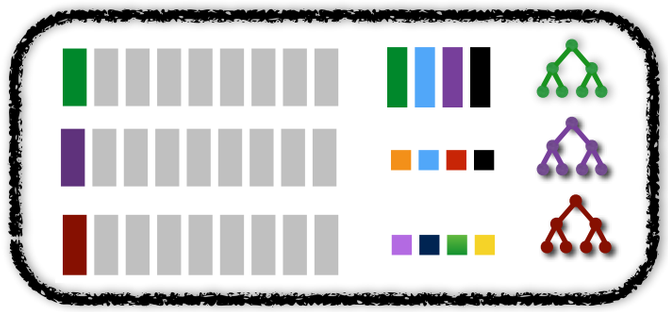
Data Model and API

Input: flat (unstructured) files

Original Input



Succinct



Search( ) = {0, 10, 14, 16, 19, 26, 29}

Extract(0, 5) = { , , , , }

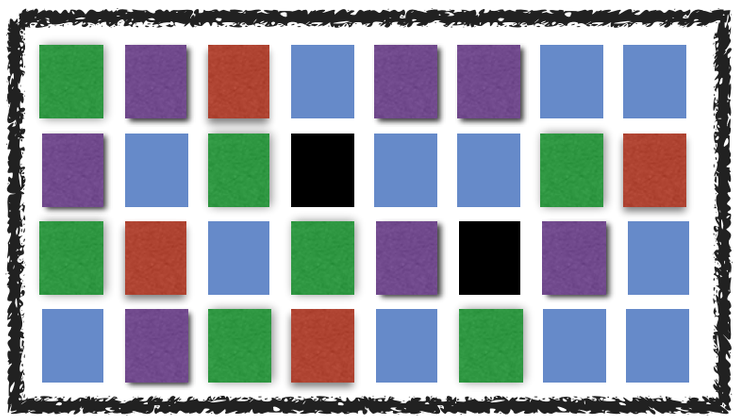
Count( ) = 7

Append(, , , ,)

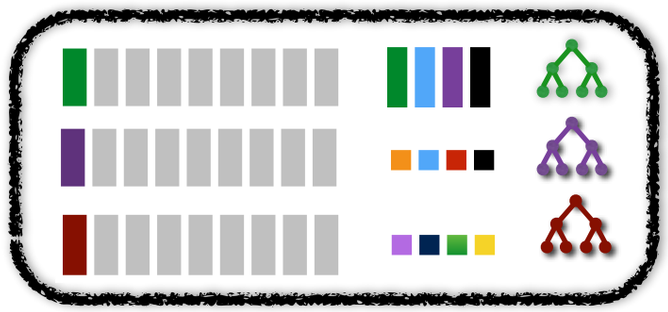
Data Model and API

Input: flat (unstructured) files

Original Input



Succinct



Search(■) = {0, 10, 14, 16, 19, 26, 29}

Extract(0, 5) = { ■ , ■ , ■ , ■ , ■ }

Count(■) = 7

Append(■ , ■ , ■ , ■ , ■)

Range and Wildcard queries

Power of queries on flat files

Why flat unstructured files?

Power of queries on flat files

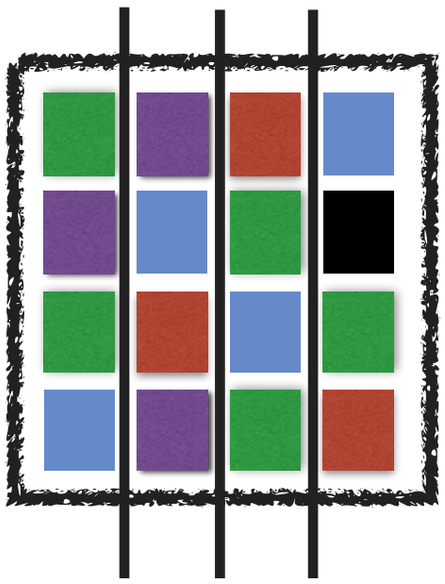
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]

Power of queries on flat files

Why flat unstructured files?

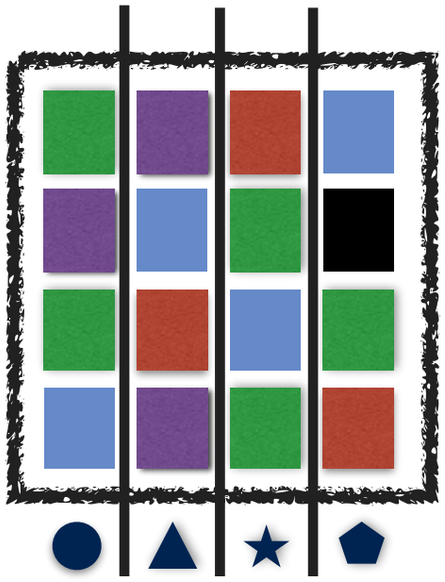
- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]



Power of queries on flat files

Why flat unstructured files?

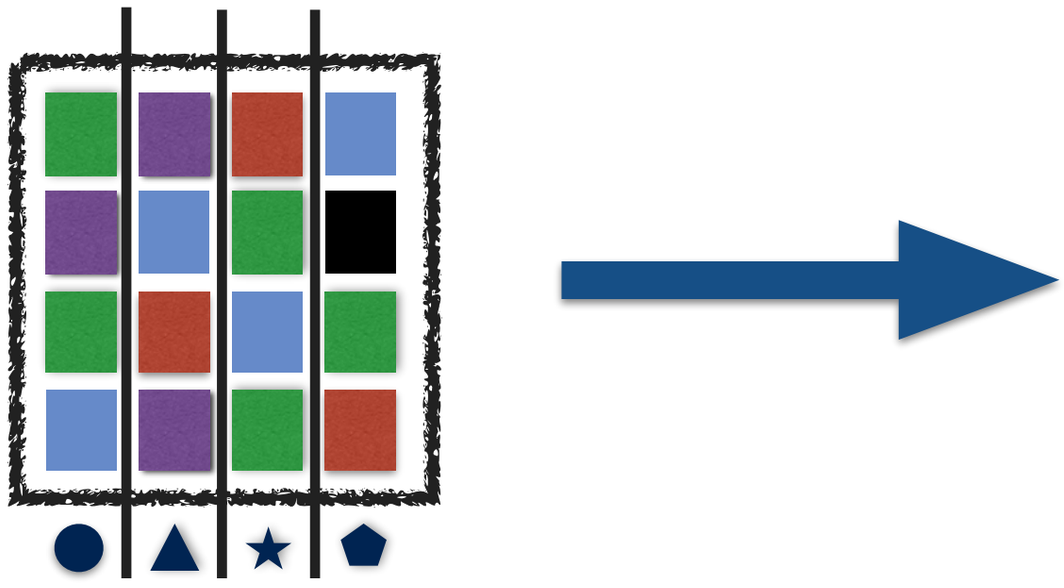
- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]



Power of queries on flat files

Why flat unstructured files?

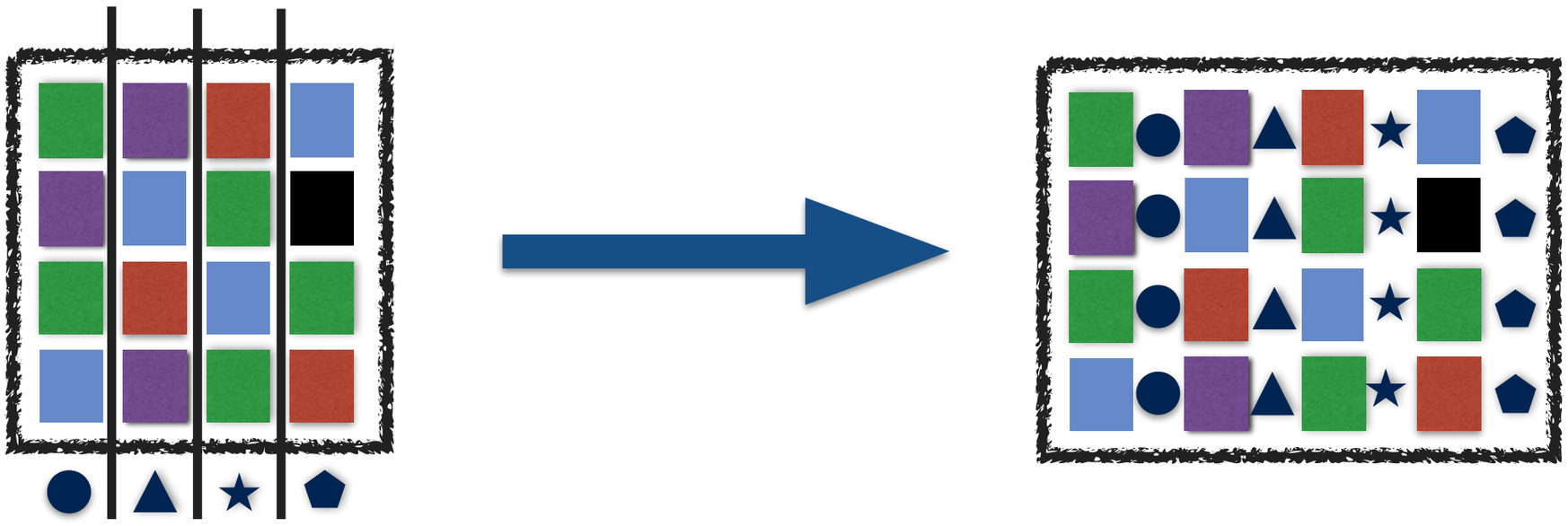
- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]



Power of queries on flat files

Why flat unstructured files?

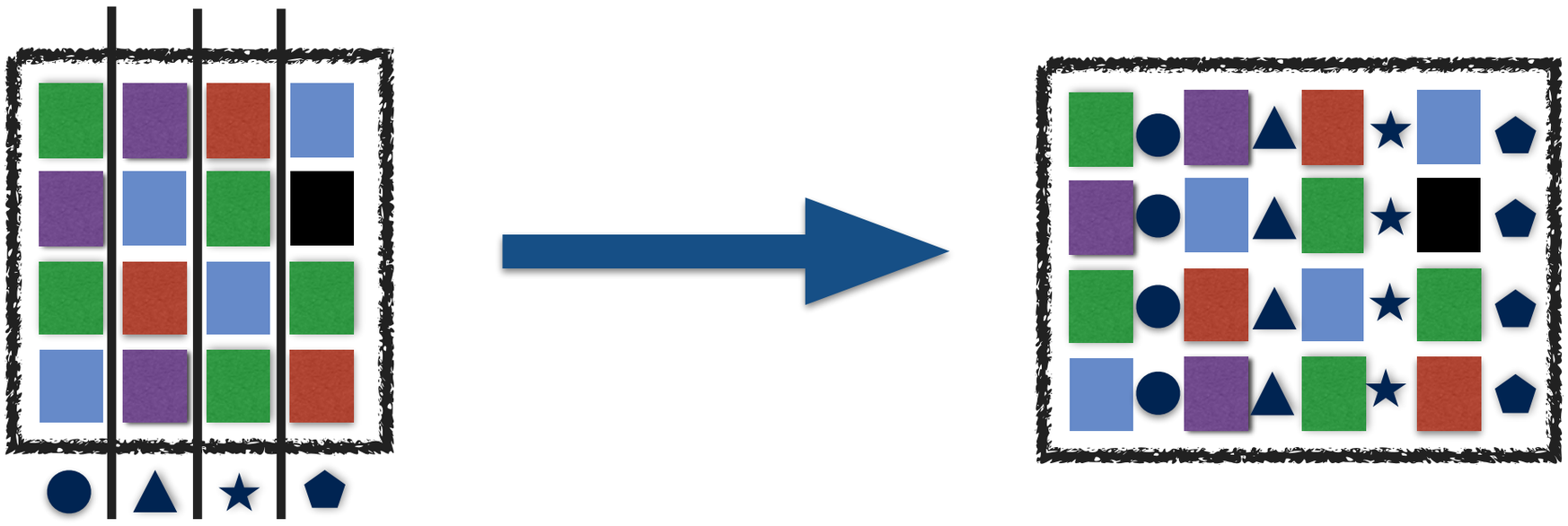
- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]



Power of queries on flat files

Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]



Search(Column1, ■)

Search(■●)

Handling Appends

Fine-grained updates of compressed data challenging

- Succinct does not support in-place updates
- Uses a multi-store architecture for appends
- Similar to SILT [SOSP'11]

Handling Appends

Fine-grained updates of compressed data challenging

- Succinct does not support in-place updates
- Uses a multi-store architecture for appends
- Similar to SILT [SOSP'11]

Challenge

Handling Appends

Fine-grained updates of compressed data challenging

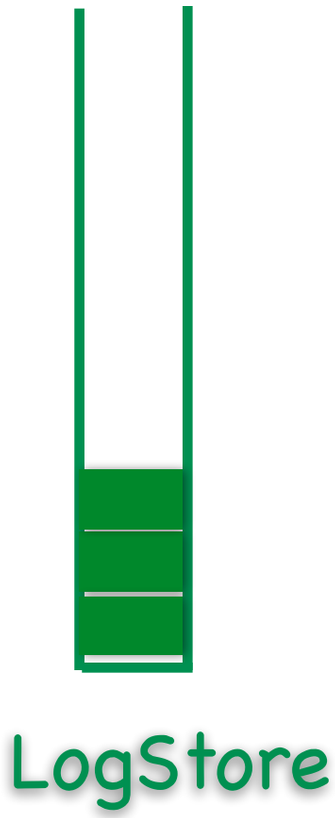
- Succinct does not support in-place updates
- Uses a multi-store architecture for appends
- Similar to SILT [SOSP'11]

Challenge

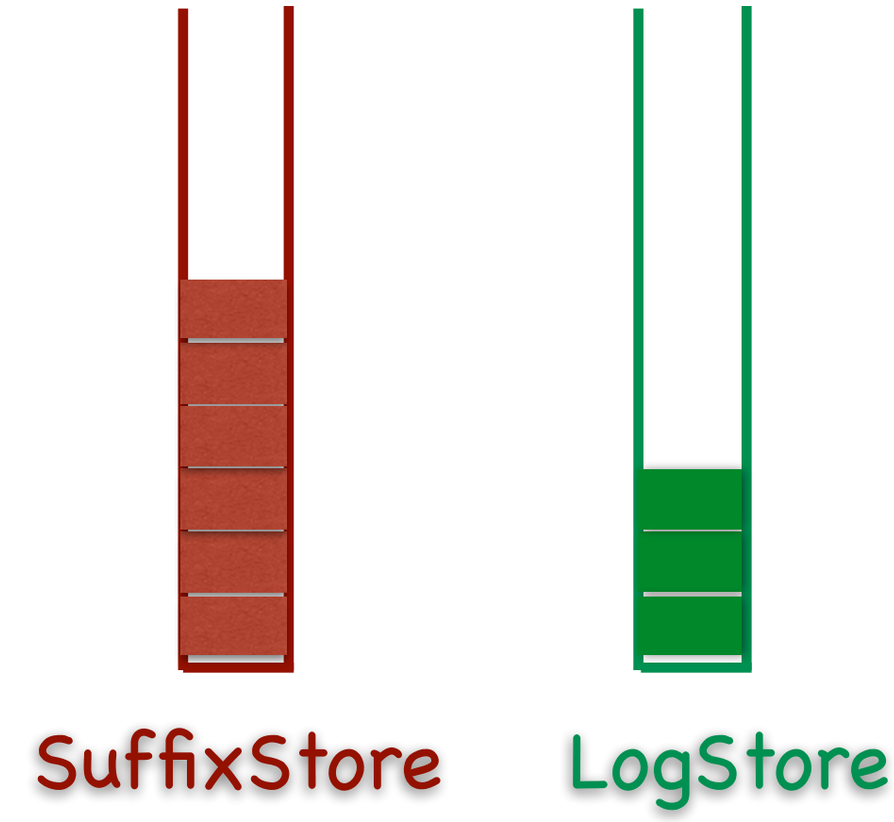
- Multi-store architectures not new
- Challenge: supporting queries efficiently
- Details in paper

End-to-end System

End-to-end System



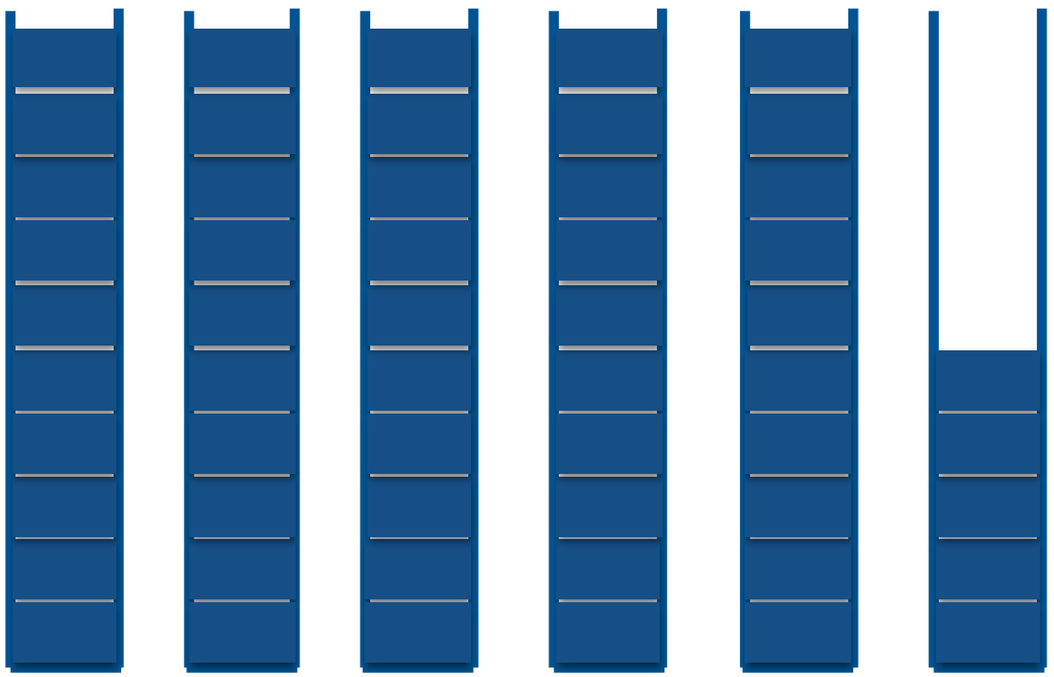
End-to-end System



End-to-end System

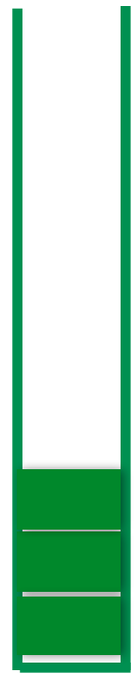
End-to-end System

Coordinator



SuccinctStores

SuffixStore



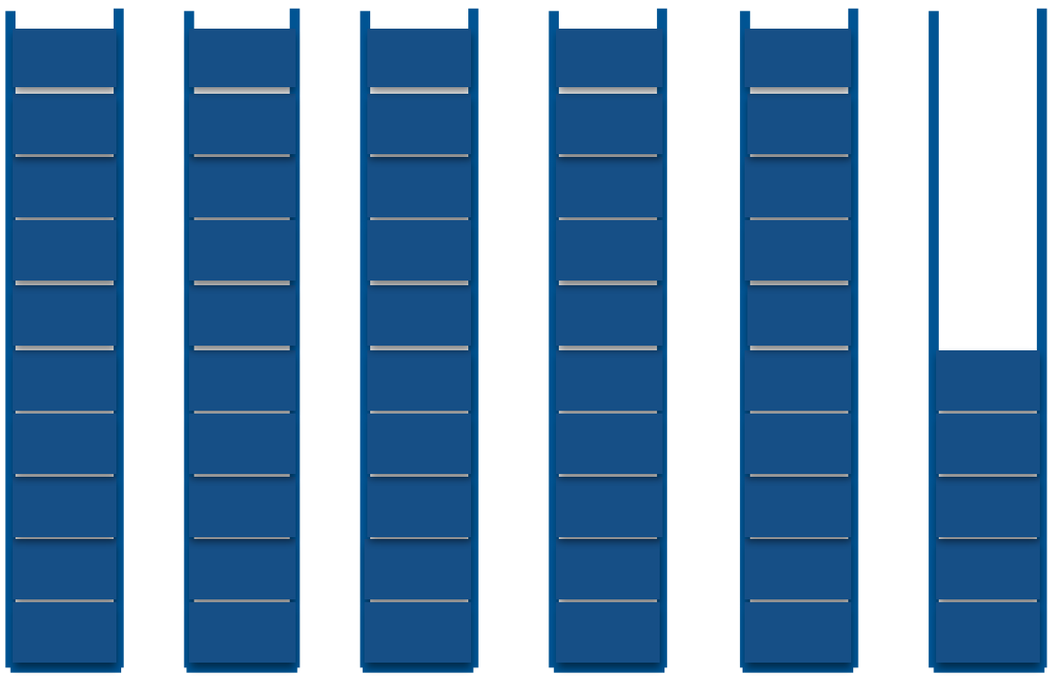
LogStore

End-to-end System

Coordinator

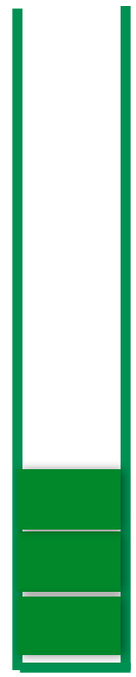
Cluster Management

- Maintains lists of active servers
- Periodic heartbeats



SuccinctStores

SuffixStore



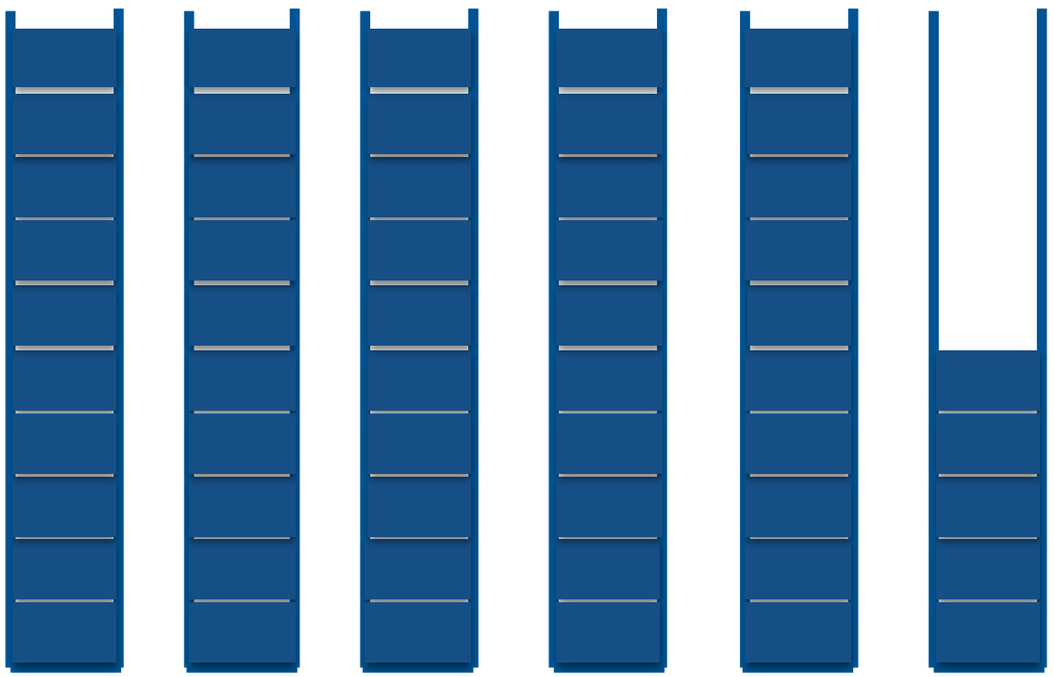
LogStore

End-to-end System

Coordinator

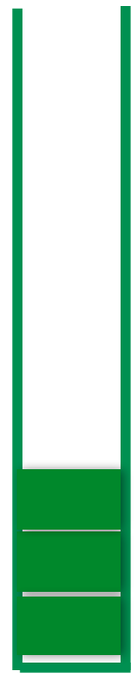
Data Management

- Updated pointers to locate data
- Pushes these pointers to servers



SuccinctStores

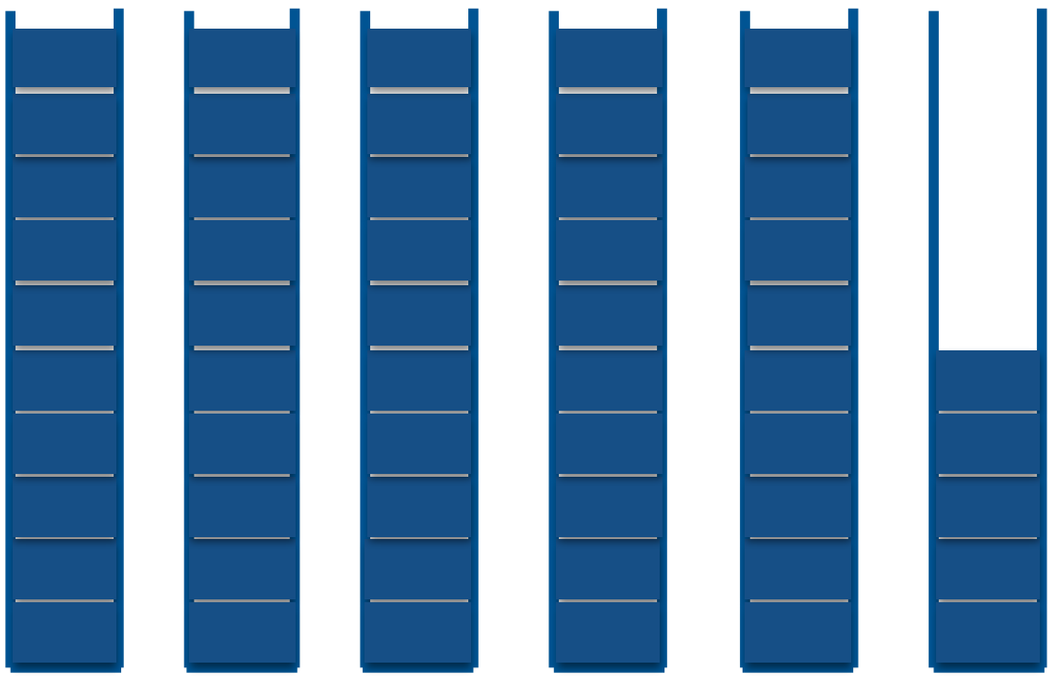
SuffixStore



LogStore

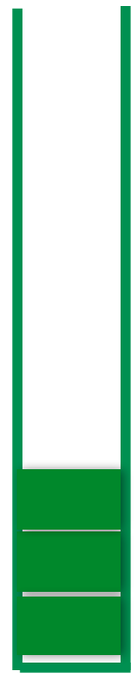
End-to-end System

Coordinator



SuccinctStores

SuffixStore

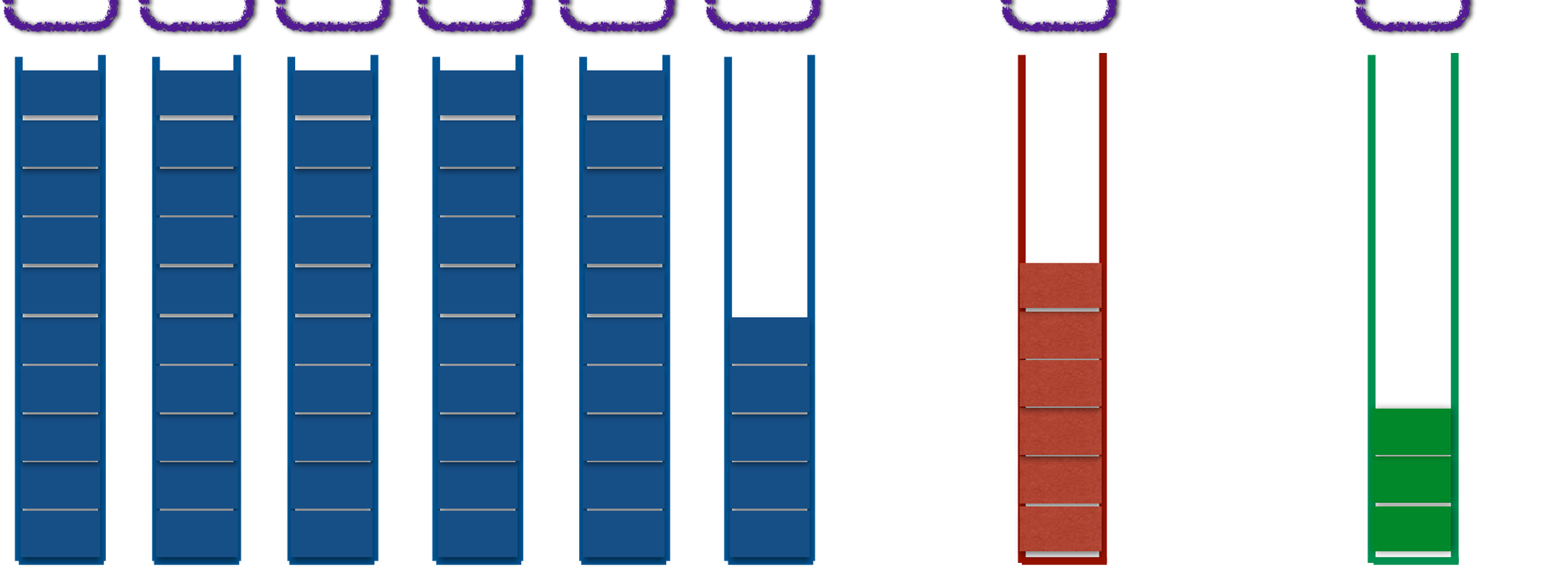


LogStore

End-to-end System

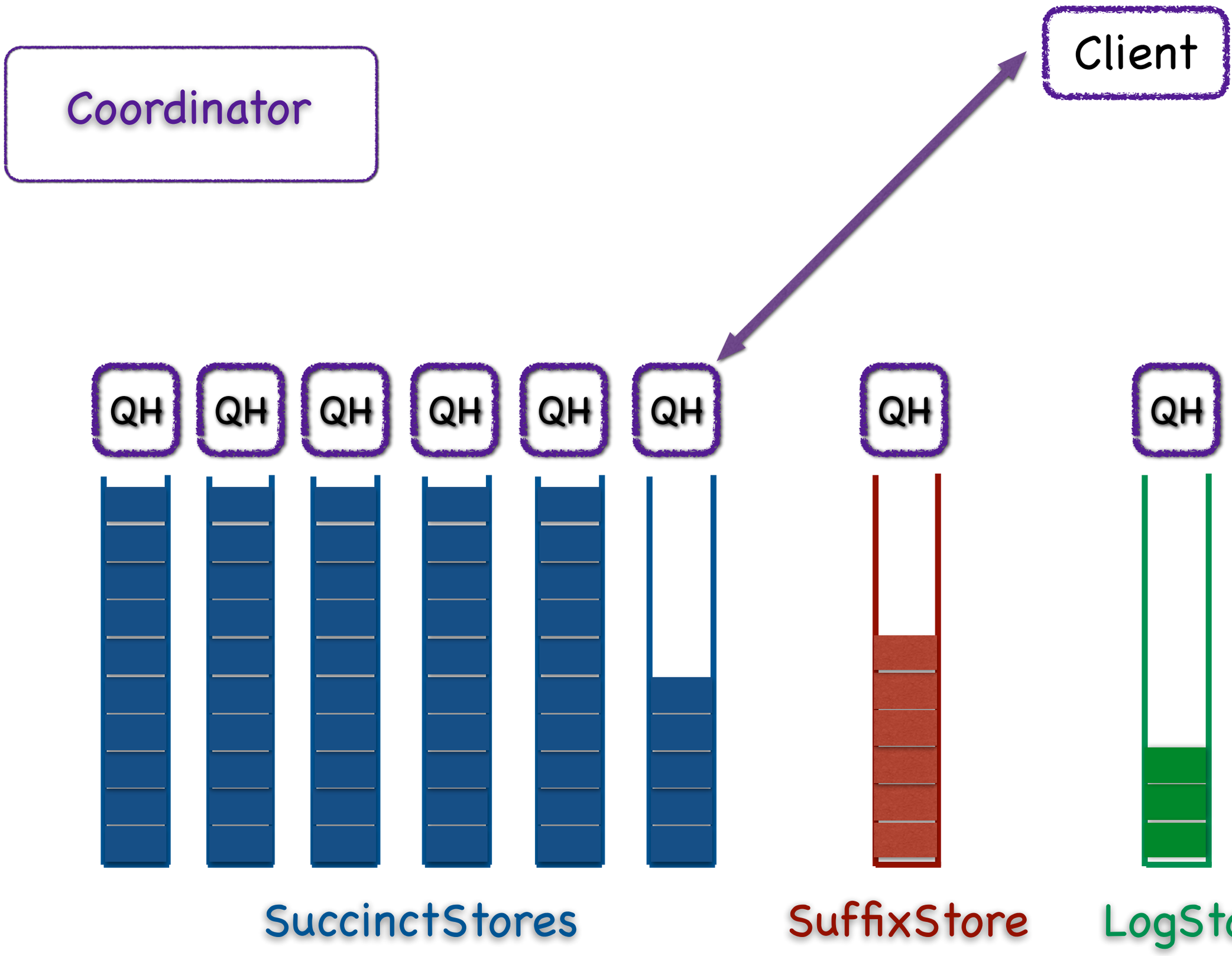
Coordinator

QH QH QH QH QH QH QH QH

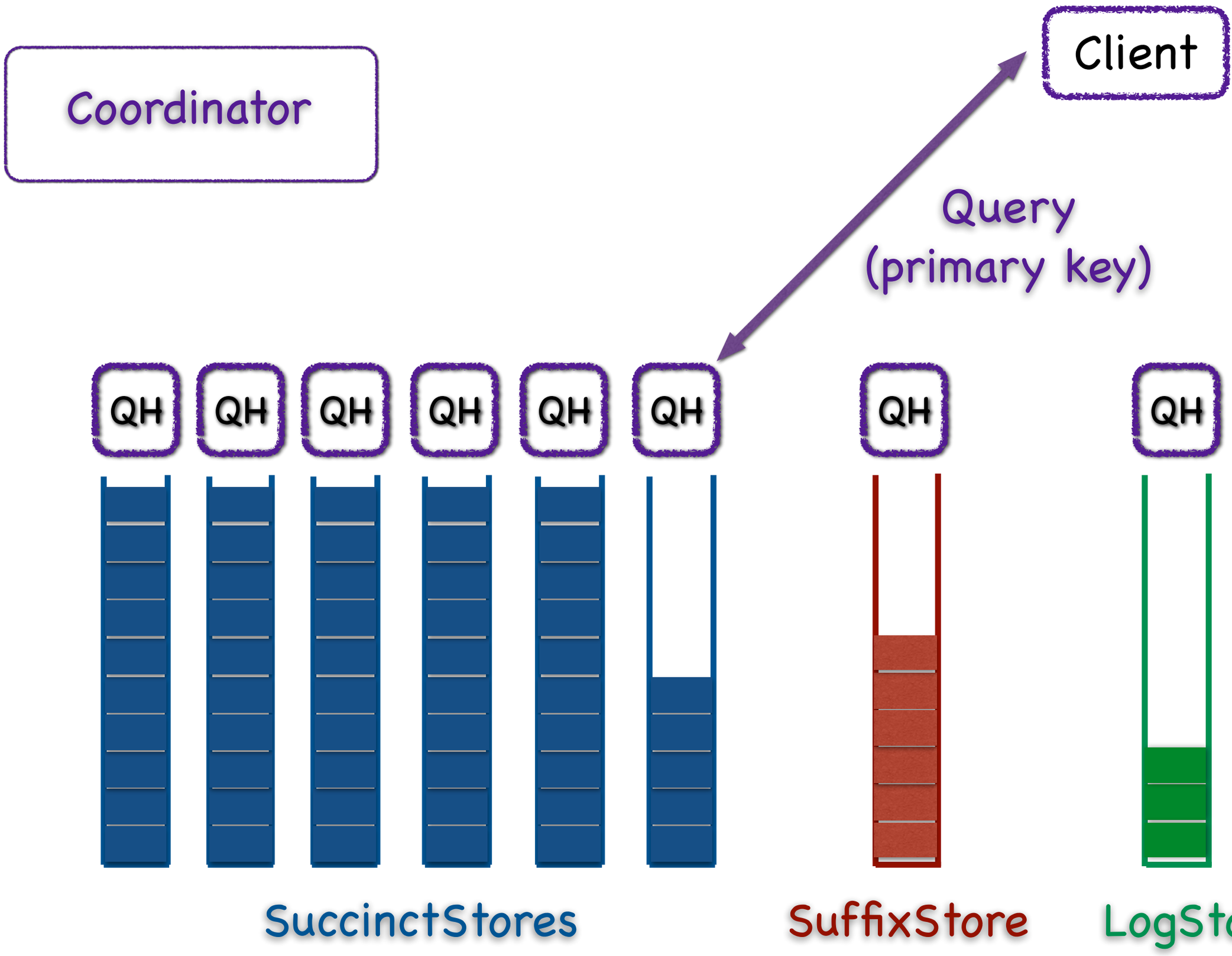


SuccinctStores SuffixStore LogStore

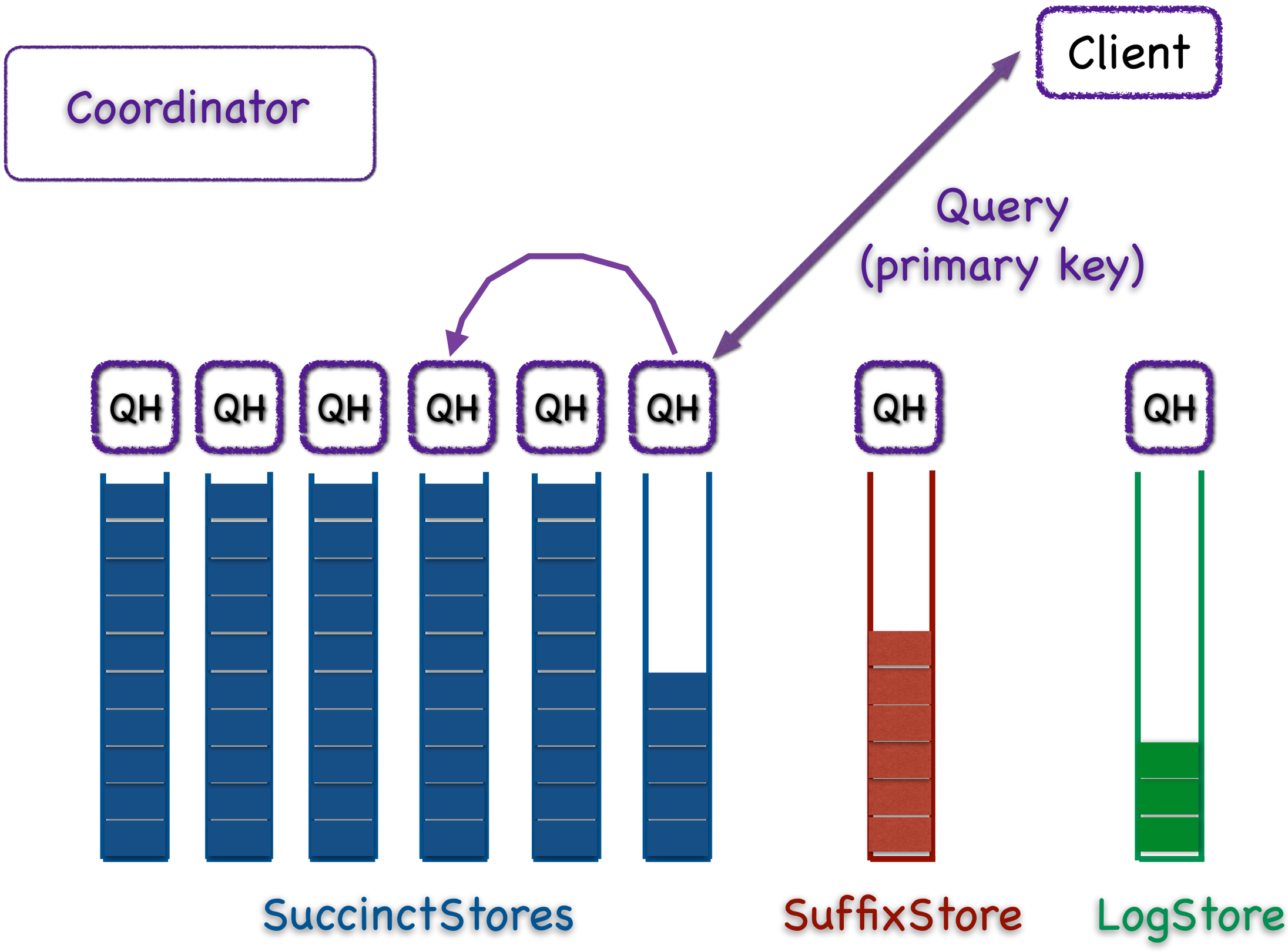
End-to-end System



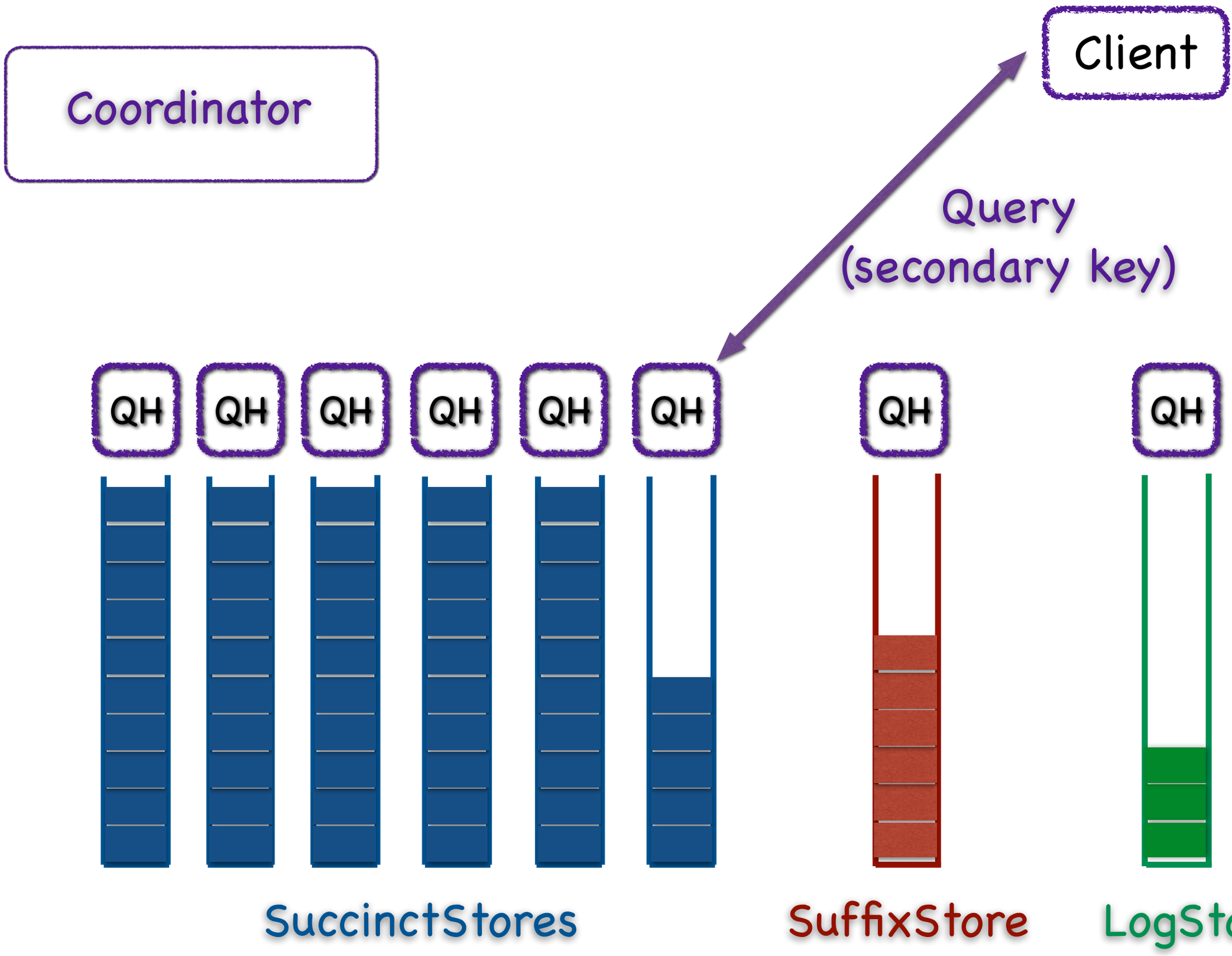
End-to-end System



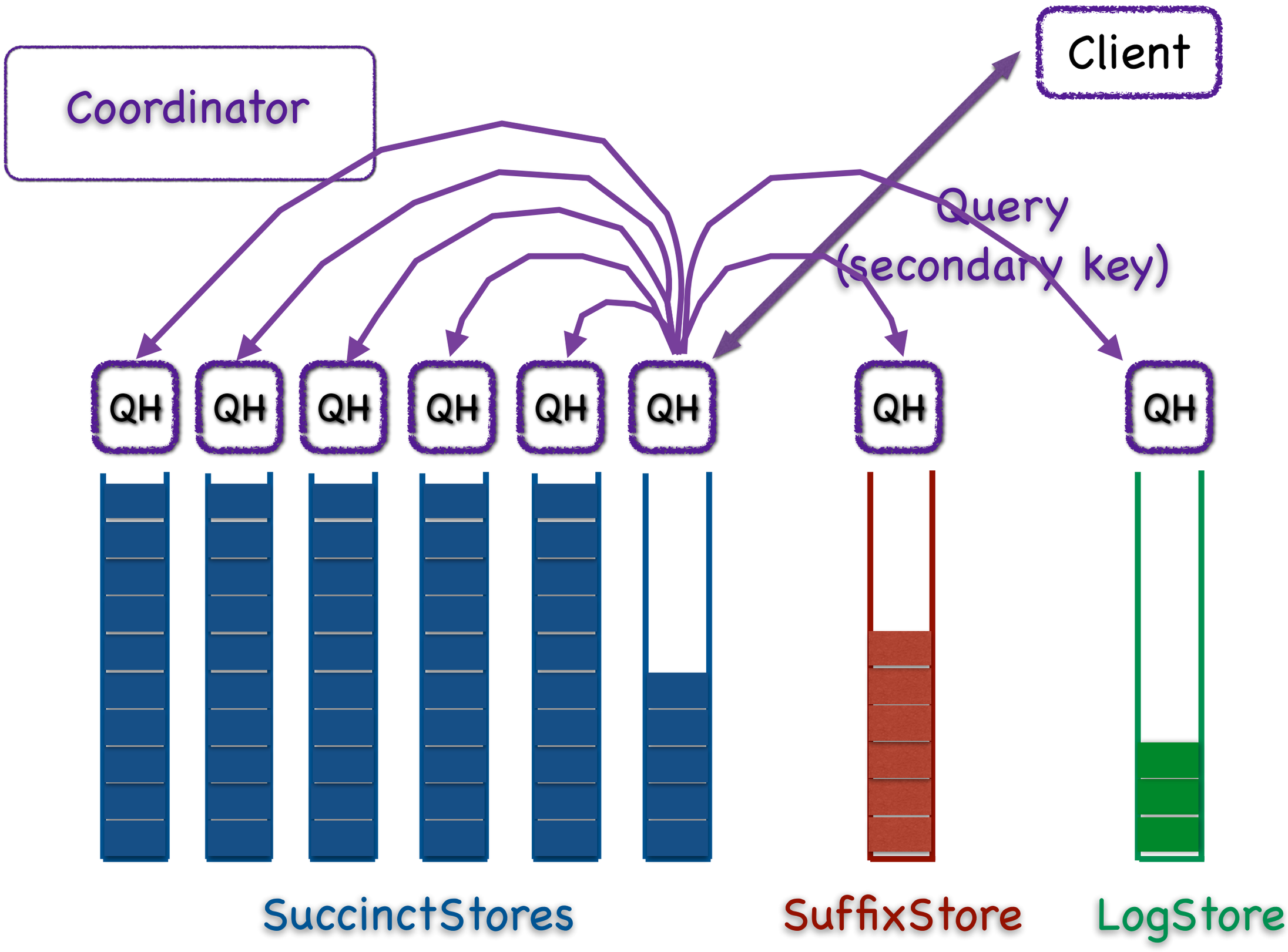
End-to-end System



End-to-end System



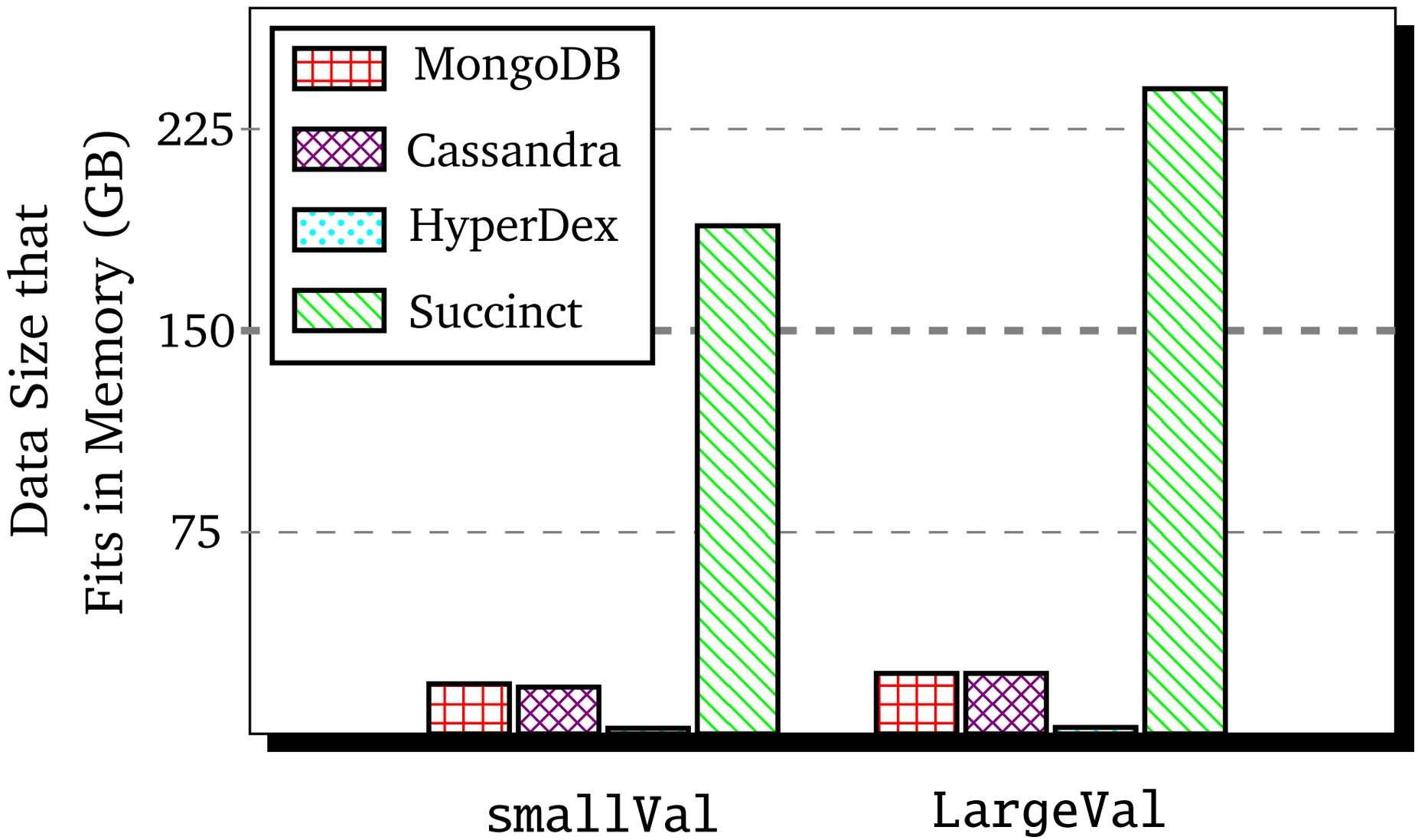
End-to-end System



Evaluation

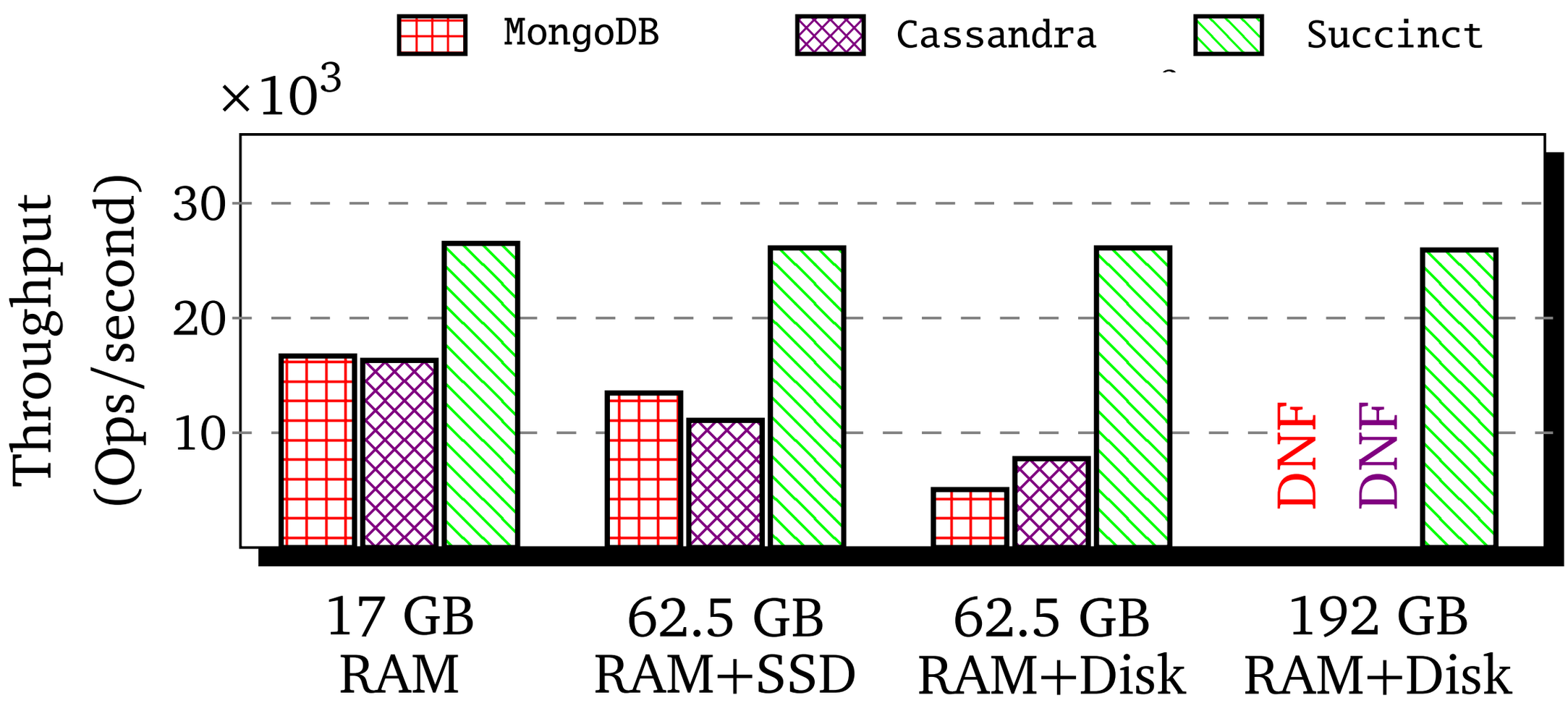
Datasets	SmallKV, LargeKV from Conviva customers
	Record length: 140B (SmallKV), 1.3kB (LargeKV)
Cluster	Amazon EC2, 10 machines, m1.xlarge machines
Workload	YCSB for primary; YCSB mapped to secondary keys
Systems	MongoDB, Cassandra, HyperDex, DB-X
Caveat	Absolute numbers are dataset dependent

10-11x lower storage



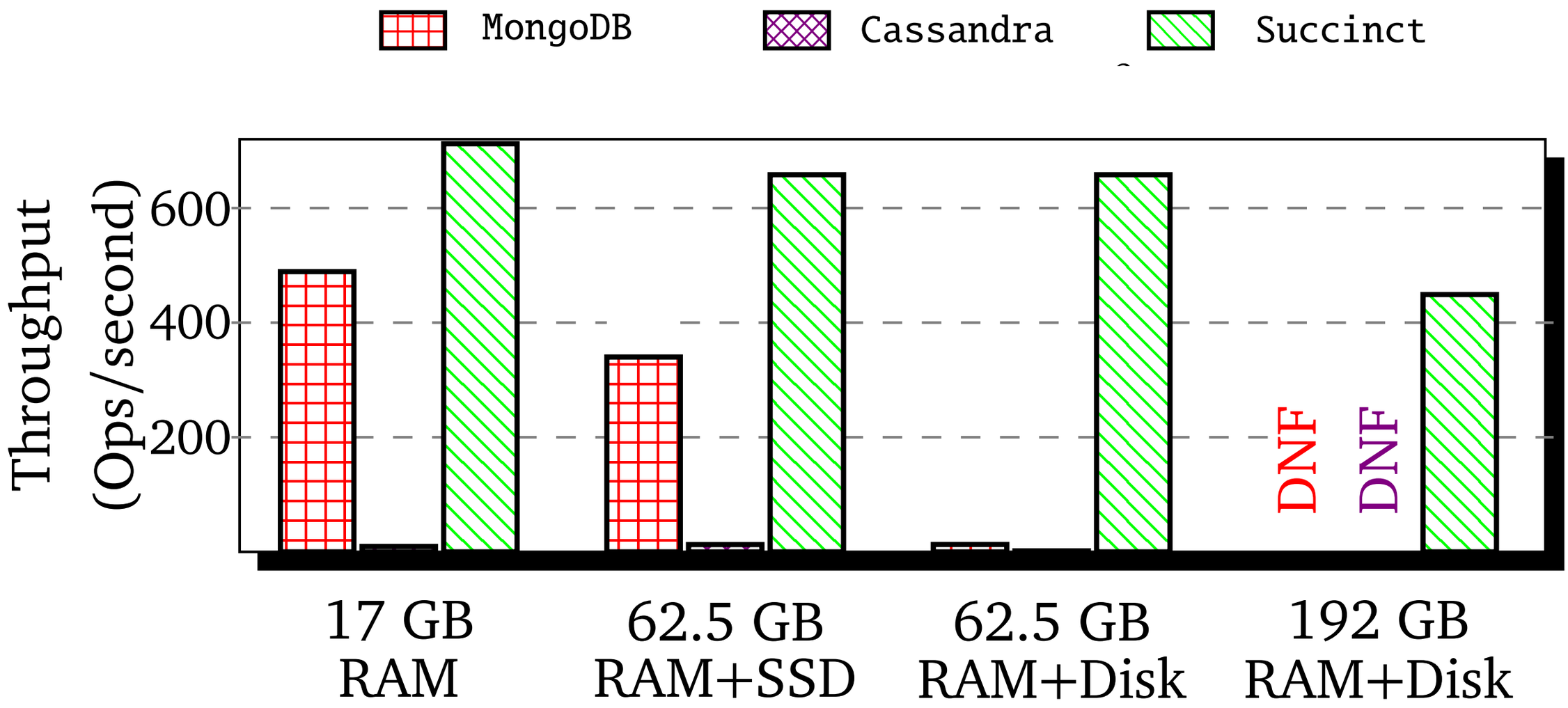
Take-away: Succinct pushes 10-11x more data in faster storage than systems with similar functionality (including every single bit stored)

Throughput for 95% GET + 5% PUT (SmallKV)



Take-away: Succinct achieves performance comparable to existing open-source systems for queries on primary attributes

Throughput for 95% SEARCH + 5% PUT (SmallKV)



Take-away: Succinct by pushing more data in faster storage provides performance similar to existing systems for 10-11x larger data sizes

Summary

Succinct pushes 10-11x more data in faster storage (compared to popular open-source systems)

Succinct executes queries directly on compressed representation

Avoids data scans and data decompression costs

Enables interactive queries on secondary keys (for semi-structured data)