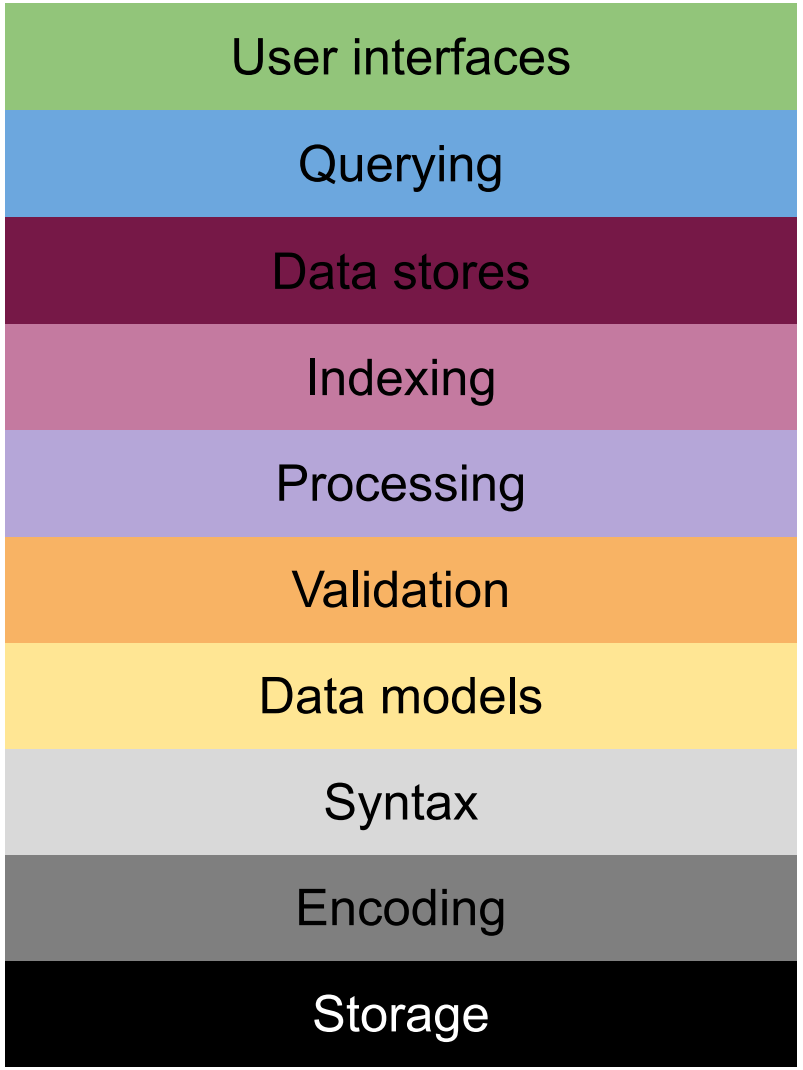
ETH zürich

Ghislain Fourny

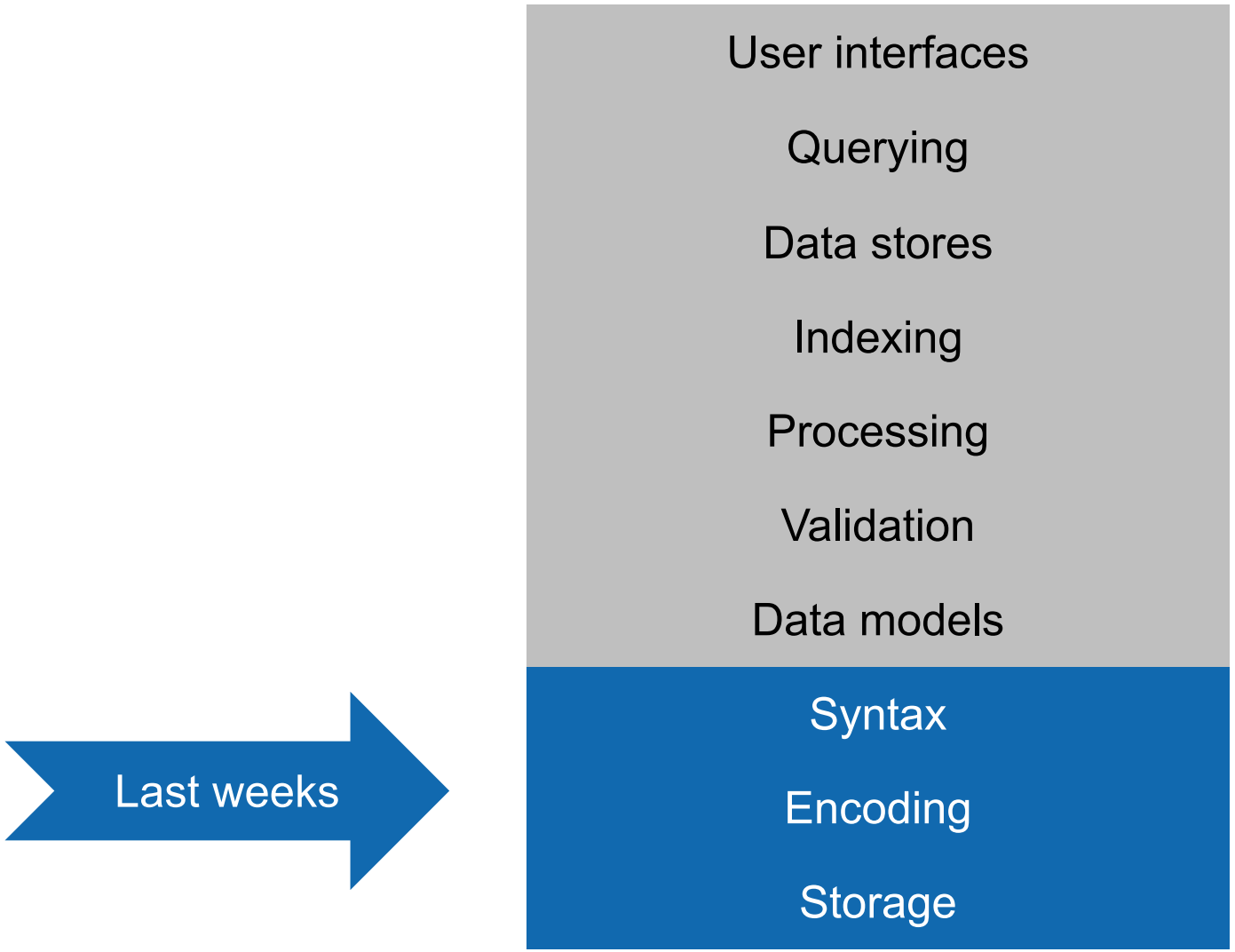
Big Data for Engineers Spring 2020

6. Wide column stores

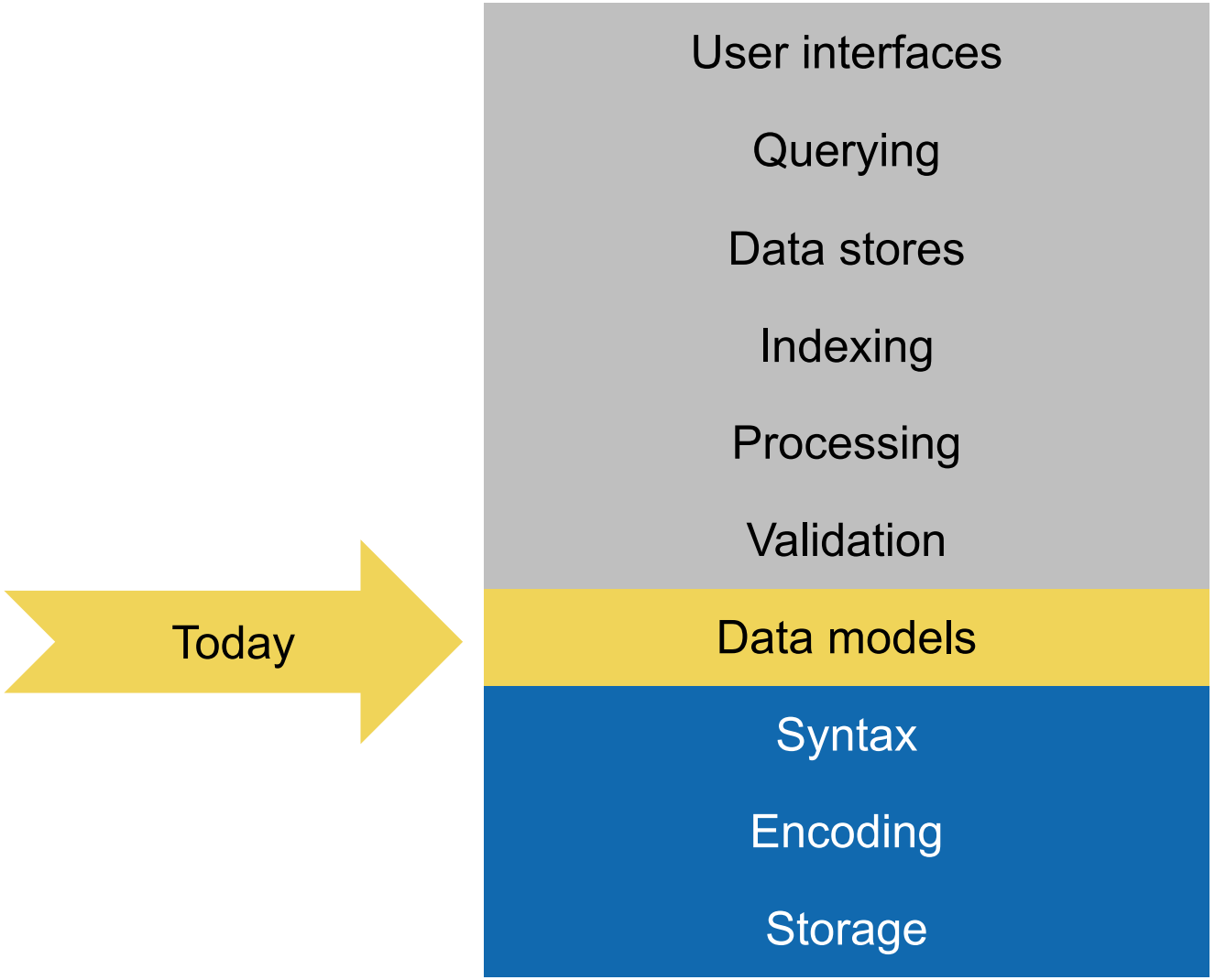
Data Technology Stack



Where we are

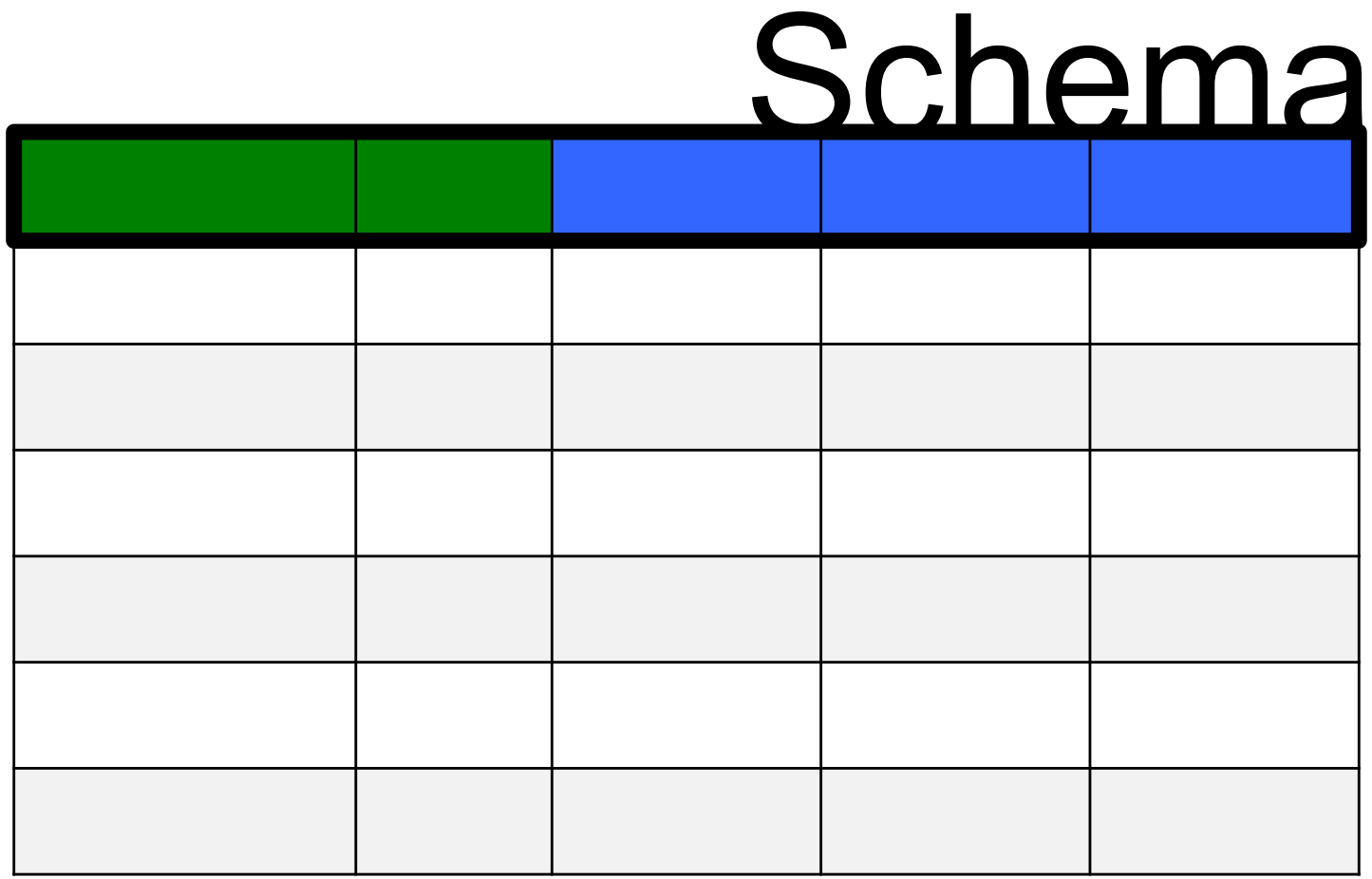

User interfaces
Querying
Data stores
Indexing
Processing
Validation
Data models
Last weeks
Syntax
Encoding
Storage

Where we are


User interfaces
Querying
Data stores
Indexing
Processing
Validation
Today
Data models
Syntax
Encoding
Storage

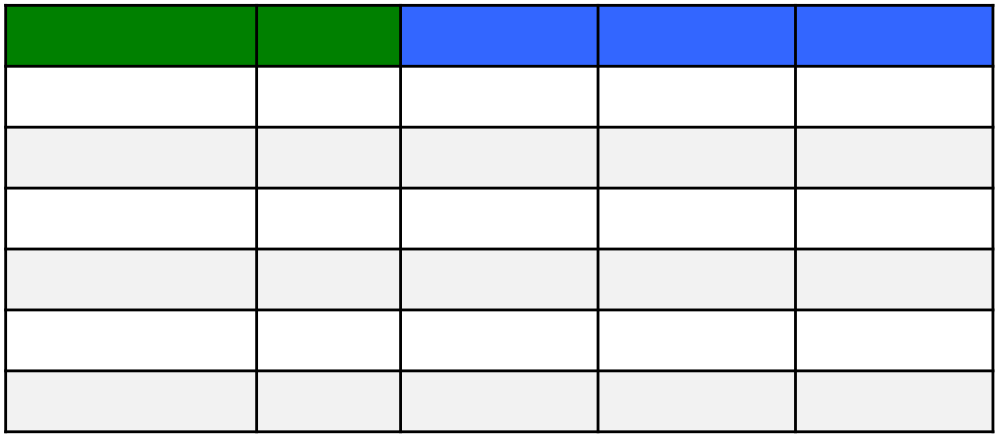
Relational model

Relational model

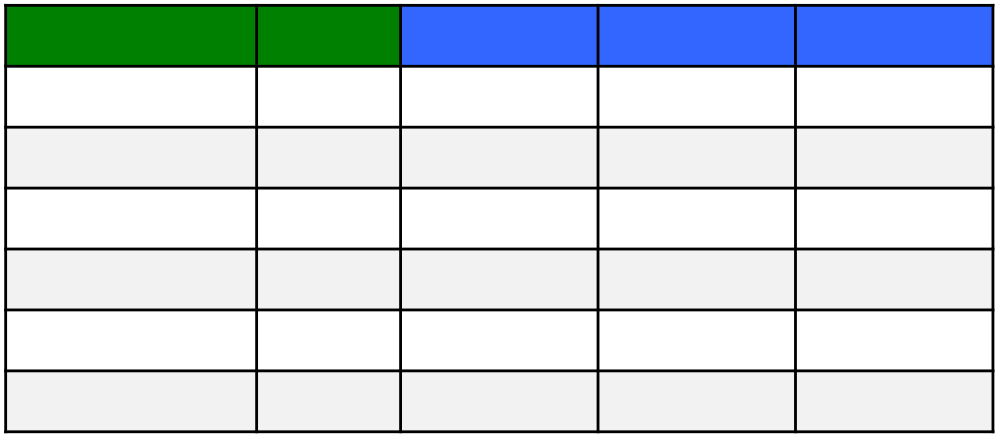


Issues with relational databases (RDBMS)

- Small scale
- Single machine



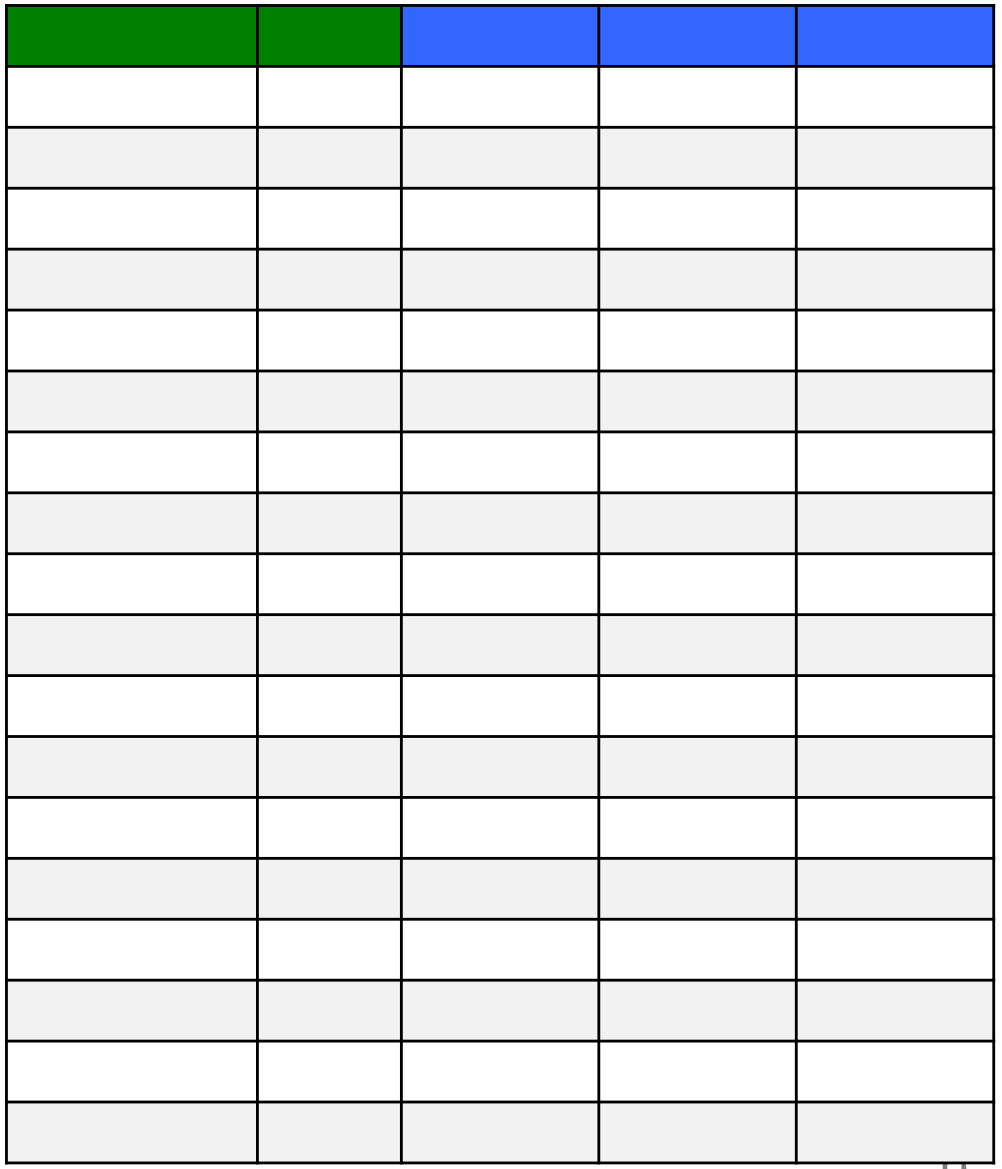
Can we *fix* a RDBMS?



Can we *fix* a RDBMS?

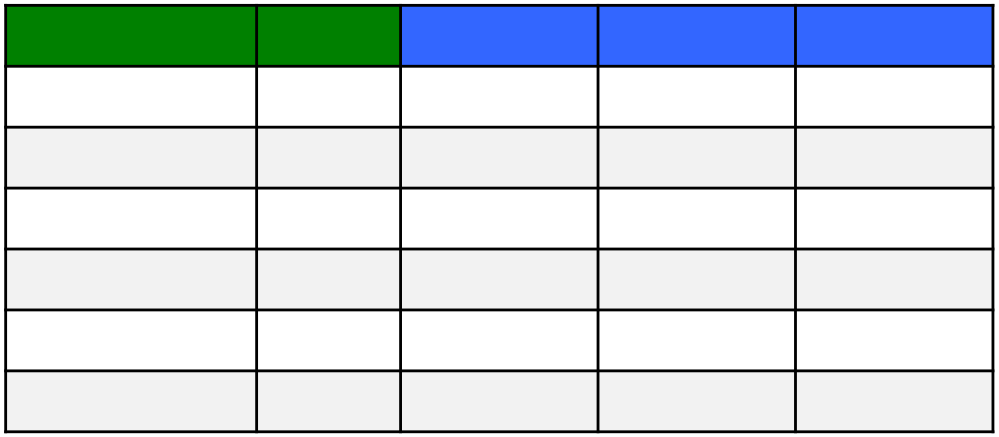
Scale up

(remember?)



Can we *fix* a RDBMS?

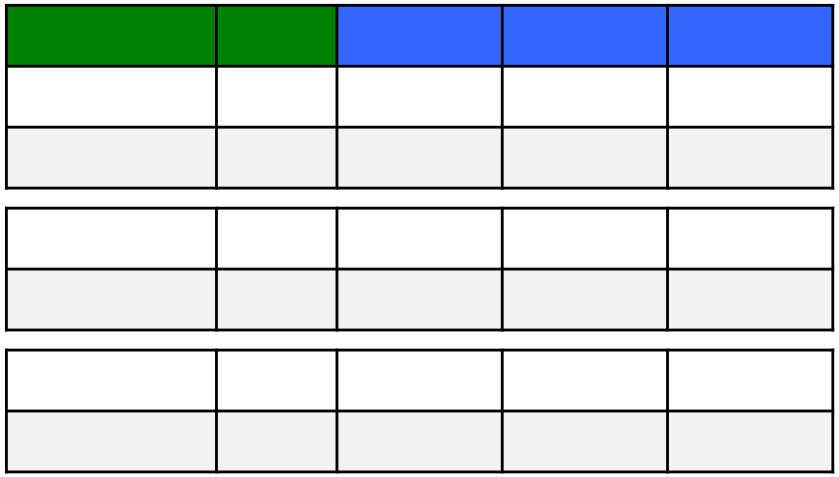
Scale out



Can we *fix* a RDBMS?

- Cluster

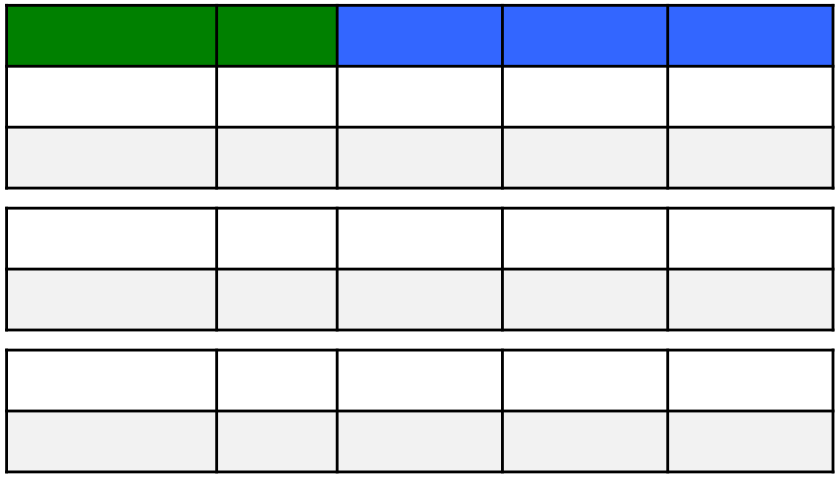
Scale out

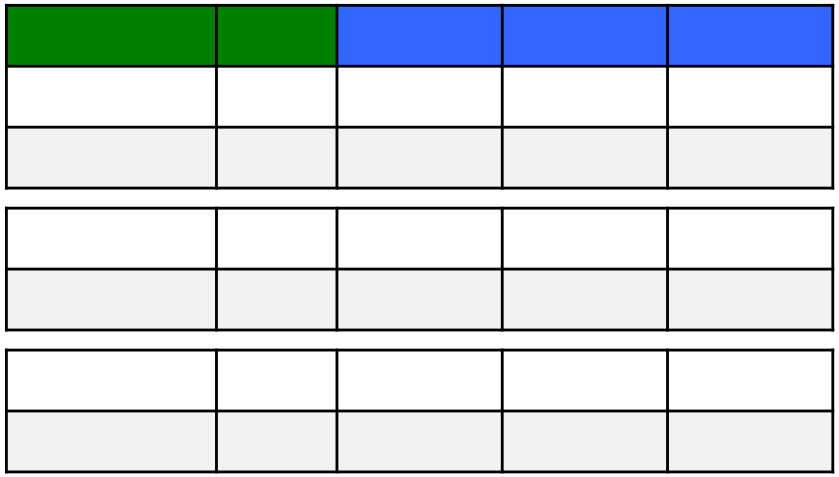


Can we *fix* a RDBMS?

- Cluster
- Replicate

Scale out

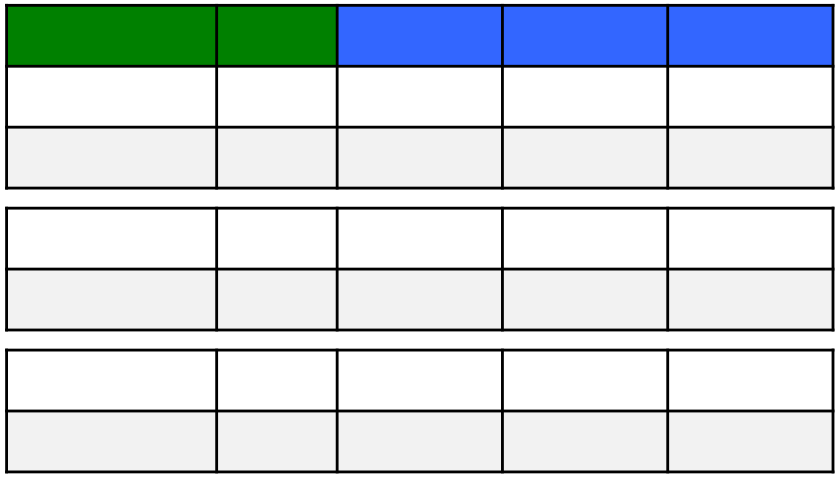


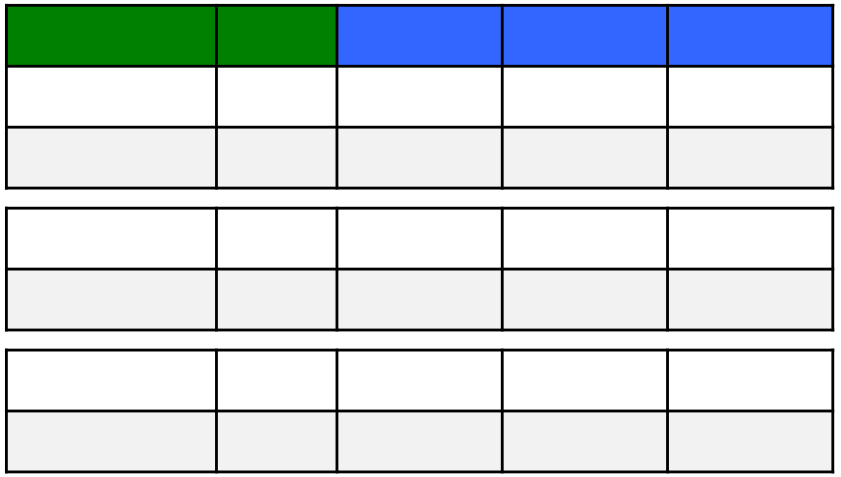


Can we *fix* a RDBMS?

- Hard to set up
- Very high maintenance costs

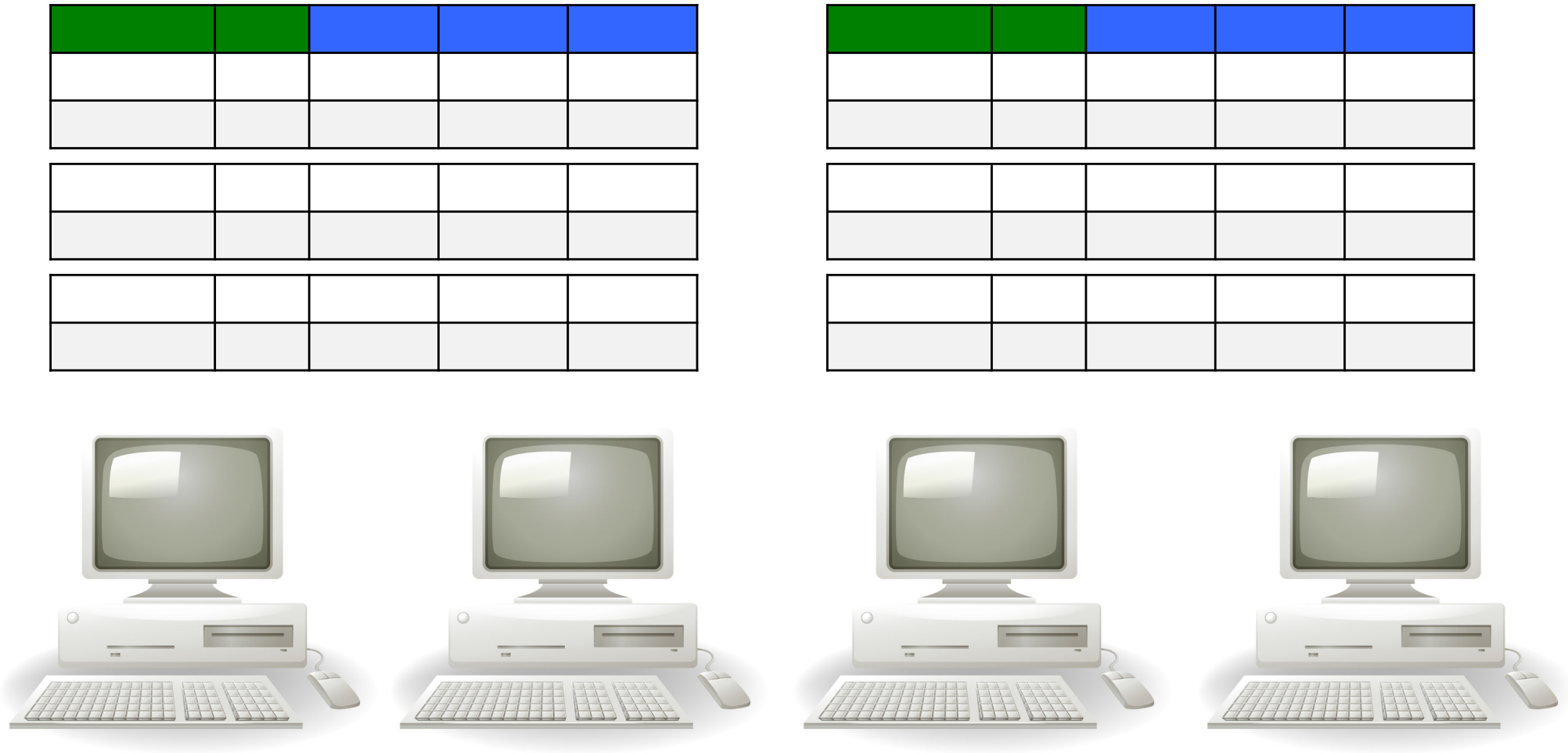
Scale out





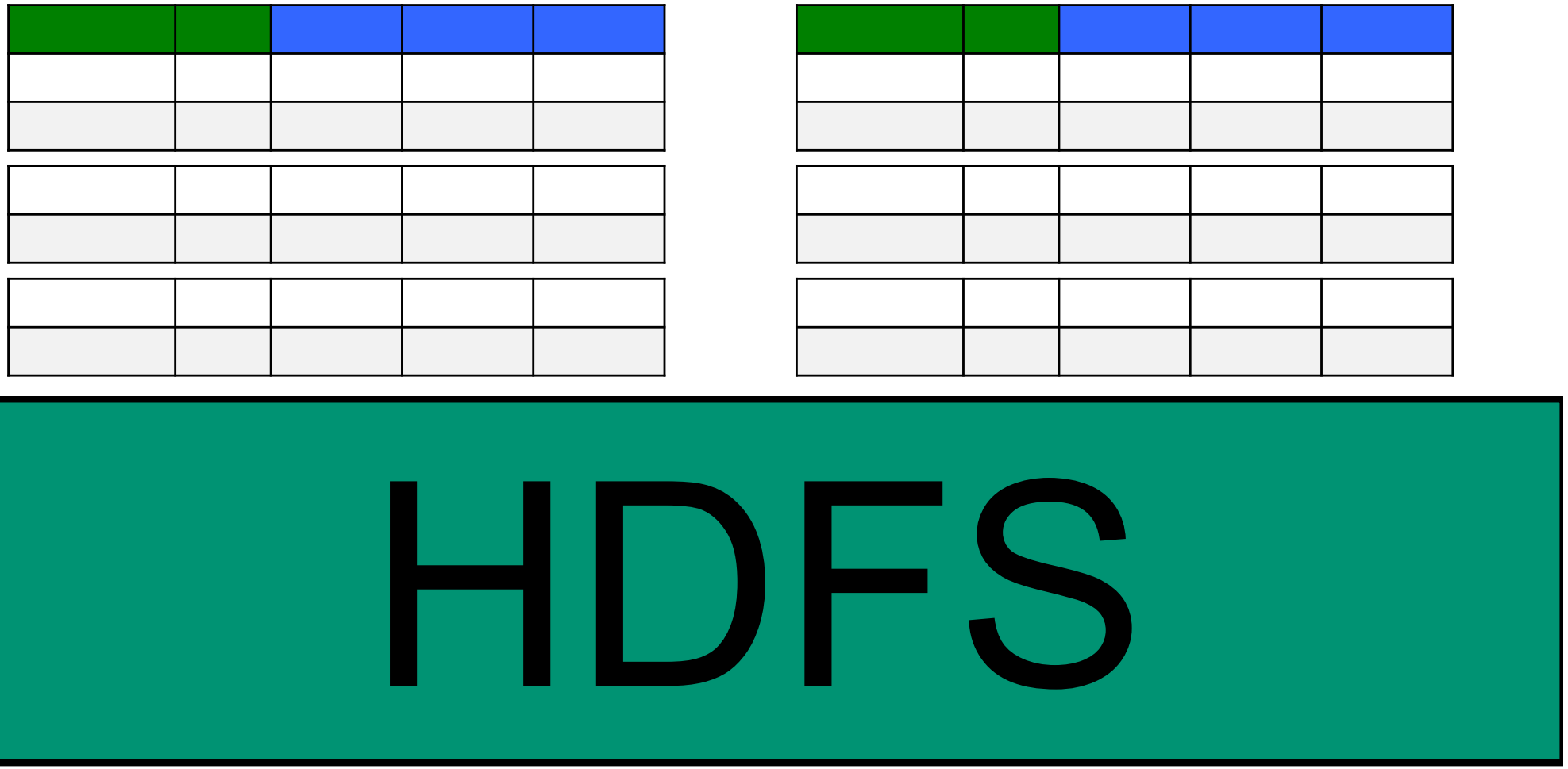
HBase

By design running on a scalable cluster of commodity hardware



HBase

By design running on a scalable cluster of commodity hardware



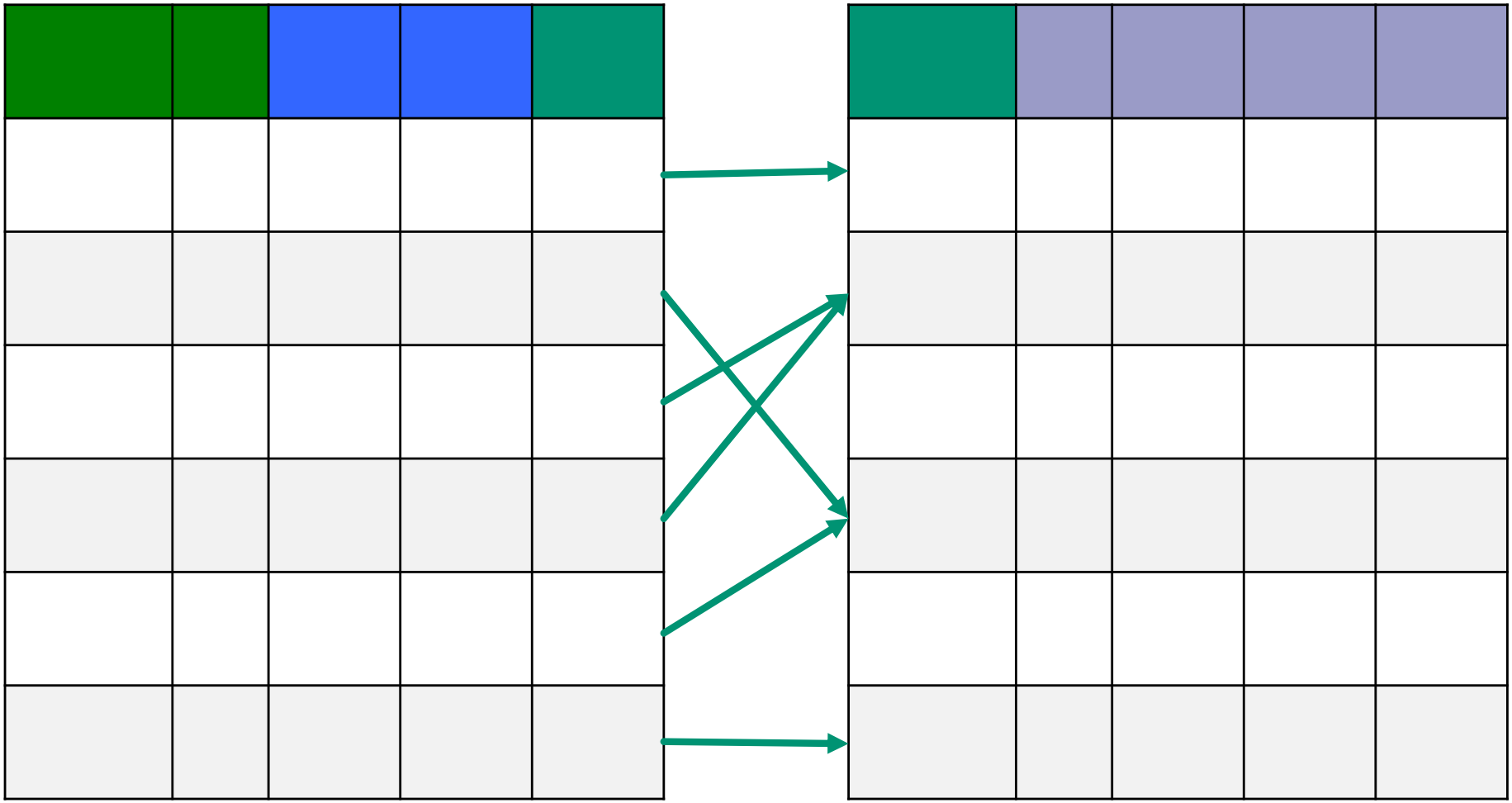
Wide column stores: data model

Founding paper

Google's BigTable

The tabular model

The tabular model: expensive joins



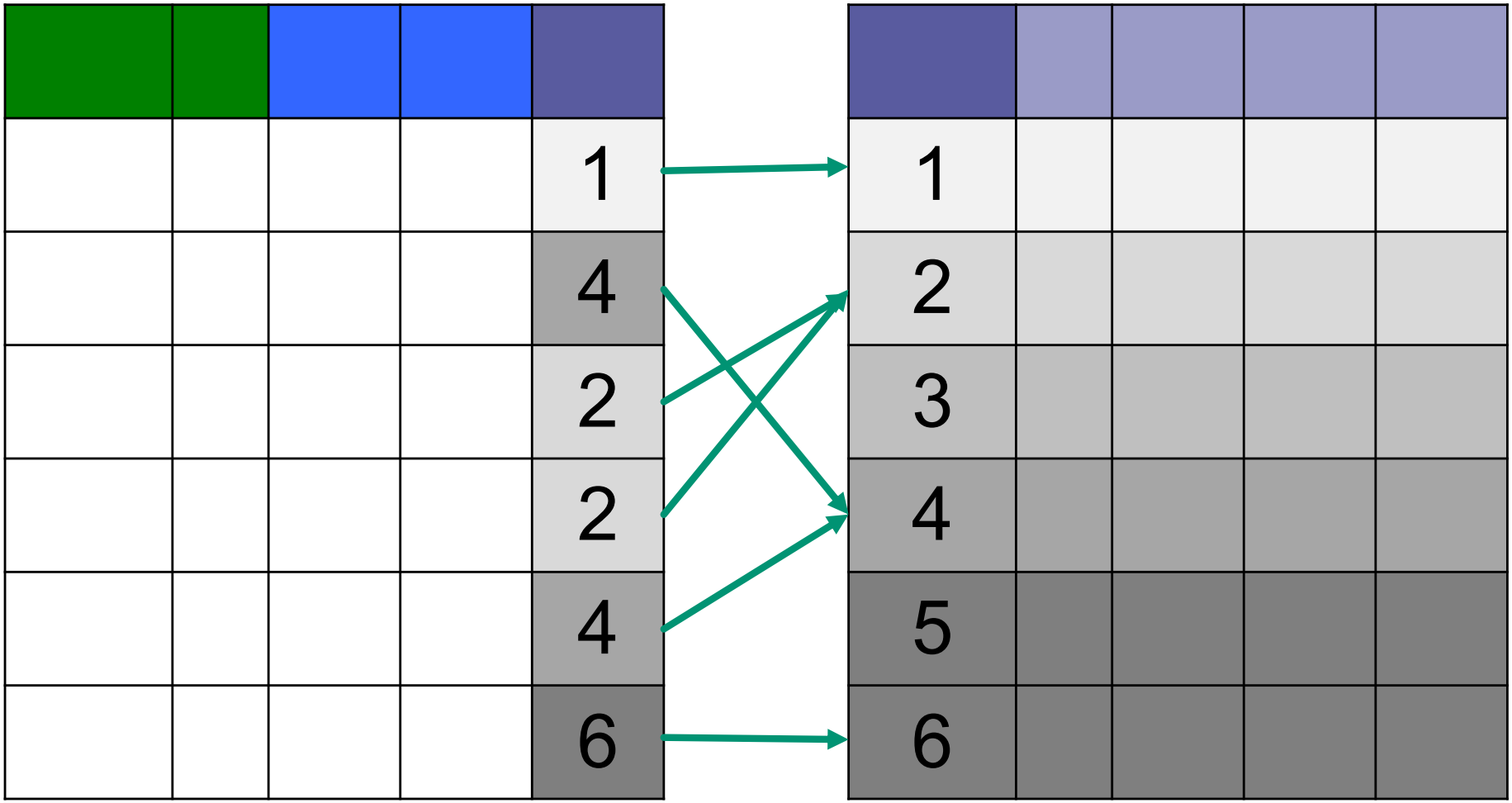
Design paradigm of BigTable

store together

what is

accessed together

The tabular model: expensive joins



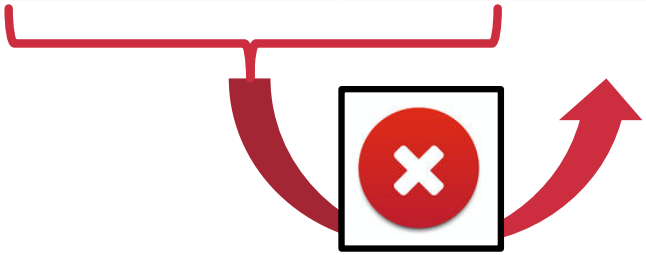
Boyce-Codd Normal Form: Example

Legi	Name	City	State
32-000-000	Alan Turing	Bletchley Park	UK
32-000-000	Alan Turing	Bletchley Park	UK
62-000-000	Georg Cantor	Pfäffikon	SZ
62-000-000	Georg Cantor	Pfäffikon	SZ
25-000-000	Felix Bloch	Pfäffikon	ZH

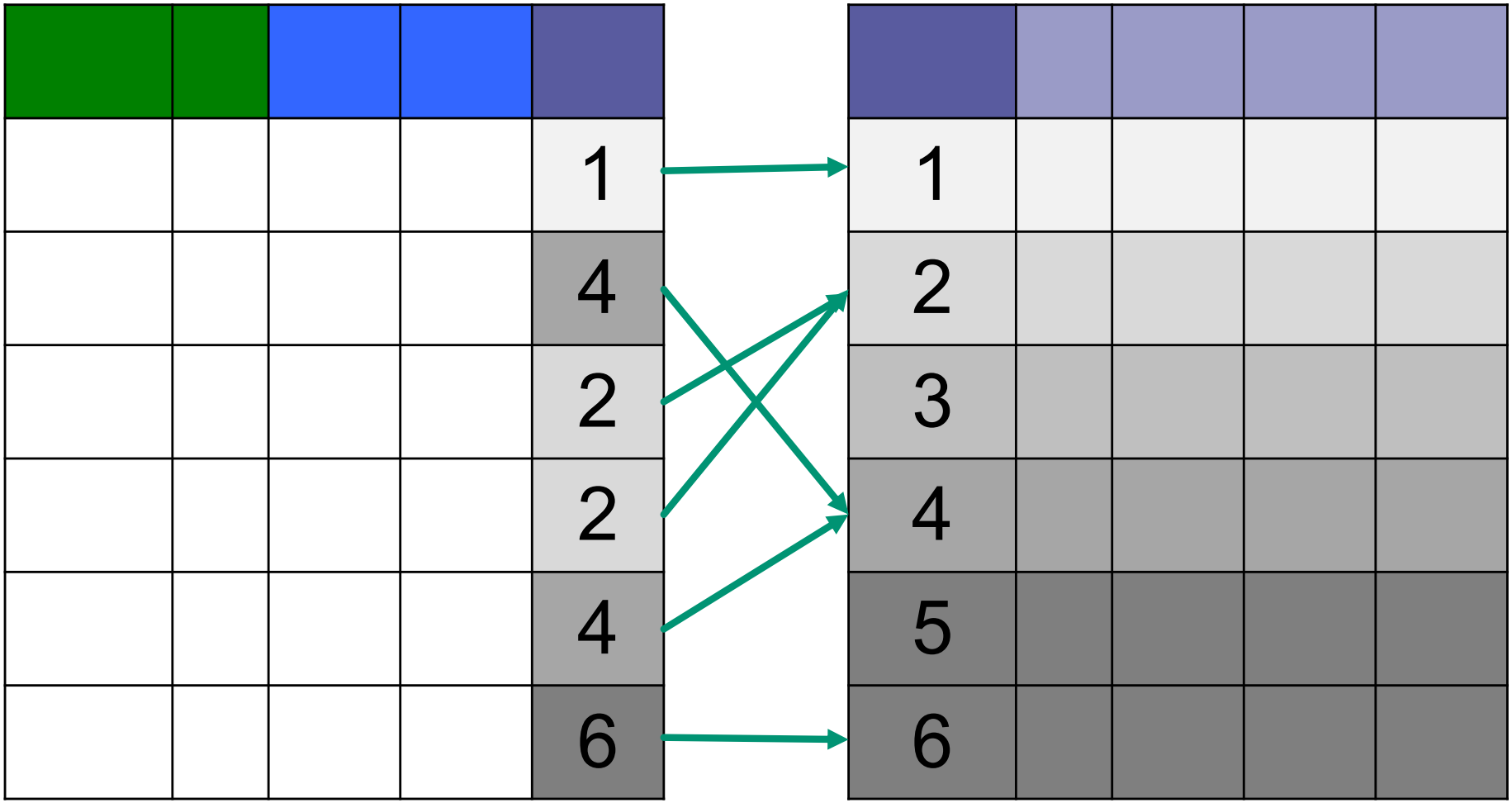
City	State	PLZ
Bletchley Park	UK	MK3 6EB
Bletchley Park	UK	MK3 6EB
Pfäffikon	SZ	8808
Pfäffikon	SZ	8808
Pfäffikon	ZH	8330

Boyce-Codd Form: Counter-Example

Legi	Name	City	State	PLZ
32-000-000	Alan Turing	Bletchley Park	UK	MK3 6EB
32-000-000	Alan Turing	Bletchley Park	UK	MK3 6EB
62-000-000	Georg Cantor	Pfäffikon	SZ	8808
62-000-000	Georg Cantor	Pfäffikon	SZ	8808
25-000-000	Felix Bloch	Pfäffikon	ZH	8330



The tabular model: expensive joins



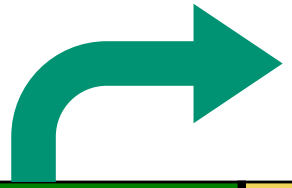
The columnar model: denormalized

				1				
				4				
				2				
				2				
				4				
				6				

Rows

Row ID				
000				
002				
0A1				
1E0				
22A				
4A2				

Rows



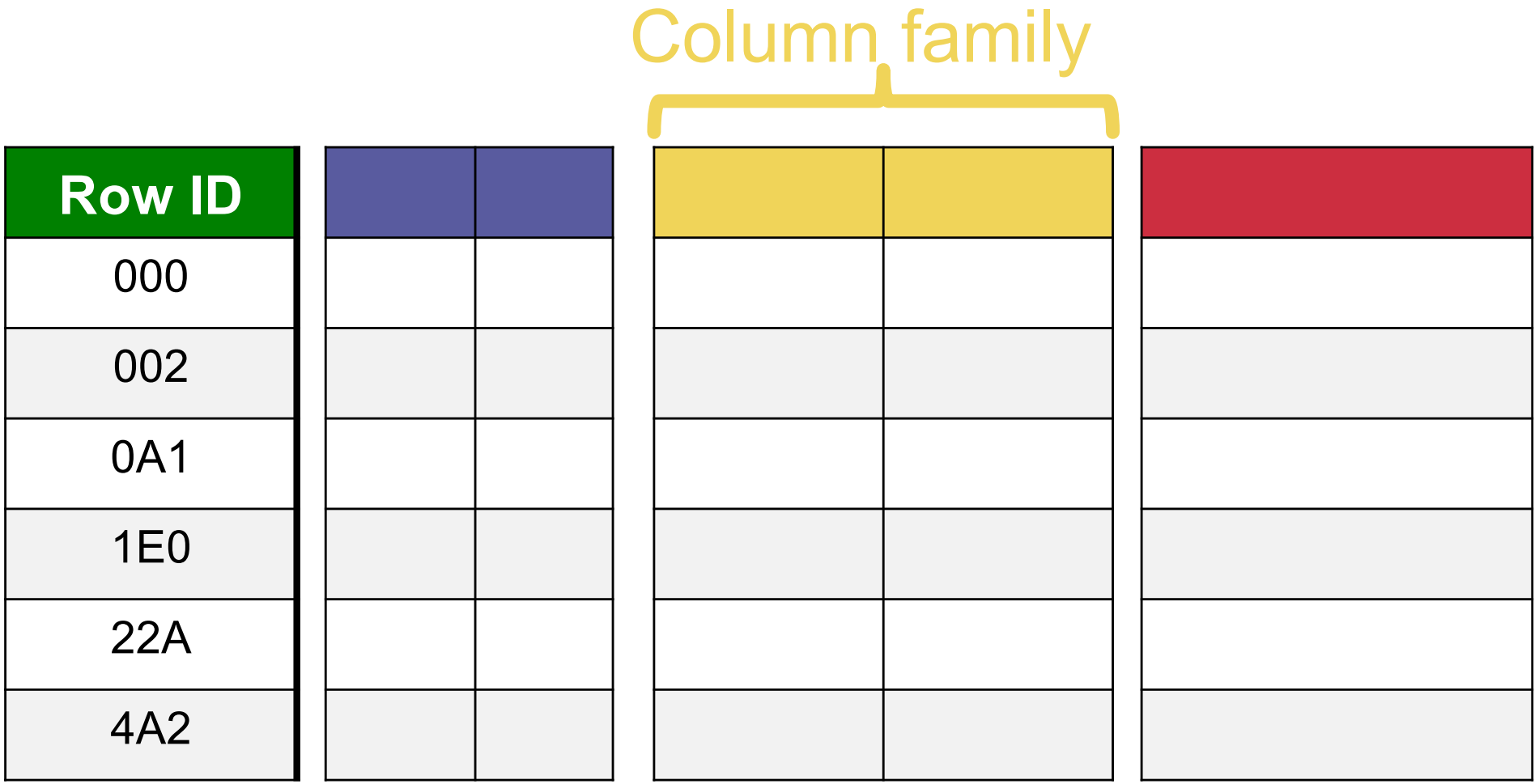
Yes, for now this actually looks pretty much like key-value storage.

Row ID				
000				
002				
0A1				
1E0				
22A				
4A2				

Columns

Row ID					
000					
002					
0A1					
1E0					
22A					
4A2					

Columns



Column families must be known in advance...

Column families must be known in advance...

Row ID	A	B	1	2	I
000					
002					
0A1					
1E0					
22A					
4A2					

... but columns can be added on the fly

Row ID	A	B	C	1	2	I	II	III	IV
000									
002									
0A1									
1E0									
22A									
4A2									

Primary queries

Get

Row ID	A	B	C	1	2	I	II	III	IV
000									
002									
0A1									
1E0									
22A									
4A2									

Put

Row ID	A	B	C	1	2	I	II	III	IV
000									
002									
0A1									
1E0									
204									
22A									
4A2									

Scan

Row ID	A	B	C	1	2	I	II	III	IV
000									
002									
0A1									
1E0									
204									
22A									
4A2									

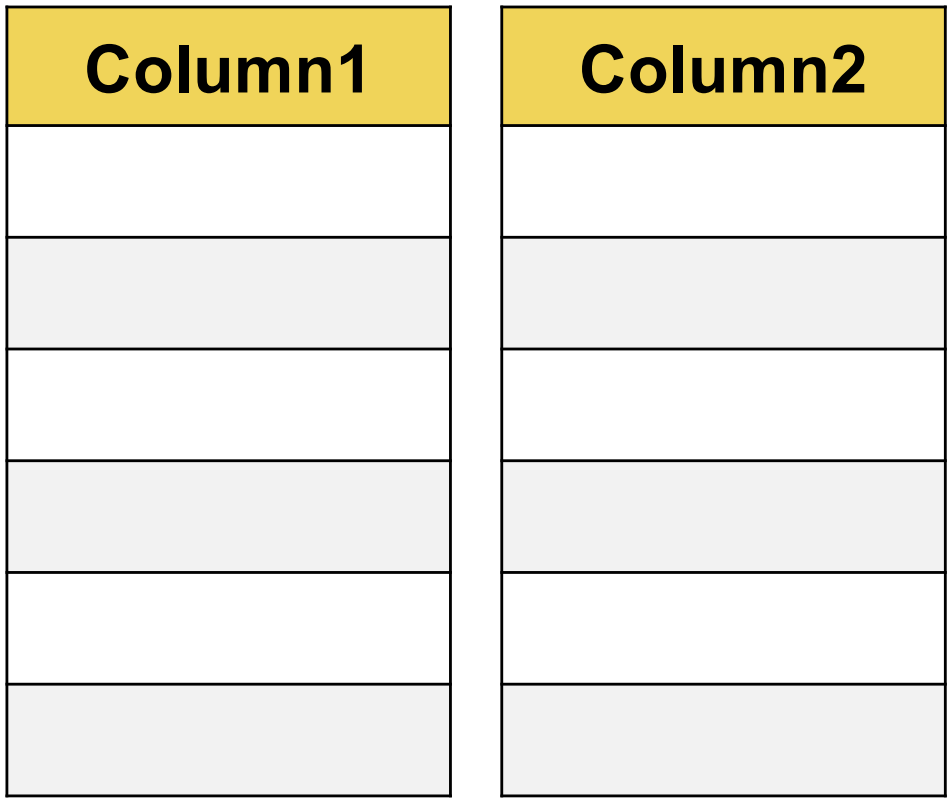
Delete

Row ID	A	B	C	1	2	I	II	III	IV
000									
002									
0A1									
1E0									
22A									
4A2									

Some terminology: Key-value model

Key	Value

Some terminology: Column-oriented storage

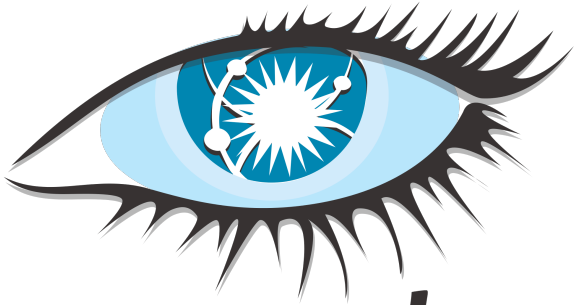


Some terminology: Wide column stores

Row ID	A	B	C	1	2	I	II	III	IV

Examples of wide column stores

Google's BigTable

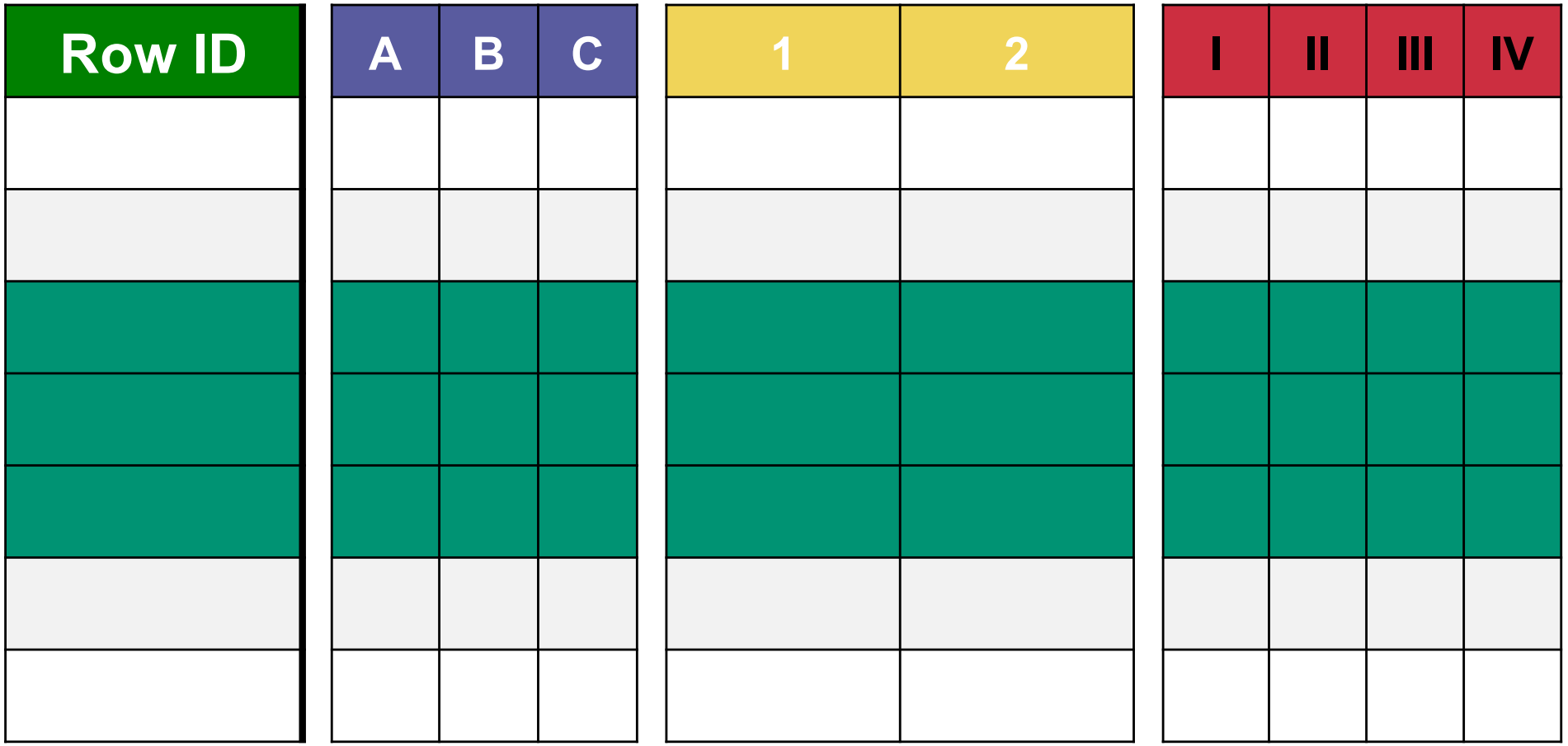


HBase: physical level

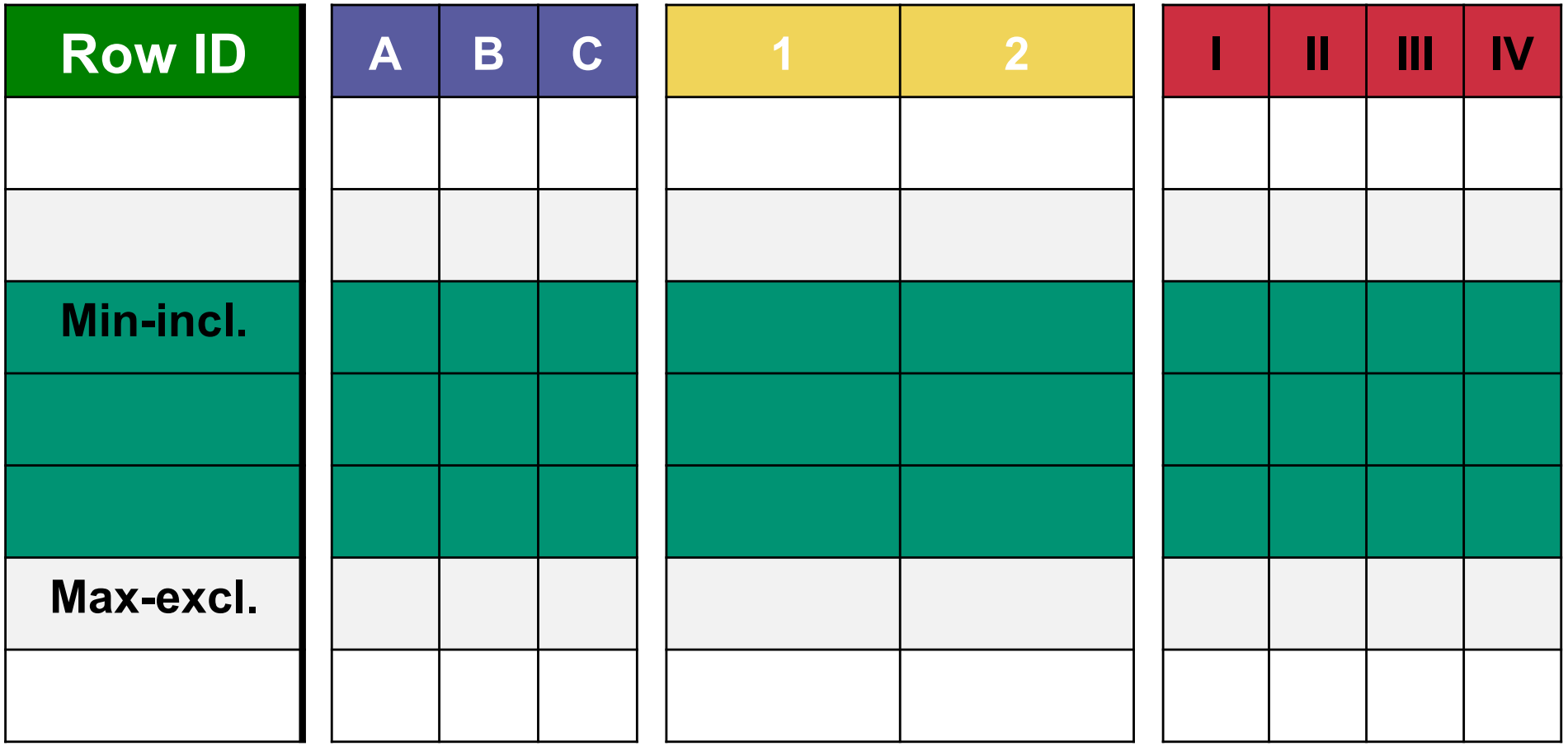
Physical layer: regions

Row ID	A	B	C	1	2	I	II	III	IV

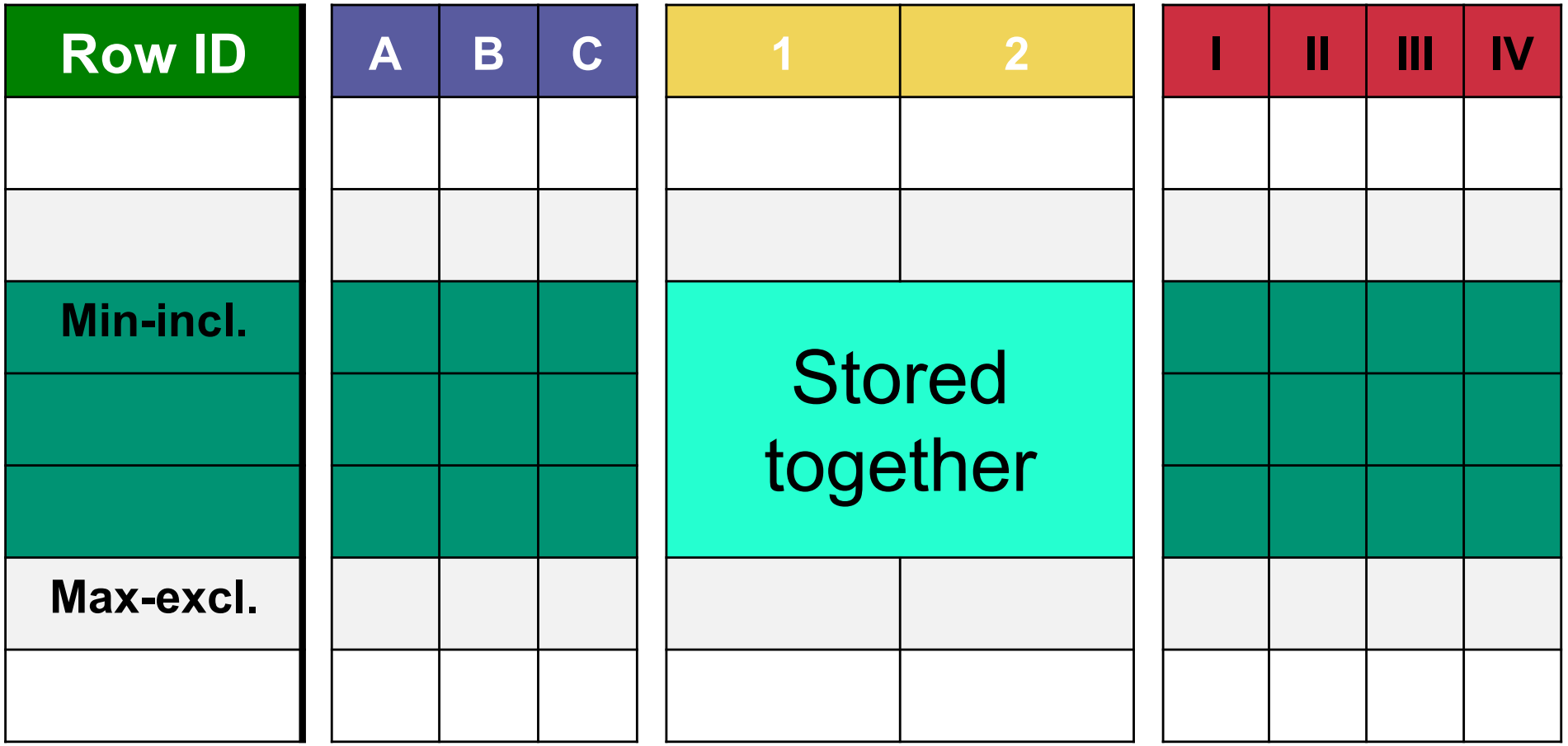
Physical layer: regions



Physical layer: regions



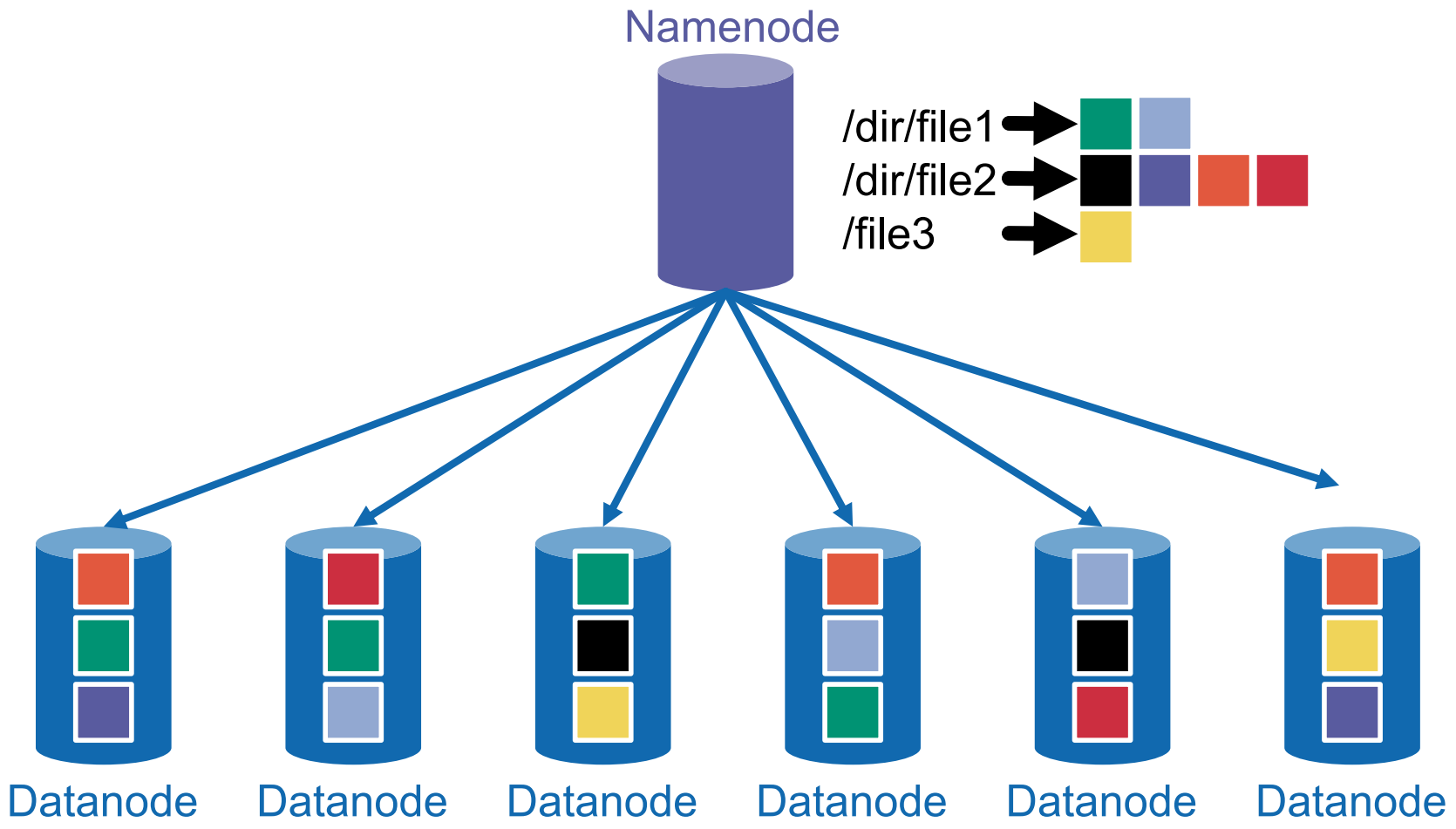
Physical layer: column families



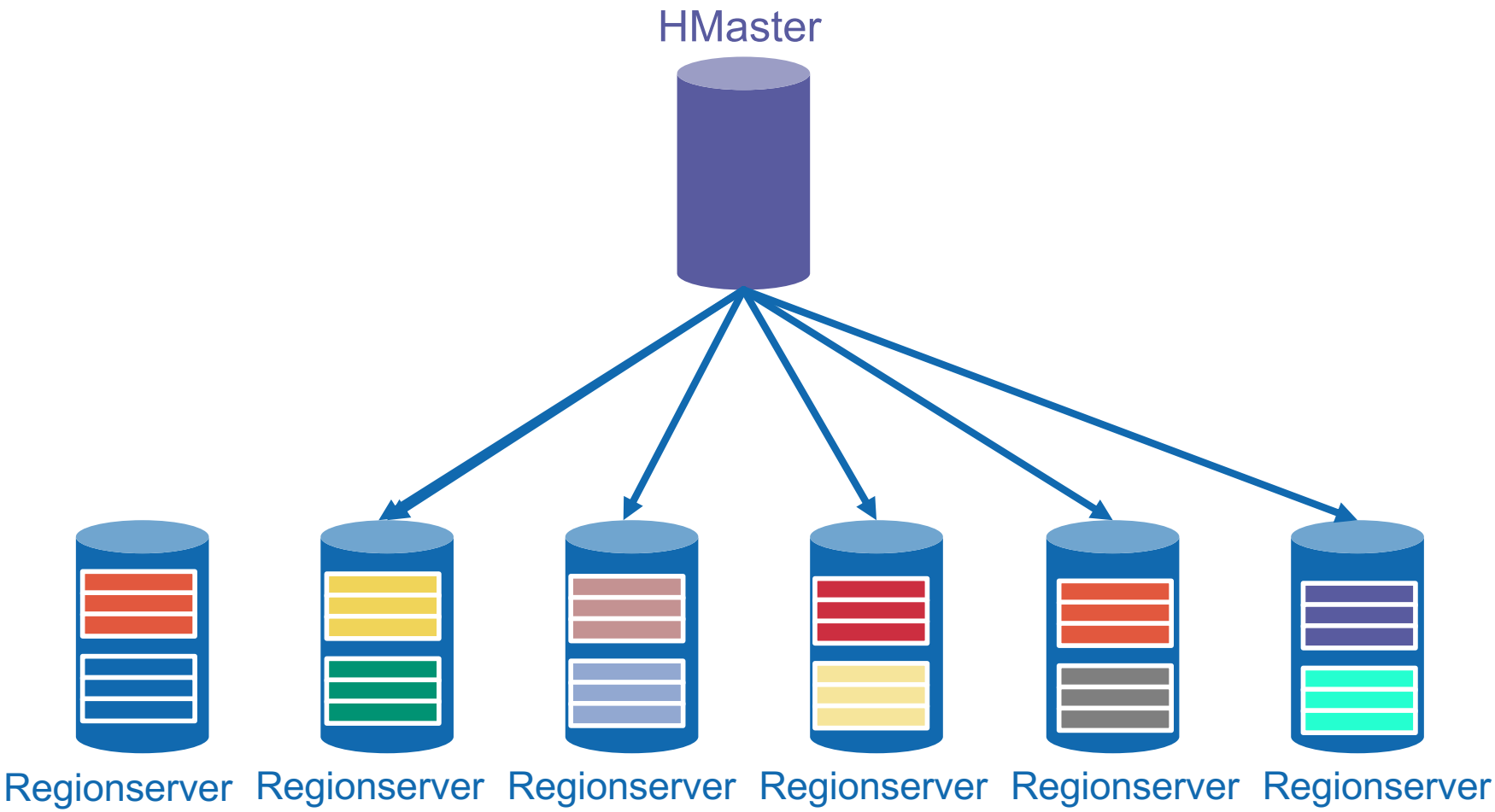
Architecture

"The same procedure as every year, James."

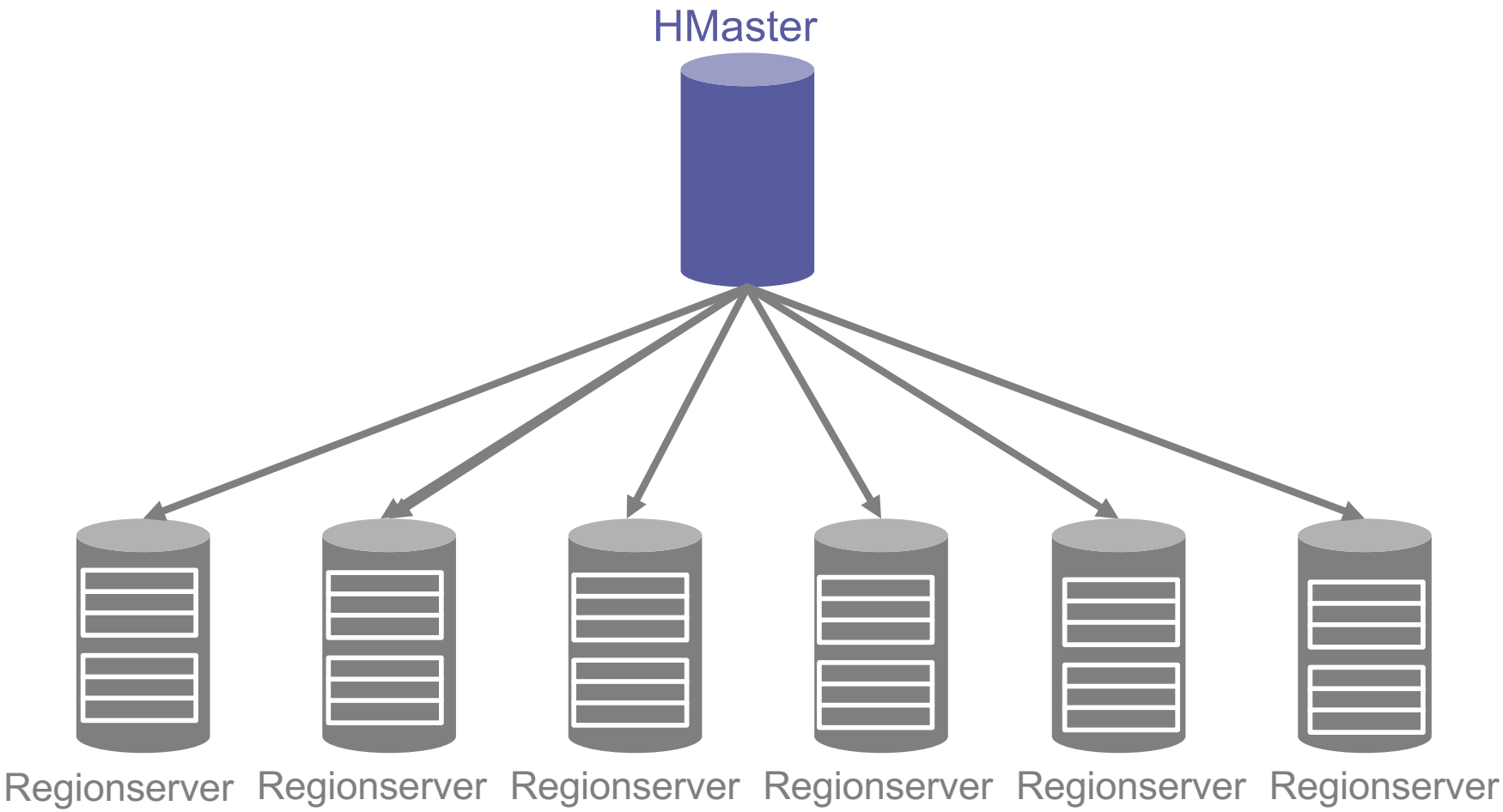
HDFS...


Namenode
/dir/file1
/dir/file2
/file3
Datanode
Datanode
Datanode
Datanode
Datanode
Datanode

HBase

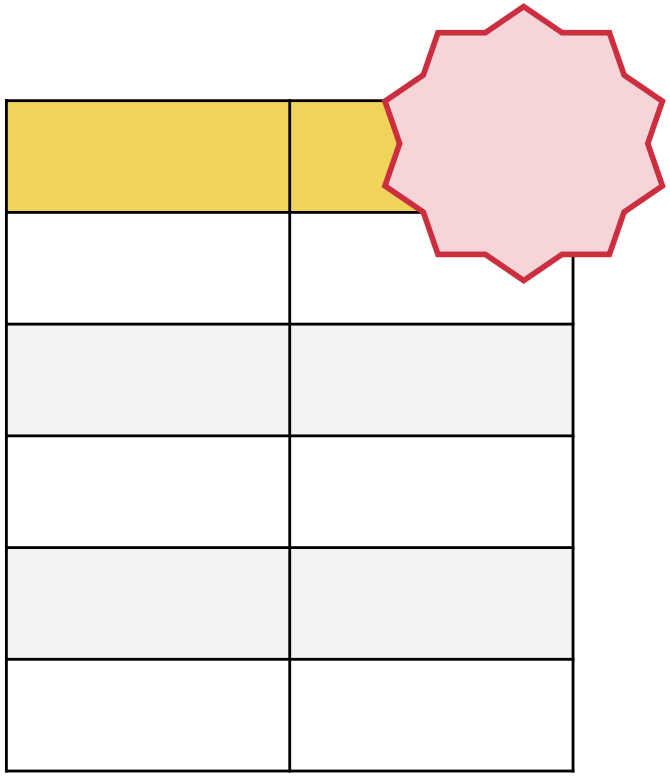

HMaster
Regionserver
Regionserver
Regionserver
Regionserver
Regionserver
Regionserver

HMaster

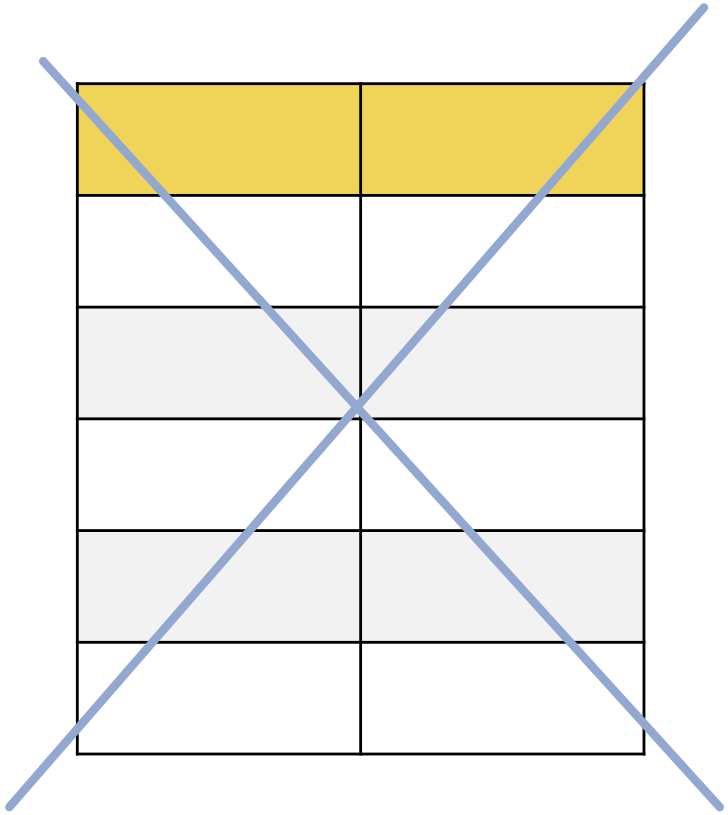

HMaster
Regionserver
Regionserver
Regionserver
Regionserver
Regionserver
Regionserver

HMaster

DDL operations

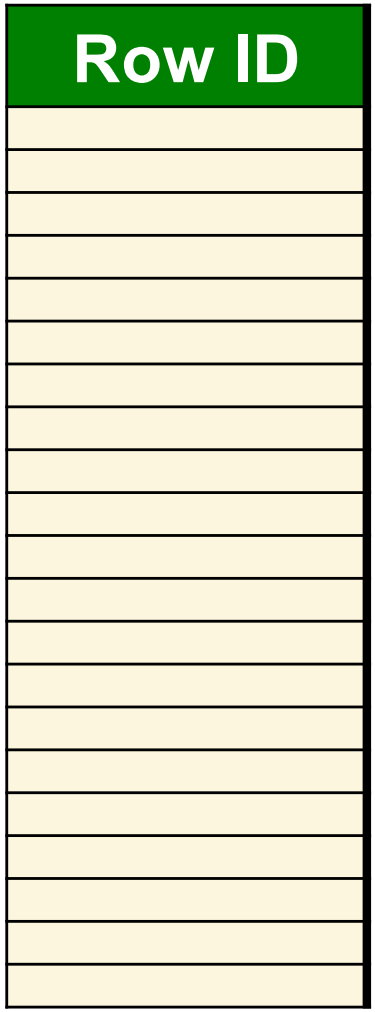


Create table

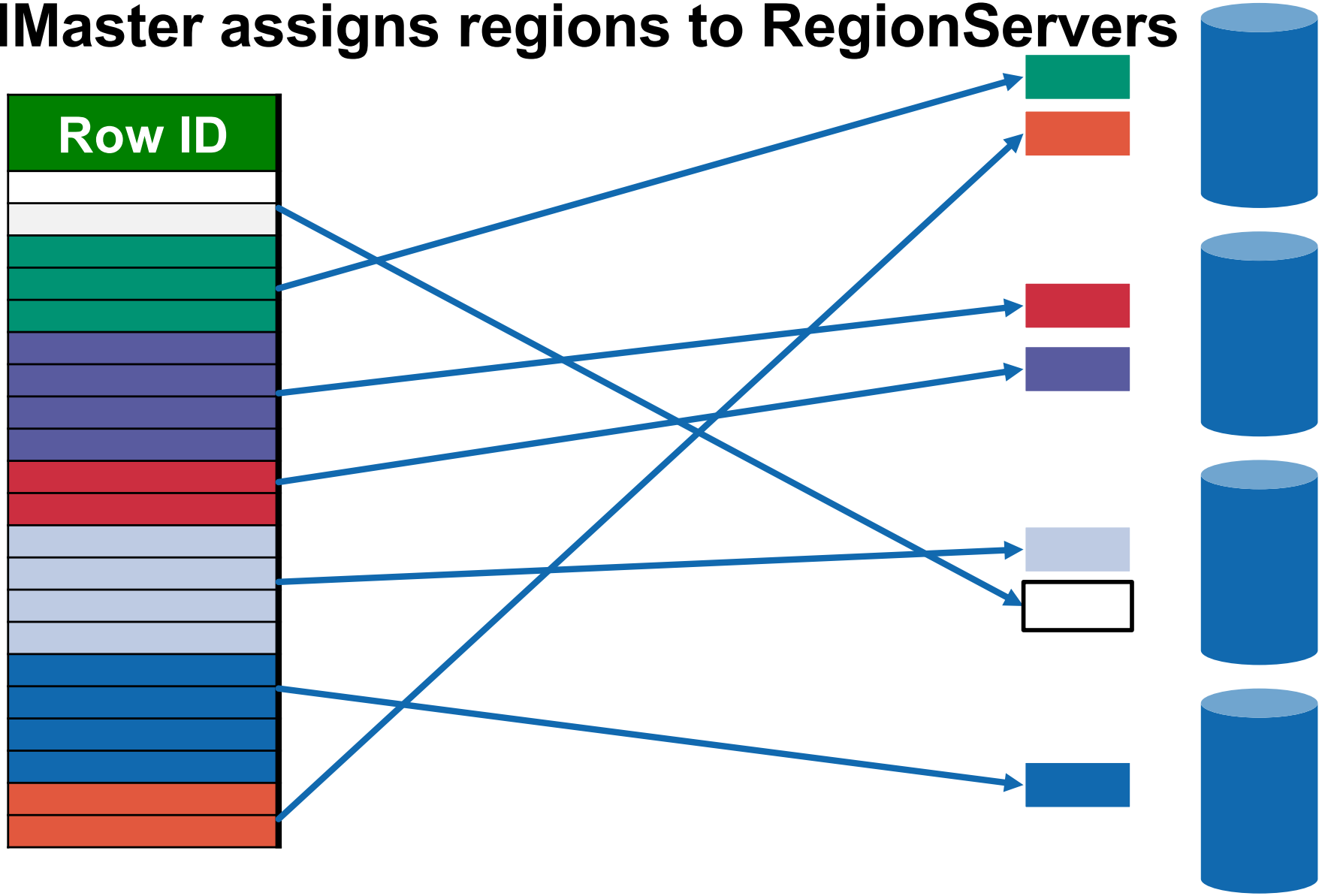


Delete table

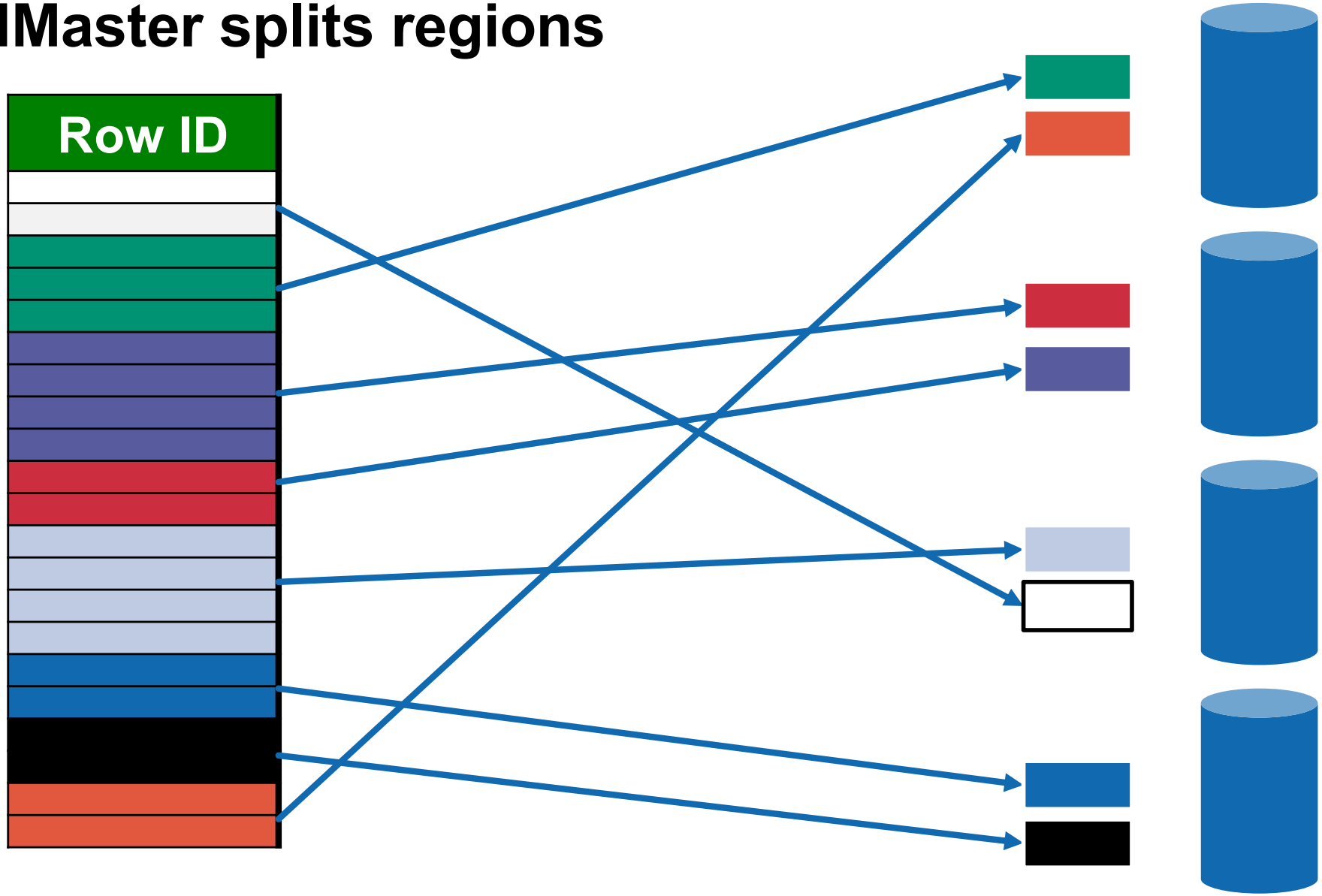
HMaster assigns regions to RegionServers



HMaster assigns regions to RegionServers

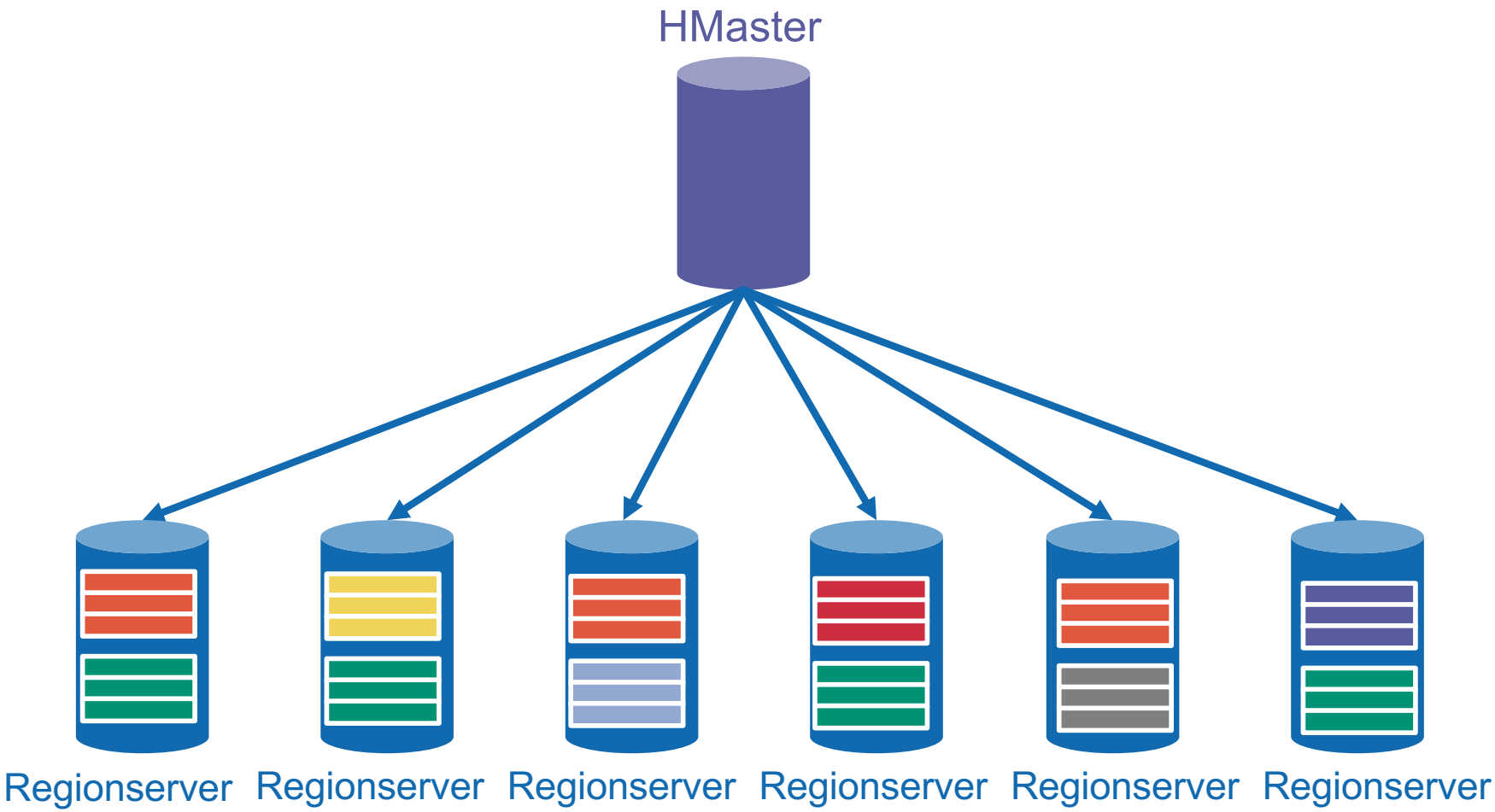


HMaster splits regions

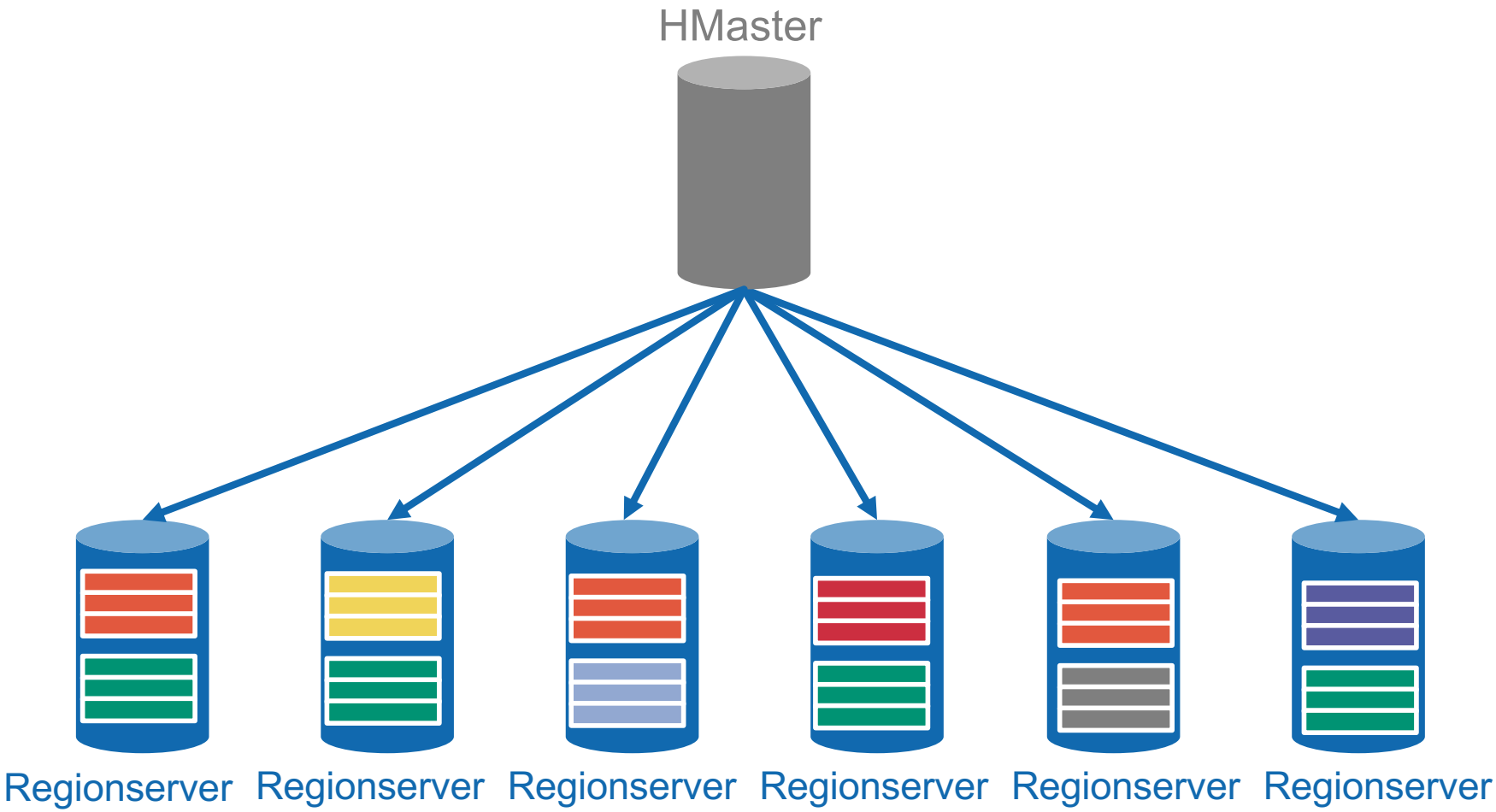


HMaster handles Regionserver failovers

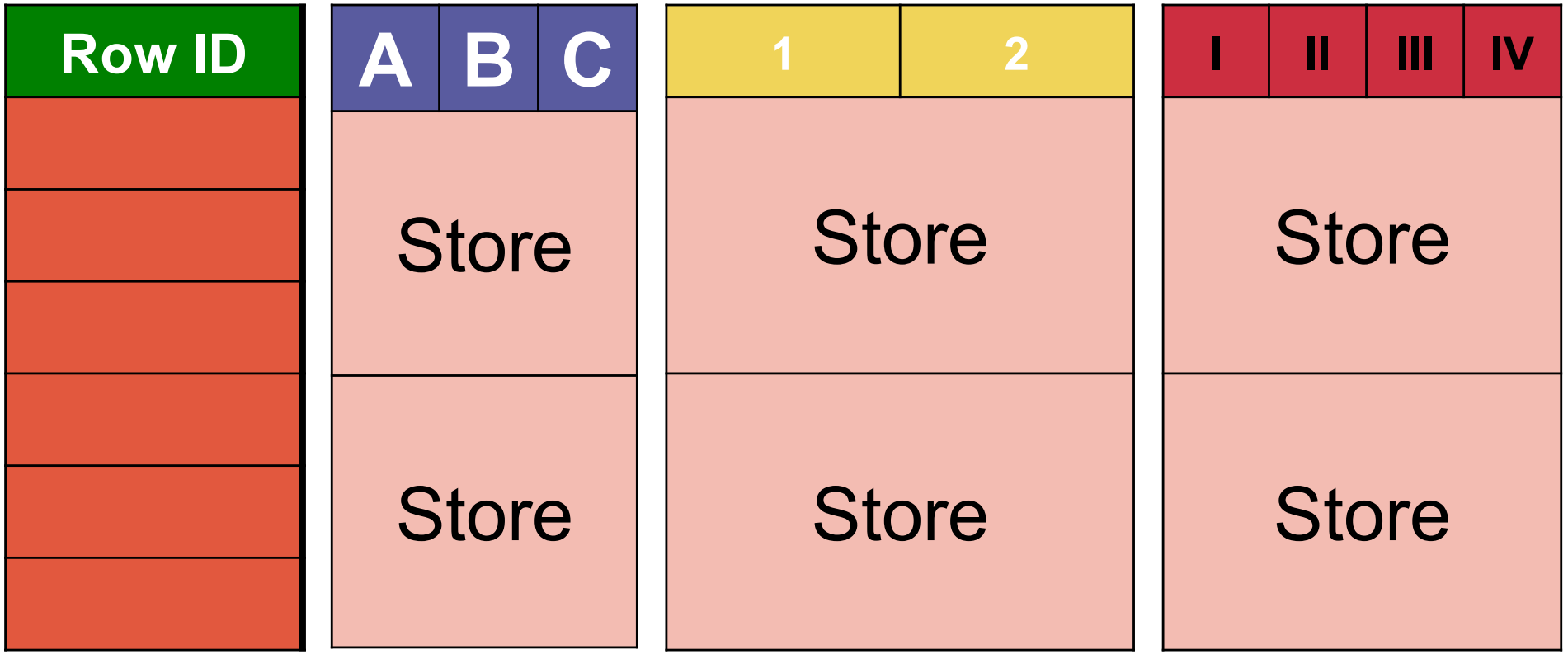
Architecture



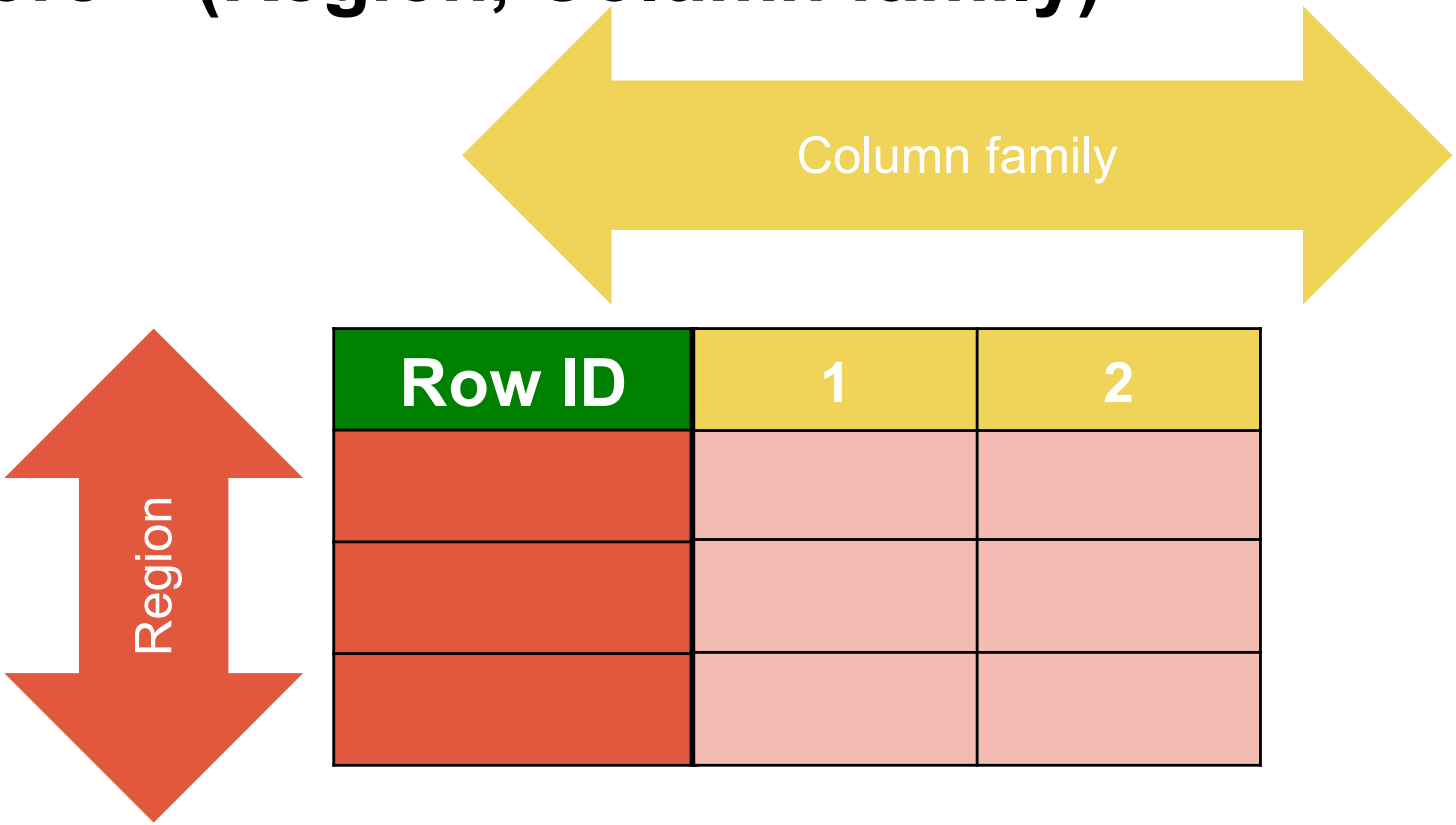
Regionserver



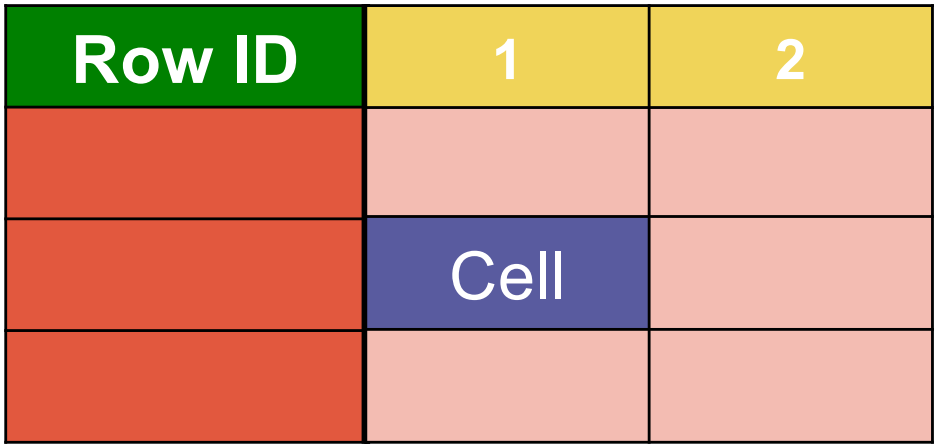
Physical storage



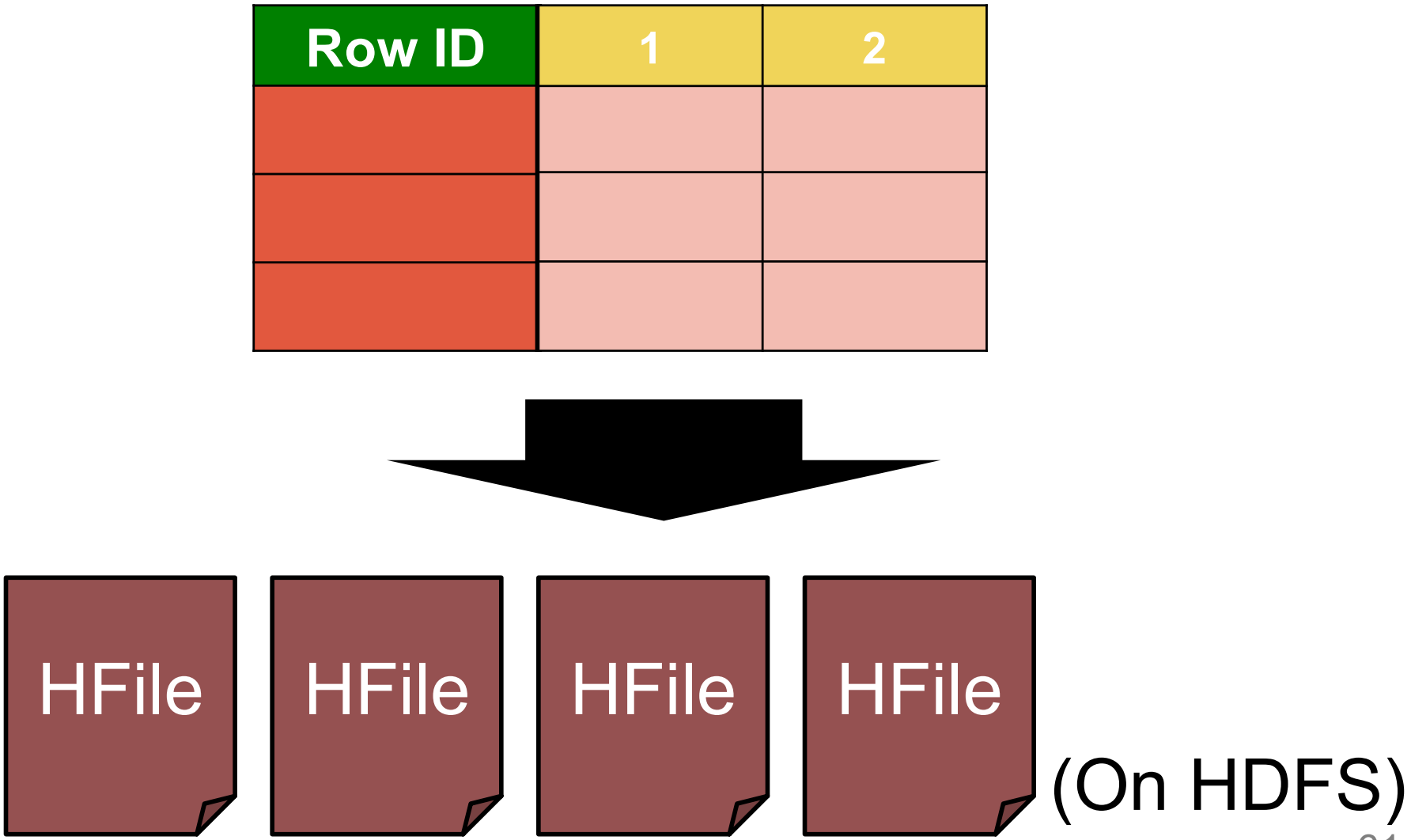
Store = (Region, Column family)



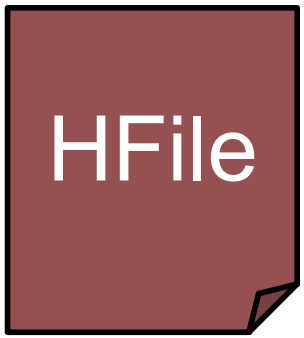
Store = column family



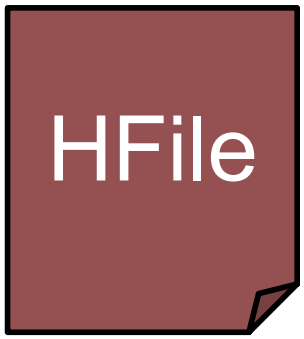
Store = column family



HFile

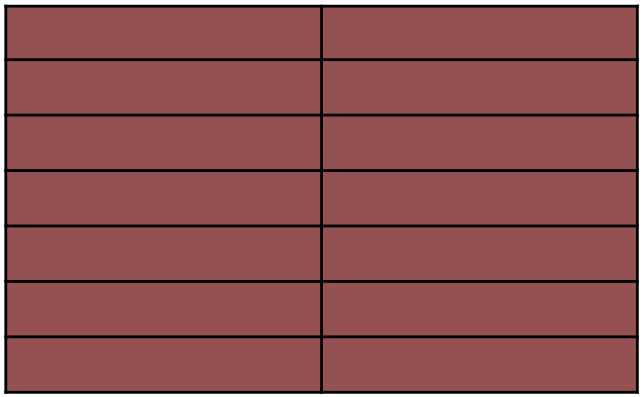


HFile



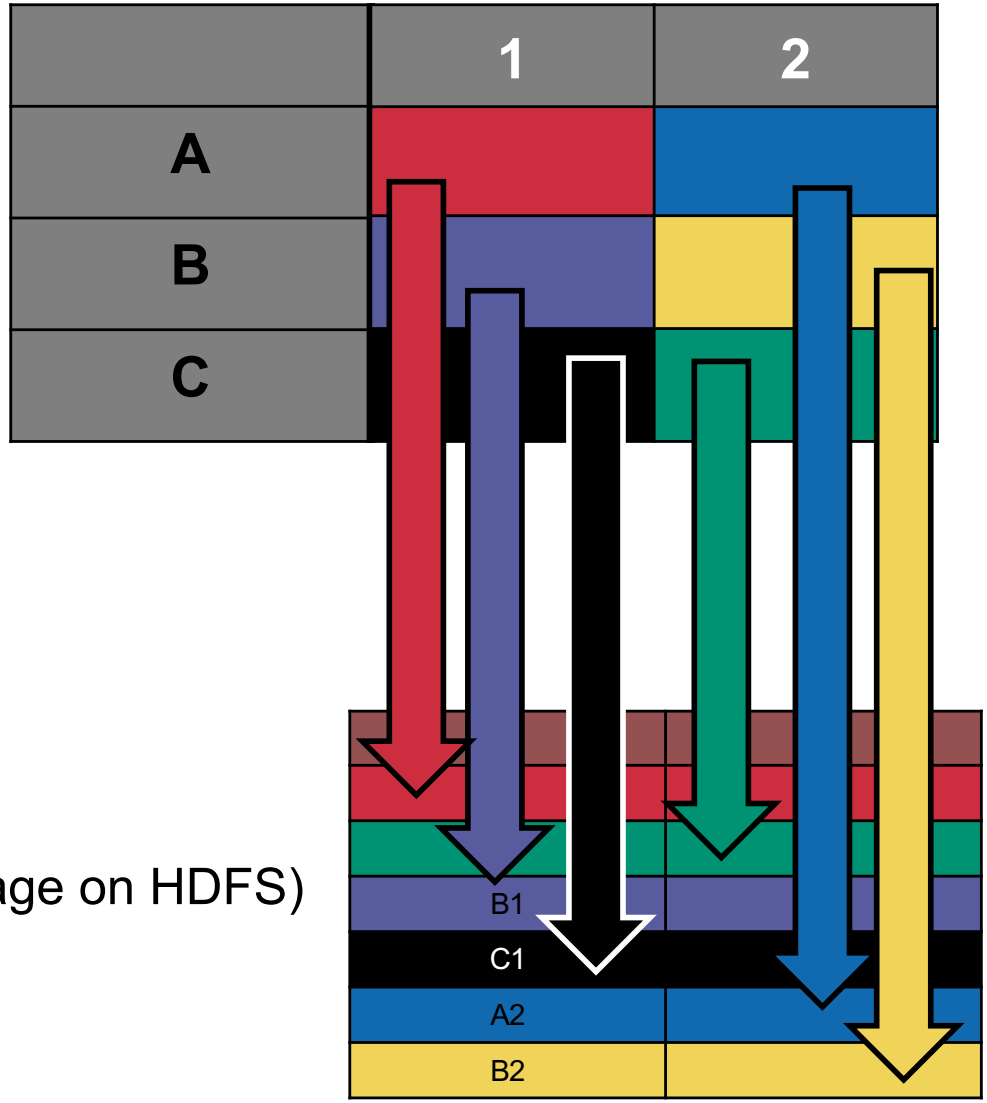
That's actually a sorted list of key-value pairs

Google



HFile

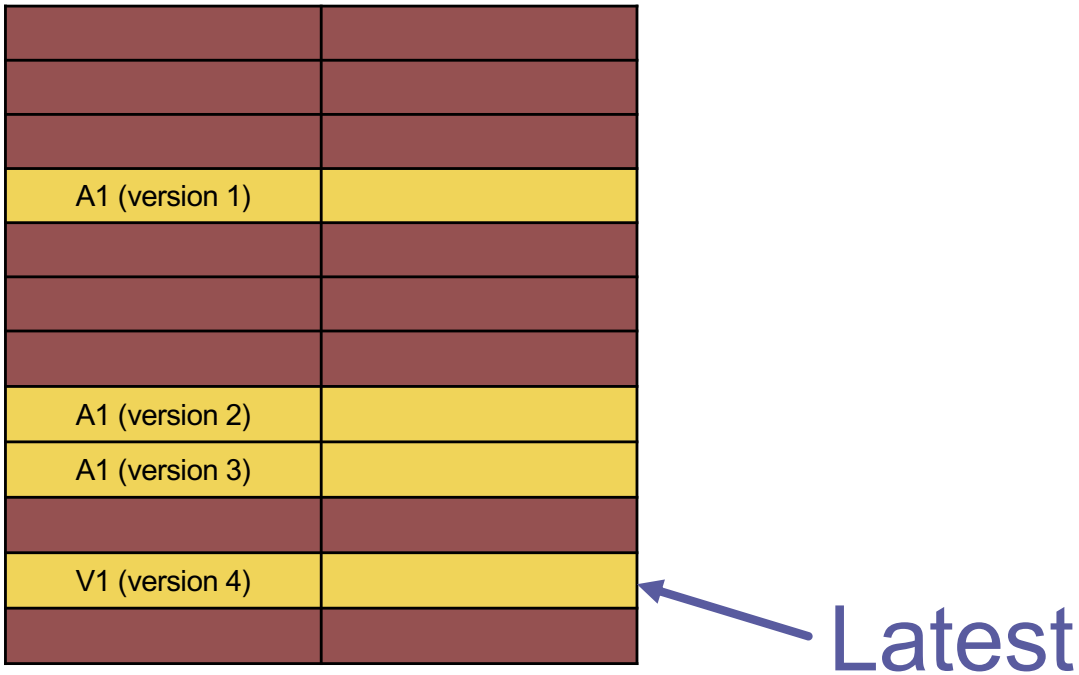
Logical table



HFile (physical storage on HDFS)

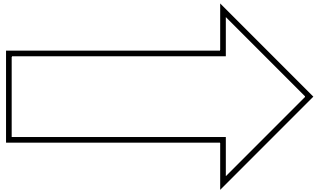
Versioning

Different versions of same cell



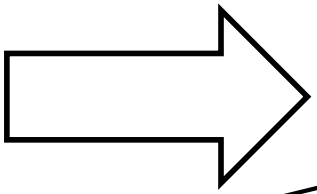
HFile: KeyValue

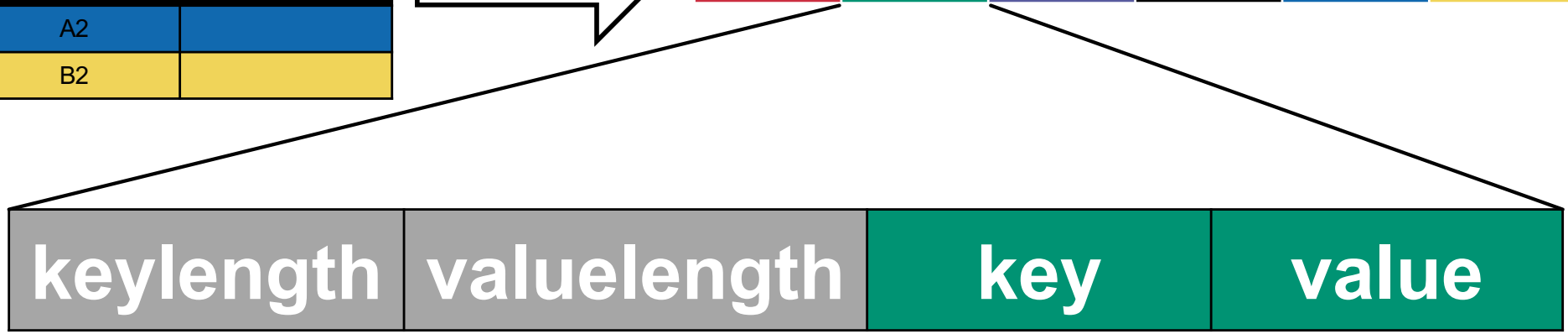
Key	Value
A1	
C2	
B1	
C1	
A2	
B2	



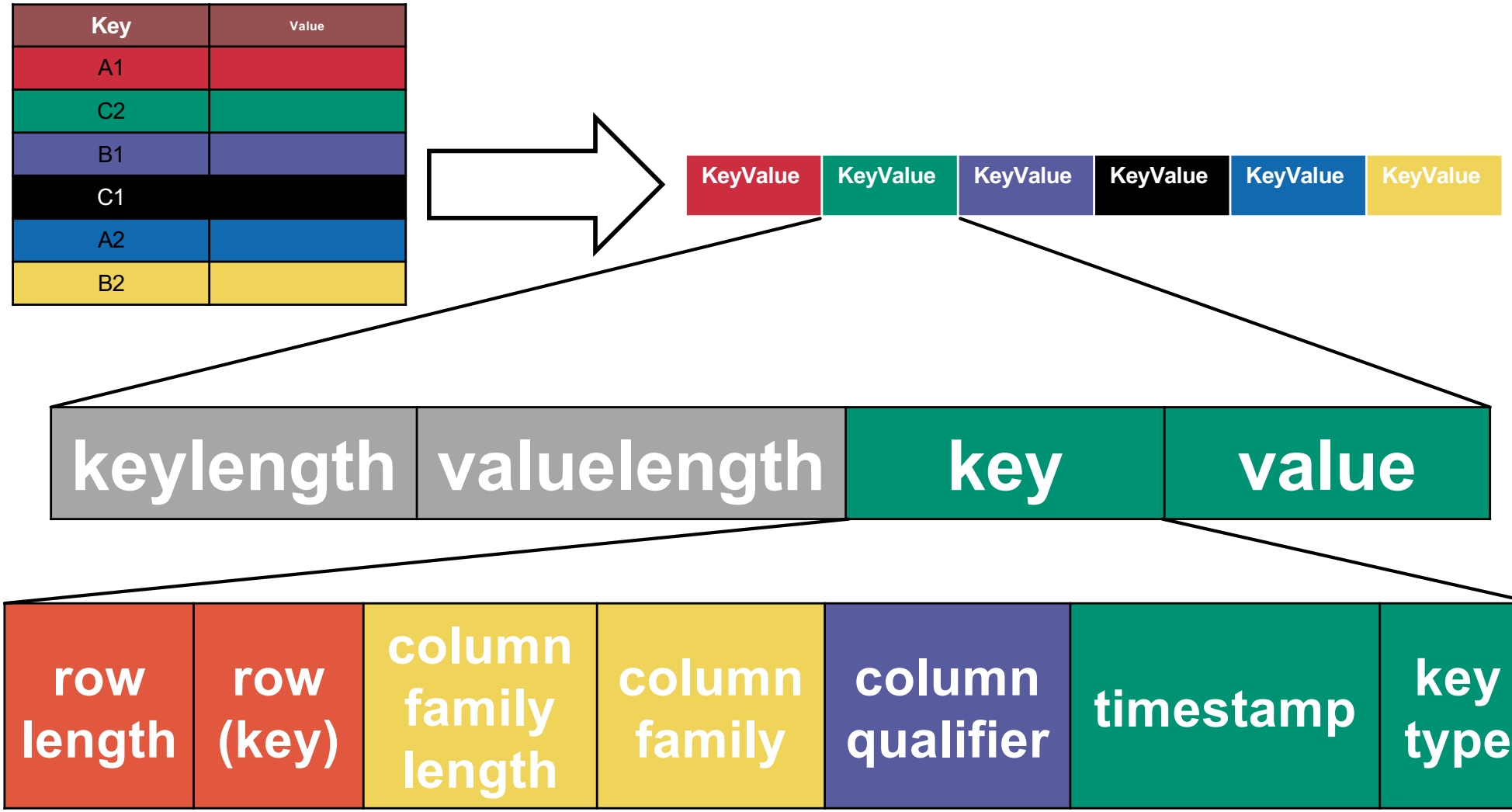
HFile: KeyValue

Key	Value
A1	
C2	
B1	
C1	
A2	
B2	

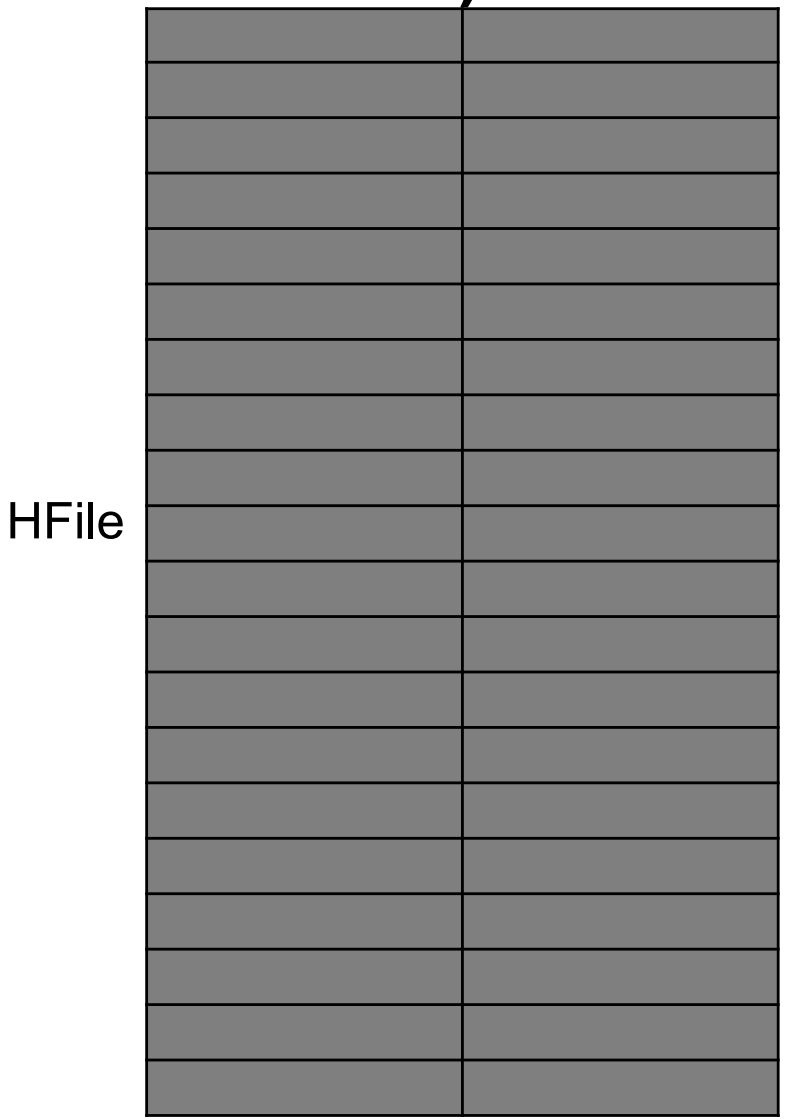




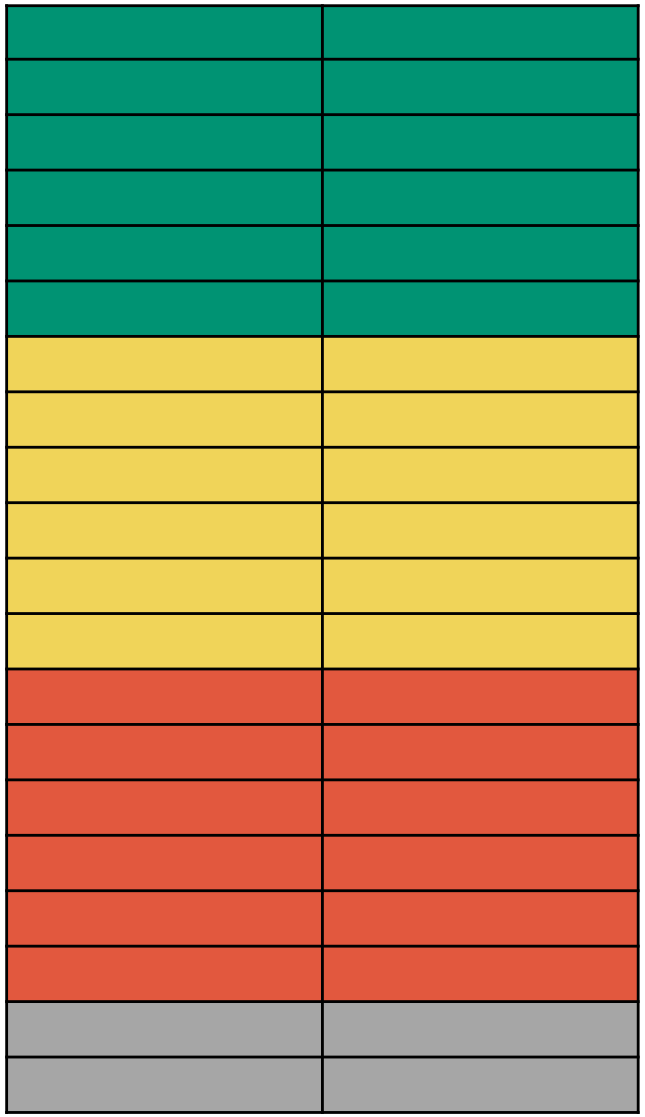
HFile: KeyValue



HBlocks (our own terminology to distinguish from HDFS blocks)

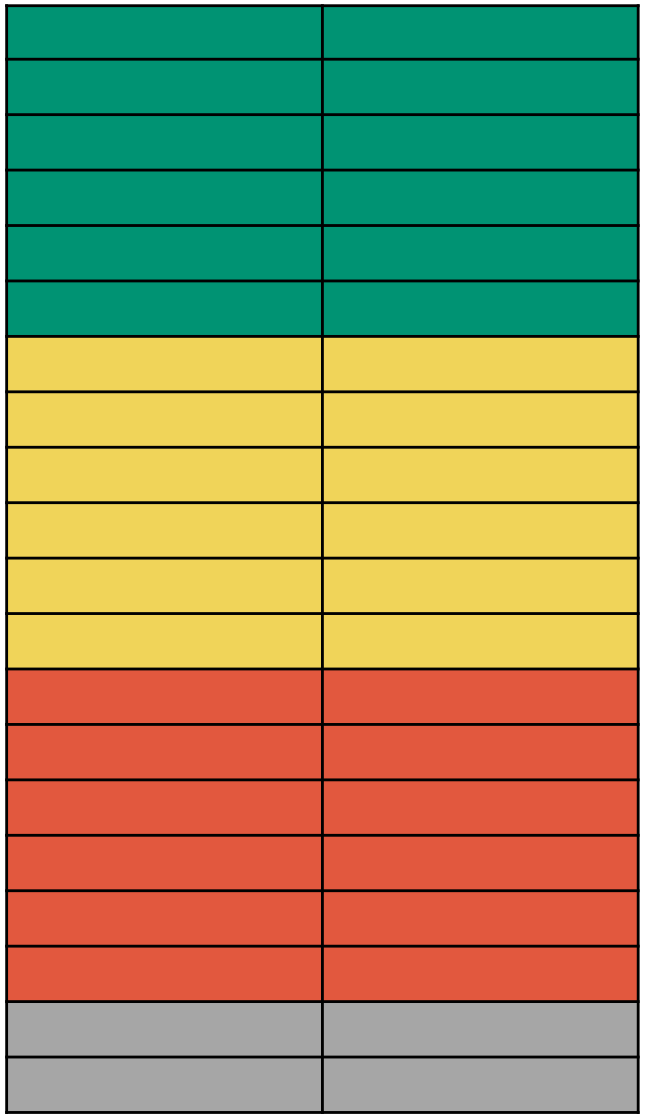


HBlocks



"Quantity" of KeyValues that get read at a time

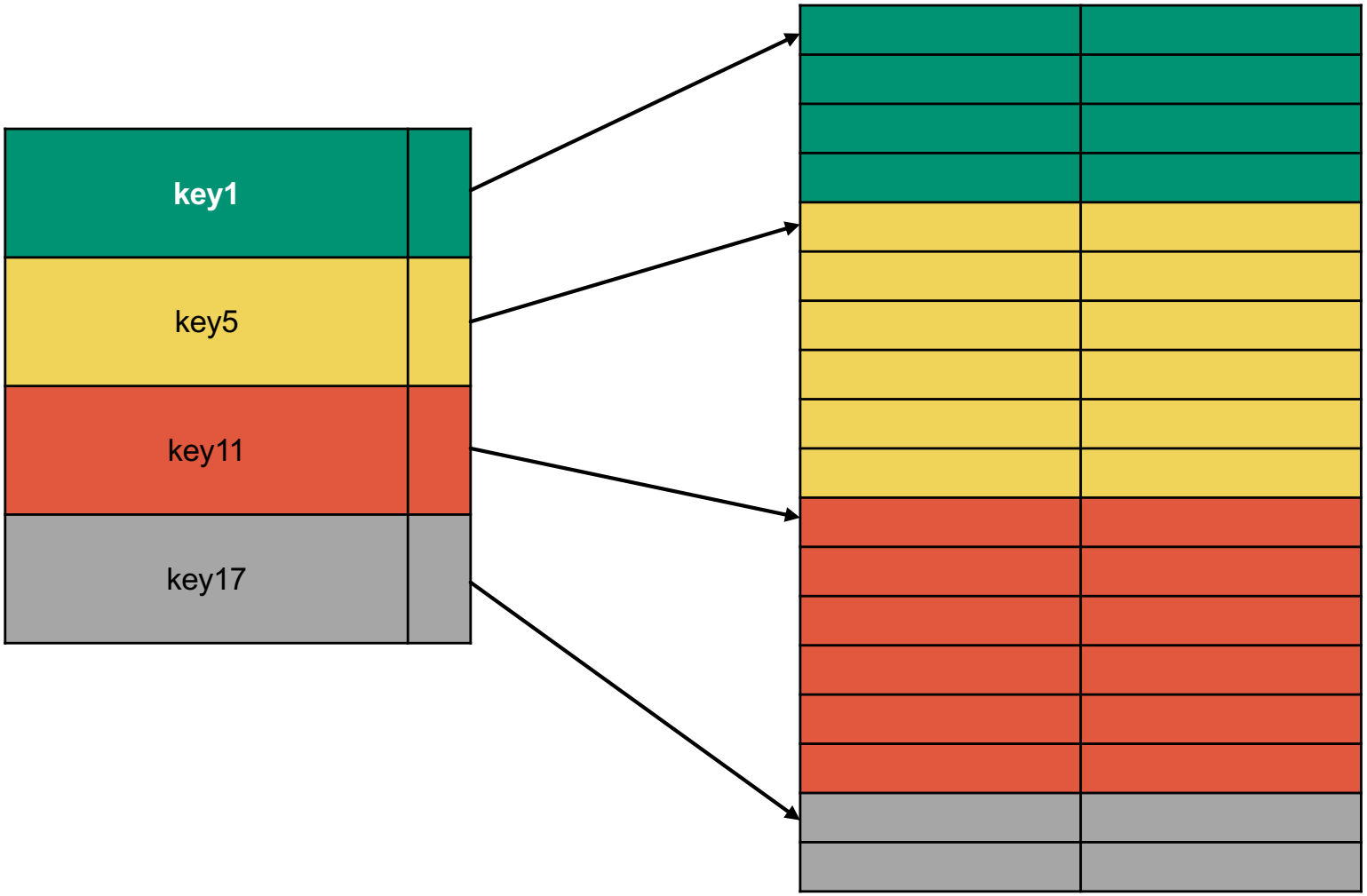
HBlocks



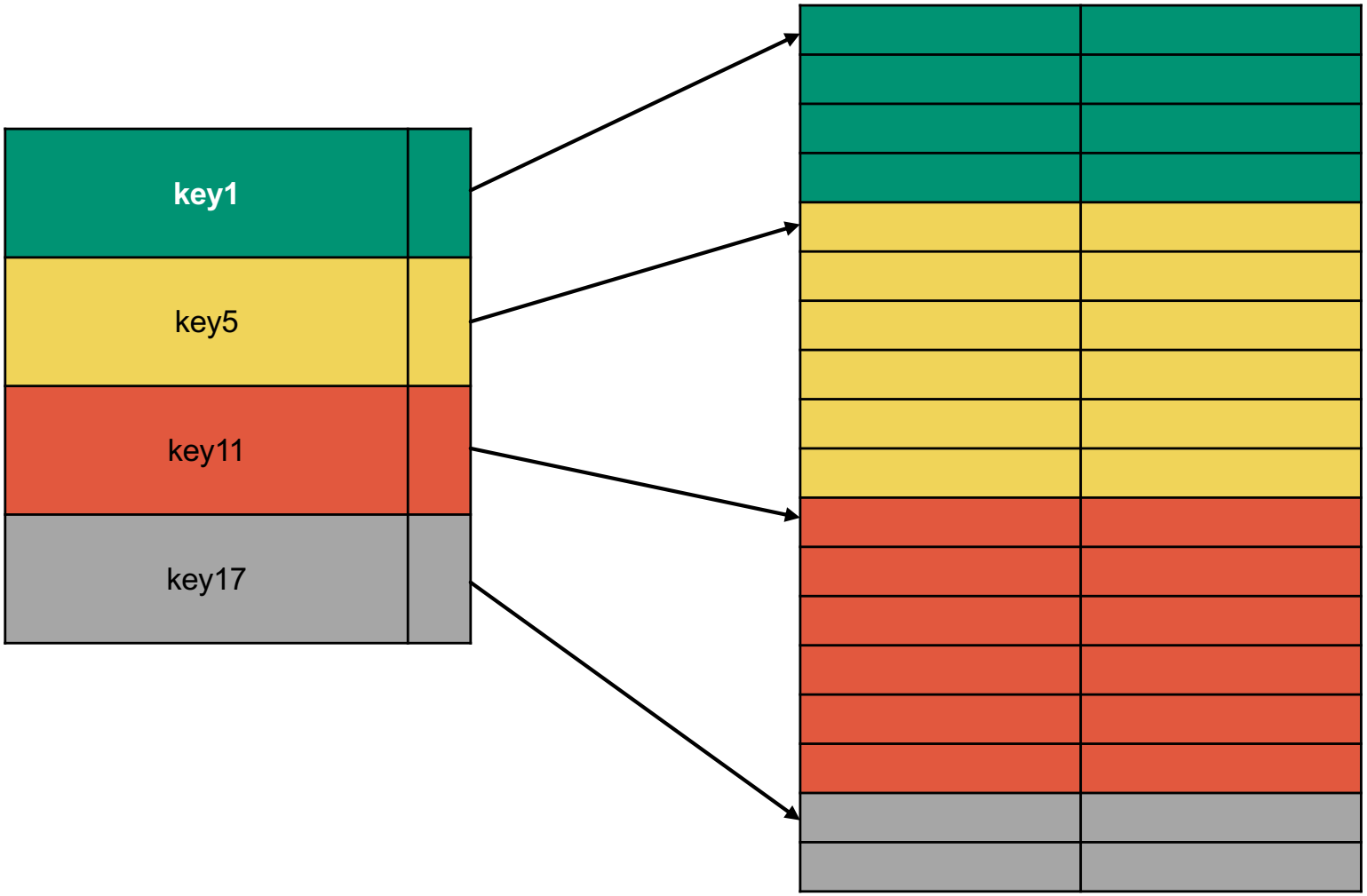
Default

64kb

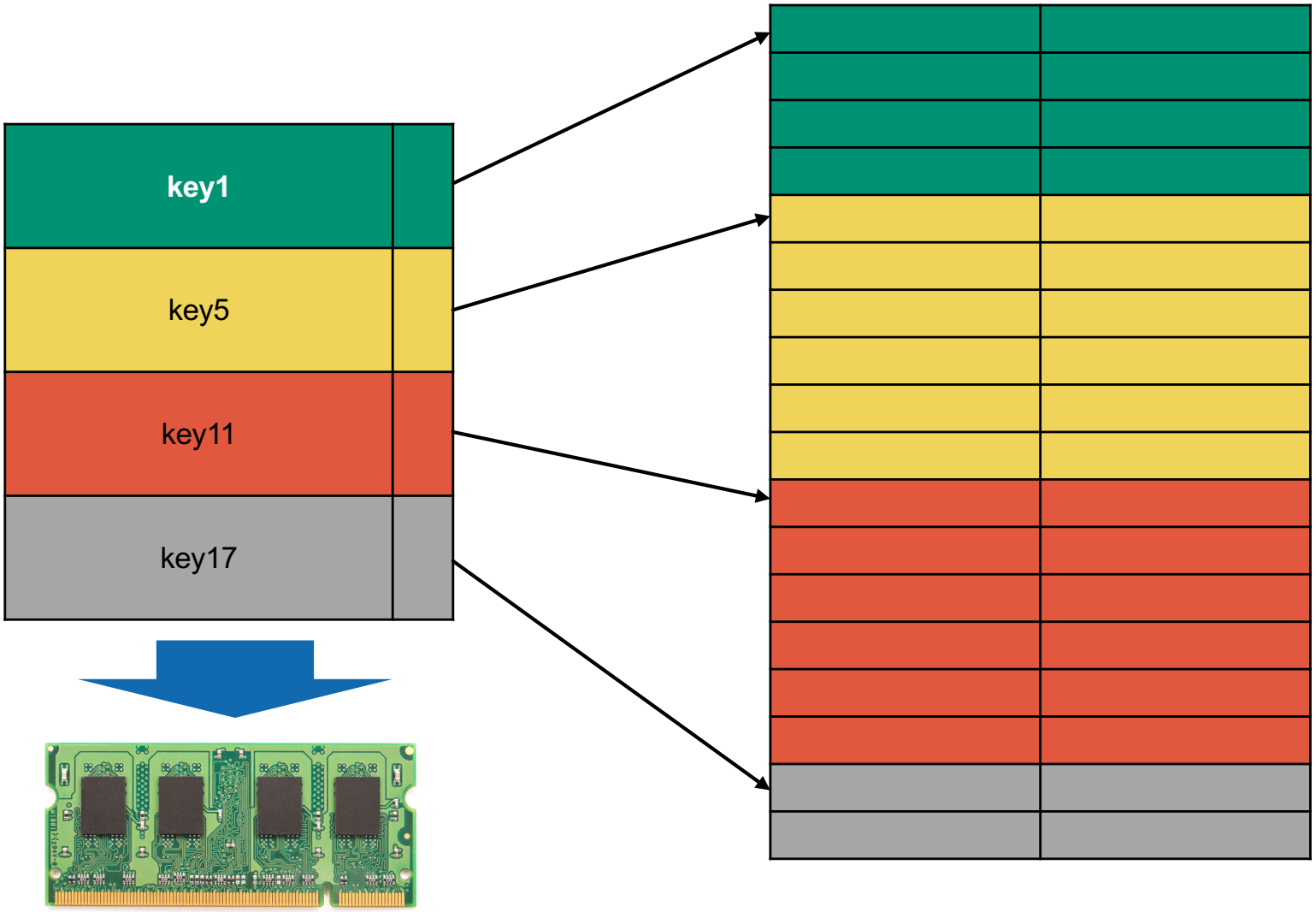
Inside an HFile



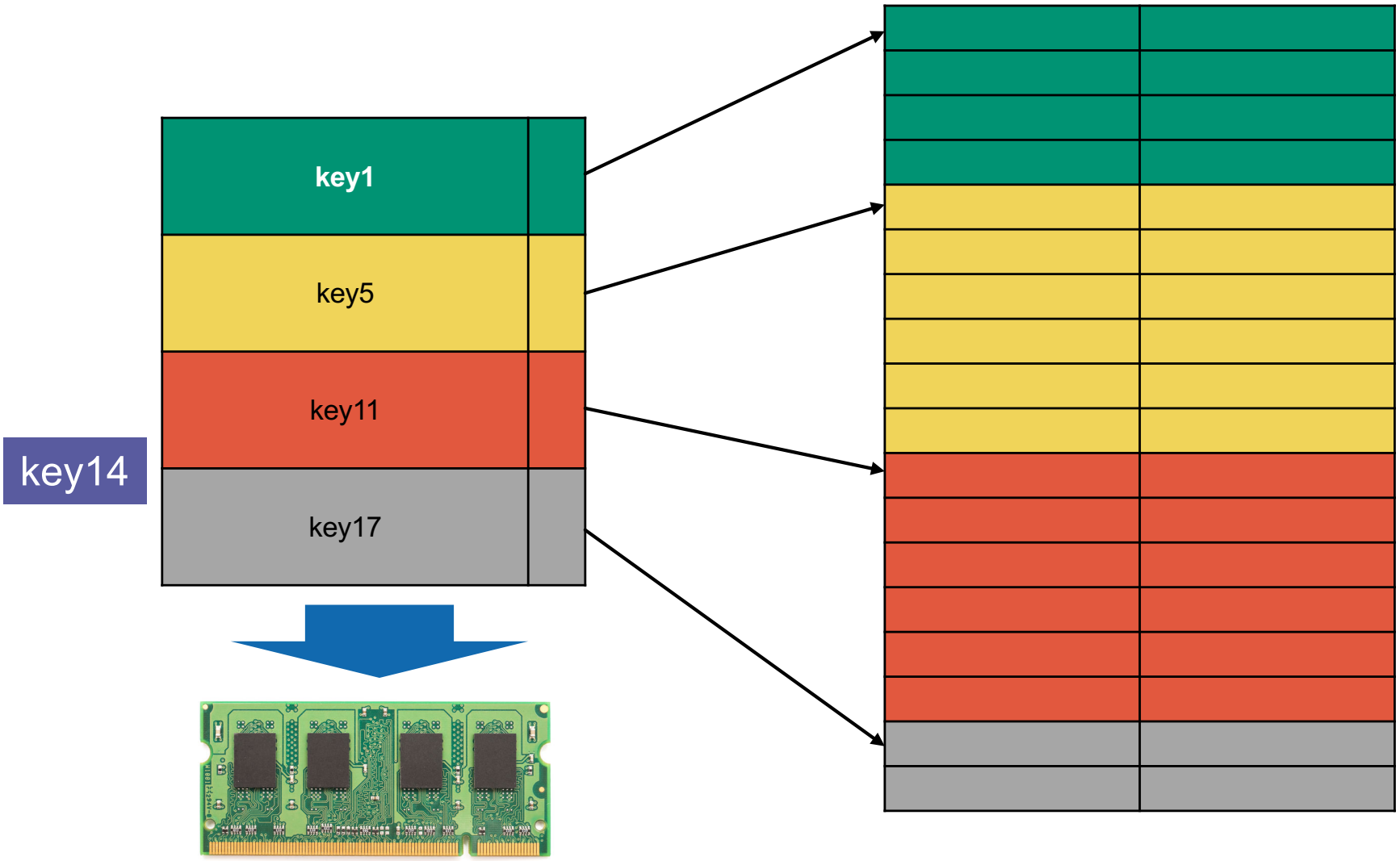
Looking up a key



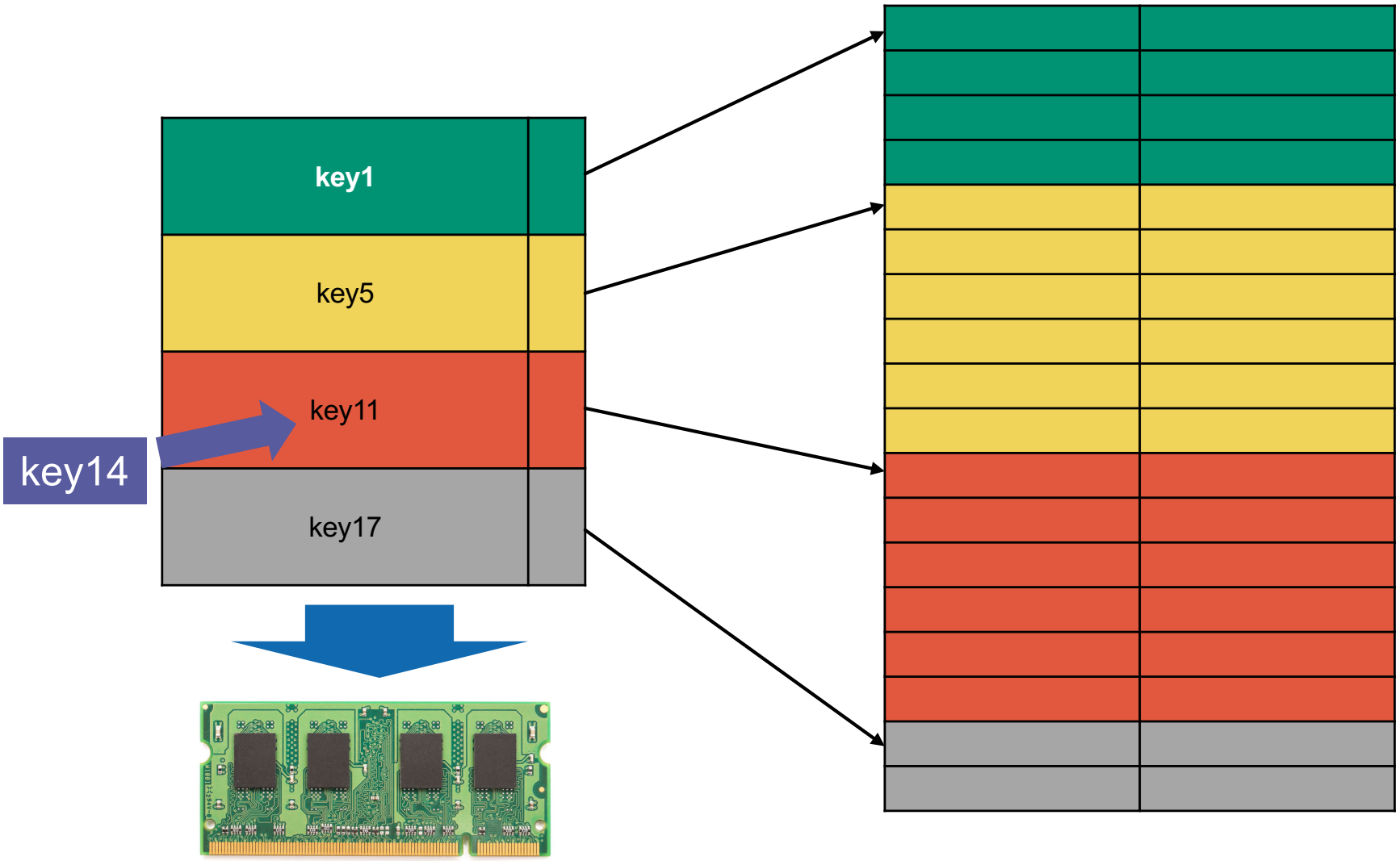
Looking up a key



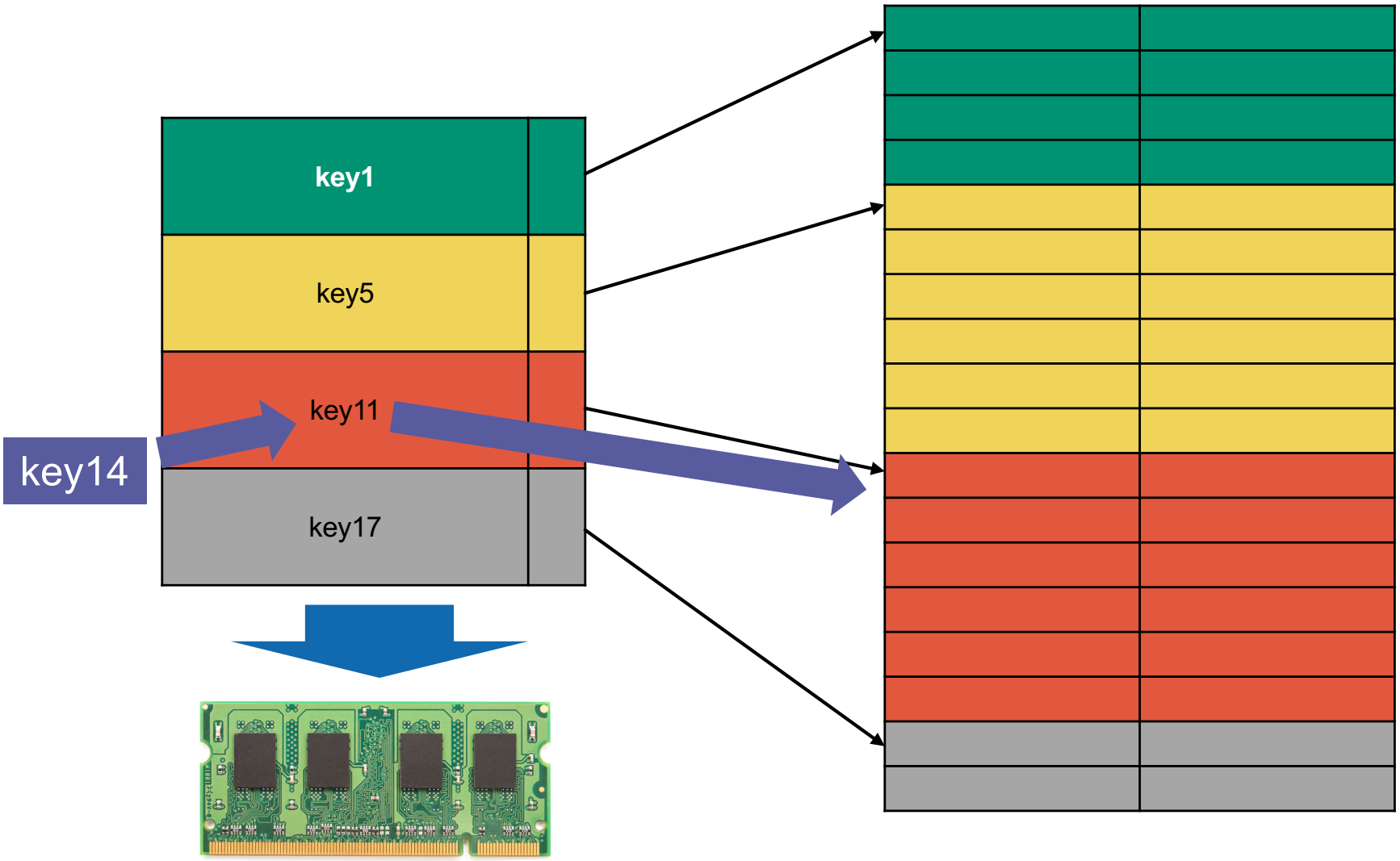
Looking up a key



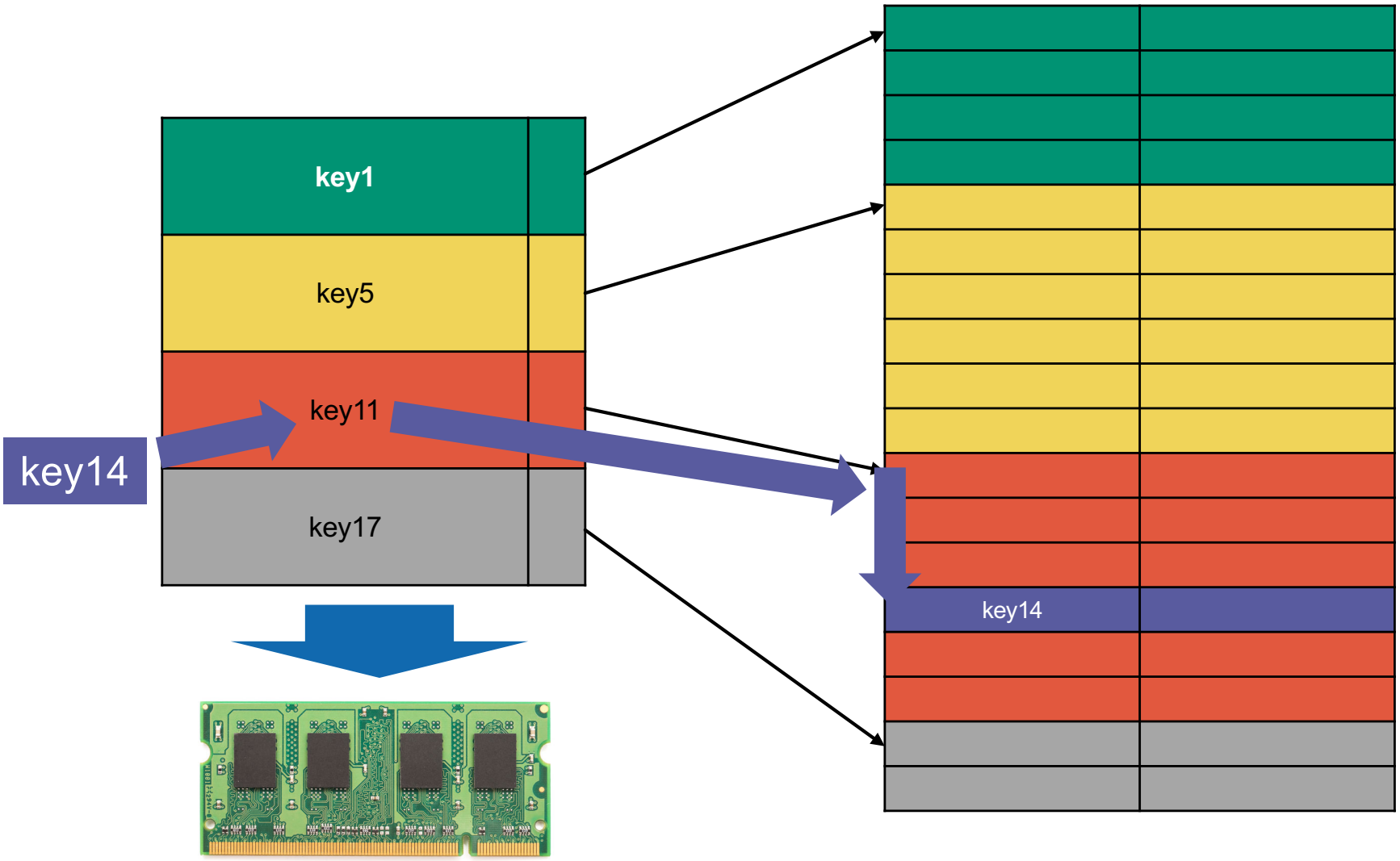
Looking up a key



Looking up a key



Looking up a key

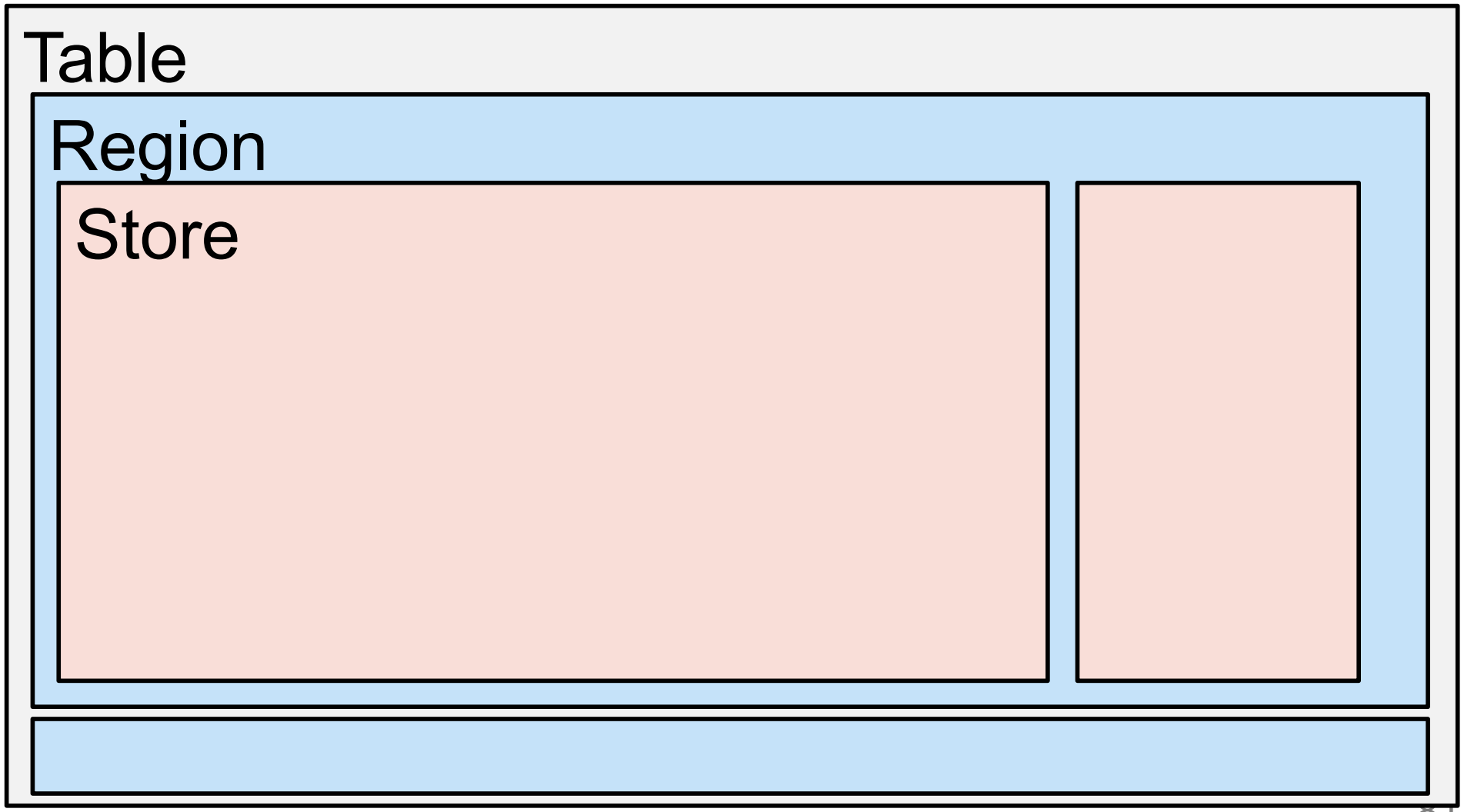


Levels of physical storage

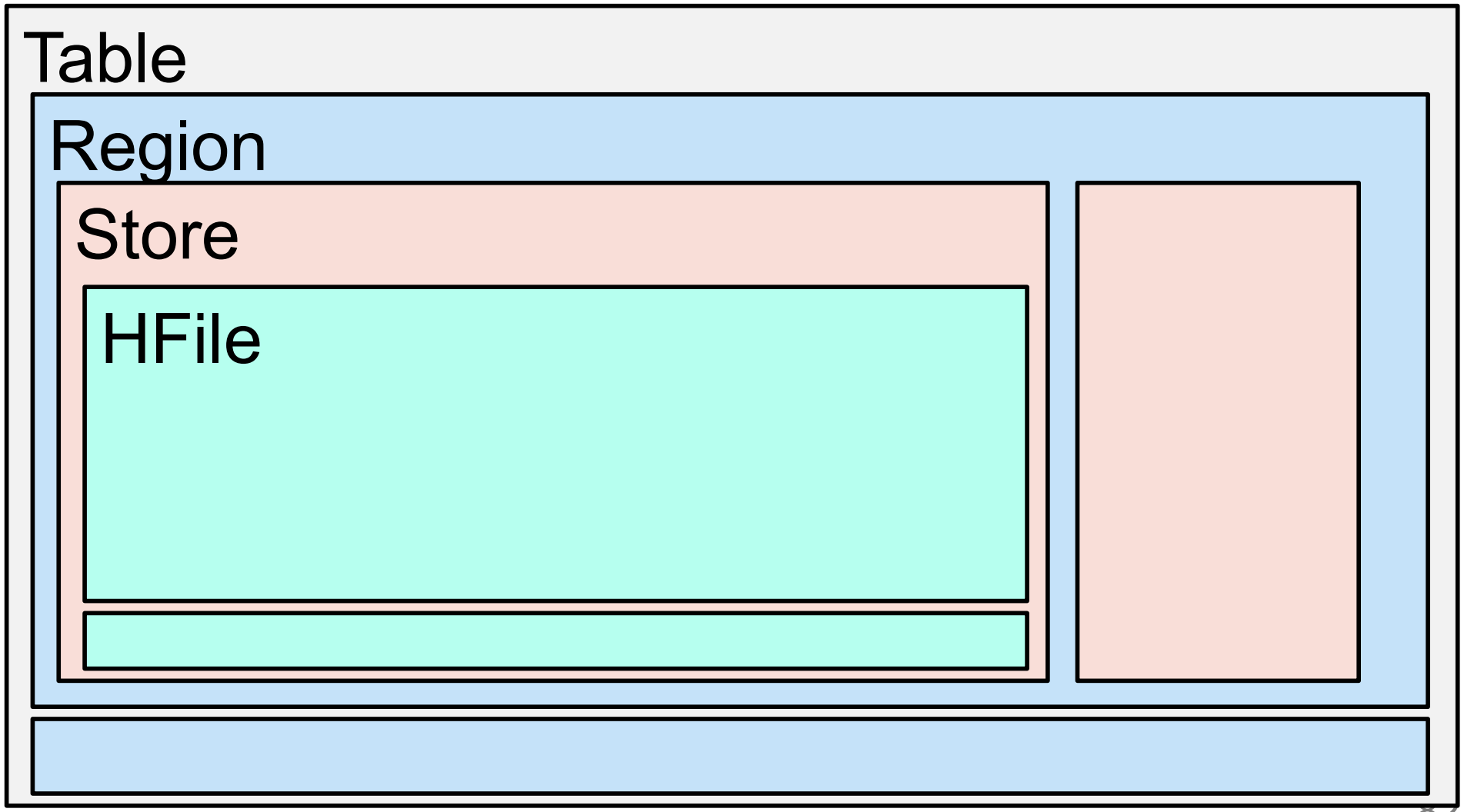
Table

Levels of physical storage

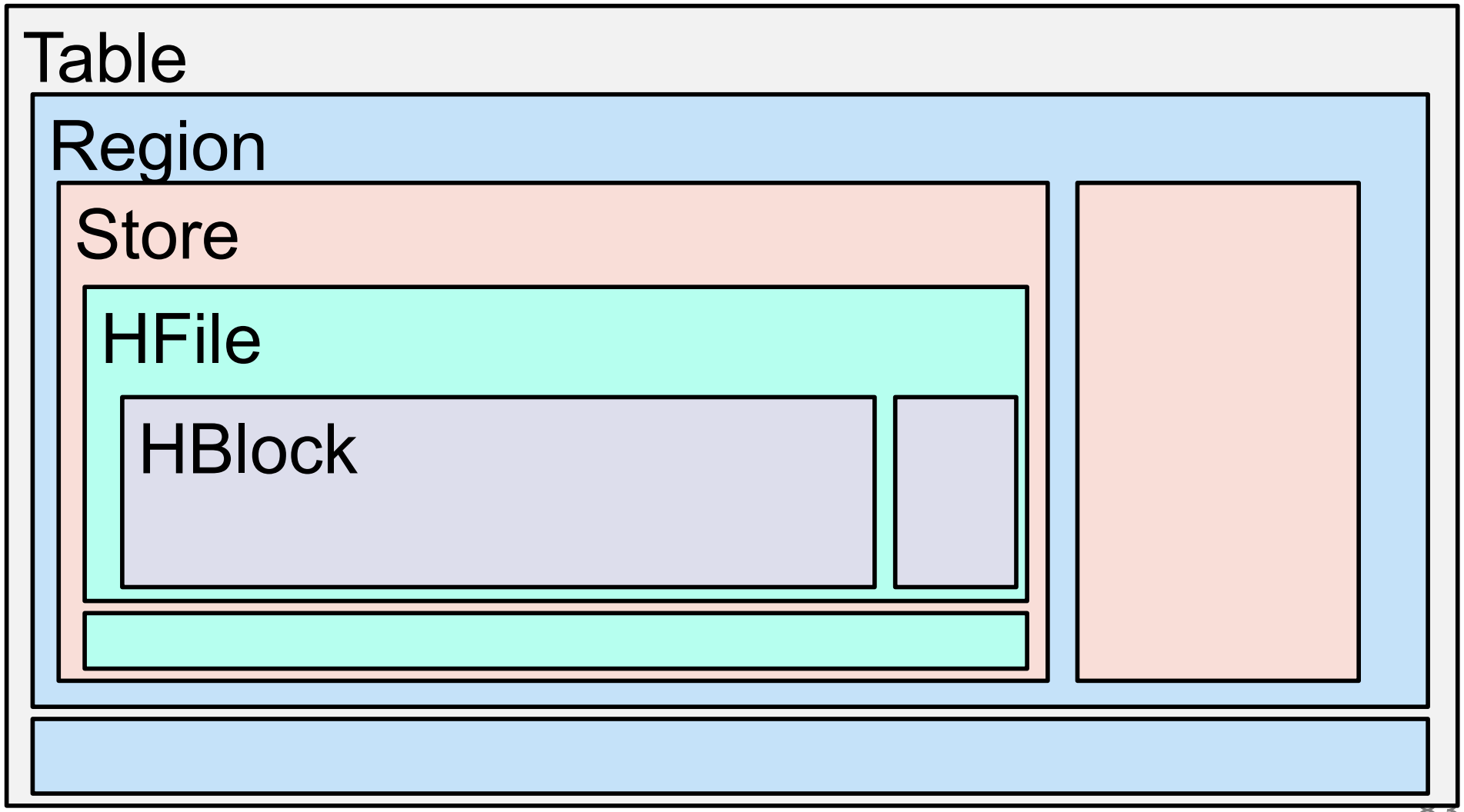
Levels of physical storage



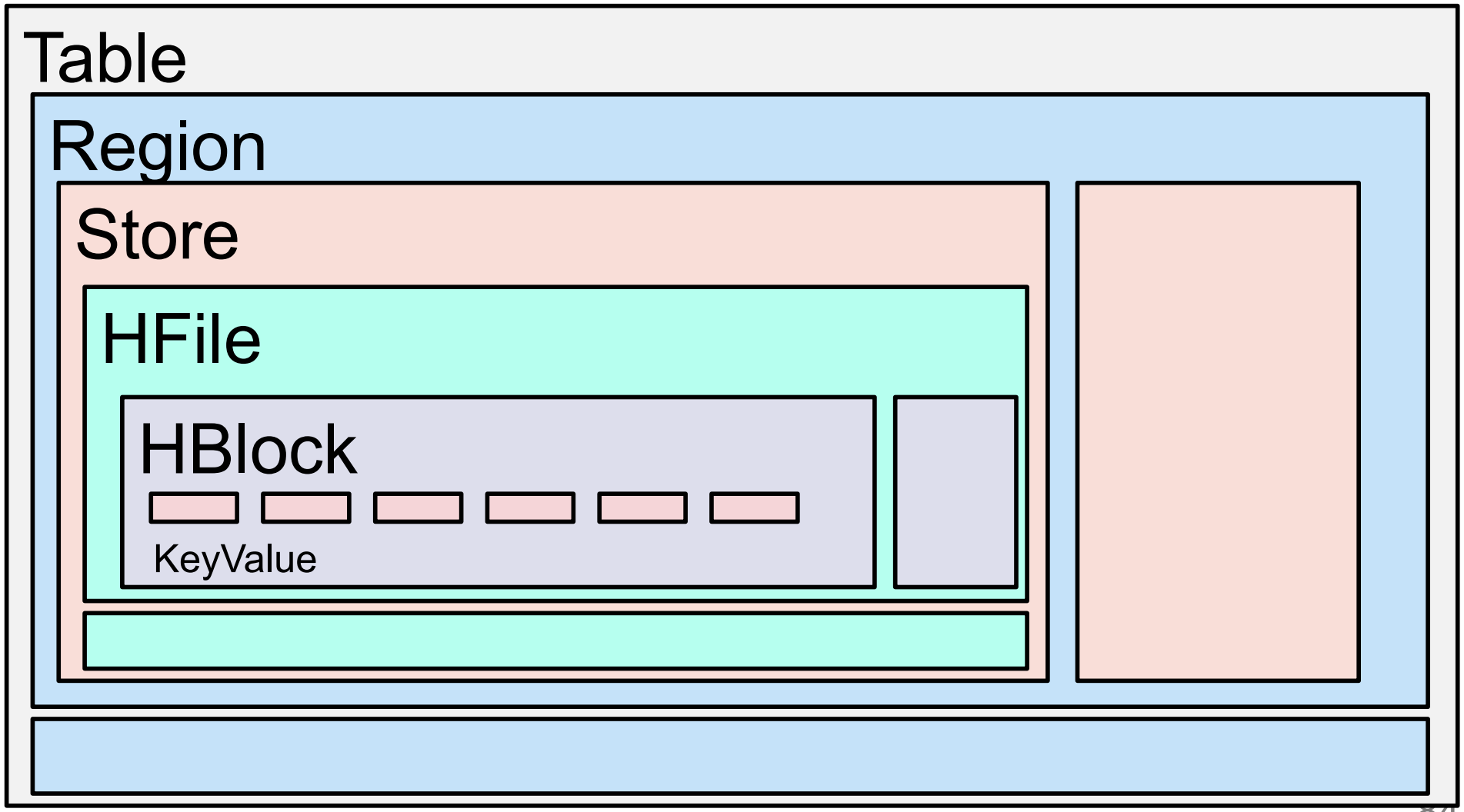
Levels of physical storage



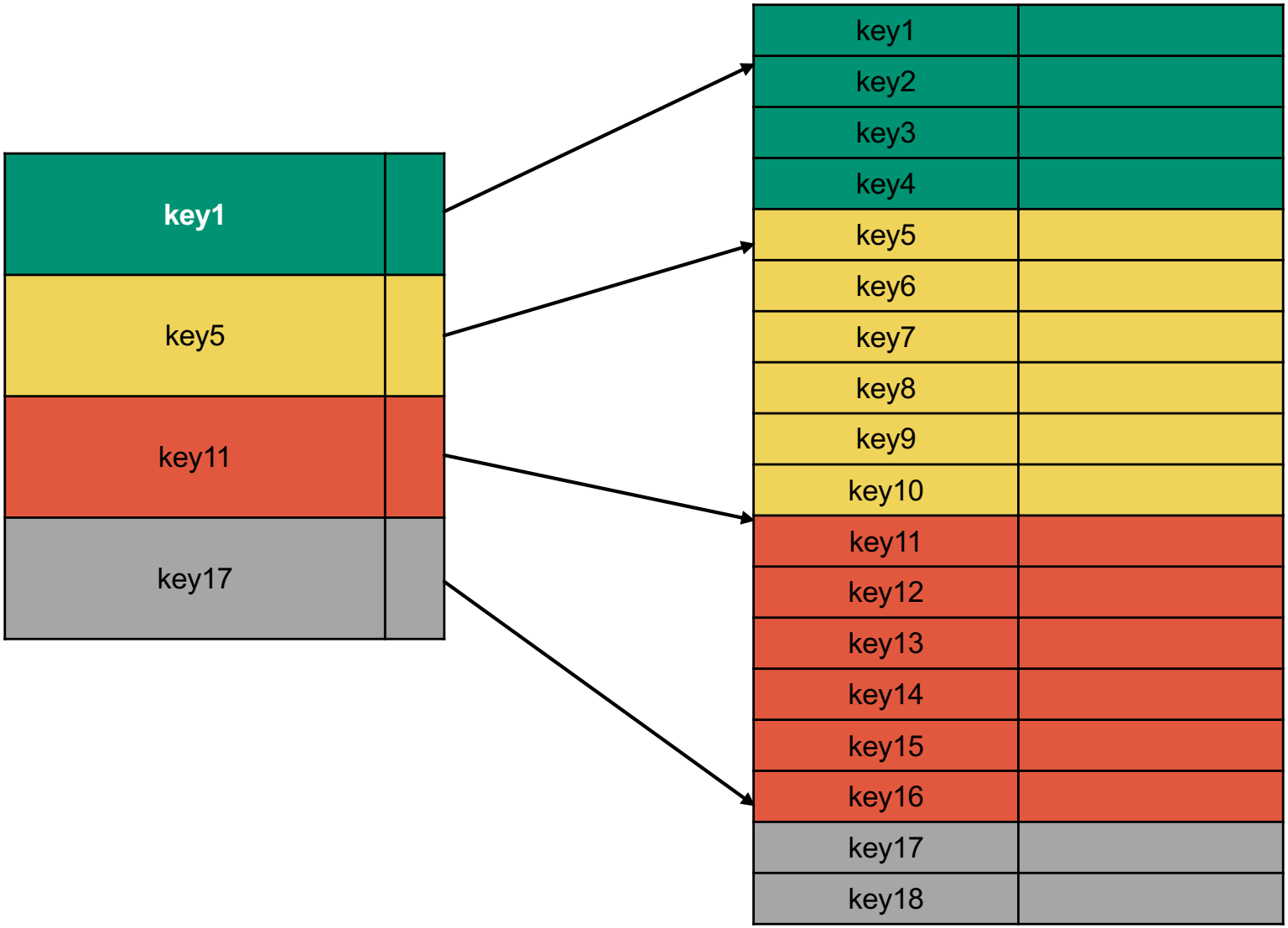
Levels of physical storage



Levels of physical storage



Problem


key1
key5
key11
key17
key1
key2
key3
key4
key5
key6
key7
key8
key9
key10
key11
key12
key13
key14
key15
key16
key17
key18

Problem

We can only write key-values in **sorted** order (append)

key1	
key2	
key3	
key4	
key5	
key6	
key7	
key8	
key9	
key10	
key11	
key12	
key13	
key14	
key15	
key16	
key17	
key18	

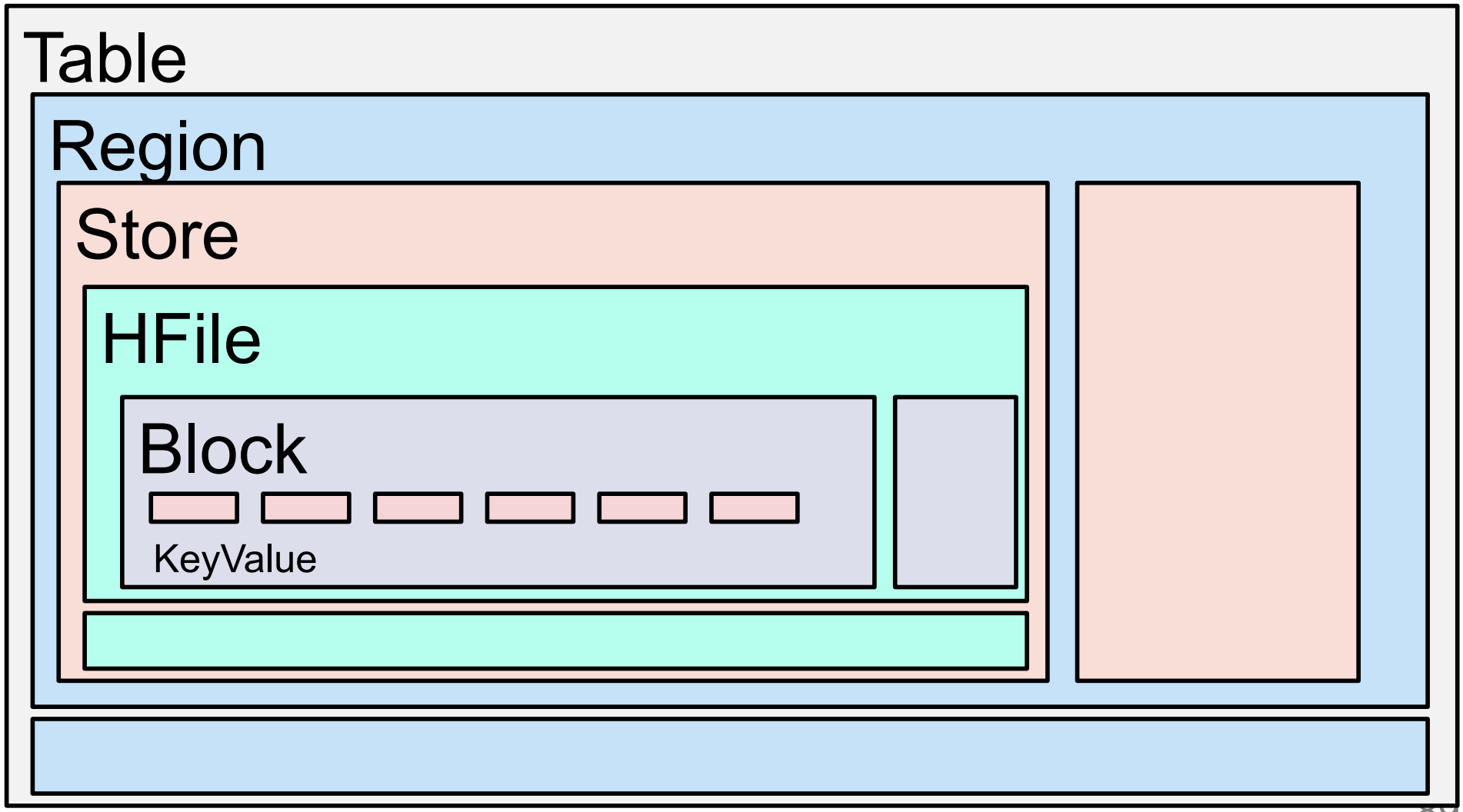
Problem

How can we insert new values ?

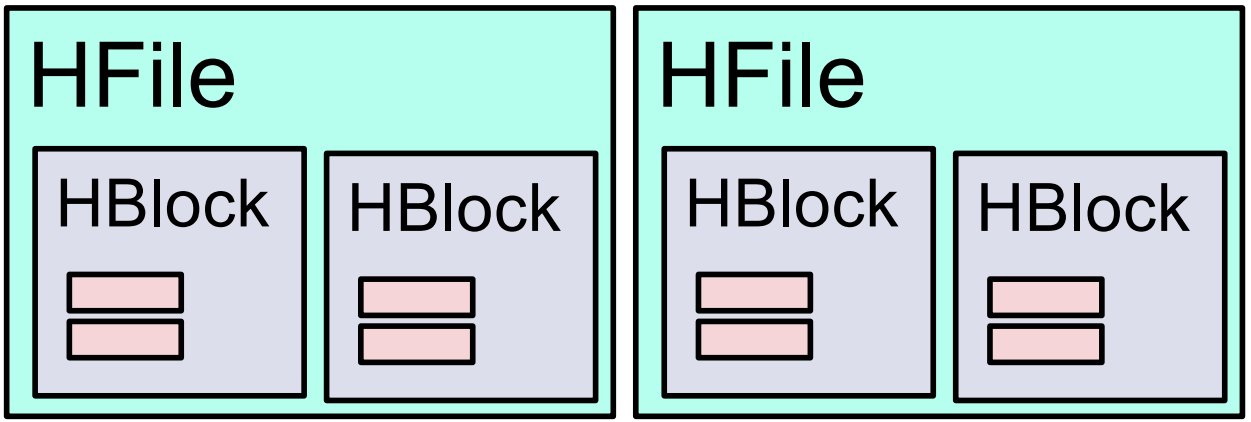
key1	
key2	
key3	
key4	
key5	
key6	
key7	
key8	
key9	
key10	
key11	
key12	
key13	
key14	
key15	
key16	
key17	
key18	

HBase: Writing new cells

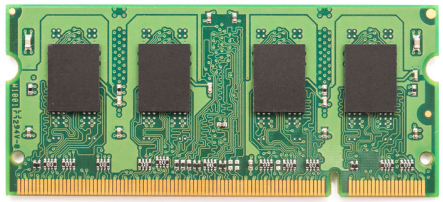
On Disk



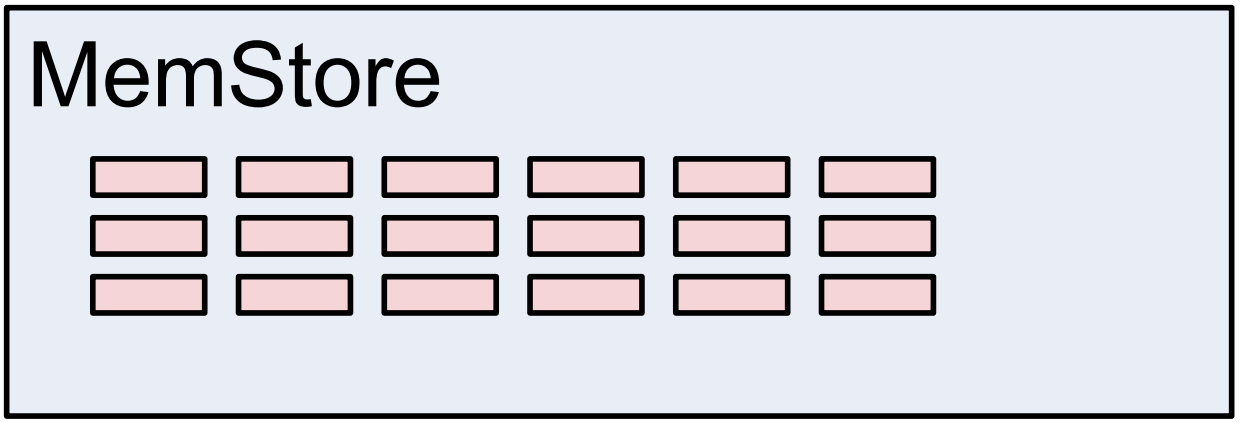
Store

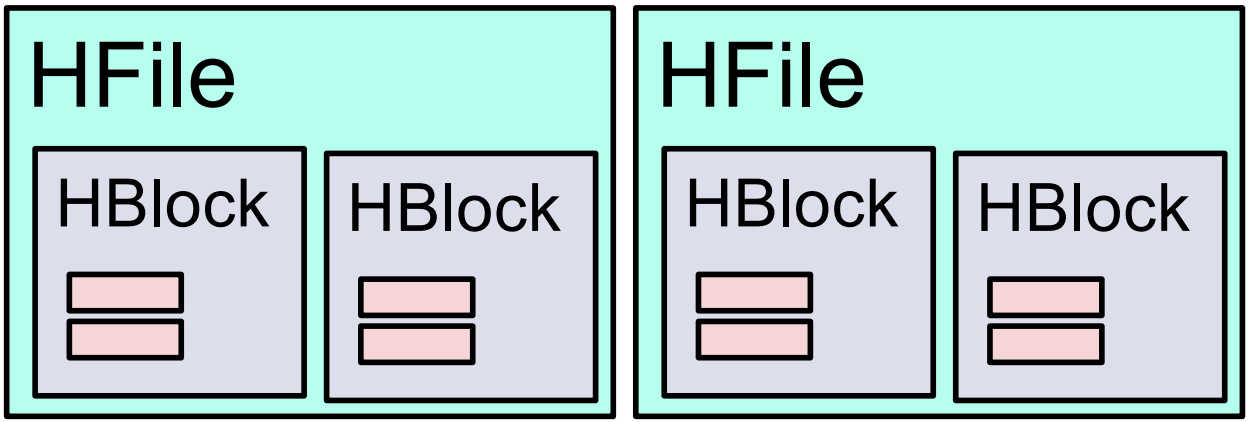


Store

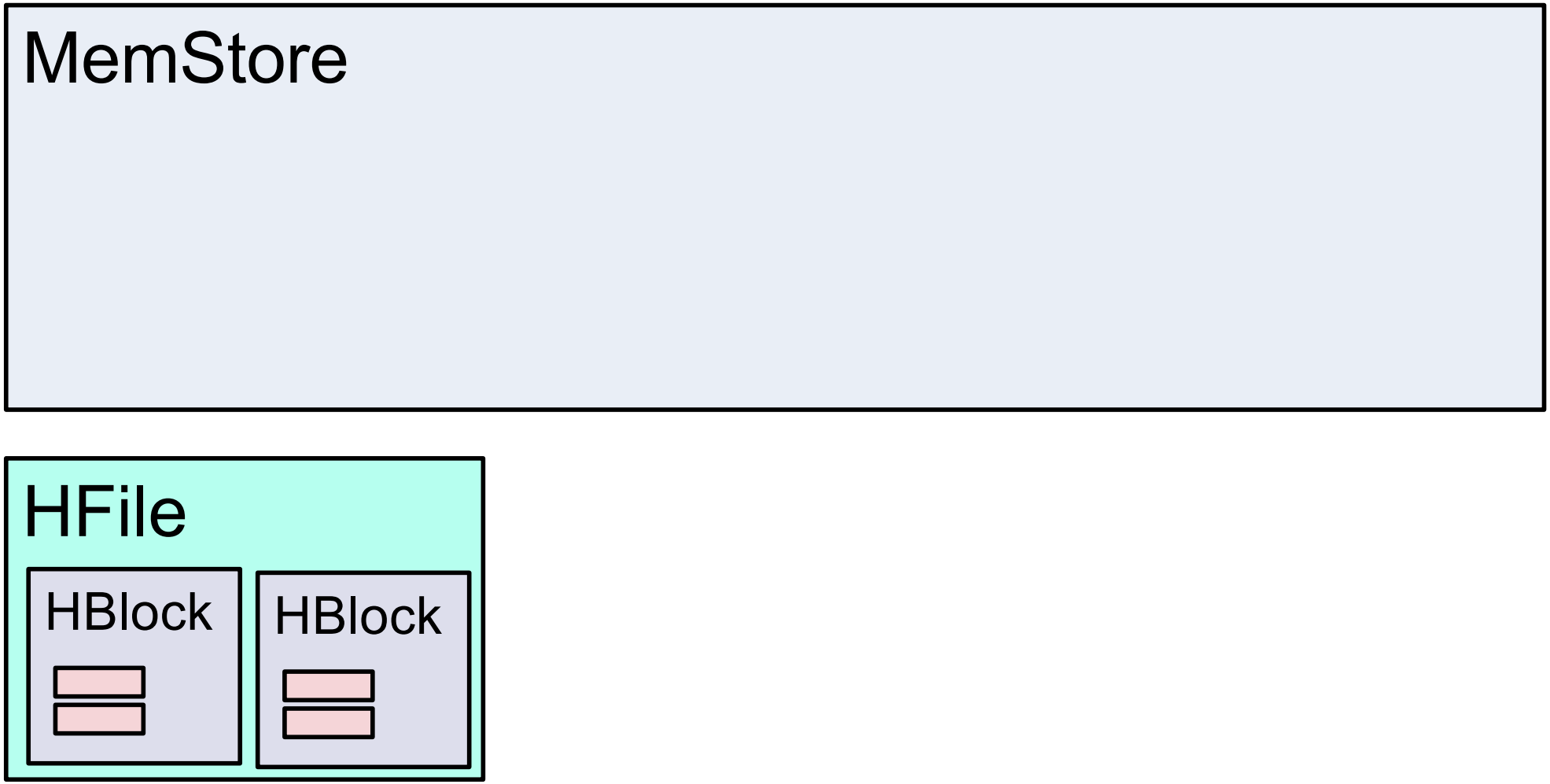


radub85 / 123RF Stock Photo

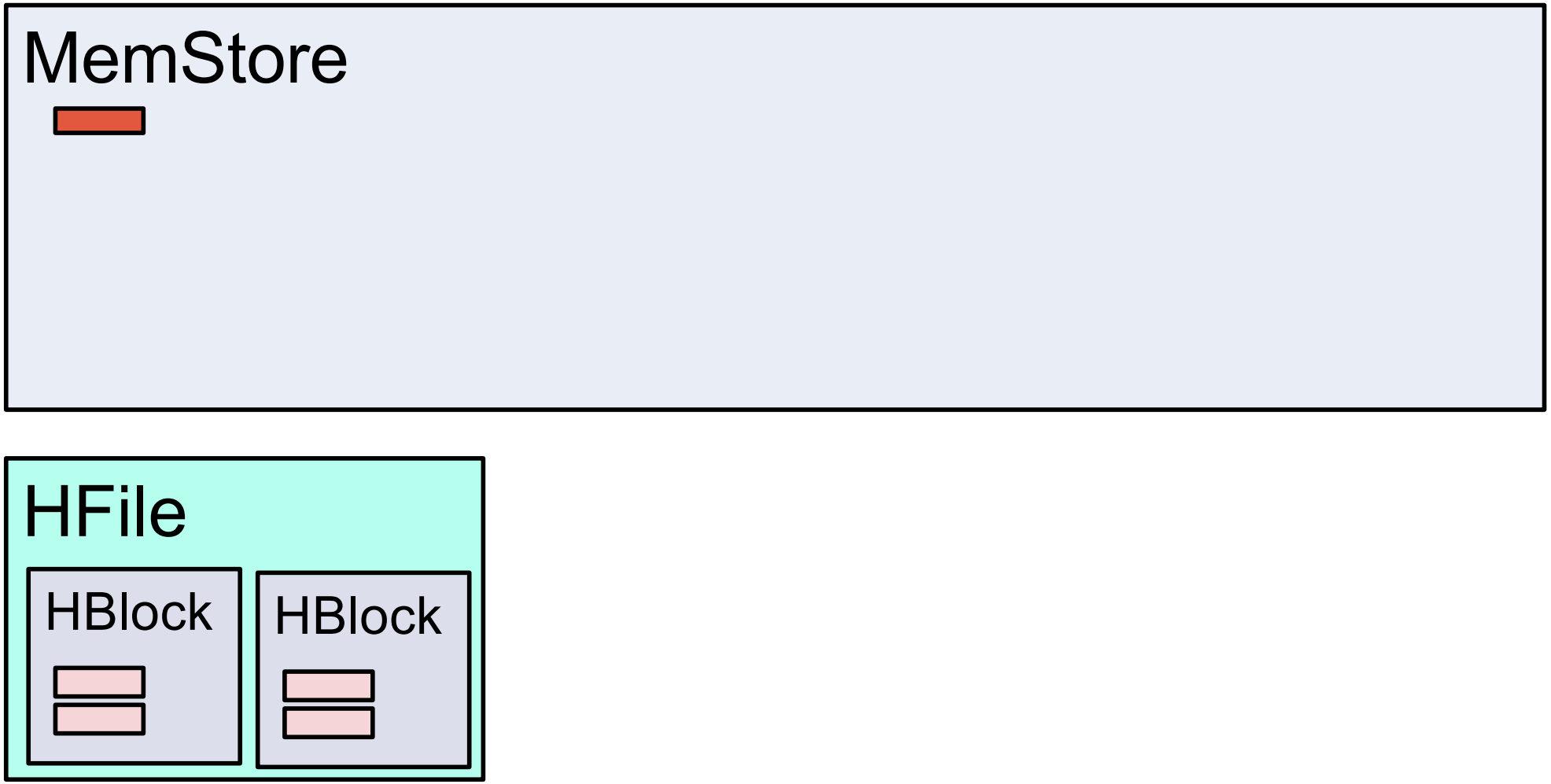




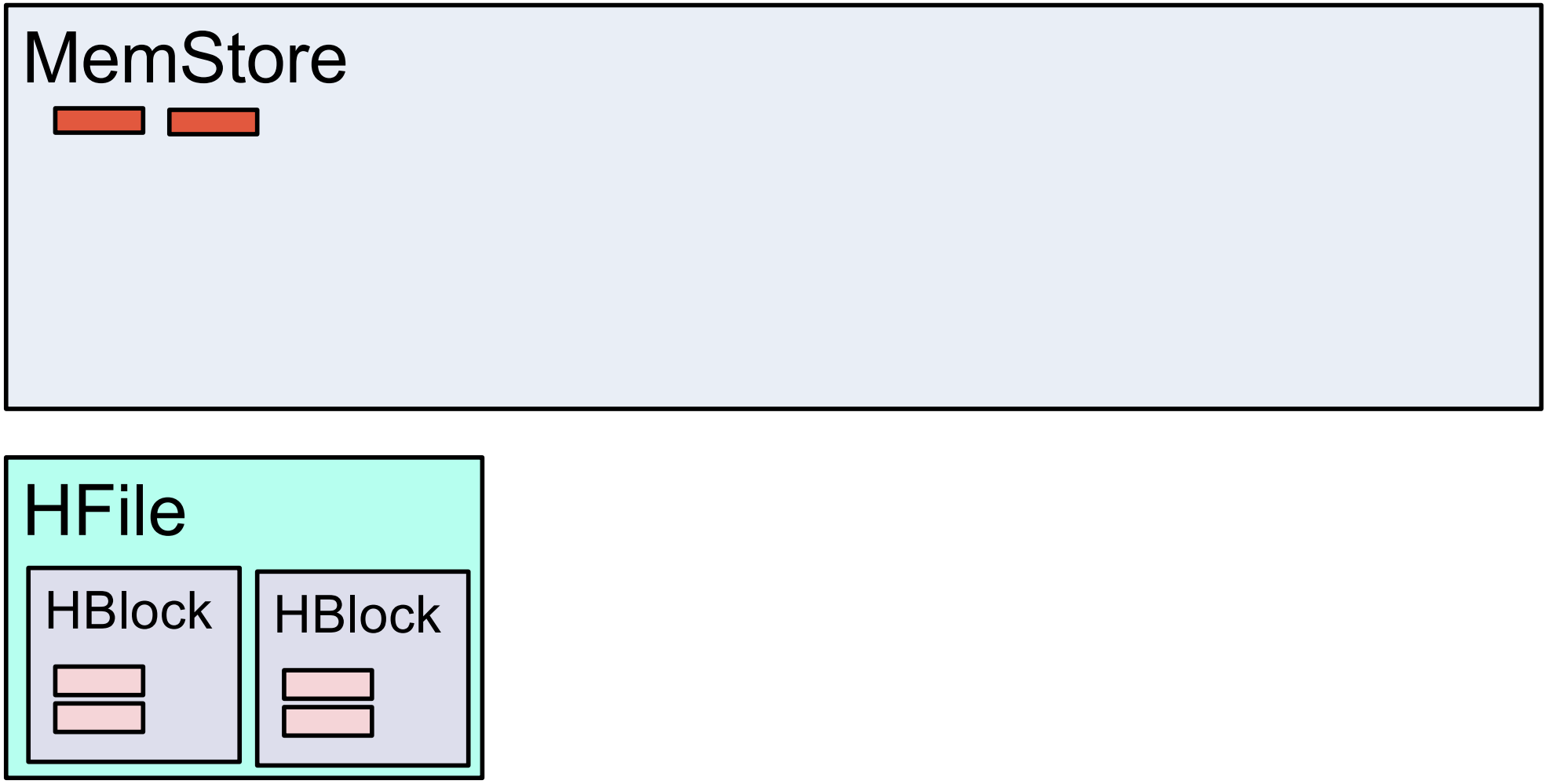
Writing new cells



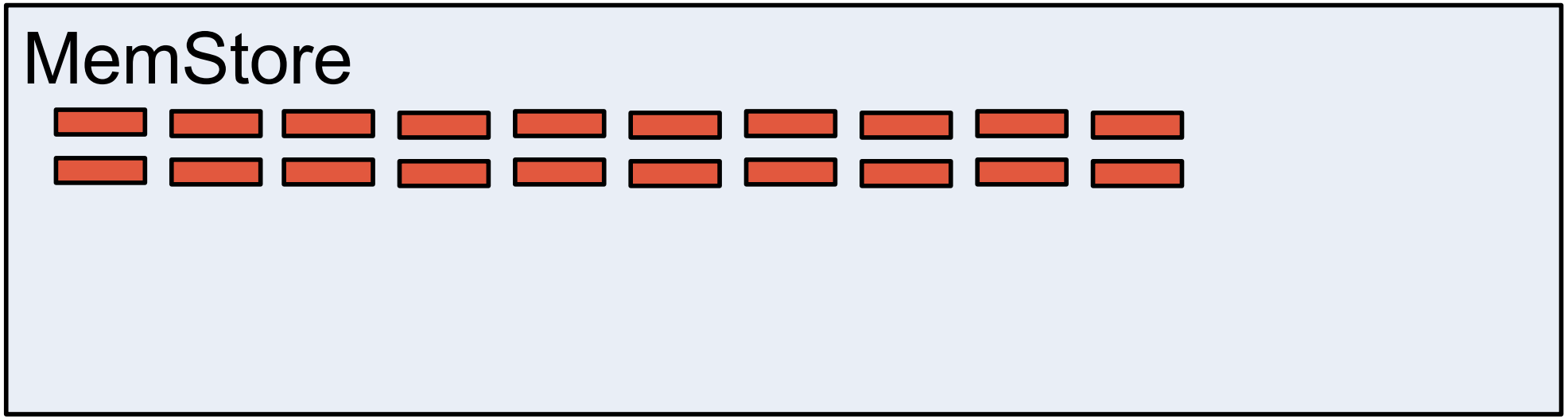
Writing new cells



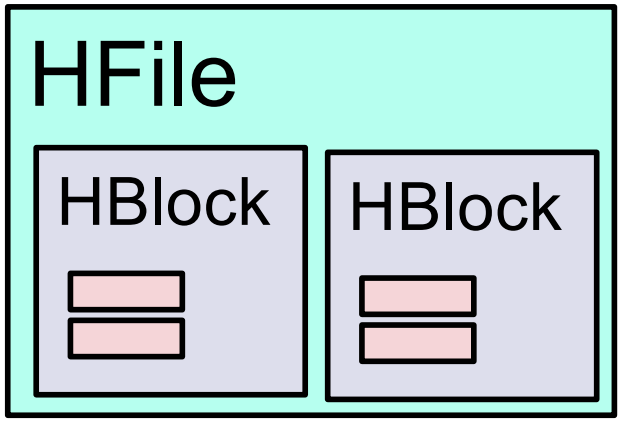
Writing new cells



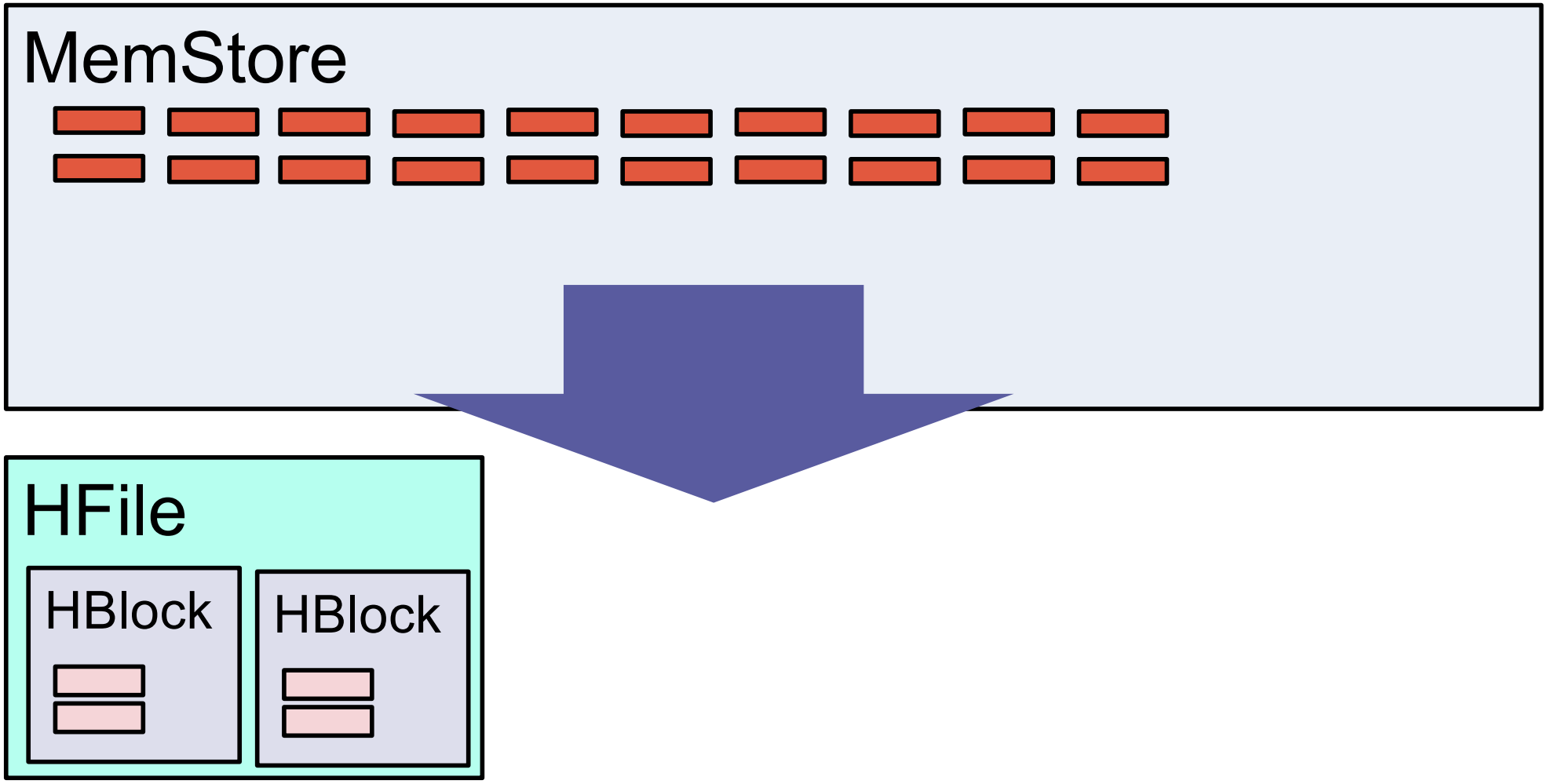
Writing new cells



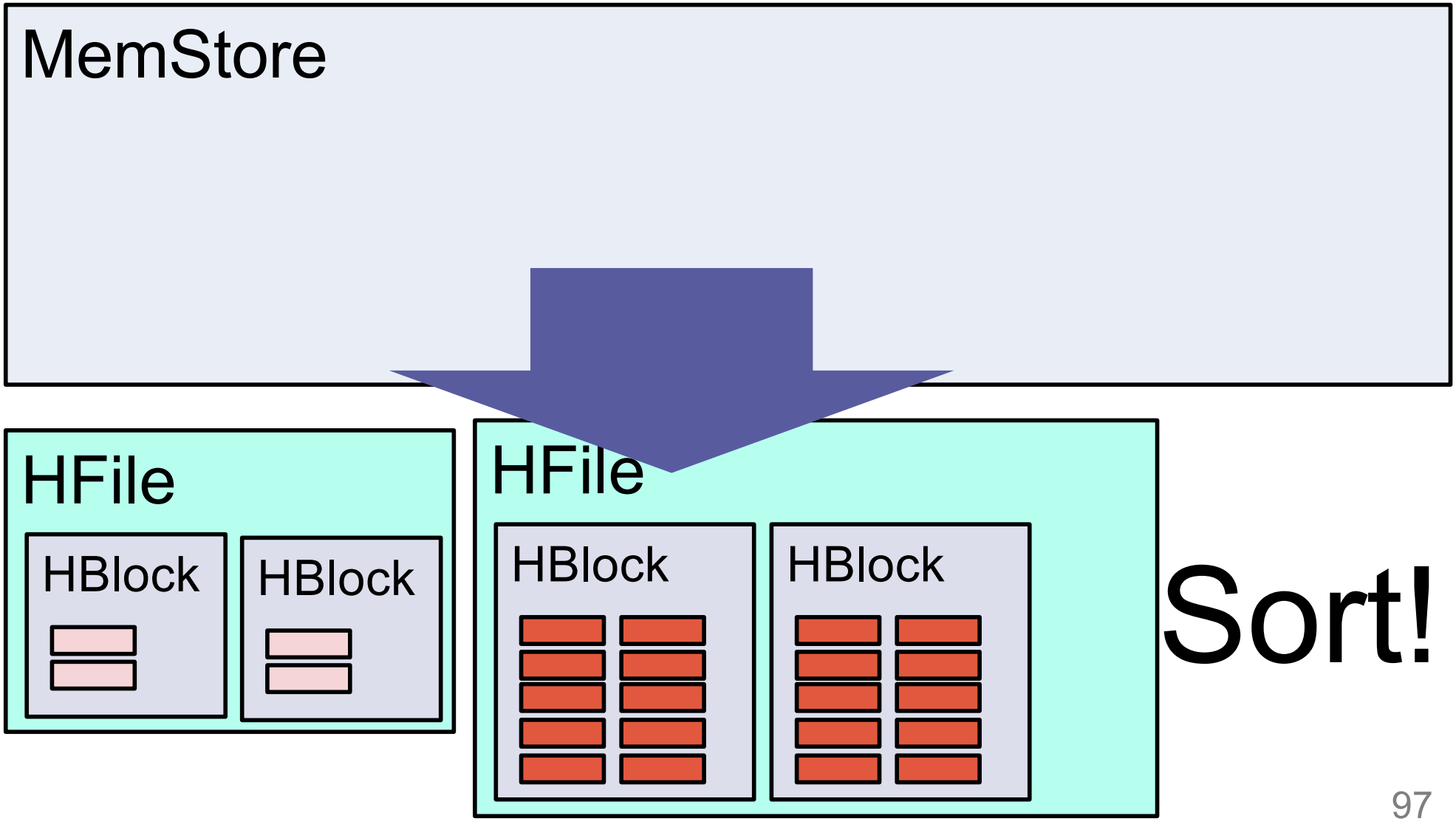
Memory full!



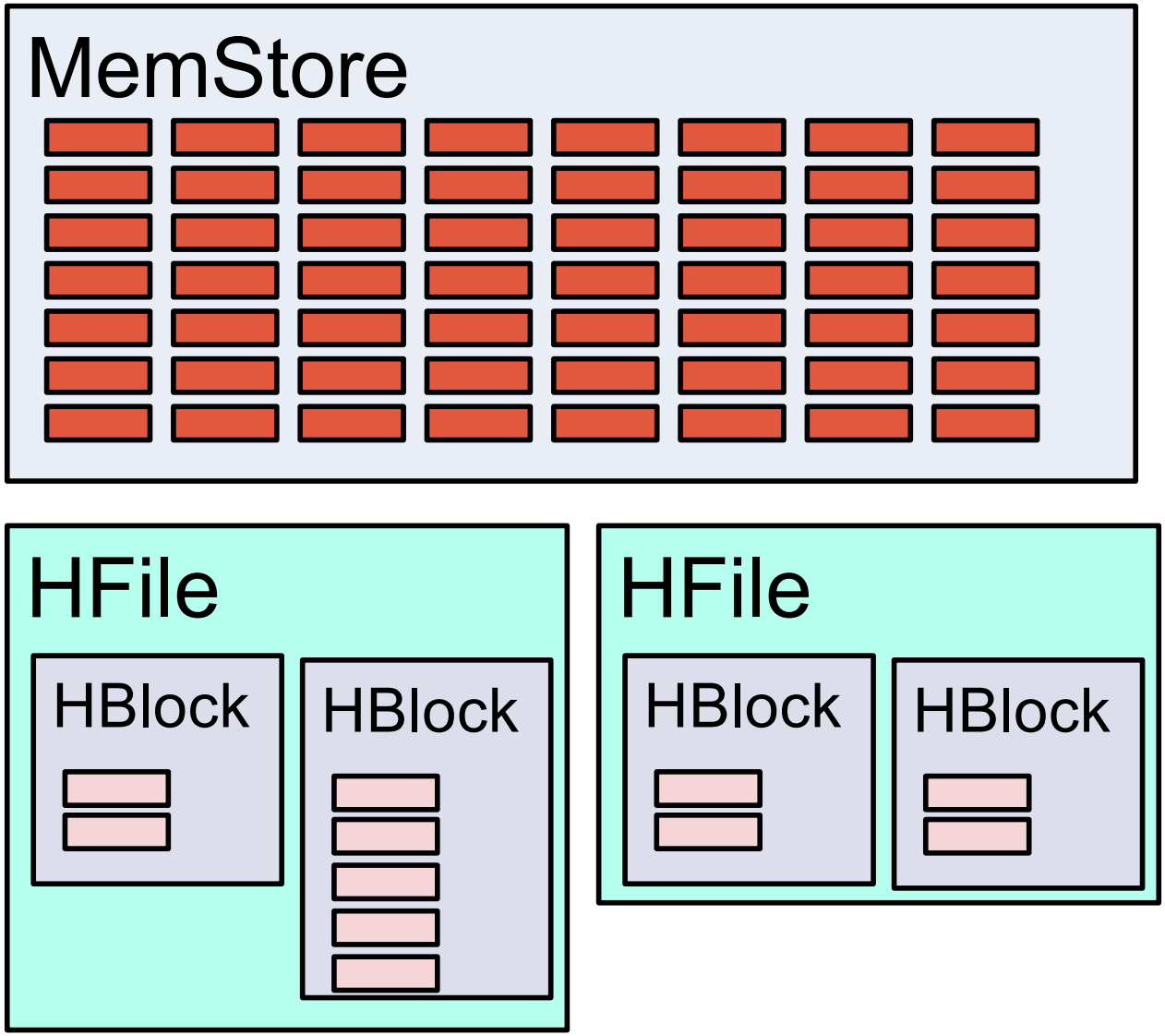
Flush



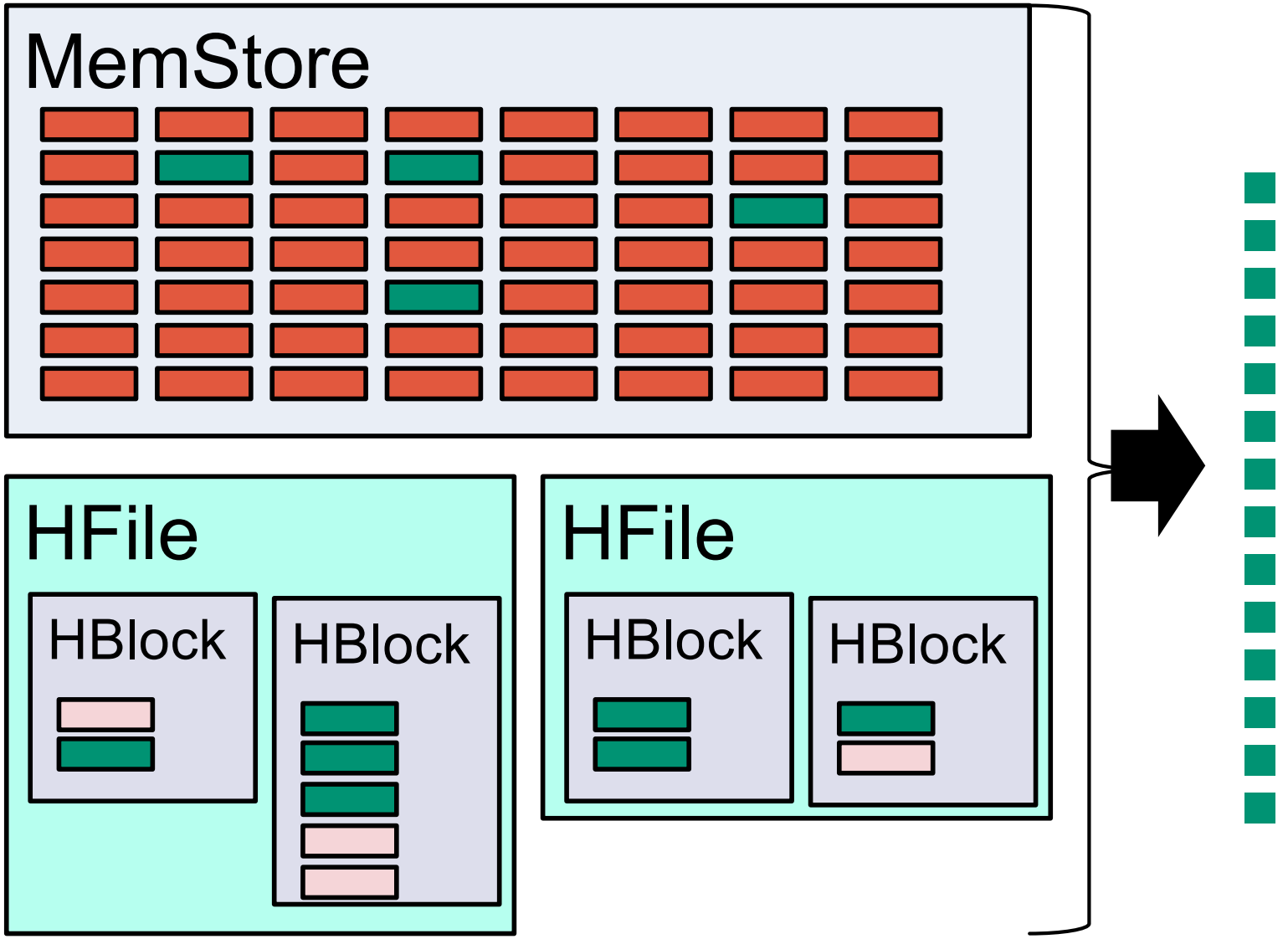
Flush



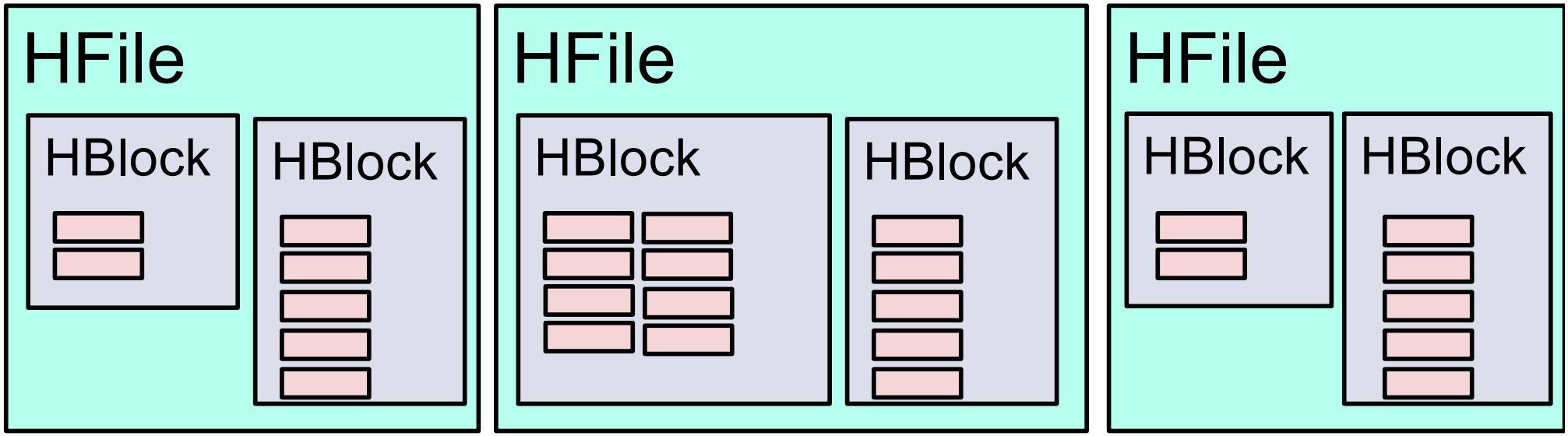
Reading from a Store



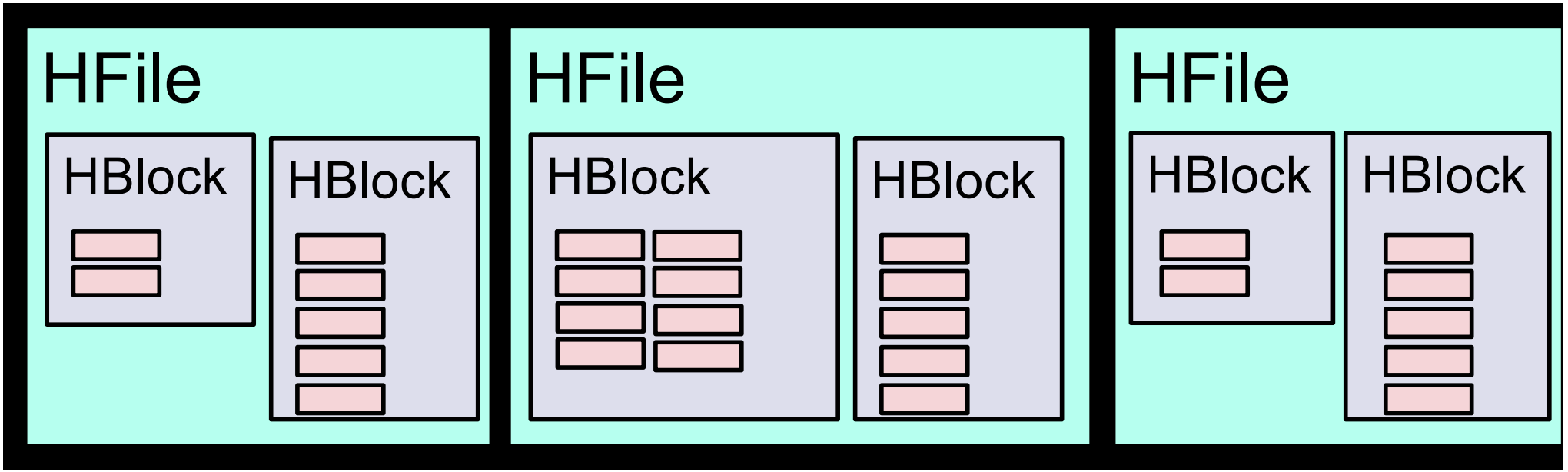
Reading from a Store



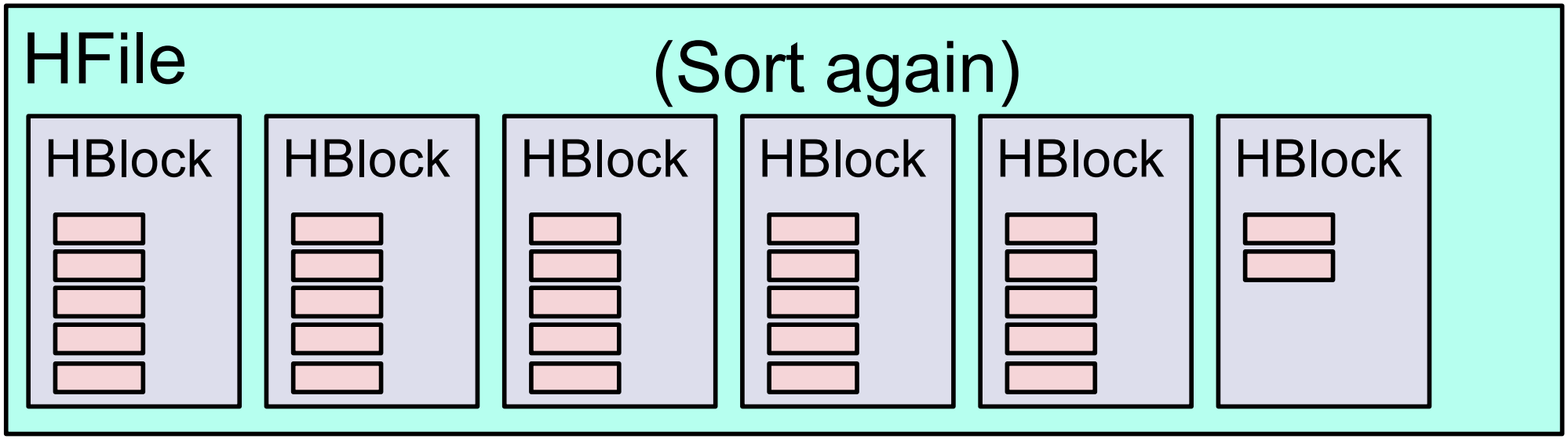
Compaction


HFile
HBlock
HBlock
HFile
HBlock
HBlock
HFile
HBlock
HBlock

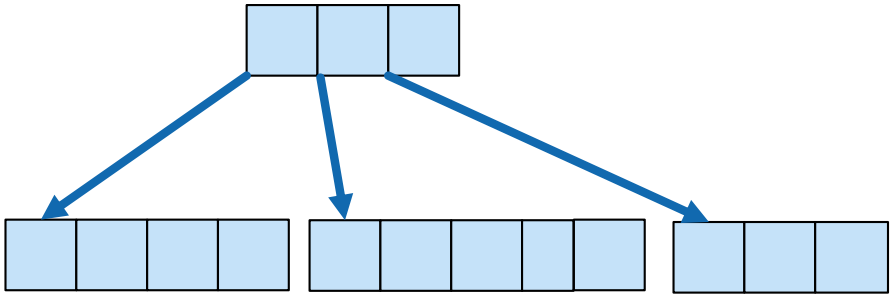
Compaction

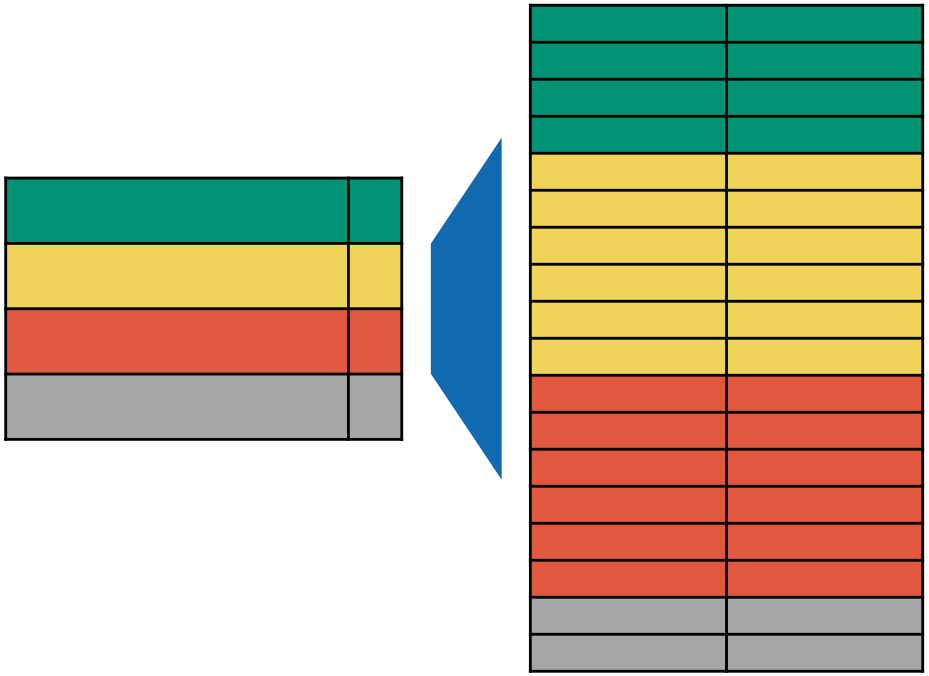

HFile
HBlock
HBlock
HFile
HBlock
HBlock
HFile
HBlock
HBlock

Compaction


HFile
(Sort again)
HBlock
HBlock
HBlock
HBlock
HBlock
HBlock

Seek vs. Transfer

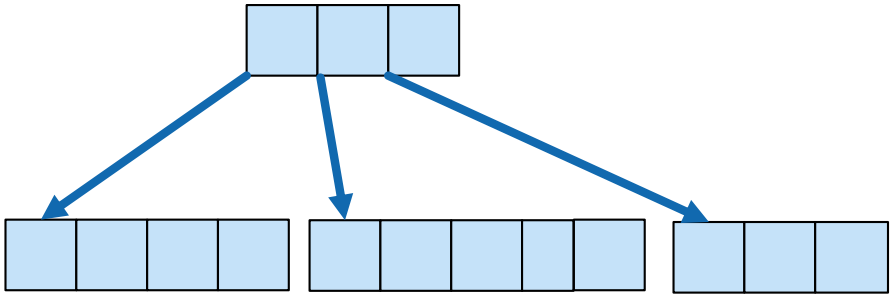


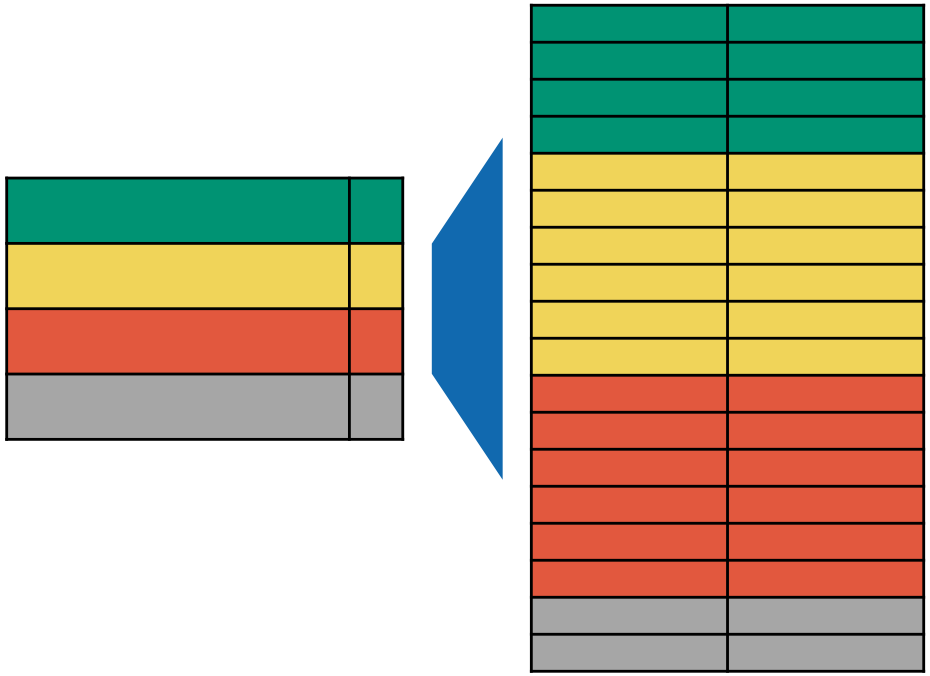


B+-trees

LSM-Trees

Seek vs. Transfer

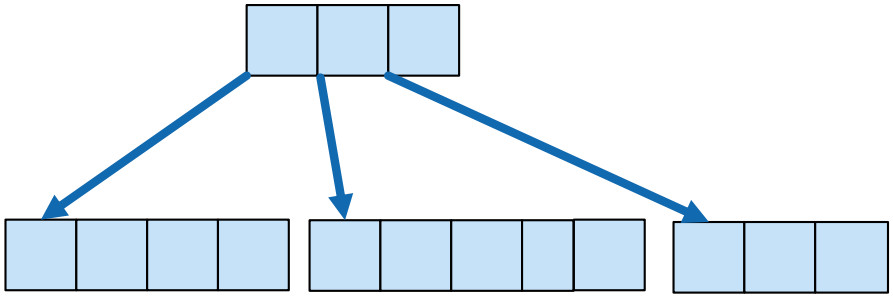


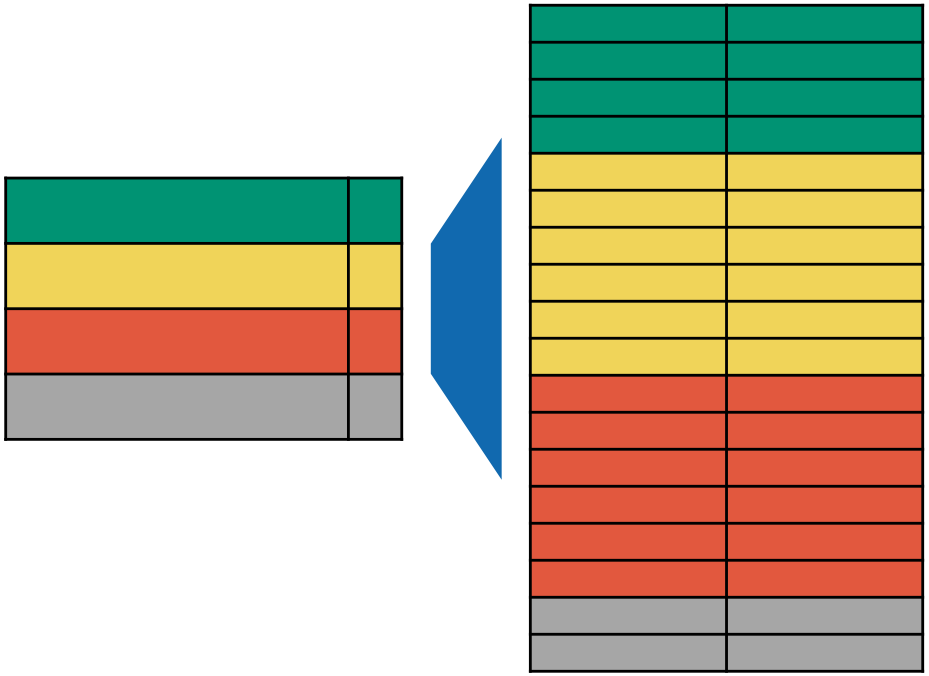


Classical RDBMS

Wide column stores

Seek vs. Transfer



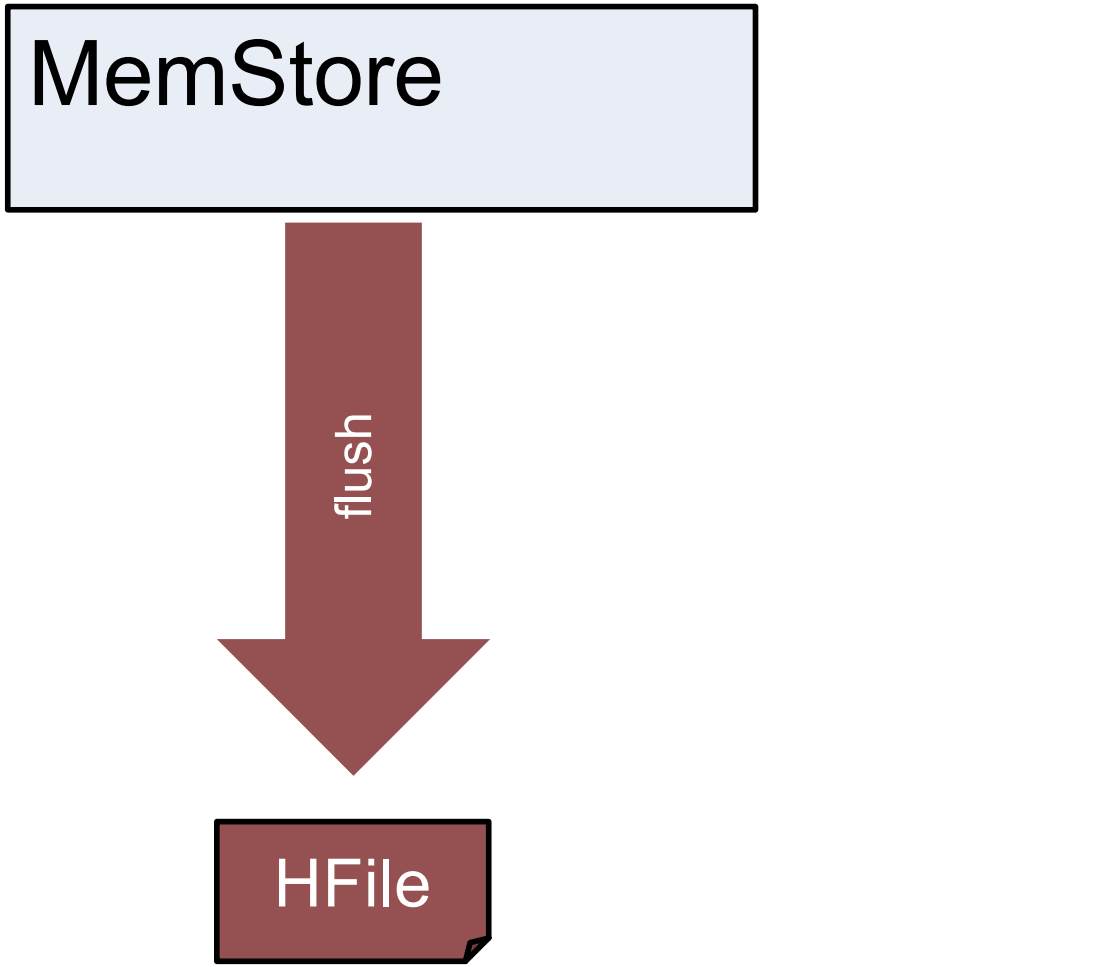


Seek-time-bound

Transfer-time-bound

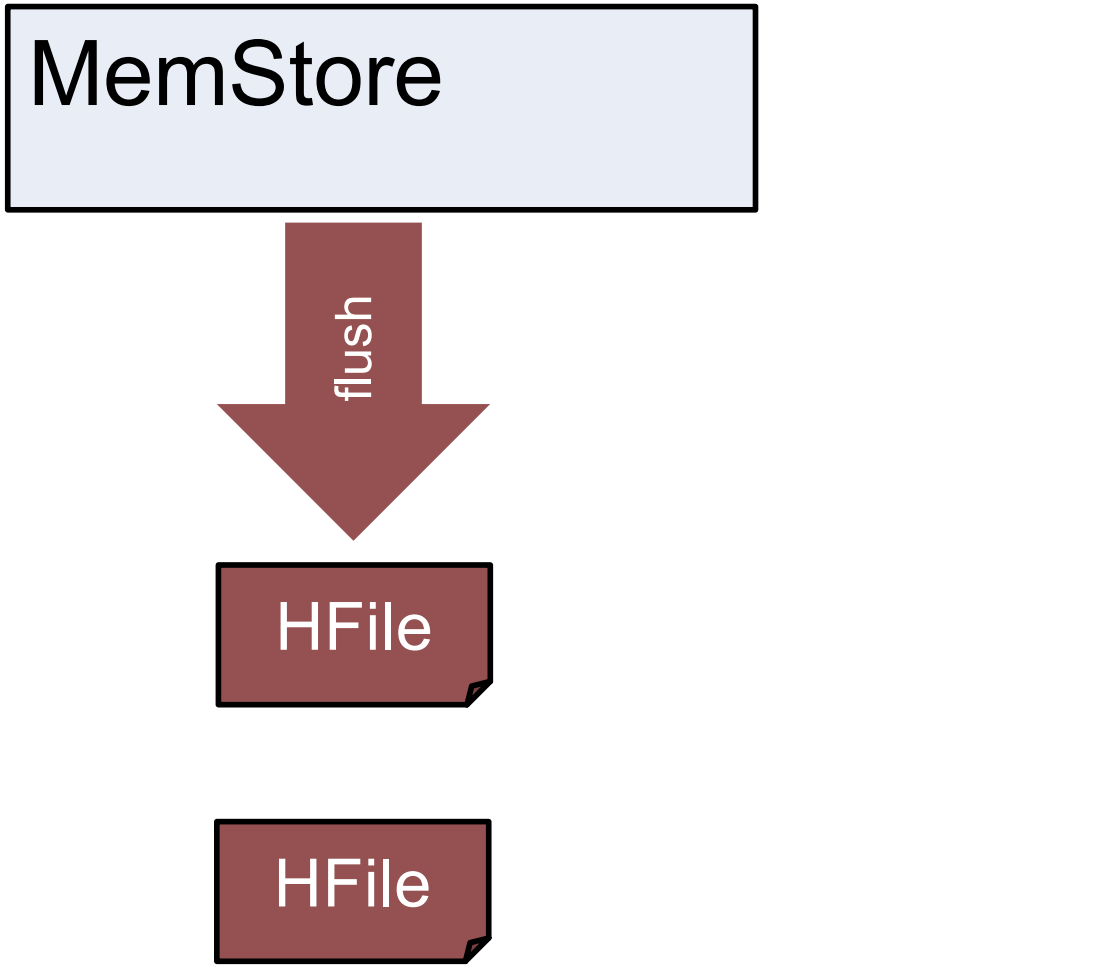
Log-Structured Merge-Trees

Log-Structured Merge-Trees



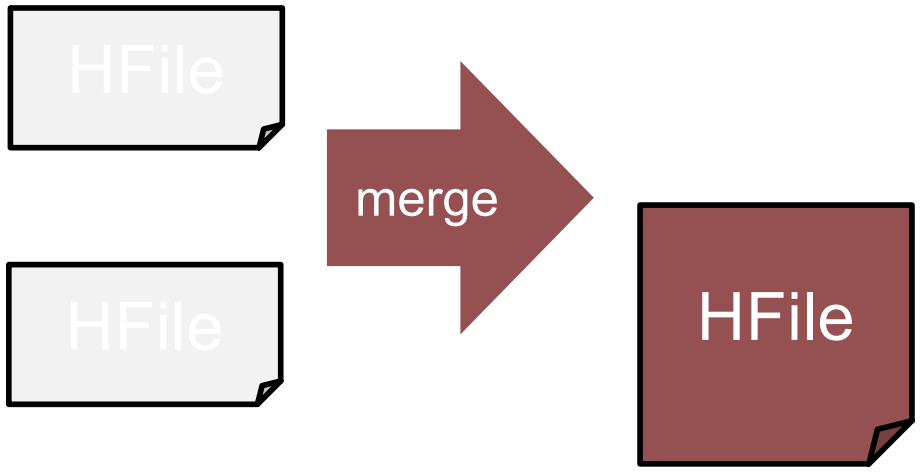
Log-Structured Merge-Trees

Log-Structured Merge-Trees

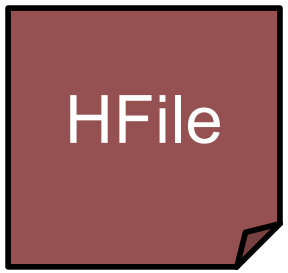


Log-Structured Merge-Trees

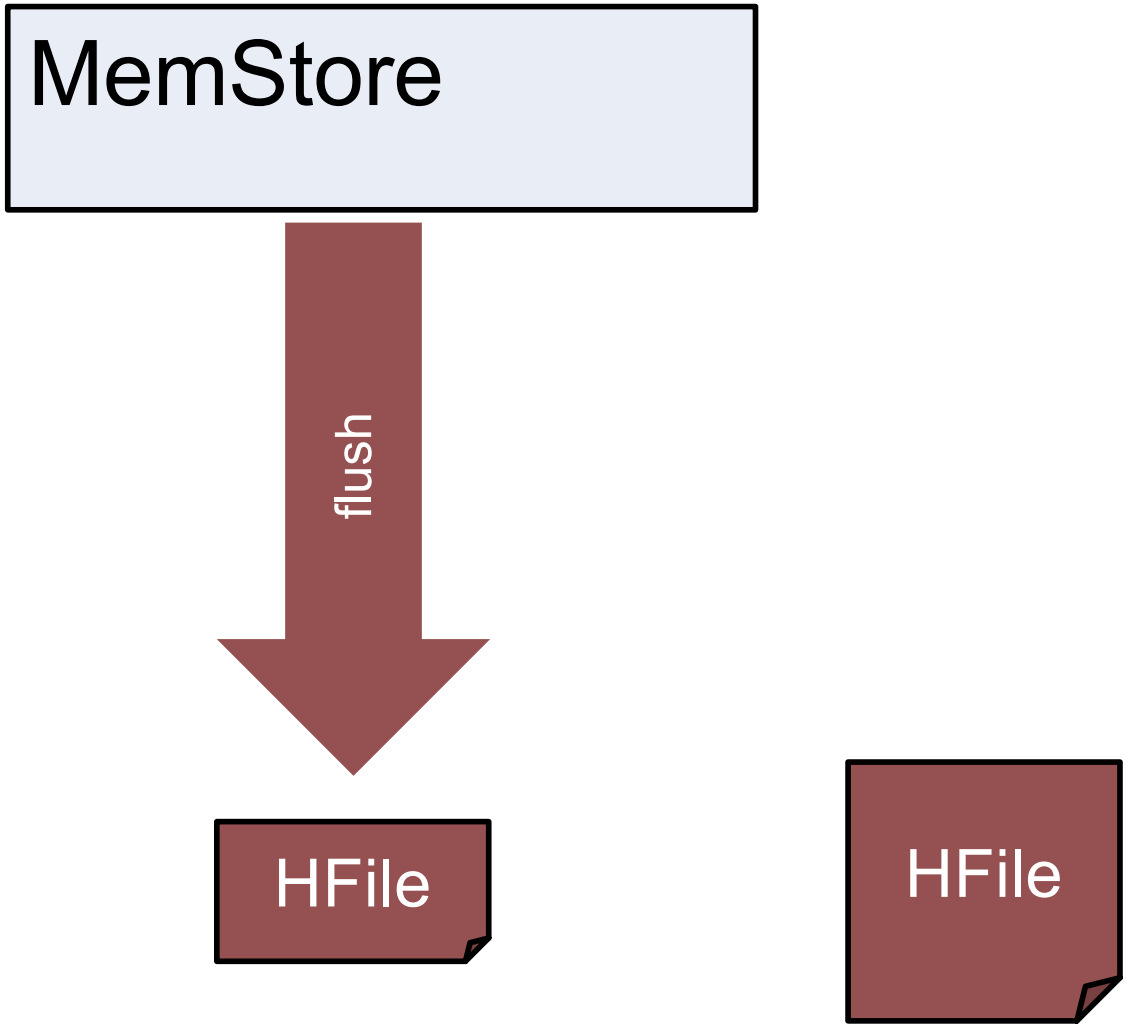
MemStore



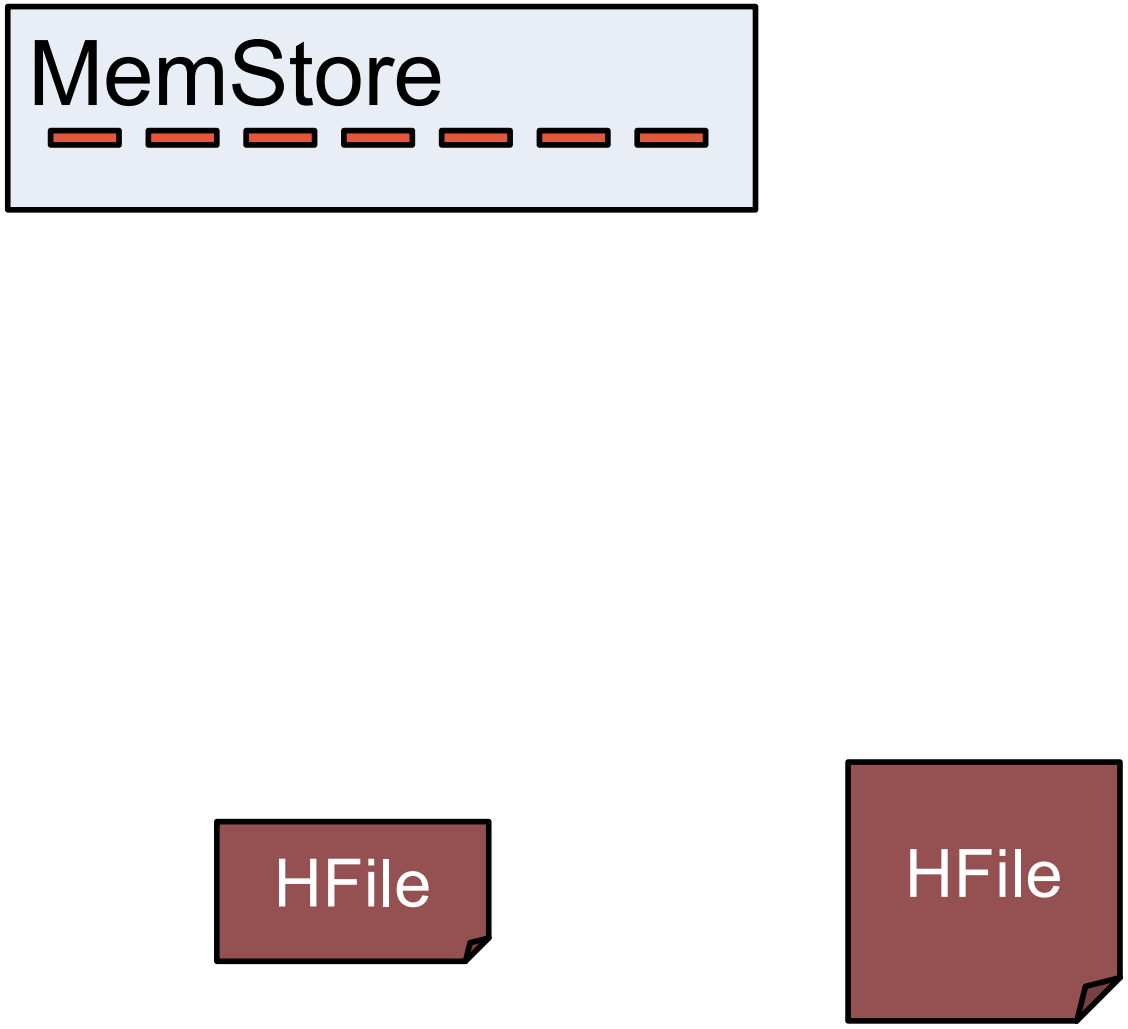
Log-Structured Merge-Trees



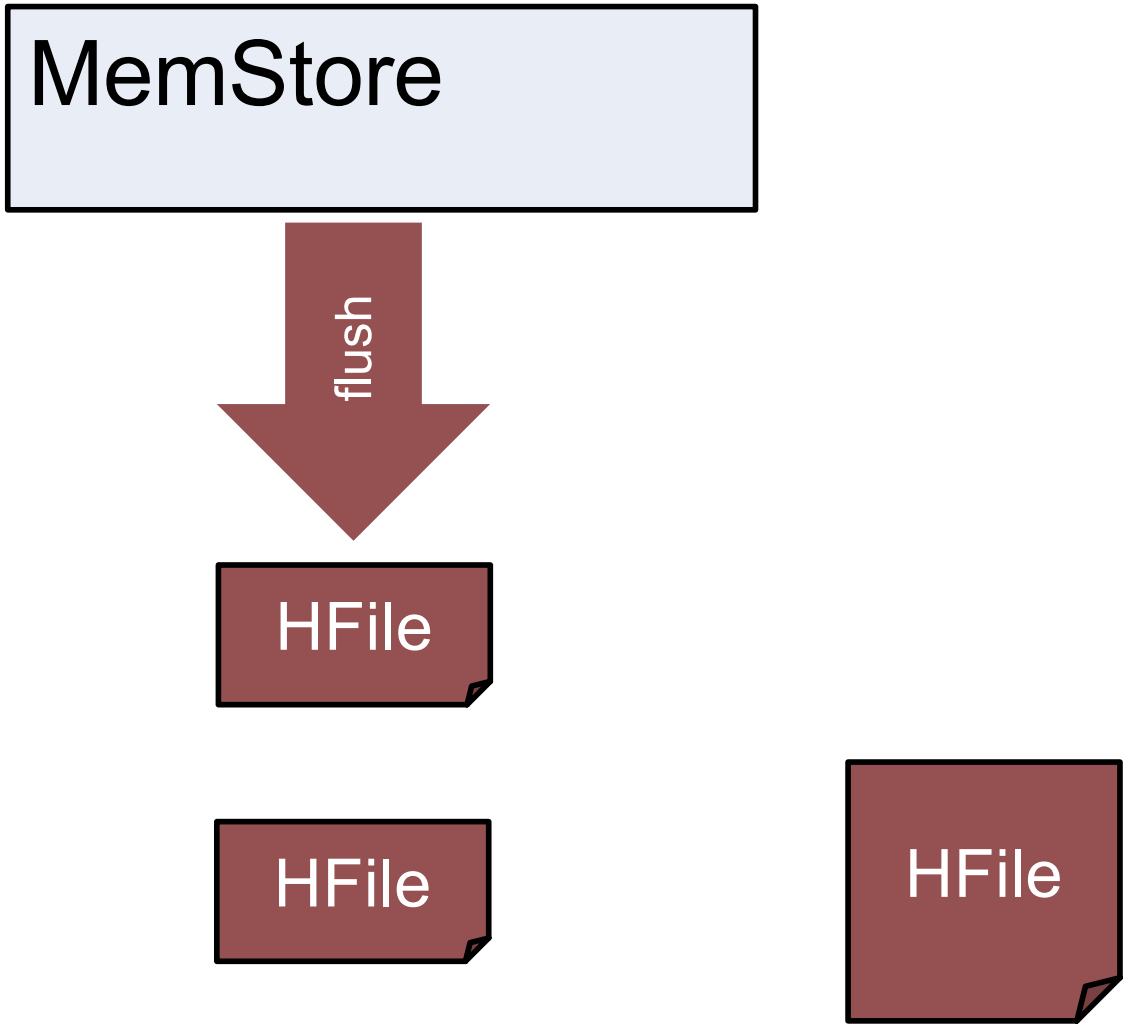
Log-Structured Merge-Trees



Log-Structured Merge-Trees

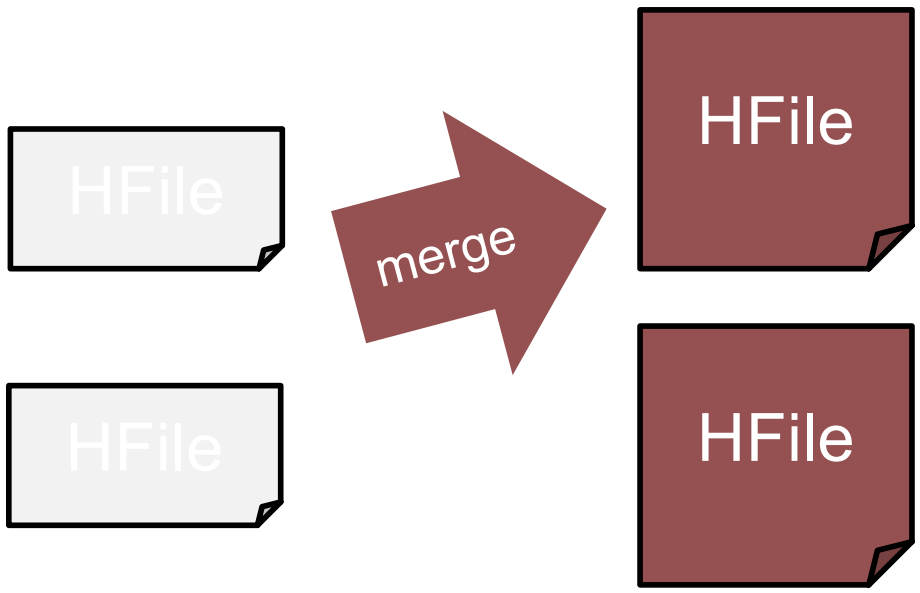


Log-Structured Merge-Trees



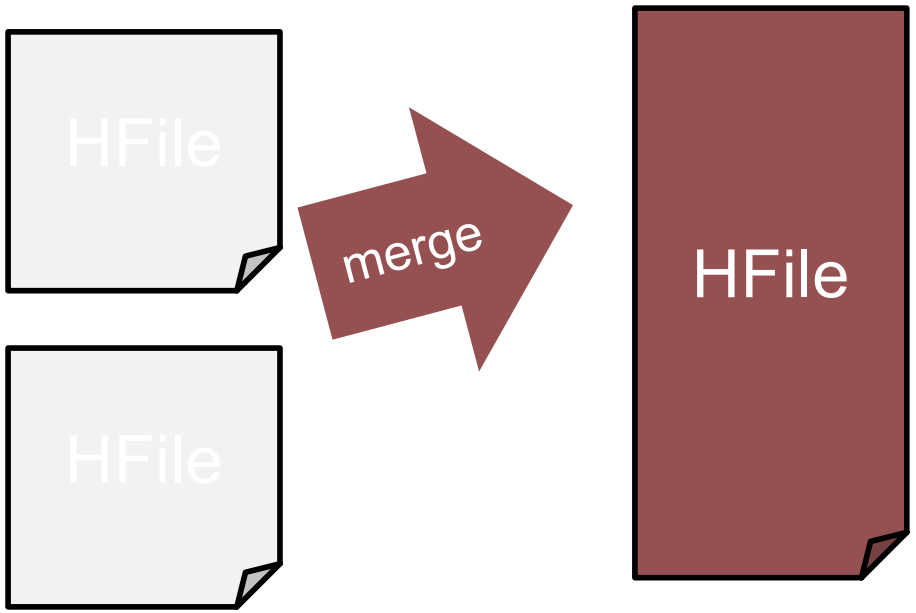
Log-Structured Merge-Trees

MemStore



Log-Structured Merge-Trees

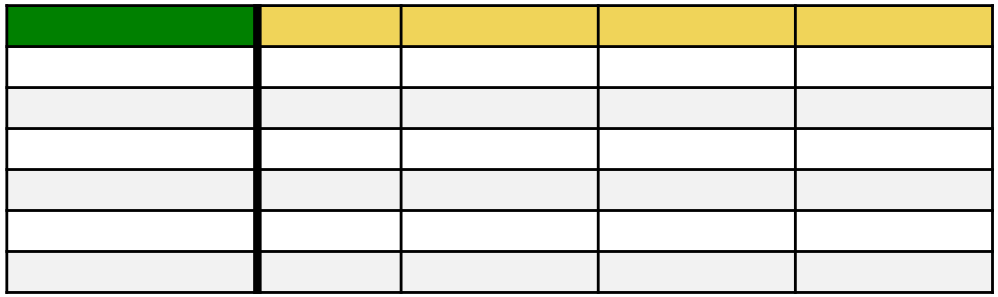
MemStore

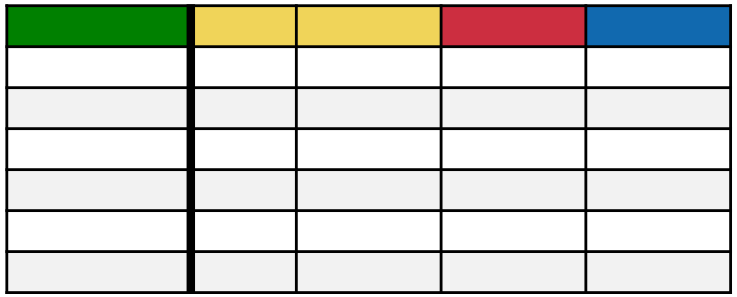


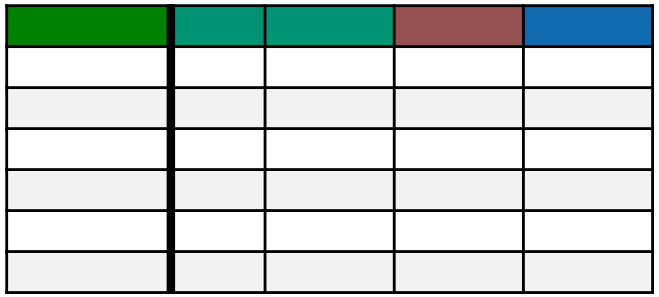
HBase Bootstrap

Meta

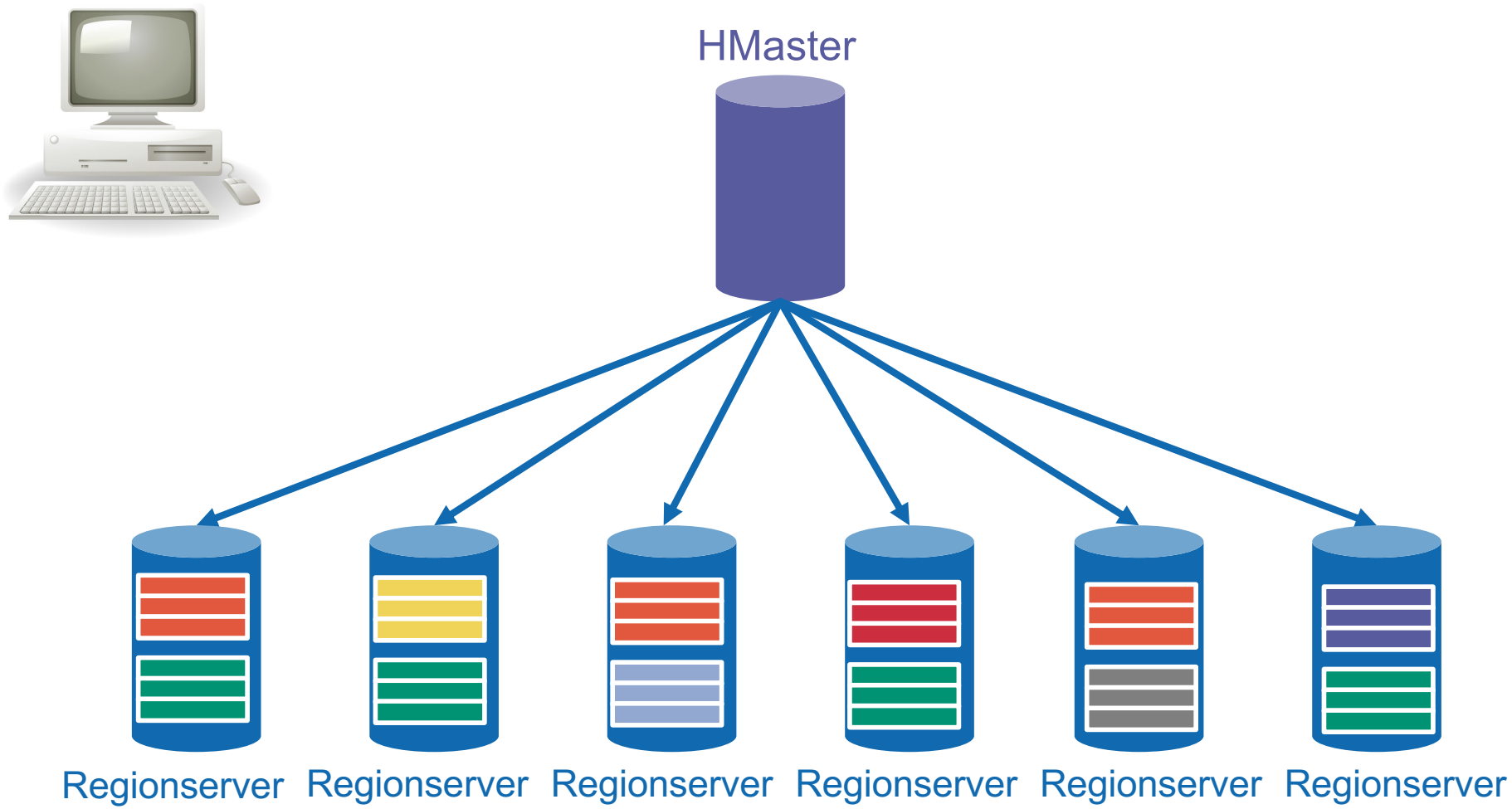
Regular tables



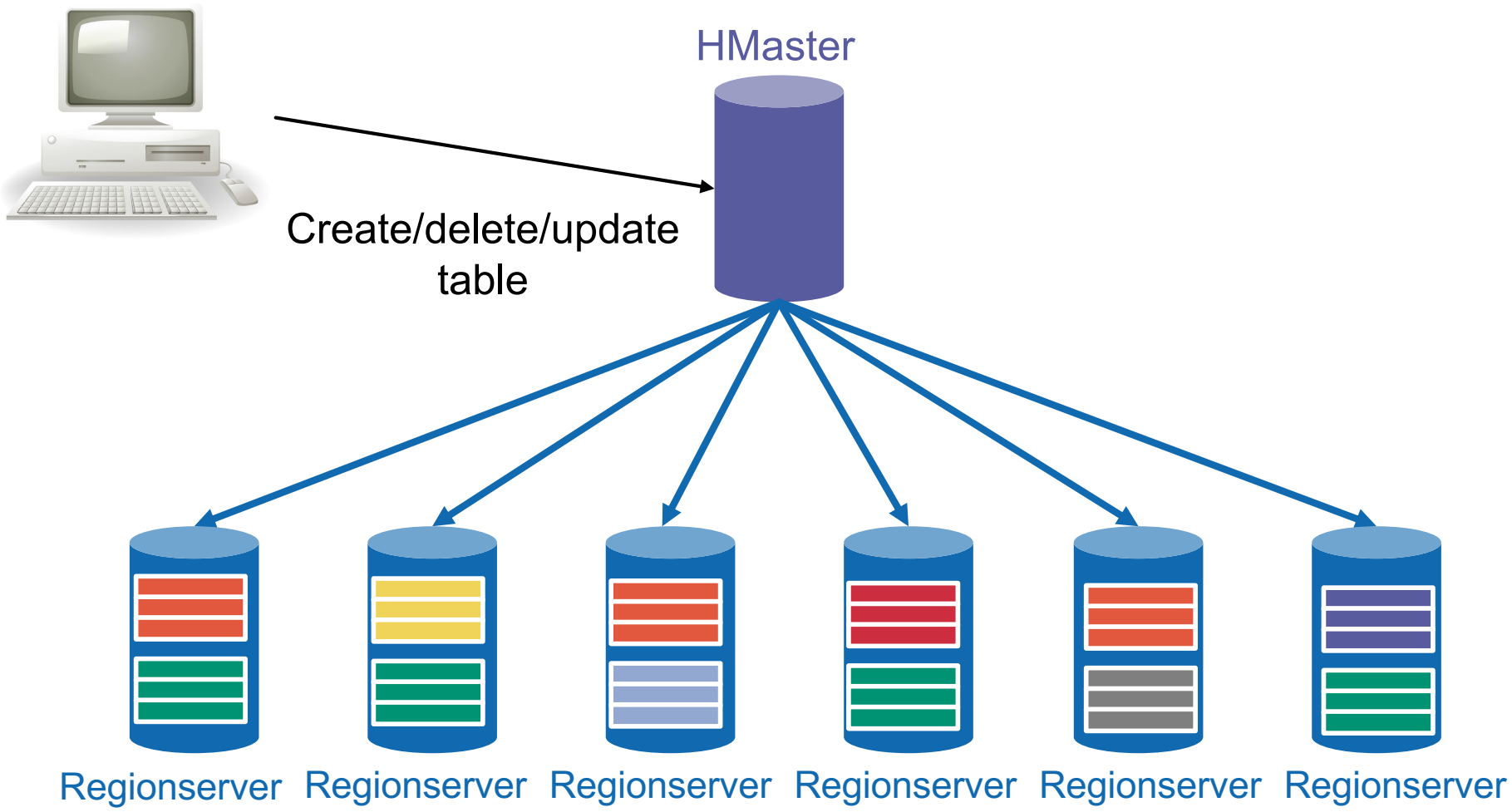




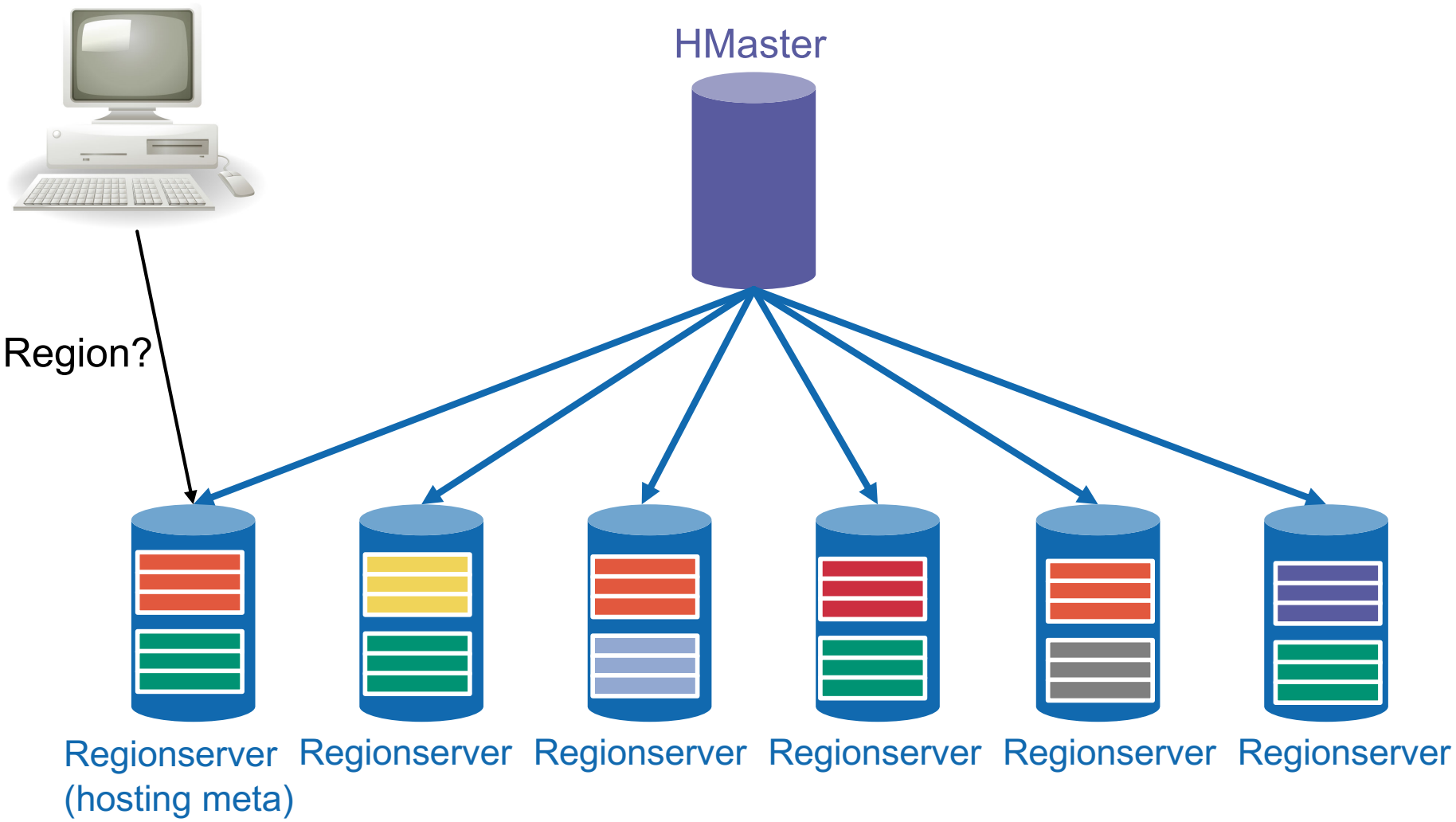
Architecture



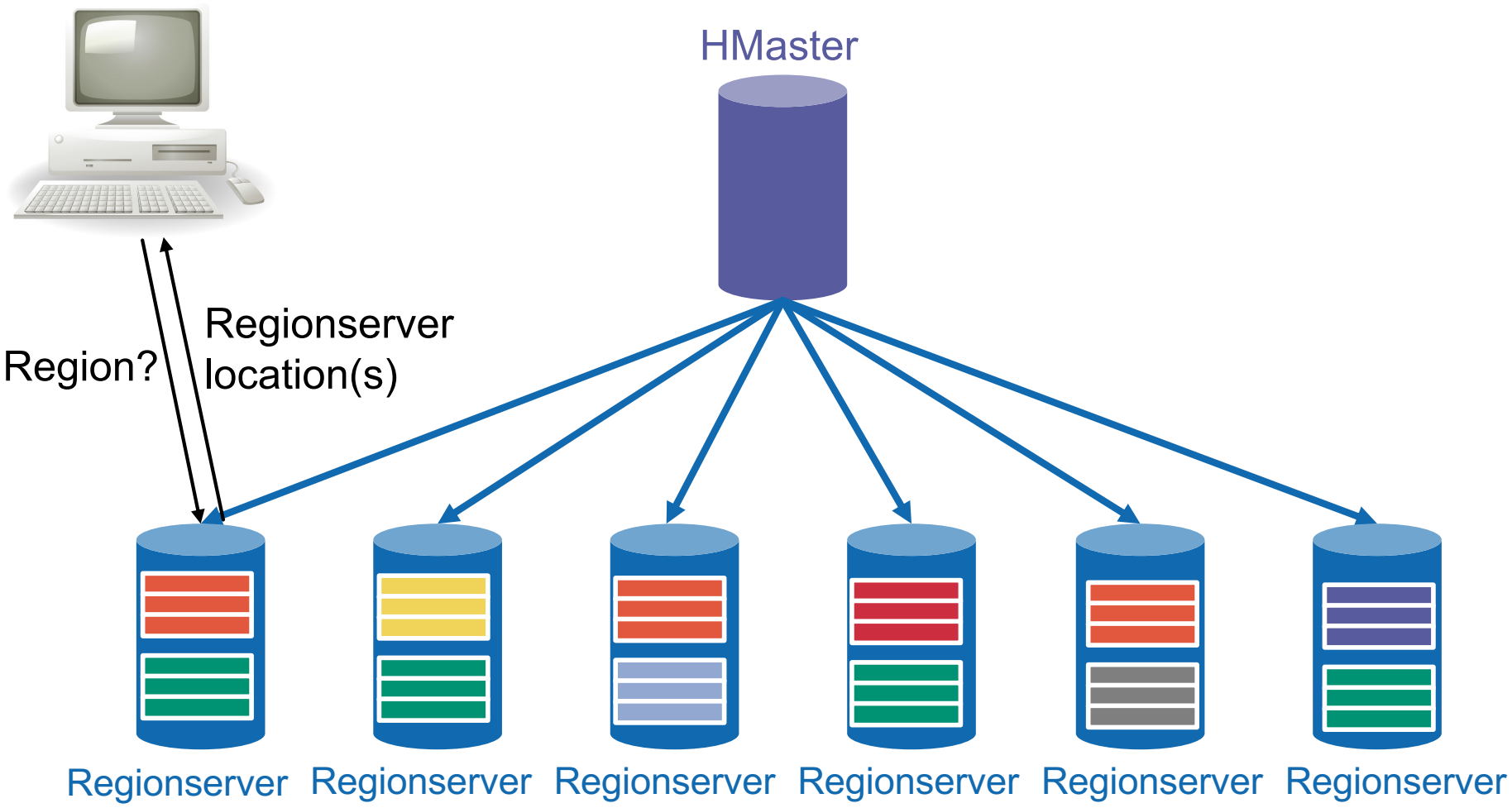
Architecture



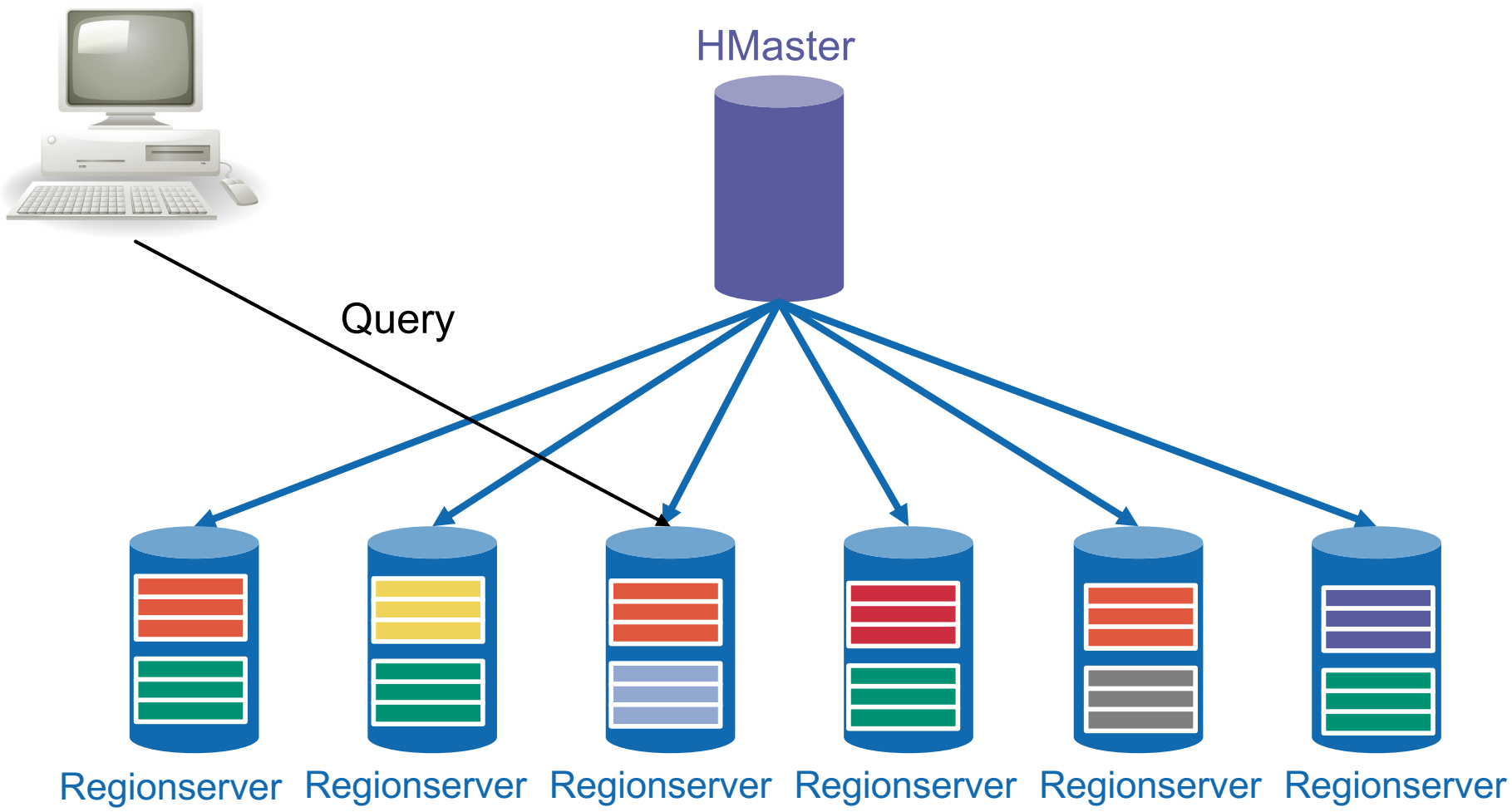
Architecture



Architecture



Architecture



HBase: Underlying APIs

HBase implementation

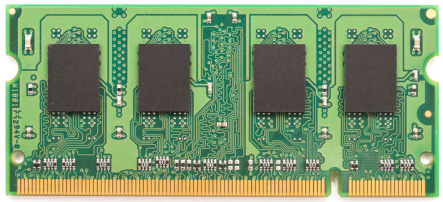
(Packaged code)

HBase APIs

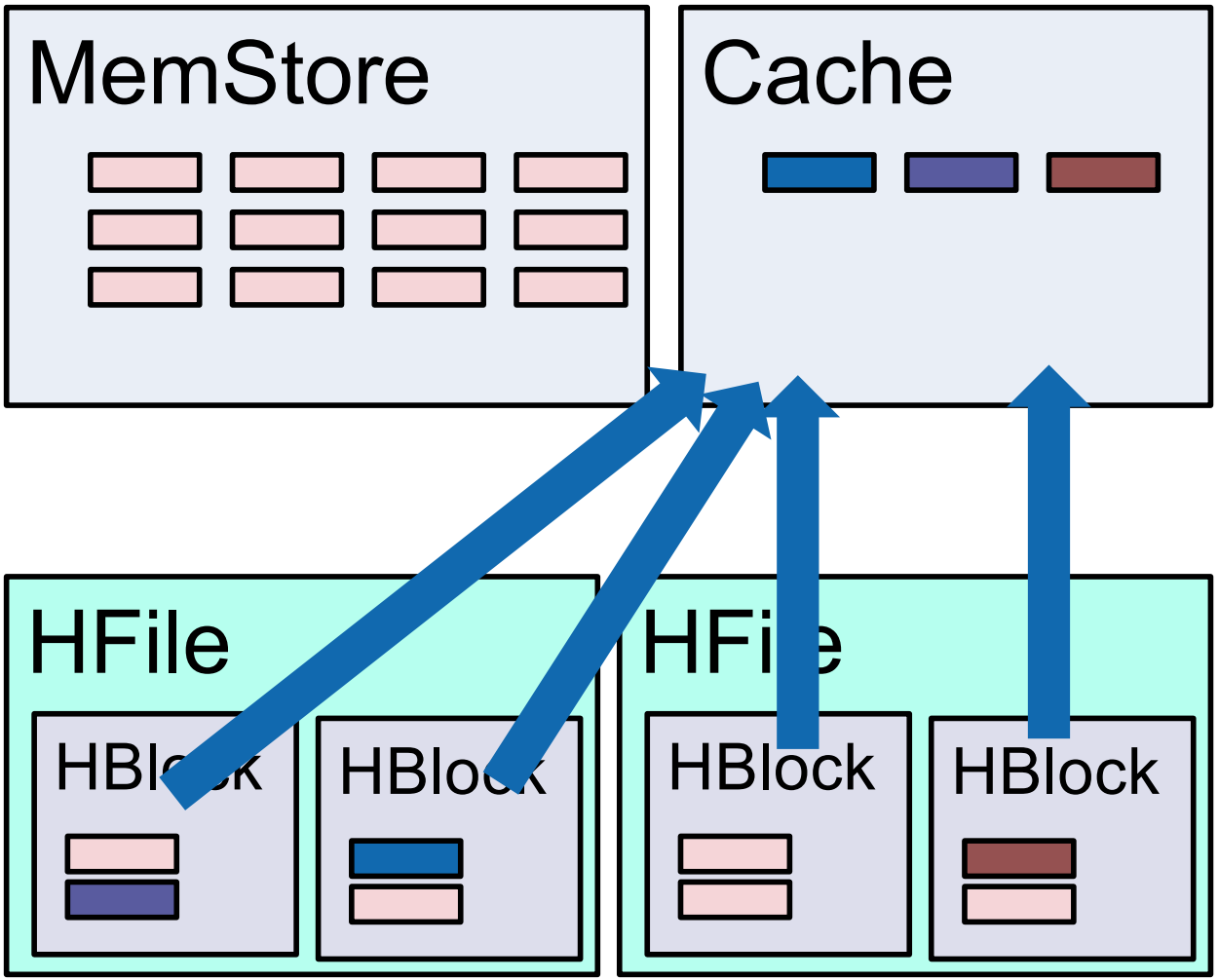
REST

HBase: caching

Store



radub85 / 123RF Stock Photo

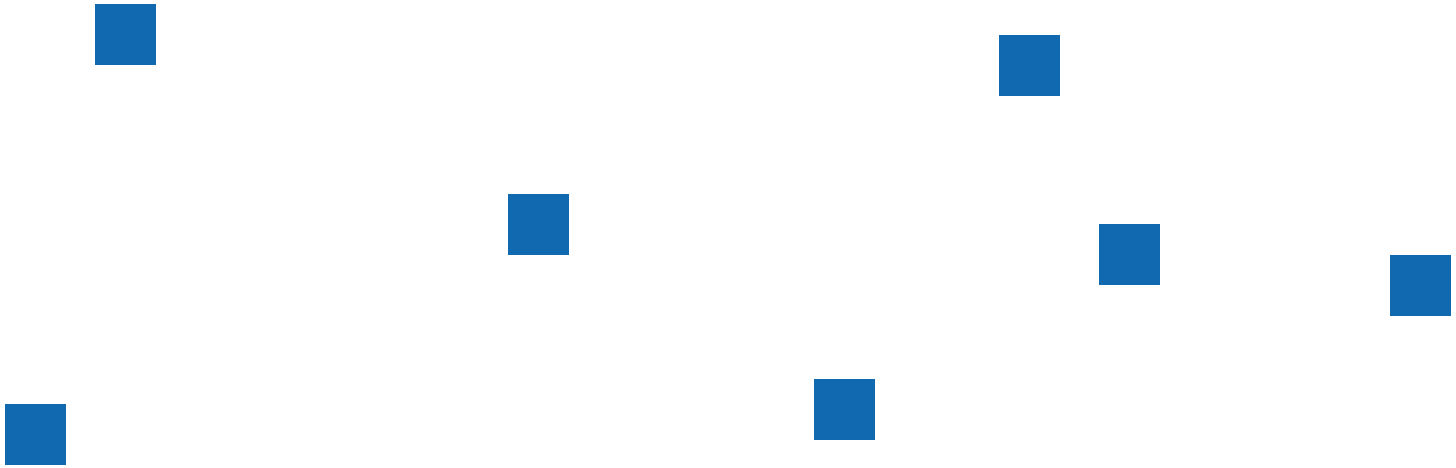


When NOT to use the cache

When NOT to use the cache

Batch processing

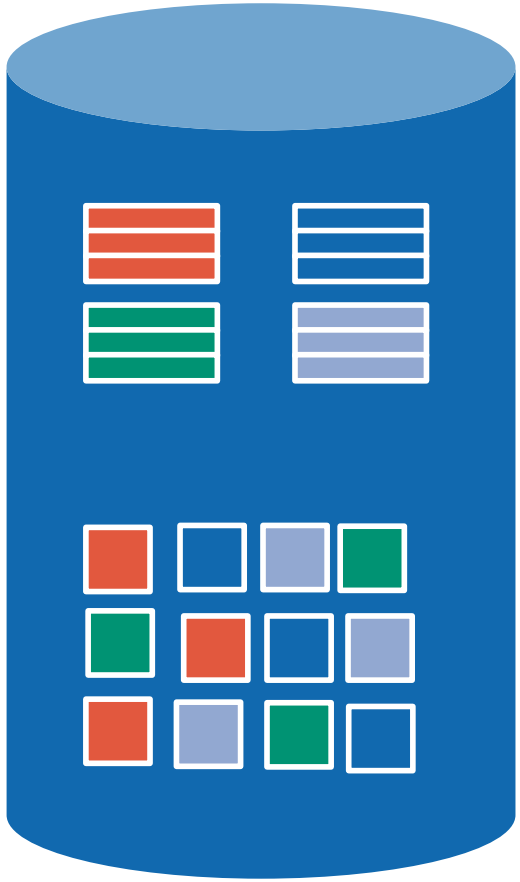
When NOT to use the cache



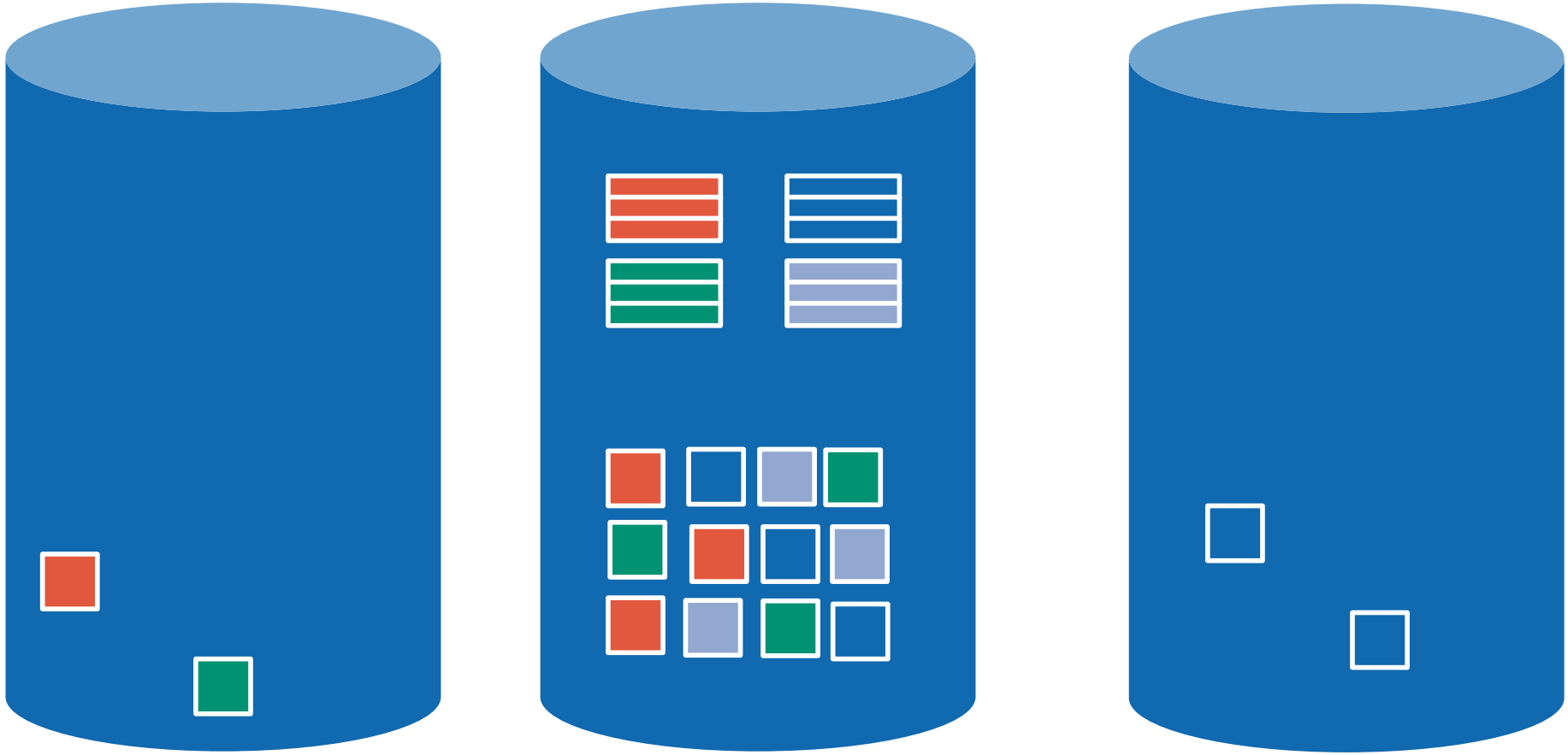
Random access

Data Locality

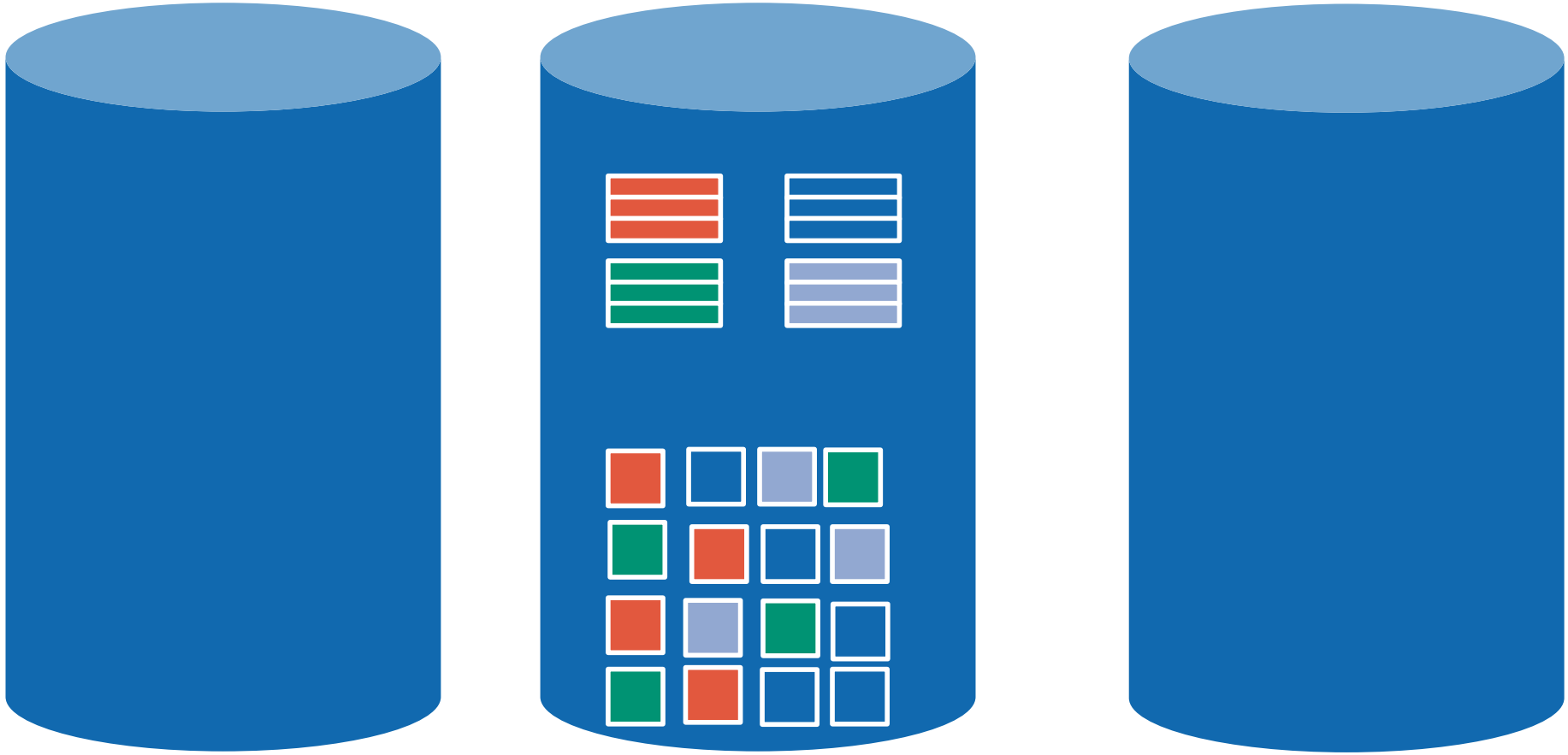
HBase vs. HDFS



With HDFS load balancer...

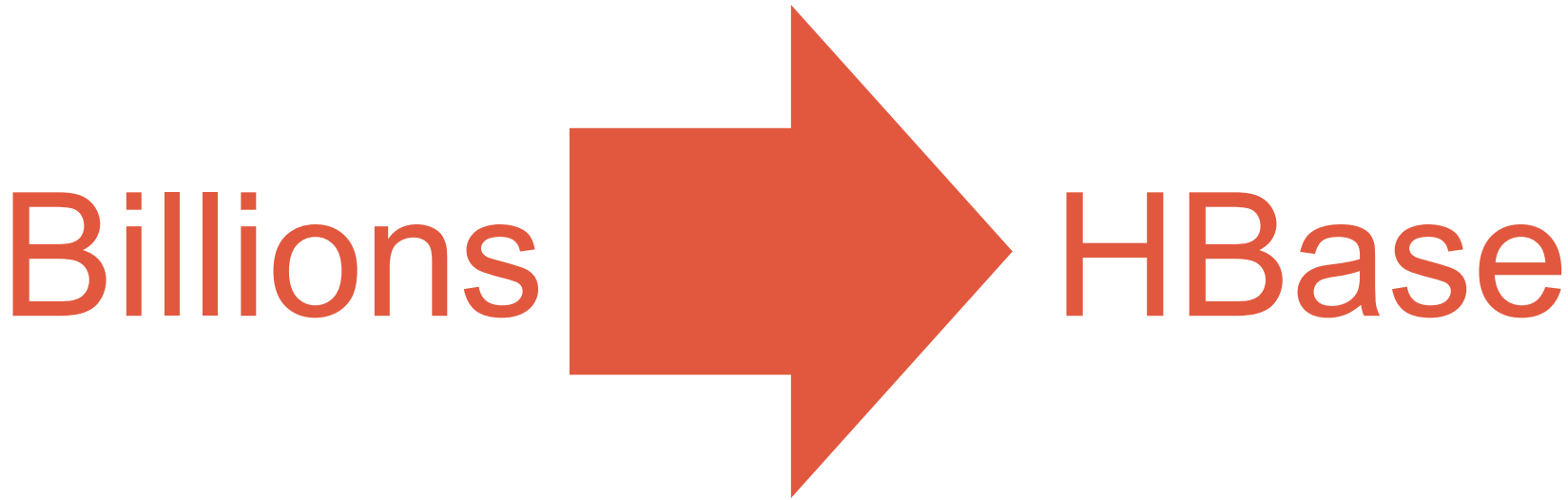


HFile compaction brings back locality



Best practices

Number of rows



Number of nodes

> 5

Row IDs and column names

keep them
>short<

why?

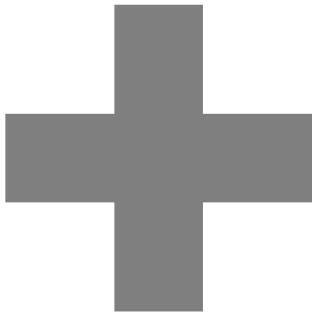
Spanner (Optional/bonus)

Spanner

Tabular Data Model

Language

ACID properties



Distribution

Scalability

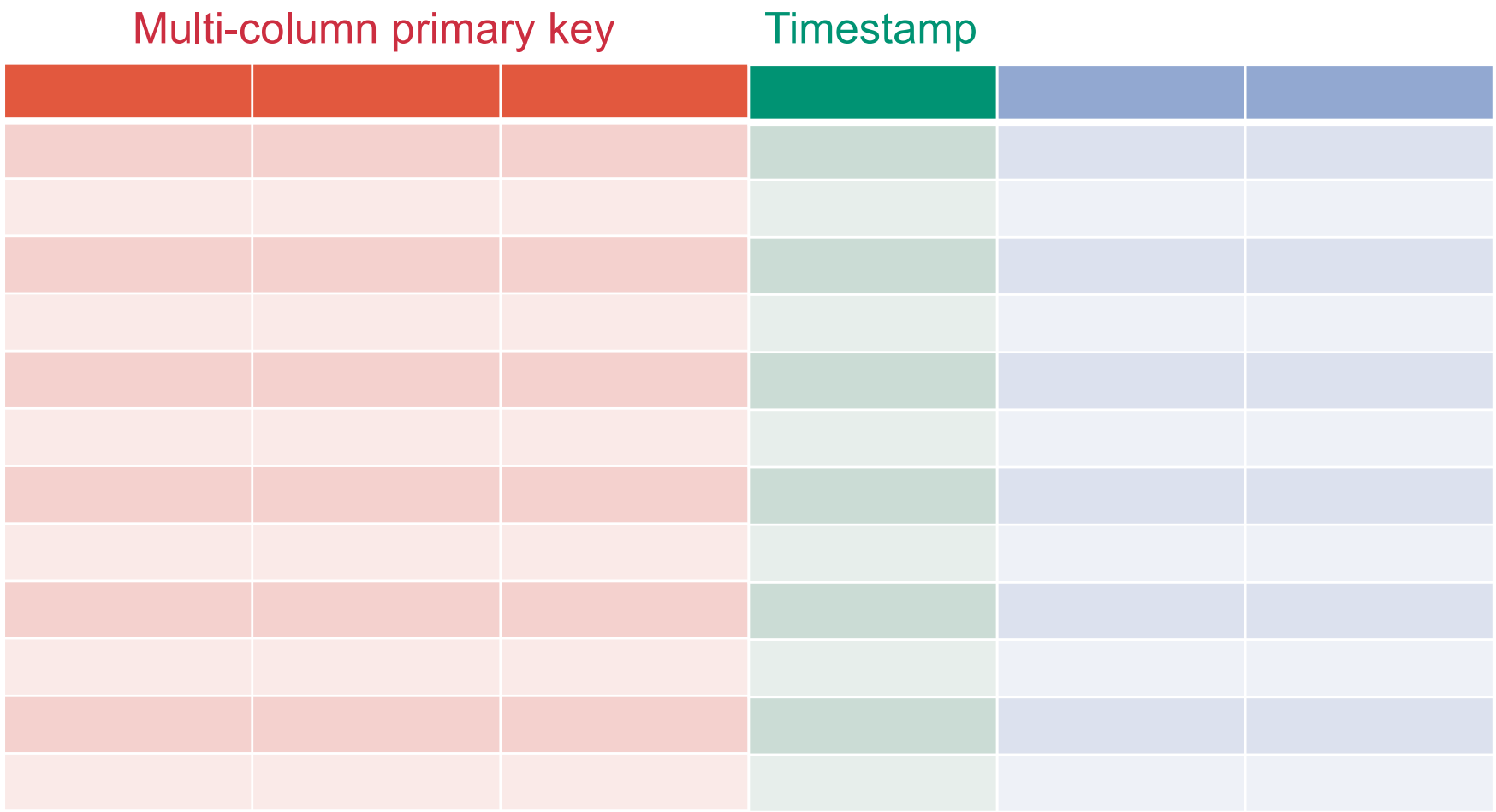
Sharding

Replicas

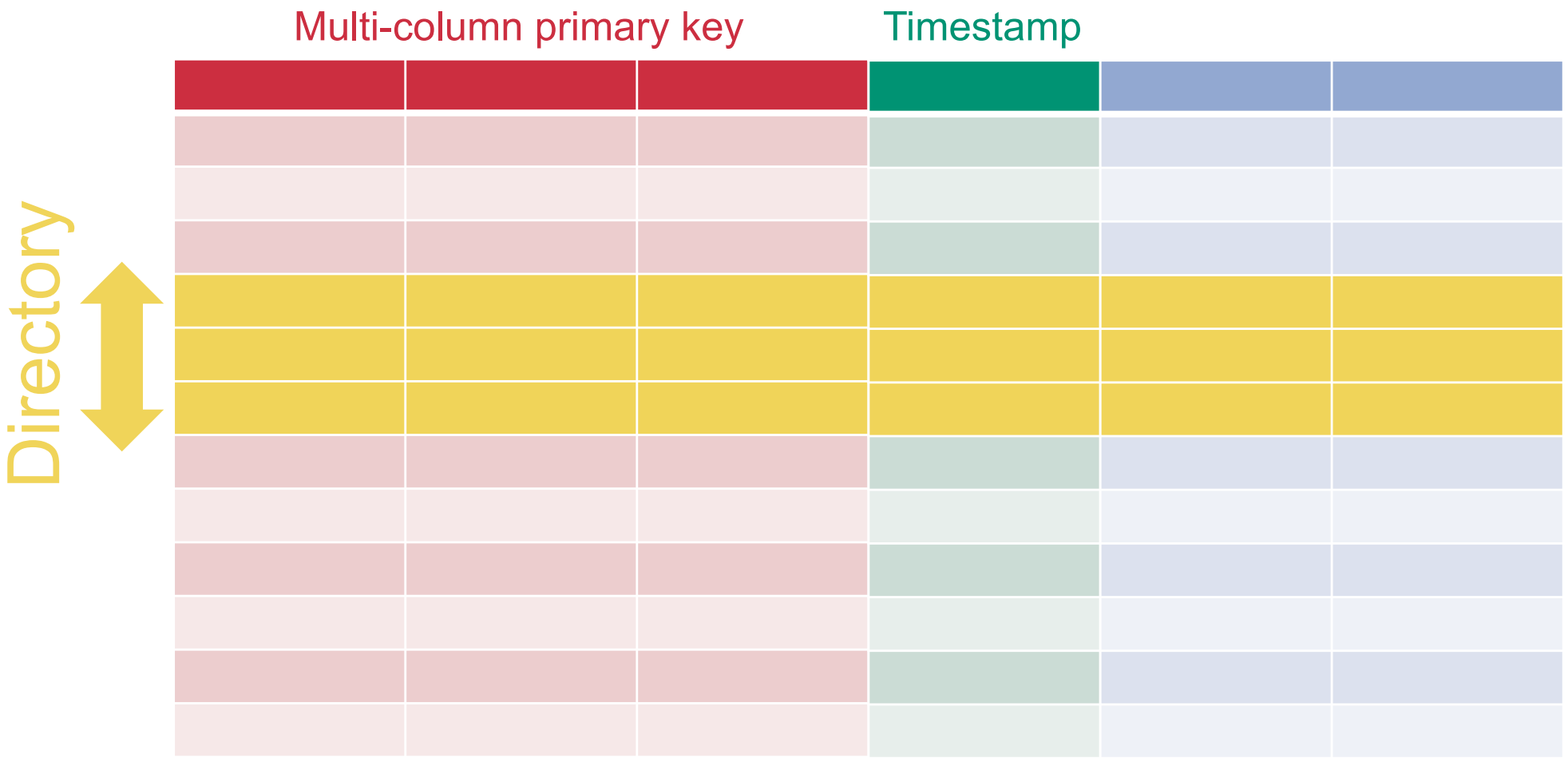
SQL

NoSQL

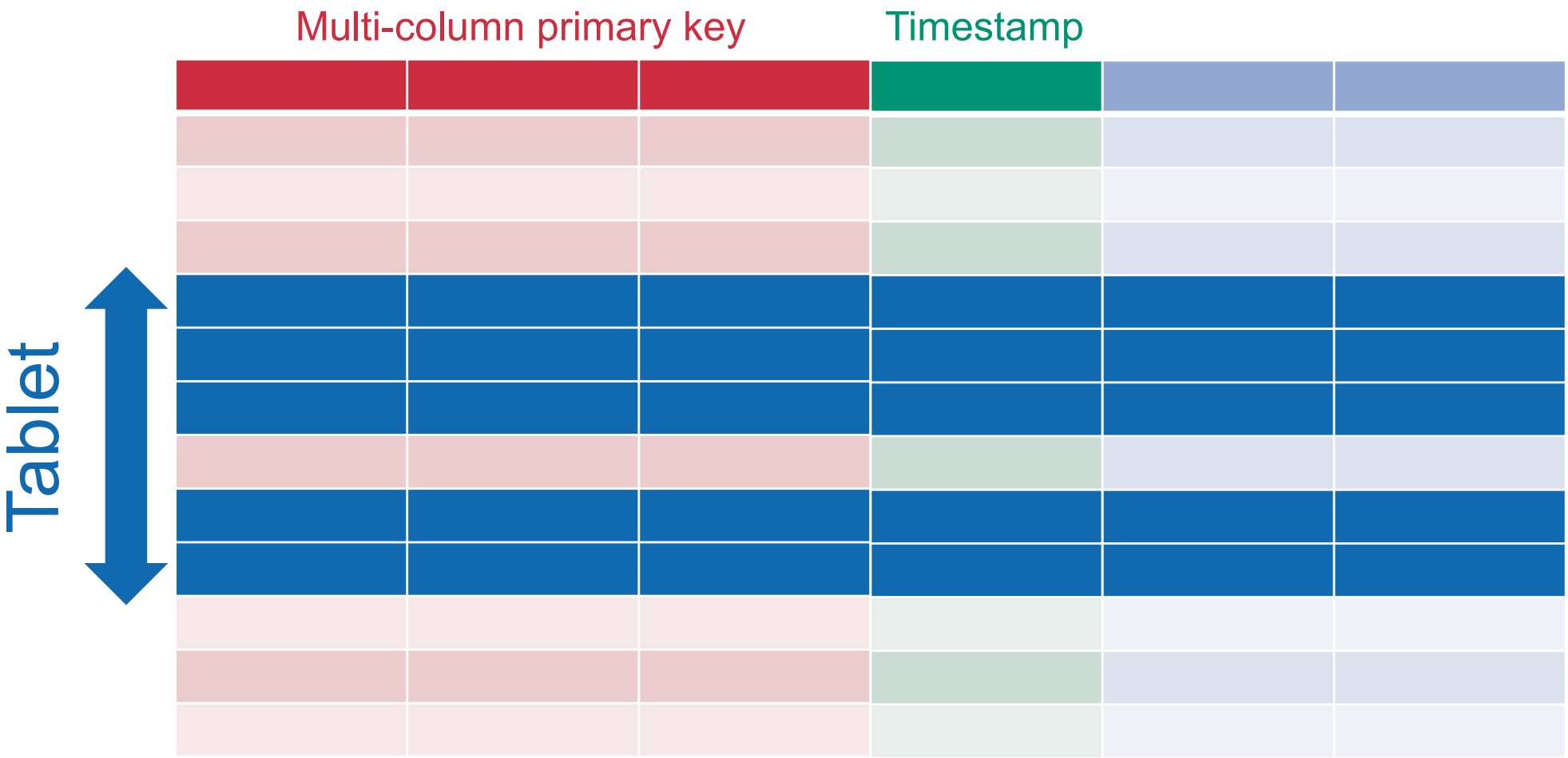
Spanner: Data Model



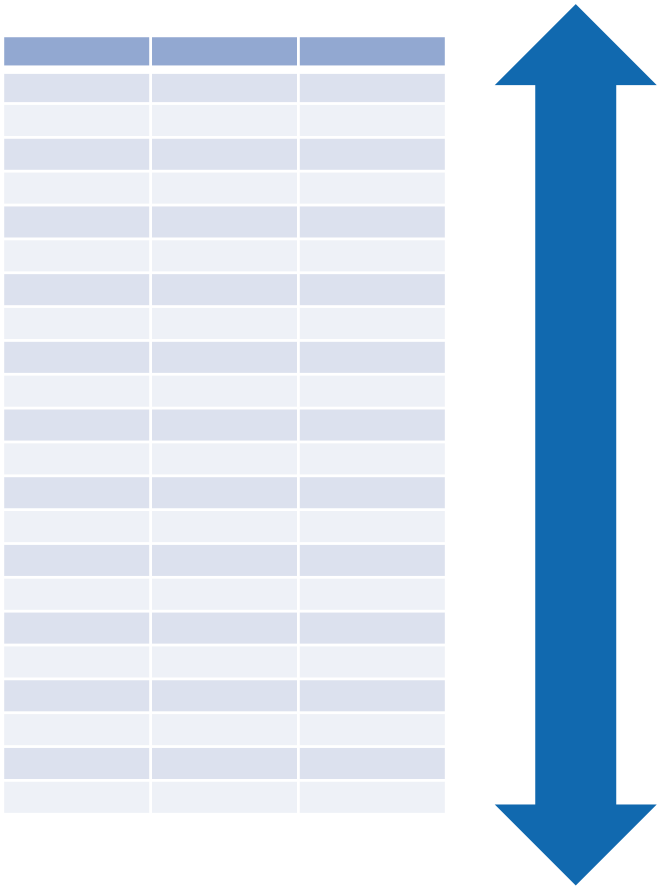
Spanner: Data Model



Spanner: Data Model

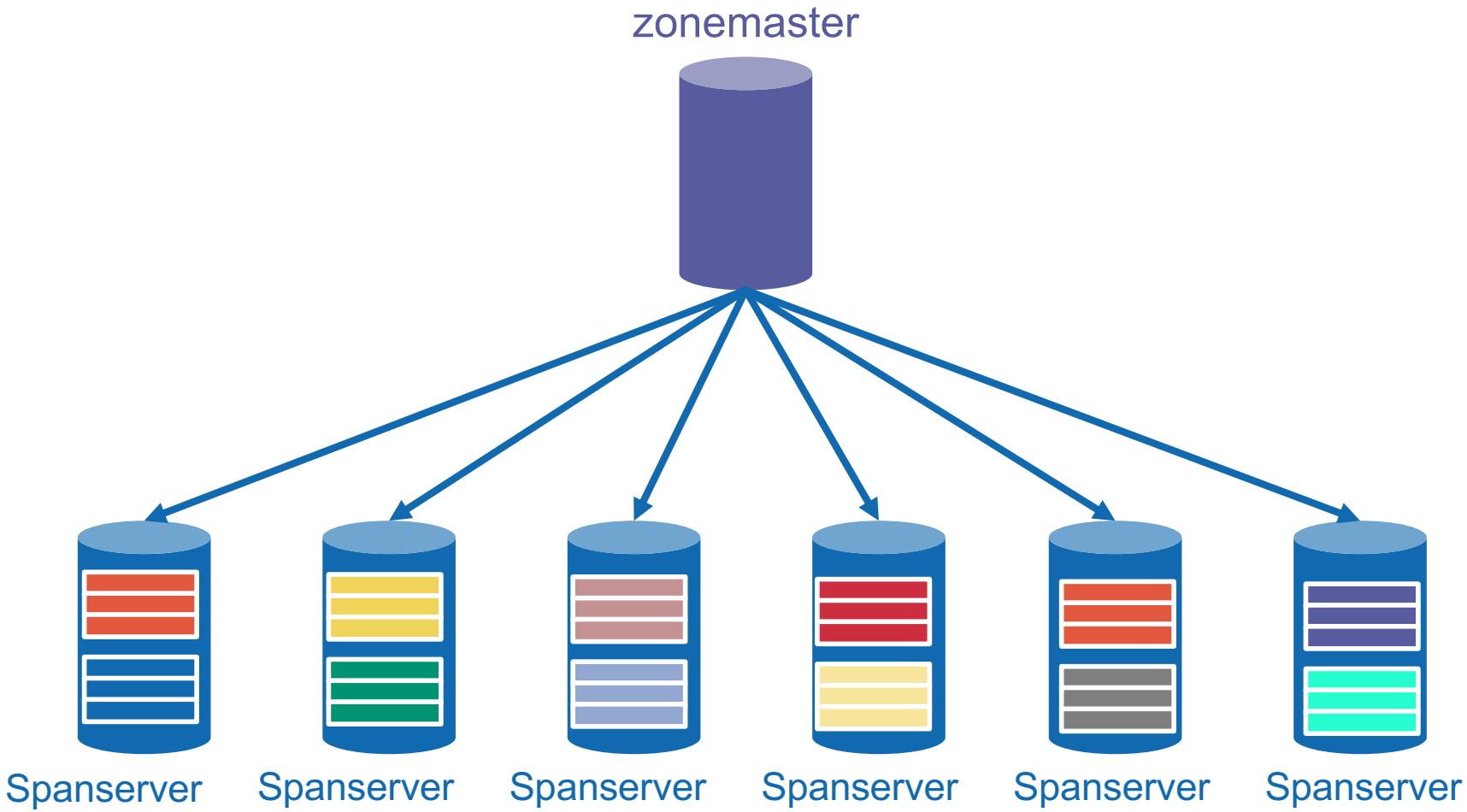


Spanner: Data Scale

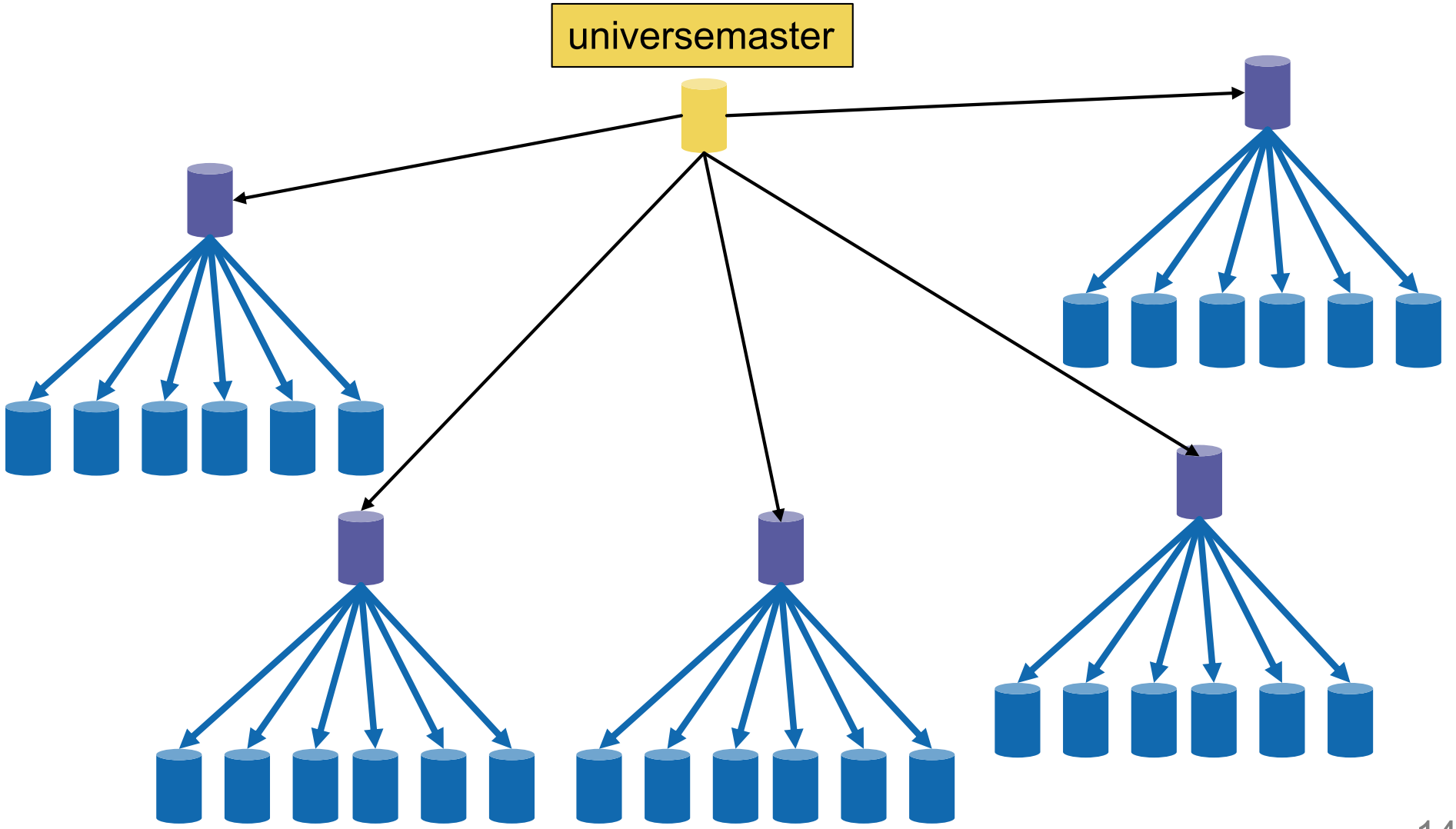


1,000,000,000,000s of rows

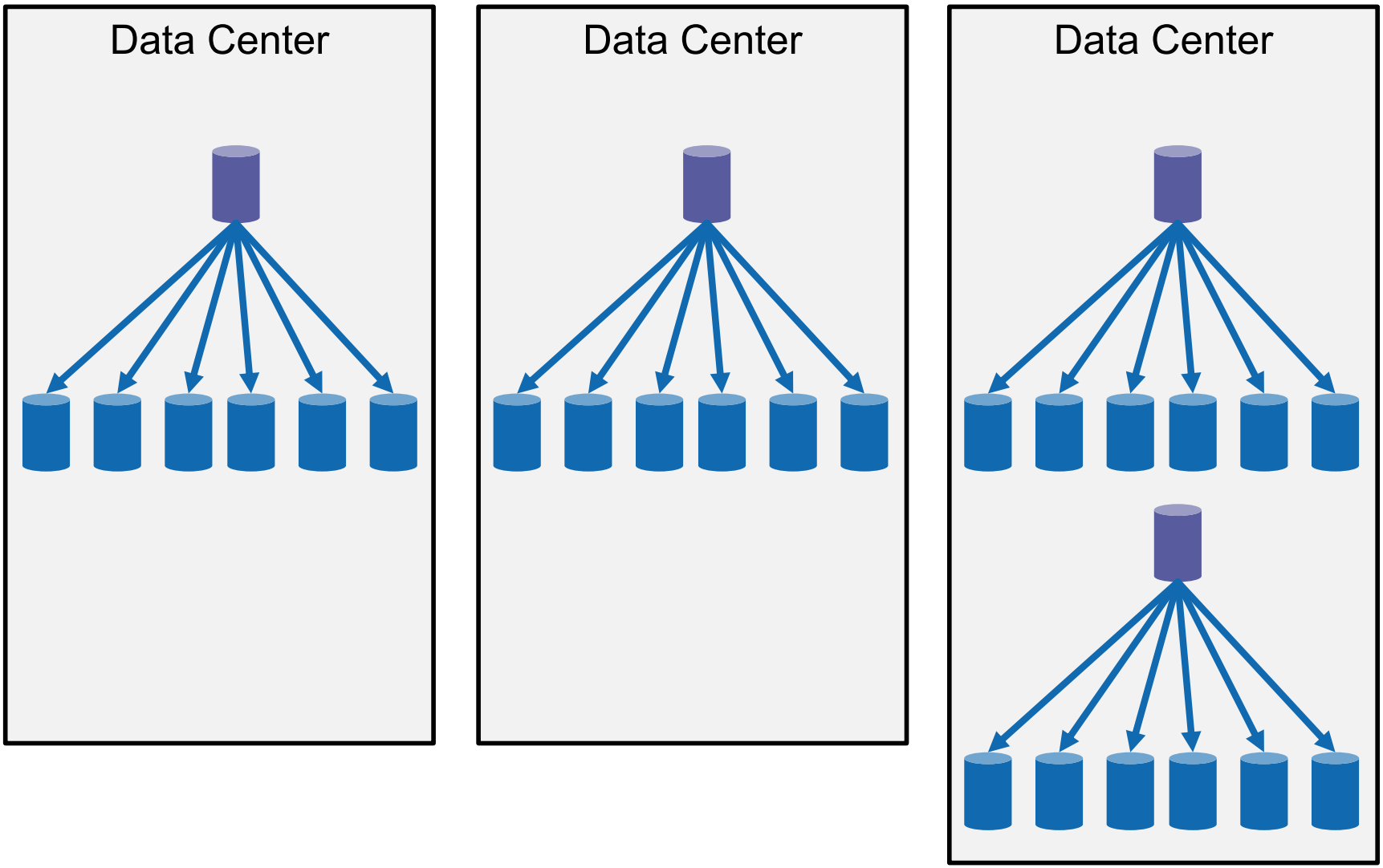
Spanner: Architecture of a Zone



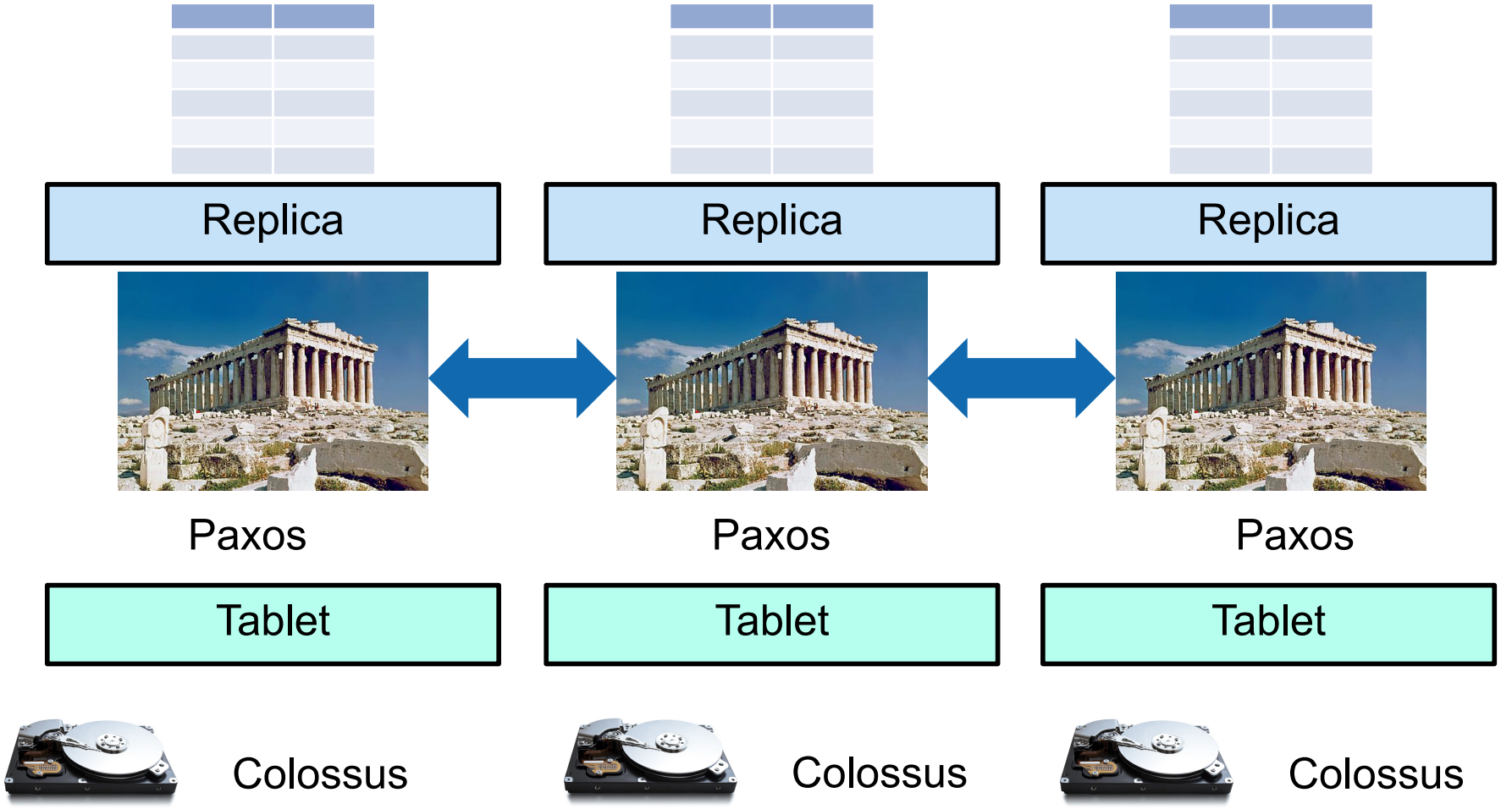
Spanner: Architecture



Spanner: Architecture

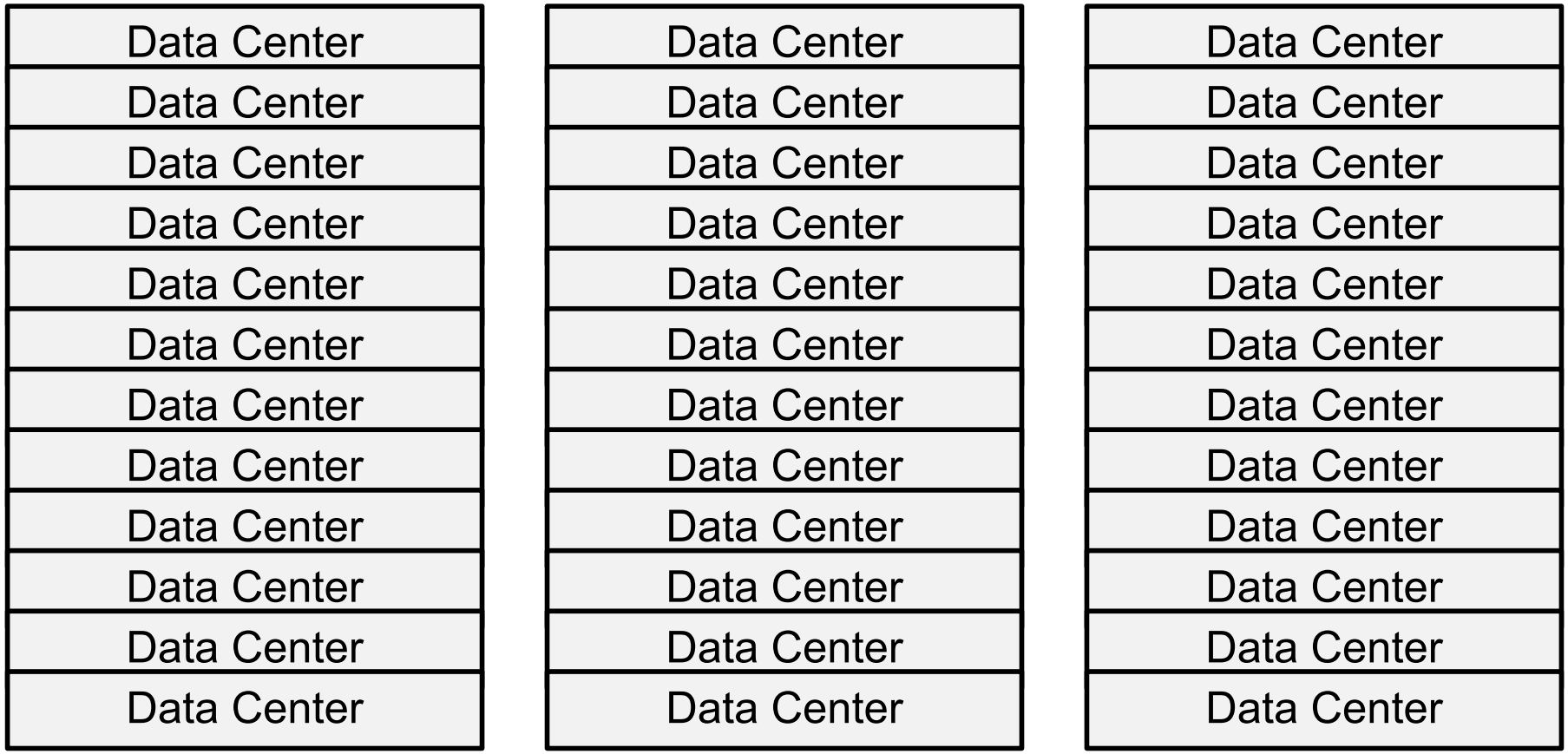


Spanner: Architecture



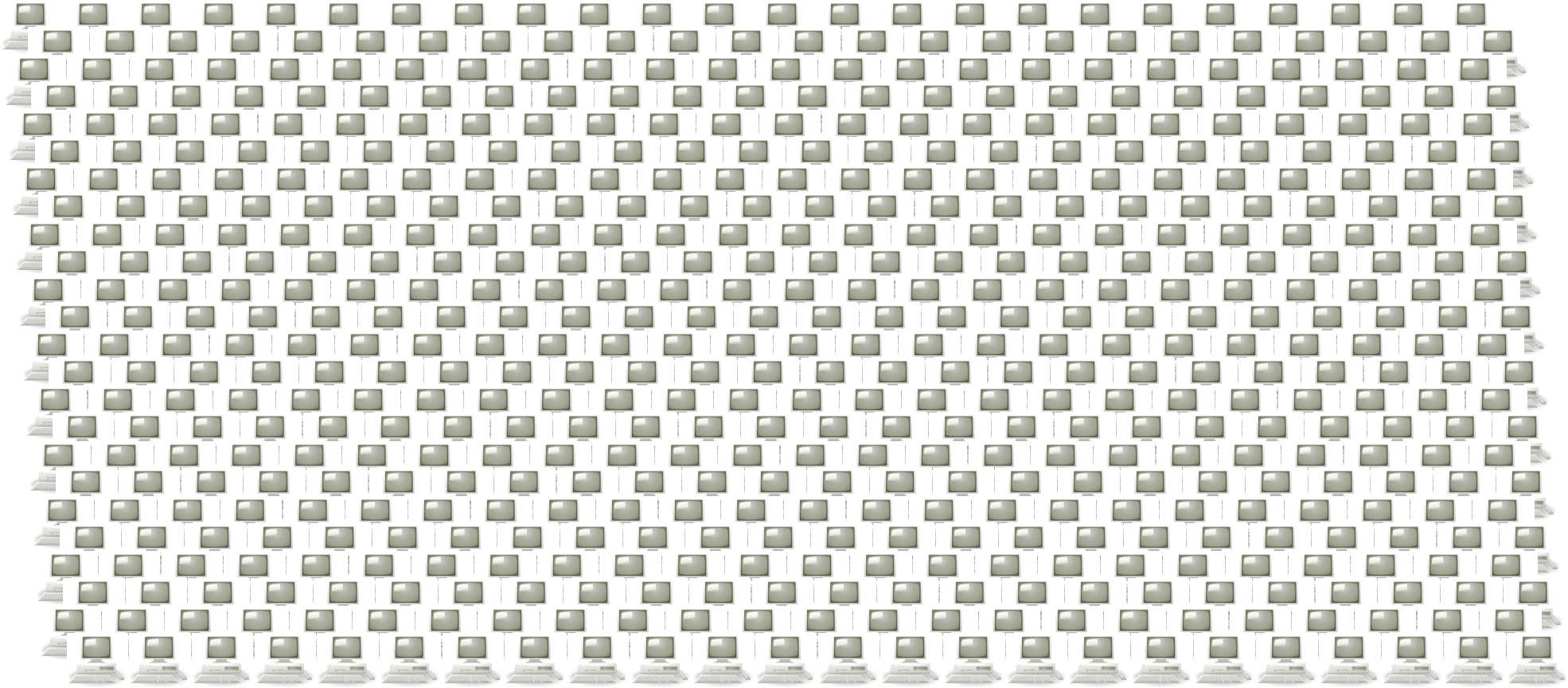
Spanner: Architecture

100s of data centers



Spanner: Architecture

1,000,000s of machines



Spanner: Architecture

~~Higher availability~~

Lower latency