Searching and Analyzing Information Inside Hadoop Platform

Abinasha Karana

25th Feb, 2013

Text Search, Range Search
Faceting, Sorting,
Aggregating

1000 columns, multi page
document in Billions

To Search a large dataset

What Didn't Work for US

Map-Reduce

Result not in a mouse click

What Didn't Work for US

Lucene

Lucene is a Java based search engine.
To handle large amount of records, the index is partitioned on a dimension and distributed to multiple machines.

Search Engine

Builds an index and answers queries using the index.

Read optimized using inverted index.

Non - Transactional

Database

Builds an index and answers queries Using the index.

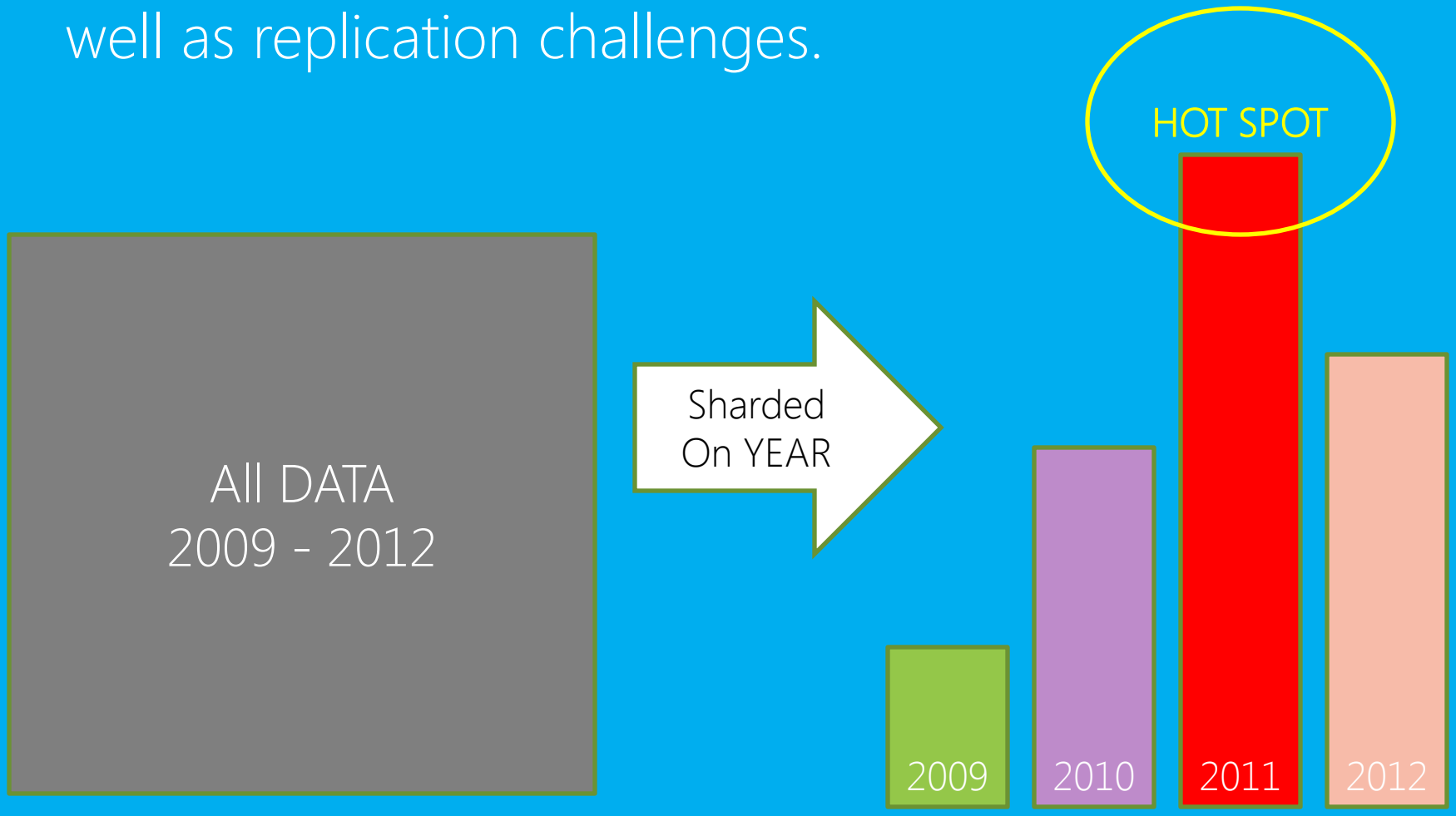
Write optimized

Transactional

Didn't work Because…

of hot spots in Shards.

Unequal shards leading to hot spotting as well as replication challenges.



What we did

We built a new search/analytics engine on HBase Platform

Leveraging HBase's auto-sharding and auto-replication

Using HBase…

HBase

...

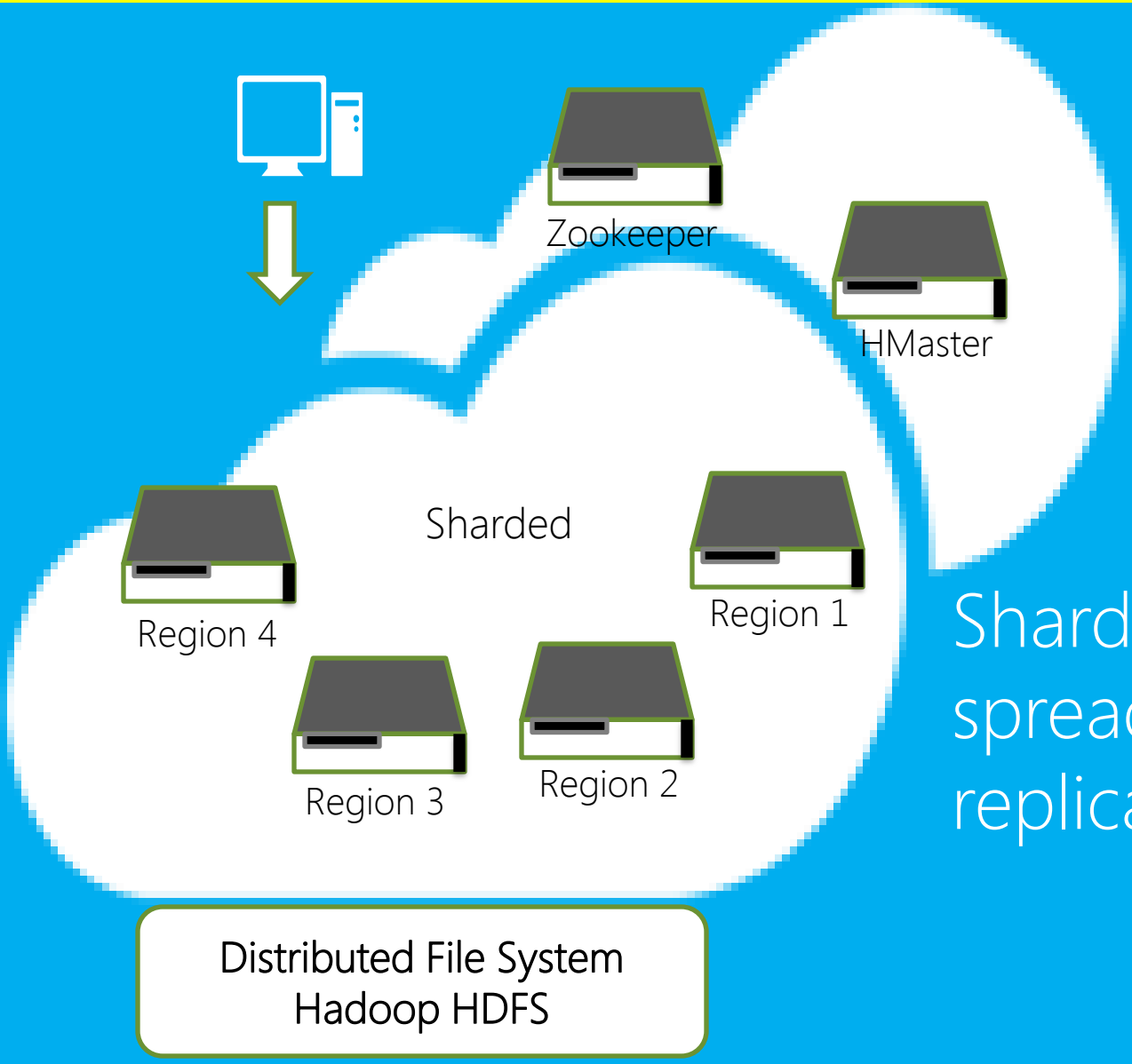
Hadoop Family Open Source columnar database modeled after Google Big Table.

Columnar Database (To match Cell2 Value, Just Load 12 Byes instead of 48 Bytes.

Cell1(int)	Cell2(Float)	Cell3(Float)	Cell4(Float)	Total Bytes
11	4.3	87.34	23.11	16 Bytes
12	8.9	91.12	19.00	16 Bytes
13	9.1	101.00	27.17	16 Bytes
12 Bytes	12 Bytes	12 Bytes	12 Bytes	48 Bytes

It is a distributed multi dimensional sorted map with each row having key value maps. Underlying Hadoop-HDFS data storage provides auto replication and auto sharding.

HBase shards the data automatically …



Sharding is just spreading and not replication/clustering

But HBase is designed for write heavy load…

In the next few slides you will hear about

My learning from designing, developing and benchmarking HSearch - a real-time search engine whose index is stored and served from HBase

https://github.com/bizosys/

HSearch Benchmarks

Version 2

Wikipedia Pages

- 100 Million Wikipedia pages of total 270GB and no stopwords.
- Data generated by repeating 10 Million Pages 10 Times.
- Search Query Response (Id + Teaser)

1. Regular word 1.5 Sec.
2. Common word such as hill found 1.6 million matches and sorted in 7 secs.

Amazon Large instance 7.5GB memory * 11 machines with a single 7.5 K SATA drive

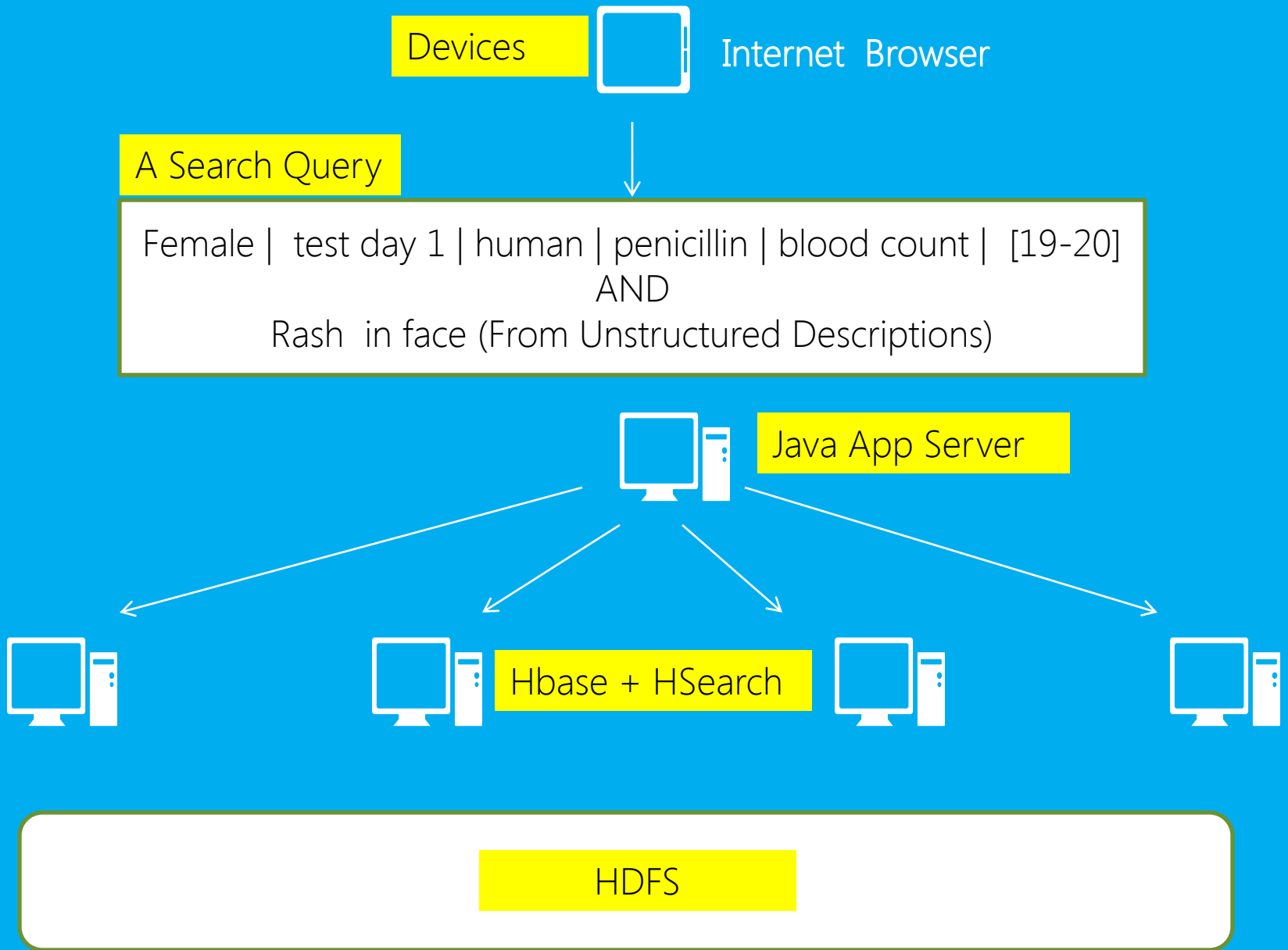
Version 3

@ Leading Pharama Research

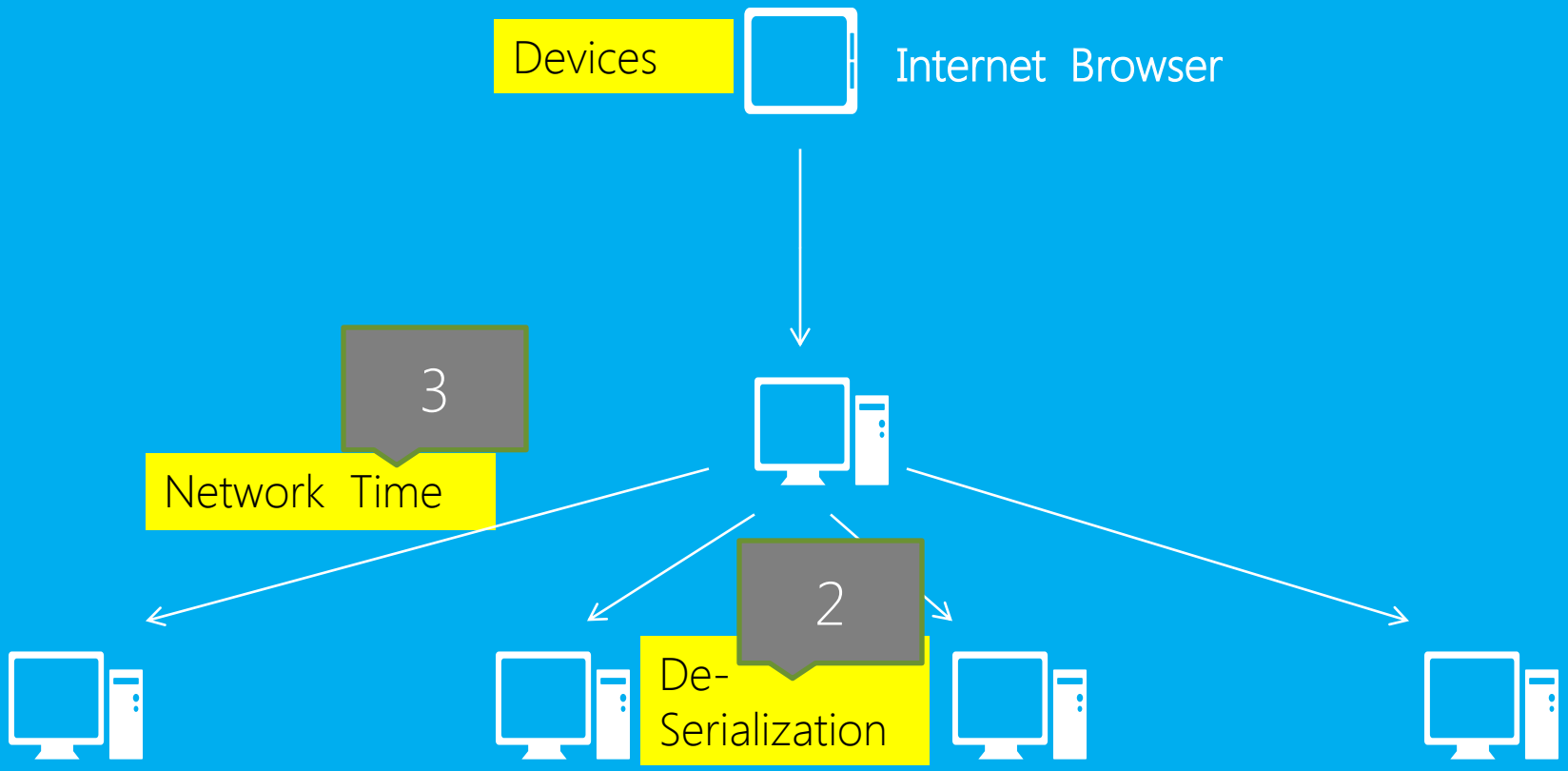
1. Table Size : 1.2 Billion rows * 800 columns + 1.2 Billion Observation data.
2. A complex query returned 1.4 Million matched rows in 600ms
3. Indexing time 8 Hours.

Amazon Large instance 7.5GB memory * 4 machines with a single 7.5 K SATA drive

HSearch architecture

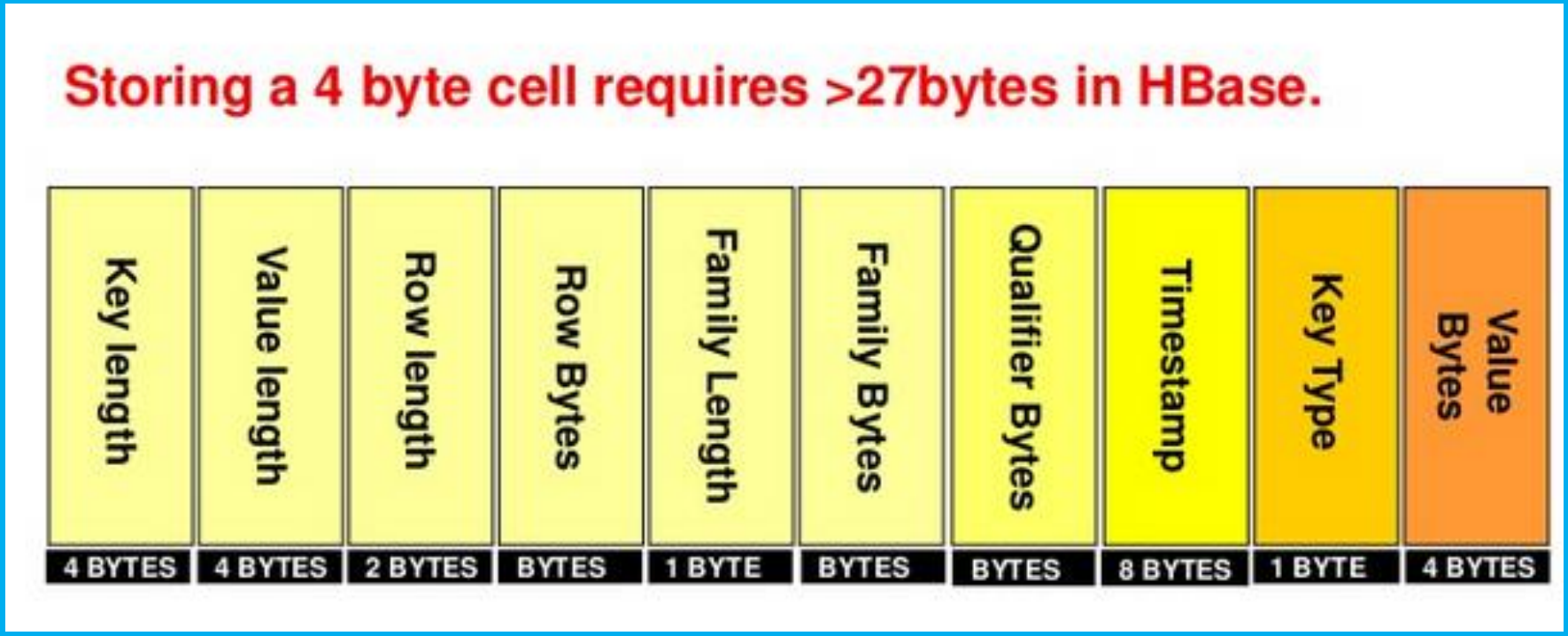


Where we slowed Down



Time spent on reading from disks…

Strategy Applied : Club records to save metadata overhead



1. Stored large cells by merging multiple cols/rows
2. Used a single character as family name
3. Reduced the qualifier name to 1 character.

Time spent on reading from disks…

Strategy Applied : Using SSD to read faster

SSD improved HSearch response time by 66% over SATA.

However, SSD is costlier

We used SSDs for Index only.

Serialization – De-Serialization …

Strategy Applied : De-Serialized needed segments

Student	Mark
001	91
008	92
002	93
007	98

16 Bytes

16 Bytes

91	92	93	98
001	008	002	007

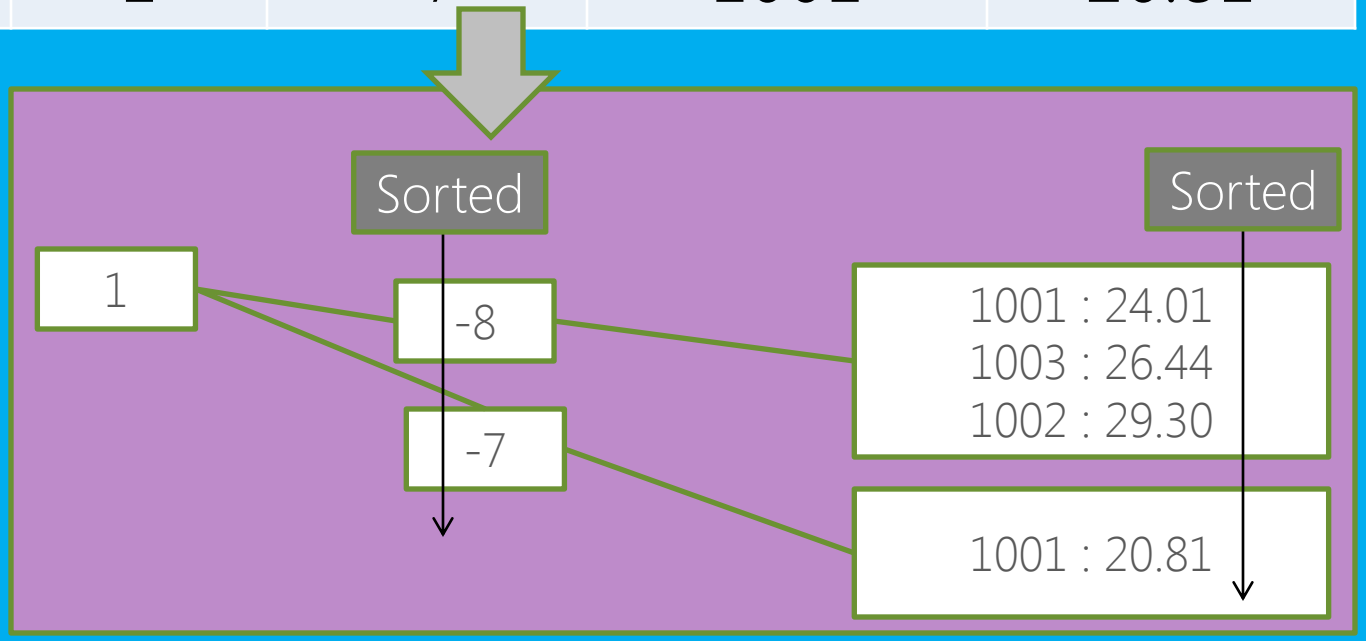
Match on Location Index

De-Serialize 16+4 Bytes to find the student index(2) scored 93 marks.

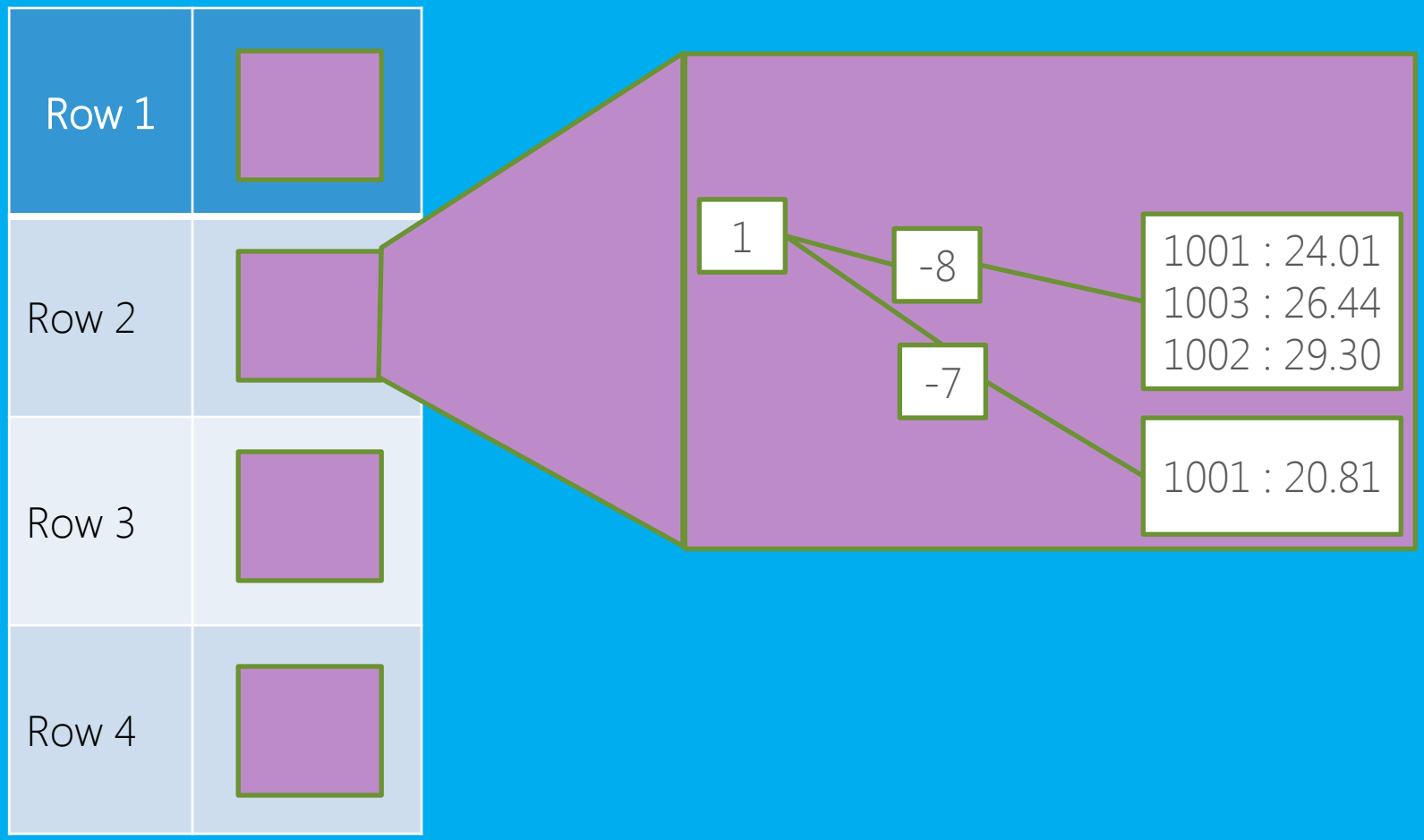
Further optimize using binary search on Byte Arrays

From multiple tabular records - To sorted tree structure

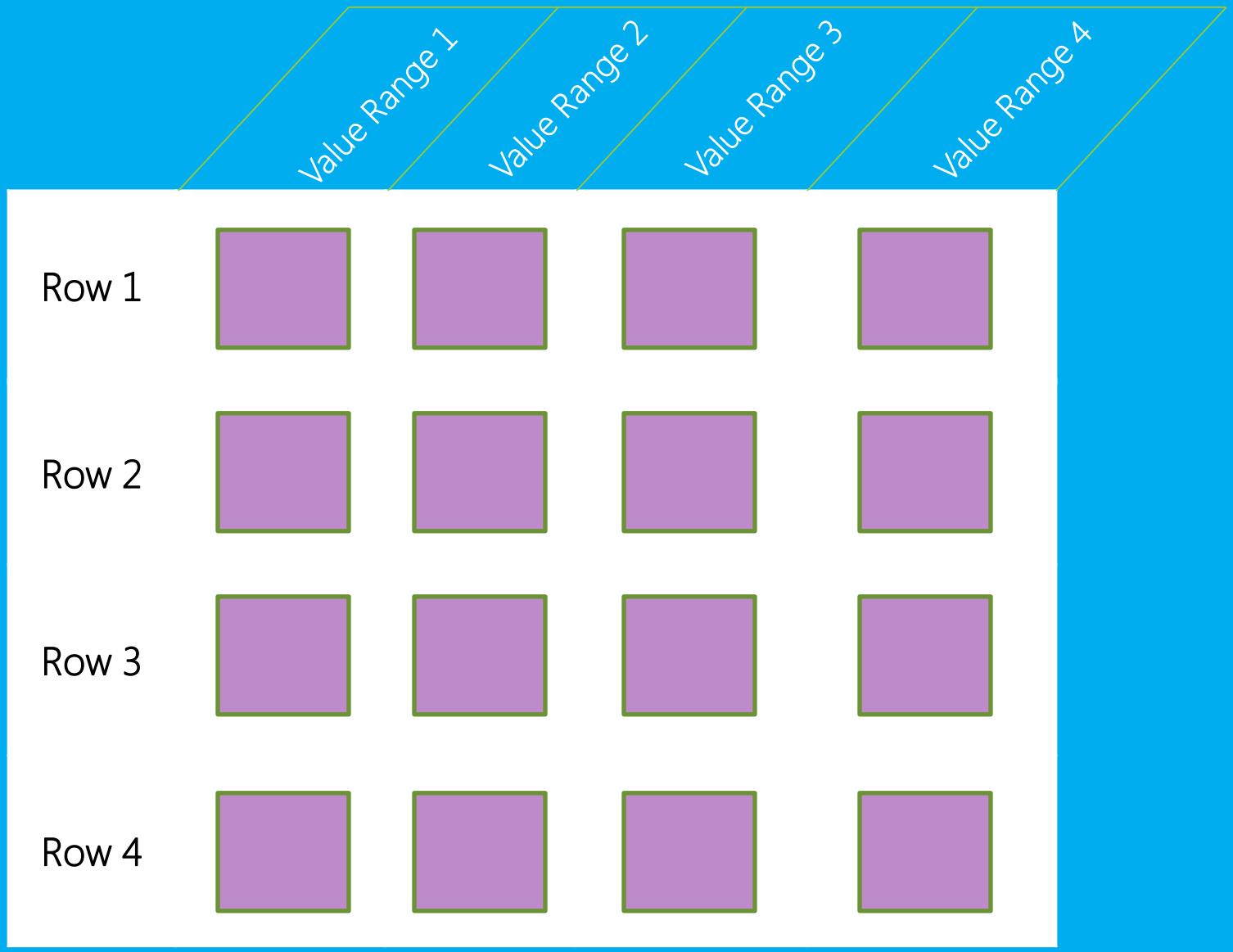
Tree Id	Cell2	Cell3	Cell4	Cell5
2	1	-8	1001	24.01
2	1	-8	1003	26.44
2	1	-8	1002	29.30
2	1	-7	1001	20.81



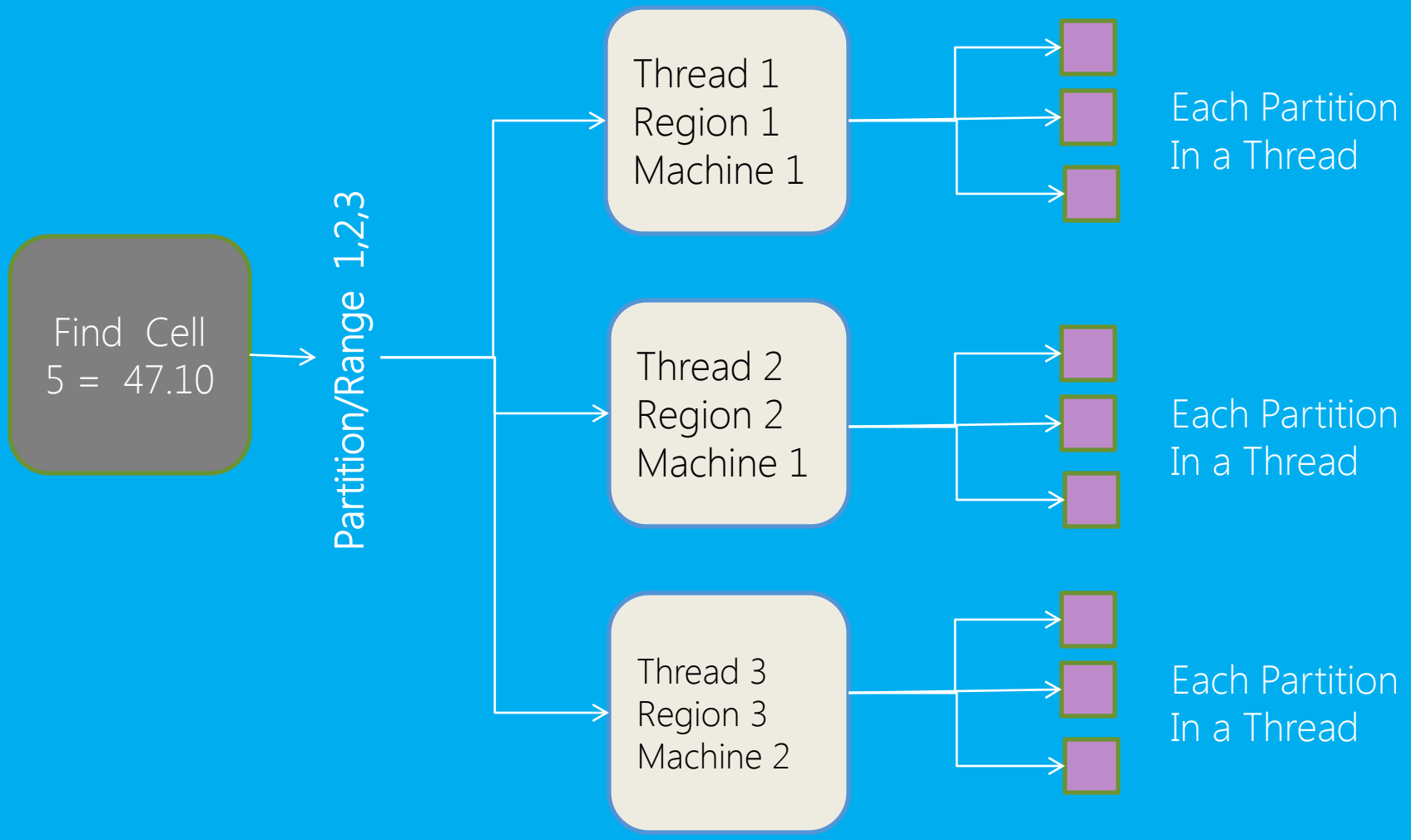
where each root level node is serialized to form a HBase Cell



Inverted Index – Enter By Value and Not Key.

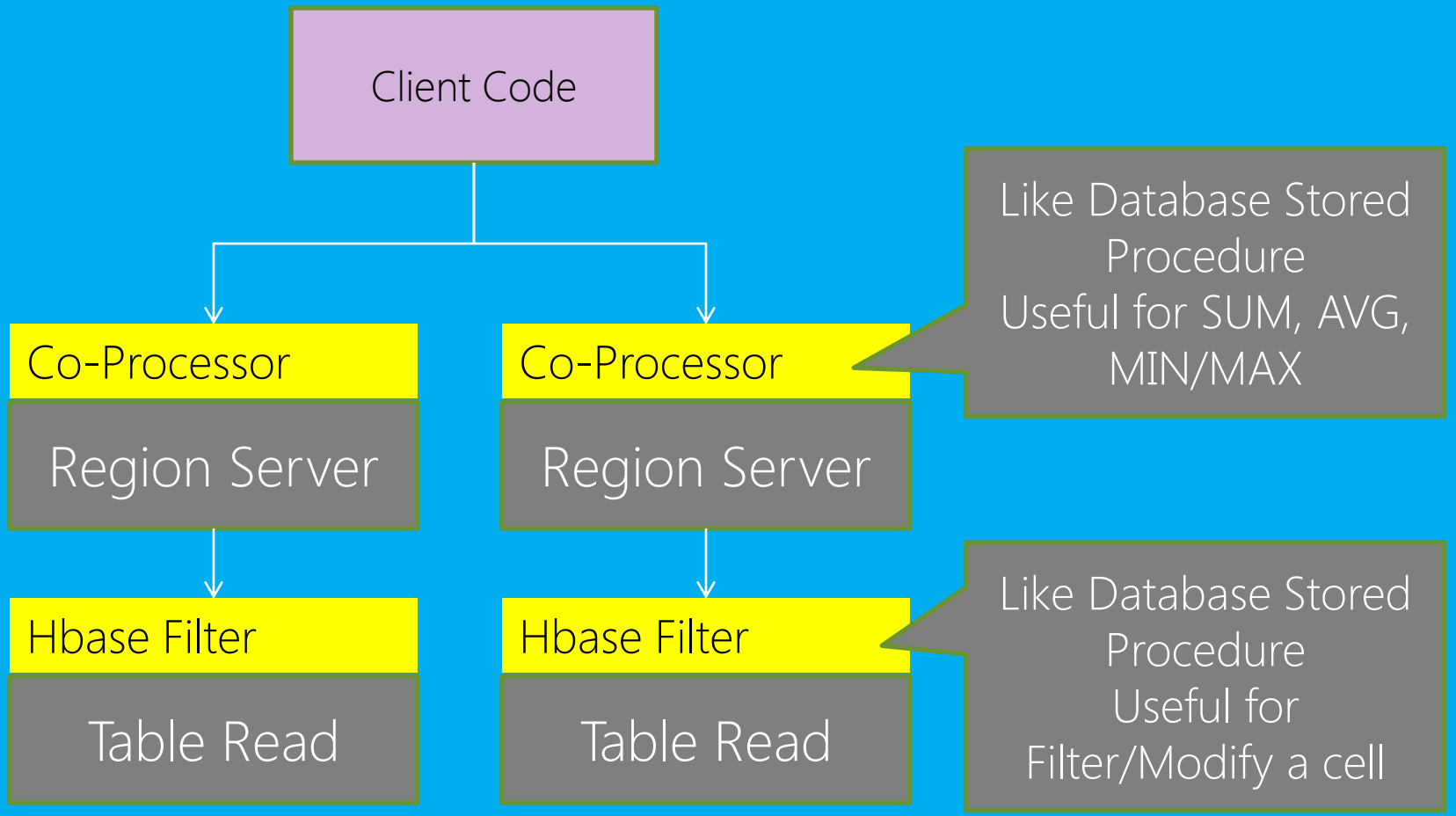


… with parallel processing of Bytes Blocks for each region servers.



Network Time

Processing moved near to DATA: Filter and Coprocessors



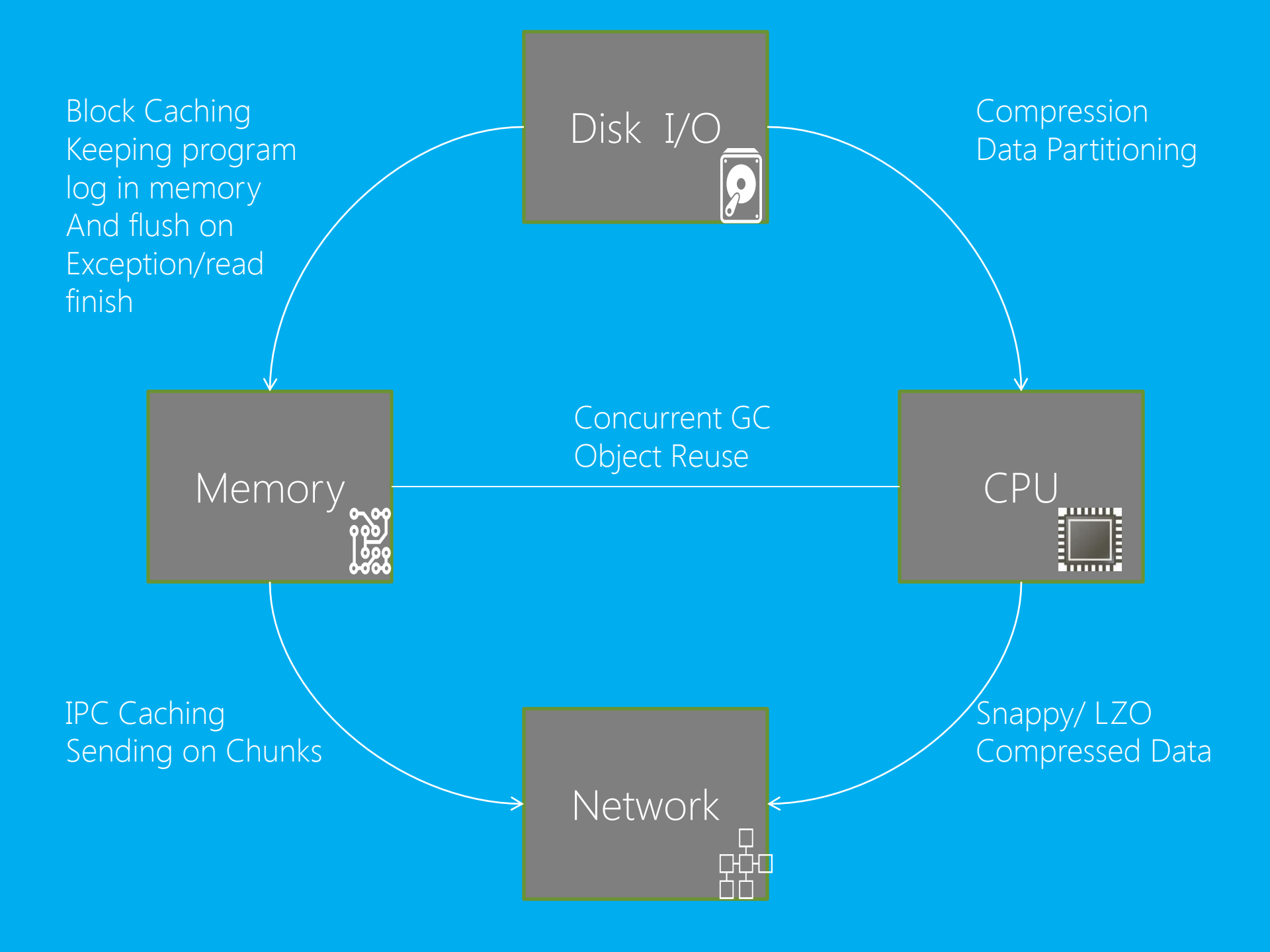
Network Time

Strategy Applied : Bytes Block Caching

Object Cache = 7 bytes + 56 bytes (pointer)

Bytes Cache = 7 bytes + 8 bytes (pointer)

To process Big Data in small time, it is needed to balance Network vs CPU vs I/O vs Memory while leveraging multiple machines.


Block Caching
Keeping program log in memory
And flush on Exception/read finish
Disk I/O
Compression
Data Partitioning
Concurrent GC
Object Reuse
Memory
CPU
IPC Caching
Sending on Chunks
Network
Snappy/ LZO
Compressed Data

And It's Configuration…

- Network
  - Increased IPC Cache Limit (hbase.client.scanner.caching)
- CPU
  - JVM agressive heap ("-server -XX:+UseParallelGC -XX:ParallelGCThreads=4 XX:+AggressiveHeap ")
- I/O
  - LZO index compression ("Inbuilt oberhumer LZO" or "Intel IPP native LZO")
- Memory
  - HBase block caching (hfile.block.cache.size) and overall memory allocation for data-node and region-server.

and parallelized to multiple machines… `

- Htable.batch (Sending/Receiving data from Region Servers in chunk)
- Parallel Htable (Multi threaded Scans)
- Co-Processors, Filters

Allocating appropriate resources

dfs.datanode.max.xcievers,
hbase.regionserver.handler.count and
dfs.datanode.handler.count

THANK YOU