Separating hot-cold data into heterogeneous storage based on layered compaction

Allan Yang（HBase Committer）

Content

01 Typical Scenarios at Alibaba

02 Hot-cold Data Separation

— Hot-cold Data Recognition

— Layered Compaction

— Query Optimizations

03 Conclusions

01 Typical Scenarios at Alibaba

Typical Scenarios at Alibaba

Contacts&Chat

AI BOTs

Risk Control

All Powered by AliHB (an Alibaba branch of HBase)

Logistics tracking

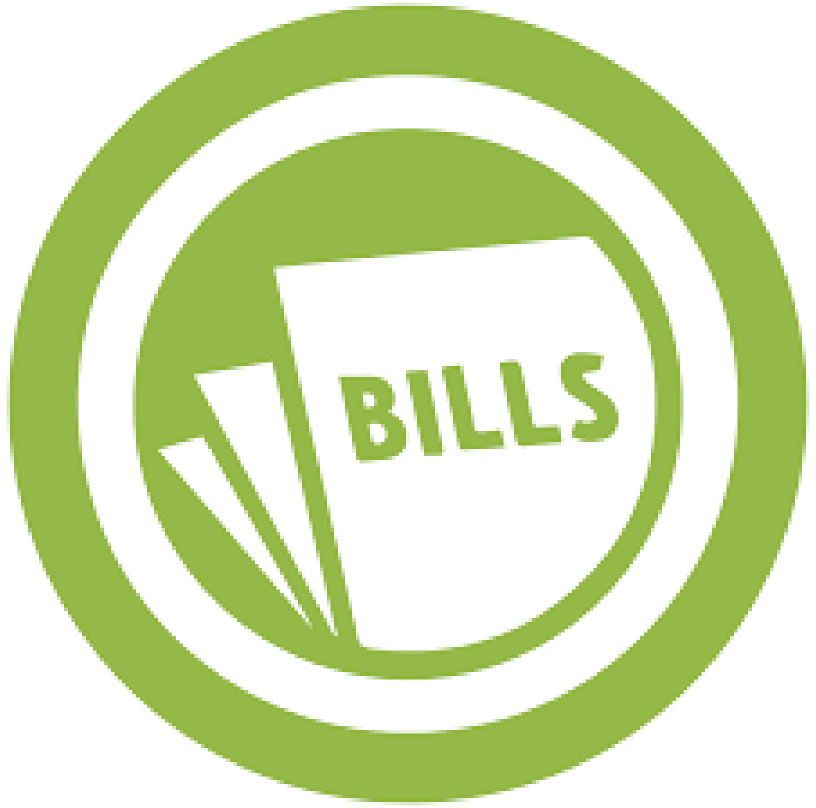
Bills

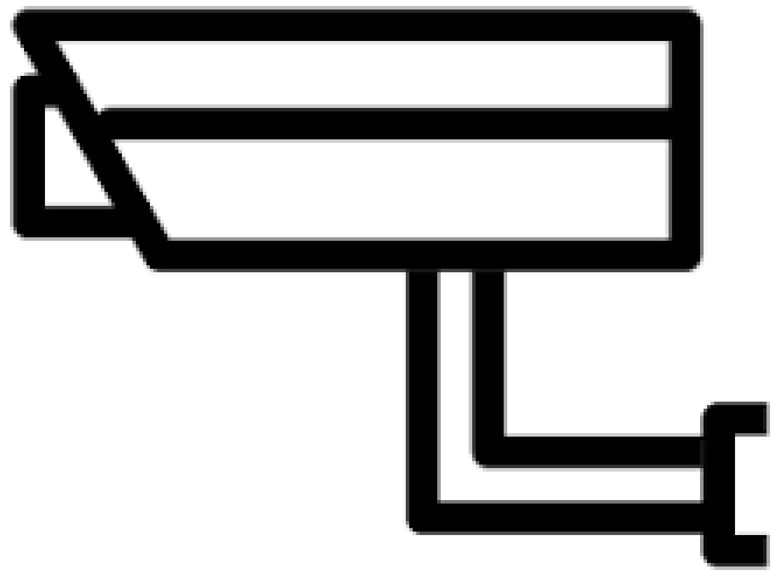
GMV

Typical Scenarios

Commonality in some Scenarios

- ☐ Mass data
- ☐ No TTLs
- ☐ Only very small parts of data is frequently visited
- ☐ Hotspots change as time goes by





Definitions

HOT

COLD

Hot Data

- Access very frequently
- Relatively small amount
- Low latency is very critical

Cold Data

- Access rarely
- Big amount
- Cost is more concerned

02 Hot-cold Data Separation

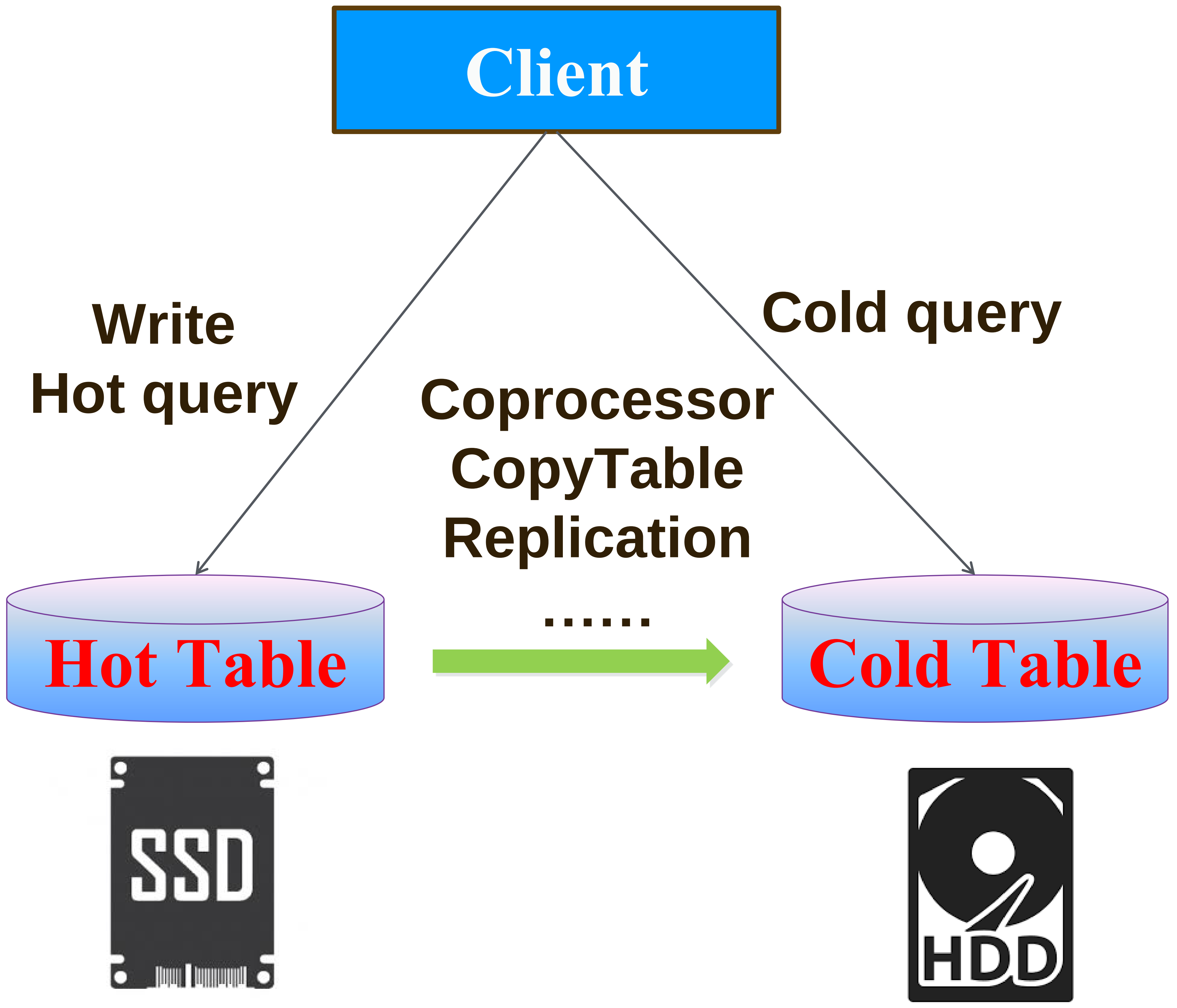
Old Architecture

Pros

- Simple, no HBase code change needed

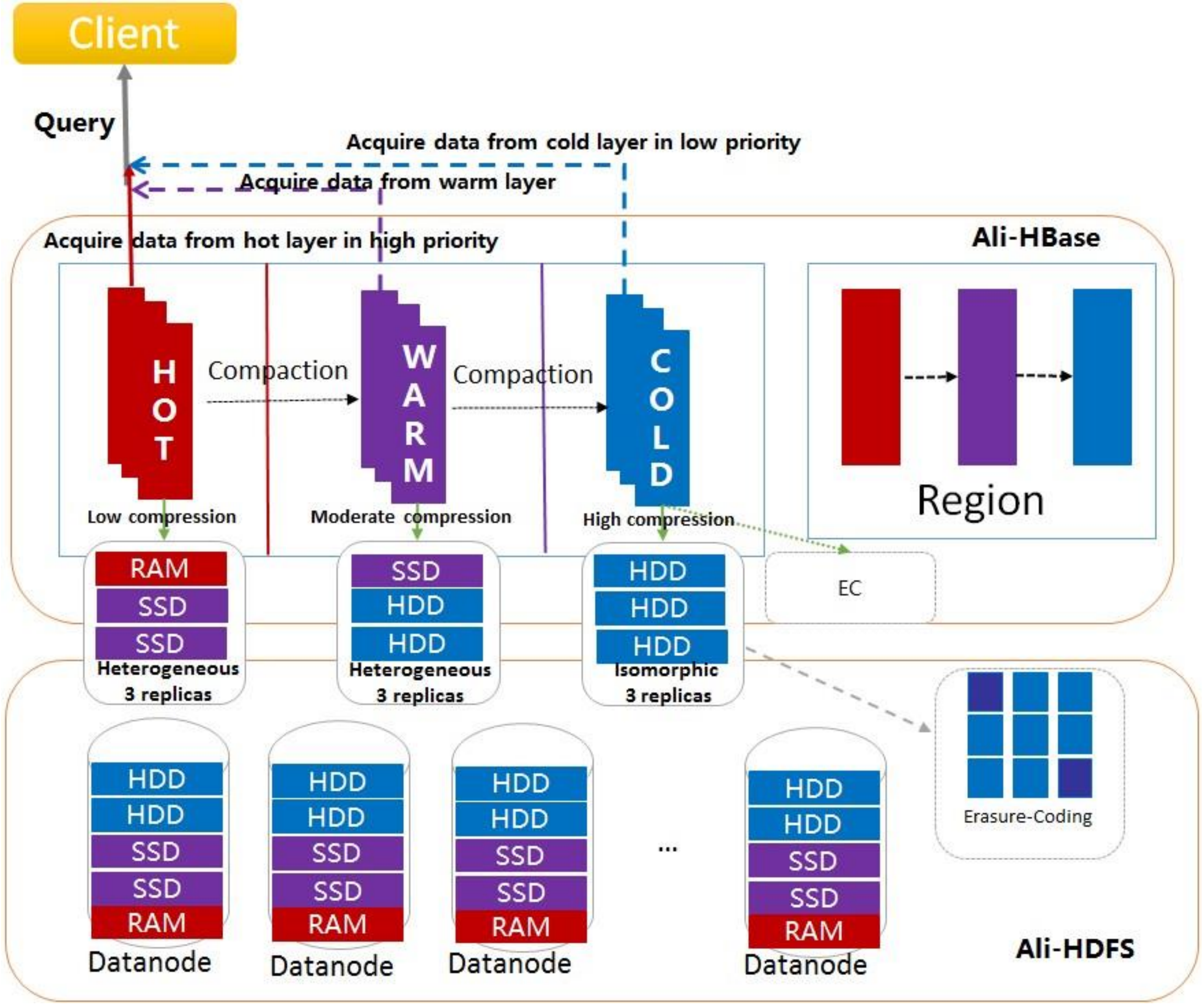
Cons

- High maintenance cost
- Client aware
- Hard to keep consistency



Current Architecture

- Separating hot cold data automatically in a single table
- Transparent to user
- Different storage policy for each layer
- Auto query optimization



Hot-cold Data Separation

— Hot-cold Data Recognition

— Layered Compaction

— Query Optimizations

Separating hot cold data

The Problems of separating data by KV timestamp

- Timestamp may not represents the heat of business data very well

e.g. Write an order ID advance in current ts
e.g. Data Source(Kafka, Spark…) delayed, resulting ts lag

- KeyValue’s timestamp is also used as version number in HBase

Secondary Field slicer

Besides ts, we provide a way to parse a Secondary Field from Rowkey, use it as the boundary of Hot/Warn/Cold data

- FixPosFieldSlicer

Rowkey:

16bit UserID	64bit timestamp	Postfix
Fixed size prefix	Secondary Field	

- DelimiterFieldSlicer

Rowkey:

Variant prefix	#	64bit timestamp	Postfix
Variant size prefix	Delimiter	Secondary Field	

Hot-cold Data Separation

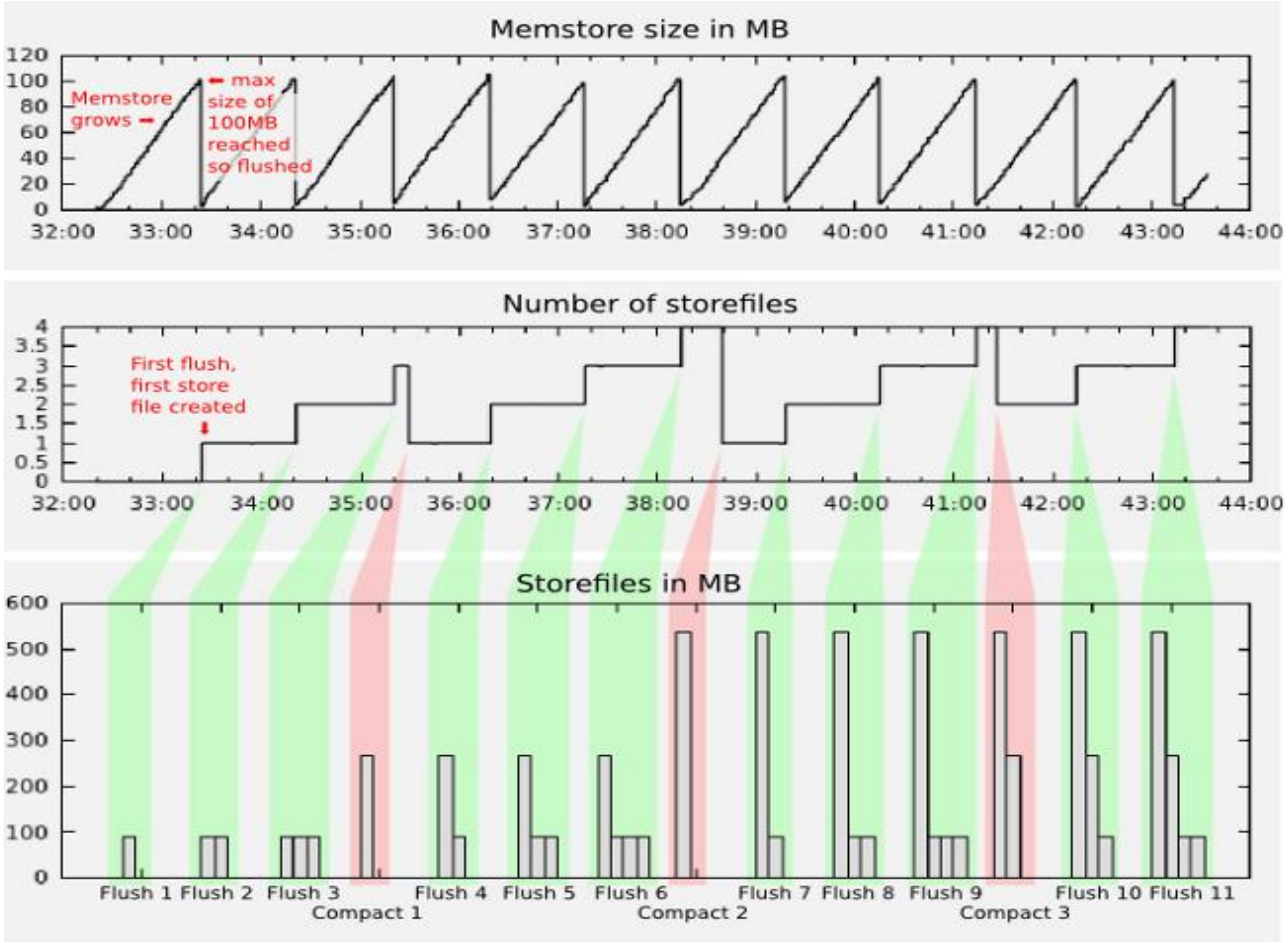
— Hot-cold Data Recognition

— Layered Compaction

— Query Optimizations

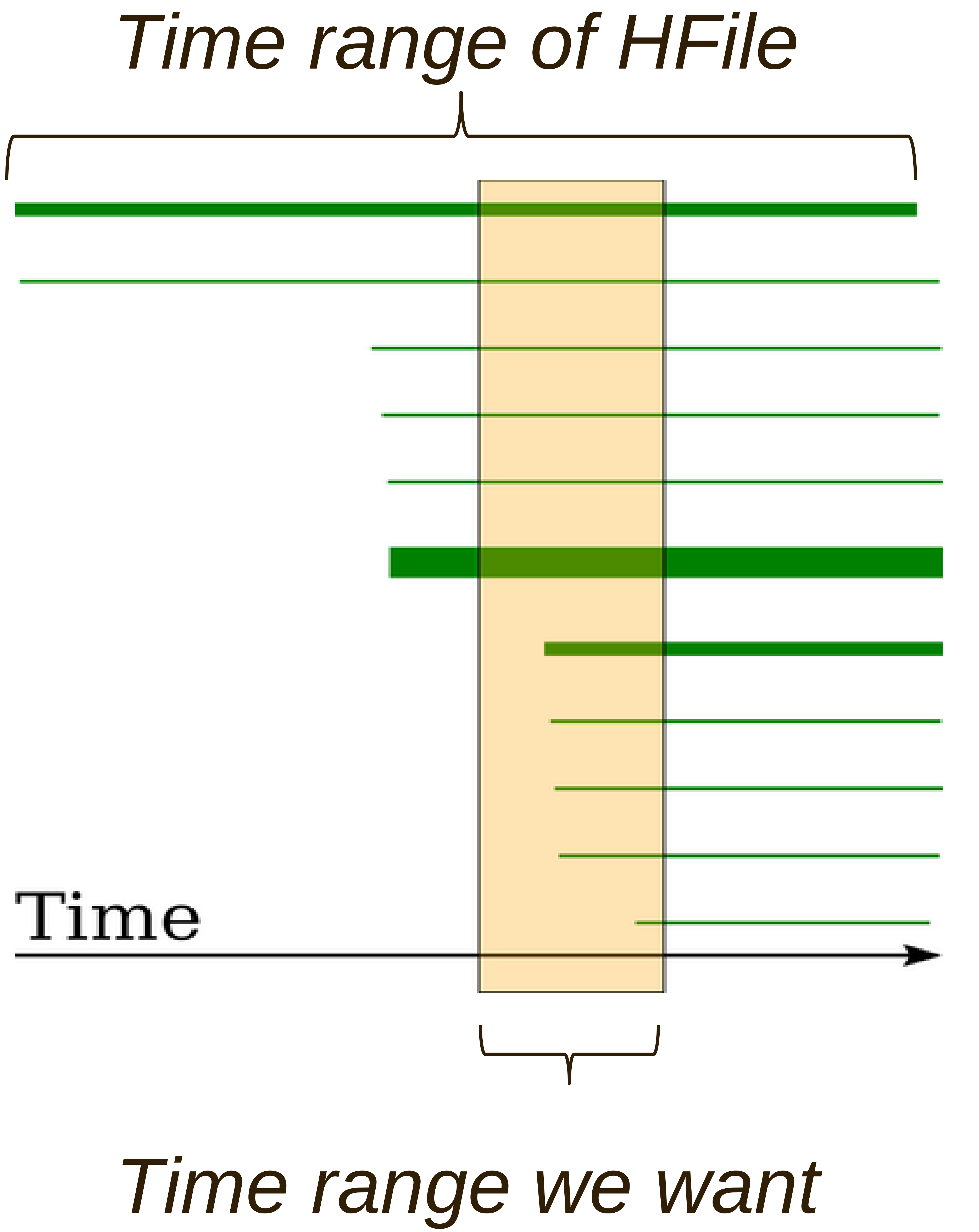
Default compaction in HBase

Default HBase compaction Strategy is Size-Tiered, which is aimed to compact small files to bigger files.



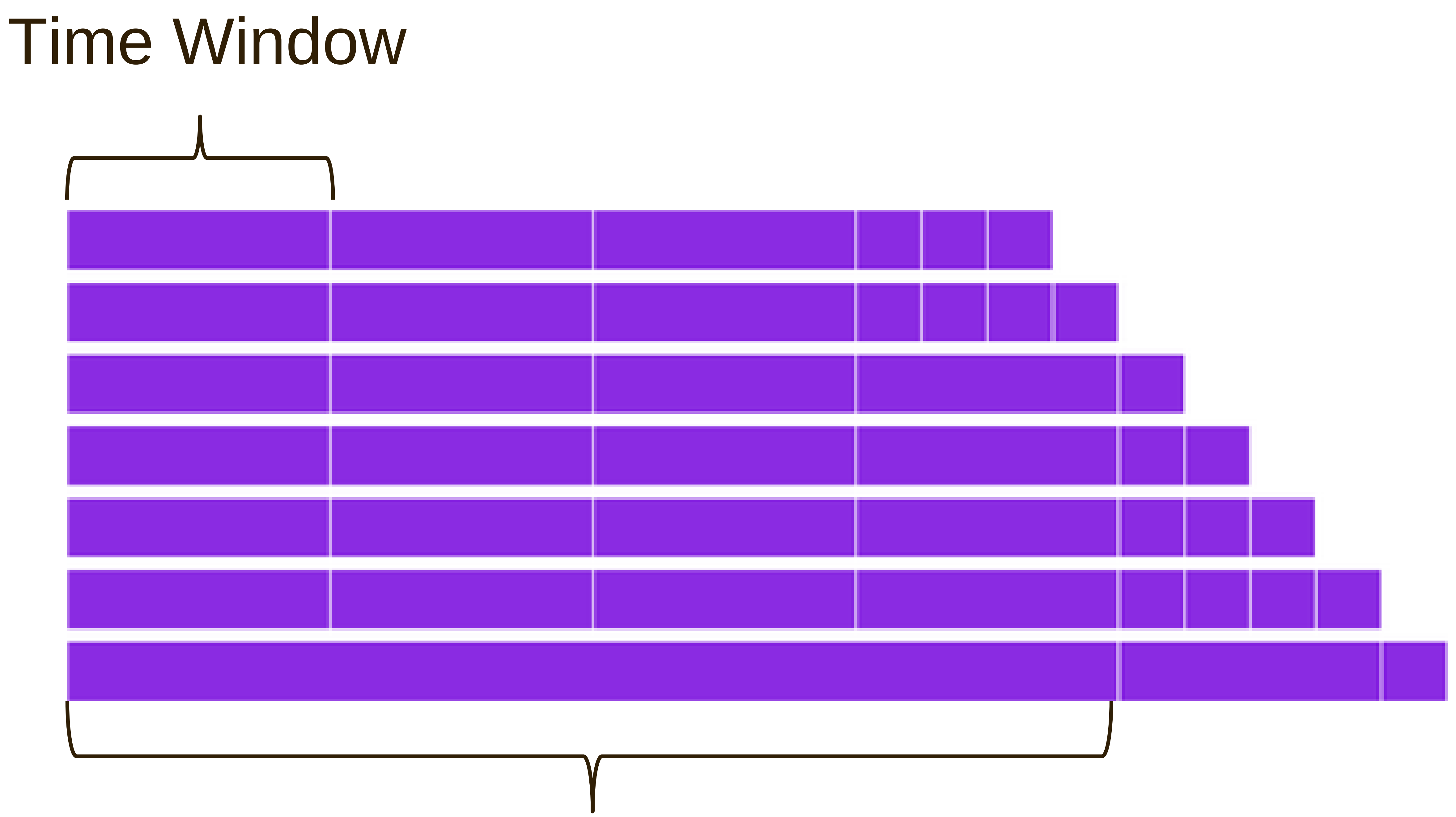
Size-Tiered Compaction Strategy

- Size is the only concern
- Old data and new data will spread around all HFiles
- Can’t be used for Separating hot cold data

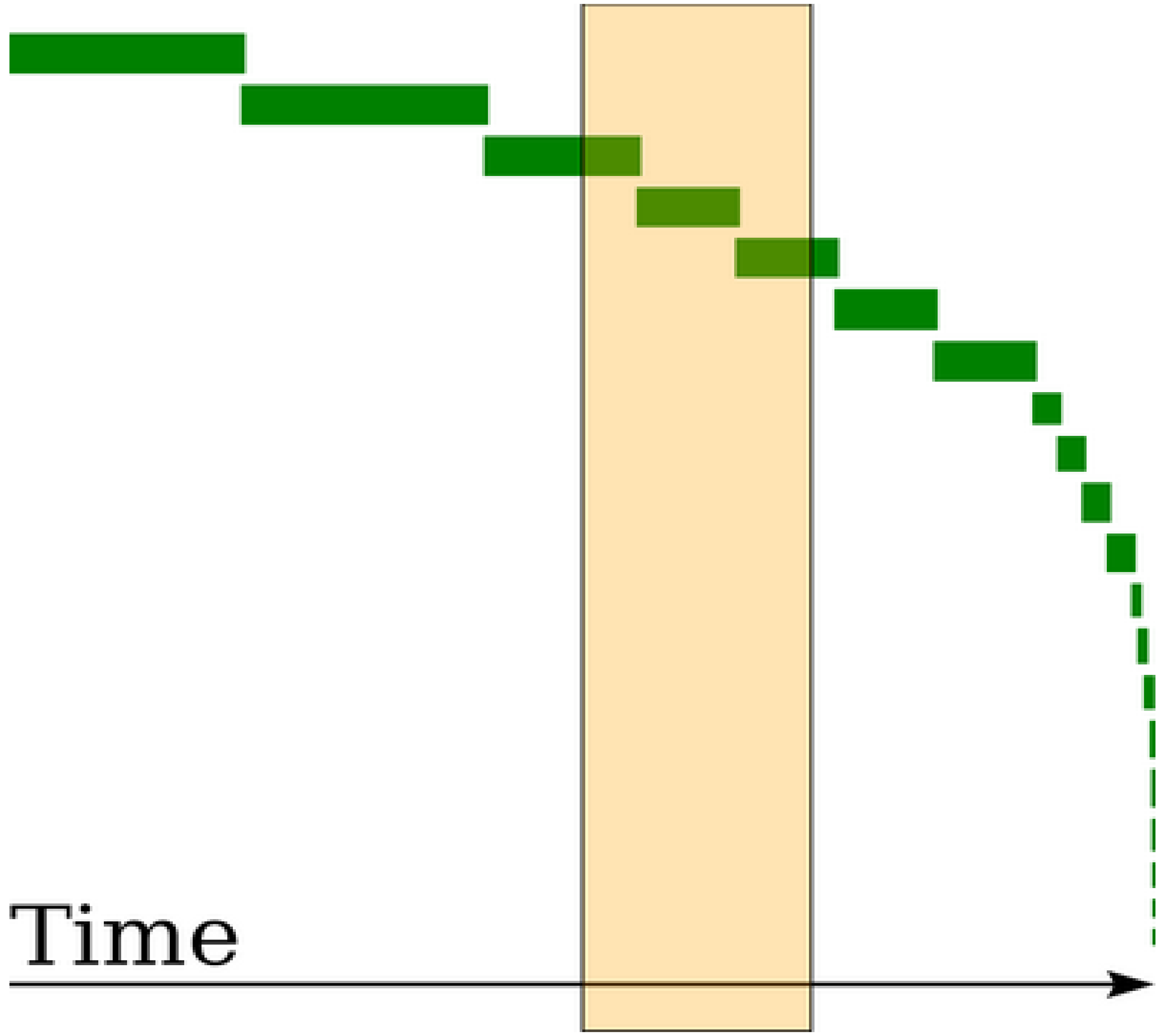


Date-Tiered Compaction Strategy

Date-Tiered Compaction Strategy(HBASE-15181)



Compact multiple time windows in to one tier when time goes by. The older, the bigger tier is.



Logic view

Physical view

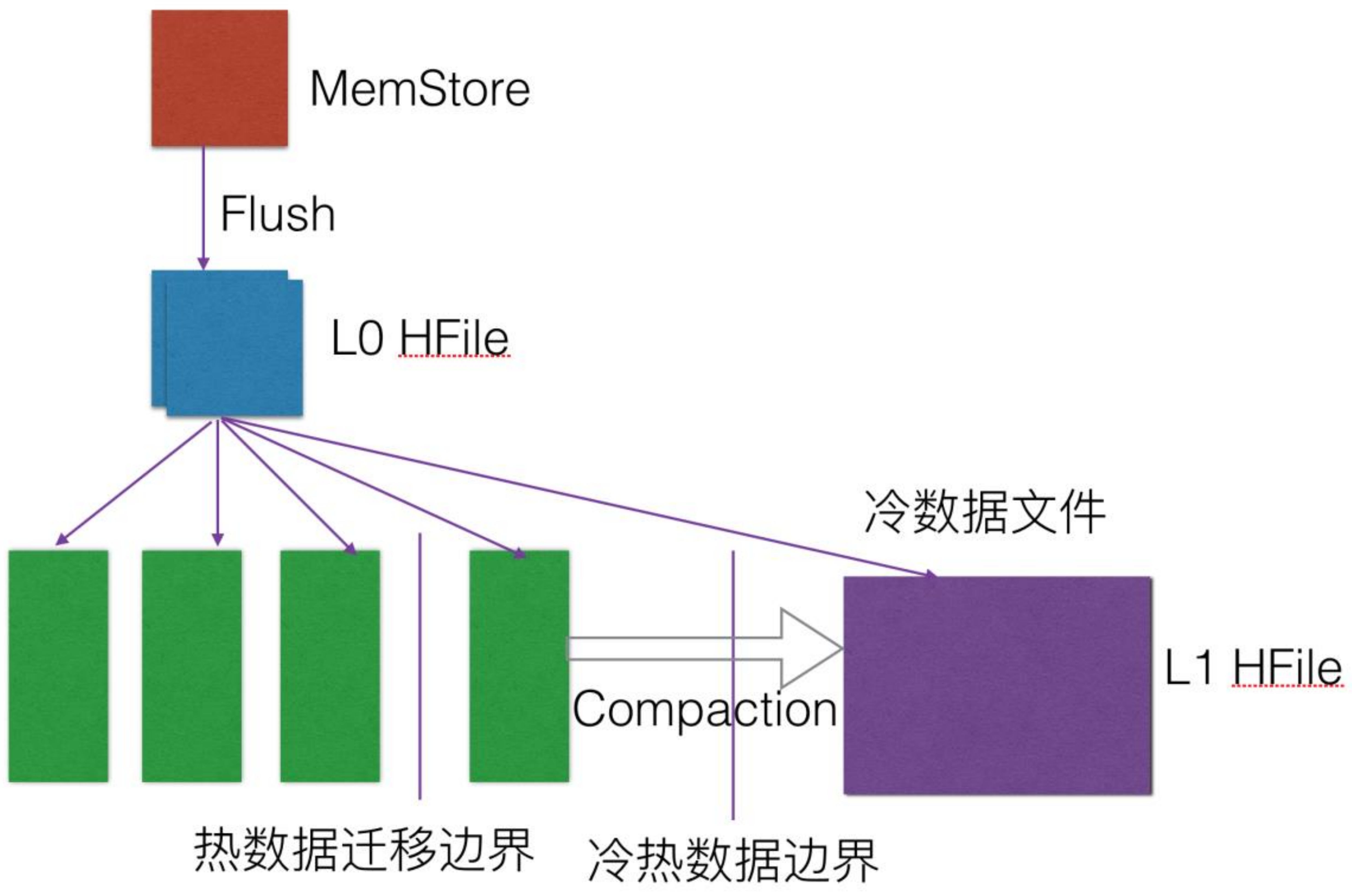
Separating hot cold data

Our layered compaction is inspired by Date-tiered Compaction.

- Only Cold/Warm/Hot window is needed
- Data will move from hot to warn then to Cold window
- Secondary Filed or timestamp is used

Layered Compaction

- HFile flushed by Memstore is always in L0
- Hot/Warm/Cold layer have their own compaction Strategy
- Data is separated by secondary field or timestamp
- Data out of boundary will be compacted out to next layer

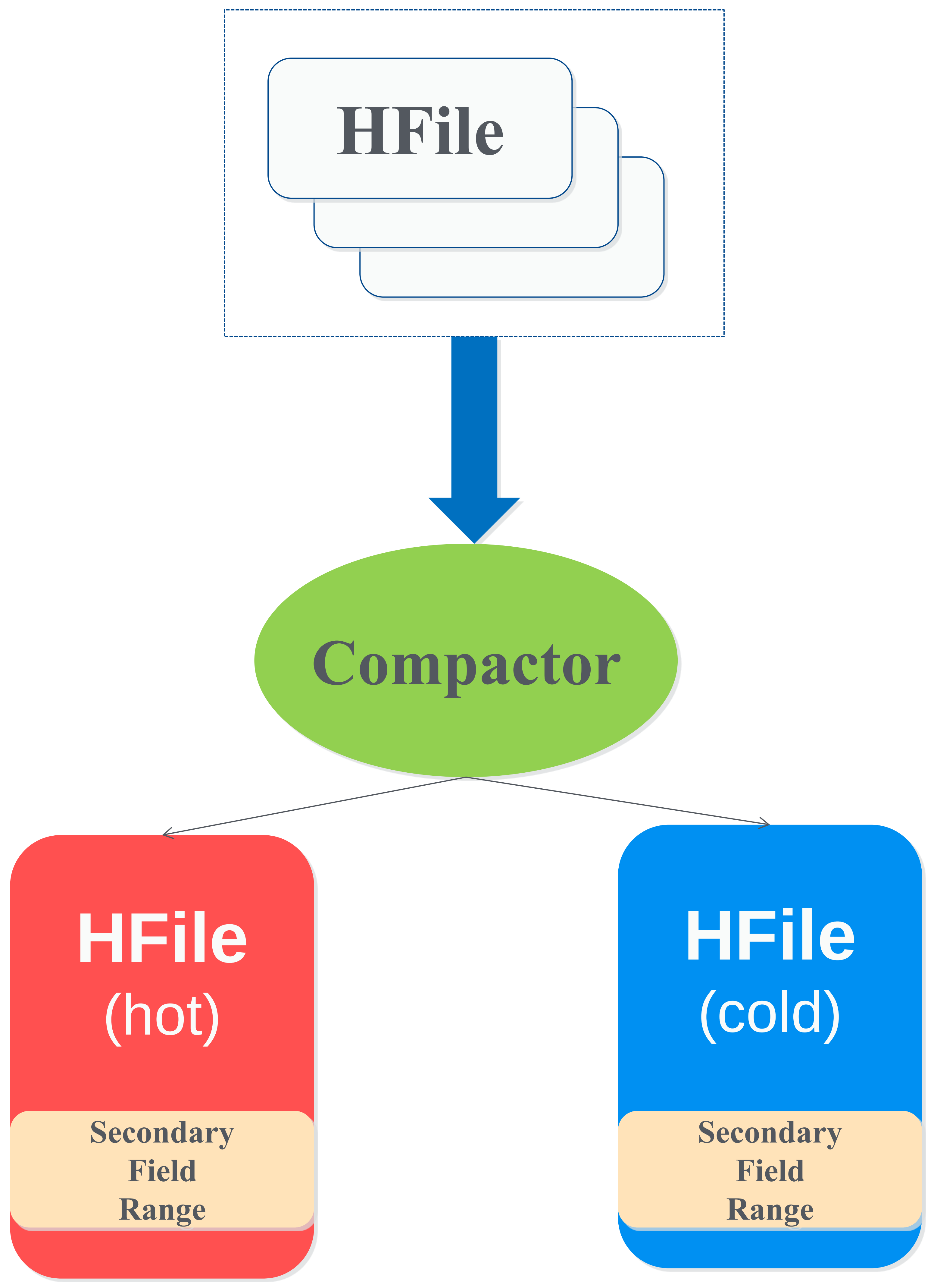


Layered Compaction

- Compactor will output multiple HFiles according to the separation boundary
- Secondary Field range will be written into the FileInfo section of HFile

e.g.

Rowkey:userid+ts
UserA002
UserA005
UserB003
UserB007
Secondary Field Range: 002…007

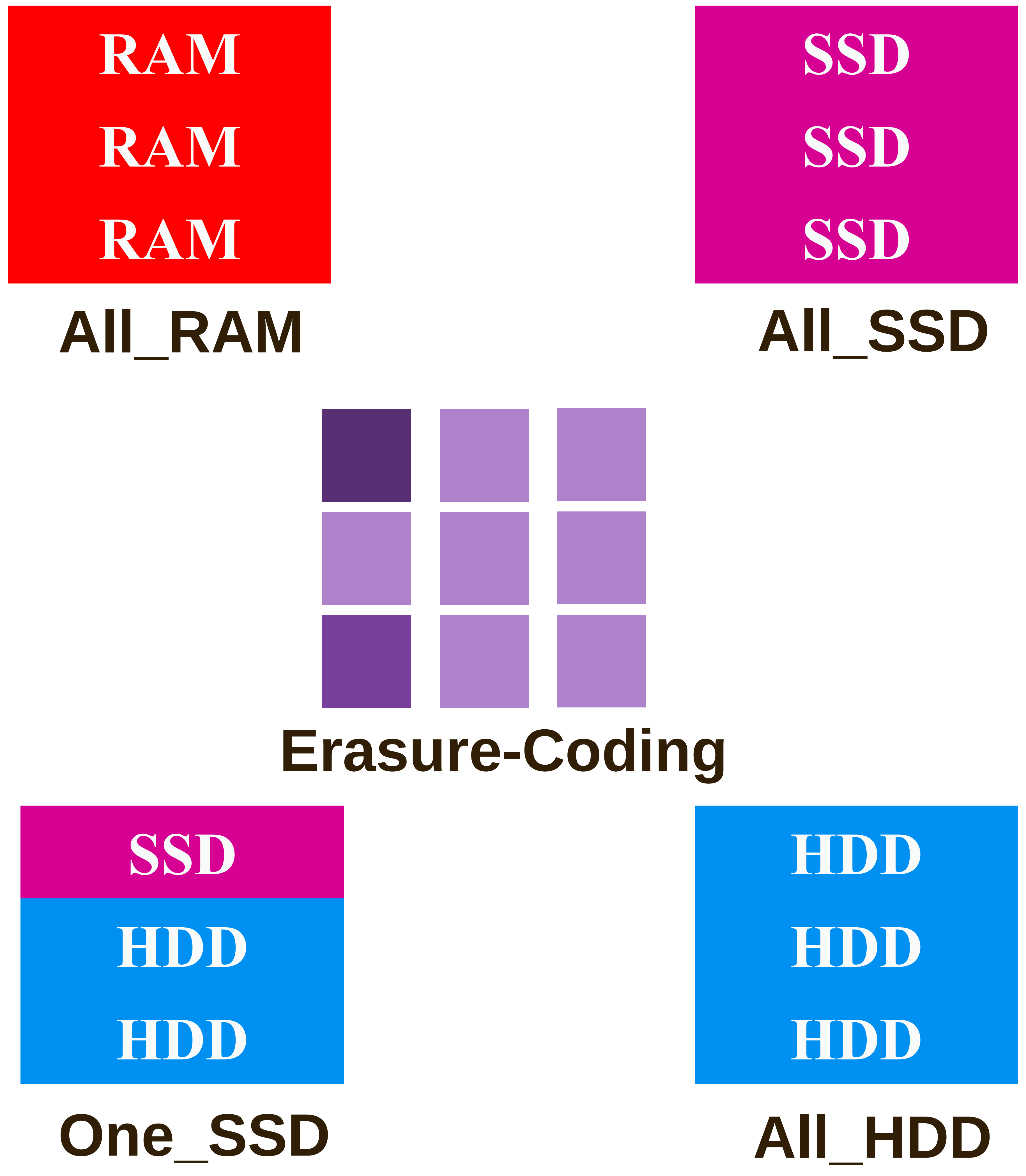


Heterogeneous storage

We can specify Data encoding, Compression, and storage type for each layer

Here is an example:

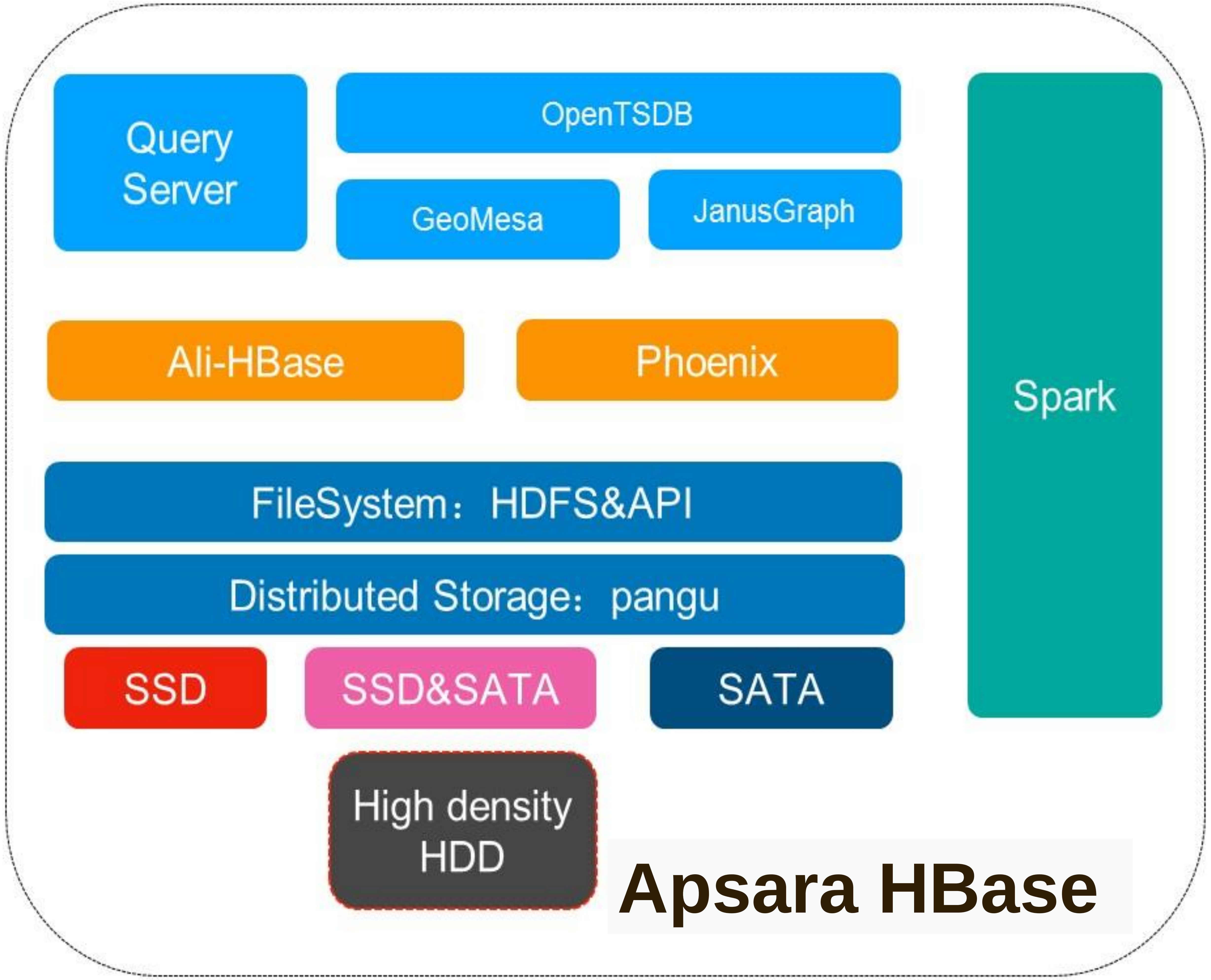
Type	Data Encoding	Compression	Storage
Hot	None	None	SSD/RAM
Warn	DIFF	LZO	One_SSD
Cold	DIFF	LZ4	HDD/EC



Storage Computing Separation

- Apsara HBase Provide a Architecture of storage computing separation

- High density HDD will be available in Apsara HBase about this September.



Welcome to try Apsara HBase at
https://www.aliyun.com/product/hbase

Hot-cold Data Separation

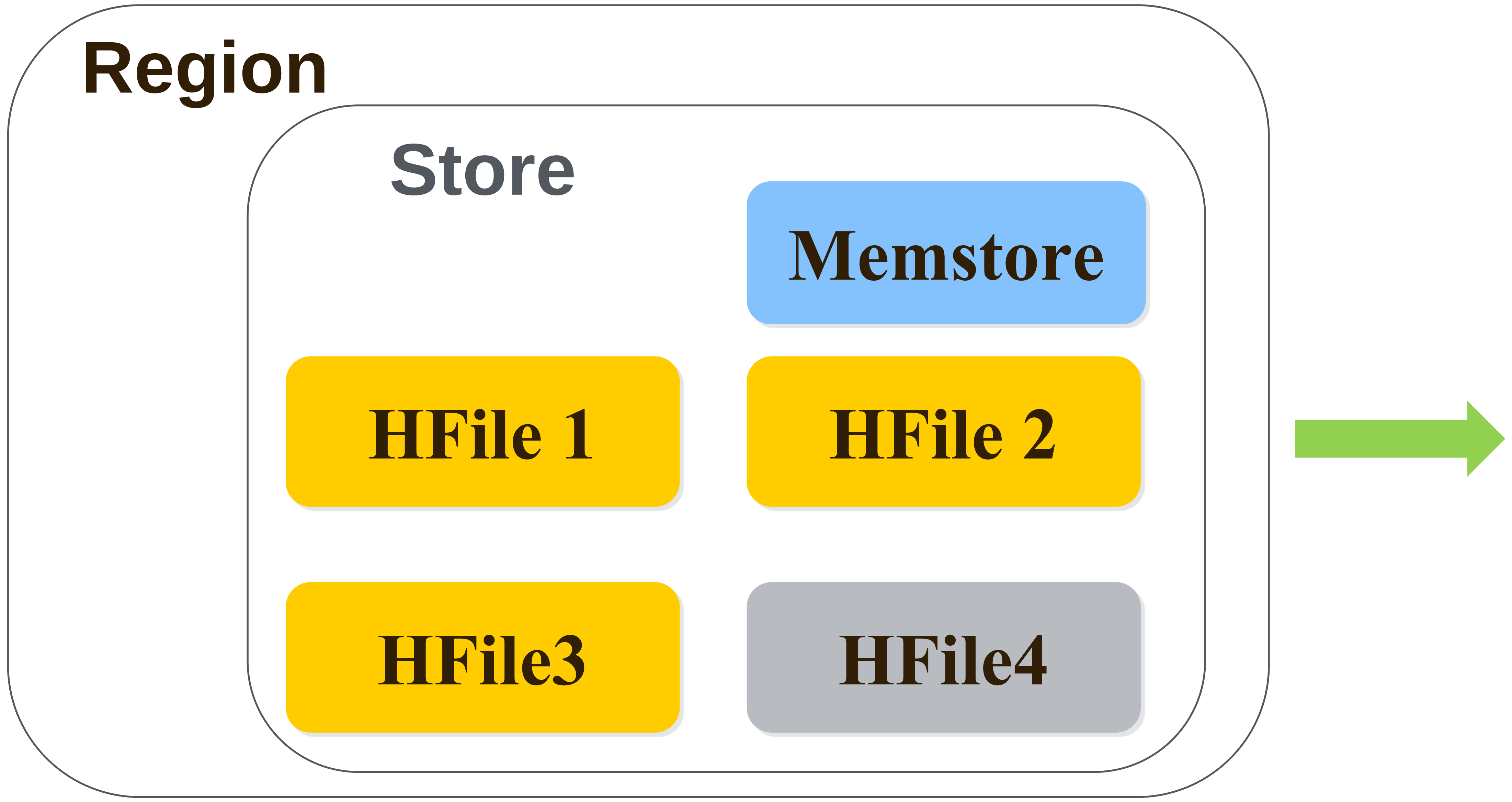
— Hot-cold Data Recognition

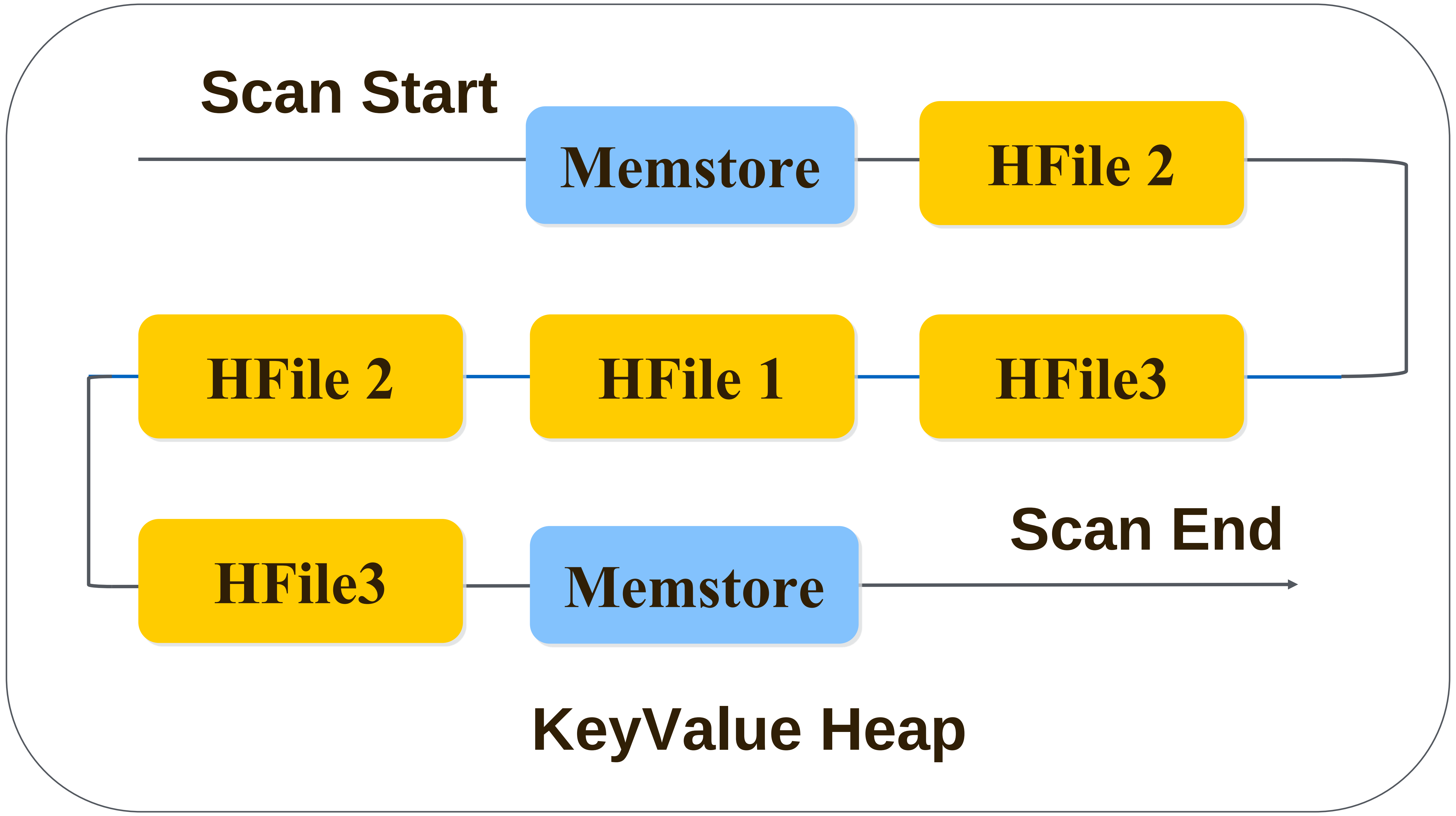
— Layered Compaction

— Query Optimizations

HBase Read Path

A quick tour of HBase read path



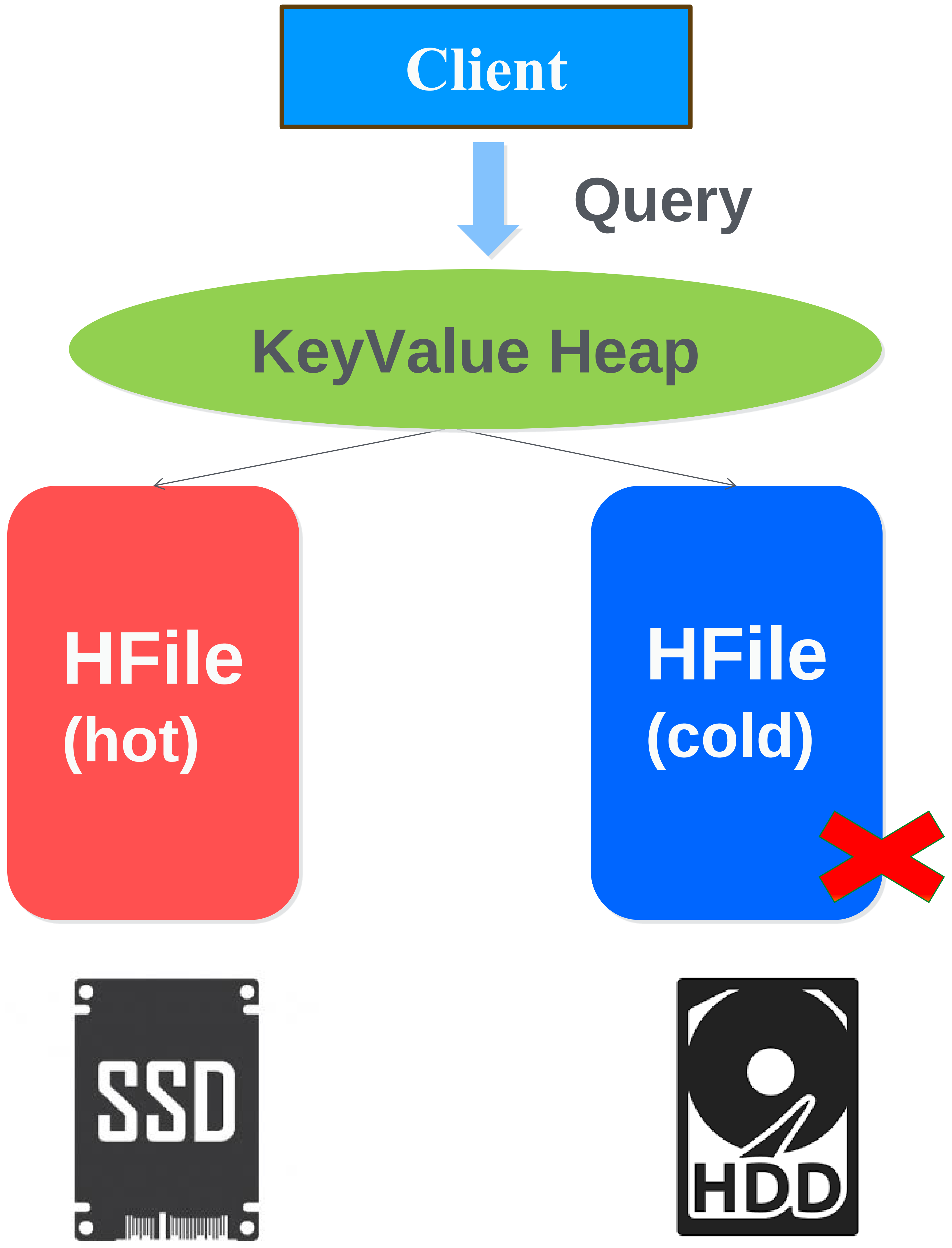


HFile4 is filtered out by:

- Bloom filter
- Time range
- Key range

Goal of Query Optimization

- Query optimization is only for hot queries
- We have to try our best to filter out the cold HFiles, avoid seek in them.
- Seeking in cold HFiles can tremendously increase RT for hot queries



Query Optimization: Case 1

- **Scenario: Monitoring, e.g. OpenTSDB**
- **Rowkey: MetricName + ts + postfix(tags)**

Rowkey	**ts**
cpuA001server1	001
cpuA002server1	002
cpuA003server1	003
cpuA004server1	004
diskB001server1	001
diskB002server1	002
diskB003server1	003
diskB004server1	004

Separate data by boundary: ts = 003

HFile(hot)	**HFile(cold)**
cpuA003server1	cpuA001server1
cpuA004server1	cpuA002server1
disk003server1	diskB001serve1
diskB004server1	diskB002server
Time Range: 003…004	Time Range: 001…002

Optimization:
Scan.setTimeRange(003, 004)

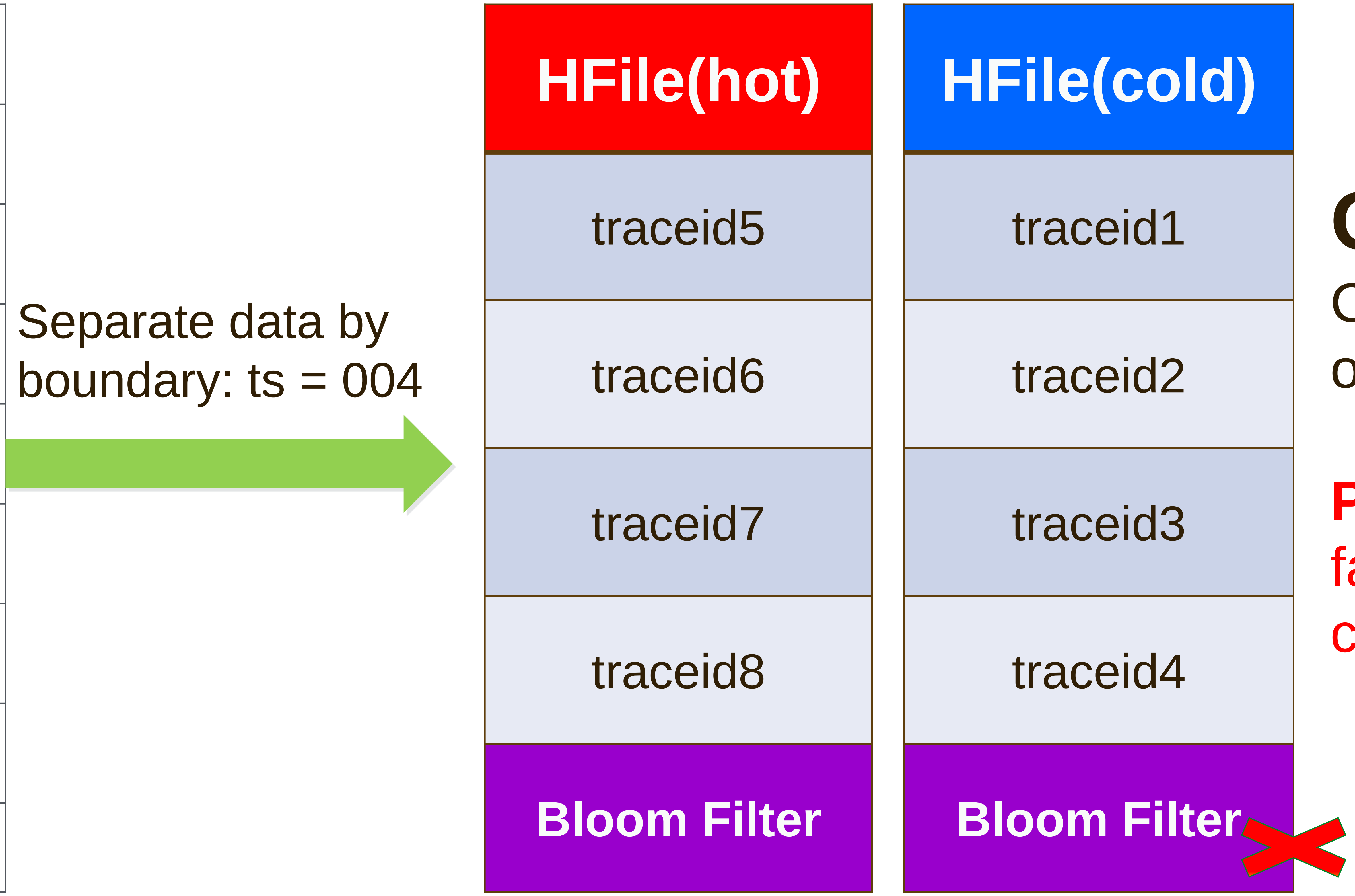
Cold HFile can be filtered out easily by time range

Query: Scan scan = new Scan(cpuA003, cpuA004)

Query Optimization: Case 2

- **Scenario: Tracing system**
- **Rowkey: TraceID (events are recorded in different column)**

Rowkey	**ts**
traceid1	001
traceid2	002
traceid3	003
traceid4	004
traceid5	005
traceid6	006
traceid7	007
traceid8	008

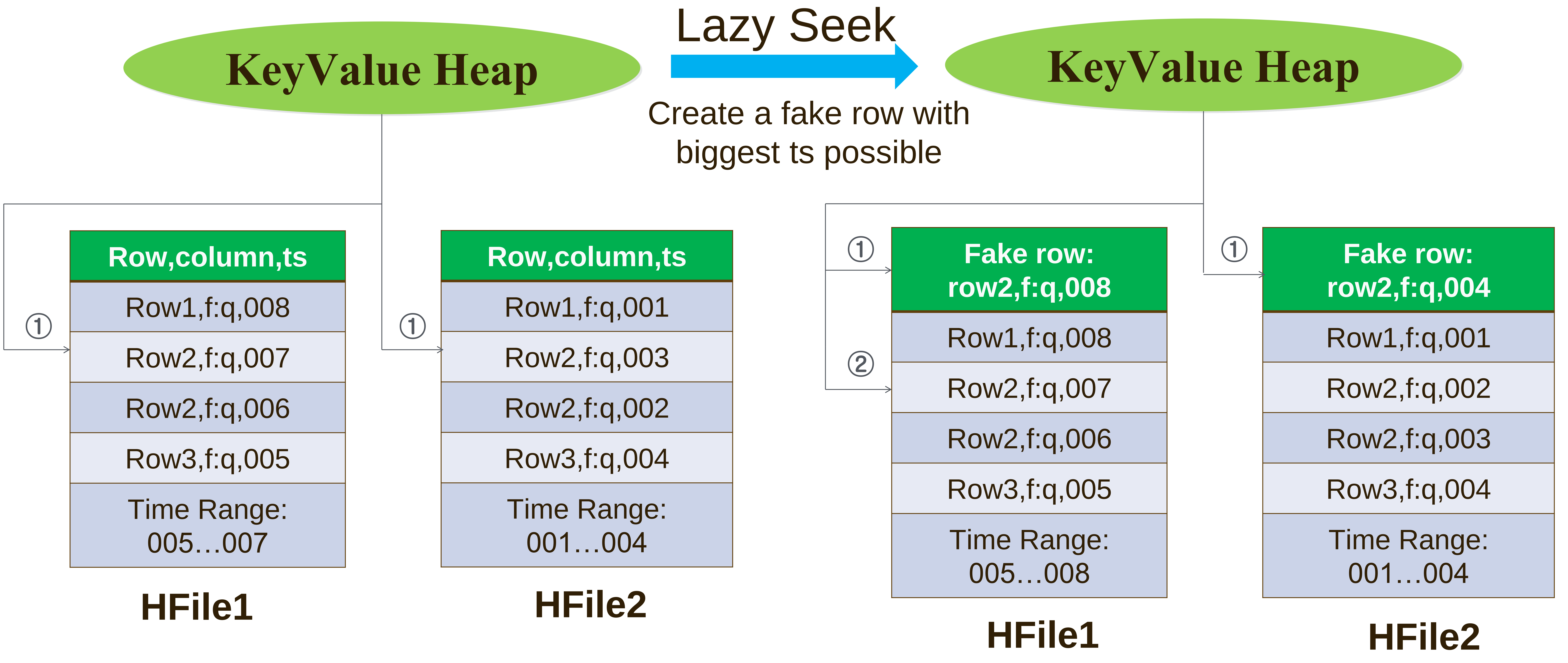


Optimization:
Cold HFile can be filtered out by Bloom Filter

Problem:
false positive of bloom filter can cause spikes

Query: Get get= new Get("traceid7")

Lazy Seek (HBASE-4465)



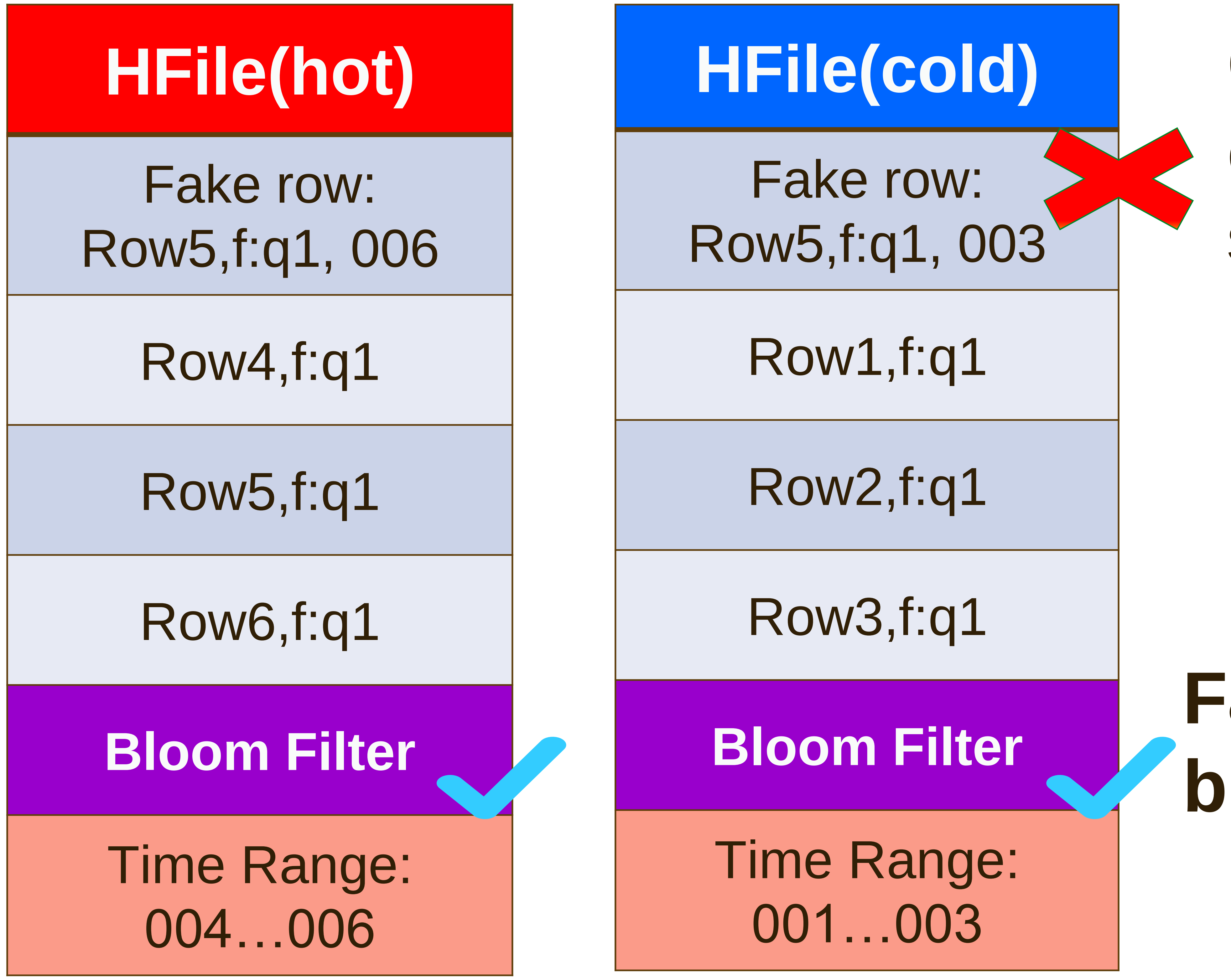
Query: Select row >= Row2,f:q and limit =1

Query Optimization: Case 3

- **Scenario: KV Store**
- **Rowkey: key（with only one qualifier）**

Row,Column	ts
Row1,f:q1	001
Row2,f:q1	002
Row3,f:q1	003
Row4,f:q1	004
Row5,f:q1	005
Row6,f:q1	006

Separate data by boundary: ts = 004



Optimization:
Cold HFile will not be seeked because of lazy seek

False positive of bloom filter

Query: Get get= new Get(“Row5,f:q1”)

Query Optimization: Case 4

- **Scenario: Logistics tracking in Alibaba**
- **Rowkey: traceNo + actionCode + ts**

Rowkey	ts
trace1Collect001	001
trace1Arrive002	002
trace1Delivery003	003
trace1Done004	004
trace2Collect005	005
trace2Arrive006	006
trace2Delivery007	007
trace2Done008	008

Separate data by boundary: ts = 004

HFile(hot)
trace2Collect005
trace2Arrive006
trace2Delivery007
trace2Done008
Time Range: 005…008

HFile(cold)
trace1Collect001
trace1Arrive002
trace1Delivery003
trace1Done004
Time Range: 001…004

Problem:

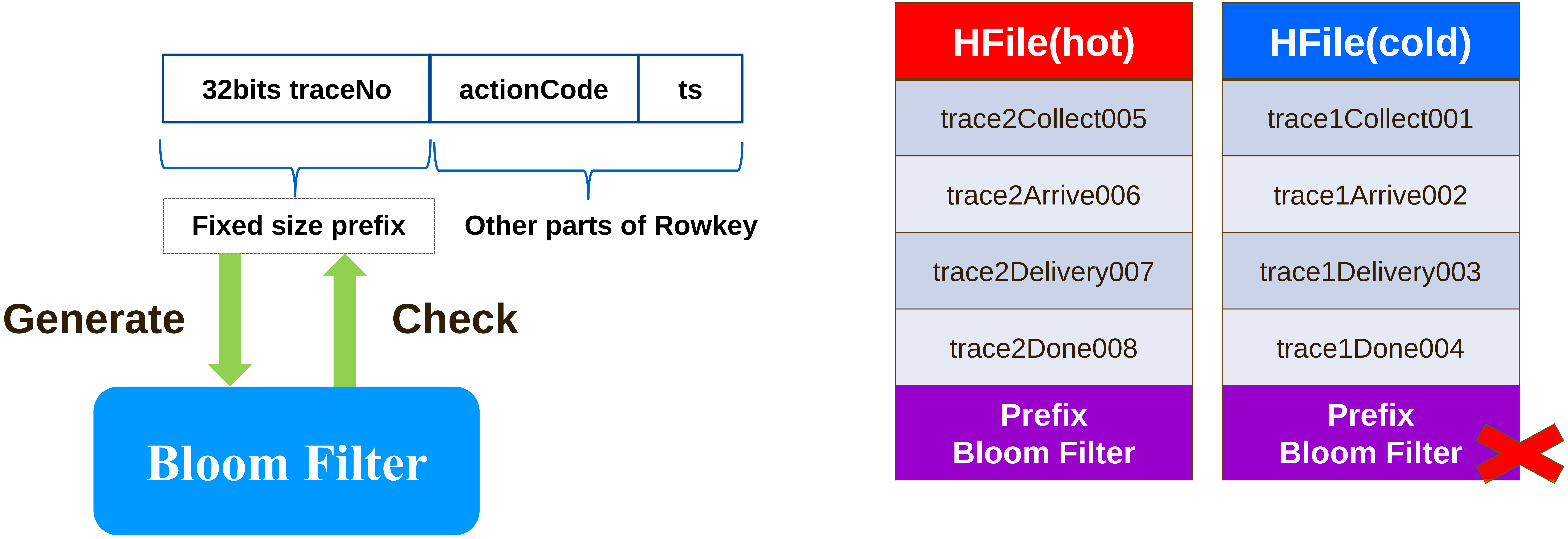
Scan with prefix, no time range can be provided

~~Key Range~~
~~Time Range~~
~~Bloom Filter~~
~~Lazy Seek~~

Query: Scan scan = new Scan("trace2" , "trace2~")

Prefix Bloom Filter

- Use the prefix part of a rowkey to generate and to check bloom filter



Query: Scan scan = new Scan("trace2" , "trace2~")

Query Optimization: Case 5

- **Scenario: Bills History in Alibaba**
- **Rowkey: userID + reverse(ts) + (oderID)**

Rowkey	**ts**
userA991	008
userA992	007
userA993	006
userA994	005
userA995	004
userA996	003
userA997	002
userA998	001

Separate data by boundary: ts = 004

HFile(hot)
userA991
userA992
userA993
userA994
Time Range: 005…008

HFile(cold)
userA995
userA996
userA997
userA998
Time Range: 001…004

Problem:

Scan with prefix, no endkey, no time range can be provided

~~Key Range~~
~~Time Range~~
~~Bloom Filter~~
~~Prefix Bloom Filter~~
~~Lazy Seek~~

Query: Scan scan = new Scan("userA"), Limit 4

Secondary Field Lazy Seek

- Store Secondary Filed Range in HFile’s FileInfo section
- Create fake key to perform lazy seek

Query: Scan scan = new Scan(“userA”), Limit 4

Query Optimization: Case 1 - revisit

- **Scenario: Monitoring, e.g. OpenTSDB**
- **Rowkey: MetricName + ts(Secondary Field) + postfix(tags)**

Rowkey	ts
cpuA001server1	001
cpuA002server1	002
cpuA003server1	003
cpuA004server1	004
cpuB001server1	001
cpuB002server1	002
cpuB003server1	003
cpuB004server1	004

Separate data by boundary: ts = 003

HFile(hot)
cpuA003server1
cpuA004server1
cpuB003server1
cpuB004server1
Secondary Field Range: 003…004

HFile(cold)
cpuA001server1
cpuA002server1
cpuB001server1
cpuB002server1
Secondary Field Range: 001…002

Optimization:

~~Scan.setTimeRange(003, 004)~~

Cold HFile can be filtered out easily by Secondary Field

Query: Scan scan = new Scan(cpuA003, cpuA004)

03 Conclusion

Conclusion

- A new approach to separate hot-cold data was introduced
- A new Secondary Field Slicer was used to decide layer boundaries besides timestamp
- Layered compaction was used to separate data to different layer
- Heterogeneous storage was used to balance cost and performance
- New technology like Prefix Bloom Filter and Secondary Field Range Lazy Seek was used to do auto query optimization
- Production test shows that our approach can lower the query RT by 50% and decrease the storage usage by 25%

We are hiring!

- If you are interested in or familiar with Hadoop ecosystem or any other No-SQL database

- If you are eager to accept challenge of building high concurrency, low latency and flexible system

Allan

浙江 杭州

扫一扫上面的二维码图案，加我微信

Thanks

FAQ