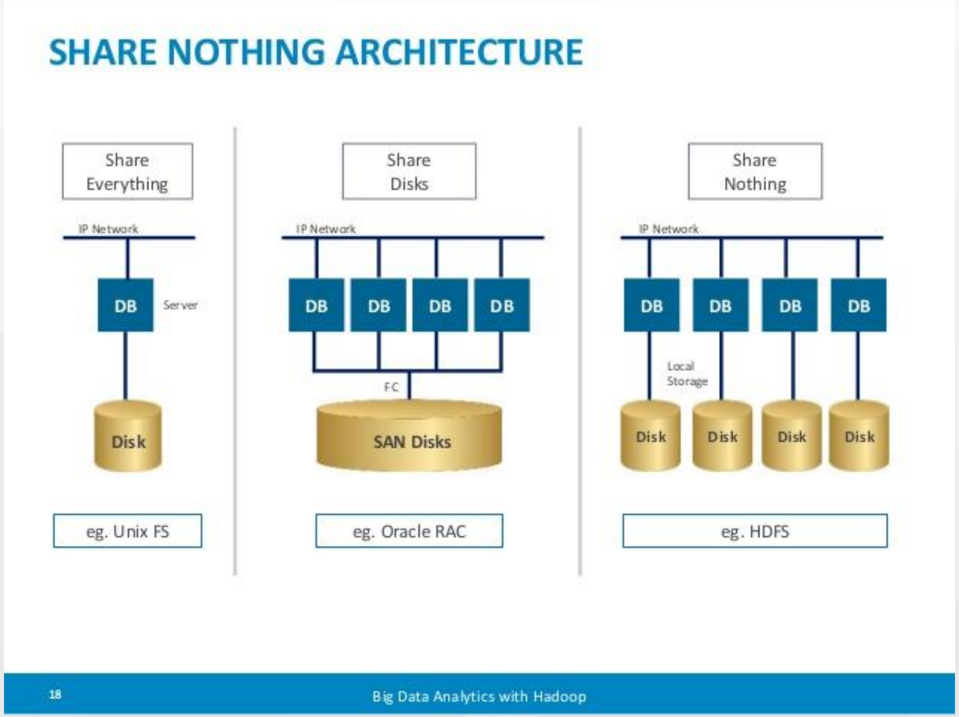


SHARE NOTHING ARCHITECTURE

18 Big Data Analytics with Hadoop

Hbase Cluster

2017.4 XenRon

CONTENTS

01 Row Key

02 Schema Design

03 API

04 Cluster & HA

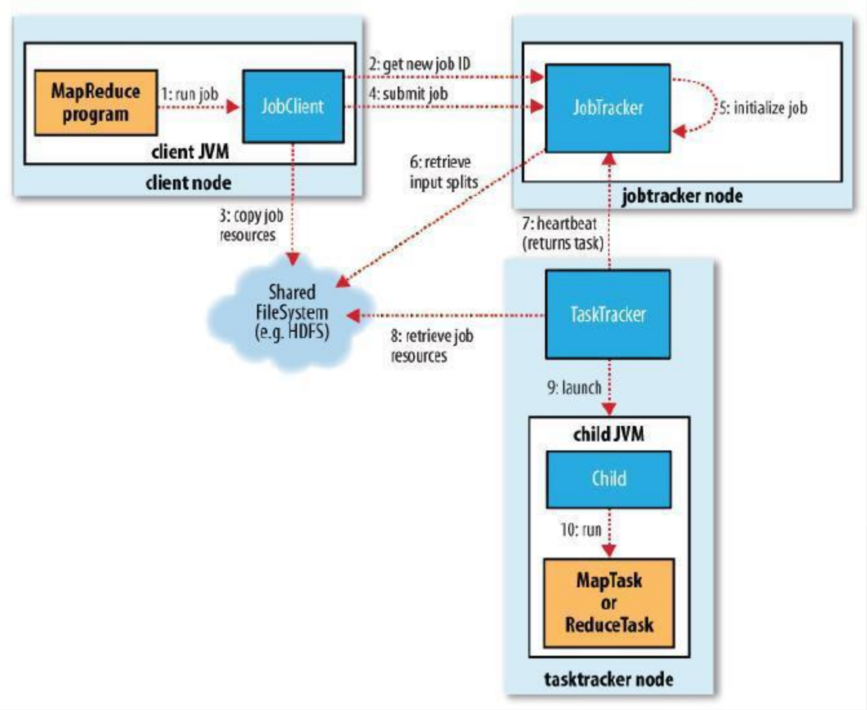
05 Backup & Restore

Review

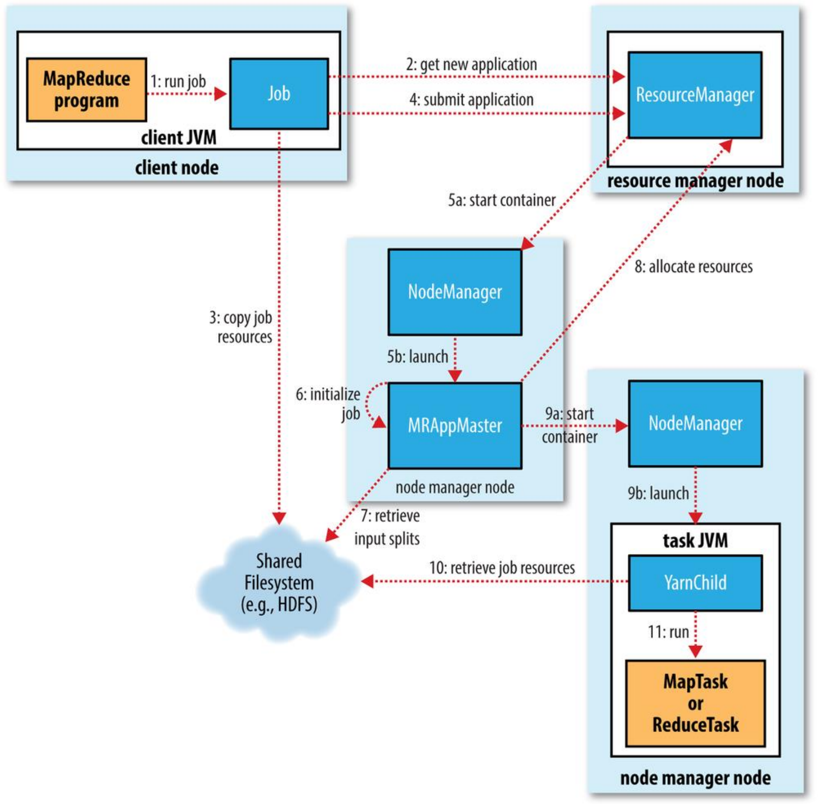
Review

Review

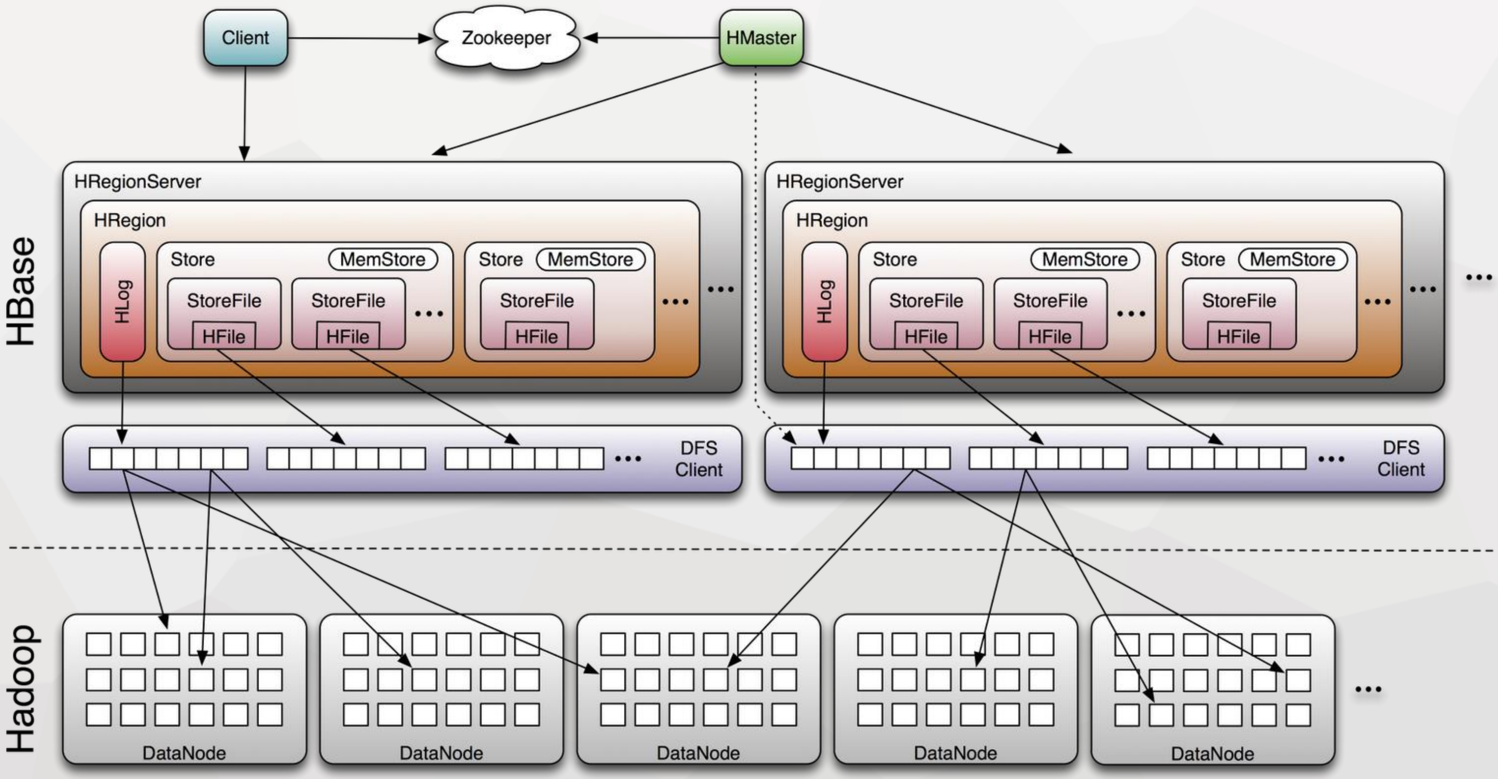
Hadoop 1.x



Hadoop 2.x



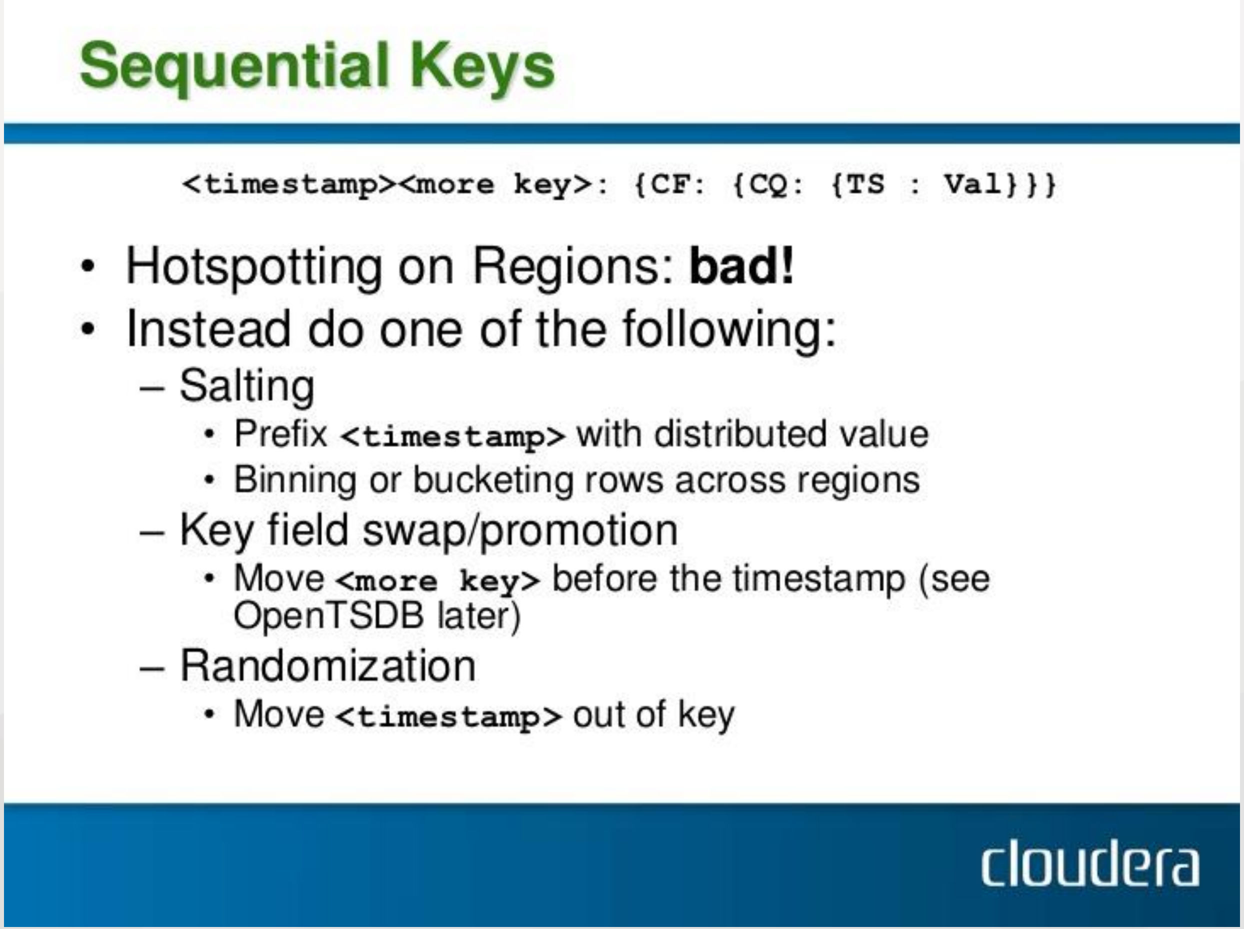
architecture



PART1

Row Key

Row Key Type



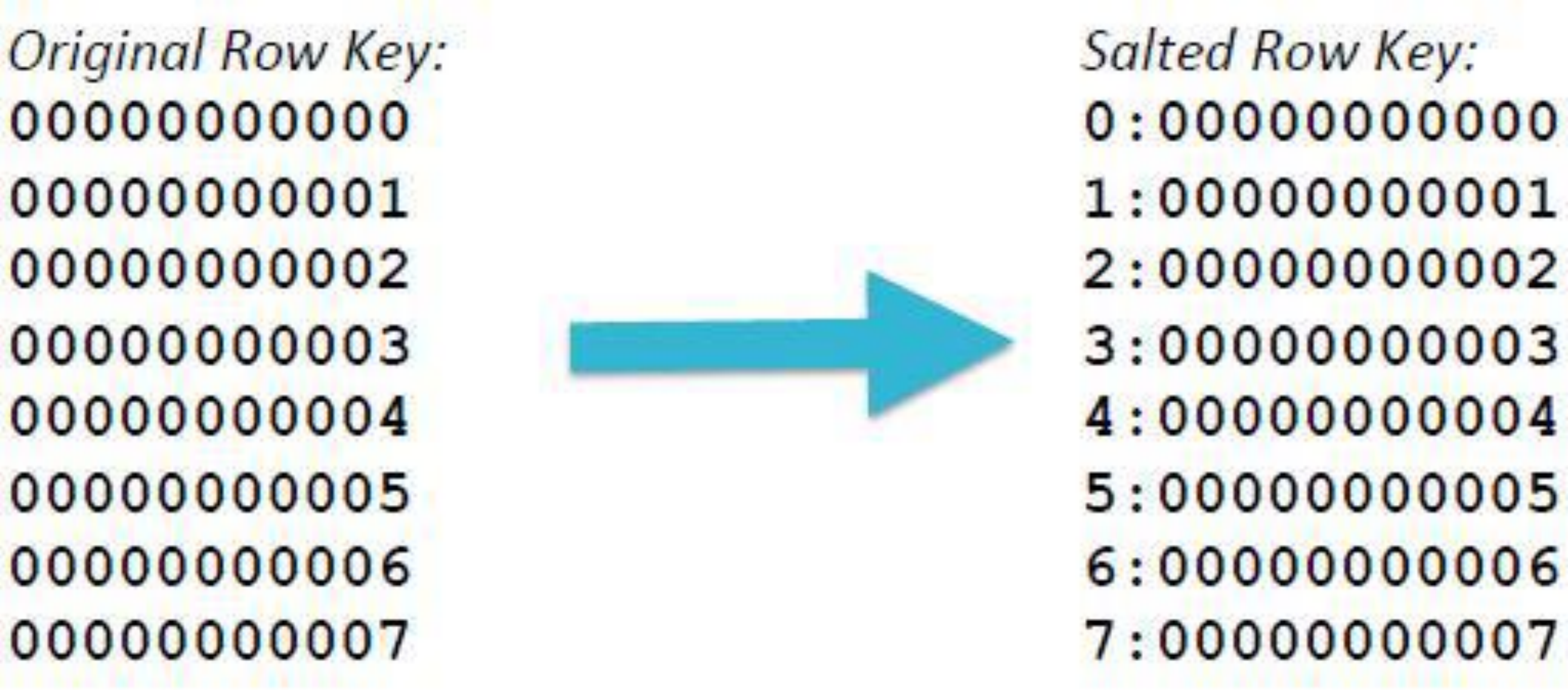
Sequential Keys

```
<timestamp><more key>: {CF: {CQ: {TS : Val}}}
```

- Hotspotting on Regions: **bad!**
- Instead do one of the following:
  - Salting
    - Prefix `<timestamp>` with distributed value
    - Binning or bucketing rows across regions
  - Key field swap/promotion
    - Move `<more key>` before the timestamp (see OpenTSDB later)
  - Randomization
    - Move `<timestamp>` out of key

cloudera

Row Key Type



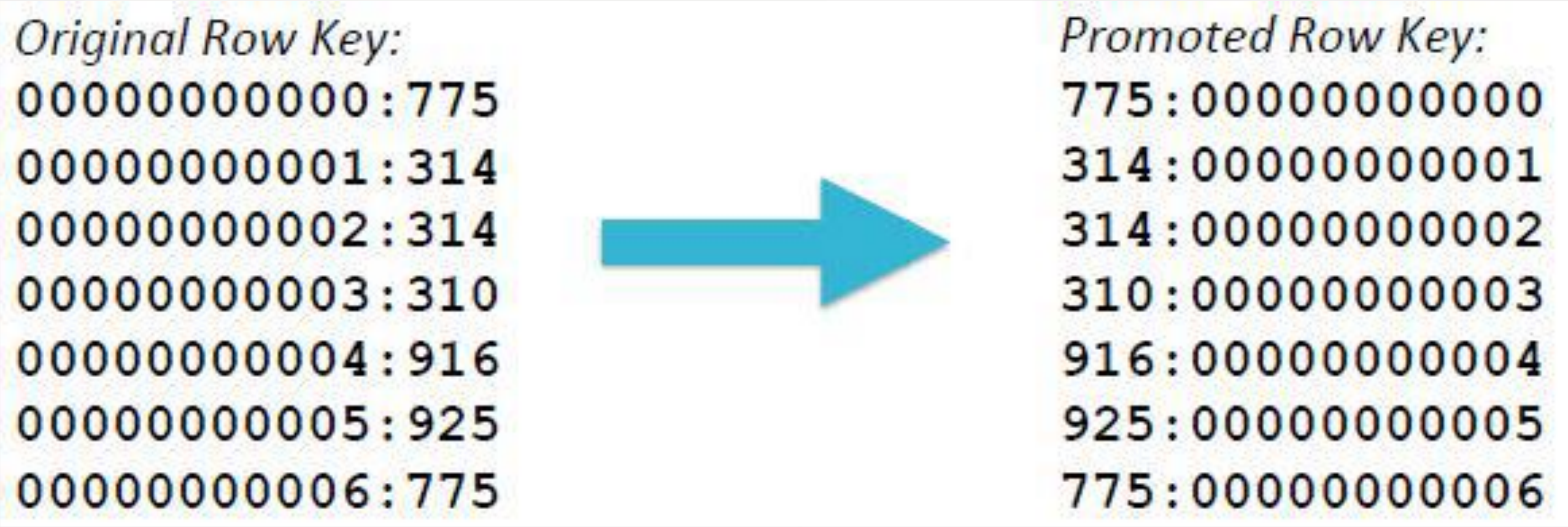
Original Row Key:

```
0000000000
0000000001
0000000002
0000000003
0000000004
0000000005
0000000006
0000000007
```

Salted Row Key:

```
0:0000000000
1:0000000001
2:0000000002
3:0000000003
4:0000000004
5:0000000005
6:0000000006
7:0000000007
```

Row Key Type



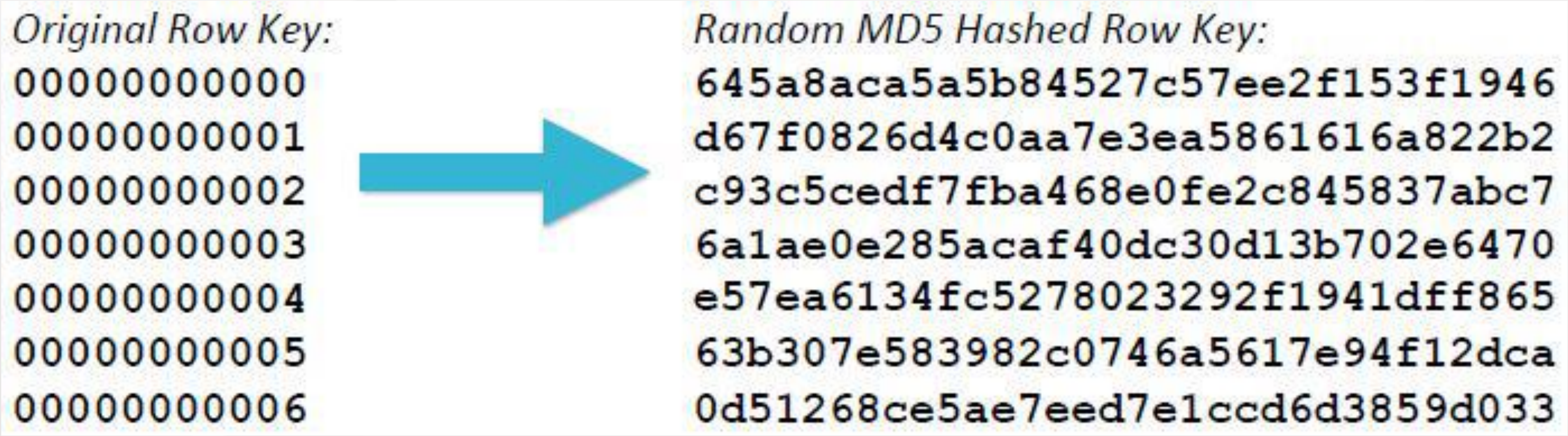
Original Row Key:

```
0000000000:775
0000000001:314
0000000002:314
0000000003:310
0000000004:916
0000000005:925
0000000006:775
```

Promoted Row Key:

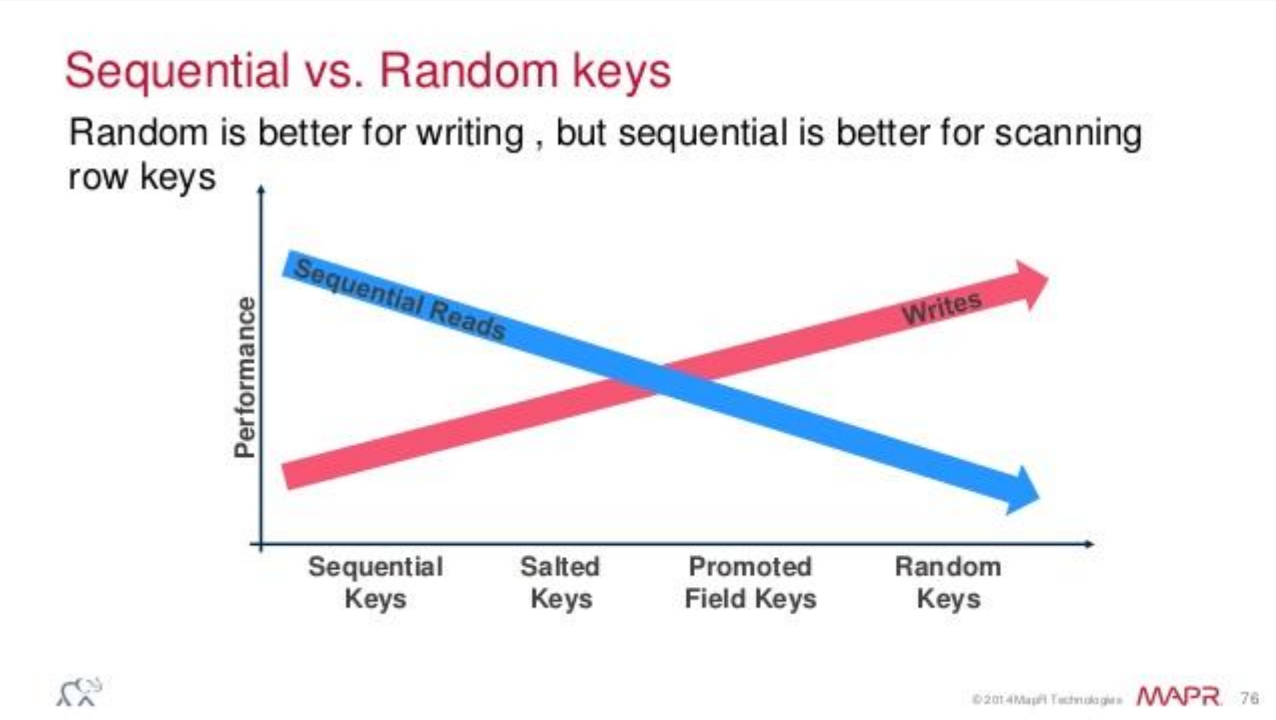
```
775:0000000000
314:0000000001
314:0000000002
310:0000000003
916:0000000004
925:0000000005
775:0000000006
```

Row Key Type



Original Row Key:	*Random MD5 Hashed Row Key:*
00000000000	645a8aca5a5b84527c57ee2f153f1946
00000000001	d67f0826d4c0aa7e3ea5861616a822b2
00000000002	c93c5cedf7fba468e0fe2c845837abc7
00000000003	6a1ae0e285acaf40dc30d13b702e6470
00000000004	e57ea6134fc5278023292f1941dff865
00000000005	63b307e583982c0746a5617e94f12dca
00000000006	0d51268ce5ae7eed7e1ccd6d3859d033

architecture



Sequential vs. Random keys

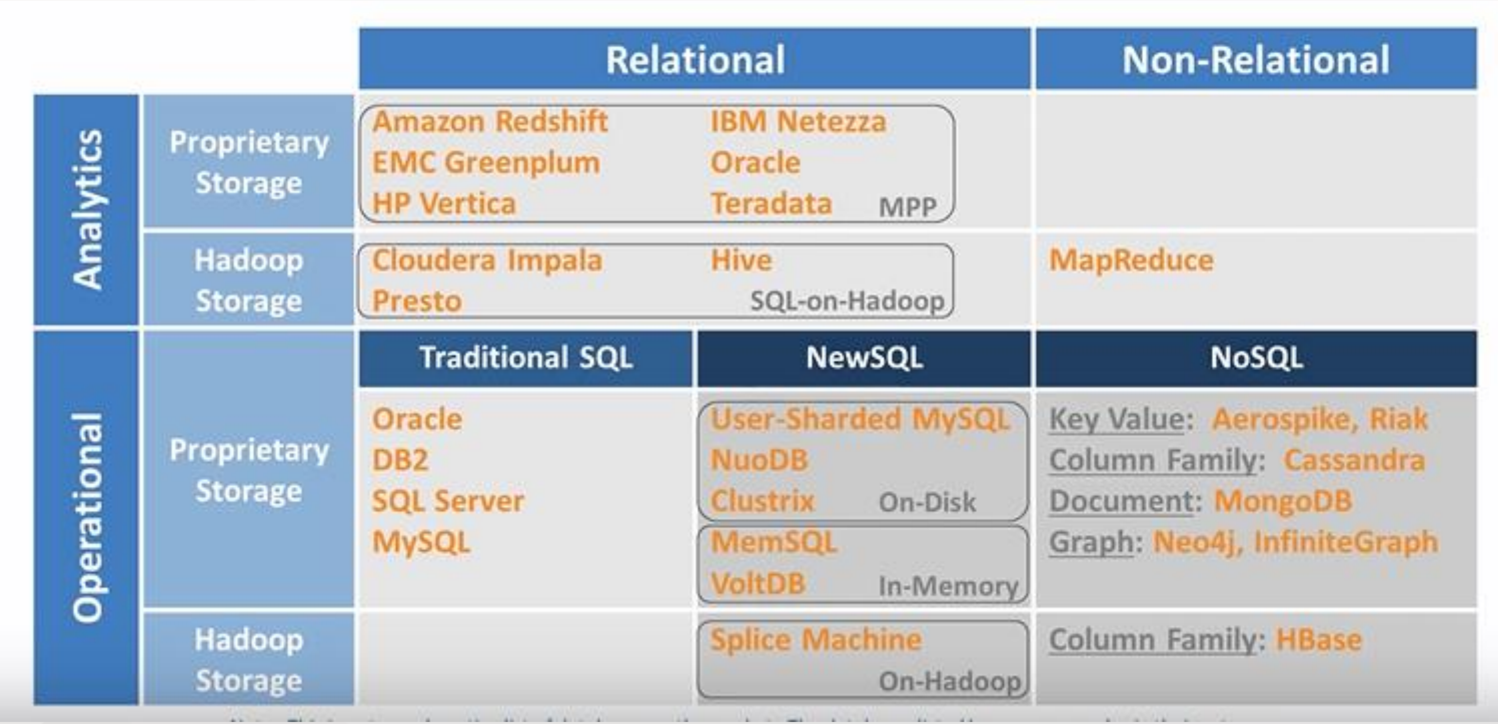
Random is better for writing , but sequential is better for scanning row keys

© 2014 MapR Technologies MAPR 76

PART2

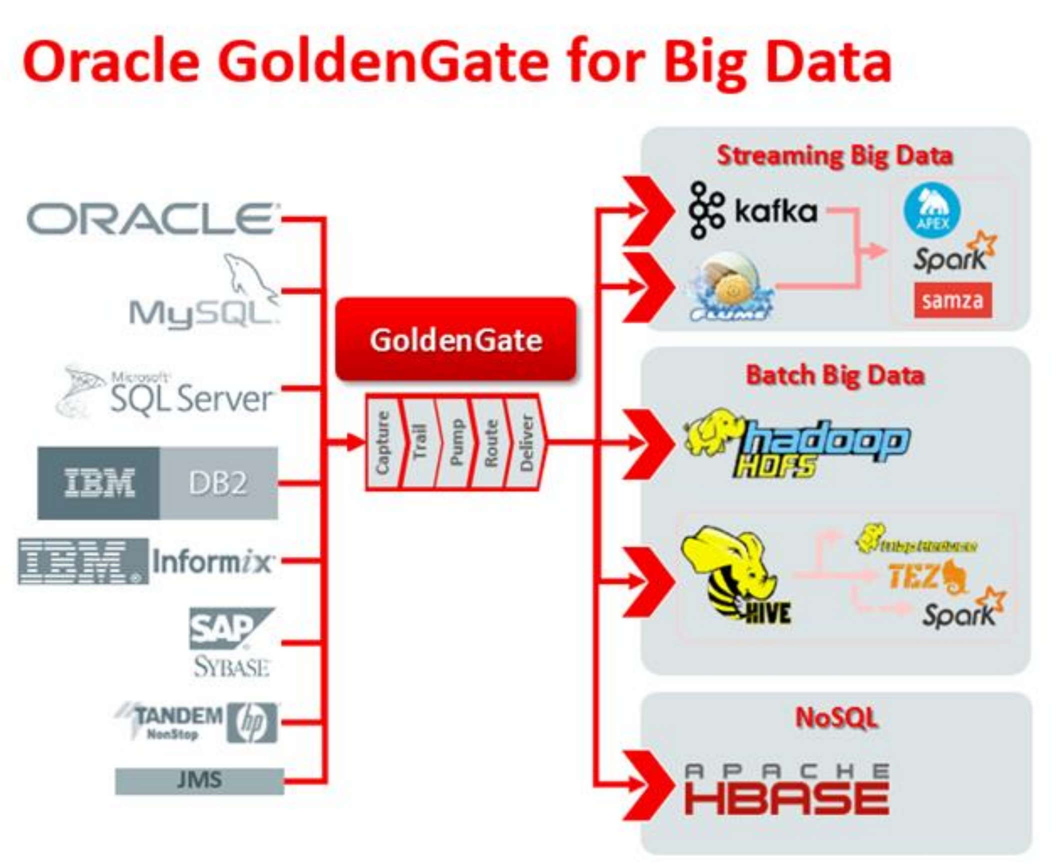
Schema Design

RDBMS VS NoSQL

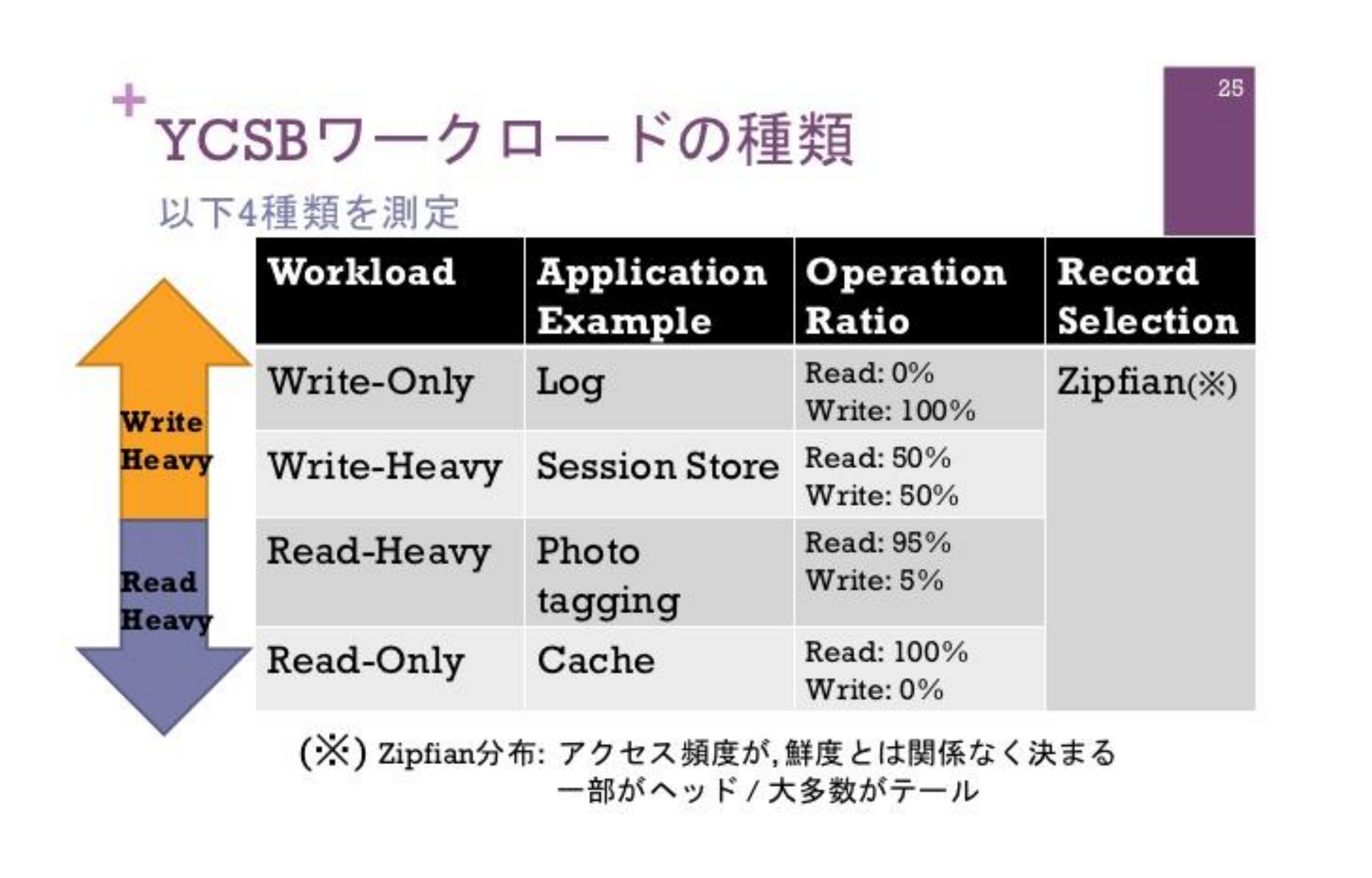


		Relational		Non-Relational
Analytics	Proprietary Storage	Amazon Redshift EMC Greenplum HP Vertica	IBM Netezza Oracle Teradata MPP	
	Hadoop Storage	Cloudera Impala Presto	Hive SQL-on-Hadoop	MapReduce
		Traditional SQL	NewSQL	NoSQL
Operational	Proprietary Storage	Oracle DB2 SQL Server MySQL	User-Sharded MySQL NuoDB Clustrix On-Disk MemSQL VoltDB In-Memory	Key Value: Aerospike, Riak Column Family: Cassandra Document: MongoDB Graph: Neo4j, InfiniteGraph
	Hadoop Storage		Splice Machine On-Hadoop	Column Family: HBase

Disparate Data Sources



Write Heavy VS Read Heavy



25

YCSBワークロードの種類

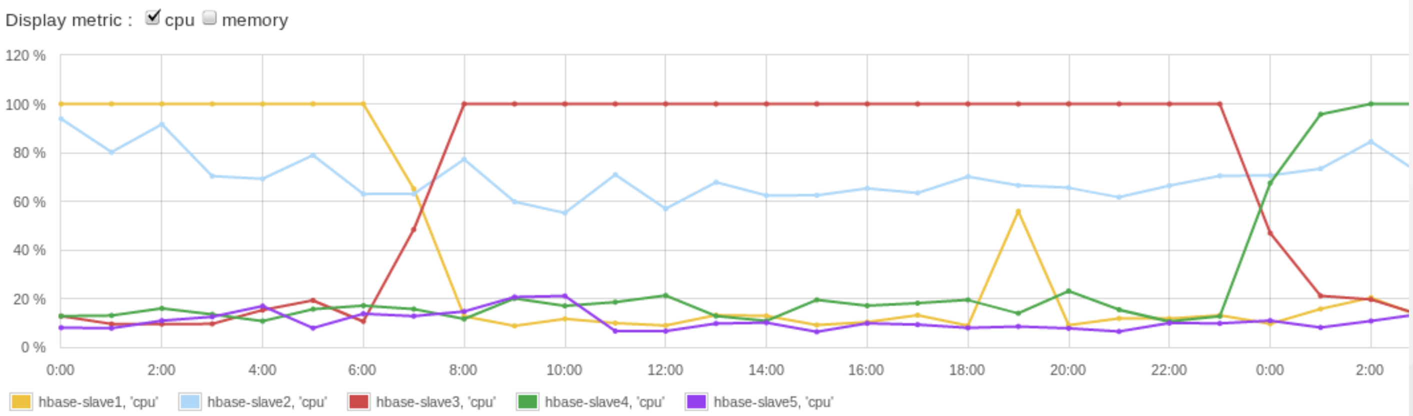
以下4種類を測定

Write Heavy
Read Heavy

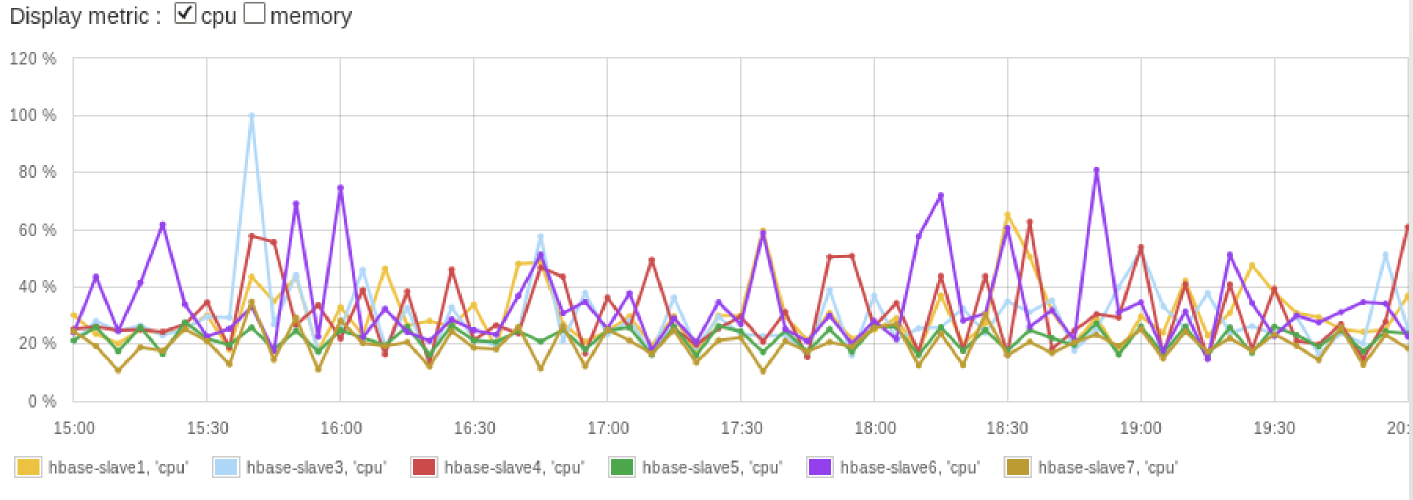
Workload	Application Example	Operation Ratio	Record Selection
Write-Only	Log	Read: 0% Write: 100%	Zipfian(※)
Write-Heavy	Session Store	Read: 50% Write: 50%	
Read-Heavy	Photo tagging	Read: 95% Write: 5%	
Read-Only	Cache	Read: 100% Write: 0%	

(※) Zipfian分布: アクセス頻度が, 鮮度とは関係なく決まる
一部がヘッド / 大多数がテール

Hotspoting



Display metric : ☑ cpu □ memory

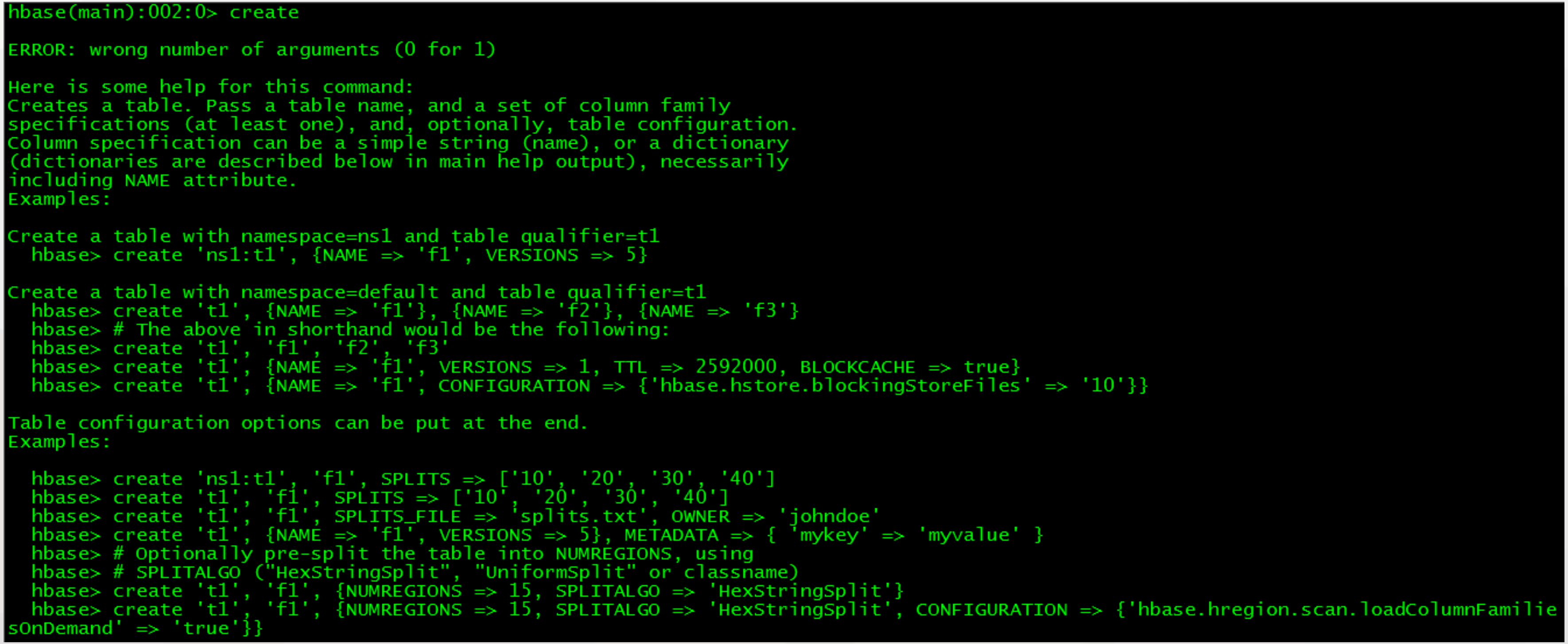


Display metric : ☑ cpu □ memory

PART3

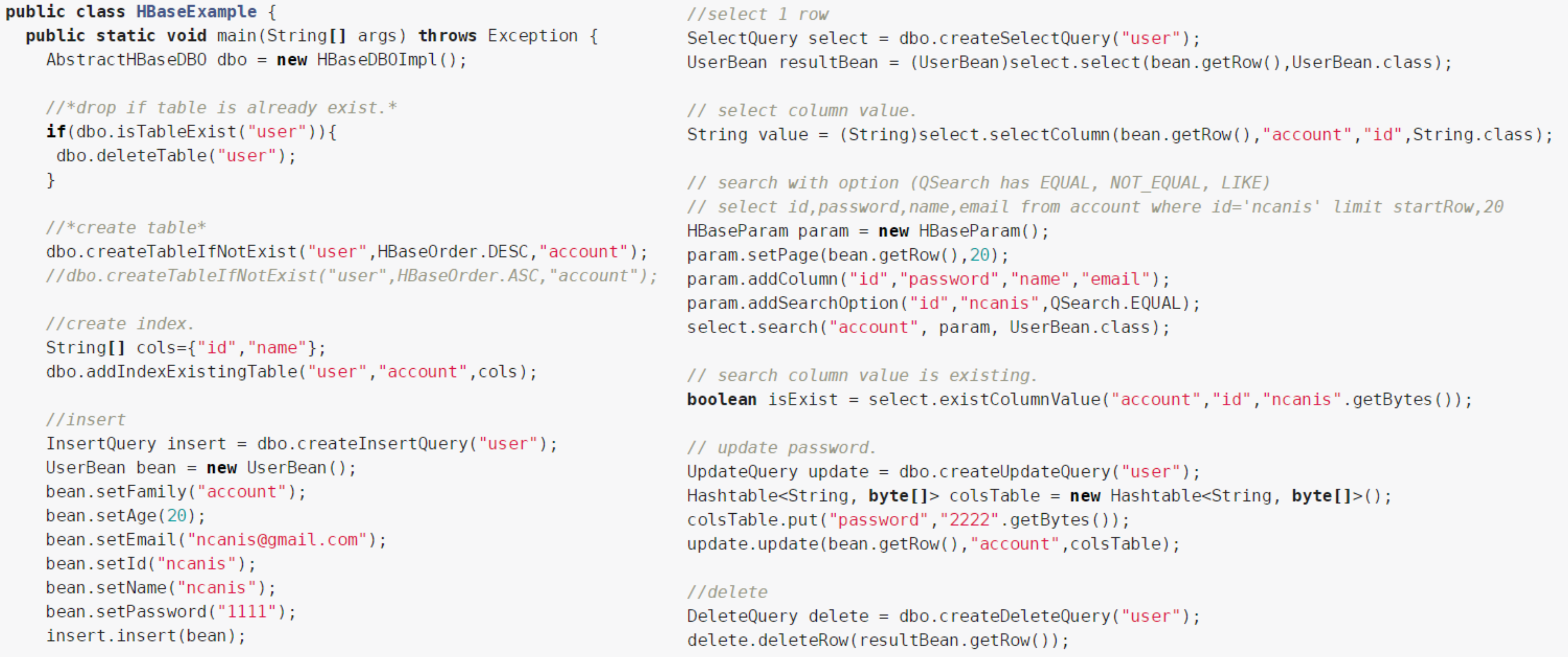
API

Hbase Shell



```
hbase(main):002:0> create

ERROR: wrong number of arguments (0 for 1)

Here is some help for this command:
Creates a table. Pass a table name, and a set of column family
specifications (at least one), and, optionally, table configuration.
Column specification can be a simple string (name), or a dictionary
(dictionaries are described below in main help output), necessarily
including NAME attribute.
Examples:

Create a table with namespace=ns1 and table qualifier=t1
  hbase> create 'ns1:t1', {NAME => 'f1', VERSIONS => 5}

Create a table with namespace=default and table qualifier=t1
  hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}
  hbase> # The above in shorthand would be the following:
  hbase> create 't1', 'f1', 'f2', 'f3'
  hbase> create 't1', {NAME => 'f1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}
  hbase> create 't1', {NAME => 'f1', CONFIGURATION => {'hbase.hstore.blockingStoreFiles' => '10'}}

Table configuration options can be put at the end.
Examples:

  hbase> create 'ns1:t1', 'f1', SPLITS => ['10', '20', '30', '40']
  hbase> create 't1', 'f1', SPLITS => ['10', '20', '30', '40']
  hbase> create 't1', 'f1', SPLITS_FILE => 'splits.txt', OWNER => 'johndoe'
  hbase> create 't1', {NAME => 'f1', VERSIONS => 5}, METADATA => { 'mykey' => 'myvalue' }
  hbase> # Optionally pre-split the table into NUMREGIONS, using
  hbase> # SPLITALGO ("HexStringSplit", "UniformSplit" or classname)
  hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}
  hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit', CONFIGURATION => {'hbase.hregion.scan.loadColumnFamilie
sOnDemand' => 'true'}}
```

Java API

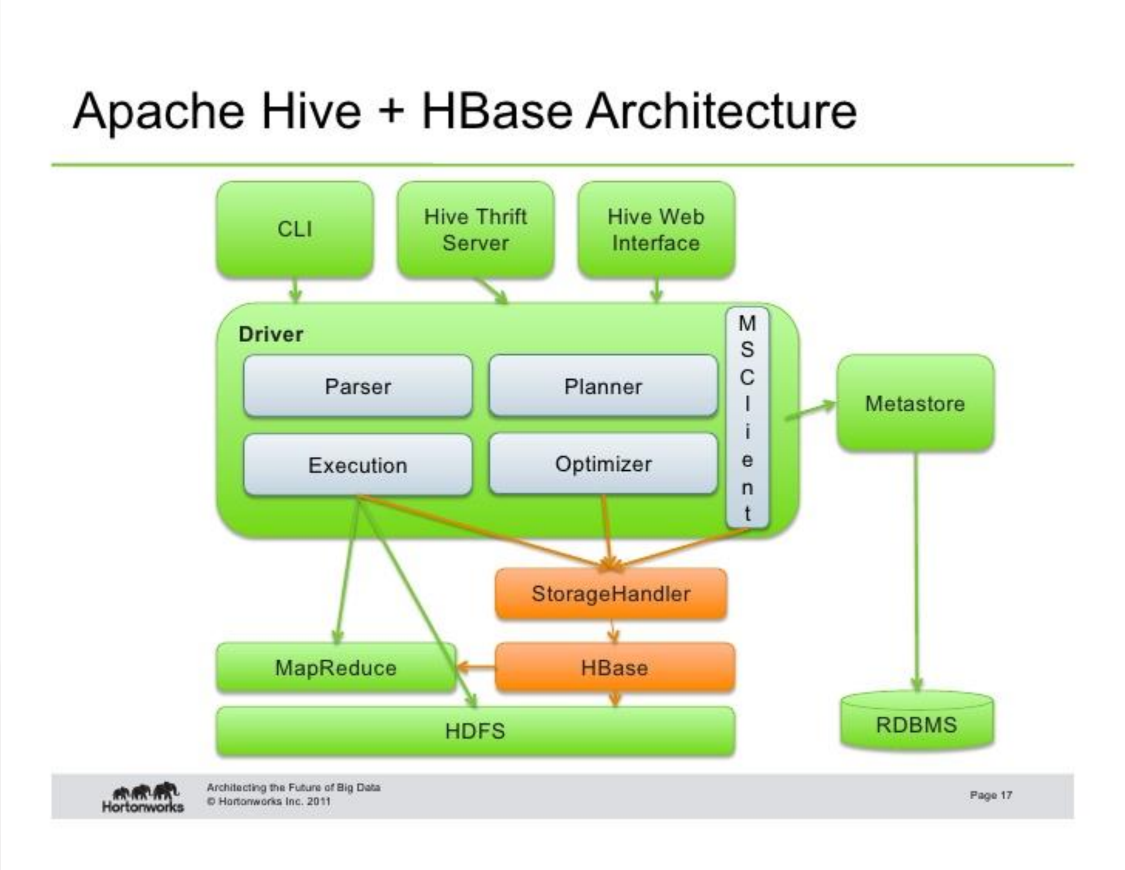


```
public class HBaseExample {
  public static void main(String[] args) throws Exception {
    AbstractHBaseDBO dbo = new HBaseDBOImpl();

    //*drop if table is already exist.*
    if(dbo.isTableExist("user")){
     dbo.deleteTable("user");
    }

    //*create table*
    dbo.createTableIfNotExist("user",HBaseOrder.DESC,"account");
    //dbo.createTableIfNotExist("user",HBaseOrder.ASC,"account");

    //create index.
    String[] cols={"id","name"};
    dbo.addIndexExistingTable("user","account",cols);

    //insert
    InsertQuery insert = dbo.createInsertQuery("user");
    UserBean bean = new UserBean();
    bean.setFamily("account");
    bean.setAge(20);
    bean.setEmail("ncanis@gmail.com");
    bean.setId("ncanis");
    bean.setName("ncanis");
    bean.setPassword("1111");
    insert.insert(bean);
```

```
//select 1 row
SelectQuery select = dbo.createSelectQuery("user");
UserBean resultBean = (UserBean)select.select(bean.getRow(),UserBean.class);

// select column value.
String value = (String)select.selectColumn(bean.getRow(),"account","id",String.class);

// search with option (QSearch has EQUAL, NOT_EQUAL, LIKE)
// select id,password,name,email from account where id='ncanis' limit startRow,20
HBaseParam param = new HBaseParam();
param.setPage(bean.getRow(),20);
param.addColumn("id","password","name","email");
param.addSearchOption("id","ncanis",QSearch.EQUAL);
select.search("account", param, UserBean.class);

// search column value is existing.
boolean isExist = select.existColumnValue("account","id","ncanis".getBytes());

// update password.
UpdateQuery update = dbo.createUpdateQuery("user");
Hashtable<String, byte[]> colsTable = new Hashtable<String, byte[]>();
colsTable.put("password","2222".getBytes());
update.update(bean.getRow(),"account",colsTable);

//delete
DeleteQuery delete = dbo.createDeleteQuery("user");
delete.deleteRow(resultBean.getRow());
```

https://hbase.apache.org/book.html#jdo
https://hbase.apache.org/apidocs/index.html

Thrift / RESTful



Apache Hive + HBase Architecture

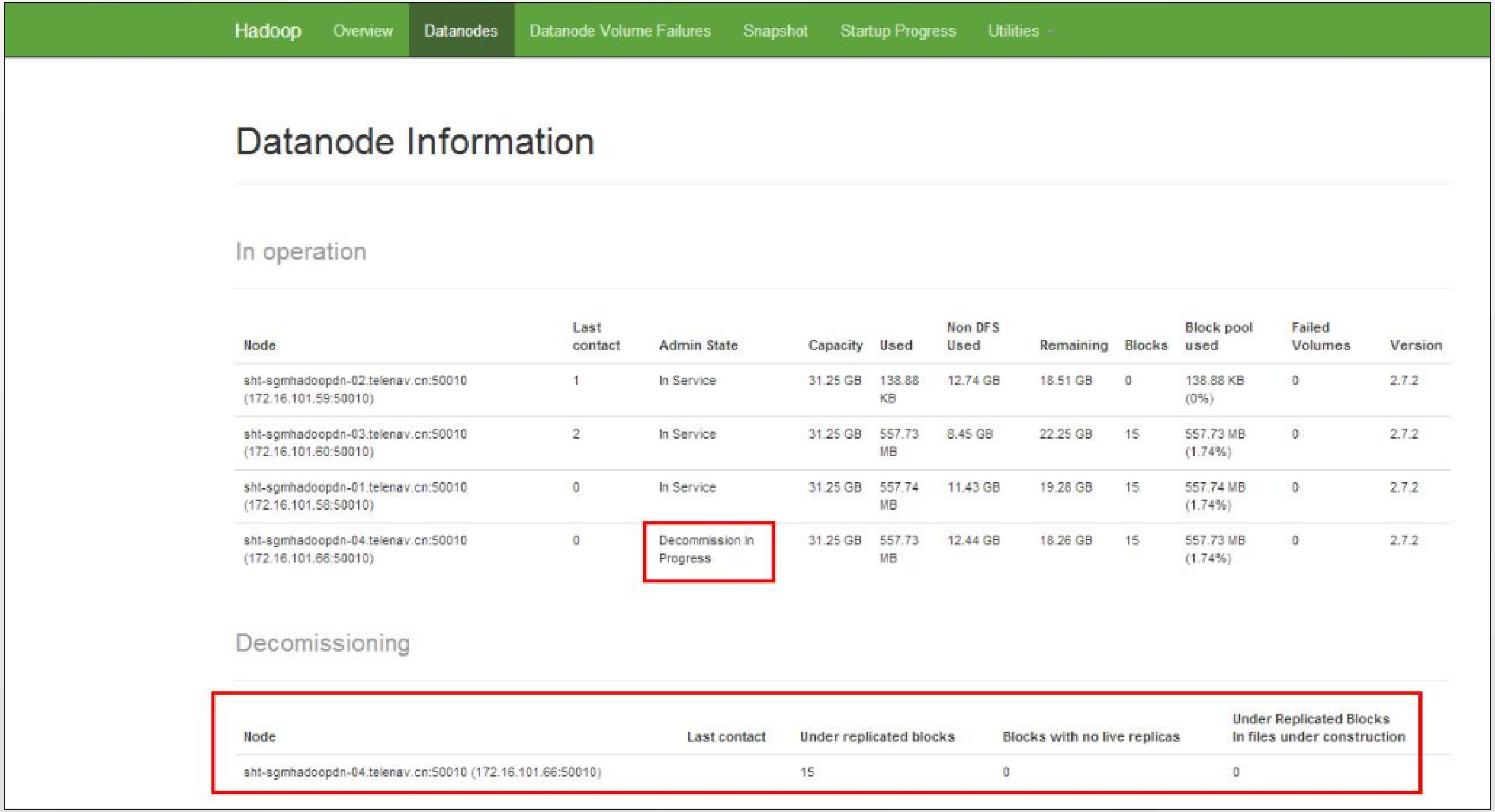
Hortonworks

Architecting the Future of Big Data
© Hortonworks Inc. 2011

PART4

Cluster & HA

Decomission



Datanode Information

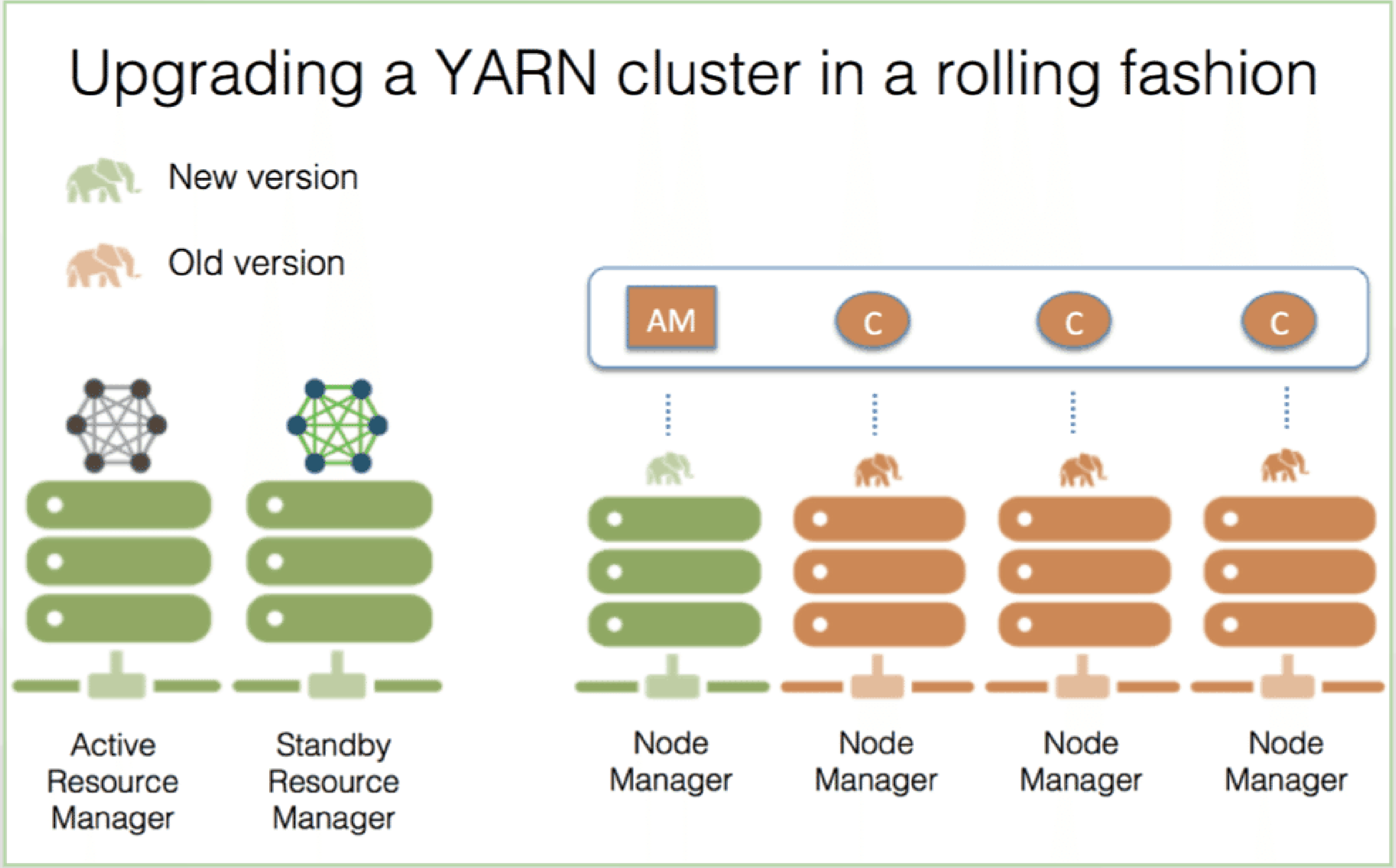
In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
sht-sgmhadoopdn-02.telenav.cn:50010 (172.16.101.59:50010)	1	In Service	31.25 GB	138.88 KB	12.74 GB	18.51 GB	0	138.88 KB (0%)	0	2.7.2
sht-sgmhadoopdn-03.telenav.cn:50010 (172.16.101.60:50010)	2	In Service	31.25 GB	557.73 MB	8.45 GB	22.25 GB	15	557.73 MB (1.74%)	0	2.7.2
sht-sgmhadoopdn-01.telenav.cn:50010 (172.16.101.58:50010)	0	In Service	31.25 GB	557.74 MB	11.43 GB	19.28 GB	15	557.74 MB (1.74%)	0	2.7.2
sht-sgmhadoopdn-04.telenav.cn:50010 (172.16.101.66:50010)	0	Decommission In Progress	31.25 GB	557.73 MB	12.44 GB	18.26 GB	15	557.73 MB (1.74%)	0	2.7.2

Decomissioning

Node	Last contact	Under replicated blocks	Blocks with no live replicas	Under Replicated Blocks In files under construction
sht-sgmhadoopdn-04.telenav.cn:50010 (172.16.101.66:50010)		15	0	0

Decomission



Upgrading a YARN cluster in a rolling fashion

New version

Old version

AM

C

C

C

Active Resource Manager

Standby Resource Manager

Node Manager

Node Manager

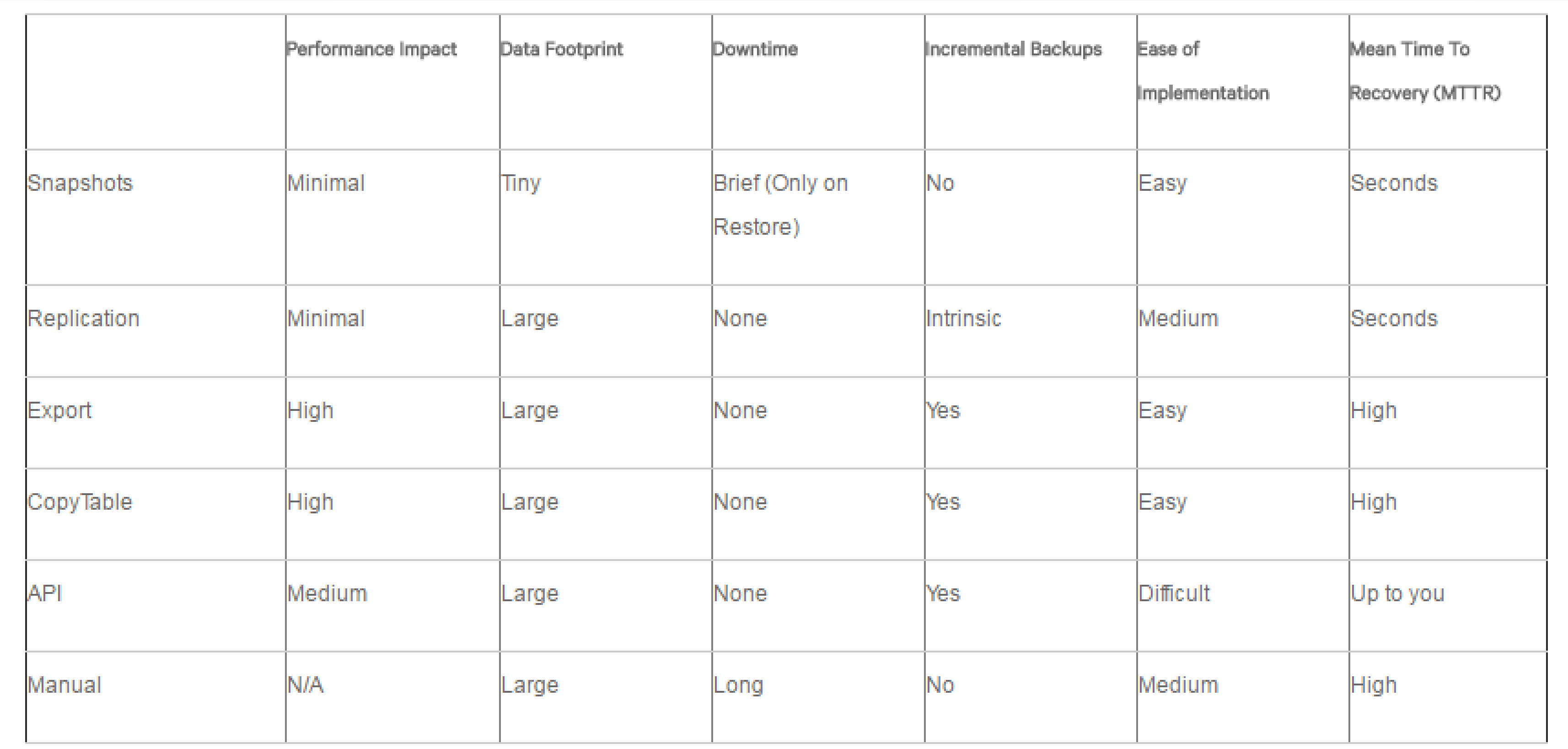
Node Manager

Node Manager

PART5

Backup & Restore

Backup & Restore



	Performance Impact	Data Footprint	Downtime	Incremental Backups	Ease of Implementation	Mean Time To Recovery (MTTR)
Snapshots	Minimal	Tiny	Brief (Only on Restore)	No	Easy	Seconds
Replication	Minimal	Large	None	Intrinsic	Medium	Seconds
Export	High	Large	None	Yes	Easy	High
CopyTable	High	Large	None	Yes	Easy	High
API	Medium	Large	None	Yes	Difficult	Up to you
Manual	N/A	Large	Long	No	Medium	High

The End