APACHE HDFS: LATEST DEVELOPMENTS & TRENDS

Jakob Homan
Microsoft CISL
Apache Software Foundation

SDC 2015

Citations needed

Lots of external references

- Easy way to reference the many, many links that will pop up…

http://bit.ly/whatever

whatever

Talk goals

Broad, cursory survey

Credit: GrandCanyonSurvey

Who I am, professionally

2008 – 2010	2010 – 2014	2014 – present
HDFS team	**Hadoop dev, Samza**	**CISL**

jghoman

Who I am, open source-ly

Distributed file system *(for our purposes)*

Distributed log

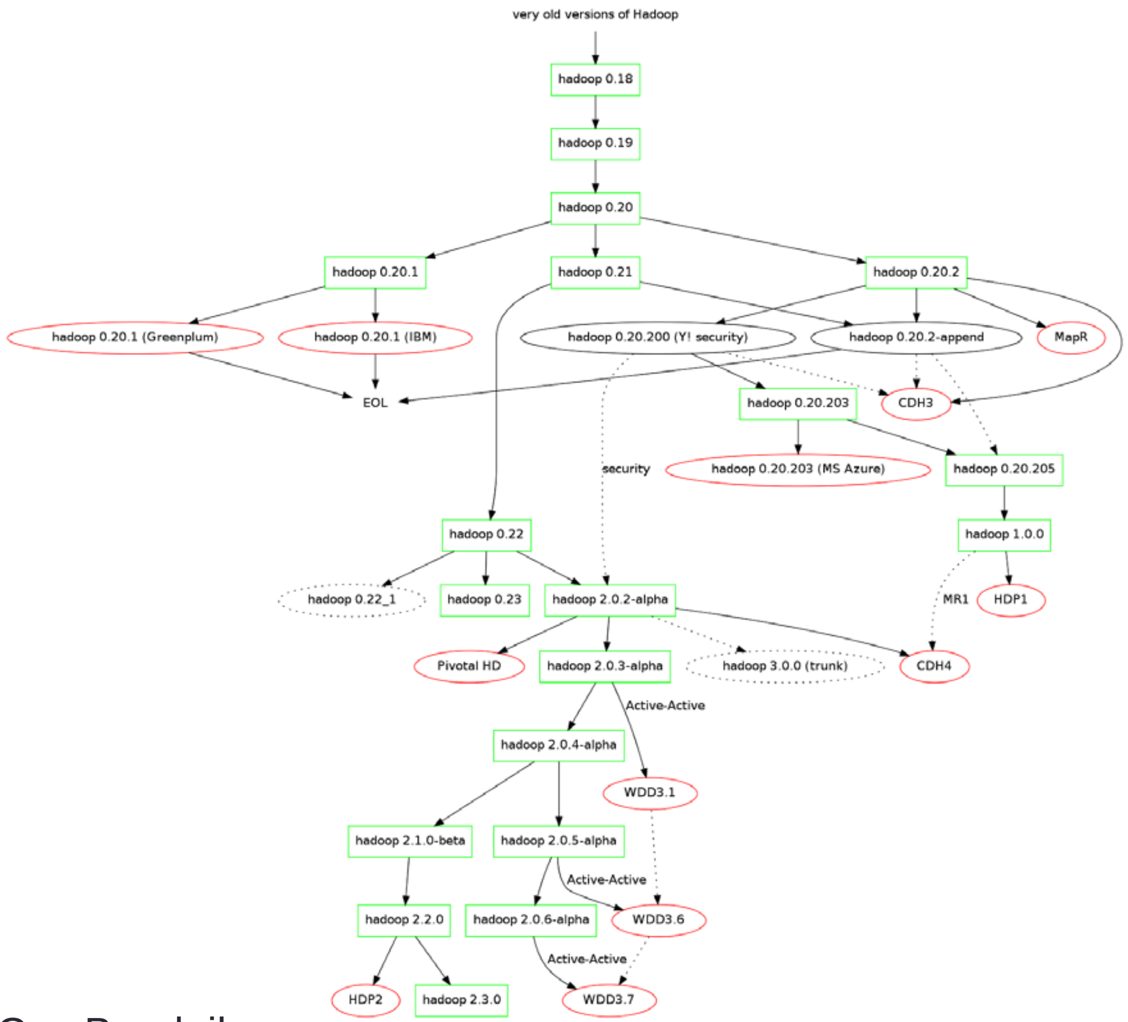
Stream processing

SQL data warehouse

Large-scale graph processing

blueboxtraveler

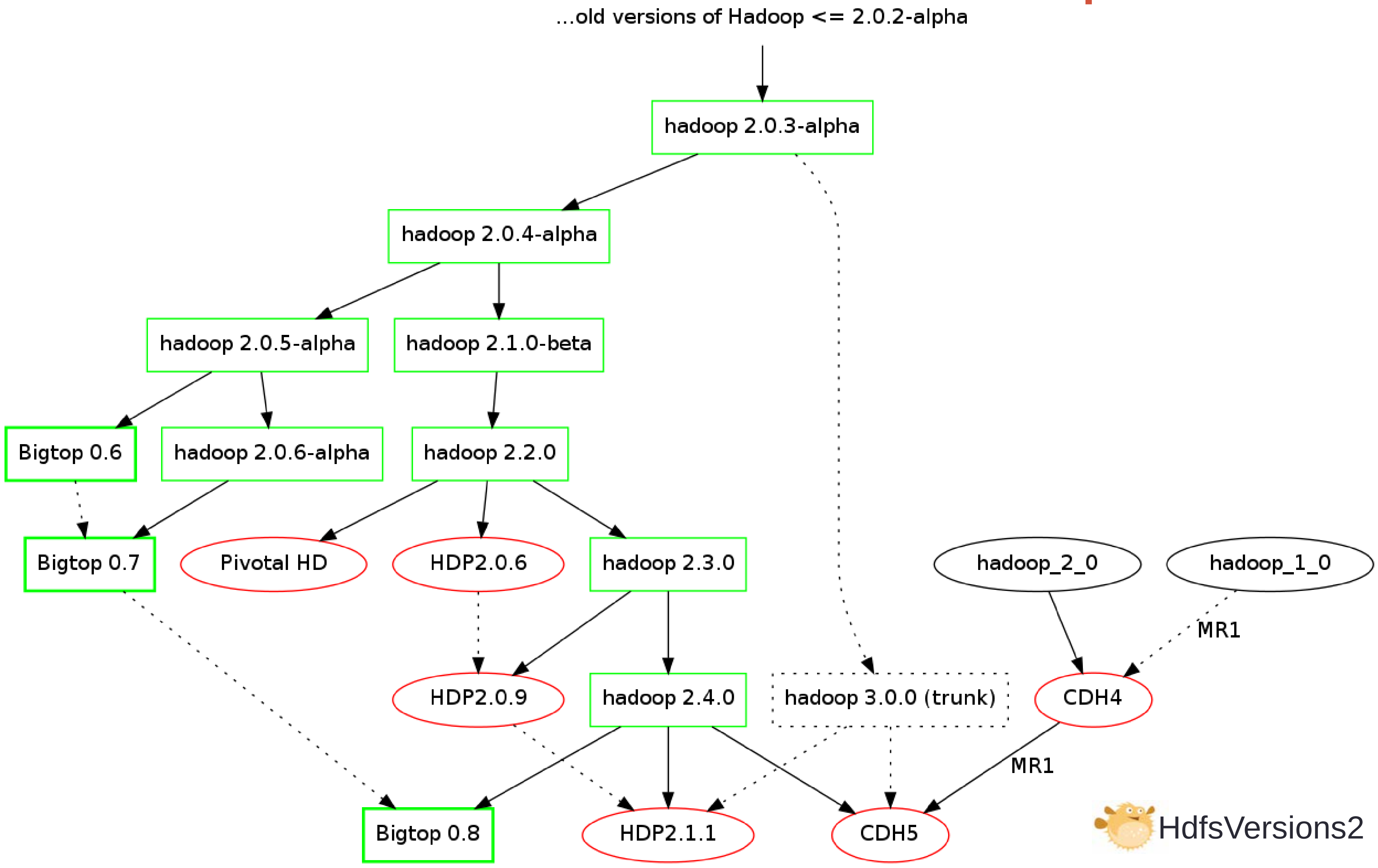
HDFS versions are complex



Graphs courtesy Cos Boudnik

HdfsVersions

HDFS versions continue to be complex



HdfsVersions2

HDFS versions, simply

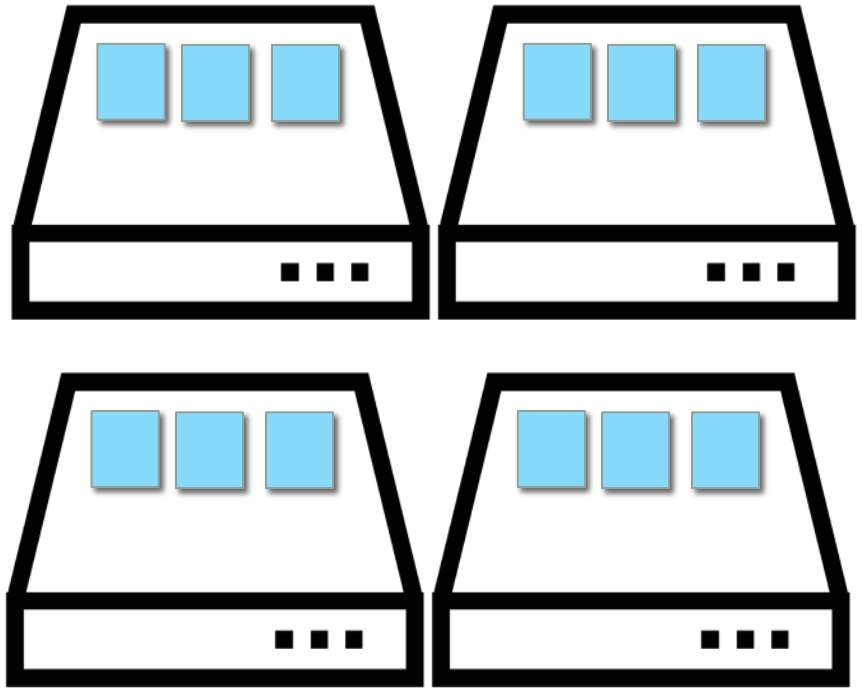
- 2.x branch
  - Modern, often released
  - New features backported
  - Bug fixes aplenty
  - 2.6.1 voted on last week
  - 2.8.0 in next few months
- Trunk
  - May some day become Hadoop 3.0
  - But maybe not…

Classic HDFS

The Namenode:

- Single server
- Store:
    - Metadata
    - Block locations
- Redirect client requests to datanodes

Classic HDFS



The Namenode:

- Single server
- Store:
  - Metadata
  - Block locations
- Redirect client requests to datanodes
- No data streams through Namenode

The Datanodes:

- Lots and lots
- Store:
  - Blocks for one namenode
- Stream client requests
- Stream replication requests

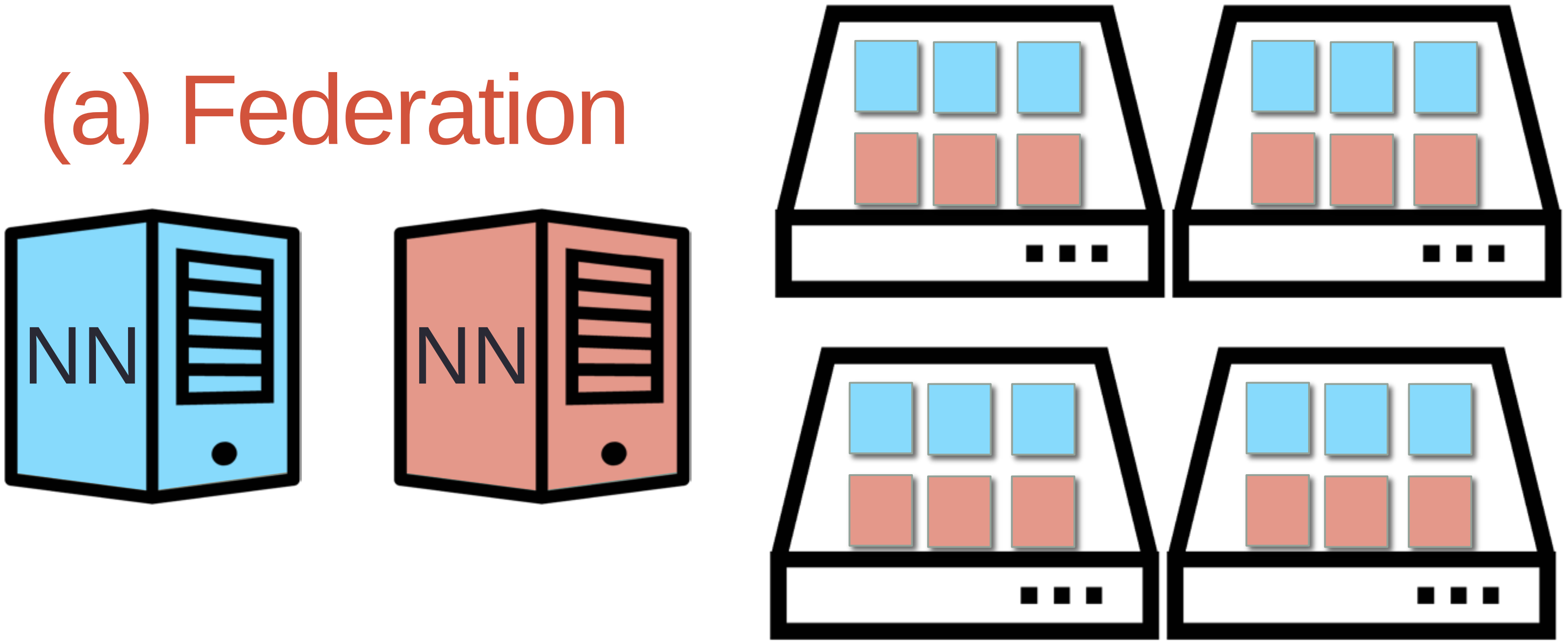
NAMENODE SCALABIILTY

Federation + Client-Side Mount Tables

Federation & client-side mount tables

- ***The problem:***
  Namenode not scaling vertically
- ***The solution:***
  (a) Partition the namespace across multiple Namenodes
  (b) Present a unified namespace to the user
- ***The implementation:***
  (a) Separate namespace and block storage
  (b) Provide NFS-style mounting to users

(a) Federation



Namenode changes:

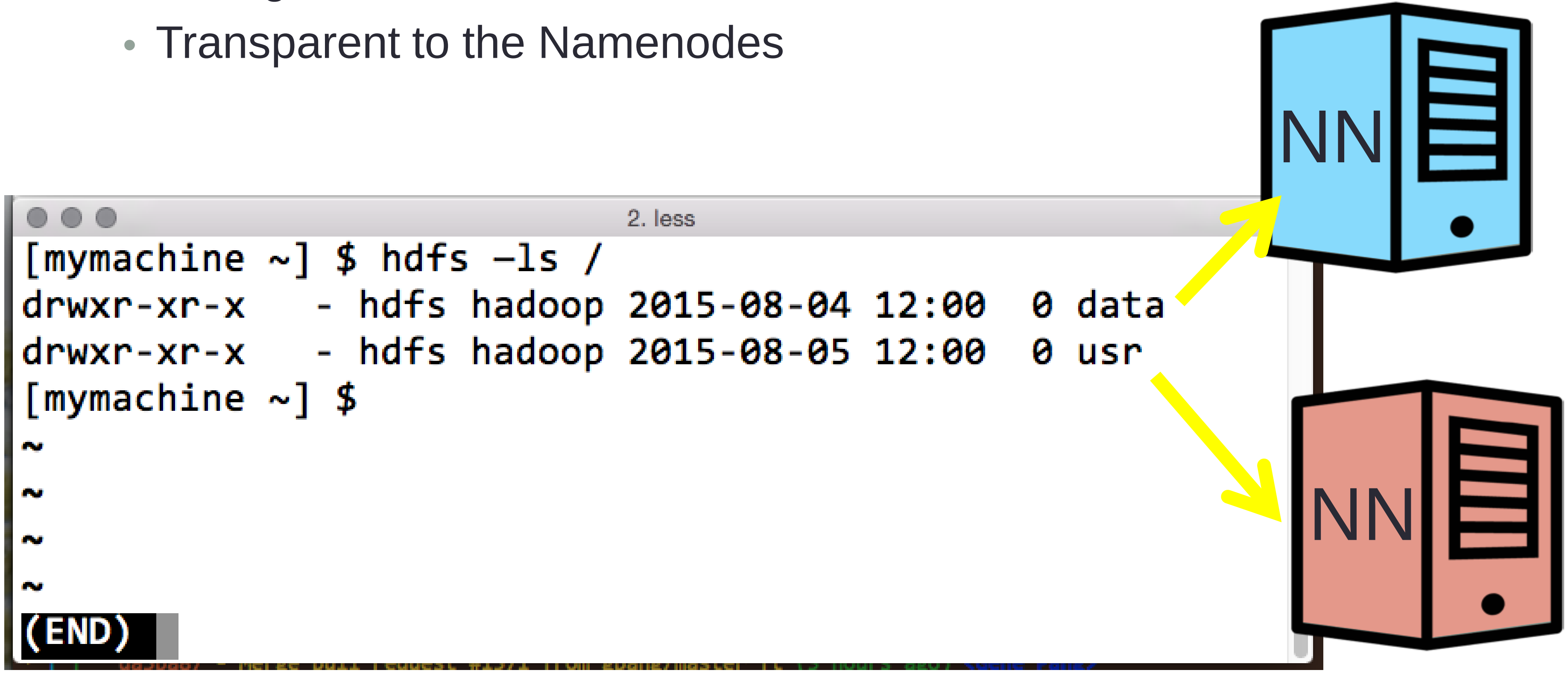
- Relatively little
- Introduce concept of blockpool ID

Datanode changes:

- Separate block storage out as a concept
- Store:
  - ~~Blocks for one namenode~~
  - Blocks for multiple namenodes

(b) Client-side mount tables

- Make federation transparent to end users
  - Configured on the client-side
  - Transparent to the Namenodes



```
[mymachine ~] $ hdfs -ls /
drwxr-xr-x   - hdfs hadoop 2015-08-04 12:00  0 data
drwxr-xr-x   - hdfs hadoop 2015-08-05 12:00  0 usr
[mymachine ~] $
~
~
~
~
(END)
```

Federation + CSMTs

- Pros
  - Relatively small changes to Namenode
  - Isolation
  - Performance gains
- Cons
  - Sidesteps inherent Namenode limitation
  - Requires configuration management

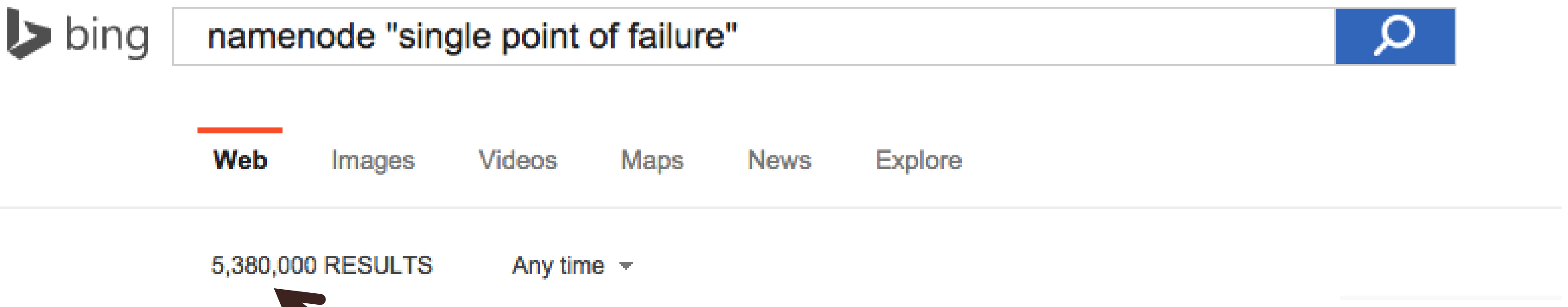
Earliest version: 0.23

More detail: HdfsFederation

Original Apache ticket: HDFS-1052

NAMENODE AVAILABILITY

Single Point Of Failure?


namenode "single point of failure"

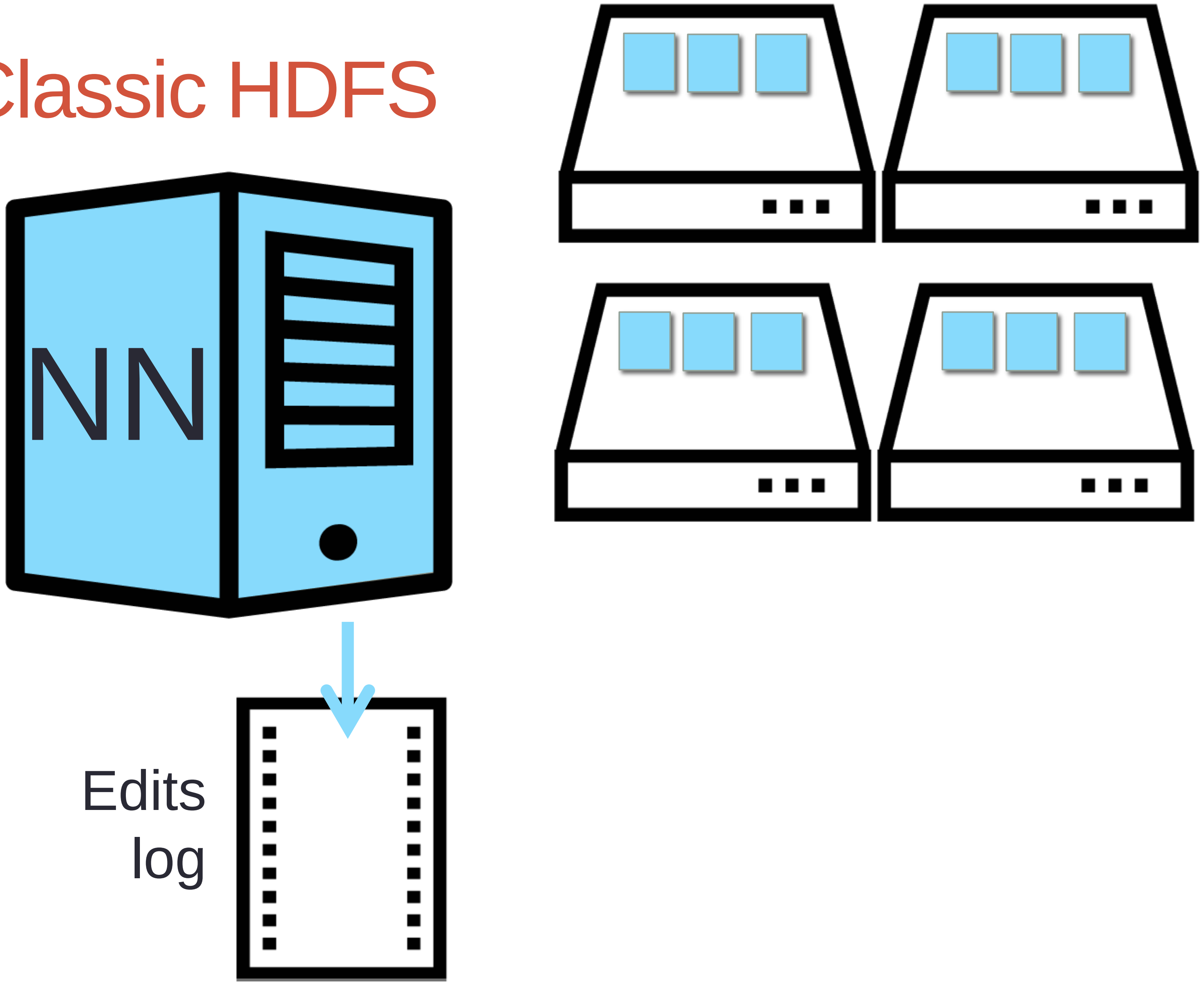
5,380,000 RESULTS

Yeah, sure, but…

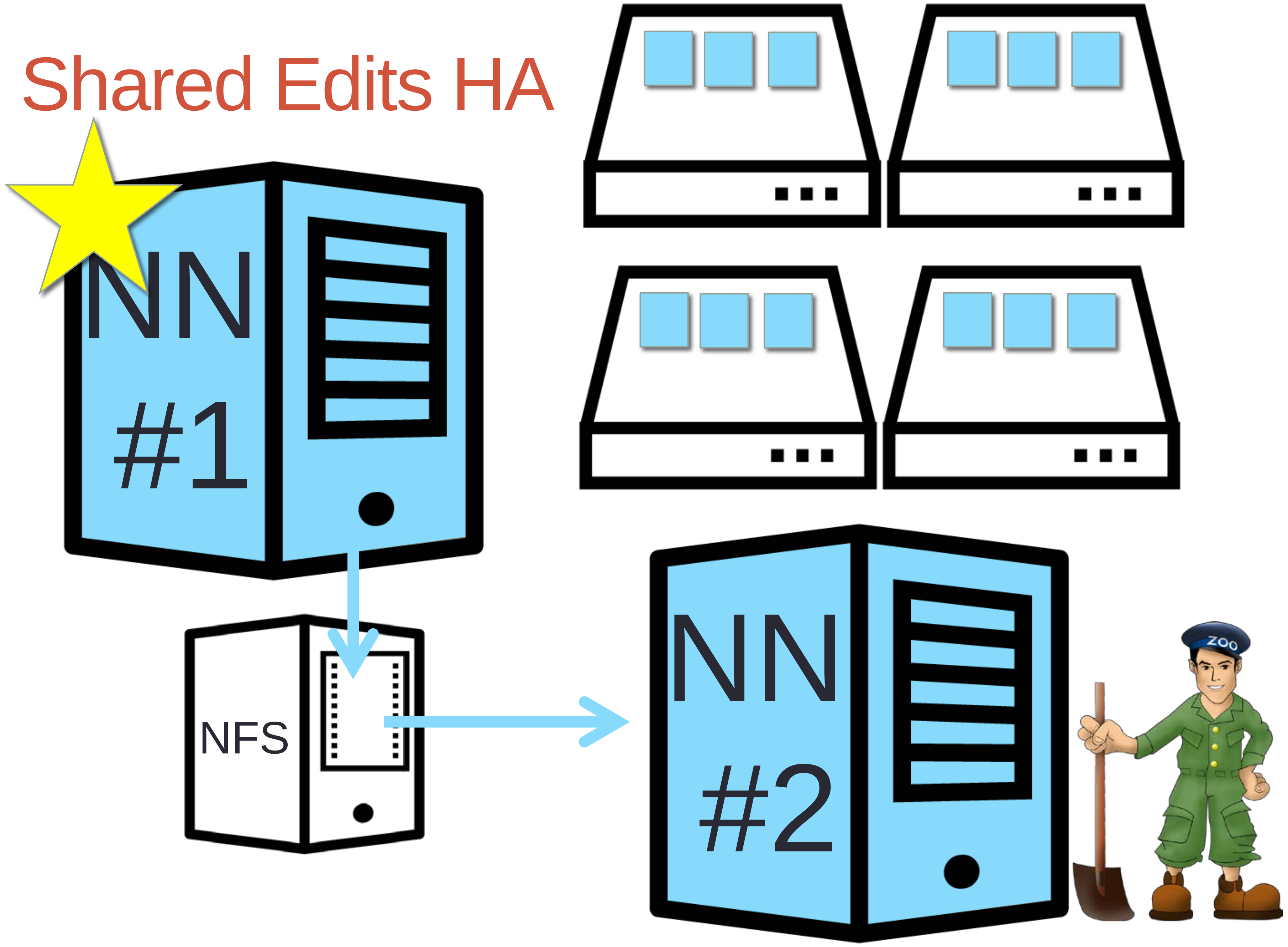
- High-availability:
  A problem so nice, we solved it twice

Approach #1: Shared edits log

Classic HDFS

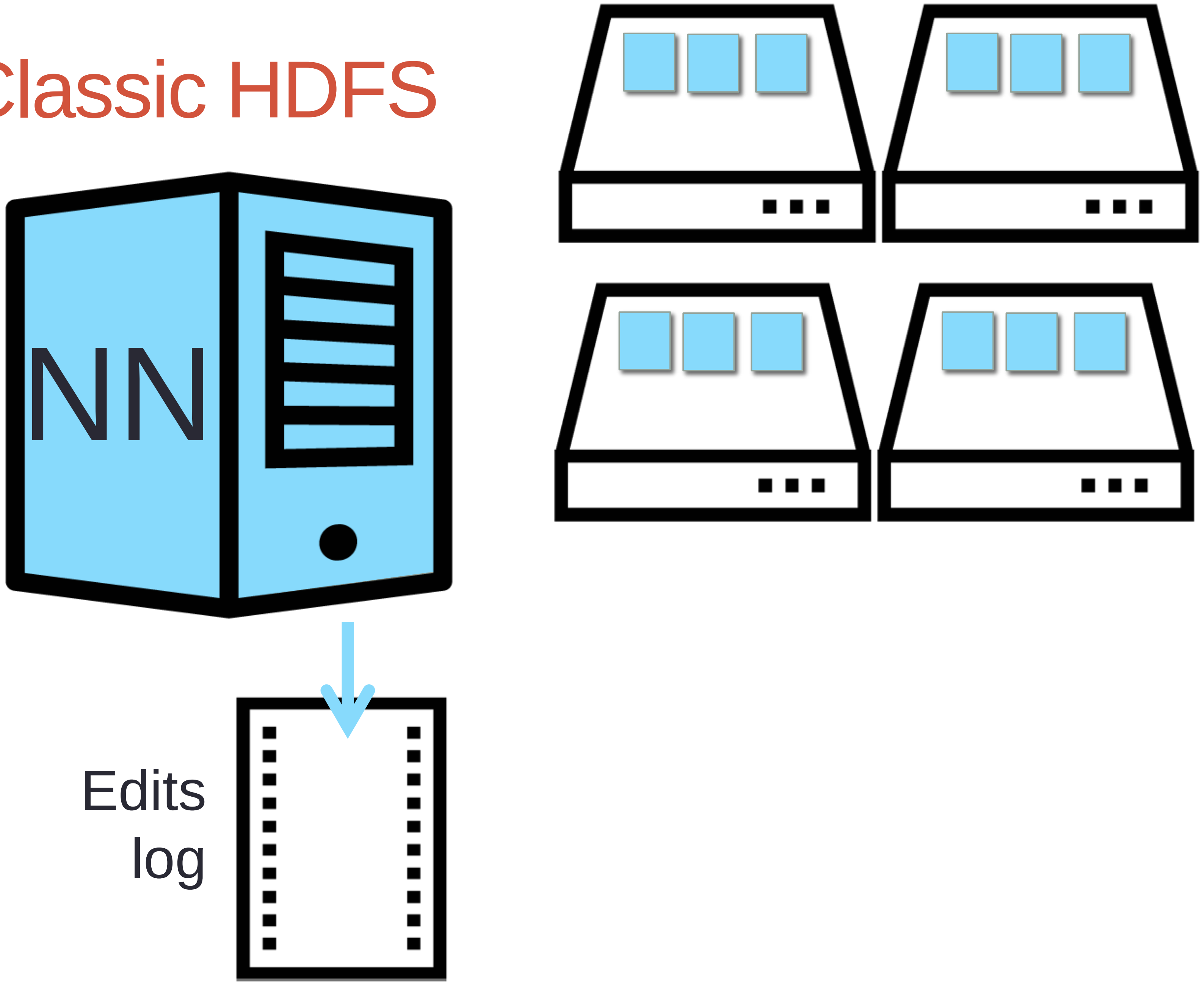


Shared Edits HA

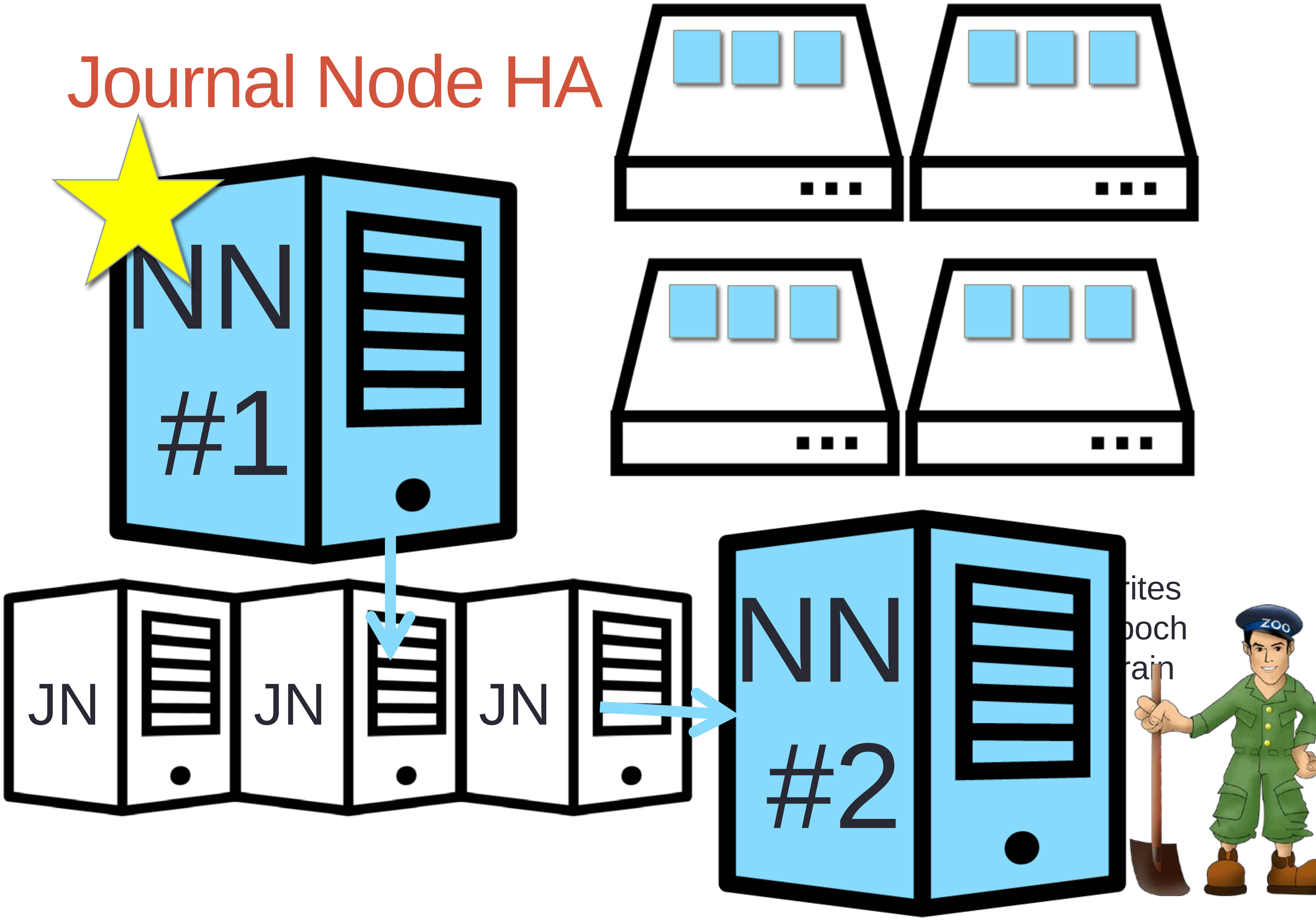


Approach #2: Quorum Journal Managers

Classic HDFS



Journal Node HA



NFS- vs JN-based High Availability

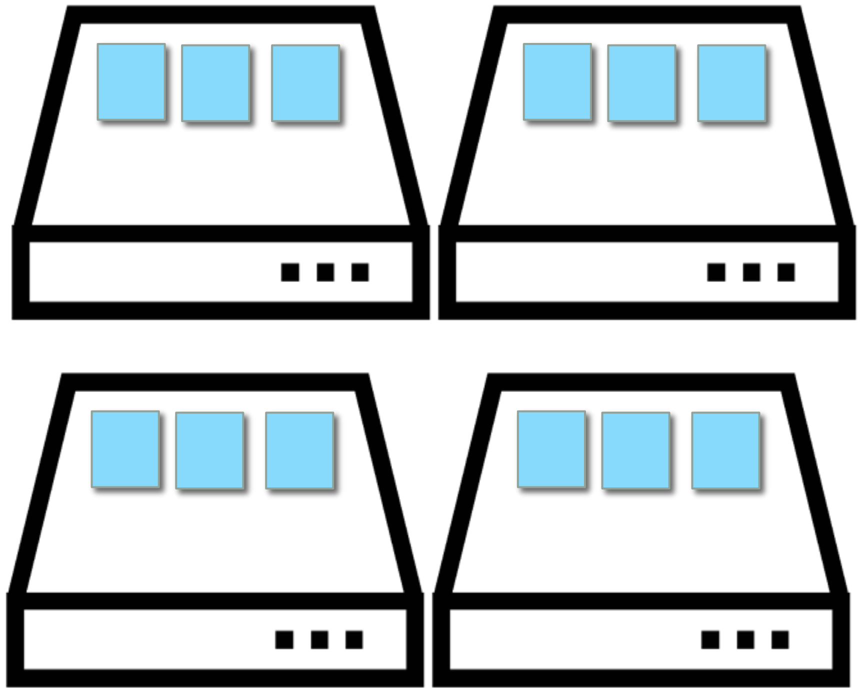
	NFS	Journal Node
New requirement	Reliable NFS	Journal nodes x 3,5,7
Remaining point of failure	NFS	Quorum majority
ZooKeeper fencing	Required	Recommended
Earliest version	2.0	2.0
JIRA ticket	HDFS-1623	HDFS-3077

Notes:

- Both Hortonworks and Cloudera recommend JN-based HA
- More than two namenodes coming in 3.0: HDFS-6440

HETEROGENEOUS STORAGE

Classic HDFS



The Namenode:

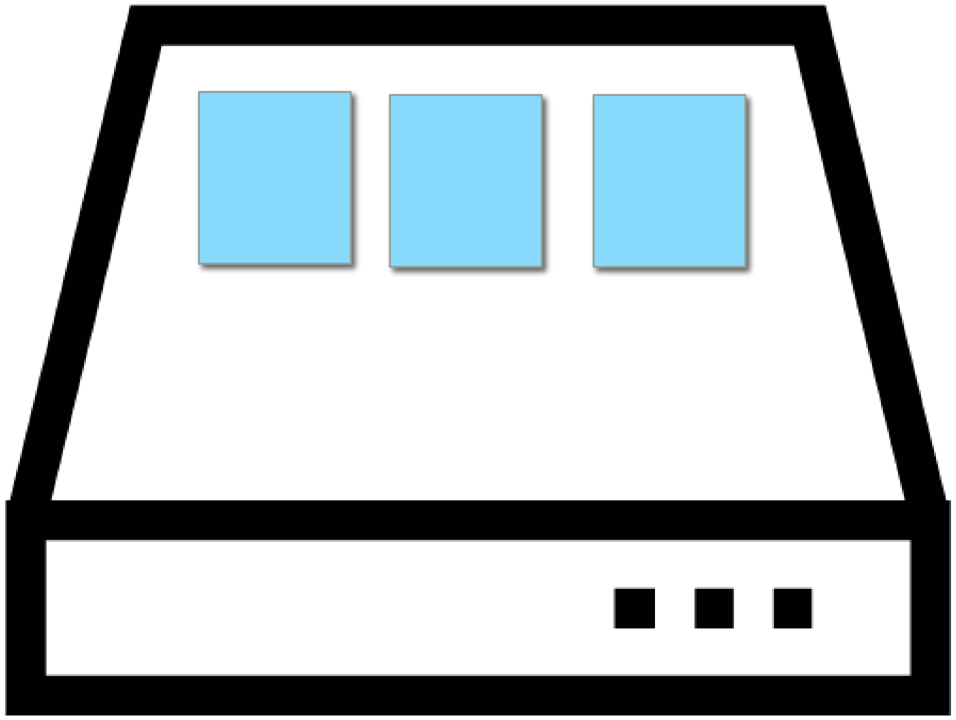
- Single server
- Store:
  - Metadata
  - Block locations
- Redirect client requests to datanodes
- No data streams through Namenode

The Datanodes:

- Lots and lots
- Store:
  - Blocks for one namenode
- Stream client requests
- Stream replication requests

Classic HDFS – looking at datanodes

No distinction between storage types



Introduce new storage types

RAM_DISK

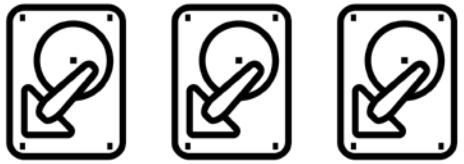
SSD

DISK

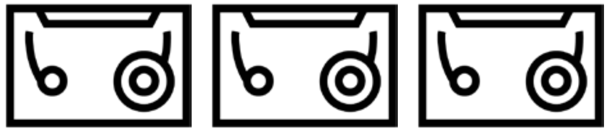
ARCHIVE

Introduce new storage strategies

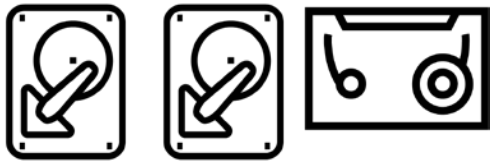
HOT



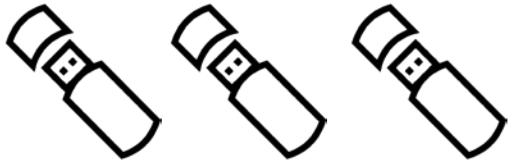
COLD



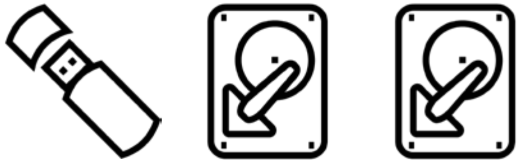
WARM



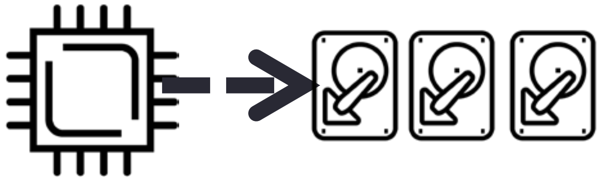
All_SSD



One_SSD



Lazy_ PERSIST



Assign policies to HDFS directories

- Policy ID
  - *Example: 72*
- Policy Name
  - *Example: ReallyBigNodeType*
- Block placement (in replicas)
  - *Example: { RAM_DISK: 1, SSD: 1, DISK: 1 }*
- Fallback file creation
  - *Example: SSD*
- Fallback replication
  - *Example: DISK*

Tools, additional efforts

hdfs mover

Tool to scan directories, looking for better storage policy compliance

het_hdfs

Block pinning

Client-side request to pin specific blocks in datanode memory

block_pinning

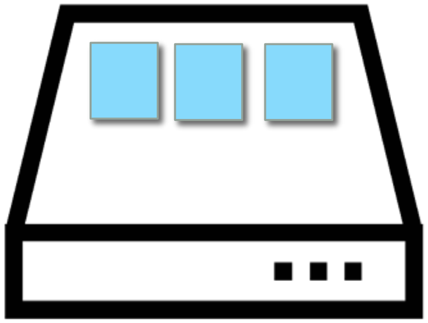
memfs

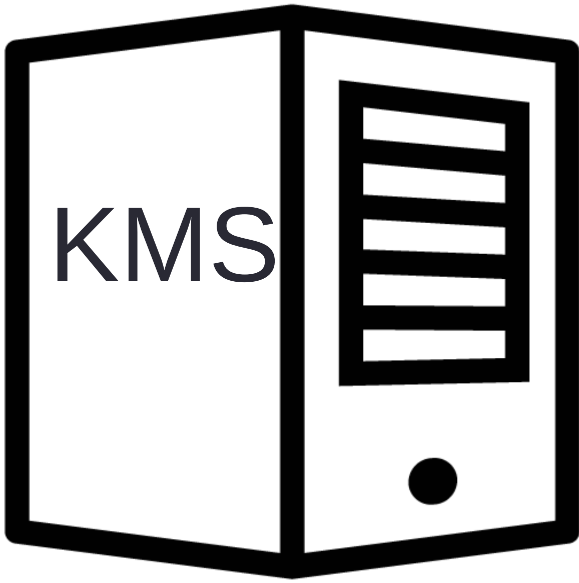
Proposal for entirely in-memory filesystem implementation

HDFS-8401

END-TO-END ENCRYPTION

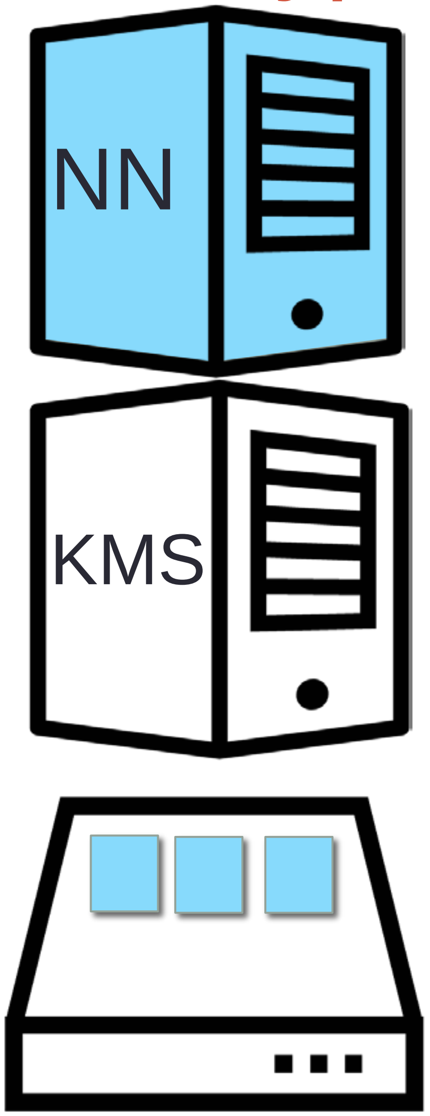
Classic HDFS





- Key Management Server
    - Stores per-directory encryption keys (DEKs)
    - NameNode stores per-file encryption keys (

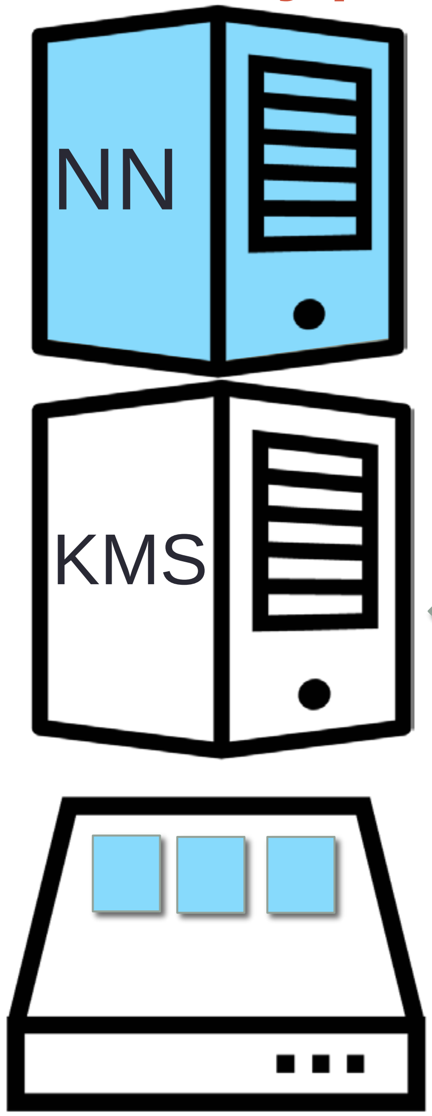
Encryption: a step-by-step guide



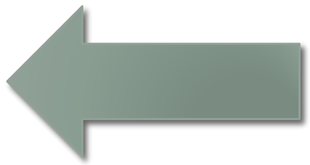
Key Management Server

- Stores per-directory encryption key used to encrypt per-file keys

Encryption: a step-by-step guide (1)

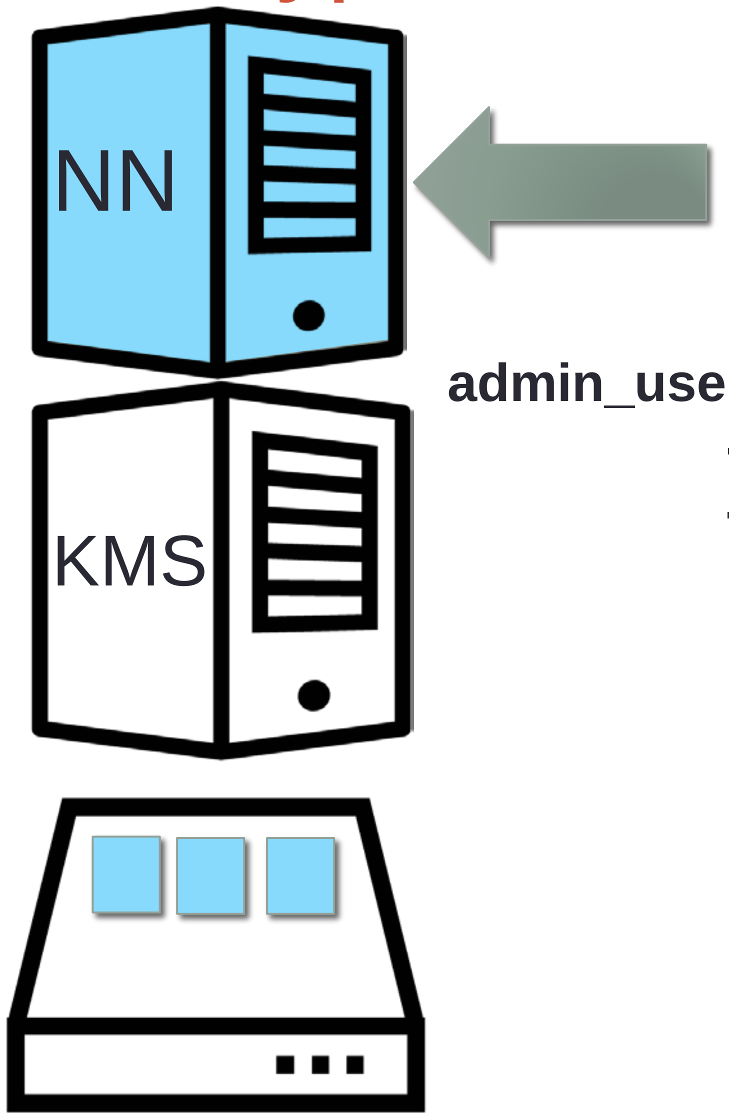


```
regular_user> hadoop key create JakobsKey
```



Persist JakobsKey to KMS

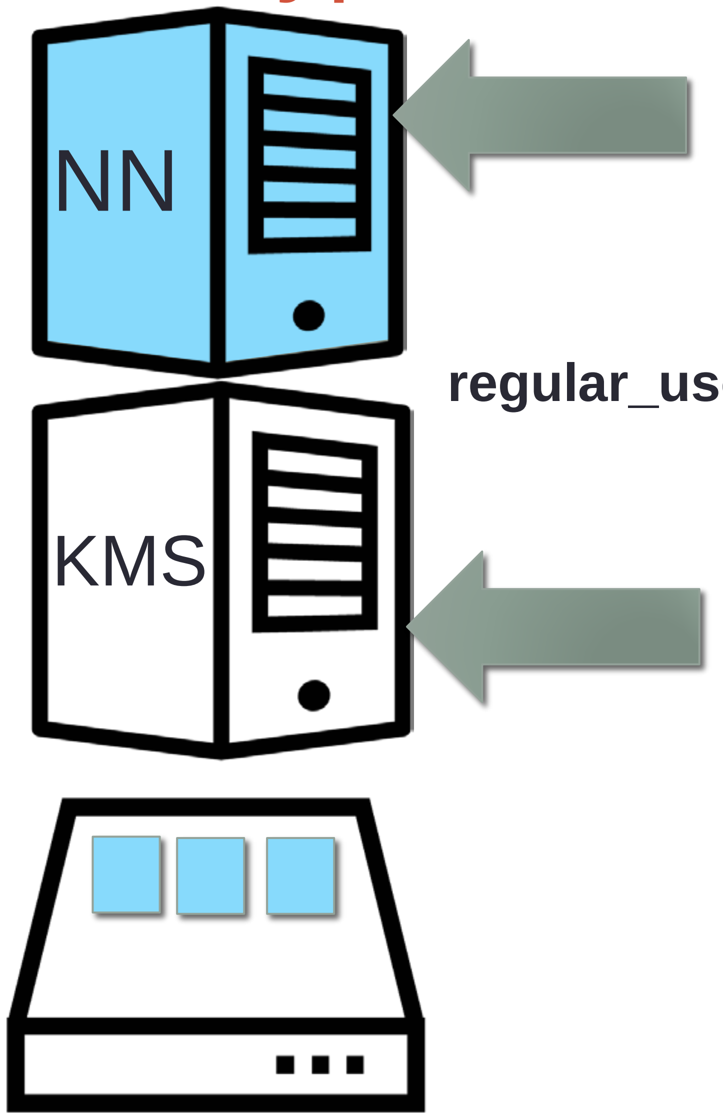
Encryption: a step-by-step guide (2)



Everything in /User/Jakob

```
admin_user> hdfs crypto -createZone \
                  -keyName JakobsKey
                  -path /Users/Jakob
```

Encryption: a step-by-step guide (3)

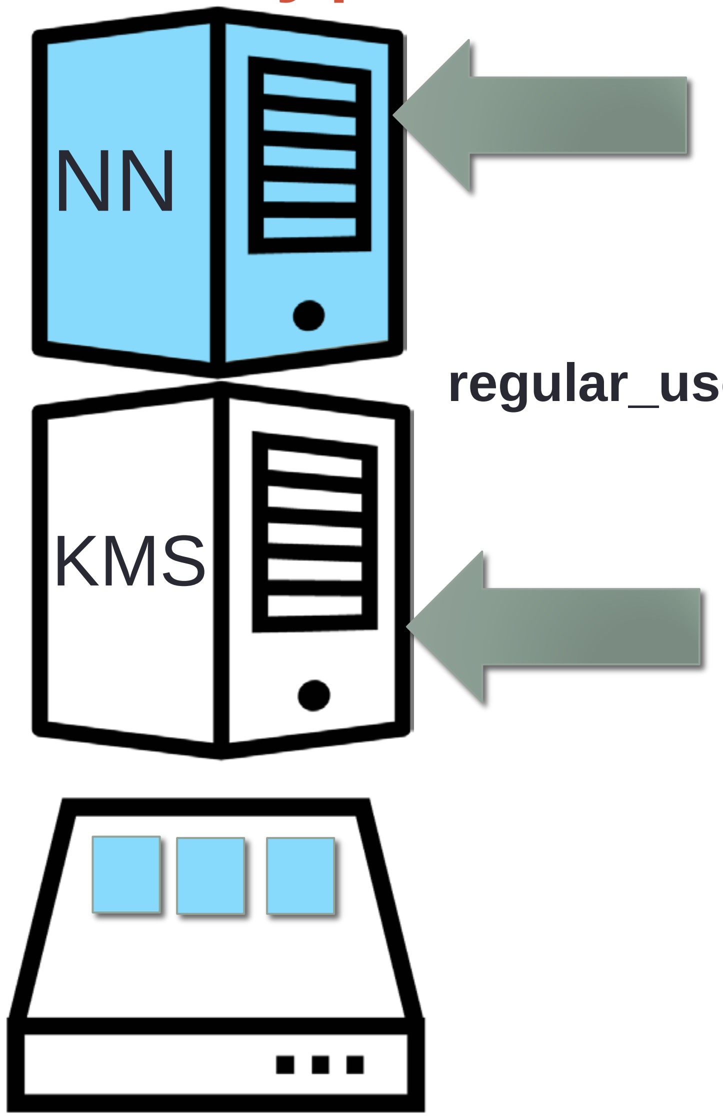


NameNode stores EDEK with *myfile* metadata

```
regular_user> hdfs –copyFromLocal myfile \
/Users/Jakob/myfile
```

KMS uses JakobKey to encrypt a new key (EDEK) for *myfile*

Encryption: a step-by-step guide (3)



Client obtains EDEK while accessing file

```
regular_user> hdfs –cat \
                    /Users/Jakob/myfile
```

Client

(a) authenticates to KMS, decrypts EDEK

(b) Decrypts file

Encryption key points

- HDFS never sees unencrypted data
- HDFS never sees unencrypted data access key
- After configuration, encryption is transparent to user
- Un-encrypted raw data available for bulk transfer

Earliest version: 2.3.0

hdfs_encrypt

NEW FILE APIs

Truncate and improved concat

Truncate

- ***Previously***
  - Append-only file access

```
stream = FS.create(somePath);
stream.write(someBytes);
stream.write(moreBytes);
stream.write(evenMoreBytes);
stream.close();
```

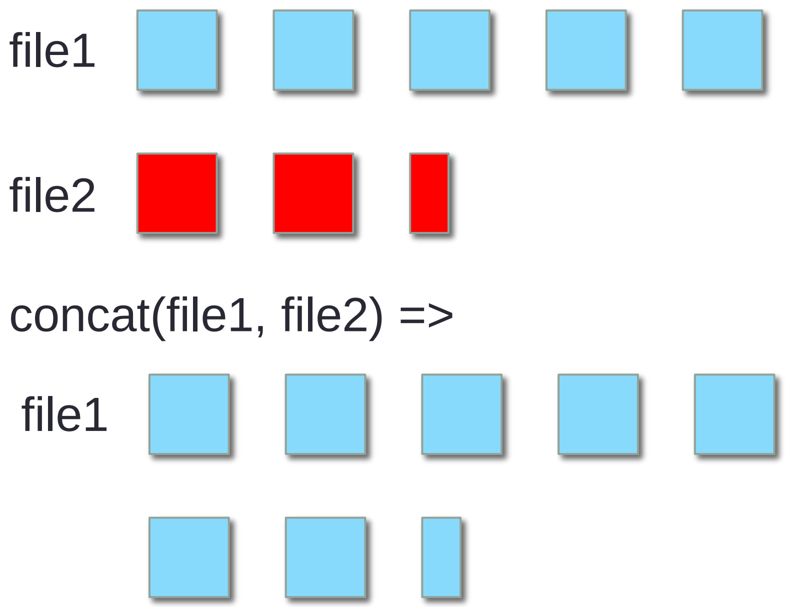
- ***Now***
  - Truncate at position
    - Undo mistakes
    - Support transactions
    - Recover from failures
  - Appears in version 2.7.0

```
stream = FS.create(somePath);
stream.write(someBytes);
stream.close()
FS.truncate(somePath, 4096);
stream = FS.append(somePath)
stream.write(newBytes)
```

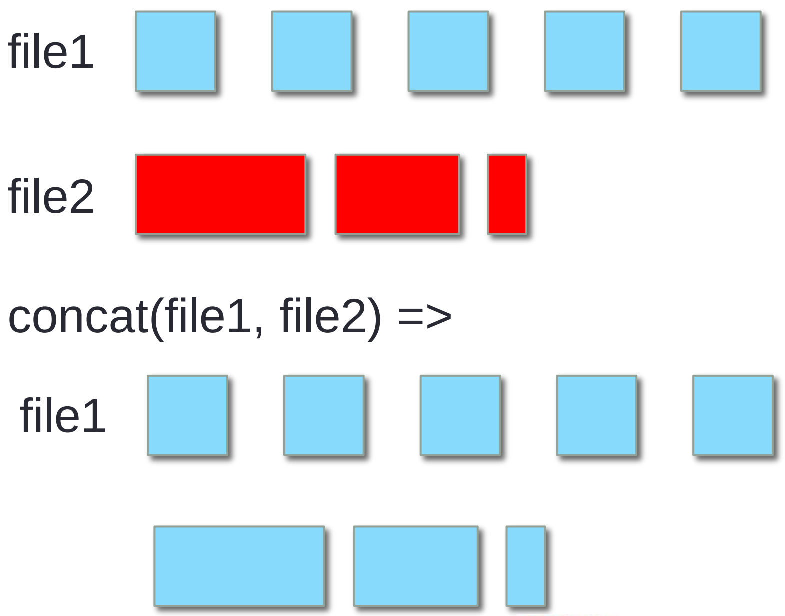
TruncateTalk HDFS-3107

Better concat operation

- ***Previously***
  - Strict requirements to concatenate files



- ***Now***
  - Variable length blocks permitted
  - Appears in version 2.7.0



HDFS-3689

ON-GOING WORK

HDFS in 2016

On the horizon

Project Ozone

Store objects other than HDFS files in DataNodes

HDFS-7240

ProjectOzone

Scaling the NameNode

Move NN metadata to pluggable kv-value store

HDFS-8286

kvNameNode

Erasure Coding

Replace 3x block replication with compressed, coded data

HDFS-7285

hdfs-ec

THANKS!

Questions?