

OK S3

Строим Систему Сами

Вадим Цесько



одноклассники
экосистема 



Odnoklassniki¹

- > **70M users** monthly
- > **10K servers**
- **7 datacenters**
- > **4 Tbps**
- > **100K rps** per node (4 cores)
- **p99 < 100 ms**
- **1 EB** and growing



¹OK tech platform hosts VK Video, VK Calls, RuStore.

Why S3?

- Docker Registry
- Sonatype Nexus
- JetBrains Teamcity
- Apache Airflow
- ML/DS, RPM, CDN, static websites, autotest artifacts, dumps/traces, backups, ...



Reference documentation

Command-line reference ▾

API reference ▾

Dockerfile reference

Compose file reference ▾

Drivers and specifications ▾

Registry image manifests ▾

Registry token authorization ▾

Registry storage drivers ▾

Storage driver overview

Aliyun OSS storage driver

Filesystem storage driver

GCS storage driver

In-memory storage driver

Microsoft Azure storage driver

S3 storage driver

Docker Registry storage driver

Estimated reading time: 2 minutes

This page contains information about hosting your own registry using the open source Docker Registry. For information about Docker Hub, which offers a hosted registry with additional features such as teams, organizations, web hooks, automated builds, etc, see [Docker Hub](#).

This document describes the registry storage driver model, implementation, and explains how to contribute new storage drivers.

Provided drivers

This storage driver package comes bundled with several drivers:

- **inmemory**: A temporary storage driver using a local inmemory map. This exists solely for reference and testing.
- **filesystem**: A local storage driver configured to use a directory tree in the local filesystem.
- **s3**: A driver storing objects in an Amazon Simple Storage Service (S3) bucket.
- **azure**: A driver storing objects in [Microsoft Azure Blob Storage](#).
- **swift**: A driver storing objects in [Openstack Swift](#).
- **oss**: A driver storing objects in [Aliyun OSS](#).
- **gcs**: A driver storing objects in a [Google Cloud Storage](#) bucket.

Storage driver API

The storage driver API is designed to model a filesystem-like key/value storage in a manner abstract enough to support a range of drivers from the local filesystem to Amazon S3 or other distributed object storage systems.

Search the docs...

Switch to another product

NEXUS REPOSITORY MANAGER 3

Product Information

Planning Your Implementation

Installation and Upgrades

Nexus Repository Administration

Administration Menu

Repository Management

Configuring Blob Stores

Cleanup Policies

Routing Rules

Repository Replication

Repository Health Check

Formats

Staging

[Nexus Repository Manager 3](#) > [Nexus Repository Administration](#) > [Repository Management](#) > [Configuring Blob Stores](#)

Configuring Blob Stores

Before configuring blob stores, be sure to check out our [documentation](#) about blob store types and planning storage requirements.

You can configure new blob stores by navigating to *Administration* → *Repository* → *Blob Stores* in Nexus Repository. You will need *nx-all* or *nx-blobstore* [privileges](#) to access this portion of Nexus Repository.

The following fields appear in the blob store listing:

- *Name* - The blob store's name as displayed in repository administration.
- *Type* - The type of the blob store backend. See the [System Requirements](#) for a full list of supported file systems. The following options are available:
 - *Azure Cloud Storage* **PRO** - Stores blobs in Azure cloud storage.
 - *File* - Store blobs in file system-based storage.
 - *Group* **PRO** - Combines multiple blob stores into one.
 - *S3* - Store blobs in AWS S3 cloud storage.



airflow.apache.org/docs/apache-airflow-providers-amazon/stable/logging/index.html



[Community](#)

[Meetups](#)

[Documentation](#)

[Use-cases](#)

[Announcements](#)

[Blog](#)

[Ecosystem](#)

Version: 3.4.0 ▾

Search docs



GUIDES

[Connection types](#)

[Operators](#)

[Secrets backends](#)

[Logging for Tasks](#)

[Home](#) / [Logging for Tasks](#)

Logging for Tasks

- [Writing logs to Amazon Cloudwatch](#)
- [Writing logs to Amazon S3](#)

[Previous](#)



▼ Managing Projects

Creating and Editing Projects

Project Export

Projects Import

▶ Configuring VCS Settings

▶ Configuring Connections

▶ Storing Project Settings in Version Control

Ordering Projects and Build Configurations

Configuring Cross-Server Projects Pop-up Menu

Archiving Projects

Customizing Statistics Charts

▼ Configuring Artifacts Storage

Storing Build Artifacts in Amazon S3

Managing Projects / [Configuring Artifacts Storage](#) / Storing Build Artifacts in Amazon S3

Storing Build Artifacts in Amazon S3

[Edit page](#) Last modified: 11 May 2022

TeamCity comes bundled with the [Amazon S3 Artifact Storage](#) ↗ plugin which allows storing build artifacts in an Amazon S3 bucket.

It is possible to replace the TeamCity built-in artifacts' storage with [Amazon S3](#) ↗ at a project level. When an S3 artifacts storage is configured, it:

- allows uploading to, downloading, and removing artifacts from S3;
- handles resolution of artifact dependencies as well as clean-up of artifacts;
- displays artifacts located externally in the TeamCity UI.

Pluggable storage backends

- OpenStack Object Storage API
- Google Cloud Storage
- Azure Blob service REST API
- WebDAV
- **Amazon S3**
 - **Customizable service endpoint**



Rich ecosystem over SDKs

- [AWS SDK for C++ Developer Guide](#)
- [AWS SDK for Go Developer Guide](#) 
- [AWS SDK for Java Developer Guide](#)
- [AWS SDK for JavaScript Developer Guide](#)
- [AWS SDK for .NET Developer Guide](#)
- [AWS SDK for PHP Developer Guide](#)
- [AWS SDK for Python \(Boto3\) Getting Started](#) 
- [AWS SDK for Ruby Developer Guide](#)



Simple Storage Service²

- AWS Service and **API reference**
- Buckets
 - Unique
 - `https://<bucket>.s3.ok.ru/`
 - `https://s3.ok.ru/<bucket>/`
- Objects
 - **Immutable** and optionally **versioned**
 - `https://<bucket>.s3.ok.ru/<object>`
 - eTag — MD5
 - Last-Modified (seconds precision)

²<https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html>



Object limits

- **Unlimited** objects per bucket
- **Key** up to 1024 chars
- **Meta** up to 2 KB
- Up to 10 **tags: name + value**
- Up to **10K parts** per object
- Minimal **part size** 5 MB
- Maximum multipart **object size** 5 TB



Core S3 HTTP API

- ListBuckets
- {Get/Head}Object + If-* headers
- {Put/Delete}Object + DeleteObjects
- CopyObject
- {Get/Put/Delete}ObjectTagging
- ListObjects + prefix/delimiter/marker
- ListObject**Versions** + prefix/delimiter/marker



ListObjects with **prefix**

```
1 GET /?prefix=photos/ HTTP/1.1
```

```
1 photos/2021/1.jpg
```

```
2 photos/2021/2.jpg
```

```
3 photos/2022/1.jpg
```

```
4 photos/2022/2.jpg
```

```
5 photos/2022/3.jpg
```

```
6 photos/dog.jpg
```

```
7 photos/logo.jpg
```



ListObjects with prefix and marker

```
1 GET /?prefix=photos/&marker=photos/2021/2.jpg HTTP/1.1
```

```
1 photos/2022/1.jpg
```

```
2 photos/2022/2.jpg
```

```
3 photos/2022/3.jpg
```

```
4 photos/dog.jpg
```

```
5 photos/logo.jpg
```



ListObjects with prefix and **delimiter**

```
1 GET /?prefix=photos/&delimiter=/ HTTP/1.1
```

Common prefixes of keys with delimiter:

- 1 photos/2021/
- 2 photos/2022/

Keys without delimiter:

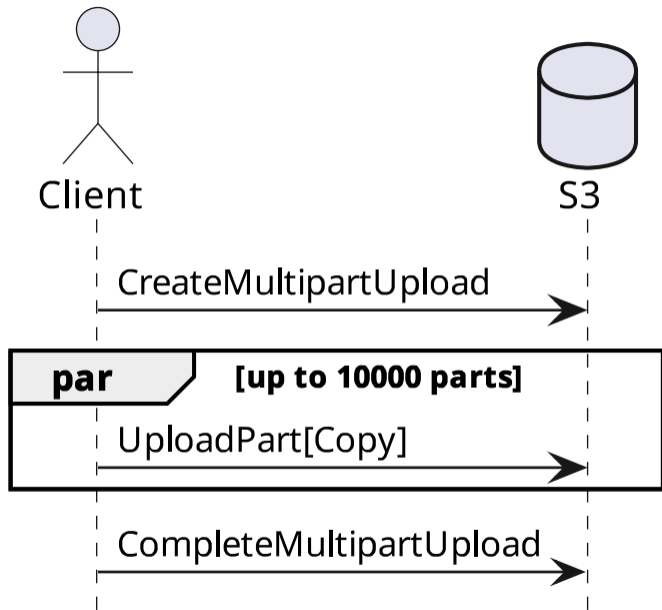
- 1 photos/dog.jpg
- 2 photos/logo.jpg

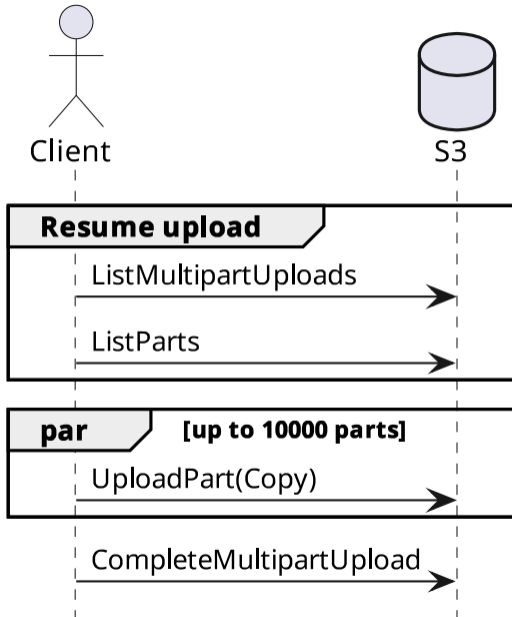


S3 Multipart API

- `CreateMultipartUpload`
- `UploadPart` with MD5
- Server-side `UploadPartCopy` from object
- `ListMultipartUploads`
- `ListParts` with [MD5]
- `CompleteMultipartUpload` with [MD5] of parts
- `AbortMultipartUpload`







Bucket Lifecycle Policies³

- Set of bucket lifecycle rules
- Filter objects by key **prefix** and/or **tags**
- Expiration after date or n days
- NonCurrentVersionExpiration after n days
- ExpiredObjectDeleteMarker
- AbortIncompleteMultipartUpload

³<https://docs.aws.amazon.com/AmazonS3/latest/dev/intro-lifecycle-rules.html>



Requirements

- **Scalability**
 - Capacity from TBs to PBs and beyond
 - Throughput from Gbps to Tbps
 - Locality (DC aware)
- **Fault-tolerance**
 - Lose any number of servers per DC
 - Disaster tolerant (offline DC)
 - No dedicated backups
- **Maintainability**
 - Automatic healing
 - Automated upscale/downscale



General components of S3

- **Metadata/object** storage (GBs/TBs)
 - File system (implicit)
 - In-memory
 - SQL/NoSQL/custom storage
- **Data/content** storage (TBs/PBs)
 - File system
 - Combined with metadata storage
 - Separate distributed blob storage



On-premises solutions

- **Ceph S3** (C++, Python)
 - USS Enterprise
 - **Complicated maintainability/scalability**
- **MinIO** (Go)
 - Filesystem + federation
 - **No huge installations described**
- **SeaweedFS** (Go)
 - Single developer
 - **No huge installations described**
- Zenko Cloudserver (JS), Riak Cloud Storage (Erlang), Joyent Triton Object Storage (JS), LeoFS (Erlang), ...



On-premises solutions

- **Ceph S3** (C++, Python)
 - USS Enterprise
 - Complicated maintainability/scalability
- **MinIO** (Go)
 - Filesystem + federation
 - No huge installations described
- **SeaweedFS** (Go)
 - Single developer
 - No huge installations described
- Zenko Cloudserver (JS), Riak Cloud Storage (Erlang), Joyent Triton Object Storage (JS), LeoFS (Erlang), ...



Data @ OK

- one-blob/cold-storage for **binary**
- NewSQL for **OLTP/meta**
- one-cloud as **runtime**



Key-value one-blob/cold-storage⁴

Hot one-blob-storage (OBS):

- 3x replication, tolerates loss of **2 disks**
- **> 100 PB**

one-cold-storage:

- Collects and **imports 40 GB segments** from OBS
- **2.1x** replication, tolerates loss of **5 disks**
- **≈ EB** and growing

⁴Alexander Khristoforov. Petabytes of video and photo storage in Odnoklassniki: make it cheaper, simpler and more reliable @ Joker 2017 (RU)



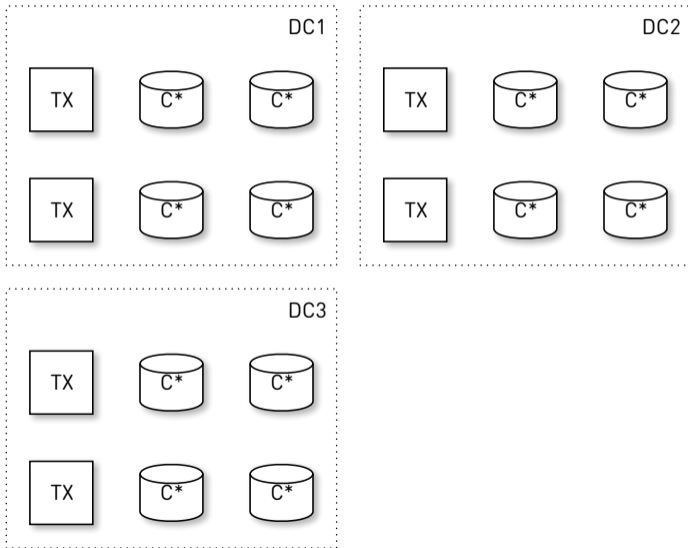
NewSQL⁵

- \approx **100** distributed clusters under heavy load
- Built upon **Apache Cassandra**
- Dedicated **transaction coordinators**
- **Partitioned** transactions
- Cassandra **fat client** pattern
- **Speculative** execution
- $p99 < \mathbf{3\ ms}$

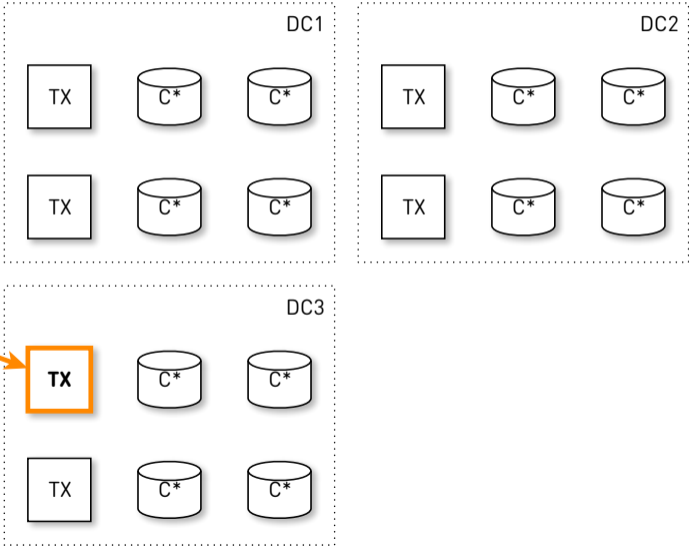
⁵Oleg Anastasev. NewSQL = NoSQL + ACID (RU)



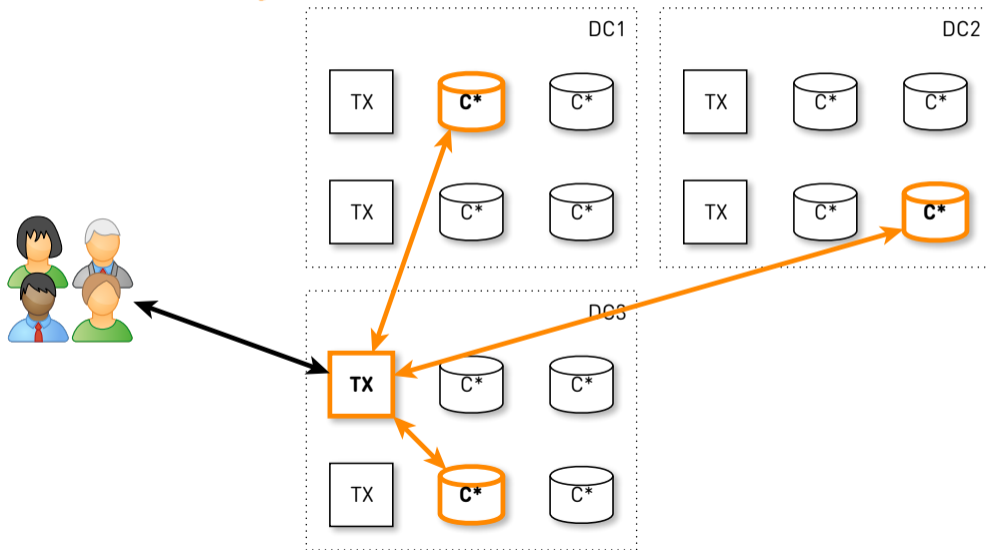
NewSQL cluster



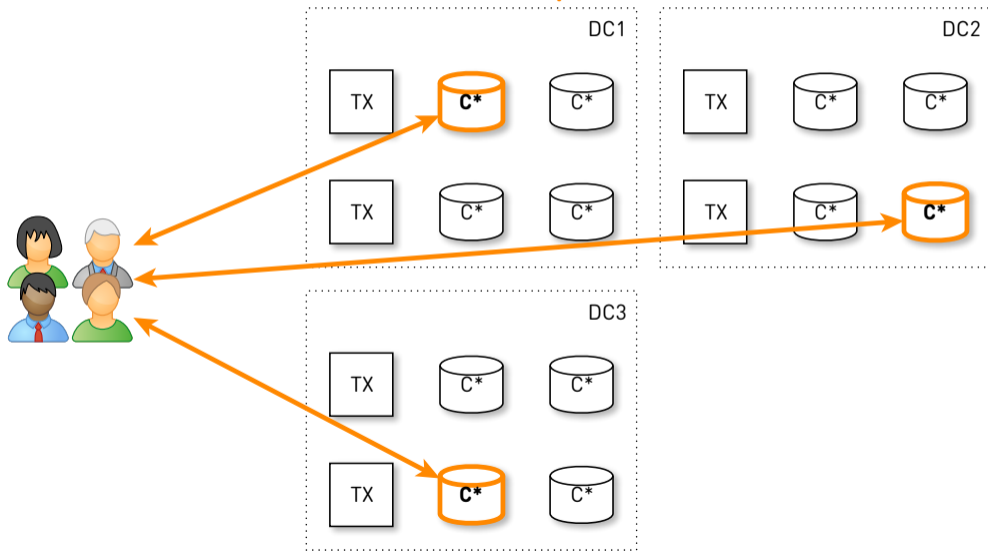
TX coordinator by partition



Read/modify/write



Non-transactional requests



C* modeling recap

```
1 CREATE TABLE example (  
2     p text,  
3     t text,  
4     i int,  
5     d blob,  
6     PRIMARY KEY ((p), t, i));
```



Partition key

```
1 CREATE TABLE example (  
2     p text,  
3     t text,  
4     i int,  
5     d blob,  
6     PRIMARY KEY ((p), t, i));
```



Clustering key

```
1 CREATE TABLE example (  
2     p text,  
3     t text,  
4     i int,  
5     d blob,  
6     PRIMARY KEY ((p), t, i));
```



Example

p	t	i	d
p2	t1	1	0xDA7A1
p2	t1	2	0xDA7A2
p2	t2	3	0xDA7A3
p1	t3	4	0xDA7A4
p1	t4	5	0xDA7A5



Partitions

p	t	i	d
p2	t1	1	0xDA7A1
	t1	2	0xDA7A2
	t2	3	0xDA7A3
p1	t3	4	0xDA7A4
	t4	5	0xDA7A5

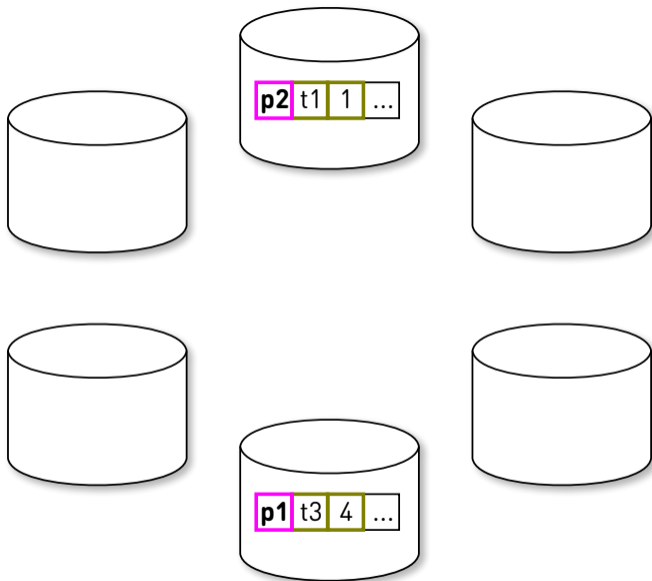


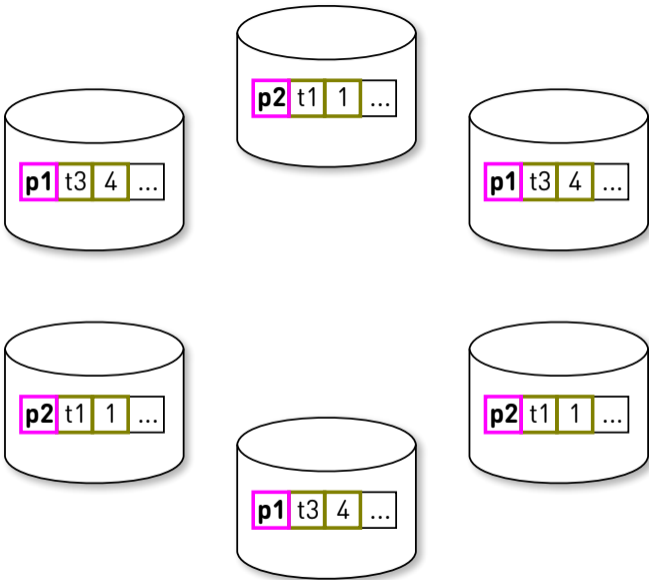
Physical data layout

p2	t1	1	0xDA7A1	t1	2	0xDA7A2	t2	3	0xDA7A3
----	----	---	---------	----	---	---------	----	---	---------

p1	t3	4	0xDA7A4	t4	5	0xDA7A5
----	----	---	---------	----	---	---------







Data @ OK

- one-blob/cold-storage for **binary**
- NewSQL for **OLTP/meta**
- one-cloud as **runtime**



one-cloud⁷

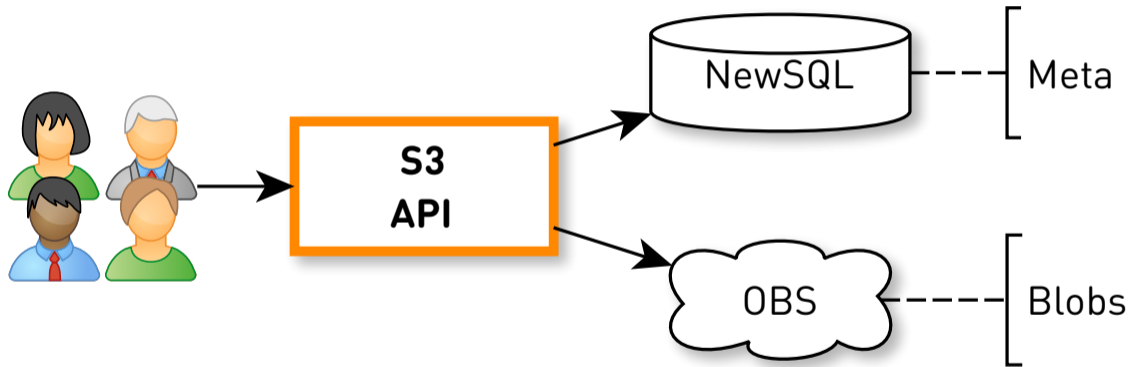
- Thread priorities (JIT, compaction, etc.)
- Storage **availability awareness**
 - Safe deployments
 - Rolling auto update/rebase
 - Auto migration
- **Automated cloud operations**⁶
 - Cluster upscale/downscale
 - Node replacement
 - Resource defragmentation

⁶Leonid Talalaev. "Rise of the Machines" is OK @ Highload++ 2019 (RU)

⁷Oleg Anastasev. Datacenter-level operating system by OK (RU)



Architecture



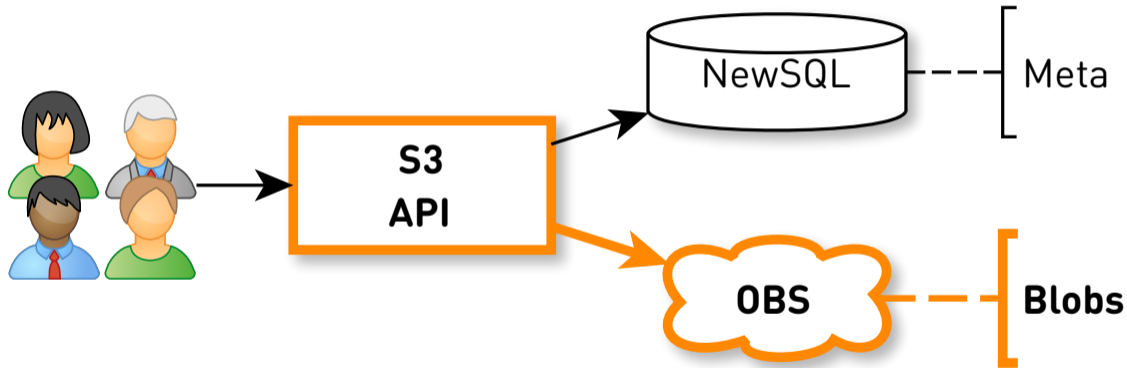
Features

- **Deduplication**
 - Transparent server-side
 - Free object copy + space economy
 - Object/upload **part** = **block**
 - **Reference tracking** by object/upload parts
 - Blocks do not move
- **Versioning**⁸
 - Rollback/restore
 - No need for backups
 - `NoncurrentVersionExpiration` lifecycle rule

⁸<https://docs.aws.amazon.com/AmazonS3/latest/userguide/versioning-workflows.html>



Blocks



Block ID

Block size 8 bytes	SHA-256 32 bytes
-----------------------	---------------------



Queries

- **Reference** block
- **Unreference** block
- Check the block is **referenced**

```
1 CREATE TABLE block_references (  
2     block blob,  
3     bucket text,  
4     name text, -- Object/upload key i.e. a/b/c  
5     version timeuuid, -- Object/upload version  
6     part int, -- Object/upload part  
7     PRIMARY KEY ((block), bucket, name, version, part));
```



```
1 CREATE TABLE block_references (  
2     block blob,  
3     bucket text,  
4     name text, -- Object/upload key i.e. a/b/c  
5     version timeuuid, -- Object/upload version  
6     part int, -- Object/upload part  
7     PRIMARY KEY ((block), bucket, name, version, part));
```

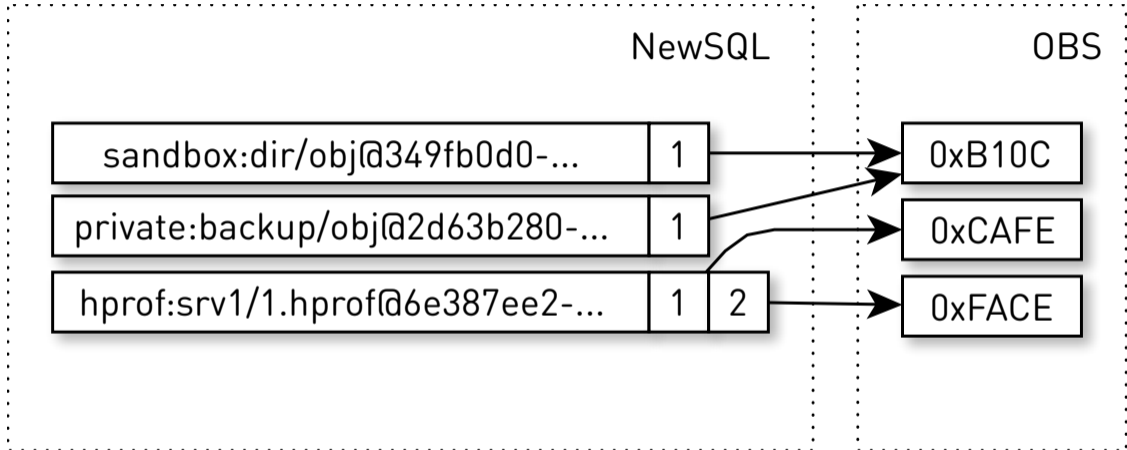


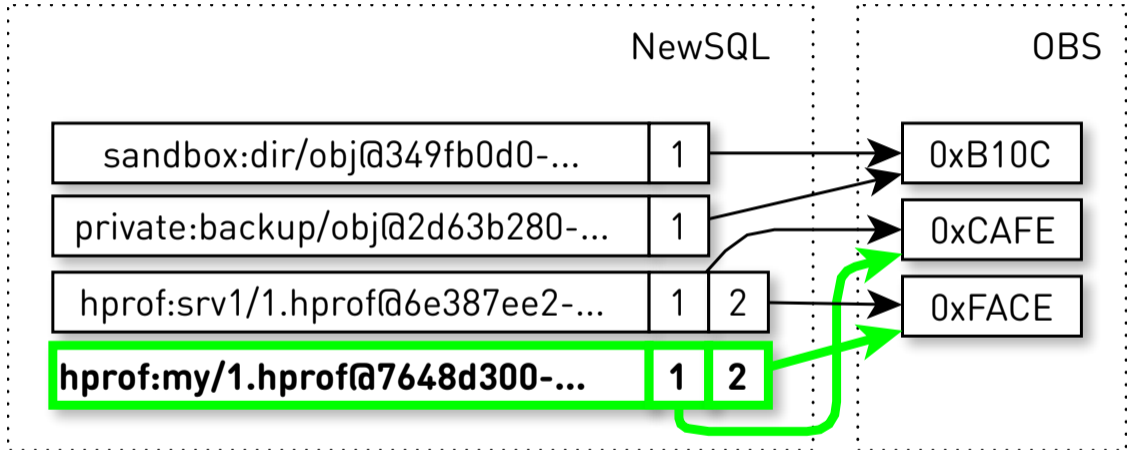
```
1 CREATE TABLE block_references (  
2     block blob,  
3     bucket text,  
4     name text, -- Object/upload key i.e. a/b/c  
5     version timeuuid, -- Object/upload version  
6     part int, -- Object/upload part  
7     PRIMARY KEY ((block), bucket, name, version, part));
```

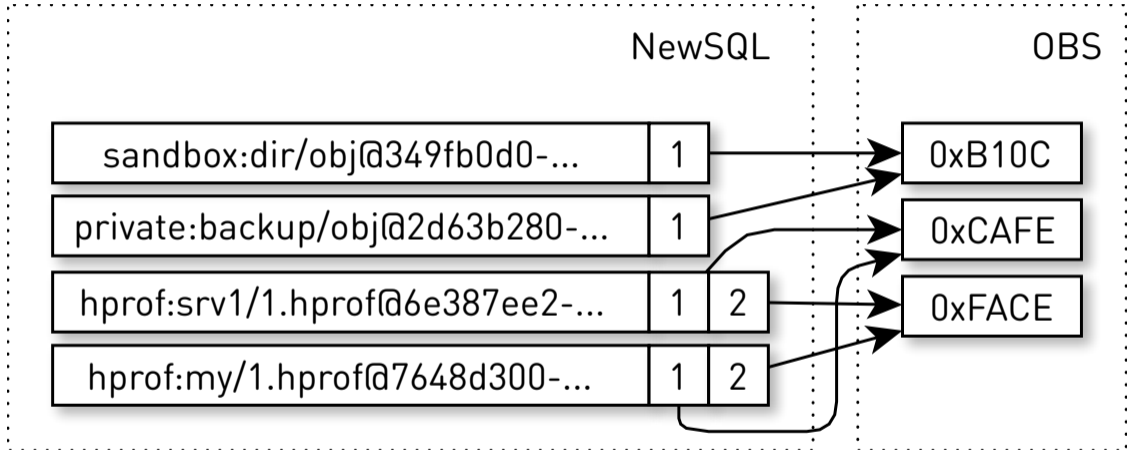


block	bucket	name	version	part
0xB10C	sandbox	dir/obj	349fb0d0-...	1
0xB10C	private	backup/obj	2d63b280-...	1
0xCAFE	hprof	srv1/1.hprof	6e387ee2-...	1
0xFACE	hprof	srv1/1.hprof	6e387ee2-...	2



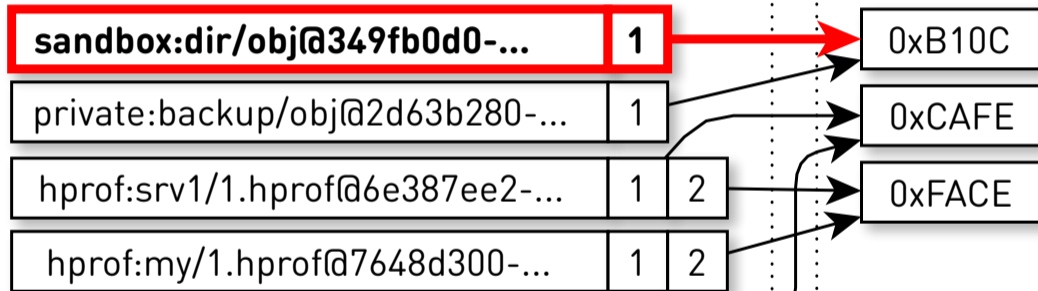


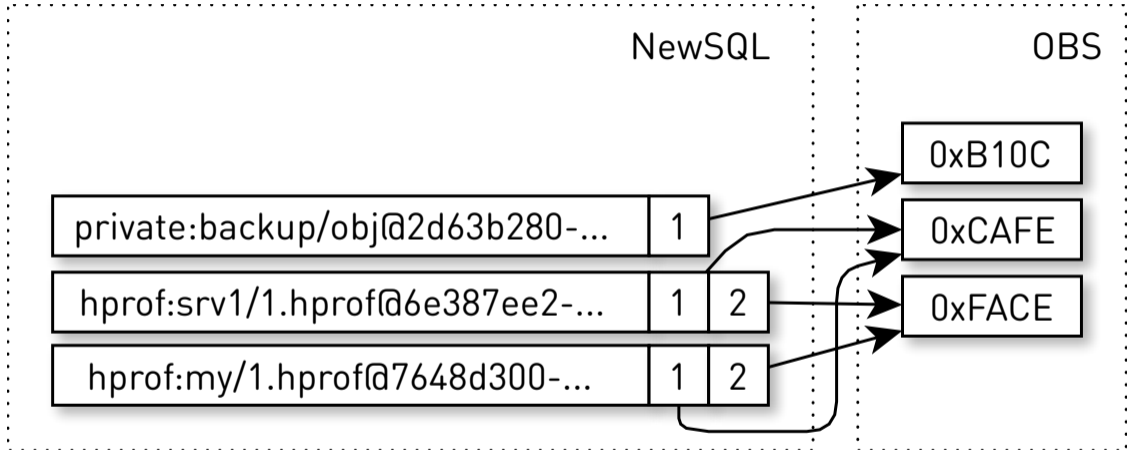




NewSQL

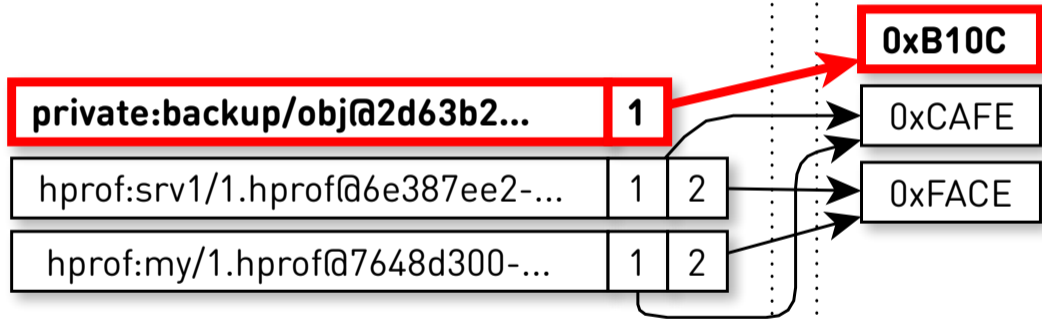
OBS





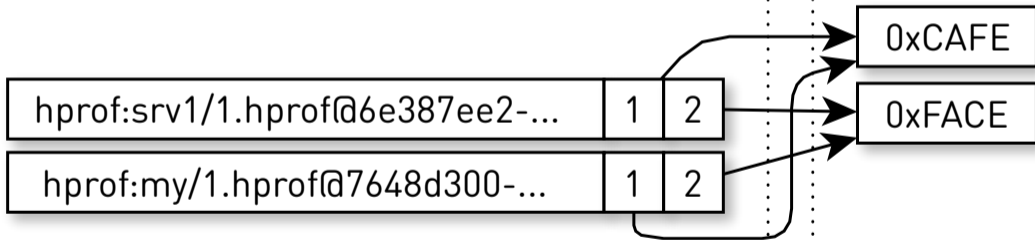
NewSQL

OBS

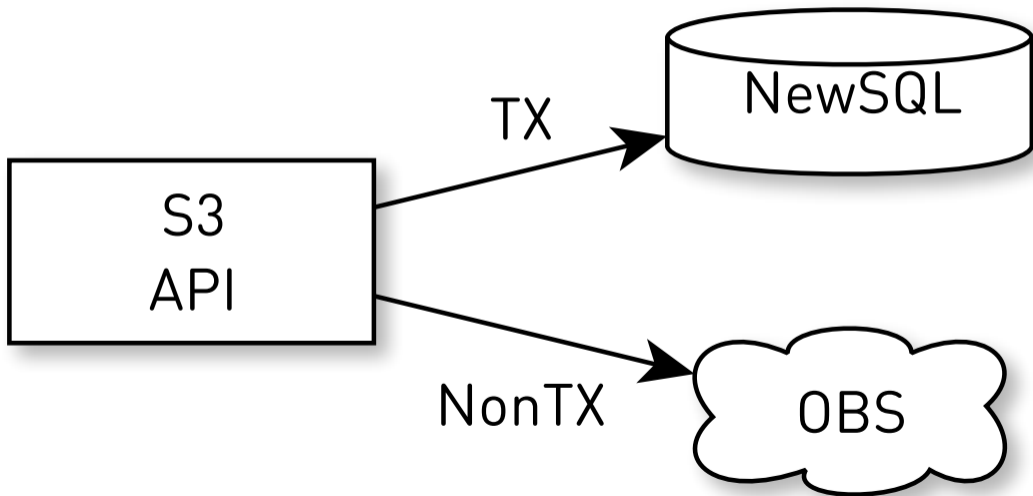


NewSQL

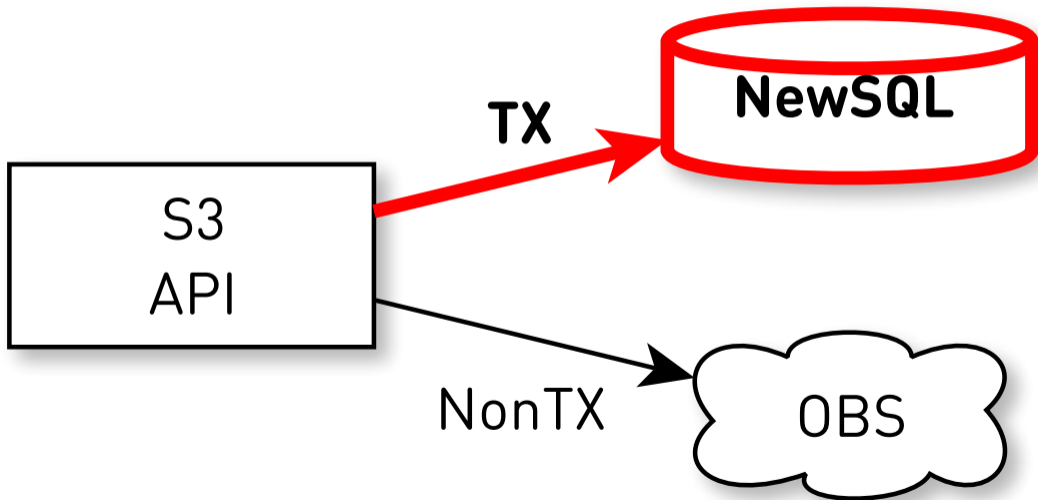
OBS



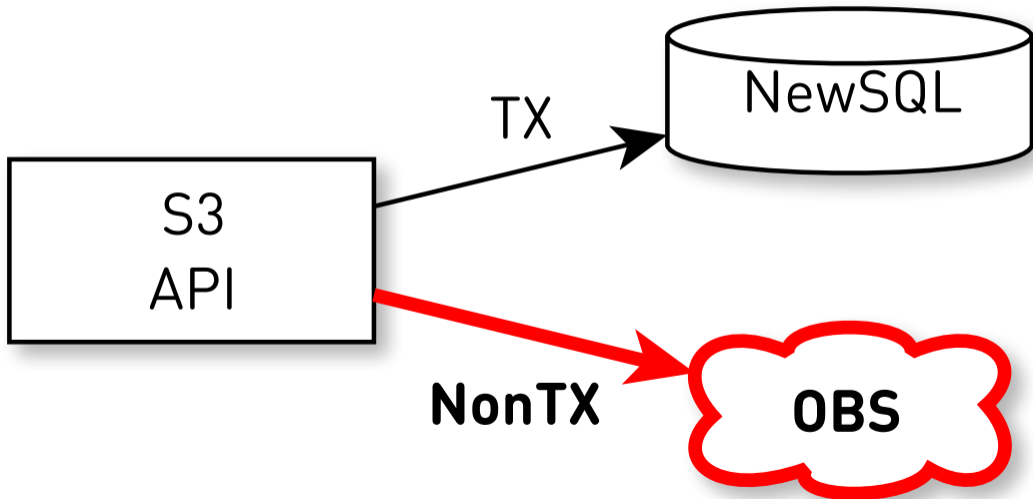
Problem

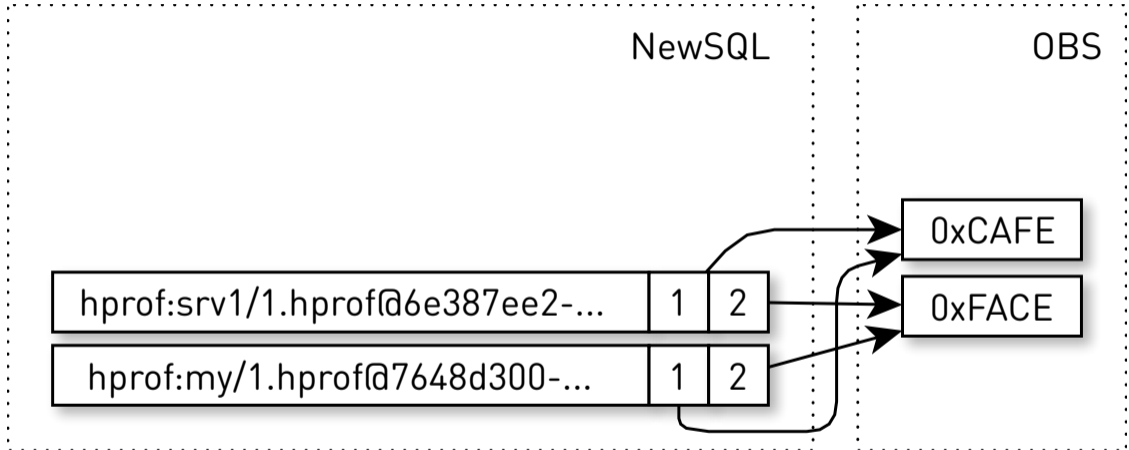


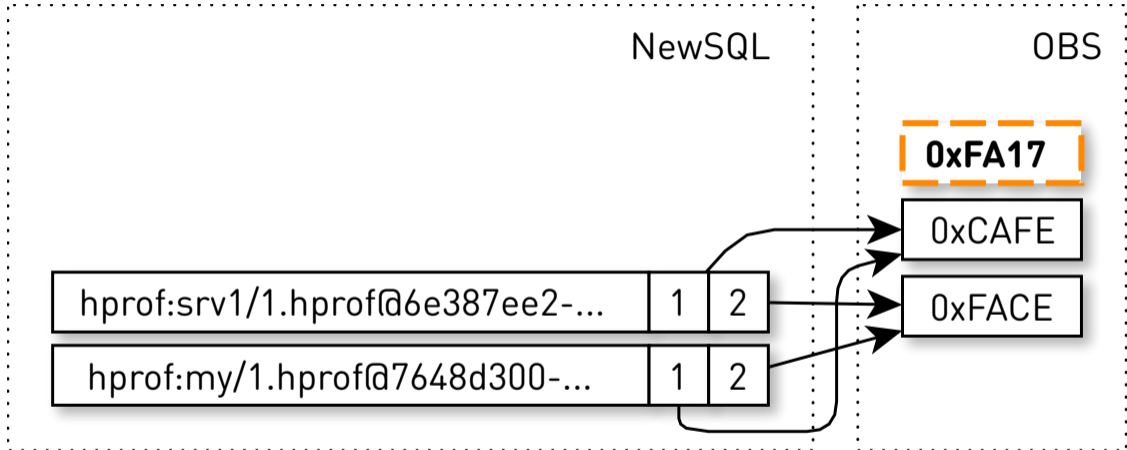
Problem



Problem

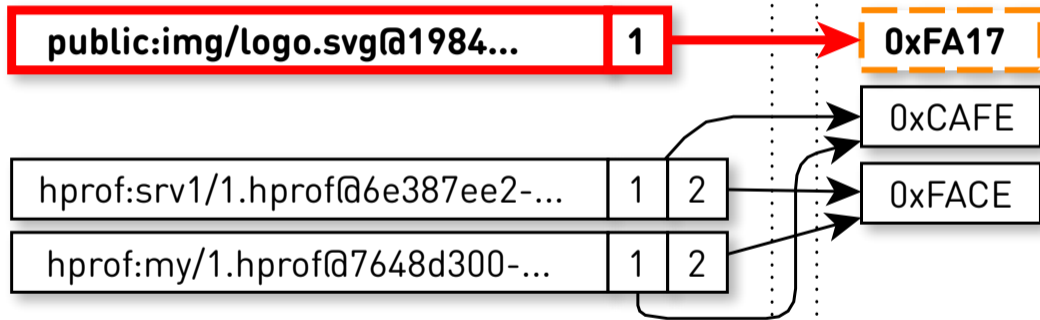






NewSQL

OBS



NewSQL

OBS

public:img/logo.svg@1984... 1

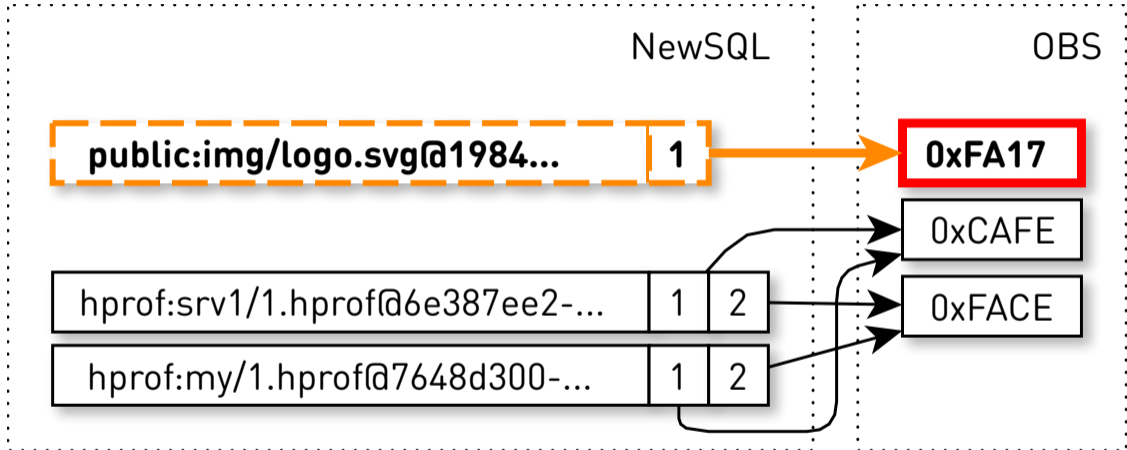
hprof:srv1/1.hprof@6e387ee2-... 1 2

hprof:my/1.hprof@7648d300-... 1 2

0xCAFE

0xFACE





FSM to the rescue

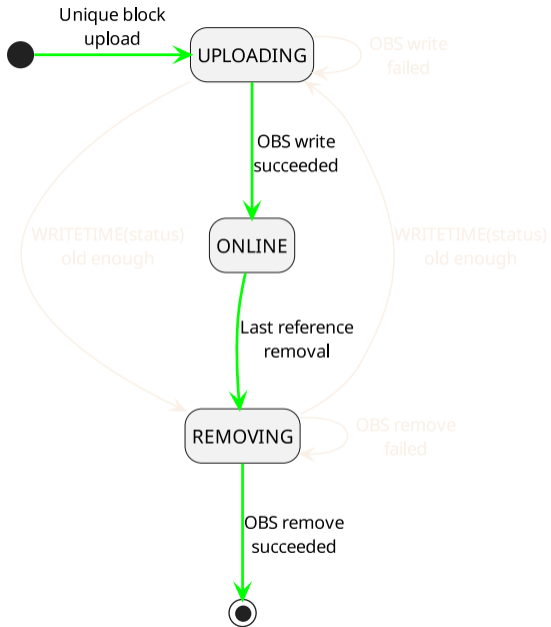
- Block **status tracking**
- Support for **retry/resume**
- Detect **race conditions**

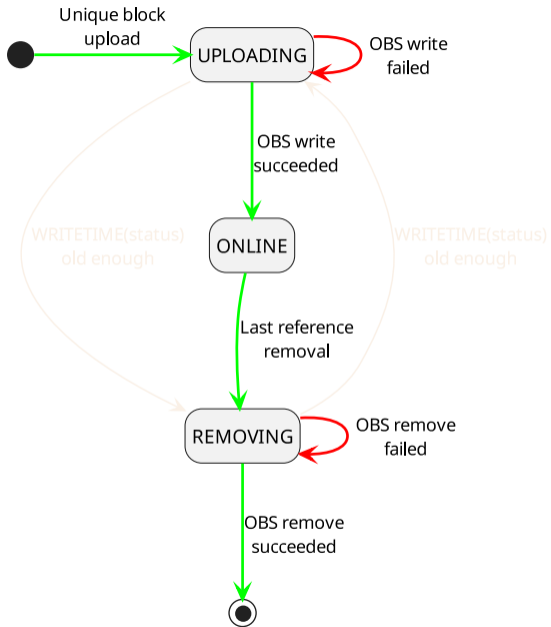


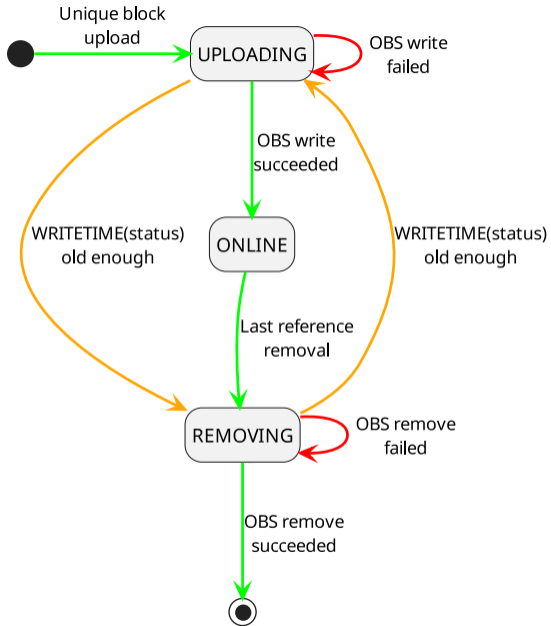
Block Status

```
1 CREATE TABLE block_status (  
2     block blob, -- Block ID  
3     status int, -- UPLOADING/ONLINE/REMOVING  
4     md5 blob, -- Block MD5  
5     PRIMARY KEY ((block)));
```

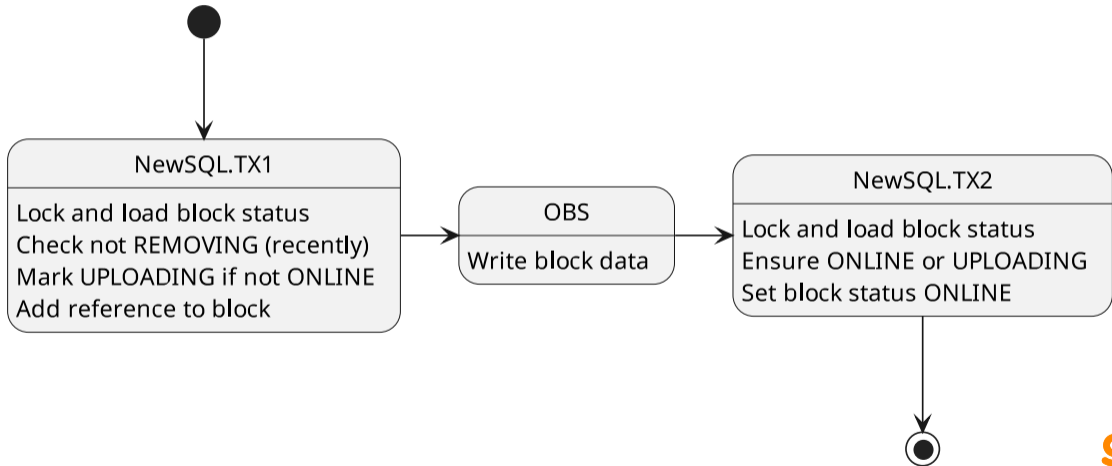




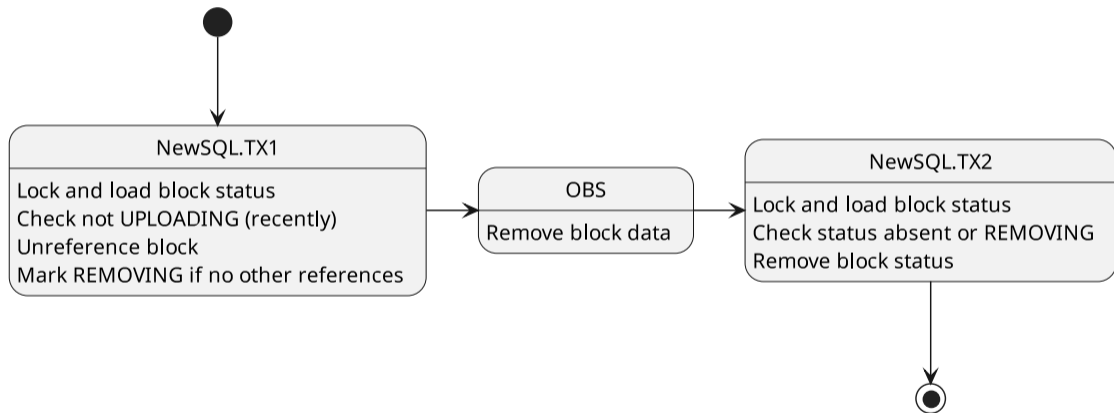




Unique block upload



Last reference removal



Once upon a time...

Timeouts on checking the block is still referenced:

- 1 Timeout executing [SELECT * FROM block_references WHERE block=? **LIMIT 1**], bound: [...]:
- 2 **org.apache.cassandra.exceptions.ReadTimeoutException:**
- 3 Operation timed out – received only 1 responses of required 2.
- 4 at one.cassandra.client.impl.EmbeddedNewSQLClientImpl.execute(
5 EmbeddedNewSQLClientImpl.java:496)
- 6 at one.cassandra.client.impl.EmbeddedCassandraClientImpl.paginate(
7 EmbeddedCassandraClientImpl.java:288)
- 8 ...



Single block references occupy 7 GB?!

```
1 $ nodetool ... cfstats ... block_references
```

```
2 ...
```

```
3 Compacted partition minimum bytes: 125
```

```
4 Compacted partition maximum bytes: 7'152'383'774
```

```
5 Compacted partition mean bytes: 721
```



Highly referenced block



Tombstone accumulation



Timeouts on scanning for live reference



Wide C* partitions

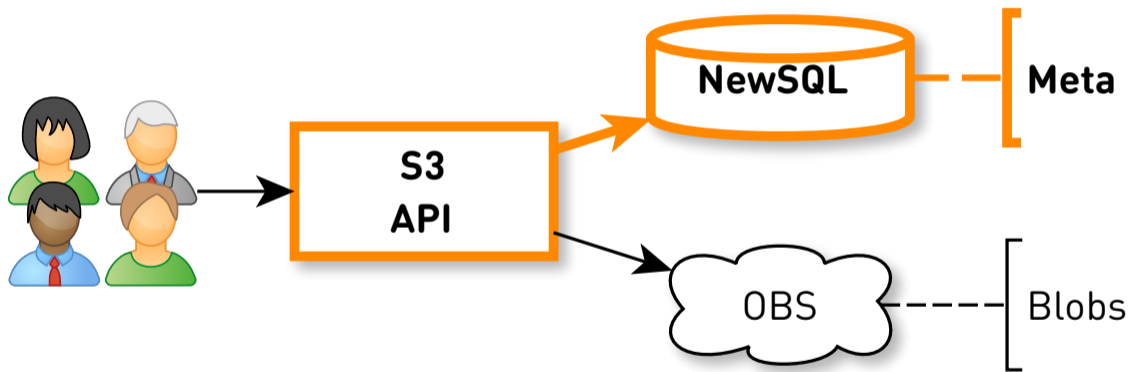
- Timeouts on `tombstone_failure_threshold`
 - **100K tombstones** by default
- Tombstones are purged after `gc_grace_period`
 - **10 days** by default
- Read repair operates **full partitions**
- `in_memory_compaction_limit`
 - **64 MB** by default



```
1 /**
2 * Cache of small eternal blocks to solve numerous problems:
3 * <ul>
4 *     <li>Wide partitions due to highly referenced blocks</li>
5 *     <li>Huge number of tombstones in wide partitions</li>
6 *     <li>Serving tiny blocks from blob storage vs serving from
7 *     memory</li>
8 * </ul>
9 */
10 public final class Primordials {
11     // The most popular empty block
12     addPrimordial(cache, new byte[0]);
13
14     // Hi, Apache Flink!
15     addPrimordial(cache, "10".getBytes(StandardCharsets.US_ASCII));
```



Object metadata



Functional requirements

- **S3 reference** for version/object/upload ops
 - Versioned and non-versioned
- **Transactional** concurrent modifications
- **Efficient** List-methods:
 - **Filtering** by prefix
 - **Grouping** by delimiter
 - Or **recursive**
 - **Sorted** by name/version/upload
 - **Paging** marker



Bucket = single partition + lock



- Might work
- Up to **100K objects per bucket**⁹

⁹See wide C* partition

Nonfunctional requirements

- **Data partitioning** for scalability
- **Transaction striping** for scalability
- **Efficient** upload publishing
 - No extra data copying
- **Fast** object/version serving
 - Handle extremely popular objects



Better way

- **Logical hierarchy** of "folders" by /
- **Partition** objects by parent folder
- **Cluster** inside parent by name/version
- Maintain **folder hierarchy** index
- Maintain **last object version** index
- **Atomicity** by (parent, name) key

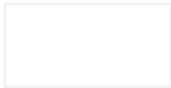
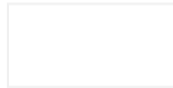
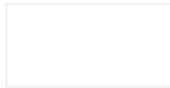
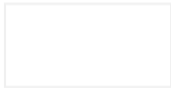
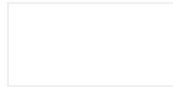
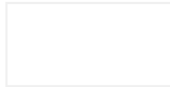
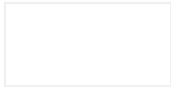


Object partitioning

parent	name	version	meta
/	README.md	1	...
	TODO	2	...
data/	1.dat	4	...
	1.dat	3	...
	2.dat	5	...
data/log/	20220626.log	6	...



locks



Transient object/upload locks

```
1 CREATE TABLE locks (  
2     parent text, -- Full parent name i.e. "a/b/"  
3     name text, -- Base child name i.e. "c"  
4     utoken int, -- xxhash32(parent + name)  
5     PRIMARY KEY ((parent), name, utoken),  
6     UPDATE TOKEN utoken);
```



Serializes (parent, name) ops

```
1 CREATE TABLE locks (  
2     parent text, -- Full parent name i.e. "a/b/"  
3     name text, -- Base child name i.e. "c"  
4     utoken int, -- xxhash32(parent + name)  
5     PRIMARY KEY ((parent), name, utoken),  
6     UPDATE TOKEN utoken);
```

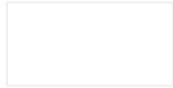
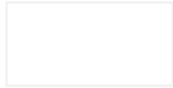
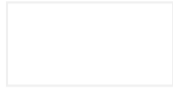


Transaction striping

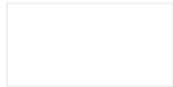
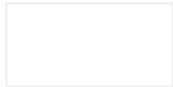
```
1 CREATE TABLE locks (  
2     parent text, -- Full parent name i.e. "a/b/"  
3     name text, -- Base child name i.e. "c"  
4     utoken int, -- xxhash32(parent + name)  
5     PRIMARY KEY ((parent), name, utoken),  
6     UPDATE TOKEN utoken);
```



locks



uploads



contain

**upload
parts**



parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

parent	name	upload	part	status	block
a/	report.doc	234-...	1	UPLOADING	0xBADDCAFE
a/	report.doc	234-...	2	ONLINE	0xCAFEFEED
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	UPLOADING	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE



parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

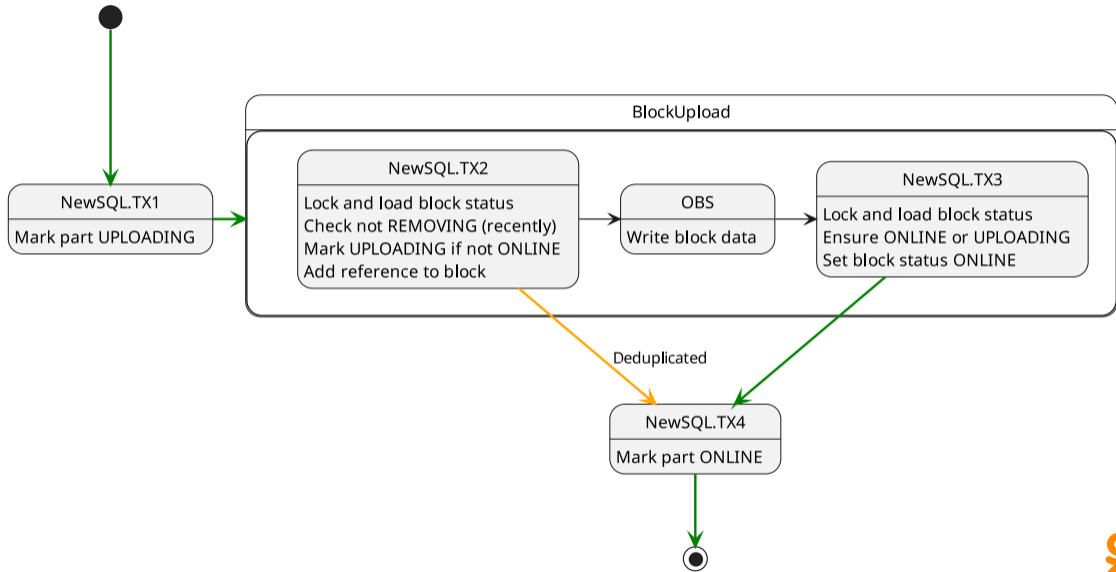
parent	name	upload	part	status	block
a/	report.doc	234-...	1	UPLOADING	0xBADDCAFE
a/	report.doc	234-...	2	ONLINE	0xCAFEFEED
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	UPLOADING	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE



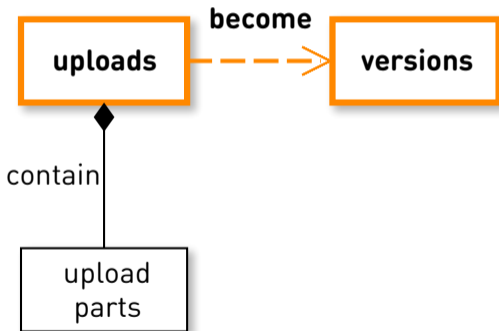
parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

parent	name	upload	part	status	block
a/	report.doc	234-...	1	UPLOADING	0xBADDCAFE
a/	report.doc	234-...	2	ONLINE	0xCAFEFEED
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	UPLOADING	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE





locks



parent	name	upload	meta
a/	report.doc	123-...	type=doc
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

parent	name	version	meta	blocks
---------------	-------------	----------------	-------------	---------------



parent	name	upload	meta
a/	report.doc	234-...	type=doc
b/	track.flac	345-...	type=flac

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]



parent	name	upload	meta
b/	track.flac	345-...	type=flac

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]



parent	name	upload	meta
---------------	-------------	---------------	-------------

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]



parent	name	upload	meta
---------------	-------------	---------------	-------------

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]



parent	name	upload	meta
---------------	-------------	---------------	-------------

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]

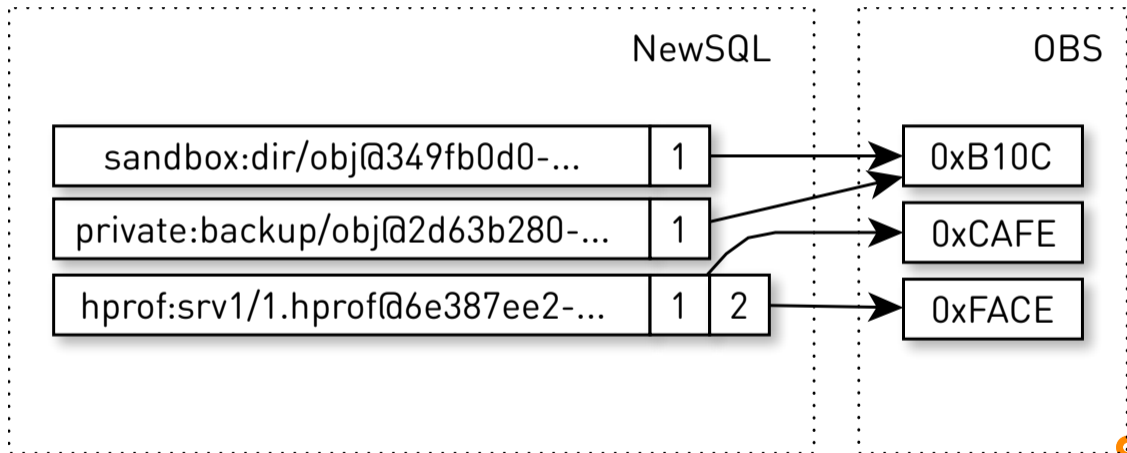


parent	name	upload	meta
--------	------	--------	------

parent	name	version	meta	blocks
a/	report.doc	123-...	type=doc	[0xFA...]
a/	report.doc	234-...	type=doc	[0xBA..., 0xCA...]
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]



Reusing block references



parent	name	upload	meta
b/	track.flac	345-...	type=flac

parent	name	upload	part	status	block
b/	track.flac	345-...	1	ONLINE	0xA5A5A5A5
b/	track.flac	345-...	2	ONLINE	0xABABABAB
b/	track.flac	345-...	3	ONLINE	0xBEBEBEBE

parent	name	version	meta	blocks

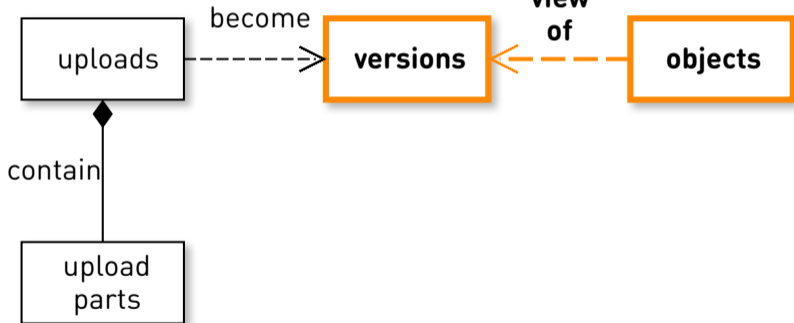
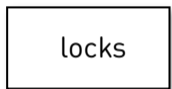


parent	name	upload	meta

parent	name	upload	part	status	block

parent	name	version	meta	blocks
b/	track.flac	345-...	type=flac	[0xA5..., 0xAB..., 0xBE...]





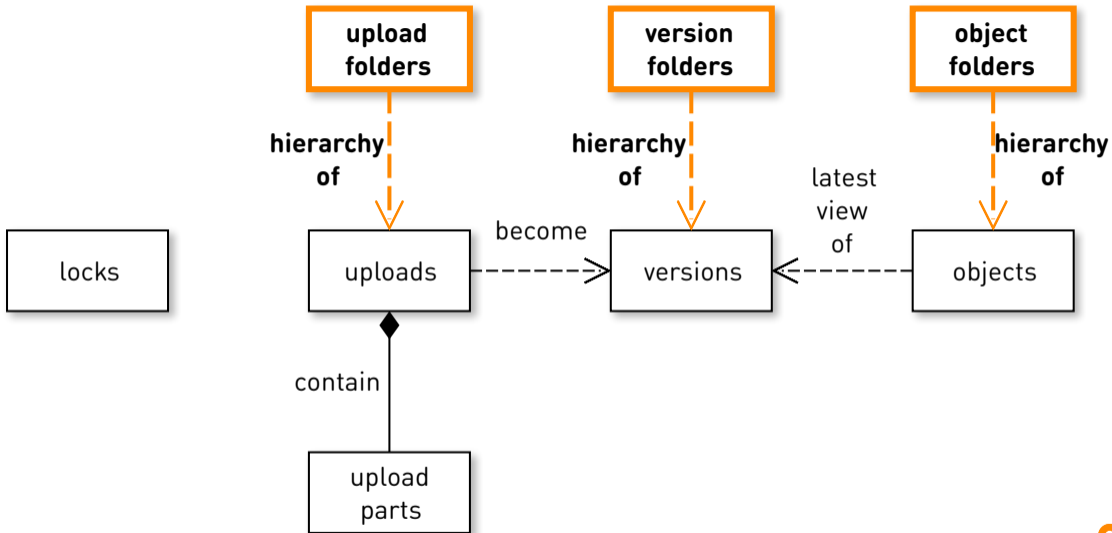
Versions:

parent	name	version	eTag	blocks
docs/	0.doc	234-...	CDCDCDCD	[0xA1]
docs/	1.doc	345-...	D00D2BAD	[0xB2]
docs/	1.doc	456-...	0D15EA5E	[0xC3]
docs/	2.doc	567-...	FEE1DEAD	[0xD4]
docs/	2.doc	678-...	NULL	NULL

Objects:

parent	name	version	eTag	blocks
docs/	0.doc	234-...	CDCDCDCD	[0xA1]
docs/	1.doc	456-...	0D15EA5E	[0xC3]





```
1 CREATE TABLE {object,version,upload}_folders (  
2     parent text, -- Full parent folder i.e. "a/b/"  
3     child text, -- Child folder name i.e. "c"  
4     ts timestamp, -- Last write timestamp  
5     PRIMARY KEY ((parent), child));
```



```
1 CREATE TABLE {object,version,upload}_folders (  
2     parent text, -- Full parent folder i.e. "a/b/"  
3     child text, -- Child folder name i.e. "c"  
4     ts timestamp, -- Last write timestamp  
5     PRIMARY KEY ((parent), child));
```



{object,version,upload}_folders

- $a/b/c/d \rightarrow (/ , a/), (a/ , b/), (a/b/ , c/)$
- **All** queries are **non-transactional**
- **Folder cleanup** with `WRITETIME(ts)` **slightly in the past**



List with delimiter a.k.a. ls

```
1 GET /?prefix=dir/  
2     &marker=dir/b  
3     &delimiter=/  
4     &max-keys=3
```

```
1 SELECT * FROM object_folders  
2     WHERE parent = 'dir/' AND child > 'b'  
3     LIMIT 4
```

```
1 SELECT * FROM objects  
2     WHERE parent = 'dir/' AND name > 'b'  
3     LIMIT 4
```



List with delimiter a.k.a. ls

```
1 GET /?prefix=dir/  
2     &marker=dir/b  
3     &delimiter=/  
4     &max-keys=3
```

```
1 SELECT * FROM object_folders  
2     WHERE parent = 'dir/' AND child > 'b'  
3     LIMIT 4
```

```
1 SELECT * FROM objects  
2     WHERE parent = 'dir/' AND name > 'b'  
3     LIMIT 4
```



List w/o delimiter a.k.a. `ls -R`

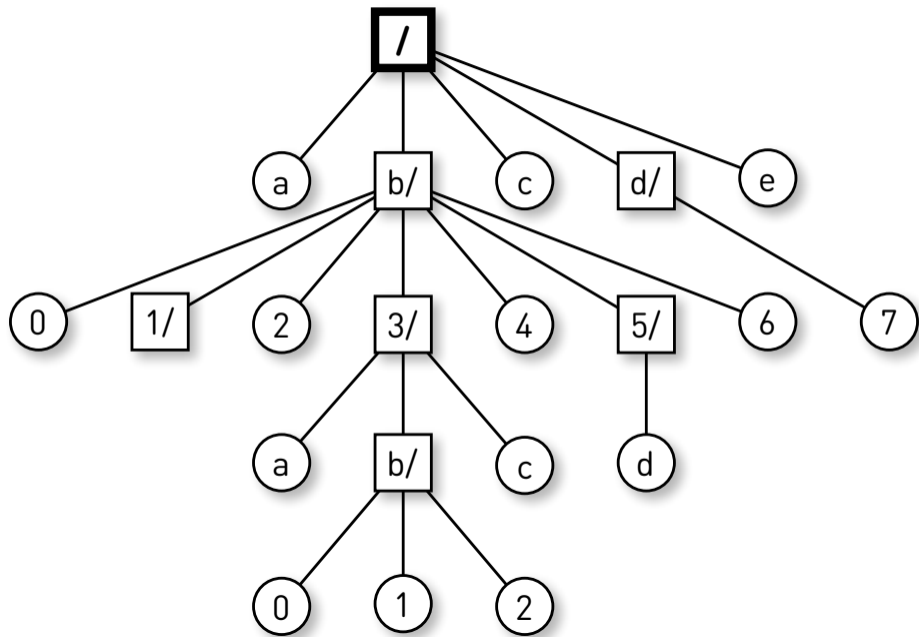
- 1 **Extract** parent from prefix
- 2 **Position** w.r.t. marker and suffix of prefix
- 3 In-order **tree traversal**
 - 1 Query sorted **subfolders**
 - 2 Query sorted **objects/versions/uploads**
 - 3 **Merge sorted**
 - 4 Dive **deeper**
- 4 Until `limit` objects/versions/uploads collected

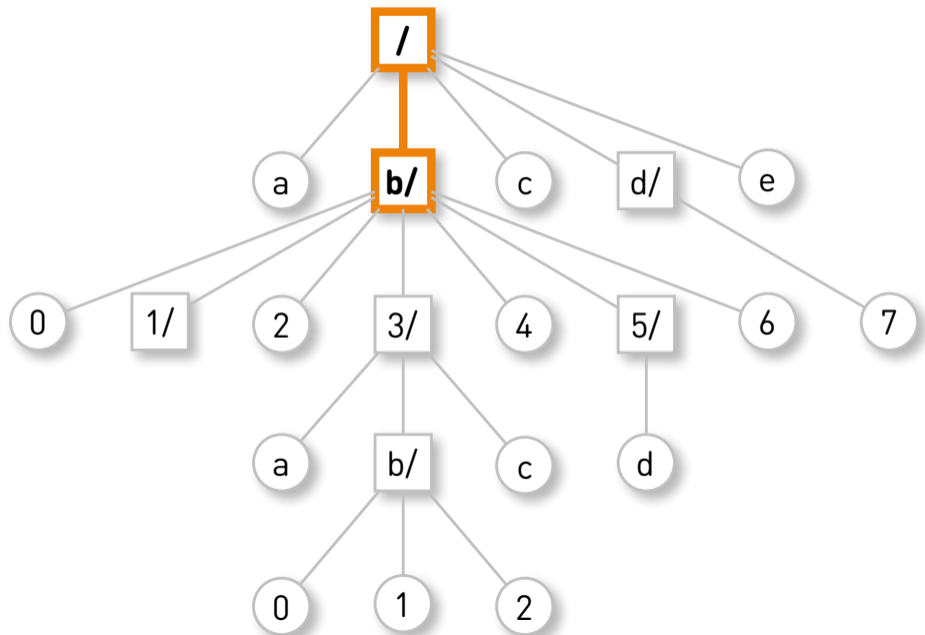


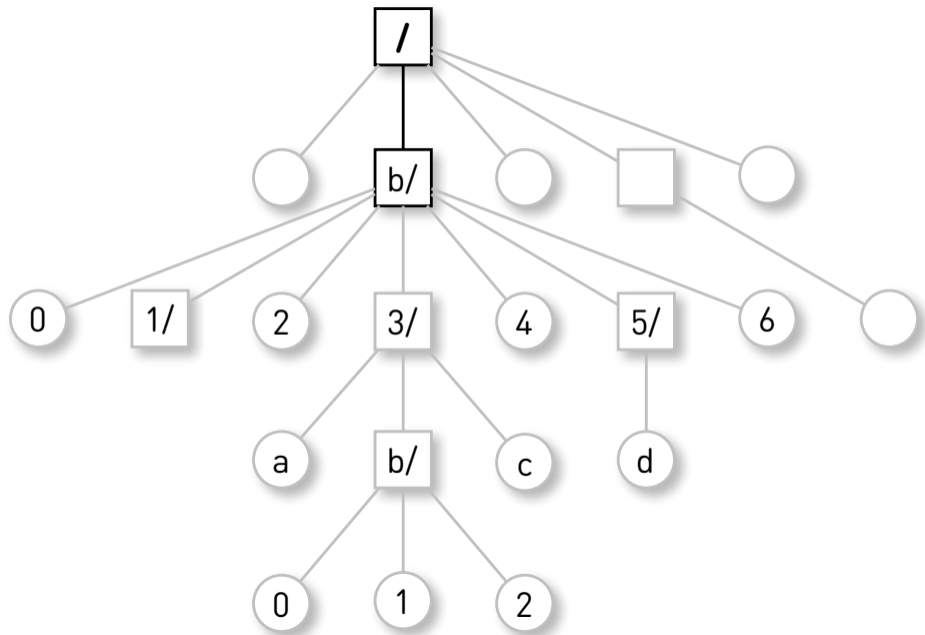
List w/o delimiter a.k.a. ls -R

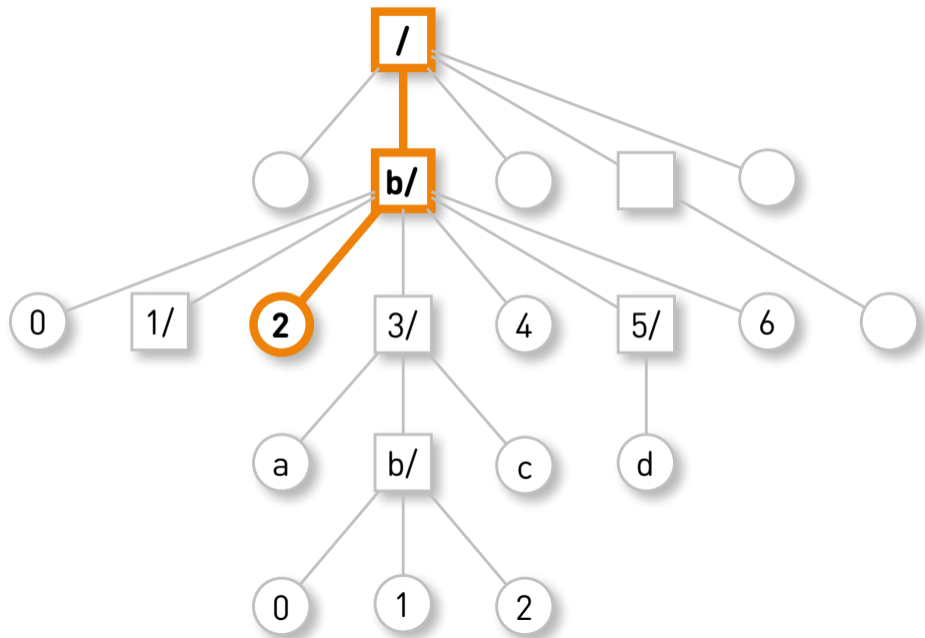
```
1 GET /?prefix=b/&marker=b/2 HTTP/1.1
```

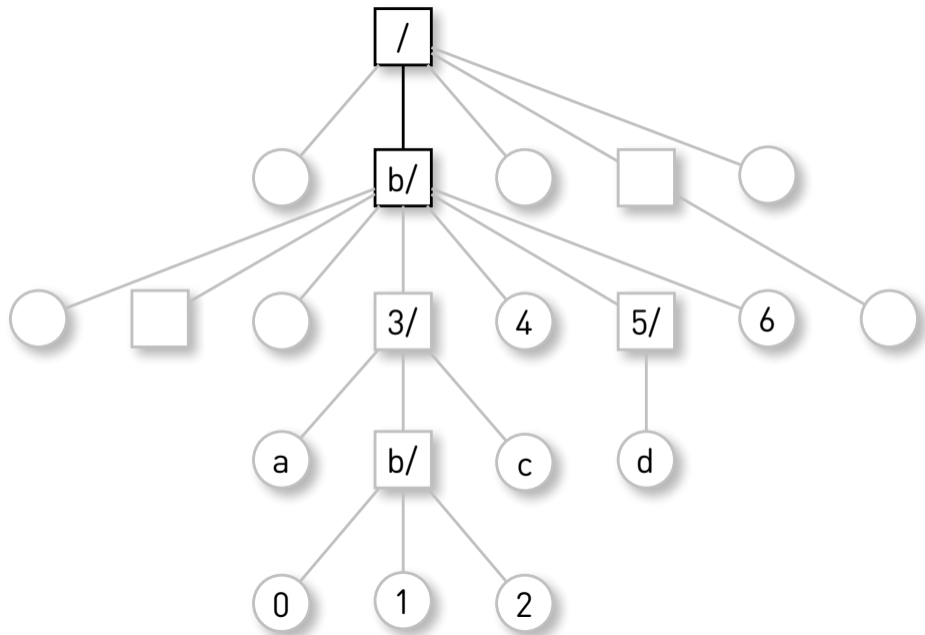


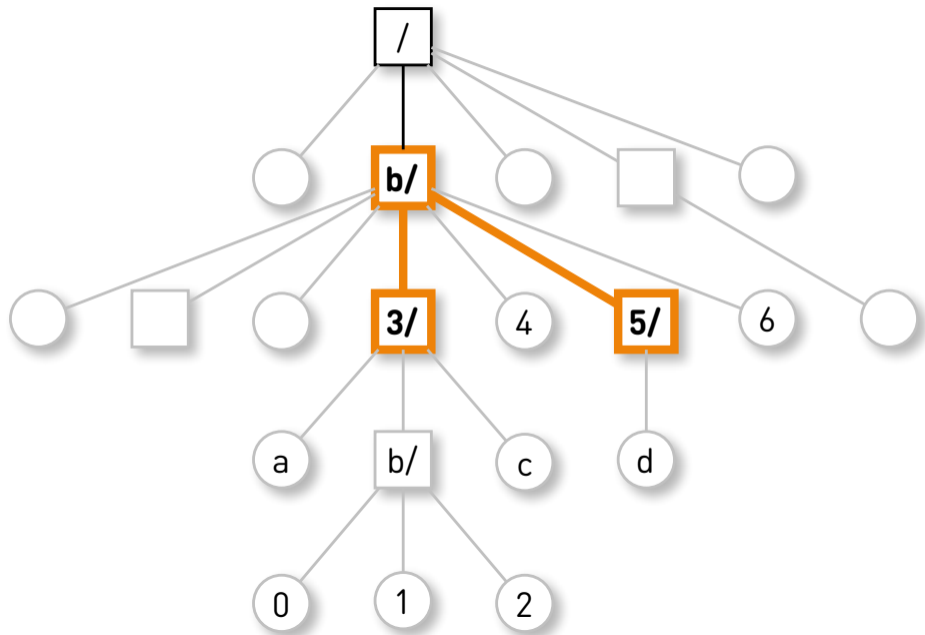


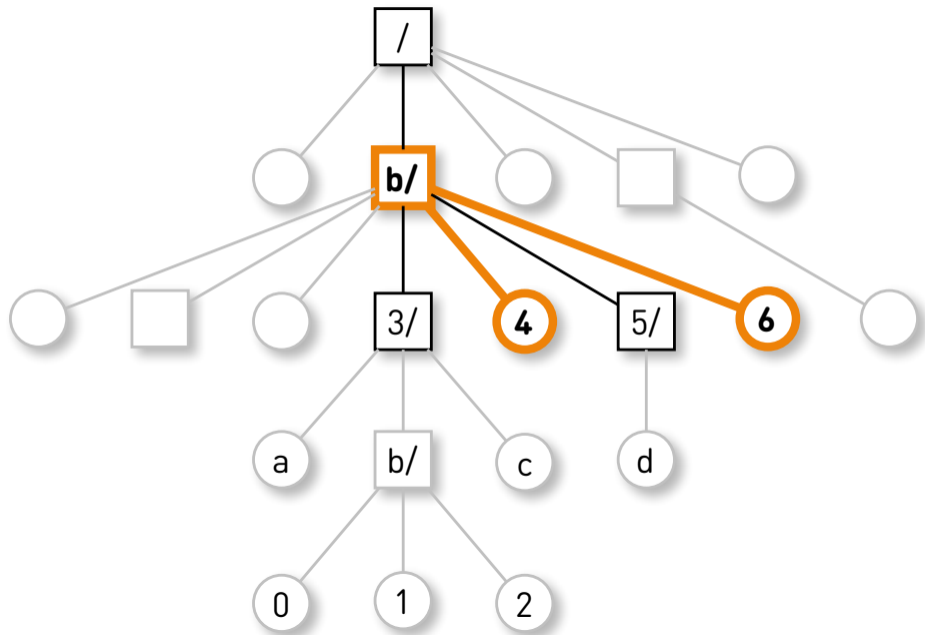


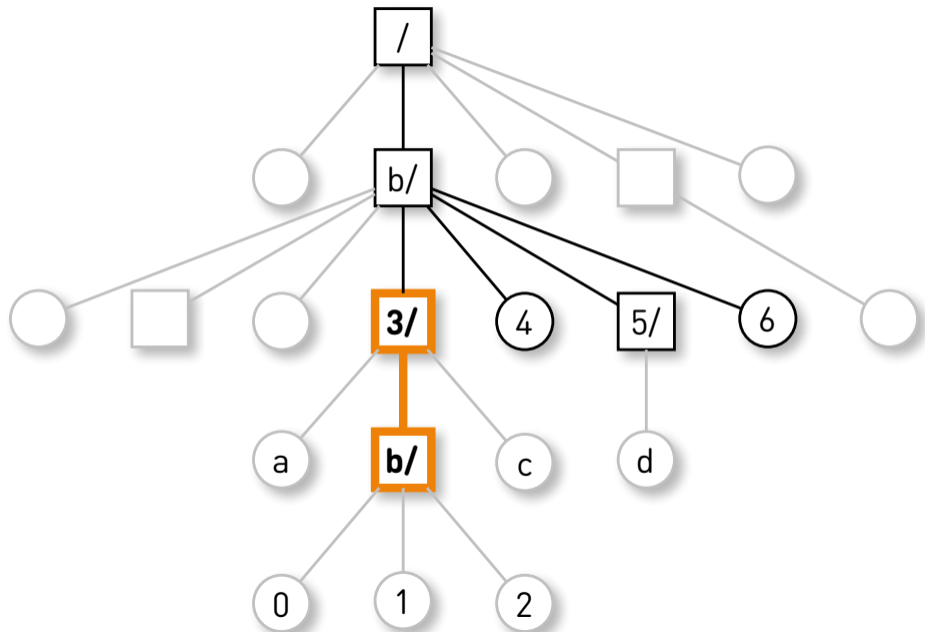


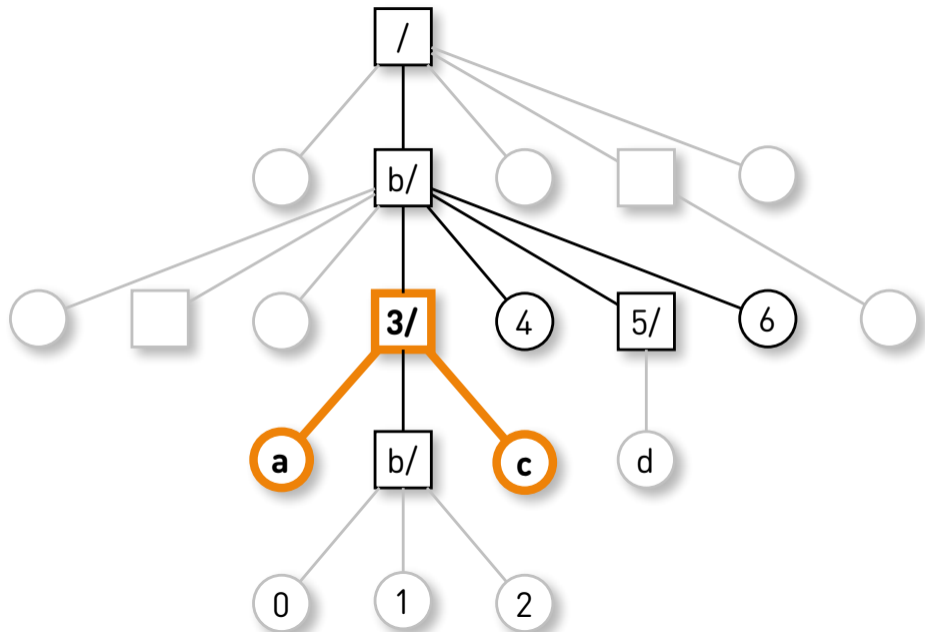


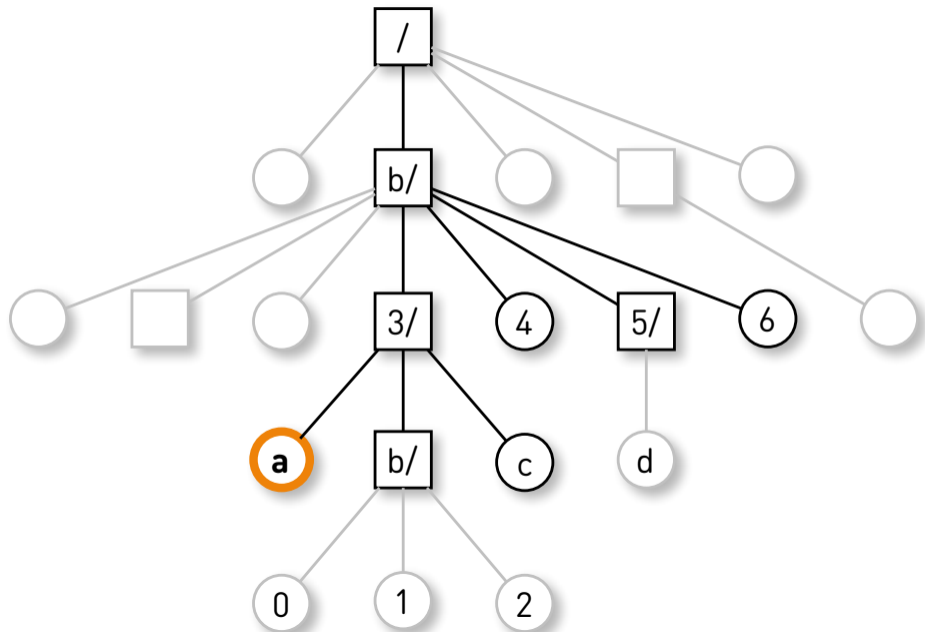


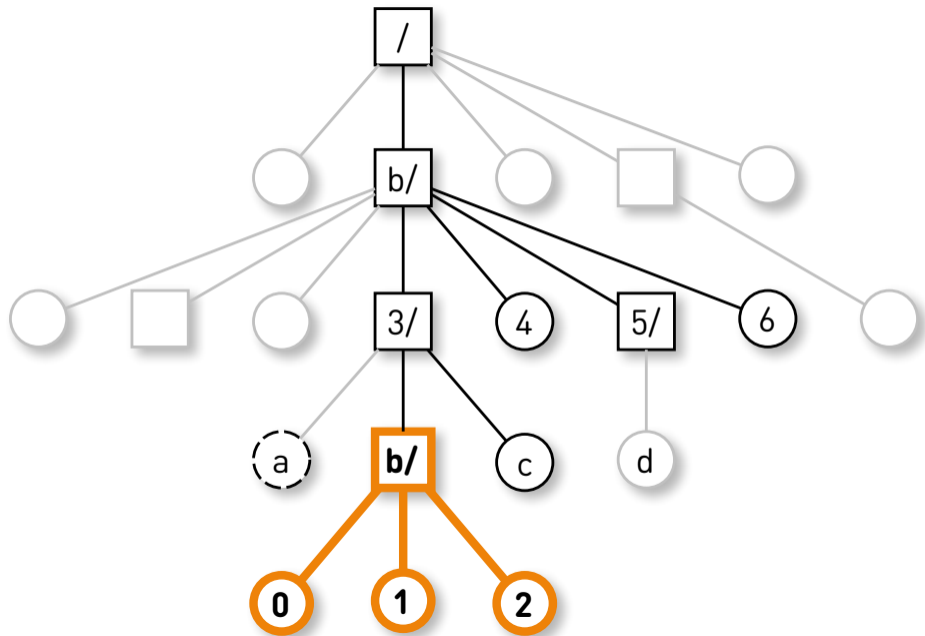


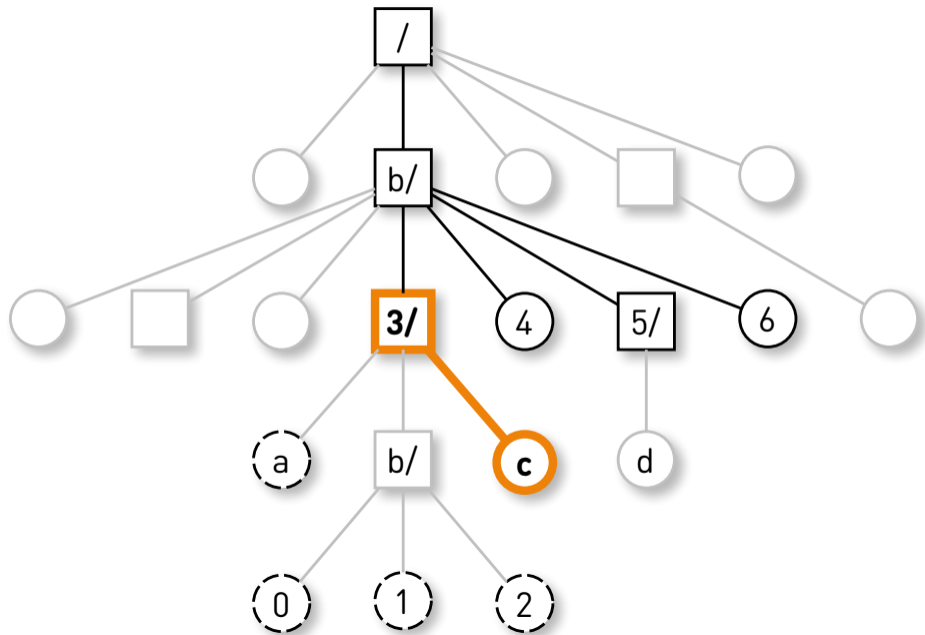


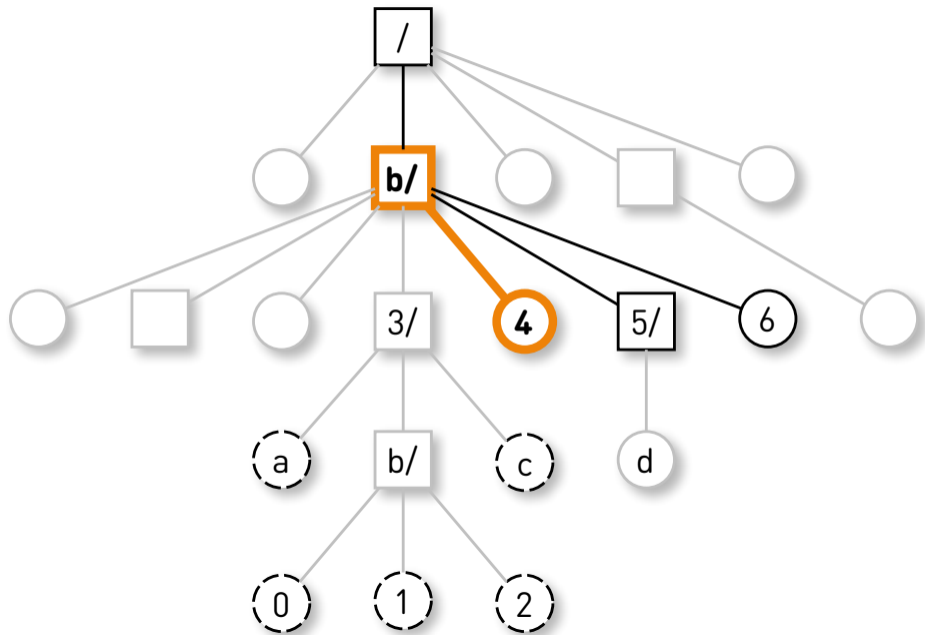


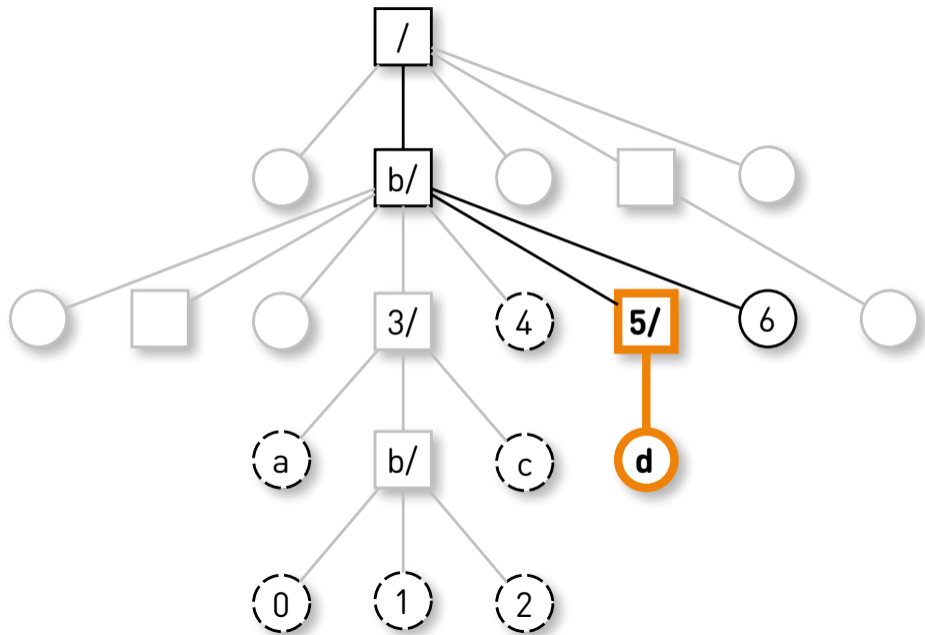


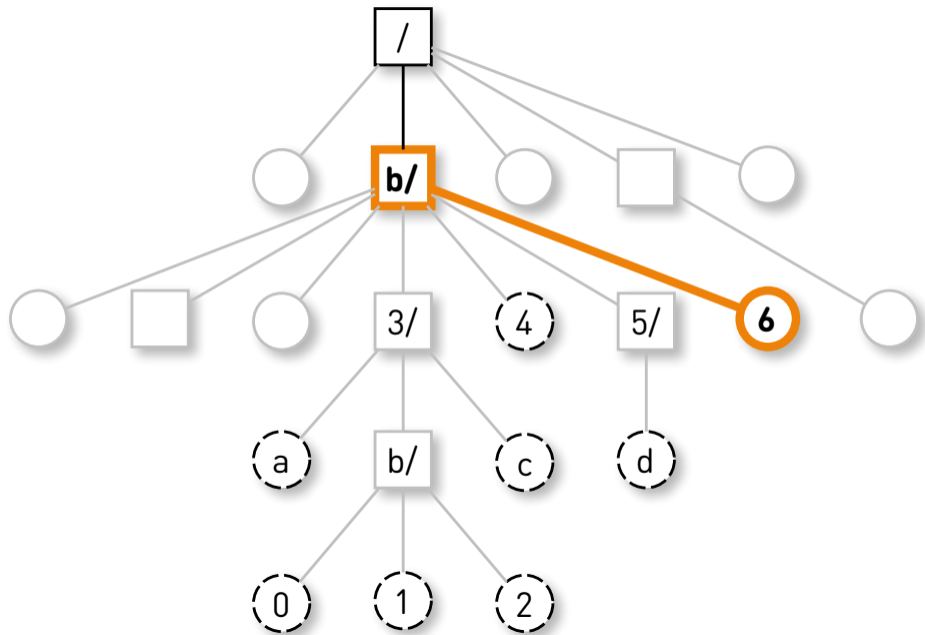


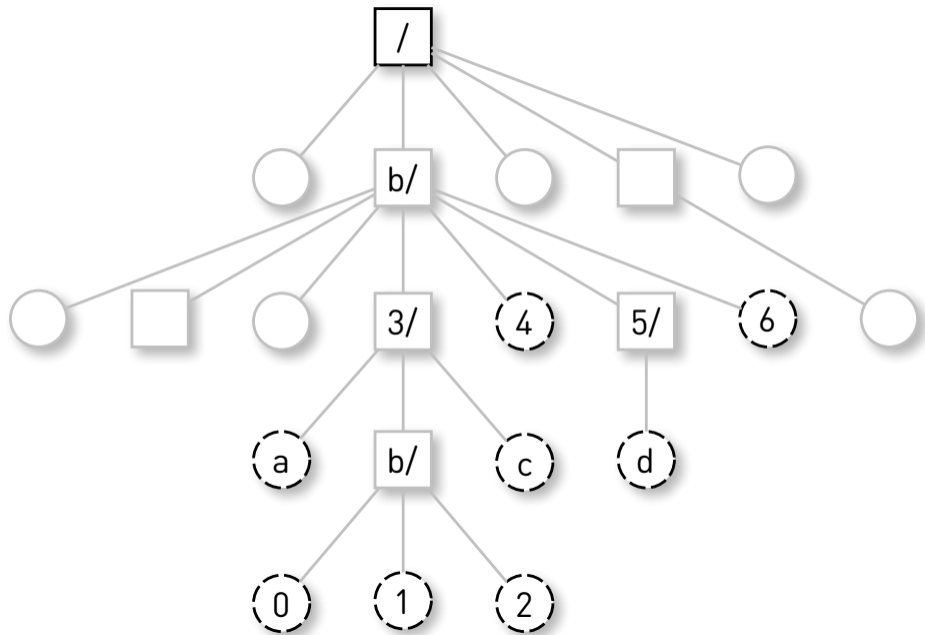


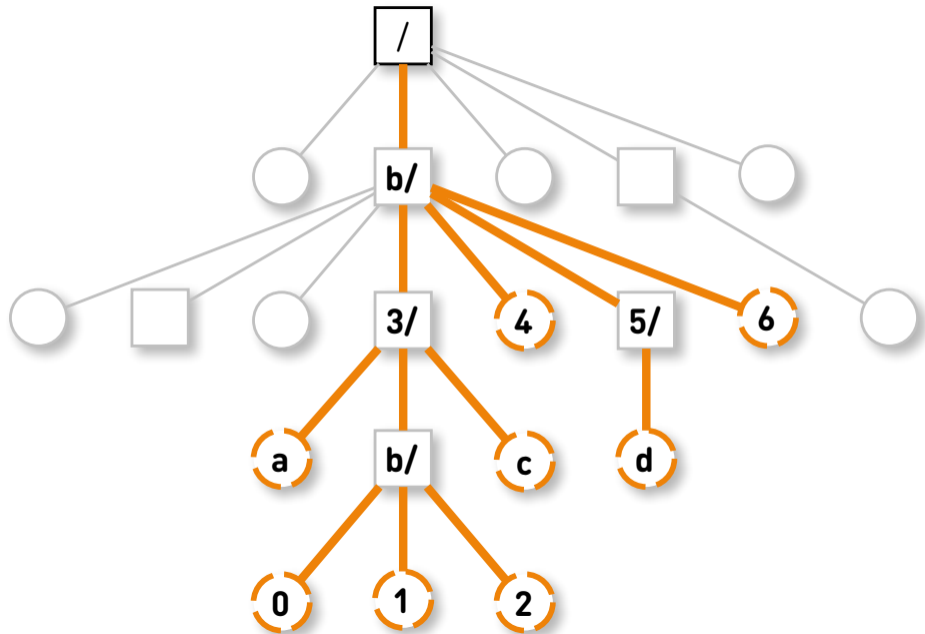












List w/o delimiter a.k.a. `ls -R`

- Each **descending step** — **two** non-transactional queries (objects/versions/uploads + subfolders)
- Non-transactional **index cleanup** on empty folders
 - WRITETIME slightly in the past

Hard Case

Deep hierarchy of almost empty folders.



C* wide partitions

- Each folder content is stored in **one partition**
 - **Not including** child subfolders

Guideline

- < **100K** objects/versions per folder
- Folder **striping**



Sonatype Nexus

content/tmp/	abc	...
---------------------	------------	-----

Move

content/tmp/	abc	∅
--------------	-----	---

content/vol-17/chap-42/	abc	...
-------------------------	-----	-----

Put

content/tmp/	abc	\emptyset	dbe	...
---------------------	------------	-------------	------------	-----

content/vol-17/chap-42/	abc	...
--------------------------------	------------	-----

Move

content/tmp/	abc	∅	dbe	∅
---------------------	------------	---	------------	---

content/vol-17/chap-42/	abc	...
content/vol-23/chap-07/	dbe	...

Put

content/tmp/	0ab	...	abc	∅	dbe	∅
---------------------	------------	-----	------------	---	------------	---

content/vol-17/chap-42/	abc	...
content/vol-23/chap-07/	dbe	...

Move

content/tmp/	0ab	∅	abc	∅	dbe	∅
---------------------	------------	---	------------	---	------------	---

content/vol-17/chap-42/	abc	...
content/vol-23/chap-07/	dbe	...
content/vol-31/chap-01/	0ab	...

Cemetery

content/tmp/	...	∅	...	∅	...	∅
	...	∅	...	∅	...	∅
	...	∅	...	∅	...	∅
	...	∅	...	∅	...	∅
	...	∅	...	∅	...	∅
	...	∅	...	∅	...	∅
	...	∅	...	∅	...	∅

Patched TemporaryLocationStrategy

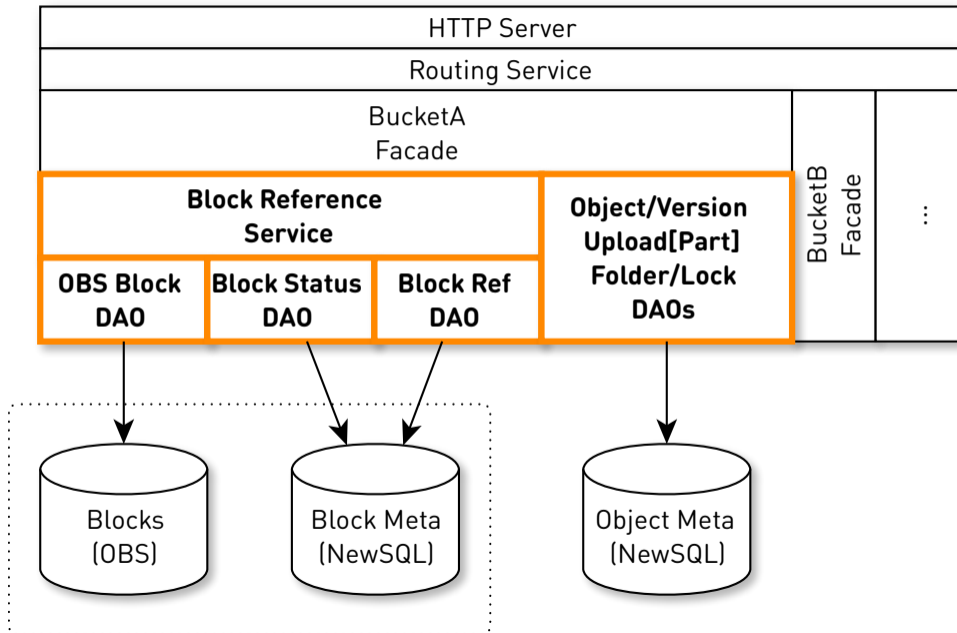
```
1 public class TemporaryLocationStrategy
2     extends VolumeChapterLocationStrategy {
3     @Override
4     public String location(final BlobId blobId) {
5         // Delegate striping to parent
6         // VolumeChapterLocationStrategy
7         return String.format(
8             "tmp/%s", super.location(blobId));
9     }
10 }
```



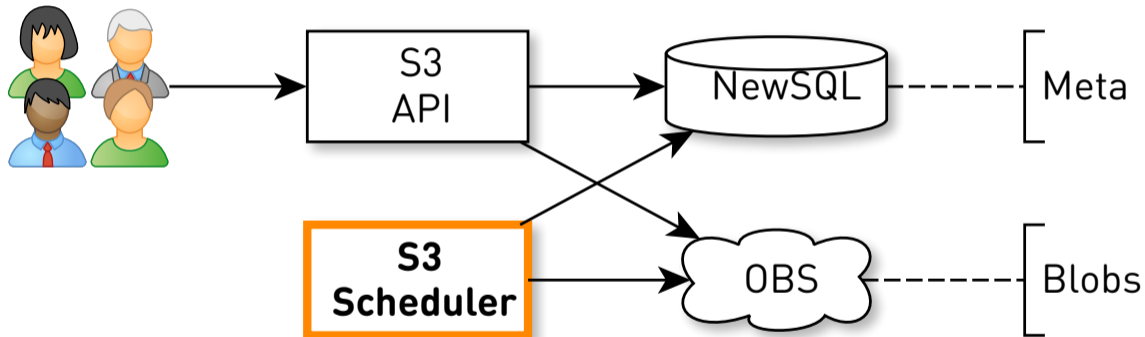
Profit

content/tmp/vol-17/chap-42/	abc	∅
content/tmp/vol-23/chap-07/	dbe	∅
content/tmp/vol-31/chap-01/	0ab	∅

content/vol-17/chap-42/	abc	...
content/vol-23/chap-07/	dbe	...
content/vol-31/chap-01/	0ab	...



Bucket Lifecycle Scheduler

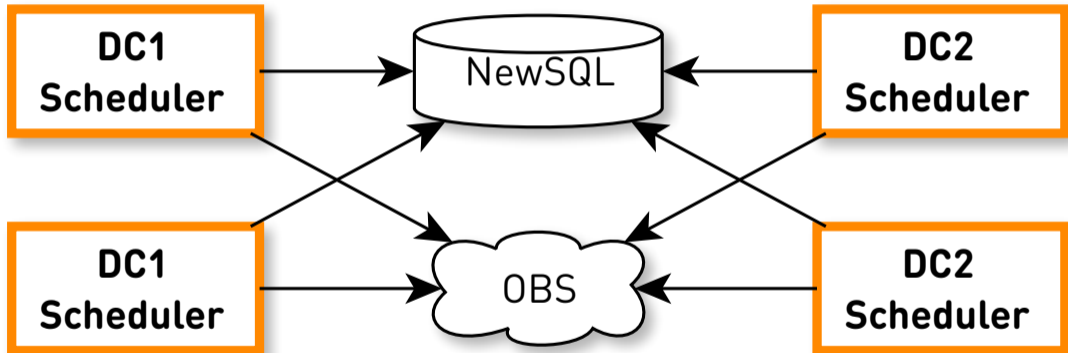


Bucket Lifecycle Scheduler

- 1 **Load** bucket lifecycle rules
- 2 Streaming **version scan**
- 3 Streaming **group by object**
- 4 **Filter** by rule
- 5 **Apply** the rule



Split work



Distributed parallel scanning

```
1 SELECT
2     parent, name, version, etag, tags
3 FROM
4     versions
5 WHERE
6     TOKEN(parent) >= ? AND TOKEN(parent) <= ?
```



1M expiring objects in a folder...

- A proven way to build a **cemetery of tombstones**
- With degraded performance and **timeouts**¹⁰

Solution

Tombstone throttling before applying rules.

¹⁰See wide C* partition

Gotchas

1. StringToSign

A string based on select request elements

2. Signing Key

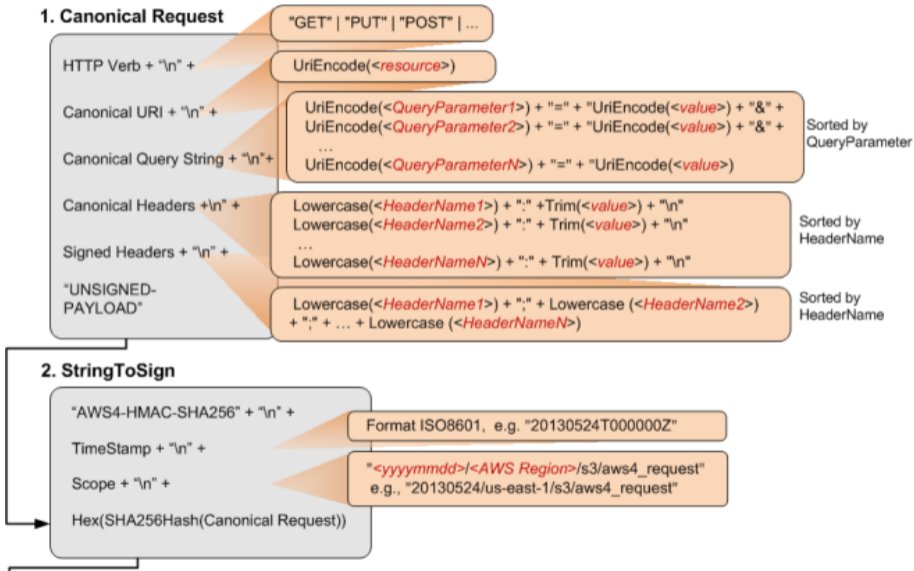
```
DateKey           = HMAC-SHA256 ("AWS4" + "<SecretAccessKey>", "<yyyymmdd>")
DateRegionKey     = HMAC-SHA256(DateKey, "<aws-region>")
DateRegionServiceKey = HMAC-SHA256(DateRegionKey, "<aws-service>")
SigningKey        = HMAC-SHA256(DateRegionServiceKey, "aws4_request")
```

3. Signature

```
signature = Hex(HMAC-SHA256(SigningKey, StringToSign))
```



AWS Signature Version 4 StringToSign



Gotchas

- Intricate canonicalization rules
- **Inconsistent** API¹¹ (i.e. Presigned URLs)
- Multipart object eTag
- **Custom** UriEncode() 😊
- Broken examples
- Expect: 100-continue
- x-amz-content-sha256:
STREAMING-AWS4-HMAC-SHA256-PAYLOAD

¹¹Developer guide - AWS SDK for Java 2.x



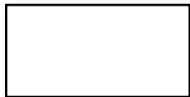
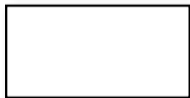
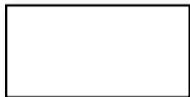
Performance

- Reads
- Writes
- Latency
- Throughput
- Hotspots



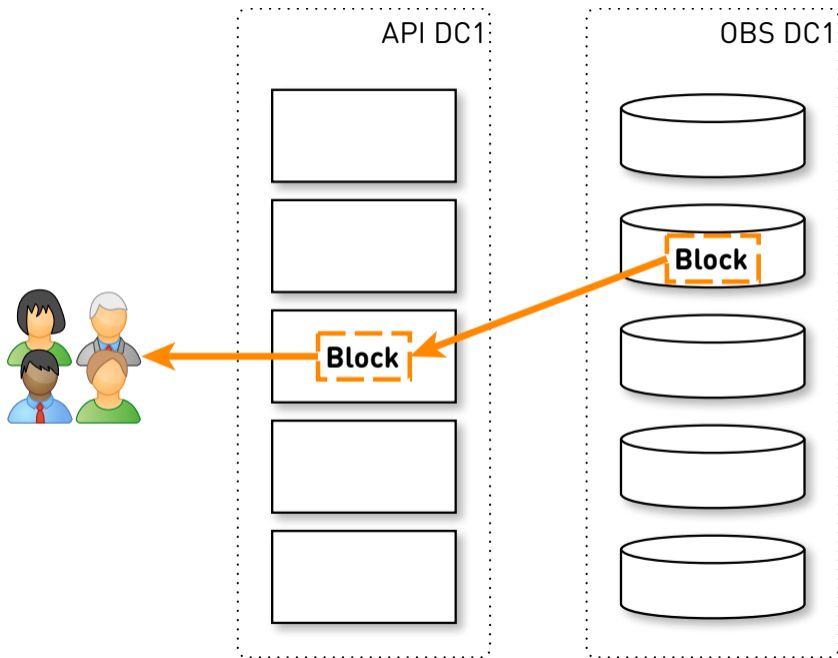


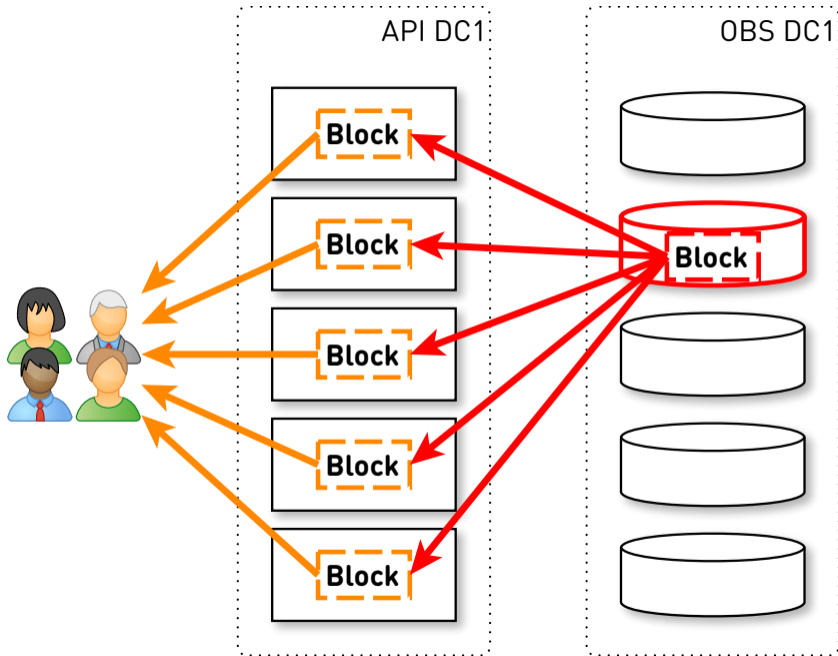
API DC1



OBS DC1



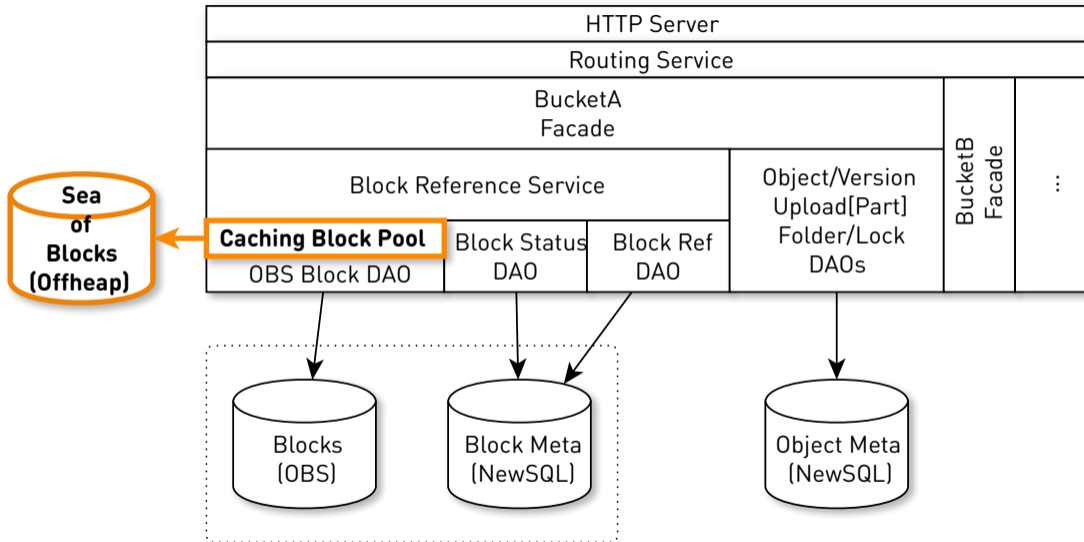


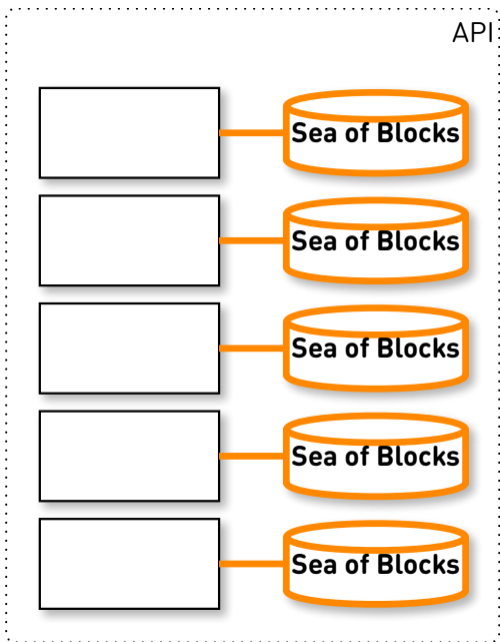


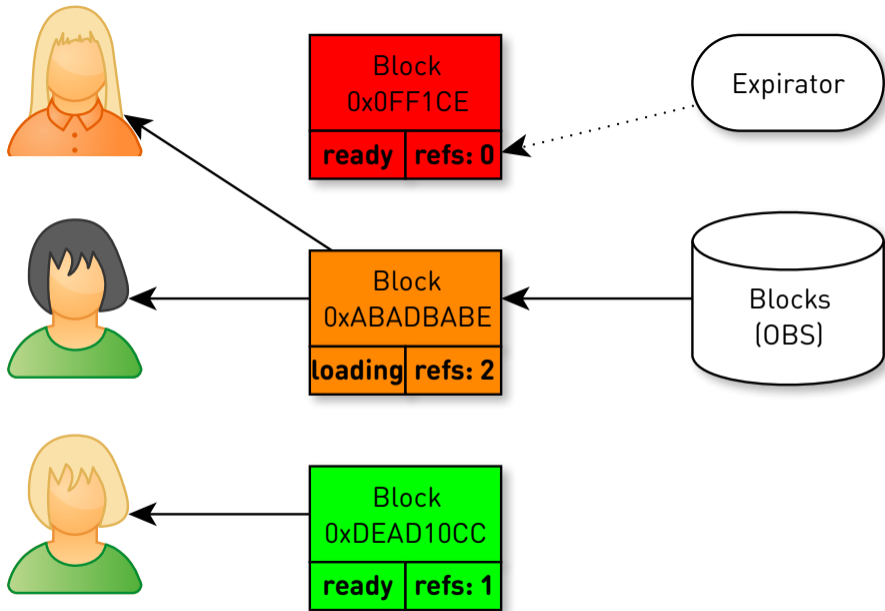
Naive block serving

- **Popular** blocks
 - Request/traffic **amplification**
 - **Identical** requests
 - Each block served by **3 blob replicas** (link saturation)
- **Extra data copying**¹²

¹²Vadim Tsesko. Reactive ok.ru/music streamer @ Joker 2018 (RU)







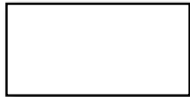
Sea of Blocks

- **Cache** recent blocks
 - Lower OBS load
 - Lower latency
 - **LRU** eviction
- Each block loaded **only once**
 - `CompletableFuture<CloseableByteBuffer>`
- **Offheap**
 - No extra memory copying
 - Reference tracking





API DC1



OBS DC1



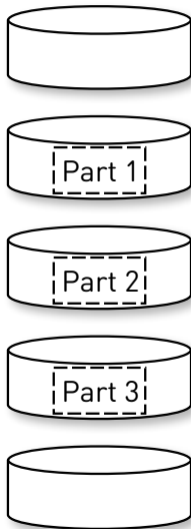


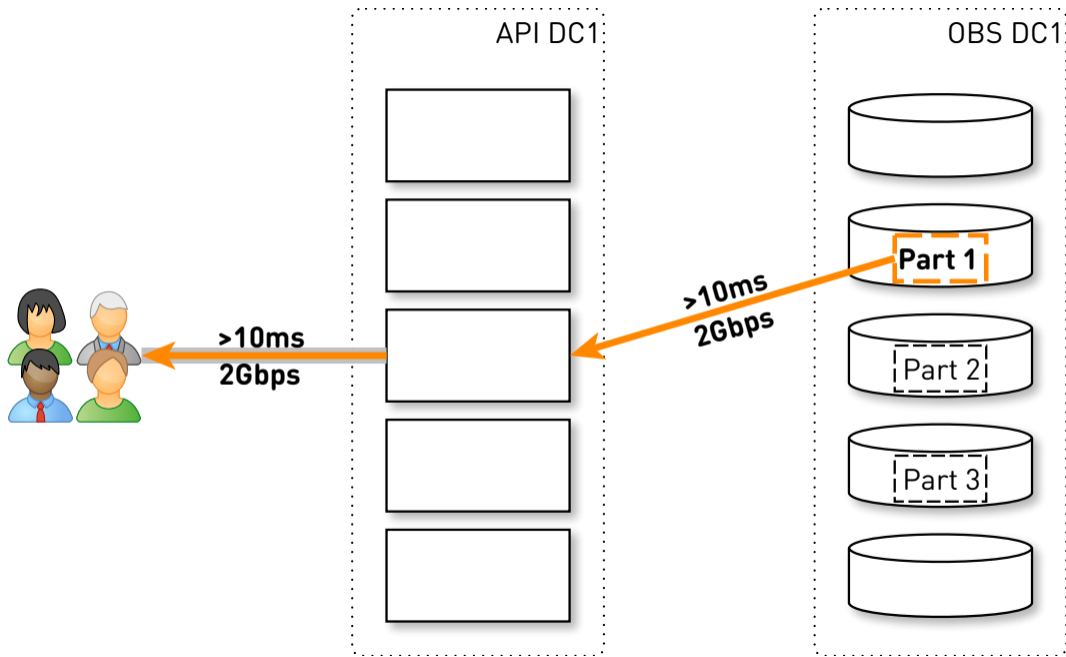
10Gbps

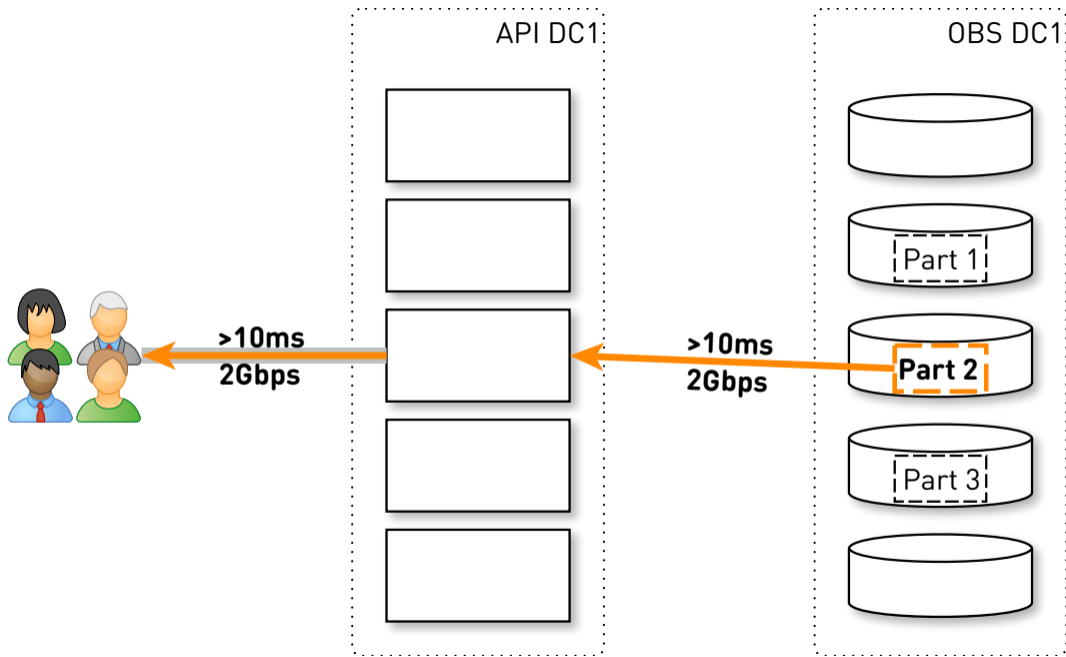
API DC1

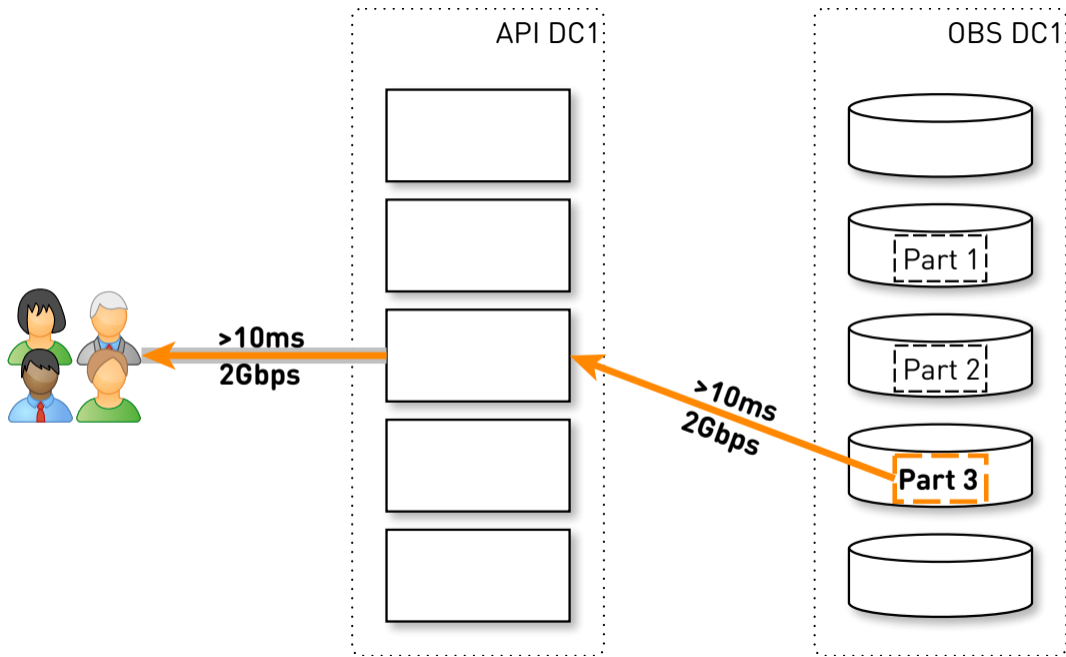


OBS DC1







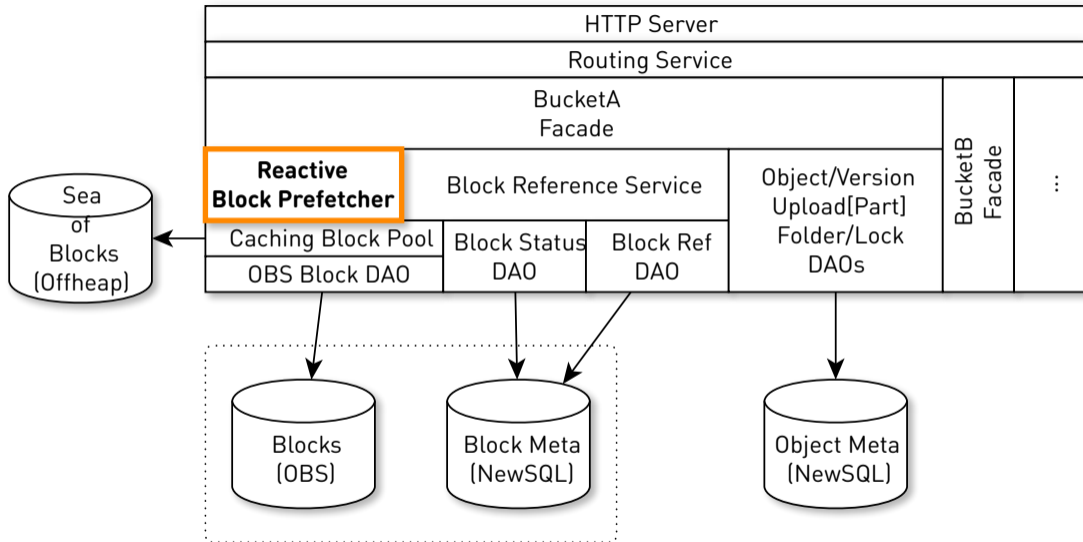


Huge Multipart Objects

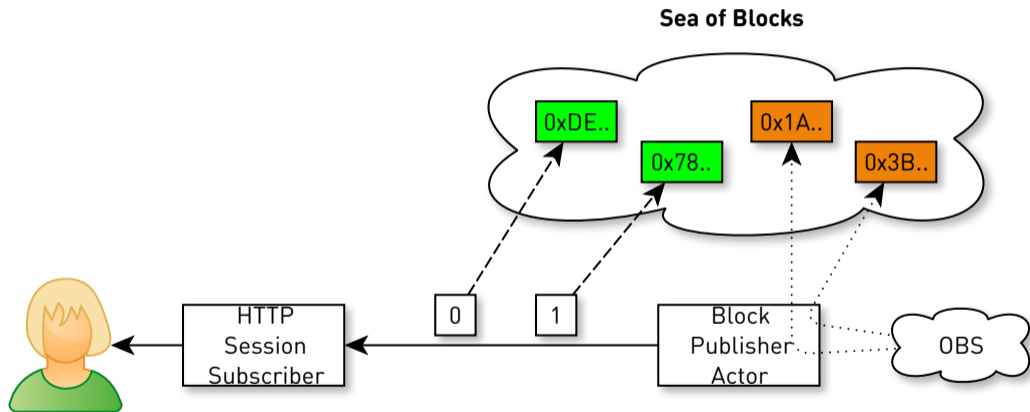
If parts served **sequentially** and **synchronously**:

- At the speed of **single OBS node**
- **Delays** to load next block
- **Underutilized** client throughput





Reactive Block Prefetcher¹³



¹³Powered by odnoklassniki/one-nio.

Reads are fast

- **Single non-transactional query** to get object meta and list of blocks
- Prefetch from **many OBS nodes** in parallel
- Wait for **the first block only if uncached**
- Reactive streams
- **Back pressure**



Modifications are slower

- **Common case**¹⁴
 - **4 sequential** transactions (10-30 ms)
 - **Write/remove** from OBS (30-300 ms)
- **Best case**¹⁵
 - **3 sequential** transactions
- **Amortized** due to **parallel** multipart upload

¹⁴Upload unique block or remove last block reference

¹⁵Deduplicated block



Numbers

Client hardware: **4x10 Gbps** + 256 GB /dev/shm

```
1 $ cat ~/.aws/config
2 ...
3 s3 =
4     signature_version = s3v4
5     max_concurrent_requests = 64
6     multipart_threshold = 16MB
7     multipart_chunksize = 16MB
```



/dev/shm/s3/200GB.random

```
1 $ dd if=<(openssl enc -aes-256-ctr -pass pass:"$(dd  
  if=/dev/urandom bs=128 count=1 2>/dev/null | base64)"  
  -nosalt < /dev/zero) of=/dev/shm/s3/200GB.random  
  bs=1G count=200 iflag=fullblock
```

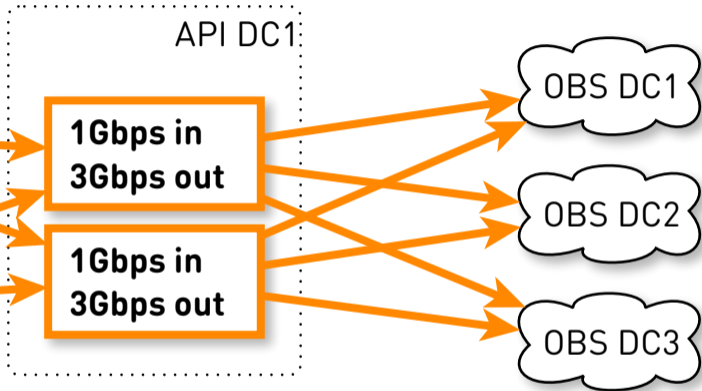
```
1 $ aws s3 cp 200GB.random \  
2   s3://sandbox/tsesko/stress/
```



1Gbps out



1Gbps out



Upload from /dev/shm

- **1 Gbps** AWS CLI upload limit
- 2 CLI instances hit **2 containers** in **1 DC**
- 4 cores, 1 Gbps IN, 3 Gbps OUT (each)
- CPU profile¹⁶
 - 47% SHA-256 (block ID)
 - 24% MD5 (checksum)
 - 15% NativeSslSocket.read() (HTTPS)
 - 6% NativeSocket.write() (OBS)

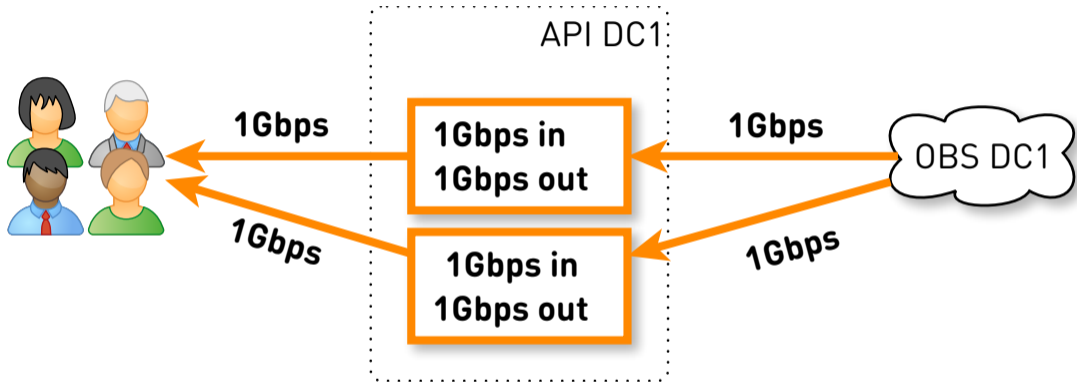
¹⁶<https://github.com/jvm-profiling-tools/async-profiler>

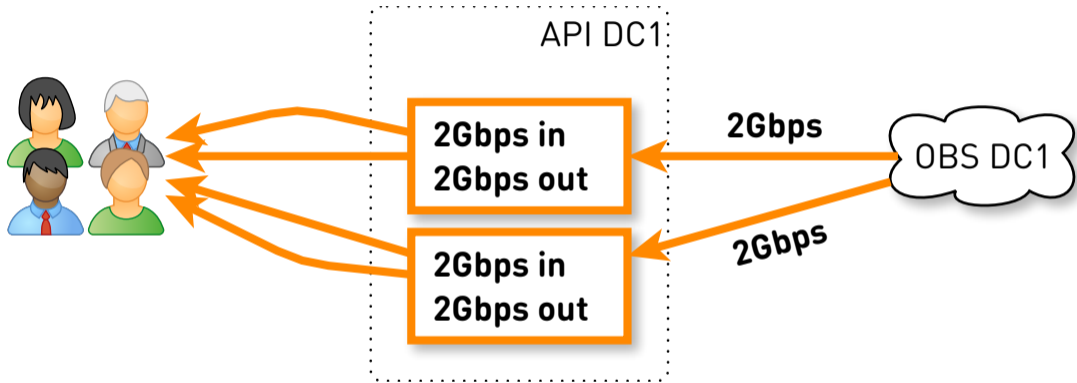


Download to /dev/shm

```
1 $ aws s3 cp \  
2   s3://sandbox/tsesko/stress/200GB.random \  
3   200GB.random
```







Download to /dev/shm

- **2 Gbps** AWS CLI download limit
- **2 processes** — **2x** throughput
- **Ranges** by AWS CLI
- 1 core, 1 Gbps IN, 1 Gbps OUT (each)
- CPU profile¹⁷
 - 55% SHA-256 (checksum)
 - 17% `NativeSocket.readRaw()` (OBS)
 - 15% `NativeSslSocket.writeRaw()` (HTTPS)

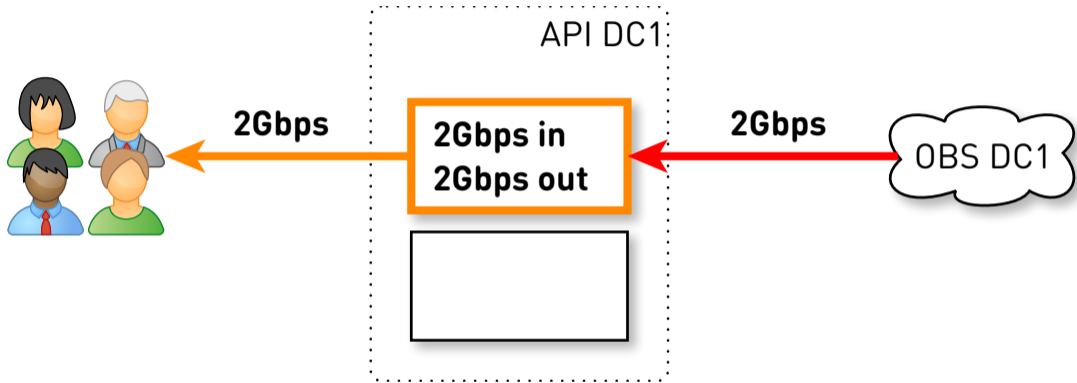
¹⁷<https://github.com/jvm-profiling-tools/async-profiler>

Download presigned B /dev/null

```
1 $ aws s3 presign s3://sandbox/tsesko/stress/200GB.random
2 <presigned URL>

1 $ curl "<presigned URL>" >/dev/null
```

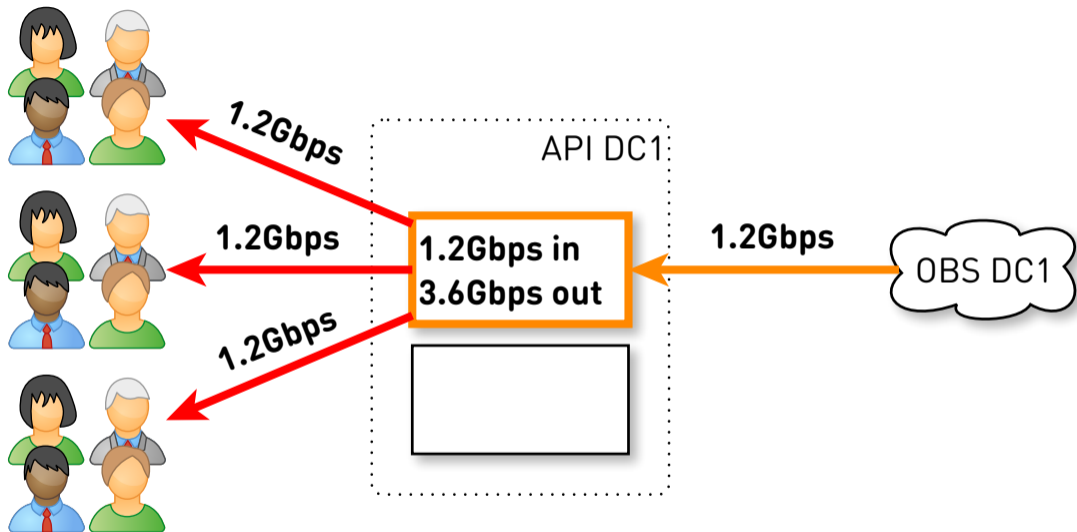




Download presigned B /dev/null

- **Single connection**
- **2 Gbps** download speed
- Limited by **inbound network** from OBS to API





Single-node stress test

- **3x** presigned curl 200GB.random
- Serves **3.6 Gbps** (4 Gbps quota)
- Downloads **1.2 Gbps** from OBS
- Block pool **cache hit 60%**
- **2 cores**
 - 46% `NativeSslSocket.writeRaw()` (HTTPS)
 - 35% SHA-256 (checksum)
 - 10% `NativeSocket.readRaw()` (OBS)
 - GC is idle (offheap)



Production

- Tens of buckets and growing
- Buckets up to **50 TB**
- Up to **100M objects** per bucket
- **> 6Krps**
- **99.9% < 15 ms** (meta)




Use Cases

- Docker registry
- Airflow Logs
- Nexus Artifacts
- Teamcity Artifacts
- Flink Checkpoints
- ...



Internal Users

- RPM repository
- Static resources
- Autotest artifacts
- Android profiling artifacts¹⁸
- Continuous profiles¹⁹
-  **RuStore** beta

¹⁸Kirill Popov. Profiling in production @ Mobius 2021 Piter (RU)

¹⁹Andrei Pangin. Continuous eBPF-assisted cloud profiling @ JPoint 2022



Q&A



Vadim Tsesko

Odnoklassniki

incubos.org/contacts

oktech.ru

