CORTX Monthly Meet an Architect Series: Multisite Replication

Tim Shaffer
CORTX Advanced Development Group/Community

December 2, 2021

Introduction

01 Key Terms

02 Multisite Replication in S3

03 Design Challenges

04 Alternative Approaches

CORTX

05 Replication Architecture

06 S3 Server

07 FDMI

08 Multisite Replicator

Key Terms

- **S3:** Simple Storage Service. Cloud-based object storage service/protocol developed by Amazon. Protocol is also used by other object storage systems (MinIO, Ceph, CORTX)
- **Object:** basic unit of data in S3. Similar to a file on a local computer, can also have user-defined tags, ACLs, multiple versions.
- **Bucket:** container for objects. Serves as a mapping from string keys to objects. *Does not* have hierarchical structure like directories. Stored in a particular site/cluster/region.
- **Replication:** automatic propagation of newly added/deleted objects from source to destination buckets. Source and destination may be in different sites. Not synchronization.

Uses of Multisite Replication

- Data redundancy
- Data locality/minimizing latency
- Copying to different storage types (S3 storage class, private cloud)
- Compliance requirements
- Sharing between accounts/users
- Application logic (collecting logs, event-based/dataflow processing, etc.)

Replication in S3

1. User sets a replication policy on a bucket
   - Filters can limit scope of replication
   - Can replicate to multiple destinations
2. User uploads a new object matching the filter(s)
   - Policy does not apply to existing objects
3. S3 server updates object metadata
   - `x-amz-replication-status = PENDING`
4. <time passes>
5. Object is uploaded to destination bucket
6. Object metadata is updated after replication
   - `COMPLETED` or `FAILED` depending on outcome of replication
   - Replicas are marked `REPLICA`

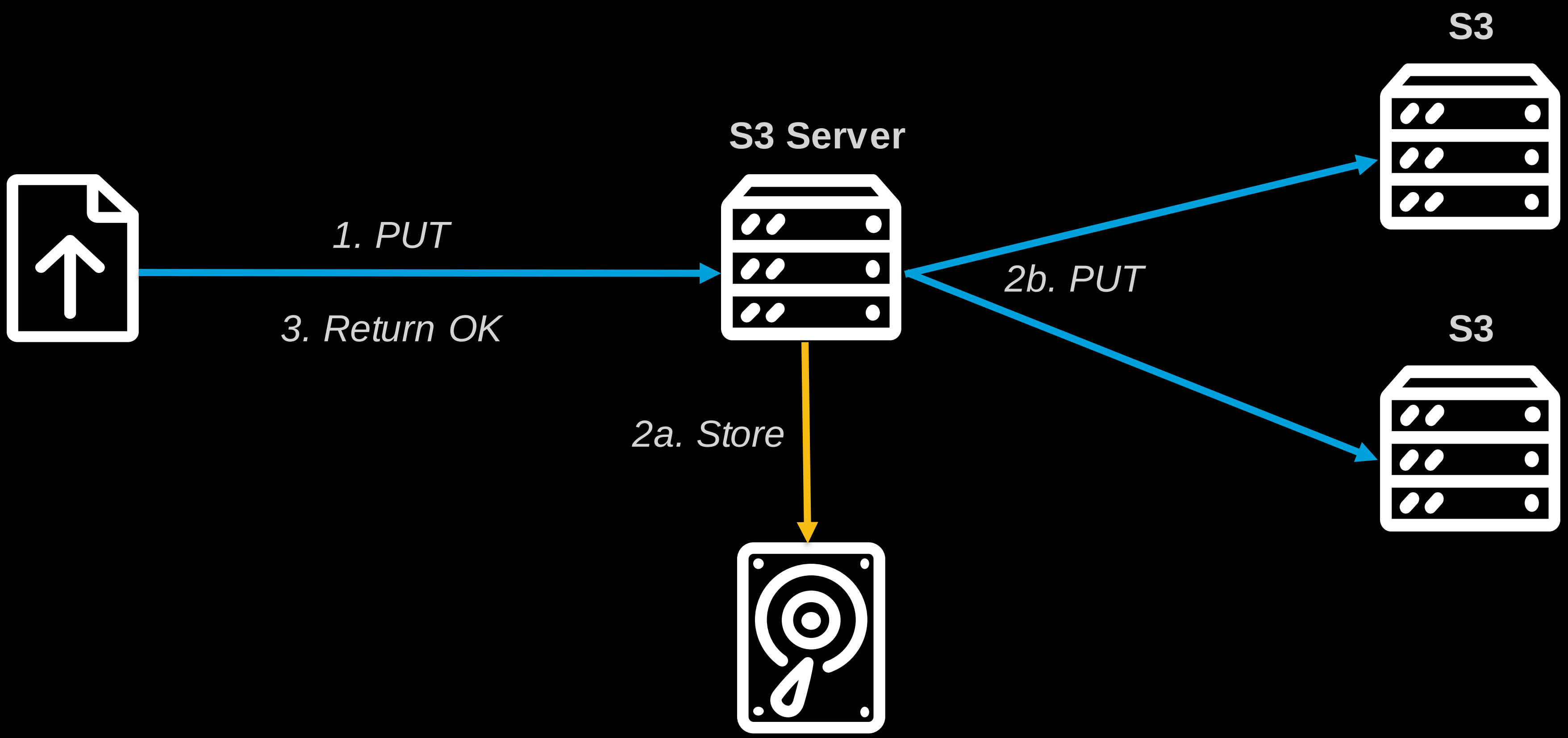
```
{
  "Role": "arn:aws:iam::123:role/replication",
  "Rules": [
    {
      "Status": "Enabled",
      "Priority": 1,
      "DeleteMarkerReplication":
          { "Status": "Disabled" },
      "Filter" : { "Prefix": "foo"},
      "Destination":
          { "Bucket":"arn:aws:s3:::BUCKET2" }
    }
  ]
}
```

Design Challenges

Replication must be

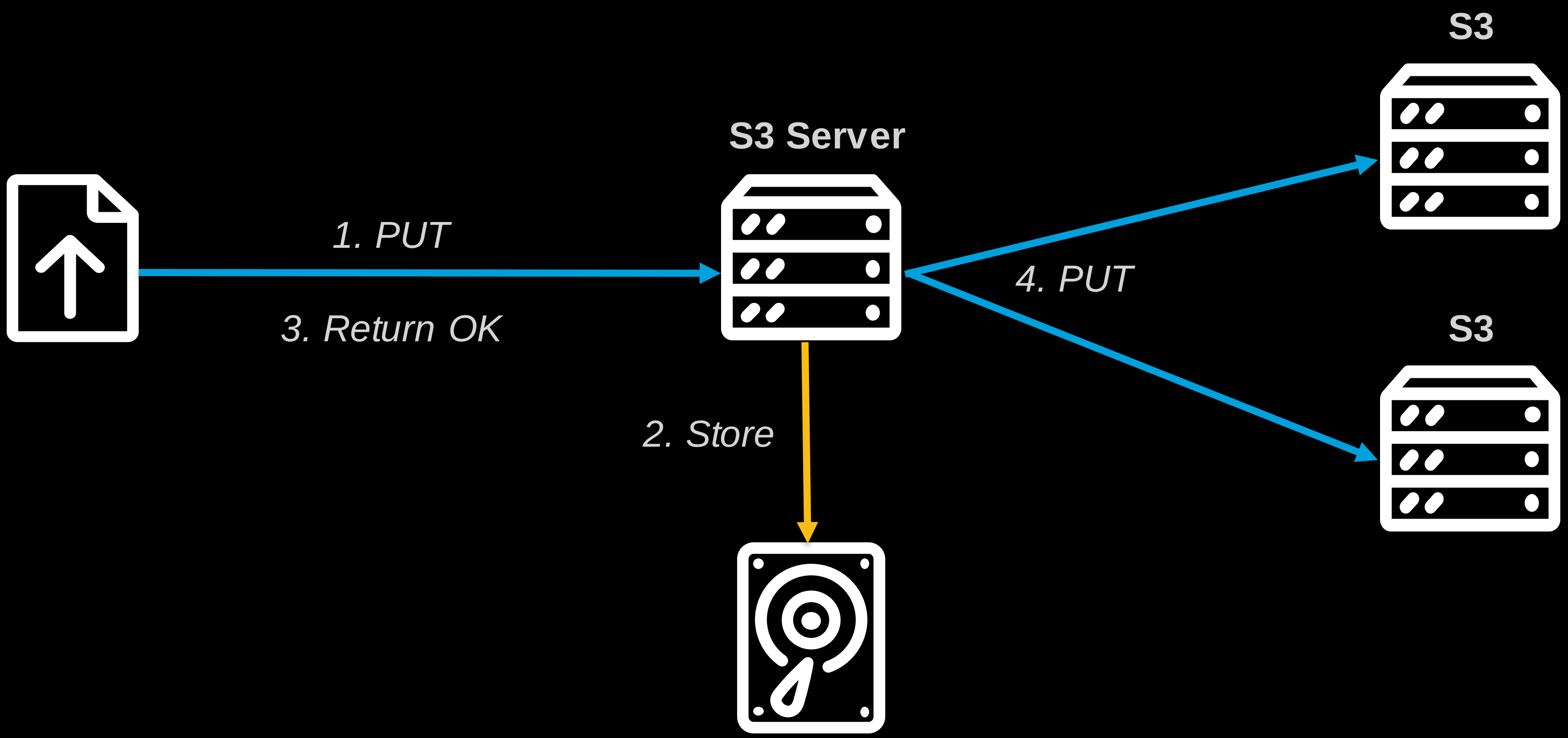
- **Reliable:** system can't "forget" to replicate objects.
- **Asynchronous:** out of the critical path on object put.
- **Fault-tolerant:** must recover/retry after failures in S3 server, network, destination site.
- **Faithful:** object contents, metadata, version IDs, etc. should be preserved.

Alternative: Synchronous Replication



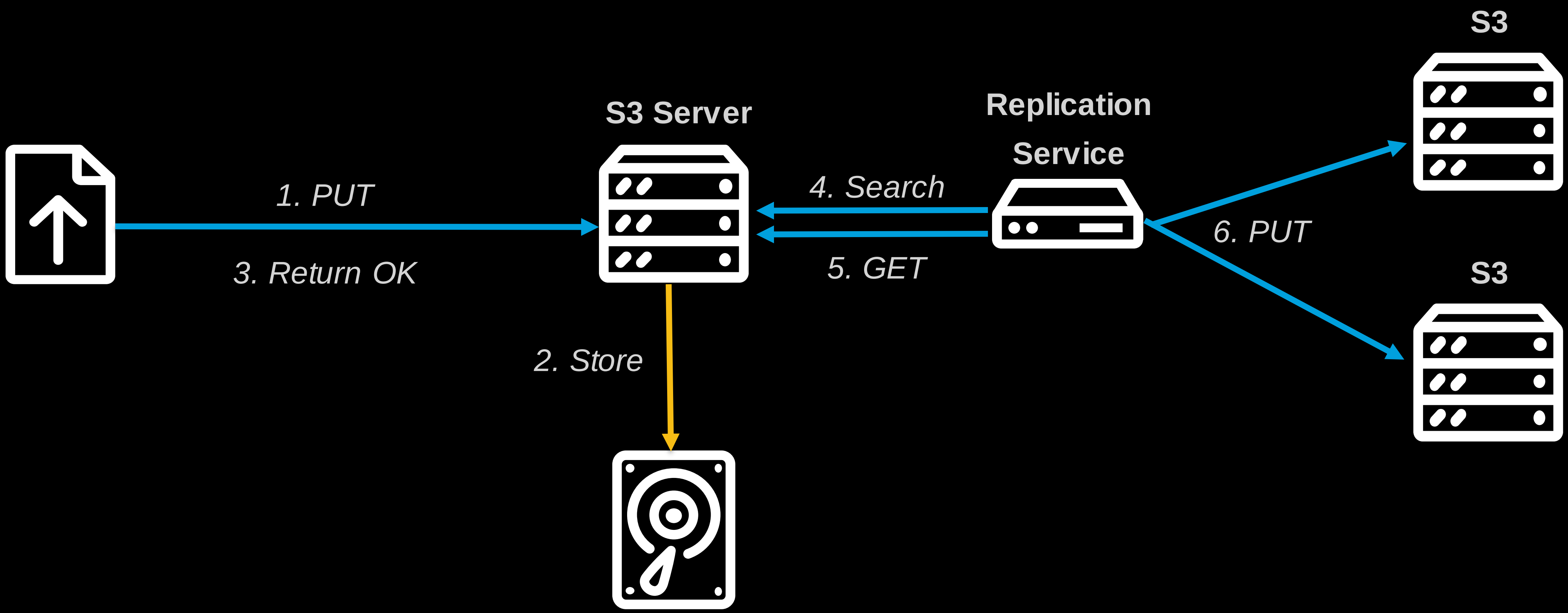
- Time to put objects now depends on all destinations (max of storage, replication times)
- Typically have a faster link within site (2a) than between sites (2b)
- More replication targets increases the chance of failure

Alternative: "Background" Replication



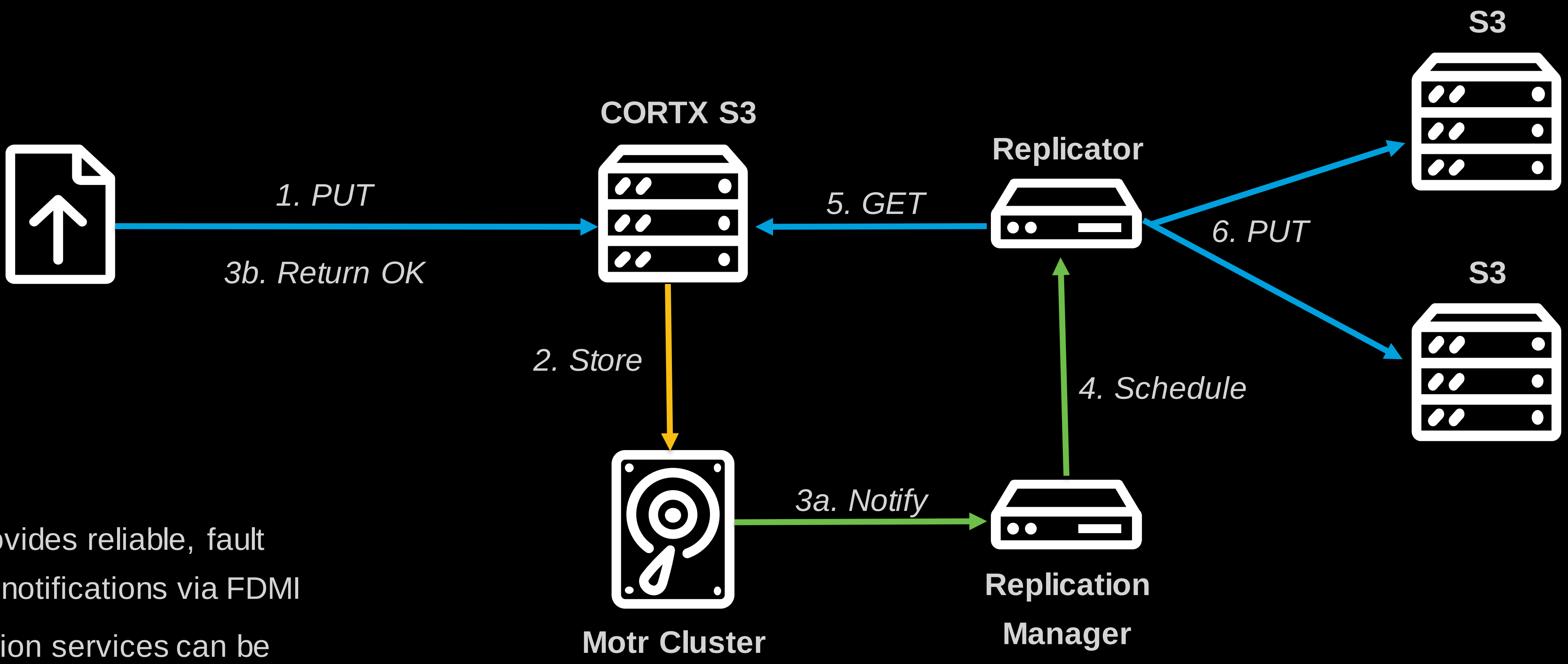
- Server returns OK before actually performing replication
- Partial/undefined replication state in case of S3 server crash
- Scales poorly: single S3 server can become overloaded

Alternative: Replication Service



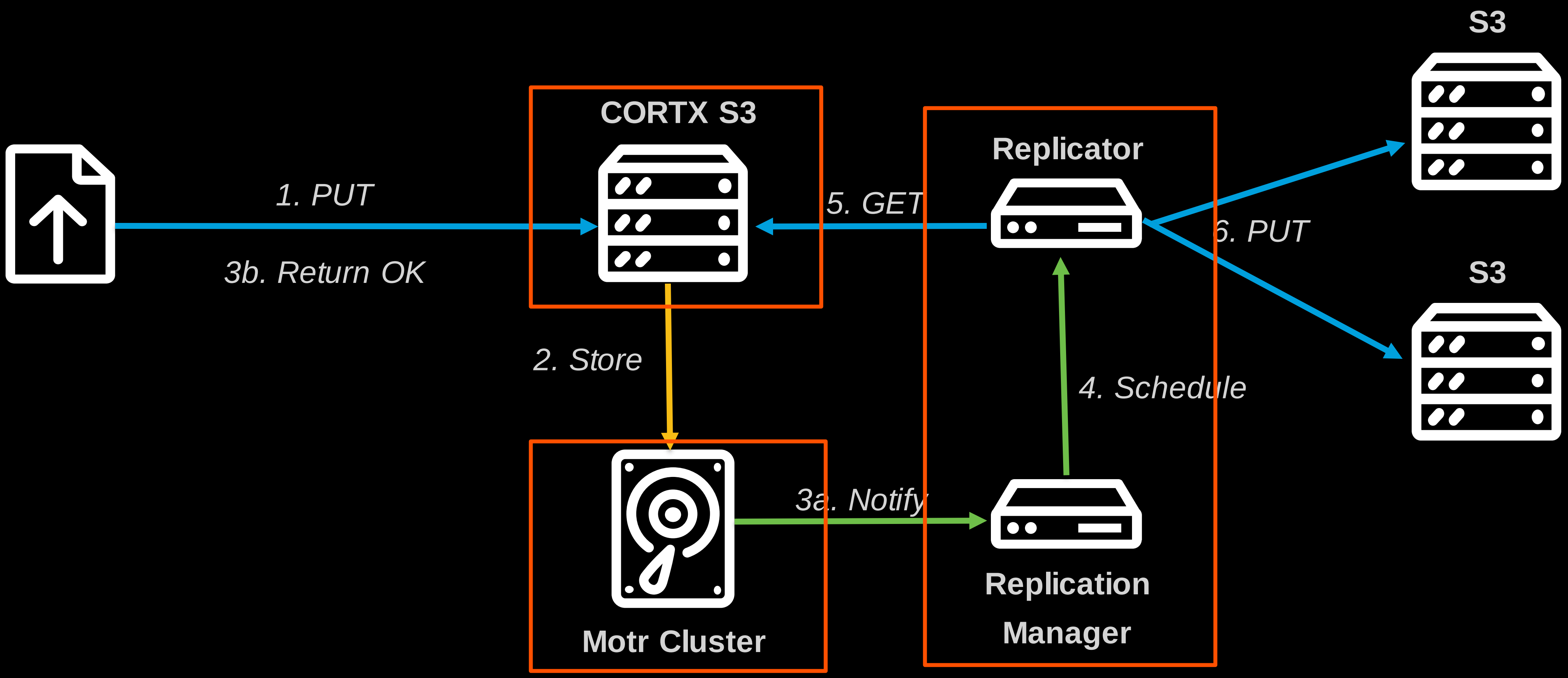
- Need to be careful replication events aren't lost if any of the components crash
- S3 server should be stateless, so how to find un-replicated objects?
  - *Linear search? Too slow on large buckets*
- Tricky to coordinate scale-out: need a multi-producer multi-consumer queue

CORTX Replication Architecture



- Motr provides reliable, fault tolerant notifications via FDMI
- Replication services can be scaled independently
- Faithful copy when destination is a CORTX cluster

CORTX Replication Architecture



Motr

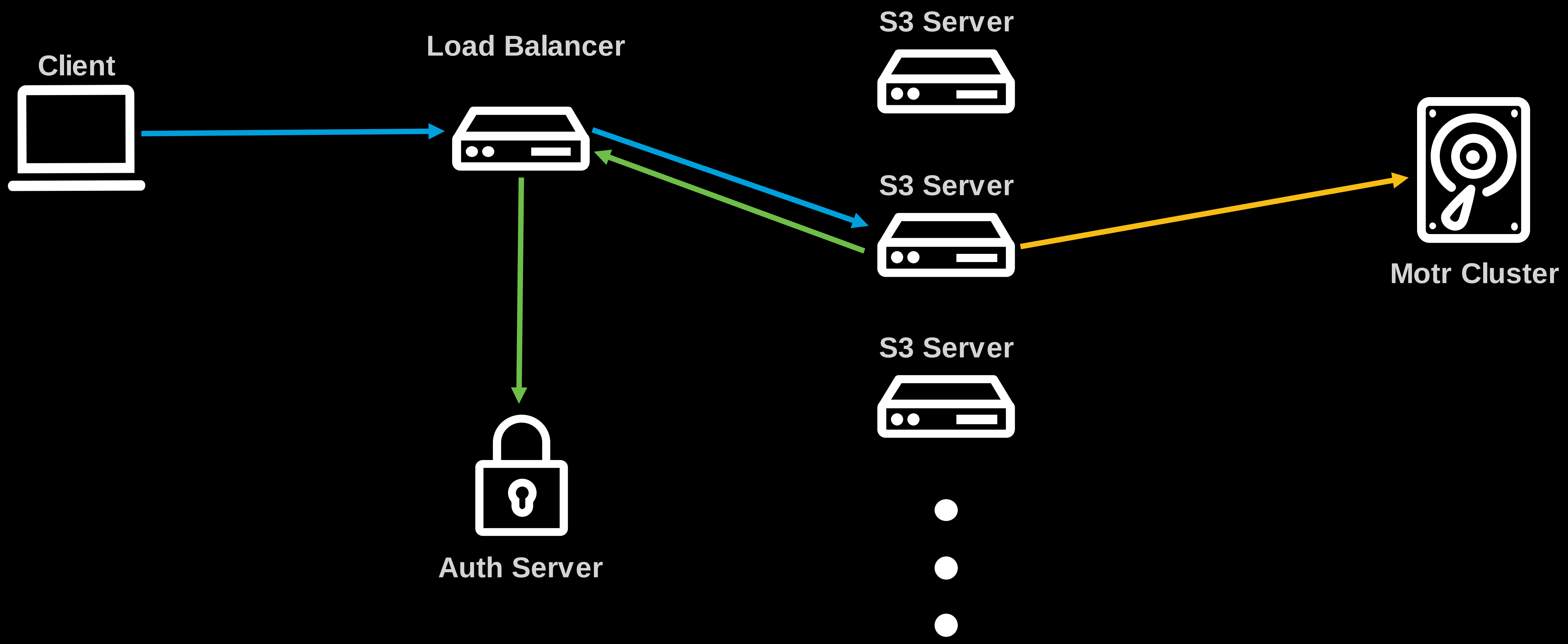
- The core storage layer of the CORTX stack
- Manages a pool of storage nodes with multilevel erasure coding
- Provides highly reliable storage of arbitrary-size objects, along with a distributed key-value store for metadata
- Designed for extreme scale, high efficiency, advanced storage configurations

We won't focus on Motr internals here: S3 server and replicator are just clients

CORTX S3 Server

- Translates S3 API actions (Put Object, List Bucket, etc.) into Motr operations (Index Create, KV Store, Write to OID)
- Designed to be stateless: provision more S3 servers to meet demand, Motr handles all persistent state.
- Request authentication/authorization is handled by a separate service
- Currently under active development, S3 API coverage is always improving

CORTX S3 Server



S3 Server: Bucket Metadata

- Metadata for each bucket is stored in a Motr index
- Includes basic info (owner, ACL) along with internal layout info (IDs to access object index, multipart uploads, etc.)
- When replication is set up, policy gets stored in bucket metadata
- For remote cluster access, also need to store credentials for role

```
{
  "ACL": "8o7sanNT80N89n7IUONh870N...
  "Owner-Account": "s3_user",
  "create_timestamp": "2021-11-29T04:54:02",
  "motr_object_list_index_layout": "GYUInAA967...
  "ReplicationConfiguration": {
  {
    "Role": "arn:aws:iam::123:role/replication",
    "Rules": [
      {
        "Status": "Enabled",
        "Priority": 1,
        "DeleteMarkerReplication":
            ...
```

S3 Server: Object Metadata

- Each bucket has an index with metadata for each object
- Basic info (size, MD5, owner, etc.) along with Motr Object ID (OID) that stores object data
- Replicated objects also have a state (`PENDING`, `COMPLETED`, `FAILED`, or `REPLICA`)
- Replication status is available to the user via the `HeadObject` API call

```
{
  "Object-Name": "foo",
  "Bucket-Name": "bar",
  "Size": 1702,
  "Content-Type": "text/plain",
  "Content-MD5": "b23a78e5b235aa98762b3548cf36...
  "Last-Modified": "2021-11-29T04:54:02",
  "motr_oid": "tQ65BAAAAAA=-CwAAAAAAdMM",
  "x-amz-replication-status": "PENDING",
  ...
```

S3 Server Replication Steps

1. Check replication policy on the bucket
    - Policy includes role ARN
2. Check if object matches one or more replication rule
    - Can select a subset of objects based on name prefix, user-defined tags
3. For remote clusters, get credentials (Access Key, Secret Key)
    - Will be attached to a role ARN
4. Apply `x-amz-replication-status = PENDING` to object metadata
5. That's it!

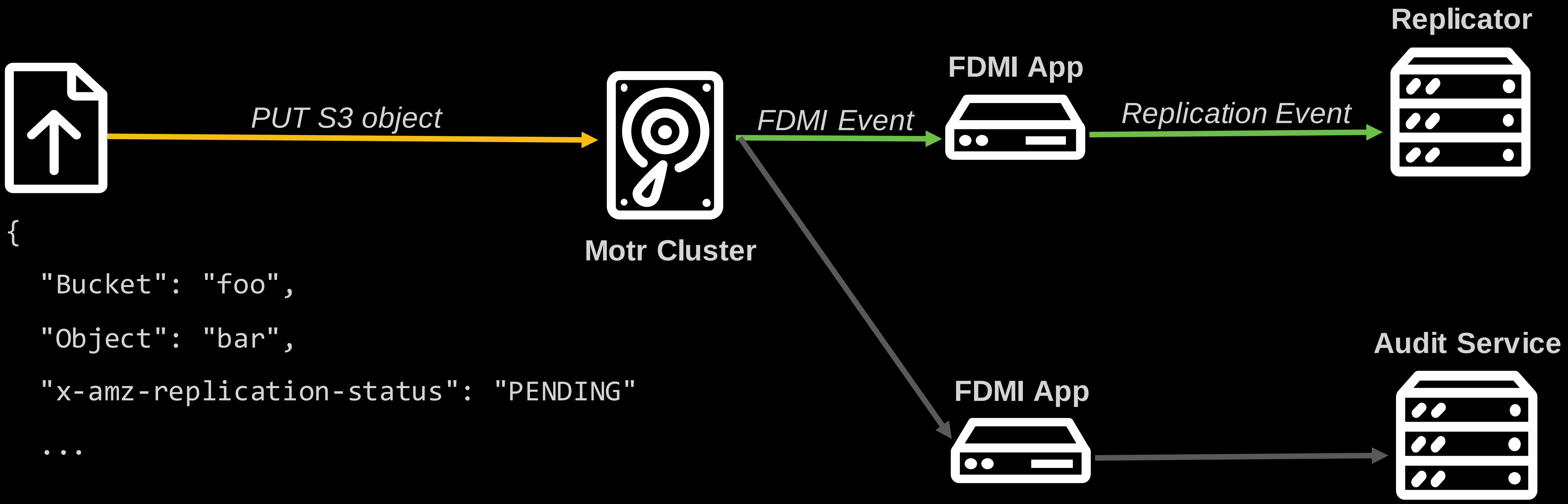
File Data Manipulation Interface (FDMI)

- Motr provides a flexible plugin interface to notify applications of events in the cluster
- Capable of fine-grained notifications on individual IO actions, filtering events of interest

Wide range of potential uses:

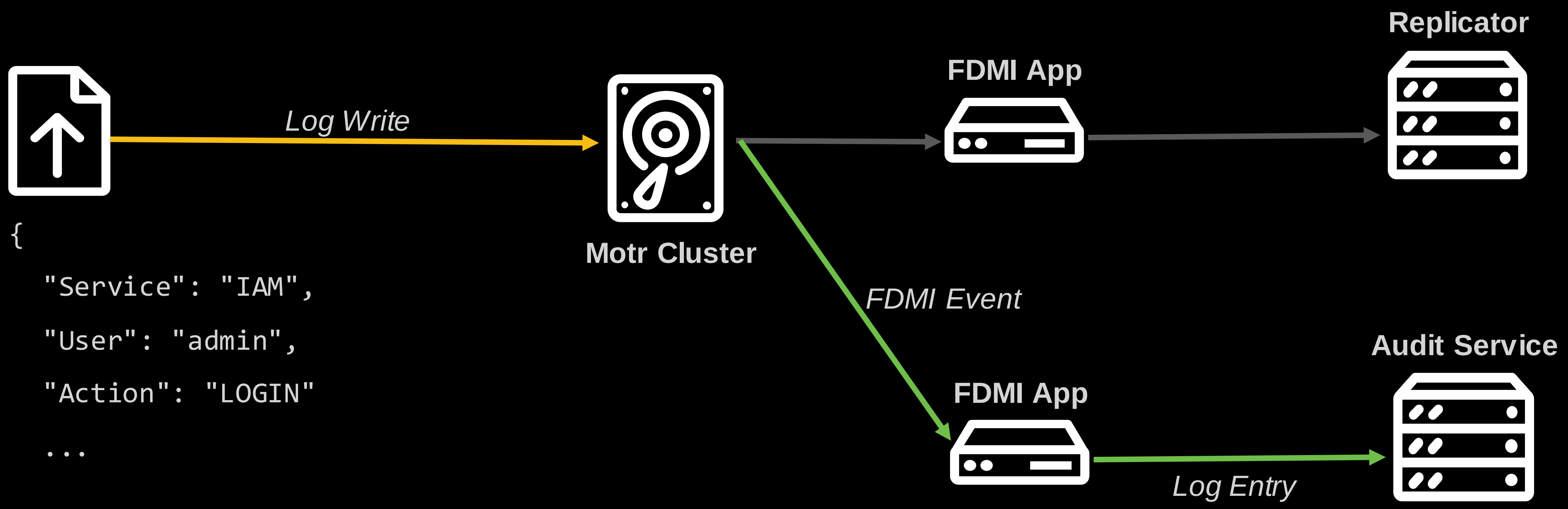
- backup, migration, replication, HSM apps, preventive file-system checking
- burst buffer prefetching and destaging, background compression, de-duplication
- online conversion of pre-existing cluster to a new format or a new meta-data schema
- audit, logging apps
- full-text indexing, searching apps

FDMI Applications



```
{
  "Bucket": "foo",
  "Object": "bar",
  "x-amz-replication-status": "PENDING"
  ...
```

FDMI Applications



```
{
  "Service": "IAM",
  "User": "admin",
  "Action": "LOGIN"
  ...
```

FDMI Filters

```
fdmi_filters:
  - name: replicator
    node: localhost
    client_index: 0
    substrings: [
      "Bucket-Name",
      "Object-Name",
      "x-amz-meta-replication"]
```

```
{0x6c|
  ((^l|1:81), 2, ^l|1:81, "", ^n|1:3, ^v|1:66,
  [3:
    "Bucket-Name",
    "Object-Name",
    "x-amz-meta-replication"],
  [1: "192.168.12.2@tcp:12345:4:1"])},
```

- FDMI filters can be specified in cluster config
- Configured with connection info for FDMI app
- Supports matching on multiple substrings
- Cluster config gets translated to low-level config on the right

FDMI Events

- Opcode specifies the type of event (here 231 is create KV entry)
- Some additional info (which node produced the event, which filter it's for)
- Also includes the key and value for the matching operation (provides full S3 object metadata)
- Substring matching on the value can select only S3 object puts that have replication configured

```
{
  "op": 231,
  "fid": "<54000000600012345:123450>",
  "cr_key": "bar",
  "cr_val": {
    "Bucket-Name": "foo",
    "Object-Name": "bar",
    "x-amz-meta-replication": "PENDING",
    "Size": 1702,
    ...
```

Key Properties of FDMI

- **Reliable delivery of events**
  - Though duplicated events are sometimes possible
  - Not a problem for multisite replication, as it's easy to detect when objects are up to date
- **Does not block critical IO path**
  - Event delivery is asynchronous via RPC mechanism
- **Provides a publish-subscribe interface to applications**
  - Applications subscribe to specific filters
  - Once the application handles the event, it must explicitly release the event
- **Fault-tolerant design**
  - On application crash, events are queued and delivered once services are restored

Digression: Why not in the S3 Server?

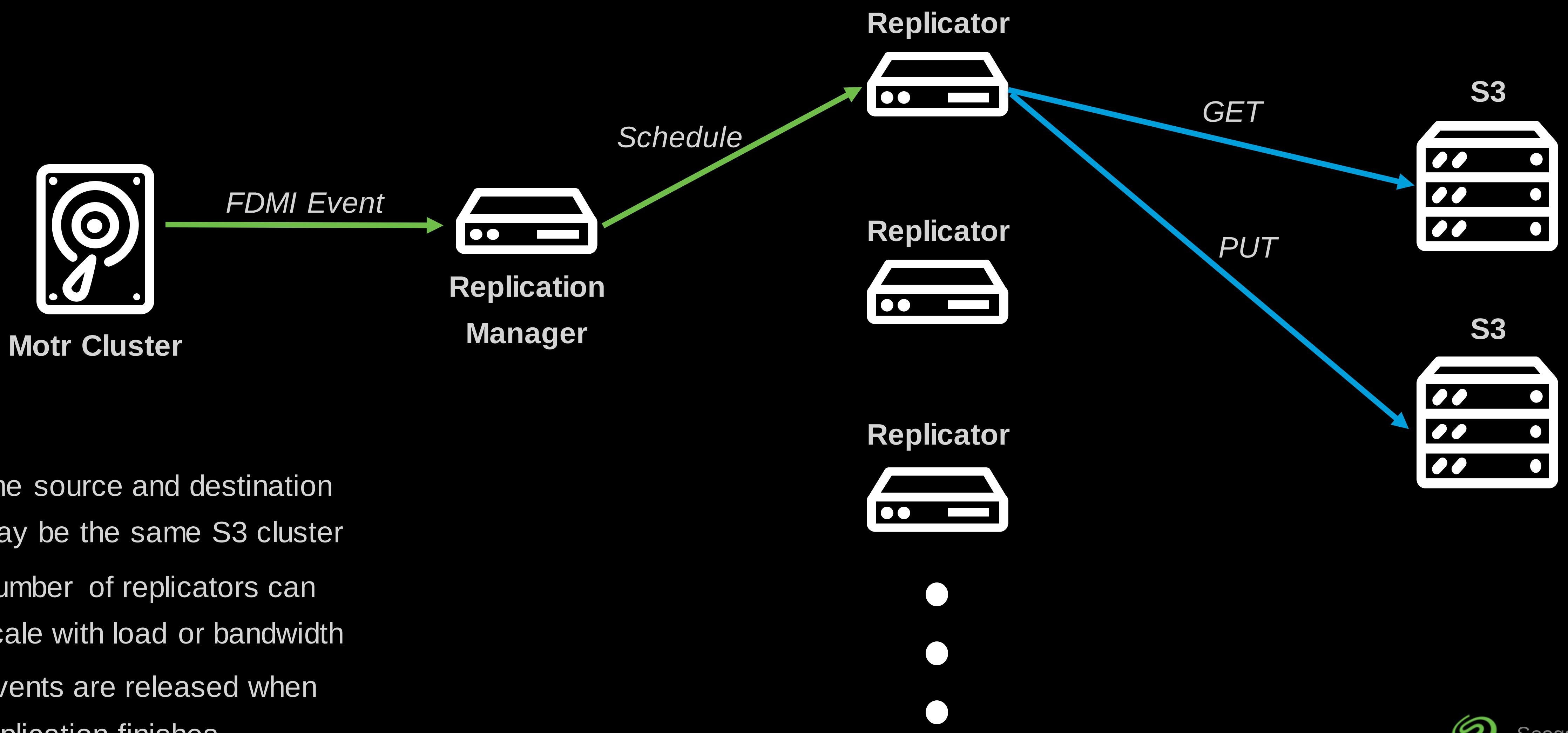
The S3 server could also implement its own queue using Motr primitives, or any number of existing pub-sub systems. Why use FDMI?

- **Deep integration with the storage layer**
  - Transactionally coupled with the core storage, can take advantage of data placement
- **Avoids architectural modifications to Motr clients**
  - S3 server only needs to add a metadata field (!)
  - More work to integrate existing applications with a separate pub-sub system
- **Abstracts commonly needed (and difficult) distributed capabilities**
  - Hard to get failure cases, distributed transactions, etc. right
  - Better to do it once inside Motr instead of reimplementing in every application

Multisite Replicator

- Responds to FDMI events on S3 object put (with matching prefix, tags)
- Carries out replication asynchronously over normal S3 API
- Can replicate to CORTX or other S3 providers (AWS, MinIO, etc.)
- Can scale up by provisioning more replicator instances
- Reliable and fault tolerant (both replicators and managers)

Multisite Replicator



- The source and destination may be the same S3 cluster
- Number of replicators can scale with load or bandwidth
- Events are released when replication finishes

Replication Process

On an FDMI event, the replication manager will

1. Validate metadata *(What if an object is named `x-amz-replication-status PENDING`?)*
2. Get replication destination(s), remote cluster credentials
   - Destination is included with object metadata, credentials can be looked up using role
3. Schedule replication event to one of the replicator instances
4. On completion, update `x-amz-replication-status` on the source object

A replicator instance performs the actual transfer

1. Get object from source bucket
2. Validate tags, metadata
3. Put object to destination bucket

Caveats for Replication

If you're replicating between two CORTX clusters (or within a single CORTX cluster), everything works great! 😁

Replicating to another S3 provider has some limitations:

- Some metadata (e.g. creation time) will not match the source object (S3 API doesn't give a way to change this)
- Version IDs will not match (S3 server chooses version IDs)

And of course, the source must be a CORTX cluster

How does CORTX do it?

Some hidden/non-standard S3 APIs allow CORTX to avoid these limitations:

- Management APIs
  - Allow direct access to metadata in Motr indices
  - Not user visible
  - Replicator can change the replication status to `COMPLETED` or `FAILED`
- Magic headers
  - On object put, check for special headers set by replicator (e.g. `x-stx-version-id`)
  - For normal users (not replicator account), these headers should be ignored

DATA IS POTENTIAL

Thank you!

Tim Shaffer
tim.shaffer@seagate.com

SEAGATE