Towards A Parallel and Restartable Data Transfer Mechanism in iRODS

Zoey Greer Jason Coposky Terrell Russell Hao Xu

June 5, 2018

Introduction

Current iRODS implementation supports limit parallel transfer and restart capability.
We introduce a design that extends current iRODS to support multiple tasks related to parallel transfer and restart in a unified, general solution.
We want to

- extend rather than completely rewrite the current iCAT.
- put, get, replication symmetrically.
- build API up from microservices.
- support parallel transfer
- support distributed storage of data.
- support partial replicas.
- support automatic restart.
- support partial synchronization.
- support distributed strorage of ICAT efficiently

The Design: Current

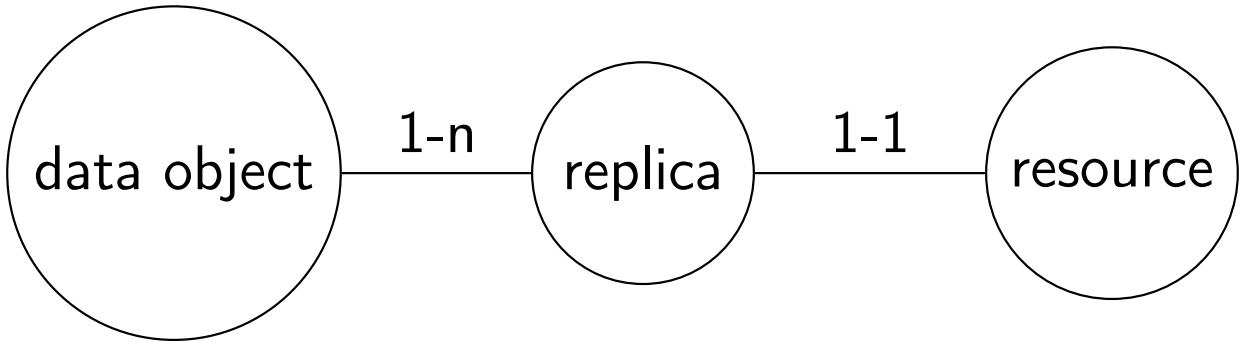


Figure: Entity-Relationship Diagram

The Design: Parallel and Restart

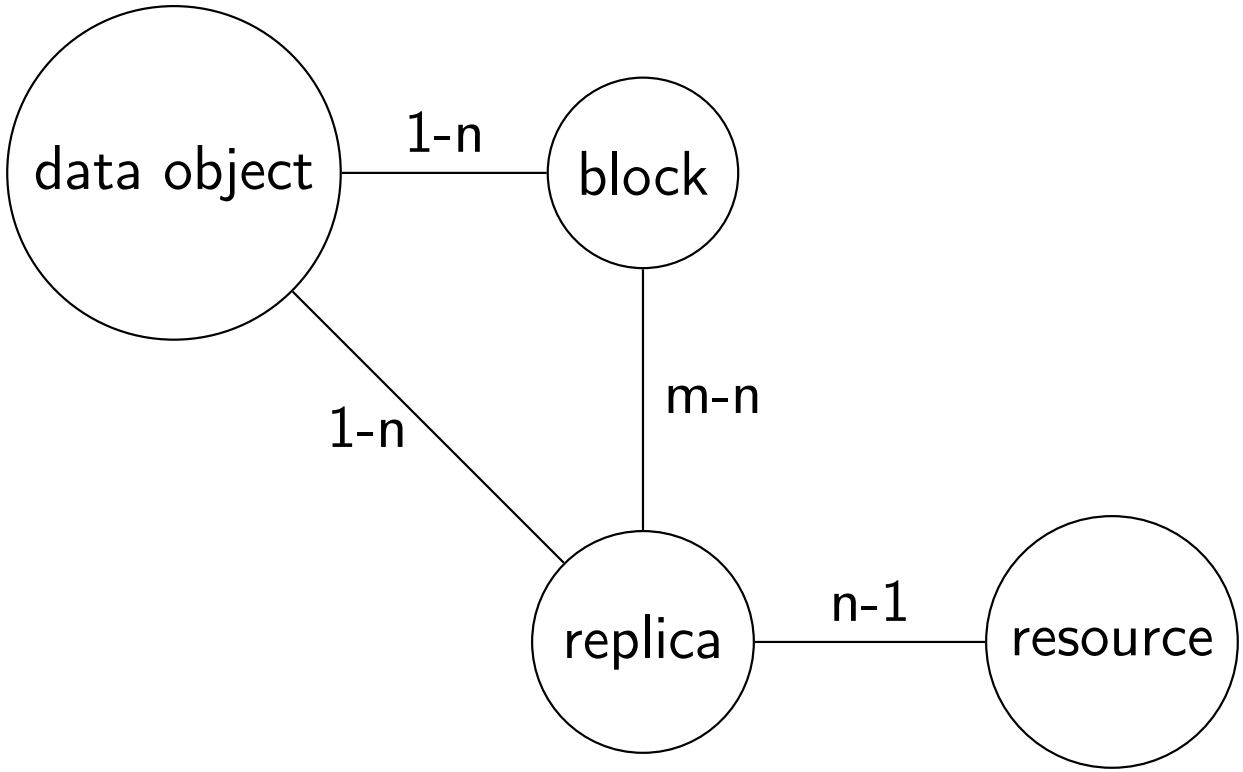


Figure: Entity-Relationship Diagram

Block Level

- Block level

	put	get
client to server	y	y
client to client	n	n
server to server	y	y/n

- Data Object level: put-get-replicate

Data Types

```
type Error
type Range -- = (Int, Bitmap)
type Block
type Data_object -- = (Path, Timestamp)
type Replica -- = (Data_object, Host, Replica_num)
```

block_put

Push a block to a resource using `block_put`. In the following, we use a default block size of 4MB.

```
block_put : (Replica, Range, [Block]) -> ()
```

This can be used in various operations.

data_object

The put operation is initiated by the client by the `data_object` operation.

```
data_object : Data_object -> [(Replica, Range)]
```

This request can be to any server.

`replica`

For each resource, the client start putting blocks into replicas using the `replica` operation.

```
replica : (Replica, Range) -> Range
```

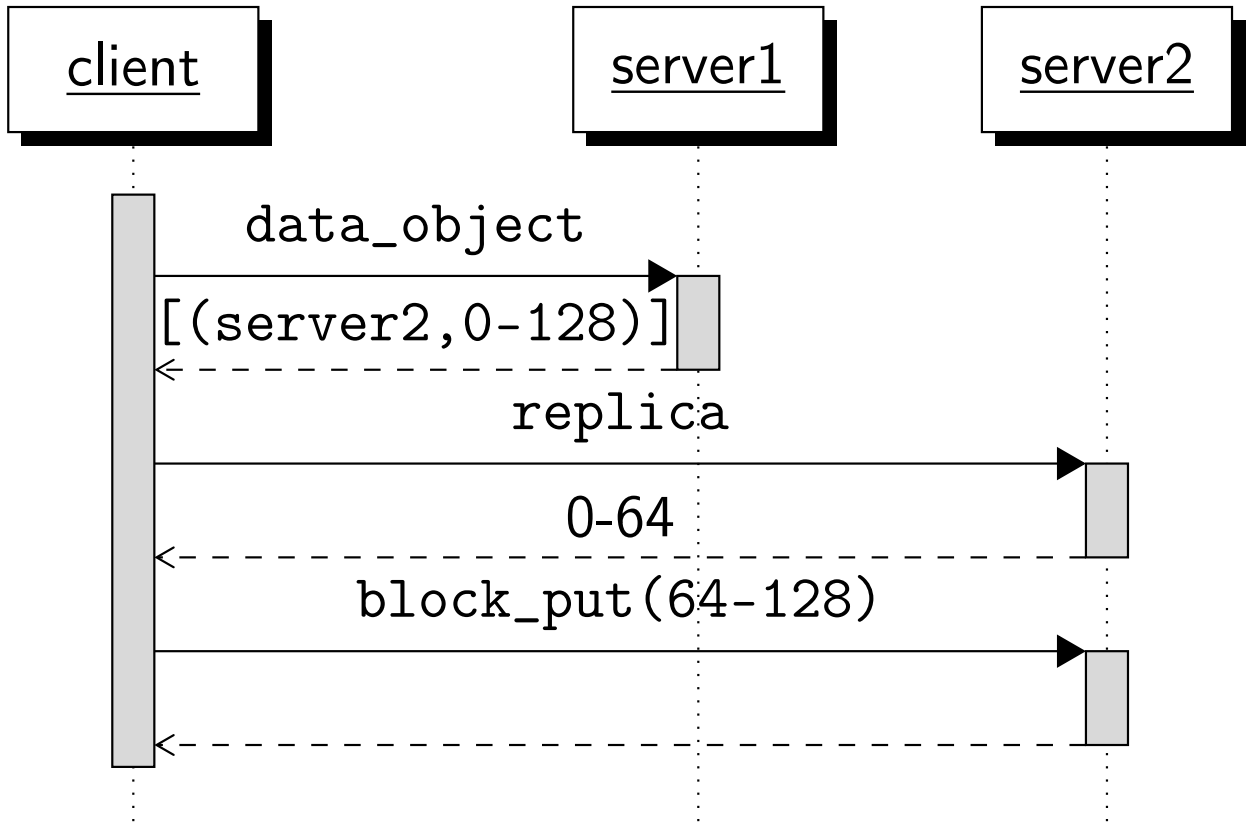
The returned range is a range of existing blocks on the resource in the input range. Based on returned range, the client sends the blocks to the resource.

block_get

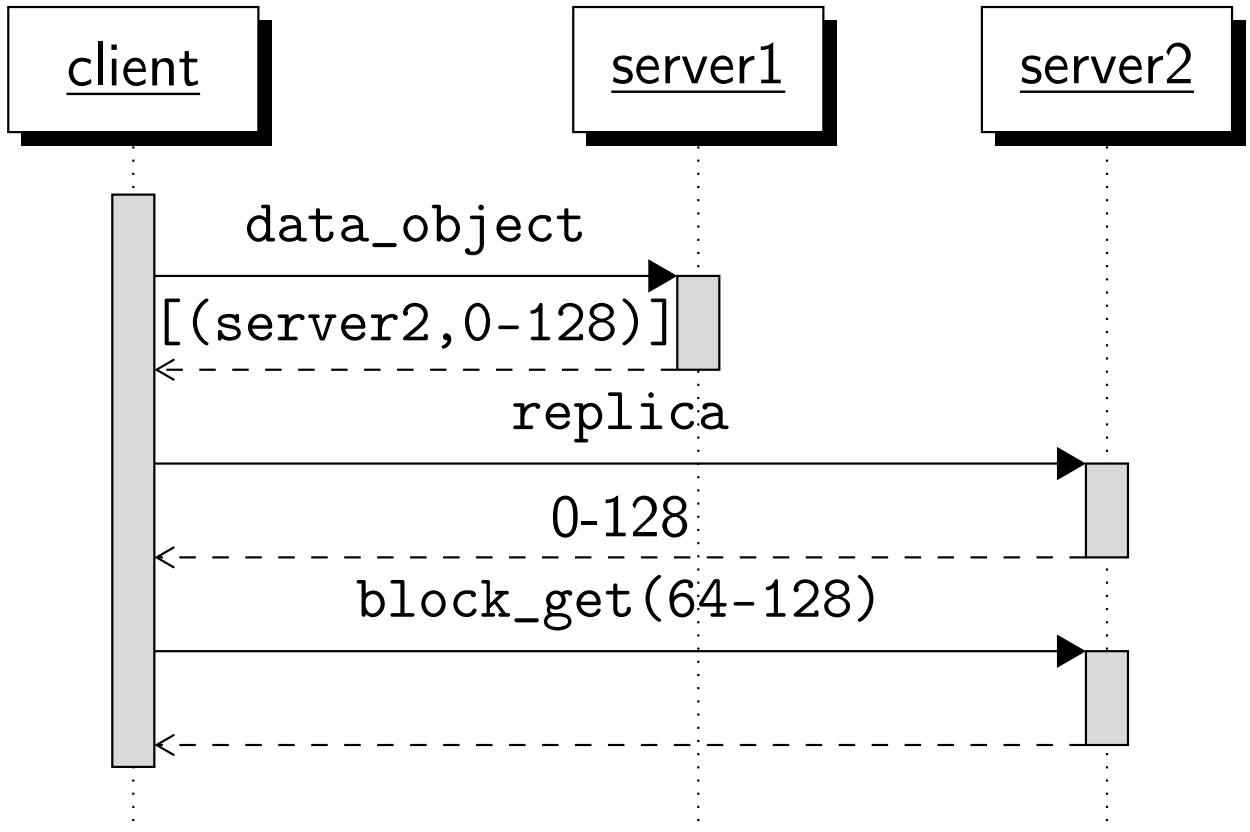
Pull a block from a resource using `block_get`.

```
block_get : (Replica, Range) -> [Block]
```

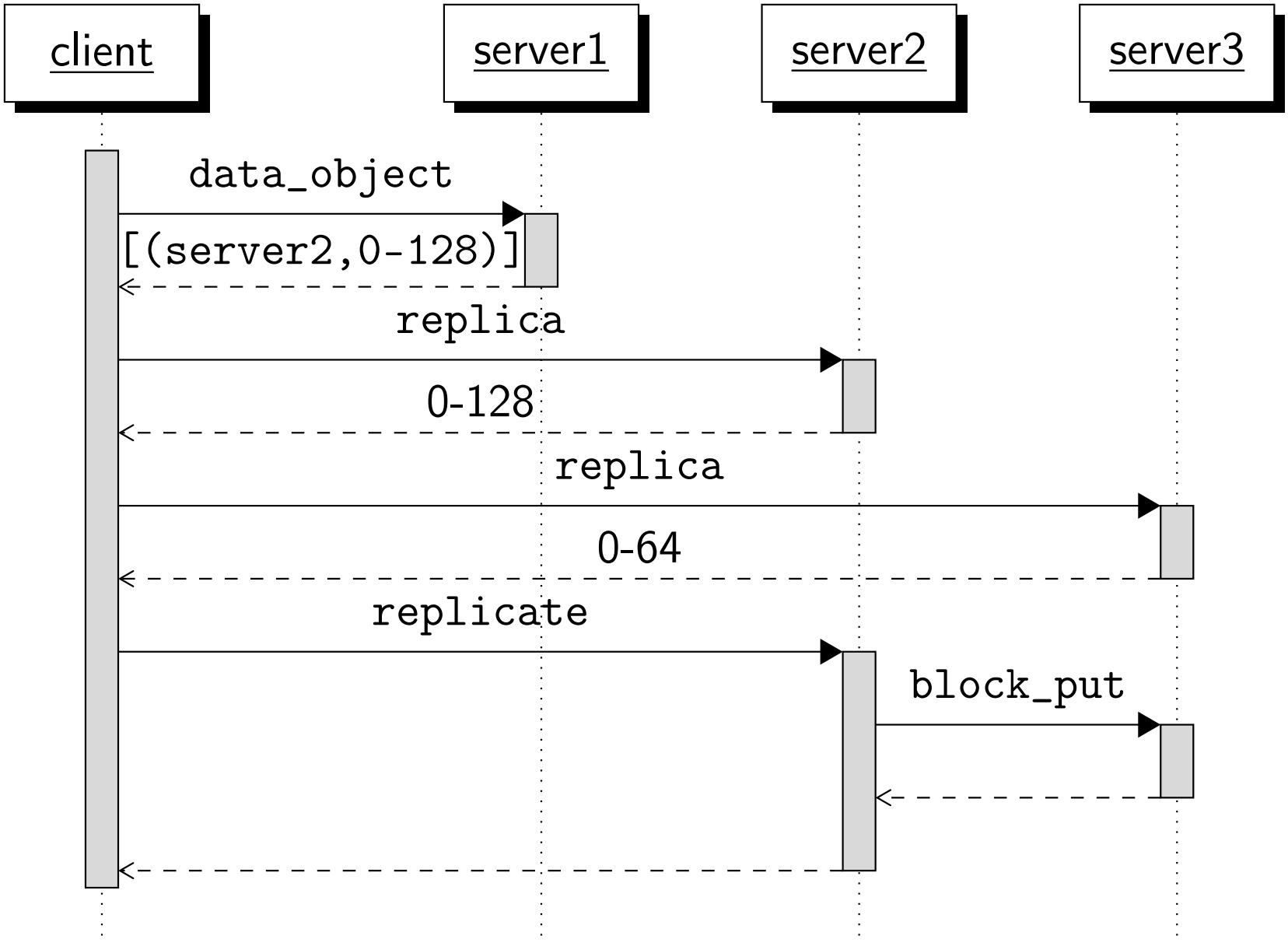
put


client
server1
server2
data_object
[(server2,0-128)]
replica
0-64
block_put(64-128)

get


client
server1
server2
data_object
[(server2,0-128)]
replica
0-128
block_get(64-128)

replicate


client
server1
server2
server3
data_object
[(server2,0-128)]
replica
0-128
replica
0-64
replicate
block_put

Storing incomplete replica

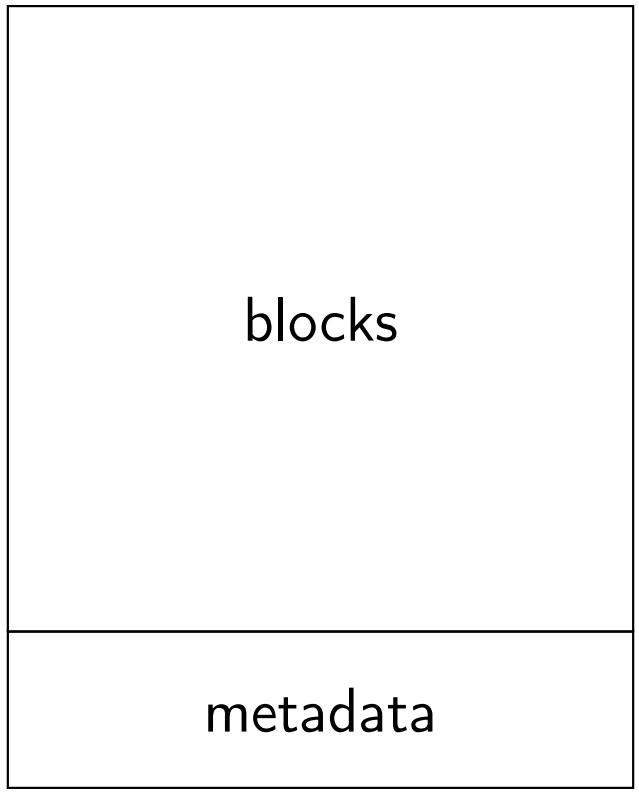


Figure: Incomplete replica

Metadata contain `Replica` and `Range` of available blocks

Parallel put

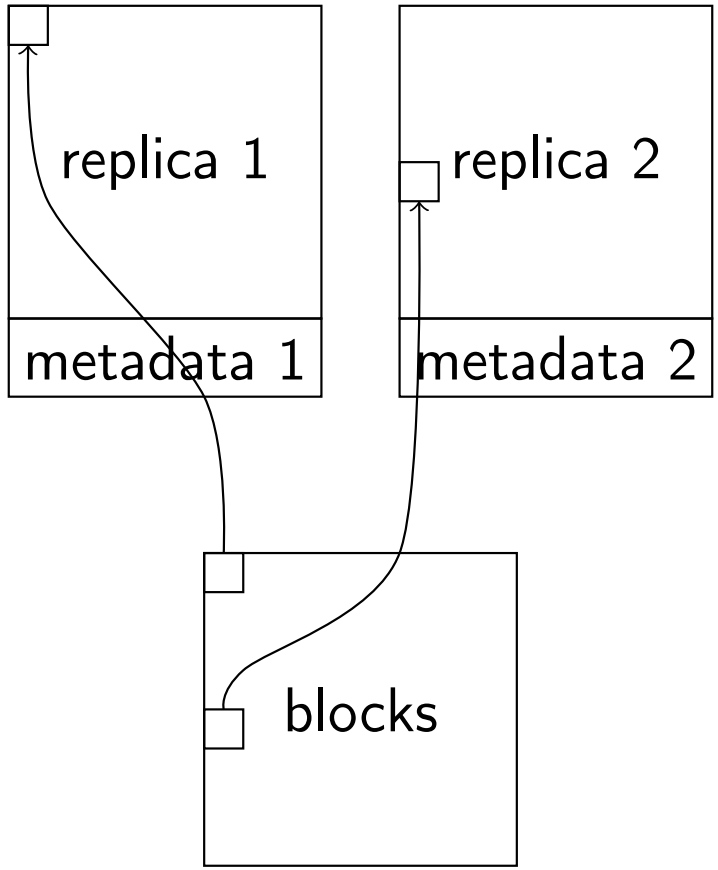


Figure: Multi-part put

Parallel get

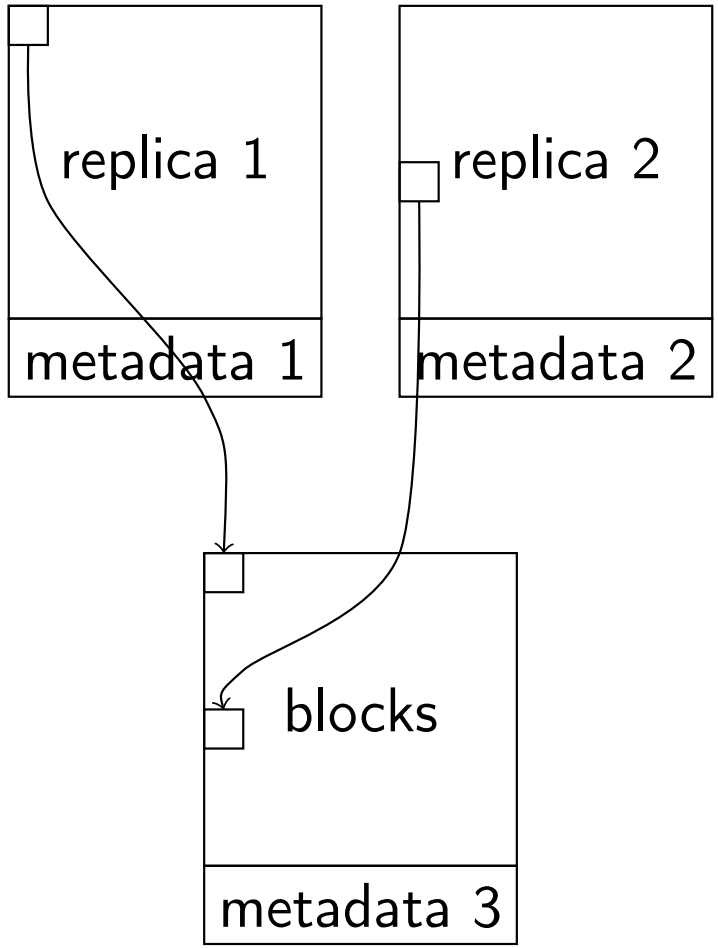


Figure: Multi-part get