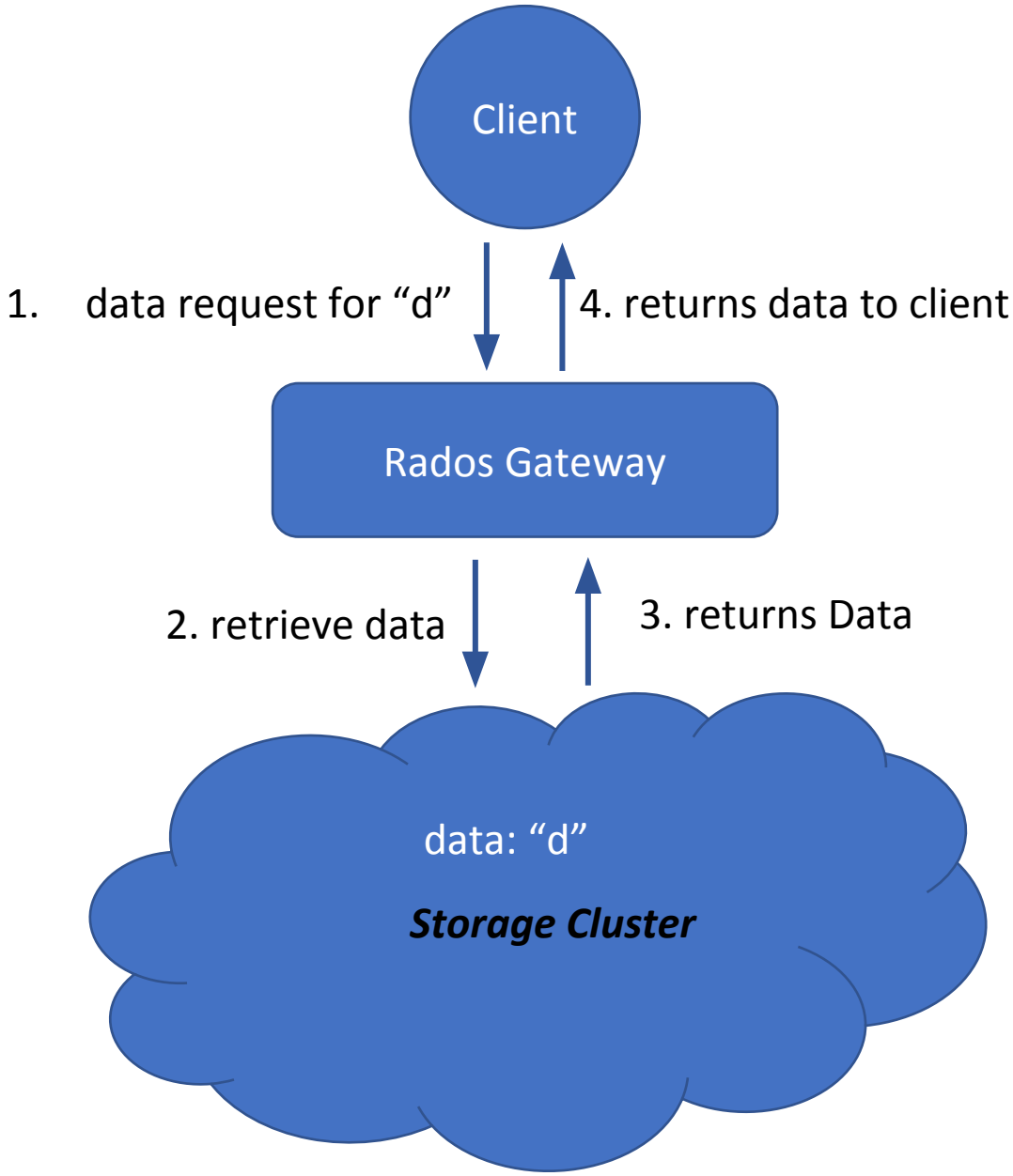
Ceph RGW-Prefetching

Ali R., Amin M., Bissenbay D.

What is CEPH?

- Distributed Storage System
- **Key Components** [Related to Project]
  - Client (Applications)
  - Rados Gateway
  - Storage Cluster



Prefetching in Ceph

- Current caching scheme is re-active
- Goal: Improve performance in first access
  - Lower latency
  - Higher throughput
- Two type of prefetching
  - on-demand prefetching
  - pro-active prefetching in rgw

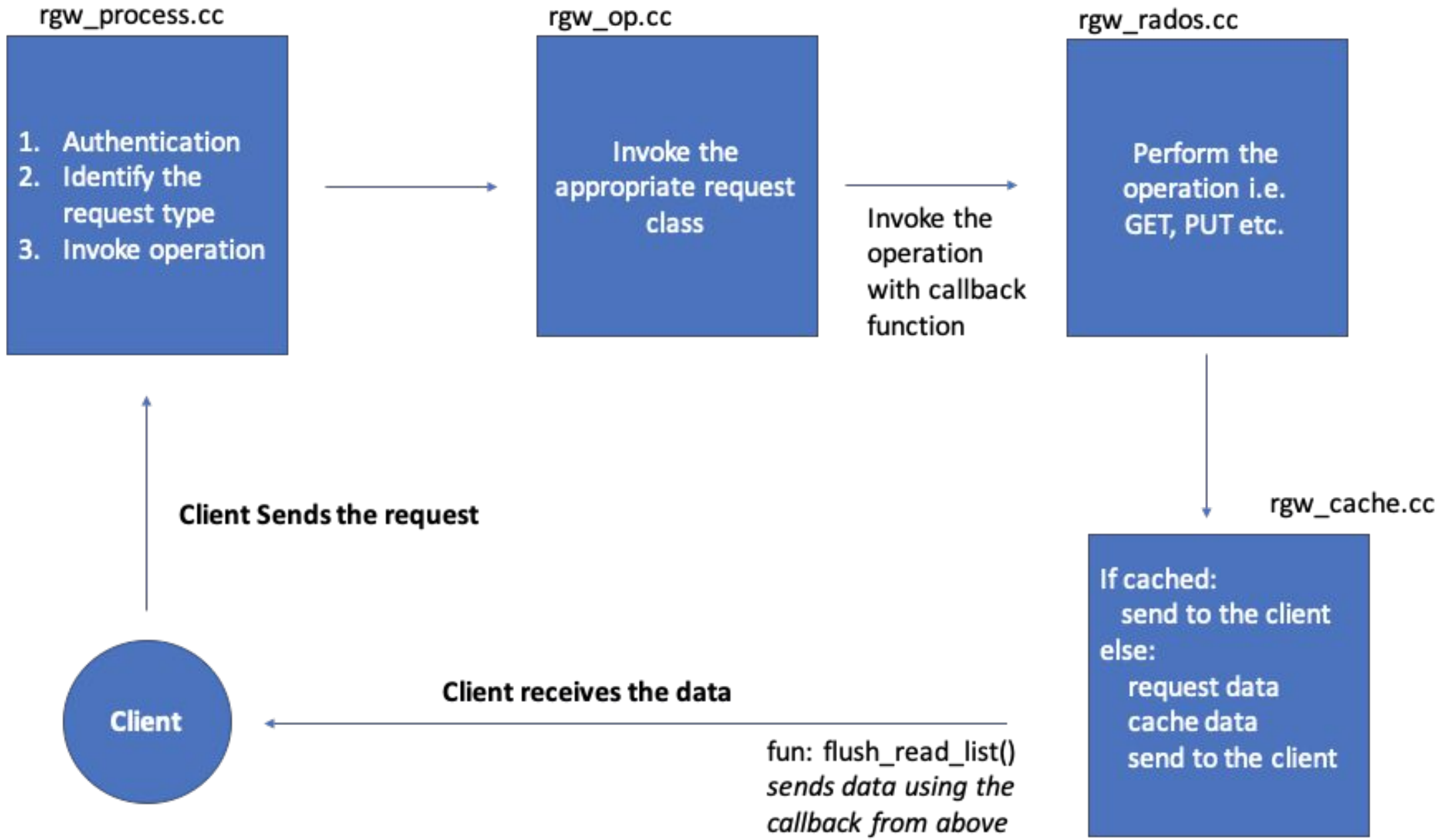
Until Sprint 3

- Uploaded & retrieved files to Ceph storage
  - S3cmd
  - Swift
  - Boto3
- Understanding the code
  - Deployed Ceph on multiple machines
  - Data flow
  - Debugging the code with GDB
  - Brainstorming with mentors to find the best development strategy
- In Progress:
  - Prefetching based on user request
  - Pro-active prefetching on block level
  - Evaluation

Sprint 4

- Done:
  - Prefetching based on user request
  - Pro-active prefetching on block level
- In progress:
  - Full testing of the above prefetching code
  - Evaluation (in terms of performance)
- Stretching the Goals:
  - Implement APIs to monitor the cache

Prefetching Based on User Request


rgw_process.cc
1. Authentication
2. Identify the request type
3. Invoke operation
rgw_op.cc
Invoke the appropriate request class
Invoke the operation with callback function
rgw_rados.cc
Perform the operation i.e. GET, PUT etc.
rgw_cache.cc
If cached:
send to the client
else:
request data
cache data
send to the client
Client Sends the request
Client
Client receives the data
fun: flush_read_list()
sends data using the callback from above

Prefetching based on user request *[approach 1]*

- Overload the GET request (add header)
- Catch the header in rgw_rados.cc
  - Instead of sending back the data, place it in the cache
  - Reply the user with normal response headers (200 OK etc. and Content-length: 0)
- Issue with this approach
  - async data read requests get cancelled

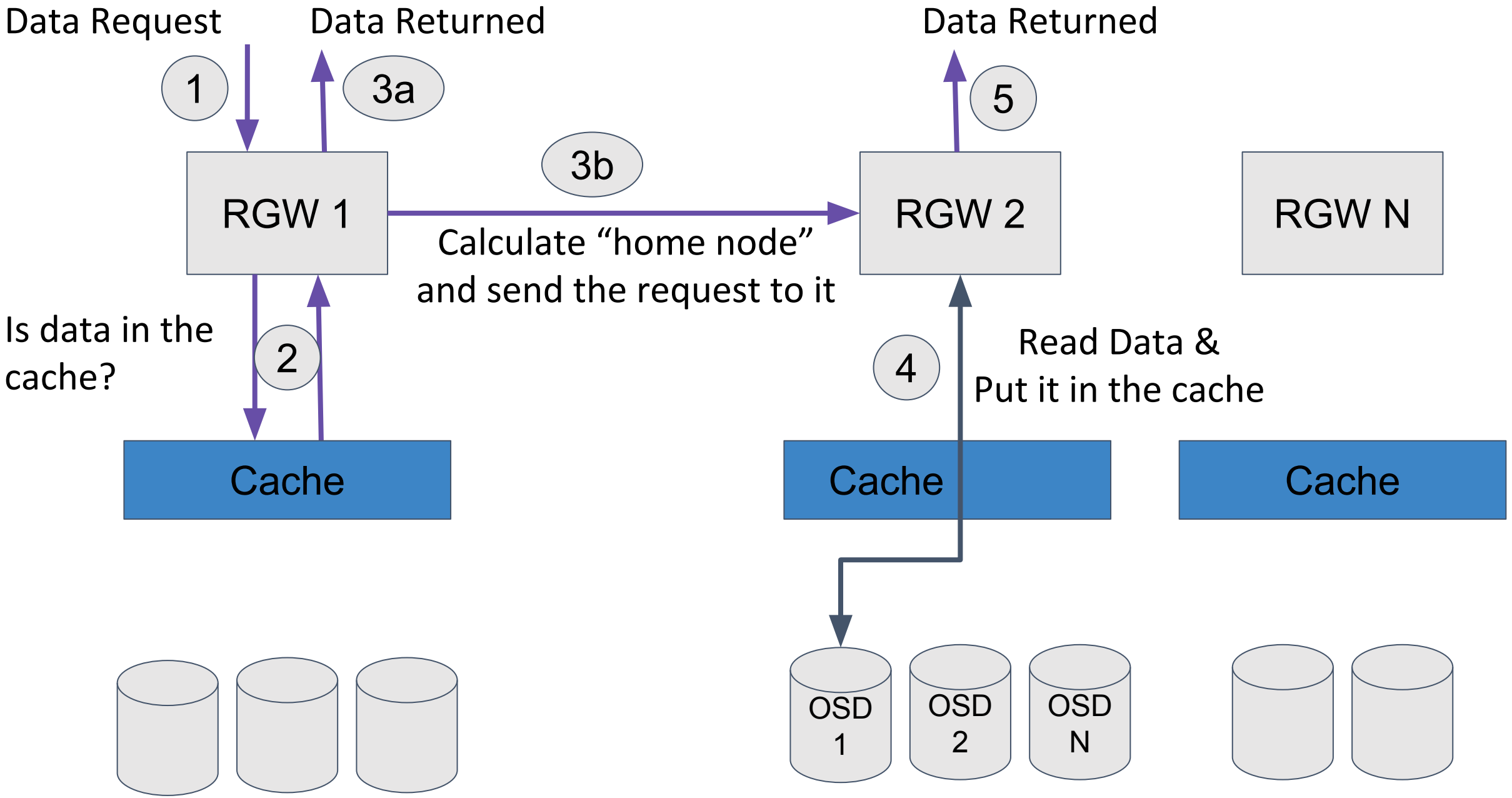
Prefetching based on user request *[approach 2]*

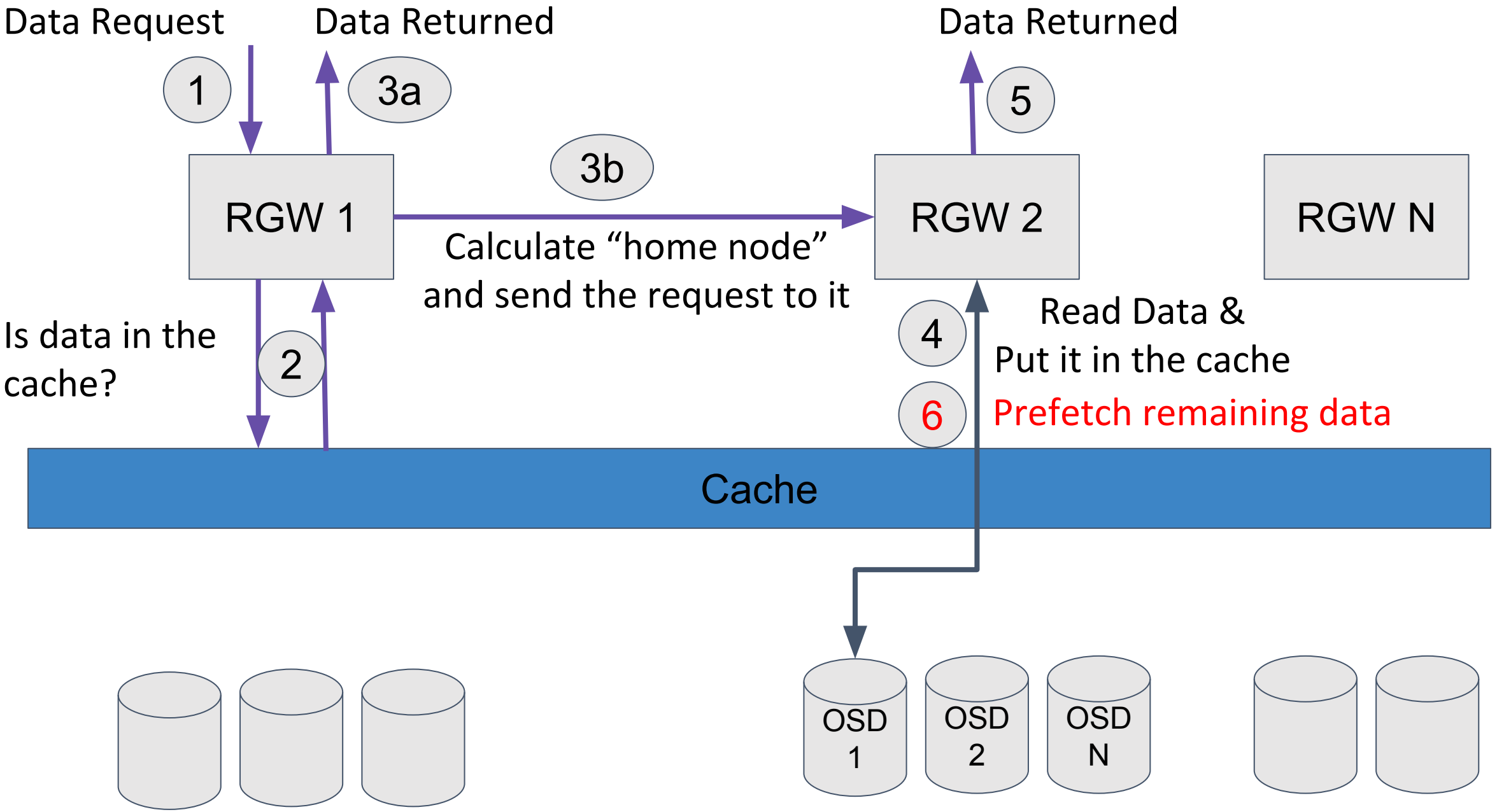
- Overload the GET request (add header)
- Catch the header in rgw_rados.cc
  - Instead of sending back the data, place it in the cache
  - **Wait for all of the data to be written in the cache and then**
  - Reply the user with normal response headers (200 OK etc. and Content-length: **x bytes**)
  -
- Issue with this approach
  - It’s a blocking call

What’s next for “on-demand prefetching”

- ❏ Talked to our mentors
- ❏ Approach 1 is more logical
- ❏ We will find a way that data-request are not cancelled even though client closes the connection

Pro-active Prefetching


Data Request
Data Returned
Data Returned
1
3a
5
3b
RGW 1
RGW 2
RGW N
Calculate “home node”
and send the request to it
Is data in the
cache?
2
4
Read Data &
Put it in the cache
Cache
Cache
Cache
OSD
1
OSD
2
OSD
N


Data Request
Data Returned
Data Returned
1
3a
5
3b
RGW 1
RGW 2
RGW N
Calculate “home node”
and send the request to it
Is data in the
cache?
2
4
Read Data &
Put it in the cache
6
Prefetch remaining data
Cache
OSD
1
OSD
2
OSD
N

Pro-active prefetching

- What do we need?
  - What data?
  - Where is it?
  - What is the size of it?
  - What chunk has the user read?
  - We should not block other tasks
  - We should not harm the performance

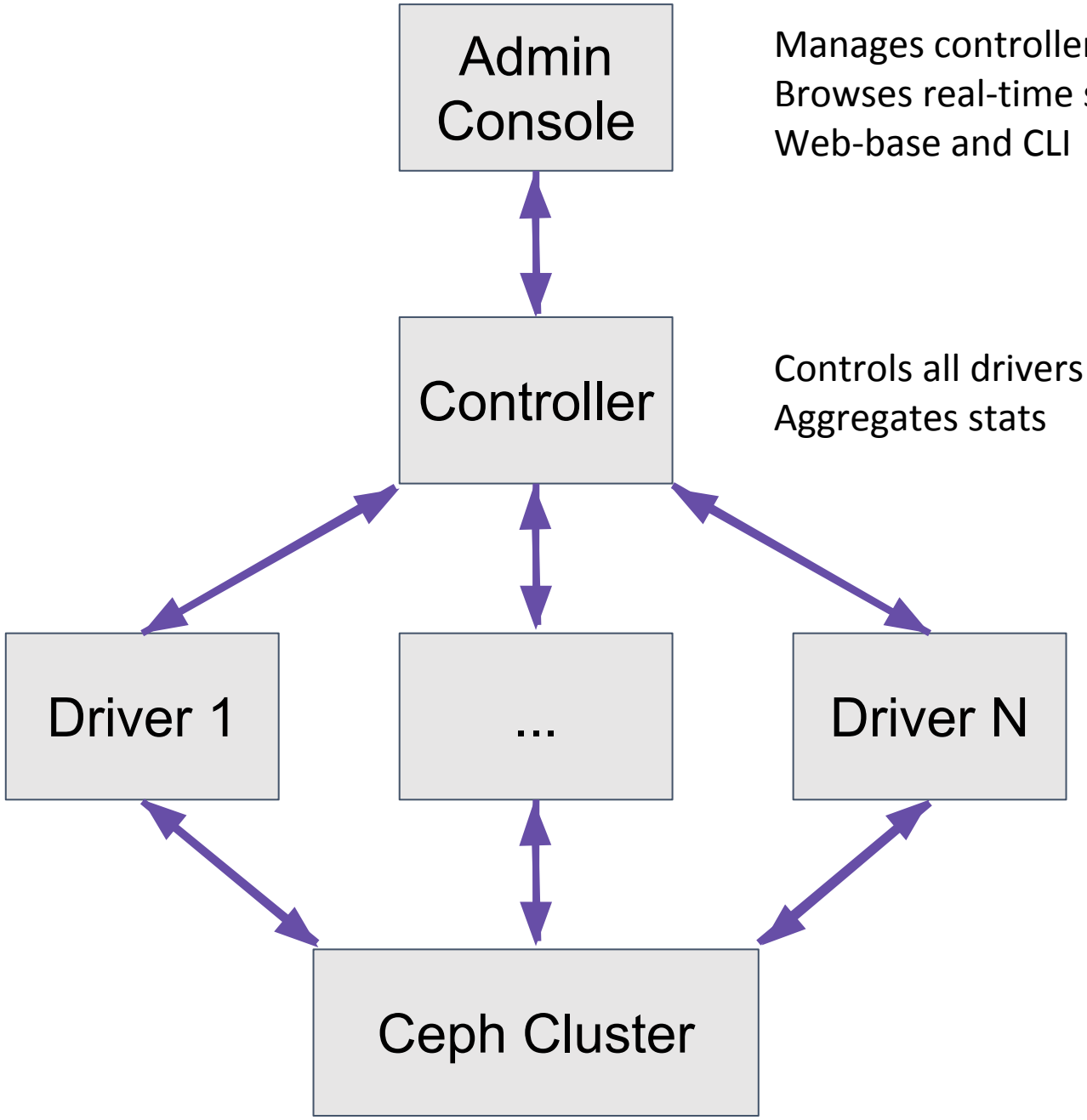
Pro-active prefetching

- A prefetching C++ class to talk with cache
  - We have implemented:
    - The requested File, size of requested chunk and its offsets
    - The size of the File and passing it to the prefetching class
    - User’s authentication
- A thread pool for prefetching
- Adding prefetching task to the thread pool

Evaluation

- ❏ COSBench
- ❏ Modifying COSBench

COSBench



Manages controller
Browses real-time stats
Web-base and CLI

Controls all drivers
Aggregates stats

Generates load

Workload Configuration

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<workload name="s3-sample" description="sample benchmark for s3">

  <storage type="s3" config="accesskey=;secretkey=;endpoint=http://localhost:8000" />

  <workflow>

    <workstage name="init">
      <work type="init" workers="1" config="cprefix=s3testqwer;containers=r(1,2)" />
    </workstage>

    <workstage name="prepare">
      <work type="prepare" workers="1" config="cprefix=s3testqwer;containers=r(1,2);objects=r(1,10);sizes=c(64)KB" />
    </workstage>

    <workstage name="main">
      <work name="main" workers="1" runtime="30">
        <operation type="read" ratio="50" config="cprefix=s3testqwer;containers=u(1,2);objects=u(1,10)" />
        <operation type="write" ratio="50" config="cprefix=s3testqwer;containers=u(1,2);objects=u(11,20);sizes=c(64)KB" />
      </work>
    </workstage>

    <workstage name="cleanup">
      <work type="cleanup" workers="1" config="cprefix=s3testqwer;containers=r(1,2);objects=r(1,20)" />
    </workstage>

    <workstage name="dispose">
      <work type="dispose" workers="1" config="cprefix=s3testqwer;containers=r(1,2)" />
    </workstage>

  </workflow>

</workload>
```

Flexible load control

Object size distribution

Read/Write Operations

```xml
<operation type="prefetch" ratio="50" config="cprefix=s3testqwer;containers=u(1,2);objects=u(1,10)" />
```

Evaluation

- Done:
  - Installed COSBench on the machine
  - Run a testing workload
  - Agreed on design change of a xml config file with mentors
- In progress:
  - Setting Up Dev Environment in Eclipse to make changes in COSBench
  - Creating a Prefetching Operator which will send headers on GET
- Next Step:
  - Come up with a workload design that will be used to test prefetching
  - Try different workloads to identify bottlenecks

Burndown chart