CEPHALOCON APAC 2018
THE FUTURE OF STORAGE
22-23 March 2018 | BEIJING

Experiences building a distributed shared-log on RADOS

Noah Watkins
UC Santa Cruz
@noahdesu

About me

- Graduate student
- UC Santa Cruz

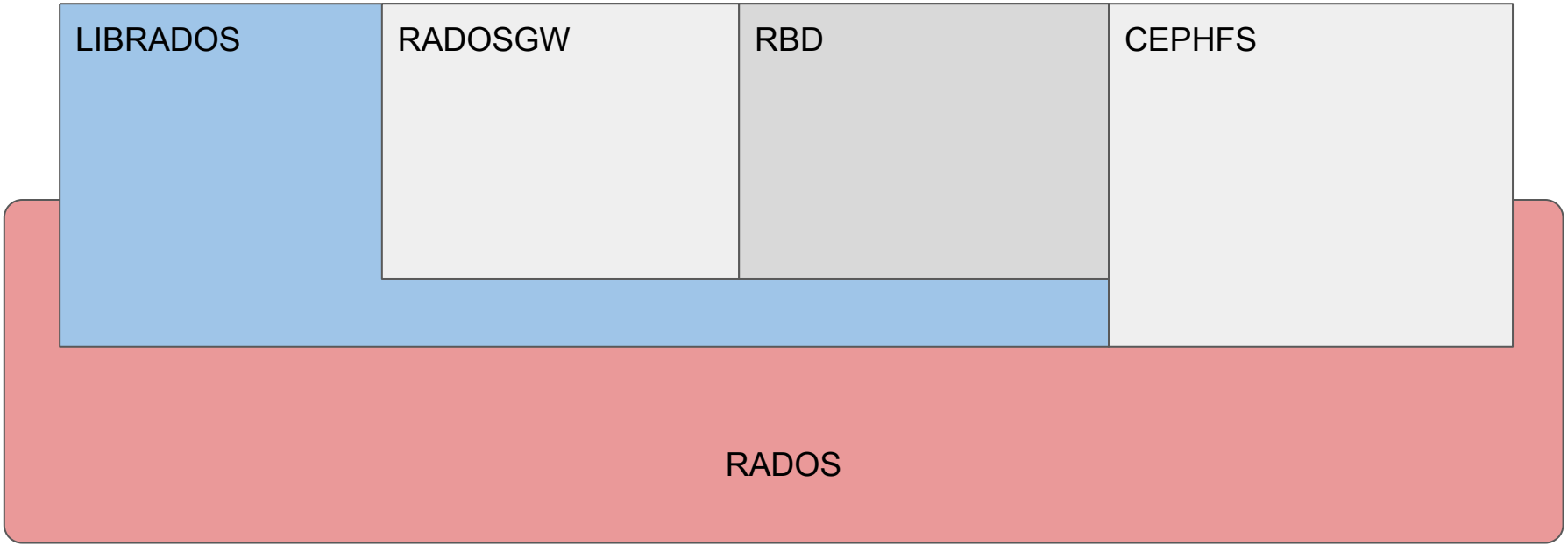

About me
Graduate student
UC Santa Cruz
Data management
Storage systems
High-performance computing
Quality of service
Ceph中国社区
IT大咖说
知识共享平台

About me

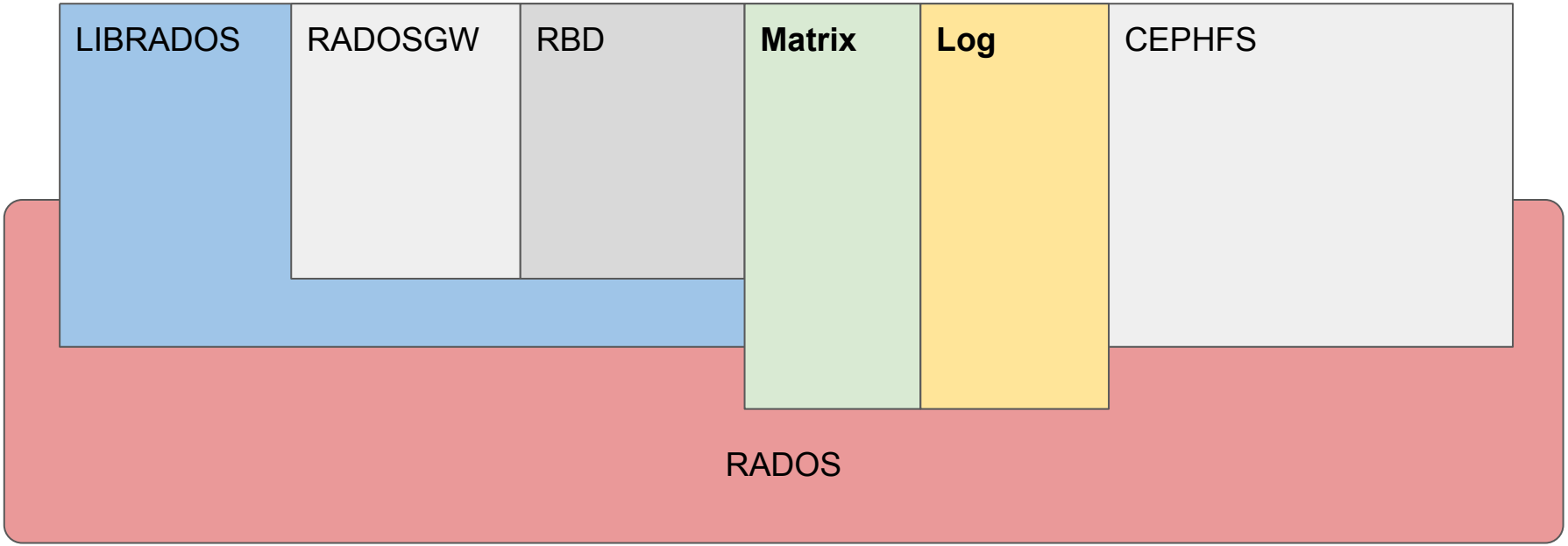
- Graduate student
- UC Santa Cruz

- Data management
- Storage systems
- High-performance computing
- Quality of service
- Ceph shop for prototyping

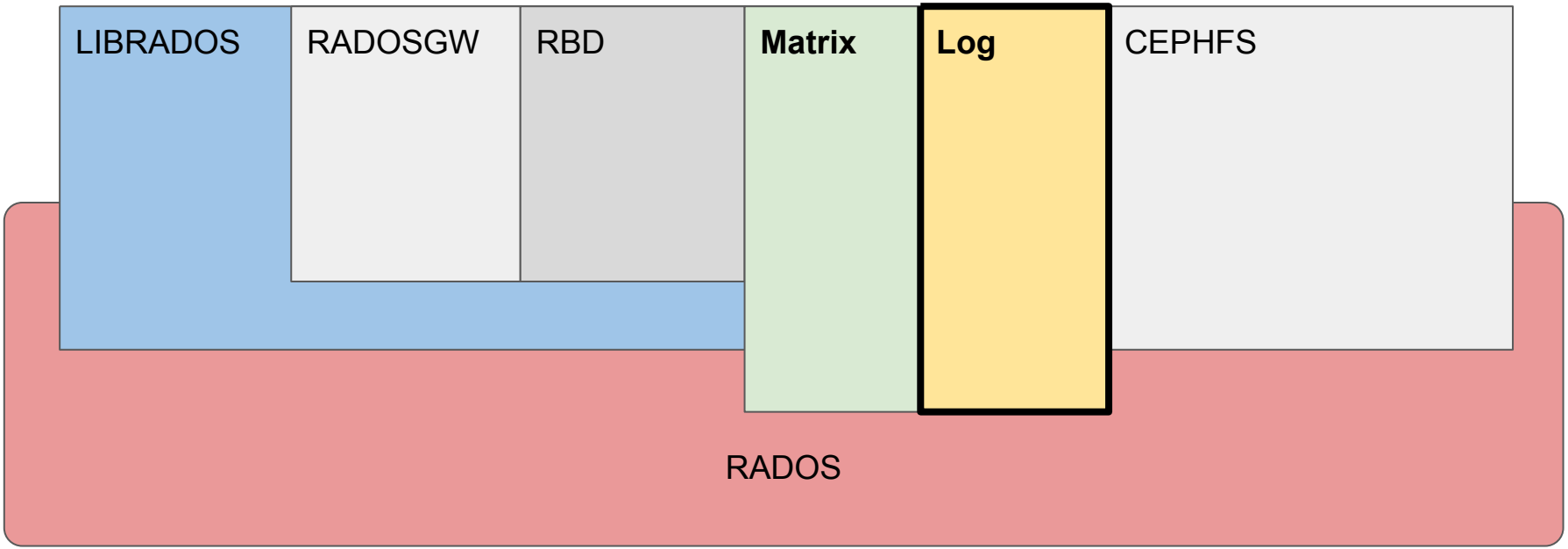
Ceph storage interfaces, a familiar sight...



Our research: programmability in storage



Our research: programmability in storage



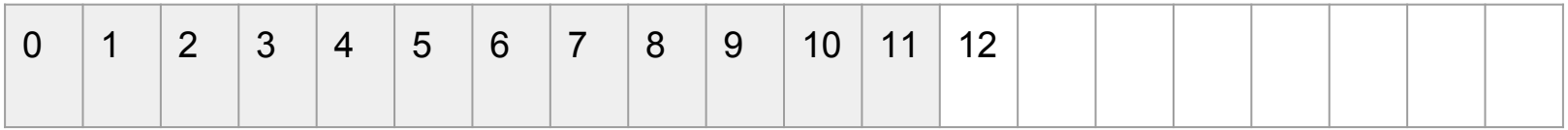
The agenda

- Why should I care about logs?
- How to build a really, really fast log
- Mapping techniques for fast logs onto Ceph
- A few usage examples

Why you should care about logs

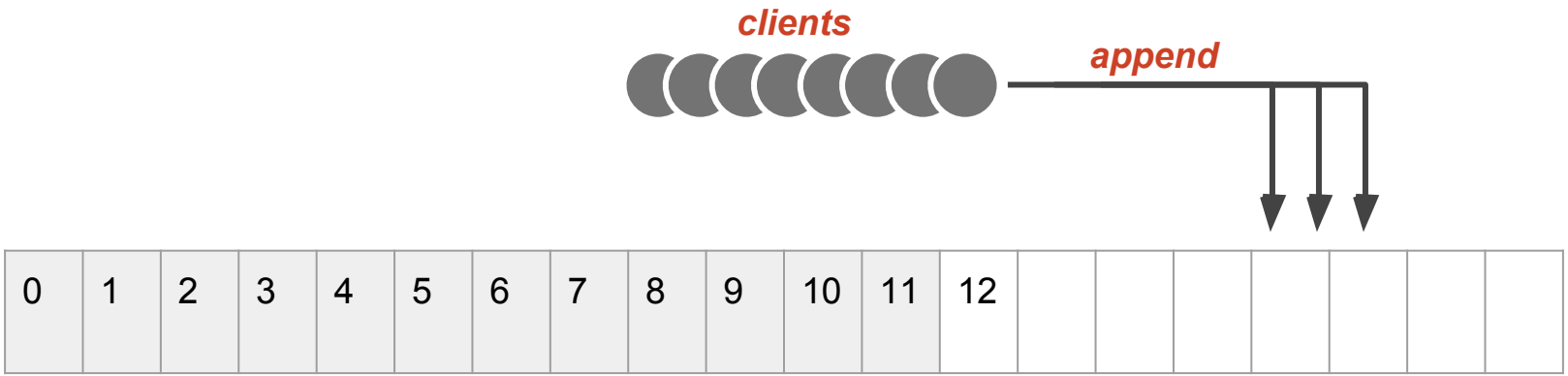
A log is an ordered list of data elements

- General abstract form of a log
- Ordered list of elements



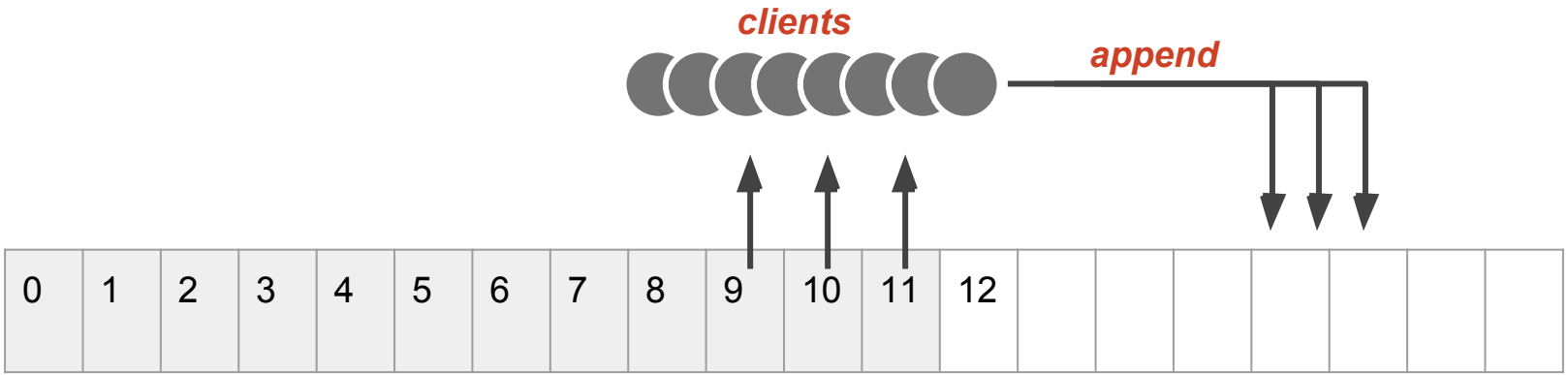
A log is an ordered list of data elements

- General abstract form of a log
- Ordered list of elements
- New entries are appended to the log



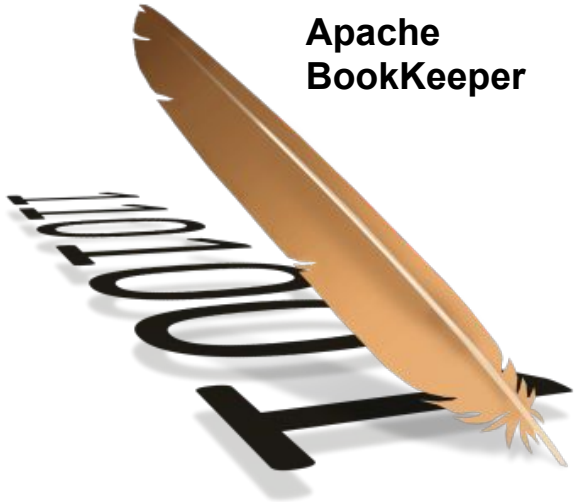
A log is an ordered list of data elements

- General abstract form of a log
- Ordered list of elements
- New entries are appended to the log
- Log entries can be read directly



Everyone is going bananas for logs

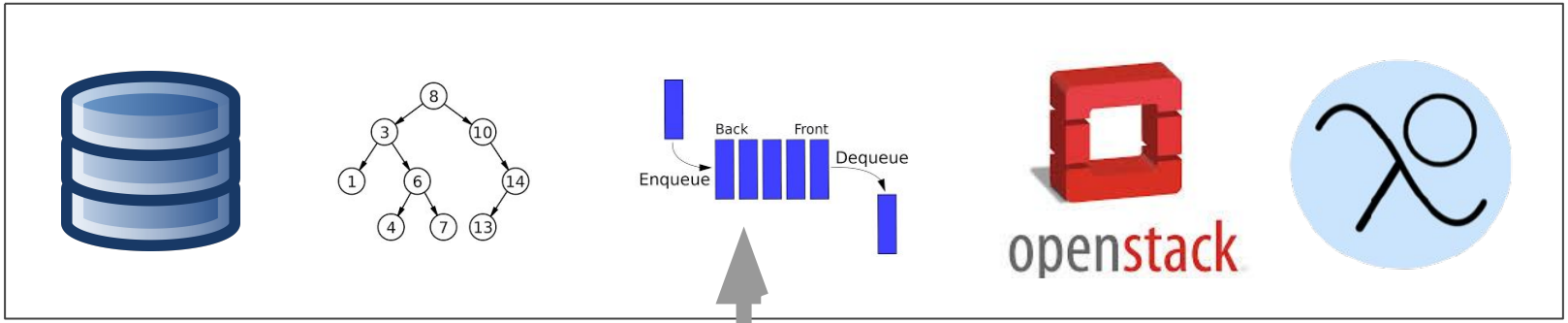
- Very high throughput logs
- No global ordering



- Strongly consistent, global ordering
- Write batching
- Single writer

Strongly consistent shared-log

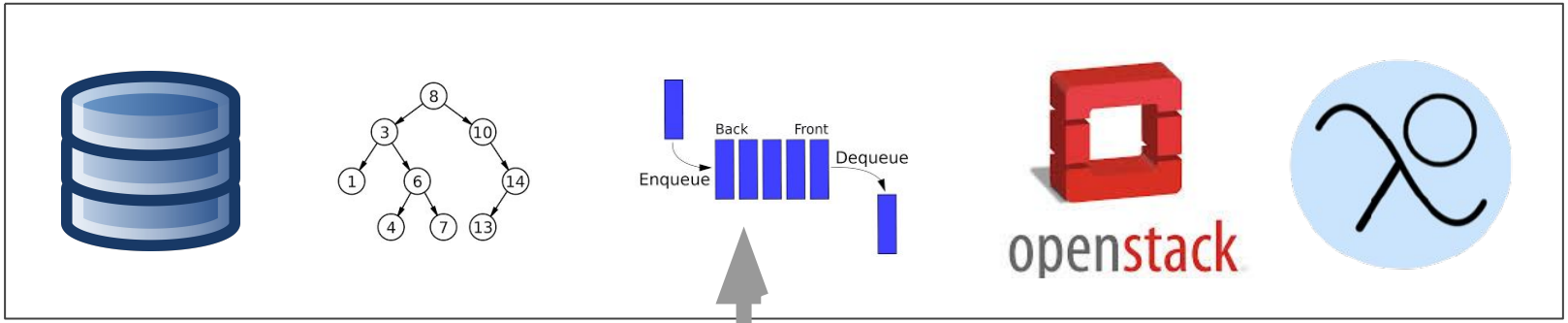
Database management systems *Replicated state machines* *Cloud-based metadata services*



0	1	2	3	4	5	6	7	8	9	10	11	12							

Strongly consistent, **high-performance** shared-log

Database management systems *Replicated state machines* *Cloud-based metadata services*



0	1	2	3	4	5	6	7	8	9	10	11	12							

Shared-logs are **challenging** to scale

Balakrishnan et al., "*Tango: Distributed Data Structures over a Shared Log*", SOSP '13

se a par-
r, a tree)
of allow-
ures that
the C++
ons came
problem-
hich use
ic work-
at stores
ficient to

vanished with the advent of flash drives that can support thousands of concurrent read and write IOPS.

The second problem with the shared log abstraction relates to scalability; existing implementations typically require appends to the log to be serialized through a primary server, effectively limiting the append throughput of the log to the I/O bandwidth of a single machine. This problem is eliminated by the CORFU protocol [10], which scales the append throughput of the log to the speed at which a centralized sequencer can hand out new offsets in the log to clients.

Shared-logs are **challenging** to scale

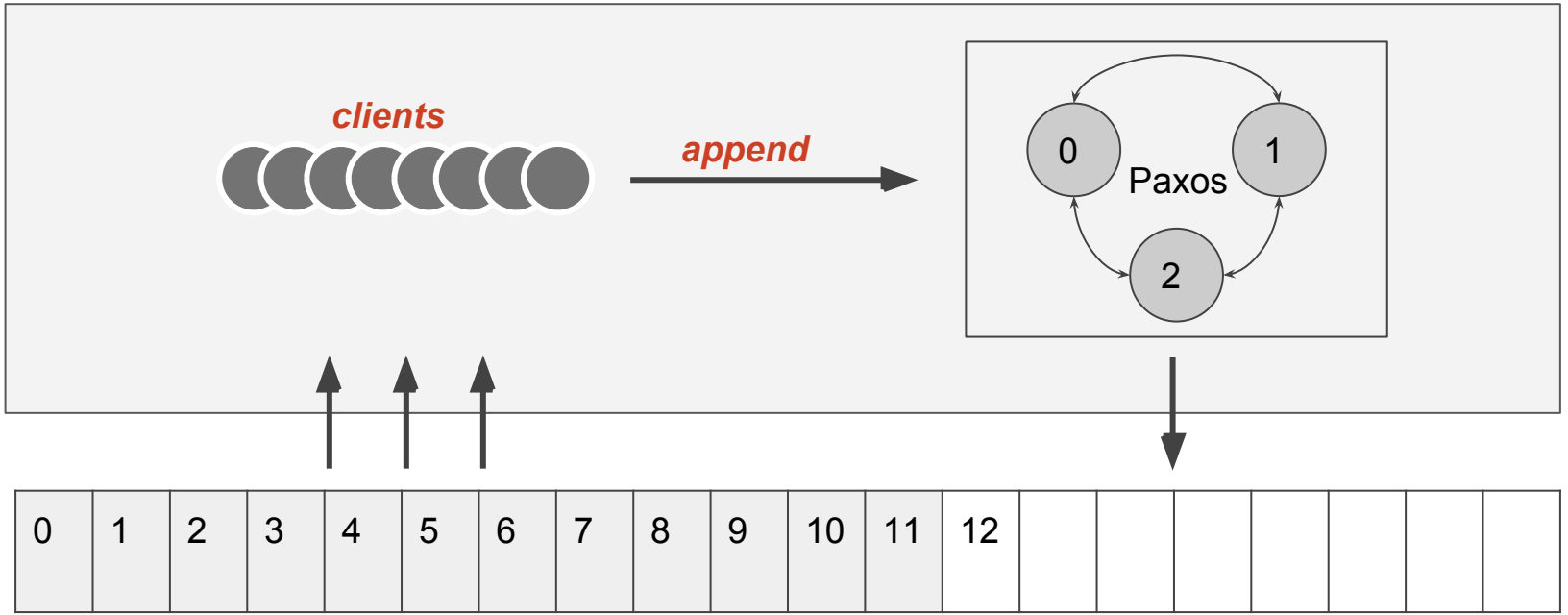
Balakrishnan et al., "*Tango: Distributed Data Structures over a Shared Log*", SOSP '13

se a par-
r, a tree)
of allow-
ures that
the C++
ons came
problem-
hich use
ic work-
at stores
ficient to

vanished with the advent of flash drives that can support thousands of concurrent read and write IOPS.

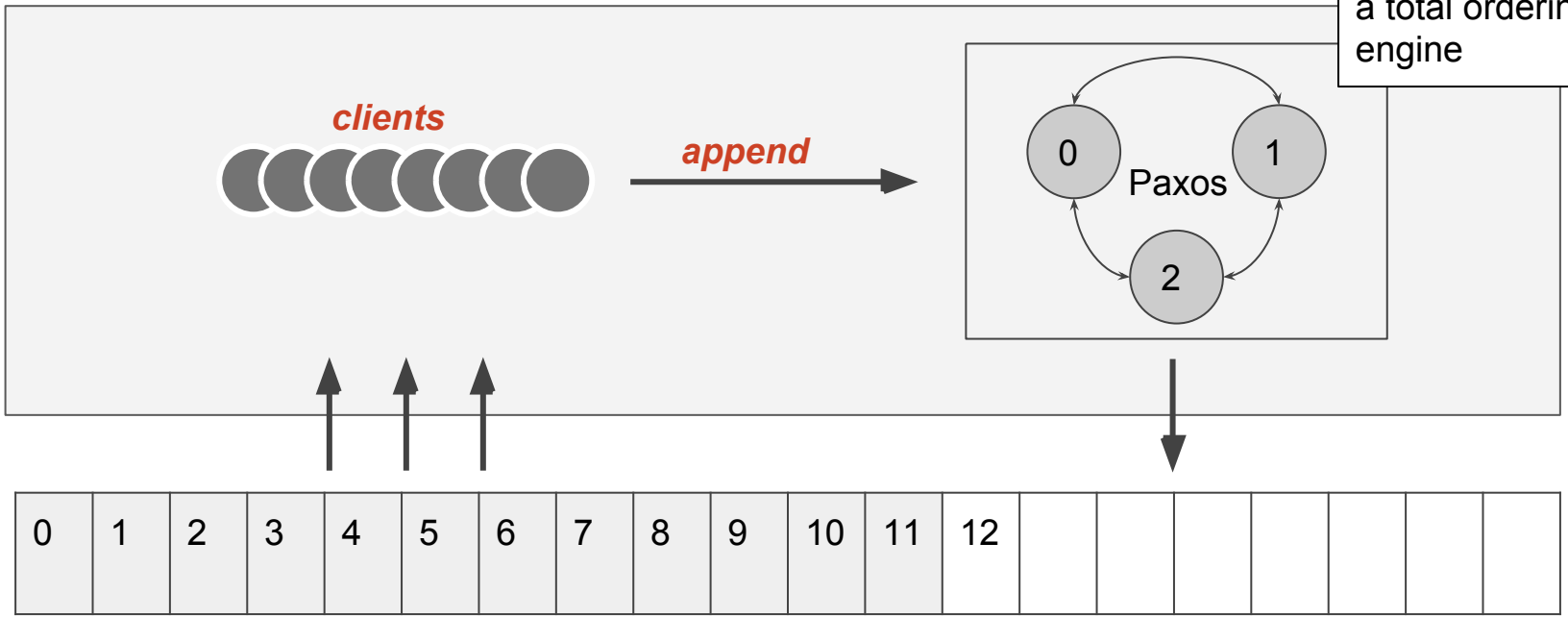
The second problem with the shared log abstraction relates to scalability; existing implementations typically require appends to the log to be serialized through a primary server, effectively limiting the append throughput of the log to the I/O bandwidth of a single machine. This problem is eliminated by the CORFU protocol [10], which scales the append throughput of the log to the speed at which a centralized sequencer can hand out new offsets in the log to clients.

Shared-logs are **challenging** to scale



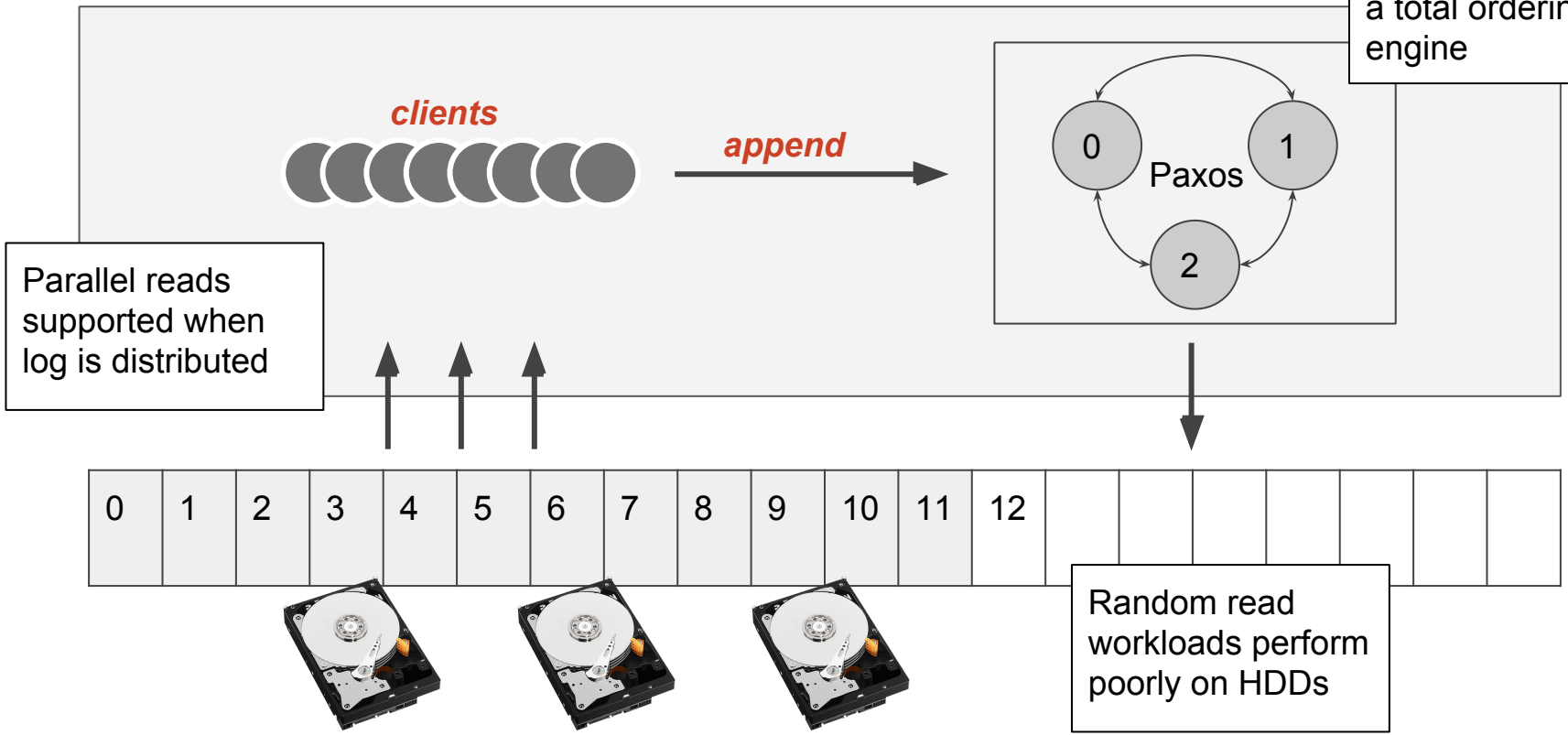
Shared-logs are **challenging** to scale

Writes are funneled through a total ordering engine



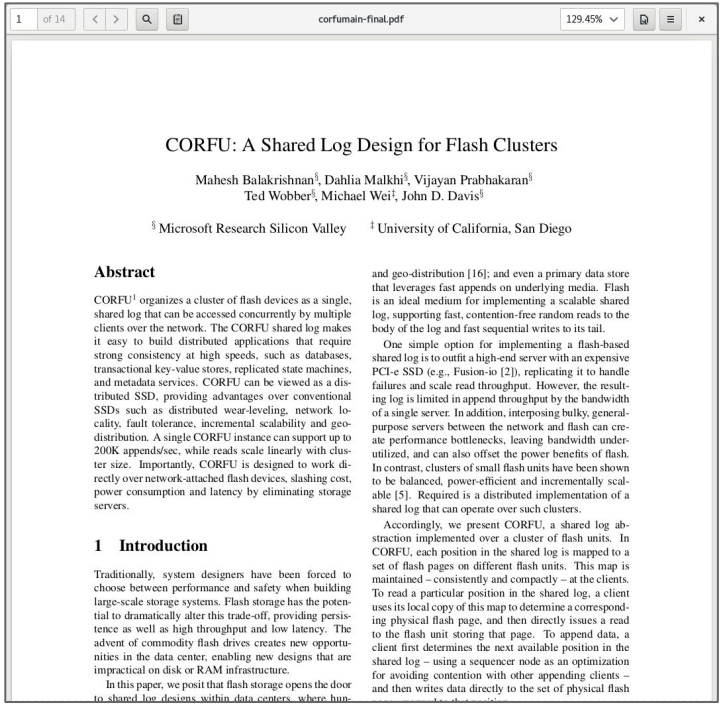
Shared-logs are **challenging** to scale

Writes are funneled through a total ordering engine



CORFU: A Shared Log Design for Flash Clusters

Balakrishnan, et. al, NSDI 2011



1 of 14 corfumain-final.pdf 129.45%

CORFU: A Shared Log Design for Flash Clusters

Mahesh Balakrishnan[§], Dahlia Malkhi[§], Vijayan Prabhakaran[§]
Ted Wobber[§], Michael Wei[‡], John D. Davis[§]

[§] Microsoft Research Silicon Valley [‡] University of California, San Diego

Abstract

CORFU[1] organizes a cluster of flash devices as a single, shared log that can be accessed concurrently by multiple clients over the network. The CORFU shared log makes it easy to build distributed applications that require strong consistency at high speeds, such as databases, transactional key-value stores, replicated state machines, and metadata services. CORFU can be viewed as a distributed SSD, providing advantages over conventional SSDs such as distributed wear-leveling, network locality, fault tolerance, incremental scalability and geo-distribution. A single CORFU instance can support up to 200K appends/sec, while reads scale linearly with cluster size. Importantly, CORFU is designed to work directly over network-attached flash devices, slashing cost, power consumption and latency by eliminating storage servers.

1 Introduction

Traditionally, system designers have been forced to choose between performance and safety when building large-scale storage systems. Flash storage has the potential to dramatically alter this trade-off, providing persistence as well as high throughput and low latency. The advent of commodity flash drives creates new opportunities in the data center, enabling new designs that are impractical on disk or RAM infrastructure.

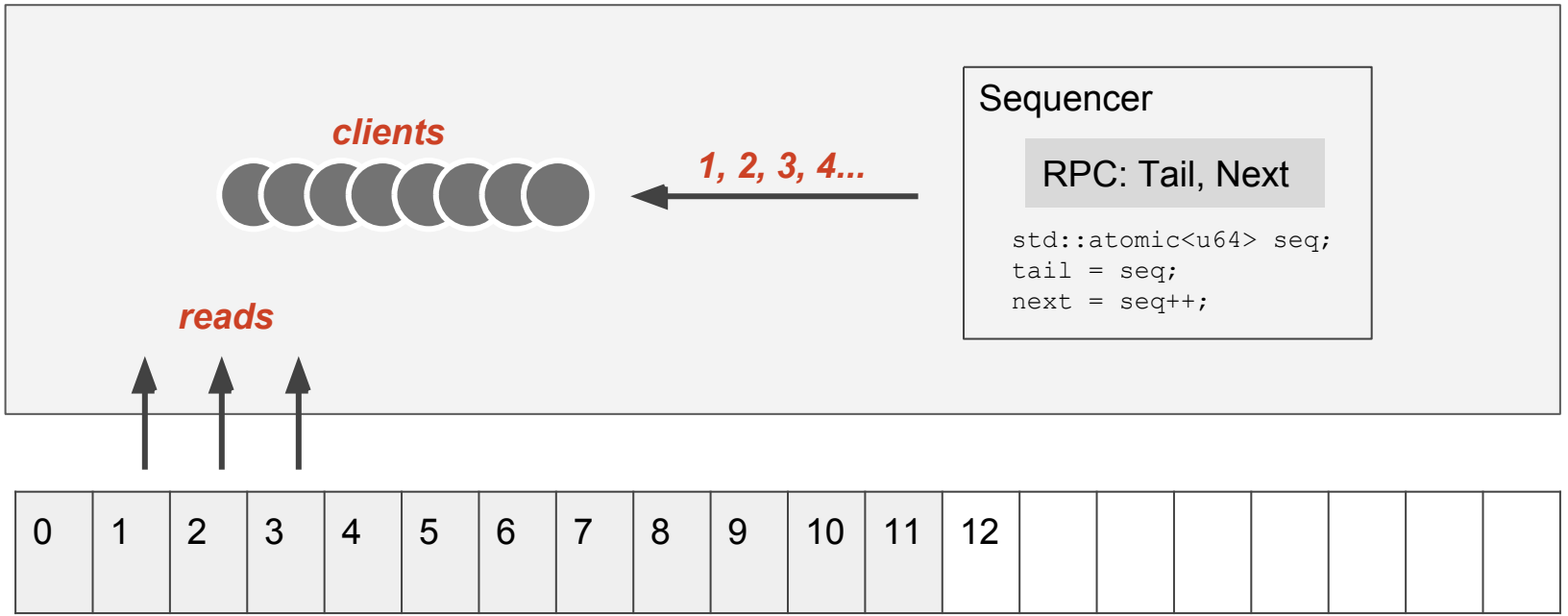
In this paper, we posit that flash storage opens the door to shared log designs within data centers, where hun-

and geo-distribution [16]; and even a primary data store that leverages fast appends on underlying media. Flash is an ideal medium for implementing a scalable shared log, supporting fast, contention-free random reads to the body of the log and fast sequential writes to its tail.

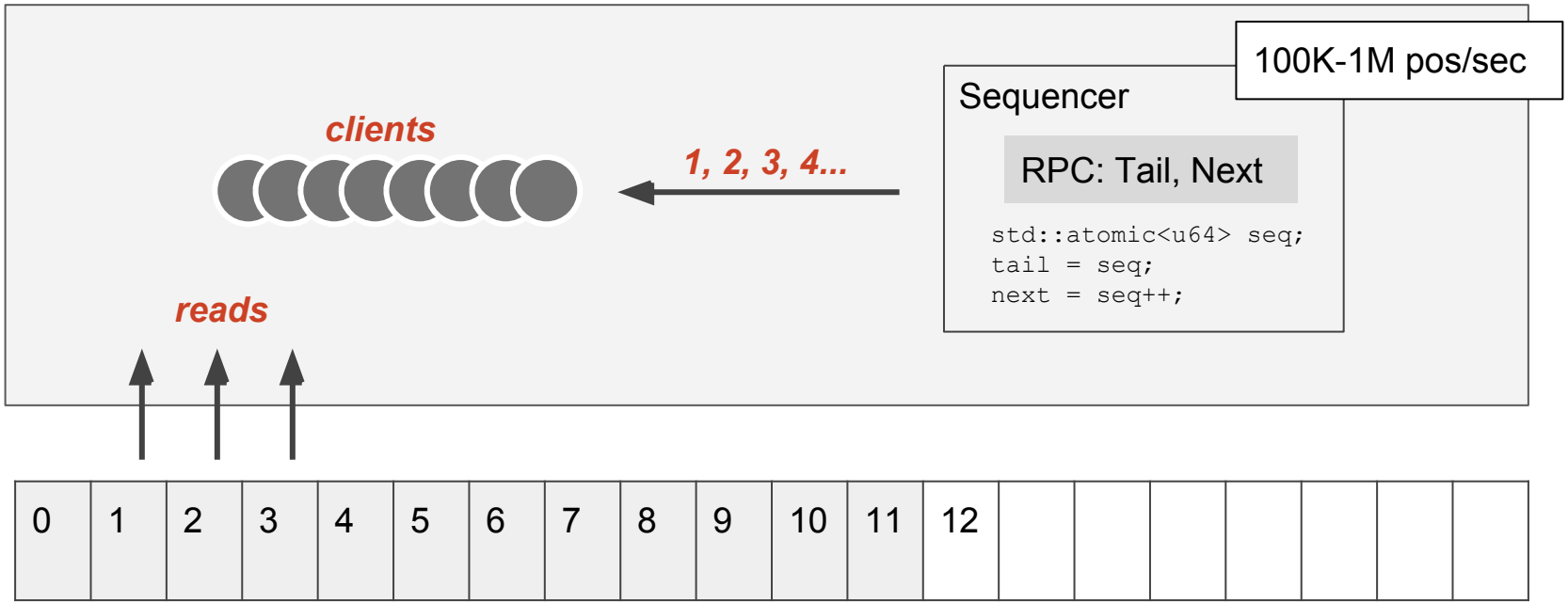
One simple option for implementing a flash-based shared log is to outfit a high-end server with an expensive PCI-e SSD (e.g., Fusion-io [2]), replicating it to handle failures and scale read throughput. However, the resulting log is limited in append throughput by the bandwidth of a single server. In addition, interposing bulky, general-purpose servers between the network and flash can create performance bottlenecks, leaving bandwidth underutilized, and can also offset the power benefits of flash. In contrast, clusters of small flash units have been shown to be balanced, power-efficient and incrementally scalable [5]. Required is a distributed implementation of a shared log that can operate over such clusters.

Accordingly, we present CORFU, a shared log abstraction implemented over a cluster of flash units. In CORFU, each position in the shared log is mapped to a set of flash pages on different flash units. This map is maintained – consistently and compactly – at the clients. To read a particular position in the shared log, a client uses its local copy of this map to determine a corresponding physical flash page, and then directly issues a read to the flash unit storing that page. To append data, a client first determines the next available position in the shared log – using a sequencer node as an optimization for avoiding contention with other appending clients – and then writes data directly to the set of physical flash

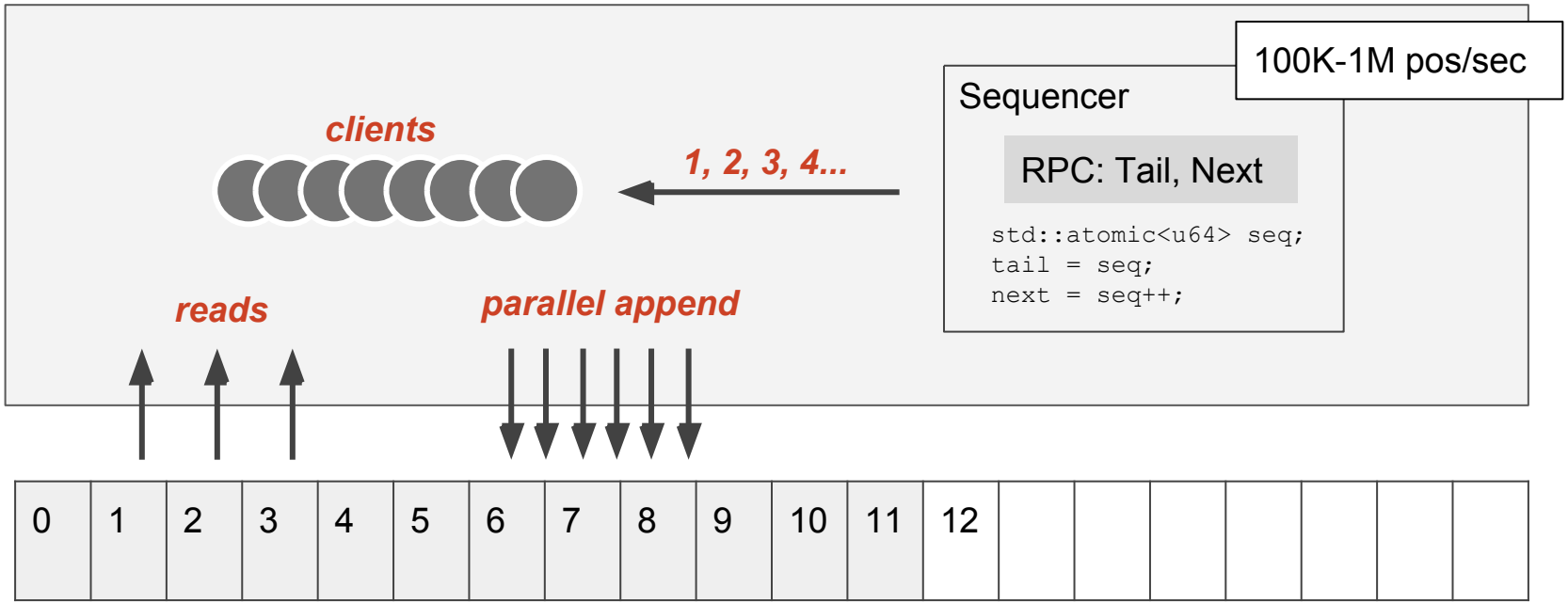
CORFU decouples I/O from ordering



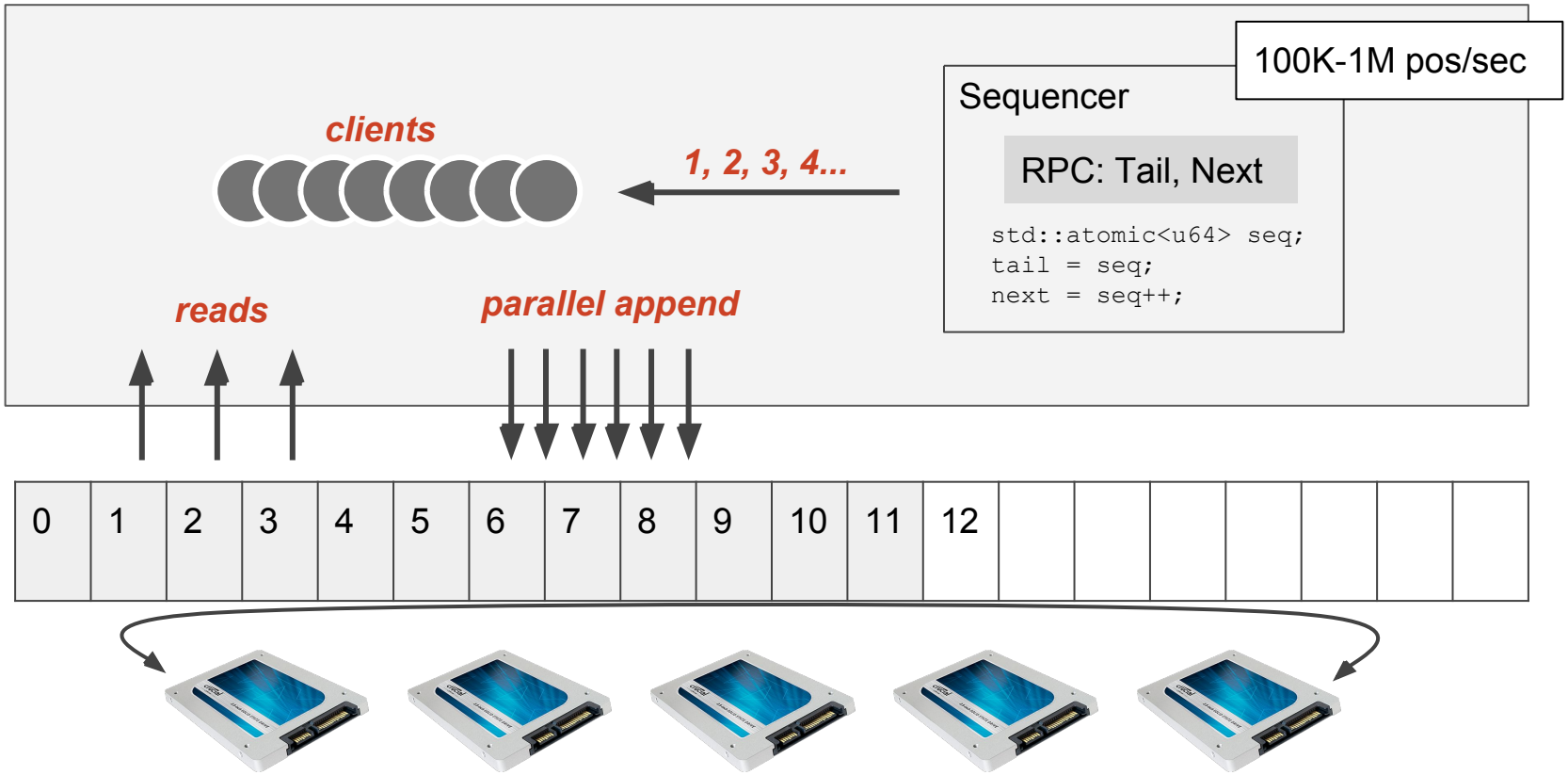
CORFU decouples I/O from ordering



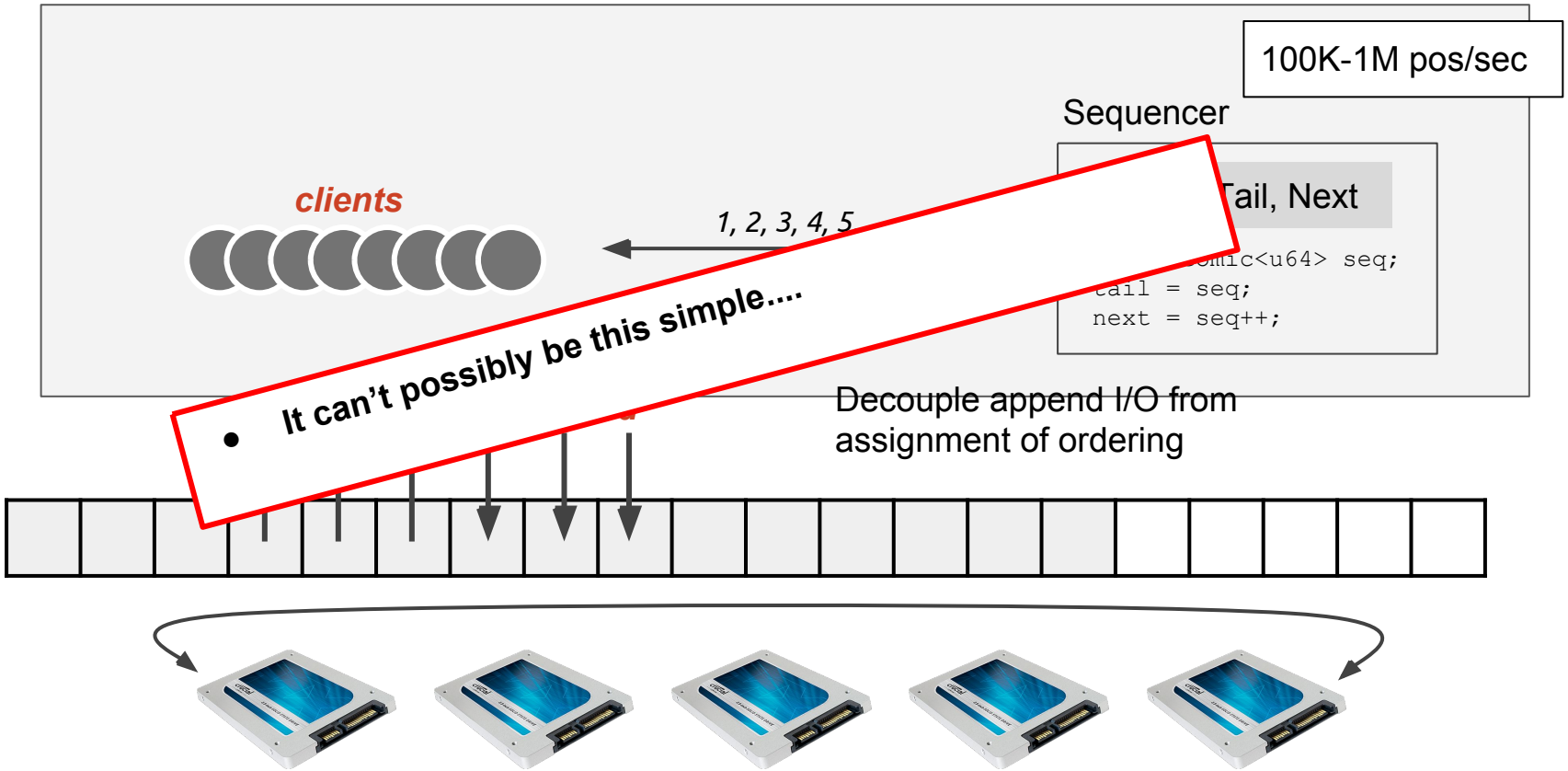
CORFU decouples I/O from ordering



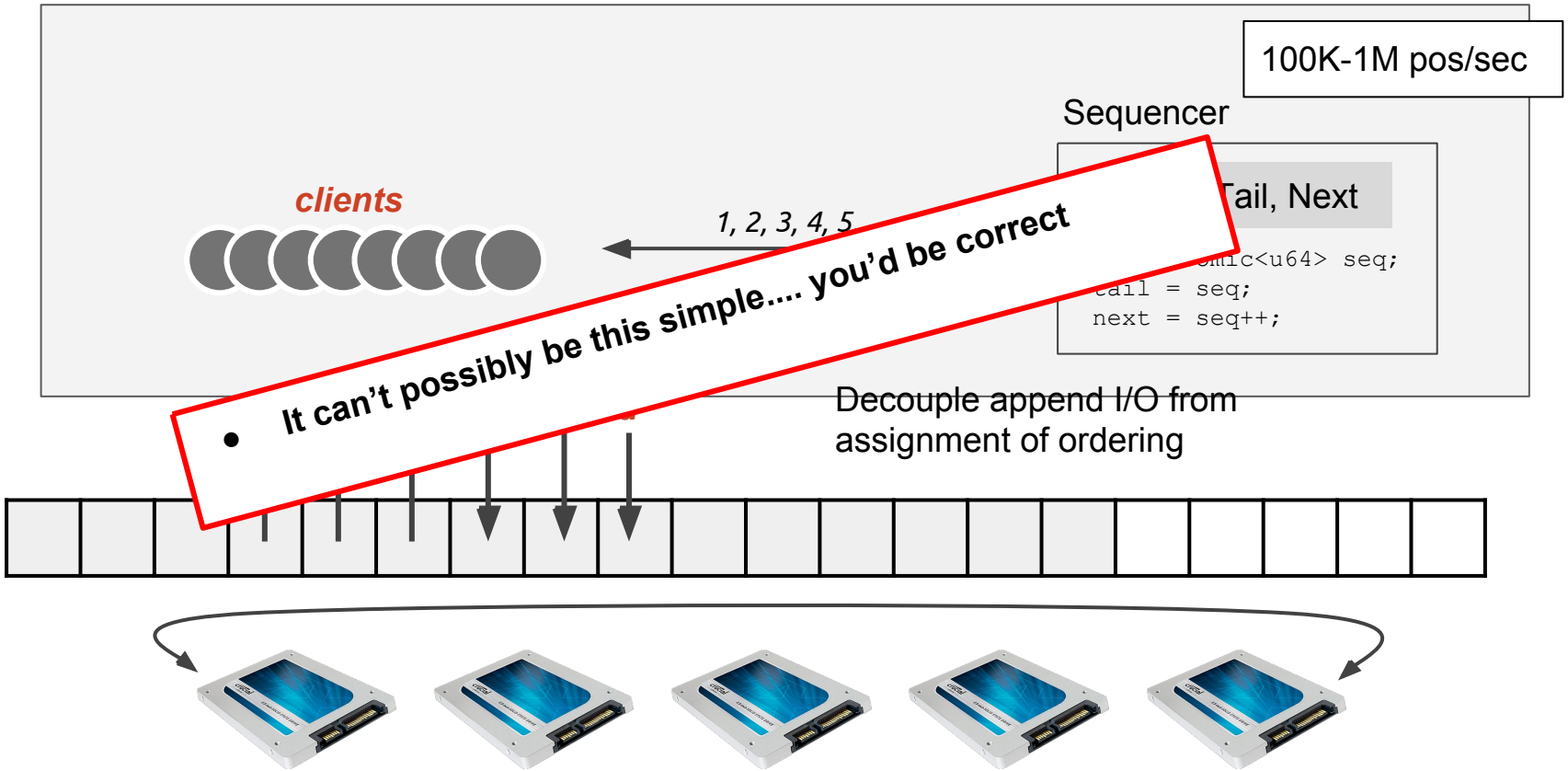
CORFU stripes log across a cluster of flash devices



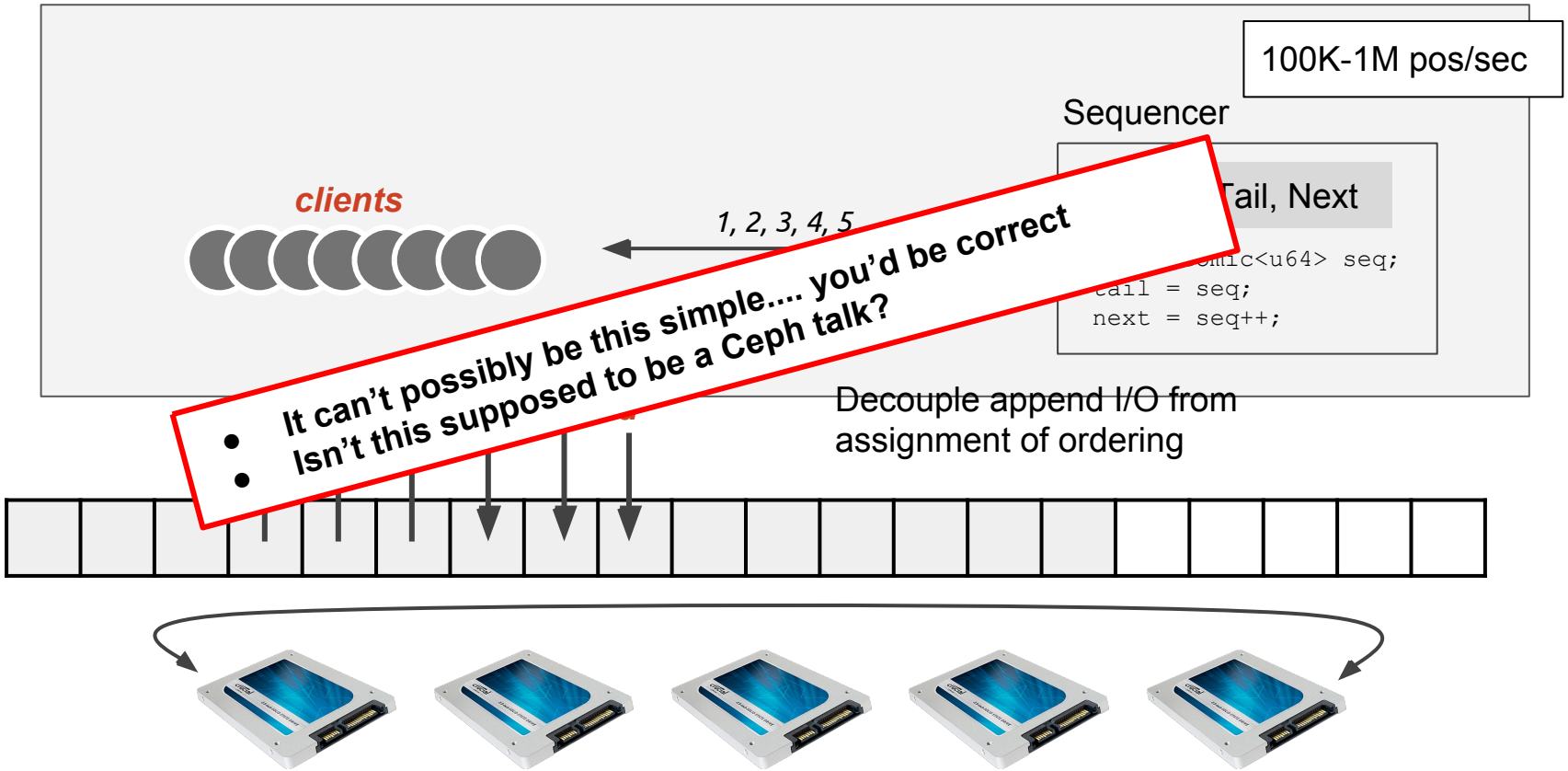
You might be thinking...



You might be thinking...



You might be thinking...

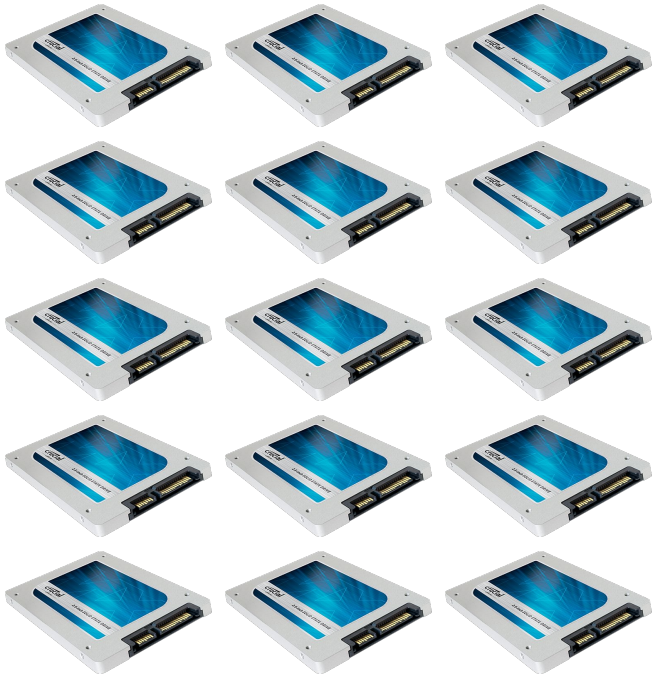


The CORFU I/O path

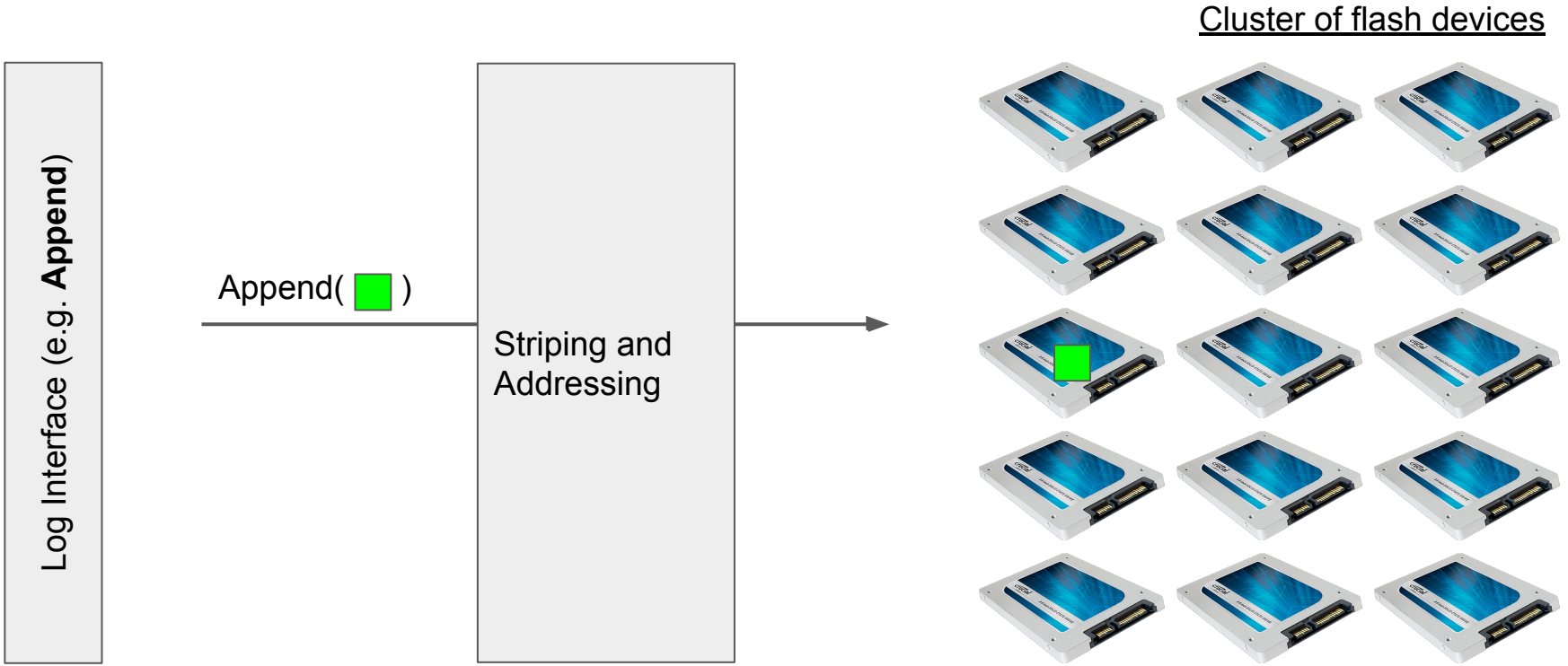
Log Interface (e.g. **Append**)

Append(■) ??

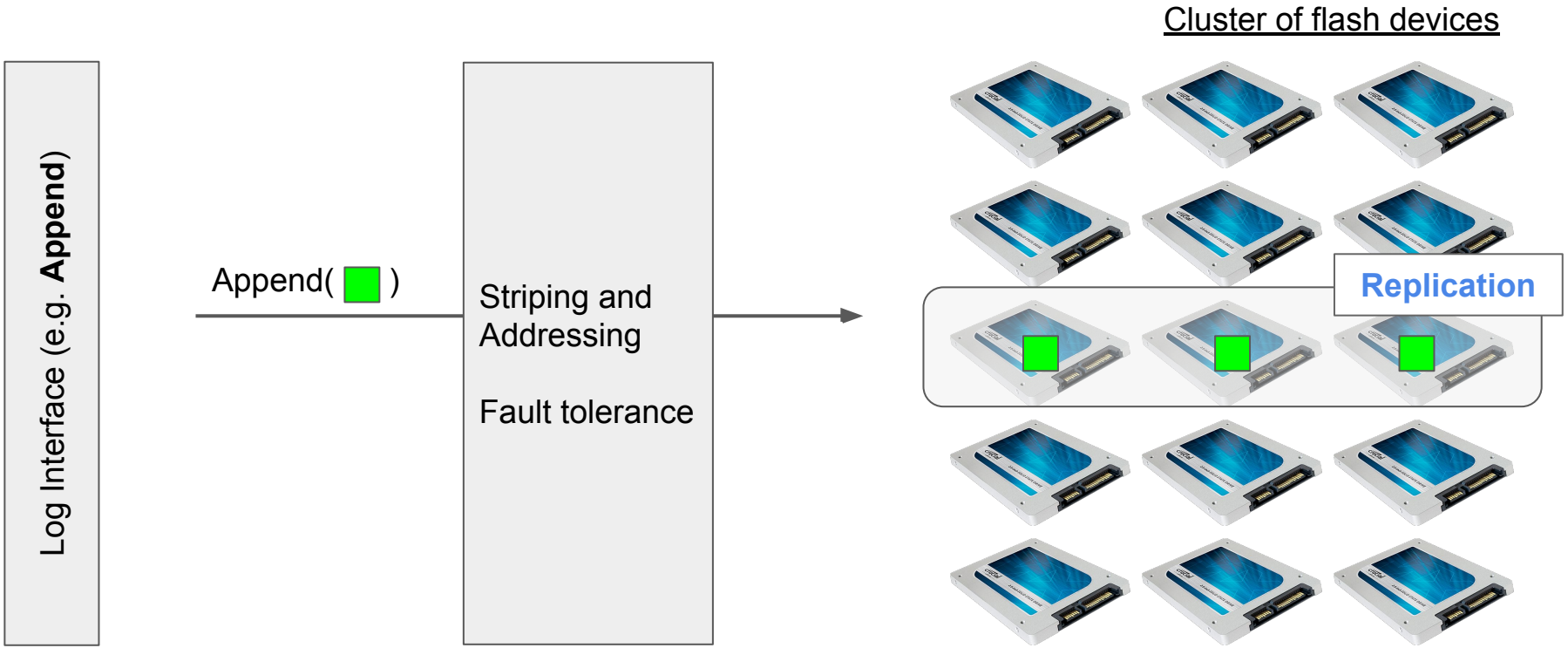
Cluster of flash devices



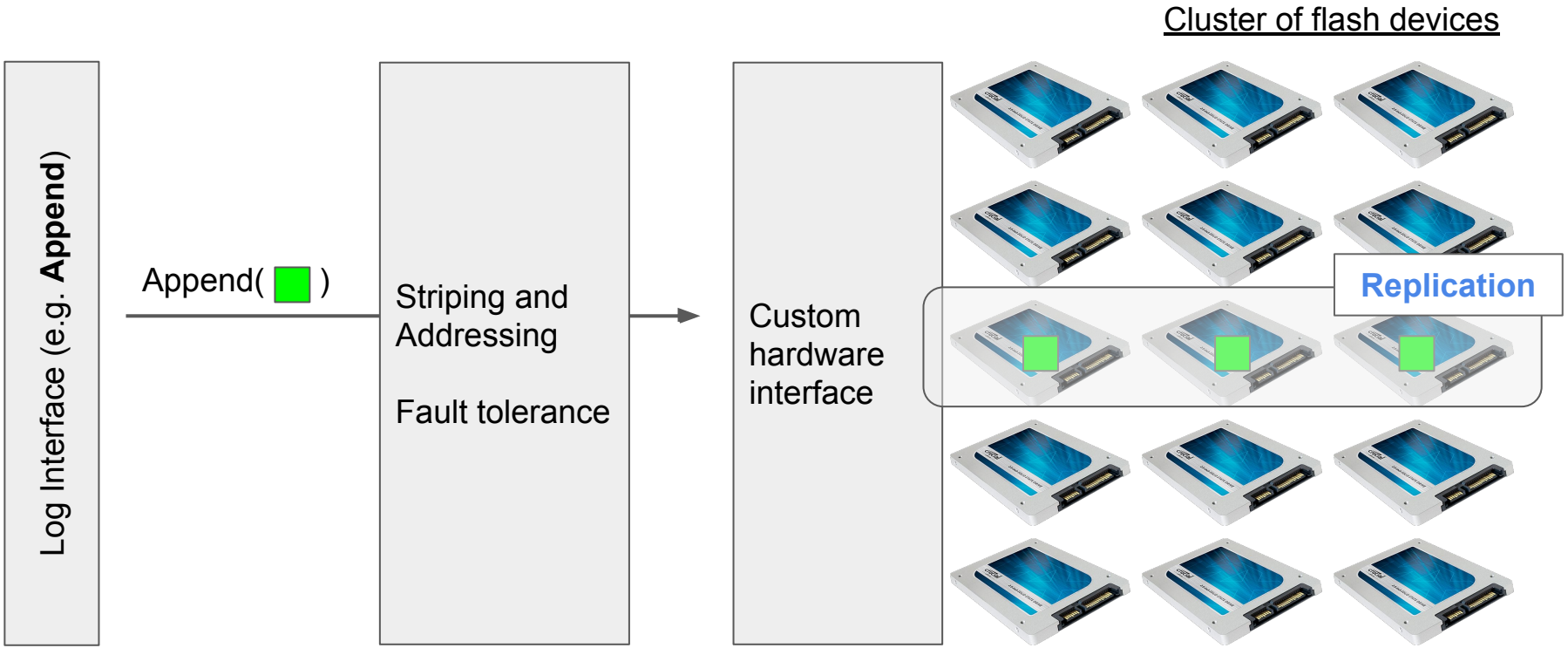
CORFU uses a cluster map and striping function



CORFU replicates log entries across the cluster



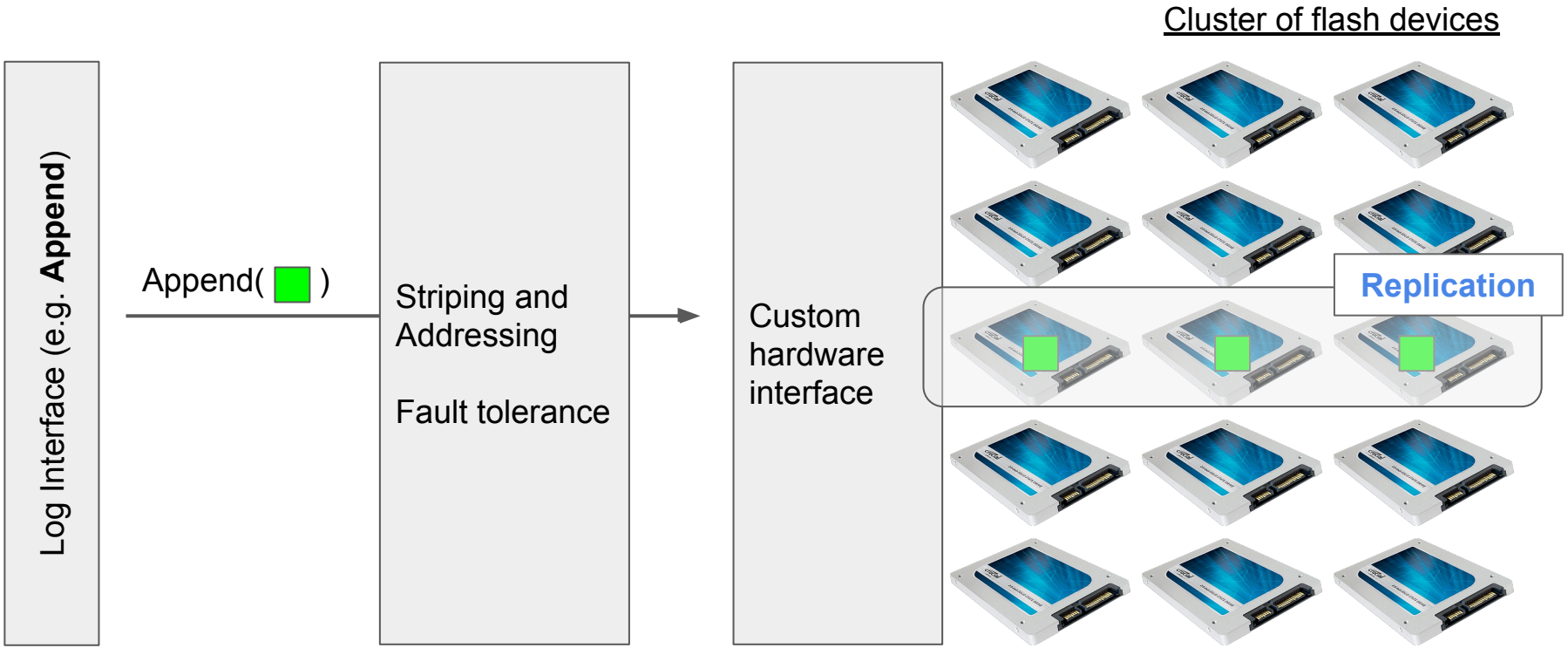
The CORFU protocol relies on custom I/O interface



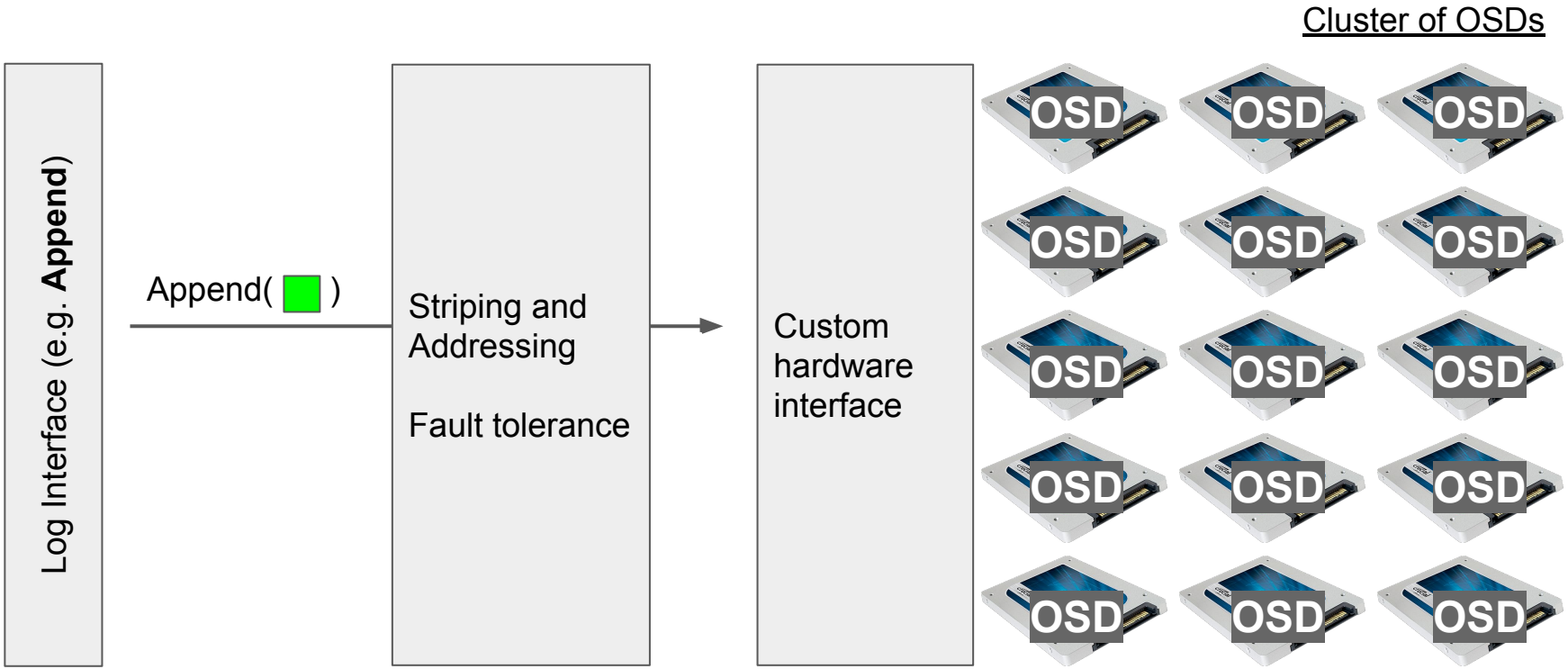
A dedicated CORFU cluster is expensive...

- Duplicated services
  - Fault-tolerance
  - Metadata management
- Dedicated / over-provisioned hardware
  - You need a full cluster of flash devices!
- Custom storage interfaces
  - Hardware or software
- Already have a Ceph cluster?
  - Too bad
- **Can we use software-defined storage?**

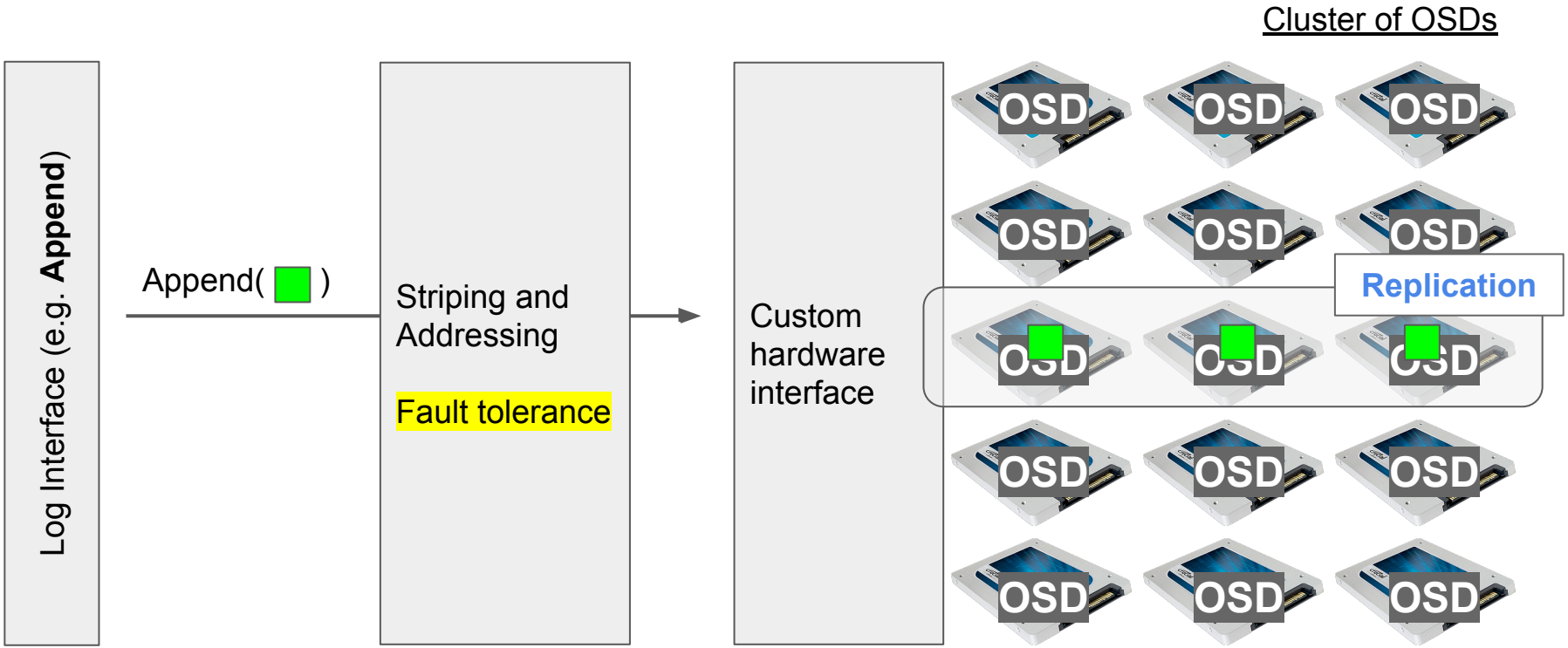
Mapping the components of CORFU onto Ceph



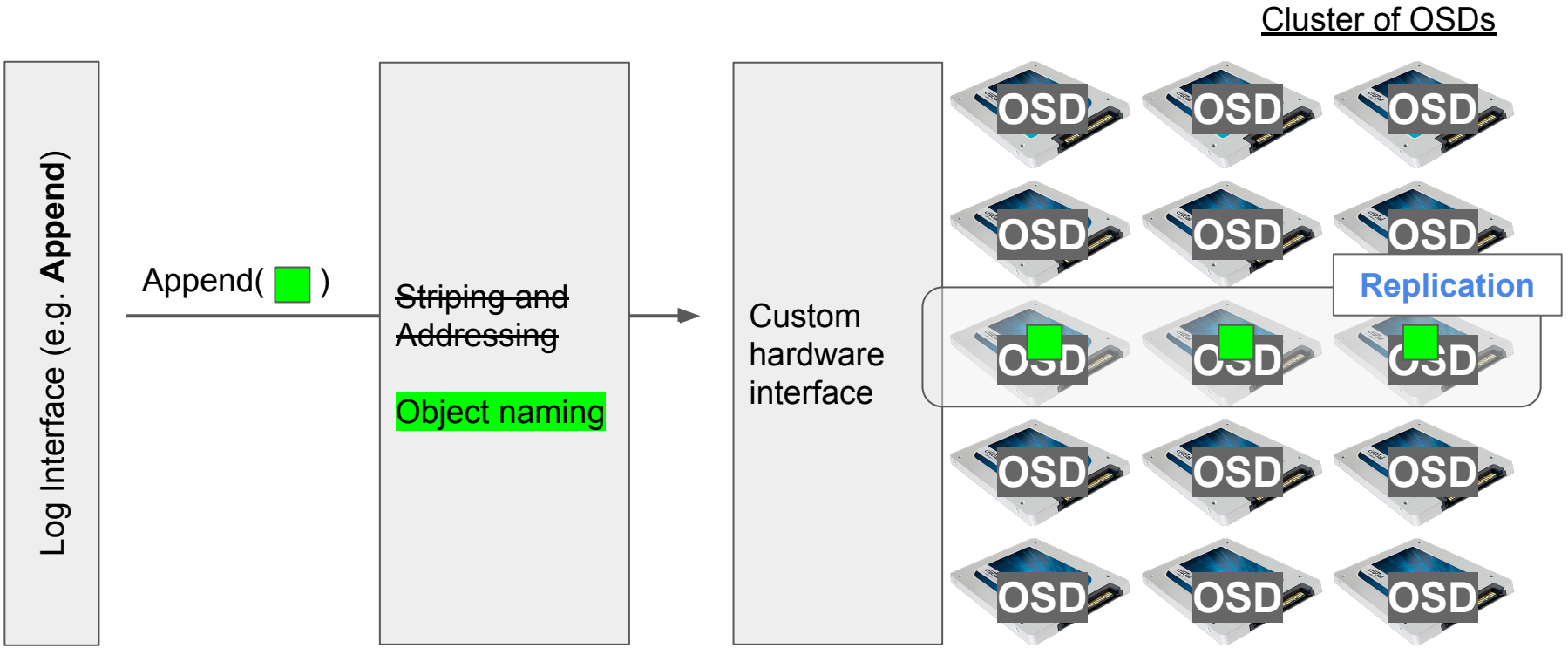
A cluster of OSDs rather than raw storage devices



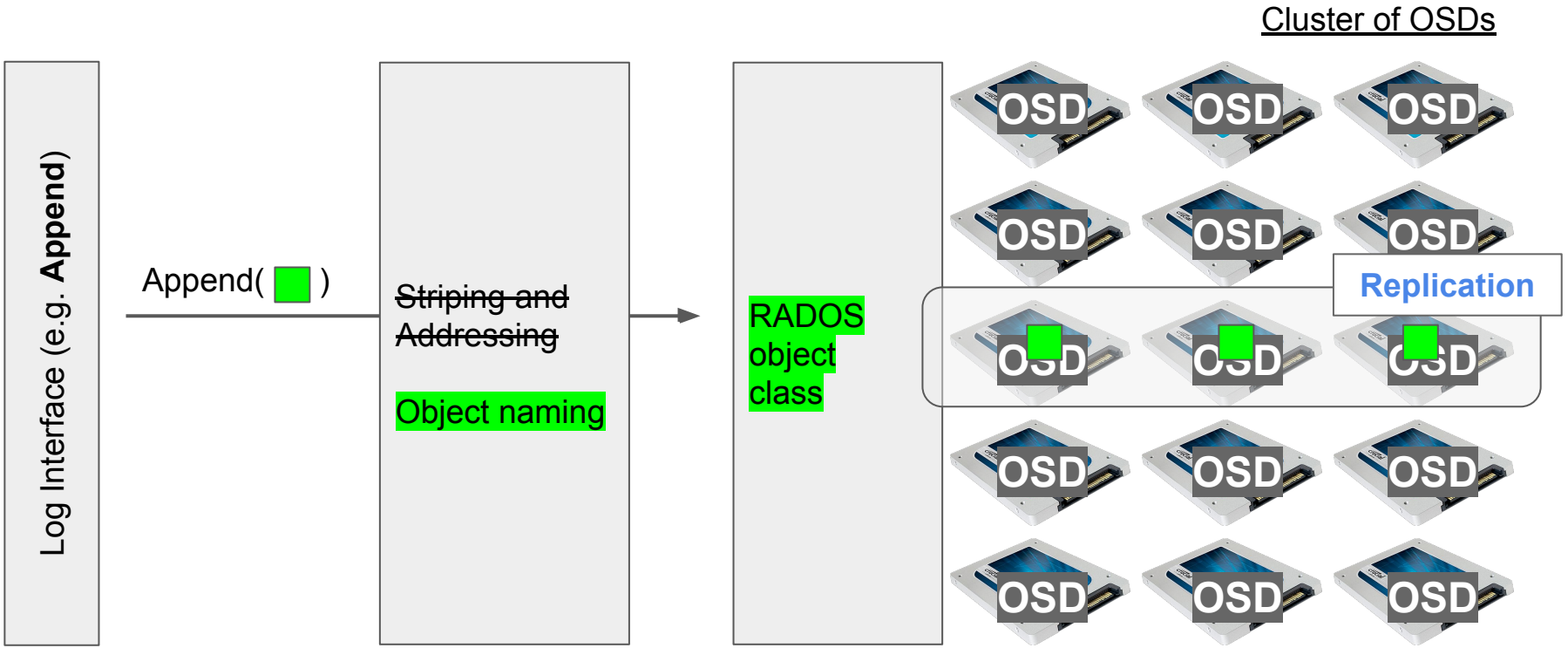
Ceph handles fault-tolerance transparently



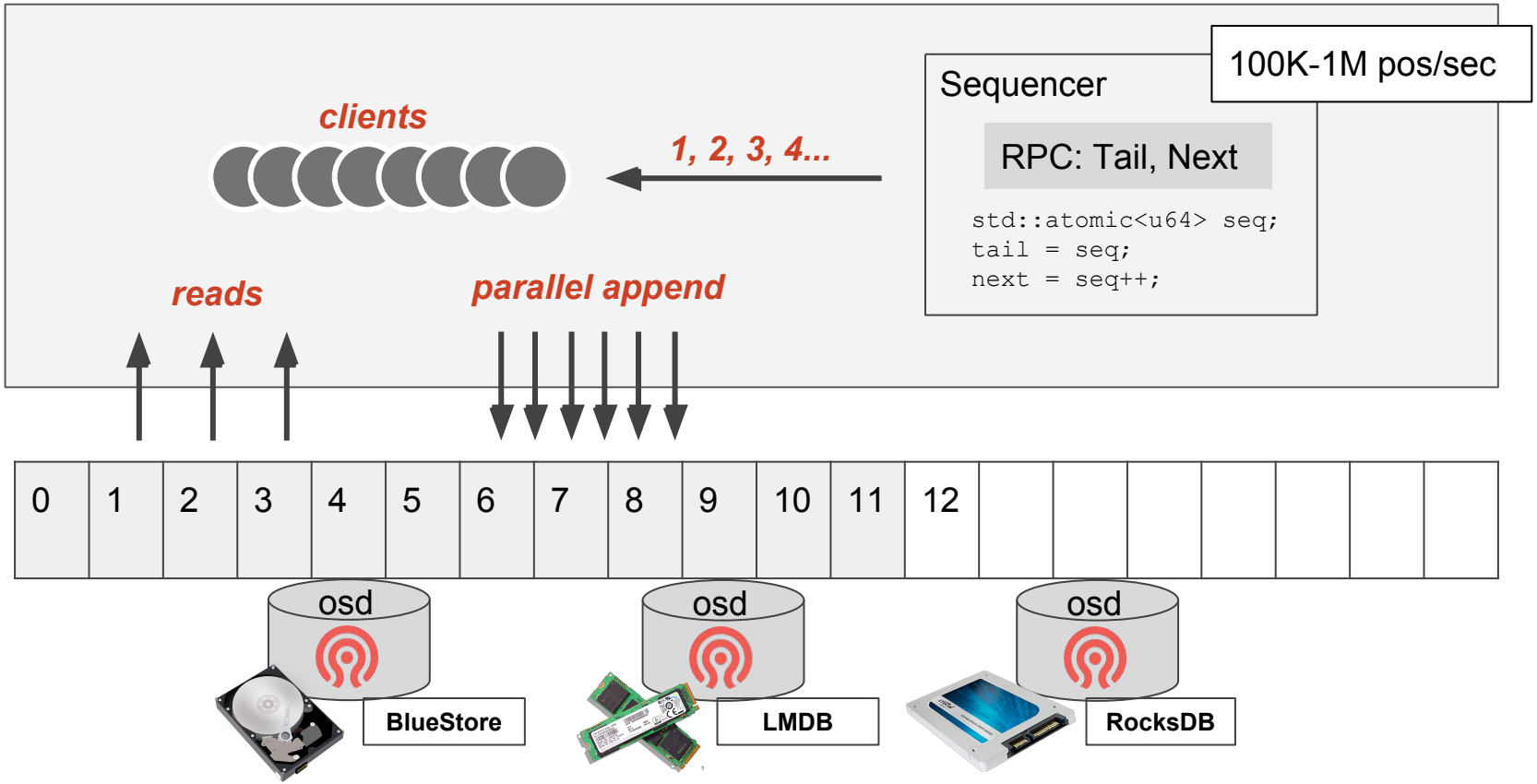
Log distribution becomes object naming issue



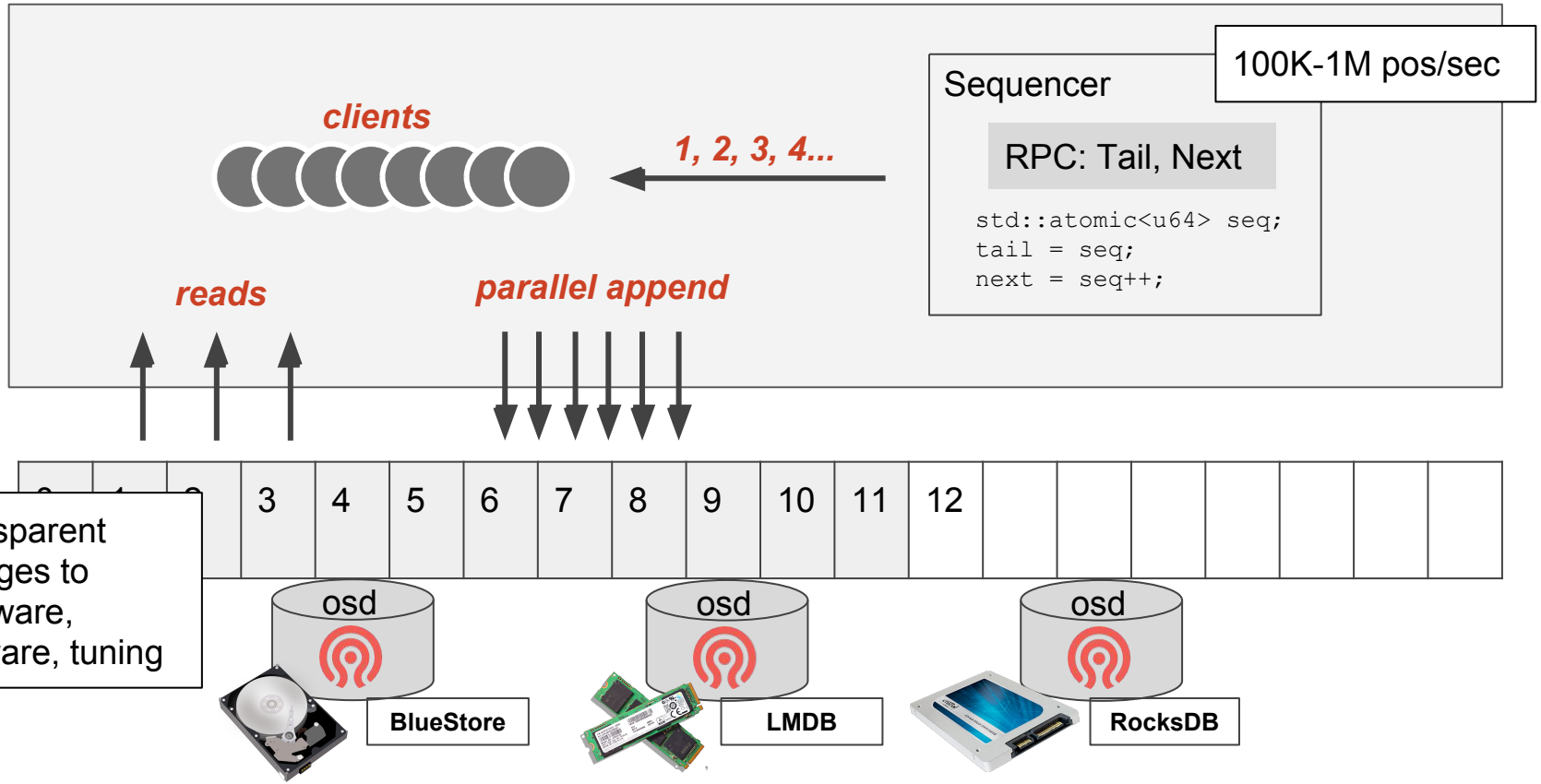
Storage interface built using RADOS object classes



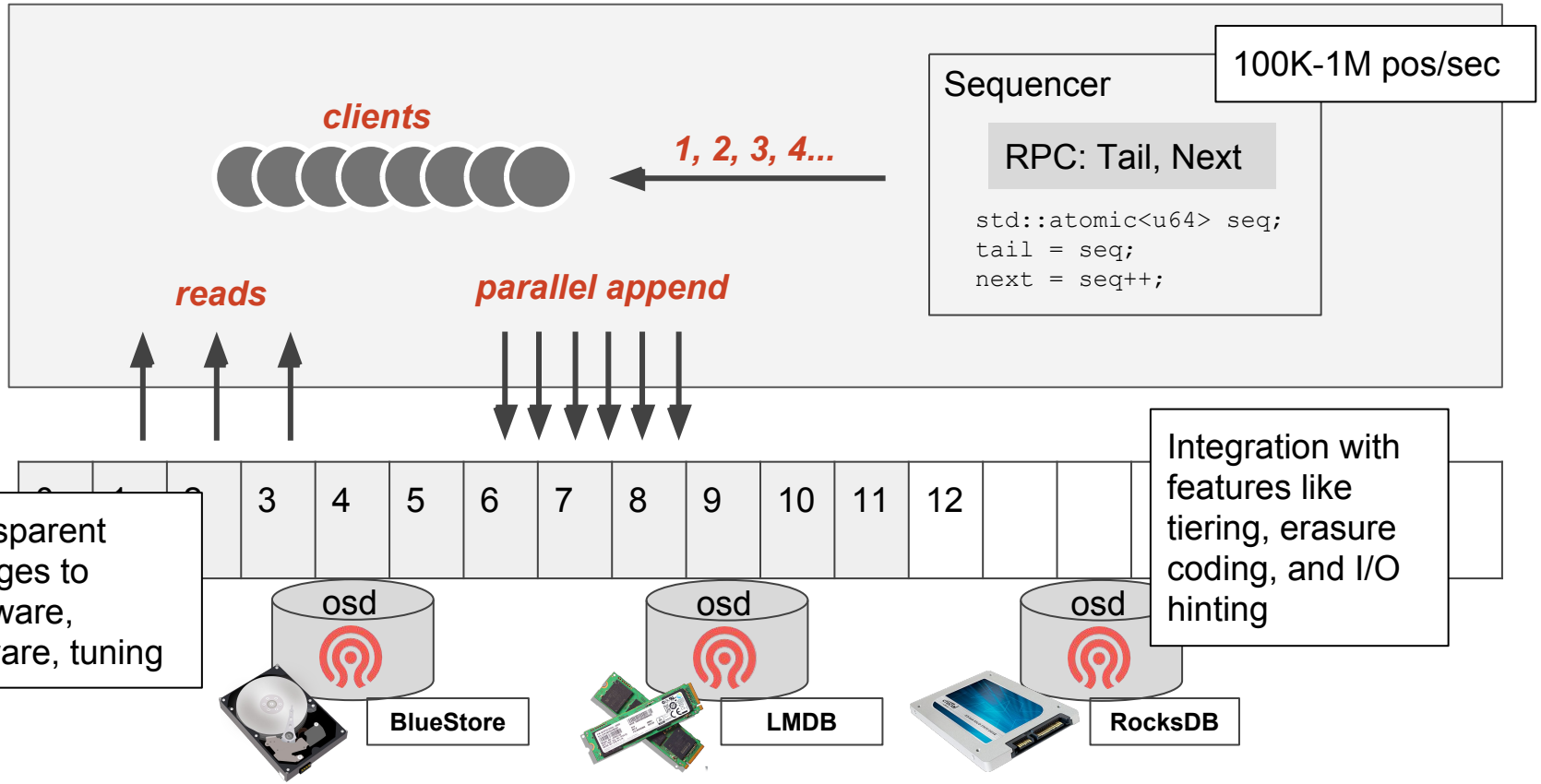
ZLog is an implementation of CORFU on Ceph



ZLog is an implementation of CORFU on Ceph



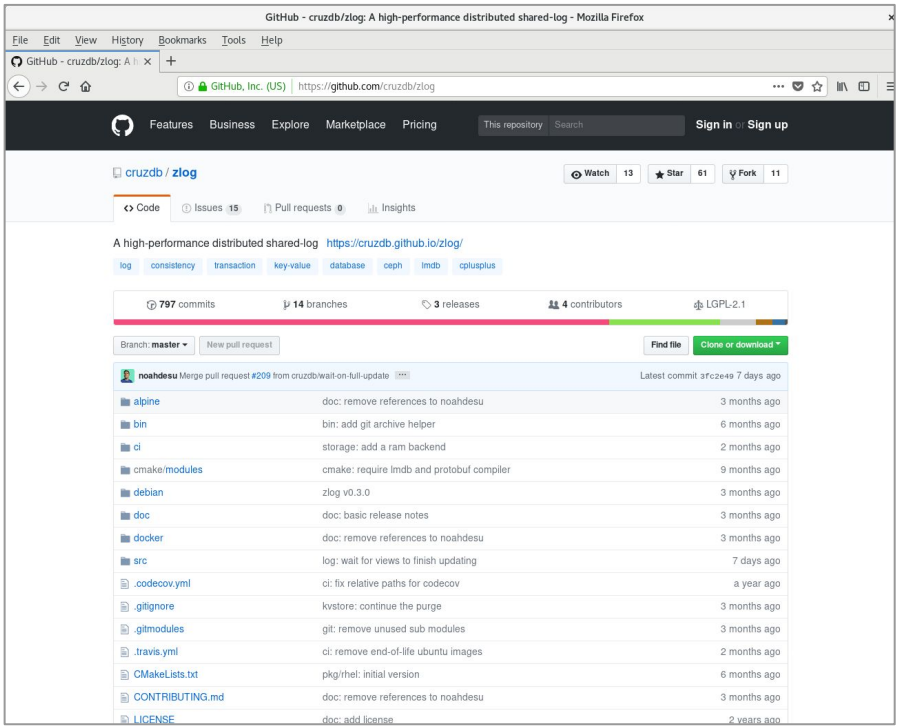
ZLog is an implementation of CORFU on Ceph



The ZLog project is open-source on Github

- https://github.com/cruzdb/zlog
- Development backend (LMDB)
- Tools to build Ceph plugin
- High and low-level APIs

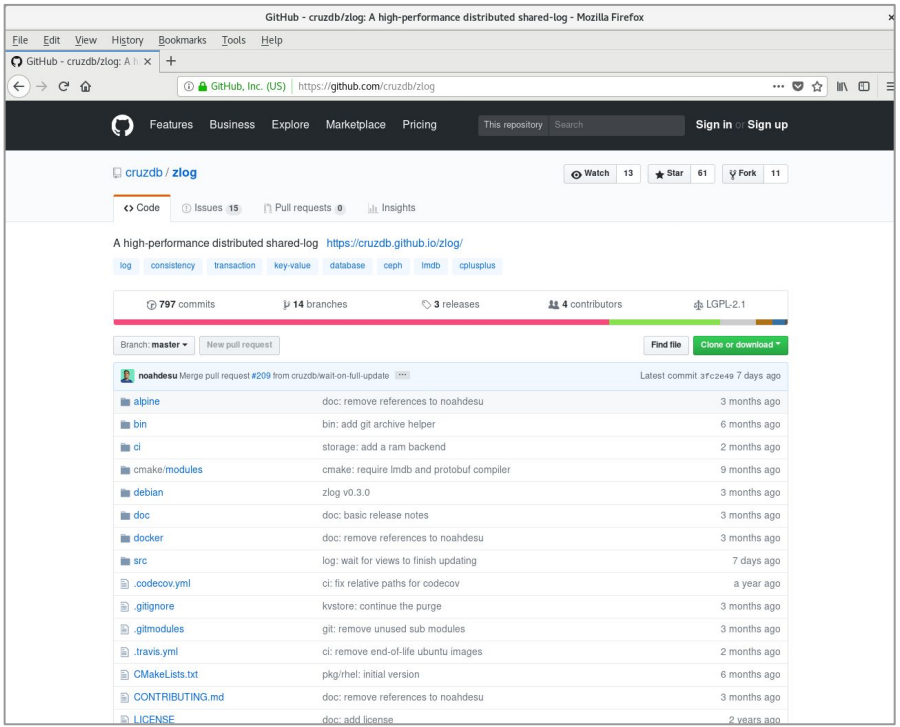
```
// build a backend instance
auto backend = std::unique_ptr<zlog::Backend>(
  new zlog::storage::ceph::CephBackend(ioctx));

// open the log
zlog::Log log;
int ret = zlog::Log::CreateWithBackend(std::move(backend),
  "mylog", &log);

// append to the log
uint64_t pos;
log.Append(Slice("foo"), &pos);
```



The ZLog project is open-source on Github

- https://github.com/cruzdb/zlog
- Development backend (LMDB)
- Tools to build Ceph plugin
- High and low-level APIs

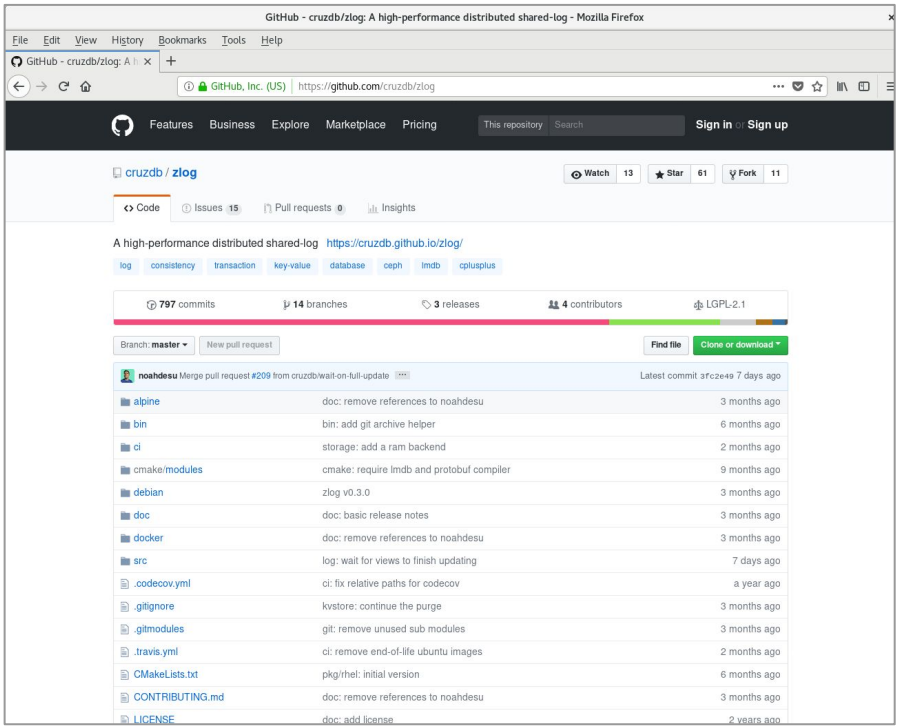
```
// build a backend instance
auto backend = std::unique_ptr<zlog::Backend>(
  new zlog::storage::ceph::CephBackend(ioctx));

// open the log
zlog::Log log;
int ret = zlog::Log::CreateWithBackend(std::move(backend),
  "mylog", &log);

// append to the log
uint64_t pos;
log.Append(Slice("foo"), &pos);
```



The ZLog project is open-source on Github

- https://github.com/cruzdb/zlog
- Development backend (LMDB)
- Tools to build Ceph plugin
- High and low-level APIs

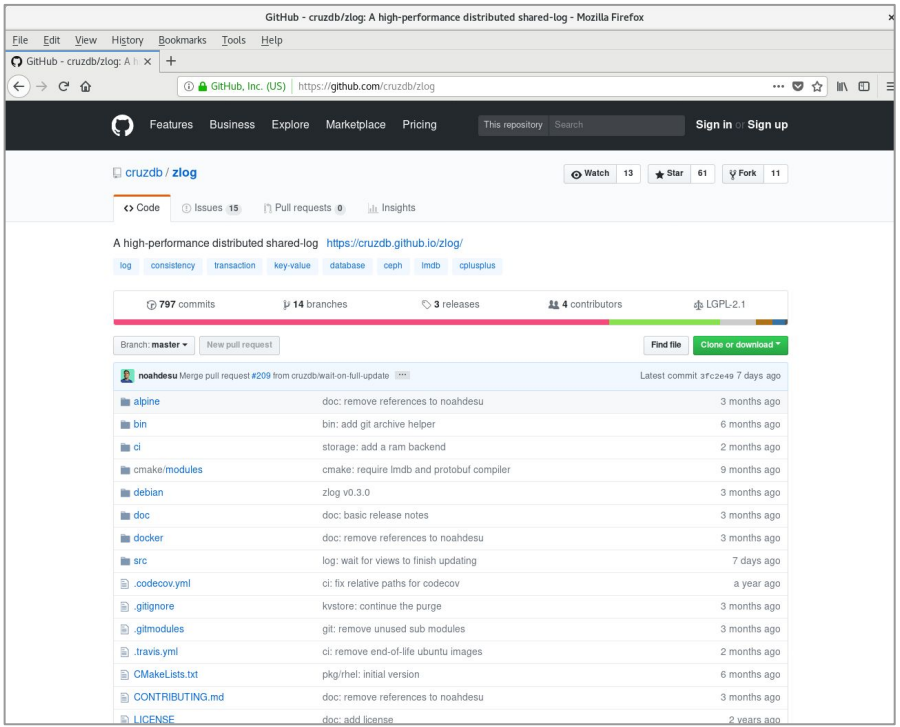
```
// build a backend instance
auto backend = std::unique_ptr<zlog::Backend>(
  new zlog::storage::ceph::CephBackend(ioctx));

// open the log
zlog::Log log;
int ret = zlog::Log::CreateWithBackend(std::move(backend),
  "mylog", &log);

// append to the log
uint64_t pos;
log.Append(Slice("foo"), &pos);
```



The ZLog project is open-source on Github

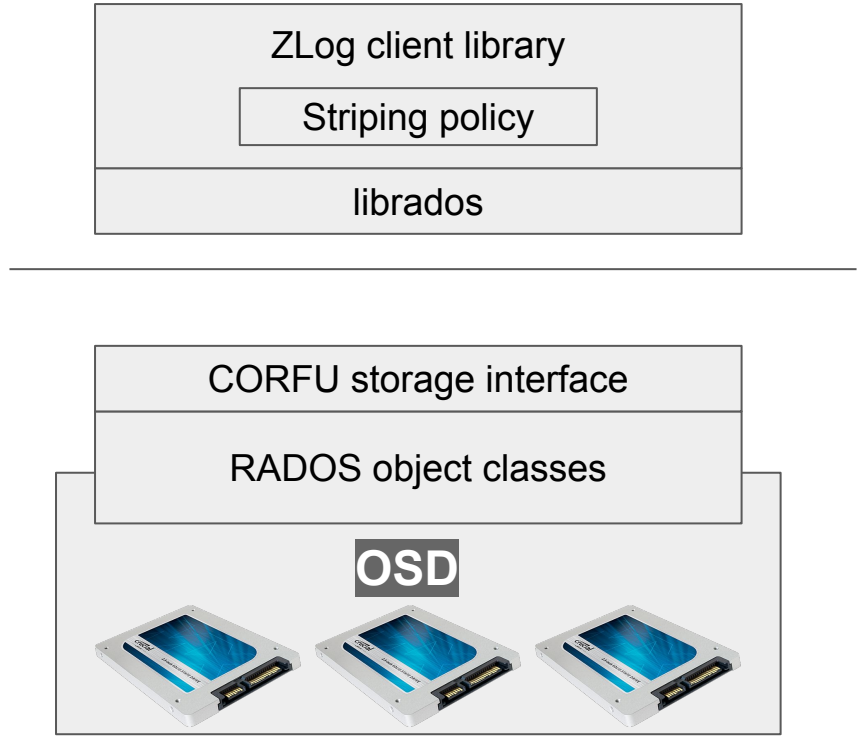
- https://github.com/cruzdb/zlog
- Development backend (LMDB)
- Tools to build Ceph plugin
- High and low-level APIs

```
// build a backend instance
auto backend = std::unique_ptr<zlog::Backend>(
  new zlog::storage::ceph::CephBackend(ioctx));

// open the log
zlog::Log log;
int ret = zlog::Log::CreateWithBackend(std::move(backend),
  "mylog", &log);

// append to the log
uint64_t pos;
log.Append(Slice("foo"), &pos);
```

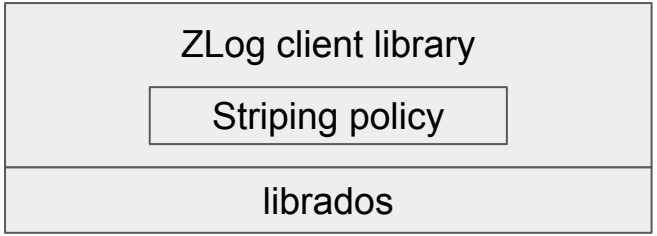


Building applications on RADOS using object classes

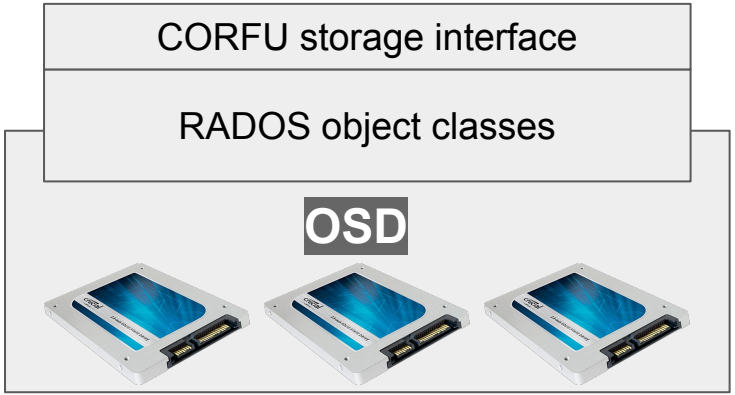
Design decisions for the ZLog I/O path



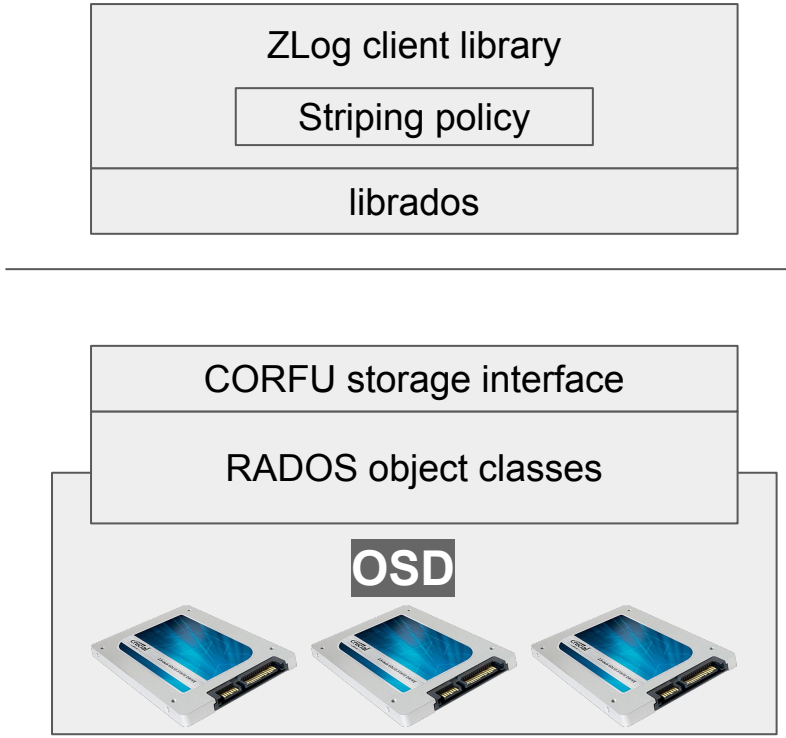
Design decisions for the ZLog I/O path



- Log entry → Object

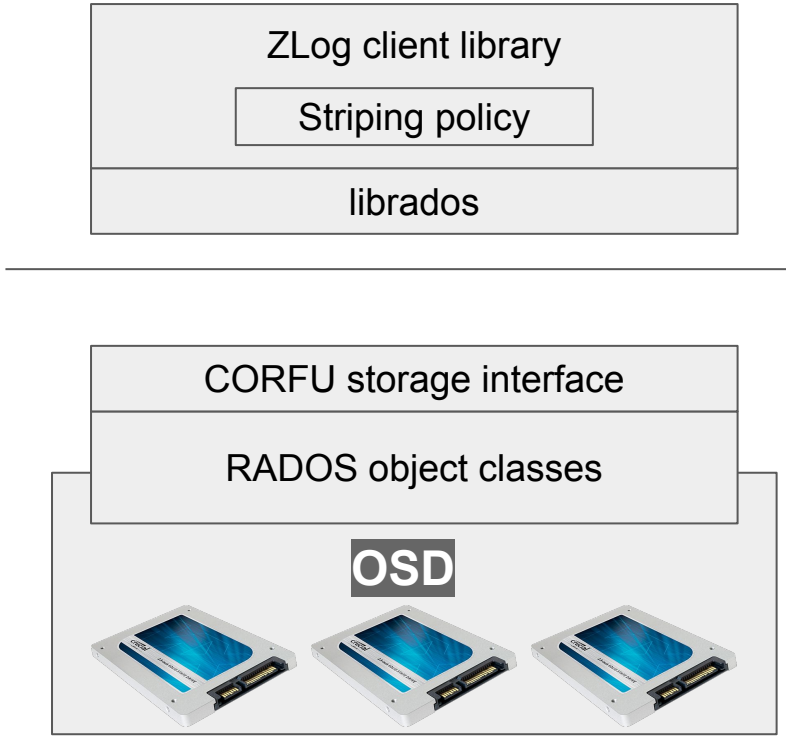


Design decisions for the ZLog I/O path



- ~~Log entry → Object~~
- Stripe log across objects

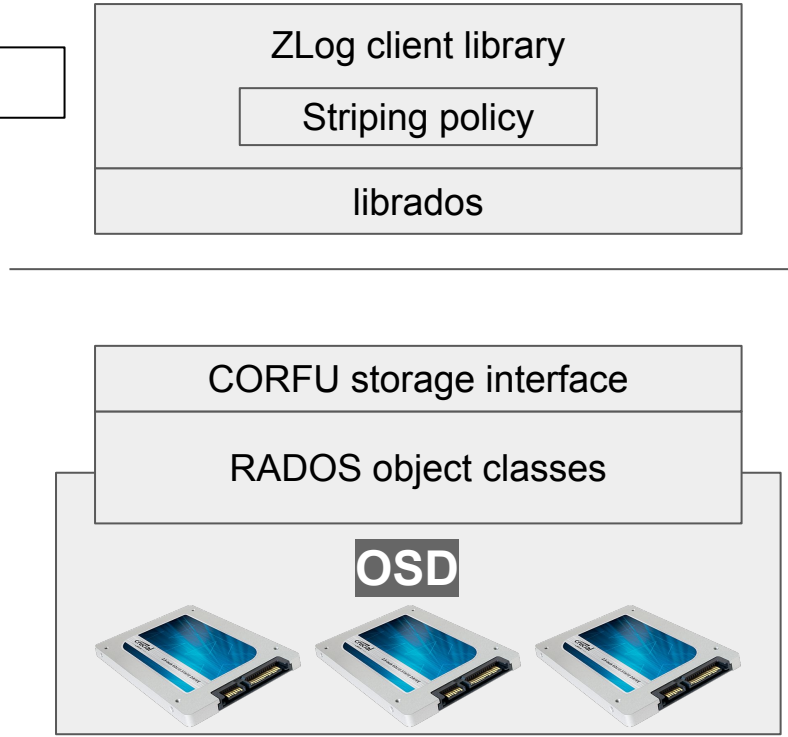
Design decisions for the ZLog I/O path



- ~~Log entry → Object~~
- Stripe log across objects
- … but not too many objects

Design decisions for the ZLog I/O path

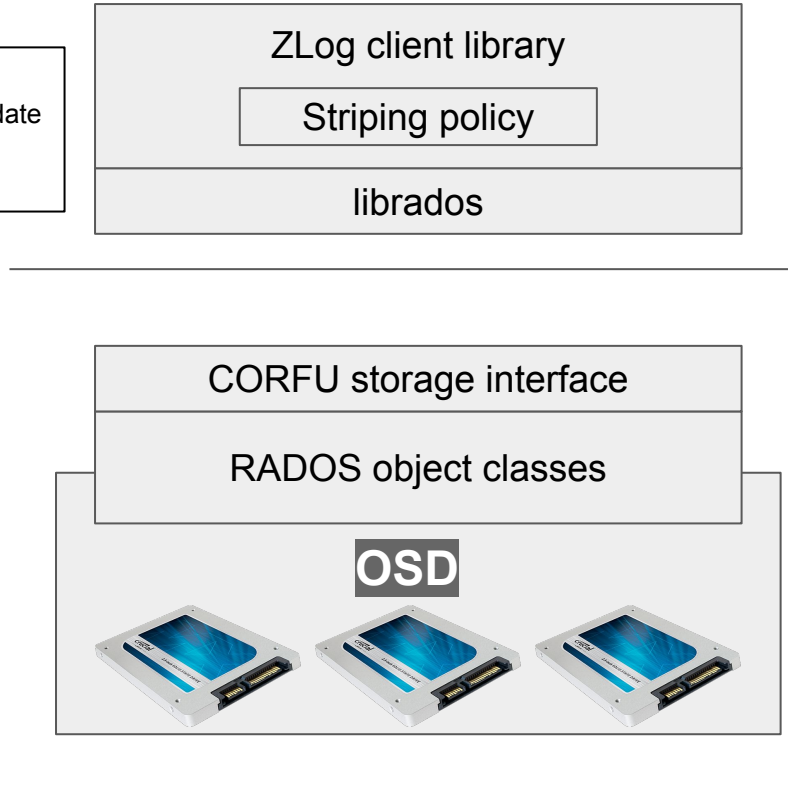
- Changes to striping policy



- ~~Log entry → Object~~
- Stripe log across objects
- … but not too many objects

Design decisions for the ZLog I/O path

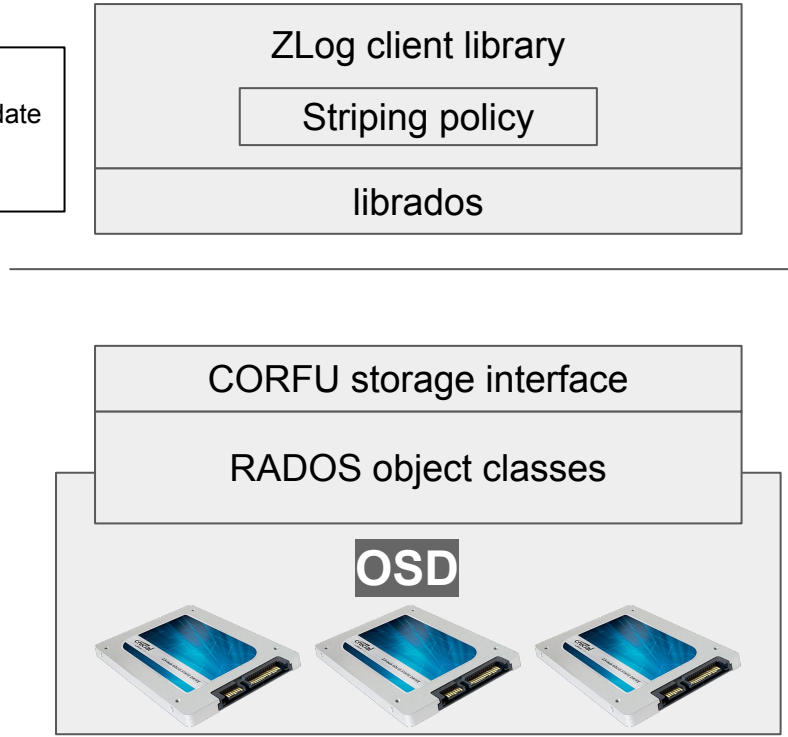
- Changes to striping policy
- Coordinated metadata update



- ~~Log entry → Object~~
- Stripe log across objects
- … but not too many objects

Design decisions for the ZLog I/O path

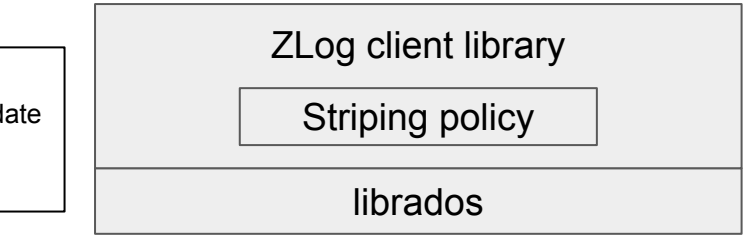
- Changes to striping policy
- Coordinated metadata update
- **Treat an object like a consensus API**



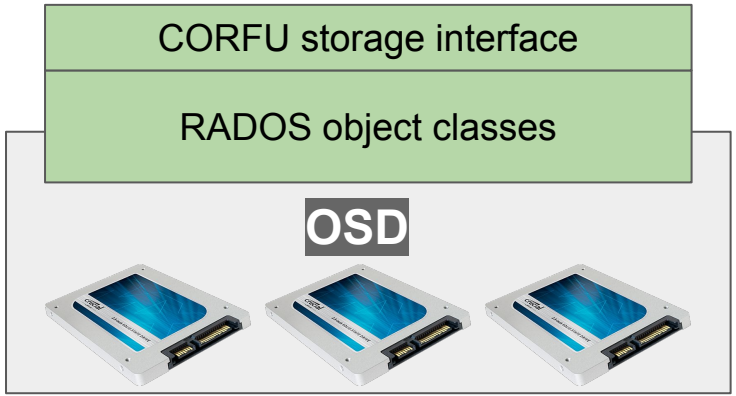
- ~~Log entry → Object~~
- Stripe log across objects
- … but not too many objects

Design decisions for the ZLog I/O path

- Changes to striping policy
- Coordinated metadata update
- **Treat an object like a consensus API**



- ~~Log entry → Object~~
- Stripe log across objects
- … but not too many objects



The CORFU storage interface

- write(obj, position, data)
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

The CORFU storage interface

- **write(obj, position, data)**
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

The CORFU storage interface

- **write(obj, position, data)**
  - Write-once: position must be open
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

The CORFU storage interface

- **write(obj, position, data)**
  - Write-once: position must be open
  - Request must be tagged with up-to-date epoch
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

The CORFU storage interface

- **write(obj, position, data)**
  - Write-once: position must be open
  - Request must be tagged with up-to-date epoch
  - Position must not fall within a GC'd range
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

The CORFU storage interface

- **write(obj, position, data)**
  - Write-once: position must be open
  - Request must be tagged with up-to-date epoch
  - Position must not fall within a GC'd range
  - Optionally update maximum position observed
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

The CORFU storage interface

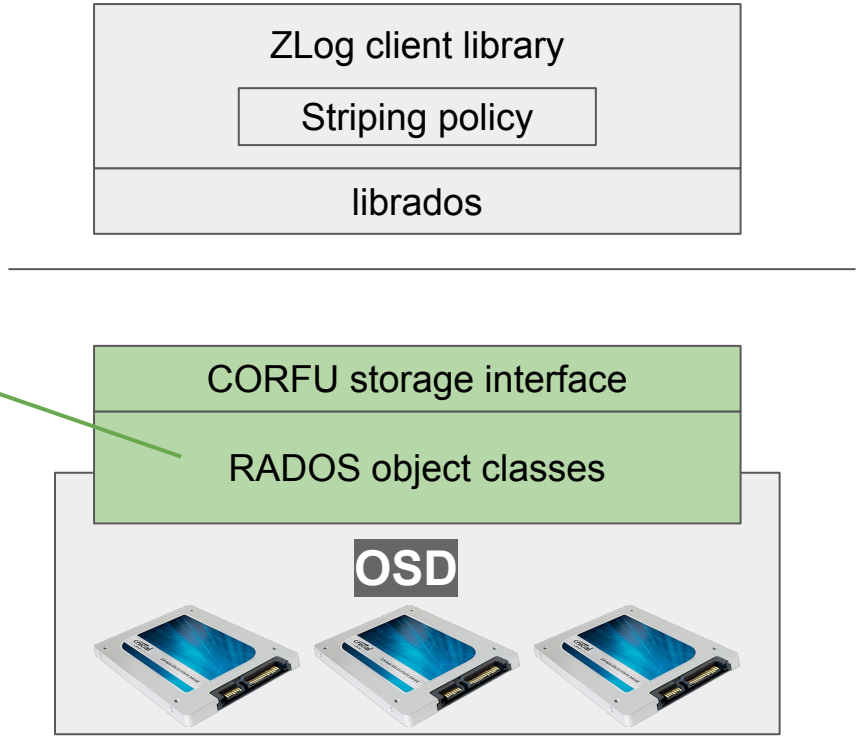
- **write(obj, position, data)**
  - Write-once: position must be open
  - Request must be tagged with up-to-date epoch
  - Position must not fall within a GC'd range
  - Optionally update maximum position observed
  - **Atomic update**
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

librados won’t [currently] cut it for this job

- **write(obj, position, data)**
  - Write-once: position must be open
  - Request must be tagged with up-to-date epoch
  - Position must not fall within a GC’d range
  - Optionally update maximum position observed
  - **Atomic update**
- read(obj, position)
- invalidate(obj, position)
- trim(obj, position)
  - Mark for GC
- seal(obj, epoch)

CORFU write interface as an object class

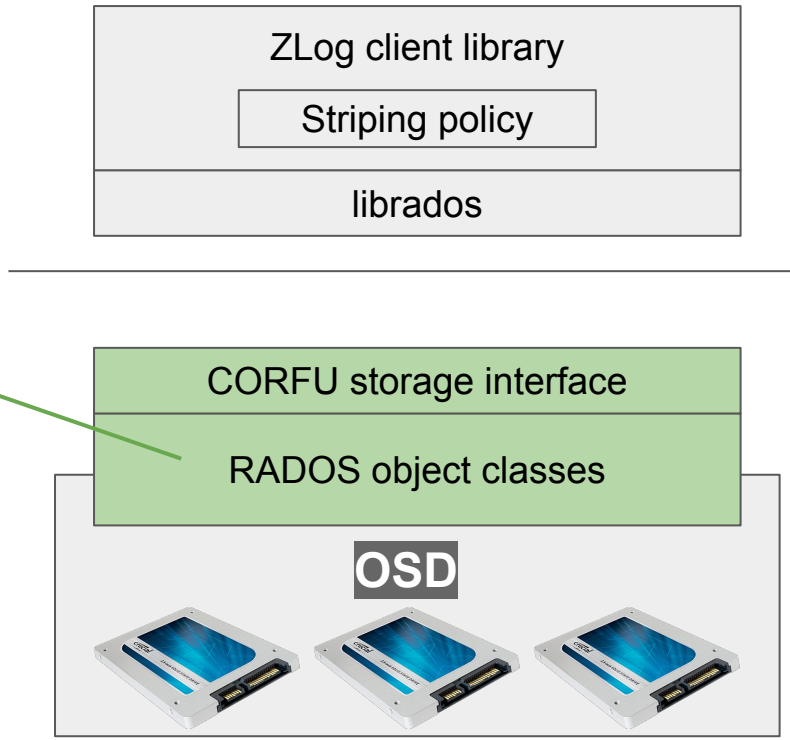
```
int write(context_t hctx, bufferlist *in, bufferlist *out) {
 // ensure up-to-date client
 if (epoch_guard(header, op.epoch(), false)) {
   return -ESPIPE;
 }

 // read entry based on request position
 ceph::bufferlist bl;
 int ret = cls_cxx_map_get_val(hctx, key, &bl);
 if (ret < 0)
   return ret;
 if (entry && !decode(bl, entry))
   return -EIO;
 return 0;

 // write entry if position is not taken
 if (ret == -ENOENT) {
   entry_write_entry(hctx, key, entry);

   if (!header.has_max_pos() || (op.pos() > header.max_pos()))
     header.set_max_pos(op.pos());

   ret = entry_write_header(hctx, header);
 } else {
  // handle errors associated with write-once semantics
 }
}
```



CORFU write interface as an object class

connection to client

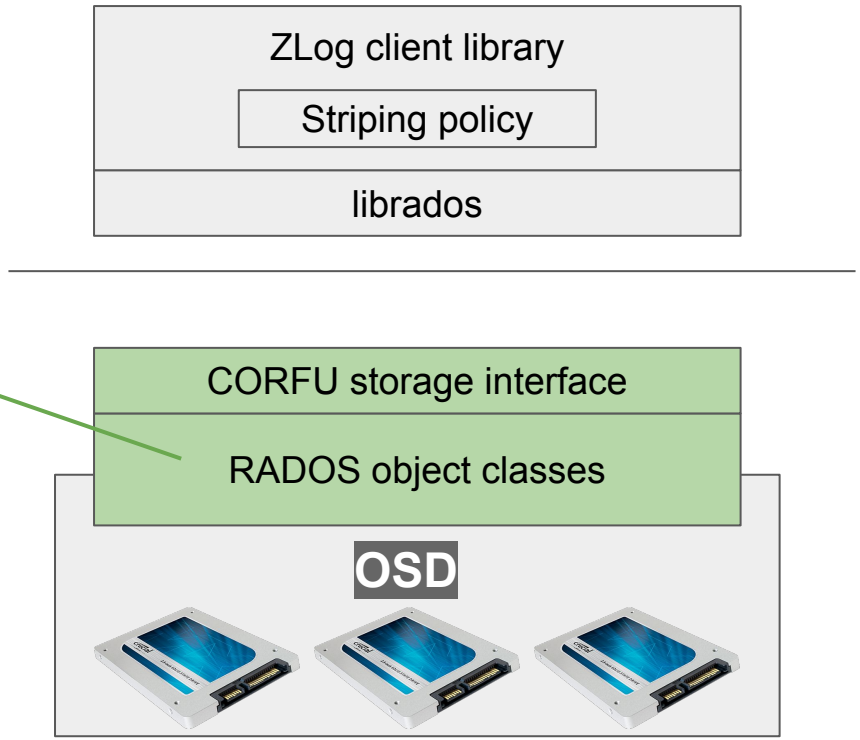
```
int write(context_t hctx, bufferlist *in, bufferlist *out) {
 // ensure up-to-date client
 if (epoch_guard(header, op.epoch(), false)) {
   return -ESPIPE;
 }

 // read entry based on request position
 ceph::bufferlist bl;
 int ret = cls_cxx_map_get_val(hctx, key, &bl);
 if (ret < 0)
   return ret;
 if (entry && !decode(bl, entry))
   return -EIO;
 return 0;

 // write entry if position is not taken
 if (ret == -ENOENT) {
   entry_write_entry(hctx, key, entry);

   if (!header.has_max_pos() || (op.pos() > header.max_pos()))
     header.set_max_pos(op.pos());

   ret = entry_write_header(hctx, header);
 } else {
  // handle errors associated with write-once semantics
 }
}
```



CORFU write interface as an object class

connection to client

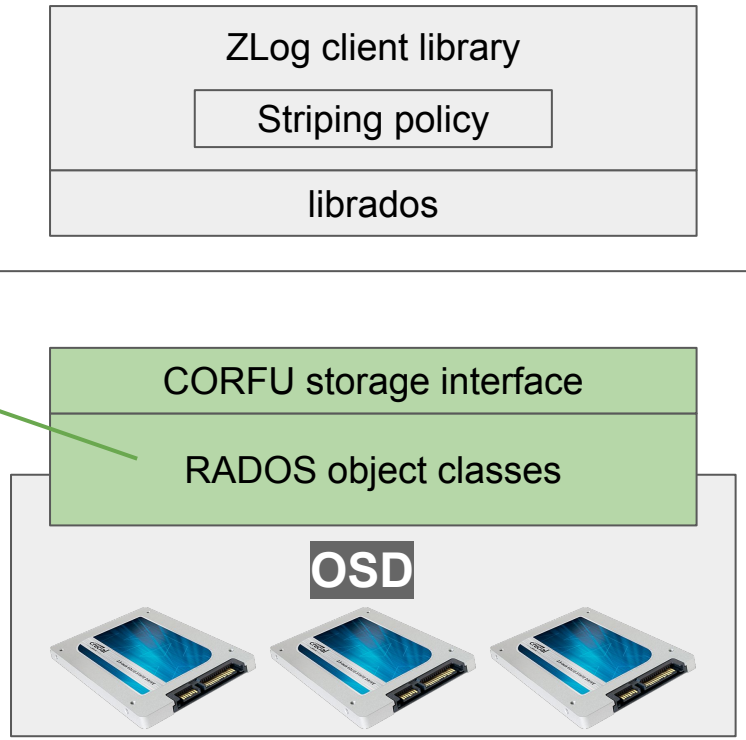
```
int write(context_t hctx, bufferlist *in, bufferlist *out) {
 // ensure up-to-date client
 if (epoch_guard(header, op.epoch(), false)) {
   return -ESPIPE;
 }

 // read entry based on request position
 ceph::bufferlist bl;
 int ret = cls_cxx_map_get_val(hctx, key, &bl);
 if (ret < 0)
   return ret;
 if (entry && !decode(bl, entry))
   return -EIO;
 return 0;

 // write entry if position is not taken
 if (ret == -ENOENT) {
   entry_write_entry(hctx, key, entry);

   if (!header.has_max_pos() || (op.pos() > header.max_pos()))
     header.set_max_pos(op.pos());

   ret = entry_write_header(hctx, header);
 } else {
  // handle errors associated with write-once semantics
 }
}
```



CORFU write interface as an object class

connection to client

```
int write(context_t hctx, bufferlist *in, bufferlist *out) {
 // ensure up-to-date client
 if (epoch_guard(header, op.epoch(), false)) {
   return -ESPIPE;
 }

 // read entry based on request position
 ceph::bufferlist bl;
 int ret = cls_cxx_map_get_val(hctx, key, &bl);
 if (ret < 0)
   return ret;
 if (entry && !decode(bl, entry))
   return -EIO;
 return 0;

 // write entry if position is not taken
 if (ret == -ENOENT) {
   entry_write_entry(hctx, key, entry);

   if (!header.has_max_pos() || (op.pos() > header.max_pos()))
     header.set_max_pos(op.pos());

   ret = entry_write_header(hctx, header);
 } else {
  // handle errors associated with write-once semantics
 }
}
```



Getting started with object classes

1. The *hello world* object class (in Ceph: src/cls/hello/cls_hello.cc)
    a. Super well documented, easy examples

Getting started with object classes

1. The *hello world* object class (in Ceph: src/cls/hello/cls_hello.cc)
   a. Super well documented, easy examples
2. Build and load the plugin into Ceph

Getting started with object classes

1. The *hello world* object class (in Ceph: src/cls/hello/cls_hello.cc)
   a. Super well documented, easy examples
2. Build and load the plugin into Ceph
3. Old way to manage plugins
   a. Manage your own Ceph fork

Getting started with object classes

1. The *hello world* object class (in Ceph: src/cls/hello/cls_hello.cc)
   a. Super well documented, easy examples
2. Build and load the plugin into Ceph
3. Old way to manage plugins
   a. Manage your own Ceph fork
4. New way to manage plugins with SDK (credit: Neha Ojha @ b7215b0)
   a. **dnf install rados-objclass-devel**
   b. **#include <rados/objclass.h>**
   c. **compile cls_hello.cc as object library**

Getting started with object classes

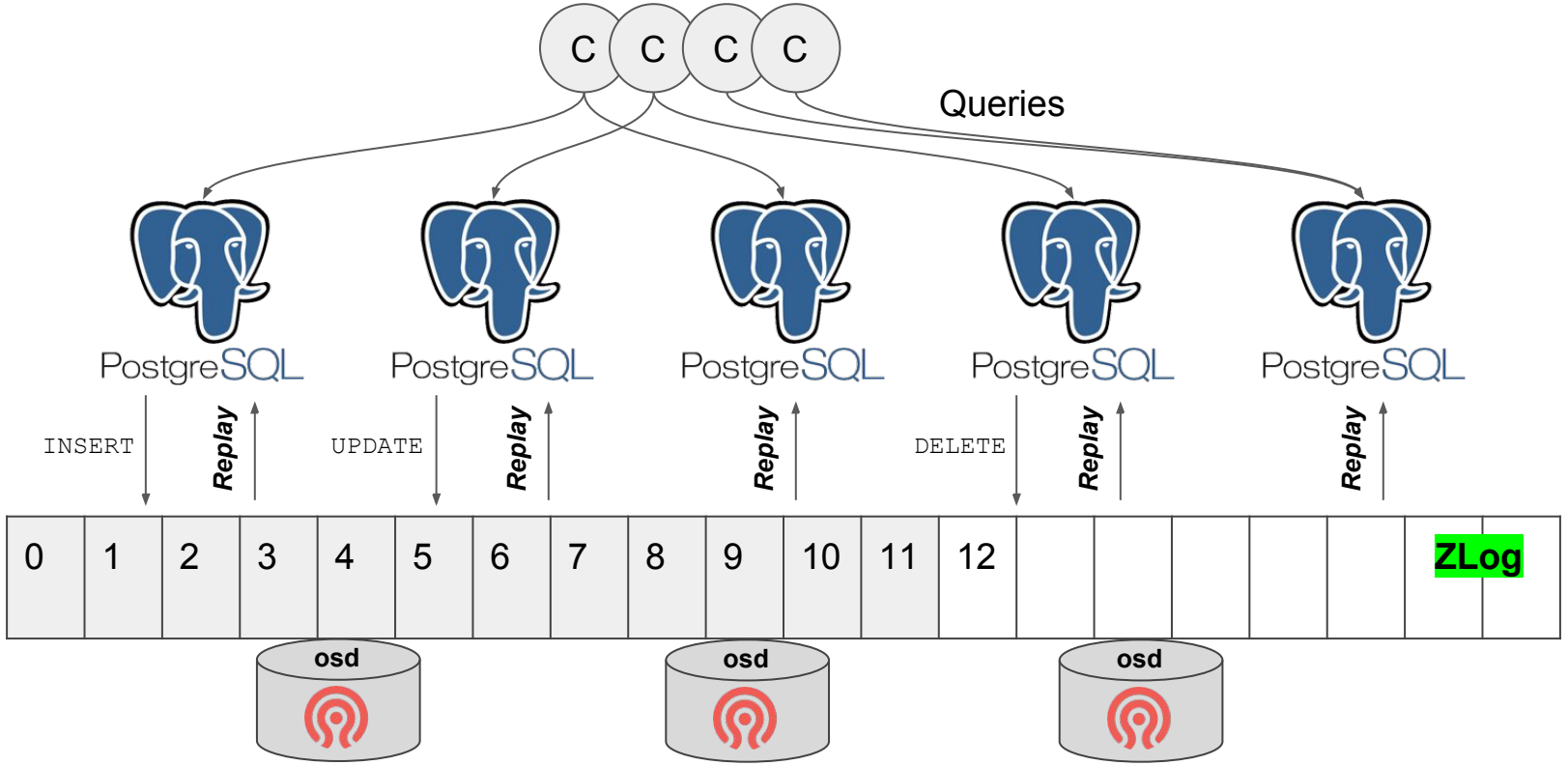
1. The *hello world* object class (in Ceph: src/cls/hello/cls_hello.cc)
   a. Super well documented, easy examples
2. Build and load the plugin into Ceph
3. Old way to manage plugins
   a. Manage your own Ceph fork
4. New way to manage plugins with SDK (credit: Neha Ojha @ b7215b0)
   a. **dnf install rados-objclass-devel**
   b. **#include <rados/objclass.h>**
   c. **compile cls_hello.cc as object library**
5. Distribute your plugin to OSDs at **<libdir>/rados-classes/cls_hello.so**
6. Starting making requests **ioctx::exec("hello", …)**

The CORFU *seal(obj, epoch)* interface is tougher

- Semantics are *do the following atomically*
  - Verify and update the stored epoch
  - Return the largest position written
- RADOS object classes don’t support this mix of ops
  - *Currently*
- Solution
  - Split the operation
  - Verify
- Luckily this isn’t a fast path

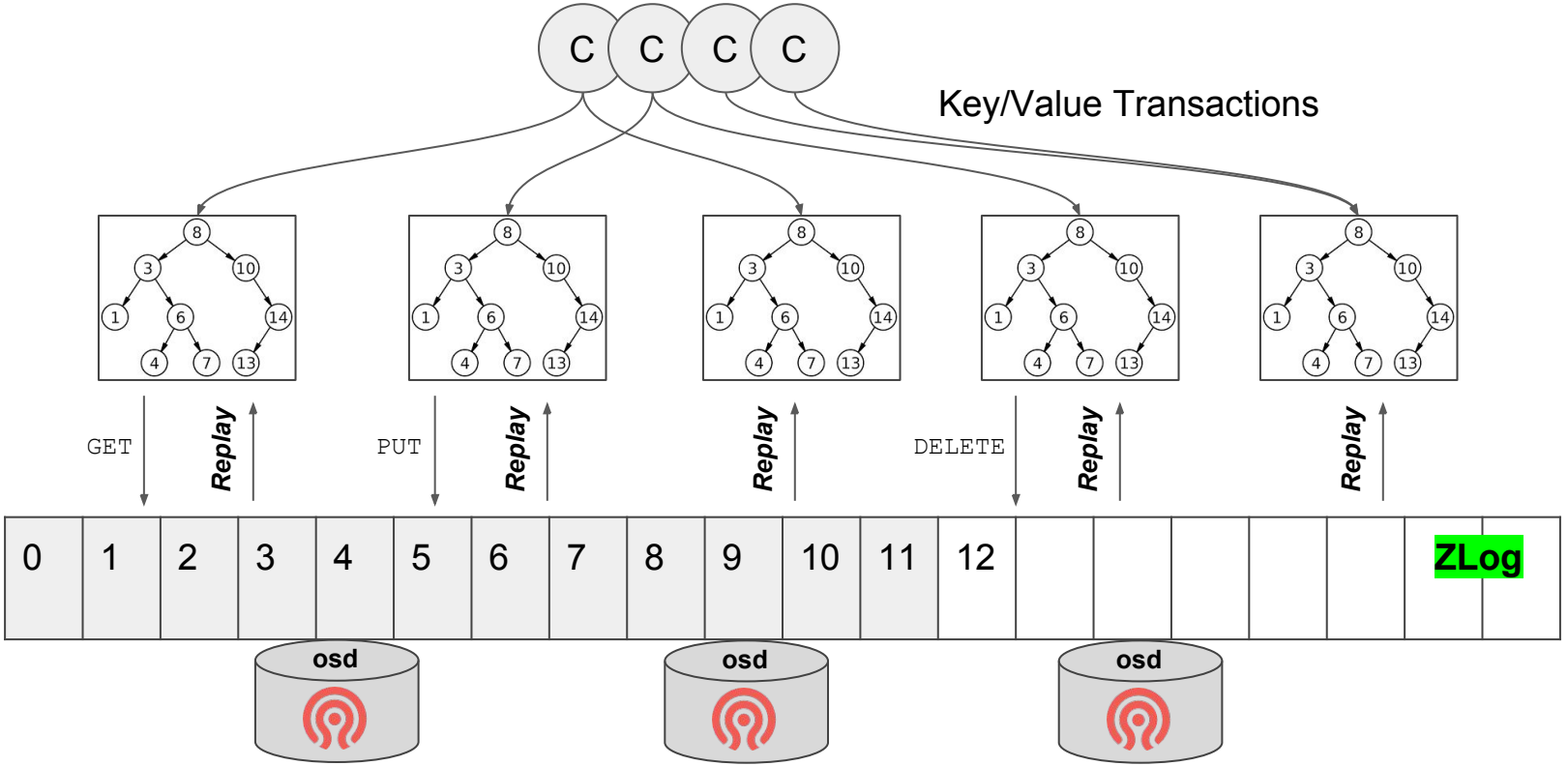
How we have been using ZLog

PostgreSQL logical replication with ZLog



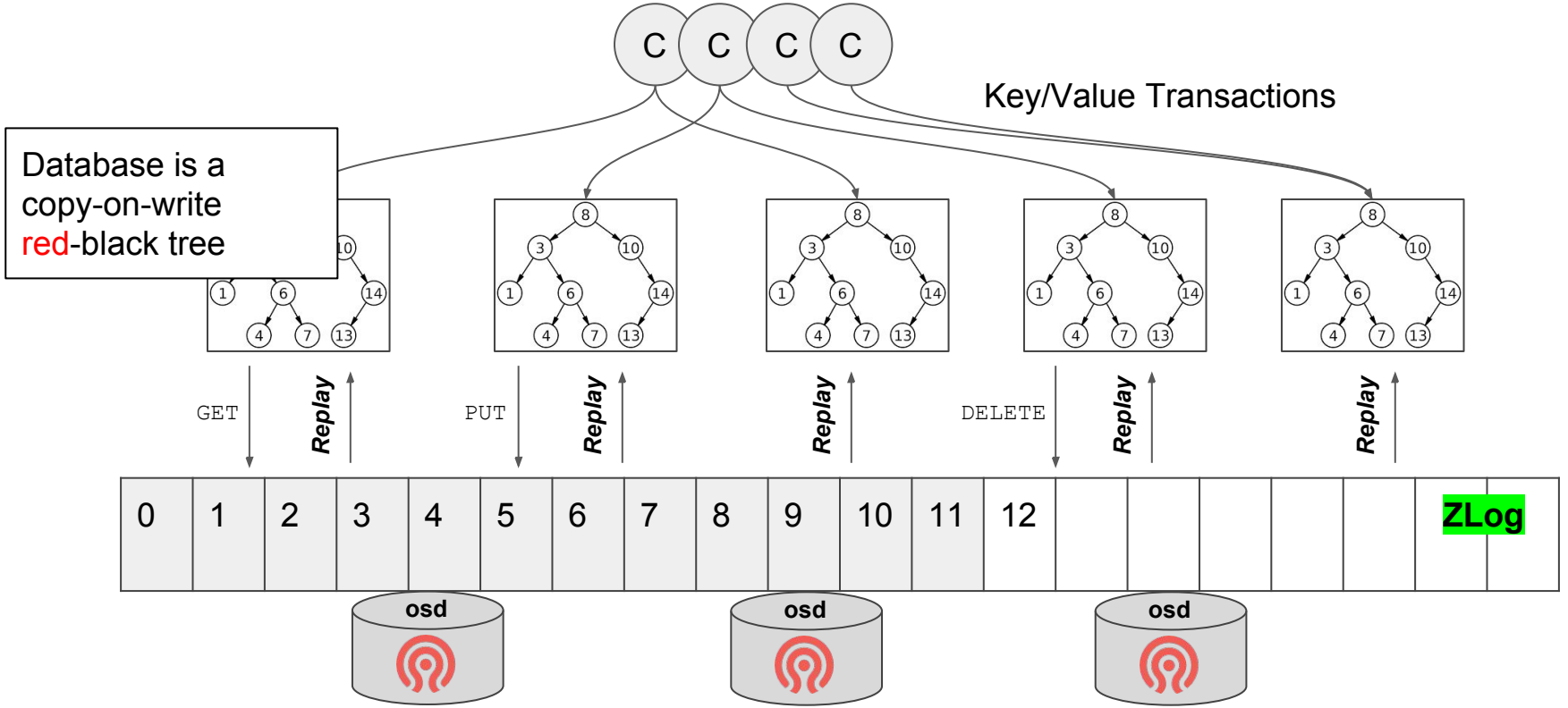
https://github.com/cruzdb/pg_zlog

CruzDB is an immutable key-value store on ZLog



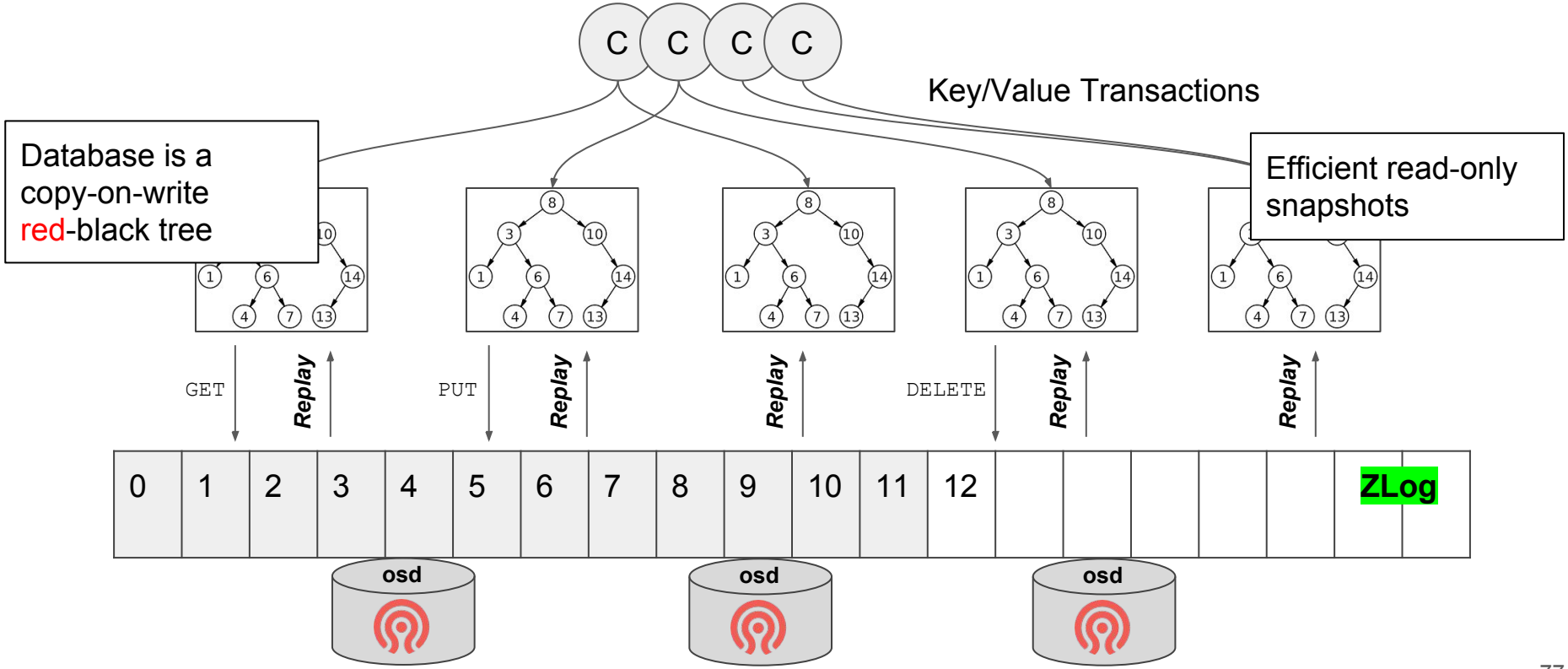
https://github.com/cruzdb/cruzdb

CruzDB is an immutable key-value store on ZLog



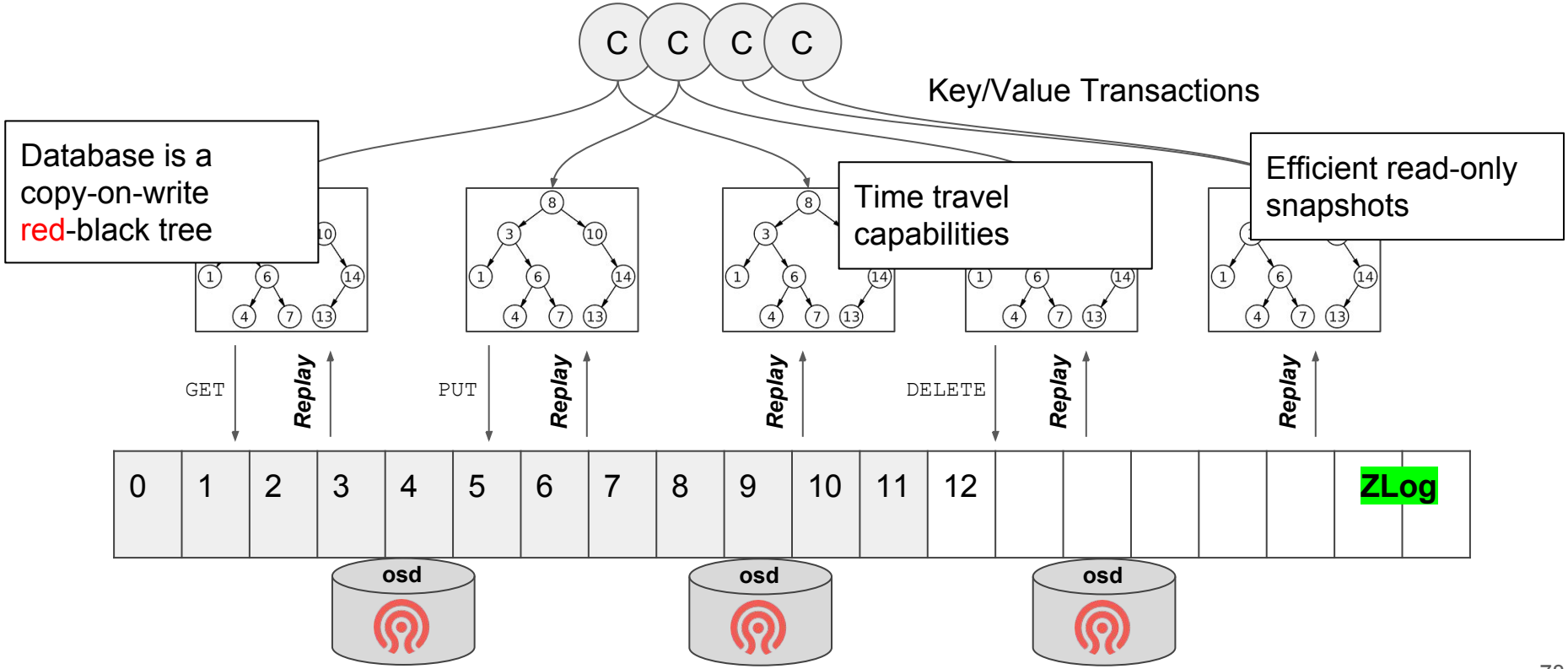
https://github.com/cruzdb/cruzdb

CruzDB is an immutable key-value store on ZLog



https://github.com/cruzdb/cruzdb

CruzDB is an immutable key-value store on ZLog



https://github.com/cruzdb/cruzdb

Wrapping up

- We use Ceph as a prototyping system
  - Application-specific interfaces
- There is a broad range of interests in log-oriented interfaces
- We've built a Ceph-based implementation of a high-performance log
  - ZLog @ https://github.com/cruzdb/zlog
- Using it for several real-world use cases
  - PostgreSQL logical replication (https://github.com/cruzdb/pg_zlog)
  - CruzDB immutable database (https://github.com/cruzdb/cruzdb)

Thank you and questions

- **Noah Watkins (@noahdesu)**
- Learning resources
  - Object class SDK documentation
    - http://docs.ceph.com/docs/master/rados/api/objclass-sdk/
  - ZLog is the only user of the SDK I'm aware of
    - https://github.com/cruzdb/zlog/blob/master/src/storage/ceph/cls_zlog.cc
  - The Ceph tree contains a ton of object classes for reference
    - https://github.com/ceph/ceph/tree/master/src/cls
  - Writing object classes using Lua
    - https://ceph.com/geen-categorie/dynamic-object-interfaces-with-lua/
    - https://nwat.xyz/blog/2018/03/13/video-of-my-talk-at-lua-workshop-2017/