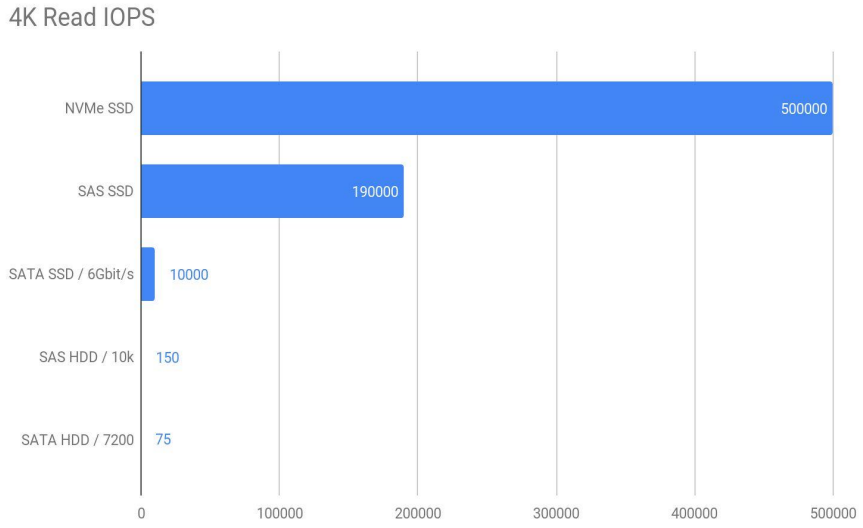
Crimson

A new osd for the age of persistent memory and fast nvme storage

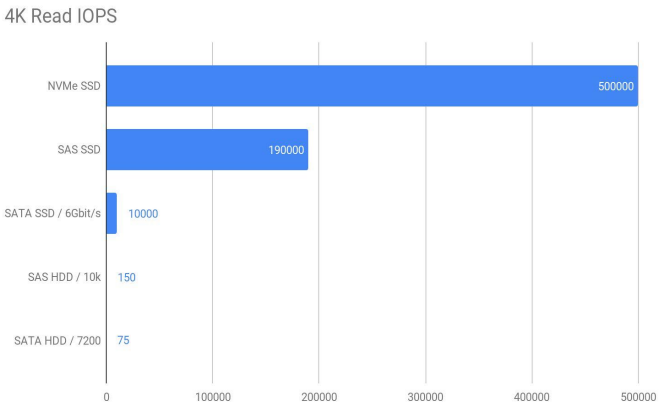
Storage Keeps Getting Faster

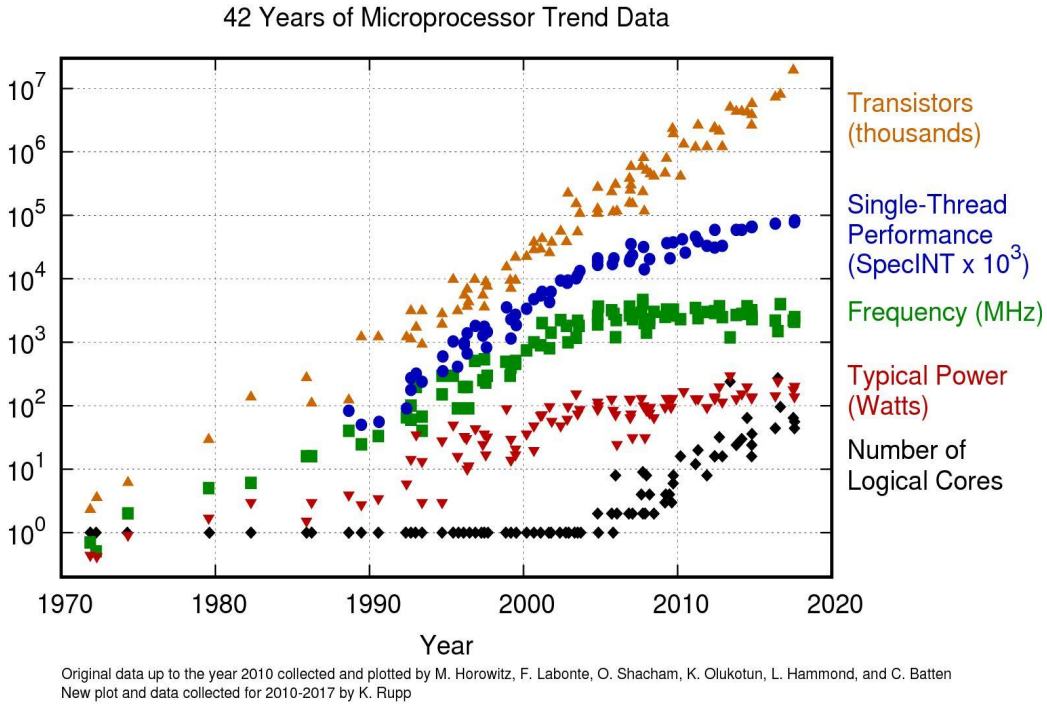


CPUs, not so much

With a cpu clocked at 3ghz you can afford:

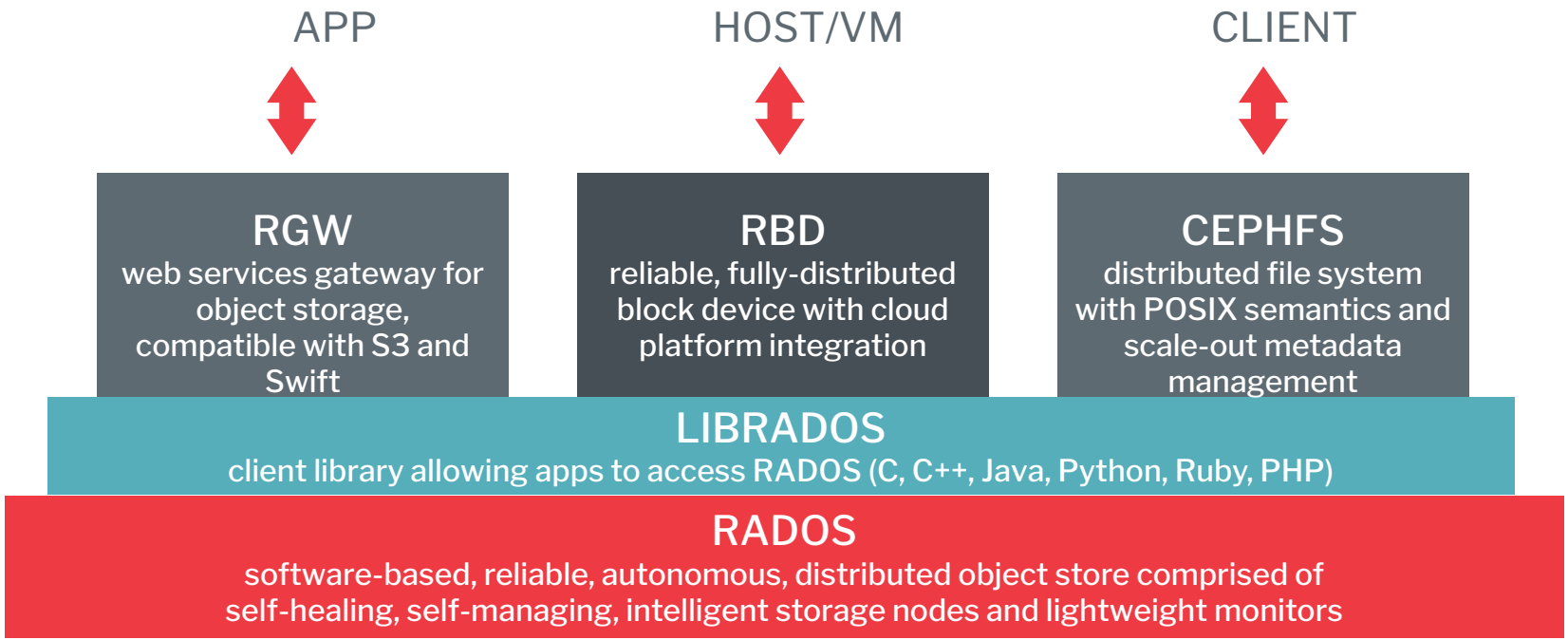
- HDD: ~20 million cycles/IO
- SSD: 300,000 cycles/IO
- NVME: 6000 cycles/IO



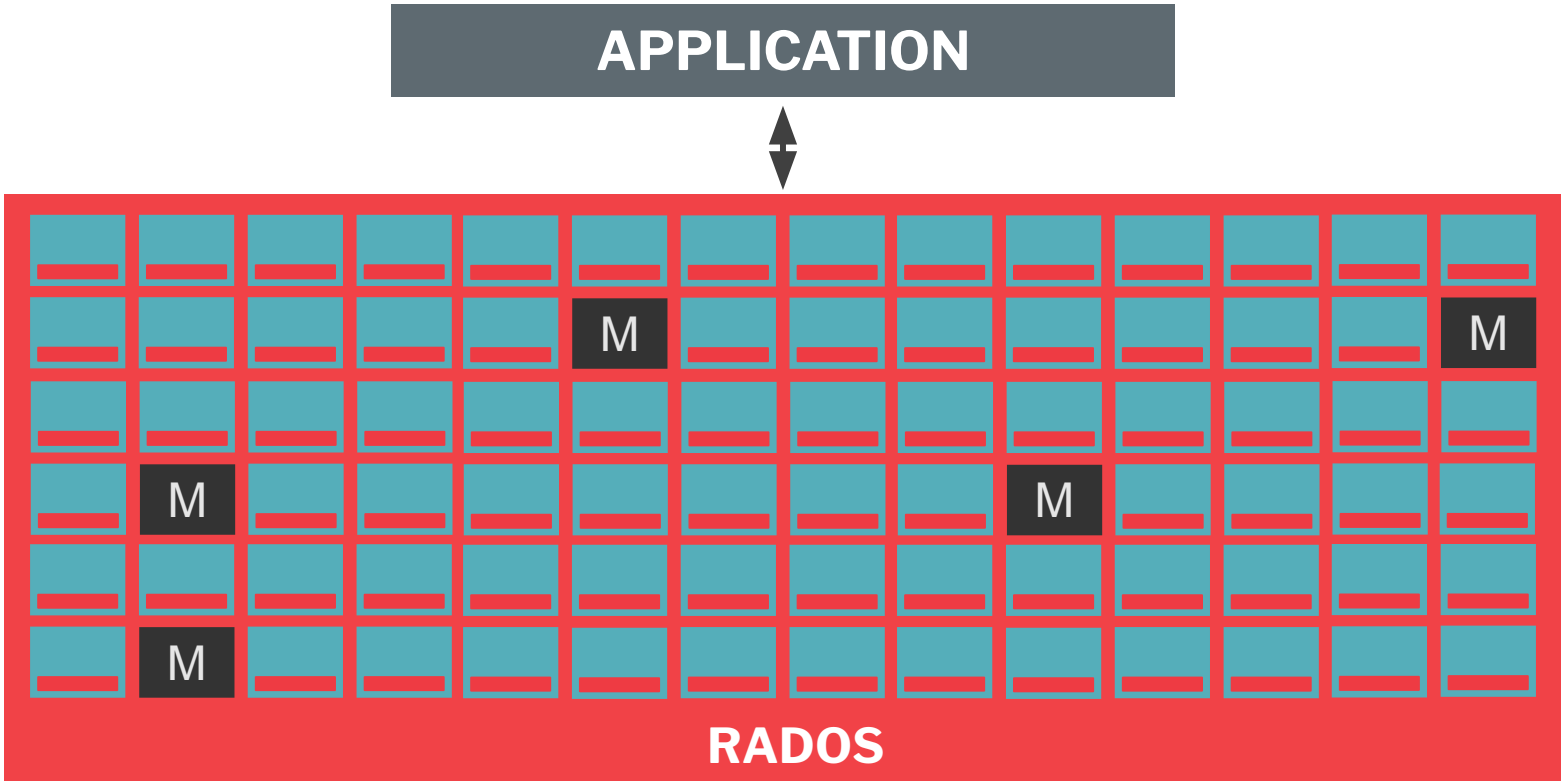


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

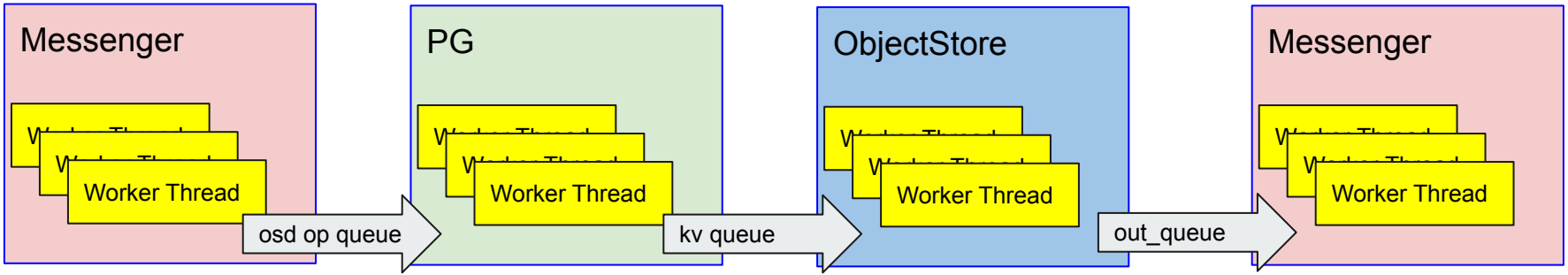
Ceph Architecture



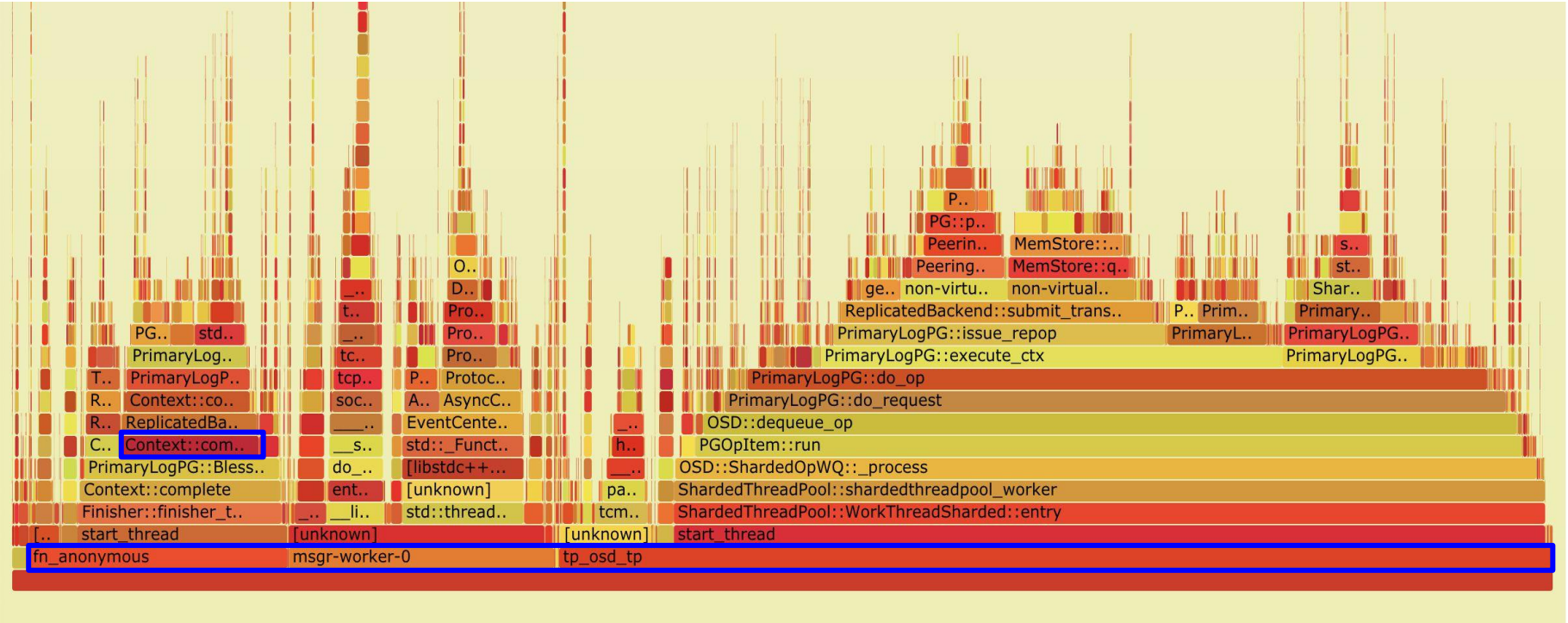
RADOS



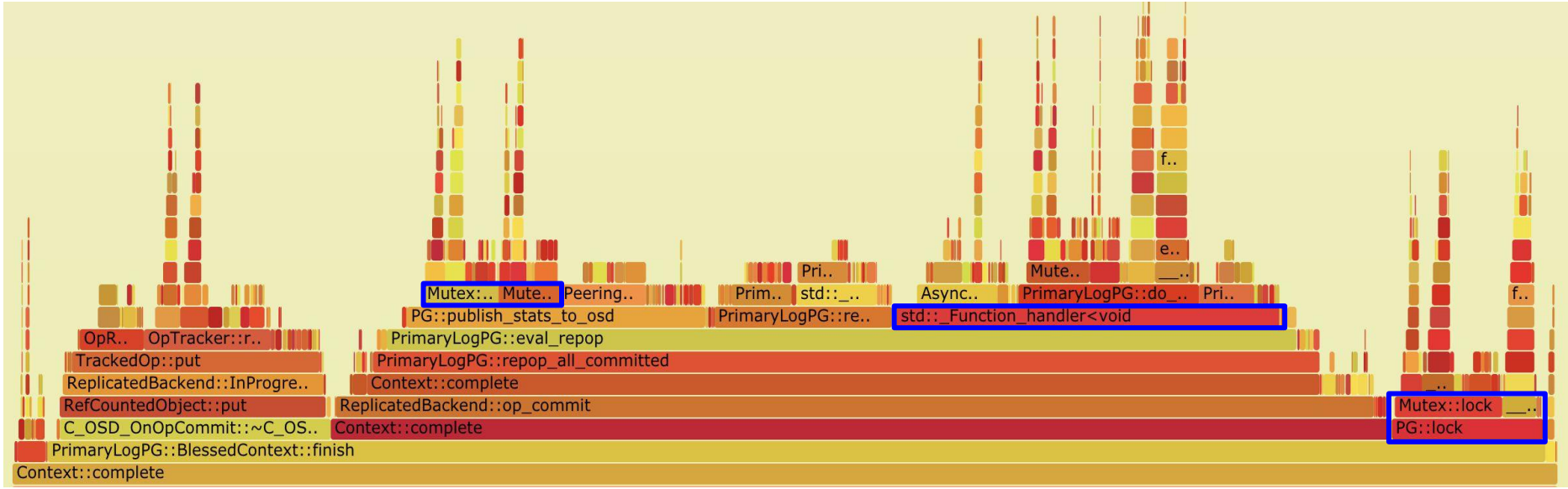
OSD Threading Architecture



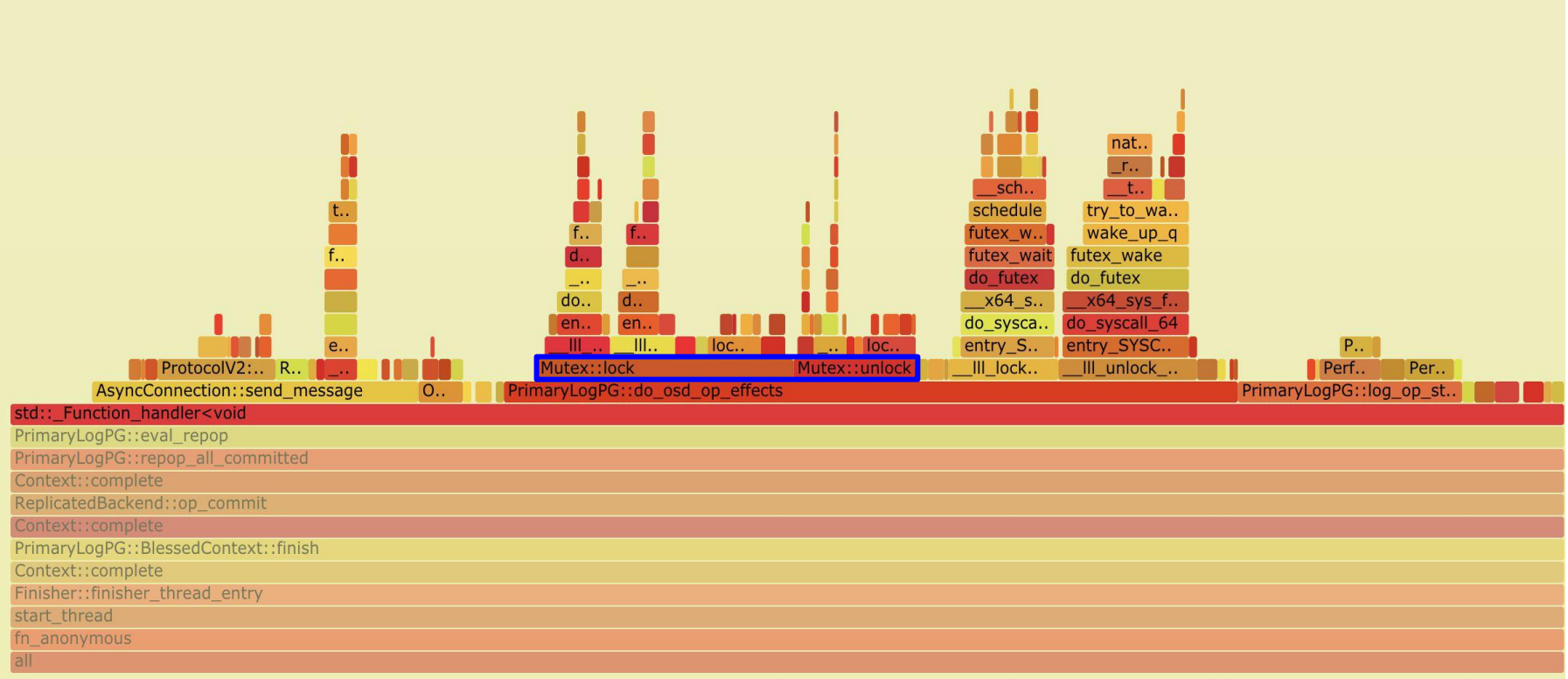
Classic OSD Threads



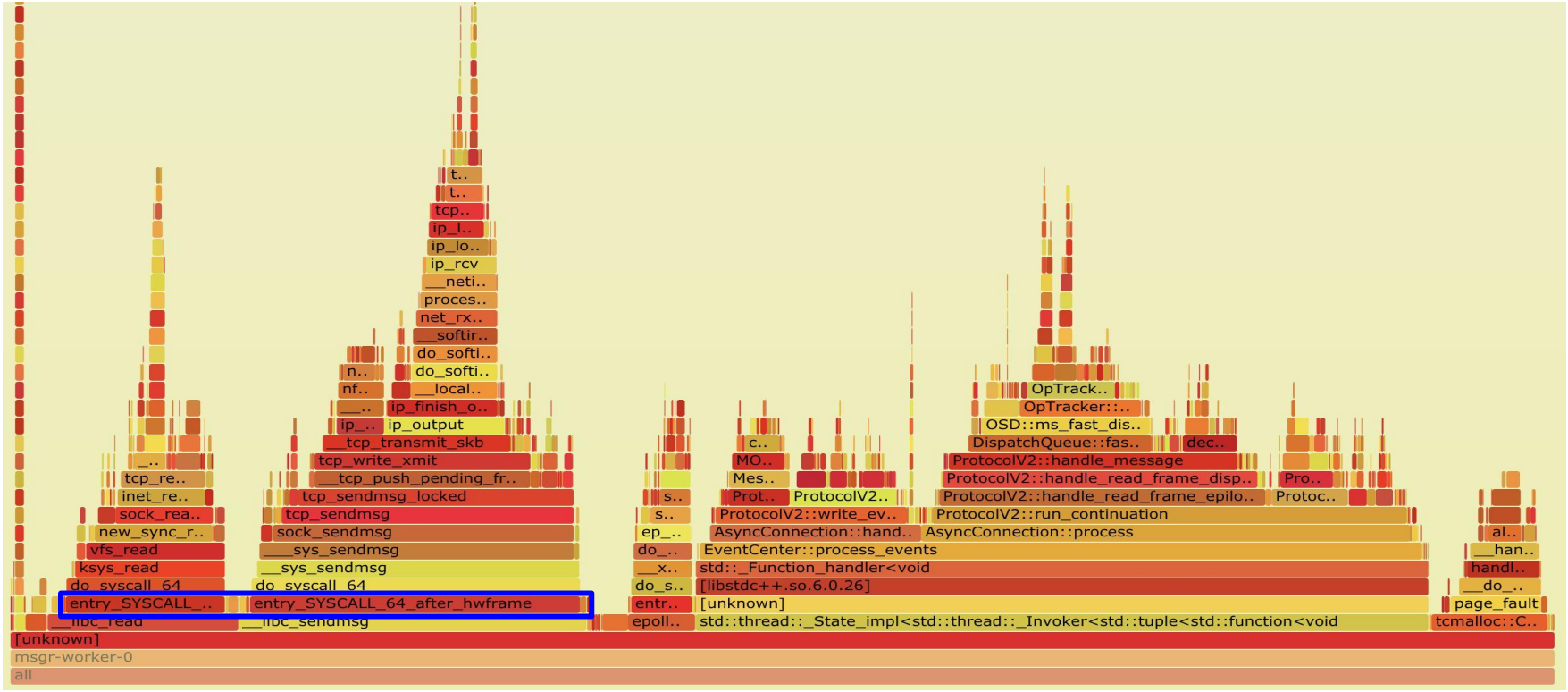
Classic OSD Threads



Classic OSD Threads



Classic OSD Threads



Crimson!

crimson-osd aims to be a replacement osd daemon with the following goals:

- Minimize cpu overhead
  - Minimize cycles/iop
  - Minimize cross-core communication
  - Minimize copies
  - Bypass kernel, avoid context switches
- Enable emerging storage technologies
  - Zoned Namespaces
  - Persistent Memory
  - Fast NVME

Seastar

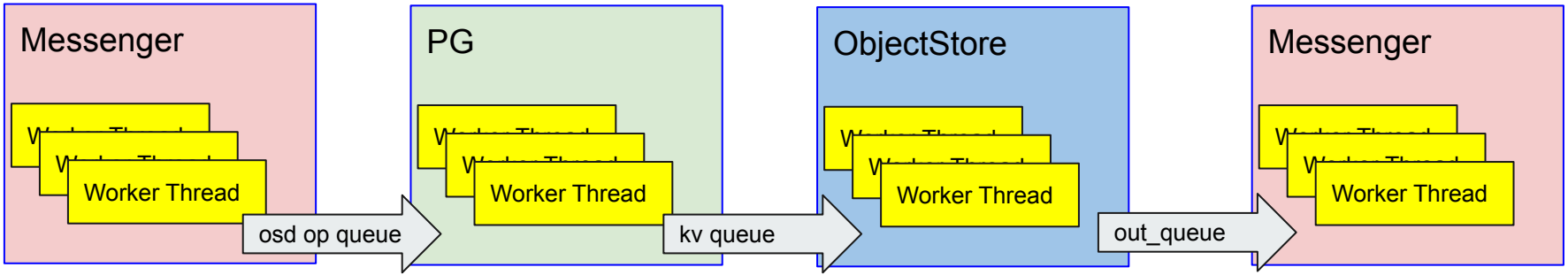
- Single thread per cpu
- Non-blocking IO
- Scheduling done in user space
- Includes direct support for DPDK, a high performance library for userspace networking

Programming Model

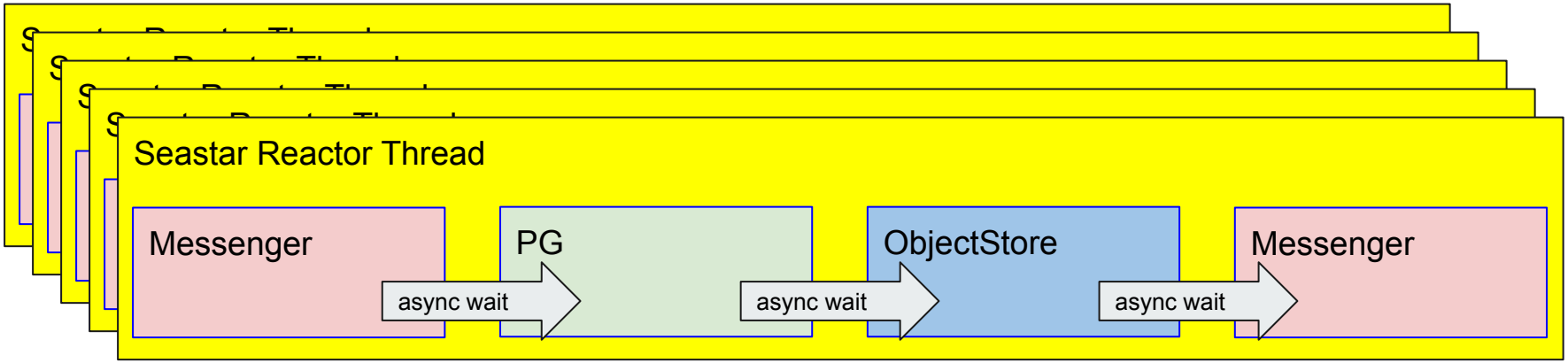
```
for (;;) {
  auto req = connection.read_request();
  auto result = handle_request(req);
  connection.send_response(result);
}
```

```
repeat([conn, this] {
  return conn.read_request().then([this](auto req)
{
    return handle_request(req);
  }).then([conn] (auto result) {
    return conn.send_response(result);
  });
});
```

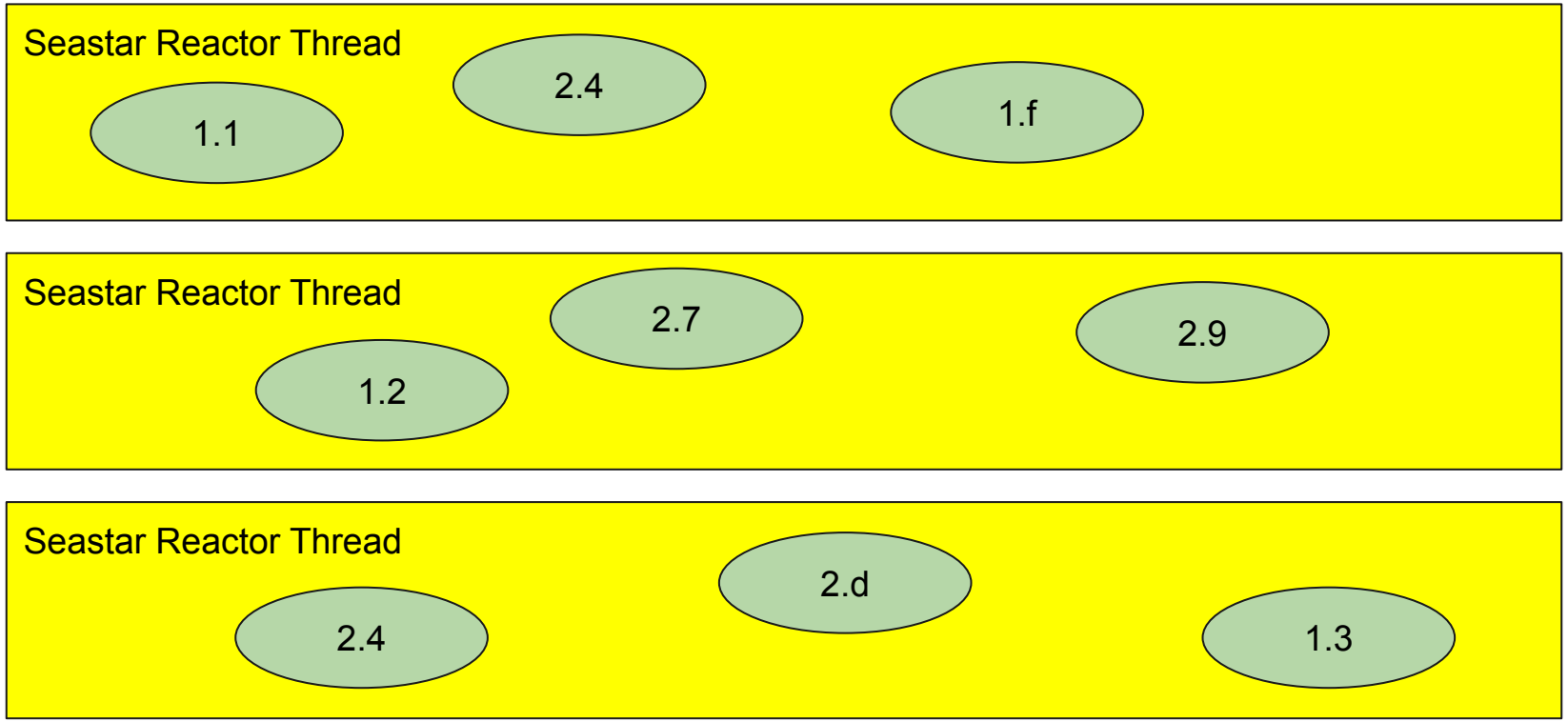
OSD Threading Architecture



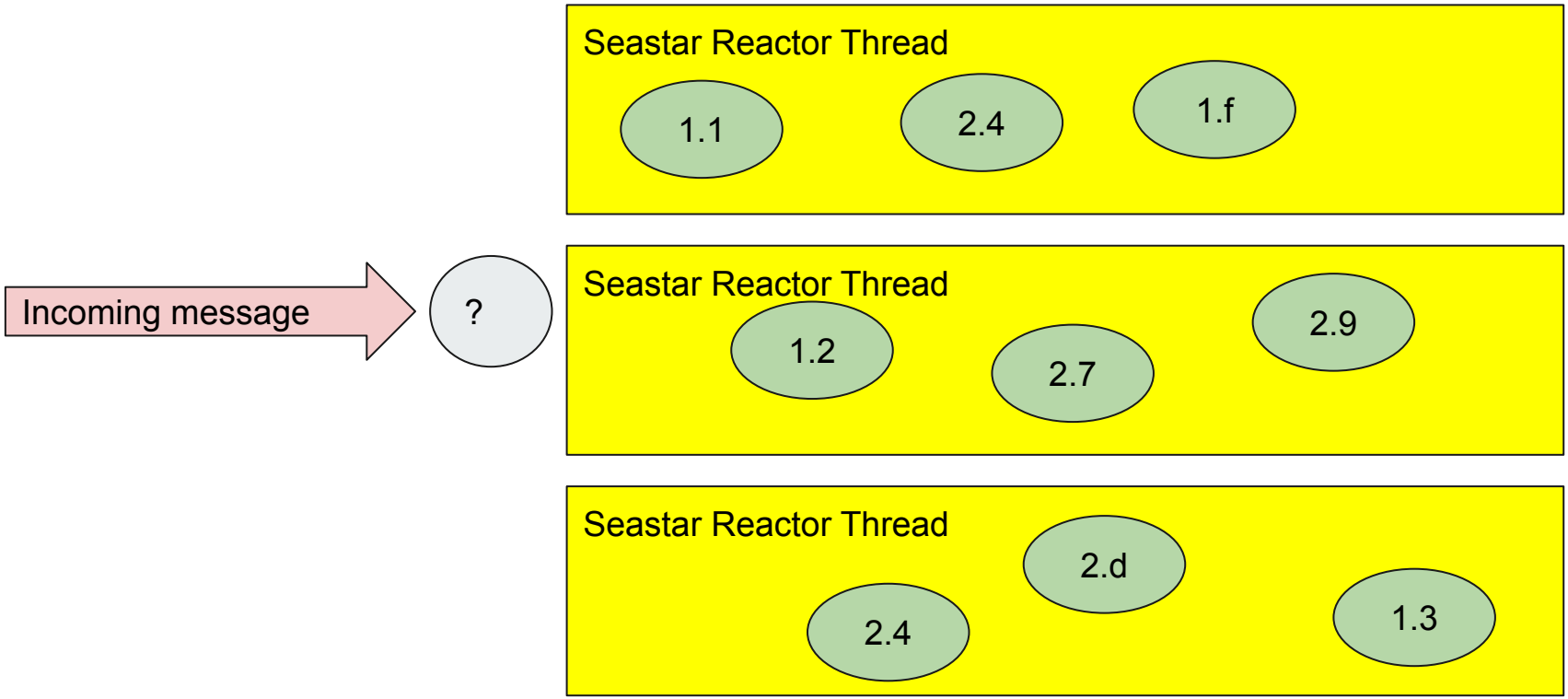
Crimson Threading Architecture



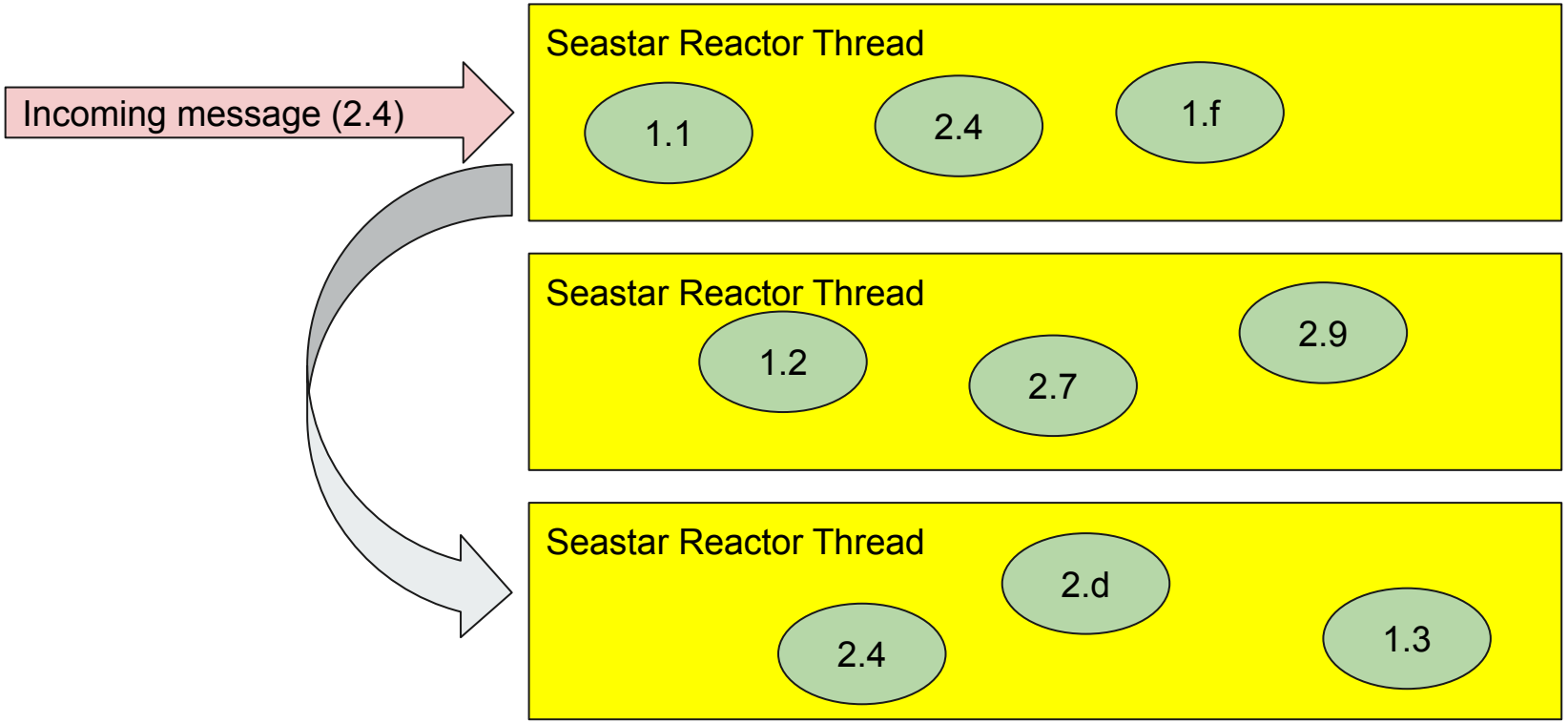
Crimson Threading Architecture



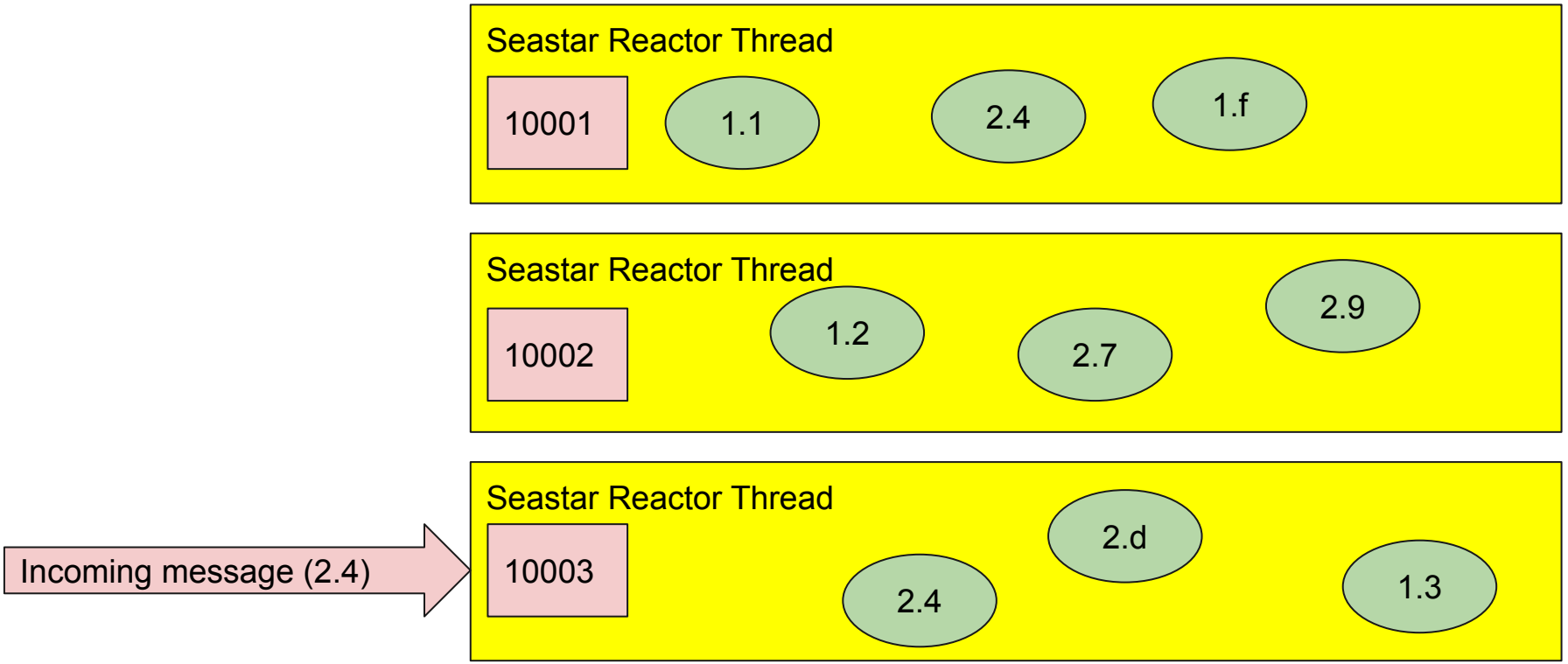
Crimson Threading Architecture



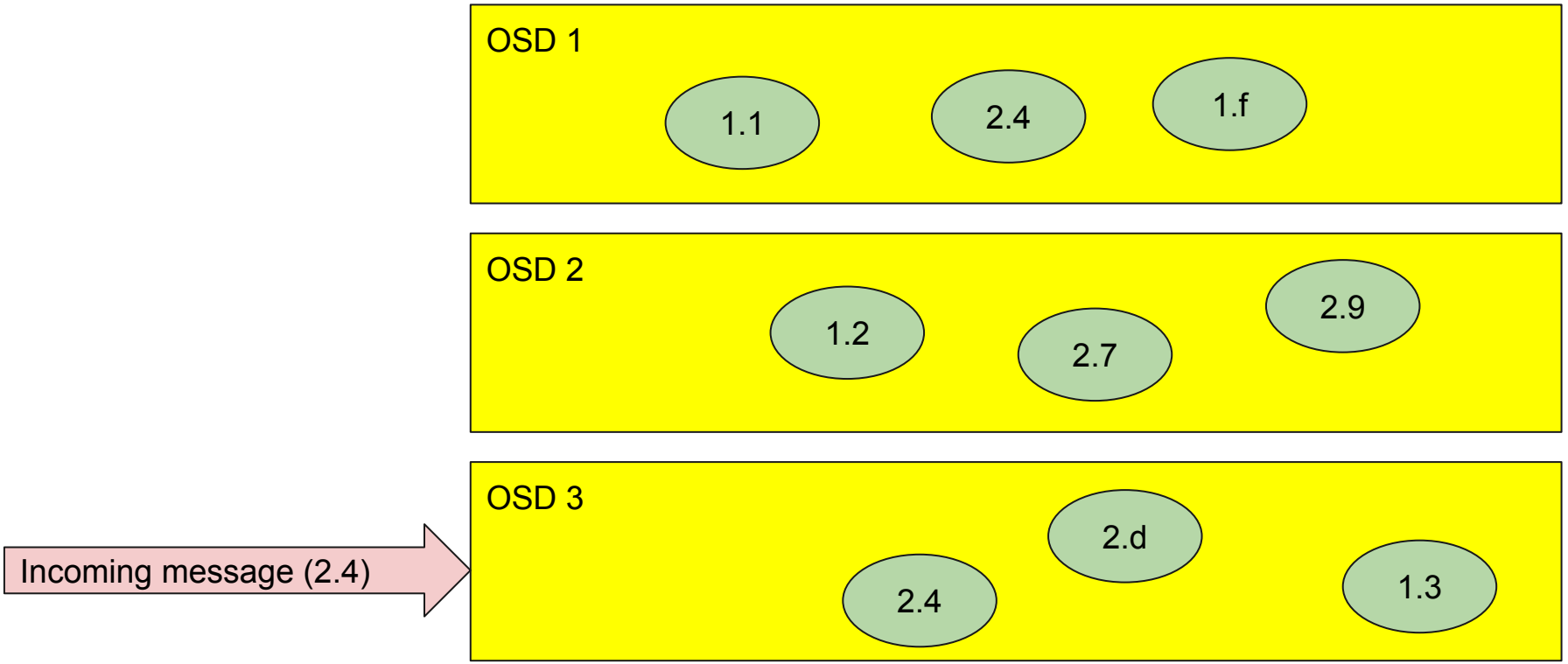
Crimson Threading Architecture



Crimson Threading Architecture



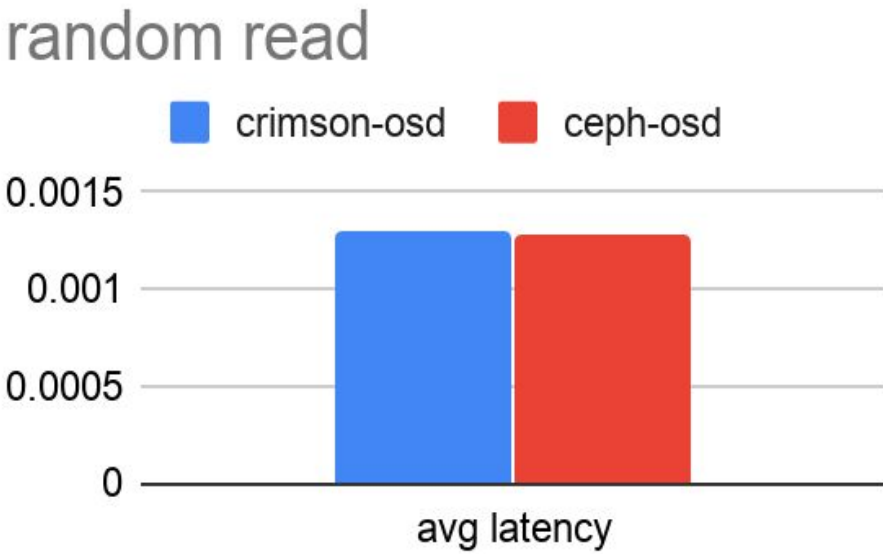
Crimson Threading Architecture

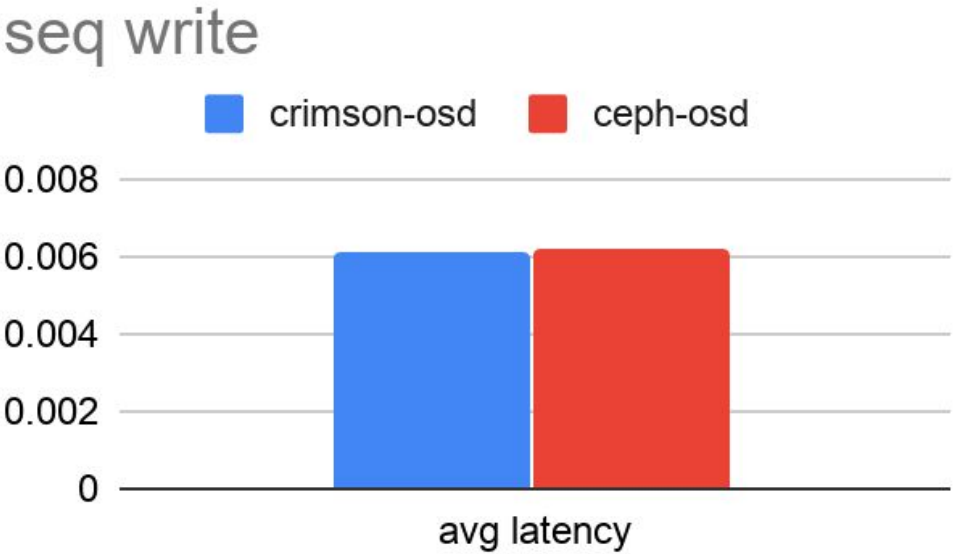


Current Status

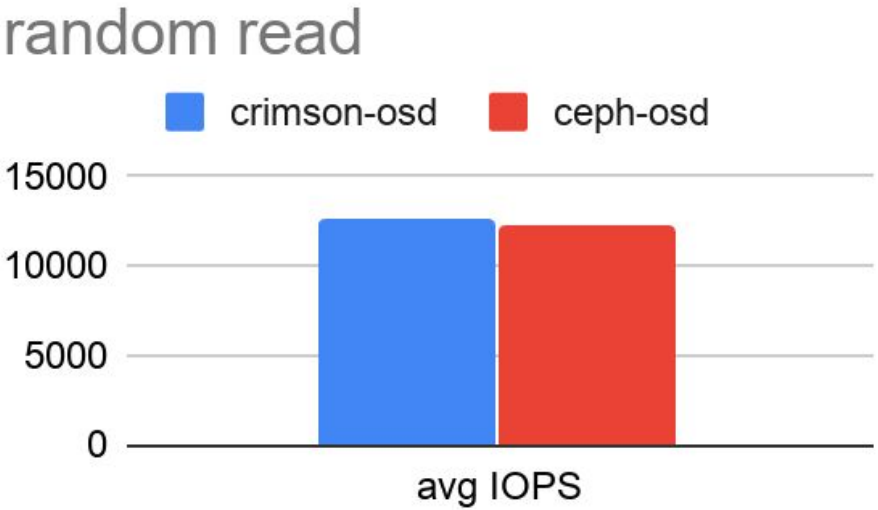
- Working:
  - Peering
  - Replication
  - Support for rbd based benchmarking
  - Memory-only ObjectStore shim
- In Progress:
  - Recovery/backfill
  - BlueStore support (via alien) -- almost ready to merge
  - Seastore implementation in progress

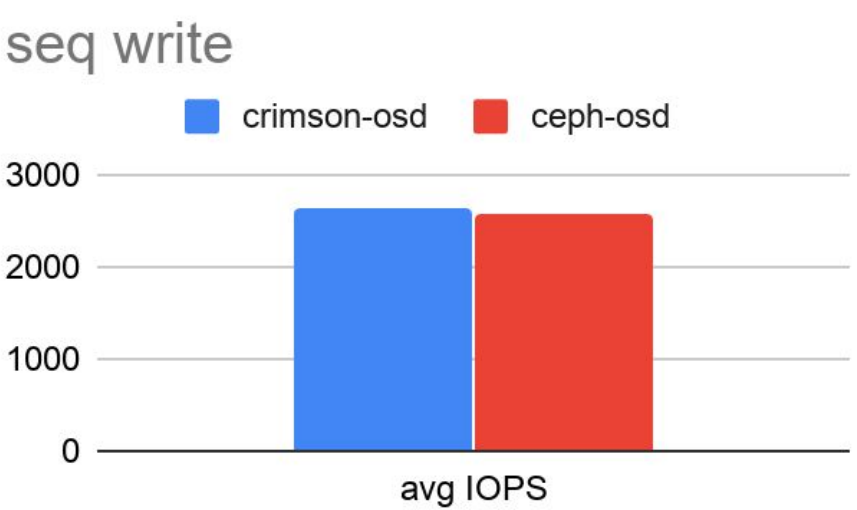
Performance



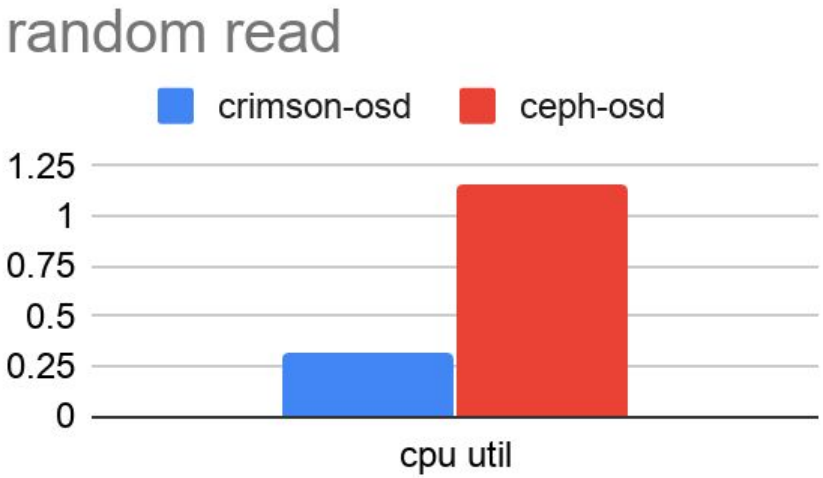


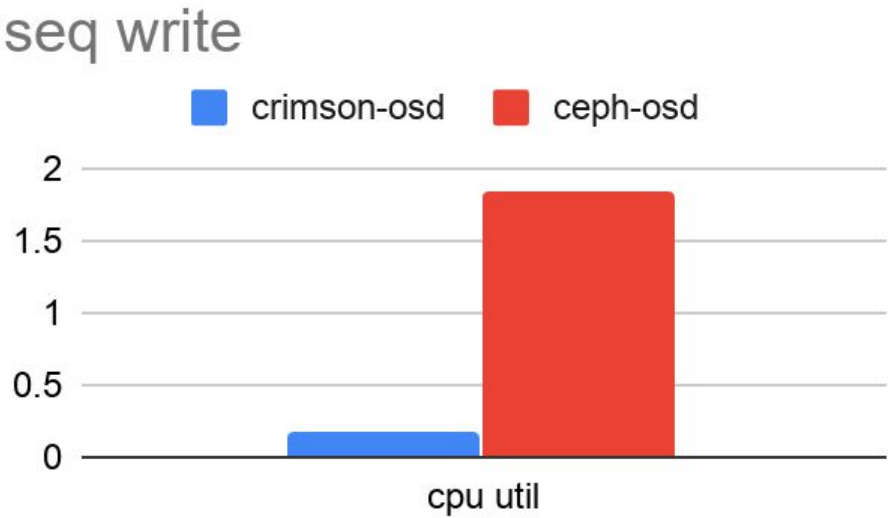
Performance



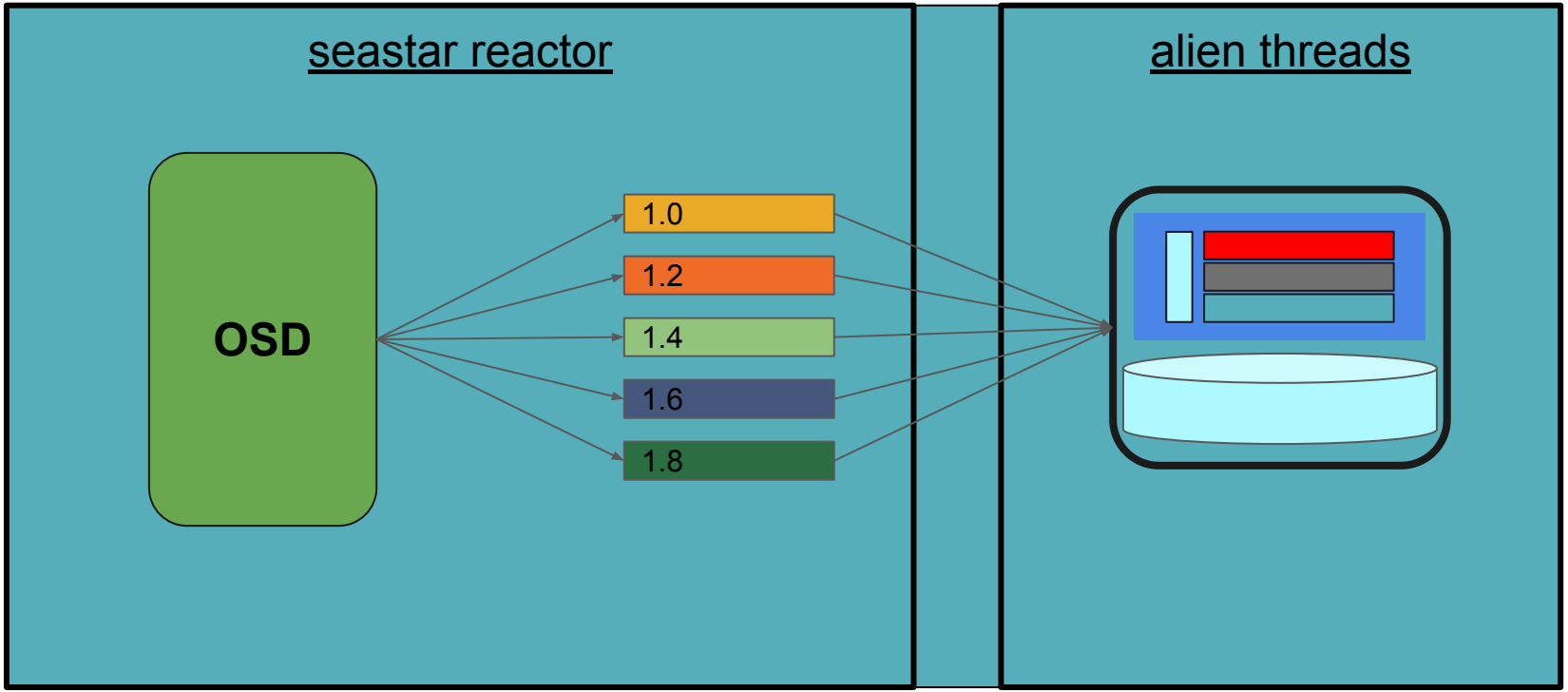


Performance

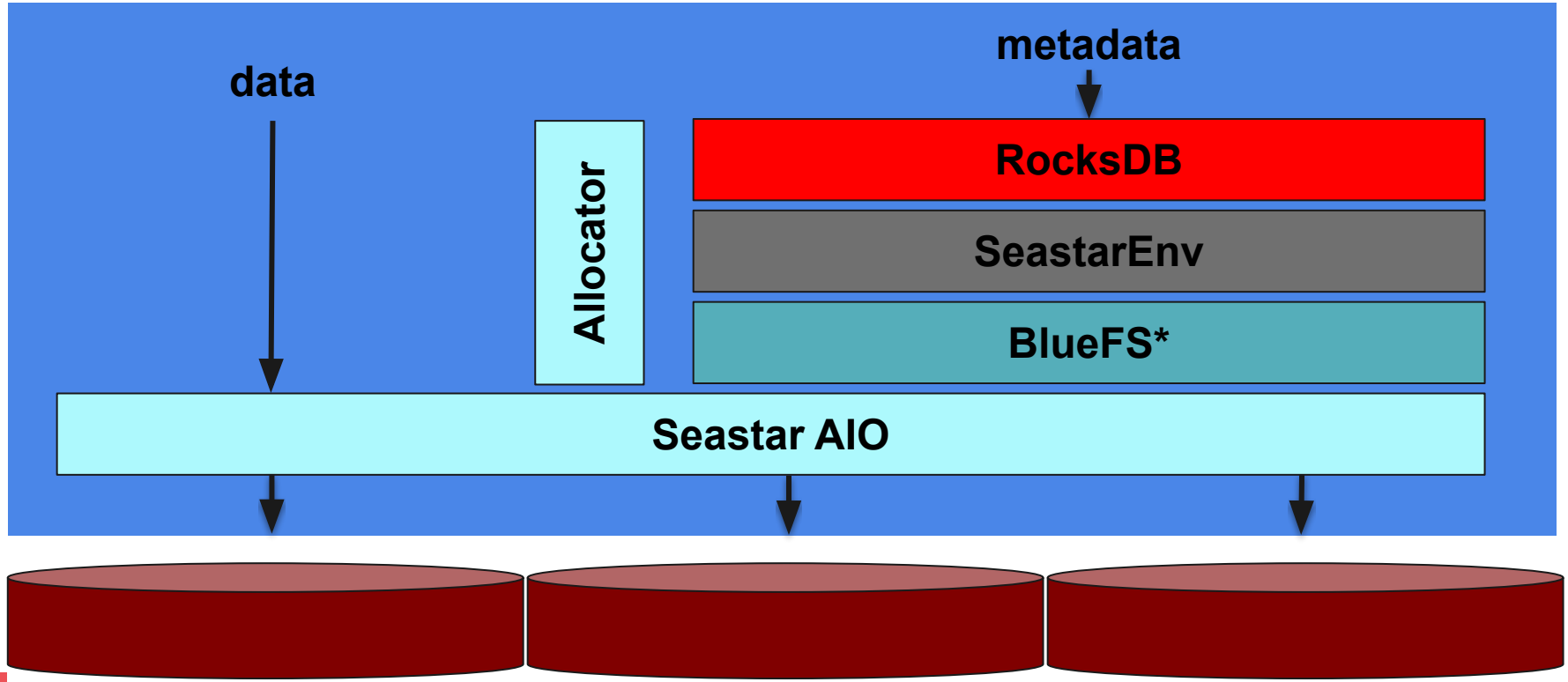




BlueStore - Alien



BlueStore - Seastar Native



Seastore

- Targets next generation storage technologies:
  - Zone Namespaces
  - Persistent Memory
- Independent metadata and allocation structures for each reactor to avoid synchronization

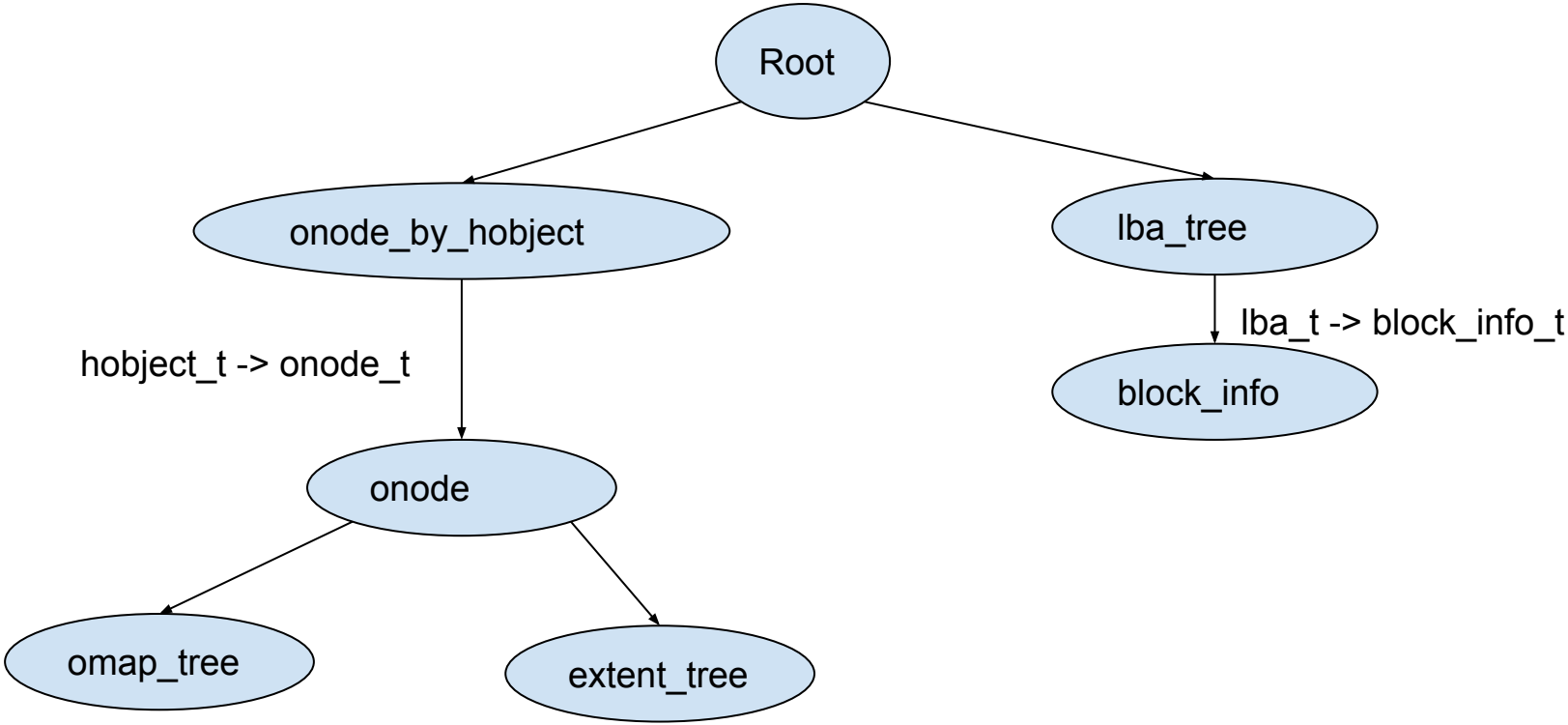
Seastore - ZNS

- New NVMe Specification
- Intended to address challenges with conventional FTL designs
  - High writes amplification, bad for qlc flash
  - Background garbage collection tends to impact tail latencies
- Different interface
  - Drive divided into zones
  - Zones can only be opened, written sequentially, closed, and released.
- As it happens, this kind of write pattern tends to be good for conventional ssds as well.

ObjectStore

- Transactional
- Composed of flat object namespace
- Object names may be large (>1k)
- Each object contains a key->value mapping (string->bytes) as well as a data payload.
- Supports COW object clones
- Supports ordered listing of both the omap and the object namespace

Seastore - Logical Structure



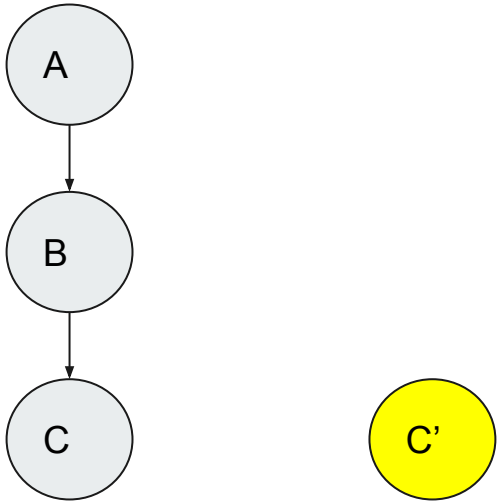
Seastore - Why use an LBA indirection?

When GCing a C, we need to do 2 things:

1. Find all incoming references (B in this case)
2. Write out a transaction updating (and dirtying) those references as well as writing out the new block C'

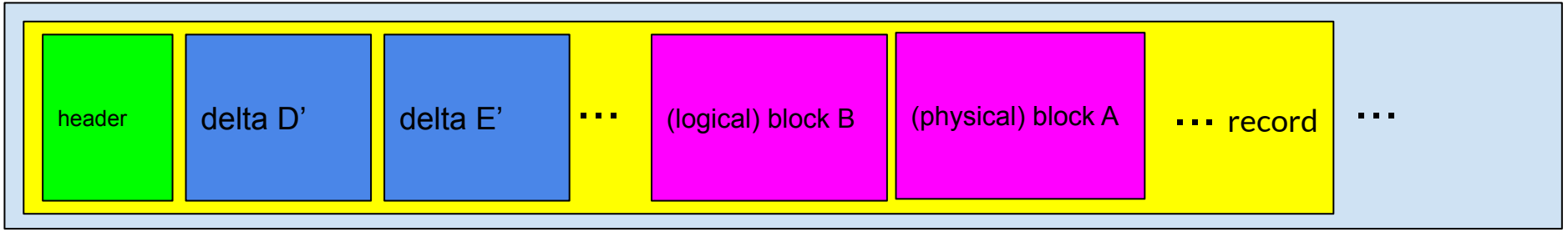
Using direct references means we still need to maintain some means of finding the parent references, and we pay the cost of updating the relatively low fanout onode and omap trees.

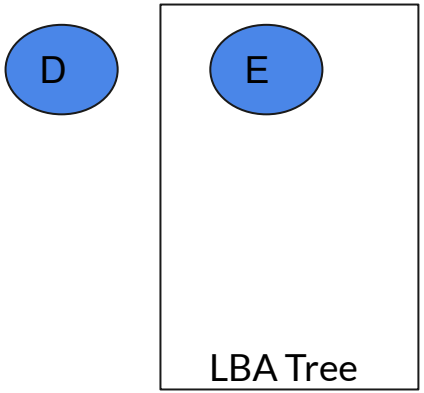
By contrast, using an LBA indirection requires extra reads in the lookup path, but potentially decreases reads and write amplification during gc.

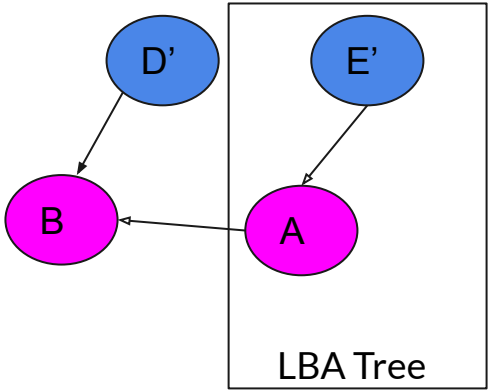


Seastore - Layout

Journal Segments

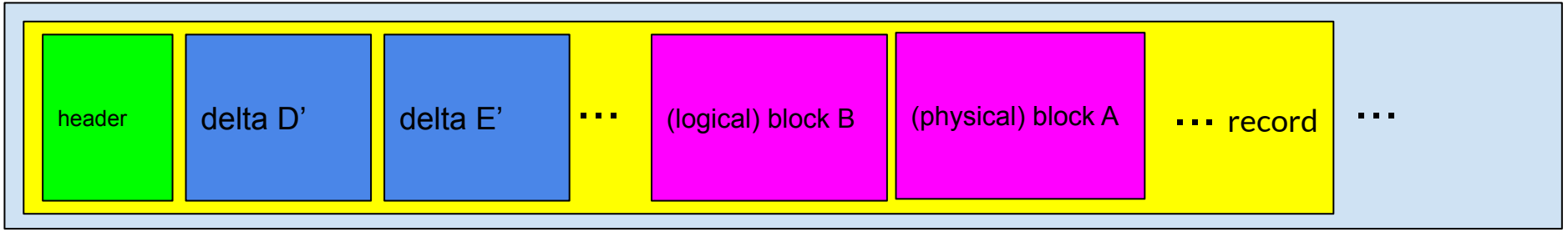


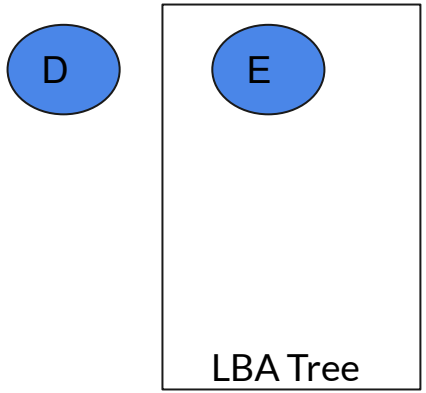


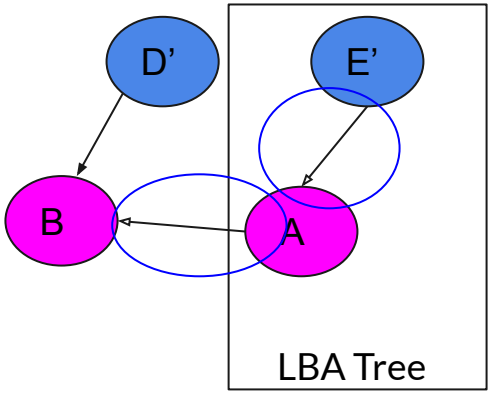


Seastore - ZNS

Journal Segments







Seastore - Persistent Memory

- Optane now available
  - Almost DRAM like read latencies
  - Write latency drastically lower than flash
  - Seems like a good fit for caching data and metadata!
  - Reads from persistent memory can simply return a ready future without waiting at all.
- Likely to be integrated into seastore as a replacement for metadata structures/journal deltas
- In particular, could be used to replace the lba mapping.

Questions?

- Roadmap: https://github.com/ceph/ceph-notes/blob/master/crimson/status.rst
- Project Tracker: https://github.com/ceph/ceph/projects/2
- Documentation: https://github.com/ceph/ceph/blob/master/doc/dev/crimson.rst
- Seastar tutorial: https://github.com/scylladb/seastar/wiki/Seastar-Tutorial