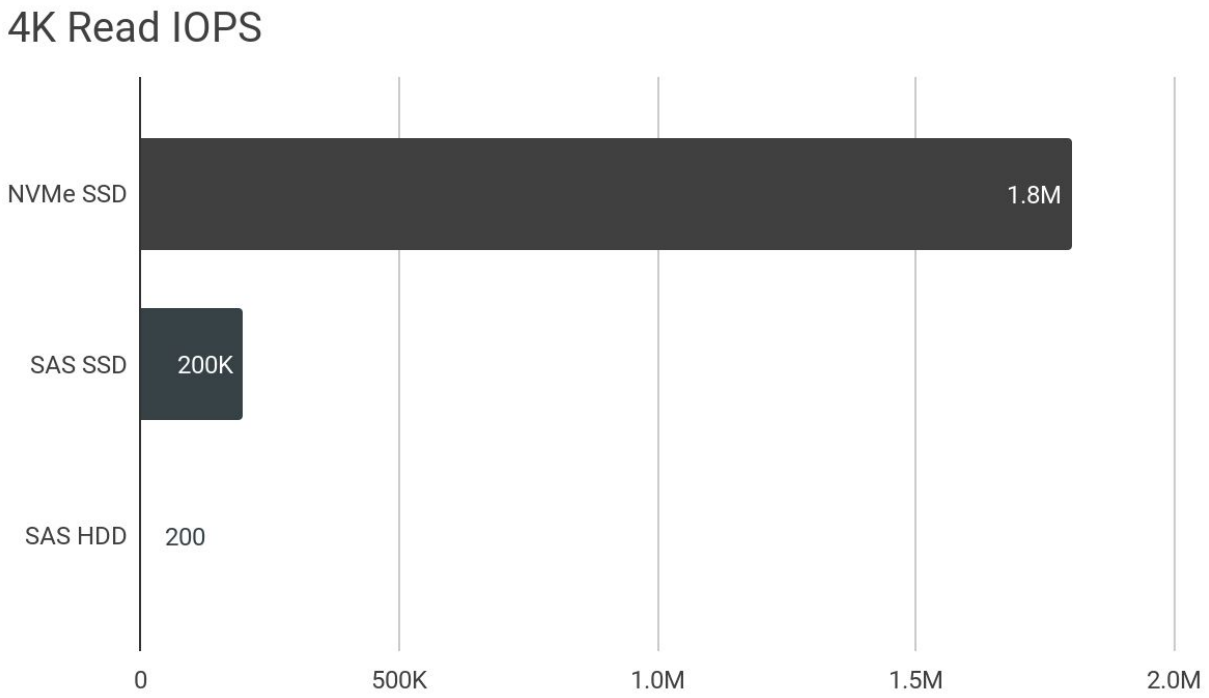
# Crimson Project
# Ceph Days London 2024

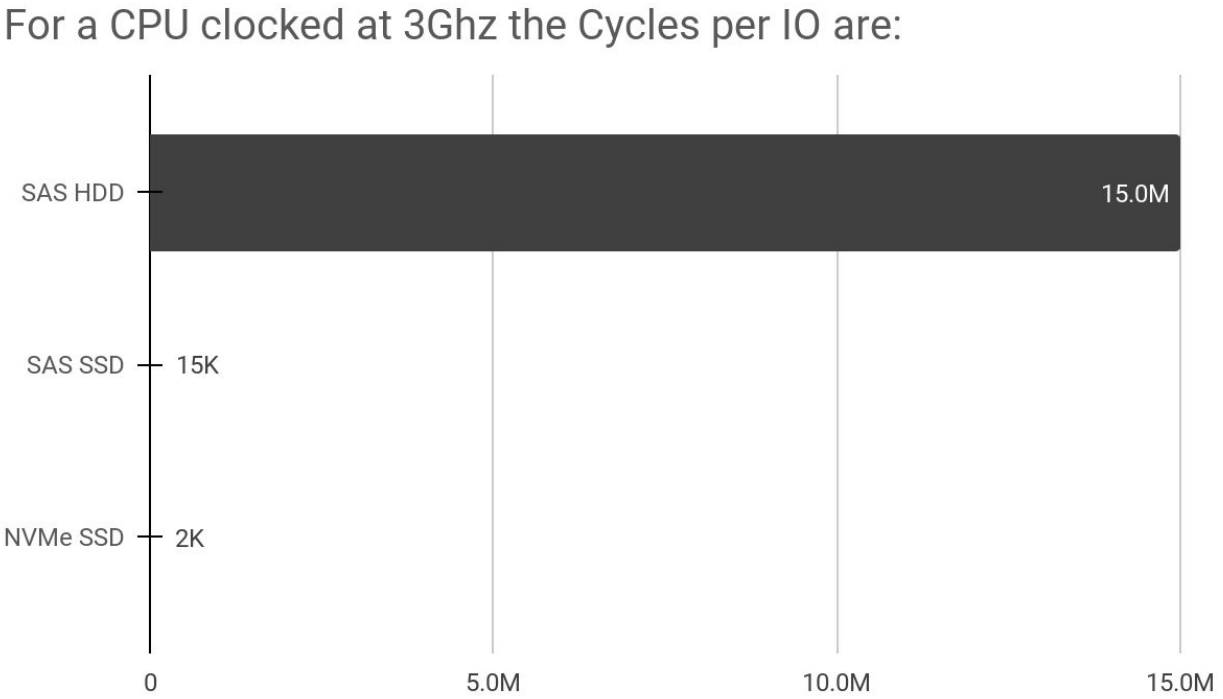
**Matan Breizman**

**17 July, 2024**

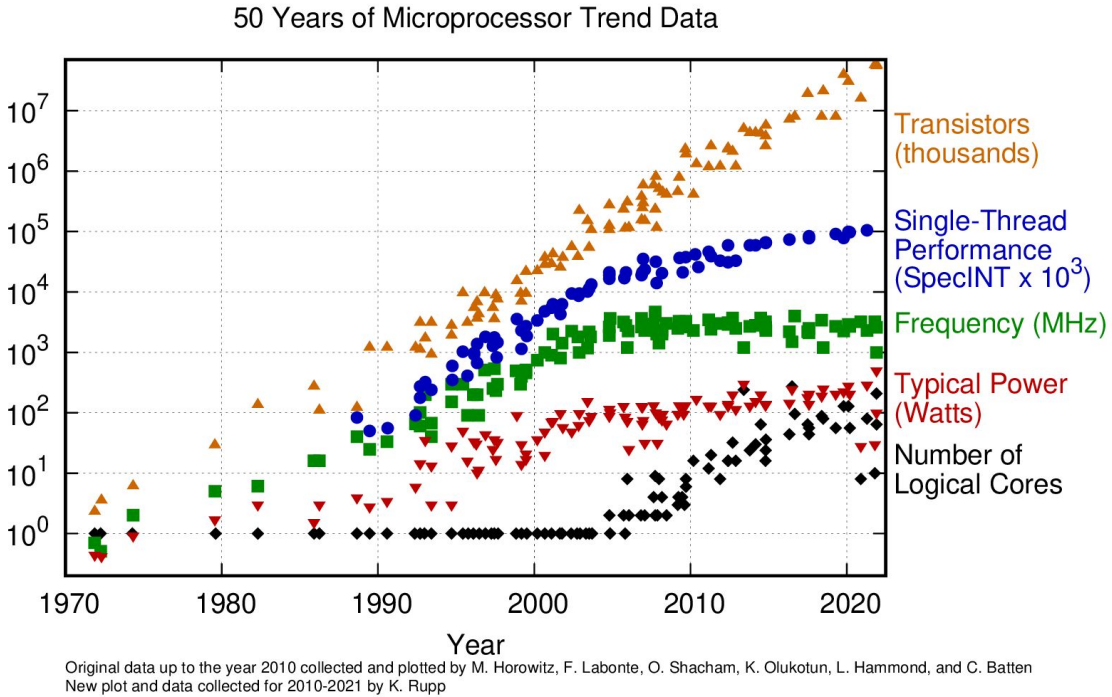
# Storage Speed Trends



# Storage Speed Trends



# CPU Trends




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 2010-2021 by K. Rupp

# Crimson

**Goals:**

- Minimize CPU overhead per IO
- Exploit fast storage devices

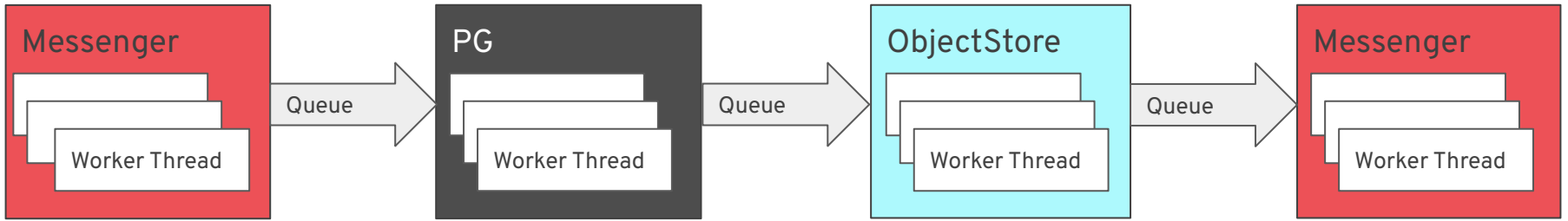
**How?**

- Classical ceph-osd replacement
- Efficient rewrite the IO path (Seastar Framework)
- Single thread per core

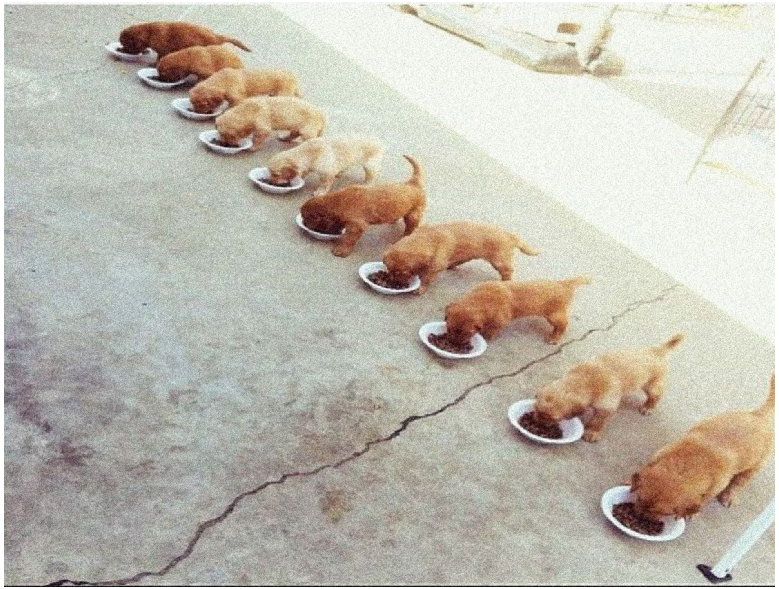
# OSD Architecture

- **Messenger:** Sends messages between daemons
- **PG:** Implements consistency protocols
- **ObjectStore** -Transactional interface to the local storage

# Classic-OSD Threading Architecture



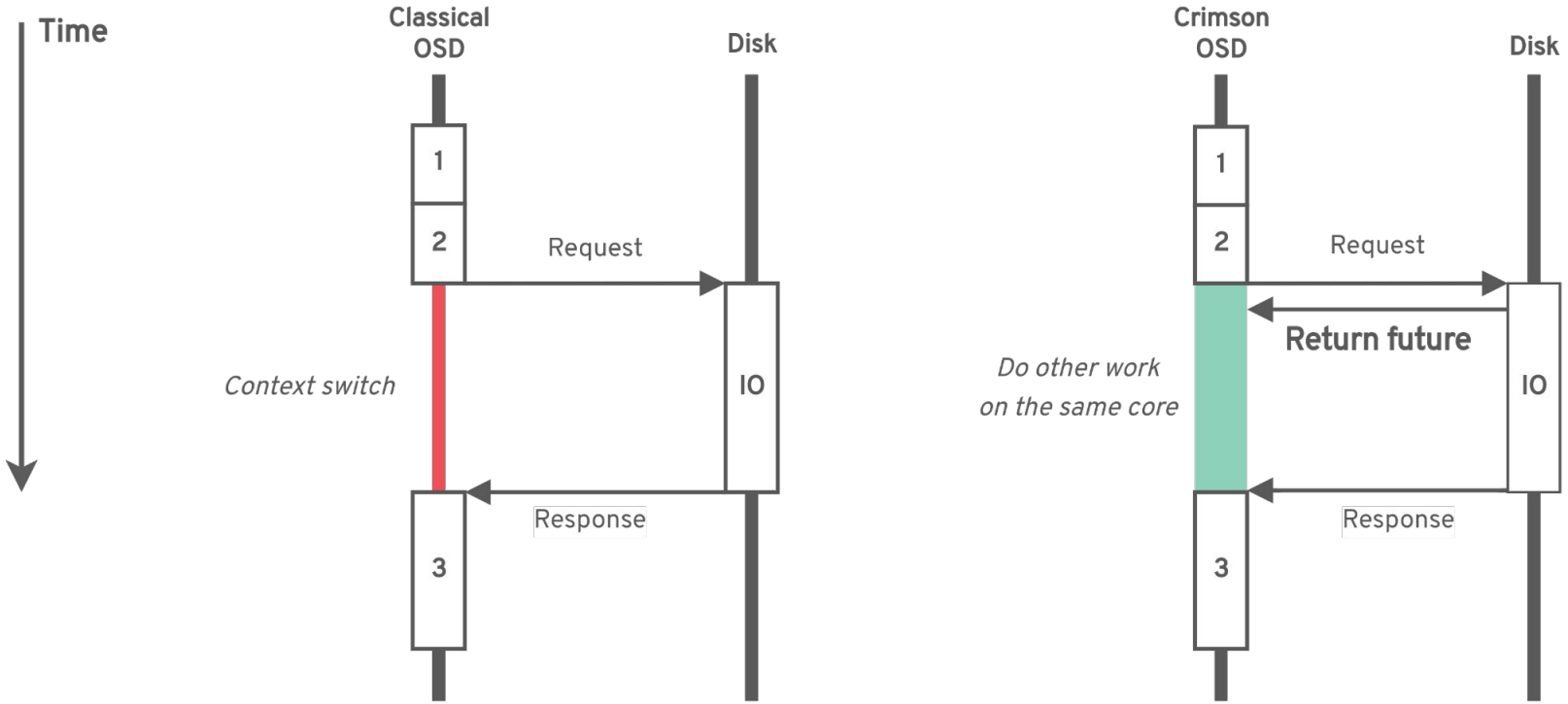
# MULTI-THREADING



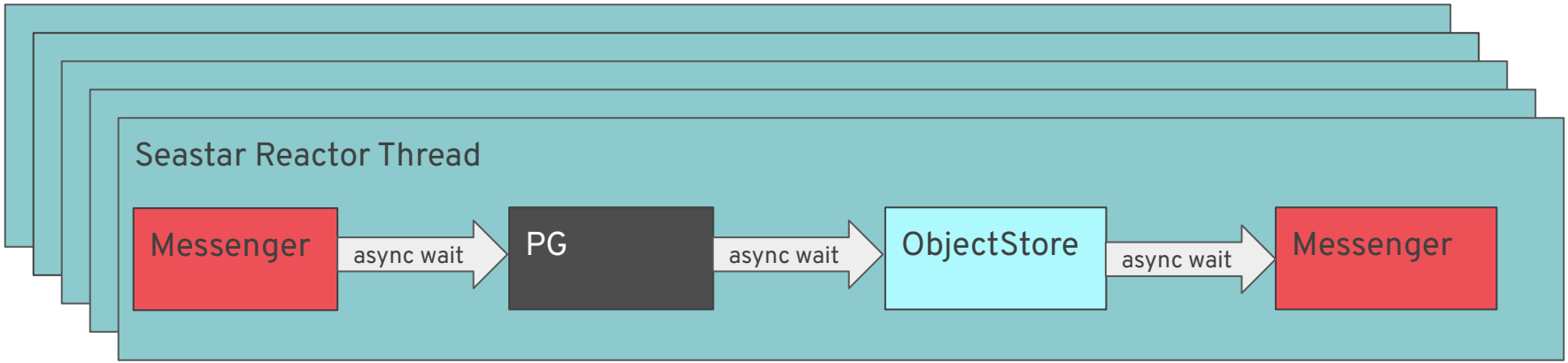
Theory

Practice

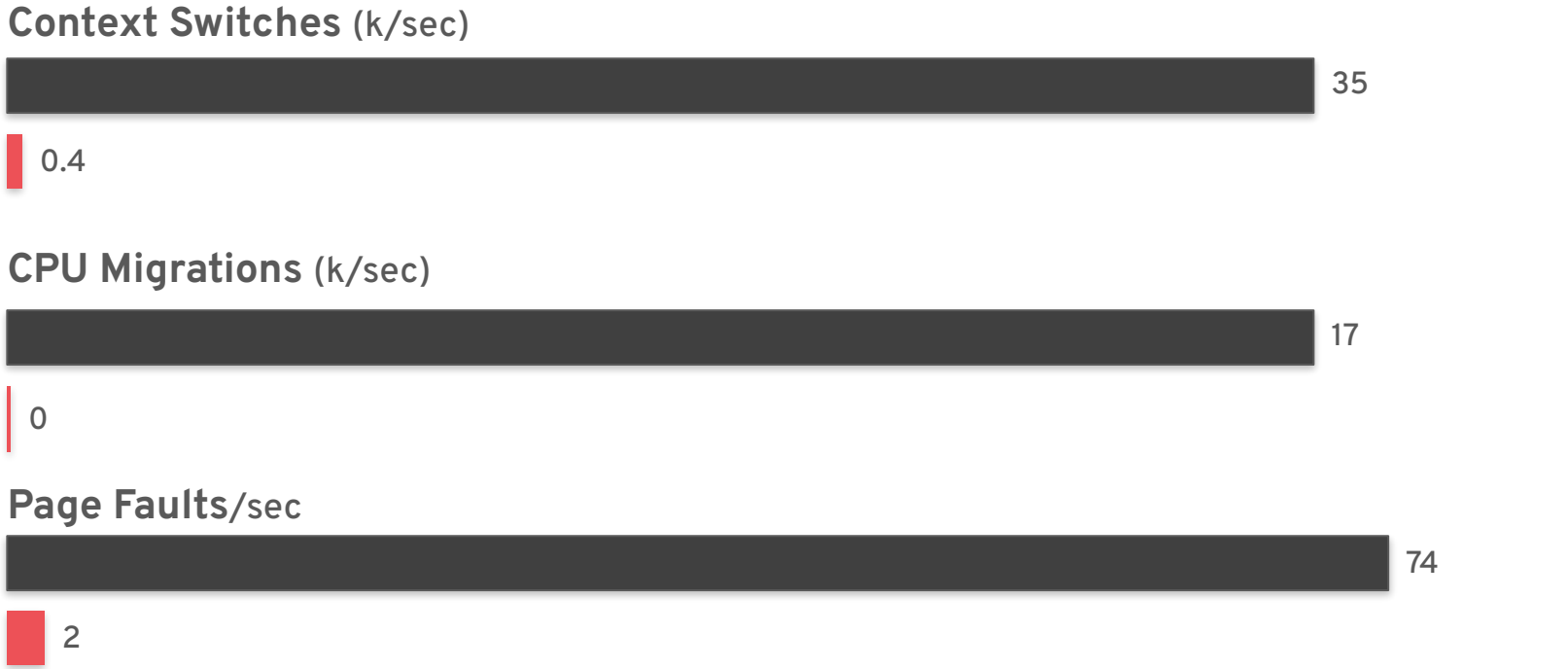
# OSD OPERATION COMPARISON



# Crimson-OSD Threading Architecture



# CPU Profiling



# Squid Crimson Status

- Multicore
- RBD workloads on Replicated pools
- Bluestore, Seastore and Cyanstore backends
- Quick deployments (Cephadm, Rook or vstart)
- Initial Scrub
- Recovery and Backfill

# Test Coverage

- Stable suite
- PR Gating (Since Reef)
- Recovery Thrashing patterns
- Seastore

# Coroutines (C++20)

```
return pg->do_osd_ops(m, obc, op_info).safe_then_unpack_interruptible(
  [this, pg, &ihref](auto submitted, auto all_completed) mutable {
    return submitted.then_interruptible([this, pg, &ihref] {
      return ihref.enter_stage<interruptor>(pp(*pg).wait_repop, *this);
    }).then_interruptible(
      [this, pg, all_completed=std::move(all_completed), &ihref]() mutable {
        return all_completed.safe_then_interruptible(
          [this, pg, &ihref](MURef<MOSDOpReply> reply) {
            return ihref.enter_stage<interruptor>(pp(*pg).send_reply, *this
            ).then_interruptible(
              [this, reply=std::move(reply)]() mutable {
                logger().debug("{}: sending response", *this);
                return conn->send(std::move(reply));
              });
          }, crimson::ct_error::eagain::handle([this, pg, &ihref]() mutable {
            return process_op(ihref, pg);
          }));
      });
  }, crimson::ct_error::eagain::handle([this, pg, &ihref]() mutable {
    return process_op(ihref, pg);
  }));
```

# Coroutines (C++20)

```
auto [submitted, all_completed] = co_await pg->do_osd_ops(
  m, r_conn, obc, op_info, snapc
);
co_await std::move(submitted);

co_await ihref.enter_stage<interruptor>(client_pp(*pg).wait_repop, *this);

auto reply = co_await std::move(all_completed);

co_await ihref.enter_stage<interruptor>(client_pp(*pg).send_reply, *this);
DEBUGDPP("{}.{}: sending response",
         *pg, *this, this_instance_id);

co_await interruptor::make_interruptible(
  get_foreign_connection().send_with_throttling(std::move(reply))
);
```

# Crimson - Next Steps

- Expand test coverage
- Erasure Coding
- PG Splitting/Merging
- Performance

# Experimental Usage

Required flags:

```
# ceph config set global 'enable_experimental_unrecoverable_data_corrupting_features' crimson
# ceph osd set-allow-crimson --yes-i-really-mean-it
# ceph config set mon osd_pool_default_crimson true
```

Upstream release support:

- Squid (19)
- Main

# BlueStore

- Default ObjectStore
- Suited for the Classical OSD architecture
- CPU heavy RocksDB dependence
- BlueStore is still being improved

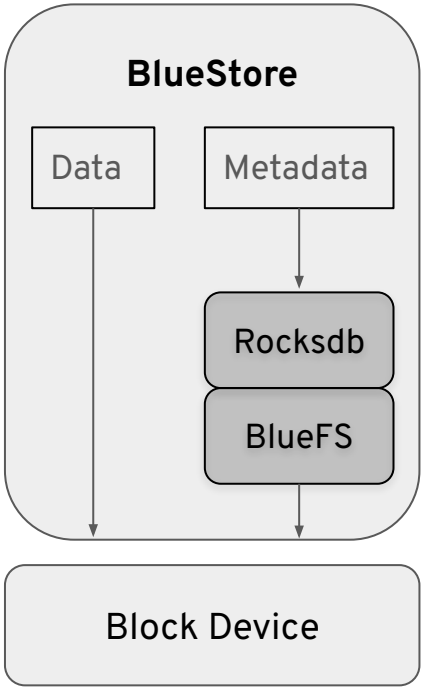
# Crimson with BlueStore

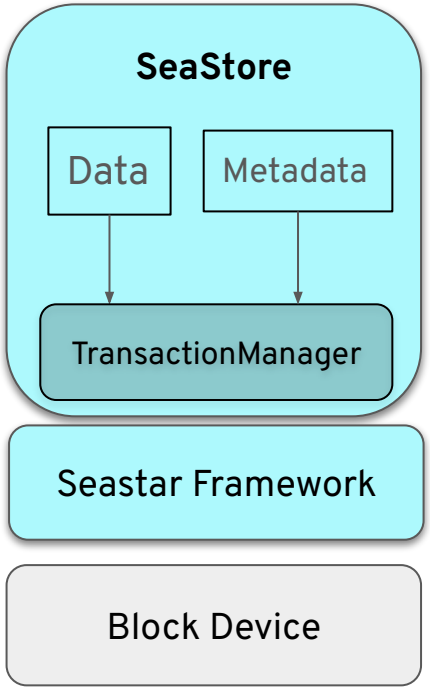
- Crimson is compatible with BlueStore
- Intermediate backend
- Throughput bottleneck

# SeaStore

- Crimson architecture native
- Designed to scale better than BlueStore/RocksDB
- Supports emerging storage technologies
- Metadata structures are B-tree based

# SeaStore


BlueStore
Data
Metadata
Rocksdb
BlueFS
Block Device


SeaStore
Data
Metadata
TransactionManager
Seastar Framework
Block Device

# SeaStore - Current Status

- Development led by Yingxin Cheng (Intel)
- Stability
- Tiering
- Multicore

# SeaStore - Next Steps

- Stability and CI
- Performance and Profiling
- Dynamic sharding
- More efficient device tiering

# Roadmap


- RBD with Replication
- Automated Test Suite
- Snapshots

- Tech Preview
- Stability
- Recovery/Backfill
- Scrub

- Full Scrub support
- EC support (for object)
- PG Splits and Merges
- **PERFORMANCE**

# Join US

- **Tracker**
  - *tracker.ceph.com/projects/crimson/issues*
- **Docs**
  - docs.ceph.com/en/latest/dev/crimson
- **Slack**
  - ceph-storage.slack.com (#crimson)
- **Weekly Meeting**
  - Ceph Community Calendar