



**CS 5412/LECTURE 19:
MAKING THE CLOUD FRIENDLIER FOR
OBJECT-ORIENTED COMPUTING**

**Ken Birman
Spring, 2022**

TODAY'S MAIN TOPIC

The cloud was built mostly from Linux systems, yet Linux was created to support interactive computing applications in offices and hospitals, databases (big files, but record-oriented), text editing.

As a result, *object orientation* has turned out to be an awkward match!

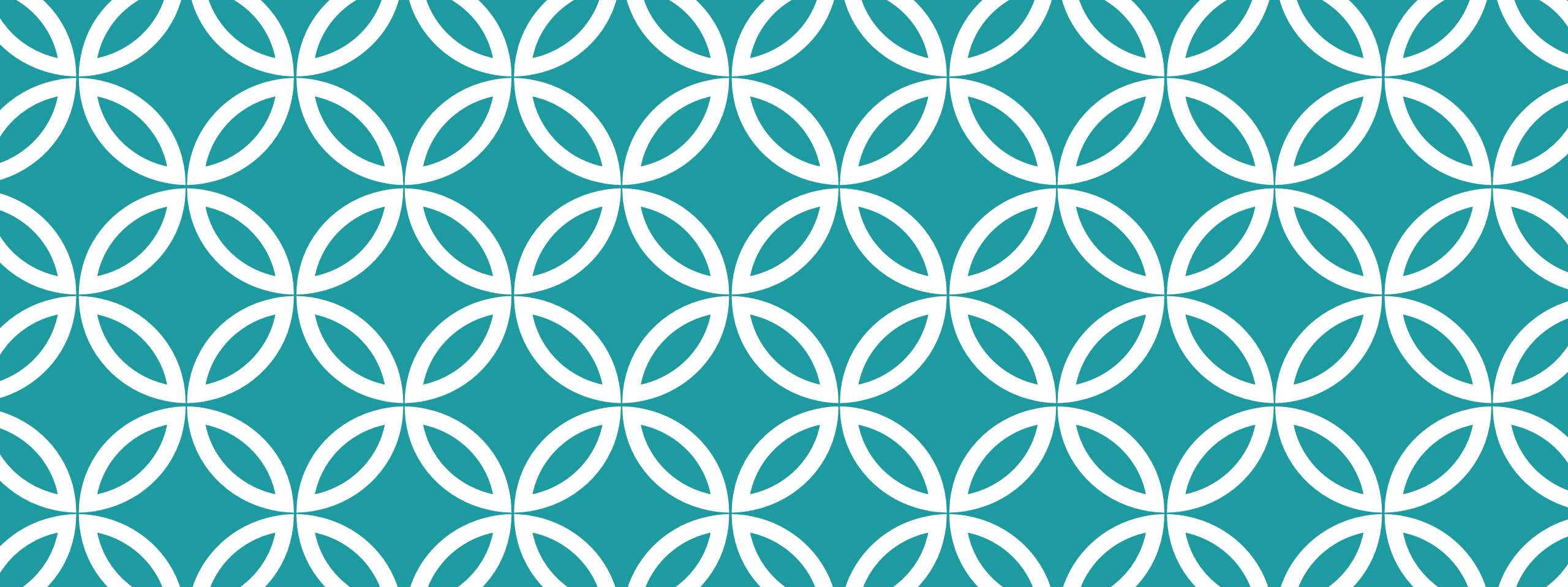
In Lecture 18 we talked about LINQ and how it extends the concept of a collection of objects to support SQL inside languages like C++ and Python. Today we will look at how big cloud systems deal with OO costs.

TODAY'S TWO SUB-TOPICS

First half of the lecture: The Ceph object oriented file system. Looks like a normal POSIX file system, but optimized for object-oriented uses.



Second half: Costs of the object-oriented model in a real air traffic control system (Ken worked on it... another bad-news/good-news stories...)



FIRST TOPIC

Ceph File System

MODERN SYSTEMS ARE OBJECT ORIENTED

Coded in languages like Java or C++

Managing huge amounts of data by creating immense data structures

Modular: there may be multiple microservices that talk to one-another through the key-value store, the file system, message buses and message queues. These should be optimized for the use case.

... ISSUES

There may be a LOT of files. Linux file systems aren't so great for this (like a directory with a billion objects in it).

Linux/POSIX lack suitable locking/synchronization API (POSIX had a solution but it was never adopted broadly)

Objects tend to be very small or very large. The Linux file system is optimized for Linux file size distributions and lifetime distributions.



CEPH PROJECT

Created by Sage Wehl, a PhD student at U.C. Santa Cruz

Later became a company and then was acquired into Red Hat Linux

Now the “InkStack” portion of Linux offers Ceph plus various tools to leverage it, and Ceph is starting to replace HDFS worldwide.

Ceph is similar in some ways to any standard cloud file system, but was created separately. Many big data systems are migrating to Ceph.

THREE PERSPECTIVES

First is the standard POSIX file system API. You can use Ceph in any situation where you might use GFS, HDFS, NFS, etc.

Second is the Ceph MetaData layer. This is a subsystem with its own API that manages objects... but doesn't store data

Third is the RADOS object storage layer. It holds the data but doesn't know about the folders (directories) in which data is organized.

OBJECT FORMATS ARE “INVISIBLE” TO CEPH

The actual format of the objects stored in the system are defined and “owned” by the applications using Ceph, not by Ceph itself.

Ceph tries to be extremely efficient, but it still is seeing serialized data (byte vectors), not objects with fields and methods it could invoke.

CEPH: A SCALABLE, HIGH-PERFORMANCE DISTRIBUTED FILE SYSTEM

Original slide set from OSDI 2006

Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrel D. E. Long

CONTENTS

Goals

System Overview

Client Operation

Dynamically Distributed Metadata

Distributed Object Storage

Performance

GOALS

Scalability

- Storage capacity, throughput, client performance. Emphasis on HPC.

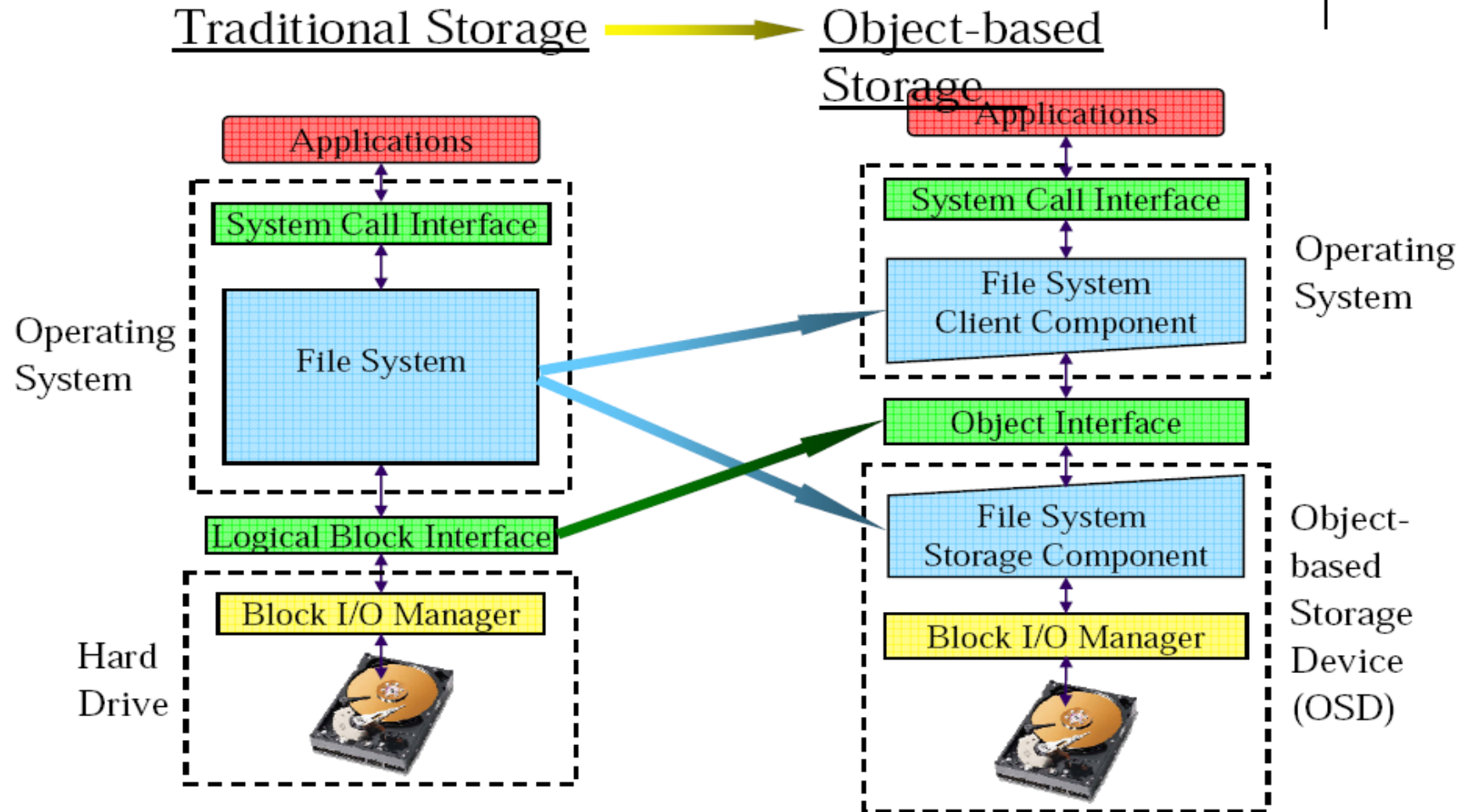
Reliability

- “...failures are the norm rather than the exception...”

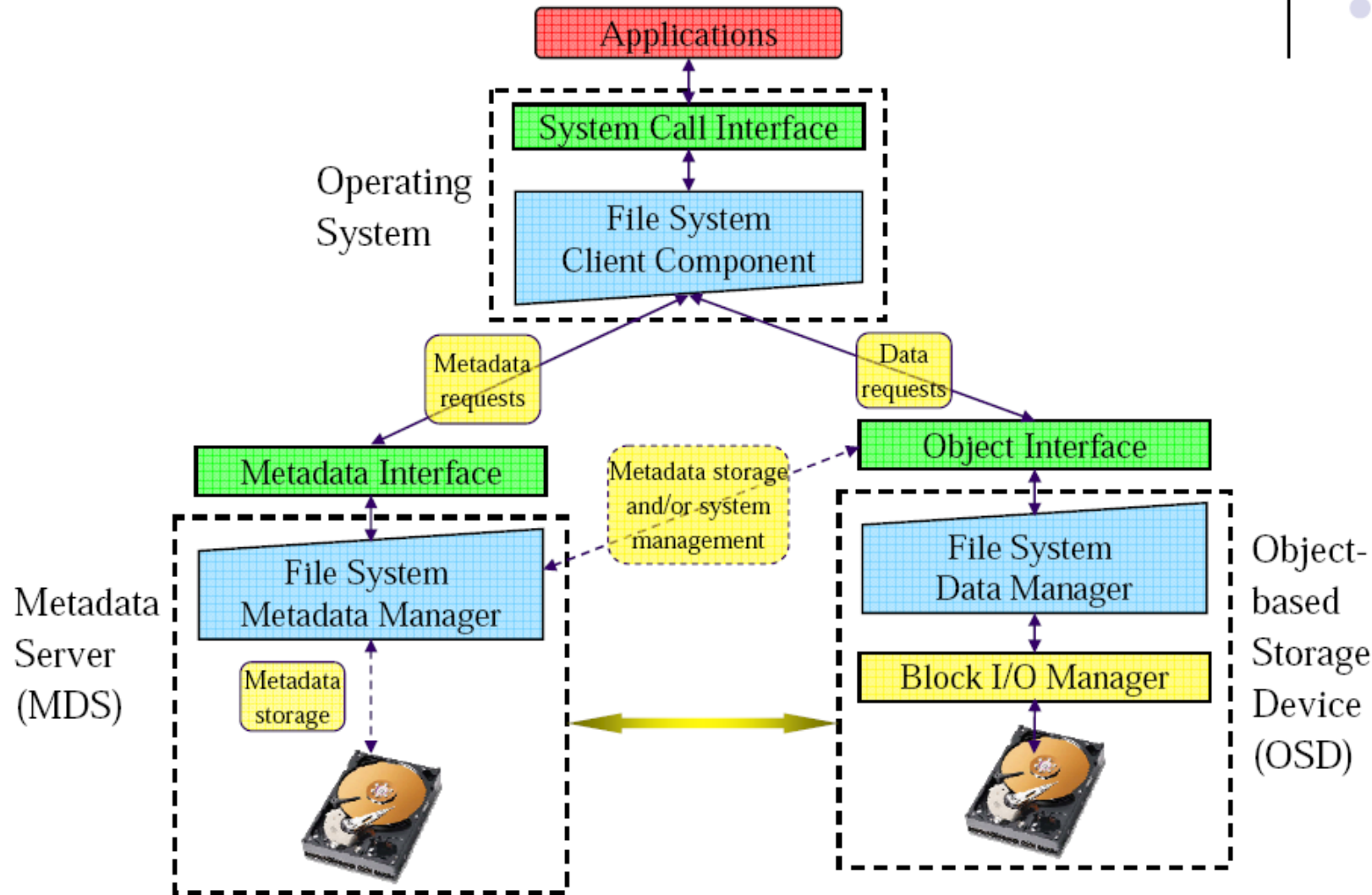
Performance

- Dynamic workloads

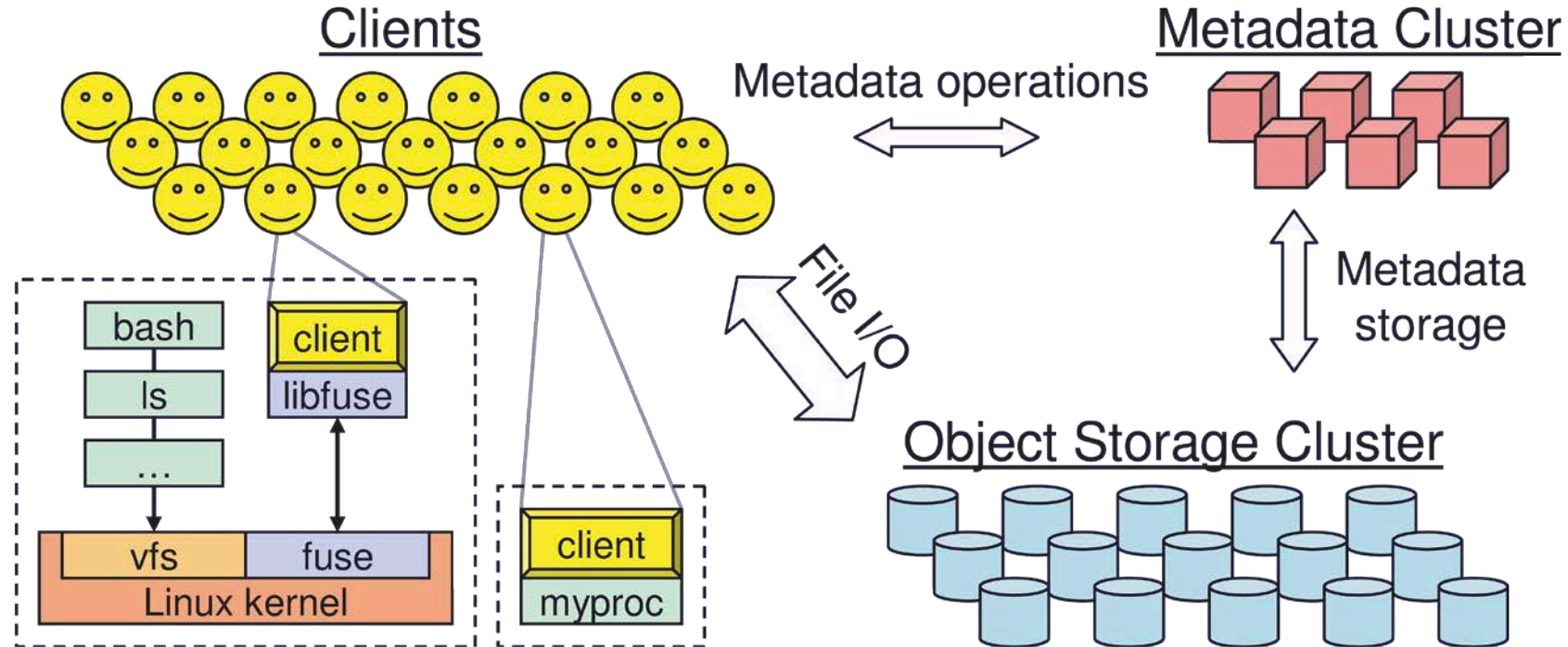
First Key Idea: Object-based Storage



Second Key Idea: Decoupled Data and Metadata



SYSTEM OVERVIEW



KEY FEATURES

Decoupled data and metadata

- CRUSH
 - Files striped onto predictably named objects
 - CRUSH maps objects to storage devices

Dynamic Distributed Metadata Management

- Dynamic subtree partitioning
 - Distributes metadata amongst MDSs

Object-based storage

- OSDs handle migration, replication, failure detection and recovery

CLIENT OPERATION

Ceph interface

- Nearly POSIX
- Decoupled data and metadata operation

User space implementation

- FUSE or directly linked

FUSE is a software allowing to implement a file system in a user space

CLIENT ACCESS EXAMPLE

Client sends open request to MDS

MDS returns capability, file inode, file size and stripe information

Client read/write directly from/to OSDs

MDS manages the capability

Client sends close request, relinquishes capability, provides details to MDS

SYNCHRONIZATION

Adheres to POSIX

Includes HPC oriented extensions

- Consistency / correctness by default
- Optionally relax constraints via extensions
- Extensions for both data and metadata

Synchronous I/O used with multiple writers or mix of readers and writers

DISTRIBUTED METADATA

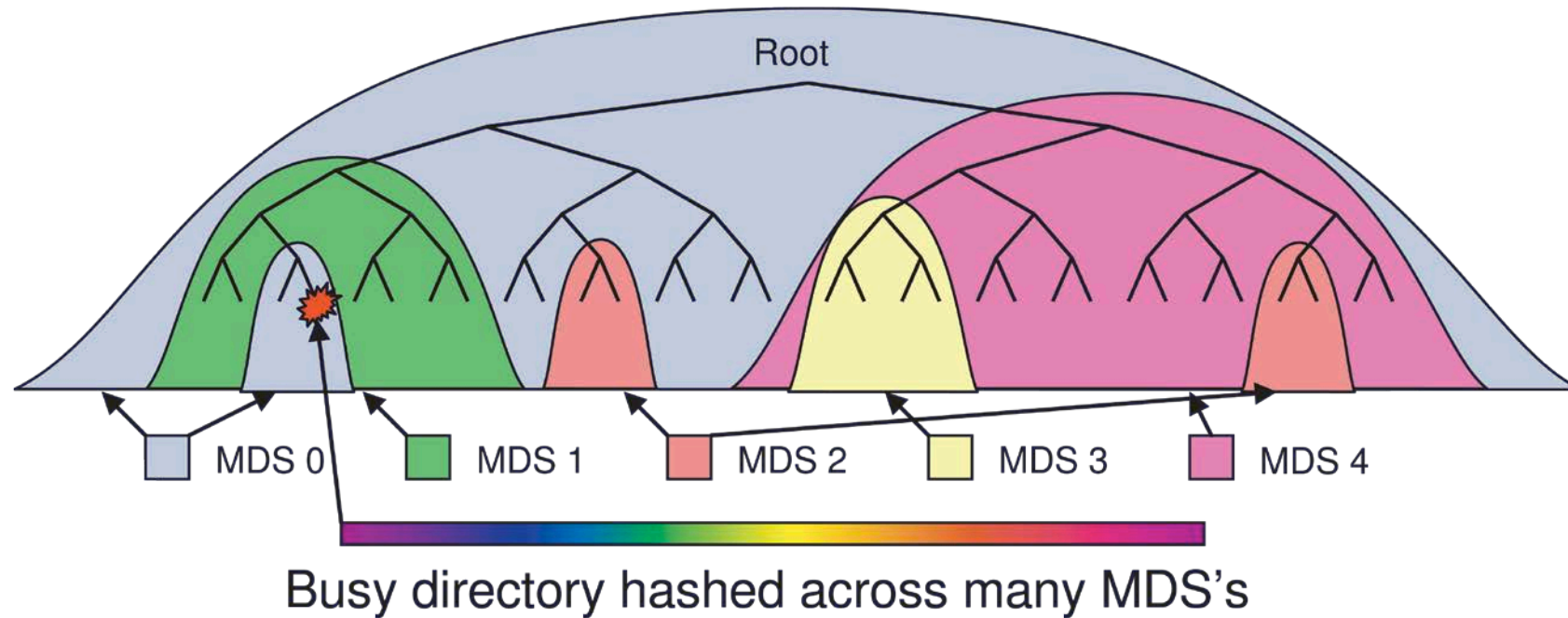
“Metadata operations often make up as much as half of file system workloads...”

MDSs use journaling

- Repetitive metadata updates handled in memory
- Optimizes on-disk layout for read access

Adaptively distributes cached metadata across a set of nodes

DYNAMIC SUBTREE PARTITIONING



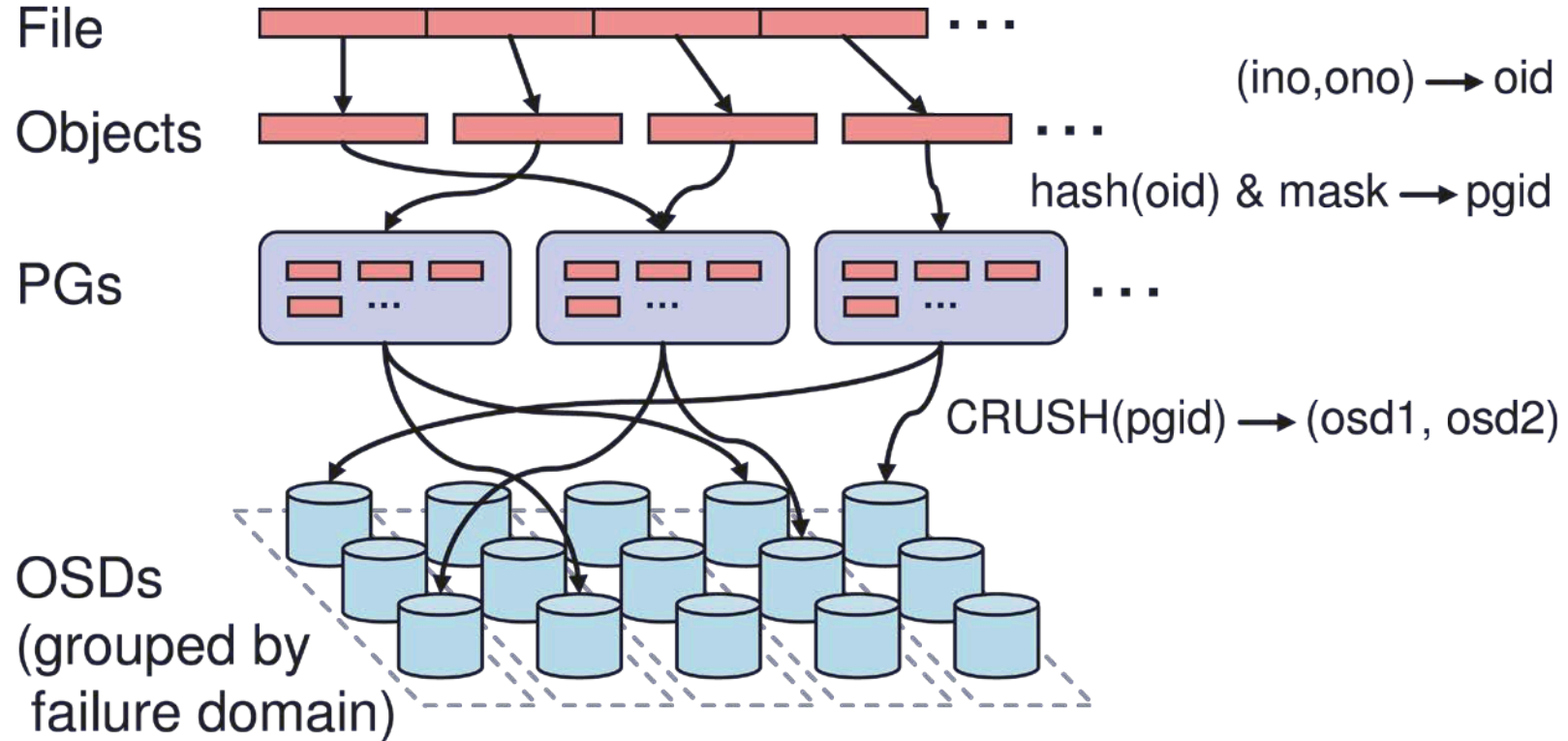
DISTRIBUTED OBJECT STORAGE

Large files are split into a set of objects

Objects are members of placement groups

Placement groups are distributed across OSDs.

DISTRIBUTED OBJECT STORAGE



CRUSH: A SPECIALIZED KEY HASHING FUNCTION

CRUSH(x): (osdn1, osdn2, osdn3)

- Inputs
 - x is the placement group
 - Hierarchical cluster map
 - Placement rules
- Outputs a list of OSDs

Advantages

- Anyone can calculate object location
- Cluster map infrequently updated

DATA DISTRIBUTION

(not a part of the original PowerPoint presentation)

Files are striped into many objects

➤ (ino, ono) → an object id (oid)

Ceph maps objects into placement groups (PGs).

➤ $\text{hash}(\text{oid}) \ \& \ \text{mask}$ → a placement group id (pgid)

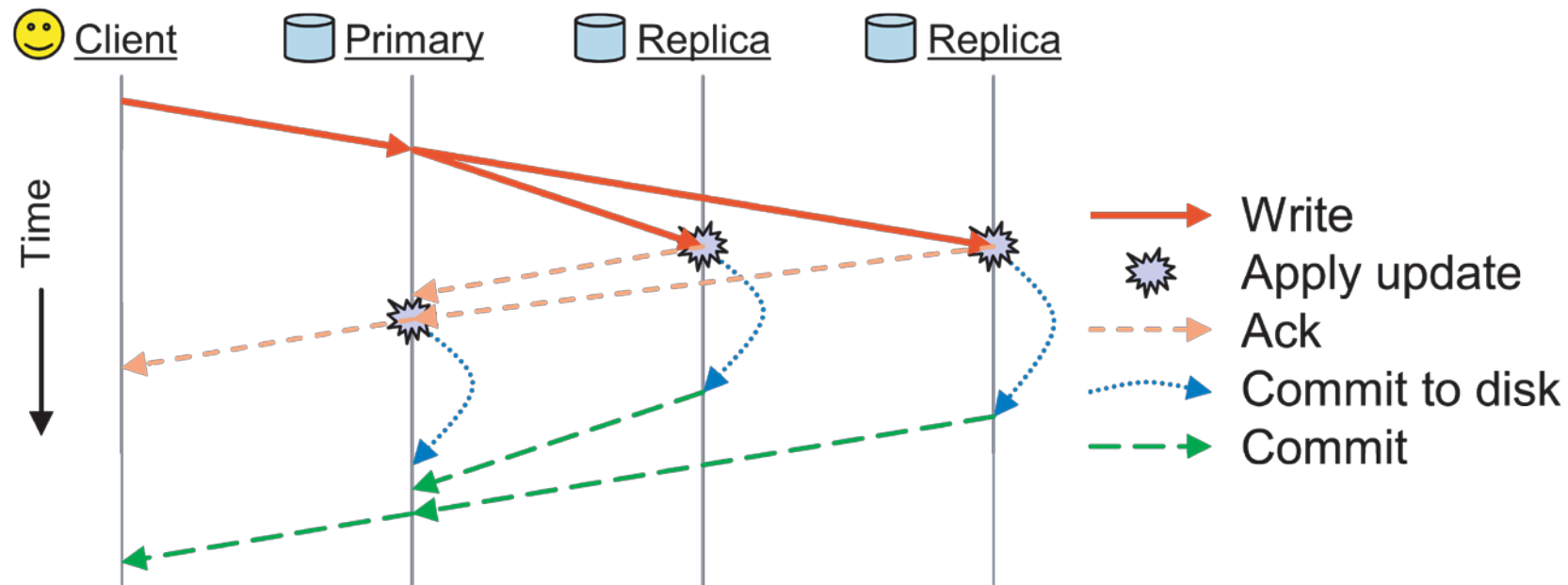
CRUSH assigns placement groups to OSDs, using what seems to be a “shard”

➤ $\text{CRUSH}(\text{pgid})$ → a replication group, (osd1, osd2)

REPLICATION: RELIABLE BUT NOT PAXOS

Objects are replicated on OSDs within same PG

- Client is oblivious to replication



FAILURE DETECTION AND RECOVERY

Down and Out

Monitors check for intermittent problems

New or recovered OSDs peer with other OSDs within PG

ACRONYMS USED IN PERFORMANCE SLIDES

CRUSH: Controlled Replication Under Scalable Hashing

EBOFS: Extent and B-tree based Object File System

HPC: High Performance Computing

MDS: MetaData server

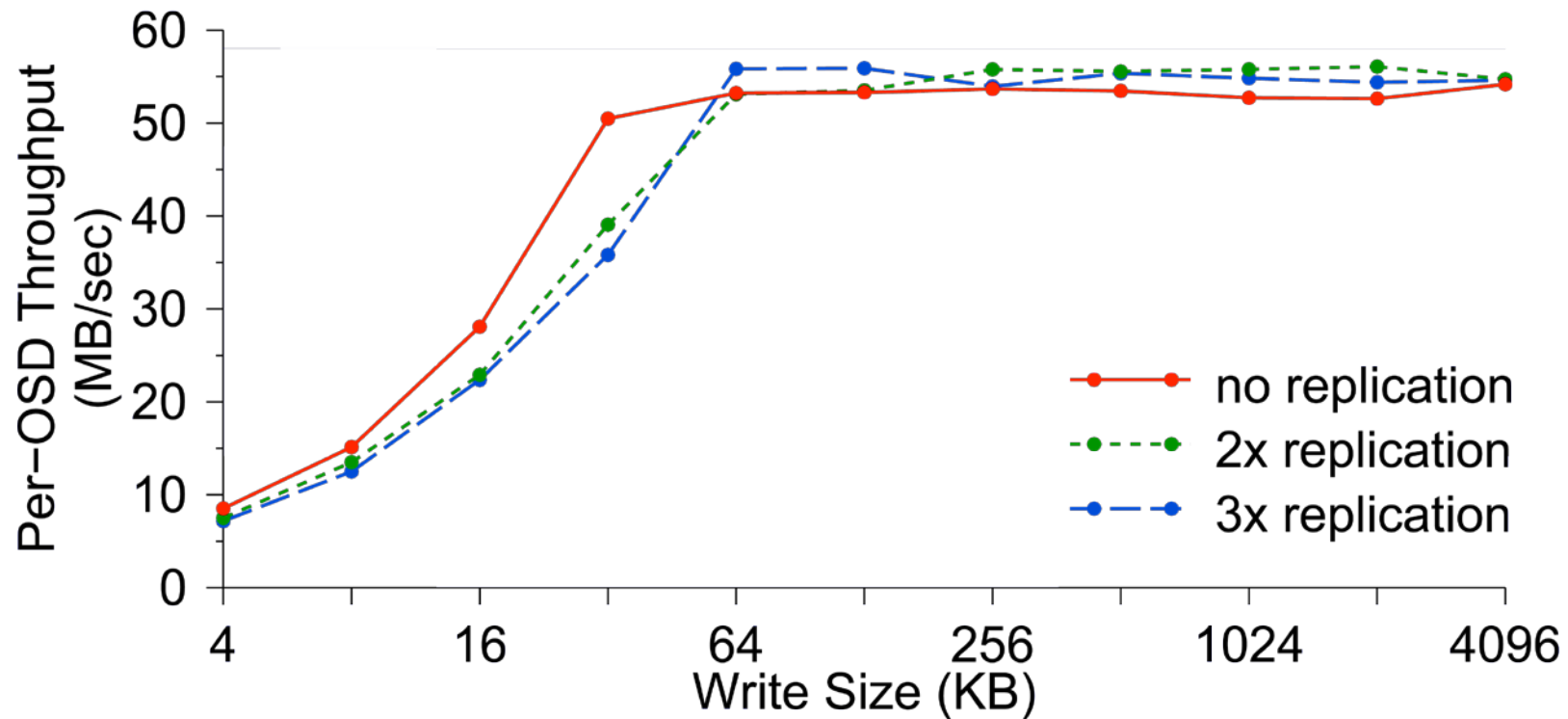
OSD: Object Storage Device

PG: Placement Group

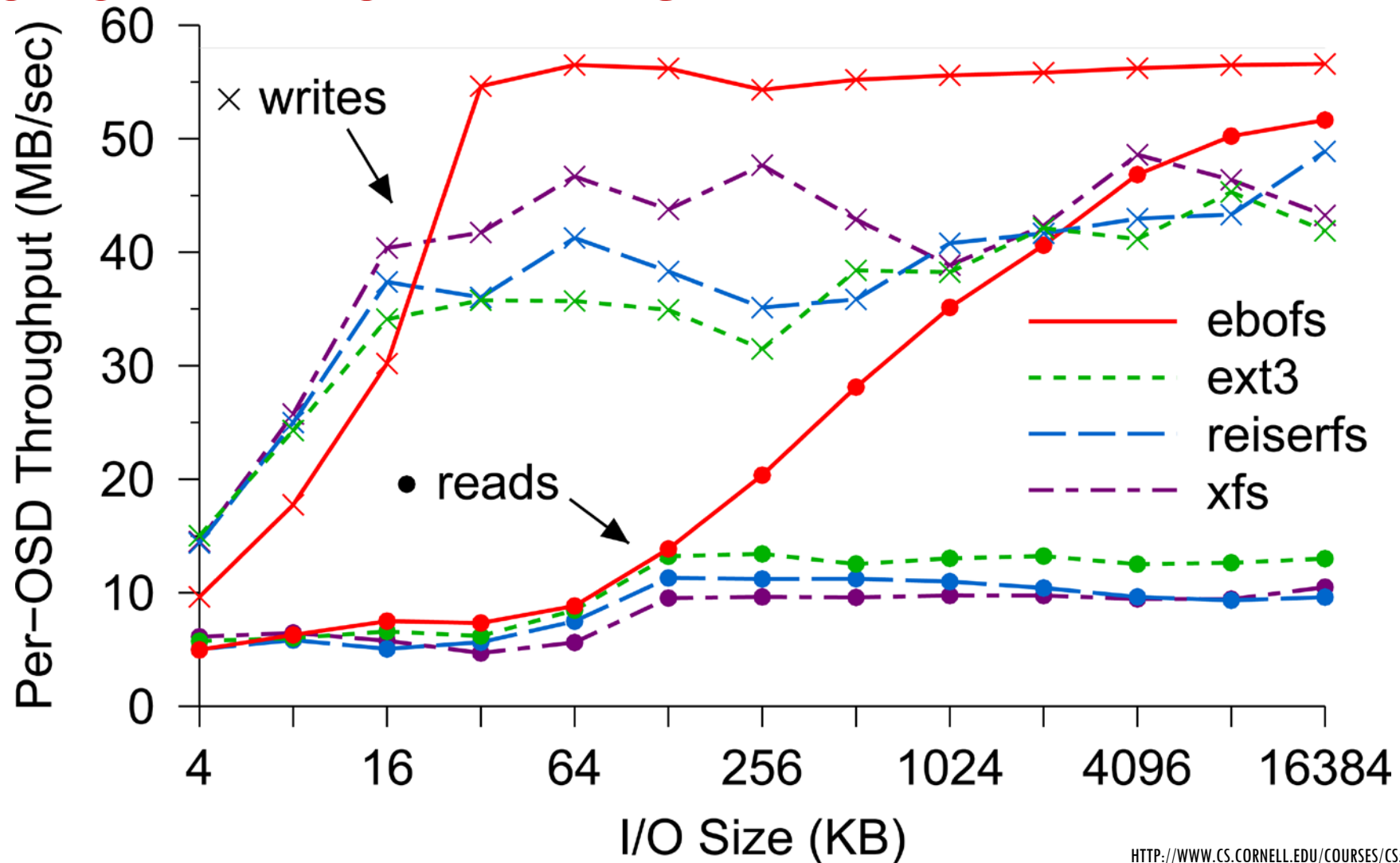
POSIX: Portable Operating System Interface for uniX

RADOS: Reliable Autonomic Distributed Object Store

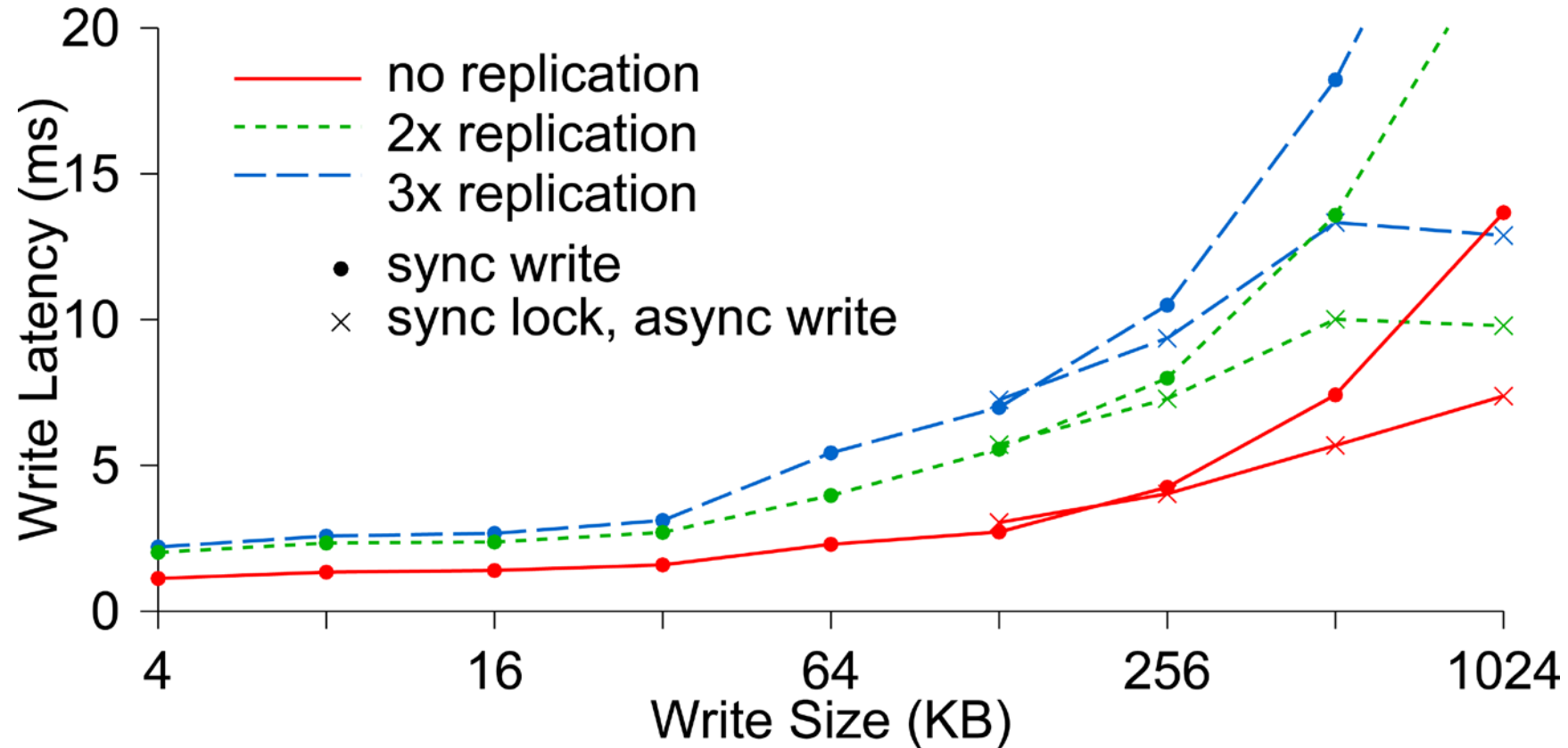
PER-OSD WRITE PERFORMANCE



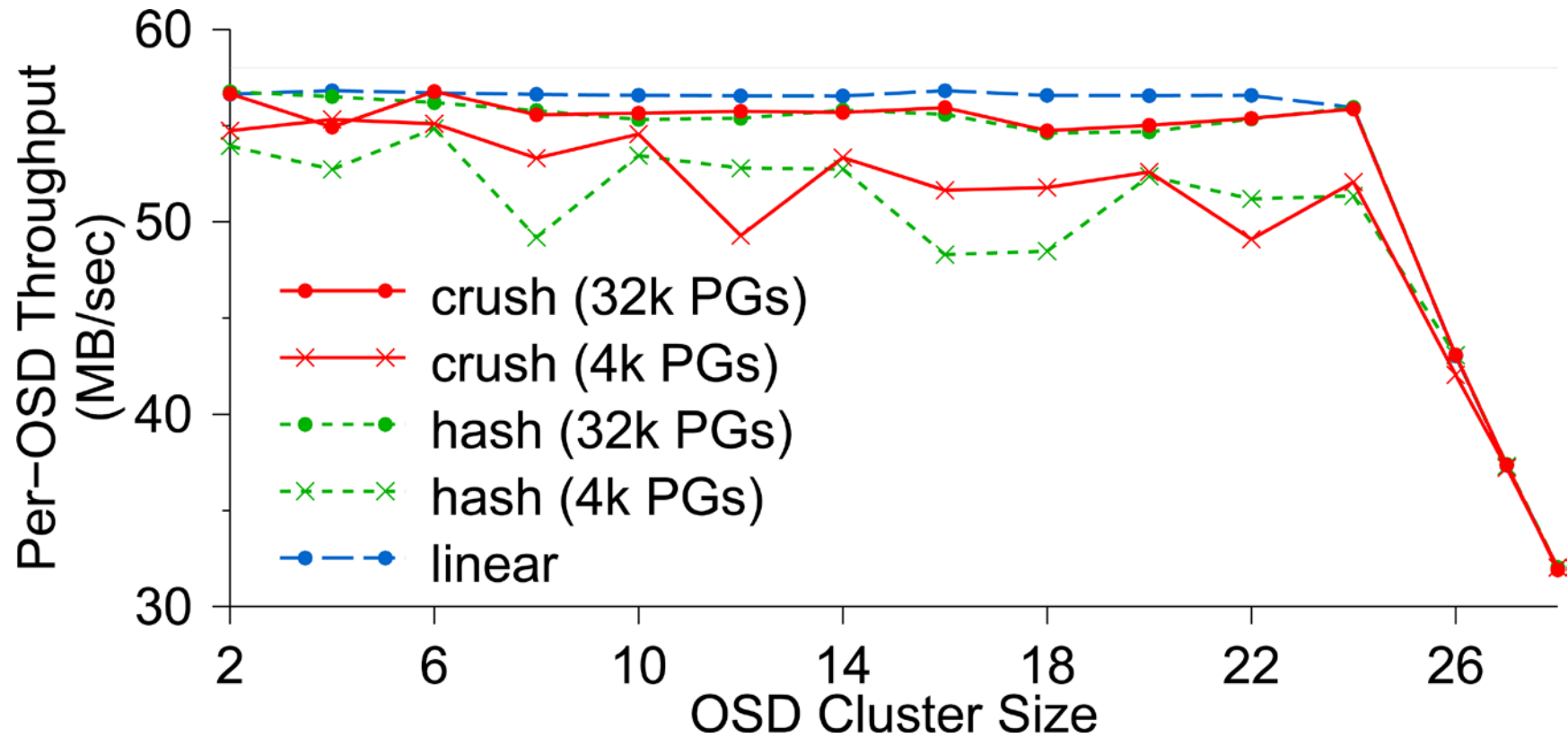
EBOFS PERFORMANCE



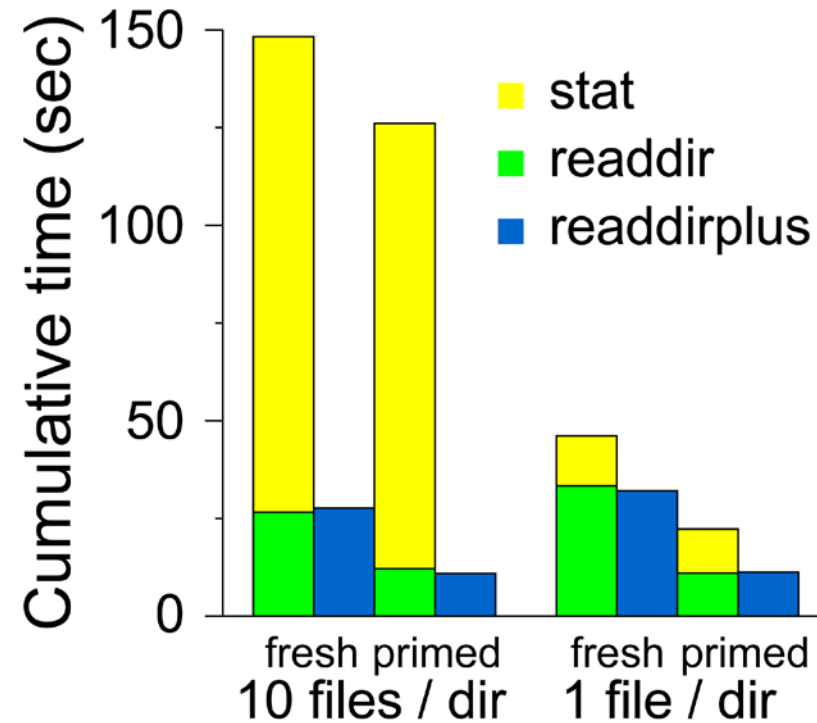
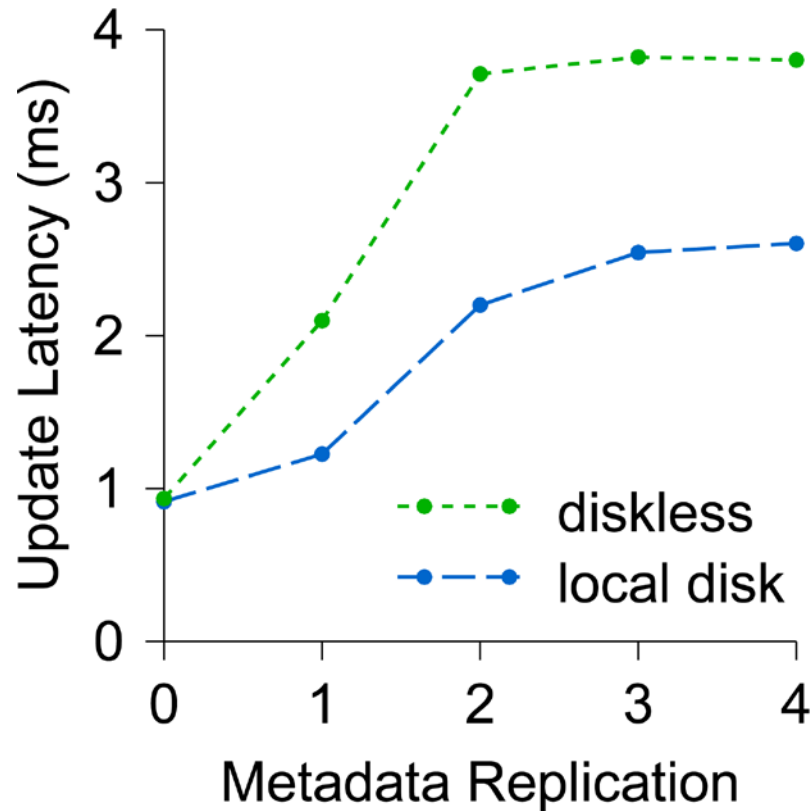
WRITE LATENCY



OSD WRITE PERFORMANCE

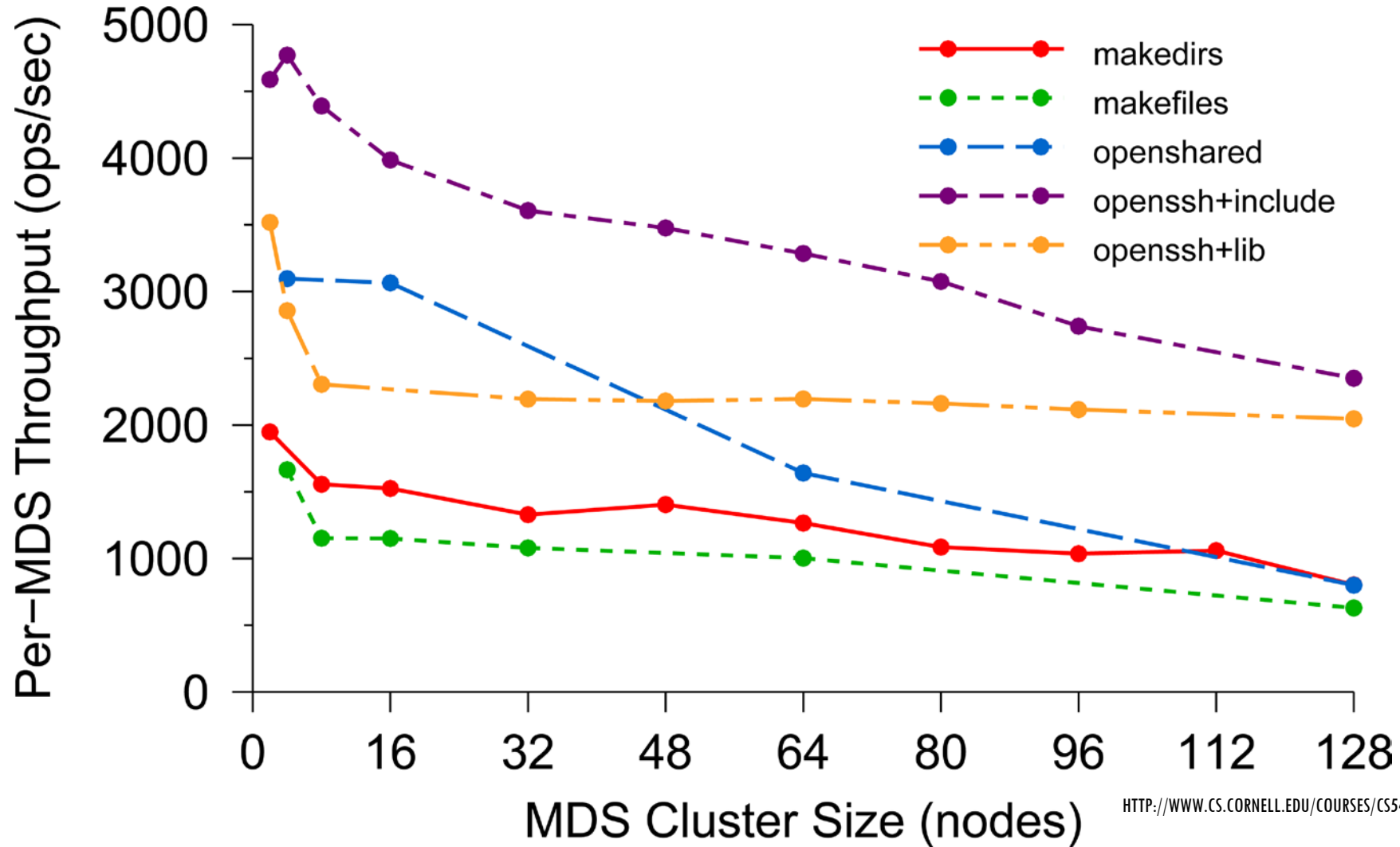


DISKLESS VS. LOCAL DISK

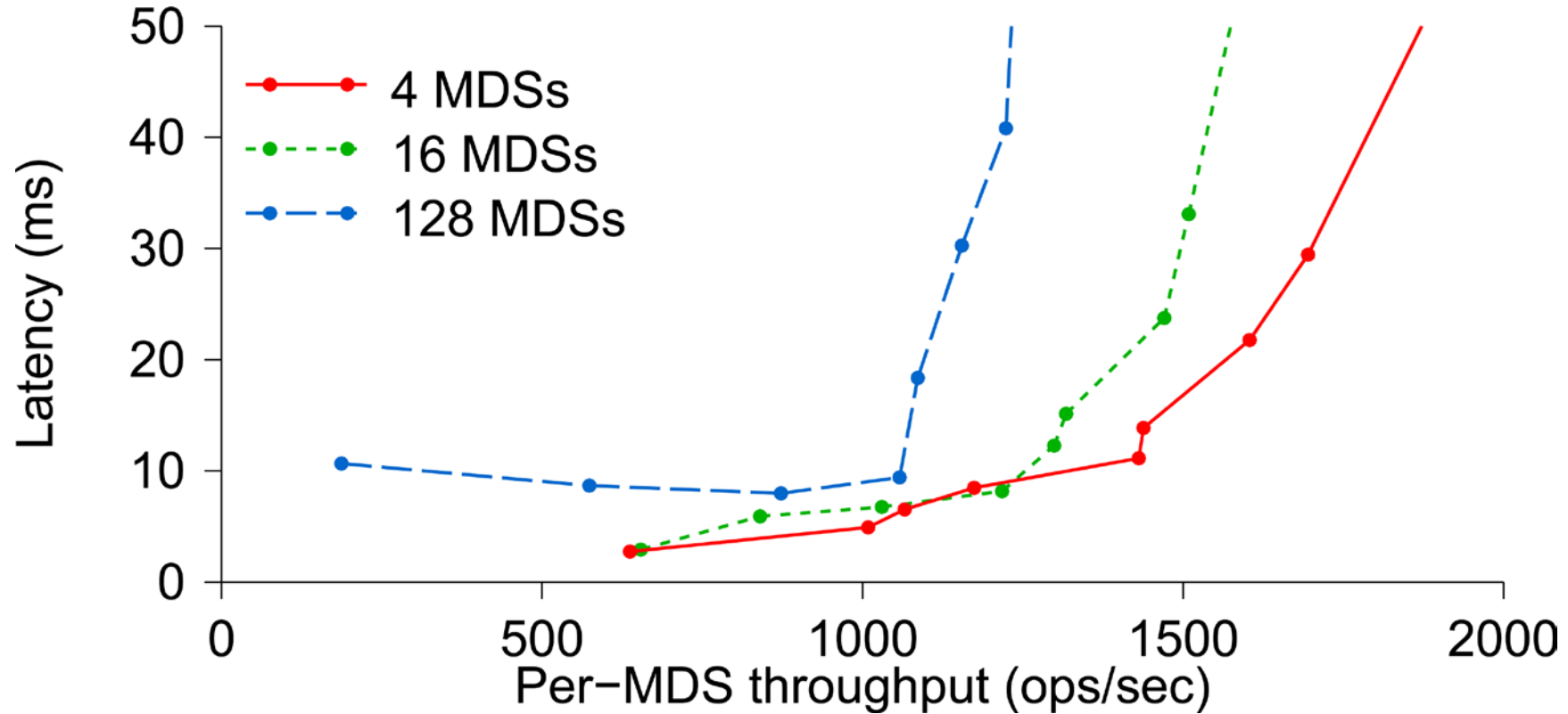


Compare latencies of (a) a MDS where all metadata are stored in a shared OSD cluster and (b) a MDS which has a local disk containing its journaling

PER-MDS THROUGHPUT



AVERAGE LATENCY

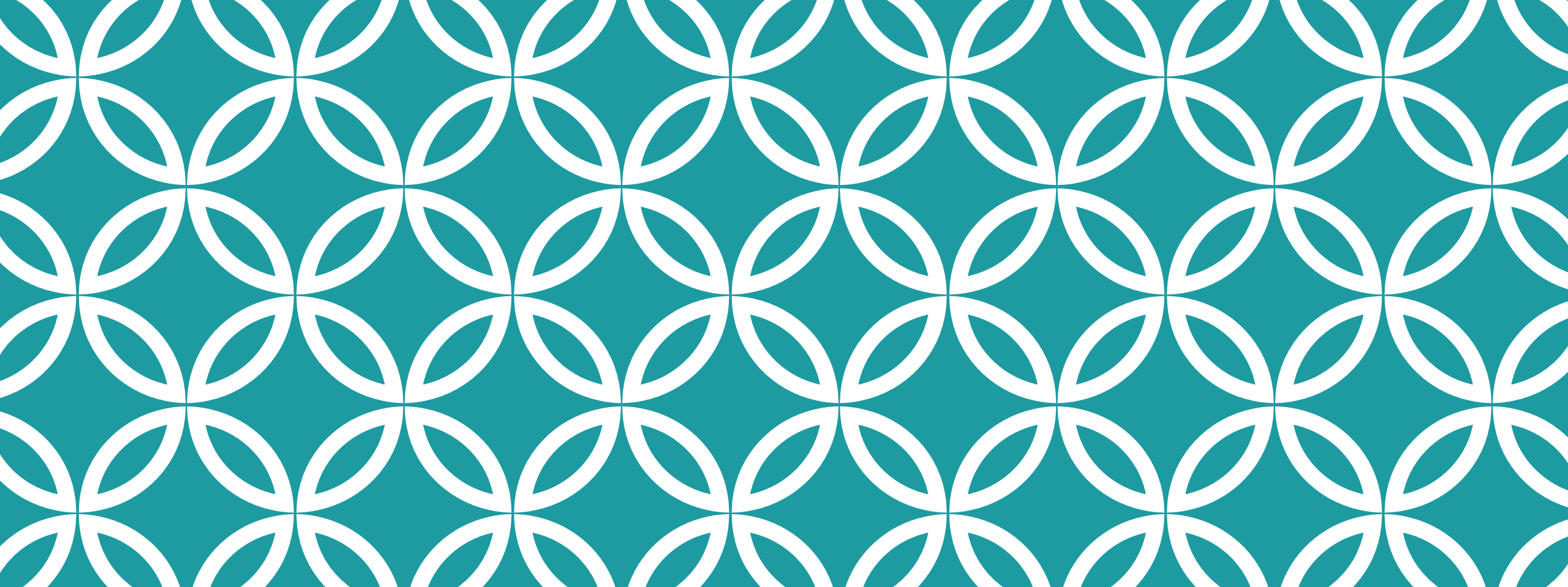


LESSONS LEARNED

If applications are object oriented, they will write huge numbers of variable-size records (some extremely large).

POSIX directories are awkward. A B+ tree index works much better.

Treat the records as byte arrays, track meta-data in one service and data in a second one. Both share the RADOS layer for actual data storage.



SECOND (DIFFERENT) TOPIC

Overheads of object orientation

LET'S DRILL DOWN ON THAT REMARK ABOUT DOUBLED OVERHEADS

We say that Ceph is more costly when used as a file system, and less so when used directly as a key-value storage layer via RADOS.

This illustrates a cost associated with abstraction. When we adopt the wrong API, the translation between that API and the true one can add overheads.

These costs matter – and we'll see that now in a second example.

CORBA AND OMG

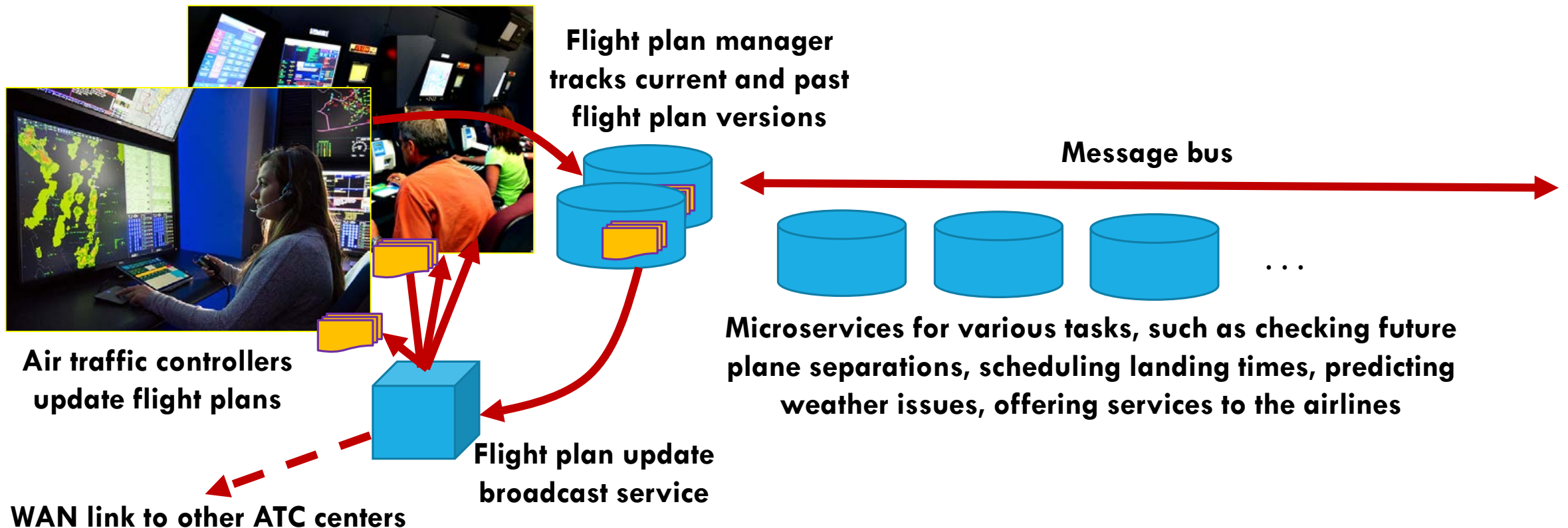
Ceph is really an outgrowth of a consortium called the “Object Management Group” or OMG.

They proposed a standard way to translate between internal representations of objects and byte array external ones. They call this the Common Object Request Broker Architecture or CORBA.

We can think of an application using Ceph as a kind of CORBA use case.

UNDERSTANDING COSTS FOR CORBA'S UNIVERSAL REPRESENTATIONS: ATC SYSTEM

A modern air traffic control system might have a structure like this:



UNDERSTANDING COSTS FOR CORBA'S UNIVERSAL REPRESENTATIONS: ATC SYSTEM

Notice first that this architecture is actually a lot like Ceph or HDFS:

- The meta-data server in Ceph and HDFS is “like” the database of flight plan versions
- The copies near the controllers are “like” the RADOS storage unit or the HDFS store.
- And the message bus is “like” a live notification service for watched files

UNDERSTANDING COSTS FOR CORBA'S UNIVERSAL REPRESENTATIONS: ATC SYSTEM

Also, think about objects in an ATC system:

- Flight plans: these are elaborate objects that might hold 10MB of data and could have a great many internal fields
- Many other kinds of objects are used too. Each microservice probably has a notifications channel of its own, and uses it to talk to individual controllers or sets of them about relevant issues
- ***“Attention: In 2h 31m, BA 123 will approach US 654 on approach to CDG. Plan corrective action to avoid a violation of flight separation rules.”***

UNDERSTANDING COSTS FOR CORBA'S UNIVERSAL REPRESENTATIONS: ATC SYSTEM

An ATC system has many components, far more than were shown.

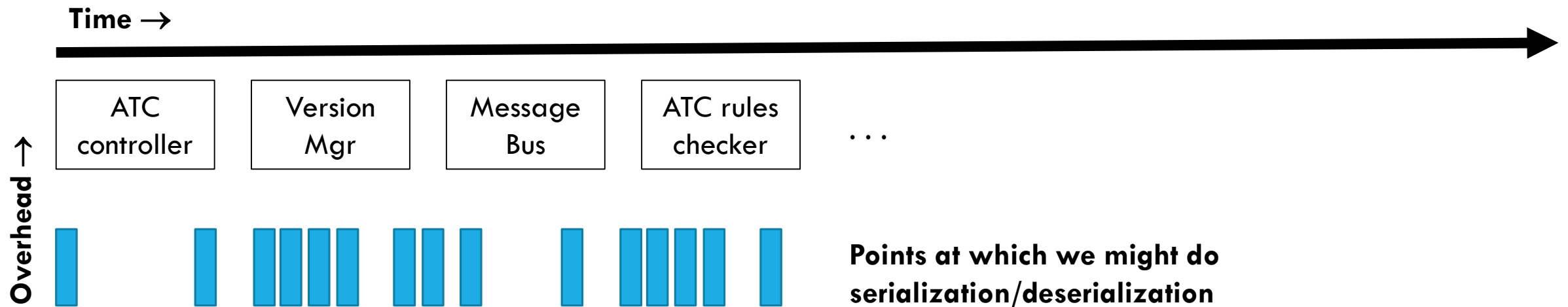
Often these are based on high-quality legacy versions and hence there can be many programming languages in simultaneous use.

- Often we will see C/C++, Java, C#, F#, O'Caml, etc.
- Some use of Python and Fortran and Ada.
- With CORBA, we can easily integrate many modules into a single system

BUT HOW OFTEN WILL WE (DE)SERIALIZE?

Each time an object is read or written (from disk or network)

Each time an object is passed from one module to another



UNIVERSAL REPRESENTATIONS ARE COSTLY!

It is very easy for a CORBA application to spend *all* its time on this one action.

Ceph designers were aware of that, and decided it should only be done under application control.

Thus Ceph is “object oriented” and yet reflects a choice not to have the whole system understand every kind of object

HOW DO ATC SYSTEMS AVOID THESE COSTS?

The trick is to use “lazy” record access.

The ATC record is the main object being shared. Suppose that we have two versions of an ATC object while in memory:

- Version A: The object is fully resident in memory and you can access all fields, edit it to create a new version, etc.
- Version B: All the same methods are offered, but the in-memory data is limited to a URL pointing to the record in the flight plan database

WHY TWO “IDENTICAL” OBJECT VARIANTS?

Notice how easy it is to switch from representation B to A (or back).

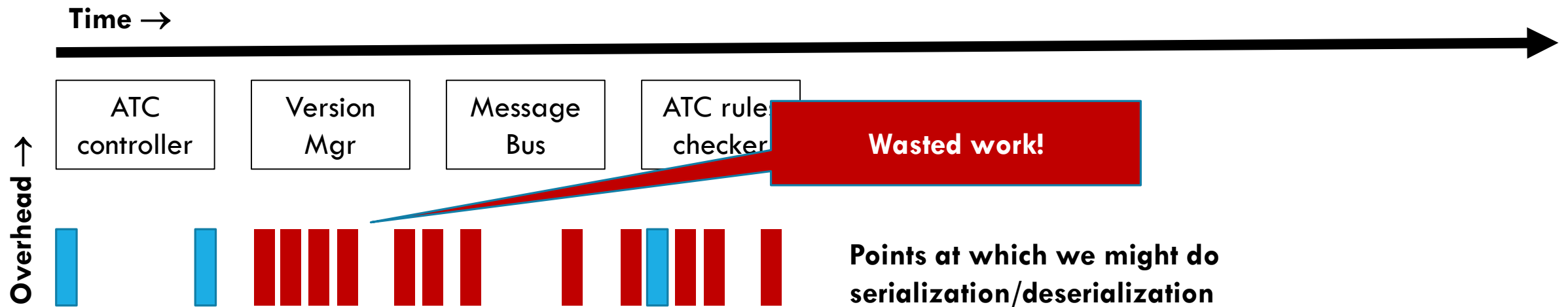
In an ATC system most components don't really look at the data fields and for this reason, most components would be happy with representation B. But a small object with just a URL in it is very cheap to serialize!

With “lazy deserialization”, we would convert from form B to form A only when an application tries to touch the data.

OLD SINGLE VERSION APPROACH

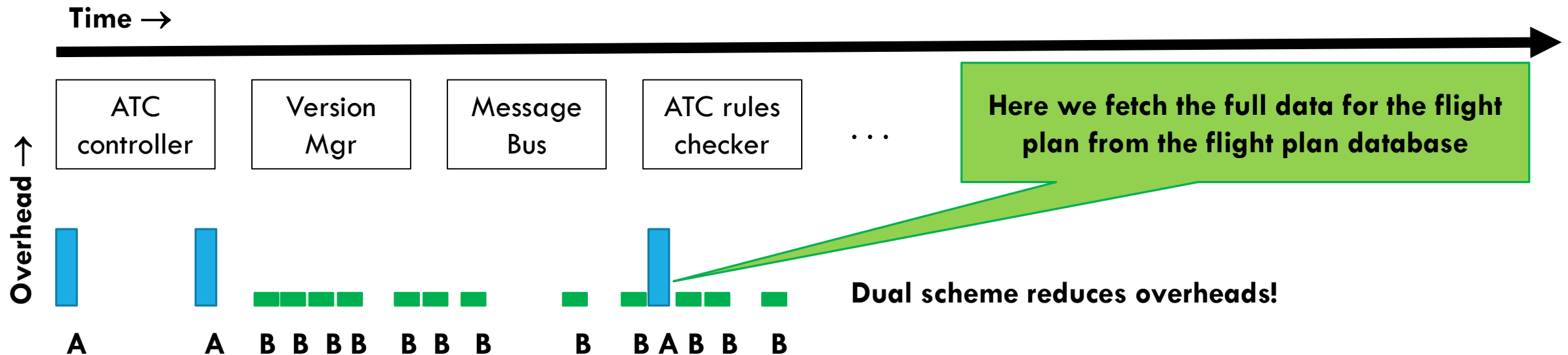
Each time an object is read or written (from disk or network)

Each time an object is passed from one module to another



DUAL VERSION APPROACH

We only do a costly action when the component will actually touch the inner data fields!



HOW SHOULD WE STORE THE FLIGHT PLAN RECORDS?

The need is for a very simple append-only log managed by the version manager.

It is easy to recognize this as a use case for state machine replication.

This situates the central safety question in one specific component, where we can formalize it and use mathematical tools to prove that each plan has just one sequence of versions, used consistently by all components.

HOW SHOULD WE IMPLEMENT THE FLIGHT PLAN MANAGER COMPONENT?

A (key-value) sharded service built on Derecho would be an ideal choice.

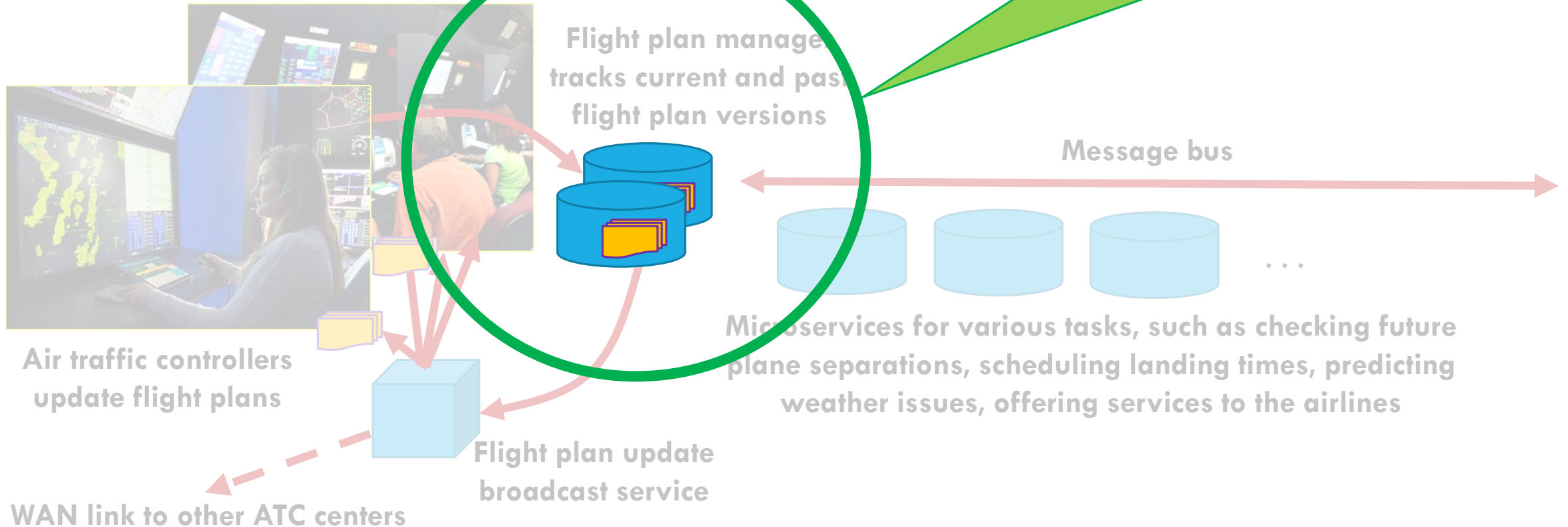
Derecho has been proved correct in several ways: by hand, but also using a machine-verified proof in the Ivy protocol verification tool.

It is also scalable and extremely fast: important because this role is central.

REVISITING THE STRUCTURE

A modern air traffic control system might have

If this one component is correct, the whole system can be proved safe!



SUMMARY – CEPH

Ceph is a file system that was created by taking the HDFS model, but then extending it to be better matched to properties of object-oriented code.

This is very popular, although it does bring overheads.

Ceph uses a simple but “weak” form of data replication. It doesn’t guarantee consistency.

SUMMARY – CORBA

Here we saw a different form of object-oriented overhead, arising in applications that adopt the standard CORBA approach to interoperability.

CORBA buys flexibility but brings steep costs.

Those costs can be managed by understanding the architecture and designing the application to avoid triggering costly serialization/deserialization.

BROADER INSIGHT LINKING THESE SUBTOPICS

Innovation brings challenges.

As the world has shifted towards object oriented languages like Java, C++, Python with its object features, etc., we are finding that they pay steep overheads if you adapt them naively to run on standard platforms.

But with modest effort, they can perform extremely well in the cloud.