Distributed File Systems with

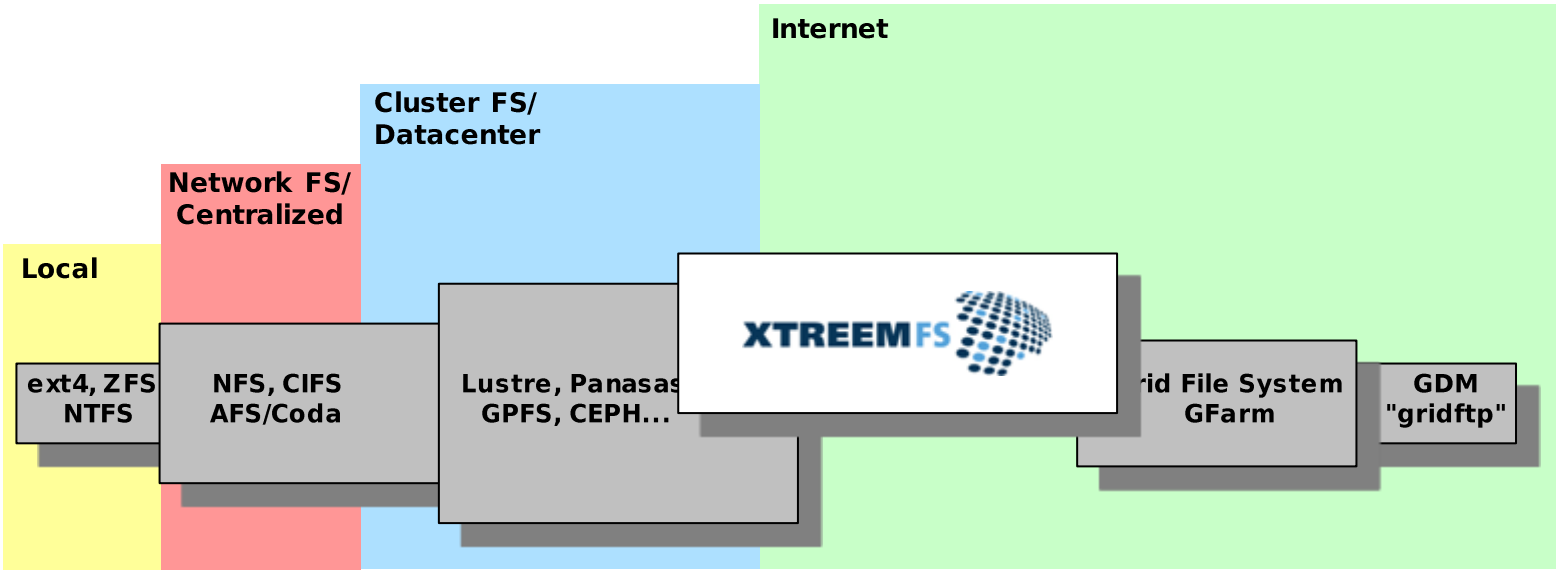
Christoph Kleineweber
Zuse Institute Berlin
kleineweber@zib.de

HARNESS Software Carpentry Workshop
July 17th, 2015

Outline

- **(Distributed) file systems**
- Introduction to XtreemFS
- Use-cases for XtreemFS
- Setting up your own XtreemFS cluster

File System Landscape



How to Use a File System in Python?

Writing to a file:

```
f = open(“myfile.txt”, ” w”)
f.write(“Hello World\n”)
f.close()
```

Reading a file:

```
f = open(“myfile.txt”, “r”)
for line in f:
    print(line)
f.close()
```

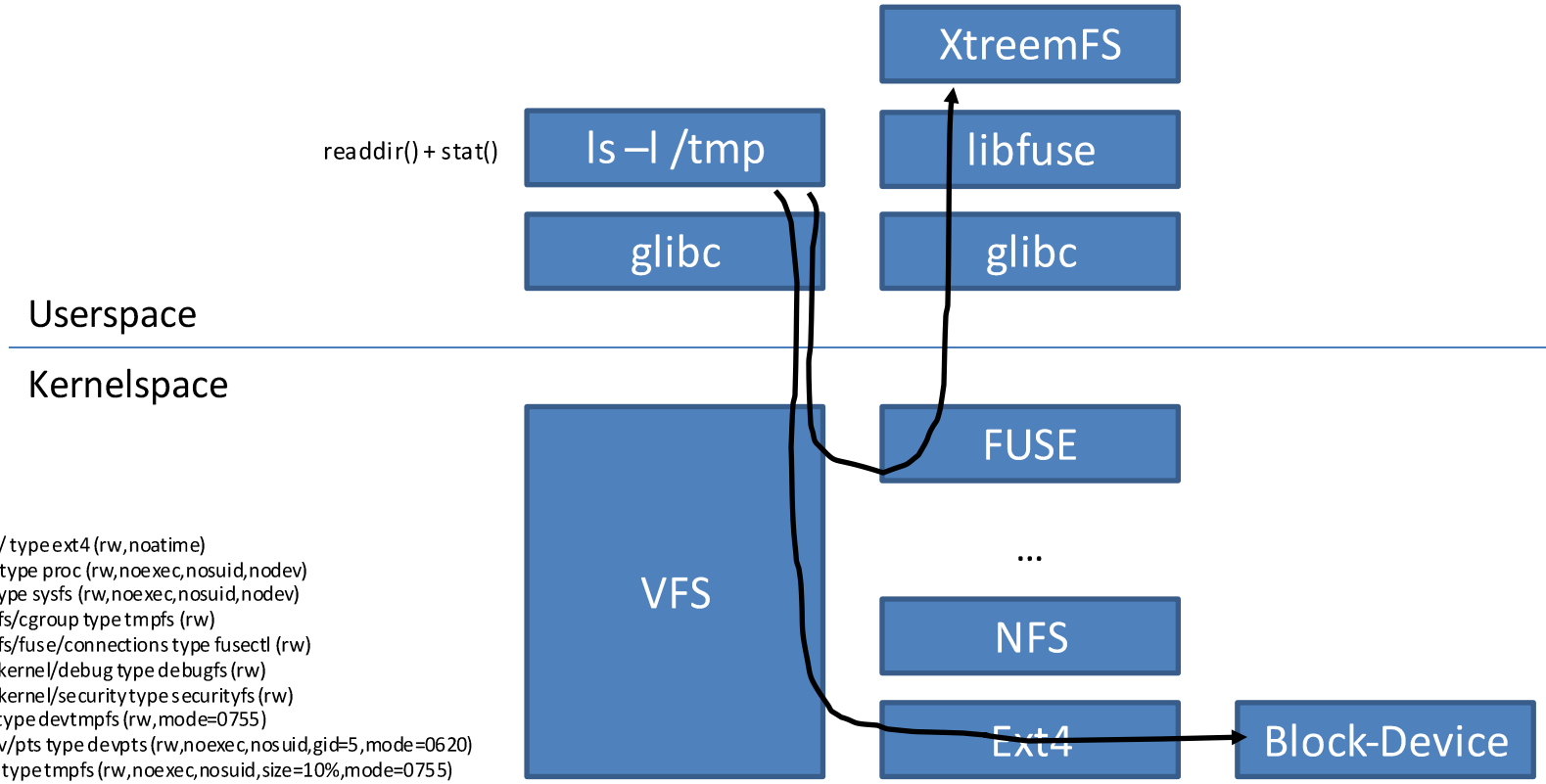
Reading directory entries:

```
import os
for f in os.listdir(“/”):
    print(f)
```

Reading file attributes:

```
import os
file_info = os.stat(“myfile.txt”)
print(file_info)
```

File Systems on Linux with FUSE




```
$ mount
/dev/sda1 on / type ext4 (rw,noatime)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/cgroup type tmpfs (rw)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
none on /run/user type tmpfs (rw,noexec,nosuid,nodev,size=104857600,mode=0755)
none on /sys/fs/pstore type pstore (rw)
/dev/sda2 on /local type ext4 (rw,nosuid,nodev,noatime)
rpc_pipefs on /run/rpc_pipefs type rpc_pipefs (rw)
systemd on /sys/fs/cgroup/systemd type cgroup (rw,noexec,nosuid,nodev,none,name=systemd)c
```

Distributed File Systems

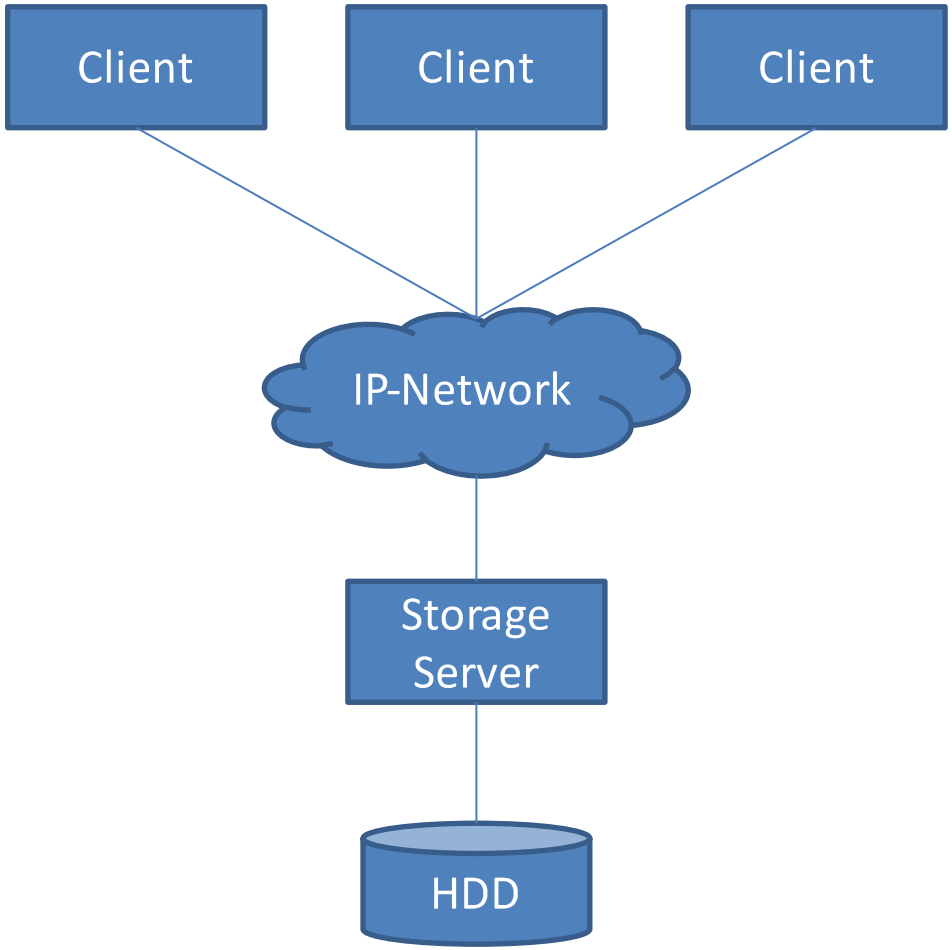
Goals:

- Transparency
  - Clients are unaware of file distribution and locality
  - Same behavior as a local file system
- Shared file access
  - Multiple users and clients
- Scalability, fault tolerance, consistency, performance, …

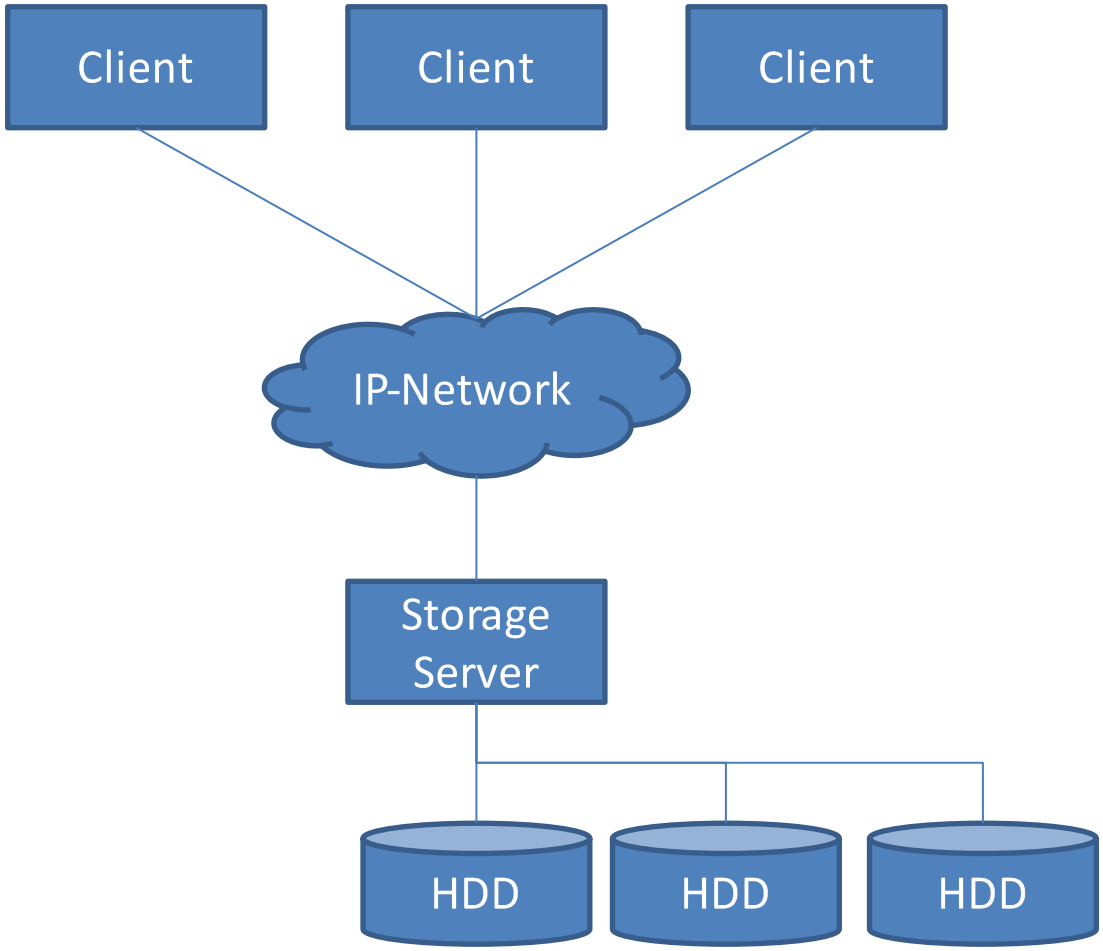
Use-cases:

- High performance computing
  - Multiple computers write checkpoints to a single file
  - Data analytics (e.g. MapReduce)
- Storing home directories
  - User wants the same environment on any PC
- Virtualization
  - Storing VM images

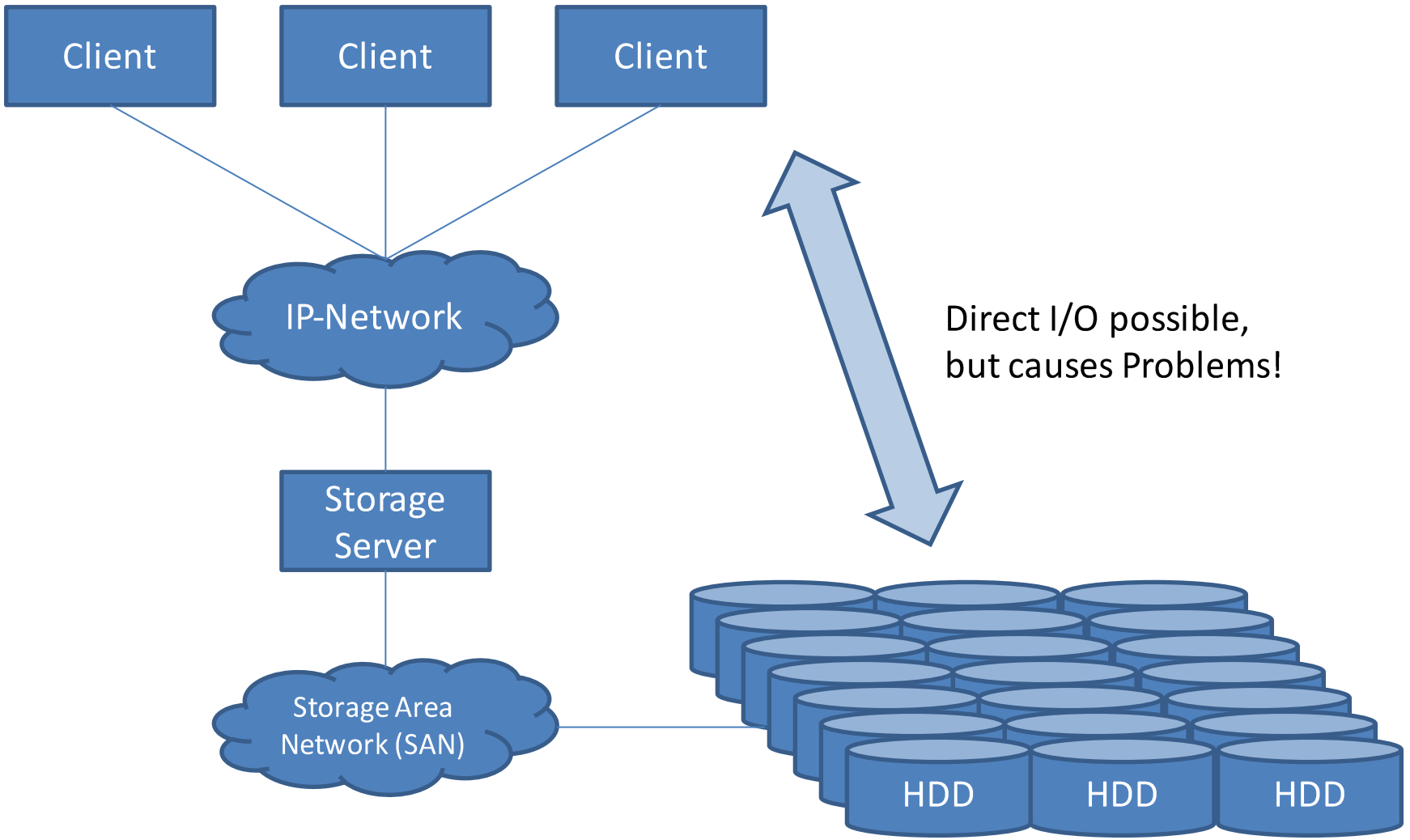
Scalable Storage Systems



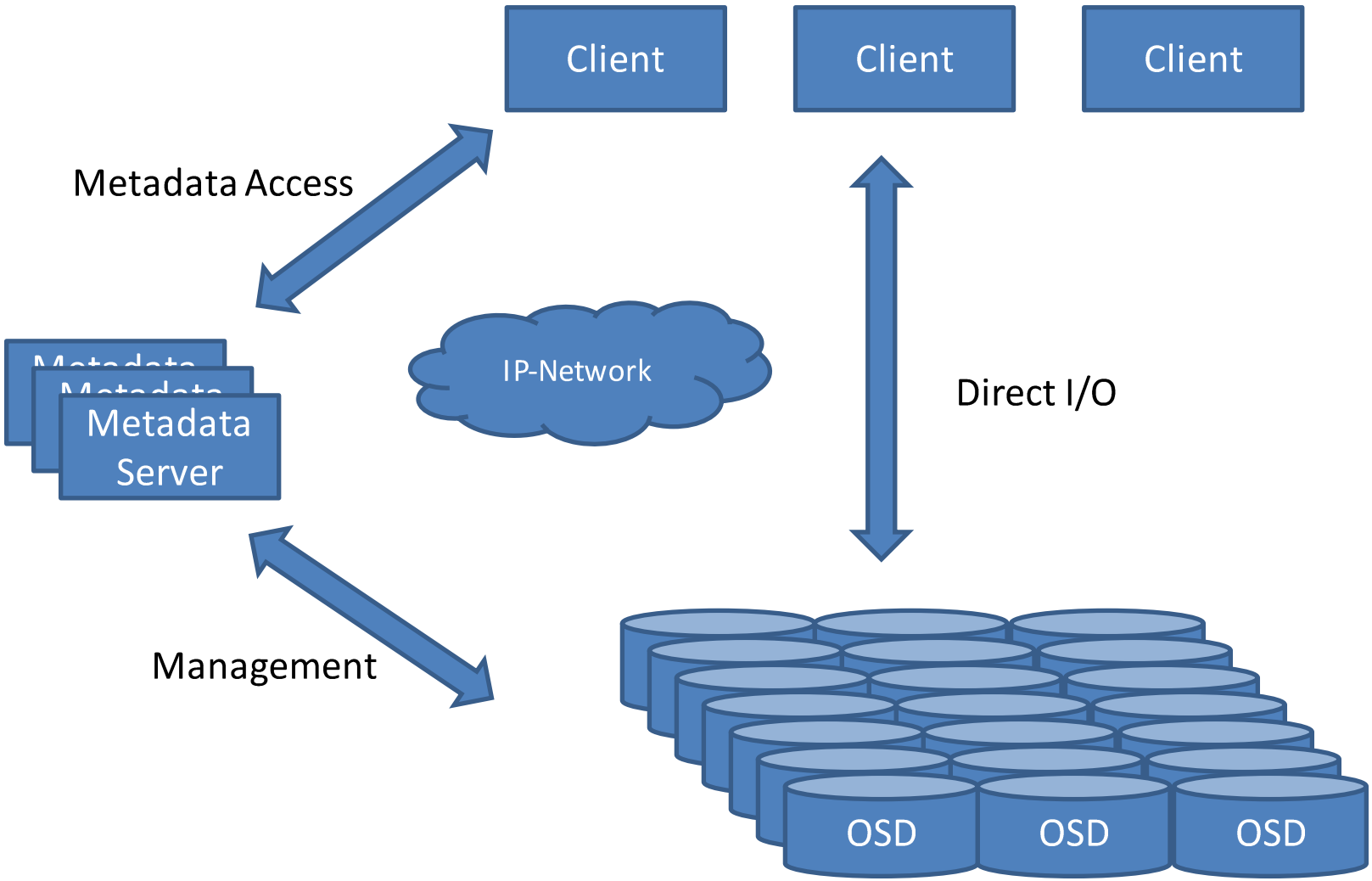
Scalable Storage Systems



Scalable Storage Systems



Object-based Storage



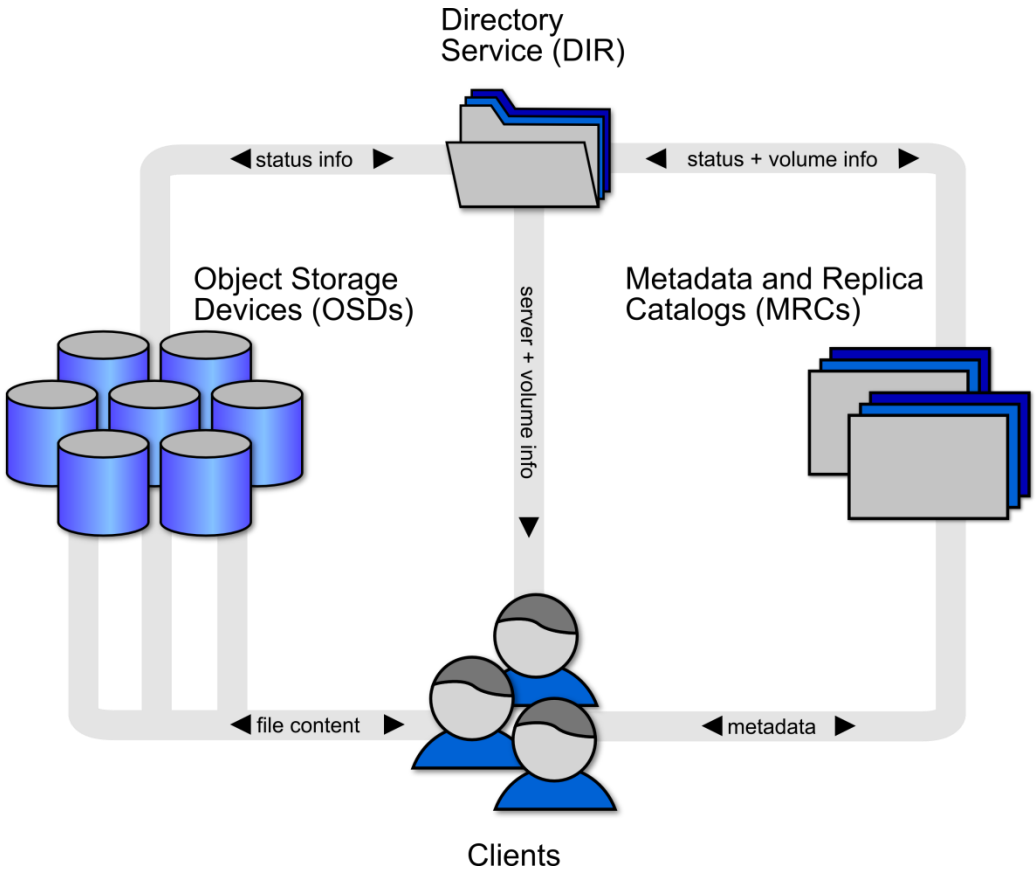
Outline

- (Distributed) file systems
- **Introduction to XtreemFS**
- Use-cases for XtreemFS
- Setting up your own XtreemFS cluster

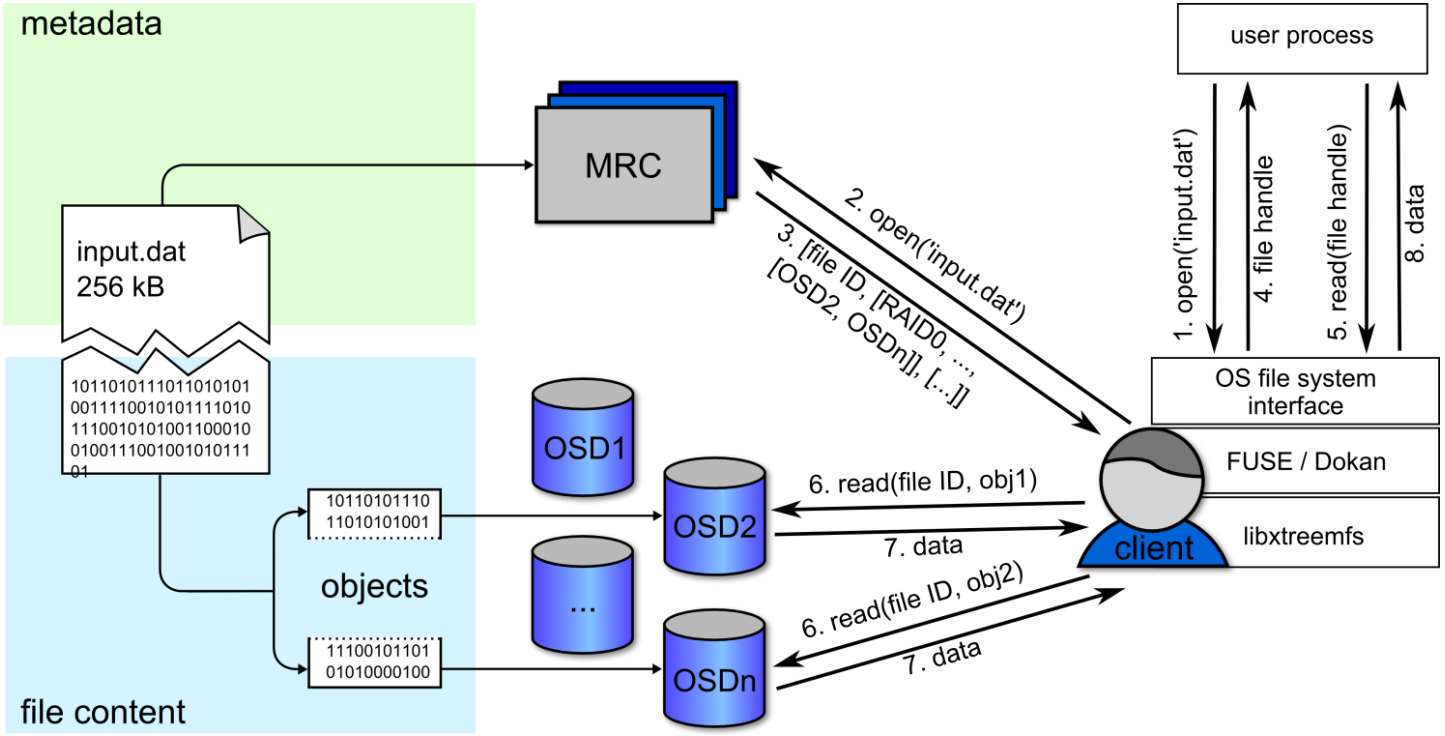
XtreemFS

- History
  - 2006 initial development in XtreemOS project
  - Since 2010 further development in different research projects
  - 2015 latest release 1.5.1 in March
- Project
  - Open source http://www.xtreemfs.org (BSD License)
  - POSIX compatible distributed file system
- Software
  - Client software (C++) runs on Linux, Mac OS X and Windows
  - Server software platform independent (Java)
  - Client library libxtreemfs for C++ and Java
  - Hadoop driver based on Java libxtreemfs

XtreemFS Architecture

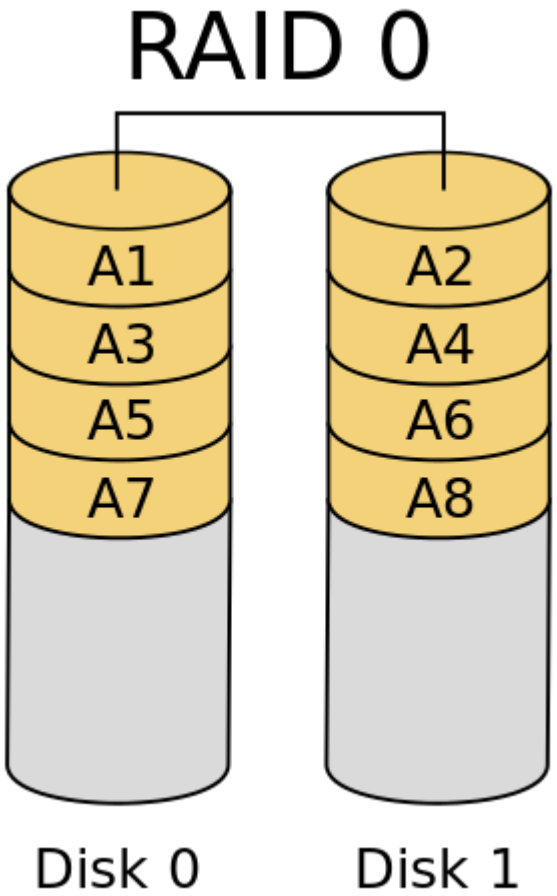


File Access


metadata
input.dat
256 kB
1011010111011010101
0011110010101111010
1110010101001100010
0100111001001010111
01
MRC
user process
1. open('input.dat')
2. open('input.dat')
3. [file ID, [RAID0, ...,
[OSD2, OSDn]], [...]]
4. file handle
5. read(file handle)
8. data
OS file system
interface
FUSE / Dokan
libxtreemfs
client
OSD1
OSD2
...
OSDn
10110101110
11010101001
objects
11100101101
01010000100
6. read(file ID, obj1)
7. data
6. read(file ID, obj2)
7. data
file content

Striping Files over OSDs

- Store file segments on different OSDs
  - Similar to RAID0
- File size can exceed single server capacity
- Possible performance gain by parallel access
- Caution: Device failure corrupts the whole data!



https://en.wikipedia.org/wiki/Standard_RAID_levels

Replication

Why?

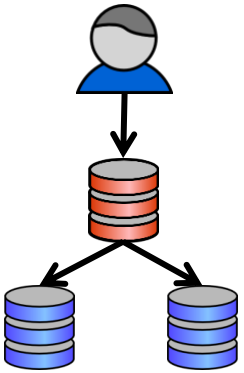
- Reliability
  - Access your data even in the case of failing servers
- Performance
  - Spread the load to multiple servers

Problems:

- Which replica holds the latest data?
  - Strong consistency desired!
- How to avoid “split-brain” situations?

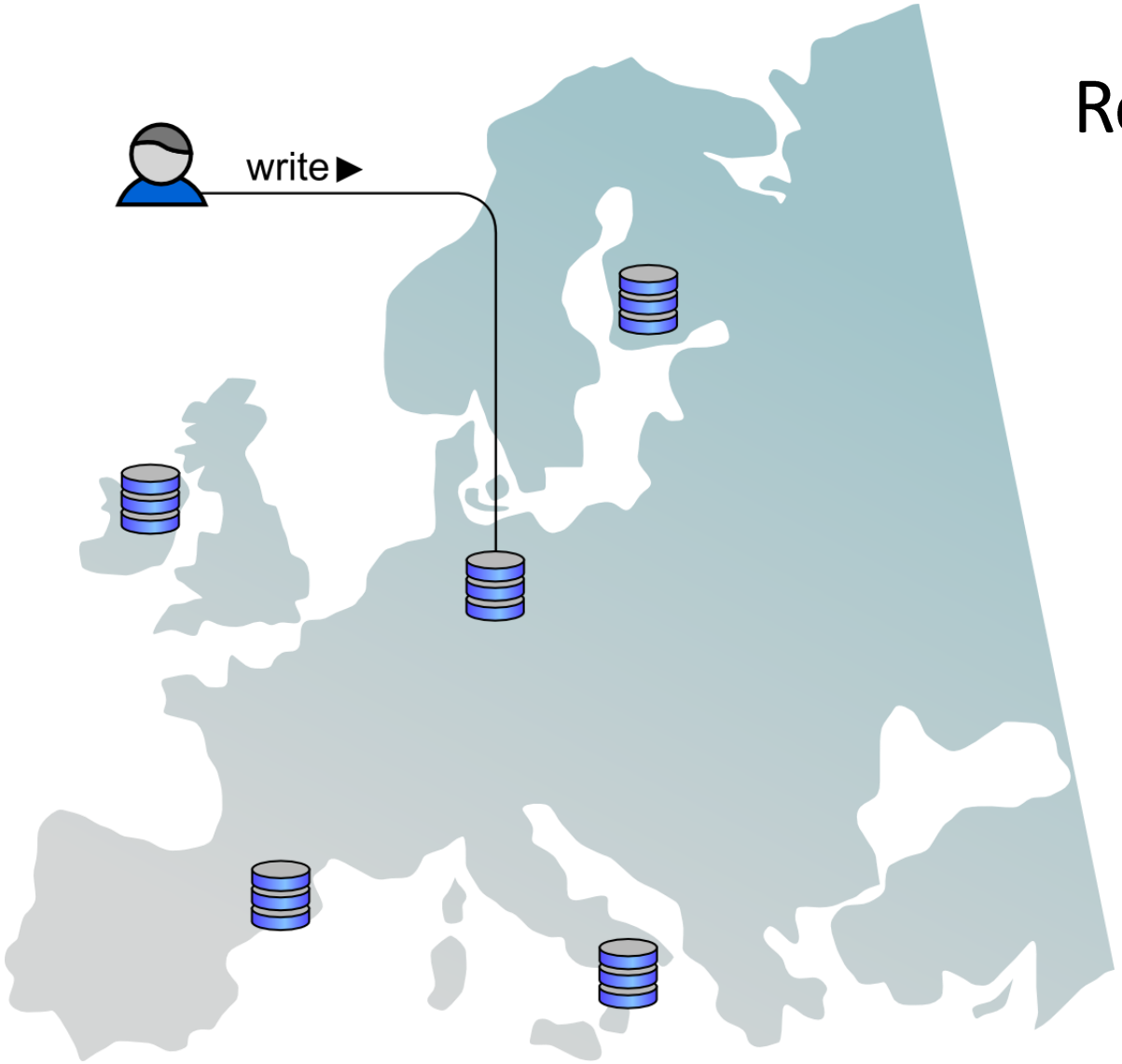
Replication with Strong Consistency

- Primary/backup scheme
  - POSIX requires total order of update operations
    →Primary/backup
  - Primary fail-over?



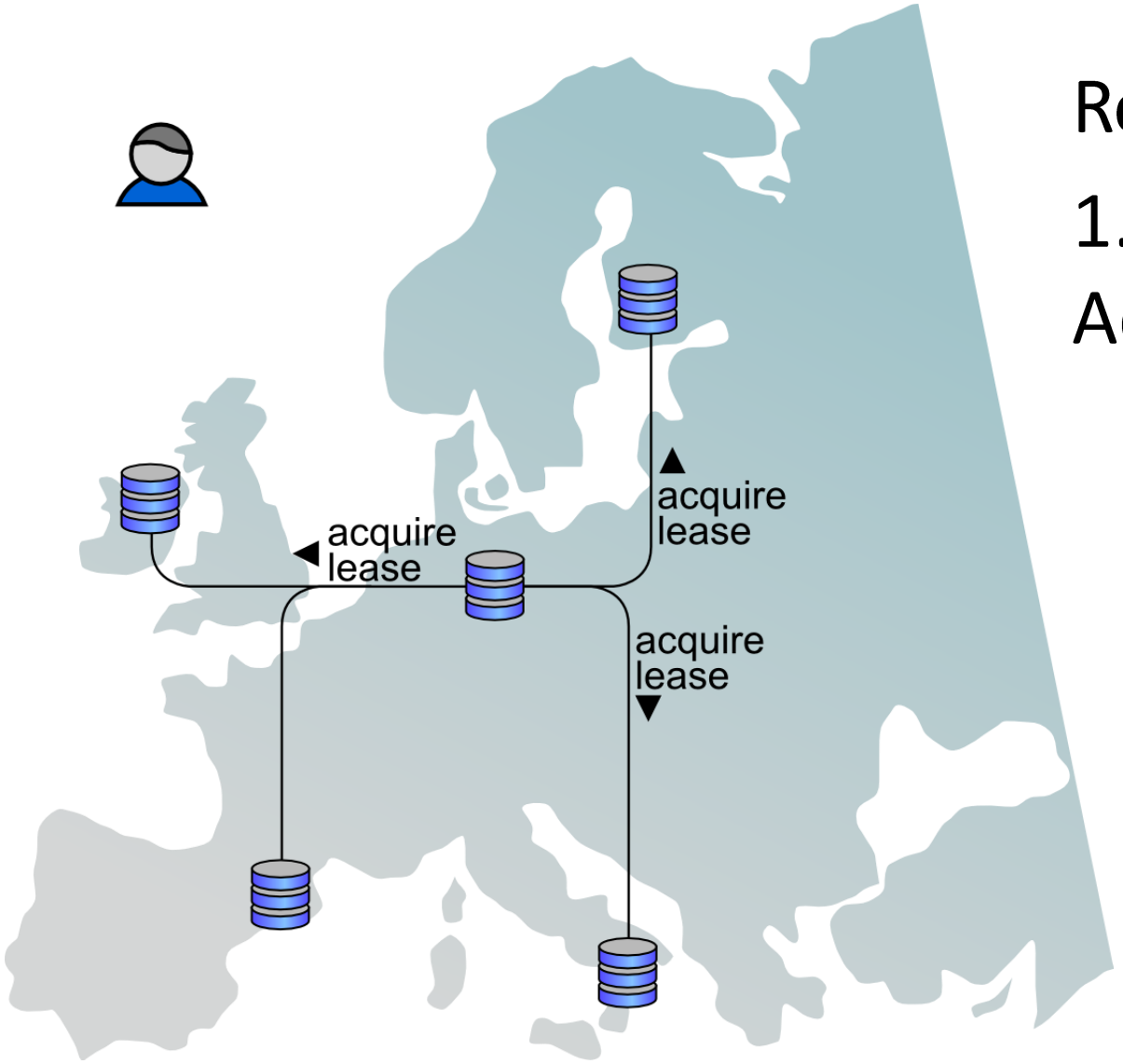
- Leases
  - Grants access to a resource (here: primary role)
    for a predefined period of time
    - Failover after timeout possible
  - Assumption: loosely synchronized clocks
    - Max drift ε

Example: Replicated Write



Replicated write():

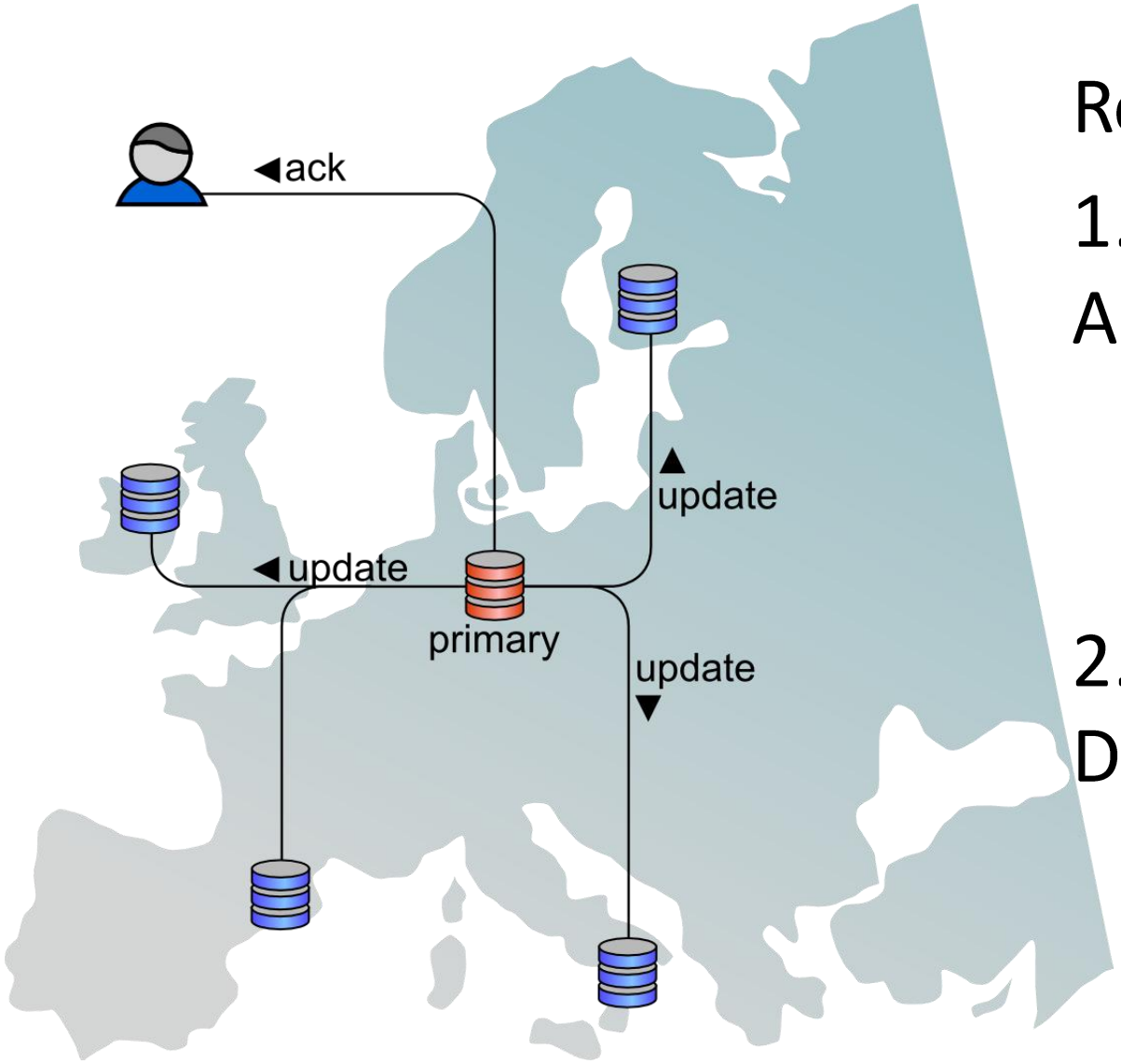
Example: Replicated Write



Replicated write():

1. Lease Acquisition

Example: Replicated Write

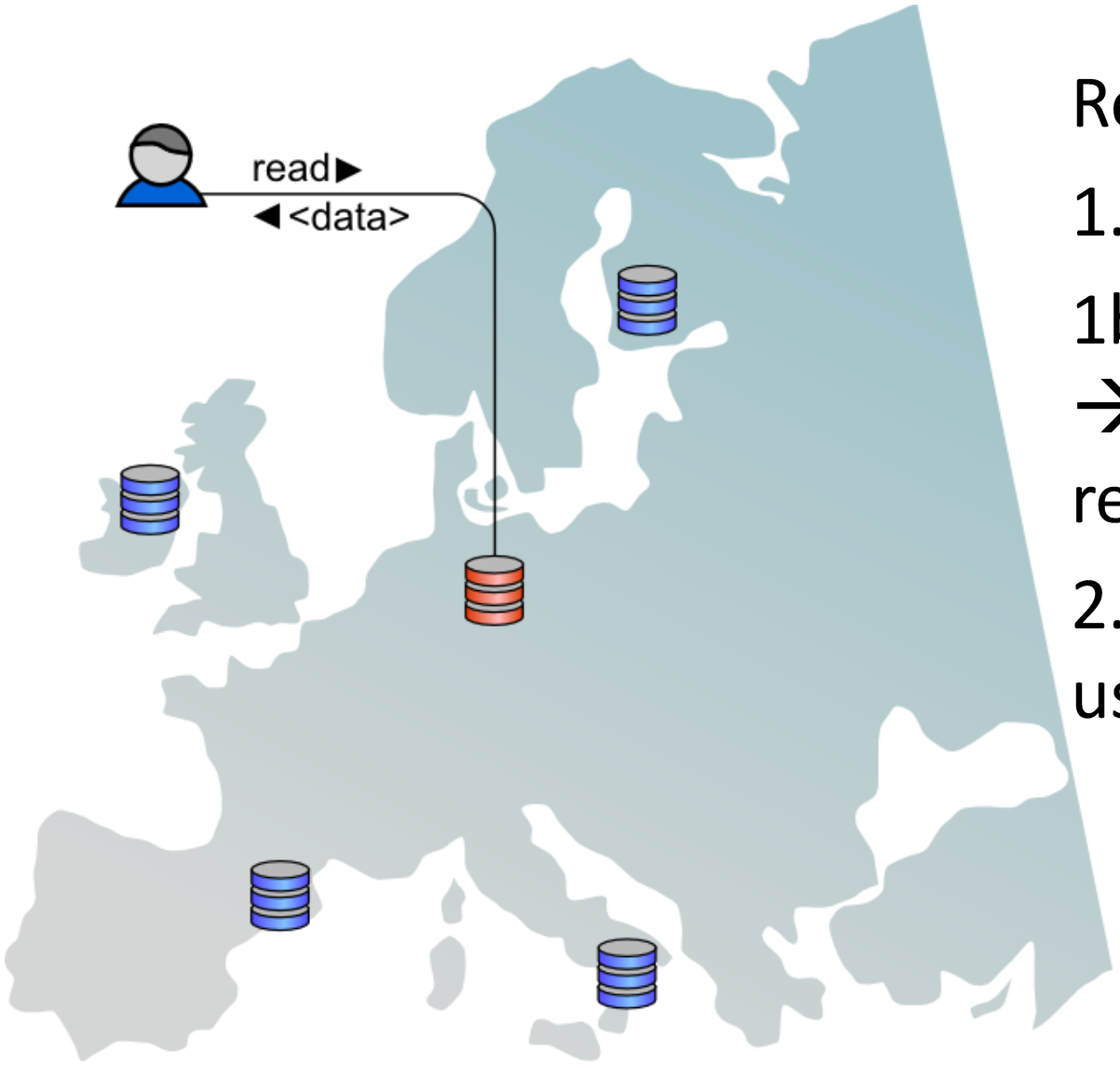


Replicated write():

1. Lease Acquisition

2. Data Dissemination

Example: Replicated Read



Replicated read():

1. Lease Acquisition

1b. “Replica Reset” → update primary’s replica

2. Respond to read() using local replica

Replication of Immutable Files

- Only for “write-once” files
  - File must be marked as “read-only”
    - done automatically after close()
  - Use Case: CDN
    - client can read from and replica

- Replica Types:
  1. Full replicas
     - complete copy, fills itself as fast as possible
  2. Partial replicas
     - Initially empty
     - on-demand fetching of missing objects
  - P2P-like efficient transfer between all replicas

File System Snapshots

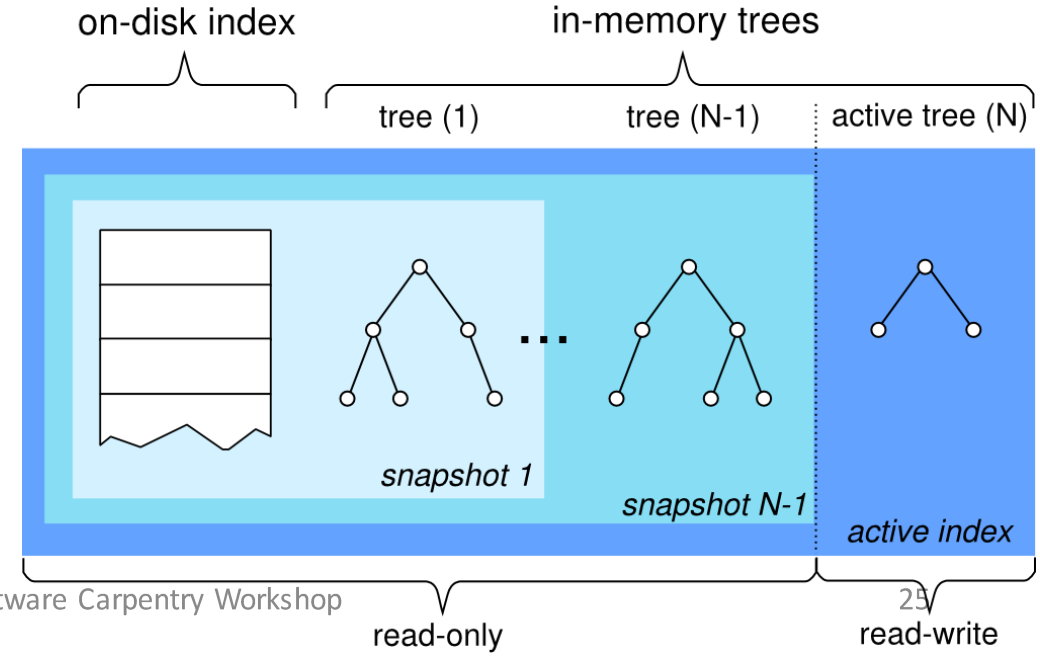
- Persistent file system content
- Use-cases:
  - File versioning
  - Get consistent state for backups
- Challenge:
  - Files are spread across multiple servers

Snapshots in XtreemFS

- Use copy-on-write on OSDs
  - Objects are not overwritten
  - New version for each write
- Take metadata Snapshot on MRC
- Periodic cleanup process deletes object versions that are not assigned to snapshots

Metadata Storage

- DIR and MRC use key-value store BabuDB for persistent storage
- Uses log-structured merge-tree internally
  - Known from Google's BigTable
  - Well suited for snapshots
- DB replication for fault tolerance
  - Master replica holds lease
  - Majority of replicas have to be updated



Outline

- (Distributed) file systems
- Introduction to XtreemFS
- **Use-cases for XtreemFS**
- Setting up your own XtreemFS cluster

Use-Case: XtreemFS for VM Images

- VM deployment
  - Create copy (clone) of original VM image
  - Run cloned VM image on hypervisor
  - (Discard cloned image after VM shutdown)
- Problems
  - Cloning is time-consuming
  - Waste of space (redundancy when running multiple instances of same image)

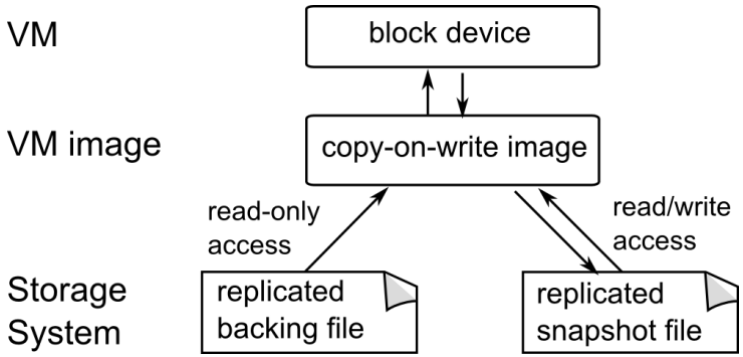
XtreemFS Replication for VM Storage

- qcow2 VM image format
  - Allows snapshots
    1. Immutable backing file
    2. Mutable, initially empty snapshot file

    → Instead of cloning, snapshot original VM image (< 1 second)

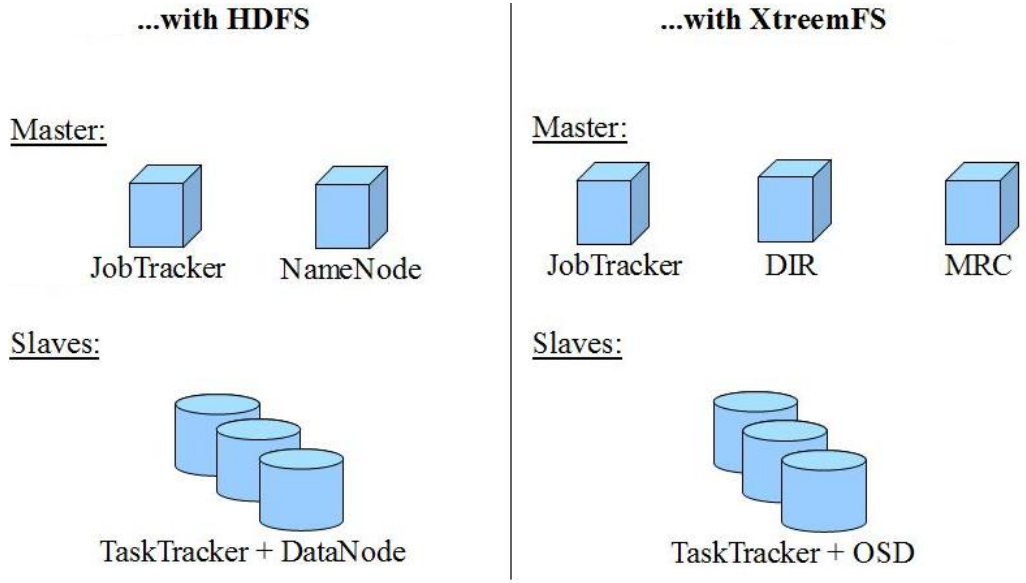
    → Use Read/Write replication for snapshot file

- Problem left: run multiple VMs simultaneously
  - Snapshot file: R/W replication scales with # OSDs and # files
  - Backing file: bottle neck
    - Use *immutable file* replication



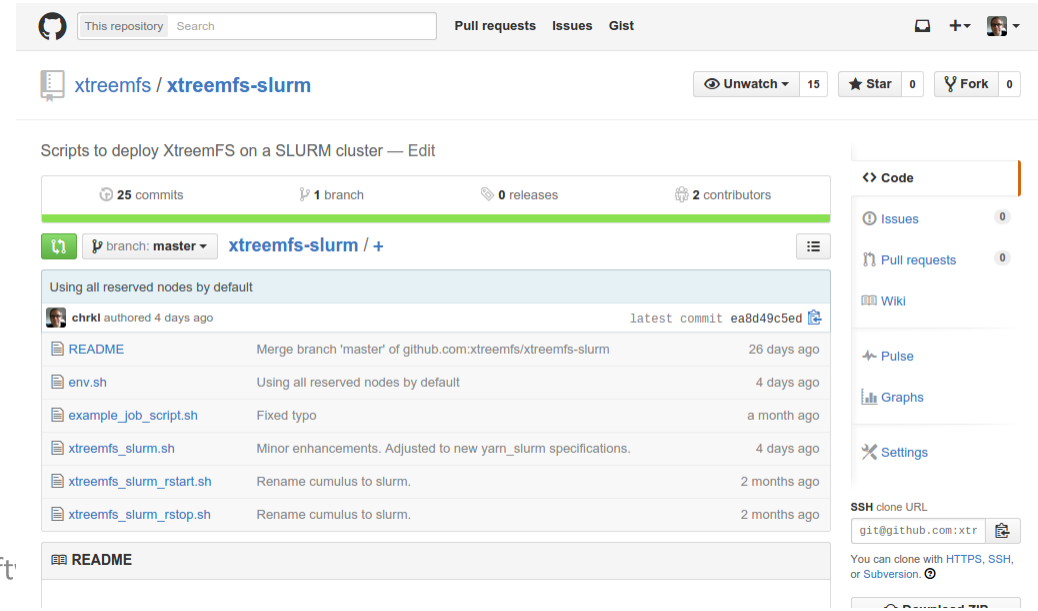
Use-Case: XtreemFS for Data Analytics

- Replace HDFS in Hadoop cluster
  - Avoid limitations of HDFS
  - Hadoop file system driver based on libxtreemfs



XtreemFS on Demand

- XtreemFS runs on commodity hardware
  - Often with large SATA disks
  - Slightly utilized disks on compute clusters
- Scripts to deploy XtreemFS on SLURM are available!



Questions?

Website:	http://www.xtreemfs.org
Mailinglist:	xtreemfs@googlegroups.com
Repository:	https://github.com/xtreemfs/xtreemfs
Twitter:	@xtreemfs

Outline

- (Distributed) file systems
- Introduction to XtreemFS
- Use-cases for XtreemFS
- **Setting up your own XtreemFS cluster**

Hands-on Session

- Getting familiar with Vagrant
- Starting XtreemFS services
- Creating and mounting volumes
- Configure striping
- Configure replications
- Creating snapshots