

# AFS + Object Storage

Hartmut Reuter  
[reuter@rzg.mpg.de](mailto:reuter@rzg.mpg.de)

Rainer Többsicke, CERN, Switzerland  
Andrei Maslennikov, Ludovico Giammarino, Roberto Belloni, CASPUR, Italy  
Hartmut Reuter, RZG, Germany

# What is object storage

- Object storage systems are distributed filesystems which store data in object storage devices (OSD) and keep metadata in metadata servers.
- Access to a file in object storage on a client consists in the following steps:
  - directory lookup of the file
  - rpc to metadata server which checks permissions and returns for each object the file consists of a special encrypted handle.
  - rpc to the OSDs to read or write objects using the handle obtained from the metadata server
    - The OSD can decrypt the handle by use of a secret shared between him and the metadata server. The advantage of this technique is that the OSD doesn't need any knowledge about users and access rights.
- The most popular object storage systems are **Lustre** and **Panasas**
- Compared to parallel filesystems such as GPFS the advantage of object storage is
  - clients cannot corrupt the filesystem
  - permission checks are done on the metadata server not on the client.
- Disadvantages of today's object storage systems
  - limited number of platforms (Lustre: only Linux, Panasas: ?)
  - no world-wide access to data in object storage (only a cluster filesystem)

# Why use object storage for AFS

- AFS infra-structure has already many components necessary for good object storage.
  - central user authentication and registration
  - AFS fileserver could act as OSD-metadata server allowing for better scalability than in other object storage systems.
  - OSDs for AFS could use rx-protocol and NAMEI-partitions, both components available for all platforms.
- Use of OSDs could
  - remove 'hot spots' in very active volumes by distributing data over many OSDs
  - increase peak performance by striping of files (HPC environment)
  - allow for RW-replication by use of mirrored objects
- Implementing object storage in the AFS environment would allow to use objects storage on any platform (not just Linux) and to use it world wide.
- AFS with object storage could offer additional features such as data migration (HSM)
  - The last MR-AFS site (RZG) could come home into the lap of OpenAFS

## Creation of the AFS+OSD project

- In 2004 Rainer Többicke from CERN implemented a first version of AFS+OSD as a proof of concept. It was tested at the CASPUR StorageLab in Rome.
  - However, Rainer couldn't find the time to develop his idea any further
- In 2005 Andrei Maslennikov from CASPUR managed to bring the interested institutions and persons together to form a real project.
  - Design decisions were made on a meeting at CERN
  - Funds were raised from CERN, and ENEA to hire system programmers to work at CASPUR

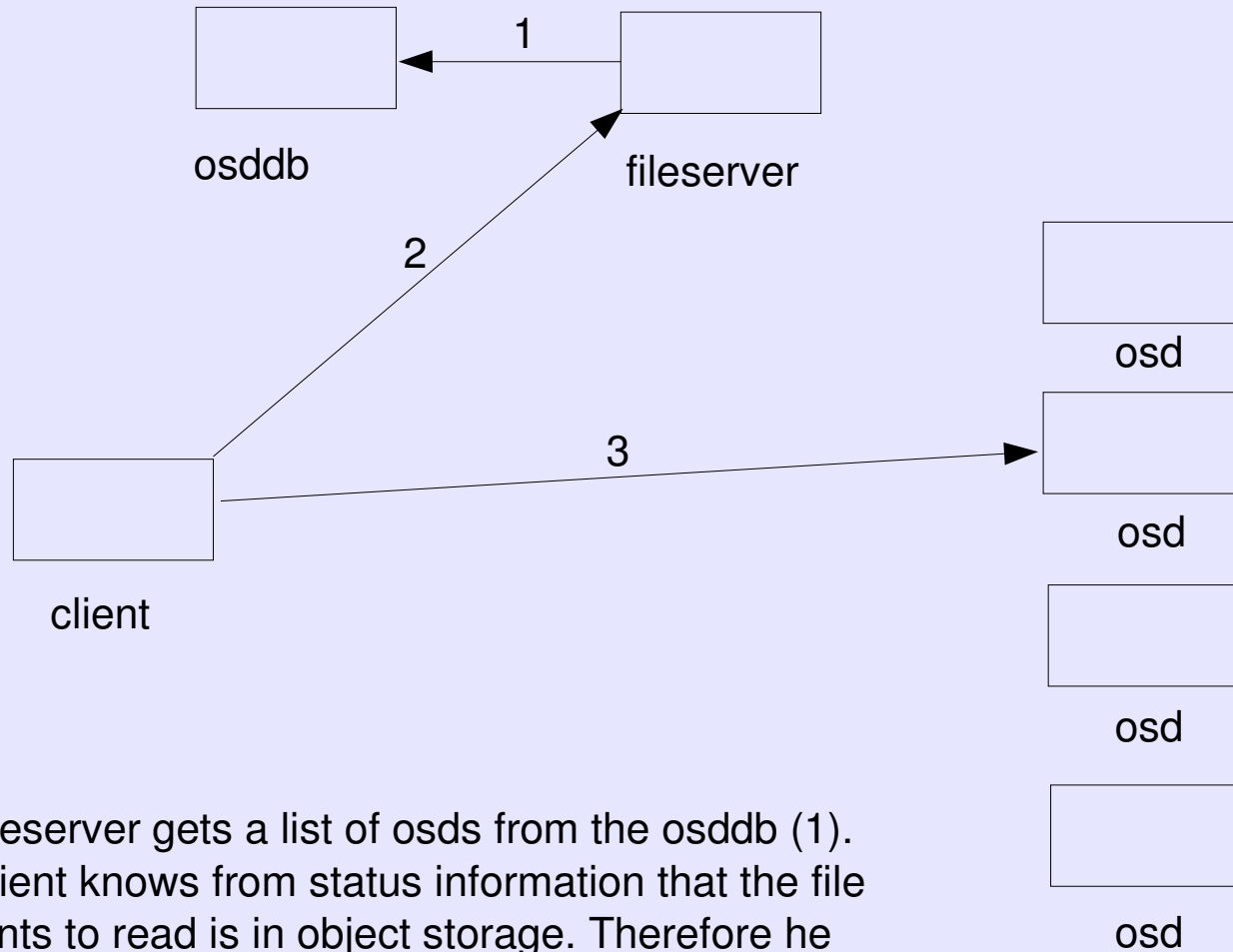
# The T10 standard

- T10 Technical Committee on SCSI Storage Interfaces defines standards for SCSI commands and devices. Two subgroups are working on object storage:
  - Object-Based Storage Device Commands (OSD)
  - Object-Based Storage Devices - 2 (OSD-2)
- There have been published some drafts about the OSD-standard which we read and analyzed. As far as possible we tried to follow these standards:
  - For each object
    - 64-bit object-id      we use vnode and uniquifier of the AFS Fid
    - 64-bit partition-id    we use the volume-id of the RW-volume
  - A data structure “cdb” is used in SCSI commands to OSDs.
    - we use a sub-structure of it to transport the encrypted object-handle
- It turned out that the T10 standard is not rich enough to support full AFS semantic
  - link counts for objects needed for volume replication are missing
  - the security model is too weak for insecure networks

# Components of AFS + OSD

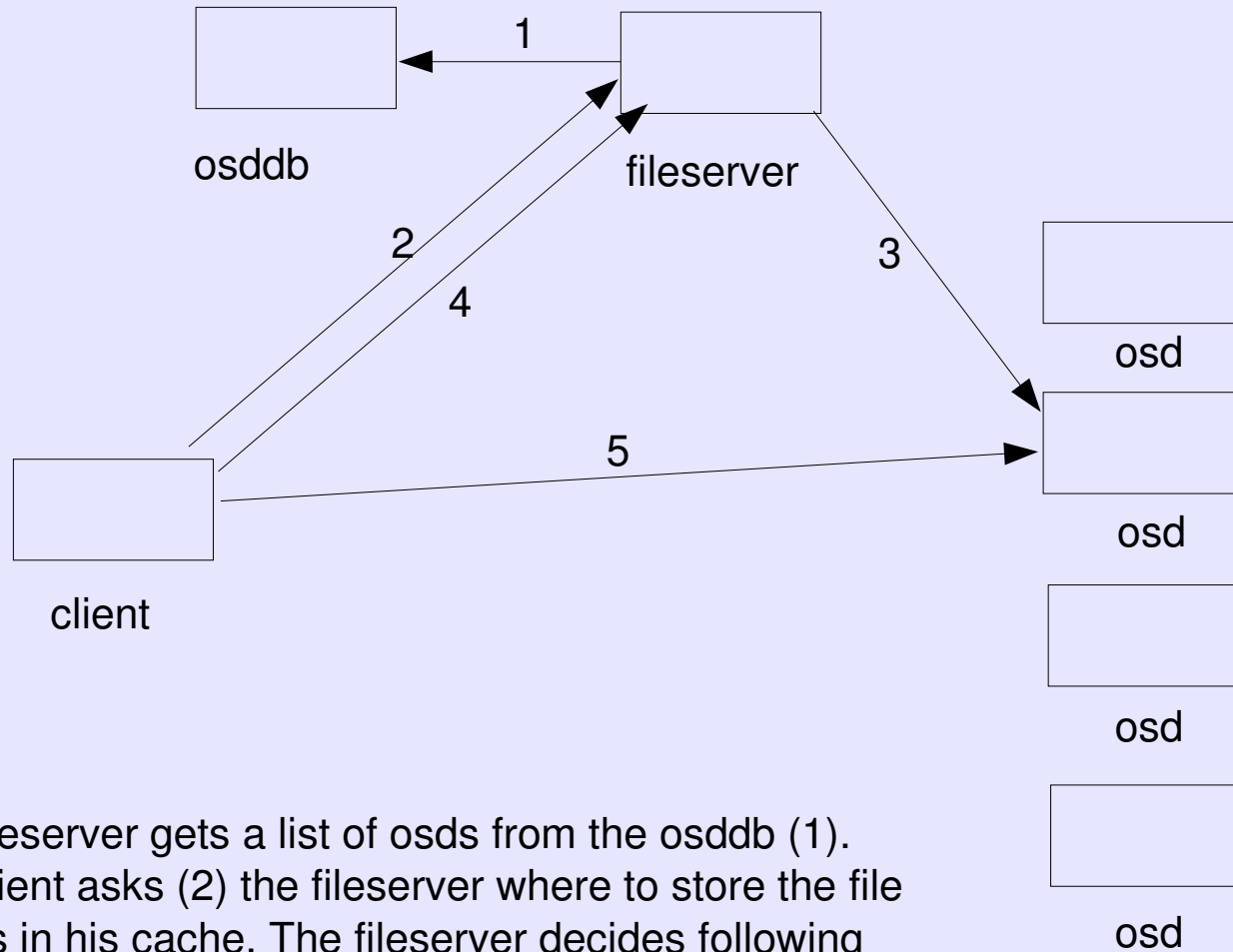
- „rxosd“ the object storage server (OSD) used for AFS
  - that's where the data are stored.
- „osddb“ a database (similar to vldb)
  - which describes all OSDs and may contain also different policies to be used in different AFS volumes.
- AFS-fileserver, acting as metadata-server
  - metadata are kept in a new volume-special file
  - new RPCs
  - extensions also to volserver and salvager
- AFS-client should behave as ever, but
  - should store and fetch data directly from OSDs
- Legacy-interface
  - to support old clients
  - to allow move of volumes to/from traditional file servers

## Example: reading a file in OSD



The fileserver gets a list of osds from the osddb (1). The client knows from status information that the file he wants to read is in object storage. Therefore he does an rpc (2) to the fileserver to get its location and the permission to read it. The fileserver returns an encrypted handle to the client. With this handle the client gets the data from osd (3).

## Example: writing a file into OSD




The fileserver gets a list of osds from the osddb (1).  
The client asks (2) the fileserver where to store the file he has in his cache. The fileserver decides following his policies to store it in object storage and chooses an osd. He allocates an object on the osd (3).  
The client asks for permission and location of the object (4) getting an encrypted handle which he uses to store the data in the osd (5).

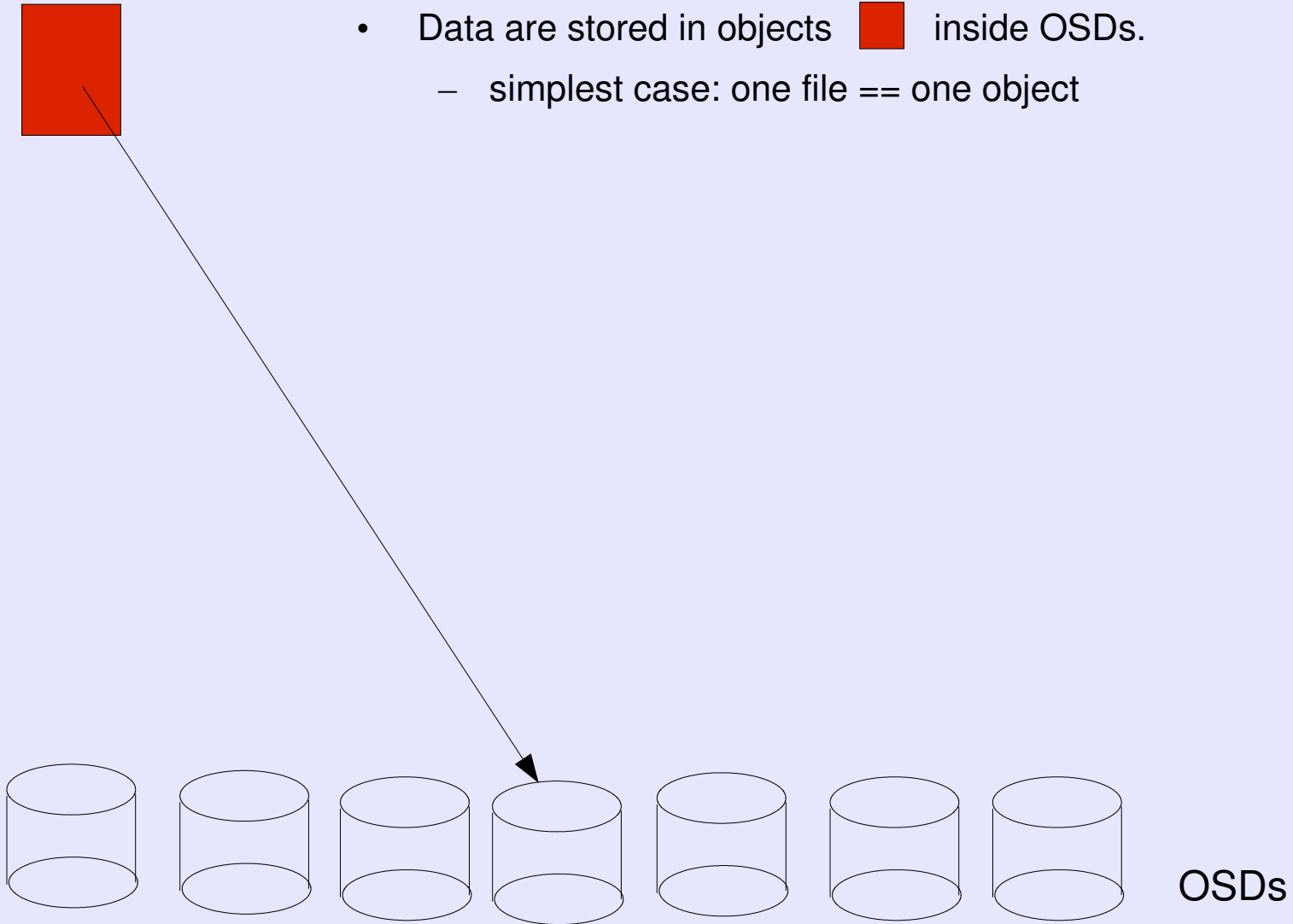


# Design considerations

- Try to follow standards (in this case SCSI T10 standard), however ...
  - following strictly would break AFS semantic (no link counts to implement volume replication)
  - using T10's security model would either break AFS's security or be very expensive
- The data (and metadata) model should be rich enough for future requirements.
  - Files may consist of many objects
  - metadata may contain additional information such as md5-checksums
  - archival versions of files may exist on special archive-OSDs ...
- If possible keep full AFS semantics (volume replication ...)
- Use as much existing mature code as possible
  - rx-protocol
  - NAMEI-interface for storing data inside the OSD servers („rxosd“)
  - ubik for the database describing the OSDs (“osddb”).
- Access to data in object storage should be possible from all clients (also old ones)
  - Implementation of „legacy-interface“ in the fileserver.

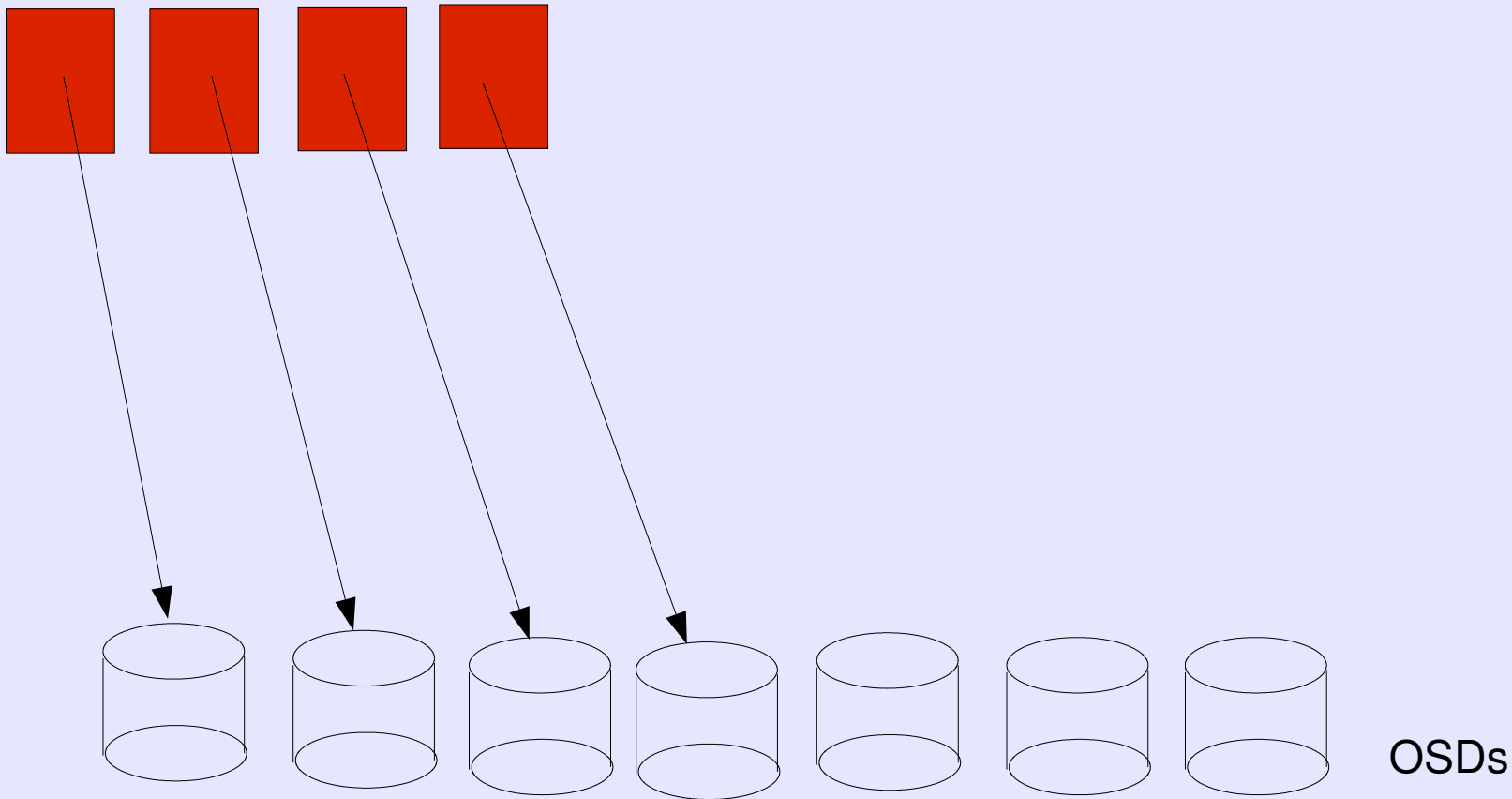
# Object

- Data are stored in objects  inside OSDs.
  - simplest case: one file == one object




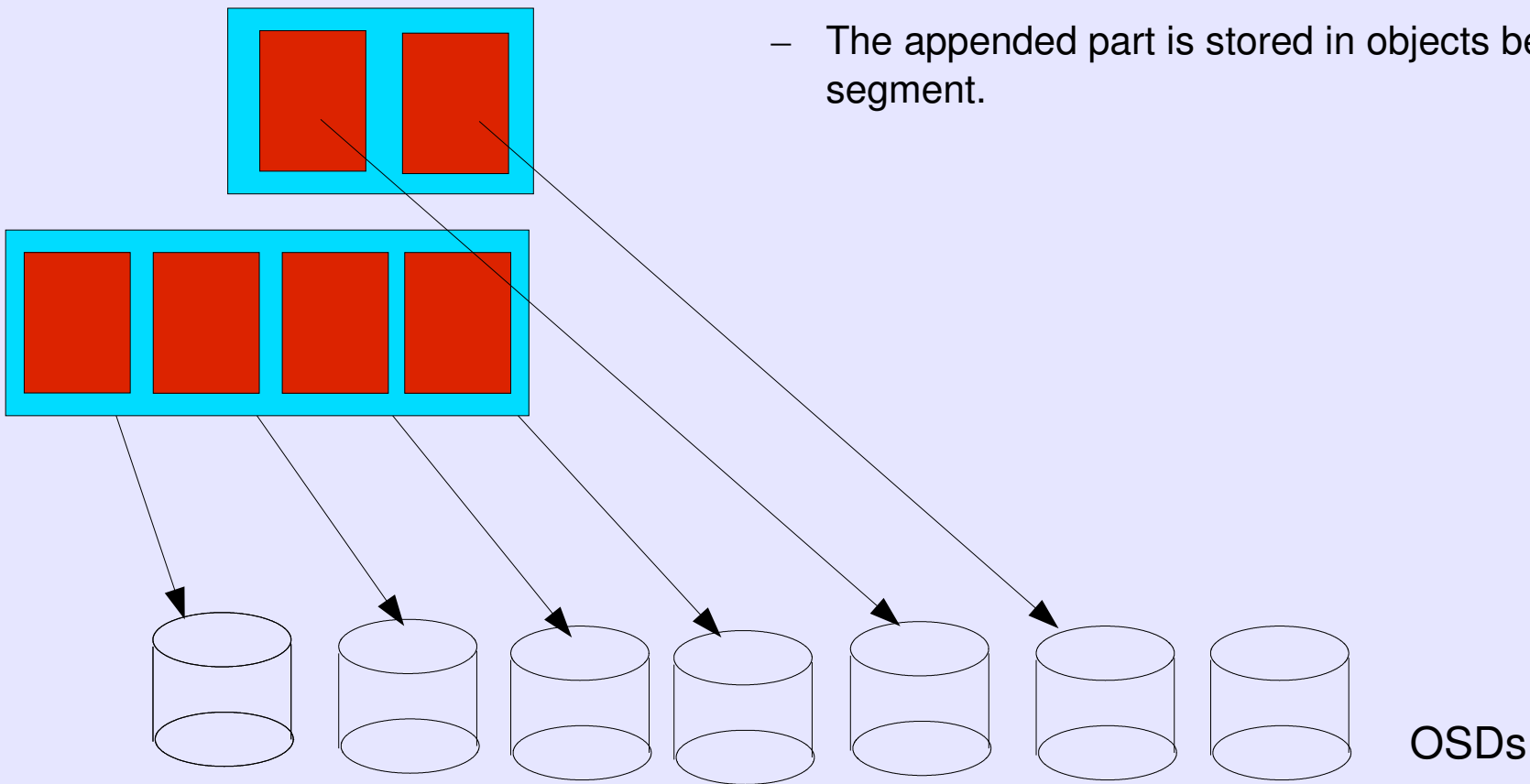
# Objects

- Data of a file could be stored in multiple objects allowing for
  - data striping (up to 8 stripes, each in a separate OSD)
  - data mirroring (up to 8 copies, each in a separate OSD)




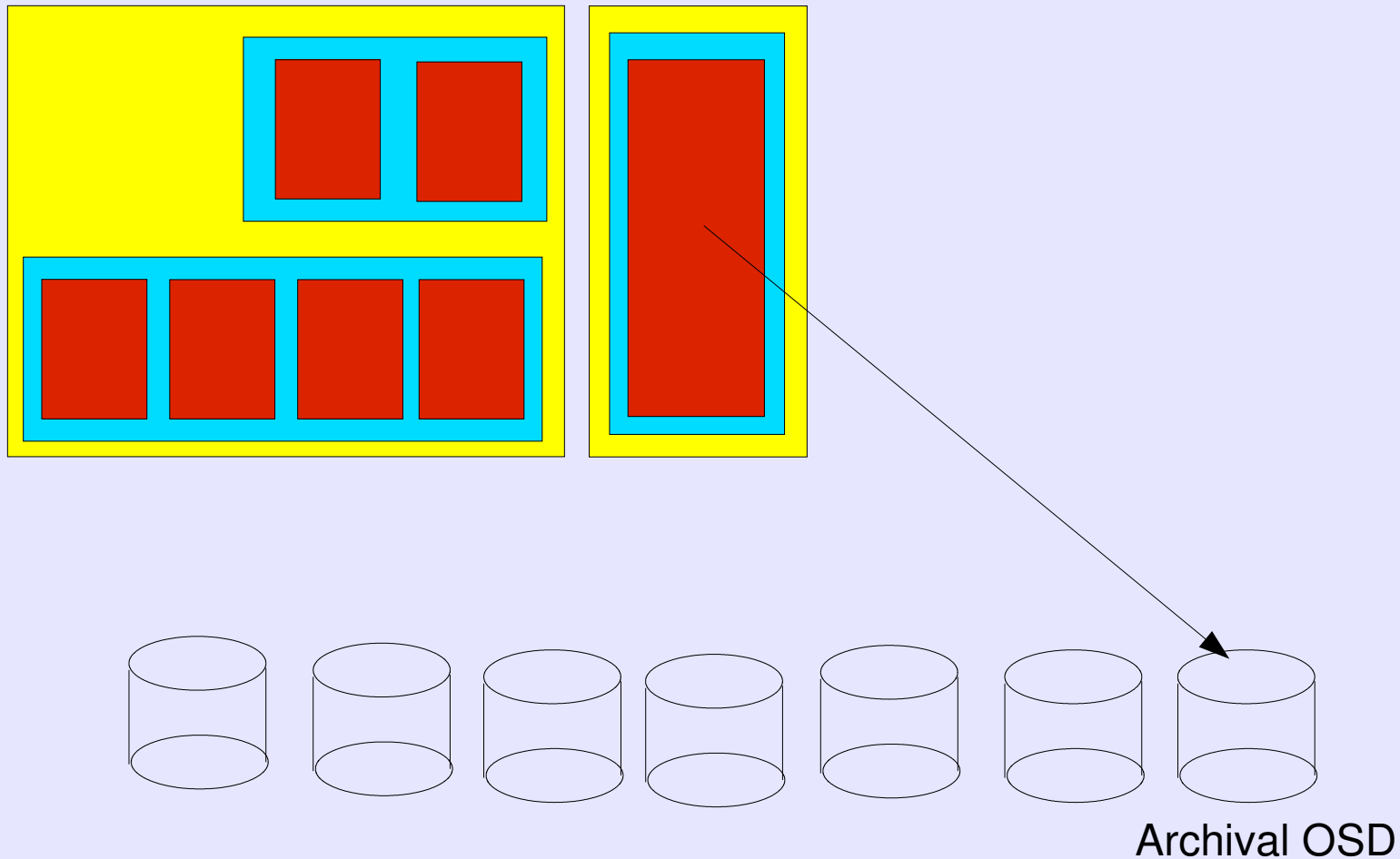
# Objects + Segments

- To a file existing on some OSDs later more data could be appended  
The appended data may be stored on different OSDs (in case there is not enough free space on the old ones)
  - This leads to the concept of segments 
  - The appended part is stored in objects belonging to a new segment.








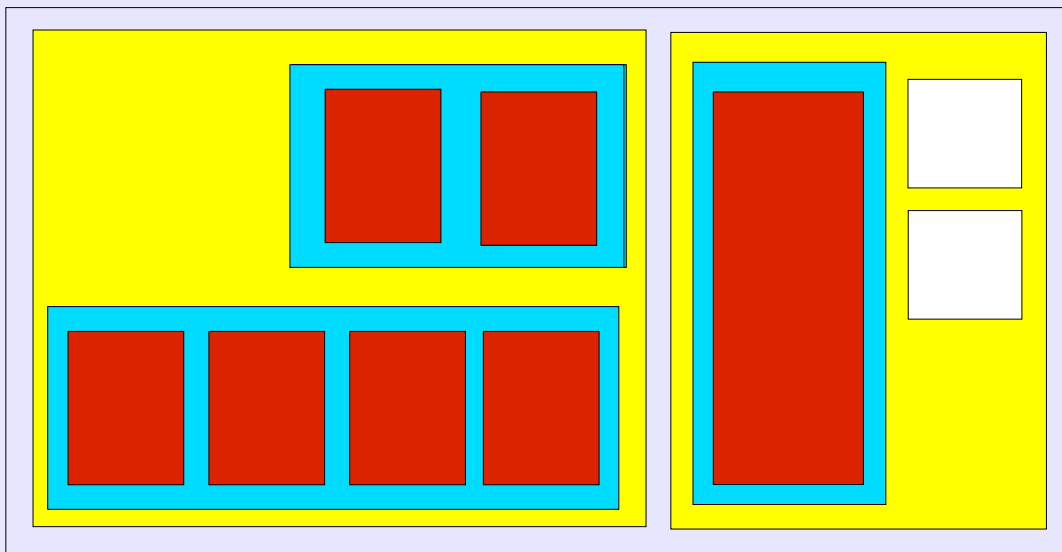
# Objects + Segments + File Copies

- The whole file could get a copy on some archival OSD (tape, HSM)
  - This leads to the concept of file copies 



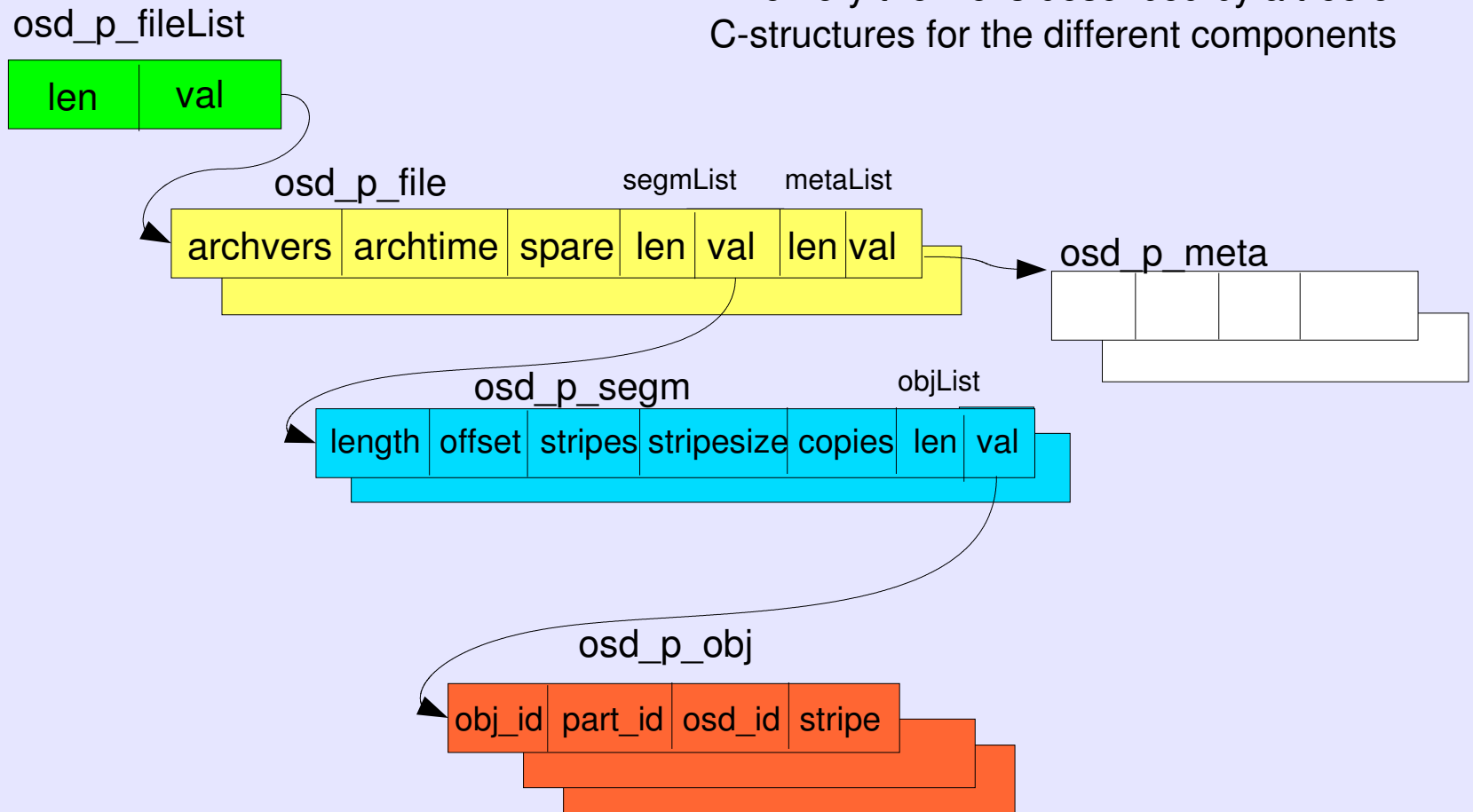
# The osd metadata

- The vnode of an AFS file points to quite a complex structure of osd metadata:
  - Objects  are contained in segments 
  - Segments  are contained in file copies 
  - additional metadata  such as md5-checksums may be included.
- Even in the simplest case of a file stored as a single object the whole hierarchy (file copy, segment, object) exists in the metadata.
- The osd metadata of all files belonging to a volume are stored together in single volume special file
  - This osdmetadata file has slots of constant length which the vnodes point to
  - In case of complicated metadata multiple slots can be concatenated
  - The osd metadata are stored in network byte order to allow easy transfer during volume move or replication to other machines.



# OSD-Metadata

- In memory the file is described by a tree of C-structures for the different components



- These structures are serialized in net-byte-order by means of rxgen-created xdr-routines into slots of the volume special file “osdmetadata”.

# The „osddb“ data base

The „osddb“ database contains entries for all OSDs with id, name, priorities, size ranges ...

The filesystem parameters are updated every 5 minutes by the OSDs.

OSD 1 is a dummy used by the default policy to determine maximum file size in filserver partition

```
~: osd l -v
id name          ---total space---      flag  prior.  ----inodes----  size range
      wr  rd
  1 local_disk
  2 backup        149 gb  22.6 % up  arch  64  64    150 M  0.4 % (4kb-64kb)
  3 raid          299 gb  69.8 % up  arch  64  64    299 M  0.2 % (64kb-1mb)
  5 tape          7442 gb  6.8 % up  arch  64  64     14 M 12.9 % (1mb-100gb)
  9 mpp-fs9-a     11079 gb  81.8 % up      64  64  4095 M  0.0 % (1mb-100gb)
 10 afs4-a        3871 gb  89.6 % up      64  64  1612 M  0.0 % (1mb-100gb)
 11 w7as          2721 gb  59.5 % up      32  32   345 M  0.0 % (1mb-100gb)
 12 test          1869 gb   2.7 % up      80  80  1870 M  0.0 % (1mb-100gb)
 13 hsmgpfs       3139 gb  99.0 % up  arch  80  80  6280 K 42.9 % (1mb-100gb)
 14 afs6-a        1228 gb  89.3 % up      64  64   526 M  0.0 % (1mb-100gb)
 16 mpp-fs4-a     1862 gb  78.7 % up      64  64  1581 M  0.0 % (1mb-100gb)
 17 mpp-fs8-a     2047 gb  79.4 % up      64  64  1680 M  0.0 % (1mb-100gb)
 18 mpp-fs3-a     2047 gb  76.4 % up      64  64  1932 M  0.0 % (1mb-100gb)
 19 mpp-fs10-a    5552 gb  86.6 % up      64  64  2958 M  0.0 % (1mb-100gb)
 20 sfsrv45-a     329 gb  83.9 % up      32  32    10 M  2.4 % (1mb-8mb)
 21 sfsrv45-b     923 gb  72.1 % up      32  32    28 M  0.0 % (8mb-10gb)
 22 mpp-fs2-a     2047 gb  79.8 % up      64  64  1650 M  0.0 % (1mb-100gb)
 32 afs8-a        1023 gb   8.4 % up      80  80  1024 M  0.0 % (1mb-100gb)
~:
```



# Example for OSD metadata

- The new „fs“-subcommand „osd“ shows the OSD-metadata of a file.
  - the three levels are „file“, „segment“, and „object“
  - The 1<sup>st</sup> file entry describes the actual disk file
  - the 2<sup>nd</sup> file entry describes an archival copy with md5-checksum
  - Only the archive contains a segment length, the actual file's length is found in the vnode.

```
~: fs osd 10mb
10mb has 284 bytes of osd metadata, v=3
file archvers=0, archtime: never          , magic ok, 1 segments, flags=0x0
  segment:
    lng=0, ofs=0, raid=0, strps=1, size=0, cop=1, magic ok, 1 objects
    object:
      pid=1108495556, oid=12884901890, osd=32, strp=0, magic ok
      obj=1108495556.2.3.0
file archvers=3, archtime: Dec  7 10:39, magic ok, 1 segments, flags=0x0
  segment:
    lng=10000000, ofs=0, raid=0, strps=1, size=0, cop=0, magic ok, 1 objects
    object:
      pid=1108495556, oid=12884901890, osd=13, strp=0, magic ok
      obj=1108495556.2.3.0
  metadata:
    md5=0376bd622bfecb568aeb2293e49b2696
~:
```

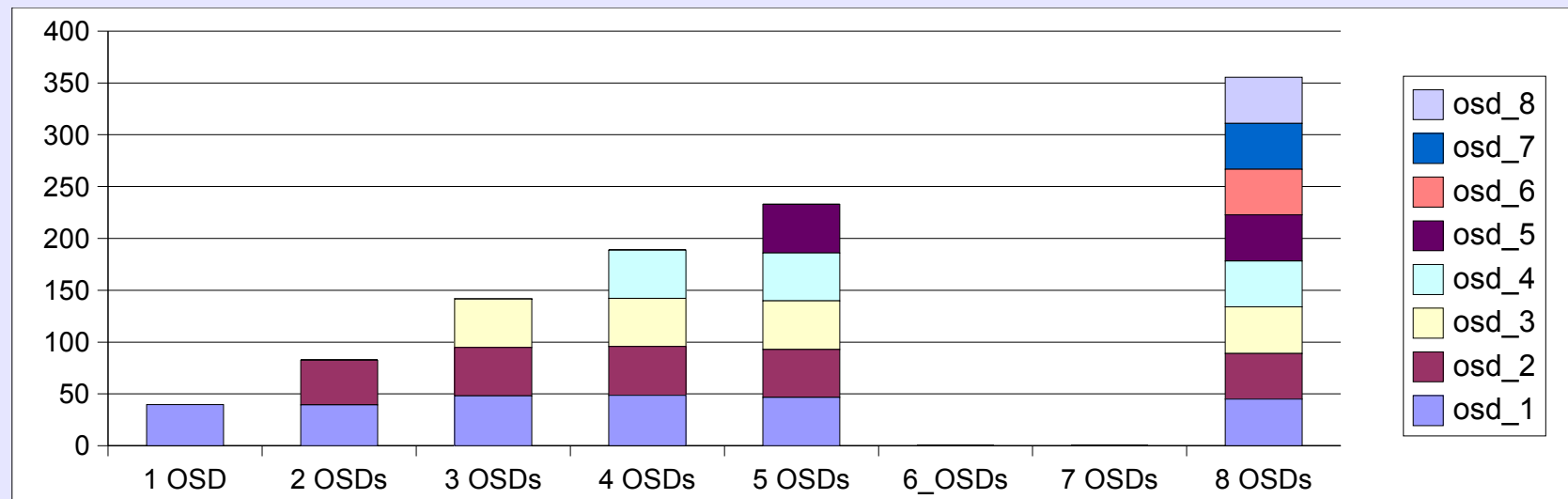
# Progress of the project

- The junior system programmers at CASPUR concentrated on a T10-compliant implementation of the OSD and on the osddb-database
- RZG became aware that this technology could be an alternative to MR-AFS
  - MR-AFS is since more than 10 years maintained at RZG by a single person which will retire in 2.5 years. It isn't open source so no fresh blood will come in.
  - RZG had the man-power to do most of the programming required for the project.
- In July 2006 the project was presented at the AFS-Hackathon in Vienna near Washington DC to the OpenAFS developers.
- In March 2007 a version of AFS+OSD was ready
  - it offers full AFS-semantics including volume replication and legacy interface.
  - still missing: support for different policies, fully functional salvager
- This system underwent a stress test at CERN on
  - 8 servers and 120 clients and different usage patterns.
  - It turned out to be stable and it offered the expected scalability and performance.

## Read/Write 50 clients, variable number of OSDs

- The diagram shows that the total throughput to a single AFS volume scales with the number of OSDs used.
  - the traffic is distributed equally over the OSDs
  - The order of read and write guaranteed that the files to be read could no be in the client's cache.
  - Total network traffic on OSDs were measured in steady-state.
  - Values for 6 and 7 OSDs are missing (NFS problem when writing the log!)
  - Each new OSD contributes ~46 MB/s as long as the metadata server (fileserver) is not the bottle-neck.

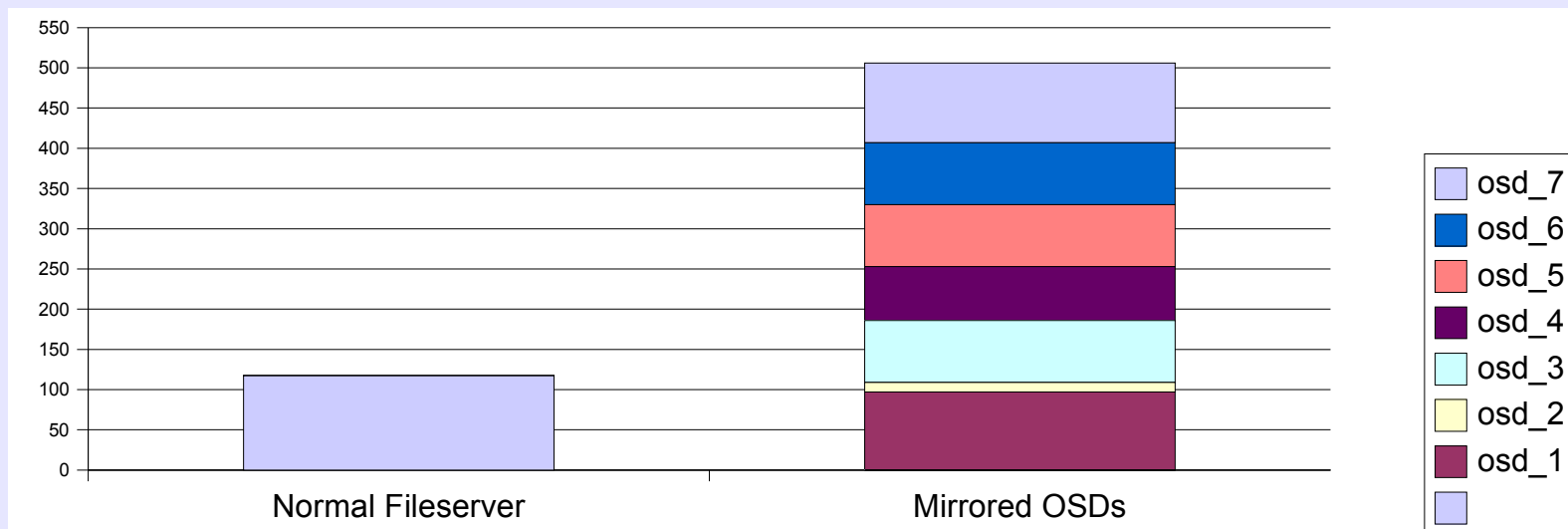
### Total Throughput (read+write) MB/s



## 50 clients reading the same 1.3 GB file

- Normal AFS file (left column)
  - The normal file read can use full bandwidth because disk I/O doesn't play a role.
- Mirrored file in 7 OSDs (right column)
  - Of course, writing the file was slower than writing a simple file.
  - The diagram shows that not all copies are accessed equally (osd\_2 much less)
- Total throughput on clients was 518 MB/s for the mirrored file!

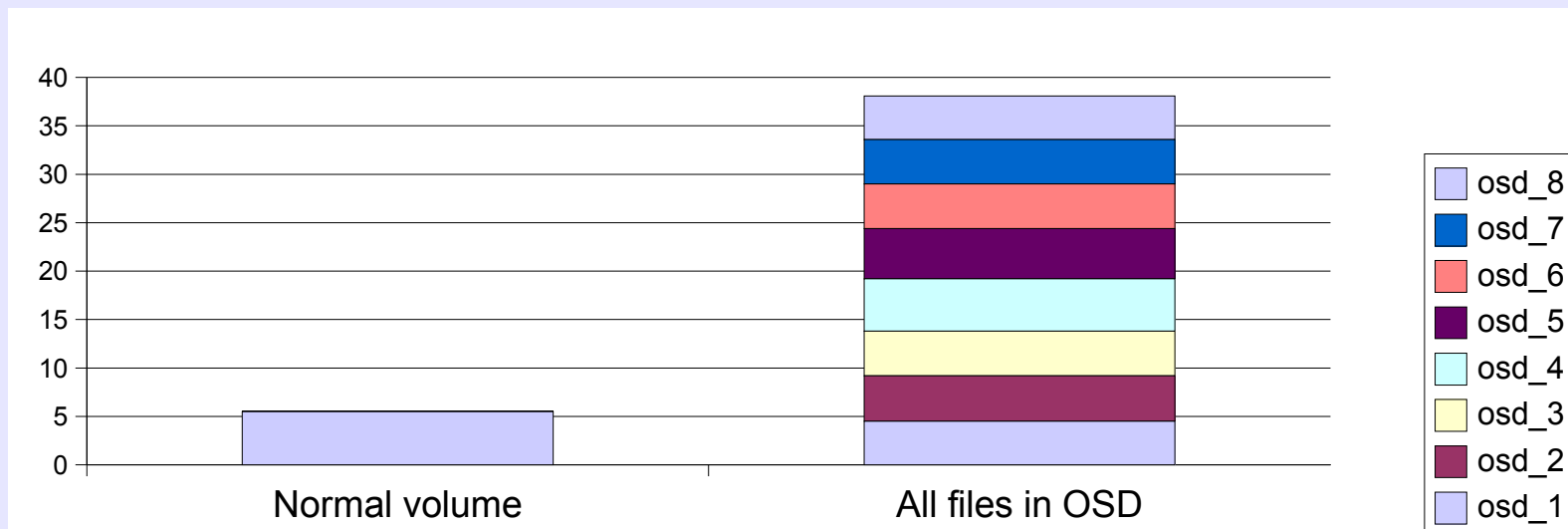
### Total read throughput MB/s



## ~120 clients reading randomly in large RW-volume

- This test simulates the access pattern to the ATLAS LCG software volume at CERN.
  - the 50 GB rw-volume has > 700,000 small files and 115,000 directories
  - nearly all accesses are “stat” or “read” with 10 times more “stat” than “read” .
- The contents of the volume was copied twice into the test cell
  - 1<sup>st</sup> copy normal AFS volume (left column)
  - 2<sup>nd</sup> copy all files distributed to 8 OSDs (right column)
- scripts running in parallel with random access to files and directories 10 times more “stat” than “read”

### Total read throughput MB/s



# Conclusion

- It works, doesn't crash and shows the expected performance.
  - Rxosd support has explicitly to be switched on in the client
  - Files are only stored in OSDs if an osd-flag is set in the volume
  - Therefore this code can be used already in production environments such as RZG.
  - Normal user's volumes remain as they are, but special volumes may use OSDs.
- Next step must be to bring the code into the OpenAFS CVS tree.
  - Once it's there it will take certainly a while until it appears in the official stable releases. (For large file support it took at least 3 years!)
- The project was an R&D project and is now officially over.
  - However, development will continue at least at RZG to achieve full MR-AFS functionality

The current code can be found under

```
/afs/ipp-garching.mpg.de/common/soft/openafs/openafs-1.4.4-osd
```

Disclaimer: This code is certainly full of bugs!

- You have to move volumes to such a server, don't start it on partitions with volumes created with normal OpenAFS (changes in vnode and volume structures)