

# 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

# The Bad News: Our good friend MR-AFS passed away



We were the last site using it. After 14 years on March 8 the last server was shut down



## Now the Good News

- OpenAFS + Object Storage has replaced MR-AFS at RZG
  - It is in full production and offers similar features as MR-AFS did
    - **It brings HSM (Hierarchical Storage Management) to OpenAFS**
  - It makes AFS scale better for high throughput
  - It offers better performance for high speed clients and large files
  - It has some extra goodies to analyze volumes and files and make administration of the cell easier
  - It behaves like „classical“ OpenAFS for volumes without „osd“ flag.

# Overview

- What is Object Storage?
- OpenAFS + Object Storage - a R&D project sponsored by CERN and ENEA
- The implementation
- Results from the March 2007 test campaign at CERN
- The current version OpenAFS-1.4.7-osd
  - New commands and subcommands related to object storage
  - Backup strategies for OpenAFS + Object Storage
  - Hierarchical Storage Management (HSM) for AFS
  - Other goodies
- The future

# 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 on a client to a file in object storage 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 OSD and metadata server. The advantage of this technique is that the OSD doesn't need any knowledge about users and access rights.
- existing object storage systems claim to follow the SCSI T10 standard
  - SCSI because there was hope to integrate the OSD logic into the disk firmware

# Object Storage Systems

- 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)

# 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, uniquifier, and tag (NAMEI inode)
    - 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

# Why AFS and Object Storage can easily go together

- 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 can
  - 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
  - offer HSM functionality in AFS
- 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.



# The R&D Project “OpenAFS + Object Storage”



- In 2004 Rainer Többsicke from CERN implemented a first version of AFS with object storage 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 two system programmers to work at CASPUR
- Most of the development was actually done at RZG because there was the most experience with the AFS source.
- In 2007 the „official“ project ended with a week of tests at CERN.
- Since then RZG has continued to implement all what was needed to replace MR-AFS

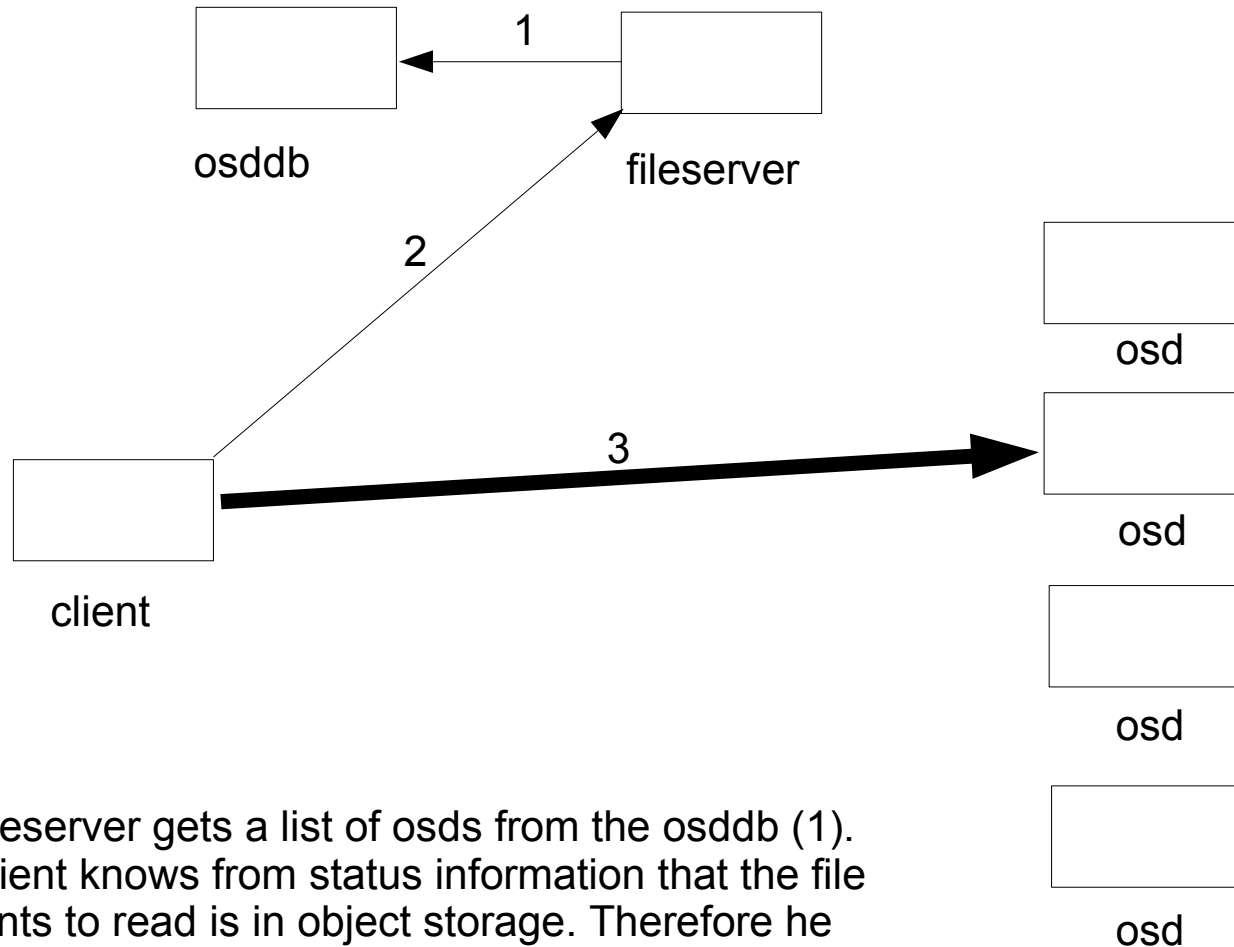
# Implementation: The Components

- „rxosd“ the object storage device (OSD) used for AFS
  - that's where the data are stored. Works on NAMEI partitions.
  - RPC-interface used by client, fileserver, volserver, archival rxosd and osd-command
- „osddb“ a ubik-database (like vldb) describing
  - OSDs
  - policies to be used in different AFS volumes for allocation of files in object storage.
- AFS-fileserver, acting as metadata-server
  - metadata are stored in a new volume-special file
  - extensions also to volserver and salvager
- AFS-client has been restructured to allow for multiple protocols
  - got the ability to store and fetch data directly from OSDs
- Legacy-interface
  - to support old clients
  - to allow move of volumes which use object storage back to “classic” fileservers

## How does it work

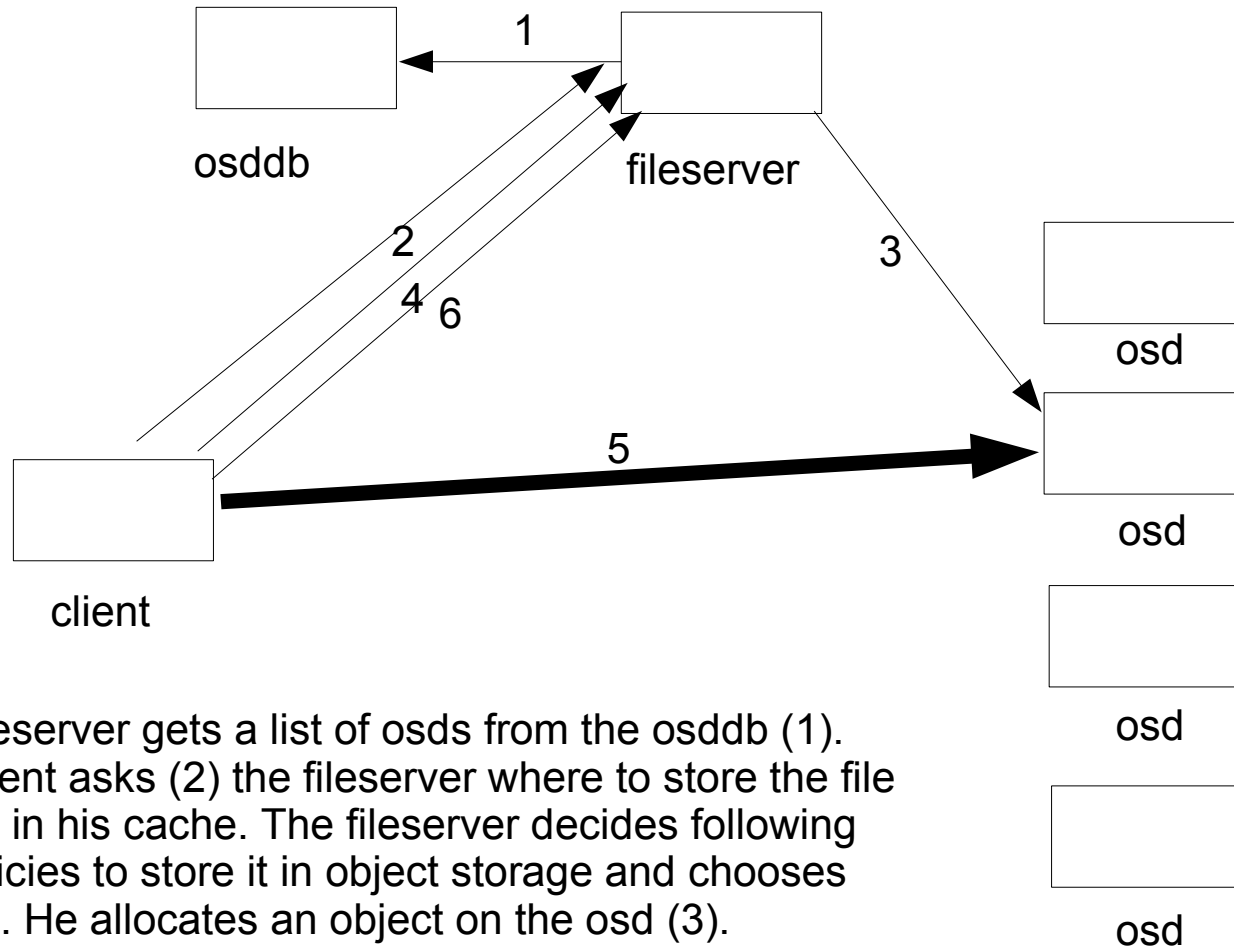
- In struct `AFSFetchStatus` the unused `SyncCounter` is replaced by `Protocol`
  - `Protocol` is set to `RX_OSD` when it is a file in object storage.
  - This information is copied into `vcache->protocol`.
- When such a file is going to be fetched from the server the client
  - does a `RXAFS_GetOSDlocation` RPC to the fileserver
  - With the information returned the client can do a `RXOSD_read` RPC to the OSD.
- Before a new file is stored the first time the client does a `RXAFS_Policy` RPC.
  - If the volume has the `osdflag` set the fileserver tries to find an appropriate OSD
  - Sophisticated policies are not yet implemented. Therefore the fileserver decides only based on the the file size.
  - The fileserver deletes the inode and allocates instead an object on an OSD.
- When `RXAFS_Policy` has returned `protocol == RX_OSD` the client
  - calls `RXAFS_GetOSDlocation` and then use `RXOSD_write` to store the data.

## 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




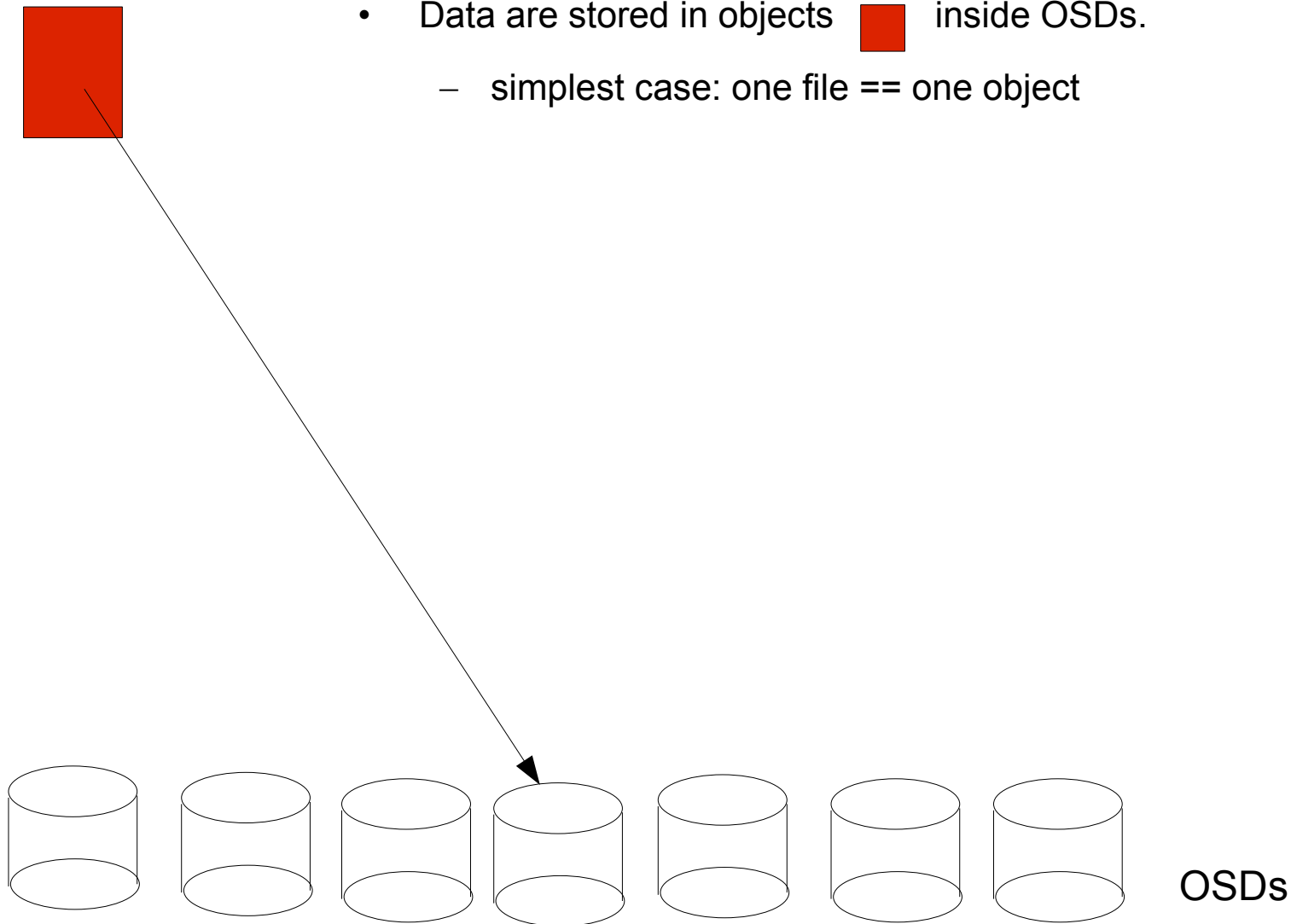
The fileserver gets a list of OSDs from the osddb (1). The client asks (2) the fileserver where to store the file he has in his cache. The fileserver decides following his policies to store it in object storage and chooses an OSD. He allocates an object on the OSD (3). The client asks for permission and location of the object (4) getting an encrypted handle which he uses to store the data in the OSD (5). Finally afs\_StoreMini informs the fileserver of the actual length of the file (6).



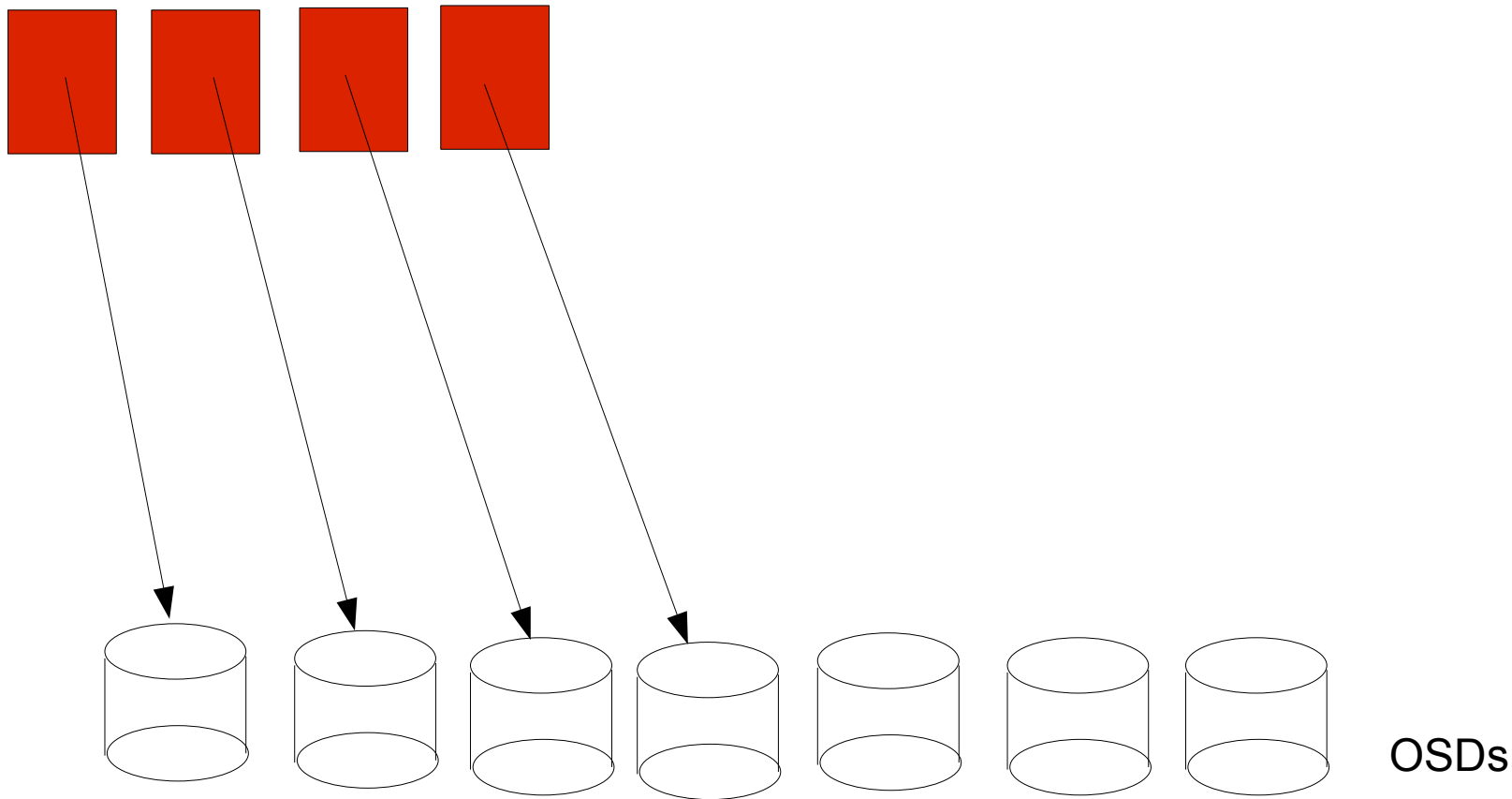
# Describing a file:

## Object

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




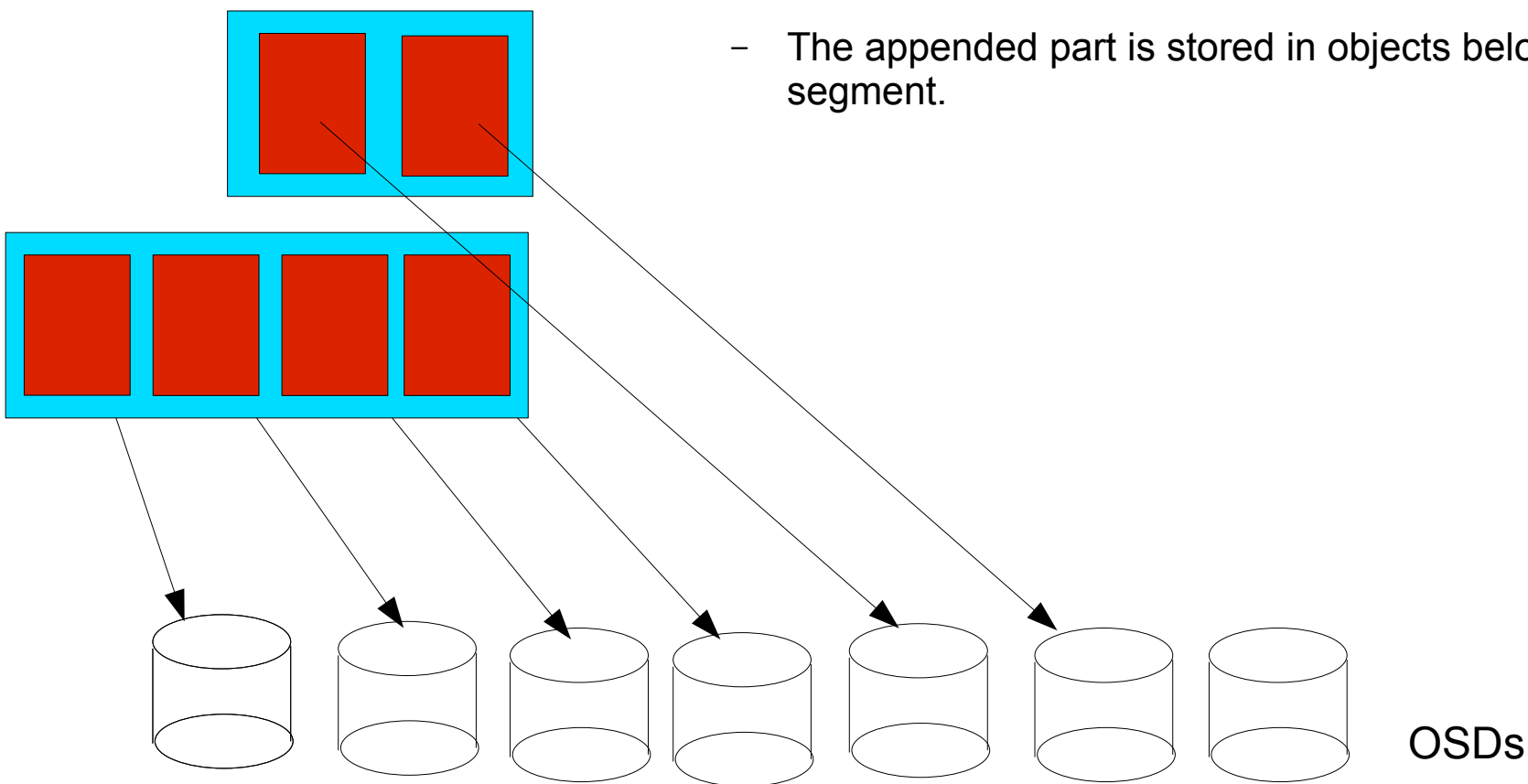
- Data of a file could be stored in multiple objects allowing for
  - data striping (up to 8 stripes, each in a separate OSD)
  - data mirroring (up to 8 copies, each in a separate OSD)




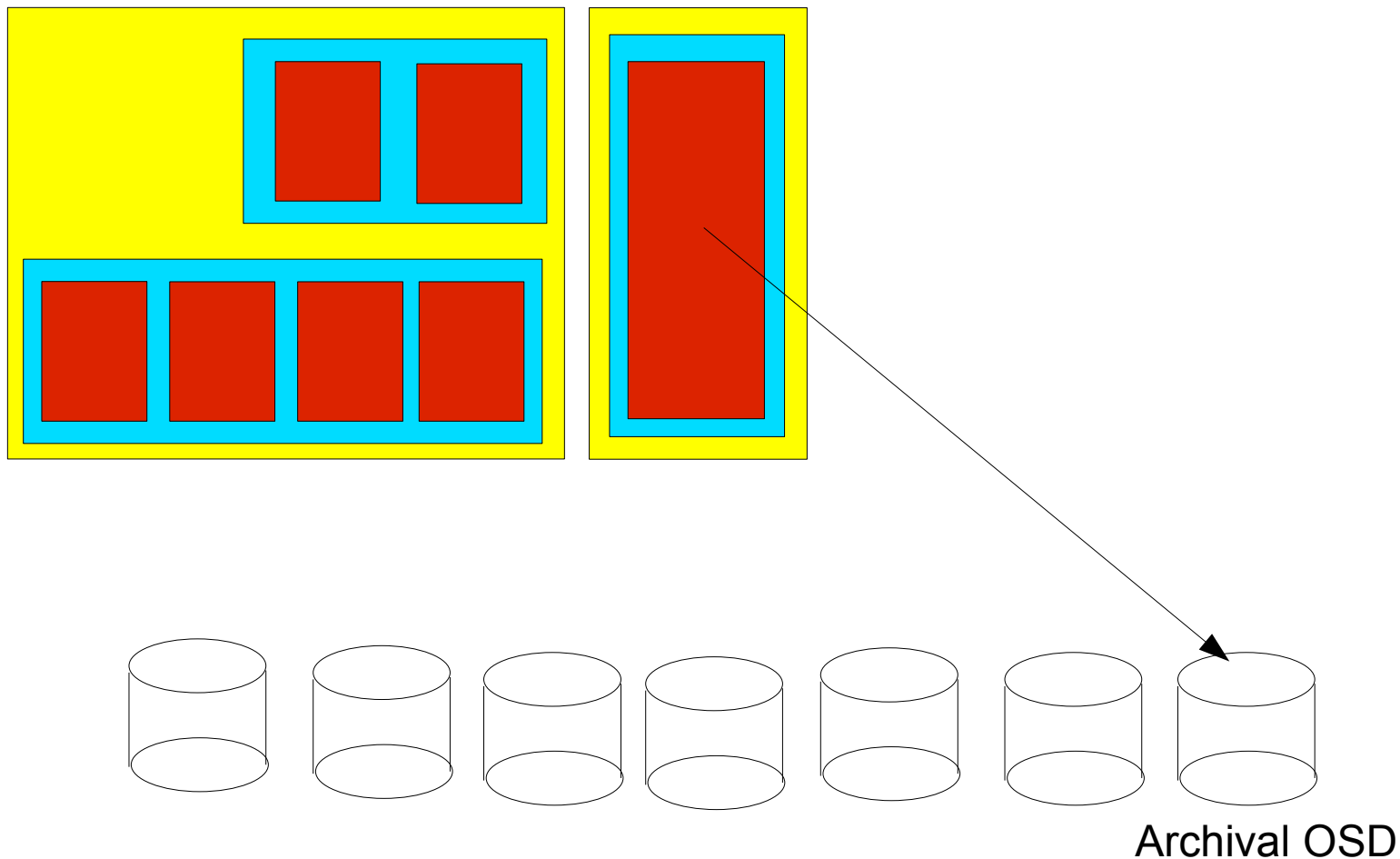
## Describing a file:

## 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.








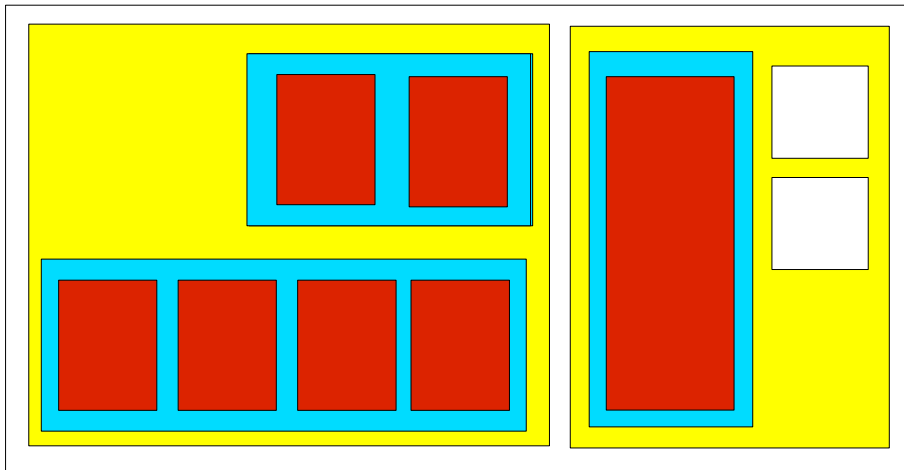
- The whole file could get a copy on an archival OSD (tape, HSM)
  - This leads to the concept of file copies 



# Describing a file:

## The complete 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 a 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 chained
  - The osd metadata are stored in network byte order to allow easy transfer during volume move or replication to other machines.

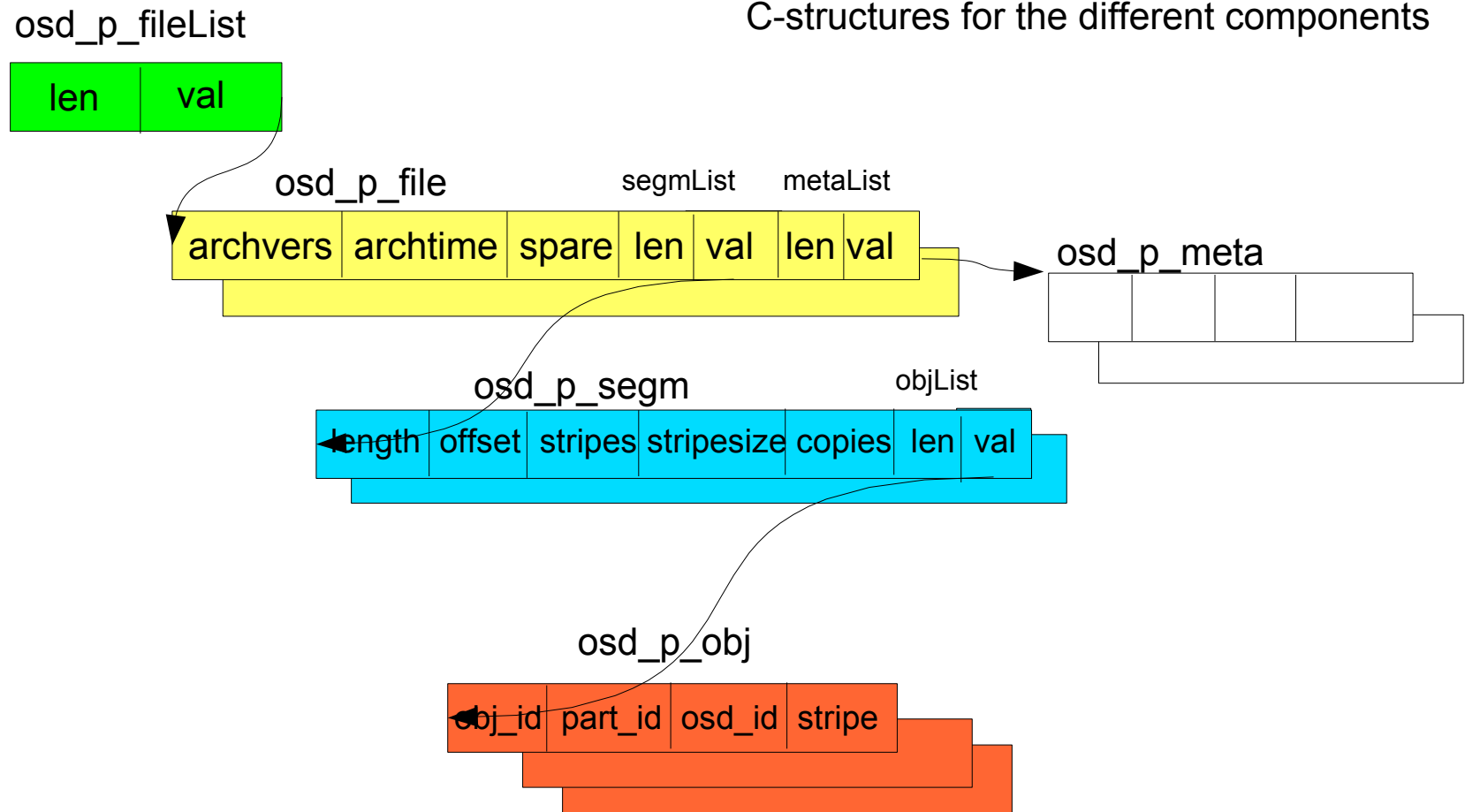




# Describing a file:

## osd metadata in memory

- 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”.

# Example for OSD metadata

- The new „fs“-subcommand „osd“ shows the OSD-metadata of a file.
  - The 1<sup>st</sup> entry describes the actual disk file (its length is taken from the vnode)
  - the 2<sup>nd</sup> and 3<sup>rd</sup> entries describe archival copies with md5-checksums
  - The on-line file has been restored from the second archive on April 1.

```
> fs osd mylargefile.tar
mylargefile.tar has 436 bytes of osd metadata, v=3
On-line, 1 segm, flags=0x0
  segment:
    lng=0, offs=0, stripes=1, strsize=0, cop=1, 1 objects
    object:
      obj=1108560417.96.161157.0, osd=8, stripe=0
Archive, dv=67, 2008-03-19 12:17:23, 1 segm, flags=0x2
  segment:
    lng=2307727360, offs=0, stripes=1, strsize=0, cop=1, 1 objects
    object:
      obj=1108560417.96.161157.0, osd=5, stripe=0
  metadata:
    md5=6e71eff09adc46d74d1ba21e14751624 as from 2008-03-19 13:43:56
Archive, dv=67, 2008-03-19 13:43:56, 1 fetches, last: 2008-04-01, 1 segm, flags=0x2
  segment:
    lng=2307727360, offs=0, stripes=1, strsize=0, cop=1, 1 objects
    object:
      obj=1108560417.96.161157.0, osd=13, stripe=0
  metadata:
    md5=6e71eff09adc46d74d1ba21e14751624 as from 2008-03-24 02:15:27
>
```

# 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 1
id name(loc)      ---total space---      flag  prior. own. server lun size range
  1 local_disk                wr  rd                (0kb-1mb)
  4 raid6           4095 gb   68.4 % up   arch  70  64      afs15.rz  0 (1mb-8mb)
  5 tape            7442 gb   31.3 % up   arch  64  30      styx.rzg  0 (1mb-100gb)
  8 afs16-a         4095 gb   84.8 % up   hsm   80  80      afs16.rz  0 (1mb-100gb)
  9 mpp-fs9-a       11079 gb    4.5 % up   hsm   80  80 mpp mpp-fs9. 0 (1mb-100gb)
 10 afs4-a          4095 gb   85.0 % up   hsm   80  80      afs4.bc.  0 (1mb-100gb)
 11 w7as(hgw)       2721 gb   61.9 % up                65  80      afs-w7as  0 (1mb-9gb)
 12 afs1-a          1869 gb   39.8 % up                70  70      afs1.rzg  0 (1mb-100gb)
 13 hsmgpfs         3139 gb   56.6 % up   arch  50  32      tsm.rzg. 12 (8mb-100gb)
 14 afs6-a          1228 gb   79.1 % up   hsm   72  72 toc afs6.rzg  0 (8mb-100gb)
 17 mpp-fs8-a       2047 gb   15.6 % up   hsm   80  80 mpp mpp-fs8. 0 (1mb-100gb)
 18 mpp-fs3-a       2047 gb   77.6 % up   hsm   80  80 mpp mpp-fs3. 0 (1mb-100gb)
 19 mpp-fs10-a      5552 gb   13.8 % up   hsm   80  80 mpp mpp-fs10 0 (1mb-100gb)
 20 sfsrv45-a        329 gb   10.3 % up                32  64      sfsrv45.  0 (1mb-8mb)
 21 sfsrv45-b        923 gb    6.4 % up                32  64      sfsrv45.  1 (8mb-10gb)
 22 mpp-fs2-a       2047 gb   79.4 % up   hsm   80  80 mpp mpp-fs2. 0 (1mb-100gb)
 23 mpp-fs11-a      6143 gb    0.6 % up   hsm   80  80 mpp mpp-fs11 0 (1mb-100gb)
 24 mpp-fs12-a      6143 gb    0.0 % up   hsm   80  80 mpp mpp-fs12 0 (1mb-100gb)
 25 mpp-fs13-a      6143 gb    0.0 % up   hsm   80  80 mpp mpp-fs13 0 (1mb-100gb)
 26 afs0-a          1862 gb   85.0 % up   hsm   80  80      afs0.rzg  0 (1mb-100gb)
 27 afs11-z         1861 gb   84.8 % up   hsm   80  80      afs11.rz 25 (1mb-100gb)
 32 afs8-z          1023 gb   80.0 % up   hsm   79  80      afs8.rzg 25 (1mb-100gb)
~:
```

# Why use OpenAFS + Object Storage?

## OpenAFS scales very well

- if thousands of clients access files in different volumes on different servers
  - Student's home directories distributed over many servers accessed from slow clients

## OpenAFS doesn't really scale

- when thousands of clients access files in the same RW-volume
  - Huge software repositories in batch environments (CERN)
  - Distributing files over many OSDs can help.
- It doesn't scale when fast clients read large files
  - Striping files over multiple OSDs can help
- It doesn't scale when multiple clients read the same large files
  - Multiple copies of the files on different OSDs can help

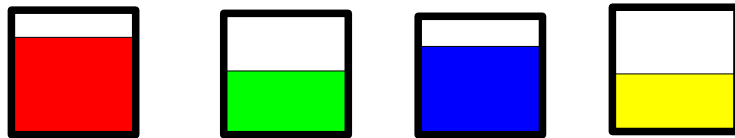
## If you have large infrequently used files HSM functionality would be nice

- Long time preservation of photos, audio- and video- data or raw data from experiments
- Archival OSDs make data migration into underlying HSM system possible.

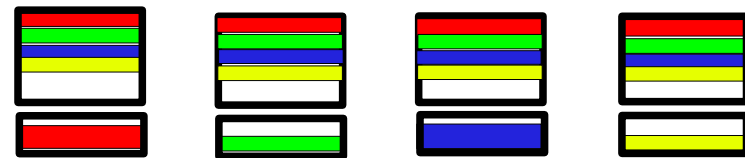
# Simple Example

Huge volumes:

Classical OpenAFS



OpenAFS + Object Storage



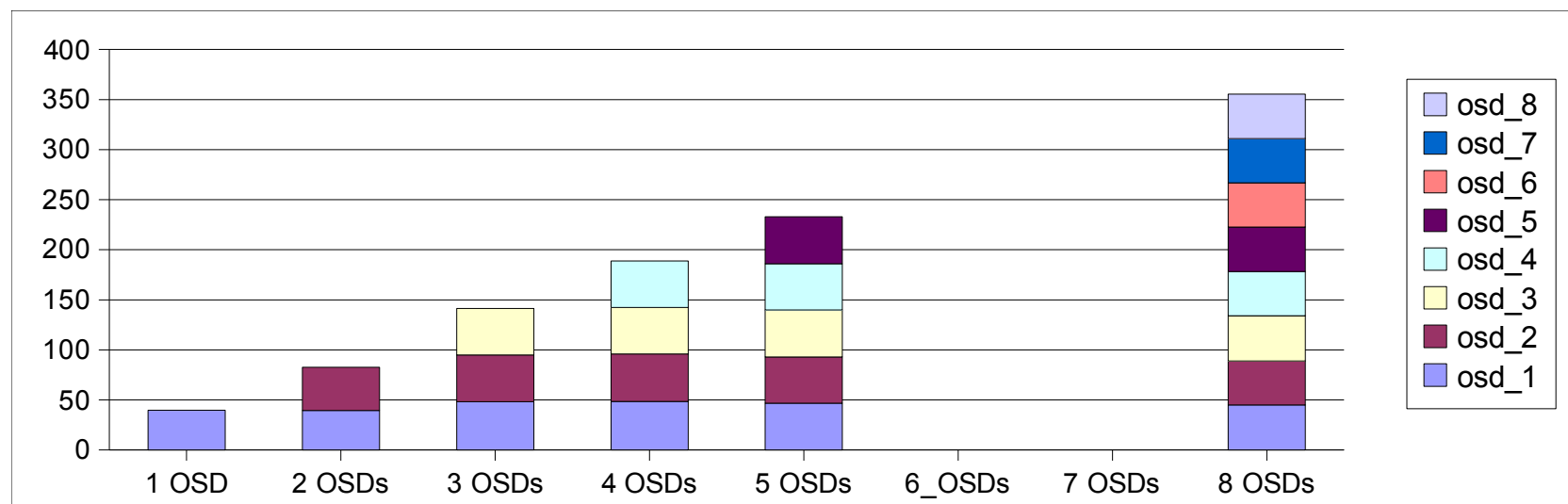
- Same amount of disk space, but split between fileserver partition (lower piece) and object storage (upper piece).
  - Load to servers is balanced even if only red volume is presently used
  - Huge volumes can still be moved or replicated



# CERN Tests: 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 (fileserv) is not the bottle-neck.

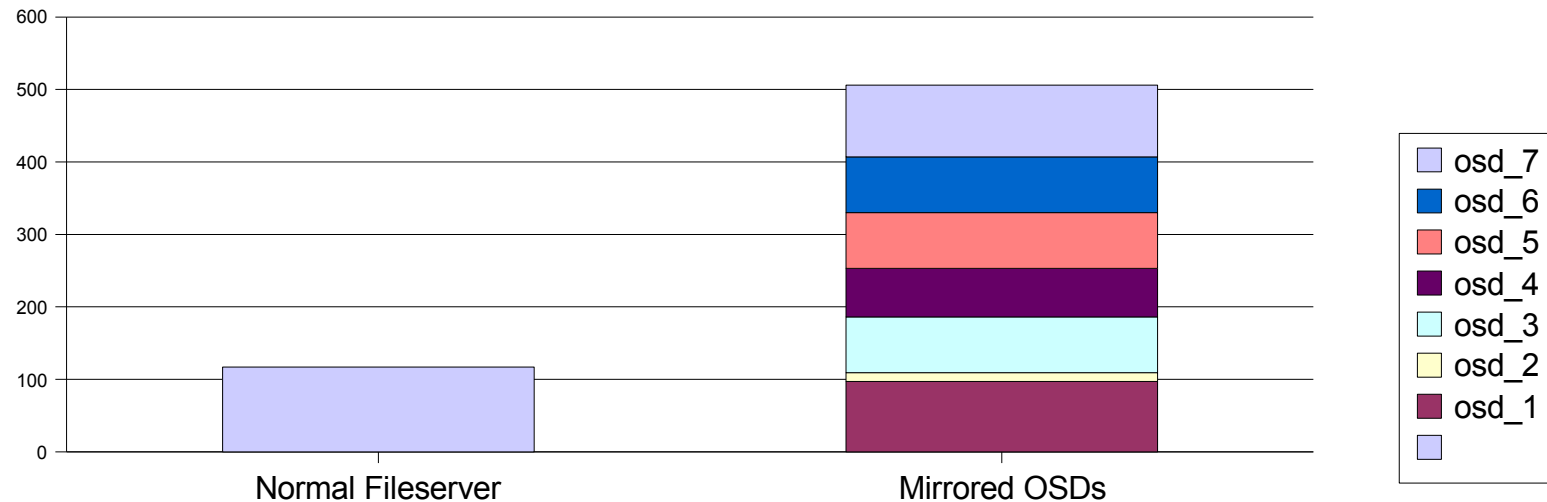
## Total Throughput (read+write) MB/s



# CERN Tests: 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!

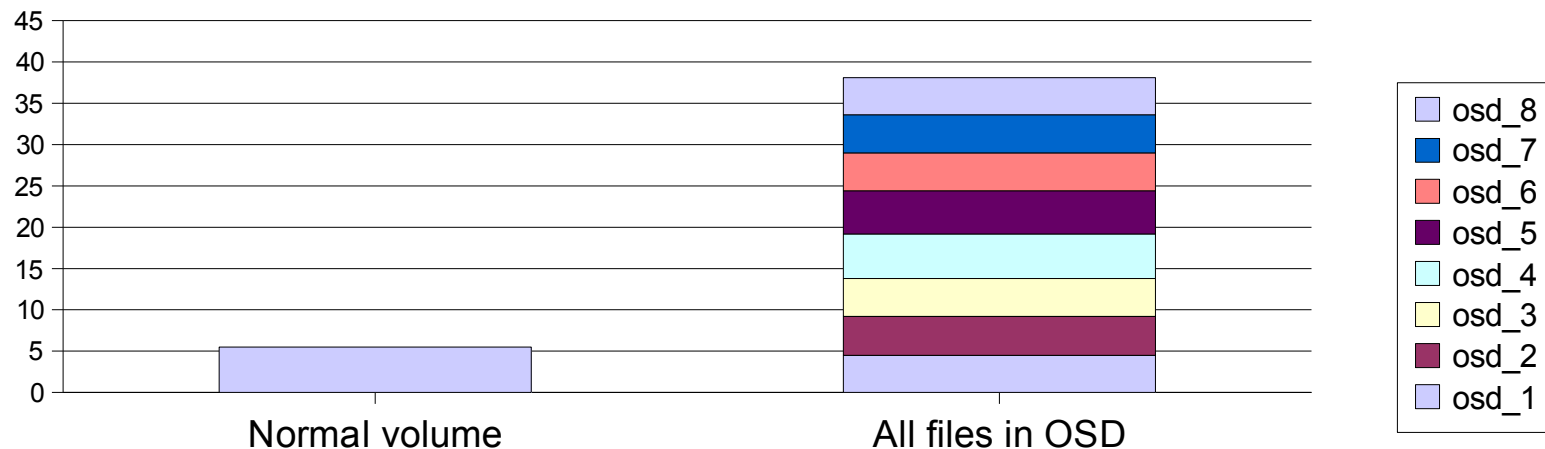
## Total read throughput MB/s



# CERN Tests: ~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 acceses 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



## New command: „osd“

This administrator command is used to manage OSDs and check their content

Some sub-commands:

- list               list OSDs in the database
- add               make new OSD entry in the database
- set               change existing OSD entry in the database
- volumes          list all volumes which have data in an OSD
- objects          list all objects in an OSD belonging to a volume
- examine          details of an object (size, time, link count ...)
- increment       increase link count of an object
- decrement       decrease link count of an object
- wipecandidates   list objects which have not been used for a long time
- md5sum          make md5 checksum of an object and display it
- threads          show currently active RPCs on an OSD

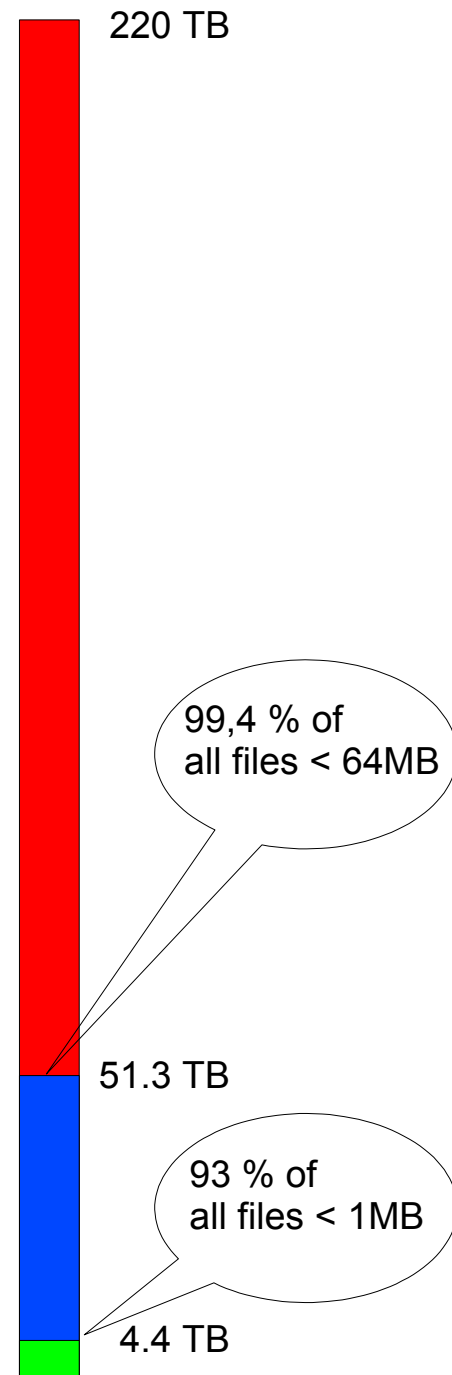
# Additions for „vos“

- small additions
  - “vos dump -metadataonly” dumps only directories and osd metadata. To be used dumptool to identify objects
  - “vos dump -osd” includes data on OSDs in the dump (not useful in HSM environment!)
  - „vos setfields -osdflag“ sets the osdflag (otherwise OSDs will not be used).
- new subcommands
  - archcand lists the oldest OSD-files without archive copy. Used in scripts for automatic archiving.
  - listobjects lists objects on a specified OSD for the whole server or a single volume
  - salvage checks (and repairs) link counts and sizes
  - spiltvolume splits large volume at subdirectory
  - traverse creates file size histogram and more



# „vos traverse“ File Size Histogram of cell ipp-garching.mpg.de

File Size Range				Files	%	run %	Data		%	run %
0 B - 4 KB	51797725	50.70	50.70	64.326 GB	0.03	0.03				
4 KB - 8 KB	8166414	7.99	58.69	44.530 GB	0.02	0.05				
8 KB - 16 KB	7353734	7.20	65.89	78.435 GB	0.03	0.08				
16 KB - 32 KB	7794417	7.63	73.52	163.583 GB	0.07	0.16				
32 KB - 64 KB	6593297	6.45	79.97	303.268 GB	0.13	0.29				
64 KB - 128 KB	4546816	4.45	84.42	403.482 GB	0.18	0.47				
128 KB - 256 KB	3316311	3.25	87.67	587.553 GB	0.26	0.73				
256 KB - 512 KB	3468586	3.39	91.06	1.188 TB	0.54	1.27				
512 KB - 1 MB	2460819	2.41	93.47	1.612 TB	0.73	2.00				
1 MB - 2 MB	1631370	1.60	95.07	2.258 TB	1.03	3.03				
2 MB - 4 MB	1535422	1.50	96.57	3.977 TB	1.81	4.84				
4 MB - 8 MB	1328764	1.30	97.87	6.973 TB	3.17	8.01				
8 MB - 16 MB	737305	0.72	98.59	7.914 TB	3.60	11.60				
16 MB - 32 MB	508383	0.50	99.09	10.500 TB	4.77	16.37				
32 MB - 64 MB	367269	0.36	99.45	15.246 TB	6.93	23.30				
64 MB - 128 MB	241509	0.24	99.68	20.205 TB	9.18	32.48				
128 MB - 256 MB	121515	0.12	99.80	20.996 TB	9.54	42.02				
256 MB - 512 MB	77494	0.08	99.88	28.404 TB	12.91	54.93				
512 MB - 1 GB	112096	0.11	99.99	76.463 TB	34.74	89.67				
1 GB - 2 GB	8735	0.01	100.00	12.161 TB	5.53	95.19				
2 GB - 4 GB	1705	0.00	100.00	4.399 TB	2.00	97.19				
4 GB - 8 GB	374	0.00	100.00	2.297 TB	1.04	98.24				
8 GB - 16 GB	121	0.00	100.00	1.236 TB	0.56	98.80				
16 GB - 32 GB	57	0.00	100.00	1.183 TB	0.54	99.34				
32 GB - 64 GB	27	0.00	100.00	1.159 TB	0.53	99.86				
64 GB - 128 GB	4	0.00	100.00	308.307 GB	0.14	100.00				
-----										
Totals:		102170269 Files			220.094 TB					



# „vos traverse“ Storage Usage

Storage usage:

		1	local_disk	99412884	files	44.564	TB
arch.	Osd	4	raid6	960456	objects	2.735	TB
arch.	Osd	5	tape	2753493	objects	175.209	TB
	Osd	8	afs-16-a	26426	objects	3.388	TB
	Osd	9	mpp-fs9-a	1192	objects	299.083	GB
	Osd	10	afs4-a	28917	objects	3.375	TB
	Osd	11	w7as	96129	objects	1.610	TB
	Osd	12	afs1-a	236967	objects	793.564	GB
arch.	Osd	13	hsmgpfs	173195	objects	25.326	TB
	Osd	14	afs6-a	5686	objects	922.579	GB
	Osd	17	mpp-fs8-a	421	objects	125.001	GB
	Osd	18	mpp-fs3-a	13766	objects	1.138	TB
	Osd	19	mpp-fs10-a	1483	objects	297.979	GB
	Osd	20	sfsrv45-a	4141	objects	11.650	GB
	Osd	21	sfsrv45-b	154	objects	47.705	GB
	Osd	22	mpp-fs2-a	4600	objects	1.566	TB
	Osd	23	mpp-fs11-a	518	objects	58.605	GB
	Osd	24	mpp-fs12-a	292	objects	32.958	GB
	Osd	25	mpp-fs13-a	107	objects	7.925	GB
	Osd	26	afs0-a	12090	objects	1.543	TB
	Osd	27	afs11-z	8968	objects	1.495	TB
	Osd	32	afs8-z	9743	objects	793.148	GB
Total				103751628	objects	265.270	TB

# Additions for „fs“

Some new subcommands (mostly for administrators)

- ls                    like “ls -l”, but differentiates between files, objects, and wiped objects
- vnode                shows the contents of a file's vnode.
- fidvnode            as above, but on “fid”. Shows also the relative AFS path inside volume.
- [fid]osd            shows OSD metadata of an OSD-file
- [fid]archive        generates a copy on an archival OSD. The md5 checksum is saved in the osd metadata
- [fid]replaceosd    moves an object from one OSD to another (or from local disk to OSD, or from an OSD to local disk, or removes a copy on an OSD)
- [fid]wipe            erases the on-line copy if file has good archive copy.
- translate           translates NAMEI-path to fid and shows relative AFS path inside the volume
- threads            shows active RPCs on the fileserver
- protocol            turns OSD usage on/off in the client
- listlocked          shows locked vnodes on the fileserver
- createstripedfile   preallocates striped and/or mirrored OSD file

# Backup Strategies

- Classical Backup
  - Volumes using object storage can be backed up by „vos dump“
  - The dumps do not contain the object's data, only their metadata.
- Backup by volume replication to another server for later use with „vos convertROtoRW“
  - The RO-volumes are just mirrors of the RW-volumes and don't contain the object's data
- Both techniques do not protect data in OSDs. Therefore files in object storage need to be backed up separately:
  - Have „archival“ OSDs (either on disk or in HSM systems with tape robots)
  - Run an “archive” script as instance on the database servers which calls
    - „vos archcand ...“ to extracts list of candidates from the file servers
    - „fs fidarchive ...“ to create the archival copy of the files on an archival OSD
  - metadata of the archival copies are stored in the volume along with md5-checksums

# HSM functionality for AFS

- Now that we have „archival“ copies - why not use them to obtain HSM-functionality?
- We run a “wiper” script as instance on the database servers which
  - checks disk usage with “osd list -wipeable” and when a high-water-mark is reached
  - gets candidates with “osd wipecandidates ...”
  - and wipes (frees) on-line version of the file in the RW-volume with „fs fidwipe ...“ .
  - and at the end also in the RO-volumes by „vos release“
- About 10,000 of our 25,000 volumes use object storage
  - To these volumes belong 175 TB == 80 % of the 220 TB total data
    - Of these 175 TB only 16 TB == less than 10 % are on-line.
- Wiped files come automatically back to disk when they are accessed
- Wiped files can be prefetched to disk by „fs prefetch ...“ to prepare for batch jobs
- The queuing of fetch requests uses an intelligent scheduler which
  - Allows users to submit many requests at a time
  - But lets compete always the 1<sup>st</sup> entries of all users.
    - So a single request of one user can bypass a long queue of another user

## Other Goodies: „vos split“

- Splits a volume at a subdirectory without moving any data
  - Just volume special files (volinfo, small and large vnode-files, osd-metadata) are copied pointing to the same real inodes/objects.
  - Real objects get hard links into the new volume's NAMEI-tree
  - In the new volume all references to the part not covered by it are removed
  - And in the old volume all references to files under the subdirectory
  - The subdirectory is replaced by a mount point to the new volume.
- Very useful for especially for volumes containing files in HSM !
- Syntax:  

```
vos splitvolume -id <volume name or ID>  
    -newname <name of the new volume>  
    -dirvnode <vnode number of directory where the volume should be split>  
    [-cell <cell name>]
```
- The vnode number of the subdirectory can be obtained by „fs getfid“

## More Goodies: „afsio“

This command can be used to write and read files bypassing the cache manager

Subcommands:

- read        reads AFS-file (normal or OSD) and writes result to stdout
- write       reads from stdin and writes AFS-file (normal or OSD)
- append      as write, but as name says, append
- fidread     as read, but with „fid“ instead of path
- fidwrite    ...
- Fidappend   ...

Note: you still need an AFS-client to locate the file and the fileserver.

“afsio” is heavily used at the AUG experiment at IPP Garching to copy large files from the data acquisition systems into AFS



# When should you use OpenAFS + Object Storage?

- Without HSM system in the background
  - you can get better throughput for data in volumes which cannot be replicated
    - Object storage distributes the data equally over many OSDs
    - Striping distributes load even for single large files
    - Mirroring useful for data that are much more often read then written
- With a HSM system
  - You can get HSM functionality for AFS
    - Infrequently used data go on secondary storage
  - Archival copies of files in object storage as backup solution along with RO-volumes for small files and directories

## Future Development



Felix Frank at DESY Zeuthen in Germany is working on the policies.

- The policies will control
  - which files go into object storage (based on name and size)
  - how they should be striped or mirrored and into which OSDs they should go
- The policies will be stored in the osddb and the directory vnodes will contain the policy number to apply for files in this directory.
- Other work will continue to be done at RZG.

# Testers are invited

- **Next step must be to bring the code into the OpenAFS CVS tree.**
  - This had been promised already a year ago! (I understand: it is a lot of work)
  - Once it's there it will take certainly a while until it appears in the official stable releases. (For large file support it took 3 years!)

The current code can be found under

```
/afs/ipp-garching.mpg.de/common/soft/openafs/openafs-1.4.7-osd
```

To build it with object storage you need to configure it at least with

```
configure --enable-object-storage --enable-namei-fileserver
```

Without „--enable-object-storage“ you should get a „classical“ OpenAFS with some of the goodies mentioned before.

Disclaimer: This code still may have some bugs!

- If configured with “--enable-object-storage” you have to move volumes to such a server, don't start it on partitions with volumes created with “classical” OpenAFS (changes in vnode and volume structures)



# Last Question

Someone here who wants to replace me when I retire next year?

- Interesting work (cell administration and AFS development)
- RZG is one of the leading supercomputer centers in Europe
- Garching near Munich is situated in the nice Bavarian landscape
  - remember the famous Biergärten and Oktoberfest
  - and the Alps and lakes near by

