Integration of AFS/OSD into OpenAFS

Hartmut Reuter
reuter@rzg.mpg.de

Agenda

- What is AFS/OSD?
- Current usage of AFS/OSD
  - kind of site report for our cell
- What is new since last year?
- Integration of AFS/OSD into OpenAFS

What is AFS/OSD?

In few words because I talked about this already on many AFS workshops:

- AFS/OSD is an extension to OpenAFS which

    1. allows to store files in (object storage) instead of the fileserver's partition. The object storage consists in many disk servers running „rxosd“.
    2. brings HSM functionality to AFS if an archival “rxosd” uses an underlying HSM system. This feature offers „infinite“ disk space.
    3. gives fast access to AFS files in clusters with “embedded filesystems“. A shared filesystem such as GPFS or Lustre is used by an „rxosd“ and the clients in the cluster can access the data directly.

A talk describing AFS/OSD was given in Newark and Graz 2008:
http://workshop.openafs.org/afsbpw08/talks/thu_3/Openafs+ObjectStorage.pdf

A talk describing “Embedded Filesystems” was given in Stanford 2009:
http://workshop.openafs.org/afsbpw09/talks/thu_2/Embedded_filesystems_opt.pdf

A tutorial about AFS/OSD was given in Rome 2009:
http://www.dia.uniroma3.it/~afscon09/docs/reuter.pdf

Usage of AFS/OSD

- So, why could sites want to deploy AFS/OSD?
    1. Better distribution of data on disk storage (the original idea of Rainer Toebbicke from CERN)
    2. To have HSM for AFS (data-migration onto tapes)
    3. Fast access to AFS files in clusters with “embedded filesystems“
- DESY Zeuthen was 2009 interested in case 3 with Lustre
    - With the uncertain future of Lustre they stopped this project
- ENEA made 2010 some tests to 3 with GPFS as “embedded filesystem“
- PSI (Paul-Scherrer-Institut) made 2011 tests to 2 and 3
    - with SamFS as HSM system and GPFS as “embedded filesystem“
    - Since this year PSI is running in production with use case 2.
- RZG is running AFS/OSD in production with use cases 1 and 2 since 2007
    - Migrating from TSM-HSM to HPSS as HSM system

20 years AFS cell „ipp-garching.mpg.de“

Twenty years AFS: I created the cell ipp-garching.mpg.de in October 1992

40 fileservers in 2 sites with 356 TB disk space

24 non-archival OSDs with 111 TB disk space (for 735 TB of data)

2 archival OSD one with TSM-HSM, the other with HPSS.

(TSM-HSM will completely be replaced by HPSS until mid 2013)

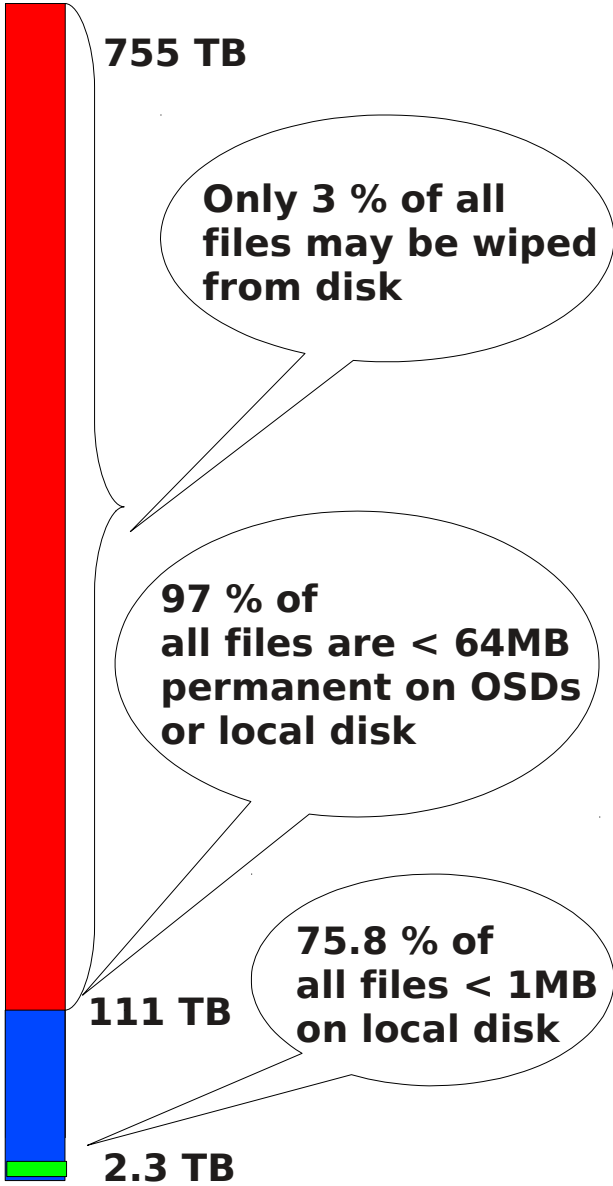
33700 volumes

10500 users

		normal (non-OSD)	OSD-volumes
204	million files	165 million	38 million
830	TB total data	80 TB	755 TB

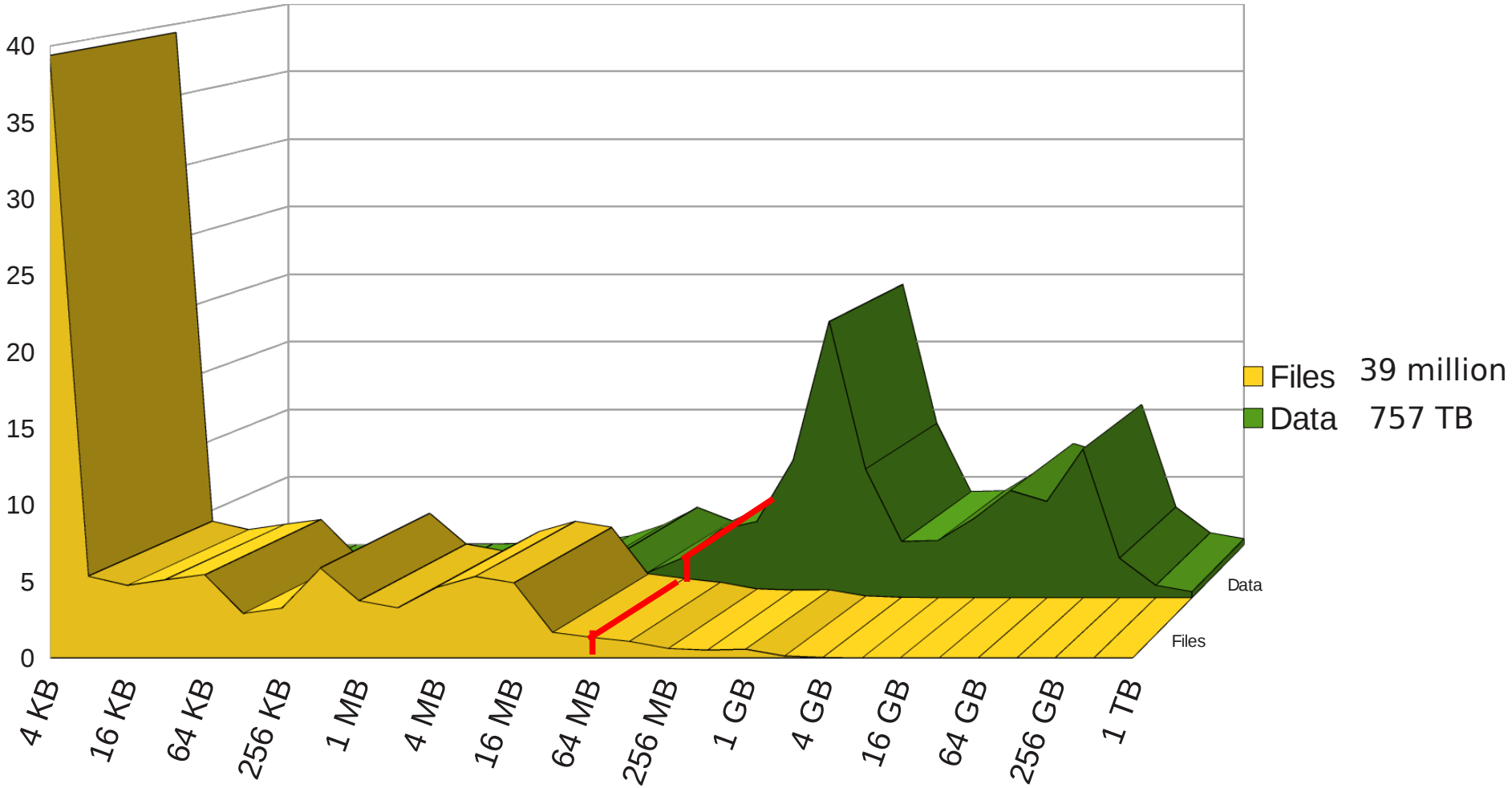
only OSD-Volumes

File Size Range	Files	%	run %	Data	%	run %
0 B - 4 KB	15083406	39.17	39.17	15.961 GB	0.00	0.00
4 KB - 8 KB	2066424	5.37	44.53	11.012 GB	0.00	0.00
8 KB - 16 KB	1839399	4.78	49.31	20.884 GB	0.00	0.01
16 KB - 32 KB	1980631	5.14	54.45	43.260 GB	0.01	0.01
32 KB - 64 KB	2109174	5.48	59.93	90.309 GB	0.01	0.02
64 KB - 128 KB	1127862	2.93	62.86	97.089 GB	0.01	0.04
128 KB - 256 KB	1261604	3.28	66.13	227.722 GB	0.03	0.07
256 KB - 512 KB	2268455	5.89	72.02	784.150 GB	0.10	0.17
512 KB - 1 MB	1447920	3.76	75.78	1001.438 GB	0.13	0.30
1 MB - 2 MB	1266450	3.29	79.07	1.826 TB	0.24	0.54
2 MB - 4 MB	1785078	4.64	83.71	4.864 TB	0.64	1.18
4 MB - 8 MB	2062908	5.36	89.06	11.147 TB	1.48	2.66
8 MB - 16 MB	1895531	4.92	93.98	20.739 TB	2.75	5.41
16 MB - 32 MB	651726	1.69	95.68	13.344 TB	1.77	7.17
32 MB - 64 MB	518999	1.35	97.02	21.298 TB	2.82	10.00
64 MB - 128 MB	412772	1.07	98.10	35.555 TB	4.71	14.71
128 MB - 256 MB	239076	0.62	98.72	40.010 TB	5.30	20.01
256 MB - 512 MB	199921	0.52	99.24	72.338 TB	9.58	29.59
512 MB - 1 GB	219228	0.57	99.81	145.502 TB	19.27	48.86
1 GB - 2 GB	49795	0.13	99.93	67.723 TB	8.97	57.84
2 GB - 4 GB	11026	0.03	99.96	29.730 TB	3.94	61.77
4 GB - 8 GB	5350	0.01	99.98	30.207 TB	4.00	65.78
8 GB - 16 GB	4068	0.01	99.99	41.293 TB	5.47	71.25
16 GB - 32 GB	2601	0.01	99.99	56.563 TB	7.49	78.74
32 GB - 64 GB	1145	0.00	100.00	50.921 TB	6.75	85.48
64 GB - 128 GB	909	0.00	100.00	78.656 TB	10.42	95.90
128 GB - 256 GB	129	0.00	100.00	21.004 TB	2.78	98.69
256 GB - 512 GB	18	0.00	100.00	6.641 TB	0.88	99.57
512 GB - 1 TB	5	0.00	100.00	3.272 TB	0.43	100.00
Totals:	38511610 Files			754.892 TB		



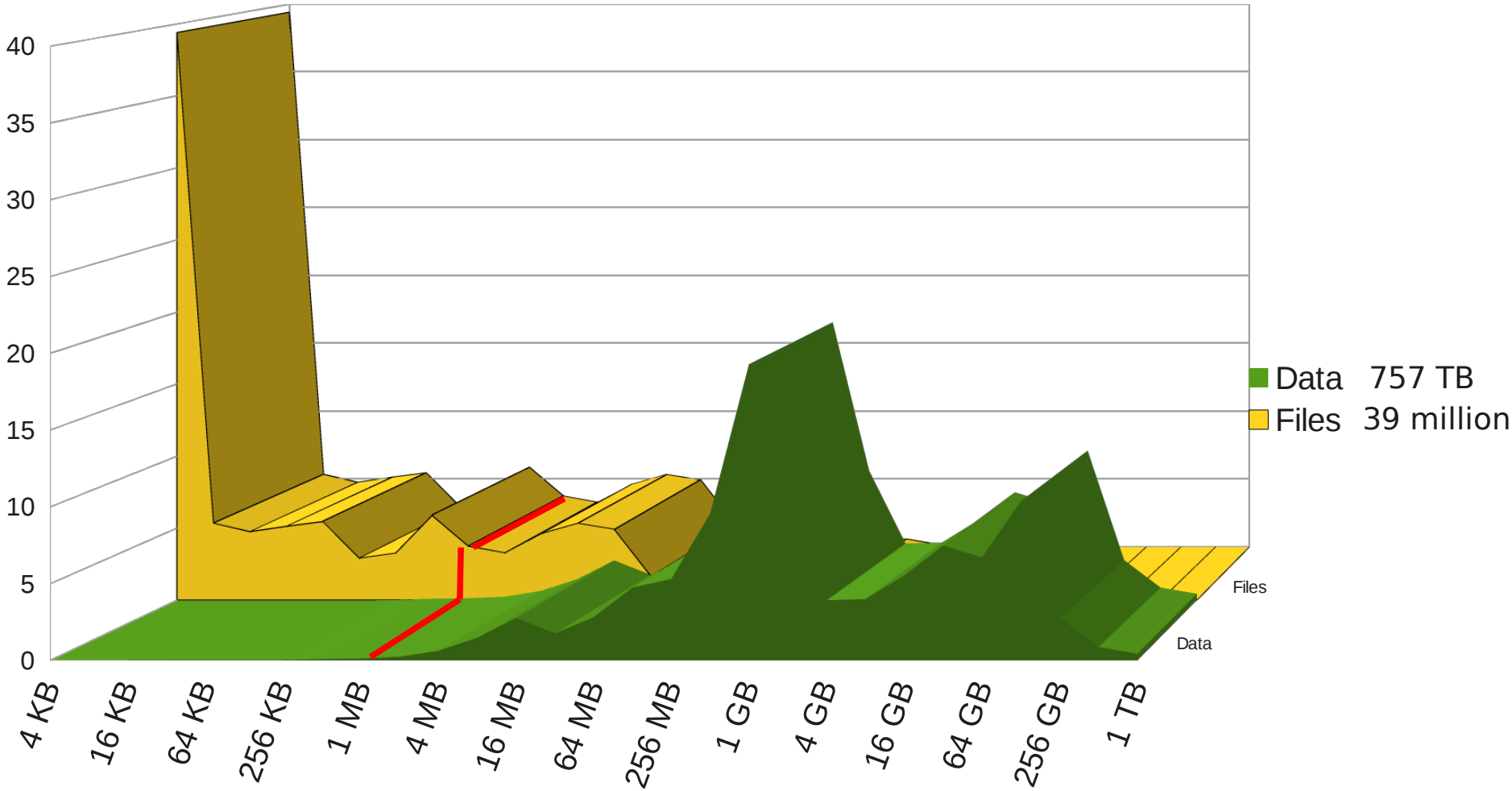
only OSD volumes

- The diagram shows number of files and amount of data over the logarithm of the file size.
- All data right of the red line at 64 MB can be wiped from disk kept only on tape
  - This are only **3 %** of the files, but **90 %** of the total data volume !



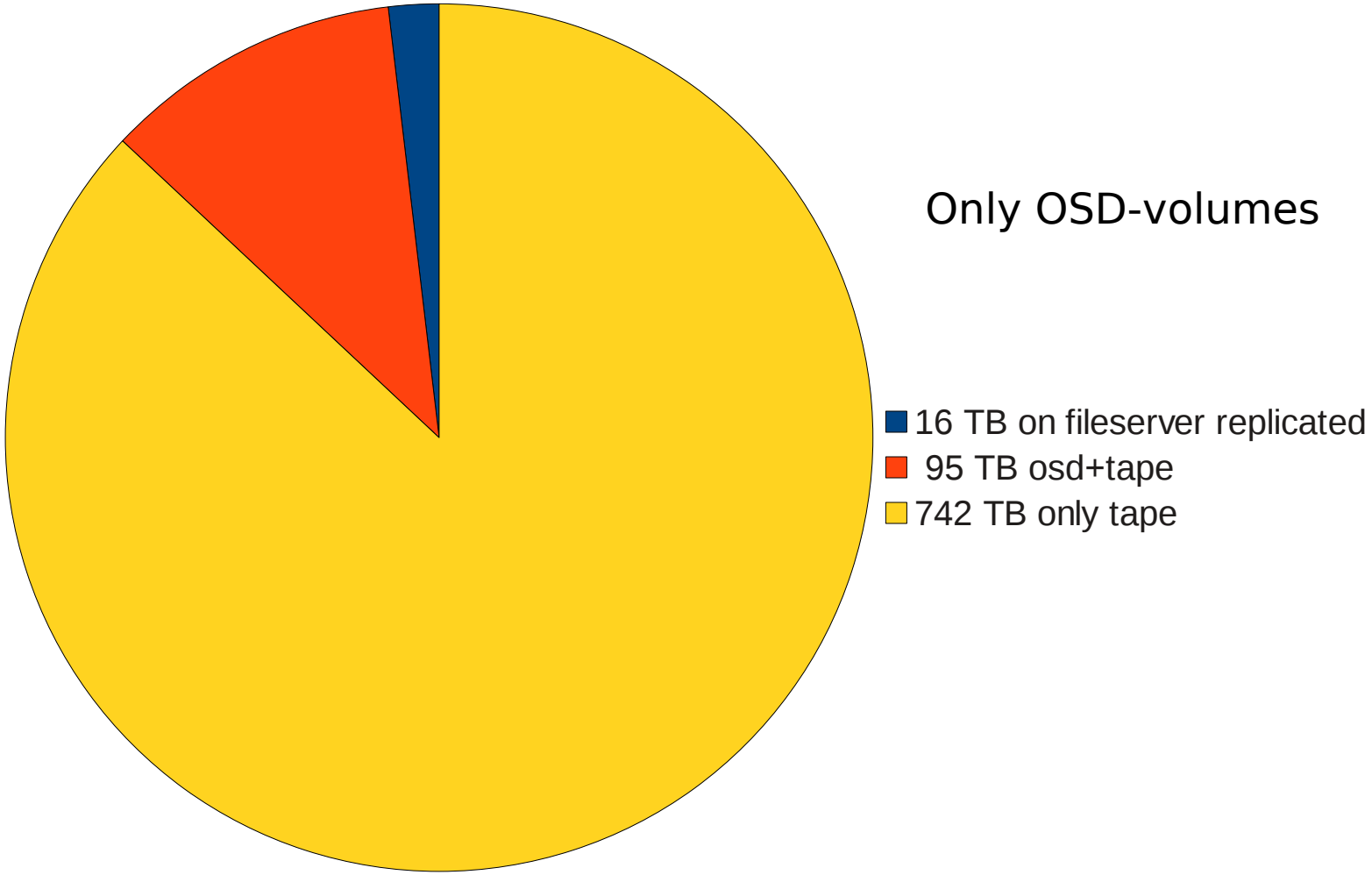
only OSD volumes

- The diagram shows number of files and amount of data over the logarithm of the file size.
- All data left of the red line at 1 MB are on the fileserver's partition
  - This are only 76 **%** of the files, but only 0.3 **%** of the total data volume !

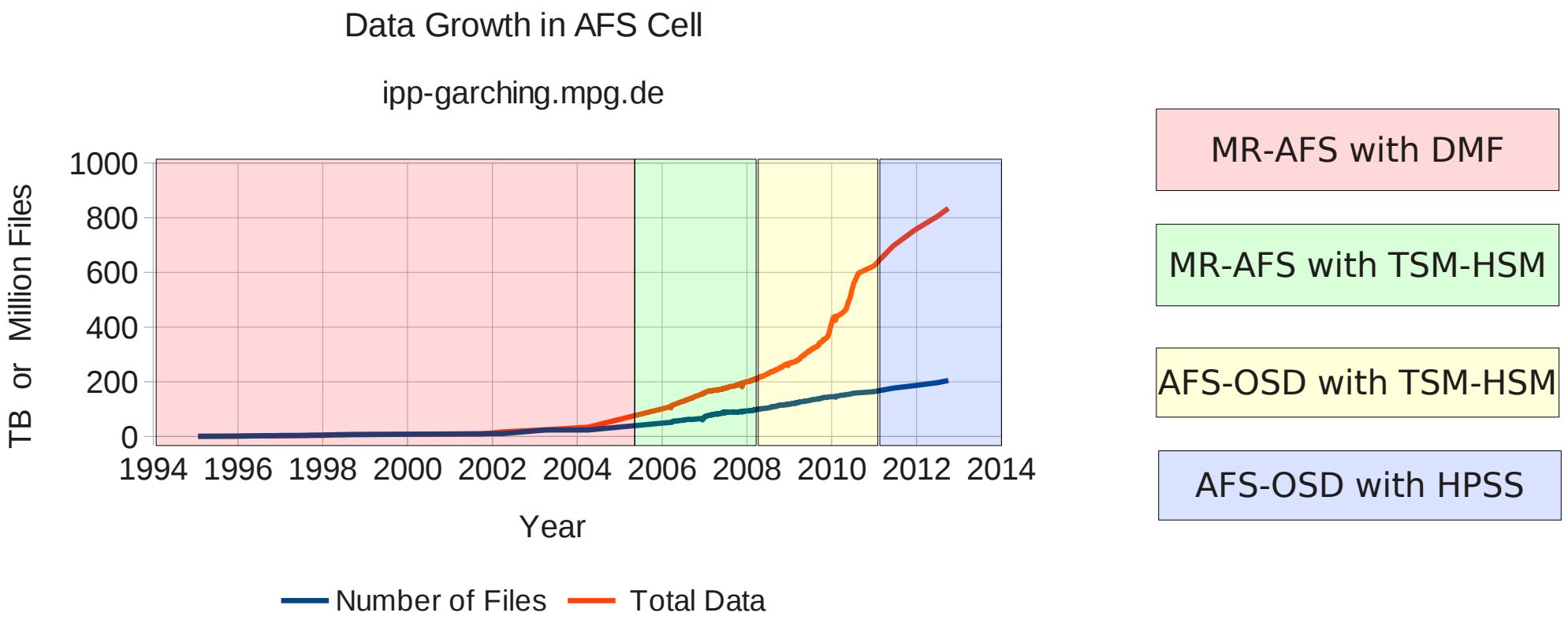


Where the data are

- All data in the local partitions of the fileservers are replicated to other fileservers
- All data in disk OSDs have at least 2 tape copies in archival OSDs



Our Data Growth



- Transparent to the user and the AFS-tree we had different HSM systems and AFS versions
- Data growth brought us to the limit of what TSM-HSM could ingest
- HPSS claims to scale much better because one can add more data movers

AFS/OSD Versions

- **1.4-osd** no big changes in this version of AFS/OSD during the last year
  - In sync with openafs 1.4 (presently 1.4.14). Some bug-fixes,
  - svn checkout http://svnsrv.desy.de/public/openafs-osd/trunk/openafs/.
- **1.6-osd** since meeting in Hamburg last year in **github**
  - in sync with the openafs 1.6 tree (presently 1.6.1a).
  - fully backward compatible to the 1.4-osd RPCs.
  - in production on few fileservers and all but one rxosds in our cell
  - git clone git://github.com/hwr/openafs-osd.git
- **git master** The version should go into **git master** of openafs
  - much simpler: without backward compatibility to 1.4-osd.
  - nearly no changes to current OpenAFS RPC interfaces
  - interfaces and services for all AFS/OSD stuff in a separate library
  - no extras such as „fs threads“, „fs stat“, „fs setvariable“ ...

Separation of code

- Last year in Hamburg Jeffrey Altman and Derrick Brashear proposed to
  - separate all AFS/OSD code into a shared library „libafsosd“
  - have in the main code only hooks for the shared library
  - have separate RPC interfaces for the AFS/OSD calls
  - integrate the OSDDB into the vlserver
  - also separate the cache manager AFS/OSD code to be loaded only with a “-libafsosd” parameter for afsd
- In the **1.6-osd** tree the code separation has now been done for the server side
  - Not much nicer because our 1.6-osd servers need to be backward compatible to 1.4-osd clients so all our RXAFS-RPCs still must be supported and sent into libafsosd for processing.
  - 1.6-osd clients pay with doubled number of connections because 2 different services are being accessed in the fileserver

How it works

- A source file libafsosd.c is compiled in both: the binary and the shared library
- In the binary an entry load_libafsosd(...) is called to
  - load the shared library
  - call the appropriate initialization routine
- In the shared library libafsosd_init(...) is called from the initialization routines inside the shared library
  - it checks the shared library's subversion number
  - provides operation vectors and pointers to data
- After initialization has completed there exist data pointer and operation vectors for both directions
  - calls from binary to shared library
  - calls from shared library to routines in the binary
- In the binary a new RPC service is started with the ExecuteRequest entry provided by the shared library.

Separation of code

in afsosd.h:

```
struct osd_viced_ops_v0 {
...
   void (*op_remove_if_osd_file) (Vnode **targetptr);
...
}
```

Example for a hook into libafsosd.so in afsfileprocs.c:

```
...
#include "../shlibafsosd/afsosd.h"

struct osd_viced_ops_v0 *osdviced = NULL;
struct osd_vol_ops_v0 *osdvol = NULL;
...
   if ((*targetptr)->delete) {
        if (osdviced)
            (osdviced->op_remove_if_osd_file) (targetptr);
        if (VN_GET_INO(*targetptr)) {
```

Separation of code

- In the clone of the **git master** the separation is now complete
  - All code going into libafsosd.so is stored in the already existing directory src/rxosd
  - 3 shared libraries:
    - libafsosd.so.1.y for instance fs
    - libdafsosd.so.1.y for instance dafs
    - libcafsosd.so.1.y for client commands
    - y the actual version, presently 10
  - Client commands (fs and vos) load dynamically libcafsosd.so to add additional commands (c for commands)
  - The cache manager contains all code, but needs a -libafsosd parameter for afsd to activate it.

My version for git master

- I am probably too stupid (or too old?) to use all fancy possibilities of git, therefore
  - I keep my changed code presently separate along with the original code: e.g. viced.c and viced.c.orig
  - I reset once in a while the clone of git master
  - Do a 'git pull'
  - Run a script which checks differences between my viced.c.orig and the current viced.c in master
  - Do the necessary updates to my viced.c and copy the current viced.c as my new viced.c.orig
  - Update my clone tree with all my changes
  - Build on Linux
  - Run tests in a VirtualBox Linux machine.

RPC interfaces 1.6

srv	prefix	created from	port	RPCs to
many clients and servers				
409	RXSTATS_	rxstat.xg		statistic layer of RX
database servers				
50	VOTE_	ubik_int.xg		all ubik database servers for voting
51	DISK_	ubik_int.xg		all ubik database servers sync data
52	SAMPLE_	utst_int.xg	3000	ubik test server (not really used)
73	PR_	ptint.xg	7002	ptserver
52	VL_	vldbint.xg	7003	vlserver
731	KAA_	kauth.rg	7004	kaserver
22314	BUDB_	budb.rg	7021	budb_server
cache manager				
1	RXAFSCB_	afsintcb.xg	7001	callback
2	PAGCB_	pagcb.xg	7001	NFS support
servers				
1	BOZO_	bozo.xg	7007	bosserver
1	RXAFS_	afsint.xg	7000	fileserver
4	AFSVol	volint.xg	7005	volserver
4	UPDATE_	update.xg	7008	upserver
4	RMTSYS_	rmtsys.xg	7009	rmtsysd
1	TC_	butc.xg	7025	butc

Port 7006 which is reserved for AFS, but currently not used. We'll take it for AFS/OSD because it probably passes fire-walls better being inside the 7000-7009 range.

RPC interfaces 1.6-osd

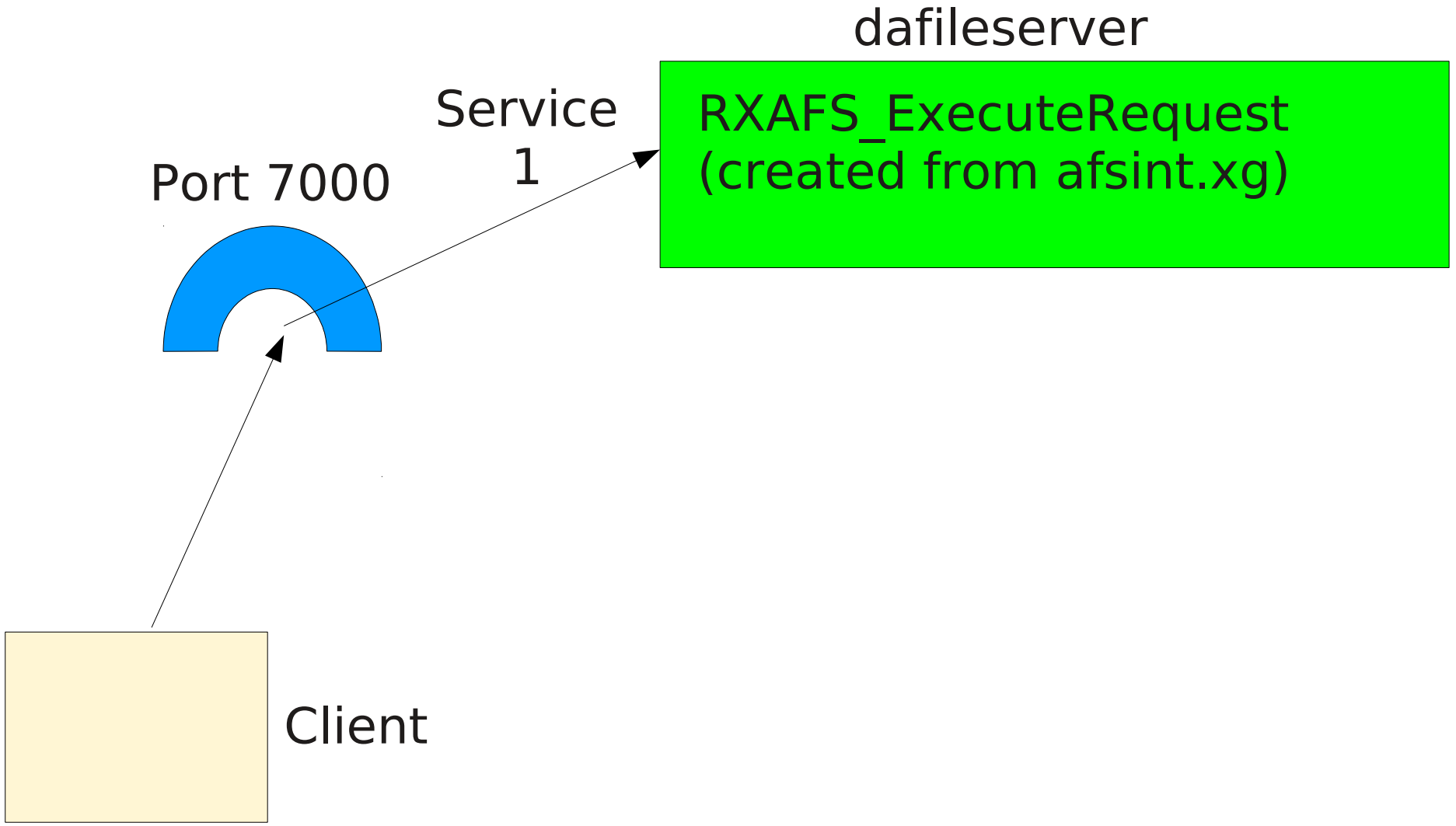
srv-id	prefix	created from	port	RPCs to
many clients and servers				
409	RXSTATS_	rxstat.xg		statistic layer of RX
database servers				
50	VOTE_	ubik_int.xg		all ubik database servers for voting
51	DISK_	ubik_int.xg		all ubik database servers sync data
52	SAMPLE_	utst_int.xg	3000	ubik test server (not really used)
73	PR_	ptint.xg	7002	ptserver
52	VL_	vldbint.xg	7003	vlserver
13	**OSDDB_**	**osddb.xg**	**7003**	**vlserver with -libafsosd**
731	KAA_	kauth.rg	7004	kaserver
13	**OSDDB_**	**osddb.xg**	**7012**	**osddbserver 1.4-osd**
22314	BUDB_	budb.rg	7021	budb_server
cache manager				
1	RXAFSCB_	afsintcb.xg	7001	callback
2	PAGCB_	pagcb.xg	7001	NFS support
servers				
1	RXAFS_	afsint.xg	7000	fileserver
2	**RXAFSOSD_**	**vicedosd.xg**	**7000**	**fileserver with -libafsosd**
4	AFSVol	volint.xg	7005	volserver
7	**AFSOSDVOL_**	**volserosd.xg**	**7005**	**volserver with -libafsosd**
900	**RXOSD_**	**rxosd.xg**	**7006**	**rxosd**
1	BOZO_	bozo.xg	7007	bosserver
4	UPDATE_	update.xg	7008	upserver
4	RMTSYS_	rmtsys.xg	7009	rmtsysd
900	**RXOSD_**	**rxosd.xg**	**7011**	**rxosd for compatibility with 1.4-osd**
1	TC_	butc.xg	7025	butc

Instance dafs

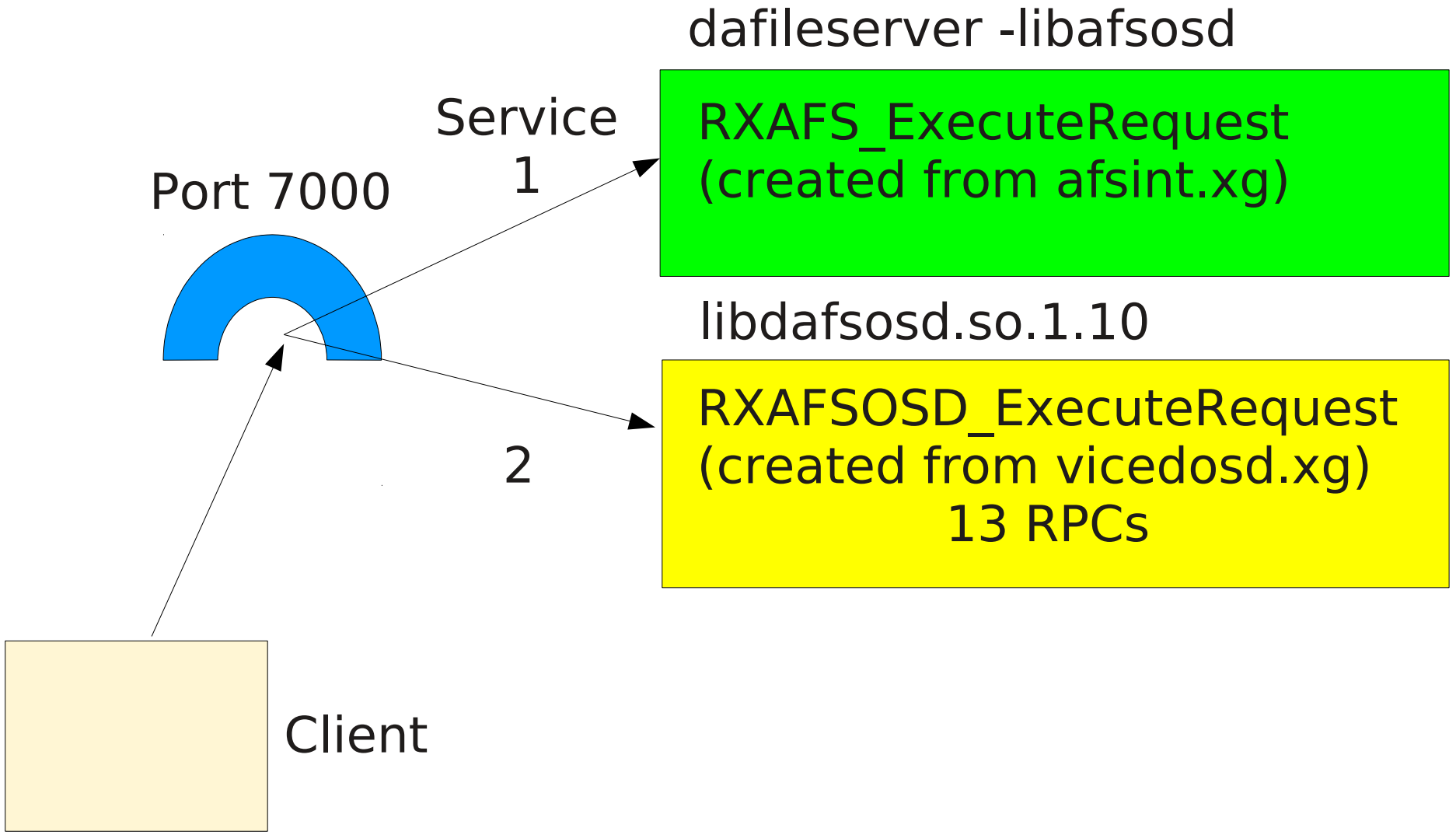

```
Instance dafs, (type is dafs) currently running normally.
    Auxiliary status is: file server running.
    Process last started at Mon Oct  1 19:00:54 2012 (3 proc starts)
    Command 1 is '/lib/openafs/dafileserver -libafsosd'
    Command 2 is '/lib/openafs/davolserver -libafsosd'
    Command 3 is '/lib/openafs/salvageserver'
    Command 4 is '/lib/openafs/dasalvager'
```

With the '-libafsosd' parameter dafileserver and davolserver load libdafsosd.so, (d for demand attach..)
The dasalvager loads the libdafsosd.so when it detects an OSD-volume.

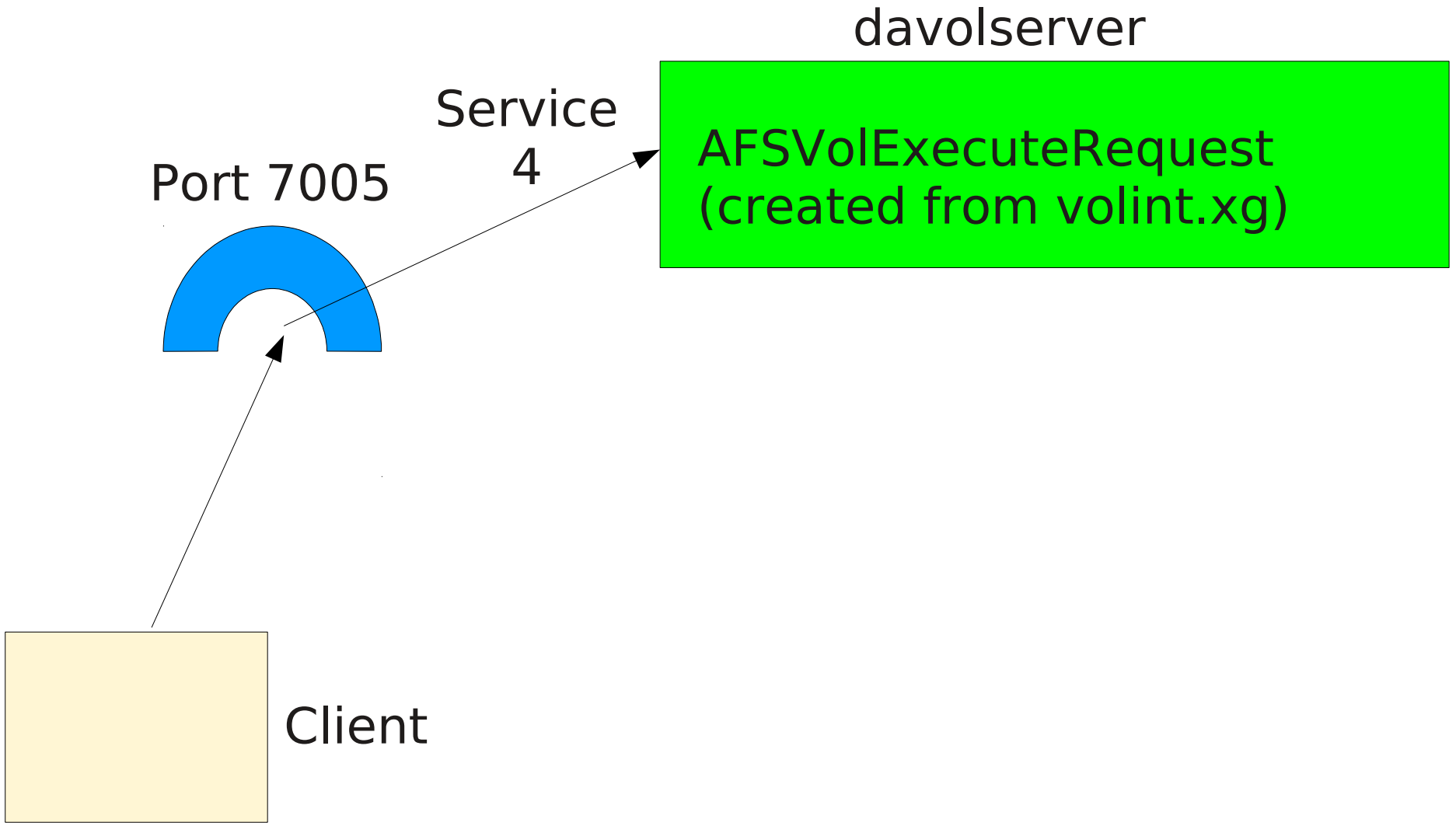
RPCs dafileserver



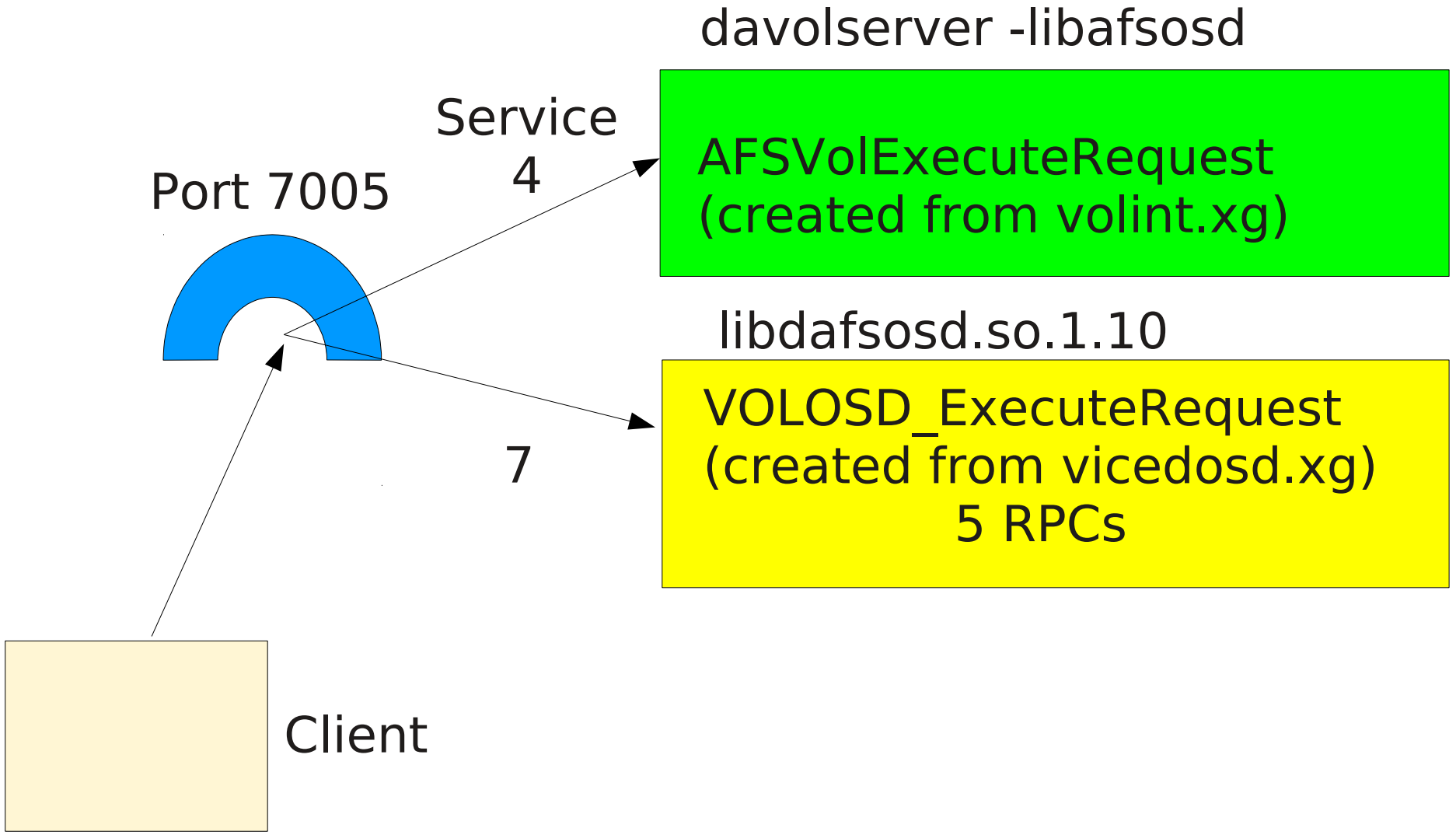
RPCs dafileserver



RPCs davolserver



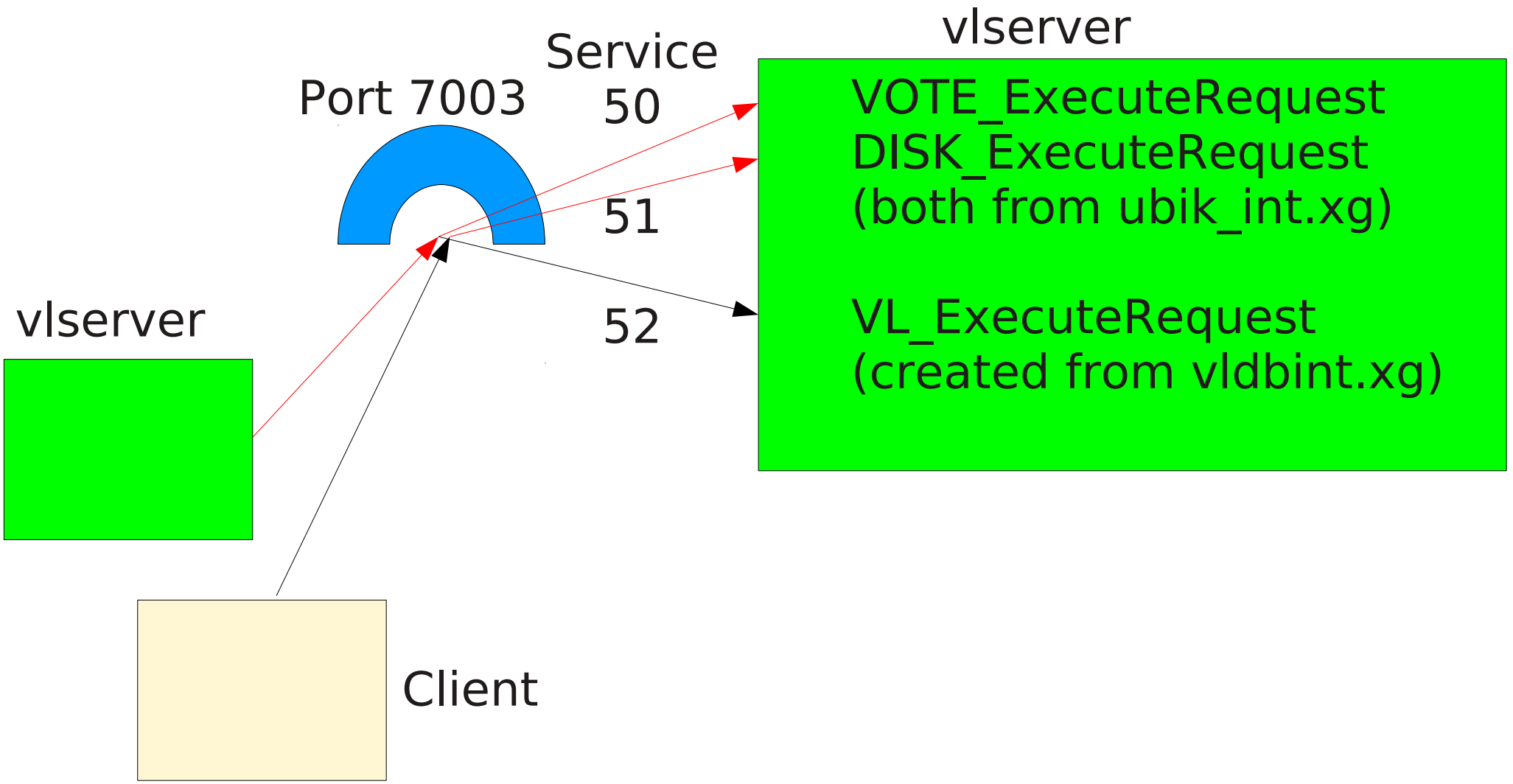
RPCs davolserver



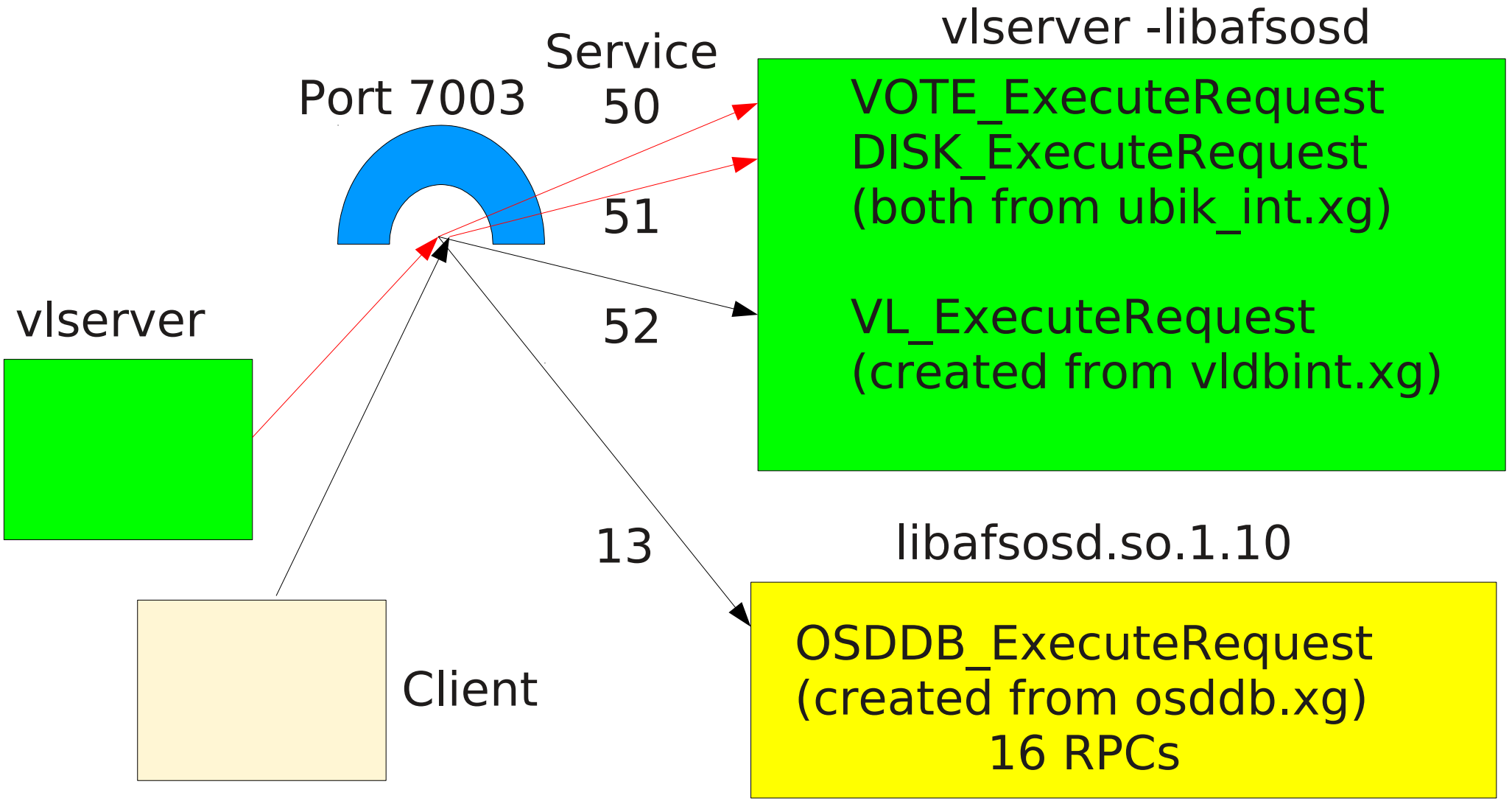
How it works

- A source file libafsosd.c is compiled in both: the binary and the shared library
- In the binary an entry load_libafsosd(...) is called to
  - load the shared library
  - call the appropriate initialization routine
- In the shared library libafsosd_init(...) is called from the initialization routines inside the shared library
  - it checks the shared library's subversion number
  - provides operation vectors and pointers to data
- After initialization has completed there exist data pointer and operation vectors for both directions
  - calls from binary to shared library
  - calls from shared library to routines in the binary
- In the binary a new service is started with the ExecuteRequest entry provided by the shared library.

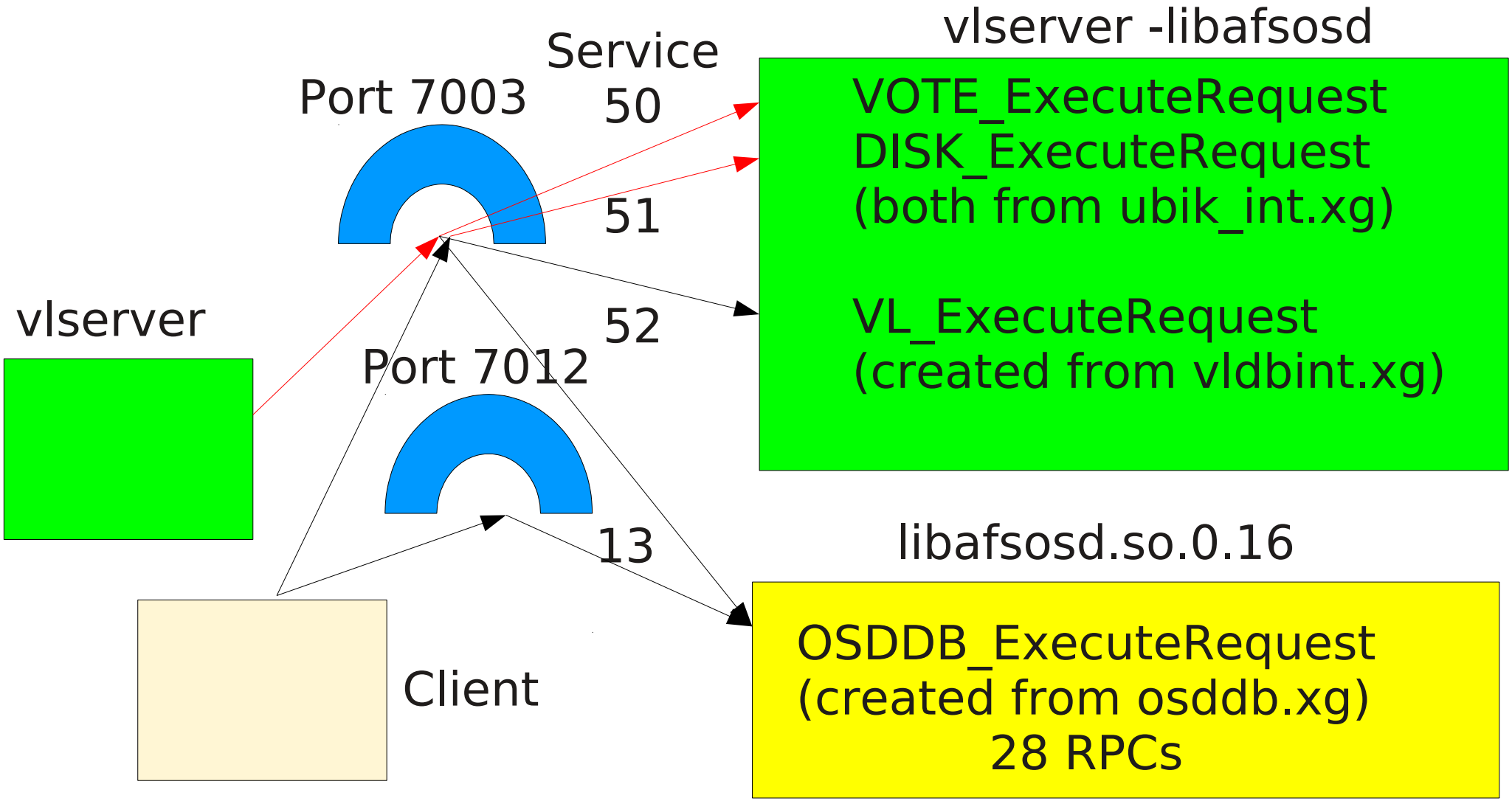
RPCs vlserver



RPCs vlserver



RPCs vlserver 1.6-osd



udebug


```
Host's addresses are: 130.183.2.114
Host's 130.183.2.114 time is Thu Oct 11 10:42:25 2012
Local time is Thu Oct 11 10:42:25 2012 (time differential 0 secs)
Last yes vote for 130.183.2.114 was 7 secs ago (sync site);
Last vote started 7 secs ago (at Thu Oct 11 10:42:18 2012)
Local dbases:
   [0] vldb    version is 1349855701.8       recovery state 1f
   [1] osddb   version is 1349944887.2       recovery state 1f
I am sync site until 53 secs from now (at Thu Oct 11 10:43:18 2012) (2 servers)
The last trans I handled on [0] was 1349855701.162
The last trans I handled on [1] was 1349944887.1288
0 locked pages, 0 of them for write
Last time a new db [0] version was labelled was:
       89244 secs ago (at Wed Oct 10 09:55:01 2012)
Last time a new db [1] version was labelled was:
       58 secs ago (at Thu Oct 11 10:41:27 2012)

Server (130.183.2.117 192.168.178.45} up = 1
   dbases: [0] vldb     dbcurrent = 1, version 1349855701.8
           [1] osddb    dbcurrent = 1, version 1349944887.2
   last vote rcvd 7 secs ago (at Thu Oct 11 10:42:18 2012),
   last beacon sent 7 secs ago (at Thu Oct 11 10:42:18 2012), last vote was no
   beaconSince=1
```

rxdebug client conn


```
Trying 130.183.2.114 (port 7003):
Free packets: 284/1111, packet reclaims: 0, calls: 1512, used FDs: 12
not waiting for packets.
0 calls waiting for a thread
22 threads are idle
0 calls have waited for a thread
Connection from host 192.168.178.45, port 7003, Cuid 8a499009/6e745e5c
  serial 145,  natMTU 1444, flags pktCksum, security index 2, client conn to service 51
  rxkad: level clear, flags pktCksum
  Received 32 bytes in 7 packets
  Sent 185428 bytes in 137 packets
    call 0: # 8, state not initialized
    call 1: # 0, state not initialized
    call 2: # 0, state not initialized
    call 3: # 0, state not initialized
Connection from host 192.168.178.45, port 7003, Cuid 8a499009/6e745e60
  serial 165,  natMTU 1444, flags pktCksum, security index 2, client conn to service 50
  rxkad: level clear, flags pktCksum
  Received 0 bytes in 82 packets
  Sent 4592 bytes in 82 packets
    call 0: # 82, state dally, mode: receiving, flags: receive_done
    call 1: # 0, state not initialized
    call 2: # 0, state not initialized
    call 3: # 0, state not initialized
```

New addition: „to service <number>“

rxdebug server conn


```
Connection from host 130.183.2.114, port 7005, Cuid 8d61c213/7e6ca3ec
  serial 94,  natMTU 1444, flags pktCksum, security index 2, server conn for service 13 (0/4)
  rxkad: level clear, flags authenticated pktCksum, expires in 818061.4 hours
  Received 276 bytes in 69 packets
  Sent 51888 bytes in 92 packets
    call 0: # 265, state dally, mode: eof, flags: receive_done
    call 1: # 0, state not initialized
    call 2: # 0, state not initialized
    call 3: # 0, state not initialized
Connection from host 130.183.2.114, port 56139, Cuid 9dbd671f/c3029630
  serial 2,  natMTU 1444, flags pktCksum, security index 2, server conn for service 52 (0/9)
  rxkad: level clear, flags authenticated pktCksum, expires in 23.6 hours
  Received 16 bytes in 1 packets
  Sent 476 bytes in 1 packets
    call 0: # 1, state dally, mode: eof, flags: receive_done
    call 1: # 0, state not initialized
    call 2: # 0, state not initialized
    call 3: # 0, state not initialized
```

New addition: „for service <number> (<active threads>/<total threads>)“

Open Problem

- To make the version for git master as slim as possible „vicep-access“ (what I called „embedded filesystems“ in earlier talks) will be possible only for shared OSDs not any more for shared fileserver partitions as in 1.4-osd.
- Nevertheless there is a problem with the Linux 3 kernel version:
  - it doesn't allow anymore to set the fs_uid
  - This makes foreground access to files in the OSD-partition impossible unless the modebits allow access to the user
  - I always thought only in the case of busy daemons those accesses would be done in the foreground, but this is not true!
  - So I have to find out how to avoid foreground accesses in this special case.

Integration into git master

- Two years ago when I sent the first patches to gerrit the reviewers had many objections against my code
  - I accept: nearly all of these objections were reasonable
  - So I redesigned much of the interface which originally tried to follow closely the SCSI T10 standard
  - This code became the basis of our 1.6-osd version in github
  - It is backward compatible to 1.4-osd
  - During the last year the OSD code was isolated into libafsosd.so
- Therefore I now would like to get the changes into the master as soon as possible. It must be checked
  - that it builds
  - the built code runs as expected
  - all other things work as before

Integration into git master

- I believe I have done what I can to restructure the code.
  - It may be not perfect, but it works.
- Once the code is integrated in git master it may be further improved
- I would appreciate the help of people knowing git better than I do to break my changes down to smaller patches which can easier be reviewed.
  - The problem is (for me) to get the dependencies between the patches correct
- To follow the permanently changing master as I do now is a pain!

Questions or comments?

Thank you