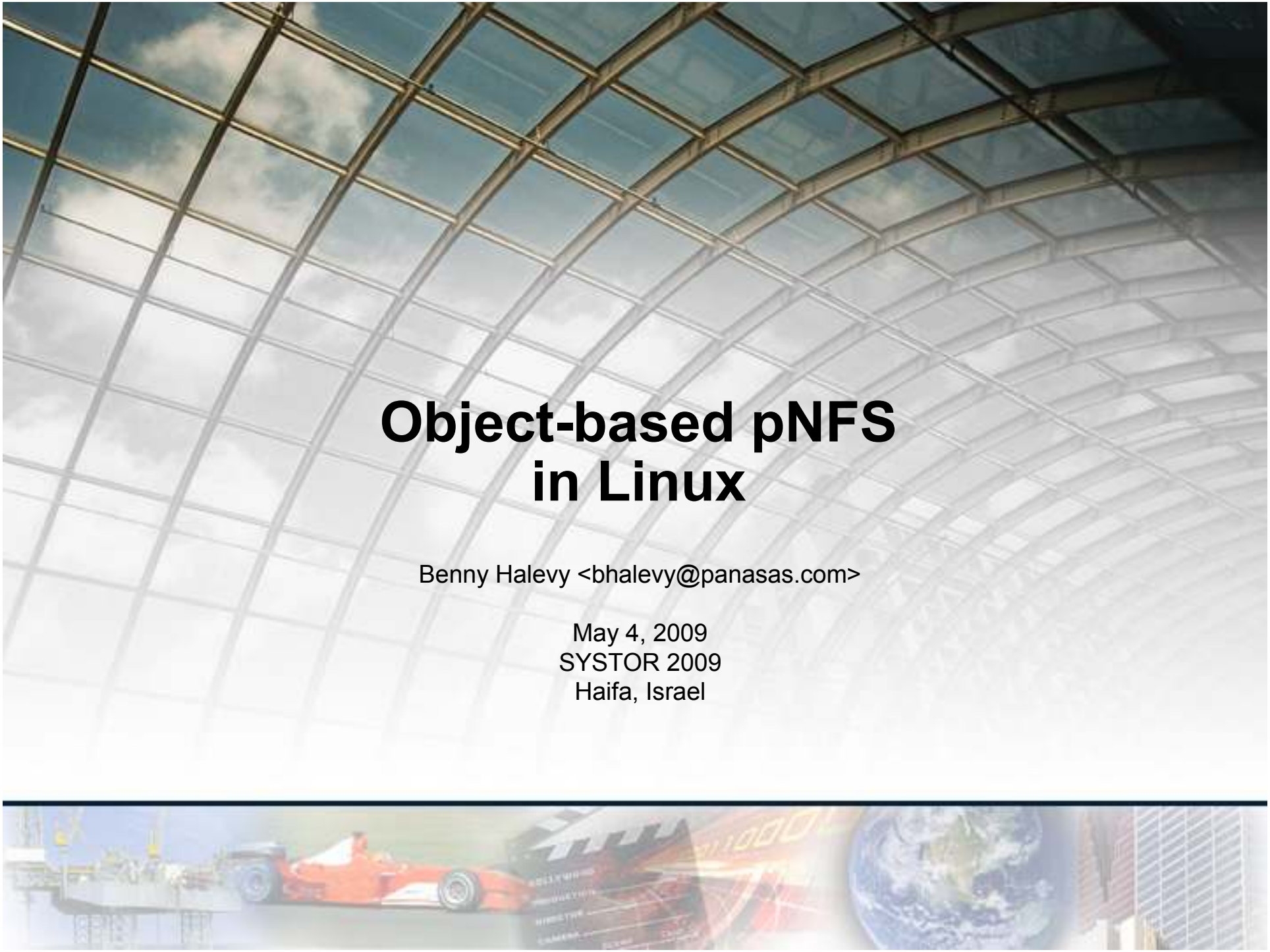


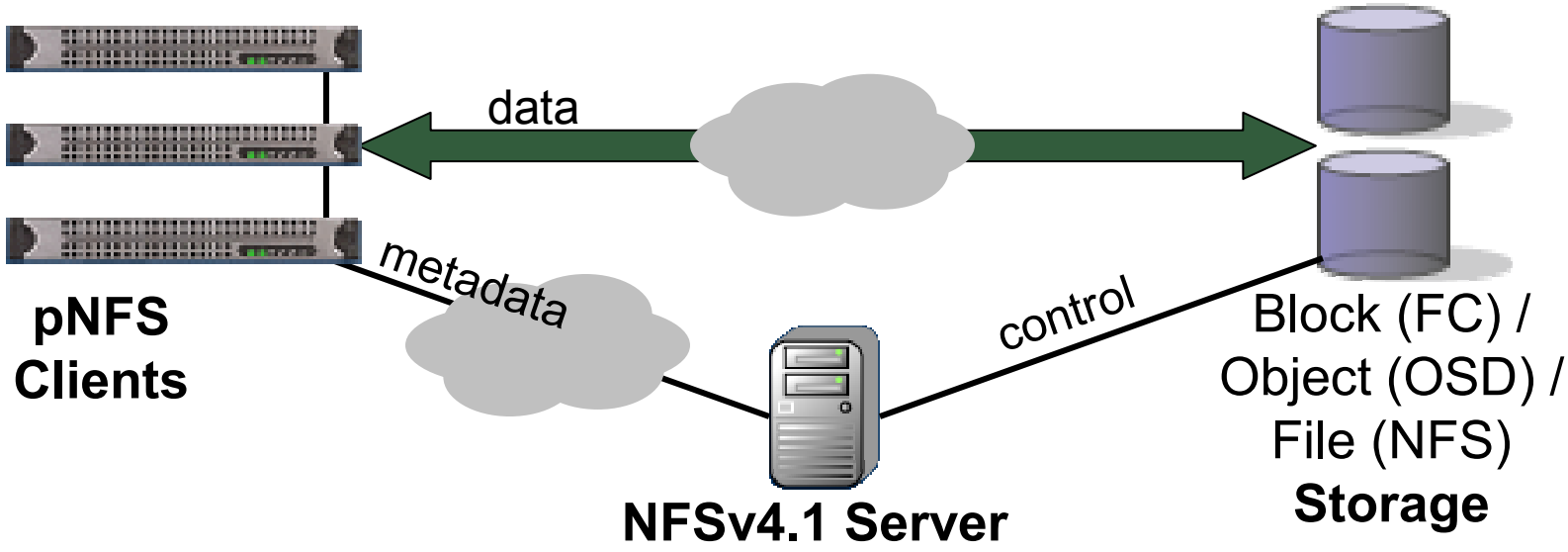
Object-based pNFS in Linux

Benny Halevy <bhalevy@panasas.com>

May 4, 2009
SYSTOR 2009
Haifa, Israel

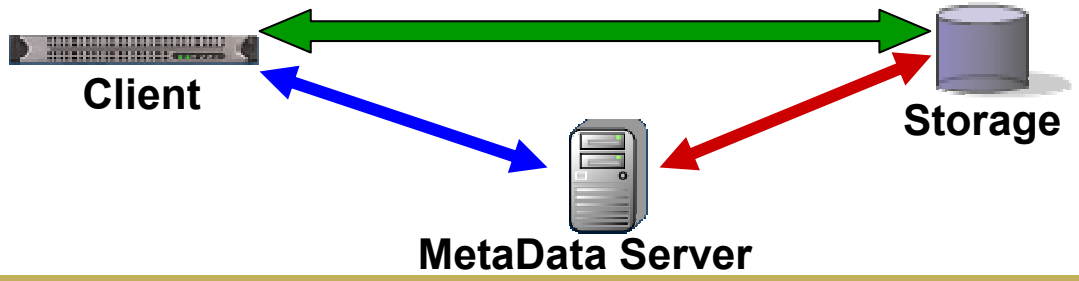
pNFS: Standard Storage Clusters

- pNFS is an extension to the Network File System v4 protocol standard
- Allows for parallel and direct access
  - From Parallel Network File System clients
  - To Storage Devices over multiple storage protocols
  - Moves the Network File System server out of the data path



The pNFS Standard

- The **pNFS** standard defines the NFSv4.1 protocol extensions between the **client** and **metadata server**
- The **I/O** protocol between the **client** and **storage** is specified elsewhere, for example:
  - SCSI **Block** Commands (**SBC**) over Fibre Channel (**FC**)
  - SCSI **Object**-based Storage Device (**OSD**) over iSCSI
  - Network **File** System (**NFS**)
- The **control** protocol between the **server** and **storage** devices is also specified elsewhere, for example:
  - SCSI **Object**-based Storage Device (**OSD**) over iSCSI

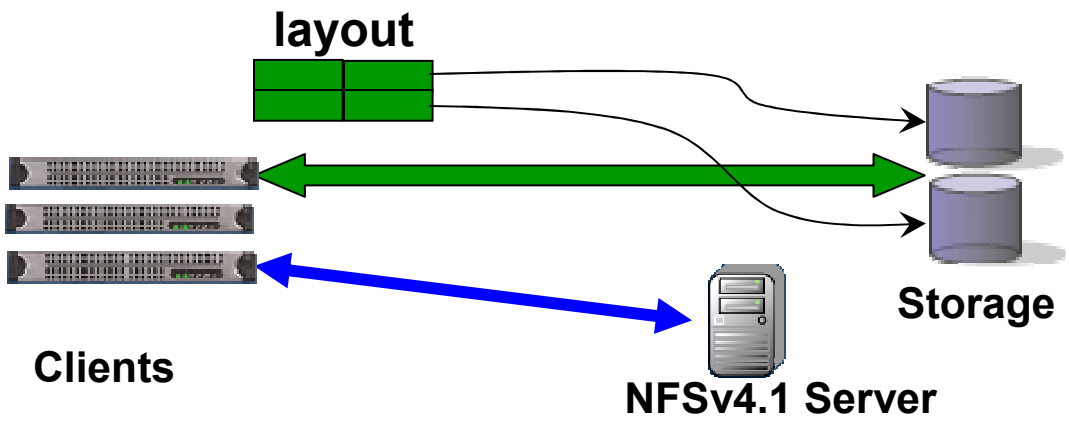


pNFS Status

- pNFS is part of the IETF NFSv4 minor version 1 standard draft
  - Draft approved by IESG for publication as RFC
- Reference Linux implementation hosted on linux-nfs.org
  - Git://linux-nfs.org/~bhalevy/linux-pnfs.git
  - Initial server patches for NFSv4.1 pushed to Linux 2.6.30
  - Client and back-channel parts destined for 2.6.31, pNFS to follow.
- Active Development:
  - CITI (Files for GFS2, Blocks layout driver, Files layout driver for Windows)
  - DESY (Files, HSM-like system)
  - EMC (Blocks, based on HighRoad)
  - IBM (Files, based on GPFS)
  - LSI Logic (Blocks back end)
  - Netapp (Files over NFSv4)
  - Panasas (Objects, based on Panasas ActiveScale Storage Cluster OSDs)
  - Sun (Files over NFSv4)
- Testing at Los Alamos National Lab. (LANL)
- See also http://pnfs.com for updates

pNFS Layouts

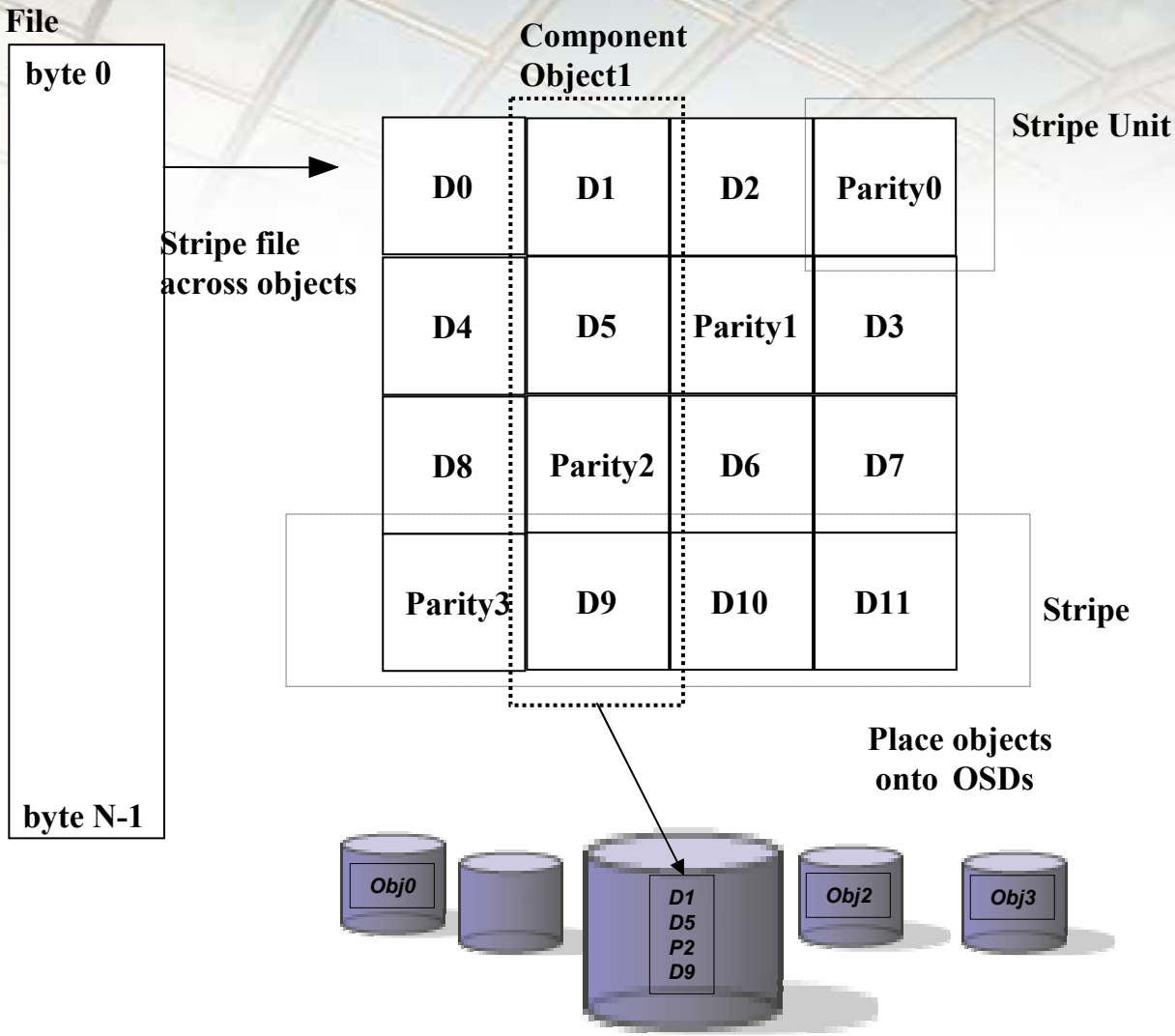
- Client gets a *layout* from the NFS Server
- The layout maps the file onto storage devices and addresses
- The client uses the layout to perform direct I/O to storage
- The server can recall the layout at any time
- Client commits changes and returns the layout when it's done
- pNFS is optional, the client can always use regular NFSv4 I/O



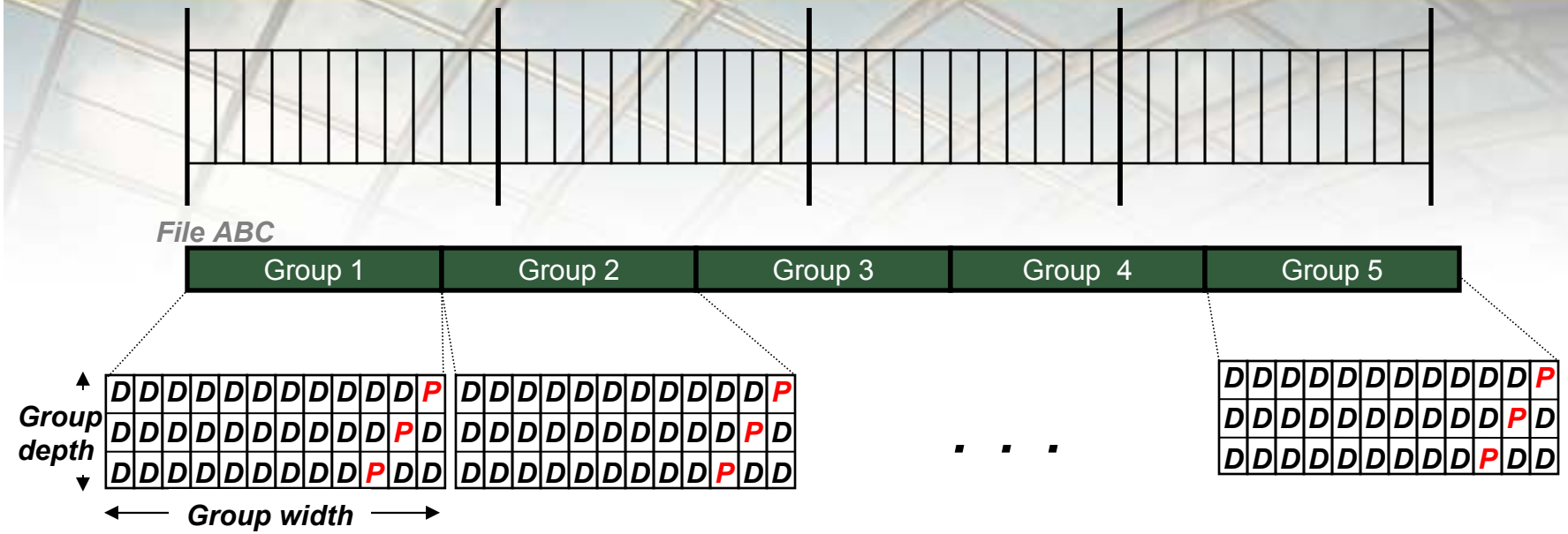
pNFS Protocol Model

- **LAYOUTGET**
  - (filehandle, io_type, byte range) -> type-specific layout
- **LAYOUTRETURN**
  - (filehandle, byte range, stateid, layout-specific info) -> server can release state about the client
- **LAYOUTCOMMIT**
  - (filehandle, byte range, updated attributes, layout-specific info) -> server ensures that data is visible to other clients
  - Timestamps and end-of-file attributes are updated
- **CB_LAYOUTRECALL, CB_RECALL_ANY**
  - Server tells the client to stop using layout(s)
- **GETDEVICELIST**, **GETDEVICEINFO**, **CB_NOTIFY_DEVICEID**
  - Map DeviceID in layout to type-specific addressing information
- **Layout_hint** attribute
  - May be set on CREATE to affect the file layout.

Object-based RAID design



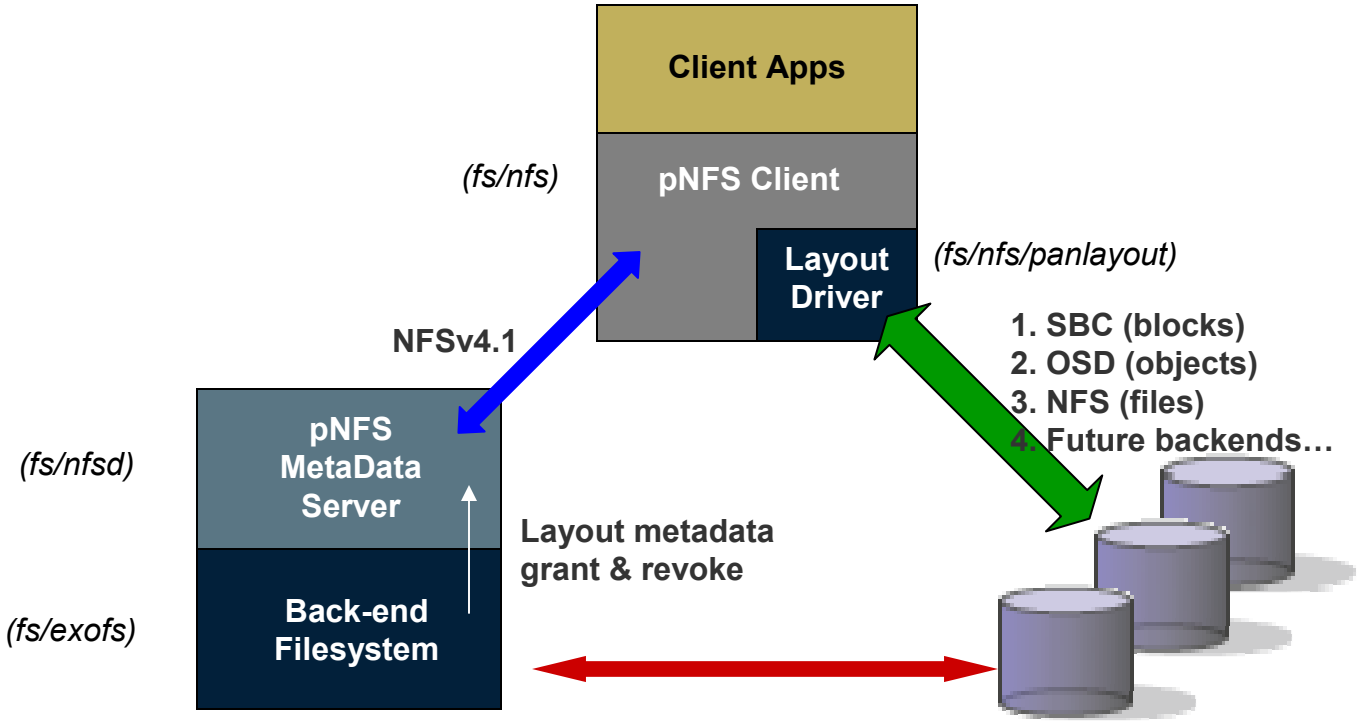
Parity stripes



- Files consist of one (or more) RAID 5 *parity stripes*.
  - A parity stripe is a collection of StorageBlades over which data is organized using the RAID 5 algorithm.
  - Each RAID 5 “parity stripe” is typically 9-11 devices in size.
  - Typically, only multi-GB files span more than one parity stripe.
  - In the above picture, File “ABC”’s layout uses five parity stripes.
    - With a single disk failure, File ABC would only have to reconstruct the data w/in a single parity stripe.

Linux pNFS Overview

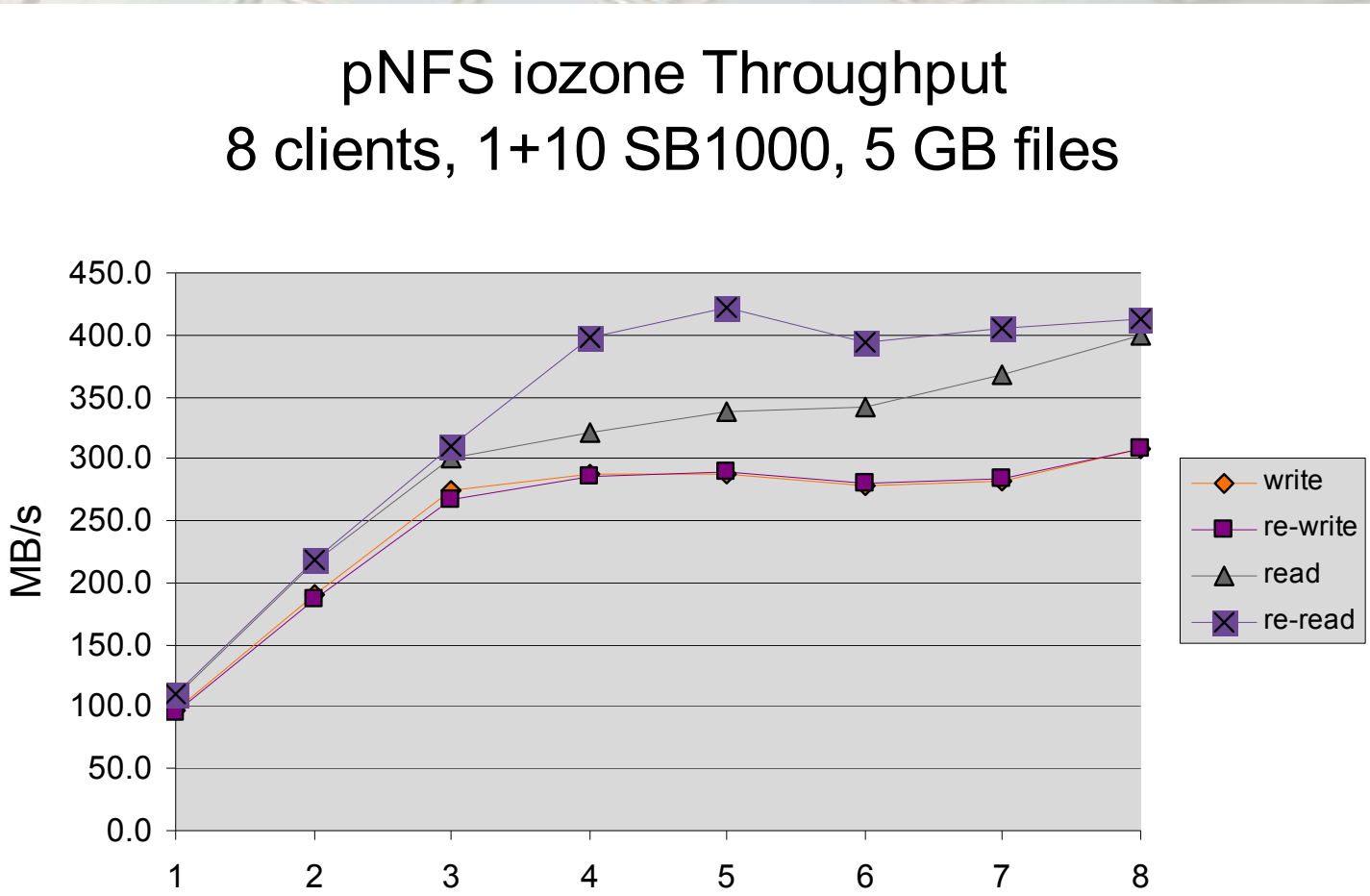
- Common client and server code for different back ends
- Integrated with existing NFSv4 codebase



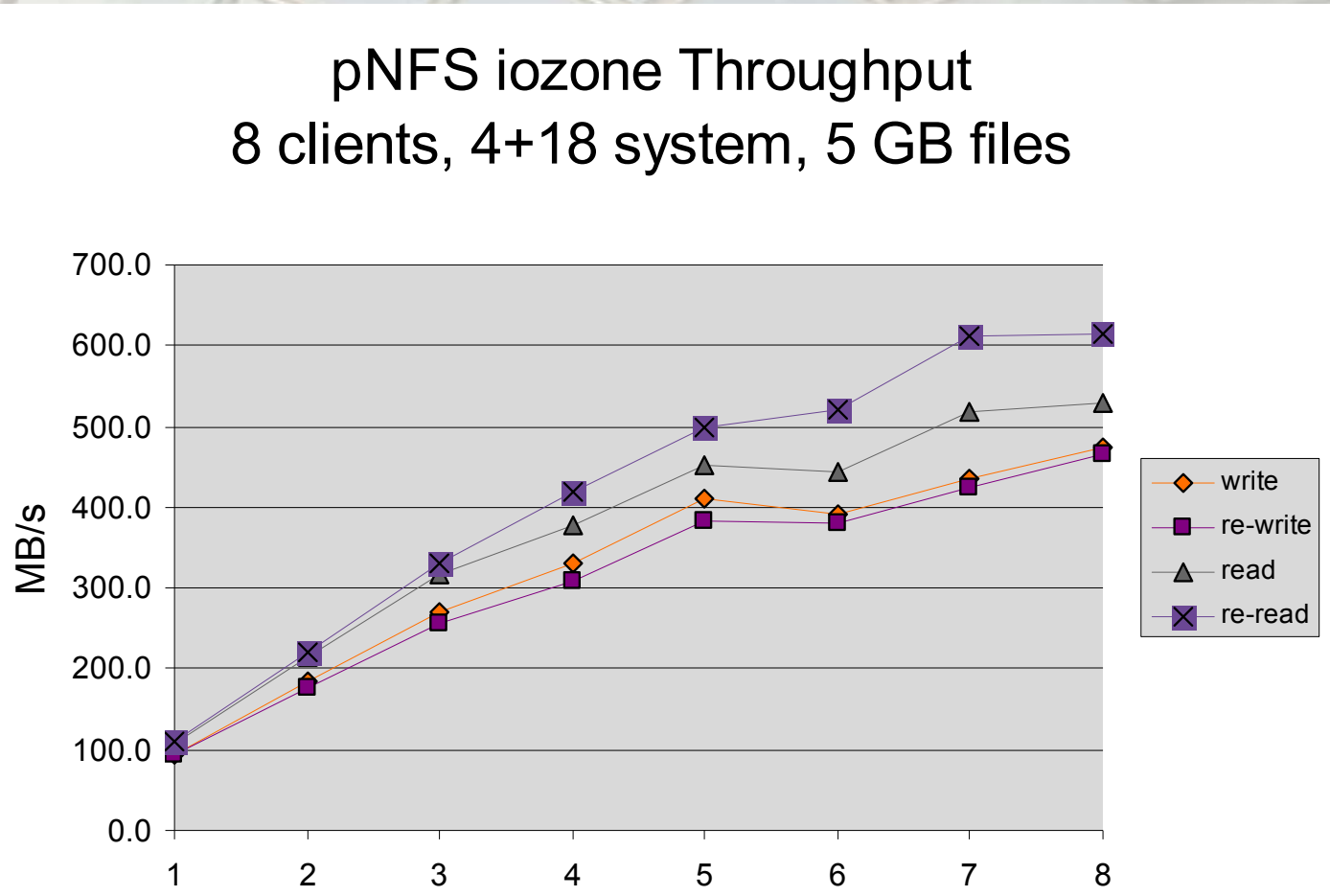
OSD on Linux

- Development hosted on http://open-osd.org
- OSD Initiator available in Linux 2.6.30
  - Implements the OSD2r05 draft standard.
- EXOFS (formerly IBM's OSDFS) available in Linux 2.6.30
  - To be augmented with pNFS exportability
  - RAID (0, 0+1, 5, …) over Objects
- OSD Target (Ohio Supercomputer Center) maintained on open-osd.org
- Experimental osdblk driver done by Jeff Garzik

Prototype implementation performance



Prototype implementation performance (2)



Resources

- Standard
  - http://www.ietf.org/html.charters/nfsv4-charter.html: NFSv4 @ IETF
  - http://tools.ietf.org/html/draft-ietf-nfsv4-minorversion1: latest NFSV4.1 I-D
  - http://tools.ietf.org/html/draft-ietf-nfsv4-pnfs-obj: latest pnfs-obj I-D
  - http://www.t10.org/cgi-bin/ac.pl?t=f&f=osd2r05.pdf: latest OSD2 draft
- Web
  - http://www.pnfs.com
  - http://www.open-osd.org
  - http://wiki.linux-nfs.org/wiki/index.php/PNFS_prototype_design
- Linux dev
  - git://git.linux-nfs.org/~bhalevy/linux-pnfs.git: linux-pnfs kernel tree
  - git://git.open-osd.org/open-osd.git: open-osd initiator + exofs

Thank You

Any Questions?

pNFS Layouts: Object

- Object layout is an array of object IDs, capabilities, and striping parameters

```
osd_layout {
  data map
  components index (for sub-stripes)
  list of components (creds)
}

object_cred {
  object_id
  osd_version
  cap_key_sec
  opaque credential (cap_key, cap)
}

objectid {
  device_id
  partition_id
  object_id
}
```

```
data_map {
  num_comps
  stripe_unit
  group_width
  group_depth
  mirror_cnt
  raid_algorithm
}
```

pnfs-obj device info

```
struct pnfs_osd_deviceaddr4 {
  pnfs_osd_targetid4           oda_targetid;
  pnfs_osd_targetaddr4         oda_targetaddr;
  uint64_t                  oda_lun;
  opaque                    oda_systemid<>;
  pnfs_osd_object_cred4 oda_root_obj_cred;
  opaque                    oda_osdname<>;
};

union pnfs_osd_targetid4 switch
  (pnfs_osd_targetid_type4 oti_type) {
  case OBJ_TARGET_SCSI_NAME:         string
  oti_scsi_name<>;
  case OBJ_TARGET_SCSI_DEVICE_ID:    opaque
  oti_scsi_device_id<>;
};

union pnfs_osd_targetaddr4 switch (bool ota_available)
```

Performing an OSD Command

- **`osd_start_request`**: allocate a request
- **`osd_req_{format,create_partition,...}`**: encode CDB
- **`osd_add_{get,set}_attr`**: add get/set attrs to CDBBoth list or page modes supported
- **`osd_finalize_request`**: set final pointers and sign the request
- **`osd_execute_request`**: execute request via block layer
- **`osd_req_decode_sense`**
- **`osd_req_decode_get_attr{,_list}`**
- **`osd_end_request`**: destroys the request