Parallel I/O on JUQUEEN

4. Februar 2014, JUQUEEN Porting and Tuning Workshop

Wolfgang Frings

w.frings@fz-juelich.de

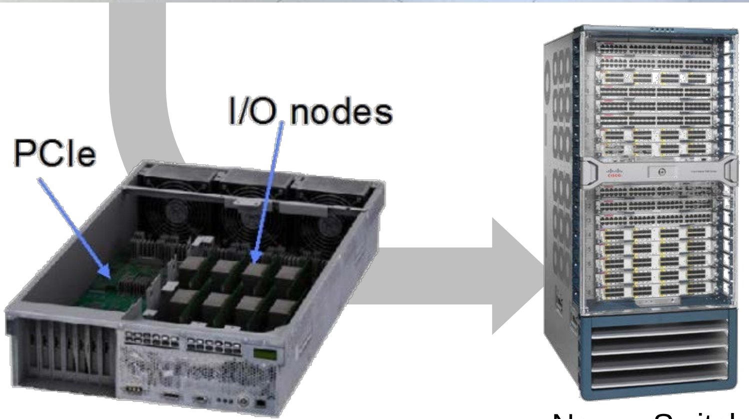
Jülich Supercomputing Centre

Overview

- Parallel I/O from different views
  - Hardware: Blue Gene/Q- and Just-I/O infrastructure
  - System Software: GPFS and I/O-forwarding
  - Application: Parallel I/O libraries
- Pitfalls
  - Small blocks, I/O to individual files, false sharing
  - Tasks per Shared File, portability
- *SIONlib* Overview
- Task-mapping to I/O-node
- I/O characterization with *darshan*
- I/O strategies

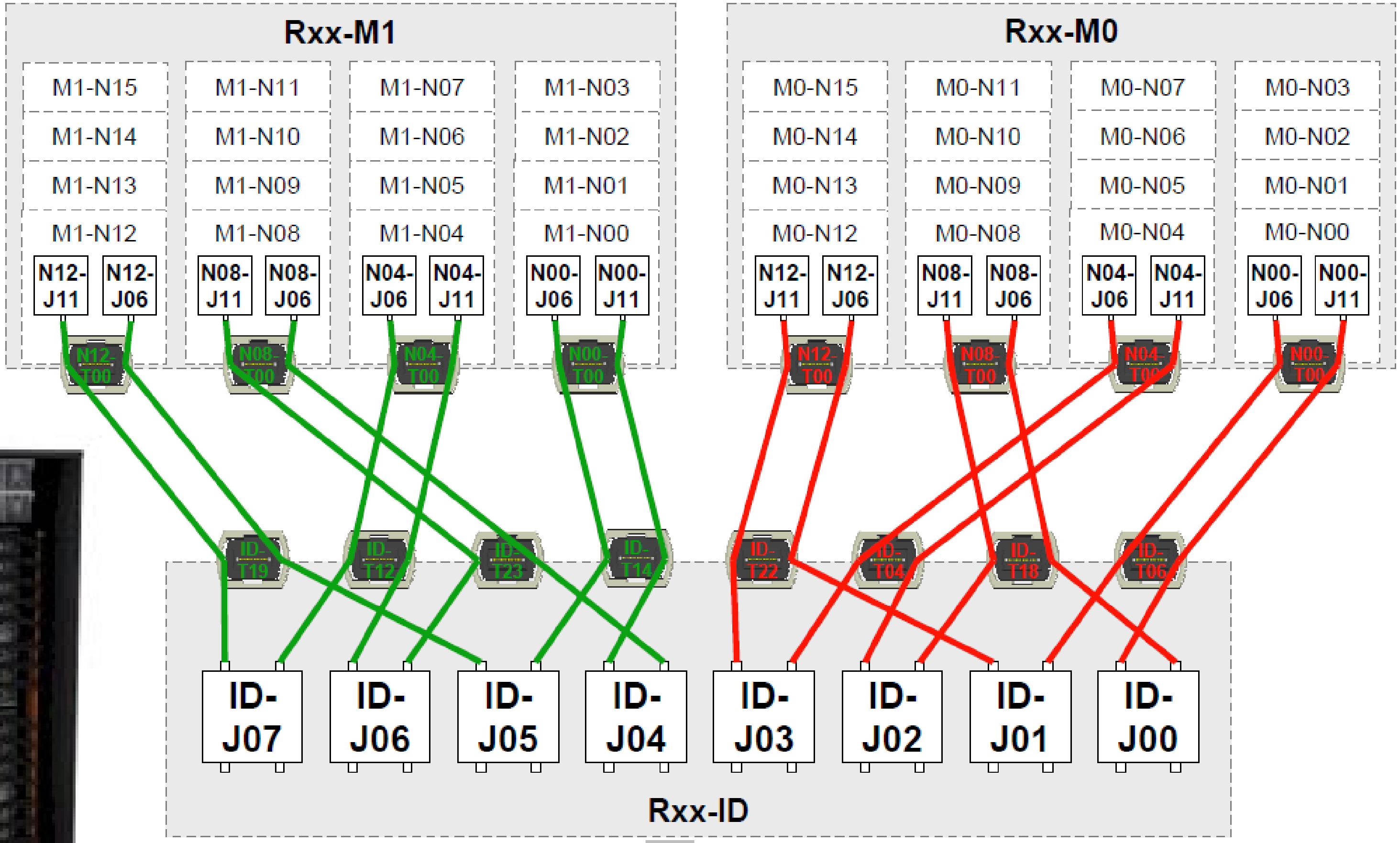
JUQUEEN: Jülich's Scalable Petaflop System

- IBM Blue Gene/Q JUQUEEN
- IBM PowerPC® A2 1.6 GHz,
  16 cores per node
  28 racks (7 rows à 4 racks)
  28,672 nodes (**458,752 cores**)
- 5D torus network
- 5.9 Pflop/s peak
  5.0 Pflop/s Linpack
- Main memory: **448 TB**
- **I/O Nodes**: **248** (27x8 + 1x32)
- **Network**: 2x CISCO Nexus 7018
  Switches (connect I/O-nodes)
  Total ports: **512 10 GigEthernet**



Nexus-Switch

Blue Gene/Q: I/O-node cabling (8 ION/Rack)



Rxx-M1

M1-N15 M1-N11 M1-N07 M1-N03
M1-N14 M1-N10 M1-N06 M1-N02
M1-N13 M1-N09 M1-N05 M1-N01
M1-N12 M1-N08 M1-N04 M1-N00

N12-J11 N12-J06 N08-J11 N08-J06 N04-J06 N04-J11 N00-J06 N00-J11

N12-T00 N08-T00 N04-T00 N00-T00

Rxx-M0

M0-N15 M0-N11 M0-N07 M0-N03
M0-N14 M0-N10 M0-N06 M0-N02
M0-N13 M0-N09 M0-N05 M0-N01
M0-N12 M0-N08 M0-N04 M0-N00

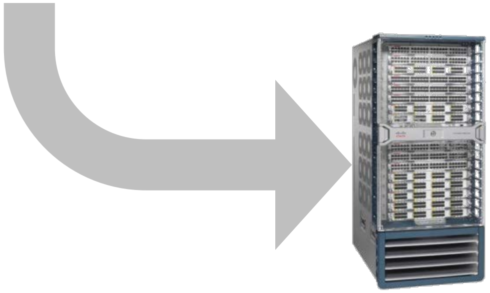
N12-J11 N12-J06 N08-J11 N08-J06 N04-J06 N04-J11 N00-J06 N00-J11

N12-T00 N08-T00 N04-T00 N00-T00

ID-T19 ID-T12 ID-T23 ID-T14 ID-T22 ID-T04 ID-T18 ID-T06

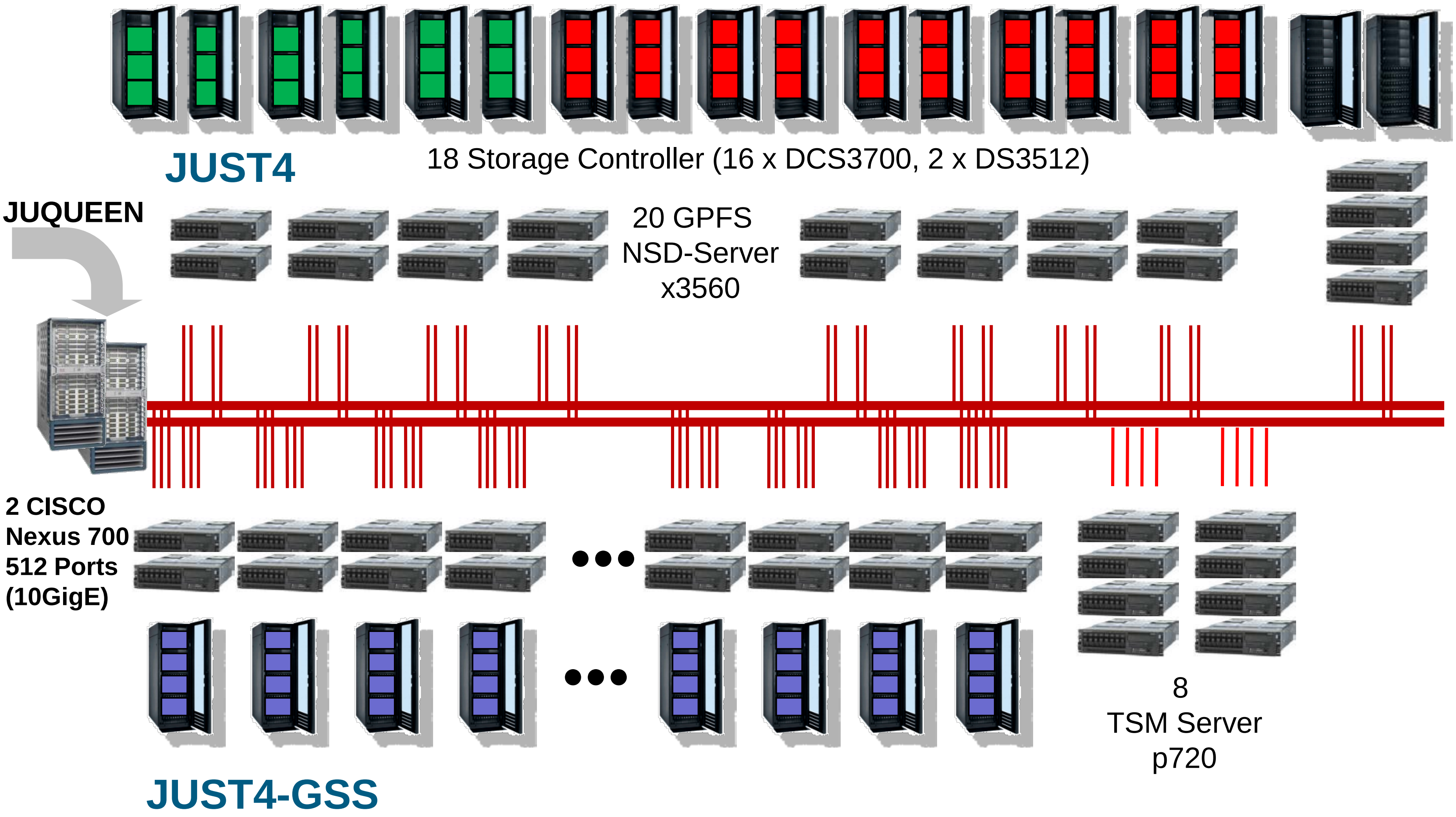
ID-J07 ID-J06 ID-J05 ID-J04 ID-J03 ID-J02 ID-J01 ID-J00

Rxx-ID



© IBM 2012

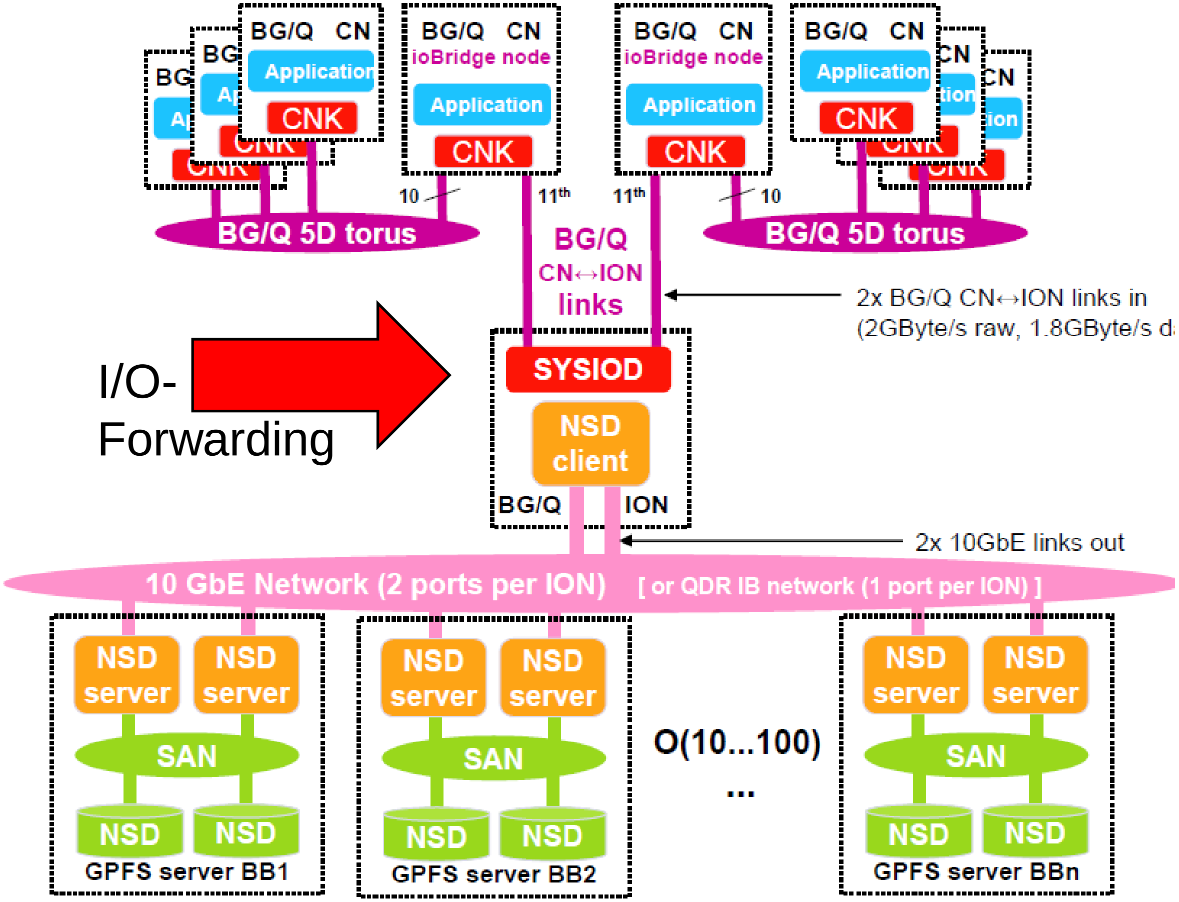
JUQUEEN and JUST I/O-Network

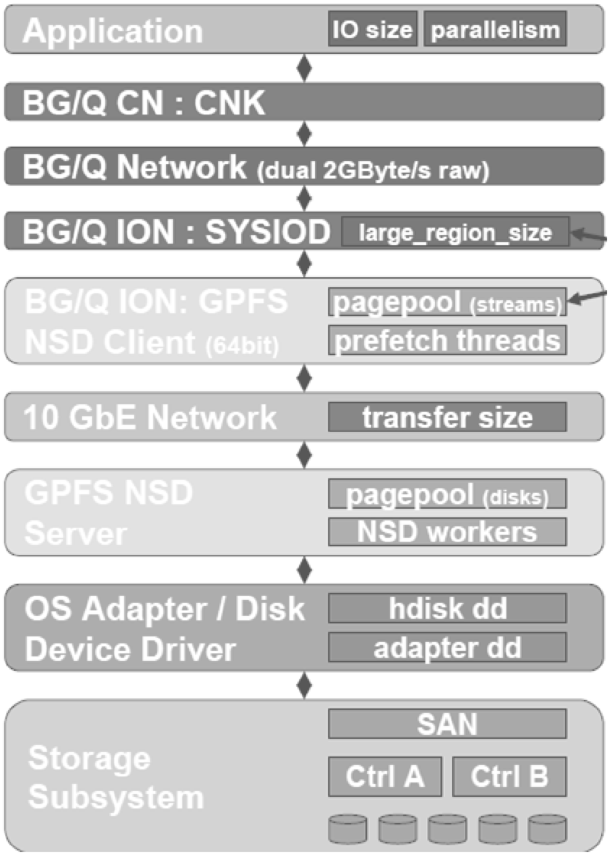


Parallel I/O Hardware at JSC (Just4, GSS)

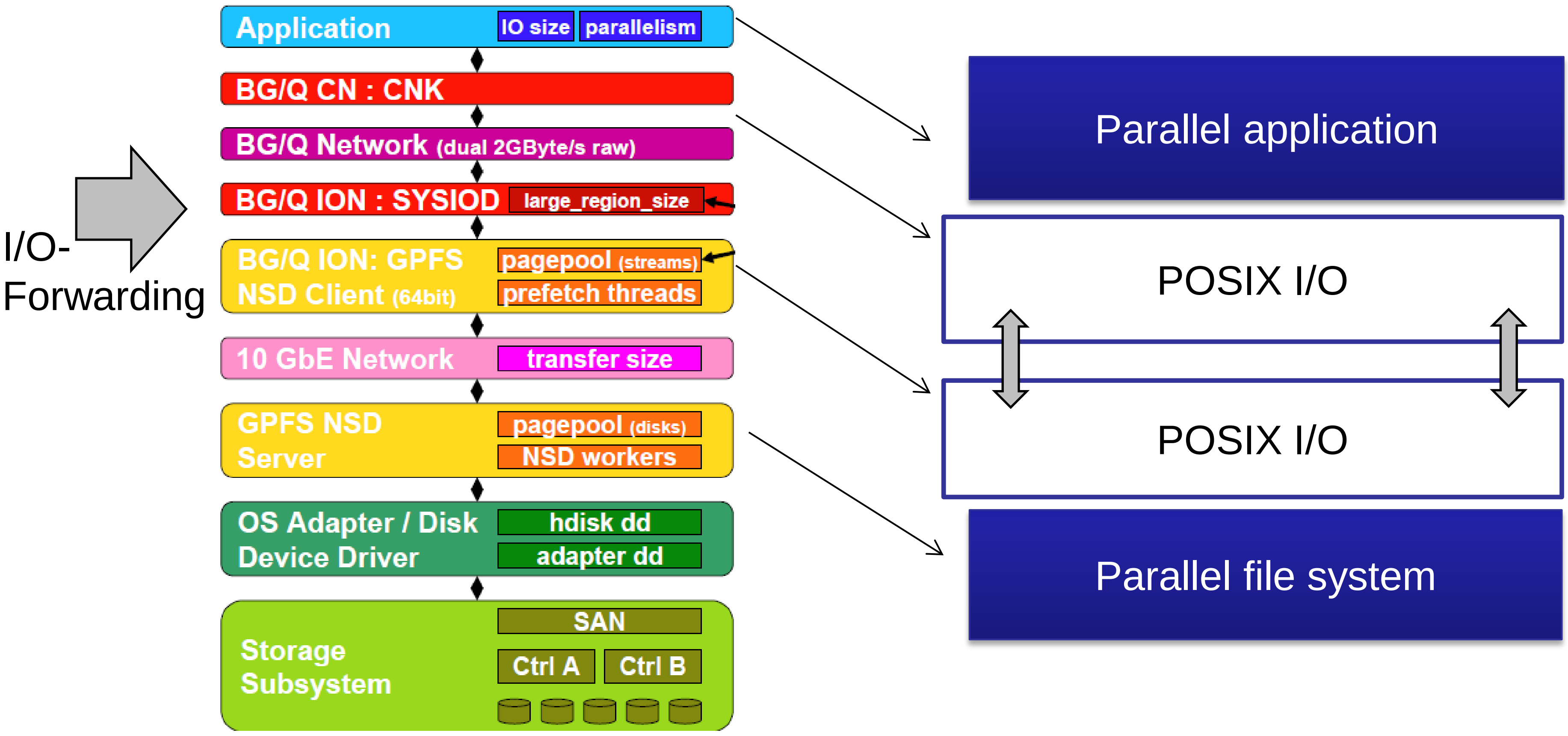
- **Ju**elich **St**orage Cluster (JUST)
- **Just4** (04/2012)
  - GPFS-Filesystems $HOME, $ARCH
  - Capacity: 3.4 Pbyte
  - Hardware: 2x DS3512, 16x DCS3700
- **Just4-GSS** (since 09/2013)
  - GPFS-Filesystem $WORK
  - Capacity: **7.4 Pbyte**
    I/O Bandwidth: up to **200 GB/sec**
  - Hardware: IBM System x® GPFS™ Storage Server solution, GPFS Native RAID
  - 20 Building blocks: each 2 x X3650 M4 server, 232 NL-SAS disks (2TB), 6 SSD

Software View to Parallel I/O: … GPFS on IBM Blue Gene/Q (I)





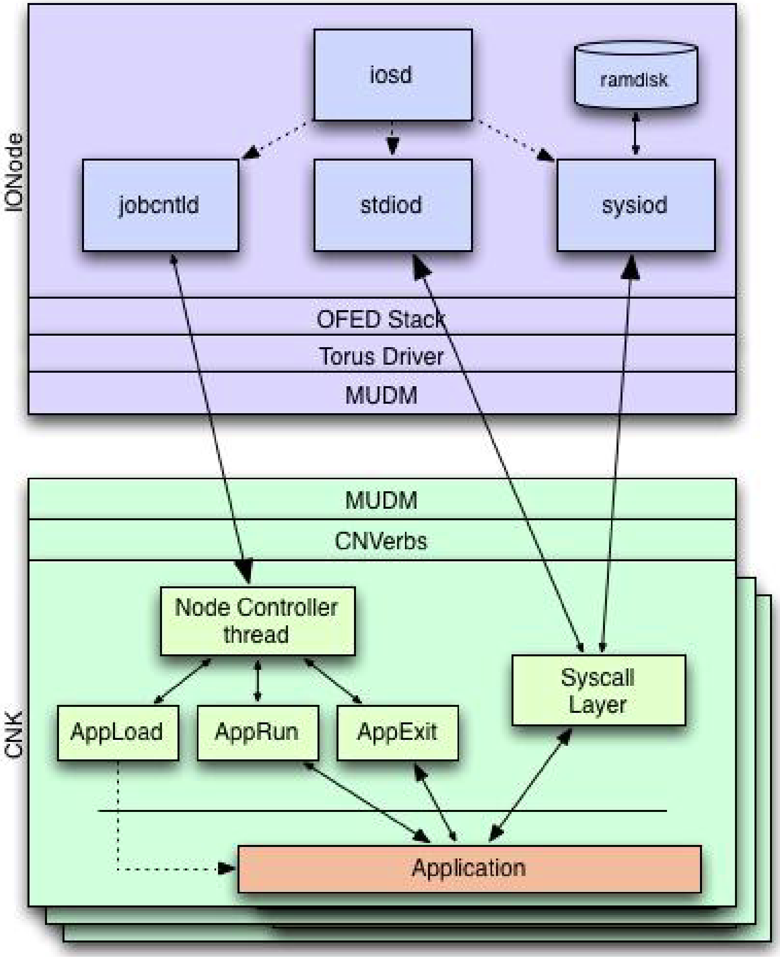
Software View to Parallel I/O: … GPFS on IBM Blue Gene/Q (II)



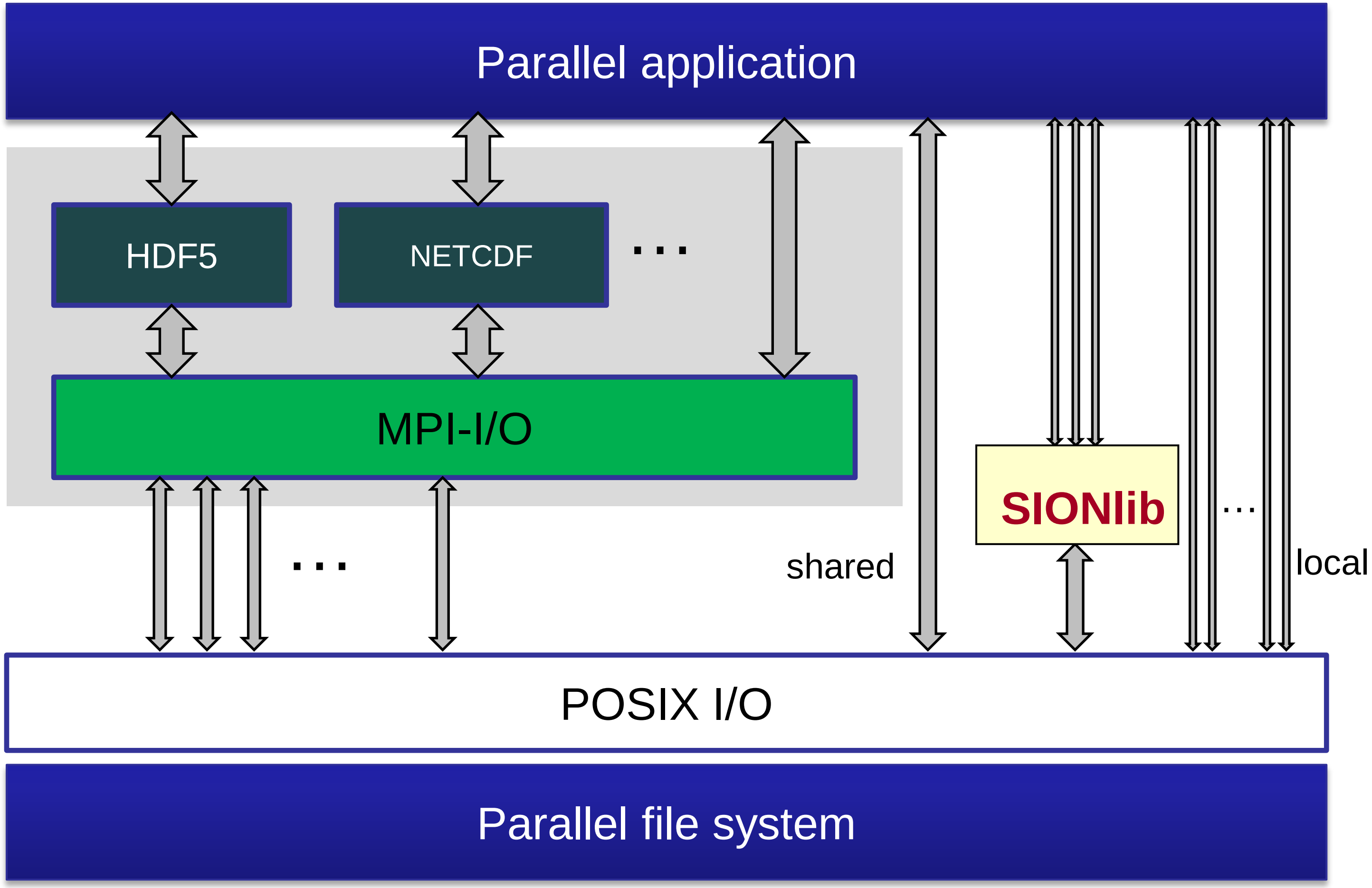
© IBM 2012

Blue Gene/Q: I/O Services

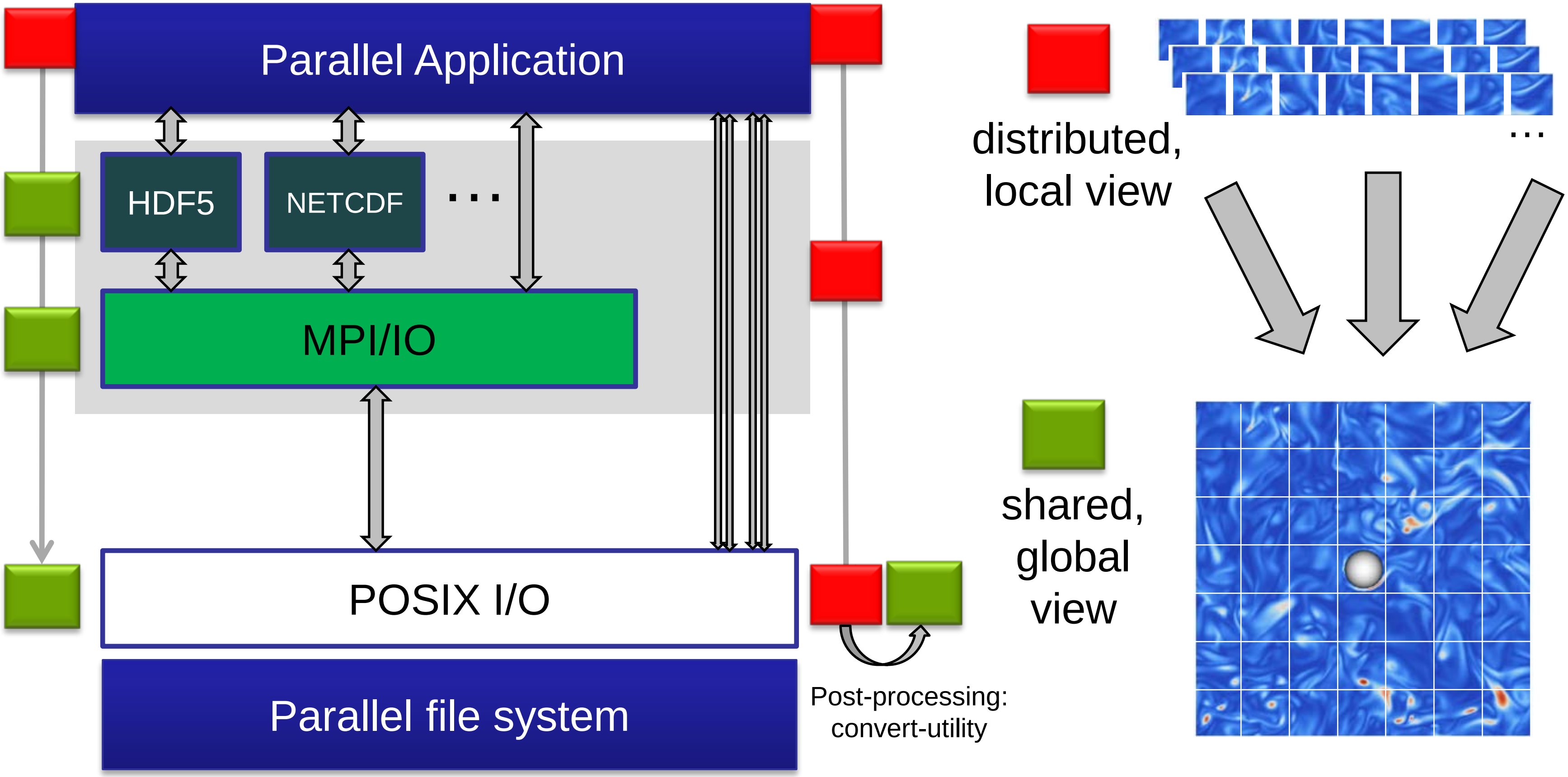
- Function shipping system calls to I/O-node
- Support NFS, GPFS, Lustre and PVFS2 filesystems
- PowerPC64 Linux running on 17 cores
- Supports ratio of **8192:1** compute task to I/O-node
  - Only 1 I/O-Proxy per compute node
  - Significant internal changes from BGP
- Standard communications protocol
  - OFED verbs
  - Using Torus DMA hardware for performance



Application View to Parallel I/O



Application View: Data Distribution

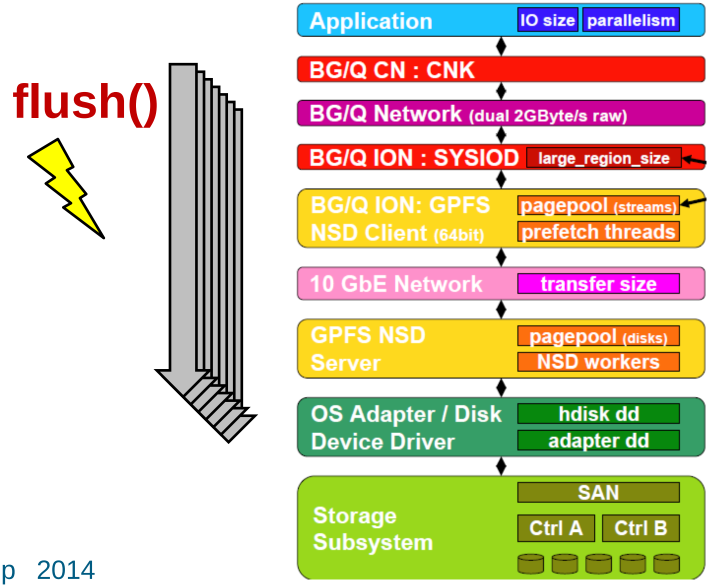


Software-view

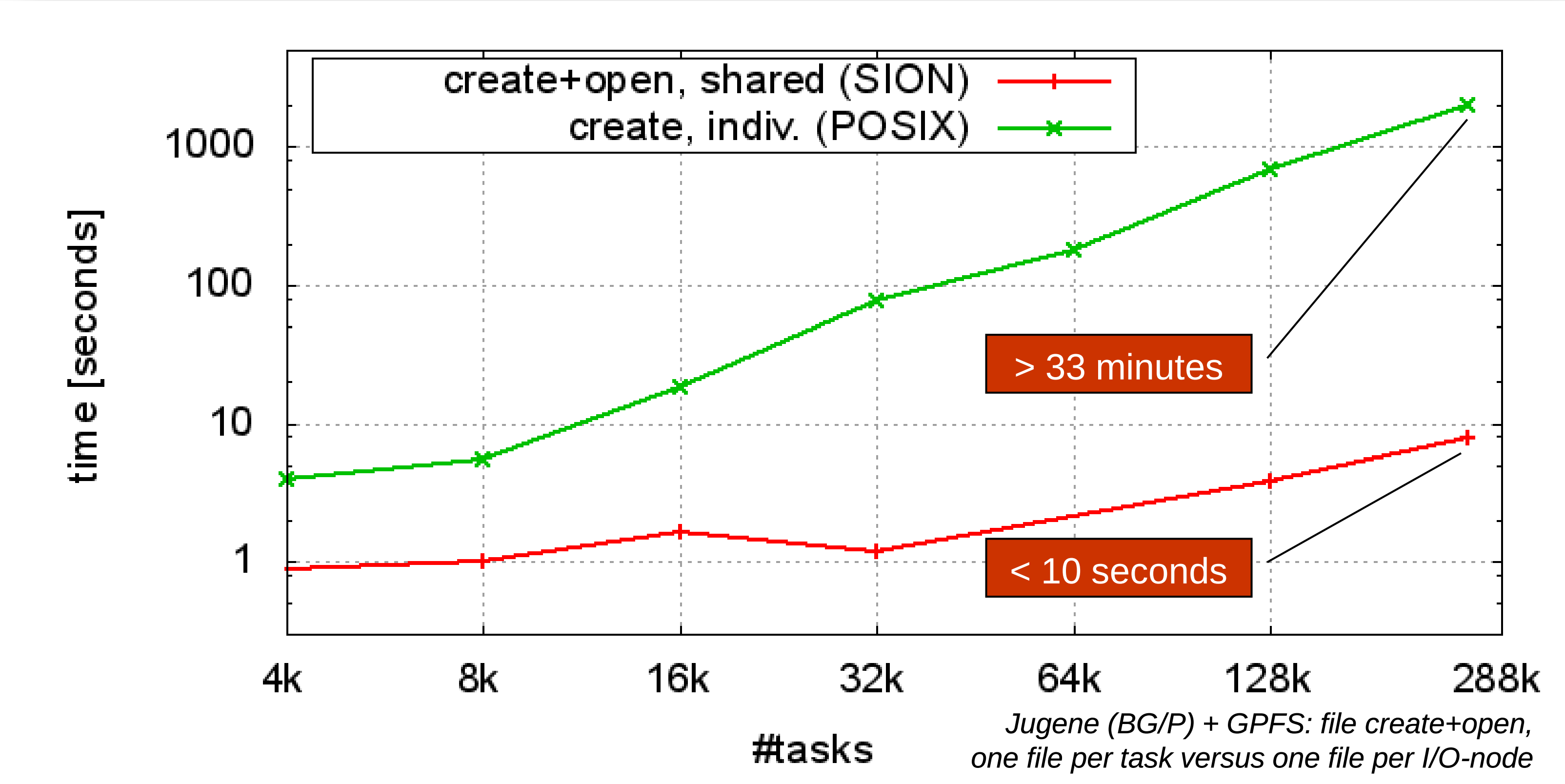
Data-view

Pitfall 1: Frequent flushing on small blocks

- Modern file systems in HPC have large file system blocks
- A flush on a file handle forces the file system to perform all pending write operations
- If application writes in small data blocks the same file system block it has to be read and written multiple times
- Performance degradation due to the inability to combine several write calls



Pitfall 2: Parallel Creation of Individual Files



Jugene (BG/P) + GPFS: file create+open, one file per task versus one file per I/O-node

- Contention at node doing directory updates (directory meta-node)
- Pre-created files or own directory per task may help performance, but does not simplify file handling
- Complicates file management (e.g. archive)

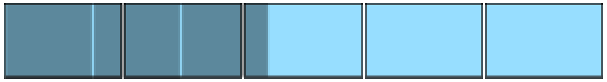
→ **shared files are mandatory**

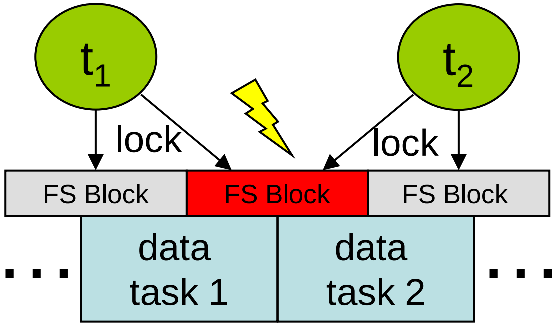
Pitfall 3: False sharing of file system blocks

- Parallel I/O to shared files (POSIX)

🟦 Free file system block

⬛ Data block



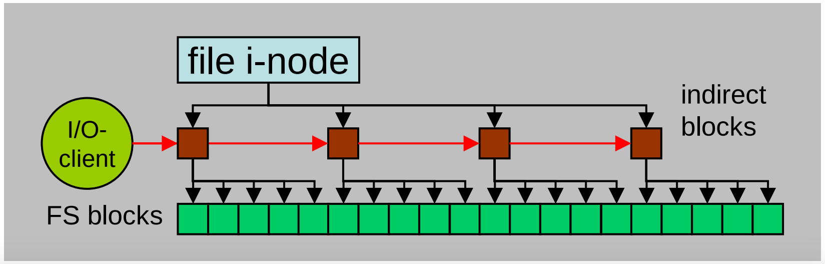


- Data blocks of individual processes do not fill up a complete file system block
- Several processes share a file system block
- Exclusive access (e.g. write) must be serialized
- The more processes have to synchronize the more waiting time will propagate

Pitfall 4: Number of Tasks per Shared File

Meta-data wall on file level

- File meta-data management
- Locking

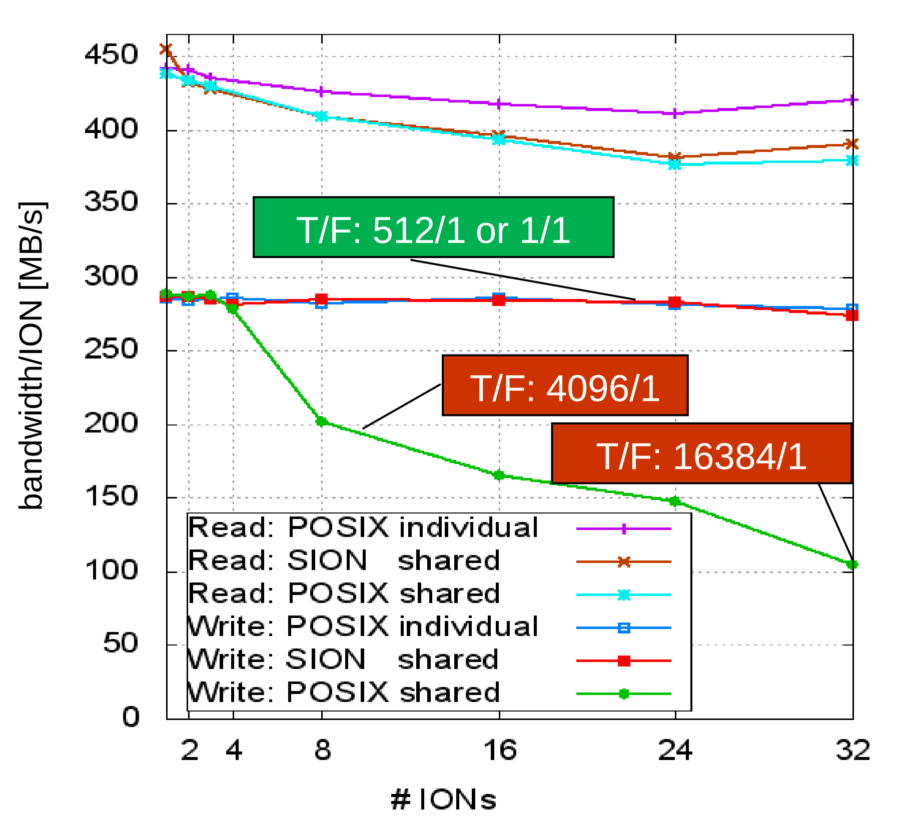


Example Blue Gene/P

- Jugene (72 racks)
- I/O forwarding nodes (ION)
- GPFS client on ION
- Solution:

  → tasks : files ratio ~ const

  → SIONlib:
  one file per ION
  implicit task-to-file mapping



Pitfall 5: Portability

- Endianess (byte order) of binary data
- Example (32 bit):

2.712.847.316

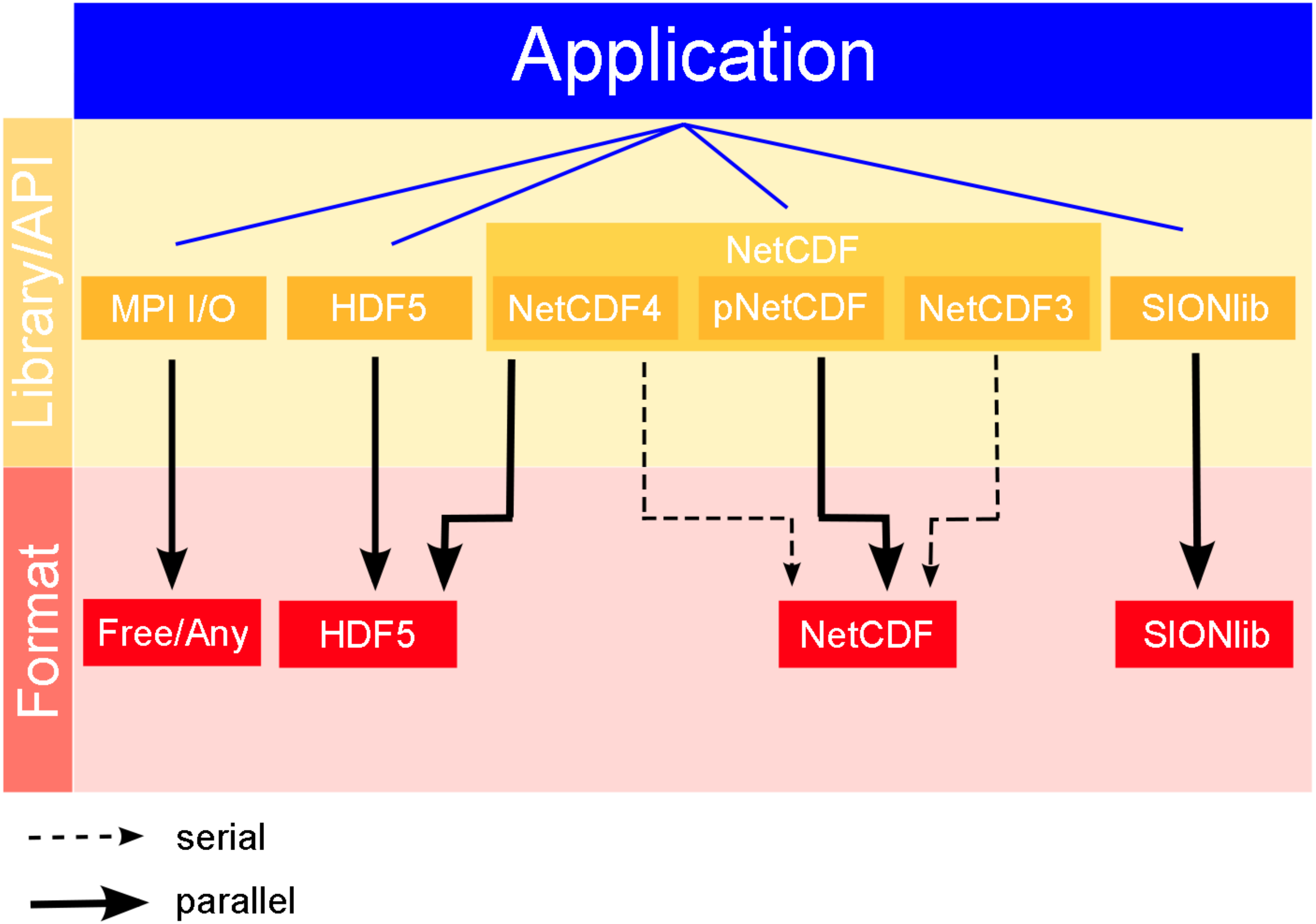
=

10100001 10110010 11000011 11010100

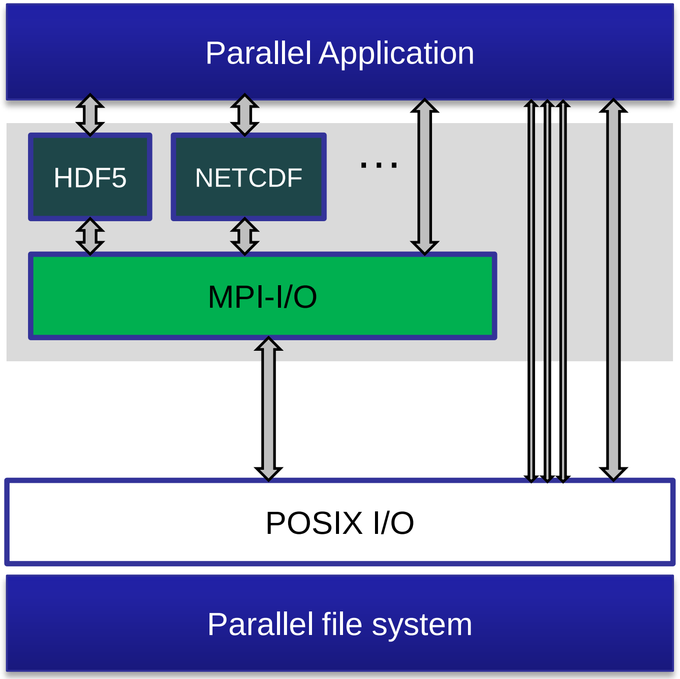
Address	Little Endian	Big Endian
1000	**11010100**	**10100001**
1001	**11000011**	**10110010**
1002	**10110010**	**11000011**
1003	**10100001**	**11010100**

- Conversion of files might be necessary and expensive
- Solution: Choosing a portable data format (HDF5, NetCDF)

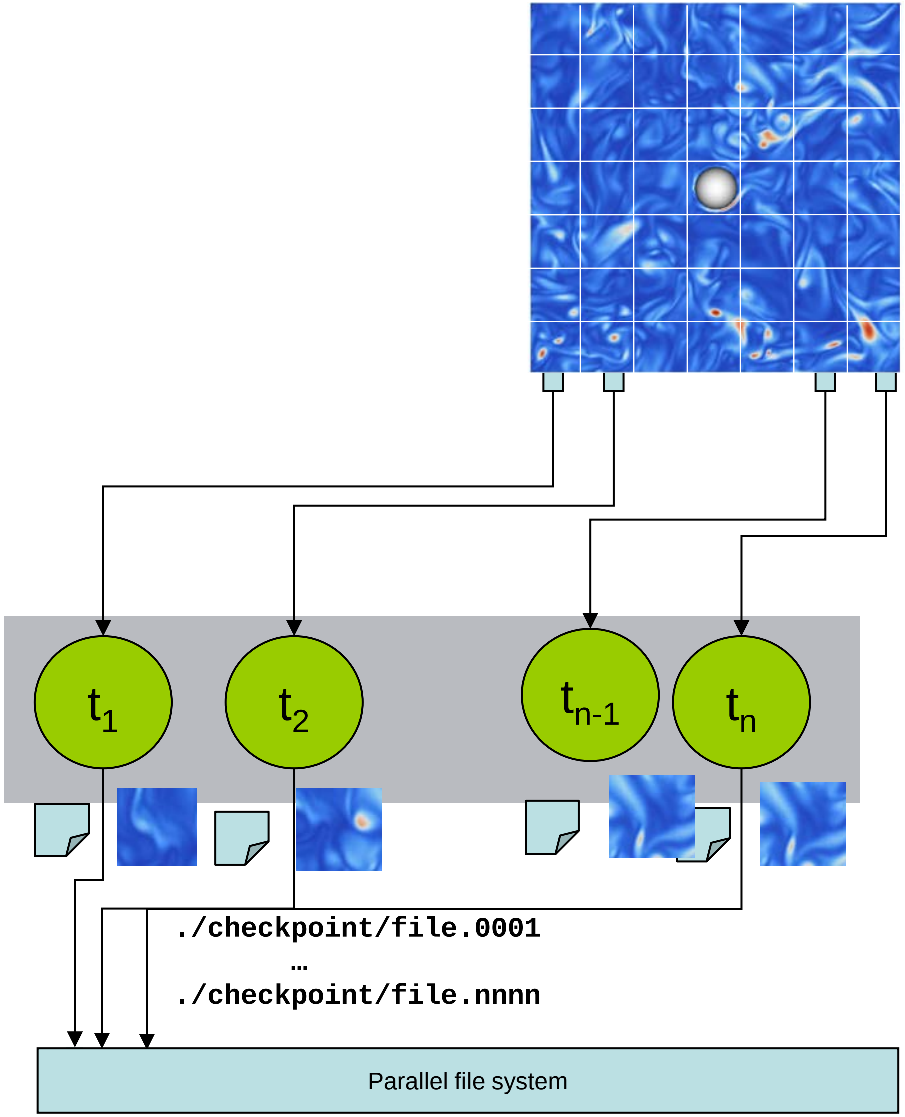
Parallel I/O Libraries: APIs + Formats



SIONlib: Shared Files for Task-local Data (I)

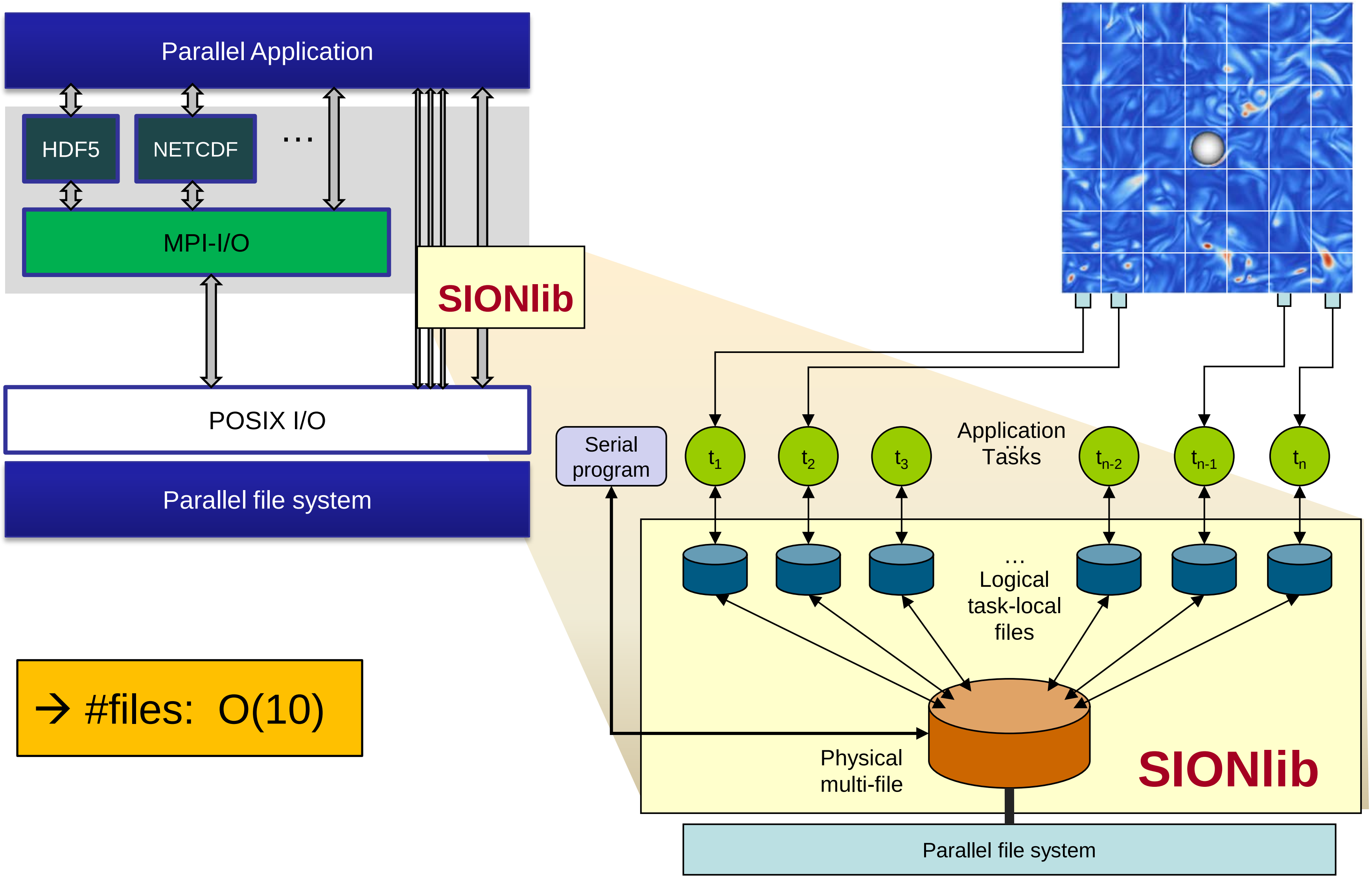


→ #files: O(10)



./checkpoint/file.0001
…
./checkpoint/file.nnnn

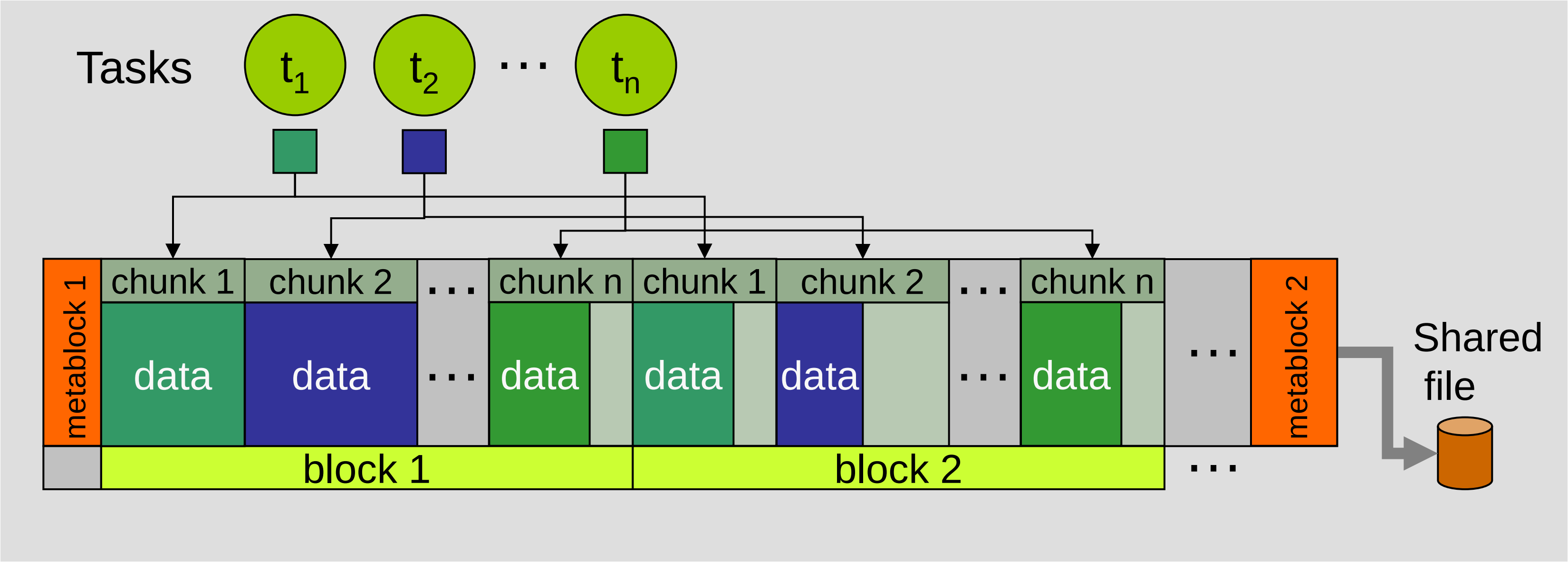
SIONlib: Shared Files for Task-local Data (II)



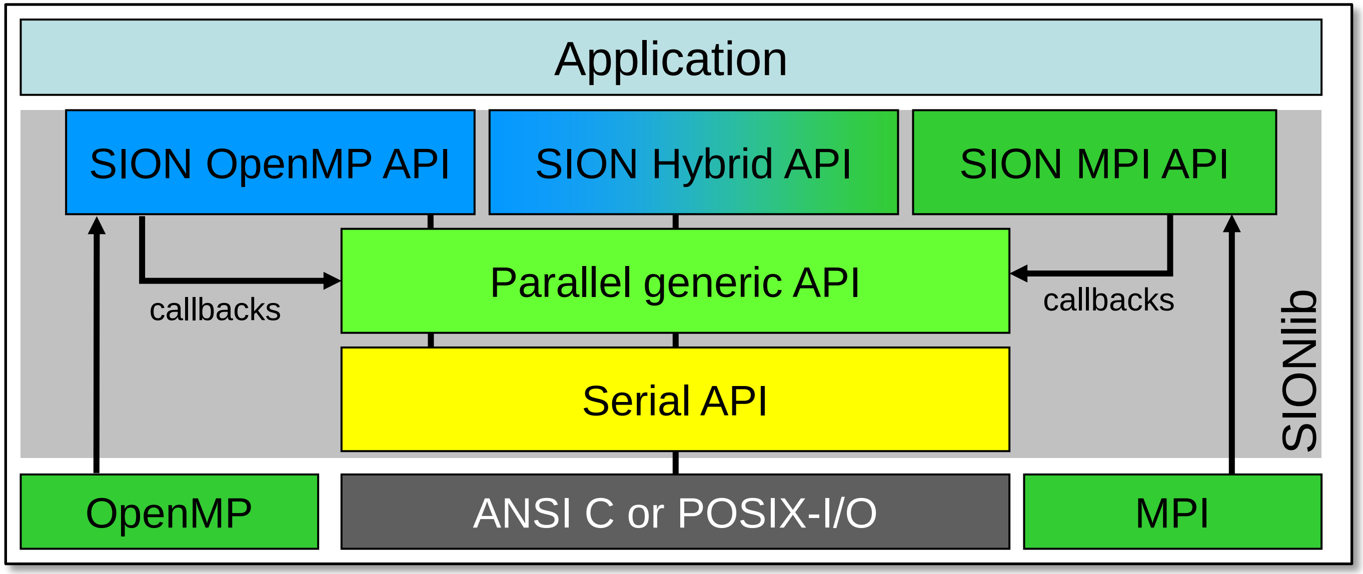
→ #files: O(10)

SIONlib: Overview & File Format

- Self-describing container format for task-local binary data
- Meta data handling (offset and data size, big/little endian, …)
- Multiple chunks per task
- Automatic alignment to file system blocks
- Transparent support of **multiple physical files** **(e.g. per ION)**
- File coalescing: support for collective I/O



SIONlib: Architecture & Example



- Extension of ANSI C-API
- C and Fortran bindings, implementation language C
- Current version: 1.4p3
- JUQUEEN: `module load sionlib`
- Open source license: http://www.fz-juelich.de/jsc/sionlib

```
/* fopen() → */
sid=sion_paropen_mpi( filename , “bw“,
                      &numfiles, &chunksize,
                      gcom, &lcom, &fileptr, ...);

/* fwrite(bindata,1,nbytes, fileptr) → */
sion_fwrite(bindata,1,nbytes, sid);

/* fclose() → */
sion_parclose_mpi(sid)
```

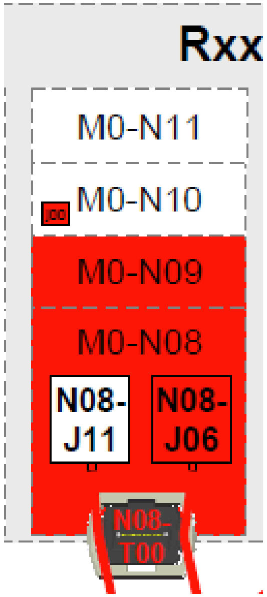
BGQ: Tasks connected to same I/O-node

- MPIX_Calls now available on BG/Q
  (see http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/MPIextensions.html)
- Communicator: All tasks belonging to same I/O Bridge Node

```
FORTRAN: MPIX_PSET_SAME_COMM_CREATE(INTEGER pset_comm_same,
                                    INTEGER ierr)
C: #include <mpix.h>
   int MPIX_Pset_same_comm_create( MPI_Comm *pset_comm_same )
```

- Usage: implementation of own I/O strategy
  (One file per I/O-bridge)
- Passing new communicator to SIONlib paropen-Call (as local communicator)

```
…
sid=sion_paropen_mpi( filename , “bw“,
                      &numfiles, &chunksize,
                      gcom, &lcom, &fileptr, ...);
...
```



Darshan – I/O Characterization

- Darshan: Scalable HPC I/O characterization tool (ANL)
  http://www.mcs.anl.gov/darshan

- Profiling of I/O-Calls (POSIX, MPI-I/O, HDF5, NetCDF) during runtime
- Replaces Compiler-Calls (mpi*xxx)* by Darshan wrappers:
  - Re-link application, Re-run application → logfile
- Generate report from logfile:
  `darshan-job-summary <logfile>` → PDF-file
- On JUQUEEN:
  - **`module load darshan`**
    → version 2.2.8 (patched for JUQUEEN)
  - Report Path: set by environment variable DARSHANLOGDIR
    e.g. **`runjob … --envs DARSHANLOGDIR=$HOME/darshanlog`**
  - Viewer on Frontend node: **`evince <pdf-file>`**

How to choose an I/O strategy?

- Performance considerations
  - Amount of data
  - Frequency of reading/writing
  - Scalability

- Portability
  - Different HPC architectures
  - Data exchange with others
  - Long-term storage

- E.g. use two formats and converters:
  - Internal: Write/read data “as-is”
    → *Restart/checkpoint files*
  - External: Write/read data in non-decomposed format
    (portable, system-independent, self-describing)
    → *Workflows, Pre-, Postprocessing, Data exchange, …*

Summary

- Application I/O has to exploit **parallelism** to make use of the full available bandwidth of HPC I/O systems
- Fast internal I/O with special data formats and I/O libraries
- Portable data formats are needed to efficiently process data in heterogeneous environments
- Multiple solutions to portable parallel I/O are available

- Training Course: **Parallel I/O and Portable Data Formats**
  21 May to 23 May 2014
  *http://www.fz-juelich.de/ *
  *SharedDocs/Termine/IAS/JSC/DE/ *
  Kurse/2014/parallelio-2014.html