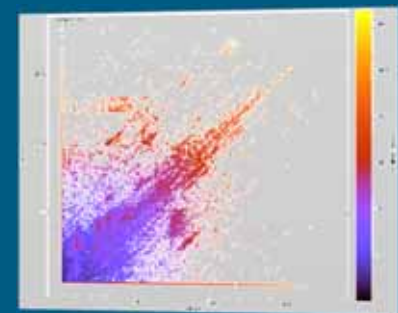


I/O Monitoring at JSC, SIONlib & Resiliency

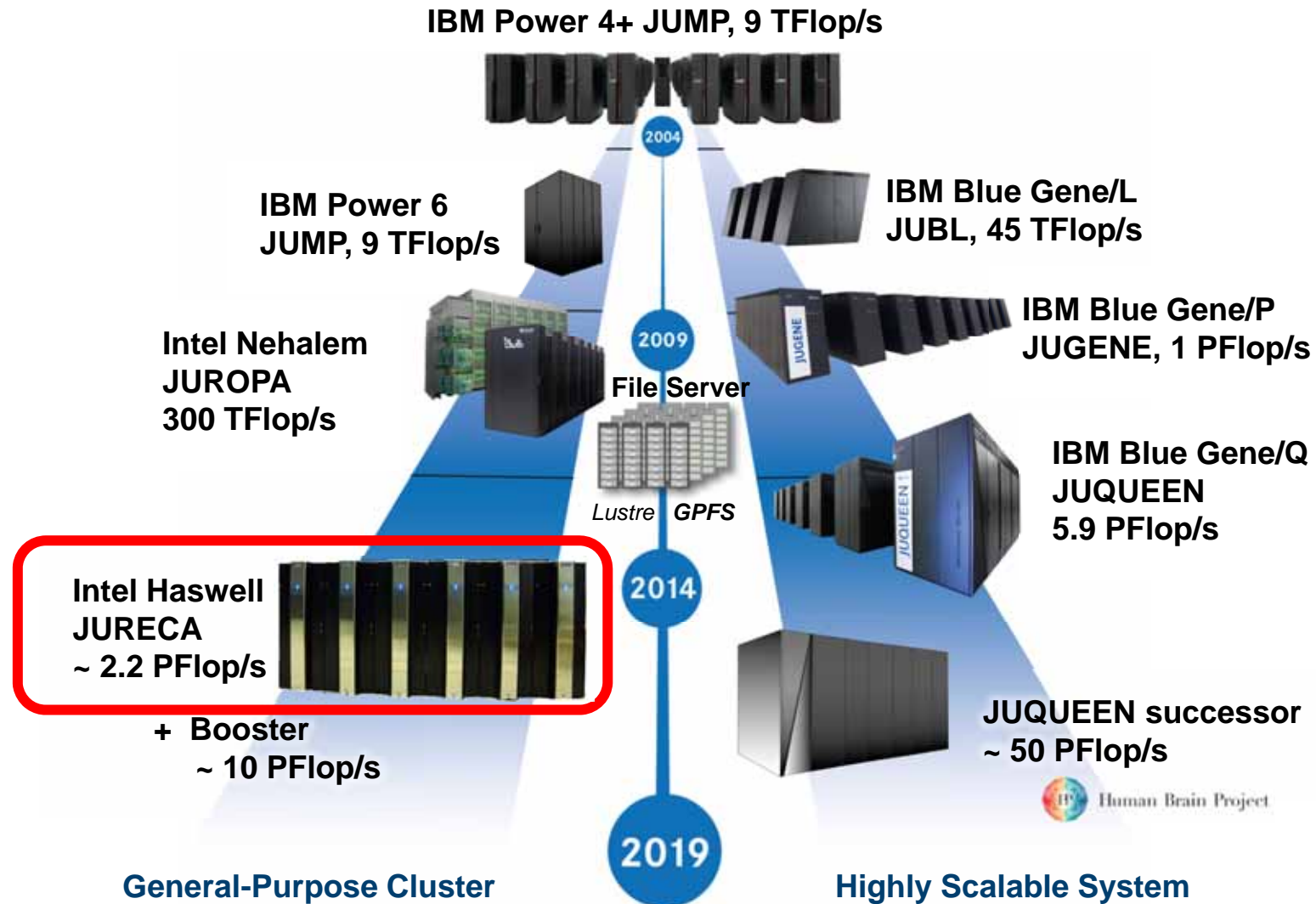
- Update: I/O Infrastructure @ JSC
- Update: Monitoring with LLview (I/O, Memory, Load)
- I/O Workloads on Jureca
- SIONlib: Task-Local I/O & Buddy Checkpointing

Wolfgang Frings, Kay Thust
[f{W.Frings,K.Thust}@fz-juelich.de](mailto:{W.Frings,K.Thust}@fz-juelich.de)
Jülich Supercomputing Centre

HPC I/O in the Data Center (HPC-IODC), ISC, Frankfurt, June 23th, 2016

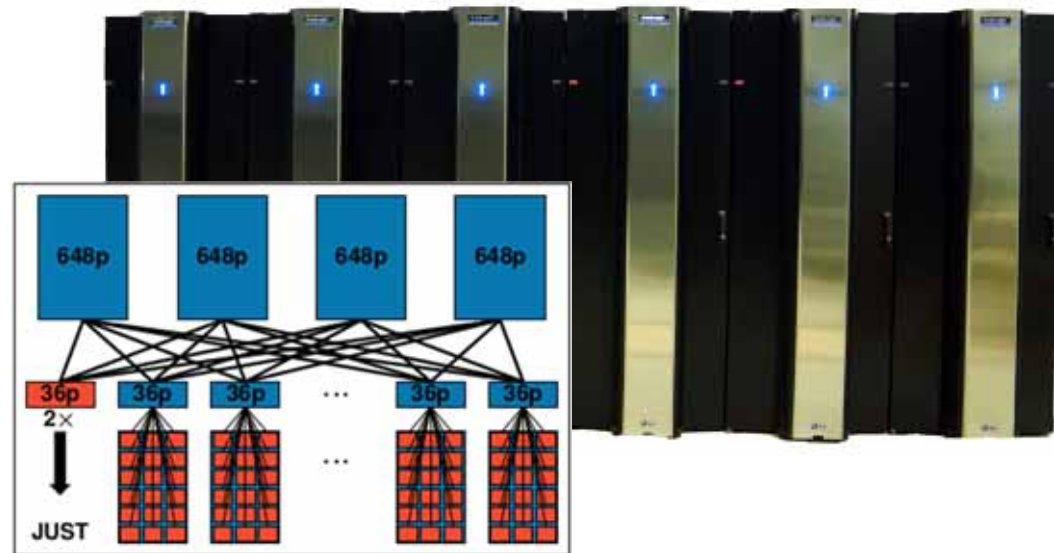


HPC Systems @ JSC: Dual Architecture Strategy



JURECA: Jülich Research on Exascale Cluster Architectures

- 2 Intel Haswell 12-core processors, 2.5 GHz, SMT, 128 GB main memory
- **1,884 compute nodes** or 45,216 cores, thereof
 - 75 nodes with 2 K80 NVIDIA graphics cards each and
 - 12 nodes with 512 GB main memory and 2 K40 NVIDIA graphics cards each for visualization
- 2.245 Petaflop/s peak (with K80 graphics cards)
- 1.425 Petaflop/s Linpack from CPUs (out of 1,693 Petaflop/s peak)
- **281 TByte memory**
- Mellanox Infiniband EDR
- Connected to the **GPFS** file system on **JUST** via IB FDR/40GigE gateway switches



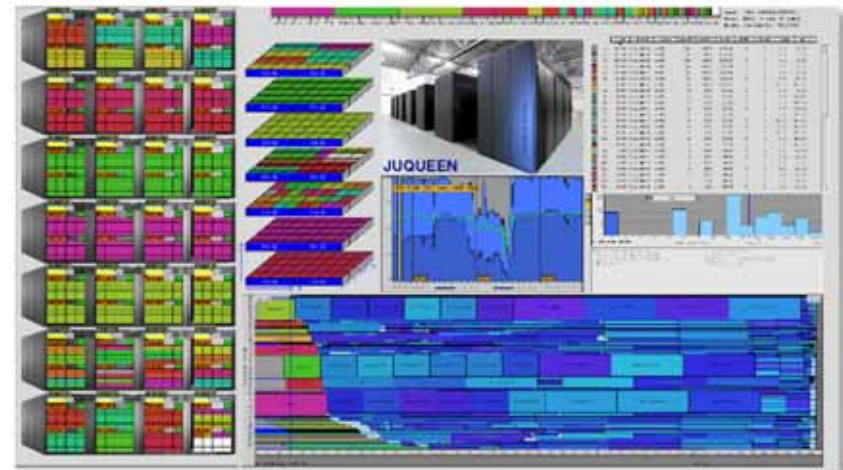
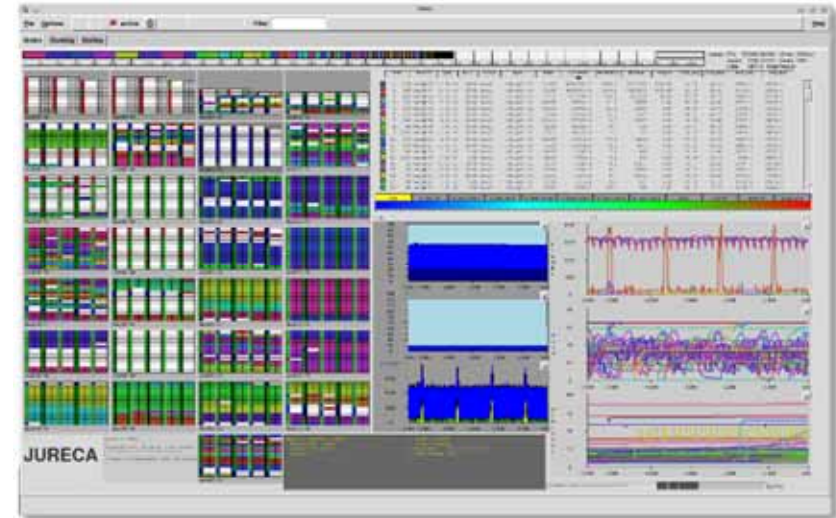
Parallel I/O Hardware at JSC (Just4, GSS)

- **Juelich Storage Cluster (JUST)**
 - GPFS Storage Server (GSS/ESS)
 - End-to-End integrity
 - Fast rebuild time on disk replacement
 - GPFS + TSM Backup + HSM
 - → JUQUEEN, JURECA, ...
- **Just4-GSS**
 - Capacity: **20.3 Pbyte**
I/O Bandwidth: up to **220 GB/sec**
 - Hardware: IBM System x® GPFS™
Storage Server solution, GPFS Native
RAID
 - 31 Building blocks: each 2 x X3650 M4
server, 232 NL-SAS disks (2TB), 6 SSD



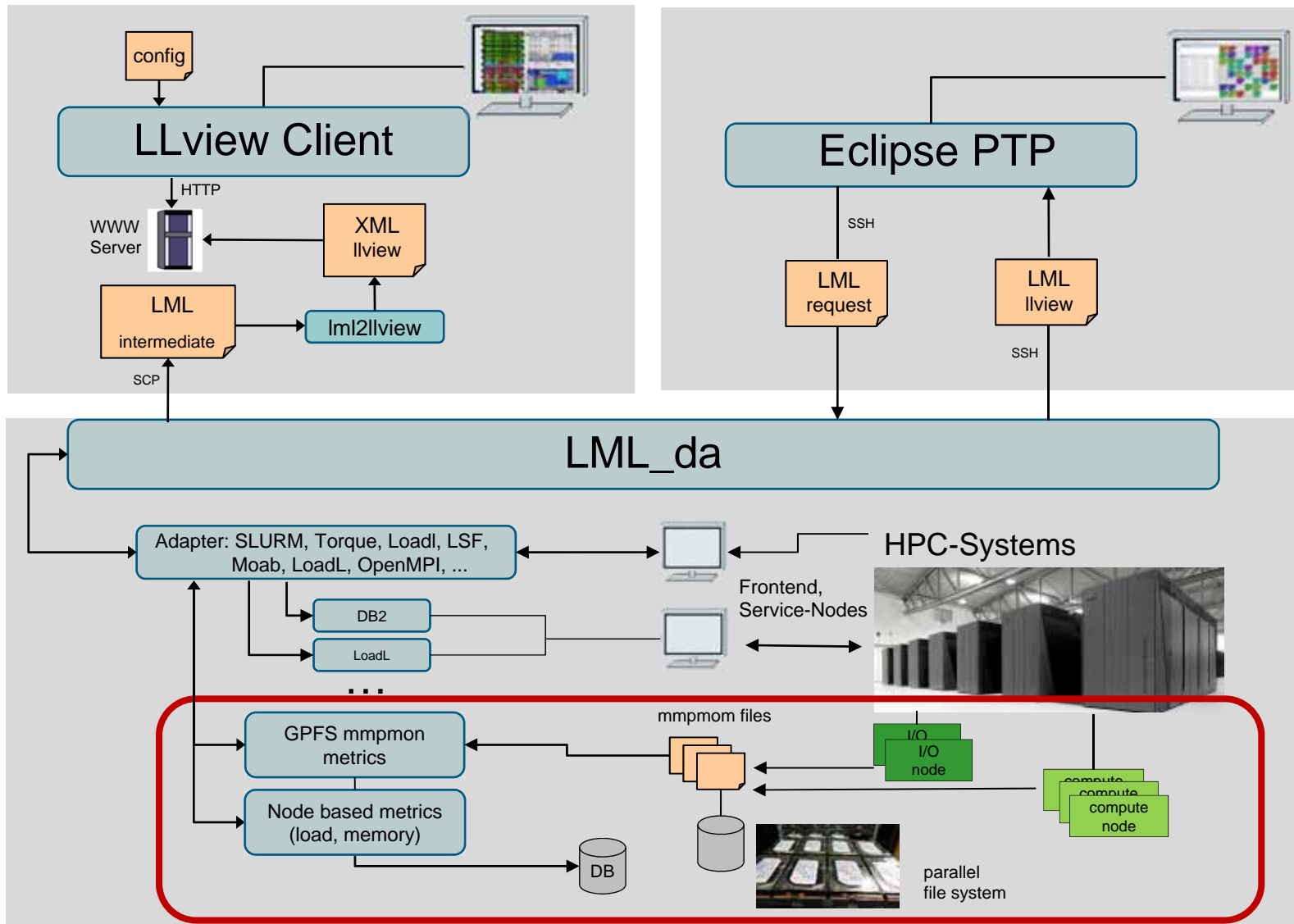
LLview: User-level Monitoring

- Efficient supervision of node usage, running jobs, statistics, history
- Prediction of system usage
- Monitoring of energy consumption, load, memory usage, I/O usage
- Interactive and mouse-sensitive
- Main data source: batch scheduler, runtime system
- No interaction with compute nodes
- Fully customizable, fast and portable client-server application
- Integrated into Eclipse/PTP
- Support for various resource manager, incl. LoadLeveler, IBM Blue Gene, Cray ALPS, PBSpro, Torque, SLURM, Grid Engine and LSF

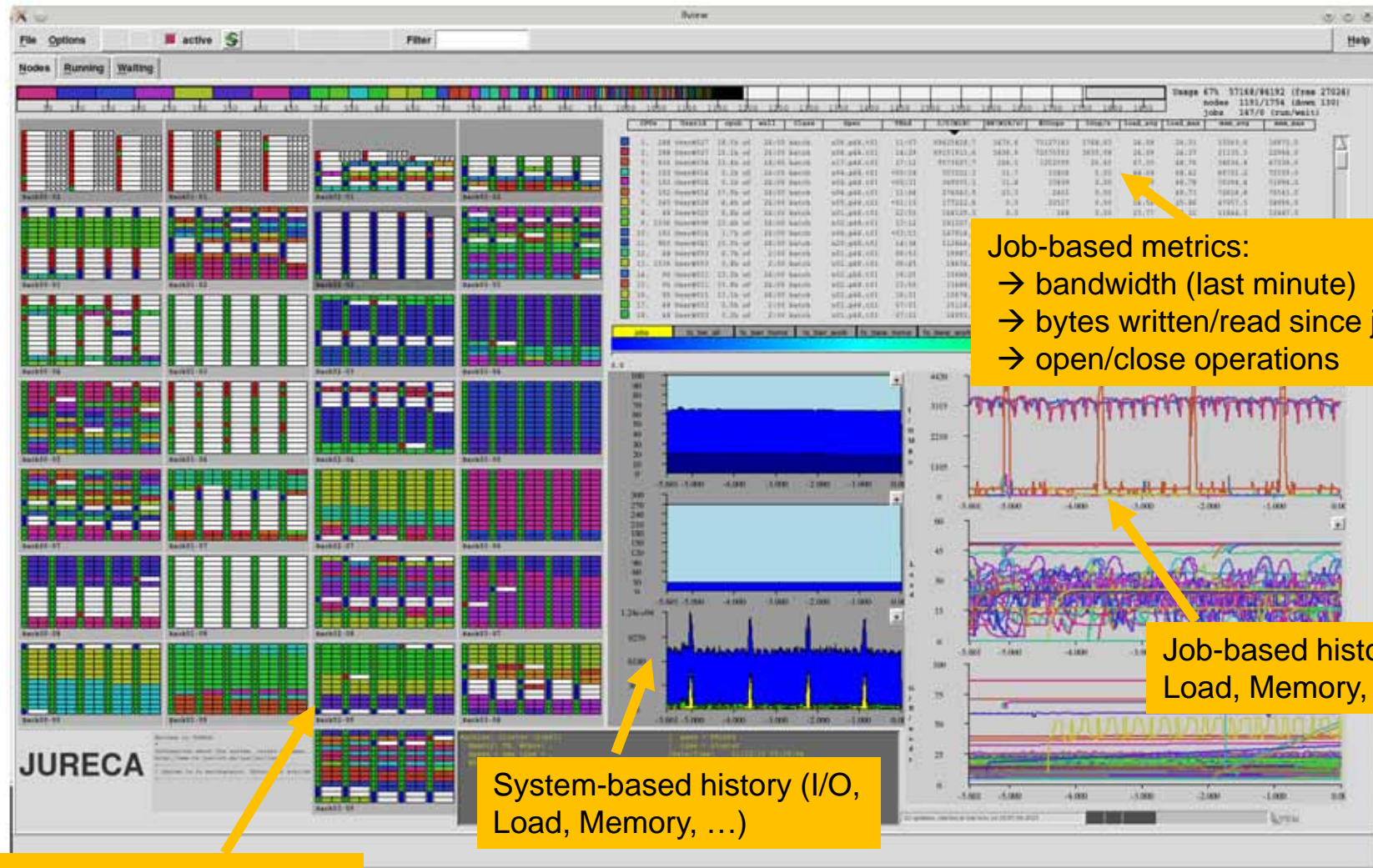


LLview download: (open source)
<http://www.fz-juelich.de/jsc/llview>

LLview Architecture & I/O monitoring



LLview: Node & File System Metrics



Job-based metrics:
 → bandwidth (last minute)
 → bytes written/read since job start
 → open/close operations

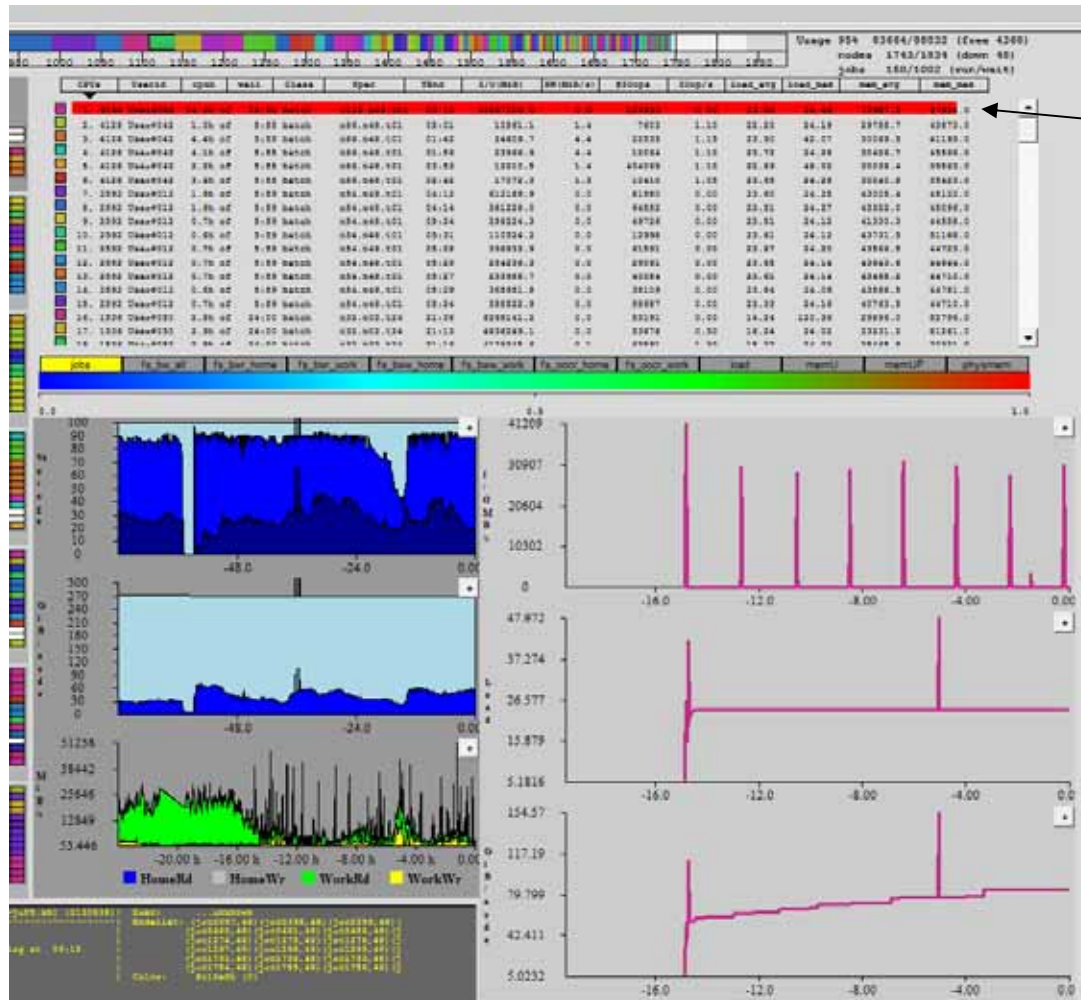
Job-based history (I/O, Load, Memory, ...)

System-based history (I/O, Load, Memory, ...)

Node-based mapping of I/O load (color-coded)

LLview: Node & File System Metrics

- Job-based monitoring



selected job:
 - using 86 nodes
 - transferring ~ 31 TiB

I/O activity of application

avg. load per node

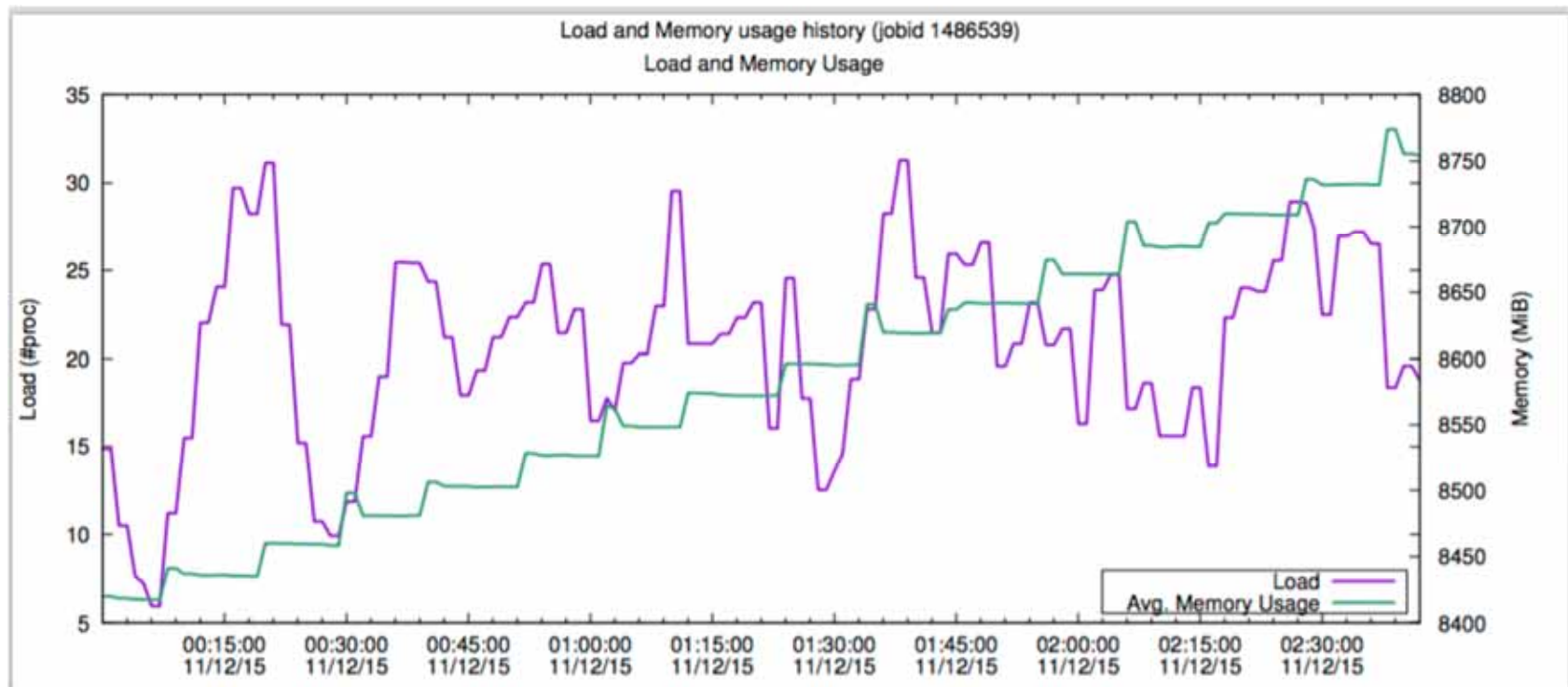
avg. memory usage per node

LLview: Offline Analysis

Accumulated Information – load & mem

Slight increase over time -> memory leak?

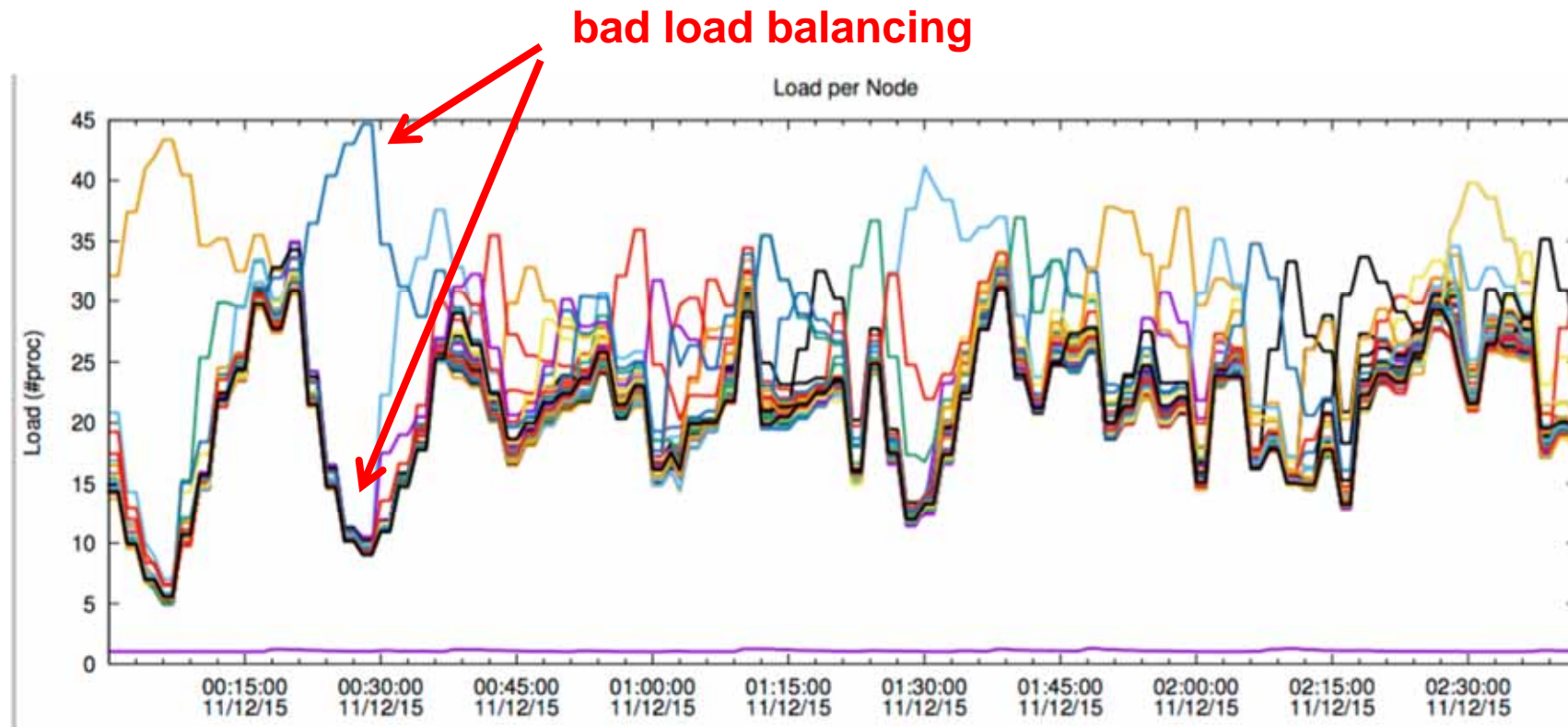
Load fluctuates -> why?



ODE for ~20.000 components , rhs distributed over 64 nodes

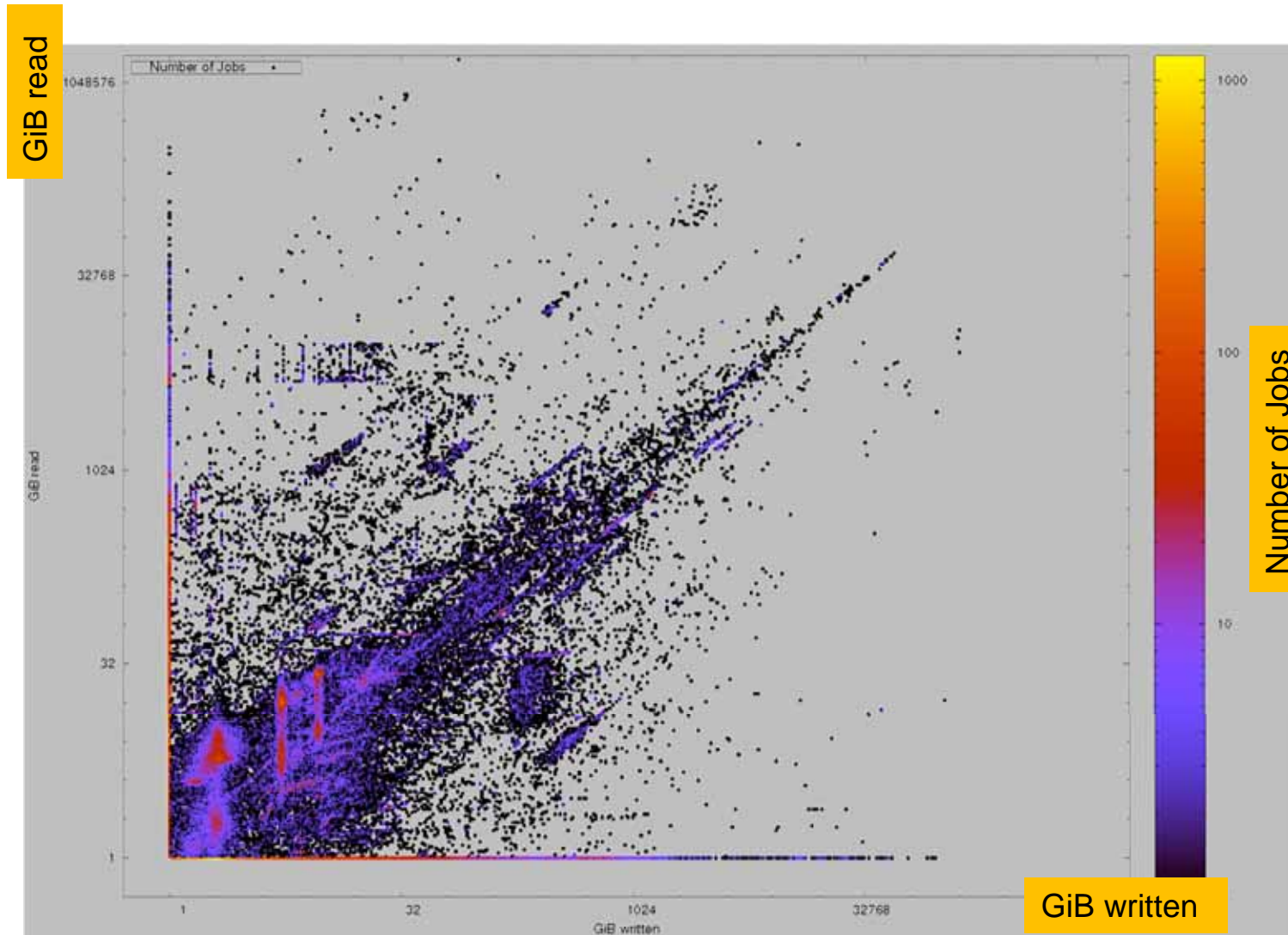
LLview: Offline Analysis

Node-specific information - load



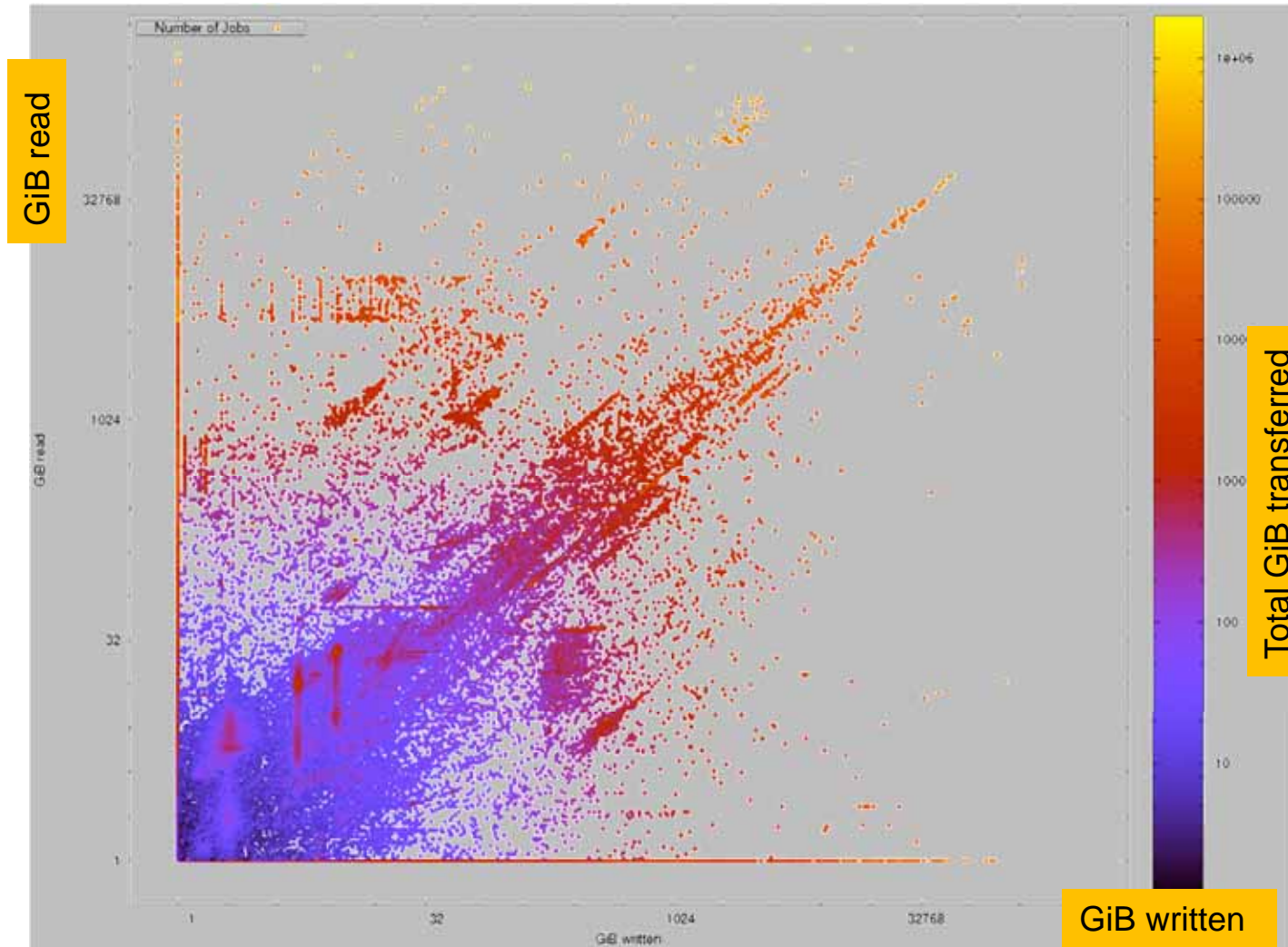
ODE for ~20.000 components , rhs distributed over 64 nodes

I/O Workload Analysis: number of jobs



Example: number of jobs

I/O Workload Analysis: transfer size

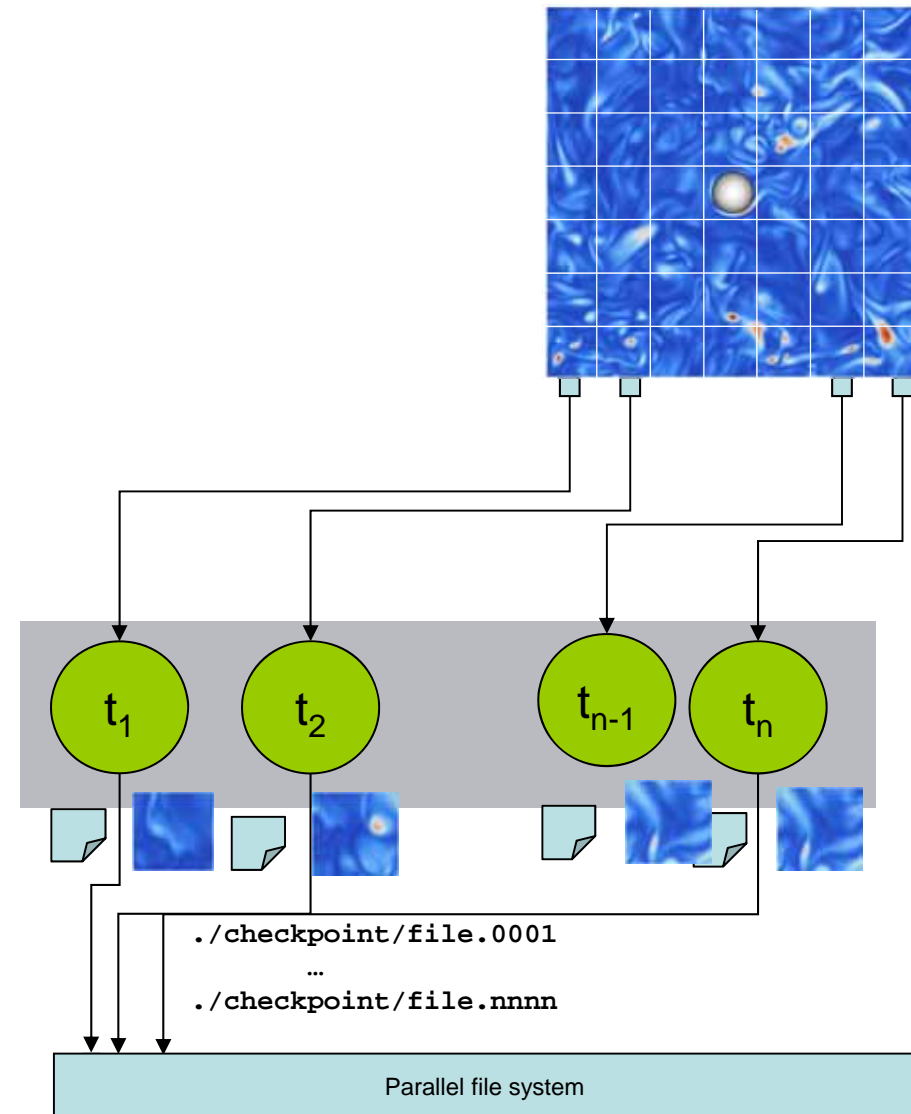
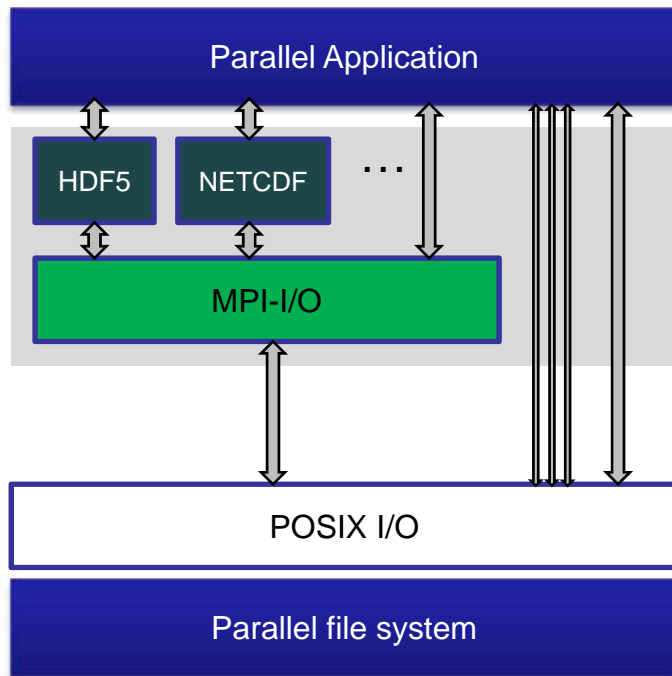


Example: transfer size

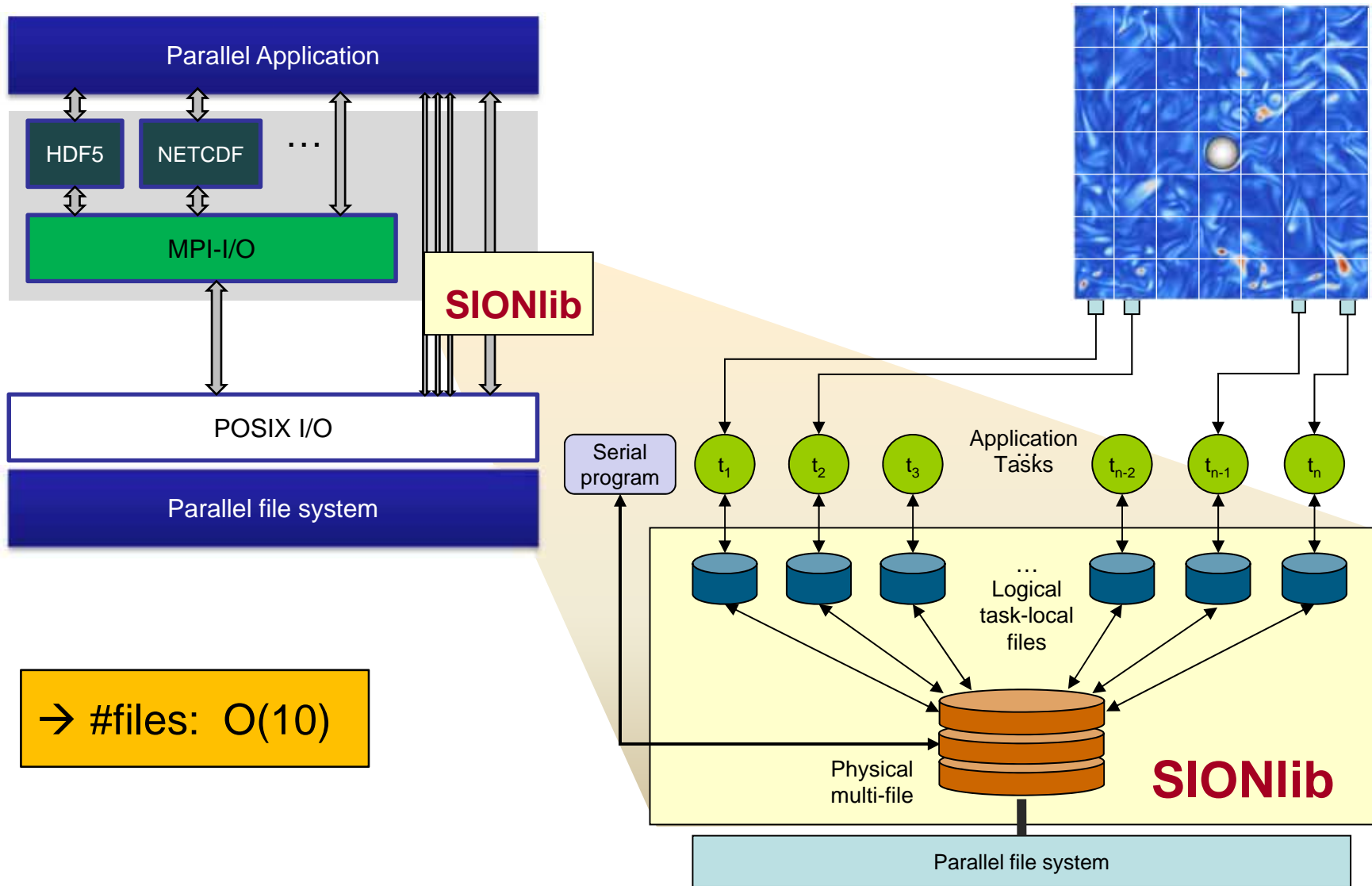
I/O Workload Analysis: Next steps

- Classification (e.g. research topic, group, account)
- Additional metrics
 - Bandwidth
 - Open/Close calls
 - Type of I/O: continuous, burst, ..
 - Parallel I/O or one writer/reader, ...
 - More?
- Report at job end, containing information and timelines for
 - I/O activity
 - Memory usage
 - Load

SIONlib: Shared Files for Task-local Data

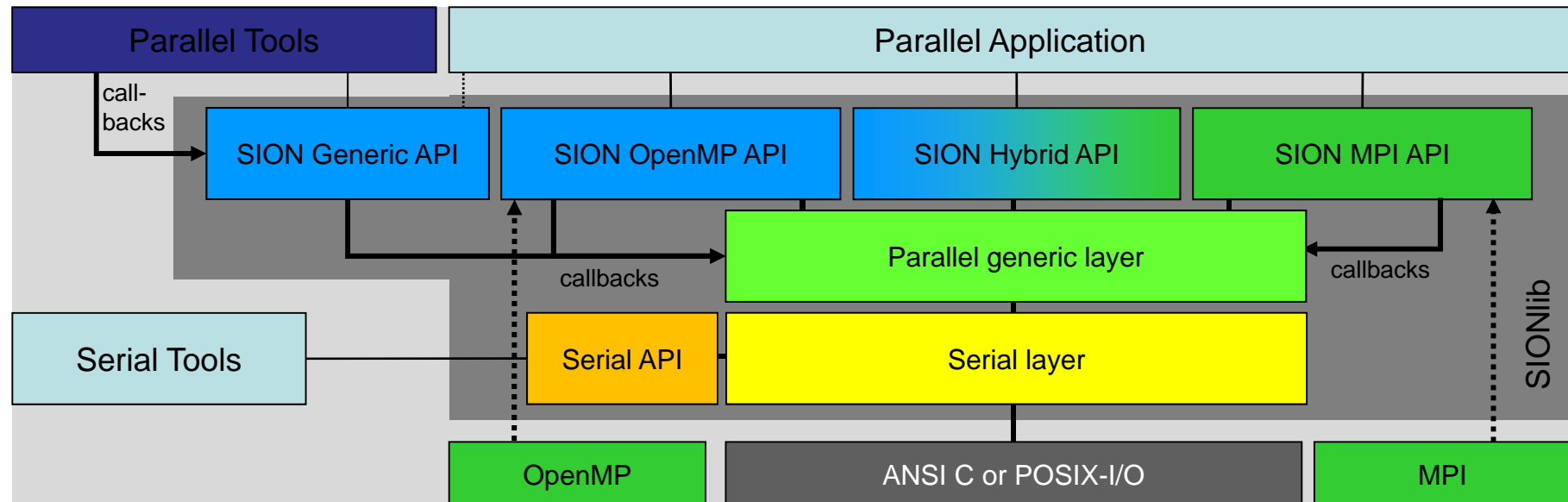


SIONlib: Shared Files for Task-local Data



→ #files: $O(10)$

SIONlib: Architecture & Example



- Extension of I/O-API (ANSI C or POSIX)
- C and Fortran bindings, implementation language C
- Current versions: 1.6.2
- 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)

```


DEEP-ER: The Project

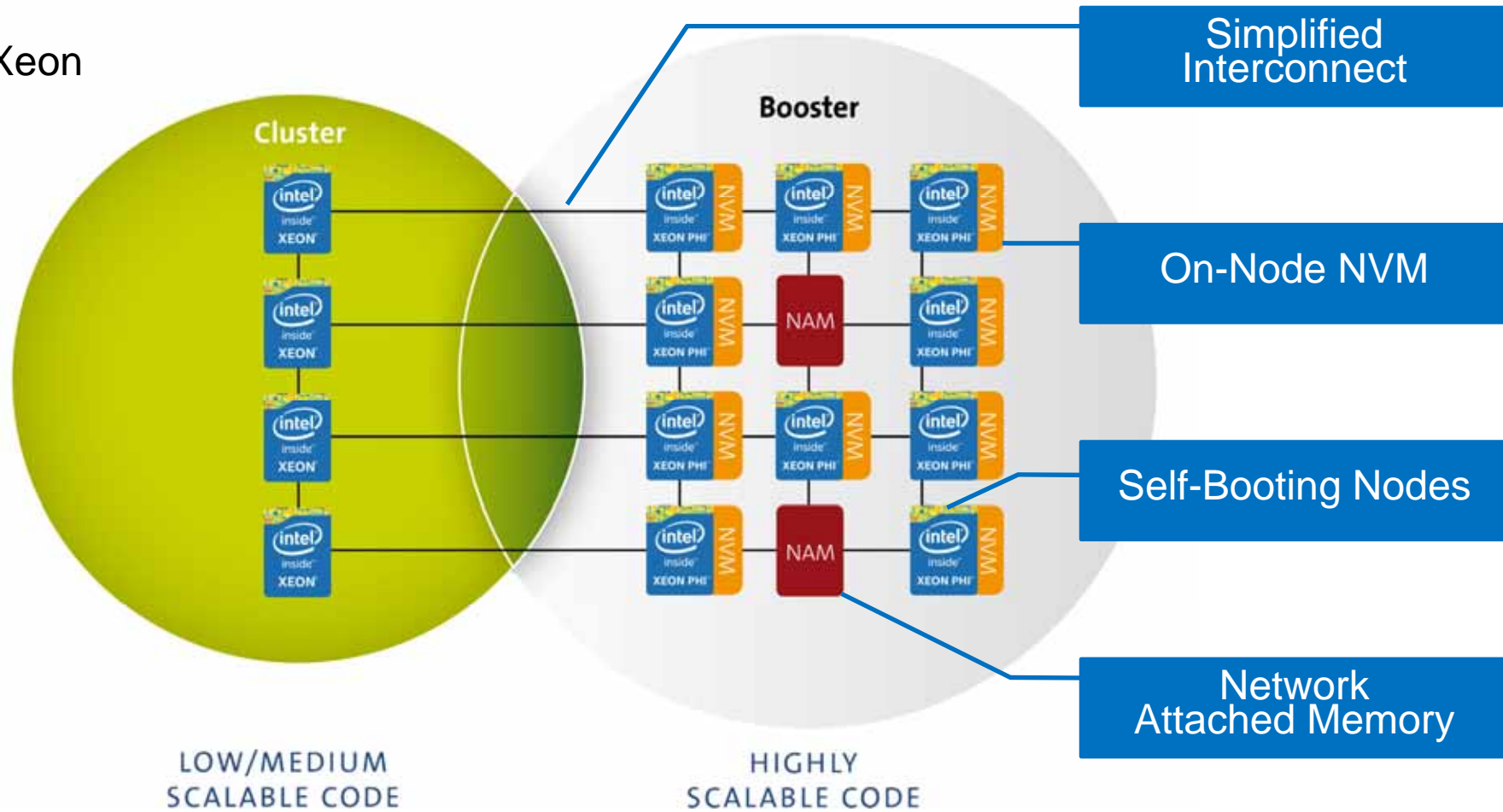
- **DEEP – Extended Reach:** EU-funded Exascale research project
- **Budget:** 10M €
 - ✓ EU-funding: 6,4 M €
 - ✓ Start: Oct'13
 - ✓ Duration: 42 months
- **The DEEP-ER Consortium**
 - ✓ Coordinator: JSC*
 - ✓ 14 Partners
 - 3 PRACE hosting members
 - 4 industry partners
 - 7 European countries



* Jülich Supercomputing Centre

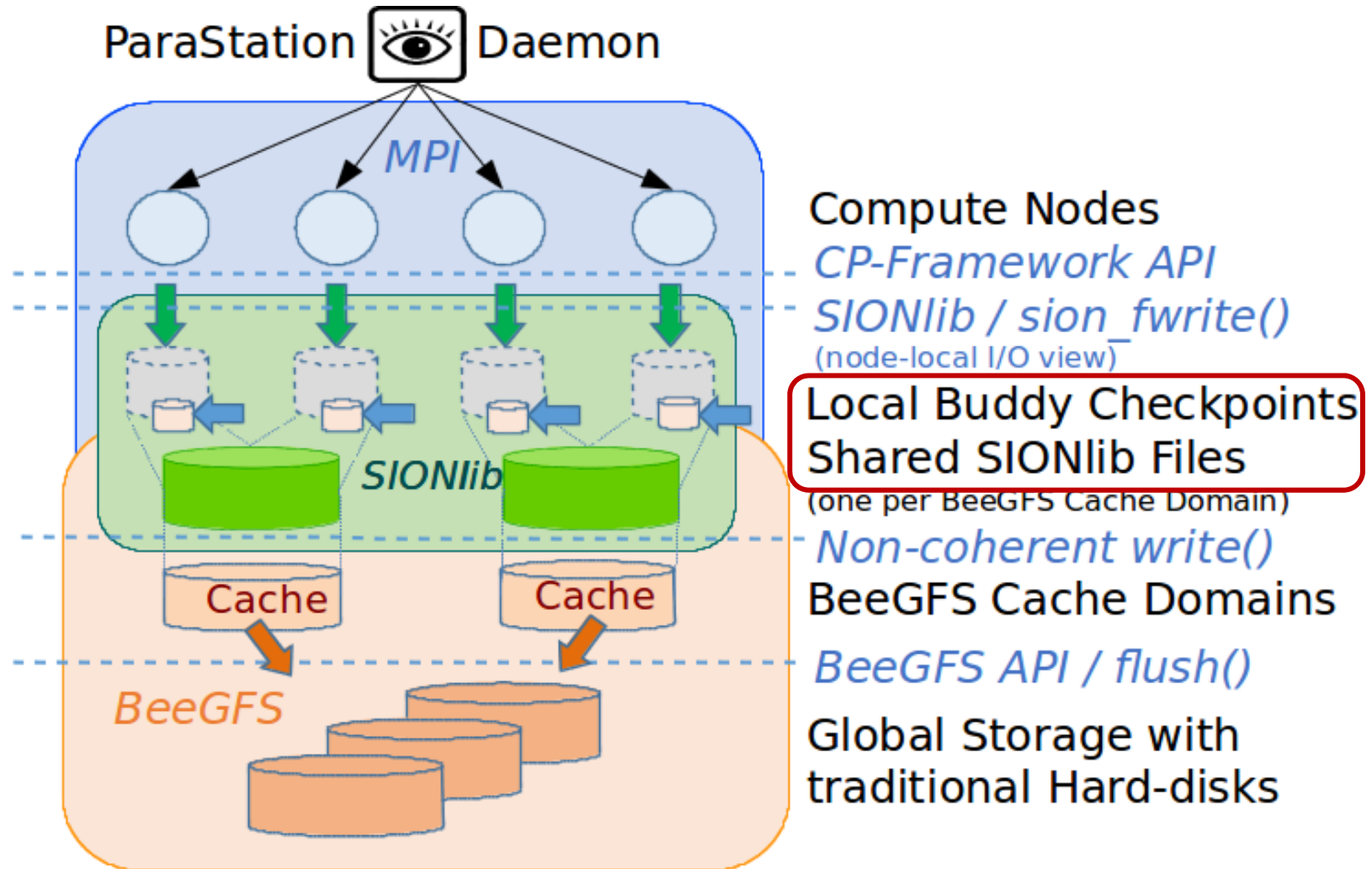
DEEP-ER: Architecture Innovation

Xeon



DEEP-ER: Resiliency

Integration SCR/SIONlib/BeeGFS



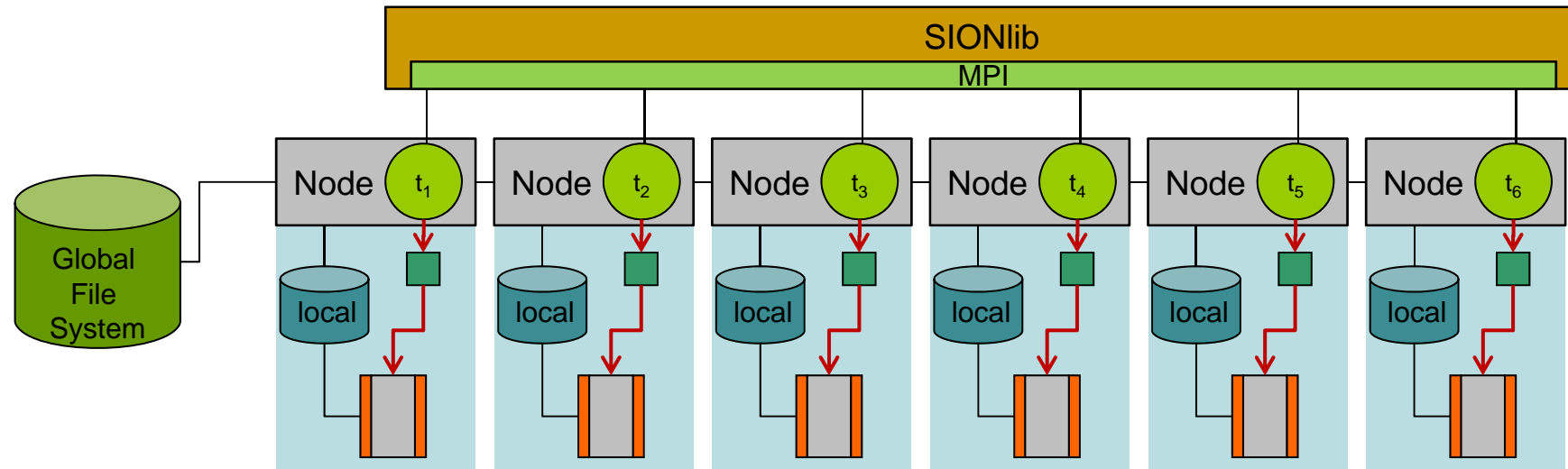
Compute Nodes
CP-Framework API
SIONlib / sion_fwrite()
 (node-local I/O view)

Local Buddy Checkpoints
Shared SIONlib Files
 (one per BeeGFS Cache Domain)

Non-coherent write()
 BeeGFS Cache Domains

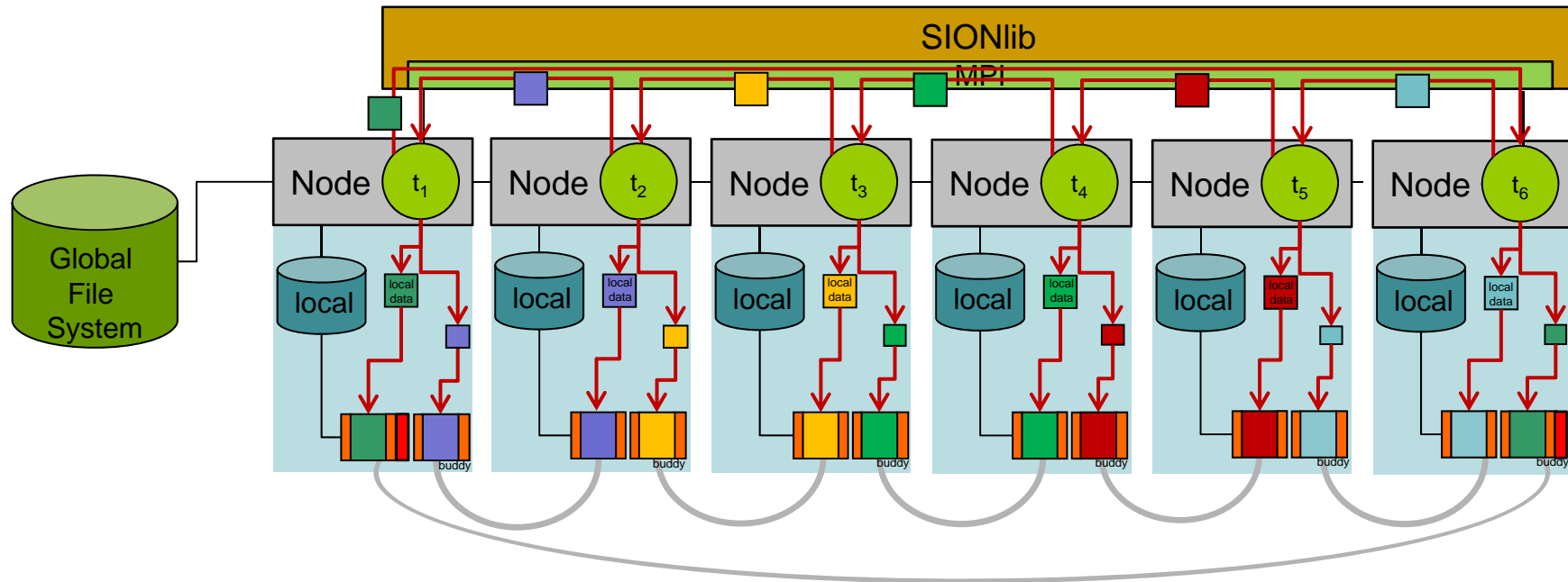
BeeGFS API / flush()
 Global Storage with traditional Hard-disks

SIONlib: Local checkpointing



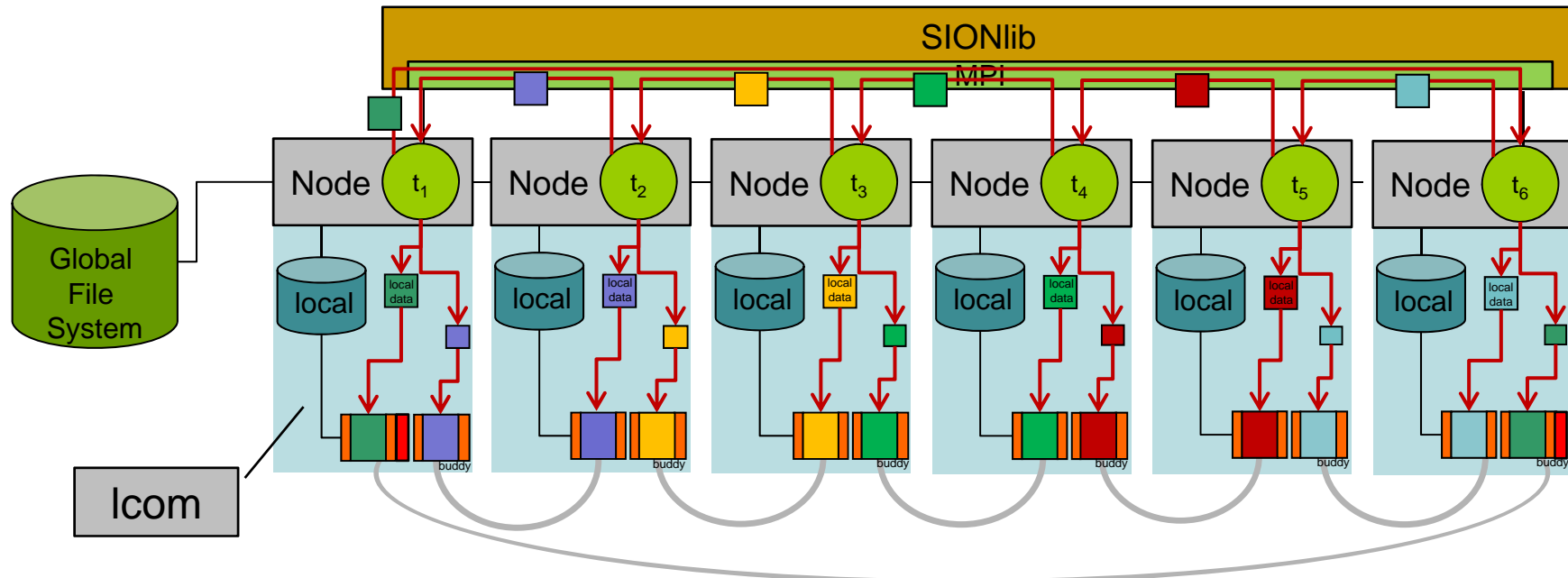
- Mapping: task writes to local storage
- Transparent access to data from application
→ when files migrated to global storage

SIONlib: Buddy-CP, Logical mapping



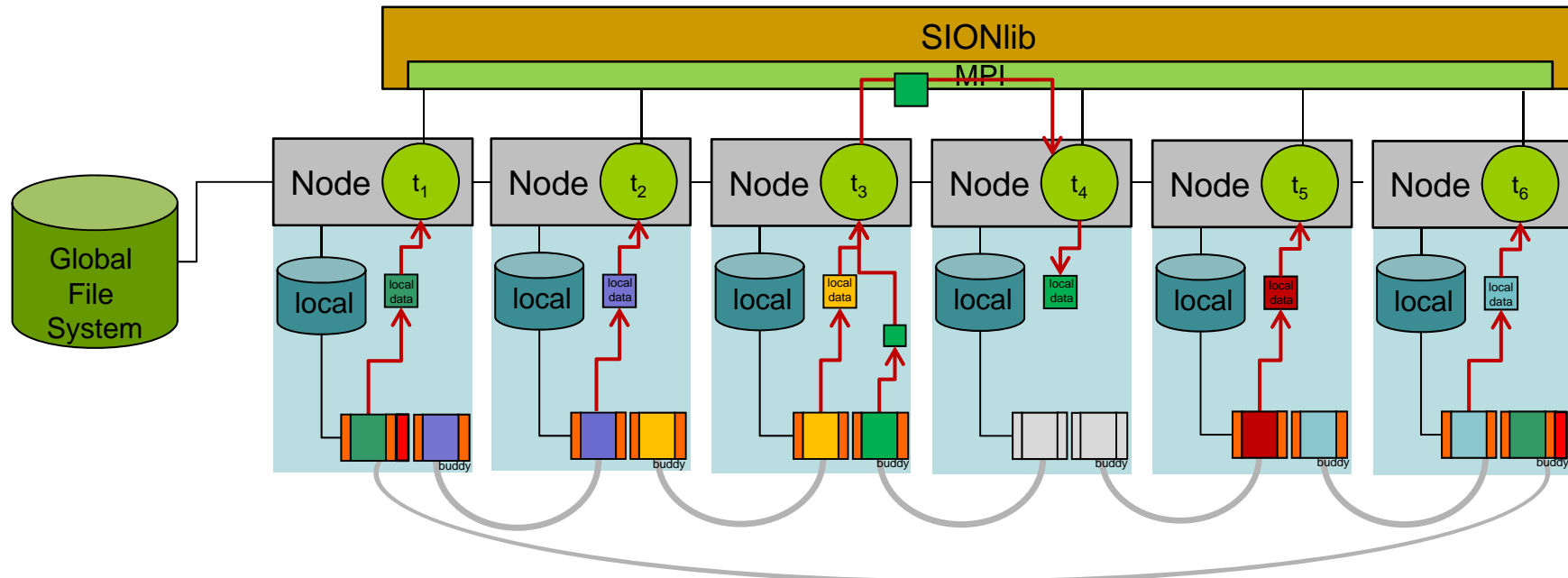
- Mapping: 1:1 task writes to local storage and storage of buddy node
1:x task writes to local storage and storage of x buddy nodes
- Data exchange to buddy node is done via SIONlib MPI/OpenMP layer
- Collective checkpoint calls required

SIONlib: Write buddy checkpoint



- Open: `sid=sion_paropen_mpi(..., "bw,buddy", MPI_COMM_WORLD, lcom, ...)`
- Write: `sion_coll_write_mpi(data, size, n, sid)`
- Close: `sion_parclose(sid)`
- Write-Call will write data first to local chunk, and then sent it to the associated buddy which writes the data to a second file

Restore checkpoint after failure



- Open: `sid=sion_paropen_mpi(..., "br,buddy", MPI_COMM_WORLD, lcom, ...)`
 - *First tries normal open and falls back to buddy data if first open fails*
- Read: `sion_coll_read_mpi(data, size, n, sid)`
- Close: `sion_parclose(sid)`

SIONlib and SCR: example

SCR_Start_checkpoint()

fn = "check1"

SCR_Route_file(fn, fn_scr)

fn_scr = "/abspath/check1"

sid = sion_paropen_mpi(fn_scr, "wb,buddy" ...)

(node0) "/abspath/check1"

(node1) "/abspath/check1.00001"

...

(node0) "/abspath/check1_BUDDY_00.00001"

(node1) "/abspath/check1_BUDDY_00.00002"

...

info = sion_get_io_info(sid)

- List of filename opened on this task
- Bytes written

sion_parclose_mpi(sid)

SCR_update_filename(nfiles, info.names, info.sizes, info.roles)

SCR_Complete_checkpoint()

Conclusion

I/O monitoring, combination of information from different sources

- LLview + GPFS mmpmon
- On-line and off-line analysis

- I/O Workload analysis
 - concepts for analysis of job-based mmpmon data
 - Automatic job reports after job end
- DEEP-ER project: SIONlib support for resiliency
 - Multi-version buddy checkpointing
 - Support for multi-level checkpointing with SCR