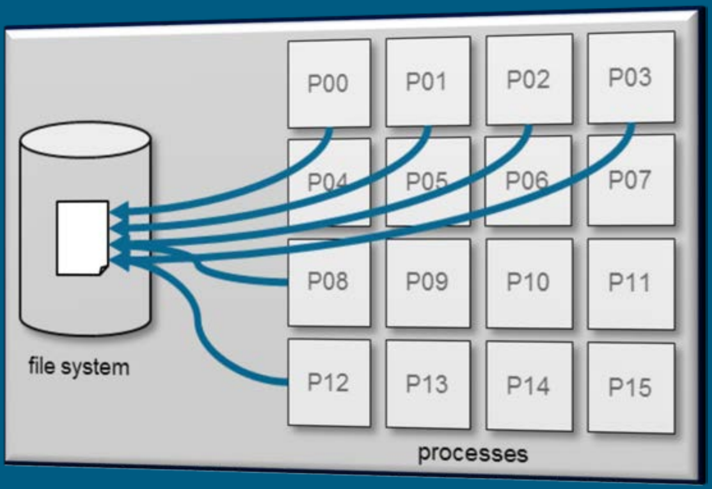
# Parallel I/O

## I/O strategies

Wolfgang Frings, Sebastian Lührs
w.frings@fz-juelich.de
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH

Jülich, June 7th, 2016

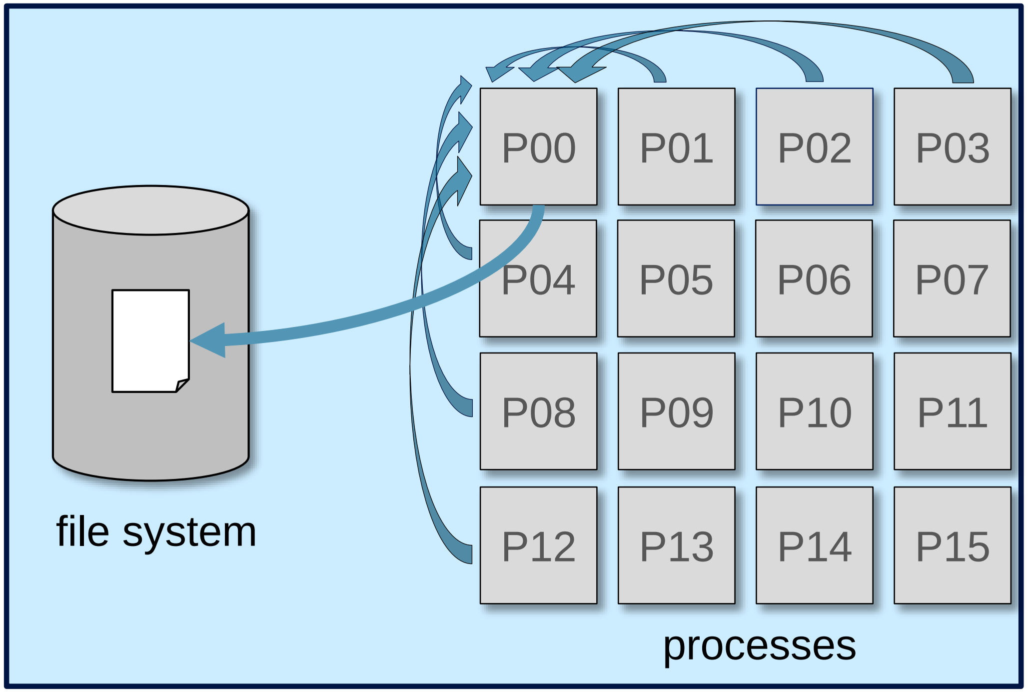


# Outline

- Common I/O strategies
- I/O workflow
- Pitfalls
- Parallel I/O software stack
- I/O on Jureca
- Application I/O performance information with Darshan

# One process performs I/O

- \+ Simple to implement
- \- I/O bandwidth is limited to the rate of this single process
- \- Additional communication might be necessary
- \- Other processes may idle and waste computing resources during I/O time



# Frequent flushing on small blocks

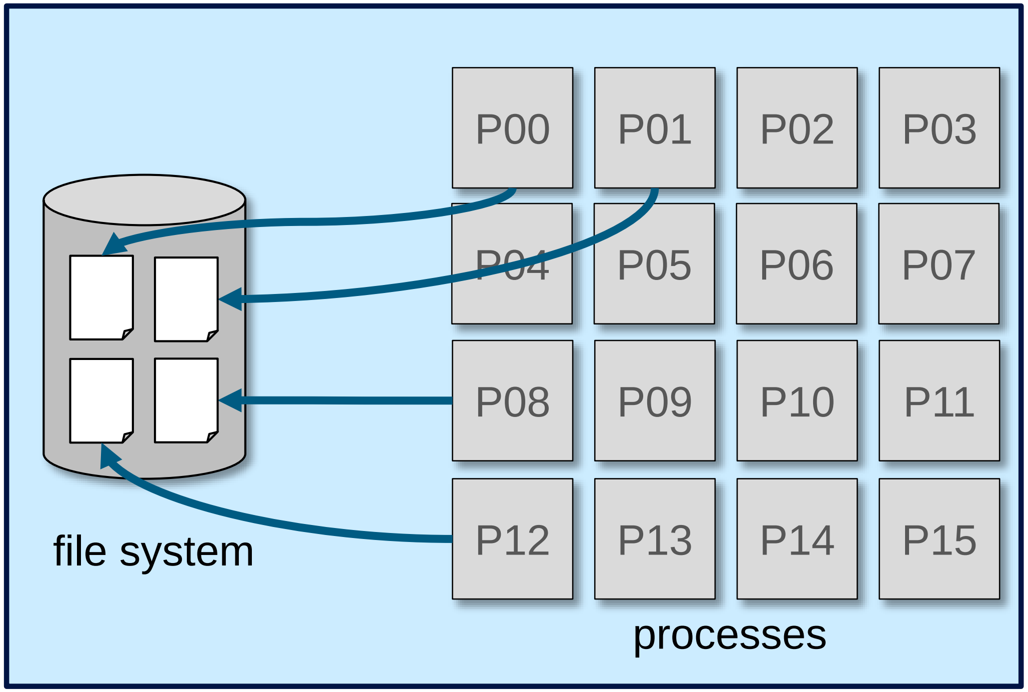
Pitfall 1

- Modern file systems in HPC have **large file system blocks** (e.g. 4MB)
- 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

# Task-local files



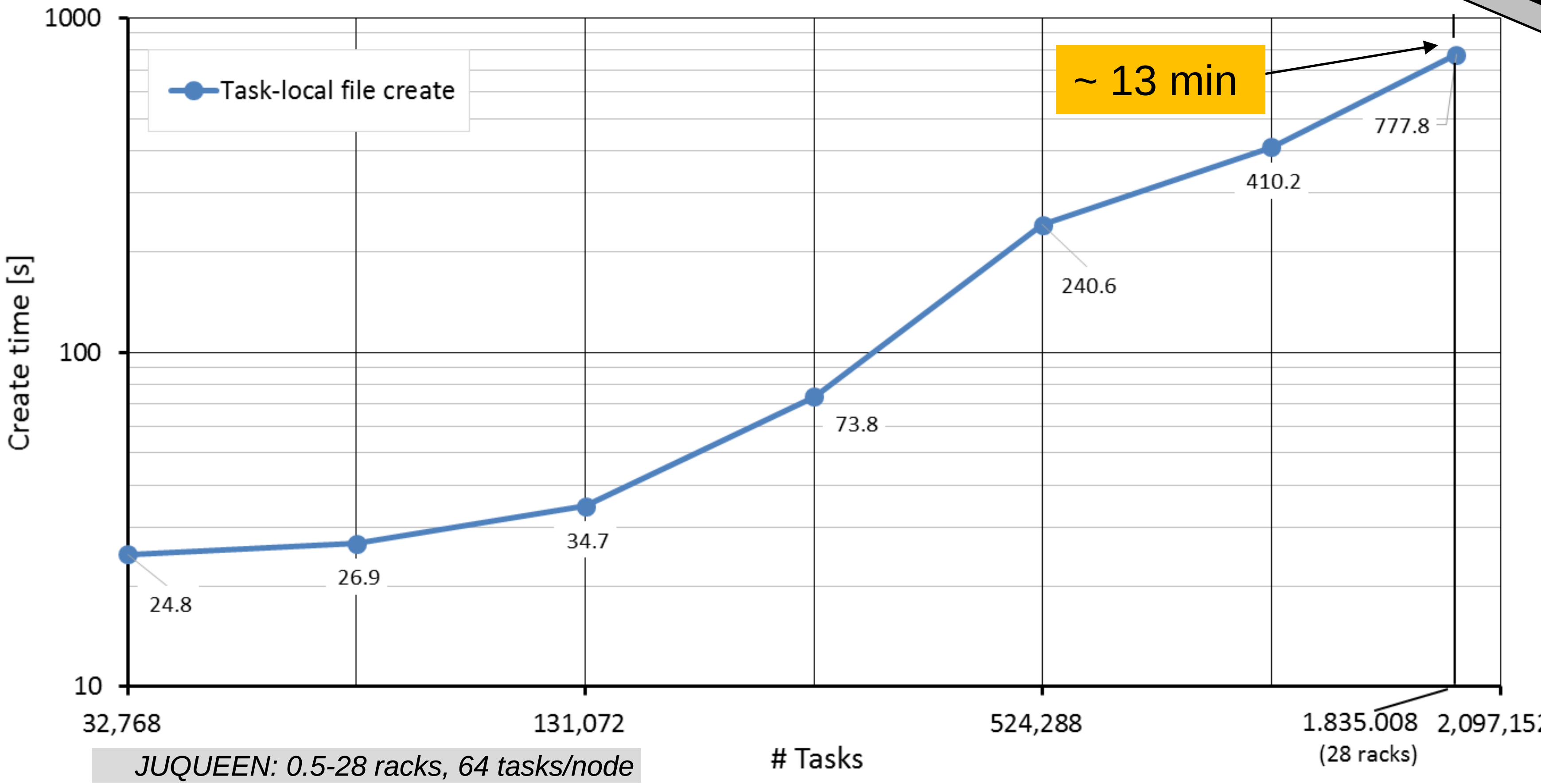
- \+ Simple to implement
- \+ No coordination between processes needed
- \+ No false sharing of file system blocks
- \- Number of files quickly becomes unmanageable
- \- Files often need to be merged to create a canonical dataset
- \- File system might serialize meta data modification



# Serialization of meta data modification

Pitfall 2

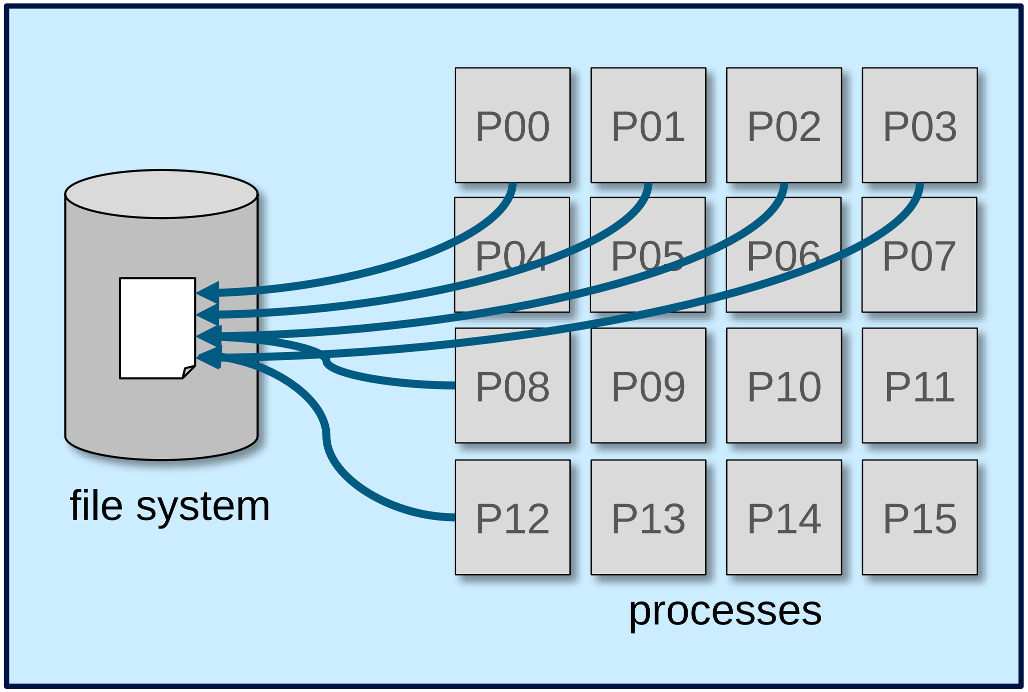
Example: Creating files in parallel in the same directory




The creation of 1.8 M files costs 99.116 core hours!

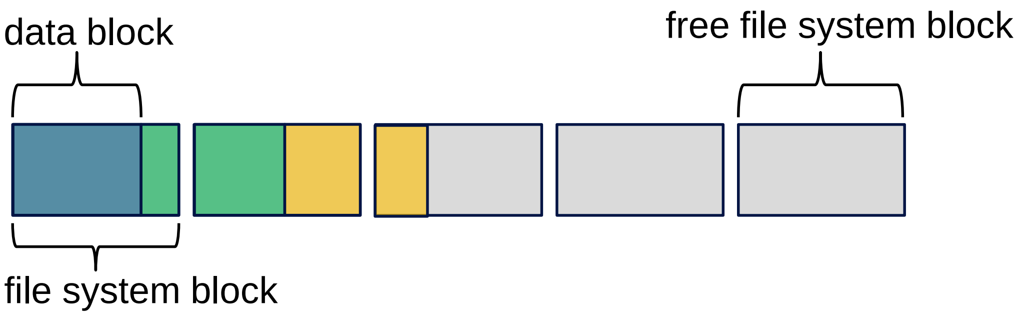
# Shared files

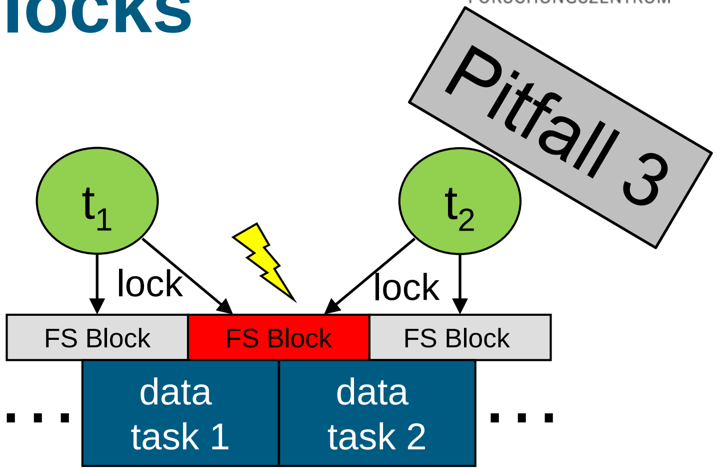
+ Number of files is independent of number of processes
+ File can be in canonical representation (no post-processing)




- Uncoordinated client requests might induce time penalties
- File layout may induce false sharing of file system blocks

# False sharing of file system blocks




- 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

# Number of Tasks per Shared File

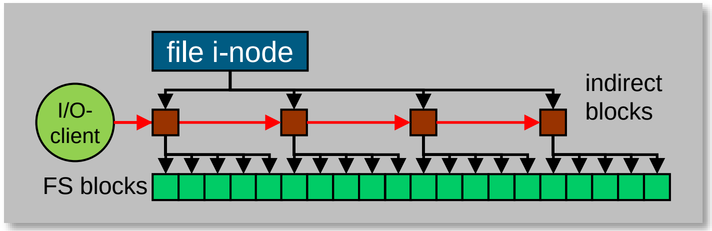
Pitfall 4

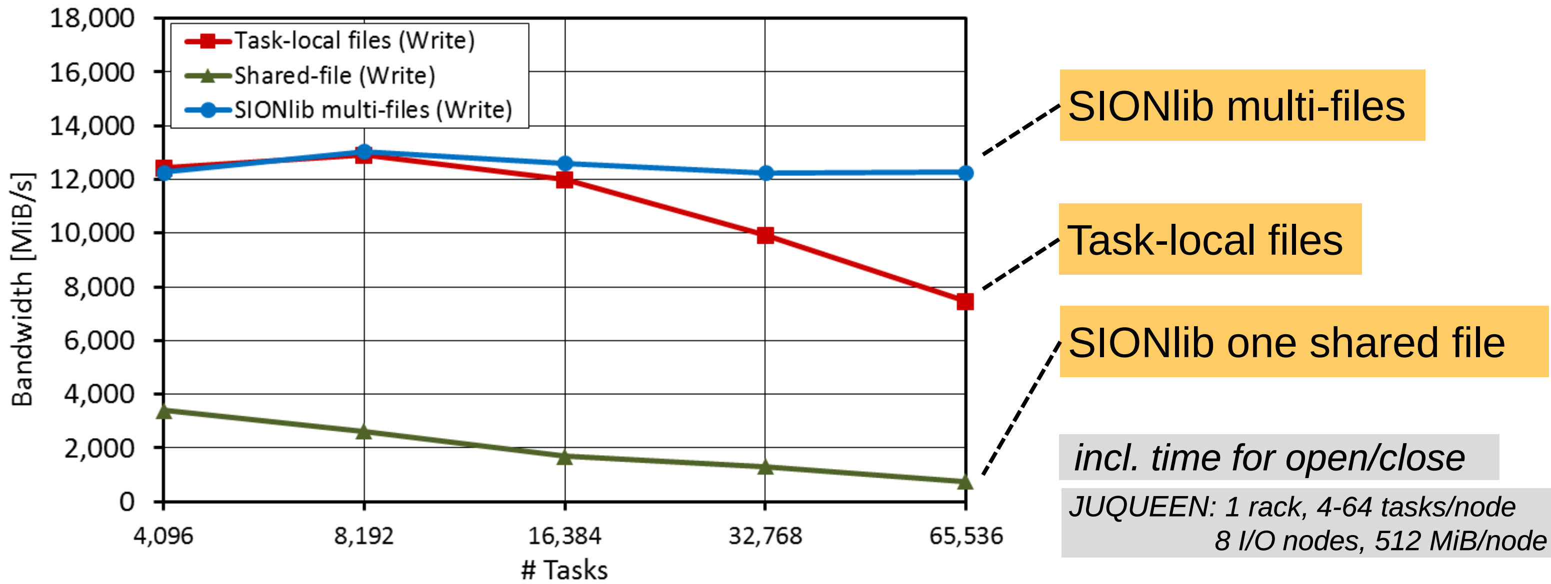
Meta-data wall on file level

- File meta-data management
- Locking

Example Blue Gene/Q

- I/O forwarding nodes (ION)
- GPFS client on ION, one file per ION

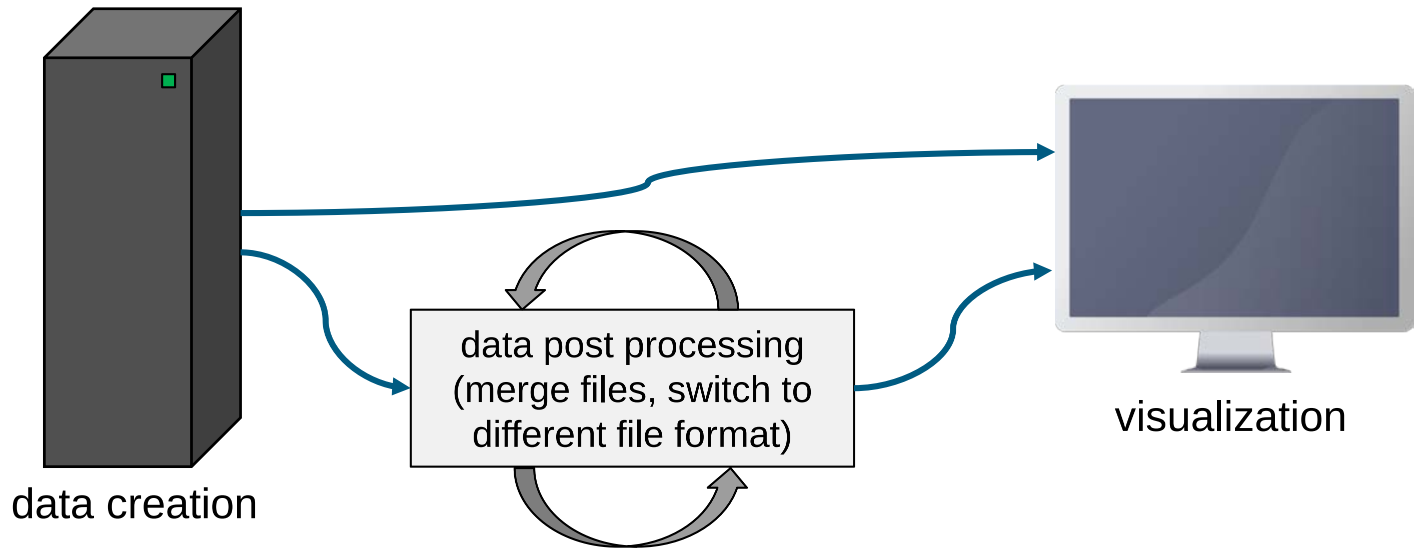



*incl. time for open/close*

*JUQUEEN: 1 rack, 4-64 tasks/node*
*8 I/O nodes, 512 MiB/node*

# I/O Workflow




- Post processing can be very time-consuming (> data creation)
    - Widely used portable data formats avoid post processing
- Data transportation time can be long:
    - Use shared file system for file access, avoid raw data transport
    - Avoid renaming/moving of big files (can block backup)

# Portability

Pitfall 5

- Endianness (byte order) of binary data

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

# Portability

Pitfall 6

- Memory order depends on programming language

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

→

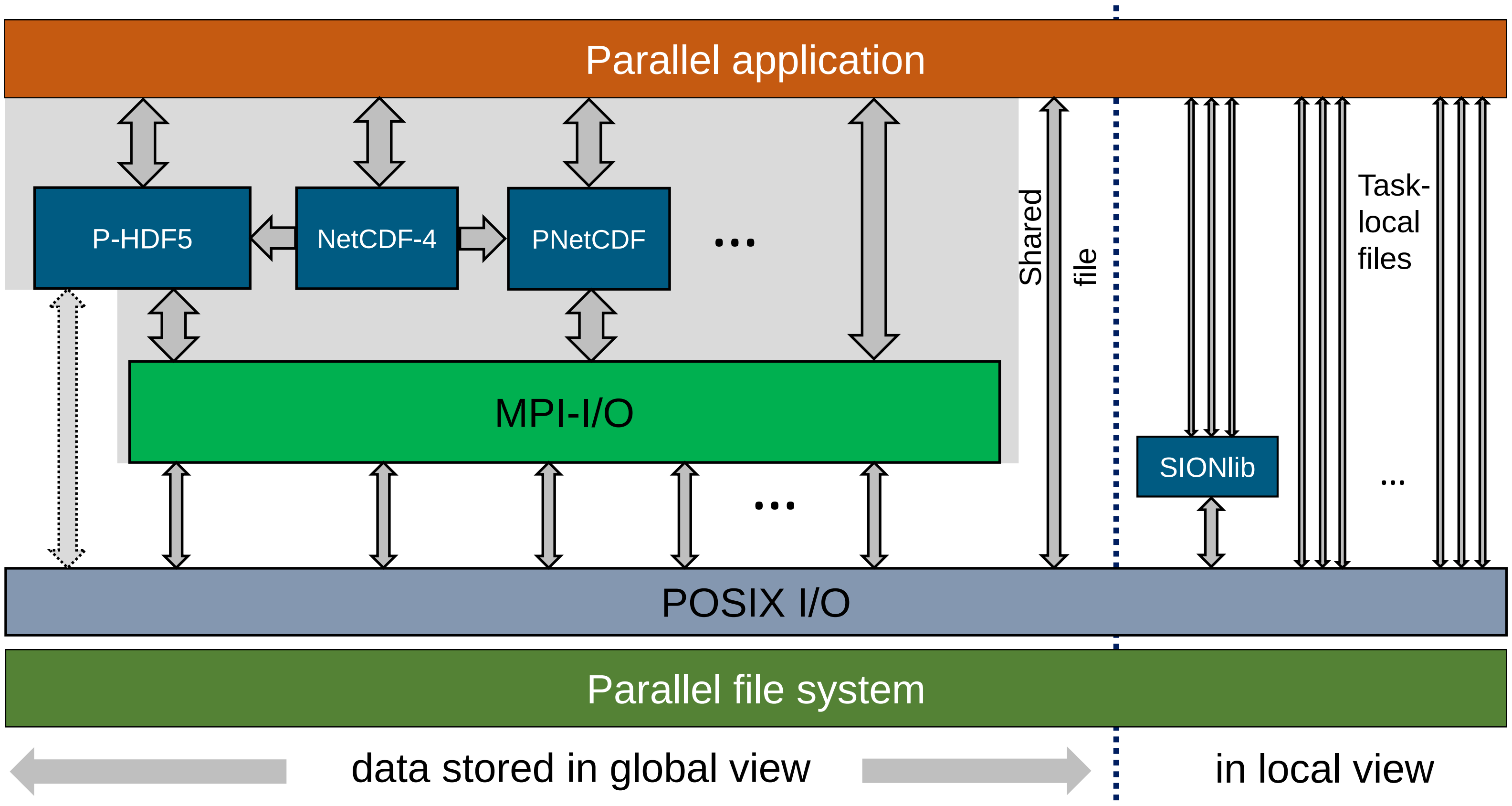
| Address | row-major order (e.g. C/C++) | column-major order (e.g. Fortran) |
|---|---|---|
| 1000 | 1 | 1 |
| 1001 | 2 | 4 |
| 1002 | 3 | 7 |
| 1003 | 4 | 2 |
| 1004 | 5 | 5 |
| … | … | … |

- Transpose of array might be necessary when using different programming languages in the same workflow
- Solution: Choosing a portable data format (HDF5, NetCDF)

# How to choose the I/O strategy?

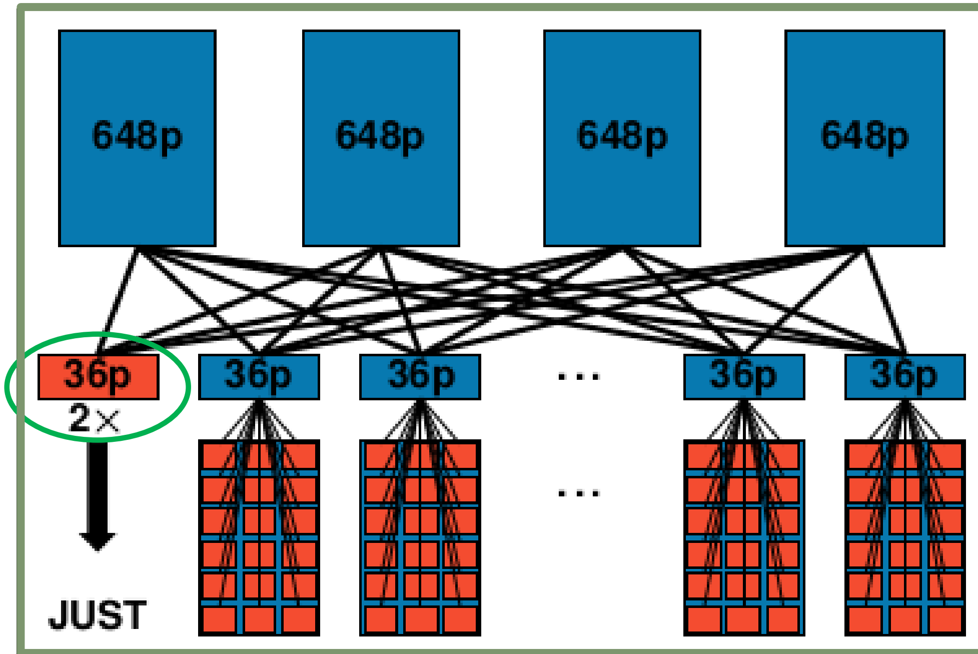
- Performance considerations
  - Amount of data
  - Frequency of reading/writing
  - Scalability
- Portability
  - Different HPC architectures
  - Data exchange with others
  - Long-term storage
- E.g. use two formats and converters:
  - Internal: Write/read data “as-is”
    → Restart/checkpoint files
  - External: Write/read data in non-decomposed format
    (portable, system-independent, self-describing)
    → Workflows, Pre-, Post-processing, Data exchange

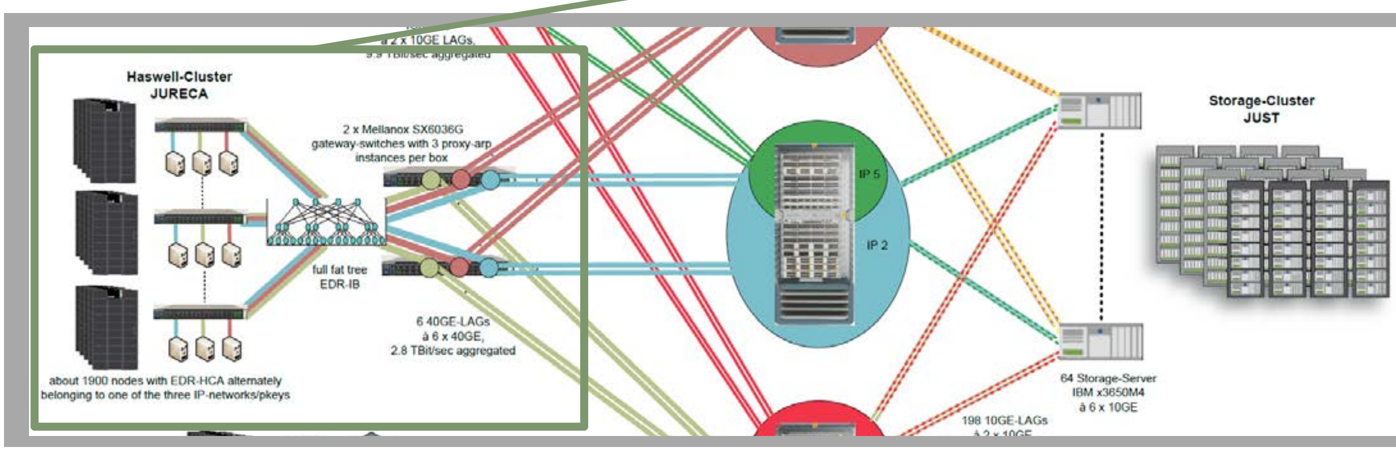
# Parallel I/O Software Stack



# Jureca: I/O infrastructure

- Overall bandwidth: > 100 GB/s
- Max. I/O bandwidth / node
  Write → 5.4 GB/s
  Read → 2.1 GB/s
  (current measurements)
- Nodes share links to gateway switches → varying bandwidth/node depending on overall system I/O load



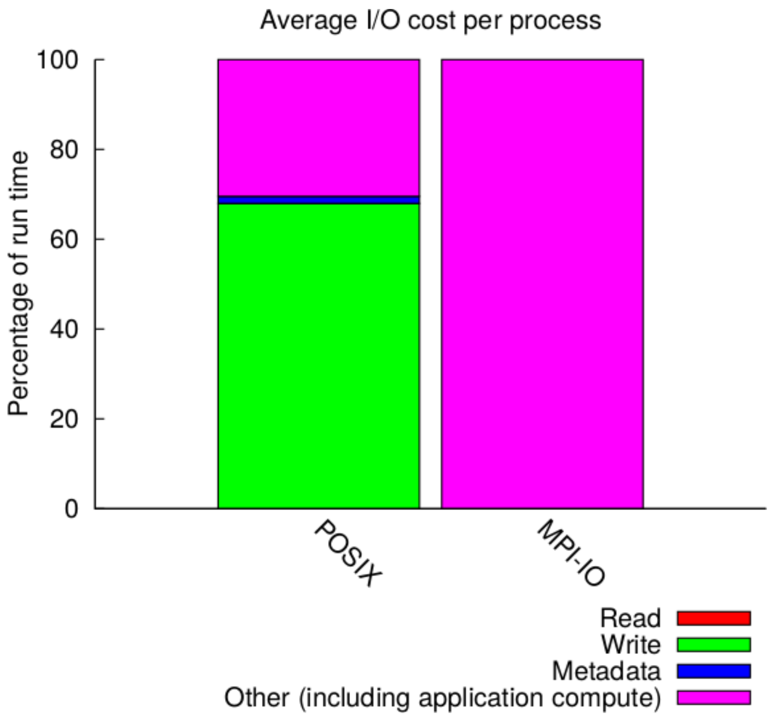



# Darshan: Usage example on JURECA

- Load module
  - module load intel-para darshan-runtime
- Tell srun to use Darshan (in submit script)
  - ```
    LD_PRELOAD=$EBROOTDARSHANMINRUNTIME/lib/libdarshan.so \
    DARSHAN_LOG_PATH=/path/to/your/logdir \
    srun -n 32 ./executable
    ```
- Analyse output
  - module load intel-para darshan-util
  - darshan-job-summary.pl mylog.darshan.gz
  - evince mylog.pdf
    ```

# Darshan: Interpret the summary

- Average and statistical information on I/O patterns
  - Relative time for I/O
  - Most common access sizes
- Additional metrics
  - File count
  - I/O size histogram
  - Timeline for read / write per task
  - …




Most Common Access Sizes

| access size | count |
|---|---|
| 4194304 | 256 |