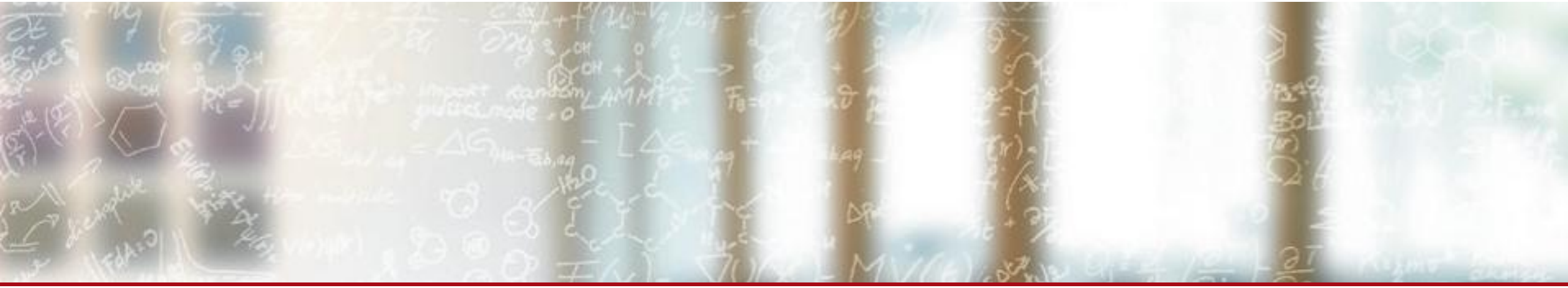
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Parallel I/O on Piz Daint

User Lab Day 2018

Samuel Omlin, CSCS

September 11th 2018

Outline

- Introduction to I/O on Piz Daint
  - Scratch – a Lustre file system
- Parallel I/O? – Common approaches
  - File-per-process
  - Shared file
  - Recommendations for Piz Daint
- General recommendations for any I/O approach on Piz Daint
- Conclusions

CSCS office building in Lugano

Introduction to I/O on Piz Daint

Scratch – a Lustre file system

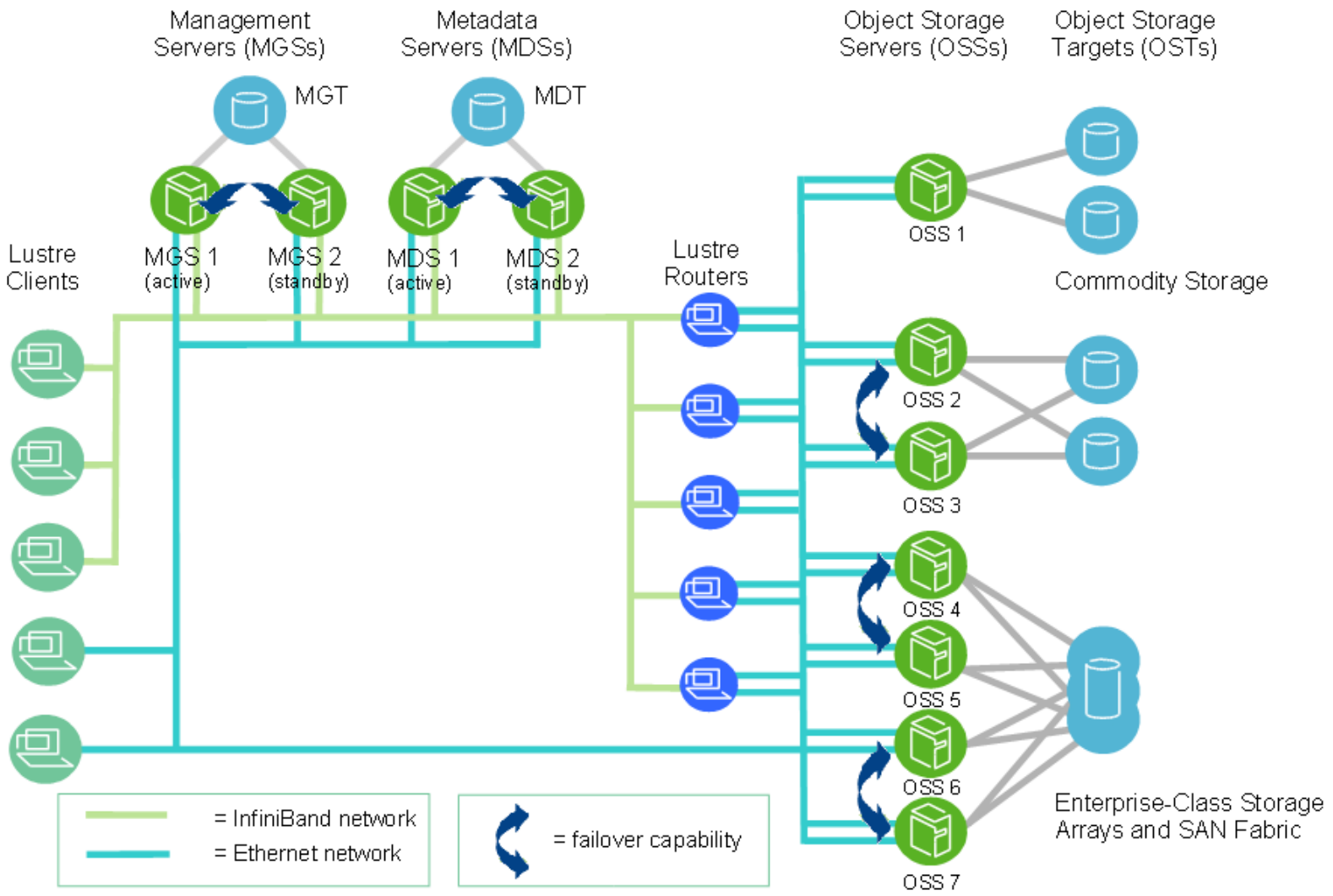
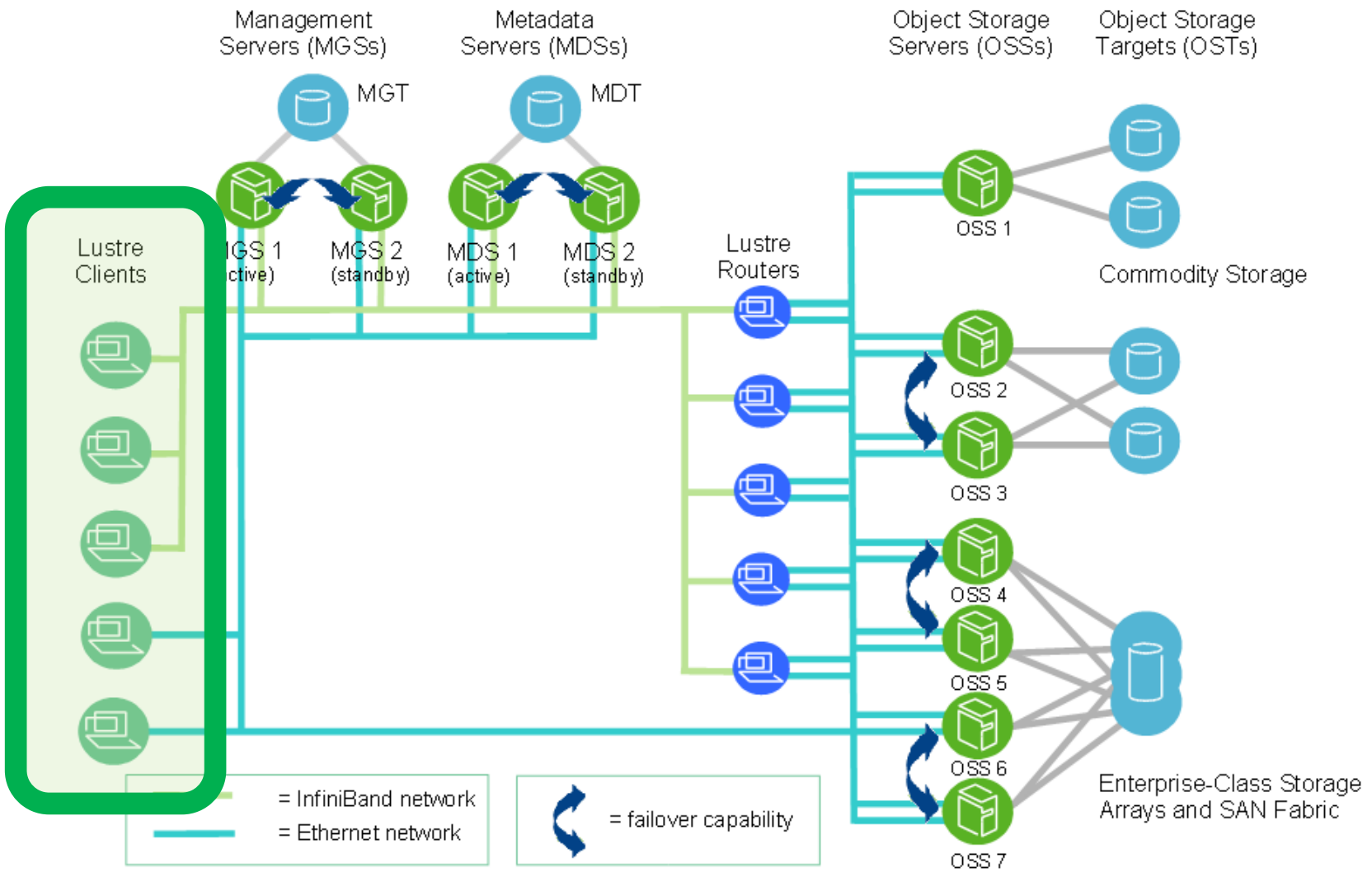
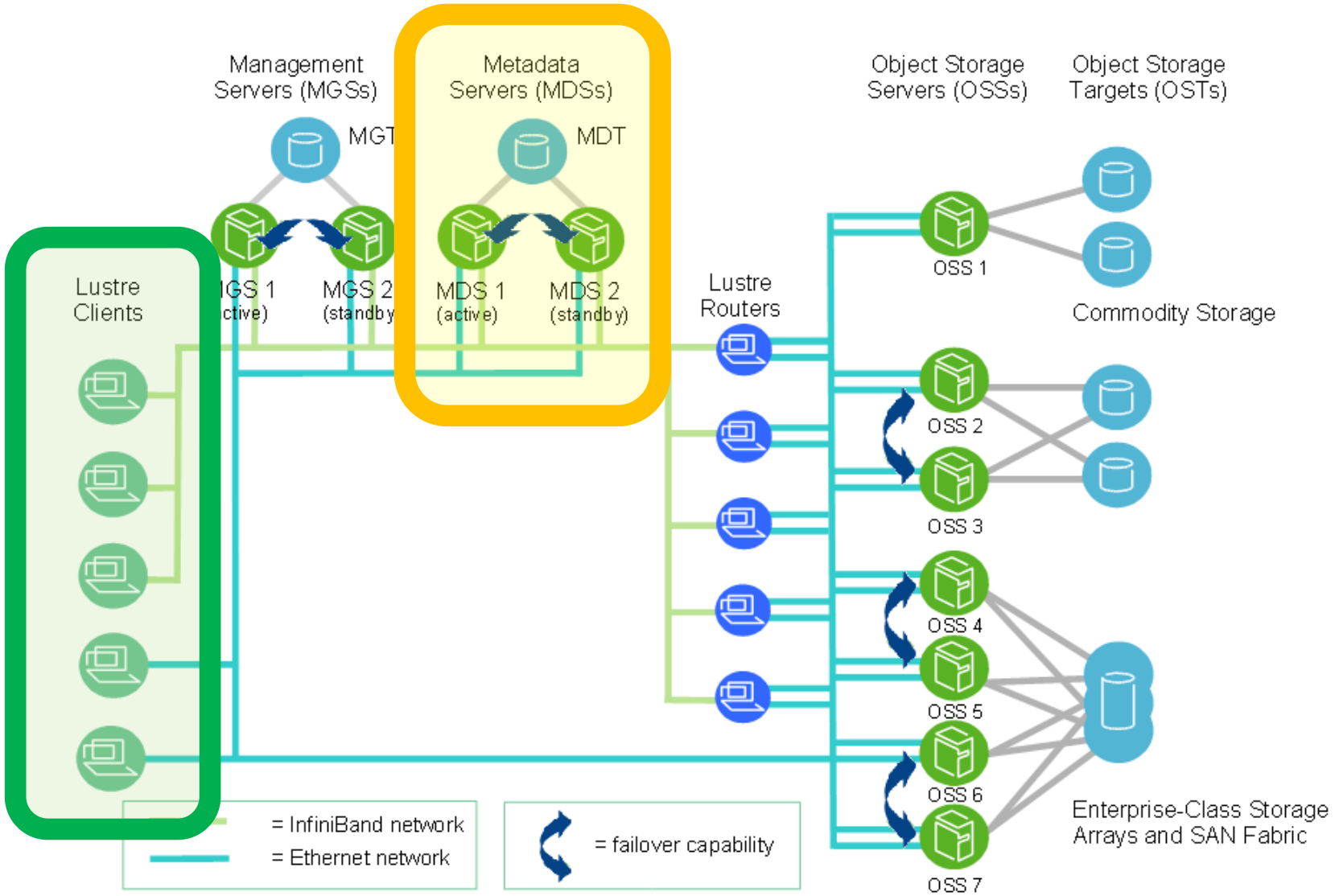


Image from lustre.org

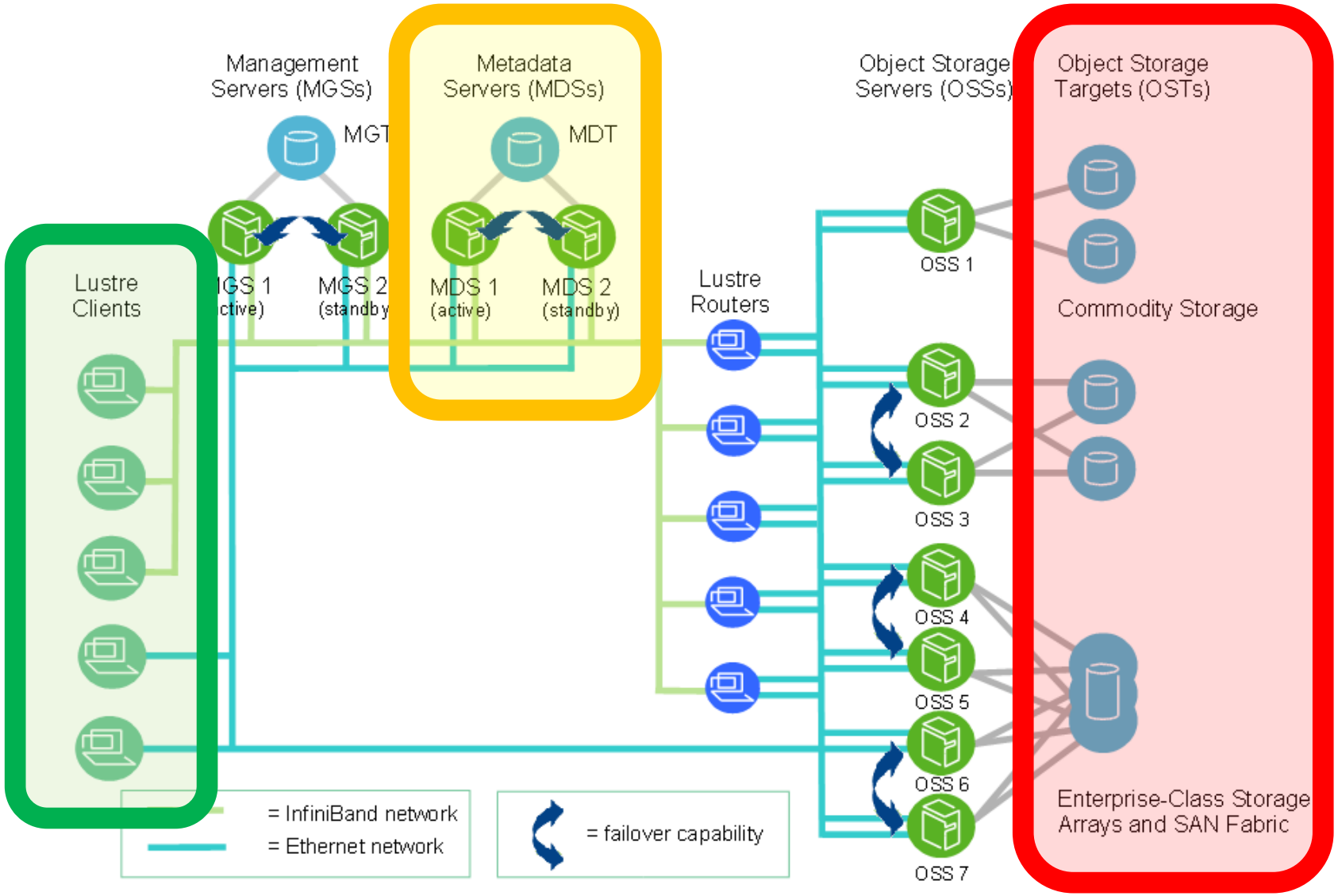
Scratch – a Lustre file system



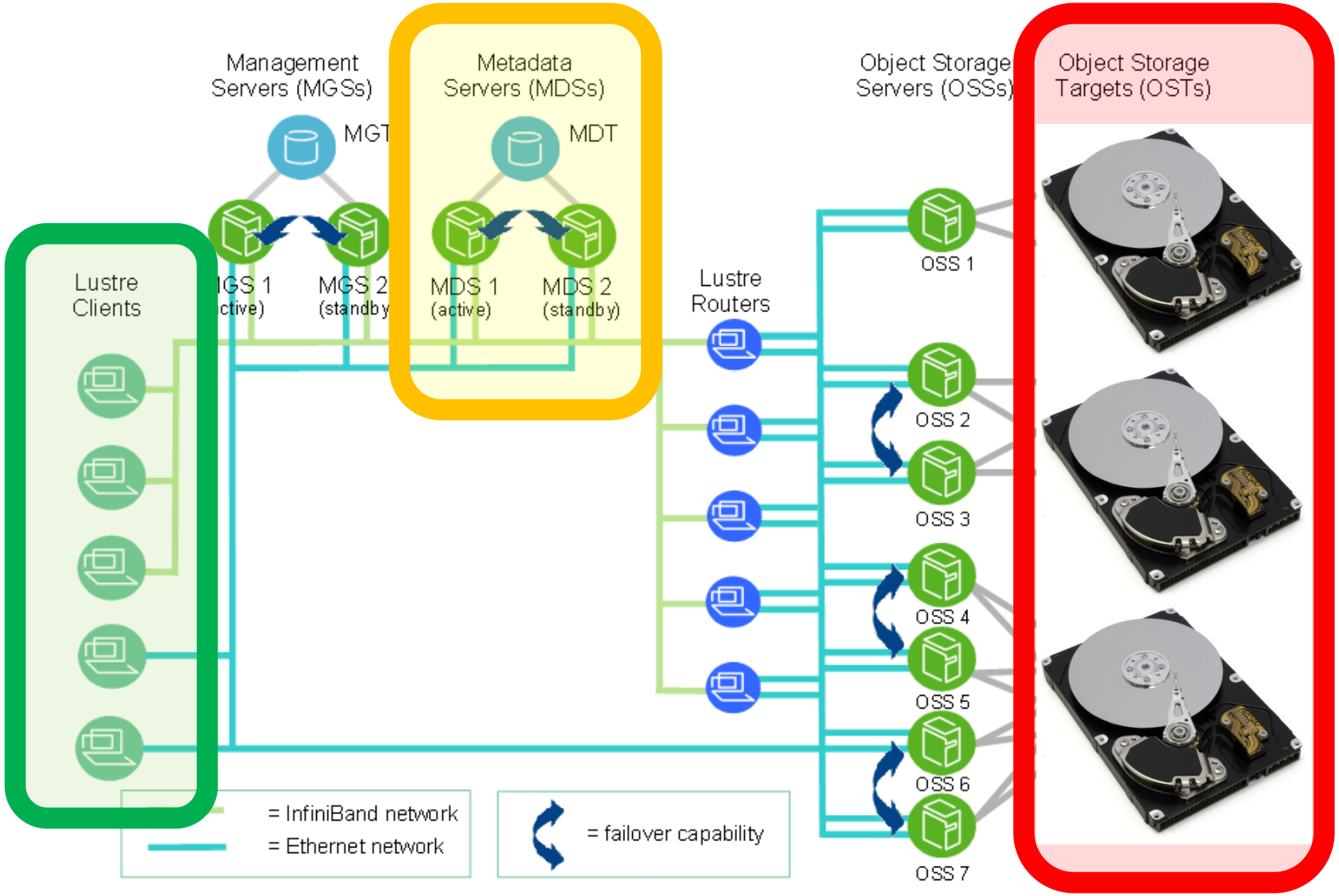
Scratch – a Lustre file system



Scratch – a Lustre file system



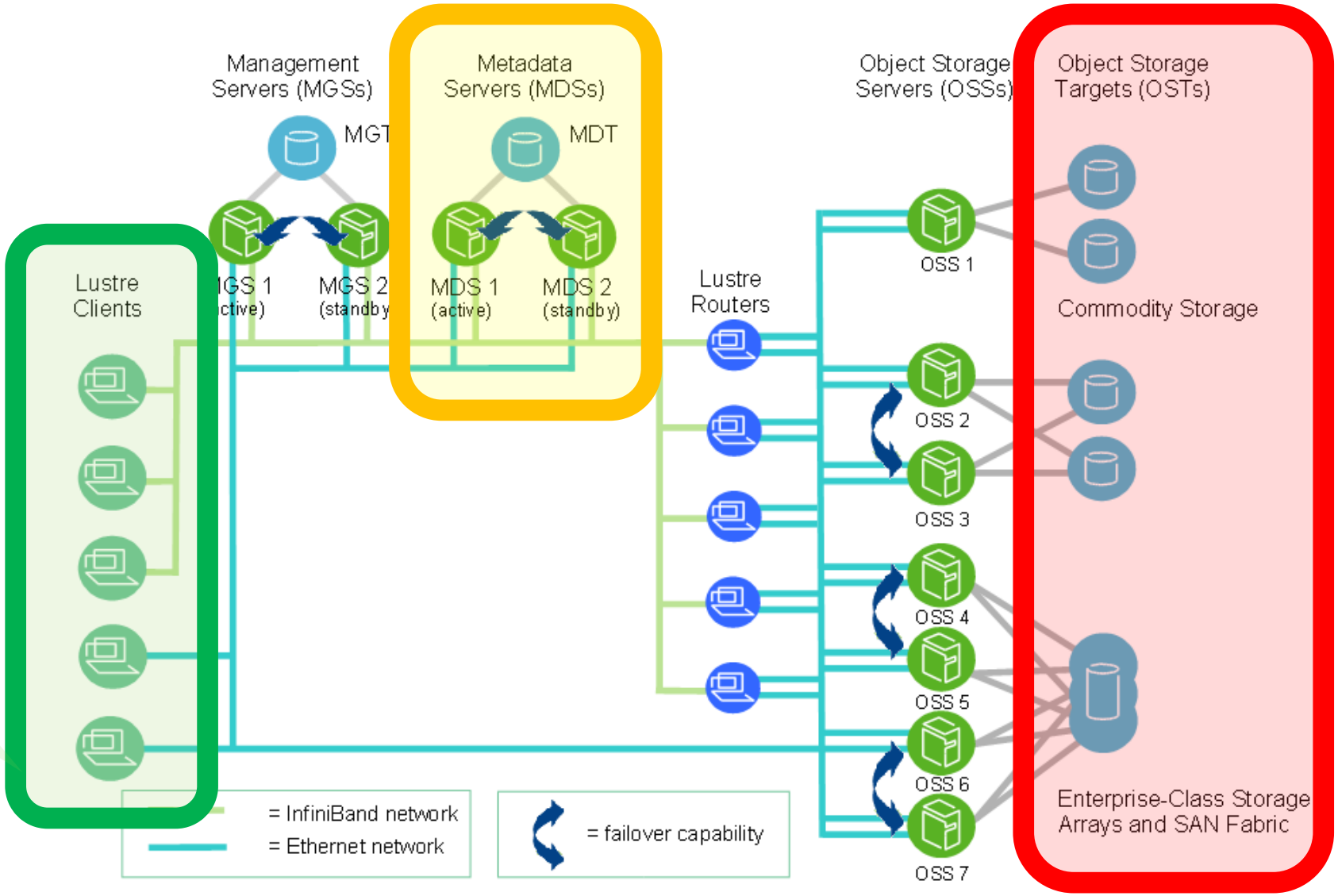
Scratch – a Lustre file system



Scratch – a Lustre file system

$SCRATCH:
/scratch/snx3000/$USER

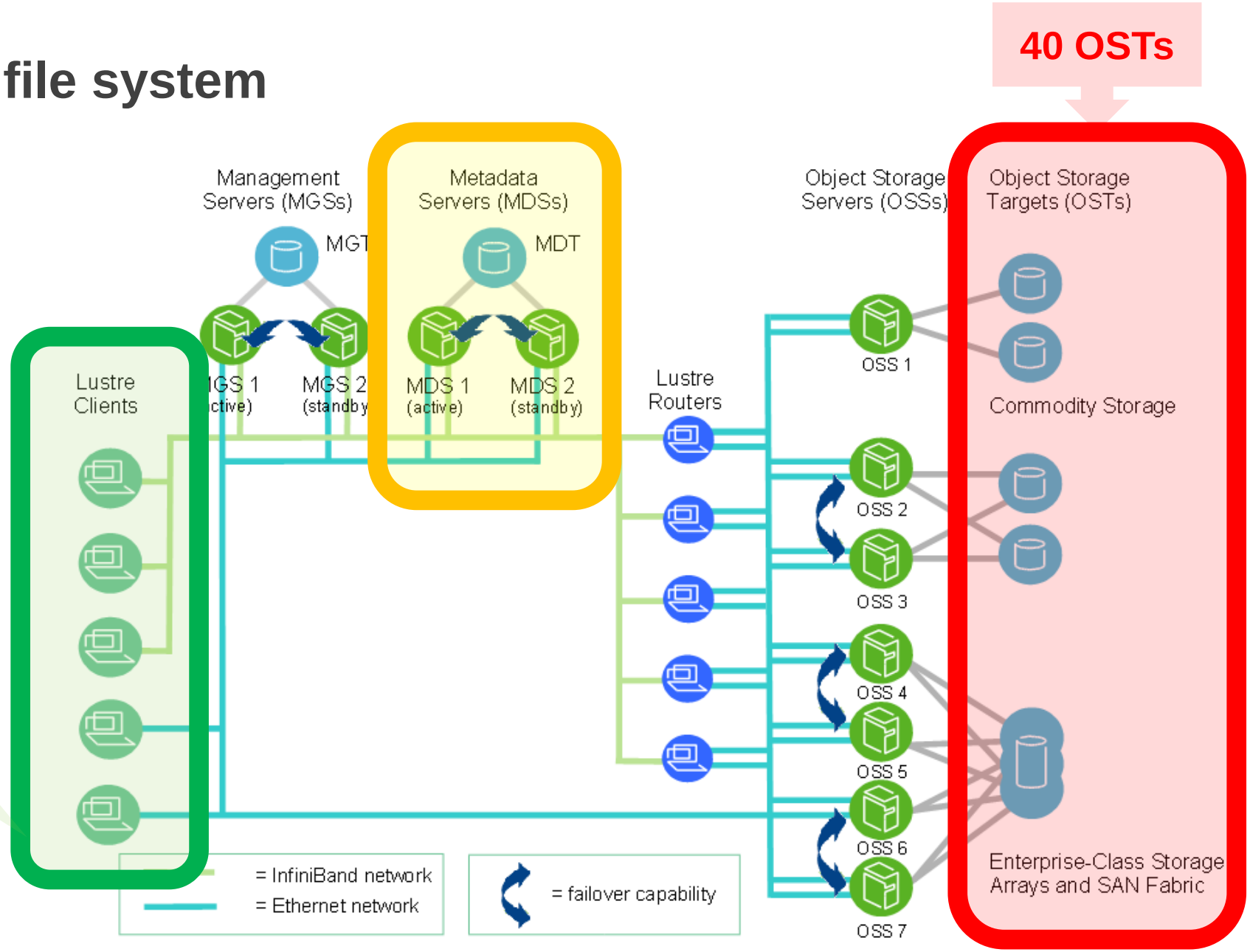
snx3000:
Cray Sonexion 3000
(>100 GiB/s)



Scratch – a Lustre file system

$SCRATCH:
/scratch/snx3000/$USER

snx3000:
Cray Sonexion 3000
(>100 GiB/s)

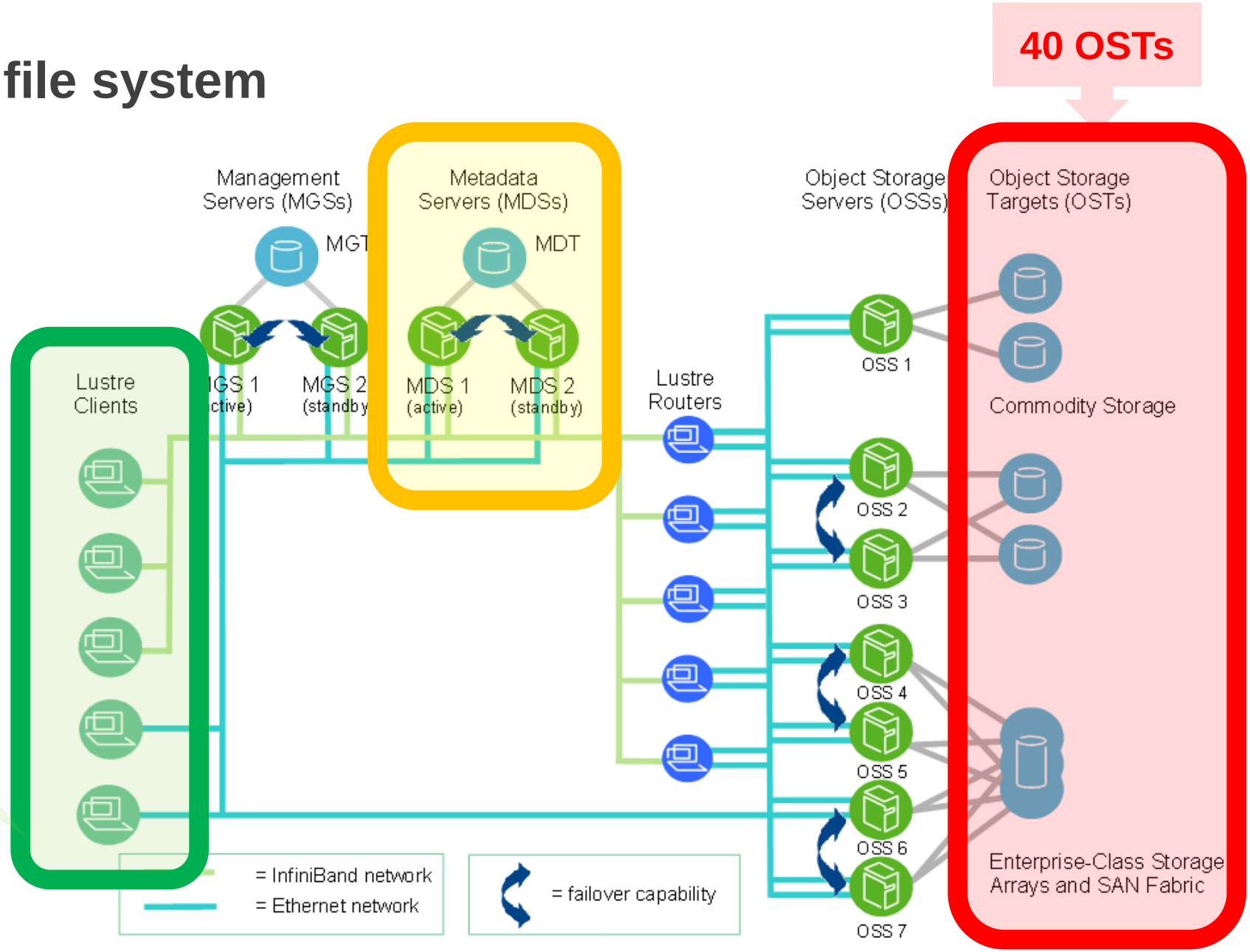


Scratch – a Lustre file system

$SCRATCH:
/scratch/snx3000/$USER

snx3000:
Cray Sonexion 3000
(**>100 GB/s**)

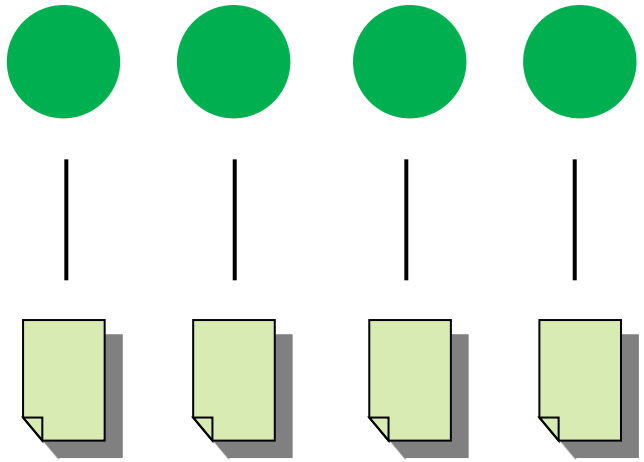
Only possible if all or most OSTs accessed => parallel I/O



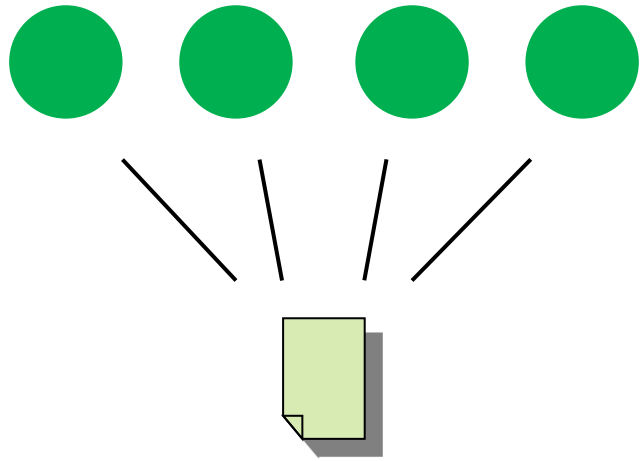
Parallel I/O? – Common approaches

Parallel I/O? – Common approaches

- File-per-process

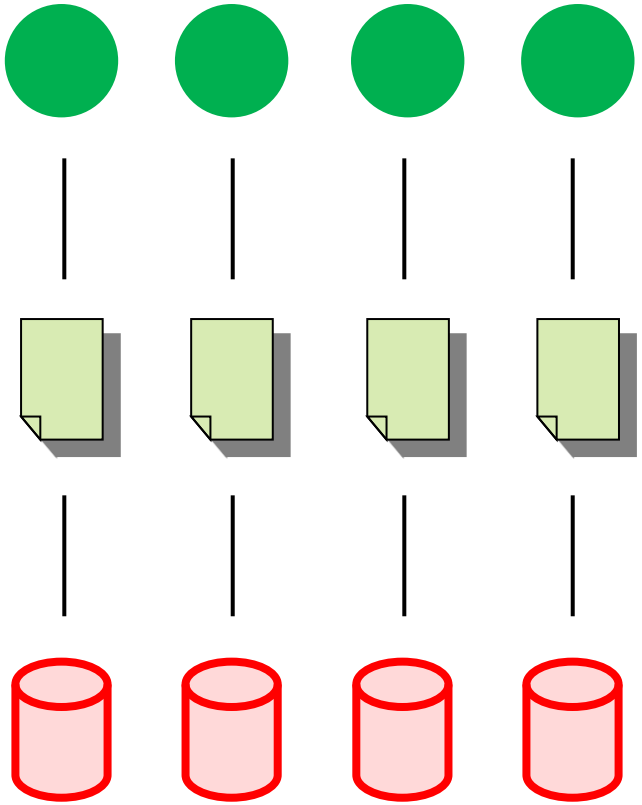


- Shared file

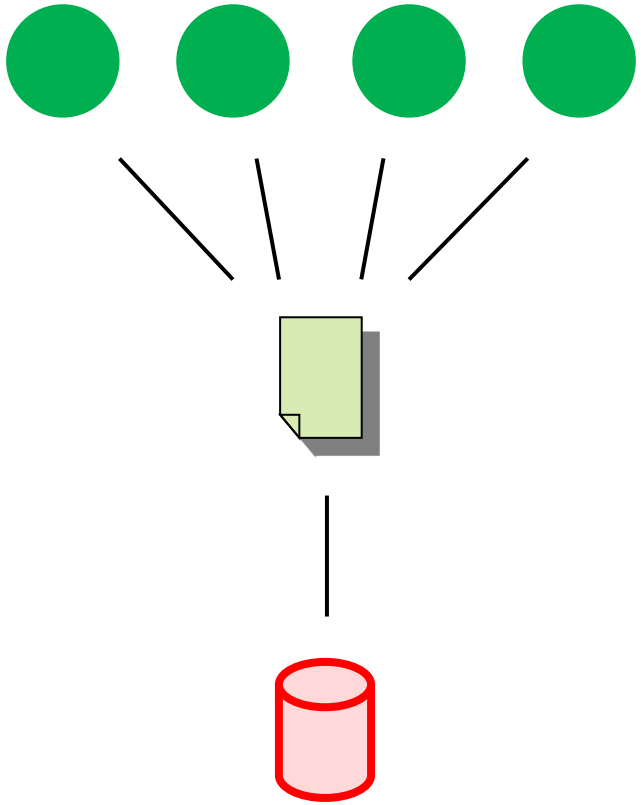


Parallel I/O? – Common approaches

- File-per-process

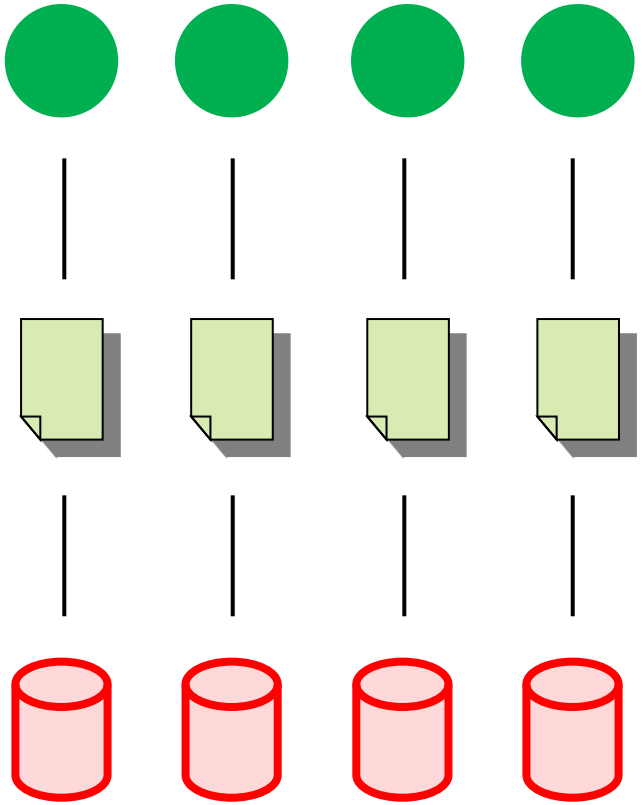


- Shared file

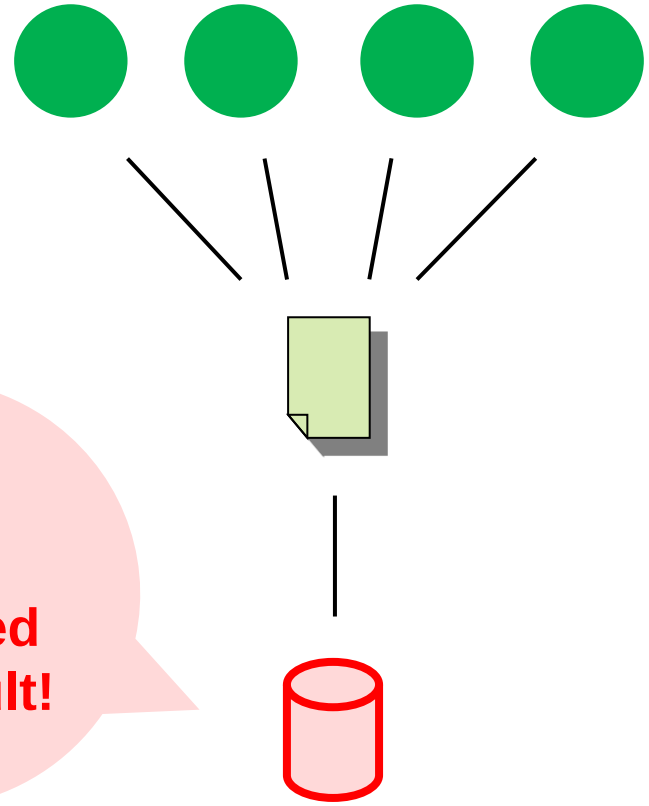


Parallel I/O? – Common approaches

- File-per-process



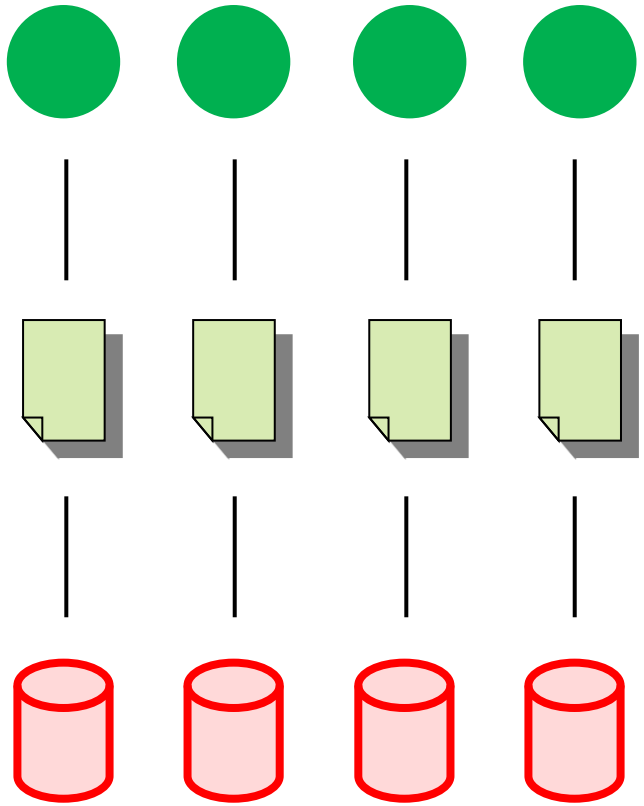
- Shared file



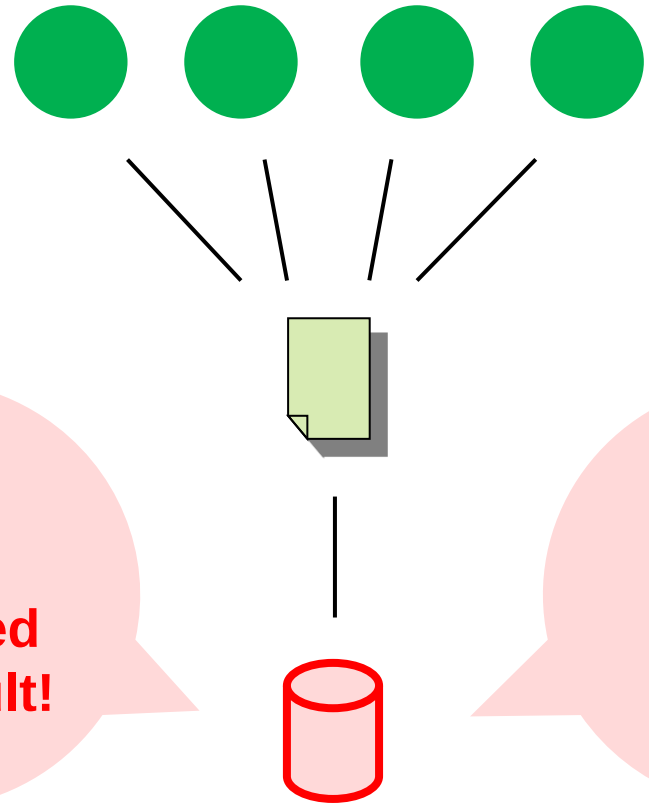
Only 1 OST accessed by default!

Parallel I/O? – Common approaches

- File-per-process



- Shared file

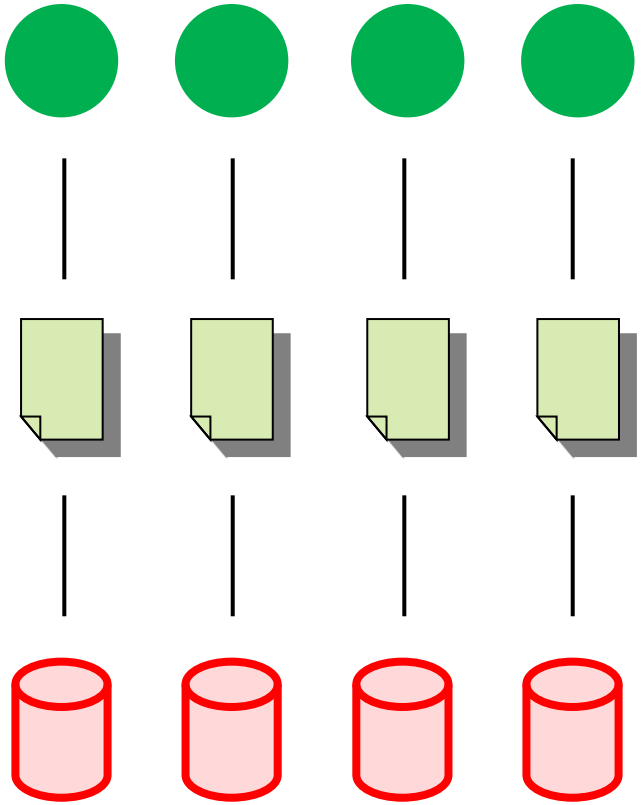


Only 1 OST accessed by default!

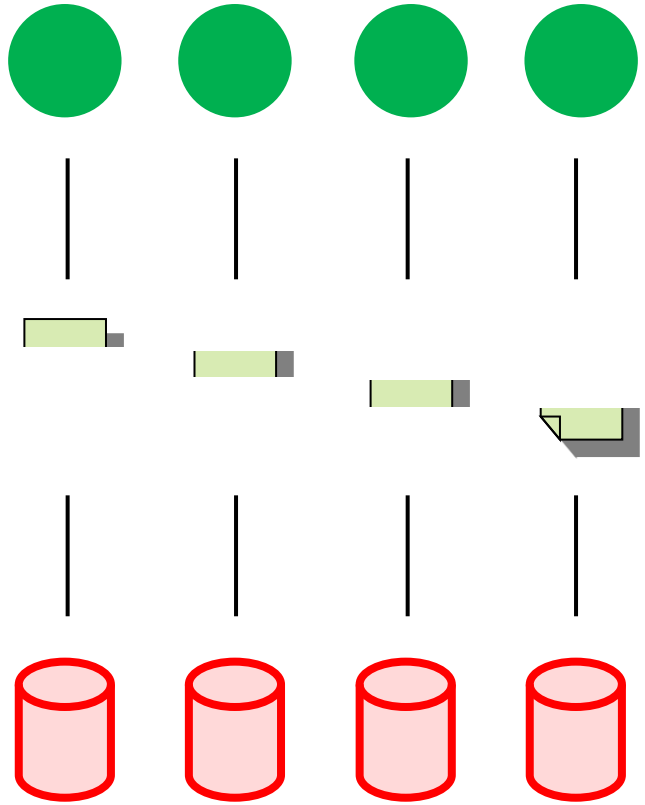
Data striping needed!

Parallel I/O? – Common approaches

- File-per-process

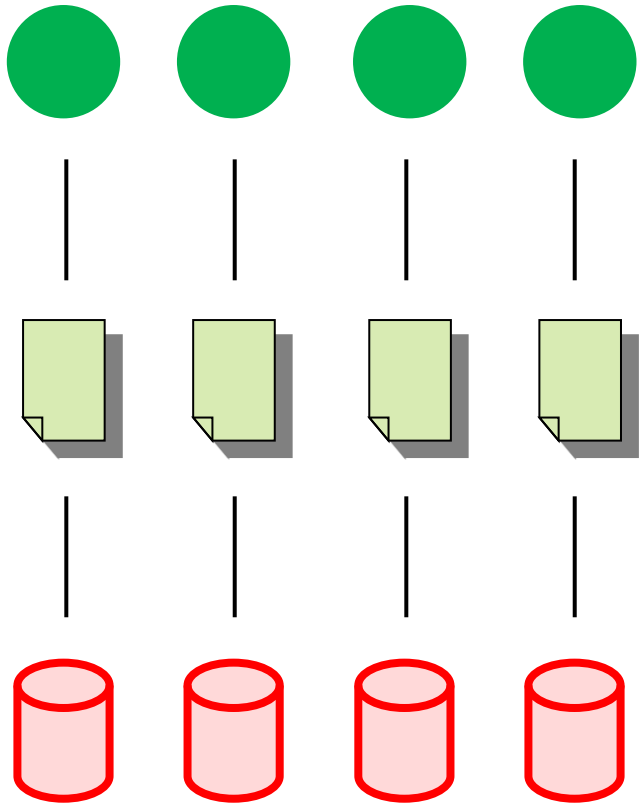


- Shared file (with striping)

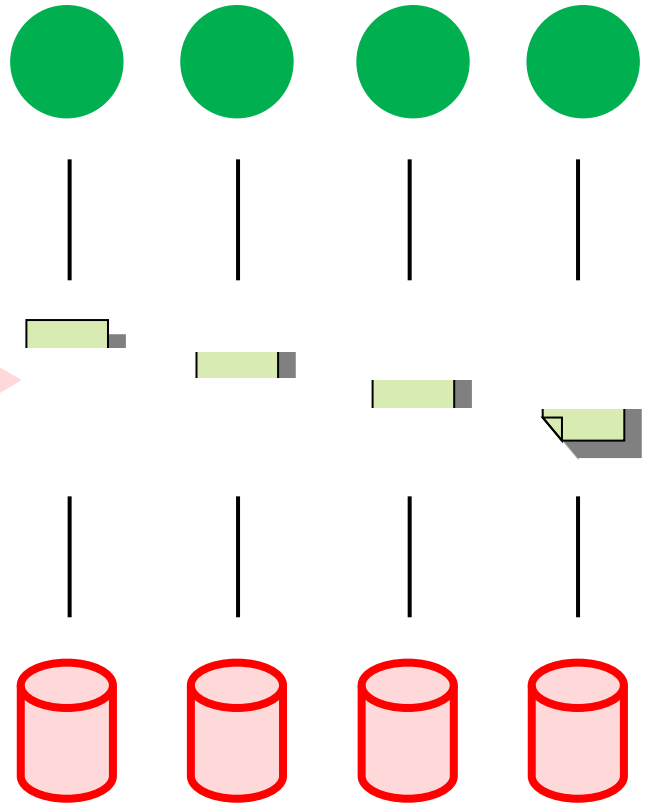


Parallel I/O? – Common approaches

- File-per-process



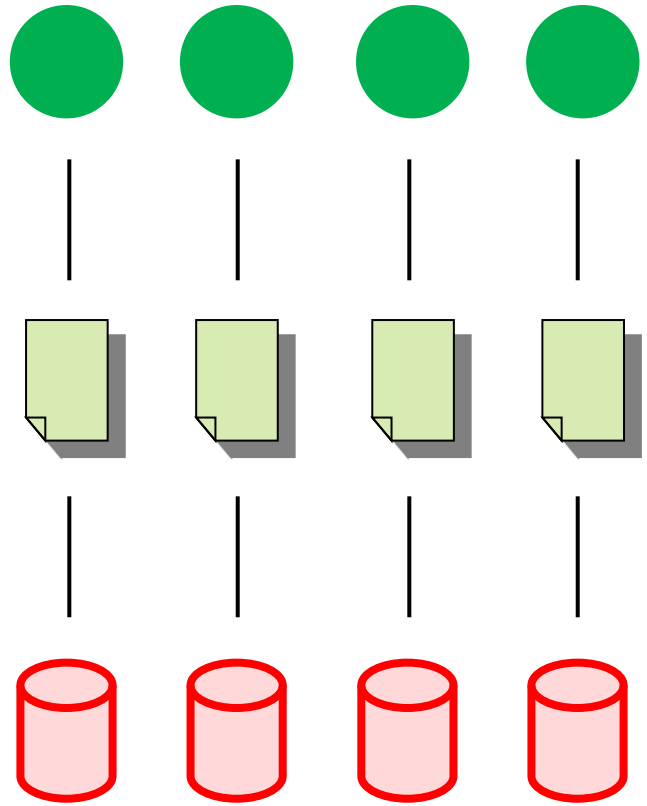
- Shared file (with striping)



Stripe count = 4 (distribute file on 4 OSTs)

Parallel I/O? – Common approaches

- File-per-process

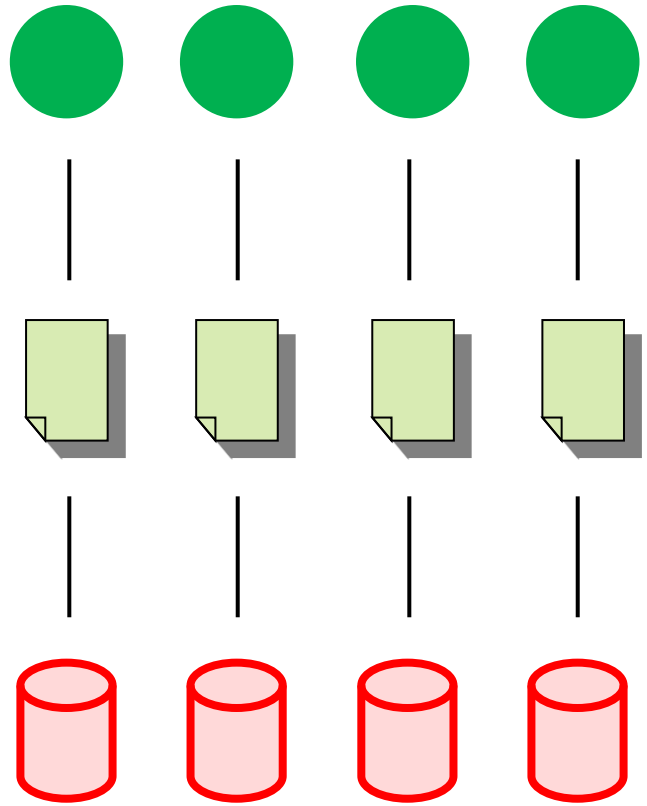


👍 **Maximal simplicity is possible** on the application level (no I/O library is needed).

👎 Creates **many files** if used at large scale (=> risk of contention of metadata servers).

Parallel I/O? – Common approaches

- File-per-process



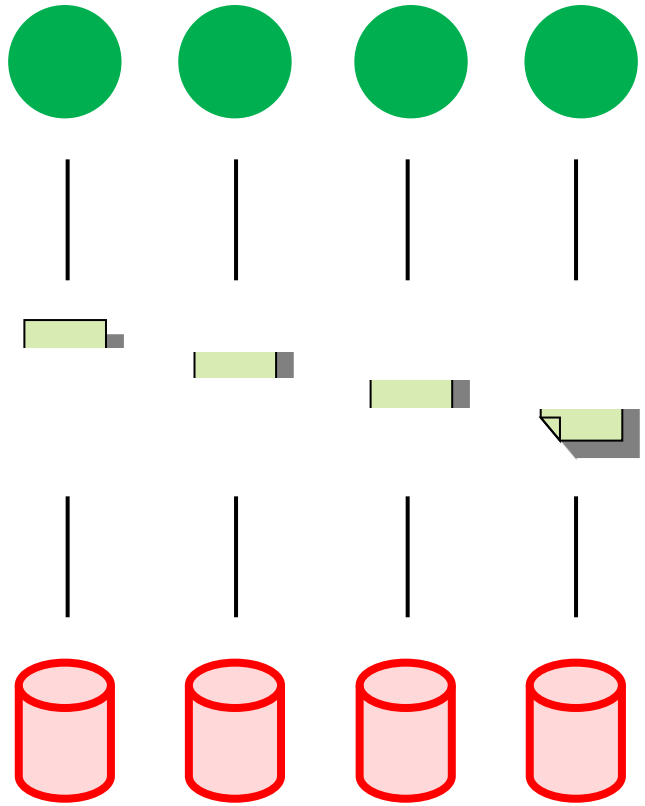
General recommendations

- Set $stripe\ count = 1$ (default).*
- Don’t create thousand of files in a single folder (group files in subfolders).
- Don’t access thousand of files simultaneously.
- Be aware that if you cause contention, all users will suffer (all I/O resources are shared).

* At very small scale, $stripe\ count > 1$ may be better.

Parallel I/O? – Common approaches

- Shared file (with striping)



👍 Keeping the **number of files** small is straightforward.

👍 **Post-processing is simple** with common tools as I/O libraries make it easy to
(1) add extensive metadata, and
(2) create shared files as if it was done by a single process.

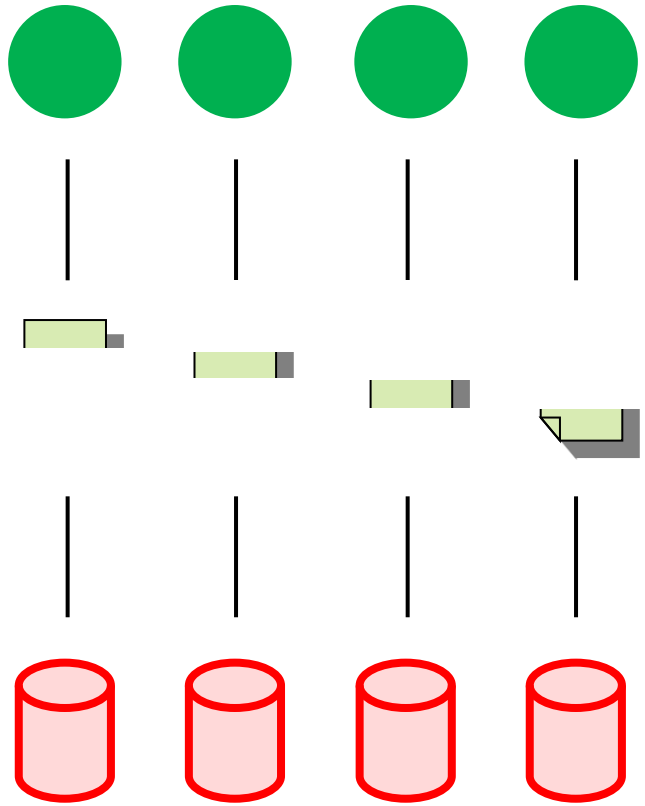
👎 **Careful tuning** is often required to reach good I/O performance.

Parallel I/O? – Common approaches

General recommendations (1/5)

- If you access **one large shared file** (GBs), set $stripe\ count = \#OSTs$.
- If you simultaneously access **multiple large shared files**, set $stripe\ count$ such that $\#files * stripe\ count = k * \#OSTs,$ $where\ k\ \in \{1, \dots 4\}$.
- $\#OSTs = 40$ on snx3000. Nevertheless, try the above formulas also with the value 32 (better alignment possible).
- For any small file, set $stripe\ count = 1$.

- Shared file (with striping)

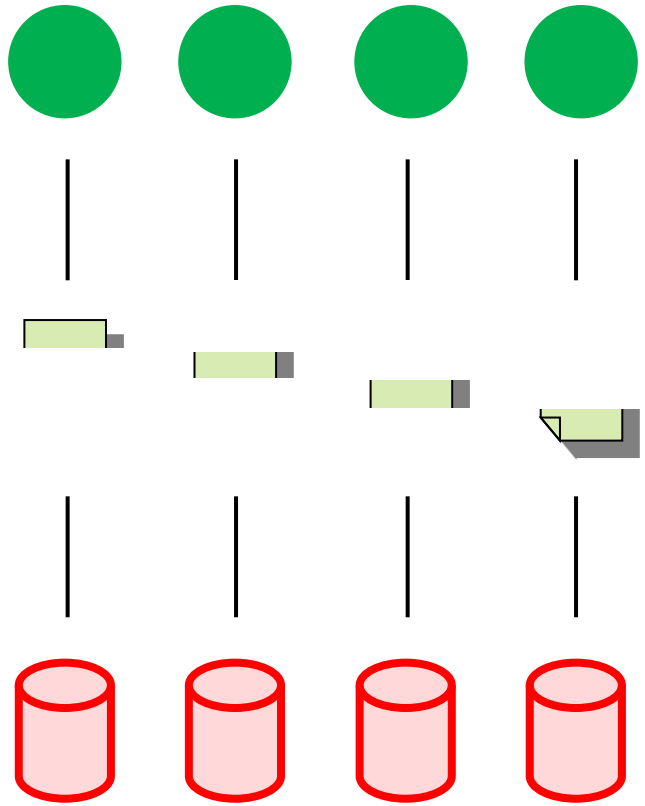


Parallel I/O? – Common approaches

General recommendations (2/5)

- Convenient: set the striping configuration for your simulation output folder(s), e.g.
  `lfs setstripe --stripe-count 32 <output folder>`
  => Inside, files will be created with the same striping configuration as the folder(s) itself.
  Don't copy the executable in there!
- Check the striping configuration of a file or folder:
  `lfs getstripe <file/folder>`
- More information: `lfs --help`

- Shared file (with striping)



Parallel I/O? – Common approaches

General recommendations (3/5)

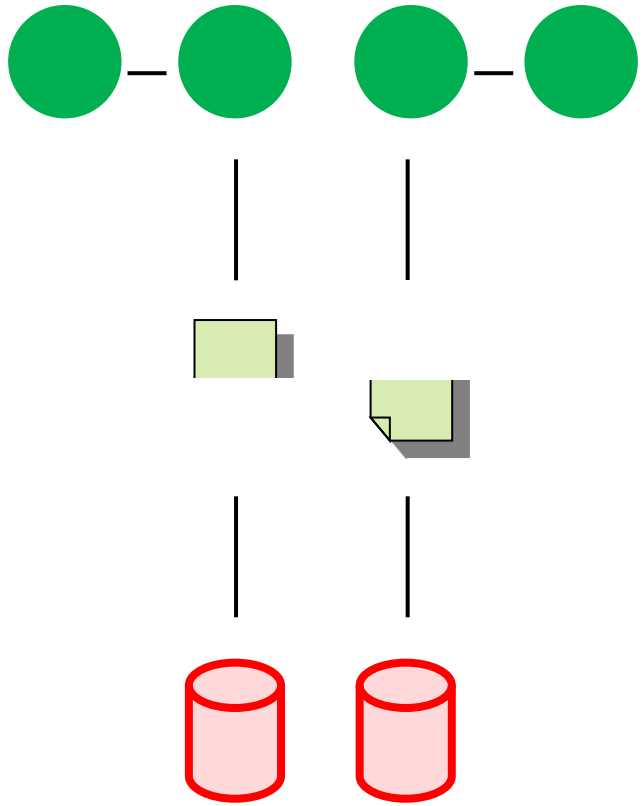
- Use **collective I/O operations** (enable merging of I/O requests of different processes into fewer larger ones).

  MPIIO:
  use functions with suffix '`_all`'
  (e.g. `MPI_File_write_all`)

  HDF5:
  `H5Pset_dxpl_mpio(…, H5FD_MPIO_COLLECTIVE);`

  NetCDF:
  `nc_var_par_access(…, NC_COLLECTIVE);`

- Shared file (with striping + collective buffering)



Parallel I/O? – Common approaches

General recommendations (3/5)

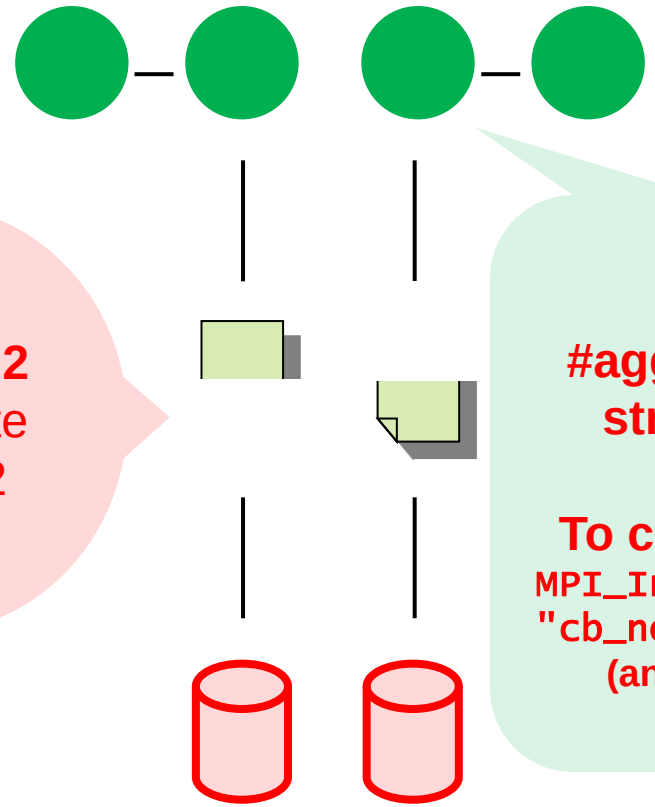
- Use **collective I/O operations** (enable merging of I/O requests of different processes into fewer larger ones).

MPIIO:
use functions with suffix '_all'
(e.g. `MPI_File_write_all`)

HDF5:
```
H5Pset_dxpl_mpio(…, H5FD_MPIO_COLLECTIVE);
```

NetCDF:
```
nc_var_par_access(…, NC_COLLECTIVE);
```

- Shared file (with striping + collective buffering)



Stripe count = 2 (distribute file on 2 OSTs)

Default: **#aggregators = stripe count**

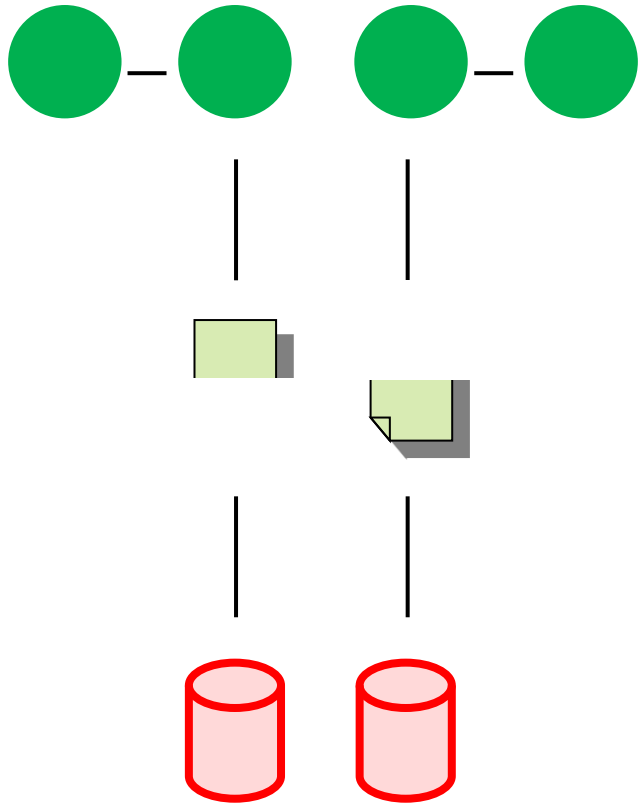
To change, e.g.:
`MPI_Info_set(info, "cb_nodes", "64");`
(and use `info`)

Parallel I/O? – Common approaches

General recommendations (4/5)

- Print used MPIIO hints (e.g. `cb_nodes`; also used for HDF5 and NetCDF as have MPIIO underneath!):
  `export MPICH_MPIIO_HINTS_DISPLAY=1`
- Print statistics about I/O:
  `export MPICH_MPIIO_STATS=1`
- Look up detailed information on MPIIO hints:
  `man intro_mpi`

- Shared file (with striping + collective buffering)

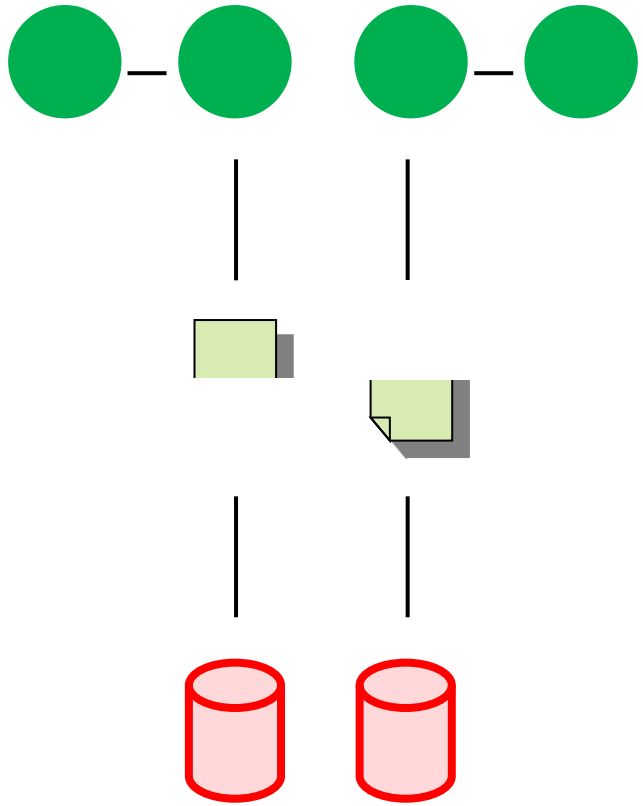


Parallel I/O? – Common approaches

General recommendations (5/5)

- Refer to the **websites of the parallel I/O libraries** for details on their best usage, e.g.:
  HDF5: www.hdfgroup.org
  NetCDF: www.unidata.ucar.edu/software/netcdf/
  ADIOS: www.olcf.ornl.gov/center-projects/adios/
- Follow widely adopted **Metadata conventions** to enable straightforward pre- and post-processing. E.g:
  NetCDF CF Metadata Conventions: cfconventions.org
  XDMF: www.xdmf.org
  GADGET: wwwmpa.mpa-garching.mpg.de/gadget

- Shared file (with striping + collective buffering)

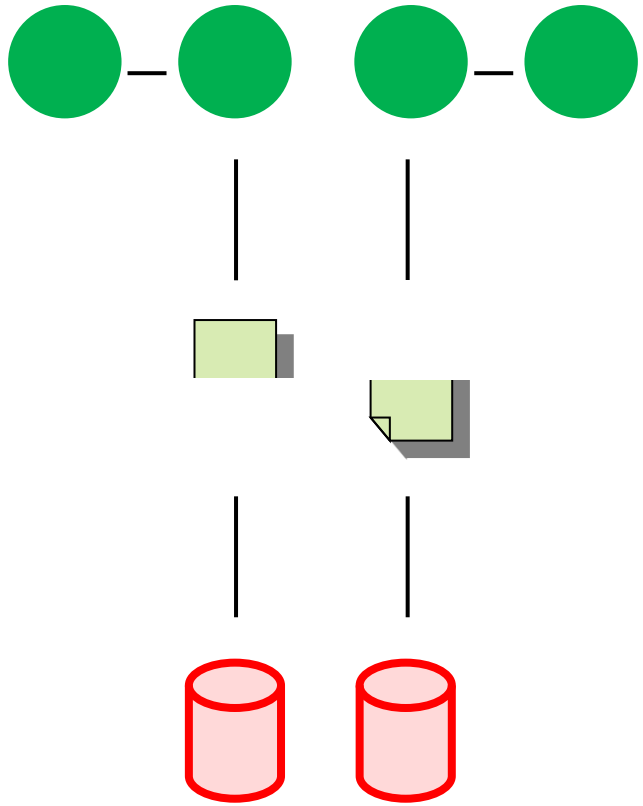


Parallel I/O? – Common approaches

Available parallel I/O libraries on Piz Daint

- `cray-hdf5-parallel` (HDF5)
- `cray-netcdf-hdf5parallel` (NetCDF using HDF5 underneath)
- `cray-parallel-netcdf` (NetCDF using PnetCDF underneath)

- Shared file (with striping + collective buffering)



General recommendations for any I/O approach on Piz Daint

General recommendations for any I/O approach on Piz Daint

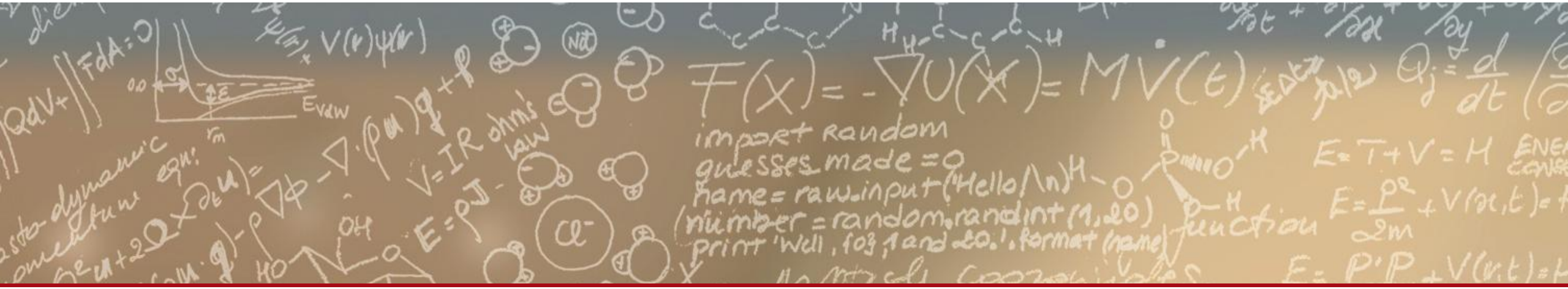
- **No ASCII**, except for small parameter files (easily 10 - 100 times slower that binary)
- Avoid opening and closing files frequently.
- Do not open files for read and write access, but instead for read-only or write-only.
- **Avoid small and frequent I/O request.**
- Avoid random file access; regular access patterns work best in general.
- Avoid multiple processes accessing the same data.
- Read small files just from one process and broadcast the data to the remaining.
- **Limit file metadata access as much as possible on the Lustre file system**; in particular, avoid the usage of `'ls -l'` and use instead `'ls'` or `'lfs find'` when possible.
- **Be aware that if you cause contention on the file system, all users will suffer the slowdowns as all I/O resources are shared.**

Conclusions

Conclusions

- $parallel\ I/O\ lib\ \neq good\ performance$
  - $\Rightarrow$ Careful tuning is often required (striping + collective optimizations)
- **Metadata conventions** enable straightforward pre- and post-processing and data portability between applications in general.

Piz Daint in the machine room at CSCS



Thank you for your kind attention