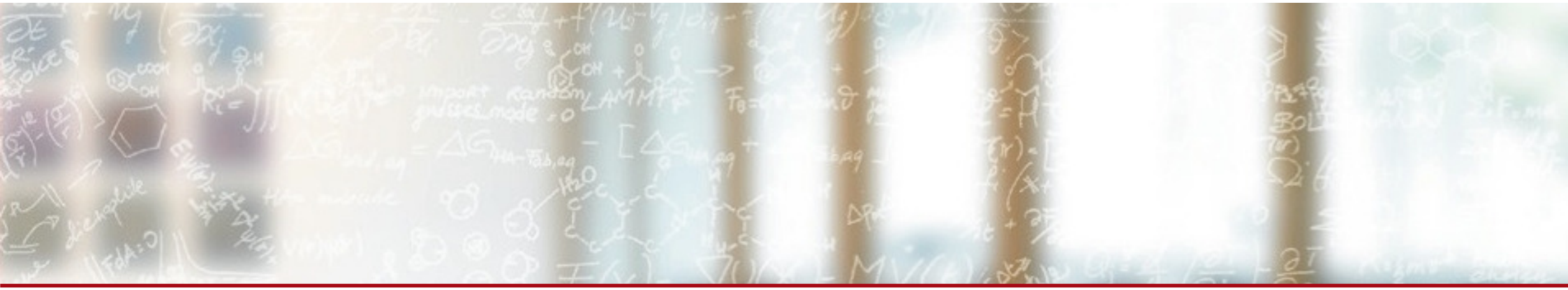
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Parallel I/O on Piz Daint

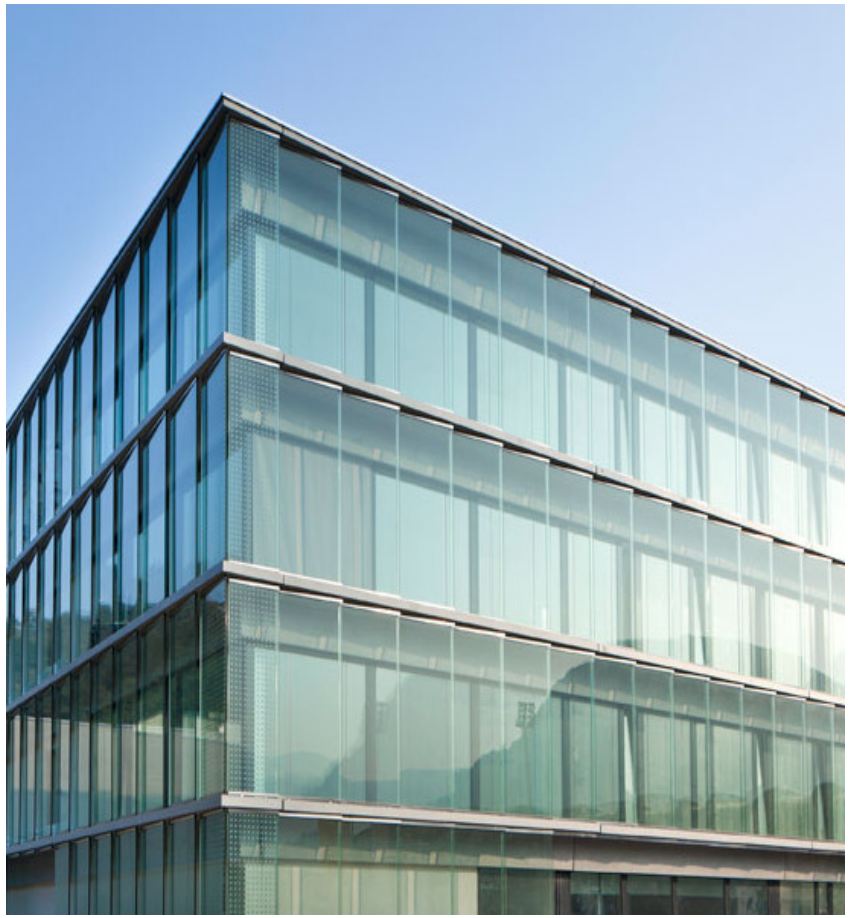
Webinar 2020

Dr. Samuel Omlin and Dr. Jean Favre, CSCS

October 22nd 2020

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
- ADIOS2 – a recent parallel “I/O” library
  - accessing the file system in parallel
  - selecting different file formats
  - including on-the-fly lossy or lossless compression
  - staging simulation output for in-situ visualization/monitoring
- Conclusions



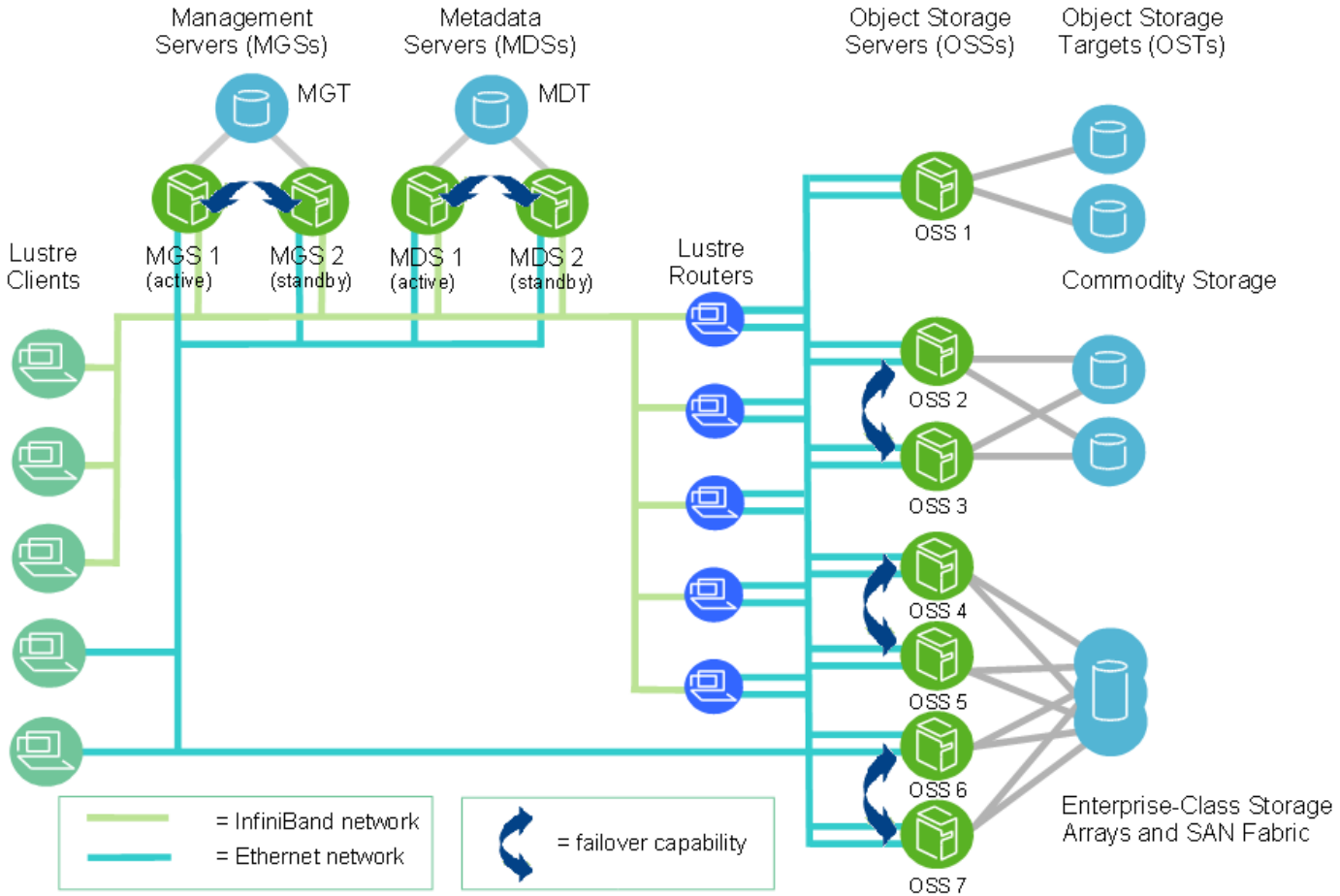
CSCS office building in Lugano

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

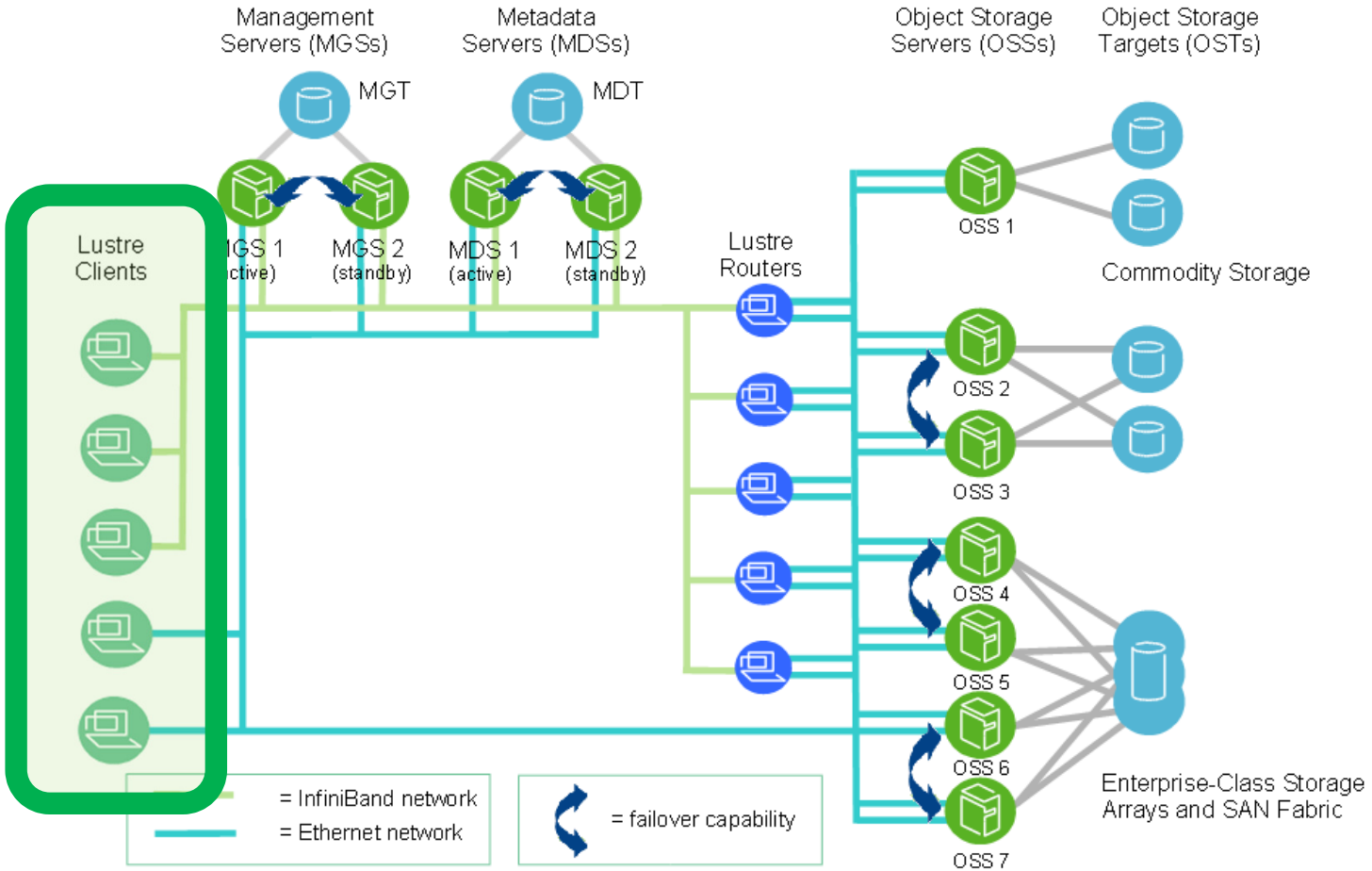
ETH zürich

Introduction to I/O on Piz Daint

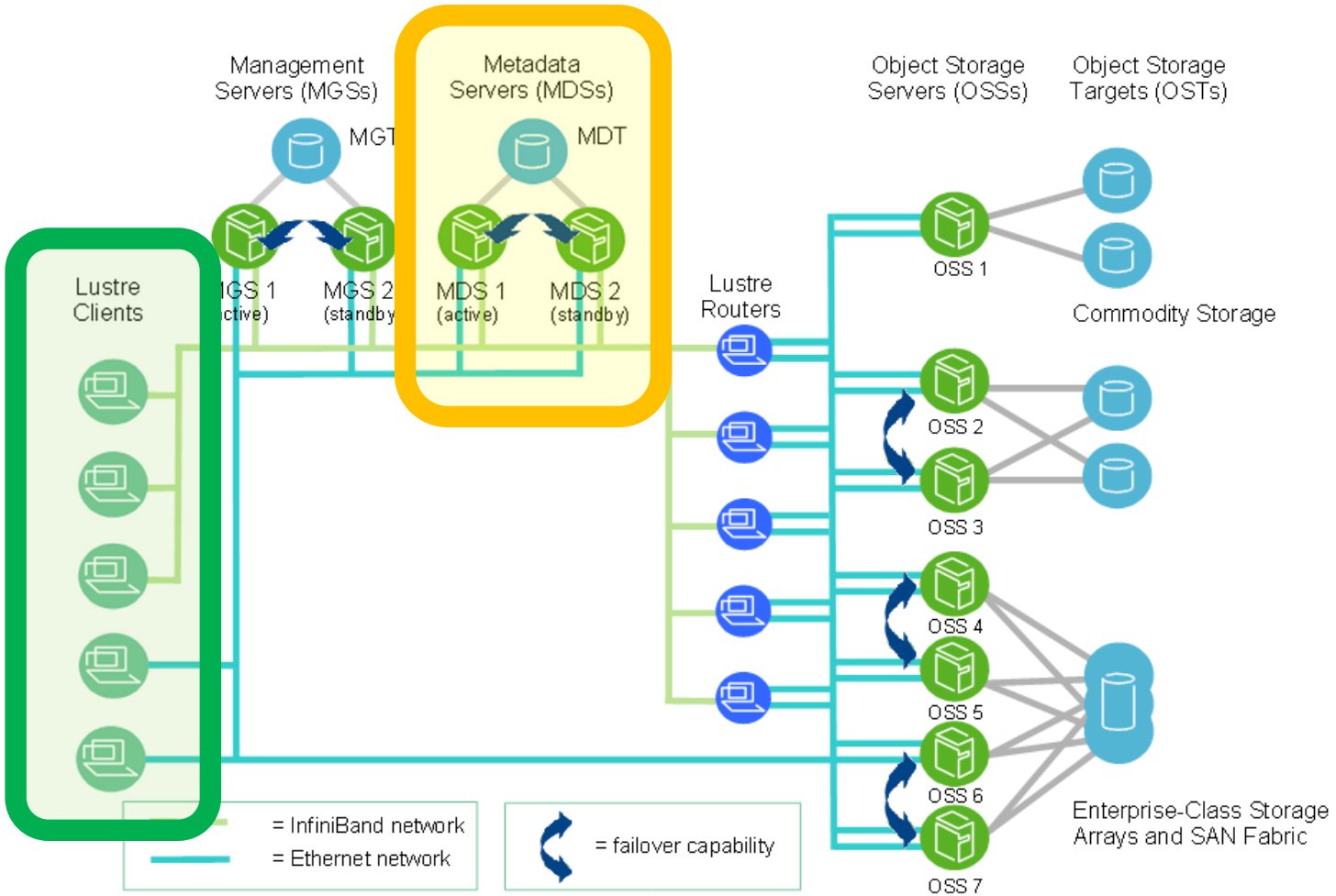
Scratch – a Lustre file system



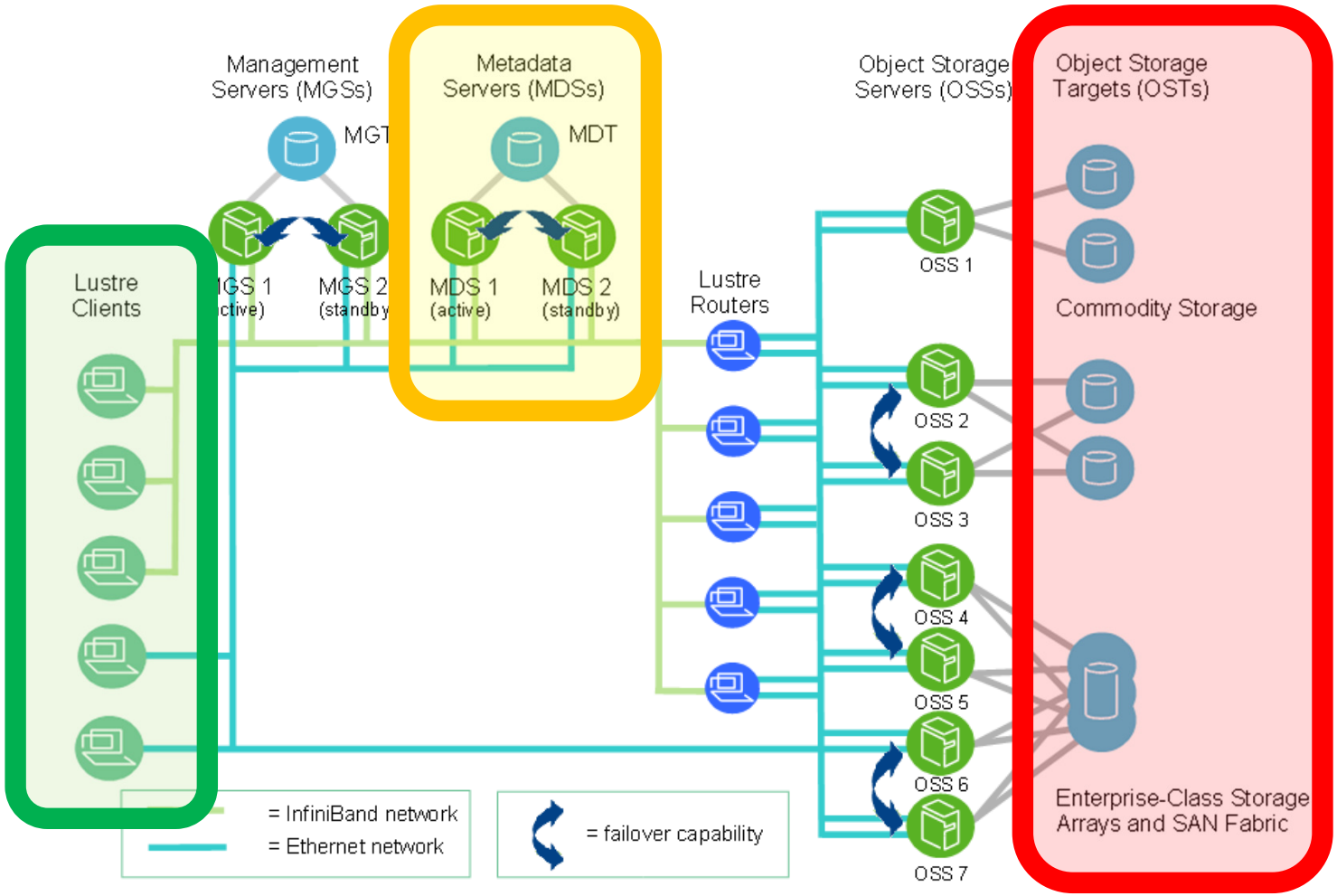
Scratch – a Lustre file system



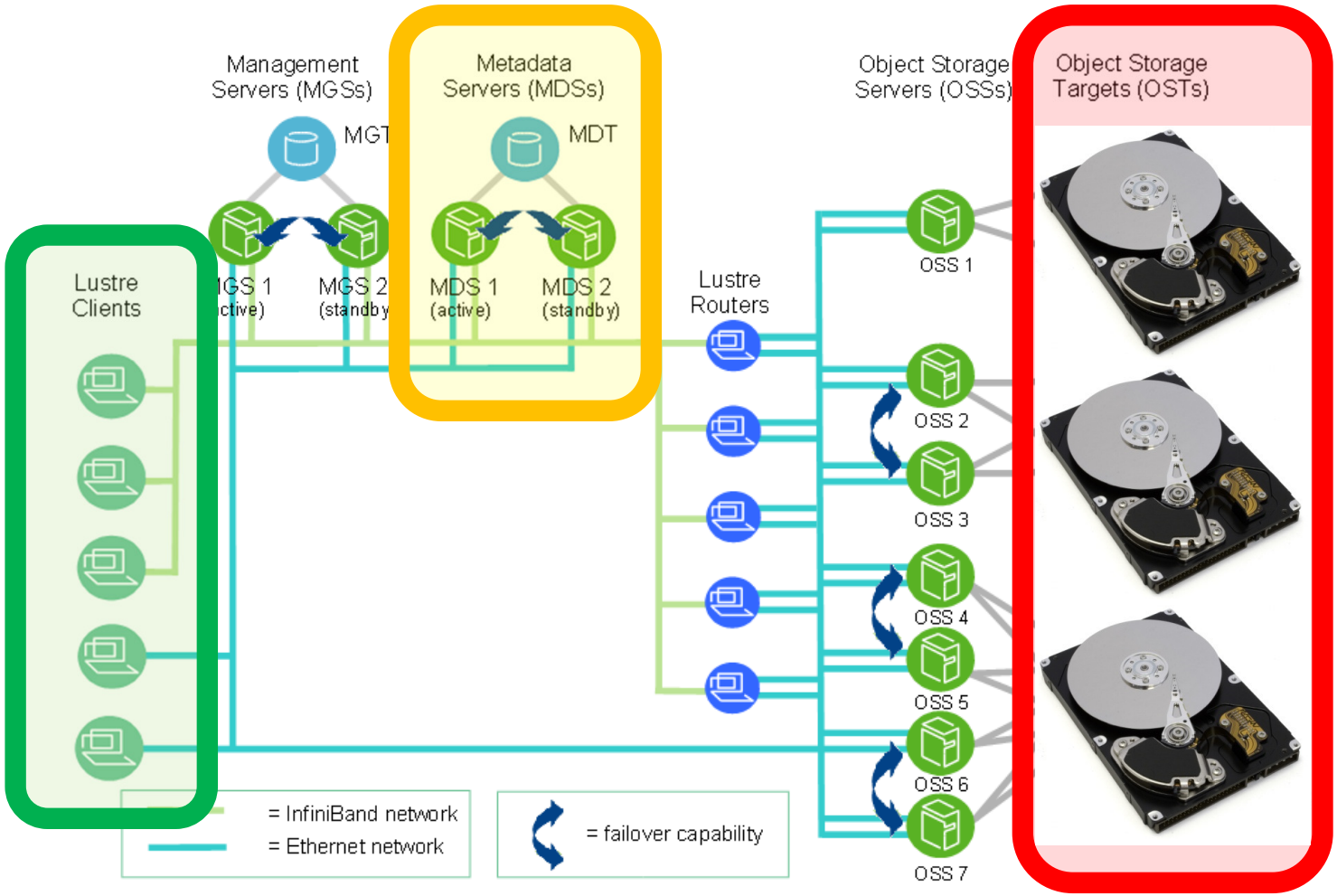
Scratch – a Lustre file system



Scratch – a Lustre file system



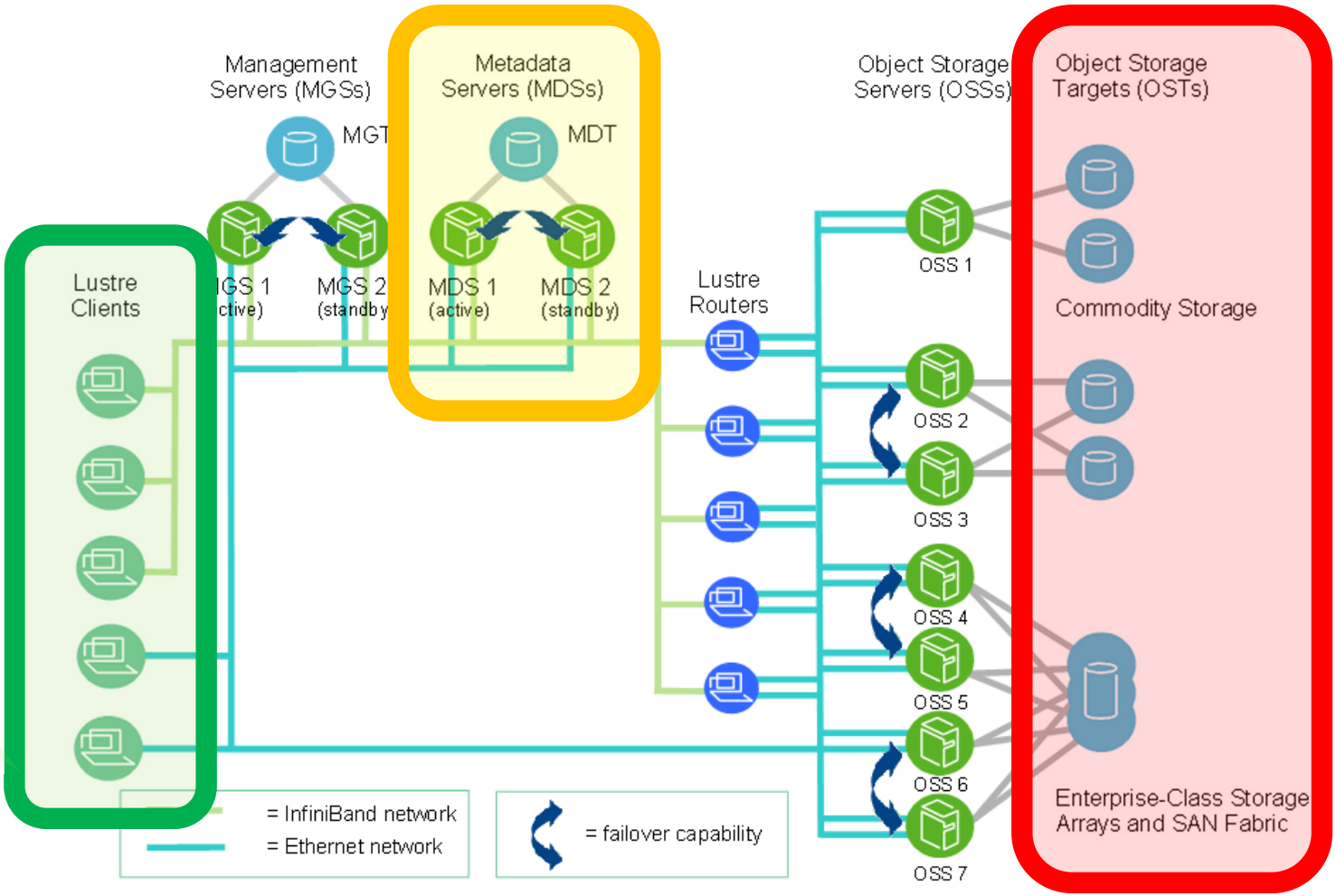
Scratch – a Lustre file system



Scratch – a Lustre file system

$SCRATCH:
/scratch/snx3000/$USER

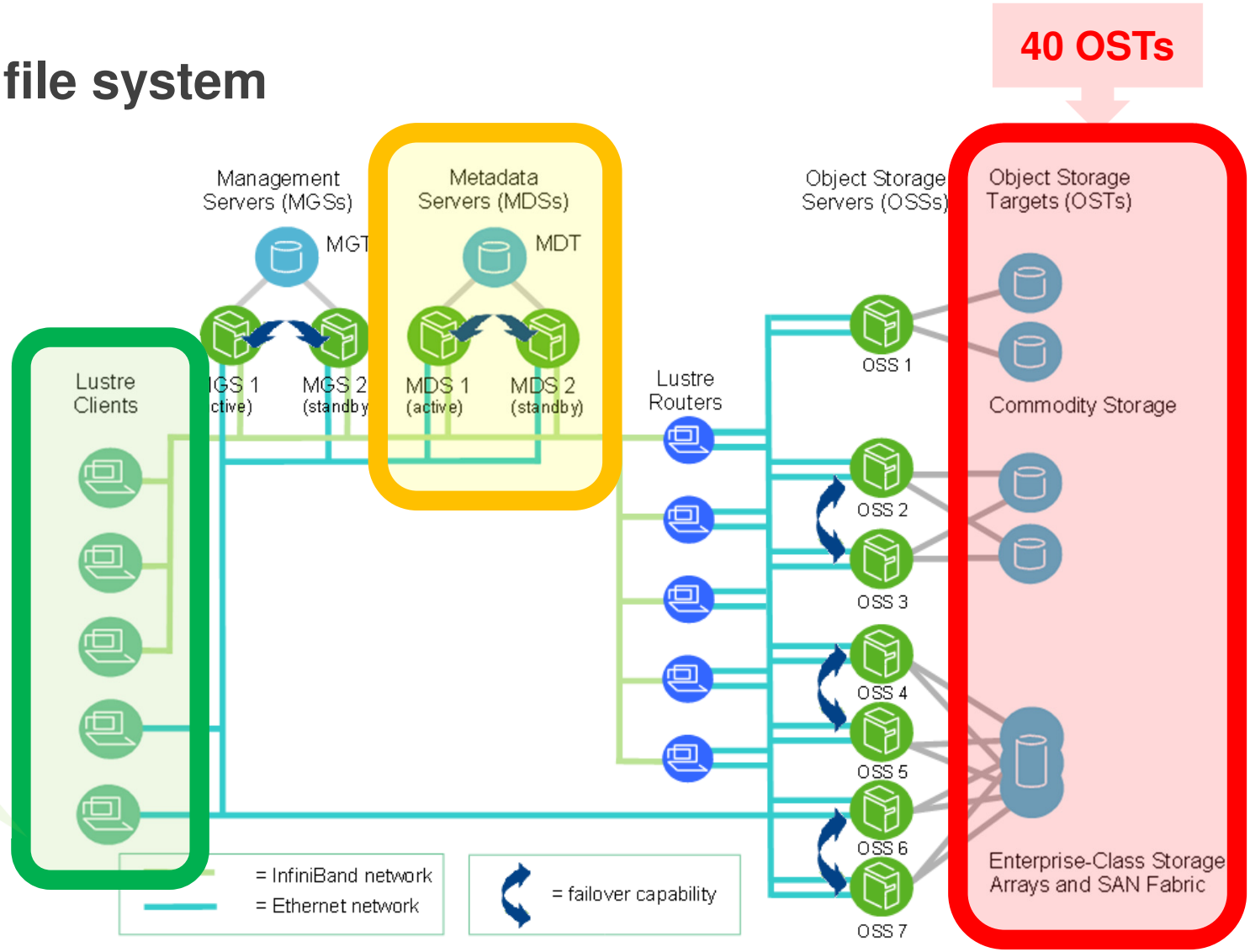
snx3000:
Cray Sonexion 3000
(>100 GB/s)



Scratch – a Lustre file system

$SCRATCH:
/scratch/snx3000/$USER

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

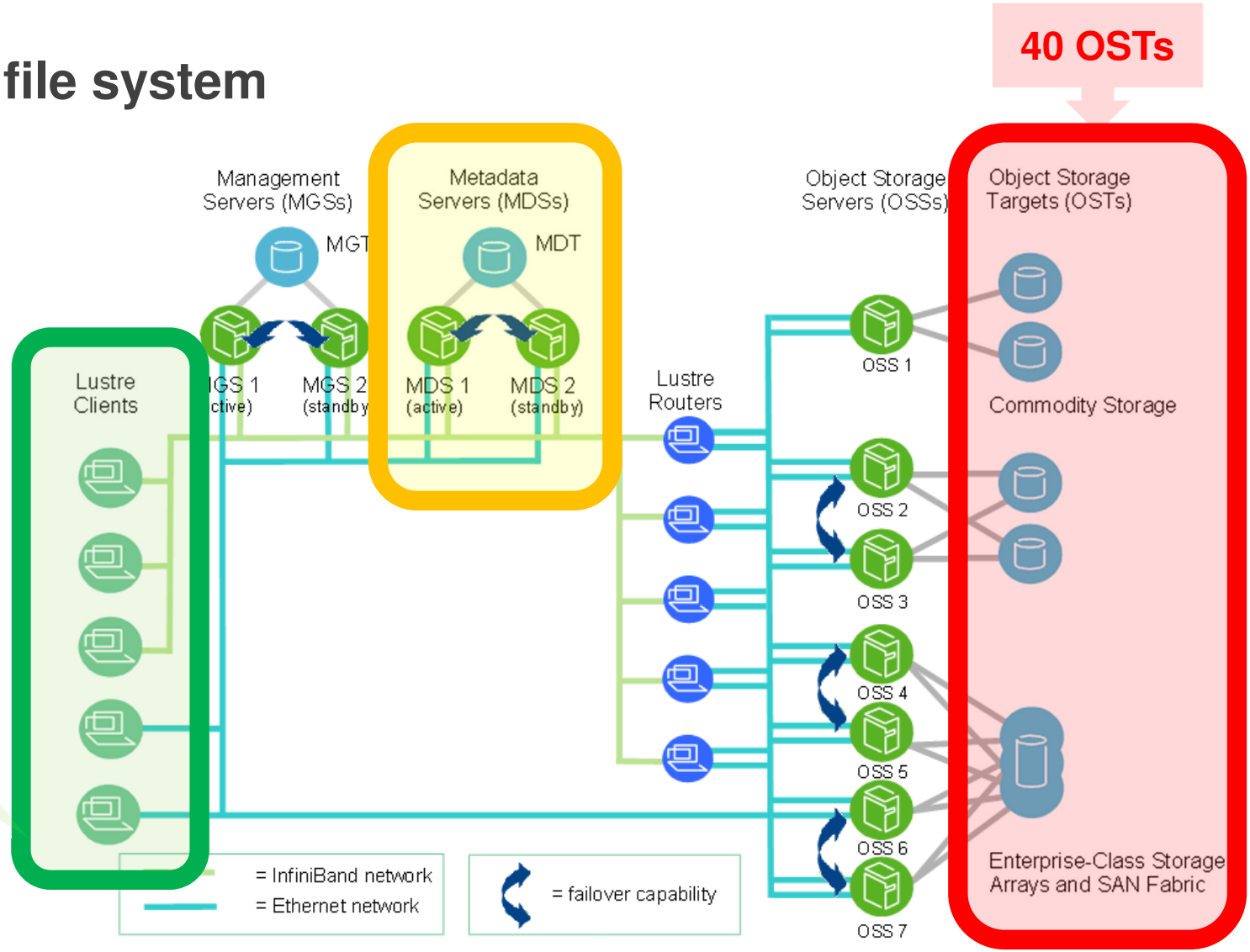


Scratch – a Lustre file system

$SCRATCH:
/scratch/snx3000/$USER

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

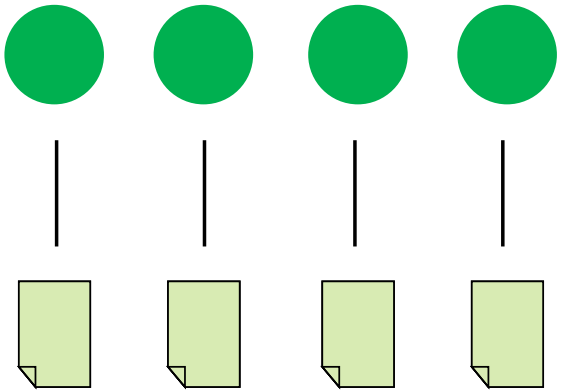
Only possible if all or most OSTs accessed => parallel I/O



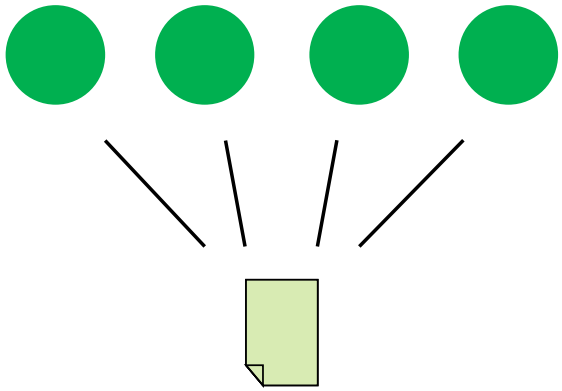
Parallel I/O? – Common approaches

Parallel I/O? – Common approaches

- File-per-process

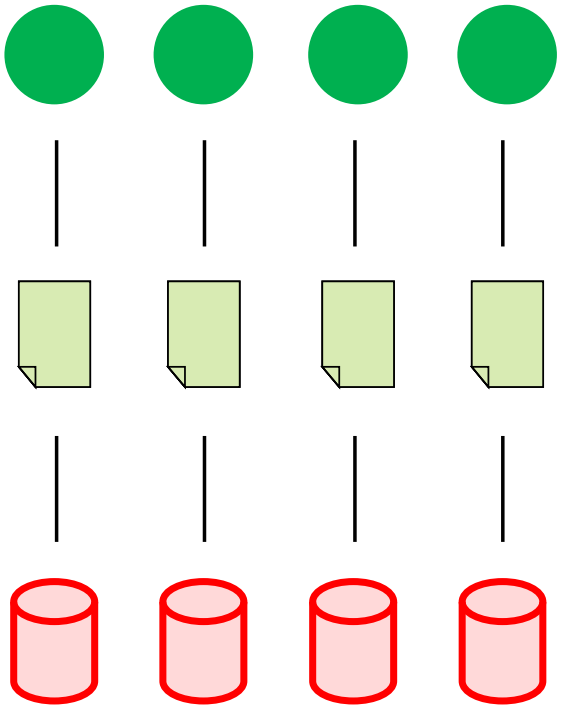


- Shared file

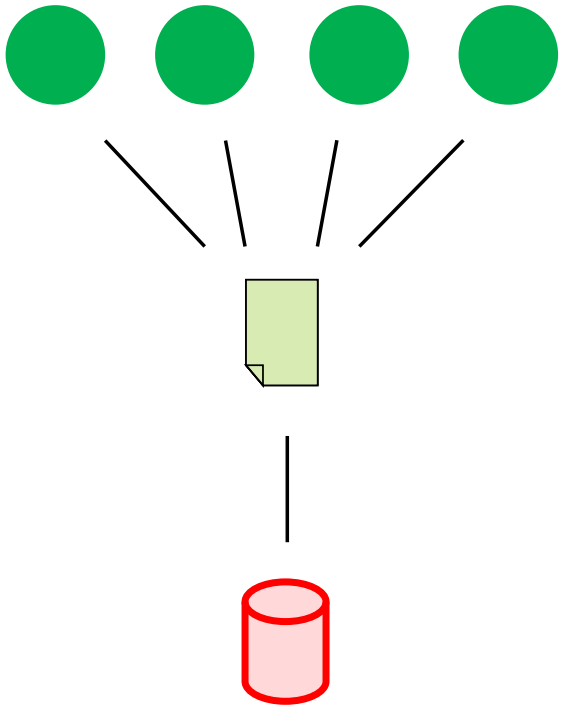


Parallel I/O? – Common approaches

- File-per-process

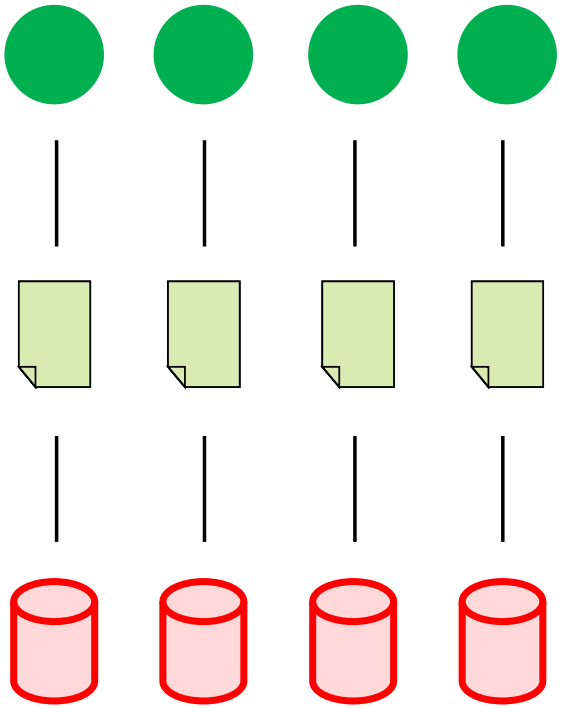


- Shared file

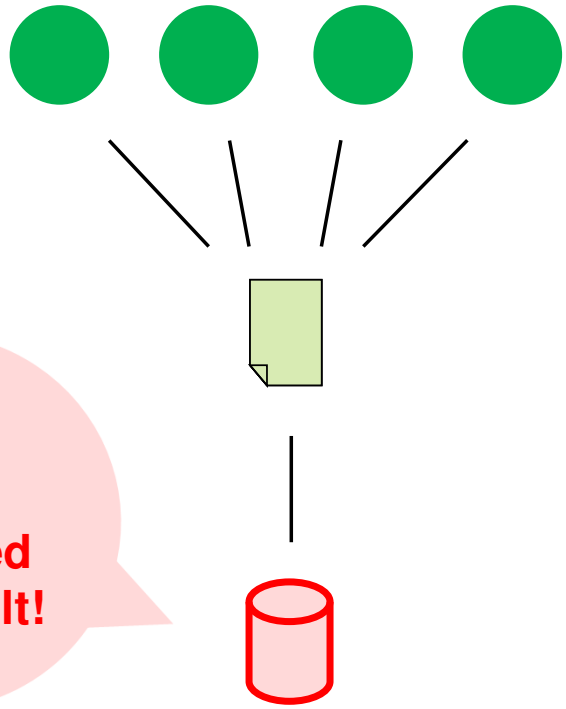


Parallel I/O? – Common approaches

- File-per-process

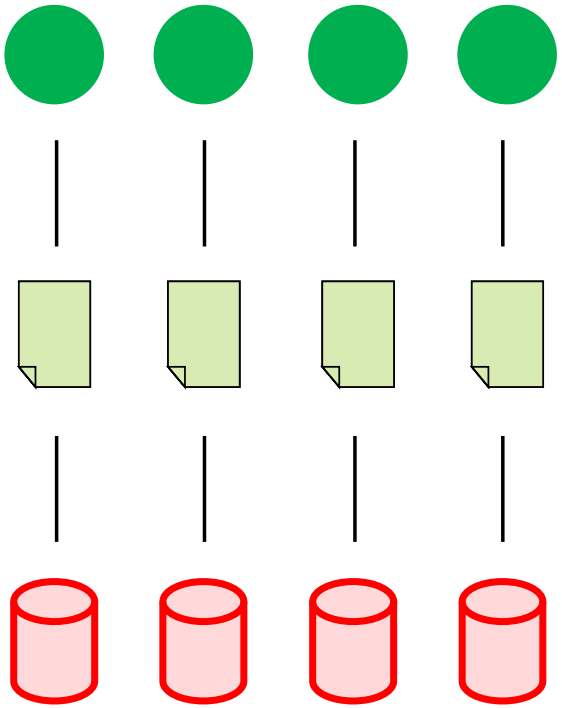


- Shared file

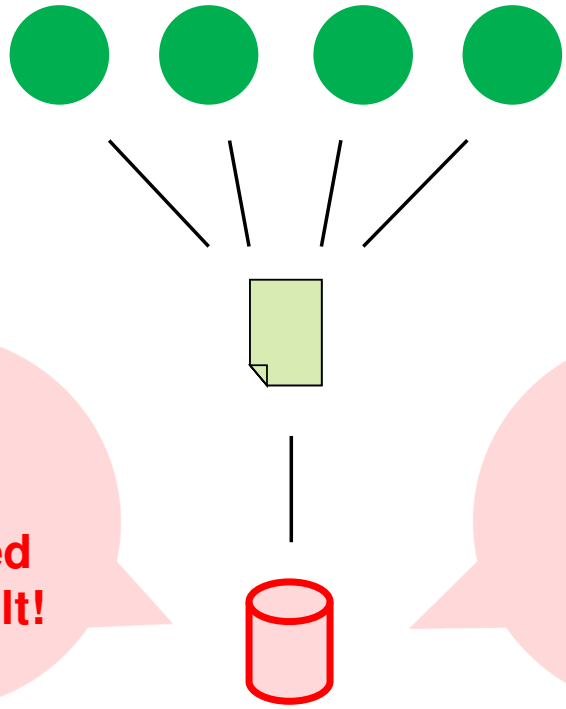


Parallel I/O? – Common approaches

- File-per-process

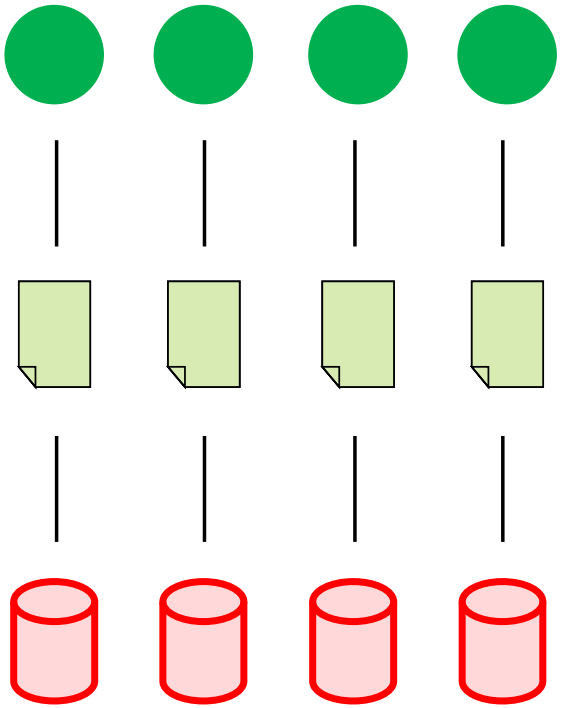


- Shared file

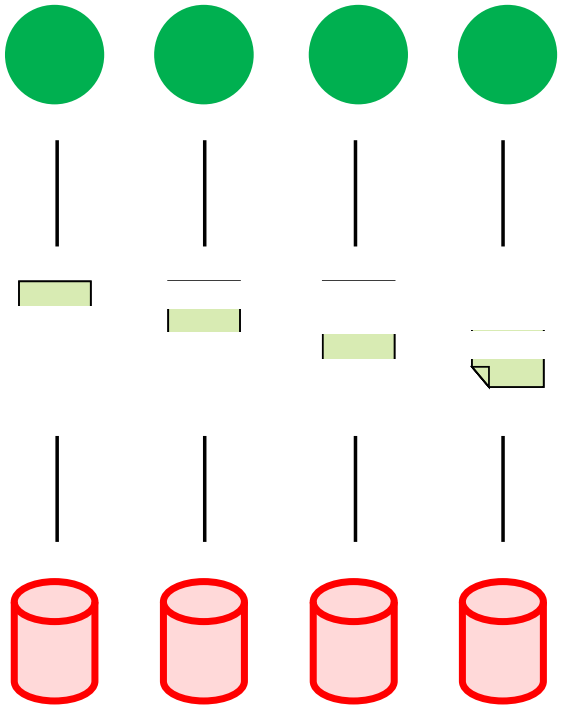


Parallel I/O? – Common approaches

- File-per-process

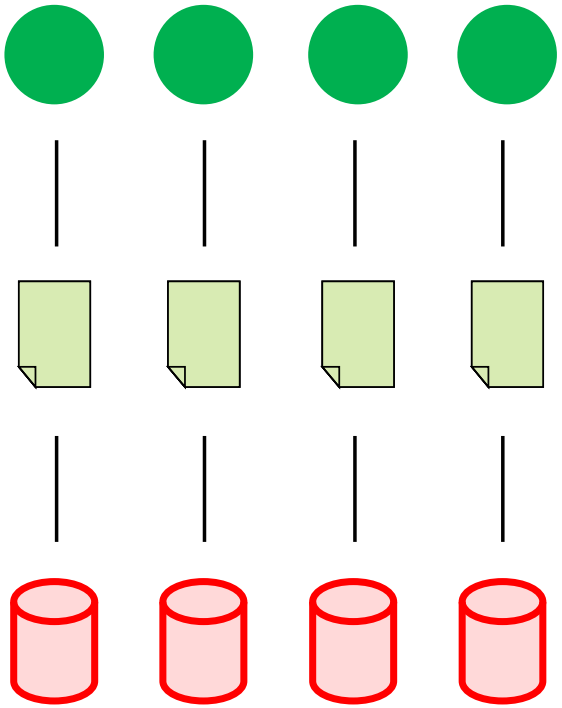


- Shared file (with striping)

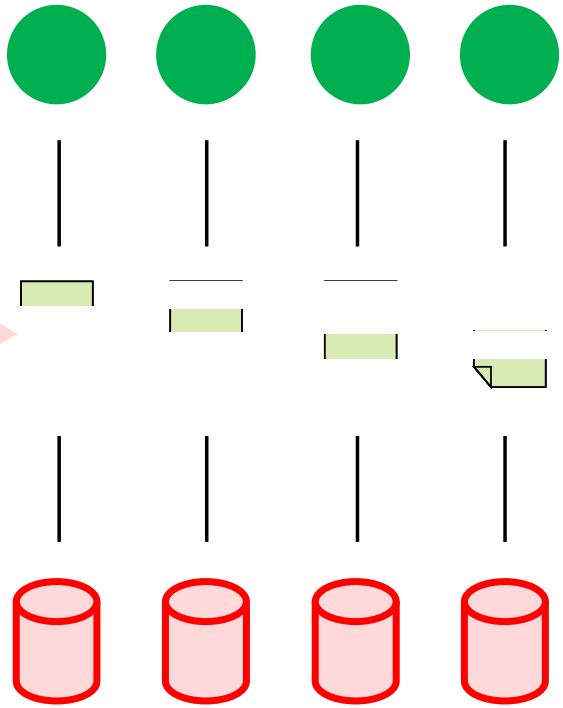


Parallel I/O? – Common approaches

- File-per-process



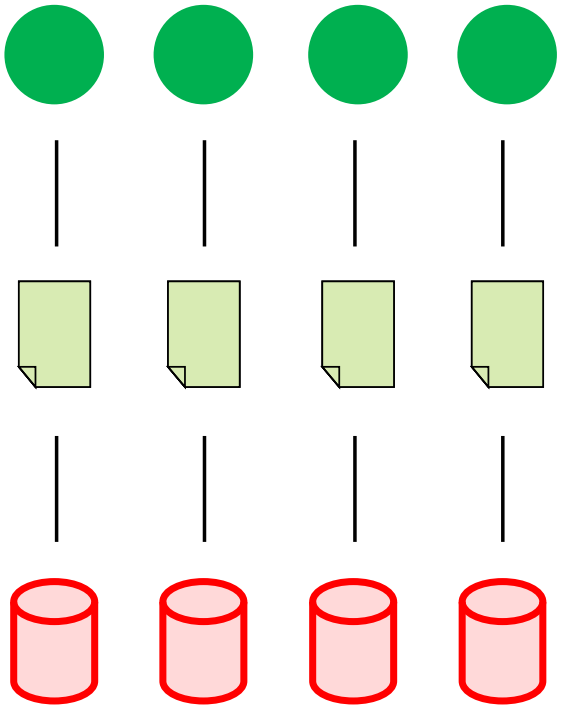
- Shared file (with striping)



Stripe count = 4 (distribute file on 4 OSTs)

Parallel I/O? – Common approaches

- File-per-process

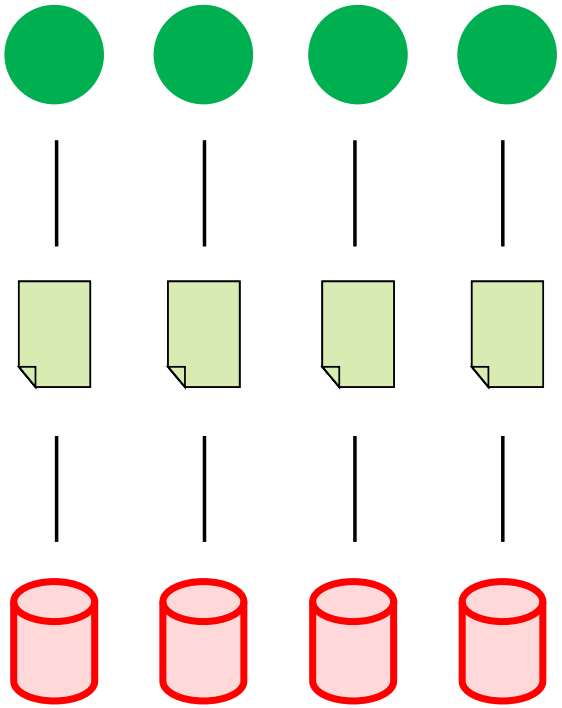


Maximal simplicity is possible on the application level (no I/O library is needed).

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

Parallel I/O? – Common approaches

- File-per-process



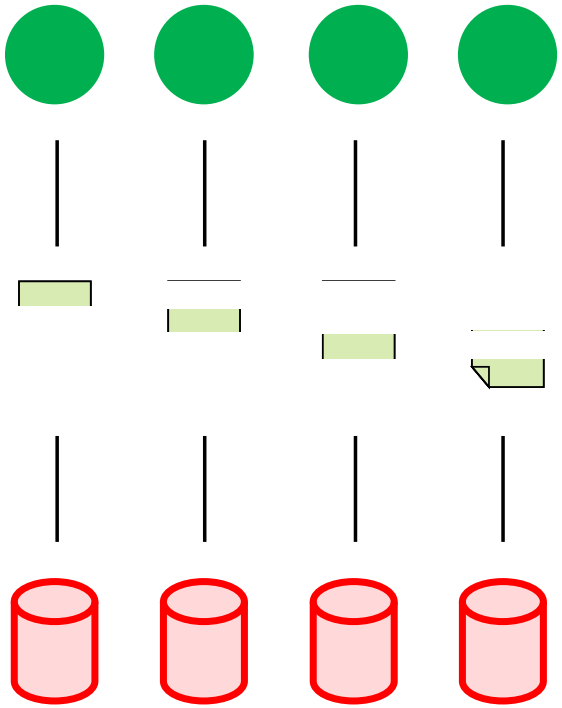
General recommendations

- 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.

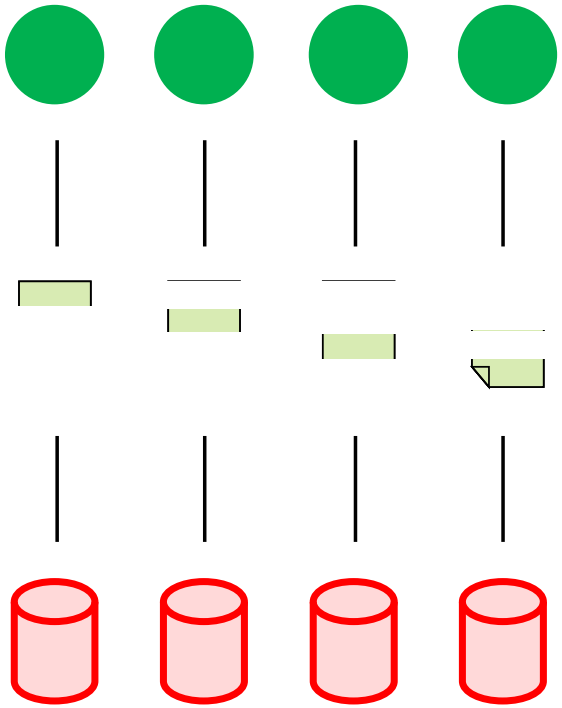
👎 **Careful tuning** is often required to reach good I/O performance.

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)

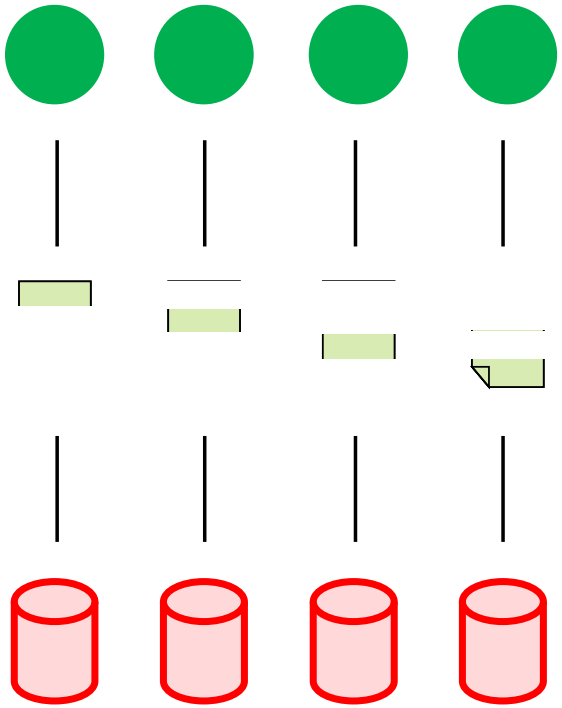


Parallel I/O? – Common approaches

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)

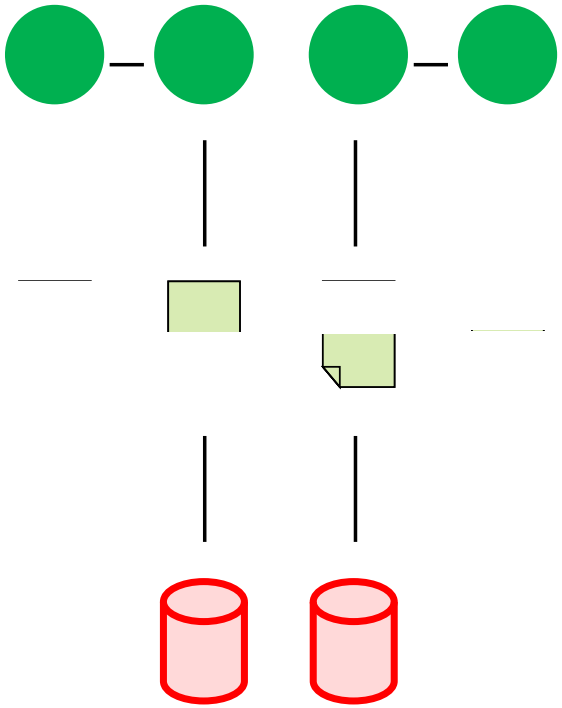
- Use **collective I/O operations** (enable merging of I/O requests of different processes into fewer larger ones).

  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)

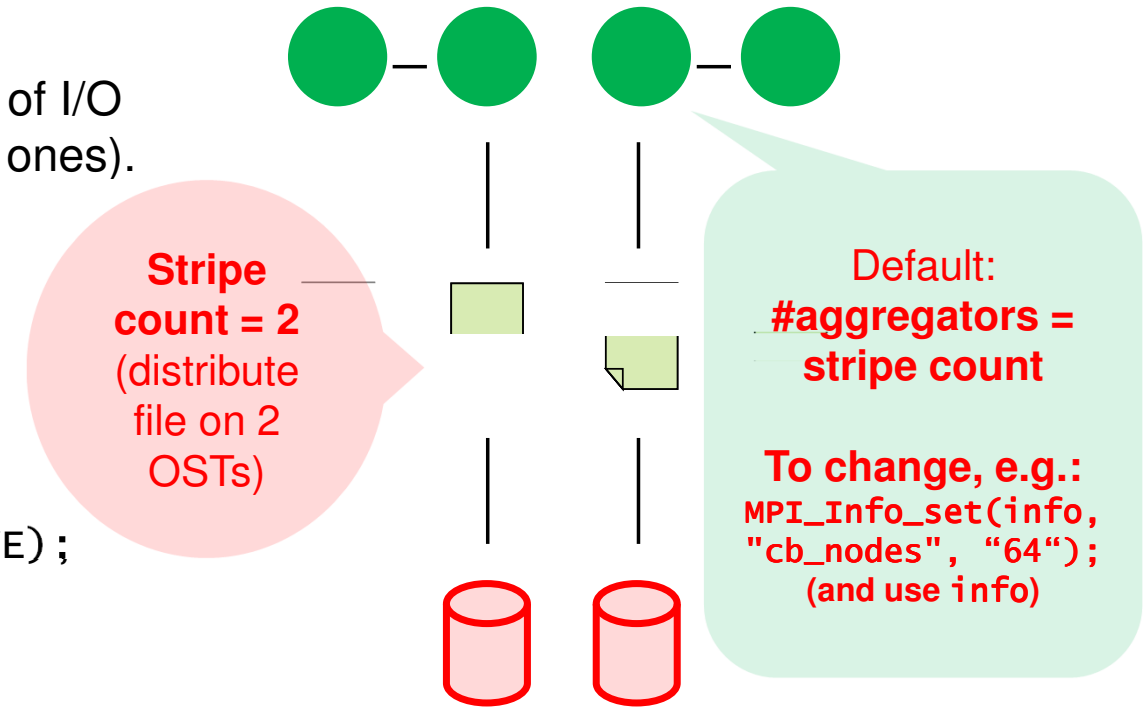
- Use **collective I/O operations** (enable merging of I/O requests of different processes into fewer larger ones).

  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)

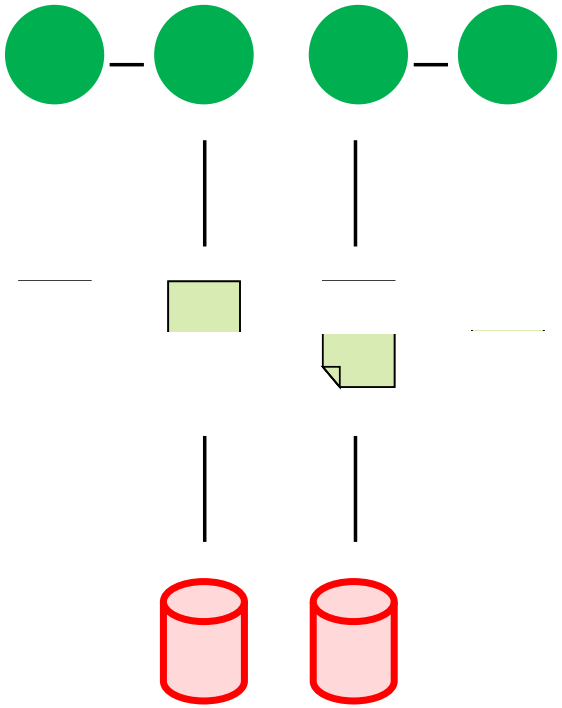


Parallel I/O? – Common approaches

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)



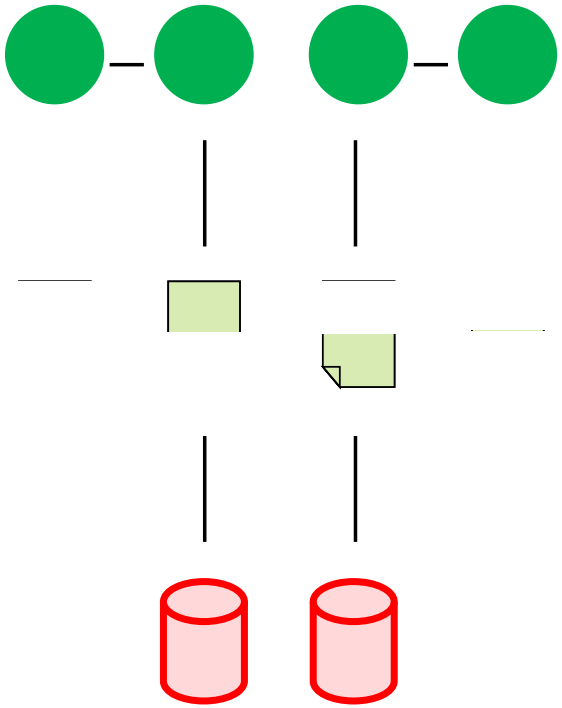
Parallel I/O? – Common approaches

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/

- 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)

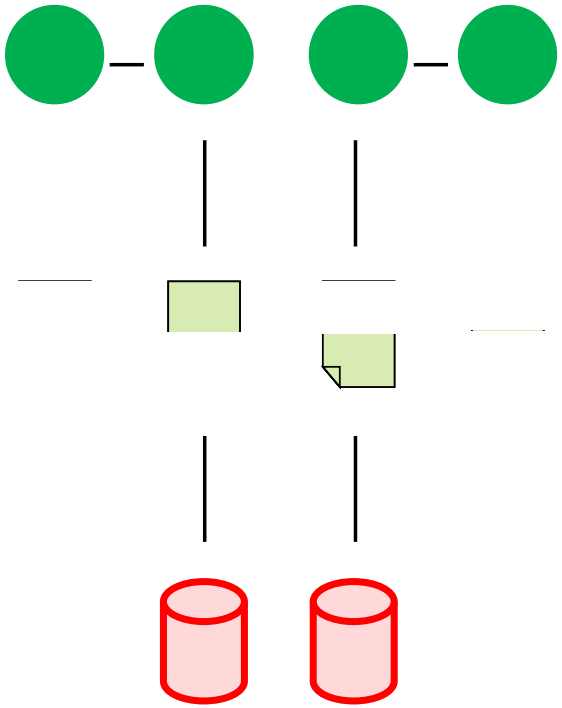


Parallel I/O? – Common approaches

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)



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

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.**

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
- Do no
- **Avoid**
- Avoid
- Avoid
- Read
- **Limit** the us
- **Be aw slowd**

How about we say "ADIOS" to most of that…

… and let some experts take car of it?

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

ADIOS2 – a recent parallel “I/O” library

ADIOS2 – a recent parallel “I/O” library

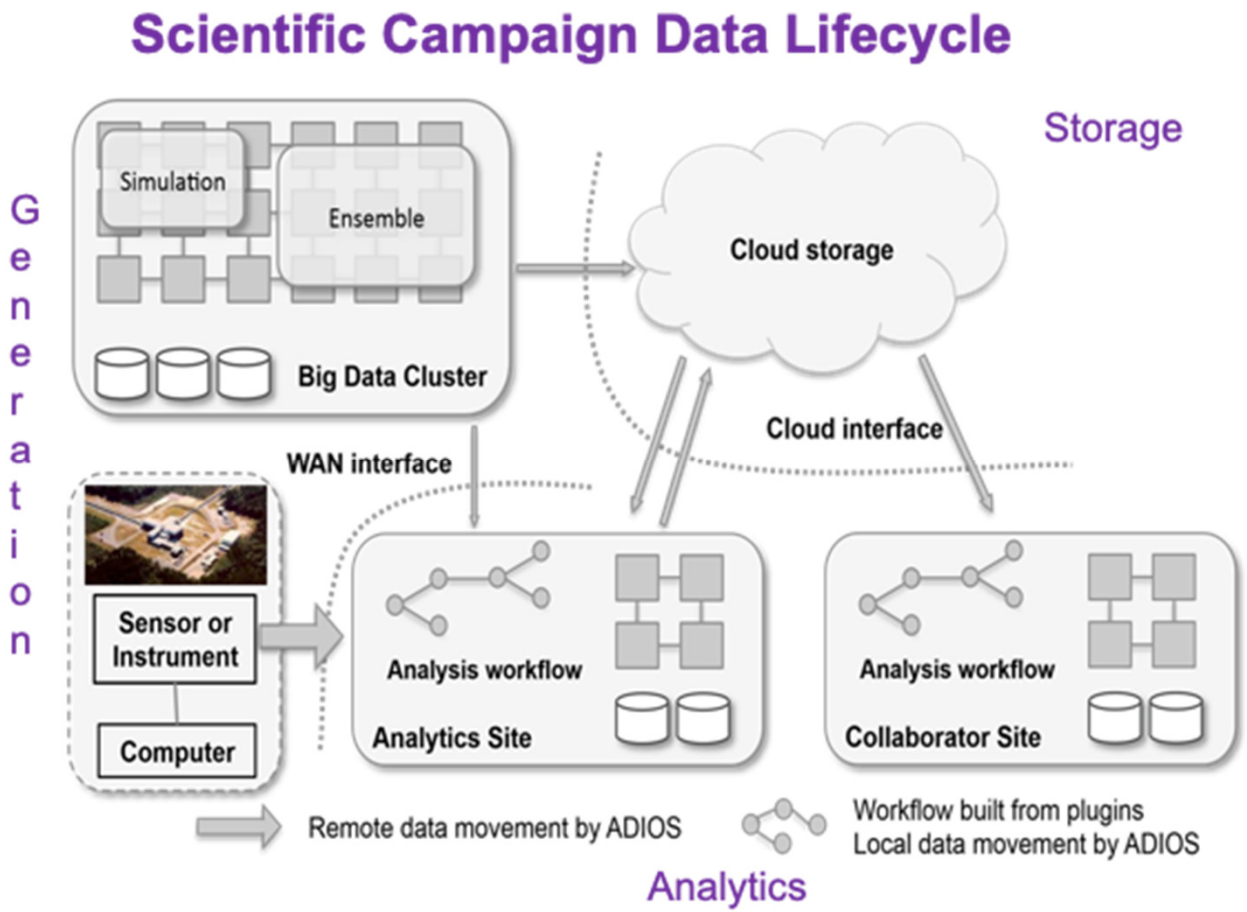


Image from https://adios2.readthedocs.io

ADIOS2 – a recent parallel “I/O” library

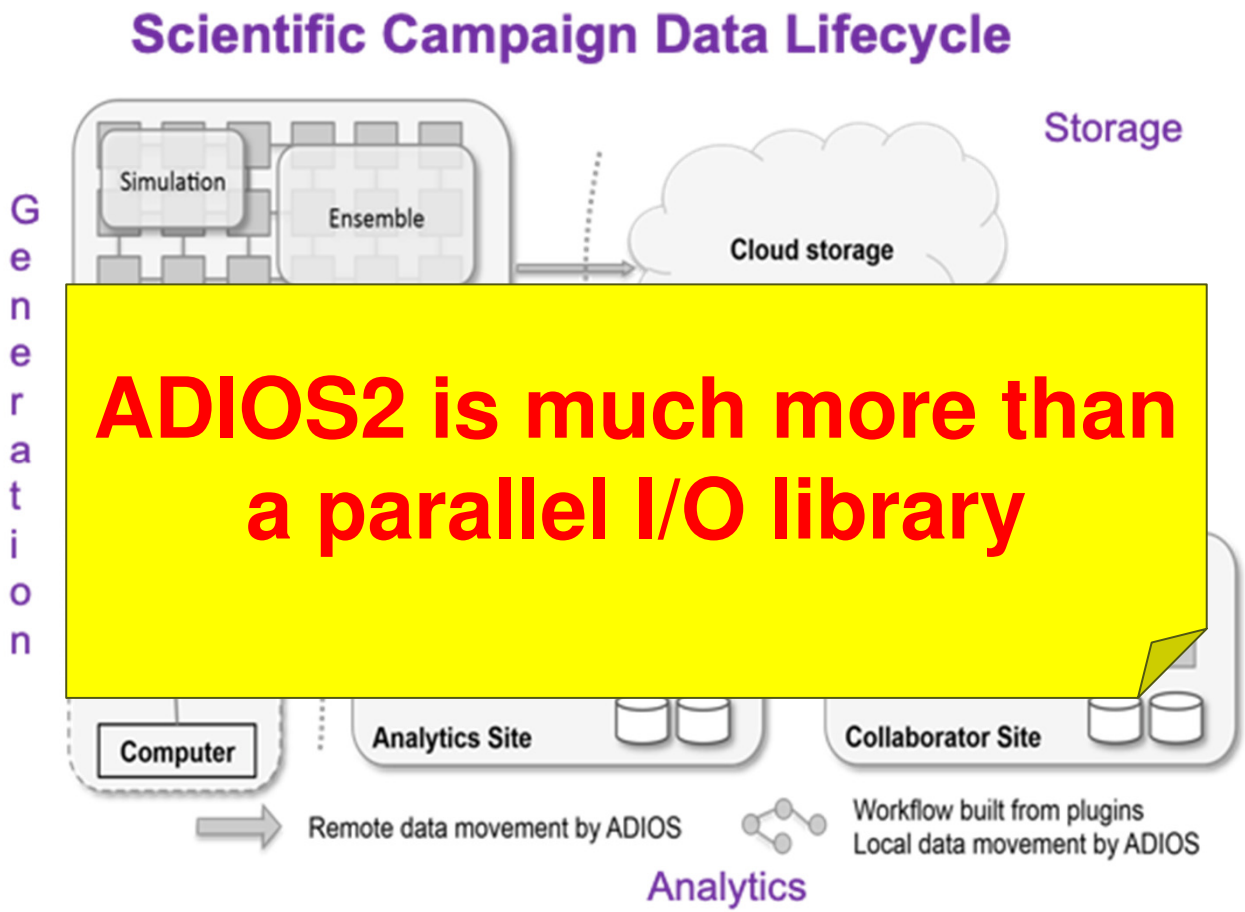


Image from https://adios2.readthedocs.io

ADIOS2 – a recent parallel “I/O” library

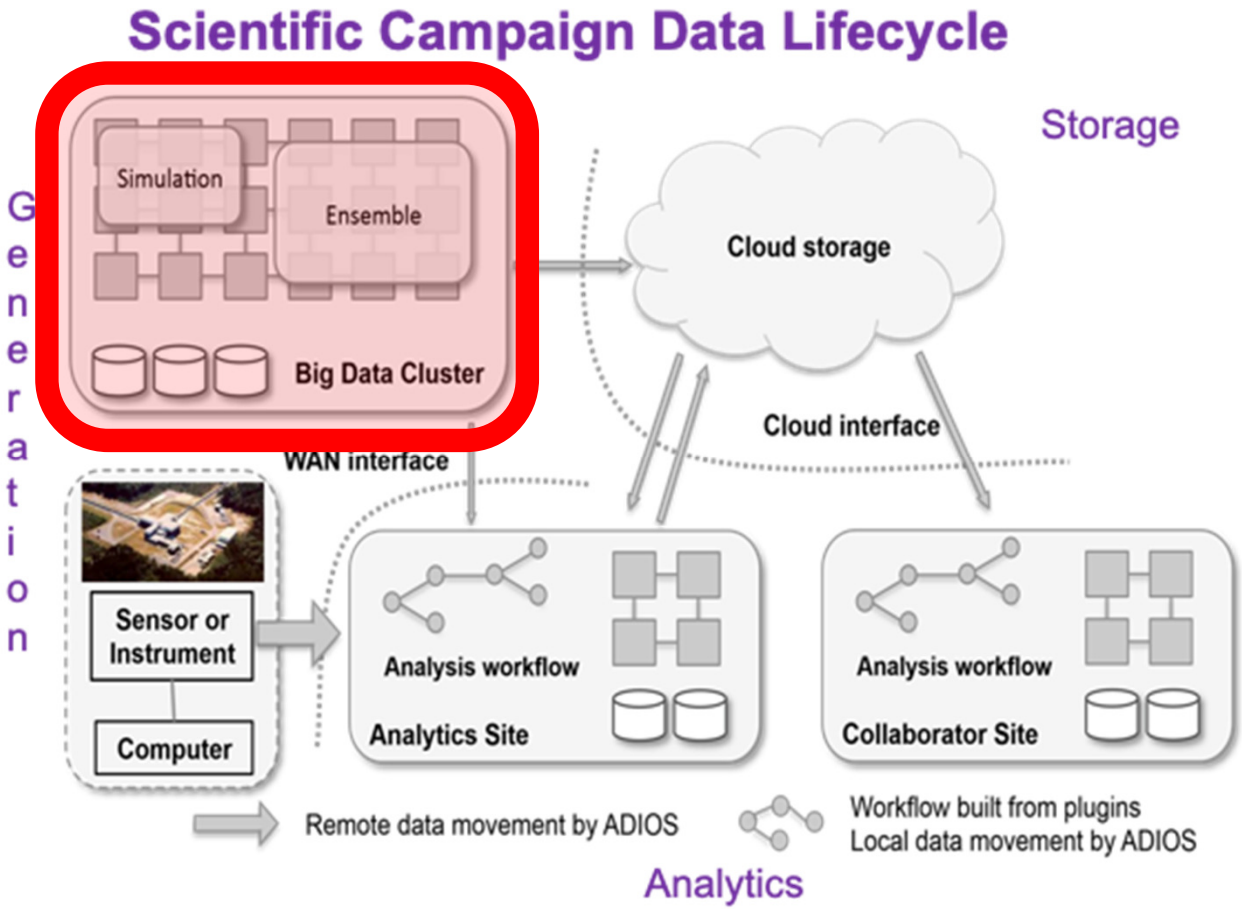
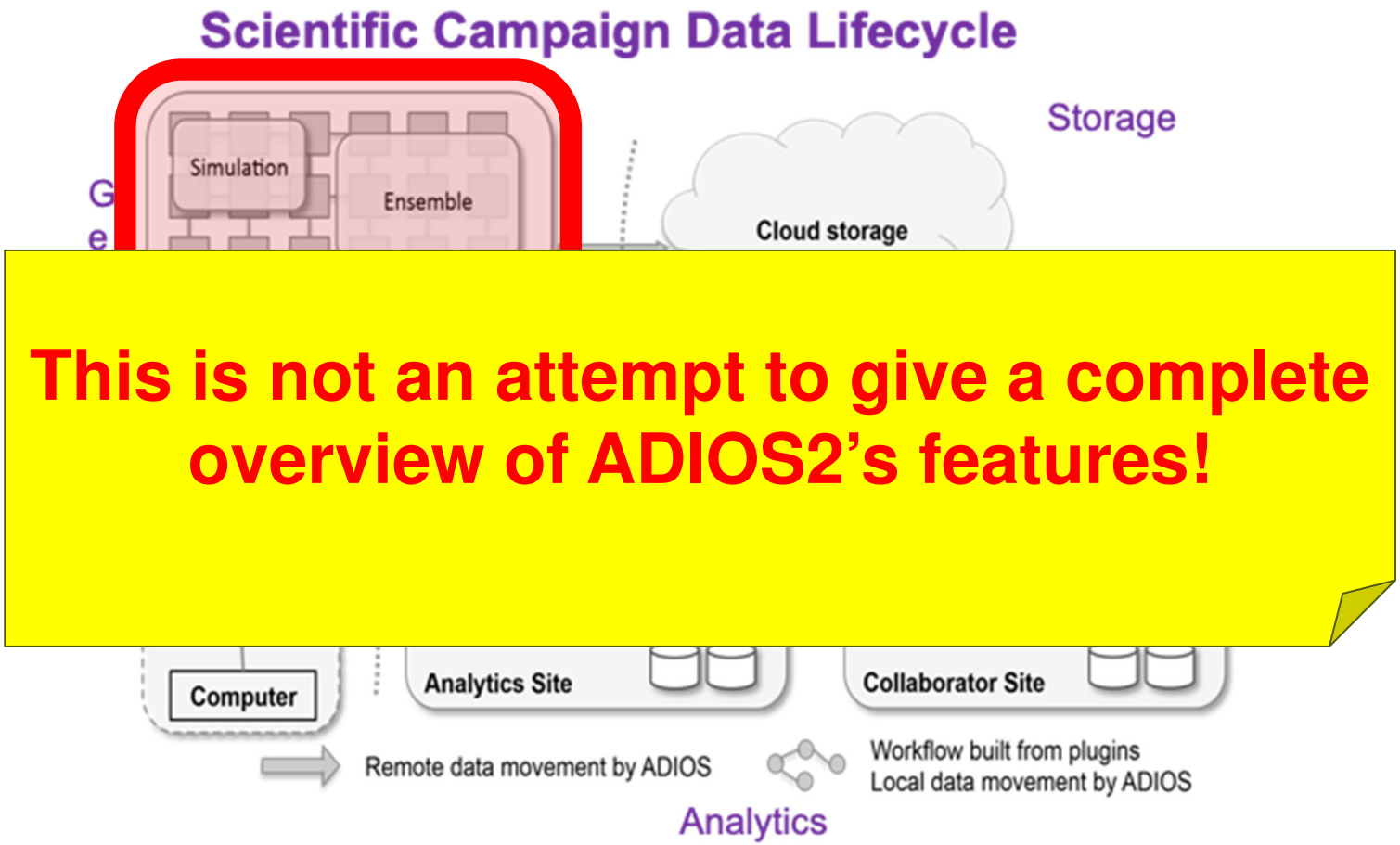


Image from https://adios2.readthedocs.io

ADIOS2 – a recent parallel “I/O” library



This is not an attempt to give a complete overview of ADIOS2’s features!

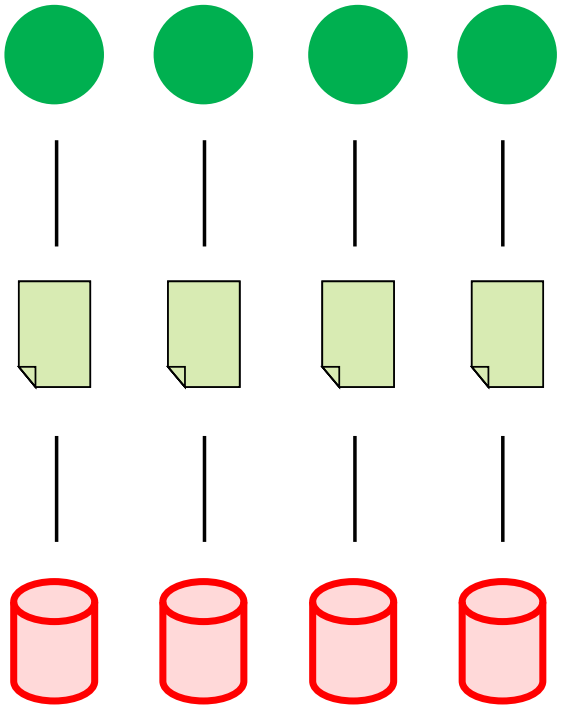
Image from https://adios2.readthedocs.io

ADIOS2 – a recent parallel “I/O” library

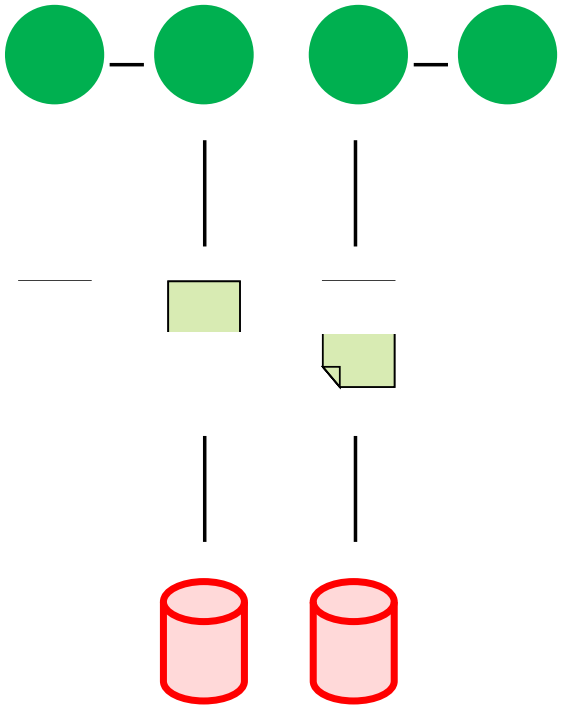
- **Much more than file I/O**: enables a variety of data staging scenarios for in-situ visualization, monitoring or analysis (on the same nodes or different nodes as the simulation or even on an external machine…) and on-the-fly lossy or loss-less compression
- **Highly declarative**: let experts take care of details
- **Unified API for file I/O and data staging**: switch by changing one configuration file entry!
- **Bindings** available for C, C++, Fortran and Python
- Developed as part of the United States Department of Energy's Exascale Computing Project

ADIOS2 – file-per-process or shared file approach?

- File-per-process

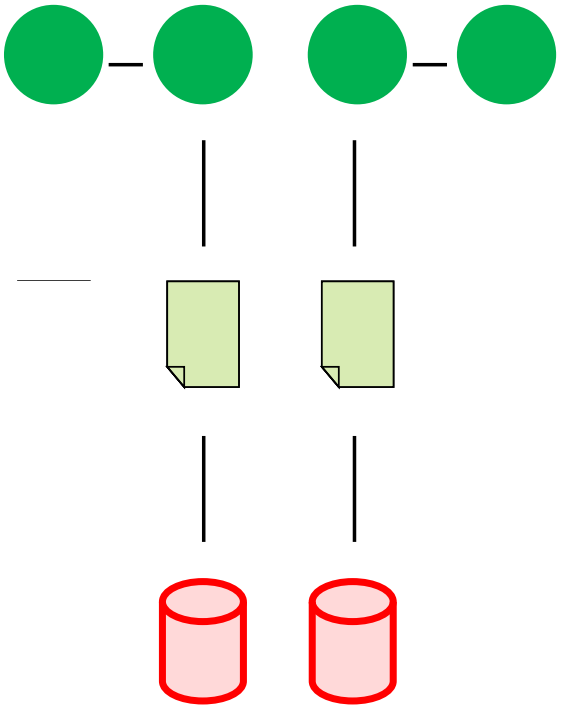


- Shared file (with striping + collective buffering)



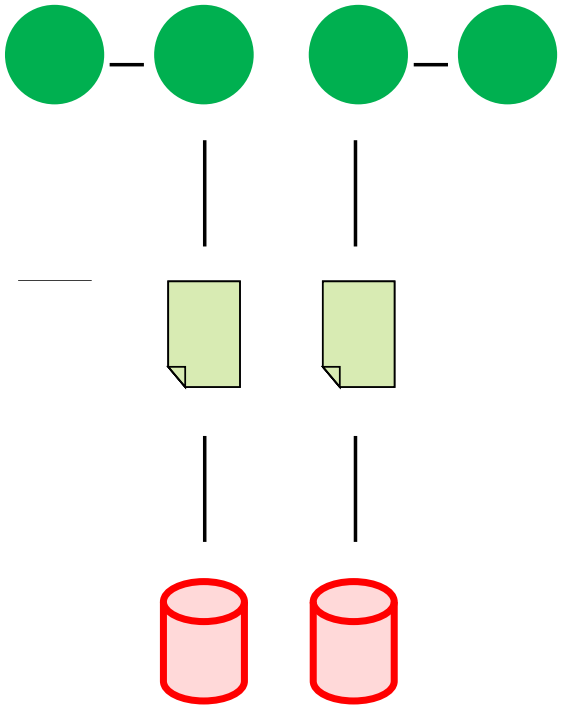
ADIOS2 – file-per-process or shared file approach?

- File-per-aggregator (collective buffering)



ADIOS2 – file-per-process or shared file approach?

- File-per-aggregator (collective buffering)



- One free parameter to optimize I/O: number of aggregators
- An ADIOS2 data set is a folder rather than a file

ADIOS2 – design

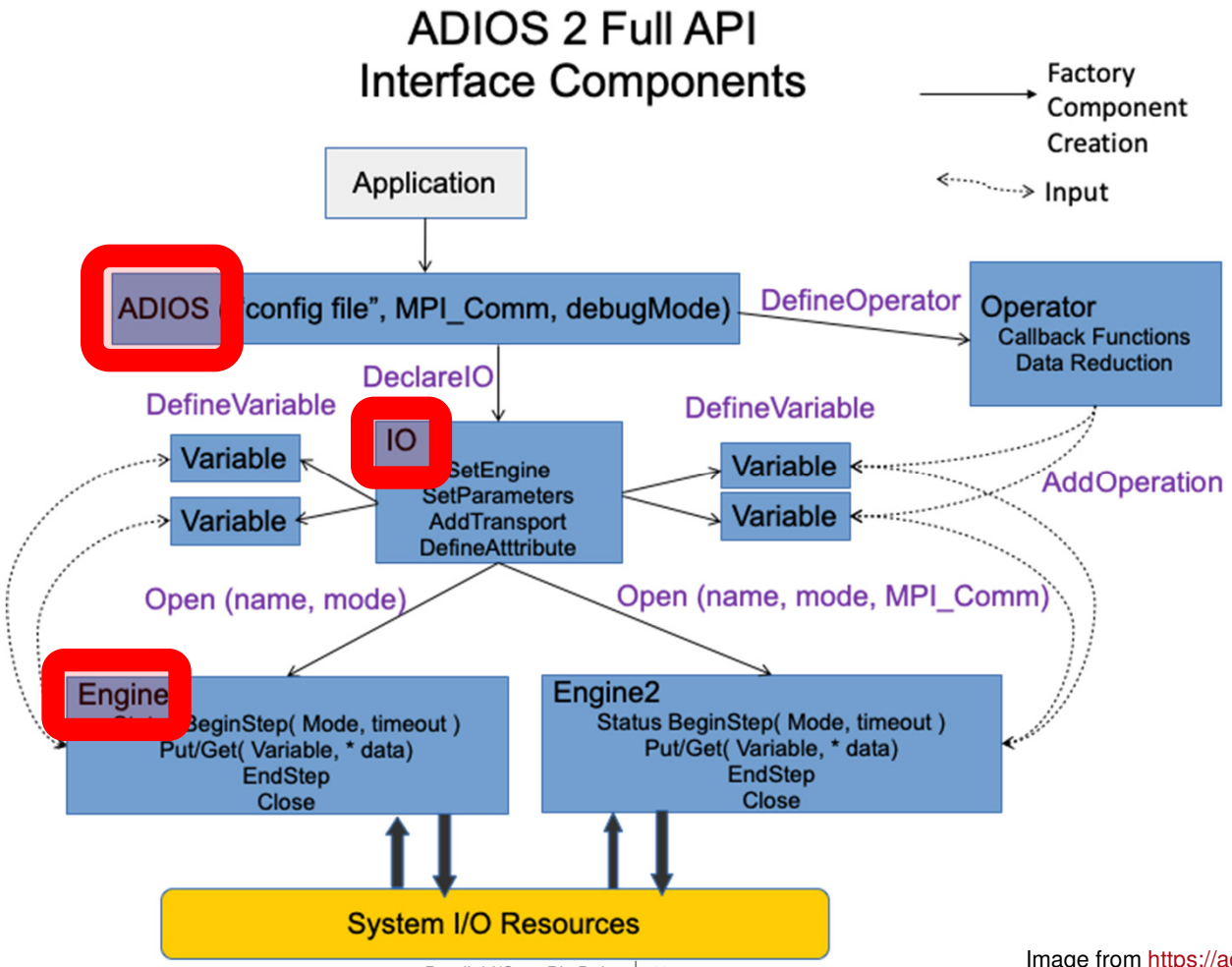


Image from https://adios2.readthedocs.io

ADIOS2 – design

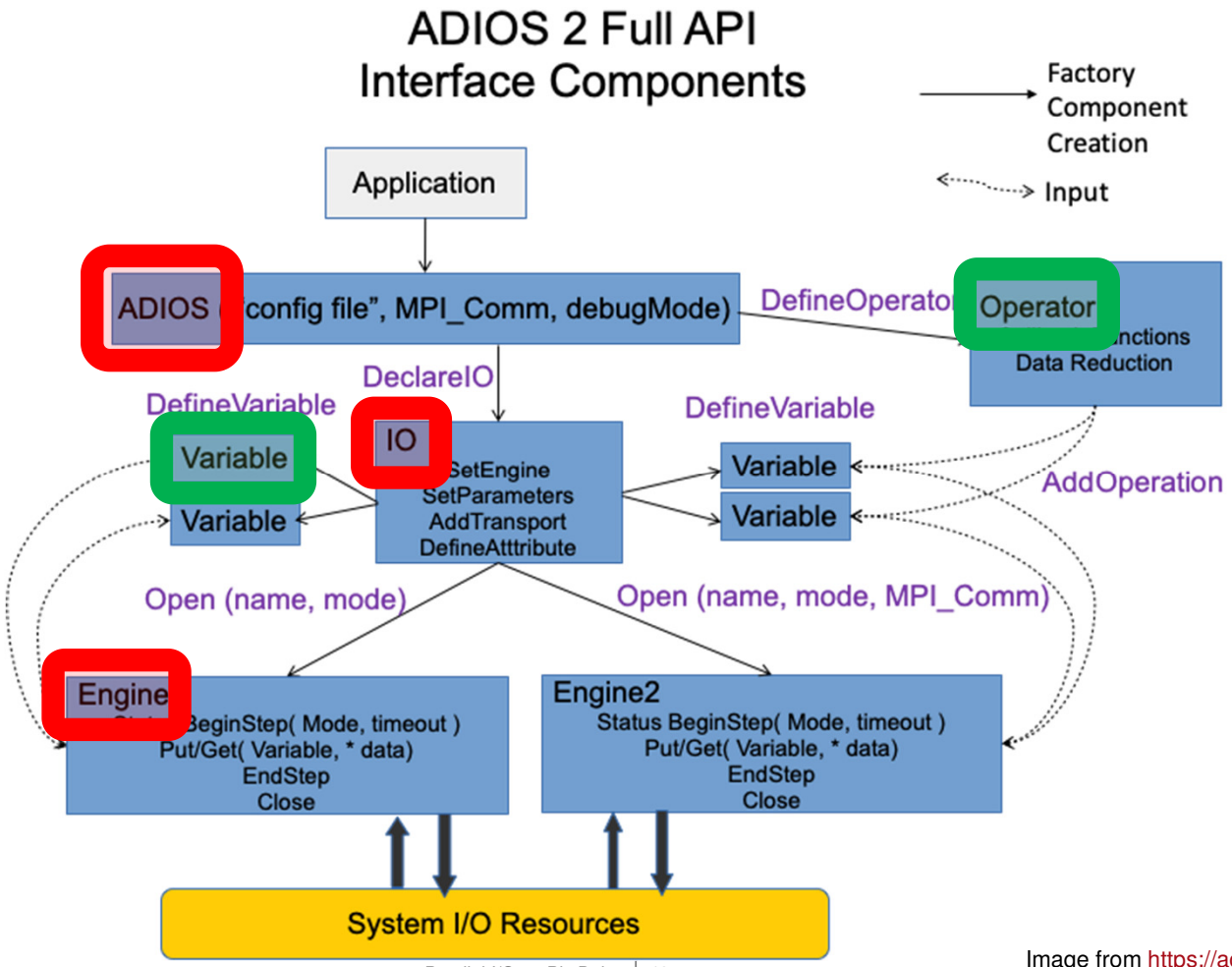


Image from https://adios2.readthedocs.io

ADIOS2 – design

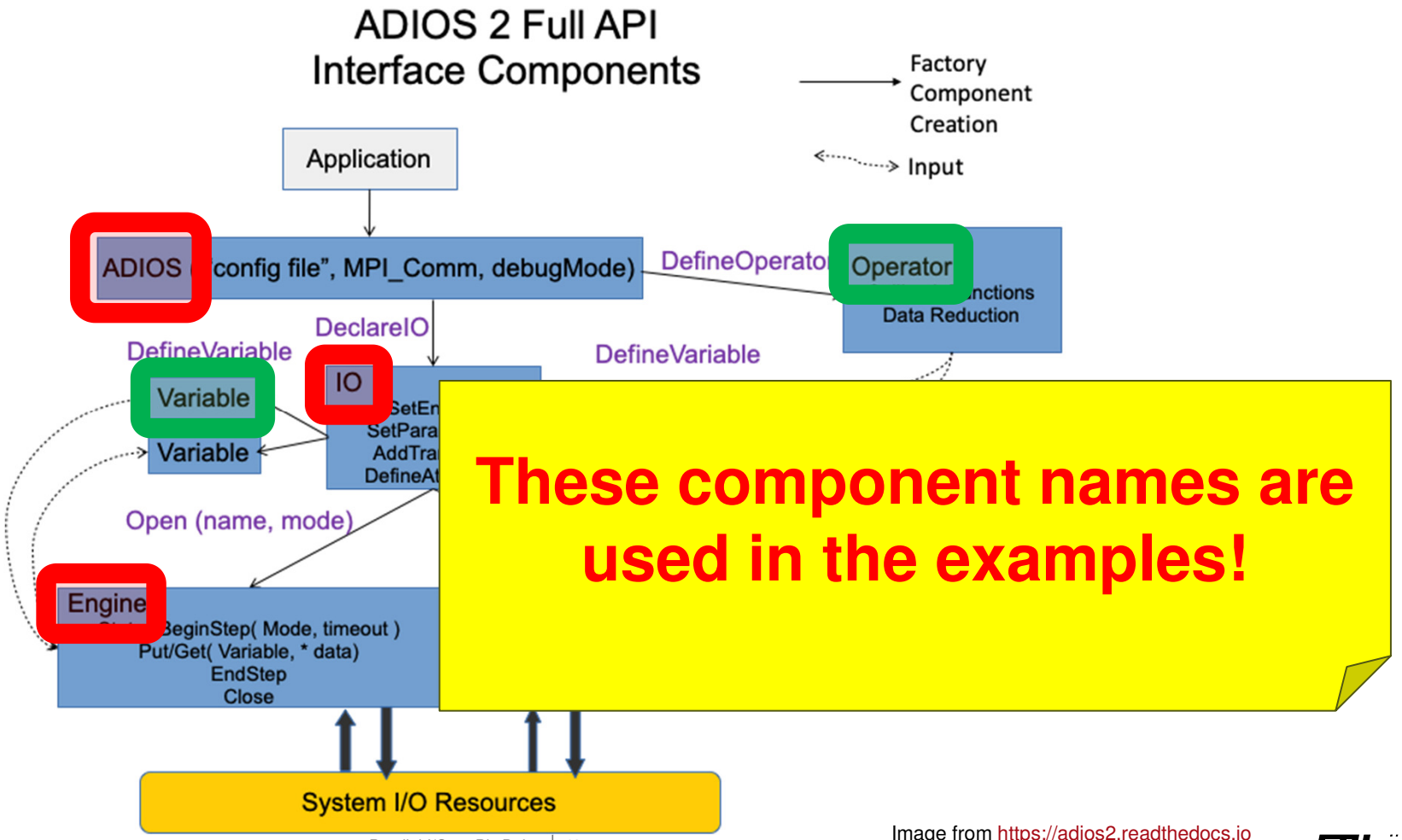
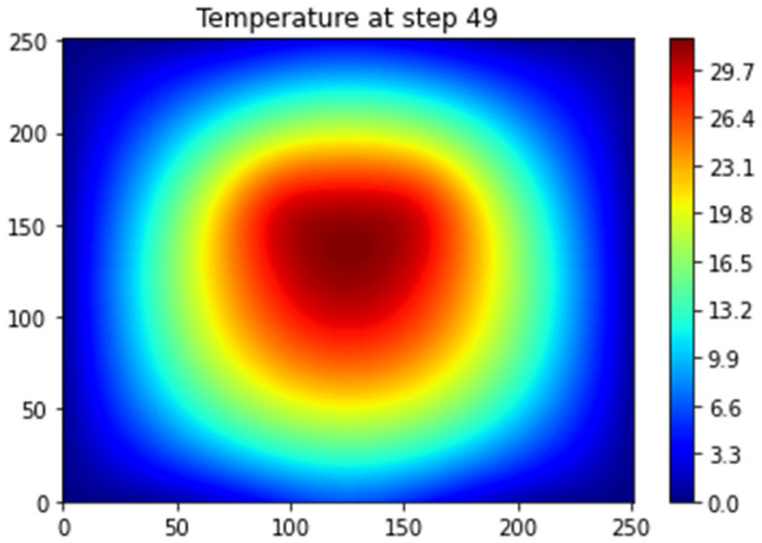


Image from https://adios2.readthedocs.io

ADIOS2 – the example

Super simple 2D heat diffusion example covering many topics

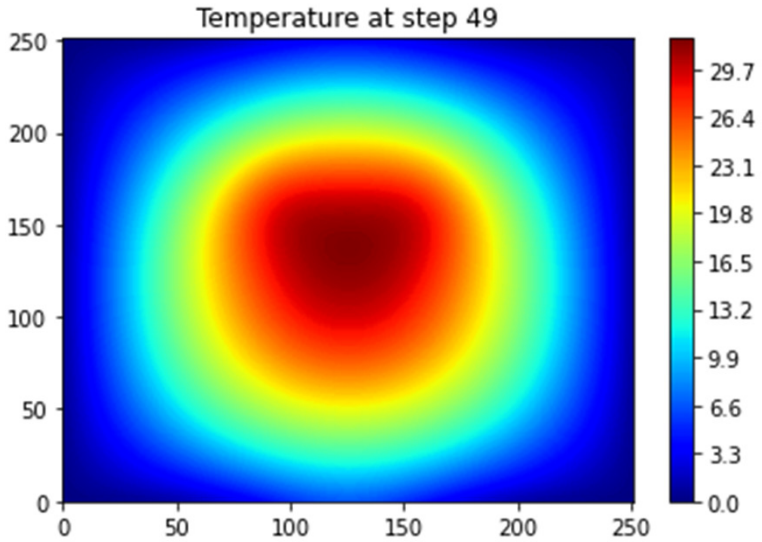
- accessing the file system in parallel
- selecting different file formats (including HDF5)
- including on-the-fly lossy or lossless compression
- staging simulation output for in-situ visualization/monitoring



ADIOS2 – the example

Super simple 2D heat diffusion example covering many topics

- accessing the file system in parallel
- selecting different file formats (including HDF5)
- including on-the-fly lossy or lossless compression
- staging simulation output for in-situ visualization/monitoring

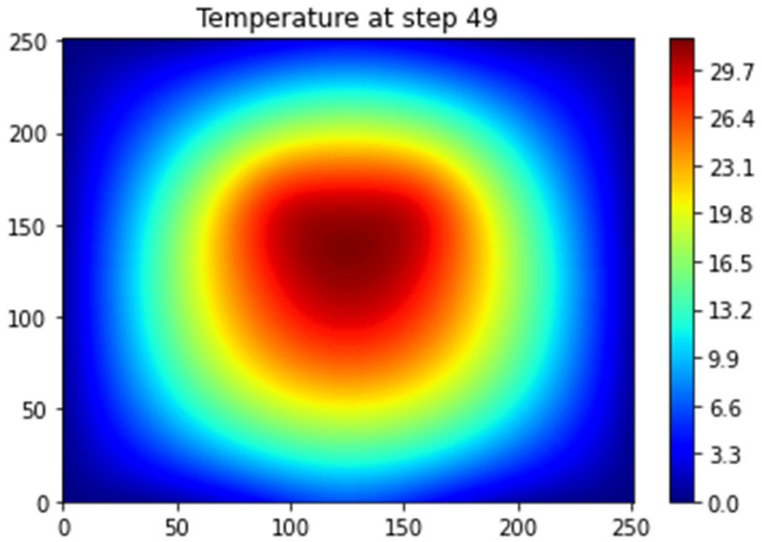


The example consists only of 3 files:
- **a Python file using mpi4py**
- **a Python notebook**
- **an adios XML configuration file**

ADIOS2 – the example

Super simple 2D heat diffusion example covering many topics

- accessing the file system in parallel
- selecting different file formats (including HDF5)
- including on-the-fly lossy or lossless compression
- staging simulation output for in-situ visualization/monitoring



The complete example with detailed comments is available here:

https://github.com/omlins/adios2-tutorial

ADIOS2 – writing to the file system in parallel

Initialization of the components adios, io and engine; declaration of variable temperature:

```
adios  = adios2.ADIOS(configFile="adios2.xml", comm=comm)
io     = adios.DeclareIO("writerIO")
engine = io.Open("diffusion2D.bp", adios2.Mode.Write)

T_id   = io.DefineVariable("temperature", T, nxy_g_nohalo, start, nxy_nohalo,
adios2.ConstantDims)
```

ADIOS2 – writing to the file system in parallel

Time loop writing nsteps (=50) times during the whole simulation:

```
for it in range(nt):
        if it % (nt/nsteps) == 0:
                engine.BeginStep()
                engine.Put(T_id, T_nohalo)
                engine.EndStep()
        # 2D heat diffusion simulation

engine.Close()
```

ADIOS2 – writing to the file system in parallel

Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <io name="writerIO">
        <engine type="BP4">
            <parameter key="OpenTimeoutSecs" value="100.0"/>
            <parameter key="SubStreams" value="128"/>
        </engine>
    </io>

    <!- (...) -->
</adios-config>
```

ADIOS2 – writing to the file system in parallel

Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <io name="writerIO">
        <engine type="BP4">
            <parameter key="OpenTimeoutSecs" value="100.0"/>
            <parameter key="SubStreams" value="128"/>
        </engine>
    </io>

    <!- (...) -->
</adios-config>
```

SubStreams is automatically set to number of processes if there are less than 128

ADIOS2 – reading from the file system using 1 process (jupyter notebook)

Initialization of the components adios, io and engine:

```
adios  = adios2.ADIOS(configFile="adios2.xml", comm=comm)
io     = adios.DeclareIO("readerIO")
engine = io.Open("diffusion2D.bp", adios2.Mode.Read)
```

No MPI communicator as we use the serial version of the ADIOS library here!

Symmetric to writing!

ADIOS2 – reading from the file system using 1 process (jupyter notebook)

While loop to read and visualize as long as there are steps incoming:

```
nprocessed=0
while engine.BeginStep(mode=adios2.StepMode.Read, timeoutSeconds=100.0) !=
adios2.StepStatus.EndOfStream :
    T_id = io.InquireVariable("temperature")
    if nprocessed == 0:
        nxy_global = T_id.Shape()
        T_type     = T_id.Type()
        T          = np.zeros(nxy_global, dtype=T_type)
    engine.Get(T_id, T)
    engine.EndStep()
    # Visualize the temperature
    nprocessed += 1
engine.Close()
```

Symmetric to writing, except for variable meta data extraction!

ADIOS2 – reading from the file system using 1 process (jupyter notebook)

Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <!- (...) -->

    <io name="readerIO">
        <engine type="BP4">
            <parameter key="OpenTimeoutSecs" value="100.0"/>
            <parameter key="SubStreams" value="128"/>
        </engine>
    </io>
</adios-config>
```

Symmetric to writing!

ADIOS2 – selecting different file formats

Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <io name="writerIO">
        <engine type="BP4">
            <!- (...) -->
        </engine>
    </io>
    <io name="readerIO">
        <engine type="BP4">
            <!- (...) -->
        </engine>
    </io>
</adios-config>
```

ADIOS2 – selecting different file formats

Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <io name="writerIO">
        <engine type="HDF5">
            <!- (...) -->
        </engine>
    </io>
    <io name="readerIO">
        <engine type="HDF5">
            <!- (...) -->
        </engine>
    </io>
</adios-config>
```

One single entry enables changing the file format!

ADIOS2 – staging simulation output for in-situ visualization/monitoring

Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <io name="writerIO">
        <engine type="SST">
            <!- (...) -->
        </engine>
    </io>
    <io name="readerIO">
        <engine type="SST">
            <!- (...) -->
        </engine>
    </io>
</adios-config>
```

One single entry enables changing to data staging for in-situ visualization/monitoring!

ADIOS2 – including on-the-fly lossy or lossless compression

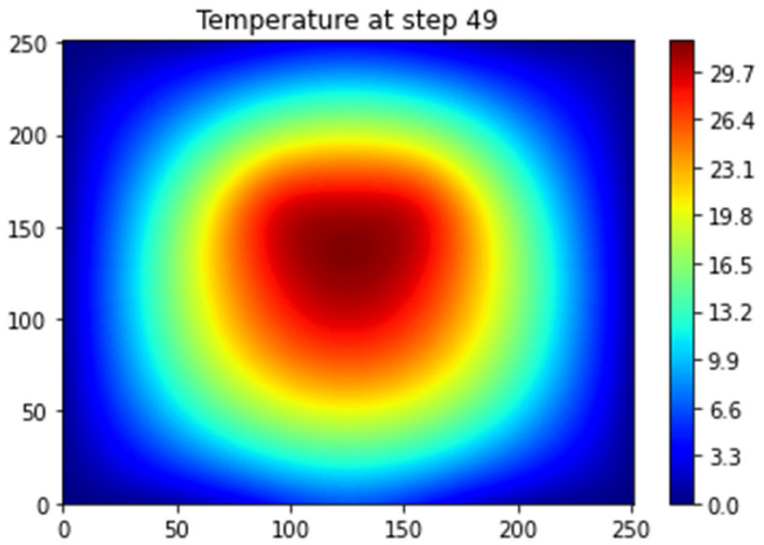
Engine parameters are in the configuration file adios2.xml:

```
<?xml version="1.0"?>
<adios-config>
    <io name="writerIO">
        <engine type="BP4">
            <variable name="temperature">
                <operation type="sz">
                    <parameter key="accuracy" value="0.0001"/>
                </operation>
            </variable>
        </engine>
    </io>
    <!- (...) -->
```

Enables setting error bound for lossy compression!

ADIOS2 – demo

Demos… !



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

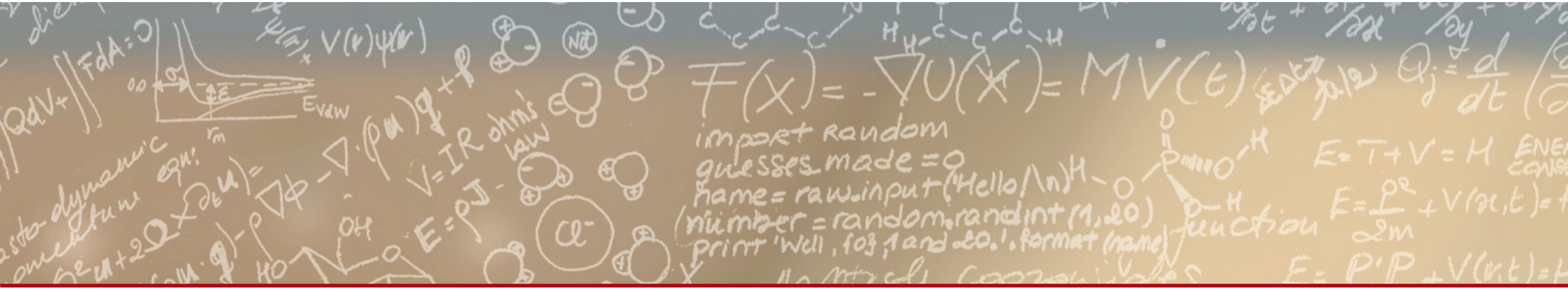
ETH zürich

Conclusions

Conclusions

- $parallel\ I/O\ lib\ \neq\ good\ performance$
  - $\Rightarrow$ Careful tuning is often required (striping + collective optimizations)
- **ADIOS2 is a great solution for parallel file I/O and data staging** (and more…)

Piz Daint in the machine room at CSCS



Thank you for your kind attention