

Best practices: files & I/O

Ine Arts, Franky Backeljauw, Michele Pugno, Robin Verschoren

or why is scratch so slow (again)

HPC@UAntwerp introduction

- `/scratch` offers highest performance, but it is sometimes *very* slow?
- in this session you will learn:
  - why some operations are fast on `/scratch` and some are not
  - how to **optimize** your **workload** when using `/scratch`
  - or when & how to use **another filesystem** instead



limited info in the HPC@UAntwerp introduction

Table of contents

1. **Mass Storage & Filesystem**
   - Memory hierarchy
   - The file concept — data and metadata, attributes
   - The filesystem — structure, interface, operations
2. **Types of filesystems**
   - Local filesystem
   - RAM disk
   - Shared network filesystem
   - Distributed parallel filesystem
   - BeeGFS @ CalcUA
   - All filesystems @ CalcUA
3. **Learning from recent observations**
   - Intermediate files in pipelines
   - OpenFOAM is I/O intensive
   - VSCode continuously scans
   - Loading software from remote mounts
4. **Mitigations & Best practices**
   - Reduce metadata operations
   - Know your software
   - Using a local filesystem
     - /tmp
     - /dev/shm
   - BeeGFS @ CalcUA : BeeOND
   - Avoid storing many small files
5. **Recent observations revisited**
   - Recognise patterns
     - File-per-process
     - Heavy temporary writes
     - High-frequency scans
6. **Concluding remarks**
   - Conclusions
   - User support



Best practices: files & I/O

Mass Storage & Filesystem

Vlaanderen
is supercomputing

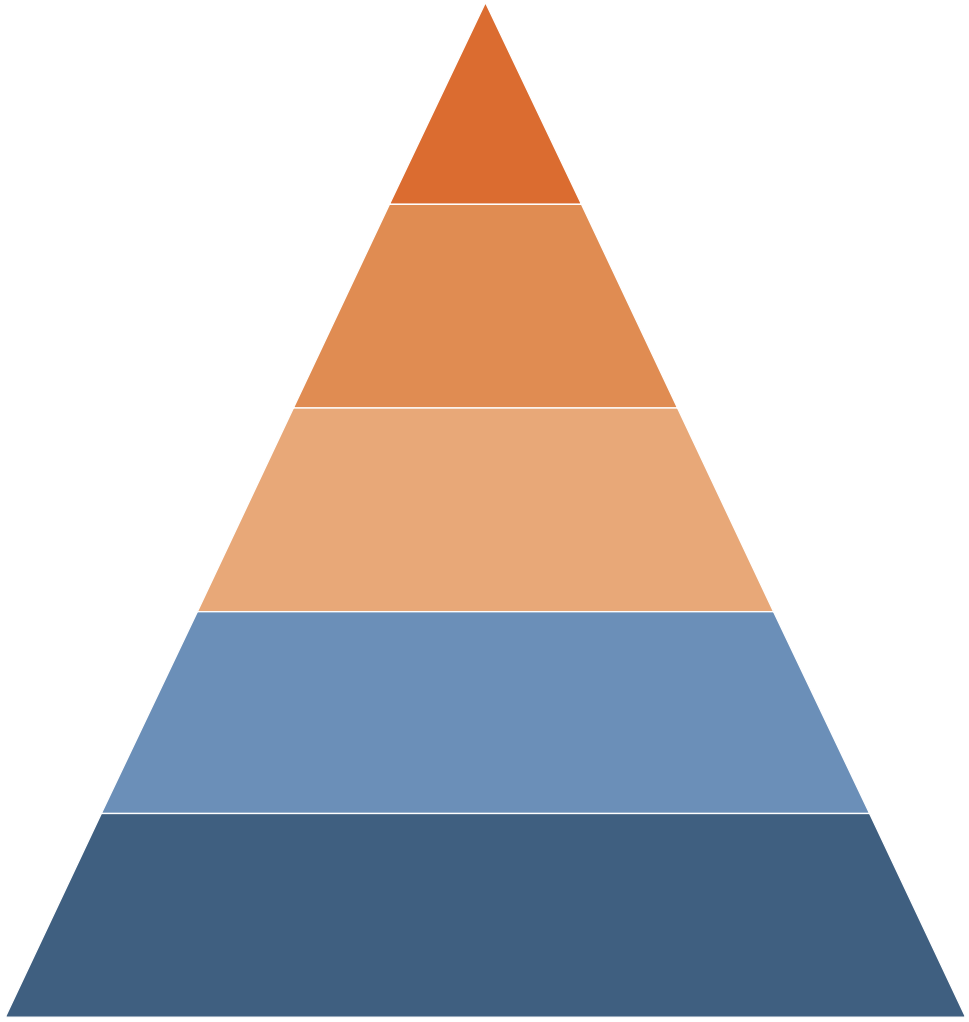
VLAAMS
SUPERCOMPUTER
CENTRUM

Innovative Computing
for A Smarter Flanders

vscentrum.be

Memory Hierarchy

fast
small



slow
large

Registers

Cache (L1/L2/L3)

RAM

HDD / SSD

Tape / archive

Volatile Memory
lost on power-off

Persistent Storage
survives reboot

VLAAMS
SUPERCOMPUTER
CENTRUM

The file concept – data and metadata

- All data on persistent storage is written in files
- A filesystem organizes files by separating two things



Metadata

Information *about* the file
(file attributes)

e.g. name, size, owner, timestamps,
permissions, …



Data

The actual file *contents*

e.g. text, image, audio,
video, arrays, …

The file concept – file attributes

- Operating system uses file attributes in the metadata to manage, organize, and control access to files

name*	human-readable label (e.g.: report.pdf)
identifier	unique number to inode within the filesystem
type	regular file, directory, symlink, …
location	pointer(s) to data blocks on the device
size	current length in bytes
modes	read / write / execute permission, ACLs
times	created, modified, last-accessed
owner	user & group ID

- The metadata of a file is stored in an **inode** (index node)

The filesystem

- A filesystem is the software layer that manages persistent storage
  - abstracts the physical storage medium (HDD, SSD, USB, network storage, etc.)
  - exposes data as files and directories instead of raw storage blocks
  - maintains an index of all files using metadata structures such as inodes
  - maps logical files to physical storage blocks
- By formatting the storage gets initialized with such a filesystem structure
  - creates inode tables, free block maps, root directory, metadata, …
- The properties of a filesystem determine its behavior and **capabilities *and limitations***
  - performance, reliability, maximum file size, permissions, journaling, scalability

The filesystem – file operations

- Programs act on files through file operations (POSIX system calls)
- The operating system mediates every read, write, and metadata change

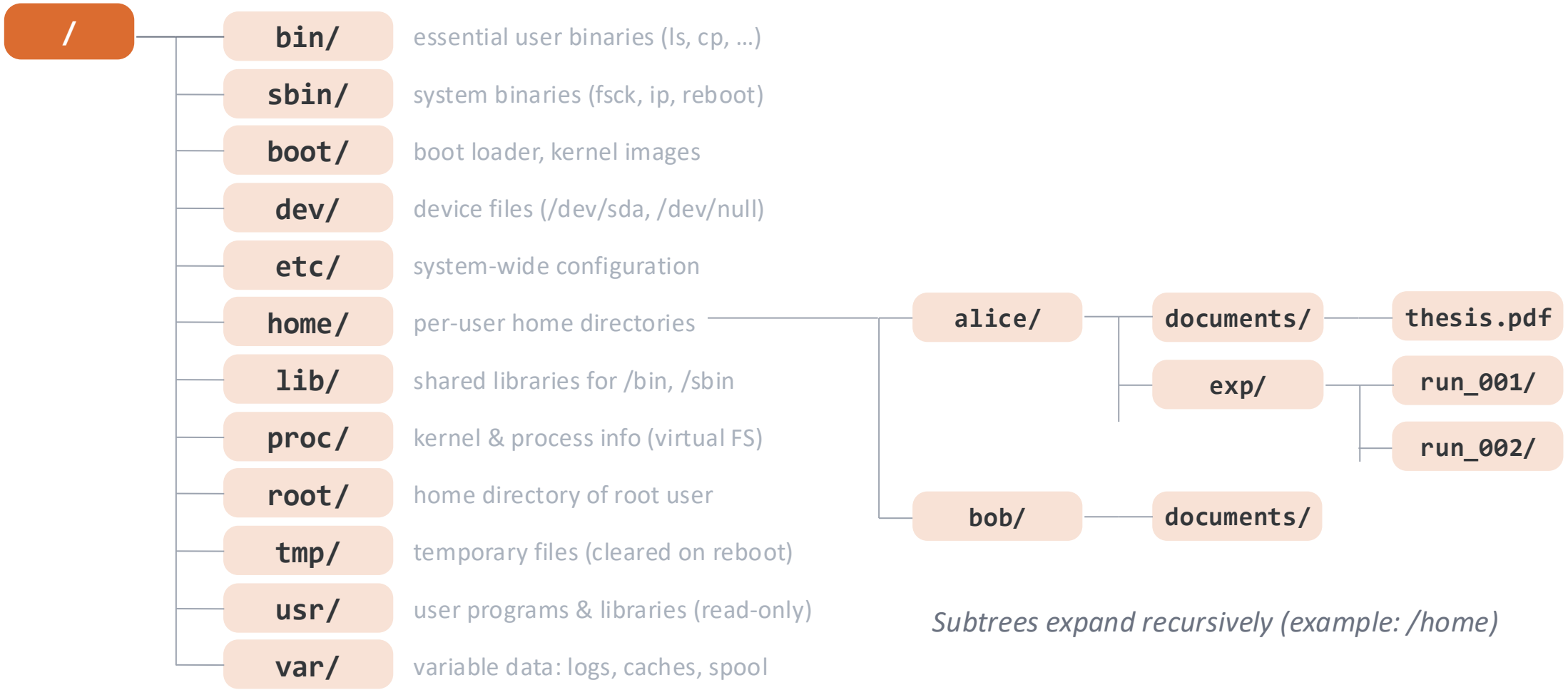
open	resolve path, return file descriptor
read	copy bytes from file into memory
write	copy bytes from memory into file
lseek	move the read/write position
close	release the file descriptor
stat	query attributes (size, times, mode)
unlink	remove a name; FS reclaims when last ref gone
truncate	set file length (usually to zero)

The filesystem – user interface

```
✓ [09:04] vsc29999@login1.vaughan $VSC_SCRATCH $ ls -al
total 44
drwxr-xr-x 6 vsc29999 vsc29999 4096 May 19 14:31 .
drwxr-xr-x 3 root     root     4096 Feb 26  2020 ..
drwxr-xr-x 2 vsc29999 vsc29999 4096 May 12 10:05 data -> /data/antwerpen/299/vsc29999
-rw-r--r-- 1 vsc29999 vsc29999 512  May 10 08:43 README.txt
drwxr-xr-x 2 vsc29999 vsc29999 4096 May 17 16:22 results
-rwxr-xr-x 1 vsc29999 vsc29999 638  May 18 11:30 run_simulation.sh
drwxr-xr-x 3 vsc29999 vsc29999 4096 May 15 09:55 scripts
-rw-r--r-- 1 vsc29999 vsc29999 8743 May 19 13:58 slurm-4829160.out

✓ [09:05] vsc29999@login1.vaughan $VSC_SCRATCH $ cat README.txt
Project: Batch example analysis
Submit jobs with: sbatch run_simulation.sh
Results are written to ./results/
Raw data in $VSC_DATA — do not modify original files
Contact CalcUA support: hpc@uantwerpen.be
```

The filesystem – tree structure

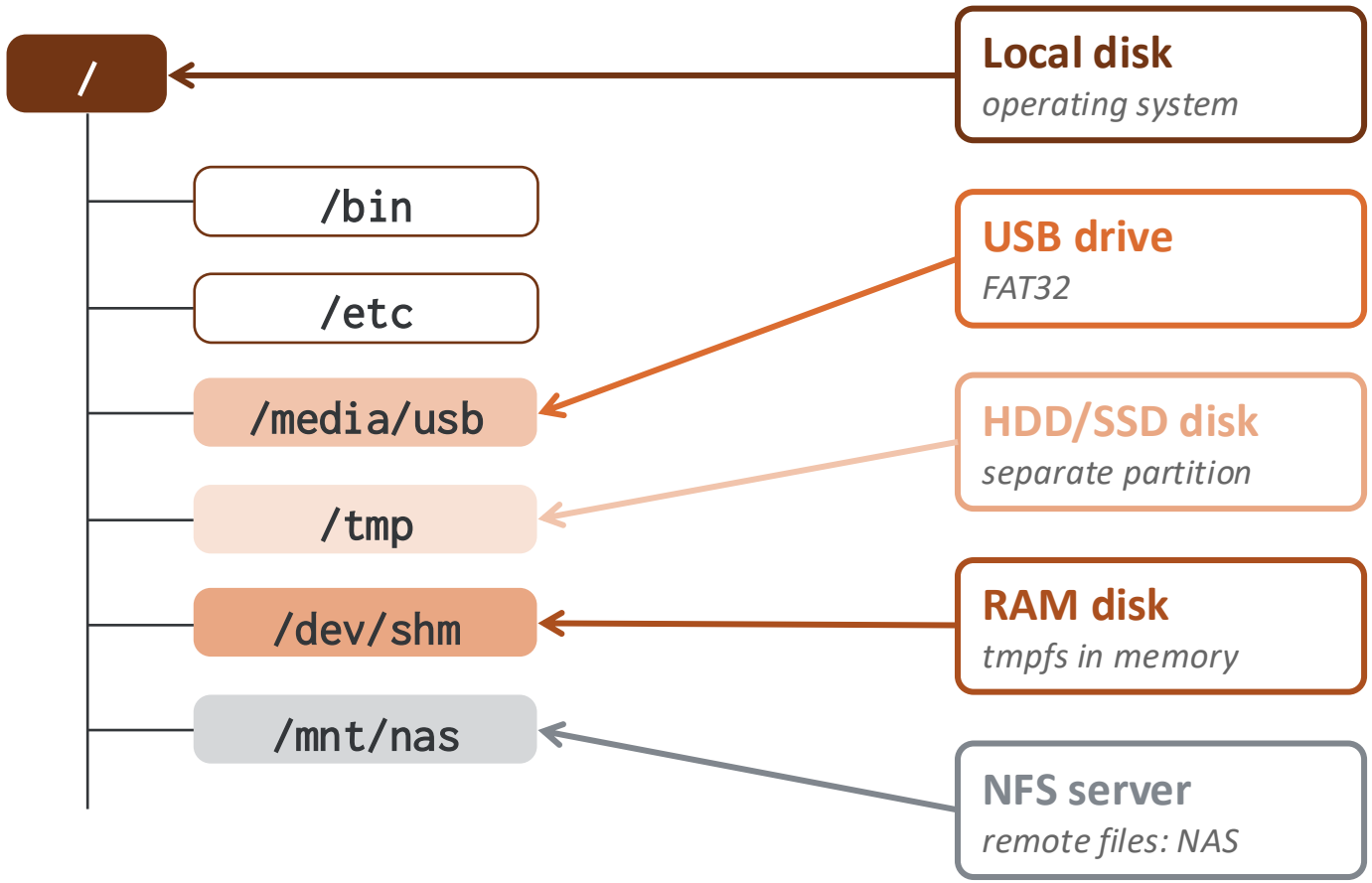


Subtrees expand recursively (example: /home)

Following the Filesystem Hierarchy Standard (FHS)

The filesystem – mounting

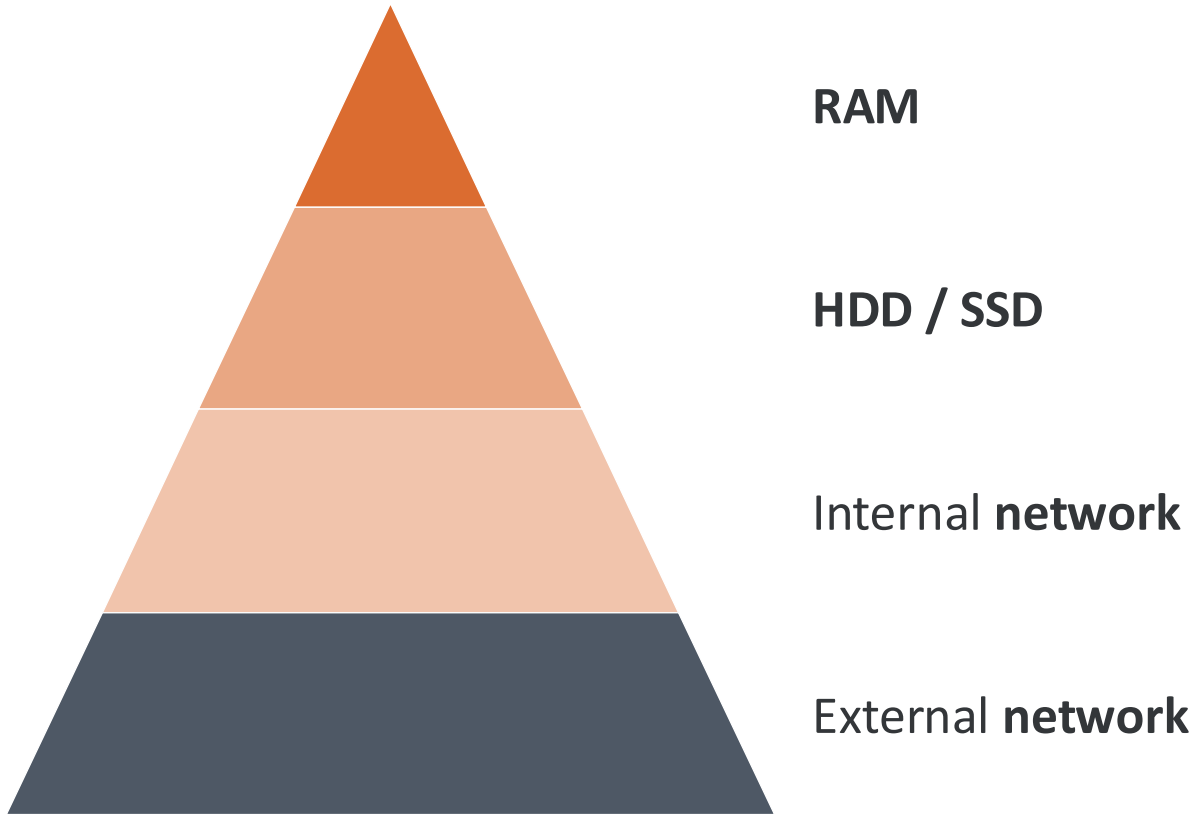
- filesystem tree is assembled from different pieces (storage devices / partitions)
- a storage device needs to be **mounted** at a specific point in the tree



The filesystem – network mounts

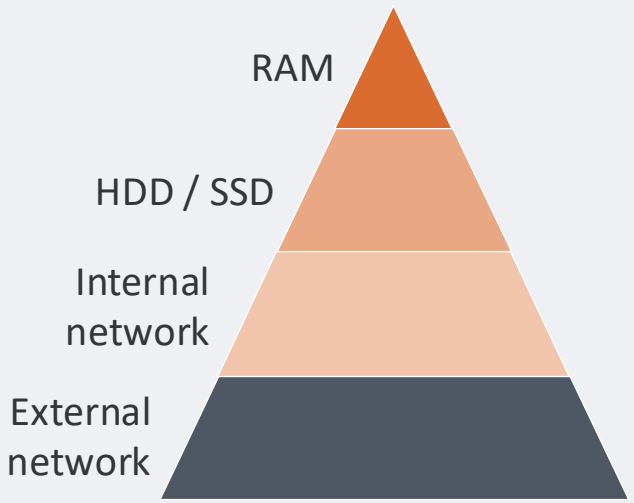
- Network mount
  - physical device is located elsewhere
  - access over the network

- Examples
  - University's N-drive
  - ~~OneDrive/Sharepoint~~ – keeps a local cache
  - at CalcUA: `/user`, `/data`, `/scratch`, … directories that are available on *all nodes*



Simplified memory graph – for the purpose of this session only

Summary – Mass Storage & Filesystem





Metadata

Data

- **Layered abstraction**
  - user sees 1 cohesive filesystem,
    not concerned with underlying details
  - operating system knows about different filesystems,
    not concerned with hardware specifics
- Storage hardware & filesystem properties:
  - performance, reliability, scalability, cost, …

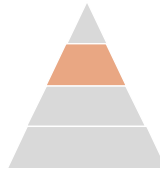
  determine their suitability
- to understand capabilities and limitations, users do need **awareness of underlying concepts**
  - **where** (hardware/filesystem) **is my data**?
  - **metadata** and **data operations**



Best practices: files & I/O

Filesystem types

Local filesystem



- Local filesystem
  - Data & metadata on same physical disk
  - OS talks directly to disk

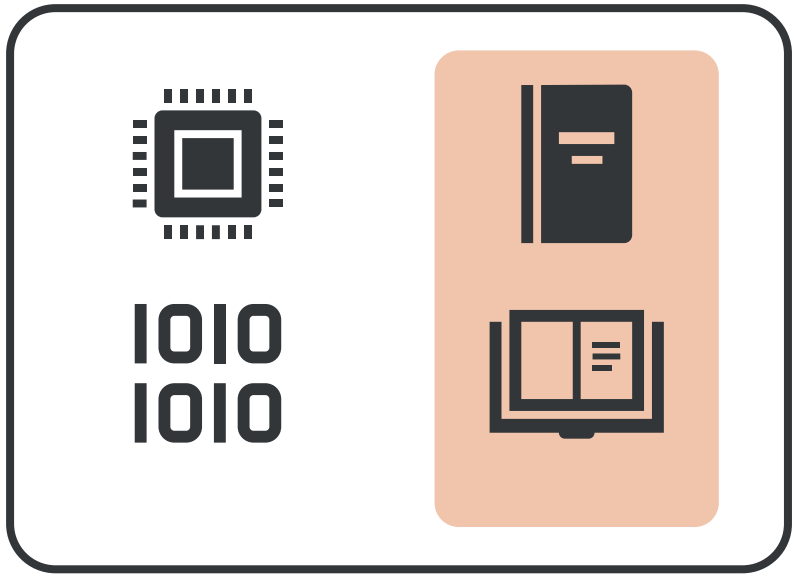
PROs

- simple

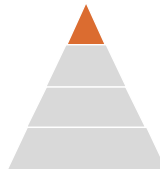
CONs

- not accessible via network
- limited capacity

  - Examples: ext4, APFS, NTFS, …



RAM disk



- RAM disk
  - Filesystem lives entirely in RAM
  - No disk I/O

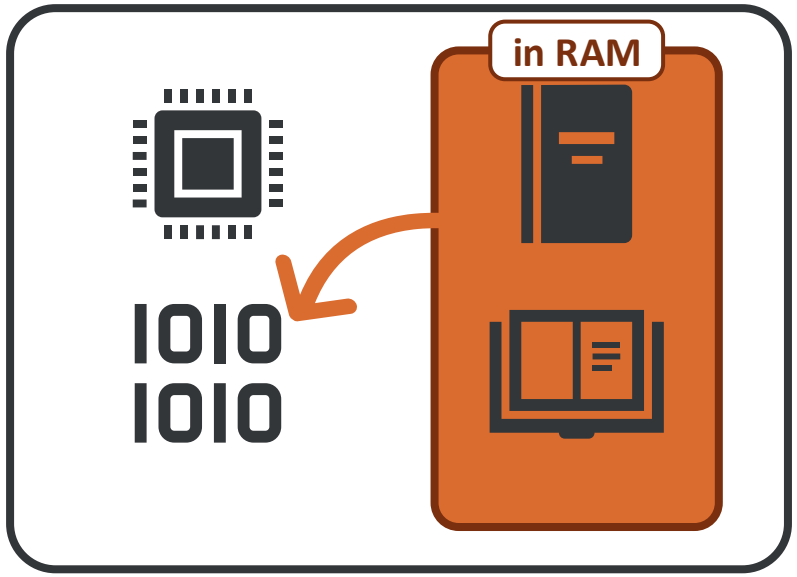
PROs

- orders of magnitude faster than SSD / HDD

CONs

- volatile — data lost on reboot or power loss
- limited capacity – fills up RAM
- local – not accessible via network

  - Examples: tmpfs, ramfs



Filesystem in RAM (no disk involved)

Shared network filesystem

- Shared network filesystem
  - One server holds all data
  - Client sends requests over network

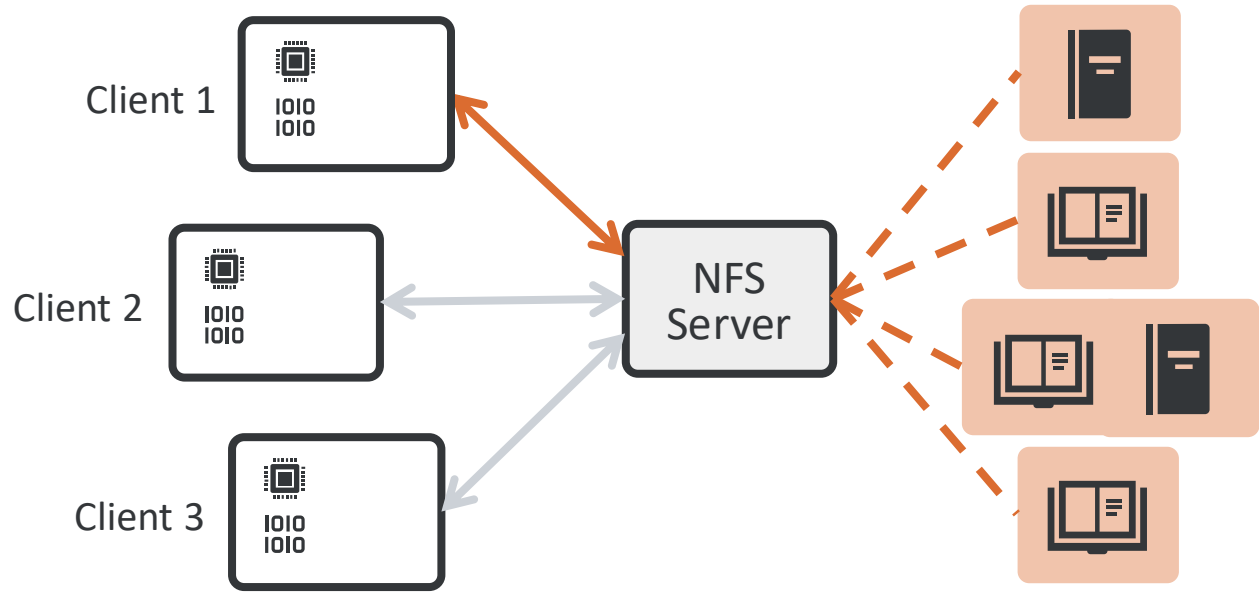
PROs

- behaves similarly to a local disk
- remotely available

CONs

- single server is bottleneck
- metadata and data through same server
- communication overhead

  - Examples: NFS, simple GPFS, SSHFS, SMB, …



Distributed parallel filesystem

- Distributed parallel filesystem
  - Dedicated metadata server
  - Parallel read/write
  - Data is striped across storage nodes

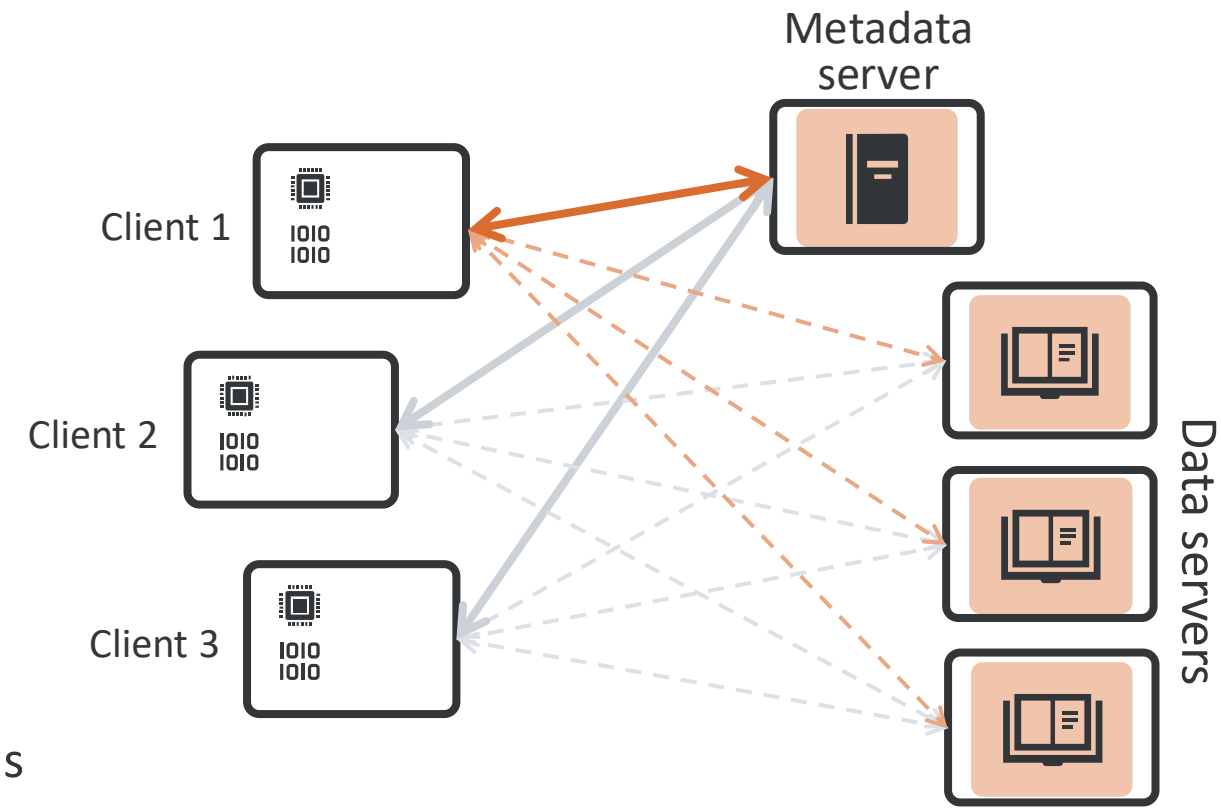
PROs

- high aggregate throughput for large files
- remotely available

CONs

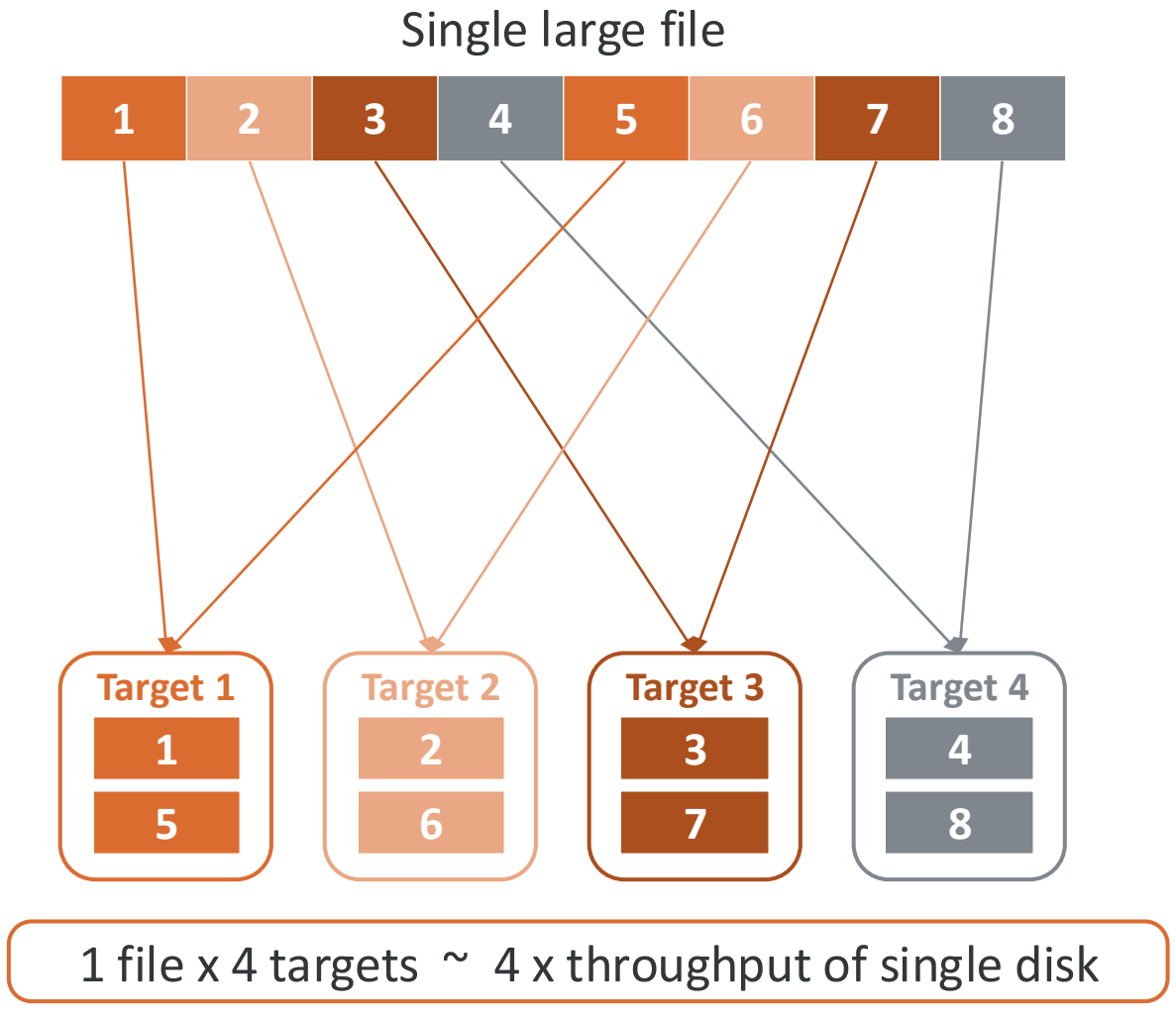
- metadata server is bottleneck: overhead for small files and random-access patterns
- communication overhead

  - Examples: BeeGFS, Lustre, GPFS/Spectrum Scale, …



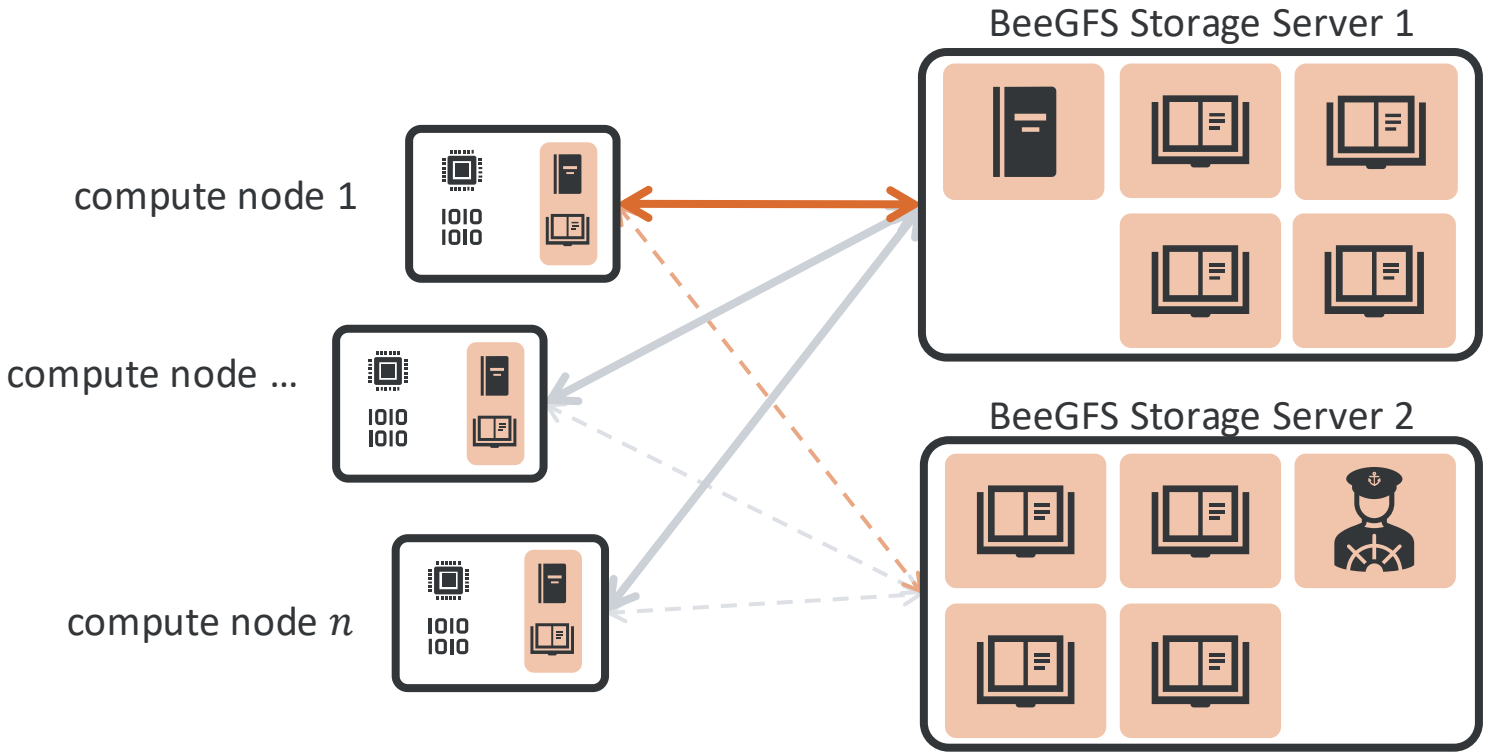
Distributed parallel filesystem — Striping

- Striping
  - A file is split into fixed-size stripes (e.g., 512 KB each)
  - Stripes distributed round-robin across *N* storage targets
  - Reads/writes hit all targets in parallel
  - Aggregate throughput scales ~linearly with the number of targets
  - **Optimized for big sequential I/O on large files**



BeeGFS @ CalcUA

- 2 dedicated storage servers with multiple *services*



All filesystems @ CalcUA — Overview

Environment variable	Type	Access	Mount locations	Backup	Default quota	Total capacity
$VSC_HOME	NFS/XFS	VSC	/user/antwerpen	YES	3 GB 20k files	3.5 TB
$VSC_DATA	NFS/XFS	VSC	/data/antwerpen	YES	25 GB 100k files	60 TB
$VSC_SCRATCH $VSC_SCRATCH_SITE	BeeGFS	Site	/scratch/antwerpen	NO	50 GB 100k files	0.6 PB
$VSC_SCRATCH_NODE	ext4 	Node	/tmp	Volatile		*node-specific*
	tmpfs 	Node	/dev/shm	Volatile		*node-specific*



Best practices: files & I/O

Learning from recent observations

Vlaanderen
is supercomputing

VLAAMS
SUPERCOMPUTER
CENTRUM

*Innovative Computing
for A Smarter Flanders*

vscentrum.be

Learning from recent observations

Why this section?

- Create awareness of pitfalls – and how easy it is to fall into them
  - how they negatively impact *yourself* in the first place
  - but sometimes *everyone* can suffer
- Show how small changes can lead to massive improvements
- NOT the goal of this section:
  - pointing fingers/scapegoating
  - calling users (or their code) bad

5 feb 2026 – Intermediate files in pipelines

- metadata ops overload:
  - unnecessary `find`'s
  - `$TEMP_DIR` on `/scratch`
- **>10k metadata ops /s**
  - reduced to 5 metadata ops /s
  - averaged over job

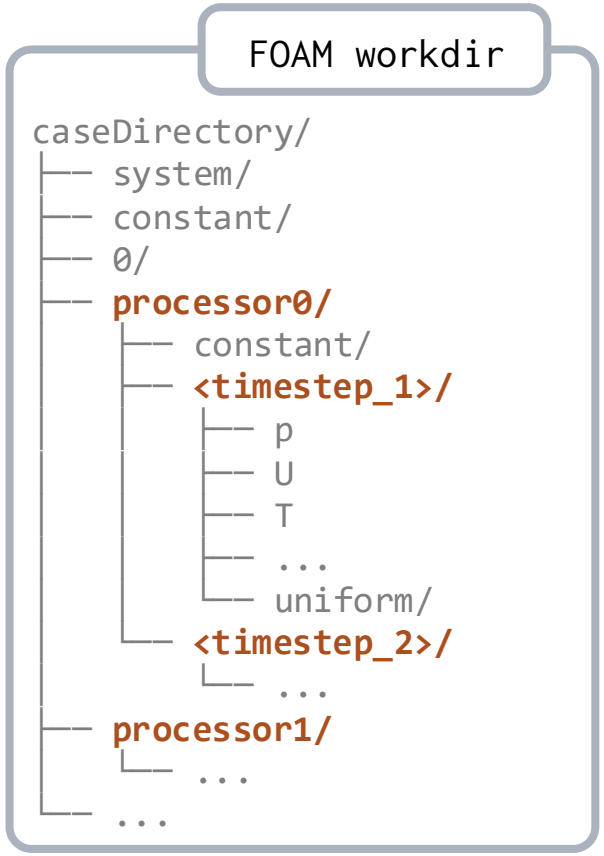
```
[…]
for i in {1..25}; do
    sbatch --array=1-2000%150 OFFSET=$($i-1) "$JOB_SCRIPT"
done
```

```
#!/bin/bash
#SBATCH --output=/scratch/antwerpen/[…]/logs/pipeline_%A_%a.out
[…]
OUTPUT_BASE_DIR="/scratch/antwerpen/[…]/output"
TEMP_BASE_DIR="/scratch/antwerpen/[…]/temp"
[…]
# Check if output already exists to avoid re-running
if [ -d "$OUT_DIR" ] && [ -n "$(find "$OUT_DIR" -maxdepth 1 -name "*.ext.gz")" ];
[…]
# Call the pipeline
if "$PIPELINE_SCRIPT" --process-single "$IN_VAR"; then
  # Verify output was created
  if [ -d "$OUT_DIR" ]; then
    echo "Output files:"
    find "$OUT_DIR" -type f -name "*.ext.gz" -o -name "*summary*" | head -5
[…]
# Cleanup job
[…]
find "${TEMP_BASE_DIR}" -maxdepth 1 -name "*${IN_VAR}*" -mmin +5 -exec rm -rf {}
```

VLAAMS SUPERCOMPUTER CENTRUM

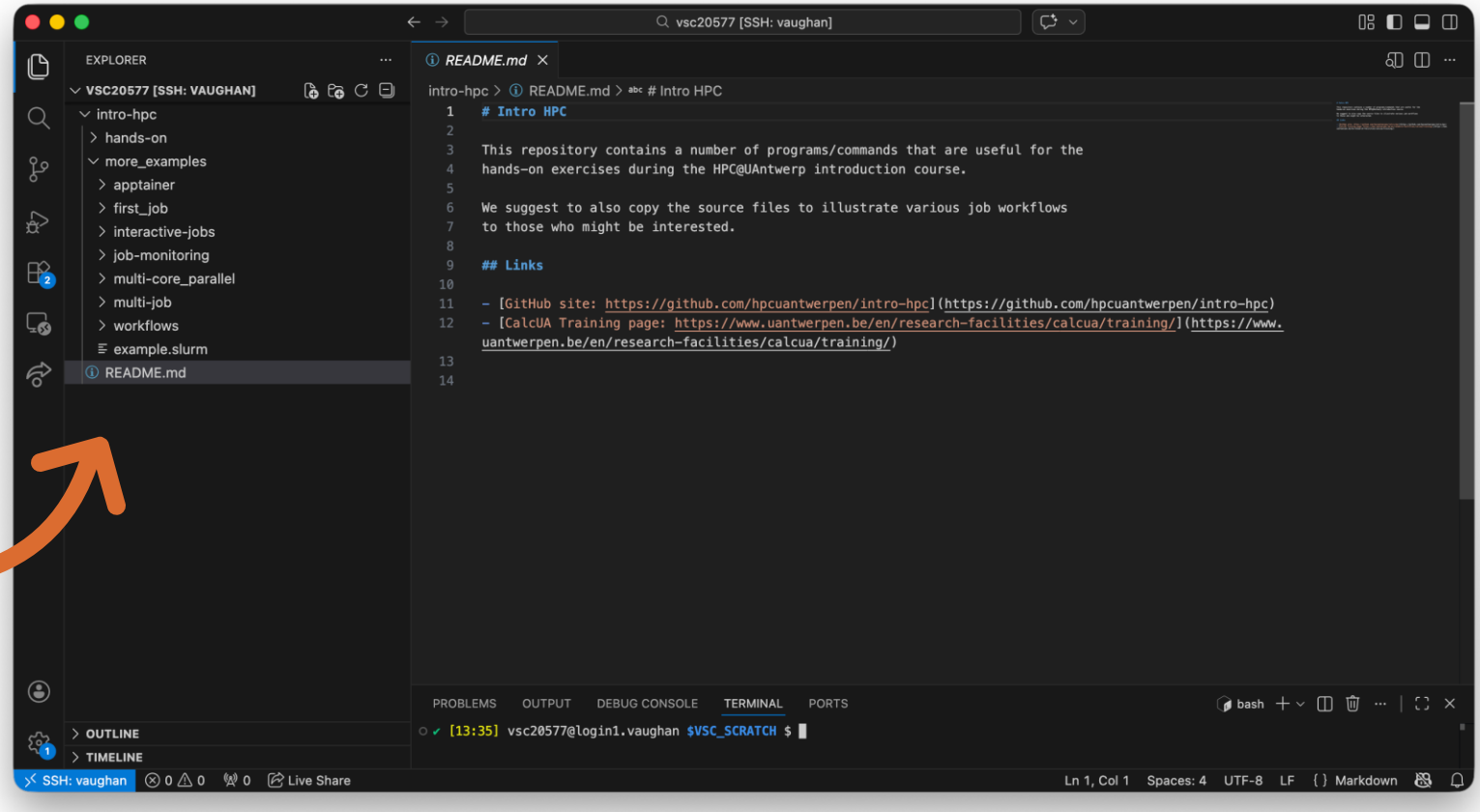
8 jan 2026 – **OpenFOAM**

- OpenFOAM is known to be I/O intensive
  - `#output_files ∝ #output_fields x #output_timesteps x #parallel_processes`
- User reported
  - Large difference in **cputime** and **walltime**
    - `ExecutionTime = 1h45`
    - `ClockTime = 9h`

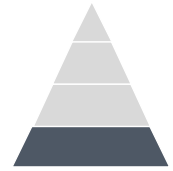


8 jan 2026 - **VSCode**

- VSCode **continuously scans** *full* directory tree
  - continuous metadata ops
  - recursive
  - in a group directory containing 7.3M files



Loading software from remote mount



- user vsc4xxxx logs into CalcUA cluster

~/.bashrc

```
[…]
conda activate $VSC_DATA/conda-env
```

  - **PATH=/data/gent/4xx/vsc4xxxx/conda-env/bin:…**
  - the shell checks this remote directory first for every command you run
    - passes through *external* network → slow

- $VSC_DATA and $VSC_HOME are located at *your* home institute
  - /data**/$VSC_INSTITUTE**/xxx/vscxxxyy/
  - home institute: VUB vsc1xxxx, UAntwerp vsc2xxxx, KULeuven vsc3xxxx, UGent vsc4xxxx
- $VSC_SCRATCH is located at institute of the *cluster*
  - /scratch**/$VSC_INSTITUTE_LOCAL**/xxx/vscxxxyy/



Best practices: files & I/O

Mitigations & Best practices

Best practices

- **Reduce metadata operations** on scratch
  - Caused by
    - Operations on *many* files
    - Opening/closing files *rapidly*

- In your jobscript:
  - Don’t use `ls -R`, `find`, `tree` … on deep directory trees
  - Be mindful of tight loops with file access

- **Your software may be doing this without you knowing**

Know your software

- **No need to be an HPC filesystem expert, but you should *know your software***
- Ask the manual
  - where does temporary data go? — `TMPDIR, scratch, workdir`
    - does is clean up its own temp files? — `cleanup, purge`
  - how many output files are created?
    - does this scale with #processes? — `collated, MPI-IO, per-rank files, n_writers`
  - can I reduce write frequency? — `writeInterval, outputFreq, checkpoints`
    - can logging verbosity be reduced? — `verbosity, logLevel, quiet, debug`
  - can I output binary instead of text? — `binary, netcdf, cff, HDF5, raw, unformatted`
  - can output be compressed? — `zlib, szip, deflate, compression level`
  - can I stop constant file reads? — `container, watcherExclude, cache`
- User support is here to help

Using a local filesystem — /tmp

- Avoid `/scratch` by **using the node's own disk**
  - mounted in **`/tmp`**
- Size of the local SSD disk in different partitions

partition	#nodes	#cores	memory	/tmp
zen2	152	64	256 GiB	200 GiB
zen3	24	64	256 GiB	400 GiB
zen3_512	16	64	512 GiB	400 GiB
ampere_gpu arcturus_gpu	1 2	64	256 GiB	400 GiB
broadwell	144	28	128 GiB	100 GiB
broadwell_256	8	28	256 GiB	100 GiB
pascal_gpu	2	28	128 GiB	100 GiB
skylake	23	28	192 GiB	400 GiB

Using a local filesystem — /tmp

- Remarks
  - `/tmp` is **temporary**: cleared after job ends
  - There are no quota on `/tmp`
  - The SSD is not part of the (Slurm-)managed resources
    - Every job on the node can write the full size of the disk
    - Request full node if you do not want to share available disk space with other jobs or users:
      `--exclusive`
  - `/tmp` is **local**: only available on the compute node itself
    - Request cores within the same node:
      `--nodes=1`

Using a local filesystem — /tmp

- EXAMPLE 1 – **USING TMP**
  - ORCA benefits from using `/tmp`
    - creates many temporary files
      can hit quota on scratch
    - heavy I/O load on these temporary files
  - ORCA testcase:
    - /scratch - 37m30s
    - /tmp - 4m12s
    - /dev/shm - 4m09s
  - Rule: do NOT run ORCA on scratch

```
#!/bin/bash
#SBATCH --ntasks=64
#SBATCH --nodes=1 --partition=zen2
[…]
module purge
module load calcua/2024a
module load ORCA/6.1.1-gompi-2024a-avx2

rsync orca-in.inp /tmp
cd /tmp || exit 1
$EBROOTORCA/bin/orca orca-in.inp  > orca-out.out
rsync orca-out.out $SLURM_SUBMIT_DIR
```

Using a local filesystem — /tmp

- EXAMPLE 2 – **CHOOSING A PARTITION**
  - Pipelines typically generate many intermediate files & input files are rapidly opened/closed
  - In this example:
    - 16 cores
    - 120 GB dataset
  - Partition: **skylake**
    - 28 cores
    - 480GB `/tmp`

```
#!/bin/bash
#SBATCH --cpus-per-task=16 --ntasks=1
#SBATCH --partition=skylake
#SBATCH --exclusive
[…]
module purge
module load calcua/2025a
module load Python/3.13.1-GCCcore-14.2.0
module load SciPy-bundle/2025.06-gfbf-2025a
source ${VSC_DATA}/virtualenvs/bin/activate

cp -r dataset /tmp
pipeline.sh --threads=$SLURM_CPUS_PER_TASK \
--workdir=/tmp/dataset
cp -r /tmp/dataset/output $SLURM_SUBMIT_DIR
```

Using a local filesystem — /dev/shm

- Avoid `/scratch` by **using the node's volatile memory**
  - mounted in **`/dev/shm`**
- Size of the local RAM disk in different partitions:

partition	#nodes	#cores	memory	/dev/shm (max)
zen2	152	64	256 GiB	240 GiB
zen3	24	64	256 GiB	240 GiB
zen3_512	16	64	512 GiB	496 GiB
ampere_gpu arcturus_gpu	1 2	64	256 GiB	240 GiB
broadwell	144	28	128 GiB	112 GiB
broadwell_256	8	28	256 GiB	240 GiB
pascal_gpu	2	28	128 GiB	112 GiB
skylake	23	28	192 GiB	176 GiB

Using a local filesystem — /dev/shm

- Remarks
  - `/dev/shm` is **temporary**: cleared after job ends
  - Available size matches the allocated memory for the job
  - `/dev/shm` is **local**: only available on the compute node itself
    - Request cores within the same node:
      `--nodes=1`
  - **Memory used for files is no longer available for calculations**

Using a local filesystem - /dev/shm

- EXAMPLE – **USING DEV/SHM**
  - BLAST+ compares sequences to a database
    - Database = 180GB
  - Memory needs:
    - < 2 GB per task
  - Timings:
    - /scratch : 30min/seq
    - /dev/shm : 30s/seq
    - startup cost : 20min

```
#!/bin/bash
#SBATCH --ntasks=32 --cpus-per-task=2
#SBATCH --nodes=1 --partition=zen3_512
[…]
module load BLAST+/2.16.0-gompi-2024a
[…]
DBDIR=/scratch/antwerpen/databases
BLASTDBDIR=${DBDIR}/bio/NCBI/downloads/2024-12-12/core_nt
cp -r $BLASTDBDIR /dev/shm
export BLASTDB=/dev/shm/core_nt
for sample in sample-*.fasta ; do
 srun -n1 -c ${SLURM_CPUS_PER_TASK} --export=BLASTDB blastn \
  -query $sample -db core_nt -num_threads ${SLURM_CPUS_PER_TASK} \
  -out ${sample}.out.${SLURM_JOB_ID} &
done
wait
```

[new!] **BeeOND** – BeeGFS on demand

- *Objective: allow using the local RAM disk from multiple nodes in a job*
- BeeOND **creates a shared parallel filesystem** on a *per-job basis*
  - out of the local drives of your job's compute nodes
  - for the runtime of the job
- Best of both worlds, but
  - needs setup (easy)
  - adds overhead
- Useful for multi-node jobs
  - with heavy I/O
  - where every node needs access to the temporary files
- Mounted in **/scratch/beeond**

```
[…]
#SBATCH --constraint=beeond
#SBATCH --exclusive
[…]

cp -r input /scratch/beeond/input
cd /scratch/beeond

srun calculation […]

cp -r output $SLURM_SUBMIT_DIR/output
```

Avoid storing many small files – using container formats

- For (intermediate) output in `/tmp`: pack before copying back to shared storage
  - Tar archive
    - create: `$ tar -czf results.tar.gz ./output_dir/`
    - extract: `$ tar -xzf results.tar.gz`

- For array/numerical data: use **container formats** to store many arrays in 1 file
  - **HDF5**, **NetCDF** - single file, parallel write support via MPI-IO
  - **Zarr** - powerful for chunked access,
    - important: use **ZipStore** storage backend on BeeGFS
    - default DirectoryStore creates one file per chunk
  - Matlab `.mat` files
  - …

Avoid storing many small files – using containers

- Conda environments create many files (>100k)
  - Do not belong on `/scratch` (*)
  - Can easily hit inode quota on `/data`
- Containerize these environments:

```
$ module load hpc-container-wrapper
$ conda-containerize new --prefix \
  "$VSC_SCRATCH/bsoup" environment.yaml
```

  - Similar for **pip-containerize**
- Use your containerized executables:

```
$ $VSC_SCRATCH/bsoup/bin/executable"
```

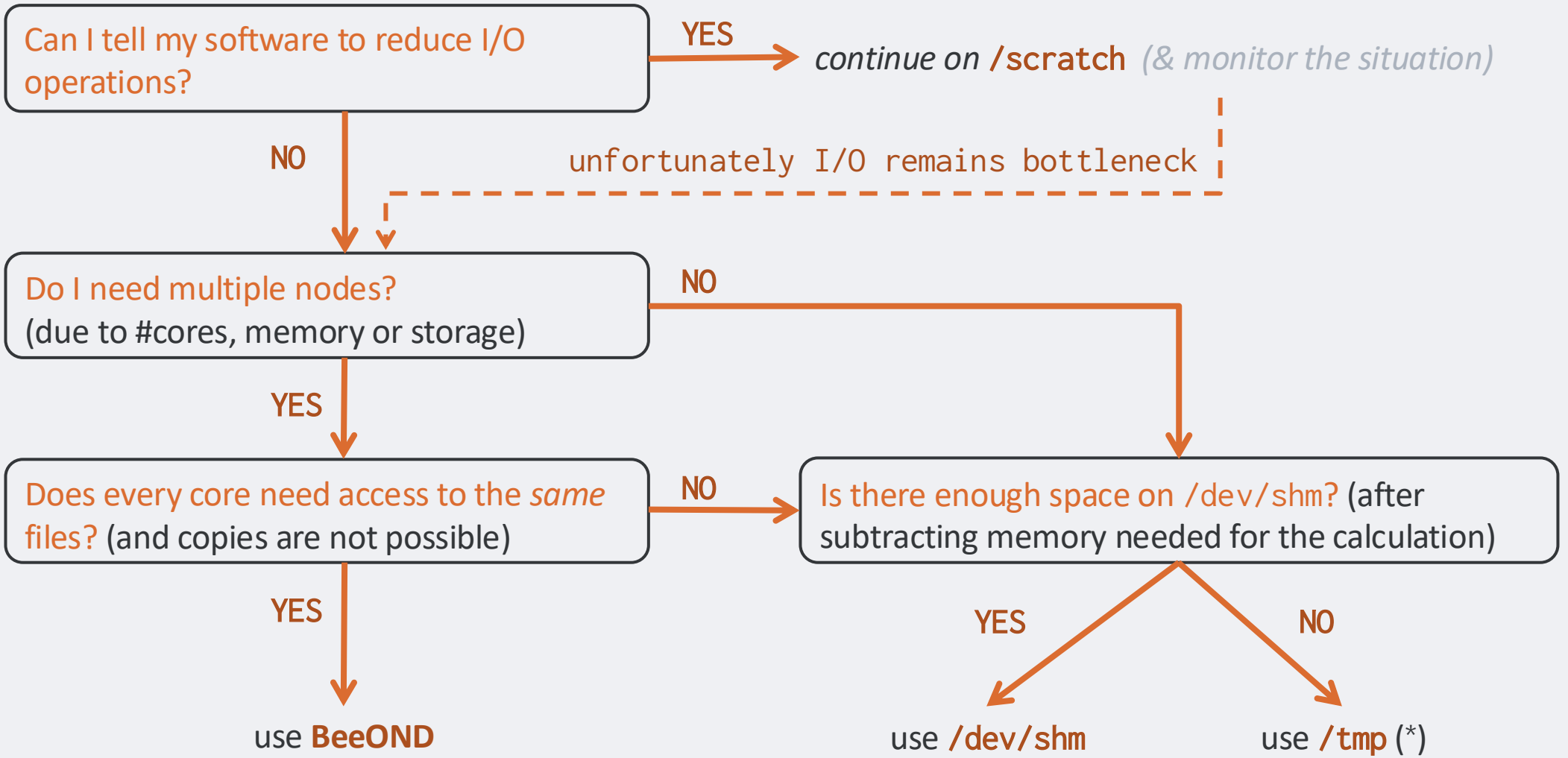
environment.yaml

```
name: bsoup4
channels:
  - conda-forge
dependencies:
  - beautifulsoup4
  - ...
```

Python package management on VSC or hpc-container-wrapper for conda

see getting scientific software installed session next week

Flowchart – when your job is I/O intensive





Best practices: files & I/O

Resent observations revisited

Pattern – file-per-process

example OpenFOAM

problem default behavior creates files per *process* per *field* per *timestep*

- #parallel_processors
  - #output_timesteps
    - #output_fields
- easily 100 x 100 x 10 files per calculation

fix

- use collated I/O
  - buffer data in memory & collect into master files per node
  - `srun simpleFoam -parallel -fileHandler hostcollated`
- use local filesystem with BeeOND

rule check your manual for output control before scaling up #cores

Pattern – heavy temporary writes

example ORCA & TELEMAC

problem dumps temporary integrals, matrices, sub-domains, etc. in files (on `/scratch`) even though they don't need to be visible to other nodes

fix use a local filesystem

- ORCA
  - change working directory to local filesystem
- TELEMAC
  - perform domain splitting (pre-) & merging (post-process) on a local filesystem
  - only copy the final merged result back

rule if a file is deleted or merged in the end, it should not live on network storage

Pattern – high-frequency small reads (scans)

example VS Code & remote IDE's

problem modern IDE's continuously scan directory tree
building git status, auto-complete indices, …
→ millions of `stat()` operations

fix

- disable file watching in `.vscode/settings.json`
  - ```
    "files.watcherExclude": { "**/*": true, },
    "git.enabled": false,
    "git.autorefresh": false,
    ```
- avoid opening deep directory trees (on `/scratch`) as project directory

rule network filesystems aren't local SSDs: exclude them from background file watchers
    ```



Best practices: files & I/O

Concluding remarks

Conclusions – Best practices: files & I/O



Metadata



Data

- Parallel filesystems like `/scratch`
  - High throughput for (large) files distributed over multiple data servers
  - Bottleneck = metadata server
  - **Reduce metadata operations** on `/scratch`
- **Know what your software does**
- Move to local filesystem for intensive I/O data
  - `/tmp` or `/dev/shm`
  - BeeOND
- **Reduce number of files** on `/scratch`
  - Archives & containers

User support

- **Questions?** → contact us via **hpc@uantwerpen.be**
  - offices @ **CMI — G.309-G.313**

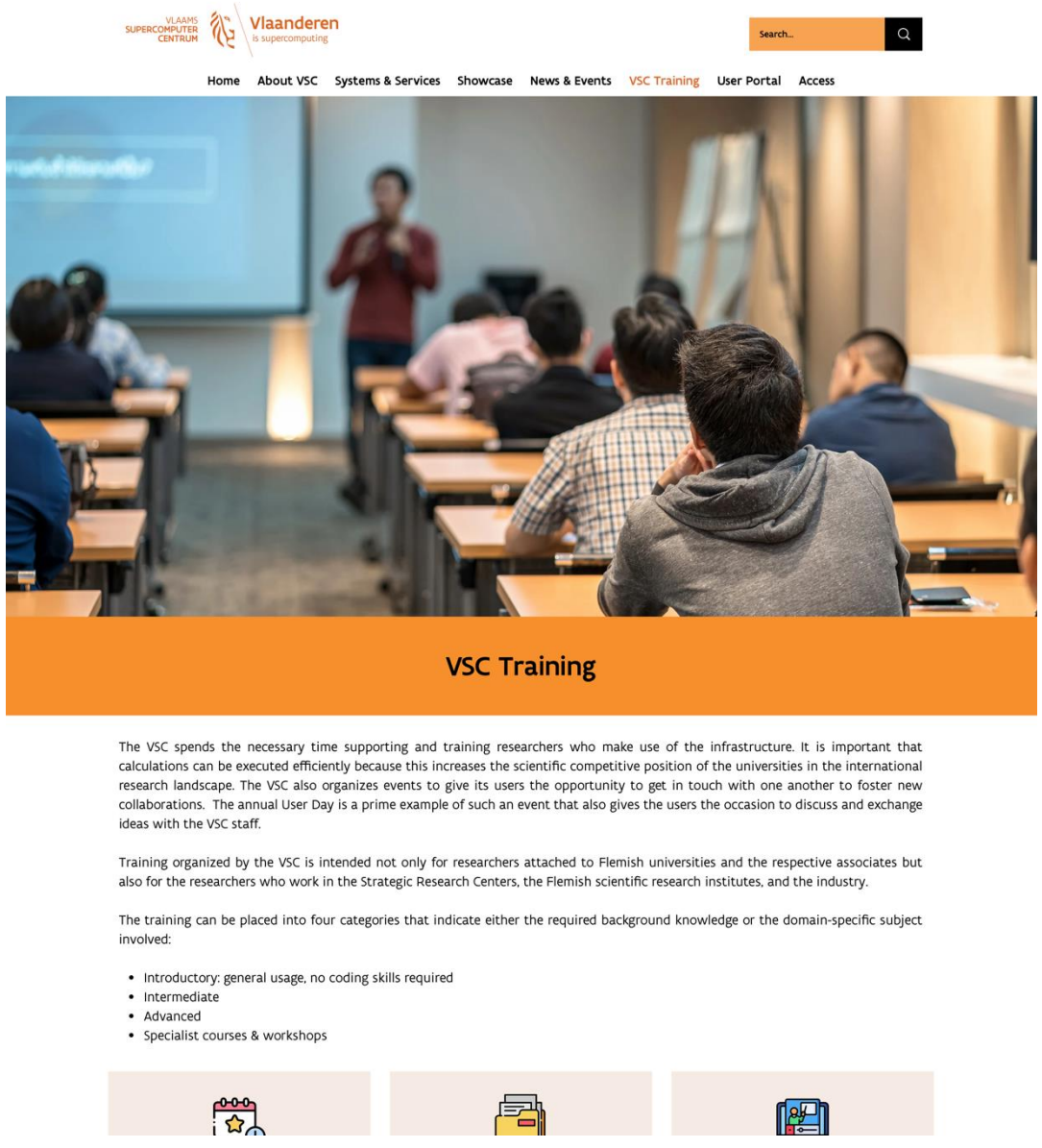
- mailing-list for announcements: calcua-announce@lists.uantwerpen.be

- Some guidelines for help
  - be as precise as possible – e.g.: give job id, submit dir, output files, …
  - help us help you – read (and understand) the relevant documentation

CalcUA website – VSC docs – Slurm docs

More training

- HPC core facility CalcUA
  - Introduction to Linux
  - HPC@UAntwerp introduction
  - Supercomputers for starters
- VSC Trainings
  - trainings organized by other VSC sites and abroad (including LUMI, and EuroCC)
- **Training sessions by Geert Jan Bex**





Best practices: files & I/O

Back-up slides for specific software

OpenFOAM

- **Collated file** options
  - using options:

```
srun simpleFoam -parallel -fileHandler collated
```

  - or using environment variables:

```
export FOAM_FILEHANDLER=collated
export FOAM_IORANKS=\($(seq 0 $SLURM_NTASKS_PER_NODE $SLURM_NTASKS)\)
                        #e.g. '(0 64 128)'
```

```
processors128
├── 2000
│   ├── k
│   ├── kMean
│   […]
```

```
processors128_0-63
processors128_64-127
├── 2000
│   ├── k
│   ├── kMean
│   […]
```

  - to convert between output formats:

```
srun foamFormatConvert -parallel -fileHandler uncollated
```

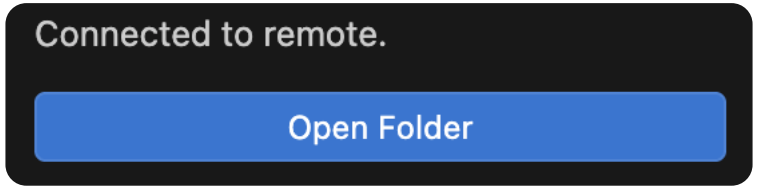
OpenFOAM

- (other) Controldict parameters
  - `maxThreadFileBufferSize 2e9;` // thread buffer size for file writes. if 0 or insufficient, threading is not used and the OpenFOAM execution waits for I/O to finish.
  - `purgeWrite N;` // only store N last timesteps (timestep -(N+1) is overwritten)
  - `runTimeModifiable false;` // do not re-read controlDict *every* timestep

OpenFOAM parallel-io guide & OpenFOAM best practices (Pawsey computing centre)

VSCode

- Avoid opening deep directory trees (on `/scratch`) as project directory



- Suggested settings:
  - symlink: `~/.vscode-server -> $VSC_DATA/.vscode-server`

$VSC_DATA/.vscode-server/data/Machine/settings.json

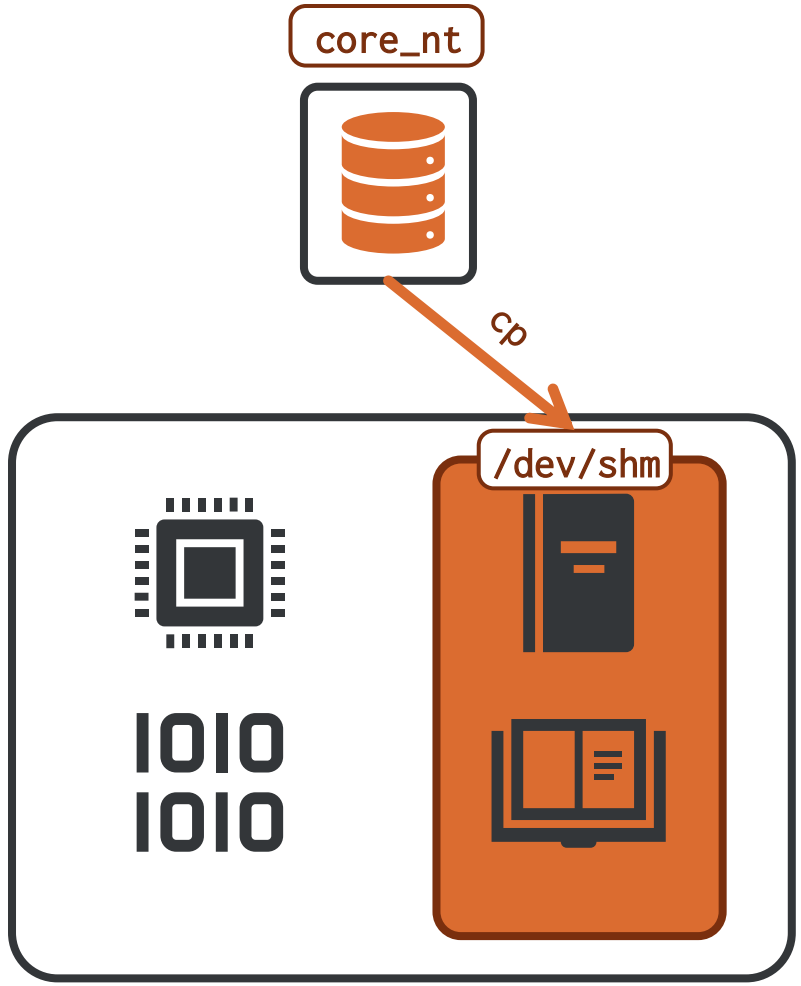
```
{
    "files.watcherExclude": { "**/*": true, },
    "search.followSymlinks": false,

    "git.enabled": false,
    "git.path": null,
    "git.autorefresh": false,
    "git.fetchOnPull": false,

    "explorer.compactFolders": false,
    "explorer.autoReveal": false,
}
```

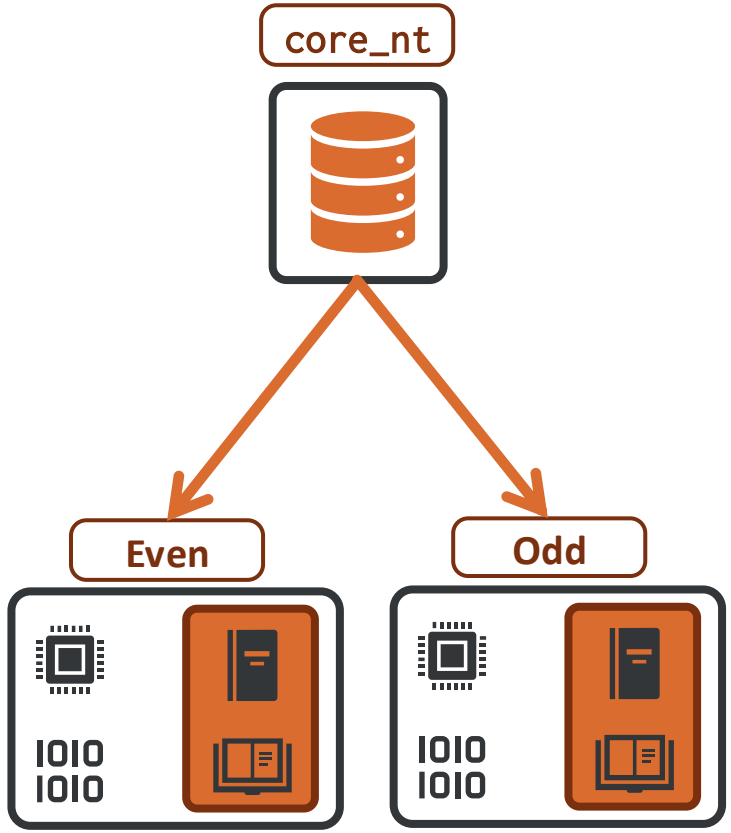
BLAST+

- The `core_nt` database is available for everyone at `/scratch/antwerpen/databases/bio/NCBI/latest/blast/core_nt`
- DB access is crucial
  - Reading the database from BeeGFS
    ⇒ metaoperation flooding
  - Copy database to `/dev/shm`
    ⇒ responsive FS and fast query time
- If the `core_nt` database in use fits in memory, a single node run is possible
  - Copy the database in `/dev/shm`
  - `export BLASTDB="/dev/shm"`
  - Execute BLAST+



BLAST+

- The latest version of `core_nt` is 267GB which is larger than the `/dev/shm` size on most of the nodes
  - A **distributed approach** is then necessary
  - The `core_nt` database already comes divided in 100 chunks
  - Copy the even numbered chunks on one node and the odd numbered ones on the second one. (or a different rule of choice for more nodes)
  - Some files must be copied on every node: `core_nt.* taxdb.* taxonomy4blast.*`
- Execute BLAST+ on every node in parallel
- Merge the results



BLAST+

- The job script
  1. Sets up the environment
  2. Lists all the available DB chunks
  3. Computes the total **DB size**
     - necessary for `-dbsize` option
  4. **Executes** multiple job steps by calling the ***worker script***.
     - this reuses the staged database multiple times for different fasta files
  5. **Merges** the results of the computation
     - ranking by bitscore
     - keeping the top 50 *subjects*

blastn_core_nt_2node.slurm

```
1  #!/bin/bash
   #SBATCH --job-name=blastn_core_nt_2n
   #SBATCH --partition=zen3_512
   #SBATCH --nodes=2 --ntasks=2 --ntasks-per-node=1
   #SBATCH --cpus-per-task=64
   #SBATCH --output=blastn_core_nt-%j.out
   #SBATCH --error=blastn_core_nt-%j.err

   set -euo pipefail
   module purge
   module load calcua/all
   module load BLAST+/2.16.0-gompi-2024a

   export DBDIR=/scratch/antwerpen/databases/bio/NCBI/latest/blast/core_nt
   export OUTDIR="$SLURM_SUBMIT_DIR/results-$SLURM_JOB_ID"
   export FMT="6 qseqid sseqid pident length evalue bitscore stitle"

   mkdir -p "$OUTDIR"

2  ls "$DBDIR"/core_nt.*.nin | xargs -n1 basename | sed 's/\.nin$//' | sort >
   "$OUTDIR/volumes.txt"

3  INFO=$(BLASTDB="$DBDIR" blastdbcmd -db core_nt -info)
   DBSIZE=$(awk '/total bases/ { gsub(/,/,""); print $3 }' <<< "$INFO")
   [[ -n "$DBSIZE" ]] || { echo "could not read core_nt size" >&2; exit 1; }
   export DBSIZE

4  # One job step per query, the first one stages the DB
   srun bash "$SLURM_SUBMIT_DIR/blastn_worker.sh" "$SLURM_SUBMIT_DIR/query_16s.fasta" 16s
   srun bash "$SLURM_SUBMIT_DIR/blastn_worker.sh" "$SLURM_SUBMIT_DIR/query_23s.fasta" 23s

5  # merge results
   for tag in 16s 23s; do
           sort -t$'\t' -k1,1 -k6,6gr "$OUTDIR/$tag"-*.tsv | awk -F'\t' '{k=$1 FS $2} !(k in
           seen){seen[k]=(++n[$1]<=50)} seen[k]' > "$OUTDIR/hits-$tag.tsv"

           echo "-> $OUTDIR/hits-$tag.tsv"
   done
```

BLAST+

- The worker script

1. Processes the command line arguments
2. **Copies** the proper DB files to the `/dev/shm` of the node if not already staged
3. **Executes BLAST+** against the given query with the correct settings and `-dbsize`

blastn_worker.sh

```
#!/bin/bash
set -euo pipefail

1 QUERY="$1"
  TAG="$2"

2 if [[ ! -f /dev/shm/core_nt.nal ]]; then
        mapfile -t VOLS < <(awk -v r="$SLURM_PROCID" -v n="$SLURM_NNODES" 'NR%n==r'
        "$OUTDIR/volumes.txt")

        FILES=( "$DBDIR"/core_nt.{ndb,njs,nos,not,ntf,nto}
                "$DBDIR"/taxdb.b* "$DBDIR"/taxonomy4blast.* )
        for v in "${VOLS[@]}"; do FILES+=( "$DBDIR/$v".* ); done
        printf '%s\n' "${FILES[@]}" | xargs -P8 -I{} cp {} /dev/shm/

        printf 'TITLE core_nt (node %s)\nDBLIST %s\n' "$SLURM_PROCID" "${VOLS[*]}" >
        /dev/shm/core_nt.nal
  fi

3 BLASTDB=/dev/shm blastn -task megablast -query "$QUERY" -db core_nt \
        -out "$OUTDIR/$TAG-$SLURM_PROCID.tsv" -outfmt "$FMT" \
        -evalue 1e-10 -max_target_seqs 50 -dbsize "$DBSIZE" \
        -num_threads "$SLURM_CPUS_PER_TASK"
```

BLAST+

- The proposed example executes two job steps associated with their query
- The first job step stages the DB in the nodes RAM then computes
- The second immediately starts computing as the DB is already staged
- More queries can be added by appending a new `srun` calls in the job script — this saves around 7 minutes of staging time per fasta file

event	elapsed time
`volumes.txt` creation (job start)	negligible
Query 1 results, both nodes	~7.5 min — staging + search
Query 2 results, both nodes	~21 s
Output merge	negligible