EESSI

EUROPEAN ENVIRONMENT FOR SCIENTIFIC SOFTWARE INSTALLATIONS

EESSI hackathon

Using EESSI +

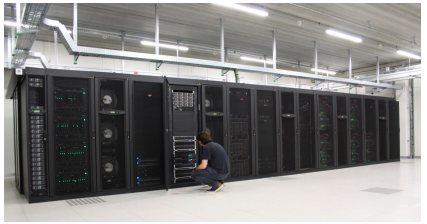
Best Practices for using HPC @ UGent

20 Nov 2025

Kenneth Hoste + Lara Peeters (HPC-UGent)

Pieter Asselman (UGent Dept. of Biology)

HPC-UGent in a nutshell



- Part of central UGent Functional Domain ICT (formerly DICT)
- Our mission:

  *HPC-UGent provides centralised **scientific computing** services, training, and support for researchers from Ghent University, industry, and other knowledge institutes.*

- Our core values:

  Empowerment - Centralisation - Automation - Collaboration

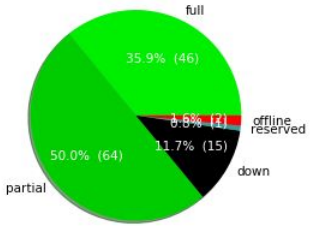
GHENT UNIVERSITY

Centralised hardware in UGent datacenter (S10 @ Sterre)

Terminology: (worker)nodes

- Example cluster from the HPC-UGent Tier-2 infrastructure: doduo *(current default cluster)*
- 128 **(worker)nodes**, also referred to as “servers”
- 1 (worker)node is the equivalent of 1 computer (but with more cores, memory, faster network, …)
- Check other HPC-UGent Tier-2 clusters

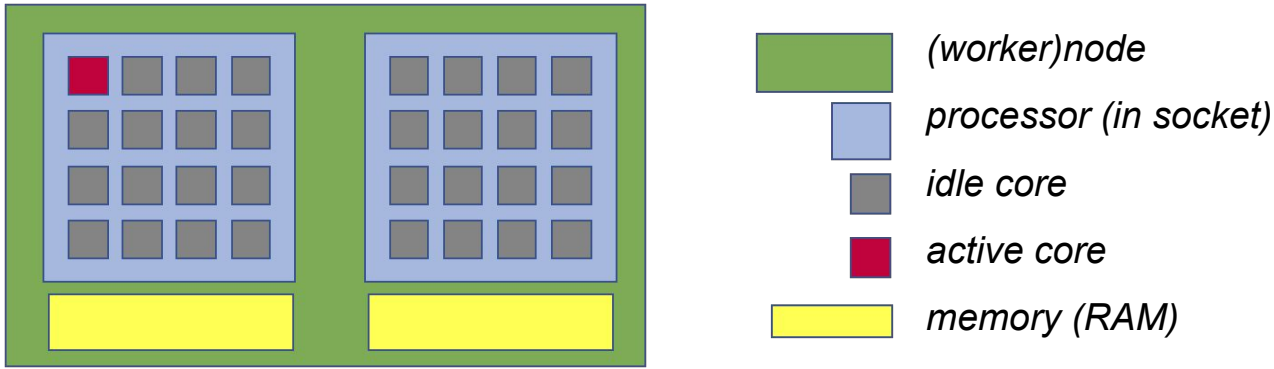
doduo



Terminology: cores, CPUs, processors

Modern servers, also referred to as **(worker)nodes** in the context of HPC, include one or more *sockets*, each housing a **multi-core processor** (next to memory, disk(s), network cards, …). A modern (micro)processor consists of **multiple cores** that are used to execute computations.

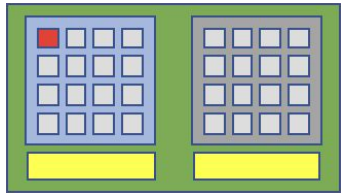
Example:
a single workernode with two 16-core processors running a single core job



Not shown here: local disk, network cards, GPUs, …

Parallel vs **sequential** software (single-core)

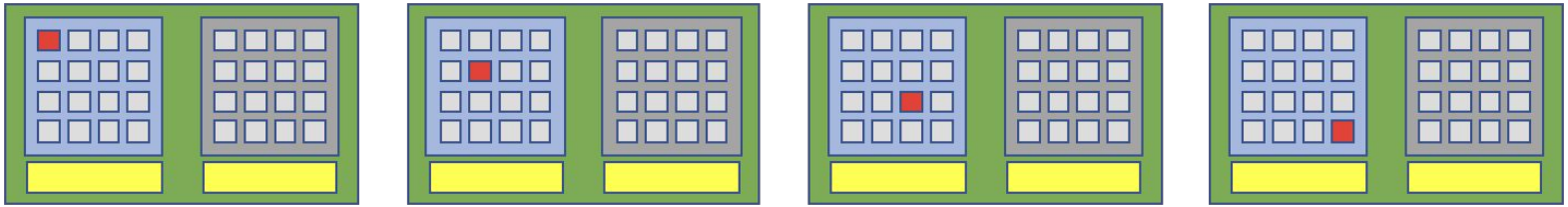
Sequential (a.k.a. serial) software does not do calculations in parallel, it only uses one **single core** of a single workernode.



This type of software does not run faster by just throwing cores (or nodes) at it...

But, you can run multiple instances of the same program at the same time!

Example: running a Python script 100 times, each on 1 core, to quickly analyse 100 datasets



Parallel vs sequential software (single-node or multi-node)

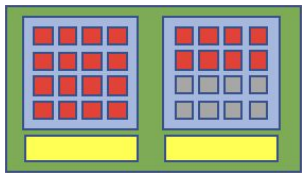
In **parallel** software, many calculations are carried out simultaneously.
This is based on the principle that large problems can often be divided into smaller tasks, which are then solved concurrently (“in parallel”).

Example: OpenFOAM can easily use 160 cores at the same time to solve a CFD problem.

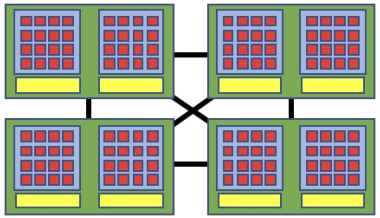
There are two common parallel programming paradigms (among others):

- **OpenMP** for shared memory systems (multi-threading) → using cores of a *single* node
- **MPI** for distributed memory systems (multi-processing) → using cores of *multiple* nodes

OpenMP software can use multiple or all cores in a single node



MPI software can use (all) cores in multiple nodes



Different “tiers” of supercomputers

eurohpc-ju.europa.eu

Tier-0
Europe

Tier-1
Regional / National

Tier-2
University

Tier-3
Personal workstation (desktop, laptop)

VLAAMS SUPERCOMPUTER CENTRUM
Vlaanderen
is supercomputing

vscentrum.be

GHENT UNIVERSITY

ugent.be/hpc

HPC-UGent Tier-2 infrastructure

- HPC-UGent Tier-2 infrastructure currently consists of **7 clusters** (+ login nodes, shared storage, …)
- Different types of clusters:
  - 3 CPU-only compute clusters
  - 3 GPU clusters
  - 1 CPU-only interactive + debug cluster (oversubcribed resources + strict user limits)
- **Available for academic researchers free of charge**, funding through FWO; usage by industry via a pay-as-you-use contract (after free exploratory period)
- All Tier-2 clusters are running Red Hat Enterprise Linux 9 (RHEL9) as operating system

GHENT UNIVERSITY

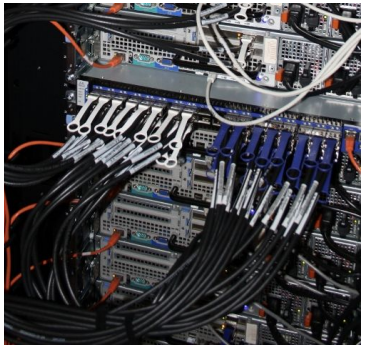
docs.hpc.ugent.be/infrastructure

Kenneth

HPC-UGent Tier-2 compute clusters

- `doduo`: 128 nodes, each with 96 cores (AMD Zen 2Rome) + ~250GiB of memory **(default)**
- `gallade`: 16 nodes, each with 128 cores (AMD Zen3 Milan) + ~940GiB of memory
- `shinx`: 48 nodes, each with 192 cores (AMD Zen4 Genoa) + ~370GiB of memory

- All with:
  - High-speed Infiniband network between nodes
  - Fast access to shared file systems
  - Fast local disk (SSD or NVMe)



docs.hpc.ugent.be/infrastructure

HPC-UGent Tier-2 GPU clusters



- `joltik`: 10 nodes,
  each with 32 CPU cores (Intel Cascade Lake),
  4 NVIDIA V100 GPUs (32GB of GPU memory),
  ~250GB of system memory
- `accelgor`: 9 nodes,
  each with 48 CPU cores (AMD Zen3 Milan),
  4 NVIDIA A100 GPUs (80GB of GPU memory),
  ~500GB of system memory
- `litleo`: 8 nodes,
  each with 48 CPU cores (AMD Zen4 Genoa),
  2 NVIDIA H100 GPUs (96GB of GPU memory),
  ~315GB of system memory

- All with high-speed network, fast access to shared filesystems, fast local disk (SSD)

docs.hpc.ugent.be/infrastructure

HPC-UGent Tier-2 interactive + debug cluster: donphan

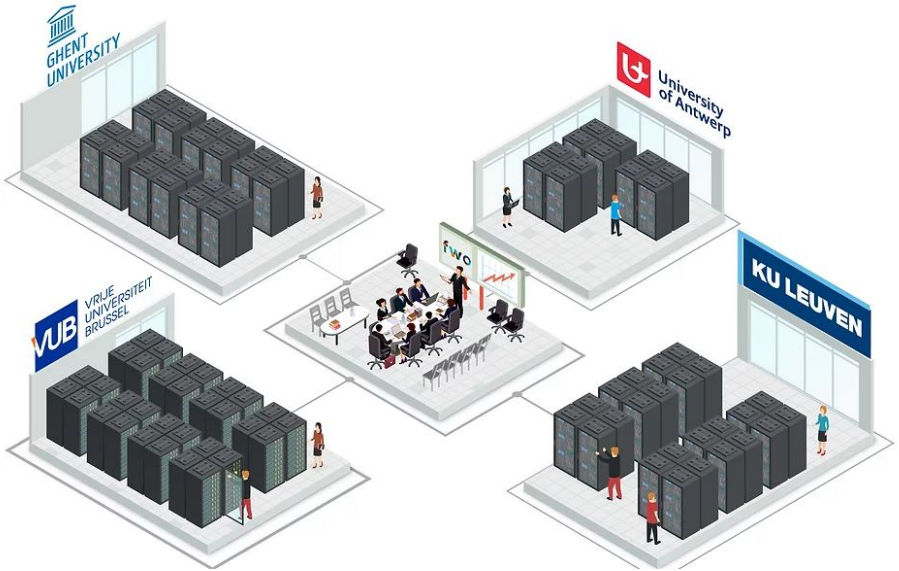
- 16 nodes, each with 36 CPU cores (Intel Cascade Lake) + ~738GB of memory

  1 *shared* NVIDIA Ampere A2 GPU (16GB of GPU memory)
- Incl. high-speed network, fast access to shared storage, local disk (NVMe)
- Recycled hardware from old `kirlia` cluster (retired in May 2023)
- **Heavily oversubscribed!** More running jobs => All jobs run slower (due to CPU sharing)
- **Strict user limits:**
  - Max. 3 jobs running, 5 jobs in queue
  - Max. 8 cores + 27GB of memory in use (in total)
- ⇒ **No waiting time for jobs to start**! Perfect for debug jobs, or interactive use (web portal)
- See also dedicated section in HPC-UGent documentation

VSC Tier-2 infrastructure

VLAAMS SUPERCOMPUTER CENTRUM
Vlaanderen is supercomputing

- You can use your VSC account to access HPC infrastructure provided by other VSC hubs
- Your `$VSC_HOME` and `$VSC_DATA` directories are available on each of these systems



docs.vscentrum.be/en/latest/hardware.html

VSC Tier-1 compute cluster “Hortense”

(a.k.a. `dodrio`)

VLAAMS SUPERCOMPUTER CENTRUM

Vlaanderen is supercomputing

compute@vscentrum.be

- Hosted, operated, and supported by HPC-UGent team since 2021
- 2x 384 CPU-only nodes (128-core AMD Rome or Milan CPUs) + 40 GPU nodes (4x NVIDIA A100)
- **Over 100,000 CPU cores in total!**
- High-speed Infiniband network (HDR-100) + 6PB of dedicated scratch storage

- **Project-based access** (free of charge, funded by FWO)
- 3 cut-off dates per year for submitting project proposals
- Project duration is typically 8 months
- 500k - 5M core hours (CPU-only) or 1k - 25k GPU hours

vscentrum.be/compute

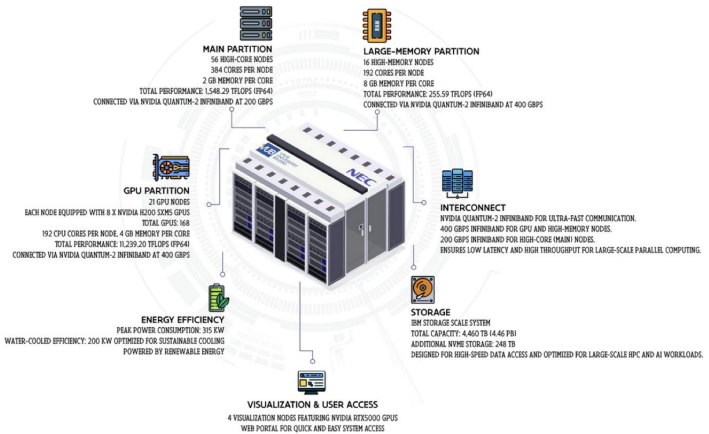
docs.vscentrum.be/en/latest/gent/tier1_hortense.html

Next VSC Tier-1 compute cluster

VLAAMS SUPERCOMPUTER CENTRUM
Vlaanderen is supercomputing

compute@vscentrum.be

- Hosted at Green Energy Park in Zellik
- Operated by Vrije Universiteit Brussel (VUB)
- Expected to be ready to use by summer 2026
- Investment of 8.6 million Euro
- **~25,000 CPU cores (with 2GB or 8 GB RAM per core)**
- **168 NVIDIA H200 GPUs**
- **NVIDIA Quantum-2 InfiniBand network at 200 Gbps**
- 4 visualisation nodes
- 4460 TB of scratch storage + 24 TB NVME
- Peak power consumption of 315 kW,200 kW water-cooled for improved energy efficiency



vscentrum.be/compute
vscentrum.be/post/flanders-invests-in-a-new-supercomputer-to-accelerate-research-and-innovation

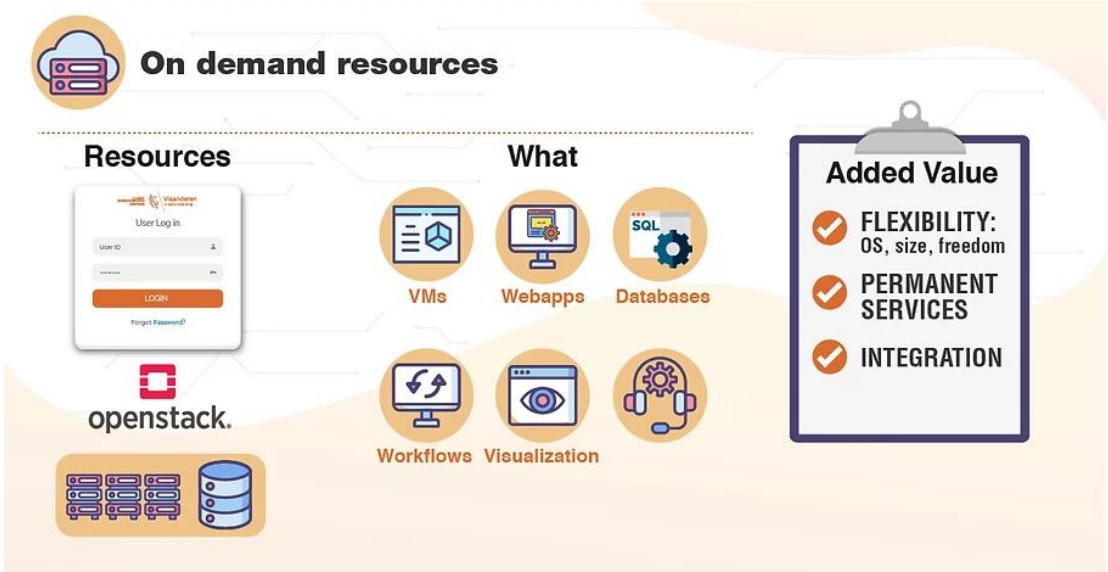
Kenneth

VSC Tier-1 cloud

VLAAMS SUPERCOMPUTER CENTRUM

Vlaanderen is supercomputing

- **Project-based access**
- Free of charge
- **Self-managed virtual machines**
- For use cases that are not a good fit for compute clusters
- More info: vscentrum.be/cloud
- Contact: cloud@vscentrum.be



Kenneth

Getting a VSC account

VLAAMS SUPERCOMPUTER CENTRUM
Vlaanderen is supercomputing

- All members of UGent association can request a VSC account
  - Researchers & staff
  - Master/Bachelor students
- **VSC account can be used to access HPC infrastructure on all VSC sites**
- Subscribed to `hpc-announce` and `hpc-users` mailing lists
- Beware of using HPC for teaching/exam purposes!
  - No guarantee on HPC availability (due unexpected power outage, maintenance, …)
  - Have a backup plan at hand
  - Advisable teaching/exam formula: project work
- See also HPC-UGent documentation

ugent.be/hpc/en/access/faq/access

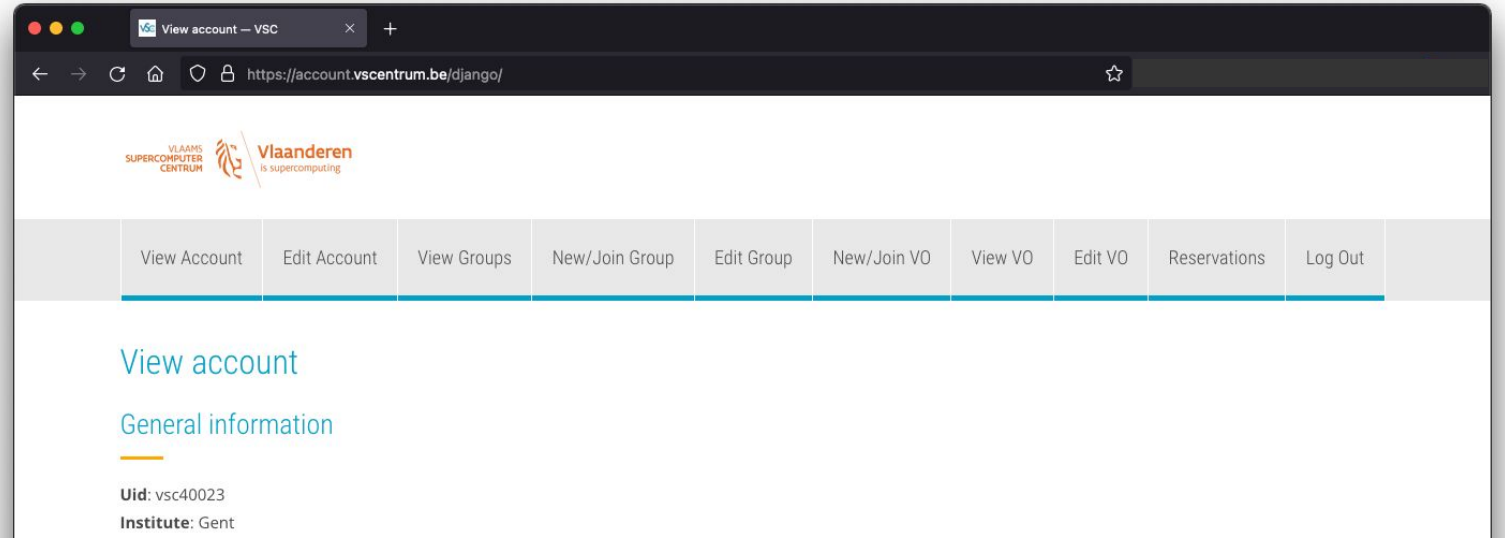
Managing your VSC account

You can manage your VSC account via the VSC account page

account.vscentrum.be

Can be used to join/leave user groups, consult storage usage, request more storage quota, …

manage your Virtual Organisation (VO), …



High-level overview of HPC-UGent infrastructure

Getting access

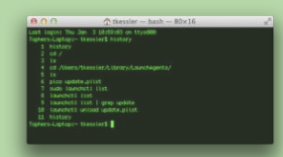
https://login.hpc.ugent.be

Web Portal (No SSH key required)

OR

ssh login.hpc.ugent.be

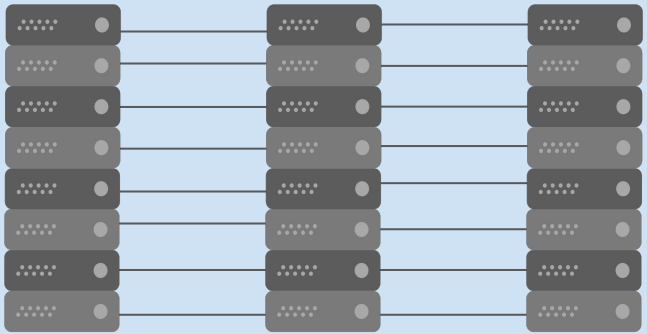
Terminal (requires SSH key)



Login nodes

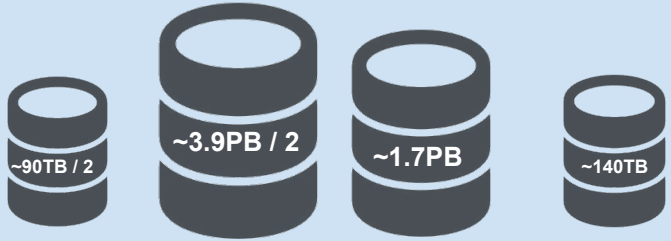
gligar09 or gligar10

Compute nodes (where your jobs will run)



Data

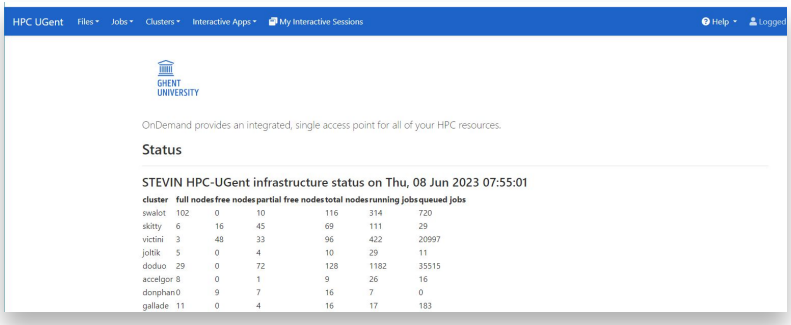
File system



Home | Data | Scratch | Arcanine Scratch (SSD)

Connecting to the HPC-UGent login nodes with web portal

Recommended!



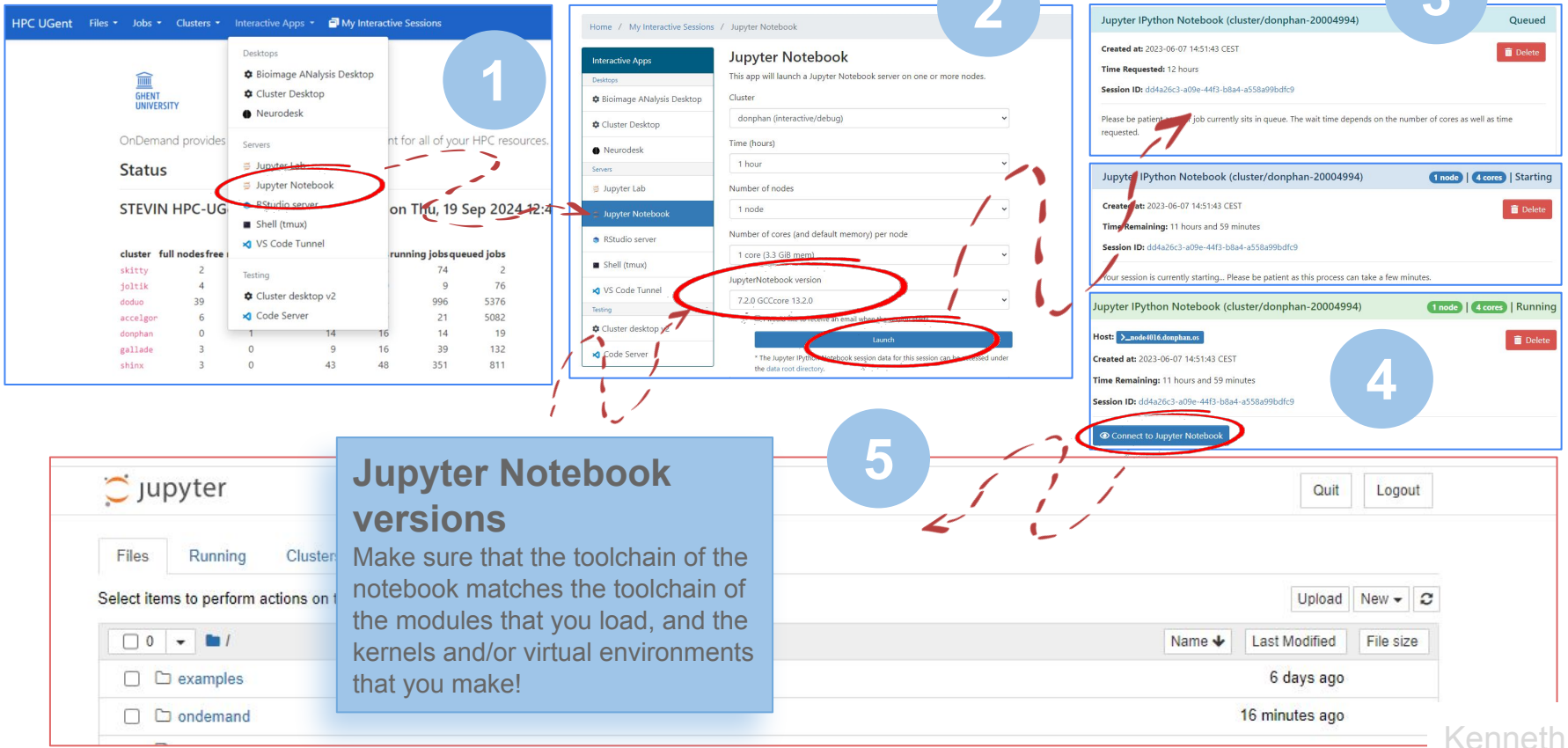
- See dedicated section of HPC-UGent docs
- https://login.hpc.ugent.be
- Powered by Open OnDemand

- **Works with a standard internet browser** (Firefox, Chrome, …)
- **Does not require SSH key pair** (only login via UGent account)
- Provides file browser, shell session, desktop environment, interactive apps, …

Kenneth

UGent web portal: interactive apps (Jupyter notebook)

(Lara)

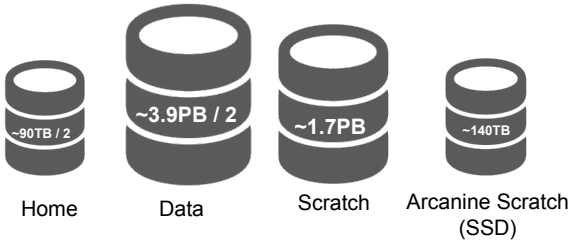


Jupyter Notebook versions

Make sure that the toolchain of the notebook matches the toolchain of the modules that you load, and the kernels and/or virtual environments that you make!

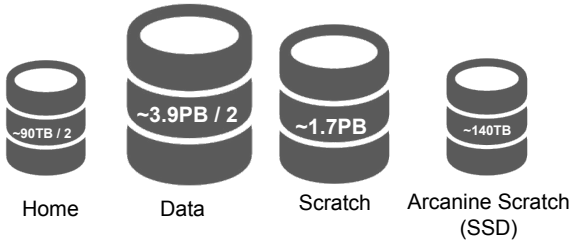
Kenneth

Input/output data and shared filesystems



- See dedicated section in HPC-UGent documentation
- Think about input/output:
  - How and where will you *stage in* your data and input files?
  - How and where will you *stage out* your output and result files?
- Manually (on login nodes) vs automatically (as a part of job script)
- **Home filesystem** (`$VSC_HOME`): only for limited number of small files & scripts
- **Data filesystem** (`$VSC_DATA*`): 'long-term' storage, large files
- **Scratch filesystems** (`$VSC_SCRATCH*`): for 'live' input/output data in jobs

Storage quota (disk space)



- Home directory (`$VSC_HOME`): 3GB (fixed!)
- Personal data directory (`$VSC_DATA`): 25GB (fixed!)
- Personal scratch directory (`$VSC_SCRATCH`): 25GB (fixed!)
- Current quota usage can be consulted on VSC accountpage
- **More storage quota (100s of GBs, even TBs) available for *virtual organisations (VOs)***; see dedicated section on VOs in HPC-UGent documentation
- Additional quota can be requested via VSC accountpage ("Edit" tab)
- Shared directories with VO members: `$VSC_DATA_VO`, `$VSC_SCRATCH_VO`
- Personal VO subdirectories: `$VSC_DATA_VO_USER`, `$VSC_SCRATCH_VO_USER`

Job scheduling

- All HPC-UGent clusters use a **fair-share scheduling** policy.
- No guarantees on when job will start (and impossible to predict), so **plan ahead**!
- Job priority is determined by various factors:
  - Historical usage
    - Aim is to balance usage over users
    - Infrequent/frequent users => higher/lower priority
  - Requested resources (# nodes/cores, walltime, memory, ...)
    - Larger resource request => lower priority
  - Time waiting in queue
    - Queued jobs get higher priority over time
  - User limits
    - Avoid that a single user fills up an entire cluster

Software installations

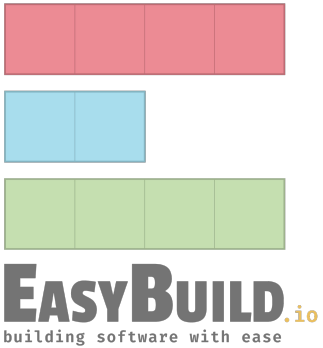
- To submit a request for software installation, use the request form:

  docs.hpc.ugent.be/software_installation_requests

- Requests may take a while to process (especially for new software), so **be patient**…
- Make the request sooner rather than later!

- All software installations are done using EasyBuild
- Originally developed by HPC-UGent,

  now a worldwide community of experts!
- See also easybuild.io



Questions, problems, getting help

Don't hesitate to contact the HPC-UGent Tier-2 support team via hpc@ugent.be

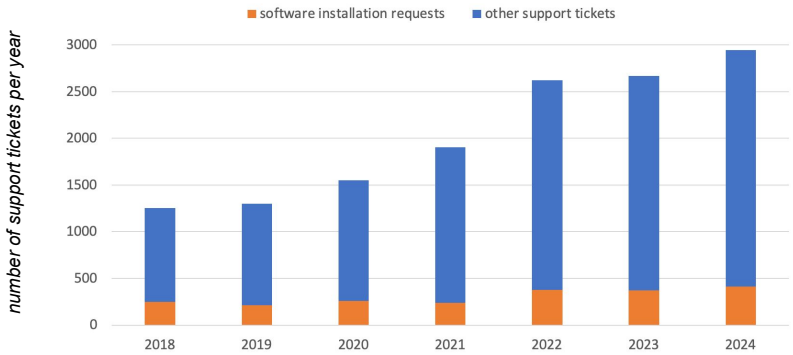
- Always include:
  - VSC login id
  - Clear description of the problem or question, include error messages, …
  - Location of job script and output/error files in your VSC account
  - Preferably don’t send files in attachment, we prefer to look at it “in context”
  - Also mention job IDs, which cluster was used, how job was submitted, etc.
- Preferably use your UGent email address
- Alternatives:
  - Short (Teams) meeting (for complex problems, big projects)
  - `hpc-users` mailing list

Questions, problems, getting help (be patient…)

Don't hesitate to contact the HPC-UGent support team via hpc@ugent.be,

but be patient…

- We're doing what we can to keep up with incoming support questions + software installation req.
- We have been getting **50-100 new tickets per week** recently, and have a backlog of ~100 tickets
- **Help us help you**: read the docs, provide sufficient details (like job IDs, output files, etc.), …
- Feel free to send a reminder in the same ticket or mail thread, especially if your work is blocked



What if you no longer have to install

*a **broad range of scientific software***

from scratch on every laptop, HPC cluster,

or cloud instance you use or maintain,

without compromising on performance?

Lara

European Environment for Scientific Software Installations (EESSI)

- **Shared repository of (optimized!) scientific software *installations***
- Avoid duplicate work across by collaborating on a shared software stack
- Uniform way of providing software to users, regardless of the system they use!
- Should work on any Linux OS and system architecture
  - From laptops and personal workstations to HPC clusters and cloud
  - Support for different CPUs, interconnects, GPUs, etc.
- Focus on **performance, automation, testing, collaboration**
- Development effort funded through **MultiXscale** EuroHPC Centre-of-Excellence

https://eessi.io

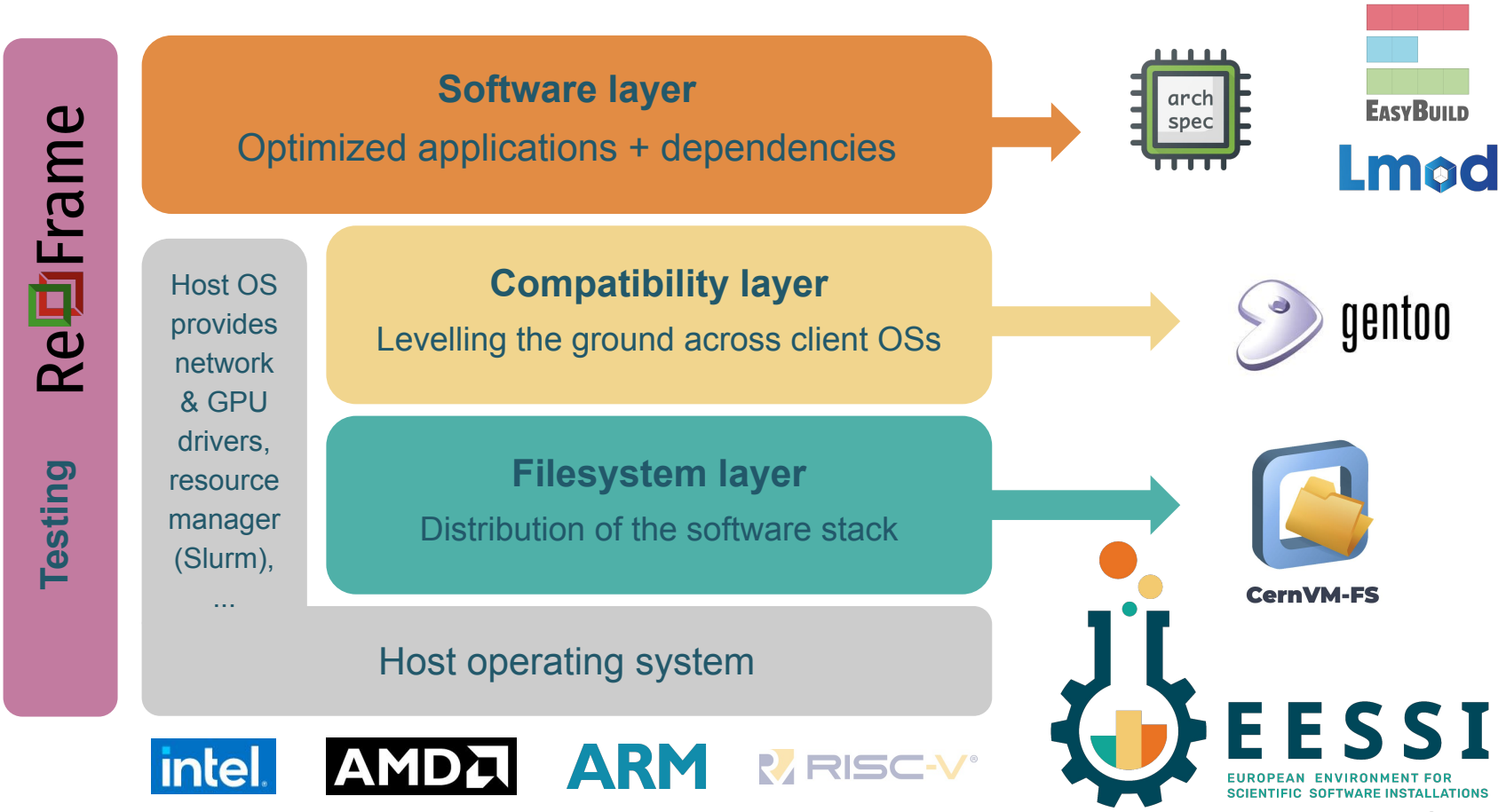
Lara

Major goals of EESSI

- Providing a truly **uniform software stack**
  - Use the (exact) same software environment everywhere
  - **Without sacrificing performance** for “mobility of compute” (like is typically done with containers/conda)
- **Avoid duplicate work** (for researchers, HPC support teams, sysadmins, …)
  - Tools that automate software installation process (EasyBuild, Spack) are not sufficient anymore
  - Go beyond sharing build recipes => work towards a shared software stack
- Facilitate HPC training, development of (scientific) software, …

Lara

High-level overview of EESSI



EESSI ingredients

Compatibility layer

Abstraction from the host operating system

Filesystem Layer

Global distribution of software installations via CernVM-FS

Software Layer

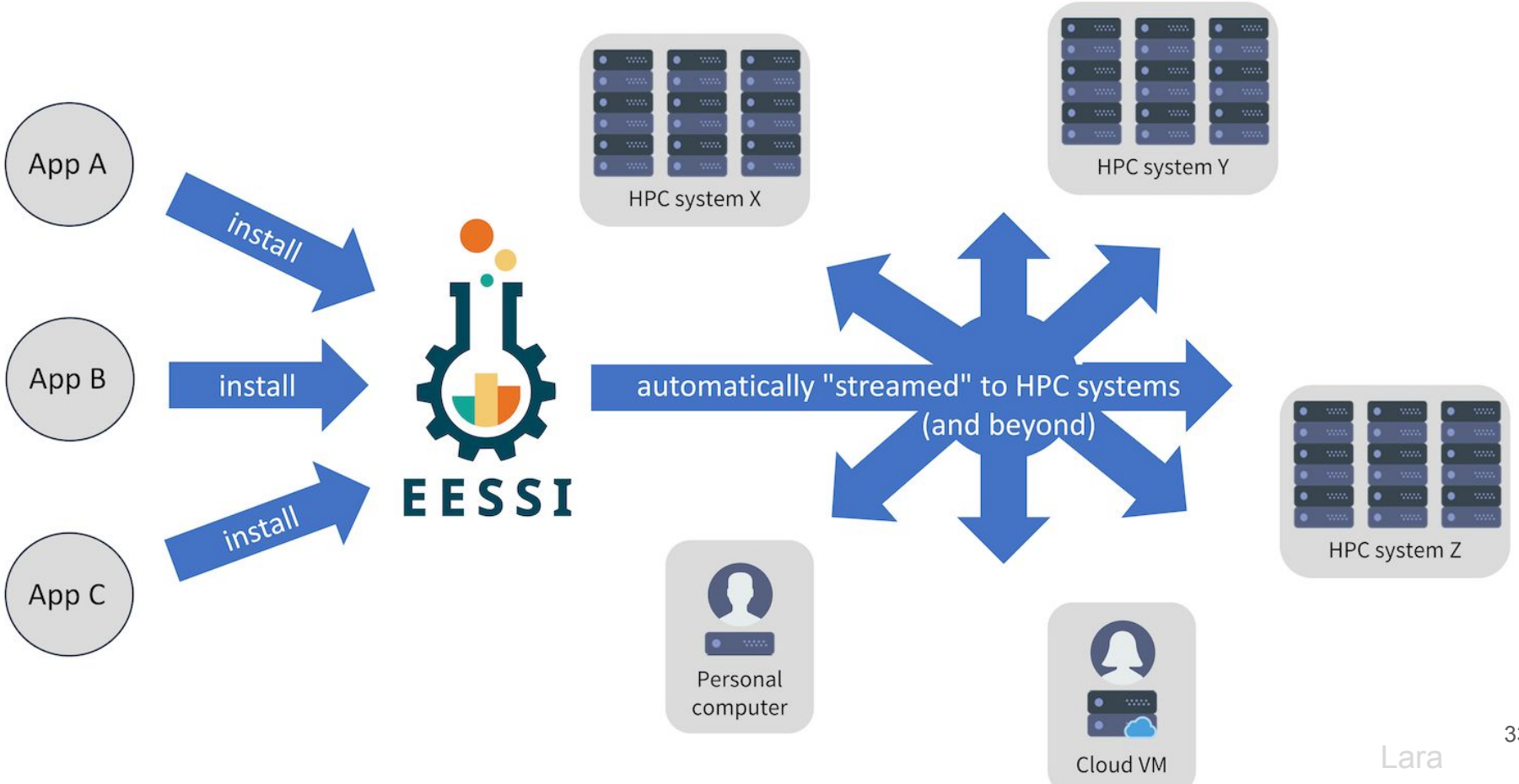
Optimized software installations for specific CPU microarchitectures

Intuitive user interface:
module avail,
module load, …

Automatic selection of best suited part of software stack for CPU microarchitectures

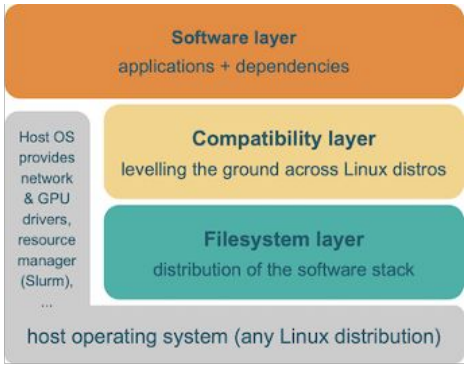
Lara

EESSI as a shared software stack



Lara

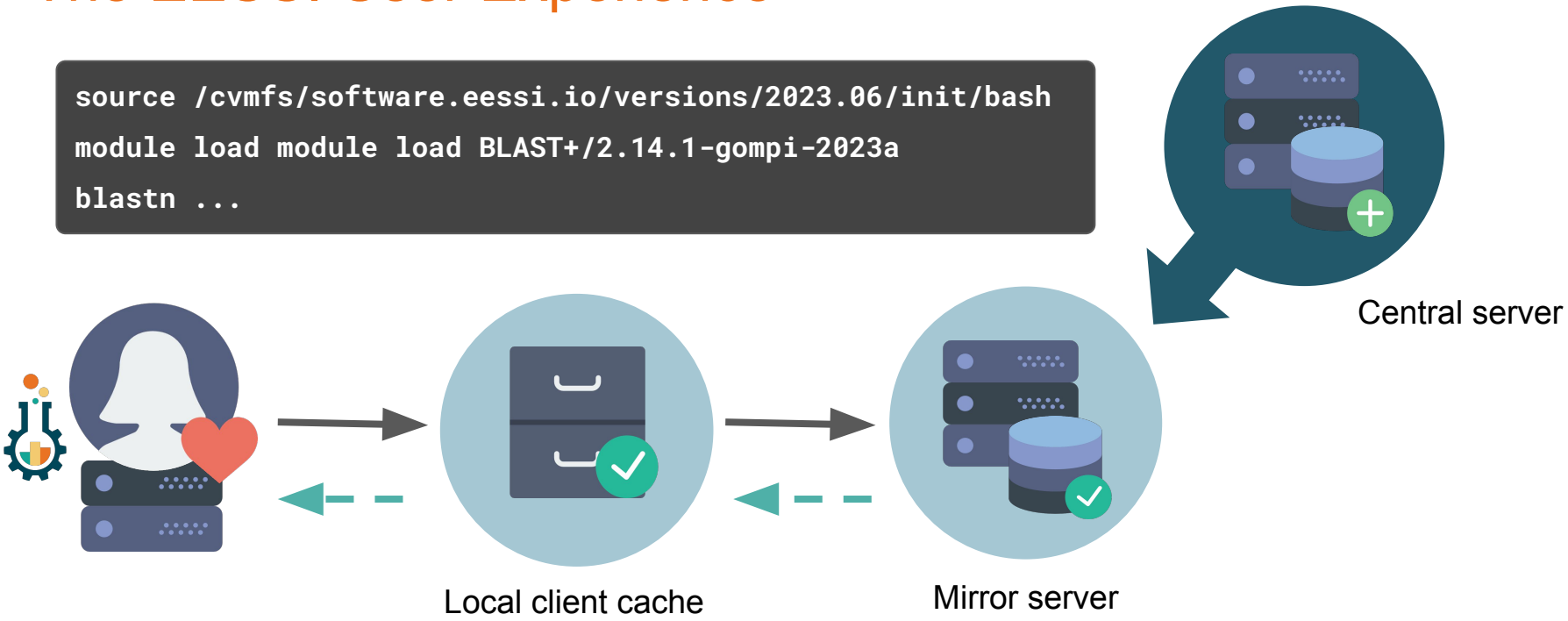
How does EESSI work?



- Software installations included in EESSI are:
  - Automatically **“streamed in” on demand** (via CernVM-FS)
  - Built to be **independent of the host operating system**
    “Containers without the containing”
  - **Optimized** for specific CPU generations + specific GPU types
- Initialization script **auto-detects** CPU + GPU of the system

The EESSI User Experience

```
source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load module load BLAST+/2.14.1-gompi-2023a
blastn ...
```



EESSI provides **on-demand streaming**

of (scientific) software (like music, TV-series, …)

Demo: Using EESSI on HPC-UGent Tier-2

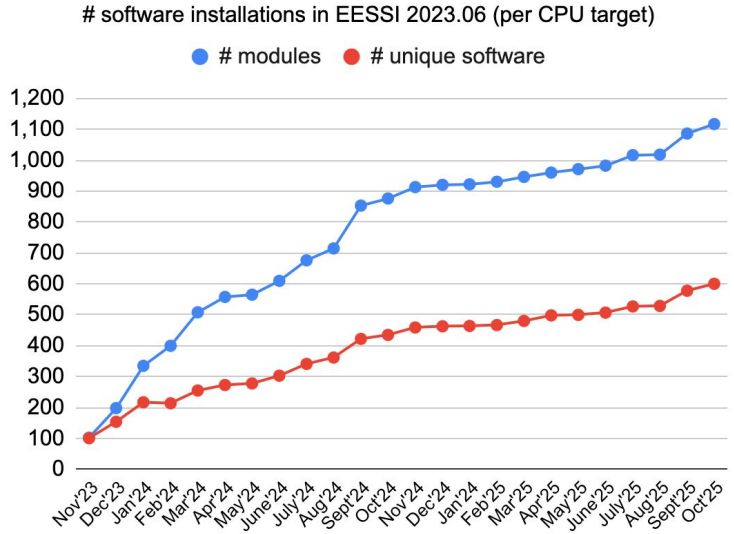
```
# Access via HPC-UGent web portal (https://login.hpc.ugent.be)
# Start interactive shell session on donphan cluster
module --force purge   # this is not required, but it is recommended to start afresh
source /cvmfs/software.eessi.io/versions/2023.06/init/bash
   Found EESSI repo @ /cvmfs/software.eessi.io/versions/2023.06!
   archdetect says x86_64/intel/cascaselake
   archdetect found supported accelerator for CPU target x86_64/intel/cascadelake: accel/nvidia/cc80
   Using x86_64/intel/cascadelake as software subdirectory.
   Using /cvmfs/software.eessi.io/versions/2023.06/software/linux/x86_64/amd/cascadelake/modules/all
   Using ...
   ...
   Environment set up EESSI (2023.06), have fun!
module avail BLAST
module load BLAST+/2.14.1-gompi-2023a
```

Alternative ways of accessing EESSI are available, via a container image, via cvmfsexec, …
eessi.io/docs/getting_access/native_installation - eessi.io/docs/getting_access/eessi_container

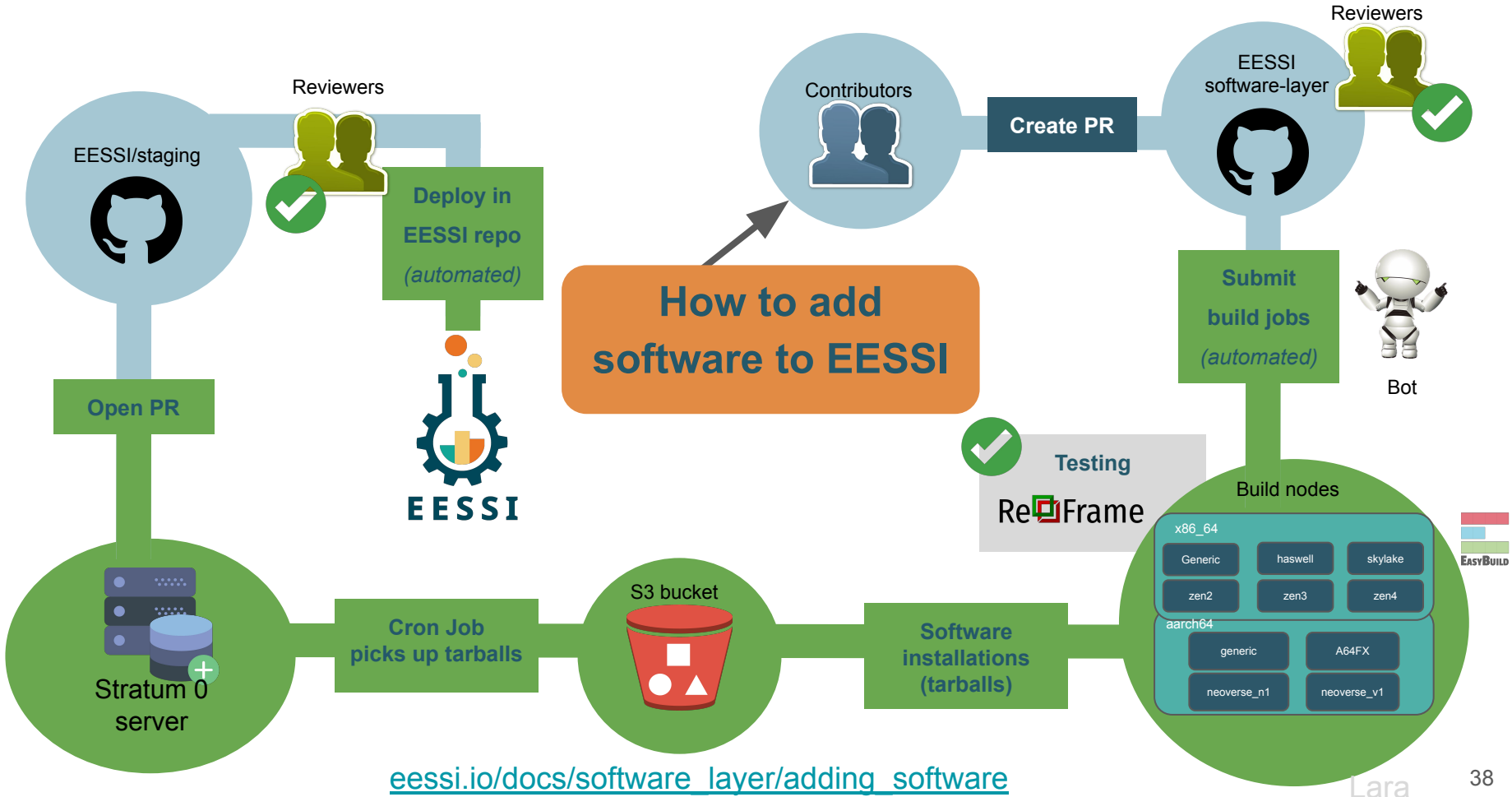
Overview of available software in EESSI

Currently more than 1,100 software installations available per supported CPU target via `software.eessi.io` CernVM-FS repository, increasing every week

- **14 fully supported CPU targets** (x86_64 + Arm), see https://eessi.io/docs/software_layer/cpu_targets
- **~600 different software packages**, excl. extensions: Python packages, R libraries
- **Almost 16,000 software installations in total**
- Including **BLAST+**, ESPResSo, **Flye**, GROMACS, LAMMPS, **MrBayes**, OpenFOAM, PyTorch, R, QuantumESPRESSO, TensorFlow, waLBerla, WRF, …
- https://eessi.io/docs/available_software/overview
- Using `foss/2023a` and `foss/2023b` toolchains in EESSI 2023.06
- Using `foss/2024a` and `foss/2025a` toolchains in EESSI 2025.06



Semi-automated workflow for adding software to EESSI



eessi.io/docs/software_layer/adding_software

Building software on top of EESSI

- You can use the software available in EESSI to

  resolve dependencies of the software you need

- To install software with EasyBuild on top of EESSI,

  first load the `EESSI-extend` module to correctly configure EasyBuild

- To manually install software on top of EESSI,

  first load the `buildenv` module

- See also https://eessi.io/docs/using_eessi/building_on_eessi

On which systems is EESSI available?

- On VSC systems:
  - **Tier-2 infrastructure at UGent + VUB + Tier-1 Hortense (+ new Tier-1 soon)**
  - Tier-1 cloud: can deploy it yourself and have access to a full software stack in minutes
- EESSI is already available on various other European systems (and beyond)
  - EuroHPC JU systems incl. Vega, Karolina, MareNostrum 5, Deucalion, Discoverer, …
  - Snellius @ SURF, EMBL, Univ. of Stuttgart, Sigma2 in Norway, etc.
- EESSI can be used in virtual machine in European Open Science Cloud (EOSC),

  see also https://www.eessi.io/docs/blog/2025/10/22/eosc
- **Overview of (known) systems that have EESSI available at https://eessi.io/docs/systems**

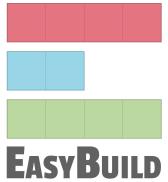
Webinar series: Different aspects of EESSI

Series of presentations (May-June 2025)

https://eessi.io/docs/training/2025/webinar-series-2025Q2

- Introduction to EESSI
- Introduction to CernVM-FS
- Introduction to EasyBuild
- EESSI for CI/CD
- Using EESSI as the base for a system stack

Slides + recordings available



Lara

Best practices for running jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

- **Read the documentation** (HPC-UGent, software, …)
- **Check resource usage** (CPUs, GPUs, memory, …)
- Take into account **I/O overhead**: try different shared filesystems
- Use **more resources** (cores/nodes/memory), but check if that helps! (**scaling**)
- Consider using **different clusters** (newer is usually faster)
- Try using r**ecent software versions**: newer is usually better (but take care)
- Check details of **how the software was installed**:
  conda vs central software stack vs EESSI vs containers
- **Don’t waste time over-optimizing…**

Resources?

VLAAMS SUPERCOMPUTER CENTRUM

Different types of resources

- CPU **cores** (# cores, a.k.a. `ppn`), also **# nodes** and # **GPUs** (if relevant)
- **Memory** per node (vmem)
- **Walltime**: amount of times that passes between start and end of width
- Also: (derived) memory per *core*, (local) storage space, network bandwidth, ...

Some tips w.r.t. requesting resources

VLAAMS SUPERCOMPUTER CENTRUM

- **Small(er) resource request** in terms of cores/nodes/memory/walltime

  => Usually **less waiting time** for jobs in queue

  (since a smaller “gap” is required)

- **Only request what you need**, and can **use efficiently**
  - Don’t blindly request more cores, check if it results in (signficant) **speedup**!
  - Evaluate impact of requesting more resources on job walltime
- **Total wait time** (for you) **= queue time + (wall)time required to complete job**
- Rule of thumb: if possible, use **quarter of a node** (or less) in terms of cores, helps to minimize waiting time in queue

Checking resource usage: `slurm_jobinfo`

VLAAMS SUPERCOMPUTER CENTRUM

```
$ slurm_jobinfo 12345678
Name                    : example
User                    : vsc40000
Partition               : shinx
Nodes                   : node4201.shinx.os
Cores                   : 8
State                   : COMPLETED
Submit                  : 2025-11-10T10:58:47
Start                   : 2025-11-10T11:01:58
End                     : 2025-11-10T18:49:44
Reserved walltime       : 3-00:00:0.0
Used walltime           :   07:47:46.0
Used CPU time           :   07:38:54.0
% User (Computation)    : 82.03
% System (I/O)          : 17.97
Mem reserved            : 15G
Max Mem used            : 14.97G (node4201.shinx.os)
```

Specify job ID
(make sure correct `cluster` module is loaded)

Used CPU time should be roughly equal to # cores times used walltime

Large ratio of system CPU is not good

Scaling up: use more resources, but…

VLAAMS SUPERCOMPUTER CENTRUM

- More cores should result in ***significant* speedup** (but maybe less than you expect), *if* the software supports multi-threading or multi-processing
- Check **used walltime vs used CPU time** for your jobs
  - CPU time should be close to `walltime` × `number-of-cores`
- Take into account **efficiency** (speedup divided by # cores), aim for min. ~50-60%
  - Avoid wasting resources, and longer waiting times in queue
- Requesting **multiple nodes** only makes sense if your software supports **multi-processing** (typically via MPI)
- Make sure the software you are using can make **efficient use of GPUs** (if at all)

Checking resource usage: live job inspection

VLAAMS SUPERCOMPUTER CENTRUM

- Sometimes it can be interesting to inspect the job **while it is running**…
- Check with `qstat -ant` on which node(s) your job is running
- Log into the node with `ssh node1234`
- A couple of useful commands:
  - `top`, `htop`, `btop` → total resource usage on node (quit with `q` or `Ctrl-C`)
  - `ps xfu` → list of running processes (for your VSC account), incl. CPU usage
  - `taskset -c -p 12345` → core pinning for process with ID 12345
  - `pidstat -u -p 12345 1` → CPU usage for with ID 12345, refresh every 1 sec.

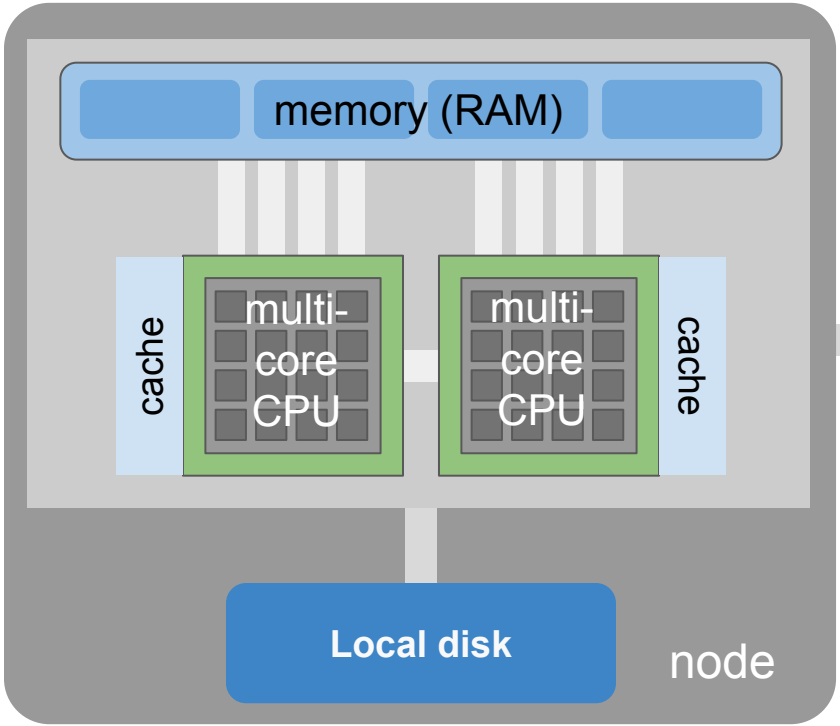
The `$OMP_PROC_BIND` "curse"

VLAAMS SUPERCOMPUTER CENTRUM

- `$OMP_PROC_BIND` is automatically set to `TRUE` in all jobs on HPC-UGent Tier-2 (and VSC Tier-1 @ UGent)
- Significant performance boost for *some* workloads that use OpenMP threading
- However, some other workloads (R, Python, BLAST+) get "locked" to a single core during startup of the software, and can not escape that jail anymore at runtime…
  - We reported a bug on this for the OpenMP runtime in GCC, but got told this is by design… see https://gcc.gnu.org/bugzilla/show_bug.cgi?id=114765
- **You may need to use "`unset OMP_PROC_BIND`" in your job script** to avoid that all threads are pinned to a single core… **Check scaling when using multiple cores!**
- If pinning of threads to cores is important, you should use the Likwid tool, see https://github.com/RRZE-HPC/likwid

High-level overview of a (worker)node

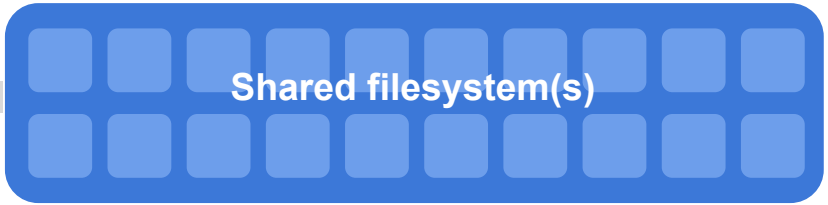
VLAAMS SUPERCOMPUTER CENTRUM



Types of disk:

HDD | SSD | NVME

slow → fast

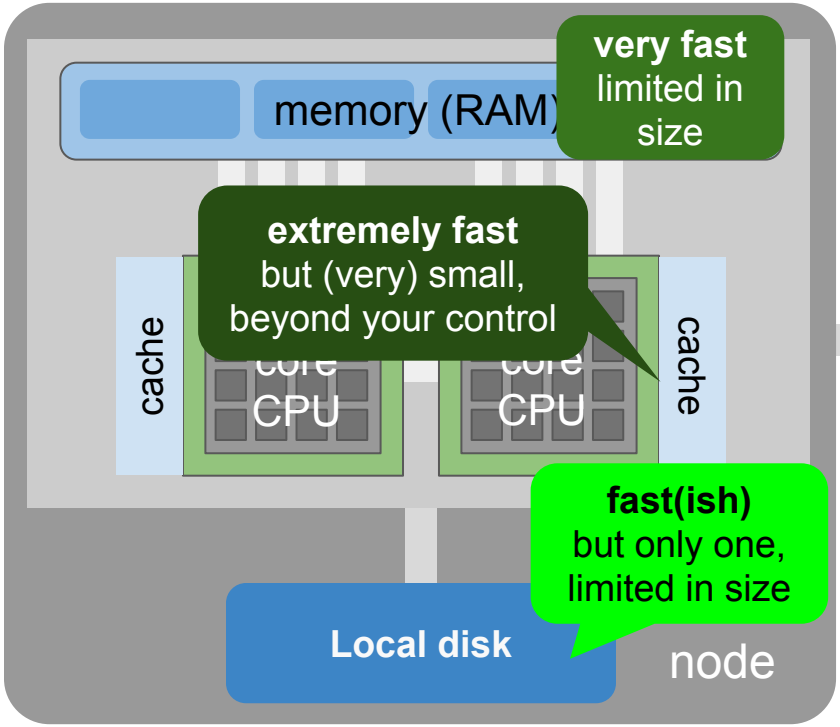


Filesystems @ HPC-UGent Tier-2:

$VSC_HOME	“slow” & small
$VSC_DATA	“slow” & large (+ safe)
$VSC_SCRATCH	fast & large
$VSC_SCRATCH_ARCANINE	faster & medium size

Speed in which data can be read (or written)

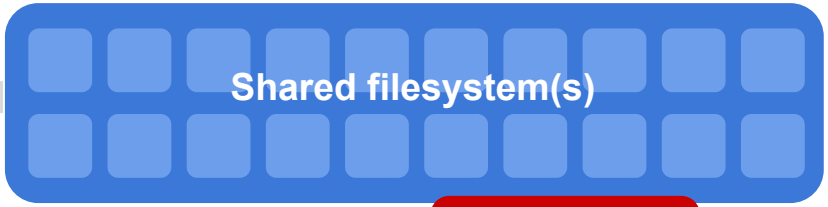
VLAAMS SUPERCOMPUTER CENTRUM



Types of disk:

HDD | SSD | NVME

slow → fast



Filesystems @ HPC-UGe

Don't use this for data

$VSC_HOME

"slow" & small

slow(ish)

$VSC_DATA

"slow" & large (+ safe)

fast (& large)

$VSC_SCRATCH

st & large

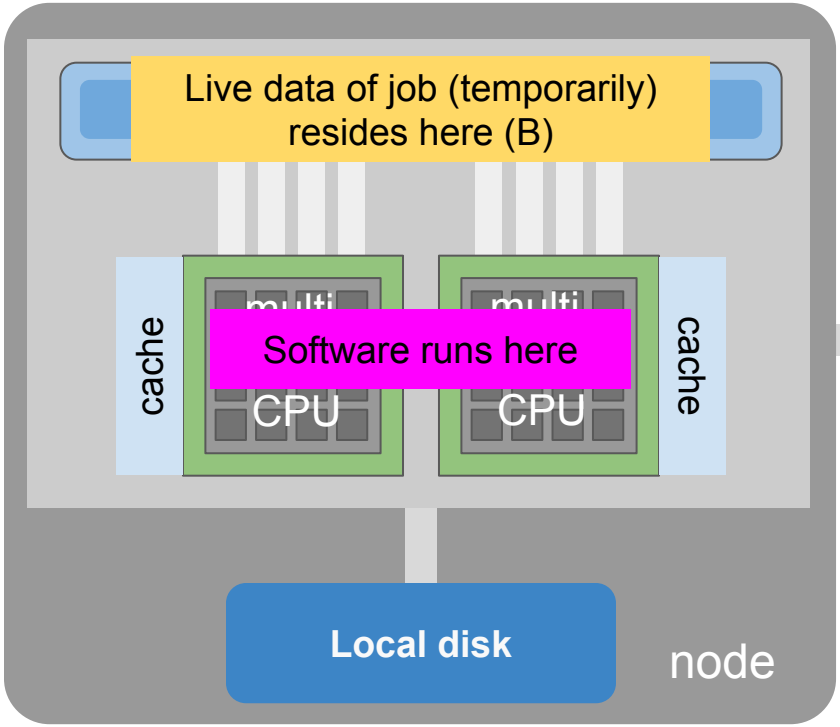
$VSC_SCRATCH_ARCANINE

edium size

faster

Where does your data come from?

VLAAMS SUPERCOMPUTER CENTRUM

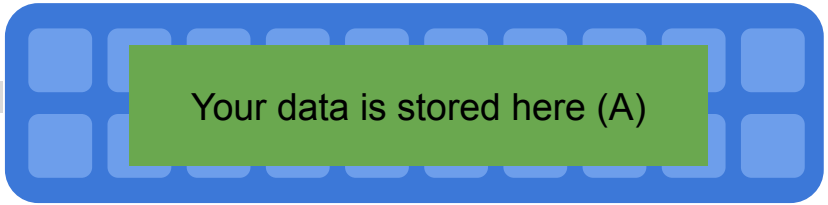


Types of disk:

HDD | SSD | NVME

slow → fast

How the (input) data for your jobs gets from A to B can matter (a lot), especially when it's large volume (GBs), lots of (small) files, or is accessed "randomly".



Filesystems @ HPC-UGent Tier-2:

$VSC_HOME	"slow" & small
$VSC_DATA	"slow" & large (+ safe)
$VSC_SCRATCH	fast & large
$VSC_SCRATCH_ARCANINE	faster & medium size

I/O overhead: time spent on reading/writing data

VLAAMS SUPERCOMPUTER CENTRUM

- **Reading data from disk (shared filesystems) can be “slow”**
  - CPUs may not be able to get useful work done because of that…
- Correlates with “`% System (I/O)`” in `slurm_jobinfo` output
  - Should be << 5%, or it will significantly slow down your job
  - Can be difficult to avoid for some type of workloads, but worth paying attention to
- Live job inspection with `htop`, `pidstat` can help you understand this better
- If used CPU time is not roughly equal to number of cores x used walltime, a lot of time is being lost to… something
- **Consider accessing the data through a different/faster shared filesystem**: `$VSC_SCRATCH`, `$VSC_SCRATCH_ARCANINE`

Escaping I/O overhead by using local or RAMdisk

VLAAMS SUPERCOMPUTER CENTRUM

- If consuming data from shared filesystems is too slow (large I/O overhead), you can try using **local (SSD) disk or RAMdisk** (mostly interesting for single-node jobs)

```
export INPUT_DATA=$VSC_DATA_VO/large_dataset
export FAST_TMPDIR=/dev/shm/$USER    # or $TMPDIR (local disk)
rsync -avP $INPUT_DATA $FAST_TMPDIR
export INPUT_DATA=$FAST_TMPDIR/large_dataset
do_work_with $INPUT_DATA
```

- Streaming copy from shared filesystem is usually fast (vs random access during job)
- Take into account that:
  - Time needed to copy the data is part of job walltime, so only interesting for long jobs
  - Size of the dataset may make this impossible (but may be possible on gallade)

Software installations - caveats with Conda & co

VLAAMS SUPERCOMPUTER CENTRUM

- ***How*** the software was compiled/installed can impact **performance** a lot
- Software should be **optimized** for the specific CPUs on which it will be run
  - This is usually not the case when using container images or Conda environments…
  - Generic binaries are used that can run anywhere
  - Performance is sacrificed for mobility of compute…
- Vast majority of software installed in central software stack on HPC-UGent infrastructure is optimized for the specific CPUs of each cluster
  - Likewise for EESSI: broad range of CPUs supported, specific CPU is auto-detected
- Additional caveats with Conda: messes up your home directory very easily…

Attention points when benchmarking performance

VLAAMS SUPERCOMPUTER CENTRUM

- You are working on a shared system, with **shared resources** => chaos…
- **Some things are outside of your control**
  - Load on shared filesystems varies a lot, you may see peaks in I/O overhead
  - Placement of jobs w.r.t. cores & sockets, memory channels, …
  - Job(s) by other users running on same node may indirectly impact your job
- **CPU clock speed varies over time** based on load on CPU cores (throttling/boosting)
- Make sure you're running the **same workload** when comparing job walltimes!
  - Watch out with "converging" workloads…
  - Some software is "smart", will detect or change things at runtime…

EESSI-Hackathon inquiry

VLAAMS SUPERCOMPUTER CENTRUM

3. Do you have experience with installing software on a linux machine?Required to answer?

More Details

yes	5
No	3

4. If you answered "Yes" to question 3, which method did you use?Required to answer.

More Details

Conda	5
Containers (Docker, Apptainer,...)	0
Using Easybuild	0
Using EESSI platform (rather usi...	0
I'm an expert, I only install from ...	0

5. Are you interested to see a quick demo on how to install software on HPC yourself via:

More Details

Conda	3
Container	0
Easybuild	1
EESSI	3
None of the above, I'm only inte...	1

“Do you use HPC for you BioInfo analyses?”

YES!

No, too many limitations.

Yes, GPU power!

Sometimes, I prefer long interactive sessions.

Yes, runs faster than my local laptop.

Yes, local servers are running out of storage.

Getting Scientific Software Installed 2.0?
https://www.vscentrum.be/

see https://github.com/vscentrum/gssi-training
(recording available soon)

Pieter

MrBayes: An overview

VLAAMS SUPERCOMPUTER CENTRUM

Bayes' theorem: what is the probability that a model parameter is true, given data and prior beliefs?

MrBayes is a software program used for **Bayesian phylogenetic inference**.

In simple terms:

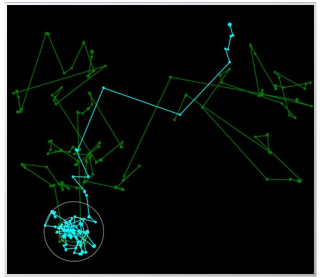
- It builds evolutionary trees using Bayesian statistics and Markov Chain Monte Carlo (MCMC) sampling to estimate the most likely tree and the support for each branch.
- **Θ** includes parameters such as topologies, branch lengths; **X** is your sequence alignment

prior distribution ← $f(\theta) f(X \mid \theta)$ → likelihood

posterior distribution ← $f(\theta \mid X) = \dfrac{f(\theta) f(X \mid \theta)}{\int f(\theta) f(X \mid \theta) d\theta}$

normalizing constant ←

Θ = model parameter
X = data



- 2-dimensional image of posterior probability space
- 2 chains
- poor mixing
- no convergence

Pieter

MrBayes: Workload info

VLAAMS SUPERCOMPUTER CENTRUM

- Compute bound, so shinx (recent) should be faster than `doduo` (old)...
- Can use multiple cores (via Beagle dependency)
  - Via **multi-threading** (native threading, not OpenMP)
  - Via **multi-procesing** (MPI)
  - How well does it scale with # cores?
  - Is it worth using multiple *nodes*?
- Can also use **GPU**, that sounds promising…
- Only a small input file (~6MB), so no problems w.r.t. I/O
- Settings in `*.nex` file:

```
mcmc nruns=2 nchains=4 temp=0.1 ngen=5000000
samplefreq=1000 printfreq=1000 diagnfreq=5000;
```

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

- Naive starting point: **single-core job**
- **Takes ~65h on shinx**

```
#!/bin/bash
#PBS -l nodes=1:ppn=1
#PBS -l walltime=72:00:00

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load MrBayes/3.2.7-gompi-2023a


cd $PBS_O_WORKDIR

mb mbtest.nex
```

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

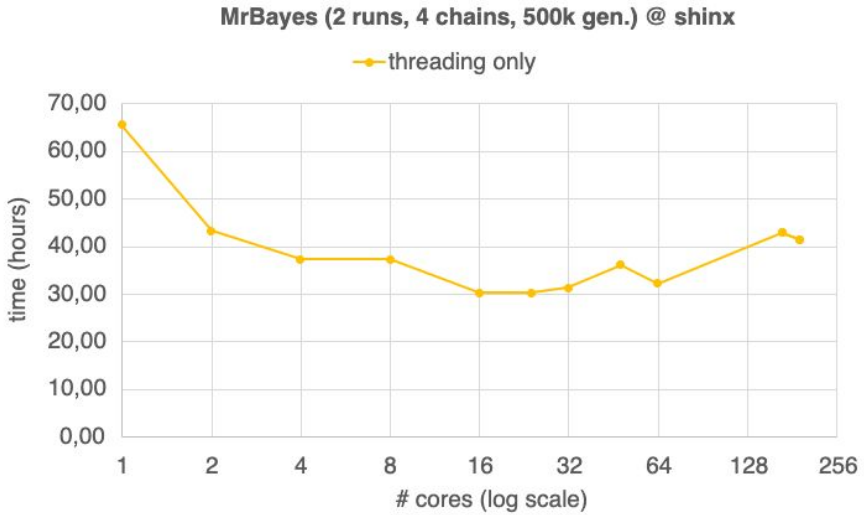
- **Multiple cores via (only) threading** => limit: # cores in a single node
- In job script: only change **`ppn=1`** to desired number of cores/threads (example: **`ppn=16`**)
- In `*.nex` file, also indicate # threads to use via `beaglethreadcount` setting (to avoid that too many threads are being used based on auto-detection of # cores)

```
begin mrbayes;
  set usebeagle=yes;
  set beagledevice=cpu;
  set beagleprecision=single;
  set beaglescaling=dynamic;
  set beaglesse=no;
  set beaglethreadcount=16;
```

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

- **Multiple cores via (only) threading** => limit: # cores in a single node
- **Some speedup**, but only 2.17x with 16 cores vs single core…
  - Only ~14% efficiency with 16 cores
  - Flatlines (or even gets worse) when trying to use more cores/threads



Lara

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

- **Multiple cores via MPI** => limit: # runs x # chains == # cores (one MPI rank per core)
- In `*.nex` file: use `beaglethreadcount=1`
- In job script: **use mympirun** tool, **change `ppn=N`** with N = 1, 2, 4, 8

```
#!/bin/bash

#PBS -l nodes=1:ppn=8
#PBS -l walltime=72:00:00

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load MrBayes/3.2.7-gompi-2023a
module load vsc-mympirun

cd $PBS_O_WORKDIR

mympirun mb mbtest.nex
```

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

- **Multiple cores via MPI** => limit: # runs x # chains == # cores (one MPI rank per core)
- **Pretty good scaling!**
  - With 8 cores (8 MPI ranks, each with 1 thread): 4.8x speedup (~60% efficiency)
  - **Still takes ~13.5h…**

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

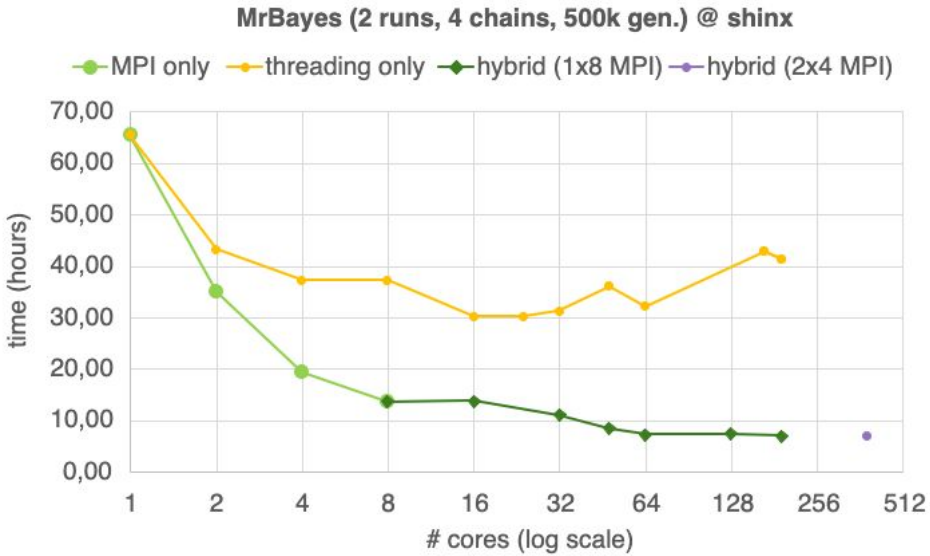
- **Multiple cores via MPI + threading** (a.k.a. “hybrid”)
  - Job script:
    - Use **ppn=C** with C number of cores (per node) to use
    - Use **`--hybrid N`** to run N MPI ranks per node (instead of 1 per core)
  - In `*.nex` file: use `beaglethreadcount=T` with T=2,4,8,...

```
#!/bin/bash

#PBS -l nodes=1:ppn=32
#PBS -l walltime=72:00:00

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load MrBayes/3.2.7-gompi-2023a
module load vsc-mympirun

cd $PBS_O_WORKDIR

mympirun --hybrid 8 mb mbtest.nex
```

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

- **Multiple cores via MPI + threading** (a.k.a. “hybrid”)
  - **More speedup beyond 8 cores!**
  - Almost 2x speedup with 8 MPI ranks with each 8 threads (64 cores): ~7h
  - No point to use multiple nodes: 2 nodes, 4x96 threads => no more speedup

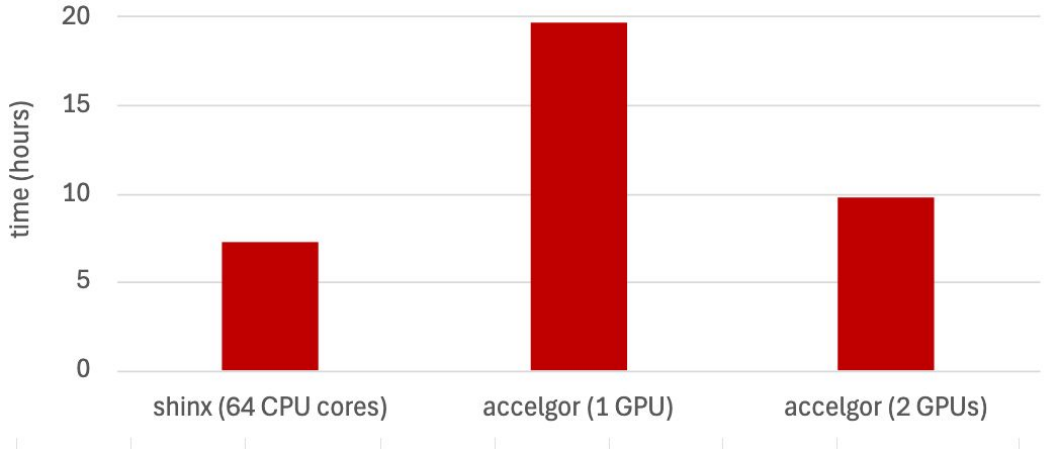


Lara

Running MrBayes jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

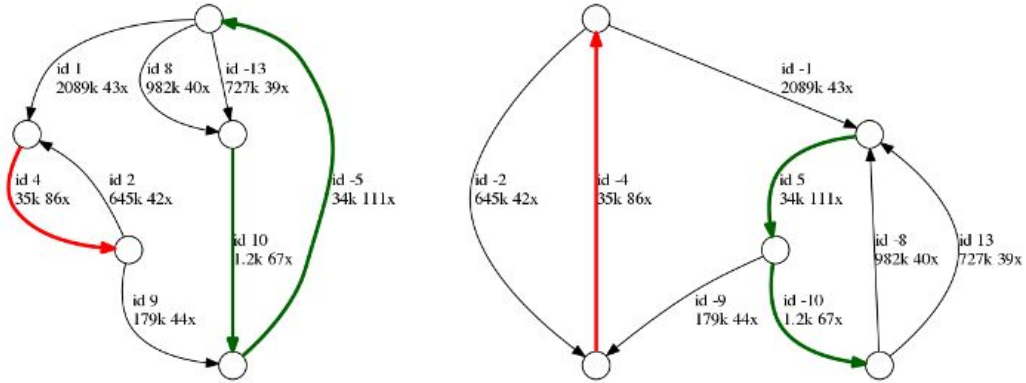
- **Can we get more speedup using GPUs? No.**
  - 1 GPU in accelgor: ~20h, 2 GPUs: ~10h
  - Good scaling, but **makes no sense** compared to just 64 cores in shinx…
    - Takes **longer** on 2 GPUs vs 64 CPU cores
    - Queue wait times for GPUs are quite long



Flye: An overview https://github.com/mikolmogorov/Flye

VLAAMS SUPERCOMPUTER CENTRUM

- de novo assembler
- PacBio & ONT
- Small to larger assemblies
- Complete pipeline (but assemblies are collapsed => HapDup alternative)
- Meta Flye => meta genome modus available



Pieter

Flye: Stages

VLAAMS SUPERCOMPUTER CENTRUM

1. **Assembly (disjointigs + k-mer counting + extending)**

   Reads analysis, detect repeats & computes frequency/coverage, initial graphs

2. **Consensus computation**

   Remap reads to initial disjoints - compute contig

3. **Repeat graph**

   Builds graph - simplify paths

4. **Contigger**

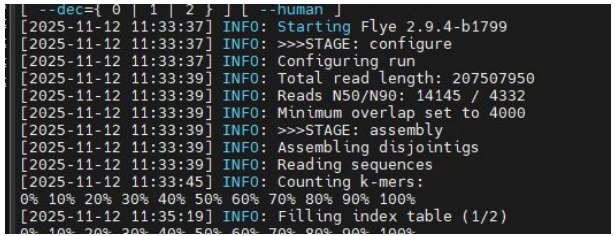
   Convert repeat graphs in contigs (scaffolding if possible)

5. **Polishing**

   Algin read back to contigs (resolve indels and mismatches

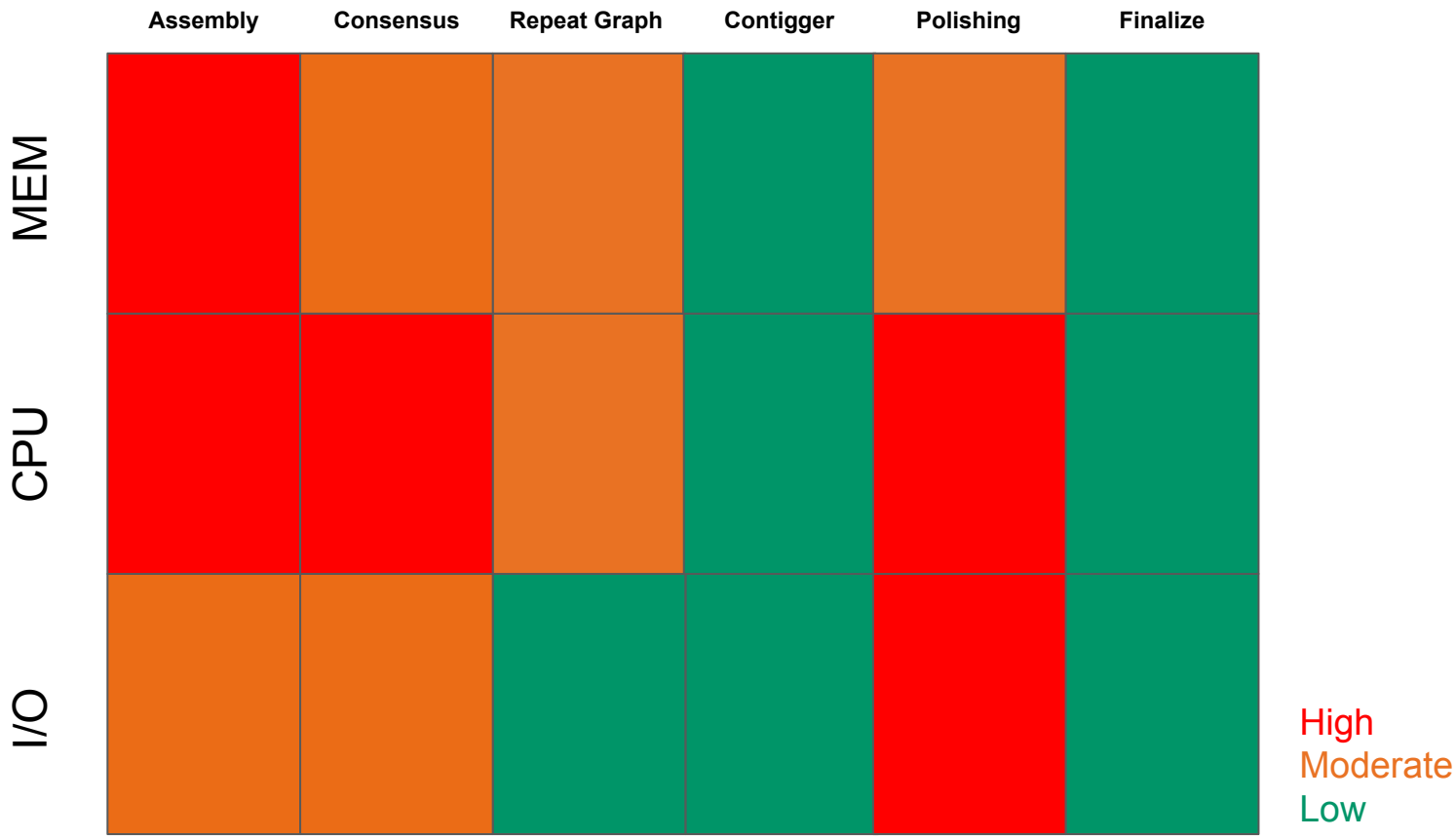
6. **Finalize**

flye.log



Flye: Computational overview

VLAAMS SUPERCOMPUTER CENTRUM



Pieter

Flye: Hardware & Software

VLAAMS SUPERCOMPUTER CENTRUM

	CPU	RAM	Arch	Model	Inter connection	Platform
LOCAL	4 cores - 8 threads	32Gib	x86_64	Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz		Conda/EESSI/Container
GROVER Tier-1 cloud	20 cores (single threaded)	360Gib	x86_64	Intel Core Processor (Broadwell) 2.2GHz		Conda/EESSI/Container
HPC (gallade)	2x64 cores	940Gib	x86_64	AMD EPYC 7773X (Milan-X @ 2.2 GHz)	HDR-100 InfiniBand	Conda/EESSI(+generic)/Container

Flye: Input data

VLAAMS SUPERCOMPUTER CENTRUM

Genome	Data	Asm.Size	NG50	CPU time	RAM
E.coli	PB 50x	4.6 Mb	4.6 Mb	2 h	2 Gb
Zymo Log	ONT meta 16 Gb	29 Mb	N/A	100 h	76 Gb

E.Coli: This dataset includes **one SMRT Cell** of data gathered with a PacBio RS II System and P6-C4 chemistry on a size selected **20kb library of E. coli K12 substrain MG1655**. Assembling this SMRT Cell results in a **single contig at >Q50 accuracy using SMRT Analysis 2.0 or higher**.

Zymo.Log: 10 species (5 Gram-positive, 3 Gram-negative, 2 yeast) ranging from 10^2 - 10^8 genomic DNA abundance (total input 5 x 10^8 cells)

Data available from:

- GridION (Zymo-GridION-LOG-BB-SN)
- PromethION (Zymo-PromethION-LOG-BB-SN)

Pieter

Flye: Methods

VLAAMS SUPERCOMPUTER CENTRUM

1. **Conda**
   a. Create env (naive)
   b. Run flye

2. **Container**
   a. Build container
   b. Run flye

3. **EESSI**
   a. Activate eessi
   b. Run flye

Pieter

Flye: Conda and Containers

VLAAMS SUPERCOMPUTER CENTRUM

- Package manager
- Multi -platform
- Easy access to programming tools
- Online repositories

LINKS

https://www.anaconda.com/docs/main
https://github.com/conda-forge/miniforge
(aarch64 support and more lightweight!)

- Container platform
- Available on HPC systems (unlike Docker)
- Many open source tools
- OS incl.
- Online repositories

LINKS

https://github.com/BioContainers/containers
https://hub.docker.com/
https://biocontainers.pro/

Pieter

Flye: Local-conda

VLAAMS SUPERCOMPUTER CENTRUM

```
# Create and activate a conda env
conda create -n flye_env python=3.12 sysstat
conda activate flye_env
conda install -c bioconda flye #will install latest version on bioconda channel
# Stage DATA
INPUTDATA="./E.coli_PacBio_40x.fasta"
PIDLOG="pidstat_flye"
OUTPUT="out_pacbio-local-conda"
# Start pidstat in the background for resource monitoring
pidstat -r -u -d -h -p ALL 5 > "${PIDLOG}.log" &
        #-r Report page faults and memory utilization
        #-u Report CPU utilization
        #-d Report I/O statistics
        #-h Display all activities horizontally on a single line, make parsing easier
        #-p Monitor all ongoing processes
        #5 Monitor every 5 seconds
# PIDSTAT_PID variable assign to later cleankill the reoccuring pidstat command
PIDSTAT_PID=$!
# run flye
flye --pacbio-raw $INPUTDATA --out-dir "$OUTPUT" --threads 4
# Kill pidstat after Flye finishes
kill $PIDSTAT_PID
# cleanup PIDSTAT logfile
awk 'NR==1 || /^# Time/ || tolower($NF) ~ /^(flye|flye-minimap2|flye-modules|flye-samtools|flye294|python3(\.10)?|starter-suid|bash|sh)$/'
"${PIDLOG}.log" > "${PIDLOG}.filtered.log"
```

```
#INSTALL CUSTOM VERSION

conda search -c bioconda flye
                .
                .
                .
conda install -c bioconda flye=2.9.4
```

CONDA

Flye: Local-container-definition file

VLAAMS SUPERCOMPUTER CENTRUM

flye-ubuntu22.04.def

```
Bootstrap: docker
From: ubuntu:22.04
        %post
            # Update and install dependencies
                    apt-get update && apt-get install -y \
                        build-essential python3 python3-pip python3-dev git zlib1g-dev libbz2-dev liblzma-dev libncurses5-dev \
                        Libncursesw5-dev libcurl4-openssl-dev libssl-dev wget cmake make

                    apt install -y squashfuse fuse-overlayfs fuse2fs gocryptfs
            # Clone Flye repository
                    cd /opt
                    git clone https://github.com/mikolmogorov/Flye.git
                    cd Flye
            # Build Flye
                    make
            # Optional: install Flye system-wide
                    ln -s /opt/Flye/bin/flye /usr/local/bin/flye

        %environment
                    export PATH=/opt/Flye/bin:$PATH
        %labels
            Author Pieter Asselman
            Version v1.0
        %runscript
            echo "Flye is ready to use. Try: flye --help"
```

apt-get install : traditional Debian/ubuntu
apt install: newer frontend, nice progress bars

Will install latest version!

APPTAINER

Flye: Local-container-definition file

flye294-ubuntu22.04.def

```
Bootstrap: docker
From: ubuntu:22.04

%post
    # Update and install dependencies
    apt-get update && apt-get install -y \
        Build-essential python3 python3-pip python3-dev git zlib1g-dev libbz2-dev liblzma-dev
        Libncursesw5-dev libcurl4-openssl-dev libssl-dev wget cmake pip sysstat make

    apt install -y squashfuse fuse-overlayfs fuse2fs gocryptfs
    pip install "git+https://github.com/mikolmogorov/Flye@2.9.4"

%labels
    Author Pieter Asselman
    Version v1.0

%runscript
    echo "Flye is ready to use. Try: flye --help"
```

APPTAINER

Pieter

Flye: Local-container - Build&Run

VLAAMS SUPERCOMPUTER CENTRUM

```
# BUILD A CONTAINER FROM DEFINITION FILE
apptainer build containername.sif containername.def

# RUN flye from A CONTAINER FROM SIF FILE
apptainer exec ~/containers/flye-ununtu22.04.sif \
   flye --pacbio-raw E.coli_PacBio_40x.fasta --out-dir flye-container --threads 4

# RRUN flye from within the container (or add applications in the container)
apptainer shell --writable flye-ubuntu22.04.sif
        #--writable by default container is read only, make file system available as read/Write
```

DON'T FORGET !

1- PUT .SIF FILE IN $VSC_SCRATCH
2- PREP ENV for container runs

https://hpcugent.github.io/vsc_user_docs/HPC/Gent/Linux/apptainer/#heappltainer

APPTAINER

Pieter

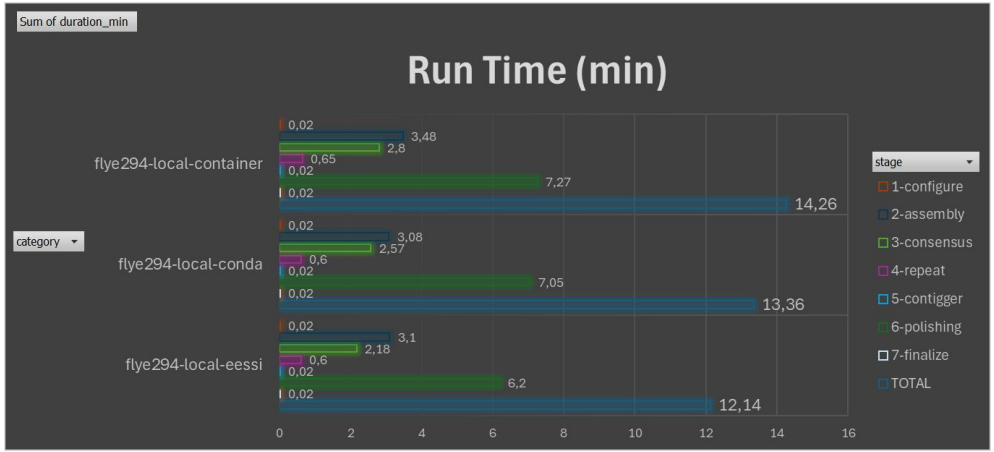
Flye: Local-EESSI

VLAAMS SUPERCOMPUTER CENTRUM

```
#!/bin/bash

# ACTIVATE EESSI
        source /cvmfs/software.eessi.io/versions/2023.06/init/bash
        ml avail Flye
        ml Flye/2.9.4-GCC-13.2.0

# Optional: Start pidstat in the background for resource monitoring
        pidstat -r \   #-r Report page faults and memory utilization
          -u \         #-u Report CPU utilization
          -d \         #-d Report I/O statistics
          -h \         #-h Display all activities horizontally on a single line, make parsing easier
          -p ALL \     #-p Monitor all ongoing processes
          5 \          #5 Monitor every 5 seconds
          > pidstat_flye-grover-container.log &

        PIDSTAT_PID=$! # PIDSTAT_PID variable assign to later cleankill the reoccuring pidstat command

#Run flye command
        flye --pacbio-raw E.coli_PacBio_40x.fasta --out-dir out_pacbio-local-eessi --threads 4

#Optional: Kill pidstat after Flye finishes
        kill $PIDSTAT_PID
```

Flye: Results - LOCAL

VLAAMS SUPERCOMPUTER CENTRUM

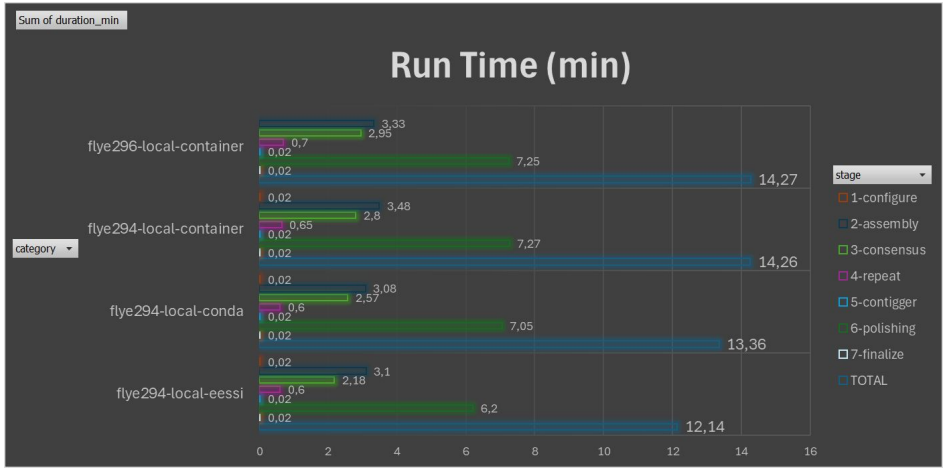


Reminder
Container: pip install
Conda: conda install
EESSI: compiled based on (micro)arch detected

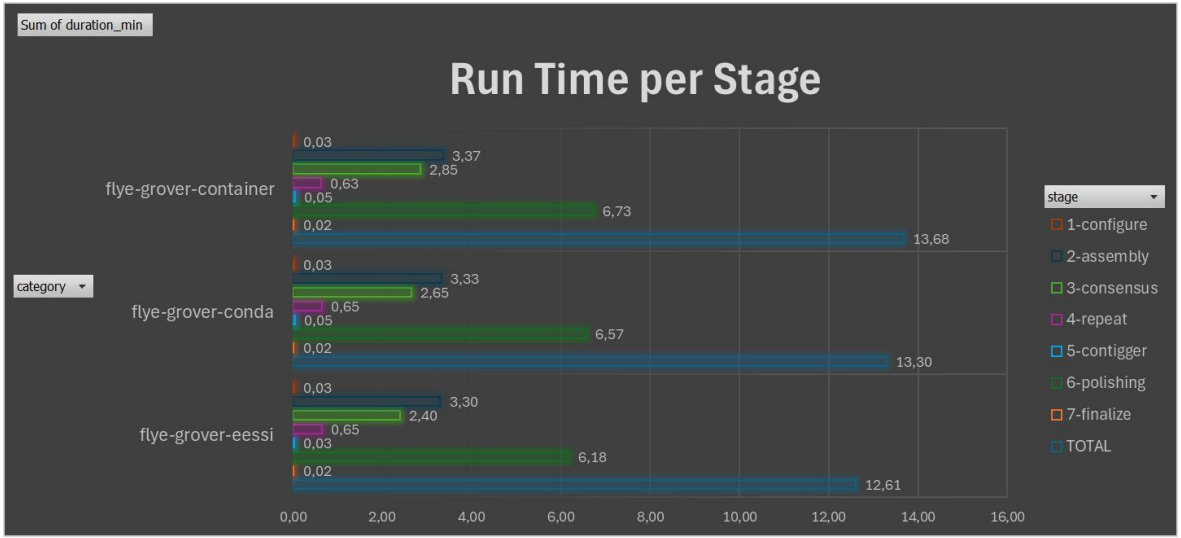
FYI
conda/container => architecture compatibility yes
=> microarchitecture optimization no

Key insights

- EESSI runs remarkably faster
- Flye 2.9.6 consensus building faster, but polishing longer



Flye: Results - GROVER



Key insights

- Same story here
- **BUT** Local EESSI(12.14m) runs faster than Grover EESSI (12.61m)???

Pieter

Flye: Results - GROVER

VLAAMS SUPERCOMPUTER CENTRUM

	CPU	RAM	Arch (CPU family)	Model (micro architecture)
LOCAL (2019)	4 cores - 8 threads	32Gib	x86_64	Intel(R) Core(TM) i7-8665U CPU **@ 1.90GHz**
GROVER Tier-1 cloud (2014)	20 cores (single threaded)	360Gib	x86_64	Intel Core Processor (Broadwell) **2.2GHz**

OLD vs NEW cores

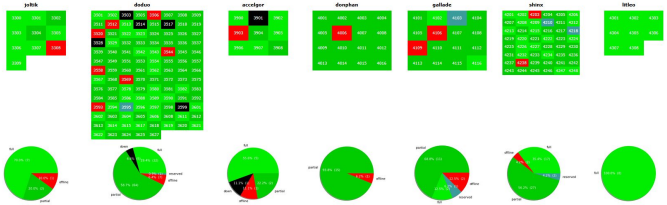
- **IPC Advantage (Instructions per cycle):** Whiskey Lake (2019) has much higher instructions-per-cycle than Broadwell (2014).
- **Turbo Boost:** i7-8665U can hit ~4.8 GHz on 1–2 cores, while Broadwell likely tops out near 3 GHz.
- **Instruction Set:** Whiskey Lake supports newer optimizations (AVX2, better branch prediction).

Flye: HPC - Can we Speed things up?

VLAAMS SUPERCOMPUTER CENTRUM

Re(search)quest Resources

We have plenty and are spoiled @UGent :-)



Pieter

Flye: HPC - Can we Speed things up?

VLAAMS SUPERCOMPUTER CENTRUM

shinx : 48 nodes - 2*96 cores - 370 GiB/node

=> Let's be modest, request half a node

BAD OR GOOD USAGE OF RESOURCES?

Name : flye-conda
User : vscxxxx
Partition : shinx
Nodes : node4232.shinx.os
Cores : 96
State : COMPLETED
Submit : 2025-11-07T11:47:07
Start : 2025-11-07T22:29:38
End : 2025-11-07T23:12:33
Reserved walltime : 10:00:0.0
Used walltime : 00:42:55.0
Used CPU time : 14:29:18.0
% User (Computation): 99.36
% System (I/O) : 0.64
Mem reserved : 185G
Max Mem used : 62.90G (node4232.shinx.os)
Max Disk Write : 92.70G (node4232.shinx.os)
Max Disk Read : 204.52G (node4232.shinx.os)
Working directory : /user/gent/433/vsc4xxxx

Let's do the math

Walltime:42min/60=0.7
CPU-time: 14+(29/60)= 14.29

#Cores Used= CPU-time/Walltime
=> 14,29/0.7 ~20 cores

Requested too many cores!

Pieter

Flye: HPC - Can we Speed things up?

VLAAMS SUPERCOMPUTER CENTRUM

shinx : 48 nodes - 2*96 cores - 370 GiB/node

=> be wise, adapt, request 8 cores and 80GiB RAM

BAD OR GOOD USAGE OF RESOURCES?

```
Name                : flye-conda
User                : vsc4xxxx
Partition           : shinx
Nodes               : node4103.shinx.os
Cores               : 8
State               : COMPLETED
Submit              : 2025-11-14T15:06:03
Start               : 2025-11-14T15:06:03
End                 : 2025-11-14T16:50:18
Reserved walltime   : 20:00:0.0
Used walltime       : 01:44:15.0
Used CPU time       : 11:32:40.0
% User (Computation): 99.26
% System (I/O)      :  0.74
Mem reserved        : 80G
Max Mem used        : 62.36G (node4103.shinx.os)
Max Disk Write      : 103.72G (node4103.shinx.os)
Max Disk Read       : 209.50G (node4103.shinx.os)
Working directory   : /user/gent/433/vsc4xxxx
```

Let's do the math

Walltime: 1+(44/60)=1.7h
CPU time: 11+(32/60)=11.5
CPU-time/Walltime => 11.5/1.7 ~6.76 cores

Requested a good amount of cores!
But walltime is double…

Flye: HPC - Can we Speed things up?

VLAAMS SUPERCOMPUTER CENTRUM

Re(search)quest Resources

- Run on 8 cores
- 16
- 32
- 48
- …
- Add MEM
- Look for I/O
- #cores=CPUtime/Walltime

Pieter

Flye: HPC - Can we Speed things up?

VLAAMS SUPERCOMPUTER CENTRUM

Genome	Data	Asm.Size	NG50	CPU time	RAM
E.coli	PB 50x	4.6 Mb	4.6 Mb	2 h	2 Gb
Zymo Log	ONT meta 16 Gb	29 Mb	N/A	100 h	76 Gb

GitHub: https://github.com/mikolmogorov/Flye

E.Coli: This dataset includes **one SMRT Cell** of data gathered with a PacBio RS II System and P6-C4 chemistry on a size selected **20kb library of E. coli K12 substrain MG1655**. Assembling this SMRT Cell results in a **single contig at >Q50 accuracy using SMRT Analysis 2.0 or higher**.

Zymo.Log: 10 species (5 Gram-positive, 3 Gram-negative, 2 yeast) ranging from 10^2 - 10^8 genomic DNA abundance (total input 5 x 10^8 cells)

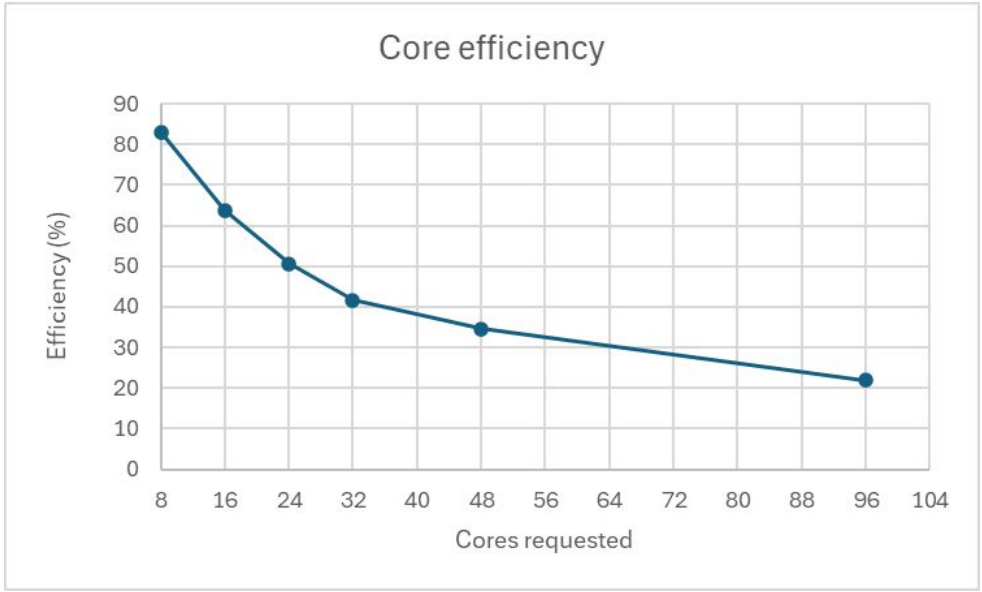
Data available from:

- GridION (Zymo-GridION-LOG-BB-SN)
- PromethION (Zymo-PromethION-LOG-BB-SN)

Pieter

Flye: HPC - Scaling # cores

VLAAMS SUPERCOMPUTER CENTRUM



Trade off…

A few minutes runtime gain

…

A whole lot more queue time

Calculations:

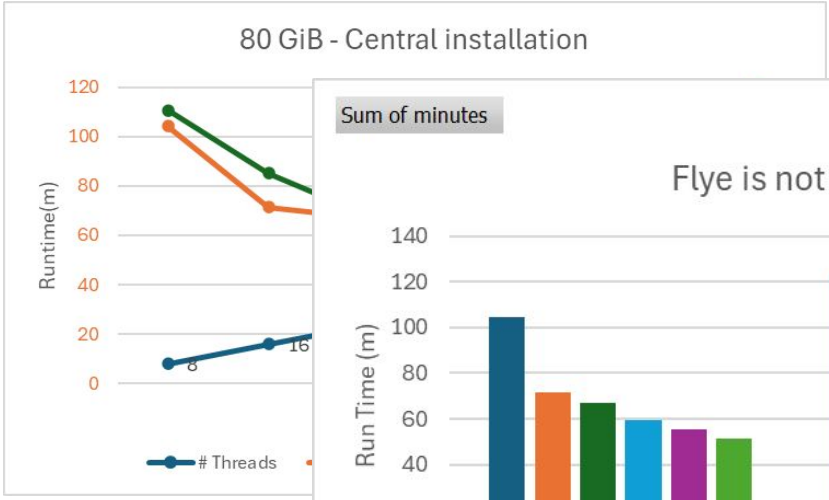
Used cores **=> CPU-time / Walltime**

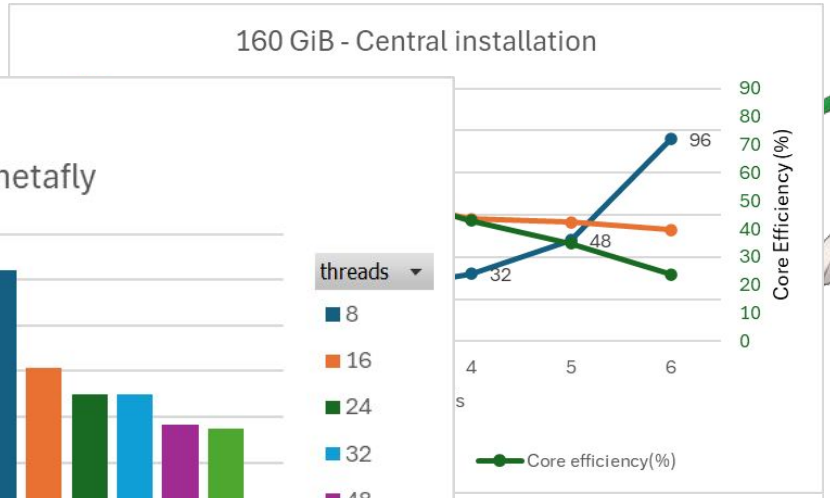
Core efficiency **=>(Used cores / Requested cores) *100**

Pieter

Flye: HPC - Scaling - Memory

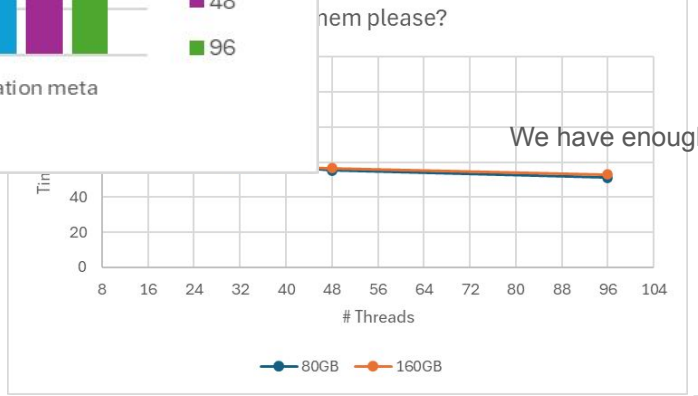
VLAAMS SUPERCOMPUTER CENTRUM





Key insights

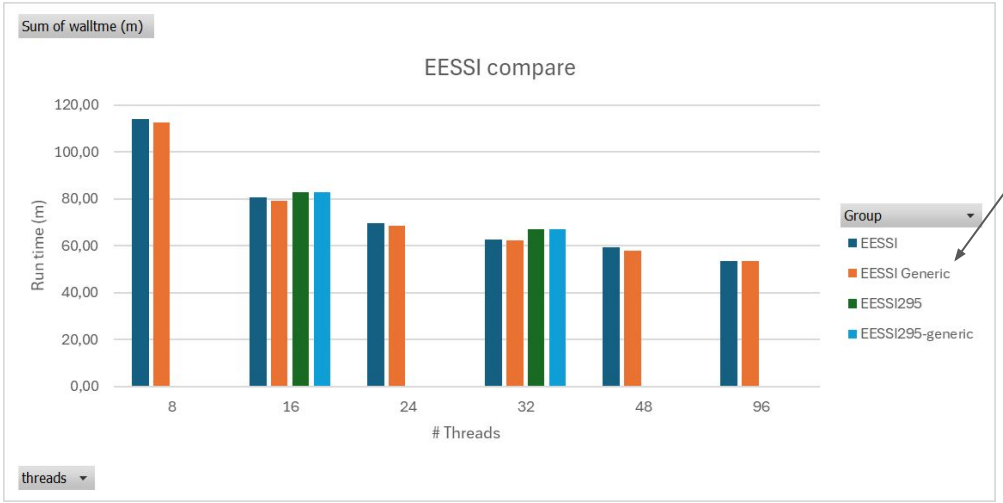
- Best balance
  - 16: Runti
  - 24: 66 mi
- >32 threads small gains
- 96 threads fastest runtime but 22% core efficiency
- Memory used tops off at ~62GB (assembly size ~24Mb)



Pieter

Flye: HPC - Scaling - EESSI

VLAAMS SUPERCOMPUTER CENTRUM



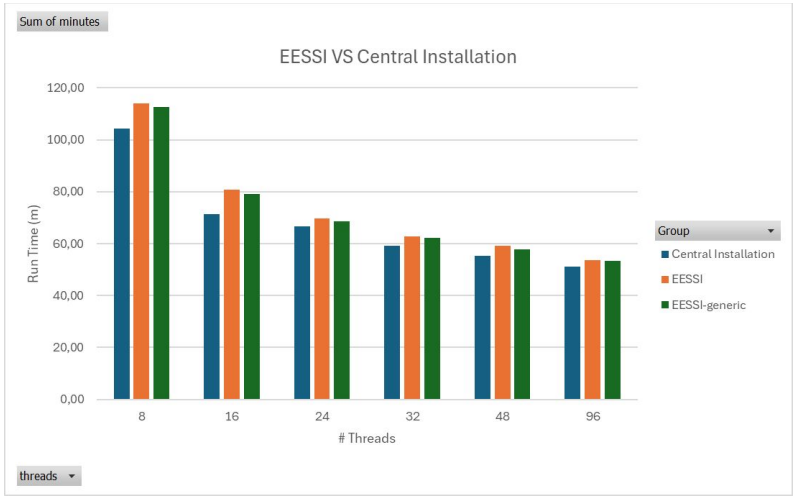
Optimization level

EESSI (full option): arch + micro arch

Generic - family level only

entries Search:

x86_64						
	amd			intel		
generic	zen2	zen3	zen4	cascadelake	haswell	icelake
x	x	x	x	x	x	x



Key insights

- EESSI optimal VS generic => same result
- Central installation runs faster than EESSI on HPC
  => Different causes … Kenneth Help!

Pieter

Flye: Conclusions

VLAAMS SUPERCOMPUTER CENTRUM

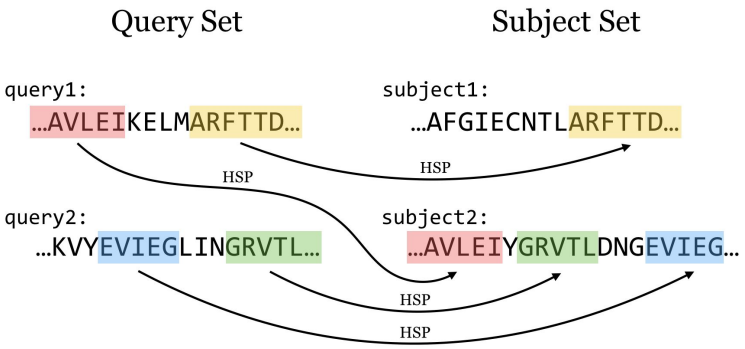
Take home

- How did you get your software on your computer?
- Don’t like queueing too much? Optimize your resource usage first.
- Do you like interactive sessions?
  => Go play with donphan (interactive/DEBUG)!
- Central software installations are heavily optimized to run on the clusters.
  => Yes, it takes time to get them installed centrally…
- Use EESSI locally, it runs faster!
- If you still see limitations, ask HPC-team if there is a solution.
  => We can't all be computer scientists.

BLAST+: An overview

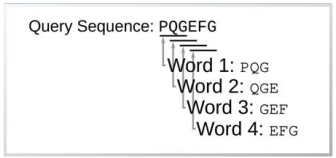
https://github.com/ncbi/blast_plus_docs

VLAAMS SUPERCOMPUTER CENTRUM



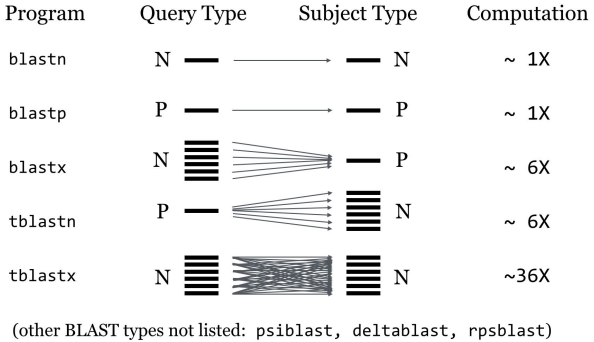
BLAST Workflow (condensed)

- Clean the query (remove low-complexity/repeats).
- Break the query into k-letter words.
- Score these words and keep only high-scoring nes.
- Build a fast lookup index for the selected words.
- Search the database for exact word matches.
- Extend matches into **high-scoring segment pairs** (**HSPs**).
- Score HSPs and evaluate significance (E-values).
- Merge HSPs into longer alignments.
- Report significant hits.



What does it do?

Basic Local Alignment Search Tool (BLAST)
finds regions of local similarity between sequences.



(other BLAST types not listed: `psiblast`, `deltablast`, `rpsblast`)

Query		**Subject**
Blastn - Nucleotide	=>	Nucleotide
Blastp - Protein	=>	Protein
Blastx - Nucleotide (translated)	=>	protein
tBlastn - Protein	=>	Nucleotide (translated
tBlastx - Nuc (transl)	=>	Prot (tranls)

Kenneth

BLAST+: Workload info

VLAAMS SUPERCOMPUTER CENTRUM

- `blastn` (nucleotide - nucleotide)
- Supports using multiple cores via multi-threading (OpenMP)
- Large input file (`*.fasta`): ~115,000 sequences
  - Easy to reduce to smaller input for testing/benchmarking: first N lines
- **Very** I/O-intensive: search in large databases
  - `core_cnt` NCBI database: ~260GB
  - `nt` NCBI database: ~700GB

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Starting point:

- **Single-core, database on `$VSC_DATA` filesystem**
- Reduced input file (5,000 sequences), ~4% of full input of 115k sequences
- Smaller database (core_nt), ~260GB

```
#!/bin/bash
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=1

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load BLAST+/2.14.1-gompi-2023a
export BLASTDB="$VSC_DATA_VO/blast/core_nt"

cd $PBS_O_WORKDIR
blastn -task megablast -query small-5k-seq.fasta -db $BLASTDB/core_nt
-outfmt '6 qseqid staxids bitscore std sscinames sskingdoms stitle' -evalue
1e-10 -max_hsps 1 -max_target_seqs 5
```

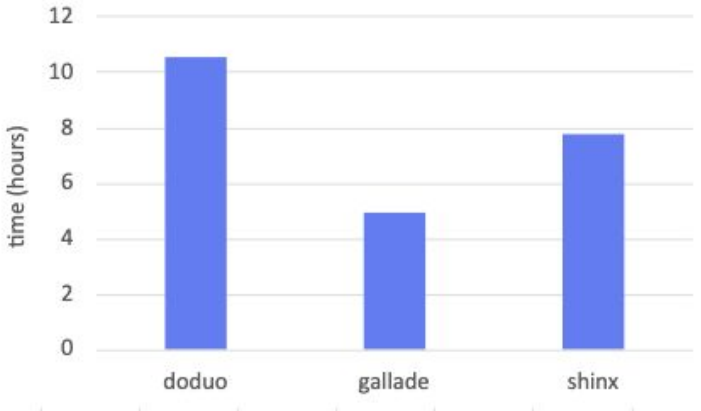
Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Starting point:

- **Single-core, database on `$VSC_DATA` filesystem**
- Reduced input file (5,000 sequences), ~4% of full input of 115k sequences
- Smaller database (core_nt), ~260GB

Results: 5-11 hours of required walltime



Smells like trouble for full input file + `nt` database, given 72h walltime limit…

Naive extrapolation: x20 x3 => over 300h of walltime!

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **multiple cores** via threading (OpenMP)?

- `core_cnt` @ `$VSC_DATA` + reduced input file (5k sequences)
- **First attempt: close to 2x speedup with 2 threads, then no further speedup?!**

```
#!/bin/bash
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=8

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load BLAST+/2.14.1-gompi-2023a
export BLASTDB="$VSC_DATA_VO/blast/core_nt"

cd $PBS_O_WORKDIR
blastn -task megablast -query small-5k-seq.fasta -db $BLASTDB/core_nt
-num_threads $PBS_NUM_PPN -outfmt '6 qseqid staxids bitscore std sscinames
sskingdoms stitle' -evalue 1e-10 -max_hsps 1 -max_target_seqs 5
```

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **multiple cores** via threading (OpenMP)?

- `core_cnt` @ `$VSC_DATA` + reduced input file (5k sequences)
- **Live job inspection: blastn process with all treads stuck on a single core!**

```
$ qstat -ant 12345          # determine workernode on which job is running
$ ssh node1234              # SSH into that workernode
$ ps xfu                    # show list of running processes - look for blastn
USER         PID %CPU %MEM    VSZ   RSS TTY STAT START   TIME COMMAND
...
vsc40023  345678 99.9 4.2   123456 56789 ?  R    12:34  34:56  \_ blastn ...

$ taskset -c p 345678       # check on which cores the blastn process is running
pid 345678 current affinity list: 0
```

Max. 100% CPU
=> only using a single core

This should be a list of core IDs
(not one single one…)

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **multiple cores** via threading (OpenMP)?

- `core_cnt` @ `$VSC_DATA` + reduced input file (5k sequences)
- Fixed job script (no more pinning to a single core)

```
#!/bin/bash
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=8

unset OMP_PROC_BIND   # avoid being stuck to a single core...

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load BLAST+/2.14.1-gompi-2023a
export BLASTDB="$VSC_DATA_VO/blast/core_nt"

cd $PBS_O_WORKDIR
blastn -task megablast -query small-5k-seq.fasta -db -db $BLASTDB/core_nt
-num_threads $PBS_NUM_PPN -outfmt '6 qseqid staxids bitscore std sscinames
sskingdoms stitle' -evalue 1e-10 -max_hsps 1 -max_target_seqs 5
```

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **multiple cores** via threading (OpenMP)?

- `core_cnt` @ `$VSC_DATA` + reduced input file (5k sequences)
- Fixed job script (no more pinning to a single core)

```
$ qstat -ant 12346          # determine workernode on which job is running
$ ssh node1234              # SSH into that workernode
$ ps xfu                    # show list of running processes - look for blastn
USER         PID %CPU %MEM    VSZ   RSS TTY STAT START   TIME COMMAND
...
vsc40023  456789 798  32.5  123456 56789 ?   R    12:34  734:56  \_ blastn ...

$ taskset -c p 456789       # check on which cores the blastn process is running
pid 345678 current affinity list: 0-7
```

Close to 800% CPU
=> using 8 cores 👍

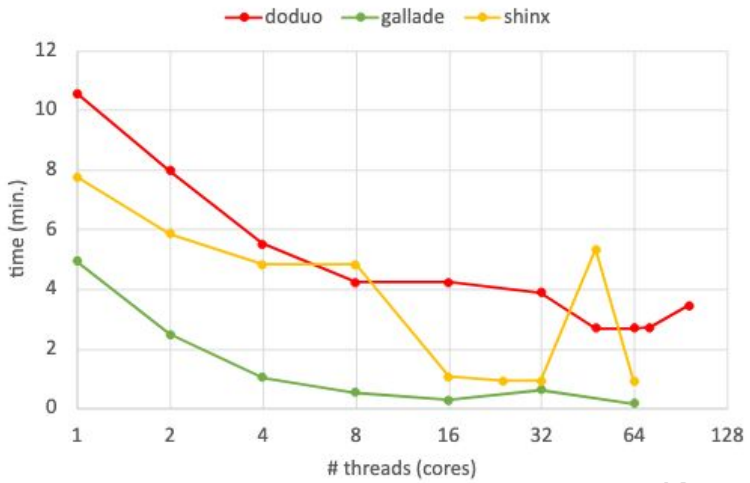
List of core IDs 👍

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **multiple cores** via threading (OpenMP)?

- `core_cnt` @ `$VSC_DATA` + reduced input file (5k sequences)
- Fixed job script (no more pinning to a single core)
- **Significant speedup** with multiple cores/threads (but **flatlines** at 8-16 cores)
- gallade cluster is *remarkably* better,
  well under 1h when using 8+ cores!
- Some quirky results…
  Using shared resources can lead to chaos



Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **different shared filesystem**?

- `core_nt` @ `$VSC_DATA` , `$VSC_SCRATCH`, `$VSC_SCRATCH_ARCANINE`
- Reduced input file (5k sequences)

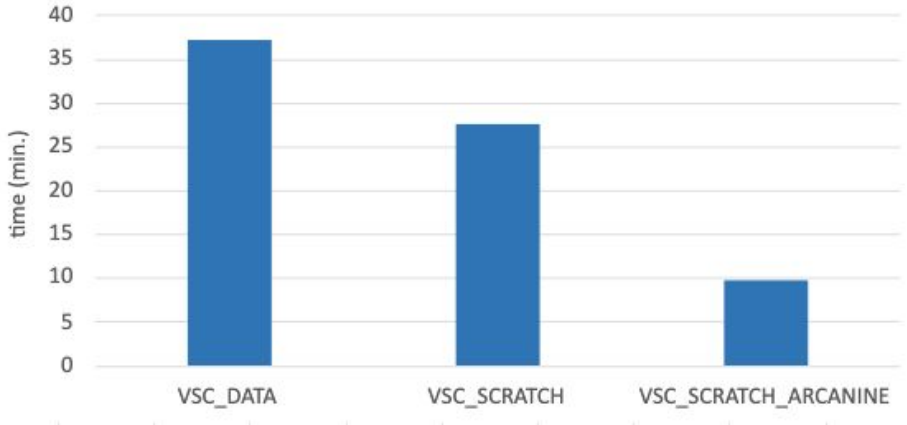
```
#!/bin/bash
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=32

unset OMP_PROC_BIND   # avoid being stuck to a single core...

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load BLAST+/2.14.1-gompi-2023a
export BLASTDB="$VSC_SCRATCH_ARCANINE_VO/blast/core_nt"

cd $PBS_O_WORKDIR
blastn -task megablast -query small-5k-seq.fasta -db -db $BLASTDB/core_nt
-num_threads $PBS_NUM_PPN -outfmt '6 qseqid staxids bitscore std sscinames
sskingdoms stitle' -evalue 1e-10 -max_hsps 1 -max_target_seqs 5
```

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we make it faster using **different shared filesystem**?

- `core_nt` @ `$VSC_DATA` , `$VSC_SCRATCH`, `$VSC_SCRATCH_ARCANINE`
- Reduced input file (5k sequences)
- gallade cluster, 32 cores, required time in *minutes*
- ~4x faster with `$VSC_SCRATCH_ARCANINE` vs `$VSC_DATA`
- Can vary a lot due to load on shared filesystem



Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we run full input (115k seq.) using large `nt` database (~700GB) in < 72h?

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we run full input (115k seq.) using large `nt` database (~700GB) in < 72h?

Database @ **`$VSC_SCRATCH_ARCANINE`**, multi-threaded (quarter of cores per node)

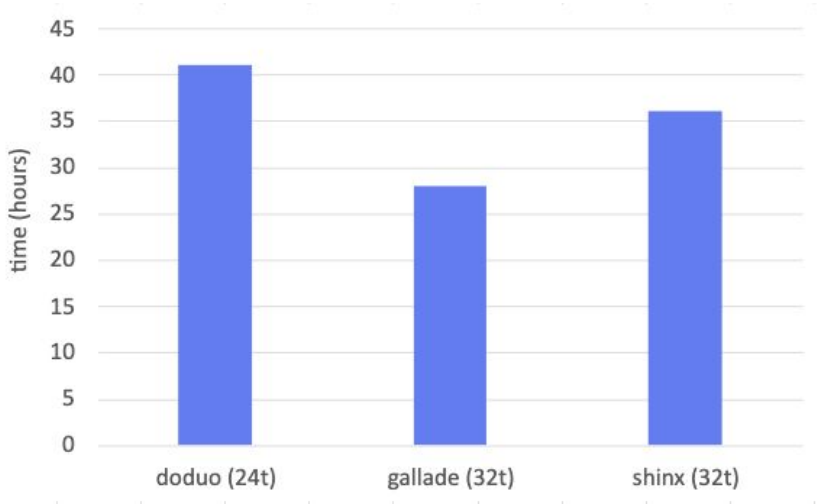
```
#!/bin/bash
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=quarter

unset OMP_PROC_BIND   # avoid being stuck to a single core...

source /cvmfs/software.eessi.io/versions/2023.06/init/bash
module load BLAST+/2.14.1-gompi-2023a
export BLASTDB="$VSC_SCRATCH_ARCANINE_VO/blast/nt"

cd $PBS_O_WORKDIR
blastn -task megablast -query fastas/full-115k.fasta -db $BLASTDB/nt
-num_threads $PBS_NUM_PPN -outfmt '6 qseqid staxids bitscore std sscinames
sskingdoms stitle' -evalue 1e-10 -max_hsps 1 -max_target_seqs 5
```

Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we run full input (115k seq.) using large `nt` database (~700GB) in < 72h? Yes!

- Database on `$VSC_SCRATCH_ARCANINE` filesystem
- **Using reasonable # cores on each CPU-only cluster**
  - doduo (24 cores): ~41h
  - **gallade (32 cores): ~28h**
  - shinx (32 cores): ~36h



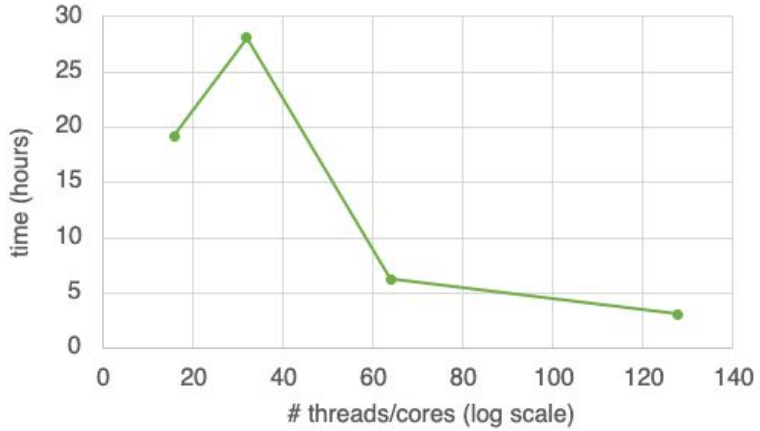
Running BLAST+ jobs @ HPC-UGent

VLAAMS SUPERCOMPUTER CENTRUM

Can we run full input (115k seq.) using large `nt` database (~700GB) in < 72h? Yes!

- Database on `$VSC_SCRATCH_ARCANINE` filesystem
- **Scaling with # cores on a single gallade node**
- **Done in ~3h15min with 128 cores (and ~900GB of memory)**

  (and if you're lucky that load on $VSC_SCRATCH_ARCANINE is sufficiently low)



Time to get your hands dirty!

VLAAMS SUPERCOMPUTER CENTRUM

- Reservations for this hackathon:
  - On doduo cluster: 4 nodes (each w/ 96 cores and ~250GB memory)
  - On gallade cluster: 1 node (128 cores and ~940GB memory)
  - On shinx cluster: 4 nodes (each w/ 192 cores and 370GB memory)
  - On accelgor cluster: 1 node (4x A100 GPUs)
- Submit into reservation with:

```
module swap cluster/example  # <== change this to name of right cluster
qsub --reservation=hackathon job-script.sh
```

- Or start interactive session in reservation via web portal
- Reservations will be active until end of today (7pm)