



**GHENT  
UNIVERSITY**



# Introduction to HPC-UGent

June 17th 2019 - VOLSTOK - Blender

[hpc@ugent.be](mailto:hpc@ugent.be)

<https://www.ugent.be/hpc>



**Vlaanderen**  
is computing

# About this training – purpose

- Inform you of HPC-UGent services and infrastructure
- Learn what the benefit can be for your research
- Get you started on the central HPC infrastructure at UGent
  - Successfully connect to the HPC infrastructure
  - Successfully launch your first job
  - Figure out how to leverage it for your research
- Answer any questions you may have

# About this training – HPC tutorial

- An HPC tutorial is available, applicable for all VSC infrastructure
- Download it here: <https://www.ugent.be/hpc/en/support/documentation.htm>
- This is work in progress. If you find errors, do let us know.
- We will specifically use information from these chapters:
  - 1/ Introduction to HPC
  - 2/ Getting an HPC account
  - 3/ Connecting to the HPC
  - 4/ Running batch jobs
  - 6/ Running jobs with input/output data
  - 8/ Fine-tuning job specifications

# What is High Performance Computing?

“High Performance Computing” (HPC) is computing on a “supercomputer”, a system at the frontline of contemporary processing capacity – particularly in terms of size, supported degree of parallelism, network interconnect and (total) available memory & disk space.

A computer cluster consists of a set of loosely or tightly connected computers that work together so that in many respects they can be viewed as a single system.

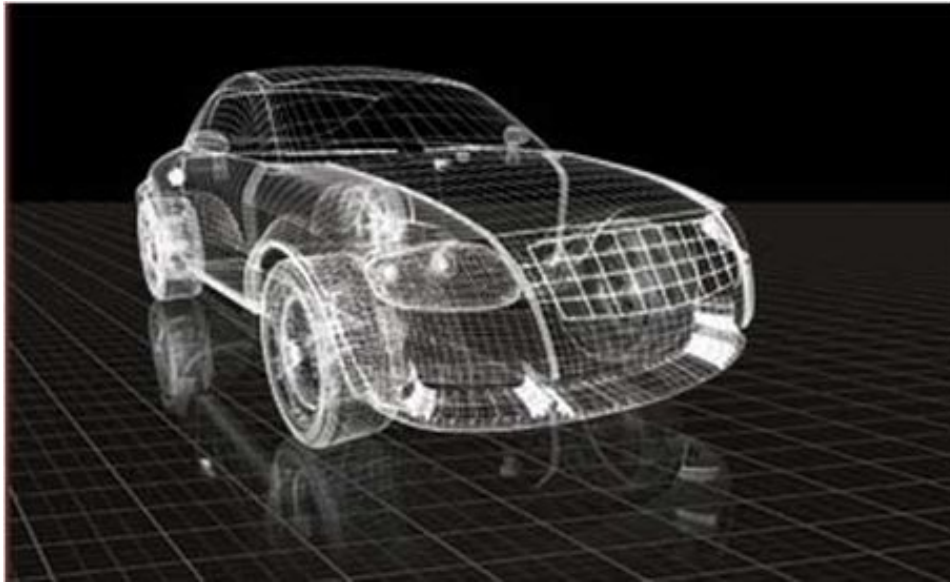
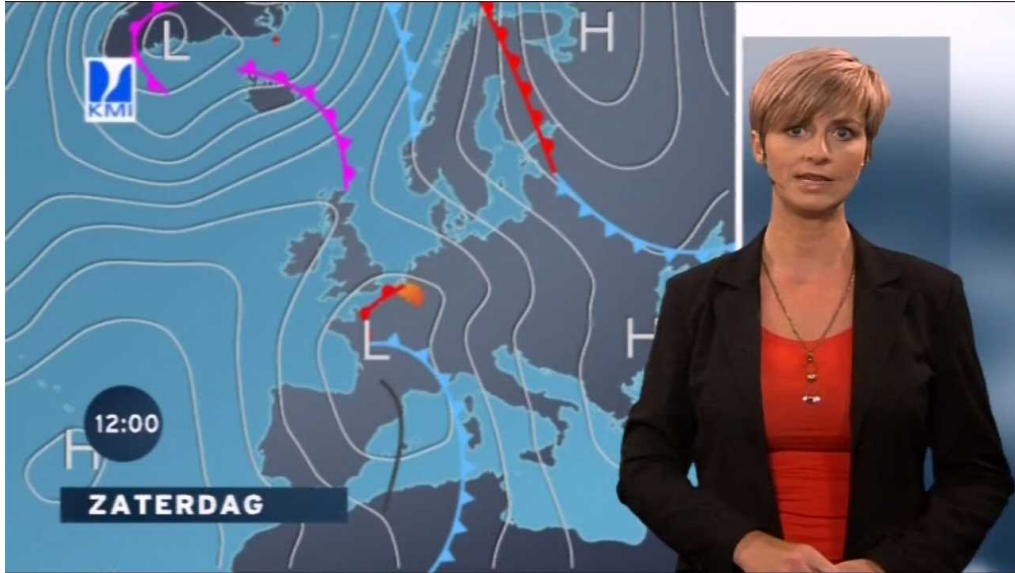
(a.k.a. “supercomputing”)

# What is High Performance Computing?

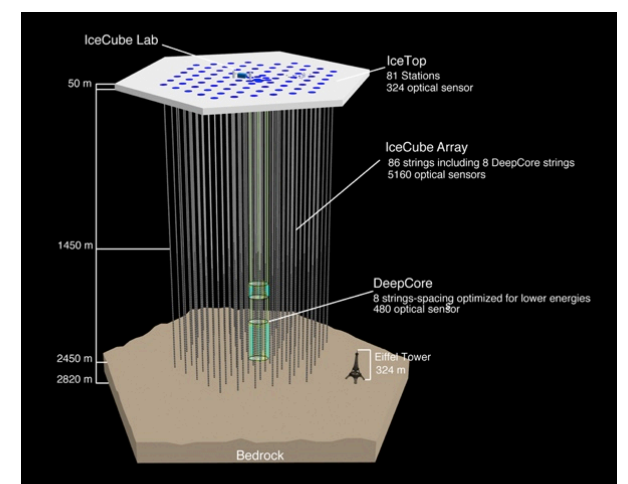
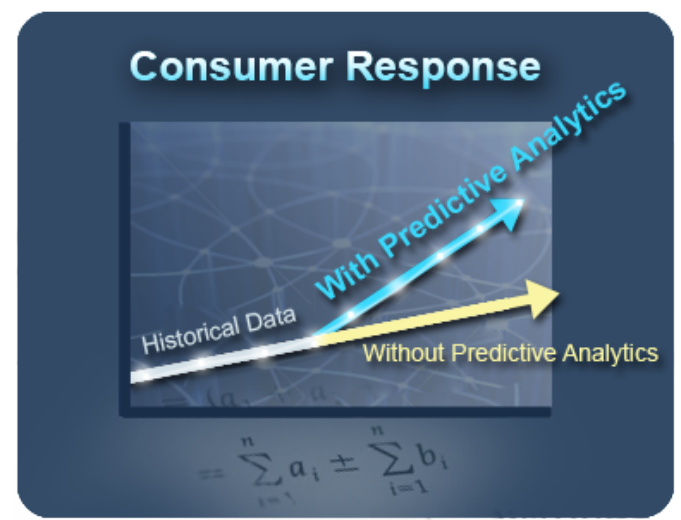
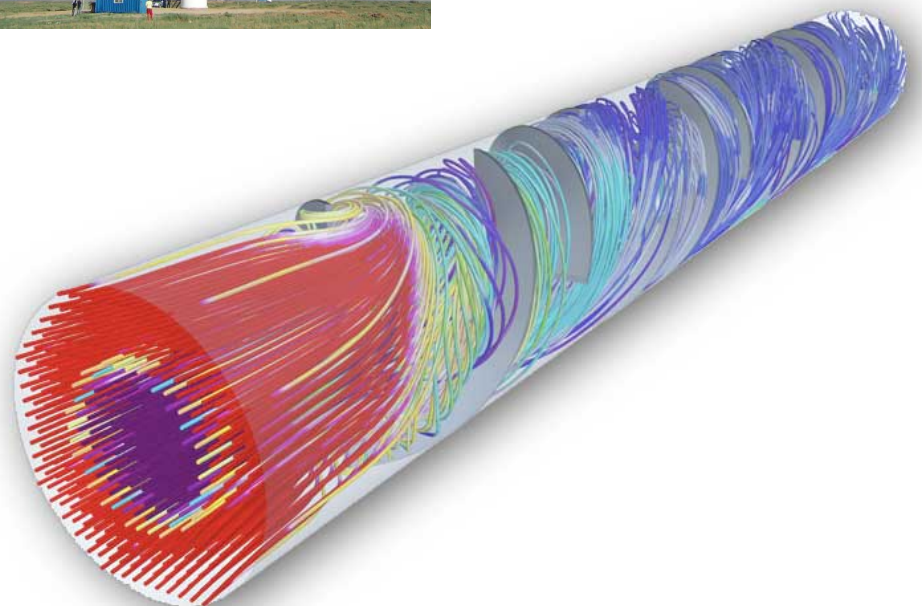
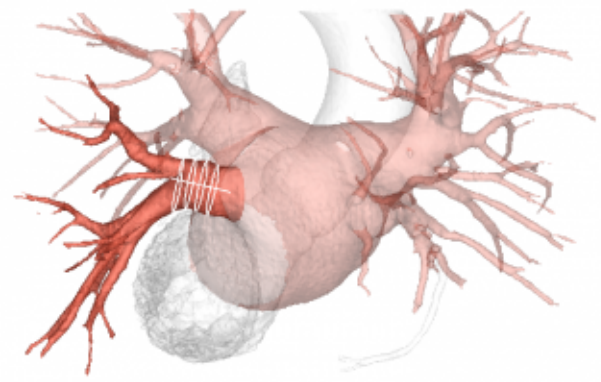
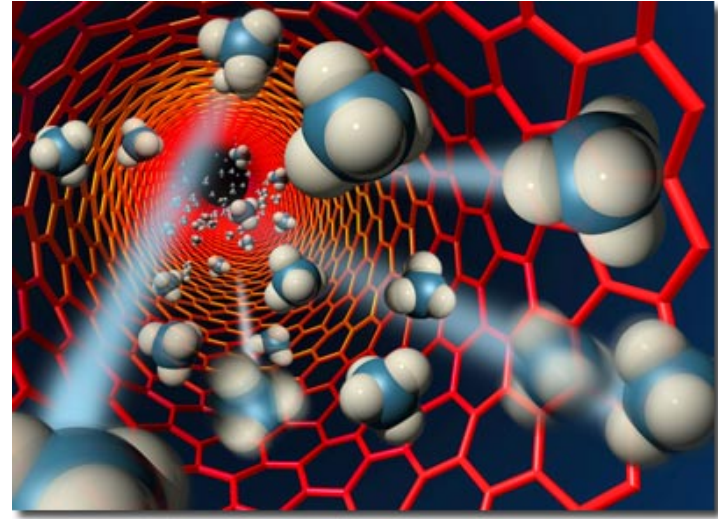
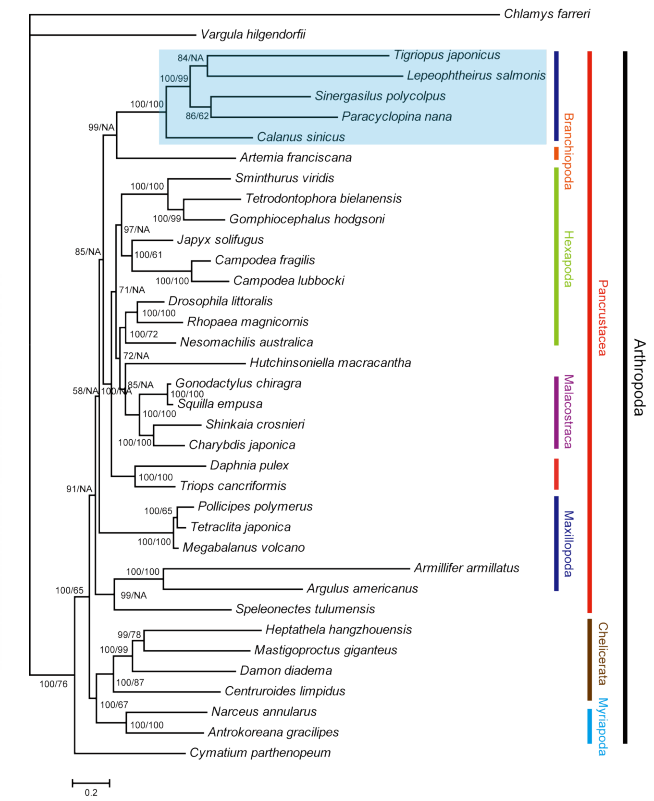
harness power of multiple interconnected cores/nodes/processing units



# Everyday applications of supercomputing



# Scientific applications of supercomputing

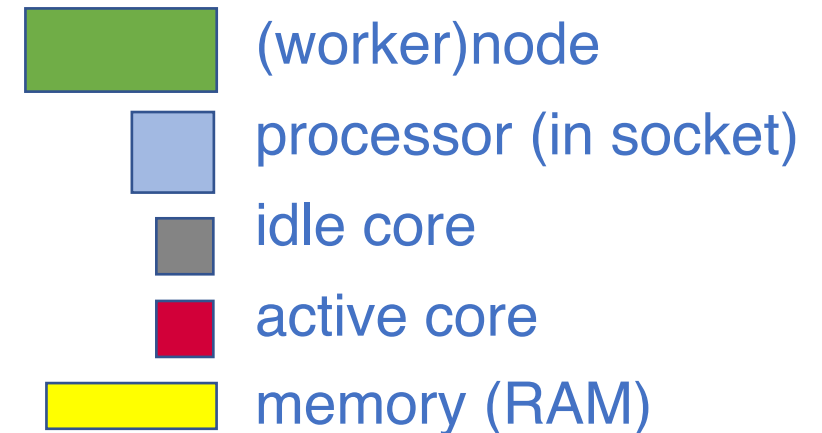
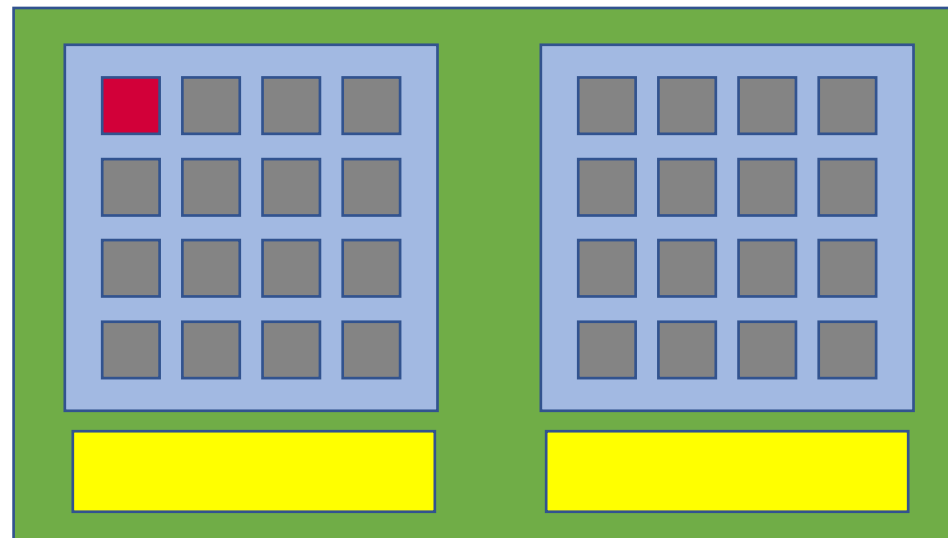


# Cores, CPUs, processors, sockets, (worker)nodes

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 CPUs or cores that are used to execute computations.

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



(not included in picture:  
local disk, network cards, ...)

# Parallel vs sequential software

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”).

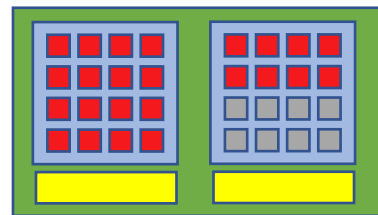
Example: OpenFOAM can easily use 160 cores at the same time to solve a CFD problem

Parallel programming paradigms:

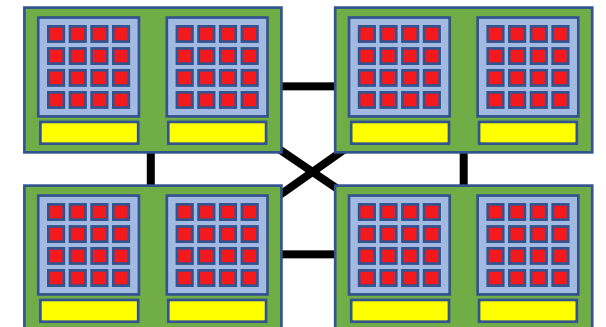
OpenMP for shared memory systems (multithreading) -> on cores of a single node

MPI for distributed memory systems (multiprocessing) -> on cores of multiple nodes

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

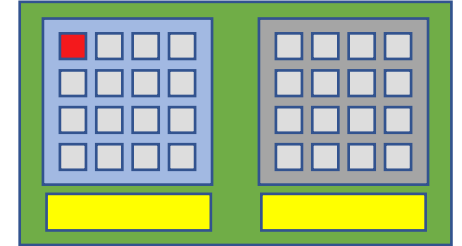


MPI software can use (all) cores in multiple nodes



# Parallel vs sequential programs

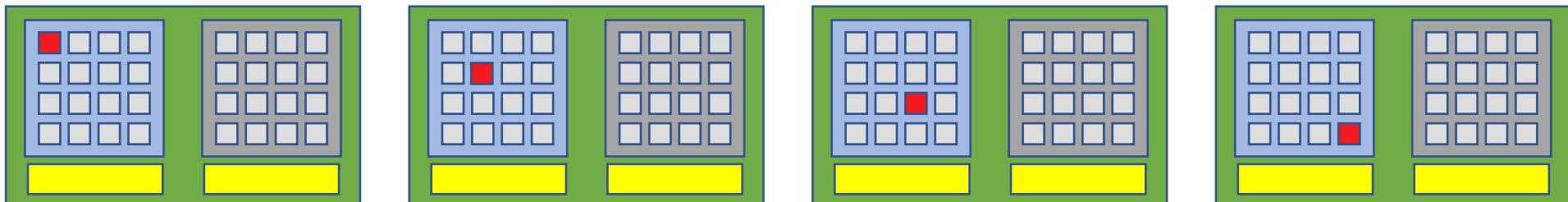
Sequential (a.k.a. serial) software does not do calculations in parallel, i.e. it only uses one single core of a single workernode.



This type of software does not run faster by just throwing cores at it...

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

e.g., you can run a Python script 100 times on 100 cores to quickly analyse 100 datasets



# HPC-UGent

[hpc@ugent.be](mailto:hpc@ugent.be)

Part of ICT Department of Ghent University

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

# HPC-UGent: staff



Stijn De Weirdt  
technical lead



Ewald Pauwels  
team lead



Kenneth Hoste  
user support & training



Wouter Depypere  
sysadmin, hardware



Andy Georges  
sysadmin, tools



Kenneth Waegeman  
sysadmin, storage



Balázs Hajgató  
sysadmin, tools

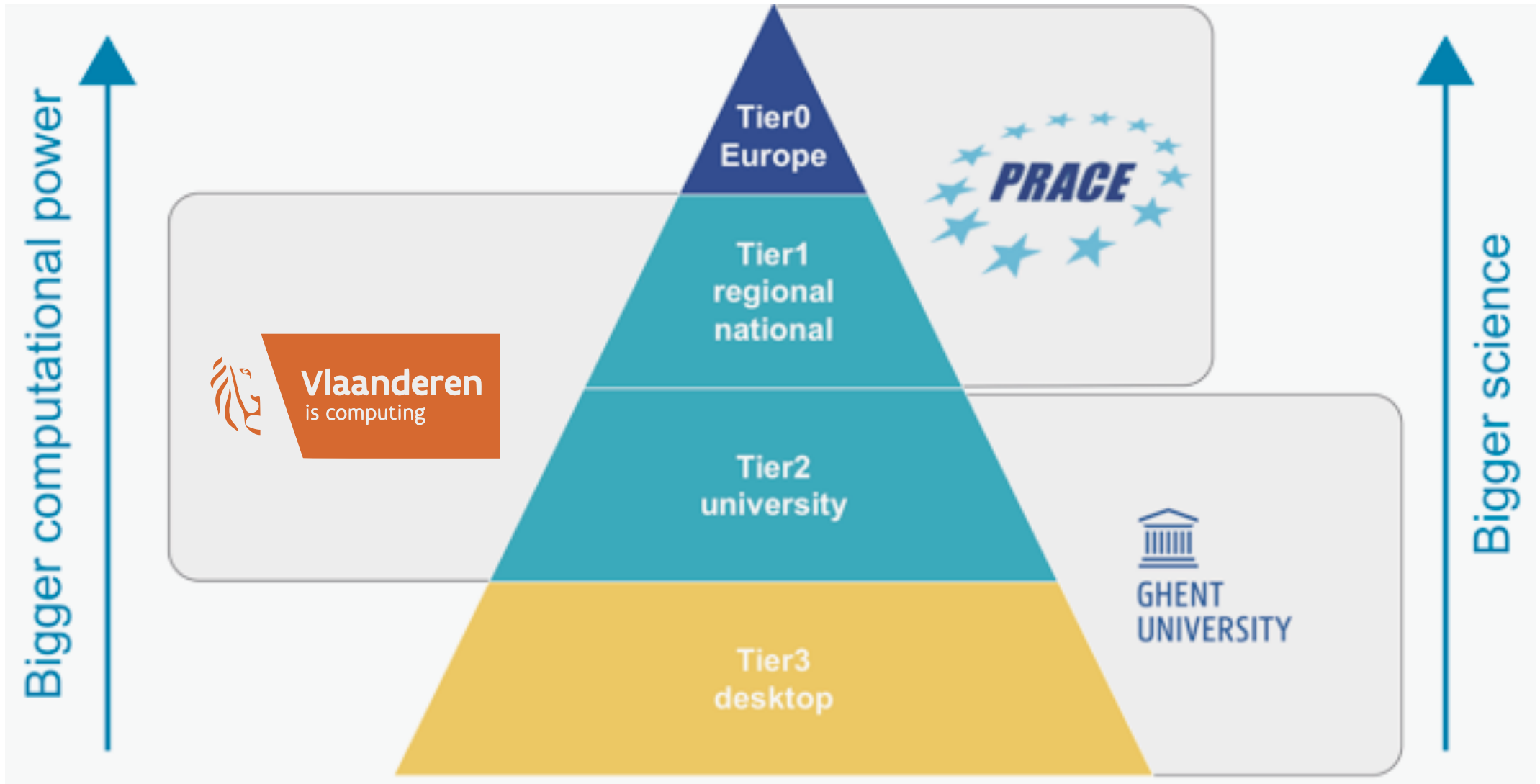


Álvaro Simón García  
cloud, user support

# Centralised hardware in the UGent datacenter at campus Sterre (building S10)



# Centralised hardware



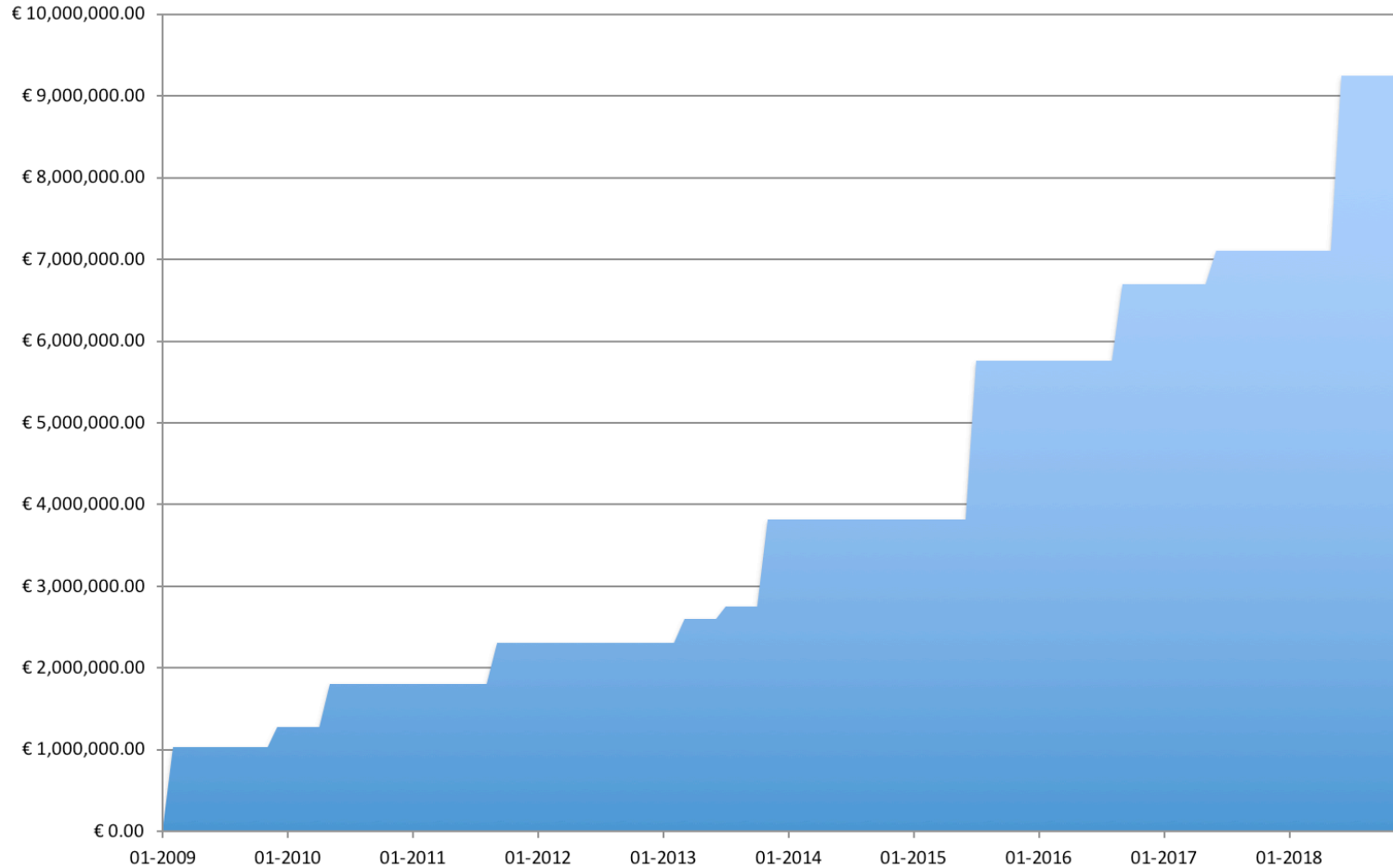
# HPC-UGent Tier-2 (STEVIN): central investments



1548 - 1620  
°Bruges

STEVIN  
HPC  
infrastructure

Total investment in HPC-UGent compute infrastructure



Financing by:



Vlaanderen  
is computing



HPC-UGent  
users

# HPC-UGent Tier-2 (STEVIN)

<https://www.vscentrum.be/infrastructure/hardware/hardware-ugent>



## Compute clusters

6 Tier-2 clusters

> 600 workernodes, > 15,000 cores

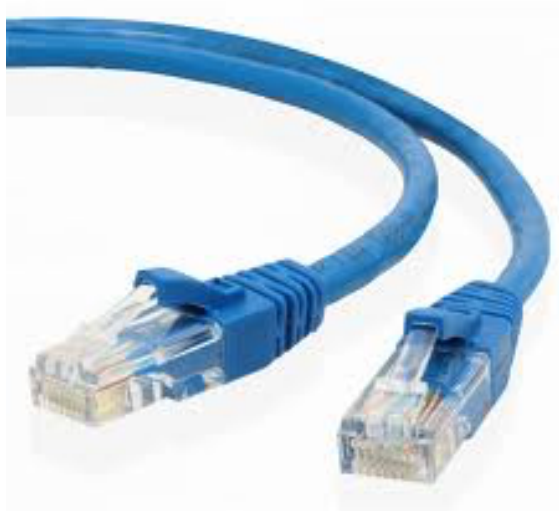
	#nodes	CPU	Mem/node	Diskspace/node	Network
	123	2 x 8-core Intel E5-2670 (Sandy Bridge @ 2.6 GHz)	64 GB	400 GB	FDR InfiniBand
	16	2 x 12-core Intel E5-2680v3 (Haswell-EP @ 2.5 GHz)	512 GB	3x 400 GB (SSD, striped)	FDR InfiniBand
	200	2 x 12-core Intel E5-2680v3 (Haswell-EP @ 2.5 GHz)	64 GB	500 GB	FDR-10 InfiniBand
	128	2 x 10-core Intel E5-2660v3 (Haswell-EP @ 2.6 GHz)	128 GB	1 TB	FDR InfiniBand
	72	2 x 18-core Intel Xeon Gold 6140 (Skylake @ 2.3 GHz)	192 GB	1 TB 240 GB SSD	EDR InfiniBand
	96	2 x 18-core Intel Xeon Gold 6140 (Skylake @ 2.3 GHz)	96 GB	1 TB 240 GB SSD	10 GbE

# HPC-UGent Tier-2 (STEVIN)

Network connections between nodes ('interconnect')

Ethernet: 1-10 Gbit/s

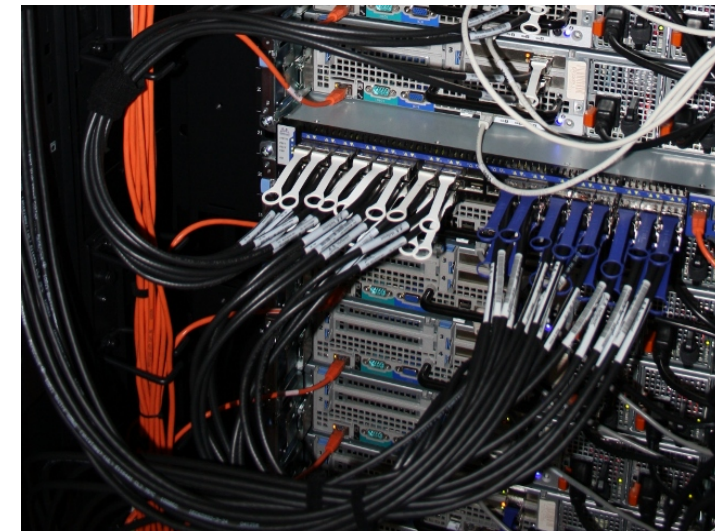
Infiniband: 50 - 100 Gbit/s



€

for single core/node jobs

(too slow for fast inter-node communication)



€€(€)

required for MPI jobs



# VSC Tier-2 infrastructure

Vlaams Supercomputer Centrum  
(Flemish Supercomputer Center)

<https://www.vscentrum.be/en/access-and-infrastructure/tier-2>

Antwerp University association

Brussels University association  
+ Grid specialization

Ghent University association  
+ Big Data specialization

KU Leuven association  
Limburg association University-Colleges  
+ Shared memory, accelerator specialization (GPU)



Vlaanderen  
is computing



# VSC Tier-1 – BrENIAC (@ KUL)

For up to date information, see:

<https://www.vscentrum.be/en/access-and-infrastructure/tier-1>

## Hardware

- 580 computing nodes (16,240 cores in total)
  - Two 14-core Intel Xeon processors (Broadwell, E5-2680v4)
  - 128 GiB RAM (435 nodes) or 256 GiB (145 nodes)
- EDR InfiniBand interconnect
  - High bandwidth (11.75 GB/s per direction, per link)
  - Slightly improved latency over FDR
- Storage system
  - Capacity of 634 TB
  - Peak bandwidth of 20 GB/s

Extension is being installed currently, which should bring total compute power to ~1.5 petaflop.

- 408 additional workernodes,  
each with 2x Intel Skylake 14-core processors
- double the scratch storage volume



# VSC Tier-1 – BrENIAC (@ KUL)

For academics (all Flemish research centers):

- Free of charge
- Starting Grant (100 node days)
  - <https://www.vscentrum.be/en/access-and-infrastructure/tier1-starting-grant>
  - Fill in application form, send it to [hpc@ugent.be](mailto:hpc@ugent.be)
- Project access (500-5000 nodedays)
  - 3 evaluation moments per year
  - Application form and more info  
<https://www.vscentrum.be/en/access-and-infrastructure/project-access-tier1>
- Don't hesitate to contact [hpc@ugent.be](mailto:hpc@ugent.be) for help!



# VSC Tier-1 – BrENIAC (@ KUL)

For industry:

- Exploratory access (100 node days)
  - Free of charge
  - Contact [hpc@ugent.be](mailto:hpc@ugent.be)
- Contract access
  - FWO/UGent/company contract
  - Payed usage (~13 euro / node / day)
  - Contact [hpc@ugent.be](mailto:hpc@ugent.be)



# Getting a VSC account



- See Chapter 2 in HPC-UGent tutorial
- <https://www.vscentrum.be/en/access-and-infrastructure/requesting-access>
- All users of AUGent can request a VSC account
  - Researchers & staff
  - Master/Bachelor students (after motivation of ZAP)
- VSC account be used to use 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 (power outage/maintenance)
  - Have a backup plan at hand
  - Advisable teaching/exam formula: project work

# Managing your VSC account



- You can manage your VSC account via the VSC account page:

<https://account.vscentrum.be>



<b>View Account</b>	Edit Account	View Groups	New/Join Group	Edit Group	New/Join VO	View VO	Edit VO	Reservations	Log Out
---------------------	--------------	-------------	----------------	------------	-------------	---------	---------	--------------	---------

## View account

### General information

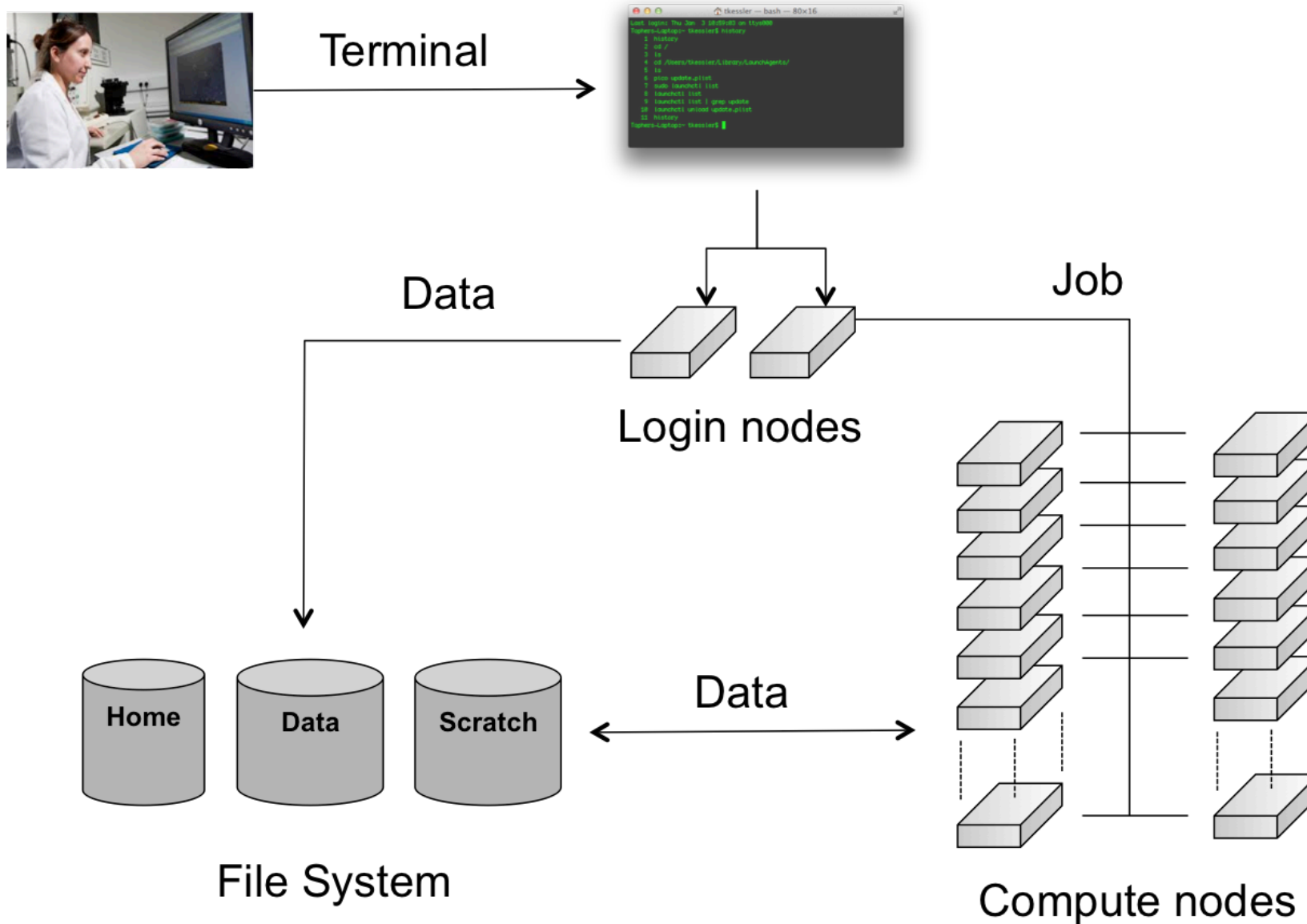
**Uid:** vsc40023

**Institute:** Gent

# Workflow on HPC infrastructure

1. Connect to login nodes
2. Transfer your files
3. (Compile your code and test it)
4. Create a job script
5. Submit your job
6. Be patient
  - Your job gets into the queue
  - Your job gets executed
  - Your job finishes
7. Move your results

# High-level overview of HPC-UGent infrastructure



# Connected to an HPC-UGent login node

```
► ssh vsc40023@login.hpc.ugent.be
Last login: Tue Jan  8 19:29:07 2019 from gligarha01.gastly.os

STEVIN HPC-UGent infrastructure status on Tue, 08 Jan 2019 19:20:01

  cluster - full - free - part - total - running - queued
            nodes nodes free  nodes jobs      jobs
-----
delcatty   2     0     0    125  N/A     N/A
golett    71     0    128   200  N/A     N/A
phanpy    15     1     0    16   N/A     N/A
swalot    46     0    42   128  N/A     N/A
skitty    63     0     1    72   N/A     N/A
victini   57     0    32    96   N/A     N/A

For a full view of the current loads and queues see:
http://hpc.ugent.be/clusterstate/
Updates on maintenance and unscheduled downtime can be found on
https://www.vscentrum.be/en/user-portal/system-status

-bash-4.2$ hostname
gligar05.gastly.os
-bash-4.2$ █
```

# Basic Linux shell usage (interactive)

- command line environment a.k.a. 'shell' a.k.a. bash
- type a command and hit "Enter" to execute it
  - think/double check before executing, commands can be destructive!
- some commands take arguments or options (these start with - or --)
- right-left arrow keys : go forward/backward on current command line
- Ctrl-A / Ctrl-E: go to start/end of command line
- up/down arrow keys : access command history
- Ctrl-R: search through command history
- any line that starts with a '#' (hash) is a comment (not a command)

# Basic Linux shell commands: navigation

`ls` list files/directories in current directory ("what's here?")

`ls -l` long listing (more information)

`ls -lrt` long listing and sorted by last changed (reversed)

`ls example` show contents of directory named 'example'

---

`cd` change directory ("go to ...")

`cd example` change to directory named 'example'

`cd -` change to previous directory

`cd` (without any argument): change back to home directory

---

`pwd` show present working directory ("where am I?")

# Basic Linux shell commands: files & directories

`mkdir` create directory with specified name (min. 1 argument required)

`mkdir -p` create directory + all missing parent directories

---

`cp` copying of files/directories (min. 2 arguments required)

`cp -a` recursive copy (& preserve permissions), required for directories

---

`mv` moving/renaming of files/directories (min. 2 arguments required)

---

`ln -s` create symbolic link between two locations (2 arguments required)

---

`rm` removing files (min. 1 argument required) **BE CAREFUL!**

`rm -f` forced removal (silent if there's nothing to remove)

`rm -r` recursive removal (required for directories)

`rm -rf` forced recursive removal (better think twice before using this...)

There is no "trash bin", if you remove something with '`rm`', it's gone forever!

# Basic Linux shell: environment variables

- environment variables are basically "labeled boxes" (with something inside)
- defining an environment variable named `$EXAMPLE` with value `12345` :

```
export EXAMPLE=12345
```

(note: no output from 'export' command, no \$, no spaces around '=')

- showing the contents of an environment variable (\$ indicates name of env. var.)

```
echo $EXAMPLE
```

- using non-existing environment variables does not produce errors!
- a non-existing environment variable is equivalent to an empty value (be careful!)
- environment variables are only defined in the current session/job (not persistent)!
- print all currently defined environment variables with `env | sort`

# Basic Linux shell: file paths

- file paths are locations to files & directories on a file system
- `.` is a shorthand for the current directory, `..` for the parent directory
- file paths can be either:
  - relative to the current directory  
examples: `file1.txt` , `dir1/file2.txt` , `../../dir2/`
  - absolute (start from `/`, the 'root' of the filesystem)  
example: `/user/gent/400/vsc40000`
- environment variables often have file paths as a value  
examples: `$HOME`, `$VSC_DATA`, `$VSC_SCRATCH`, `$TMPDIR`, ...
- we strongly recommend to use the provided environment variables  
examples: `$VSC_DATA/project1`, `$VSC_SCRATCH/project1/12345.out`

# Basic Linux shell: file contents, editing, output redirection

- you can inspect the contents of (short) files using the `cat` command
- for long files, you can use:
  - `head` or `tail` to inspect the first/last lines of the file
  - a pager command like `less` (scroll with arrow keys or space bar, exit with 'q')
- `nano` is a relatively easy-to-use command line editor (^ means `Ctrl`)
- to capture the output of a command, you can use output redirection:
  - capturing `stdout` (normal output): `command > out.txt`
  - capturing `stderr` (errors & warnings): `command 2> err.txt`
  - capturing *both* in a single file: `command &> err.txt`

# Basic Linux tutorial

- a basic Linux tutorial is available in the HPC-UGent documentation, available at <https://www.ugent.be/hpc/en/support/documentation.htm>
- covers basic usage of the shell environment
- explains commonly used commands
- focus on HPC context & job scripts
- includes a couple of basic exercises
- for questions or problems, don't hesitate to contact [hpc@ugent.be](mailto:hpc@ugent.be) !

```
▶ ssh vsc40023@login.hpc.ugent.be
Last login: Tue Jan  8 19:29:07 2019 from gligarha01.gastly.os

STEVIN HPC-UGent infrastructure status on Tue, 08 Jan 2019 19:20:01

cluster - full - free - part - total - running - queued
         nodes nodes free nodes jobs jobs
-----
delcatty  2    0    0   125   N/A   N/A
golett    71    0  128   200   N/A   N/A
phanpy    15    1    0    16   N/A   N/A
swalot    46    0   42   128   N/A   N/A
skitty    63    0    1    72   N/A   N/A
victini   57    0   32    96   N/A   N/A

For a full view of the current loads and queues see:
http://hpc.ugent.be/clusterstate/
Updates on maintenance and unscheduled downtime can be found on
https://www.vscenrum.be/en/user-portal/system-status

-bash-4.2$ hostname
gligar05.gastly.os
-bash-4.2$ █
```

# Workflow on HPC infrastructure

1. Connect to login nodes
2. Transfer your files
3. (Compile your code and test it)

See Chapter 3 in HPC-UGent tutorial

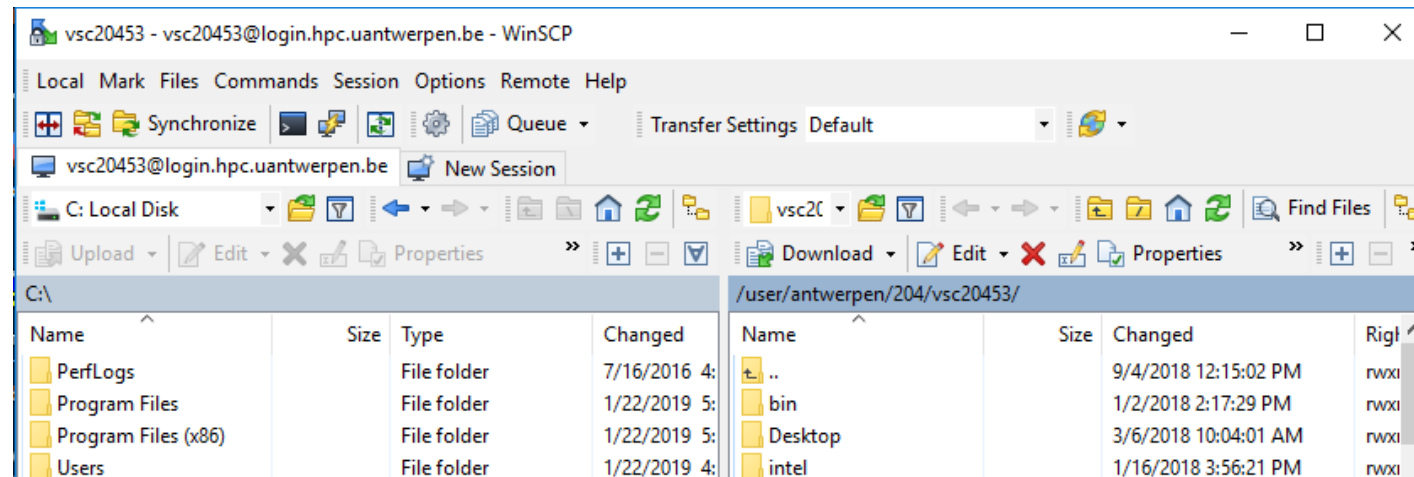
- Users interact with the HPC infrastructure via the login nodes
- No direct access to the workernodes (except when a job is running on it)

- Your job gets executed
- Your job finishes

7. Move your results

# Transferring files to/from the HPC-UGent infrastructure

- see section 3.2 in HPC-UGent tutorial for detailed information
- via login nodes
- on Linux or macOS:
  - using 'scp' in terminal window (use 'scp -r' for directories)
    - or 'rsync' for large transfers (can be restarted)
  - or graphical tool like built-in file manager or Cyberduck
- on Windows: WinSCP tool (left: own system; right: HPC; drag 'n drop)



# Workflow on HPC infrastructure

1. Connect to login nodes
2. Transfer your files
3. (Compile your code and test it)
4. Create a job script
5. Submit your job

- Choose correct PBS directives (Chapter 4, 8)
- Load software modules (Chapter 3)
- Useful environment variables (Chapter 4)
- Access files on shared filesystems (Chapter 6)

7. Move your results

# What is a job script?

```
#!/bin/bash  
echo "hello world"
```

A job (shell) script is a text file that specifies:

- the resources that are required by the calculation  
(number of nodes/cores, amount of memory, how much time, ...)
- the software that is used for the calculation  
(via `module load` commands)
- the steps that should be done to execute the calculation  
(starting from `$HOME`), specified as shell commands, typically:
  - 1) staging in of input files
  - 2) running the calculation
  - 3) staging out of results

# Job scripts: required resources via #PBS directives

```
#!/bin/bash
#PBS -N solving_42          ## job name
#PBS -l nodes=1:ppn=4      ## single-node job, 4 cores
#PBS -l walltime=10:00:00  ## max. 10h of wall time
#PBS -l vmem=50gb          ## max. 50GB virtual memory
<rest of job script>
```

- required resources can be specified via #PBS lines in job script (or via qsub)
- maximum walltime: 72 hours
- for longer jobs, use checkpointing
  - preferable internal/application checkpointing
  - external checkpointing by submitting jobs via csub
    - see Chapter 14 in HPC-UGent tutorial

# Job scripts: software modules

- All user-end software is made available via modules
- Modules prepare the environment for using the software
- Module naming scheme: `<name>/<version>-<toolchain>[-<suffix>]`

Load a module to use the software:

```
$ module load Python/3.6.6-intel-2018b
```

See currently loaded modules using:

```
$ module list    or    $ ml
```

Get overview of available modules using:

```
$ module avail  or    $ ml av
```

- Only mix modules built with the same (version of) compiler toolchain.  
e.g., `intel` (Intel compilers, Intel MPI, Intel MKL (BLAS, LAPACK))
- See also section 4.1 in HPC-UGent tutorial

# Job scripts: useful environment variables

(most of these are only defined in the context of jobs!)

- **\$PBS\_JOBID**
  - job id of running job
- **\$PBS\_O\_WORKDIR**
  - directory from which job was submitted on login node
  - common to use 'cd \$PBS\_O\_WORKDIR' at beginning of job script
- **\$PBS\_ARRAYID**
  - array id of running job; only relevant when submitting array jobs (`qsub -t`)
- **\$TMPDIR**
  - Local directory specific to running job
  - Cleaned up automatically when job is done!
- **\$EBROOTFOO, \$EBVERSIONFOO**
  - root directory/version for software package Foo
  - only available when module for Foo is loaded

# Job scripts: input data & filesystems

- See Section 6.2 in HPC-UGent tutorial
- Think about input/output:
  - How will you stage in your data and input files?
  - How will you stage out your output files?
- Manually (on login nodes) vs automatically (as a part of job script)
- Home filesystem: 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

- home directory (`$VSC_HOME`): 3GB (fixed)
- personal data directory (`$VSC_DATA`): 25GB (fixed)
- personal scratch directory (`$VSC_SCRATCH`): 25GB (fixed)
- current quota usage can be consulted on VSC accountpage  
<https://account.vscenrum.be>
- more storage quota (GBs, TBs) available for virtual organisations (VOs)
- see Section 6.6 in HPC-UGent tutorial
- additional quota can be requested via <https://account.vscenrum.be/django/vo/edit>
- shared directories with VO members: `$VSC_DATA_VO`, `$VSC_SCRATCH_VO`
- personal VO subdirectories: `$VSC_DATA_VO_USER`, `$VSC_SCRATCH_VO_USER`

# Current storage usage - personal directories

- consult VSC accountpage - <https://account.vscentrum.be> ("View Account" tab)  
(for now, only data volumes, not number of files (inode quota))

## Usage

---

### Personal

Storage name	Used	Quota	%
VSC_HOME	1.98 GiB	2.85 GiB	69.57%
VSC_DATA	0 B	23.75 GiB	0.00%
VSC_SCRATCH_KYUKON	0 B	23.75 GiB	0.00%
VSC_SCRATCH_PHANPY	0 B	512.0 KiB	0.00%

# Current storage usage - own VO directories

- consult VSC accountpage - <https://account.vscenrum.be> ("View Account" tab)  
(for now, only data volumes, not number of files (inode quota))

## Virtual Organisation

Storage name	Virtual Organisation	Used	Quota	%
VSC_DATA_VO	gvo00002	1.22 TiB	1.64 TiB	74.41%
VSC_SCRATCH_KYUKON_VO	gvo00002	3.24 TiB	4.52 TiB	71.55%
VSC_SCRATCH_PHANPY_VO	gvo00002	2.29 TiB	6.78 TiB	33.79%

# Current storage usage - total VO usage

- consult VSC accountpage - <https://account.vscentrum.be> ("View VO" tab)  
(for now, only data volumes, not number of files (inode quota))
- detailed info per VO member can only be consulted by VO administrators!

## Virtual Organisation quota

Name	Used	Quota	%
VSC_DATA_VO	2.8 TiB	3.28 TiB	85.20%
VSC_DATA_SHARED_VO	0 B	1.9 GiB	0.00%
VSC_SCRATCH_KYUKON_VO	3.94 TiB	9.05 TiB	43.61%
VSC_SCRATCH_PHANPY_VO	2.29 TiB	9.05 TiB	25.34%

## VSC\_DATA\_VO

User	Used	Quota	%
vsc40023	1.22 TiB	1.73 TiB	70.69%
vsc40002	146.76 GiB	1.73 TiB	8.29%
vsc41206	0 B	1.73 TiB	0.00%

# Job scripts: full example (single-core job)

```
#!/bin/bash
#PBS -N count_example          ## job name
#PBS -l nodes=1:ppn=1         ## single-node job, single core
#PBS -l walltime=2:00:00      ## max. 2h of wall time

module load Python/3.6.6-intel-2018b
# copy input data from location where job was submitted from
cp $PBS_O_WORKDIR/input.txt $TMPDIR
# go to temporary working directory (on local disk) & run
cd $TMPDIR
python -c "print(len(open('input.txt').read()))" > output.txt
# copy back output data, ensure unique filename using $PBS_JOBID
cp output.txt $VSC_DATA/output_${PBS_JOBID}.txt
```

# Job scripts: full example (multi-node job)

```
#!/bin/bash
#PBS -N mpi_hello          ## job name
#PBS -l nodes=2:ppn=all    ## 2 nodes, all cores per node
#PBS -l walltime=2:00:00   ## max. 2h of wall time

module load intel/2018b
module load vsc-mypirun

# go to working directory, compile and run MPI hello world
cd $PBS_O_WORKDIR
mpicc mpi_hello.c -o mpi_hello
mypirun ./mpi_hello
```

# Jobs scripts: generated output files

- Your job script may produce informative/warning/error messages.
  - Two output files are created for each job: stdout (\*.o) + stderr (\*.e)
  - Located in directory where job was submitted from (by default)
  - Messages produced by a particular command in the job script can be "caught" and redirected to a particular file instead.

```
example > out.log 2> err.log
```

(see section 5.1 of our Linux tutorial for more details)

- In addition, the software used for the calculation may have generated additional output files (very software-specific).

# Workflow on HPC infrastructure

1. Connect to login nodes
2. Check the status of the system
3. Check the status of your job
4. Create a job script
5. Submit your job
6. Be patient
  - Your job gets into the queue
  - Your job gets executed
  - Your job finishes
7. Move your results

# Demo: qsub, qstat, qdel

- Submit job scripts from a login node to a cluster for execution using qsub:

```
$ module swap cluster/golett
$ qsub example.sh
12345.master19.golett.gent.vsc
```

- An overview of the active jobs is available via qstat:

```
$ qstat
```

Job id	Name	User	Time Use	S	Queue
-----	-----	-----	-----	-	-----
12345.master19	example	vsc40000	07:39:30	R	long

- To remove a job that is no longer necessary, use qdel:

```
$ qdel 12345
```

# Job scheduling

- All our clusters use a fair-share scheduling policy.
- No guarantees on when job will start, so plan ahead!
- Job priority is determined by:
  - historical usage
    - aim is to balance usage over users
    - infrequent/frequent users => higher/lower priority
  - requested resources (# nodes/cores, walltime, memory, ...)
    - large 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

# Embarrassingly parallel jobs

- Use case: lots of ((very) short) single-core tasks
- Submitting lots of tiny jobs (minutes of walltime) is not a good idea
  - overhead for each job (node health checks), lots of bookkeeping (job scripts, failed jobs, output files)
- Better approach:
  - Array jobs
    - Single job script, but still lots of submitted jobs
    - Each job is assigned a unique id (`$PBS_ARRAYID`); can be used to select input file, parameters, ...
  - GNU parallel ([https://www.gnu.org/software/parallel/parallel\\_tutorial.html](https://www.gnu.org/software/parallel/parallel_tutorial.html))
    - General-purpose tool to easily running shell commands in parallel with different inputs
    - Use 'parallel' command in your job script
  - Worker (<https://www.vscentrum.be/cluster-doc/running-jobs/worker-framework>)
    - One single job that processes a bunch of tasks (multi-core or even multi-node)
    - Job script is parameterized, submit with 'wsub' rather than 'qsub'

# Software installations

To submit a request for software installation:

<https://www.ugent.be/hpc/en/support/software-installation-request>

Always include:

- software name and website
- location to download source files
  - or make install files available in your account
- build instructions (if you have them)
- a simple test case with expected output
  - including instructions on how to run it

Requests may take a while to process; make the request sooner rather than later!

# Questions, problems, getting help

Don't hesitate to contact HPC-UGent support: [hpc@ugent.be](mailto:hpc@ugent.be)

Always include:

- VSC login id
- clear description of problem (or question)
- location of job script and output/error files in your account
  - don't send them in attachment, we prefer to look at it 'in context'
- job IDs, which cluster

Preferably use your UGent email address.

Alternatives:

- short meeting (for complex problems, big projects)
- [hpc-users mailing list](#)

# Working with Blender on the HPC infrastructure

(command-line) interactive use/testing on login nodes:

- log in with SSH
- load a Blender module
  - pick one, see output of `module avail Blender`
  - `module load Blender/2.79b-intel-2018b-Python-3.6.6`
- use `blender` command, see output of `blender --help`
- **only run short test commands on login nodes!**

# Working with Blender on the HPC infrastructure

Running compute-intensive Blender calculations:

- log in with SSH
- create a job script
  - specify requires resources (walltime, cores, memory)
  - load `Blender` module (& tweak environment)
  - navigate to correct directory
  - run `blender` command
- submit job script using `qsub`, check job status using `qstat`
- once job completes, check results in directory

# Working with Blender on the HPC infrastructure

Example job script:

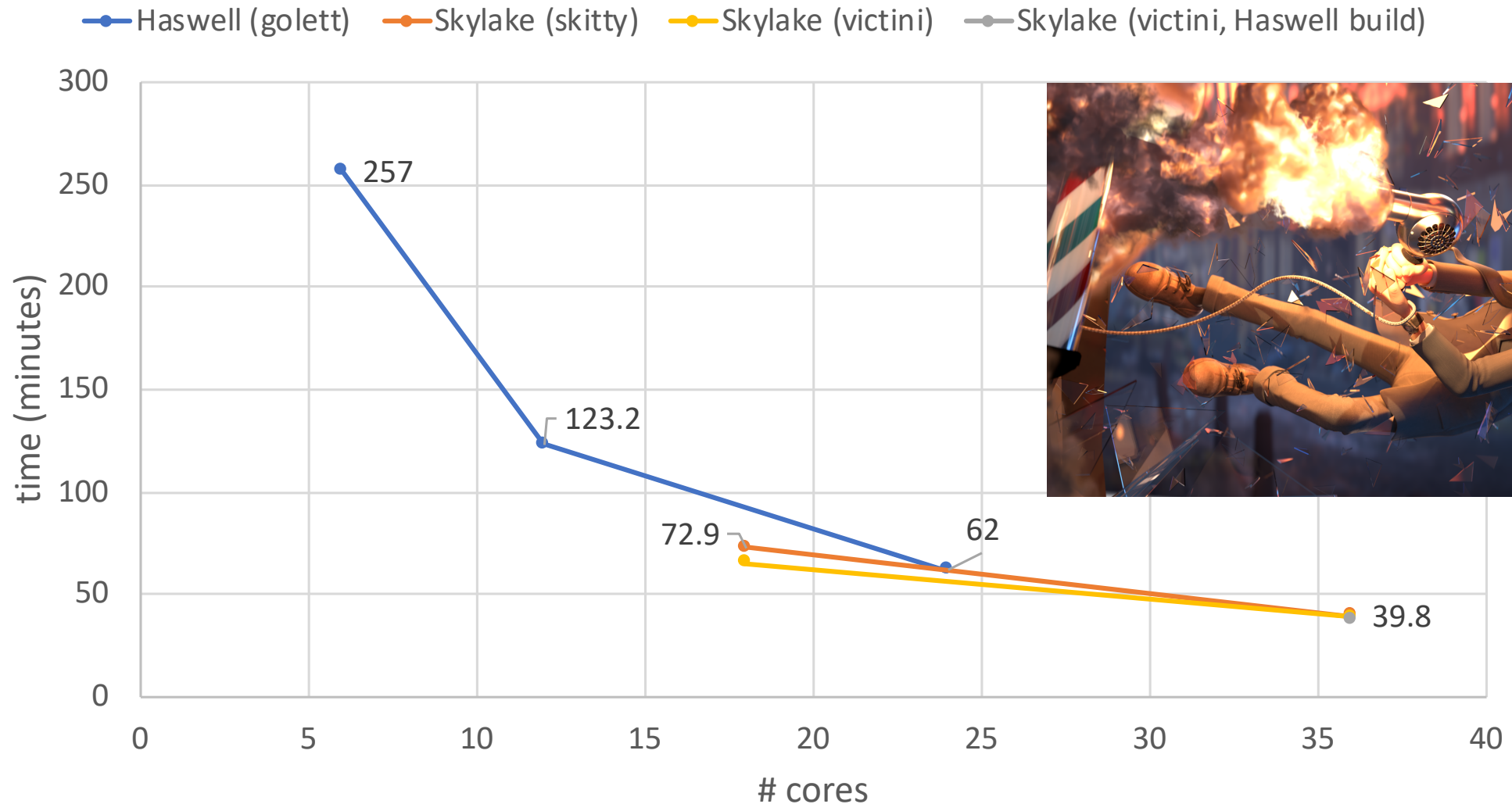
```
#!/bin/bash
#PBS -l nodes=1:ppn=all
#PBS -l walltime=2:0:0

module load Blender/2.79b-intel-2018b-Python-3.6.6
export OMP_NUM_THREADS=1

cd $PBS_O_WORKDIR
time blender -y -b splash279.blend -o $PWD/out -f 11
```

# Working with Blender on the HPC infrastructure

## Benchmark results with Agent 327 demo (single-frame)



# Working with Blender on the HPC infrastructure

## TODO:

- further tuning for better performance (thread pinning, ...)
- look into enhancements made by IT4Innovations
  - better vectorisation support + BVH/Embree optimisations
  - multi-core + distributed rendering via OpenMP/MPI
  - using HPC infrastructure as "backend" rendering farm
  - see <https://www.youtube.com/watch?v=KkndP8dDz50>