



Webinar #4

Introduction to Biomedical Applications on High Performance Computers

7 June 2018

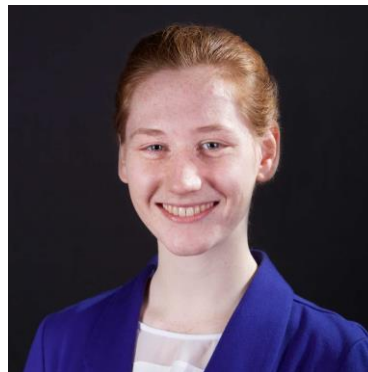
The webinar will start at
12pm CET / 11am GMT



Dr Gavin J. Pringle ([EPCC](#))



Dr Gábor Závodszy ([UvA](#))



Moderated by **Britt Van Rooij** (UvA)





Webinar #4

Introduction to Biomedical Applications on High Performance Computers

7 June 2018

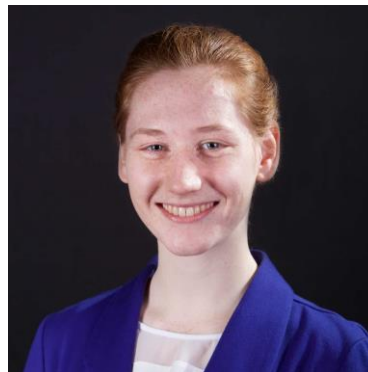
Welcome!



Dr Gavin J. Pringle ([EPCC](#))



Dr Gábor Závodszy ([UvA](#))



Moderated by **Britt Van Rooij** (UvA)



Introduction to Biomedical Applications on High Performance Computers

13 June 2018

Dr Gavin J. Pringle
EPCC, University of Edinburgh
Gábor Závodszy
University of Amsterdam

Reusing this material



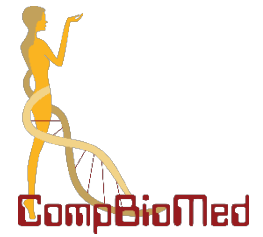
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

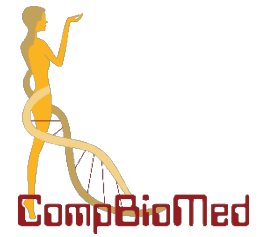
Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

Overview (1/2)



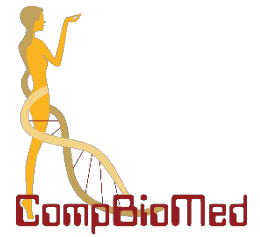
- Why HPC?
 - Examples of who uses it
- Anatomy of a regular computer
 - Performance considerations
 - Multi-core
 - OS, processes and threads
- Anatomy of a High Performance Computer
 - Typical layout
 - Modern HPC Architectures
 - Parallel programming models
- Practical details
 - Login and command line access
 - Text editors
 - Job submission via a batch systems
- Measuring Performance
 - How many cores should I employ?!

Overview (2/2)



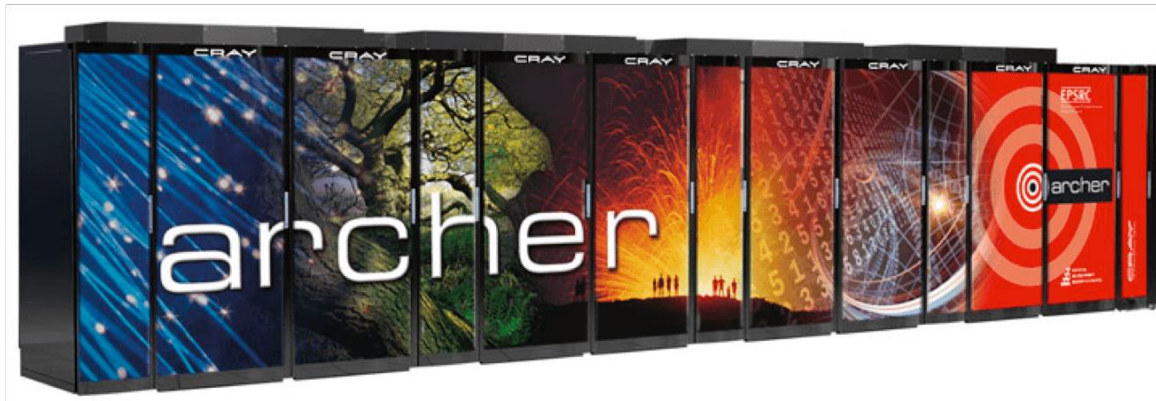
- Meet an HPC code and an HPC machine
 - HemoCell
 - Lisa @ SurfSARA
- How to set up the development environment
 - What makes it different from working on your laptop?
 - Obtaining the source
 - File transfer
 - Module system in a nutshell
 - Compilation
- Executing a simulation
 - Development short-runs on a login node
 - Queueing system in practice (PBS)
 - Handling multiple jobs
- Evaluate the output
 - Parallel output and how to handle it
 - Post-processing large datasets
 - Visualize information of interest

Why HPC?



- Scientific **simulation** and modelling drive the need for greater computing power.
- Single systems cannot not be provided that had enough resource for the simulations needed.
 - Making faster single chip is difficult due to both physical limitations and cost.
 - Adding more memory to single chip is expensive and leads to complexity.
- Solution: **parallel computing**
 - divide up the work among numerous linked systems.

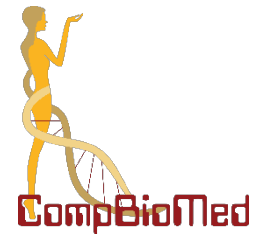
What does a High Performance Computer look like? UK's ARCHER



Spain's MareNostrum

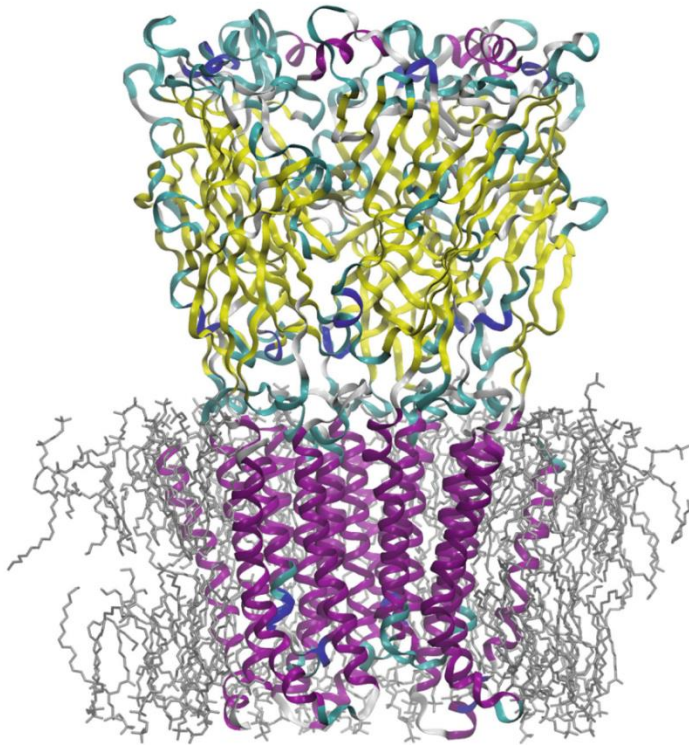


Traditionally, who uses HPC?



- materials science / solid state physics
- computational chemistry
- biomolecular simulations
- particle physics
- environmental modelling
 - weather & climate
 - Geosciences
 - oceanography
- many engineering applications
 - designing trains, planes, automobiles, combustion engines, flight simulators, mining, earthquake/tsunami warning systems...

GROMACS Biomolecular Example



Ligand-gated ion-channel membrane protein GLIC (colored), embedded in a lipid membrane (grey), solvated in water (not shown)

145,000 atoms

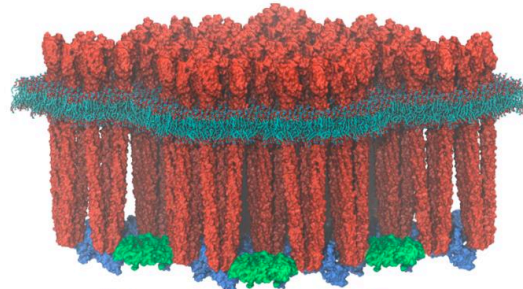
Taken from:

https://doi.org/10.1007/978-3-319-15976-8_1

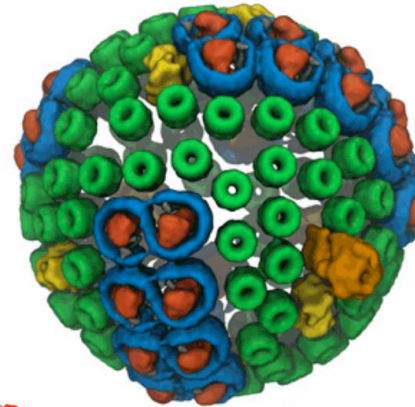
(Lindahl E. *et al.*) (2015) Tackling Exascale Software Challenges in Molecular Dynamics Simulations with GROMACS. In: Markidis S., Laure E. (eds) Solving Software Challenges for Exascale. EASC 2014. Lecture Notes in Computer Science, vol 8759

NAMD Biomolecular Examples

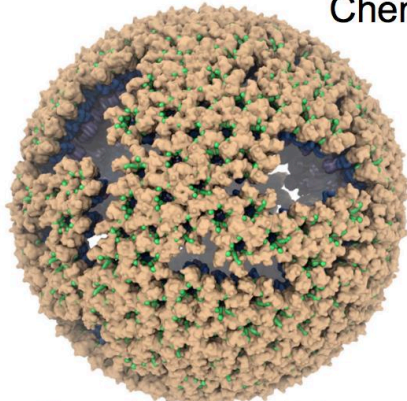
A Sampling of Petascale Projects Using NAMD



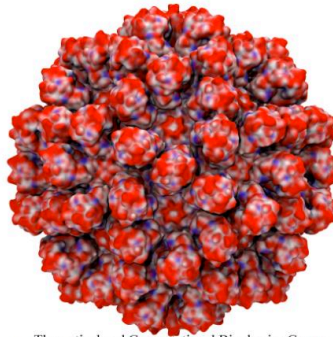
Chemosensory Array



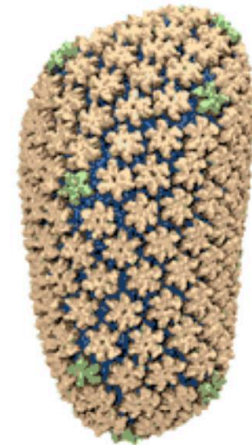
Chromatophore



Rous Sarcoma Virus



Rabbit Hemorrhagic Disease

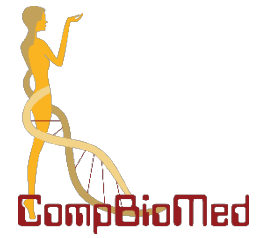


HIV

Theoretical and Computational Biophysics Group

Theoretical and Computational Biophysics Group
University of Illinois at Urbana-Champaign

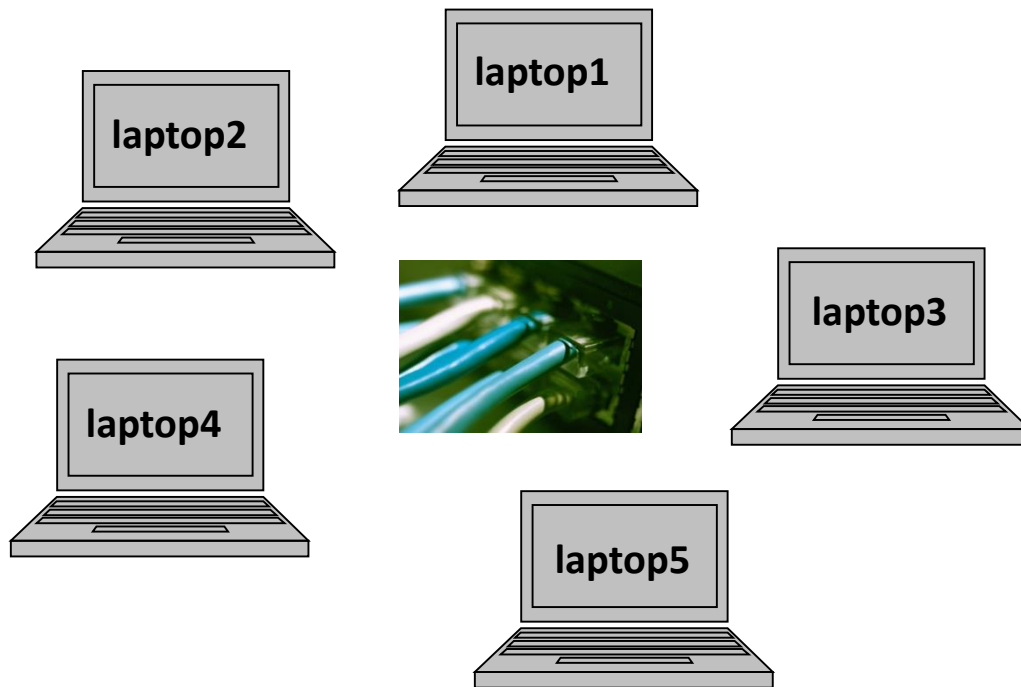
CompBioMed Virtual Human video



- <https://www.youtube.com/watch?v=1FvRSJ9W734>

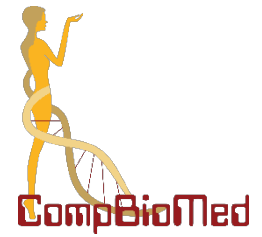
Generic Parallel Machine (computer cluster)

- Rough conceptual model is a collection of laptops
- Connected together by a **network** so they can all communicate



- Consider each laptop as a **compute node**
 - has a processor, hard disk, memory, etc.
 - Each runs a copy of an operating system (Linux)
 - If each processor has 4 **cores**, total system has 20 cores

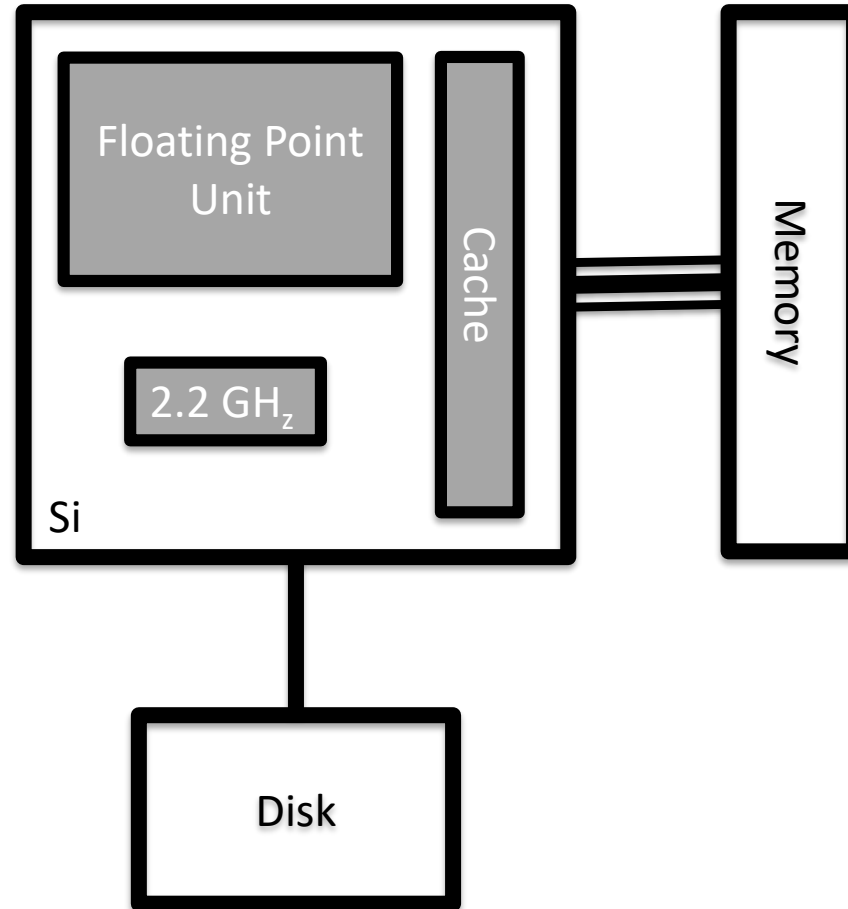
HPC vs other types of computing



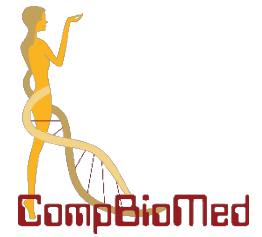
- HPC is one extreme in a continuum of computing:
 - Individual desktop/laptop
 - University research group / departmental machine (server or cluster)
 - University-wide, regional or national-level HPC machine
- Commercial **data** centres (Amazon, Google, Facebook, etc.) have enormous computing clusters
 - These do not cater for scientific computing
 - As such, they do not need a fast interconnect
- HPC machines optimised for traditional science applications:
 - strong floating-point performance (“number crunching”)
 - fast networking
 - software stack that includes scientific / maths libraries

Anatomy of a regular computer

Anatomy of a computer

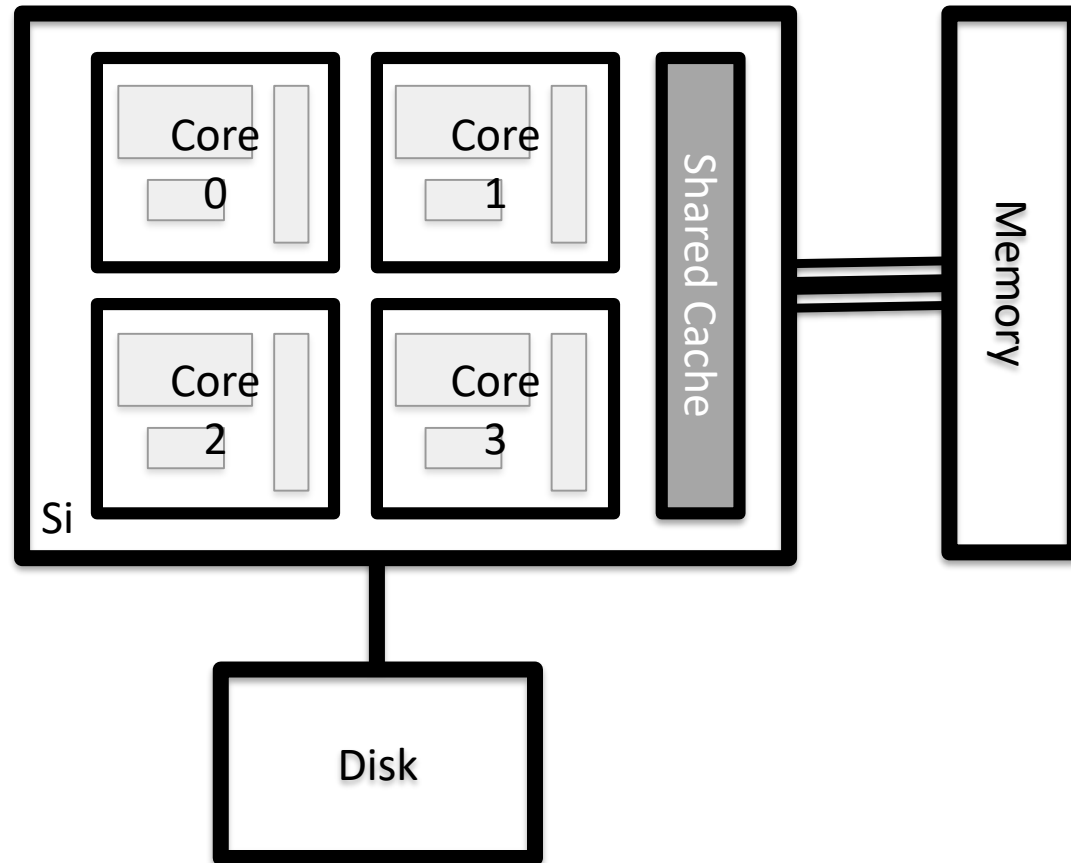


Performance (time to solution) on a single computer depends on...

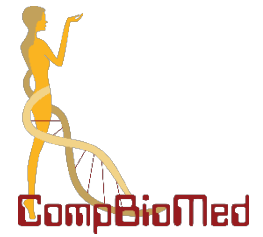


- Clock speed
 - how fast the processor is
- Floating point unit
 - how many operands can be operated on and what operations can be performed?
- Memory latency
 - how fast can we access the data?
- Memory bandwidth
 - how much data can we access in one go?
- Input/Output (IO) to storage
 - how quickly can we access persistent data (files)?

Processors with multiple cores



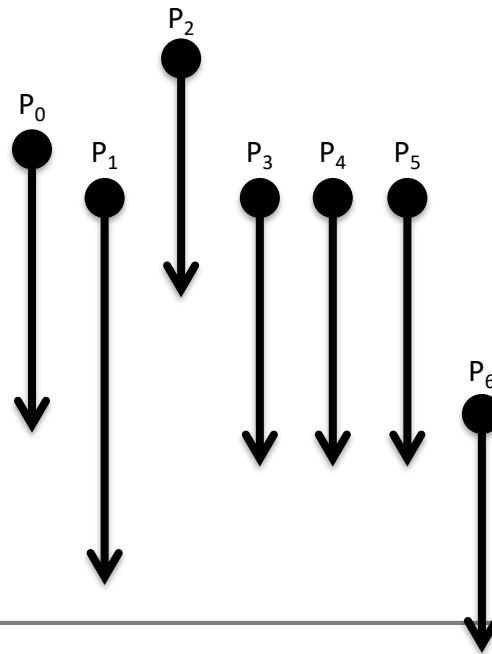
Operating System (OS)



- The OS is responsible for orchestrating access to the hardware by applications.
 - Which cores is an application running on?
 - How is the memory allocated and deallocated?
 - How is the filesystem accessed?
 - Who has authority to access which resources?
 - How do we deal with oversubscription (e.g. more applications running than cores available).
- Running applications are controlled through the concepts of *processes* and *threads*.
- *HPC systems typically use **Linux** (of various flavours)*

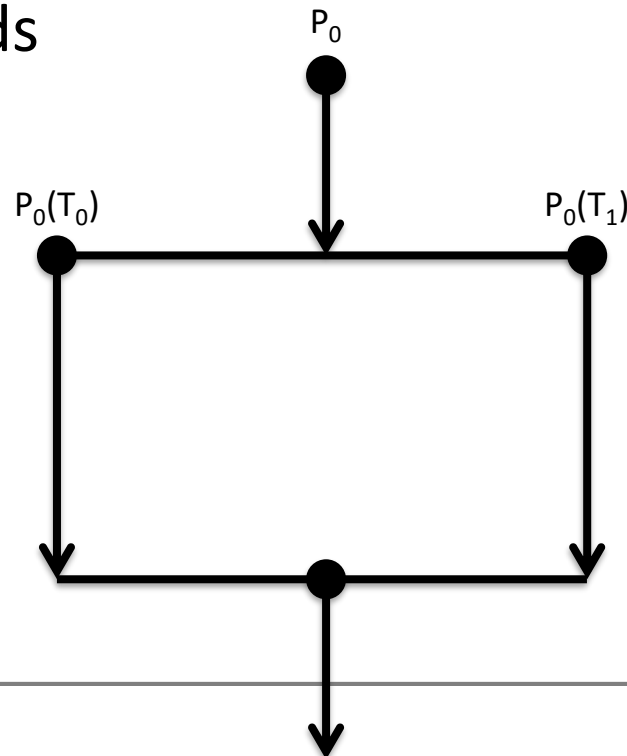
Processes

- Each application is a separate *process* in the OS
 - A process has its **own memory space** which is not accessible by other running process.
 - Each process is scheduled to run by the OS – it can be **typed** to a particular core or can **migrate** between cores



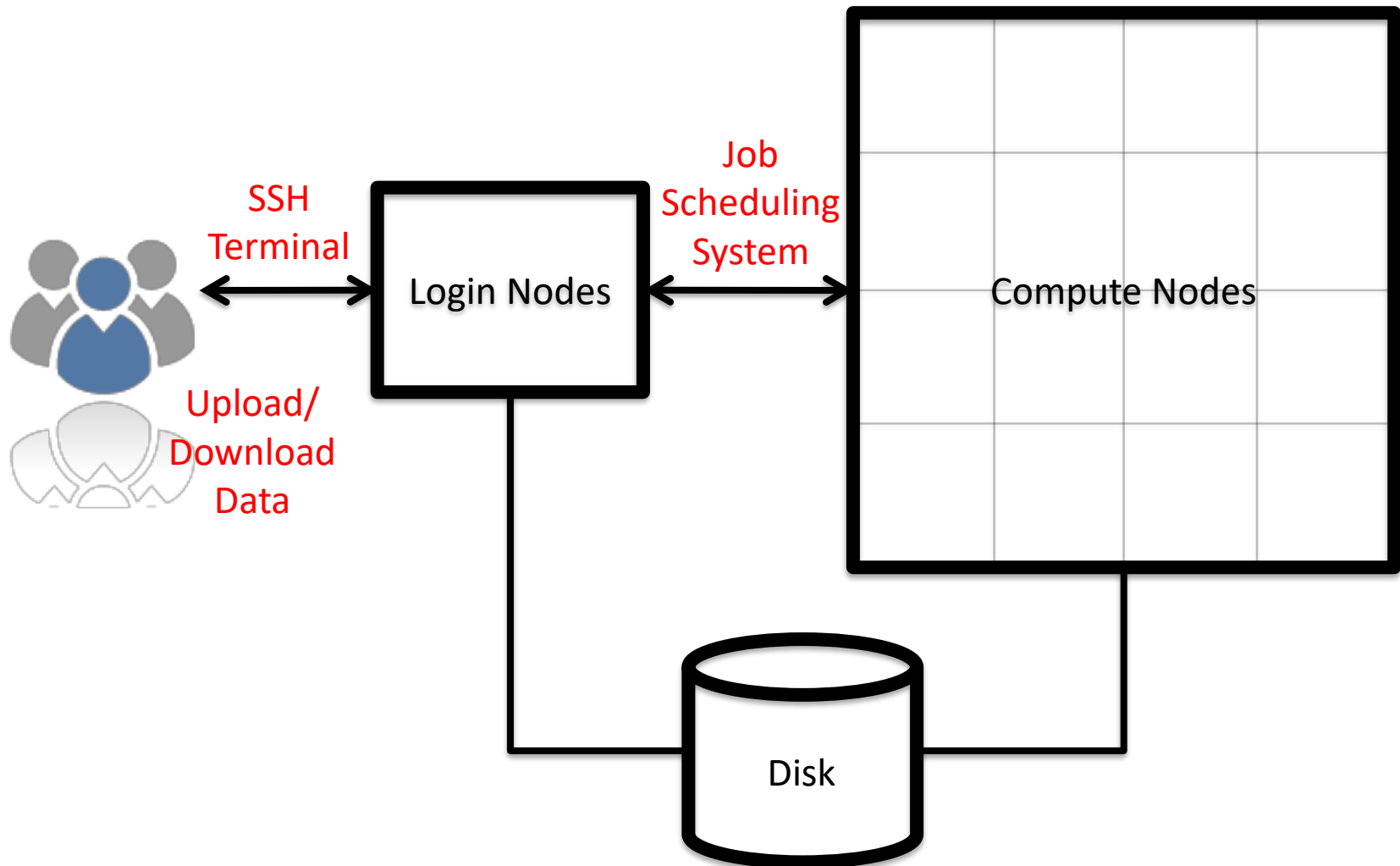
Threads

- For many applications each process has a single *thread*...
- ...but with the advent of multicore processors it is becoming more common for a process to contain multiple threads

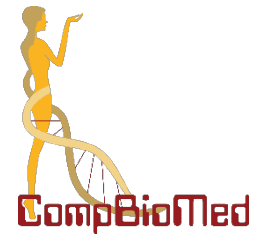


Anatomy of an High Performance Computer

Typical HPC system layout



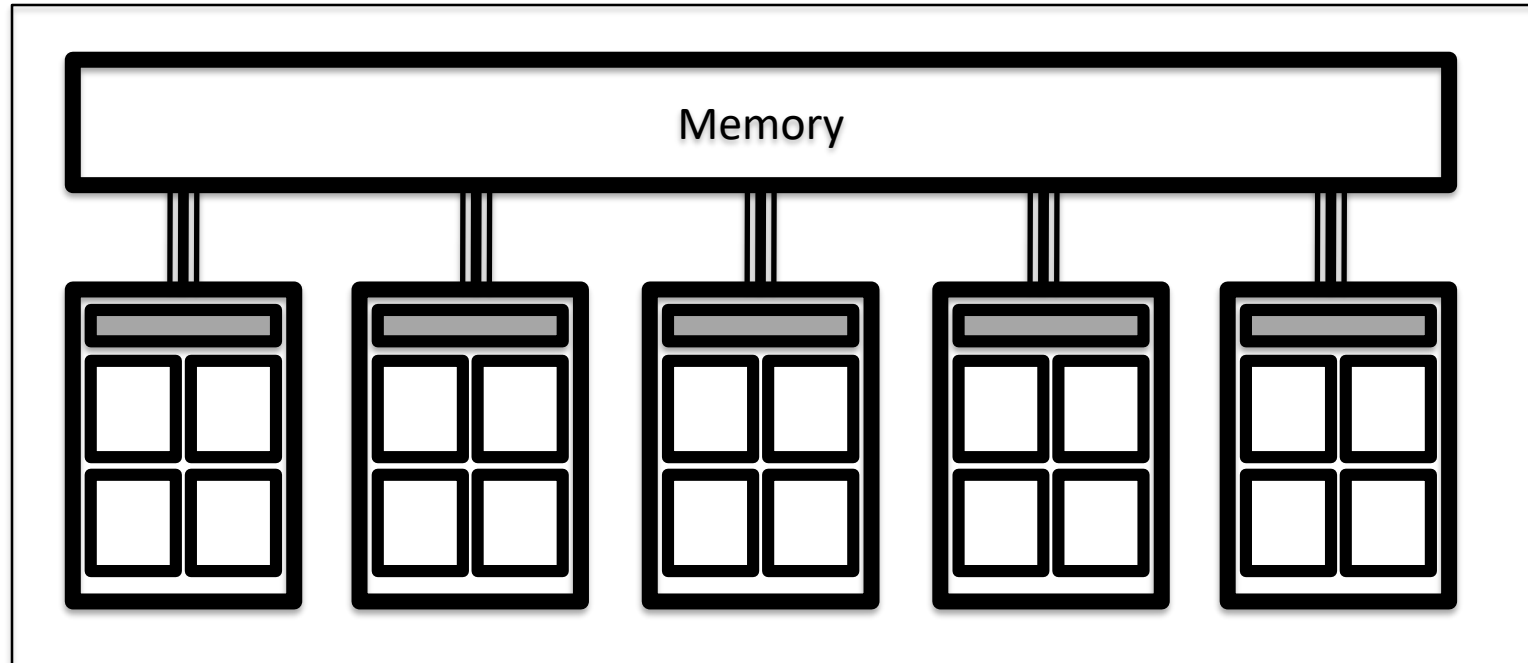
Compute node OS



- On HPC systems the “compute nodes” (“back-end nodes”) often run an optimised OS to improve performance
 - “Login nodes” (“front-end nodes”) nodes usually run a full OS
 - Often means that you are “*cross-compiling*”
 - *Compiling on a front-end node which may be different architecture for the target back-end nodes.*
- How is the OS optimised?
 - Remove features that are not needed (e.g. USB support)
 - Restrict scheduling flexibility and increase interrupt period
 - Remove support for virtual memory (paging)
 - Your application will simply stop if it runs out of physical memory
 - Often, nodes do not have their own disk

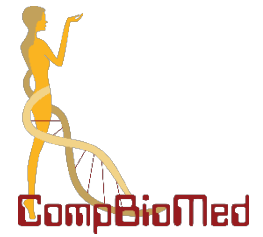
- **All** HPC machines are **parallel** architectures
- Often constructed by combining many pieces of *commodity* hardware
- There are two fundamental parallel architectures:
 - **Shared**-memory systems
 - **Distributed** memory systems
- **HPC** systems typically combine features of **both** shared- and distributed-memory architectures

Shared-Memory Architectures



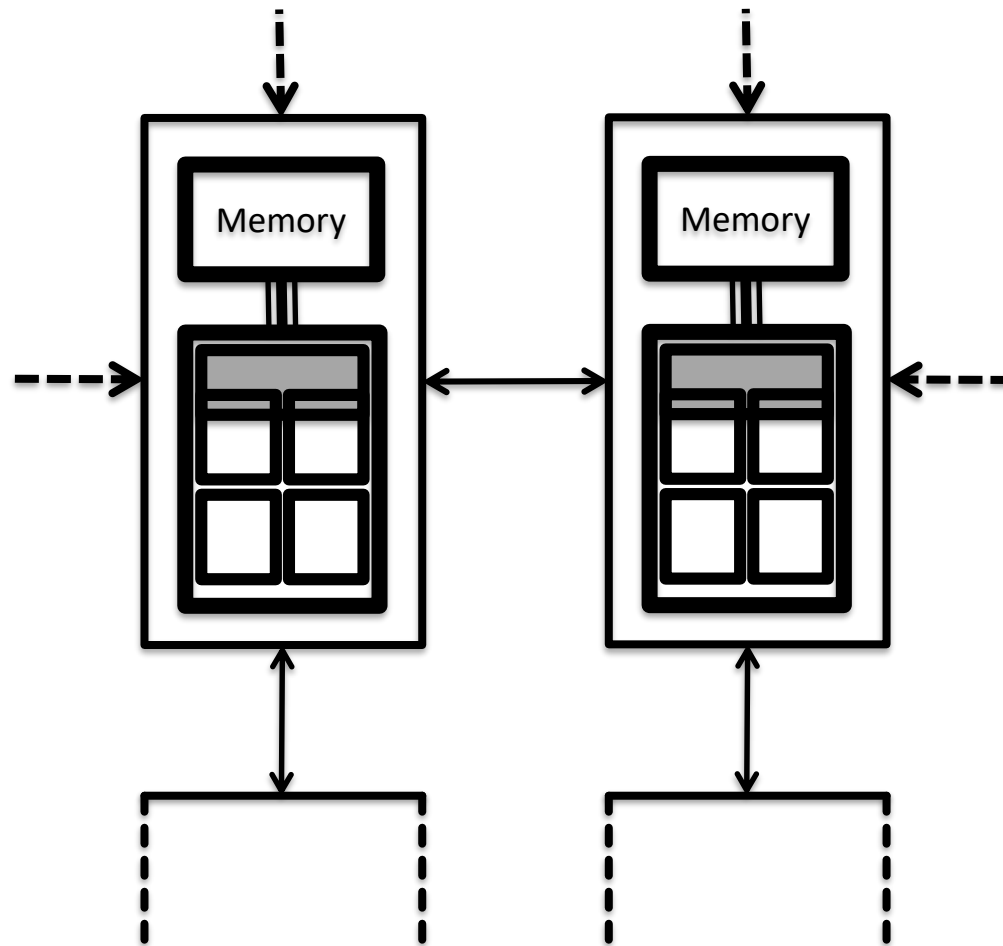
- Multiple processors connected to memory
 - Each processor has multiple cores

Shared-memory architectures

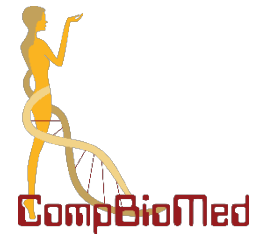


- Most computers are now shared memory machines due to the advent of multicores
 - Mobile phones and laptops included
- Difficult to build shared-memory systems with large core numbers (> 1024 cores)
 - Expensive and power hungry
 - Some systems manage by using software to provide shared-memory capability

Distributed-Memory Architectures

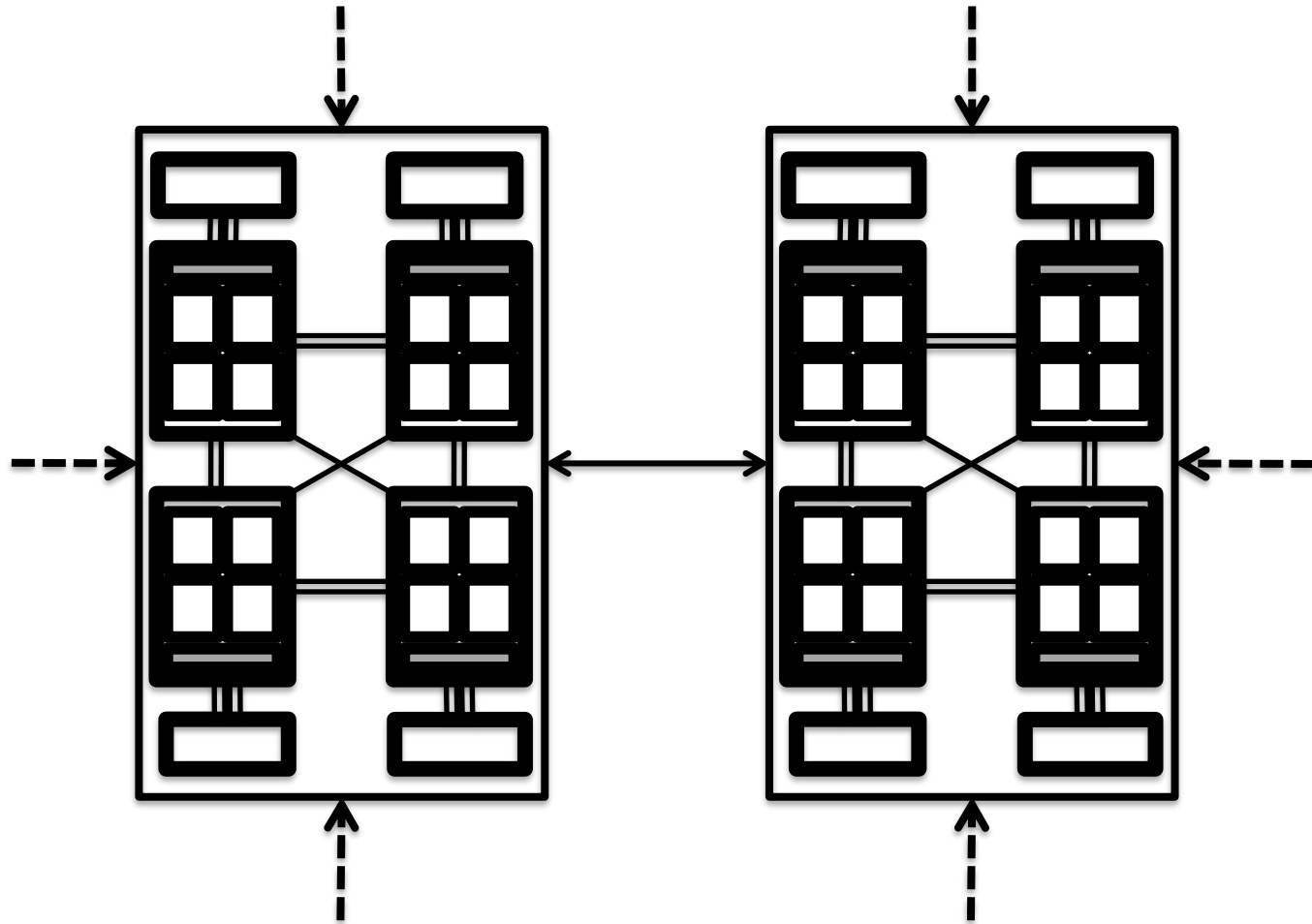


Distributed-Memory architectures

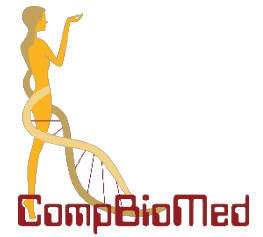


- Each self-contained part is called a ***node***.
- Almost all HPC machines are **distributed** memory
 - All tend to be **shared-memory within** a node.
 - With at least one multi-core processor
- The performance of parallel programs often depends on the **interconnect** performance

Hybrid Architectures



Hybrid Architectures

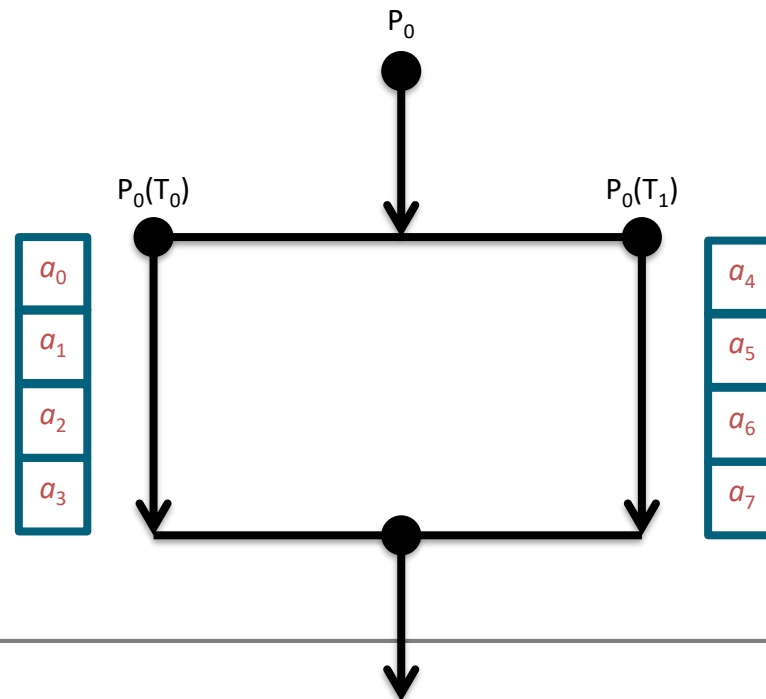


- Almost all HPC machines fall in this class
- Most applications use a message-passing model for programming
 - Typically use MPI, with a single MPI task per core
- Can use threaded programming, i.e. OpenMP or POSIX
 - NB Cannot be used across nodes

HPC Programming Models

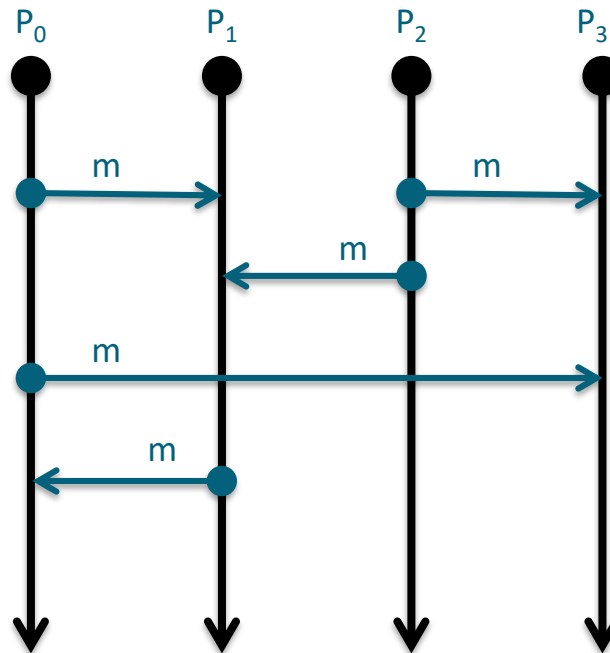
Shared-Memory Concepts

- Threads “communicate” by having access to the same memory space
 - **Any** thread can alter **any** bit of data
 - No explicit communications between the parallel tasks



Message-Passing Concepts

- Each process does not have access to another process's memory
- Communication is usually explicit



Hybrid Programming?

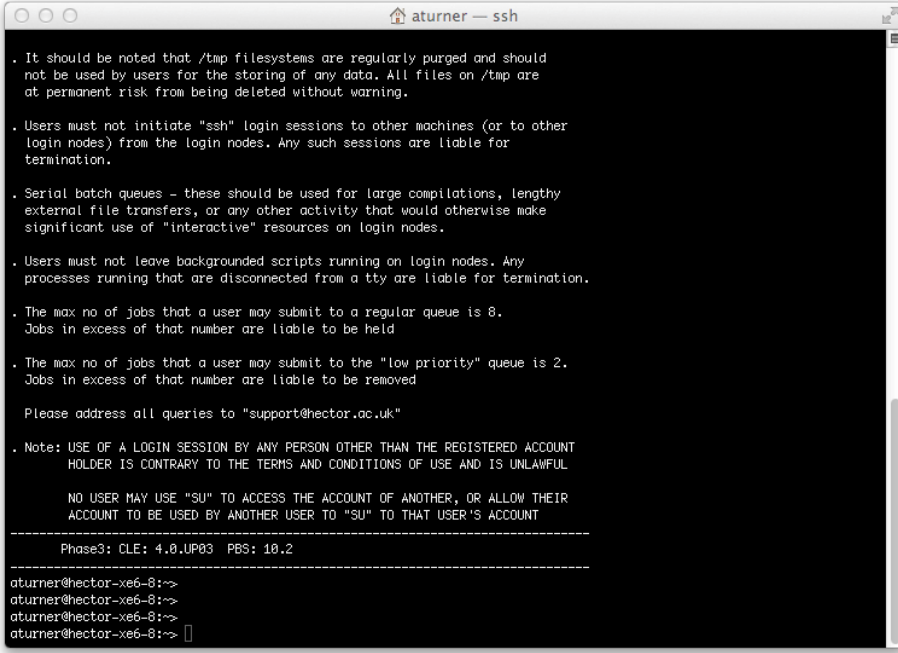
- Increased use of hybrid message-passing + shared memory programming on hybrid architectures.
 - MPI + OpenMP
- Usually use 1 or more MPI process per *NUMA* region and then the appropriate number of shared-memory threads to occupy all the cores
 - **multiple MPI processes per node** gives best performance.
- Placement of processes and threads can become complicated on these machines
 - And will affect performance

Using HPC Systems

How does it work in practice?

Accessing HPC resources: SSH

- Systems usually accessed via SSH
 - Natively on Mac and Linux machines
 - Install PuTTY on Windows machines



```
aturner — ssh

. It should be noted that /tmp filesystems are regularly purged and should
not be used by users for the storing of any data. All files on /tmp are
at permanent risk from being deleted without warning.

. Users must not initiate "ssh" login sessions to other machines (or to other
login nodes) from the login nodes. Any such sessions are liable for
termination.

. Serial batch queues - these should be used for large compilations, lengthy
external file transfers, or any other activity that would otherwise make
significant use of "interactive" resources on login nodes.

. Users must not leave backgrounded scripts running on login nodes. Any
processes running that are disconnected from a tty are liable for termination.

. The max no of jobs that a user may submit to a regular queue is 8.
Jobs in excess of that number are liable to be held

. The max no of jobs that a user may submit to the "low priority" queue is 2.
Jobs in excess of that number are liable to be removed

Please address all queries to "support@hector.ac.uk"

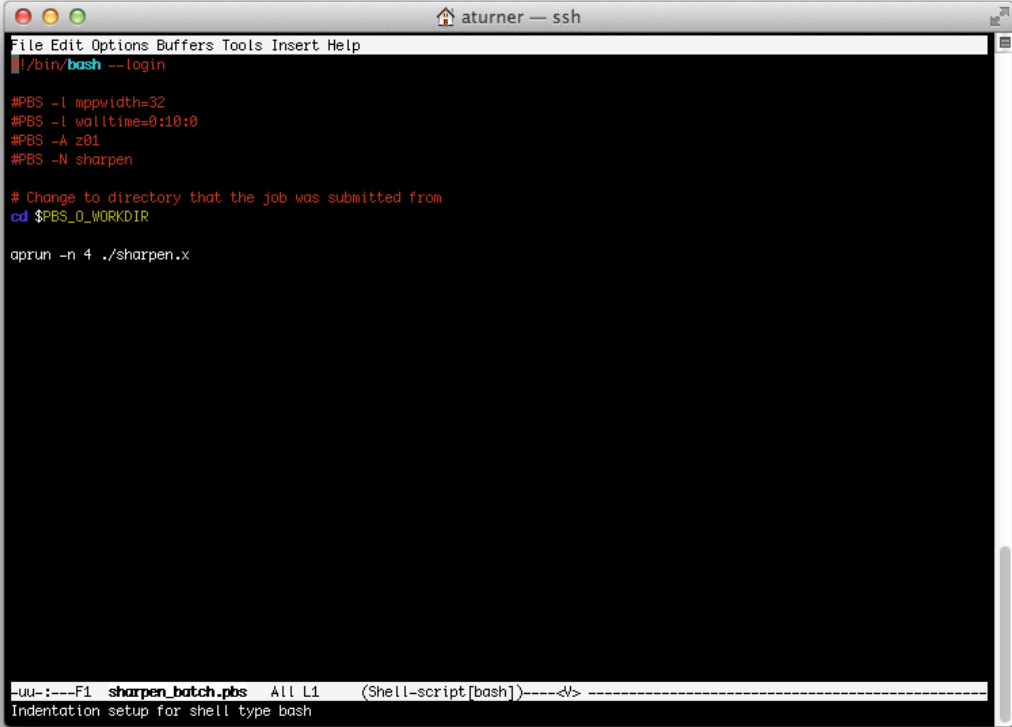
. Note: USE OF A LOGIN SESSION BY ANY PERSON OTHER THAN THE REGISTERED ACCOUNT
HOLDER IS CONTRARY TO THE TERMS AND CONDITIONS OF USE AND IS UNLAWFUL

NO USER MAY USE "SU" TO ACCESS THE ACCOUNT OF ANOTHER, OR ALLOW THEIR
ACCOUNT TO BE USED BY ANOTHER USER TO "SU" TO THAT USER'S ACCOUNT

-----
Phase3: CLE: 4.0,UP03 PBS: 10.2
-----
aturner@hector~xe6-8:~>
aturner@hector~xe6-8:~>
aturner@hector~xe6-8:~>
aturner@hector~xe6-8:~> 
```

Using HPC resources: File editing

- Editing files is often easiest using an “in-terminal” editor such as **emacs** or **vim**



```
aturner — ssh
File Edit Options Buffers Tools Insert Help
~/bin/bash --login

#PBS -l mppwidth=32
#PBS -l walltime=0:10:0
#PBS -A z01
#PBS -N sharpen

# Change to directory that the job was submitted from
cd $PBS_0_WORKDIR

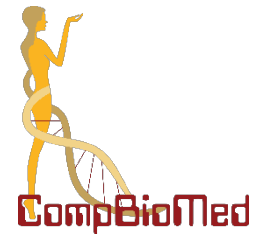
aprun -n 4 ./sharpen.x

--uu-:---F1 sharpen_batch.pbs All L1 (Shell-script[bash])---</>
Indentation setup for shell type bash
```

What is a batch system?

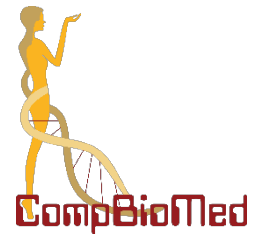
- Mechanism to submit your jobs to the compute nodes
 - control access by many users to shared computing resources
- Queuing / scheduling system for users' jobs
- Manages the reservation of resources and the job execution
- Allows users to “fire and forget” large, long calculations or a single large group of jobs
 - Called “production runs”
 - Typically not interactive
 - Typically no GUI

Why do we need a batch system?



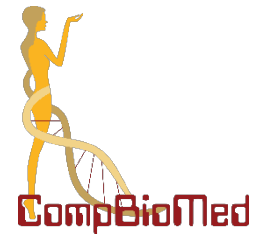
- To ensure the machine is utilised as efficiently as possible
- Ensure all users get a fair chance to use compute resources
 - demand often exceeds supply
- To track usage
 - accounting and budget control
- To mediate access to other resources
 - software licences, etc.

Reservation and Execution



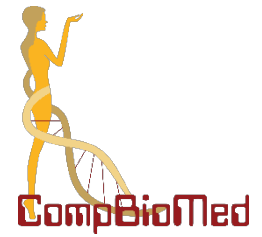
- When you submit a job to a scheduling system you specify the resources you require:
 - Number of cores, maximum job time, etc.
- The scheduling system then *reserves* a block of resources
- You can then use that block as you want, for example:
 - For a single job that spans all cores and the full time
 - For multiple shorter jobs in sequence
 - For multiple smaller jobs running in parallel

Batch system queues



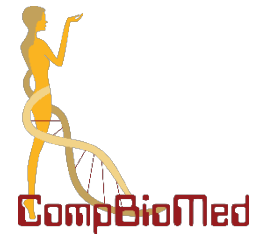
- Jobs contend for resources within their queue
- Queue - **logical** scheduling category, can correspond to:
 - Different time constraints
 - Special feature nodes (large memory, GPUs, etc.)
 - Nodes reserved for access by a subset of users
 - training courses
 - **Urgent Computing**
 - Surgical simulations, Modelling forest fires, etc.
- Some HPC systems choose your queue; others require to you name it explicitly

How to use a batch system



1. Write a job script specifying
 - Number of cores, maximum job time, etc.
 - Commands to run your calculations
2. Submit your job to the batch system
 - Job placed in a queue by a scheduler
 - Will be executed when there is space and time available
 - Job runs until
 - it finishes successfully, or
 - It is terminated due to errors, or
 - It exceeds the your specified time limit
3. Examine outputs and any error messages

Job script example – PBS



```
#!/bin/bash -login
#PBS -N Weather1
#PBS -l select=3:ncpus=36
#PBS -l walltime=1:00:00
#PBS -A d411
cd $PBS_O_WORKDIR
module load fftw
mpirun -n 108 weathersim
```

Linux shell to run job script in

Job name

Number of nodes and number cores per node

Requested job duration

Budget to employ

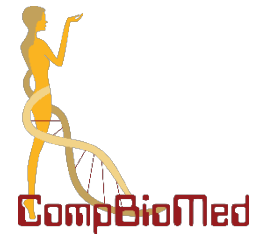
Changing to directory to run in

Parallel application launcher

Number of parallel instances of program to launch

Program name

PBS Batch system commands and job states



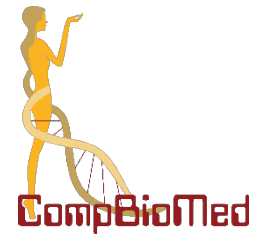
Job status	PBS
Job submit command	<code>qsub <batch_script.txt></code>
Job status command	<code>qstat -u \$USER</code>
Job delete command	<code>qdel <job_id></code>

PBS job state	Meaning
Q	The job is queued and waiting to start
R	The job is currently running
E	The job is currently exiting (not error)
F	The job is finished (not failed)
H	The job is held and not eligible to run

Using HPC Systems

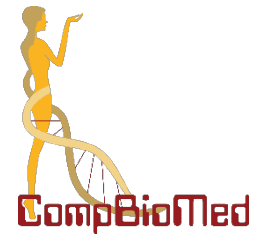
Example: how many cores should I employ for my simulation?

Why care about parallel performance?



- Why do we run applications in parallel?
 - so we can get solutions more quickly
 - so we can solve larger, more complex problems
- If we use 10x as many cores, *ideally*
 - we'll get our solution 10x faster
 - we can solve a problem that is 10x bigger or more complex
 - unfortunately this typically not the case...
- Measuring parallel performance can help us understand
 - whether an application is making efficient use of many cores
 - what factors affect this
 - how best to use the application and the available HPC resources

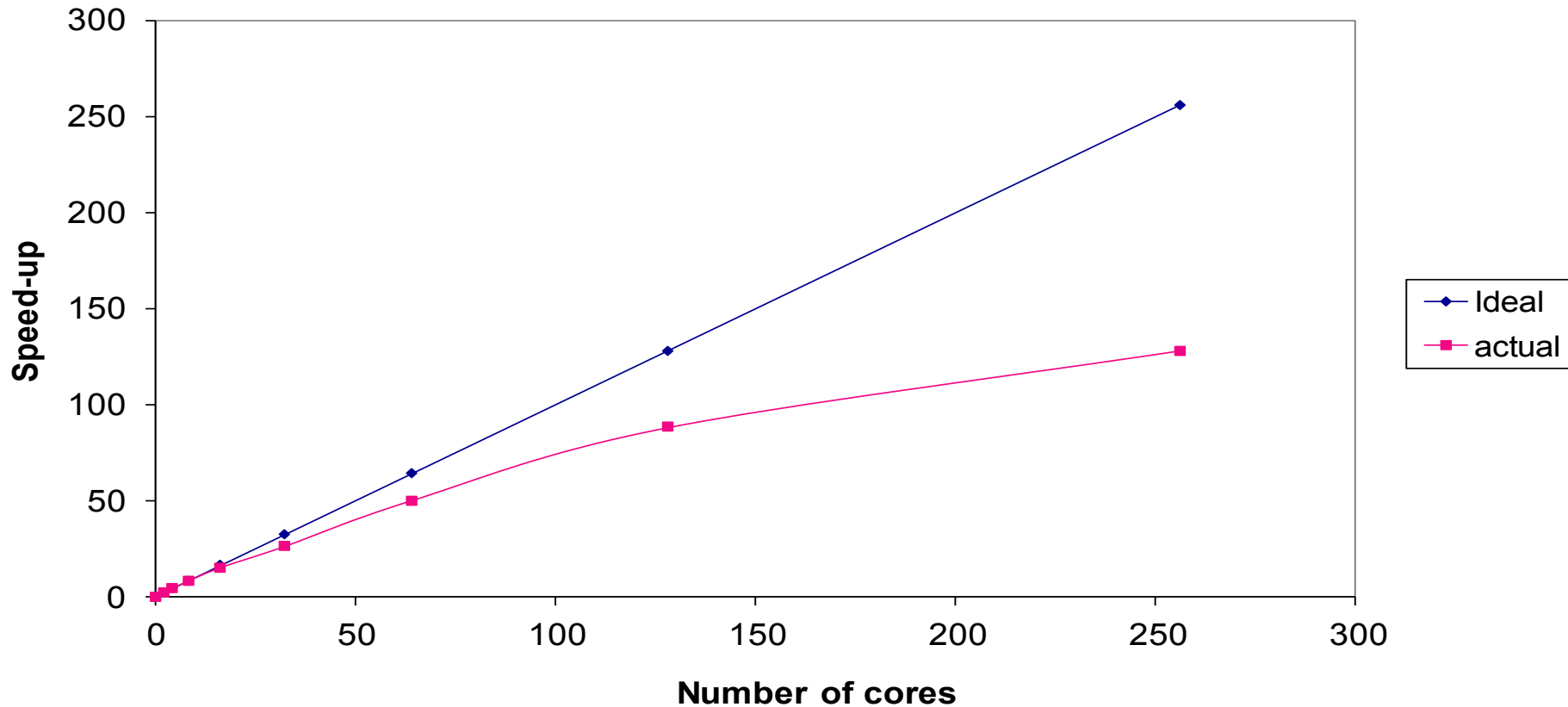
Performance Metrics



- How do we quantify performance when running in parallel?
- Consider **execution time** $T(N,P)$ measured whilst running on P cores with problem size/complexity N
- **Speedup:**
 - $S(N,P) = T(N,1)/T(N,P)$
 - Time on one core divided by time on P cores
- **Parallel efficiency:**
 - $E(N,P) = S(N,P)/P$
 - Speedup on P cores divided by P

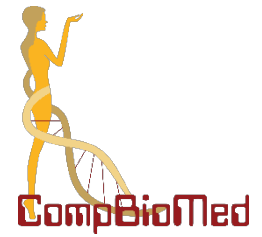
Typical strong scaling behaviour

Speed-up vs Number of cores



- **Scaling** describes how the runtime of a parallel application changes as the number of processors is increased
- **Strong Scaling** (increasing C , constant N)
 - problem size/complexity stays the same as the number of cores increases, decreasing the work per core
- **Weak Scaling** (increasing C , increasing N)
 - problem size/complexity increases as the number of cores increases, keeping the amount of work per core the same.

I have a particular simulation to run: how many cores should I employ?

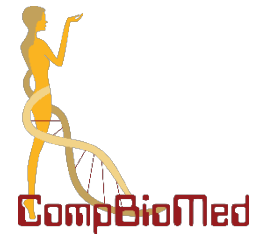


- Let's say you have access to a target HPC machine with, say
 - 500 nodes, with 32 cores per a node
- Can I use all the cores in the 500 nodes?
 - Code might run 16,000 times faster!?
- Depends on the package running the simulation
 - Is it parallelised using threads only?
 - Limited to a single node with its 32 cores
 - Is it parallelised using MPI?
 - Can use all 16,000 cores
- But, how many cores to use?
 - We need to benchmark!?

How to benchmark? (1/2)

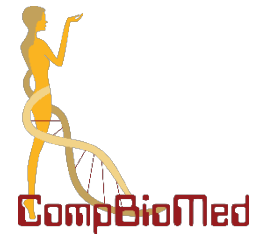
- First, pick a **typical** target simulation
- **Large enough** to reflect the **characteristics** of your **target simulation**
- **Small enough** not to burn too much resources
 - Aim for a simulation which does not run faster than 1 second
 - To avoid OS flutter
 - But not longer than, say, 10 hours
 - So we don't waste cycles *not* producing science

How to benchmark? (2/2)



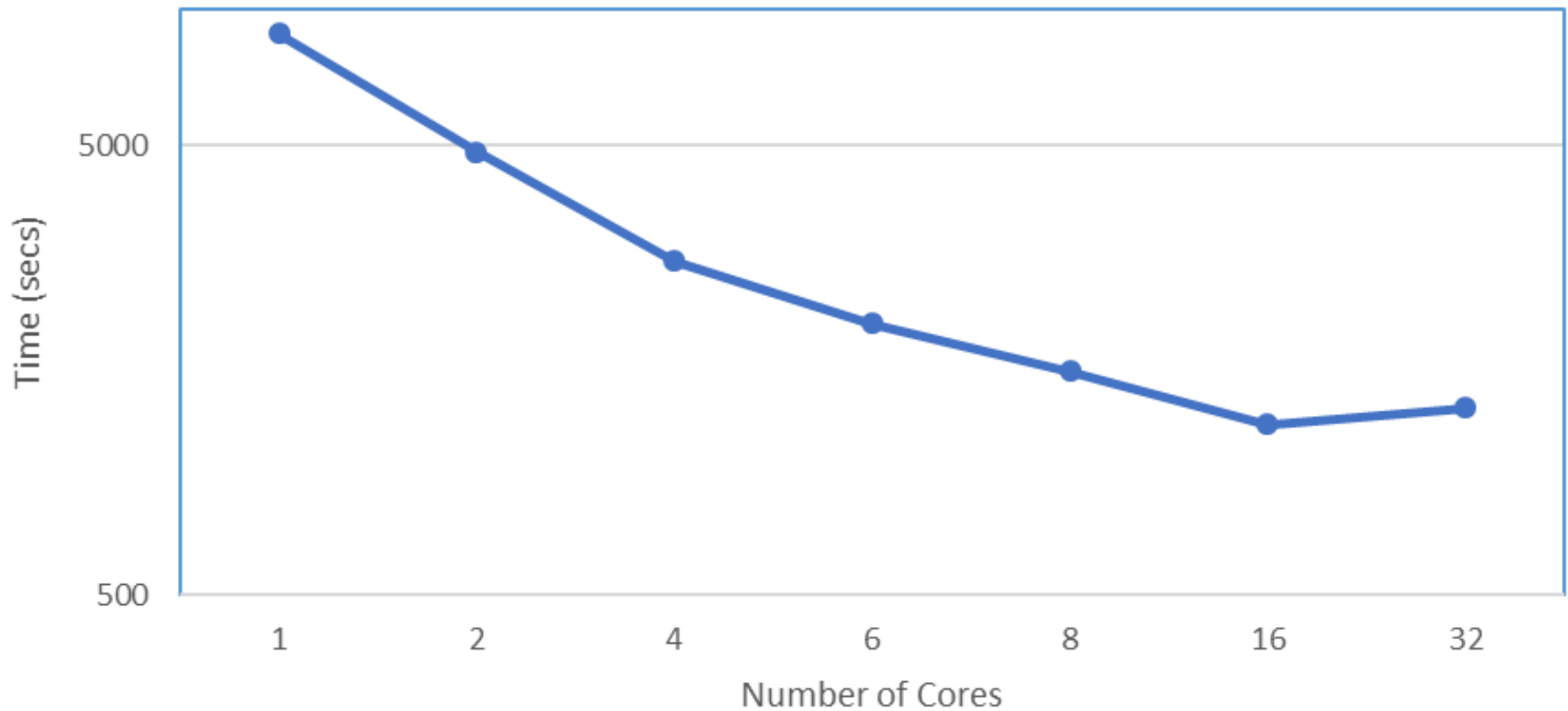
- All benchmarks should be run multiple times
 - Typically one MPI task and/or one thread per core is best
- Exclusive node use (only you have access to the nodes)
 - **Run three times and take minimum**
 - Measures the fastest times
- Shared node use (you share access to the nodes)
 - **Run 10 times** and record associated statistics
 - minimum, maximum, mean and 95% confidence limits
 - Measures how “busy” the machine can be

Benchmarking Example

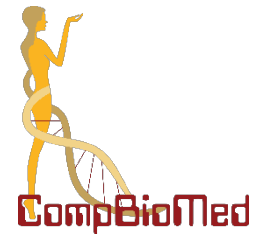


- QIIME™ is an open-source bioinformatics pipeline for performing microbiome analysis from raw DNA sequencing data
 - Quantitative Insights Into Microbial Ecology.
 - pronounced chime
 - www.qiime.org
- Python scripts, parallelised using threads

Example execution times for a particular Qiime routine

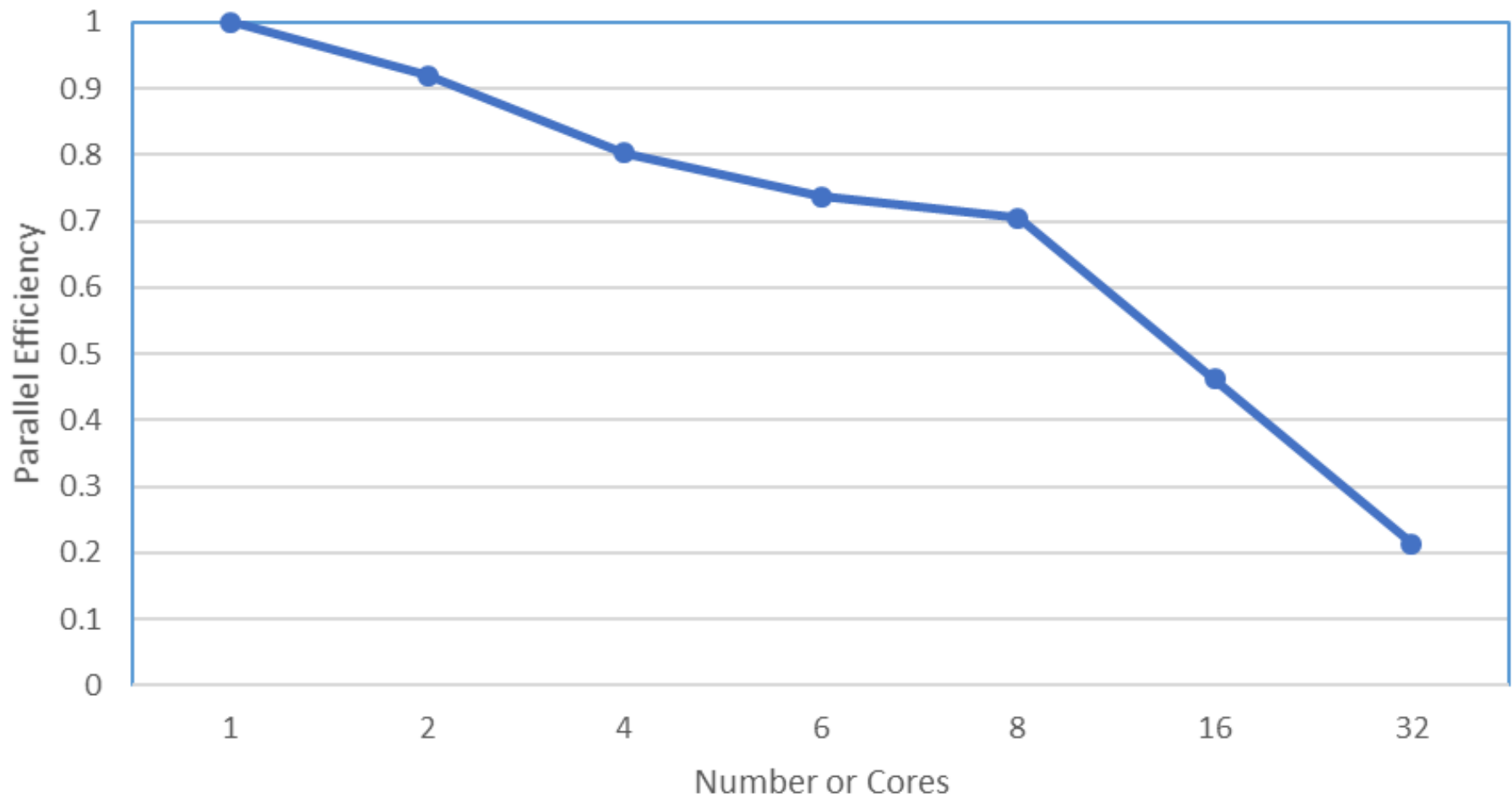


Execution times discussion



- Typical Execution Times graph
 - Log/log graphs present data clearly
 - As the core count increases: time reduces, levels out and then starts increasing
 - Communication starts to dominate over computation
- Execution times alone shows that 16 cores is the fastest
 - but is that an efficient use of the cores?
 - what about the Parallel Efficiencies?

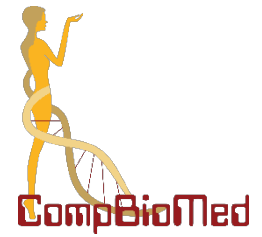
Parallel Efficiencies for Qiime on Cirrus



Discussion of Parallel Efficiencies

- A typical target parallel efficiency is 70%.
 - Some groups use 50%, others use 80%.
 - Using 70% gives the target number of cores as 8
- The efficiency for the **fastest** execution time
 - 16 cores has ~46%
 - Very poor efficiency
- The efficiency when using all the cores in one node?
 - 32 cores has ~21%
- Parallel Efficiency plots often reveal much more than execution times alone
 - i.e. “jump” from 8 to 16 cores may be due to NUMA region affects

So is the target core count for our example is 8?

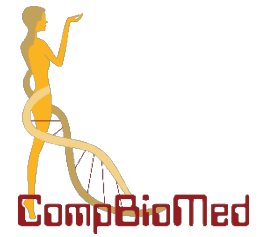


- Most efficient core count is 8 for this example
 - **16 may be faster but it is not efficient**
 - Do not waste your project's shared time budget
 - “cycles”
- However, some HPC platforms charge you by the node
 - We would pay for all 32 cores
 - Fastest result now wins
 - 16 cores is best choice
 - NB further testing might show 24 cores is even faster

HPC Jargon (1/2)

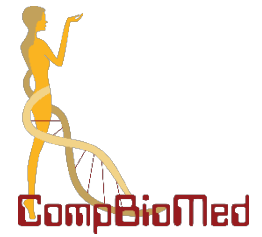
- HTC: High Throughput Computing
 - Task Farms on HPC
- HPDA: High Performance Data Analysis
 - Data analysis performed on huge nodes
- Network
 - Interconnect, Communications Fabric, hardware used to send messages between cores on different Nodes
- Nodes
 - Servers, Shared Memory System, Boxes
- Processors
 - Sockets, CPUs
- Cores
 - CPUs

HPC Jargon (2/2)



- Memory hierarchy
 - Cache, L1, L2, L3, NUMA regions: placing groups of tasks within particular memory regions affect performance
- Scripts
 - ASCII files containing commands
 - Batch scripts: batch system, operating system, executables, etc.
- Time budget
 - Cycles, core hours, CPU hours, CPUh
- Production Runs
 - Simulations which are not tests but are producing science
- Efficiency
 - How fast a task is compared to a single core
 - Sometimes a group of cores or a single node if simulation is huge
- Scaling
 - Particular **simulations** with a fixed size are said to ‘scale’ if they remain efficient as the number of cores becomes larger

HPC Summary



- HPC driven by need for more computing power
- HPC is synonymous with **parallel computing**
 - Parallelism is available at many levels
 - Parallel computing is now expanding to all computers
 - **Good scaling** to large numbers of tasks is difficult
- Access is still usually via **command line**
 - Need to learn to use an in-terminal editor
- Access to compute resources mediated by **batch** job submission system
- **Benchmark** to determine how many nodes/cores to use

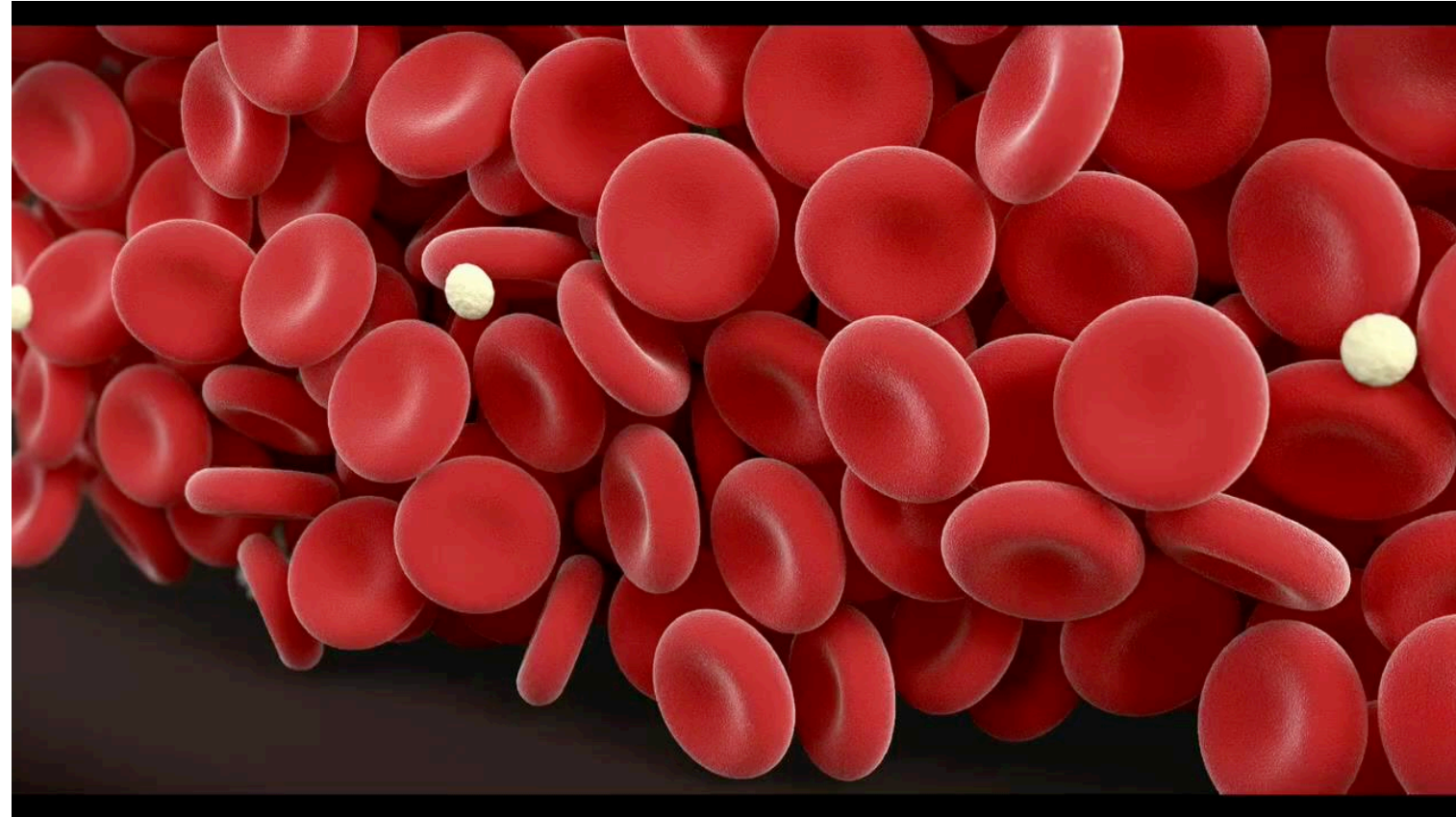
Part II - Overview



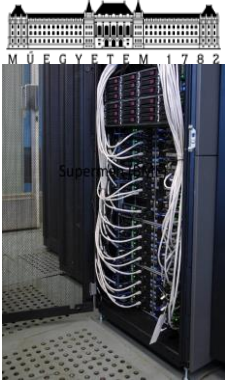
- Meet an HPC code and an HPC machine
 - HemoCell
 - Lisa @ SurfSARA
- How to set up the development environment
 - What makes it different from working on your laptop?
 - Obtaining the source
 - File transfer
 - Module system in a nutshell
 - Compilation
- Executing a simulation
 - Development short-runs on a login node
 - Queueing system in practice (PBS)
 - Handling multiple jobs
- Evaluate the output
 - Parallel output and how to handle it
 - Post-processing large datasets
 - Visualize information of interest

HemoCell - A high-performance framework for dense cellular suspension flows

- www.hemocell.eu
- Open-source code (AGPLv3)
- Fully validated
- Suitable for high hematocrits
- Suitable for high shear-rates
- Three dimensional version
- Two dimensional version
Higher performance / larger domains



Deployed on several HPC systems



Superman (BME, Budapest)



Supermuc (LRZ, Munich)



Sanam (KACST)



Marenosturm (BSC, Barcelona)



Cartesius and Lisa (SurfSARA, Amsterdam)



Aspire I (NSCC, Singapore)

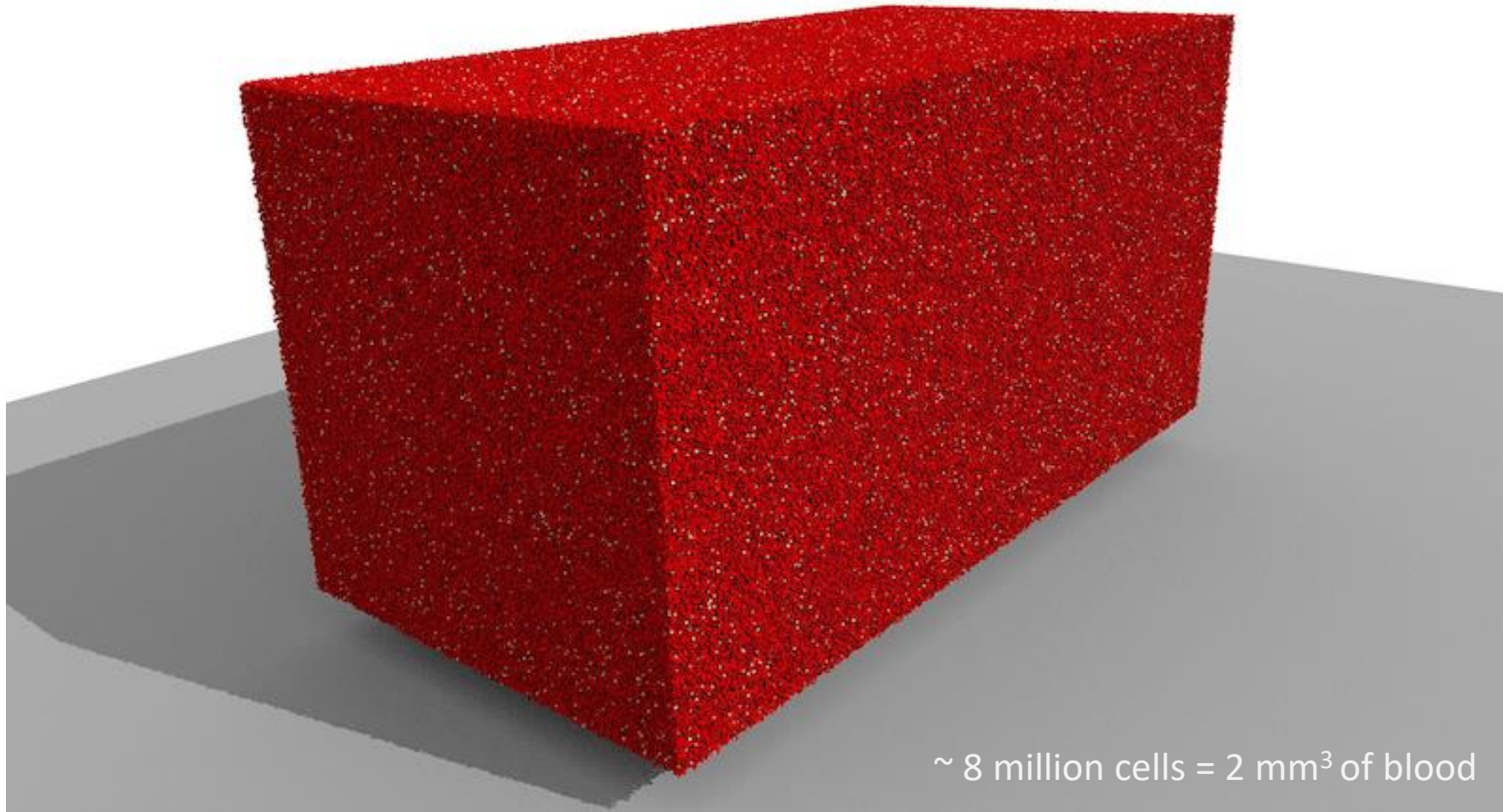


Lomonosov (MSU, Moscow)



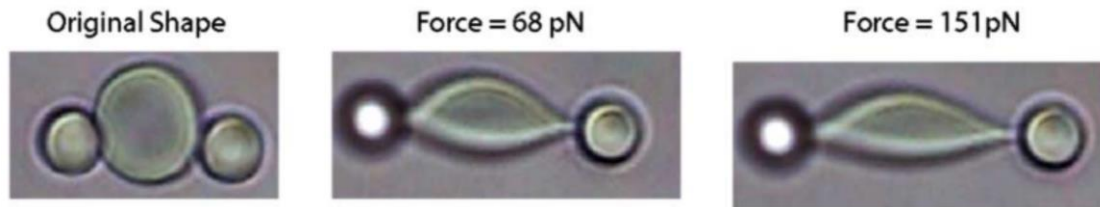
Eagle (PSNC, Poznan)

HPC allows for large domains!



Simulation cases

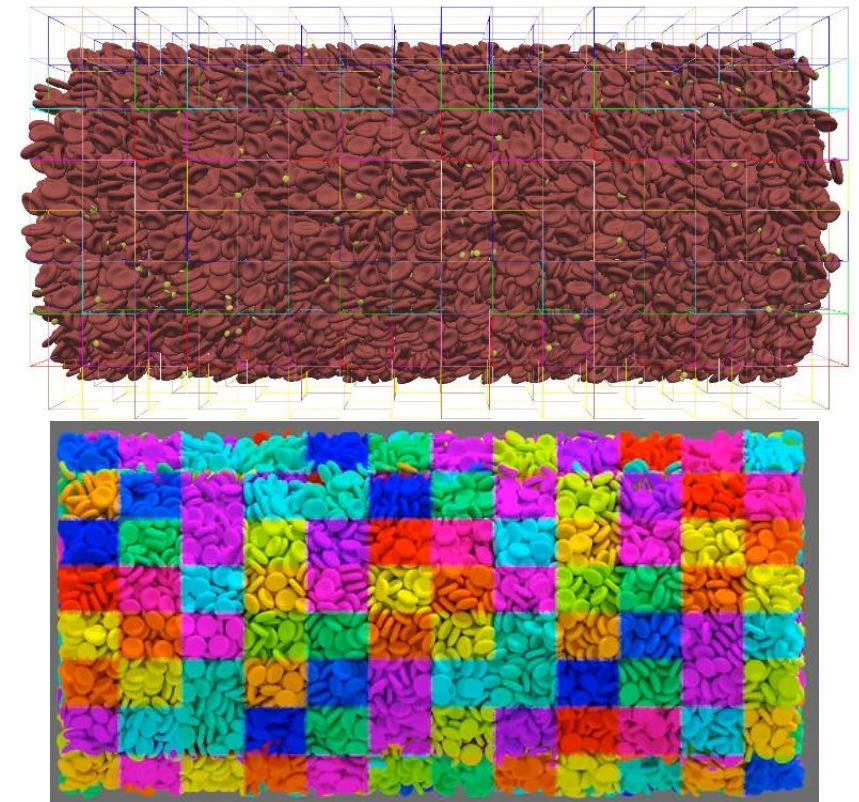
Case #1



Mills et al., *Mech. Chem. Biosyst.*, 2004.



Case #2

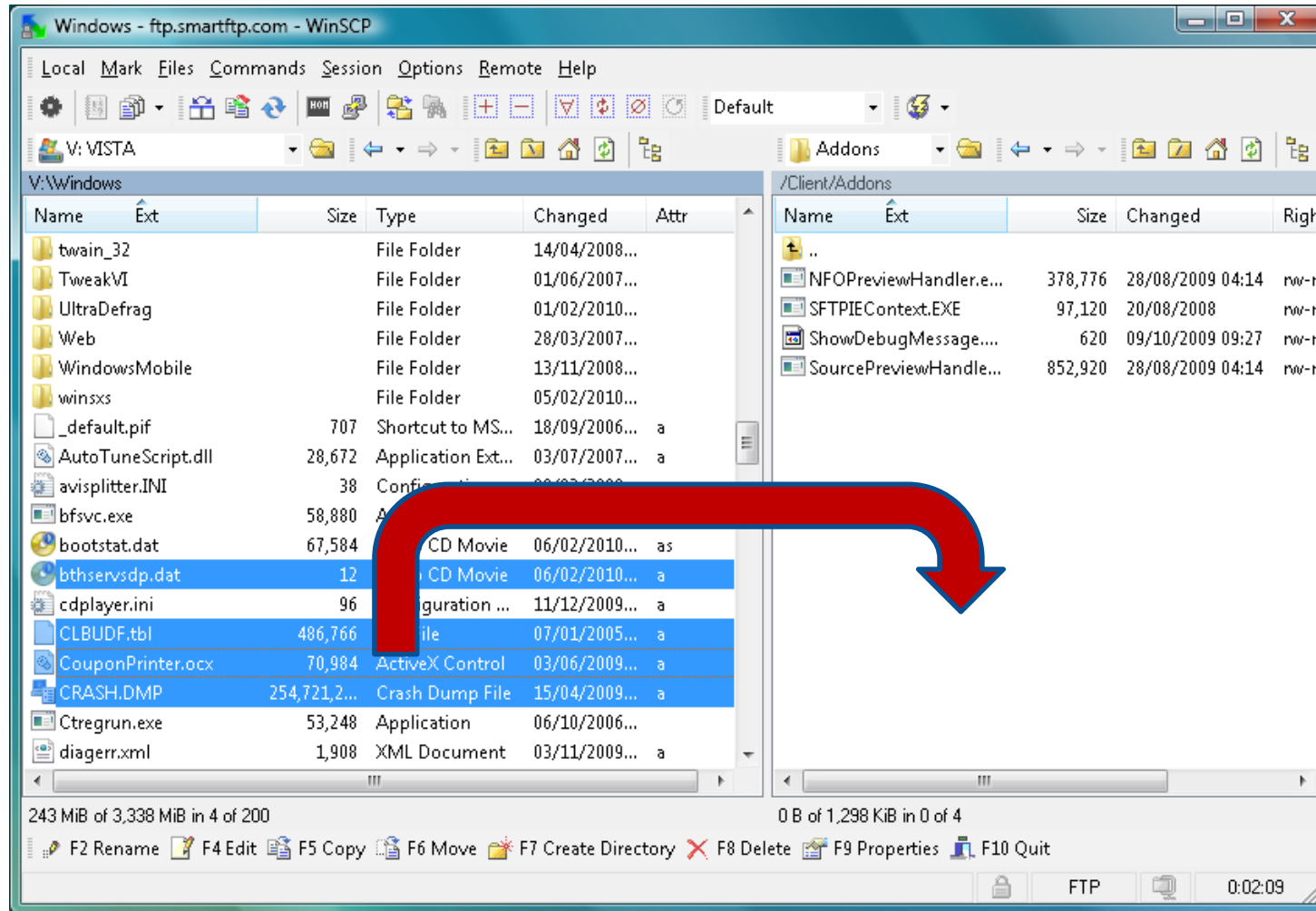


Suggested development environment



- Suggested environment - make sure you have an app. installed for every category:
 - **Terminal + ssh:** putty / termius / ssh / ...
 - **File transfer:** winscp / cyberduck / mc (shell connection) / scp / totalcmd /...
 - **Editor:** SublimeText / vim / emacs / Notepad++ / joe / nano / ...
 - **Visualizer:** Paraview 5.5 / (VisIt) / ...

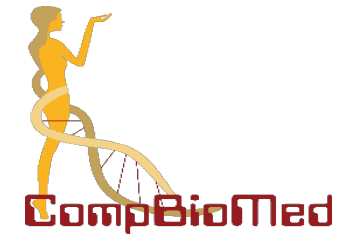
Transfer files (scp, sftp)



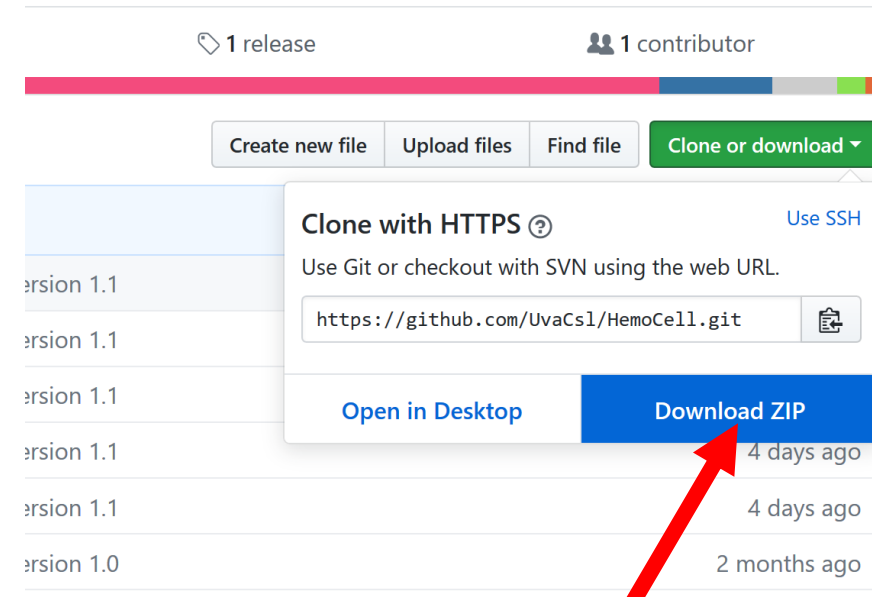
Tip: It can be convenient to set up a local file editor in your scp client.

This way you can edit your files locally with your favourite editor, and they are automatically uploaded when you save.

Obtain HemoCell



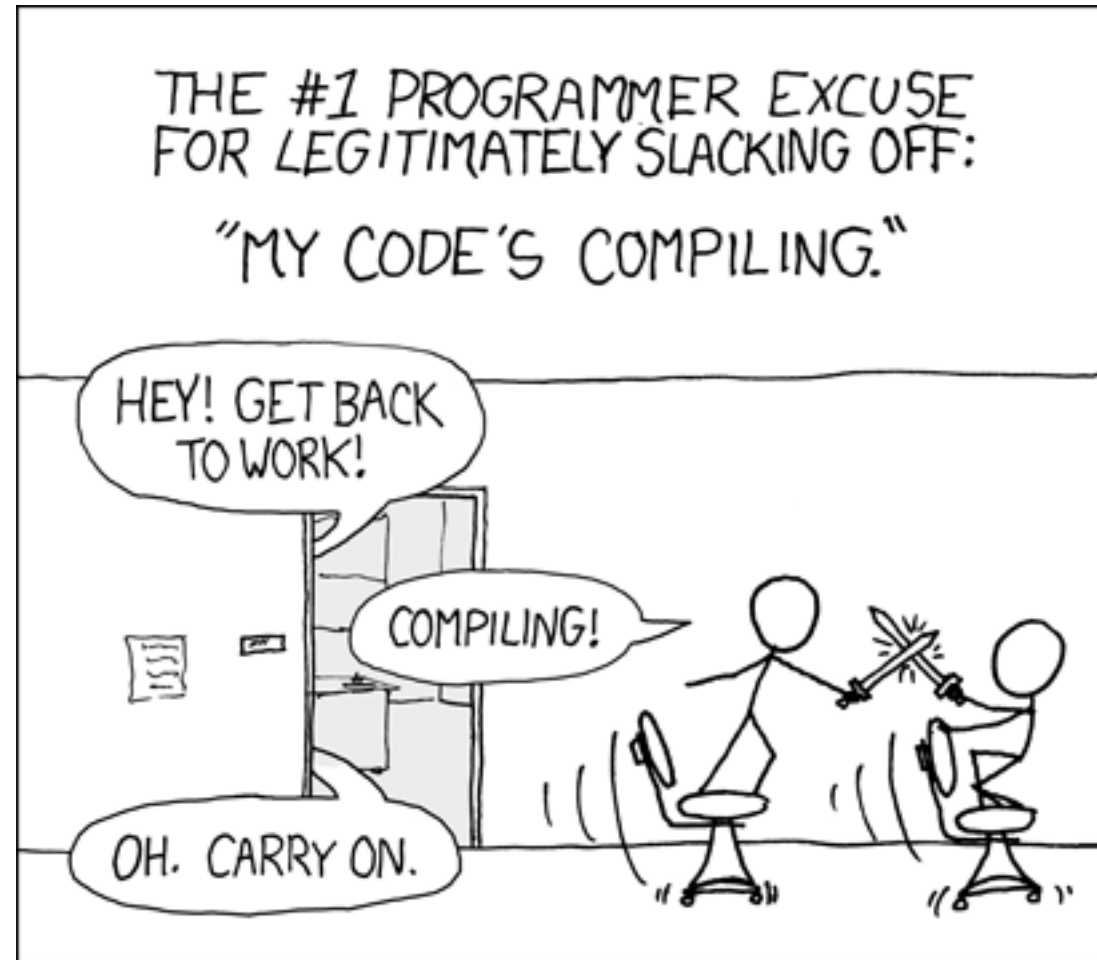
1. Go to www.hemocell.eu
2. Navigate to 'Docs & Downloads'
3. Click the 'Code repository' button
4. Download zip archive from GitHub
 - a) Direct download
 - b) Copy URL and use download manager / wget
5. Unzip it
6. Run 'setup.sh' Note: never do something like this blindly!
7. Compile a simulation case



Terminal session #1



Let's discuss the next phase while the code compiles!



Source: <https://xkcd.com>

Execution

- Small simulations (such as "stretchCell") can be developed and executed locally on your notebook:
- Larger simulations (such as "pipeflow") can be developed locally, but the execution requires a larger machine:



“Test – develop” cycle

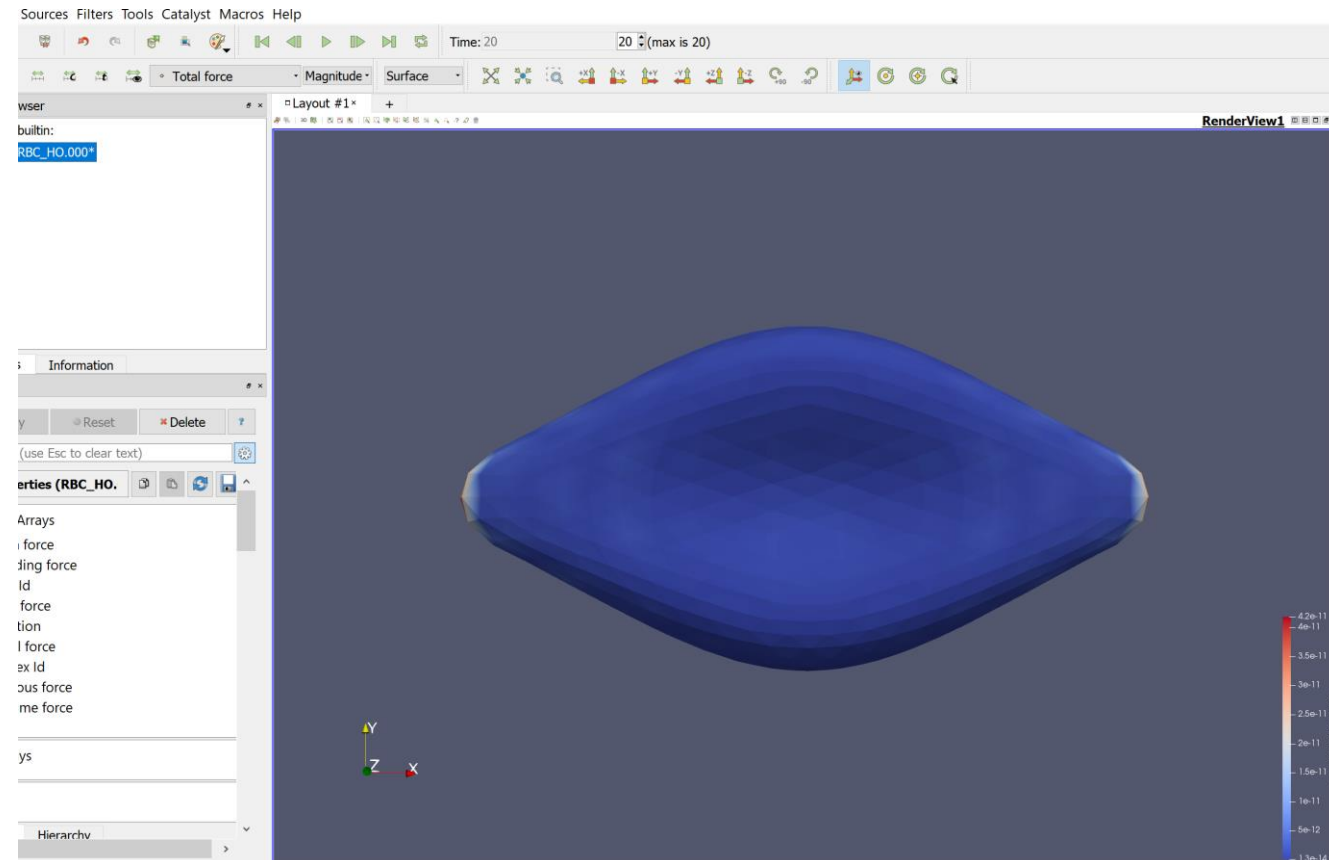
- Testing the code if it start up might be possible on the **login node**.
- Better to submit it to a designated “**short**” **queue**.
- Alternatively, request an **interactive session**.
- When you are convinced everything is in order, submit a larger run!
- You can submit multiple jobs and use command-line tools to manage them.
- After the execution of the simulation run post-processing:
 - Lightweight post-processing locally on your computer
 - File-operation heavy through the login node
 - Computation heavy as a separate job

Terminal session #2

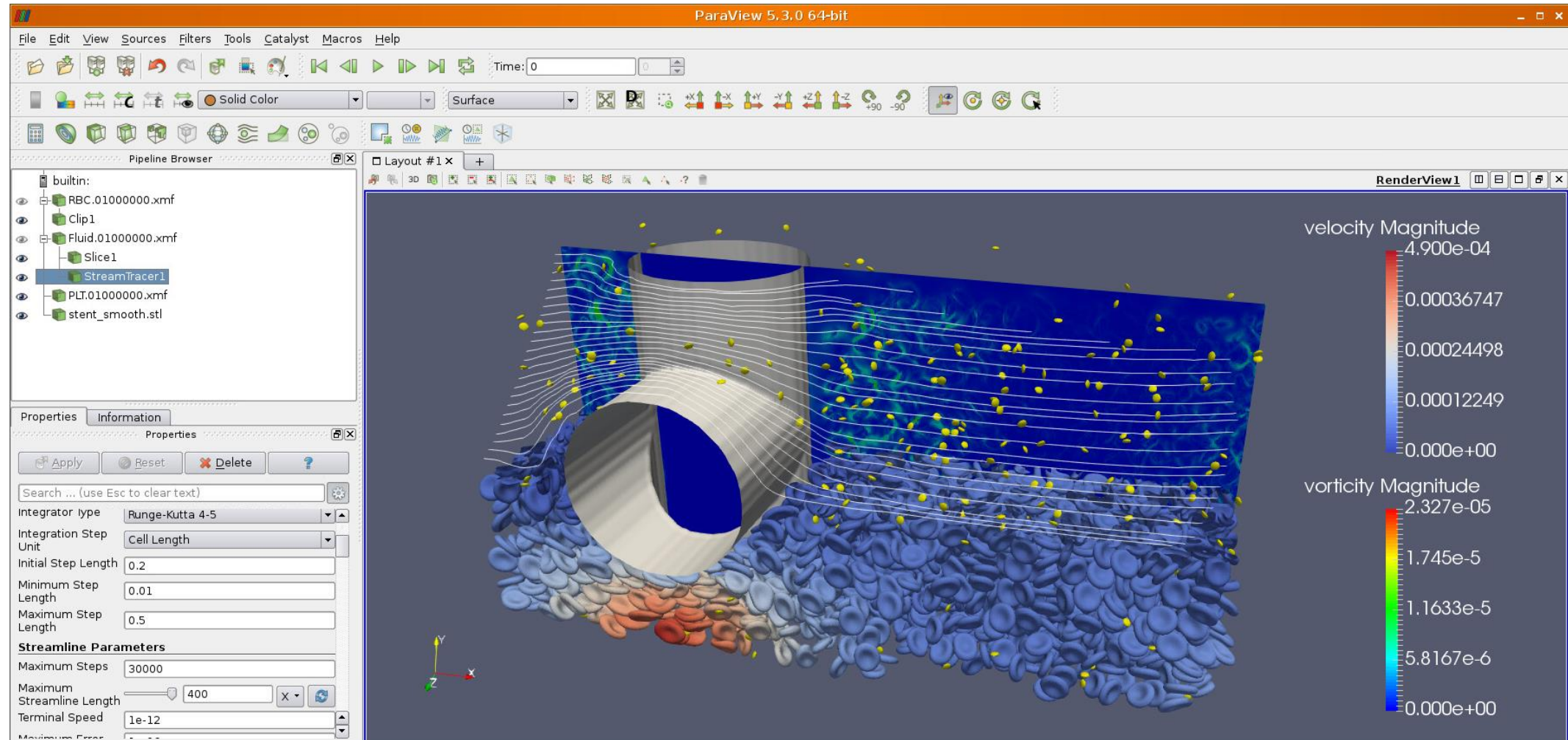


Let's check the results!

1. Open Paraview
2. File -> Open File -> navigate to '*tmp*' folder of the simulation
3. Select RBC_HO...xmf
4. Select 'Xdmf Reader' and NOT Xdmf3!
5. On the 'Properties' panel to the left press 'Apply'
6. Put colouring to 'Total force'
7. Animate with the 'Play' button



Paraview session





Q&A

To pose a question, you can write your question
in the “Questions” tab



Thank you for participating!

Visit the CompBioMed website (www.compbioMed.eu/training)
for a full recording of this and other webinars,
to download the slides
and to keep updated on our upcoming trainings

