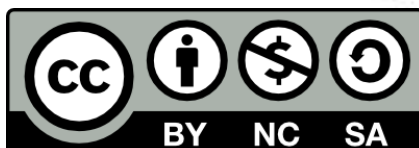




CSC Autumn School in Computational Physics 2014





All material (C) 2014 by CSC – IT Center for Science Ltd. and the authors.
This work is licensed under a **Creative Commons Attribution-NonCommercial-ShareAlike 3.0**
Unported License, <http://creativecommons.org/licenses/by-nc-sa/3.0/>

CSC Autumn School in Computational Physics 2014

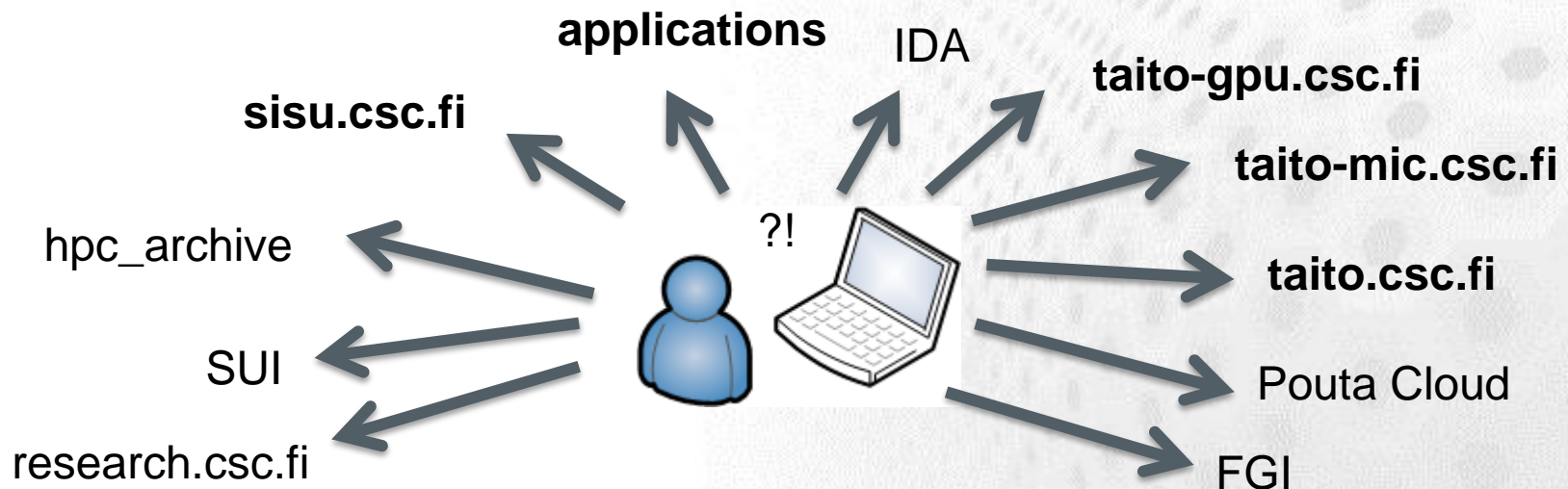
Monday December 8		Tuesday December 9
9.00-10.15	Intro, Physics@CSC Round robin (T. Malkiewicz)	Unix for physicists (J. Lento)
10.15-10.45	Coffee break	Coffee break
10.45-12.00	Parallel computations (T. Malkiewicz)	Advanced unix for physicists (J. Lento)
12.00-13.00	Lunch	Lunch
13.00-14.30	Applications for physicists (J. Enkovaara)	Computational physics with Xeon Phi and GPU (F. Robertsén)
14.30-14.45	Coffee break	Coffee break
14.45-15.45	Coding (J. Åström)	Massively parallel computations (J. Åström)
15.45-16.30	Debugging and code optimization (T. Malkiewicz)	Scientific visualization (J. Hokkanen)

+ *supercomputers' guided tour* on Tuesday at 12:40

Learning targets



- Know what CSC has to **offer** for physicists and which servers (resources) to use
 - *Applications*
 - *Data processing and visualization*
- Be able to **use**/run efficiently on **Taito** and **Sisu**
- Be able to **use** Bull (**taito-gpu** and **taito-mic**)



Practicalities



- Keep the name tag visible
- *If you came by car: parking is being monitored - ask for a temporary parking permit from the reception (tell which school you're participating)*
- Lunch is served in the same building
- Toilets are in the lobby
- Visiting outside: doors by the reception desks are open
- Room locked during lunch
 - lobby open, use lockers
- Network:
 - WIFI: eduroam, HAKA authentication
 - Ethernet cables on the tables
 - CSC-Guest accounts upon request
- Username and password for *workstations*: given on-site

How we run this school



- Rather lecture than conference-oriented presentations
 - Try to make potentially difficult things look relatively easy to learn and understand
 - Skip items that have less significance in everyday work of physicists
- Demos
- A hands-on sessions included in most lecture session
 - practice the just learned subjects



Physics at CSC

Content

- Physics at CSC's supercomputers
- Resources available for physicists
 - What's new
 - Future
- Why and when to use supercomputers
- Courses of interest for physicists
- Physics' people at CSC

CSC at glance



- Founded in 1971
- Owned by Ministry of Education and Culture
- Operates on a *non-profit* principle
- Staff ~265 people
- Facilities in Espoo and Kajaani
- **Free of charge services for higher education institutions in Finland**



Physics at supercomputers



Physics is a branch of science concerned with the nature, structure and properties of matter, ranging from the smallest scale of atoms and sub-atomic particles, to the Universe as a whole.

Physics includes *experiment* and *theory* and involves both fundamental research driven by curiosity, as well as applied research linked to technology.

EPS report, 2013

Supercomputer is a computer at the frontline of contemporary processing capacity – particularly speed of calculation.

Fastest supercomputer: China Tianhe-2 with 33.86 petaFLOP/s (quadrillions of calculations per second) on the LINPACK benchmark

Currently available computing resources

CSC

➤ Sisu

- 40 512 cores, 64 TB memory

➤ Taito

- Small and medium-sized tasks

➤ Application server Hippu

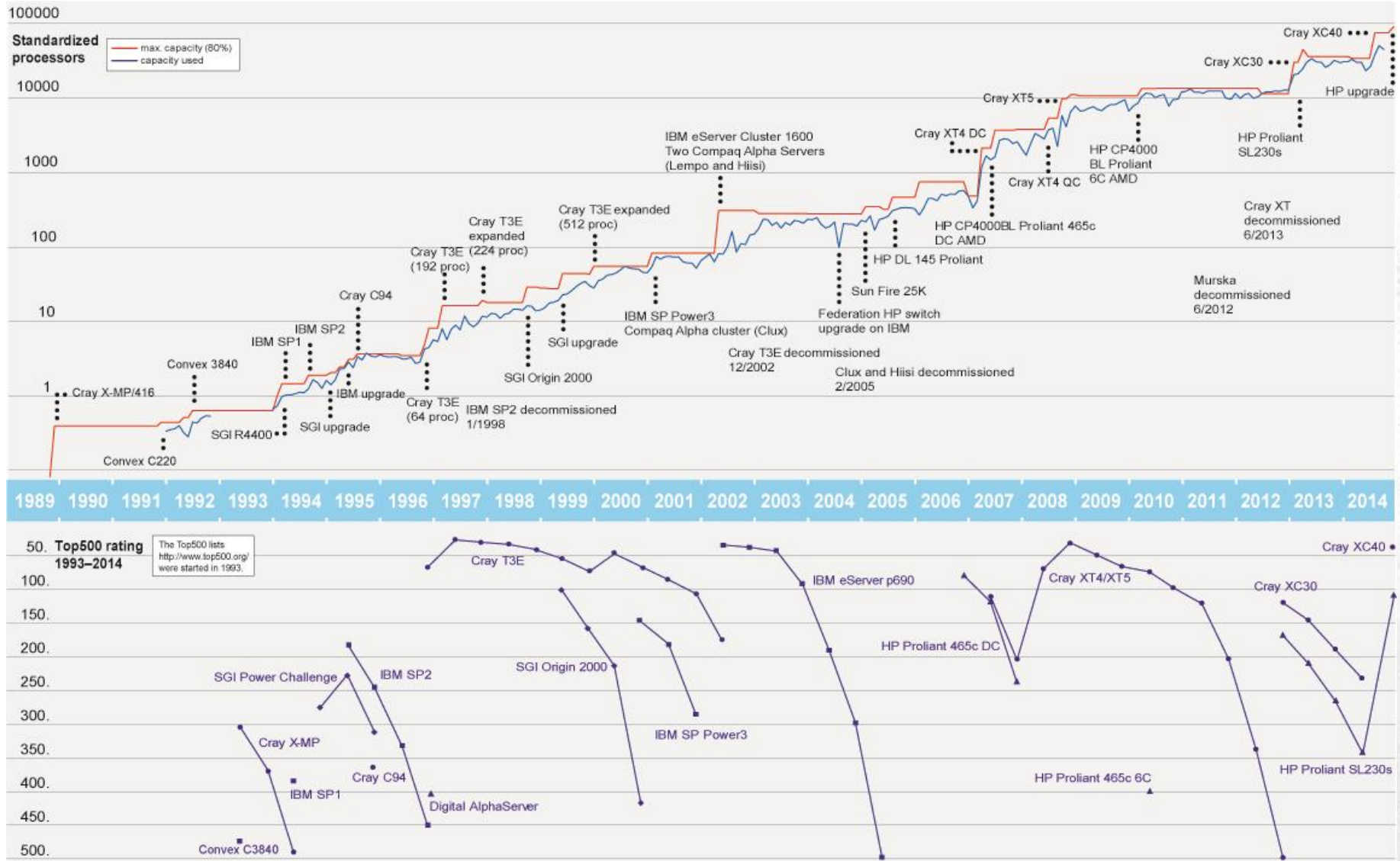
- Going to be decommissioned by the end of 2014
- Replaced by Taito-shell

➤ FGI

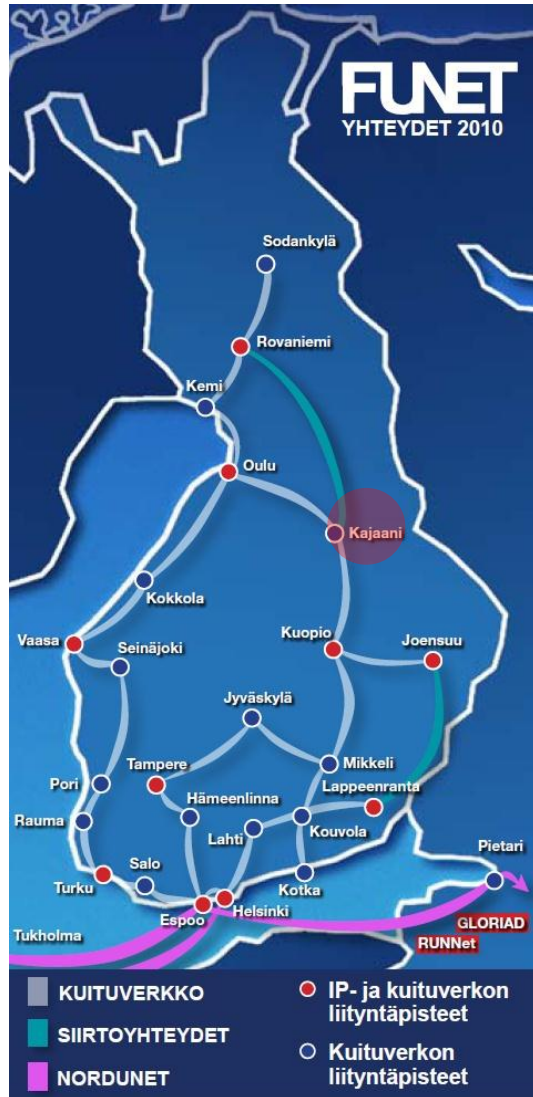
➤ Cloud

➤ Bull system

CSC Computing Capacity 1989–2014



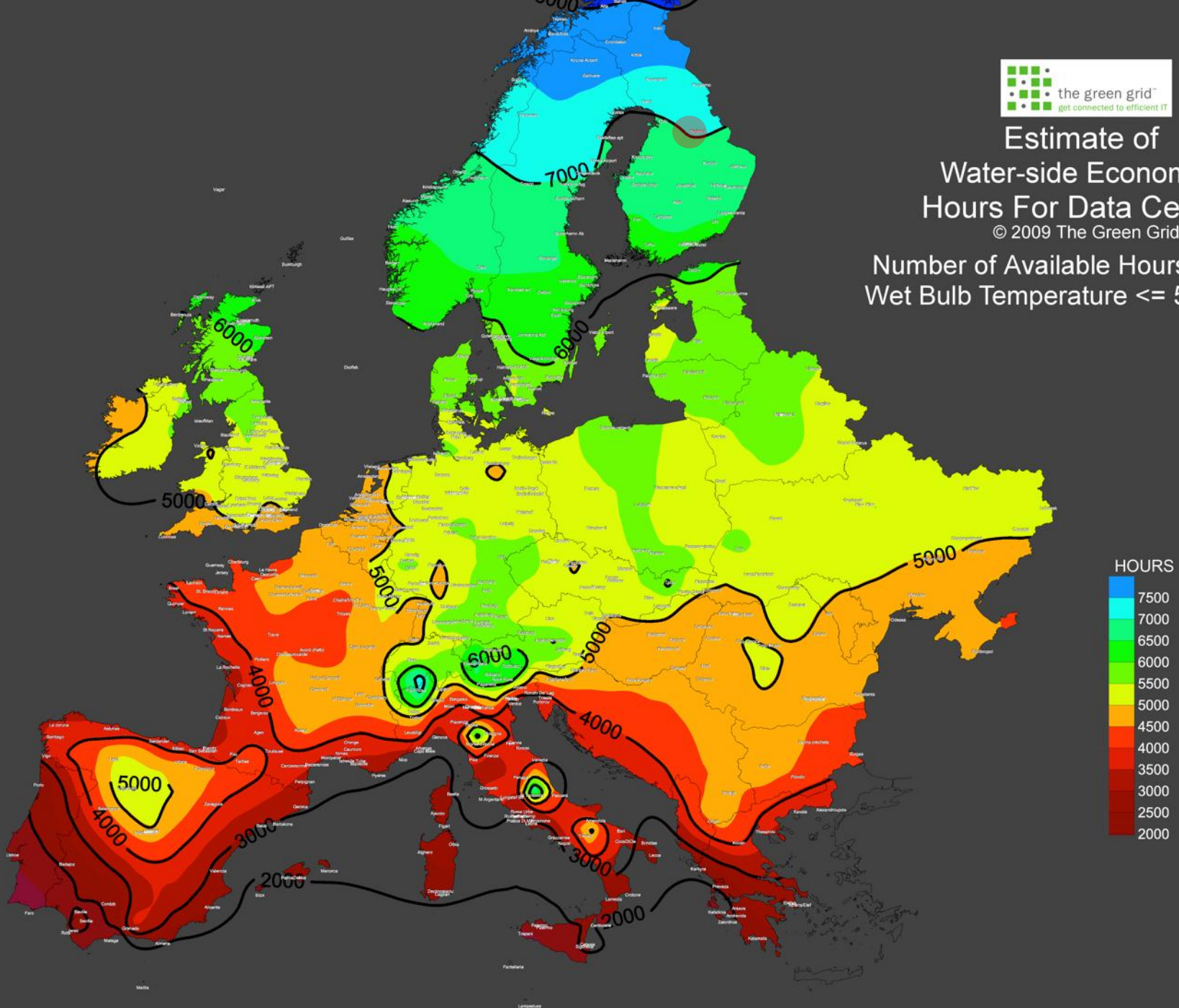
Datacenter CSC Kajaani



Estimate of Water-side Economizer Hours For Data Centers

© 2009 The Green Grid

Number of Available Hours Where:
Wet Bulb Temperature \leq 50F (10C)





Users

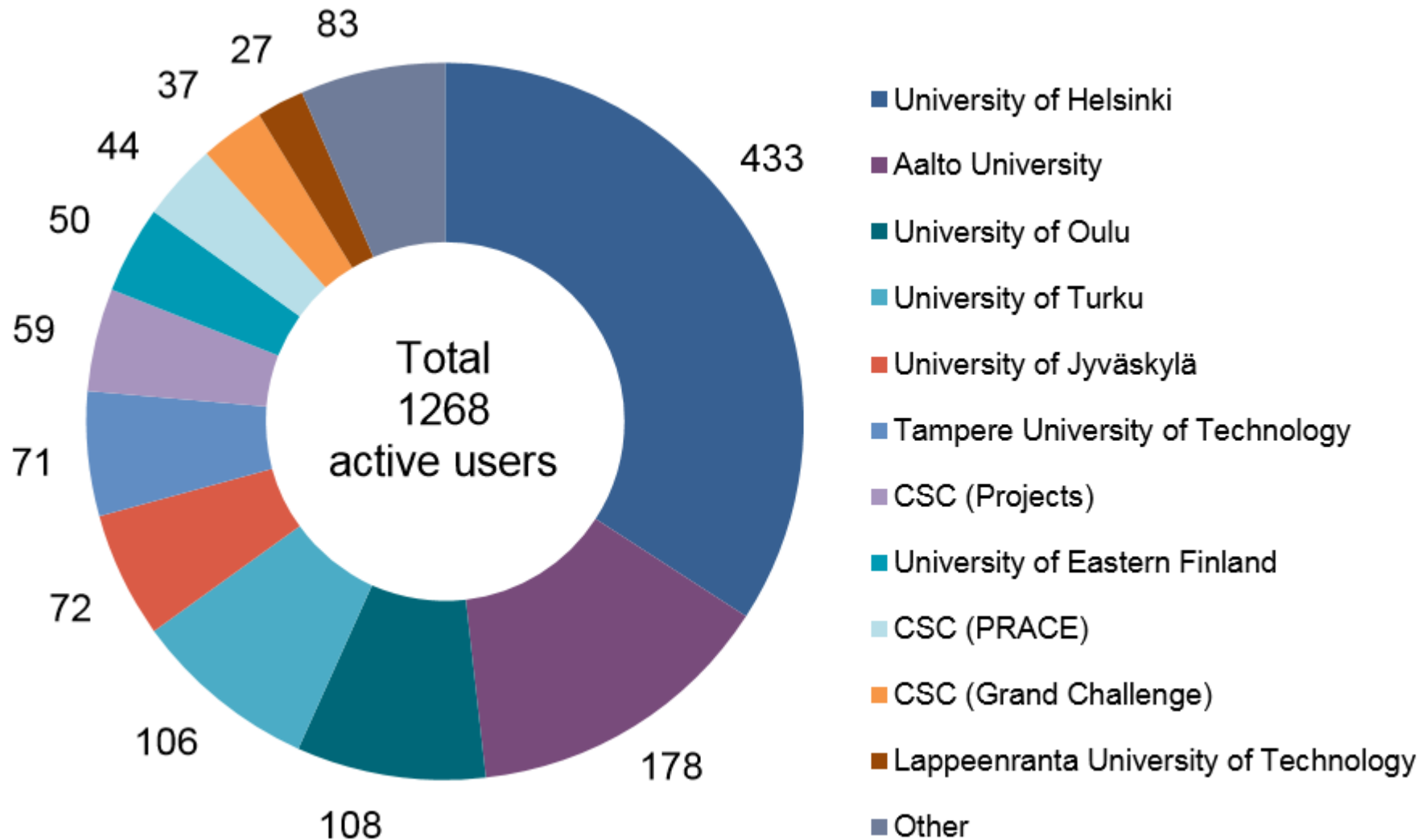


- About 700 active computing projects
 - 3000 researchers use CSC's computing capacity
 - 4250 registered customers
- Haka-identity federation covers all universities and higher education institutes (287 000 users)
- Funet - Finnish research and education network
 - Total of 370 000 end users

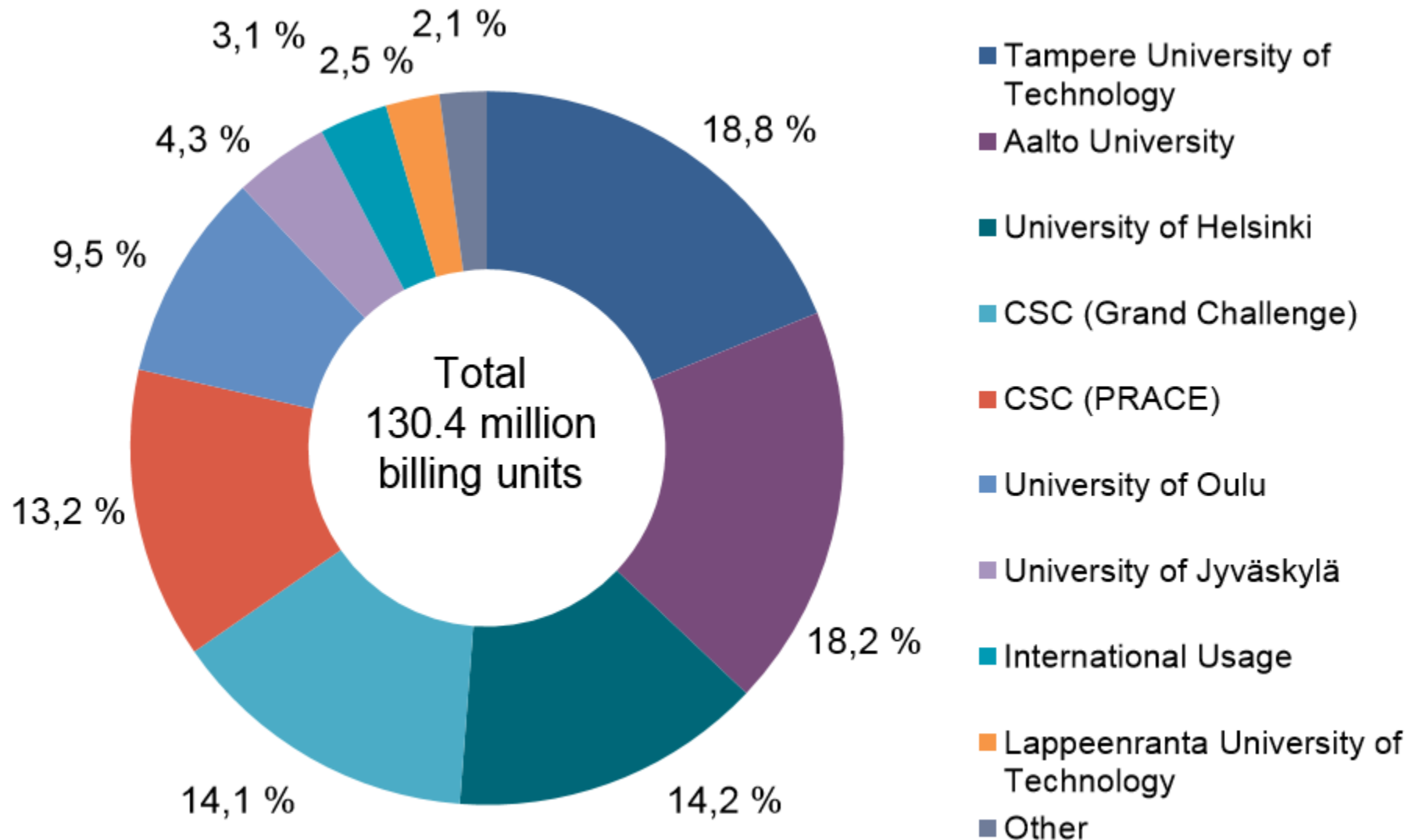


Users of computing resources by organization

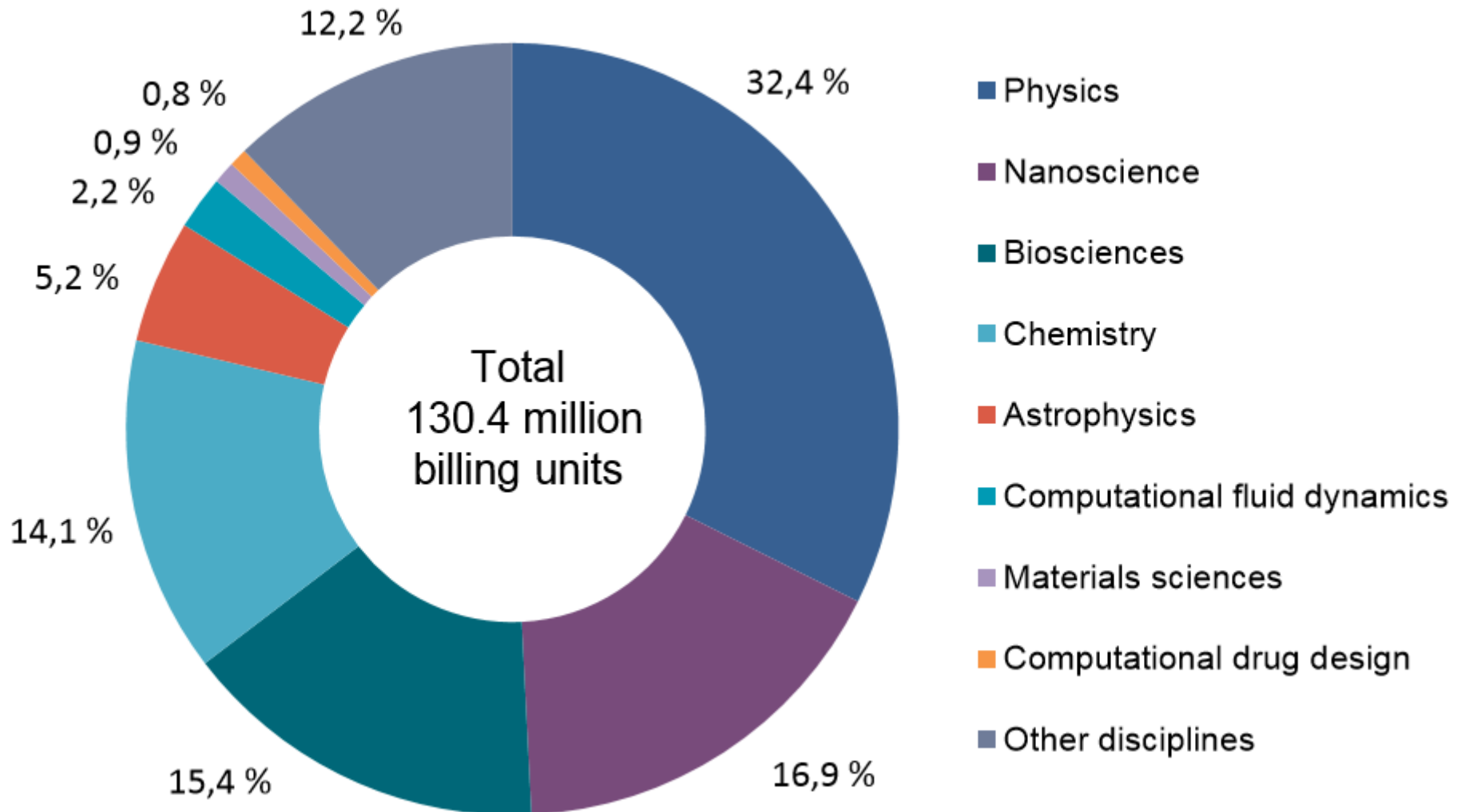
1H2014



Computing usage by organization 1H2014



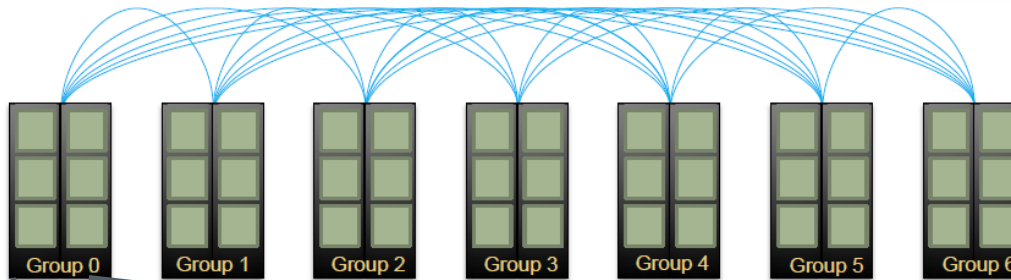
Computing usage by discipline 1H2014



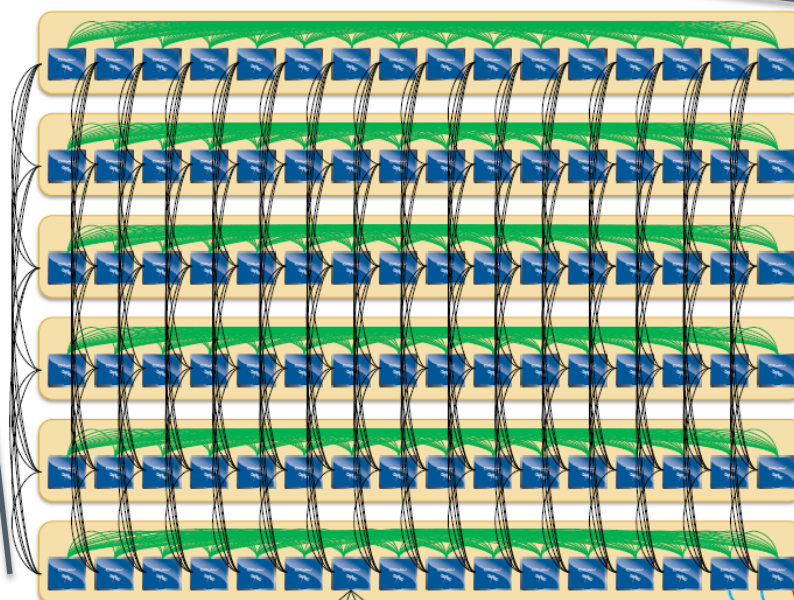
Sisu: Cray XC40 Supercomputer

- For large parallel jobs
- Intel Haswell processor E5-2690 v3 product family; 2,6 GHz (phase 1 Sandy Bridges replaced)
- Cray Aries Interconnect
- 40 512 cores
- 24 cores per node
- 64 GB memory per node

Cray Dragonfly Topology



All-to-all network
between groups



2 dimensional
all-to-all network
in a group



4 nodes connect
to a single Aries

Optical uplinks to
inter-group net

Source:
Robert Alverson, Cray
Hot Interconnects 2012 keynote

Running on Sisu Phase 2

➤ Sisu guide

- <https://research.csc.fi/sisu-user-guide>
- Phase 1 binaries (static) may or may not run, **CSC strongly advises to recompile your code (and compare performance)**
- Login nodes are still based on Sandy Bridge (as they were in Phase 1)
 - Cross compiling is required
 - Haswell optimized code will not run in login nodes

➤ Scalability tests required for more than 1008 cores

- <https://research.csc.fi/sisu-scalability-tests>
- Large test queue available

Sisu Phase 2 features



- AVX-2
 - May need to optimize for wider vectors' size
 - Max 16 flop/cycle
- DDR4
 - Higher bandwidth, lower power consumption
- Max job size increased
 - 400 nodes = 9600 cores
- Native SLURM on the way
 - We might be moving to it at some point

Taito: HP Supercluster



- For serial and small parallel jobs
- Heterogeneous: Intel Sandy Bridge (phase 1) & Intel Haswell (phase 2, not yet installed) processors
- FDR InfiniBand interconnect
- ~18 000 cores
- Different memory configurations: 64, 128, 256 GB and 1.5 TB per node



Taito is a heterogeneous cluster



- Different jobs need different resources
 - Bulk Haswell compute nodes
 - Bulk Sandy Bridge compute nodes
 - Largemem Sandy Bridge compute nodes
 - Hugemem Sandy Bridge compute nodes
- Local */tmp* disk 2 TB on each compute node
 - reserve only what you need

One SLURM to serve them all...



- Do old applications run on new CPUs?
 - May run, CSC **recommends re-compiling**
 - Build your software for both (old and new) architecture
 - Gain depends on architecture
- Batch job scripts need to be updated
 - Number of cores per node: Phase 1: 16, Phase 2: 24
 - Memory changes
 - Instructions will be available through user guides
 - Partition CPU architecture can be specified

SLURM configuration: Fair usage



- ➊ SLURM uses fair share: the highest priority jobs go into execution next
 - Priority is decreased by the total amount of resources used in last 2 weeks per user
 - Priority is increased by time spent queueing
 - Backfiller will try to put small jobs into gaps due to current available resources and highest priority job
 - Jobs labeled "Association limit" are not eligible to run (due to too many jobs in queue by the user)
- ➋ *Due to abuse, a maximum limit of jobs in queue now enforced*
- ➌ Chain jobs (--dependency -flag for SLURM) if you need long running time
- ➍ Don't overallocate memory (add this command to your batch script
`used_slurm_resources.bash` will print requests vs. used at stdout)
 - If you request a full node (-N 1), use `--mem=55000` instead of `--mem-per-core=something`
 - If you see abuse or think that the setup is unfair, contact helpdesk@csc.fi
- ➎ SUI has a monitoring tool for your jobs and used resources (Services -> eServices -> My Project)

How to prepare for Taito Phase 2?

➊ Porting strategy

- Getting started document and a User Guide for Sisu prepared
- Compilers, libraries, flags, ...
- Preliminary performance data
- Add **AVX-2 flag** when compiling your code
- CSC ports and optimizes a number of applications for the new architectures
- Consider testing your code on Sisu, which has Haswell CPUs

- Official opening on **1.10.2014**
- Direct liquid cooled, very energy-efficient
- ***Accelerators and co-processors***
 - 38 NVIDIA K40 nodes = 76 GPUs
 - 12 GB memory per card
 - 45 Intel Xeon Phi (MIC) nodes = 90 Xeon Phis
 - 16 GB memory per card
 - Energy efficient (slow ...) CPU's

How to access Bull

- Logically part of Taito
- Accessing the resources
 - Intel Xeon Phi: `ssh taito-mic (from taito.csc.fi)`
 - Still in beta phase
 - NVIDIA K40: `ssh taito-gpu.csc.fi`
- See Taito user guide
 - `taito-gpu`
 - `taito-mic`

Fast and large storage: DDN Phase 3

- HPC storage used by Sisu and Taito
- Lustre parallel file system
- System size increased to ~4 PB
 - About 1.9 PB added to the current configuration in early October 2014
 - Aggregate bandwidth > 80 GB/s (previously ~48 GB/s)
- Available together with Phase2 supercomputers

Disks in total



- **4.0 PB on DDN**
 - \$HOME directory (on Lustre)
 - \$WRKDIR (*not backed up*), soft quota 5 TB / user
 - Up to 100 TB / project
- **HPC Archive**
 - 2 TB / user, common between Sisu and Taito
 - Up to 100 TB / project
- **3 PB disk space through TTA/IDA**
 - 1 PB for Universities
 - 1 PB for Finnish Academy (SA)
 - 1 PB to be shared between SA and ESFRI
 - more could be requested
- **/tmp on Sisu and Taito (around 1.8 TB) to be used for compiling codes on login nodes**

Software and database offered by CSC



- Large selection (over 200) of software and database packages for research <https://research.csc.fi/software>
- Mainly for academic research in Finland
- Centralized national offering: software consortia, better licence prices, continuity, maintenance, training and support

Services for Research

Home Sciences Computing **Software** News Support Sci

Services for Research → Software → Software Packages

Software

Software Packages

Programming

Parallel Computing

Code Optimization

Visualization

Source Software

Development at CSC

Software Package

All software packages in alphabet

Title

Abaqus

ABYSS

Acquis Communautaire Multiling

ADF

afterburner

Ajatella, Miettä, Pohtia, Harkita

ALLPATHS-LG

Amber

ANSYS Academic Research

ANSYS Academic Teaching Intr

ANSYS CFX

ANSYS Fluent

ANSYS ICEM CFD

ARB

ArcGIS

AutoDock

Babel

BEAST



The collage features several logos and interface snippets:

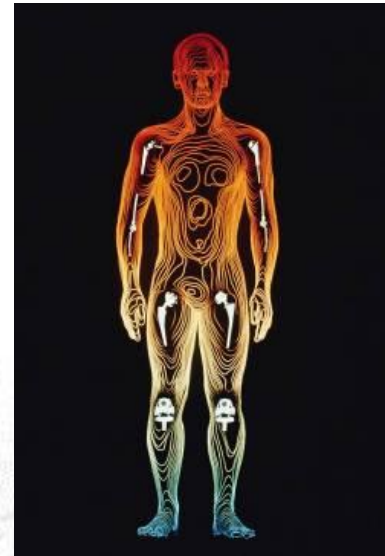
- Gromacs**: A logo with a blue and white molecular structure.
- Gaussian**: A logo with a blue and white molecular structure and the text "THE OFFICIAL GAUSSIAN 03 WEB SITE".
- ADF**: A logo with a blue and white molecular structure and the text "Scientific Computing & Modeling".
- PERCH**: A logo with a red and white molecular structure and the text "Solutions Ltd".
- NWChem**: A logo with a blue and white molecular structure and the text "high performance computational chemistry software".
- PEAK research**: A logo with a blue and white molecular structure and the text "NMR Software".

Applying for account/resources

[Apply for CSC account:](#)

<https://research.csc.fi/accounts-and-projects>

- Most of CSC services are free for academic researchers, but usually a **CSC user account** is required.
 - Basic usage: register as CSC customer via SUI
 - Larger computing resources via an application form
- **Benefits**
 - A wide selection of scientific programs and databases available at CSC servers.
 - ICT resources and science-aware support (helpdesk@csc.fi)
 - Courses and events covering many areas are organized regularly.
 - Guide books and magazines in PDF.
 - CSC's research and development to improve services.
 - Networks bring together people with similar interests in science and technology.



Taito-shell replaces Hippu

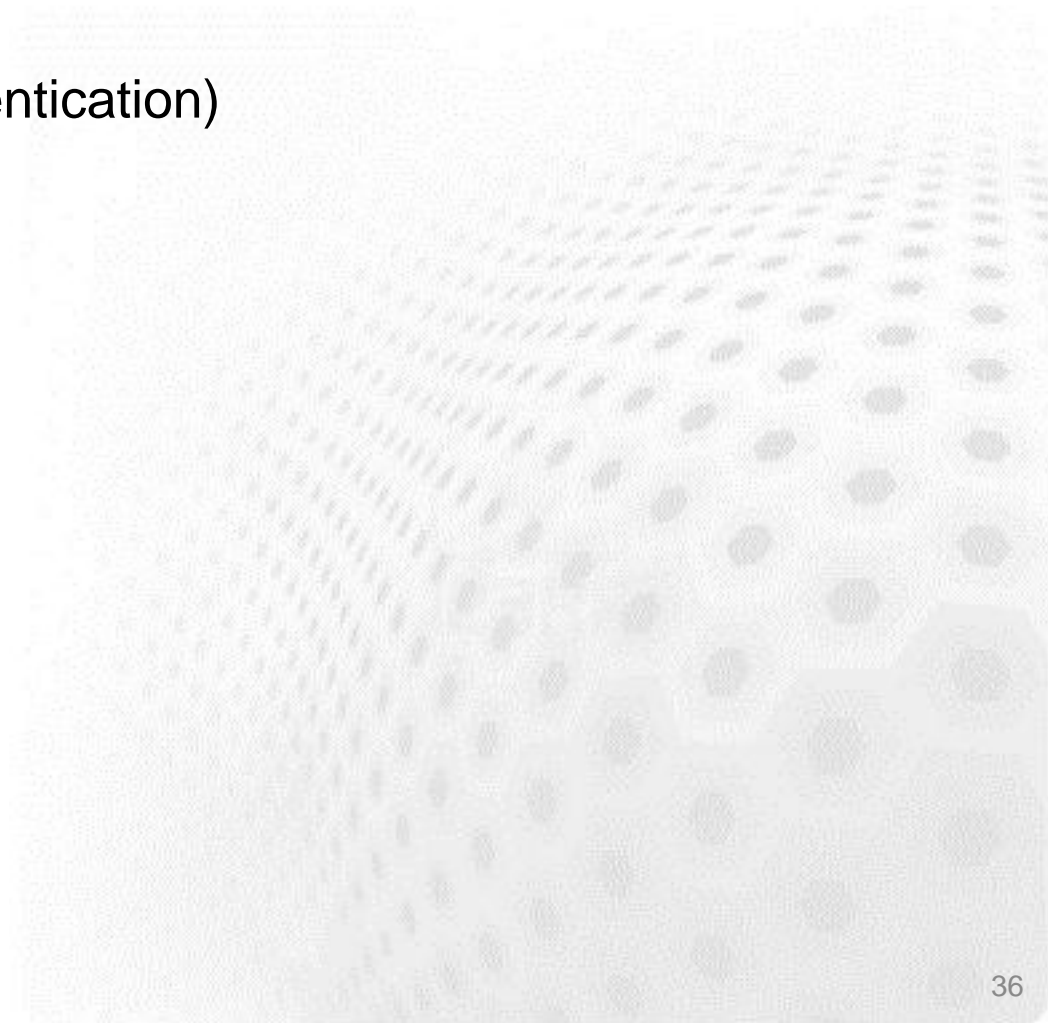
- Interactive session on a Taito compute node
 - E.g. run a GUI, run long non-intensive jobs, etc.
- Two 256GB nodes allocated, easy to expand
 - Maximum of 4 cores/128GB per user, no time limit
- Access: `ssh -X taito-shell.csc.fi`
 - Also via drop down menu in nxkajaani
 - Technically a slurm job without dedicated resources
 - Processes killed when logged out
 - Can be left running via screen (on Taito) or via nxkajaani (exit with suspend)
- Feedback welcome!
- <https://research.csc.fi/taito-shell-user-guide>

How to get access to CSC supercomputers?



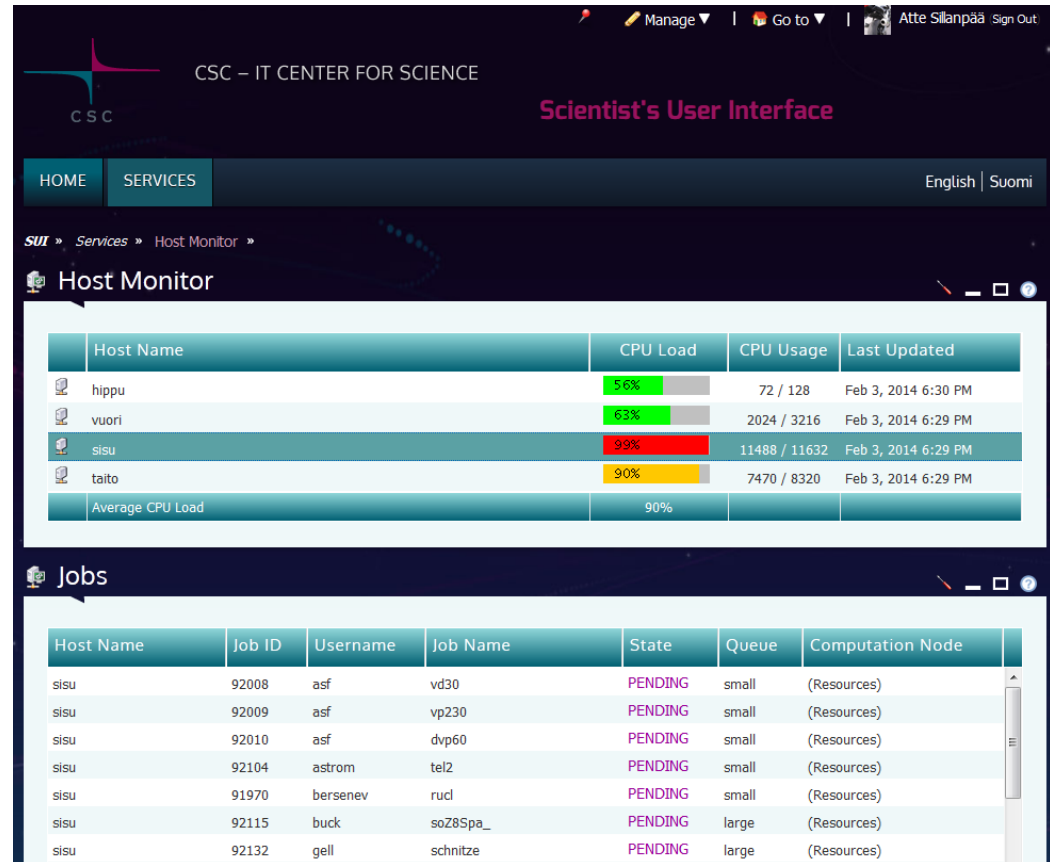
➡ **sui.csc.fi** (HAKA authentication)

– Sing up



Host Monitor in SUI

- ➡ Load on servers
- ➡ Running jobs (queue)
- ➡ sui.csc.fi



The screenshot displays the SUI (Scientist's User Interface) Host Monitor page. The interface includes a navigation bar with 'HOME' and 'SERVICES' tabs, and a user profile section for 'Atte Sillanpää'. The main content area is titled 'Host Monitor' and contains two tables.

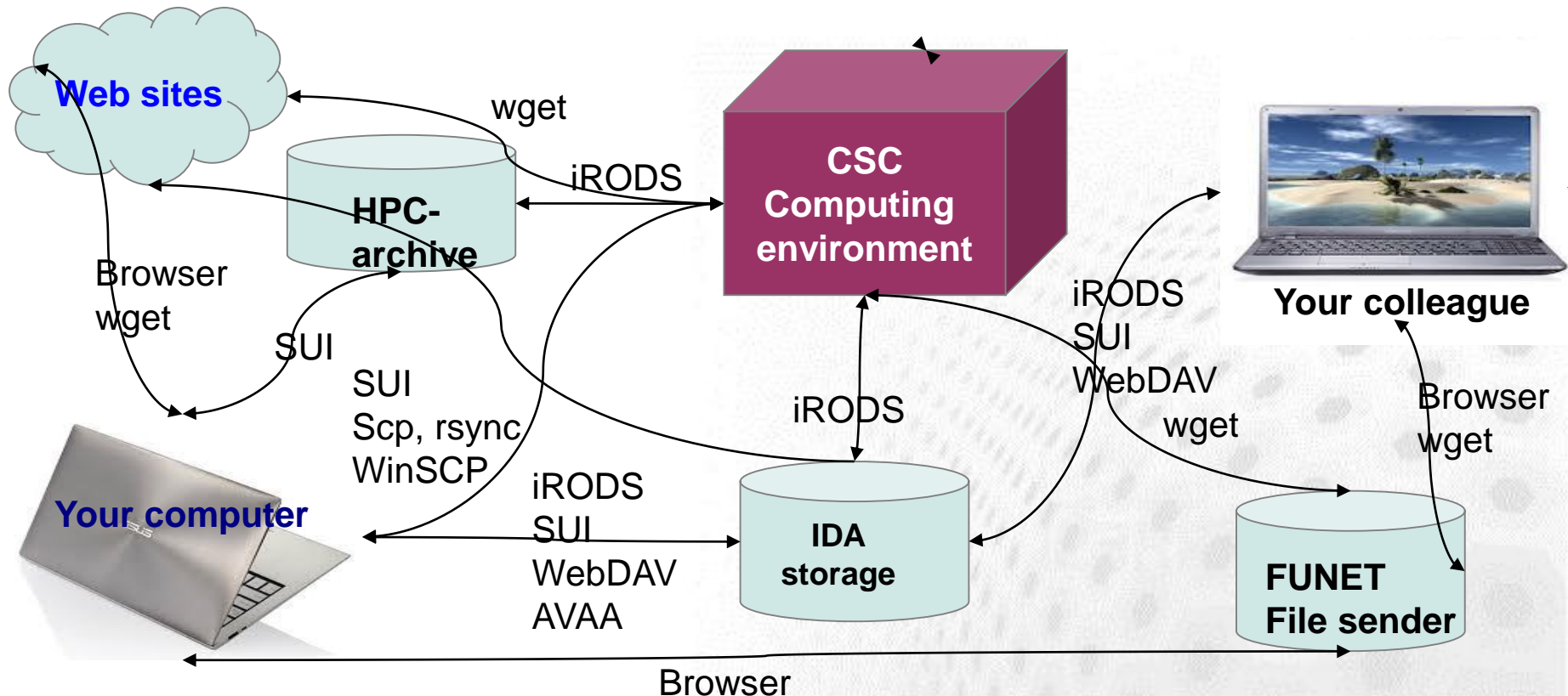
Host Monitor Table:

Host Name	CPU Load	CPU Usage	Last Updated
hippu	56%	72 / 128	Feb 3, 2014 6:30 PM
vuori	63%	2024 / 3216	Feb 3, 2014 6:29 PM
sisu	99%	11488 / 11632	Feb 3, 2014 6:29 PM
taito	90%	7470 / 8320	Feb 3, 2014 6:29 PM
Average CPU Load	90%		

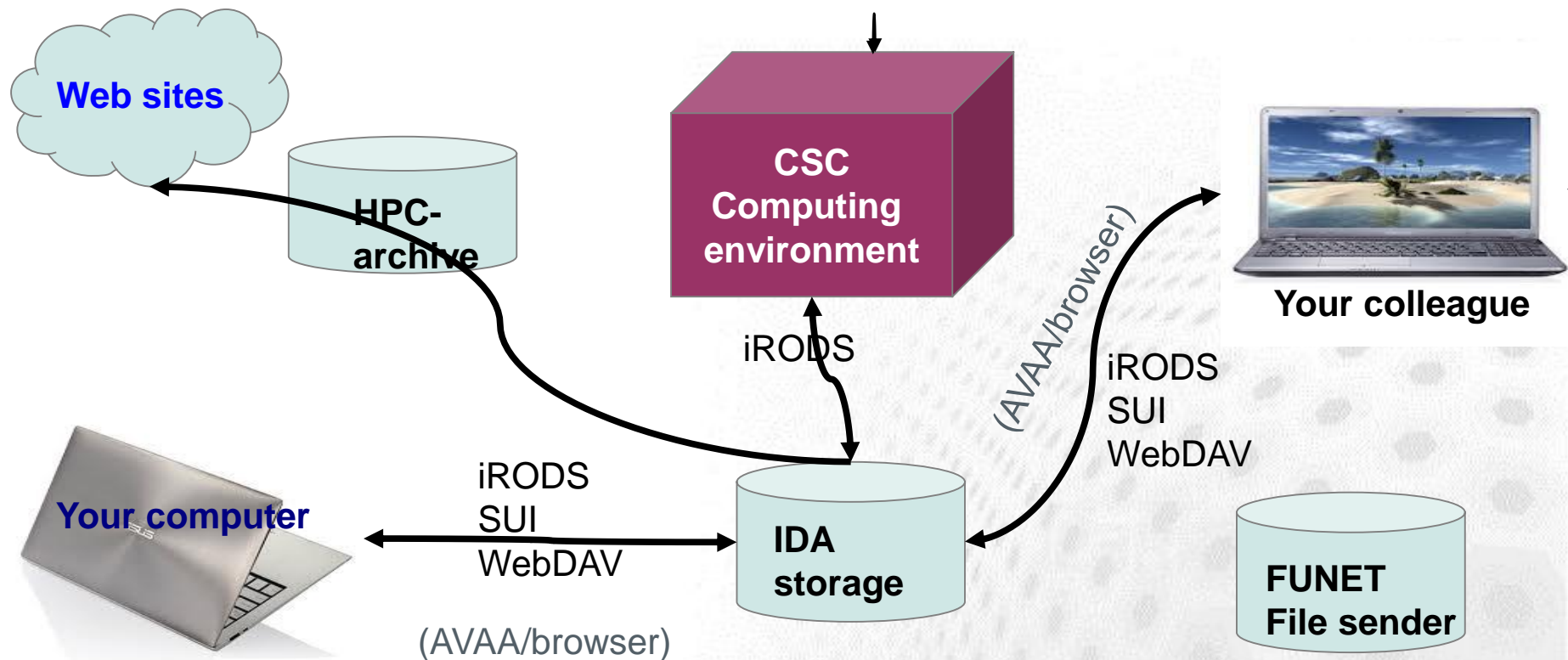
Jobs Table:

Host Name	Job ID	Username	Job Name	State	Queue	Computation Node
sisu	92008	asf	vd30	PENDING	small	(Resources)
sisu	92009	asf	vp230	PENDING	small	(Resources)
sisu	92010	asf	dvp60	PENDING	small	(Resources)
sisu	92104	astrom	tel2	PENDING	small	(Resources)
sisu	91970	bersenev	rucl	PENDING	small	(Resources)
sisu	92115	buck	so28Spa_	PENDING	large	(Resources)
sisu	92132	gell	schnitze	PENDING	large	(Resources)

Moving data to and from CSC



IDA storage service



IDA

- Part of ATT
- Quotas granted by universities and Academy of Finland
- Several interfaces (WWW/SUI, network disk, i-commands)
- Internet accessible
- Project based structure
- Flexible sharing
- Data can be made public through AVAA

HPC-archive

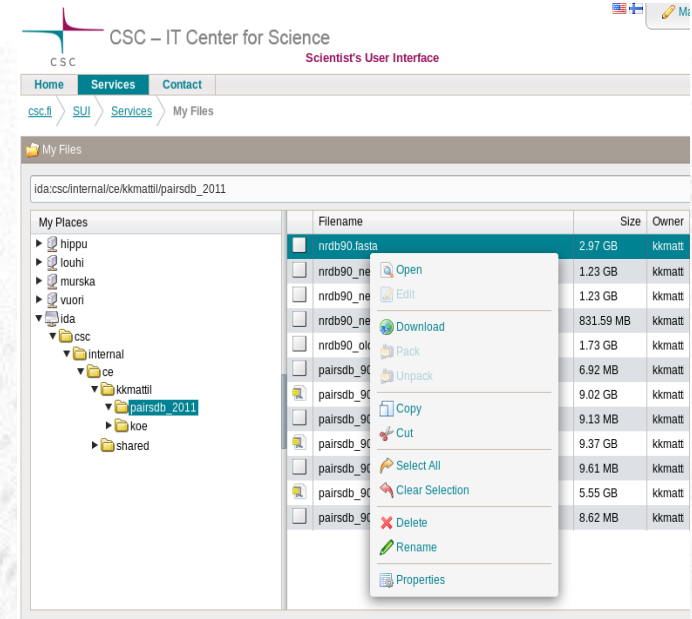
- Part of CSC computing environment
- 2 TB default quotas for CSC users
- Usage with i-commands
- Visible only to CSC environment
- Personal storage area
- Replaced the old \$ARCHIVE service

iCommands

- `iput file` move file to IDA
- `iget file` retrieve file from IDA
- `ils` list the current IDA directory
- `icd dir` change the IDA directory
- `irm file` remove file from IDA
- `imv file file` move file inside IDA
- `imeta command` view and edit metadata
- `irsync` synchronize the local copy with the copy in IDA
- `imkdir` create a directory to IDA
- `iinit` Initialize your IDA account



IDA in Scientist's User Interface



Cloud computing: three service models



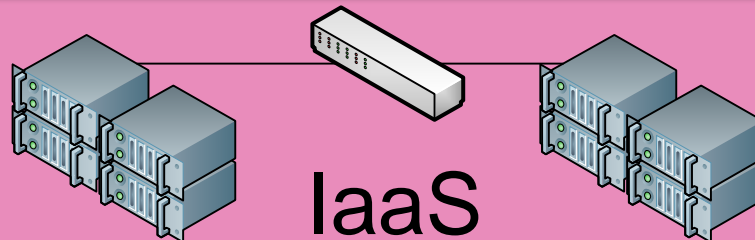
Software



Operating systems



Computers and
networks



cPouta on Taito



Taito cluster:

two types of nodes, HPC and cloud

HPC
node

HPC
node

Cloud
node

Cloud node

Host OS: RHEL

Virtual machine
• Guest OS:
Ubuntu

Virtual machine
• Guest OS:
Windows



Web interface



Instances

+ Launch Instance

Terminate Instances

<input type="checkbox"/>	Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
<input type="checkbox"/>	oli_test3	192.168.1.19 86.50.168.20	medium 30GB RAM 10GB Disk	oli-bombay	Active	None	Running	Create Snapshot
<input type="checkbox"/>	kalletest	192.168.1.26						
<input type="checkbox"/>	lalves_test	192.168.1.26						
<input type="checkbox"/>	pj-ubuntu	192.168.1.26 86.50.168.22						
<input type="checkbox"/>	HarriPerformanceTests_1_4	192.168.1.26 86.50.168.20	Disk					More
<input type="checkbox"/>	HarriPerformanceTests_1_3	192.168.1.26 86.50.168.22	tiny 1GB RAM 1 VCPU 10GB Disk	keypair-harri	Active	None	Running	Create Snapshot More

```
khappone@pikkulintu:~$ nova list
```

ID	Name	Status	Task State	Power State	Networks
781d4a2f-c21c-4dfd-8d58-87428e4c7502	CT-IFTest1	ACTIVE	None	Running	CThomas Deployment=10.5.5.10, 86.50.168.30
7abbe103-c7f0-4db0-87a7-8758aa8c086a	DS40-server	ACTIVE	None	Running	csc=192.168.1.32, 86.50.168.64
21e2f4f3-9c4b-4561-8a4e-2c4c62141237	Jarin testijärjestelmä	SUSPENDED	None	Shutdown	csc=192.168.1.34
0532b4d0-9ac6-4e8a-8637-4192f1039039	PoutaMon	ACTIVE	None	Running	csc=192.168.1.33, 86.50.168.35
b997c581-e047-4c17-acf4-ee73962f1f71	lalvesFedCloudTest	ACTIVE	None	Running	csc=192.168.1.2, 86.50.168.7

```
khappone@pikkulintu:~$
```

Command line tools

<https://pouta.csc.fi:8777/v2/csc/servers/0532b4d0-9ac6-4e8a-8637-4192f1039039>

<https://pouta.csc.fi:8777/v2/csc/flavors/1a0f1143-47b5-4e8a-abda-eba52ae3c5b9>

<https://pouta.csc.fi:8777/v2/csc/images/>

REST API

cPouta's use cases



- Enhanced security – isolated virtual machines
- Advanced users – able to manage servers
- Difficult workflows – can't run on Taito
- Complex software stacks
- Ready made virtual machine images
- Deploying tools with web interfaces
- "We need root access"

*If you can run on Taito – run on Taito
If not – Pouta might be for you*

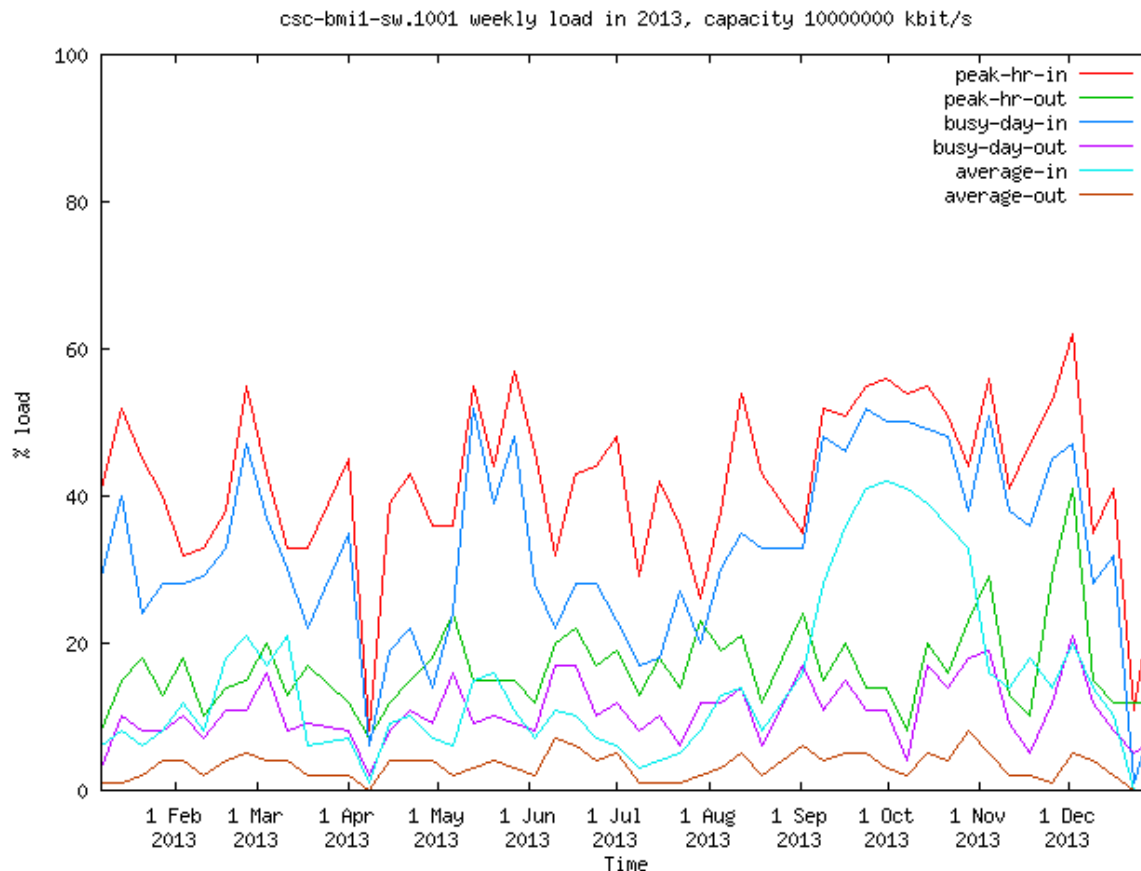
- Pouta user guide: <https://research.csc.fi/pouta-user-guide>

ePouta

- Renewing the cloud cluster equipment in Espoo in 2015
 - Changes to OpenStack cloud middleware (autumn 2014)
 - Focus on secure computing and service for organisations
 - Idea: seamless scaling of local resources using a trusted compute center (in Finland)
 - Requires local IT admin contact
 - Funding model and resource allocation policy is still under debate, supported by ELIXIR Finland



CSC – Meilahti genomics laaS data traffic 2013



5.8 PB in
1.4 PB out

Avg. 221 MB/s 24
hours a day all year
round

Grid computing with Finnish Grid Infrastructure (FGI)



ARC Grid Monitor

2014-05-27 CEST 12:45:37

Processes: ■ Grid ■ Local



Country	Site	CPU	Load (processes: Grid+local)	Queueing
+ Finland	Aesyle (FGI)	72	0+35	0+0
	Alcyone (CMS)	892	156+312	1040+0
	Alcyone (FGI)	892	6+461	19+0
	Asterope (FGI)	192	84+0	10+1
	Celaeno (FGI)	448	172+0	9+0
	Electra (FGI)	672	0+478	0+0
	Jade (HIP)	768	227+541	25+49
	Maia (FGI)	768	360+408	14+0
	Merope (FGI)	1612	0+1319	14+0
	Pleione (FGI)	288	144+0	13+0
	Taygeta (FGI)	360	42+174	15+0
	Triton (FGI)	6972	182+0	2+0
	Usva (CSC/FGI/test)	144	12+0	0+0
TOTAL	13 sites	14080	1385 + 3728	1161 + 50

- In grid computing you can use several computing clusters to run your jobs
- Grids suits well for array job like tasks where you need to run a large amount of independent sub-jobs
- You can also use FGI to bring cluster computing to your local desktop
- FGI: 12 computing clusters, about **10 000** computing cores
- Software: Run Time Environment include applications from all fields, e.g., bioinformatics, chemistry, physics:
 - <https://confluence.csc.fi/display/fgi/Runtime+Environments>

Using grid



- The jobs are submitted using the ARC middleware (<http://www.nordugrid.org/arc/>)
 - Using ARC resembles submitting batch jobs in Taito or Sisu
- ARC is installed in Hippu and Taito, but you can install it to your local machine too.
 - Setup command in Hippu:
 - `module load nordugrid-arc`
 - Basic ARC commands:

➤ <code>arcproxy</code>	(Set up grid proxy certificate for 12 h)
➤ <code>arcsub <i>job.xrsl</i></code>	(Submit job described in file <i>job.xrsl</i>)
➤ <code>arcstat -a</code>	(Show the status of all grid jobs)
➤ <code>arcget <i>job_id</i></code>	(Retrieve the results of a finished grid job)
➤ <code>arckill <i>job_id</i></code>	(kill the given grid job)
➤ <code>arcclean -a</code>	(remove job related data from the grid)

Sample ARC job description file



```
&
(executable=runbwa.sh)
(jobname=bwa_1)
(stdout=std.out)
(stderr=std.err)
(gmlog=gridlog_1)
(walltime=24h)
(memory=8000)
(disk=4000)
(runtimeenvironment>="APPS/BIO/BWA_0.6.1")
(inputfiles=
( "query.fastq" "query.fastq" )
( "genome.fa" "genome.fa" )
)
(outputfiles=
( "output.sam" "output.sam" )
)
```


Getting started with FGI-Grid



1. Apply for a grid certificate from TERENA (a kind of grid passport)
2. Join the FGI VO (Access to the resources)
3. Install the certificate to Scientists' User Interface and Hippu.
4. Install ARC client to your local Mac or Linux machine for local use)
5. Instructions: *<http://research.csc.fi/fgi-preparatory-steps>*

Please ask help to get started: helpdesk@csc.fi

FGI user guide: <http://research.csc.fi/fgi-user-guide>

- **CSC courses:** <http://www.csc.fi/courses>
 - Introduction to Linux and Using CSC Environment Efficiently *10.-11.2.2015*
 - Pouta training 23.3.2015
 - CSC HPC Summer School
 - Spring, Autumn, Winter Schools
 - Parallel Programming
 - Some courses have possibility for remote participation
 - Course materials often available from event website for self study
- **Taito Phase 2 workshop**
 - *Spring 2015*

Grand Challenges



- ➔ Normal GC (*call in half a year / year intervals*)
 - New CSC resources available for a year
 - No limit for number of cores
 - *Next call beginning of 2015*
- ➔ Remember also PRACE/DECI calls
 - CSC supports the technical aspects of the applications



CSC Phase2 resources' summary



- ***Sisu* supercomputer**
 - General availability since **9.9.2014**
- ***Taito* supercluster**
 - Installation ongoing
 - Part of Taito used for *Pouta Cloud*
 - *taito-shell* replacing *Hippu* service
- ***Bull* system**
 - General availability since **1.10.2014**
 - *45 nodes with 2 Intel Xeon Phi coprocessors each*
 - *38 nodes with 2 NVIDIA Tesla K40 accelerators each*
- ***DDN* HPC storage system**
 - Totaling *4 PB of fast parallel storage*



Physics people at CSC



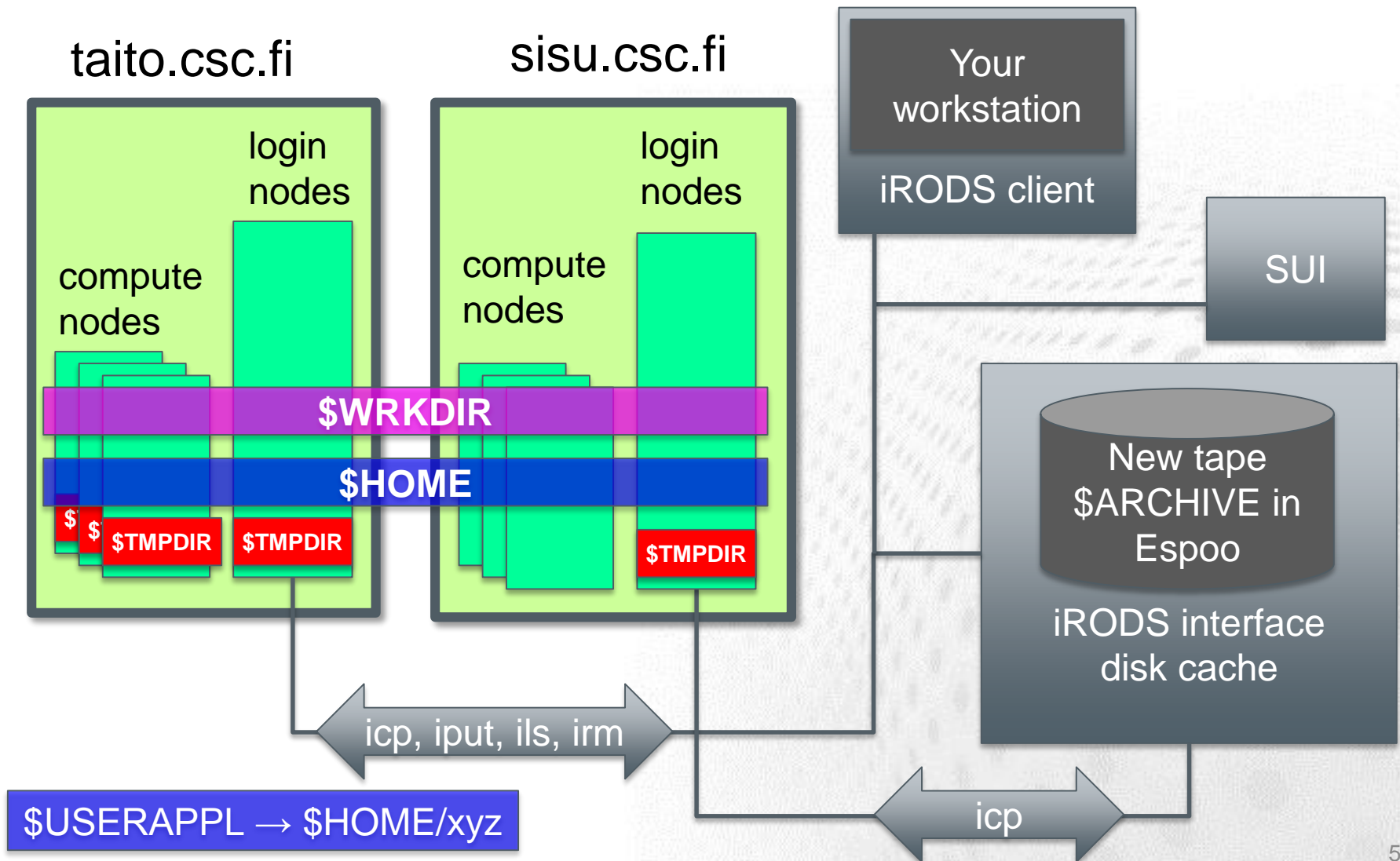
- *Particle based methods*: Jan Åström
- *Materials physics*: Jussi Enkovaara
- *Geophysics/glaciology*: Thomas Zwinger
- *Nanoscience/semiconductors*: Jura Tarus
- *Nuclear/particle physics*: Tomasz Malkiewicz
- *Partial differential equations/ELMER*: Peter Råback
- A few with background in *DFT* e.g. Juha Lento
- *Quantum chemistry*: Nino Runeberg
- A few with *numerical mathematics* background
- Several with advanced code optimisation skills
- Everything related to HPC in general

Q/A: Need disk space



- 4 PB on DDN
 - \$HOME, \$USERAPPL: 50 GB
 - \$WRKDIR (not backed up), soft quota: 5 TB
- HPC ARCHIVE: 2 TB / user, common between Cray and HP, up to 100 TB upon request
- /tmp (around 1.8 TB) to be used for *compiling codes*
- Disk space through IDA

Disks at Kajaani



Q/A: Need large capacity -> Grand Challenges

- Normal GC (*in half a year / year*)
 - new CSC resources available for a year
 - no bottom limit for number of cores, up to 50%
- Special GC call (mainly for Cray) (*when needed*)
 - possibility for short (day or less) runs with the whole Cray
- Remember also PRACE/DECI
 - <http://www.csc.fi/english/csc/news/news/pracecalls>

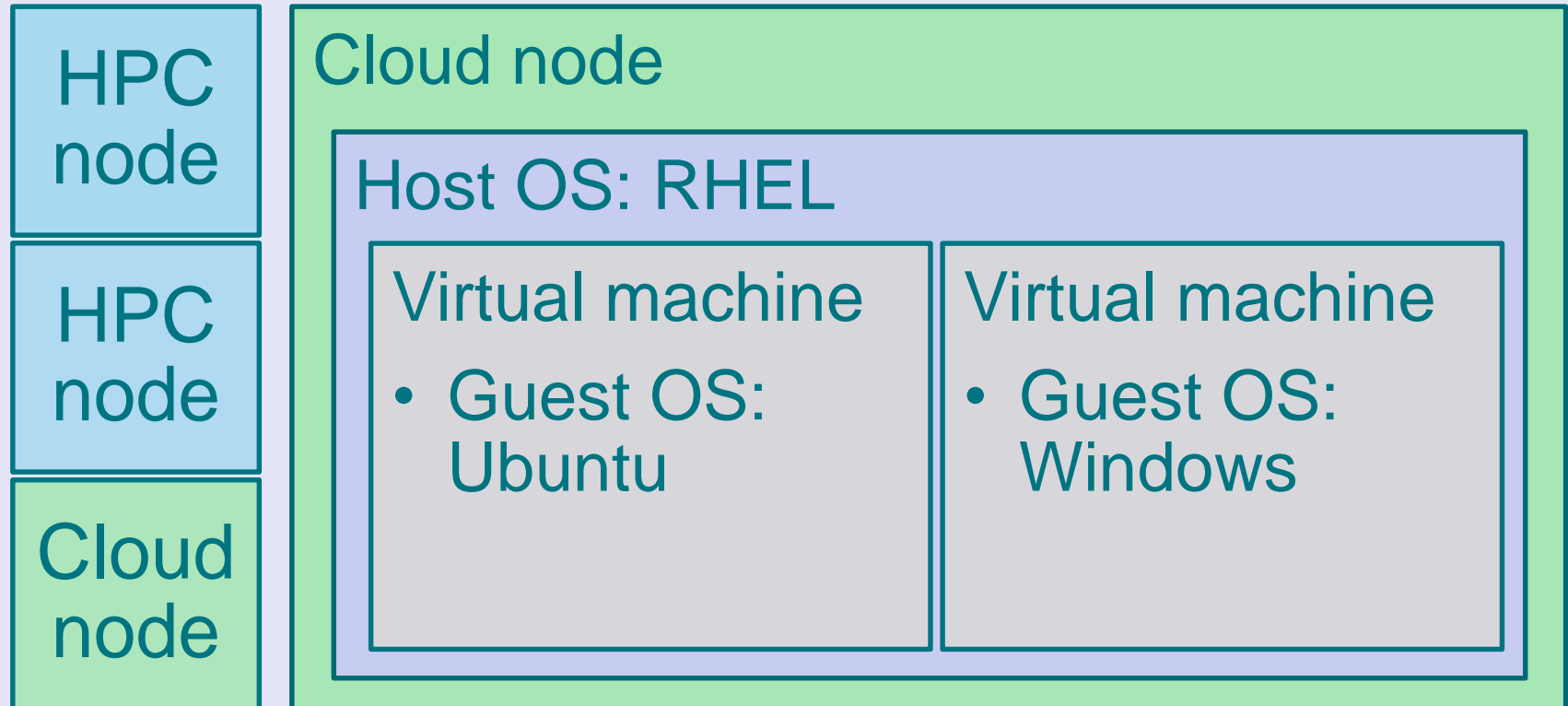
Q/A: Is Cloud something for me?

->example: Taito



Taito cluster:

two types of nodes, HPC and cloud



Q/A: How fast is the I/O?

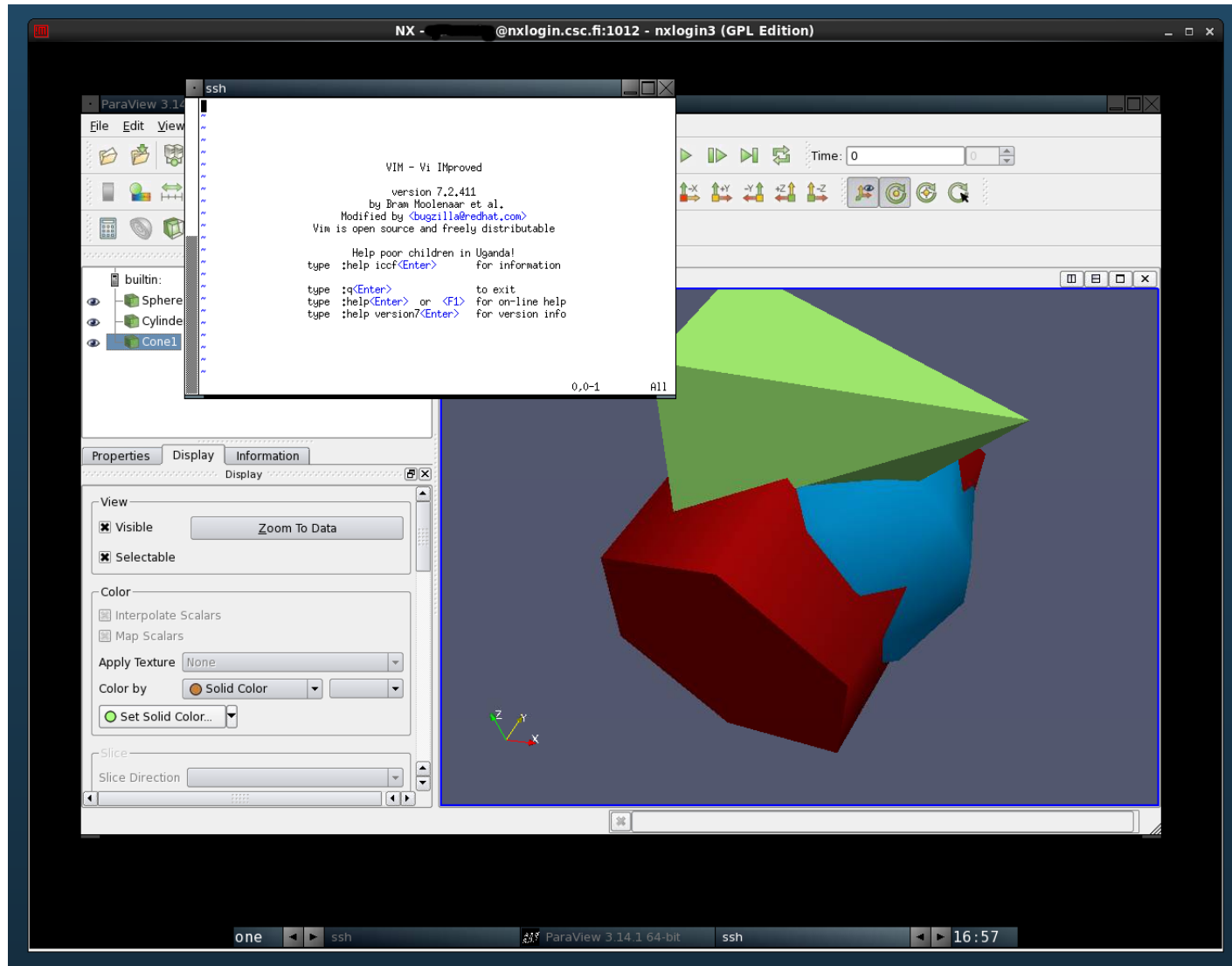


I/O speed

- Infiniband interconnect 56 Gbit/s, tested to give 20 GB/s (peak, on DDN)
- i-commands 100 MB/s = 1 Gbit/s (10-16 thread, if > 32 MB then spreads, Kernel schedules)
- SUI: 11 MB/s, 1 GB = 1 min
- Fastest laptop: 120 MB/s, disc speed 40 MB/s write
- 10 Gbit/s ethernet = 1.2 GB/s
- Metadata operations for Lustre take long, therefore not good to have many small files

Q/A: Fastest way to connect?

NoMachine NX server for remote access



Q/A: Is there a single place to look for info regarding supercomputers?

- User manuals
 - *<http://research.csc.fi/guides>*
- Support
 - *helpdesk@csc.fi*



Round robin

Round robin



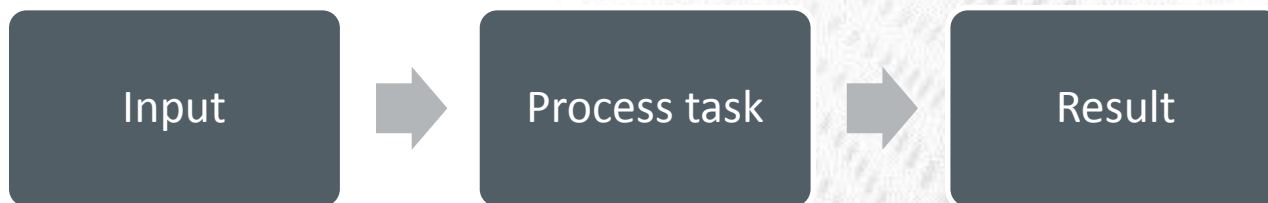
- ➡ *What are your research interest?*
- ➡ *What are your needs in terms of computing?*
- ➡ *Which applications/codes are you using?*
- ➡ *How CSC can help?*



Parallel computations

Computing in parallel

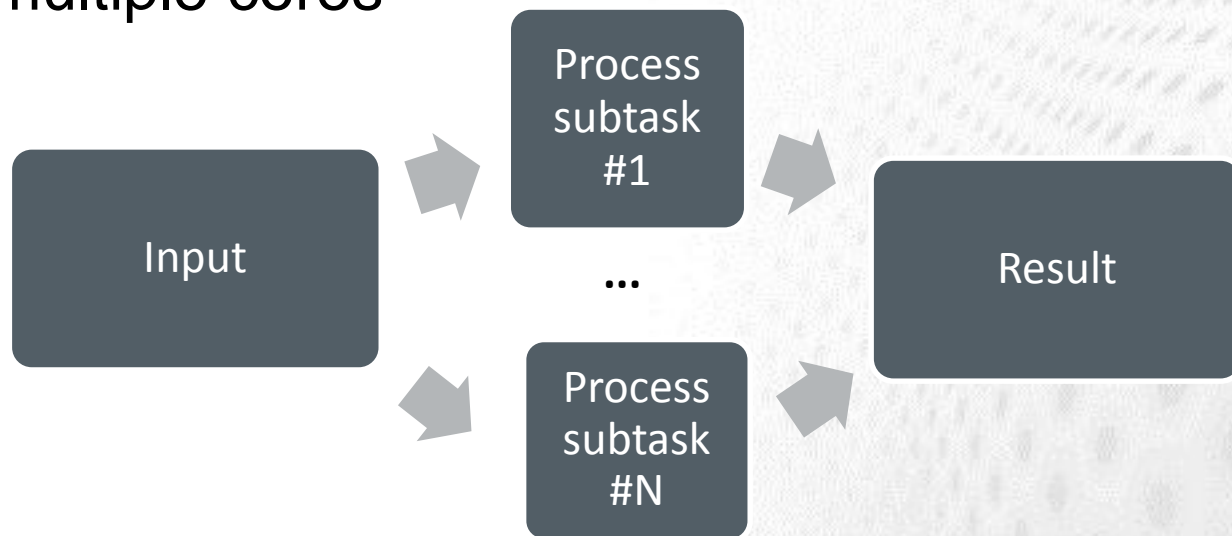
- Serial computing
 - single processing unit (“core”) is used for solving a problem



Computing in parallel

● Parallel computing

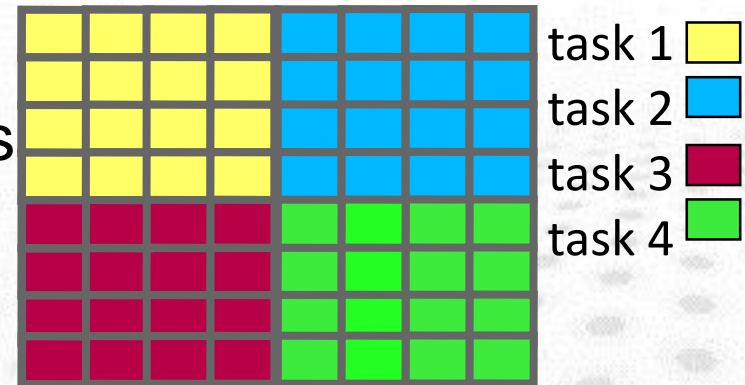
- A problem is split into smaller subtasks
- multiple subtasks are processed *simultaneously* using multiple cores



Exposing parallelism

• Data parallelism

- Data is distributed to processing cores
- Each core performs simultaneously (nearly) identical operations with different data



• Task parallelism

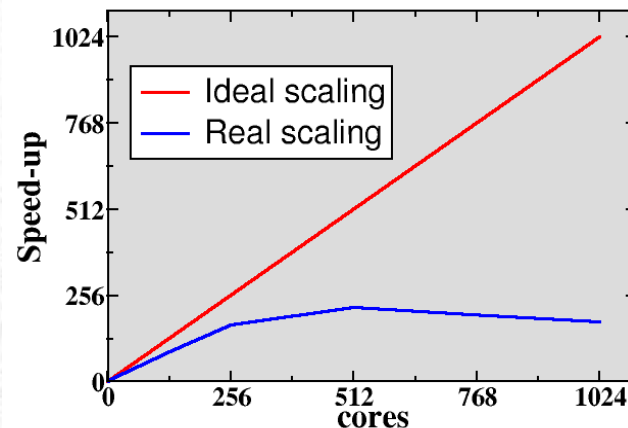
- Different cores perform different operations with (the same or) different data

• These can be combined

Parallel scaling

➤ Strong parallel scaling

- constant problem size
- execution time decreases in proportion to the increase in the number of cores



➤ Weak parallel scaling

- increasing problem size
- execution time remains constant when number of cores increases in proportion to the problem size

Amdahl's law

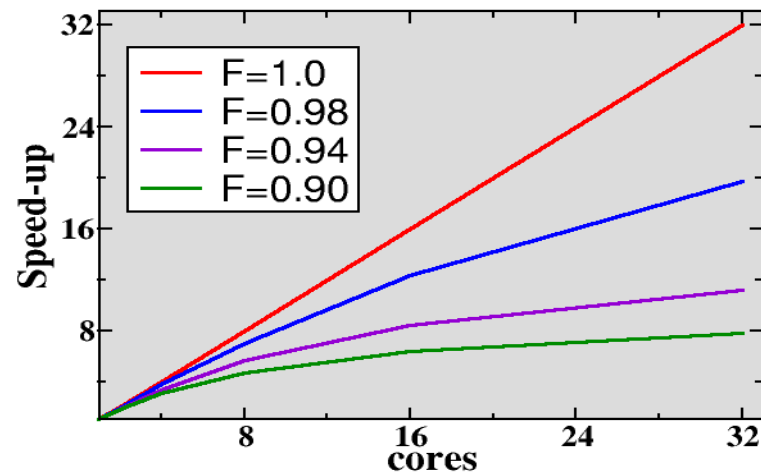
- Parallel programs contain often sequential parts
- Amdahl's law* gives the maximum speed-up in the presence of non-parallelizable parts

Maximum speed-up:

$$\frac{1}{(1 - F) + F/N}$$

F: parallel fraction

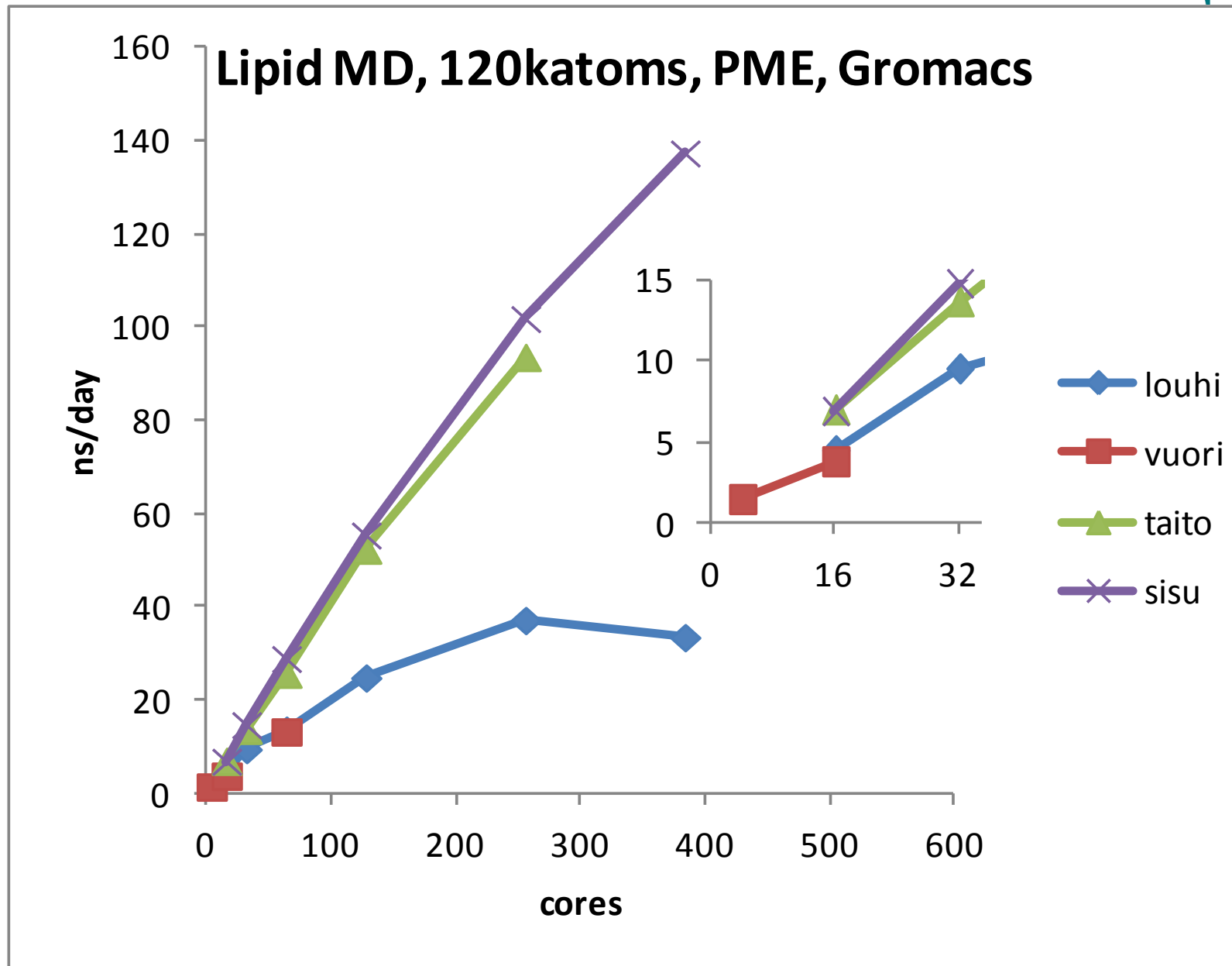
N: number of cores



Parallel computing concepts

- Load balance
 - distribution of workload to different cores
- Parallel overhead
 - additional operations which are not present in serial calculation
 - synchronization, redundant computations, communications

Why and when to use HPC?



Warm-up: quick hands-on on Taito

Live demo/hands-on (Taito)

➡ ***ssh trng01 - trng20 @taito.csc.fi***

➡ ***module avail***

Live demo/hands-on cont.



➡ ***nano test_hostname.sh*** CTRL+O; CTRL+X to exit

```
#!/bin/bash -l
#SBATCH -J print_hostname
#SBATCH -o output.txt
#SBATCH -e errors.t
#SBATCH -t 00:01:00
#SBATCH -p test
#
echo "This job runs on the host: "; hostname
```

➡ ***sbatch test_hostname.sh***

Live demo/hands-on cont.

➡ Check out the output:

– ***less output.txt*** (type ***q*** to quit)

– ***less errors.t*** (type ***q*** to quit)

Modules

- Some software installations are conflicting with each other
 - For example different versions of programs and libraries
- Modules facilitate the installation of conflicting packages to a single system
 - User can select the desired environment and tools using module commands
 - Can also be done "on-the-fly"

Taito module system

- ➔ *module avail* shows only those modules that can be loaded to current setup (no conflicts or extra dependencies)
 - Use *module spider* to list all installed modules and solve the conflicts/dependencies
- ➔ No PrgEnv- modules (*on Taito*)
 - Changing the compiler module switches also MPI and other compiler specific modules

Typical module commands

<code>module avail</code>	shows available modules (compatible modules in taito)
<code>module spider</code>	shows all available modules in taito
<code>module list</code>	shows currently loaded modules
<code>module load <name></code>	loads module <name> (default version)
<code>module load <name/version></code>	loads module <name/version>
<code>module switch <name1> <name2></code>	unloads module name1 and loads module name2
<code>module purge</code>	unloads all loaded modules

Taito has "meta-modules" named e.g. gromacs-env, which will load all necessary modules needed to run gromacs.

Example serial batch job script on Taito

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial
```

```
module load myprog
srun myprog -option1 -option2
```



```
#!/bin/bash -l
```

- Tells the computer this is a script that should be run using bash shell
- Everything starting with "**#SBATCH**" is passed on to the batch job system (Slurm)
- Everything (else) starting with "**#** " is considered a comment
- Everything else is executed as a command

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2 82
```


#SBATCH -J myjob

- Sets the name of the job
- When listing jobs e.g. with **squeue**, only 8 first characters of job name are displayed.

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2 83
```



```
#SBATCH -e myjob_err_%j
```

```
#SBATCH -o myjob_output_%j
```

- Option **-e** sets the name of the file where possible error messages (stderr) are written
- Option **-o** sets the name of the file where the standard output (stdout) is written
- When running the program interactively these would be written to the command prompt
- What gets written to stderr and stdout depends on the program. If you are unfamiliar with the program, it's always safest to capture both
- **%j** is replaced with the job id number in the actual file name

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```



```
#SBATCH --mail-type=END
```

```
#SBATCH --mail-user=a.user@foo.net
```

- Option **--mail-type=END** = send email when the job finishes
- Option **--mail-user** = your email address.
- If these are selected you get a email message when the job is done. This message also has a resource usage summary that can help in setting batch script parameters in the future.
- To see actually used resources try also: **sacct -l -j <jobid>** (more on this later)

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```


#SBATCH -n 1

- Number of cores to use
- It's also possible to control on how many nodes you job is distributed. Normally, this is not needed. By default use all cores in allocated nodes:
 - **--ntasks-per-node=16**
- Check documentation: <http://research.csc.fi/software>
 - There's a lot of software that can only be run in serial
- OpenMP applications can only use cores in one node

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```


#SBATCH --mem-per-cpu=4000

- The amount of memory reserved for the job in MB
 - 1000 MB = 1 GB
- Memory is reserved on per-core basis even for shared memory (OpenMP) jobs
- Keep in mind the specifications for the nodes. Jobs with impossible requests are rejected (try **squeue** after submit)
- If you reserve too little memory the job will be killed (you will see a corresponding error in the output)
- If you reserve too much memory your job will spend much longer in queue and potentially waste resources (idle cores)

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```


#SBATCH -t 02:00:00

- Time reserved for the job in hh:mm:ss
- When the time runs out the job will be terminated!
- With longer reservations the job queue longer
- Limit for normal serial jobs is 3d (72 h)
 - if you reserve longer time, the job will go to "longrun" queue (limit 7d)
 - In the longrun queue you run at your own risk. If a batch job in that queue stops prematurely no compensation is given for lost cpu time!
 - In longrun you likely queue for a longer time: shorter jobs and restarts are better (safer, more efficient)
- Default job length is 5 minutes → need to be set by yourself.

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```


#SBATCH -p serial

- The queue the job should be submitted to
- Queues are called "partitions" in SLURM
- You can check the available queues with command
sinfo -l

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

PARTITION	AVAIL	TIMELIMIT	JOB_SIZE	ROOT	SHARE	GROUPS	NODES	STATE	NODELIST
serial*	up	3-00:00:00	1	no	YES:4	all	514	mixed	c[5-274,276-453,455-473, ...
serial*	up	3-00:00:00	1	no	YES:4	all	3	idle	c[275,454,474]
parallel	up	3-00:00:00	1-28	no	NO	all	514	mixed	c[5-274,276-453,455-473, ...
parallel	up	3-00:00:00	1-28	no	NO	all	3	idle	c[275,454,474]
longrun	up	7-00:00:00	1	no	YES:4	all	514	mixed	c[5-274,276-453,455-473,...
longrun	up	7-00:00:00	1	no	YES:4	all	3	idle	c[275,454,474]
test	up	30:00	1-2	no	YES:4	all	1	drained	c4
test	up	30:00	1-2	no	YES:4	all	3	idle	c[1-3]


```
module load myprog
srun myprog -option1 -option2
```

```
#!/bin/bash -l
#SBATCH -J myjob
#SBATCH -e myjob_err_%j
#SBATCH -o myjob_output_%j
#SBATCH --mail-type=END
#SBATCH --mail-user=a.user@foo.net
#SBATCH --mem-per-cpu=4000
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH -p serial

module load myprog
srun myprog -option1 -option2
```

➤ Your commands

- These define the actual job to be performed: these commands are run on the compute node.
- See application documentation for correct syntax
- Some examples also from batch script wizard in SUI

➤ Remember to load modules if necessary

➤ By default the working directory is the directory where you submitted the job

- If you include a `cd` command, make sure it points to correct directory

➤ Remember that input and output files should be in `$WRKDIR` (or in some case `$TMPDIR`)

➤ `srun` tells your program which cores to use. There are also exceptions...

Most commonly used sbatch options

Slurm option

--begin=*time*
-c, --cpus-per-task=*ncpus*
-d, --dependency=*type:jobid*
-e, --error=*err*
--ntasks-per-node=*n*
-J, --job-name=*jobname*
--mail-type=*type*
--mail-user=*user*
-n, --ntasks=*ntasks*
-N, --nodes=*N*
-o, --output=*out*
-t, --time=*minutes*
--mem-per-cpu=<number in MB>
--mem=<number in MB>

Description

defer job until HH:MM MM/DD/YY
 number of cpus required per task
 defer job until condition on jobid is satisfied
 file for batch script's standard error
 number of tasks per node
 name of job
 notify on state change: BEGIN, END, FAIL or ALL
 who to send email notification for job state changes
 number of tasks to run
 number of nodes on which to run
 file for batch script's standard output
 time limit in format hh:mm:ss
 maximum amount of real memory per allocated cpu
 required by the job in megabytes
 maximum memory per node

Submitting and cancelling jobs

- The script file is submitted with command
`sbatch batch_job.file`
- *Optional:* sbatch option are usually listed in the batch job script, but they can also be specified on command line, e.g.
`sbatch -J test2 -t 00:05:00 batch_job_file.sh`
- Job can be deleted with command
`scancel <jobid>`

Queues

- The job can be followed with command `squeue`:

<code>squeue</code>	(shows all jobs in all queues)
<code>squeue -p <partition></code>	(shows all jobs in single queue (partition))
<code>squeue -u <username></code>	(shows all jobs for a single user)
<code>squeue -j <jobid> -l</code>	(status of a single job in long format)

- To estimate the start time of a job in queue

```
scontrol show job <jobid>
```

row "StartTime=..." gives an estimate on the job start-up time, e.g.

```
StartTime=2014-02-11T19:46:44 EndTime=Unknown
```

- `scontrol` will also show where your job is running
- If you add this to the end of your batch script, you'll get additional info to stdout about resource usage (works for jobs run with `srun`)
 - `used_slurm_resources.bash`

Job logs

**TIP: Check
MaxRSS to see
how much
memory you
need and avoid
overbooking**

- Command `sacct` can be used to study past jobs
 - Usefull when deciding proper resource requests

<code>sacct</code>	Short format listing of jobs starting from midnight today
<code>sacct -l</code>	long format output
<code>sacct -j <jobid></code>	information on single job
<code>sacct -S YY:MM:DD</code>	listing start date
<code>sacct -o</code>	list only named data fields, e.g.
<code>sacct -u <username></code>	list only jobs submitted by username

```
sacct -o jobid,jobname,maxrss,state,elapsed -j <jobid>
```


Available nodes/queues

- You can check available nodes in each queue with command:
`sjstat -c`

Scheduling pool data:

Pool	Memory	Cpus	Total	Usable	Free	Other	Traits
serial*	64300Mb	16	501	501	5		
serial*	258000Mb	16	16	16	0	bigmem	
parallel	64300Mb	16	501	501	5		
parallel	258000Mb	16	16	16	0	bigmem	
longrun	64300Mb	16	501	501	5		
longrun	258000Mb	16	16	16	0	bigmem	
test	64300Mb	16	4	3	3		
hugemem	1551000Mb	32	2	2	2	bigmem	

Most frequently used SLURM commands



Command	Description
<code>srun</code>	Run a parallel job.
<code>salloc</code>	Allocate resources for interactive use .
<code>sbatch</code>	Submit a job script to a queue.
<code>scancel</code>	Cancel jobs or job steps.
<code>sinfo</code>	View information about SLURM nodes and partitions.
<code>squeue</code>	View information about jobs located in the SLURM scheduling queue
<code>smap</code>	Graphically view information about SLURM jobs, partitions, and set configurations parameters
<code>sjstat</code>	display statistics of jobs under control of SLURM (combines data from <code>sinfo</code> , <code>squeue</code> and <code>scontrol</code>)
<code>scontrol</code>	View SLURM configuration and state.
<code>sacct</code>	Displays accounting data for batch jobs.

Parallel jobs (1/2)

- Only applicable if your program supports parallel running
- Check application documentation on number of cores to use
 - Speed-up is often not linear (communication overhead)
 - Maximum number can be limited by the algorithms
 - Make sure (test) that using more cores speeds up calculation
- Mainly two types: MPI jobs and shared memory (OpenMP) jobs
 - OpenMP jobs can be run only inside one node
 - All cores access same memory space
 - MPI jobs can span several nodes
 - Each core has its own memory space

Parallel jobs (2/2)

- Memory is normally reserved per-core basis
 - For OpenMP jobs divide total memory by number of cores
 - Take care to only request possible configurations
 - If you reserve a complete node, you can also ask for all the memory
- Each server has different configuration so setting up parallel jobs in optimal way requires some thought
- See server guides for specifics: <http://research.csc.fi/guides>
 - Use Taito for large memory jobs
 - Sisu for massively parallel jobs
 - Check also the software specific pages for examples and detailed information: <http://research.csc.fi/software>

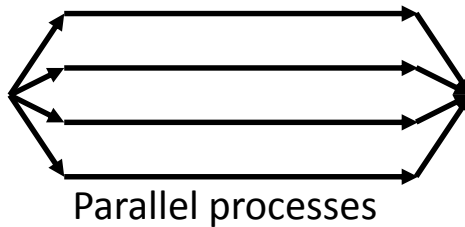
Array jobs (advanced usage)

- Best suited for running the same analysis for large number of files
- `#SBATCH --array=1-100`
- Defines to run 100 jobs, where a variable `$SLURM_ARRAY_TASK_ID` gets each number (1,2,...100) in turn as its value. This is then used to launch the actual job (e.g. `srun myprog input_
$SLURM_ARRAY_TASK_ID > output_
$SLURM_ARRAY_TASK_ID`)
- Thus this would run 100 jobs:

```
srun myprog input_1 > output_1  
srun myprog input_2 > output_2  
...  
srun myprog input_100 > output_100
```
- For more information
 - <http://research.csc.fi/taito-array-jobs>

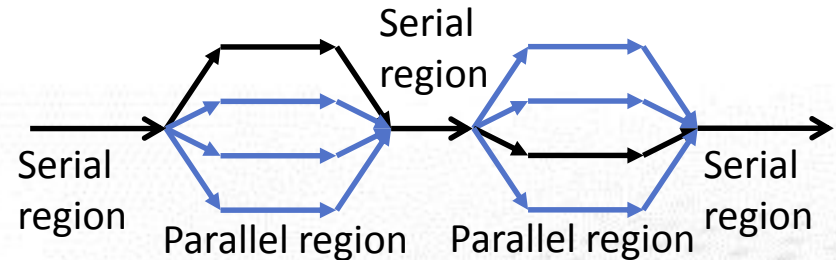
OPENMP AND MPI

Threads and processes



Process

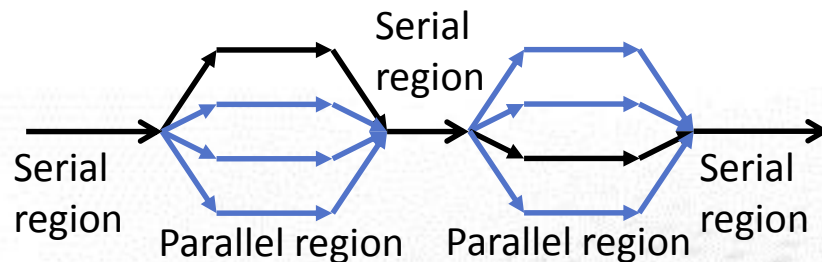
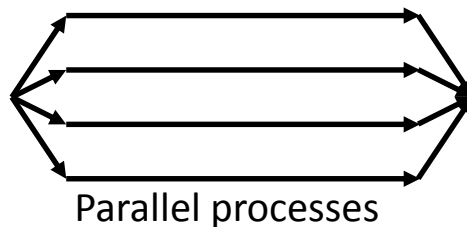
- Independent execution units
- Have their own state information and use their *own address spaces*



Thread

- A single process may contain multiple threads
- All threads within a process share the same state and *same address space*

Threads and processes



Process

- Spawned when starting the parallel program and killed when its finished
- Typically communicate using MPI in supercomputers

Thread

- Short-lived: threads are created by forking and destroyed by joining them
- Communicate directly through the shared memory

Three components of OpenMP

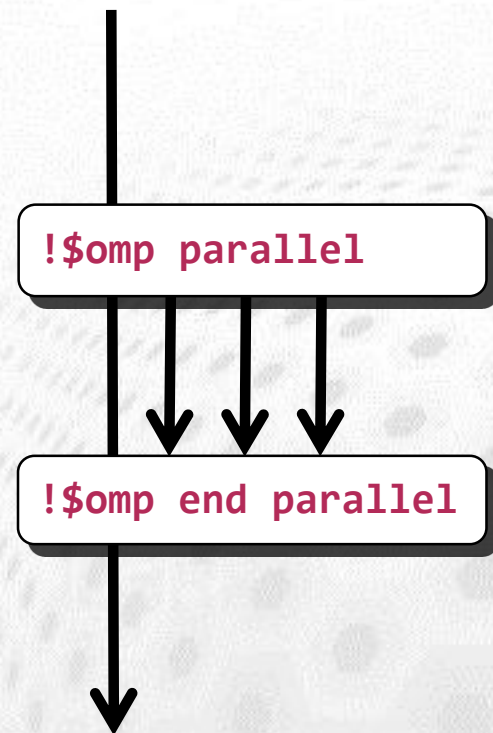
- Compiler directives and constructs
 - Expresses shared memory parallelization
 - Preceded by sentinel, can compile serial version
- Runtime library routines
 - Small number of library functions
 - Can be discarded in serial version via conditional compiling
- Environment variables
 - Specify the number of threads, etc.

OpenMP directives

- Sentinels precede each OpenMP directive
 - C/C++: `#pragma omp`
 - Fortran free form: `!$omp`
- Compilers that support OpenMP usually require an option (flag) that enables the feature
 - Without an enabling flag the OpenMP sentinels are treated as comments and a serial version will be compiled

Parallel construct

- ➡ Defines a *parallel region*
 - Prior to it only one thread, master
 - Creates a team of threads: master+slave threads
 - At end of the block is a barrier and all shared data is synchronized



Example: Helloworld with OpenMP

```

program hello
  use omp_lib
  integer :: omp_rank
  !$omp parallel private(omp_rank)
    omp_rank = omp_get_thread_num()
    print *, 'Hello world! by &
      thread ', omp_rank
  !$omp end parallel
end program hello
  
```

```

> ftn omp_hello.f90 -o omp
> setenv OMP_NUM_THREADS 4
> aprun -n 1 -d 4 ./omp
Hello world! by thread      0
Hello world! by thread      2
Hello world! by thread      3
Hello world! by thread      1
  
```

```

#include <stdio.h>
#include <omp.h>
int main(int argc, char argv[]){
  int omp_rank;
  #pragma omp parallel private(omp_rank)
  {
    omp_rank = omp_get_thread_num();
    printf("Hello world! by
      thread %d", omp_rank);
  }
}
  
```

```

> cc omp_hello.c -o omp
> setenv OMP_NUM_THREADS 4
> aprun -n 1 -d 4 ./omp
Hello world! by thread      2
Hello world! by thread      3
Hello world! by thread      0
Hello world! by thread      1
  
```


How do the threads interact?

- Because of the shared address space threads can “communicate” using *shared* variables
- Threads often need some *private* work space together with shared variables
 - For example the index variable of a loop
- Visibility of different variables is defined using *data-sharing clauses* in the parallel region definition
 - private, firstprivate, lastprivate, shared, default

Work sharing

- Parallel region creates an "Single Program Multiple Data" instance where each thread executes the same code
- How can one split the work between the threads of a parallel region?
 - Loop construct
 - Single/Master construct
 - Sections
 - Task construct (in OpenMP 3.0 and above)

Loop constructs

- ➊ Directive instructing compiler to share the work of a loop
 - Fortran: **\$OMP DO**
 - C/C++: **#pragma omp for**
 - Directive must be inside a parallel region
 - Can also be combined with parallel:
\$OMP PARALLEL DO / #pragma omp parallel for
- ➋ Loop index is private by default
- ➌ Work sharing can be controlled using *schedule* clause
 - static, dynamic, guided, or runtime

Reduction clause

reduction(operator:var_list)

- Performs reduction on the (scalar) variables in list
- Private reduction variable is created for each thread's partial result
- Private reduction variable is initialized to operator's initial value
- After parallel region the reduction operation is applied to private variables and result is aggregated to the shared variable

Execution controls

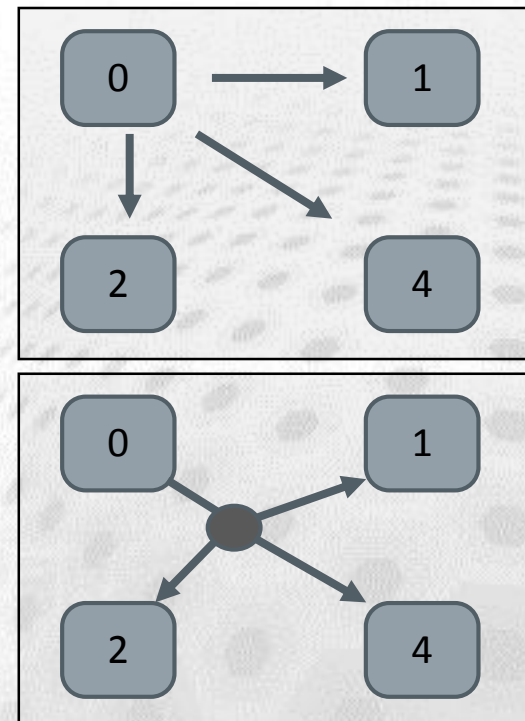
- Sometimes a part of parallel region should be executed only by the master thread or by a single thread at time
 - I/O, initializations, updating global values, etc.
 - Remember the synchronization!
- OpenMP provides clauses for controlling the execution of code blocks
 - barrier
 - master & single
 - critical

Execution model in MPI

- ➊ Parallel program is launched as set of *independent, identical processes*
 - The same program code and instructions
- ➋ MPI runtime assigns each process a *rank*
 - identification of the processes
 - Processes can perform different tasks and handle different data basing on their rank
 - Can reside in different nodes
- ➌ The way to launch parallel program is implementation dependent

Communication

- Data is local to the MPI processes
 - they need to *communicate* to coordinate work
- Point-to-point communication
 - Messages are sent between two processes
- Collective communication
 - Involving a number of processes at the same time



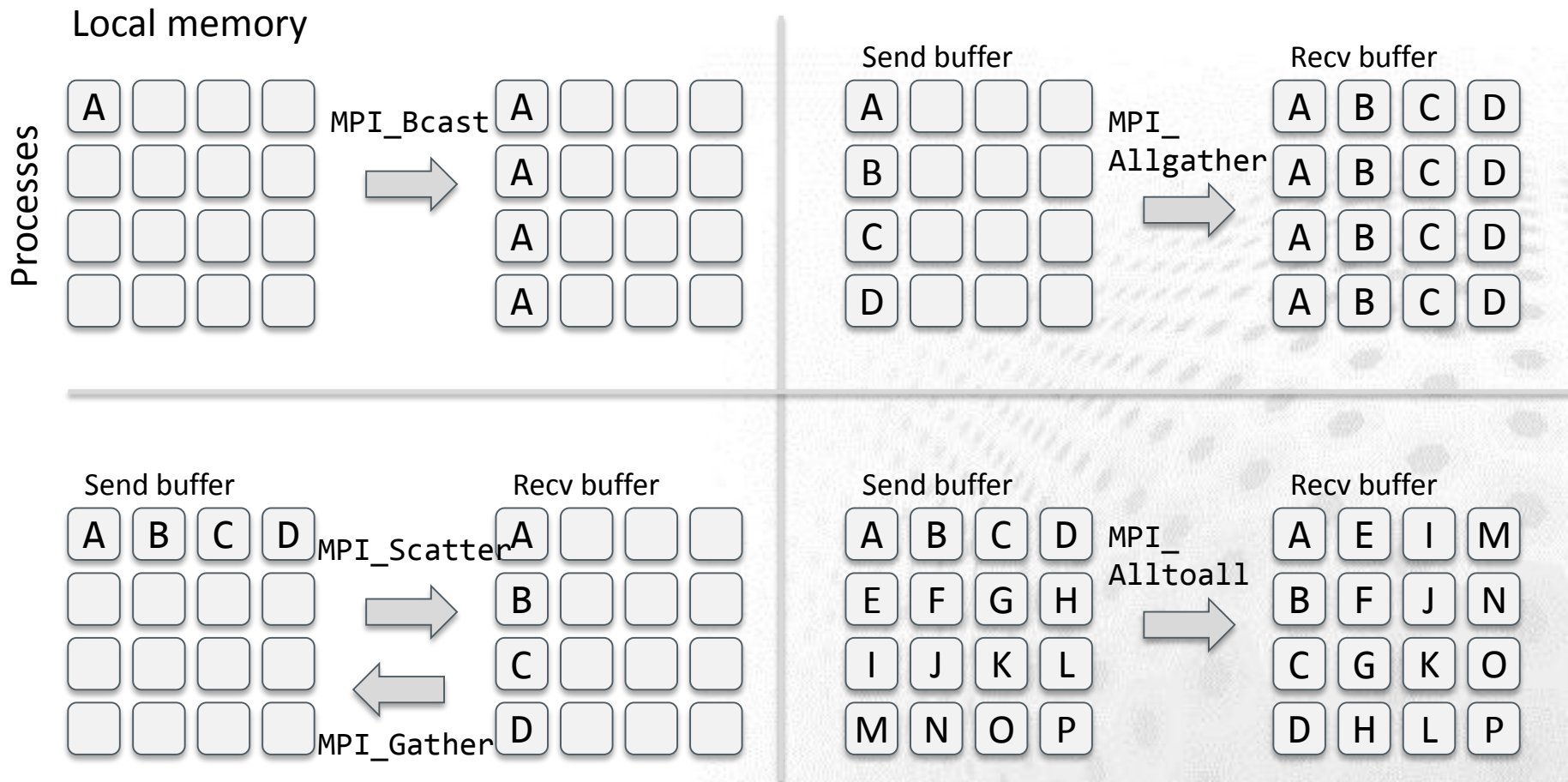
MPI point-to-point operations

- One process *sends* a message to another process that *receives* it with MPI_Send and MPI_Recv routines
- Sends and receives in a program should match – one receive per send
- Each message (envelope) contains
 - The actual *data* that is to be sent
 - The *datatype* of each element of data
 - The *number of elements* the data consists of
 - An identification number for the message (*tag*)
 - The ranks of the *source* and *destination*

Non-blocking communication

- Non-blocking communication is usually the smarter way to do point-to-point communication in MPI
 - Enables some computing concurrently with communication
 - Avoids many common dead-lock situations
- Non-blocking communication realization
 - `MPI_Isend`
 - `MPI_Irecv`
 - `MPI_Wait` / `MPI_Waitall`

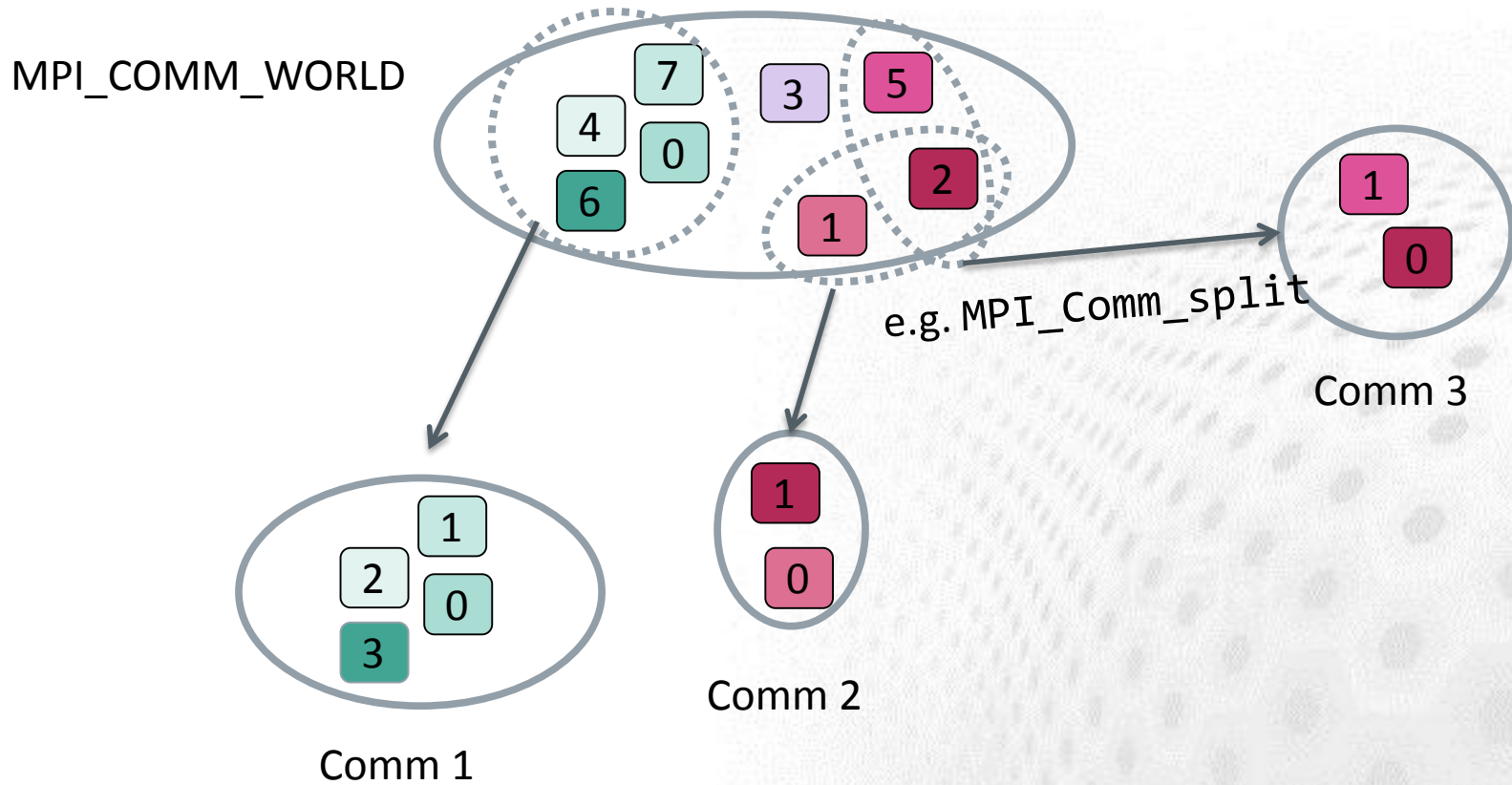
Collective operations examples



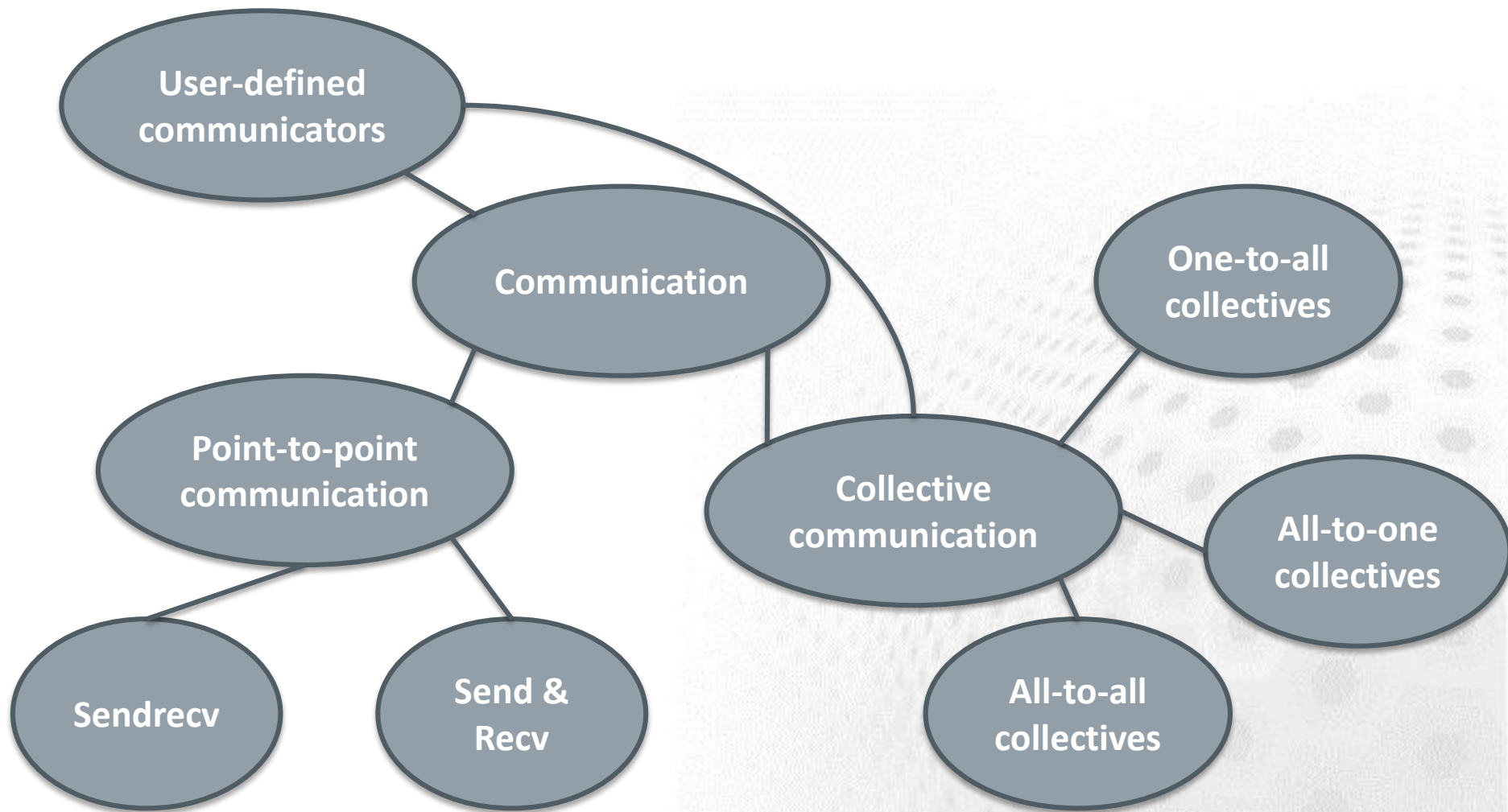
MPI datatypes

- MPI has a number of predefined datatypes to represent data
- Each C or Fortran datatype has a corresponding MPI datatype
 - C examples: MPI_INT for int and MPI_DOUBLE for double
 - Fortran example: MPI_INTEGER for integer
- One can also define custom datatypes

Communicators



Basic MPI summary



First five MPI commands

C & Fortran bindings

```
int MPI_Init(int *argc, char **argv)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Barrier(MPI_Comm comm)
MPI_Finalize()
```

```
MPI_INIT(ierr)
MPI_COMM_SIZE(comm, size, ierr)
MPI_COMM_RANK(comm, rank, ierr)
MPI_BARRIER(comm, ierr)
MPI_FINALIZE(ierr)
integer comm, size, rank, ierr
```


Send operation

➡ C/C++ binding

```
int MPI_Send(void *buffer, int count, MPI_Datatype  
             datatype, int dest, int tag, MPI_Comm comm)
```

- ➡ The return value of the function is the error value

➡ Fortran binding

```
MPI_SEND(buffer, count, datatype,  
          dest, tag, comm, ierror)
```

```
<type>, dimension(*) :: buf
```

```
integer :: count, datatype, dest, tag, comm, ierror
```

- ➡ **ierror**: the error value

Receive operation

• C/C++ binding

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype,  
             int source, int tag, MPI_Comm comm, MPI_Status *status )
```

• Fortran binding

```
mpi_recv(buf, count, datatype, source, tag, comm, status,  
         ierror)
```

```
<type>, dimension(*) :: buf
```

```
integer :: count, datatype, source, tag, comm, ierror
```

```
integer, dimension(MPI_STATUS_SIZE) :: status
```


MPI datatypes

<u>MPI type</u>	<u>C type</u>
MPI_CHAR	signed char
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_INT	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	

MPI datatypes

<u>MPI type</u>	<u>Fortran type</u>
MPI_CHARACTER	CHARACTER
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_REAL8	REAL*8 (nonstandard)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_DOUBLE_COMPLEX	DOUBLE COMPLEX
MPI_LOGICAL	LOGICAL
MPI_BYTE	

Combined send & receive

• C/C++ binding

```
int MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype  
    sendtype, int dest, int sendtag, void *recvbuf, int  
    recvcount, MPI_Datatype recvtype, int source, int  
    recvtag, MPI_Comm comm, MPI_Status *status )
```

• Fortran binding

```
mpi_sendrecv(sendbuf, sendcount, sendtype, dest, sendtag,  
    recvbuf, recvcount, recvtype, source, recvtag, comm,  
    status, ierror)  
<type>, dimension(*) :: sendbuf, recvbuf  
integer :: sendcount, sendtype, dest, sendtag, recvcount,  
    recvtype, source, recvtag, comm, ierror  
integer, dimension(MPI_STATUS_SIZE) :: status
```


Non-blocking send

➤ C/C++ binding

```
int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int  
              dest, int tag, MPI_Comm comm, MPI_Request *request )
```

➤ Fortran binding

```
MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)  
<type> :: BUF(*)  
INTEGER :: COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR
```


Non-blocking receive

• C/C++ binding

```
int MPI_Irecv( void *buf, int count, MPI_Datatype datatype,  
              int source, int tag, MPI_Comm comm, MPI_Request *request  
              )
```

• Fortran binding

```
MPI_IRecv(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM,  
          REQUEST, IERROR)  
  
<type> :: BUF(*)  
INTEGER :: COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST,  
          IERROR
```


Wait for non-blocking operation

- C/C++ binding

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

- Fortran binding

```
MPI_WAIT(REQUEST, STATUS, IERROR)
```

```
INTEGER :: REQUEST, STATUS(MPI_STATUS_SIZE), IERROR
```


Wait for non-blocking operations

➤ C/C++ binding

```
int MPI_Waitall(int count, MPI_Request  
               *array_of_requests, MPI_Status *array_of_statuses)
```

➤ Fortran binding

```
MPI_WAITALL(COUNT, ARRAY_OF_REQUESTS, ARRAY_OF_STATUSES,  
            IERROR)
```

```
INTEGER :: COUNT, ARRAY_OF_REQUESTS(:),  
        ARRAY_OF_STATUSES(MPI_STATUS_SIZE,:), IERROR
```


Creating a communicator

• C and Fortran bindings

```
int MPI_Comm_split (MPI_Comm comm, int color, int key,
                   MPI_Comm newcomm)
```

```
MPI_COMM_SPLIT (comm, color, key, newcomm, rc)
integer :: comm, color, key, newcomm, rc
```

• Return code values

MPI_SUCCESS No error; MPI routine completed successfully.

MPI_ERR_COMM Invalid communicator. A common error is to use

a null communicator in a call

MPI_ERR_INTERN This error is returned when some part of the

implementation is unable to acquire memory.

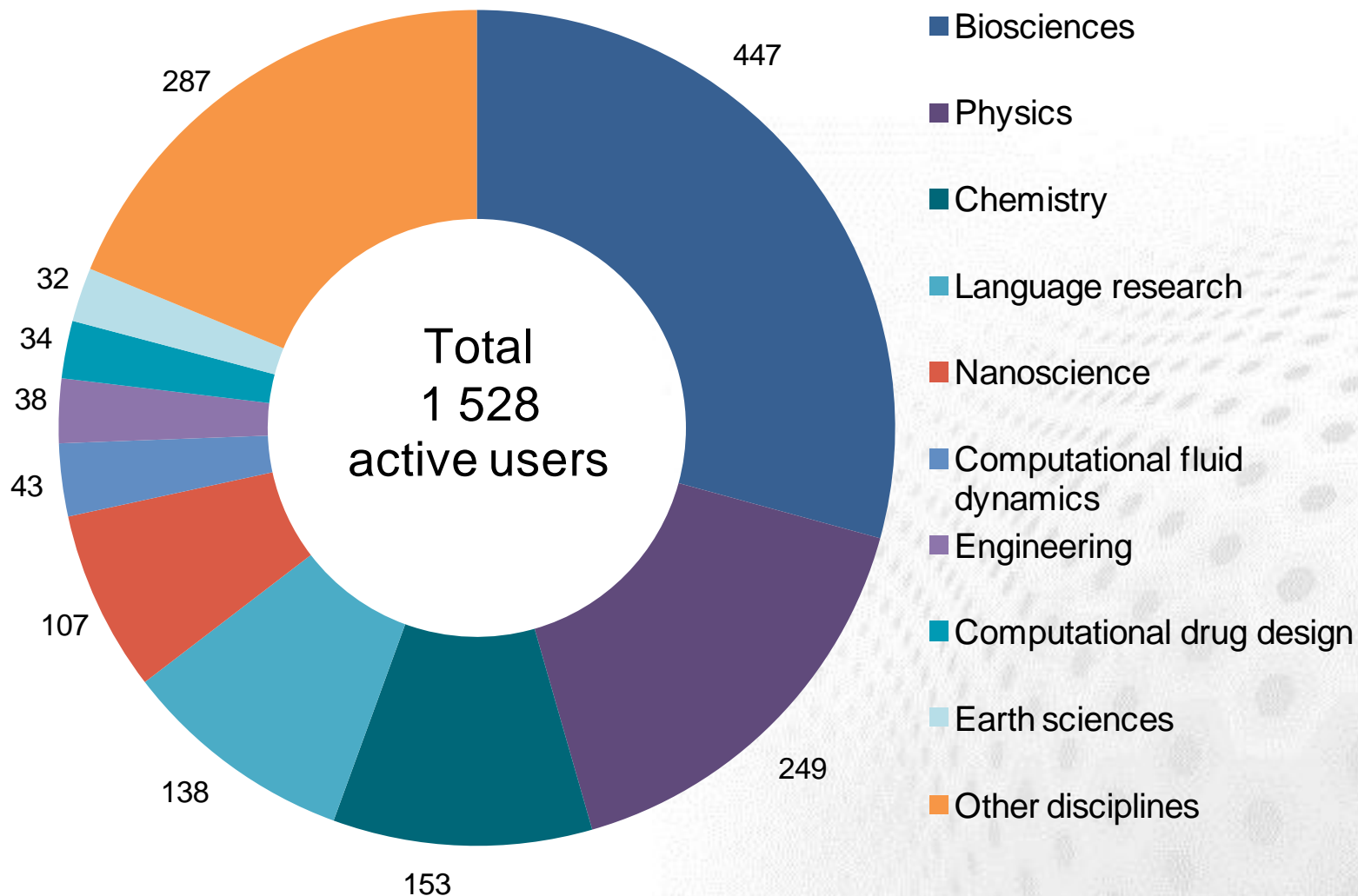
Time for hands-on



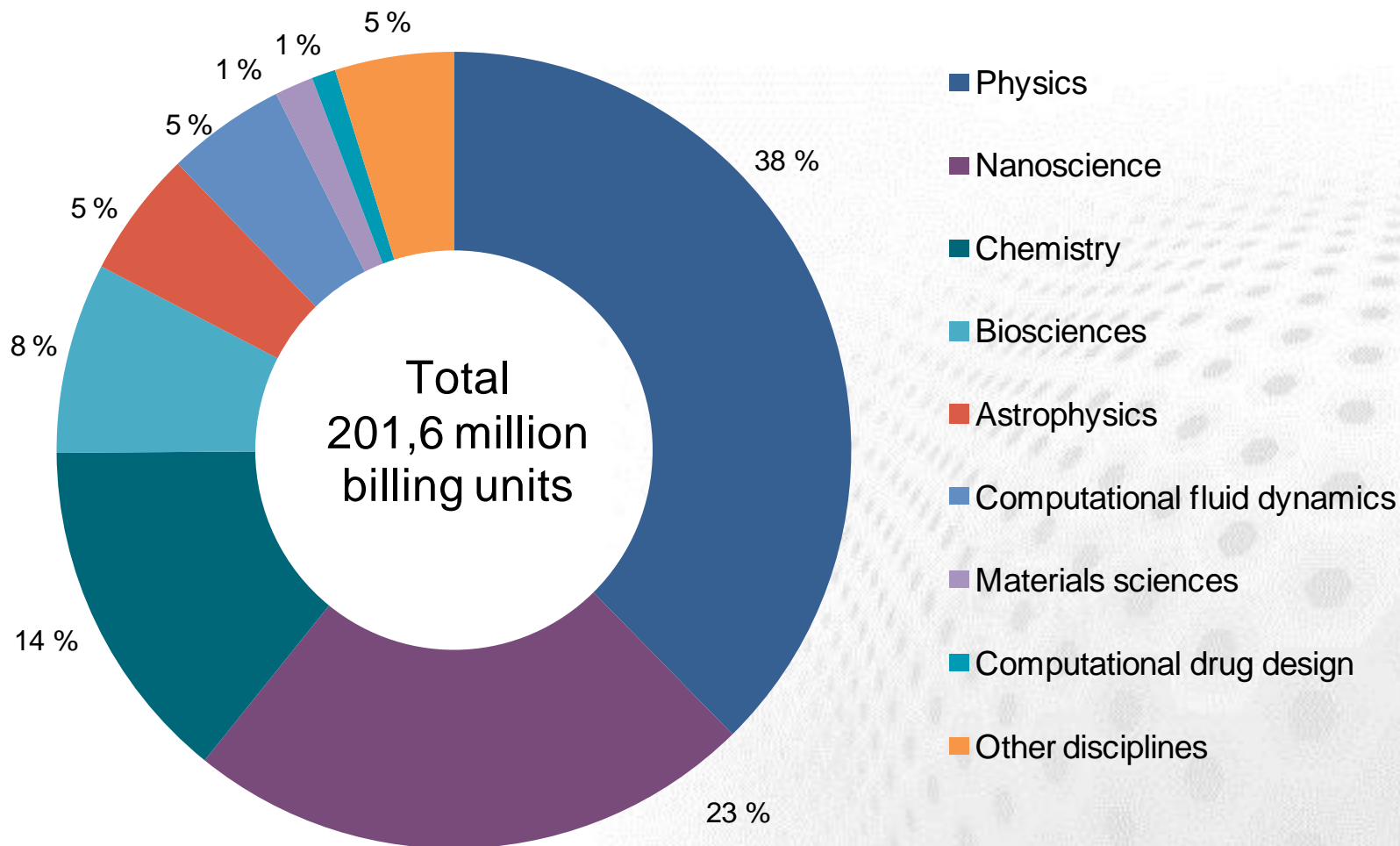
Applications for Physicists

Jussi Enkovaara
High Performance Computing support

Users of computing resources by discipline 2013

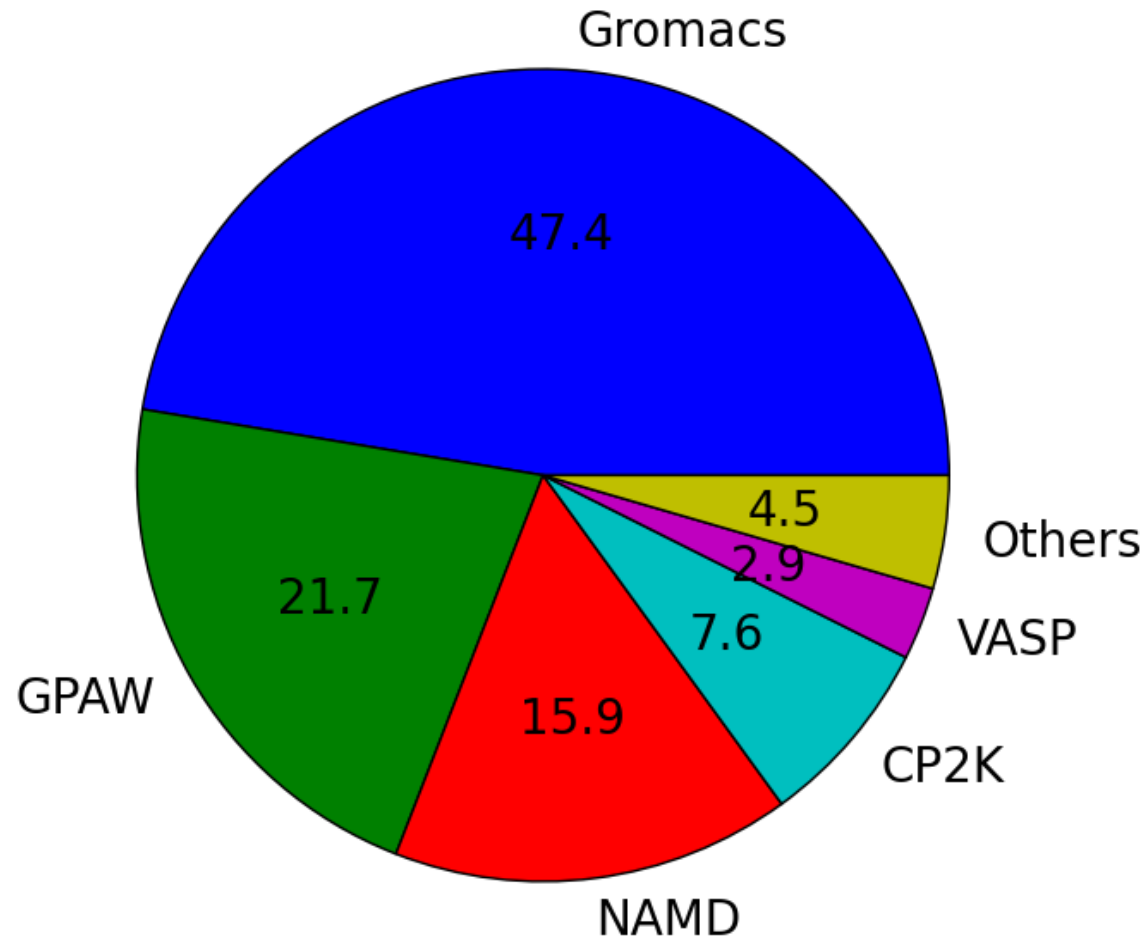


Computing usage by discipline 2013



Software usage 2014

➤ Software maintained by CSC



CLASSICAL MOLECULAR DYNAMICS

Molecular dynamics

- Numerical integration of Newton's law: $\mathbf{F} = m\mathbf{a}$
- Forces can be calculated quantum mechanically (*ab initio* MD) or from classical force fields

$$\mathbf{F}_i(t) = -\nabla_i V(\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n; t)$$

- The form of the force field (or potential) V is chosen to represent physics of the problem
 - Empirically parameterized
 - Large variety of force fields exists

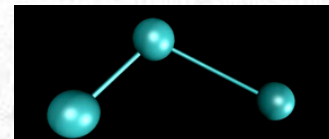
Classical force fields

Typical terms in force fields:

Torsional $V(\phi) = \sum_n K_n \cos(n\phi + \delta)$

Vibrational $V(\mathbf{r}_i, \mathbf{r}_j) = \frac{1}{2}k(\mathbf{r}_i - \mathbf{r}_j)^2$

Coulombic $V(\mathbf{r}_i, \mathbf{r}_j) = \frac{q_i q_j}{r_{ij}}$



Effects of surroundings (temperature, pressure etc.)

- Canonical ensemble, isothermal-isobaric ensemble, ...
- Thermostats

Molecular dynamics

- Basic outcome is the time evolution of system of atoms (positions and velocities) *i.e* sample of phase space
- Different distribution functions provide information about the system
- Collect statistics to obtain representative ensemble of the phase space
- Typically length and time scales in MD
 - $10^4 - 10^6$ atoms, time up to μs

Molecular models

- All atoms
 - OPLS-AA/L, CHARM, AMBER
 - Each atom is treated as particle
- Coarse grained superatoms
 - Group of atoms (*i.e.* 4 is treated as particle)

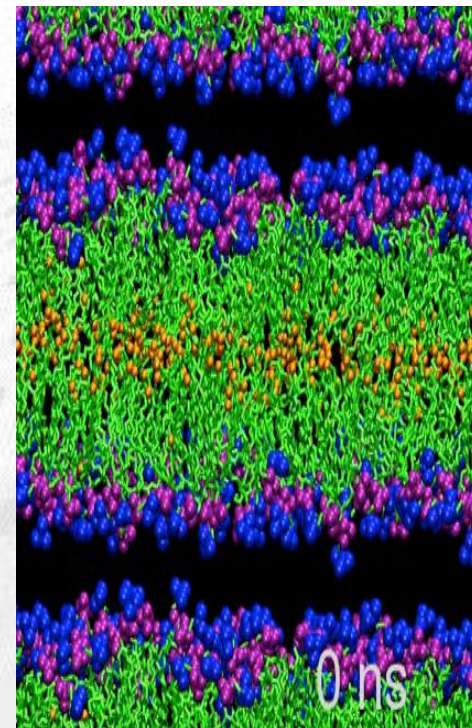
MD software at CSC

➤ Gromacs

- Widely used open source software package
- Mainly biomolecules
- Different molecular models and algorithms

➤ NAMD

- Main emphasis also on biomolecules
- Massively parallel
- Freeware



QUANTUM MECHANICS

Interacting electrons and nuclei

- Nanosystems: 1 – 100 Å
- Nuclei as point particles with charge Z_a and mass M_a (nuclear radii $< 10^{-4}$ Å)
- Hamiltonian for electron-nuclei system

$$H = -\sum_i \frac{\nabla_i^2}{2} + \frac{1}{2} \sum_{i \neq j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{i,a} \frac{Z_a}{|\mathbf{r}_i - \mathbf{R}_a|} - \sum_a \frac{\nabla_a^2}{2M_a} + \frac{1}{2} \sum_{a \neq a'} \frac{Z_a Z_{a'}}{|\mathbf{R}_a - \mathbf{R}_{a'}|}$$

Atomic units: $\hbar = m = e = \frac{4\pi}{\epsilon_0} = 1$

Born-Oppenheimer approximation

- Nuclei are much heavier than electrons ($M_i > 10^3$)
- Electronic time scales are often significantly shorter than the nuclear ones
- Decouple the dynamics of electrons and nuclei
- Electronic Hamiltonian:

$$H = - \sum_i \frac{\nabla_i^2}{2} + \frac{1}{2} \sum_{i \neq j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{i,a} \frac{Z_a}{|\mathbf{r}_i - \mathbf{R}_a|}$$

Many-body Schrödinger equation

$$H = - \sum_i \frac{\nabla_i^2}{2} + \frac{1}{2} \sum_{i \neq j} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_i V_{ext}(\mathbf{r}_i)$$

$$H\Psi(\mathbf{r}_1, \dots, \mathbf{r}_N) = E\Psi(\mathbf{r}_1, \dots, \mathbf{r}_N)$$

- Can be solved analytically for single electron
- Storing 8 electron wavefunction in 6x6x6 cartesian grid requires **$\sim 10^{10}$ GB !**

Wave-function based methods

- Hartree-Fock approximation: many-body wave-function as determinant of single particle orbitals
 - Often only qualitatively correct
- Post-Hartree-Fock methods: configuration integration (CI), coupled cluster (CC), Møller–Plesset perturbation theory (MP2, MP3, MP4)
 - Accurate, computational scaling $O(N^m)$, $m > 4$
 - Only small molecules

Wave-function based methods

- Physical quantities

- Formation and dissociation energies, geometry optimizations, excited state energies, various spectra

- Software packages at CSC

- Turbomole, Gaussian, Molpro, NWChem
- Typically not massively parallel, up to few tens of CPU cores

Density-functional theory

- Hohenberg-Kohn theorems
 - The ground state properties of many-electron system are unique functionals of the ground state density $n(r)$
 - The ground state density minimizes the energy functional
- Density depends only on three spatial variables
- The exact energy functional is not known but must be approximated

Kohn-Sham equations

- Formulate problem in terms of single particle orbitals $\left(-\frac{\nabla^2}{2} + v_s(\mathbf{r})\right) \psi_i(\mathbf{r}) = e_i \psi_i(\mathbf{r})$

$$v_s[n](\mathbf{r}) = V_{ext}(\mathbf{r}) + V_H[n](\mathbf{r}) + V_{xc}[n](\mathbf{r})$$

$$V_H = \int \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' \quad V_{xc} = \frac{\delta E_{xc}[n]}{\delta n(\mathbf{r})}$$

$$n(\mathbf{r}) = \sum_i \psi_i^*(\mathbf{r}) \psi_i(\mathbf{r})$$

- Non-linear problem, computational scaling $O(N^3)$

Density-functional theory

• Physical quantities

- Energetics, geometry optimization, elastic constants, phonons, ...
- Analysis of electronic structure
- Excited state properties with time-dependent density-functional theory

• Accuracy depends on the exchange-correlation approx.

- Local density approximation, generalized gradient approximation, ...

Numerical solution

- The strong Coulomb potential may be replaced with a smoother one
 - Pseudopotential vs. all-electron methods
- Wave-functions (or Kohn-Sham orbitals) are expanded in a basis:

$$\psi_i = \sum_n C_{i,n} \Phi_n$$

- Matrix equations:

$$\begin{aligned} \mathbf{HC} &= e\mathbf{SC} \\ H_{ij} &= \langle \Phi_i | \hat{H} | \Phi_j \rangle \\ H_{ij} &= \langle \Phi_i | \Phi_j \rangle \end{aligned}$$

Plane-wave basis

- Periodic functions can be expanded in plane waves

$$u_{n\mathbf{k}}(\mathbf{r}) = \frac{1}{\Omega^{1/2}} \sum_{\mathbf{G}} C_{\mathbf{G},n\mathbf{k}} e^{i\mathbf{G}\cdot\mathbf{r}} \Rightarrow \psi_{n\mathbf{k}}(\mathbf{r}) = \frac{1}{\Omega^{1/2}} \sum_{\mathbf{G}} C_{\mathbf{G},n\mathbf{k}} e^{i(\mathbf{G}+\mathbf{k})\cdot\mathbf{r}}$$

$$n(\mathbf{r}) = \sum_{\mathbf{G}} n_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}}$$

$$V(\mathbf{r}) = \sum_{\mathbf{G}} V_{\mathbf{G}} e^{i\mathbf{G}\cdot\mathbf{r}}$$

- Relies on fast Fourier transforms
- Systematic convergence with the number of plane waves
- Requires pseudopotential approximation

Localized atomic orbital basis

- Use chemical insight for constructing basis
 - Radial solution of isolated atom
 - Gaussian basis
 - Slater basis
- Nomenclature: single-zeta (SZ), double-zeta (DZ), double-zeta + polarized (DZP), ...
- Systematic convergence of basis can be difficult
- All-electron or pseudopotential methods

Real-space grids

- Represent wave-functions, potentials, densities etc. on real-space grid
- Derivatives with finite-differences
- Systematic convergence with grid spacing
- Requires pseudopotential approximation

Comparison on basis sets

- Plane waves
 - systematic convergence with single parameter
 - parallelization more limited due FFTs
- Localized basis set
 - compact basis
 - systematic convergence can be difficult
- Real-space grids
 - systematic convergence with single parameter
 - good parallelization prospects

DFT software packages



● GPAW

- Real-space grids, plane waves, atomic orbital basis
- Projector augmented wave approximation
- Time-dependent density-functional theory
- Good parallelization in real-space mode (> 10 000 CPU cores)
- Flexible Python interface, simple GUI
- Open source

DFT software packages

➤ VASP

- Plane waves
- Projector augmented wave approximation
- Standard ground state features, some excited state functionality
- Widely used, stable software package
- Parallelization up to few hundreds of CPU cores
- Requires a license



DFT software packages

➤ CP2K

- Mixed Gaussian and plane wave basis
- Norm conserving pseudopotentials
- Well suited for ab-initio molecular dynamics of insulating systems
- Good parallelization (thousands of CPU cores) depending on usage mode
- Open source



User interfaces for DFT software

- Atomic simulation environment (ASE)
 - Python interface for setting up atomic structures and other non-computational intensive tasks
 - Can be used with several “computational engines” (GPAW, VASP, ...)
 - Simple GUI
 - Open source, runs either locally or remotely (Linux, Mac, Windows)

User interfaces for DFT software

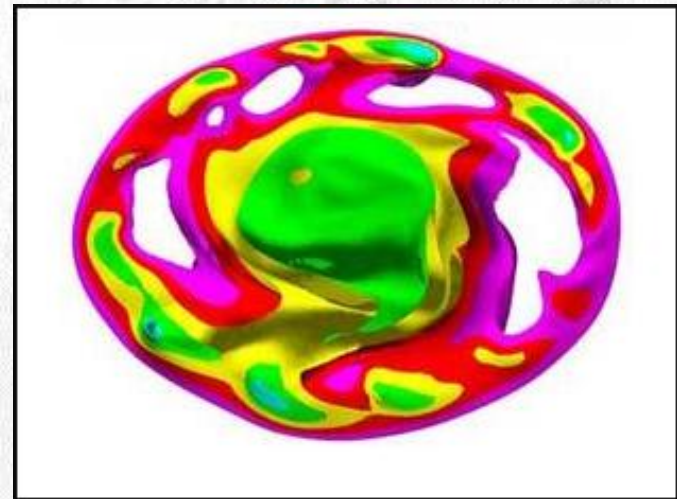
● Materials Studio

- Extensive graphical user interface for materials simulations
- Setting up of structures, performing calculations, analysis
- DFT calculations with CASTEP (plane-wave pseudopotential code), also several classical simulation methods
- Proprietary, user interface only for Windows

OTHER SOFTWARE

Other software

- Elmer – multiphysical finite-element software
 - Fluid dynamics, structural mechanics, electromagnetics, heat transfer, acoustics, ...
 - Open source



Other software

• Mathematica

- Symbolic mathematics (including derivatives and integrals)
- Visualization

• Matlab

- Numerical mathematics especially with matrices
- Toolboxes for specific tasks

• Python (+ NumPy + SciPy + matplotlib + ...)

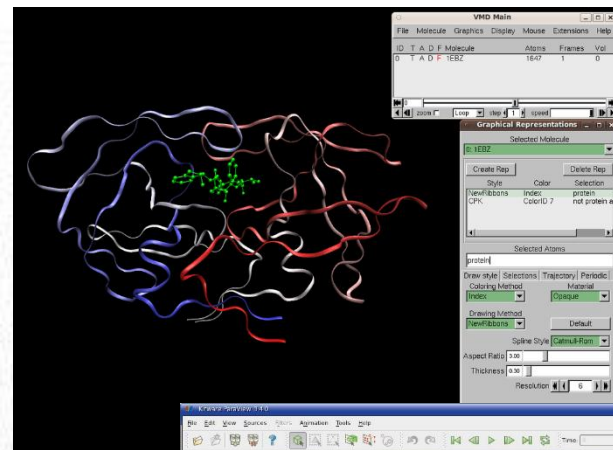
- General purpose programming, numeric, and visualization

Other software



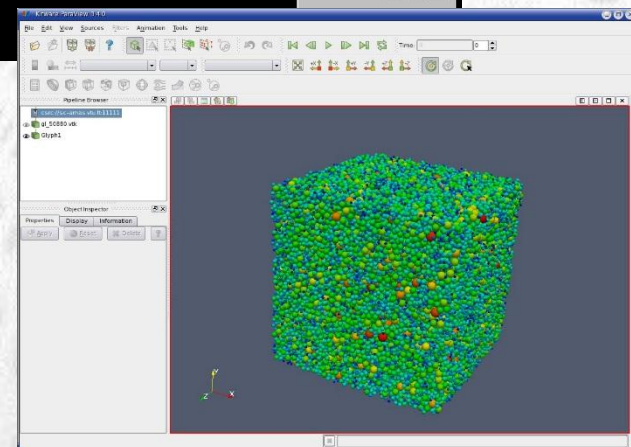
VMD

- Molecular visualization with balls, sticks, ribbons etc.
- Isosurfaces



ParaView

- Analysis and visualization
- Interactive or batch processing



Summary

- CSC offers large selection of software suitable for physicists
 - Chemistry databases can also be useful
- No single “best” tool
- See research.csc.fi/software for full listing of available software



Coding



Debugging and code optimization

Debugging

- Debugging is inevitable but often difficult
- Naive approach: "print *, 'foo 1' "
- Parallel debuggers can be of great help
 - Totalview
 - LGDB
 - Other, e.g., DDT, gdb, ...

Debugging demo

Code optimization

➡ Obvious benefits

- Better throughput => more science
- Cheaper than new hardware
- Save energy, compute quota etc.

➡ ..and some non-obvious ones

- Collaboration opportunities
- Potential for cross-disciplinary research
- Deeper understanding of application

Code optimization

- Several trends making code optimization even more important
 - More and more cores
 - CPU's vector units getting wider
 - The gap between CPU and memory speed ever increasing
 - Datasets growing rapidly but disk I/O performance lags behind

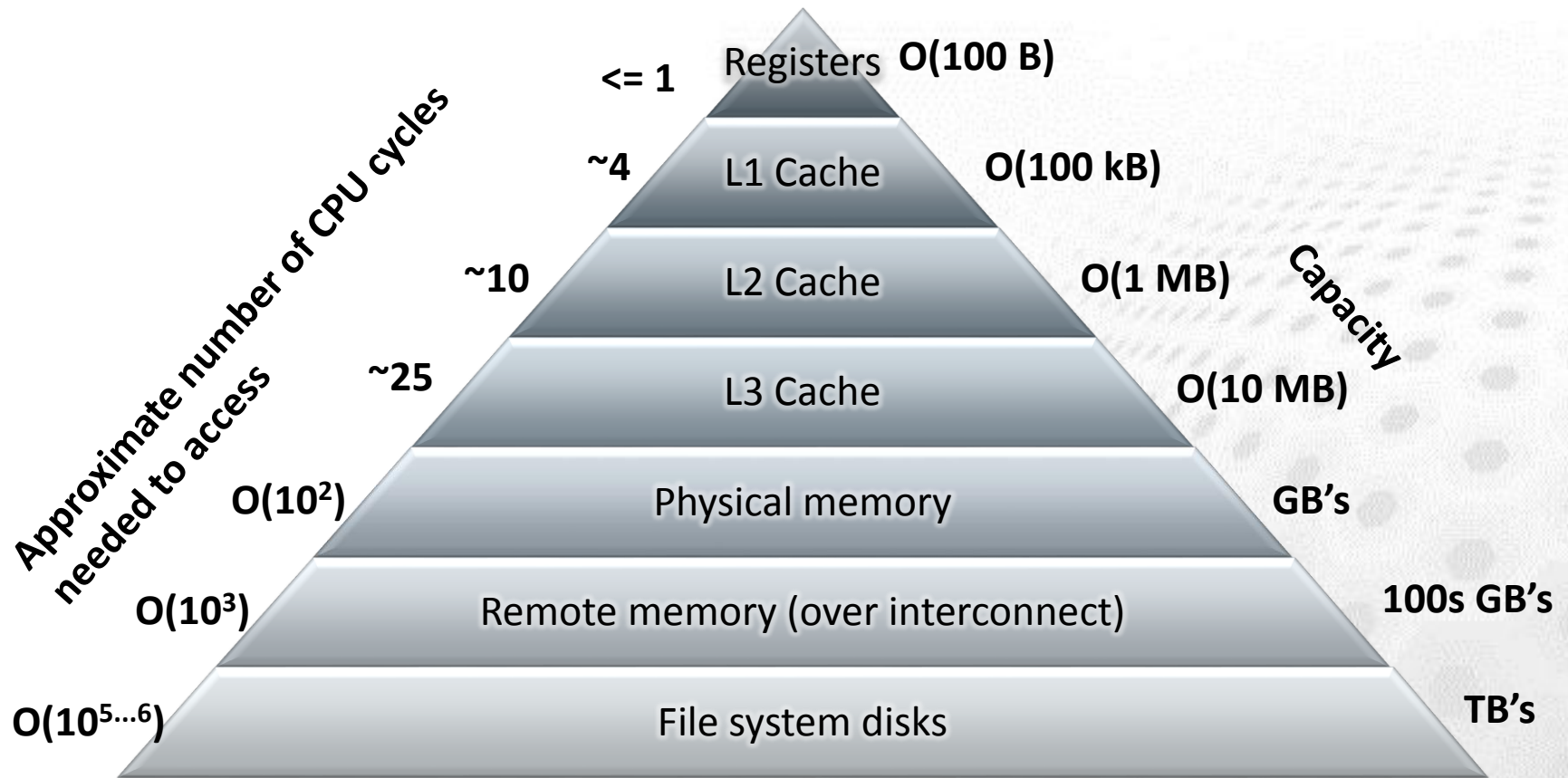
Code optimization

- Adapting the problem to the underlying hardware
- Combination of many aspects
 - Effective algorithms
 - Implementation: Processor utilization & efficient memory use
 - Parallel scalability
- Important to understand interactions
 - Algorithm – code – compiler – libraries – hardware
- Performance is not portable!

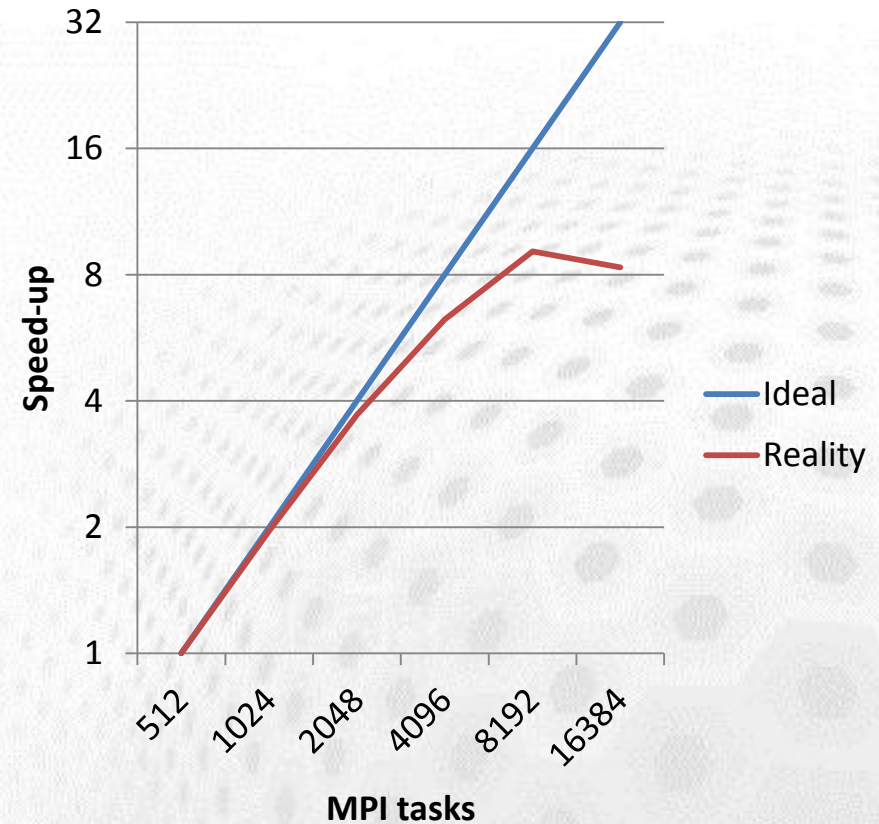
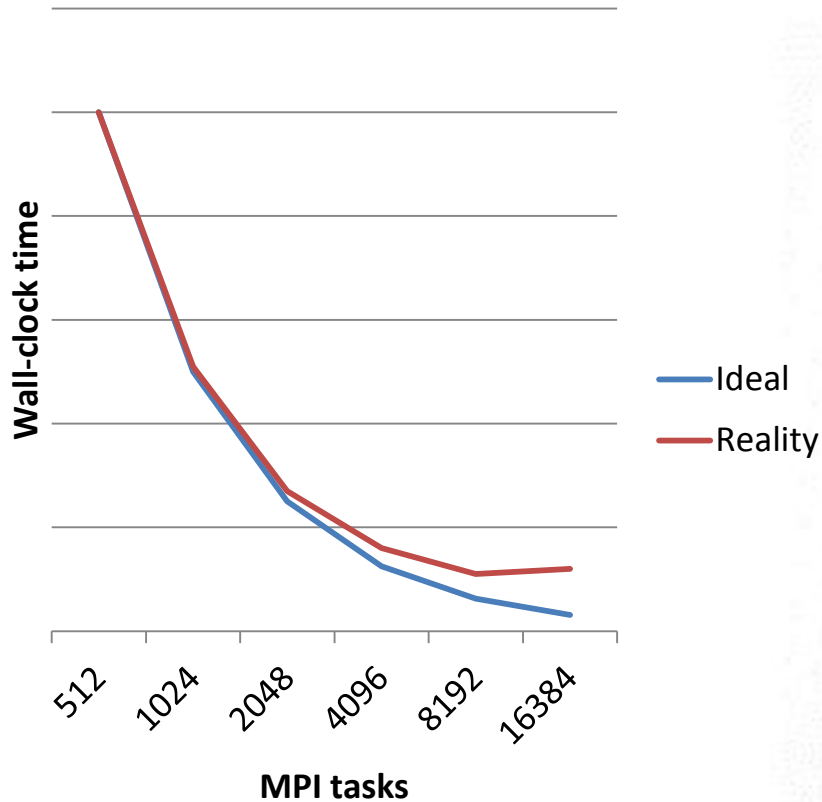
Not going to touch the source code?

- Find the *compiler* and its *compiler flags* that yield the best performance
- Employ *tuned libraries* wherever possible
- Find suitable settings for *environment parameters*
- Mind the *I/O*
 - Do not checkpoint too often
 - Do not ask for the output you do not need

Memory hierarchy



Why does scaling end?



Why does scaling end?

- Amount of data per process small - computation takes little time compared to communication
- Amdahl's law in general
 - E.g., single-writer or stderr I/O
- Load imbalance
- Communication that scales badly with N_{proc}
 - E.g., all-to-all collectives
- Congestion on network – too many messages or lots of data

Application timing



- Most basic information: total wall clock time
 - Built-in timers in the program (e.g. MPI_Wtime)
 - System commands (e.g. time) or batch system statistics
- Built-in timers can provide also more fine-grained information
 - Have to be inserted by hand
 - Typically no information about hardware related issues
 - Information about load imbalance and communication statistics of parallel program is difficult to obtain

Performance analysis tools

- ➊ *Instrumentation of code*
 - Adding special measurement code to binary
 - Normally all routines do not need to be measured
- ➋ *Measurement*: running the instrumented binary
 - Profile: sum of events over time
 - Trace: sequence of events over time
- ➌ *Analysis*
 - Text based analysis reports
 - Visualization

Profiling

- Purpose of the profiling is to find the "hot spots" of the program
 - Usually execution time, also memory
- Usually the code has to be recompiled or relinked, sometimes also small code changes are needed
- Often several profiling runs with different techniques is needed
 - Identify the hot spots with one approach, identify the reason for poor performance

Profiling: sampling

The application execution is interrupted at constant intervals and the program counter and call stack is examined

Pros

- ➡ Lightweight
- ➡ does not interfere the code execution too much

Cons

- ➡ Not always accurate
- ➡ Difficult to catch small functions
- ➡ Results may vary between runs

Profiling: tracing

Hooks are added to function calls (or user-defined points in program) and the required metric is recorder

Pros

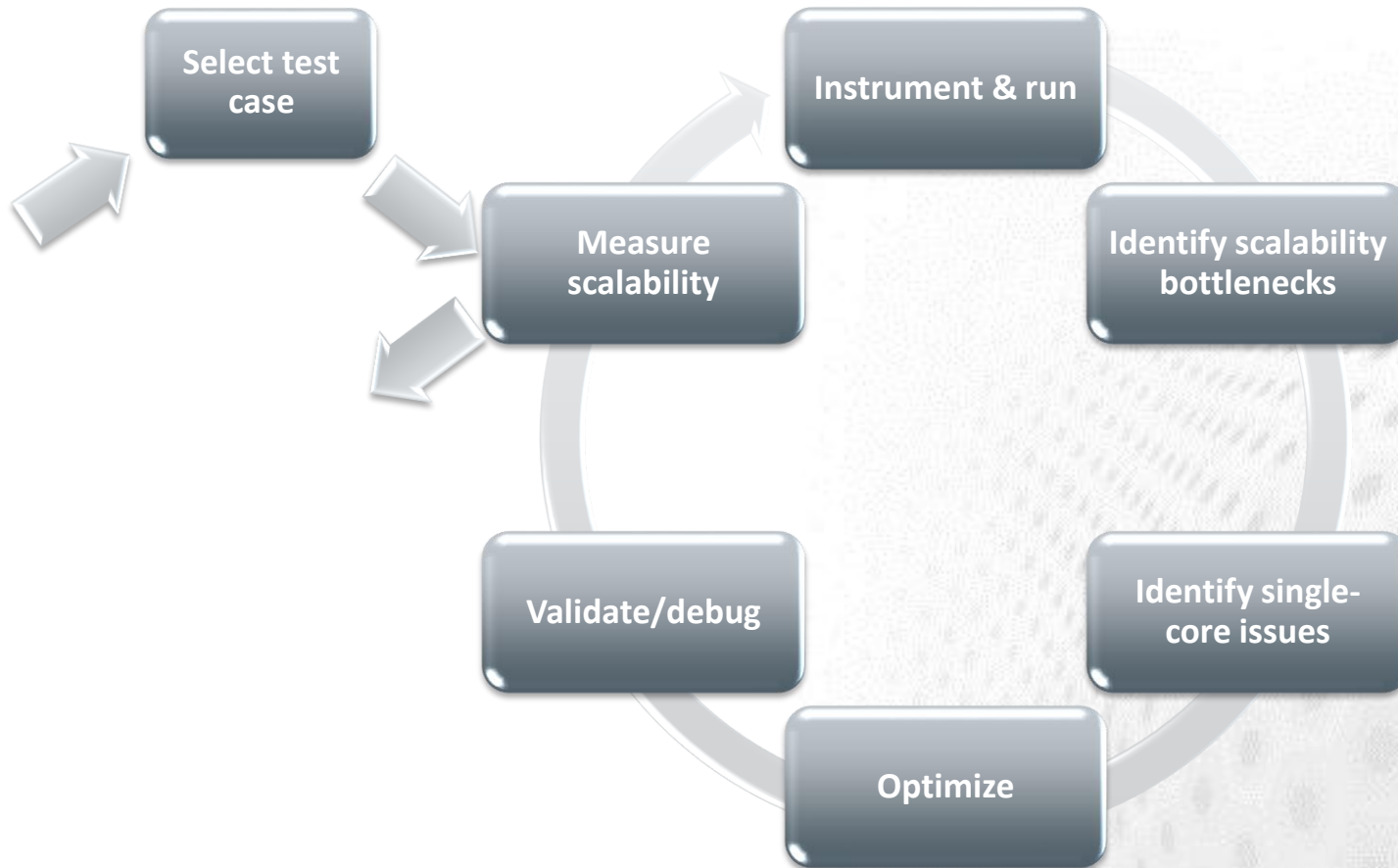
- ➡ Can record the program execution accurately and repeatably

Cons

- ➡ More intrusive
- ➡ Can produce prohibitely large log files
- ➡ May change the performance behaviour of the program

CODE OPTIMIZATION CYCLE

Code optimization cycle



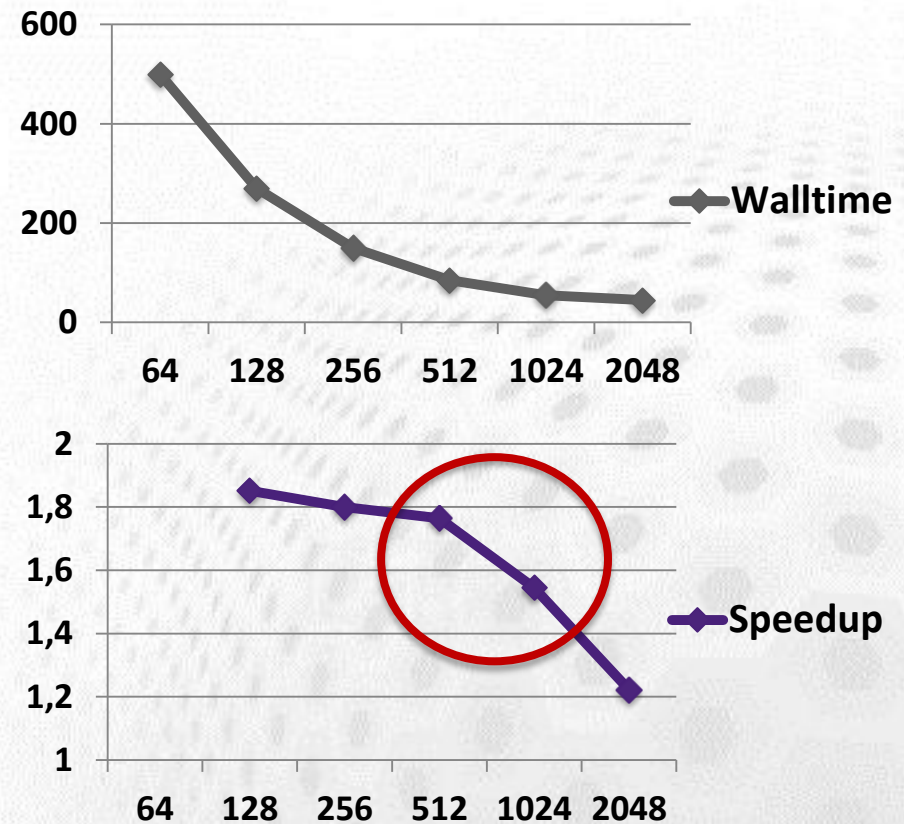
Step 1: Choose a test problem



- The dataset used in the analysis should
 - Make sense, i.e. resemble the intended use of the code
 - Be large enough for getting a good view on scalability
 - Be runnable in a reasonable time
 - For instance, with simulation codes almost a full-blown model but run only for a few time steps
- Should be run long enough that initialization/finalization stages are not exaggerated
 - Alternatively, we can exclude them during the analysis

Step 2: Measure scalability

- ➡ Run the uninstrumented code with different core counts and see where the parallel scaling stops
- ➡ Often we look at strong scaling
 - Also weak scaling is definitely of interest



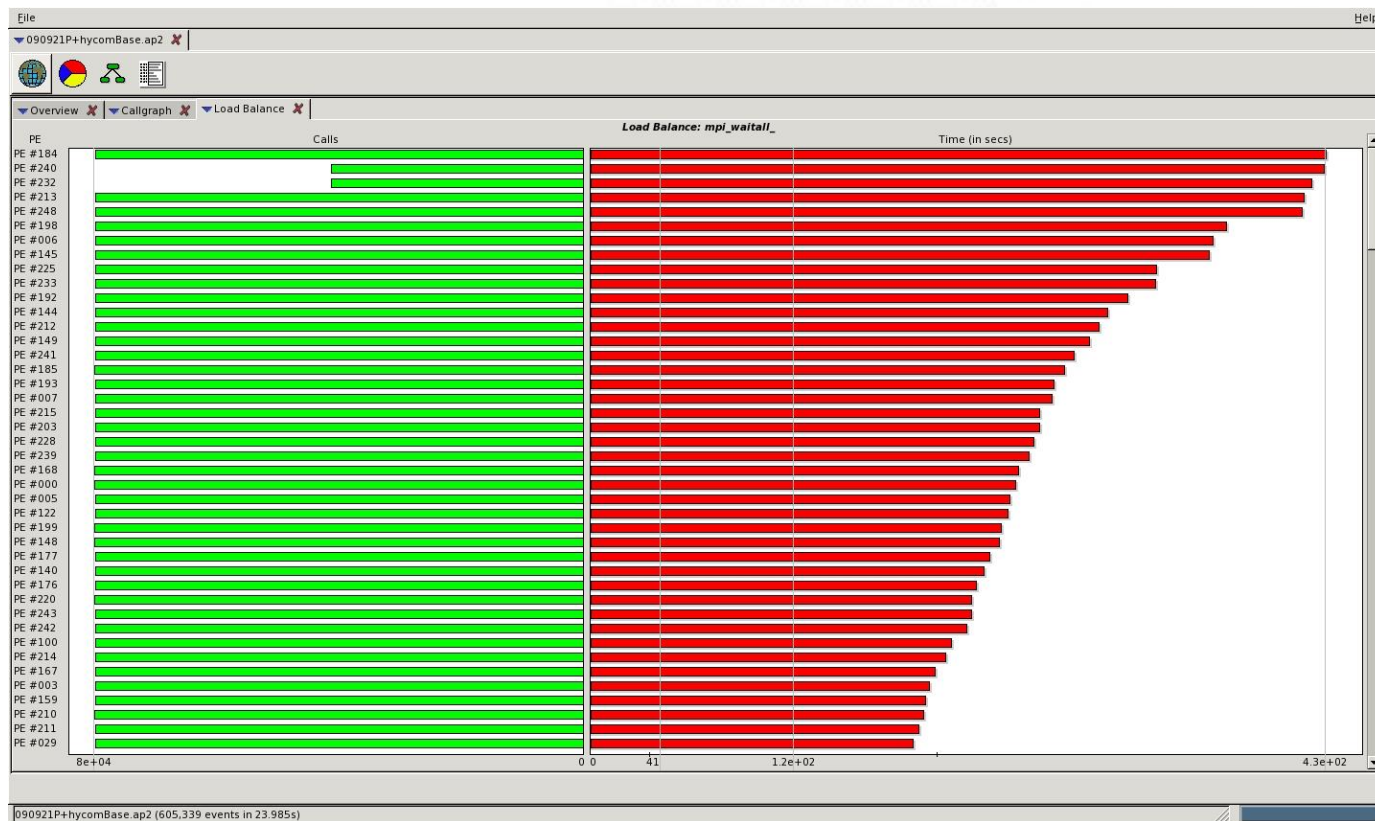
Step 3: Instrument & run

- Obtain first a sampling profile to find which user functions should be traced
 - With a large/complex software, one should not trace them all: it causes excessive overhead
 - Tracing also e.g. MPI, I/O and library (BLAS, FFT,...) calls
 - Execute and record the first analysis with
 - The core count where the scalability is still ok
 - The core count where the scalability has ended
- and identify the largest differences between these profiles

Step 4: Identify scalability bottlenecks

- ➊ What communication pattern and routines are dominating the true time spent for communication (excluding the sync times)?
- ➋ How does the message-size profile look like?
- ➌ Note that the analysis tools may report load imbalances as "real" communication
 - Put an MPI_Barrier before the suspicious routine - load imbalance will aggregate into it

Example with CrayPAT



Example with CrayPAT

Table 4: MPI Message Stats by Caller

	MPI Msg Bytes	MPI Msg Count	MsgSz <16B Count	4KB<= MsgSz <64KB Count	Function Caller PE[mmm]
	15138076.0	4099.4	411.6	3687.8	Total
	15138028.0	4093.4	405.6	3687.8	MPI_ISEND
	8080500.0	2062.5	93.8	1968.8	calc2_ MAIN_
3					
4	8216000.0	3000.0	1000.0	2000.0	pe.0
4	8208000.0	2000.0	--	2000.0	pe.9
4	6160000.0	2000.0	500.0	1500.0	pe.15
	=====				
...					

Step 4: Identify scalability bottlenecks

- Signature: User routines scaling but MPI time blowing up
 - Issue: Not enough to compute in a domain
 - Weak scaling could still continue
 - Issue: Expensive collectives
 - Issue: Communication increasing as a function of tasks
- Signature: MPI_Sync times increasing
 - Issue: Load imbalance
 - Tasks not having a balanced role in communication?
 - Tasks not having a balanced role in computation?
 - Synchronous (single-writer) I/O or stderr I/O?

Step 5: Find single-core hotspots



- Remember: pay attention only to user routines that consume significant portion of the total time
- Collect the key hardware counters, for example
 - L1 and L2 cache metrics (PAT_RT_PERFCTR=2)
 - use of vector (SSE/AVX) instructions (PAT_RT_PERFCTR=13)
 - Computational intensity (= ratio of floating point ops / memory accesses) (PAT_RT_PERFCTR=1, default)
- Trace the “math” group to see if expensive operations (exp, log, sin, cos,...) have a significant role

Step 5: Find single-core hotspots

- Signature: Low L1 and/or L2 cache hit ratios
 - <96% for L1, <99% for L1+L2
 - Issue: Bad cache alignment
- Signature: Low vector instruction usage
 - Issue: Non-vectorizable (hotspot) loops
- Signature: Traced "math" group featuring a significant portion in the profile
 - Issue: Expensive math operations

The **Golden Rules** of profiling

- **Profile your code**
 - The compiler/runtime will not do all the optimisation for you.
- **Profile your code yourself**
 - Don't believe what anyone tells you. They're wrong.
- **Profile on the hardware you want to run on**
 - Don't profile on your laptop if you plan to run on a Cray system.
- **Profile your code running the full-sized problem**
 - The profile will almost certainly be qualitatively different for a test case.
- **Keep profiling your code as you optimize**
 - Concentrate your efforts on the thing that slows your code down.
 - This will change as you optimise.
 - So keep on profiling.

Wrap-up of day 1



Unix for Physicists

Contents



- Shells and commands on CSC supercomputers
 - bash (recommended)
 - tcsh
- NX
- Dealing with files and directories
- Programs
- Useful tools
- Use cases

What is shell?

- A *shell* is a program which provides the traditional, text-only user interface for Linux (and other Unix like systems)
- *Shell's* primary function is to read *commands* that are typed into a console or terminal window and then *execute* them.

What is shell cont., bash on Taito

- Text shell: Terminal with a set of commands
- Different flavors
 - **bash** (default)
 - **tcsh** (old default)
 - **zsh**,
 - **corn-shell**, ...



```

@taito-login3:~
File Edit View Search Terminal Help
[████████@ruskohaikara ~]$ ssh taito
Last login: Tue Feb  4 20:10:16 2014 from vpn16-153.csc.fi
Welcome
      CSC - Tieteen tietotekniikan keskus - IT Center for Science
      HP Cluster Platform SL230s Gen8 TAITO
-----
Contact
Helpdesk   : 09-457 2821, helpdesk@csc.fi   Switchboard : 09-457 2001
Usermanager : 09-457 2075, usermgr@csc.fi   Fax           : 09-457 2302
-----
User Guide
http://research.csc.fi/taito-user-guide
-----
Software
Available modules can be listed with command: module avail | module spider
-----
Partitions
parallel  : 16-448 cores / 5mins/3days      def/max runtime
serial    : 1-16 cores / 5mins/3days        def/max runtime
longrun   : 1-16 cores / 5mins/7days        def/max runtime
test      : 1-32 cores / 5mins/30mins       def/max runtime
-----
News
Tip: Use command cmake28 if you need CMake 2.8

[████████@taito-login3 ~]$ uname -a
Linux taito-login3.csc.fi 2.6.32-358.18.1.el6.x86_64 #1 SMP Fri Aug 2 17:04:38 EDT 2013 x86_64 x86_64 x86_64 GNU/Linux
[████████@taito-login3 ~]$
  
```


bash and tcsh comparison

	bash	tcsh	invoking	bash output	tcsh output
Shell variables	x=2	set x = 2	echo \$x	2	2
Env. variables	export z=3	setenv z 3	echo \$z	3	3
PATH	export PATH=/a:/b	set path=(/a /b)	echo \$path; echo \$PATH;	- /a:/b	/a /b /a:/b
Aliases	alias ls="ls -l"	alias ls "ls -l"	ls	<i>same as ls -l</i>	<i>same as ls -l</i>
Command prompt	PS1=abc-	set prompt=a bc-	[ENTER]	abc-	abc-
Redirection	prog > ofile 2> efile	(prog > ofile) >& efile	[ENTER]	<i>stdout -> ofile stderr -> efile</i>	<i>stdout -> ofile stderr -> efile</i>

Shell commands

- ➊ A *command* is an instruction given by a user telling a computer to do something, e.g.:
 - run a single *program*
 - run a group of *linked programs*
- ➋ Commands are generally issued by typing them in at the command line and then pressing the ENTER key, which passes them to the *shell*

Commands cont.

➤ Structure of a command:

`command -option [optional input]`

➤ Examples

- `apropos list`
- `ls -l`
- `clear`
- `finger username (Taito)`
 - `finger -m username (Sisu)`

ls

- Prints names of files in current directory
- Prints contents of a directory, if given as *ls directory*
- Only print filenames matching a wildcard expression
 - *ls *.txt*
- Option *-l* gives more info
- May find useful on Taito and Sisu
 - *ls -lrt* (reverse time ordered)
 - *ls -d /* --color=tty* (list directories, colorize the output)

mkdir [directory]

- Make a new directory
- `-p` to not complain about already existing directory and to make missing parent directories as needed

cd [directory]

- Change the current working directory
- `cd ..` to go up a directory

`mv [source] [dest]`

- Moves files or directories
- Can also rename files

`rm [file]`

- Removes files (*be careful!*)
- `-r` to remove a directory recursively
- `-f` to force removal (*be supercareful!*)
- Sometimes, e.g., on Taito, alias: `rm = 'rm -i'`

find [directory] [options]

- Finds files in a directory and its subdirectories that match the criteria given with the options
- Common use case, find files with certain names in the current directory:

```
find . -name '*.c' -print
```


grep -e 'searchterm' [files]

- Search for matching lines inside files
- *-i* for case insensitive
- *-n* to print line numbers

`pwd`

- Print the current working directory

`cat [file]`

- Prints contents of file to screen
- `cat -n` to precede lines with line numbers

less [file]

- Opens a scrollable view of a file
- q to quit
- / to search forward, ? to search backwards
- n to find the next match, N for previous
- Some people prefer **more [file]**, it allows to scroll down, but not up

`man [command]`

- Show the manual of command in less

`cp [source] [destination]`

- Copy a file
- `-r` to copy recursively a directory and its contents
- `-v` for verbose

scp [source] [dest]

- Like **cp**, but used for remote transfer
- For example: scp my_file
user@taito.csc.fi:'/absolute/path/to/dir'

rsync [source] [dest]

- Fast, versatile tool, remote and local usage
- E.g.: rsync my_file taito.csc.fi:

tar [commands] [file]

- Versatile tool used most in two ways
 - tar xvf some_file.tar
 - Extracts from file some_file.tar the contents of the archive **verbosely**
 - tar cvf my_files.tar my_dir/
 - **C**reates **verbosely** a new archive in file my_files.tar from the directory my_dir/
 - tar cvzf my_files.tar.gz my_dir/
 - Apply **gzip** (i.e., compress the tar archive)

wget *URL*

- Used to download files from the internet without a graphical browser such as Firefox or Chrome
- For example: `wget`
`http://ftp.gnu.org/gnu/hello/hello-2.7.tar.gz`
to download the gnu program hello

Selected Taito aliases

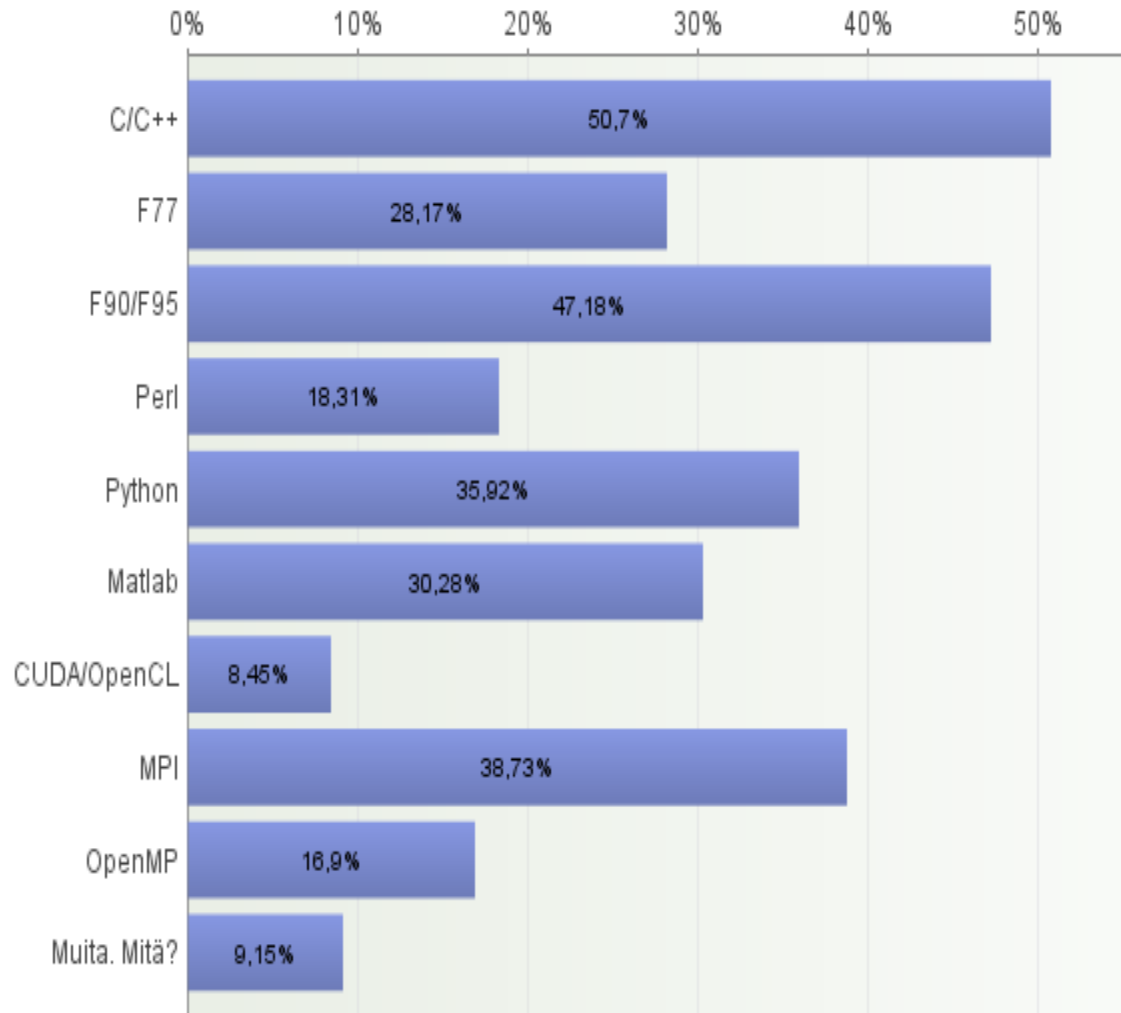
- Type *alias* to get the full list
 - alias chsh='/usr/alt/uadm2/bin/chsh'
 - alias mv='mv -i'
 - alias passwd='/usr/alt/uadm2/bin/passwd'
 - alias quota='/etc/profile.d/csc/csc-quota.bash'
 - alias sj='scontrol show job'
 - alias sn='scontrol show node'
 - alias vi='vim'

What is a program?



- A *program* is a sequence of instructions understandable by a computer's central processing unit (CPU) that indicates which operations the computer should perform
 - Ready-to-run programs are stored as *executable* files
 - An executable file is a file that has been converted from source code into machine code, by a specialized program called a compiler

Programming languages at supercomputers



gcc [source files] [-o prog]



- Compiles C source files into a program
- -o to give the name of the program, defaults to a.out
- -c to compile into .o -files

Compiling and installing programs



- For most programs, the three commands to compile and install in directory `/home/user/programs` are:
`./configure --prefix=/home/user/programs`
`make`
`make install`
- *make* will be discussed in detail later today
- Common destination: `$USERAPPL`

More useful tools

- `head`
- `tail`
- `wc`
- `which`
- `time`
- `ps`
- `top`
- `touch`
- `sed`
- `sort`
- `uniq`
- `cut`
- `paste`
- `awk`

Use case: set command prompt on Taito



1) Edit your profile file, e.g., with *vi* or *nano*

- *vi .profile*

add:

- *export*

```
PS1='\[\033[1;30m\]\u\[\033[0m\]@\[\033[1;34m\]\h\[\033[0m\]: [\w]# '
```

2) Apply changes

- *source .profile*

Using NoMachine Remote Desktop

Direct ssh connection – Unix/Linux

- From UNIX/Linux/OSX command line
- Use `-X` (or `-Y`) to enable remote graphics*

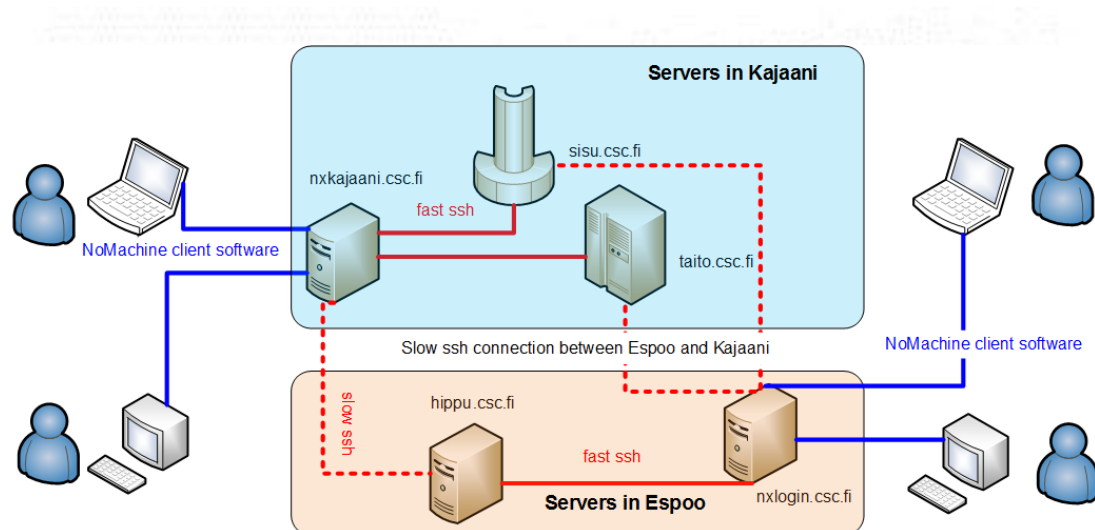
```
ssh -X yourid@taito.csc.fi
```

```
ssh -l yourid -Y taito.csc.fi
```

* In Windows you'd also need a windows emulator, but there is a better way

NoMachine Remote Desktop

- Client connection between user and gateway
- Good performance even with slow network
- **Ssh** from gateway to server (fast if local)
- Connect to right gateway
 - nxkajaani.csc.fi
 - nxlogin.csc.fi
- Persistent connection
- Suspendable
 - Continue later at another location
- Read the [instructions](#)...
 - ssh-key, keyboard layout, mac specific workarounds, ...
- Choose an application or server to use (right click)



Bunch of files use case:

You have received a bunch of files from a colleague, describing a glacier. They are in the directory "data". Your mission is to make sense of it all, in a possibly unfamiliar environment called bash.

Your team will consist of two veteran agents **Google** and **man**. As usual, ask for help from the friendly locals (person next to you and instructors) to succeed in the mission.

Part 1. The files in general

- How many files is there?
- How big are they in kilobytes?
- Are they small enough to be emailed?
- How about copying over Internet by some other means?
- What means there is to move or share files?
- How many lines and words is there in them?

Part 1. The files in general

Simple commands do the job:

```
$ cd data
```

```
$ ls
```

```
bed.txt README stake_positions.txt surface.txt thick.txt
```

```
$ ls -lh
```

```
total 812K
```

```
-rw-r--r-- 1 svali-user svali-user 271K Apr 26 13:11 bed.txt
```

```
-rw-r--r-- 1 svali-user svali-user 364 Apr 26 13:11 README
```

```
-rw-r--r-- 1 svali-user svali-user 150 Apr 26 13:12 stake_positions.txt
```

```
-rw-r--r-- 1 svali-user svali-user 267K Apr 26 13:11 surface.txt
```

```
-rw-r--r-- 1 svali-user svali-user 263K Apr 26 13:11 thick.txt
```

```
$ wc -lw *
```

```
231 46431 bed.txt
```

```
12 63 README
```

```
10 20 stake_positions.txt
```

```
231 46431 surface.txt
```

```
231 46431 thick.txt
```

```
715 139376 total
```


Part 1. The files in general

Also try:

```
$ file *
```

```
bed.txt:          ASCII text, with very long lines
```

```
README:          ASCII English text
```

```
stake_positions.txt: ASCII text
```

```
surface.txt:      ASCII text, with very long lines
```

```
thick.txt:        ASCII text, with very long lines
```

```
$ less README
```

```
$ less bed.txt
```


Part 2. Lot's of numbers?

Let's filter the numbers in the files so that you can calculate meaningful characteristics of the data. For example:

- What are the minimum and maximum values in bed.txt?
 - Notice the null values -99.0
 - Data does not need to be kept in matrix format to calculate minimum and maximum
 - use the pipes, cat, tr, grep, sort (because data is not that big)

Part 2. Lot's of numbers?

Let' pipe the data to tr, which can replace characters with other characters (like space “ ” with the end of line “\n”)

```
$ cat bed.txt | tr ' ' '\n' | less
```

Now there is one value per line (and if you look closely, we accidentally inserted some blank lines, too).

Let's remove null values and blank lines with grep and regular expressions.

```
$ cat bed.txt | tr ' ' '\n' | grep -v -e '-99.0' -e '^$' | less
```

Then sort the numbers and put them into a file.

```
$ mkdir -p ~/tmp
```

```
$ cat bed.txt | tr ' ' '\n' | grep -v -e '-99.0' -e '^$' | sort -n >  
~/tmp/bed.values
```


Part 2. Lot's of numbers?

Now that the numbers are ordered, it is pretty easy to see the maximum and the minimum

```
$ head -1 ~/tmp/bed.values
```

```
44.1
```

```
$ head -1 ~/tmp/bed.values
```

```
644.2
```

Let's next sum the values in file thick.txt. That would be very close to actually calculating the volume of the glacier, right?

Let's first extract the values, one value per line, the same thing as we did to bed.txt.

```
$ cat thick.txt | tr ' ' '\n' | grep -v -e '-99.0' -e '^$' | sort -n >
```

```
~/tmp/thick.values
```


Part 2. Lot's of numbers?

Now, enter awk!

```
$ awk '{sum+=$1}END{print sum}' ~/tmp/thick.values
```

```
983583
```

What happened? Uh, a lot. Awk is a programming language, that

- reads files, line by line

- for each line matching the condition, it applies commands in the curly braces {} after the condition. If there is no condition before the {}, those commands are applied to all lines. BEGIN and END are special conditions. The commands in them are executed before and after any/all lines are read, respectively.

- The fields in the lines can be referenced using \$0 (all fields), \$1 (the first field), \$2 (the second field), etc.

Part 2. Lot's of numbers?

What about the volume of the ice in the glacier? The grid size $dx=20m$, so the volume is

```
$ bc -l <<< "983583*20*20"  
393433200
```

Average thickness is also easy to calculate (NR is one of the special variables in awk, it tells the number of lines read so far).

```
$ awk '{sum+=$1}END{print sum/NR}' ~/tmp/thick.values  
75.4224
```

And so is the median (because we sorted the values).

```
$ wc -l ~/tmp/thick.values  
13041 /home/svali-user/tmp/thick.values  
$ awk 'NR == int(13041/2){print $1}' ~/tmp/thick.values  
73.2
```


Part 3. Make your own command!

In order to visualize the data, we often need to filter it, or change the format. I have written two commands, `elop`, that can be used to manipulate matrix formatted data, and `txt2xyz`, that transforms data in matrix format to data in xyz-format understood by gnuplot.

Let's first have a small look at how shell scripts (the commands above are written as shell scripts), and interactive shell interpreter work. Very simplified:

1. shell reads files (or standard input) line by line as `awk`
2. shell replaces variables (`$variable`, etc.) by their value
3. shell interprets the first word of each “block” as a command (or alias or function), rest of the words are arguments to the command

Part 3. Make your own command!



For example, elop:

```
$ cat ../bin/elop
#!/bin/bash
```

```
function usage {
  echo 'Usage: cat M0.txt | ./elop EXPRESSION [M1] [M2] ...'
  echo
  echo 'Reads matrix M0 from stdin, modifies each element according to the'
  echo 'expression, and writes the resulting matrix to stdout. The matrix'
  echo 'M0 elements are referred using m[0] in the EXPRESSION. elop accepts'
  echo 'optional files containing matrixes M1, M2, ... which elements'
  echo 'can be referred in EXPRESSION using m[1] and m[2], etc.'
  echo
  echo ' Examples:'
  echo
  echo '1\) Multiply each element of the matrix by 10'
  echo '   cat M0.txt | ./matoper "m[0]*10"'
  echo
  echo '2\) Find out which elements in M0 are larger than corresponding'
  echo '   elements in matrix M1'
  echo '   cat M0.txt | ./matoper "m[0] > m[1]" M1.txt'
  echo
  echo '3\) Multiply the elements of matrix M0 and M1, and add the elements'
  echo '   in M2'
  echo '   cat M0.txt | ./matoper "m[0] * m[1] + m[2]" M1.txt M2.txt'
  echo
}
```

First line tells that this is a bash, so use bash to interpret the rest of it.

Second line starts a function definition. This function is defined only inside the script, and only prints out usage instructions.

Part 3. Make your own command!



```
case $# in
```

```
0)
```

```
  usage
```

```
  exit
```

```
;;
```

```
*)
```

```
  n=$#
```

```
  expression=$1
```

```
  shift
```

```
  paste - $@ | awk "
```

```
{
```

```
  for(i=1;i<=NF/$n;i++){
```

```
    for(j=0;j<$n;j++){
```

```
      m[j]=\$(i+NF/$n*j)
```

```
    }
```

```
    printf "%s ",$expression
```

```
  }
```

```
  printf "\n\
```

```
}"
```

```
;;
```

```
esac
```

`$#` expands to the number of the arguments for the script. If there is none, print usage and exit script.

By default, put the number of arguments to variable `n`, the first argument to variable `expression`, “move arguments one step to left” and paste standard input with the files given as arguments to `awk`.

Paste concatenates files “side by side”, whereas `cat` concatenates them one after the other.

Part 3. Make your own command!

We need to do two things to use the file ../bin/elop as a command

1.give the script execution permissions

2.add the location of the script to the search path of our command line interpreter

```
$ chmod u+x ../bin/elop
```

```
$ export PATH=${PATH}:${PWD}../bin
```


Part 3. Make your own command!

Let's check that “surface - bed = thick” really, with for example
\$ cat surface.txt | elop '(m[1]>0)*(m[0] - m[1] - m[2])' thick.txt
bed.txt | less

Now, the elop-script that I presented, is already far from simple. Try understanding it little by little. To see what “\$#” and “shift” in the script do, write a little test script:

```
$ emacs ../bin/mytest &  
$ cat ../bin/mytest  
#!/bin/bash  
echo "Number of arguments: $#"  
shift  
echo "Number of arguments after shift: $#"  
$ chmod u+x ../bin/mytest  
$ mytest arg1 arg2 arg3
```

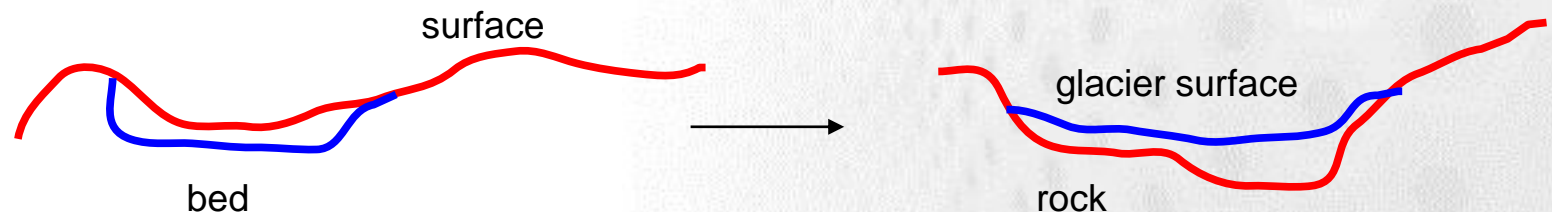

Part 4. Visualize it!

File `surface.txt` contains the surface, whether it be rock surface or glacier surface. File `bed.txt` gives the interface between the glacier and rock. It would make a better picture if we would have the rock surface, would it be under the glacier or not, and the surface of the glacier. That small `elop` command may be of help now ;)

```
$ cat surface.txt | elop '(m[1]>0)*m[1] + (m[1]<0)*m[0]' bed.txt >
~/tmp/rock.txt
```

```
$ cat surface.txt | elop '(m[1]>0)*m[0] + (m[1]<0)*(-99.0)' bed.txt >
~/tmp/glacier_surface.txt
```

Great!



Part 4. Visualize it!

Next, we pick a visualization program: gnuplot

Gnuplot can draw lines in 3D. The lines are given in xyz-format.

Xyz-format has three values on each line, x-, y-, and z-coordinates of points, and blank lines separating each line segment.

I wrote a small command to do the transform, `../bin/txt2xyz`, let's first have a look how it works and then what's inside the script.

```
$ cat ~/tmp/glacier_surface.txt | txt2xyz > ~/tmp/glacier_surface.xyz
```

```
$ cat ~/tmp/rock.txt | txt2xyz > ~/tmp/rock.xyz
```



```
#!/bin/bash
```

```
# Transform grid formatted data to xyz-formatted data
```

```
# Coordinates of the SW-corner
```

```
xmin=434000
```

```
ymin=8756400
```

```
# Grid spacing
```

```
dx=20
```

```
# 1. Put the matrix formatted data into pipe last line first with tac
```

```
# 2. With awk
```

```
# 2.1 Add a blank line in front of every line to separate lines drawn
```

```
# by gnuplot, i.e. draw lines in x-direction
```

```
# 2.2 loop over all elements in the row
```

```
# 2.3 for each element, print its x-, y-, and z-coordinates, if the
```

```
# matrix element value is positive, otherwise, print a blank line.
```

```
# This is to allow discontinuous lines in the x-direction.
```

```
# 3. remove excessive blank lines with cat -s
```

```
tac - | awk "
```

```
{  
  printf "\\n\\n"  
  for(i=1;i<=NF;i++){  
    if($i>0){  
      printf "%s %s %s\\n", $xmin+(i-1)*$dx, $ymin+(NR-1)*$dx, $i  
    } else {  
      printf "\\n\\n"  
    }  
  }  
}  
}" | cat -s
```



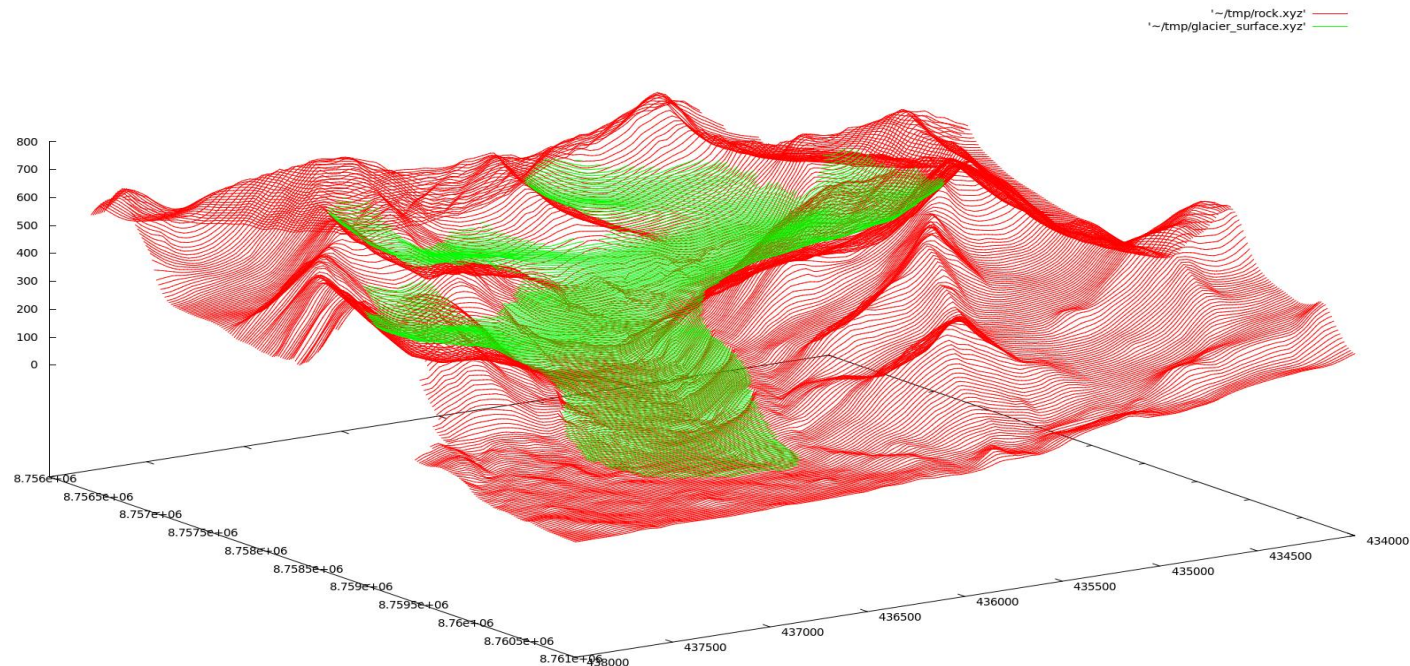
Part 4. Visualize it!

Part 4. Visualize it!

Then, let's draw the plot

```
$ gnuplot
```

```
> splot '~/tmp/rock.xyz' w l, '~/tmp/glacier_surface.xyz' w l
```



Part 5. Stakes

File `stake_positions.txt` has the x- and y-coordinates of the stakes. We'd like to plot them in the picture, too. But, there is no z-coordinate...

Let's take the z-coordinate from the surface data. The problem is that the stakes are not at the grid points, and we need to interpolate the z-coordinate value at stake positions. Proper interpolation starts to be a job that might be better done using more specialized software or compiled programming language. But, for the sake of exercise, let's have a look at a small script that I wrote to do a very simple interpolation :)

```
$ less zinterp
```


Part 5. Stakes

```
#!/bin/bash
```

```
dx=20
```

```
function usage {  
    echo 'usage: cat xy-data.txt | zinterp xyz-data.xyz'  
    echo  
    echo 'Interpolate z values from xyz-data.xyz to points in'  
    echo 'xy-data.txt using the value closest to xy-point'  
    echo 'in xyz-data.xyz.'  
    echo  
    echo 'File xy-data.txt contains two values (x- and y-coordinate)'  
    echo 'per line, and file xyz-data.xyz contains three values'  
    echo '(x-, y-, and z-coordinates). Output is in xyz-format.'  
    echo  
}
```


Part 5. Stakes



```
case $# in
1)
cat - $1 | awk -v dx=$dx '
BEGIN {
    n = 0
}
NF == 2 {
    n++
    x[n]=$1
    y[n]=$2
    z[n]=-99.0
    d[n]=dx^2+dx^2+1
}
NF == 3 {
    for(i=1;i<=n;i++) {
        l = (x[i]-$1)^2 + (y[i]-$2)^2
        if( l < d[i] ) {
            d[i] = l
            z[i] = $3
        }
    }
}
END {
    for(i=1;i<=n;i++) {
        print x[i],y[i],z[i]
    }
}'
;;
*)
usage
exit
;;
esac
```

- Notice how we pass a variable to awk
- 1.First collect all stake positions and initialize z-value to “null” and distance d to a value large, but not too large
 - 2.Read in the xyz-coordinate. Check if it is close to the any of the stake positions. If it is closer than any before it, use the z-value for stake position z-value.
 - 3.Print the result

Part 5. Stakes

Let's see how it works:

```
$ cat data/stake_positions.txt | zinterp ~/tmp/surface.xyz >  
~/tmp/stake_positions.xyz  
$ gnuplot  
> splot '~/tmp/rock.xyz' w l, '~/tmp/glacier_surface.xyz' w l,  
 '~/tmp/stake_positions.xyz' w p
```


Part 5. Stakes



And last, we would like to plot the thickness of the glacier along the path spanned by the stakes. To make it simple, the path will consist of straight line segments between the stakes, and the thickness is plotted at about constant interval ($d=20\text{m}$) along the path.

The script generates a list of points along the path, including the stake positions.

```
$ less bin/pathinterp
```

```
#!/bin/bash
```

```
d=20
```

```
function usage {
```

```
    echo 'usage: cat xy-data.txt | pathinterp'
```

```
    echo
```

```
    echo 'Interpolate evenly spaced xy-values between points.'
```

```
    echo
```

```
    echo 'File xy-data.txt contains two values (x- and y-coordinate)'
```

```
    echo 'per line'
```

```
    echo
```

```
}
```

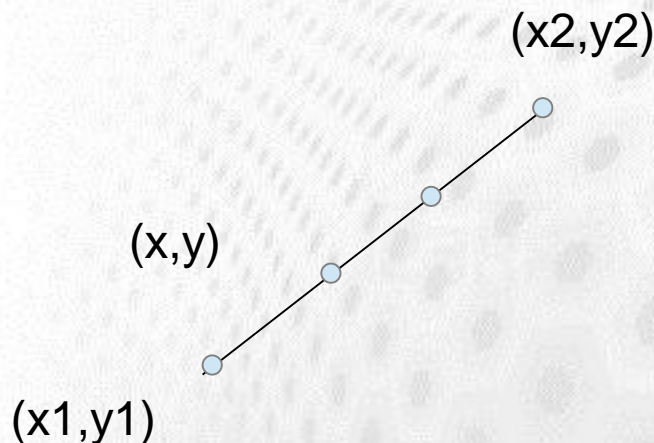

Part 5. Stakes

```

case $# in
0)
  cat - | awk -v d=$d '
  NR == 1{
    x1 = $1
    y1 = $2
    print x1,y1
  }
  NR != 1 {
    x2 = $1
    y2 = $2
    n = int(sqrt((x2-x1)^2+(y2-
y1)^2)/d)
    dx=(x2-x1)/n
    dy=(y2-y1)/n
    for (i=1;i<n;i++) {
      x = x1+i*dx
      y = y1+i*dy
      printf "%f %f\n",x,y
    }
    print x2, y2
    x1 = x2
    y1 = y2
  }'
;;
*)
  usage
  exit
;;
esac

```

Let's assume that the stake positions are in correct order



Part 5. Stakes

With the extended list of points, and thickness in xyz-format, we run zinterp, and plot the result.

```
$ txt2xyz < data/thick.txt > ~/tmp/thick.xyz
```

```
$ pathinterp < data/stake_positions.txt > ~/tmp/path.txt
```

```
$ cat ~/tmp/path.txt | zinterp ~/tmp/thick.xyz > ~/tmp/path_thick.xyz
```

```
$ gnuplot
```

```
> splot '~/tmp/path_thick.xyz' w impulses, '~/tmp/path.txt' using 1:2:(0),  
'data/stake_positions.txt' using 1:2:(0)
```


Congratulations!

- . That's all!
- . No, not really ;)
- . Continue by making small modifications to the example scripts. Especially the plots could be prettier.
- . Write your own command line filters and scripts. They do not need to be perfect in the beginning... or later :)
- . Always, visualize your data!

Computational physics using GPUs and Xeon Phi

Fredrik Robertsen

Åbo Akademi

froberts@abo.fi



Outline

- What is a **GPU** and what is a **Xeon Phi** ?
- Why and when should you use them ?
- How can they be programmed ?
- Demo
- Advanced usage

Your average CPU

- Fairly good at everything
- Excellent at single threaded work
 - Out of order execution
 - High clock rates, 3.5 GHZ and beyond
- Multicore with shared memory
 - Up to 18 cores
- Vector instructions
 - 256 bit registers with Advanced Vector Instructions
AVX

What is a GPU and a Xeon Phi

- Floating point computation accelerators
- Many cores
- Some do vector instructions
- Additional PCI-e cards that need to be added to a host
- They are only good for certain types of jobs
 - Massively data parallel
 - Not control bound
 - Vectorized or vectorizable code



What are they II

GPU

- Two vendors: NVIDIA and AMD
- Streaming architectures
- Large core count, 2000+
 - Organized into larger units
- Very little cache memory
 - ~40 byte per CUDA core on Nvidia

Xeon Phi

- Evolution of Intel's GPU architecture
- More like a “normal” CPU
 - X87 FPU
- Large L2 cache
- 512 bit vector instructions



Reasons to use computational accelerators

Reason #1

- **Tianhe-2 (MilkyWay-2)**
- #1 on Top500
- 33.9 Pflop/s
- 16000 computer nodes
 - two Intel Ivy Bridge Xeon processors
 - three Xeon Phi cards

Reason #2

- **Titan - Cray XK7**
- #2 on Top500
- 17.6 Pflop/s
- 18688 computer nodes
 - One 16-core AMD Opteron
 - One Nvidia Tesla K20X GPU

1 Pflop/s = 10^{15} flop/s



More realistic reasons ...

GPUs

- Cheap
 - 300-600 € for a normal graphics processor
 - 2-6000 € for a dedicated GPGPU
- Performance
 - 4-6 Tflop single precision
 - 1+ Tflop double precision
- Excellent way of adding performance to existing computers
- Massive internal memory bandwidth

Xeon Phi

- X86 compatible
 - Programs need to be recompiled
- “Easy” to program
- Performance
 - 2+ Tflop single precision
 - 1+ Tflop double precision
- Massive memory bandwidth
- Tricky to build a Xeon Phi system

Taito node:
0,324 Tflop DP
0,649 Tflop SP

GPUs

- **CUDA**
 - Nvidia only
 - Like C
- **OpenACC**
 - Nvidia only for now..
 - Directive based
- **OpenCL**
 - Like CUDA but cross platform

Xeon Phi

- **Native**
 - Run code only on the device
- **Offload**
 - Directive based
 - OpenMP 4.0 offload
- **OpenCL**
 - Existing OpenCL code will work

Programming II

- Main issues when programming
 - Data movements between host and card
 - “Any order execution” there are no guarantees that threads are executed in a certain order
 - Optimizing it for the accelerator
- Next a few examples

Basic example with OpenMP

- Basic code example parallelized with OpenMP
- Simple vector addition
- 100000000 elements
- Computation runtime on an Intel Ivy Bridge i7-3770k:
 - 0.0653523 sec with 8 threads
 - 0.124101 sec with 1 thread

```
1  int main()
2  {
3      int count = 100000000;
4      float *A = new float[count];
5      float *B = new float[count];
6      float *C = new float[count];
7      for (int i = 0; i < count; ++i)
8      {
9          A[i] = i;
10         B[i] = count - i;
11     }
12     #pragma omp parallel for
13     for (int i = 0; i < count; ++i)
14     {
15         C[i] = A[i] + B[i];
16     }
17 }
```



OpenACC

- Similar to the OpenMP version
- Replace #omp pragmas with equivalent OpenAcc versions
- Computational runtime on an Nvidia K20:
 - 0.00935793 sec
 - 0.452739 sec with data transfers
- PGI, CAPS, Cray compiler support

```
1  int main()
2  {
3      int count = 100000000;
4      float *A = new float[count];
5      float *B = new float[count];
6      float *C = new float[count];
7      for (int i = 0; i < count; ++i)
8      {
9          A[i] = i;
10         B[i] = count - i;
11     }
12     #pragma acc data copyin(A[0:count],B[0:count]) copyout(C[0:count])
13     {
14         #pragma acc parallel loop
15         for (int i = 0; i < count; ++i)
16         {
17             C[i] = A[i] + B[i];
18         }
19     }
20 }
21
```

Move data

Offload to GPU

Segment executed on the GPU

Simple CUDA example

- Computational runtime on an Nvidia K20:
 - 0.00845494 sec
 - 0.456792 sec with data transfers
- Code is compiled with nvcc compiler that is part of NVIDIA's software development toolkit

```

1  __global__ void kernel(float *A, float *B, float *C, int count)
2  {
3      int i = (blockIdx.x*blockDim.x)+threadIdx.x;
4      if (i >= count)
5          return;
6      C[i] = A[i] + B[i];
7  }
8
9  int main()
10 {
11     int count = 100000000;
12     float *A = new float[count];
13     float *B = new float[count];
14     float *C = new float[count];
15     float *A_dev, *B_dev, *C_dev;
16
17     cudaMalloc((void**)&A_dev, sizeof(float)*count);
18     cudaMalloc((void**)&B_dev, sizeof(float)*count);
19     cudaMalloc((void**)&C_dev, sizeof(float)*count);
20
21     for (int i = 0; i < count; ++i)
22     {
23         A[i] = i;
24         B[i] = count - i;
25     }
26     dim3 thread;
27     dim3 grid;
28     thread.x=256;
29     grid.x=1+(count/thread.x);
30
31     cudaMemcpy(A_dev, A, sizeof(float)*count, cudaMemcpyHostToDevice);
32     cudaMemcpy(B_dev, B, sizeof(float)*count, cudaMemcpyHostToDevice);
33
34     kernel<<<grid,thread>>>>(A_dev,B_dev,C_dev, count);
35
36     cudaMemcpy(C, C_dev, sizeof(float)*count, cudaMemcpyDeviceToHost);
37 }

```

Code executed on the GPU

Allocate memory on the GPU

Set threadblock and grid size

Transfer data to the device

Start kernel

Transfer data to the host

CUDA unified memory example

- Introduced in Cuda 6.0
- Simplifies data movement between host and device
- Will not speed up code and there are cases where it will give worse performance
- Compile with “nvcc -gencode arch=compute_35,code=sm_35”
- Only works on newer cards

```

1  __global__ void kernel(float *A, float *B, float *C, int count)
2  {
3      int i = (blockIdx.x*blockDim.x)+threadIdx.x;
4      if (i >= count)
5          return;
6      C[i] = A[i] + B[i];
7  }
8
9  int main()
10 {
11     int count = 100000000;
12     float *A;
13     float *B;
14     float *C;
15
16     cudaMallocManaged((void**)&A, sizeof(float)*count);
17     cudaMallocManaged((void**)&B, sizeof(float)*count);
18     cudaMallocManaged((void**)&C, sizeof(float)*count);
19
20     for (int i = 0; i < count; ++i)
21     {
22         A[i] = i;
23         B[i] = count - i;
24     }
25     dim3 thread;
26     dim3 grid;
27     thread.x=256;
28     grid.x=1+(count/thread.x);
29
30     kernel<<<grid,thread>>>(A,B,C, count);
31     cudaDeviceSynchronize();
32 }

```

Code
executed on
the GPU

Allocate memory
for the host and the
GPU

Use A, B pointers in
host code

Set threadblock and grid size

Start kernel, with A, B, C
pointers

Sync device before data
is used on the host

Xeon Phi offload

- Computational runtime on an Intel Xeon Phi 5110P
 - 0.118713 sec
- Room for further optimizations
- Easy to get running, hard to get running well
- Compiled normally with the Intel compiler

```
1  int main()
2  {
3      int count = 100000000;
4      float *A = new float[count];
5      float *B = new float[count];
6      float *C = new float[count];
7
8      for (int i = 0; i < count; ++i)
9      {
10         A[i] = i;
11         B[i] = count - i;
12     }
13     #pragma offload target(mic) in(A,B:length(count)) inout(C:length(count))
14     {
15         #pragma omp parallel for
16         #pragma ivdep
17         for (int i = 0; i < count; ++i)
18         {
19             C[i] = A[i] + B[i];
20         }
21     }
22 }
```

Move data & offload

Ingnore dependencies

Segment executed on the Phi

Xeon Phi native

- Xeon Phi's "party piece"
- Compile with Intel's compiler and `-mmic` flag
- Log in to the card with `ssh`
- Run on the Xeon Phi card!

Demo

- Port a simple N-body solver
- True $O(n^2)$ complexity
- Not really tuned for any specific architecture
- Should however give a decent performance
- Parallelized per particle
 - Calculate all the forces on a particle
 - One particle per iteration of a the parallel loop
 - Update the position and velocity of each particle



Xeon Phi native

- Ssh into taito.csc.fi then ssh to m1
- Do a “module purge”
- Then “module load intel/15”
- Code is in: NbodyNative
- Have a look in the makefile
 - Set architecture to be mic
- Recompile and run
 - “make”
 - “./solver” or “srun -n 1 -p mic -gres=mic:1 ./solver”
- (non CSC systems: ssh into mic0 and then run)
- No libpng, would need to be recompiled for the card

Xeon Phi offload

- Ssh into taito.csc.fi then ssh to m1
- Do a “module purge”
- Then “module load intel/15”
- Code is in: NbodyOffload
- All modifications are done to Solver.cpp
- Add a `#pragma offload target (mic) in(var1, var2...:length(var lenght) inout(..)` to the block that should be run on the accelerator
- Add `__attribute__((target(mic)))` before the declaration of any function that will be called from accelerator code
- Recompile and run
 - “make”
 - “./solver” or “srun -n 1 -p mic -gres=mic:1 ./solver”



Cuda

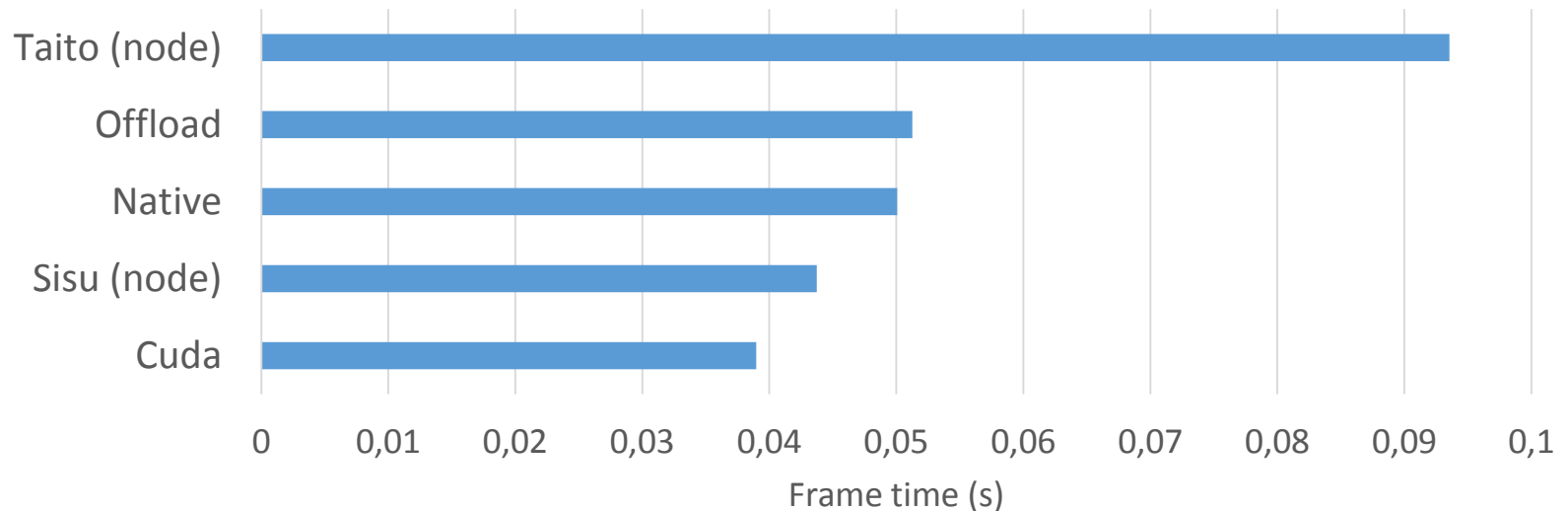
- Ssh into taito-gpu.csc.fi
- All modifications happen in Solver.cu located in NbodyCuda
- Allocate memory for dev_ pointers, copy the host memory to the corresponding pointers
- Add `__device__` to declarations of functions to be run on the accelerator
- Add `__global__` to your kernel functions, the ones you call from host code to be executed on the device
- Add calculations for getting the current thread id to the kernel functions
- Add launch parameters
- Compile and run
 - “make”
 - “srun -n 1 -p gpu -gres=gpu:1 ./solver”



Performance

Subject to change

- Taito node: 0.09354179 sec / frame
- Xeon phi offload: 0.051267 sec / frame
- Xeon phi native: 0.050066 sec / frame
- Sisu node: 0.043749 sec / frame
- Cuda: 0.038971 sec / frame



Optimization

- Xeon phi general
 - Vectorization
 - Cache behavior
 - Prefetching
- Xeon phi native
 - Memory alignment
- Cuda
 - More parallelism
 - In this case manually caching particles for reuse by other threads

Theoretical vs actual performance

- Theoretical floating point performance is measured using FMA (fused multiply add)
 - If your code cannot be structured for FMA performance can never get to more than 50% of theoretical
- Memory bandwidth
 - GPU
 - Able to get to 85++% of theoretical peak
 - Phi
 - Good if you get 50%



Multi node accelerated code

- Separate cards in the system
- Communication needs to go through the host*
 - You need to move the data back
 - Xeon phi native you can just call MPI functions

*currently changing and there are systems where the accelerator and NIC can communicate without the host

Cuda and MPI

- Normal workflow:
 - Move data from host to device, pass device buffer to MPI
- GPU aware MPI
 - Give MPI the pointer to the data on the device, it will take care of the transfers
- GPU Direct
 - Give MPI the device pointer and the data will be moved from the device to the network without going through the host

Summary

- Floating point accelerators
 - Cheap performance
- Surprisingly easy to program
 - Basically C
- Great performance



Thank you

Questions?

froberts@abo.fi





Massively parallel computations

Visualization at CSC

visualization@csc.fi

Jyrki Hokkanen



CSC - Visualization

https://research.csc.fi/visualization

CSC Front Page Suomeksi

Services for Research

Home Sciences Computing **Software** News Support Scientific Customer Panel

Services for Research → Software → Visualization

Software

- Software Packages +
- Programming
- Parallel Computing
- Code Optimization +
- Visualization -**
 - Image examples
- Open Source Software Development at CSC

Visualization

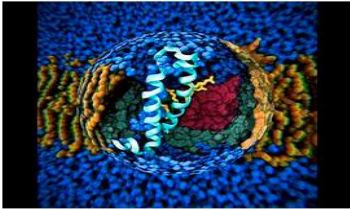
Visualization of large data sets is possible with software utilizing parallel processing and both shared and distributed memory available on CSC's computing servers. Visualization specialists at CSC can help in fine tuning the images.

Support


Visualization support is included in Grand Challenge projects. Resources permitting support is available also for other projects.

Besides data, also scientific principles and ideas can be illustrated at CSC using general purpose 3D graphics programs. Questions about CSC's visualization services should be posted to visualization@csc.fi.


Examples




PLoS Computation Biology 2/2009 cover picture. Conformational Changes and Slow Dynamics through Microsecond Polarized Atomistic Molecular Simulation of an Integral Kv1.2 Ion Channel. Bjelkmar, Niemelä, Vattulainen, Lindahl / Stockholm University, VTT, TKK, Stockholm University.

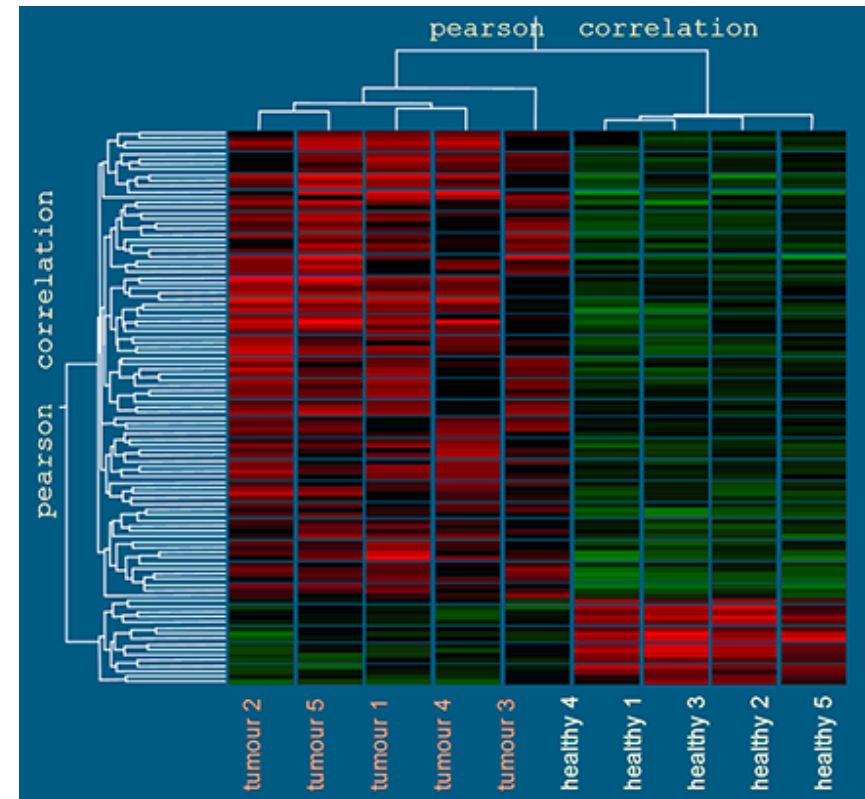
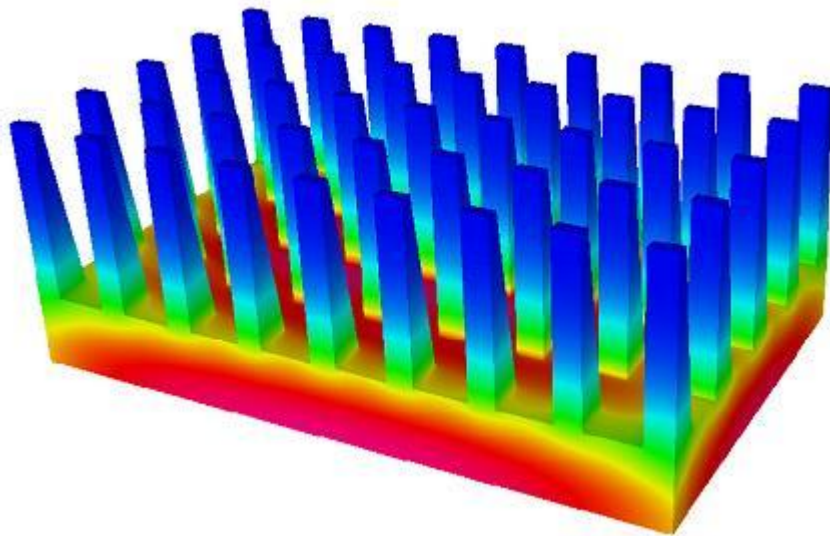


Gold-atom clusters colliding with the surface of a gold substrate and forming craters. Grand Challenge project, Samela, Nordlund / University of Helsinki. Honorary mention in the scientific image competition of the Tiede magazine Oct. 5, 2010.





Scientific visualization - Information visualization

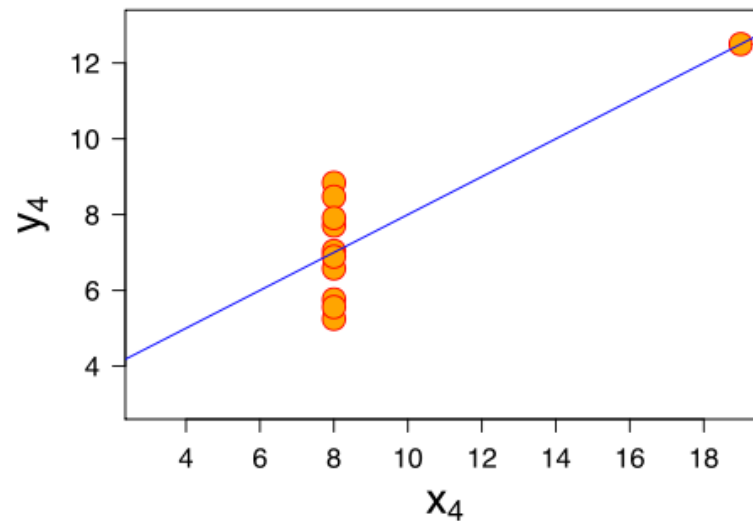
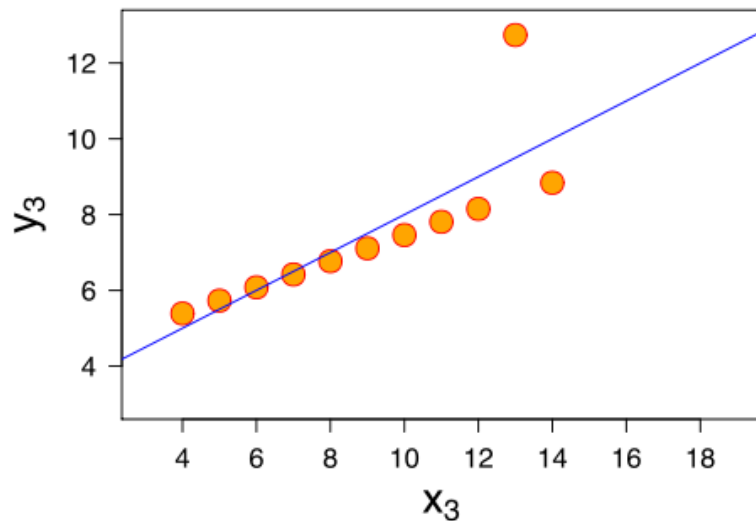
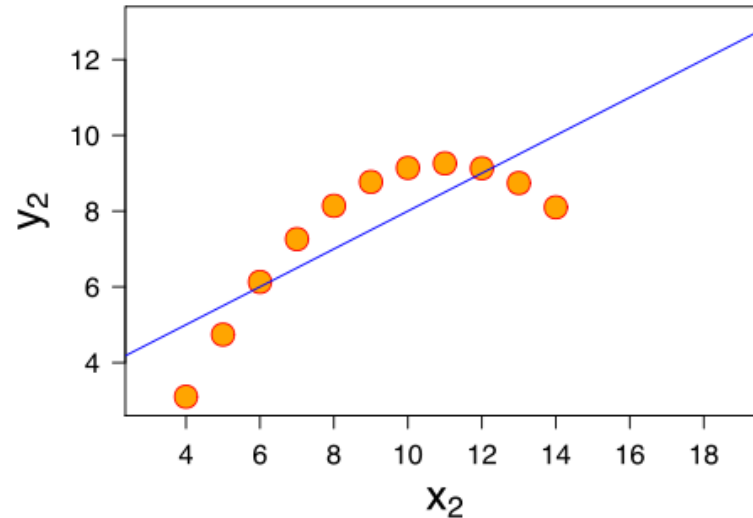
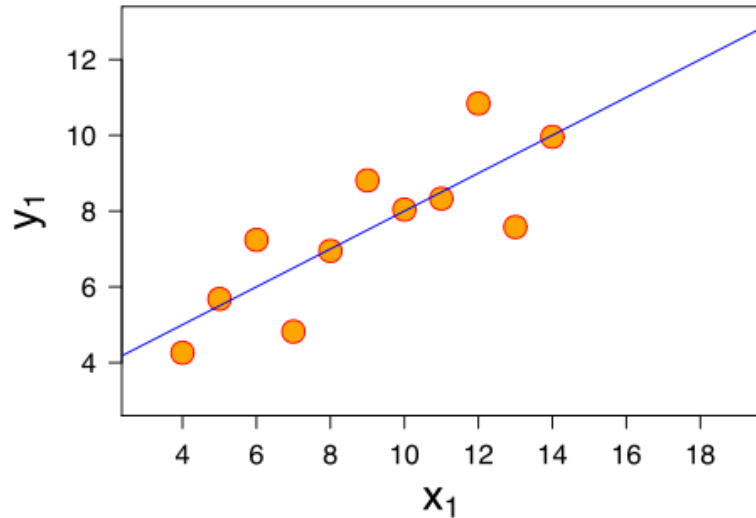


Anscombe's quartet example

Data sets **I**, **II**, **III**, **IV** have identical simple statistical properties (mean, variance, correlation, linear regression)

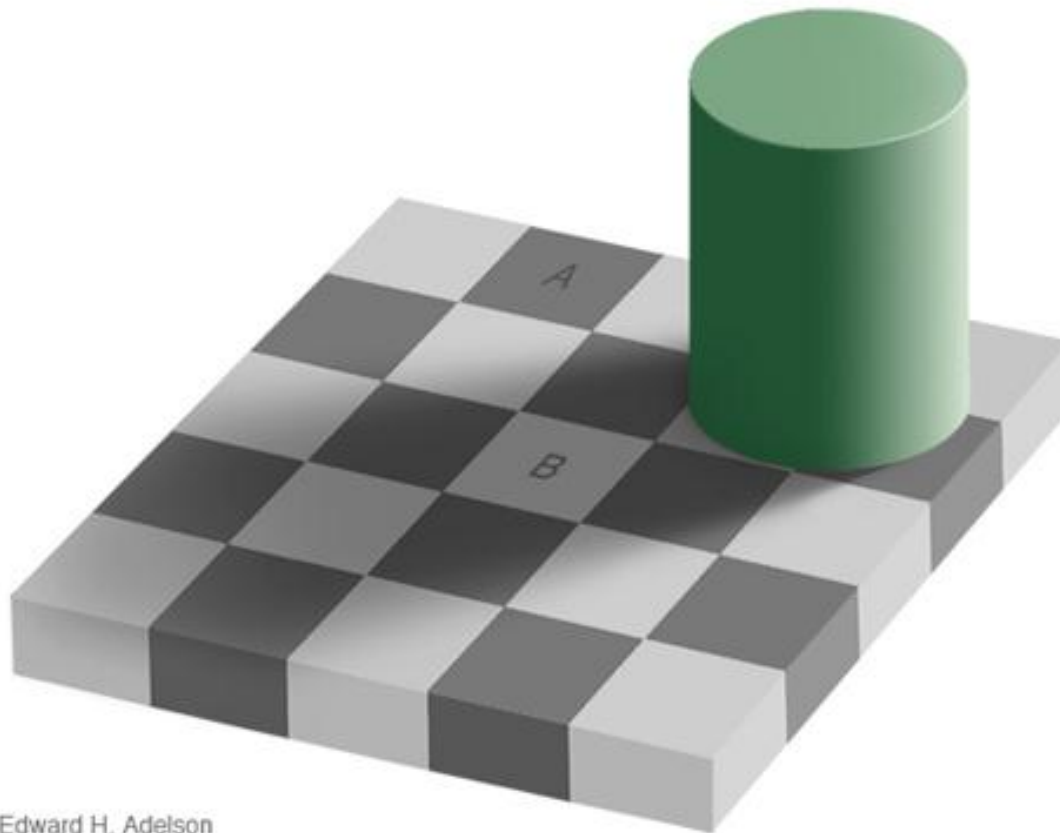
I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Always take a look at the data!



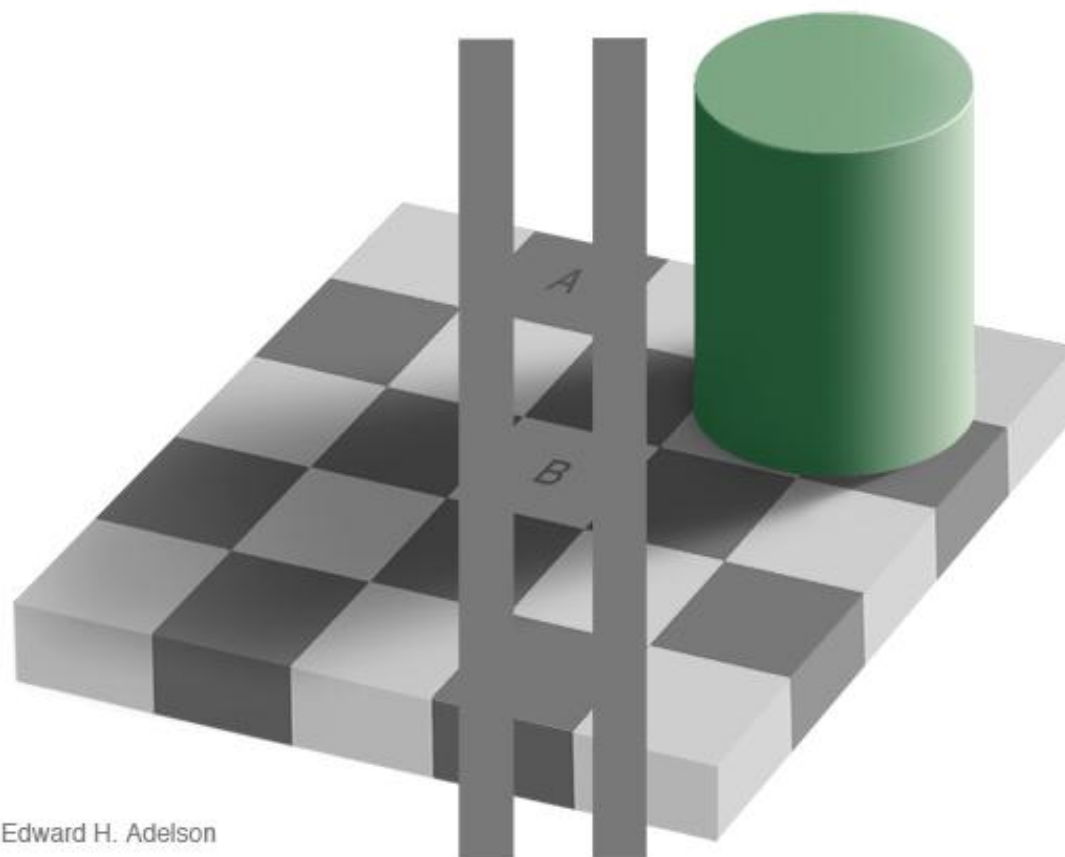
http://en.wikipedia.org/wiki/Anscombe%27s_quartet

Be aware of "information processing"



Edward H. Adelson

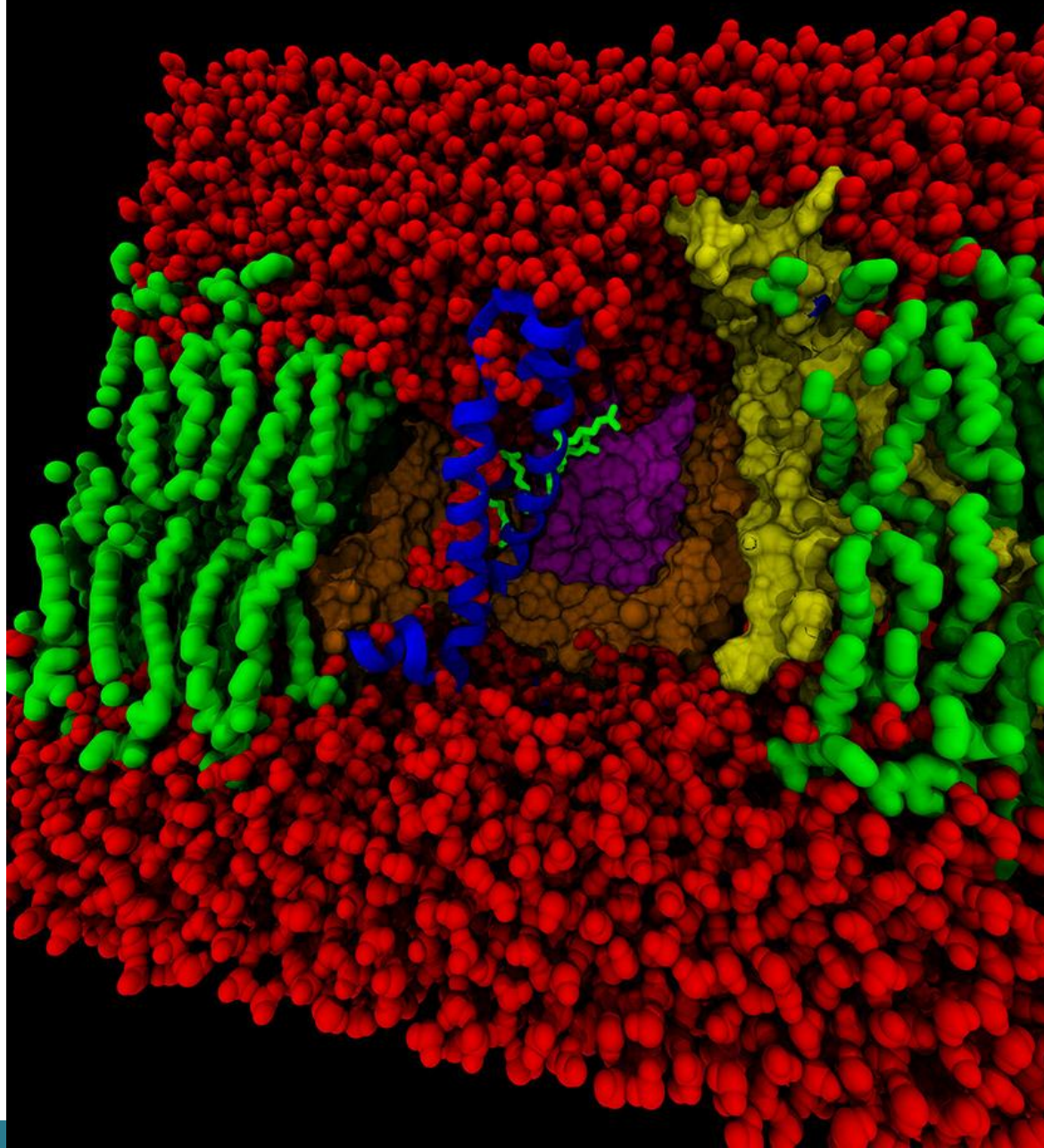
The squares marked A and B are the same shade of gray.

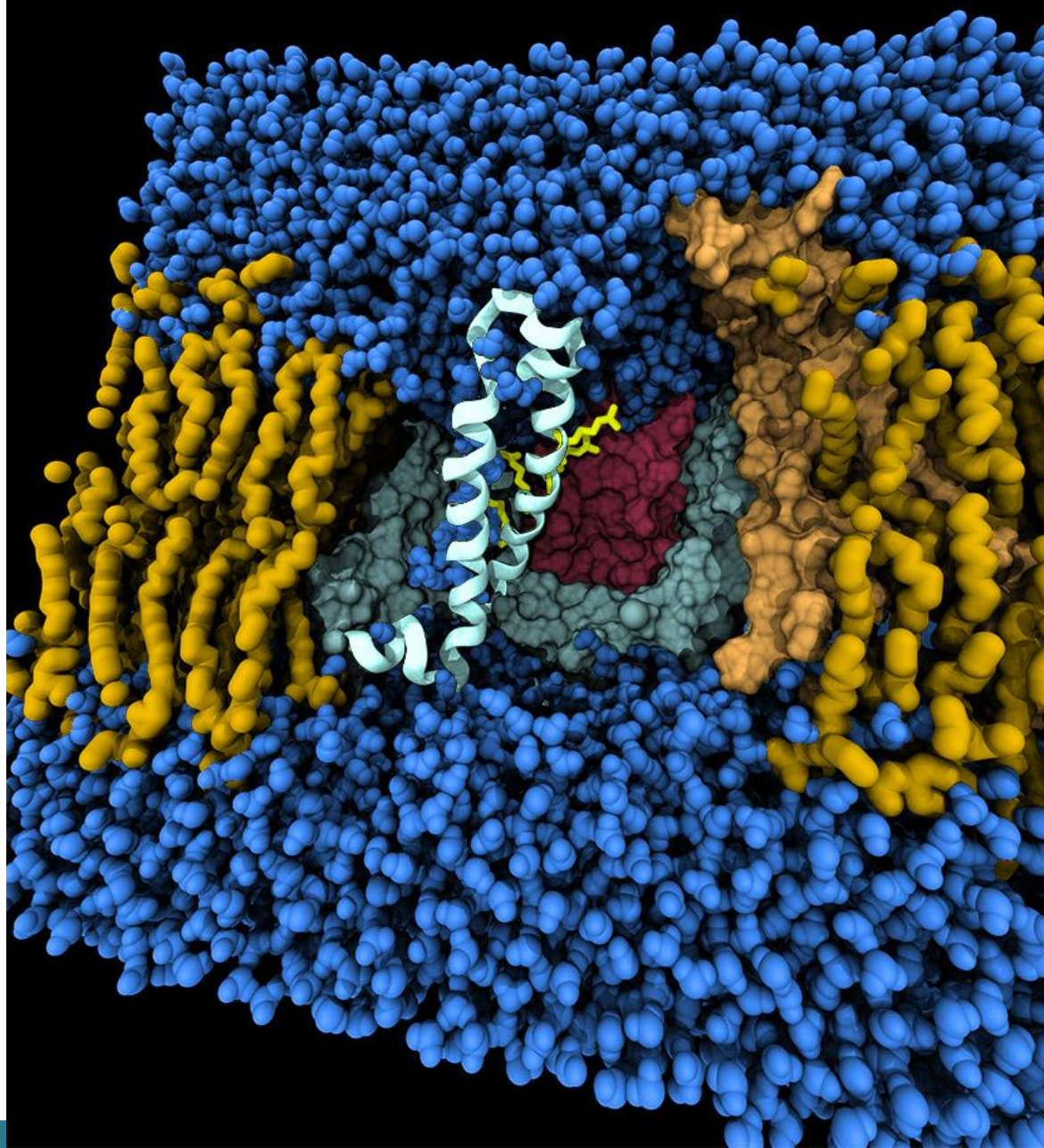


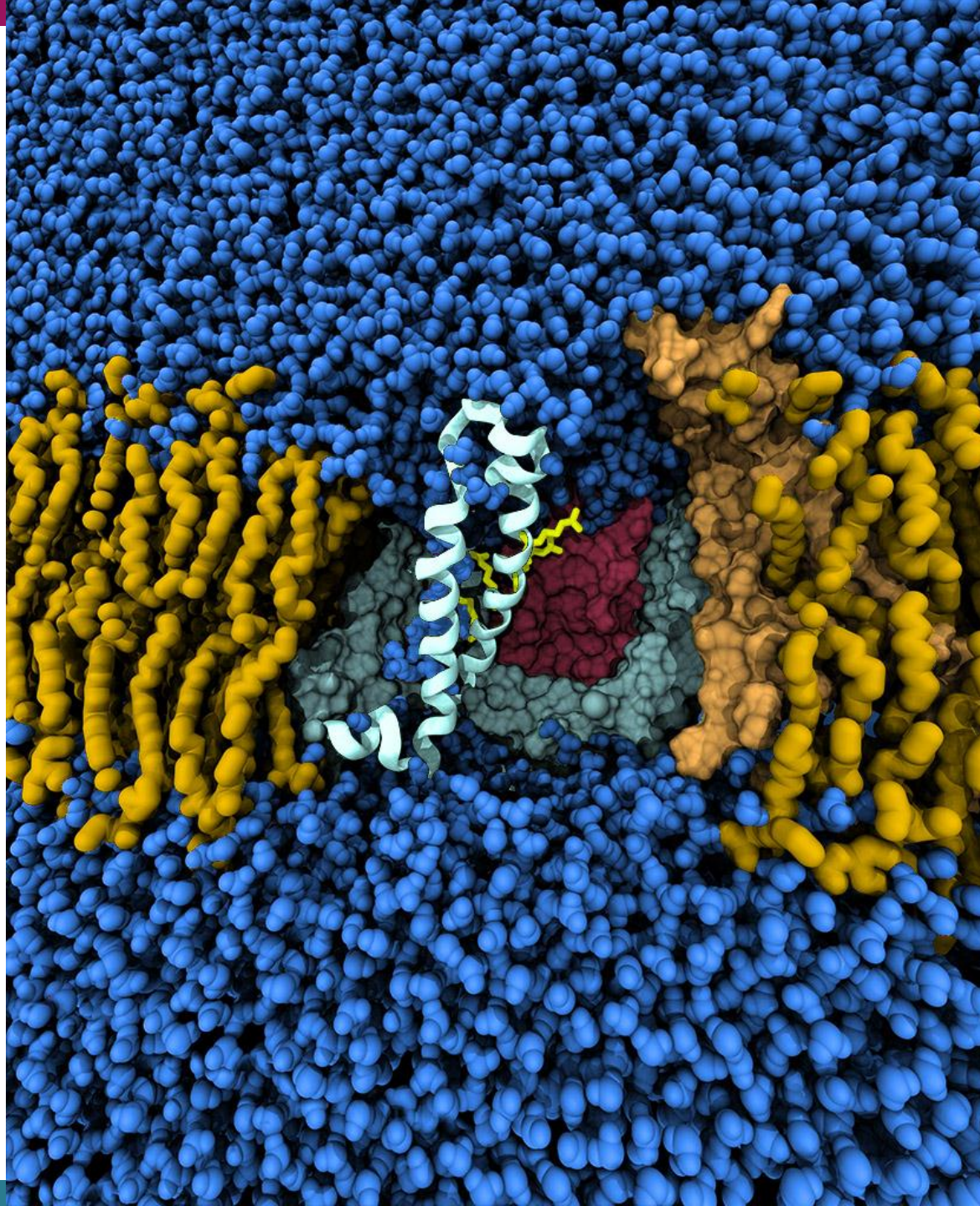
Edward H. Adelson

The background of the slide is an abstract scientific visualization. It features a complex, glowing blue structure that resembles a molecular model or a biological pathway. The structure has a central, elongated, and somewhat twisted form, with various smaller, circular and ring-like components attached to it. The overall color palette is dominated by deep blues and purples, with some lighter, almost white, highlights that give the structure a three-dimensional, ethereal appearance. The lighting is dramatic, with strong highlights and deep shadows, creating a sense of depth and complexity.

Scientific visualization examples, techniques, tips

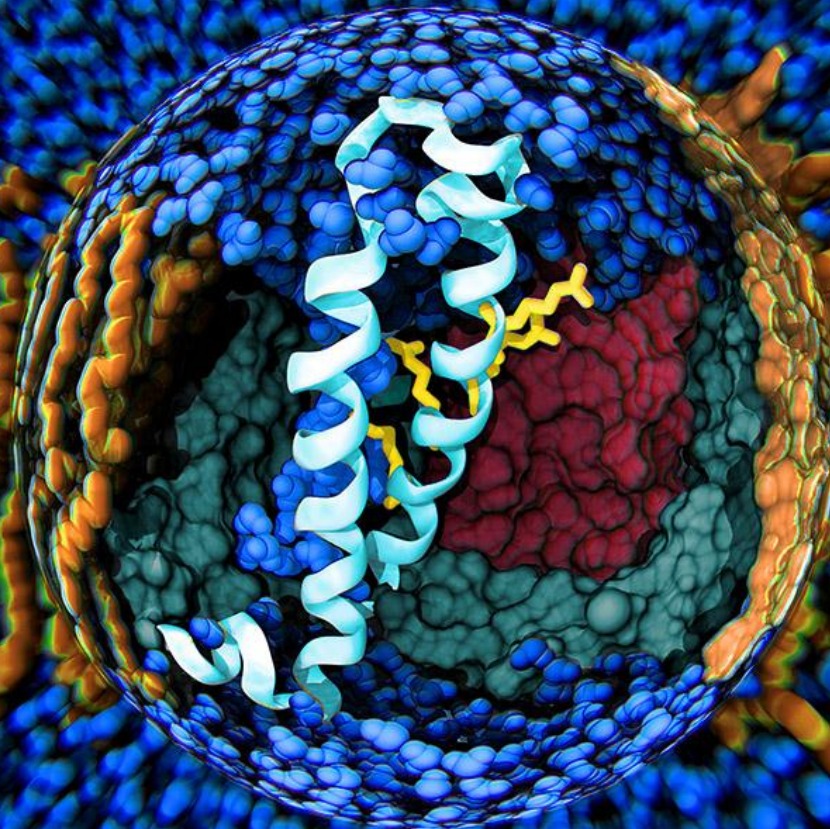






PLoS Computation Biology

February 2009 Cover



CSC Grand Challenge project:
Voltage-gated ion channels
Vattulainen, Lindahl

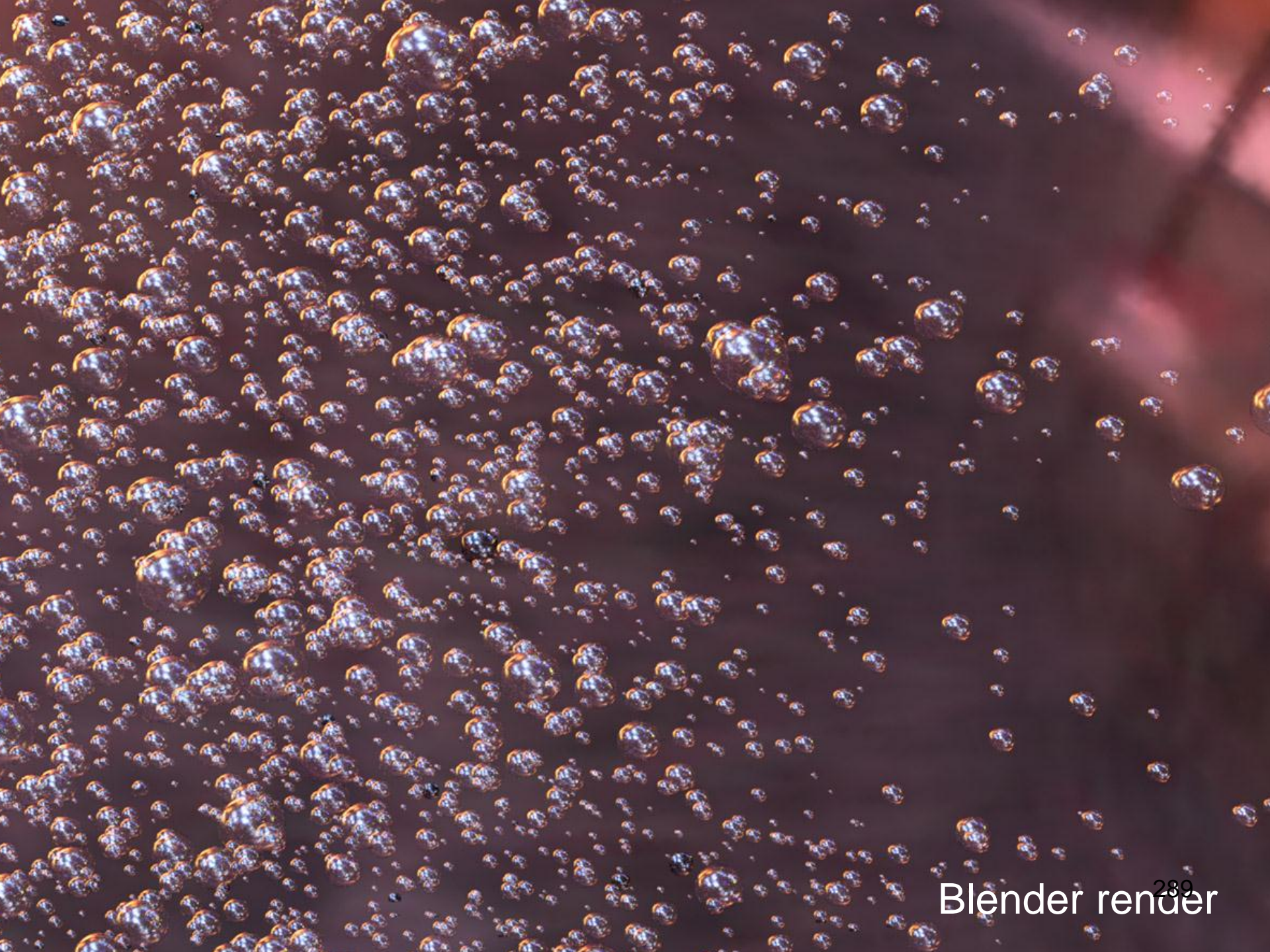
CSC Jyrki Hokkanen

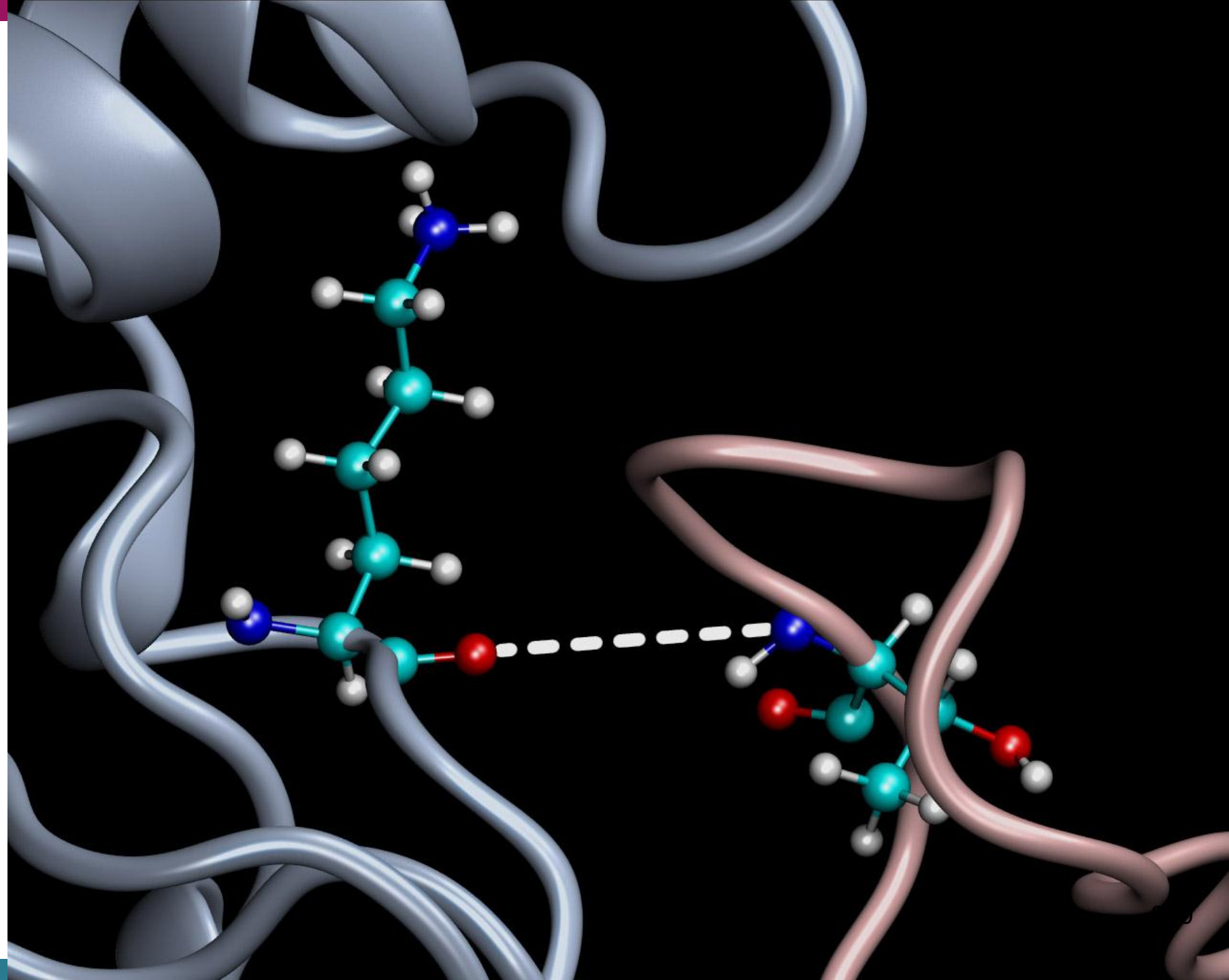
External rendering for extra impact

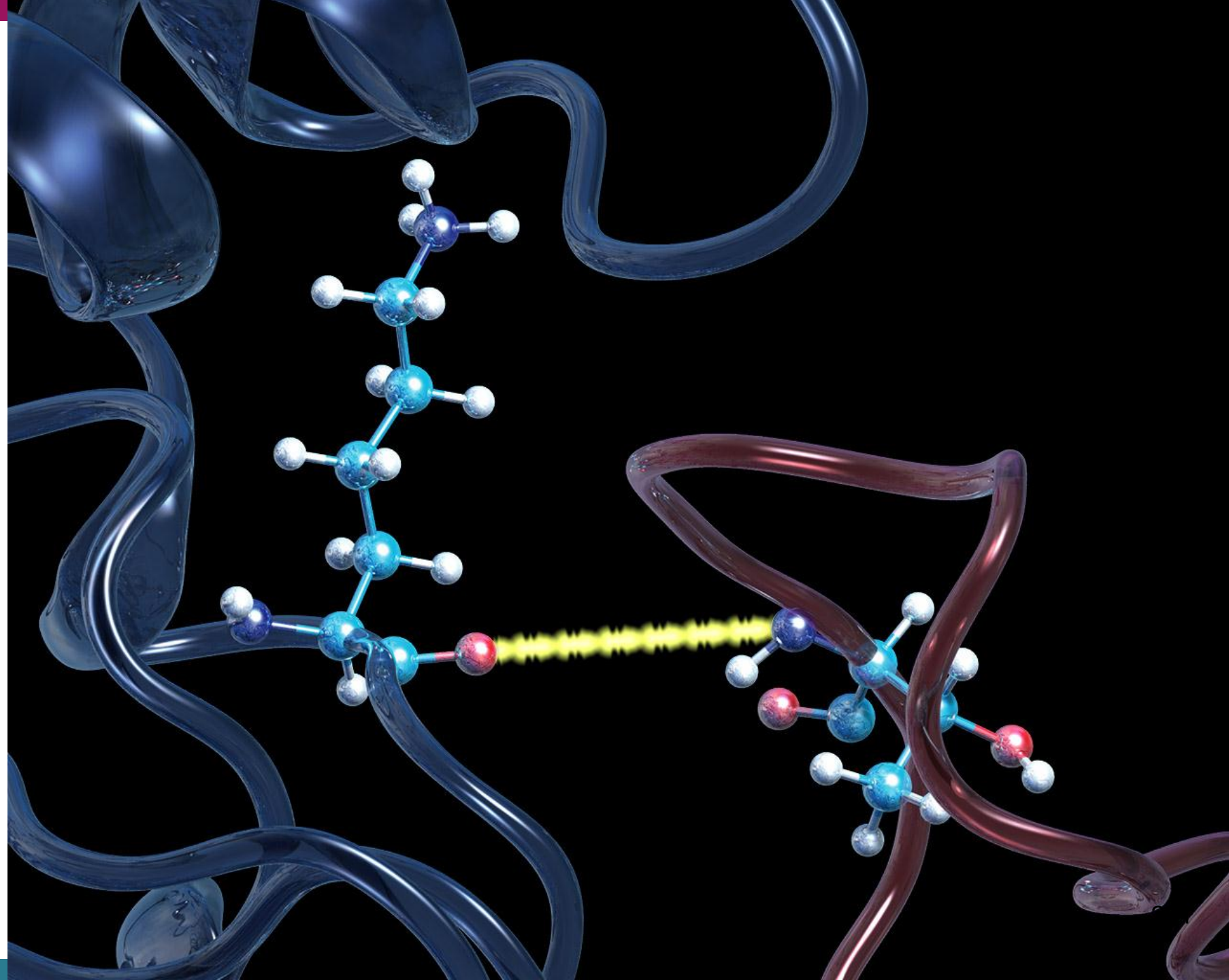
Export from scientific software, import to 3D-graphics application

- most examples here rendered with free app **Blender** (blender.org)
- is excessively versatile, has two internal and several plug-in renderers
- POV-Ray** is another widely used free stand alone renderer (povray.org)

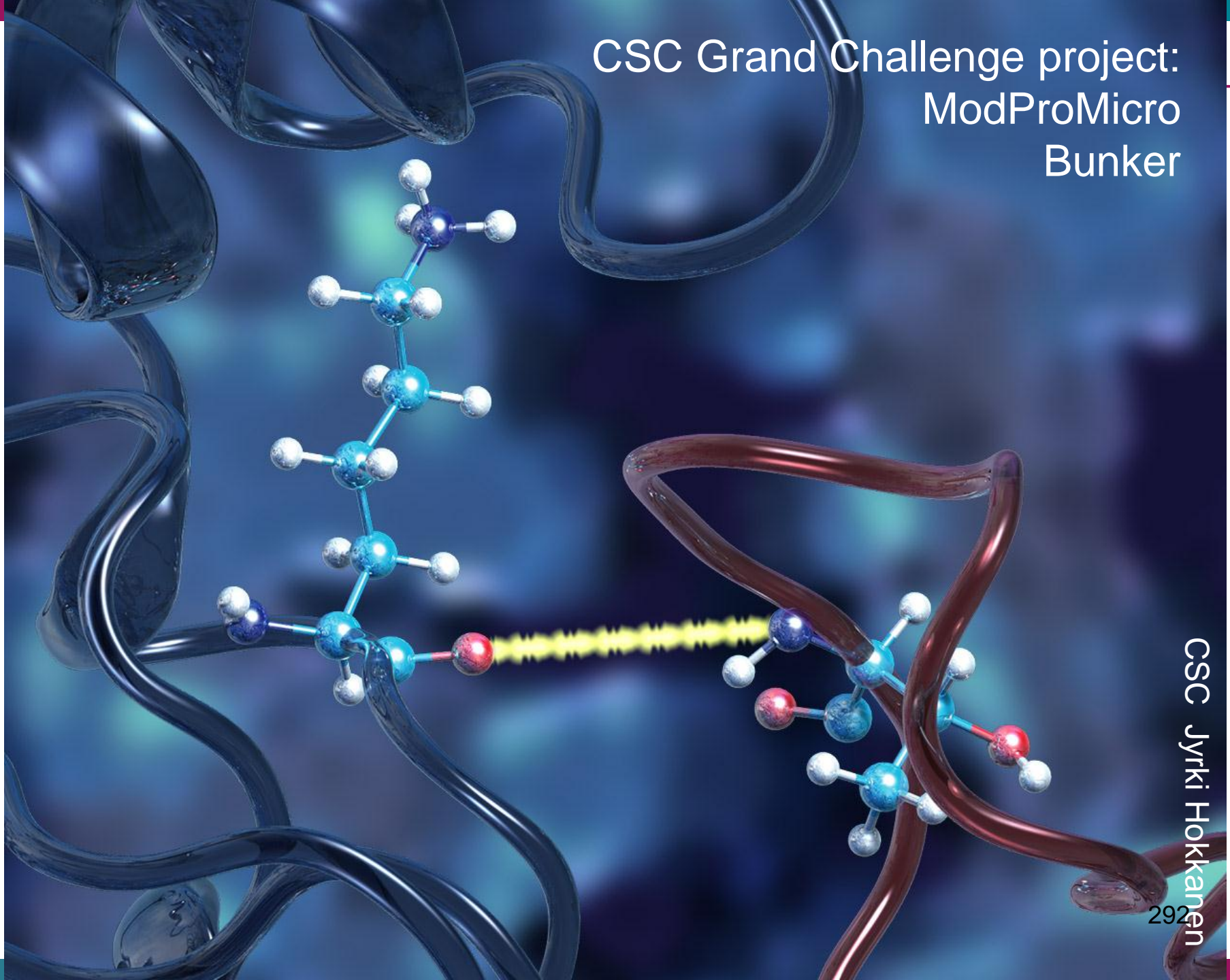


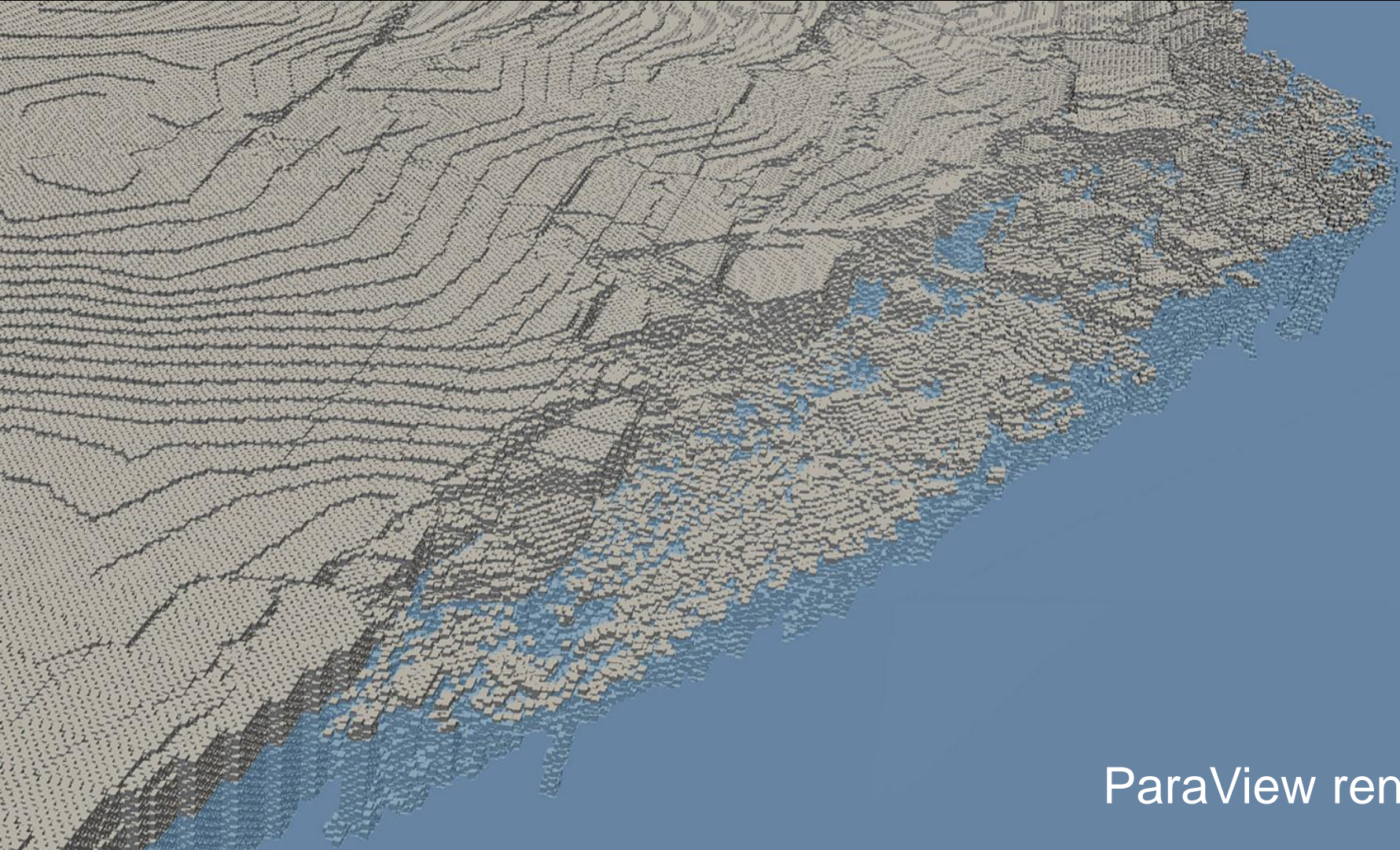




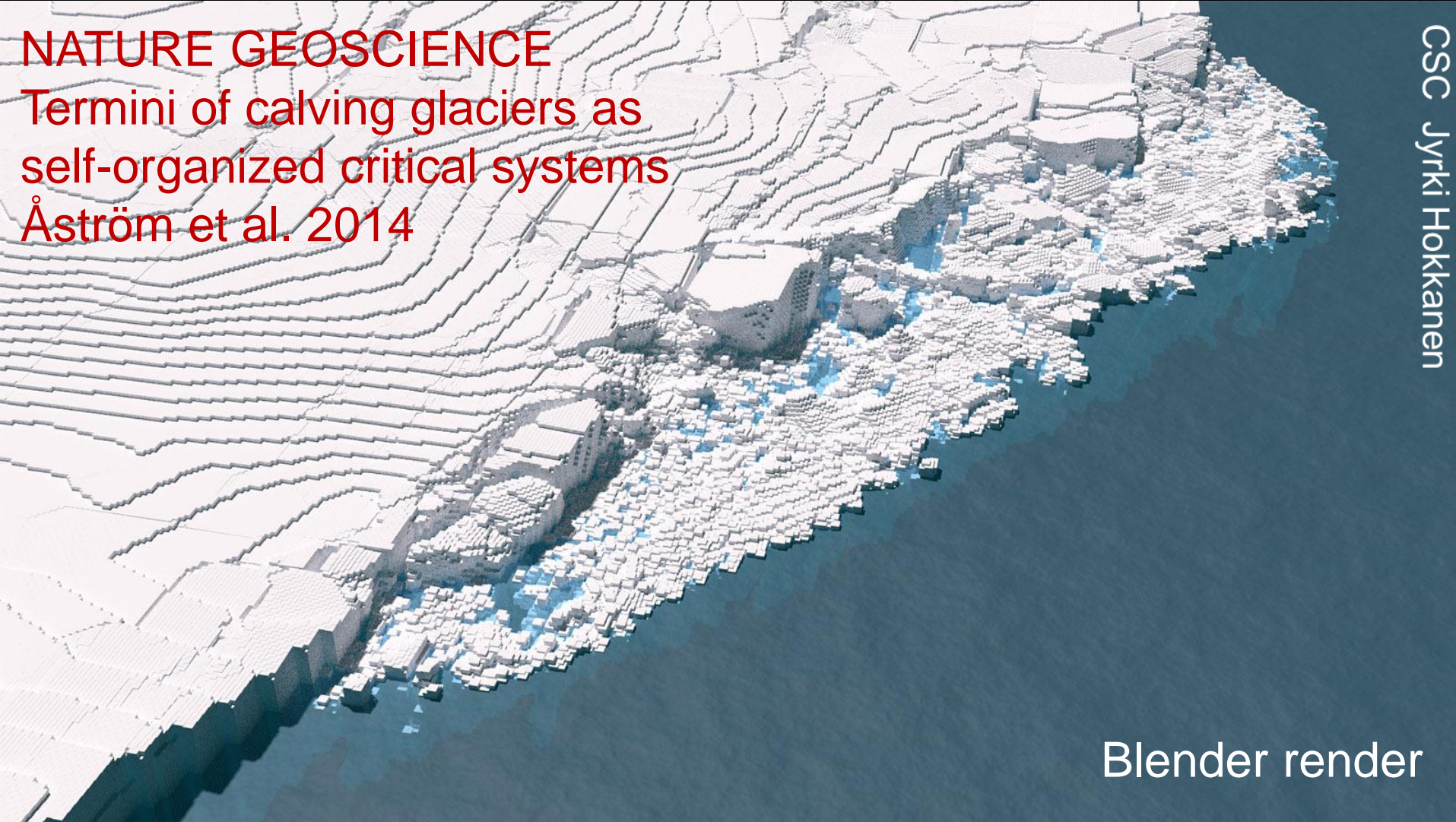


CSC Grand Challenge project: ModProMicro Bunker





ParaView render



NATURE GEOSCIENCE
Termini of calving glaciers as
self-organized critical systems
Åström et al. 2014

CSC Jyrki Hokkanen

Blender render



Blender render

ParaView hands-on session



Our short ParaView session is an extract from *The ParaView Tutorial*

http://www.paraview.org/Wiki/The_ParaView_Tutorial

Links for further information:

The ParaView Users Guide

http://www.paraview.org/Wiki/ParaView/Users_Guide/Table_Of_Contents

Comprehensive *ParaView wiki*

<http://paraview.org/Wiki/ParaView>

Video *Introduction to ParaView* (1h 20min)

<https://vimeo.com/41009606>

Search *ParaView users mailing list*

<http://paraview.markmail.org>

ParaView on CSC's servers



Most people run ParaView locally on their desktop PC. You might want to use ParaView on CSC's servers if you need more power/memory or your big data sits on CSC's disks.

Information about ParaView installations on CSC servers
<https://research.csc.fi/-/paraview>

Tips

- the old Hippu's were for interactive use, are about to be run down
- Taito-shell is the new Hippu, intended for interactive use
- Taito is for batch jobs but you can allocate time via SLURM for interactive use
- use NX for faster interactive graphics (<https://research.csc.fi/-/nomachine>)
- running ParaView in parallel mode via MPI is useful only if interaction is slow and your data is suitable for parallel tasks (is a structured file, for example)

Wrap-up of day 2 and concluding remarks