



Kimmo Mattila
Ari-Matti Sarén



CSC BioWeek 2016

Using Taito cluster for high throughput data analysis

3.-4. 2. 2016



CSC Environment

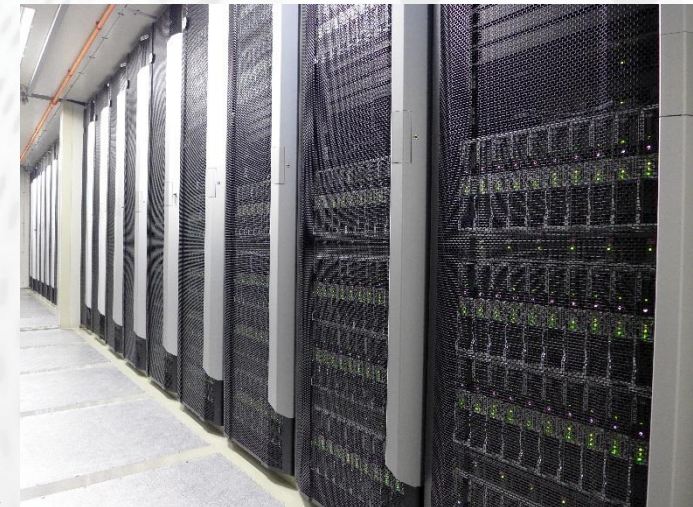
Sisu

- Cray XC40 Massively Parallel Processor (MPP) supercomputer
 - 3376 12-core 2.6-GHz Intel Haswell 64-bit processors
 - 40512 cores
 - 2,67 GB memory/core
 - Aires interconnects
- Meant for jobs that parallelize well
 - Normally 64-4096 cores/job (MPI)
 - can be increased for Grand Challenge projects
- Modest selection of bioinformatics tools
 - Molecular dynamics codes: gromacs, namd, Amber
- Sisu user's guide
 - <http://research.csc.fi/sisu-user-guide>



HP Apollo 6000 XL230a/SL230s Supercluster

- **New nodes:** 2 x 12 core 2.6 GHz Intel Haswell processors
 - 397 nodes with 128 GB memory (5,3 GB/core)
 - 10 nodes with 256 GB memory (10,7 GB/core)
 - Total of 9768 Haswell cores
- **Old nodes:** 2 x 8-core 2.6 GHz Intel Sandy Bridge processors
 - 496 nodes with 64 GB memory (4 GB/core)
 - 16 nodes with 256 GB memory (16 GB/core)
 - 2 nodes with 1,5 TB memory and 32 cores (47 GB/core)
 - 4 login nodes with 64 GB memory (4 GB/core)
 - Total of 9344 Sandy bridge cores
- Also GPU and MIC nodes available
- Meant for serial and mid-size parallel jobs
- 1-448 cores/job (more posible after scalability tests)
- Maximum of 896 simultaneous jobs/user
- Taito user's guide: <http://research.csc.fi/taito-users-guide>



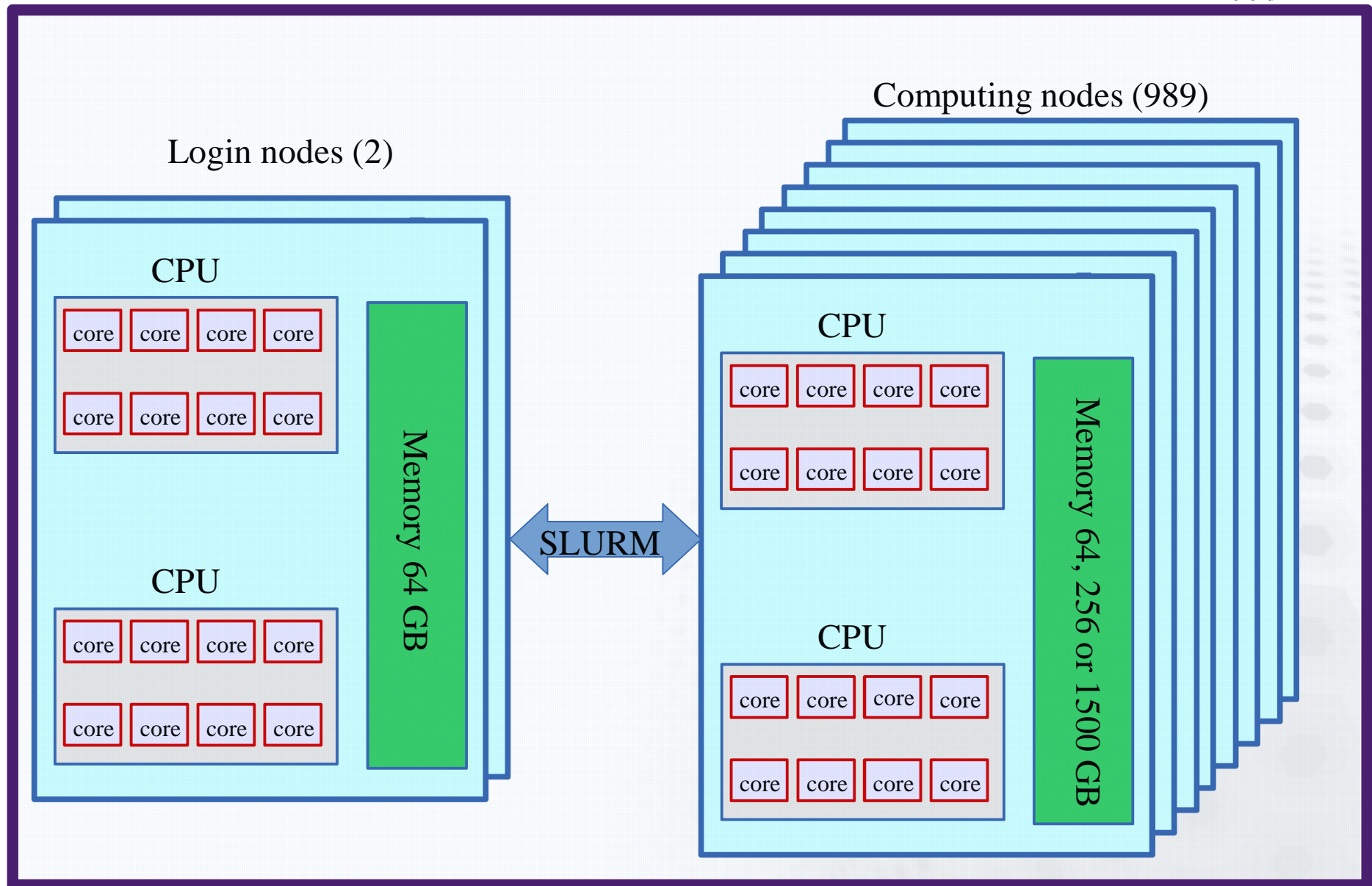
Taito-shell.csc.fi

- Service for interactive computing
- Direct access to an interactive batch queue in Taito cluster
- Limits: 4 cores, 128 GB of memory, unlimited runtime.
- When you log out, all processes will be terminated

`ssh taito-shell.csc.fi`

- User's guide: <https://research.csc.fi/taito-shell-user-guide>

Taito cluster



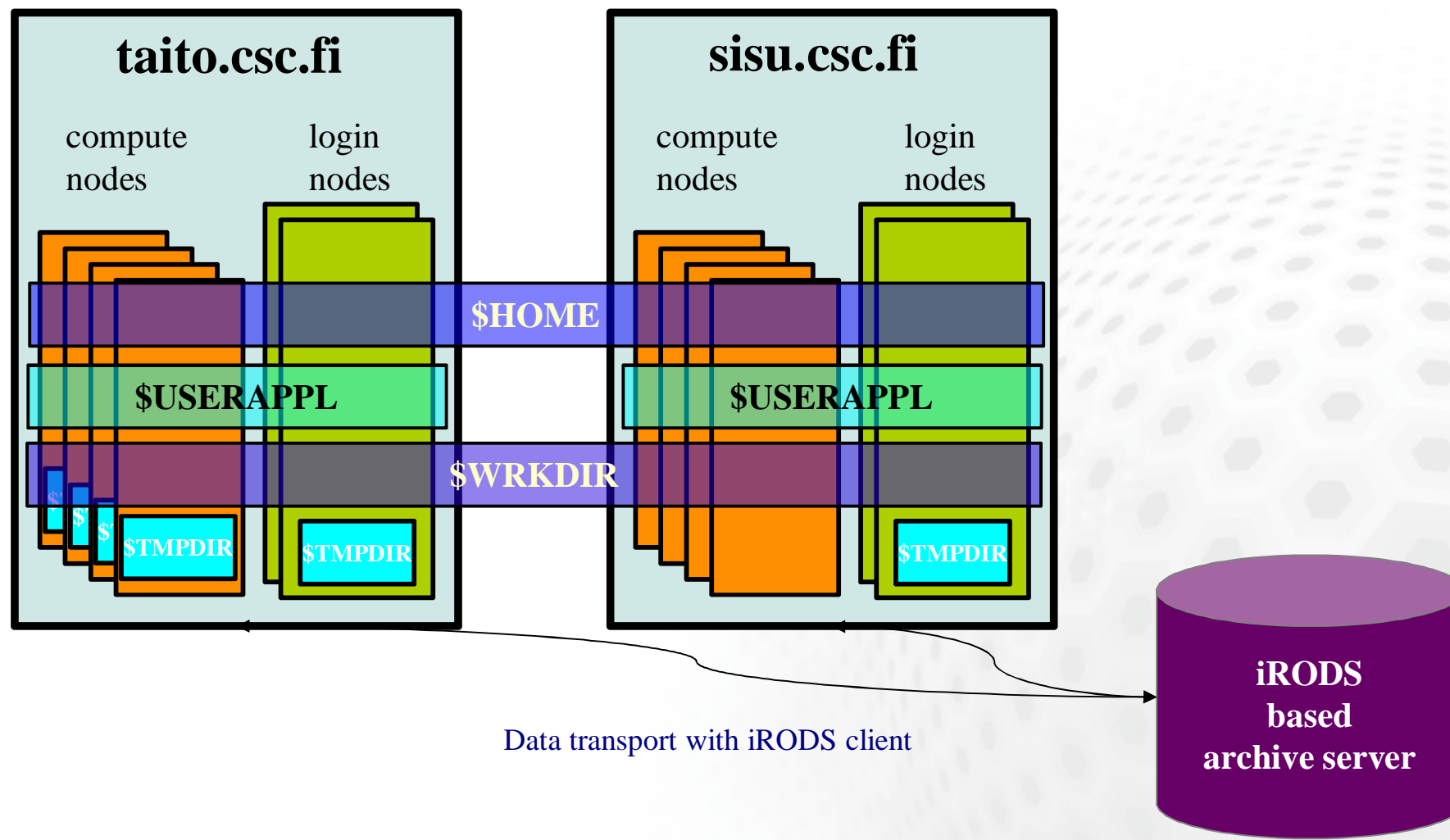
File systems and directories



Directory	Intended use	Default quota/user	Storage time	Backup
\$HOME	Initialization scripts, source codes, small data files. Not for running programs or research data.	50 GB	permanent	Yes
\$USERAPPL	Location for users' own application software installations	50 GB	permanent	Yes
\$TMPDIR	Run-time temporary files.		~ 2 days	No
\$WRKDIR	Temporary data files.	5 TB	Until further notice.	No
project	Common storage for project members. A project can consist of one or more user accounts.	On request	permanent	Yes
HPC-Archive	Long term storage	2 TB	permanent	Yes

<http://research.csc.fi/csc-guide-directories-and-data-storage-at-csc>

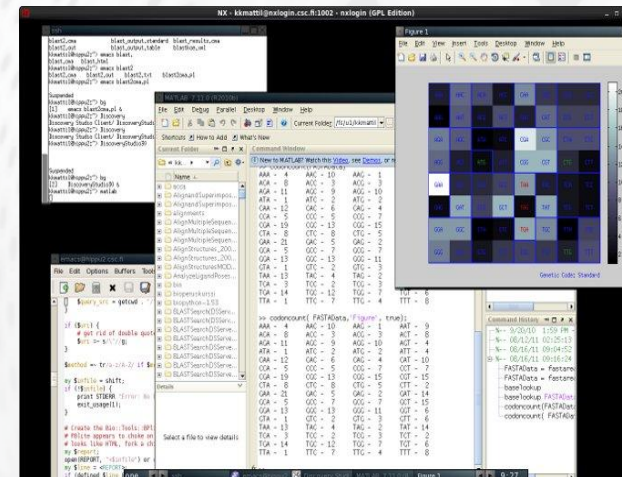
File system (Sisu and Taito)



Connecting to the Servers of CSC

- Terminal connections (ssh, PuTTY, SUI)
 - usage through typed commands
 - Graphics requires Xterm connection
- Scientist's User Interface
 - Usage through web interface
 - Mostly used for managing your account and files
 - No bioscience applications
- NoMachine/FreeNX virtual desktop
 - requires local client installation
 - Norman terminal connections can be used
 - Enables using graphical interfaces and displaying images

<http://research.csc.fi/csc-guide-connecting-the-servers-of-csc>





Managing files in unix command line

<http://research.csc.fi/csc-guide-linux-basics-for-csc>



Unix/linux commands

Basic syntax:

comand -option argument

```
ls
```

```
ls -l
```

```
ls -l myDirectory
```

Use *man* command to get information about possible options

```
man ls
```

Commands for directories:

<code>cd</code>	change directory
<code>ls</code>	list the contents of a directory
<code>pwd</code>	print (=show) working directory
<code>mkdir</code>	make directory
<code>rmdir</code>	remove directory

Commands for files:

cat	print file to screen
cp	copy
less	view text file
rm	remove
mv	move/rename a file
head	show beginning of a file
tail	show end of a file
grep	find lines containing given text
wc	count number of words or lines
file	check the type of the file



Special characters:

*(asterisk), wild card, means any text

```
ls *.fasta
```

| (pipe) guides output of a command to an input of another commands

```
ls *.fasta | less
```

> Writes output to a new file

```
ls > files_of_the_directory.txt
```

~ (tilde) means your home directory as does \$HOME

```
cp test.txt ~/file.txt
```

```
cp test.txt $HOME
```

& runs command in background

```
gzip my_big_file.tar &
```

\ (backslash) escape, used to tell the system to ignore special meanings

```
cp this\ filename\ has\ spaces.txt $WRKDIR
```


Piping

- It is also possible to "pipe" output of one command to another command using "|" characters
- This can be faster than using files as there is no disk I/O

```
ls -l | less
```

```
cat myfile.txt | sort | uniq
```



Redirection

- It is often useful to redirect the output (stdout) of a command to a file
 - ">" will overwrite the contents
 - Try:

```
ls > filelist
cat filelist
```
 - Depending on your bash settings, may cause error if target file exists
 - ">>" will append to a file
 - Try:

```
echo "one" > test
echo "two" > test
echo "three" >> test
cat test
```
- Sometimes it's necessary to capture stderr as well

```
command > out.file 2> err.file
```

Redirection

- Redirection can also be done in the other direction
 - Redirect the contents of the file to the standard input (stdin) of a command

```
cmd < file
```
 - Redirect a bunch of lines to the stdin. If 'EOL' is quoted, text is treated literally.

```
cmd << EOL  
line1  
line2  
EOL
```
 - Redirect a single line of text to the stdin of a command

```
cmd <<< "string"
```

Variables and arrays

To set a variable:

```
variable=value
```

To use a variable

```
$variable
```

```
var1="Hello"
```

```
var2="World"
```

```
echo $var1 $var2
```

To set an array

```
array=( value1 value2 valueN )
```

To use a value in an array (note: zero based)

```
${array[n]}
```

```
array=( a b c )
```

```
echo ${array[1]}
```

Variables and arrays

Sometimes it is necessary to separate variable name from rest of the command:

This would not work:

```
sed -n ${SLURM_ARRAY_TASK_ID}p namelist
```

So instead we can use:

```
sed -n ${SLURM_ARRAY_TASK_ID}p namelist
```

or

```
sed -n "$SLURM_ARRAY_TASK_ID"p namelist
```

Environment variables

- Normal variables only visible to the process that set them
- To make a variable visible also to any child processes (e.g. any programs run from a shell), you must use **export** command:

```
export PATH=${PATH}:${USERAPPL}/mcl/version-12-068/bin
```
- Typical examples are the system variables that point to different file system locations: **\$HOME**, **\$USERAPPL**, **\$WRKDIR** etc
- SLURM has it own set of useful system variables:
\$SLURM_CPUS_PER_TASK, **\$SLURM_ARRAY_TASK_ID** etc

Quotes

➤ Different quotes have different functionalities

- ' ' Take text enclosed within quotes literally
- ` ` Take text enclosed within quotes as command and replace with output
- " " Take text within quotes literally after substituting any variables

➤ Compare the results of these commands:

```
var="test"; echo 'echo $var'  
var="test"; echo `echo $var`  
var="test"; echo "echo $var"
```



Some useful commands for parsing lines

Try these to see what they do!

sed

```
echo "one this two this three" | sed s/this/that/  
echo "one this two this three" | sed s/this/that/g
```

awk

```
echo "one two three" | awk '{print $2}'  
echo "one;two;three" | awk -F";" '{print $2 $3}'
```

cut

```
echo "123456789" | cut -c 4  
echo "123456789" | cut -c -4  
echo "123456789" | cut -c 4-  
echo "123456789" | cut -c 4-7  
echo "one_two_three" | cut -d "_" -f 2
```

All of these have much more options. See `man` pages for details.



Some useful commands for parsing lines

grep is a powerful tool for finding regular expressions in files

<code>grep pattern file</code>	returns the lines from <code>file</code> containing the pattern
<code>grep -c pattern file</code>	returns the count of lines containing the pattern
<code>grep -v pattern file</code>	reverses output, <i>i.e.</i> returns lines not containing the pattern
<code>grep -w pattern file</code>	returns only complete word matches
<code>grep -f file1 file2</code>	returns lines in <code>file2</code> that also exist in <code>file1</code>
<code>grep "^pattern" file</code>	<code>^</code> matches beginning of line
<code>grep "pattern\$" file</code>	<code>\$</code> matches end of line

It's good to remember that **grep** operates line by line, *i.e.* matches separated into two lines are not found.



Data handling

Some brief generalizations:

- Directories containing tens of thousands of files are often problematic (use subdirectories and/or aggregation)
- It's usually faster to move one large file than many small ones
- On the other hand you should avoid too large files
 - it's nicer to re-send one 100 GB chunk than the whole 1 TB file
- Consider compression
- Prefer file formats that have checksums or other verification mechanisms
- Data should be packaged for saving in Archive server or IDA

➤ tar

- concatenates a set of files into one file. Does not compress by default
- preserves directory structure
 - many compression programs don't handle directories well/at all
 - answer: first tar, then compress
- making a tar package:

```
tar cf myfolder.tar myfolder
```
- opening a tar package:

```
tar xf myfolder.tar
```
- checking tar file contents

```
tar tf myfolder.tar
```
- Tar can automatically use gzip (z) and bzip2 (j) compression

```
tar zcf myfolder.tgz myfolder
```

<http://research.csc.fi/csc-guide-packing-and-compression-tools#2.6.1>

File compression

- File compression/decompression takes time, but saves storage space and time on upload/download
 - net gain depends on data size
- Files used in bioinformatics (sequences, pedigree files etc) are often text-based and compress well (to ~30% of original size)
- Compressed file formats typically include checksums
 - if you can uncompress the file without error messages you know your data is intact
- Commonly used compression programs:
 - zip
 - gzip
 - bzip2

<http://research.csc.fi/csc-guide-packing-and-compression-tools>

➤ **zip**

- compressing
`zip myfiles.zip file1 file2`
- uncompressing
`unzip myfiles.zip`
- leaves original file intact

➤ **gzip**

- compressing
`gzip myfile`
- uncompressing
`gunzip myfile.gz`
- replaces original file with the compressed file

➤ **bzip2**

- slightly better compression ratio than zip/gzip
- mostly linux specific
- compressing
`bzip2 myfile`
- uncompressing
`bunzip2 myfile.bz2`
- replaces original file with the compressed file



➤ **Linux**

- tar, zip, gzip, bzip2 part of most standard distributions

➤ **Windows**

- 7-Zip

free

makes and opens tar, zip, gzip, bzip2

<http://www.7-zip.org/>

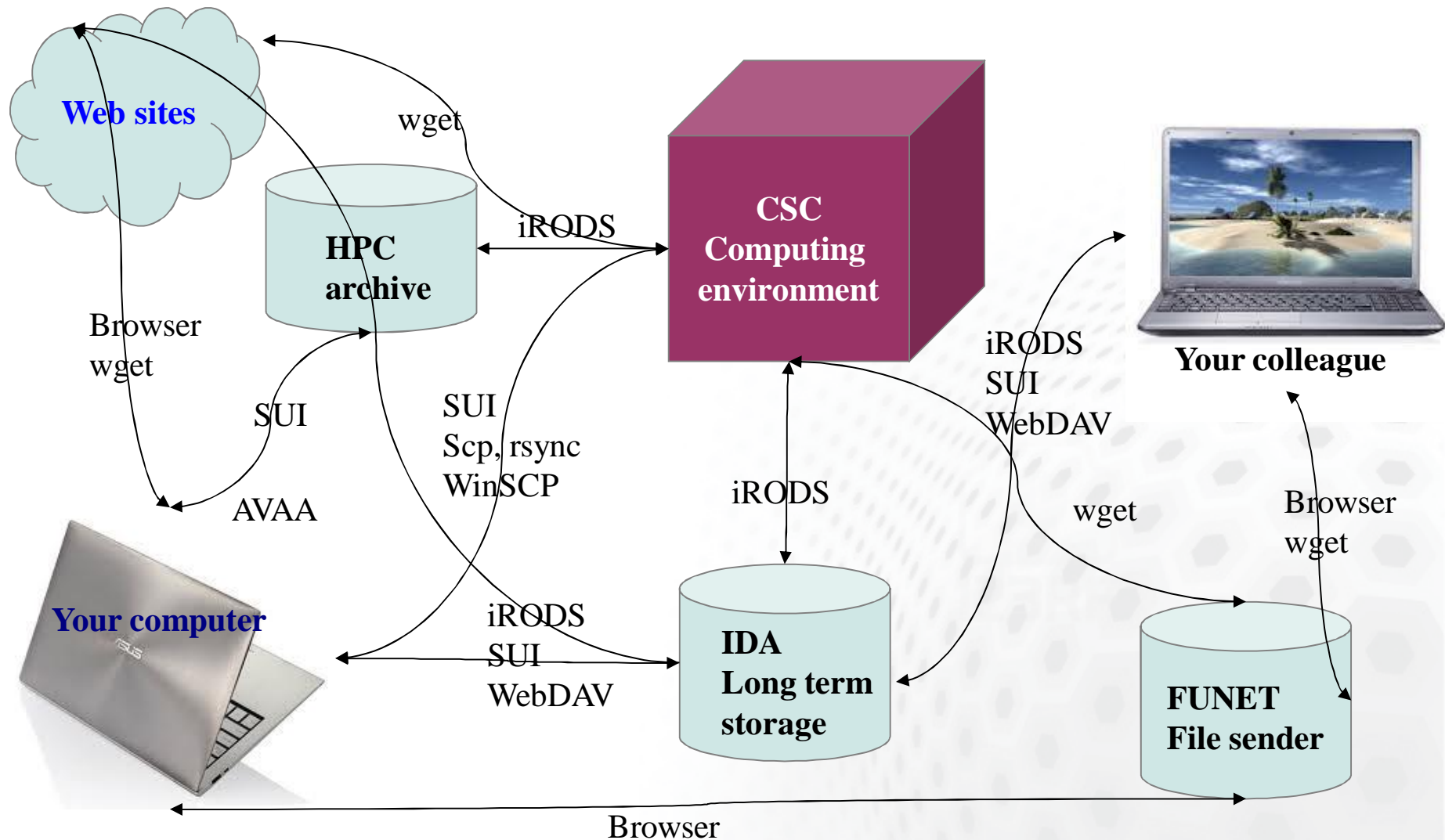
➤ **Mac**

- tar, zip, gzip available on standard installation

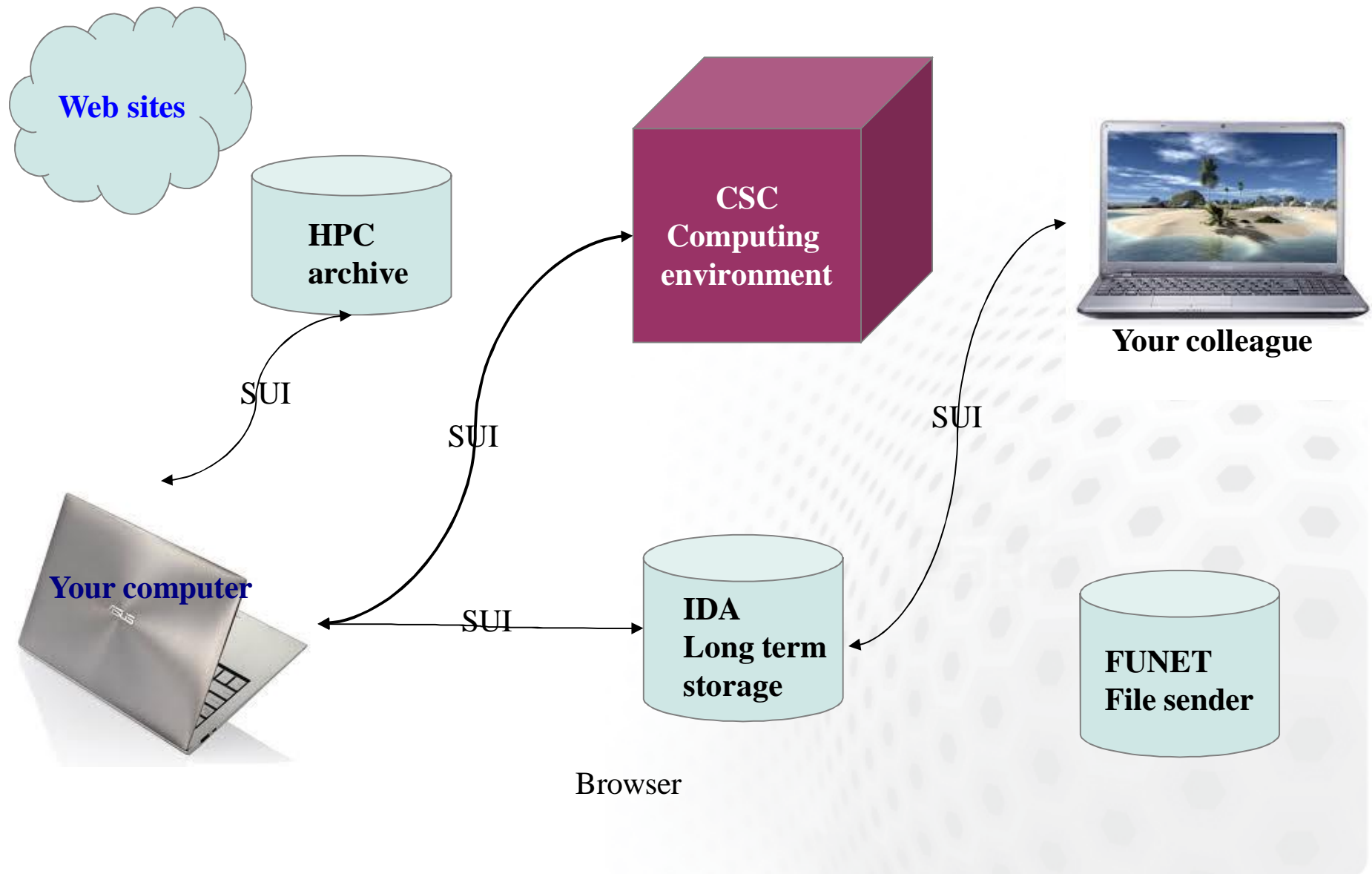
Moving data to and from CSC

<http://research.csc.fi/csc-guide-moving-data-between-csc-and-local-environment>

Moving data to and from CSC

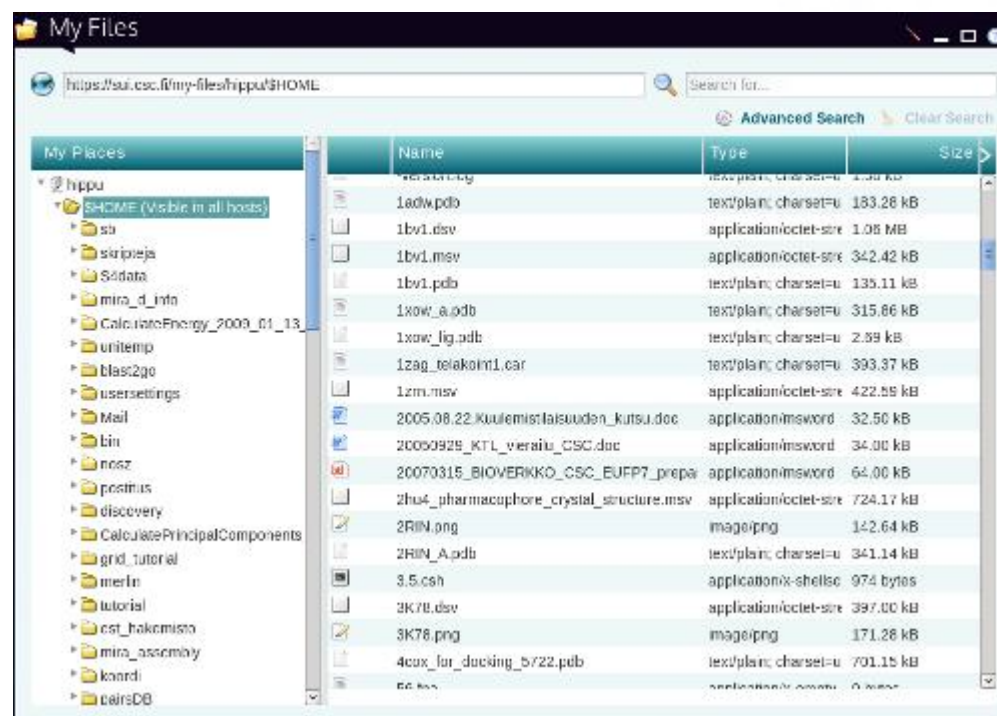


Scientist's User Interface



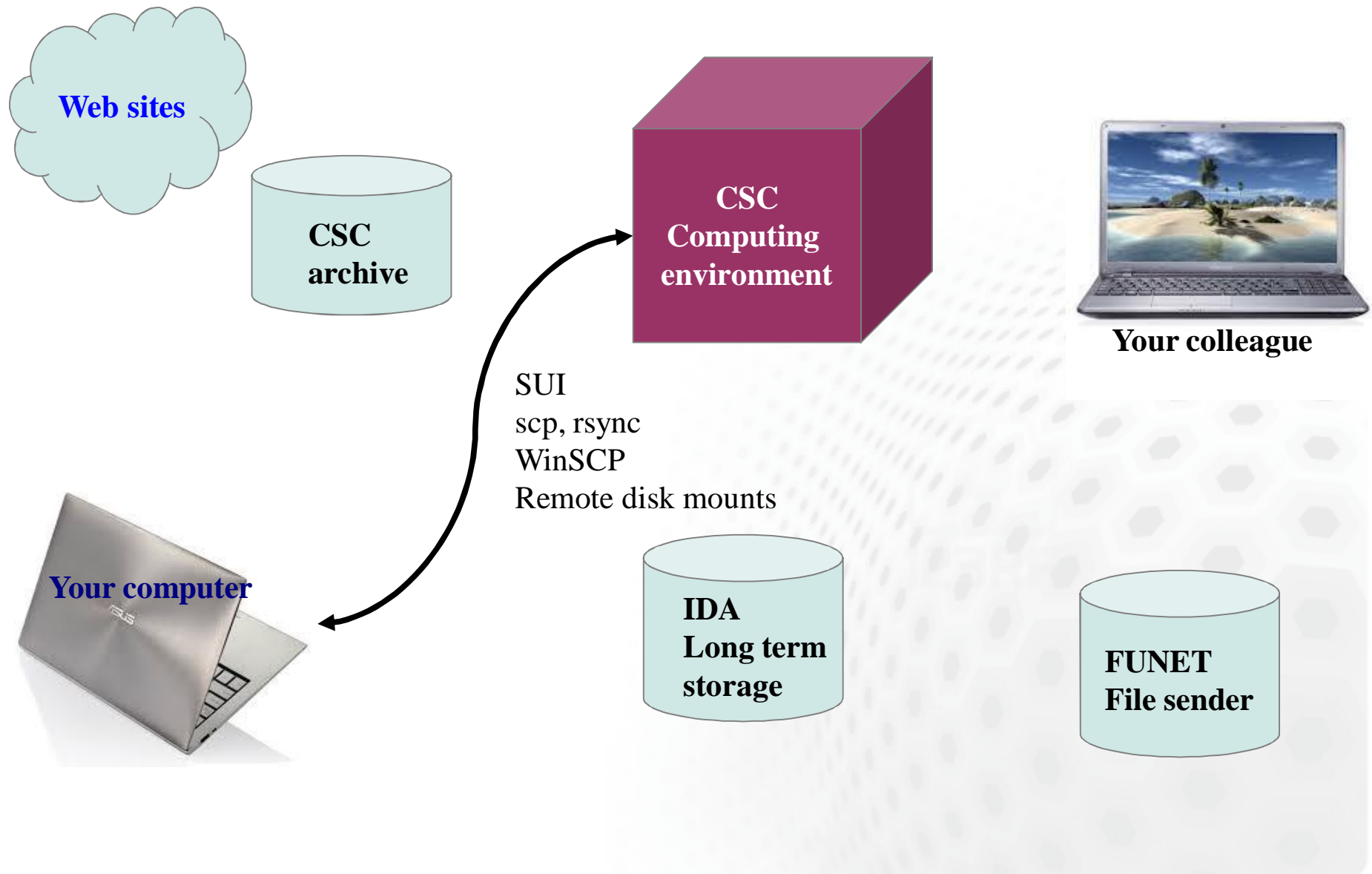
➤ SUI Scientist's User Interface

- WWW based interface for CSC
- MyFiles tool can be used for file upload and download
 - Uploading files larger than 2 GB works on some browsers (Chrome, Safari)
- GSI-SSH Console based SFTP for larger files



<http://research.csc.fi/csc-guide-data-transport-with-scientist-s-user-interface>

Traditional data transport tools



File transport tools for Linux and Mac



- scp is a standard tool and works well

```
scp myfiles.tar.gz 'user1@hippu3.csc.fi:$WRKDIR'  
scp 'user1@hippu3.csc.fi:$WRKDIR/myfiles.tar.gz'
```

<http://research.csc.fi/csc-guide-copying-files-from-linux-and-mac-osx-machines-with-scp>

- rsync can be used for data mirroring and moving very large files and directories

```
rsync -avz -e ssh my_data kkayttaj@hippu4.csc.fi:/wrk/kkayttaj
```

<http://research.csc.fi/csc-guide-using-rsync-for-data-transfer-and-synchronization>

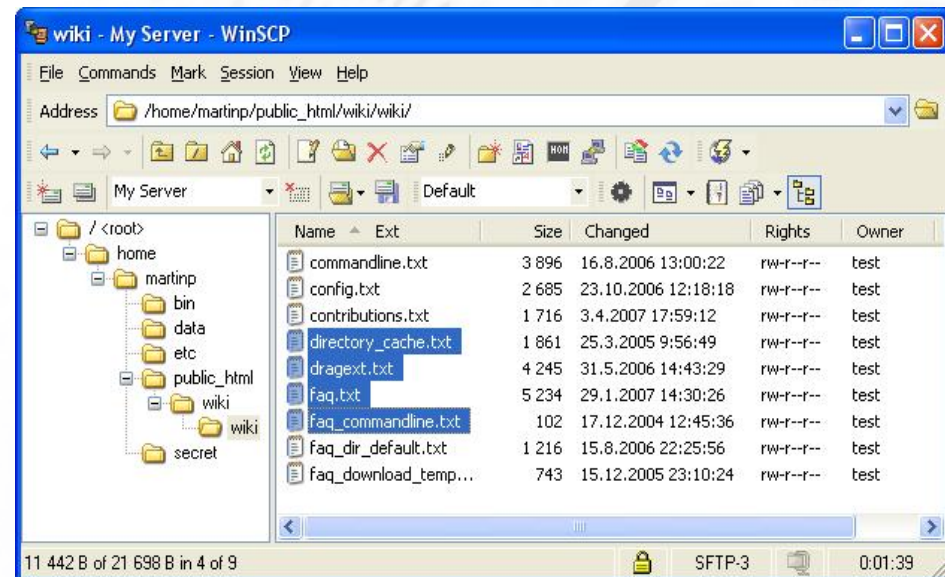
- Several graphical file transport tools exists
 - e.g. Fugu for mac

File transport tools for Windows



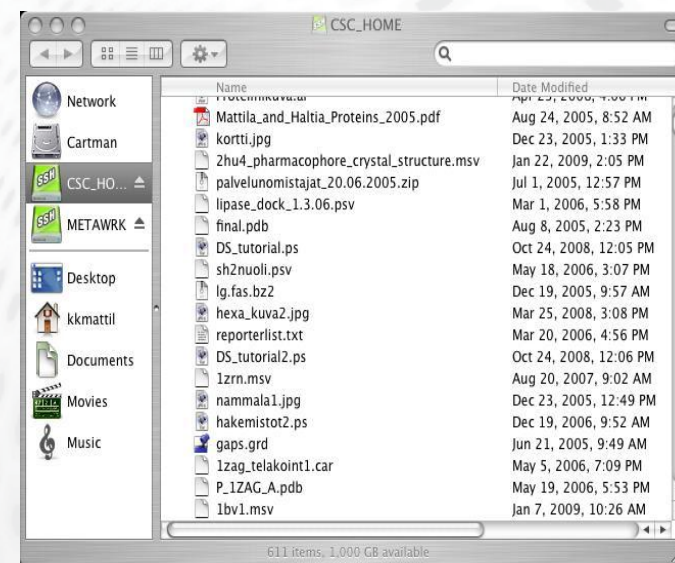
- most commercial ssh programs have graphical file moving interfaces
- commonly used PuTTY does not (it does have command line based scp and sftp)
- winSCP is good free option

<http://winscp.net/eng/index.php>



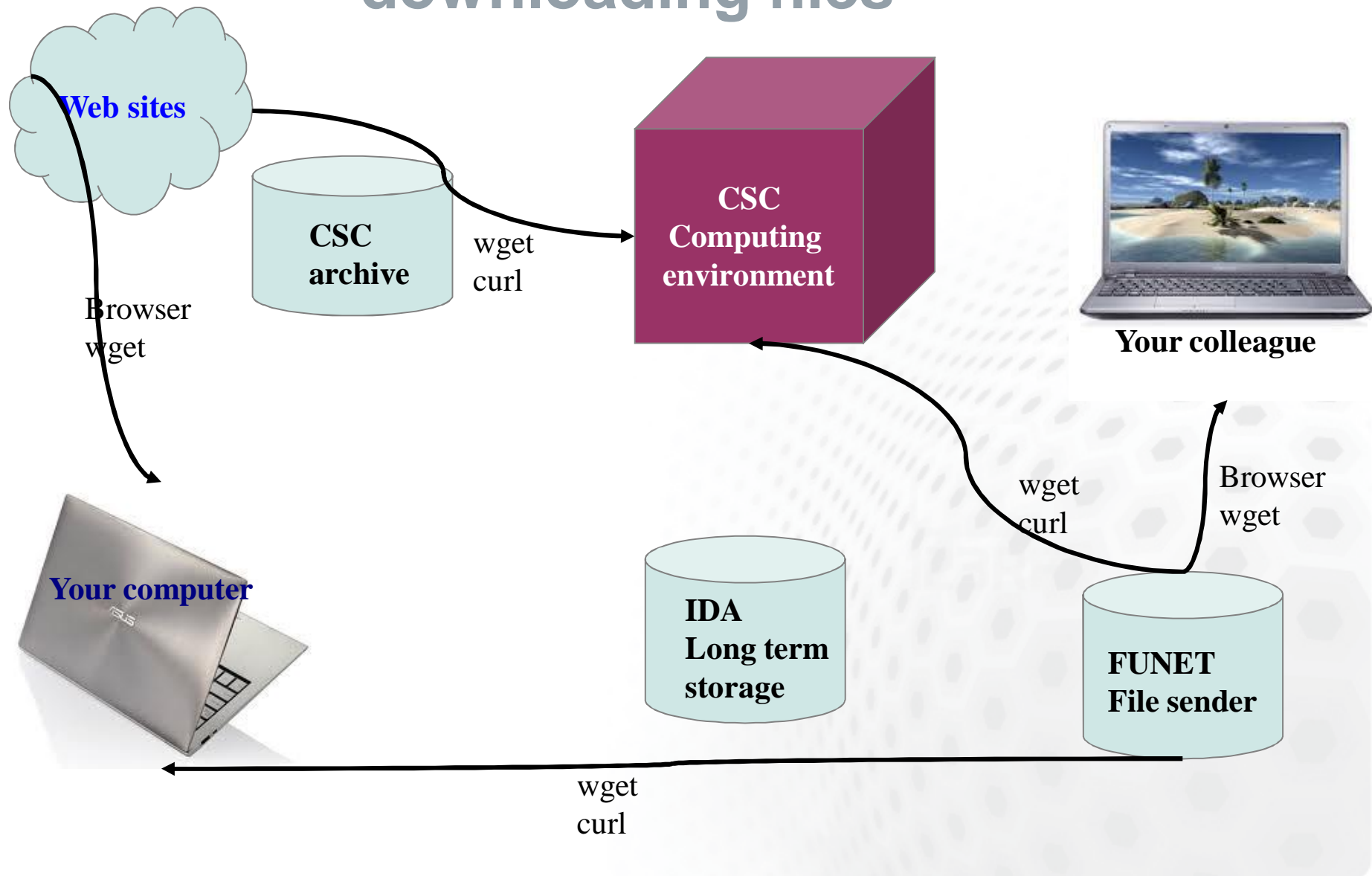
Remote disc mounts

- Fuse (linux) OSXFUSE (Mac) allow you to mount you disk areas at CSC to your local computer
- With this arrangement you can use locally installed tools to work with data that locates at CSC
- Can be used in to link CSC directories to virtual machines (linux) running in cPouta cloud service



http://www.csc.fi/english/pages/data-services/transport/remote_mounts

Wget and curl: command line tools for downloading files



wget and curl: command line tools for downloading files



- **wget** and **curl** are simple command line tools to download data from a given URL
- handy tools to move a file from internet to the servers of CSC

Syntax:

wget URL

curl URL > file

For example:

```
wget ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chrY.fa.gz
```

Use your local browser to locate the file you need and then download it directly to CSC by using *wget* or *curl* command in Sisu or Taito.

Bioinformatics specific tools



- **wget** and **curl** are general purpose tools for data download.

Some data sources can be used through source specific tools in Taito

- **Ensemblfetch**: download genomes from ensembl:

```
ensemblfetch pseudomonas_aeruginosa_pa7
```

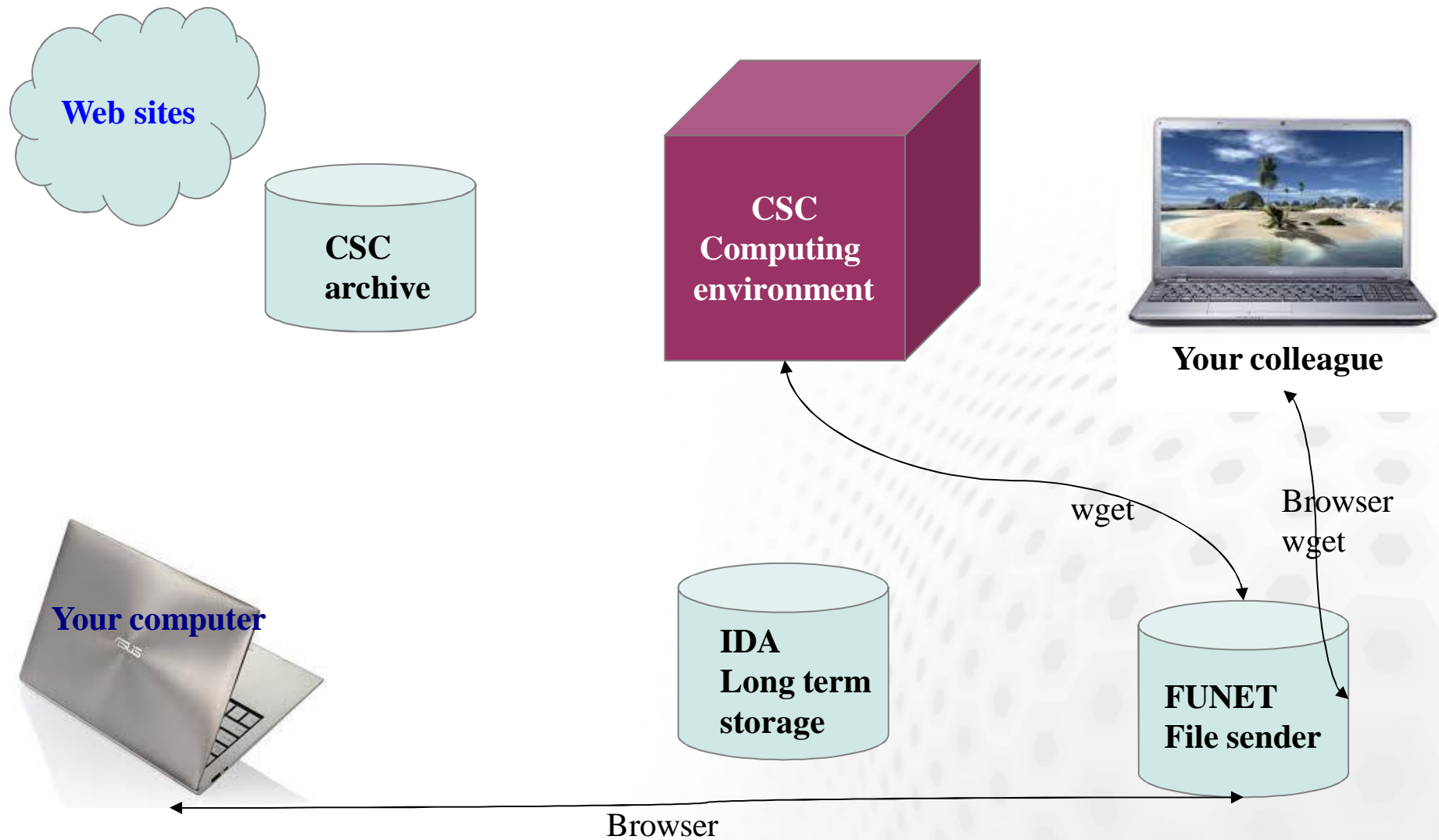
- **Edirect** package to download data from NCBI:

```
esearch -db nucleotide -query "NC_009656" | \  
efetch -format fasta > NC_009656.edirect.fna
```

- **SRA-toolkit** to download data from SRA databases:

```
sam-dump SRR490207 > SRR490207.sam
```


FUNET File Sender

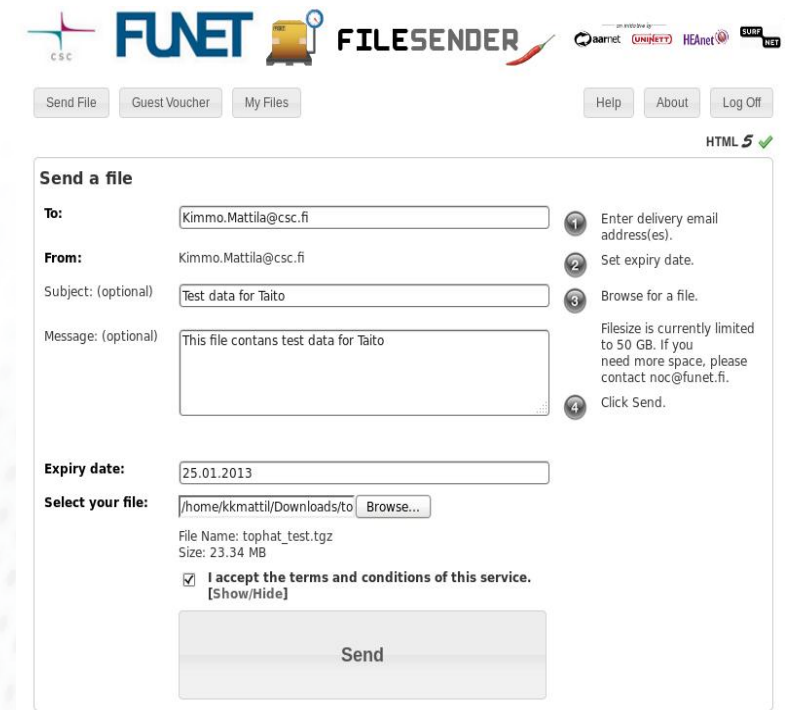


Funet FileSender

<https://filesender.funet.fi/>



- DropBox like service provided by CSC
- Intended to be used as a replacement for e-mail attachments
- Browser based, no need for client installations
- Maximum file size 200 GB
- Storage time 14 days
- Uploaded data is readable to anybody (who knows the link)
- Can be accessed from CSC too with wget and curl.

The screenshot shows the Funet FileSender web interface. At the top, there are logos for CSC, FUNET, and FILESENDER, along with navigation links like 'Send File', 'Guest Voucher', 'My Files', 'Help', 'About', and 'Log Off'. The main section is titled 'Send a file' and contains a form with fields for 'To:', 'From:', 'Subject: (optional)', and 'Message: (optional)'. The 'To:' field is filled with 'Kimmo.Mattila@csc.fi'. The 'From:' field is also 'Kimmo.Mattila@csc.fi'. The 'Subject:' field is 'Test data for Taito'. The 'Message:' field contains 'This file contains test data for Taito'. To the right of the form, there are numbered instructions: 1. Enter delivery email address(es), 2. Set expiry date, 3. Browse for a file, and 4. Click Send. Below the form, there is an 'Expiry date:' field set to '25.01.2013' and a 'Select your file:' field with a 'Browse...' button. Below these, it shows 'File Name: tophat_test.tgz' and 'Size: 23.34 MB'. A checkbox is checked with the text 'I accept the terms and conditions of this service.' and a '[Show/Hide]' link. At the bottom, there is a large 'Send' button.

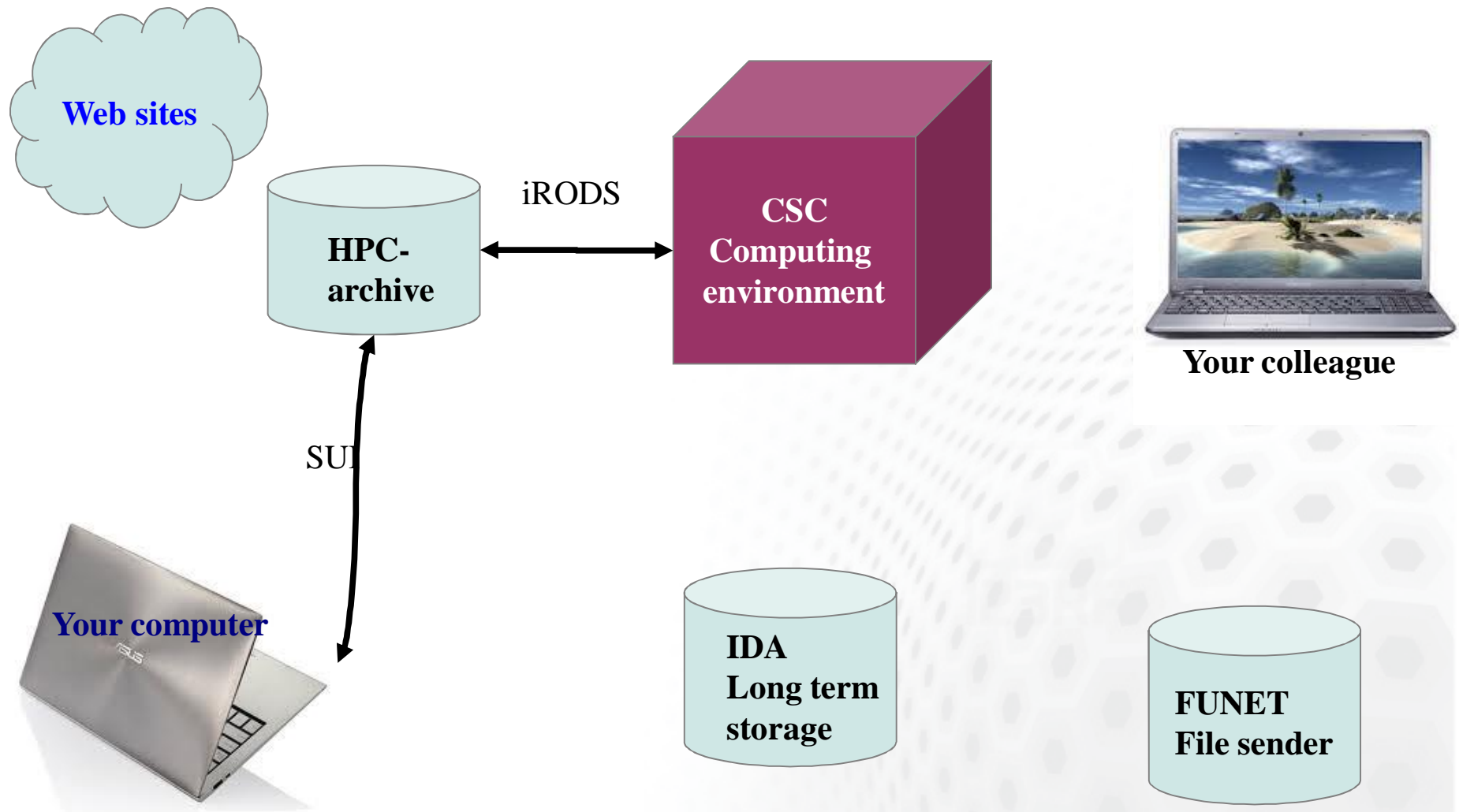


HPC-Archive and IDA storage services

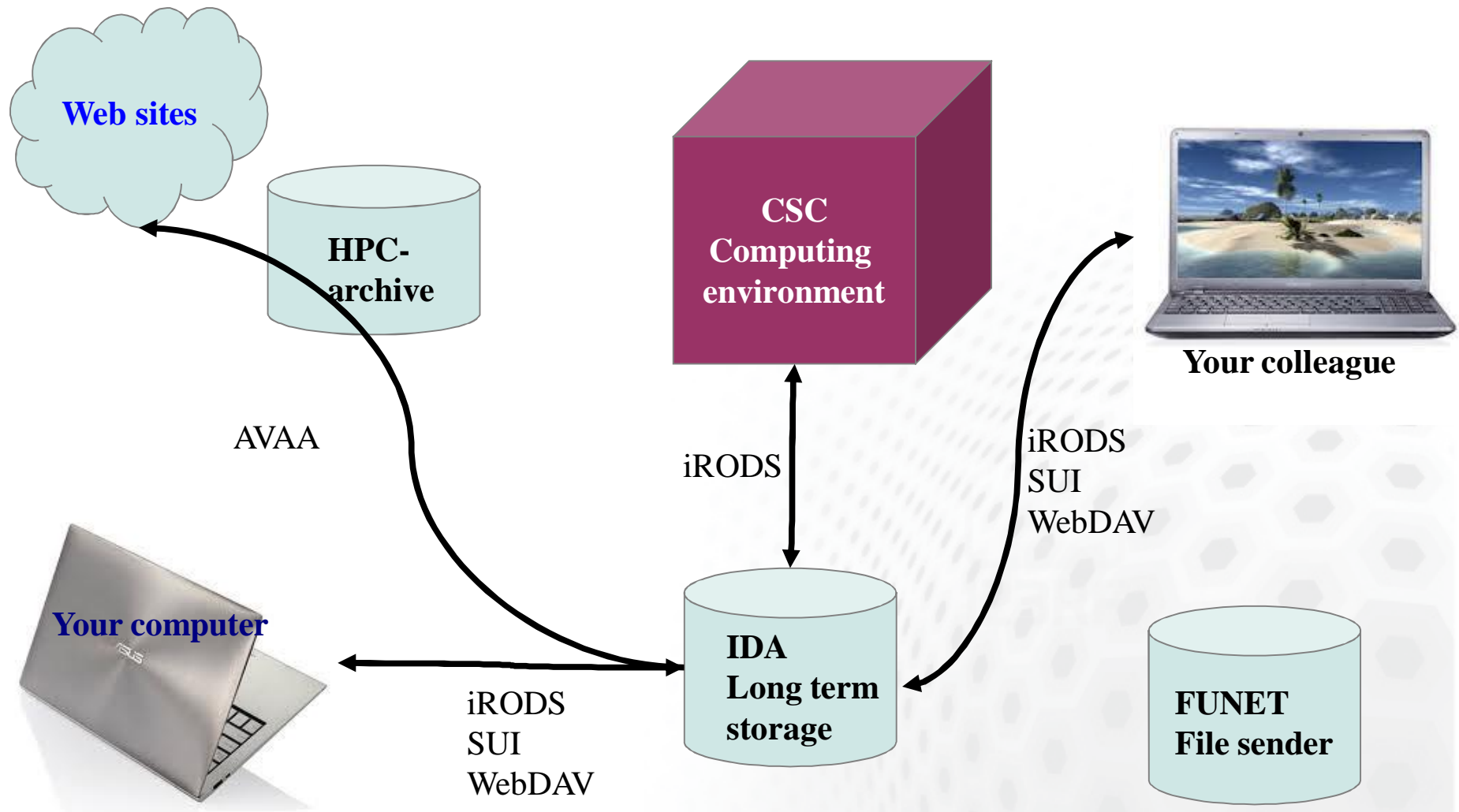
Based on iRODS technology
(Integrated Rule-Oriented Data System)

<http://www.tdata.fi/ida>

HPC archive



IDA storage service



Using archive servers



- Storage service – not a mounted disk
 - very large capacity
 - for stable datasets only
 - retrieving the files may take a few minutes
 - Speed: about 2 GB/min at the servers of CSC (1 TB takes ~ 1,5 days!)
- Can be accessed with SUI and iRODS commands: iput, iget etc.
 - Visible to Sisu and Taito
- You have to reconfigure your iRODS connection when you want start using another archive server.
- Avoid archiving small individual files on the servers

If you have to archive small files, you should first combine them to tar format and compress

<http://research.csc.fi/csc-guide-archiving-data-to-the-archive-server>

IDA storage service

- iRODS based storage system for storing, archiving and sharing scientific data
- The service was launched 2012. Guaranteed until 2017.
- Usage through personal accounts and projects
- Each project has a shared directory
- Speed: about 2 GB/min at the servers of CSC (1 TB takes ~ 1,5 days!)
- CSC host's the service but universities and academy of Finland allocate the storage space

Three interfaces:

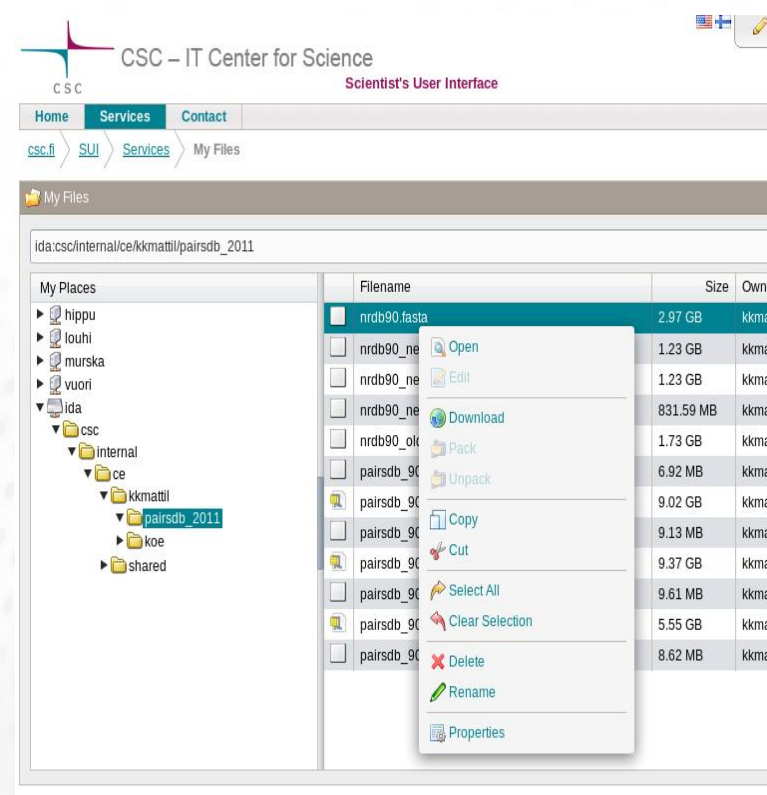
- WWW interface in Scientists' User Interface
- network directory interface for Mac and Windows
- command line tools (i-commands are available at CSC)

HPC Archive and IDA interfaces at CSC

Some iRODS commands

- `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
- `irsync` synchronize the local copy with the copy in IDA
- `imkdir` create a directory to IDA
- `iinit` Initialize your IDA account
- `imeta command` manage metadata

IDA In Scientist's User Interface



IDA

- Part of ATT, you need to apply for a storage quota.
- Quotas granted by universities and Academy of Finland
- Several interfaces (WWW/SUI, network disk, i-commands)
- Internet accessible
- Project based structure
- Data can be made public through Etsin (metadata) and AVAA (data)
- <http://avointiede.fi/ida>

HPC-archive

- Part of CSC computing environment
- 2,5 TB default quotas for CSC users
- Usage with i-commands
- Visible only to CSC environment
- Personal storage area
- <https://research.csc.fi/csc-guide-archiving-data-to-the-archive-servers>

EUDAT <https://eudat.eu/>



B2DROP

- Secure and trusted data exchange service for researchers and scientists to keep their research data synchronized and up-to-date and to exchange with other researchers

B2SHARE

- A user-friendly, reliable and trustworthy way for researchers, scientific communities and citizen scientists to store and share small-scale research data from diverse contexts.

B2SAFE

- A robust, safe and highly available service which allows community and departmental repositories to implement data management policies on their research data across multiple administrative domains in a trustworthy manner

Module system on Taito

Module system



- Different software packages have different, possibly conflicting, requirements.
- **LMOD module system** is used to manage software and programming environments
- **module load biokit** sets up most of the bioinformatics tools (but not all the tools)
 - See software web pages for details

Most commonly used module commands:



<code>module load <i>modulename</i></code>	Loads the given environment module
<code>module load <i>modulename/version</i></code>	
<code>module list</code>	List the loaded modules
<code>module avail</code>	List modules that are available to be loaded (i.e. compatible with your current environment)
<code>module spider</code>	List all existing modules
<code>module spider <i>name</i></code>	Searches the entire list of existing modules
<code>module swap <i>module1 module2</i></code>	Replaces a module with a another module and tries to re-load compatible versions of other loaded modules
<code>module unload <i>modulename</i></code>	Unloads the given environment module
<code>module purge</code>	Unloads all modules



Running jobs on CSC servers

Types of jobs



- Serial jobs
 - Use only one core
 - Many older bioinformatics tools

- Embarrassingly parallel tasks:
 - Job can be split to numerous sub jobs
 - You can use array jobs and splitting utilizing tools like pb blast, cluster_interproscan, trinity, miso.

Types of jobs



- Threads/ OpenMP based parallelization
 - Many bioinformatics tools use this approach. Bowtie2, BWA, Tophat,
 - All the parallel processes must see the same memory -> all processes must run within one node -> can utilize max 16/24 cores
 - Applications rarely benefit from more than 4-8 cores



Types of jobs



- MPI parallelization.
 - Each task has own memory -> can utilize several nodes
 - Check scaling before launching big jobs
 - Using too many cores can actually make your job run slower



Parallel jobs

- Only applicable if your program supports parallel running
- Check application documentation on number of cores to use
 - Speed-up is often not linear
 - Maximum number of cores can be limited by the algorithms
 - Using too many cores can actually make your job run slower

Interactive vs. Batch jobs

➤ Typical interactive jobs

- Short jobs
- Serial jobs (or small shared memory parallel jobs)
- Software with GUI

➤ Typical batch jobs

- Long jobs
- Parallel jobs
- Jobs that need specific resources (e.g. hugemem nodes, GPU nodes etc.)

Interactive jobs in Taito-shell

➤ Interactive jobs are best run on Taito-shell

- Login to: `taito-shell.csc.fi`
- Resources reserved automatically
 - Currently 4 cores/128 GB memory
- no time limit on jobs
- Note: `screen`/`nohup` will not work!
 - When you log out/disconnect all jobs will be killed
- <https://research.csc.fi/taito-shell-user-guide>

➤ If the job:

- Takes long
- Can be run in batch mode
- Can use more than one core

you should consider running it as a batch job

Interactive jobs in Taito

- Only very small tasks should be done on the login nodes
- Any "real" jobs should be run on Taito-shell
 - You can use command `sinteractive` to start taito-shell session in Taito
- Bigger interactive jobs can be run on Taito by reserving resources through the batch job system
 - Mainly necessary if you need specific resources (e.g. more memory or cores than in Taito-shell)
- <https://research.csc.fi/taito-interactive-batch-jobs>



Interactive jobs in Taito

- Example using `srun`

```
srun -n 1 -mem=256000 -t02:00:00 --x11=first --pty $SHELL
module load myprog
myprog
```

Here the option "`--x11=first`" sets up the x11 connection so that graphical user interfaces can be used, and option "`--pty $SHELL`" runs the default command shell

- Example using `salloc`

```
salloc -n 32 --ntasks-per-node=16 --mem-per-cpu=1000 -t00:30:00 -p parallel
srun mdrun_mpi -s topol1 -dlb yes
srun mdrun_mpi -s topol2 -dlb yes
exit
```

screen



- screen is a virtual window manager
 - available on Taito
 - your session stays "as is" even if you disconnect
- Basic commands
 - open a new screen
`screen`
 - list open screens
`screen -ls`
 - re-attach to a screen (if only one open)
`screen -r`
 - re-attach to screen with id 12345 (as shown by `screen -ls`)
`screen -r 12345`
 - detach from screen
`screen -d`
 - screen exits when all processes (including the shell) exit. Or type
`Ctrl+a Shift+k`

screen



➤ Using screen with Taito-shell

Open a connection to Taito:

```
ssh taito.csc.fi
```

Take note of the login node. Let's assume taito-login3.

Open a screen session using command:

```
screen -R
```

In the screen session, open a taito-shell session with command:

```
sinteractive
```

When you want to leave the session running in the background, detach from screen using `Ctrl-a d`.

Now you can logout from Taito, but your screen session in taito-login3 and the Taito-shell session within it is preserved.

To reattach to your session, connect first to the Taito login node where you have your screen session running. For example:

```
ssh taito-login3.csc.fi
```

Then, reattach the screen session with command

```
screen -R
```


Batch jobs

➤ Steps for running a batch job

1. Write a batch job script

- Script format depends on server, check the user guides, e.g:
<http://research.csc.fi/taito-user-guide>
<http://research.csc.fi/sisu-user-guide>
- You can use the Batch Job Script Wizard in Scientist's User Interface:
<https://sui.csc.fi/group/sui/batch-job-script-wizard>

2. Make sure you have all the necessary input files where the program can find them

- Usually best to use \$WRKDIR
- \$HOME has limited space
- Login \$TMPDIR is not available in compute nodes

3. Submit your job

```
sbatch myscript
```

Batch jobs



- User has to specify necessary resources
 - Can be added to the batch job script or given as command line options for `sbatch` (or a combination of script and command line options)
- Resources need to be adequate for the job
 - Too small memory reservation will cause the job to use swap disk (very slow) or even fail
 - When the time reservation ends, the job will be terminated whether finished or not
- But: Requested resources can affect the time the job spends in the queue
 - Especially core number and memory reservation
- So: Realistic resource request give best results
 - Not always easy to know beforehand
 - Usually best to try with smaller tasks first and check the used resources

Batch Job Script wizard in Scientist's User Interface



taito

Select application...

Form

+

-

?

▼ General

Job Name:

Shell:

Email Address:

▼ Output

Standard Output File Name:

Standard Error File Name:

▼ Computing Resources

Computing Time:

Number of Cores:

Memory Size:

▼ Script Commands

example run commands

srunch ./my_mpi_program

Script Result

+

-

?

```
#!/bin/bash -l
# created: Jan 15, 2015 12:55 PM
# author: saren
#SBATCH -J test1
#SBATCH -o test_out-%j
#SBATCH -e test_err-%j
#SBATCH -n 8
#SBATCH -t 02:00:00
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=END
#SBATCH --mail-user=ari-matti.saren@csc.fi

# commands to manage the batch script
# submission command
# sbatch [script-file]
# status command
# squeue -u saren
# termination command
# scancel [jobid]

# For more information
# man sbatch
# more examples in Taito guide in
# http://research.csc.fi/taito-user-guide

# example run commands
srunch ./my_mpi_program

# This script will print some usage statistics to the
# end of file: test_out-%j
# Use that to improve your resource request estimate
# on later jobs.
used_slurm_resources.bash
```

Save

Reset

Batch Job Script wizard in Scientist's User Interface



taito

Select application...

Form

Script Result

General

Job Name: test1

Shell: /bin/bash

Email Address: ari-matti.saren@csc.fi

Output

Standard Output File Name: test_out-%j

Standard Error File Name: test_err-%j

Computing Resources

Computing Time: 2:00:00

Number of Cores: 8

Memory Size: 4000

Script Commands

example run commands
srun ./my_mpi_program

#!/bin/bash -l
created: Jan 15, 2015 12:55 PM
author: saren
#SBATCH -J test1
#SBATCH -o test_out-%j
#SBATCH -e test_err-%j
#SBATCH -n 8
#SBATCH --mem-per-cpu=4000
#SBATCH --mail-type=END
#SBATCH --mail-user=ari-matti.saren@csc.fi

commands to manage the batch script
submission command
sbatch [script-file]
status command
squeue -u saren
termination command
scancel [jobid]

For more information
man sbatch
more examples in Taito guide in
//research.csc.fi/taito-user-guide

This script will print some usage statistics to the
end of file: test_out-%j
Use that to improve your resource request estimate
on later jobs.
used_slurm_resources.bash

Save

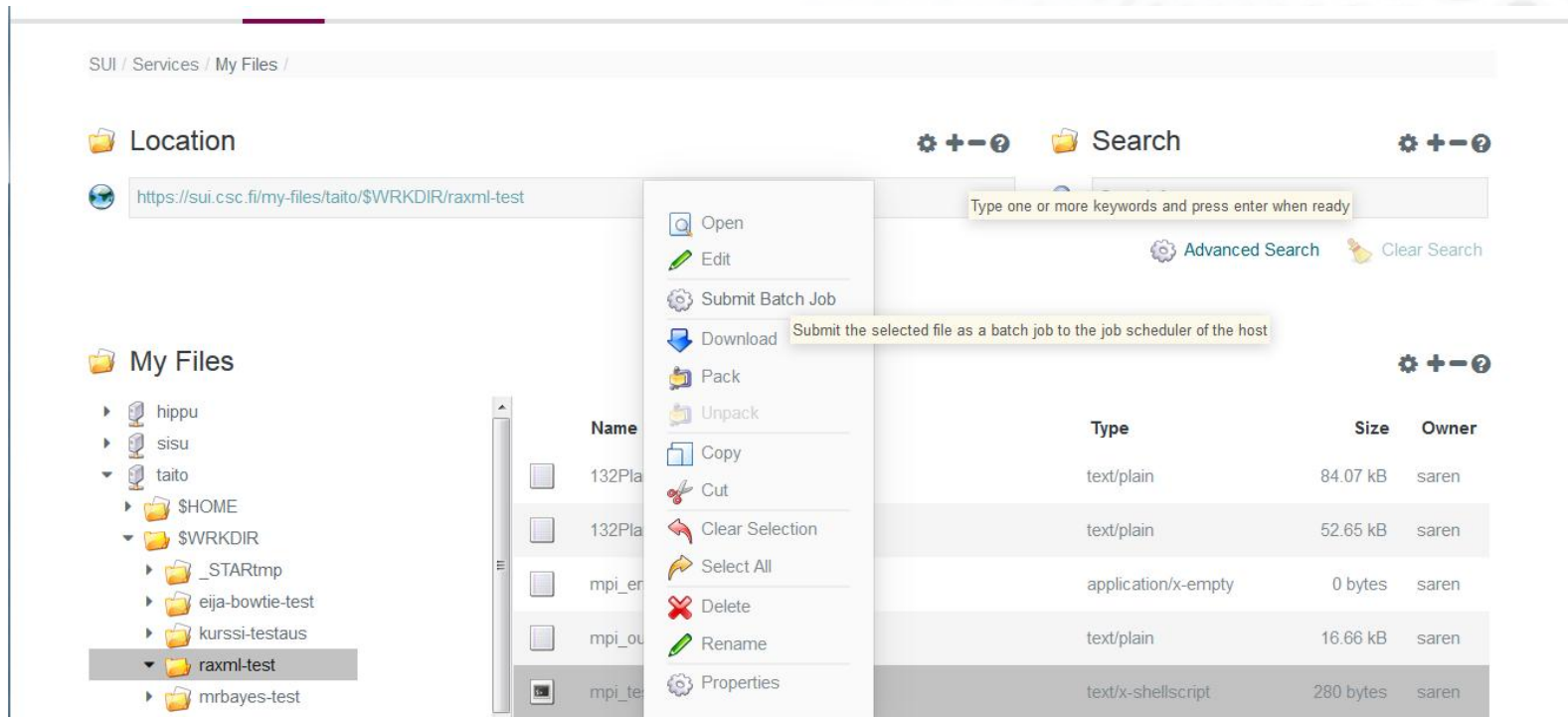
Reset

Computing time must be in format: hh:mm:ss
Supply computing time for a job in hh:mm:ss format. Accurate estimation for
computing time will improve turnover time for the job

Submitting a Batch Job Scientist's User Interface



- Go to My Files
- Select a file
- From the pop-up menu select "Submit batch job"



The screenshot displays the CSC Scientist's User Interface. The top navigation bar shows the path "SUI / Services / My Files /". Below this, the "Location" section shows the URL "https://sui.csc.fi/my-files/taito/\$WRKDIR/raxml-test". The "My Files" section on the left lists several directories, including "hippu", "sisu", "taito", "\$HOME", "\$WRKDIR", "_STARTmp", "eija-bowtie-test", "kurssi-testaus", "raxml-test" (which is selected), and "mrbayes-test". A context menu is open over the "raxml-test" directory, showing options: "Open", "Edit", "Submit Batch Job", "Download", "Pack", "Unpack", "Copy", "Cut", "Clear Selection", "Select All", "Delete", "Rename", and "Properties". A tooltip for the "Submit Batch Job" option reads: "Submit the selected file as a batch job to the job scheduler of the host". The right side of the interface shows a search bar and a table of files.

Type	Size	Owner
text/plain	84.07 kB	saren
text/plain	52.65 kB	saren
application/x-empty	0 bytes	saren
text/plain	16.66 kB	saren
text/x-shellscript	280 bytes	saren



Example serial batch job script on Taito:

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK chr_18 reads.fq > out.sam
```



`#!/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
- Everything starting with "# " is considered a comment
- Everything else is executed as a command

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```



#SBATCH -J bowtie2

- Sets the name of the job
- Job names can be used to manage jobs, but unlike jobids they are not necessarily unique, so care should be taken
 - E.g.
scancel -n bowtie2
- When listing jobs e.g. with `squeue`, only 14 first characters of job name are displayed.

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```




```
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
```

- Option `-o` sets the name of the file where the standard output (stdout) is written
- Option `-e` sets the name of the file where possible error messages (stderr) are written
- When running the program interactively these would be written to the command prompt
- What gets written to stdout and stderr 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 bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```



```
#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 might spend longer in the queue
- Limit for jobs is 3d (72h)
 - if you require longer time, you can specify "longrun" queue (limit 14d)
 - In the longrun queue your job size is limited to one node

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```



```
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
```

- In this case we are running a shared memory program. It must run inside one node, so we specify:

- 1 task (-n)
- 1 node (--nodes)
- 6 cores (--cpus-per-task)

- For a MPI program we would not need to run inside one node, so we might specify simply something like:

```
#SBATCH -n 36
```

- Check software documentation

- Many bioinformatics software can not utilize more than one core
- Some can use threads and run as a shared memory job
- Only very few utilize MPI

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```



#SBATCH --mem=6000

- The amount of memory reserved for the job in MB
 - 1000 MB = 1 GB
- `--mem` should be used for shared memory (OpenMP) jobs
- `--mem-per-cpu` must be used for MPI jobs
 - Example: Specifying `--n 8` and `--mem-per-cpu=1000` reserves 8 GB memory (8 cores x 1 GB)
- Keep in mind the specifications for the nodes. Jobs with impossible requests are rejected
- If you reserve too little memory the job will use swap disk and become very slow
- If you reserve too much memory your job will spend much longer in queue

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=6000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```



#SBATCH -p serial

- The queue (partition) the job should be submitted to
- You can check the available queues with command
`sinfo -l`
- Available queues in Taito:

Queue	Max cores	Max time	Max memory
serial (default)	16 (1 node)	3d	256 GB
parallel	448 (28 nodes)	3d	256 GB
longrun	16 (one node)	14d	256 GB
hugemem	32 (one node)	7d	1,5 TB
test	32 (2 nodes)	30 min	64 GB

```
#!/bin/bash -l
#SBATCH -J bowtie2
#SBATCH -o output_%j.txt
#SBATCH -e errors_%j.txt
#SBATCH -t 02:00:00
#SBATCH -n 1
#SBATCH --nodes=1
#SBATCH --cpus-per-task=6
#SBATCH --mem=1000
#SBATCH -p serial
#

module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK
chr_18 reads.fq > out.sam
```

Choosing processor architecture

For Haswell:

```
#SBATCH --constraint=hsx
```

For Sandy Bridge

```
#SBATCH --constraint=sbx
```

- Necessary if code has been compiled with processor-specific optimizations
- Often not necessary
- Check software web pages



```
module load biokit
bowtie2-build chr_18.fa chr_18
bowtie2-align -p $SLURM_CPUS_PER_TASK chr_18 reads.fq > out.sam
```

- 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
- Command syntax depends on the software
 - It's not enough to reserve the cores: Also remember to tell the program to use them!
 - See application documentation for correct syntax
 - You can use system variable `$SLURM_CPUS_PER_TASK`
- MPI programs must be run through `srun`. Depending on the software you may need to specify some additional options
 - See application documentation for each software
 - For example:

```
srun raxmlHPC-MPI -N 100 -n test1 -s cox1.phy -m GTRGAMMAI
```

Array jobs

- Best suited for running the same analysis for large number of files
- Defined by adding `--array` option to batch job script
 - Can be defined as a range or a list. For ranges step size can be defined

```
#SBATCH --array=1-50
#SBATCH --array=1,2,10
#SBATCH --array=1-100:20
```
- When run, variable **`$SLURM_ARRAY_TASK_ID`** will be replaced with the current array job index
 - Note that the range of the **`$SLURM_ARRAY_TASK_ID`** variable is limited between 0 and 896
- Note that the batch job script is executed for each iteration, so things that should be done only once should not be included in the script



Simple array job example

```
#!/bin/bash
#SBATCH -J array_job
#SBATCH -o array_job_out_%A_%a.txt
#SBATCH -e array_job_err_%A_%a.txt
#SBATCH -t 02:00:00
#SBATCH --mem=4000
#SBATCH --array=1-50
#SBATCH -n 1

# run the analysis command
my_prog data_${SLURM_ARRAY_TASK_ID}.inp data_${SLURM_ARRAY_TASK_ID}.out
```

In this example the actual command run at each iteration will be:

```
myprog data_1.inp data_1.out
myprog data_2.inp data_2.out
..
myprog data_50.inp data_50.out
```

Using a list of file names in an array job



- Often it is easiest to use a list of input filenames
- You can use a combination of sed and the **\$SLURM_ARRAY_TASK_ID** variable
 - To create a list of filenames

```
ls data_*.inp > namelist
```
 - To print a single row in a file by row number:

```
sed -n "row_number"p inputfile
```
 - Example commands in batch job script

```
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)
my_prog ${name} ${name}.out
```

Example batch job script using a list of file names in an array job



```
#!/bin/bash -l
#SBATCH -J array_job
#SBATCH -o array_job_out_%j.txt
#SBATCH -e array_job_err_%j.txt
#SBATCH -t 02:00:00
#SBATCH --mem=4000
#SBATCH --array=1-50
#SBATCH -n 1

# set input file to be processed
name=$(sed -n ${SLURM_ARRAY_TASK_ID}p namelist)
# run the analysis command
my_prog $name $name.out
```

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=MB`
`-p`

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
 Specify queue (partition) to be used. In Taito the
 available queues are: serial, parallel, hsw_par,
 longrun, test and hugemem.

SLURM: Managing batch jobs in Taito

Submitting and cancelling jobs

- The script file is submitted with command
`sbatch batch_job.file`
- sbatch options 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></code>	(shows status of a single job)

- 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=2013-02-27T19:46:44 EndTime=Unknown
```

Job logs

- 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.

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


Available queues

- You can check available queues on each machine with command:

```
sinfo -l
```

PARTITION	AVAIL	TIMELIMIT	JOB_SIZE	ROOT	SHARE	GROUPS	NODES	STATE	NODELIST
serial*	up	3-00:00:00	1	no	YES:4	all	525	mixed	c[5-497,500-505,508,510-516,518-528,570-576]
parallel	up	3-00:00:00	1-28	no	NO	all	525	mixed	c[5-497,500-505,508,510-516,518-528,570-576]
longrun	up	14-00:00:00	1	no	YES:4	all	525	mixed	c[5-497,500-505,508,510-516,518-528,570-576]
test	up	30:00	1-2	no	YES:4	all	4	idle	c[1-4]
hugemem	up	7-00:00:00	1	no	YES:4	all	2	mixed	c[577-578]

Available nodes

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

Scheduling pool data:

Pool	Memory	Cpus	Total	Usable	Free	Other Traits
serial*	258000Mb	16	14	14	0	bigmem
serial*	64300Mb	16	500	499	88	
parallel	258000Mb	16	14	14	0	bigmem
parallel	64300Mb	16	500	499	88	
hsw_par	128600Mb	24	397	397	304	hsw
hsw_par	258000Mb	24	10	10	6	hsw
longrun	64300Mb	16	500	499	88	
longrun	258000Mb	16	8	8	0	bigmem
test	64300Mb	16	4	4	4	
hugemem	1551000Mb	32	2	2	1	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.



pouta.csc.fi cloud service

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

pouta.csc.fi cloud service

- Infrastructure as a Service (IaaS) a type of cloud computing service
- Users set up and run virtual machines at the servers of CSC (Taito)
- Motivation: The user does not need to buy hardware, network it and install operating systems, as this has already been handled by the cloud administrators
- Ready made virtual images available for CentOS, ScientificLinux and Ubuntu.
- Independent from the CSC environment (no direct connection to CSC disk environment and software selection).
- Possible solution for cases where the normal servers of CSC can't be used:(very long run times, unusual operating system or software selection.)

pouta.csc.fi usage

- Send a request for Pouta access with the MyCloud Projects tool in Scientist's User Interface. (<https://research.csc.fi/pouta-application>)

- Once you have the access, log in to Pouta-portal:

<https://pouta.csc.fi>

- Set up and launch a virtual machine according to the instructions in the Pouta user guide:

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

- Login to the virtual machine with **ssh** and start using your virtual server.

	Traditional HPC environment	Cloud environment virtual machine
Operating system	Same for all: CSC's cluster OS	Chosen by the user
Software installation	Done by cluster administrators, customers can only install software to their own directories, no administrative rights	Installed by the user, the user has admin rights
User accounts	Managed by CSC's user administrator	Managed by the user
Security e.g. software patches	CSC administrators manage the common software and the OS	User has more responsibility: e.g. patching of running machines
Running jobs	Jobs need to be sent via the cluster's Batch Scheduling System	The user is free to use or not use a batch job system
Environment changes	Changes to software happen.	The user can decide on versions.
Snapshot of the environment	Not possible	Can save as a Virtual Machine image

Pouta virtual machine sizes

	Cores	Memory	Disk (root)	Disk (ephemeral)	Disk (total)	Memory/core	Billing Units/h
tiny	1	1 GB	10 GB	110 GB	120 GB	1	2
mini	1	3,5 GB	10 GB	110 GB	120 GB	3	2
small	4	15 GB	10 GB	220 GB	230 GB	4	8
medium	8	30 GB	10 GB	440 GB	450 GB	4	16
large	12	45 GB	10 GB	660 GB	670 GB	4	24
bigroot	16	60 GB	80 GB	500 GB	580 GB	4	32
fullnode	16	60 GB	10 GB	900 GB	910 GB	4	32