



# University of Zurich<sup>UZH</sup>

## High Performance Computing | Lecture 1

Douglas Potter  
Jozef Bucko  
Noah Kubli

# Organization

- 14 Lectures and 14 Exercise Classes
  - Wednesday: Concepts and overview
  - Friday: Hands-on exercises
  - 6 ECTS = 8+ hours per week
- Microsoft Teams: ESC401 team
  - <https://teams.Microsoft.com/> (or application)
- Laptop required
  - CSCS Account – follow email instructions
  - Need an “ssh” terminal
    - Linux & OS X: already there
    - Windows: putty or WSL2
  - Part of the first exercise sheet ☺

# Prerequisites and Grading

- No prerequisites, but,
  - Having experience with C would help
  - Having written scripts (Python, BASH, etc.)
- Goals
  - Understand High Performance Computing
  - Understand how to make things faster
  - Use HPC resources
  - Change a serial code to a parallel code
- Grading
  - Must get 60% on Exercises
  - Examination on January 17<sup>th</sup> (tentative)
    - Good results on the exercises can improve grade

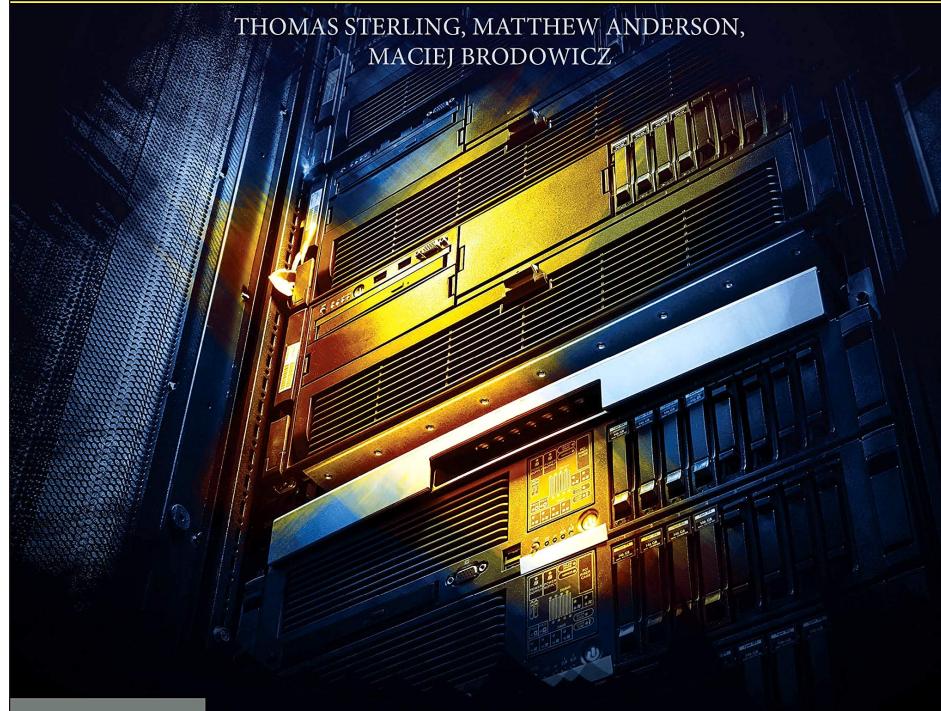
# Optional

- Not required for the course
- Much greater detail on HPC topics
- Available in the Library
- ... and on Kindle
  
- Other resources listed in Teams
- ... including these slides

# HIGH PERFORMANCE COMPUTING

MODERN SYSTEMS AND PRACTICES

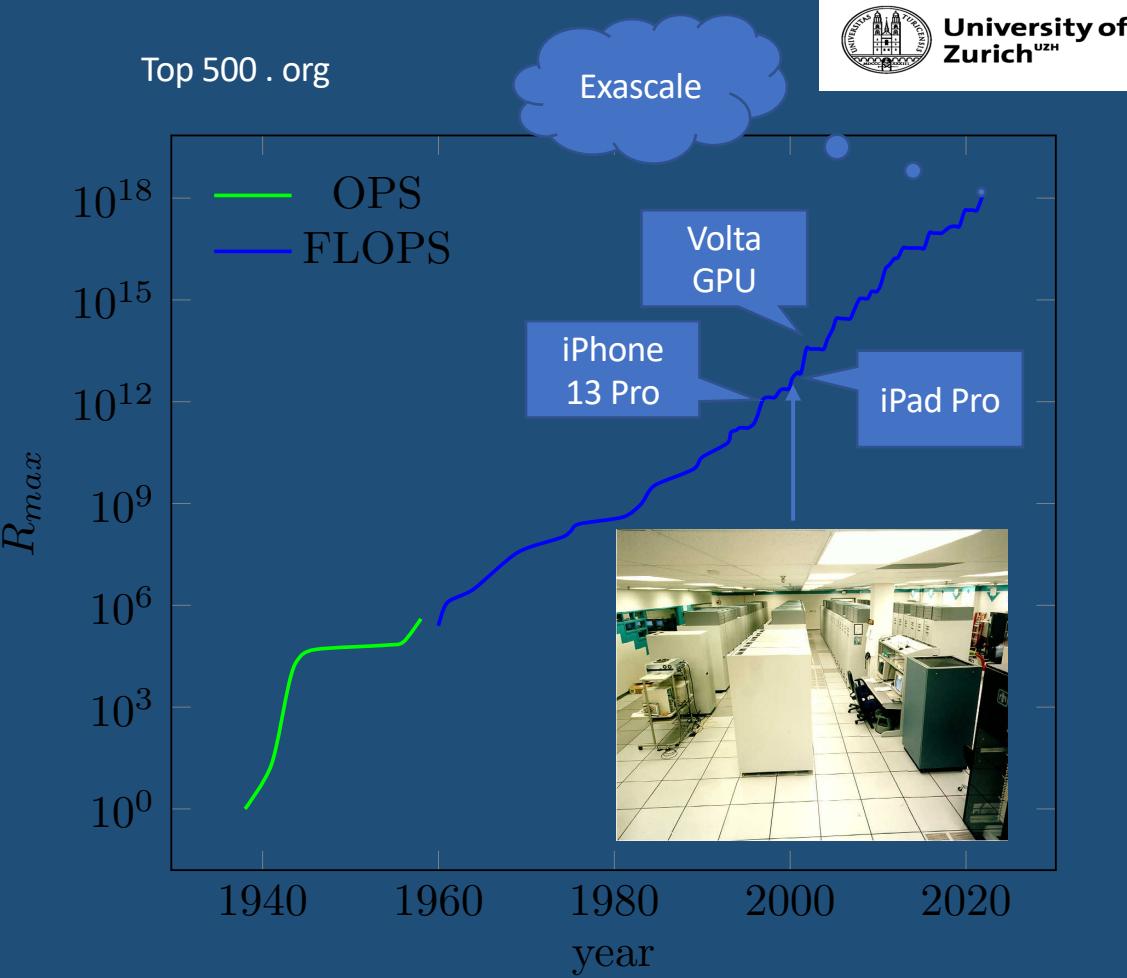
THOMAS STERLING, MATTHEW ANDERSON,  
MACIEJ BRODOWICZ



FOREWORD BY C. GORDON BELL

# What is High Performance Computing?

- Time to solution
- Normal progression
  - Laptop
  - Workstation
  - Server
  - Special Hardware
  - Supercomputer
- More and more?
  - FLOPS generally
  - I/O maybe
  - Memory capacity
  - Memory speed



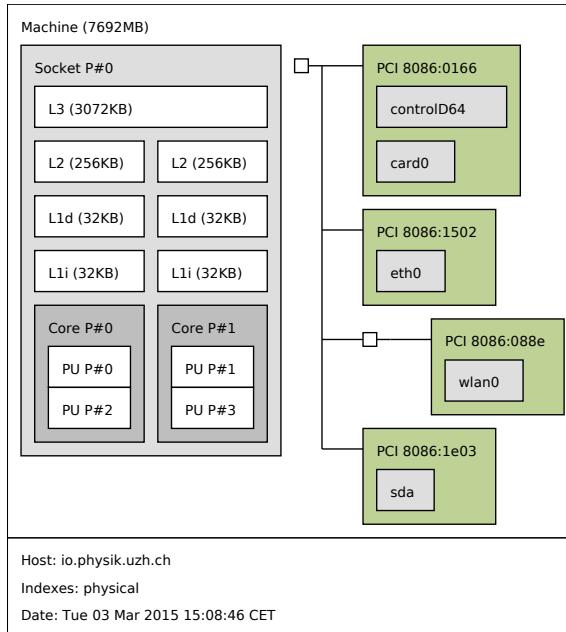
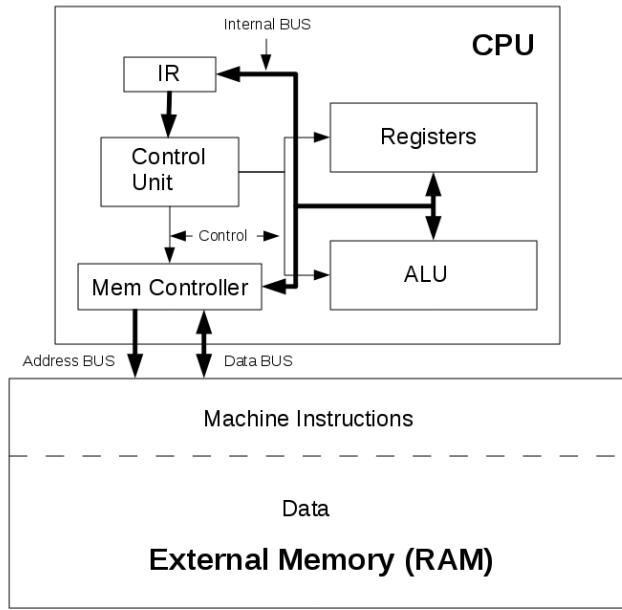
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	<b>Sunway TaihuLight</b> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	<b>Tianhe-2A</b> - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	<b>Piz Daint</b> - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray/HPE Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	<b>Gyoukou</b> - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	28,192.0	1,350
5	<b>Titan</b> - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray/HPE DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
6	<b>Sequoia</b> - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890

Top500.org

June 2022

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	<b>Frontier</b> - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	<b>Supercomputer Fugaku</b> - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	<b>LUMI</b> - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	214.35	2,942
4	<b>Summit</b> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
23	<b>Piz Daint</b> - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21.23	27.15	2,384

# What is a computer really?

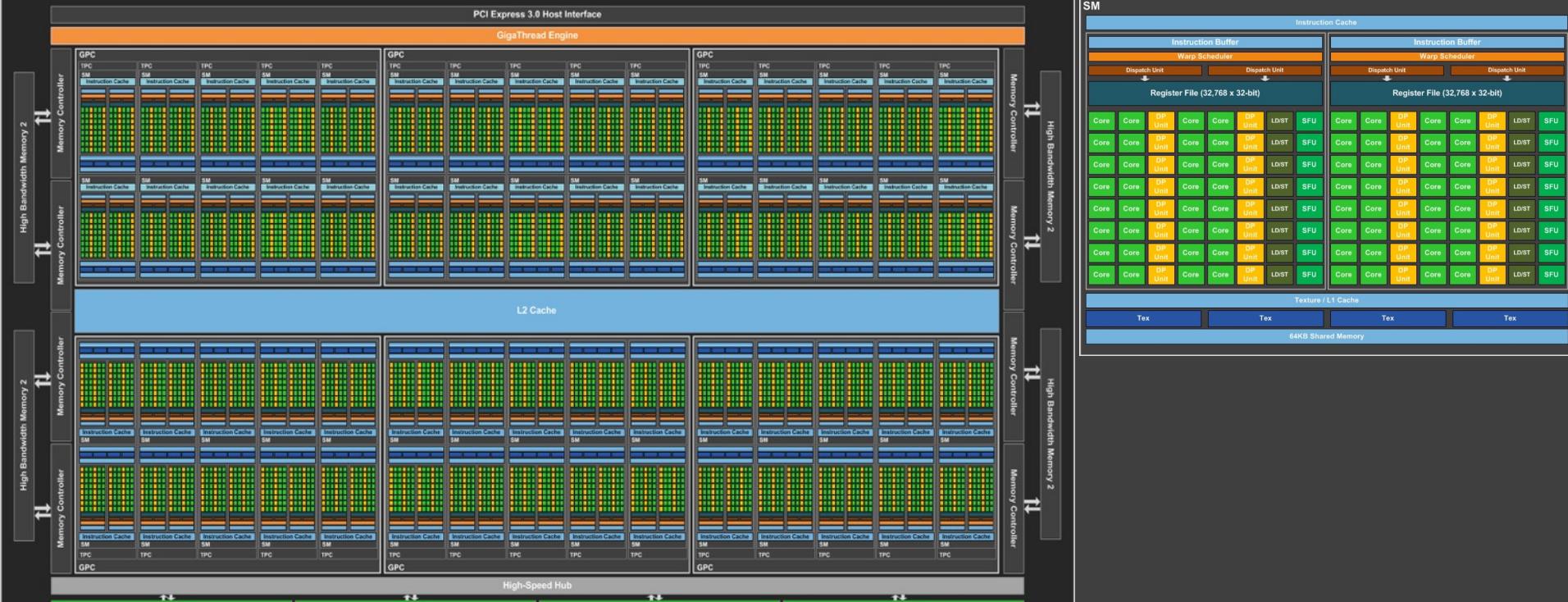


# A 24 core NUMA node



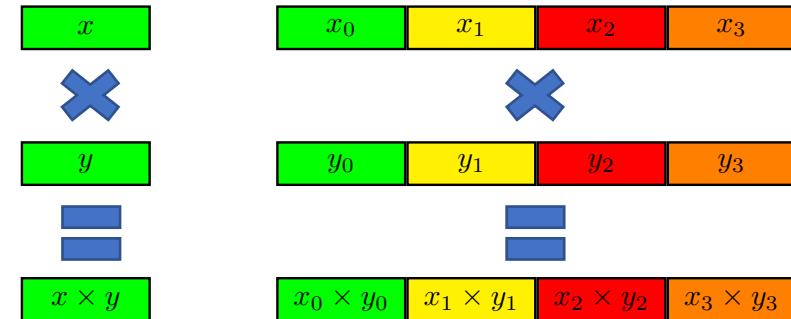
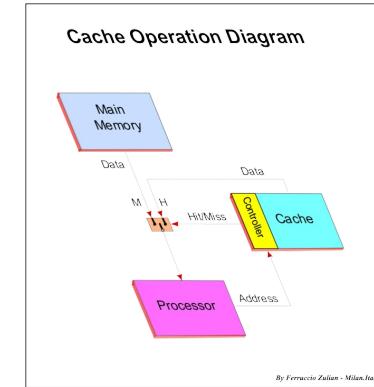
# “Eiger” – 128 cores

# GPU Computing – P100 – 3584 “cores”

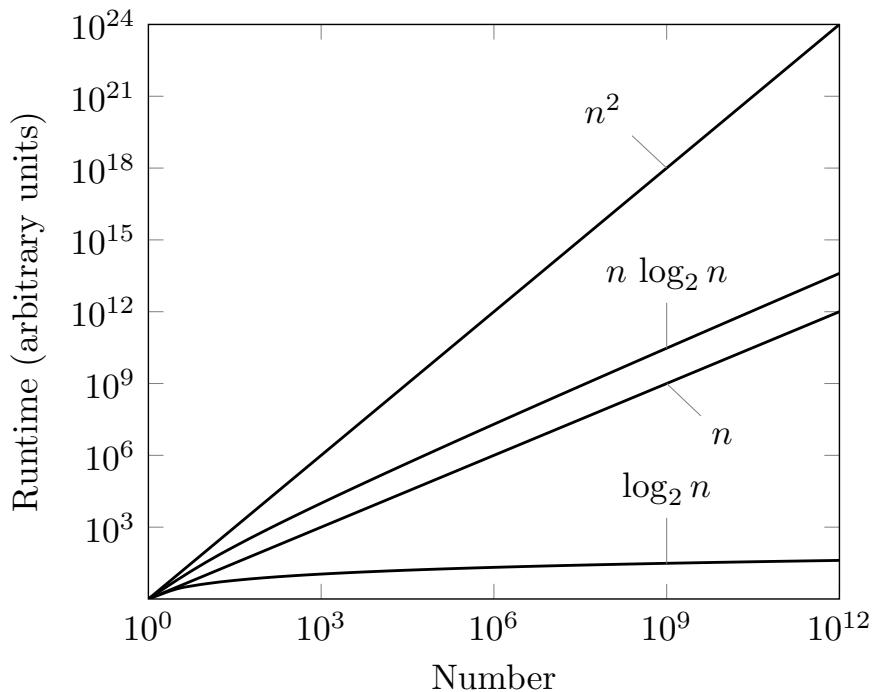


# How do we make “it” faster?

- Increase the clock speed
  - 1 MHz to 4 GHz in around 35 years
  - 12 doublings of performance
- Optimize the hardware
  - Faster memory
  - Cache
  - Cycles per operation
- SIMD: AVX or SSE



# Algorithmic Changes



	Thousand	Million	Billion	Trillion
Log N	10 ns	20 ns	30 ns	40 ns
N	1 us	1 ms	1 s	17 minutes
$N \log N$	10 us	20 ms	30 s	11 hours
$N^2$	1 ms	17 minutes	32 years	32 mega-years

# How do we make “it” faster 2

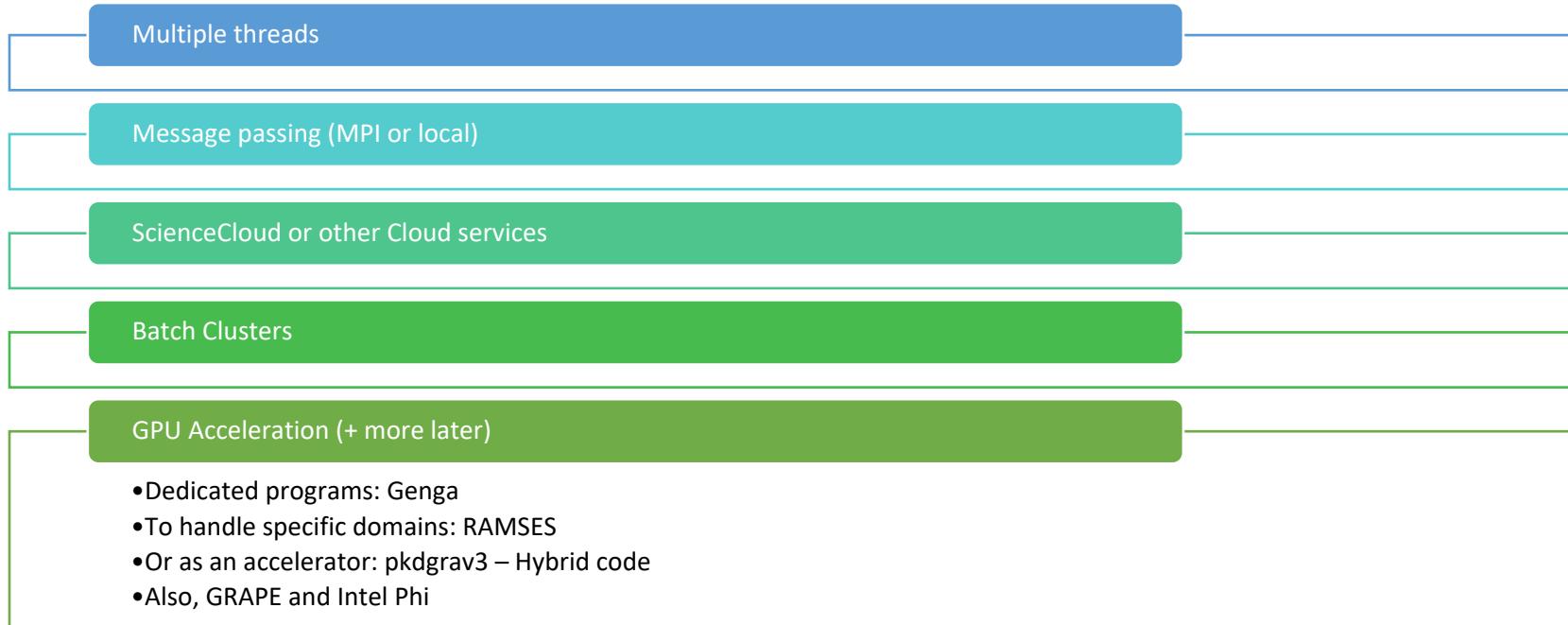
More is  
better?

- Add additional CPUs
- Increase the number of cores
- Add additional nodes

How do we  
use all this?

- Embarrassingly parallel
- Goal: truly parallel

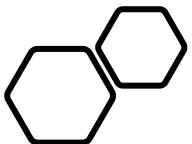
# Working on the “Same Problem”

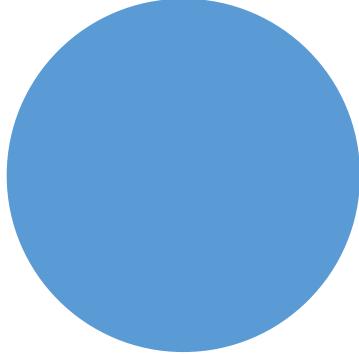




# Super Computers

- A bunch of nodes together
- Special Interconnect (Network)
- Far away from you
- Runs Unix / Linux (usually)
- Non-interactive (usually)





# Connecting to the Supercomputer

High Performance Computing

# Online Documentation

The screenshot shows a web browser window titled "Running jobs" with the URL [https://user.cscs.ch/getting\\_started/running\\_jobs/piz\\_daint/](https://user.cscs.ch/getting_started/running_jobs/piz_daint/). The page is part of the CSCS User Portal, featuring the CSCS logo and ETH Zürich branding. The main content area is titled "Piz Daint". It provides information about accessing the system via ssh from the front end `ela.cscs.ch` as `daint.cscs.ch`, using the SLURM `srun` command for parallel programs, and the `slurm jobscript generator` for creating template scripts. A code block shows a sample SLURM script:

```
#!/bin/bash -l
#SBATCH --job-name=job_name
#SBATCH --time=01:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-core=2
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=2
#SBATCH --partition=normal
```

# Teletype

- Circa 1963
- MODEM
- Paired
- Paper tape
  - Writer
  - Reader



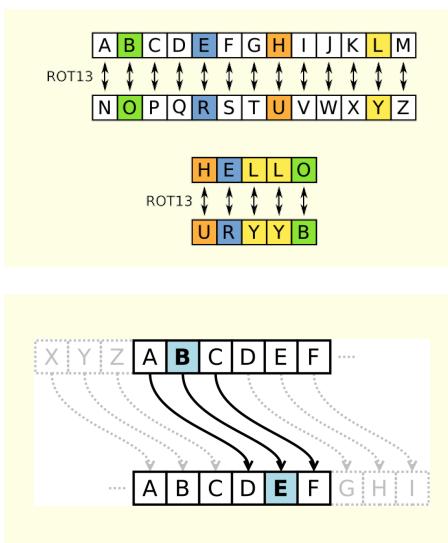
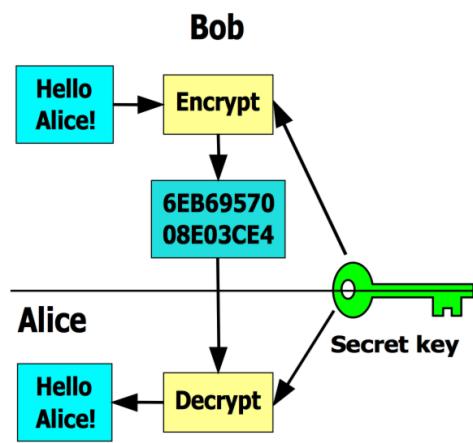
# Terminals

- VT100 (1978)
- Direct connection
- RS-232 originally
- Later networked
- TELNET protocol (TELetype NETwork)
- SSH protocol (Secure SHell)
  - Encryption

High Performance Computing



# Encryption



- First in ca. 1900 BC
- Julius Caesar 100 BC
- More complex keys
- E.g., "One time PAD"
- Relies on sharing "keys"

```

1. dpotter@daint103:~ (ssh)
Last login: Tue Feb 20 17:21:03 on ttys002
dhcp-94-191-$ ssh eld
Warning: No xauth data; using fake authentication data for X11 forwarding.
Last login: Tue Feb 20 17:09:34 2018 from dhcp-94-191.physik.uzh.ch

=====
IMPORTANT REMINDER FOR USERS OF CSCS facilities
help@csccs.ch - http://user.csccs.ch
=====

CSCS next scheduled preventive maintenance is scheduled for:
Wednesday February 21, 2018
Center-Wide 07:00 - 20:00

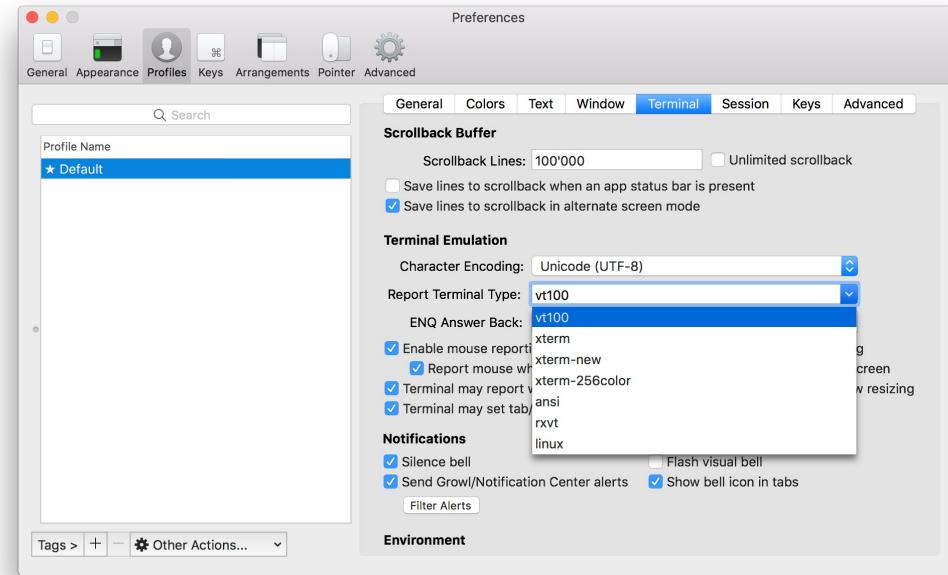
dpotter@ela3:~> ssh daint
Last login: Mon Feb 19 13:48:34 2018 from 148.187.1.7
=====
IMPORTANT REMINDER FOR USERS OF CSCS facilities
help@csccs.ch - +41 91 610 82 10 - http://user.csccs.ch
=====

Please load 'daint-gpu' module for using the GPU/Haswell nodes
or
load 'daint-mc' module for the Multicore/Broadwell nodes

For more info, please refer to the User Portal:
http://user.csccs.ch/getting_started/running_jobs/piz_daint

dpotter@daint103:~>

```

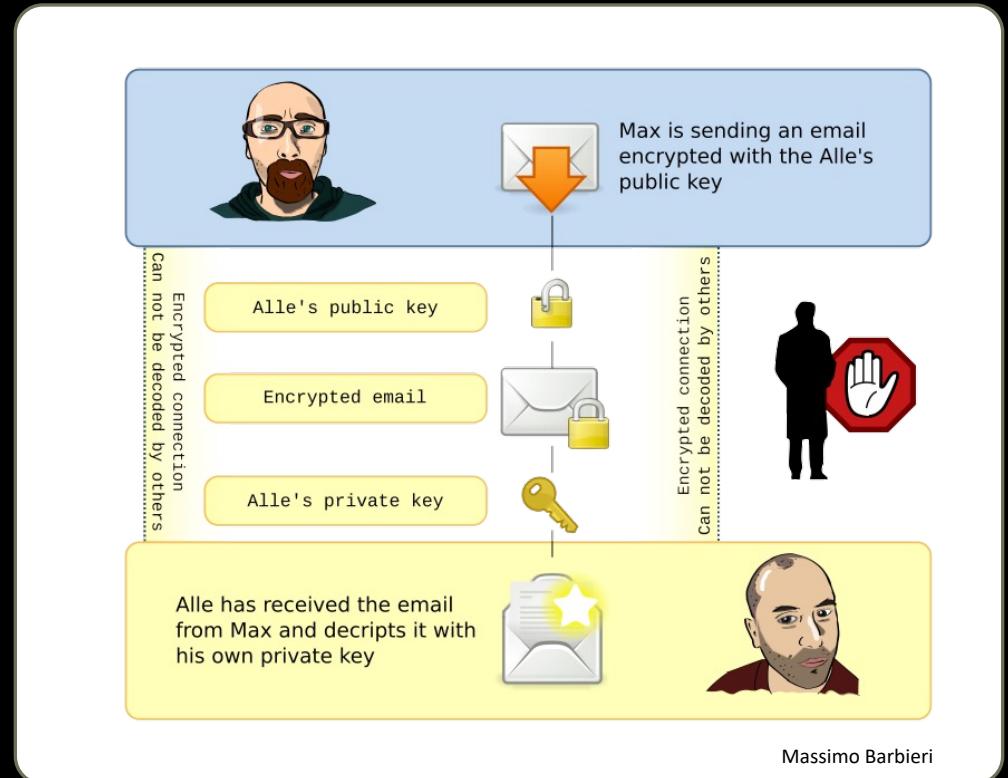


## Secure Shell

- We want to connect securely to remote systems...
- ... but sharing keys in advance is cumbersome.

# Public Key Encryption

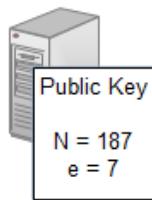
- Enter public keys
- Two keys: public & private
- I “publish” my public key
  - And you publish yours.
  - Private keys we keep secret.
- Encrypt with a public key
  - Can decrypt with private key
  - I can send secret messages!
- Encrypt with a private key
  - Decrypt with public key
  - I can prove who I am!
- The details are more complicated of course.



Massimo Barbieri

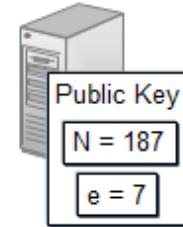
## • PUBLIC KEY

- 2 giant primes,  $p$  and  $q$ 
  - $p = 17, q = 11$
- $p * q = N$ 
  - $N = 187$
- Pick another prime,  $e$ 
  - $e = 7$



## • PRIVATE KEY

- Private key =  $d$
- $e * d = 1 \pmod{(p-1)*(q-1)}$
- $7 * d = 1 \pmod{16 * 10}$
- $7 * d = 1 \pmod{160}$
- $d = 23$



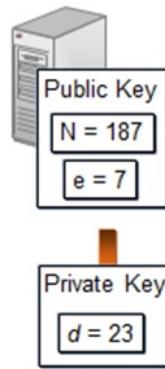
# Factoring Prime Numbers

- Public key encryption depends on factoring primes being hard
- The security of the Internet depends on it
- Quantum computing introduces some concerns

- Let's send "X"

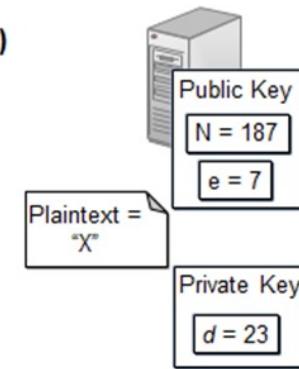
- $X = 88 \text{ ASCII}$
- Encryption:**
  - $\text{Ciphertext} = 88^e \pmod{N}$
  - $\text{Ciphertext} = 88^7 \pmod{187}$
  - $\text{Ciphertext} = 11$

Ciphertext  
C = 11



- Decryption:

- $\text{Plaintext} = C^d \pmod{N}$
- $\text{Plaintext} = 11^{23} \pmod{187}$
- $\text{Plaintext} = 88$
- $\text{ASCII } 88 = "X"$
- $\text{Plaintext} = "X"$



# Factoring Prime Numbers

# SSH Key Types

## RSA

- Very common and supported
- Relies on prime numbers
  - Integer factorization

$$N = p \times q$$

- Find  $p$  and  $q$  given  $N$

## ED25519

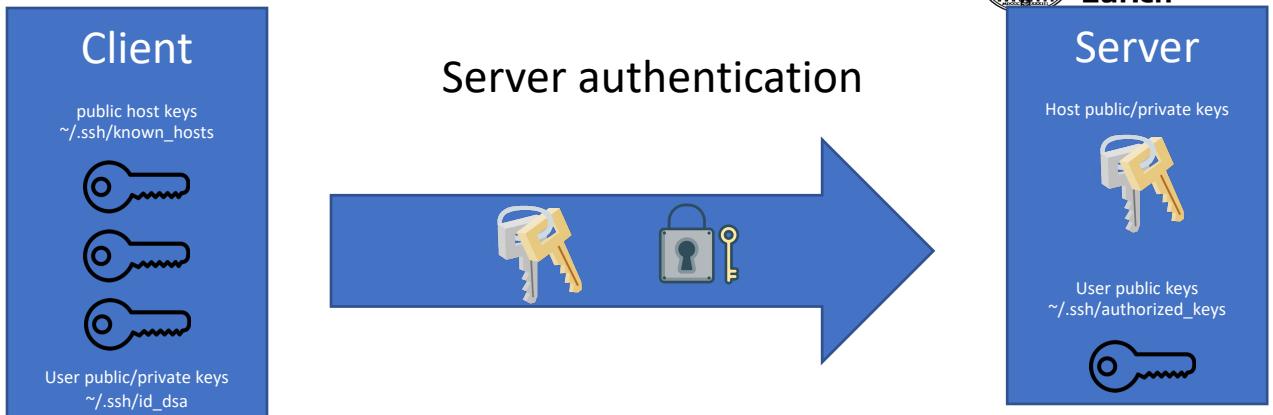
- Support on newer systems; faster
- Elliptical Curve
  - Discrete Logarithm problem

$$x = a^b \bmod n$$

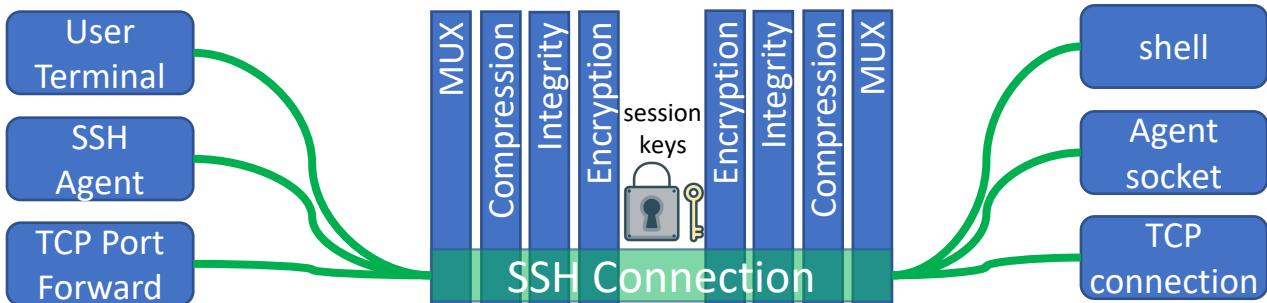
- Solve for  $b$  given  $x, a, n$

# SSH Components

- Servers have public keys too!
- This way we can avoid man-in-the-middle attacks.
- Public host key is saved first time we connect to a host.
  - Thereafter it is verified.
- Session keys are exchanged for faster encryption & decryption.
- Traffic is “tunneled” through our connection.
  - Port forwarding for example.

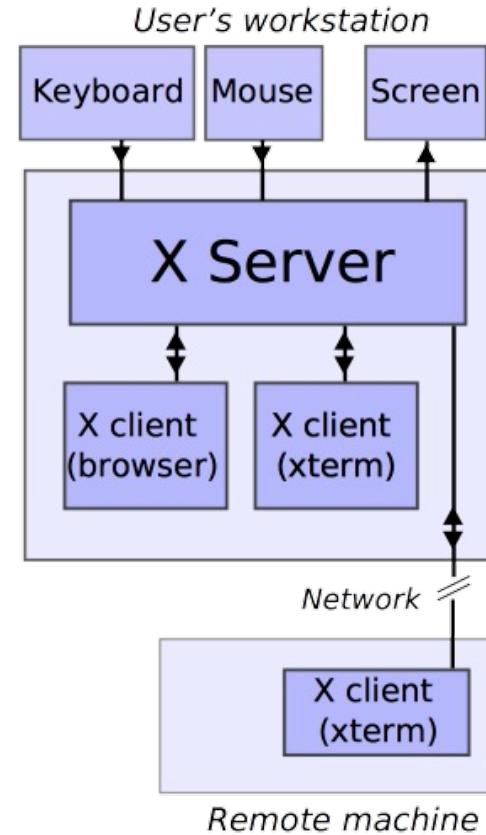


```
dhcp-94-183:~$ ssh dpotter@ela.csics.ch
The authenticity of host 'ela.csics.ch (148.187.1.20)' can't be established.
ED25519 key fingerprint is SHA256:CTt/IUUObWMase08gHXQjysMbc8nBahV+gHigMkvLY.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ela.csics.ch' (ED25519) to the list of known hosts.
```

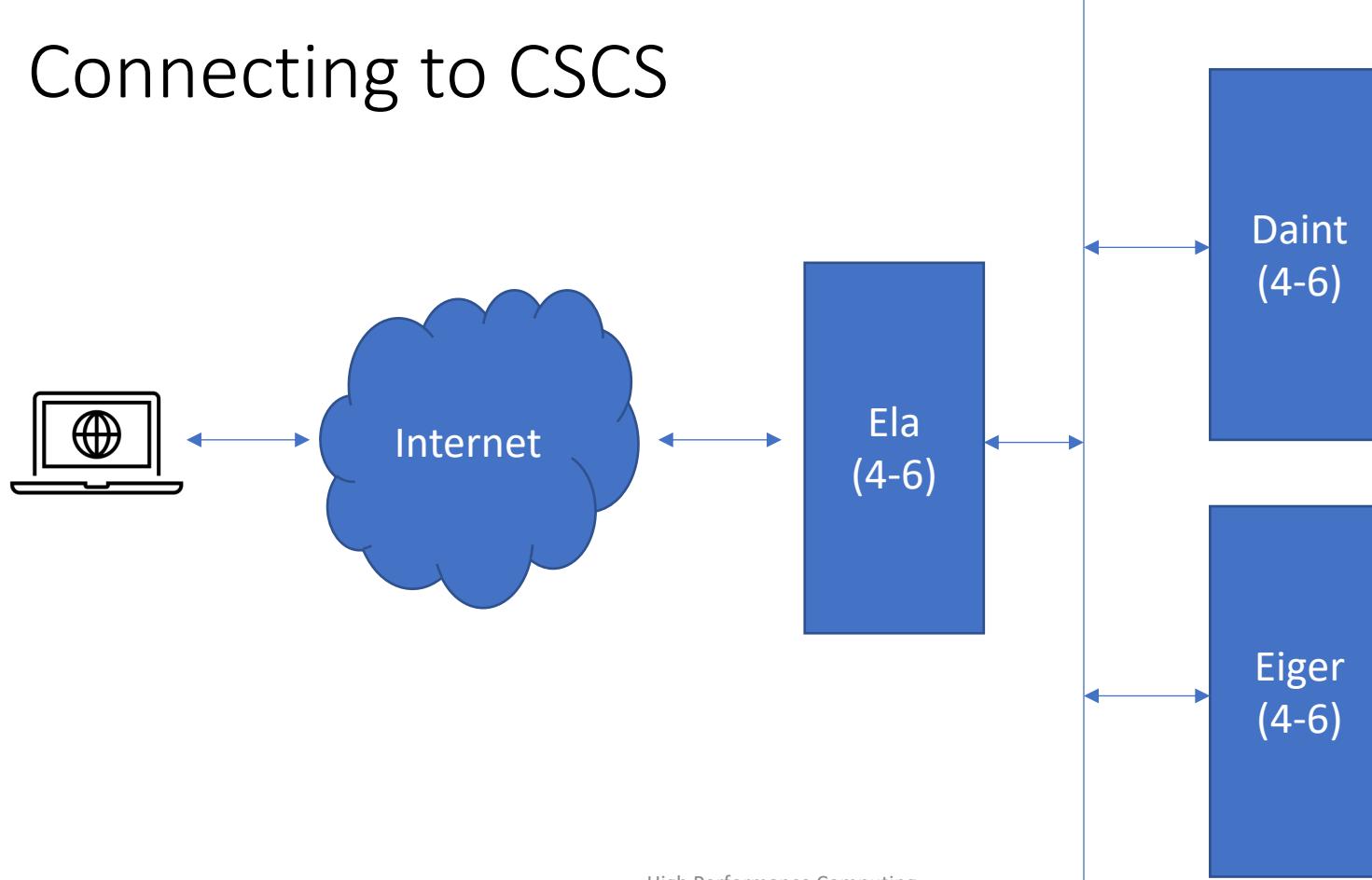


# X11 for Graphical Interfaces

- Just “works” on Linux
- XQuartz on OS X
- VcXsrv on Windows
  - Not needed for WSL2 (Windows 11)
- DISPLAY environment variable
  - export DISPLAY=localhost:0
  - Just for windows probably
- “ssh” must tunnel usually
  - ssh -X hostname
  - ForwardX11 yes (config)



# Connecting to CSCS



# ~/.ssh/config

```
Host ela
  Hostname ela.cscs.ch
  User dpotter
  ForwardX11 yes
  ForwardX11Trusted yes
  ForwardAgent yes

Host eiger
  ProxyJump ela
  User dpotter
  ForwardX11 yes
  ForwardX11Trusted yes
  ForwardAgent yes
```

```
dhcp-94-183:~$ ssh dpotter@ela.cscs.ch
[dpotter@ela3 ~]$  
  
dhcp-94-183:~$ ssh ela
[dpotter@ela2 ~]$ ssh eiger
[eiger] [dpotter@nid001080 ~]$  
  
dhcp-94-183:~$ ssh eiger
[eiger] [dpotter@nid001000 ~]$
```

# Putting it together

SSH directory: `~/.ssh` or `$HOME/.ssh`

“config” file (`man ssh_config`)

“known\_hosts” – prevents “man in the middle”

“authorized\_keys” – keys allowed access

“`id_rsa`” – your private key (default name)

“`id_rsa.pub`” – your public key (default name)

“`ssh-keygen`” – generate a public/private key pair

“`ssh-copy-id`” – convenience program

“`ssh-agent`” and “`ssh-add`”

# Why do we use the Command Line?

- Because we've always done it that way?
- Graphical User Interfaces are slow over distances
  - Web forms are possible (e.g., Jupyter Notebook)
- Need more flexibility
  - Build and run codes (ours and others)
  - Designed workflows.

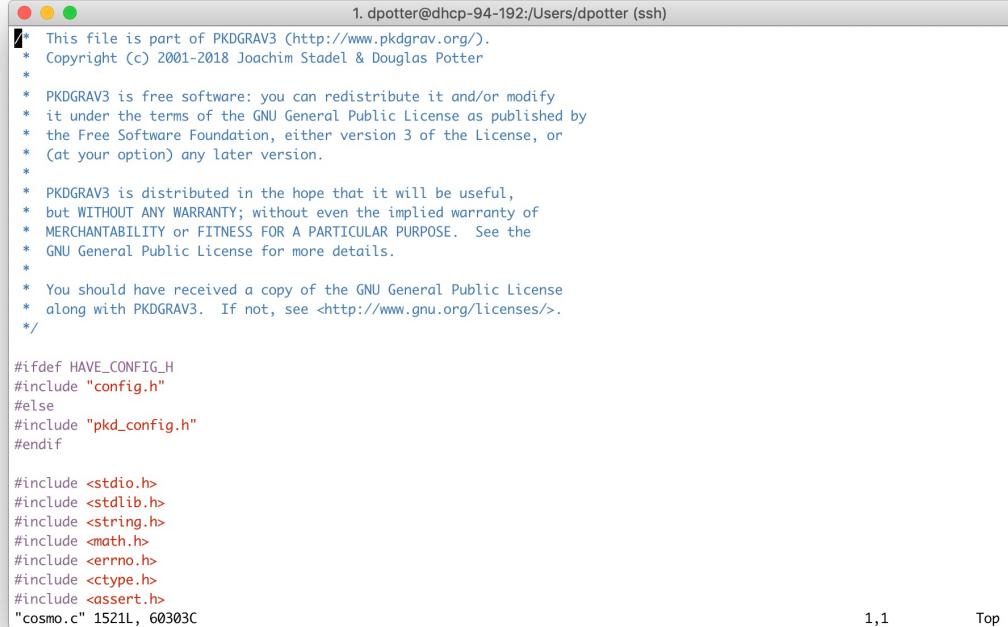
```
FILE *fpLog = NULL;
char achFile[256]; /*DEBUG use MAXPATHLEN here (& elsewhere)? --- DCR*/
double dTime;
double E=0,T=0,U=0,Eth=0,L[3]={0,0,0},F[3]={0,0,0},W=0;
double dMultiEff=0;
long lSec=0;
int i,iStep,iStartStep,iSec=0,iStop=0;
uint64_t nActive;
int bKickOpen=0;
int bDoOpeningKick=0;
int bDoCheckpoint=0;
int bDoOutput=0;
int64_t nRungs[MAX_RUNG+1];
uint8_t uRungMax;
double diStep;
double ddTime;
int bRestore;
printf("%s\n", PACKAGE_STRING );
bRestore = msrInitialize(&msr,mdl,argc,argv);
msr->lStart=time(0);

/*
** Establish safety lock.
*/
if (!msrGetLock(msr)) {
    msrFinish(msr);
    return NULL;
}

/* a USR1 signal indicates that the queue wants us to exit */
#ifndef _MSC_VER
timeGlobalSignalTime = 0;
signal(SIGUSR1,NULL);
signal(SIGUSR1,USR1_handler);
#endif
```

# Text Editors

“Vim”  
(text only)



1. dpotter@dhcp-94-192:~/Users/dpotter (ssh)

```
/* This file is part of PKDGRAV3 (http://www.pkdgrav.org/).
 * Copyright (c) 2001-2018 Joachim Stadel & Douglas Potter
 *
 * PKDGRAV3 is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * PKDGRAV3 is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with PKDGRAV3. If not, see <http://www.gnu.org/licenses/>.
 */

#ifndef HAVE_CONFIG_H
#include "config.h"
#else
#include "pkd_config.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>
#include <ctype.h>
#include <assert.h>
"cosmo.c" 1521L, 60303C
```

1,1 Top

## How To Use SSHFS to Mount Remote File Systems Over SSH

Updated November 9, 2016 1:4m LINUX BASICS

Paul White

### Introduction

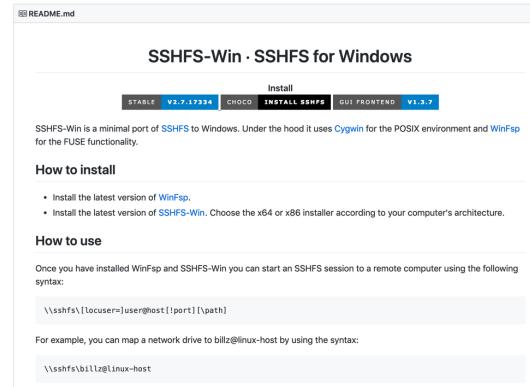
In many cases it can become cumbersome to transfer files to and from a droplet. Imagine a development usage scenario where you are coding apps remotely and find yourself uploading a script repeatedly to your virtual server to test. This can become quite a hassle in a very short period of time. Luckily there is a way to mount your VPS file system to your local computer so you can make changes on the fly and treat your droplet as local storage. In this article, we will show you how to do exactly that.

### Installing SSHFS

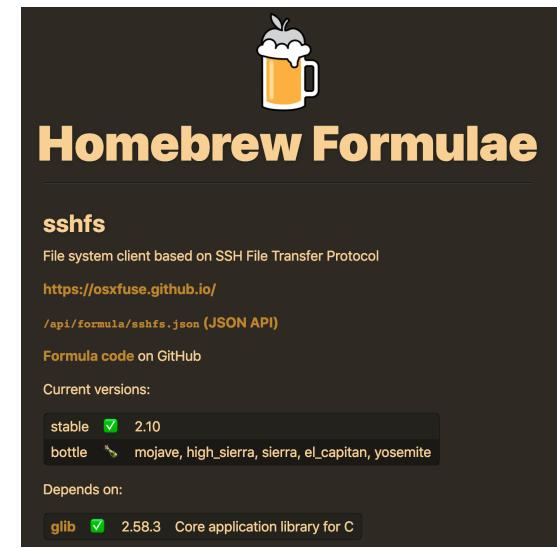
#### On Ubuntu/Debian

SSHFS is Linux based software that needs to be installed on your local computer. On Ubuntu and Debian based systems it can be installed through apt-get.

```
sudo apt-get install sshfs
```



The screenshot shows the GitHub page for the `SSHFS-Win` project. The page title is `SSHFS-Win · SSHFS for Windows`. It features a "Install" button and links for "STABLE V2.7.17334", "CHOCO", "INSTALL SSHFS", and "GUI FRONTEND V1.3.7". Below the header, a note states: "SSHFS-Win is a minimal port of `SSHFS` to Windows. Under the hood it uses `Cygwin` for the POSIX environment and `WinFsp` for the FUSE functionality." A section titled "How to install" contains two bullet points: "Install the latest version of `WinFsp`." and "Install the latest version of `SSHFS-Win`. Choose the x64 or x86 installer according to your computer's architecture." Another section, "How to use", provides syntax examples: "\sshfs\[\!username\]\user@host\[\!port\]\[\!path\]" and "\\sshfs\billz@linux-host".



The screenshot shows the Homebrew Formulae page for `sshfs`. The page title is **Homebrew Formulae**. It includes a "sshfs" icon (a beer mug) and a brief description: "File system client based on SSH File Transfer Protocol". Below this are links to the formula's GitHub repository (<https://osxfuse.github.io/>) and its JSON API ([/api/formula/sshfs.json \(JSON API\)](/api/formula/sshfs.json)). It also links to the formula's code on GitHub. A "Current versions:" section shows "stable" (version 2.10, checked) and "bottle" (multiple macOS versions: mojave, high\_sierra, sierra, el\_capitan, yosemite). A "Depends on:" section lists "glib" (version 2.58.3, checked).

# Remote Access to files



ela

parse.py    X    ela    X    master.py    X    master.c    X    master.h

```
1 {  
2     // The tab key will cycle through the s  
3     // Visit http://wbond.net/sublime_package  
4  
5     // sftp, ftp or ftps  
6     "type": "sftp",  
7  
8     "sync_down_on_open": true,  
9     "sync_same_age": true,  
10  
11    "host": "ela",  
12    "user": "dpotter",  
13    //"password": "password",  
14    //"port": "22",  
15  
16    "remote_path": "/users/dpotter",  
17    //"file_permissions": "664",  
18    //"dir_permissions": "775",  
19  
20    //"extra_list_connections": 0,  
21  
22    "connect_timeout": 30,  
23    //"keepalive": 120,  
24    //"ftp_passive_mode": true,  
25    //"ftp_ehev_passive_host": false
```

fio.c  
fio.h  
fof.c  
fof.h  
grav.h  
grav2.c  
gridinfo.cxx  
gridinfo.hpp  
group.c  
group.h  
groupstats.c  
groupstats.h

Built into  
SublimeText

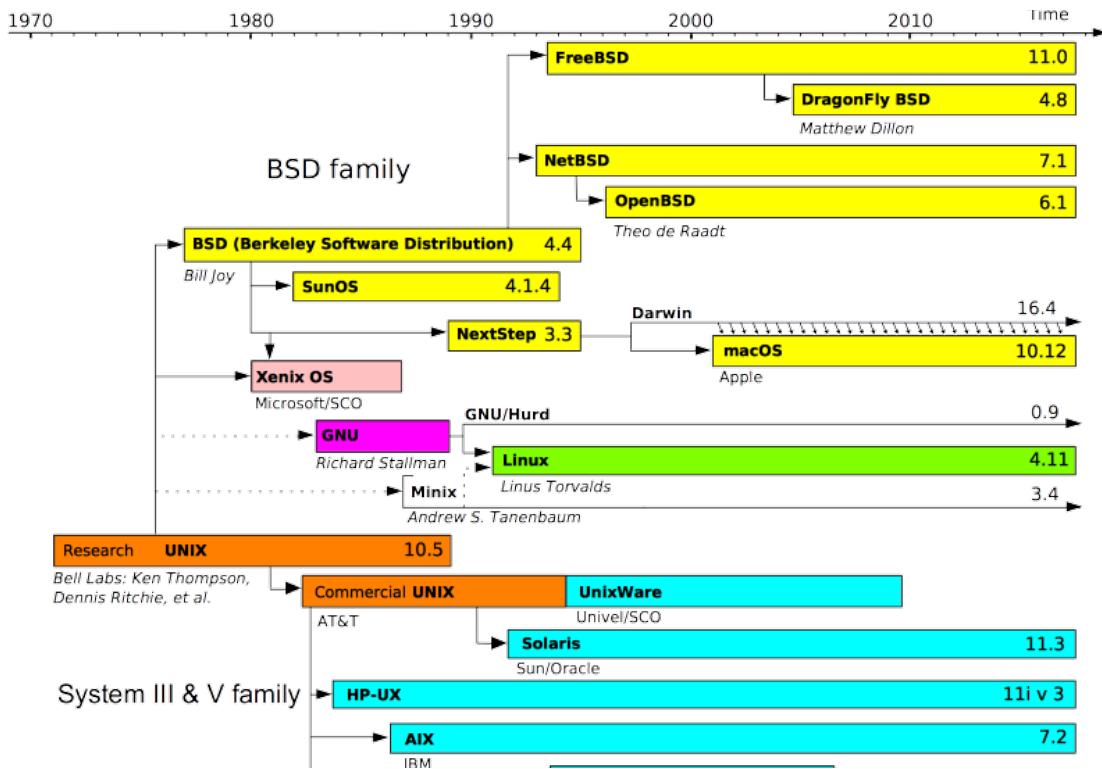
---



# What is Linux / Unix

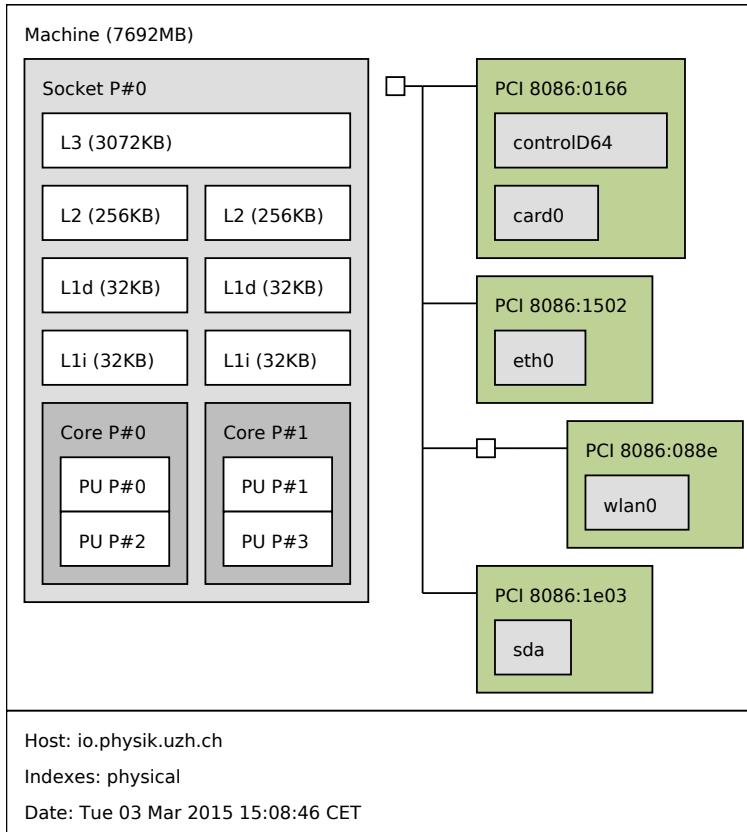
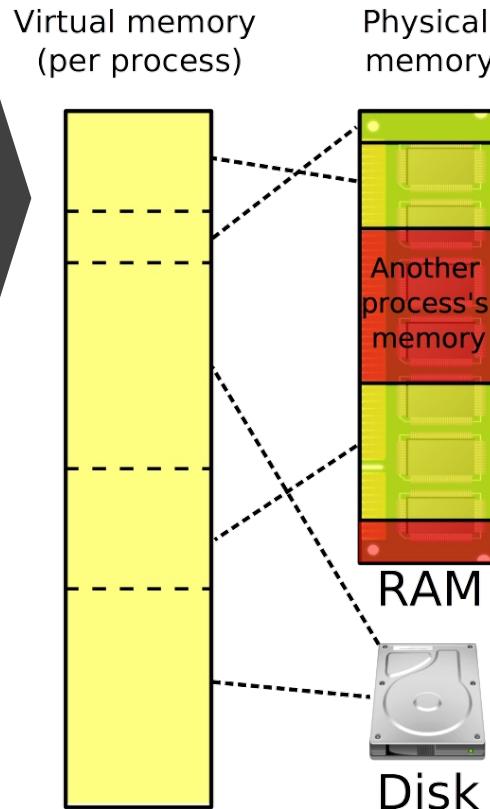
The HPC Operating System

# Unix History



# Threads and Processes?

- Share memory?
- Share CPU?
- Registers?



# “top”

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
17005	bp000253	20	0	181664	5192	1432	S	5.3	0.0	4:14.73 sshd: <a href="#">bp000253@notty</a>
17006	bp000253	20	0	137856	2128	1204	R	2.3	0.0	3:42.73 rsync --server --sender -vlogDtpre.iLs . /sto-
2194	root	20	0	1065348	3644	788	S	0.7	0.0	174:18.84 /opt/IBM/zimon/sbin/pmsensors -C /opt/IBM/zim-
14541	root	0	-20	5886420	1.5g	245092	S	0.7	9.8	1418:31 /usr/lpp/mmfs/bin/mmfsd
74403	dpotter	20	0	172752	3272	2108	R	0.7	0.0	0:00.19 top -c
74421	root	20	0	112864	4280	3256	S	0.7	0.0	0:00.02 sshd: [accepted]
870	root	20	0	47732	10352	9948	S	0.3	0.1	54:34.11 /usr/lib/systemd/systemd-journald
1498	root	16	-4	55520	492	372	S	0.3	0.0	9:21.19 /sbin/auditd
1568	root	20	0	300824	1928	1236	S	0.3	0.0	40:46.86 /usr/bin/vmtoolsd
1569	root	20	0	90500	288	284	S	0.3	0.0	31:32.09 /sbin/rngd -f
2163	root	20	0	2008520	50284	5468	S	0.3	0.3	1199:18 /usr/share/metricbeat/bin/metricbeat -c /etc/+
2188	root	20	0	3266008	129848	6244	S	0.3	0.8	126:12.44 /usr/bin/python2 /usr/lpp/mmfs/bin/mmsysmon.py
82198	root	20	0	112864	1612	592	S	0.3	0.0	8:09.58 /usr/sbin/sshd -D
93225	astagni	20	0	179760	3260	1444	S	0.3	0.0	2:45.77 sshd: astagni
120562	sobbeck	20	0	179416	2656	1212	S	0.3	0.0	0:11.09 sshd: sobbeck@pts/36
1	root	20	0	194616	5816	2536	S	0.0	0.0	101:01.04 /usr/lib/systemd/systemd --switched-root --sy-
2	root	20	0	0	0	0	S	0.0	0.0	0:04.52 [kthreadd]
3	root	20	0	0	0	0	S	0.0	0.0	6:34.95 [ksoftirqd/0]
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 [kworker/0:0H]
7	root	rt	0	0	0	0	S	0.0	0.0	0:14.60 [migration/0]
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00 [rcu_bh]
9	root	20	0	0	0	0	S	0.0	0.0	30:06.72 [rcu_sched]
10	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 [rtu-add-drain]
11	root	rt	0	0	0	0	S	0.0	0.0	0:11.75 [watchdog/0]
12	root	rt	0	0	0	0	S	0.0	0.0	0:10.22 [watchdog/1]
13	root	rt	0	0	0	0	S	0.0	0.0	0:14.83 [migration/1]
14	root	20	0	0	0	0	S	0.0	0.0	7:34.07 [ksoftirqd/1]
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 [kworker/1:0H]
17	root	rt	0	0	0	0	S	0.0	0.0	0:10.16 [watchdog/2]
18	root	rt	0	0	0	0	S	0.0	0.0	0:16.70 [migration/2]
19	root	20	0	0	0	0	S	0.0	0.0	5:56.64 [ksoftirqd/2]
21	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 [kworker/2:0H]
22	root	rt	0	0	0	0	S	0.0	0.0	0:10.23 [watchdog/3]
23	root	rt	0	0	0	0	S	0.0	0.0	0:15.60 [migration/3]
24	root	20	0	0	0	0	S	0.0	0.0	6:39.99 [ksoftirqd/3]
26	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 [kworker/3:0H]
27	root	rt	0	0	0	0	S	0.0	0.0	0:10.40 [watchdog/4]
28	root	rt	0	0	0	0	S	0.0	0.0	0:15.52 [migration/4]
29	root	20	0	0	0	0	S	0.0	0.0	5:52.68 [ksoftirqd/4]
31	root	0	-20	0	0	0	S	0.0	0.0	0:00.00 [kworker/4:0H]
32	root	rt	0	0	0	0	S	0.0	0.0	0:10.42 [watchdog/5]

# What have we learned?

- What is High Performance Computing
- What is a Supercomputer
- Some ways to make workflows faster
- What is a terminal emulator
- How to connect remotely
  - public key encryption and how to use it
  - Connection alternatives (X Windows, sshfs)
- What is the “Linux” operating system
- What is a “program” or “process”