Programming Workshop with Trial Account

Supercomputing for Beginners

29th Sep. 2022

Information Technology Center, The Univ. of Tokyo

11th Oct. 2022 v2.0

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

Agenda

11:00 – 12:00

Introduction, How to use Zoom for this tutorial

Login tutorial (lecture + exercise)

13:00 – 14:15

Write, compile, and run programs (lecture + exercise)

Compile and run parallel programs (lecture + exercise)

14:30 – 16:00

How to run machine learning on supercomputers (lecture + exercise)

Important notices

- Your account will be valid for one month (from today).
  - Comply with “Terms of use” (https://www.cc.u-tokyo.ac.jp/en/guide/application/rules.php)
    - ✓ You may use supercomputers only for contributing to academic research, education, and society.
  - Your account and your files on the supercomputer will be deleted afterwards.
  - The account will be disabled if you do not keep attending to this lecture till its end.
- For participants from industries: please finish one of the series of these workshops@ITC, if you want to apply to “trial usage”.
- If you have any questions, do not ask ITC official e-mail. Please write instead to the lecturer in charge:

  shiba [at] cc.u-tokyo.ac.jp

For mutually communication on Zoom

■ Please ask questions by….

(A) Dropping a line on Zoom chat

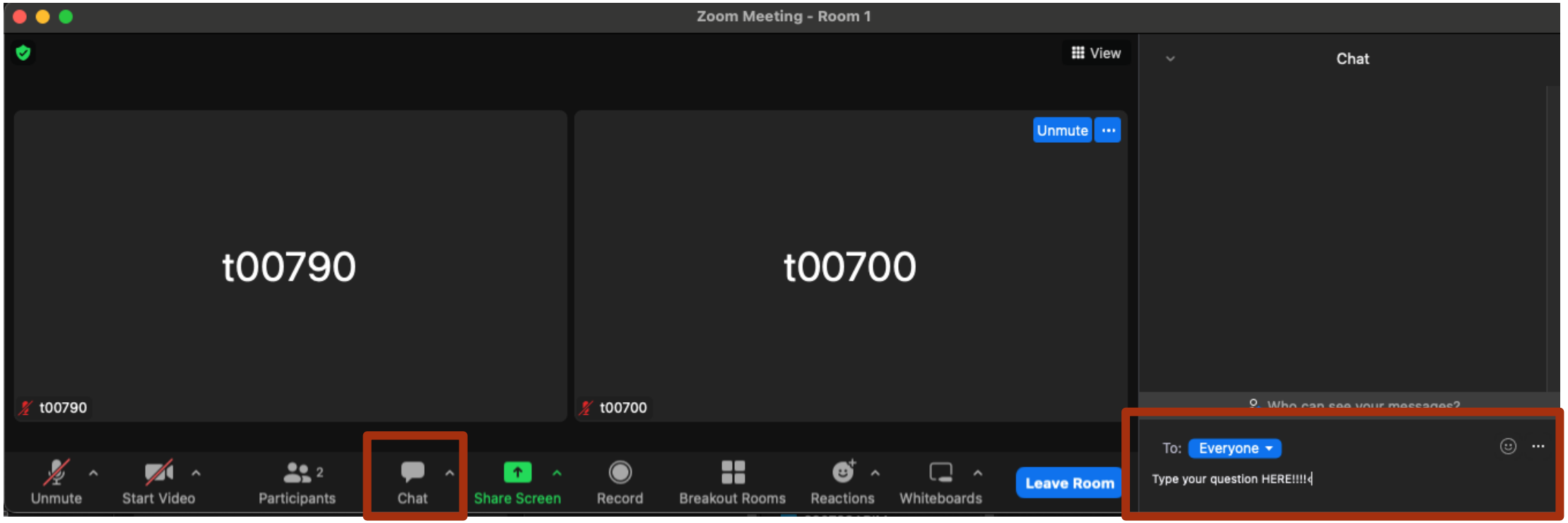
(B) “Raising hand” and then speak via your microphone

- https://utelecon.adm.u-tokyo.ac.jp/zoom/how_to_use

How to make questions on Zoom - (A) drop a line in CHAT

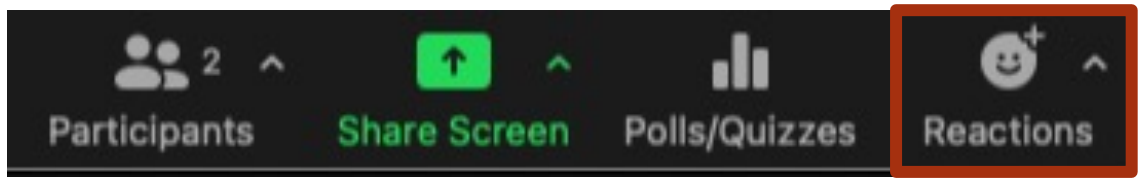
Recommended way to ask a question :

— Click “Chat”, then directly type.

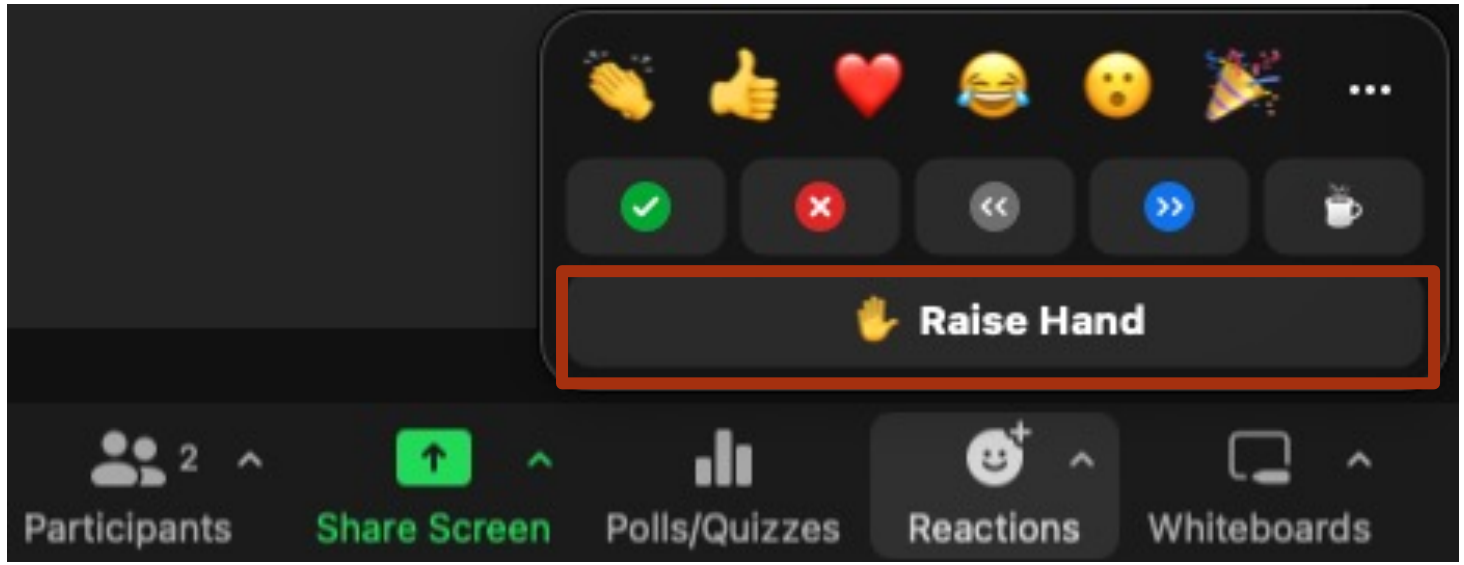


How to make questions on Zoom - (B) Raise Hand

1. Click “Reactions ” on the menu bar



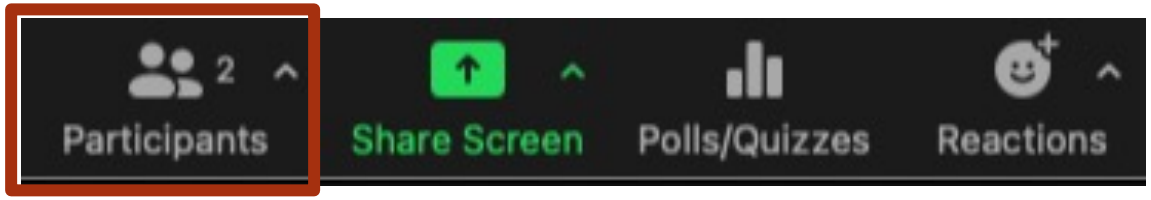
2. Then click “Raise hand”



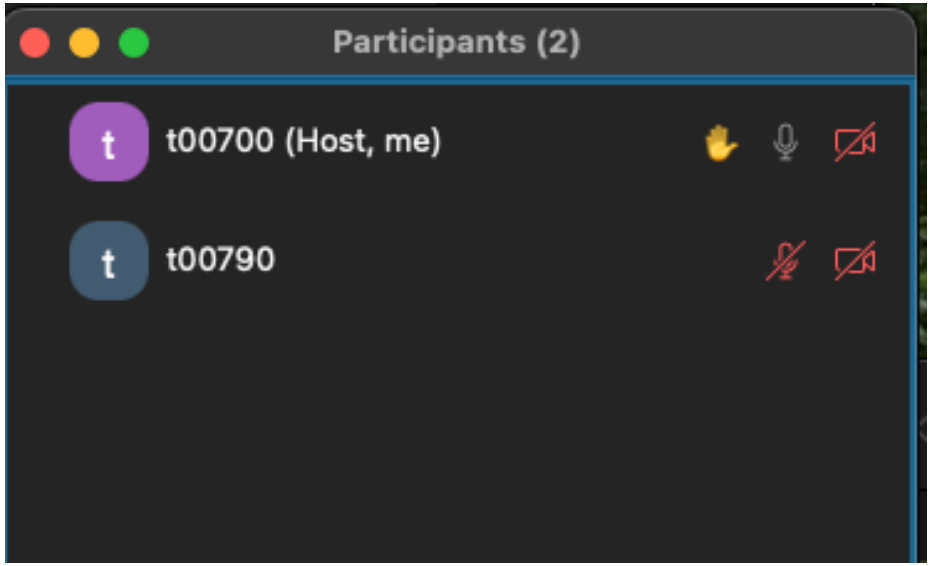
Lecturer may also ask you to react by “Raising Hand”.

How to make questions on Zoom - (B) Raise Hand

1. Click “Participants” in the menu bar, then you will find the participant list

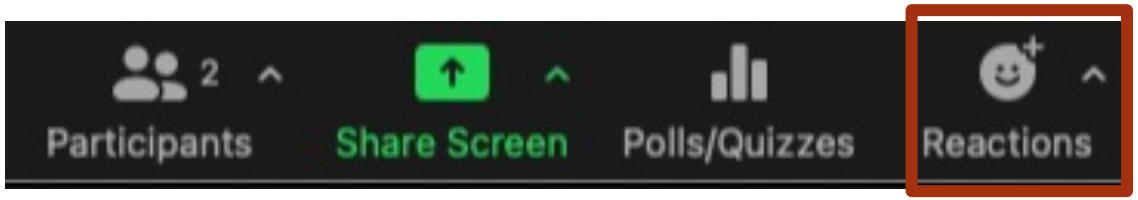


2. You can confirm if your hand is raised or not in the participants list.

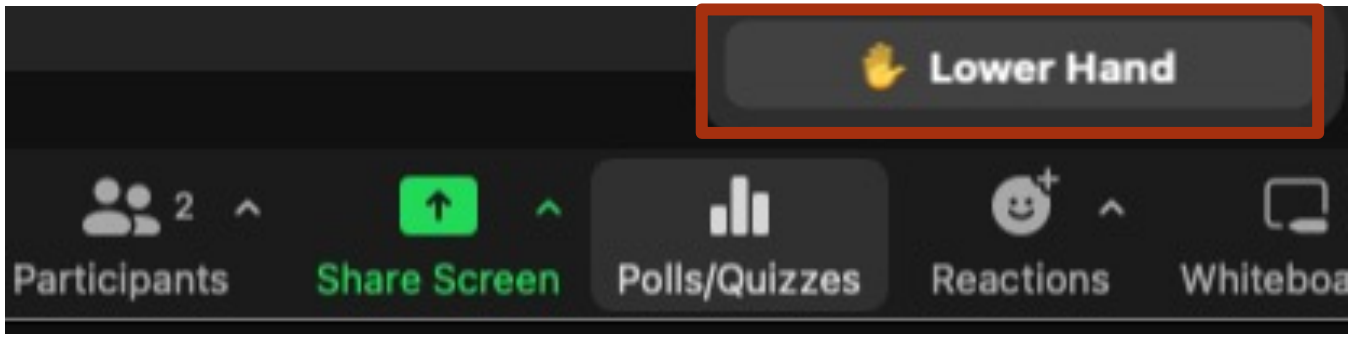


When Q&A finishes or when the Lecturer asks you to do so

1. Click “Reactions” and …



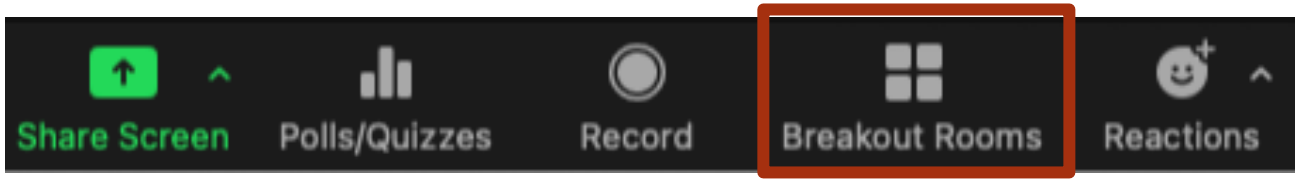
2. Click “Lower hand”



Breakout room (1/4)

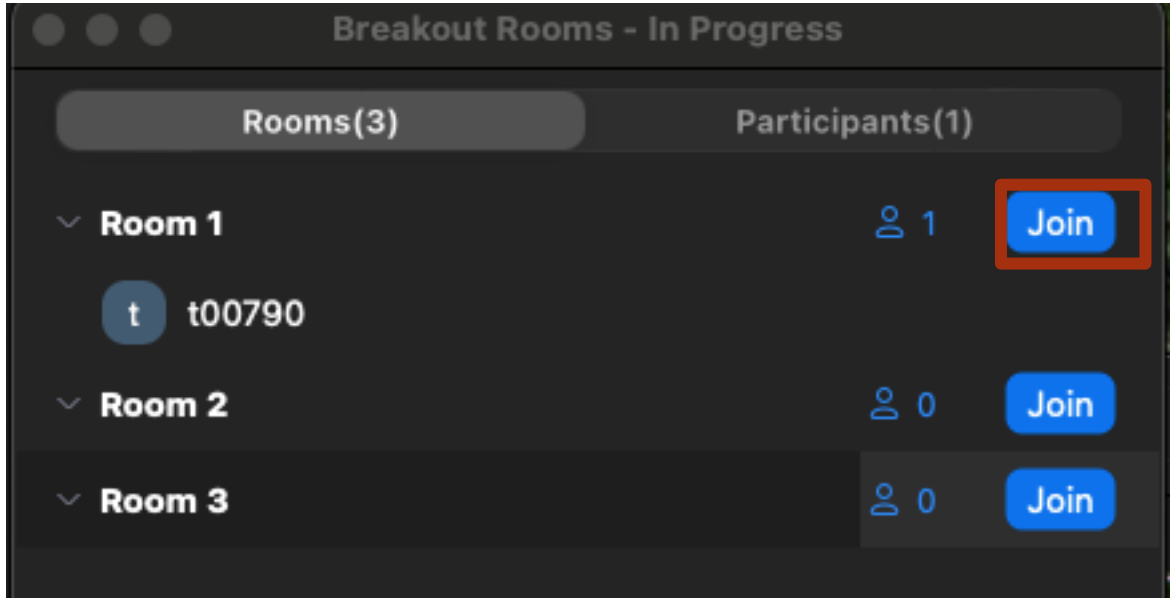
- We may perhaps use a breakout room.
  - — you can enter into a separate session with a teaching staff

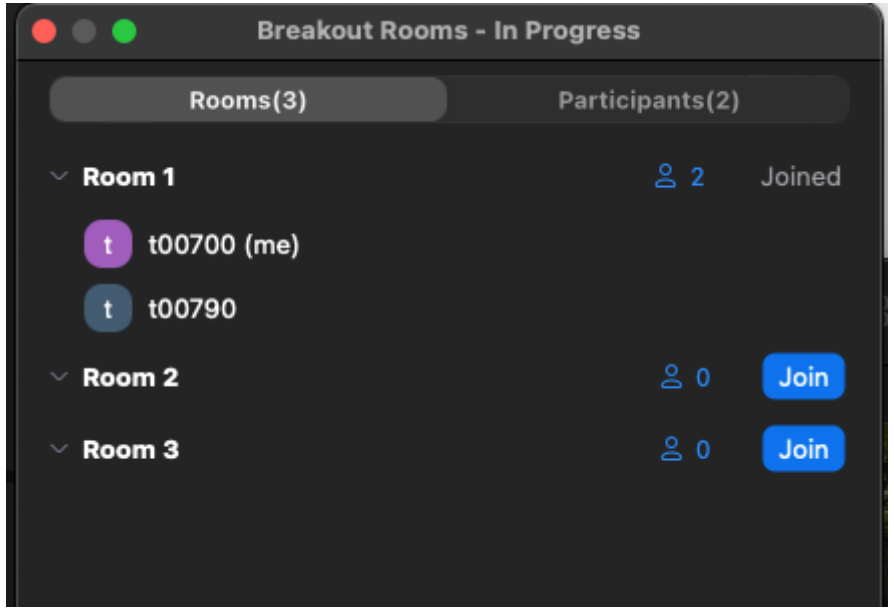
- Click “Breakout Rooms” in the Zoom menu bar.



Breakout room (2/4)

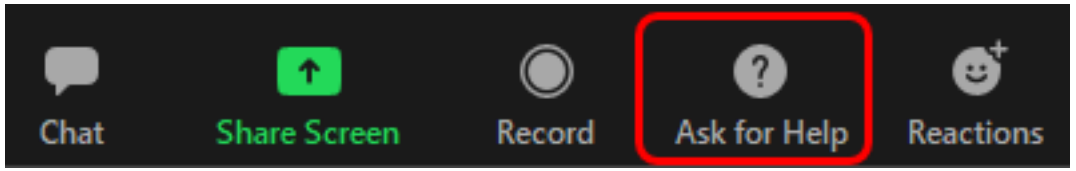
- You will can enter a ongoing brakeout room.
  Left: You can join Room 1 by clicking “Join”, other rooms are empty.
  Right: You are already in Room 1.



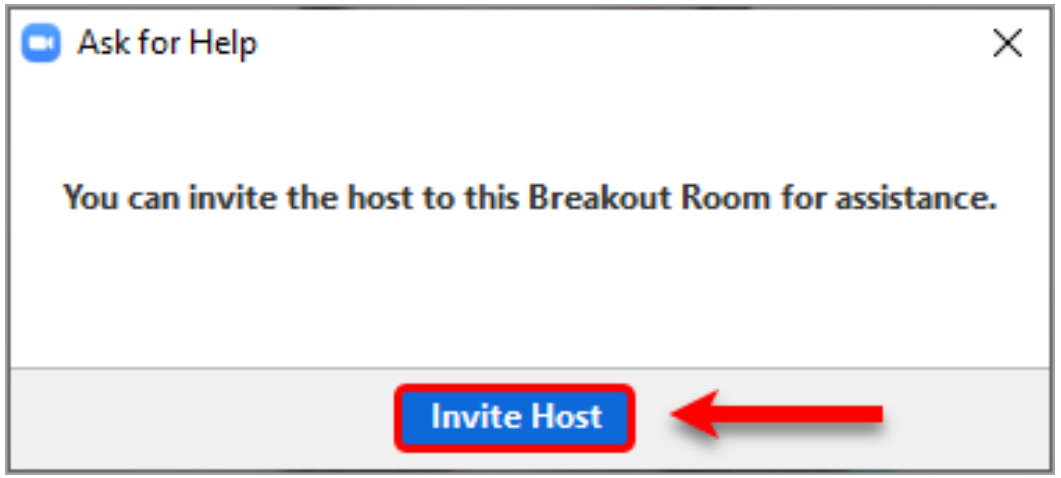


Breakout room (3/4)

When you are in a brakeout room, you have a button:

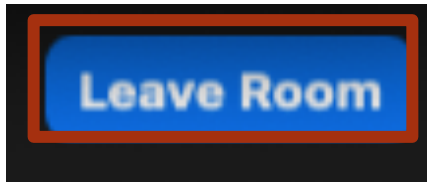


Then you will have a popup for inviting a teaching staff into the breakout room.

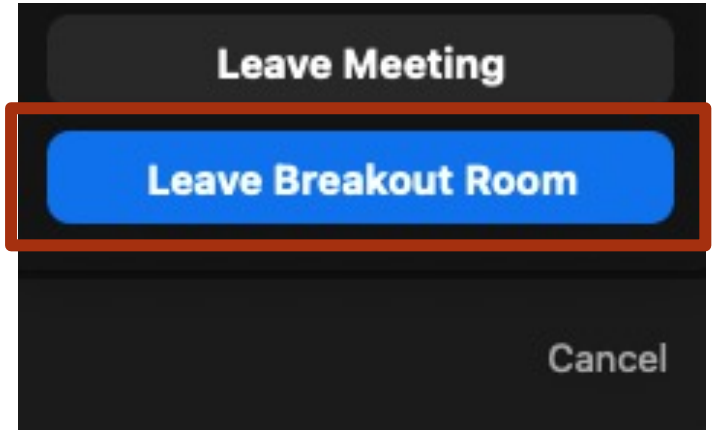


Breakout room (4/4)

After finishing discussions, you can go out of the breakout room by clicking “Leave Room”



and then choosing “Leave Breakout Room”. Then you get back to the main session. DO NOT CLICK “Leave Meeting”.



Supercomputing Division, Information Technology Center, Univ. of Tokyo

- Originally founded as
  “Supercomputing Center, The Univ. of Tokyo” in 1965
  - the oldest academic supercomputer center in Japan

- Has become a part of Information Technology Center (ITC) in 1999.
  - Affording nation-wide academic infrastructure for supercomputing.
  - Core organization of the “Joint Usage/Research Center for Interdisciplinary Large-Scale Information Infrastructures (JHPCN)”.
  - Also a part of HPCI in Japan.

- Supercomputers at ITC has 2,600+ users,
  55% are from outside U-Tokyo.

Supercomputers at ITC, University of Tokyo

Now our supercomputers are in Kashiwa campus

(suburbs Tokyo, 50km away)

Oakbridge-CX

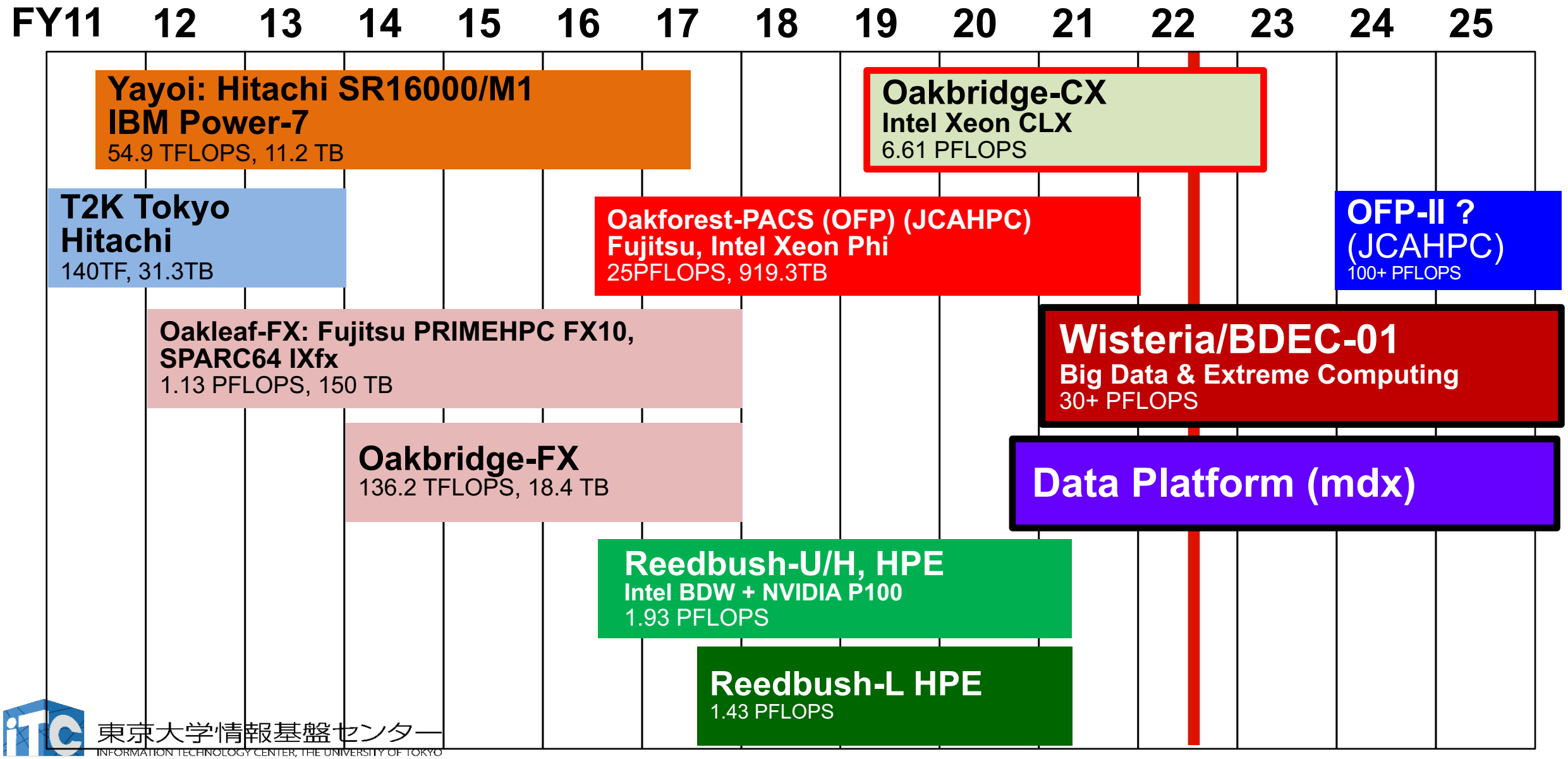
Oakforest-PACS

Wisteria/BDEC-01

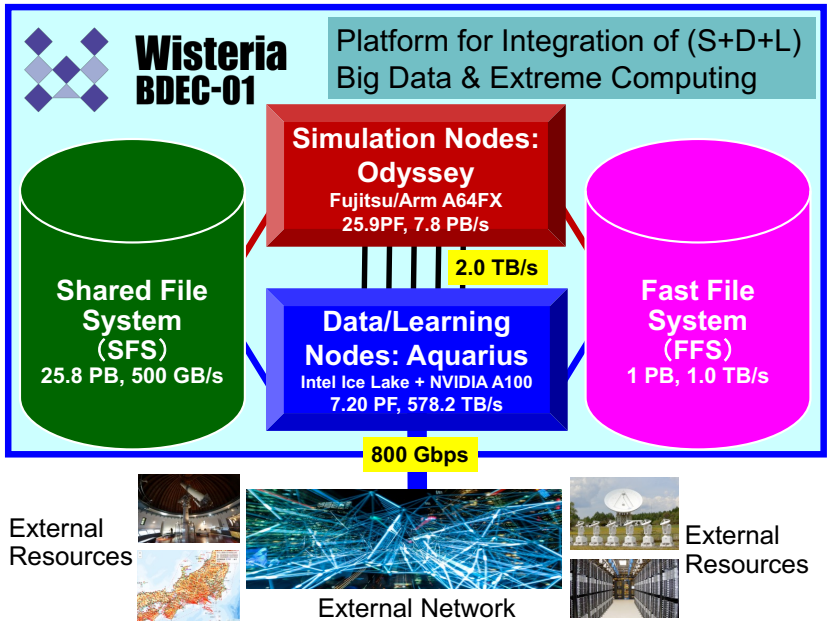
mdx

+ ABCI (by AIST)

Supercomputers at Information Technology Center, Univ. of Tokyo



Supercomputers at ITC, Univ. of Tokyo.

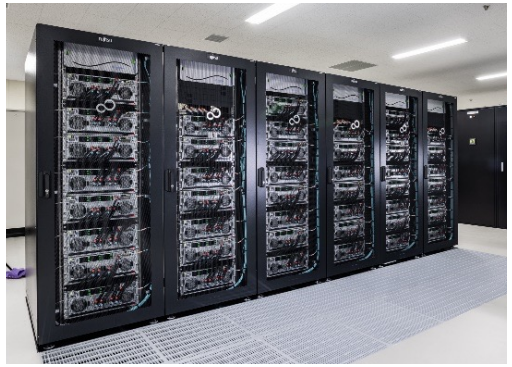


Wisteria/BDEC-01（Fujitsu） since 5/14 2021

- Simulation nodes（Odyssey）:
  Fujitsu A64FX CPUs 368,640 cores in total
- Data/Learning nodes（Aquarius）:
  Intel IceLake + NVIDIA A100 GPUs (45 nodes, 360 GPUs)
  - 33.1 PetaFlops, #20 in 59th TOP 500
  - Designed for combination of “simulation + data + learning”

Oakbridge-CX (OBCX) (Fujitsu), 6/2019 – **7/2023** (plan)
Intel Xeon Platinum 8280 (CLX), 1,368 nodes
6.61 PF, #119 in 59th TOP500 (June 2022)

Simulation Nodes
（Wisteria-Odyssey）



Data/Learning Nodes
（Wisteria-Aquarius）

Oakbridge-CX
(OBCX)

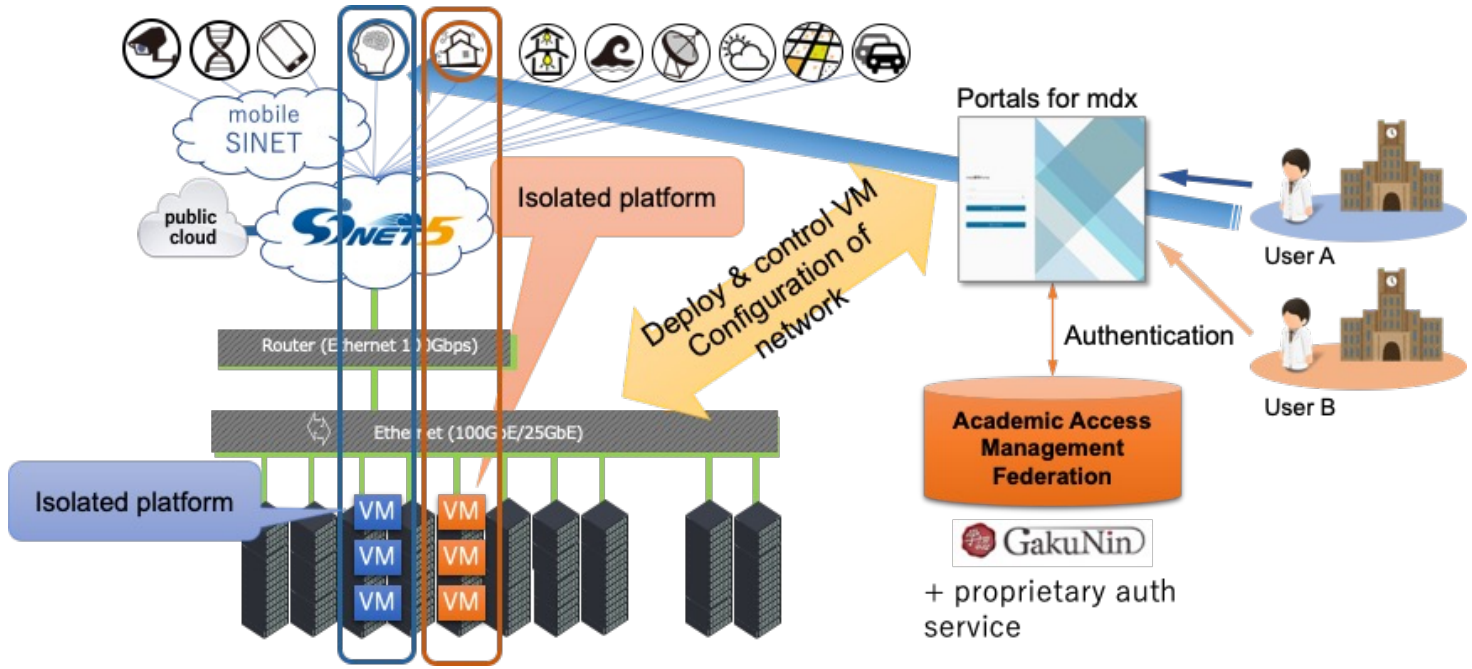
Ipomoea-01
25.9 PB
（大規模共通ストレージ）

Supercomputers at ITC, Univ. of Tokyo.

mdx https://mdx.jp

"Cloud"-type infrastructure = supercomputer of unconventional type
Operated by 9 univs. and 2 institutes (incl. Data Science Div. ITC, UTokyo)

- Virtual machine on PaaS
- Secure and high-speed storage.
- Linked to SINET6
  = 400 Gbps nation-wide academic network



The supercomputer for the hands-on today

Oakbridge-CX @ Kashiwa Campus (2019)

First thing to do for using supercomputers = Login by SSH !

login node = entrance where you can run commands

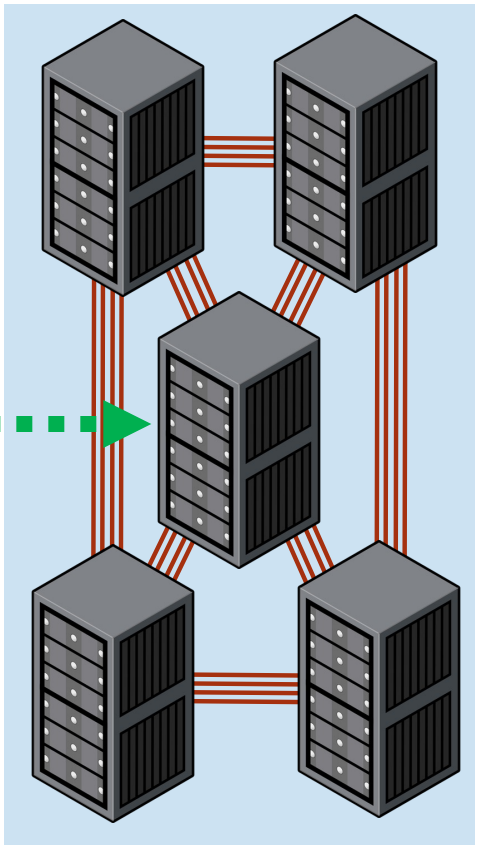
```
[tut138@obcx02~]$ ls -l
drwxr-x--- 2 shiba group 10 1
Apr 13:00 test.out
[tut138@obcx02~]$ ./test.out
Hello world
[tut138@obcx02~]$ qsub a.sh
```

Your terminal

Login

SSH

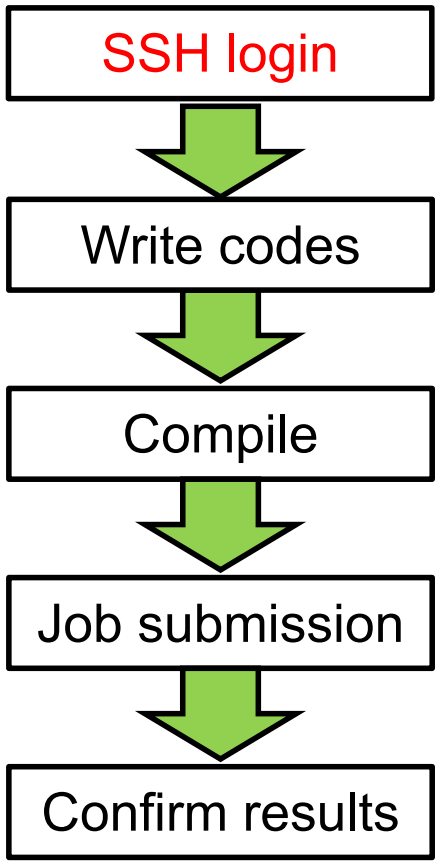
Login nodes



Compute nodes

You need to prepare your SSH key to establish connection

How to use supercomputers ?



SSH, key authentication

How to use editors (emacs, vi, nano), Linux basics

Environment module, compiler options

Writing job scripts, how to manage jobs

Secure Shell protocol

- Shell = command base software, connecting the user and the OS
- SSH = encrypted connection that enables remote connection

Via the SSH connection, you can

- Copy files
- Forward the GUI
- Tunneling
- Mount to your directory

Shell command example after SSH login

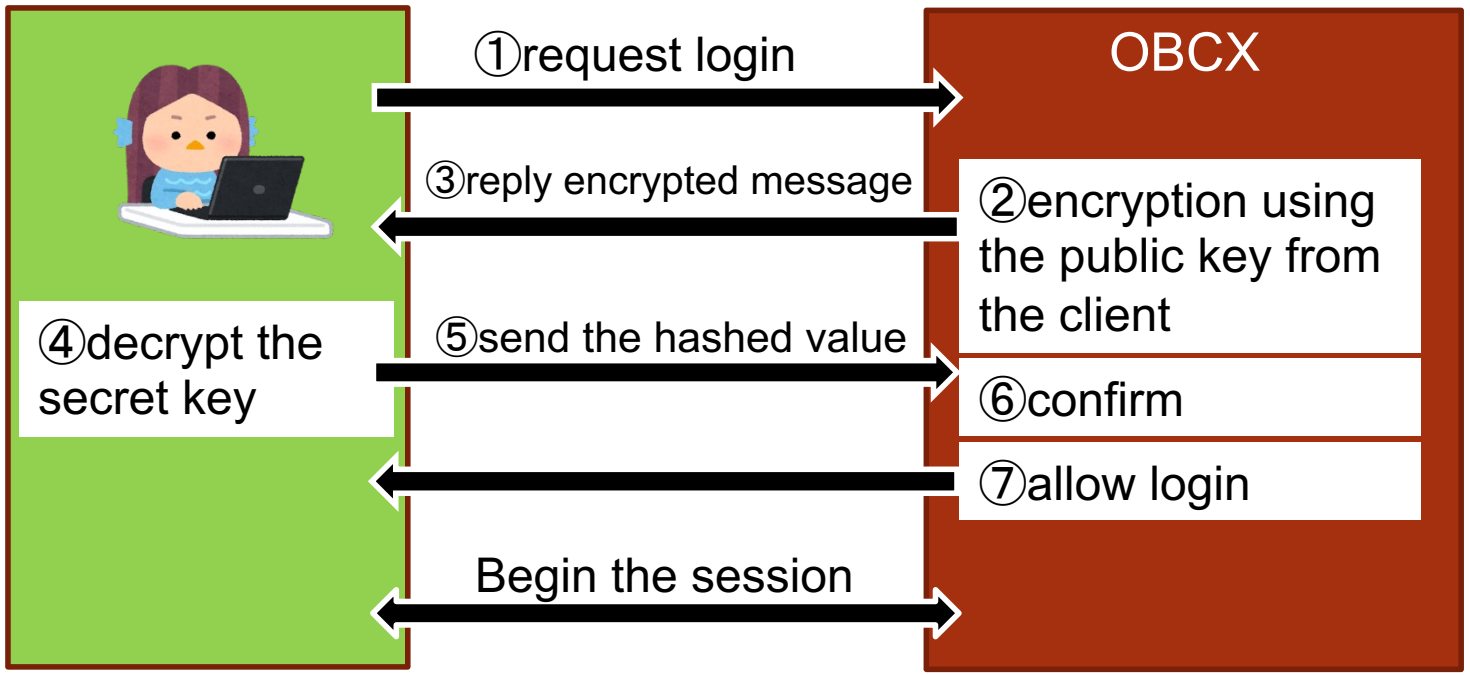
```
[ tUVXYZ @obcx05 ~]$ pwd
/home/ tUVXYZ
[ tUVXYZ @obcx05 ~]$ cd /work/gt00/z30113
[ tUVXYZ @obcx05  tUVXYZ ]$ cd ../
[ tUVXYZ @obcx05 gt00]$ pwd
/work/gt00
[ tUVXYZ @obcx05 gt00]$ cd ~/
[ tUVXYZ @obcx05 ~]$ pwd
/home/z30113
[ tUVXYZ @obcx05 ~]$ cd /work/gt00/z30113
[ tUVXYZ @obcx05  tUVXYZ ]$ mkdir test
[ tUVXYZ @obcx05  tUVXYZ ]$ ls
test
[ tUVXYZ @obcx05  tUVXYZ ]$
```

SSH key authentication

enables secure connection to supercomputers

Use “SSH key pairs” instead of (plain text) passwords.

Initial settings (for the first login)
- Generate key pairs
- Register the public keys on the login node



Cautions on SSH key authentication

Your secret key should be kept strictly confidential

✓ to prevent illegal login of other people.

- Do not copy your secret key to any other places

  = The key pairs should not be recycled (on other machines)

- Set PASSPHRASE when generating your SSH key!
- The passphrase of your key should be different from other passwords, including those of OBCX user portal and user password on OBCX.

SSH key authentication

Generate your key pair on your local computer (1/3)

Open Cygwin or Terminal and begin the following operation

```
$ ssh-keygen -t rsa

Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa): Return
Enter passphrase (empty for no passphrase): Set your passphrase Return
Enter same passphrase again: The same passphrase Return
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.

The key fingerprint is:
SHA256:vt88O+PTcscHkOyabvxGjeRsMWLAWds+ENsDcReNwKo tut138@ITCUT-VAIO
The key's randomart image is:
+---[RSA 2048]----+
|        . o=oo.o+ |
|         +  O... .|
|          .+o+.   |
|           +oB.   |
|         So *o*   |
|        .E   B.o  |
|         .. = . o |
|          .=oB o +|
|         .+o+*O ..|
+----[SHA256]-----+
```

How to generate key

- **ssh-keygen -t rsa <Return>**
- <Return>
- Passphrase as you like<Return>
- The same passphrase<Return>

SSH key authentication

Confirm that there are both secret and public keys on your local computer(2/3)

If you do not understand the meaning, type letters and confirm a similar output

```
$ cd .ssh

$ ls

 id_rsa                          ⇒  Private Key
 id_rsa.pub                      ⇒  Public Key

$ cat id_rsa.pub

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDa6InmOYYaCrWjQDukjiNEfdW8veUwJyZtEI3oDuOA28eey6pOwbtI7JB
O9xnI17O7HG4yYvOM81+/nlAHy5tAfJlyOdsPzjTgdTBLdgi3cSf5pWEY6U96yaErOEi8Wge1HkXrhcwUjGDVTz
vTORefe6zLdRziL/KNmmesSQfR5lsZ/ihsjMgFxGaKsHHq/IErCtHIIIf9V/Ds2yj6vkAaWH6asBn+ZsRiRFvwH
PhkYAnp/j3LY6b8QfqgOp4WZRenh/HgySWTYIGi8x67VzMaUlm9qlKOQFMCaK2rivX1fmbwyWJ/vrWDqiek6YXo
xLDu+GPeQ4CPvxJcZnqF9gf3 tut138@ITCUT-VAIO
```

SSH key authentication

Copy public key（3/3）

Cut and paste your “id_rsa.pub” file

```
$ cd .ssh

$ ls

 id_rsa
 id_rsa.pub

$ cat id_rsa.pub

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDa6InmOYYaCrWjQDukjiNEfdW8veUwJyZtEI3oDuOA28eey6pOwbtI7JB
O9xnI17O7HG4yYvOM81+/nlAHy5tAfJlyOdsPzjTgdTBLdgi3cSf5pWEY6U96yaErOEi8Wge1HkXrhcwUjGDVTz
vTORefe6zLdRziL/KNmmesSQfR5lsZ/ihsjMgFxGaKsHHq/IErCtHIIIf9V/Ds2yj6vkAaWH6asBn+ZsRiRFvwH
PhkYAnp/j3LY6b8QfqgOp4WZRenh/HgySWTYIGi8x67VzMaUlm9qlKOQFMCaK2rivX1fmbwyWJ/vrWDqiek6YXo
xLDu+GPeQ4CPvxJcZnqF9gf3 tut138@ITCUT-VAIO
```

Procedure

- **cat id_rsa.pub <Return>**
- Drag and Drop from “ssh-rsa” to the filnal part(“tut138@ITCUT-VAIO” in this case)

SSH key authentication

id_rsa

- Private Key — keep it on your PC
- Keep it confidential, do not move it from where it is generated, do not send it to others

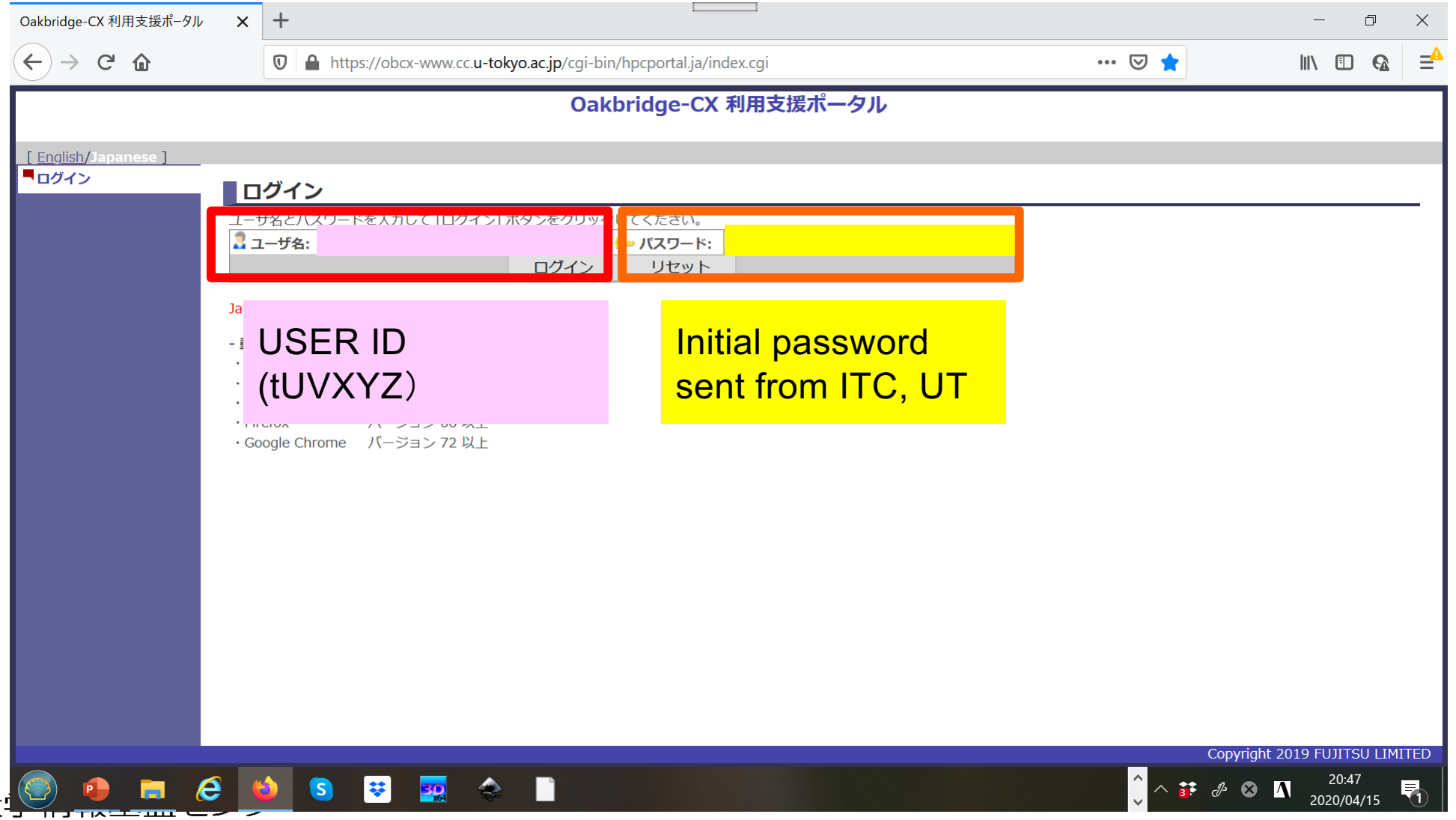
id_rsa.pub

- Public Key — put on supercomputer
- You may copy and send it to others by e-mail.

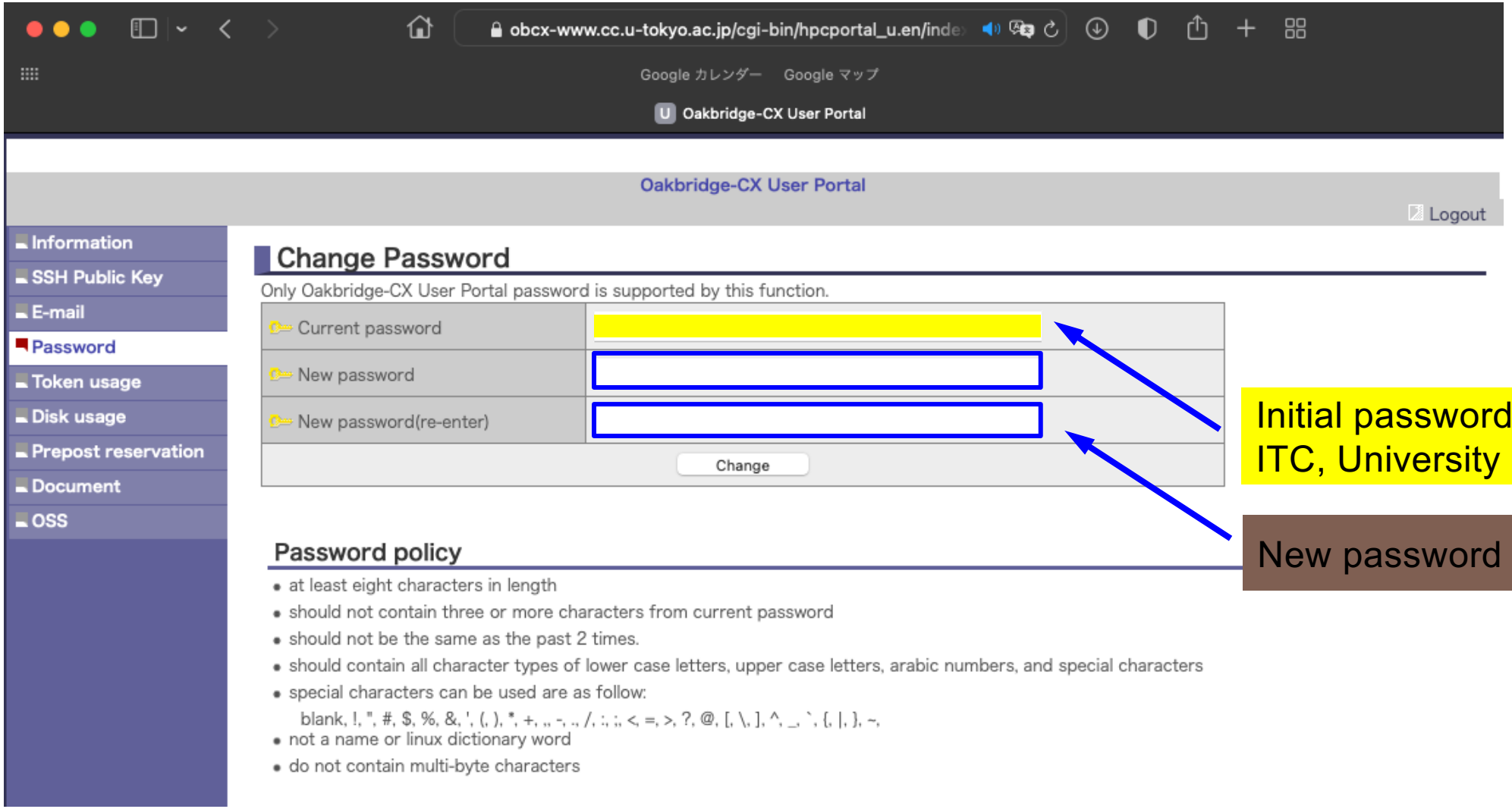
- If you want to log into a supercomputer from multiple local computers, then generate a pair of public and private keys on each local computer. You can register multiple public keys.

(1) Login to OBCX user portal

https://obcx-www.cc.u-tokyo.ac.jp/cgi-bin/hpcportal.en/index.cgi



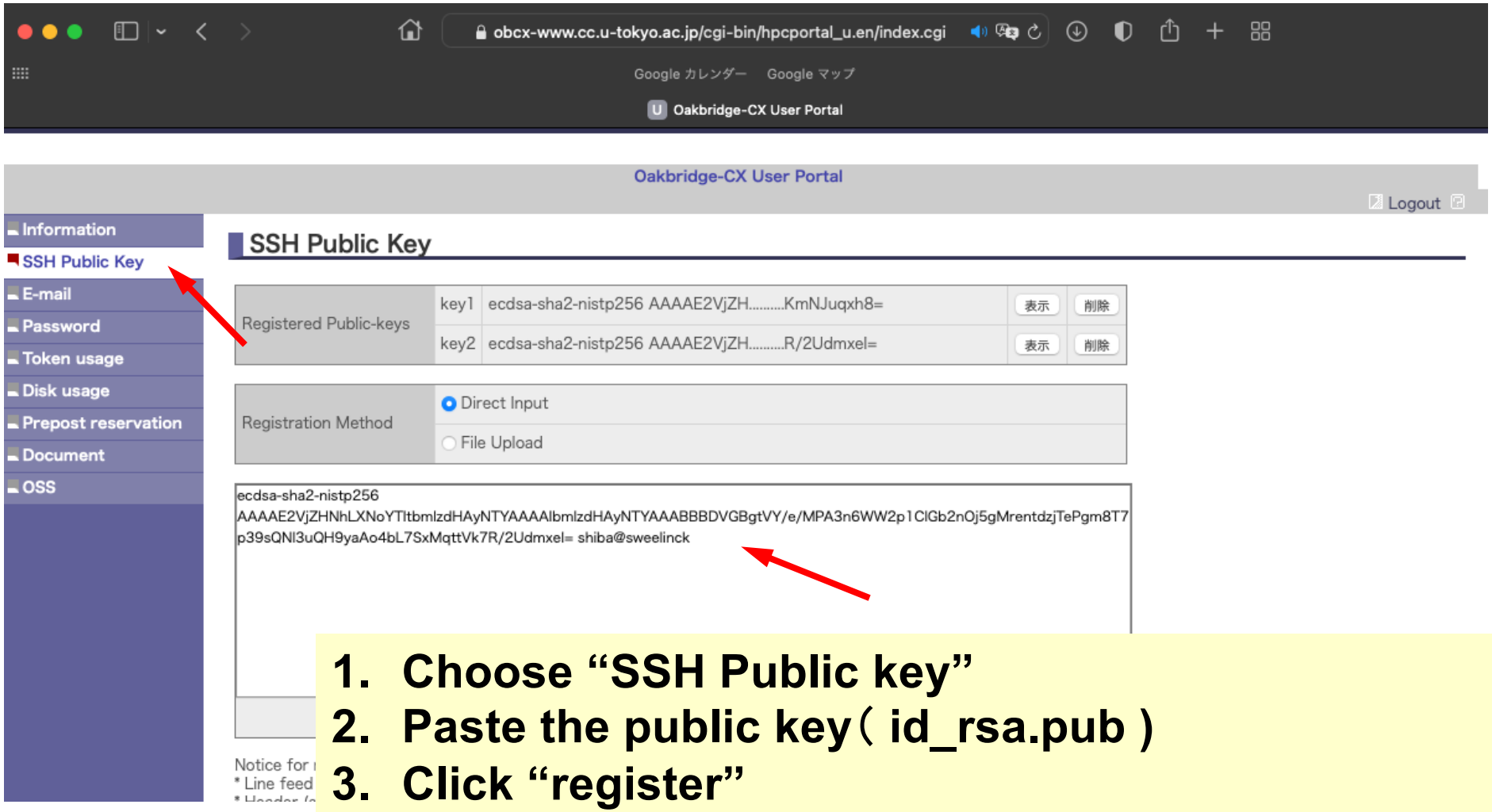
(2) Change your portal password



Initial password sent from ITC, University of Tokyo

New password (twice)

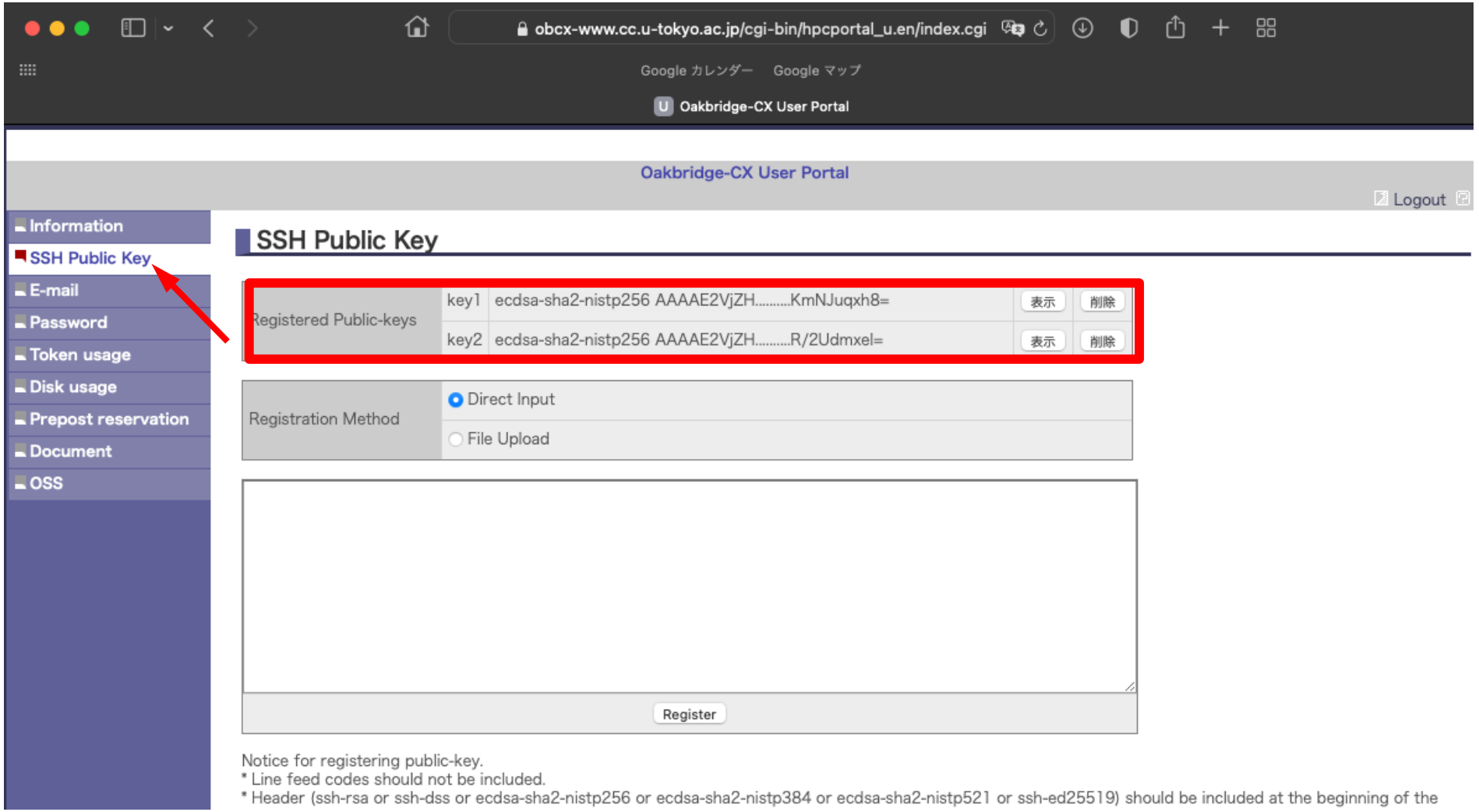
(3) public key registration（id_rsa.pub）



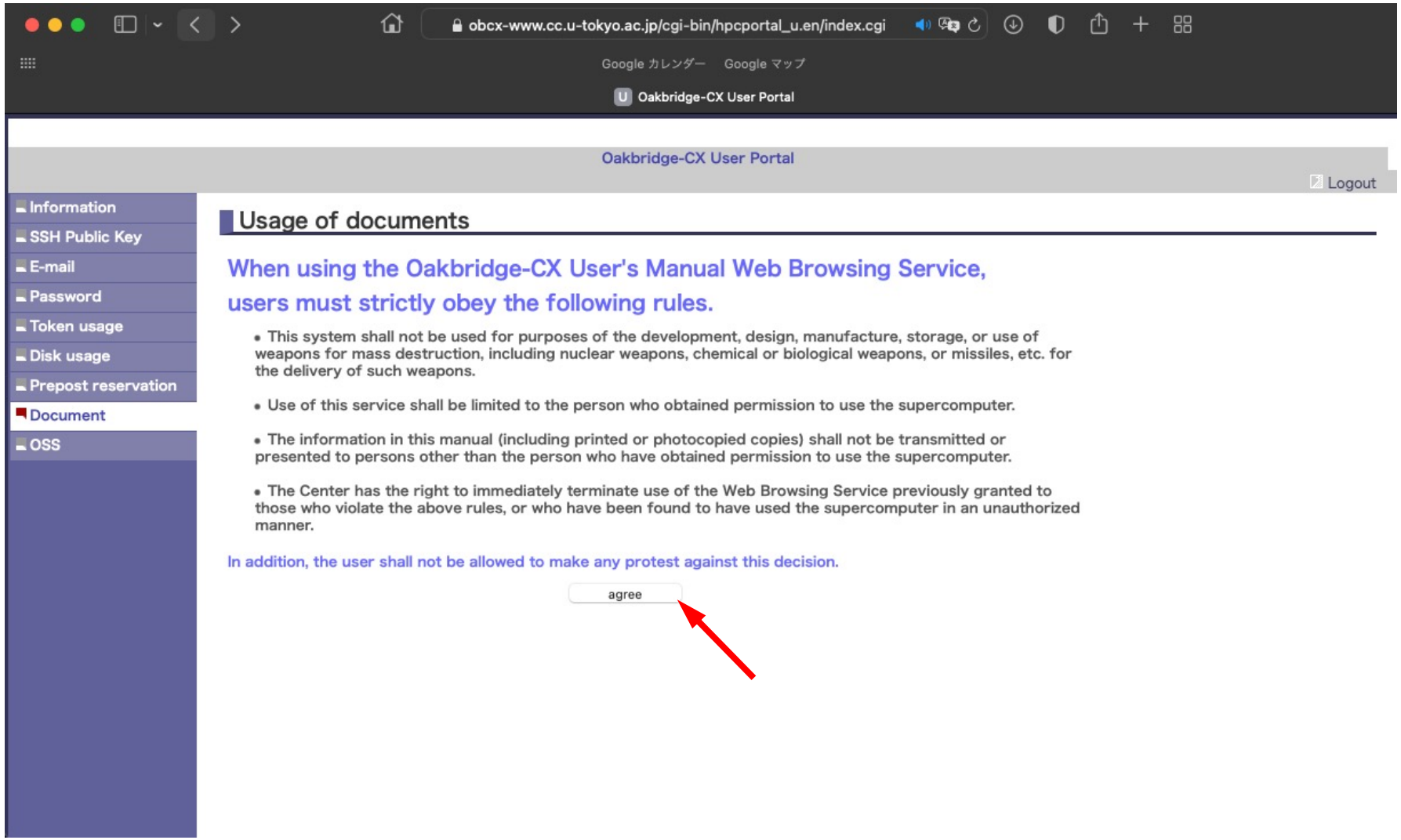
1. Choose “SSH Public key”
2. Paste the public key（ id_rsa.pub ）
3. Click “register”

Ask the staffs if you have troubles

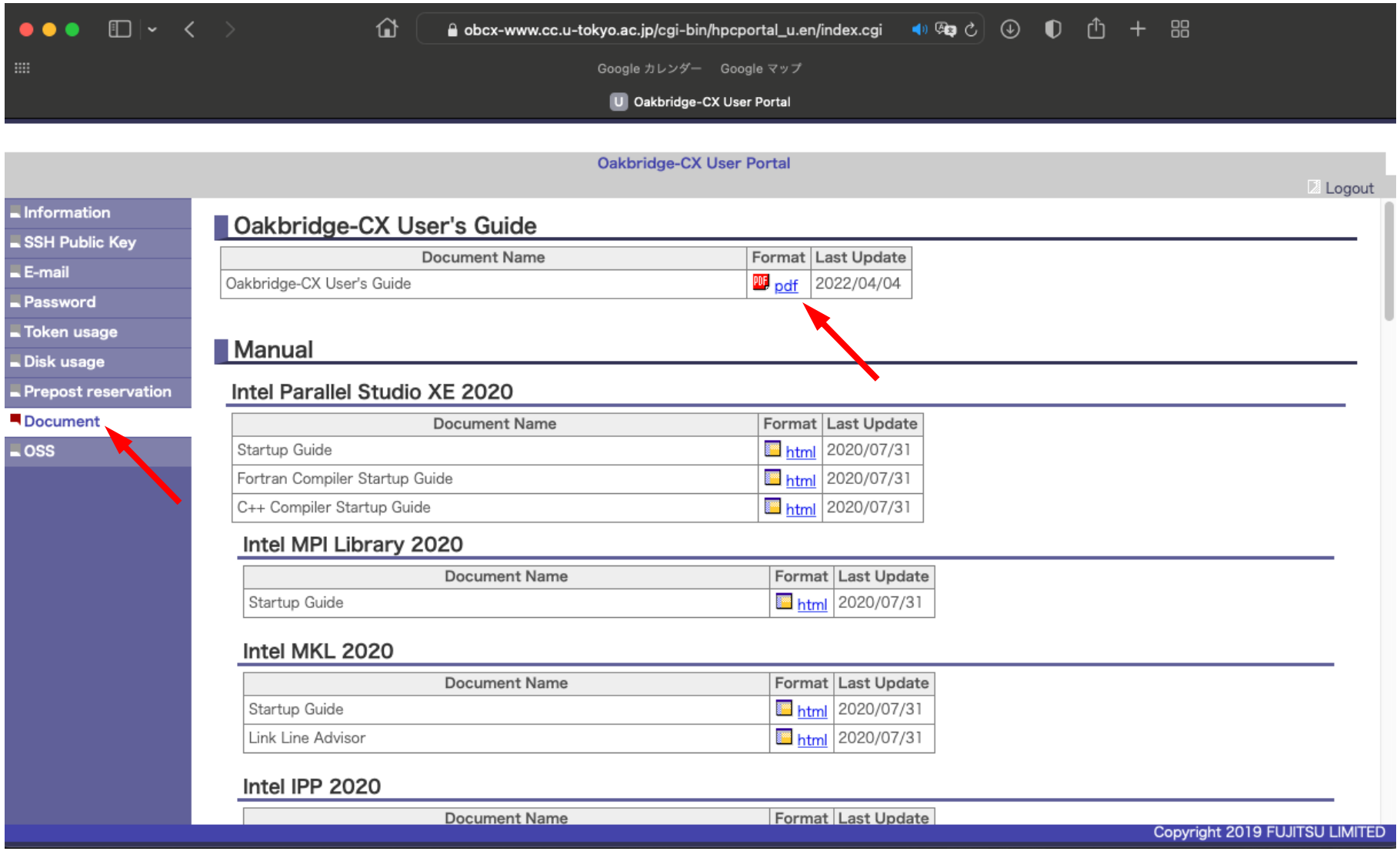
More than one public keys can be registered on OBCX



Get Manual PDF files on OBCX portal （1/2）



Get Manual PDF files on OBCX portal （2/2）



Login to OBCX supercomputer

```
$ ssh tUVXYZ@obcx.cc.u-tokyo.ac.jp
Enter passphrase for key '/home/tut138/.ssh/id_rsa:
```

Your Passphrase | Return

1. ssh tUVXYZ@obcx.cc.u-tokyo.ac.jp <Return>
2. Passphrase <Return>

Login to OBCX

```
Last login: Sun Apr 12 15:05:47 2020 from obcx01.cc.u-tokyo.ac.jp
------------------------------------------------------------------------
  Oakbridge-CX Information                              Date: Apr. 03, 2020
------------------------------------------------------------------------

  Welcome to Oakbridge-CX system

  * Operation Schedule
     04/24(Fri) 09:00 - 04/24(Fri) 20:00  System Maintenan
     04/24(Fri) 20:00 -                   Normal Operation

    For more information about this service, see
    https://www.cc.u-tokyo.ac.jp/supercomputer/schedule.php

  * How to use
    Users Guide can be found at the User Portal (https://obcx-www.cc.u-
tokyo.ac.jp/).

If you have any questions, please refer to the following URL and contact us:

    https://www.cc.u-tokyo.ac.jp/supports/contact/

  * Updated OBCX Users Guide
     10/01(Tue): v1.0

Set your email address on the User Portal [https://obcx-www.cc.u-tokyo.ac.jp]

[tUVXYZ@obcx01 ~]$
```

You will find maintenance schedule (+etc) If you have successfully logged in

After login

You will be on a Linux shell → type commands

```
[tUVXYZ@obcx01 ~]$ pwd  Return
/home/tUVXYZ
```

pwd — shows the present directory

Break

The supercomputer for the hands-on today

Oakbridge-CX @ Kashiwa Campus (2019)

OBCX specifications

Many cores & high memory bandwidth

		MacBook Air M1 (2020)	OBCX 1 node
CPU	# of cores	8	56 (2 sockets)
	clock frequency [GHz]	< 3.2	2.7
	Theoretical peak perfomance [TFlops]	0.65	4.84
Memory	amount [GByte]	8	192
	bandwidth [GByte/秒]	68.25	281.6

Water cooling
(very hot !)

OBCX = “massive scalar” supercomputer

Many CPUs work together

×4

×17

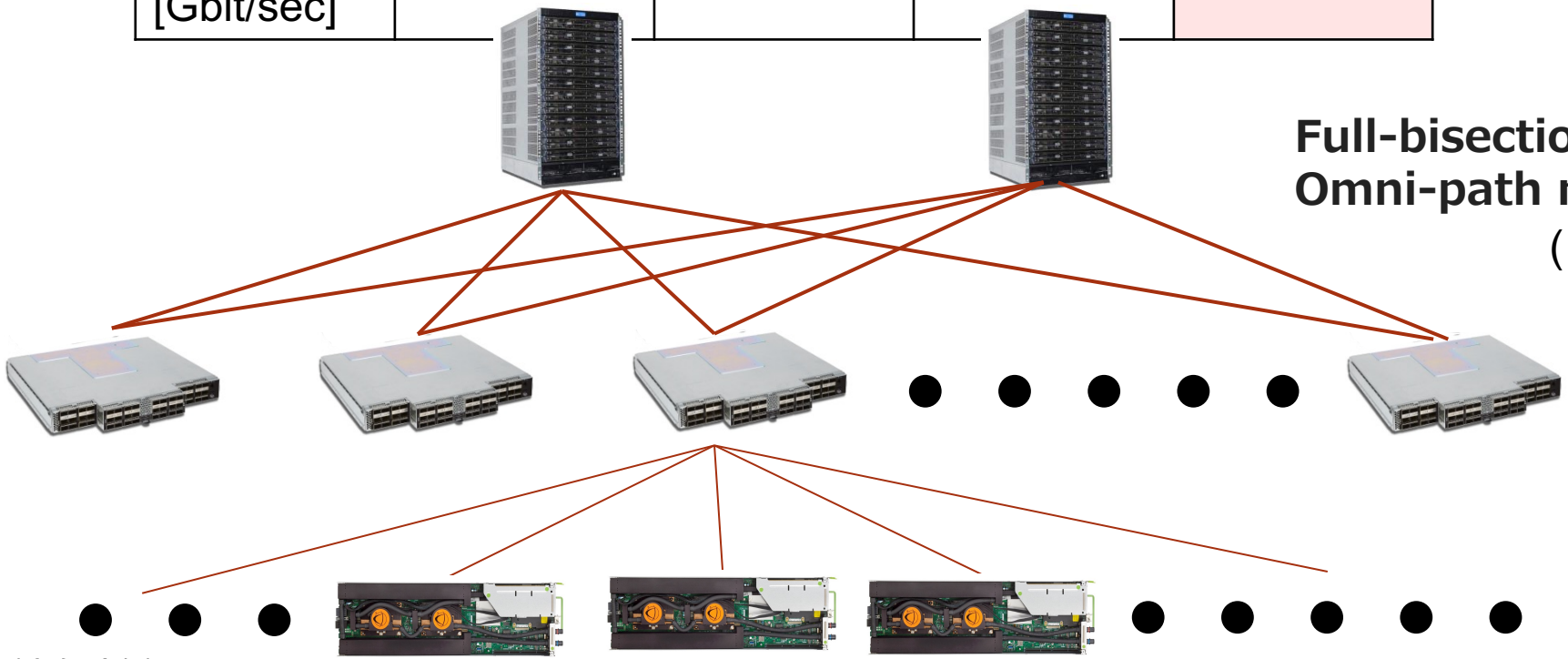
×21

1,368 nodes in total

Compute nodes are connected via high-bandwidth networks

Data transfer rate between nodes are notably high

	Mobile 4 G	Wifi 11ac	Gigabit Ethernet	Omni-path (OBCX)
Data rate [Gbit/sec]	0.1 ~ 1.0	6.9	1	**100**



Full-bisection Fat Tree
Omni-path network
(≒ infiniband)

High-performance Storages are important

Large capacity, high-speed

OBCX Lustre file system

- Capacity: 12.4PByte (12400TByte)
  8TB x 1,360 HDDs
- I/O speed — 193.9GByte/sec

＊ Typpical effective transfer rate of a storage device:
HDD ：0.1～0.2 GByte/sec
SSD ： 4～5GByte/sec

as of Sep. 2020.

The total performance

FLOPS値（Floating Point Operations per Second）

Indicating how many real number arithmetic operations can be performed

- 10^6 FLOPS= 1 Mega FLOPS = 1 MFLOPS (million)
- 10^9 FLOPS= 1 Giga FLOPS = 1 GFLOPS (billion)
- 10^{12} FLOPS= 1 Tera FLOPS = 1 TFLOPS (trillion)
- 10^{15} FLOPS= 1 Peta FLOPS = 1 PFLOPS
- 10^{18} FLOPS= 1 Exa FLOPS = 1 EFLOPS

OBCX

– Intel Xeon Platinum 8280, 86.4 GFLOPS per core

» 86.4 billion floating number operations per second

– 1 node = 56 cores

» 4,838 GFLOPS= 4.838 TFLOPS

– In total：1,368 nodes → **6.618 PFLOPS**

OBCX hardware specification

Compute nodes

Chassis: PRIEMRGY CX400 M1 x342 <4node / Chassis>
Node: PRIMERGY CX2550 M5 x1,240, CX2560 M5 x128

x1,368 node

Overall performance

Theoretical calculating performance : 6.61PF
Main storage capadity: 256.5TiB
Memory bandwidth : 385.1TB/s
Rack: 21
SSD: 128 node

Node

Theoretical calculating performance : 4.8384 TF
Main storage capadity : 192GiB
Memory bandwidth : 281.6GB/s

Network between compute nodes (Omni-Path Architecture)
Communication performance 100Gbps

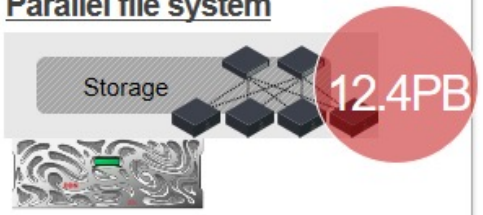
Login node

FUJITSU Server
PRIMERGY CX2560 M5 x 10

Administrative server

FUJITSU Server
PRIMERGY RX2530 M4 x 15
(Job,Operation,Authentication,
Web, Security log storage)

Parallel file system



Sorage: DDN ES18KE x 2
File system: DDN ExaScaler
(Lustre base file system)

What should you learn for supercomputing (1/3)

Case A ） Write a new program code
or speed up an existing probram made by others

Hotspots = iteration (“do” or “for”)

```
do i=1, 1000000
        c(i) = a(j) * x(i,j) + b(i)
enddo
```

1. Rewrite the programs in a way that is “friendly” to CPUs
   SIMD operations, utilization of CPU cache, improving memory alignment
2. Distributed processing by using OpenMP and/or MPI.
3. Utilize libraries for scientific computation that are already tuned and parallelized.

What should you learn for supercomputing ? (2/3)

Case B ） How to Use existing software for supercomputers

Computational Fluid Dynamics, structural analysis, statistics, molecular dynamics, electronic structure calculation, Quantum chemistry,

It is important to know the outline of how these softwares are parallelized (through the experience of using them).

What should you learn for supercomputing ? (3/3)

Case C ） Data processing (machine learning)

1. Fast data processing via Python

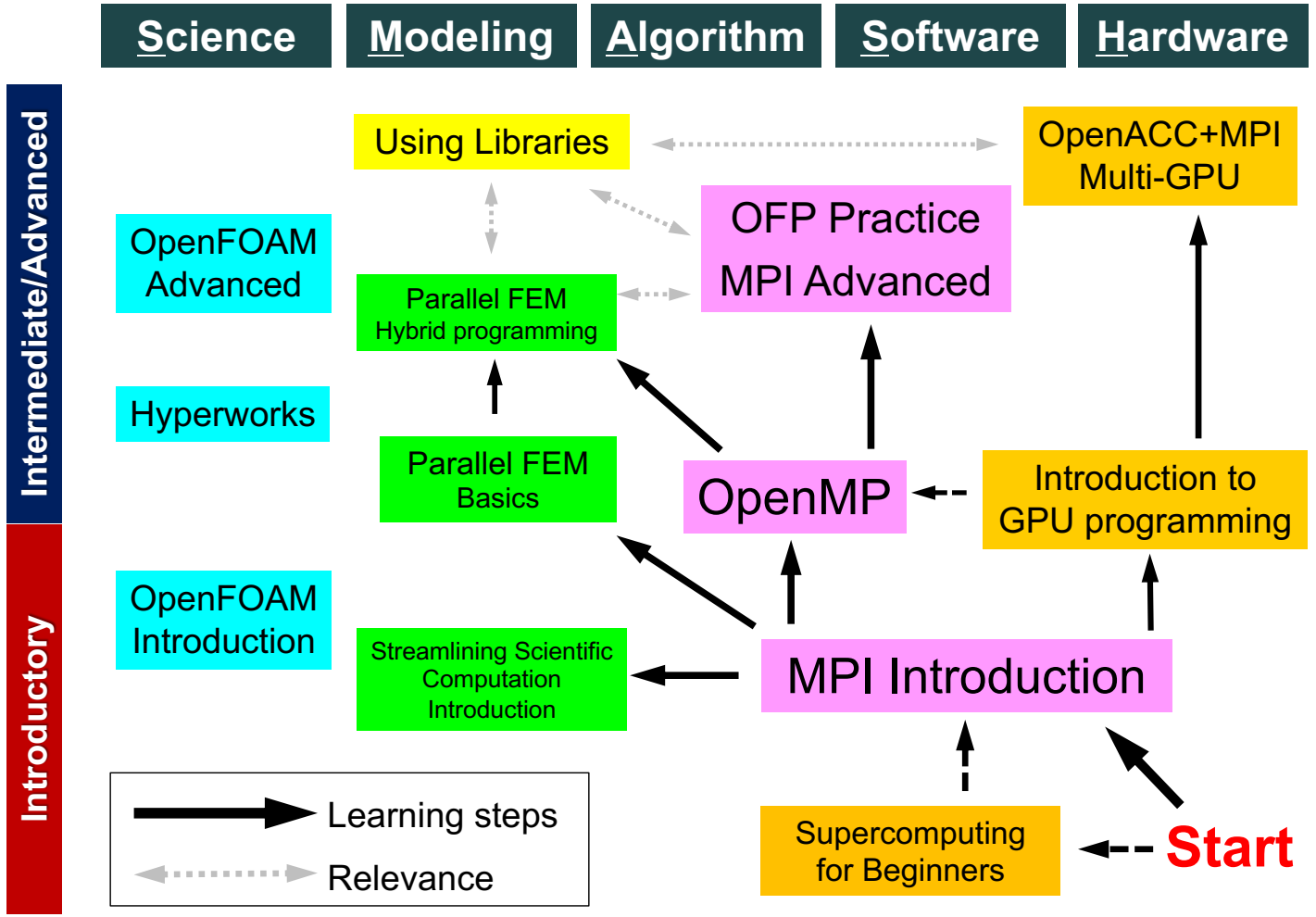
2. Environment setting specific to supercomputers (incl. multi-GPU)
   Use of ”containers” environments

3. Create batch job scripts to execute learning on supercomputers.

Note) Compared to cloud services in the private sectors,
environment setting may be a bit complex and thus annoying.

Supercomputer workshop (tutorials) at ITC, U-Tokyo

Roadmap



* all other tutorials are currently in Japanese

How to use supercomputer

login node = entrance where you can run commands

```
[tut138@obcx02~]$ ls -l
drwxr-x--- 2 shiba group 10 1 Apr 13:00 test.out
[tut138@obcx02~]$ ./test.out
Hello world
[tut138@obcx02~]$ qsub a.sh
```

Your terminal

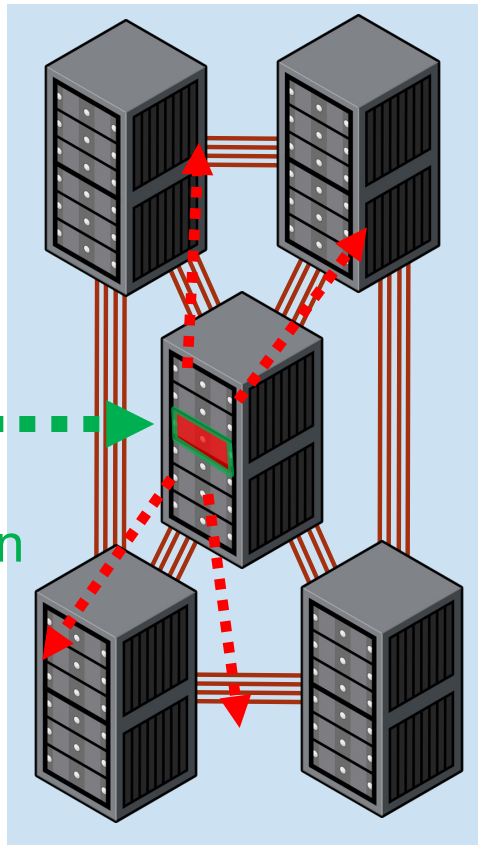
1. Login

SSH

Login nodes

2. Job submission

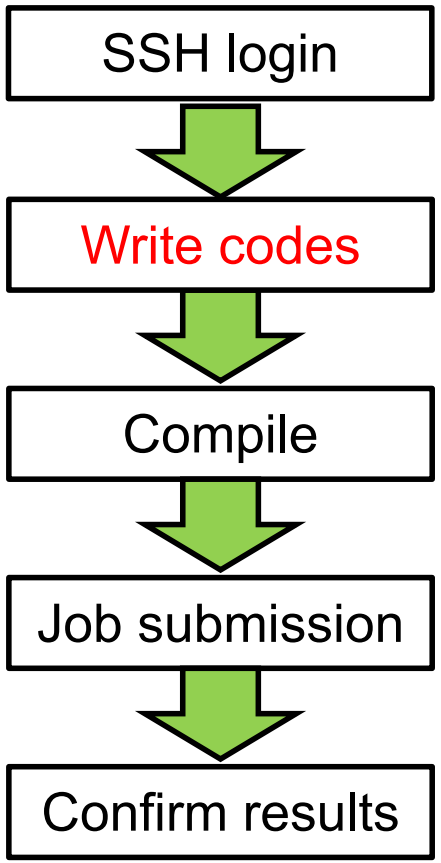
3. Program execution



Compute nodes = “main body”

You need to prepare your SSH key to establish connection

How to use supercomputers ?



Putty、Cygwin、SSH、key authentication

How to use editors (emacs, vi, nano), Linux basics

Environment module, compiler options

Writing job scripts, how to manage jobs

Commands

Order the execution of a specific function

$ *command* [option1, option2・・・] arg1, arg2・・・・・

example $ cd /home/t00270/ (change working directory)
$ cp *a.txt* *b.txt*
(make a file “b.txt” with the same content as “a.txt”)

Command examples

command	用途	Very common option
pwd	show the current directory	
cd	change the working directory	
ls	display the list of files	-l : view details
cp	copy a file or a directory	-r : recursive copy
rm	delete a file or a directory	-r : recursive deletion
exit	end the session (=log out from supercomputer)	

Other command examples

command	function	frequently used option
mv	Move a file or a directory	
mkdir	Create a new directory	
man	Display a reference manual of a command	
cat	Display the content of a file (the whole)	
less	Display the content of a file (one page)	
head	Display the content of a file (beginning part)	-n ## : number of lines
tail	Display the content of a file (tail end part)	-n ## : number of lines
grep	Search specified strings in the files	
diff	Compare the text contents of a pair of files	
echo	Text/string that are passed as an argument	
exit	End the session	

Specific to bash

Default shell on OBCX = “bash”

Basic tools for facilitating command input

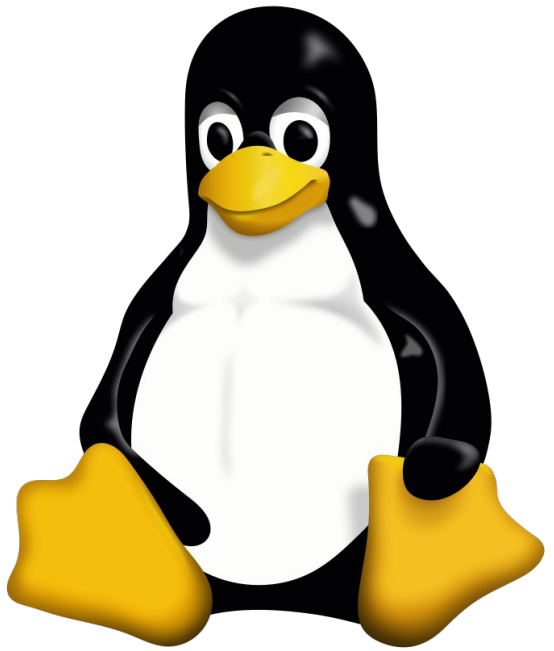
	Operation	behavior
Auto fill	tab	Complete the rest of command or paths that are partially typed
History	↑	Display previously executed commands one by one
History search	ctrl+r	Search commands that are previously executed
Redirect	>	Write output of the command into a file. EX) ls > list.txt = write the result of “ls” into “list.txt”.
Pipe	|	Use the output of a command as the input of another EX) ls | sort = sort the result of “ls”.

Linux OS

Majority of worldwide supercomputers operate on Linux OS.

- 100% share among the TOP500 supercomputers
- All components are Open Source.
- Command Line Interface, sophisticated shells
- Directory file system (“Folders” on Windows/Mac OS) .

Tux

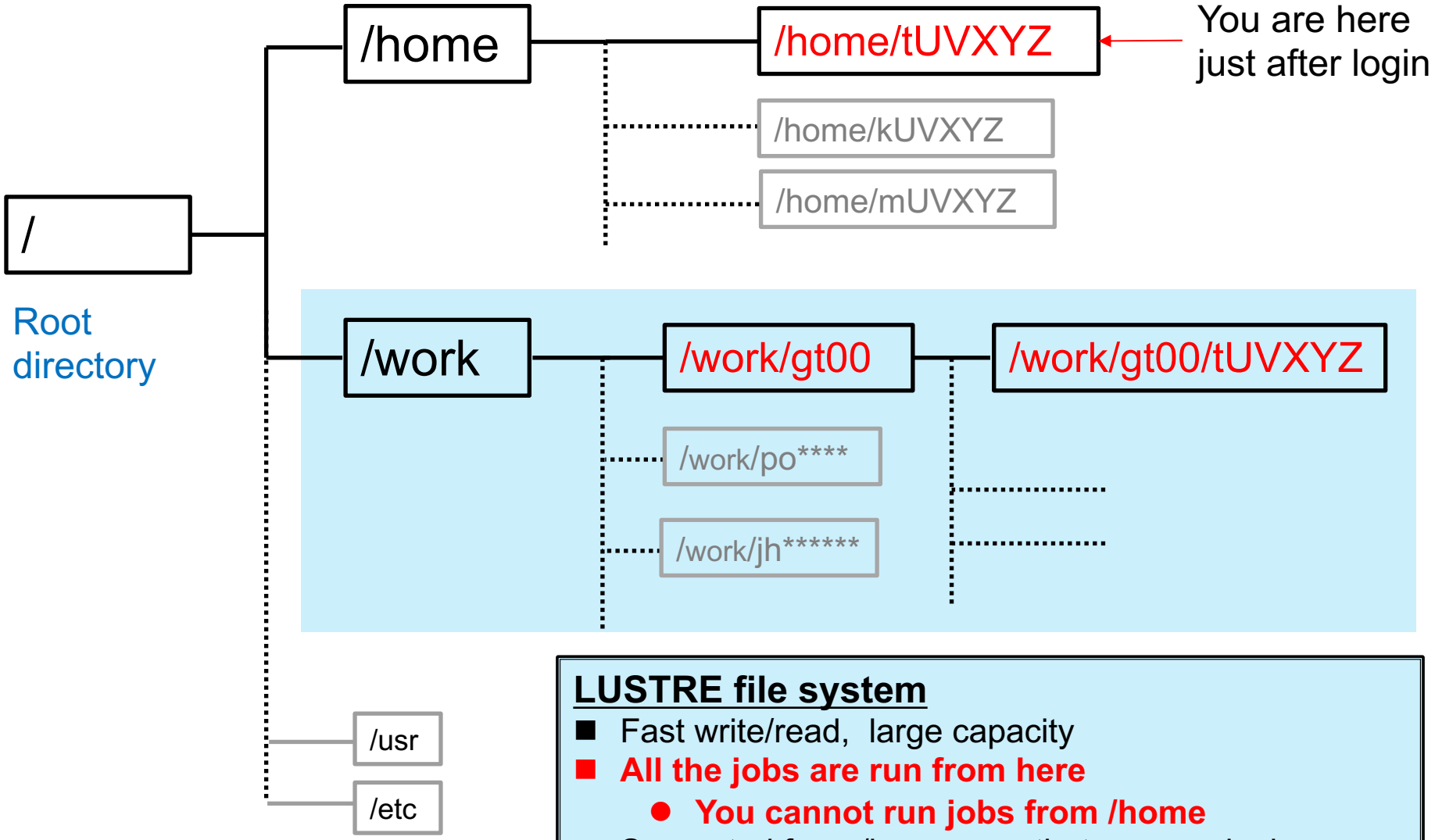


Editors

Softwares for editing files

- Emacs
  - High functionality
  - Many add-ons (based on LISP)
- Vim
  - lightweight
  - Installed as default on all Linux.
- Nano
  - Easy interface

Linux Directories on OBCX Linux



LUSTRE file system

- Fast write/read, large capacity
- **All the jobs are run from here**
  - **You cannot run jobs from /home**
- Separated from /home , so that you can login even when /work is in trouble

After you logged in…

```
$ pwd    Return

/home/tUVXYZ

$ cd /work/gt00/tUVXYZ  Return
$ pwd    Return

/work/gt00/tUVXYZ

$ cd     Return
$ pwd    Return

/home/t00XYZ
```

1. You are at “/home/tUVXYZ”
2. Move to working directory:
   “ /work/gt00/tUVXYZ”
3. You can get back by “cd”

Exercise: computing Fibonacci sequence

defined as a recurrence relation

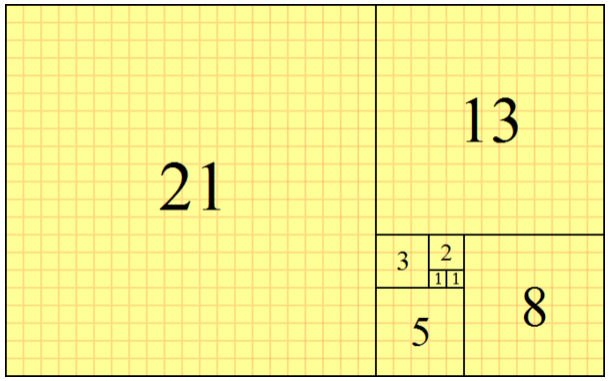
$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \geq 2)$$

1, 1, 2, 3, 5, 8, 13, 21 ….

Iterative computation is a starting point for using supercomputers



From Wikipedia
https://ja.wikipedia.org/wiki/フィボナッチ数

Fibonacci sequence, up to 92nd

1	75025	7778742049	806515533049393
1	121393	12586269025	1304969544928657
2	196418	20365011074	2111485077978050
3	317811	32951280099	3416454622906707
5	514229	53316291173	5527939700884757
8	832040	86267571272	8944394323791464
13	1346269	139583862445	14472334024676221
21	2178309	225851433717	23416728348467685
34	3524578	365435296162	37889062373143906
55	5702887	591286729879	61305790721611591
89	9227465	956722026041	99194853094755497
144	14930352	1548008755920	160500643816367088
233	24157817	2504730781961	259695496911122585
377	39088169	4052739537881	420196140727489673
610	63245986	655747031 9842	679891637638612258
987	102334155	10610209857723	1100087778366101931
1597	165580141	17167680177565	1779979416004714189
2584	267914296	27777890035288	2880067194370816120
4181	433494437	44945570212853	4660046610375530309
6765	701408733	72723460248141	7540113804746346429
10946	1134903170	117669030460994	
17711	1836311903	190392490709135	
28657	2971215073	308061521170129	
46368	4807526976	498454011879264	

Task:
Compute this sequence, and find “747031” in the result

Compose a program using text editors

First, make a directory where you work on.

```
[tUVXYZ@obcx01 ~]$ cd                  Return
[tUVXYZ@obcx01 ~]$ mkdir fibonacci     Return
[tUVXYZ@obcx01 ~]$ cd fibonacci        Return
```

Please use text editors such as Emacs, vim, and nano.

例1： Compose a Fortran code using Emacs

```
[tUVXYZ@obcx01 fibonacci]$ emacs fibonacci.f90    Return
```

例2： Compose a Python script using vim

```
[tUVXYZ@obcx01 fibonacci]$ vim fibonacci.py       Return
```

例3： Compose a C code using nano

```
[tUVXYZ@obcx01 fibonacci]$ nano fibonacci.c       Return
```

Exercise: computing Fibonacci sequence

C

Edit the code

```
[tUVXYZ@obcx01 fibonacci]$ emacs fibonacci.c
```

Return

```
#include <stdio.h>

int main(void) {
    int i;
    long a, b, tmp;

    a=1;
    b=1;
    printf("%ld\n", a);

    for (i=2; i<=92; i++) {
        tmp = b;
        b = a + b;
        a = tmp;
        printf("%ld\n", a);
    }
}
```

Exercise: computing Fibonacci sequence

Fortran

Edit a code

```
[tUVXYZ@obcx01 fibonacci]$ emacs fibonacci.f90
```

Return

```
program main

    implicit none
    integer(kind=4)  i
    integer(kind=8)  a,b,tmp

    a = 1
    b = 1

    do i=2, 92
       tmp = b
       b= a + b
       a = tmp
       print *, a
    end do

end program main
```

Exercise: computing Fibonacci sequence

Python

Edit a code

```
[tUVXYZ@obcx01 fibonacci]$ emacs fibonacci.py
```

Return

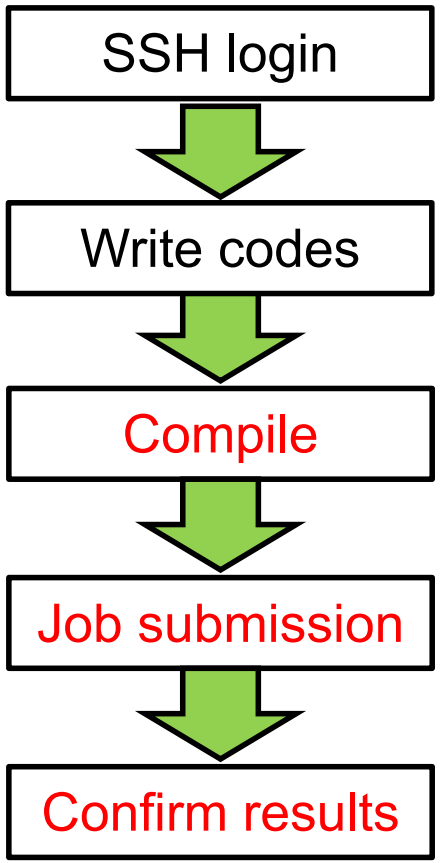
```
a, b = 1, 1
print(a)

for i in range(1,92):
    a, b = b, a+b
    print(a)
```

In python…

- Data type is auto detected (e.g. 64 bit integrer)
- Multiple variables can be updated at once.
- Compilation not necessary.

How to use supercomputers ?



Putty、Cygwin、SSH、key authentication

How to use editors (emacs, vi, nano), Linux basics

Environment module, compiler options

Writing job scripts, how to manage jobs

Environment modules (1)

We can choose compilers (etc.)

gnu compiler, intel compiler, python interpreters, …

We can choose libraries (for accelerating computations)

Fast Fourier Transform, linear algebra/simultaneous linear equations, and etc…

Show modules which is in use

```
[tUVXYZ@obcx01 ~]$ module list        Return
Currently Loaded Modulefiles:
  1) impi/2019.9.304    2) intel/2020.4.304
```

Intel MPI library Intel compiler

Clear up current settings (we want to switch to others)

```
[tUVXYZ@obcx01 ~]$ module purge       Return
[tUVXYZ@obcx01 ~]$ module list        Return
No Modulefiles Currently Loaded.
```

Displaying available environment modules

```
[tUVXYZ@obcx01 ~]$ module avail
```

Return

```
------ /home/opt/local/modulefiles/L/mpi/intel/2019.5.281/impi/2019.5.281 ------
alps/2.3.0(default)                      phdf5/1.10.5(default)
feram/0.26.04(default)                   pnetcdf/1.11.2(default)
frontflow_blue/8.1(default)              ppohBEM/0.5.0(default)
frontistr/4.5(default)                   ppohDEM_util/1.0.0(default)
modylas/1.0.4(default)                   ppohFDM/0.3.1(default)
mpi-fftw/3.3.8(default)                  ppohFEM/1.0.1(default)
netcdf-fortran-parallel/4.4.5(default)   ppohFVM/0.3.0(default)
netcdf-parallel/4.7.0(default)           pt-scotch/6.0.6(default)
openmx/3.8(default)                      revocap_coupler/2.1(default)
parmetis/4.0.3(default)                  superlu_dist/6.1.1(default)
petsc/3.11.2(default)                    xtapp/rc-150401(default)
phase/2019.01(default)

---------- /home/opt/local/modulefiles/L/compiler/intel/2019.5.281 ----------
R/3.6.0(default)                 metis/5.1.0(default)
akaikkr/cpa2002v010(default)     mt-metis/0.6.0(default)
bioconductor/3.10(default)       netcdf/4.7.0(default)
blast/2.9.0(default)             netcdf-cxx/4.3.0(default)
bwa/0.7.17(default)              netcdf-fortran/4.4.5(default)
fftw/3.3.8(default)              paraview/5.6.1(default)
gsl/2.5(default)                 povray/3.7.0.8(default)
hdf5/1.10.5(default)             ppohAT/1.0.0(default)
hdf5/1.8.21                      revocap_refiner/1.1.04(default)
impi/2019.5.281(default)         samtools/1.9(default)
intelpython/2.7                  scotch/6.0.7(default)
intelpython/3.6(default)         superlu/5.2.1(default)
mesa/19.0.6(default)             superlu_mt/3.1(default)
metis/4.0.3                      xabclib/1.03(default)

--------------------- /home/opt/local/modulefiles/L/core ---------------------
acusolve/2019.1.0(default)       intel/2018.3.222
advisor/2019.3.0.591490          intel/2019.3.199
advisor/2019.4.0.597843          intel/2019.4.243
advisor/2019.5.0.602216(default) intel/2019.5.281(default)
anaconda/2-2019.03               intel/2020.1.217
anaconda/3-2019.03(default)      itac/2019.4.036
bioperl/1.007002(default)        itac/2019.5.041(default)
bioruby/1.5.2(default)           julia/1.4.0(default)
cmake/3.0.2                      llvm/7.1.0(default)
cmake/3.14.5(default)            massivethreads/0.97
```

To see at once complex module dependencies on OBCX

```
[tUVXYZ@obcx01 ~]$ show_module
```
Return

ApplicationName	ModuleName	Node	BaseCompiler/MPI
ALPS	alps/2.3.0	login,compute	intel/2020.4.304/impi/2019.9.304
Acusolve	acusolve/2019.1.0	login,compute	-
Advisor	advisor/2019.4.0.597843	login,compute	-
Advisor	advisor/2019.5.0.602216	login,compute	-
Advisor	advisor/2020.3.0.607294	login,compute	-
Advisor	advisor/2019.3.0.591490	login,compute	-
AkaiKKR	akaikkr/cpa2002v010	login,compute	intel/2020.4.304
Anaconda	anaconda/2-2019.03	login,compute	-
Anaconda	anaconda/3-2019.03	login,compute	-
Arm DDT	ddt/19.1	compute	-
Arm DDT	ddt/20.2.1	compute	-
Arm DDT	ddt/20.0.2	compute	-
BLAST	blast/2.11.0	login,compute	intel/2020.4.304
BWA	bwa/0.7.17	login,compute	intel/2020.4.304
BioPerl	bioperl/1.007002	login,compute	-
BioRuby	bioruby/1.5.2	login,compute	-
Bioconductor	bioconductor/3.10	login,compute	intel/2020.4.304
CMake	cmake/3.0.2	login,compute	-
CMake	cmake/3.14.5	login,compute	-
CP2K	cp2k/v8.1	login,compute	intel/2020.4.304/impi/2019.9.304
Devtoolset	devtoolset/7	login,compute	-
FFTW	fftw/3.3.8	login,compute	intel/2020.4.304
FFTW	mpi-fftw/3.3.8	login,compute	intel/2020.4.304/impi/2019.9.304
FeRAM	feram/0.26.04	login,compute	intel/2020.4.304/impi/2019.9.304
GATK	gatk/4.1.2.0	login,compute	-
GCC	gcc/4.8.5	login,compute	-
GCC	gcc/7.5.0	login,compute	-
GNU Octave	octave/6.1.0	login,compute	-
GNU Scientific Library	gsl/2.6	login,compute	intel/2020.4.304
GROMACS	gromacs/2020.5	login,compute	intel/2020.4.304/impi/2019.9.304
Go	go/1.12.6	login,compute	-
HDF5	hdf5/1.12.0	login,compute	intel/2020.4.304
HDF5	hdf5/1.8.22	login,compute	intel/2020.4.304
HDF5	phdf5/1.12.0	login,compute	intel/2020.4.304/impi/2019.9.304
HDF5	phdf5/1.8.22	login,compute	intel/2020.4.304/impi/2019.9.304
HyperWorks	hyperworks/2019.1.0	login,compute	-
IASP91	iasp91/default	compute	-
Inspector	inspector/2019.5.0.602103	login,compute	-
Inspector	inspector/2019.3.0.591484	login,compute	-

BaseCompiler/MPI needs to be loaded before loading each module

Compilation & setting up your job

By environment modules

$ module [option] *args*

option	
avail	show available environment modules
list	show loaded environment modules
load	load specified environment module(s)
unload	Unload specified environment module(s)
switch	load and unload
purge	unload all the current modules

If you use Python3:

$ module load python/3.7.3

Environment modules (2)

Load a previous version of Intel compiler

```
[tUVXYZ@obcx01 ~]$ module load  intel/2020.1.217 Return
[tUVXYZ@obcx04 ~]$ module list  Return
Currently Loaded Modulefiles:
  1) impi/2019.7.217    2) intel/2020.1.217
```

An appropriate Intel MPI library is also loaded

Load newer Python3 (system default is Python 2.7.5)

```
[tUVXYZ@obcx01 ~]$ module load python/3.7.3 Return
[tUVXYZ@obcx01 ~]$ module list  Return
Currently Loaded Modulefiles:
  1) impi/2019.7.217    2) intel/2020.1.217   3) python/3.7.3
```

Compile the program

If you use C or Fortran

→ Compile the code to generate an executable (binary) file.

C

```
[tUVXYZ@obcx01 fibonacci]$ icc fibonacci.c -o fibonacci.out
```

Return

program executable

Fortran

```
[tUVXYZ@obcx01 fibonacci]$ ifort fibonacci.f90 -o fibonacci.out
```

Return

Compilers

Intel compiler / GNU compilers are available

Source code type		lang	command
Serial codes	Intel	C	icc
		Fortran	ifort
	GNU	C	gcc
		Fortran	gfortran
MPI parallel codes	Intel	C	mpiicc
		Fortran	mpiifort
	GNU	C	mpicc
		Fortran	mpif77 or mpif90

Ex 1) Compiling a C code by Intel compiler
$ icc [option] “source.c” –o “exe.out”
Ex 2) Compiling a Fortran90/95 code by GNU compiler
$ gfortran [option] -free-line-length-none “source.f90” –o “exe.out”

-free-line-length-none : specifying Fortran free form

Compiler option examples

	Language	Intel option	GNU option	
parallelization	C/Fortran	-qopenmp	-fopenmp	Enable OpenMP
debugging	C/Fortran	-g		Embed debugging information into the executables
		-Wall		Display all warnings (potential bugs)
		-traceback	-fbacktrace	Trace where an error happens in running the executable
	Fortran	-check bounds	-fbounds-check	Detect whether a variable is within bounds
optimization	C/Fortran	-O0、-O1、-O2、-O3		Optimization (larger values are more aggressive)
		-xHost	-march=native	CPU-architechture-specific optimization

How to use supercomputer

login node = entrance where you can run commands

```
[tut138@obcx02~]$ ls -l
drwxr-x--- 2 shiba group 10 1 Apr 13:00 test.out
[tut138@obcx02~]$ ./test.out
Hello world
[tut138@obcx02~]$ qsub a.sh
```

Your terminal

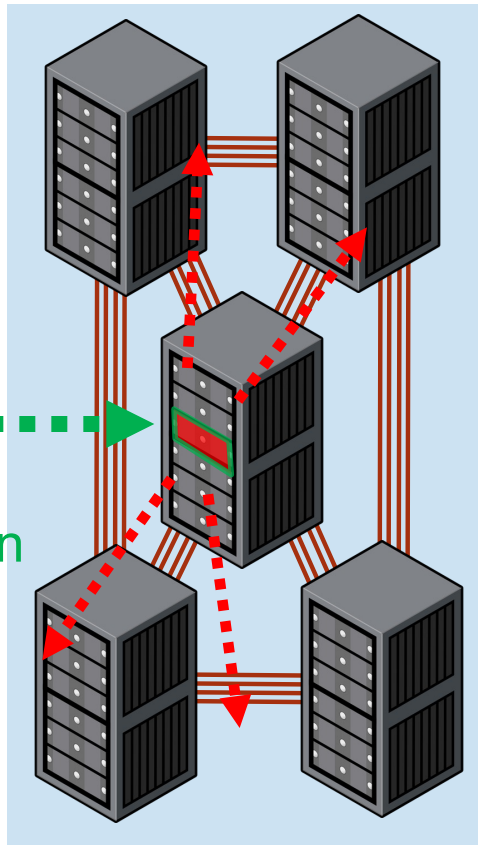
1. Login

SSH

Login nodes

2. Job submission

3. Program execution



Compute nodes = “main body”

You need to prepare your SSH key to establish connection

Job script = work instructions for supercomputers

Write a script to let your program run on compute nodes.
It is forbidden to run your program (binary) on the login node after compilation.
— login nodes are shared with other users, so should not be under load

C, Fortran (fibonacci.sh)

```
#!/bin/bash
#PJM  -L  rscgrp=tutorial     [resource group]
#PJM  -L  node=1              [number of nodes]
#PJM  -L elapse=0:01:00       [ maximum time]
#PJM  -g  gt00                [group name]
#PJM  -N  fibonacci           [job name]
#PJM  -o   stdout.txt         [stdout file]
#PJM  -j                      [merge error into stdout]

module purge
module load intel/2020.1.217
./fibonacci.out
```

Python (fibonacci.sh)

```
#!/bin/bash
#PJM  -L  rscgrp=tutorial
#PJM  -L  node=1
#PJM  -L  elapse=0:01:00
#PJM  -g  gt00
#PJM  -N  fibonacci
#PJM  -o  stdout.txt
#PJM  -j

module load python/3.7.3
python ./fibonacci.py
```

We have already arranged your job scripts

Copy a job script we have prepared in advance.

```
[tUVXYZ@obcx01 ~]$ cd
[tUVXYZ@obcx01 ~]$ cd fibonacci
```

Return

Return

C, Fortran

```
[tUVXYZ@obcx01 fibonacci]$ cp /work/gt00/share/z30122/fibo_c/fibonacci.sh .
```

Return

Python

```
[tUVXYZ@obcx01 fibonacci]$ cp /work/gt00/share/z30122/fibo_python/fibonacci.sh .
```

Return

In case you could not edit the program in time

You may copy the one we have prepared in advance.

C

```
[tUVXYZ@obcx01 fibonacci]$ cp /work/gt00/share/z30122/fibo_c/fibonacci.c .
```

Return

Fortran

```
[tUVXYZ@obcx01 fibonacci]$ cp /work/gt00/share/z30122/fibo_fortran/fibonacci.f90 .
```

Return

Python

```
[tUVXYZ@obcx01 fibonacci]$ cp /work/gt00/share/z30122/fibo_python/fibonacci.py .
```

Return

Running your jobs on OBCX

You cannot run your code from /home directory on OBCX

→ Copy your program into your /work directory

```
[tUVXYZ@obcx01 ~]$ cd                                          Return
[tUVXYZ@obcx01 ~]$ ls                                          Return
fibonacci
[tUVXYZ@obcx01 ~]$ cp -r fibonacci  /work/gt00/tUVXYZ/         Return

[tUVXYZ@obcx01 ~]$ cd  /work/gt00/tUVXYZ/fibonacci             Return
```

Submit the job script, then your program will be run on the compute nodes.

```
[tUVXYZ@obcx01 fibonacci]$ pjsub fibonacci.sh    Return
```

Check you job status & display the results

Confirm your jobs that are now running.
（Perhaps nothing will be displayed if they terminate at once).

```
[tUVXYZ@obcx01 fibonacci]$ pjstat
```

Display the results.

```
[tUVXYZ@obcx01 fibonacci]$ more stdout.txt
```

[Return will let you go downwards]

Search 747031 in the results.

```
[tUVXYZ@obcx01 fibonacci]$ grep 747031 stdout.txt
```

How to manage your job – scheduler commands

command	role	how to use
pjsub	submit a job	$pjsub “script.sh”
pjdel	delete a job	$pjdel “job ID”
pjstat	query job status	$pjstat

Caution) these commands are specific to Fujitsu Supercomputers

pjstat options

- -H : confirm jobs that have finished
- --rsc -b : query job “congestion” status in each resource group
- --rsc -x : query the number of nodes (resources) a user can request in a job
- --nodeuse: Check the current resource usage over the whole OBCX system

On your trial account for this workshop

You can use this account for one month.
"lecture" resource group is available after the workshop.
"tutorial" is available only today (13-17 pm).

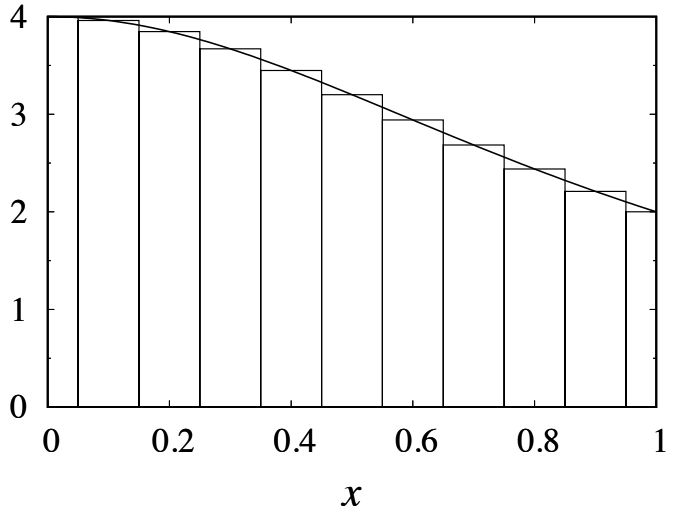
Job script
fibonacci.sh

```
#!/bin/bash
#PJM  -L  rscgrp=lecture        [resource group]
#PJM  -L  node=1
#PJM  -L elapse=0:01:00
#PJM  -g  gt00
#PJM  -N  fibonacci
#PJM  -o   stdout.txt
#PJM  -j

module purge
module load intel/2020.1.217
./fibonacci.out
```

Exercise: parallel computation of π

$$I = \int_0^1 \frac{4}{1+x^2} dx = \pi$$



quadrature by parts

↓

sub-billion meshes

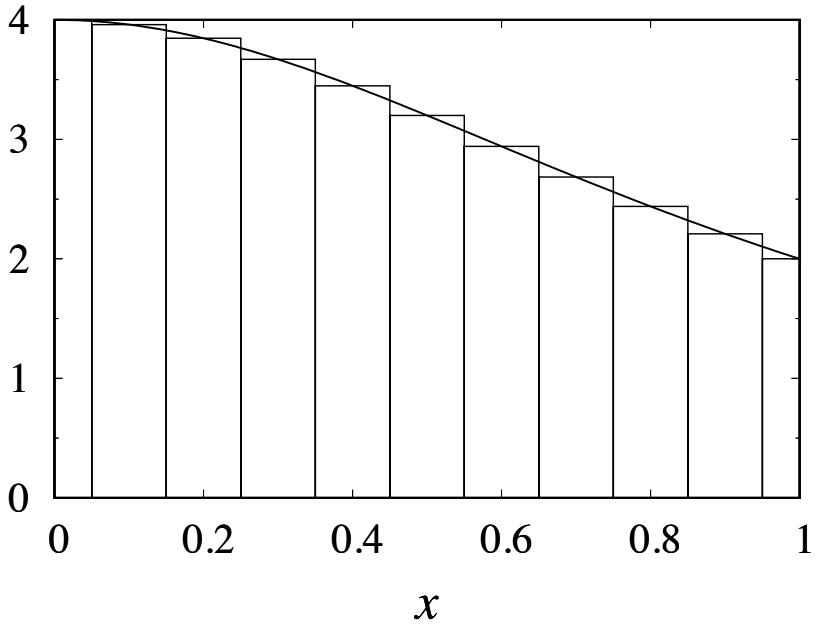
We will see parallelization speedup.

We prepared 3 sample programs: C, Fortran, Python

Exercise: measure execution time for various degree of parallelism
(understanding of MPI is not our focus now, just run !)

Example problem: Compute π by piecewise quadrature

$$I = \int_0^1 \frac{4}{1+x^2} dx$$



change of variables $x = \tan\theta$

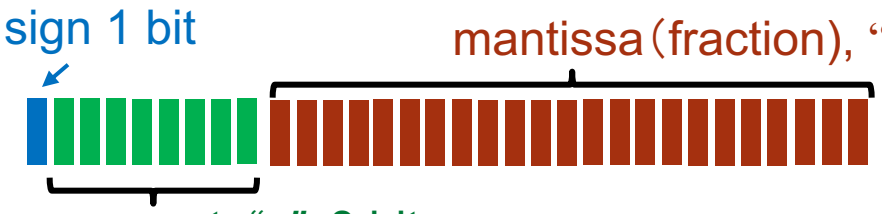
$$dx = \frac{d\theta}{\cos^2\theta}$$

$$\Rightarrow I = \int_0^{\pi/4} \frac{4}{1+\tan^2\theta} \frac{d\theta}{\cos^2\theta} = 4\int_0^{\pi/4} d\theta = \pi$$

Arithmetic operations on computers - floating point

On (Neumann-type) computers, all numbers are represented by bit (0 or 1)

single floating point = representing a real number with 32 bits



$$\log_{10}(2^{23}) \sim 7 \text{ dicimal digits}$$

$$(-1)^{\text{sign}}(1.b_1 b_2 \ldots b_{23})_2 \times 2^{e-127}$$

Variable type FORTRAN: real(kind=4)
C, C++: float
Python: none

Double floating point = representing a real number with 64 bits

$$\log_{10}(2^{52}) \sim 15 \text{ dicimal digits}$$

$$(-1)^{\text{sign}}(1.b_1 b_2 \ldots b_{52})_2 \times 2^{e-1023}$$

Variable type FORTRAN: real(kind=8)
C, C++: double
Python: double precision as default

C source codes

pi.c [serial]

```
#include <stdlib.h>
#include <stdio.h>

int main(void) {
  int i;
  int ndiv = 10000000;


  double width = 1.0 / (double)ndiv;
  printf("width = %.15f\n", width);
  double sum = 0.0;
  double x = 0.0;


  for (i=0; i<ndiv; i++) {
    x = (i+0.5)*width;
    sum += width * 4.0 / (1.0 + x*x);
  }


  printf("PI = %.15f\n", sum);
  return 0;
}
```

pi_mpi.c [parallel]

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
   int ndiv = 5600000000;
   int ierr,  myrank, nprocs;
   int ndiv_local, i;
   double x, width, sum, total_sum;
   double t1, t2;

   width = 1.0 / (double)ndiv;

   ierr = MPI_Init(&argc, &argv);
   ierr = MPI_Comm_rank(MPI_COMM_WORLD,
                        &myrank);
  ierr = MPI_Comm_size(MPI_COMM_WORLD,
                        &nprocs);

  t1 = MPI_Wtime();

  sum = 0.0;
  ndiv_local = ndiv / nprocs;
  for (i = myrank*ndiv_local;
        i < (myrank+1)*ndiv_local; i++) {
        x = (i + 0.5) * width;
     sum = sum + width * 4.0 / (1.0 + x*x);
  }

  MPI_Reduce(&sum, &total_sum, 1, MPI_DOUBLE,
                   MPI_SUM, 0, MPI_COMM_WORLD);

  t2 = MPI_Wtime();

  if (myrank == 0) {
    printf("PI(MPI) = %.18f\n", total_sum);
    printf("Number of cores utilized = %d\n", nprocs);
    printf("Execution time = %.8f (sec.)\n", t2 - t1);
  }

  ierr = MPI_Finalize();

  return 0;
}
```

compile

```
[tUVXYZ@obcx01 calc_pi_mpi]$ mpiicc pi_mpi.c -o pi_mpi.out
```

Fortran source codes

pi.f90 [serial]

```
program main

  implicit none

  integer i, ndiv
  real(kind=8) unit, width, sum, x

  ndiv = 560000000
  width = 1.0 / dble(ndiv)

  print *, "width"
  print '(F18.14)', width

  sum = 0.0d0
  x = 0.0d0

  do i = 1, ndiv
    x = ( dble(i-1) + 0.5) * width
    sum += width * 4.0 / (1.0 + x*x)
  end do

  print *, "sum"
  print '(F18.14)', sum

end program main
```

pi_mpi.f90 [parallel]

"_8" indicates 8-byte integer (larger # of digits)

```
program main

  Use mpi
  implicit none

  integer ndiv, ierr, myrank, nprocs
  integer ndiv_local, i;
  real(kind=8) unit, width, sum, x, total_sum
  real(kind=8) t1, t2

  ndiv = 5600000000_8
  width = 1.0 / dble(ndiv)

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD,
                     myrank, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD,
                     nprocs, ierr)

  sum = 0.0d0
  ndiv_local = ndiv / nprocs

  t1 = MPI_Wtime()

  do i=myrank*ndiv_local+1, (myrank+1)*ndiv_local
    x = ( dble(i-1) + 0.5)*width
    sum = sum + width * 4.0 / (1.0 + x*x)
  end do

  call MPI_REDUCE(sum, total_sum, 1, MPI_REAL8,
MPI_SUM, 0, MPI_COMM_WORLD, ierr)

  t2 = MPI_Wtime()

  if (myrank .eq. 0) then
    print "('PI(MPI) = ', F18.16)", total_sum
    print "('Number of cores utilized = ', i0)", nprocs
    print "('Execution time = ', F12.8)", t2 - t1
  endif

  call MPI_FINALIZE(ierr)

end program main
```

compile

```
[tUVXYZ@obcx01 calc_pi_fortran]$ mpiifort pi_mpi.f90 -o pi_mpi.out
```

Python source codes

pi.py [serial]

```
program main
  implicit none

  integer i, ndiv
  real(kind=8) unit, width, sum, x

  ndiv = 560000000
  width = 1.0 / dble(ndiv)

  print *, "width"
  print '(F18.14)', width

  sum = 0.0
  x = 0.0

  do i = 1, ndiv
     x = ( dble(i-1) + 0.5) * width
     sum = sum + width * 4.0 / (1.0 + x*x)
  end do

  print *, "sum"
  print '(F18.14)', sum

end program main
```

pi_mpi.py [parallel]

```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
myrank = comm.Get_rank();
nprocs = comm.Get_size();

ndiv = 560000000
width = 1.0/ndiv

t1 = MPI.Wtime()

sum = numpy.zeros(1)
total_sum = numpy.zeros(1)
ndiv_local = ndiv // nprocs

for i in range (myrank*ndiv_local, (myrank+1)*ndiv_local):
    x = width * (i+0.5)
    sum[0] = sum[0] + width * 4.0 / (1.0 + x*x)

comm.Reduce([sum, MPI.DOUBLE], total_sum, op=MPI.SUM, root=0)

t2 = MPI.Wtime()

if comm.rank == 0:
    print("PI(MPI) = ", total_sum[0])
    print("Number of cores utilized = ", nprocs)
    print("Exectution time = ", t2 - t1, " (sec.)"
```

Copy the codes into your own “/work” directory

```
[tUVXYZ@obcx01 ~]$ cd /work/gt00/tUVXYZ
[tUVXYZ@obcx01 tUVXYZ]$ mkdir calc_pi_mpi
[tUVXYZ@obcx01 tUVXYZ]$ cd calc_pi_mpi
[tUVXYZ@obcx01 calc_pi_mpi]$ pwd
/work/gt00/tUVXYZ/calc_pi_mpi
```

Fortran

```
$ cp /work/gt00/share/z30122/pi_fortran_mpi/* .
```

asterisk = wild card (everything)

C

```
$ cp /work/gt00/share/z30122/pi_c_mpi/* .
```

Python

```
$ cp /work/gt00/share/z30122/pi_python_mpi/* .
```

Compile parallel programs

For C & Fortran, we need to compile the source code.

C

```
[tUVXYZ@obcx01 calc_pi_mpi]$ mpiicc pi_mpi.c -o pi_mpi.out
```

Fortran

```
[tUVXYZ@obcx01 calc_pi_mpi]$ mpiifort pi_mpi.f90 -o pi_mpi.out
```

For Python users

You need to set up in advance

```
[tUVXYZ@obcx01 ~]$ cd /work/gt00/tUVXYZ/calc_pi_mpi
[tUVXYZ@obcx01 calc_pi_mpi]$ emacs setenv.sh   (or vim setenv.sh)
```

Change the following:

```
export PYTHONUSERBASE=/work/gt00/tUVXYZ/.local
```

⇩

your own user ID

Run setup.sh (install numpy and mpi4py)

```
[tUVXYZ@obcx01 ~]$ ./setup.sh
```

Submit parallel jobs

We prepared job scripts with various degrees of parallelism, to see performance increase.

```
[tUVXYZ@obcx01 calc_pi_mpi]$ pjsub run_n1c0004.sh
[tUVXYZ@obcx01 calc_pi_mpi]$ pjsub run_n1c0028.sh
[tUVXYZ@obcx01 calc_pi_mpi]$ pjsub run_n1c0056.sh
[tUVXYZ@obcx01 calc_pi_mpi]$ pjsub run_n2c0112.sh
[tUVXYZ@obcx01 calc_pi_mpi]$ pjsub run_n4c0224.sh
[tUVXYZ@obcx01 calc_pi_mpi]$ pjsub run_n8c0448.sh
```

- 1 node, 4 cores
- 1 node, 28 cores
- 1 node, 56 cores
- 2 nodes, 112 cores
- 4 nodes, 224 cores
- 8 nodes, 448 cores

Confirm the job status

Check your jobs and wait until all your jobs finish.

```
[tUVXYZ@obcx01 calc_pi_mpi]$ pjstat
```

You will get the results after the jobs finish.

```
[tUVXYZ@obcx01 calc_pi_mpi]$ ls
```

pi_mpi.c	result_n2c0112.txt	run_n1c0056.sh
pi_mpi.out	result_n4c0224.txt	run_n2c0112.sh
result_n1c0004.txt	result_n4c0448.txt	run_n4c0224.sh
result_n1c0028.txt	run_n1c0004.sh	run_n8c0448.sh
result_n1c0056.txt	run_n1c0028.sh	

“result_n*c****.txt” are output files wherein the results are stored.

Confirm the results and compare execution times

You will find elapse times of your jobs in the output files.
Let's see the beginning part of each output file.

```
[tUVXYZ@obcx01 calc_pi_mpi]$ head result*.txt
```

We find that the execution time is shorter with smaller number of nodes.

```
==> result_n1c0004.txt <==
PI(MPI) = 3.141592653589913464
Number of cores utilized = 4
Execution time = **.******** (sec.)

==> result_n1c0028.txt <==
PI(MPI) = 3.141592653589770912
Number of cores utilized = 28
Execution time = **.******** (sec.)

==> result_n1c0056.txt <==
PI(MPI) = 3.141592653589800221
Number of cores utilized = 56
Execution time = **.******** (sec.)

==> result_n2c0112.txt <==
PI(MPI) = 3.141592653589794892
Number of cores utilized = 112
Execution time = **.******** (sec.)

==> result_n4c0224.txt <==
PI(MPI) = 3.141592653589791340
Number of cores utilized = 224
Execution time = **.******** (sec.)

==> result_n8c0448.txt <==
PI(MPI) = 3.141592653589797557
Number of cores utilized = 448
Execution time = **.******** (sec.)
```

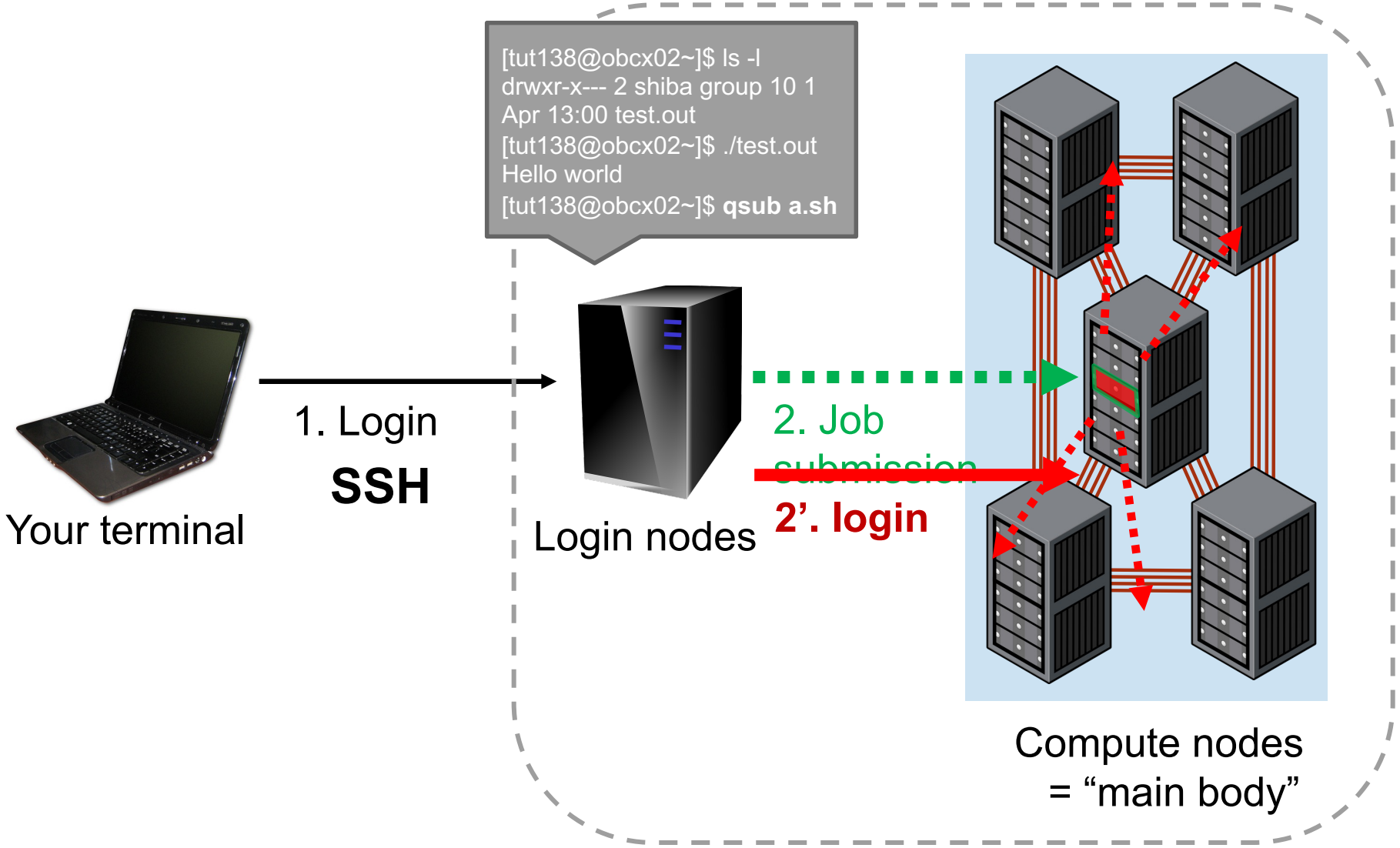
Reference Information: Resource Groups on OBCX

Allowed # of nodes and maximum elapse time for each resource group

resource group	# nodes	Max. elapse time	explanation
debug	1～16	30 min	For debugging
short	1～8	8 hours	For short jobs
regular	1～128	48 hours	For regular jobs
	129～256	24 hours	
interactive	1	2 hours	Interactive job (explained later)）
	2～8	10 min	
tutorial	1～8	15 min	For tutorial workshop
lecture	1～8	15 min	For tutorial workshop (after the event)

- Please specify “tutorial” during the workshop (today).
- After the workshop, please specify “lecture” (until a month from now)
- Your trial account is not privileged for use other resource groups.

If you want to “log in” to the computation node…



If you want to “log in” to the computation node…

Interactive job

We can run a job on the compute nodes as if we were working in the bash shell on them.

```
[tUVXYZ@obcx04 tUVXYZ]$ pjsub --interact -g gt00 -L rscgrp=interactive,node=1
[INFO] PJM 0000 pjsub Job 517079 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 517079 started.
[tUVXYZ@cx0065 tUVXYZ]$
```

You are “virtually” logging into a compute node (cx0065) via the login node (node04).

Compute nodes are not shared with other users but are for your exclusive use.
You may run any jobs you like in the interactive job.

Finally: transfer your simulation data to your laptop

Log out from supercomputer

```
[tUVXYZ@obcx01 ****]$ exit
Mac-mini:~ shiba$
```

Transfer a file using “sftp”

```
Mac-mini:~ shiba$ sftp tUVXYZ@obcx.cc.u-tokyo.ac.jp
sftp > cd /work/gt00/tUVXYZ/fibo
sftp > get fibonacci.txt
Fetching /work/00/gt00/tUVXYZ/fibonacci.txt fibonacci.txt
/work/00/gt00/tUVXYZ/fibonacci.txt      100%  171    11.9KB/s   00:00

sftp > exit
Mac-mini:~ shiba$ ls
fibonacci.txt
```

Appendix: how to subscribe to for using OBCX

- General use course
  - Apply as a group of multiple users.
  - Group head = a staff at a Japanese university or public research institute

⚒ Estimate in advance the amount of resources you need

System	Annual fee (incl. tax)	Available # of nodes	Resource amount (token)	Disk amount @ /work
Oakbridge-CX OBCX	**100,000 JPY/Year** (1set)	Max. of 256 nodes	**8,640 node hours** (1 nodes×360 days)	4TB

- the fee is appropriated for the electricity fees, and thus is subject to change.

Appendix: cost required for using Wisteria/BDEC-01

- 1 set = 60,000 JPY / year (subject to change)

	Max. # of nodes	token amount	Disk amount
Wisteria-Odyssey	2,304 nodes = 110,592 cores	8640 (=24x360) / (node hours)	2TB / set
Wisteria-Aquarius	8 nodes = 64 GPUs	2880 (=24x120) / (GPU hours)	2TB / set

other services: fixed node usage, occupied GPUs, node connected to the external network

Details are on our web - https://www.cc.u-tokyo.ac.jp

If you want to use ITC supercomputers *free of charge.*

!!! Annual call for proposal of projects.

- Recommendation Program for Young Researchers and Women Researchers
  - those who are aged <40 or female
  - called twice a year
- The Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (JHPCN)
  - group proposal for joint research with supercomputing centers in Japan.
  - called once a year (in January)
- HPCI project
  - called once a year. Proposal review is managed by RIST