第57回お試しアカウント付き講習会
「超初心者利用」
2023年12月11日

名古屋大学 情報基盤センター
特任助教 河合 直聡

名古屋大学
NAGOYA UNIVERSITY

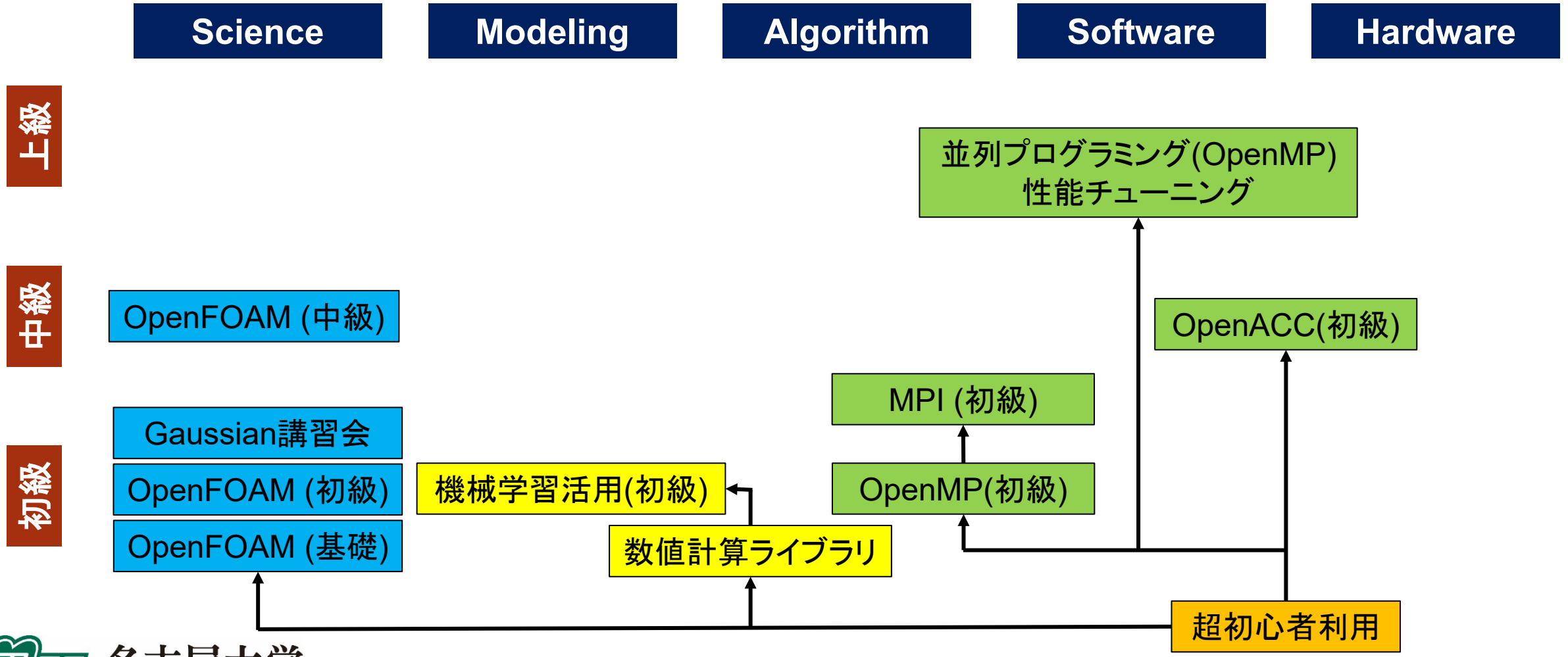
2023/12/11 v2.0

本講習会の狙い

- これからスーパーコンピュータ（スパコン）の利用を開始される方のための講習会
  - 研究等でスパコンを利用予定
  - 本学または他学のスパコン講習会を受講予定
- ログインからプログラム実行、結果の確認まで
- 以下の単語（内容）について紹介
  - ssh(ログイン)
  - Linuxコマンド(ファイル操作、コンパイル)
  - ジョブスケジューラー（プログラム実行）
- 受講者の皆様の需要にこたえられるかどうかは別の講習会で
  - 高速化したい
  - 大規模な問題を扱いたい
  - 試行回数を増やしたい

スパコン向け最適化に関する講習会 @ 名大ITC

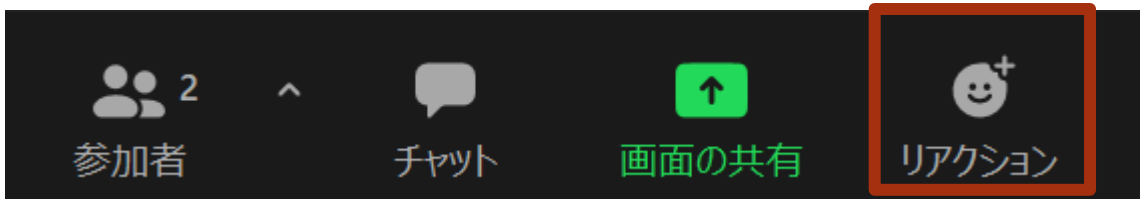
目的に応じて受講をご検討ください。



講習会を始める前に

■ 質問は随時受付とします。
- Zoomの挙手機能を利用ください。
- 短く答えられる物は
  その場で対応します。
- 操作方法など、時間を必要とする
  質問は後程の実習時間で対応します。

1. Zoomメニュー中の「リアクション」
   をクリック



2. ポップアップで表示された
   「手を挙げる」をクリック

- スーパーコンピュータとは？
- スーパーコンピュータで何ができるのか？
- スーパーコンピュータの使い方
  - ✓ ログイン
  - ✓ プログラム作成
  - ✓ コンパイル
  - ✓ 実行
  - ✓ 結果の確認

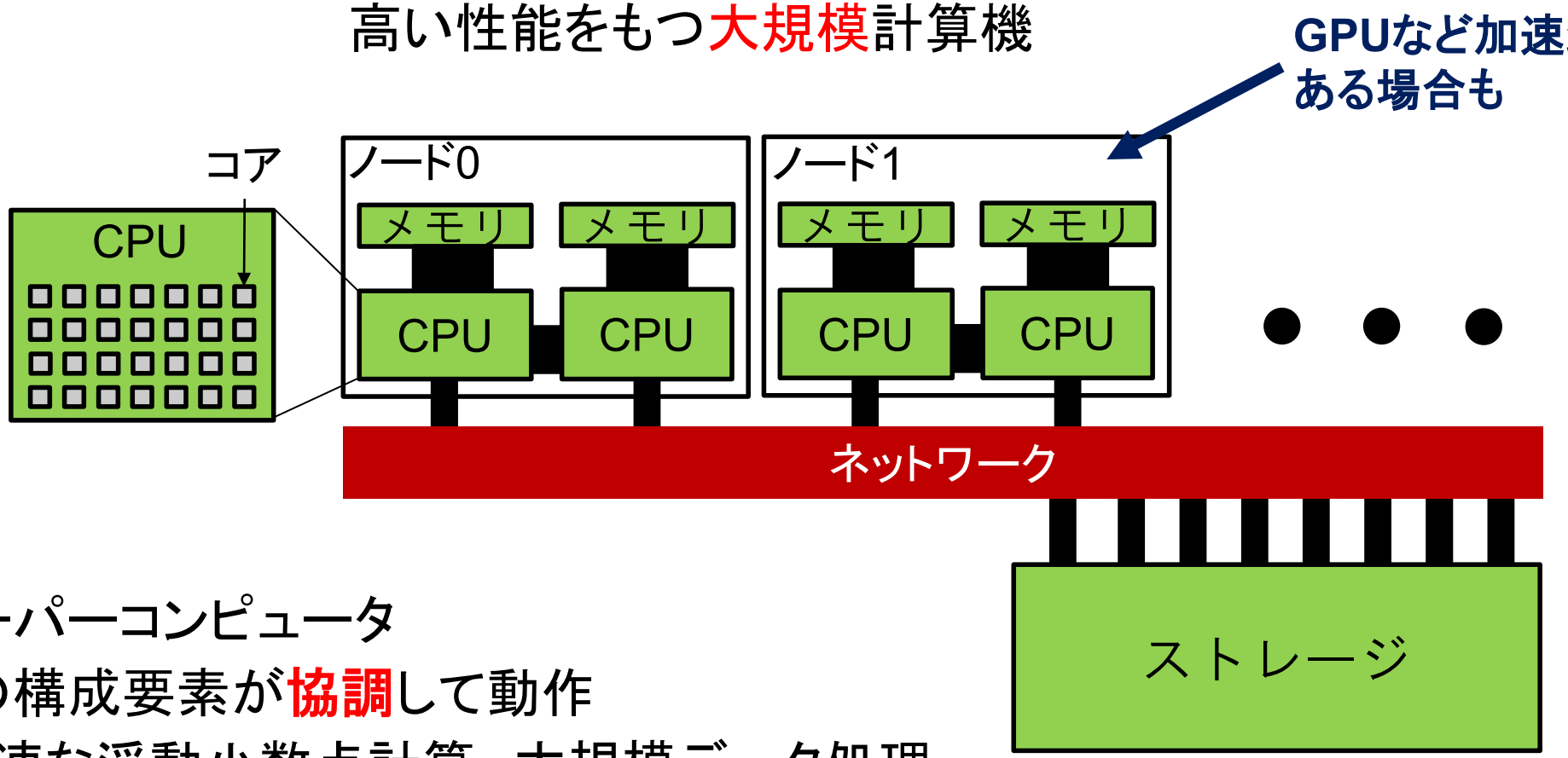
名大スーパーコンピュータ 不老 TypeⅠ、Ⅱ

TypeⅠ

TypeⅡ

スーパーコンピュータ、略してスパコン

高い性能をもつ大規模計算機



ここ20年のスーパーコンピュータ

＝ **多数**の構成要素が**協調**して動作

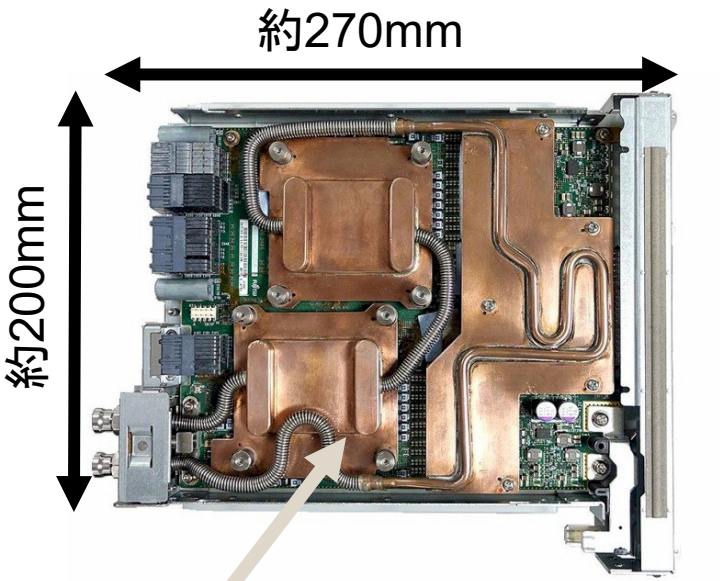
→ 高速な浮動小数点計算、大規模データ処理

高性能なノード、高速なネットワーク、大容量かつ高速なストレージ

性能指標 ～ 不老 Type Ⅰ を例に (1)

多数のコア、高いメモリ転送速度

		Core i9 11900	不老 Type Ⅰ
CPU	コア数	8	48
	周波数 [GHz]	2.5	2.2
Memory	容量 [GByte]	128	32
	転送速度 [GByte/秒]	50	1,024



出展：https://www.nextplatform.com/2021/06/22/u-s-institutions-put-fujitsu-a64fx-through-the-paces/

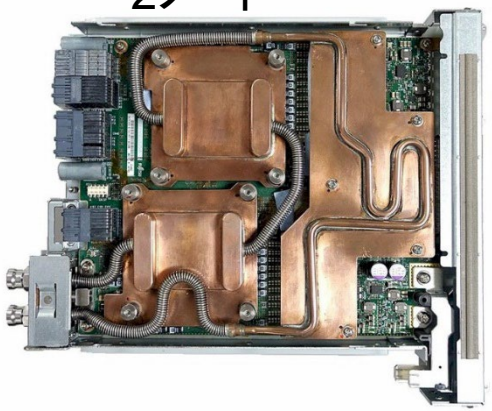
ボードには水冷
（とにかく熱い）！

大きいことはいいこと（もある）

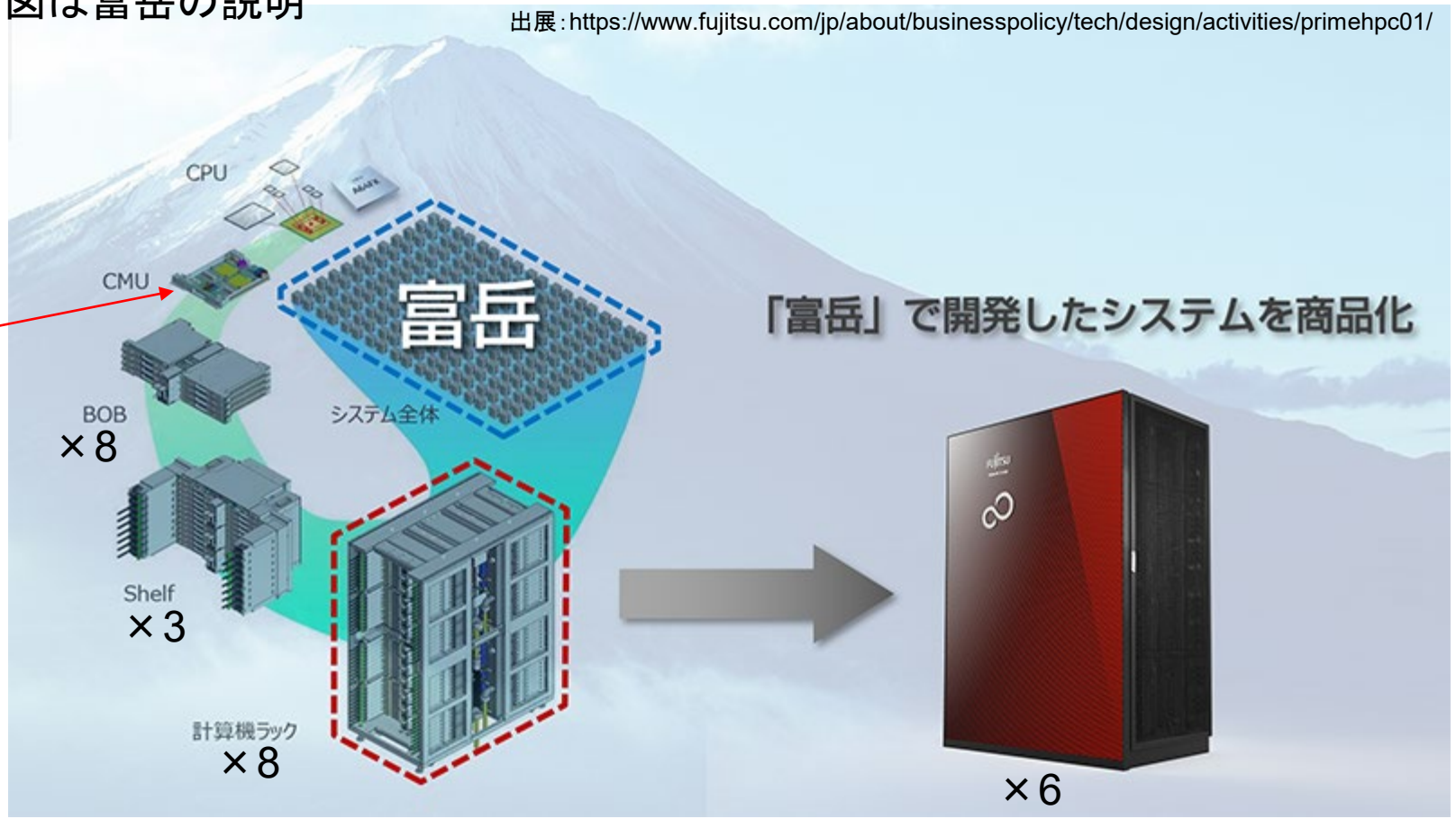
たくさんのパソコンを同時に連携して使用するようなもの

図は富岳の説明

2ノード



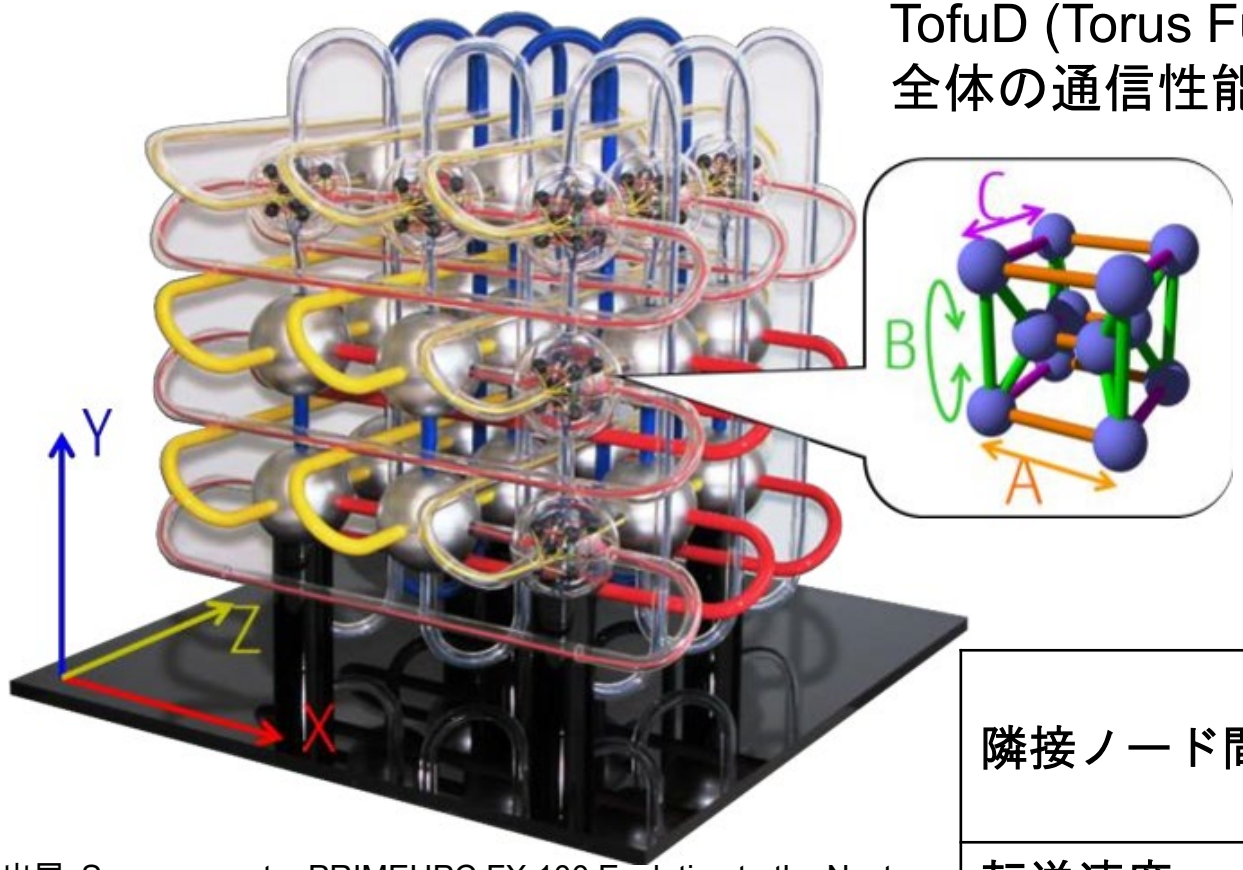
出展：https://www.fujitsu.com/jp/about/businesspolicy/tech/design/activities/primehpc01/



全部で2,304ノード

性能指標 ～ 不老Type Ⅰを例に (2)

多数のノード数を全体で繋ぐためにネットワークは非常に複雑かつ高速



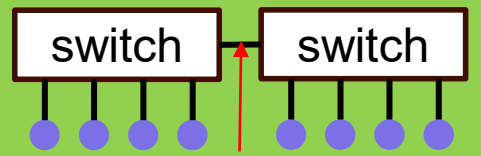
出展：Supercomputer PRIMEHPC FX 100 Evolution to the Next Generation Next Generation Technical Computing Unit

TofuD (Torus Fusion-high Density)
全体の通信性能とスケールを両立させたネットワークトポロジー

全ノードで通信してもそれほど性能が落ちない。

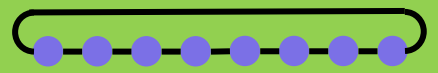
簡単な接続例と問題点

通信性能、スケール性ともに低



ここがネック

高スケール性、低信性能



隣接ノード間	携帯 4G	無線 (Wifi 11ac)	有線 (Gigabit Ether)	Tofu	Infiniband EDR (Type Ⅱ)
転送速度 [Gbit/秒]	0.1 ~ 1.0	6.9	1	54.4*1	200

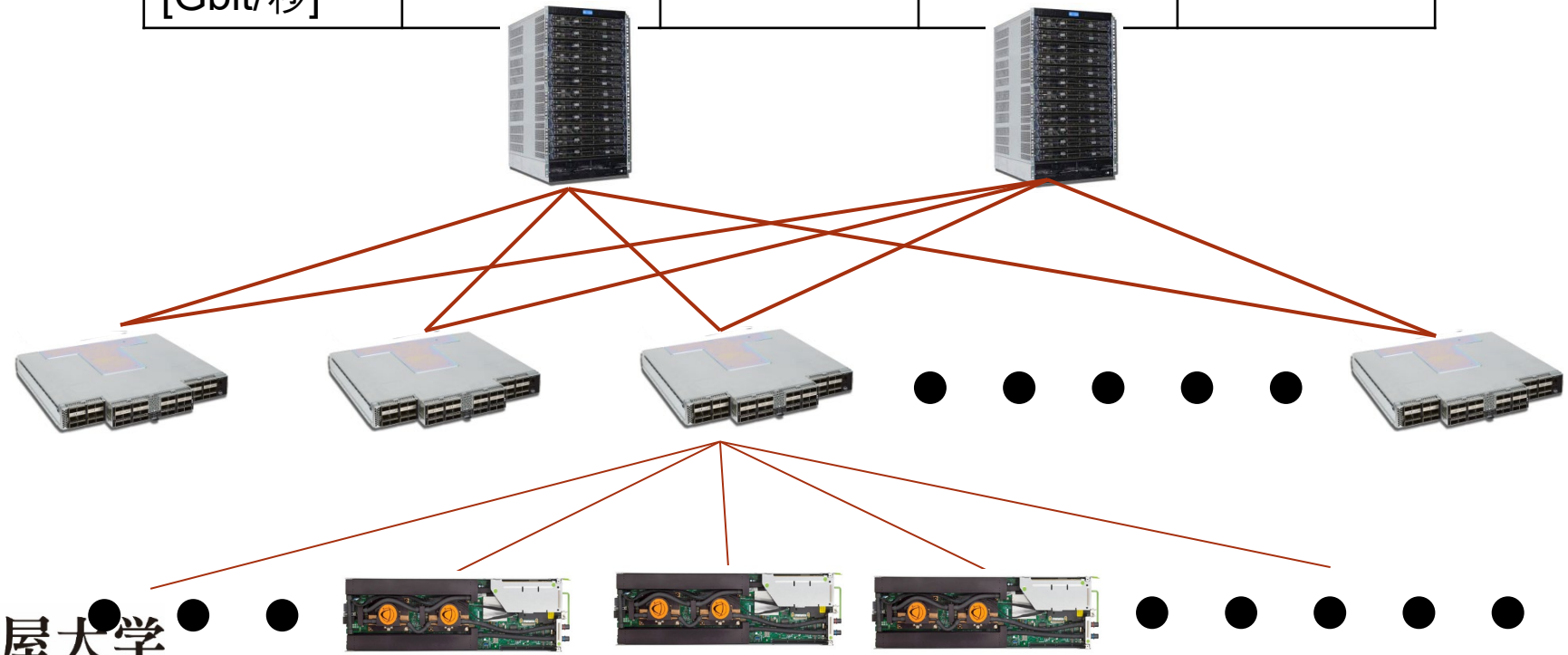
*1 隣接ノード6台と同時に通信しても性能は落ちない。

名古屋大学 NAGOYA UNIVERSITY

別のスパコン(OBCX)のネットワーク構成

こちの方が高性能だが、低スケール性能かつ高額

	携帯 ４G	無線 (Wifi 11ac)	有線 (Gigabit Ether)	Omni-path (OBCX)
転送速度 [Gbit/秒]	0.1 ~ 1.0	6.9	1	100



性能指標 ～ OBCXを例に (3)

大容量・高速のストレージ

不老 ホットストレージ（FEFS）

- 容量 ： 30.44PByte (33,440TByte)
- 構成 ： HDD 14TB x 730本
- 転送速度 ： おおよそ100GByte/秒

HDD,SSDの転送速度は
HDD ：0.1～0.2GByte/秒
SSD ： ～7GByte/秒
*2023年9月現在

全体性能

FLOPS値（Floating Point Operations per Second, 浮動小数点演算）

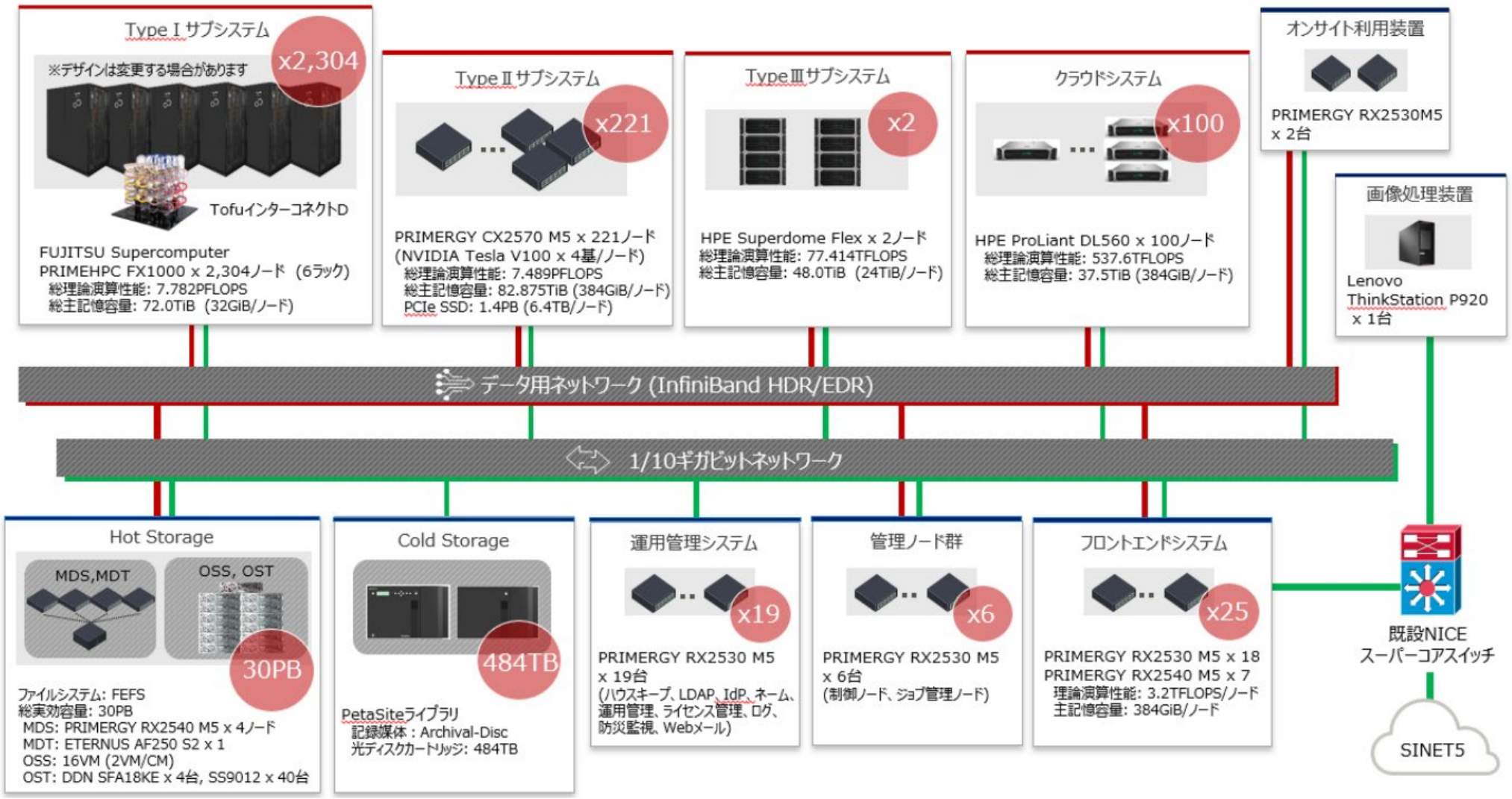
1秒当たりの実数演算性能

- 10^6 FLOPS= 1 Mega FLOPS = 1 MFLOPS （百万）
- 10^9 FLOPS= 1 Giga FLOPS = 1 GFLOPS （十億）
- 10^{12} FLOPS= 1 Tera FLOPS = 1 TFLOPS （兆）
- 10^{15} FLOPS= 1 Peta FLOPS = 1 PFLOPS （千兆）
- 10^{18} FLOPS= 1 Exa FLOPS = 1 EFLOPS （百京）

不老 Type Ⅰ

- Fujitsu A64fx：1コアの性能は86.4GFLOPS
  - » 1秒間に864億回の倍精度実数演算
- 1ノード：48コア
  - » 3,379.2 GFLOPS= 3.3792 TFLOPS ～ 3兆3,792億回
- 全システム：2,304ノード **7.782 PFLOPS** ～7,782兆回

不老全体の構成



名古屋大学
NAGOYA UNIVERSITY

- スーパーコンピュータとは？
- **スーパーコンピュータで何ができるのか？**
- スーパーコンピュータの使い方
  - ✓ ログイン
  - ✓ プログラム作成
  - ✓ コンパイル
  - ✓ 実行
  - ✓ 結果の確認

スパコン利用に向けて何を学ぶのか？

Case A ） 新しい計算向けのプログラムを作る
誰かが作ったプログラムを速くする

特有の技術必要！

繰り返し計算を高速化したい

基本のところは C/C++ と FORTRAN で共通

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

1. CPU、GPUが動作しやすい形にプログラムを書き換える方法

   SIMD演算、キャッシュの活用、メモリアクセス改善

2. 協調動作のための分散処理方法、CPU間など通信方法、GPUの利用

   OpenMP　　MPI　　OpenACC、CUDA

3. 上記1. 2. を最適にして演算を行うライブラリの呼出方法

スパコン利用に向けて何を学ぶのか？

Case B ） スパコン向けに作られたソフトウェアを利用する

数値流体力学、構造解析、統計解析、
分子動力学、電子状態計算、量子化学計算、…

ソフトウェアがどのように協調動作するのかの概略を
理解することにつとめる

→ 並列プログラミングをしない人でも、使ってみた
速さなどから、計算の仕組みのイメージを獲得

スパコン利用に向けて何を学ぶのか？

Case C） 新しいタイプのデータ処理（学習）をしたい

ー　多くの方は、手元のマシンで何かされてる方かもしれません

1. Python等による高速なデータ処理

2. スパコン特有の環境構築の方法（コンテナ環境の利用など）
   多数のGPUの使用方法

3. バッチジョブシステム特有の事項

注：大学設置スパコンだとどうしても（民間のクラウドと比較すると）
少し環境準備が手間がかかるところはあります。

スパコン利用に向けて何を学ぶのか？

Case D ） 多数のパラメータ、パターンの組み合わせを試したい

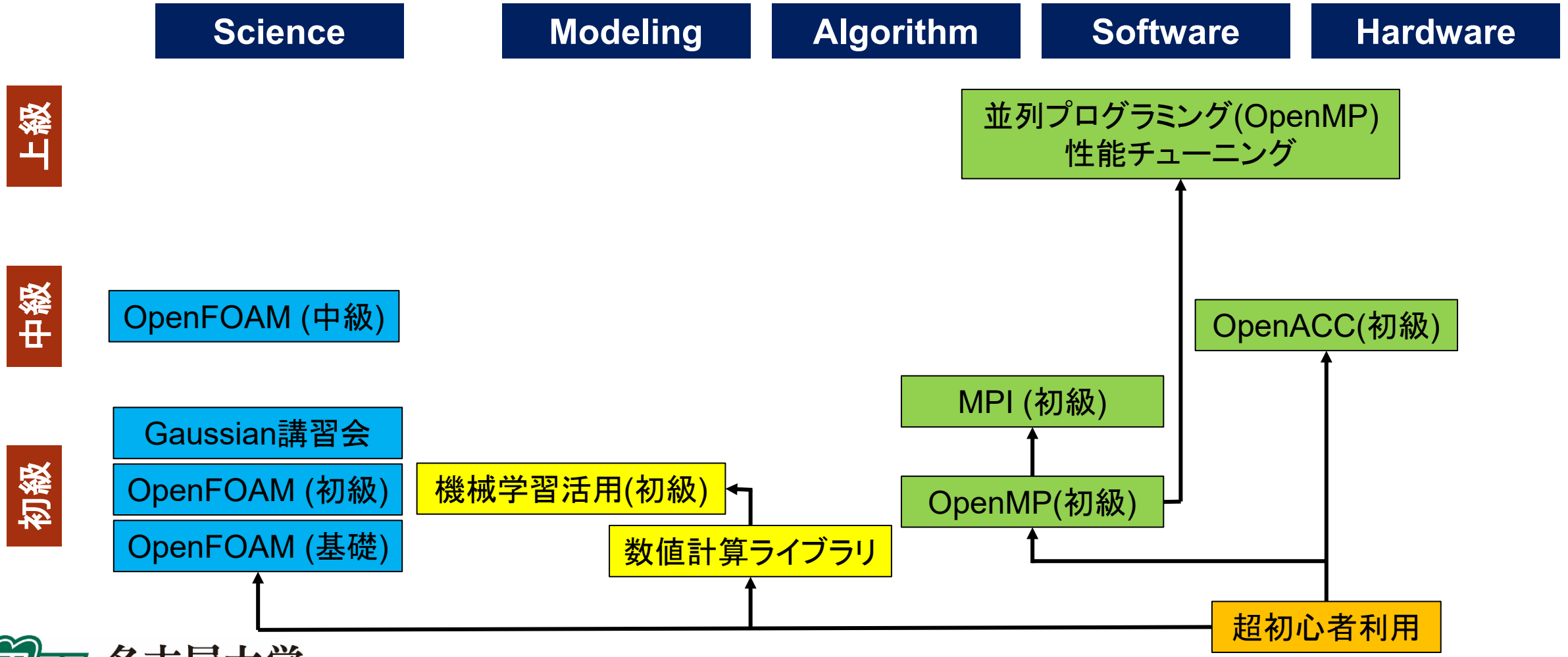
Embarrassingly Parallel(バカパラ)

1.バッチジョブシステムとどう付き合うか

Xcryptというツールがあります。
講習会も検討中

スパコン向け最適化に関する講習会 @ 名大ITC

目的に応じて受講をご検討ください。



名古屋大学 NAGOYA UNIVERSITY

- スーパーコンピュータとは？
- スーパーコンピュータで何ができるのか？
- スーパーコンピュータの使い方
  - ✓ Linux
  - ✓ ログイン
  - ✓ プログラム作成
  - ✓ コンパイル
  - ✓ 実行
  - ✓ 結果の確認

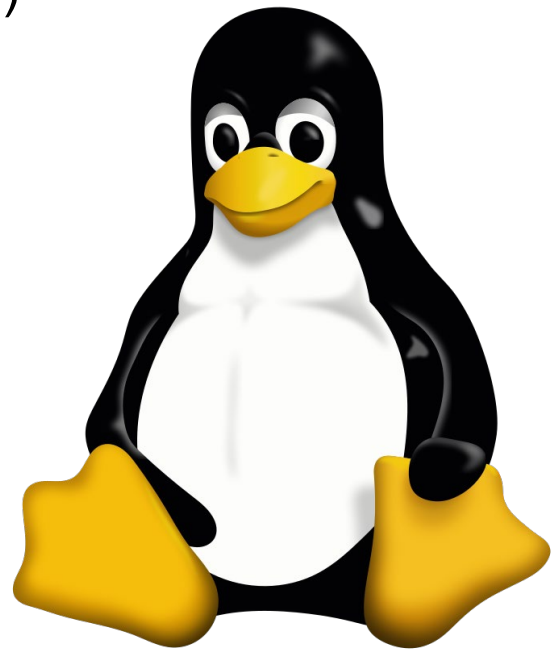
Linux OS

世界中のスパコンのほとんどが採用しているOS
→これが使えないと何もできない・・・・ので最初にやります！

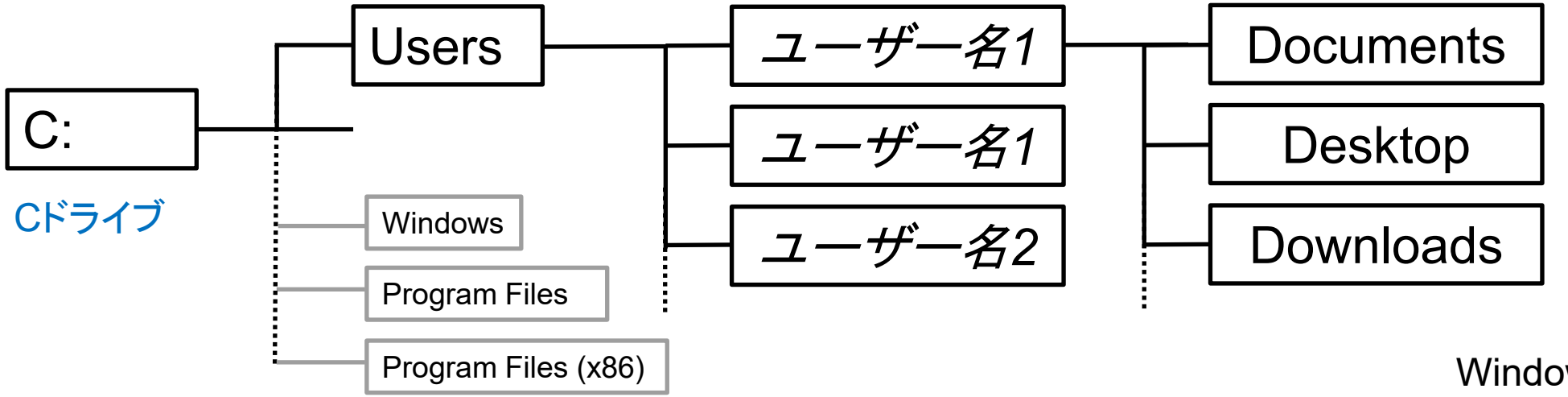
- Operating Systemの1つ (他にはWindowsやOS X, iOS, Androidなど・・・)
- 現在のTOP500（上位500台のスパコンランキング）のシェアは100%
- スパコンだけでなく、サーバーの多くはLinuxで動作
- 基本的にOpen Sourceで構成
- 無料のOS
- CLI（Command Line Interface)で完結、高機能なShellが用意されている。
  - 本講習ではBashを使用
  - マウスでする操作を文字で行うイメージ
  - GUI(Graphical User Interface)もあり。
    スパコンではあまり使わない。

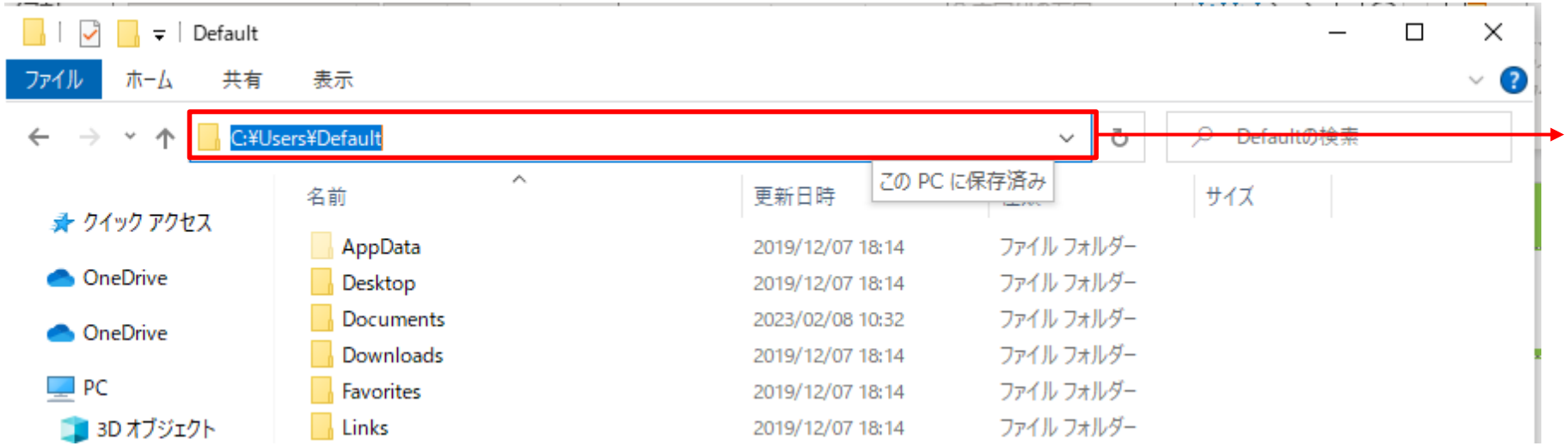
講習会では基本的なコマンドとBashについて扱います。
bashにコマンドを渡して必要な操作を実施
bash = windowsでいうターミナルやPowerShellみたいな物
LinuxではWindows、Macでいうフォルダをディレクトリといいます。（この後頻出します。）

Tux



Windowsのフォルダ構成





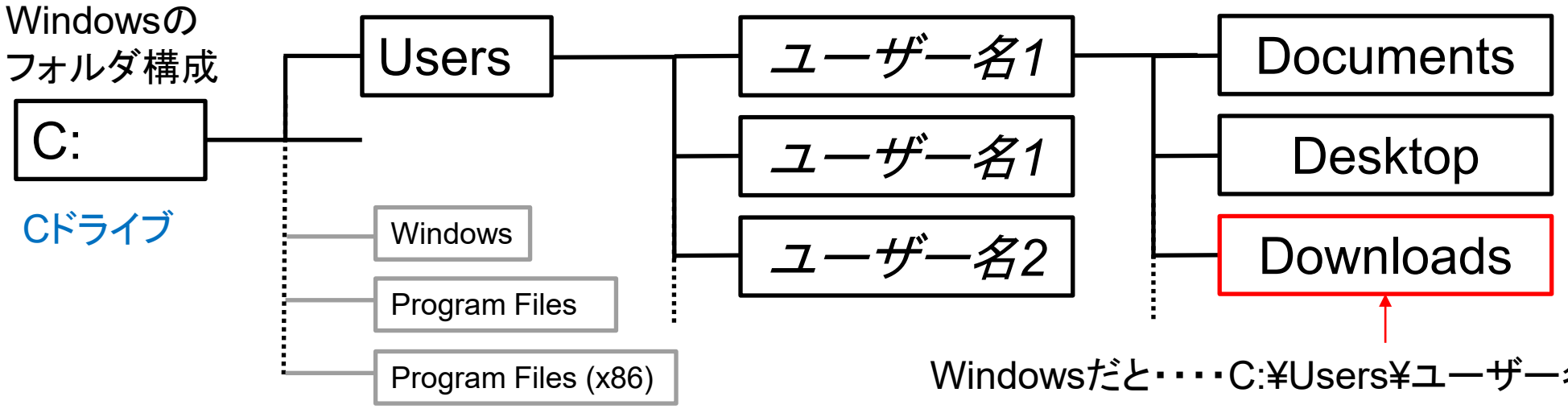
Windowsの場合、ここを
クリックすると、パスが出てくる。

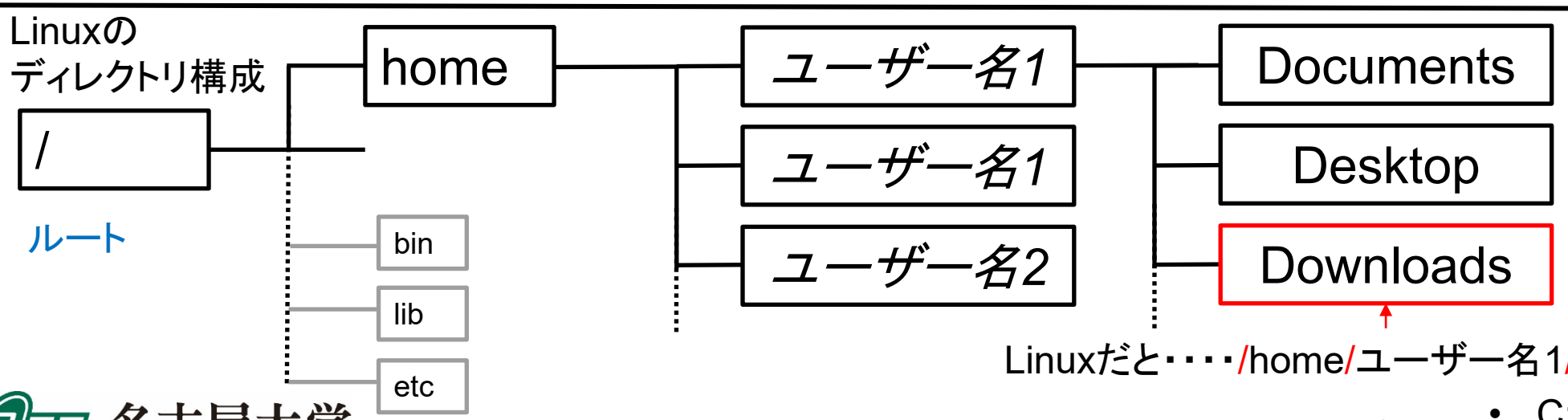
¥区切りでフォルダが表示されている。

例だと
- Cドライブの下の
- Usersの下の
- Defaultというディレクトリ

Linuxのディレクトリ構成

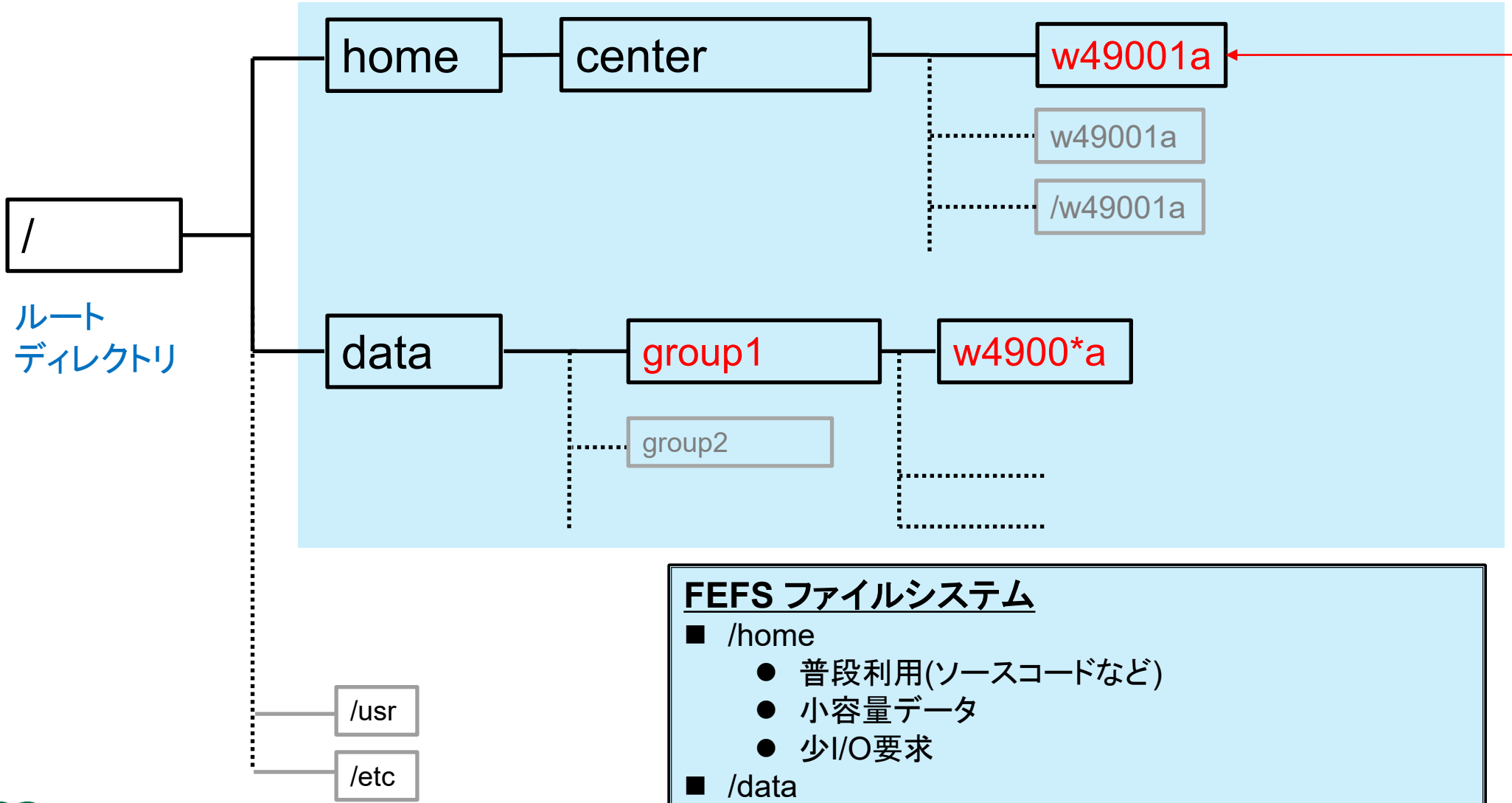
ディレクトリ(Windows、Macではフォルダ)は、名前は違えど考え方はほとんど変わらない。





違いは
- C:から始まらない
- 区切り文字はスラッシュ

スパコン(不老)でのディレクトリ構造



FEFS ファイルシステム

- /home
  - 普段利用(ソースコードなど)
  - 小容量データ
  - 少I/O要求
- /data
  - 大容量データ
  - 多I/O要求

コマンドとは？

特定の機能(GUIだとマウスを使ってする操作)を実行する命令

$コマンド [オプション1、オプション2・・・] 引数1、引数2・・・・・

$ cd /home/center/w4900*a/ :作業ディレクトリを変更(Windowsだと、フォルダのダブルクリック)
$ cp a.txt b.txt :a.txt と同じ内容のファイルを b.txt として作成
(Windowsだと、右クリック→コピー、ペースト＆名前の変更)

コマンドの例

コマンド	用途	よく使うオプション
pwd	現在のディレクトリ(カレントディレクトリ)を表示	
cd	ディレクトリを変更 (引数なしだとホームに移動)	
ls	ファイル一覧を表示	-l ： 詳細表示
cp	ファイルまたはディレクトリのコピー	-r ： 再帰的にコピー
rm	ファイルまたはディレクトリの削除	-r ： 再帰的に削除
exit	セッションの終了（スパコンからログアウト）	

$ hogehoge と書いてあると、hogehoge部はコマンドだと思ってください。

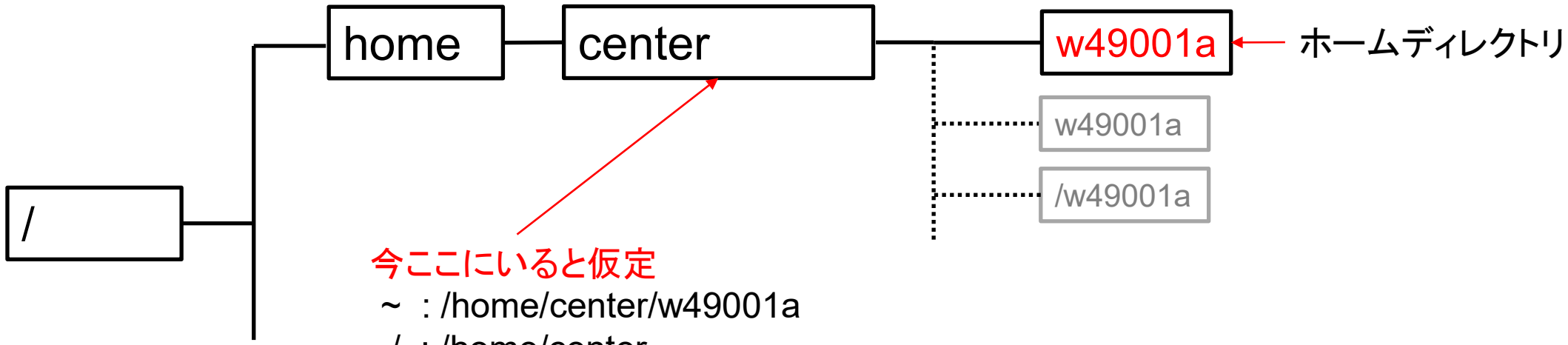
Linuxコマンドで特別な意味を持つ記号 (メタキャラクタ)

以下の文字列は特別な意味を持つので、ディレクトリ、ファイル名に使わないようにしてください。
(正確には使えますが、慣れないうちはトラブルの元です。)

- (Space) : アンダーライドなどに置き換えてください。
- . (Dot)
- * (Asterisk)
- / (Slash)
- | (Pipeline)
- ~ (Tilde)
- ? (Question)
- ‘ (Single Quote)
- “ (Double Quote)
- [] (Square Bracket)
- {} (Curly Bracket)

メタキャラクタの使い方

- ~：ホームディレクトリ
- ./ ：今のディレクトリ(カレントディレクトリ)
- ../：1つ上のディレクトリ



- *：全て

例　：test*　→ testで始まる全ての文字列 test1, test2, test11, test22……..

コマンドの実行例

以下のコマンドを実際に実行してみてください。

各コマンドで何をしているか意識してください。
常に、自分はどのディレクトリにいるのかを意識して下さい。

```
[w4900*a @flow-fx01 ~]$ pwd                    現在のディレクトリを出力
/home/center/w4900*a
[w4900*a @flow-fx01 ~]$ mkdir test             testディレクトリの作成
[w4900*a @flow-fx01 ~]$ ls                     現在のディレクトリの中身の出力
test
[w4900*a @flow-fx01 ~]$ cd test                testディレクトリに移動
[w4900*a @flow-fx01 test]$ pwd
/home/center/w4900*a/test
[w4900*a @flow-fx01 test]$ cd                  引数なしでcdコマンドを実行すると、ホームディレクトリに移動
[w4900*a @flow-fx01 ~]$ pwd
/home/center/w4900*a
```

Bash特有の機能 1/2

コマンド入力を簡易にする基本機能一覧

	操作	動作内容
Auto fill	tab	途中まで入力したコマンドやパスの残りを補完
履歴	↑	過去に実行したコマンドを1つずつさかのぼる
履歴探索モード	ctrl+r	過去に実行したコマンドを検索
リダイレクト	>	結果をファイルなどに出力 例：ls > list.txt　list.txtにlsの結果が出力される
パイプ	|	結果を次のコマンドに渡す 例：ls | sort　　lsコマンドの結果をソートする

コマンドとbashの機能を使用したスクリプトも記述可能

bashの便利機能 2/2

特に、以下の機能は初めての方にも便利

- Auto fill
  - ✓ Tabキーを押すと入力中の文字を予測してその後を補完
    - ➢ testtesttesttest.txtというファイルがある
    - ➢ testtまで入力した後にTabキーを押す
    - ➢ testtに続くほかのファイルがなければ、esttesttest.txtを勝手に入力
    - ➢ testtに続くほかの候補がある場合は、tabキーを2回押すと、全ての候補を出力
- 履歴、履歴探索モード
  - ✓ ↑キーを押すと、過去のコマンドを1つずつ遡る
  - ✓ ctrl+rを押してから、過去に実行したコマンドを入力すると、履歴から後ろを自動で入力
  - ✓ ctrl+rを何度も押すとどんどん過去に

課題

当日のお楽しみ

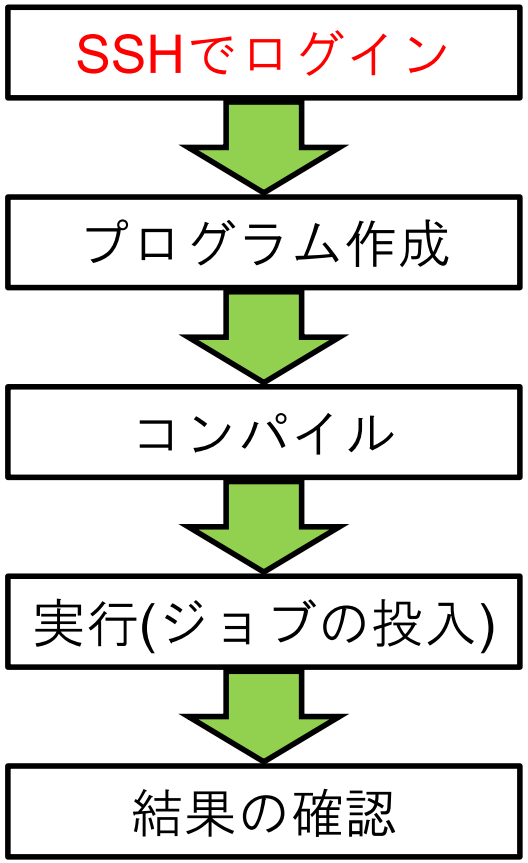
問題

当日のお楽しみ

ほかのコマンドの例

コマンド	用途	よく使うオプション
mv	ファイルまたはディレクトリの移動	
mkdir	ディレクトリの作成	
man	コマンドの説明	
cat	ファイル内容の表示（全体）	
less	ファイル内容の表示（一画面ごと）	
head	ファイル内容の表示（先頭部分）	-n ## ： 表示行数の指定
tail	ファイル内容の表示（末尾部分）	-n ## ： 表示行数の指定
grep	ファイルに含まれる文字列を検索	
awk	文字列の抽出	
sed	文字列の置換	
diff	２つのファイルの内容比較	
echo	変数や文字列の値を出力	
exit	セッションの終了	

スパコンを使うための手順



SSH、鍵認証、鍵ペア作成、ログイン

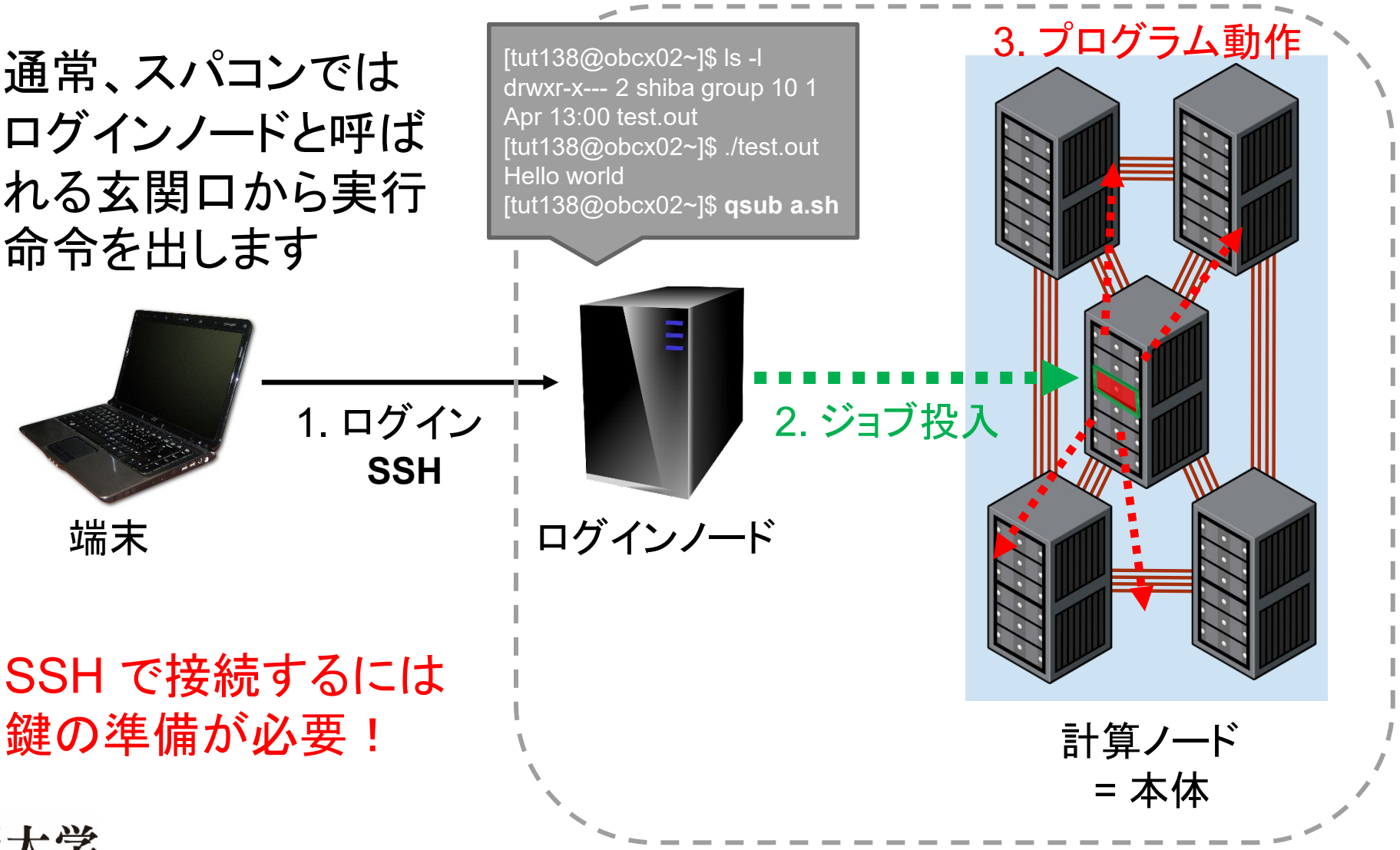
エディタ(emacs, vi, nano)、Linux 利用法

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

スーパーコンピュータ (スパコン) 利用のイメージ

通常、スパコンではログインノードと呼ばれる玄関口から実行命令を出します



SSH で接続するには鍵の準備が必要！

Secure Shell(SSH) プロトコル

通信が暗号化されたShell

- ShellはOSとユーザーの仲介をする
  コマンドベースのソフトウェア
- 通信データを暗号化し、リモートマシンに
  アクセスできる方法:SSH

暗号化された通信を使用して、
様々なことが可能

- ファイルのコピー
- グラフィカル画面の転送
- トンネリング
- ディレクトリのマウント

ログイン後の画面の一例

```
[ 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 ]$
```

操作はsshで暗号化される・・・・ログインは？→公開鍵暗号方式を使います。

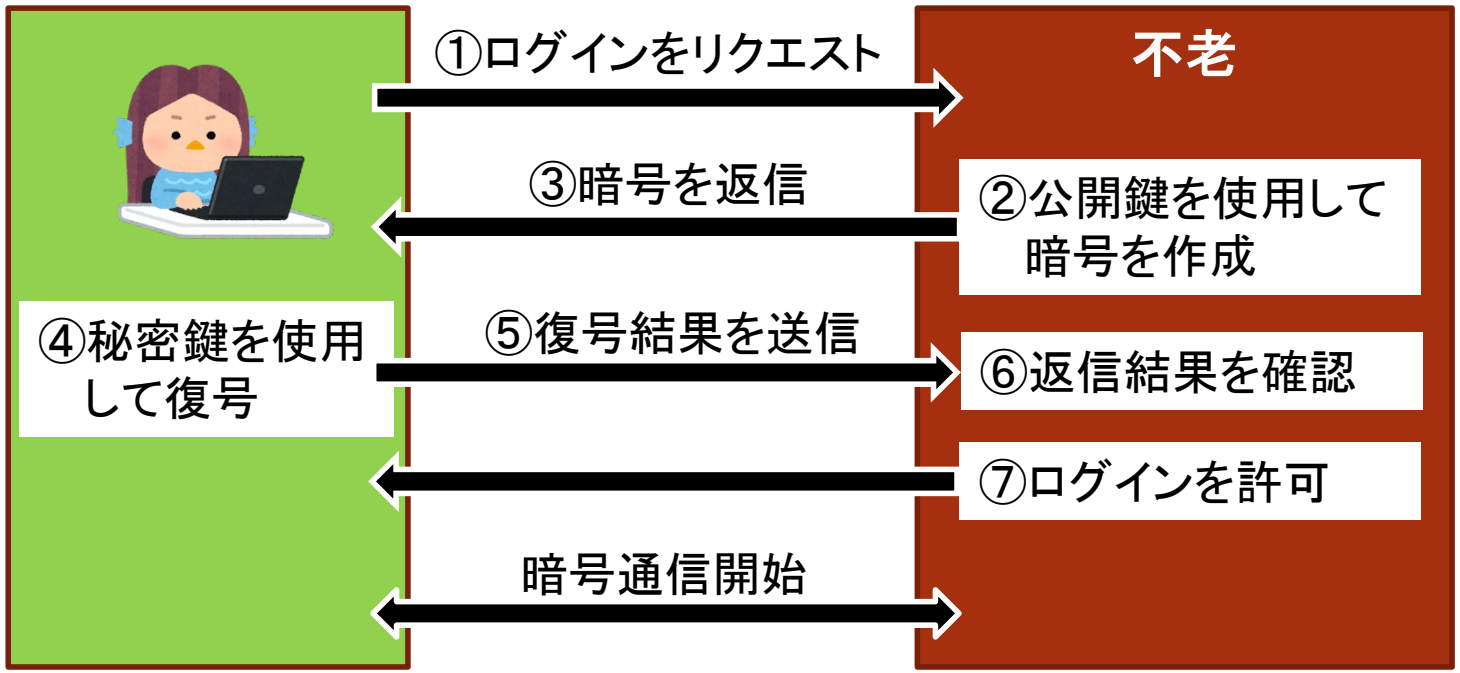
鍵認証方式 (1/2)

より安全な接続をする→鍵認証方式

- パスワードではなく、秘密鍵と公開鍵のペアを使用してログインする方法
- 秘密鍵にもパスワードを設定可能

初期設定（初回ログイン時のみ）

- 鍵ペアを作成
- 公開鍵をログインノードに登録

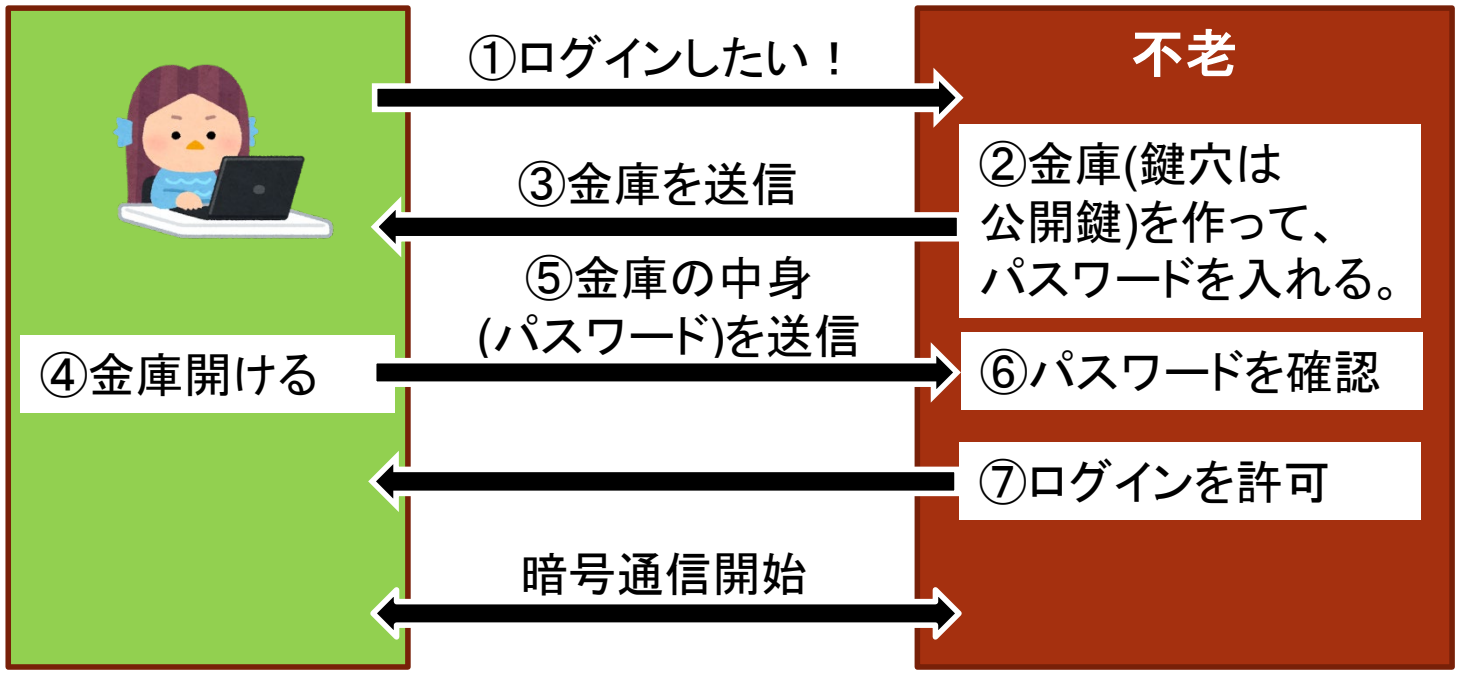


鍵認証方式 (2/2)

鍵を開けてスパコンに入る！（実際はもうちょっと複雑）

- 秘密鍵→物理鍵
- 公開鍵→鍵穴

- 秘密鍵を持っていって、ログインを試行
  →通信経路を傍受されていると、物理鍵が見られて終わり



鍵認証方式の注意点

注意点

■秘密鍵の取り扱いに注意

- 厳重に管理してください。
  - ✓漏洩すると容易にログインできてしまうため。
- 秘密鍵は他のところにコピーしたりしないでください。
- 秘密鍵の入ったPCの紛失などがあった場合は速やかに公開鍵を更新してください。

■ 鍵の生成時には必ずパスワードを設定してください。

よく間違える点

■ 秘密鍵のパスワードはHPポータルのログインパスワードやログイン後のアカウントのパスワードとは異なります。

SSHを使えるようにする。

- Mac、Linuxの方：何もする必要ありません。
- Windowsの方：すでにubuntu on windowsまたはCygwinをインストール済みなので、sshが利用可能です。

Windowsの場合、以下の方法もあります。

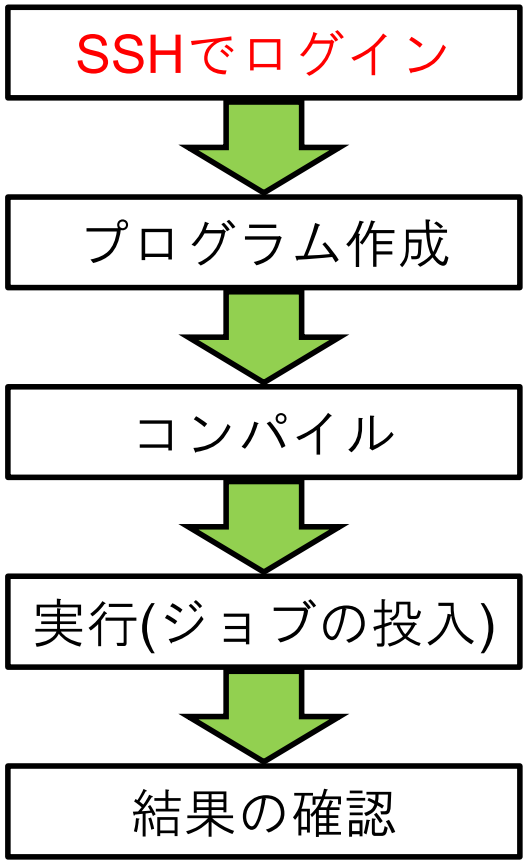
1. PuTTY or TeraTerm + WinSCP

   sshの機能だけをインストール

2. OpenSSH client from PowerShell ：要管理者権限？

   Windows標準のCLIにsshの機能をインストール

スパコンを使うための手順



SSH、鍵認証、**鍵ペア作成**、ログイン

エディタ(emacs, vi, nano)、Linux 利用法

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

鍵の作成

PC上で鍵(秘密鍵，公開鍵)を生成(1/3)

Cygwin またはターミナルを開いたその場所から下記の操作を開始

```
$ 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): パスフレーズを設定 Return
Enter same passphrase again: 同じパスフレーズ 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:vt880+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]-----+
```

操作手順

- **ssh-keygen -t rsa <Return>**
- <Return>
- お好きなパスフレーズ <Return>
- 同じパスフレーズ <Return>

鍵が作成できたはず

名古屋大学 NAGOYA UNIVERSITY

鍵の作成

PC上に秘密鍵、公開鍵があることを確認(2/3)

コマンドの意味がわからない方も、黄色の文字を打って、同じような画面になる確認してください。

```
$ cd .ssh

$ ls

 id_rsa                                ⇒秘密鍵 (Private Key)
 id_rsa.pub                            ⇒公開鍵 (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
```

鍵の作成

公開鍵をコピーする(3/3)

通常のカットアンドペーストの手順で「公開鍵」をコピー

```
$ 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
```

操作手順

- **cat id_rsa.pub <Return>**
- “ssh-rsa”にカーソルを合わせ
- 最後の行の”f3”までを選択して「Copy」によって記憶
- 最後の「tut138@ITCUT-VAIO」まで含んでも良いが，ここに漢字が含まれていると登録に失敗します。

SSH 公開鍵認証

id_rsa

- Private Key（秘密鍵）：PC上
- 文字通り「秘密」にしておく
  作成した場所からコピー・移動しない、他の人に送らない

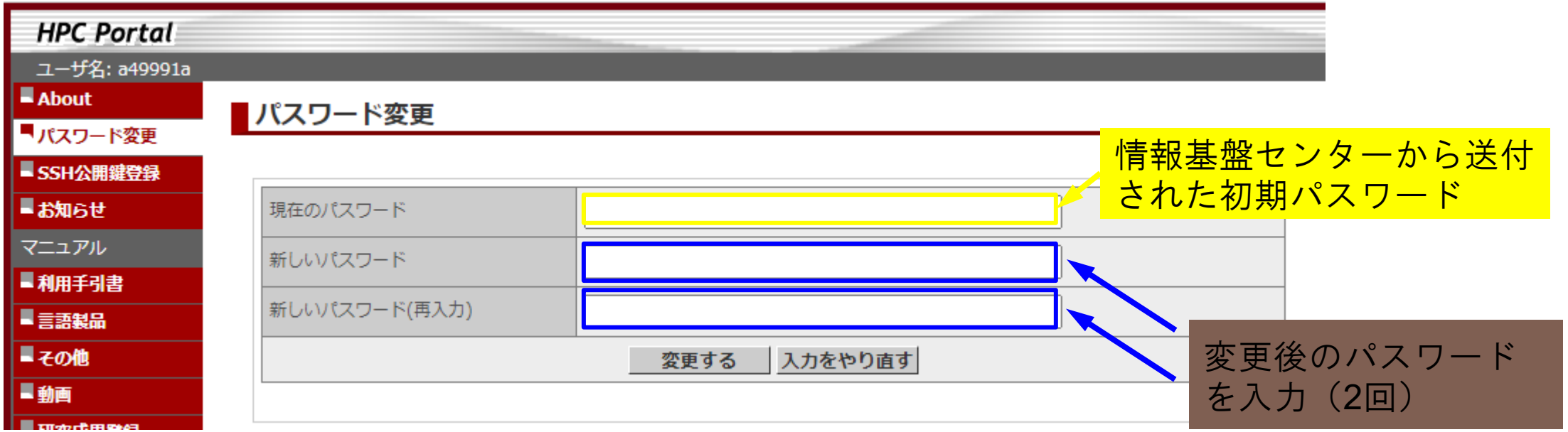
id_rsa.pub

- Public Key（公開鍵）：スパコン上
- コピー可能，他の人にe-mailで送ることも可能

- もし複数のPCからスパコンにログインする場合は，各PCごとに「公開鍵・秘密鍵」のペアをssh-keygenによって作成
- 各スパコンには、複数の公開鍵を登録できます

スパコン上の公開鍵のうちの一つがPC上の「秘密鍵＋Passphrase」とマッチすると確認されるとログインできる

②初期パスワードの変更



③公開鍵登録（id_rsa.pub）

1. 「SSH公開鍵登録」を選択
2. 先ほどCopyした公開鍵（ id_rsa.pub) を貼り付ける
3. 「登録」をクリック

よくトラブルが起きますので、先に進めなければ質問してください

名古屋大学 NAGOYA UNIVERSITY

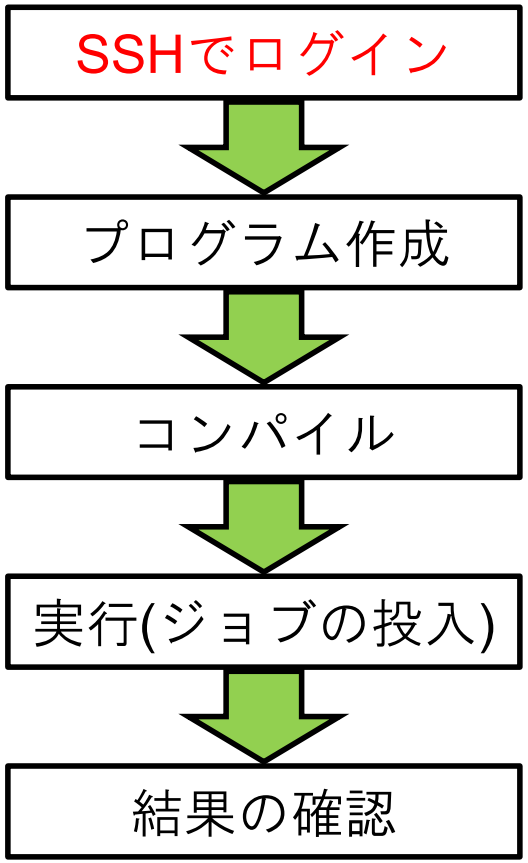
複数の鍵を登録する場合

- HPCポータルからの登録は1回限り
  その一回で複数の登録は可能
- 2つ目以降は不老にログイン後、追記
  /home/ユーザ名/.ssh/authorized_keys

HPCポータルについて

- HPCポータルではスパコンのマニュアルも公開されています。
  必要に応じてご閲覧ください。

スパコンを使うための手順



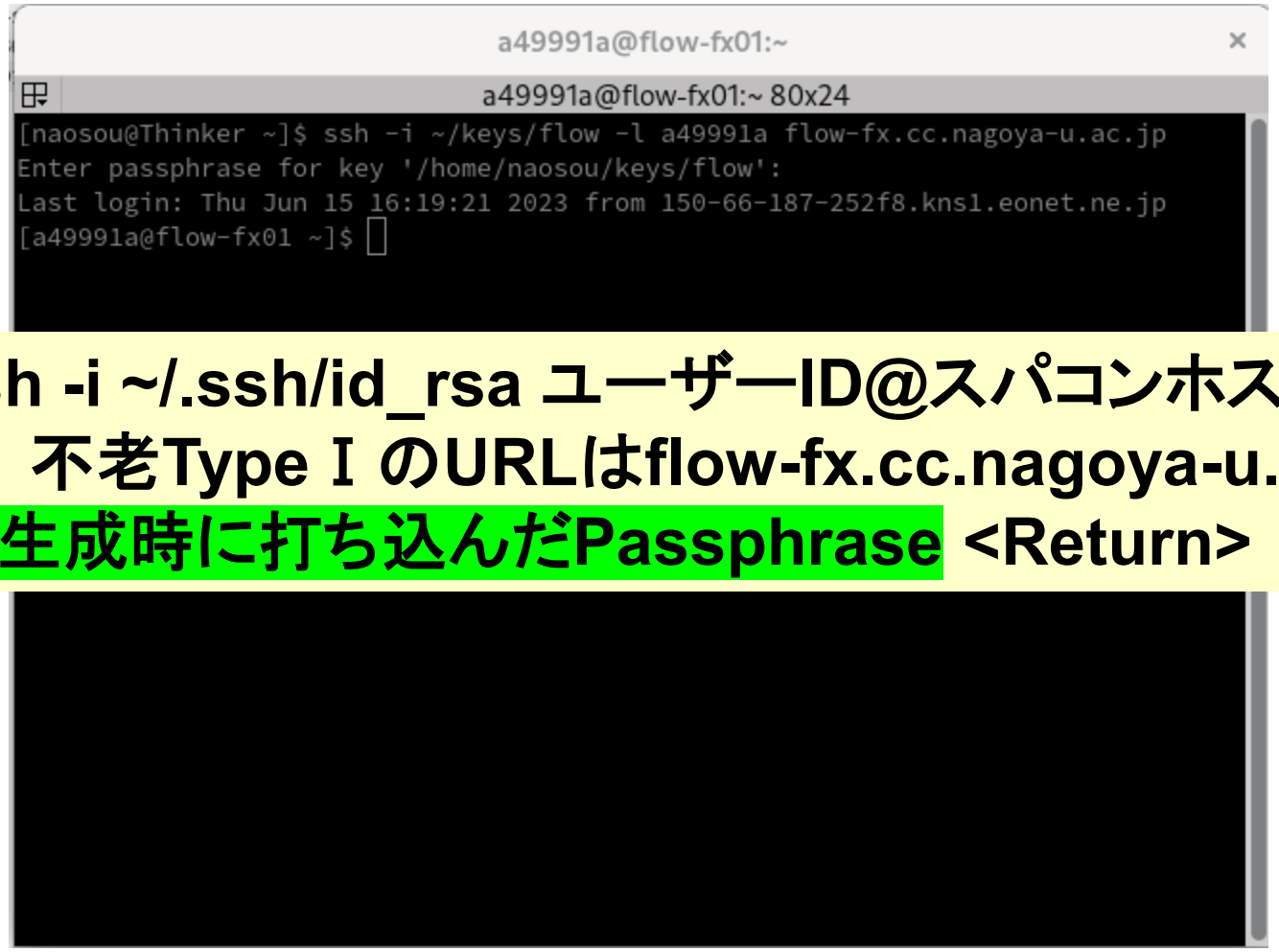
SSH、鍵認証、鍵ペア作成、ログイン

エディタ(emacs, vi, nano)、Linux 利用法

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

スパコンにログイン (Cygwin, Mac, Linux)

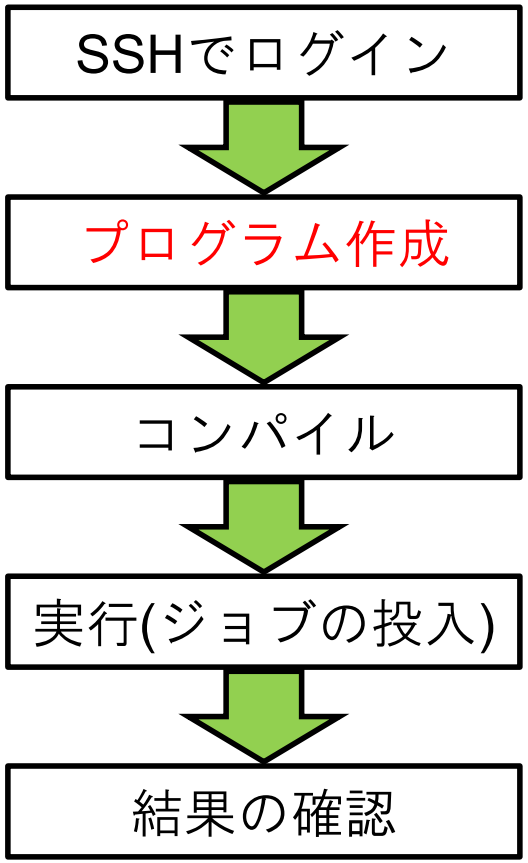


1. ssh -i ~/.ssh/id_rsa ユーザーID@スパコンホスト名<Return>
   不老Type ＩのURLはflow-fx.cc.nagoya-u.ac.jp
2. 鍵生成時に打ち込んだPassphrase <Return>

スパコンへのログインまで完了しました！

休憩

スパコンを使うための手順



SSH、鍵認証、鍵ペア作成、ログイン

エディタ(emacs, vi, nano)

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

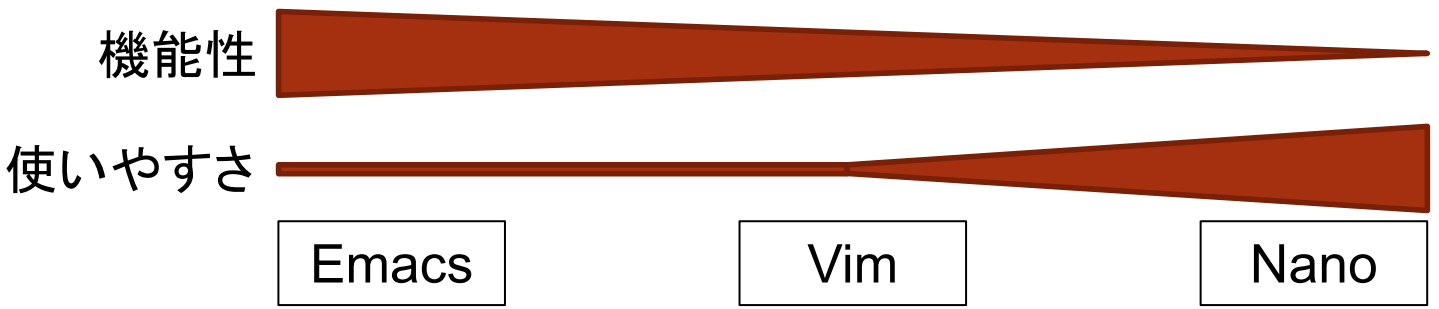
エディタ

ファイルを編集するためのソフトウェア

Windowsの標準だとメモ帳に該当

Linuxでは以下のエディタが有名（ほかにもたくさんあります）

- Emacs
  - 高機能
  - 拡張機能も豊富 (LISPで記述されている)
- Vim
  - 軽量
  - Linuxの初期状態でインストールされている
- Nano
  - 一番簡易
  - コマンドが下段に表示されるため、わかりやすい



エディタでプログラムを作成する

さて、実際に作業してみましょう。
プログラムを編集するディレクトリを作成します。

```
[w4900*a@flow-fx01 ~]$ cd Return
[w4900*a@flow-fx01 ~]$ mkdir test Return
[w4900*a@flow-fx01 ~]$ cd test Return
```

ログインノード上では、Emacs, vim, または nano のエディタを利用してください。

例1： Emacs を使用，Fortranプログラムを作成

```
[w4900*a@flow-fx01 test]$ emacs hello_world.f90 Return
```

例2： vim を使用，Python プログラムを作成

```
[w4900*a@flow-fx01 test]$ vim hello_world.c Return
```

例3： nano を使用, C プログラムを作成

```
[w4900*a@flow-fx01 test]$ nano hello_world.c Return
```

Emacs チートシート

$ emacsで起動

Emacs ＝「control (ctl) キー ＋ 文字」でコマンドに移行

コマンド	
Ctl-x Ctl-s	編集中のファイルを同じファイル名で保存
Ctl-x Ctl-w	編集中のファイルを名前をつけて保存
Ctl-x Ctl-f	既存のファイルを開く
Ctl-x Ctl-c	Emacs を終了する （保存するか聞かれたらY/N で答える）
Ctl-g	コマンド入力
Ctl-a	行の先頭にカーソルを持ってくる
Ctl-k	カーソルから行末までを削除し、 （クリップボードみたいなところに）コピー
Ctl-y	コピーされた内容を貼り付ける
C-space	Mark Set
Mark Set, 範囲選択, M, w	コピー
Mark Set, 範囲選択, C-w	切り取り
Ctl-s	文字列を検索する
Esc-%	文字列を置換する

コマンド確認画面

vim チートシート

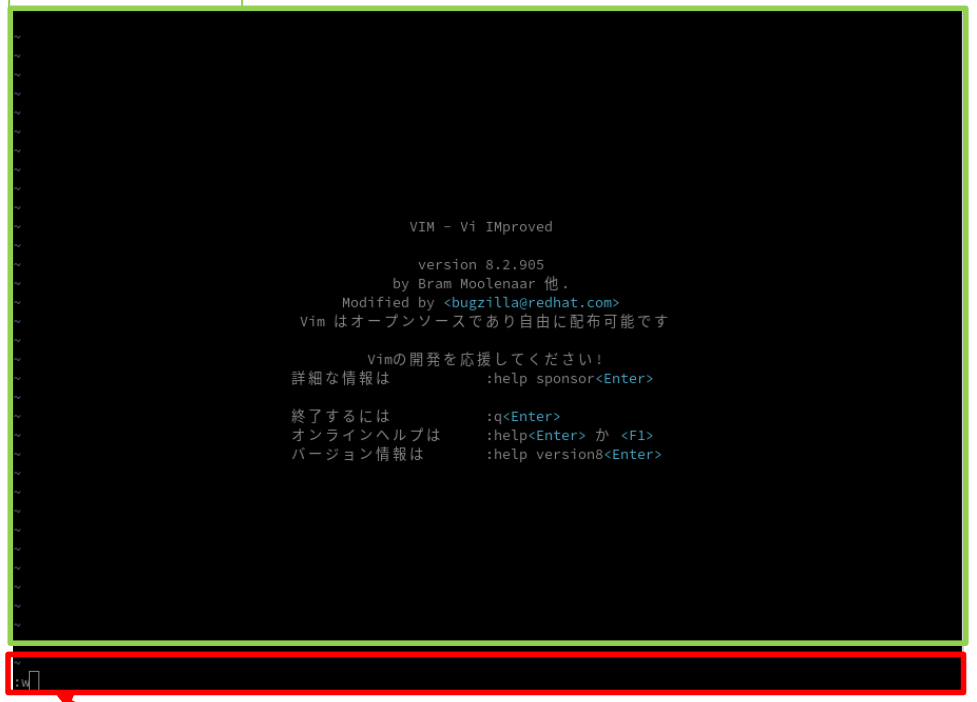
$ vi で起動

ノーマルモード / 入力モード / コマンドを切り替え

現在のモード	移行先モード	キー
ノーマル	入力	i, a, Ins
ノーマル	コマンドライン	:
入力	ノーマル	esc, ctrl-c

キー	
Esc	ノーマルモードに移行する
i	入力モード（挿入）に移行する
o	新しい行を追加して入力モード（挿入）に移行
R	入力モード（上書き）に移行する
:w ###	コマンド - ファイル名 ### で保存する
:q	コマンド - vim を終了する
:q!	コマンド - 保存せずにvim を強制終了する
/###	### という文字列を検索する
yy	カーソルのある行をコピーする
p	貼り付けする

編集画面



モードおよびコマンド確認画面
- ブランク → ノーマルモード
- : → コマンドモード
  - :w Ret 上書き保存
- --挿入-- → 入力モード

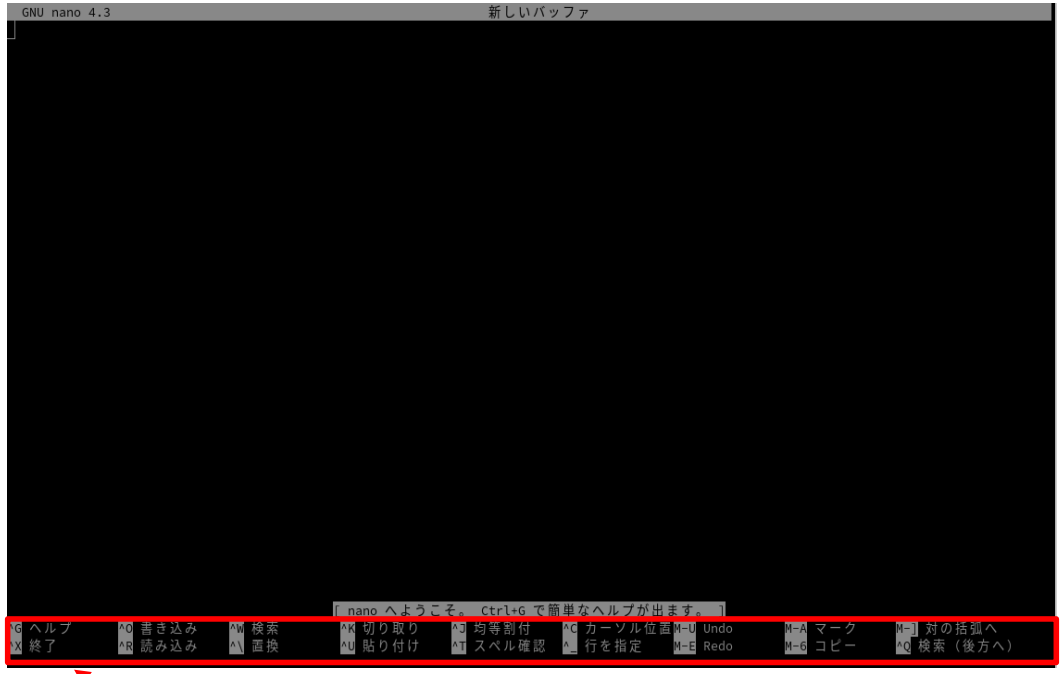
nano チートシート

nano = 最も操作方法が単純　　ー予習漏れの場合はお使いください。

$ nano で起動

- M-U → Escキー、Uキーを順番に押す

キー	
Ctl+S	ファイルを保存
Ctl+O	ファイルを名前をつけて保存
Ctl+X	nano エディタ を終了する
Ctl+W	文字列を検索（順方向）
Ctl+D	カーソルのある文字を削除
M-A	マーク
M-6	コピー
Ctl+K	1行カットして削除
Ctl+U	ペースト



コマンド確認画面

- ^ → Ctl
- M → Esc
- - → 順次押す

プログラム例題：Hello World

C言語

プログラム作成 `[w4900*a@flow-fx01 test]$ emacs hello_world.c`

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

int main(int argc, char *argv[]){
  int me_proc, me_thrd;
  int num_procs, num_thrds;
  int prov;

  MPI_Init_thread(&argc, &argv, MPI_THREAD_FUNNELED, &prov);
  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
  MPI_Comm_rank(MPI_COMM_WORLD, &me_proc);

#pragma omp parallel private(num_thrds, me_thrd) firstprivate(num_procs, me_proc)
  {
    num_thrds = omp_get_num_threads();
    me_thrd = omp_get_thread_num();
    printf("Hello world!!, Number of processes = %d, Number of threads = %d, process-ID = %d, thread-ID = %d¥n",
                    num_procs, num_thrds, me_proc, me_thrd);
  }

  MPI_Finalize();

  return 0;
}
```

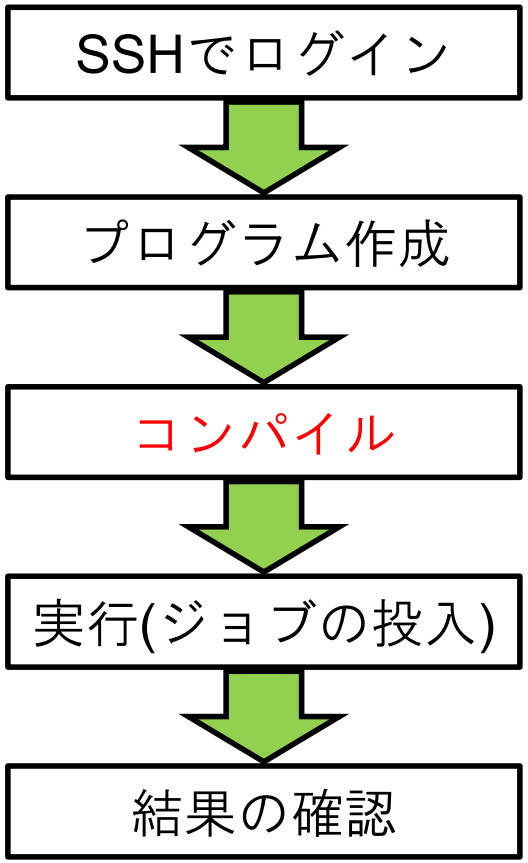
プログラム例題：Hello World

Fortran

プログラム作成 `[w4900*a@flow-fx01 test]$ emacs hello_world.f90`

```
program main
  use mpi
  !$ use omp_lib
  implicit none
  integer me_proc, me_thrd;
  integer num_procs, num_thrds;
  integer prov, ierr

  call MPI_Init_thread(MPI_THREAD_FUNNELED, prov, ierr)
  call MPI_Comm_size(MPI_COMM_WORLD, num_procs, ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD, me_proc, ierr)
  me_proc = me_proc + 1

!$OMP parallel private(num_thrds, me_thrd) firstprivate(num_procs, me_proc)
  num_thrds = omp_get_num_threads()
  me_thrd = omp_get_thread_num() + 1

  write(*, '(4(a, i3))') "Hello world!!, Number of processes =", num_procs, ", Number of threads =", num_thrds, &
                         ", process-ID =", me_proc, ", thread-ID =", me_thrd
!$OMP end parallel

  call MPI_Finalize(ierr)

end program main
```

スパコンを使うための手順



SSH、鍵認証、鍵ペア作成、ログイン

エディタ(emacs, vi, nano)

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

Environment modulesの利用 (1)

コンパイラ等の選択

富士通コンパイラ、gnu コンパイラ，インテル製コンパイラ，

ライブラリを利用した計算の高速化

離散フーリエ変換，線形代数・連立１次方程式，…

使用中モジュールの表示 (デフォルトは富士通コンパイラ)

```
[w4900*a@flow-fx01 ~]$ module list  Return
Currently Loaded Modulefiles:
  1) tcs/1.2.37
```

もしGCCコンパイラに切り替えるなら・・・・

```
[w4900*a@flow-fx01 ~]$ module purge  Return
[w4900*a@flow-fx01 ~]$ module list  Return
No Modulefiles Currently Loaded.
[w4900*a@flow-fx01 ~]$ module load gcc/10.4.0  Return
```

利用可能なEnvironment modulesの一覧を表示できます

```
[w4900*a@flow-fx01 ~]$ module avail
```

Return

```
------------------------------------------ /home/center/opt/aarch64/modulefiles ------------------------------------------
amber/19.0(default)                  mumps/5.2.0(default)                 py-matplotlib/3.2.1(default)
boost_gcc/1.77.0(default)            mxnet/1.6.0                          py-mpi4py/3.0.3(default)
caffe/1.0(default)                   mxnet/1.8.0(default)                 py-numpy/1.18.5
cmake/3.17.1                         namd/2.14(default)                   py-numpy/1.19.0(default)
cmake/3.21.1(default)                namd/2.14b1                          py-onnx/1.6.0(default)
conda4aarch64/1.0.0(default)         netcdf-c/4.7.3                       py-pillow/7.0.0(default)
dhpmm-f/alpha(default)               netcdf-c/4.7.4(default)              py-pytest-check-links/0.3.4(default)
ffb/8.1(default)                     netcdf-cxx/4.2(default)              py-scikit-image/0.14.2(default)
ffr/3.0.000                          netcdf-cxx4/4.3.1(default)           py-scikit-learn/0.23.1(default)
ffr/3.1.004(default)                 netcdf-fortran/4.5.2                 py-scipy/1.4.1(default)
fftw/3.3.8(default)                  netcdf-fortran/4.5.3(default)        py-tensorflow/2.1.0(default)
fftw/3.3.9-tune                      openblas/0.3.17(default)             py-theano/1.0.4(default)
fftw-tune/3.3.8(default)             opencv/4.2.0(default)                python/2.7.16
fjmpi-gcc/10.2.0                     opencv/4.5.4                         python/3.7.7(default)
fjmpi-gcc/10.4.0                     openfoam/v1812-tune                  python/3.8.12
fjmpi-gcc/8.3.1(default)             openfoam/v2006(default)              py-torch/1.5.0(default)
frontistr/5.0                        openfoam/v2106                       qe/6.4.1-tune
frontistr/5.0-tune(default)          parallel-netcdf/1.12.1(default)      qe/6.5(default)
gamess/2020R2(default)               parmetis/4.0.3(default)              r/4.0.0
gaussian16/c01(default)              petsc/3.13.1                         r/4.0.5(default)
gcc/10.2.0                           petsc/3.16.0(default)                ruby/2.7.1(default)
gcc/10.4.0(default)                  phdf5/1.10.6                         scotch/6.0.8(default)
geant4/10.6.2(default)               phdf5/1.10.7(default)                specfem3d_globe/7.0.2-tune(default)
gromacs/2021.2                       pio/2.5.3(default)                   superlu/5.2.1(default)
gromacs/2021.4(default)              ppohAMR_FDM/0.3.0(default)           superlu-dist/6.1.1(default)
gromacs-gcc/2022.4                   ppohAT/1.0.0(default)                superlu-mt/3.1(default)
hdf5/1.10.6                          ppohBEM/0.5.0(default)               tcs/1.2.24
hdf5/1.10.7(default)                 ppohBEM_AT/0.1.0(default)            tcs/1.2.25
jhpcn-df/1.1.0(default)              ppohDEM_util/1.0.0(default)          tcs/1.2.26
kmath_random/1.1(default)            ppohFDM/0.3.1(default)               tcs/1.2.27
lammps/20200505                      ppohFDM_AT/1.0.0(default)            tcs/1.2.30
lammps/7Aug19-tune(default)          ppohFEM/1.0.1(default)               tcs/1.2.31
lapack/3.10.0(default)               ppohFVM/0.3.0(default)               tcs/1.2.33
linsys-v/alpha(default)              ppohMATH_MP/1.0.0(default)           tcs/1.2.35
metis/5.1.0(default)                 ppohMATH_VIS/0.2.0(default)          tcs/1.2.37(default)
modylas/1.0.4(default)               py-chainer/7.2.0(default)            texlive/2021(default)
mpas/6.2-tune(default)               py-json5/0.9.4(default)              xabclib/1.03(default)
mt-metis/0.6.0(default)              py-keras/2.2.4(default)

------------------------------------ /home/center/local/app/a64fx/hpci/modulefiles ------------------------------------
```

コンパイル、実行環境構築

コンパイル、実行環境を整えるために、moduleコマンドを使用

$ module [オプション] *引数*

オプション	内容
avail	利用可能な環境の一覧を表示
list	現在ロードしている環境一覧を表示
load	指定した環境のロード
unload	指定した環境のアンロード
switch	環境のロードとアンロードを同時に実行
purge	環境を全てアンロード

プログラムのコンパイル

C または Fortran を使用される方
→ プログラムをコンパイルして、実行ファイル（バイナリ）を生成します。

C を使われる方

```
[w4900*a@flow-fx01 test]$ mpifccpx -Kopenmp hello_world.c
```
Return

Fortran を使われる方

```
[w4900*a@flow-fx01 test]$ mpifrtpx -Kopenmp hellow_world.f90
```
Return

その他、コンパイルオプションなどはマニュアルを参照ください。

Tips!
- Type I はログインノードと計算ノードのアーキテクチャが異なります。
- この場合、異なるシステム用のプログラムをコンパイルする必要があります。
- このようなコンパイルをクロスコンパイルといいます。

プログラムのコンパイル (Type Ⅰ)

富士通コンパイラの使用方法

ソースコードのタイプ	コンパイルターゲット	言語	呼び出しコマンド
非MPIコード	クロスコンパイル*1	C	fccpx
		Fortran	frtpx
	セルフコンパイル*2	C	fcc
		Fortran	frt
MPIコード	クロスコンパイル*1	C	mpifccpx
		Fortran	mpifrtpx
	セルフコンパイル*2	C	mpifcc
		Fortran	mpifrt

*1 クロスコンパイル：開発環境と実行環境が異なる場合のコンパイル
*2 セルフコンパイル：開発環境と実行環境が同一の場合のコンパイル

コンパイルオプション例（Type Ⅰ、参考情報）

富士通コンパイラでよく使うコンパイルオプション一覧

タイプ	言語	Intel オプション	内容
共通	共通	-Kopenmp	OpenMPを有効化
デバッグ オプション	共通	-g	実行ファイルにソース コードの情報を付与
		-Nquickdbg	コンパイル時にバグの元になりそうな 箇所等を警告
	Fortran	-Eg	実行中デバッグを有効化(遅くなる)
最適化 オプション	共通	-O0、-O1、-O2、-O3	数字が大きいほど積極的な 最適化を適用
		-Kfast	CPUアーキテクチャを考慮した様々な 最適化を有効化

プログラムのコンパイル (TypeⅡ)

IntelおよびGNUコンパイラがインストール済み

ソースコードのタイプ	コンパイラ開発元	言語	呼び出しコマンド
非MPIコード	Intel	C	icc
		Fortran	ifort
	GNU	C	gcc
		Fortran	gfortran
MPIコード	Intel	C	mpiicc
		Fortran	mpiifort
	GNU	C	mpicc
		Fortran	mpif77 or mpif90

IntelでC言語のコードをコンパイルする場合
$ icc [オプション] “ソースコード.c” –o “出力ファイル名”
GNUでFortran90/95のコードをコンパイルする場合
$ gfortran [オプション] -free-line-length-none “ソースコード.f90” –o “出力ファイル名”

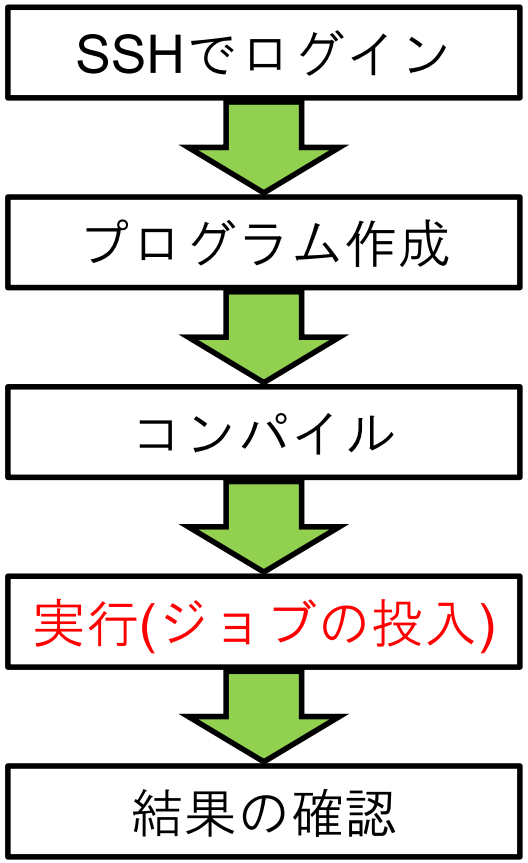
-free-line-length-none : Fortranの自由形式を有効化

コンパイルオプション例（参考情報）

IntelおよびGNUでよく使うコンパイルオプション一覧

タイプ	言語	Intel オプション	GNU オプション	内容
共通	共通	-qopenmp	-fopenmp	OpenMPを有効化
デバッグ オプション	共通	-g		実行ファイルにソースコードの情報を付与
		-Wall		コンパイル時にバグの元になりそうな箇所等を警告
		-traceback	-fbacktrace	実行時にエラー発生個所を特定
	Fortran	-check bounds	-fbounds-check	実行中に初期化していない値へのアクセスなどを検知
最適化 オプション	共通	-O0、-O1、-O2、-O3		数字が大きいほど積極的な最適化を適用
		-xHost	-march=native	CPUアーキテクチャを考慮した最適化

スパコンを使うための手順



SSH、鍵認証、ソフトのインストール、鍵ペア作成、ログイン

エディタ(emacs, vi, nano)

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

スパコンでプログラムを実行する方法

ジョブスケジューラーを使用

- スパコンは複数のユーザーが共有する資源
- ユーザーの要求に合わせて、資源を割り当てる仕組みが必要
  →ジョブスケジューラ
- レストランでお客様を席に案内するイメージ？

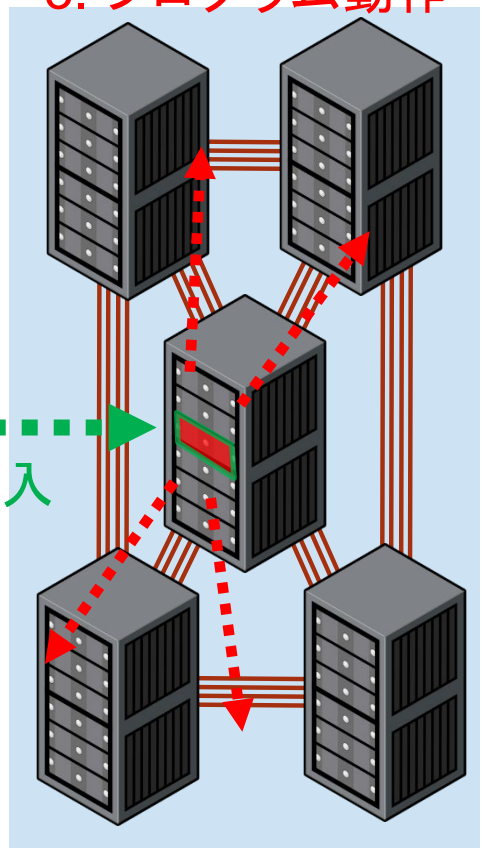
端末

1. ログイン
SSH

ログインノード

2. ジョブ投入

3. プログラム動作



計算ノード
= 本体

- ジョブスケジューラーに資源を要求
  → ジョブスクリプトを記述

ジョブスクリプト＝ スパコンへの指示書

プログラムをコンパイルしたあとにログインノード上で
直接 プログラム実行することはしないでください！

ー 負荷がかかると、他のユーザーの同時作業の妨げとなります

かわりに計算ノードに計算をしてもらうための「指示書」が必要となります。

hello_world.sh

```
#!/bin/bash
#PJM  -L  rscunit=fx
#PJM  -L  rscgrp=fx-workshop        [リソースグループ(後述)]
#PJM  -L  node=2                    [使用するノードの数]
#PJM  --mpi proc=8                  [プロセス数]
#PJM  -o  test.txt                  [出力結果格納ファイル名*1]
#PJM  -L  elapse=0:01:00            [実行時間上限（1分）]
#PJM  -j                            [エラー出力を標準出力にマージ]

export OMP_NUM_THREADS=12           [スレッド数]
mpirun ./a.out                      [プログラム実行]
```

*1ファイル名を指定すると、
古い結果は新しい結果で
上書きされます。

プロセス数×スレッド数 = ノード数×ノードたりのコア数(Type1は48)

参考情報：Type Ⅰのリソースグループ

リソースグループ毎のノード数および最長実行時間一覧

リソースグループ名	ノード数	最長実行時間	説明
fx-debug	1～36	60分	デバッグ用
fx-small	1～24	168時間	通常使うリソースグループ
fx-middle	12～96	72時間	
fx-large	96～192		
fx-xlarge	96～768	24時間	
fx-special	7,304	制限なし	事前予約制
fx-middle2	1～96	72時間	優先実行、事前予約制
fx-iteractive	1～4	24時間	会話型
fx-workshop	1~12	20分	講習会用

■ 講習会ではfw-workshopをご利用ください。

スパコンでのジョブの実行

作成したジョブスクリプトを投入、
計算ノードで実行してもらいます。

```
[w4900*a@flow-fx01 test]$ pjsub hellow_world.sh Return
```

実行されているジョブを確認します。
（一瞬で実行が終わるので、ジョブ一覧に出ないかも）

```
[w4900*a@flow-fx01 test]$ pjstat Return
```

得られた結果を表示します。

```
[w4900*a@flow-fx01 test]$ more test.txt Return
```

[Return で下の方に進めていくことができます]

ジョブの管理コマンド

コマンド	内容	使い方
pjsub	ジョブの投入	$pjsub “script.sh”
pjdel	ジョブの削除	$pjdel “job ID”
pjstat	ジョブの状態	$pjstat

注＊これらのコマンドはスパコンによって異なる

pjstatのオプション

■ -H　　：終了したジョブの確認

参考情報：計算ノードで直接操作したい場合

インタラクティブジョブ
対話型のジョブ デバッグなどに便利

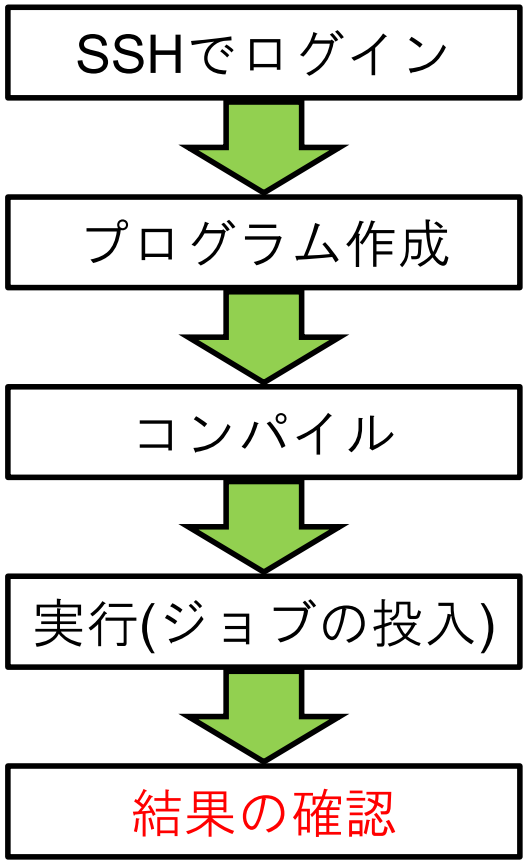
手元で実行しているのと同様の挙動

```
[w4900*a@flow-fx01 ~]$ pjsub --interact -L rscgrp=fx-interactive,node=2
[INFO] PJM 0000 pjsub Job 517079 submitted.
[INFO] PJM 0081 .connected.
[INFO] PJM 0082 pjsub Interactive job 517079 started.
[w4900*a@fx0391 ~]$
```

ログインノード(flow-fx*)から計算ノード(fx****)へログインしたような状態で
プログラムの実行が可能

ログインノードとは違い専用の領域にいるので、負荷のある実行をして良い
注意！：クロスコンパイルじゃなくなるので、コンパイル時のコマンドが変わります！
mpifccpx→mpifcc、mpifrtpx→mpifrt

スパコンを使うための手順



SSH、鍵認証、鍵ペア作成、ログイン

エディタ(emacs, vi, nano)

環境モジュール、コンパイルオプション

ジョブスクリプトの作成、ジョブ管理

計算結果のファイルを手元に転送する

scpを使ってスパコン上のファイルを手元のマシンにコピー

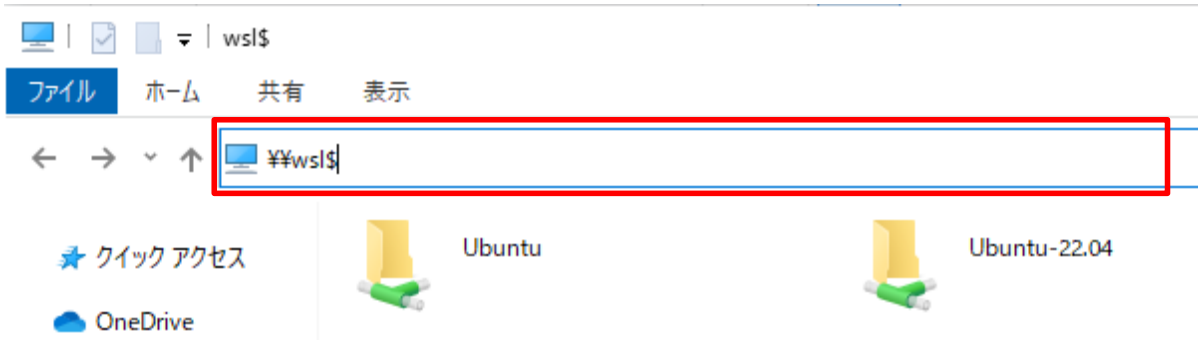
scp -i “*秘密鍵*“ *ユーザーID*@flow-fx.cc.nagoya-u.ac.jp:~/test/test.txt ./result.txt

コピー元　　コピー先

```
$ scp -i ~/.ssh/id_rsa w4900*a@flow-fx.cc.nagoya-u.ac.jp:~/test/test.txt ./result.txt
$ ls
result.txt
```

ローカルのデータをスパコンにコピーする場合は、
先のコマンドのコピー元とコピー先を入れ替える形

Windows OSからファイルを確認する方法
Ubuntu on Windows：エクスプローラーのパスに
¥¥wsl$と入力→ubuntuのルートにアクセス可能
Cygwin：C:¥cygwinの下がcygwinのディレクトリ



おまけ1 sshfs (Mac、Cygwin、Ubuntu on windows)

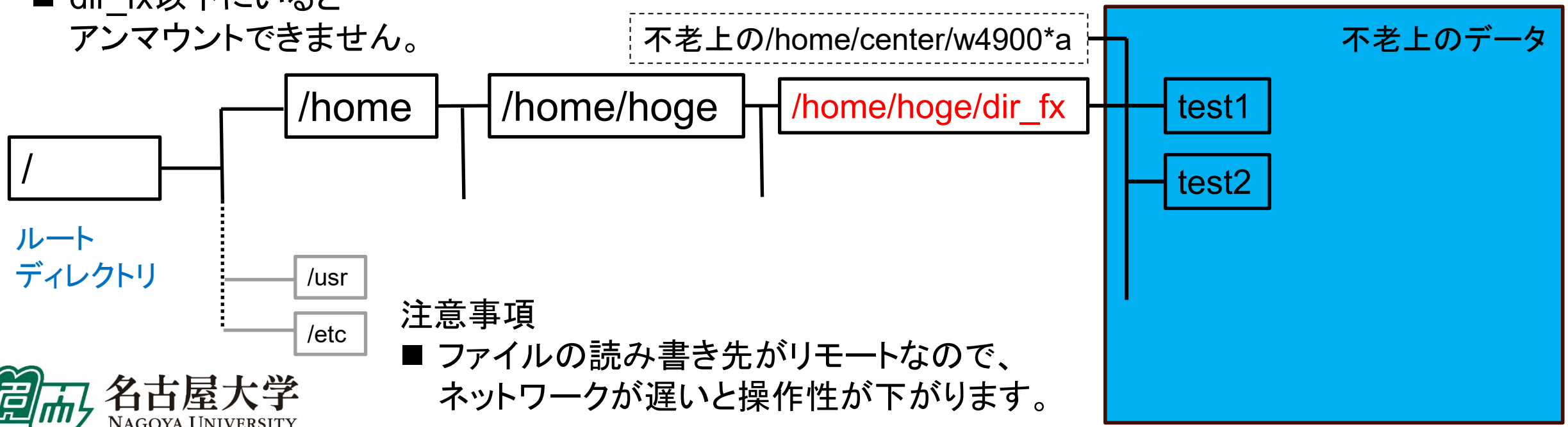
sshfsを使用すると、ローカルのディレクトリにリモートのディレクトリをマウントできます。

```
$ sshfs -o ssh_command='ssh -i /home/hoge/.ssh/id_rsa'  w4900*a@/home/center/w4900*a ~/dir_fx
```

- マウント:リモートのディレクトリをローカルのディレクトリに繋げる操作
- コピー等の操作なく、リモートのファイル操作が可能
- アンマウント(接続を切る操作)は以下

```
$ fusermount -u ~/dir_fx
```

- dir_fx以下にいると
  アンマウントできません。



注意事項

- ファイルの読み書き先がリモートなので、
  ネットワークが遅いと操作性が下がります。

おまけ2 X11 forwarding (Mac、Cygwin、Ubuntu on windows)

リモートのグラフィカルな画面をssh経由でローカルに表示できます(上級者向け)。

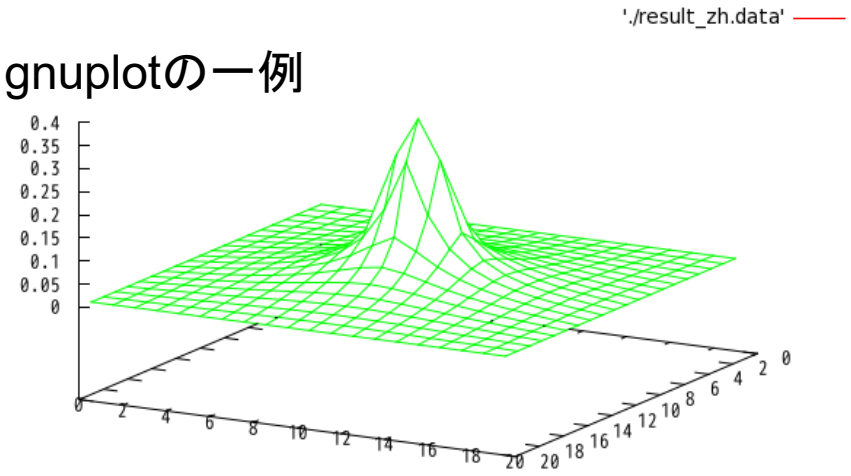
```
$ ssh -i ~/.ssh/id_rsa ユーザーID@スパコンホスト名
```

↓ -CXオプションを追加

```
$ ssh -CX -i ~/.ssh/id_rsa ユーザーID@スパコンホスト名
```

注意
Cygwin、ubuntu on windowsでは以下のソフトウェア
のインストールが必要
Cygwin：X11（Cygwinが管理、openssh（p39-44)と同じ手順）、
Cygwin X(Cygwinとは別のパッケージ)
ubuntu on windows：Xmingのインストール、
$apt install x11-appsの実行

gnuplotの一例



お疲れ様でした！

この後は、質問等ございましたら受け付けます。
内容によってはブレイクアウトルームを用意いたします。