内容に関する質問は
katagiri@cc.nagoya-u.ac.jp
まで

2024年2月28日（水）10:00～17:30
Zoomによるオンライン開催

第66回 スーパーコンピュータ「不老」利用型講習会 MPI（初級）

名古屋大学情報基盤センター　片桐孝洋

プログラム

- 9:30 - 10:00　受付（ZOOM接続可能）
- 10:00 - 10:30　端末設定など（演習）
  - スーパーコンピュータ「不老」Type I サブシステム（以降、FX1000システ ム）へのログイン
  - FX1000システムへのジョブの投入方法
- 10 :30 – 12 :00（演習）
  - 名古屋大学情報基盤センターの計算機および利用形態
  - FX1000システムの計算機構成、利用方法
  - サンプルプログラムの実行
- 13:30　- 15:00　並列プログラミングの基本（座学）
  - 並列計算の基礎、性能評価指標、アムダールの法則
  - MPIインターフェース説明、集団通信関数（コレクティブ通信）
  - データ分散方式
  - 先進的並列化技法：
    - ピュアMPI実行、ハイブリッドMPI実行、NUMA最適化、など

プログラム

- 15:15 - 17:00 MPIプログラム並列化実習（演習）
  - 行列-行列積の並列アルゴリズム
  - 行列-行列積の並列化実習１（簡易並列化方式での演習）
  - 行列-行列積の並列化実習２（完全並列化方式での演習）
- 17:00 - 17:30 自由演習、および、スパコン利用相談会

名大情報基盤センターのスパコンのご紹介

スーパーコンピュータ「不老」の役割

1. 全国共同利用・共同研究拠点として学内外へ計算資源提供
   - **全国共同利用・共同研究拠点**として国が位置づけ
     - 全国の研究者の世界トップレベル研究を強力に支援

世界トップレベル研究の支援

HPCI (High Performance Computing Infrastructure) 利用者

国策スパコン利用支援

名大拠点利用者

JHPCN利用者

名大「不老」Type I システム（富岳型ノード）

スーパーコンピュータ「富岳」

簡便な移行支援

導入支援／高性能化／特殊処理／長時間実行

世界トップレベル研究成果創出

2. ものつくり企業支援（地域イノベーションコア形成）
   - **産業利用制度（公開、非公開）**
   - 計算機利用型講習会による並列処理・大規模計算普及（地域特有の中小企業支援）
3. 新しい計算需要に向けたサービス開拓
   - **データサイエンス（ビッグデータ）、AI基盤の提供による新サービスの開拓**
4. 指定国立大学として重要な役割
   - **数理・データ科学教育**
   - 人材育成・研究力強化・社会との連携

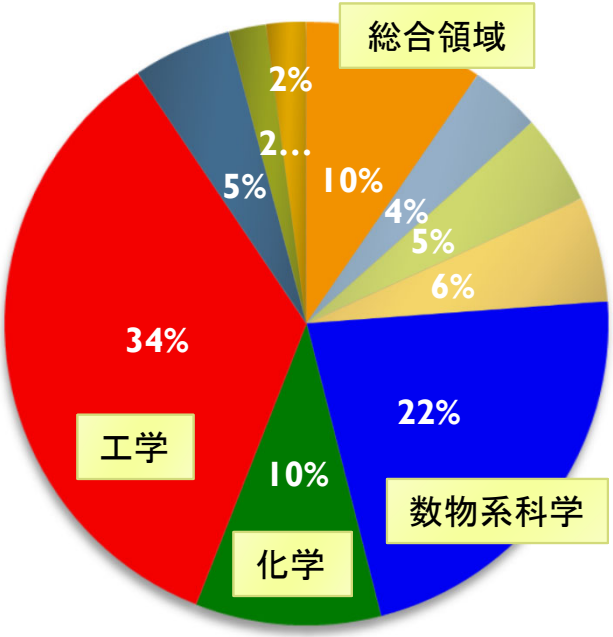
・数理データ科学分野の人材育成
・AI技術基盤提供（大規模AI計算基盤、大規模ストレージ、サイバーフィジカル）
・技術コンサルティング・・・等による社会貢献

ユーザの研究分野の変遷

※2020年10月現在

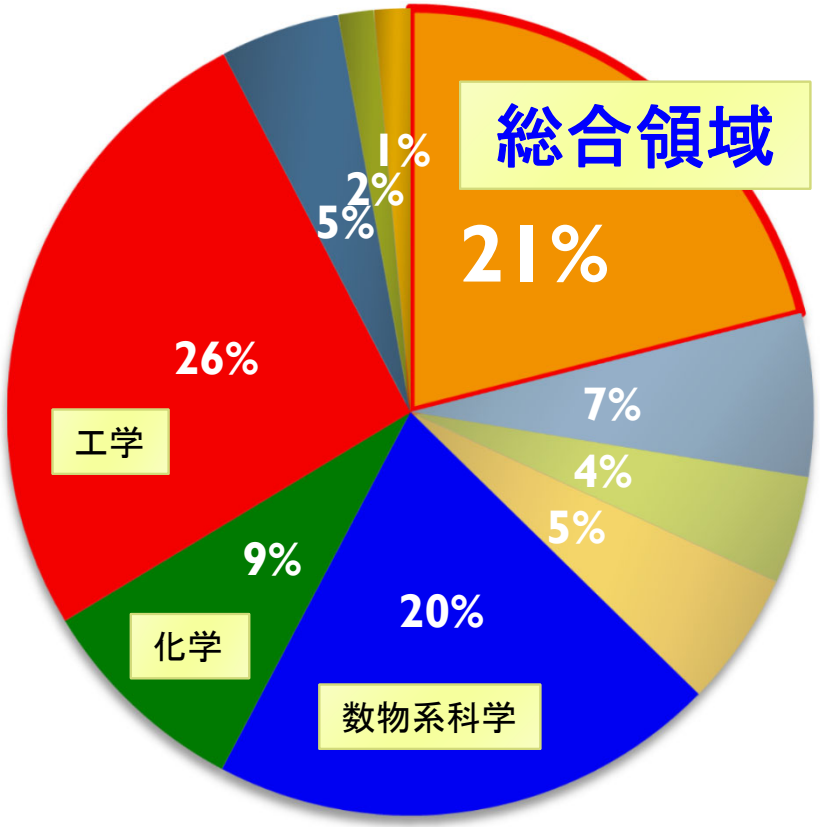
スーパーコンピュータ「不老」

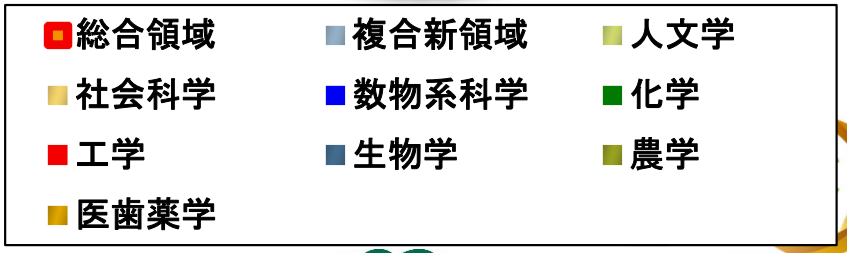
旧システム（FX100）



「**総合領域**」
の割合向上
10% → **21%**

「**情報学**」
分野ユーザ数
31 → **105**





AI・データサイエンス研究が増加？

スーパーコンピュータ「不老」の特徴

- 以下の新しいスパコン利用法を提供しています:
  - **①Type Ⅰサブシステム(「富岳」型)による超並列処理**
  - **②Type Ⅱサブシステム(GPUクラスタ)による大規模機械学習**
  - **③ 数値計算 ＋ AI 処理**
  - **④ ①～③のシームレスなデータ可視化**
  - **⑤ ①～④で必要な大規模・堅牢なデータ蓄積**

- ④に必要なTypeⅢサブシステム(大規模共有メモリ(48TB))、精細／遠隔可視化装置が、伝統的に名古屋大学は充実
- ⑤のコールドストレージ(100年保存可能な光ディスク)搭載スパコンは業界初

名古屋大学情報基盤センターのスパコン利用の特徴

- **計画書提出なし／論文出版等の義務なく利用可能**
  - 名古屋大学拠点利用の場合（HPCI、JHPCN利用を除く）
  - 年間随時募集（ただし、提供資源がなくなった場合を除く）
  - アカデミックの方
    - 研究目的（平和利用）、予算確保（運営費、科研費等）が明確であれば、必ず利用承認されます
      - 研究目的の記載は数行
    - 成果報告書は１ページ
      - 事実上、発表・出版論文などの情報の登録をお願いするのみ
    - 申込書提出後、１週間程度で承認、アカウント発行
  - 民間利用の方
    - 課題の計画書提出が必要
    - 審査委員会で審議の上で承認（１～２週間程度）。その後、アカウント発行。
    - 報告書はアンケート程度

スーパーコンピュータ「富岳」との連携

1. 同型計算機・同一ソフトウェアスタック
   - 「不老」Type I サブシステムはCPUが「富岳」と同型
   - OS・コンパイラ等のソフトウェアスタックも同一と想定
     - ☞「不老」のほうがより進んだバージョンが提供される可能性があり
2. 国策ソフトウェアの「不老」Type I サブシステムのプリインストール・講習会実施
   - 国費開発ソフトウェアで「富岳」で動作するソフトウェアのいくつかを、一般財団法人　高度情報科学技術研究機構（RIST）の協力のもと、「不老」Type I で提供、ハンズオン講習会も実施
     - ☞講習会予定をご覧ください
3. 「富岳」向けチューニングと「富岳」への移行支援
   - コンサルティング・教員との共同研究で「富岳」向けと想定されるコードチューニング（☞「不老」Type I 向けと等価）を支援します

名古屋大学情報基盤センターのスパコンの歴史

スーパーコンピュータ「不老」導入

96 00 05 09 13 15 20 24

Fujitsu M-1800 → Fujitsu GP7000F/90 → Fujitsu HX600, M9000 → Fujitsu CX400 → CX400

メインフレーム系


Fujitsu HPC2500

スカラー型スパコン

アプリケーションサーバ


Fujitsu VPP500 → Fujitsu VPP5000 → Fujitsu FX1 → Fujitsu FX10 → FX100

ベクトル型スパコン


スーパーコンピュータ



複合型「不老」

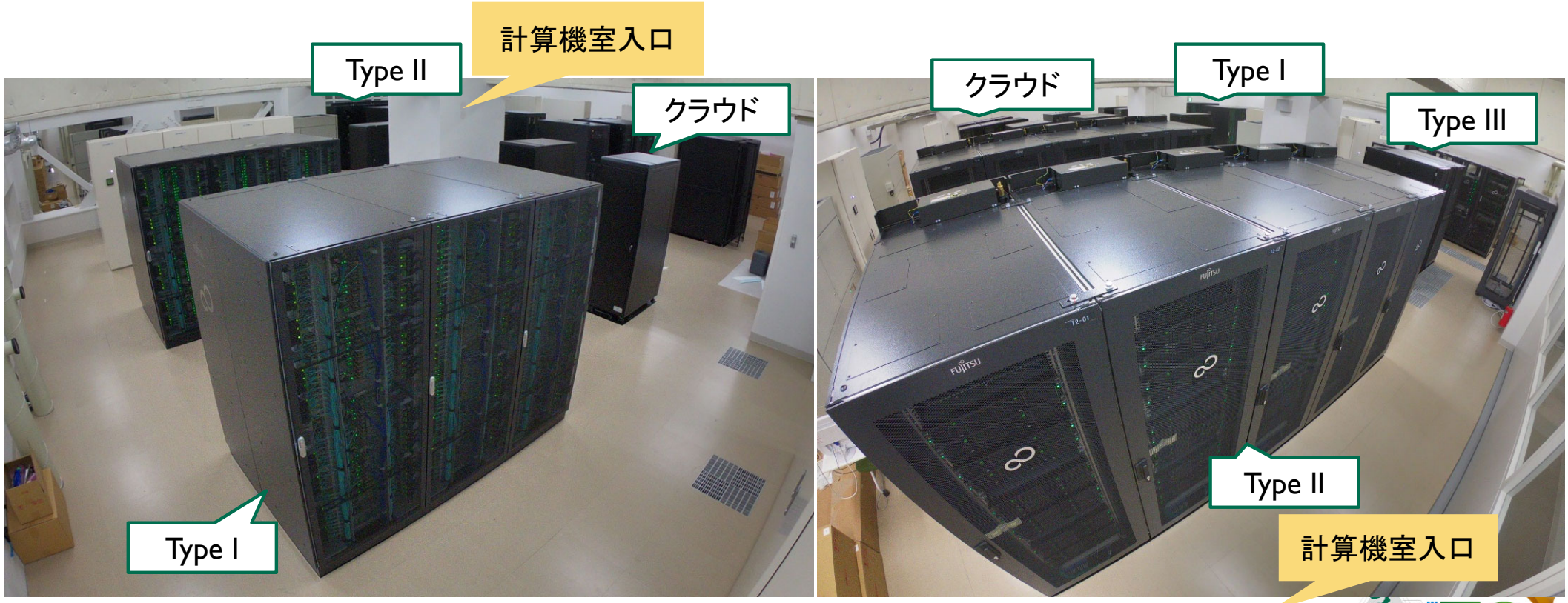
- Fujitsu CX2570
  GPUサーバ
  
- Fujitsu FX1000
  超並列
  
- HPE Superdome Flex
  大規模共有メモリ
- HPE ProLiant DL560
  クラウド

次期不老へ

- これまで約5年間隔でリプレイス
- 「不老」も5年弱（4年9ヶ月）の稼働を予定

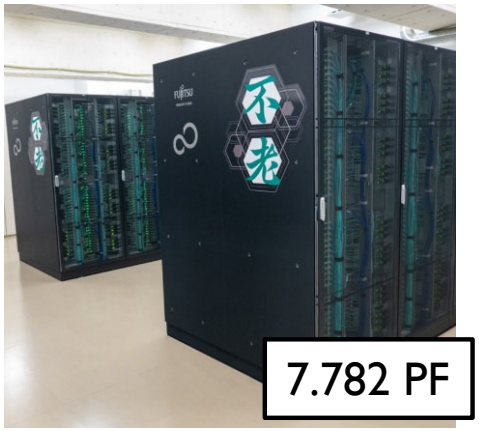
設置状況

- 7月1日、スーパーコンピュータ「不老」が稼働開始しました。現在も順調に稼働中です。
- 名古屋大学 情報基盤センター 本館地下1階の様子



スーパーコンピュータ「不老」
主な構成要素

Type I, II, III, クラウドの合計で15.886PFLOPS
（旧システムの約4倍）



Type Iサブシステム
FUJITSU Supercomputer FX1000
「富岳」型

Type IIサブシステム
FUJITSU Server PRIMERGY CX2570 M5
GPUスパコン

Type IIIサブシステム
HPE Superdome Flex
大容量メモリ・可視化

クラウドシステム
HPE ProLiant DL560
バッチ＆インタラクティブ

ホットストレージ
FUJITSU PRIMGERY RX2540 M5
FUJITSU ETERNUS AF250 S2
DDN SFA18KE
DDN SS9012

コールドストレージ
SONY PetaSite 拡張型 Library

性能諸元（主要サブシステム群）

		Type I	Type II	Type III	クラウド
ノードあたり	CPU	A64FX ×1 (Armv8.2-A + SVE) 48+2コア、2.2GHz	Xeon Gold 6230×2 (Cascade Lake) 20コア、2.10-3.90 GHz	Xeon Platinum 8280M×16 (Cascade Lake) 28コア、2.70-4.00 GHz	Xeon Gold 6230×4 (Cascade Lake) 20コア、2.10-3.90 GHz
	メインメモリ	HBM2, 32GB	DDR4, 384GB	DDR4, 24TB	DDR4, 384GB
	GPU	-	Tesla V100×4 (Volta) HBM2, 32GB	Quadro RTX6000×4 (Turing) GDDR6, 24GB	-
	理論性能	3.3792 TFLOPS(DP) 1,024 GB/s	・CPU 1.344 TFLOPS(DP)×2 140.784 GB/s×2 ・GPU 7.8 TFLOPS(DP)×4 900 GB/s×4	・CPU 2.4192 TFLOPS(DP)×16 140.784 GB/s×16	1.344 TFLOPS(DP)×4 140.784 GB/s×4
ノード数		2,304	221	2	100
ノード間接続		TofuインターコネクトD	InfiniBand EDR ×2	InfiniBand EDR	InfiniBand EDR
総理論性能		7.782 PFLOPS(DP) 2.359 PB/s	7.489 PFLOPS(DP) 857.8 TB/s	77.414 TFLOPS(DP) 2.253 TB/s	537.6 TFLOPS(DP) 56.314 TB/s
冷却方式		水冷	水冷	空冷	空冷

消費電力・省電力対策

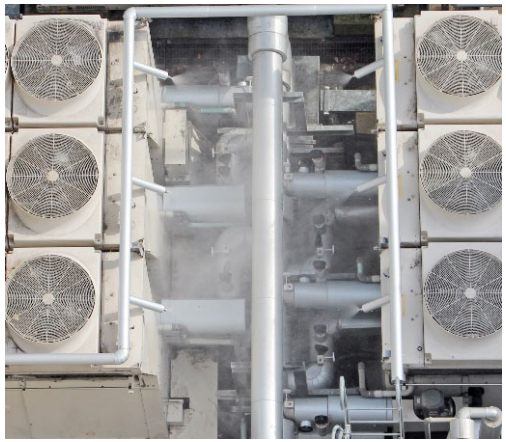
最大消費電力

サブシステム名	消費電力
Typelサブシステム	628.1kVA
TypeIIサブシステム	393.5kVA
TypeIIIサブシステム	21.6kVA
クラウドシステム	93.0kVA
ストレージ	49.9kVA
フロントエンド	19.6kVA
運用管理システム他	52.3kVA
冷却設備	641.9kVA
合　計	1,899.9kVA

電力可視化

湧水を用いた冷却

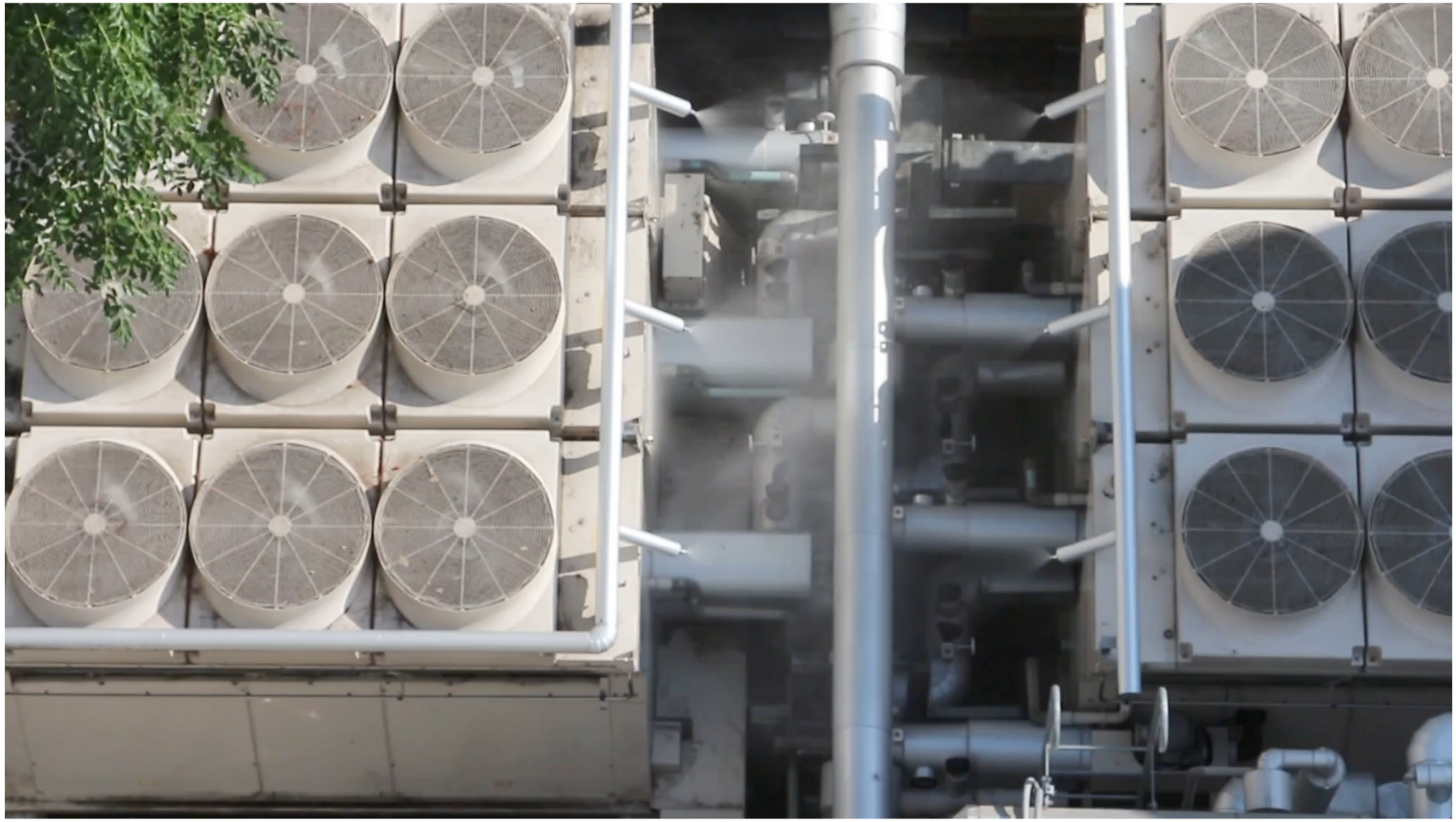
- 地下の湧水を活用したら総合評価時加点
- 屋外チラーに散水して冷却



湧水による冷却システム

- 情報基盤センターの地下は夏季でも18℃程度の湧き水が毎分30L程度湧く
- この湧き水は、地下からポンプで吸い上げて雨水扱いで捨てていた
- 今回の仕様で、湧き水を冷熱源として使用する場合は加点
  - 冷却水としての利用許可・水質検査済み

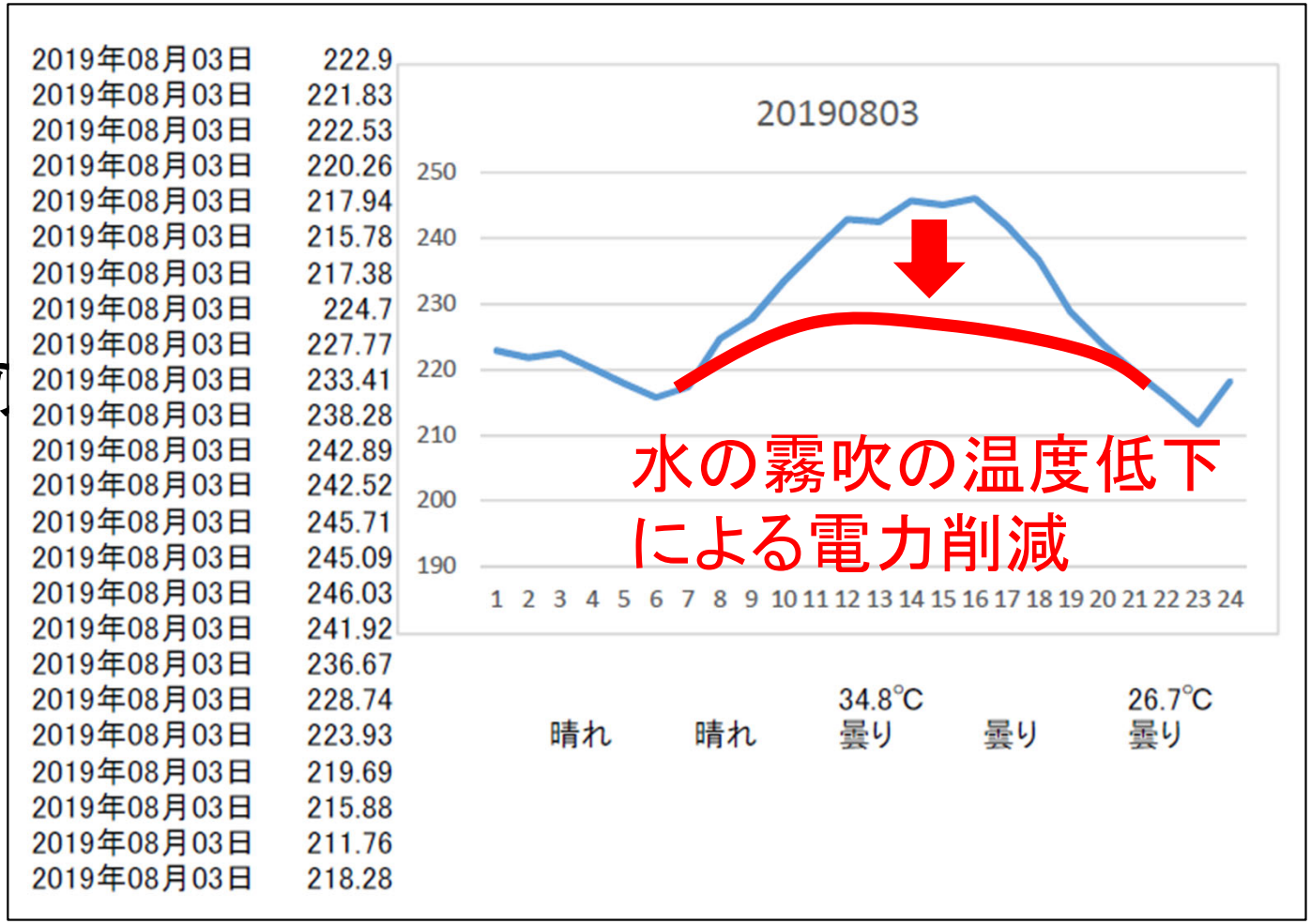
湧水による冷却システム



湧水による冷却システム

- 気温の高い4月から11月の間で利用
- 夏季の1日（2019年8月3日）の（旧）FX100システムの水冷チラーの電気使用量(KW)

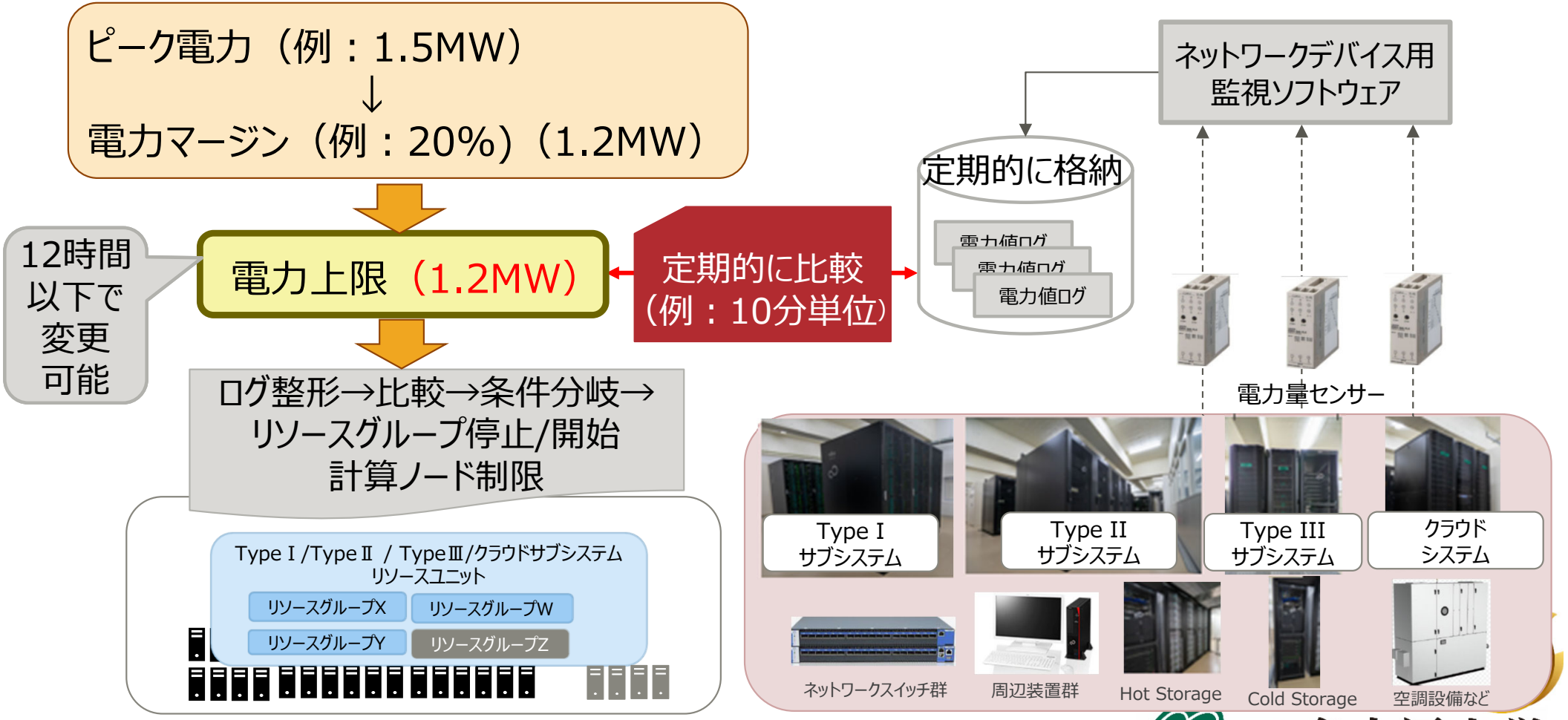
2019年08月03日	222.9
2019年08月03日	221.83
2019年08月03日	222.53
2019年08月03日	220.26
2019年08月03日	217.94
2019年08月03日	215.78
2019年08月03日	217.38
2019年08月03日	224.7
2019年08月03日	227.77
2019年08月03日	233.41
2019年08月03日	238.28
2019年08月03日	242.89
2019年08月03日	242.52
2019年08月03日	245.71
2019年08月03日	245.09
2019年08月03日	246.03
2019年08月03日	241.92
2019年08月03日	236.67
2019年08月03日	228.74
2019年08月03日	223.93
2019年08月03日	219.69
2019年08月03日	215.88
2019年08月03日	211.76
2019年08月03日	218.28



年間数百万円程度の電気代削減を予想

使用最大電力の動的制御機構

- 監視ソフトウェアから一定時間毎に電力値を取得
- 出力された電力値と、あらかじめ規定したシステム全体の使用最大電力の上限値を比較し、最大電力の上限を超えないよう、計算ノードやジョブ実行可能範囲を制限

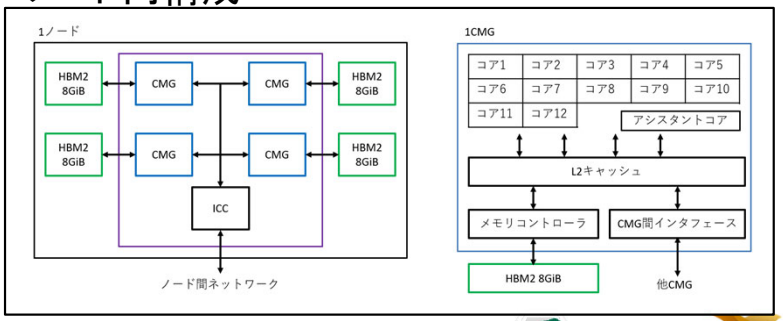


各サブシステムの仕様と特徴：Type I サブシステム

機種名		FUJITSU Supercomputer PRIMEHPC FX1000
計算ノード	CPU	A64FX (Armv8.2-A + SVE), 48コア+2アシスタントコア(I/O兼計算ノードは48コア+ 4アシスタントコア), 2.2GHz, 4ソケット相当のNUMA
	メインメモリ	HBM2, 32GiB
	理論演算性能	倍精度 3.3792 TFLOPS, 単精度 6.7584 TFLOPS, 半精度 13.5168 TFLOPS
	メモリバンド幅	1,024 GB/s (1CMG=12コアあたり256 GB/s, 1CPU=4CMG)
ノード数、総コア数		2,304ノード, 110,592コア (+4,800アシスタントコア)
総理論演算性能		7.782 PFLOPS
総メモリ容量		72 TiB
ノード間 インターコネクト		TofuインターコネクトD 各ノードは周囲の隣接ノードへ同時に合計 40.8 GB/s × 双方向 で通信可能(1リンク当たり 6.8 GB/s × 双方向, 6リンク同時通信可能)
ユーザ用 ローカルストレージ		なし
冷却方式		水冷

- 世界初正式運用の スーパーコンピュータ「富岳」型 システム
- 自己開発のMPIプログラム向き
- 超並列処理用
- AIツールも提供

ノード内構成

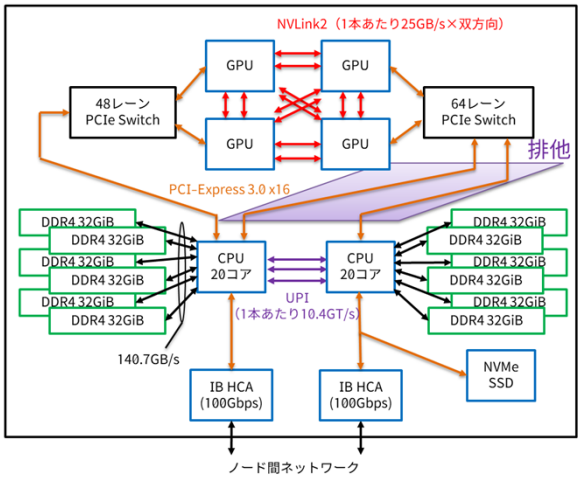


各サブシステムの仕様と特徴：Type II サブシステム

機種名		FUJITSU Server PRIMERGY CX2570 M5
計算ノード	CPU	Intel Xeon Gold 6230, 20コア, 2.10 - 3.90 GHz × 2 ソケット
	GPU	NVIDIA Tesla V100 (Volta) SXM2, 2,560 FP64コア, up to 1,530 MHz × 4ソケット
	メモリ	メインメモリ(DDR4 2933 MHz)：384 GiB (32 GiB × 6 枚 × 2 ソケット) デバイスメモリ(HBM2)：32 GiB × 4 ソケット
	理論演算性能	倍精度 33.888 TFLOPS (CPU 1.344 TFLOPS × 2 ソケット, GPU 7.8 TFLOPS × 4 ソケット)
	メモリバンド幅	メインメモリ 281.5 GB/s (23.464 GB/s × 6 枚 × 2 ソケット) デバイスメモリ 900 GB/s × 4 ソケット
	GPU間接続	NVLINK2 (1GPUから他の3GPUに対してそれぞれ50GB/s×双方向)
	CPU-GPU間接続	PCI-Express 3.0 (x16)
ノード数、総コア数		221ノード、8,840 CPUコア + 2,263,040 FP64 GPUコア
総理論演算性能		7.489 PFLOPS (CPU 0.594 PFLOPS, GPU 6.895 PFLOPS)
総メモリ容量		メインメモリ 82.875 TiB、デバイスメモリ 28.288 TiB
ノード間インターコネクト		InfiniBand EDR 100 Gbps × 2, 200 Gbps
ユーザ用 ローカルストレージ		NVMe SSD 6.4TB, 一部ノードにて BeeGFS/BeeOND/NVMesh (ローカルストレージを使用した共有ファイルシステム) を提供
冷却方式		水冷

- データサイエンス研究、機械学習用のGPUクラスタ型
- 最新GPU (Volta) 4台／ノード
- 充実したAIツール
- 高速SSDローカルディスク

ノード内構成

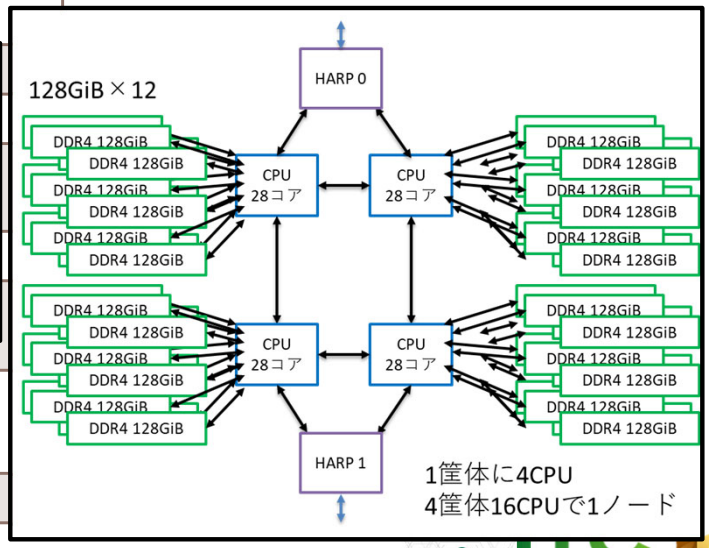


各サブシステムの仕様と特徴：Type III サブシステム

機種名		HPE Superdome Flex
計算ノード	CPU	Intel Xeon Platinum 8280M, 28コア, 2.70 - 4.00 GHz × 16 ソケット
	GPU	NVIDIA Quadro RTX6000 × 4
	メモリ	メインメモリ(DDR4 2933 MHz)：24 TiB (128 GiB × 12枚 × 16ソケット) デバイスメモリ(GDDR6)：24 GiB × 4
	理論演算性能	倍精度 38.7072 TFLOPS (CPU 2419.2 TFLOPS × 16 ソケット)
	メモリバンド幅	メインメモリ 2252.544 GB/s (23.464 GB/s × 12枚(6チャネル) × 16ソケット)
	CPU-GPU間接続	PCI-Express 3.0 (x16)
ノード数		2
総理論演算性能		77.414 TFLOPS (38.7072 TFLOPS × 2 ノード)
総メインメモリ容量		48 TiB
ノード間インターコネクト		InfiniBand EDR 100 Gbps
ユーザ用 ローカルストレージ		一方のノードに102.4 TB SSD、 もう一方のノードに1008 TB 共有ストレージを接続
冷却方式		空冷

- 大規模共有メモリ（24TiB）
- プリポスト処理用・可視化処理用
- NICE DCVを用いたリモート可視化

ノード内構成

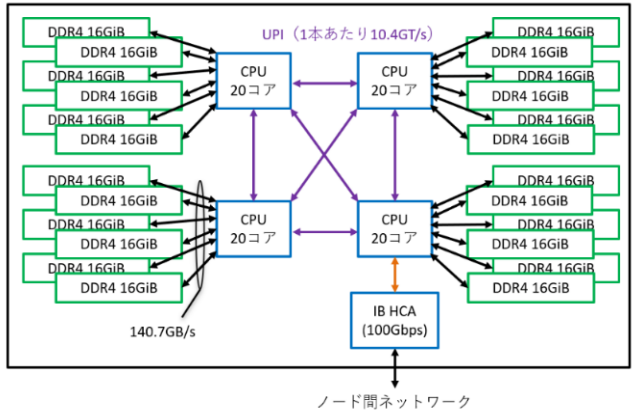


各サブシステムの仕様と特徴：クラウドシステム

機種名		HPE ProLiant DL560
計算ノード	CPU	Intel Xeon Gold 6230, 20コア, 2.10 - 3.90 GHz × 4 ソケット
	メモリ	メインメモリ(DDR4 2933 MHz) 384 GiB (16 GiB × 6 枚 × 4 ソケット)
	理論演算性能	倍精度 5.376 TFLOPS (1.344 TFLOPS × 4 ソケット)
	メモリバンド幅	メインメモリ 563.136 GB/s (23.464 GB/s × 6枚 × 4 ソケット)
ノード数		100
総理論演算性能		537.6 TFLOPS (5.376 TFLOPS × 100 ノード)
総メインメモリ容量		37.5 TiB
ノード間インターコネクト		InfiniBand EDR 100 Gbps
ユーザ用ローカルストレージ		なし
冷却方式		空冷

- 研究室クラスタから移行しやすいIntel CPU搭載システム
- 高いノードあたりCPU性能（4ソケット）
- 時刻を指定してのバッチジョブ・インタラクティブ利用が可能

ノード内構成



ホットストレージの仕様と特徴

メタデータサーバ(MDS)	
機種名	FUJITSU PRIMERGY RX2540 M5
CPU	Intel Xeon Gold 5222 (3.80GHz, 4コア) × 2
メインメモリ	DDR4 192 GiB
HDD	SAS 900 GB 10krpm × 2 (RAID1)
Interconnect	InfiniBand EDR × 2
SAN	FibreChannel 32 Gbps &tiems; 2
OS	RedHat Enterprise Linux
ノード数	4台
メタデータストレージサーバ(MDT)	
機種名	FUJITSU ETERNUS AF250 S2
SSD	RAID1+0 [4D+4M] × 2 + 2HS RAID1+0 [3D+3M] × 1 + 2HS
ノード数	1台

データストレージ(OSS/OST)	
機種名	DDN SFA18KE × 1台 DDN SS9012 × 10 台
HDD	NL-SAS 14TB 7.2krpm × 730、RAID6 [8D+2P] 30 Device × 24 DCR Pool + 10HS
Interconnect	InfiniBand EDR × 8
搭載セット数	4
総容量	
物理容量	40.32 PB (Global Spareを除く)
実効容量	約30.44PB

- HDD RAID
- 大容量：30.44 PB（実効容量）
- 超高速アクセス性能：384 GB/s

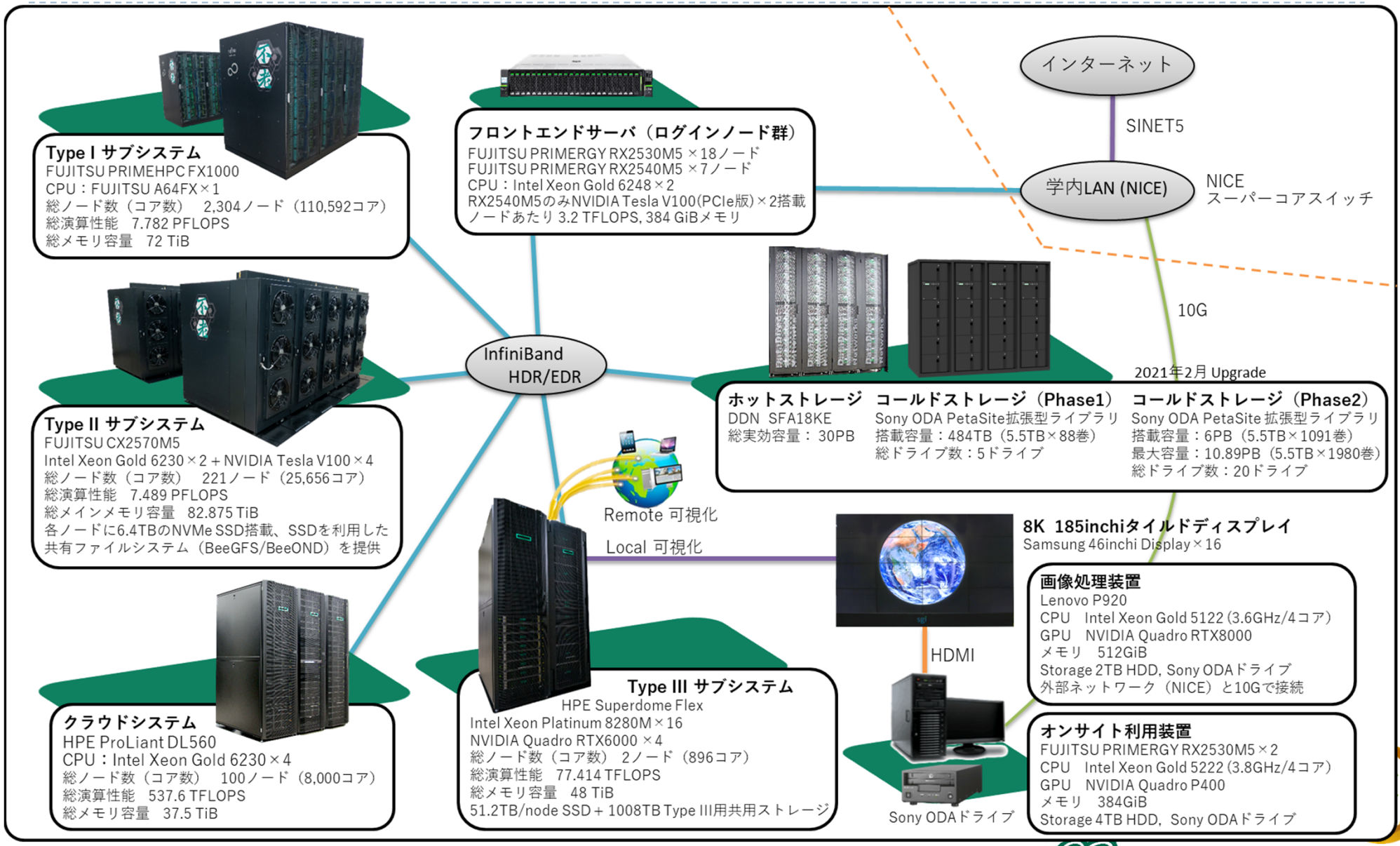
コールドストレージの仕様と特徴

フェーズ2: 2021年2月1日より稼働

機種名	PetaSite拡張型 Library
総スロット数 (最大搭載可能カートリッジ数)	1,980巻
総物理容量 / 最大搭載可能容量	**6 PB** / 10.89 PB
総ドライブ数	20
ODAサーバ数	4

- 6PBの大容量（スパコンで初）
- １度書き込み（追記）のみの光ディスクストレージ
- 実験データ等の長期データ保存用
- 理論上100年データ保持可能
- 水にぬれても読み出せる
- サービス終了後ユーザに光ディスクを返却

全体システム構成


インターネット
SINET5
学内LAN (NICE)
NICE
スーパーコアスイッチ
10G
2021年2月 Upgrade
InfiniBand
HDR/EDR
Type I サブシステム
FUJITSU PRIMEHPC FX1000
CPU：FUJITSU A64FX×1
総ノード数（コア数） 2,304ノード（110,592コア）
総演算性能 7.782 PFLOPS
総メモリ容量 72 TiB
フロントエンドサーバ（ログインノード群）
FUJITSU PRIMERGY RX2530M5 ×18ノード
FUJITSU PRIMERGY RX2540M5 ×7ノード
CPU：Intel Xeon Gold 6248×2
RX2540M5のみNVIDIA Tesla V100(PCIe版)×2搭載
ノードあたり 3.2 TFLOPS, 384 GiBメモリ
Type II サブシステム
FUJITSU CX2570M5
Intel Xeon Gold 6230×2＋NVIDIA Tesla V100×4
総ノード数（コア数） 221ノード（25,656コア）
総演算性能 7.489 PFLOPS
総メインメモリ容量 82.875 TiB
各ノードに6.4TBのNVMe SSD搭載、SSDを利用した
共有ファイルシステム（BeeGFS/BeeOND）を提供
ホットストレージ
DDN SFA18KE
総実効容量： 30PB
コールドストレージ（Phase1）
Sony ODA PetaSite拡張型ライブラリ
搭載容量：484TB（5.5TB×88巻）
総ドライブ数：5ドライブ
コールドストレージ（Phase2）
Sony ODA PetaSite 拡張型ライブラリ
搭載容量：6PB（5.5TB×1091巻）
最大容量：10.89PB（5.5TB×1980巻）
総ドライブ数：20ドライブ
Remote 可視化
Local 可視化
8K 185inchiタイルドディスプレイ
Samsung 46inchi Display×16
画像処理装置
Lenovo P920
CPU Intel Xeon Gold 5122 (3.6GHz/4コア)
GPU NVIDIA Quadro RTX8000
メモリ 512GiB
Storage 2TB HDD, Sony ODAドライブ
外部ネットワーク（NICE）と10Gで接続
HDMI
クラウドシステム
HPE ProLiant DL560
CPU：Intel Xeon Gold 6230×4
総ノード数（コア数） 100ノード（8,000コア）
総演算性能 537.6 TFLOPS
総メモリ容量 37.5 TiB
Type III サブシステム
HPE Superdome Flex
Intel Xeon Platinum 8280M×16
NVIDIA Quadro RTX6000 ×4
総ノード数（コア数） 2ノード（896コア）
総演算性能 77.414 TFLOPS
総メモリ容量 48 TiB
51.2TB/node SSD＋1008TB Type III用共用ストレージ
Sony ODAドライブ
オンサイト利用装置
FUJITSU PRIMERGY RX2530M5×2
CPU Intel Xeon Gold 5222 (3.8GHz/4コア)
GPU NVIDIA Quadro P400
メモリ 384GiB
Storage 4TB HDD, Sony ODAドライブ

高精細可視化システム

Type IIIサブシステム（HPE Superdome Flex）

77.4TFLOPS/48TiB MEM

Intel Xeon Platinum 8280M(2.7GHz,28Core)×16CPU×2　24TiB×2
NVIDIA Quadro RTX6000×4×2
HDD:実効容量500TB(RAID6)×2, NVMe:51.2TB×2

Quadro RTX6000

光ケーブル

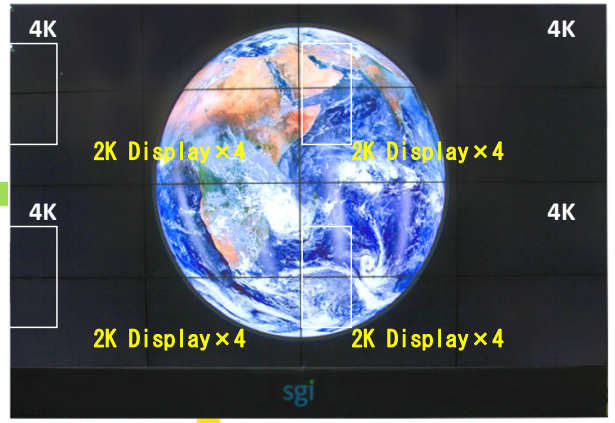
SINET

リモート
可視化

大規模共有メモリ:
24TiB ×2ノード

8Kタイルドディスプレイ

185inchi　8K高精細大画面タイルドディスプレイ
（総解像度：7680×4320）
Samsung 46inchi Display×16



HDMI

画像処理装置（Windows可視化サーバ）

Intel Xeon Gold 5122
　　3.6GHz 4 Cores, 512 GiB MEM
NVIDIA Quadro RTX8000
SATA SSD 2TB
光Disk 5.5TBドライブ　ODS-D380U

スーパーコンピュータ共有ストレージ

ホットストレージ
DDN　SFA18KE
総実行容量：30PB

コールドストレージ
Sony ODA PetaSite 拡張型ライブラリ
搭載容量：6PB（5.5TB×1091巻）
最大容量：10.89PB（5.5TB×1980巻）

スーパーコンピュータ「不老」ソフトウェア利用環境

プログラム開発環境など

		フロントエンド システム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウドシステム	画像処理 サーバ	オンサイト 利用装置	企業利用
Intel Parallel Studio Computing Suite	コンパイラ (Fortran, C/C++), プロファイラ/デバッガ, MPI, 数値計算ライブラリ	○		○	○	○			●
NVIDIA HPC SDK	コンパイラ (Fortran, C/C++, OpenACC, CUDA Fortran), プロファイラ/デバッガ, MPI, 数値計算ライブラリ	○		○					●
FUJITSU Technical Computing Suite	コンパイラ (Fortran,C/C++), プロファイラ/デバッガ, MPI, 数値計算ライブラリ	○	○						●
Arm Forge Proffesional	プロファイラ/デバッガ/最適化	○		○	○				●
NVIDIA CUDA SDK	GPU統合開発環境	○		○					●
Singularity	コンテナ環境	○		○					●
その他	GV, Gfortran, GCC, perl, Python, Ruby, R, Emacs, vi, nkf, etc.	○	○	○	○	○		○	●
	OpenGL	○		○	○	○		○	●

スーパーコンピュータ「不老」
ソフトウェア利用環境

ライブラリ利用環境

		フロントエンドシステム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウド サブシステム	画像処理 サーバ	オンサイト 利用装置	企業利用
数値計算ライブラリ	FFTW, SuperLU, SuperLU MT, SuperLU DIST, METIS, MT-METIS, ParMETIS, Scotch, PT-Scotch, PETSc, MUMPUS, Xabclib ppOpen-HPCライブラリ：ppOpen-APPL, ppOpen-AT, ppOpen-MATH 精度保障ライブラリ：LINSYS_V, DHPMM_F	○	○	○	○	○			●
入出力フォーマットライブラリ	NetCDF, Parallel netCDF, HDF5, JHPCN-DF	○	○	○	○	○			●
画像処理ソフトウエア	OpenCV, Geant4	○	○	○	○	○			●
機械学習ソフトウエア	Caffe, Chainer, Keras, PyTorch, TensorFlow, Theano, Mxnet, ONNX パッケージ：conda, Numpy, Scipy, scikit-image, pillow, matplotlib, jupyterlab	○	○	○	○	○			●

スーパーコンピュータ「不老」ソフトウェア利用環境

解析ソフトウェア利用環境

•*1 Type III サブシステムの会話型ノード(lm01)で利用可。
•*2 CPU並列版の他にGPU対応版が利用可。

		フロントエンドシステム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウド サブシステム	画像処理サーバ	オンサイト利用装置	企業利用
流体解析	OpenFOAM, FrontFlow blue/red		○	○	○	○			●
構造解析	LS-Dyna			○					
	FrontISTR		○	○	○	○			●
計算化学解析	AMBER		○	○ *2		○			
	Gaussian, Gamess, Gromacs, LAMMPS, NAMD		○	○ *2		○			●
	Modylas		○	○		○			●
メッシャー	Pointwise	○			○ *1				
統合ソフトウエア	HyperWorks			○ *2		○			

スーパーコンピュータ「不老」ソフトウェア利用環境

可視化ソフトウェア利用環境

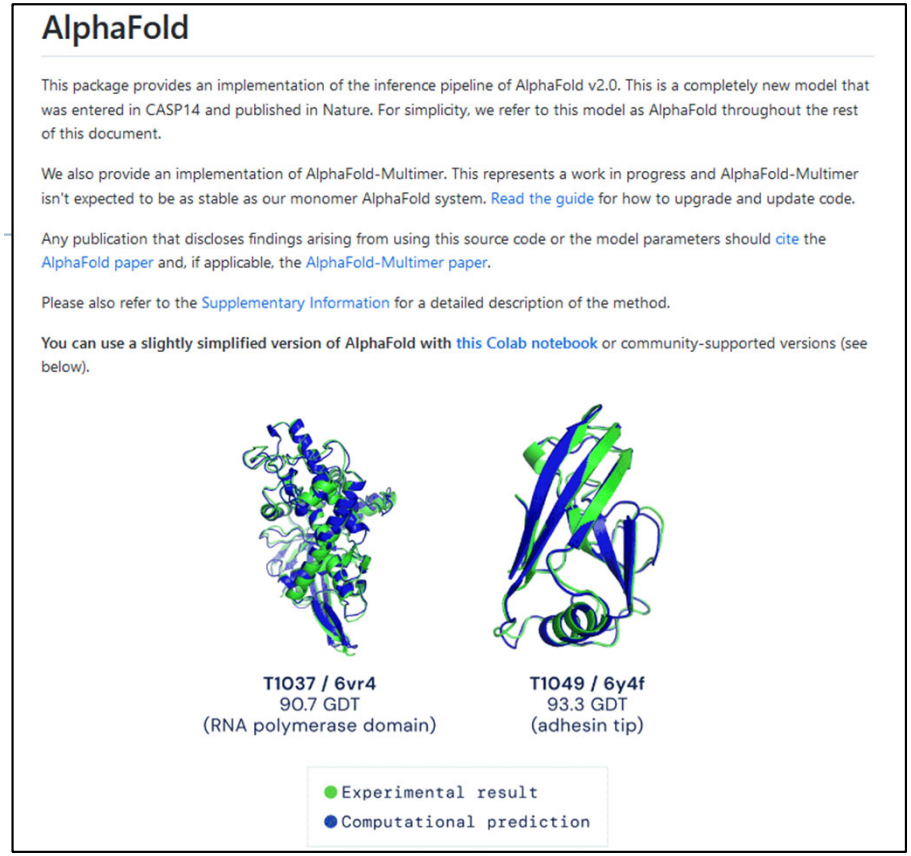
•*1 Type III サブシステムの会話型ノード(lm01)で利用可。

		フロントエンドシステム	Type I サブシステム	Type II サブシステム	Type III サブシステム	クラウドサブシステム	画像処理サーバ	オンサイト利用装置	企業利用
リモート可視化	NICE DCV	○			○ *1				●
可視化ソフトウエア	FieldView	○			○		○	○	
	AVS/Express, Paraview, POV-Ray, VMD	○			○		○	○	●
	3D AVS Player, ffmpeg, ffplay	○			○ *1		○	○	●
	IDL, ENVI	○			○		○		
	MicroAVS						○		●
	3dsMax, Visual Studio Pro						○		

OSS活用事例

- タンパク質構造予測ソフト **AlphaFold2** をTypeⅡサブシステム上で簡単に利用できるように整備 (2022年2月2日)

https://icts.nagoya-u.ac.jp/ja/sc/news/maintenance/2022-01-28-alphafold.html



(source: https://github.com/deepmind/alphafold)

- AlphaFoldの利用規約が更新され民間利用のユーザも利用可能
- 分散ノードのSSDを利用可能とするNVMESHにデータベース配置
  ⇒10時間以上かかるHHblits処理が10分に短縮

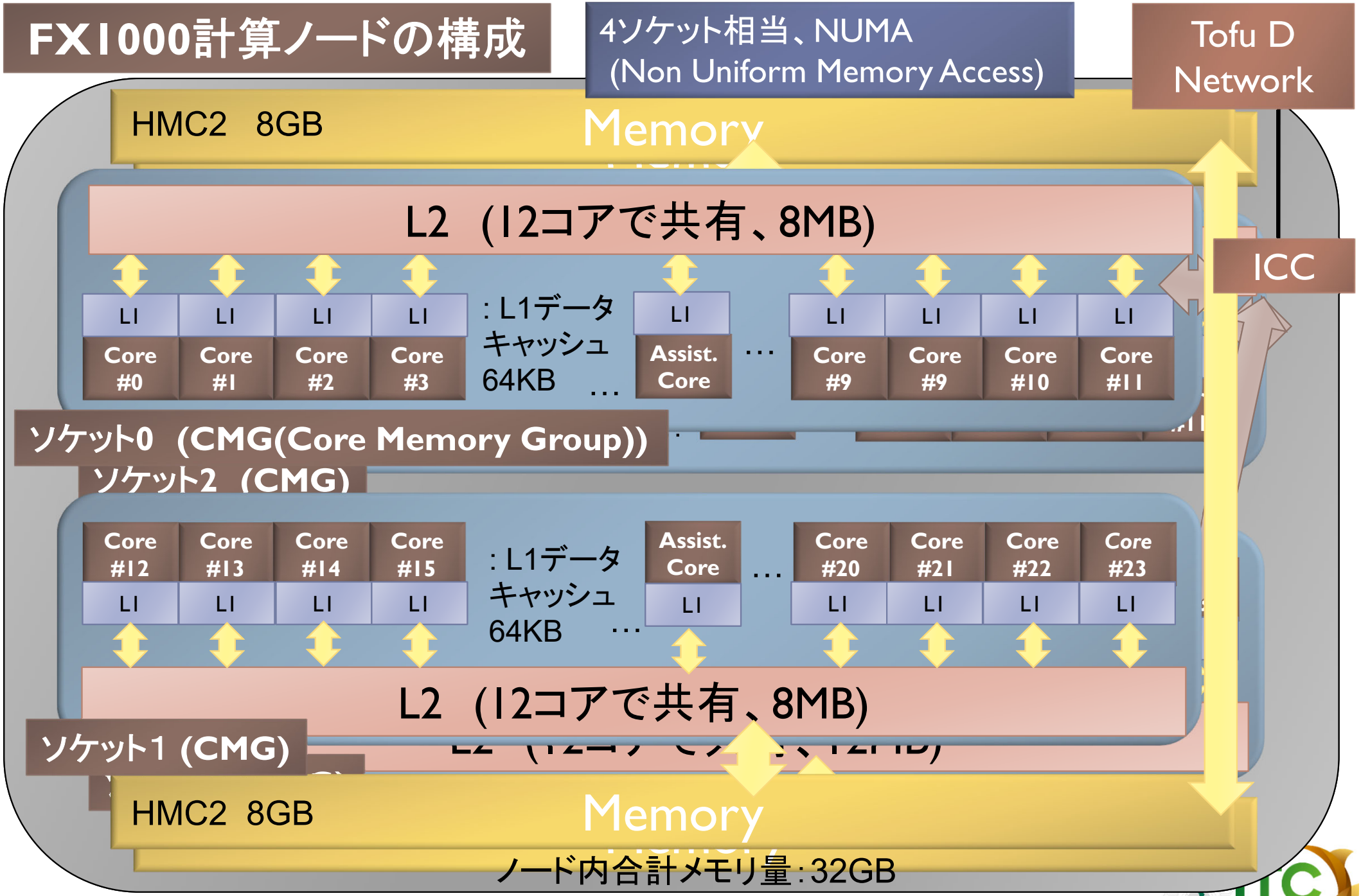
Fujitsu PRIMEHPC FX1000 の計算機アーキテクチャ

FX100とFX1000のアーキテクチャ比較

	FX100	FX1000(「不老」Typel)
演算能力／ノード	倍精度：1.011 TFLOPS 単精度：2.022 TFLOPS	倍精度 3.3792 TFLOPS, 単精度 6.7584 TFLOPS, 半精度 13.5168 TFLOPS
演算コア数	32	48
アシスタントコア	2	2
SIMD幅	256	512
SIMD命令	整数演算、ストライド＆間接ロード／ストアを強化	・8/16/32ビット整数演算 ・Combined Gather (128バイト アラインブロック単位)
L1Dキャッシュ／コア	64KB、4ウェイ	64KB、4ウェイ
L2キャッシュ／ノード	24MB	32MB, 16ウェイ/CMG
メモリバンド幅	480GB/秒	1024GB/秒

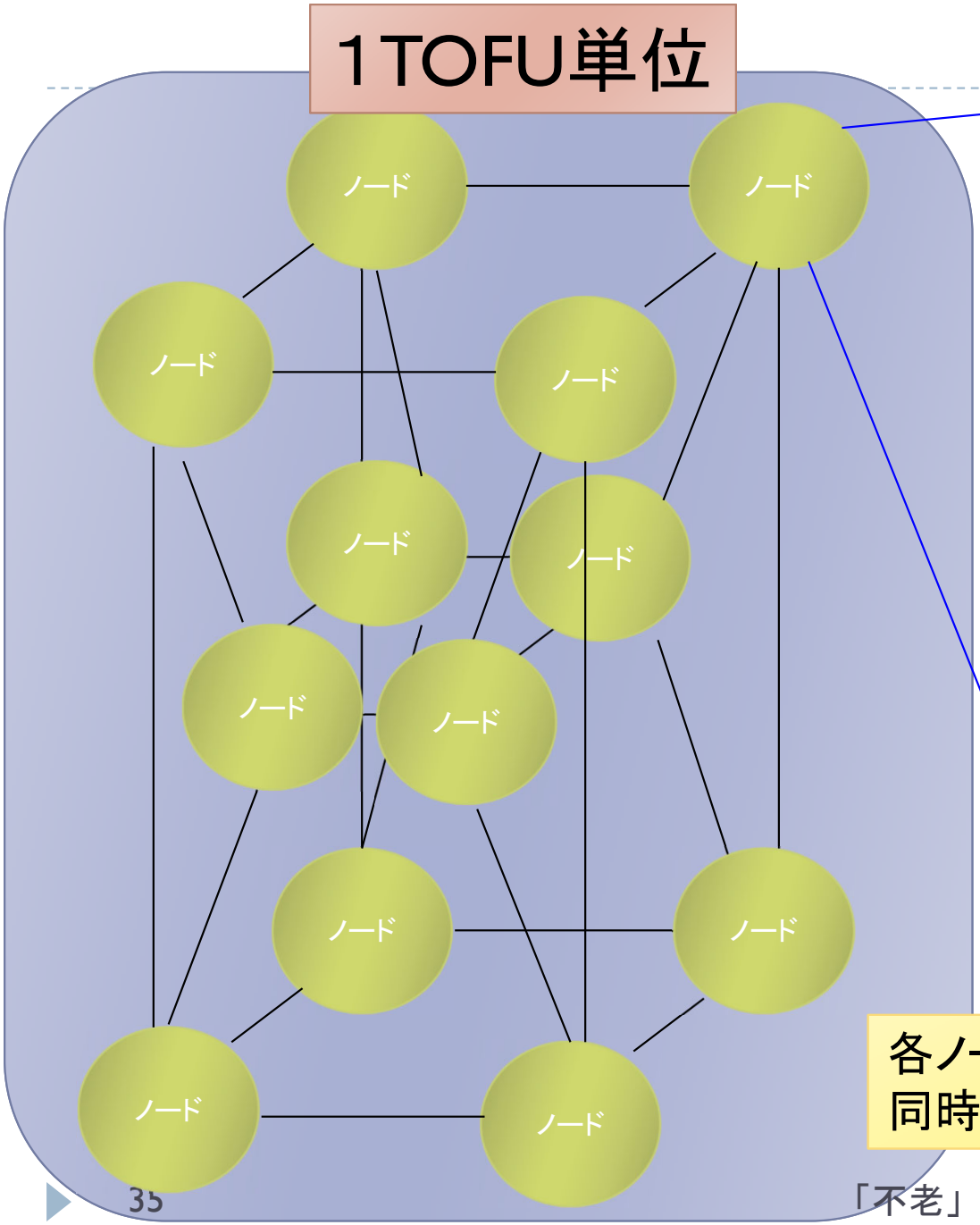
出典：https://www.ssken.gr.jp/MAINSITE/event/2015/20151028-sci/lecture-04/SSKEN_sci2015_miyoshi_presentation.pdf
https://monoist.atmarkit.co.jp/mn/articles/1905/07/news013.html

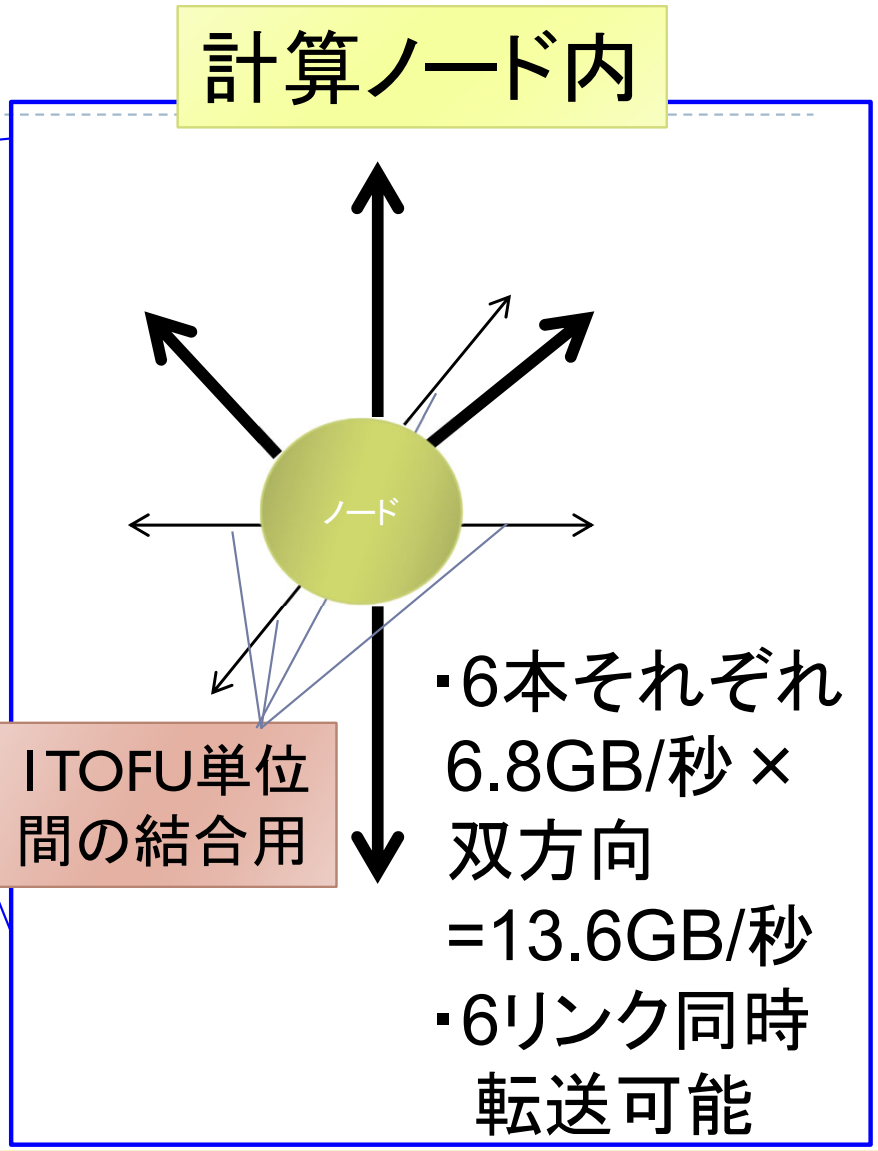
FX1000計算ノードの構成



読込み:512GB/秒
書込み:512GB/秒=合計:1024GB/秒

FX1000通信網：TofuインターコネクトD

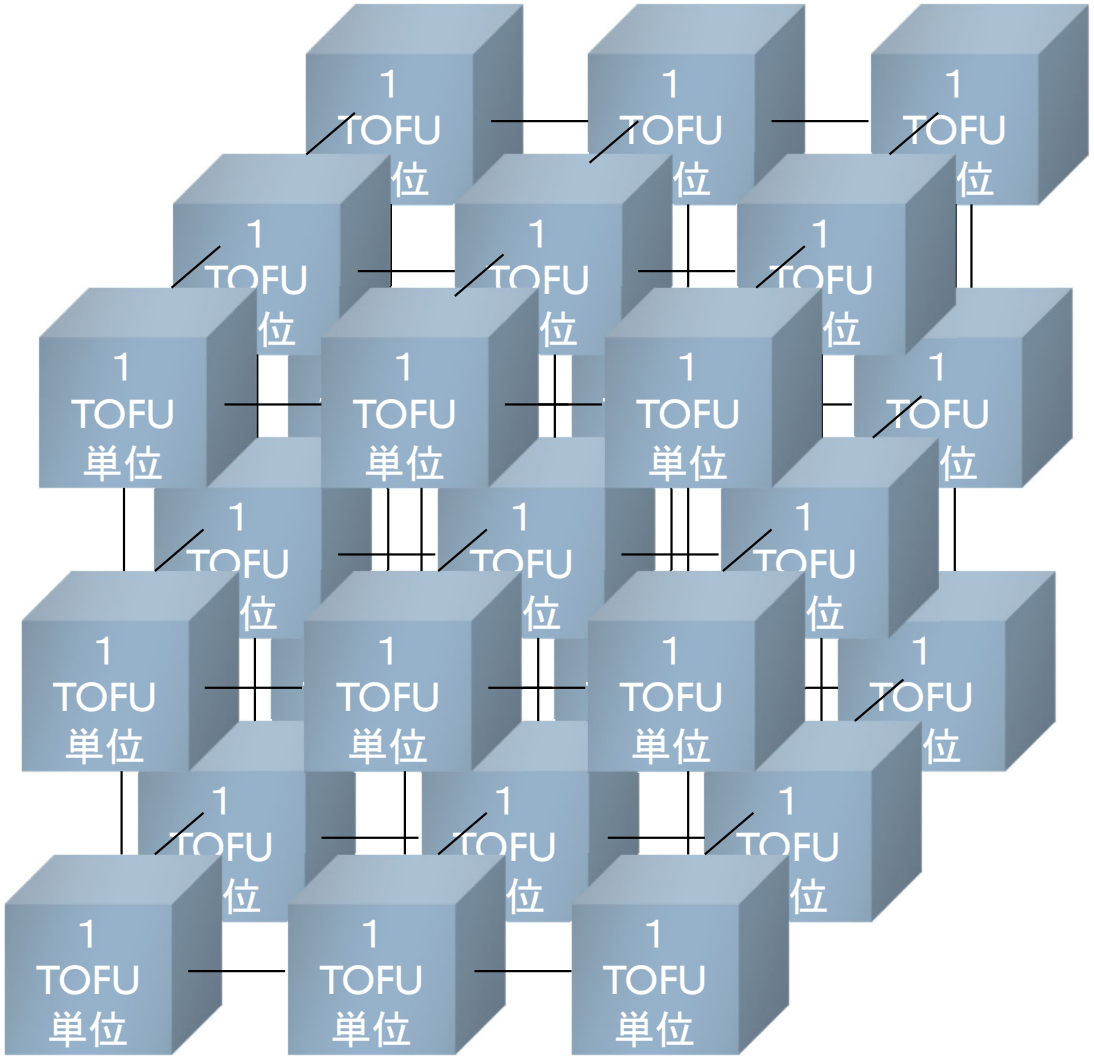




各ノードは周囲の隣接ノードへ
同時に合計 40.8 GB/s×双方向 で通信が可能

FX1000通信網：TofuインターコネクトD

3次元接続



- ユーザから見ると、
  X軸、Y軸、Z軸について、
  奥の1TOFUと、手前の
  1TOFUは、繋がってみえます
（3次元トーラス接続）
- ただし物理結線では
  - X軸はトーラス
  - Y軸はメッシュ
  - Z軸はメッシュまたは、
    トーラス

になっています

課金体系

スーパーコンピュータ「不老」課金体系 (1/8)

- 前払い定額制(プリペイド形式)
  - 利用すべき資源の料金を前払いして利用
  - 利用ポイントに変換して利用
- 単年度会計(4月1日～翌年3月31日)
  - 年度途中で申込み可能だが、利用終了は年度末
  - 年度末に余った利用ポイントは没収
- 一度の申込みで全てのサブシステムと可視化システムを利用可能
  - Type I、Type II、Type III、クラウド　全て共通

スーパーコンピュータ「不老」課金体系 (2/8)

- アカデミックユーザ（大学、研究機関など所属）向けプラン
  - 基本負担金
    - 利用登録1名につき年額10,000円（登録料）
    - 10,000⇒**6,500**利用ポイントを付加
      （他ユーザへの譲渡不可）
  - 追加負担金
    - 1,000円単位で追加が可能
    - 50万円未満：　1円あたり1⇒**0.65** ポイント付加
    - 50万円以上：　1円あたり1.25⇒**0.8125**ポイント付加

スーパーコンピュータ「不老」課金体系 (3/8)

- TypeIサブシステム(「富岳」型ノード)消費ポイント
  - 計算課金:利用ノード数×経過時間[s] × 0.0056
    - 基本負担金1万円＝0.65万ポイント付加で利用可能な目安
      - 1ノードを　約21日　⇒　約13．7日
      - 4ノードを　約　5日　⇒　約　3．2日
    - 10万円(基本利用料金1万円、追加料金9万円)＝6.5万ポイント付加で利用可能な目安
      - 1ノードを　約207日　⇒　約135日
      - 4ノードを　約　52日　⇒　約　34日
      - 8ノードを　約　26日　⇒　約　17日
    - 1ノードの年間利用額:約16万9000円⇒約25万8500円
      - 保守日等を考慮し年間350日利用できると仮定、以下同様

スーパーコンピュータ「不老」課金体系 (4/8)

- TypeIIサブシステム（GPUノード）消費ポイント
  - 計算課金：利用GPU数×経過時間[s] × 0.007
    - 基本負担金1万円
      ＝0．65万ポイント付加で利用可能な目安
      - 1ノード（1GPU）を　約17日　⇒　約11．1日
      - 1ノード（4GPU）を　約　4日　⇒　約　2．6日
    - 10万円（基本利用料金1万円、追加料金9万円）
      ＝6．5万ポイント付加で利用可能な目安
      - 1ノード（1GPU）を　約165日　⇒　約108日
      - 1ノード（4GPU）を　約　41日　⇒　約　27日
      - 4ノード（16GPU）を　約　10日　⇒　約　7日
    - 1ノード（4GPU）の年間利用額：約84万7000円
      ⇒約128万5000円

スーパーコンピュータ「不老」課金体系 (5/8)

- Type III：1ソケット当たり28コア
- クラウド：1ソケット当たり20コア

- TypeIIIサブシステム（大規模共有メモリノード）、およびクラウドシステムの消費ポイント
  - 計算課金：利用ソケット数×経過時間[s] × 0.002
    - 基本負担金1万円
      ＝0．65万ポイント付加で利用可能な目安
      - 1ソケットを　約58日　⇒　約37．9日
      - 4ソケットを　約14日　⇒　約　9．1日
    - 10万円（基本利用料金1万円、追加料金9万円）
      ＝6．5万ポイント付加で利用可能な目安
      - 　4ソケットを　約145日　⇒　約95日
      - 32ソケットを　約　18日　⇒　約12日
    - 2ソケットの年間利用額：約12万1000円
      ⇒約18万5000円

スーパーコンピュータ「不老」課金体系 (6/8)

- 会話型利用：
  ログインノード上での処理の消費ポイント
  - 課金：無料
    - 主にインストール作業での利用想定です。
      計算利用はご遠慮ください。

  - Type IIIサブシステム（会話型処理）の課金はありますのでご注意ください。
  - 各計算ノードの備えるinteractiveジョブクラスはバッチジョブ扱いの課金です
    （fx-interactive, cx-interactive, cl-interactive）

スーパーコンピュータ「不老」課金体系 (7/8)

- ホットストレージ
  - ファイル課金
    - 1TB 以下の場合(Home + Large) : 徴収しない
    - ファイルの使用容量が1TB を超えた場合 :
      超えた容量について、1GB につき 1日当たり 0.01 ポイント
    - 例) 2TB(2000GB)利用 : 1000GBが課金対象
      ⇒ 10ポイント/日　⇒300円/30日、3，500円/350日
      （保守などで停止する日については徴収しない）

※128TBを超える場合は、全体容量を考慮して、削除依頼をさせていただくことがあります【予定】。
※128TBを超える容量が必要な場合は、事前に相談ください。

スーパーコンピュータ「不老」
課金体系 (8/8)

- コールドストレージ
  - ファイル課金
    - **1口：50TB**
      - 1回だけ書き込める（追記可能）の光ディスク×10枚（1枚約5TB）
    - ファイル負担経費（初回利用時のみ必要）：
      1口　190,000円
    - ファイル管理経費（毎年必要、基本負担金とは別）：
      1口　10,000円

●事前納入の光ディスク６PB売り切り後は、ユーザによる持ち込みのみなります（上限１０PBまで）。
●管理料１万円／年と格安です！
すでに、３PB程度が売れています！
●ご購入はお早めに！

※ユーザの利用終了時、もしくは、「不老」運用終了時に、光ディスクを持ち帰りいただけます。

コールドストレージ　サービス紹介

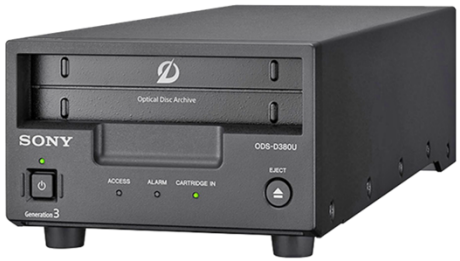
2021 年 2 月より光ディスクを使ったコールドストレージサービスを開始していますが、**2021 年 5 月に光ディスクカートリッジが利用できる単体ドライブの貸し出しサービスも開始しました。**

コールドストレージ内部

名古屋大学
NAGOYA UNIVERSITY

Windows/Mac OSを搭載したPCに、USB3.2又はUSB3.0で単体ドライブを接続して、専用のFilerソフトを使って利用できます。

光ディスクカートリッジ



単体ドライブ

輸送には専用のジェラルミンケースをご用意しています。

輸送用ジェラルミンケース

コールドストレージシステム概要：

詳細は、以下をご覧ください：
https://icts.nagoya-u.ac.jp/ja/sc/pdf/pamphlet-cold-202012.pdf

スーパーコンピュータ「不老」新サービス：ノード準占有利用

- ノード準占有利用
  - 1時間以内のジョブ実行開始を保証
  - バッチ利用のみ
- 1ノード、1ヶ月間の利用負担金
  - Type IIサブシステム：
    210,000円⇒**330,000円**（通常価格の約2.8倍）
  - クラウドシステム：
    62,000円⇒**95,000円**（通常価格の約2.8倍）

提供資源量が無くなり次第、受付終了
→お早めに

スーパーコンピュータ「不老」新サービス：クラウドノード予約利用

- クラウドノード予約利用
  - 専用の予約システム「UNCAI」（Webブラウザで操作）でノードを予約して利用する
- 利用料金
  - 計算課金：利用コア数×経過時間[s] × 0.0001
    （ソケット当たりコア数を考えればバッチ実行と同等）
    - 1ソケット当たり20コア、10コア（0.5ソケット）から利用可能
    - 利用可能なメモリ容量もコア数に比例
    - 基本負担金1万円でXeon Gold 20コア1ソケットを約58日⇒**約38日間使用可能**

スーパーコンピュータ「不老」新サービス：グループ利用

売れています！

- グループ利用
  - 1口10人まで、10万円で100,000 ⇒ **65,000** ポイント付与
  - 登録料なし
  - 個人利用（個別に1万円×10人が個別に基本負担金を払う）との違い
    - 65,000ポイントを10人で共有利用できます
    - 個人利用では、購入した6,500ポイントを他者と共有できません

お試し利用、リテラシー利用

- トライアルユース
  - ソフトウェアの動作確認などを、無料で行える制度です。
  - お1人様1回限りで申請できます。
  - 企業においては、同一の課で1回のみです。
  - アカデミックユーザ（無審査）、企業ユーザ（書類審査）
  - 10,000ポイント付加 （値上前と同じ）
  - 有効期限1ヶ月
- リテラシー利用（アカデミックユーザのみ）
  - 名大学内外の学部・大学院等の講義や演習で利用いただける制度です。
  - 利用登録25 件につき10,000 円、50,000ポイント付与（値上前と同じ）
  - 有効期限：上限6ヶ月（講義・演習実施期間に依存）

スーパーコンピュータ「不老」民間利用制度（産業利用）

- 書類での審査があります。追加負担金も同額です。
- 公開型
  - 10アカウントまで20万円
  - アカデミック利用の料金の2倍（20万円当たり100,000⇒65,000ポイント）
  - 企業名、課題名、報告書をWebで公開（延期制度あり）
- 非公開型
  - 10アカウントまで40万円
  - アカデミック利用の料金の4倍（40万円当たり100,000 ⇒65,000ポイント）
  - 外部に情報は非公開（ただし内部会議では情報が出ます）

申込み金額に応じたポイント優遇はございません。
詳しくは、産業利用のパンフレットをご参照ください。

優先ジョブクラス（アカデミック・民間）

- TypeI、TypeII、クラウドの各サブシステム
  - ポイントを通常の2倍消費することで利用可能なジョブクラス
  - 専用のキューにジョブを投げることで利用
  - 通常のジョブクラスが混んでいるときでも早く実行したい、
    というユーザの利用を想定

  - （優先ジョブクラスも混んでしまったらすいません）

コンサルティング

- 並列化、利用高度化、ISVアプリの利用方法などに関するコンサルティングを行っています。
- 本センター教職員や学内外の専門家で構成される専門分野相談員によるコンサルティング（面談）ができます。
  - Web受付 Q&A SYSTEM
    - 各種ご質問、ご相談等は下記Webサイトからお問合せください。
    - https://qa.icts.nagoya-u.ac.jp/
  - 面談相談（※ZOOMによる遠隔面談も可）
    - 実際に画面を見ながらなど、電話やメールでは伝えにくいご質問やご相談は面談でも受け付けています。
    - 事前にお約束の上、本センター3階図書室内のIT相談コーナーにお越しください。または、相談員が訪問させていただくことも可能です。
      - 連絡先：052-789-4366（IT相談コーナー直通）、または上記のQ&A SYSTEMから

可視化設備

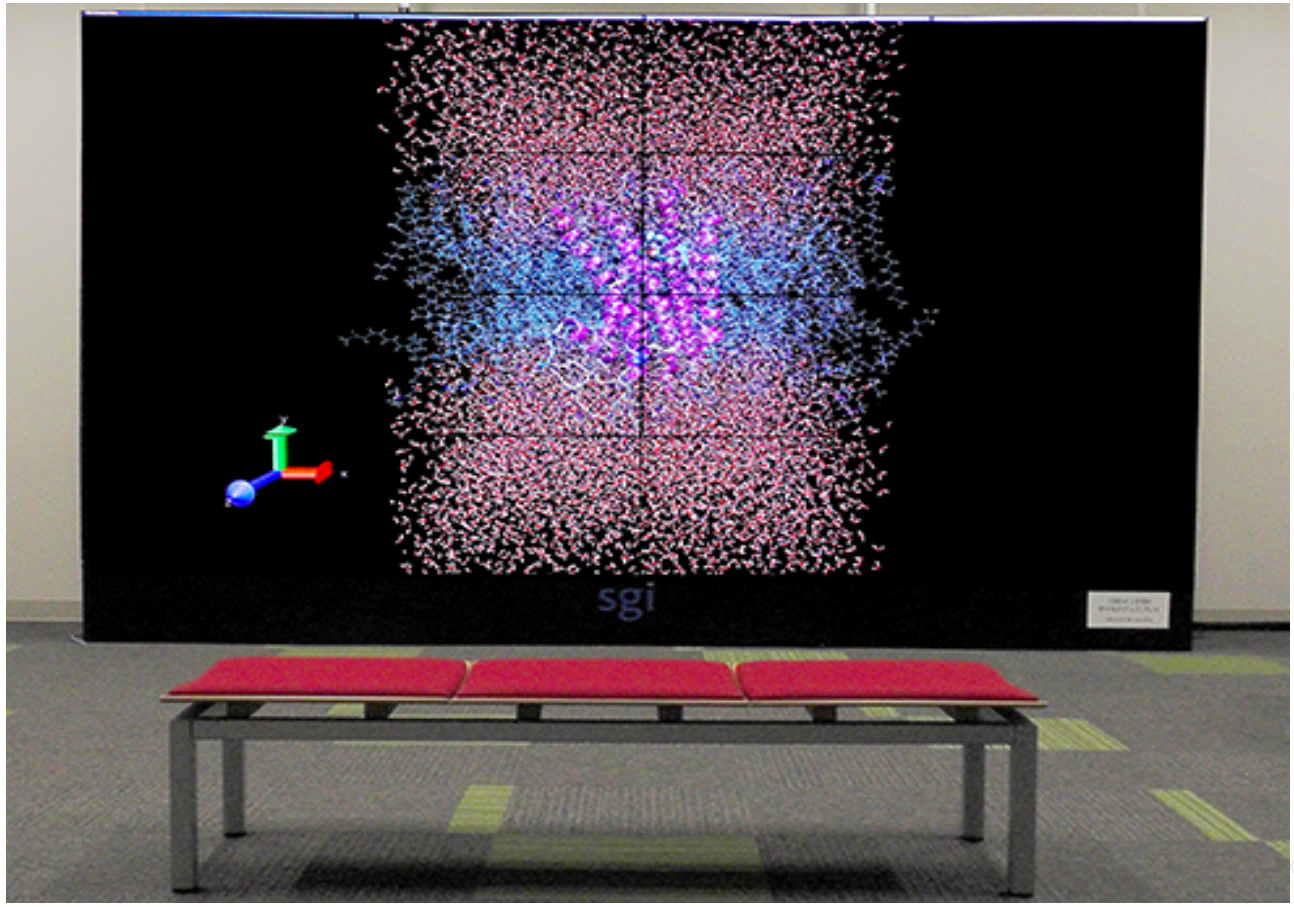
- 情報基盤センター可視化室（本館1階）
  - 可視化室の利用は予約制となります（無料）

- 8K 185型タイルドディスプレイ
- 全天周映像視聴システム
- 円偏光立体視システム　など

可視化設備

- 情報基盤センター可視化室（本館1階）
  - 可視化室の利用は予約制となります（無料）

8K 185型
タイルド
ディスプレイ



利用者支援室（２部屋）

- 情報基盤センター利用者支援室（本館3階）
  - 利用者支援室の利用は予約制となります（無料）

訪問者が現地で利用可能な機材

画像処理装置（1F可視化室）

- Windows10が動作するデスクトップパソコン（1台）
- 対話型のプリ・ポスト処理や共有ファイルストレージの大容量ファイル取り扱い用
- 10Gbps SINETによる高速インターネットが利用可能
- USB外付けHDDやディスクメディア（Blu-rayディスク）を持ち込んでホットストレージに対するデータの読み書きが可能
- 可視化室に設置された185インチ8K高精細ディスプレイに接続、大規模データの高品位なプリポスト処理やコンテンツ生成に活用可能

オンサイト利用装置（3F利用者支援室）

- Linuxが動作する計算サーバ（2台）
- 対話型のプリ・ポスト処理、スーパーコンピュータシステムとの大容量ファイル取り扱い用
- USB外付けHDDや光ディスクメディア（Blu-rayディスク）を持ち込んでホットストレージに対するデータの読み書きが可能
- 予約制の部屋貸し切りで、ハードディスク持ち込みで大規模データの入力、回収が可能

画像処理装置

ODA単体ドライブ

- コールドストレージの光ディスクアーカイブと互換性のあるUSB接続ドライブ
- 画像処理装置・オンサイト利用装置に接続して利用可能

オンサイト利用装置

- ODA単体ドライブの貸し出しできます（宅配便で郵送できる専用ケースも有）
- ご相談ください

スーパーコンピュータ「不老」講習会等 開催情報（2024年2月～）

2023年度「不老」利用型講習会 開催予定（オンライン開催）

2024年

- 3月11日：OpenMP（初級）
- 3月12日：「不老」およびmdxオンライン利用説明会

- 2024年4月以降も、多数開催予定
- 最新状況は以下のHPをご覧ください
  https://www2.itc.nagoya-u.ac.jp/cgi-bin/kousyu/csview2.cgi

スパコン利用の方法

マニュアルなどの入手方法

- スパコンへのログインなど初歩的なことや最新のTips（便利な使い方）、最適化のためのお役立ち情報等
  - 情報基盤センターのWebページで誰でも入手可能
  - 「スーパーコンピュータ「不老」基本マニュアルおよび関連資料」
  - http://www.icts.nagoya-u.ac.jp/ja/sc/usage.html

- 各サブシステムのマニュアルなど
  - HPC Portalからダウンロード
  - 「不老」のアカウントを持つ利用者のみがアクセス可能
  - https://portal.cc.nagoya-u.ac.jp/

基本的な使い方

- SSH公開鍵の設定、マニュアルの閲覧：
  HPC PortalにWebアクセス
- バッチジョブ：SSHで接続し、バッチジョブシステムにジョブを登録
- クラウドシステムの時間指定ジョブ実行：Webシステム（UNCAI）から予約、時間になったらSSHで接続して利用
- 多くのスパコンと同様、「不老」もバッチジョブシステムでプログラムの実行を管理
  - 多数のユーザの多数のジョブを効率よく処理するため
- 利用者はSSH接続環境を整備しジョブスクリプトの書き方を習得する必要がある

接続先（ログインノードとHPC PortalとUNCAI）

- SSH接続先

対象計算サブシステム	SSH接続先	備考
Type Iサブシステム	flow-fx.cc.nagoya-u.ac.jp	
Type IIサブシステム	flow-cx.cc.nagoya-u.ac.jp	GPU搭載
Type IIIサブシステム	flow-lm.cc.nagoya-u.ac.jp	GPU搭載
Type IIIサブシステム（可視化用）	post.cc.nagoya-u.ac.jp	リモート可視化用
クラウドシステム	flow-cloud.cc.nagoya-u.ac.jp	

- HPC Portal : https://portal.cc.nagoya-u.ac.jp/
- UNCAI : https://portal.cc.nagoya-u.ac.jp/reserve/

共有ファイルシステム上の大規模ファイルの収納について

- /home領域は容量制限があります。性能的にも良くありません。
  - 1TB以上のデータは容量制限があり、デフォルト設定では置くことができません。
- 大規模ファイル、もしくは、高性能が必要なファイルは、以下に収納してください
  - /data/group1/
    の自分のID名があるフォルダの下
- Type Ⅱ サブシステム利用の場合は、ノードローカルのメモリ(SSD)を一時的に使うことで高速化できる可能性があります。

バッチジョブシステムの操作

- （スパコンでは一般的ですが）計算ノードにジョブを実行させたり情報を取得したりするには、内容と実行方法を記述したジョブスクリプトと、ジョブ制御用のコマンドを使います
- Type I, II, III, クラウドの各計算サブシステムで共通のジョブ制御用コマンドが使えます
- 主なジョブ制御用コマンド
  - pjsub：ジョブを投入する（プログラムの実行を指示する）
  - pjstat, pjstat2（推薦）：ジョブの投入状況を確認する
  - pjdel：投入したジョブを削除する
  - 各コマンドの詳細や指定できる引数については manコマンドや--helpオプションで確認してください

バッチジョブスクリプトの基本的な書き方

- シェルスクリプトで記述する

```
#!/bin/bash -x
#PJM -L rscunit=fx
#PJM -L rscgrp=fx-small
#PJM -L node=2
#PJM --mpi proc=4
#PJM -L elapse=1:00:00
#PJM -j
#PJM -S

module avail
module list

export
OMP_NUM_THREADS=24
mpiexec ./a.out
```

(-xを付けておくと出力ファイルにコマンド自体も出力される)
リソースユニット(利用するサブシステム)名をfx=Type IIに指定
リソースグループ(ジョブキュー)名をfx-smallに指定
2ノード実行
合計4プロセス実行
最大実行時間を1時間に指定
標準エラー出力を標準出力に統合
実行時の統計情報を出力する

利用できるmodulefilesを確認
ロードしてあるmodulefilesを確認

プロセス当たりのスレッド数を24に指定
MPIを用いてプログラムを実行

リソースグループの選択

- 実行したいジョブの設定に合わせて
  リソースグループを選択する必要がある
  - ユーザから見れば何も気にせずに実行できるのが楽だが、
    大量のジョブをスムーズに実行するにはユーザの協力が
    不可欠
  - 基本的には利用するノード数や実行したい時間が適切なもの
    を選べば良い
    - 指定した利用時間を過ぎると問答無用で強制終了されます
  - ちょうど良いリソースグループ・空いているリソース
    グループを適切に選んで効率よく利用してください

Type Iサブシステムのリソースグループ一覧

デフォルトの割当方法を「離散」に変更（2021年6月）

※現在の状況はHPをご覧ください。

リソースグループ名	最小ノード数	最大ノード数	最大CPUコア数	最長実行時間(デフォルト値)	最長実行時間(最大値)	最大メモリ容量 (*)	割当方法(トーラス)	割当方法(離散)	備考
fx-interactive	1	4	192	1時間	24時間	28 GiB x 4	不可	可	会話型バッチ
fx-debug	1	36	1,728	1時間	1時間	28 GiB x 36	不可	可	短時間デバッグ用
fx-small	1	24	1,152	12時間	168時間	28 GiB x 24	不可	可	1 BoB単位
fx-middle	12	96	4,608	12時間	72時間	28 GiB x 96	可	可	2 シェルフ単位
fx-large	96	192	9,216	12時間	72時間	28 GiB x 192	可	可	1/2 ラック単位
fx-xlarge	96	768	36,864	12時間	24時間	28 GiB x 768	可	可	2 ラック単位
fx-special	1	2,304	110,592	unlimited	unlimited	28 GiB x 2,304	可	可	事前予約制 (†)
fx-middle2	1	96	4,608	12時間	72時間	28 GiB x 96	可	可	実行優先度強化型 (‡)

Type IIサブシステムのリソースグループ一覧

※現在の状況はHPをご覧ください。

リソース グループ名	最大 ノード数	最大 CPUコア数	最長実行時間 （デフォルト値）	最長実行時間 （最大値）	最大 メモリ容量(*)	ローカルストレージ			備考
						NVMeSSD 6.4TB 利用可能	BeeOND 利用可能	BeeGFS (NVMesh) 利用可能(申請制)	
cx-interactive	1	1	1時間	24時間	338 GiB	○			会話型バッチ
cx-debug	4	160	1時間	1時間	338 GiB x 4	○			短時間デバッグ用、準占有利用ノードと共有
cx-share	1/4(共有)	10	1時間	168時間	84 GiB	○(共有)			ノード共有(**)、資源を1/4に分割。ジョブ実行およびインタラクティブジョブ実行が可能
cx-single	1	40	1時間	336時間	338 GiB x 1	○			
cx-small	8	320	1時間	168時間	338 GiB x 8	○	○		
cx-middle	16	640	1時間	72時間	338 GiB x 16	○	○		
cx-large	64	2,560	1時間	72時間	338 GiB x 64	○	○		
cx-special	221	8,840	unlimited	unlimited	338 GiB x 221	○			事前予約制 (†)
cx-middle2	16	640	1時間	72時間	338 GiB x 16	○	○		実行優先度強化型 (‡)
cxgfs-interactive	1	40	1時間	168時間	338 GiB x 1			○	会話型バッチ
cxgfs-share	1/4(共有)	10	1時間	168時間	84 GiB x 1			○(共有)	ノード共有(**)、資源を1/4に分割。ジョブ実行およびインタラクティブジョブ実行が可能
cxgfs-single	1	40	1時間	336時間	338 GiB x 1			○	
cxgfs-small	8	320	1時間	168時間	338 GiB x 8			○	
cxgfs-middle	16	640	1時間	72時間	338 GiB x 16			○	
cxgfs-special	50	2,000	1時間	72時間	338 GiB x 50			○	事前予約制 (†)
準占有利用	契約次第	契約次第	unlimited	unlimited	338 GiB x 実行ノード数	○	要相談	要相談	

Type IIIサブシステムのリソースグループ一覧

※現在の状況はHPをご覧ください。

リソースグループ名	最大ノード数	最大CPUソケット数(CPUコア数)	最長実行時間(デフォルト値)	最長実行時間(最大値)	備考
lm-middle	1	6(168)	24時間	72時間	8,034 GiB
lm-large	1	16(448)	24時間	24時間	21,424 GiB

クラウドシステムのリソースグループ一覧

※現在の状況はHPをご覧ください。

リソースグループ名	最小ノード数	最大ノード数	最大CPUコア数	最長実行時間（デフォルト値）	最長実行時間（最大値）	最大メモリ容量	備考
cl-interactive	1	1	80	1時間	168時間	338 GiB x 1	インタラクティブ実行用
cl-debug	1	4	320	1時間	1時間	338 GiB x 4	短時間デバッグ用、準占有利用ノードと共有
cl-share	1	1	20	1時間	168時間	84 GiB	ノード共有
cl-single	1	1	80	1時間	168時間	338 GiB x 1	
cl-small	2	8	640	1時間	168時間	338 GiB x 8	
cl-middle	8	16	1,280	1時間	72時間	338 GiB x 16	
cl-special		50	4,000	unlimited	unlimited	338 GiB x 50	全ノード、事前予約制

計算サブシステムの使い分け

どのサブシステムを使えば良いのか

各サブシステムの特徴から選ぶ

- Type Iサブシステム
  - 特徴：FX100の後継機、「富岳」と同じアーキテクチャ、ノード数が多い
  - 主な対象利用者：FX100や「富岳」で実績のあるプログラムを動かしたい、富士通コンパイラを使いたい、大規模分散(MPI)並列実行したい
- Type IIサブシステム
  - 特徴：Intel CPU + NVIDIA GPU 、ローカルSSD搭載
  - 主な対象利用者：GPUを使いたい、高いI/O性能が欲しい
- Type IIIサブシステム
  - 特徴：大容量メモリ環境、可視化システムと接続
  - 想定される利用者：大容量メモリを使いたい、可視化システムを使いたい
- クラウドシステム
  - 特徴：インタラクティブ実行、Intel CPU×4ソケット搭載
  - 想定される利用者：インタラクティブ処理がしたい、高いノード内CPU並列演算性能が欲しい、研究室のワークステーション・小規模クラスタの代わりに使いたい

用途から選ぶ（1／3）

- 旧システム利用者はどのシステムを使うのがオススメ？
  - 旧FXシステム　→　Type Iサブシステム
  - 旧CXシステム　→　Type IIサブシステム　またはクラウドシステム
  - 旧UVシステム　→　Type IIIサブシステム

用途から選ぶ（2／3）

- 対象プログラムの属性とサブシステムの対応
  - 大規模分散(MPI)並列　→　Type Iサブシステム
  - GPU、高速I/O　→　Type IIサブシステム
  - 可視化、大容量メモリ　→　Type IIIサブシステム
  - ノード内CPU並列実行で高い計算性能　→　クラウドシステム
    - OpenMP並列化は行えているがMPI並列化は行えていない場合など
  - 機械学習　→　Type Iサブシステム　または　Type IIサブシステム
    - 対象プログラム（利用するフレームワークなど）がGPU向けに最適化されているならType II、「富岳」向けに最適化されているならType I
  - インタラクティブ実行　→　大容量メモリも必要ならType IIIサブシステム、それ以外はクラウドシステム
  - 研究室のワークステーションや小規模PCクラスタ（Core i, Xeonなどx86系のCPUを搭載）の代わりに使いたい、高速化・大規模化したい
    → GPUを使いたいならType IIサブシステム、それ以外はクラウドシステム

用途から選ぶ（3／3）

- 「○○というソフトウェアを使いたいのだが？」
  - 「ソフトウェア利用環境」の表に従う、以下のWebページでも公開中
    - http://www.icts.nagoya-u.ac.jp/ja/sc/overview.html#software
  - それ以外は、利用実績を元に考えると良い
    - 「富岳」での利用実績がある　→　Type Iサブシステム
    - GPUを用いた利用実績がある　→　Type IIサブシステム
    - それ以外　→　クラウドシステム、大容量メモリが必要であればType IIIサブシステム
  - 各サブシステムに適したコンパイラの違いも参考に
    - Type I：富士通、LLVM
    - Type II：Intel、LLVM、GNU、PGI
    - Type III：Intel、LLVM、GNU
    - クラウド：Intel、LLVM、GNU
    - ※OSSとして公開されているソフトウェア等を自分でコンパイルして利用する場合は、対応するコンパイラや関係するライブラリの情報も確認してサブシステム選択に役立ててください

以降、Cygwinなどのターミナルを利用する人に特化されています

MobaXtermなどを利用の方は、送付済みの別資料をご覧ください。

テストプログラム起動

ＵＮＩＸ備忘録

- emacsの起動： emacs <編集ファイル名>
  - ^x ^s （^はcontrol）：テキストの保存
  - ^x ^c ： 終了
    （ ^z で終了すると、スパコンの負荷が上がる。絶対にしないこと。）
  - ^g ： 訳がわからなくなったとき。
  - ^k ：カーソルより行末まで消す。
    消した行は、一時的に記憶される。
  - ^y ：^kで消した行を、現在のカーソルの場所にコピーする。
  - ^s 文字列： 文字列の箇所まで移動する。
  - ^M x goto-line： 指定した行まで移動する。
    (^MはESCキーを押す)

ＵＮＩＸ備忘録

- rm ファイル名： ファイル名のファイルを消す。
  - rm *~ : test.c~ などの、~がついたバックアップファイルを消す。
- ls : 現在いるフォルダの中身を見る。
- cd フォルダ名： フォルダに移動する。
  - cd .. :一つ上のフォルダに移動。
  - cd ~ :ホームディレクトリに行く。訳がわからなくなったとき。
- cat ファイル名： ファイル名の中身を見る
- make :実行ファイルを作る
  （Makefile があるところでしか実行できない）
  - make clean :実行ファイルを消す。
    （clean がMakefileで定義されていないと実行できない）

ノートパソコンの設定：鍵の生成、ログイン

ユーザ名の確認

- 本講習会でのユーザー名：

  利用者番号：

- 本講習会のキュー名（後ほど説明）：

  fx-workshop

無線ＬＡＮの設定

- 各自のパソコンにおいて、無線LANの設定をしてください
  - 詳細は会場にある無線LAN情報をご覧ください

鍵の作成

1. ターミナルを起動する
2. 以下を入力する

```
$ ssh-keygen  –t  rsa
```

3. 鍵の収納先を聞かれるので、リターンを押す
4. 鍵を使うためのパスワードを聞かれるので、センターのパスワードではない、自分の好きなパスワードを入れる（パスフレーズとよぶ）
5. もう一度、上記のパスフレーズを入れる
6. 鍵が生成される

鍵の利用（１/２）

1. 生成した鍵は、以下に入っている
   .ssh/
2. 以下を入力する
   $ cd .ssh/
3. 以下を入力すると、ファイルが見える
   $ ls
   id_rsa id_rsa.pub known_hosts

- ここで、以下のファイルを区別する
  id_rsa　　：秘密鍵
  id_rsa.pub：公開鍵

この公開鍵の収納ディレクトリを覚えておく（後で使います）

鍵の利用（２/２）

4. 以下を入力して、公開鍵を表示する

   $ cat id_rsa.pub

   ＜公開鍵が表示される＞

5. “ssh-rsa …”で始まる部分を、マウスでカットアンドペーストし、公開鍵の登録に使う 。

「不老」への公開鍵の登録

- 以下をアクセスする。

https://portal.cc.nagoya-u.ac.jp/cgi-bin/hpcportal.ja/index.cgi

- ユーザ名とパスワードを聞かれるので、
  センターから発行されたユーザ名とパスワードを入れる。

ポータル画面（ログイン前）

センターから配られた
利用者番号　と　パスワード
を入れる

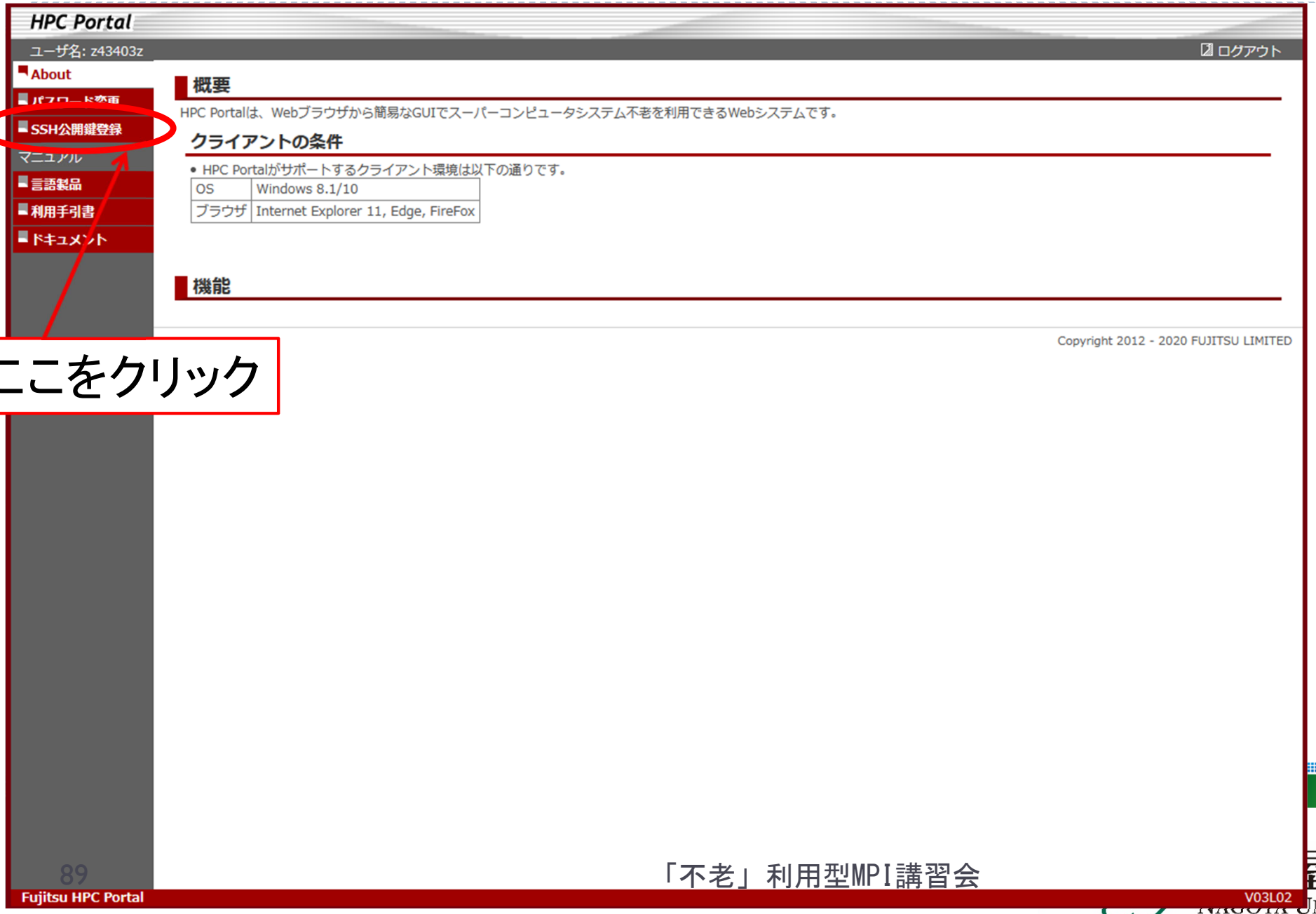
古屋大学
OYA UNIVERSITY

鍵の登録

1. 「ログイン」ボタンを押す
2. 成功すると、ログインメッセージがでる
3. 左側メニューの「SSH公開鍵登録」を
   クリックする
4. 「公開鍵」の画面に、公開鍵を
   カットアンドペーストする
5. 「新規登録」ボタンを押す

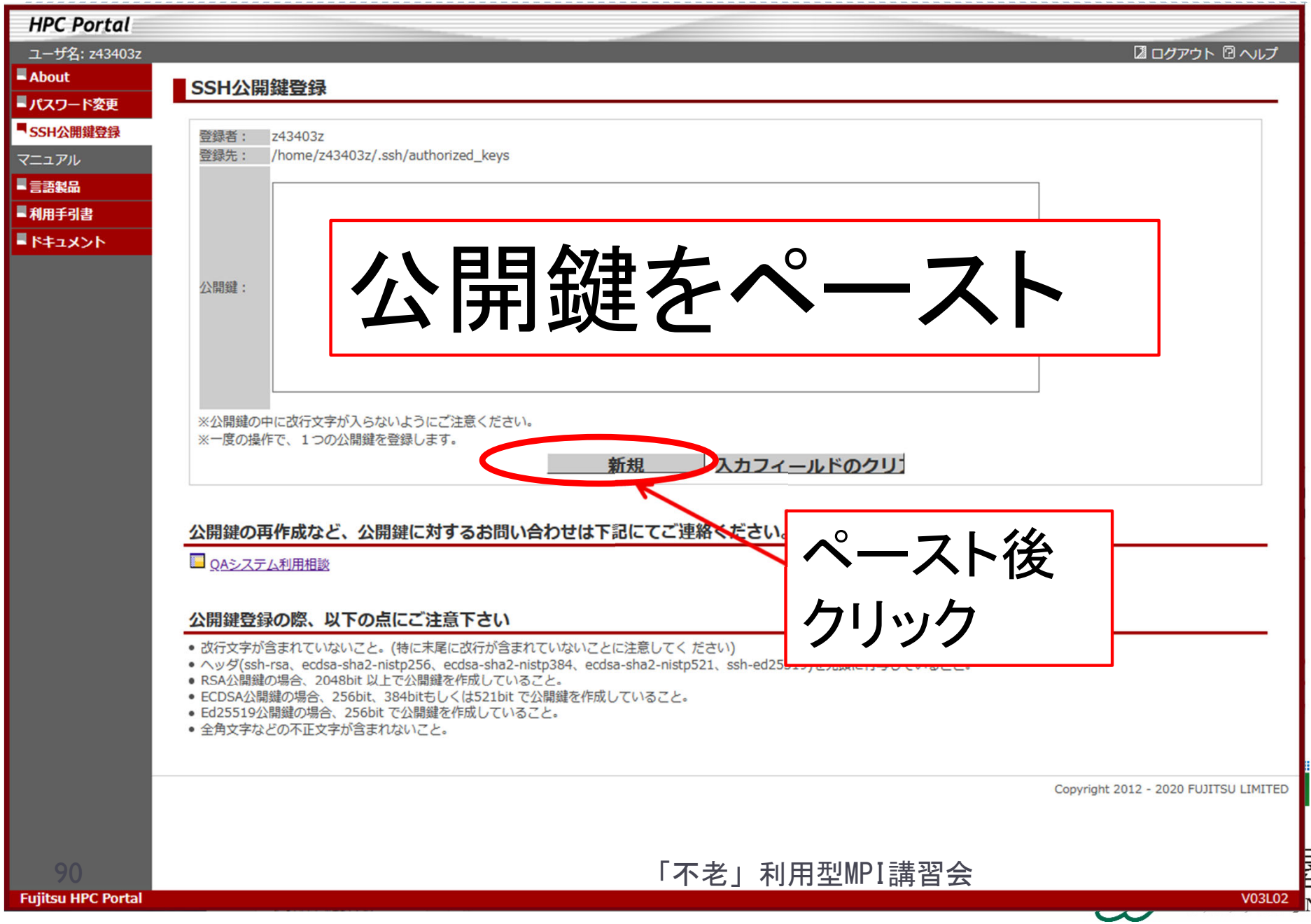
- ２度は鍵登録できません
- １度登録すれば「不老」全サブシステムで使えます

ポータル画面（ログイン後）



ここをクリック

ポータル画面（公開鍵登録）：



「不老」Type I サブシステムへログイン

- ターミナルから、以下を入力する

  $ ssh flow-fx.cc.nagoya-u.ac.jp -l w490xx

  「-l」はハイフンと小文字のL、

  「aYYxxx」は利用者番号（数字）

  “aYYxxx”は、利用者番号を入れる
- 接続するかと聞かれるので、 yes を入れる
- 鍵の設定時に入れた

  自分が決めたパスワード（パスフレーズ）

  を入れる
- 成功すると、ログインができる

「不老」TypeIサブシステム上のデータをPCに取り込む

- ターミナルから、以下を入力する

```
$ scp aYYxxx@flow-fx.cc.nagoya-u.ac.jp:~/a.f90   ./
```

「aYYxxx」は利用者番号（数字）
“aYYxxx”は、利用者番号を入れる

- 「不老」上に“a.f90”というファイルが無いとだめです。
- 「不老」上のホームディレクトリにある”a.f90”を、PCのカレントディレクトリに取ってくる
- ディレクトリごと取ってくるには、”-r” を指定

```
$ scp -r aYYxxx@flow-fx.cc.nagoya-u.ac.jp:~/SAMP   ./
```

- 「不老」上のホームディレクトリにあるSAMPフォルダを、その中身ごと、PCのカレントディレクトリに取ってくる

PCのファイルを「不老」TypeI サブシステム上に置く

- ターミナルから、以下を入力する

```
$ scp ./a.f90  aYYxxx@flow-fx.cc.nagoya-u.ac.jp:
```

「aYYxxx」は利用者番号（数字）
“aYYxxx”は、利用者番号を入れる

- PCのカレントディレクトリにある”a.f90”を、「不老」上のホームディレクトリに置く
- ディレクトリごと置くにはには、”-r” を指定

```
$ scp -r ./SAMP aYYxxx@flow-fx.cc.nagoya-u.ac.jp:
```

- PCのカレントディレクトリにあるSAMPフォルダを、その中身ごと、「不老」上のホームディレクトリに置く

参考：emacs の tramp機能（必要な人のみ）

- emacs が自分のパソコンに入っている人は、
  Tramp機能により、遠隔のファイルが操作できます
- 「不老」の秘密鍵を、SSHに登録します
- emacs を立ち上げます
- ファイル検索モードにします
  ^x ^f （^はcontrol）
- “Find file: ”の現在のパス名部分を消し、以下を
  入れます（ただし、t～は自分のログインIDにする）
  Find file:/ssh:aYYxxx@flow-fx.cc.nagoya-u.ac.jp:
- パスフレーズを入れると、ローカルファイルのようにF
  「不老」上のファイルが編集できます。

サンプルプログラムの実行

初めての並列プログラムの実行

サンプルプログラム名

- C言語版・Fortran90版共通ファイル：

  Samples-flow-fx.tar

- tarで展開後、C言語とFortran90言語のディレクトリが作られる
  - C/ ：C言語用
  - F/ ：Fortran90言語用
- 上記のファイルが置いてある場所

  /center/a49904a

並列版Helloプログラムをコンパイルしよう (1/2)

1. /center/a49904a にある Samples-flow-fx.tar を自分のディレクトリにコピーする
   $ cp /center/a49904a/Samples-flow-fx.tar ./
2. Samples-fx.tar を展開する
   $ tar xvf Samples-flow-fx.tar
3. Samples フォルダに入る
   $ cd Samples
4. C言語 : $ cd C
   Fortran90言語 : $ cd F
5. Hello フォルダに入る
   $ cd Hello

並列版Helloプログラムをコンパイルしよう (2/2)

6. ピュアMPI用のMakefileをコピーする
   $ cp Makefile_pure Makefile
7. make する
   $ make
8. 実行ファイル(hello)ができていることを確認する
   $ ls

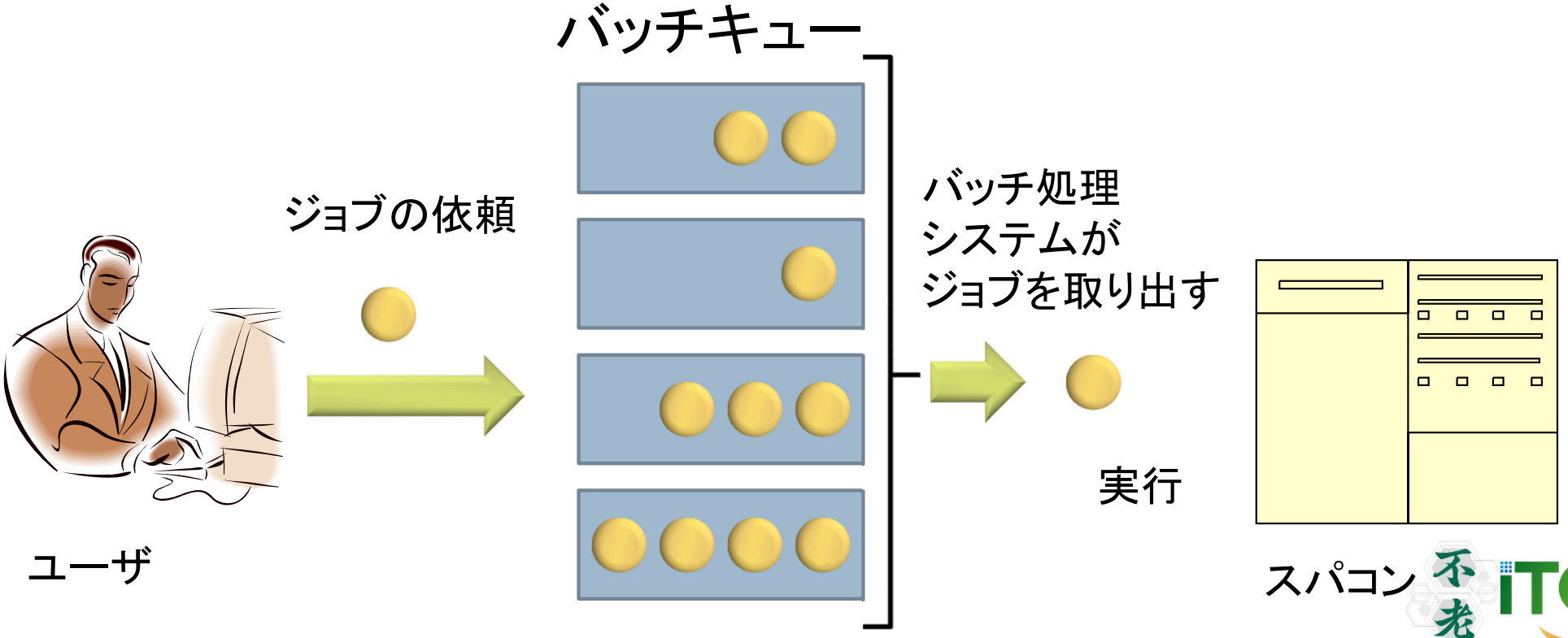
バッチ処理とジョブの投入

「不老」のジョブ実行形態の例

- 以下の2通りがあります
- インタラクティブジョブ実行
  - PCでの実行のように、コマンドを入力して実行する方法
  - スパコン環境では、あまり一般的でない
  - デバック用、大規模実行はできない
  - 「不老」Type Iサブシステムでは以下に限定
    - **最大4ノード（192コア）（標準1時間まで、最大で24時間）**
- バッチジョブ実行
  - バッチジョブシステムに処理を依頼して実行する方法
  - スパコン環境で一般的
  - 大規模実行用
  - 「不老」Type Iサブシステムでは：
    - **通常サービス： 最大768ノード（36,864コア）（24時間まで）**
    - **申込み制： 2304ノード（110,592コア）（実行時間制限無し）**

バッチ処理とは

- スパコン環境では、インタラクティブ実行（コマンドラインで実行すること）は提供されていないことがあります。特に、大規模並列実行ができないようになっています。
- ジョブはバッチ処理で実行します。



コンパイラの種類とインタラクティブ実行およびバッチ実行の例

- インタラクティブ実行、およびバッチ実行で、利用するコンパイラ（C言語、C＋＋言語、Fortran90言語）の種類が違います
- インタラクティブ実行では
  - オウンコンパイラ（そのノードで実行する実行ファイルを生成するコンパイラ）を使います
- バッチ実行では
  - クロスコンパイラ（そのノードでは実行できないが、バッチ実行する時のノードで実行できる実行ファイルを生成するコンパイラ）を使います
- それぞれの形式（富士通社の例）
  - オウンコンパイラ：＜コンパイラの種類名＞
  - クロスコンパイラ：＜コンパイラの種類名＞px
  - 例）富士通Fortran90コンパイラ
    - オウンコンパイラ：frt
    - クロスコンパイラ：frtpx

バッチキューの設定のしかた

- バッチ処理は、富士通社のバッチシステム(PJM)で管理されています。
- 以下、主要コマンドを説明します。
  - ジョブの投入：
    pjsub <ジョブスクリプトファイル名>
  - 自分が投入したジョブの状況確認：
    pjstat　もしくは　pjstat2
  - 投入ジョブの削除：　pjdel　<ジョブID>
  - バッチキューの状態を見る：　pjstat2 --rsc -b
    もしくは　pjstat2 --use
  - システム制限値を見る：pjstat2 --rsc –x

インタラクティブ実行のやり方の例

- コマンドラインで以下を入力
  - 1ノード実行用

```
$ pjsub -L "rscgrp=fx-interactive" --interact
```

  - 4ノード実行用

```
$ pjsub -L "rscgrp=fx-interactive,node=4"
--interact
```

pjstat2 --rsc の実行画面例

```
$ pjstat2 --rsc

RSCGRP          STATUS              NODE
fx-interactive  [ENABLE,START]      384: 4x6x16
fx-small        [ENABLE,START]      384: 4x6x16
fx-debug        [ENABLE,START]      384: 4x6x16
fx-extra        [ENABLE,START]      384: 4x6x16
fx-middle       [ENABLE,START]     1536: 8x12x16
fx-large        [ENABLE,START]     1536: 8x12x16
fx-xlarge       [ENABLE,START]     1536: 8x12x16
fx-special      [ENABLE,START]     2304: 12x12x16
fx-middle2      [ENABLE,START]     1536: 8x12x16
```

使える
キュー名
(リソース
グループ)

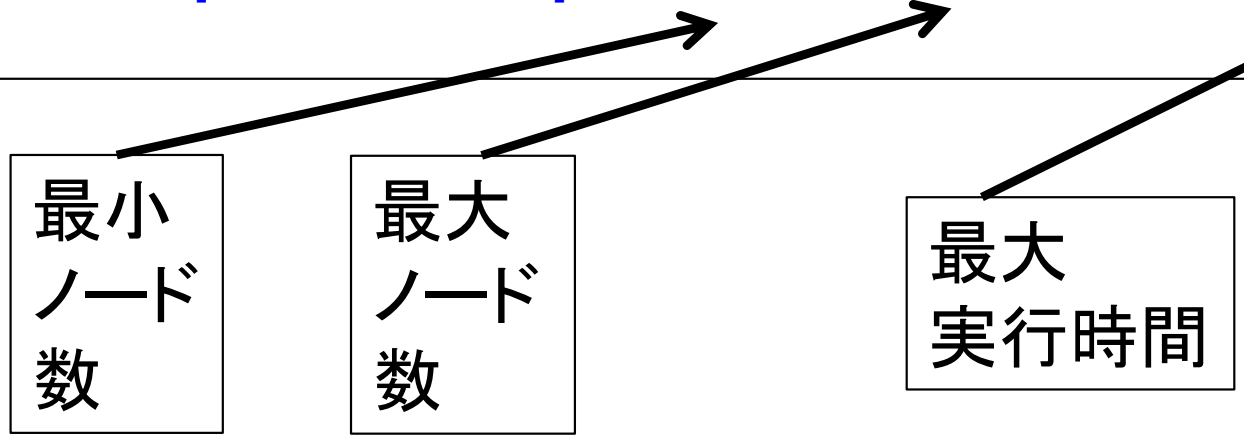
現在使えるか
ENABLE: キューに
ジョブ投入可能
START: ジョブが
流れている

ノードの
物理構成情報

pjstat2 --rsc -x の実行画面例

```
$ pjstat2 --rsc -x

RSCGRP          STATUS           MIN_NODE   MAX_NODE   MAX_ELAPSE MEM(GB)
fx-interactive  [ENABLE,START]      1          4        24:00:00     28
fx-small        [ENABLE,START]      1         24       168:00:00     28
fx-debug        [ENABLE,START]      1         36        01:00:00     28
fx-extra        [ENABLE,START]      1         36        12:00:00     28
fx-middle       [ENABLE,START]     12         96        72:00:00     28
fx-large        [ENABLE,START]     96        192        72:00:00     28
fx-xlarge       [ENABLE,START]     96        768        24:00:00     28
fx-special      [ENABLE,START]      1       2304         unlimited     28
fx-middle2      [ENABLE,START]      1         96        72:00:00     28
```



pjstat2 --rsc -b の実行画面例

```
$ pjstat2 --rsc -b

RSCGRP          STATUS           TOTAL  RUNNING  QUEUED  HOLD OTHER     NODE
fx-interactive  [ENABLE,START]     1        1        0       0     0   384:4x6x16
fx-small        [ENABLE,START]    83       33       50       0     2   384:4x6x16
fx-debug        [ENABLE,START]    40       22       18       0     0   384:4x6x16
fx-extra        [ENABLE,START]    20        8       12       0     2   384:4x6x16
fx-middle       [ENABLE,START]    33        7       26       0     0  1536:8x12x16
fx-large        [ENABLE,START]    10        3        7       0     0  1536:8x12x16
fx-xlarge       [ENABLE,START]     5        1        4       0     0  1536:8x12x16
fx-special      [ENABLE,START]     2        0        2       0     0  304:12x12x16
fx-middle2      [ENABLE,START]    10        2        8       0     0  1536:8x12x16
```

総合ジョブ数

実行中ジョブ数

待たされているジョブ数

pjstat2 --use の実行画面例

```
$ pjstat2 --use

RSCGRP                                  Used nodes/  Total nodes
fx-debug groups (total 768 nodes)                172/      768
   fx-interactive    ----------------------    0%          2
   fx-small          ******----------------   22%        170
   fx-debug          ----------------------    0%          0
fx groups (total 1536 nodes)                     751/     1536
   fx-middle          **--------------------    7%        111
   fx-middle2        ----------------------    0%          0
   fx-large          *******-----------------   25%        384
   fx-xlarge          *****-------------------   17%        256
fx-special           ----------------------    0%          0/     2304
```

当該キューに
割り当てられた
ノード数の何%が
使われているか

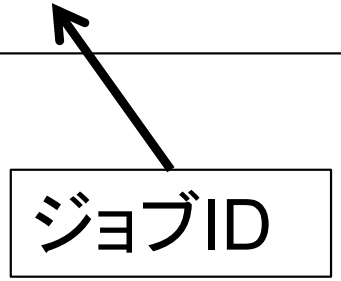
使われている
ノード数

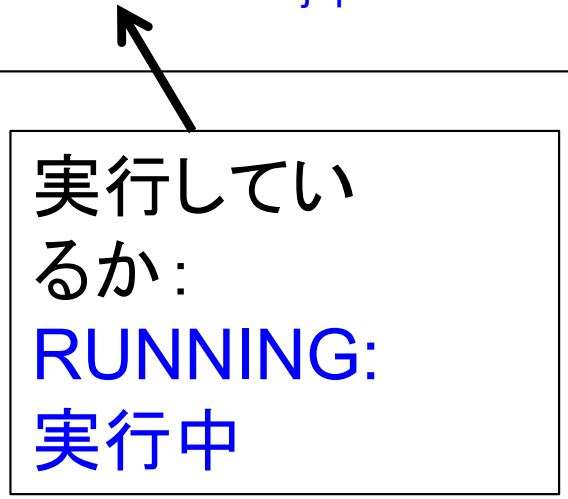
当該キューに割り当て
られているノード数
（キュー間で重複あり）

pjstat2 の実行画面例

```
$ pjstat2

JOB_ID      JOB_NAME   STATUS      GROUP      RSCGROUP      START_DATE       ELAPSE     NODE
95647       hello-pure       RUNNING     jhpcn2602  fx-debug          (09/15 16:47)<     00:00:00     12
```





JOBスクリプトサンプルの説明（ピュアMPI）
(hello-pure.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PJM -L "rscunit=fx"
#PJM -L "rscgrp=fx-workshop"
#PJM -L "node=12"
#PJM --mpi "proc=576"
#PJM -L "elapse=1:00"
mpirun ./hello
```

リソースグループ名
：fx-workshop

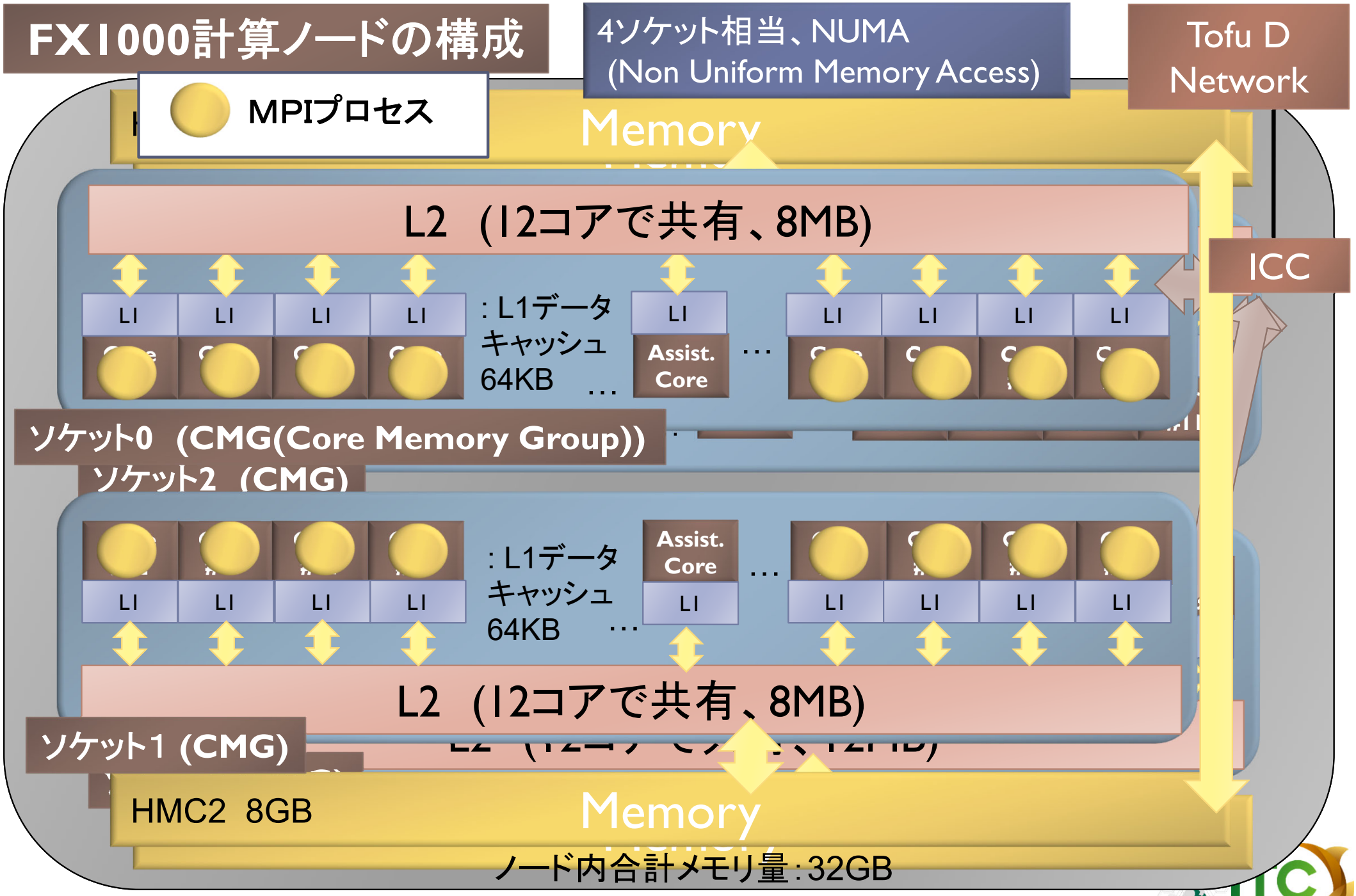
利用ノード数

利用コア数
（MPIプロセス数）

実行時間制限
：1分

MPIジョブを48＊12ノード
＝576プロセス で実行する

FX1000計算ノードの構成



読込み:240GB/秒
書込み:240GB/秒=合計:480GB/秒

並列版Helloプログラムを実行しよう（ピュアMPI）

1. Helloフォルダ中で以下を実行する
   $ pjsub hello-pure.bash
2. 自分の導入されたジョブを確認する
   $ pjstat
3. 実行が終了すると、以下のファイルが生成される
   hello-pure.bash.XXXXXX.err
   hello-pure.bash.XXXXXX.out （XXXXXXは数字）
4. 上記の標準出力ファイルの中身を見てみる
   $ cat hello-pure.bash.XXXXXX.out
5. “Hello parallel world!”が、
   48プロセス*12ノード=576個表示されていたら成功。

バッチジョブ実行による標準出力、標準エラー出力

- バッチジョブの実行が終了すると、標準出力ファイルと標準エラー出力ファイルが、ジョブ投入時のディレクトリに作成されます。
- 標準出力ファイルにはジョブ実行中の標準出力、標準エラー出力ファイルにはジョブ実行中のエラーメッセージが出力されます。

ジョブ名.XXXXX.out --- 標準出力ファイル
ジョブ名.XXXXX.err --- 標準エラー出力ファイル
(XXXXX はジョブ投入時に表示されるジョブのジョブID)

並列版Helloプログラムを実行しよう（ハイブリッドMPI）

- (準備)

  Helloフォルダ中で以下を実行する

```
$  make clean
$  cp Makefile_hy48  Makefile
$  make
```

並列版Helloプログラムを実行しよう（ハイブリッドMPI）

1. Helloフォルダ中で以下を実行する
   $ pjsub hello-hy48.bash
2. 自分の導入されたジョブを確認する
   $ pjstat
3. 実行が終了すると、以下のファイルが生成される
   hello-hy48.bash.XXXXXX.err
   hello-hy48.bash.XXXXXX.out （XXXXXXは数字）
4. 上記標準出力ファイルの中身を見てみる
   $ cat hello-hy48.bash.XXXXXX.out
5. “Hello parallel world!”が、
   1プロセス*12ノード=12 個表示されていたら成功。

JOBスクリプトサンプルの説明（ハイブリッドMPI）
(hello-hy48.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PJM -L "rscunit=fx"
#PJM -L "rscgrp=fx-workshop"
#PJM -L "node=12"
#PJM --mpi "proc=12"
#PJM -L "elapse=1:00"
export OMP_NUM_THREADS=48
mpirun ./hello
```

リソースグループ名
：fx-workshop

利用ノード数

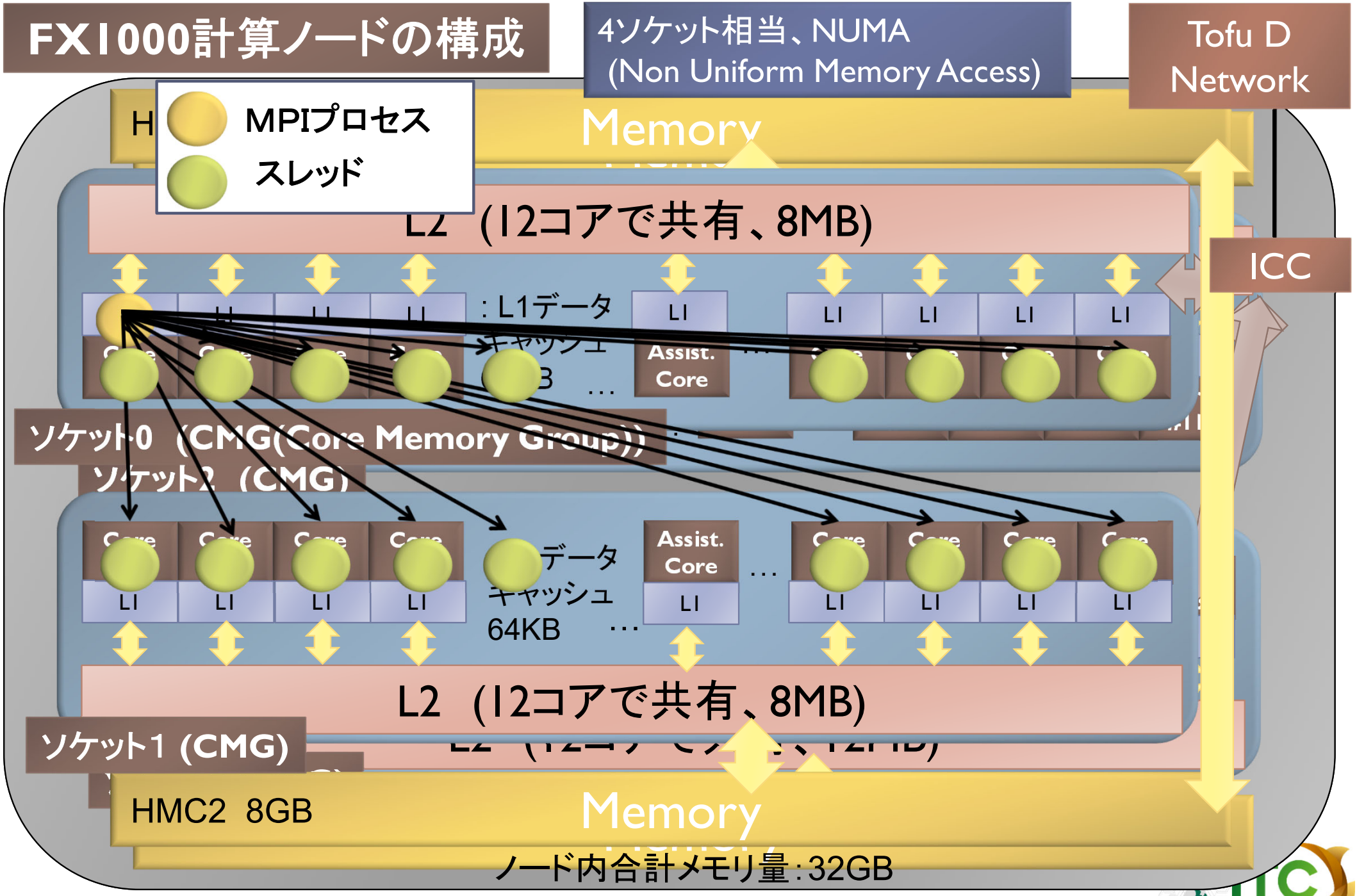
利用コア数
（MPIプロセス数）

実行時間制限：1分

1MPIプロセス当たり
48スレッド生成
※ただし効率的な
実行形式ではありません

MPIジョブを1＊12 = 12 プロセスで
実行する。

FX1000計算ノードの構成



読込み:512GB/秒
書込み:512GB/秒=合計:1024GB/秒

名古屋大学
NAGOYA UNIVERSITY

MPI実行時のリダイレクトについて

- 一般に、スーパーコンピュータでは、
  MPI実行時の入出力のリダイレクトができません
  - ×例）mpirun ./a.out < in.txt > out.txt

- 専用のリダイレクト命令が用意されています。
- FX1000でリダイレクトを行う場合、以下のオプションを指定します。
  - ○例) mpirun --stdin ./in.txt --stdout out.txt ./a.out

MPIプロセスのノード割り当て

- 「不老」TypeIサブシステムでは、何もしないと(デフォルトでは)、確保ノードが物理的に連続に確保されない
  ⇒通信性能が劣化する場合がある
- 物理的に連続したノード割り当てをしたい場合は、ジョブスクリプトにその形状を記載する
  - ただしノード割り当て形状を指定すると、待ち時間が増加する
- 記載法: #PJM -L "node= <形状>:<機能>"
  - <形状> := { 1次元 | 2次元 | 3次元 }
    - 1次元 := { a }, 2次元 := {a x b}, 3次元 := {a x b x c}
  - <機能> := { 離散 | メッシュ | トーラス }
    - 離散 := { noncont }, メッシュ := {mesh}, トーラス := {torus} : 12ノード以上
- 例:24ノード、3次元(2x4x3)、トーラス
  - #PJM -L "node= 2x4x3 : torus"

NUMA affinity指定について

- NUMA計算機では、MPIプロセスのソケットへの割り当てが性能面で重要となる（NUMA affinityとよぶ）
- MPIプロセスのソケット（富士通用語でCMC）の割り当ては、「不老」TypeIサブシステムでは富士通社のNUMA affinity (MCAパラメタ)で設定する
- 環境変数で設定する

NUMAメモリポリシー指定

- 環境変数名 : plm_ple_memory_allocation_policy
- 代入する値
  - localalloc: プロセスが動作中のCPU(コア)の属するNUMAノードからメモリを割り当てる。
  - interleave_local: プロセスの「ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てる。
  - interleave_nonlocal: プロセスの「非ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てる。
  - interleave_all: プロセスの「全ノード集合」内の各NUMAノードから交互にメモリを取得する。
  - bind_local: プロセスの「ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行う。
  - bind_nonlocal: プロセスの「非ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行う。
  - bind_all: プロセスの「全ノード集合」のNUMAノードにバインドする。
  - prefer_local: プロセスの「ローカルノード集合」のうち、NUMAノードIDが最も若いものを「優先ノード」とし、「優先ノード」からメモリ割り当てを行う。
  - prefer_nonlocal: プロセスの「非ローカルノード集合」のうち、NUMAノードIDが最も若いも
  - のを「優先ノード」とし、「優先ノード」からメモリ割り当てを行う。
- 通常は、localallocでよい。
- export plm_ple_memory_allocation_policy=localalloc

CPU(コア)割り当てポリシー指定

- 環境変数名：plm_ple_numanode_assign_policy
- 代入する値
  - simplex: NUMAノードを占有するように割り当てる。
  - share_cyclic: NUMAノードを他のプロセスと共有するように割り当てる。異なるNUMAノードに順番にプロセスを割り当てる。
  - share_band: NUMAノードを他のプロセスと共有するように割り当てる。同一NUMAノードに連続してプロセスを割り当てる。
- 例） export plm_ple_numanode_assign_policy=simplex
- 各ソケットを各MPIプロセスで独占したいときはsimplexを指定
  - 各ノードへ割り当てるMPIプロセス数が2個で、それぞれのMPIプロセスは16個のスレッド実行するとき
- MPIプロセスをプロセス順に各ソケットに詰め込みたいときは、share_bandを指定
  - ノード当たり32個のMPIプロセスを、ランク番号が122近い順に割り当てたい場合

サンプルプログラムの説明

- Hello/
  - 並列版Helloプログラム
  - hello-pure.bash, hello-hy16.bash : ジョブスクリプトファイル
- Cpi/
  - 円周率計算プログラム
  - cpi-pure.bash ジョブスクリプトファイル
- Wa1/
  - 逐次転送方式による総和演算
  - wa1-pure.bash ジョブスクリプトファイル
- Wa2/
  - 二分木通信方式による総和演算
  - wa2-pure.bash ジョブスクリプトファイル
- Cpi_m/
  - 円周率計算プログラムに時間計測ルーチンを追加したもの
  - cpi_m-pure.bash ジョブスクリプトファイル

MPIプログラム実例

MPIの起動

- MPIを起動するには
  1. MPIをコンパイルできるコンパイラでコンパイル
     - 実行ファイルは a.out とする（任意の名前を付けられます）
  2. 以下のコマンドを実行
     - インタラクティブ実行では、以下のコマンドを直接入力
     - バッチジョブ実行では、ジョブスクリプトファイル中に記載

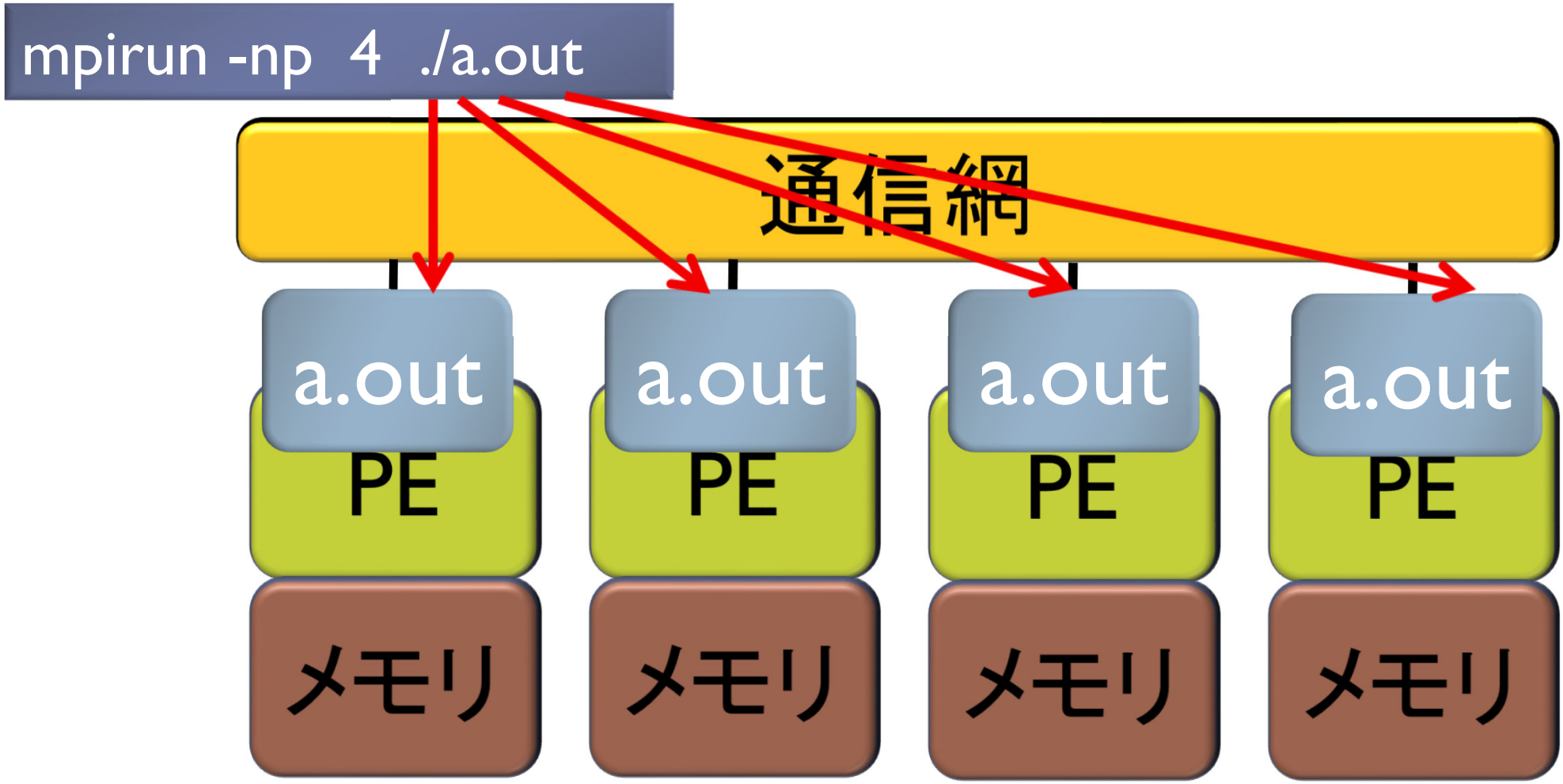
```
$ mpirun  –np  8  ./a.out
```

- mpirun: MPI起動コマンド
- –np 8: MPIプロセス数
- ./a.out: MPIの実行ファイル名

※スパコンのバッチジョブ実行では、MPIプロセス数は専用の指示文で指定する場合があります。
その場合は以下になることがあります。

$mpirun ./a.out

MPIの起動


mpirun -np 4 ./a.out
通信網
a.out
PE
メモリ
a.out
PE
メモリ
a.out
PE
メモリ
a.out
PE
メモリ

並列版Helloプログラムの説明（C言語）

このプログラムは、全PEで起動される

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

void main(int argc, char* argv[]) {

    int    myid, numprocs;
    int    ierr, rc;

    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

    printf("Hello parallel world!  Myid:%d ¥n", myid);

    rc = MPI_Finalize();

    exit(0);
}
```

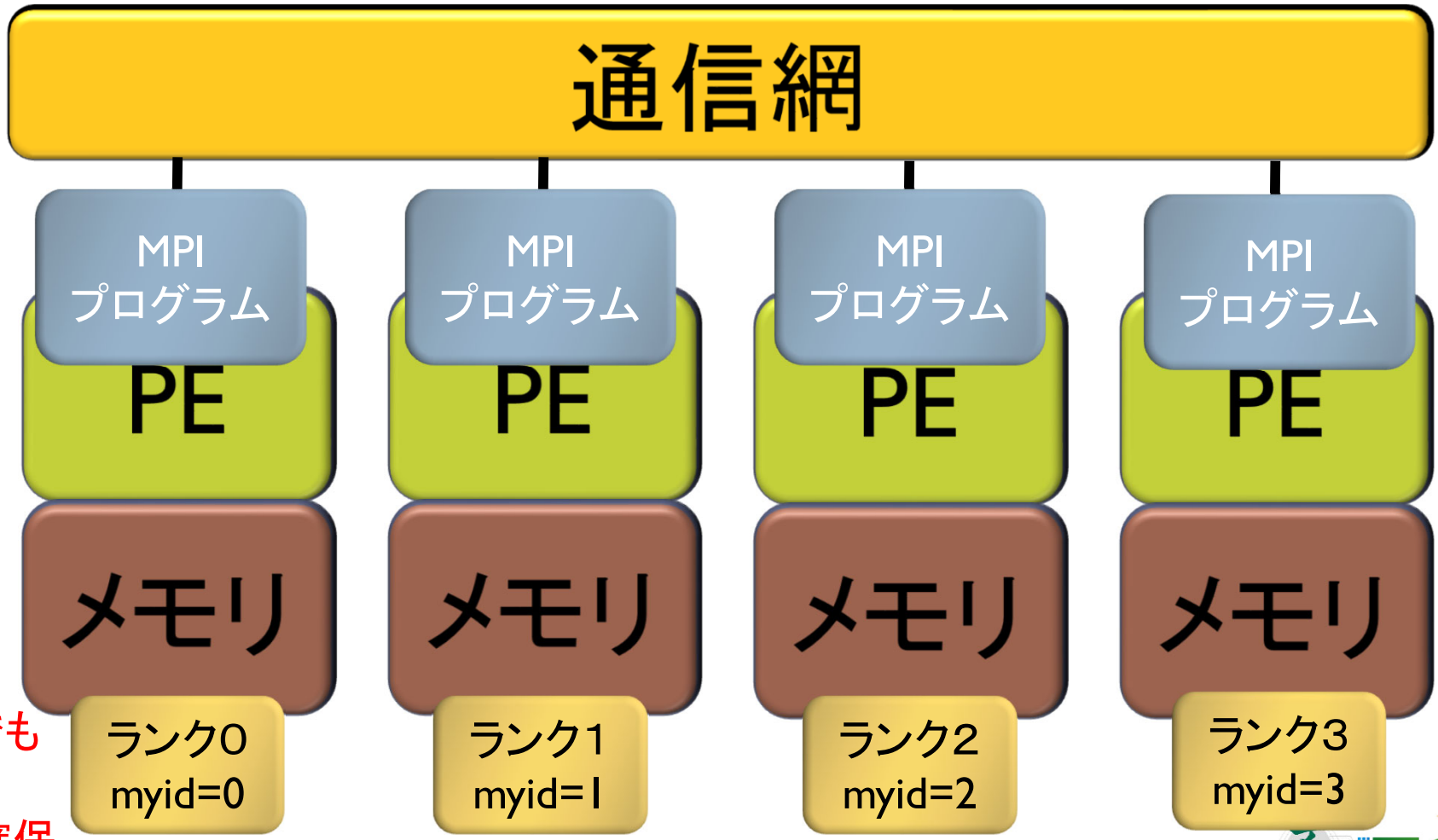
MPIの初期化

自分のID番号を取得
：各PEで値は異なる

全体のプロセッサ台数を取得
：各PEで値は同じ

MPIの終了

変数myidの説明図



同じ変数名でも
別メモリ上
に別変数で確保

並列版Helloプログラムの説明（Fortran言語）

このプログラムは、全PEで起動される

```
program main
include 'mpif.h'
common /mpienv/myid,numprocs

integer  myid, numprocs
integer  ierr

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

print *, "Hello parallel world!  Myid:", myid

call MPI_FINALIZE(ierr)

stop
end
```

MPIの初期化

自分のID番号を取得
：各PEで値は異なる

全体のプロセッサ台数を取得
：各PEで値は同じ

MPIの終了

プログラム出力例

- 4プロセス実行の出力例

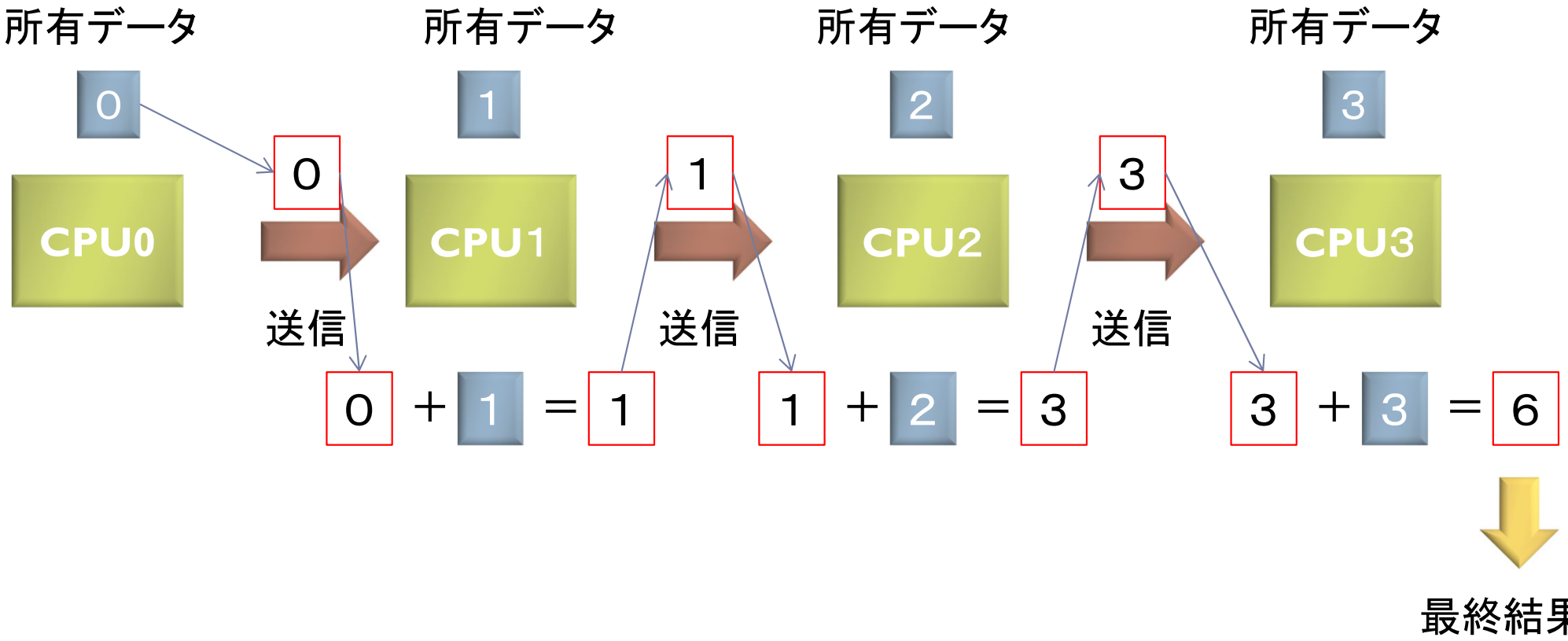
```
Hello parallel world!  Myid:0
Hello parallel world!  Myid:3
Hello parallel world!  Myid:1
Hello parallel world!  Myid:2
```

- 4プロセスなので、表示が4個でる
  （1000プロセスなら1000個出力がでる）
- myid番号が表示される。全体で重複した番号は無い。
- 必ずしも、myidが0から3まで、連続して出ない
  →各行は同期して実行されていない
  →実行ごとに結果は異なる

総和演算プログラム（逐次転送方式）

- 各プロセスが所有するデータを、全プロセスで加算し、
  あるプロセス1つが結果を所有する演算を考える。
- 素朴な方法（逐次転送方式）
  1. （0番でなければ）左隣のプロセスからデータを受信する；
  2. 左隣のプロセスからデータが来ていたら；
     1. 受信する;
     2. ＜自分のデータ＞と＜受信データ＞を加算する;
     3. （最終ランクでなければ）右隣のプロセスに＜2の加算した結果を＞送信する;
     4. 処理を終了する;
- 実装上の注意
  - 左隣りとは、（myid-1）のIDをもつプロセス
  - 右隣りとは、（myid+1）のIDをもつプロセス
    - myid=0のプロセスは、左隣りはないので、受信しない
    - myid=p-1のプロセスは、右隣りはないので、送信しない

バケツリレー方式による加算



１対１通信利用例（逐次転送方式、C言語）

```
void main(int argc, char* argv[]) {
  MPI_Status istatus;
….
  dsendbuf = myid;
  drecvbuf = 0.0;
  if (myid != 0) {
     ierr = MPI_Recv(&drecvbuf, 1, MPI_DOUBLE, myid-1, 0,
                MPI_COMM_WORLD, &istatus);
  }
  dsendbuf = dsendbuf + drecvbuf;
  if (myid != nprocs-1) {
    ierr = MPI_Send(&dsendbuf, 1, MPI_DOUBLE, myid+1, 0,
                MPI_COMM_WORLD);
  }
  if (myid == nprocs-1) printf ("Total = %4.2lf ¥n", dsendbuf);
….
}
```

受信用システム配列の確保

自分より一つ少ないID番号（myid-1）から、double型データ1つを受信しdrecvbuf変数に代入

自分より一つ多いID番号（myid+1）に、dsendbuf変数に入っているdouble型データ1つを送信

1対1通信利用例（逐次転送方式、Fortran言語）

```
      program main
      integer  istatus(MPI_STATUS_SIZE)
      ....
      dsendbuf = myid
      drecvbuf = 0.0
      if (myid .ne. 0) then
        call MPI_RECV(drecvbuf, 1, MPI_DOUBLE_PRECISION,
     &              myid-1, 0, MPI_COMM_WORLD, istatus, ierr)
      endif
      dsendbuf = dsendbuf + drecvbuf
      if (myid .ne. numprocs-1) then
        call MPI_SEND(dsendbuf, 1, MPI_DOUBLE_PRECISION,
     &          myid+1, 0, MPI_COMM_WORLD, ierr)
      endif
     if (myid .eq. numprocs-1) then
        print *, "Total = ", dsendbuf
      endif
      ....
      stop
      end
```

受信用システム配列の確保

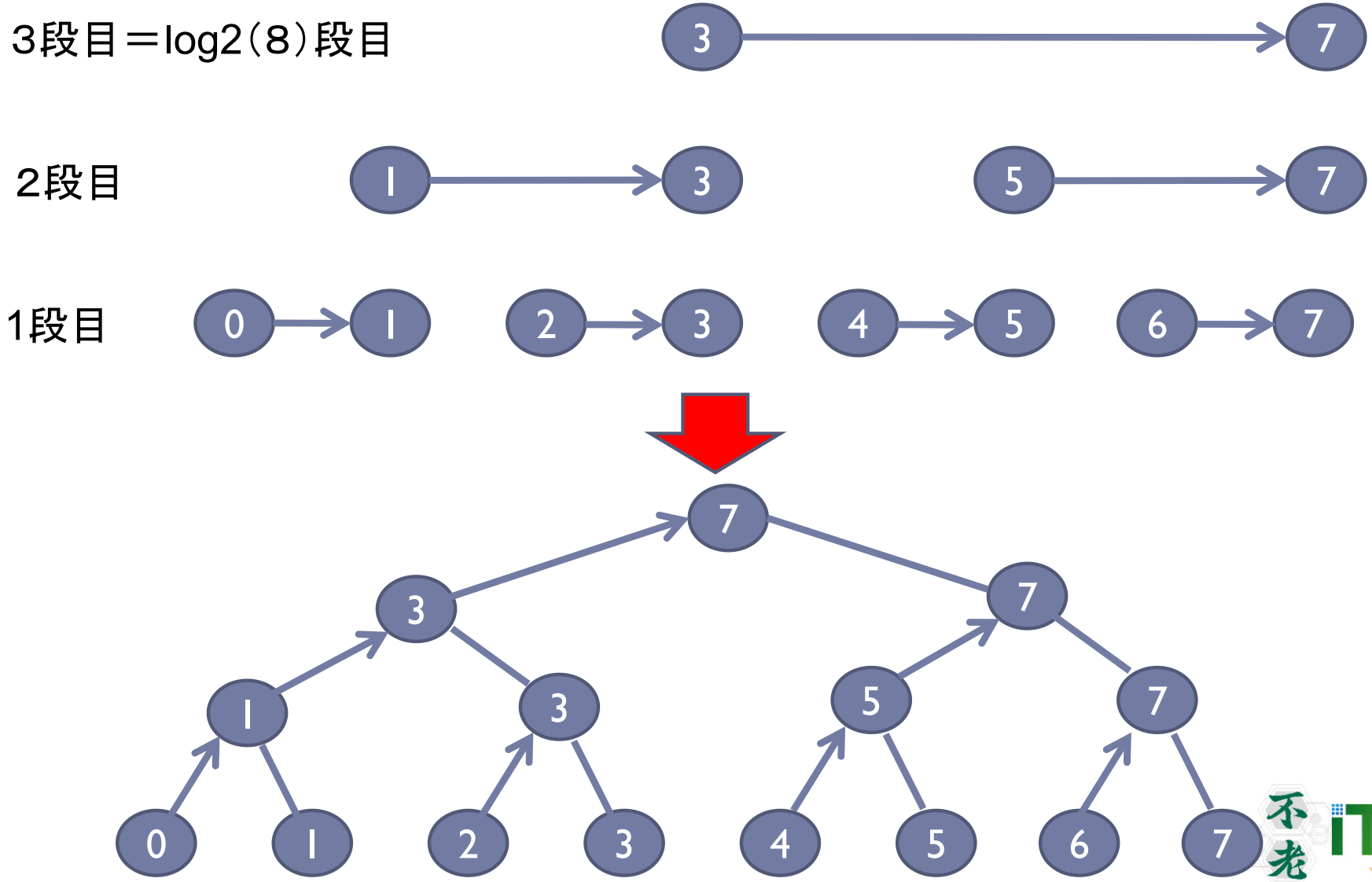
自分より一つ少ないID番号（myid-1）から、double型データ1つを受信しdrecvbuf変数に代入

自分より一つ多いID番号（myid+1）に、dsendbuf変数に入っているdouble型データ1つを送信

総和演算プログラム（二分木通信方式）

- 二分木通信方式
  1. k = 1;
  2. for (i=0; i < log2(nprocs); i++)
  3. if ((myid & k) == k)
     - (myid – k)番 プロセス からデータを受信;
     - 自分のデータと、受信データを加算する;
     - k = k * 2;
  4. else
     - (myid + k)番 プロセス に、データを転送する;
     - 処理を終了する;

総和演算プログラム（二分木通信方式）



総和演算プログラム（二分木通信方式）

- 実装上の工夫
  - 要点：プロセス番号の2進数表記の情報を利用する
  - 第i段において、受信するプロセスの条件は、以下で書ける：

    myid & k が k と一致
    - ここで、k = 2^(i-1) 。
    - つまり、プロセス番号の2進数表記で右からi番目のビットが立っているプロセスが、送信することにする
  - また、送信元のプロセス番号は、以下で書ける：

    myid + k
    - つまり 、通信が成立するPE番号の間隔は2^(i-1) ←二分木なので
  - 送信プロセスについては、上記の逆が成り立つ。

総和演算プログラム（二分木通信方式）

- 逐次転送方式の通信回数
  - 明らかに、nprocs－１ 回
- 二分木通信方式の通信回数
  - 見積もりの前提
    - 各段で行われる通信は、完全に並列で行われる（通信の衝突は発生しない）
  - 段数の分の通信回数となる
  - つまり、log2(nprocs) 回
- 両者の通信回数の比較
  - プロセッサ台数が増すと、通信回数の差（＝実行時間）がとても大きくなる
  - 1024構成では、1023回　対　10回！
  - でも、必ずしも二分木通信方式がよいとは限らない（通信衝突の多発）

性能プロファイラ

- 富士通コンパイラには、性能プロファイラ機能がある
- 富士通コンパイラでコンパイル後、実行コマンドで指定し利用する
- 以下の3種類があります

1. 基本プロファイラ
   - 主な用途：プログラム全体で、最も時間のかかっている関数を同定する
2. 詳細プロファイラ
   - 主な用途：最も時間のかかっている関数内の特定部分において、メモリアクセス効率、キャッシュヒット率、スレッド実行効率、MPI通信頻度解析、を行う
3. CPU性能解析レポート
   - 主な用途：詳細プロファイラデータを視覚的に表示する（Excel利用）

性能プロファイラの種類の詳細

- 基本プロファイラ
  - コマンド例：fipp –C
  - 表示コマンド：fipppx
  - ユーザプログラムに対し一定間隔(デフォルト時100 ミリ秒間隔)毎に割り込みをかけ情報を収集する。
  - 収集した情報を基に、コスト情報等の分析結果を表示。
  - 測定場所を指定可能。
- 詳細プロファイラ
  - コマンド例：fapp –C
  - 表示コマンド：fapppx
  - ユーザプログラムの中に測定範囲を設定し、測定範囲のハードウェアカウンタの値を収集。
  - 収集した情報を基に、MFLOPS、MIPS、各種命令比率、キャッシュミス等の詳細な分析結果を表示。

基本プロファイラ利用例

- プロファイラデータ用の空のディレクトリがないとダメ
- /Wa2 に Profディレクトリを作成
  $ mkdir Prof
- Wa2 の wa2-pure.bash 中に以下を記載
  fipp -C -d Prof mpirun ./wa2
- 実行する
  $ pjsub wa2-pure.bash
- テキストプロファイラを起動
  $ fipppx –A -d Prof

基本プロファイラ出力例（1/2）

```
------------------------------------------------------------------------------------
Fujitsu Instant Profiler Version 1.2.0
  Measured time             : Thu Apr 19 09:32:18 2012
  CPU frequency             : Process     0 -    127  1848 (MHz)
  Type of program           : MPI
  Average at sampling interval  : 100.0 (ms)
  Measured range            : All ranges
  Virtual coordinate        : (12, 0, 0)
------------------------------------------------------------------------------------
____________________________________________________________________________________

Time statistics
      Elapsed(s)      User(s)     System(s)
  ------------------------------------------
        2.1684       53.9800       87.0800   Application
  ------------------------------------------
        2.1684        0.5100        0.6400   Process     11
        2.1588        0.4600        0.6800   Process     88
        2.1580        0.5000        0.6400   Process     99
        2.1568        0.6600        1.4200   Process    111
...
```

基本プロファイラ出力例（2/2）

```
Procedures profile
  ********************************************************************************************
     Application - procedures
  ********************************************************************************************
          Cost       %         Mpi       %   Start     End
  ----------------------------------------------------------------
           475  100.0000         312   65.6842      --      --  Application
  ----------------------------------------------------------------
           312   65.6842         312  100.0000       1      45  MAIN__
            82   17.2632           0    0.0000      --      --  __GI___sched_yield
            80   16.8421           0    0.0000      --      --  __libc_poll
             1    0.2105           0    0.0000      --      --  __pthread_mutex_unlock_usercnt
  ********************************************************************************************
     Process    11 - procedures
  ********************************************************************************************
          Cost       %         Mpi       %   Start     End
  ----------------------------------------------------------------
             5  100.0000           4   80.0000      --      --  Process    11
  ----------------------------------------------------------------
             4   80.0000           4  100.0000       1      45  MAIN__
             1   20.0000           0    0.0000      --      --  __GI___sched_yeld
....
```

CPU解析レポート（エクセル形式）

- 性能プロファイルは見にくい
- 性能プロファイルデータ(マシン語命令の種類や、実行時間に占める割合など)を、Excelで可視化してくれるツール
- コマンド例：fapp –c –Hevent=pa1 ./a.out
- 単体レポート：　１回測定
- 標準レポート：１１回測定
- 詳細レポート：１７回測定

CPU解析レポート（エクセル形式）

- 手順
  1. 対象箇所（ループ）を、専用のAPIで指定する
  2. プロファイルを入れるフォルダを＜測定数分＞か所をつくる
  3. プロファイルのためのコマンドで＜測定数分＞回実行する
  4. エクセル形式に変換する
  5. 4のエクセル形式を手元のパソコンに持ってくる
  6. 5のファイルを、指定のエクセルと同一のフォルダに入れてから、指定のエクセルを開く

CPU解析レポートのための指示ＡＰＩ

- 以下のAPIで、対象となるループを挟む（Fortranの場合）

```
call fapp_start (“region”, 1)
    <対象となるループ>
call fapp_stop (“region”, 1)
```

- 詳細プロファイラの指定APIと同じです
- “region”は、対象となる場所の名前なので、任意の名前を付けることが可能（後で、専用エクセルを開くときに使う）
- “1”は、レベルの指定で、数字を書く
  - -L オプションで指定したレベル以上を測定

実行のさせ方

- a.out という実行ファイルの場合、以下のように11回実行する
- 実行するディレクトリに、pa1、pa2、…、pa11という、ディレクトリを作っておく必要がある
- 以下のように実行する

```
fapp -C –d pa1 -Hpa=1 mpiexec a.out
fapp -C –d pa2 -Hpa=2 mpiexec a.out
fapp -C –d pa3 -Hpa=3 mpiexec a.out
fapp -C –d pa4 -Hpa=4 mpiexec a.out
fapp -C –d pa5 -Hpa=5 mpiexec a.out
fapp -C –d pa6 -Hpa=6 mpiexec a.out
   ….
fapp -C –d pa11 -Hpa=11 mpiexec a.out
```

エクセルデータへの変換

- pa1、pa2、…、pa11の中に、プロファイルデータがあることを確認する
- 以下のコマンドを実行する

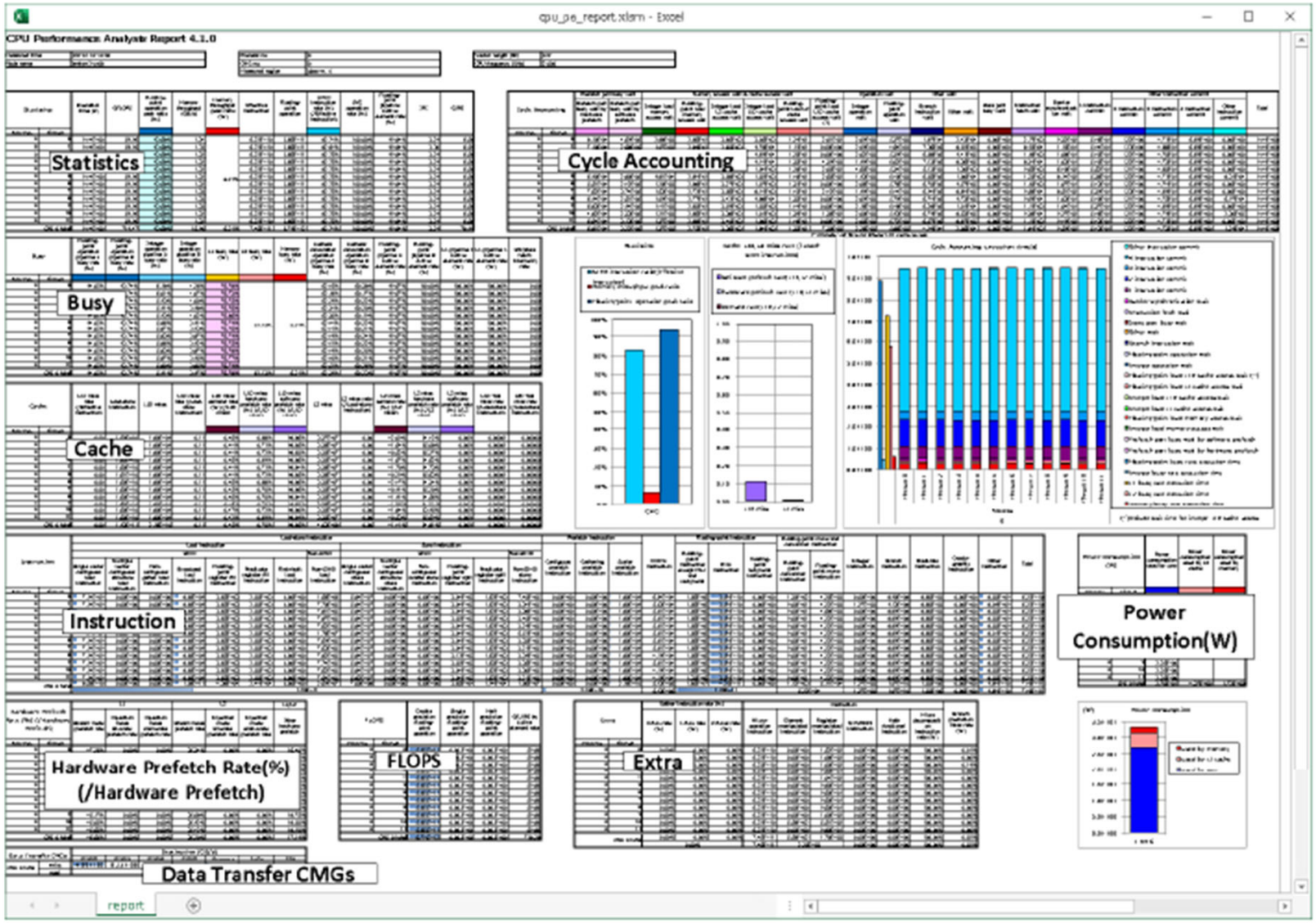
```
fapppx -A -d pa1 -o output_prof_1.csv -tcsv -Hpa
fapppx -A -d pa2 -o output_prof_2.csv -tcsv -Hpa
fapppx -A -d pa3 -o output_prof_3.csv -tcsv -Hpa
fapppx -A -d pa4 -o output_prof_4.csv -tcsv -Hpa
fapppx -A -d pa5 -o output_prof_5.csv -tcsv -Hpa
fapppx -A -d pa6 -o output_prof_6.csv -tcsv -Hpa
  ….
fapppx -A -d pa11 -o output_prof_11.csv -tcsv -Hpa
```

エクセルデータを手元のＰＣに転送

- 以下の「不老」上のエクセルデータを、手元のPCに転送
  - output_prof_1.csv、output_prof_2.csv、…、output_prof_11.csv
- 上記のエクセルデータが入ったフォルダで、ポータル上で公開されている専用エクセルを開く
- 詳細なエクセルデータの利用法、分析されたデータの見方は、マニュアル参照

表示例

図4.5 CPU性能解析レポートの構成



ソース：FUJITSU Software Technical Computing Suite V4.0L20 Development Studioプロファイラ使用手引書, J2UL-2483-02Z0(00), 2020年3月

表示例

図4.6 表の構成

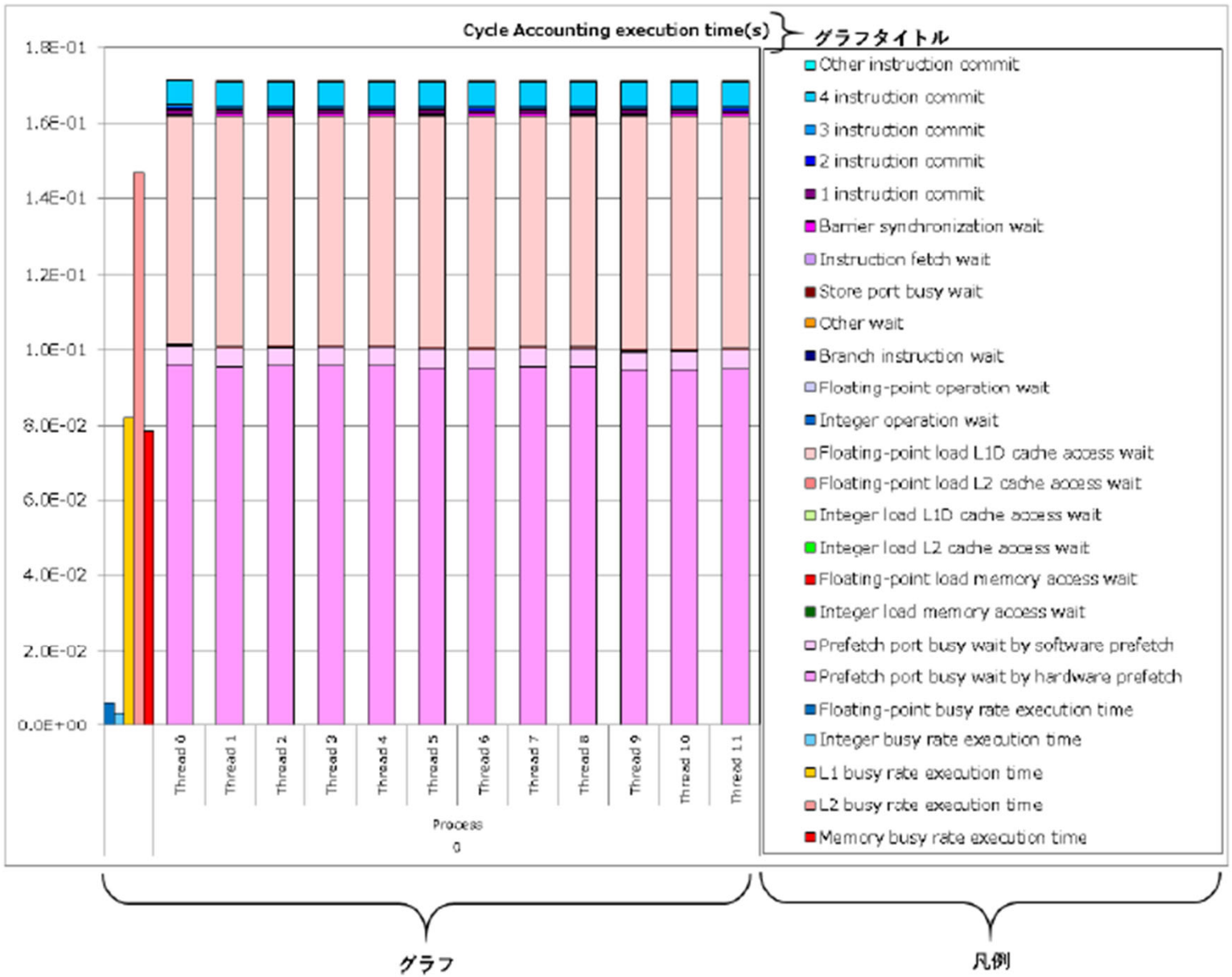
Cycle Accounting		Operation wait		Other wait	
		Integer operation wait	Floating-point operation wait	Branch instruction wait	Other wait
Process	Thread				
0	0	5.27E-06	4.14E-07	4.92E-06	1.86E-05
0	1	2.47E-06	3.34E-07	1.85E-06	1.36E-05
0	2	3.43E-06	3.91E-07	1.81E-06	1.48E-05
0	3	3.35E-06	3.56E-07	2.12E-06	1.49E-05
0	4	2.51E-06	4.36E-07	2.15E-06	1.37E-05
0	5	3.30E-06	3.46E-07	2.08E-06	1.49E-05
0	6	3.20E-06	4.47E-07	2.20E-06	1.47E-05
0	7	3.50E-06	3.39E-07	2.12E-06	1.55E-05
0	8	3.22E-06	4.13E-07	2.08E-06	1.49E-05
0	9	3.36E-06	3.27E-07	2.05E-06	1.49E-05
0	10	3.25E-06	4.37E-07	1.63E-06	1.53E-05
0	11	3.15E-06	3.78E-07	2.52E-06	1.43E-05
CMG 0 total		3.33E-06	3.85E-07	2.29E-06	1.50E-05

表タイトル
項目名
項目色
プロセス番号 + スレッド番号
算出値
CMG番号
代表値

ソース：FUJITSU Software Technical Computing Suite V4.0L20 Development Studioプロファイラ使用手引書, J2UL-2483-02Z0(00), 2020年3月

表示例

図4.7 グラフの構成



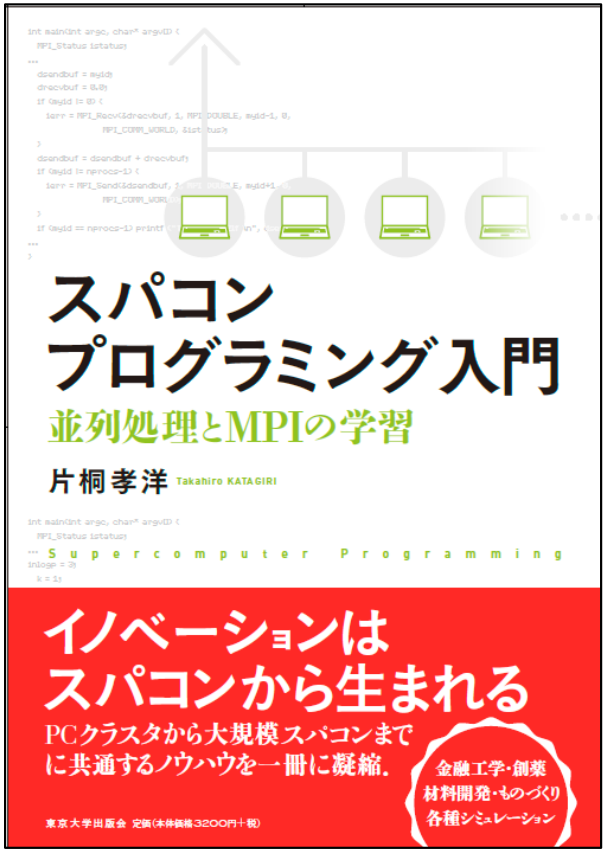
ソース：FUJITSU Software Technical Computing Suite V4.0L20 Development Studioプロファイラ使用手引書, J2UL-2483-02Z0(00), 2020年3月

演習課題

1. 逐次転送方式のプログラムを実行
   - Wa1 のプログラム
2. 二分木通信方式のプログラムを実行
   - Wa2のプログラム
3. 時間計測プログラムを実行
   - Cpi_mのプログラム
4. プロセス数を変化させて、サンプルプログラムを実行
5. Helloプログラムを、以下のように改良
   - MPI_Sendを用いて、プロセス0からChar型のデータ“Hello World!!”を、その他のプロセスに送信する
   - その他のプロセスでは、MPI_Recvで受信して表示する

並列プログラミングの基本（座学）

教科書（演習書）

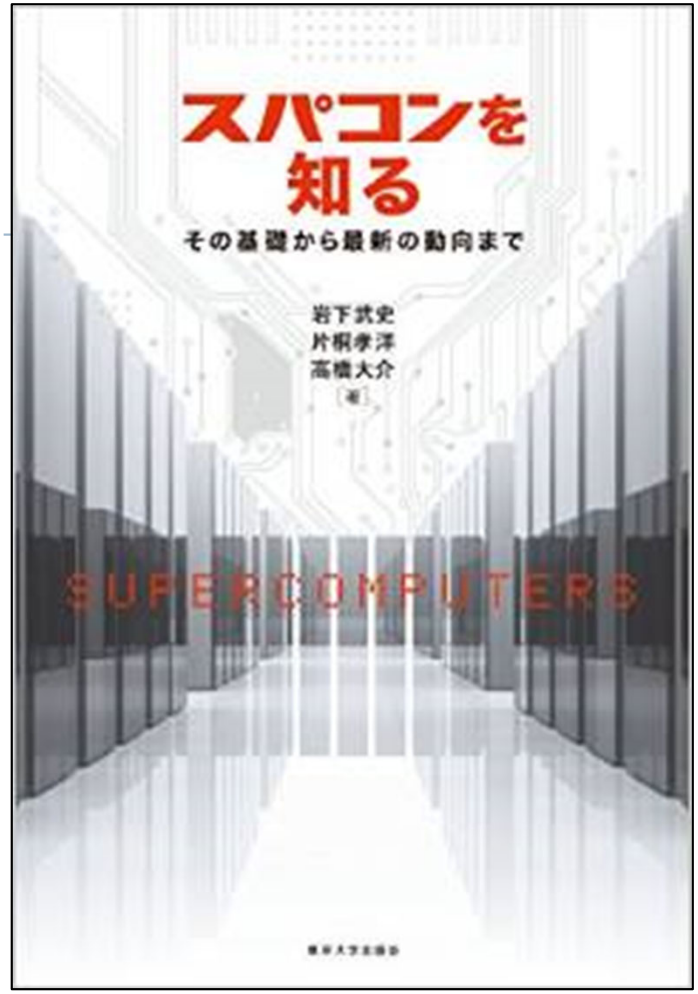


- 「スパコンプログラミング入門
  ー並列処理とMPIの学習ー」
  - 片桐 孝洋　著、
  - 東大出版会、ISBN978-4-13-062453-4、
    発売日：2013年3月12日、判型:A5, 200頁
  - 【本書の特徴】
    - C言語で解説
    - C言語、Fortran90言語のサンプルプログラムが付属
    - 数値アルゴリズムは、図でわかりやすく説明
    - 本講義の内容を全てカバー
    - 内容は初級。初めて並列数値計算を学ぶ人向けの
      入門書

教科書（演習書）

- 「並列プログラミング入門：
  サンプルプログラムで学ぶOpenMPとOpenACC」
  - 片桐 孝洋　著
  - 東大出版会、ISBN-10: 4130624563、
    ISBN-13: 978-4130624565、発売日： 2015年5月25日
  - 【本書の特徴】
    - C言語、Fortran90言語で解説
    - C言語、Fortran90言語の複数のサンプルプログラムが入手可能
      （ダウンロード形式）
    - 本講義の内容を全てカバー
    - Windows PC演習可能(Cygwin利用)。スパコンでも演習可能。
    - 内容は初級。初めて並列プログラミングを学ぶ人向けの
      入門書

参考書



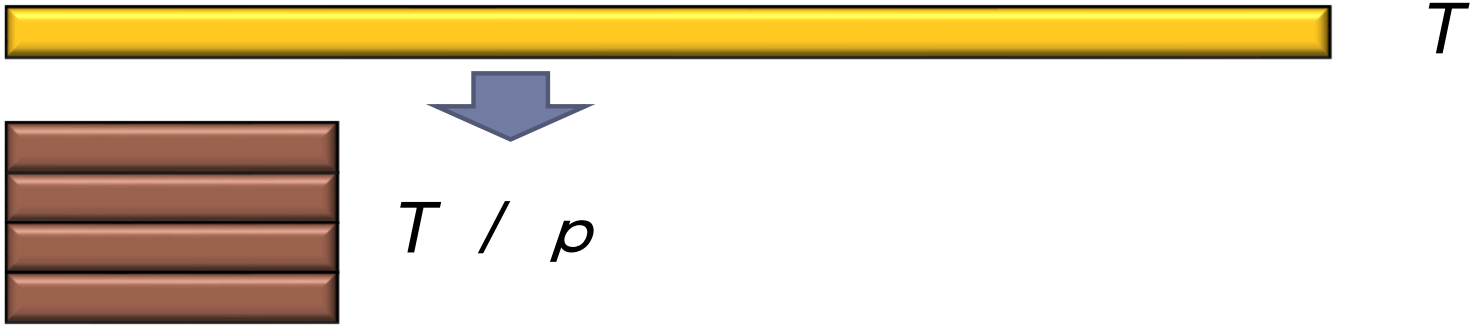
- 「スパコンを知る:
  その基礎から最新の動向まで」
  - 岩下武史、片桐孝洋、高橋大介 著
  - 東大出版会、ISBN-10: 4130634550、
    ISBN-13: 978-4130634557、
    発売日：2015年2月20日、176頁
  - 【本書の特徴】
    - スパコンの解説書です。以下を
      分かりやすく解説します。
      - スパコンは何に使えるか
      - スパコンはどんな仕組みで、なぜ速く計算できるのか
      - 最新技術、今後の課題と将来展望、など

参考書

- 「並列数値処理 – 高速化と性能向上のために –」
  - 金田康正 東大教授 理博 編著、
    片桐孝洋 東大特任准教授 博士(理学) 著、黒田久泰 愛媛大准教授 博士(理学) 著、山本有作 神戸大教授 博士(工学) 著、五百木伸洋 ㈱日立製作所 著、
  - コロナ社、発行年月日:2010/04/30, 判 型: A5, ページ数:272頁、ISBN:978-4-339-02589-7, 定価:3,990円(本体3,800円＋税5%)
  - 【本書の特徴】
    - Fortran言語で解説
    - 数値アルゴリズムは、数式などで厳密に説明
    - 本講義の内容に加えて、固有値問題の解法、疎行列反復解法、FFT、ソート、など、主要な数値計算アルゴリズムをカバー
    - 内容は中級～上級。専門として並列数値計算を学びたい人向き

スパコンの基本：並列処理
並列化とは何か？

- 逐次実行のプログラム（実行時間T）を、p台の計算機を使って、T / pにすること。



- 素人考えでは自明。
- 実際は、できるかどうかは、対象処理の内容（アルゴリズム）で　大きく　難しさが違う
  - アルゴリズム上、絶対に並列化できない部分の存在
  - 通信のためのオーバヘッドの存在
    - 通信立ち上がり時間
    - データ転送時間

並列と並行

- 並列（Parallel）
  - 物理的に並列（時間的に独立）
  - ある時間に実行されるものは多数

- 並行（Concurrent）
  - 論理的に並列（時間的に依存）
  - ある時間に実行されるものは1つ（＝1プロセッサで実行）

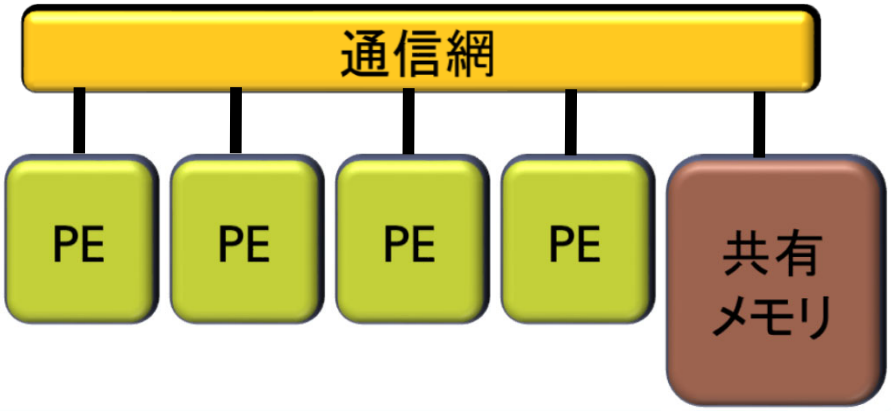
  - 時分割多重、疑似並列
  - OSによるプロセス実行スケジューリング（ラウンドロビン方式）

並列計算機の分類

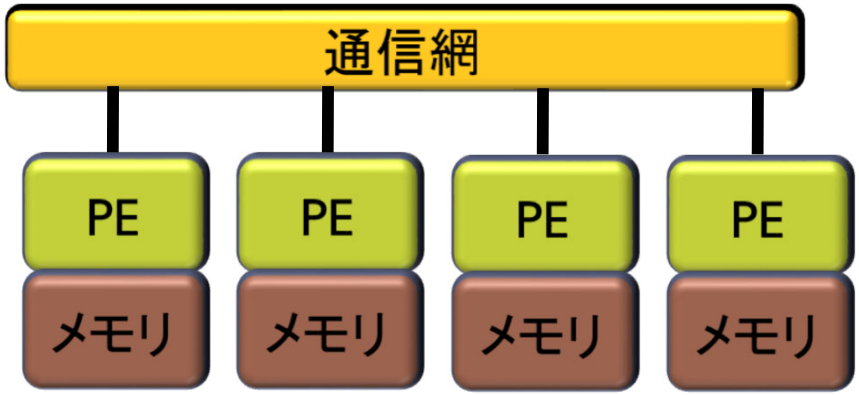
- Michael J. Flynn教授（スタンフォード大）の分類（1966）
- 単一命令・単一データ流
  （SISD, Single Instruction Single Data Stream）
- 単一命令・複数データ流
  （SIMD, Single Instruction Multiple Data Stream）
- 複数命令・単一データ流
  （MISD, Multiple Instruction Single Data Stream）
- 複数命令・複数データ流
  （MIMD, Multiple Instruction Multiple Data Stream）

並列計算機のメモリ型による分類

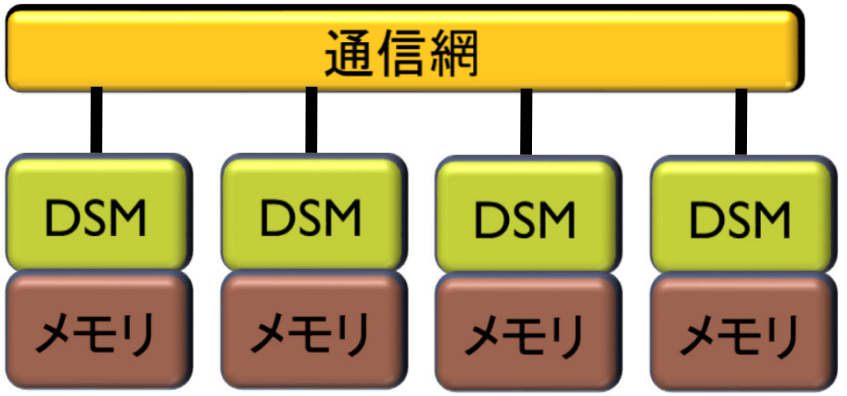
1. 共有メモリ型
（SMP、
Symmetric Multiprocessor）



2. 分散メモリ型
（メッセージパッシング）

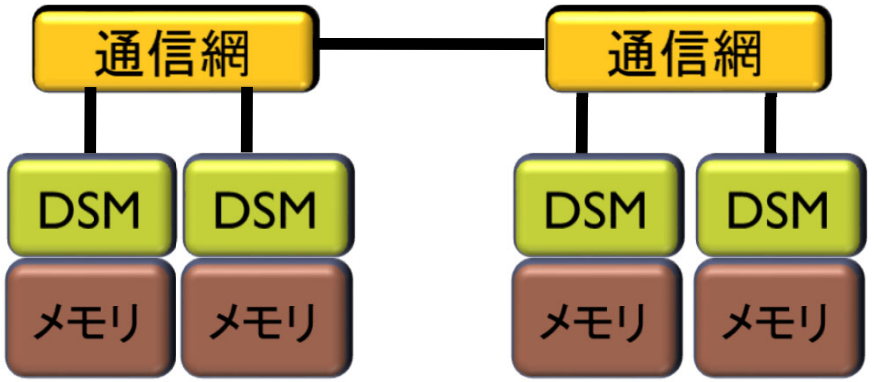


3. 分散共有メモリ型
（DSM、
Distributed Shared Memory）



並列計算機のメモリ型による分類

4. （分散メモリ型並列計算機）
   共有・非対称メモリ型
（ccNUMA、Cache Coherent Non-Uniform Memory Access）

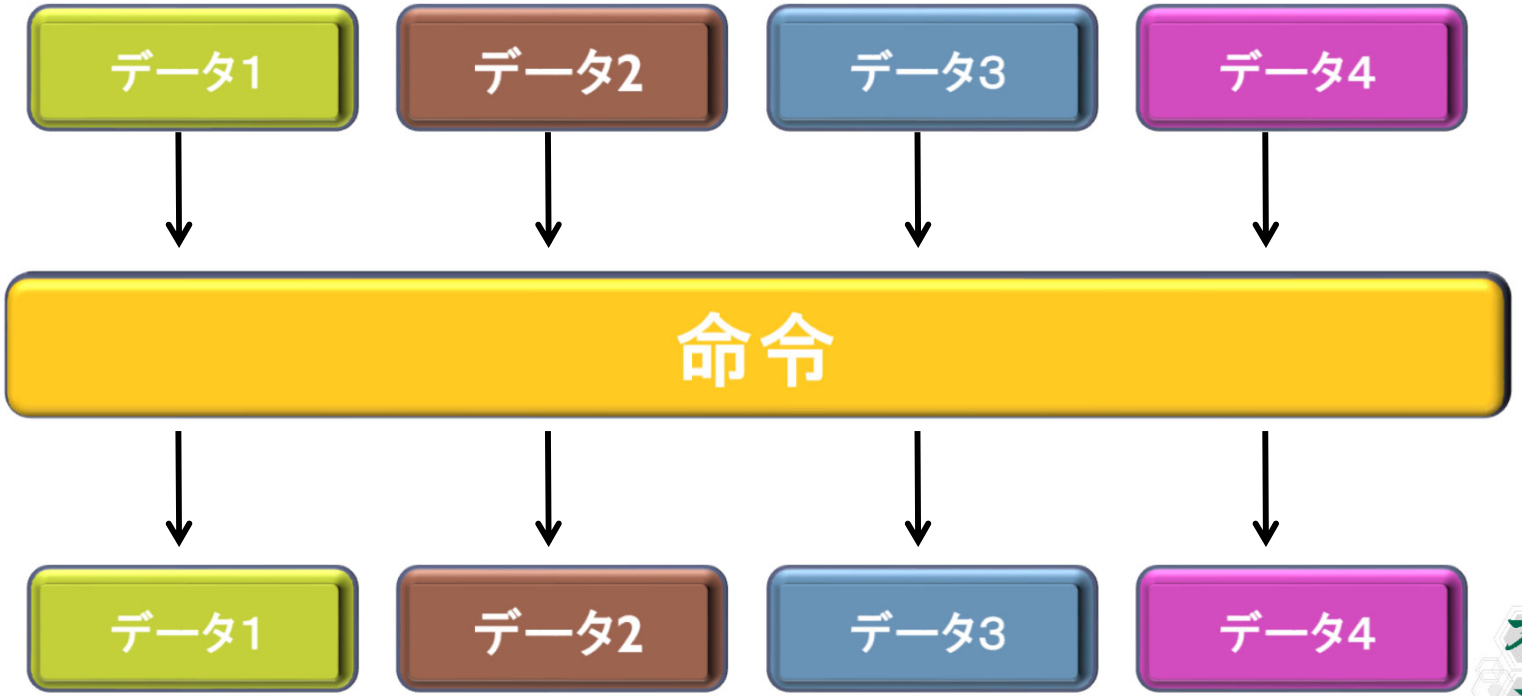


並列計算機の分類とMPIとの関係

- MPIは分散メモリ型計算機を想定
  - MPIは、分散メモリ間の通信を定めているため
- MPIは共有メモリ型計算機でも動く
  - MPIは、共有メモリ内でもプロセス間通信ができるため
- MPIを用いたプログラミングモデルは、（基本的に）SIMD
  - MPIは、（基本的には）プログラムが1つ（＝命令と等価）しかないが、データ（配列など）は複数あるため

並列プログラミングのモデル

- 実際の並列プログラムの挙動はMIMD
- アルゴリズムを考えるときは＜SIMDが基本＞
  - 複雑な挙動は理解できないので



並列プログラミングのモデル

- MIMD上での並列プログラミングのモデル
  1. SPMD（Single Program Multiple Data）
     - 1つの共通のプログラムが、並列処理開始時に、全プロセッサ上で起動する
     - MPI（バージョン1）のモデル

  2. Master / Worker（Master / Slave）
     - 1つのプロセス（Master）が、複数のプロセス（Worker）を管理（生成、消去）する。

並列プログラムの種類

- マルチプロセス
  - MPI（Message Passing Interface）
  - HPF（High Performance Fortran）
    - 自動並列化Fortranコンパイラ
    - ユーザがデータ分割方法を明示的に記述
- マルチスレッド
  - Pthread (POSIX スレッド)
  - Solaris Thread (Sun Solaris OS用)
  - NT thread (Windows NT系、Windows95以降)
    - スレッドの Fork（分離）と Join（融合）を明示的に記述
  - Java
    - 言語仕様としてスレッドを規定
  - OpenMP
    - ユーザが並列化指示行を記述

プロセスとスレッドの違い
•メモリを意識するかどうかの違い
　•別メモリは「プロセス」
　•同一メモリは「スレッド」

マルチプロセスとマルチスレッドは
共存可能
→ハイブリッドMPI/OpenMP実行

並列処理の実行形態（1）

- データ並列
  - データを分割することで並列化する。
  - データの操作（＝演算）は同一となる。
  - データ並列の例：行列ー行列積

SIMDの
考え方と同じ

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1*9+2*6+3*3 & 1*8+2*5+3*2 & 1*7+2*4+3*1 \\ 4*9+5*6+6*3 & 4*8+5*5+6*2 & 4*7+5*4+6*1 \\ 7*9+8*6+9*3 & 7*8+8*5+9*2 & 7*7+8*4+9*1 \end{pmatrix}$$

●並列化

全CPUで共有

CPU0 $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$

CPU1 $\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$

CPU2 $\begin{pmatrix} 7 & 8 & 9 \end{pmatrix}$

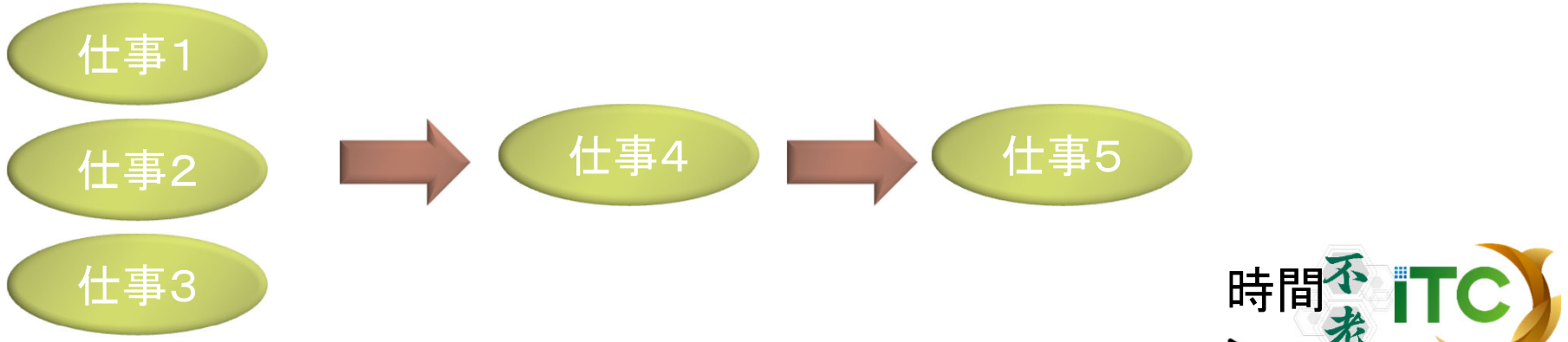
$\begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1*9+2*6+3*3 & 1*8+2*5+3*2 & 1*7+2*4+3*1 \\ 4*9+5*6+6*3 & 4*8+5*5+6*2 & 4*7+5*4+6*1 \\ 7*9+8*6+9*3 & 7*8+8*5+9*2 & 7*7+8*4+9*1 \end{pmatrix}$

並列に計算：初期データは異なるが演算は同一

並列処理の実行形態（２）

- タスク並列
  - タスク(ジョブ)を分割することで並列化する。
  - データの操作(＝演算)は異なるかもしれない。
  - タスク並列の例:カレーを作る
    - 仕事１:野菜を切る
    - 仕事２:肉を切る
    - 仕事３:水を沸騰させる
    - 仕事４:野菜・肉を入れて煮込む
    - 仕事５:カレールゥを入れる

●並列化

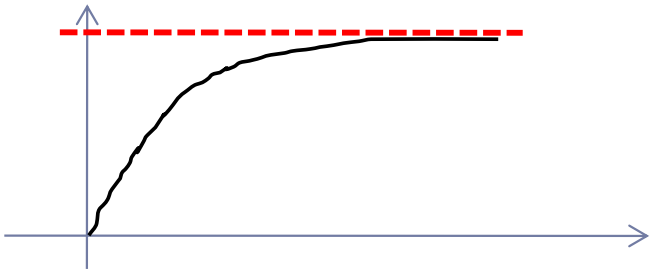


性能評価指標

並列化の尺度

性能評価指標－台数効果

- 台数効果
  - 式:　$S_P = T_S / T_P \quad (0 \leq S_p)$
  - T_S :逐次の実行時間、　T_P :P台での実行時間
  - P台用いて　$S_P = P$ のとき、理想的な(ideal)速度向上
  - P台用いて　$S_P > P$ のとき、スーパリニア・スピードアップ
    - 主な原因は、並列化により、データアクセスが局所化されて、キャッシュヒット率が向上することによる高速化
- 並列化効率
  - 式:　$E_P = S_P / P \times 100 \ (0 \leq E_p) \ [\%]$
- 飽和性能
  - 速度向上の限界
  - Saturation、「さちる」



アムダールの法則

- 逐次実行時間を　K　とする。
  そのうち、並列化ができる割合を　α　とする。
- このとき、台数効果は以下のようになる。

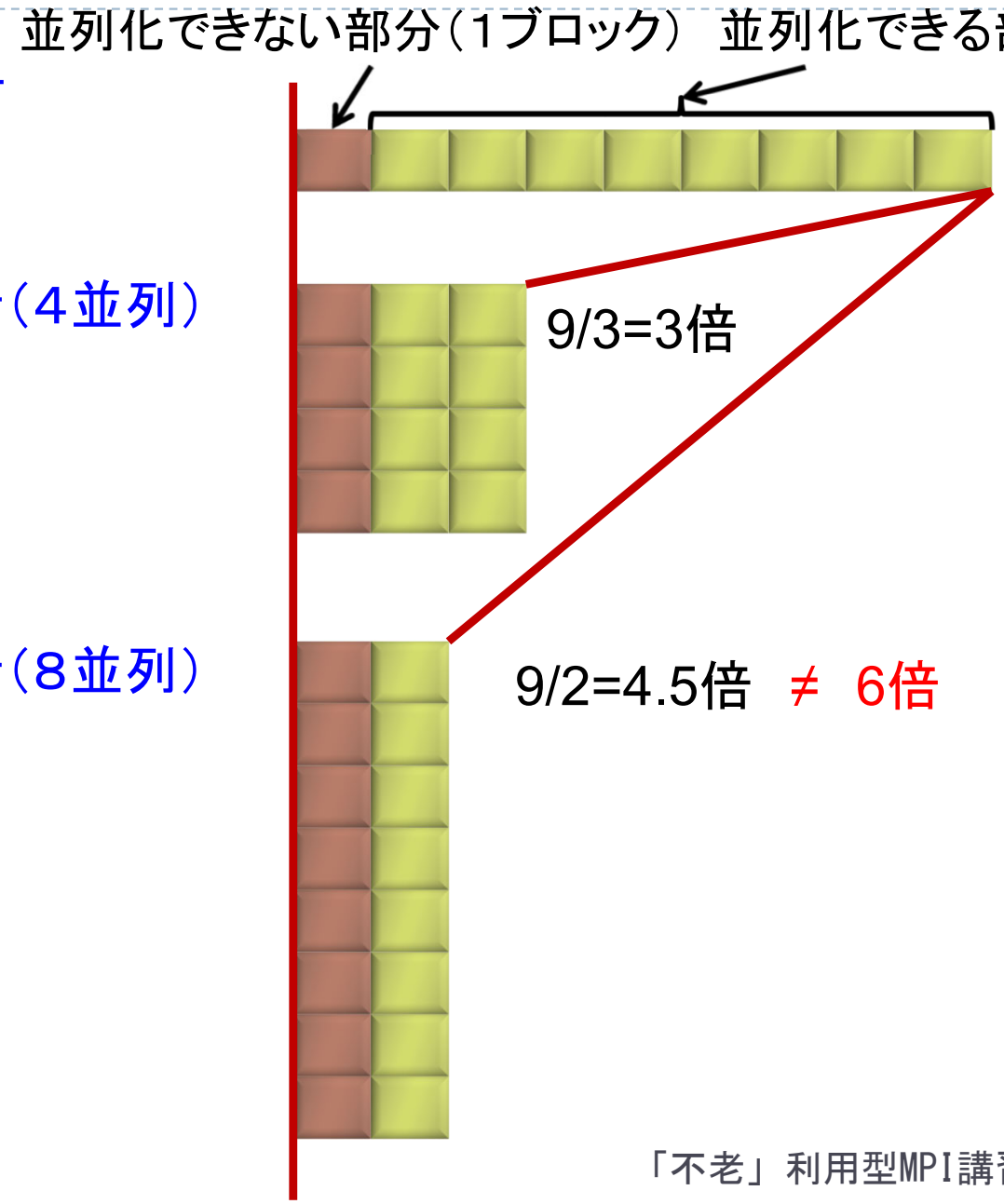
$$S_P = K/(K\alpha/P + K(1-\alpha))$$

$$= 1/(\alpha/P + (1-\alpha)) = 1/(\alpha(1/P-1)+1)$$

- 上記の式から、たとえ無限大の数のプロセッサを使っても（P→∞）、台数効果は、高々　1／（1－α）　である。
  （アムダールの法則）
  - 全体の90%が並列化できたとしても、無限大の数のプロセッサをつかっても、1/(1-0.9) = 10 倍　にしかならない！

→高性能を達成するためには、少しでも並列化効率を上げる実装をすることがとても重要である

アムダールの法則の直観例



●逐次実行

＝88.8%が並列化可能

●並列実行（４並列）

9/3=3倍

●並列実行（８並列）

9/2=4.5倍 ≠ 6倍

内容に関する質問は
katagiri@cc.nagoya-u.ac.jp
まで

MPIの基礎

名古屋大学情報基盤センター　片桐孝洋

MPIの特徴

- メッセージパッシング用のライブラリ規格の1つ
  - メッセージパッシングのモデルである
  - コンパイラの規格、特定のソフトウエアやライブラリを指すものではない！
- 分散メモリ型並列計算機で並列実行に向く
- 大規模計算が可能
  - 1プロセッサにおけるメモリサイズやファイルサイズの制約を打破可能
  - プロセッサ台数の多い並列システム（MPPシステム、Massively Parallel Processingシステム）を用いる実行に向く
    - 1プロセッサ換算で膨大な実行時間の計算を、短時間で処理可能
  - 移植が容易
    - API（Application Programming Interface）の標準化
- スケーラビリティ、性能が高い
  - 通信処理をユーザが記述することによるアルゴリズムの最適化が可能
  - プログラミングが難しい（敷居が高い）

MPIの経緯（1/2）

- MPIフォーラム（http://www.mpi-forum.org/）が仕様策定
  - 1994年5月1.0版（MPI-1）
  - 1995年6月1.1版
  - 1997年7月1.2版、および 2.0版（MPI-2）
- 米国アルゴンヌ国立研究所、およびミシシッピ州立大学で開発
- MPI-2 では、以下を強化：
  - 並列I/O
  - C++、Fortran 90用インターフェース
  - 動的プロセス生成/消滅
    - 主に、並列探索処理などの用途

MPIの経緯 MPI3.1策定

- 以下のページで経緯・ドキュメントを公開中
  - http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
    (Implementation Status, as of June 4, 2015)
- 注目すべき機能
  - ノン・ブロッキングの集団通信機能
    （MPI_IALLREDUCE、など）
  - 片方向通信（RMA、Remote Memory Access)
  - Fortran2008 対応、など

MPIの経緯 MPI4.0策定

- 以下のページで経緯・ドキュメントを公開中
  - http://mpi-forum.org/mpi-40/
- 検討されている機能
  - ハイブリッドプログラミングへの対応
  - MPIアプリケーションの耐故障性（Fault Tolerance, FT）
  - いくつかのアイデアを検討中
    - Active Messages (メッセージ通信のプロトコル)
      - 計算と通信のオーバラップ
      - 最低限の同期を用いた非同期通信
      - 低いオーバーヘッド、パイプライン転送
      - バッファリングなしで、インタラプトハンドラで動く
    - Stream Messaging
    - 新プロファイル・インターフェース

MPIの実装

- MPICH（エム・ピッチ）
  - 米国アルゴンヌ国立研究所が開発
- LAM（Local Area Multicomputer）
  - ノートルダム大学が開発
- その他
  - OpenMPI （FT-MPI、LA-MPI、LAM/MPI、PACX-MPIの統合プロジェクト）
  - YAMPII（（旧）東大・石川研究室）（SCore通信機構をサポート）
- 注意点:メーカ独自機能拡張がなされていることがある

MPIによる通信

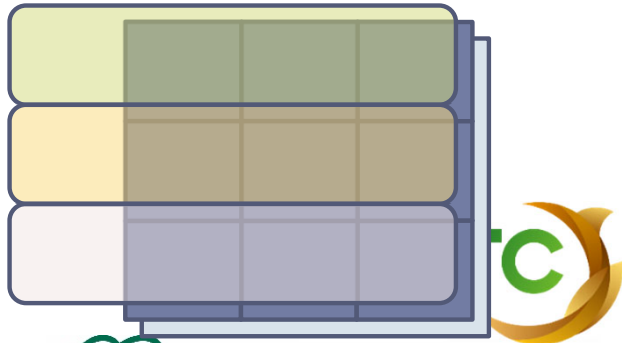
- 郵便物の郵送に同じ
- 郵送に必要な情報：
  1. 自分の住所、送り先の住所
  2. 中に入っているものはどこにあるか
  3. 中に入っているものの分類
  4. 中に入っているものの量
  5. （荷物を複数同時に送る場合の）認識方法（タグ）
- MPIでは：
  1. 自分の認識ID、および、送り先の認識ID
  2. データ格納先のアドレス
  3. データ型
  4. データ量
  5. タグ番号

MPI関数

- システム関数
  - MPI_Init；MPI_Comm_rank；MPI_Comm_size；MPI_Finalize；
- １対１通信関数
  - ブロッキング型
    - MPI_Send；　MPI_Recv；
  - ノンブロッキング型
    - MPI_Isend；　MPI_Irecv；
- １対全通信関数
  - MPI_Bcast
- 集団通信関数
  - MPI_Reduce；　MPI_Allreduce；　MPI_Barrier；
- 時間計測関数
  - MPI_Wtime

コミュニケータ

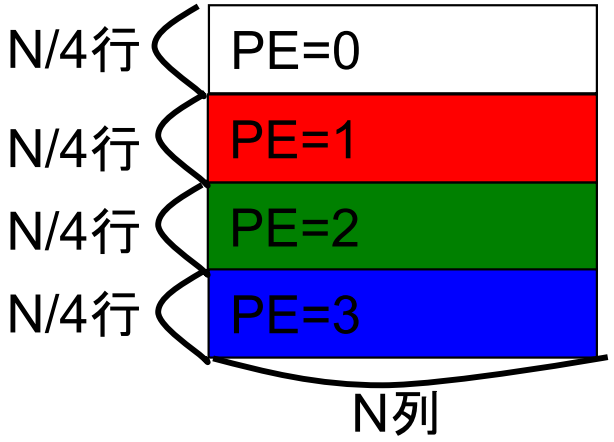
- MPI_COMM_WORLDは、コミュニケータとよばれる概念を保存する変数
- コミュニケータは、操作を行う対象のプロセッサ群を定める
- 初期状態では、0番～numprocs –1番までのプロセッサが、1つのコミュニケータに割り当てられる
  - この名前が、“MPI_COMM_WORLD”
- プロセッサ群を分割したい場合、MPI_Comm_split 関数を利用
  - メッセージを、一部のプロセッサ群に放送するときに利用
  - “マルチキャスト”で利用



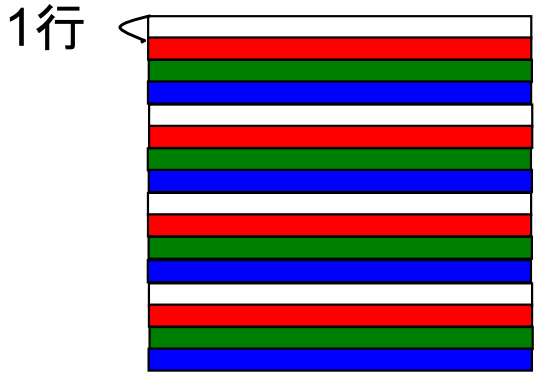
基本演算

- 逐次処理では、「データ構造」が重要
- 並列処理においては、「データ分散方法」が重要になる！
  1. 各PEの「演算負荷」を均等にする
     - ロード・バランシング： 並列処理の基本操作の一つ
     - 粒度調整
  2. 各PEの「利用メモリ量」を均等にする
  3. 演算に伴う通信時間を短縮する
  4. 各PEの「データ・アクセスパターン」を高速な方式にする
     （＝逐次処理におけるデータ構造と同じ）
- 行列データの分散方法
  - ＜次元レベル＞： 1次元分散方式、2次元分散方式
  - ＜分割レベル＞： ブロック分割方式、サイクリック（循環）分割方式

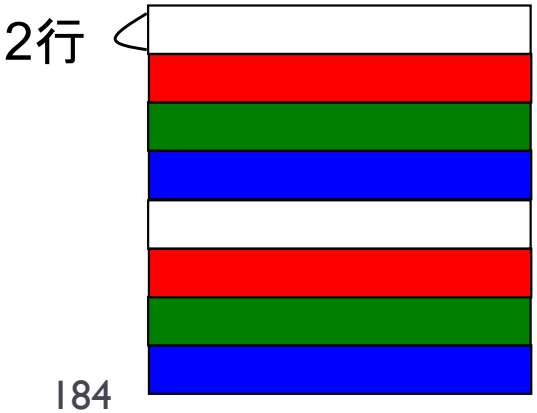
1次元分散



•（行方向）ブロック分割方式
•(Block, *) 分散方式



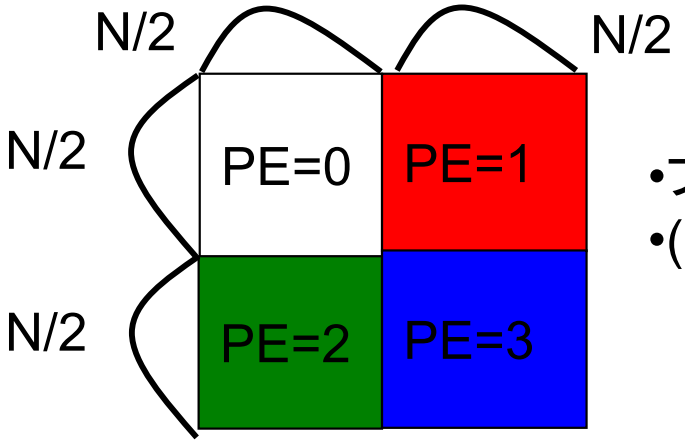
•(行方向) サイクリック分割方式
•(Cyclic, *) 分散方式



•(行方向)ブロック・サイクリック分割方式
•(Cyclic(2), *) 分散方式

この例の「2」: ＜ブロック幅＞とよぶ

2次元分散



- ブロック・ブロック分割方式
- (Block, Block)分散方式

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

- サイクリック・サイクリック分割方式
- (Cyclic, Cyclic)分散方式

0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3

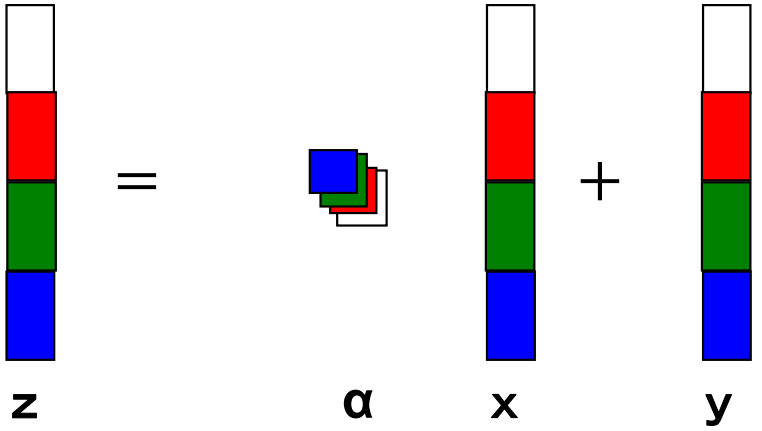
- 二次元ブロック・サイクリック分割方式
- (Cyclic(2), Cyclic(2))分散方式

ベクトルどうしの演算

- 以下の演算

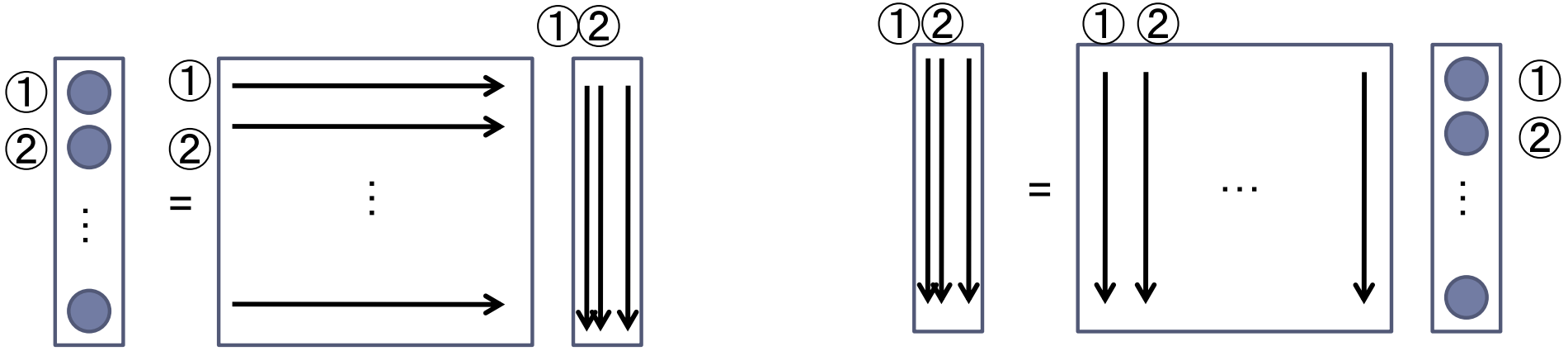
$$z = ax + y$$

  - ここで、αはスカラ、z、x、y　はベクトル
- どのようなデータ分散方式でも並列処理が可能
  - ただし、スカラ α は全PEで所有する。
  - ベクトルはO(n)のメモリ領域が必要なのに対し、スカラはO(1)のメモリ領域で大丈夫。
    →スカラメモリ領域は無視可能
  - 計算量：O(N/P)
  - あまり面白くない



行列とベクトルの積

- ＜行方式＞と＜列方式＞がある。
  - ＜データ分散方式＞と＜方式＞組のみ合わせがあり、少し面白い




```
for(i=0;i<n;i++){
     y[i]=0.0;
     for(j=0;j<n;j++){
          y[i] += a[i][j]*x[j];
     }
}
```

＜行方式＞： 自然な実装
C言語向き

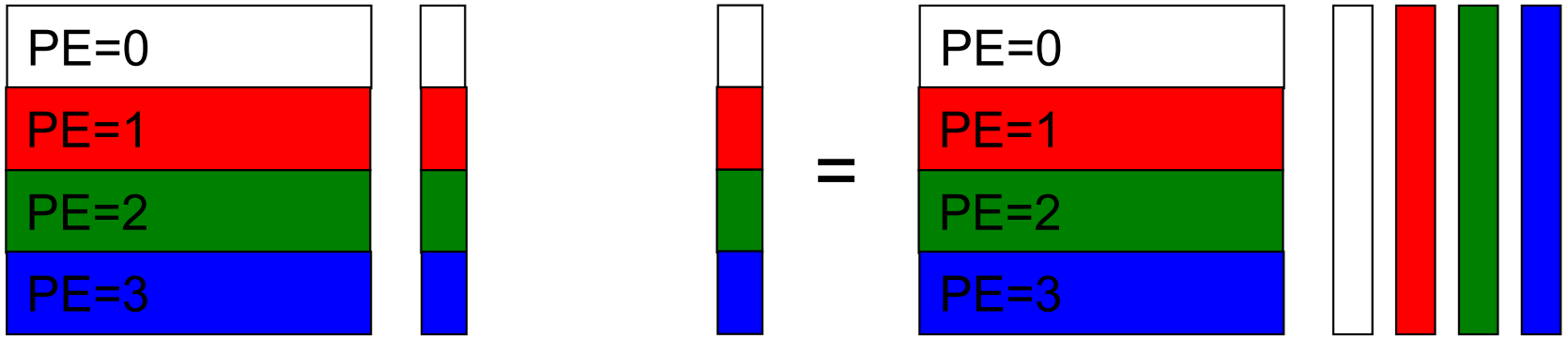
```
for(j=0; j<n; j++) y[j]=0.0;
for(j=0; j<n; j++) {
   for (i=0; i<n; i++) {
      y[i] += a[i][j]*x[j];
   }
}
```

＜列方式＞： Fortran言語向き

行列とベクトルの積

＜行方式の場合＞

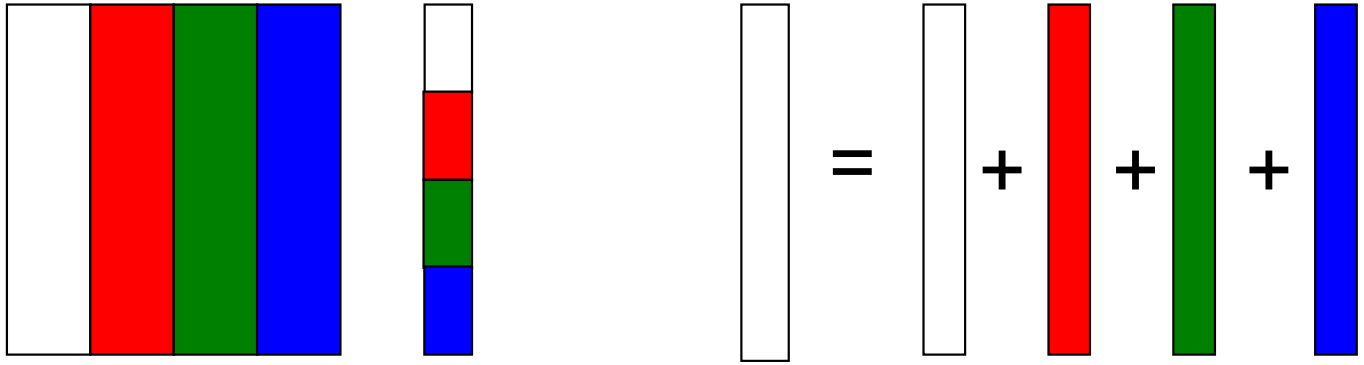
＜行方向分散方式＞　:行方式に向く分散方式



右辺ベクトルを MPI_Allgather関数を利用し、全PEで所有する

各PE内で行列ベクトル積を行う

＜列方向分散方式＞　:ベクトルの要素すべてがほしいときに向く



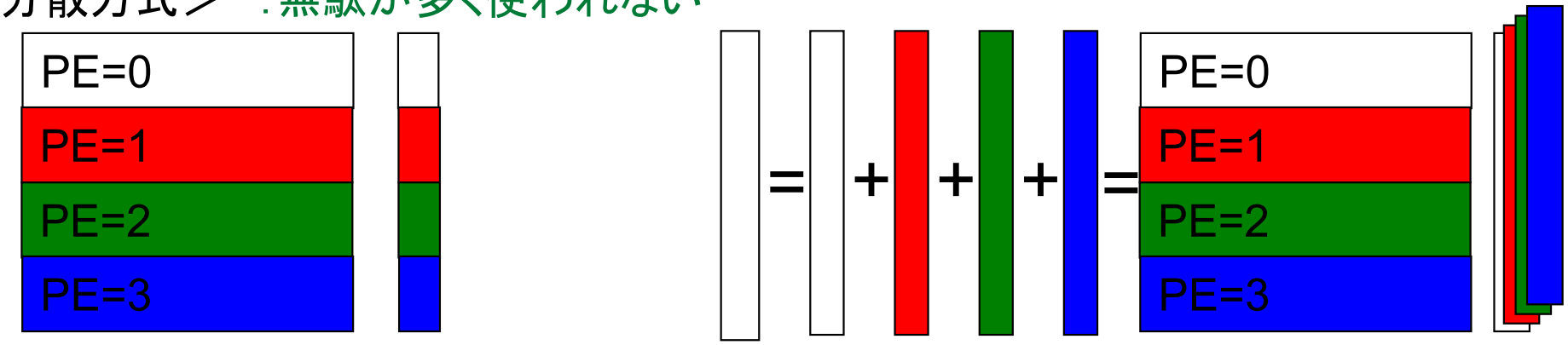
各PE内で行列-ベクトル積を行う

MPI_Reduce関数で総和を求める
（※ある1PEにベクトルすべてが集まる）

行列とベクトルの積

＜列方式の場合＞

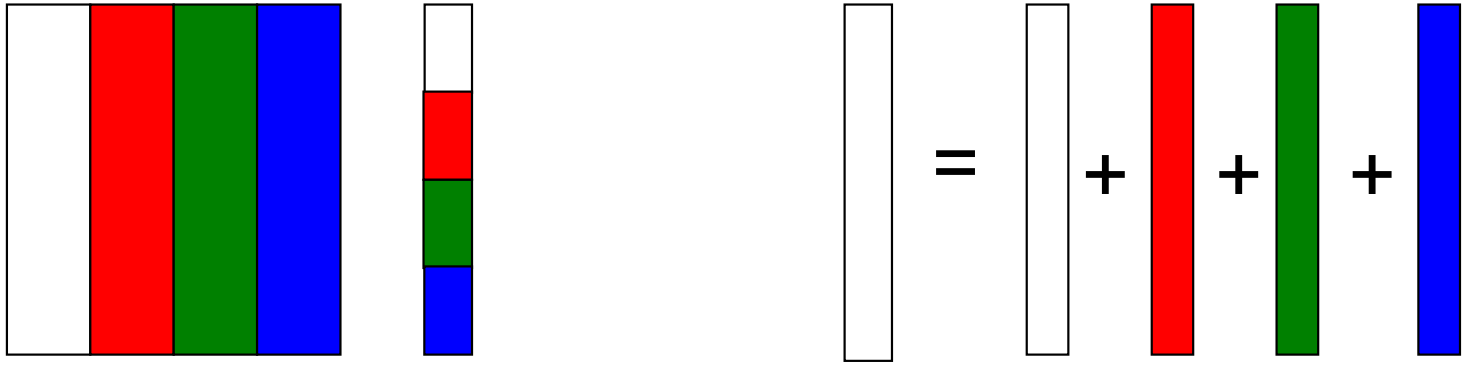
＜行方向分散方式＞　：無駄が多く使われない



右辺ベクトルを MPI_Allgather関数を利用して、全PEで所有する

結果をMPI_Reduce関数により総和を求める

＜列方向分散方式＞　：列方式に向く分散方式



各PE内で行列-ベクトル積を行う

MPI_Reduce関数で総和を求める
（※ある1PEにベクトルすべてが集まる）

基本的なMPI関数

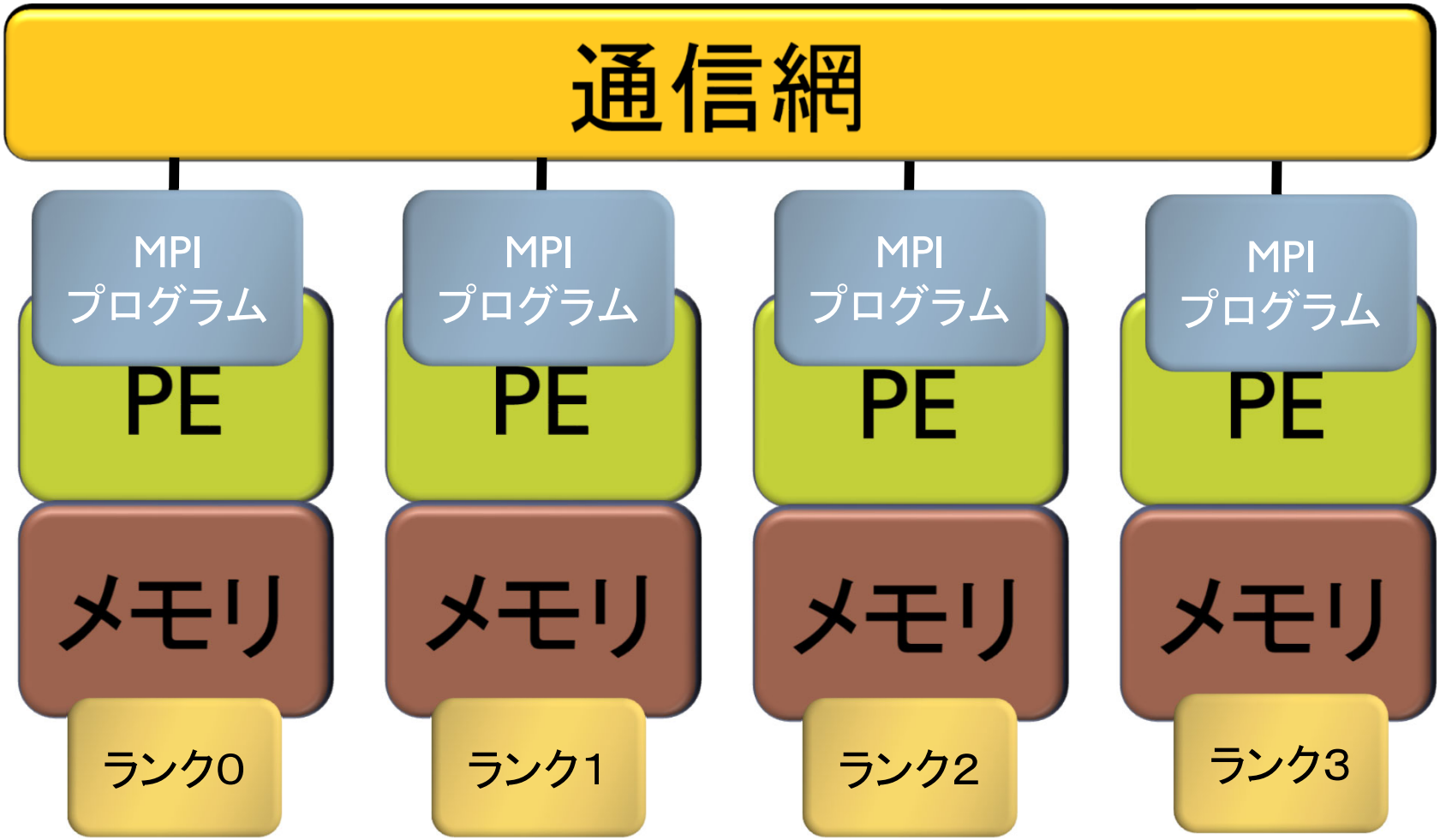
送信、受信のためのインタフェース

略語とMPI用語

- MPIは「プロセス」間の通信を行います。
- プロセスは、HT（ハイパースレッド）などを使わなければ、「プロセッサ」（もしくは、コア）に1対1で割り当てられます。
- 今後、「MPIプロセス」と書くのは長いので、ここでは PE（Processer Elementsの略）と書きます。
  - ただし用語として「PE」は、現在あまり使われていません。
- ランク（Rank）
  - 各「MPIプロセス」の「識別番号」のこと。
  - 通常MPIでは、MPI_Comm_rank関数で設定される変数（サンプルプログラムではmyid）に、0～全PE数－1　の数値が入る
  - 世の中の全MPIプロセス数を知るには、MPI_Comm_size関数を使う。

（サンプルプログラムでは、numprocs に、この数値が入る）

ランクの説明図


通信網
MPI
プログラム
MPI
プログラム
MPI
プログラム
MPI
プログラム
PE
PE
PE
PE
メモリ
メモリ
メモリ
メモリ
ランク0
ランク1
ランク2
ランク3

C言語インターフェースと Fortranインターフェースの違い

- C版は、 整数変数ierr が戻り値

  ierr = MPI_Xxxx(….);

- Fortran版は、最後に整数変数ierrが引数

  call MPI_XXXX(…., ierr)

- システム用配列の確保の仕方
  - C言語

    MPI_Status istatus;
  - Fortran言語

    integer istatus(MPI_STATUS_SIZE)

C言語インターフェースと Fortranインターフェースの違い

- MPIにおける、データ型の指定
  - C言語

MPI_CHAR (文字型) 、 MPI_INT (整数型)、
MPI_FLOAT (実数型)、MPI_DOUBLE(倍精度実数型)

  - Fortran言語

MPI_CHARACTER (文字型) 、MPI_INTEGER (整数型)、
MPI_REAL (実数型)、MPI_DOUBLE_PRECISION(倍精度実数型) 、MPI_COMPLEX(複素数型)

- 以降は、C言語インタフェースで説明する

基礎的なMPI関数—MPI_Recv（１／２）

- ierr = MPI_Recv(recvbuf, icount, idatatype, isource, itag, icomm, **istatus**);

  - recvbuf : 受信領域の先頭番地を指定する。
  - icount : 整数型。受信領域のデータ要素数を指定する。
  - idatatype : 整数型。受信領域のデータの型を指定する。
    - MPI_CHAR (文字型) 、MPI_INT (整数型)、MPI_FLOAT (実数型)、 MPI_DOUBLE(倍精度実数型)
  - isource : 整数型。受信したいメッセージを送信するPEのランクを指定する。
    - 任意のPEから受信したいときは、MPI_ANY_SOURCE を指定する。

基礎的なMPI関数—MPI_Recv（２／２）

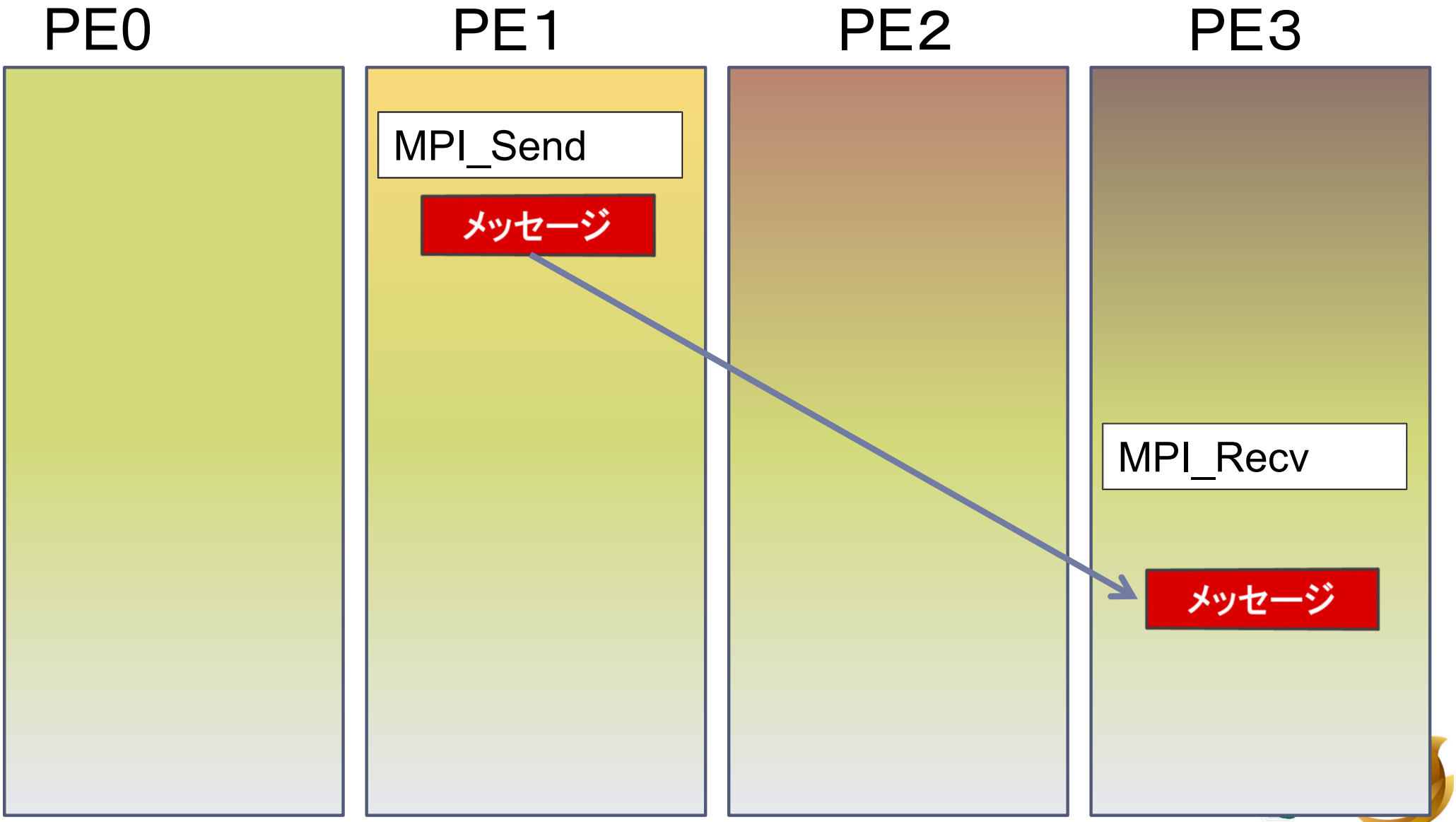
- itag： 整数型。受信したいメッセージに付いているタグの値を指定。
  - 任意のタグ値のメッセージを受信したいときは、MPI_ANY_TAG を指定。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定。
  - 通常ではMPI_COMM_WORLD を指定すればよい。
- istatus： MPI_Status型（整数型の配列）。受信状況に関する情報が入る。かならず専用の型宣言をした配列を確保すること。
  - 要素数がMPI_STATUS_SIZEの整数配列が宣言される。
  - 受信したメッセージの送信元のランクが istatus[MPI_SOURCE]、タグが istatus[MPI_TAG] に代入される。
  - **C言語: MPI_Status istatus;**
  - **Fortran言語: integer istatus(MPI_STATUS_SIZE)**
- ierr(戻り値)： 整数型。エラーコードが入る。

基礎的なMPI関数—MPI_Send

```
ierr = MPI_Send(sendbuf, icount, idatatype, idest,
       itag, icomm);
```

- sendbuf : 送信領域の先頭番地を指定
- icount : 整数型。送信領域のデータ要素数を指定
- idatatype : 整数型。送信領域のデータの型を指定
- idest : 整数型。送信したいPEのicomm内でのランクを指定
- itag : 整数型。受信したいメッセージに付けられたタグの値を指定
- icomm : 整数型。プロセッサー集団を認識する番号である コミュニケータを指定
- ierr (戻り値) : 整数型。エラーコードが入る。

Send－Recvの概念（１対１通信）

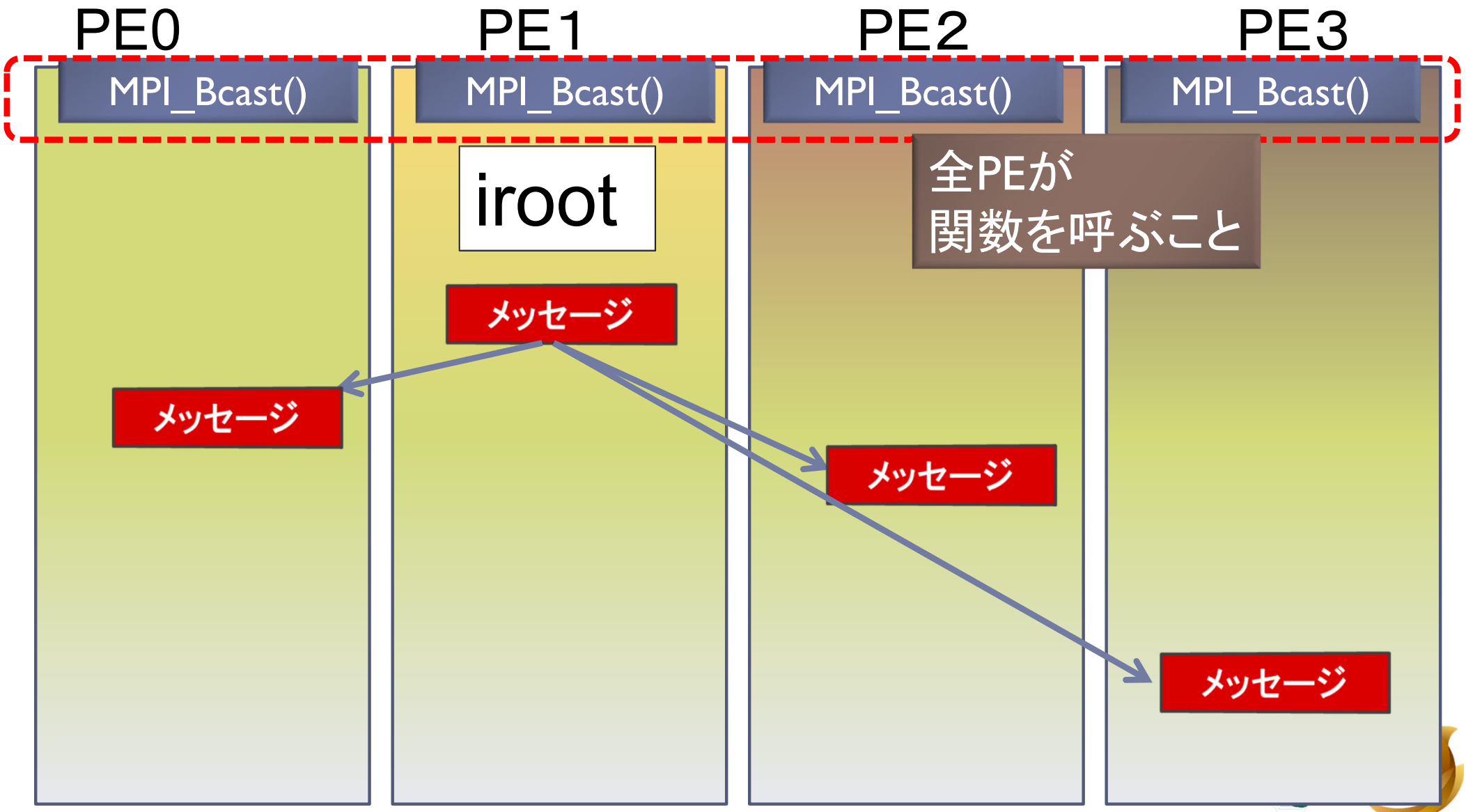


基礎的なMPI関数—MPI_Bcast

```
ierr = MPI_Bcast(sendbuf, icount, idatatype,
       iroot, icomm);
```

- sendbuf： 送信および受信領域の先頭番地を指定する。
- icount： 整数型。送信領域のデータ要素数を指定する。
- idatatype： 整数型。送信領域のデータの型を指定する。
- iroot： 整数型。送信したいメッセージがあるPEの番号を指定する。全PEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr (戻り値)： 整数型。エラーコードが入る。

MPI_Bcastの概念（集団通信）



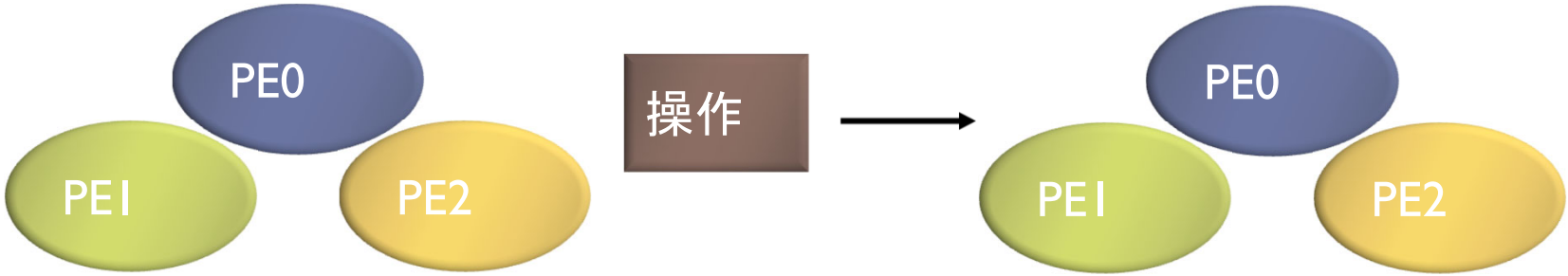
リダクション演算

- ＜操作＞によって＜次元＞を減少（リダクション）させる処理
  - 例：　内積演算
    ベクトル（n次元空間）　→　スカラ（1次元空間）
- リダクション演算は、通信と計算を必要とする
  - 集団通信演算（collective communication operation）と呼ばれる
- 演算結果の持ち方の違いで、2種のインタフェースが存在する

リダクション演算

- 演算結果に対する所有PEの違い
  - MPI_Reduce関数
    - リダクション演算の結果を、ある一つのPEに所有させる

  - MPI_Allreduce関数
    - リダクション演算の結果を、全てのPEに所有させる



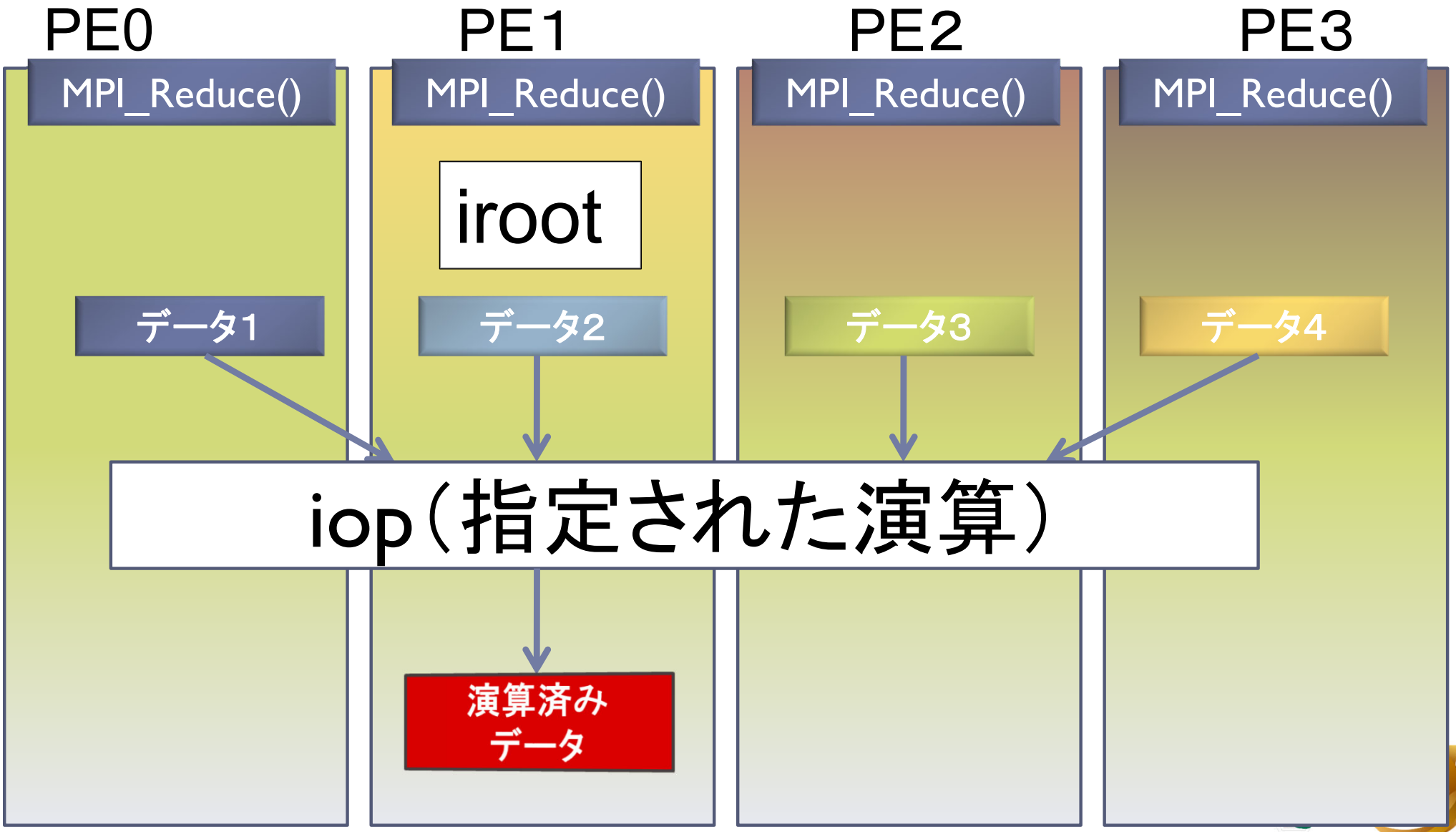
基礎的なMPI関数—MPI_Reduce

- ierr = MPI_Reduce(sendbuf, recvbuf, icount, idatatype, iop, iroot, icomm);
  - sendbuf : 送信領域の先頭番地を指定する。
  - recvbuf : 受信領域の先頭番地を指定する。irootで指定したPEのみで書き込みがなされる。
    送信領域と受信領域は、同一であってはならない。
    すなわち、異なる配列を確保しなくてはならない。
  - icount : 整数型。送信領域のデータ要素数を指定する。
  - idatatype : 整数型。送信領域のデータの型を指定する。
    - (Fortran)＜最小／最大値と位置＞を返す演算を指定する場合は、MPI_2INTEGER(整数型)、 MPI_2REAL (単精度型)、MPI_2DOUBLE_PRECISION(倍精度型) 、を指定する。

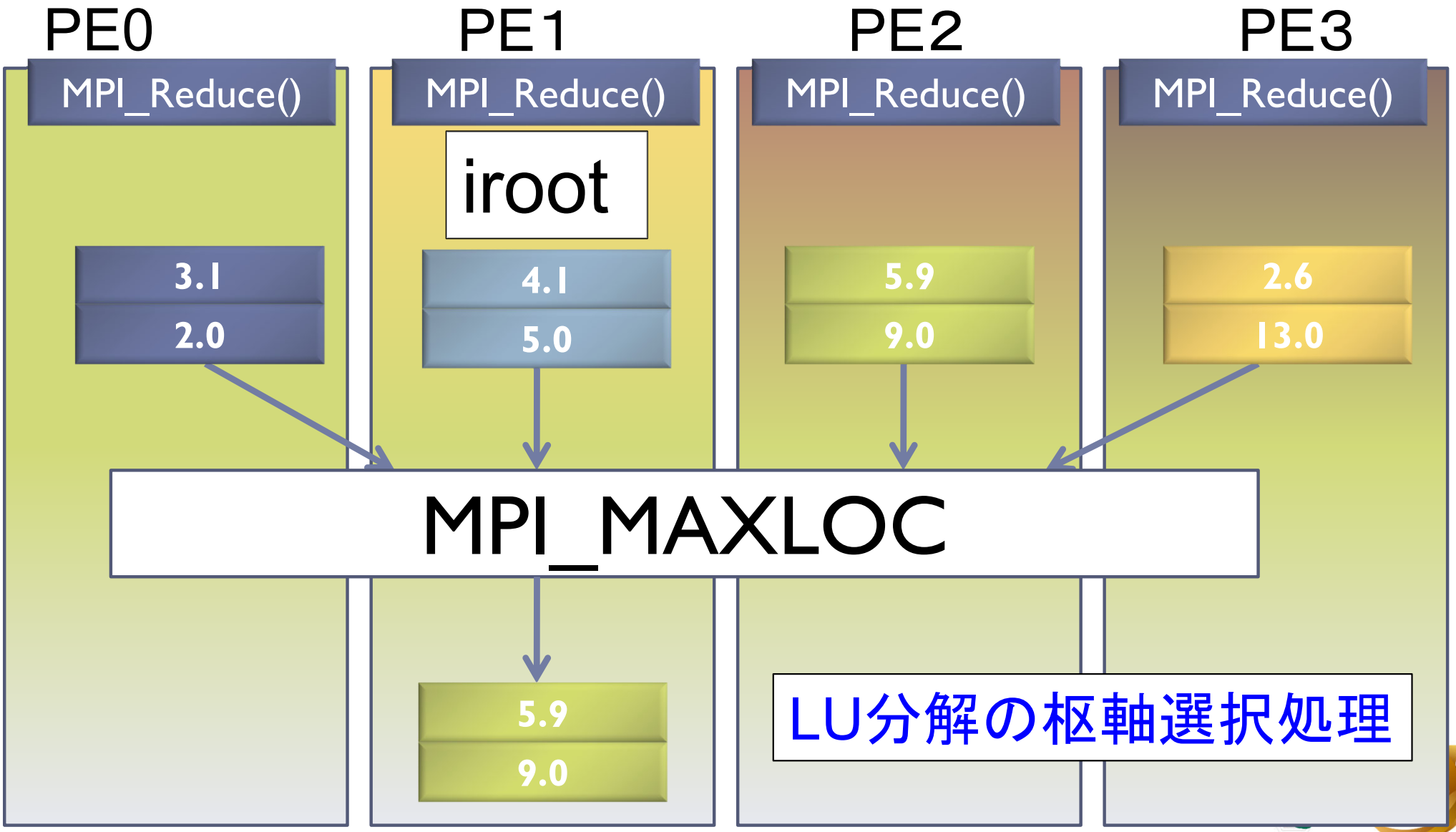
基礎的なMPI関数—MPI_Reduce

- iop： 整数型。演算の種類を指定する。
  - MPI_SUM (総和)、MPI_PROD (積)、MPI_MAX (最大)、MPI_MIN (最小)、 MPI_MAXLOC (最大と位置)、MPI_MINLOC (最小と位置) など。
- iroot： 整数型。結果を受け取るPEのicomm 内でのランクを指定する。全てのicomm 内のPEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr : 整数型。 エラーコードが入る。

MPI_Reduceの概念（集団通信）



MPI_Reduceによる２リスト処理例 (MPI_2DOUBLE_PRECISION と MPI_MAXLOC)



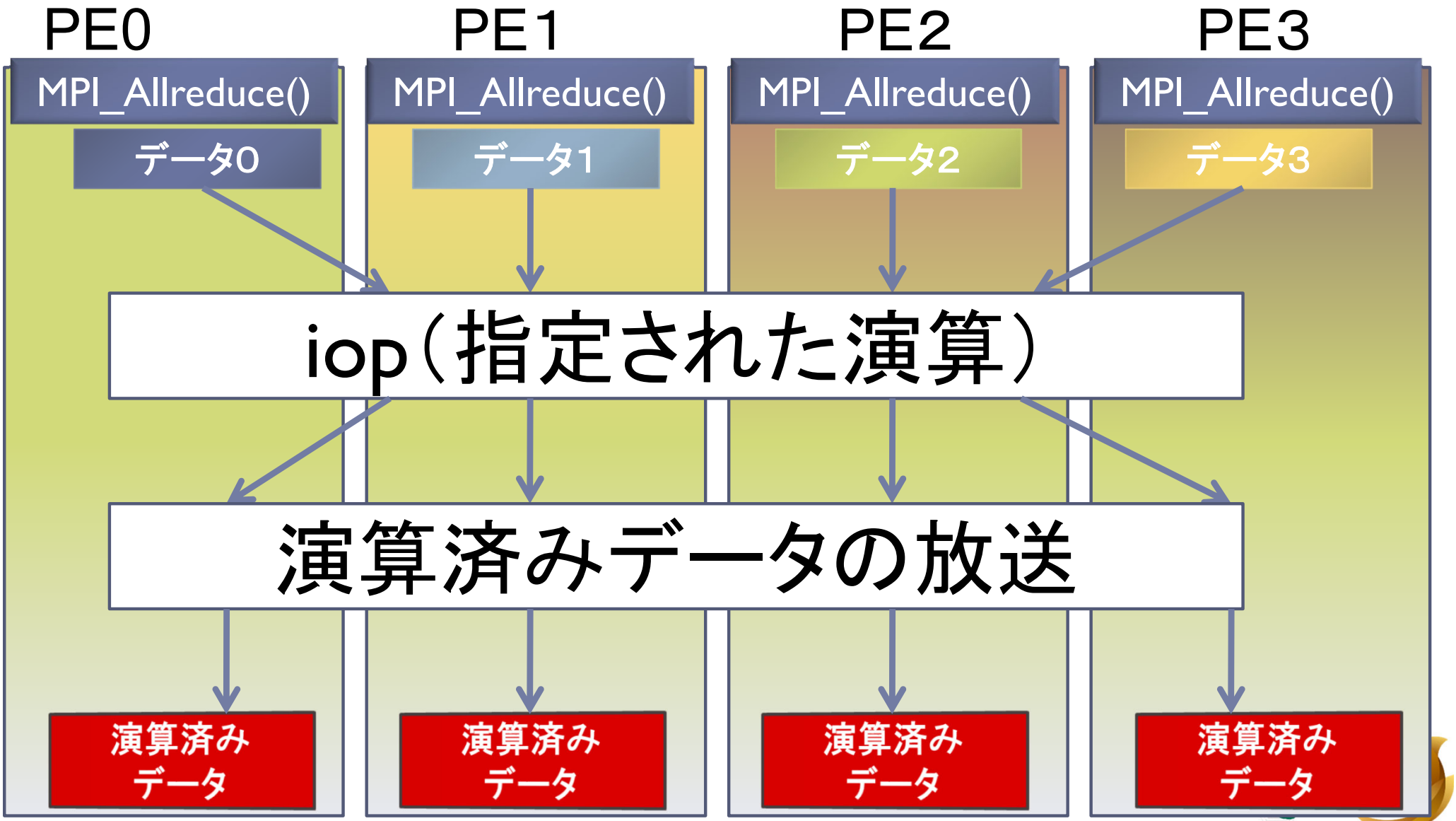
基礎的なMPI関数—MPI_Allreduce

- ierr = MPI_Allreduce(sendbuf, recvbuf, icount, idatatype, iop, icomm);
  - sendbuf : 送信領域の先頭番地を指定する。
  - recvbuf : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
    送信領域と受信領域は、同一であってはならない。
    すなわち、異なる配列を確保しなくてはならない。
  - icount : 整数型。送信領域のデータ要素数を指定する。
  - idatatype : 整数型。送信領域のデータの型を指定する。
    - 最小値や最大値と位置を返す演算を指定する場合は、MPI_2INT(整数型)、MPI_2FLOAT (単精度型)、MPI_2DOUBLE(倍精度型) を指定する。

基礎的なMPI関数—MPI_Allreduce

- iop： 整数型。演算の種類を指定する。
  - MPI_SUM (総和)、MPI_PROD (積)、MPI_MAX (最大)、MPI_MIN (最小)、 MPI_MAXLOC (最大と位置)、MPI_MINLOC (最小と位置) など。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr : 整数型。 エラーコードが入る。

MPI_Allreduceの概念（集団通信）

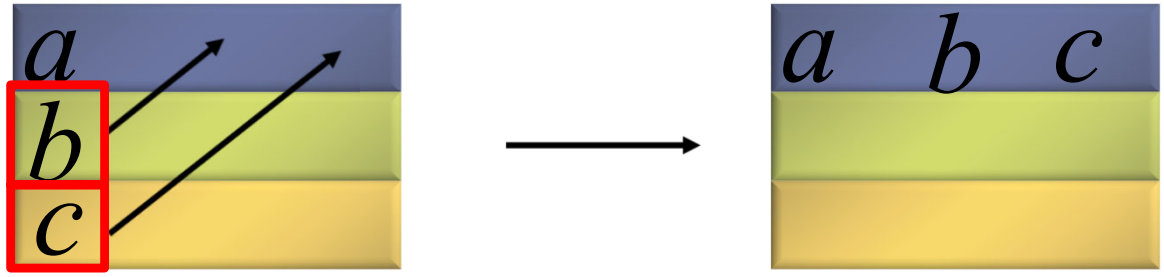


リダクション演算

- 性能について
  - リダクション演算は、1対1通信に比べ遅い
    - プログラム中で多用すべきでない！
  - MPI_Allreduce は MPI_Reduce に比べ遅い
    - MPI_Allreduce は、放送処理が入る。
    - なるべく、MPI_Reduce を使う。

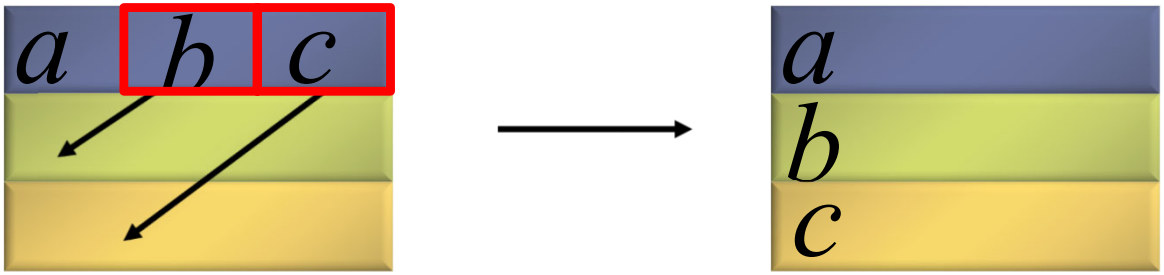
行列の転置

- 行列 A が(Block, *)分散されているとする。
- 行列 A の転置行列 A^T を作るには、MPIでは次の2通りの関数を用いる
  - MPI_Gather関数



集めるメッセージサイズが各PEで均一のとき使う

  - MPI_Scatter関数



集めるサイズが各PEで均一でないときは:
MPI_GatherV関数
MPI_ScatterV関数

基礎的なMPI関数—MPI_Gather

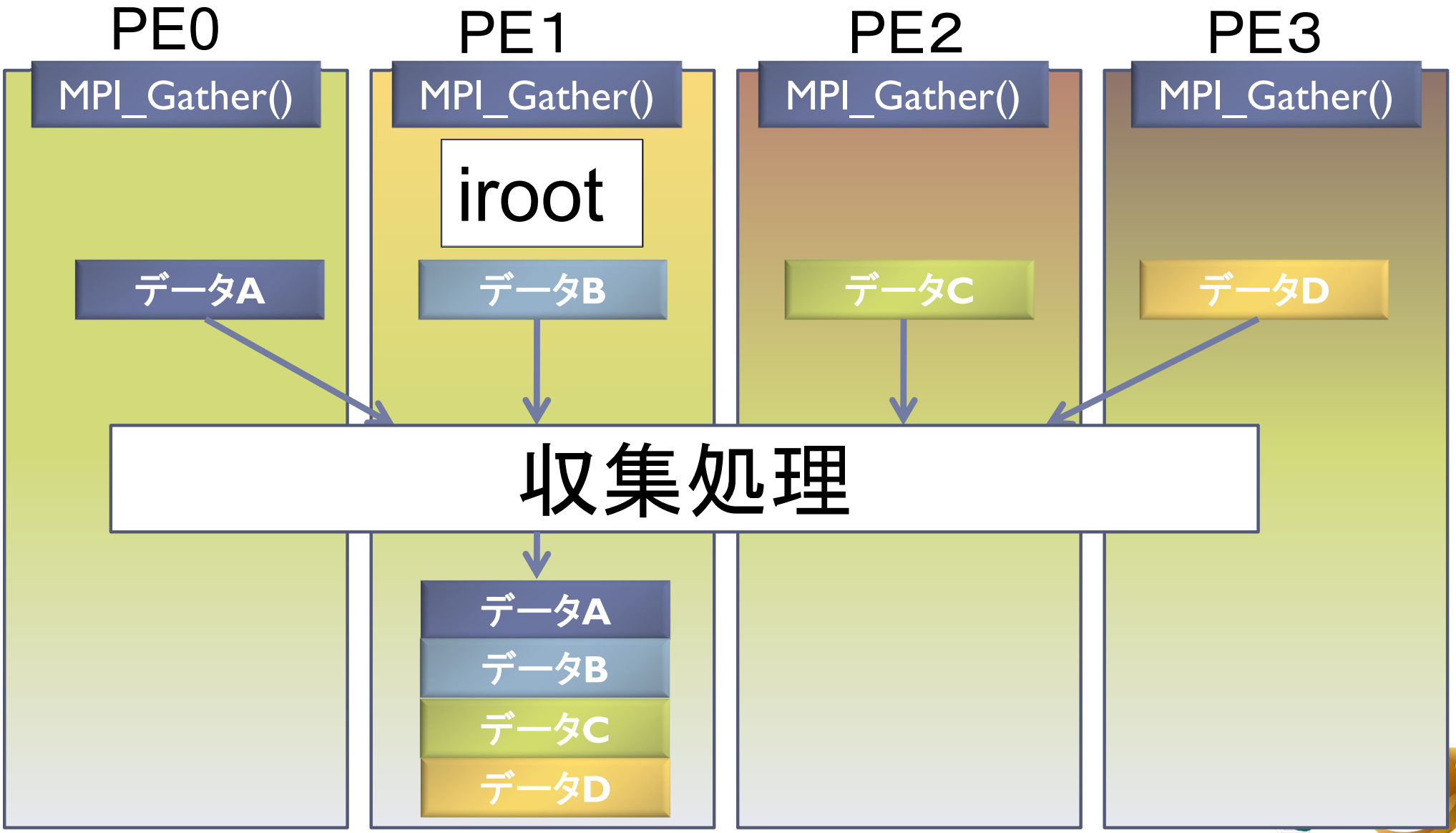
```
ierr = MPI_Gather (sendbuf, isendcount, isendtype,
                   recvbuf, irecvcount, irecvtype, iroot, icomm);
```

- sendbuf : 送信領域の先頭番地を指定する。
- isendcount: 整数型。送信領域のデータ要素数を指定する。
- isendtype : 整数型。送信領域のデータの型を指定する。
- recvbuf : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
  - なお原則として、送信領域と受信領域は、同一であってはならない。すなわち、異なる配列を確保しなくてはならない。
- irecvcount: 整数型。受信領域のデータ要素数を指定する。
  - この要素数は、1PE当たりの送信データ数を指定すること。
  - MPI_Gather 関数では各PEで異なる数のデータを収集することはできないので、同じ値を指定すること。

基礎的なMPI関数—MPI_Gather

- irecvtype： 整数型。受信領域のデータ型を指定する。
- iroot： 整数型。収集データを受け取るPEのicomm 内でのランクを指定する。
  - 全てのicomm 内のPEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr： 整数型。エラーコードが入る。

MPI_Gatherの概念（集団通信）



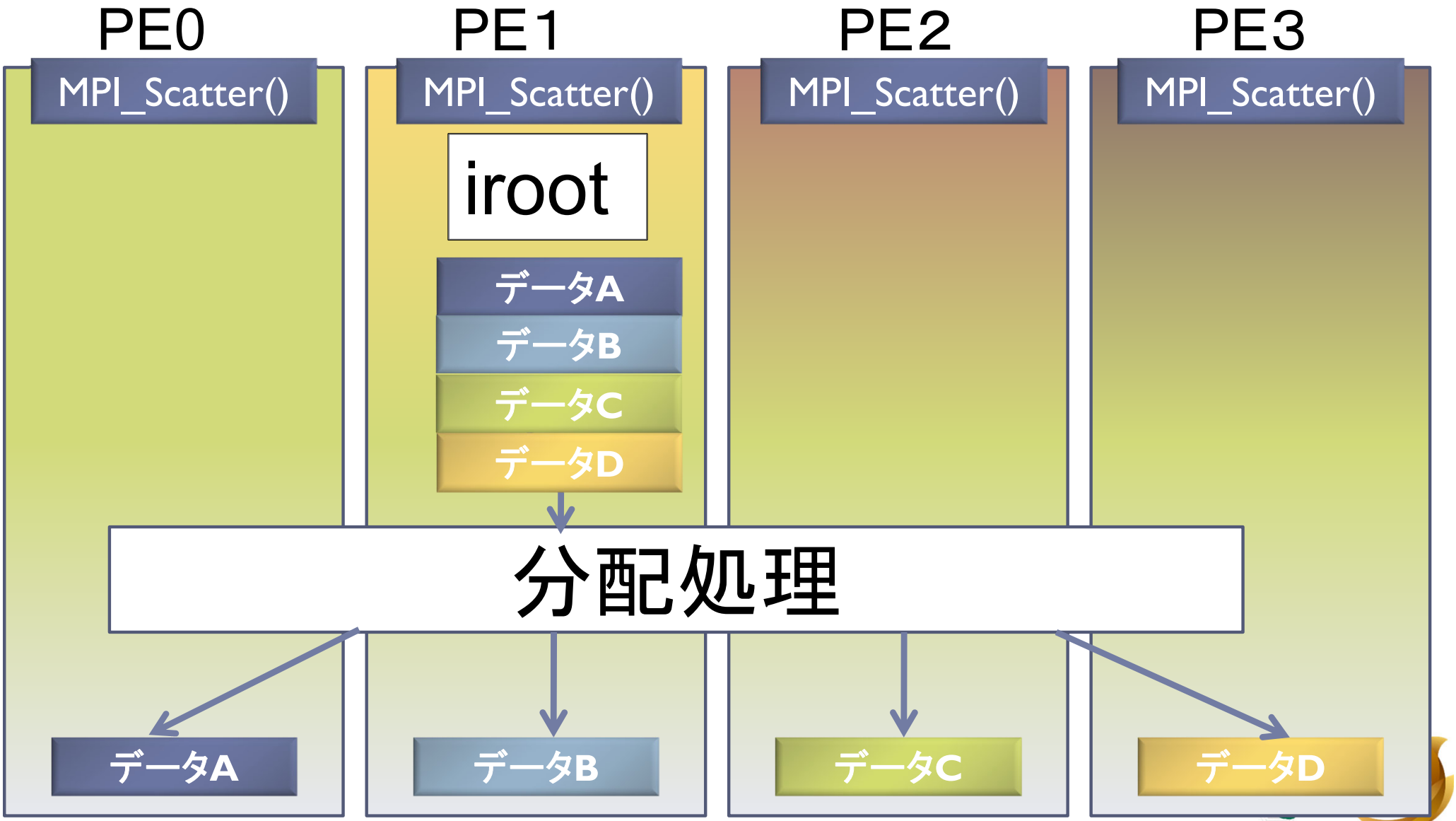
基礎的なMPI関数—MPI_Scatter

- ierr = MPI_Scatter (sendbuf, isendcount, isendtype, recvbuf, irecvcount, irecvtype, iroot, icomm);
  - sendbuf : 送信領域の先頭番地を指定する。
  - isendcount: 整数型。送信領域のデータ要素数を指定する。
    - この要素数は、1PE当たりに送られる送信データ数を指定すること。
    - MPI_Scatter 関数では各PEで異なる数のデータを分散することはできないので、同じ値を指定すること 。
  - isendtype : 整数型。送信領域のデータの型を指定する。 iroot で指定したPEのみ有効となる。
  - recvbuf : 受信領域の先頭番地を指定する。
    - なお原則として、送信領域と受信領域は、同一であってはならない。すなわち、異なる配列を確保しなくてはならない。
  - irecvcount: 整数型。受信領域のデータ要素数を指定する。

基礎的なMPI関数—MPI_Scatter

- irecvtype： 整数型。受信領域のデータ型を指定する。
- iroot： 整数型。収集データを受け取るPEのicomm 内でのランクを指定する。
  - 全てのicomm 内のPEで同じ値を指定する必要がある。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ierr： 整数型。エラーコードが入る。

MPI_Scatterの概念（集団通信）



参考文献

1. MPI並列プログラミング、P.パチェコ 著 ／ 秋葉 博 訳
2. 並列プログラミング虎の巻MPI版、青山幸也 著、
   高度情報科学技術研究機構（RIST）神戸センター
   （ http://www.hpci-office.jp/pages/seminar_text ）
3. Message Passing Interface Forum
   （ http://www.mpi-forum.org/ ）
4. 並列コンピュータ工学、富田眞治著、昭晃堂（1996）

MPIプログラミング実習（演習）

講義の流れ

1. 行列-行列とは（30分）
2. 行列-行列積のサンプルプログラムの実行
3. サンプルプログラムの説明
4. 演習課題（1）：簡単なもの
5. 演習課題（2）：ちょっと難しいもの（中級）

行列 - 行列積の演習の流れ

- 演習課題（Ⅱ）
  - 簡単なもの（30分程度で並列化）
  - 通信関数が一切不要
- 演習課題（Ⅲ）
  - ちょっと難しい（1時間以上で並列化）
  - 1対1通信関数が必要
  - 演習課題（Ⅰ）が早く終わってしまった方は、やってみてください。

行列-行列積とは

実装により性能向上が見込める基本演算

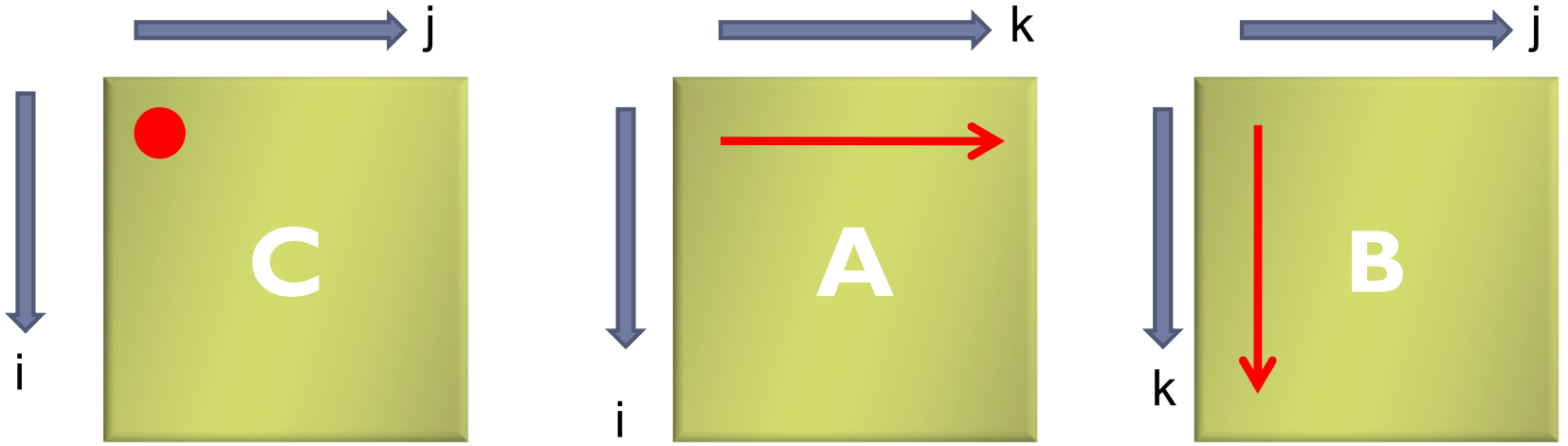
１.５　行列の積

- 行列積　Ｃ＝Ａ・Ｂは、コンパイラや計算機のベンチマークに使われることが多い
  - 理由１：実装方式の違いで性能に大きな差がでる
  - 理由２：手ごろな問題である（プログラムし易い）
  - 理由３：科学技術計算の特徴がよく出ている
    1. 非常に長い＜連続アクセス＞がある
    2. キャッシュに乗り切らない＜大規模なデータ＞に対する演算である
    3. メモリバンド幅を食う演算（メモリ・インテンシブ）な処理である

行列積コード例（C言語）

- コード例

```
for (i=0; i<n;  i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      C[i][j] += A[i][k] *B[k][j];
```



1.5　行列の積

- 行列積　$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \ (i, j = 1, 2, \ldots, n)$

  の実装法は、次の二通りが知られている:

1. ループ交換法
   - 連続アクセスの方向を変える目的で、行列-行列積を実現する3重ループの順番を交換する
2. ブロック化（タイリング）法
   - キャッシュにあるデータを再利用する目的で、あるまとまった行列の部分データを、何度もアクセスするように実装する

1.5　行列の積

- ループ交換法
  - 行列積のコードは、以下のような3重ループになる（C言語）

```
for(i=0; i<n; i++) {
  for(j=0; j<n; j++) {
    for(k=0; k<n; k++) {
      c[ i ][ j ] = c[ i ][ j ] + a[ i ][ k ] * b[ k][ j ];
    }
  }
}
```

  - 最内部の演算は、外側の3ループを交換しても、計算結果が変わらない
    → 6通りの実現の方法がある

1.5　行列の積

- ループ交換法
  - 行列積のコードは、以下のような3重ループになる（Fortran言語）

```
do i=1, n
  do j=1, n
    do k=1, n
      c( i , j ) = c( i, j) + a( i , k ) * b( k , j )
    enddo
  enddo
enddo
```

  - 最内部の演算は、外側の3ループを交換しても、計算結果が変わらない
    → 6通りの実現の方法がある

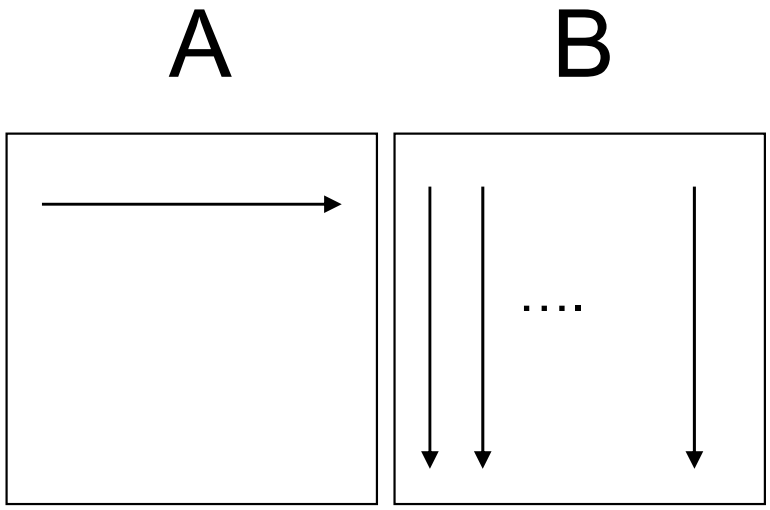
1.5　行列の積

- 行列データへのアクセスパターンから、以下の3種類に分類できる
  1. 内積形式 (inner-product form)
     最内ループのアクセスパタンが
     ＜ベクトルの内積＞と同等
  2. 外積形式 (outer-product form)
     最内ループのアクセスパタンが
     ＜ベクトルの外積＞と同等
  3. 中間積形式 (middle-product form)
     内積と外積の中間

1.5　行列の積

- 内積形式 (inner-product form)
  - ijk, jikループによる実現(C言語)

```
for (i=0; i<n; i++) {
   for (j=0; j<n; j++) {
      dc = 0.0;
      for (k=0; k<n; k++){
         dc = dc + A[ i ][ k ] * B[ k ][ j ];
      }
      C[ i ][ j ]= dc;
   }
}
```



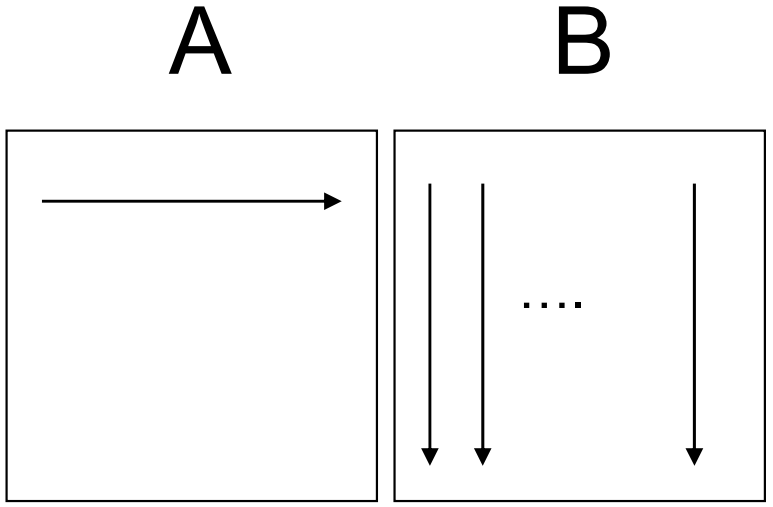
●行方向と列方向のアクセスあり
→行方向・列方向格納言語の
両方で性能低下要因
解決法:
A, Bどちらか一方を転置しておく

※以降、最外のループからの変数の順番で実装法を呼ぶ。たとえば上記のコードは<ijkループ>。

1.5　行列の積

- 内積形式 (inner-product form)
  - ijk, jikループによる実現(Fortran言語)

```
do i=1, n
   do j=1, n
      dc = 0.0d0
      do k=1, n
         dc = dc + A( i , k ) * B( k , j )
      enddo
      C( i , j ) = dc
   enddo
enddo
```



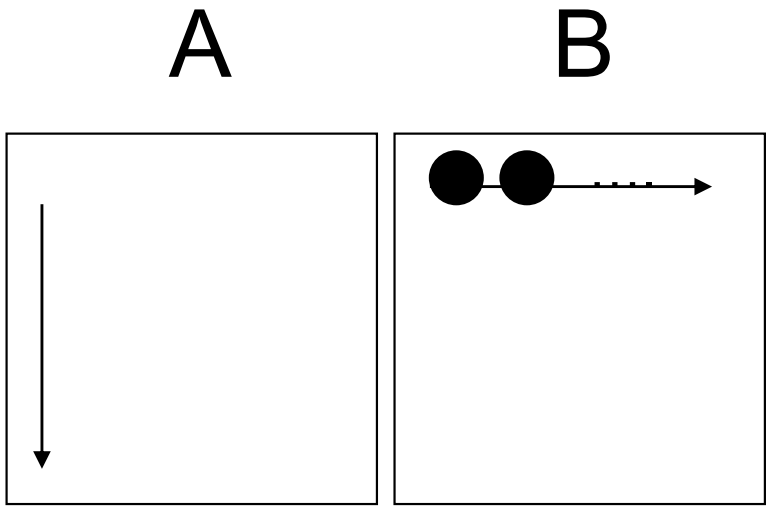
●行方向と列方向のアクセスあり
→行方向・列方向格納言語の
両方で性能低下要因
解決法:
A, Bどちらか一方を転置しておく

※以降、最外のループからの変数の順番で実装法を呼ぶ。たとえば上記のコードは＜ijkループ＞。

1.5　行列の積

- 外積形式 (outer-product form)
  - kij, kjiループによる実現（C言語）

```
for (i=0;  i<n;  i++) {
   for (j=0;  j<n;  j++) {
      C[ i ][ j ] = 0.0;
   }
}
for (k=0;  k<n;  k++) {
  for (j=0;  j<n;  j++) {
     db = B[ k ][ j ];
     for (i=0;  i<n;  i++) {
      C[ i ][ j ]= C[ i ][ j ]+ A[ i ][ k ]* db;
     }
  }
}
```

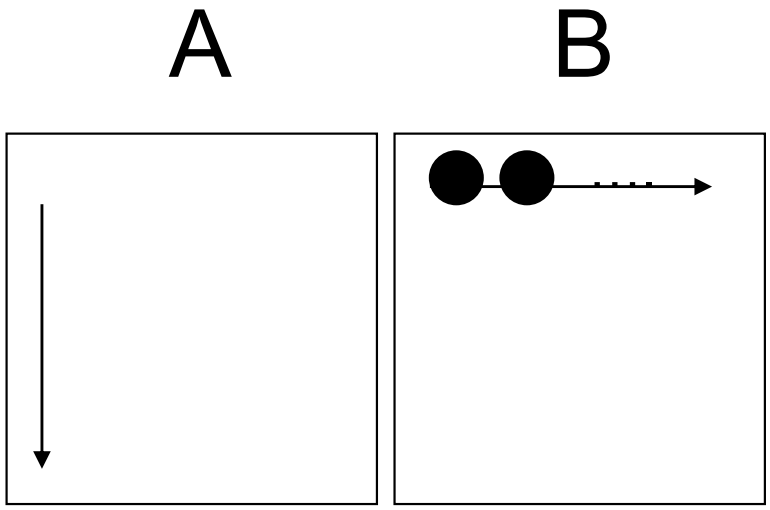


●kjiループでは
列方向アクセスがメイン
→列方向格納言語向き
（Fortran言語）

1.5　行列の積

- 外積形式 (outer-product form)
  - kij, kjiループによる実現（Fortran言語）

```
do i=1, n
  do j=1, n
    C( i , j ) = 0.0d0
  enddo
enddo
do k=1, n
  do j=1, n
    db = B( k , j )
    do i=1, n
      C( i , j ) = C( i , j )+ A( i , k ) * db
    enddo
  enddo
enddo
```

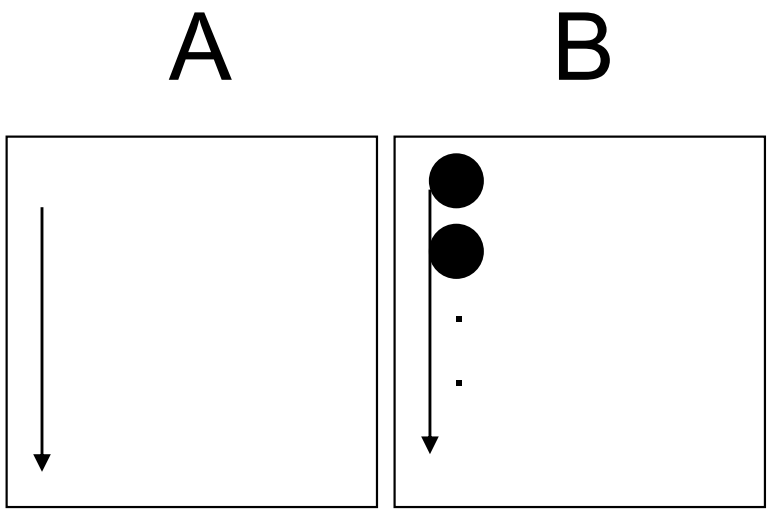


●kjiループでは
列方向アクセスがメイン
→列方向格納言語向き
（Fortran言語）

1.5　行列の積

- 中間積形式 (middle-product form)
  - ikj, jkiループによる実現(C言語)

```
for (j=0;  j<n;  j++)  {
   for (i=0;  i<n;  i++) {
      C[ i ][ j ] = 0.0;
   }
   for (k=0;  k<n;  k++) {
     db = B[ k ][ j ];
     for (i=0;  i<n;  i++) {
      C[ i ][ j ] = C[ i ][ j ] + A[ i ][ k ] * db;
     }
   }
}
```

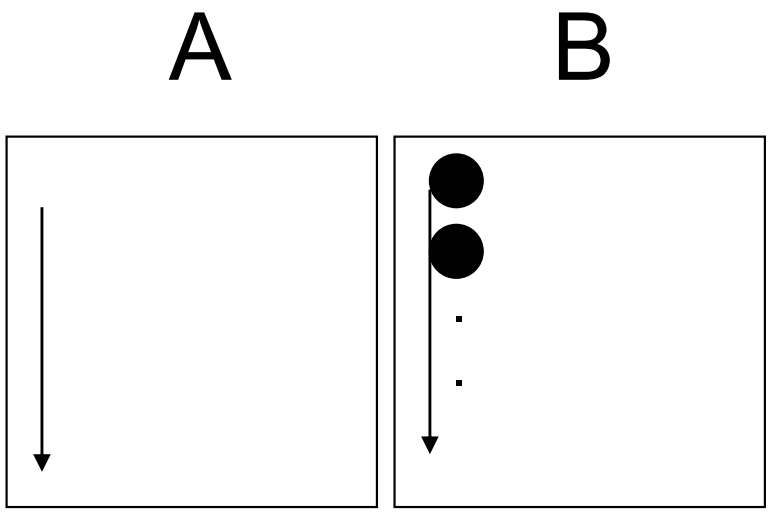


●jkiループでは
全て列方向アクセス
→列方向格納言語に
最も向いている
(Fortran言語)

1.5　行列の積

- 中間積形式 (middle-product form)
  - ikj, jkiループによる実現(Fortran言語)

```
do j=1, n
  do i=1, n
    C( i , j ) = 0.0d0
  enddo
  do k=1, n
    db = B( k , j )
    do i=1, n
      C( i , j ) = C( i , j ) + A( i , k ) * db
    enddo
  enddo
enddo
```

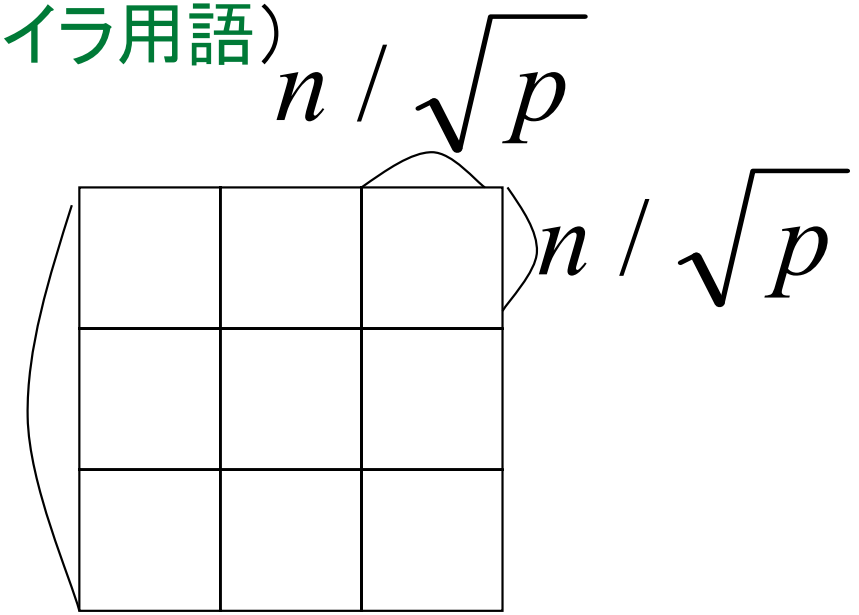


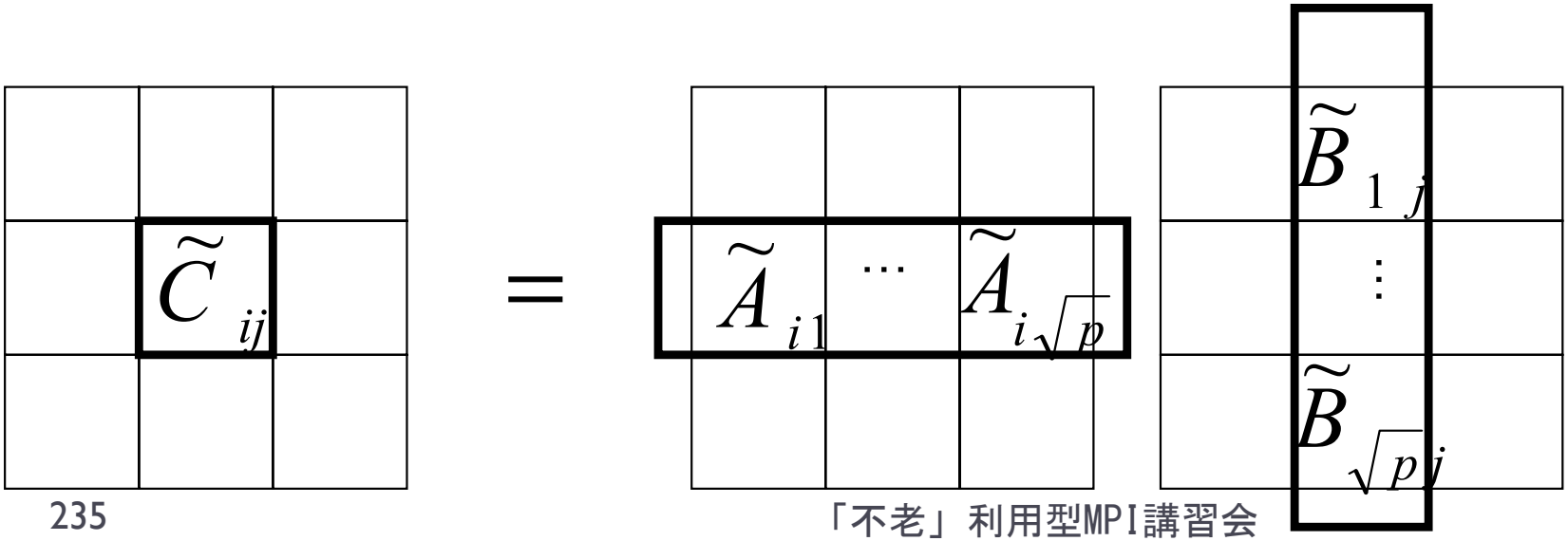
●jkiループでは
全て列方向アクセス
→列方向格納言語に
最も向いている
(Fortran言語)

1.5　行列の積

- 小行列ごとの計算に分けて（配列を用意し）計算
  （＜ブロック化＞, ＜タイリング＞:コンパイラ用語）
- 以下の計算

$$\tilde{C}_{ij} = \sum_{k=1}^{\sqrt{n}} \tilde{A}_{ik} \tilde{B}_{kj}$$





1.5　行列の積

- 各小行列をキャッシュに収まるサイズにする。
  1. ブロック単位で高速な演算が行える
  2. 並列アルゴリズムの変種が構築できる
- 並列行列積アルゴリズムは、データ通信の形態から、以下の2種に分類可能:
  1. セミ・シストリック方式
     - 行列A、Bの小行列の一部をデータ移動（Cannonのアルゴリズム）
  2. フル・シストリック方式
     - 行列A、Bの小行列のすべてをデータ移動（Foxのアルゴリズム）

サンプルプログラムの実行（行列-行列積）

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版の共通ファイル名

  Mat-Mat-flow-fx.tar

行列-行列積のサンプルプログラムの実行

- 以下のコマンドを実行する

```
$ cp /center/a49904a/Mat-Mat-flow-fx.tar ./
$ tar xvf Mat-Mat-flow-fx.tar
$ cd Mat-Mat
```

- 以下のどちらかを実行

```
$ cd C :C言語を使う人
$ cd F :Fortran言語を使う人
```

- 以下共通

```
$ make
$ pjsub mat-mat.bash
```

- 実行が終了したら、以下を実行する

```
$ cat mat-mat.bash.XXXXXX.out
```

行列-行列積のサンプルプログラムの実行（C言語）

- 以下のような結果が見えれば成功

N = 2000

Mat-Mat time = 0.324356 [sec.]

49328.561198 [MFLOPS]

OK!

1コアのみで、49.3GFLOPSの性能

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

NN = 2000

Mat-Mat time[sec.] = 0.3214948605746031

MFLOPS = 49767.51395343402

OK!

1コアのみで、49.7GFLOPSの性能

サンプルプログラムの説明

- #define N 2000
  の、数字を変更すると、行列サイズが変更できます
- #define DEBUG 1
  の「0」を「1」にすると、行列-行列積の演算結果が検証できます。
- MyMatMat関数の仕様
  - Double型N×N行列AとBの行列積をおこない、Double型N×N行列Cにその結果が入ります

Fortran言語のサンプルプログラムの注意

- 行列サイズNの宣言は、以下のファイルにあります。

  mat-mat.inc
- 行列サイズ変数が、NNとなっています。

  integer　NN

  parameter (NN=2000)

演習課題（１）

- MyMatMat関数を並列化してください。
  - #define　N　　576
  - #define　DEBUG　1

  として、デバッグをしてください。
- 行列A、B、Cは、各PEで重複して、かつ全部（N×N）所有してよいです。

演習課題（１）

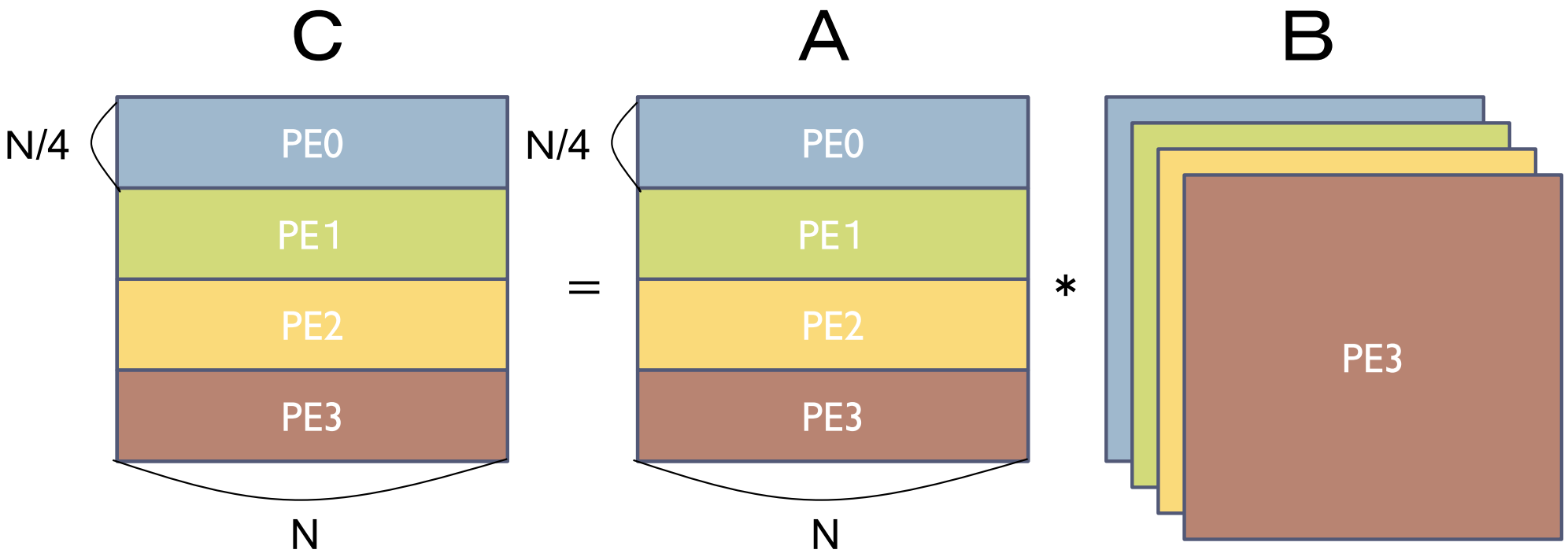
- サンプルプログラムでは、行列A、Bの要素を全部1として、行列-行列積の結果をもつ行列Cの全要素がNであるか調べ、結果検証しています。デバックに活用してください。
  - 行列Cの分散方式により、

演算結果チェックルーチンの並列化が必要

になります。注意してください。

並列化のヒント

- 以下のようなデータ分割にすると、とても簡単です。



- 通信関数は一切不要です。

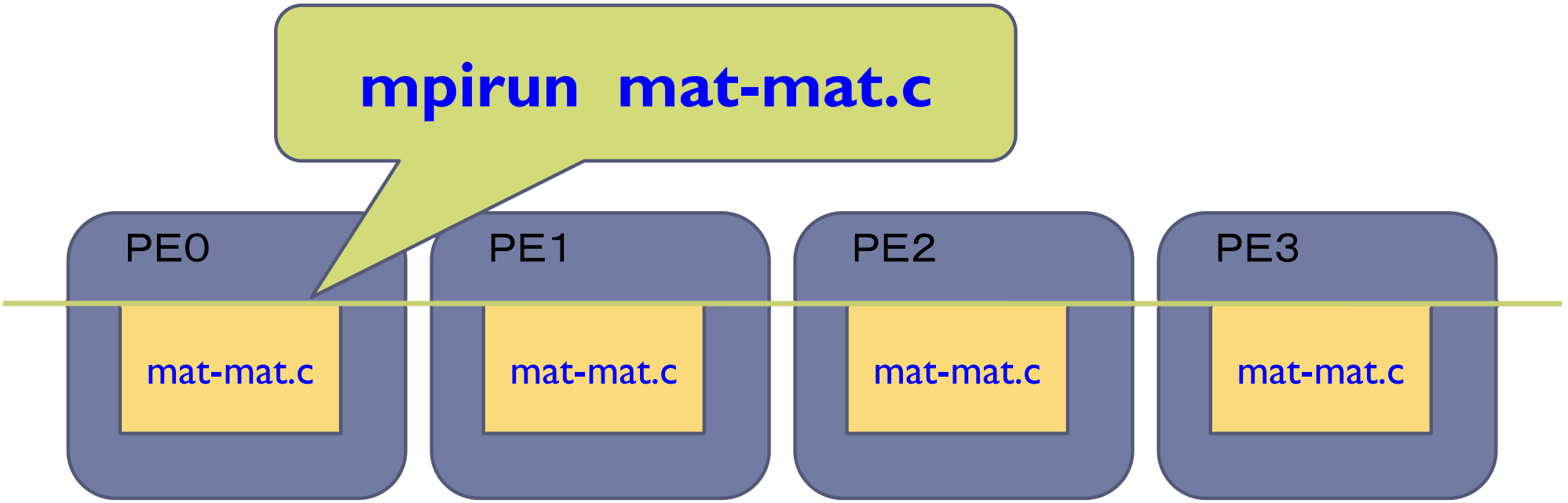
MPI並列化の大前提（再確認）

- SPMD
  - 対象のプログラム（mat-mat.c, mat-mat.f）は、
    - すべてのPEで、かつ、
    - 同時に起動された状態

    から処理が始まる。
- 分散メモリ型並列計算機
  - 各PEは、完全に独立したメモリを持っている。他のPEからは、通信なしには参照できない。（共有メモリではない）

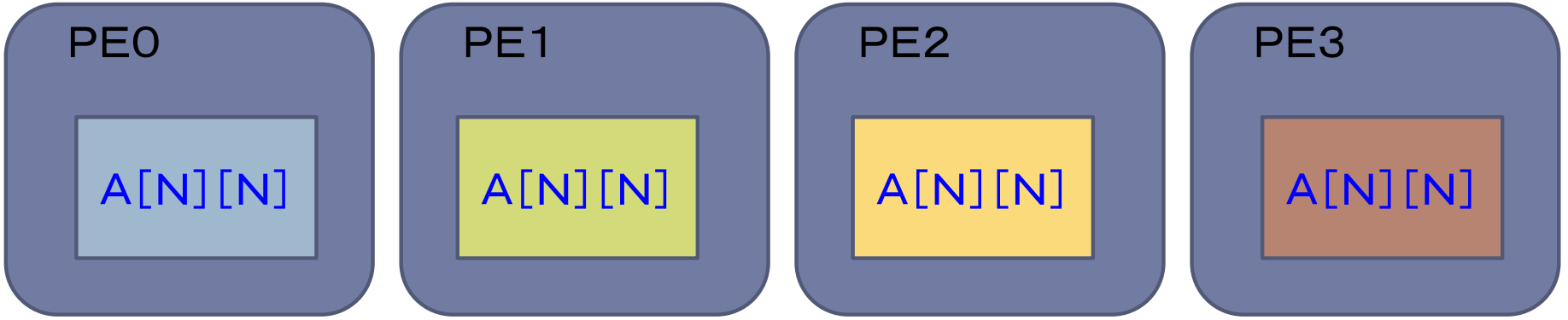
MPI並列化の大前提（再確認）

- 各PEでは、＜同じプログラムが同時に起動＞されて開始されます。

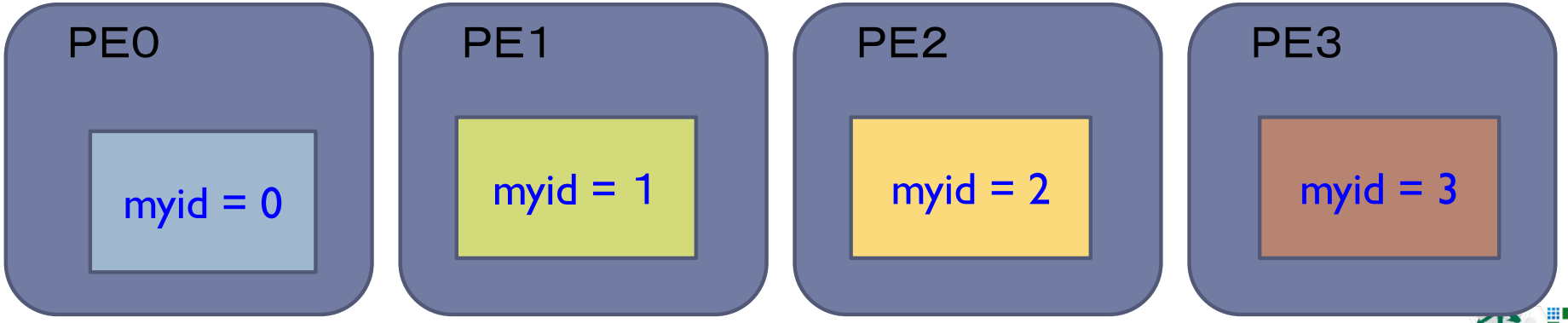


MPI並列化の大前提（再確認）

- 各PEでは、＜別配列が個別に確保＞されます。

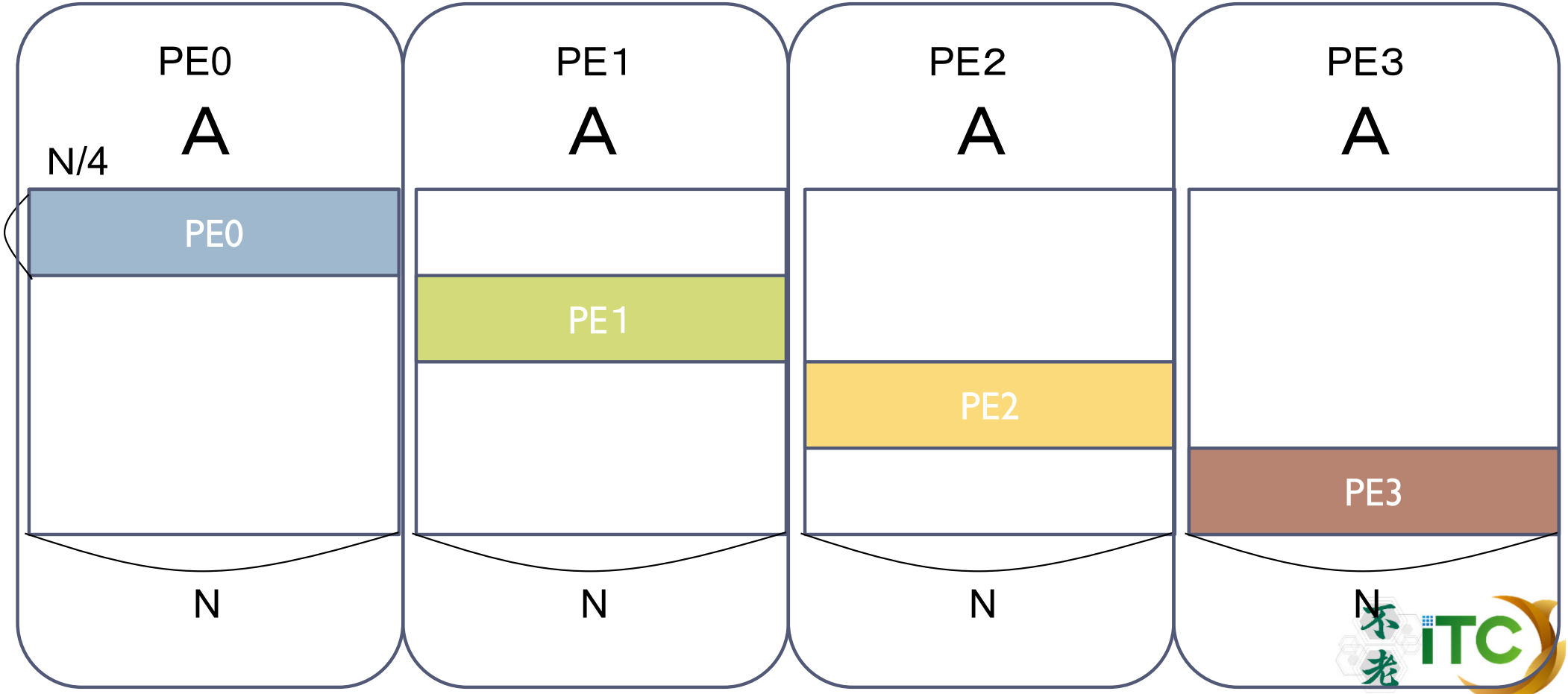


- myid変数は、MPI_Init()関数（もしくは、サブルーチン）が呼ばれた段階で、＜各PE固有の値＞になっています。



各PEでの配列の確保状況

- 実際は、以下のように配列が確保されていて、
  部分的に使うだけになります

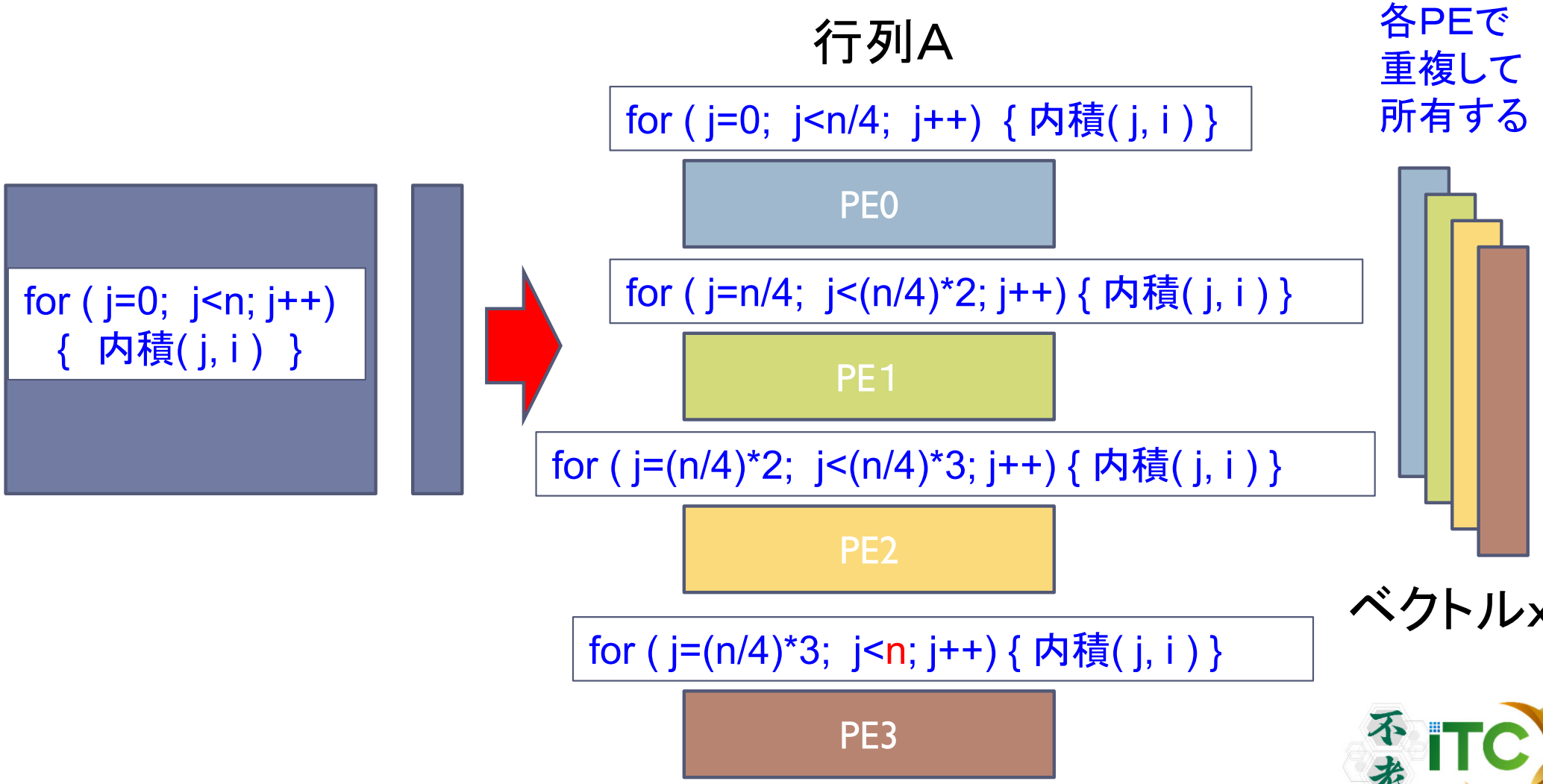


本実習プログラムのTIPS

- myid, numprocs は大域変数です
  - myid (=自分のID)、および、numprocs(=世の中のPE台数)の変数は大域変数です。MyMatVec関数内で、引数設定や宣言なしに、参照できます。
- myid, numprocs の変数を使う必要があります
  - MyMatMat関数を並列化するには、myid 、および、numprocs変数を利用しないと、並列化ができません。

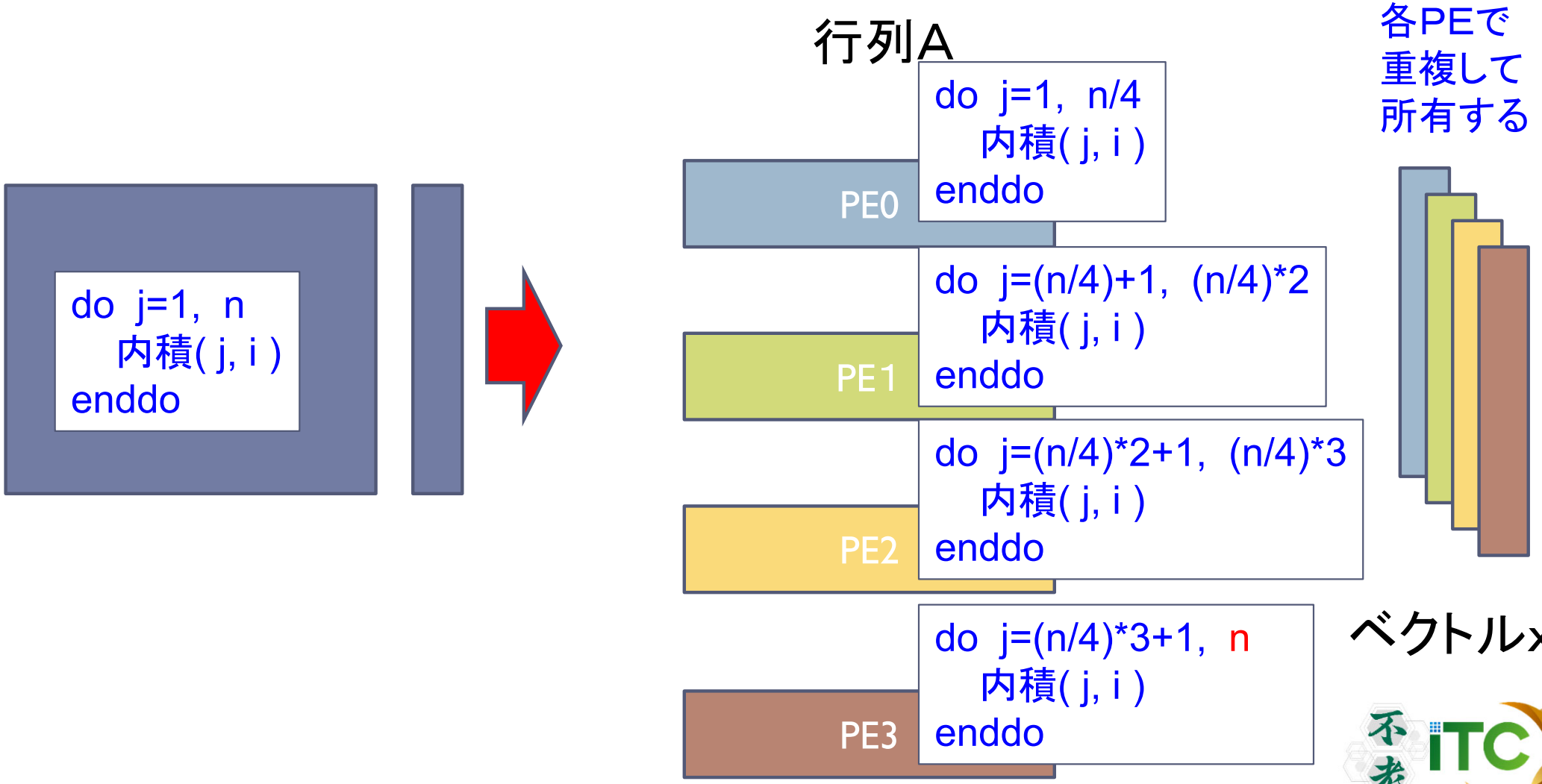
並列化の考え方 （行列-ベクトル積の場合、Ｃ言語）

- SIMDアルゴリズムの考え方（4PEの場合）



並列化の考え方（行列-ベクトル積の場合、Fortran言語）

- SIMDアルゴリズムの考え方（4PEの場合）



並列化の方針（C言語）

1. 全PEで行列AをN×Nの大きさ、ベクトルx、yをNの大きさ、確保してよいとする。
2. 各PEは、担当の範囲のみ計算するように、ループの開始値と終了値を変更する。
   - ブロック分散方式では、以下になる
     （n が numprocs で割り切れる場合）

```
ib = n / numprocs;
for ( j=myid*ib;  j<(myid+1)*ib;  j++)    { … }
```

3. （2の並列化が完全に終了したら）各PEで担当のデータ部分しか行列を確保しないように変更する。
   - 上記のループは、以下のようになる。

```
for ( j=0;  j<ib;  j++)  { … }
```

並列化の方針（Fortran言語）

1. 全PEで行列AをN×Nの大きさ、ベクトルx、yをNの大きさ、確保してよいとする。
2. 各PEは、担当の範囲のみ計算するように、ループの開始値と終了値を変更する。
   - ブロック分散方式では、以下になる
     （n が numprocs で割り切れる場合）

```
ib = n / numprocs
do j = myid*ib+1, (myid+1)*ib
    …..
enddo
```

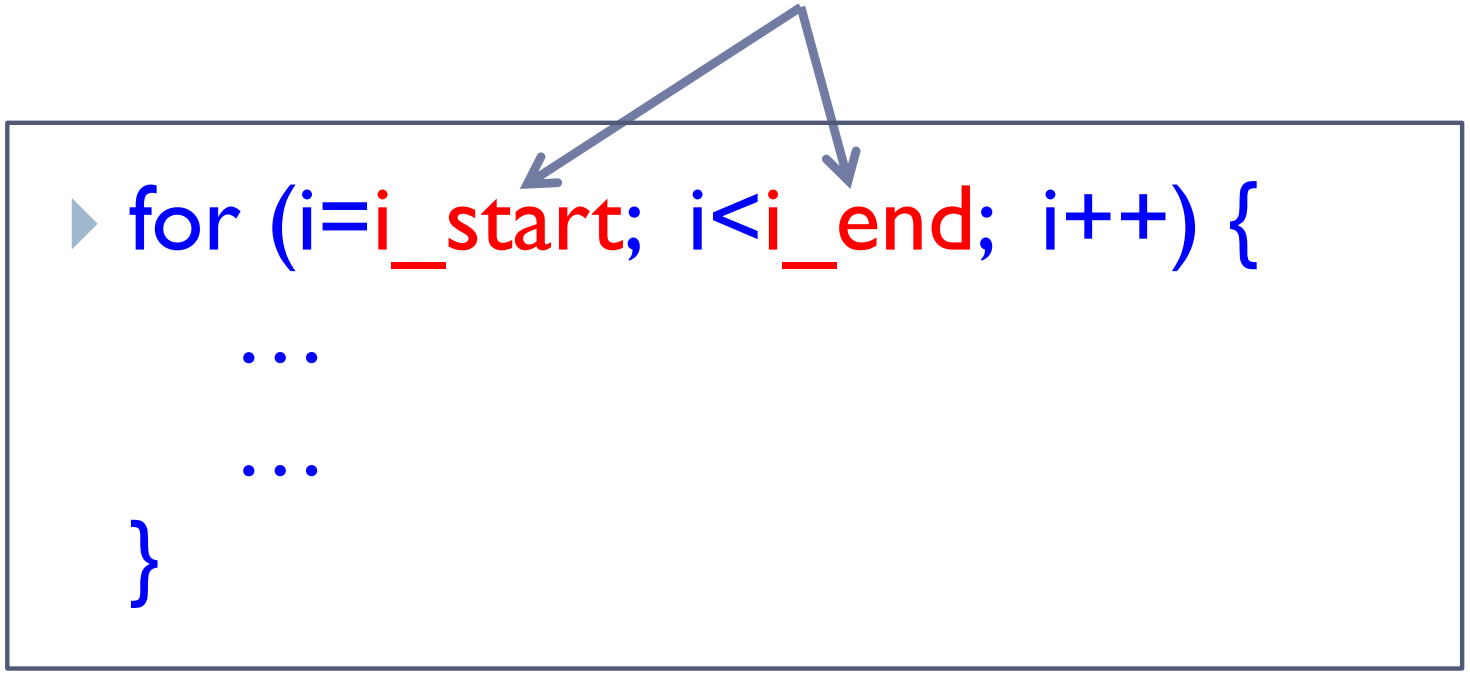
3. （2の並列化が完全に終了したら）各PEで担当のデータ部分しか行列を確保しないように変更する。
   - 上記のループは、以下のようになる。

```
do j=1, ib
    …
enddo
```

実装上の注意

- ループ変数をグローバル変数にすると、性能が出ません。
  必ずローカル変数か、定数（　2　など）にしてください。

ローカル変数にすること



参考資料

サンプルプログラムの実行（行列-行列積（その２））

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版のファイル名

  Mat-Mat-d-flow-fx.tar

行列-行列積(2)のサンプルプログラムの実行

- 以下のコマンドを実行する
  $ cp /center/a49904a/Mat-Mat-d-flow-fx.tar ./
  $ tar xvf Mat-Mat-d-flow-fx.tar
  $ cd Mat-Mat-d
- 以下のどちらかを実行
  $ cd C :C言語を使う人
  $ cd F :Fortran言語を使う人
- 以下共通
  $ make
  $ pjsub mat-mat-d.bash
- 実行が終了したら、以下を実行する
  $ cat mat-mat-d.bash.XXXXXX.out

行列-行列積のサンプルプログラムの実行（C言語版）

- 以下のような結果が見えれば成功

```
N  = 576
Mat-Mat time  = 0.000918 [sec.]
 416291.869587 [MFLOPS]
 Error! in ( 0 , 1 )-th argument in PE 0
 Error! in ( 0 , 1 )-th argument in PE 16
 Error! in ( 0 , 1 )-th argument in PE 13
 Error! in ( 0 , 1 )-th argument in PE 101
 Error! in ( 0 , 1 )-th argument in PE 105
 Error! in ( 0 , 1 )-th argument in PE 129
....
```

並列化が完成していないのでエラーが出ます。ですが、これは正しい動作です

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

```
 NN  =  576
Mat-Mat time  =  5.62599748373032E-03
 MFLOPS =  68709.95008167029
 Error! in ( 1 , 2 )-th argument in PE  0
 Error! in ( 1 , 2 )-th argument in PE  21
 Error! in ( 1 , 2 )-th argument in PE  22
 Error! in ( 1 , 2 )-th argument in PE  43
 Error! in ( 1 , 2 )-th argument in PE  16
 Error! in ( 1 , 2 )-th argument in PE  17
 Error! in ( 1 , 2 )-th argument in PE  19
 Error! in ( 1 , 2 )-th argument in PE  3
…
```

並列化が
完成して
いないので
エラーが出ます。
ですが、
これは正しい
動作です。

サンプルプログラムの説明

- #define N 576
  - 数字を変更すると、行列サイズが変更できます
- #define DEBUG 1
  - 「0」を「1」にすると、行列-行列積の演算結果が検証できます。
- MyMatMat関数の仕様
  - Double型の行列A((N/NPROCS)×N行列)と B((N×(N/NPROCS)行列))の行列積をおこない、Double型の(N/NPROCS)×N行列Cに、その結果が入ります。

Fortran言語のサンプルプログラムの注意

- 行列サイズNの宣言は、以下のファイルにあります。

  mat-mat-d.inc
- 行列サイズ変数が、NNとなっています。

  integer NN

  parameter (NN=576)

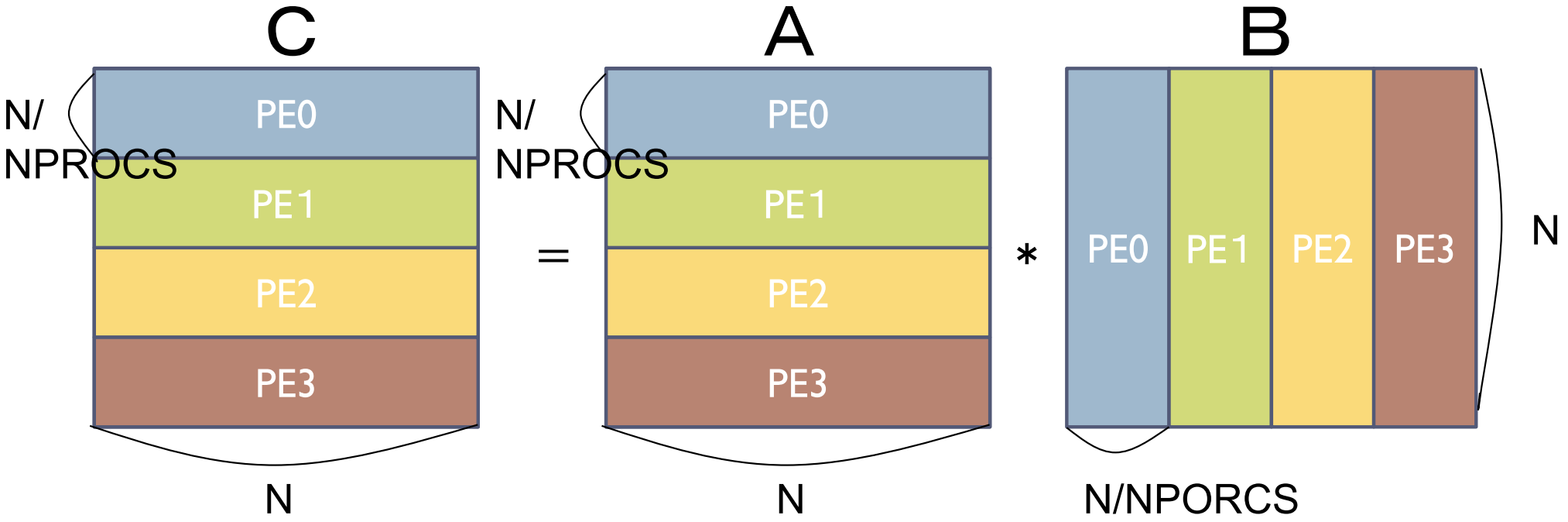
演習課題（１）

- MyMatMat関数（手続き）を並列化してください。
  - デバック時は
    #define　N　576
    としてください。

- 行列A、B、Cの初期配置（データ分散）を、十分に考慮してください。

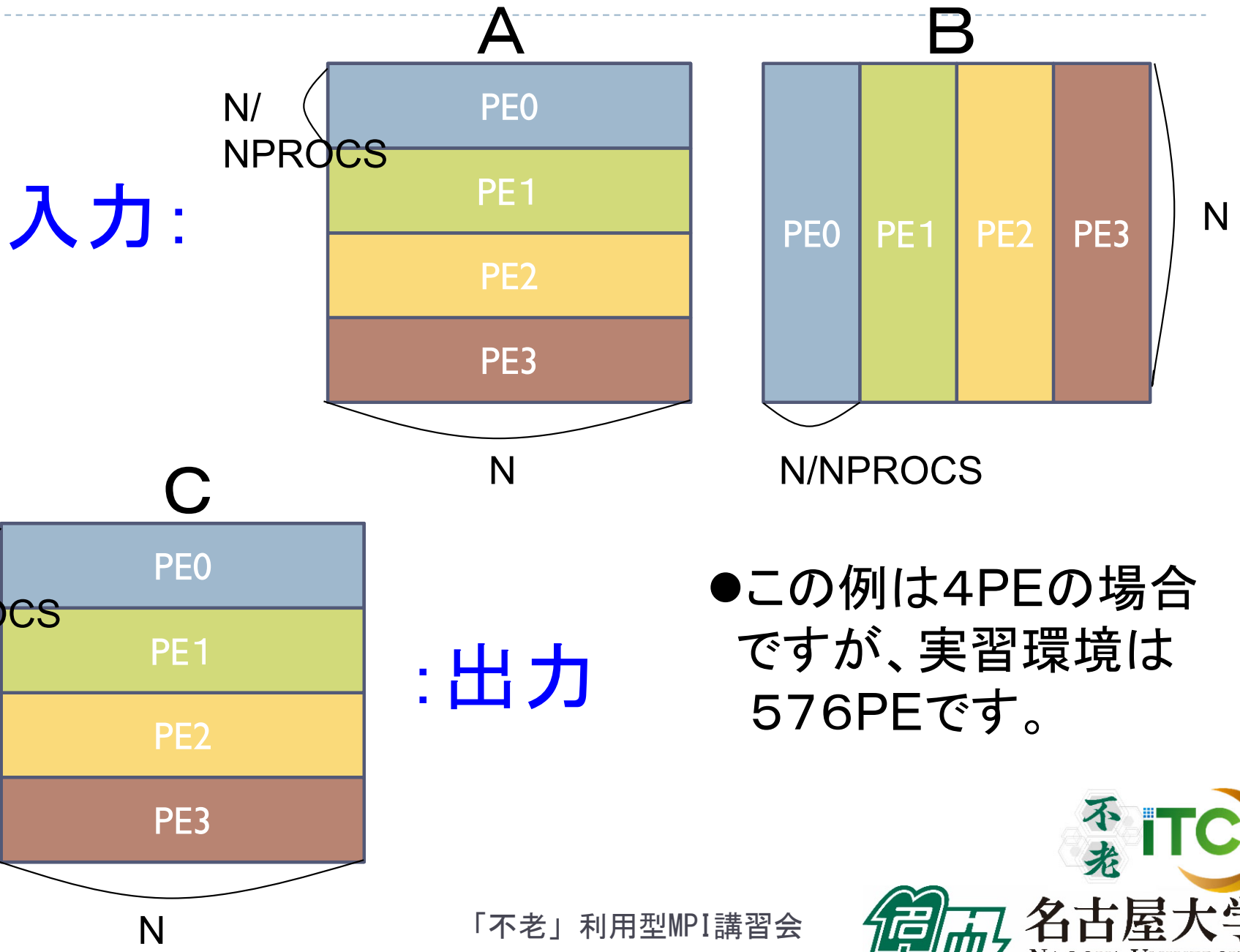
行列A、B、Cの初期配置

- 行列A、B、Cの配置は以下のようになっています。
  （ただし以下は4PEの場合で、実習環境は576PEです。）



- 1対1通信関数が必要です。
- 行列A、B、Cの配列のほかに、受信用バッファの配列が必要です。

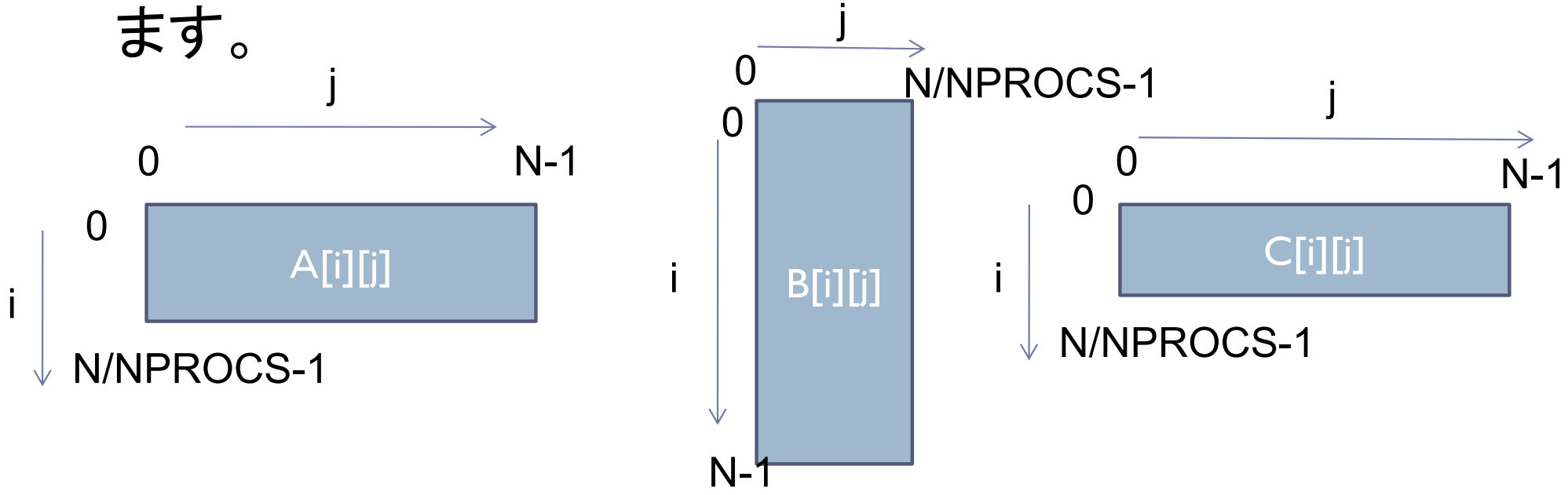
入力と出力仕様



●この例は4PEの場合ですが、実習環境は576PEです。

並列化の注意（C言語）

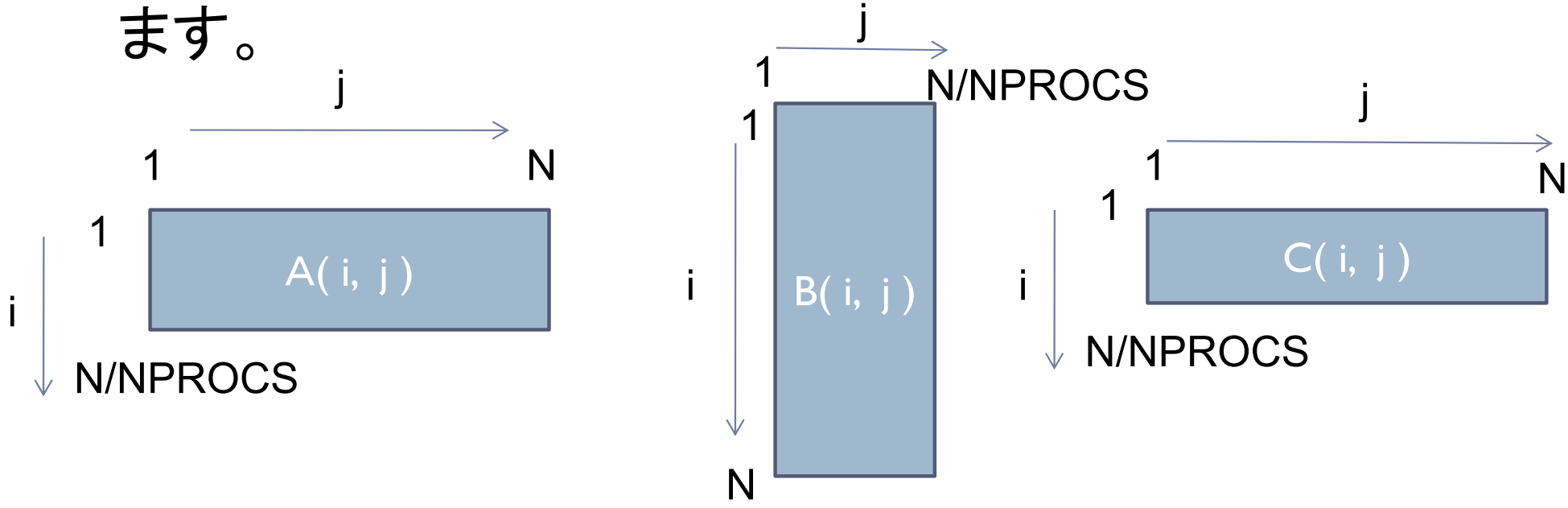
- 各配列は、完全に分散されています。
- 各PEでは、以下のようなインデックスの配列となっています。



- 各PEで行う、ローカルな行列-行列積演算時のインデックス指定に注意してください。

並列化の注意（Fortran言語）

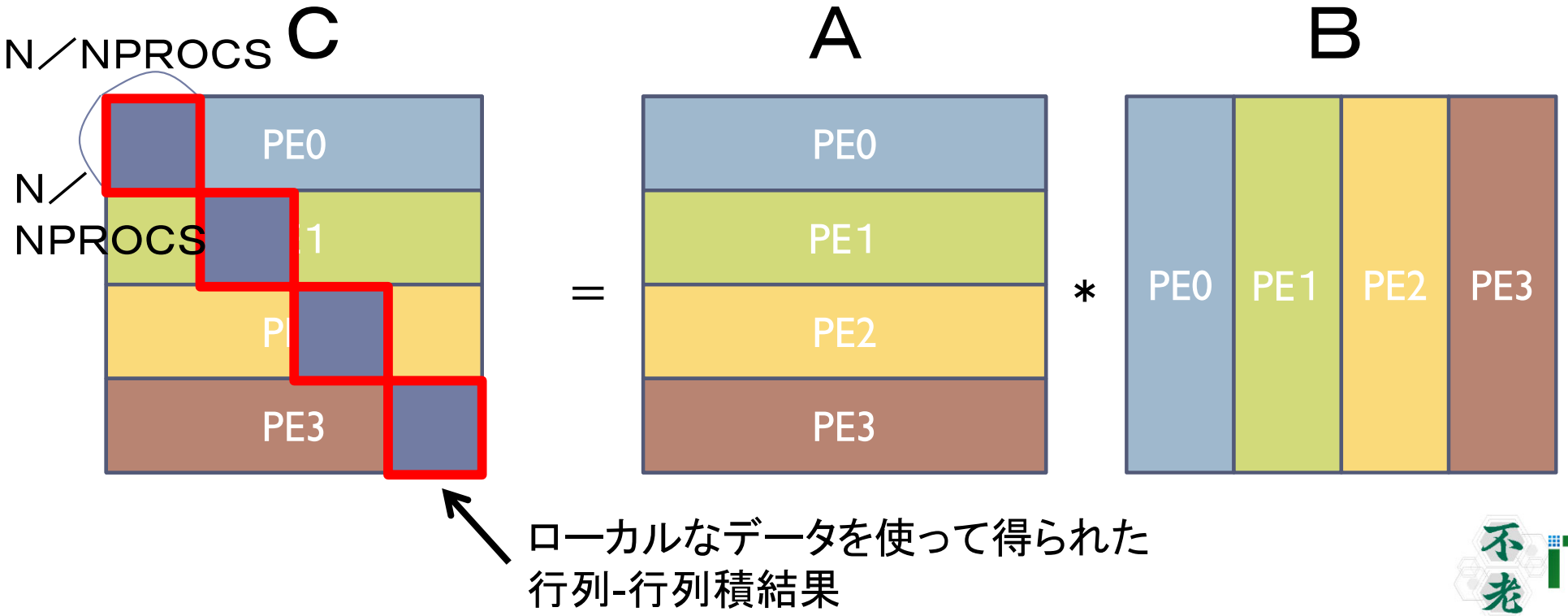
- 各配列は、完全に分散されています。
- 各PEでは、以下のようなインデックスの配列となっています。



- 各PEで行う、ローカルな行列-行列積演算時のインデックス指定に注意してください。

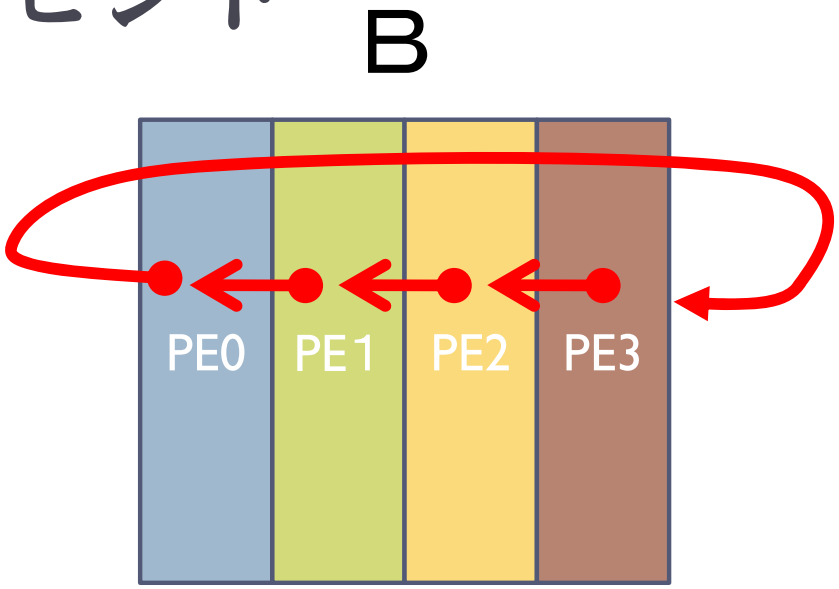
並列化のヒント

- 行列積を計算するには、各PEで完全な行列Bのデータがないので、行列Bのデータについて通信が必要です。
- たとえば、以下のように計算する方法があります。
- ステップ1

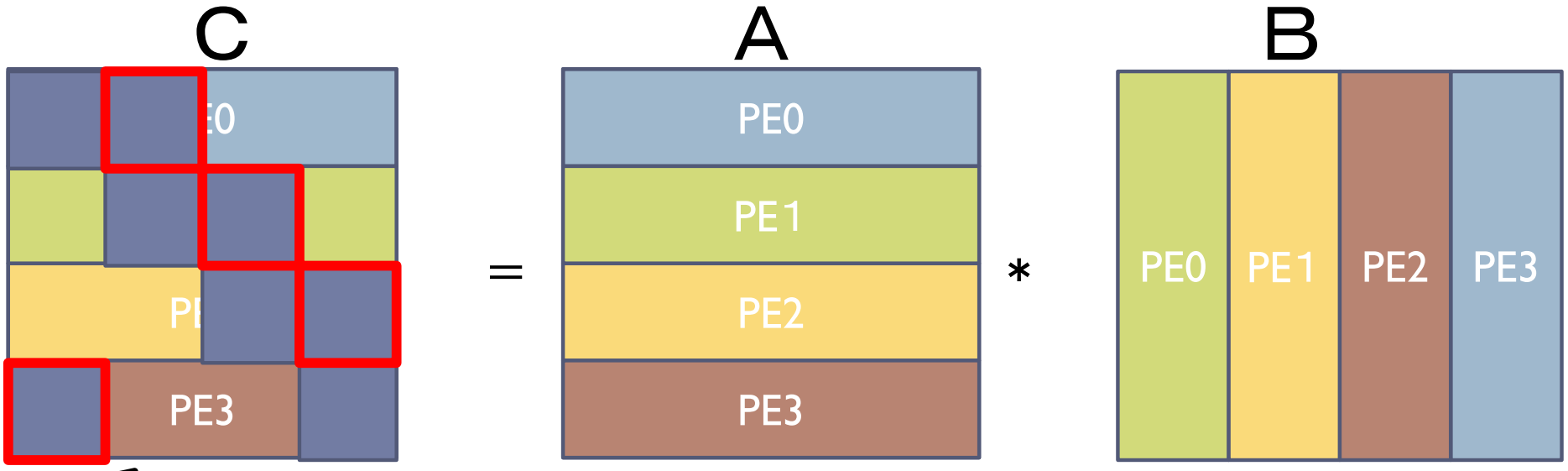


並列化のヒント

- ステップ2



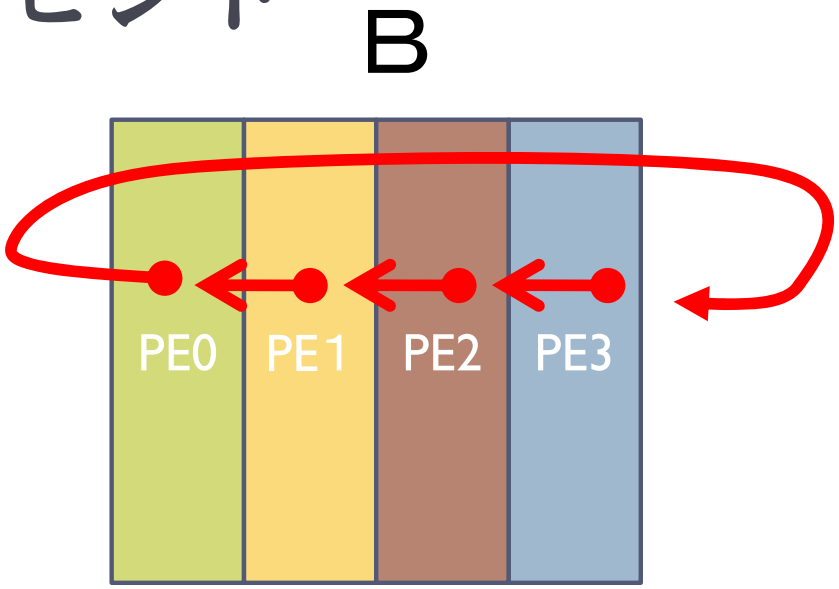
自分の持っているデータを
ひとつ左隣りに転送する
（PE0は、PE3に送る）
【循環左シフト転送】



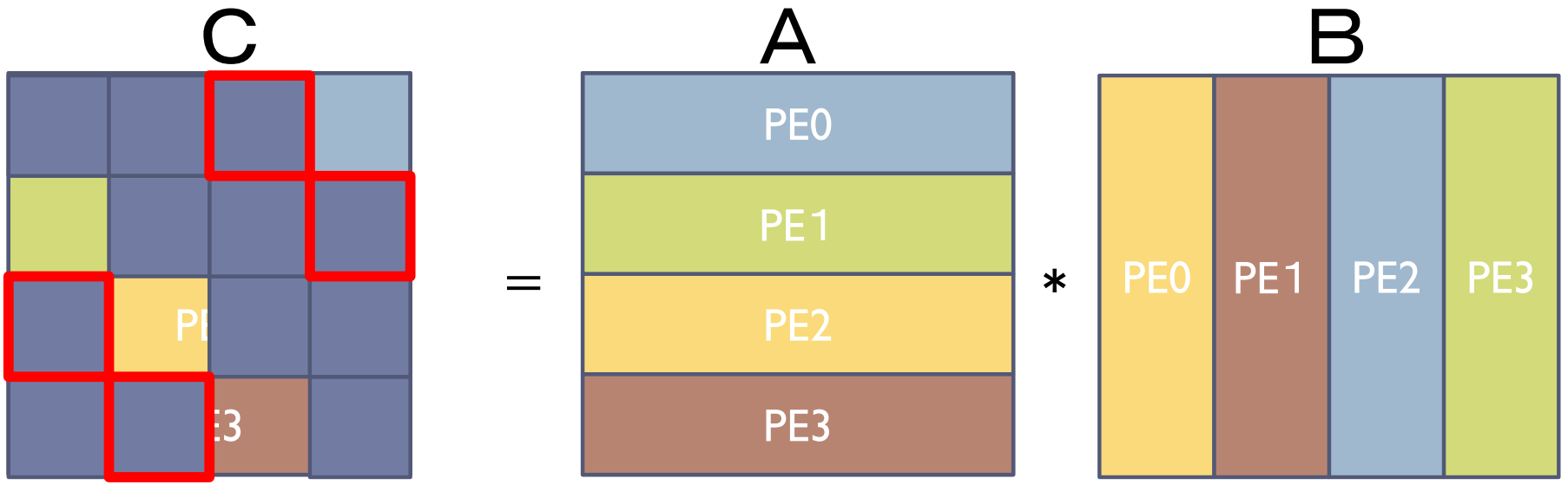
ローカルなデータを使って得られた
行列-行列積結果

並列化のヒント

- ステップ3



自分の持っているデータを
ひとつ左隣りに転送する
（PE0は、PE3に送る）
【循環左シフト転送】

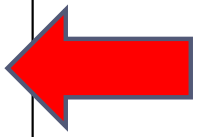


ローカルなデータを使って得られた
行列-行列積結果

並列化の注意（１／３）

- 循環左シフト転送を実装する際、全員がMPI_Sendを先に発行すると、その場所で処理が止まる。
（正確には、動いたり、動かなかったり、する）

```
…
  MPI_Send(…);
  MPI_Recv(…);
…
```



このMPI_Sendで止まる

並列化の注意（２／３）

- MPI_Send で止まる理由
  1. MPI_Sendの処理中で、大きいメッセージを送るとき、システムのバッファ領域がなくなる。
  2. バッファ領域が空くまで待つ（スピンウェイトする）。
  3. バッファ領域が空くには、相手の受信が呼ばれる必要がある。
  4. しかし、世の中に受信（MPI_Recv）をコールする人はいない。
  5. バッファ領域が、永遠に空かない。
     →一生、スピンウェイトから脱出できない。
     （MPI_Sendの箇所でずっと止まる）

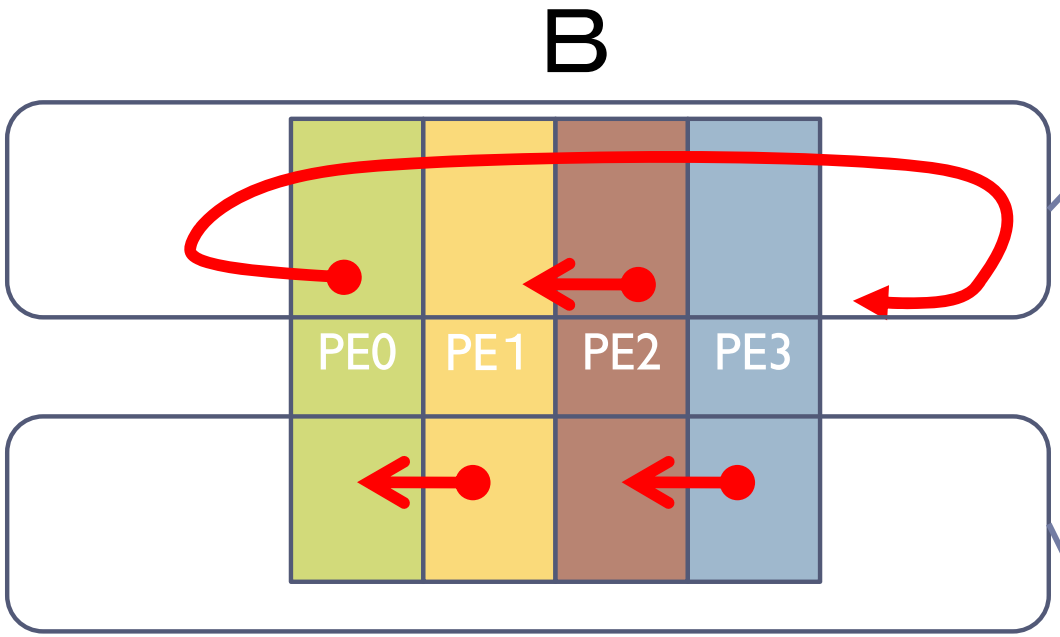
並列化の注意（３／３）

- これ（デットロック）を回避するため、以下の実装を行う。
  - PE番号が2で割り切れるPE：
    - MPI_Send();
    - MPI_Recv();
  - それ以外のPE：
    - MPI_Recv();
    - MPI_Send();

それぞれに対応

デットロック回避の通信パターン

- 以下の2ステップで、循環左シフト通信をする



ステップ1：
2で割り切れるPEがデータを送る

ステップ2：
2で割り切れないPEがデータを送る

基礎的なMPI関数—MPI_Recv（１／２）

- ierr = MPI_Recv(recvbuf, icount, idatatype, isource, itag, icomm, istatus);

  - recvbuf : 受信領域の先頭番地を指定する。
  - icount : 整数型。受信領域のデータ要素数を指定する。
  - idatatype : 整数型。受信領域のデータの型を指定する。
    - MPI_CHAR (文字型) 、MPI_INT (整数型)、MPI_FLOAT (実数型)、 MPI_DOUBLE(倍精度実数型)
  - isource : 整数型。受信したいメッセージを送信するPEのランクを指定する。
    - 任意のPEから受信したいときは、MPI_ANY_SOURCE を指定する。

基礎的なMPI関数—MPI_Recv（２／２）

- itag： 整数型。受信したいメッセージに付いているタグの値を指定する。
  - 任意のタグ値のメッセージを受信したいときは、MPI_ANY_TAG を指定する。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
  - 通常ではMPI_COMM_WORLD を指定すればよい。
- istatus： MPI_Status型（整数型の配列）。受信状況に関する情報が入る。専用の配列を宣言すること。
  - 要素数がMPI_STATUS_SIZEの整数配列が宣言される。
  - 受信したメッセージの送信元のランクが istatus[MPI_SOURCE]、タグが istatus[MPI_TAG] に代入される。
- ierr(戻り値)： 整数型。エラーコードが入る。

実装上の注意

- タグ(itag)について
  - MPI_Send(), MPI_Recv()で現れるタグ（itag）は、任意のint型の数字を指定してよいです。
  - ただし、同じ値（0など）を指定すると、どの通信に対応するかわからなくなり、誤った通信が行われるかもしれません。
  - 循環左シフト通信では、MPI_Send()と MPI_Recv()の対が、2つでてきます。これらを別のタグにした方が、より安全です。
  - たとえば、一方は最外ループの値iloopとして、もう一方をiloop+NPROCSとすれば、全ループ中でタグがぶつかることがなく、安全です。

さらなる並列化のヒント

以降、本当にわからない人のための資料です。
ほぼ回答が載っています。

並列化のヒント

1. 循環左シフトは、PE総数-1回 必要
2. 行列Bのデータを受け取るため、行列B[][]に関するバッファ行列B_T[][]が必要
3. 受け取ったB_T[][] を、ローカルな行列-行列積で使うため、B[][]へコピーする。
4. ローカルな行列-行列積をする場合の、対角ブロックの初期値：ブロック幅*myid。
   ループ毎にブロック幅だけ増やしていくが、Nを超えたら0に戻さなくてはいけない。

並列化のヒント（ほぼ回答，C言語）

- 以下のようなコードになる。

```
ib = n/numprocs;
for (iloop=0;  iloop<NPROCS;  iloop++ )  {
    ローカルな行列-行列積  C  =  A * B;
    if (iloop != (numprocs-1) ) {
      if (myid % 2 == 0 ) {
          MPI_Send(B, ib*n, MPI_DOUBLE, isendPE,
                iloop, MPI_COMM_WORLD);
          MPI_Recv(B_T, ib*n, MPI_DOUBLE, irecvPE,
                iloop+numprocs, MPI_COMM_WORLD, &istatus);
      } else {
          MPI_Recv(B_T, ib*n, MPI_DOUBLE, irecvPE,
                iloop, MPI_COMM_WORLD, &istatus);
          MPI_Send(B, ib*n, MPI_DOUBLE, isendPE,
                iloop+numprocs, MPI_COMM_WORLD);
      }
      B[ ][ ]  へ  B_T[ ][ ]  をコピーする;
    }
  }
```

並列化のヒント（ほぼ回答，C言語）

- ローカルな行列-行列積は、以下のようなコードになる。

```
jstart=ib*( (myid+iloop)%NPROCS );
for (i=0;  i<ib;  i++) {
   for(j=0;  j<ib;  j++) {
      for(k=0;  k<n;  k++) {
         C[ i ][ jstart + j ] += A[ i ][ k ] * B[ k ][ j ];
      }
   }
}
```

並列化のヒント（ほぼ回答，Fortran言語）

- 以下のようなコードになる。

```
ib = n/numprocs
do iloop=0,  NPROCS-1
    ローカルな行列-行列積  C  =  A * B
    if (iloop .ne. (numprocs-1) ) then
      if (mod(myid, 2)  .eq. 0 ) then
         call MPI_SEND(B, ib*n, MPI_DOUBLE_PRECISION,  isendPE,
&                iloop, MPI_COMM_WORLD,  ierr)
         call MPI_RECV(B_T,  ib*n,  MPI_DOUBLE_PRECISION,  irecvPE,
&                iloop+numprocs, MPI_COMM_WORLD,  istatus,  ierr)
      else
         call MPI_RECV(B_T,  ib*n,  MPI_DOUBLE_PRECISION,  irecvPE,
&                iloop,  MPI_COMM_WORLD,  istatus,  ierr)
         call MPI_SEND(B,  ib*n,  MPI_DOUBLE_PRECISION,  isendPE,
&                iloop+numprocs,  MPI_COMM_WORLD,  ierr)
      endif
      B  へ  B_T  をコピーする
    endif
  enddo
```

並列化のヒント（ほぼ回答，Fortran言語）

- ローカルな行列-行列積は、以下のようなコードになる。

```
imod = mod( (myid+iloop), NPROCS )
jstart = ib* imod
do i=1, ib
   do j=1, ib
      do k=1, n
         C( i , jstart + j ) = C( i , jstart + j ) +  A( i , k ) * B( k , j )
       enddo
     enddo
 enddo
```

おわり

お疲れさまでした