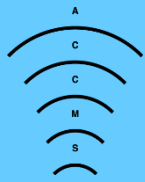




計算科学演習B

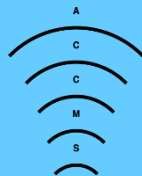
スーパーコンピュータ & 並列計算 概論

学術情報メディアセンター
情報学研究科・システム科学専攻
中島 浩



目次

- **科目概要**
 - **目標・スケジュール・スタッフ・講義資料・課題**
- **スーパーコンピュータ概論**
 - **一般のスーパーコンピュータ**
 - **京大のスーパーコンピュータ**
 - **スーパーコンピュータの構造**
- **並列計算概論**
 - **並列計算の類型・条件**
 - **Scaling & Scalability、問題分割、落とし穴**
 - **プロセス並列 & スレッド並列、バリア同期**
 - **バッチジョブ**



目標・スケジュール・スタッフ・講義資料

■ 目標

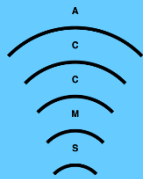
- 拡散方程式の初期値(&境界値)問題を題材に
- MPI と OpenMP を用いた並列プログラムを作成して
- 並列プログラミングの基礎と (やや高度な) 応用を学ぶ

■ スケジュール

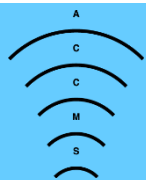
第1日	概論(中島)	拡散方程式(深沢)	逐次プログラム作成
第2日	OpenMP基礎(深沢)		WS型プログラム作成
第3日	MPI基礎(中島)		1D分割プログラム作成
第4日	MPI発展(中島)		2D分割プログラム作成
第5日	OpenMP発展(深沢)		SPMD/Hybrid型プログラム作成
~10/5(金)	レポート仕上げ		

■ 講義資料

http://ais.sys.i.kyoto-u.ac.jp/~fukazawa/CSEB_2018/

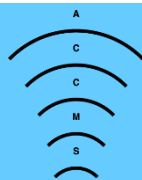


- 課題 = **拡散方程式**の初期値・境界値問題 by 陽解法
 - C or Fortran + **MPI** / **OpenMP**
 - 課題1: 逐次プログラム
 - 課題2(1): **MPI** + 1次元分割
 - 課題2(2): **MPI** + 2次元分割
 - 課題3(1): **OpenMP** + Work Sharing
 - 課題3(2): **OpenMP** + SPMD/Hybrid
- 提出物・提出先・期限
 - 作成したプログラム5種 (以上) のソースファイル
 - 課題内容・手法説明・プログラム概要・結果考察のレポート (MS Word or PDF)
 - h.nakashima@media.kyoto-u.ac.jp
fukazawa@media.kyoto-u.ac.jp
 - **10月5日(金) 23:59 必着**

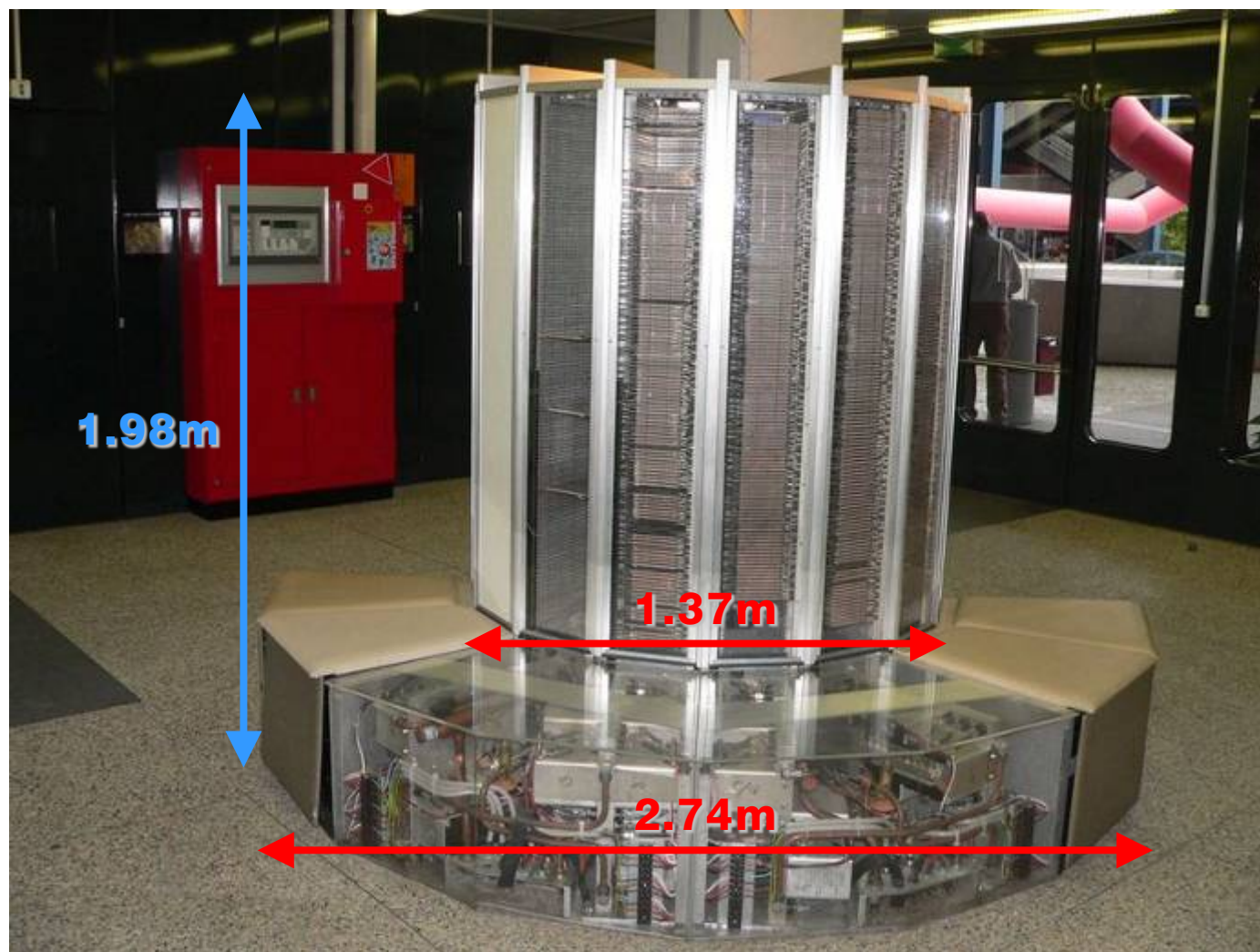


一般のスパコン:ベクトルマシン (1/2)

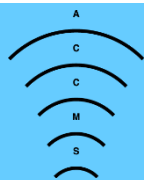
- 1976年:最初のスパコン**Cray-1**登場
 - 動作周波数=80MHz (< 携帯電話)
 - 演算性能=**160MFlops** (< 携帯電話)
 - Flops: Floating-point Operation Per Second
= 10進16桁精度の数値($10^{-308} \sim 10^{308}$)の加減乗算回数/秒
→160MFlops = 毎秒1億6千万回の加減乗算
 - 消費電力=115kW
 - **大量の数値データ** (ベクトル) に対する同種演算が得意
- その後
 - 1980年代:スパコン≡ベクトル (並列) マシンの時代
 - 1990年代:スカラ並列マシン (後述) との激闘
 - 2002年:**地球シミュレータ**が7年振りにベクトルで最速に
 - 現在:TOP500 (後述) にはランクせず



一般のスパコン:ベクトルマシン (2/2)



source: <http://en.wikipedia.org/wiki/Image:Cray-1-p1010221.jpg>



一般のスパコン: スカラ並列マシン

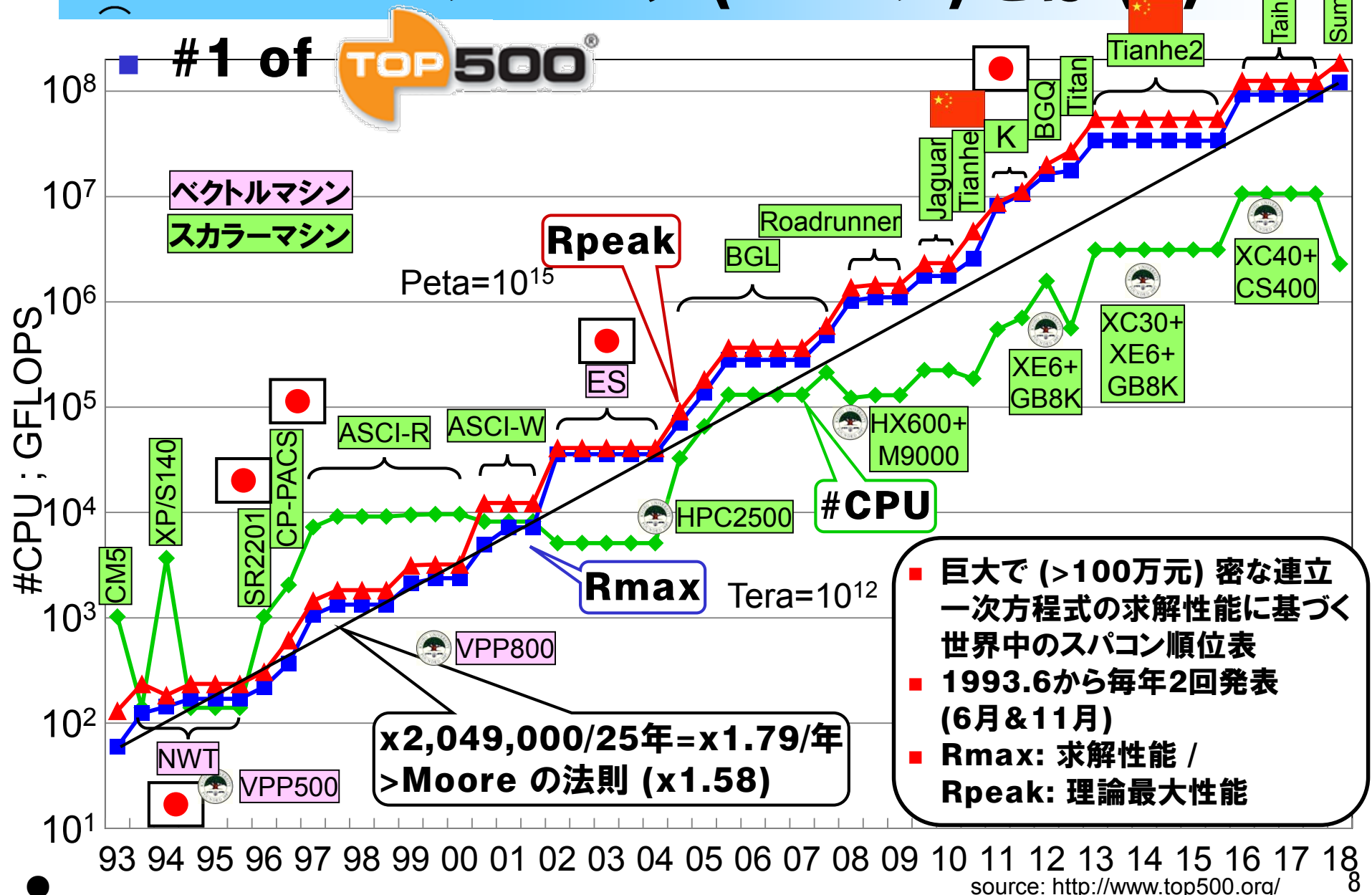
■ 1980年代に出現

- Sequent Balance : 20 x NS32016 ('84)
- Intel iPSC/1: 128 x i80286 ('85)

■ 多数のパソコン (のようなもの) の集合体

- 個々の部品 (CPU, メモリなど) \equiv パソコン (& ゲーム機)
 - 実際にTOP500 (後述) では ...
 - x86 = 477(95.4%) v.s. others = 23(4.6%)
- ただしメチャクチャに数が多い
 - パソコン = 1~16 CPU
 - 京大スパコン = 154,152 CPU
 - 世界最高速スパコン = 202,752 CPU + 2,211,840 SM
 - 世界最大規模スパコン = 10,649,600 CPU
- 同じような計算の集合体としての巨大計算が得意

スーパーコンピュータ (スパコン) とは (3)




京大のスーパーコンピュータ (1)

Camphor 2




CRAY XC40

 Xeon Phi KNL @ 1.4GHz/68c
 3.05TFlops : (16+96)GB : 15.8GB/s
 → 5.48PFlops (1,800n / 122,400c) :
 197TB : 15.5TB/s
DATAWARP (burst buffer)
 230TB, 200GB/sec

Storage



ExaScaler


16+8PB, 100+50GB/sec
 **IME** (burst buffer)
 230TB, 250GB/sec



InfiniBand FDR/EDR (6.8/12.1GB/s•link)

Laurel 2


CRAY CS400 2820XT

 Xeon Broadwell @ 2.1GHz/18c x 2
 1.21TFlops : 128GB : 12.1GB/s
 → 1.03PFlops (850n / 30,600c) : 106TB



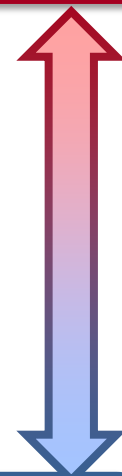
Cinnamon 2

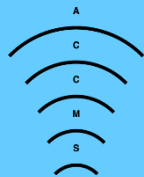
CRAY CS400 4840X

 Xeon Haswell @ 2.3GHz/18c x 4
 2.65TFlops : 3TB : 24.2GB/s
 → 42.3TFlops (16n / 1,152c) : 48TB



Omni-Path (12.1GB/s•link, BB= 5.15TB/s)





京大のスーパーコンピュータ (2)

- 日本で第**9**&**29**位
世界で第**54**&**397**位の性能
- パソコンなどと比べると ...

毎秒6554兆回の
加減乗算

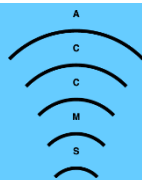
	京大スパコン	パソコン	倍率
演算性能	6554 TFlops	67.2 GFlops	x 97500
メモリ容量	351 TByte	8 GByte	x 44900
通信性能	32.3 TByte／秒	100 MByte／秒 (フレッツ光ネクスト)	x 323000
ディスク容量	24PByte	550 GByte	x 43600

Peta = 10^{15} = 1000兆

Giga = 10^9 = 10億

Tera = 10^{12} = 1兆

Mega = 10^6 = 100万

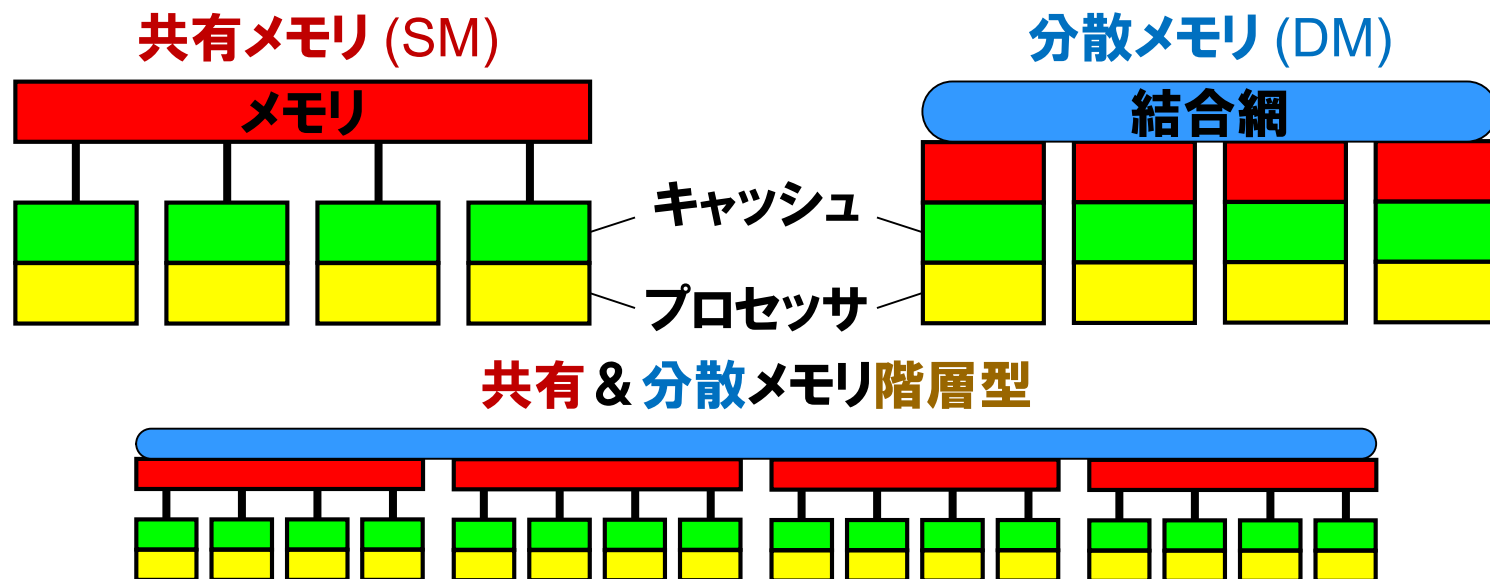


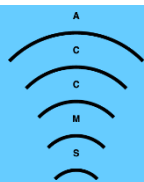
スーパーコンピュータ概論:スパコンの構造

共有メモリと分散メモリ

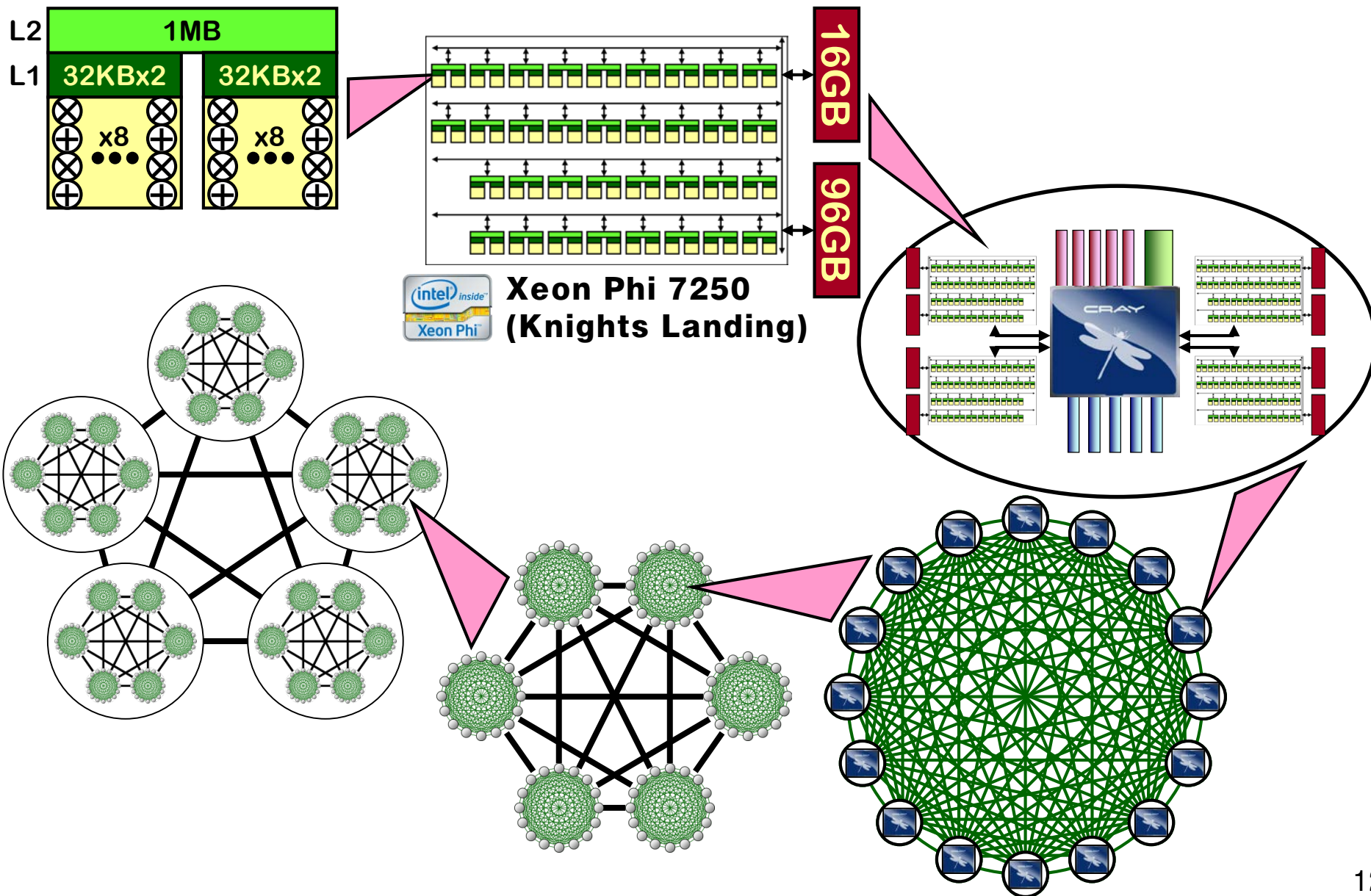
© 2009-2018 H. Nakashima

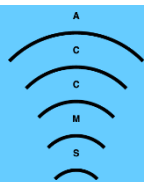
- **共有メモリ型**
 - (論理的に) 1つのメモリをプロセッサが共有 → 変数共有可能
→ あるプロセッサが代入した値を別のプロセッサが参照可能
 - 一般に小規模 (プロセッサ数 = $10^0 \sim 10^2$ のオーダー)
- **分散メモリ型**
 - 別々のコンピュータをネットワークで繋いだもの
→ プロセッサ間のデータのやり取りには陽に「通信」が必要
 - 大規模な構成が比較的容易 ($\sim 10^5$ のオーダー)
- **共有 & 分散メモリ階層型: 最近の主流**





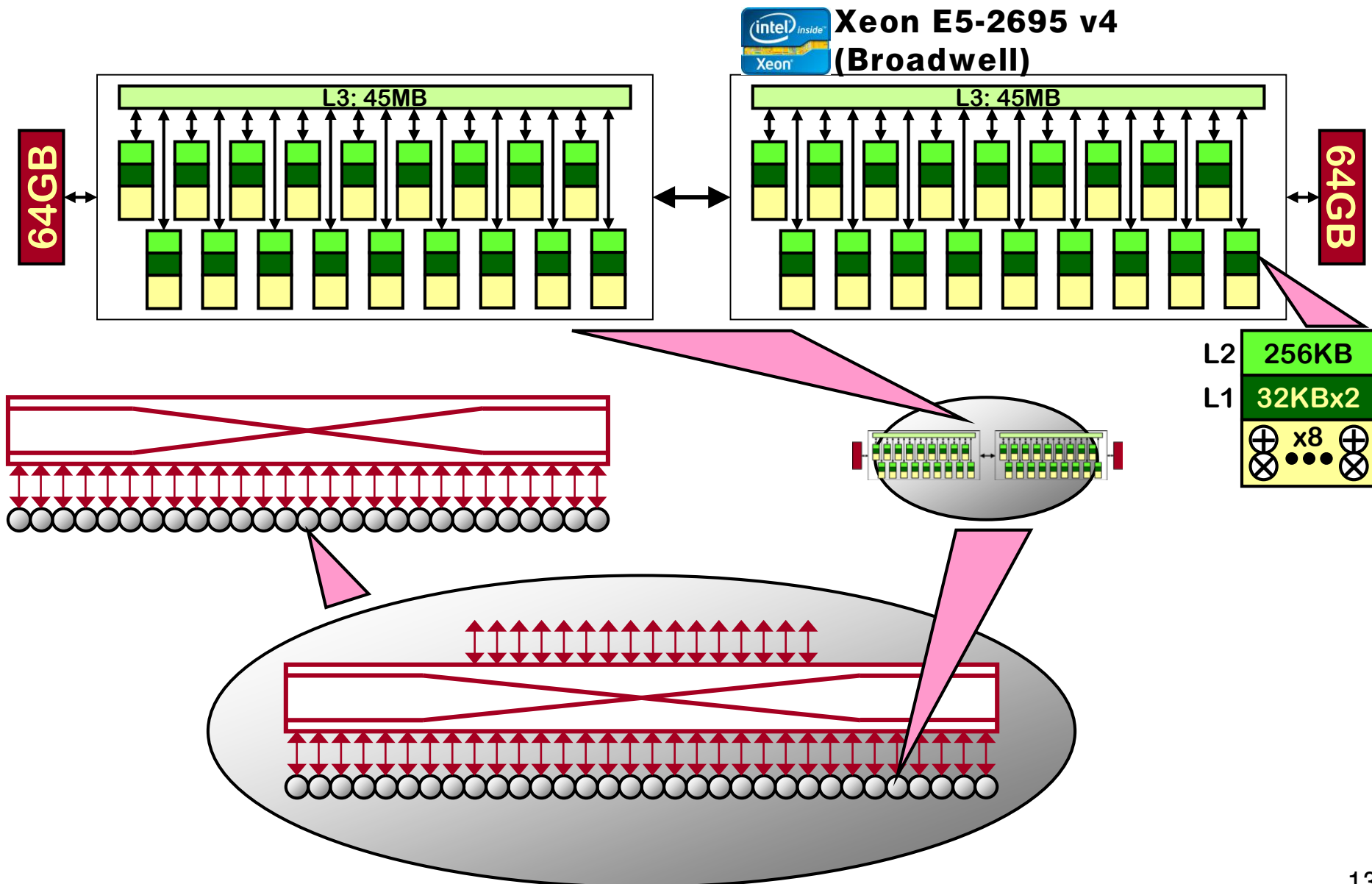
京大スパコンの構造 (Camphor2=XC40)

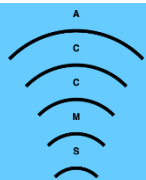




スーパーコンピュータの構造 京大スパコンの構造 (Laurel2=CS400)

© 2009-2018 H. Nakashima





並列計算の類型

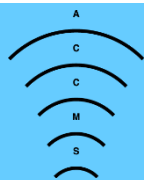
■ スパコンを使って計算をする理由＝高速計算

- 高速なプロセッサを使う＝（特に今後は）**困難**
- 多数のプロセッサを使う＝（昔も今後も）**可能**

■ 問題 **X** の逐次実行時間 **$T(X,1)$**

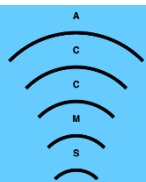
P 個のプロセッサでの並列実行時間 **$T(X,P)$**

- $T(X,P) \doteq T(X,1) / P$ **→ strong scaling**
- $X^P = X$ の P 倍の規模（メモリ量、計算量など）の問題
 $T(X^P, P) \doteq T(X,1)$ **→ weak scaling**
- $X_1, \dots, X_P = X$ の P 個のインスタンス
 $T(\{X_1, \dots, X_P\}, P) \doteq T(X,1)$ **→ capacity computing**



並列計算の条件

- **P** 倍程度の性能を得るための**必要**条件
 - 問題を計算量が $1/P$ 程度の**部分問題**に分割可能
 - 部分問題の必要メモリ \leq 問題の必要メモリの k/P ($+\alpha$)
(特に weak scaling で重要)
 - 分割不能計算量 X^{seq} \ll 分割可能計算量 X^{para}
(strong scaling の一般的な限界)
 - 部分問題について通信時間／計算時間 $< O(1)$
- **P** 並列計算時間の粗い見積
 - $T(X, P) \doteq T(X^{\text{seq}}, 1) + T(X^{\text{para}}, 1) / P + \text{通信時間}$
 - **通信時間** \doteq 通信データ量／ B + 通信回数 $\times L$
 - B : バンド幅 $\doteq 1 \sim 15 \text{GB/s}$
 - L : 遅延 + オーバヘッド $\doteq 1 \sim 50 \mu\text{s}$

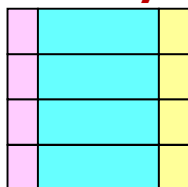


Scaling & Scalability

分割不能計算 → 分割可能計算

strong scaling
($P \uparrow \Rightarrow \text{問題} =$)

$P=4$



weak scaling
($P \uparrow \Rightarrow \text{問題} \uparrow$)

通信

$P=16$



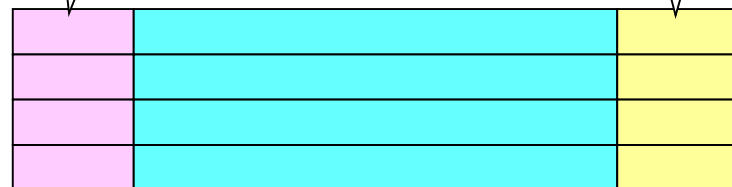
Amdahl則



分割不能
 \propto 分割可能

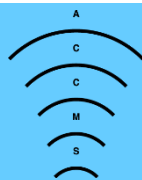
通信量 $\propto P$

スケールしない



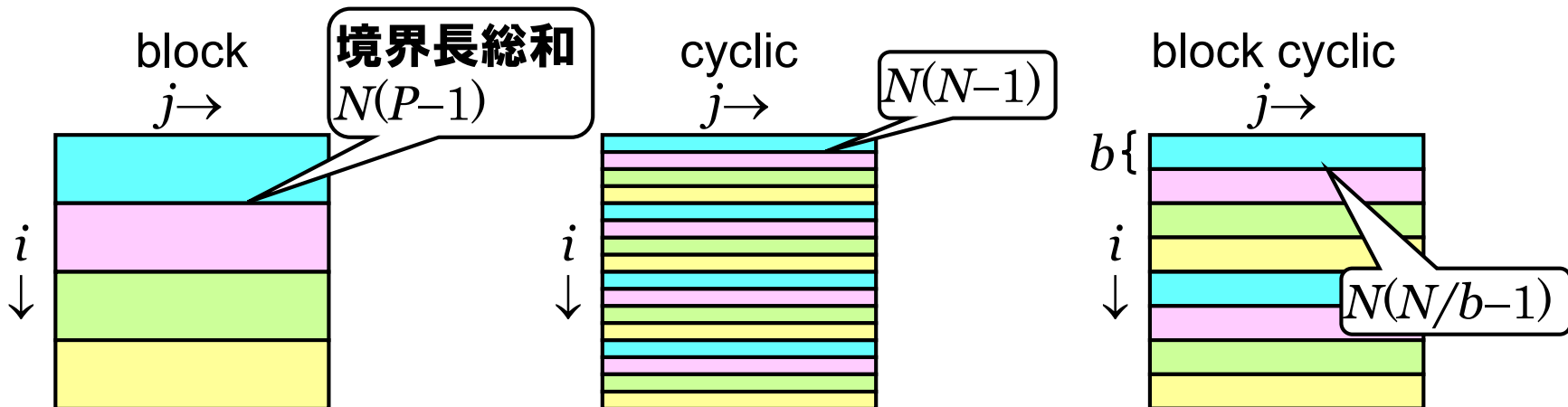
⋮





問題分割 (1/2)

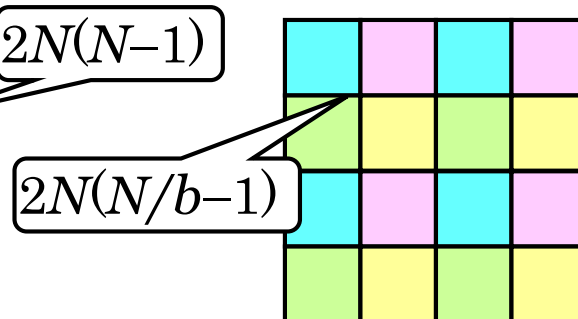
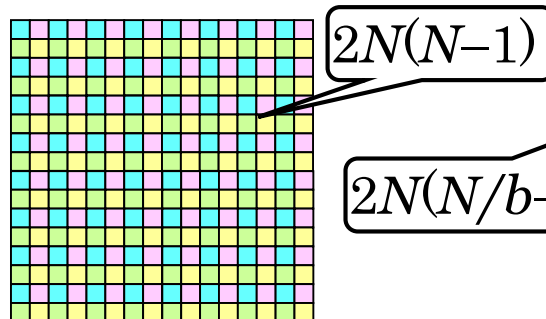
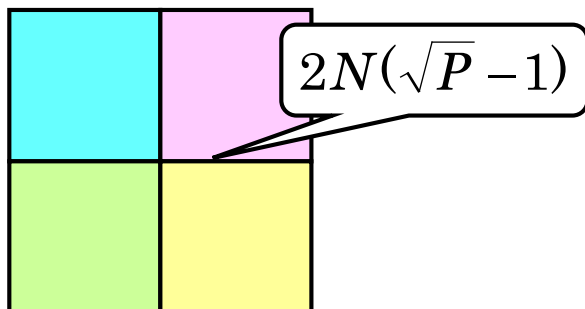
■ $N \times N$ の2次元配列 (空間) 問題のP分割法

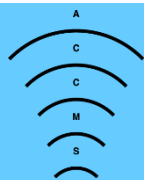


計算負荷が均一分布ならOK
(不均一分布 \rightarrow 負荷不均衡)
境界長総和 (\equiv 通信量) は最小

計算負荷の不均一分布に強い
境界長総和 (\equiv 通信量) は最大

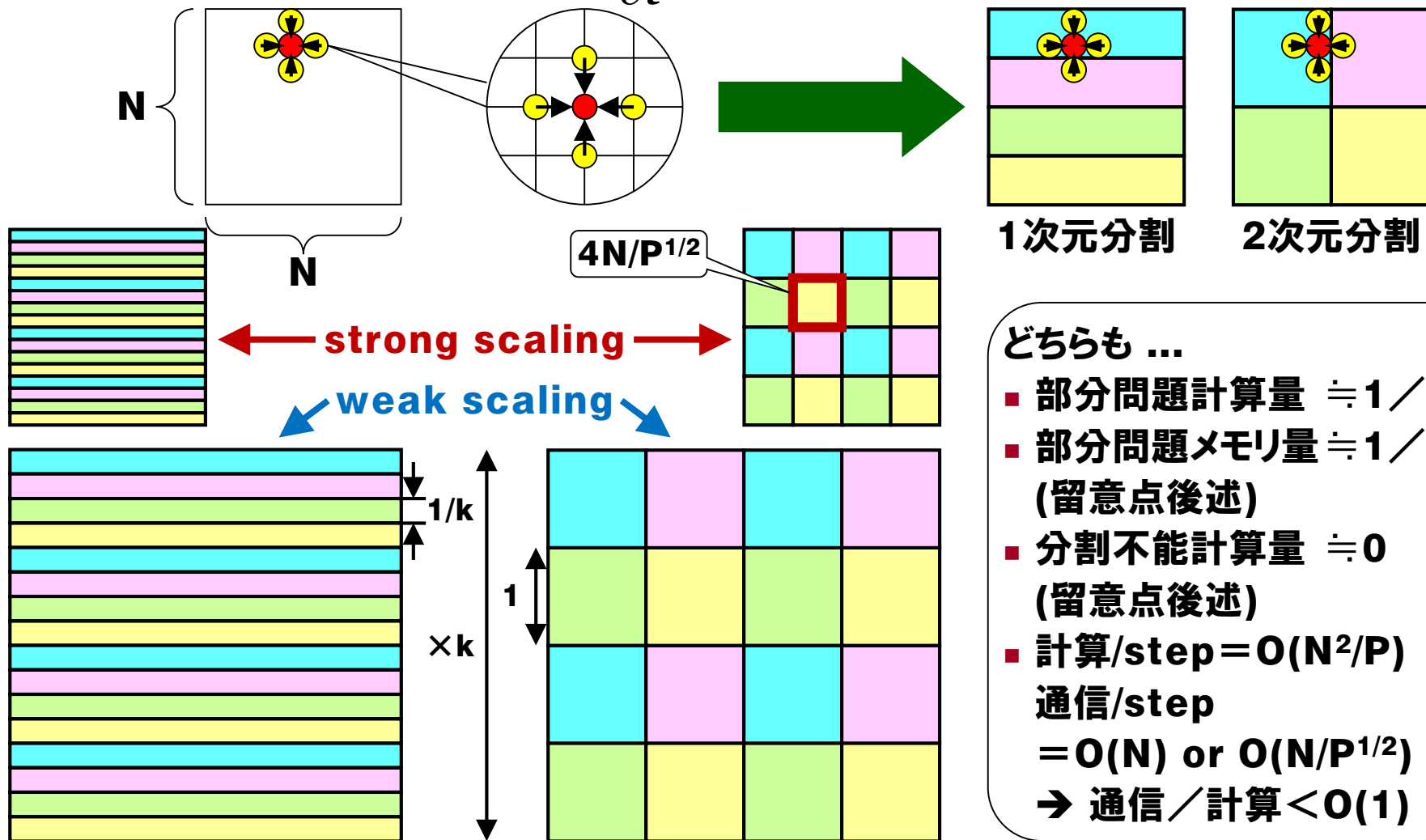
block/cyclicの折衷
負荷分布と境界長の
トレードオフが必要なら有効

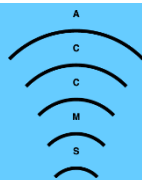




問題分割 (2/2): 拡散方程式では...

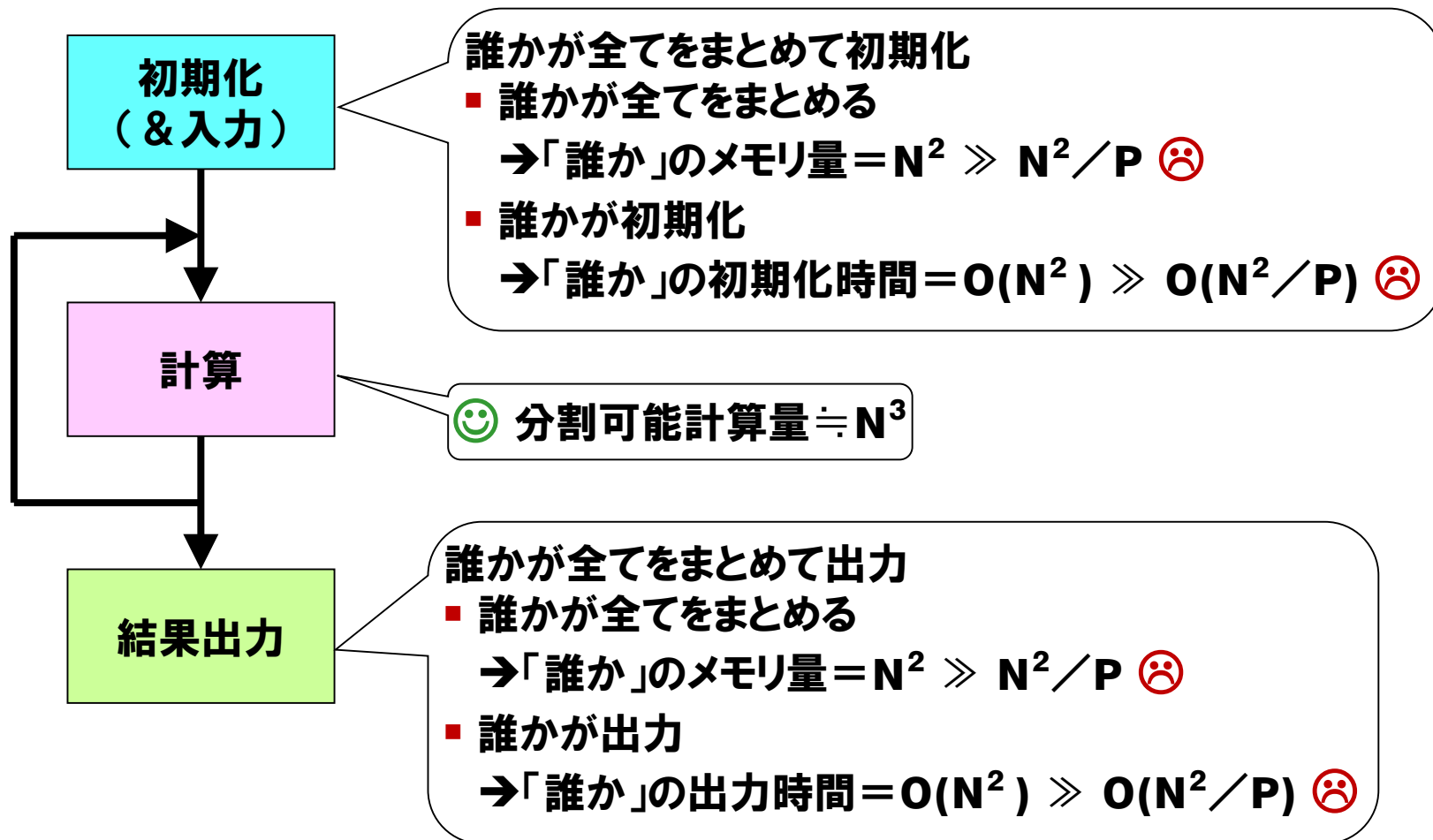
- 拡散方程式 $\nabla^2 \varphi = \frac{\partial \varphi}{\partial t}$ の初期値問題求解 by 陽解法

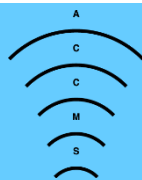




拡散方程式プログラムの落とし穴

- 部分問題メモリ量 $\doteq 1/P$ & 分割不能計算量 $\doteq 0$?





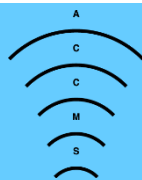
プロセス並列 & スレッド並列 (1/2)

- **プロセス並列 (aka Message Passing) v.s. スレッド並列 (aka Shared Memory)**
 - プログラミング・パラダイムとしての分類 (≠H/W の分類)
 - ライブラリ／言語のコンセプト

	プロセス並列(PP)	スレッド並列(TP)
並列実行単位	プロセス	スレッド
アドレス空間	(プロセスに) 固有	スレッド間で 共有
通信(&同期)	通信用プリミティブ (e.g. send, recv)	共有変数アクセス + 同期プリミティブ
ライブラリ	(e.g.) MPI	(e.g.) OpenMP
H/W=DM	◎	△ (S/W DSM, 言語)
H/W=SM	○	◎
折衷型	ハイブリッド並列	

計算の
歩調合せ

実質的
には×



プロセス並列 & スレッド並列 (2/2)

■ PP v.s. TP のアナロジー

■ PP ≡ メールの添付ファイル

- データ転送の主導者 = 生産者
- 受信データ = 必要なデータ

「ファイルを更新したので添付します」

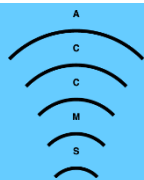
■ TP ≡ web page (にリンクされたファイル)

- データ転送の主導者 = 消費者
- 獲得データ = 必要なデータ ... とは限らない
- 生産者 → 消費者の同期が必要
- 消費者 → 生産者の同期も必要

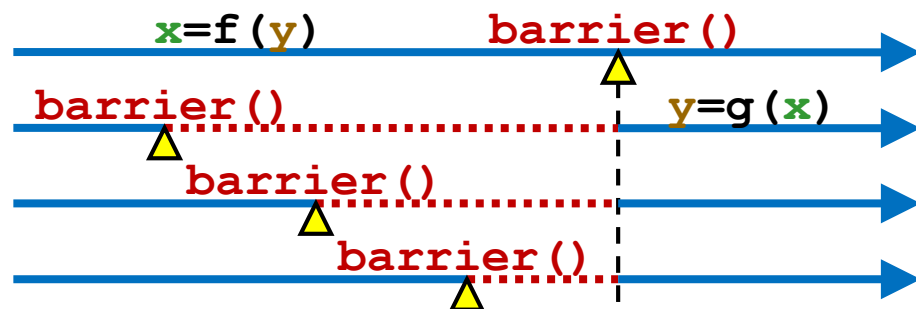
web を見ても
更新の有無は不明

「download したので
更新してもいいよ」

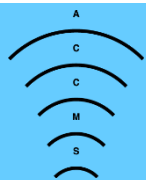
「ファイルを更新したので download してね」or
web に更新日付を記載



- スレッド並列プログラミングでの同期操作の定番
 - 全員がバリア (e.g. ループ終了) に到達するまで待つ
 - バリア前に更新した変数 → バリア後に安全に参照
 - バリア前に参照した変数 → バリア後に安全に更新

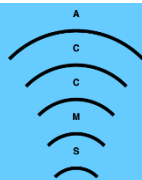


- OpenMP では並列ループの終了時に自動バリア
- プロセス並列でも使用することがある
 - (論理的ではなく) 性能的な歩調合せ
 - ファイル I/O などの通信以外の協調動作のための同期



バッチジョブ (1/3)

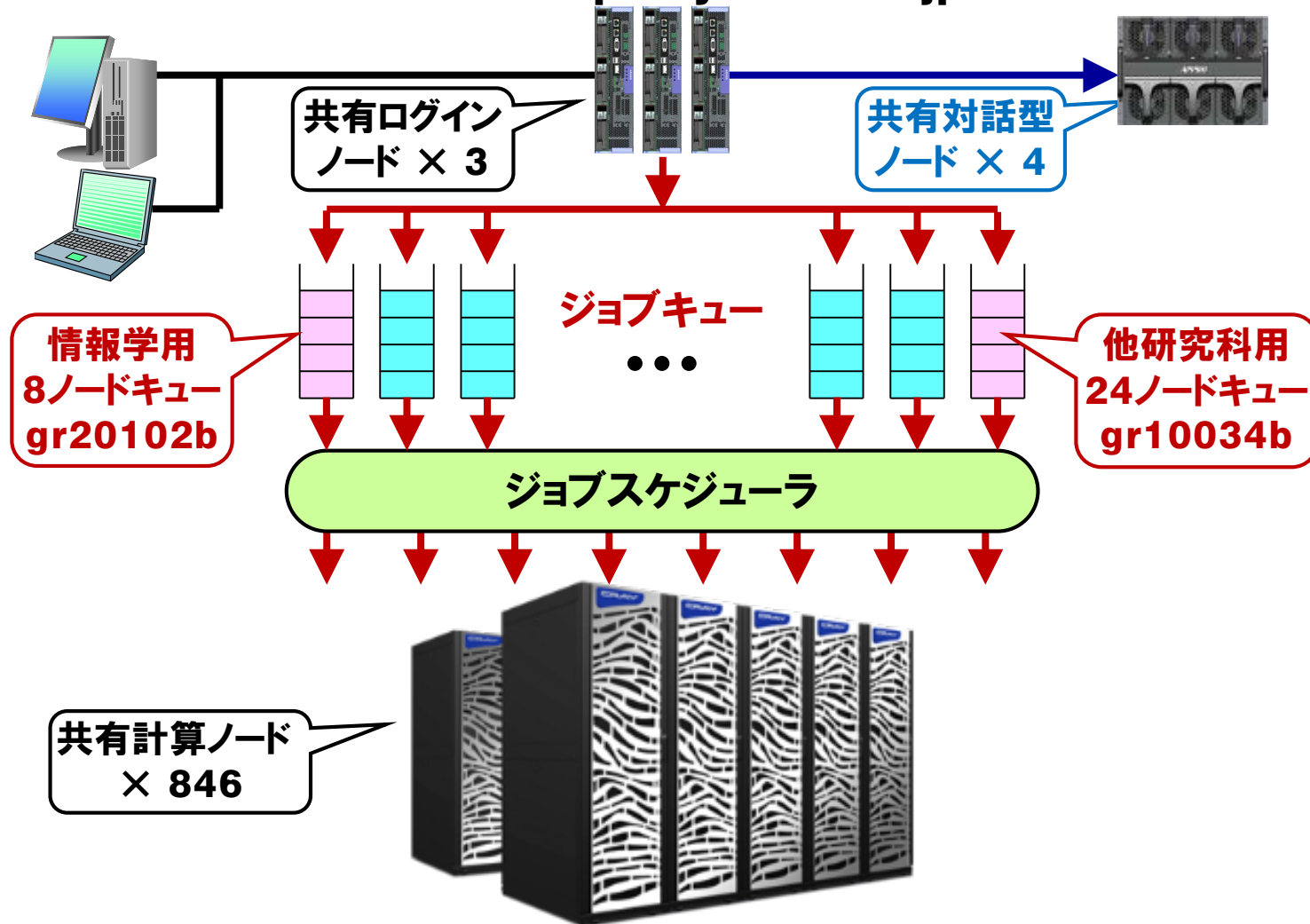
- 普通の実行 = **interactive** (対話的)
 - コマンド入力／クリックでプログラムが直ちに起動
← (普通は) 計算資源を **占有** & **余裕** あり
 - 実行中にプログラムと **対話** (キー入力・クリック)
 - 例外: virus scan, 自動検索, update, ...
- スパコンでの実行 = **batch**
 - **ジョブ** (Prog+Data) 実行を依頼 \doteq **願書郵送**
 - 資源があれば／空けば実行開始 \doteq **担当者開封**
 - (普通は長時間) **非対話的** に実行 \doteq **受験票到着**
← 計算資源は **共有** & 余裕 **僅少**
 - 例外: edit, コンパイル, 小規模テスト, ジョブ投入, ...

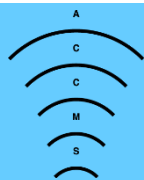


バッチジョブ (2/3)

■ Laurel 2 のバッチジョブスケジューラ

laurel.kudpc.kyoto-u.ac.jp





バッチジョブ (3/3)

■ ジョブの投入

% **qsub jobscript** 

```
#!/bin/bash
```

```
#QSUB -q gr20102b # または gr10034b
```

```
#QSUB -g gr20102 # または gr10034
```

```
#QSUB -A p=1:t=1 # プロセス数=スレッド数=1
```

```
./a.out # プログラム実行
```

■ ジョブの状態確認・削除

% **qstat** または **qs**

% **qdel jobid**

■ 詳細は

<http://web.kudpc.kyoto-u.ac.jp/manual-new/ja/run/systembc>