講義５

MPI-IO
(+MPI)

理化学研究所 AICS
システムソフトウェア研究チーム
堀　敦史

並列プログラミングの基本（MPI）

- 並列プログラムとは
- MPIの基礎
- MPIの基本的な通信

- ネットワークトポロジーと性能
- 片方向通信
- Derived Data Type
- MPI-IO
- MPI の仕様について

並列プログラミングの基本（MPI）

- 並列プログラムとは
- MPIの基礎
- MPIの基本的な通信

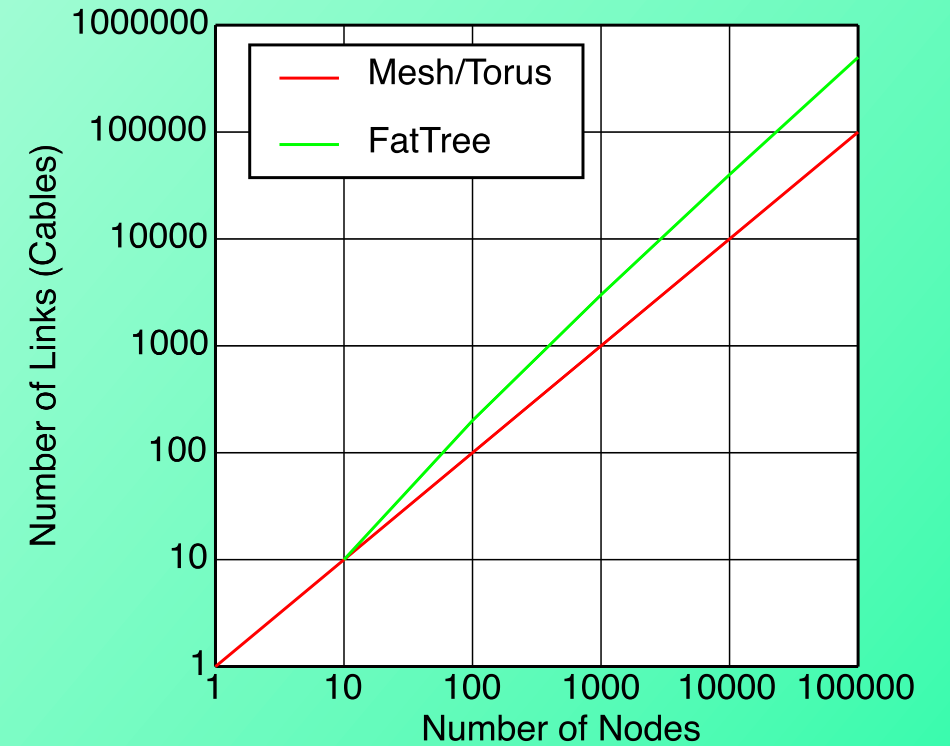
- **ネットワークトポロジーと性能**
- 片方向通信
- Derived Data Type
- MPI-IO
- MPI の仕様について

ネットワークトポロジーと性能

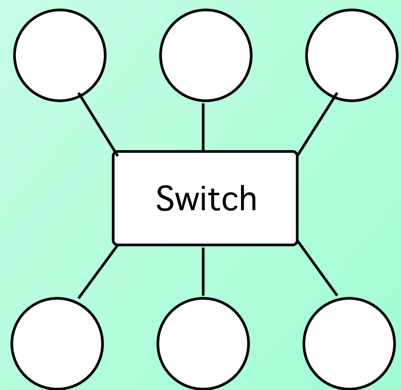
- 並列計算機
  - （計算）ノードがネットワークで結合
- ネットワークトポロジー
  - どのように結合されているか
- 性能との関係
  - ホットスポット：通信が集中する箇所
  - ホットスポットが生じると通信時間が大
  - 特に大型の並列計算機ではホットスポットが生じ易い

代表的なネットワークトポロジー

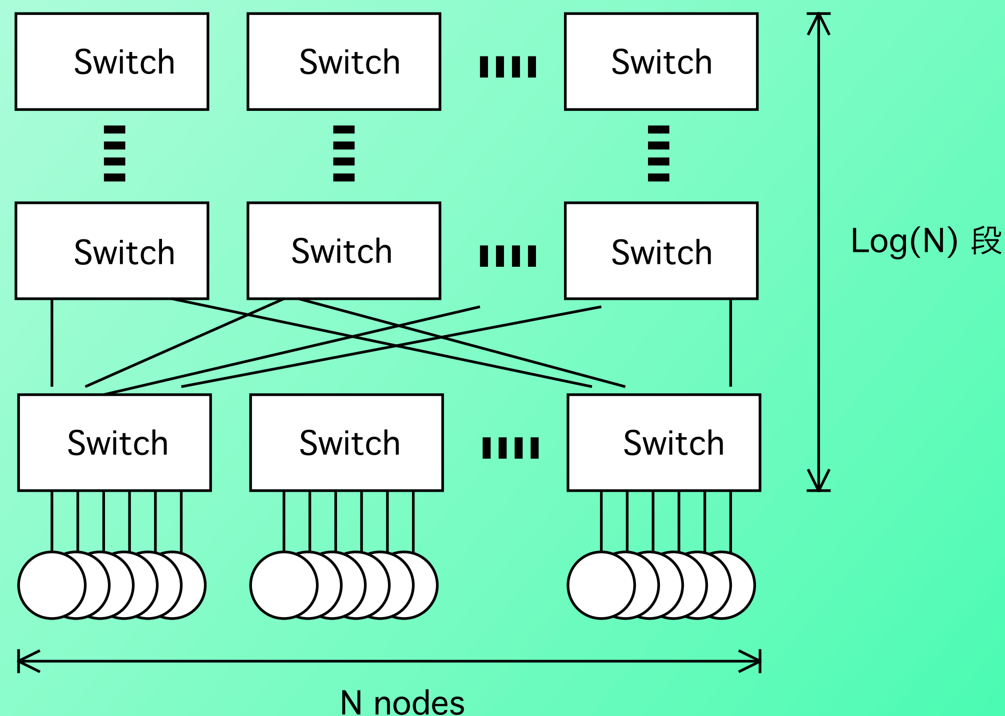
- 大規模なネットワーク
  - 相対的に性能低下
  - ランクマッピングが重要性



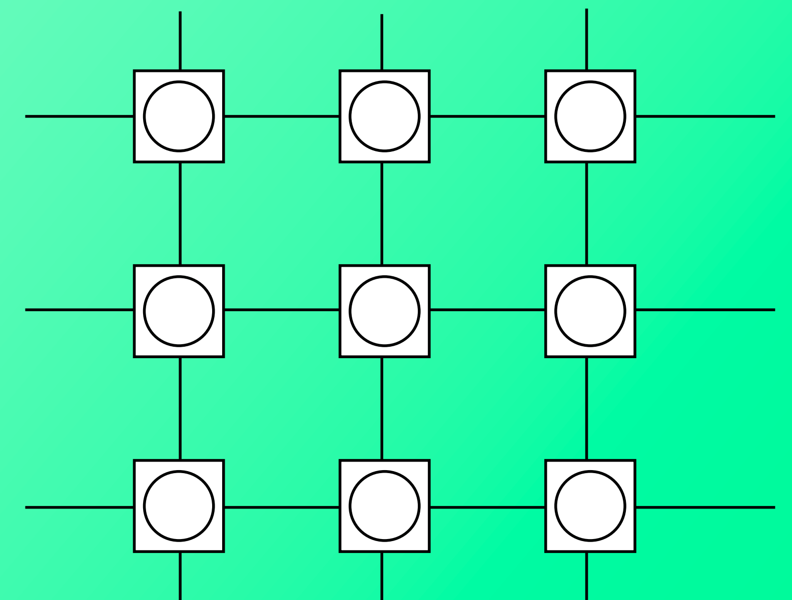
スタートトポロジー
小規模クラスタ



Fat Tree トポロジー
中規模クラスタ

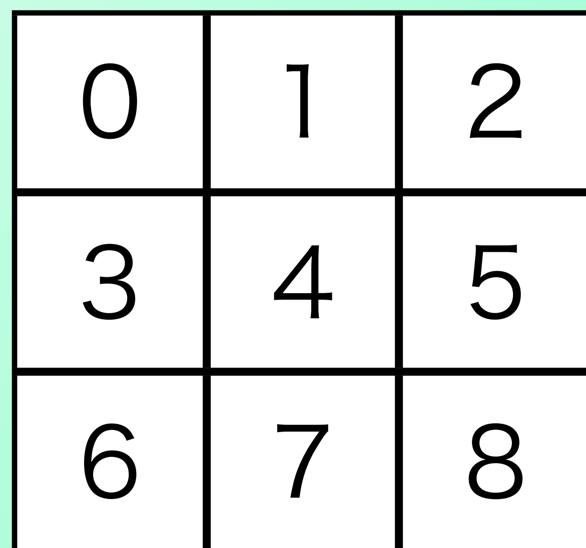


M-D メッシュ／トーラス
トポロジー
大規模クラスタ

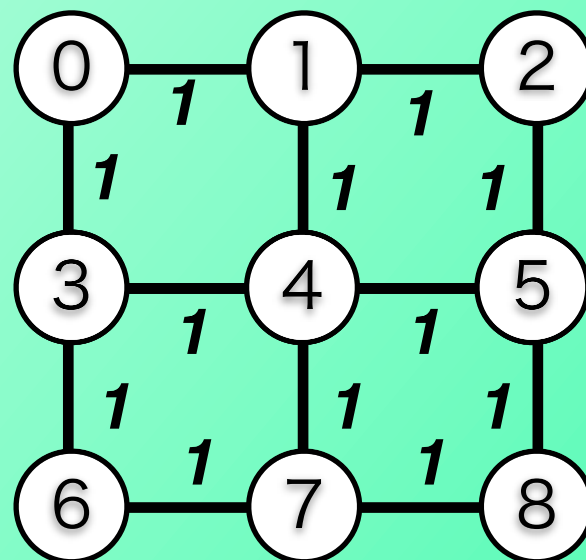


トポロジーとランクマッピング

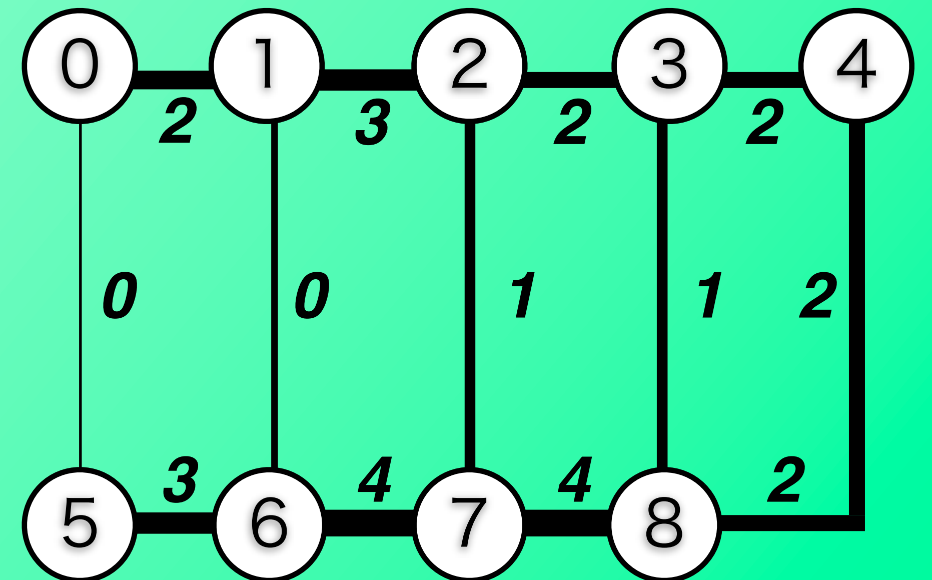
- データの分割とトポロジーのマッピングが通信速度に影響を与える
  - Fat Tree ネットワークでは影響小さい
  - Mesh/Torus では影響大きい
- 特に、京クラスの超大型並列計算機では影響が特に大きい



2次元データの分割
隣接通信のみ



3x3 の
マッピング



4x2 の
マッピング

並列プログラミングの基本（MPI）

- 並列プログラムとは
- MPIの基礎
- MPIの基本的な通信

- ネットワークトポロジーと性能
- **片方向通信**
- Derived Data Type
- MPI-IO
- MPI の仕様について

MPI通信の種別

- 1対1通信 (point-to-point communication)
- 集団通信 (collective communication)
- **片方向通信 (one-sided communication)**

片方向通信

- ここまで通信においては、送信側と受信側でそれぞれ対応する関数を呼ぶ必要があった
- 片方向通信では、**通信の相手の状態に無関係に**他のプロセスのデータにアクセスできる
  - Get　相手のデータを取ってくる
  - Put　相手にデータを書込む
- 特に大きなデータの通信においては rendezvous が不要かつ、ハードウェアの機構を直接用いることができ高速に実行される場合が多い

片方向通信における用語

- **Origin**
  - Get あるいは Put の呼出しをしたプロセス
- **Target**
  - Get あるいは Put の対象となるプロセス
- **Window**
  - Get あるいは Put の対象となるプロセスのメモリ領域
- **Get**
  - Target プロセスの指定したメモリ領域の値を読込む
- **Put**
  - Target プロセスの指定したメモリ領域に値を書込む

Window

```
C: MPI_Win_create( void *base, MPI_Aint size, int unit,
            MPI_Info info, MPI_Comm comm, MPI_Win *win )
F: MPI_WIN_CREATE(  base, size, unit, info, comm, win,
                 ierr )
```

- Collective な操作
- Get/Put でアクセス可能なメモリ領域 win を生成する
- **base, size**　メモリ領域の指定。base から size バイト
- **unit**　Get/Put で指定されるデータの単位（1）
- **info**　ヒント情報（MPI_INFO_NULL）
- **comm**　コミュニケータ
- **win**　生成された window

片方向通信 - Get

```
C: MPI_Get( void *oaddr, int ocount, MPI_Datatype otype, int target,
        MPI_Aint tdisp, int tcount, MPI_Datatype ttype, MPI_Win win )
F:  MPI_GET( oaddr, ocount, otype, target, tdisp, tcount, ttype, win,
                ierr )
```

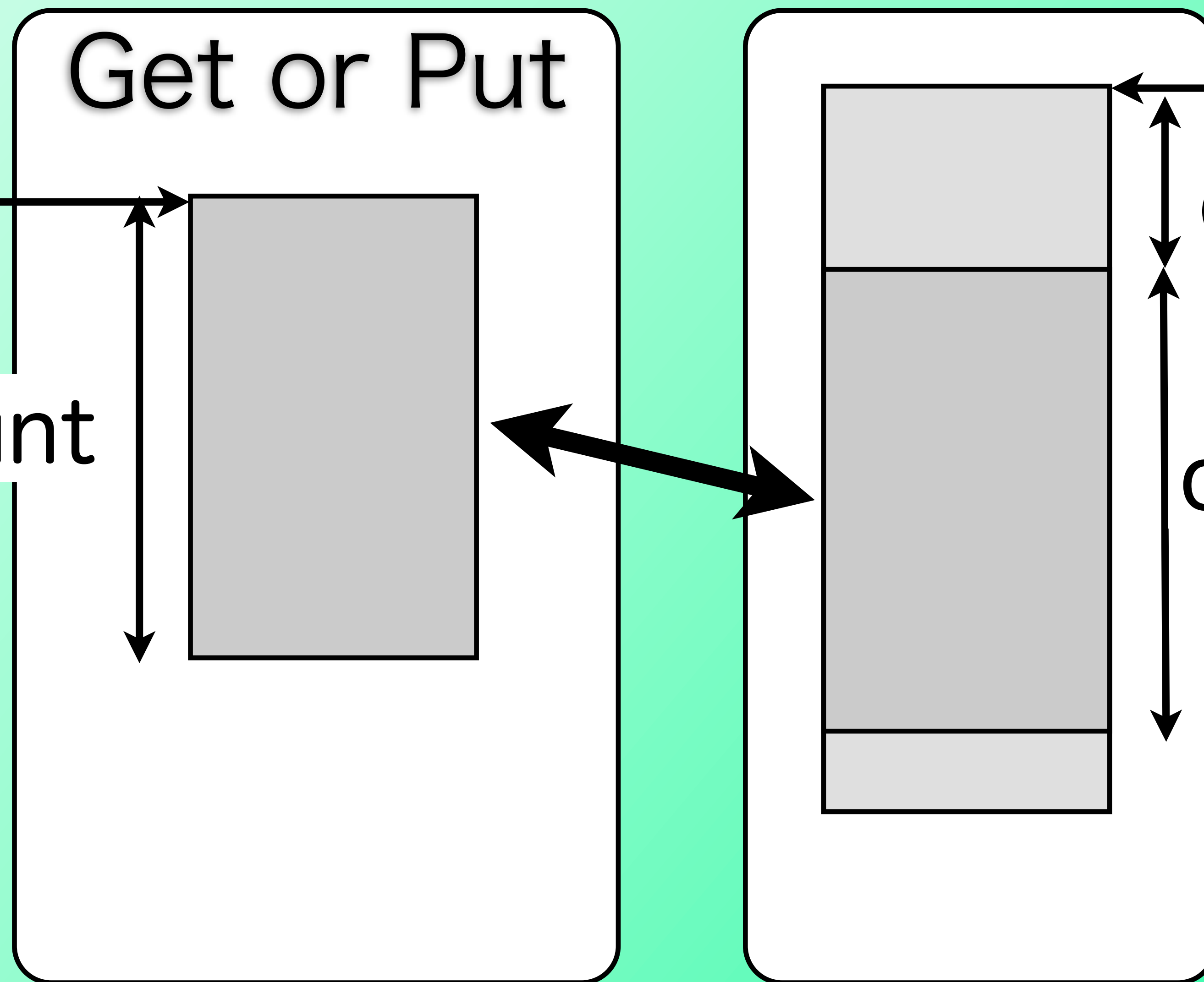
- Target プロセスの win の tdisp オフセットで始まる、長さ tcount の値を、オリジンプロセスの oaddr に読込む
- Target の対象となるアドレス
    - target_addr = window_base + target_disp × disp_unit
    - ここで window_base と disp_unit は Window を生成した時に指定した値

片方向通信 - Put

```
C: MPI_Put( void *oaddr, int ocount, MPI_Datatype otype, int target,
        MPI_Aint tdisp, int tcount, MPI_Datatype ttype, MPI_Win win )
F: MPI_PUT( oaddr, ocount, otype, target, tdisp, tcount, ttype, win,
                ierr )
```

- Target プロセスの win の tdisp オフセットで始まる、長さ tcount の領域に、オリジンプロセスの oaddr の値を書込む
- Target の対象となるアドレス
  - target_addr = window_base + target_disp × disp_unit
  - ここで window_base と disp_unit は Window を生成した時に指定した値

Get and Put



片方向通信 - Accumulate

```
C:  MPI_Accumulate( void *oaddr, int ocount, MPI_Datatype otype,
            int target, MPI_Aint tdisp, int tcount, MPI_Datatype ttype,
            MPI_Op op, MPI_Win win )
F:  MPI_ACCUMULATE( oaddr, ocount, otype, target, tdisp,
            tcount, ttype, op, win, ierr )
```

- Target プロセスの win の tdisp オフセットで始まる、長さ tcount の領域に、そこの値と、オリジンプロセスの oaddr の値を、Op で指定された操作をおこなった結果を書込む
- Op：MPI_Reduce 等で指定した操作と同じ
- Target の対象となるアドレス
  - Get や Put と同じ

片方向通信の同期

```
C:  MPI_Win_fence( MPI_WIN_FENCE, MPI_Win win )
F:  MPI_WIN_FENCE( MPI_WIN_FENCE, win, ierr )
```

- Collective な操作
- この呼出しの前に行われた全ての片方向通信の終了を待つ

- 他にも MPI_Win_lock 等いくつかの同期の手段が提供されている

片方向通信の Tips

- 高速な片方向通信のために
  - Derived datatype（後述）は使わない
    - ハードウェアの機構が使えなくなる
  - 小さいデータの Get/Put はあまり速くない
- MPI の実装や機種依存の部分が大きいので注意すること

並列プログラミングの基本（MPI）

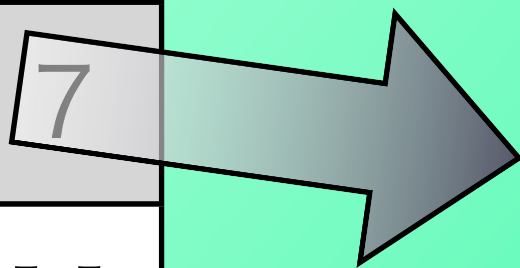
- 並列プログラムとは
- MPIの基礎
- MPIの基本的な通信

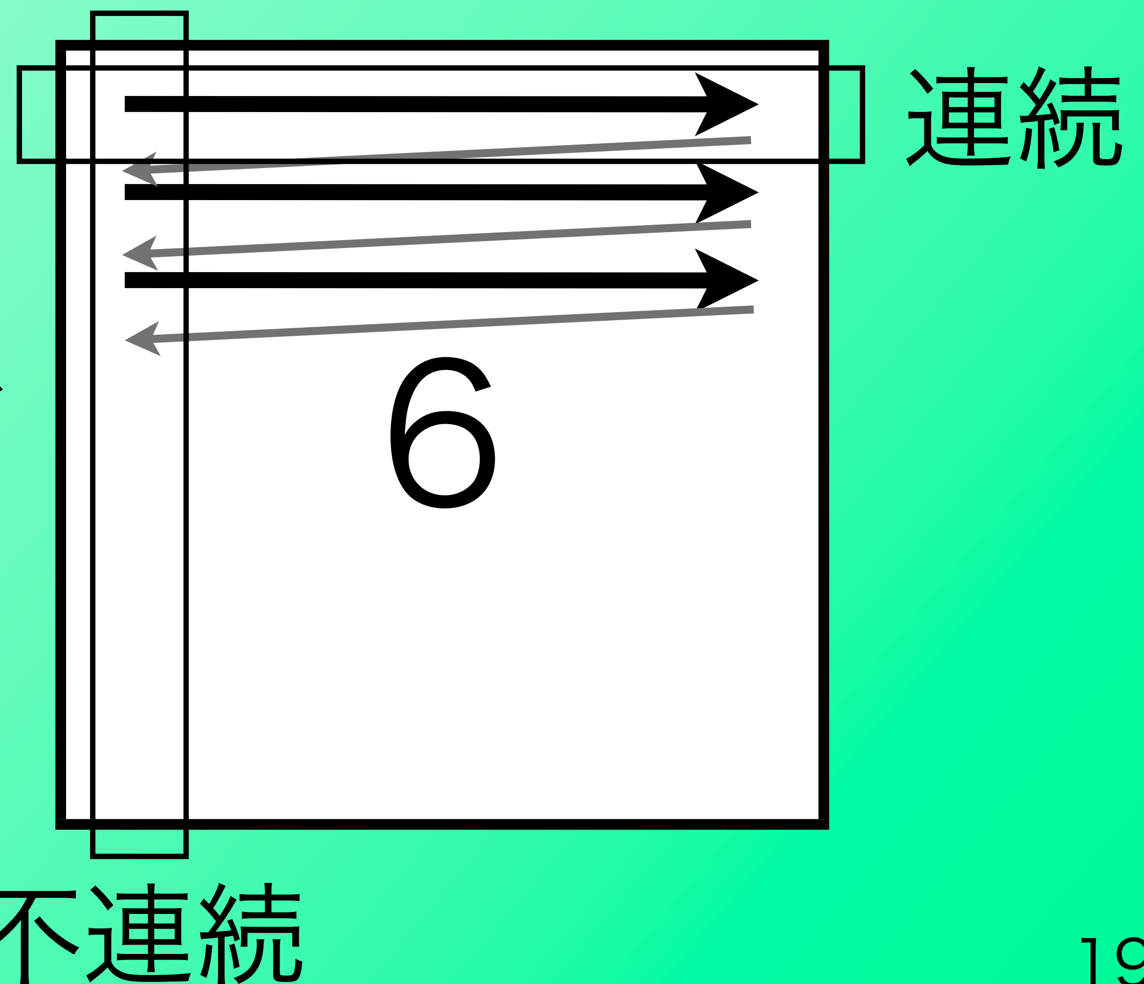
- ネットワークトポロジーと性能
- 片方向通信
- **Derived Data Type**
- MPI-IO
- MPI の仕様について

例：多次元配列のブロック分割

- 多次元配列の場合、あるひとつの次元のみデータの並びが連続である ⇒ 他の次元の並びは不連続
- 例えば、ステンシル計算において隣のブロックと halo 領域を交換しようとすると、不連続なデータの通信が発生する

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15





Derived Datatype

- 派生データ型（Derived Datatype）
- 不連続なデータを、ひとまとめに表現することで、不連続なデータの通信の高速化への対処（MPI の実装）を可能とする
- Basic (Predefined) Datatype と（配列内の）オフセットの並びで表現される

MPIにおけるデータ型 (復習)

C言語のデータ型との対応

MPI datatype	C datatype
MPI_CHAR	`char` (treated as printable character)
MPI_SHORT	`signed short int`
MPI_INT	`signed int`
MPI_LONG	`signed long int`
MPI_LONG_LONG_INT	`signed long long int`
MPI_LONG_LONG (as a synonym)	`signed long long int`
MPI_SIGNED_CHAR	`signed char` (treated as integral value)
MPI_UNSIGNED_CHAR	`unsigned char` (treated as integral value)
MPI_UNSIGNED_SHORT	`unsigned short int`
MPI_UNSIGNED	`unsigned int`
MPI_UNSIGNED_LONG	`unsigned long int`
MPI_UNSIGNED_LONG_LONG	`unsigned long long int`
MPI_FLOAT	`float`
MPI_DOUBLE	`double`
MPI_LONG_DOUBLE	`long double`
MPI_WCHAR	`wchar_t` (defined in `<stddef.h>`) (treated as printable character)
MPI_C_BOOL	`_Bool`
MPI_INT8_T	`int8_t`
MPI_INT16_T	`int16_t`
MPI_INT32_T	`int32_t`
MPI_INT64_T	`int64_t`
MPI_UINT8_T	`uint8_t`
MPI_UINT16_T	`uint16_t`
MPI_UINT32_T	`uint32_t`
MPI_UINT64_T	`uint64_t`
MPI_C_COMPLEX	`float _Complex`
MPI_C_FLOAT_COMPLEX (as a synonym)	`float _Complex`
MPI_C_DOUBLE_COMPLEX	`double _Complex`
MPI_C_LONG_DOUBLE_COMPLEX	`long double _Complex`
MPI_BYTE	
MPI_PACKED	

FORTRAN言語のデータ型との対応

MPI datatype	Fortran datatype
MPI_INTEGER	`INTEGER`
MPI_REAL	`REAL`
MPI_DOUBLE_PRECISION	`DOUBLE PRECISION`
MPI_COMPLEX	`COMPLEX`
MPI_LOGICAL	`LOGICAL`
MPI_CHARACTER	`CHARACTER(1)`
MPI_BYTE	
MPI_PACKED	

C言語とFORTRAN言語両方に対応するデータ型

MPI datatype	C datatype	Fortran datatype
MPI_AINT	`MPI_Aint`	`INTEGER (KIND=MPI_ADDRESS_KIND)`
MPI_OFFSET	`MPI_Offset`	`INTEGER (KIND=MPI_OFFSET_KIND)`

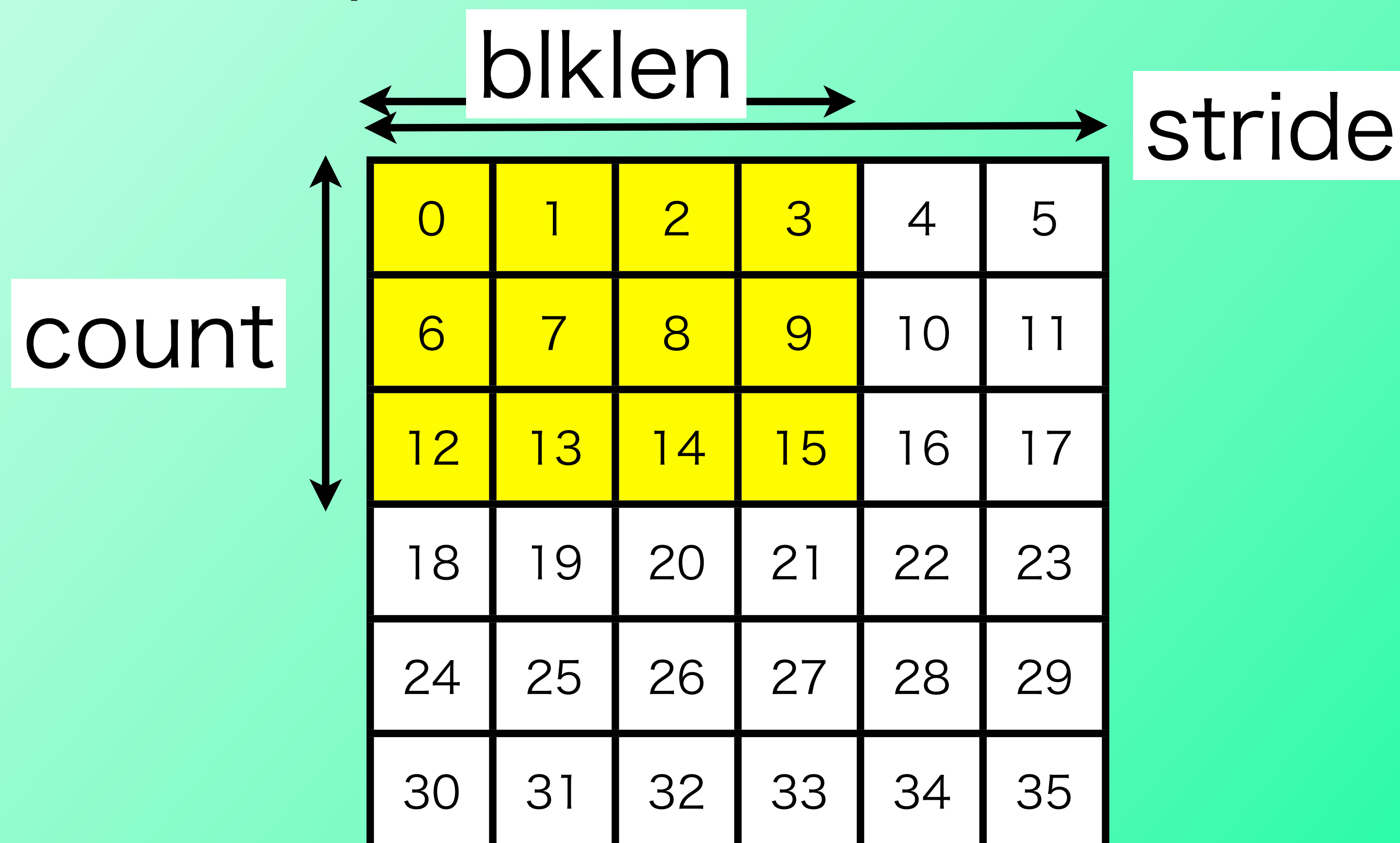
Derived Datatype の定義

- 以下の関数が用意されている
  - MPI_Type_contiguous
  - MPI_Type_vector
  - MPI_Type_indexed
  - MPI_Type_create_indexed_block
  - MPI_Type_create_subarray
  - MPI_Type_create_darray など
- 定義された「派生データ型」は commit して初めて使うことができる
  - **MPI_Type_commit <<<< 忘れないように！**

MPI_Type_vector

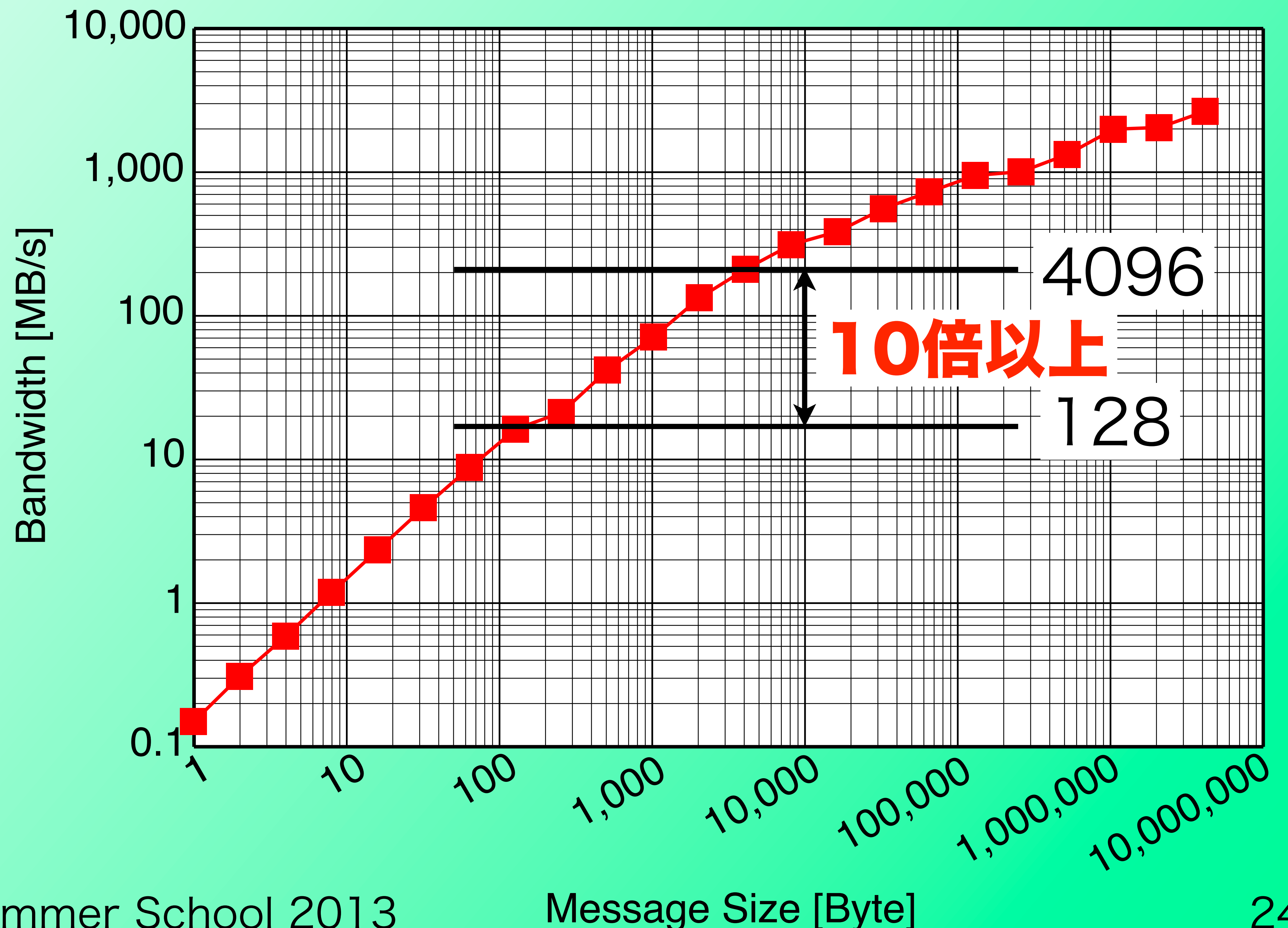
```
C:    MPI_Type_vector( int count, int blklen, in stride,
          MPI_Datatype otype, MPI_Datatype *ntype )
F:    MPI_TYPE_VECTOR( count, blklen, stride,
          otype, ntype, ierr )
```

MPI_type_vector(3, 4, 6, MPI_DOUBLE, &newtype)



メッセージ通信の特性

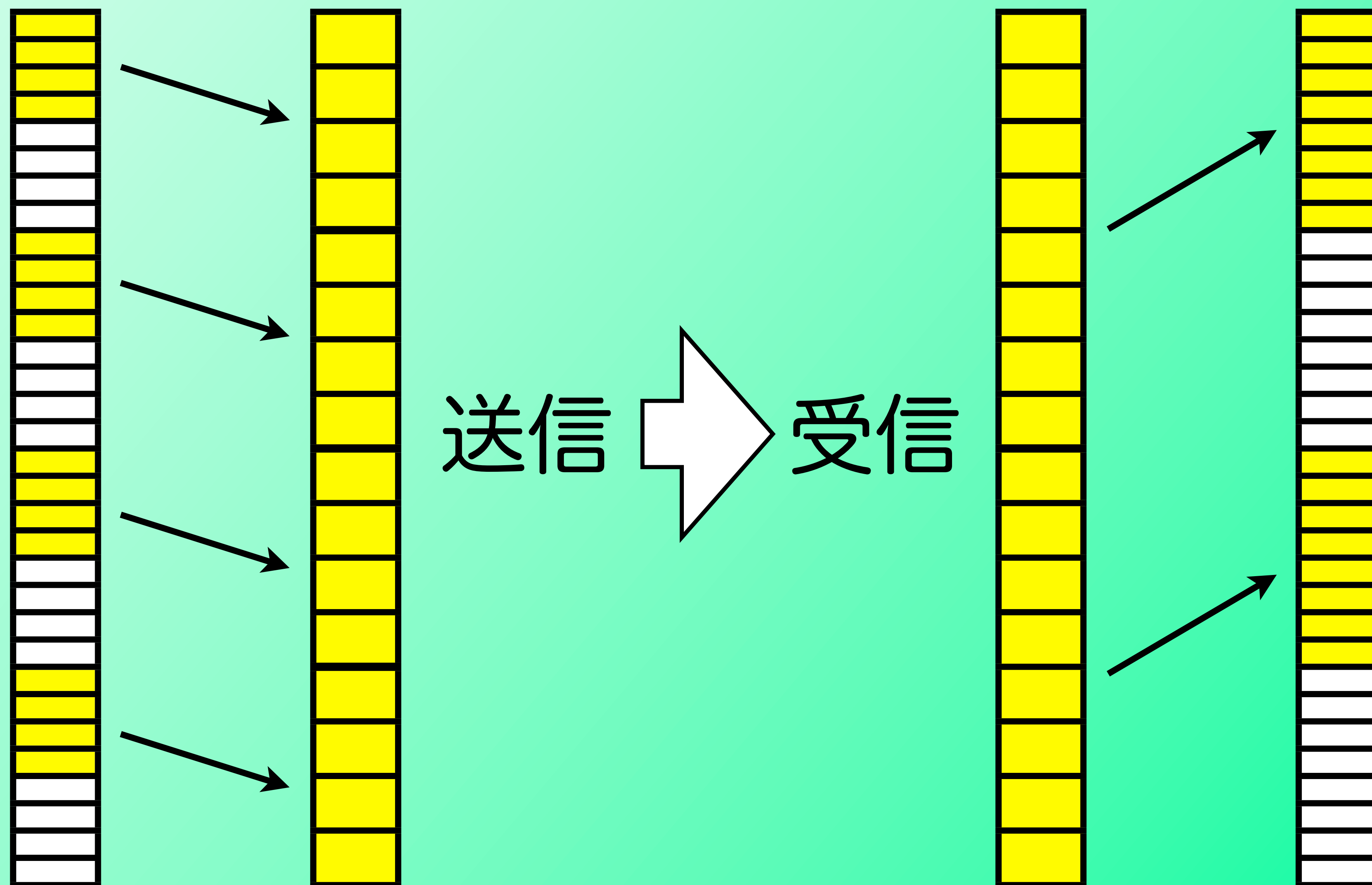
- 128バイトを10回送る（1,280バイト）より4,096バイトを１回で送った方が速い



Intel Nehalem
(2.67 GHz)
Infiniband QDR
MPICH-SCore

Derived Datatype の内部処理

- 不連続なメモリ領域をいったん連続領域に “pack” して、それを送信し、受信側では連続領域をバラバラ（“unpack”）にする。

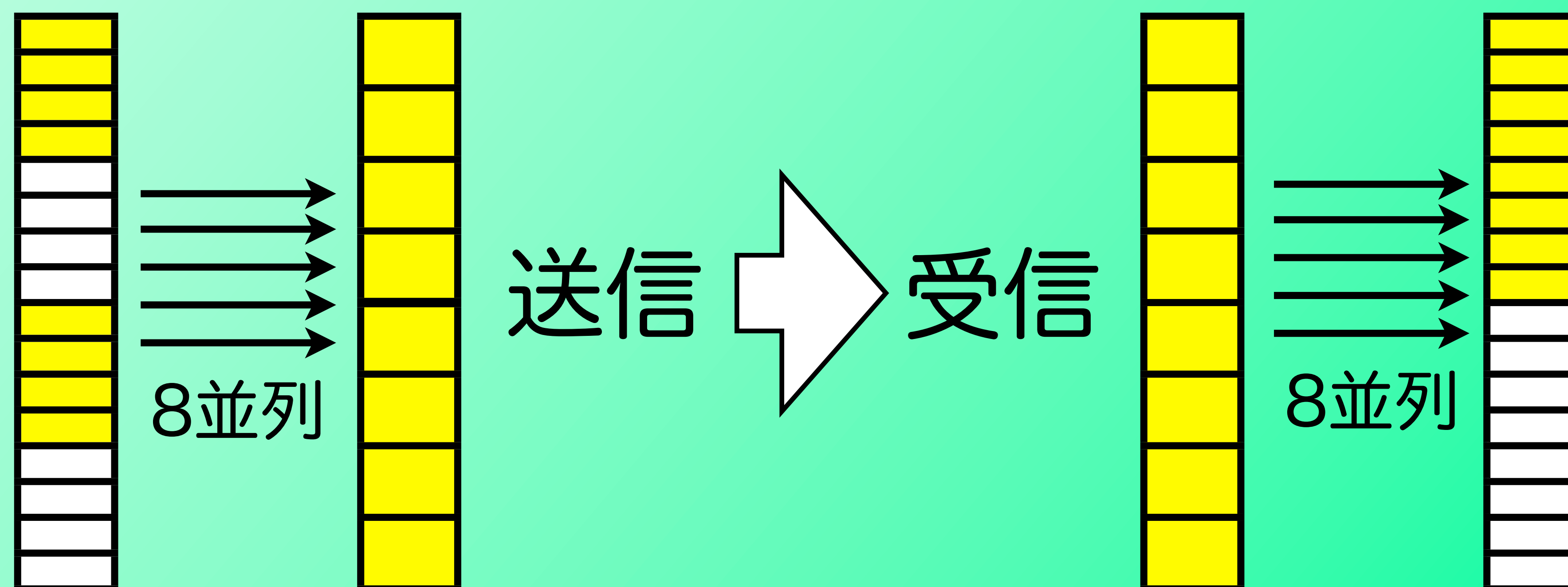


Derived Datatype のまとめと注意

- 基本データ型から派生データ型を生成
- 生成した後で commit すること
- 作られたデータ型は、MPI の中の通信や IO 等で使うことができる
- 不連続なメモリ領域をひとまとめに処理できる
  - **ただし、これで実際に「通信が高速」になるかどうかは MPI の実装や、機種に依存する**
  - **一方、MPI-IO の多くの場合は、派生データ型を用いることで高速化が可能**

「京」の場合

- 「京」では一般的に **Hybrid MPI** が用いられている
  - ノード内の8コアは、コンパイラの自動並列化あるいはOpenMP で並列化される
  - 逆に MPI の呼出は逐次処理になる
- Derived Datatype の pack/unpack は並列化されないので1コアでしか動作しない
- Pack/unpack するプログラムを**陽に**（例えば OpenMP）書いてノード内並列化した方が速い（場合が多い）



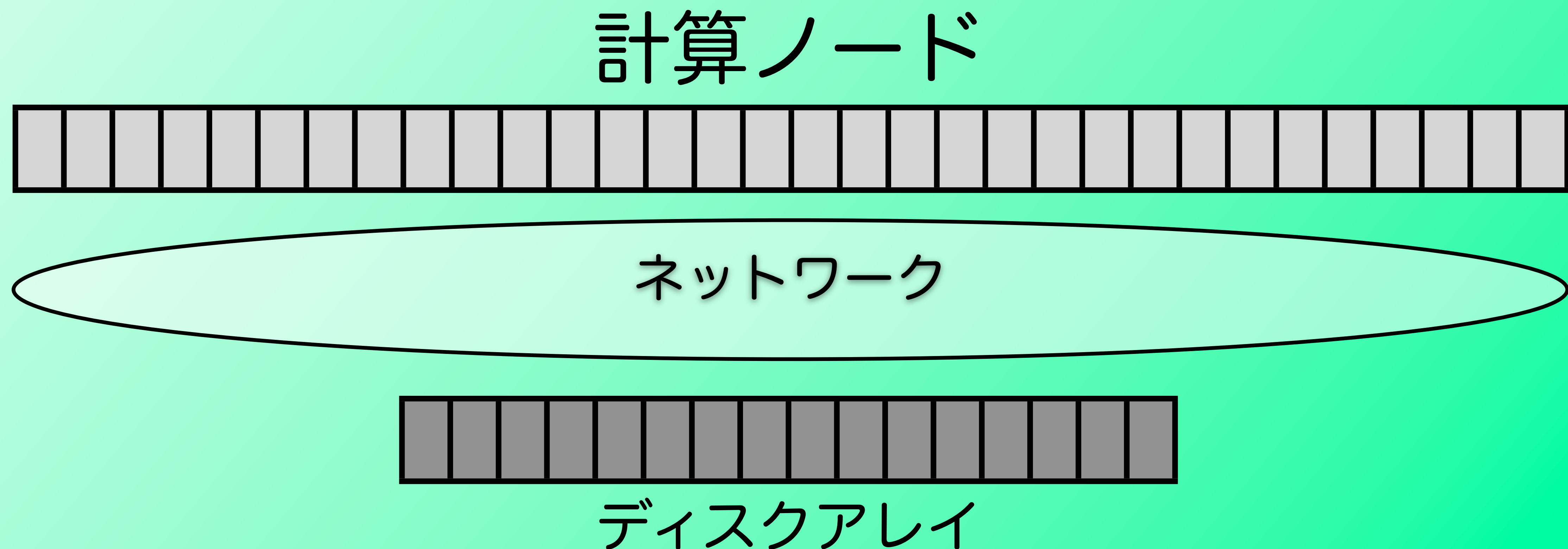
並列プログラミングの基本（MPI）

- 並列プログラムとは
- MPIの基礎
- MPIの基本的な通信

- ネットワークトポロジーと性能
- 片方向通信
- Derived Data Type
- **MPI-IO**
- MPI の仕様について

並列ファイルシステムの構成

- 並列ファイルシステムでは、ネットワークを経由してファイルにアクセスする
- 通信の特性として、小さいメッセージは遅いので、結果として、小さい単位でのファイル IO は遅くなる



MPI-IO のソフトウェア構造

- MPI-IO は（並列）ファイルシステムを MPI という枠組みで抽象化したもの
- 性能向上のためにヒント情報を渡すことができるようになっている
  - MPI-IO の実装に依存したパラメータ
  - 機種やファイルシステム依存したパラメータ
- MPI-IO を用いた並列 IO ライブラリも存在する
  - Parallel netCDF　http://trac.mcs.anl.gov/projects/parallel-netcdf/

MPI-IO - open と close

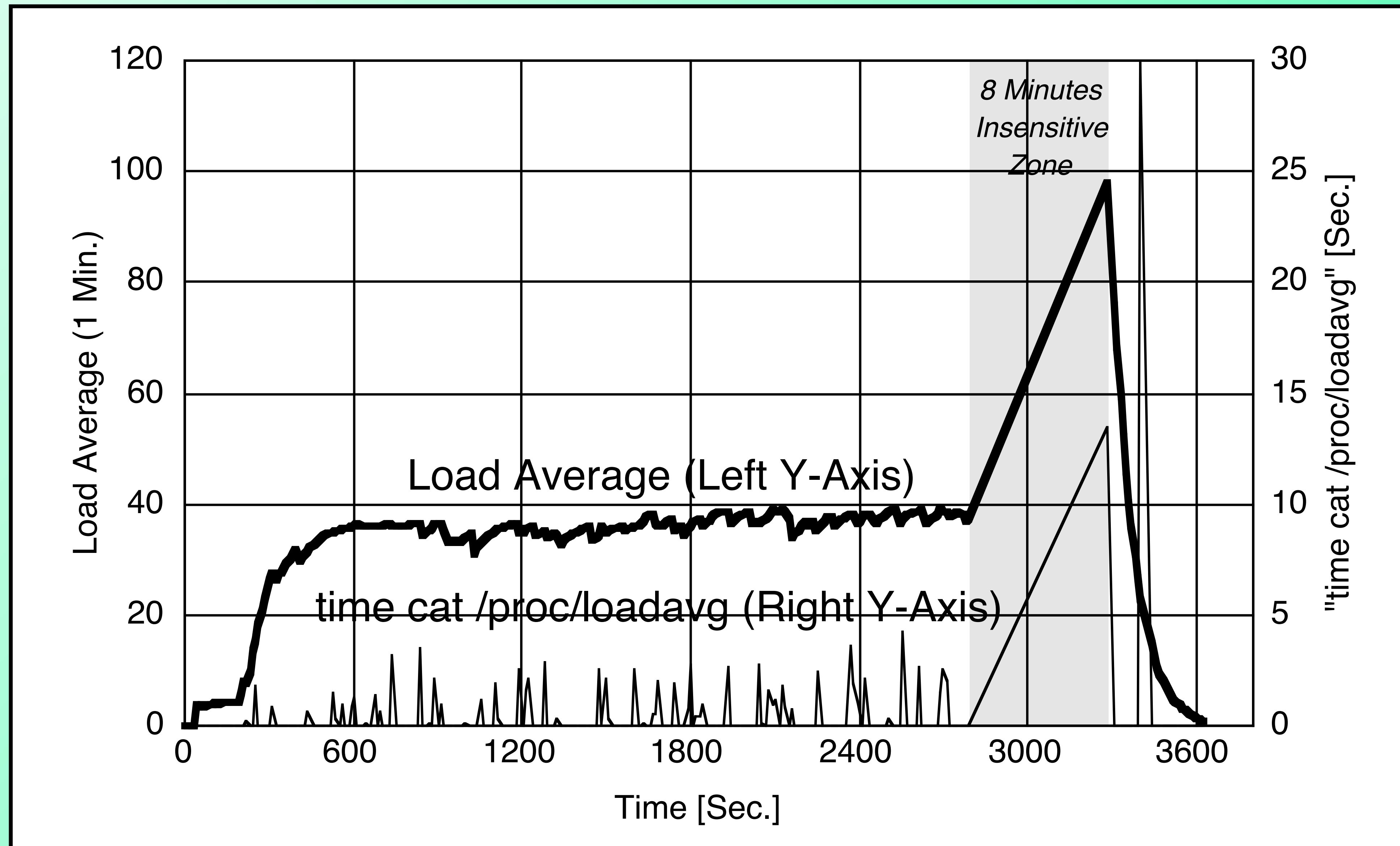
```
C: MPI_File_open( MPI_Comm comm, char *filename,
                  int amode, MPI_Inof info, MPI_File *fh)
   MPI_File_close( MPI_File *fh )
F: MPI_FILE_OPEN( comm, filename, amode, info, fh, ierr )
   MPI_FILE_CLOSE( fh, ierr )
```

- 集団呼出し（コミュニケータ comm に属する全てのプロセスで同じように呼び出す必要がある）

- **amode**：MPI_MODE_RDONLY, MPI_MODE_RDWR, MPI_MODE_WRONLY など
- **info**：ファイルシステムへのヒント（MPI_INFO_NULL)
- **fh**：ファイルハンドル - これに対しファイルを操作する

並列ファイルシステムの必要性

- よく使われている NFS では、まったく性能が出ない！



A timeline of NFS server load average when 64 processes (4 nodes) are creating 1GB file

並列ファイルシステム

- 計算ノード数に（ほぼ）比例する IO バンド幅を提供する
- 普通のディスクのバンド幅は 60 MB/s 前後
  - 普通の SSD で 100 MB/s 前後
- これより大きなバンド幅は、複数のディスクを高速ネットワークで束ねて実現している
  - **小さな単位のアクセスでは性能が全く出ない！**
- MPI-IO ではさまざまな工夫がある

ファイルの IO

```
C:  MPI_File_read( MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)
    MPI_File_read_all( MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)
    MPI_File_read_at( MPI_File fh, MPI_Offset off, void *buf, int count, MPI_Datatype type,
                      MPI_Status *status)
    MPI_File_read_at_all( MPI_File fh, MPI_Offset off, void *buf, int count, MPI_Datatype type,
                      MPI_Status *status)

    MPI_File_write( MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)
    MPI_File_write_all( MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)
    MPI_File_write_at( MPI_File fh, MPI_Offset off, void *buf, int count, MPI_Datatype type,
                      MPI_Status *status)
    MPI_File_write_at_all( MPI_File fh, MPI_Offset off, void *buf, int count, MPI_Datatype type,
                      MPI_Status *status)

F:  MPI_FILE_READ( fh, buf, count, datatype, status, ierr )
    MPI_FILE_READ_ALL( fh, buf, count, datatype, status, ierr )
    MPI_FILE_READ_AT( fh, offset, buf, count, datatype, status, ierr )
    MPI_FILE_READ_AT_ALL( fh, offset, buf, count, datatype, status, ierr )

    MPI_FILE_WRITE( fh, buf, count, datatype, status, ierr )
    MPI_FILE_WRITE_ALL( fh, buf, count, datatype, status, ierr )
    MPI_FILE_WRITE_AT( fh, offset, buf, count, datatype, status, ierr )
    MPI_FILE_WRITE_AT_ALL( fh, offset, buf, count, datatype, status, ierr )
```

MPI_Status

```
C:  MPI_Get_count( MPI_Status status, MPI_Datatype type, int *count )
F:  MPI_GET_COUNT( status, type, count, ierr )
```

- MPI_Status
  - 1対1通信における受信
  - MPI-IO におけるデータの read/write
- MPI_Status に対し、MPI_GET_COUNT を呼ぶと、その status を返した操作におけるデータの個数が返る
  - 1対1通信の受信：受信したデータの個数
  - MPI-IO：アクセスしたデータの個数

ファイルアクセス 関数

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking & split collective	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking & split collective	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking & split collective	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

MPI-IO におけるファイルのアクセス

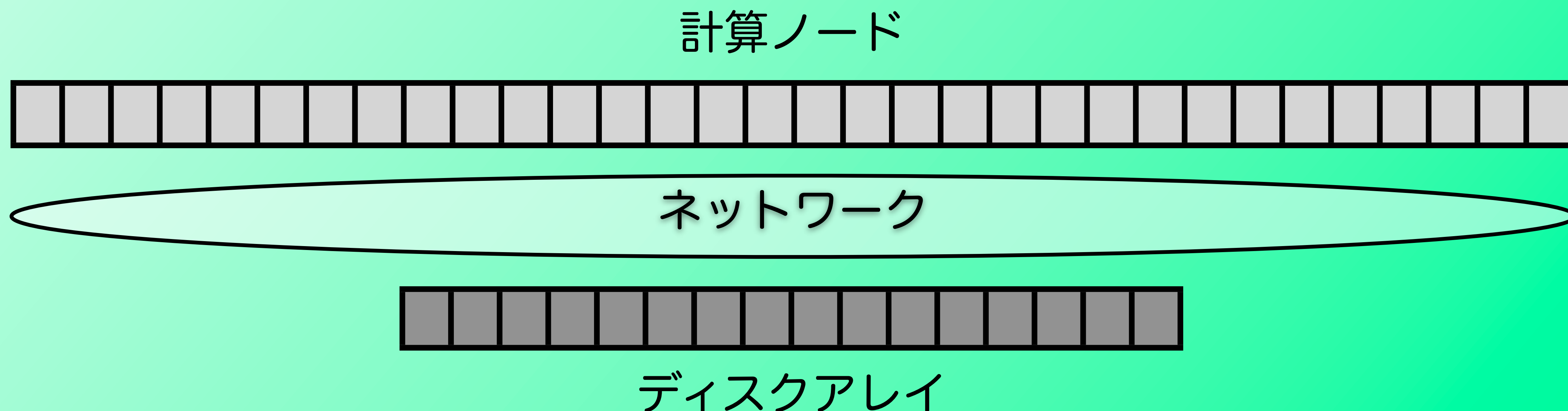
- Collective IO
  - 集団 IO
- File Pointer
  - アクセスする場所の指定
  - File View
    - 個々のプロセスへの割り付け
- (Non) Blocking IO
  - IO と計算のオーバーラップ

ファイルアクセス 関数

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking & split collective	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking & split collective	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking & split collective	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

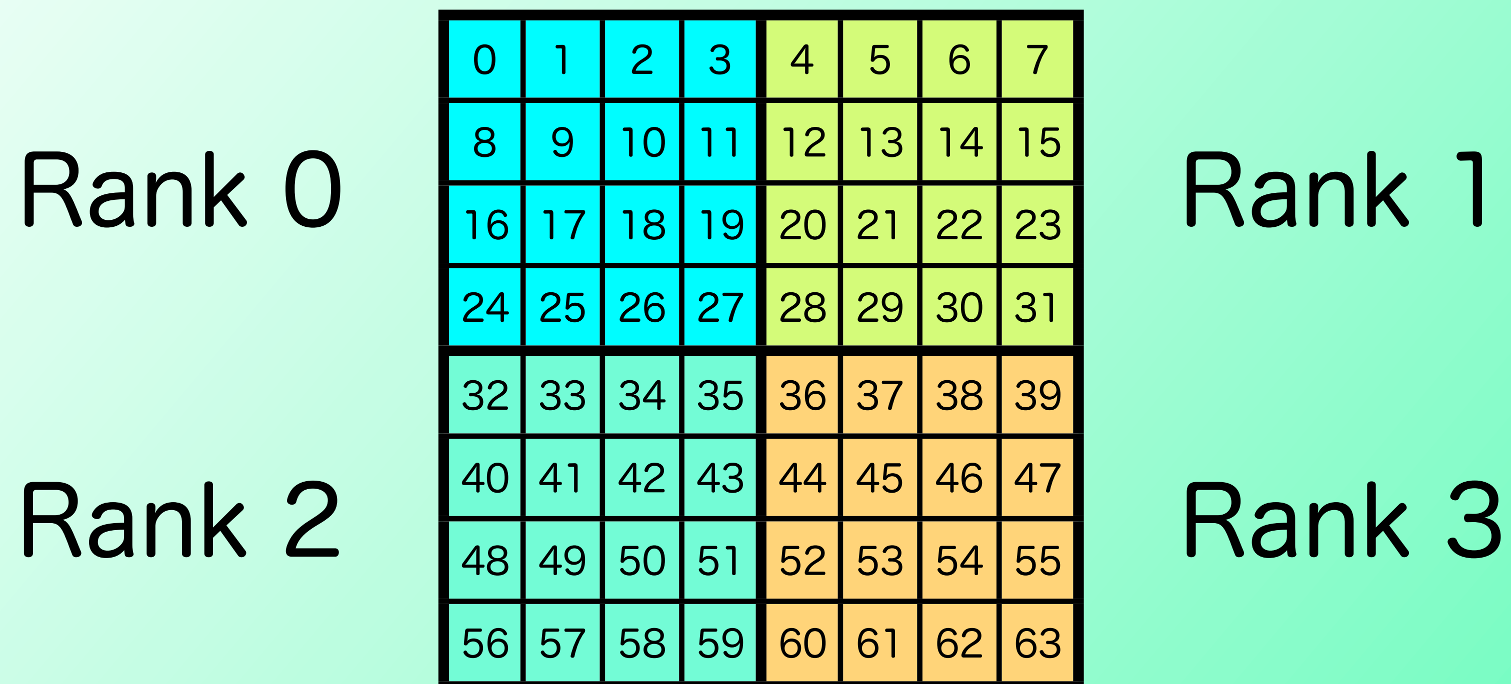
並列ファイルシステムの特性と並列プログラムの特性

- 並列ファイルシステムの特性
  - 複数のディスクを同時に使うことでバンド幅を稼ぐ
  - 複数のディスクを同時に使うほど、大きな単位でアクセスしないと、バンド幅がでない
- 並列プログラムの特性
  - **Weak Scaling**　　ノード数に比例して問題サイズが大きくなる
  - **Strong Scaling**　　ノード数に依らず問題サイズが一定
    - ノード数が大きくなるとノードあたりのデータは小さくなる
    - IO の性能が悪くなる



配列の block 分割の場合

Rank 0



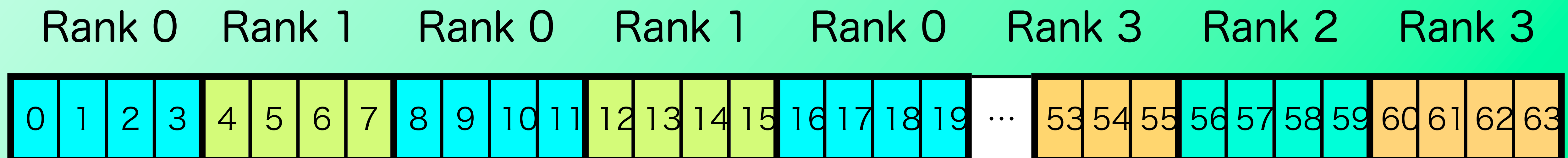
Rank 1

Rank 2

Rank 3

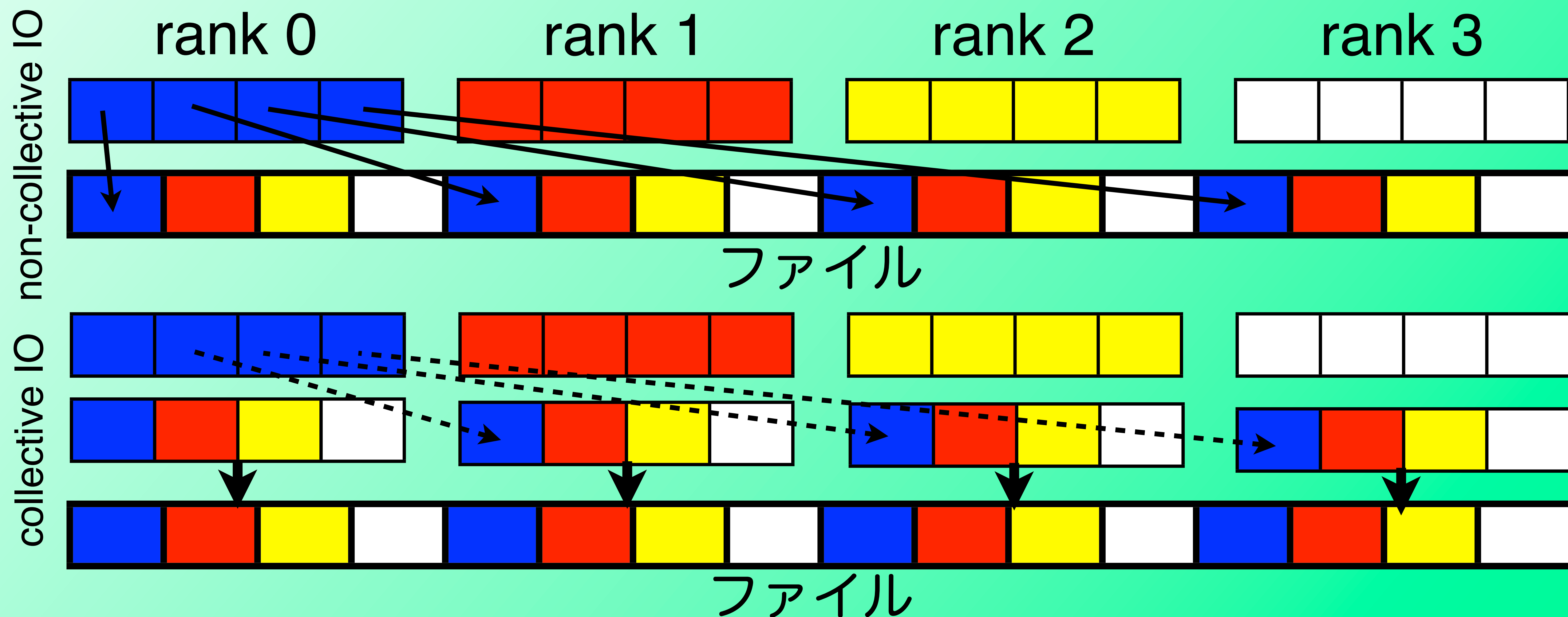
Rank数が大きくなる程、配列の次元数が大きくなる程、細かい IO になる

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



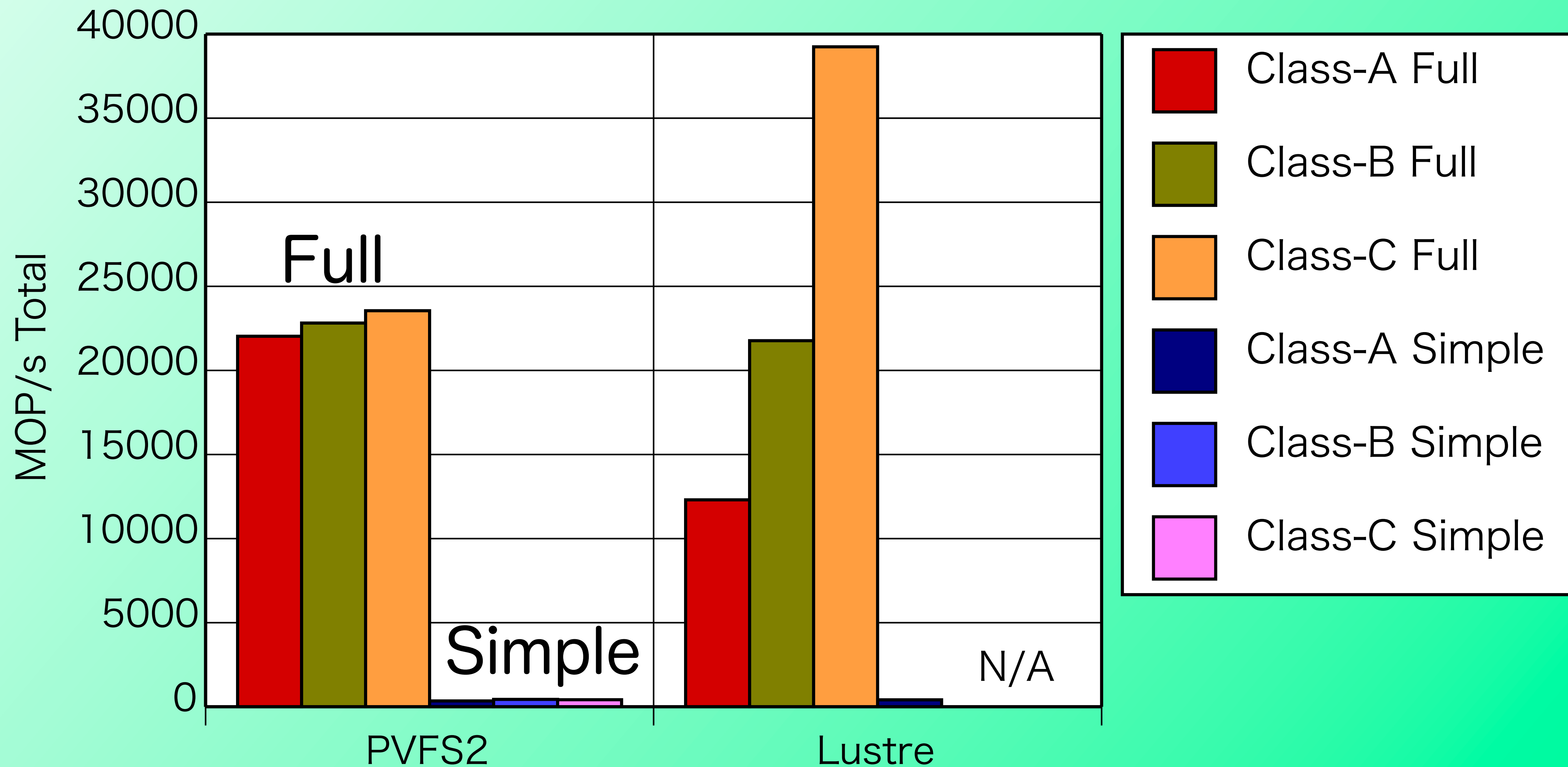
Collective IO

- 全てのノードが同時に IO することで、細かいデータをとりまとめることができ（MPI の通信）、結果として IO のデータ量を大きくすることができる。Derived Datatype も同時に使い、１回の IO 呼出でより多くの範囲のデータを指定することが大事
- 同時に、ファイルにアクセスする場所を連続にすることが可能
- 結果として、IO 性能が大きく向上する



MPI-IO ベンチマーク

- BT-IO： MPI-IO ベンチマークプログラム
  - Class： 問題の大きさ（A < B < C)
  - Full： Colletive IO
  - Simple： Non-Collective IO



コミュニケータを指定しない Collective な操作

- 片方向通信の Window や MPI-IO のファイルハンドル
- それらの生成時にコミュニケータを指定しているから
  - 生成時に指定されたコミュニケータを MPI_COMM_DUP して内部に持っている

ファイルアクセス 関数

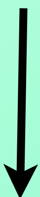
positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking & split collective	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking & split collective	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking & split collective	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

File Pointer 問題

- 逐次プログラムの IO では、file pointer（アクセスする位置）がひとつだけ（暗黙に）ある
- 並列プログラムでは、複数のプロセスが同時に書込むために、file pointer の扱いに注意する必要がある

逐次

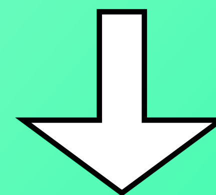
write(“abc”)
write(“def”)



a	b	c	d	e	f		

並列

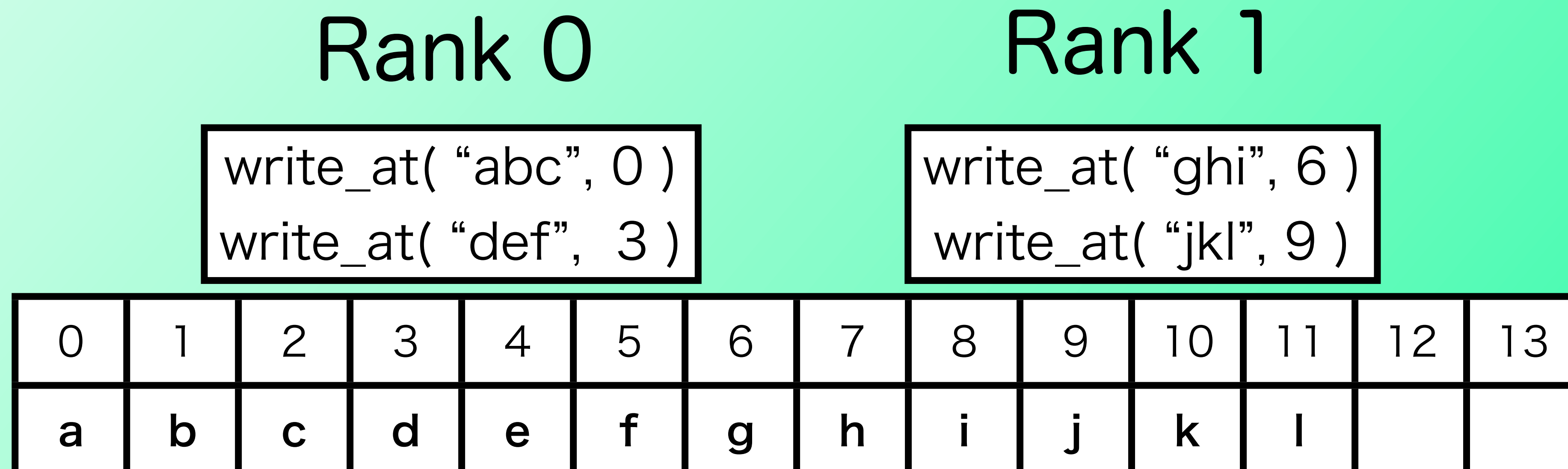
Rank 0	Rank 1	Rank 2	Rank 3
write(“abc”)	write(“ghi”)	write(“mno”)	write(“stu”)
write(“def”)	write(“jkl”)	write(“pqr”)	write(“vwx”)

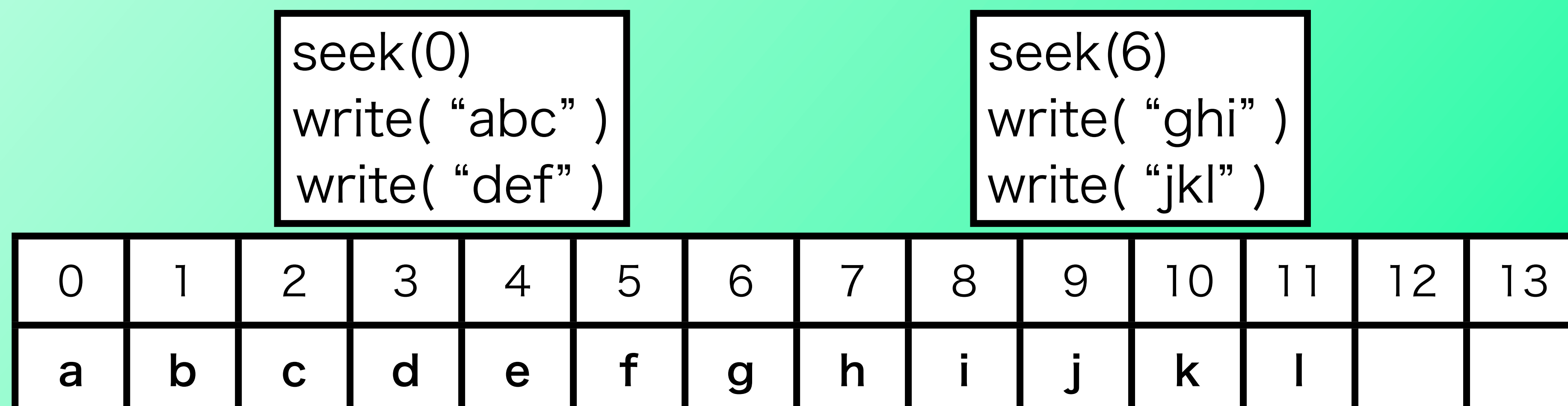


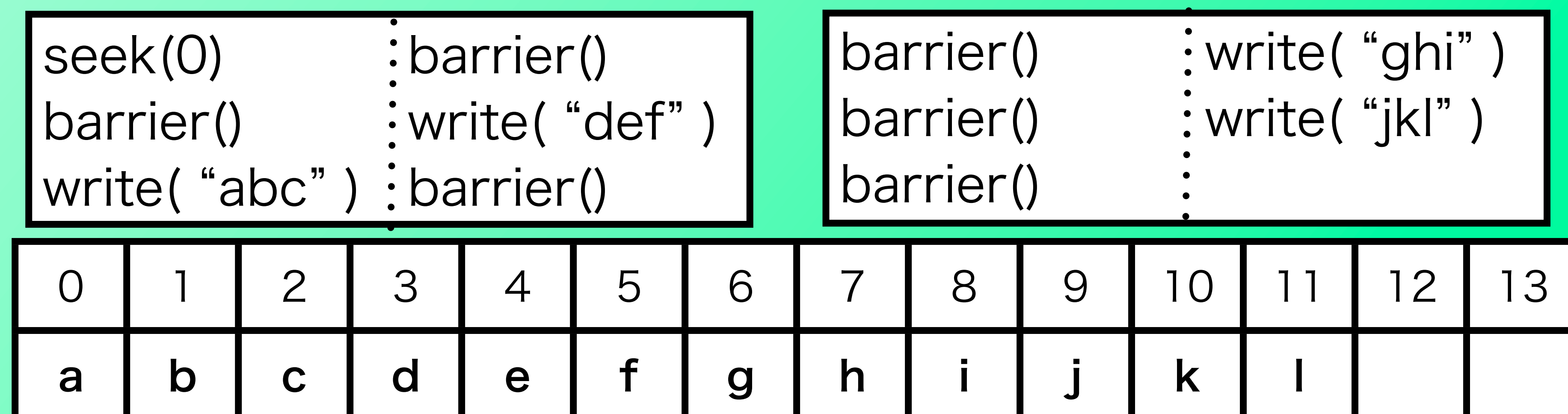
a	b	c	d	e	f	g	h	i	j	k	[illegible]	[illegible]	[illegible]	o	p	q	r	s	t	u	v	w	x	

アクセスする場所の指定の仕方（3種類）

1. FP を使わないで、IO 時に常に場所を指定する
2. プロセス毎に FP が存在する
3. ファイルにひとつだけ FP が存在する
   ⇒ **遅い！！**







MPI_File_set_view

```
C: MPI_File_set_view( MPI_File *fh, MPI_Offset disp,
                      MPI_Datatype etype, MPI_Datatype ftype,
                      char *datarep, MPI_Info info )
F: MPI_FILE_SET_VIEW( fh, disp, etype, ftype, datarep, ierr )
```

- Collective 関数
- **ftype**：etype の派生データ型で、これを呼んだプロセスがそのプロセスの **FP を用いて**アクセスするファイルの部分を示す
- **datarep**：ファイルに格納する数値データの表現方式
  - “native”　　マシン固有のバイナリー表現（無変換）
  - “internal”　　（native と external32 の中間）
  - “external32”　もっとも一般的で可搬性のあるバイナリー表現
- **info**：ファイルシステムに与えるヒント情報（システム依存）

MPI-IO における Datatype

- etype (elementary type)
  - ファイルにアクセスする場所を示す基本的な型
  - READ/WRITE で指定されるデータ型と *type signature* が同じでなければならない（各要素は同じ基本データ型であること）
- ftype (file type)
  - ファイルアクセスを分割するためのテンプレート
- デフォルト
  - etype：MPI_BYTE
  - ftype：MPI_BYTE

2種類の File Pointer

```
C:  MPI_File_seek( MPI_File fh, MPI_Offset off, int whence )
    MPI_File_seek_shared( MPI_File fh, MPI_Offset off, int whence )
    MPI_File_get_position( MPI_File fh, MPI_Offset *off )
    MPI_File_get_position_shared (MPI_File *fh, MPI_Offset *off )
F:  MPI_FILE_SEEK( fh, offset, whence, ierr )
    MPI_FILE_SEEK_SHARED( fh, off, whence, ierr )
    MPI_FILE_GET_POSITION( fh, off, ierr )
    MPI_FILE_GET_POSITION_SHARED( fh, off, ierr )
```

- MPI_FILE_SEEK
  - プロセス固有の FP を設定する
- MPI_FILE_SEEK_SHARED
  - 共有 FP を設定する
- MPI_FILE_GET_POSITION
  - プロセス固有の FP の値を得る
- MPI_FILE_GET_POSITION_SHARED
  - 共有 FP の値を得る

ファイルの操作

```
C: MPI_File_delete( char *filename, MPI_Info info )
   MPI_File_preallocate( MPI_File fh, MPI_Offset size )
   MPI_File_get_size( MPI_File fh, MPI_Offset *size )
F: MPI_FILE_DELETE( filename, info, ierr )
   MPI_FILE_PREALLOCATE( fh, size, ierr )
   MPI_FILE_GET_SIZE( fh, size, ierr )
```

- MPI_FILE_DELETE
  - ファイルの削除
- MPI_FILE_PREALLOCATE
  - ファイルサイズを実際に書込む前に確保する
- MPI_FILE_GET_SIZE
  - ファイルの大きさを得る

MPI-IO のまとめ

- 3種類のアクセスする場所の指定方法
  - 陽に指定する MPI_File_***_at()
  - プロセス固有の File Pointer
    - File View の指定が可能
  - ファイルハンドルで共有される File Pointer
    - 遅いのでできるだけ使わないように
- Collective IO
  - Derived Datatype と Collective IO を組み合わせて使うと速くなることが多い

時間の計測

```
C: double MPI_Wtime( void )
F: DOUBLE PRECISION MPI_WTIME()
```

- 秒単位の時刻を返す

```
double dt, ts, te;

MPI_Barrier( MPI_COMM_WORLD );
ts = MPI_Wtime();
...
MPI_Barrier( MPI_COMM_WORLD );
te = MPI_Wtime();
dt = te - ts;
```

並列プログラミングの基本（MPI）

- 並列プログラムとは
- MPIの基礎
- MPIの基本的な通信

- ネットワークトポロジーと性能
- 片方向通信
- Derived Data Type
- MPI-IO
- **MPI の仕様について**

MPI の仕様

- 本サマースクールにおける MPI は全て MPI 2.2 の仕様に基づく
  - 参考資料：MPI Version 2.2
- MPI の仕様策定 MPI Forum
  - http://www.mpi-forum.org/
  - だれでも参加可能
  - 仕様は参加者の投票で決まる
  - およそ３回／年のペースで開催
  - 来年9月は日本（AICS）で開催予定

MPI 3.0

- 2012 に公開
  - 主な新仕様
    - Non-blocking な collective（集団）通信
    - マルチスレッドでの性能向上、などなど

- 一部、既に MPI 4 に向けた検討が始まっている
  - 耐故障機能など

まとめ

- 通信
  - 1対1通信 - 8/5
  - コミュニケータと collective 通信 - 8/5
  - 片方向通信 - 8/7
- 派生データ型 - 8/7
- MPI-IO - 8/7

本講習でやらなかった機能

- Communication Mode
- Persistent Communication
- Topology
- Dynamic Process Creation
- Error Handling

など