# MPI-2: Односторонние операции

# MPI-2: Односторонние операции

- RMA – удаленный доступ к памяти
- В передаче данных необходимо участие лишь одного процесса
- RMA-механизм позволяет разработчикам воспользоваться преимуществом быстрых механизмов связи, обеспечиваемых различными платформами ()

# Терминология

- Инициатор – процесс, который выполняет вызов
- Адресат – процесс, к памяти которого выполняется обращение
- 3 коммуникационных вызова
  - MPI_Put()
  - MPI_Get()
  - MPI_Accumulate()

# Порядок работы с RMA-операциями

- Создание «окна» – определение участка памяти, который будет использован а RMA-операции
- Определить, какие данные будут передаваться и куда
- Определить способ оповещения о готовности данных

# Создание «окна»

- «Окно» –участок памяти, который может использоваться в RMA-операциях
- Этот участок должен быть непрерывным (contiguous) блоком
- Для создания «окна» достаточно указать начальный адрес и количество байт
- Создание «окна» – коллективная операция, должна вызываться всеми процессами внутри коммуникатора
- «Окно» - объект со скрытой структурой, который используется для всех дальнейших RMA-операций

# Создание «окна»

int MPI_Win_create(void *base,
MPI_Aint size, int disp_unit,
MPI_Info info, MPI_Comm comm,
MPI_Win *win)

- IN base начальный адрес окна
- IN size размер окна в байтах (неотрицательное целое число)
- IN disp_unit размер локальной единицы смещения в байтах (положительное целое)
- IN info аргумент info (дескриптор)
- IN comm коммуникатор (дескриптор)
- OUT win оконный объект, вызвращаемый вызовом (дескриптор)

# Атрибуты «окна»

- MPI_WIN_BASE - базовый адрес «окна»
  - MPI_Win_get_attr(win, MPI_WIN_BASE, &base, &flag)
- MPI_WIN_SIZE - размер «окна» в байтах
  - MPI_Win_get_attr(win, MPI_WIN_SIZE, &size, &flag)
- MPI_WIN_DISP_UNIT – смещение
  - MPI_Win_get_attr(win,MPI_WIN_DISP_UNIT, &disp_unit,&flag)
- Группа процессов, присоединенных к «окну»
  - int MPI_Win_get_group(MPI_Win win, MPI_Group *group)

# Освобождение памяти

int MPI_Win_free(MPI_Win *win)

- INOUT win оконный объект (дескриптор)

- освобождает оконный объект и возвращает пустой дескриптор (со значением MPI_WIN_NULL)
- коллективный вызов, выполняемый всеми процессами в группе, связанной с окном win
- может вызываться процессом только после того, как тот завершил участие в *RMA* взаимодействиях с «окном» win

# Перемещение данных

- Перемещение данных – MPI_Put, MPI_Get, MPI_Accumulate
- Все операции по перемещению данных – неблокирующие
- Синхронизация обязательна (чтобы убедиться, что операция заверщена)

**int MPI_Put(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)**

- IN origin_addr начальный адрес буфера инициатора (по выбору)
- IN origin_count число записей в буфере инициатора (неотрицательное целое)
- IN origin_datatype тип данных каждой записи в буфере инициатора (дескриптор)
- IN target_rank номер получателя (неотрицательное целое)
- IN target_disp смещение от начала окна до буфера получателя (неотрицательное целое)
- IN target_count число записей в буфере получателя (неотрицательное целое)
- IN target_datatype тип данных каждой записи в буфере получателя (дескриптор)
- IN win оконный объект, используемый для

**int MPI_Get(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)**

- OUT origin_addr начальный адрес буфера инициатора (по выбору)
- IN origin_count число записей в буфере инициатора (неотрицательное целое)
- IN origin_datatype тип данных каждой записи в буфере инициатора (дескриптор)
- IN target_rank ранк получателя (неотрицательное целое)
- IN target_disp смещение от начала окна до буфера адресата (неотрицательное целое)
- IN target_count число записей в буфере адресата (неотрицательное целое)
- IN target_datatype тип данных каждой записи в буфере адресата (дескриптор)
- IN win оконный объект, используемый для коммуникации (дескриптор)

**int MPI_Accumulate(void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)**

- IN origin_addr начальный адрес буфера (выбор)
- IN origin_count число записей в буфере инициатора (неотрицательное целое)
- IN origin_datatype тип данных каждой записи в буфере (дескриптор)
- IN target_rank ранг адресата (неотрицательное целое)
- IN target_disp смещение от начала окна до буфера адресата (неотрицательное целое)
- IN target_count число записей в буфере адресата (неотрицательное целое)
- IN target_datatype тип данных каждой записи в буфере адресата (дескриптор)
- IN op операция - как в MPI_Reduce (дескриптор)
- IN win оконный объект (дескриптор)

# Барьерная синхронизация

- int MPI_Win_fence(int assert, MPI_Win win)
  - IN **assert** программное допущение (целое)  (assert = 0)
  - IN **win** объект окна (дескриптор)
- Вызовы MPI_WIN_FENCE должны как предшествовать, так и следовать за вызовами get, put или accumulate, которые синхронизируются с помощью вызовов fence.

# Барьерная синхронизация

```
MPI_Win_create (A,
    ...., &win);
MPI_Win_fence (0,
    win);
If (rank == 0) {
MPI_Put (....., win);
MPI_Put (....., win);
.....
MPI_Put (....., win);
}
```

```
MPI_Win_fence (0,
    win);
MPI_Get (.... , win);
MPI_Win_fence (0,
    win);
A[rank] = 4;
MPI_Win_fence (0,
    win);
MPI_Put ( ... , win);
MPI_Win_fence (0,
    win);
```

# Синхронизация Lock/Unlock

- int MPI_Win_lock(int lock_type, int rank, int assert, MPI_Win win)
  - IN lock_type или MPI_LOCK_EXCLUSIVE или MPI_LOCK_SHARED (состояние)
  - IN rank ранк процессора, имеющего доступ к окну (неотрицательное целое)
  - IN assert программный ассерт (целое)
  - IN win объект окна (дескриптор)
- int MPI_Win_unlock(int rank, MPI_Win win)
  - IN rank ранк процессора, имеющего доступ к окну (неотрицательное целое)
  - IN win объект окна (дескриптор)

# Синхронизация Lock/Unlock: пример

```
If (rank == 0) {
   MPI_Win_lock (MPI_LOCK_SHARED,
   1, 0, win);
   MPI_Put (outbuf, n, MPI_INT, 1, 0, n,
   MPI_INT, win);
   MPI_Win_unlock (1, win);
}
```

# Пример: Глобальная синхронизация

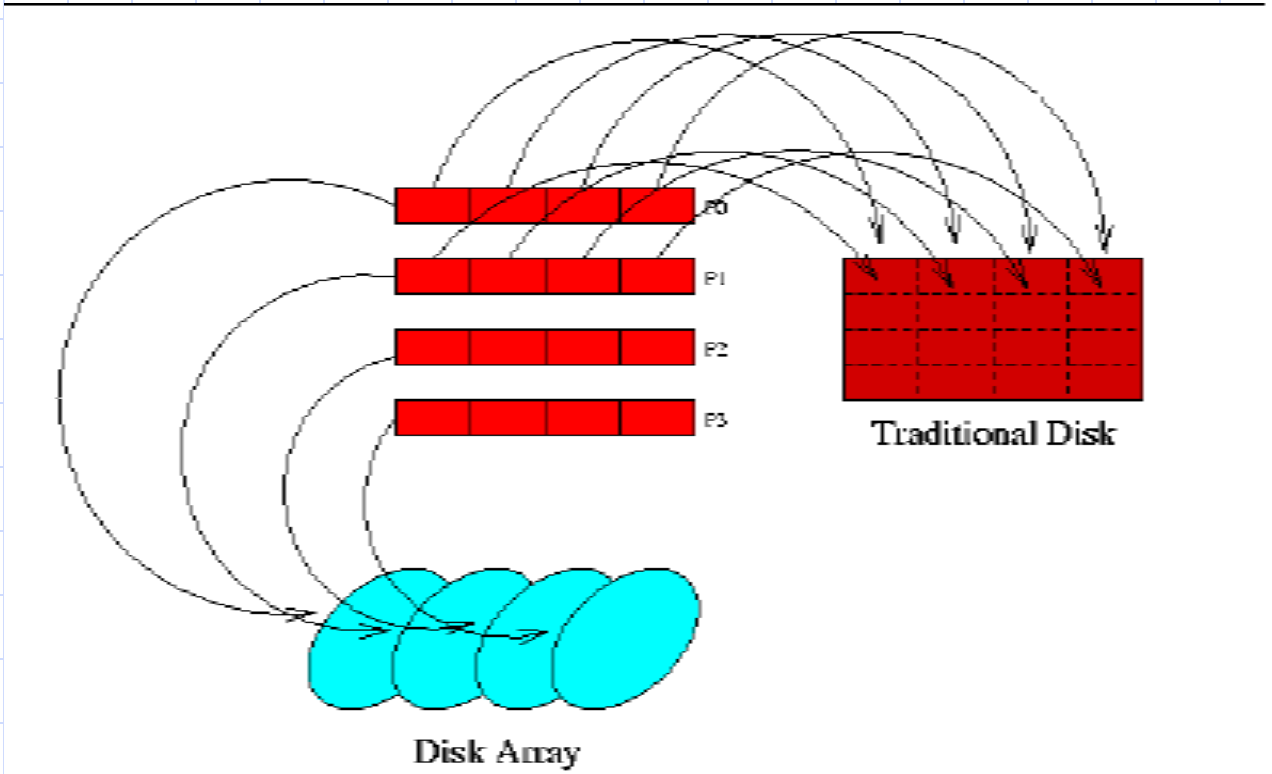
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

/* this sample uses 1 side communication ("remote memory access") to
   get the first and last value of an array on remote processes. */

int main(int argc, char *argv[]){
  int i;
  int nProc, myRank;
  int *data;
  int N;
  int *fl;
  MPI_Win win;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
  MPI_Comm_size(MPI_COMM_WORLD, &nProc);
  fl = (int *)calloc(2*nProc, sizeof(int));

  /* generate some data */
  N = 100;
  data = (int *)calloc(N, sizeof(int));
  for (i=0;i<N;i++){
    data[i] = N*myRank + i ;
  }
```

```
  /* create the window on memory to expose */
  MPI_Win_create(data, N*sizeof(int), sizeof(int),
          MPI_INFO_NULL, MPI_COMM_WORLD, &win);

  /* check everything is up to date */
  MPI_Win_fence(0, win);
  for (i=0;i<nProc;i++){
    /* get the first and last value from the remote process */
    MPI_Get(fl+2*i, 1, MPI_INT, (myRank +i) % nProc, 0, 1, MPI_INT,
   win);
    MPI_Get(fl+2*i+1, 1, MPI_INT, (myRank +i) % nProc, N-1, 1, MPI_INT,
   win);
  }
  /* perform global synchronization */
  MPI_Win_fence(0, win);

  /* after synchronization, we can use the results */
  for (i=0;i<2*nProc;i+=2){
    printf("%d: %d\t%d\n", myRank, fl[i], fl[i+1]);
  }

  MPI_Finalize();
  return(0);
}
```

# MPI-2: Параллельный ввод/вывод

# Параллельный ввод/вывод

- Несколько процессов имеют доступ к одному файлу



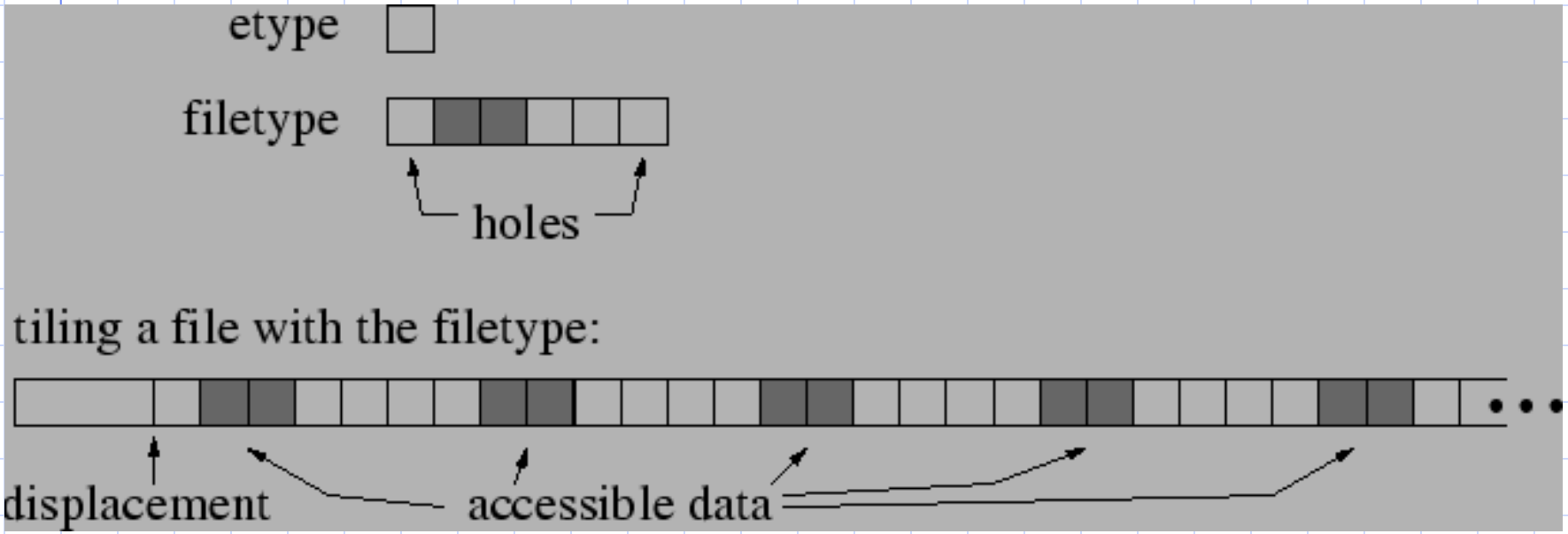
# MPI-2 I/O: Терминология

- ***MPI файл*** - это упорядоченная последовательность типизированных блоков данных. *MPI* поддерживает произвольный или последовательный доступ к любому цельному набору этих блоков. Файл открывается коллективно группой процессов. Все коллективные обращения к файлу коллективны на этой группе.
- ***Смещение файла*** - это абсолютное положение байта, относящегося к началу этого файла.

# MPI-2 I/O: Терминология

- ***E-тип (элементарный тип данных)*** - это единица доступа к данным и позиционирования. Это может быть любой определенный в *MPI* или производный тип данных.
- ***Файловый тип*** - это базис для разбиения файла в среде процессов, он определяет шаблон доступа к файлу. Файловый тип - это обычный e-тип или производный тип данных *MPI*, состоящий из нескольких элементов одного и того же e-типа. Кроме того, размер любой «дыры» в файловом типе должен быть кратным размеру этого e-типа.

# E-типы и Файловые типы



# MPI-2 I/O: Терминология

- ***Вид*** определяет текущий набор данных, видимый и доступный из открытого файла как упорядоченный набор е-типов. Каждый процесс имеет свой вид файла, определенный тремя параметрами: смещением, е-типом и файловым типом. Шаблон, описанный в файловом типе, повторяется, начиная со смещения, чтобы определить вид.
- ***Типовое смещение*** - это позиция в файле относительно текущего вида, представленная как число е-типов. «Дыры» в файловом типе вида пропускаются при подсчете номера этой позиции. Нулевое смещение - это позиция первого видимого е-типа в виде (после пропуска смещения и начальных «дыр» в виде).

# MPI-2 I/O: Терминология

- ***Размер MPI файла*** измеряется в байтах от начала файла
- ***Конец файла*** - это смещение первого е-типа, доступного в данном виде, начинающегося после последнего байта в файле
- ***Указатель на файл*** - это постоянное смещение, устанавливаемое MPI
  - «Индивидуальные файловые указатели» - файловые указатели, локальные для каждого процесса, открывающего файл.
  - «Общие файловые указатели» - это указатели, которые используются одновременно группой процессов, открывающих файл.
- ***Дескриптор файла*** - это закрытый объект, создаваемый MPI_FILE_OPEN и уничтожаемый MPI_FILE_CLOSE. Все операции над открытым файлом работают с файлом через его дескриптор

# MPI-2 I/O: Базовый алгоритм работы

- Определение необходимых переменных и типов данных
- Открытие файла (MPI_File_open)
- Установка вида файла (MPI_File_set_view)
- Запись/чтение (MPI_File_write, MPI_File_read)
- Для неблокирующих операций, ожидание их завершения (напр., MPI_Wait)
- Закрытие файла (MPI_File_close)

# Открытие файла

- int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)
  - IN *comm* коммуникатор (дескриптор)
  - IN *filename* имя открываемого файла (строка)
  - IN *amode* тип доступа к файлу (целое)
  - IN *info* информационный объект (дескриптор)
  - OUT *fh* новый дескриптор файла (дескриптор)
- открывает файл с именем *filename* для всех процессов из группы коммуникатора comm
- все процессы должны обеспечивать одинаковое значение *amode* и имена файлов, указывающие на один и тот же файл
- info используется как «подсказка» (шаблоны доступа)

# Типы доступа

- MPI_MODE_RDONLY -- только чтение,
- MPI_MODE_RDWR -- чтение и запись,
- MPI_MODE_WRONLY -- только запись,
- MPI_MODE_CREATE -- создавать файл, если он не существует,
- MPI_MODE_EXCL -- ошибка, если создаваемый файл уже существует,
- MPI_MODE_DELETE_ON_CLOSE -- удалять файл при закрытии,
- MPI_MODE_UNIQUE_OPEN -- файл не будет параллельно открыт где-либо еще,
- MPI_MODE_SEQUENTIAL -- файл будет доступен лишь последовательно,
- MPI_MODE_APPEND -- установить начальную позицию всех файловых указателей на конец файла.

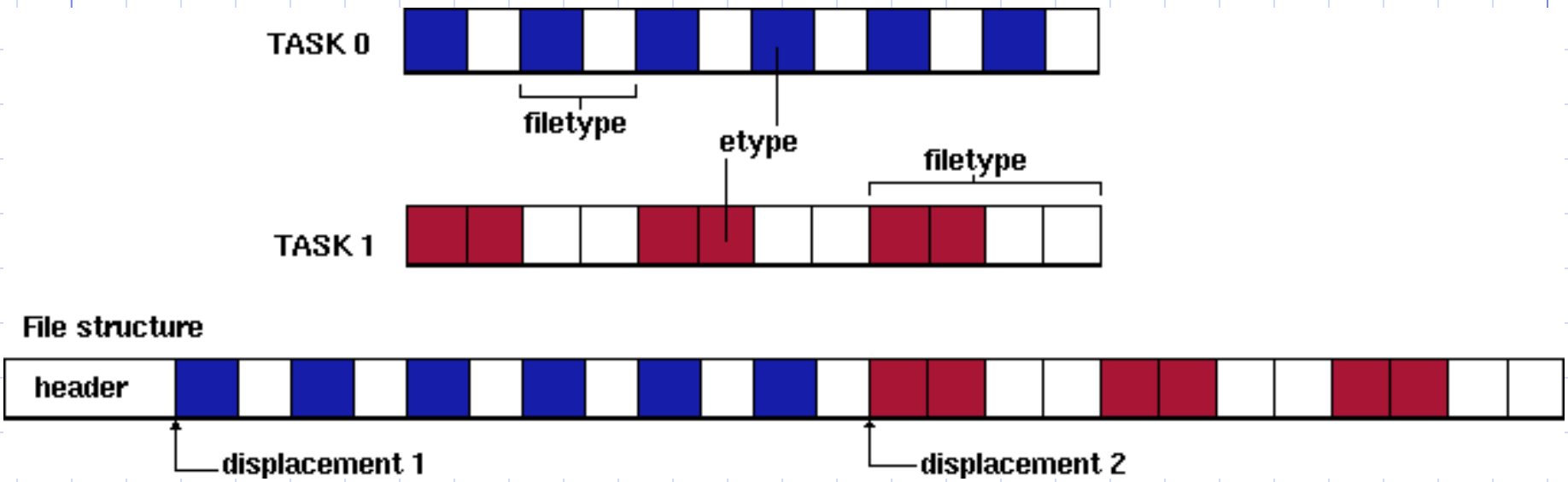
# Закрытие файла

- int MPI_File_close(MPI_File *fh)
  - INOUT *fh* дескриптор файла (дескриптор)
- сначала синхронизирует состояние файла затем закрывает файл, ассоциированный с *fh*
- пользователь должен обеспечить, чтобы все ожидающие обработки неблокирующие запросы и разделенные коллективные операции над *fh*, производимые процессом, были выполнены до вызова MPI_FILE_CLOSE

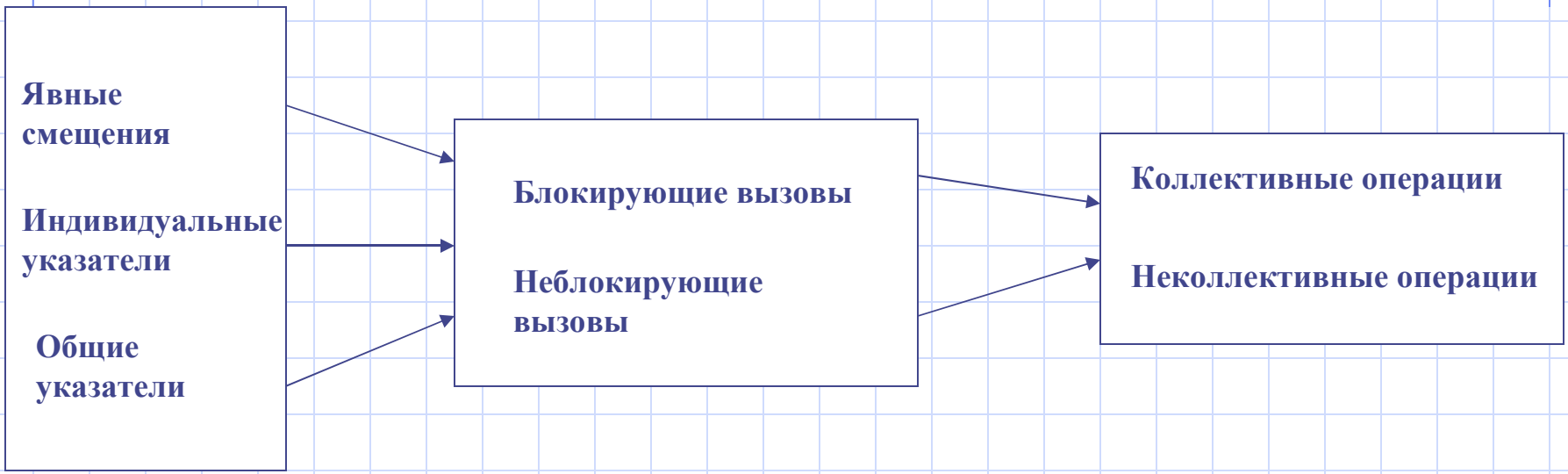
# Файловые виды

- int MPI_File_set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char *datarep, MPI_Info info)
  - INOUT *fh* дескриптор файла (дескриптор)
  - IN *disp* смещение (целое)
  - IN *etype* элементарный тип данных (дескриптор)
  - IN *filetype* тип файла (дескриптор)
  - IN *datarep* представление данных (строка)
  - IN *info* информационный объект (дескриптор)
- int MPI_File_get_view(MPI_File fh, MPI_Offset *disp, MPI_Datatype *etype, MPI_Datatype *filetype, char *datarep)
  - IN *fh* дескриптор файла (дескриптор)
  - OUT *disp* смещение (целое)
  - OUT *etype* элементарный тип данных (дескриптор)
  - OUT *filetype* тип файла (дескриптор)

# Файловые виды и структура файла



# Доступ к данным



| Позиционирование | Синхронизация | Координация | |
|---|---|---|---|
| | | неколлективные | коллективные |
| Явные смещения | блокирующие | **MPI_FILE_READ_AT**<br>**MPI_FILE_WRITE_AT** | **MPI_FILE_READ_AT_ALL**<br>**MPI_FILE_WRITE_AT_ALL** |
| | неблокирующие и расщепленные коллективные | **MPI_FILE_IREAD_AT**<br>**MPI_FILE_IWRITE_AT** | **MPI_FILE_READ_AT_ALL_BEGIN**<br>**MPI_FILE_READ_AT_ALL_END**<br>**MPI_FILE_WRITE_AT_ALL_BEGIN**<br>**MPI_FILE_WRITE_AT_ALL_END** |
| Индивидуальные указатели | блокирующие | **MPI_FILE_READ**<br>**MPI_FILE_WRITE** | **MPI_FILE_READ_ALL**<br>**MPI_FILE_WRITE_ALL** |
| | неблокирующие и расщепленные коллективные | **MPI_FILE_IREAD**<br>**MPI_FILE_IWRITE** | **MPI_FILE_READ_ALL_BEGIN**<br>**MPI_FILE_READ_ALL_END**<br>**MPI_FILE_WRITE_ALL_BEGIN**<br>**MPI_FILE_WRITE_ALL_END** |
| Общие указатели | блокирующие | **MPI_FILE_READ_SHARED**<br>**MPI_FILE_WRITE_SHARED** | **MPI_FILE_READ_ORDERED**<br>**MPI_FILE_WRITE_ORDERED** |
| | неблокирующие и расщепленные коллективные | **MPI_FILE_IREAD_SHARED**<br>**MPI_FILE_IWRITE_SHARED** | **MPI_FILE_READ_ORDERED_BEGIN**<br>**MPI_FILE_READ_ORDERED_END**<br>**MPI_FILE_WRITE_ORDERED_BEGIN**<br>**MPI_FILE_WRITE_ORDERED_END** |

# (индивидуальные указатели)

- **int MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - INOUT *fh* - дескриптор файла (дескриптор)
  - OUT *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера (дескриптор)
  - OUT *status* - объект состояния (Status)
- **int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - INOUT *fh* - дескриптор файла (дескриптор)
  - IN *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера (дескриптор)

# Доступ к данным (явные смещения)

- **int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - INOUT *fh* - дескриптор файла (дескриптор)
  - IN *offset* – смещение
  - OUT *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера (дескриптор)
  - OUT *status* - объект состояния (Status)
- **int MPI_File_write_at(MPI_File fh, MPI_Offset offsetб void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - INOUT *fh* - дескриптор файла (дескриптор)
  - IN *offset* – смещение
  - IN *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера

# смещения, коллективные)

- **int MPI_File_read_at_all(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - INOUT *fh* - дескриптор файла (дескриптор)
  - IN *offset* – смещение
  - OUT *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера (дескриптор)
  - OUT *status* - объект состояния (Status)
- **int MPI_File_write_at_all(MPI_File fh, MPI_Offset offsetб void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - INOUT *fh* - дескриптор файла (дескриптор)
  - IN *offset* – смещение
  - IN *buf* - начальный адрес буфера (выбор)
  - IN *count* - количество элементов в буфере (целое)
  - IN *datatype* - тип данных каждого элемента буфера

# Фрагмент программы (базовый алгоритм)

```
MPI_File fh;
MPI_Datatype filetype;
MPI_Status status;
MPI_Offset offset;
int mode;
float data[100];

/* other code */
/* set offset and filetype (covered later) */

mode = MPI_MODE_CREATE|MPI_MODE_RDWR;
 MPI_File_open(MPI_COMM_WORLD, "myfile", mode,
    MPI_INFO_NULL &fh);
MPI_File_set_view(fh, offset, MPI_FLOAT, filetype, "native",
    MPI_INFO_NULL);
MPI_File_write(fh, data, 100, MPI_FLOAT, &status);
MPI_File_close(&fh);
```

# Поддержка целостности

- Коллективные операции не синхронизируются => могут возникать конфликты по доступу
- Совокупность операций доступа к данным последовательно непротиворечива, если она ведет себя так, как будто операции были выполнены последовательно в порядке, соответствующем порядку программы
- Каждая попытка доступа выглядит атомарной, несмотря на то, что точный порядок попыток доступа

# Поддержка целостности

- int MPI_File_set_atomicity(MPI_File fh, int flag)
  - все записи в файл немедленно записываются на диск; коллективная операция
    - INOUT fh Дескриптор файла (дескриптор)
    - IN flag true для установки атомарного режима, false для отмены атомарного режима (логическая)
- int MPI_File_get_atomicity(MPI_File fh, int *flag)
  - возвращает текущее значение семантики непротиворечивости для операций доступа к данным
    - IN fh Дескриптор файла (дескриптор)
    - INOUT flag true при атомарном режиме, false при неатомарном режиме (логическая)

# Поддержка целостности

- int MPI_File_sync(MPI_File fh) – записывает все буферизованные изменения, инициированные процессом, на диск
  - INOUT fh Дескриптор файла (дескриптор)
- int MPI_File_close(MPI_File fh) – тоже поддерживает целостность! – записывает все буферизованные изменения на диск перед закрытием файла

# Пример: перемножение матриц

# Алгоритм

- Каждый процессор считает один блок результирующе матрицы в соответствии со следующей схемой (схема дана для 4 MPI-процессов):