MPI-IO

Creating a Standard for Parallel I/O

Birds Of a Feather Session

Supercomputing '94

WHAT IT IS

- An interface definition for parallel I/O in an MPI environment
- Express high-level information about I/O structure
- Targeted primarily for scientific applications
- Favors common usage patterns and real-world requirements

WHY WE NEED IT

- MPI programs are not really portable without a standard I/O interface
- Current parallel systems have divergent I/O interfaces
- Unix file interface is ill-suited for parallel computing
  - No support for parallel access by multiple processes
  - No support for strided access in one operation

WHO IS RESPONSIBLE...

Peter Corbett, Dror Feitelson, Yarsun Hsu, Jean-Pierre Prost, Marc Snir
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Sam Fineberg[1], Bill Nitzberg[1], Bernard Traversat[1], Parkson Wong[1]
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

[1]Computer Sciences Corporation, NASA Ames Research Center, under Contract NAS2-12961

HOW IT WORKS

Bright idea #1:
Model I/O as message passing

- Reading from a file is like receiving a message
- Writing to a file is like sending a message
- Non-blocking I/O is like non-blocking messages

Bright idea #2:
Use MPI derived datatypes to express access pattern into file

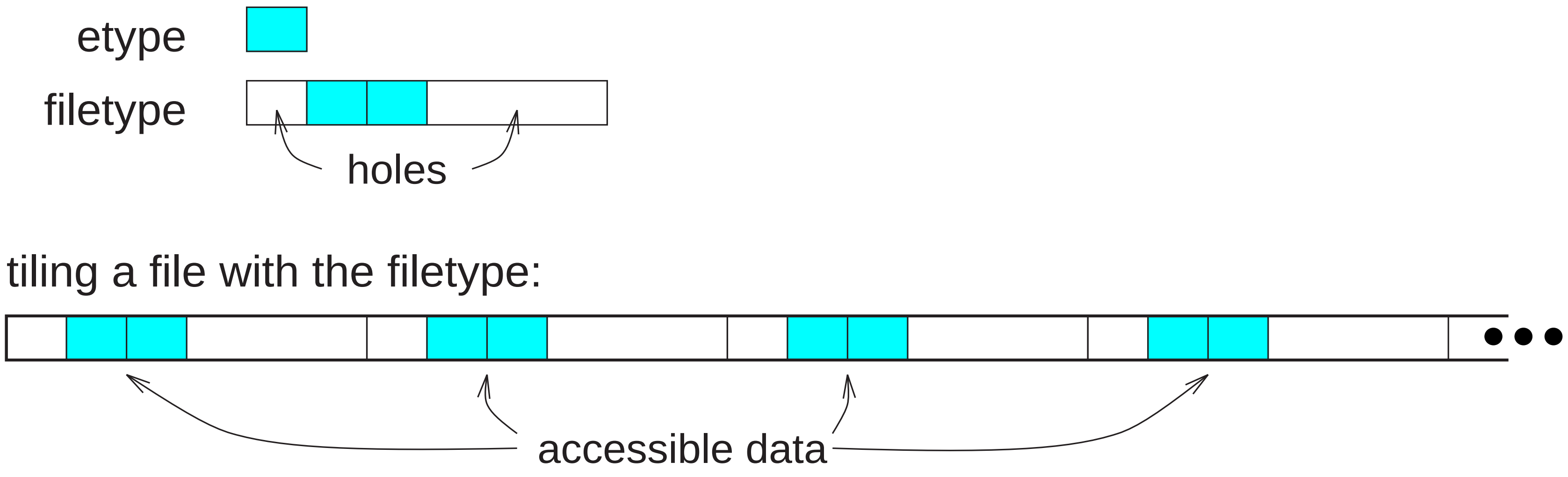
- Flexible way to express strided access
- Flexible way to express partitioning of data among processes
- Generalization of Vesta partitioning

ETYPE

- Defines elementary data type when file is opened
- Can be MPI basic type or derived datatype
- Used as unit of offset into file
- A common building block of filetype and buftype
  - Guarantees match of type signatures
- Full flexibility by using MPI_BYTE

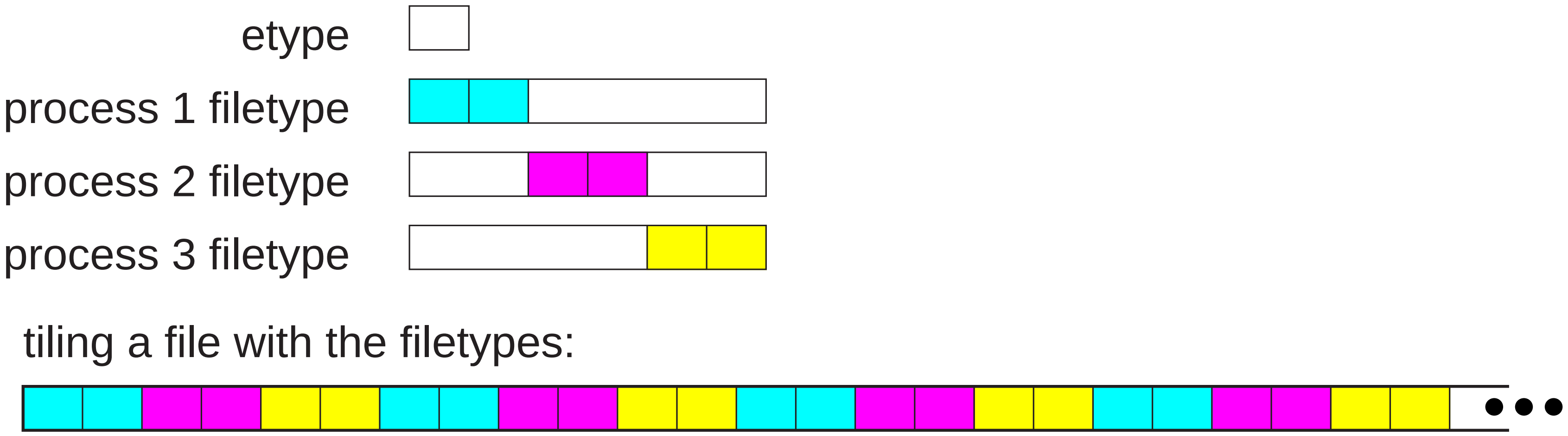
FILETYPE

- Defines file data type when file is opened
- Derived from etype
- Used to express access pattern into file



PARTITIONING FILE DATA

Use complementary file types



- Data items can be of different sizes
- Assignment need not be in rank order
- Overlap between partitions is possible

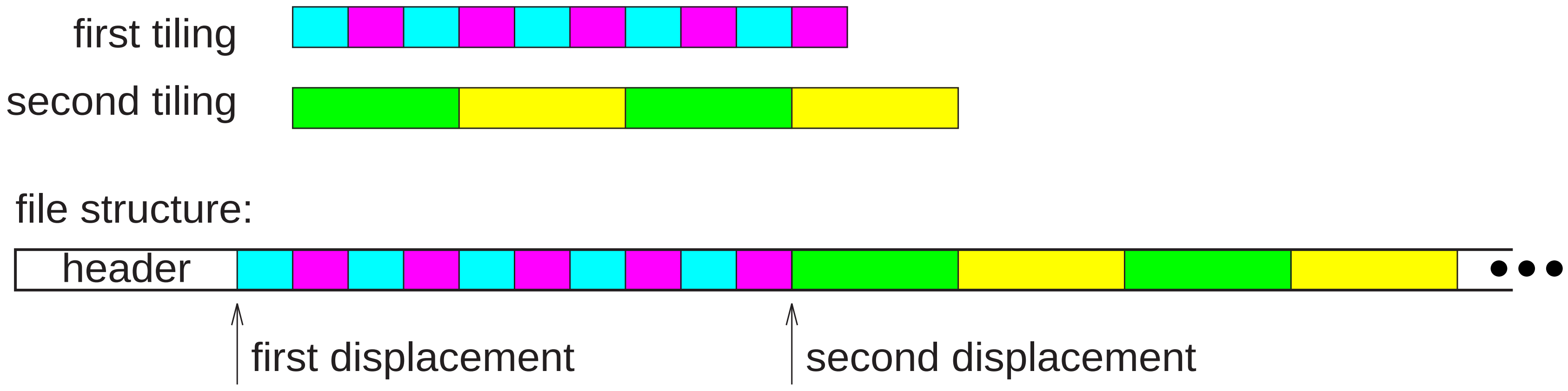
FILE OPEN (collective)

MPI_OPEN(comm, filename, amode, pmode, disp, etype,
filetype, moffset, hints, fh)

IN	comm	[SAME] Process group communicator (handle)
IN	filename	[SAME] Name of file to be opened (string)
IN	amode	[SAME] File access mode (integer)
IN	pmode	[SAME] Permission mode (integer)
IN	disp	Absolute displacement (nonnegative offset)
IN	etype	Elementary datatype (handle)
IN	filetype	File datatype (handle)
IN	moffset	Relative/Absolute offset flag (integer)
IN	hints	Hints to the file system (array of choice)
OUT	fh	Returned file handle (handle)

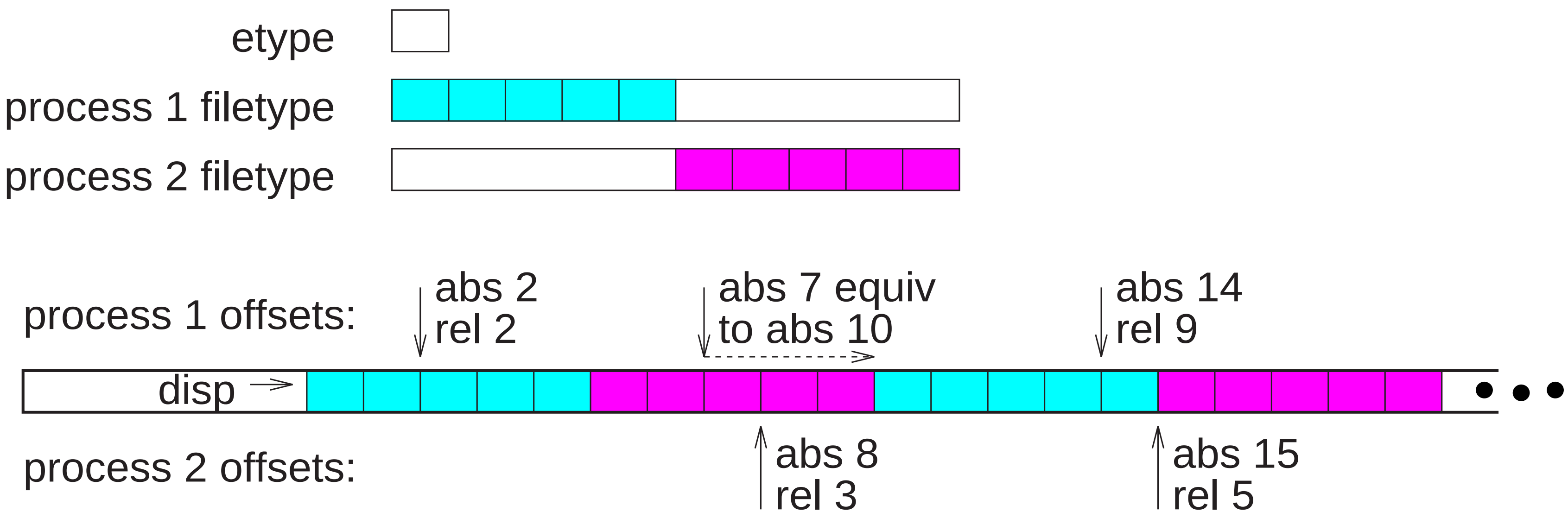
DISPLACEMENTS

- Skip over segment at beginning of file
- Expressed in bytes



OFFSETS

- Expressed in etype units
- Starting from the displacement
- Absolute offsets count all data in the file
  if inaccessible, refers to next accessible item
- Relative offsets only count accessible data



HINTS

- Hints do not change the semantics
- Provide suggestions about how to use the underlying system
- Can be expanded by implementations
- Implemented as attribute list with (name,value) pairs
  - Striping unit
  - Striping factor
  - I/O node list
  - Compute node number
  - Access style (sequential, random, readmostly)
  - Partitioning pattern

DATA ACCESS FUNCTIONS

- No need for special functions for broadcast, scatter, gather: this is expressed by filetypes
- Separate functions for independent or collective behavior
- Separate functions for blocking or nonblocking behavior
- Separate functions for different offset types
  - Offset passed explicitly in call
  - Independent offsets maintained by system
  - Shared offset maintained by system

FUNCTION GRID

offset	synchronization	independent	collective
explicit offset	blocking	MPI_Read MPI_Write	MPI_Read_all MPI_Write_all
	nonblocking	MPI_Iread MPI_Iwrite	MPI_Iread_all MPI_Iwrite_all
independent file pointer	blocking	MPI_Read_next MPI_Write_next	MPI_Read_next_all MPI_Write_next_all
	nonblocking	MPI_Iread_next MPI_Iwrite_next	MPI_Iread_next_all MPI_Iwrite_next_all
shared file pointer	blocking	MPI_Read_shared MPI_Write_shared	– –

INDEPENDENT READ with EXPLICIT OFFSET

MPI_READ(fh, offset, buff, buftype, bufcount, status)

IN	fh	Valid file handle (handle)
IN	offset	File offset (offset)
OUT	buff	Address of user's buffer (integer)
IN	buftype	User's buffer datatype (handle)
IN	bufcount	Number of buftype elements (integer)
OUT	status	Status information (Status)

Buftype used to distribute data in user's buffer, as in MPI

COLLECTIVE OPERATIONS

- All members of the communicator group must participate
- Does not imply a barrier synchronization point
- Allows implementation to optimize access

MPI_READ_ALL(fh, offset, buff, buftype, bufcount, status)

IN	fh	[SAME] Valid file handle (handle)
IN	offset	File offset (offset)
OUT	buff	Address of user's buffer (integer)
IN	buftype	User's buffer datatype (handle)
IN	bufcount	Number of buftype elements (integer)
OUT	status	Status information (Status)

SYSTEM MAINTAINED OFFSETS

MPI_READ_NEXT(fh, buff, buftype, bufcount, status)

IN	fh	Valid file handle (handle)
OUT	buff	Address of user's buffer (integer)
IN	buftype	User's buffer datatype (handle)
IN	bufcount	Number of buftype elements (integer)
OUT	status	Status information (Status)

In nonblocking and shared functions, internal offset is updated before the operation completes

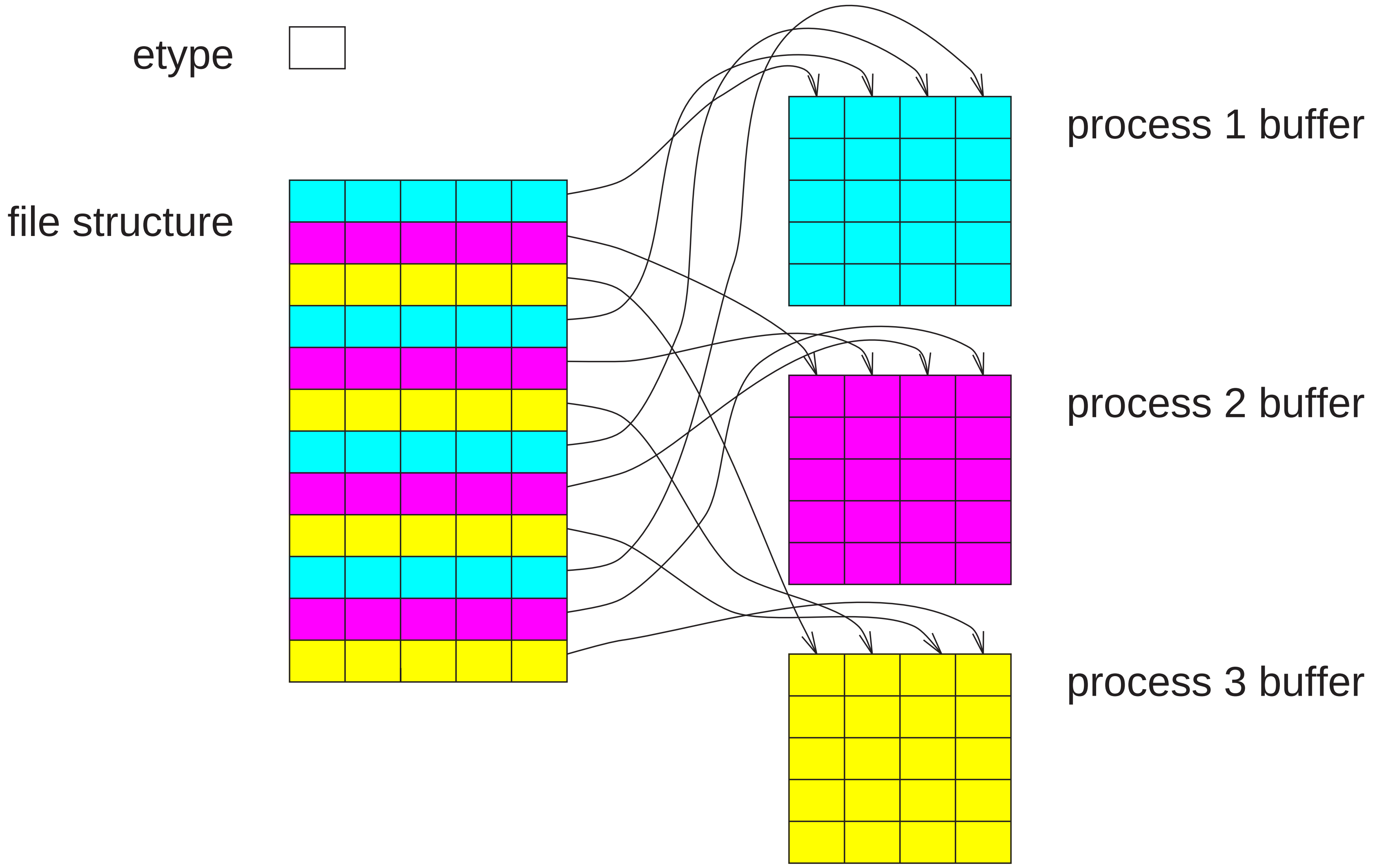
FILETYPE CONSTRUCTORS

Generate filetypes for common patterns

- Read_broadcast
- Write_reduce
- Scatter_gather
- Scatterv_gatherv
- HPF_block
- HPF_cyclic
- HPF_2d

EXAMPLE

Partition a matrix in row-cyclic pattern and transpose



EXAMPLE CODE

```
MPI_Type_hpf_cyclic (MPI_COMM_WORLD,
        allrows*cols, cols, etype, &filetype);

MPI_Open (MPI_COMM_WORLD, "matfile", MPI_RDONLY,
        0644, 0, etype, filetype, MPI_RELATIVE, NULL, &fh);

MPI_Type_vector (cols, 1, myrows, etype, &column_t);
MPI_Type_hvector (myrows, 1, size(etype), column_t, &buftype);

MPI_Read (fh, 0, mat, buftype, 1, &status);
```