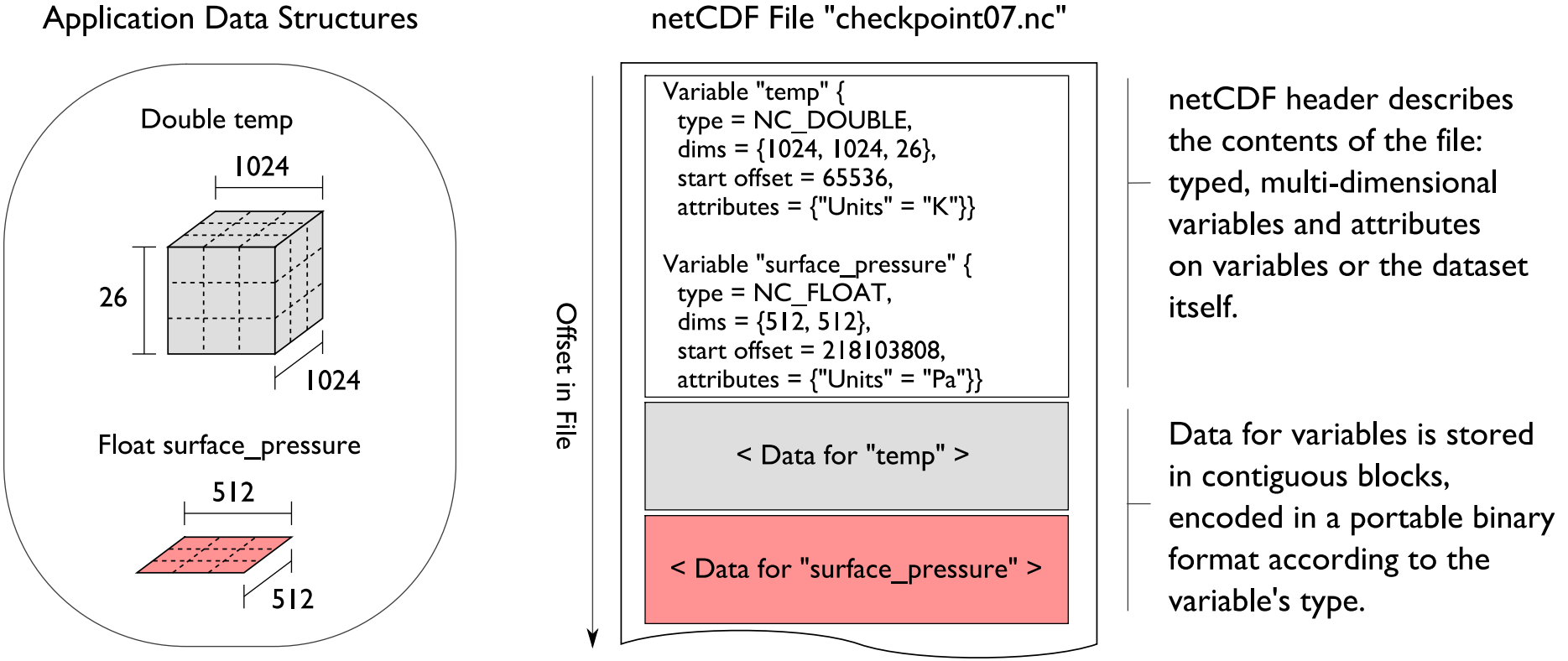
Parallel NetCDF (PnetCDF)

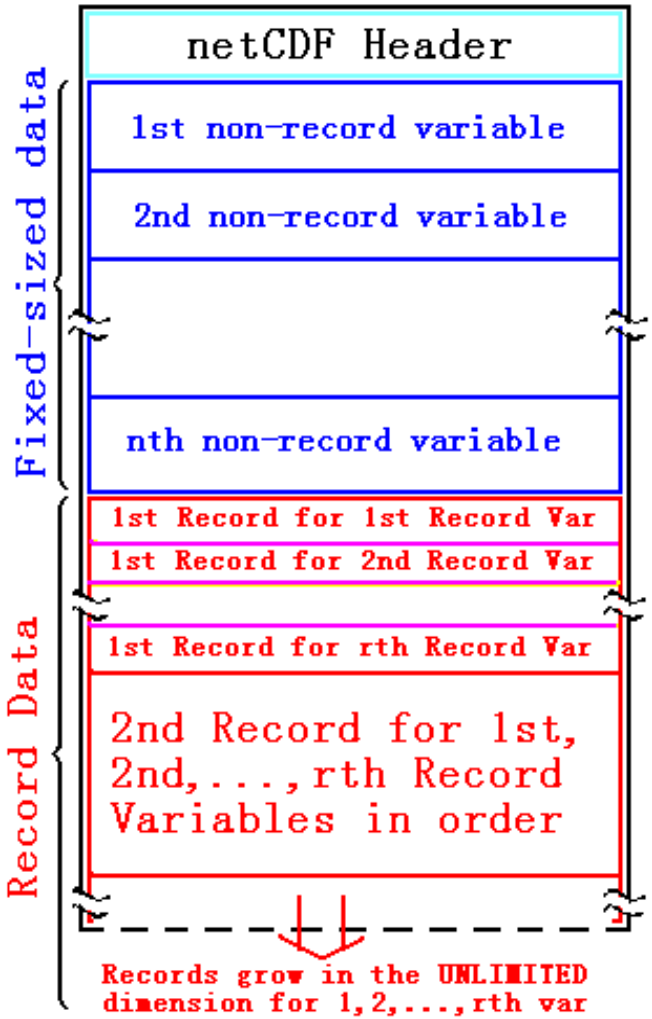
- Based on original “Network Common Data Format” (netCDF) work from Unidata
  - Derived from their source code
- Data Model:
  - Collection of variables in single file
  - Typed, multidimensional array variables
  - Attributes on file and variables
- Features:
  - C, Fortran, and F90 interfaces
  - Portable data format (identical to netCDF)
  - Noncontiguous I/O in memory using MPI datatypes
  - Noncontiguous I/O in file using sub-arrays
  - Collective I/O
  - Non-blocking I/O
- Unrelated to netCDF-4 work
- Parallel-NetCDF tutorial:
  - http://trac.mcs.anl.gov/projects/parallel-netcdf/wiki/QuickTutorial

Data Layout in netCDF Files



Record Variables in netCDF

- Record variables are defined to have a single “unlimited” dimension
  - Convenient when a dimension size is unknown at time of variable creation
- Record variables are stored after all the other variables in an interleaved format
  - Using more than one in a file is likely to result in poor performance due to number of noncontiguous accesses



Storing Data in PnetCDF

- Create a dataset (file)
  - Puts dataset in define mode
  - Allows us to describe the contents
    - Define dimensions for variables
    - Define variables using dimensions
    - Store attributes if desired (for variable or dataset)
- Switch from define mode to data mode to write variables
- Store variable data
- Close the dataset

Example: FLASH with PnetCDF

- FLASH AMR structures do not map directly to netCDF multidimensional arrays
- Must create mapping of the in-memory FLASH data structures into a representation in netCDF multidimensional arrays
- Chose to
  - Place all checkpoint data in a single file
  - Impose a linear ordering on the AMR blocks
    - Use 4D variables
  - Store each FLASH variable in its own netCDF variable
    - Skip ghost cells
  - Record attributes describing run time, total blocks, etc.

Defining Dimensions

```
int status, ncid, dim_tot_blks, dim_nxb,
  dim_nyb, dim_nzb;
MPI_Info hints;
/* create dataset (file) */
status = ncmpi_create(MPI_COMM_WORLD, filename,
  NC_CLOBBER, hints, &file_id);
/* define dimensions */
status = ncmpi_def_dim(ncid, "dim_tot_blks",
  tot_blks, &dim_tot_blks);
status = ncmpi_def_dim(ncid, "dim_nxb",
  nzones_block[0], &dim_nxb);
status = ncmpi_def_dim(ncid, "dim_nyb",
  nzones_block[1], &dim_nyb);
status = ncmpi_def_dim(ncid, "dim_nzb",
  nzones_block[2], &dim_nzb);
```

Each dimension gets a unique reference

Creating Variables

```
int dims = 4, dimids[4];
intvarids[NVARS];
/* define variables (X changes most quickly) */
dimids[0] = dim_tot_blks;
dimids[1] = dim_nzb;
dimids[2] = dim_nyb;
dimids[3] = dim_nxb;
for (i=0; i< NVARS; i++) {
   status = ncmpi_def_var(ncid, unk_label[i],
     NC_DOUBLE, dims, dimids, &varids[i]);
}
```

Same dimensions used for all variables

Storing Attributes

```
/* store attributes of checkpoint */
status = ncmpi_put_att_text(ncid, NC_GLOBAL,
   "file_creation_time", string_size,
   file_creation_time);
status = ncmpi_put_att_int(ncid, NC_GLOBAL,
   "total_blocks", NC_INT, 1, tot_blks);
status = ncmpi_enddef(file_id);

/* now in data mode … */
```

Writing Variables

```
double *unknowns; /* unknowns[blk][nzb][nyb][nxb]
   */
size_t start_4d[4], count_4d[4];
start_4d[0] = global_offset; /* different for each
   process */
start_4d[1] = start_4d[2] = start_4d[3] = 0;
count_4d[0] = local_blocks;
count_4d[1] = nzb;  count_4d[2] = nyb;
   count_4d[3] = nxb;
for (i=0; i< NVARS; i++) {
    /* ... build datatype "mpi_type" describing
       values of a single variable ... */
    /* collectively write out all values of a
       single variable */
    ncmpi_put_vara_all(ncid, varids[i], start_4d,
       count_4d, unknowns, 1, mpi_type);
}
status = ncmpi_close(file_id);
```

Typical MPI buffer-count-type tuple

Inside PnetCDF Define Mode

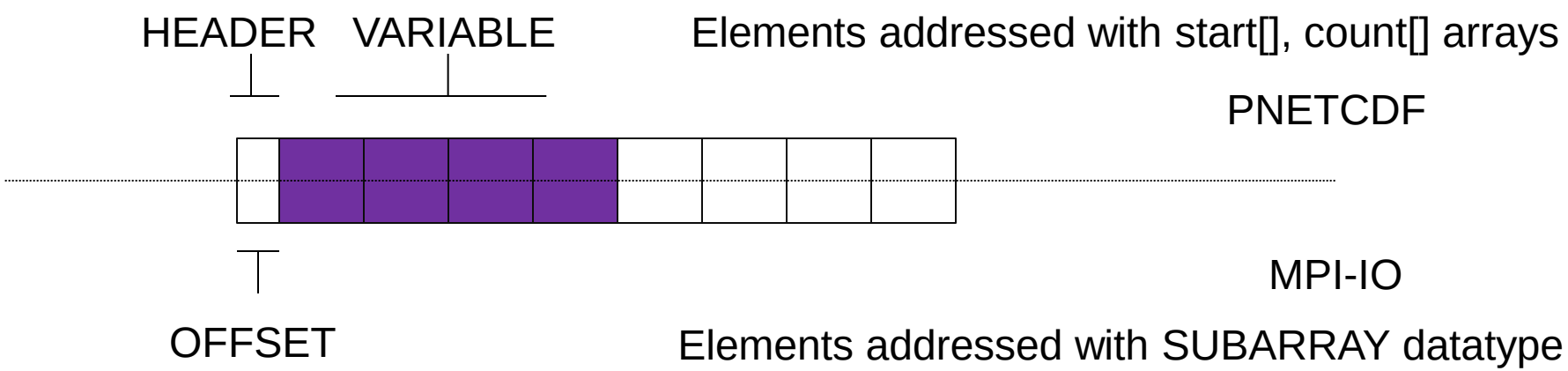
- In define mode (collective)
  - Use `MPI_File_open` to create file at create time
  - Set hints as appropriate (more later)
  - Locally cache header information in memory
    - All changes are made to local copies at each process
- At ncmpi_enddef
  - Process 0 writes header with `MPI_File_write_at`
  - `MPI_Bcast` result to others
  - Everyone has header data in memory, understands placement of all variables
    - No need for any additional header I/O during data mode!

Inside PnetCDF Data Mode

- Inside `ncmpi_put_vara_all` (once per variable)
  - Each process performs data conversion into internal buffer
  - Uses `MPI_File_set_view` to define file region
    - Contiguous region for each process in FLASH case
  - `MPI_File_write_all` collectively writes data
- At ncmpi_close
  - `MPI_File_close` ensures data is written to storage

- MPI-IO performs optimizations
  - Two-phase possibly applied when writing variables
- MPI-IO makes PFS calls
  - PFS client code communicates with servers and stores data

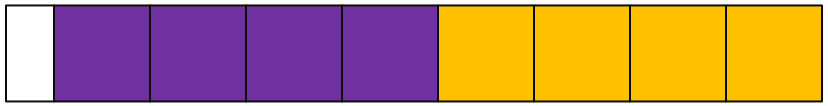
Parallel-NetCDF and MPI-IO



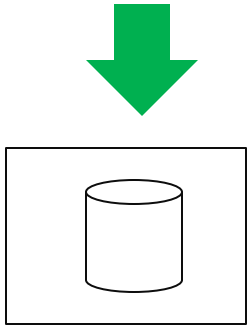
- `ncmpi_put_vara_all` describes access in terms of arrays, elements of arrays
  - E.g. Give me a 3x3 subcube of this larger 1024x1024 array
- Library translates into MPI-IO calls
  - `MPI_Type_create_subarray`
  - `MPI_File_set_view`
  - `MPI_File_write_all`

Parallel-NetCDF write-combining optimization

```
ncmpi_iput_vara(ncfile, varid1,
    &start, &count, &buffer1,
    count, MPI_INT, &requests[0]);
ncmpi_iput_vara(ncfile, varid2,
    &start,&count, &buffer2,
    count, MPI_INT, &requests[1]);
ncmpi_wait_all(ncfile, 2, requests, statuses);
```



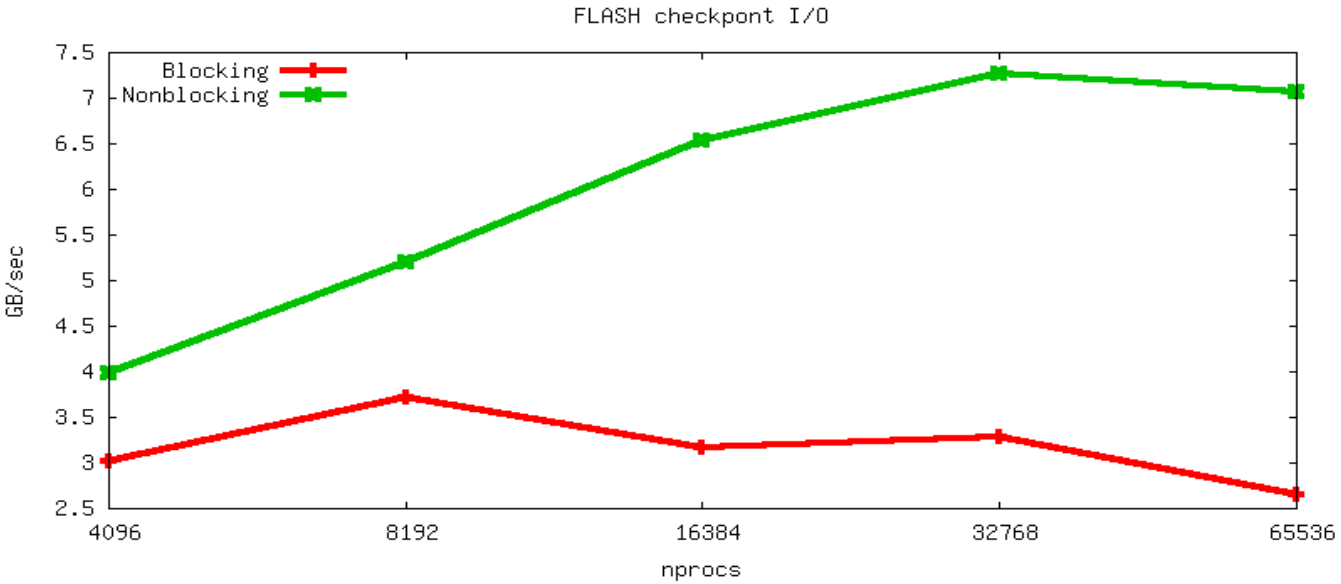
HEADER VAR1 VAR2



- netCDF variables laid out contiguously
- Applications typically store data in separate variables
  - temperature(lat, long, elevation)
  - Velocity_x(x, y, z, timestep)
- Operations posted independently, completed collectively
  - Defer, coalesce synchronization
  - Increase average request size

FLASH Astrophysics and the write-combining optimization

- FLASH writes one variable at a time
- Could combine all 4D variables (temperature, pressure, etc) into one 5D variable
  - Altered file format (conventions) requires updating entire analysis toolchain
- Write-combining provides improved performance with same file conventions
  - Larger requests, less synchronization.
  - Convinced HDF to develop similar interface



PnetCDF Wrap-Up

- PnetCDF gives us
  - Simple, portable, self-describing container for data
  - Collective I/O
  - Data structures closely mapping to the variables described
- If PnetCDF meets application needs, it is likely to give good performance
  - Type conversion to portable format does add overhead
- Some limits on (old, common CDF-2) file format:
  - Fixed-size variable: < 4 GiB
  - Per-record size of record variable: < 4 GiB
  - 2^{32} -1 records
  - New extended file format to relax these limits (CDF-5, released in pnetcdf-1.1.0; Integrated in Unidata NetCDF-4.4)