

NATIONAL ENERGY RESEARCH
SCIENTIFIC COMPUTING CENTER

National Energy Research Scientific Computing Center (NERSC)

I/O Patterns from NERSC Applications

NERSC Center Division, LBNL





User Requirements

- Write data from multiple processors into a single file
- Undo the “domain decomposition” required to implement parallelism
- File can be read in the same manner regardless of the number of CPUs that read from or write to the file
 - we want to see the logical data layout... not the physical layout
- Do so with the same performance as writing one-file-per-processor
 - Use one-file-per-processor because of performance problems (would prefer one-file per application)



Usage Model

(focus here is on append-only I/O)

- **Checkpoint/Restart**
 - Most users don't do hero applications: tolerate failure by submitting more jobs (and that includes apps that are targeting hero-scale applications)
 - Most people doing "hero applications" have written their own restart systems and file formats
 - Typically close to memory footprint of code per dump
 - Must dump memory image ASAP!
 - Not as much need to remove the domain decomposition (recombiners for MxN problem)
 - not very sophisticated about recalculating derived quantities (stores all large arrays)
 - Might go back more than one checkpoint, but only need 1-2 of them online (staging)
 - Typically throw the data away if CPR not required
- **Data Analysis Dumps**
 - Time-series data most demanding
 - Typically run with coarse-grained time dumps
 - If something interesting happens, resubmit job with higher output rate (and take a huge penalty for I/O rates)
 - Async I/O would make 50% I/O load go away, but nobody uses it! (*cause it rarely works*)
 - Optimization or boundary-value problems typically have flexible output requirements (typically diagnostic)



Common Storage Formats

- **ASCII:** *(pitiful... this is still common... even for 3D I/O... and you want an exaflop??)*
 - Slow
 - Takes more space!
 - Inaccurate
- **Binary**
 - Nonportable (eg. byte ordering and types sizes)
 - Not future proof
 - Parallel I/O using MPI-IO
- **Self-Describing formats**
 - NetCDF/HDF4, HDF5, Silo
 - Example in HDF5: API implements Object DB model in portable file
 - Parallel I/O using: pHDF5/pNetCDF (hides MPI-IO)
- **Community File Formats**
 - FITS, HDF-EOS, SAF, PDB, Plot3D
 - Modern Implementations built on top of HDF, NetCDF, or other self-describing object-model API



Common Data Models/Schemas

- **Structured Grids:**
 - 1D-6D domain decomposed mesh data
 - Reversing Domain Decomposition results in strided disk access pattern
 - Multiblock grids often stored in chunked fashion
- **Particle Data**
 - 1D lists of particle data (x,y,z location + physical properties of each particle)
 - Often non-uniform number of particles per processor
 - PIC often requires storage of Structured Grid together with cells
- **Unstructured Cell Data**
 - 1D array of cell types
 - 1D array of vertices (*x,y,z locations*)
 - 1D array of cell connectivity
 - Domain decomposition has similarity with particles, but must handle ghost cells
- **AMR Data**
 - Chombo: Each 3D AMR grid occupies distinct section of 1D array on disk (*one array per AMR level*).
 - Enzo (Mike Norman, UCSD): One file per processor (*each file contains multiple grids*)
 - BoxLib: One file per grid (*each grid in the AMR hierarchy is stored in a separate, cleverly named, file*)
- **Increased need for processing data from terrestrial sensors (read-oriented)**
 - NERSC is now a net importer of data



Physical Layout Tends to Result in Handful of I/O Patterns

- **2D-3D I/O patterns (small-block strided I/O)**
 - 1 file per processor (*Raw Binary and HDF5*)
 - *Raw binary assesses peak performance*
 - *HDF5 determines overhead of metadata, data encoding, and small accesses associated with storage of indices and metadata*
 - 1-file reverse domain decomp (*Raw MPI-IO and pHDF5*)
 - *MPI-IO is baseline (peak performance)*
 - *Assess pHDF5 or pNetCDF implementation overhead*
 - 1-file chunked (*looks like 1D I/O pattern*)
- **1D I/O patterns (large-block strided I/O)**
 - *Same as above, but for 1D data layouts*
 - *1-file per processor is same in both cases*
 - *Often difficult to ensure alignment to OST boundaries*



Common Themes for Storage Patterns

(alternative to prev slide)

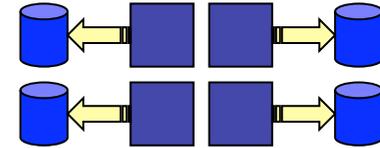
- **Diversity of I/O data schemas derive down two handful of I/O patterns at disk level**
- **1D I/O**
 - Examples: GTC, VORPAL particle I/O, H5Part, ChomboHDF5, FLASH-AMR
 - Interleaved I/O operations with large transaction size (hundreds of kilobytes to megabytes)
 - Three categories
 - Equal sized transactions per processor
 - Slightly unequal sized transactions per processor (not load-imbalanced, but difficult to align)
 - Unequal sized transactions with load-imbalance (not focus of our attention)
- **2D, 3D, >3D I/O pattern**
 - Examples: Cactus, Flash (unchunked), VORPAL 3D-IO
 - Reverse domain decomposition
 - Use chunking to increase transaction sizes
 - Chunking looks like 1D I/O case
 - Results in interleaved output with very small transaction sizes (kilobyte sized)
- **Out-of-Core I/O**
 - Examples: MadCAP, MadBench, OOCore
 - Very large-block transactions (multimegabyte or multigigabyte)
 - Intense for both read and write operations



Common Physical Layouts For Parallel I/O

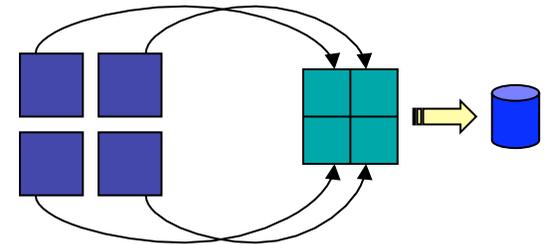
- **One File Per Process**

- Terrible for HPSS!
- Difficult to manage



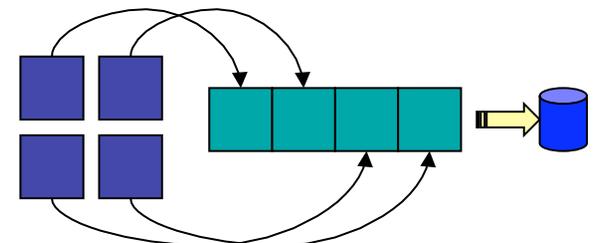
- **Parallel I/O into a single file**

- Raw MPI-IO
- pHDF5 pNetCDF



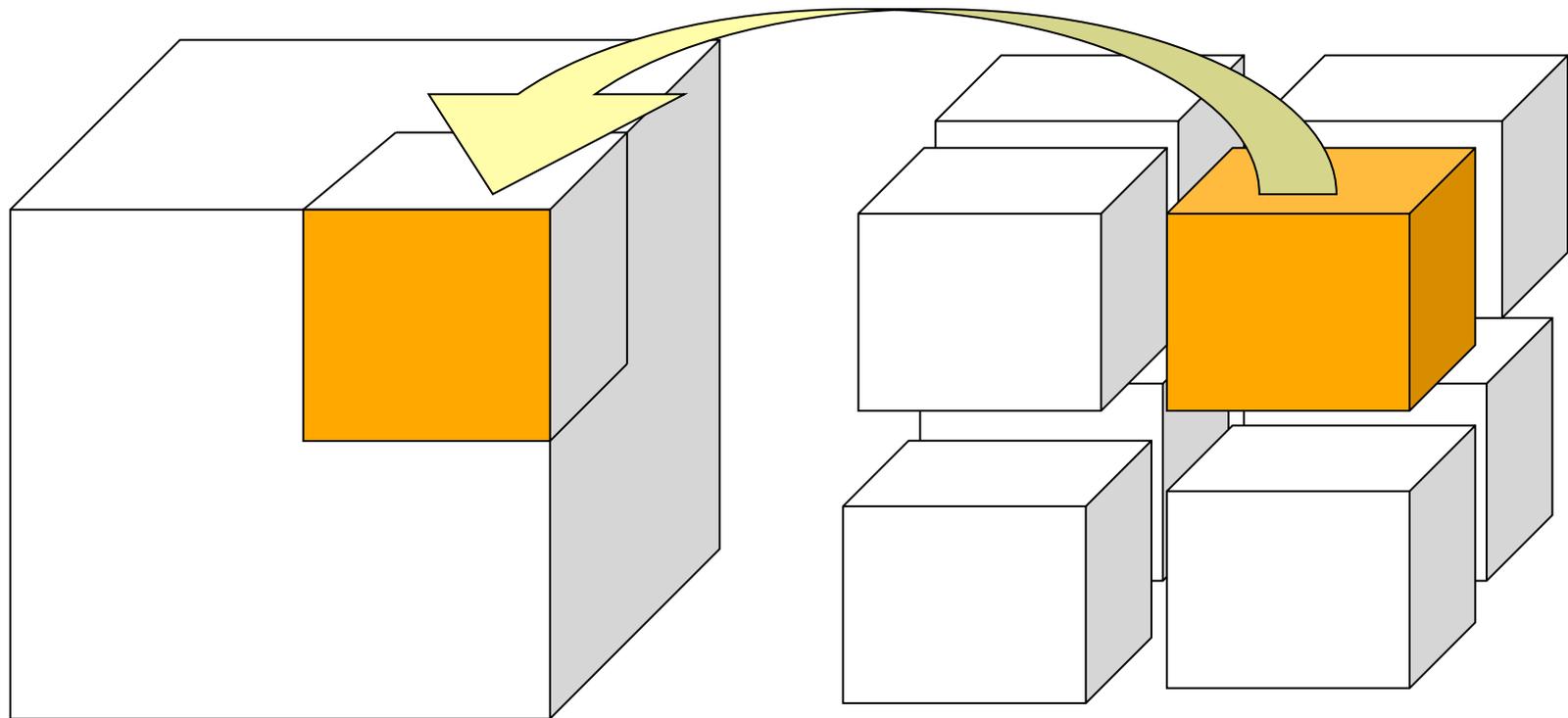
- **Chunking into a single file**

- Saves cost of reorganizing data
- Depend on API to hide physical layout
- (eg. expose user to logically contiguous array even though it is stored physically as domain-decomposed chunks)



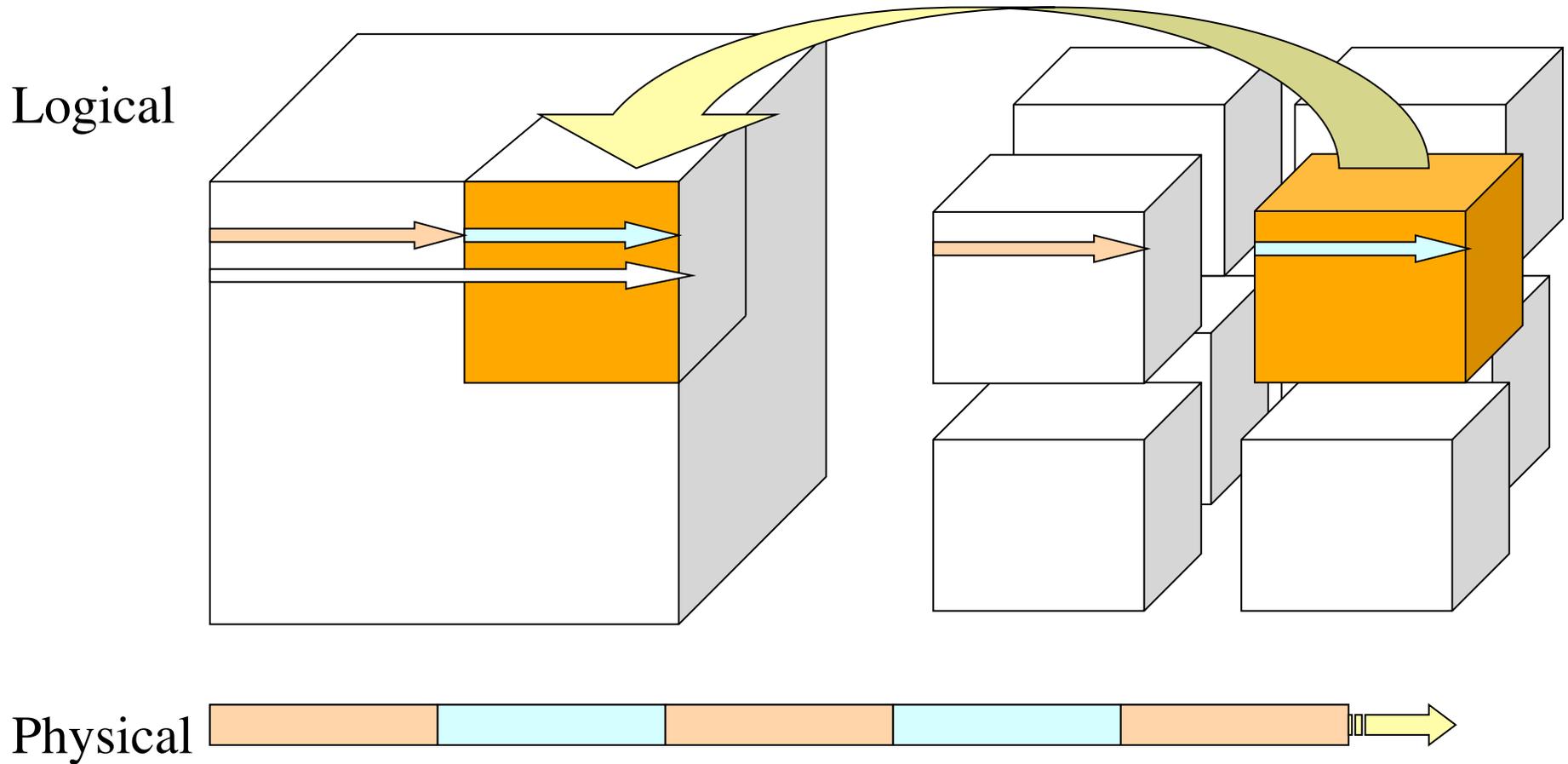


3D (reversing the domain decomp)



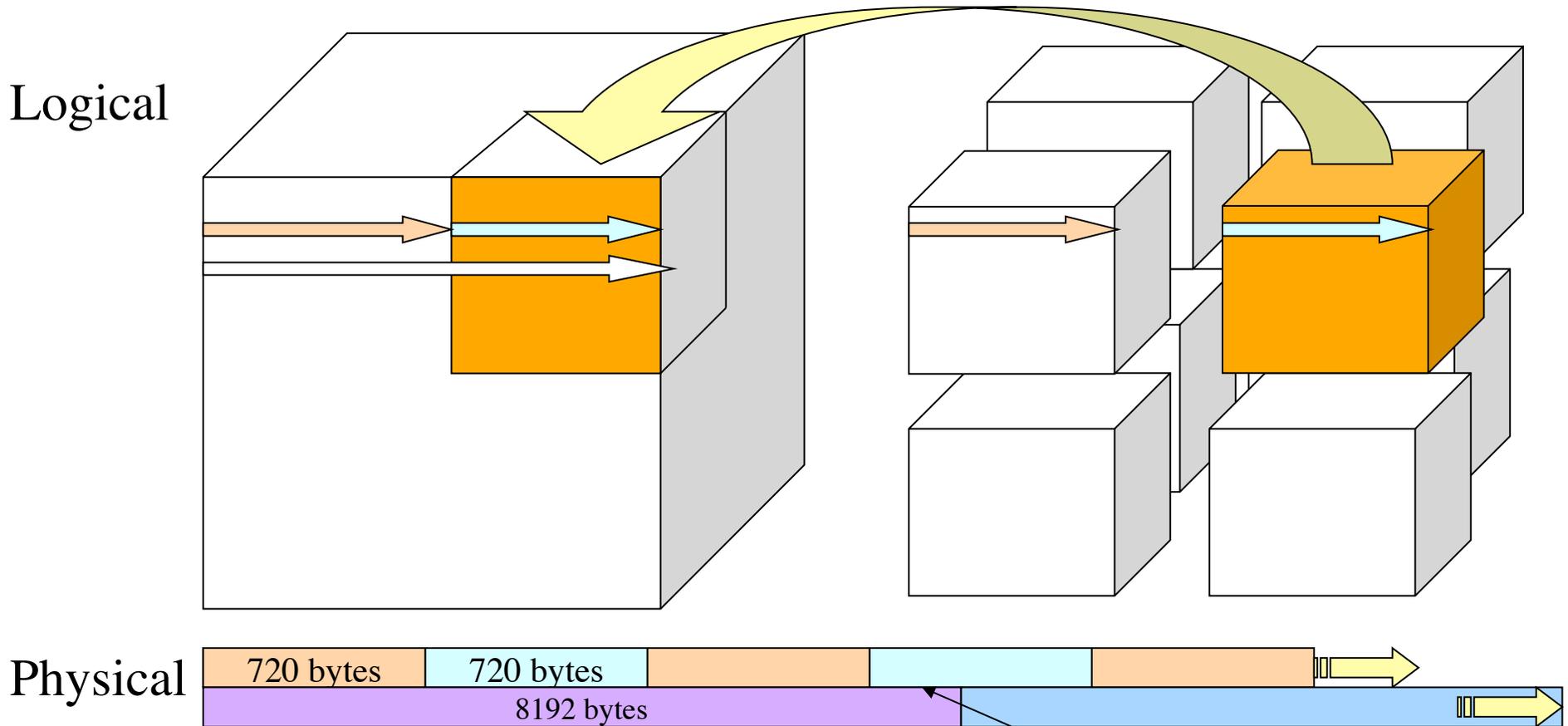


3D (reversing the decomp)





3D (block alignment issues)



Physical

- Block updates require mutual exclusion
- Block thrashing on distributed FS
- I/O efficiency for sparse updates! (8k block required for 720 byte I/O operation)
- Unaligned block accesses can kill performance! (but are necessary in practical I/O solutions)

Writes not aligned to block boundaries



Interleaved Data Issues Accelerator Modeling Data

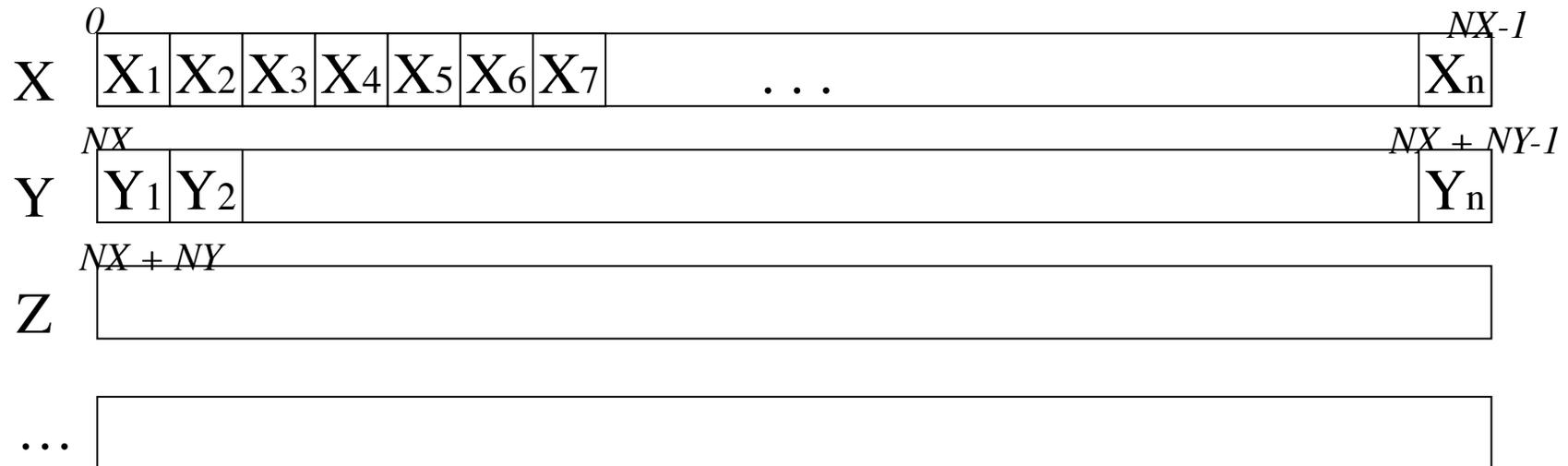
- **Point data**
 - Electrons or protons
 - Millions or billions in a simulation
 - Distribution is non-uniform
 - Fixed distribution at start of simulation
 - Change distribution (load balancing) each iteration
- **Attributes of a point**
 - Location: (double) x,y,z
 - Phase: (double) mx,my,mz
 - ID: (int64) id
 - Other attributes

**Interleaving and transaction sizes are similar to
AMR and chunked 3D Data.**



Nonuniform Interleaved Data (*accelerator modeling*)

Storage Format

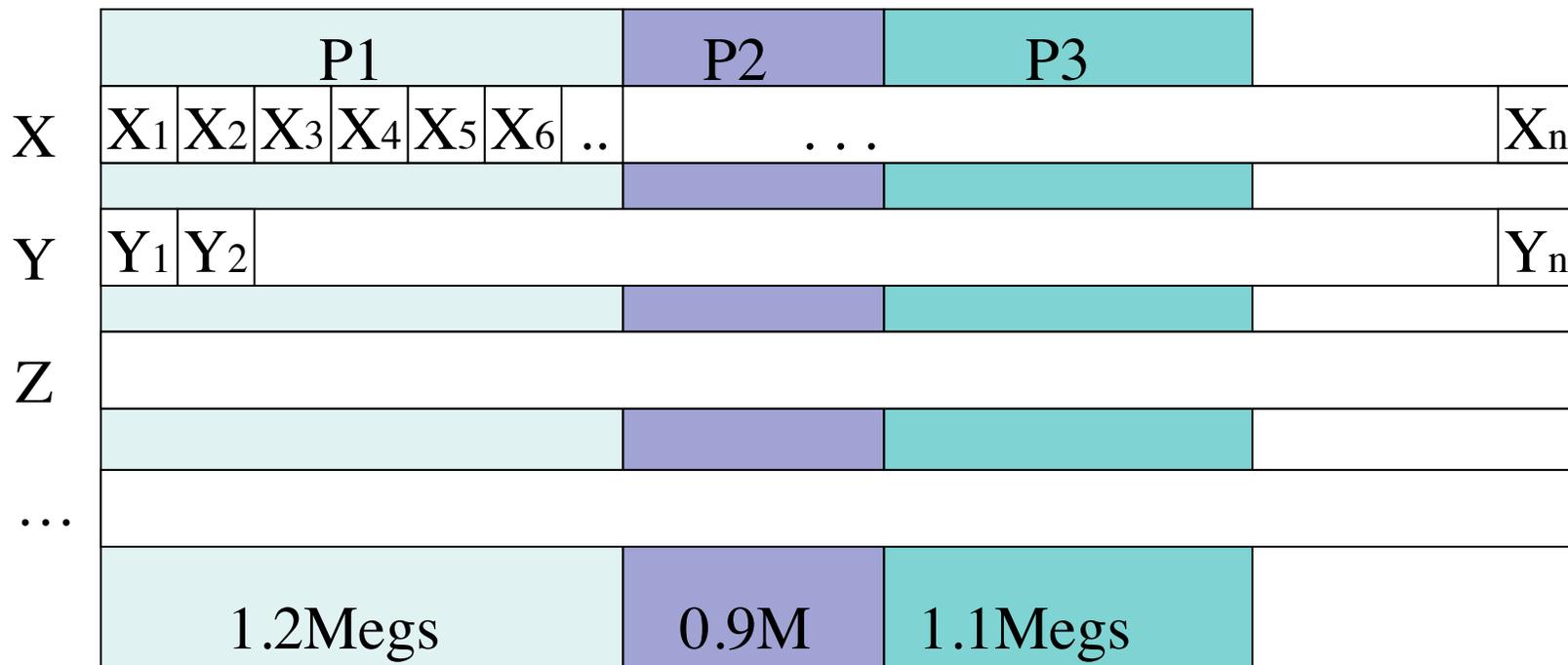


Laid out sequentially on disk
but view is interleaved on per-processor basis



Nonuniform Interleaved Data (*accelerator modeling*)

Storage Format



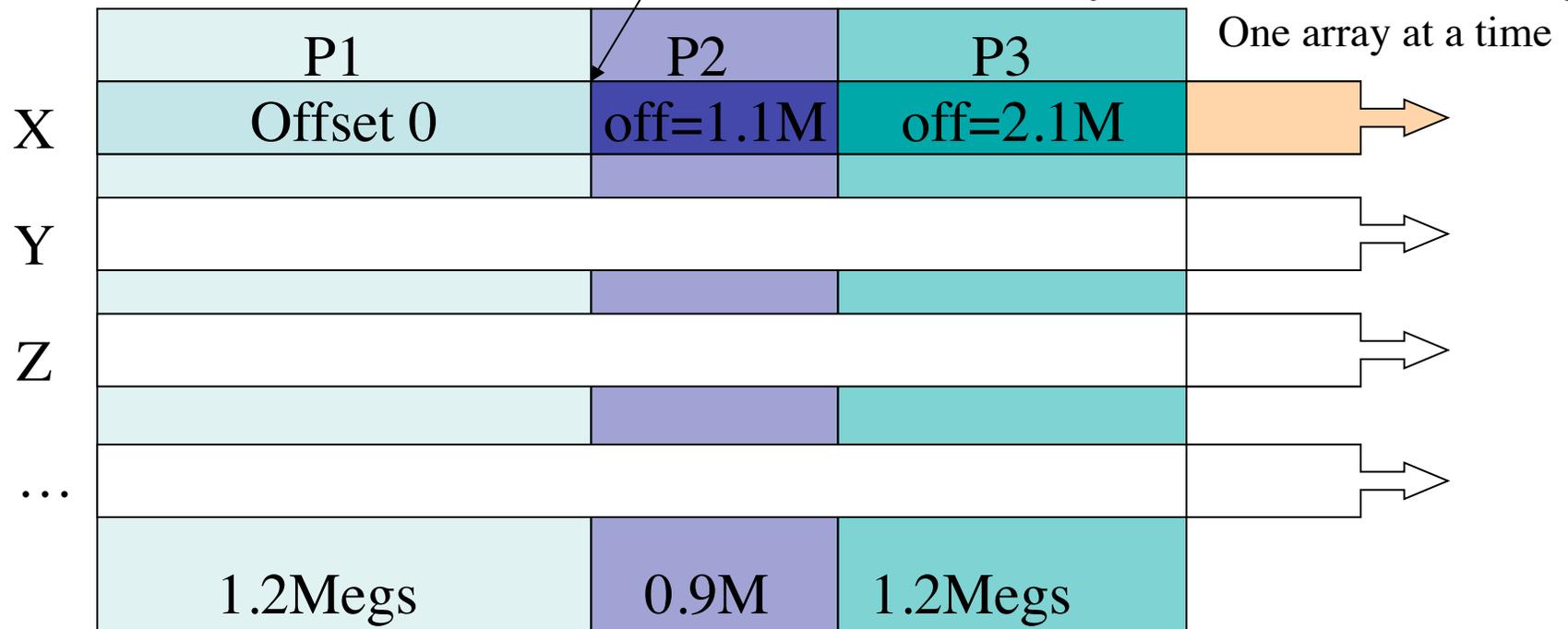
Slight load imbalance, but not substantial
nonuniform alignment has huge penalty!



Nonuniform interleaved

Calculate Offsets using Collective (AllGather)

Then write to mutually exclusive sections of array



Still suffers from alignment issues...

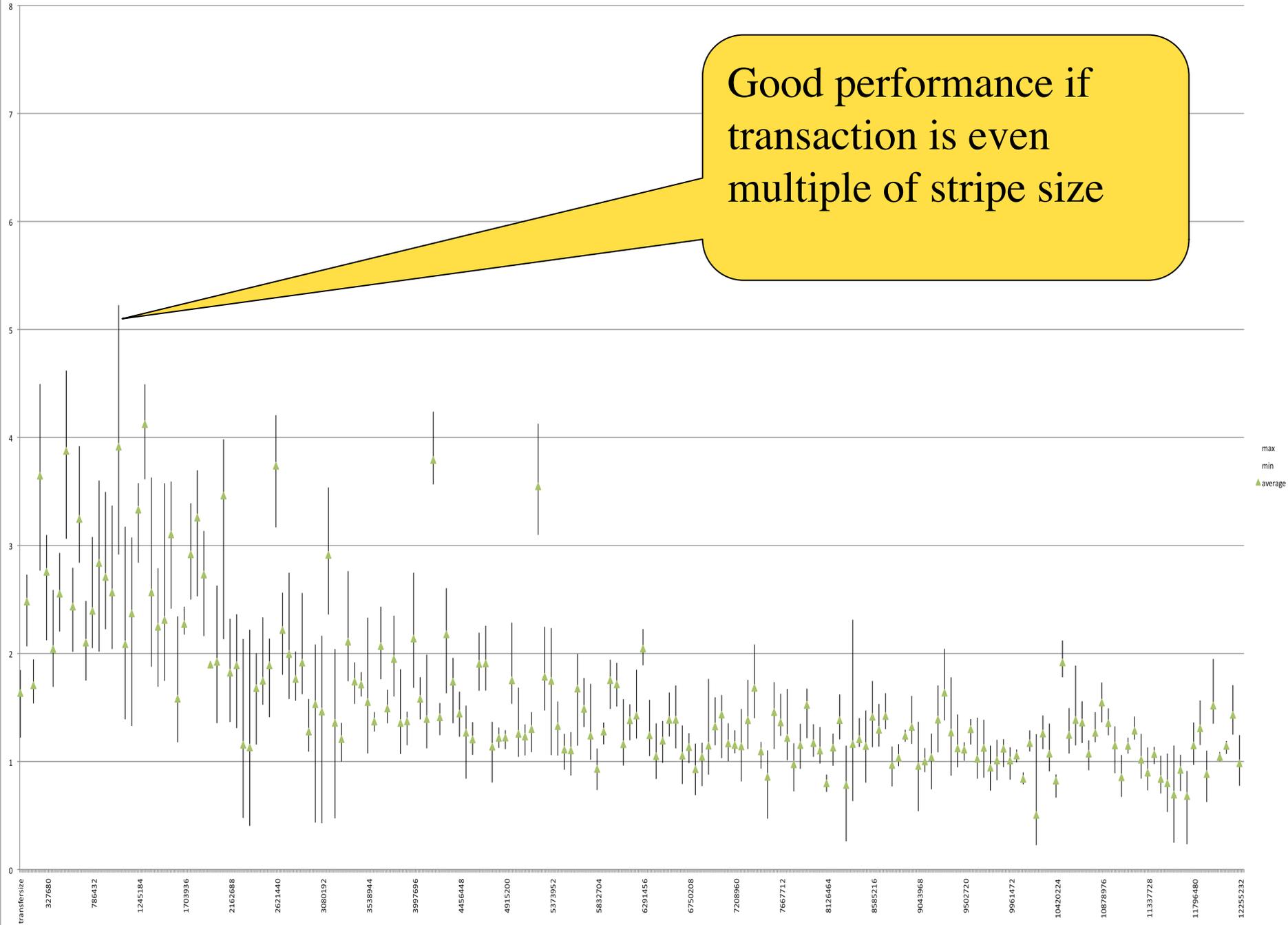


Performance Experiences

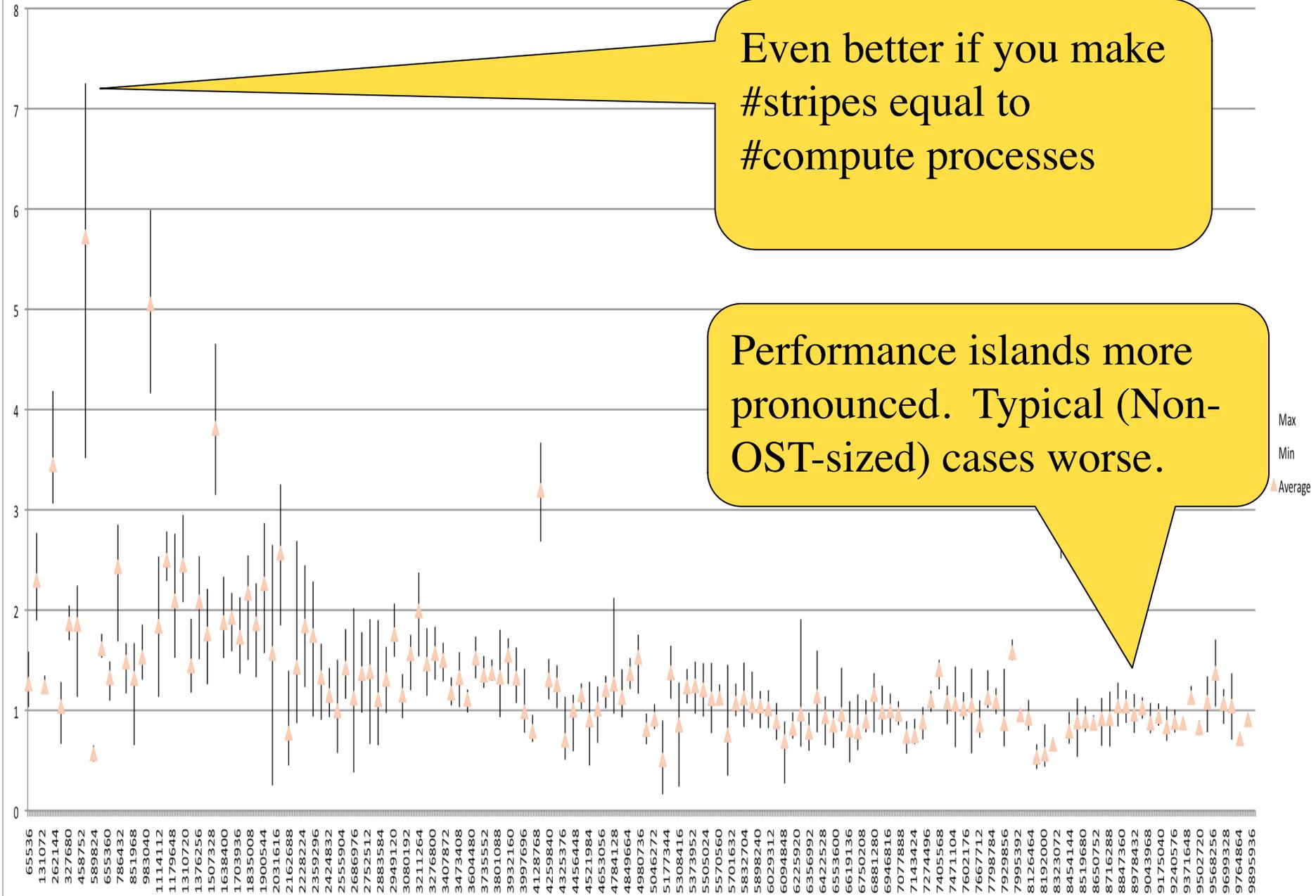
*(navigating a seemingly impossible
minefield of constraints)*

128 processors with 40 OSTs

Good performance if transaction is even multiple of stripe size



80 processors with 80 OSTs (Shane case)



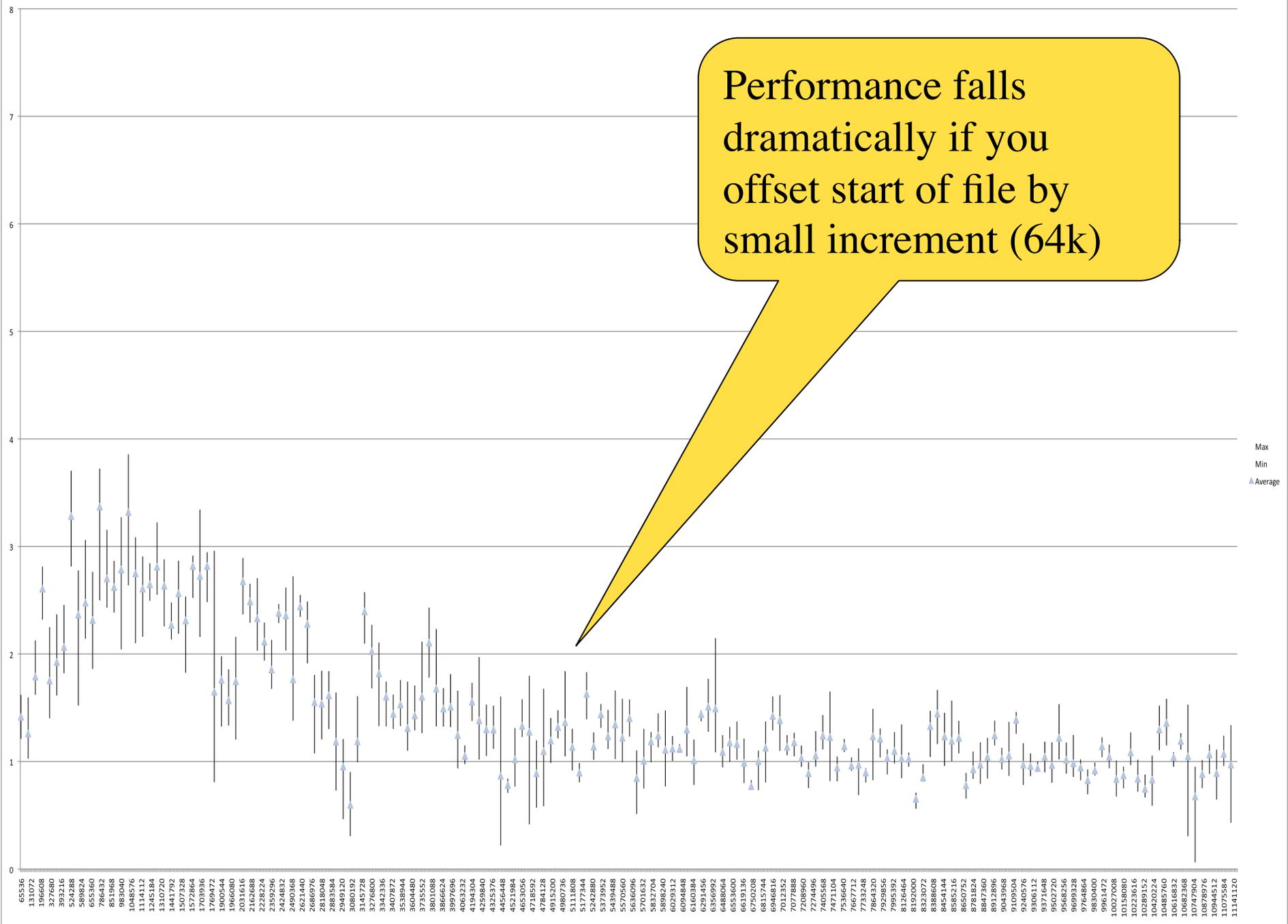
Even better if you make #stripes equal to #compute processes

Performance islands more pronounced. Typical (Non-OST-sized) cases worse.

Max
Min
Average

80 processors with 40 OST : offset file start by 64k

Performance falls dramatically if you offset start of file by small increment (64k)





Impractical to aim for such small “performance islands”

- **Transfer size for interleaved I/O must always match OST stripe width**
 - Difficult to constrain domain-decomposition to granularity of I/O
 - Compromises load balancing for particle codes
 - Not practical for AMR codes (load-balanced, but not practical to have exactly identical domain sizes)
- **Every compute node must write exactly aligned to OST boundary**
 - How is this feasible if users write metadata or headers to their files?
 - Difficult for high-level self-describing file formats
 - Not practical when domain-sizes are slightly non-uniform (such as AMR, particle load balancing, outer-boundary conditions for 3D grids)