From Xfiles to SAF: The NNSA Labs and The Early Pedigree of HDF5

MARK C. MILLER, LLNL

HUG 2023, OHIO SUPERCOMPUTER CENTER

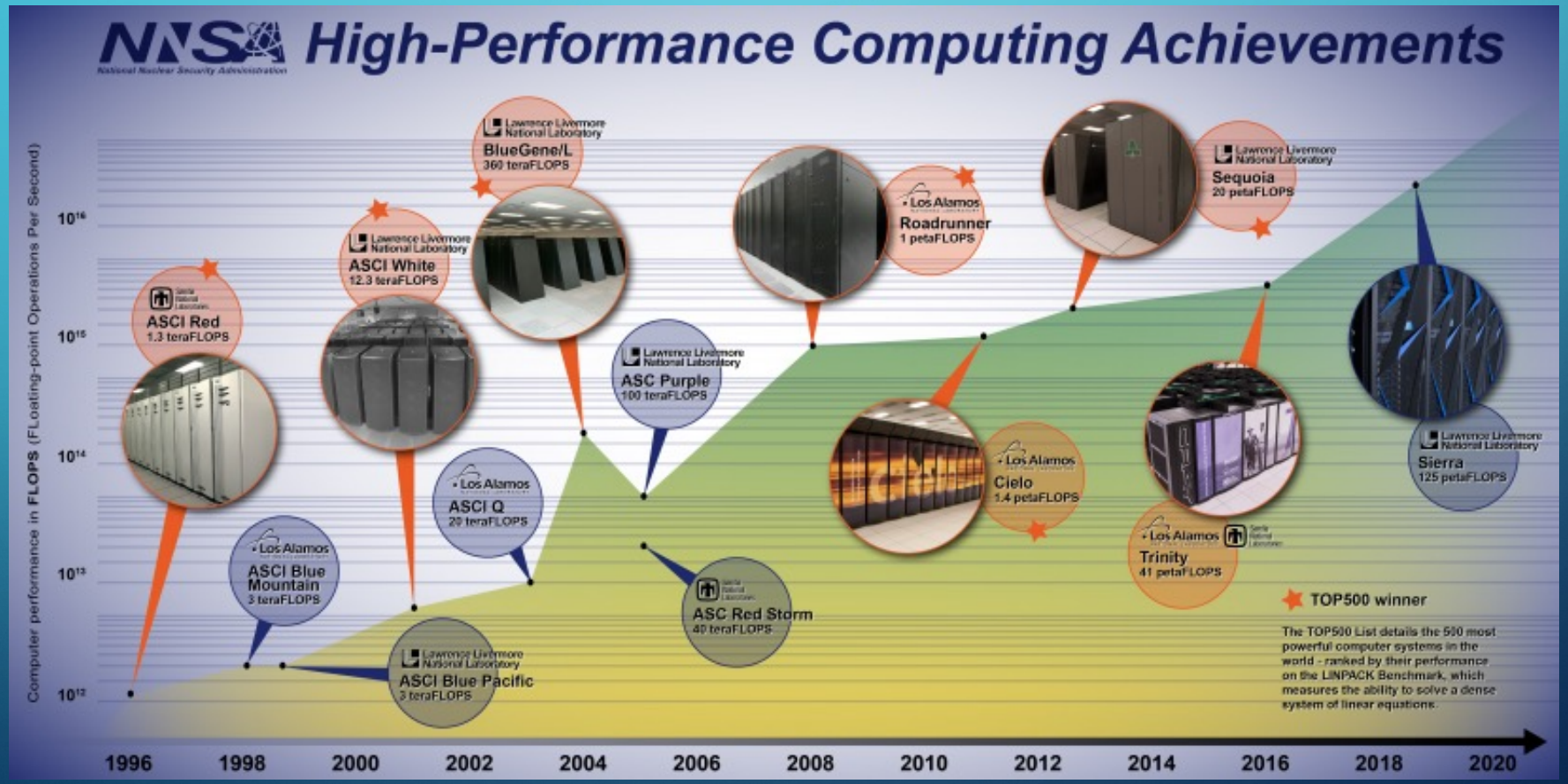
AUGUST 17, 2023

LLNL-PRES-853264

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Computing has been part of NNSA Labs From the very Beginning

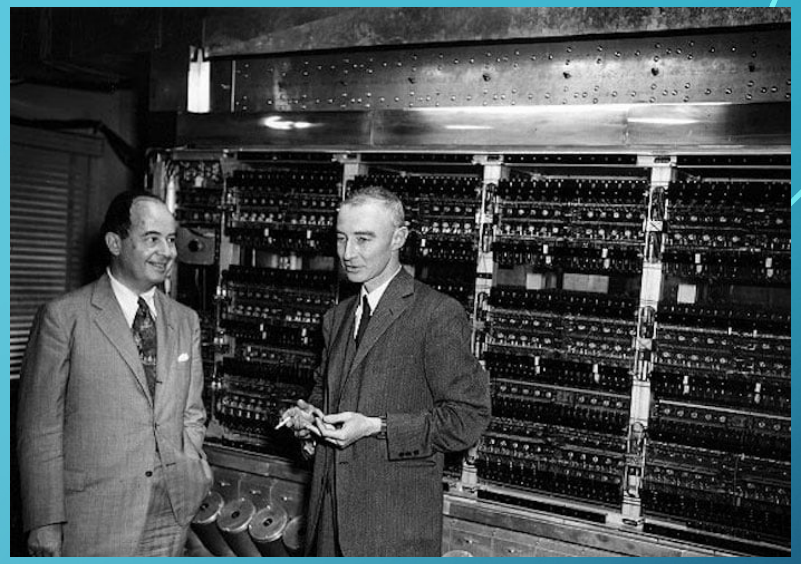


High Performance "Calculating" (HPc) And the Manhattan Project

- Needed to understand various physics…
  - Neutron mean free paths
  - Reactivities & critical densities of U and Pu
  - Detonation properties of explosives
- Some experimentation not possible
- Relied upon theoretical calculations
  - Initially used armies of women (kilo-girls)
  - Next, IBM punch card calculating machines
  - von Neumann & Feynman perfected the calculations (3 months → 3 weeks)

High Performance Computing (HPC) And The NNSA labs

Their first all (or mostly) Digital Machines



Lab	Year Opened	Earliest *Digital* Computing Capability	Computer Type
Los Alamos	1943	1952	MANIAC (LANL built)
Sandia	1948	~1960	IBM 7090 / CDC 3600
Livermore	1952	1953	UNIVAC 1

- National Nuclear Security Administration (NNSA)
  - Los Alamos & Livermore focus on design
  - Sandia tests, manufactures, deploys and maintains
- Sandia formed out of LANL Z division
  - Two sites near LANL and LLNL…no serious HPC until ~1990

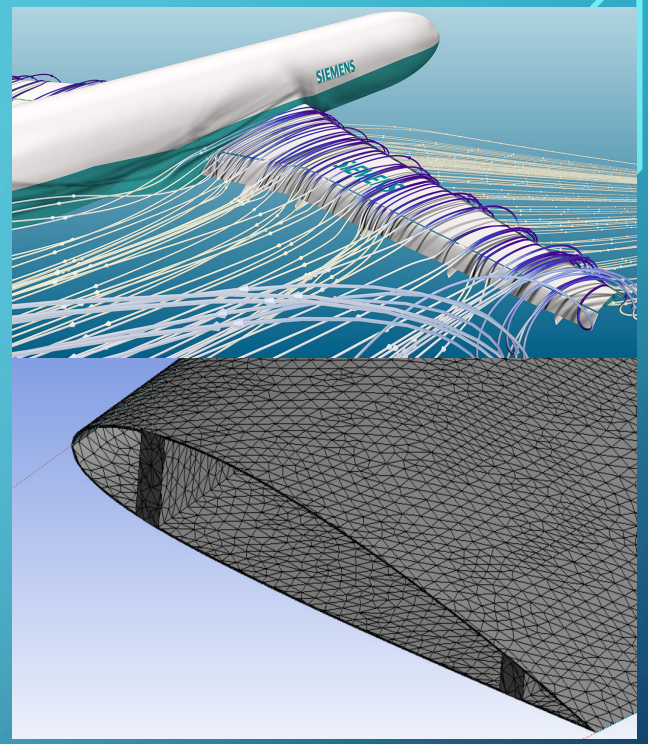
Accelerated Strategic Computing Initiative (ASCI) 1995

Replace testing with science-based modeling and simulation

- A key cross-cutting effort: The ASCI Data Models and Formats Effort (ASCI-DMF)
  - Provided high performance, *scalable*, parallel I/O
  - Supported a wide diversity of scientific data and be *extensible to new kinds*
  - Provided a *Rich feature set* (e.g. DIT operations, partial I/O, compression, etc.)
  - Ensured scientific data was *portable*, *exchangeable*, and *easily shareable*
  - Suitable for *restart* (bit-for-bit binary match to in-memory data) as well as *exchange*

Work with 3 radically different ASCI system architectures (Red, Blue-Pacific, Blue-Mountain)

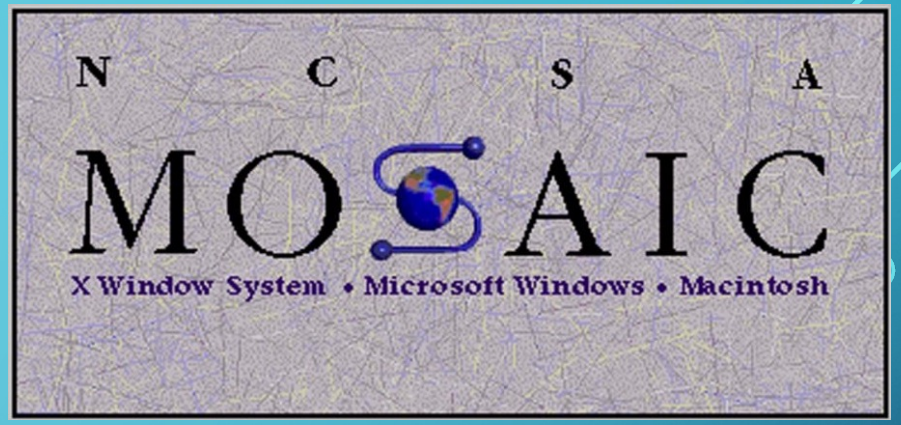
Scientific Data as *Numerical* Fields



- Ideal Fields: $F(x)$ in the continuous world
  - All scientific data is a real (or imagined) observable in the real, continuous world
  - Infinitely precise values at infinitely many points
  - Something you might actually measure by experiment if possible
  - Example: velocity of air flow over a wing
- Numerical Fields: $F(x) \cong \tilde{F}(\tilde{x}) = \sum_{i=0}^{N-1} f_i b_i(\tilde{x})$ in the *virtual* world
  - f_i are the degrees of freedom (DOFs) or *weights* in the numerical representation of the field
  - $b_i(x)$ are basis functions
  - The set of functions, $\{b_i(x)\}$, often determined by how x in $F(x)$ is *discretized* (e.g. meshed)

This is how ideal fields are represented (implemented) in a computer program

First ASCI-DMF Meeting at LLNL (Fall of 1996)



- NCSA's HDF Team (Mike, Quincey, Elena, Albert, others)
  - Linnea Cook was aware that HDF team was looking to update their format (from HDF4)
- Limit Point Systems (David Butler)
  - Mathematician and developer of Vector Bundle, Fiber Bundle and Sheaf Data Models
- LLNL (Silo/PDB), SNL (Exodus/netCDF), LANL (XFiles)

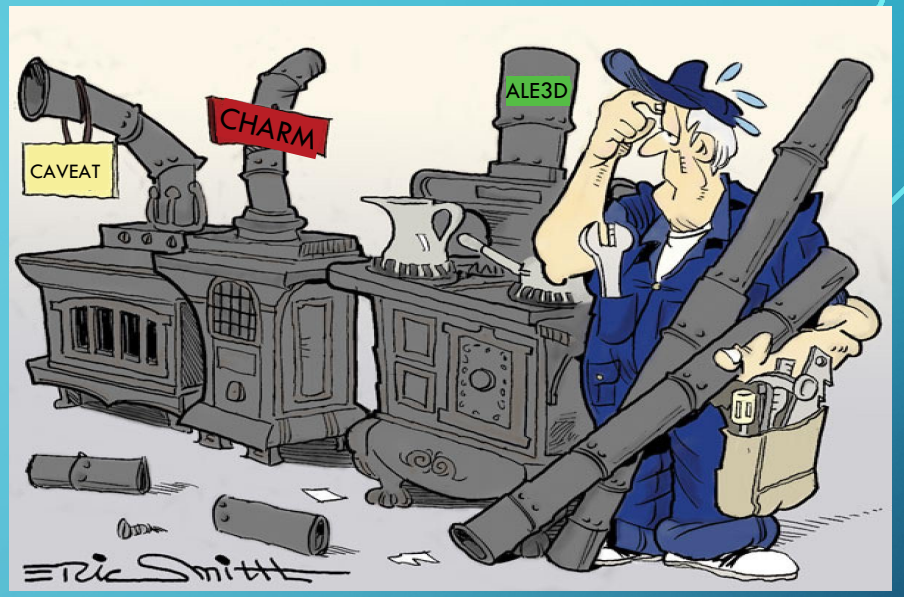
The Sets and Fields (SAF) data model grew out of this effort

A Brief look at State of NNSA Lab's I/O technologies in mid-1990s

- LANL (X Division Code Linking file)
  - Primarily a serialization format…fixed format, ordering, float formats, no random access
- LLNL (Silo/PDB) – Mainly Structured Codes for Hydrodynamics
  - PDB provides lower level arrays of ints/floats/etc and random access
- SNL (Exodus II/netCDF) – Mainly Unstructured codes for Mechanics
  - netCDF provides lower level arrays of ints/floats/etc (pseudo-random access)

Sandia was only lab using low-level format that was developed externally and used by a larger community of users and had a fledgling ecosystem of tools (netCDF)

1960 – 1980: The I/O Stovepipe Era



- Each application writes its own unique file
  - Many companion tools
    - Data management, browsing, editing, differencing, analysis, plotting and viz. tools
  - Application and all companion tools are *integrated* together via this format
- Data Exchanges among these tools used common (app-specific) file format
  - Every tool written to conform to the application's file format
  - Works fine within a stovepipe
- Exchanges across stovepipes ➔ Code "Linkers"
  - Brute force data exchange

1960 – 1980 : Improvements in HPC Ecosystem

- Improvements in Languages…

  …eliminate need for custom pre-processors tools

- Improvements in Operating & Batch systems…

  …eliminate need for custom job scheduling and control tools

- Improvements in File systems…

  …eliminate need for custom data storage, management & browsing tools

- What about data differencing, analysis, plotting, viz.?
  - These remained a serious challenge

1980 – 2000 : Emergence of two Kinds of General Purpose I/O Libraries

- Kind 1: Data Structure Abstractions
  - Read/Write an Array, a Struct or a Linked-List
  - CDF (Common Data Format), HDF (Hierarchical Data Format), PDB (Portabl DataBase)
- Kind 2: Computational Modeling Abstractions
  - Read/Write a Mesh, a Field, a Material a Time history
  - Exodus (SNL-1992), Silo (LLNL-1994), CDMLib (LANL-1998)
  - Typically these take advantage of (e.g. Use) Kind 1 libraries (Exodus/netCDF, Silo/PDB)

Software Engineering vs. “Data” Engineering

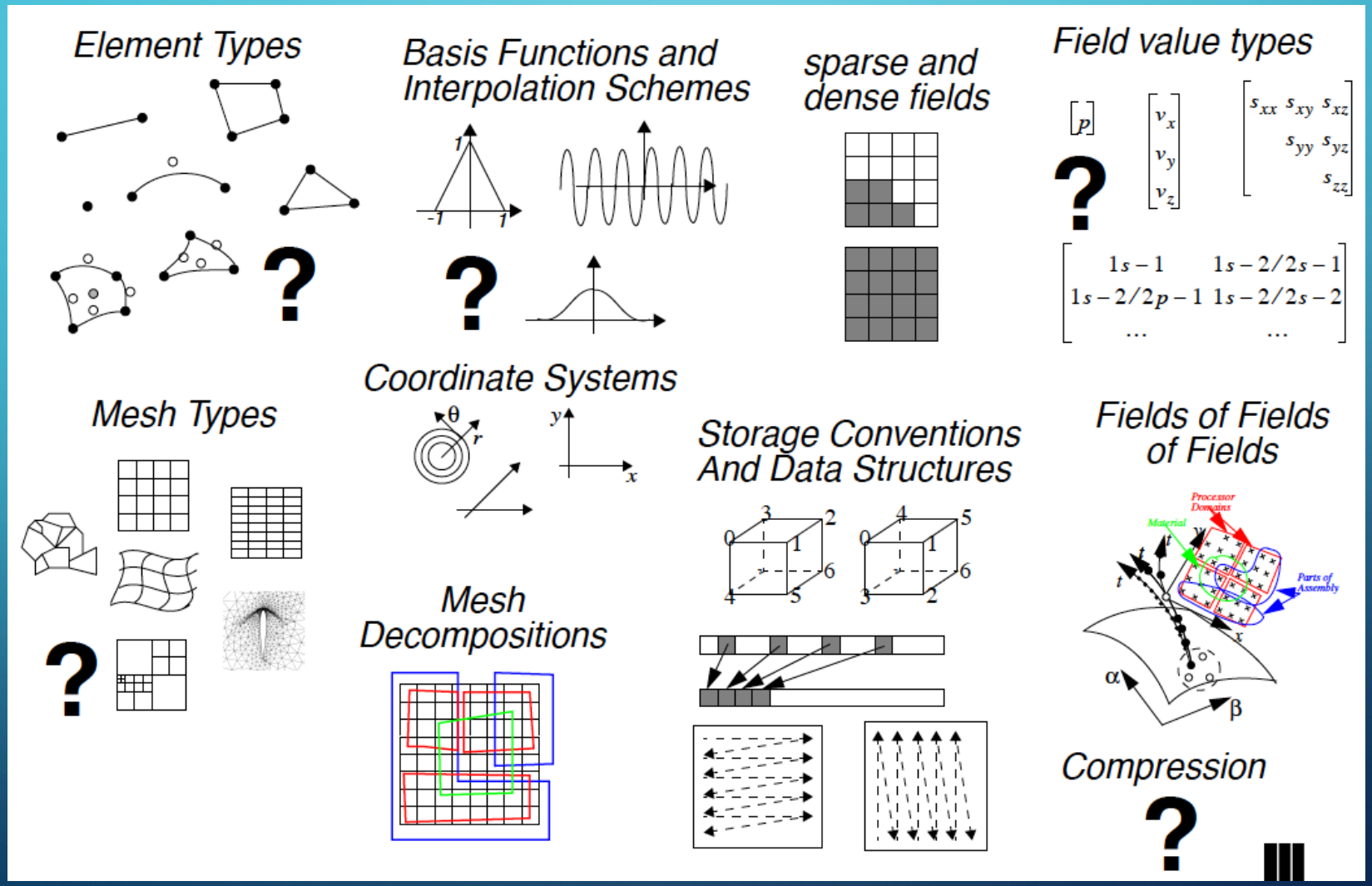
The *level of abstraction* at which data is characterized…

…governs entirely a community’s ability
to write general purpose tools to process it.

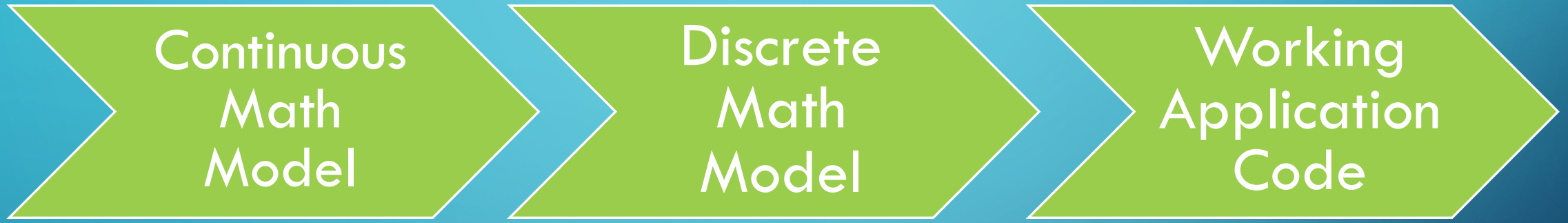
I/O isn’t about I/O…Its about Data Exchange

LLNL-PRES-853264

Describing Scientific Data Is Challenging



The Design Life Cycle of Scientific Computing Applications



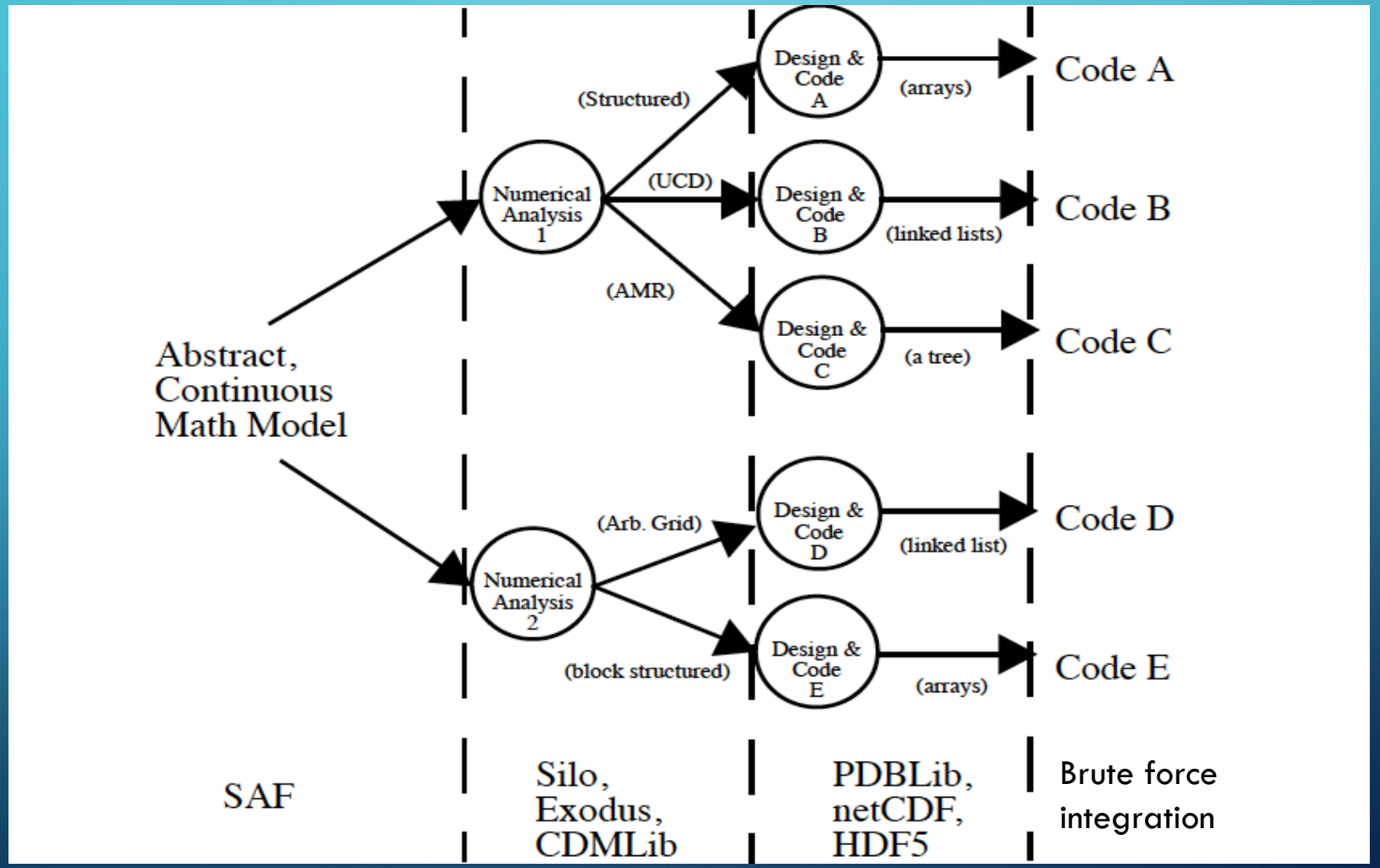
- Sets
- Fields
- Relations (PDEs)

- Mesh Elements
- Variable DOFs
- Basis functions

- Types/Weights
- Buckets of #s
- Languages

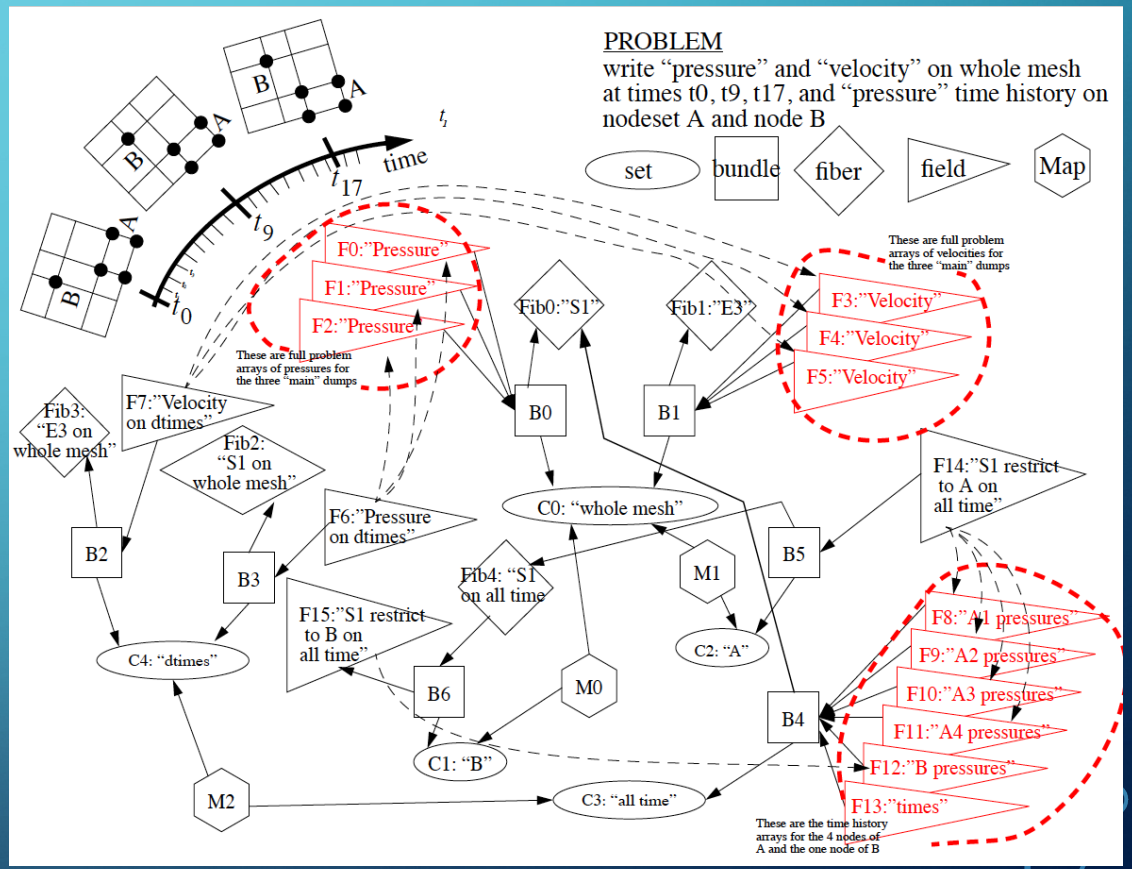
At each phase, different teams will make different choices

Abstraction is key to Wide Scale Integration of Scientific Software



Unfortunately, An Abstract Math Model can present A Conceptual Hurdle

- Vector Bundles
- Fiber Bundles
- Sheafs
- Multi-Sheafs
- The Sets and Fields (SAF) data model aimed to simplify the abstraction



LLNL-PRES-853264
LLNL-PRES-687907

HDF5 features inspired by or related to Needs of NNSA Labs

- Numeric formats and type conversions (from Silo/PDB)
  - Pre-defined HDF5 numeric types (e.g. CRAY, INTEL, IEEE, ALPHA, etc.)
- Groups, mounts and symlinks (from file system metaphor)
- MIF Parallel I/O Paradigm (from Silo/PDB parallel I/O)
- MPI-IO (from Exodus II need for scalable I/O to single file)
- Data spaces (from Exodus II nodesets/sidesets and SAF data model)
- Virtual File Drivers (from file systems / dump strategies used @ NNSA labs)
  - Family (Windows dev), split (ASCI RED, ASCI Blue Mountain), MPI-IO (ASCI Blue Pacific)
- File “Images” (2012) (from need to leverage I/O code for MPI messaging)
- Virtual Datasets (2016) (from Silo multi-block objects)

LLNL-PRES-853264
LLNL-PRES-687907

The HDF Team got a Front Row Seat at NNSA, Tri-Lab Data Integration Efforts

- Many meetings 1996-2001
- Except in isolated, one-off instances, the three labs had never shared data
- Three different experiences and expertise in integrating via files
  - Exodus II/netCDF, Silo/PDB, XFiles
- Three different computing and file system architectures
  - ASCI Red, ASCI Blue Pacific, ASCI Blue Mountain
- There was also a parallel “Common Viz. Tool” effort

LLNL-PRES-853264
LLNL-AR-743064

Role of HDF Team in ASCI DMF and Role of ASCI DMF in HDF5

- The HDF Team served as an objective third party
- Took the best ideas coming from all three labs and used those to inform the design of HDF5
- Found a way to incorporate the only available, portable, scalable, parallel interface (MPI-IO)
- Found other user communities who wanted similar features (leverage)
- Cultivated and curated HDF5 and associated eco-system to what it is today

LLNL-PRES-853264
LLNL-AR-743064

What I treasure most about my HDF5 experience

- The people I've worked with (across NNSA and of course with THG)
- The small role I got to play in helping HDF5 to come about
- The impact HDF5 has had on various communities
- The rock solid API stability THG has maintained
- The commitment to providing quality, open source software
- The really cool stuff I still get to do (and impress others with) with HDF5
- The "idea" map of HDF5 future features
- The knowledge and experience we've gained to start developing HDF6 🤪

Happy Birthday, HDF5. 25 Years is a big deal!!

- Some software projects "succeed" for inexplicable reasons
- Most software projects succeed because of the blood, sweat and tears of the people who support it.

Thank you Team HDF5 for your dedication and hard work

End

LLNL-PRES-853264

Backup Slides

LLNL-PRES-853264

Home Grown vs. Community Grown

- Home grown is a heavy lift
  - Every new tool requires development effort and time that sponsor has to pay for
  - But, you control everything including all important I/O performance
  - Can quickly and reliably address show-stoppers
- Community standard leverages community developed tools
  - But, you must use the technology in a way the community expects
    - Counter-example: Exodus II files in netCDF … not a natural fit due to unstructured gridding
  - How will show-stoppers be handled
  - What if community doesn't achieve sufficient critical mass?
  - What if community's interests and NNSA lab's interest don't sufficiently overlap?

Restart vs. Plot (vs. Time Histories Vs. Linkers)

- Restart files require bit-for-bit identical match with memory resident data
  - PDB still read across disparate CPU architectures if necessary
- Plot files can be more flexible (all of the data for some of the times)
  - Most often single precision is fine
  - Subset of variables code used
- Time histories (some of the data for all of the times)
  - Often just ascii csv files
- Inter-code linker files
  - Used to integrate two codes (structured grid and unstructured grid)
  - Generally requires constant care and feeding as either code evolved

Can lead to four separate I/O interfaces on a code

Standardized interface vs. Standardized Container

- File format specification is more of a standardized container (byte layout)
  - Can avoid dependencies if your willing to own the serialization code
  - `obj->precision = 8;`
- API Specification is a standardized interface (function)
  - Underlying storage can be anything (HDF5, netCDF, XML, etc.) and anywhere
  - `obj->SetPrecision(8);`
  - Like a level of indirection over the standardized container approach
- Standardizing on both is really the ideal

Binary vs. ASCII data

- Base 10, human readable ASCII not bit-for-bit reproducible with mem-resident data
  - Hadn't considered base64 encodings back then (not sure why…but are not human readable)
- XDR was too slow for decent I/O performance
- Constant conversion (binary to ascii on write, etc.) was bandwidth limiter … not so much nowadays
- Binary data not machine portable
  - Until we had PDB and/or IEEE-754 standardization
- Variable length ASCII strings impede random access

```
solid DataSet1
   facet normal 0.000000e+000 0.000000e+000 1.000000e+000
      outer loop
         vertex 2.114699e+002 1.089402e+002 0.000000e+000
         vertex 2.352096e+002 1.437085e+002 0.000000e+000
         vertex 1.798614e+002 1.305223e+002 0.000000e+000
      endloop
   endfacet
   facet normal 0.000000e+000 0.000000e+000 1.000000e+000
      outer loop
         vertex 1.798614e+002 1.305223e+002 0.000000e+000
         vertex 2.352096e+002 1.437085e+002 0.000000e+000
         vertex 2.000674e+002 1.837107e+002 0.000000e+000
      endloop
   endfacet
   facet normal 0.000000e+000 0.000000e+000 1.000000e+000
      outer loop
         vertex 1.798614e+002 1.305223e+002 0.000000e+000
         vertex 2.000674e+002 1.837107e+002 0.000000e+000
         vertex 1.617475e+002 1.711867e+002 0.000000e+000
      endloop
   endfacet
   facet normal 0.000000e+000 0.000000e+000 1.000000e+000
```

Single-sided vs. Double-sided I/O

- Single sided: Application does the work and has the logic to decide when to move data between memory and disk
- Double-sided: Every exchange of data, even I/O, involves two clients
  - Application's publish their data for prospective clients
  - Clients decide if and when to perform an exchange
  - Even for I/O to a file, a Restart Manager or a Plot Manager client handle the I/O
  - Moves all of the logic for deciding when and how to move data from application to be shared with other applications

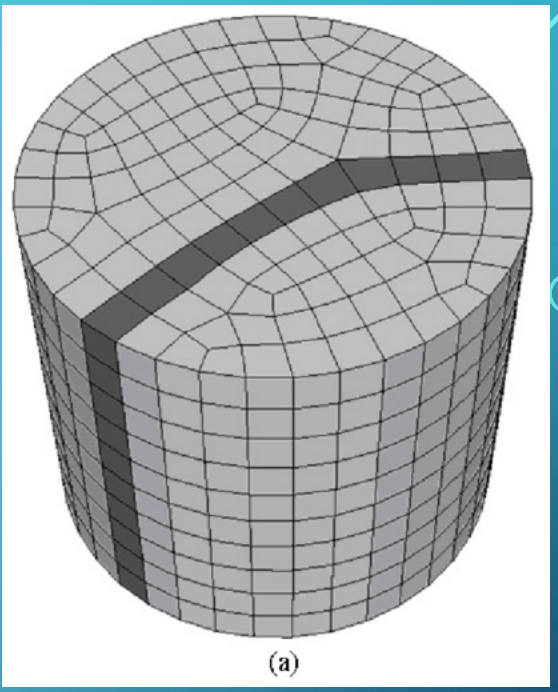
Cray vs. SGI

- Computed on Crays, visualized on SGIs
- Restart files stayed on Crays
- Plot files migrated to SGIs
- Had to convert floats, byte swap ints, and handle “Cray pointers”

Hub and Spoke vs. Parameterized Formats

- Hub and Spoke
  - Conversion on write and conversion on read for every client not 1:1 with HUB
  - Conversion costs worsen for clients semantically “further” from HUB
  - Likely HUB cannot be bit-for-bit identical for all client’s data (impacts restart utility)
- Parameterized
  - Parameterizations can be complicated (e.g. row-major vs. col.-major)
  - Conversions happen only when producer and consumer incompatible
  - Restart use cases covered whenever parameterizations fit

Structured vs. Unstructured Meshes



- For structured meshes, coordinates and connectivities are implied (not explicitly stored)
- For unstructured meshes, they are explicit
- For a 3D mesh of hexahedra, the mesh costs 11 problem-sized arrays
  - 3 coordinate arrays for X, Y and Z coordinates of each node
  - 8 connectivity arrays for the 8 corners of each hexahedra

>10x memory hit in viz. apps if do not support structured meshes *natively*

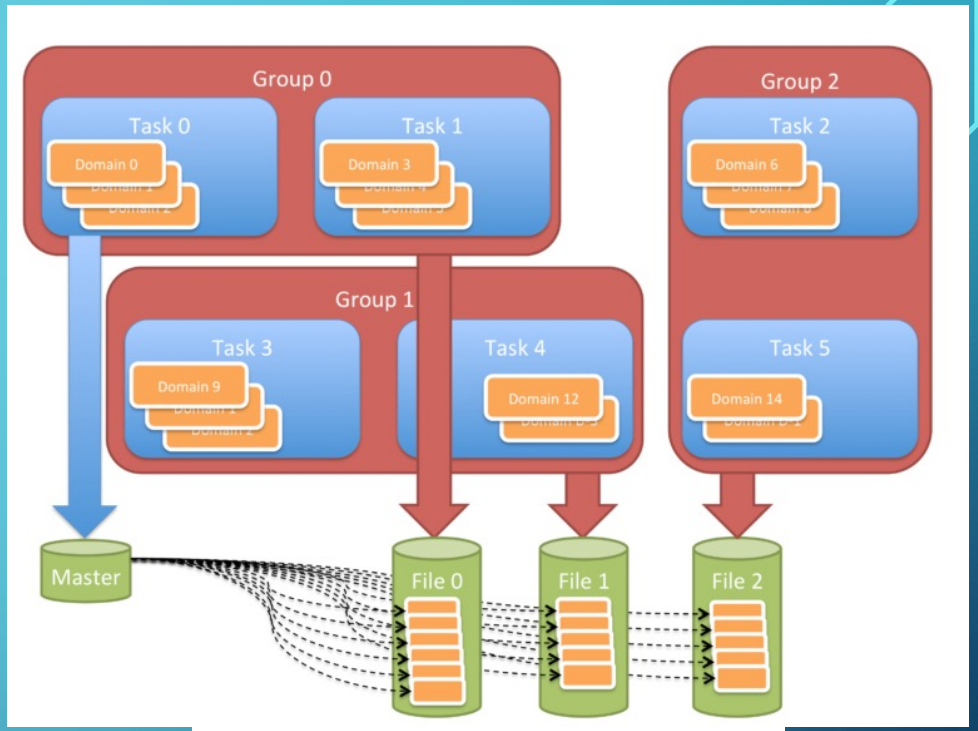
Problem-sized (raw) vs. Log-Problem-sized or Fixed-sized (meta) Data

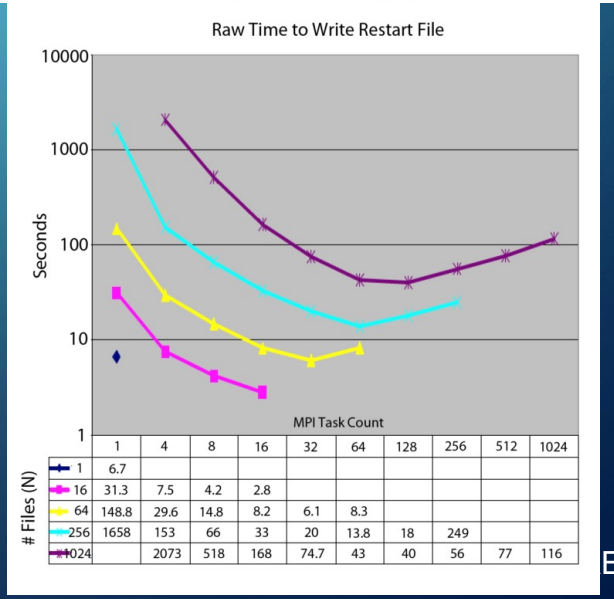
- Some data grows proportionally as we scale problem up (Fields)
  - Basically anything having to do with the main mesh and its variables
  - Coordinates, connectivities, physics variables
- Some data grows by other (slower growing) factors (funky in-betweeners)
  - Compute partition & connectivities (neighbor info) (# of processors, cores/threads, GPUs)
  - Parent/child relationships in fine-grained AMR
  - Surface to volume relations and mixing materials at late time
  - Coordinates of rectilinear (structured) meshes
- Some data fixed or so slowly growing treating as fixed is ok (Sets)

Multiple Independent File (MIF) vs. Single Shared File (SSF) Parallel I/O

- Simpler I/O coding (module baton passing)
- Application managed & throttled concurrency
- Works on any file system (laptop to LCF)
- Handles disparities in data across MPI ranks
- Easily combined with HDF5 DIT operations
- No implied global re-ordering on read/write
- Entirely analogous to Big Data I/O "shards"

All tools designed to handle "domain overload"





Menu-oriented vs. Model-oriented Scientific Data Description

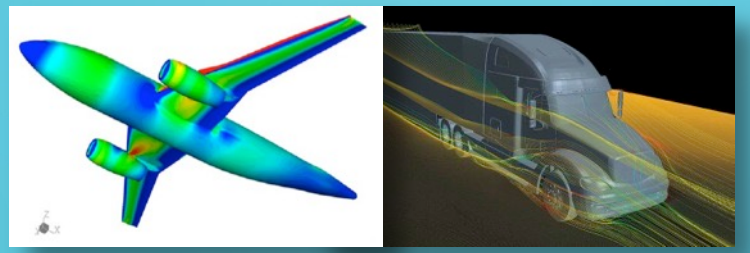
MENU ORIENTED

- Fixed set of objects on the menu (API methods)
- UCD-mesh, point-mesh, xy-curve, zone- or node-centered scalar or vector variable…
- Find object on menu that matches your data
  - Some conversion to/from menu on read/write
- Or, ask chef to cook up a new menu item
- API and system complexity grows as user base grows

MODEL ORIENTED

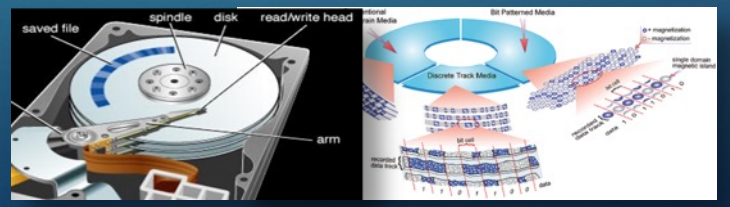
- Fixed (and small) set of modeling primitives
- Build up representations of any object from these primitives (literally *model* your data)
- API and system complexity remain more stable as user base grows
- Modeling primitives are necessarily math-oriented and require deeper understanding of abstract mathematical concepts than most developers are maybe willing to try

Levels of Abstraction of Scientific Data

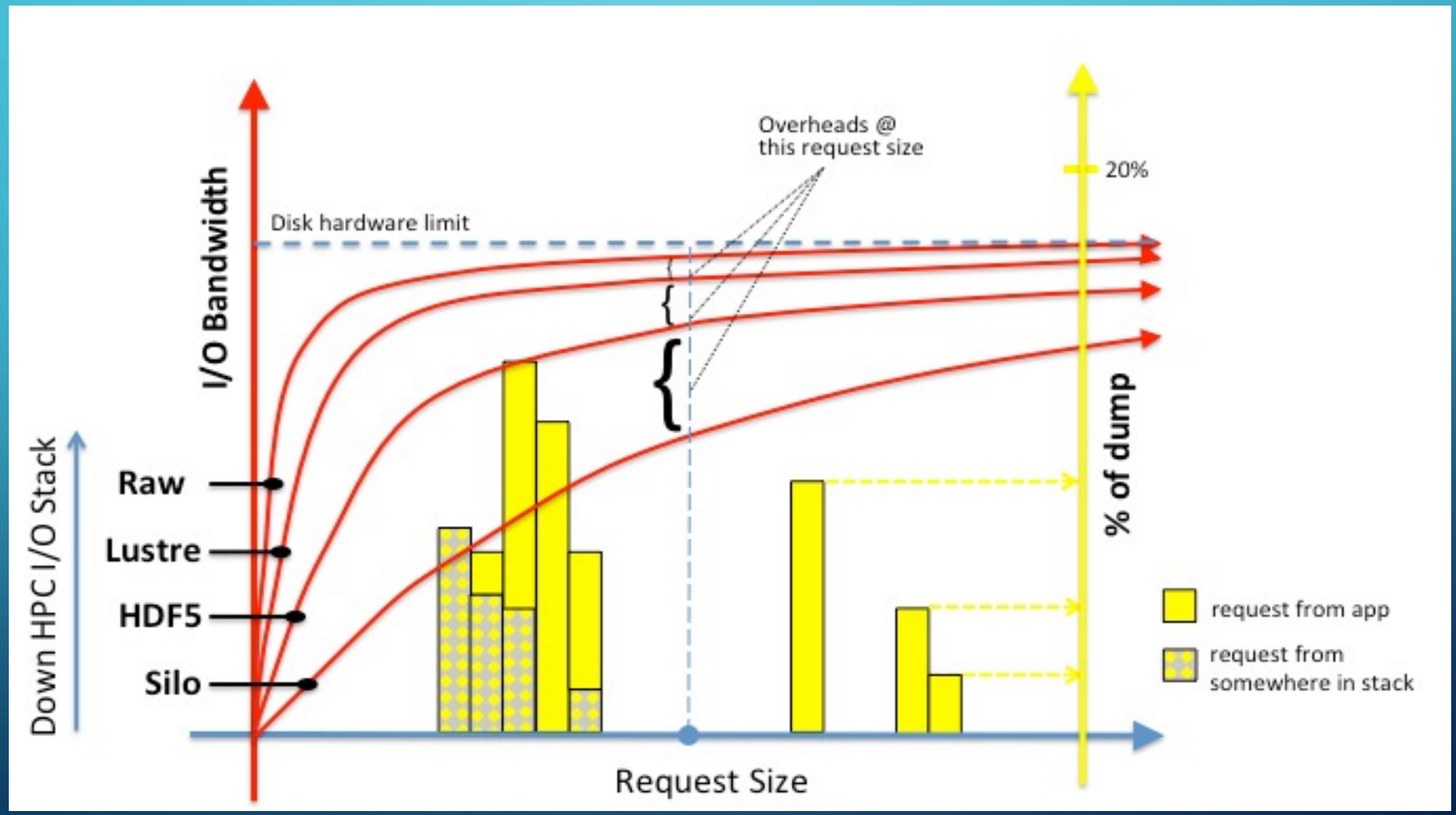


Increasing LOA

Abstractions	Objects	Implementations
Real World Phenomena	Physics, Chemistry, Materials…	ALE3D, Albany, LAMPS
Continuous Mathematics	PDEs, Fields, Topologies, Manifolds	MFEM, FiberBundles, Sheafs, SAF
Discrete Numerical Models	Meshes, Materials, Variables	Silo, LibMesh, Exodus, ITAPS, VTK
Prog. Language Data Constructs	Arrays, Structs, Lists, Trees	HDF5, netCDF, ArrayIO, PDB
Primary Storage (Main Mem.)	Ints, Floats, Pointers, Offsets, Lengths	MPI-IO, XDR, stdio, aio, mmap
File System	Files, Dirs, Links, Permissions, Modes	POSIX IO
Secondary Storage (logical)	Pages, Inodes, FATs, OSTs, OSDs	GPFS, HDFS, Lustre / ext2/3, zfs, xfs, hfs
Secondary Storage (physical)	Bits, Volumes, Sectors, Tracks	hd, sd, cd, dvd, tape, ramdisk, flash



Data Abstractions vs. I/O Performance



Fast To Implement vs. Fast I/O

- Structs of structs of structs…serialization,
  or, C++ object serialization,
  or, scientific object serialization
  - Use HDF5 “Group” for aggregation
  - Use HDF5 ”dataset” for struct/obj data
  - Massively simplified I/O coding
  - Massively inefficient I/O performance