DataMods

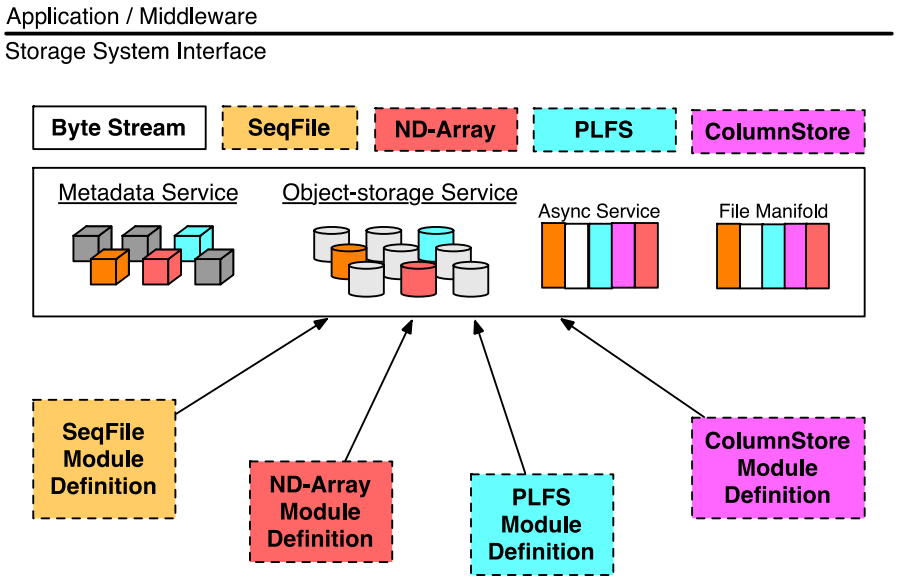
Programmable File System Services

Noah Watkins*, Carlos Maltzahn, Scott Brandt
*UC Santa Cruz, *Inktank*

Adam Manzanares
California State University, Chico

Talk Agenda

1. Middleware and modern IO stacks
2. Services in *middleware* and *parallel file systems*
3. Avoid duplicating work with DataMods
4. Case study: Checkpoint/restart

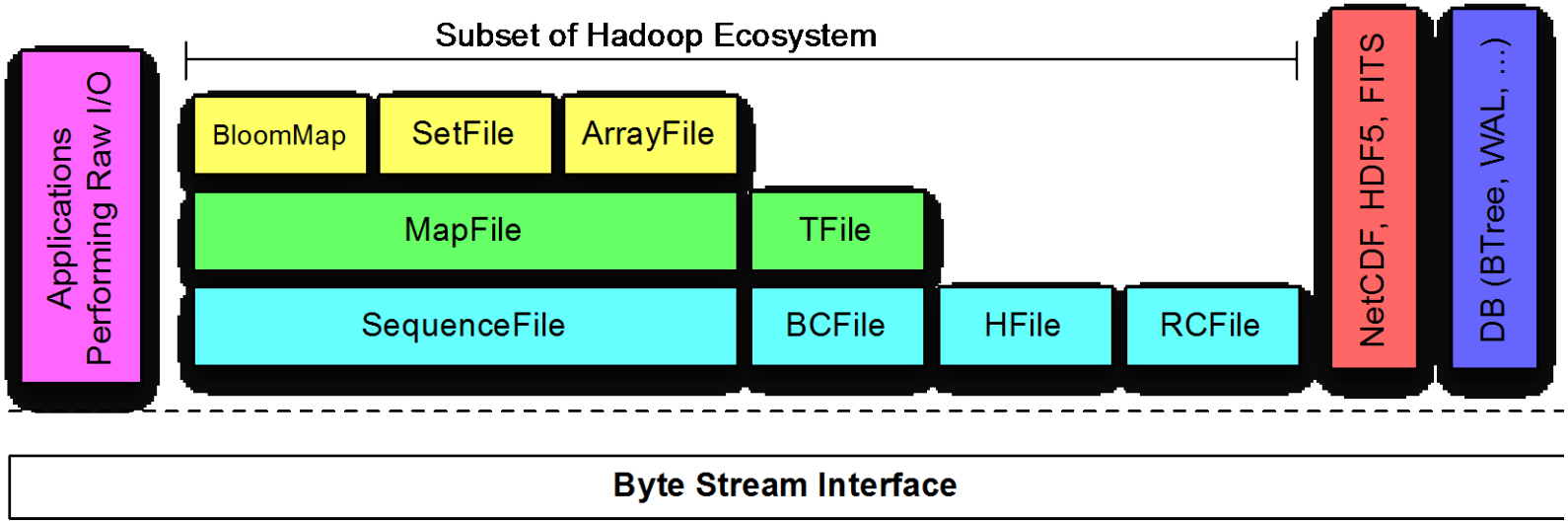


Why DataMods?

- Applications struggle to scale on POSIX I/O
- Parallel FS rarely provide other interfaces
  - POSIX I/O designed to prevent lock-in
- Open-source PFS are now available
  - Ability to avoid lock-in
- Can we generalize PFS services to provide new behavior to new users?

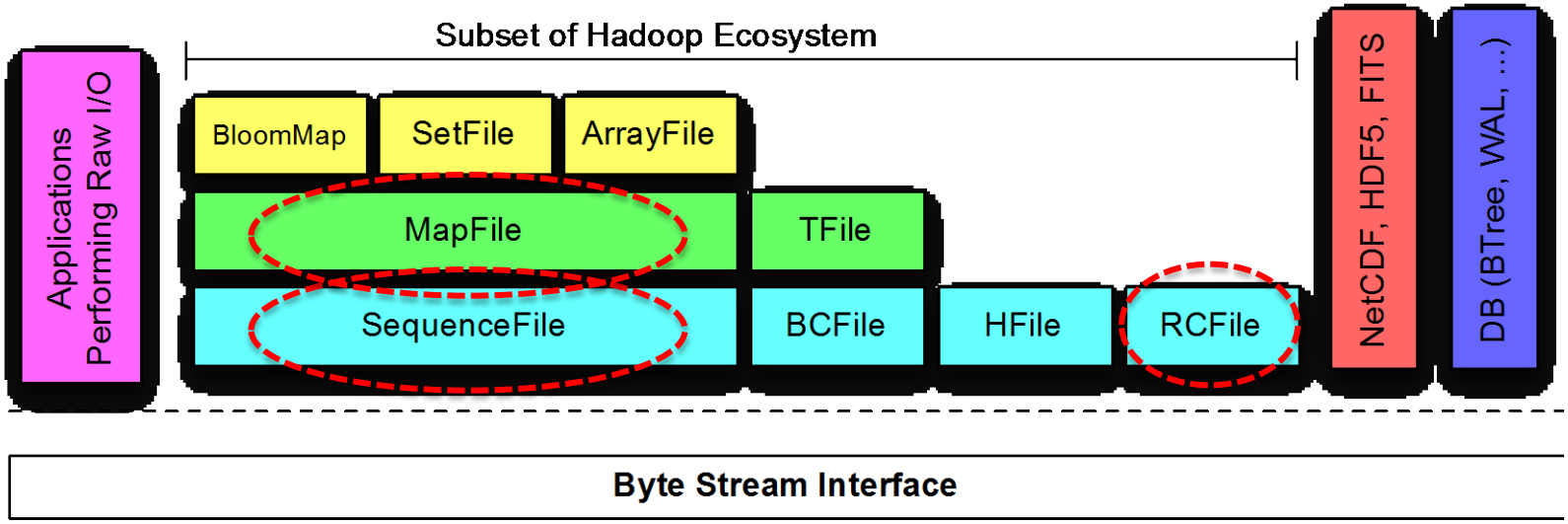
Application Middleware

- Complex data models and interfaces
- Difficult to work directly with simple byte stream
- Middleware maps the complex onto the simple



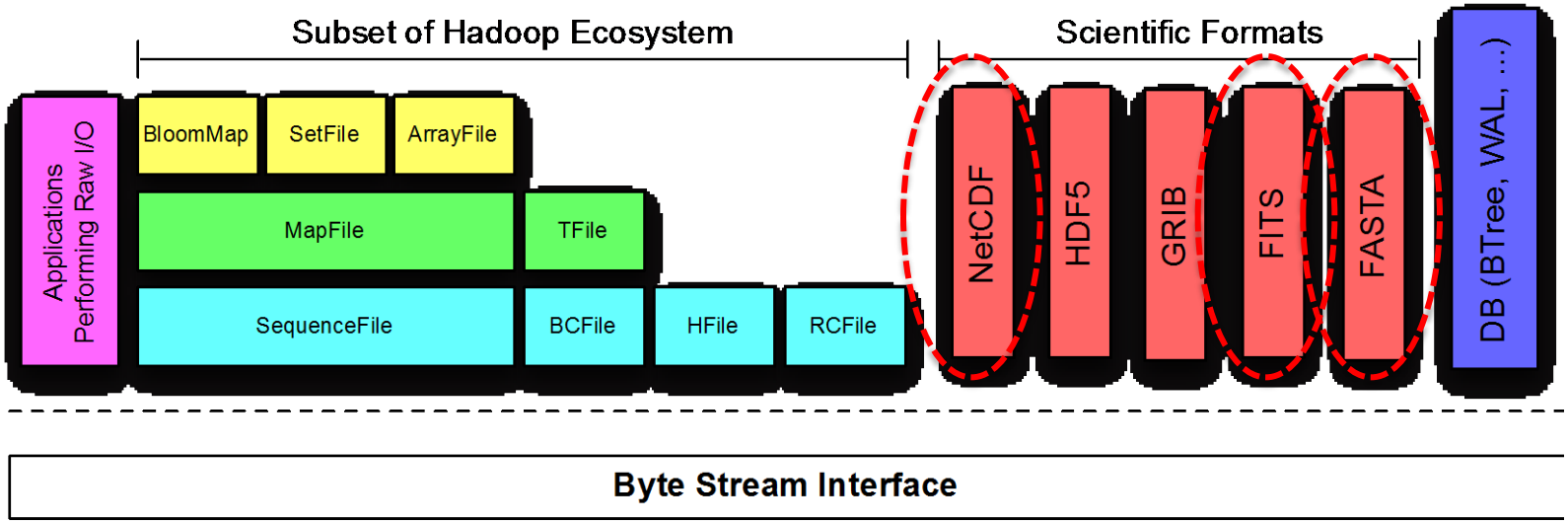
Middleware ~~Complexity~~ Bloat

- Hadoop and "Big Data" data models
  - Ordered key/value pairs stored in file
  - Dictionary for random key-oriented access
  - Common table abstractions



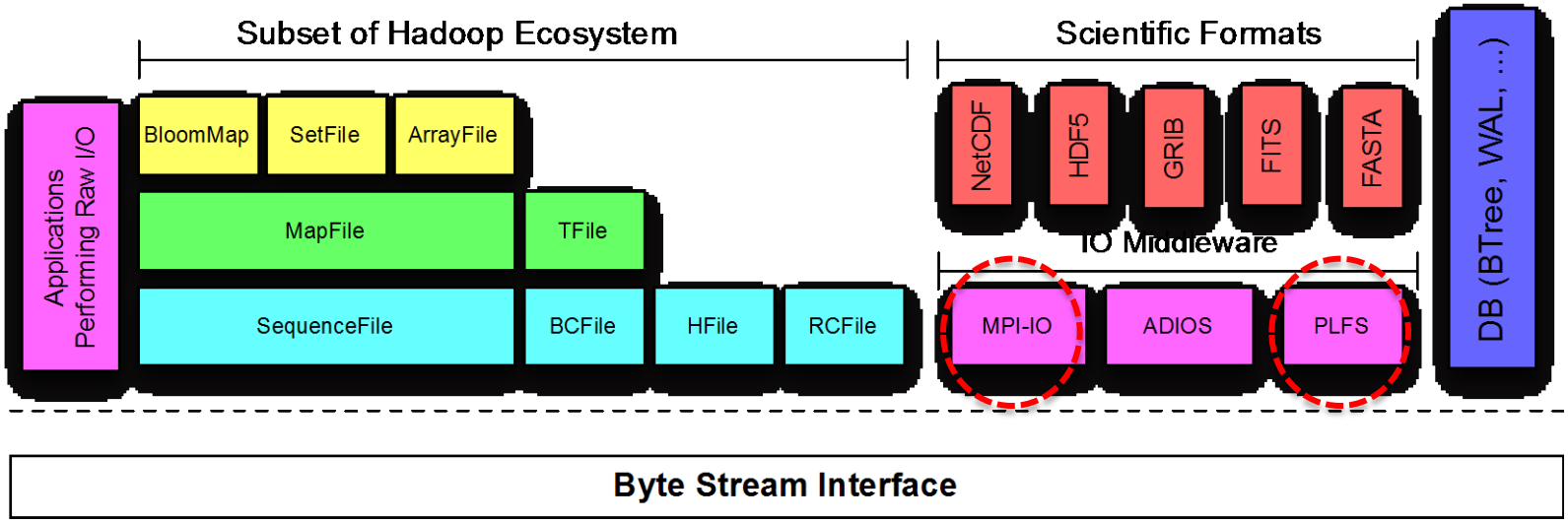
Middleware ~~Complexity~~ Bloat

- Scientific data
  - Multi-dimensional arrays
  - Imaging
  - Genomics



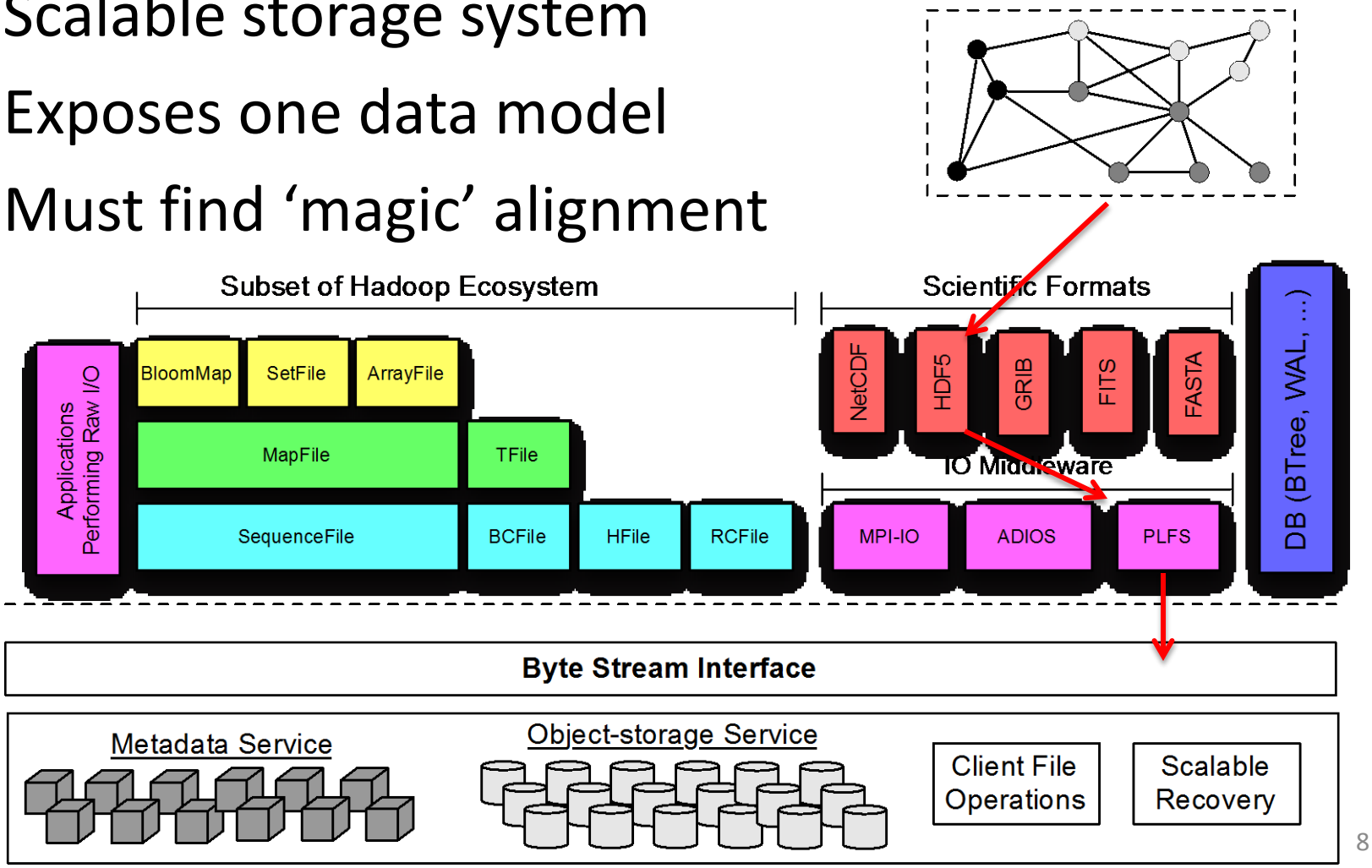
Middleware ~~Complexity~~ Bloat

- IO Middleware
  - Low-level data models and I/O optimization
  - Transformative I/O avoids POSIX limitations



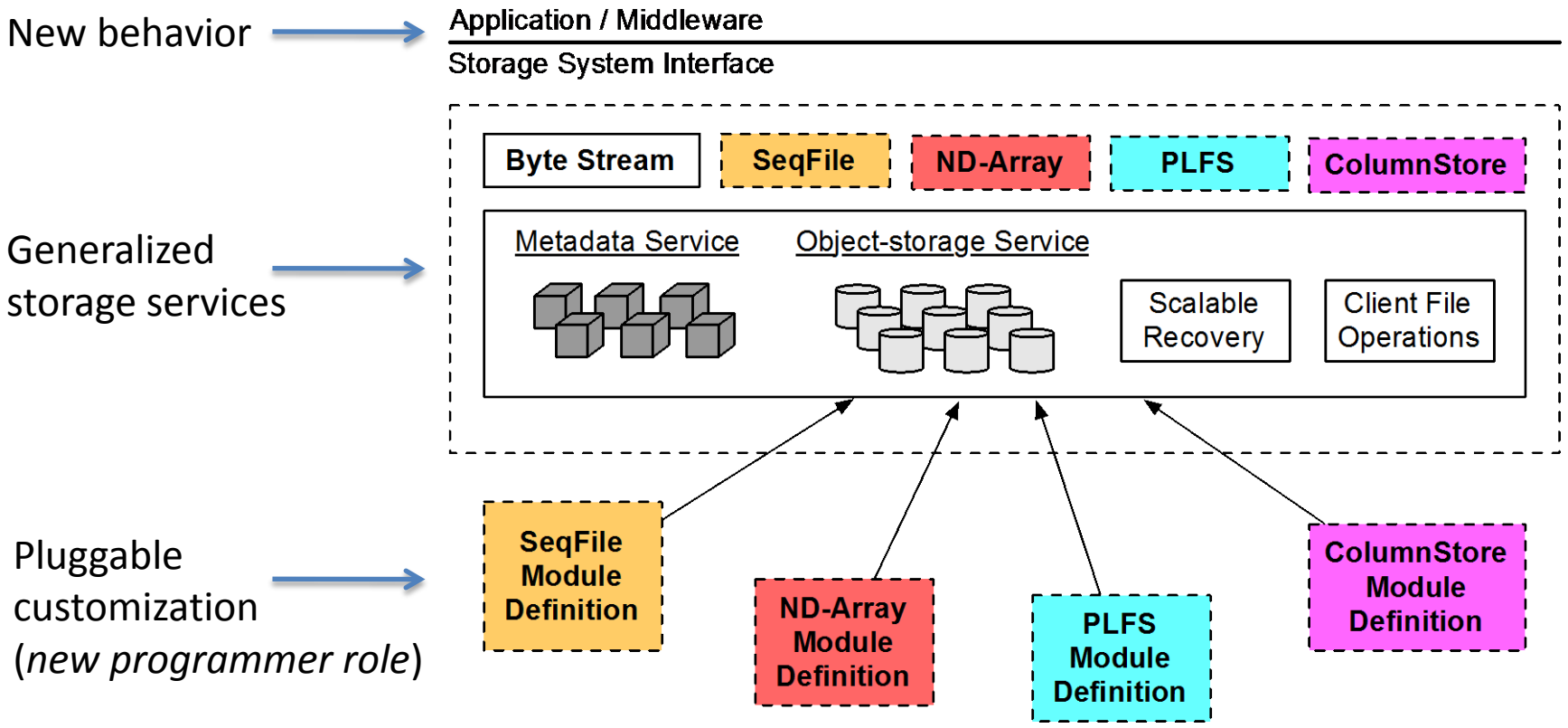
Middleware Scalability Challenges

- Scalable storage system
- Exposes one data model
- Must find 'magic' alignment

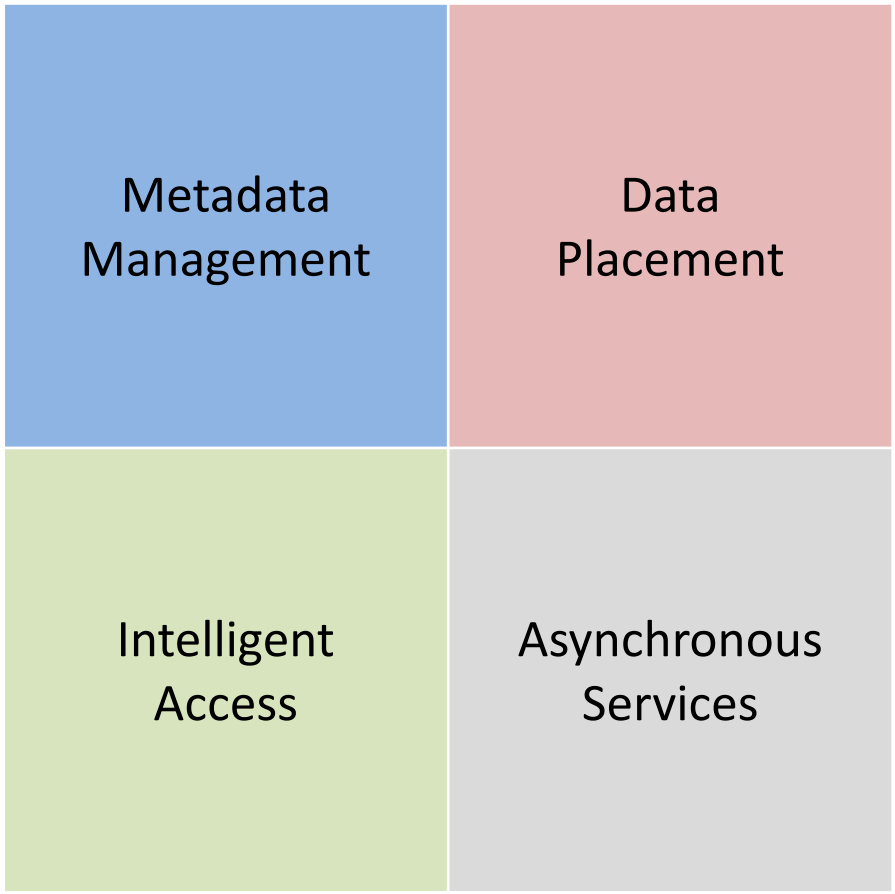


Data Model Modules

- **Plugin** new "*file*" interfaces and behavior
- Native support; atop existing scalable services

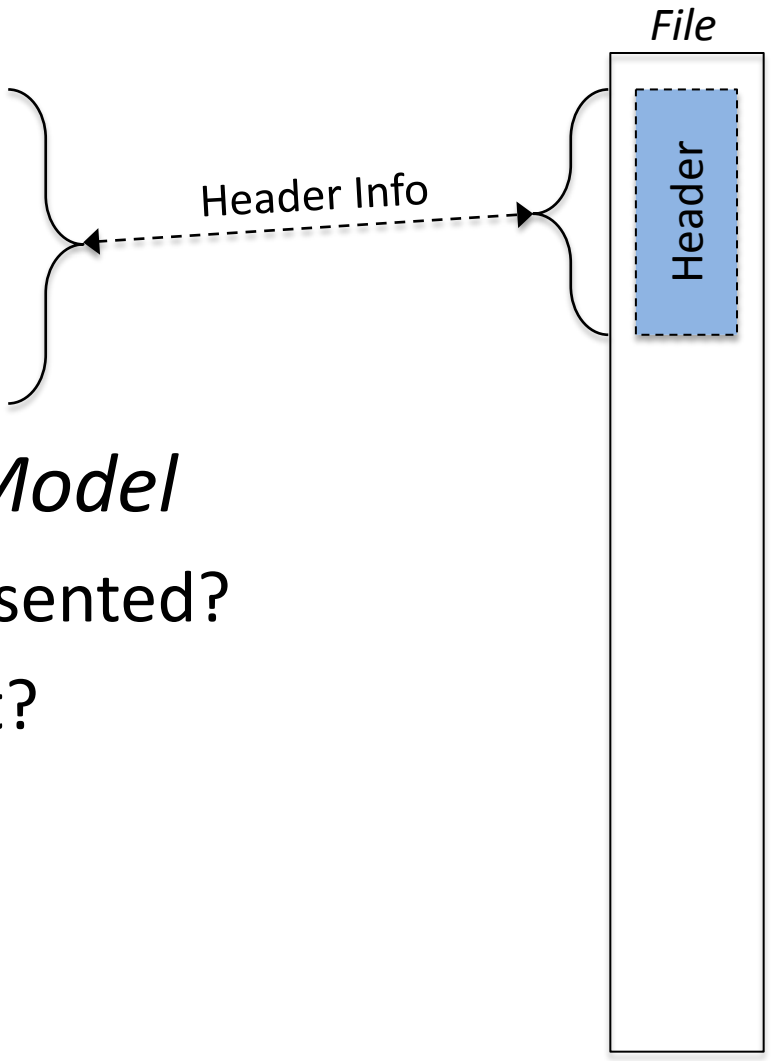


What does middleware do?



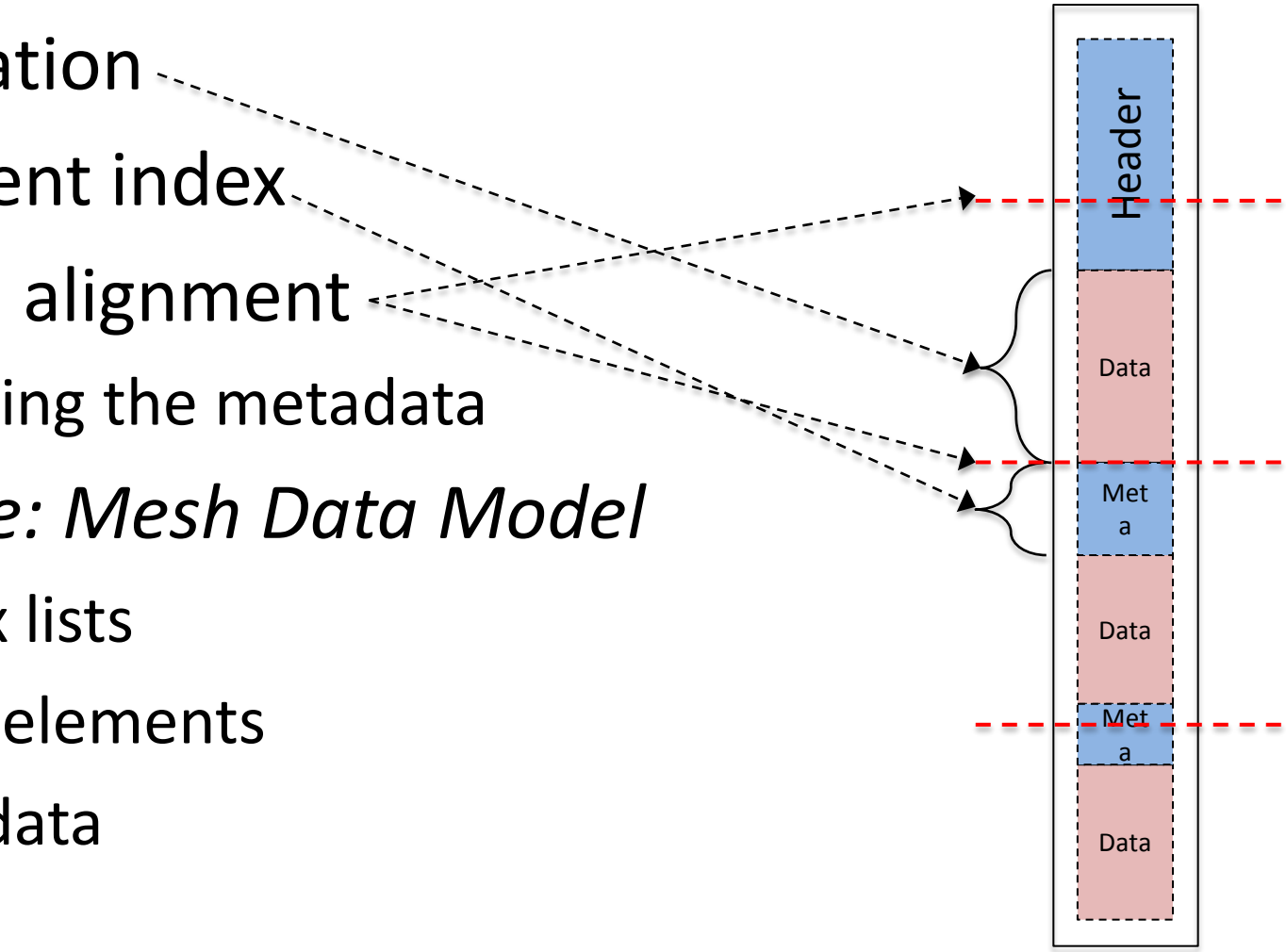
Middleware: Metadata Management

- Byte stream layout
- Data type information
- Data model attributes
- *Example: Mesh Data Model*
  - How is the mesh represented?
  - What does it represent?



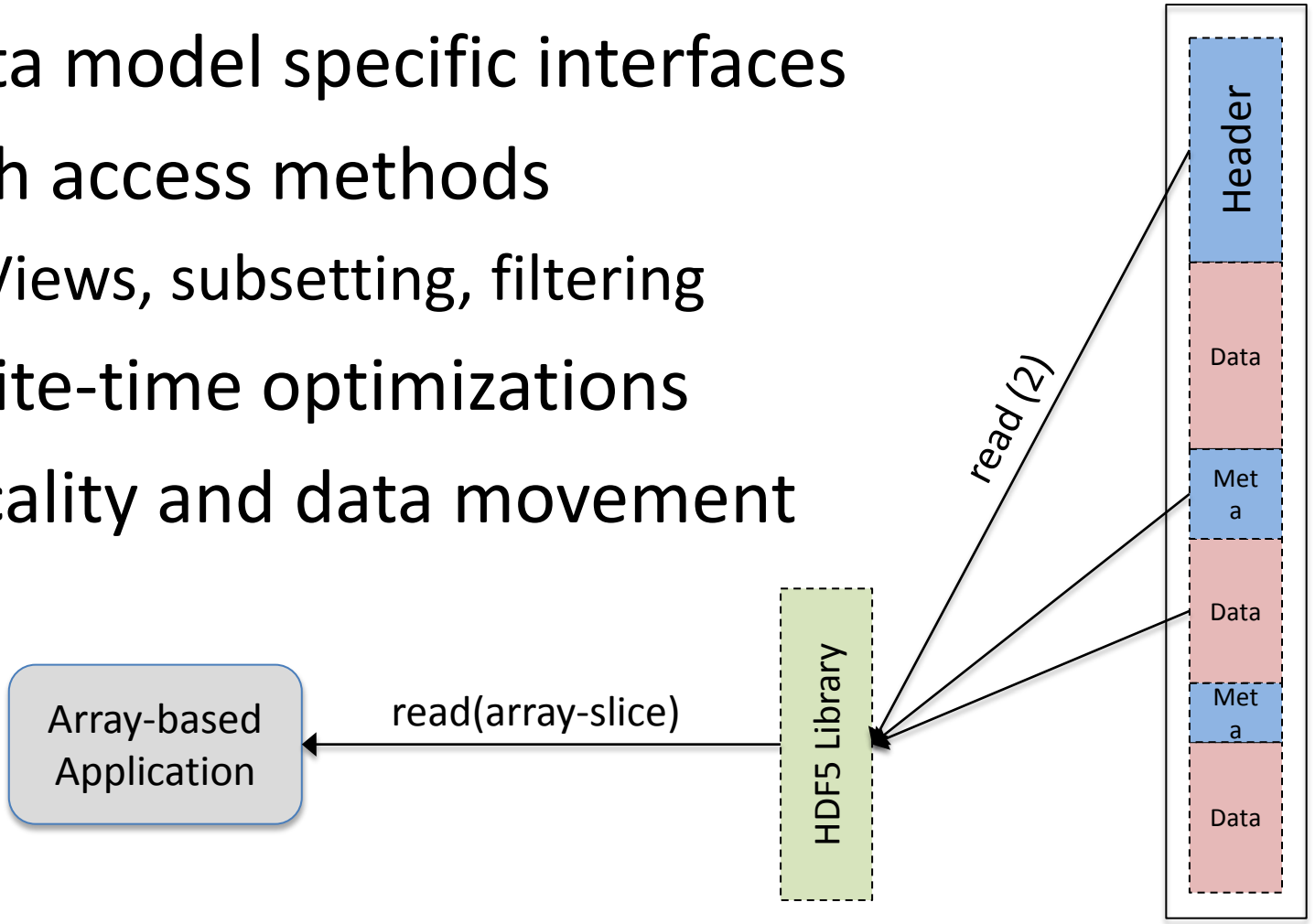
Middleware: Data Placement

- Serialization
- Placement index
- Physical alignment
  - Including the metadata
- *Example: Mesh Data Model*
  - Vertex lists
  - Mesh elements
  - Metadata



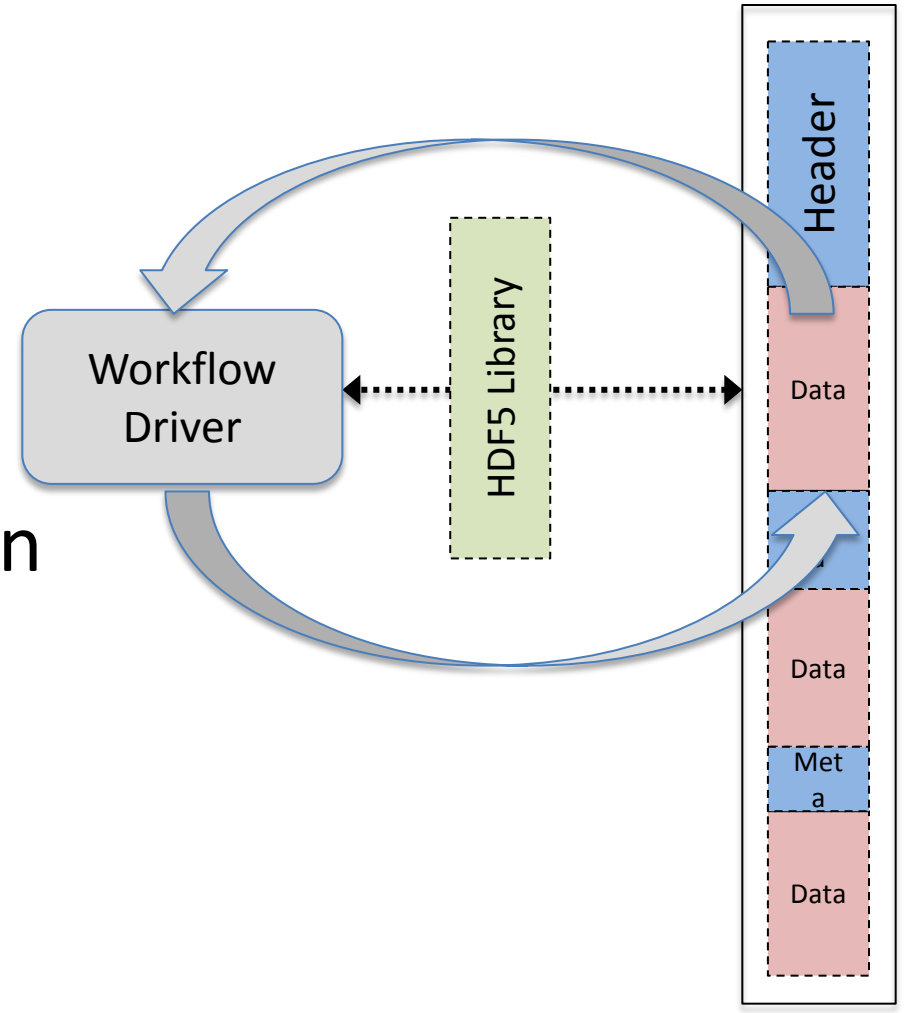
Middleware: Intelligent Access

- Data model specific interfaces
- Rich access methods
  - Views, subsetting, filtering
- Write-time optimizations
- Locality and data movement



Middleware: Asynchronous Services

- Workflows
  - Regridding
- Compression
- Indexing
- Layout optimization
- *Performed online*

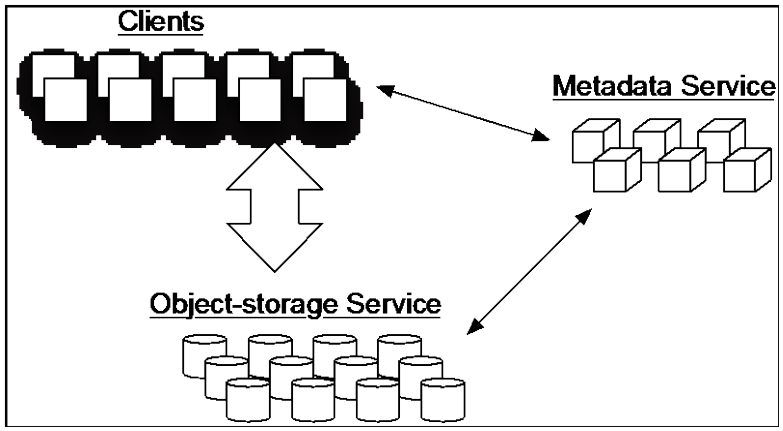


Middleware Challenges

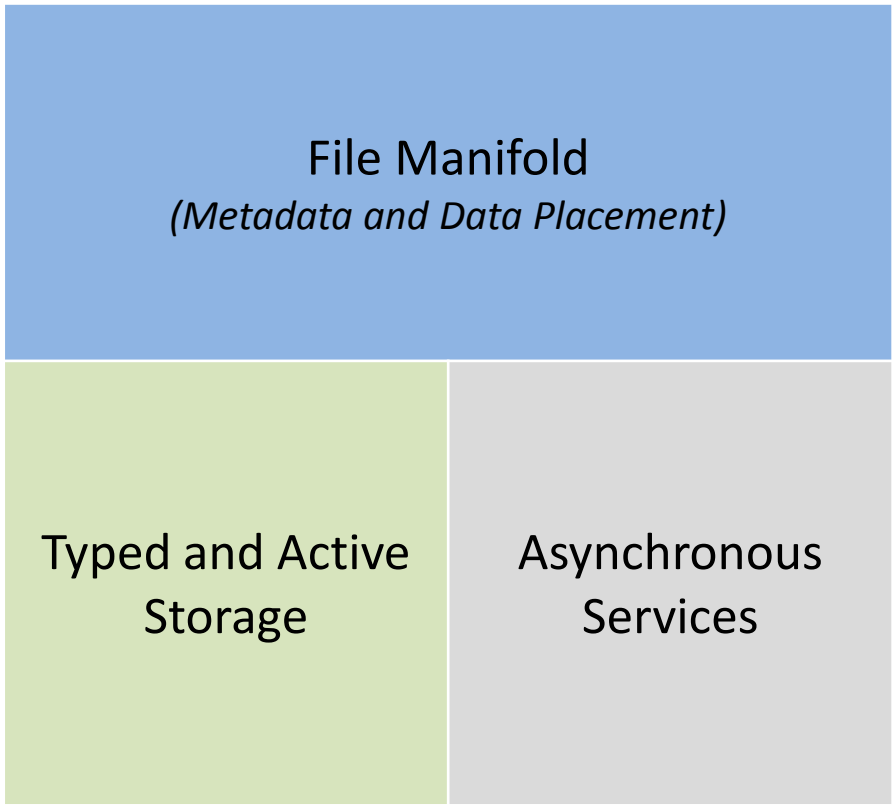
- Inflexible byte stream abstraction
- Scalability rules are simple
  - But middleware is complex
- Applying ‘magic number’
  - Unnatural and difficult to propogate
- Loss of detail at lower-levels
  - Difficult for in-transit / co-located compute

Storage System Services

- Scalable meta data
  - Clustered service
  - Scalability invariants
- Distributed object store
  - Local compute resources
  - Define new behavior
- File operations
  - POSIX
- Fault-tolerance
  - Scrubbing and replication

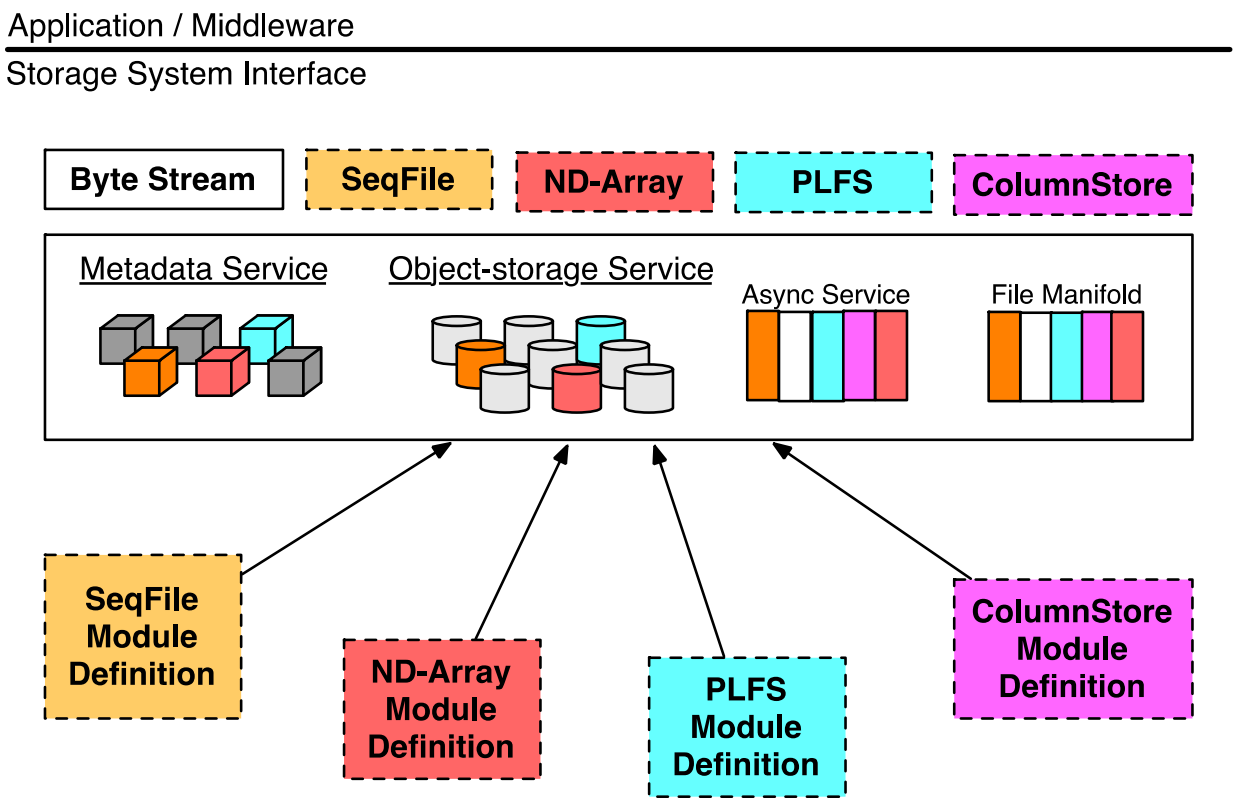


DataMods Abstraction



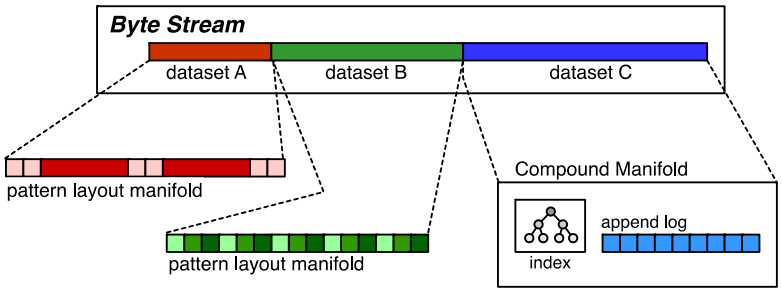
DataMods Architecture

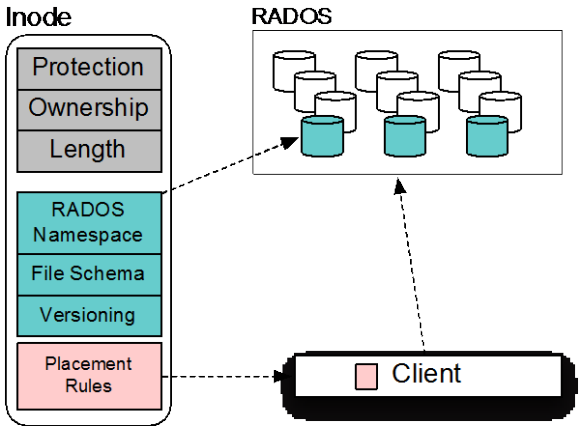
- Generalized file system services
- Exposed through programming model



File Manifold

- Metadata management and data placement
  - Flexible, custom layouts
- Extensible interfaces
- Object namespace managed by manifold
- Placement rules evaluated by system





Typed and Active Storage

- Active storage adoption has been slow
  - Code injection is scary
  - Security and QoS
- Reading, writing, and checksums are not free
- Exposing scalable services is tractable
  - Well-defined data models supports optimization
  - Programming model support data model creation
  - Indexing and filtering

Asynchronous Services

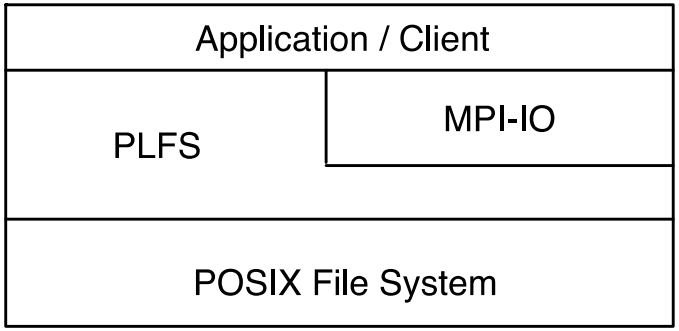
- Re-use of active / typed storage components
- Temporal relationship to file manifold
  - Incremental processing
  - After file is closed
  - Object update trigger
- Scheduling
  - Exploit idle time
  - Integrate with larger ecosystem
  - Preempted or aborted

Case Study: PLFS Checkpoint/Restart

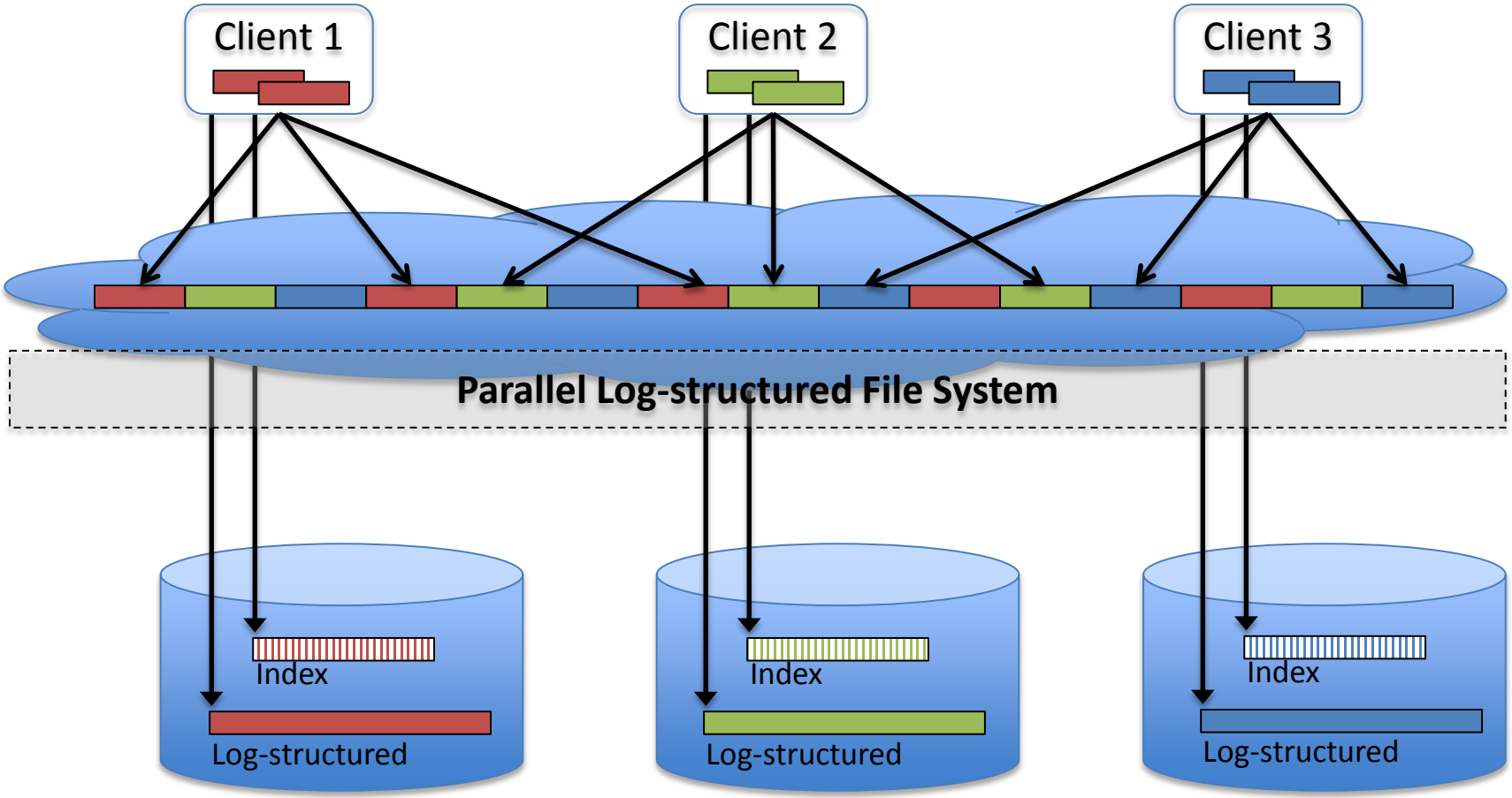
- Long-running simulations need fault-tolerance
  - Checkpoint simulation state
- Simulations run on expensive machines
  - Very expensive machines. Really, very expensive.
- Decrease cost (time) of checkpoint/restart
- **Translation: increase bulk I/O bandwidth**

Overview of PLFS

- Middleware layer
  - Transforms I/O pattern
- IO Pattern: **N-1**
  - Most common
- IO Pattern: **N-N**
  - File system friendly
- Convert N-1 into N-N
- Applications see the same logical file

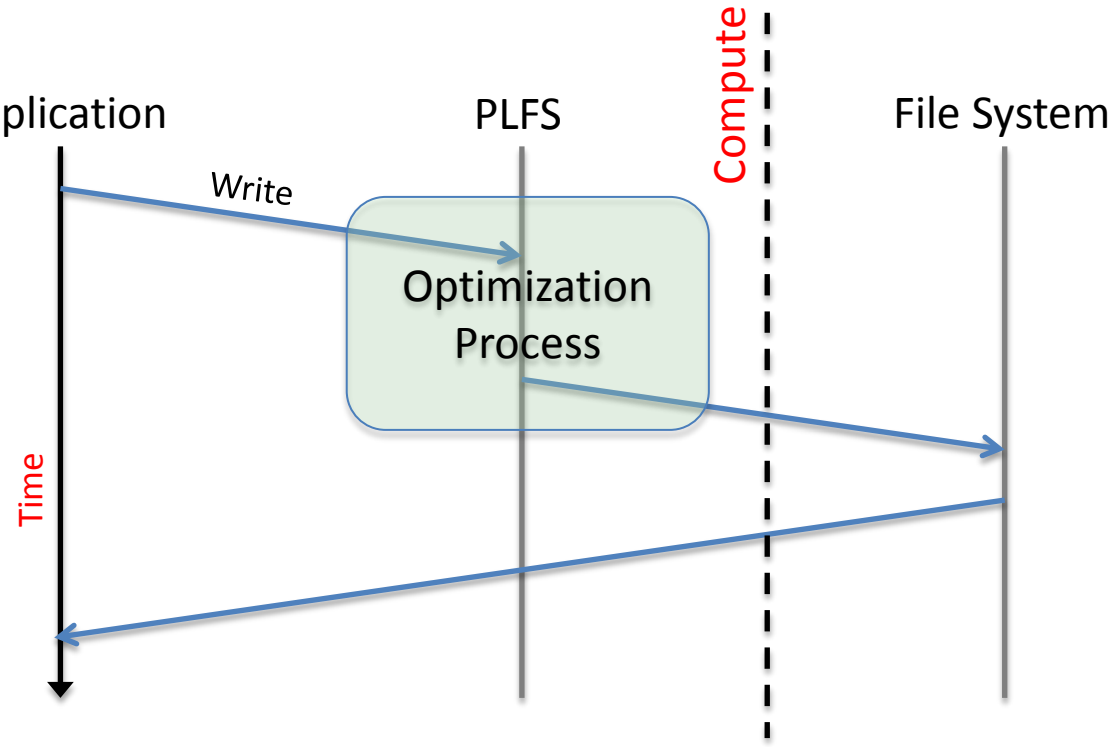


Simplified PLFS I/O Behavior



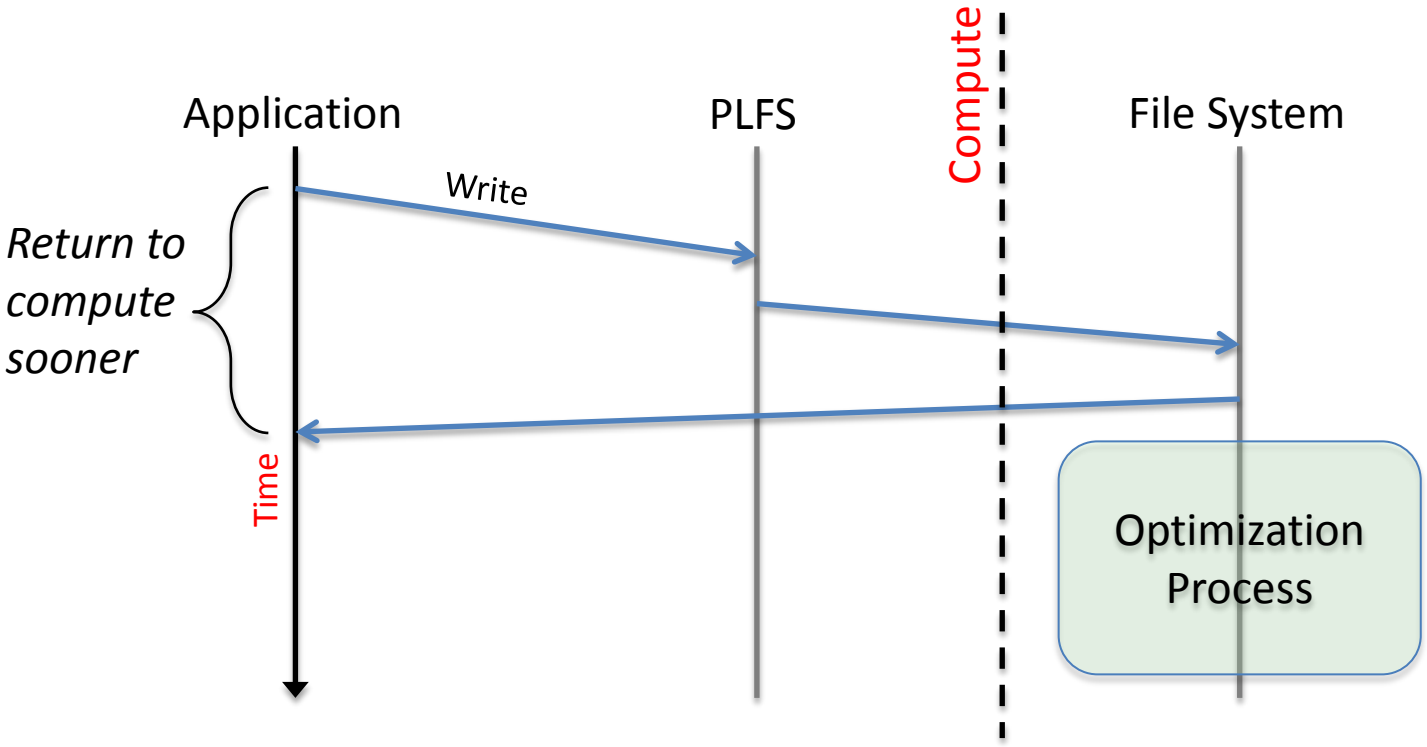
PLFS Scalability Challenges

- **Index maintenance and volume**
- Optimization above file system
  - Compression and reorganization



Moving Overhead to Storage System

- Checkpoints are not read immediately (if at all)
  - Index maintenance and optimization in storage

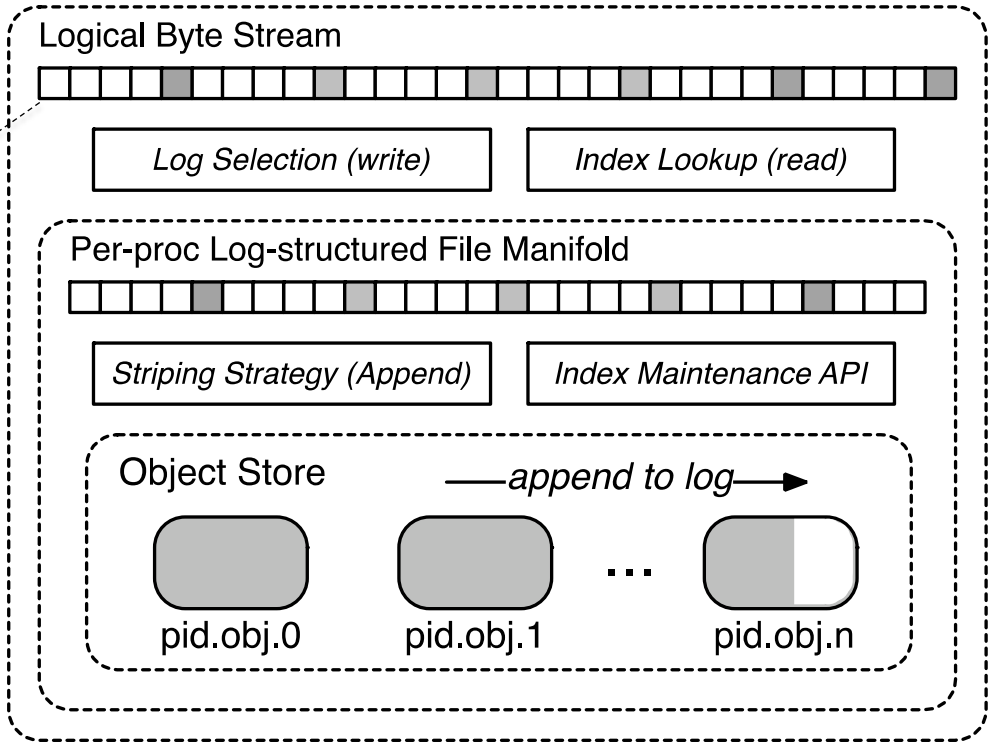


DataMods Module for PLFS

- File Manifold
  - Logical file view
  - Per-process log-structured files
  - Index
- Hierarchical Solution
  - Top-level manifold routes to logs
  - Inner manifold implements log-structured file
  - Automatic namespace management (metadata)

PLFS ***Outer*** File Manifold

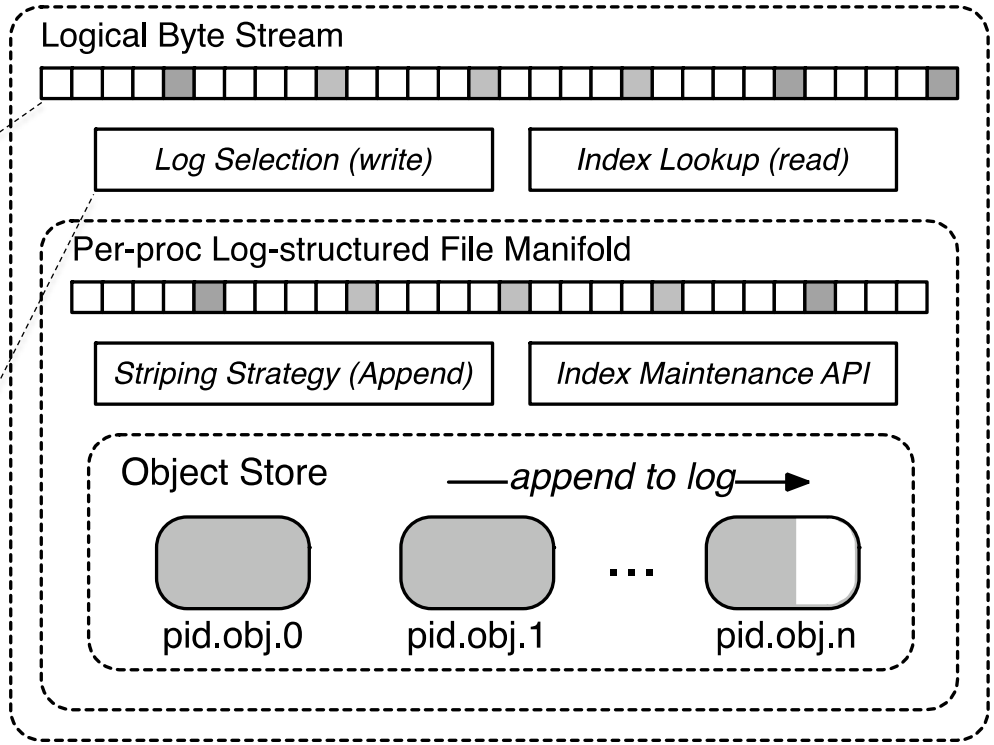
Logical top-half file is not materialized



PLFS ***Outer*** File Manifold

Logical top-half file is not materialized

Routes to per-process log file

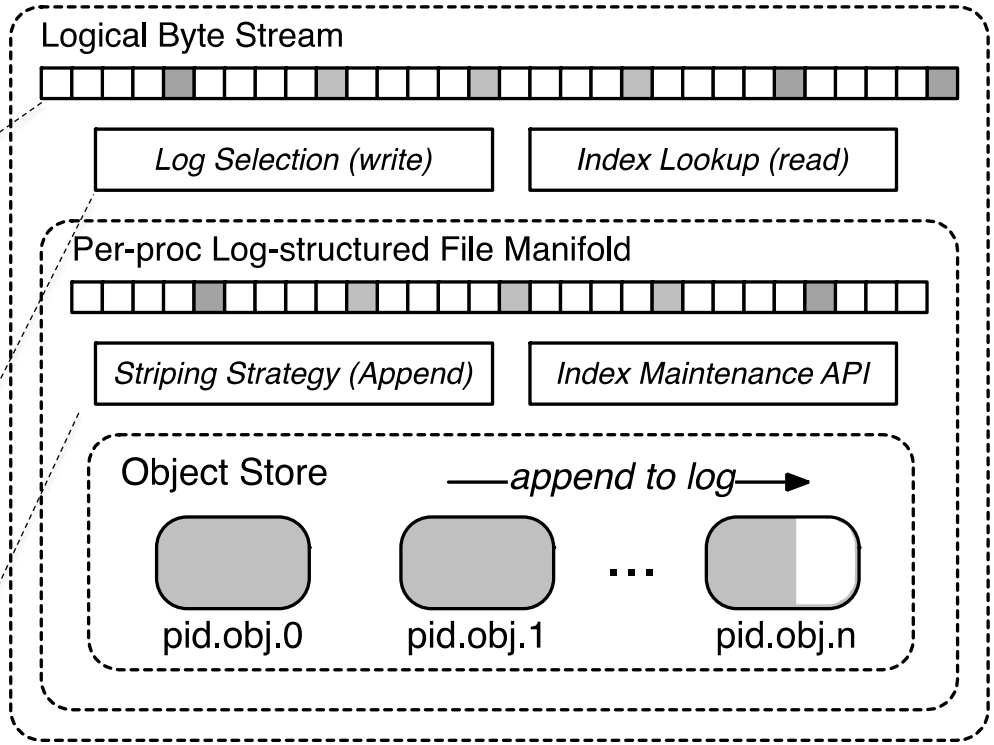


PLFS ***Inner*** File Manifold

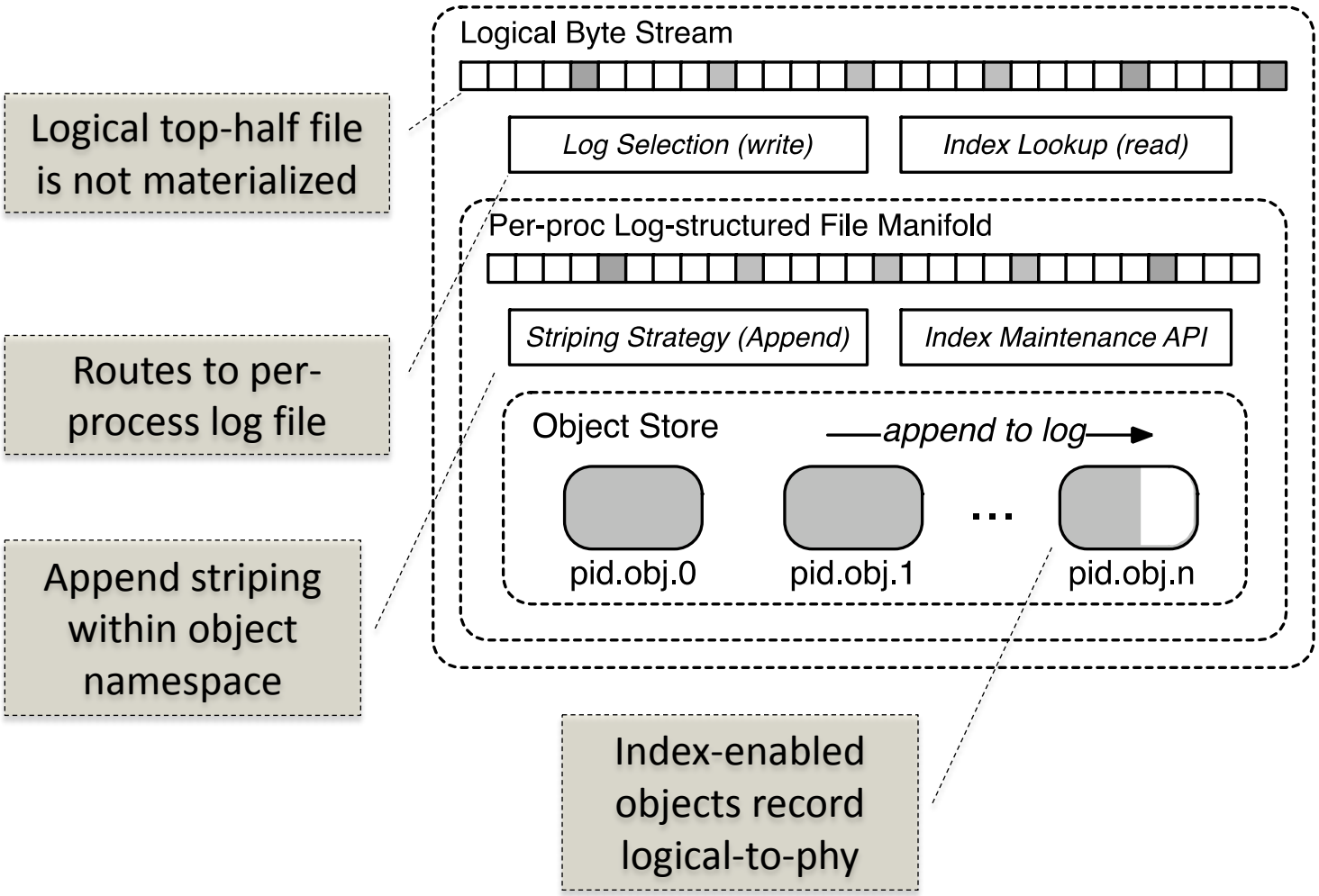
Logical top-half file is not materialized

Routes to per-process log file

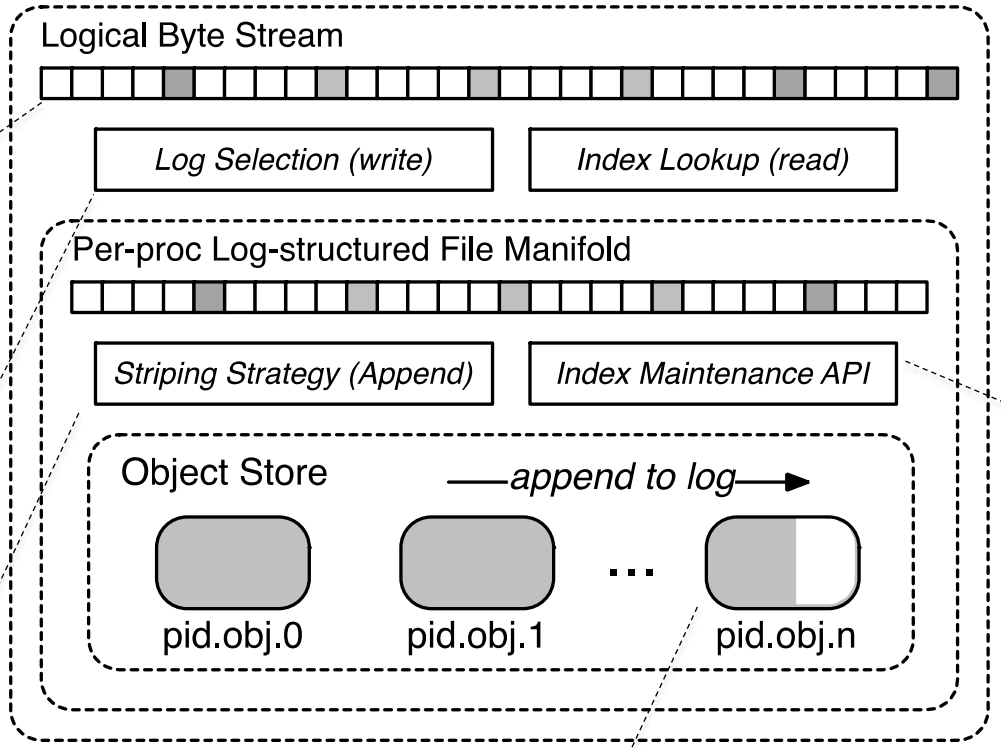
Append striping within object namespace



PLFS ***Inner*** File Manifold



PLFS ***Inner*** File Manifold



Logical top-half file is not materialized

Routes to per-process log file

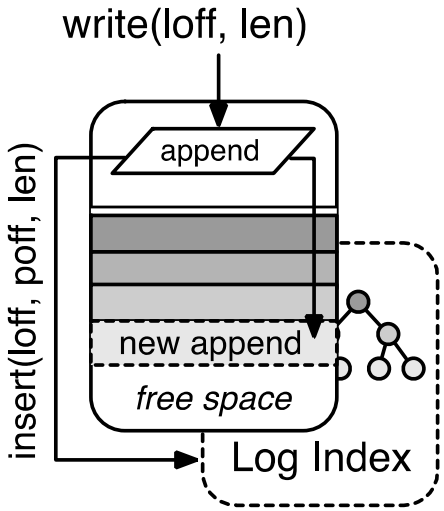
Append striping within object namespace

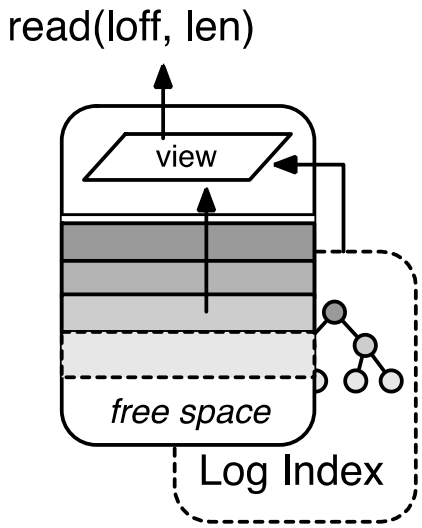
Interface to index maintenance routines

Index-enabled objects record logical-to-phy

Active and Typed Objects

- Append-only object
- Automatic indexing
- Managed layout
- Built on existing services
- Logical view at lowest level
- Index maintenance interface



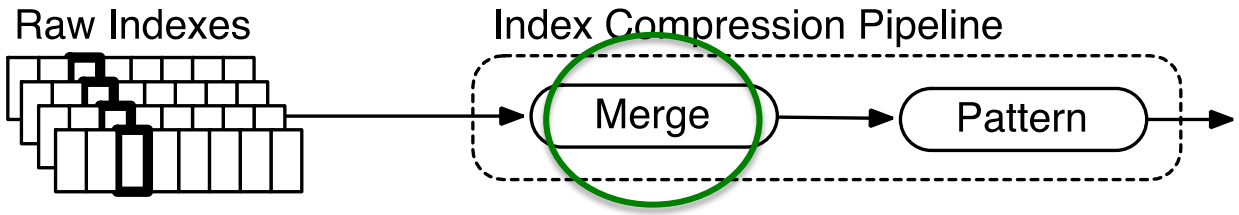


Offline Index Optimization

- Extreme index fragmentation (per-object)
- Exploit opportunities for optimization
  - Storage system idle time
  - Re-use of analysis I/O
  - Piggy-backed on scrubbing / healing
- Index Compression
  - Merging contiguous entries
  - Pattern discovery and replacement
  - Consolidation

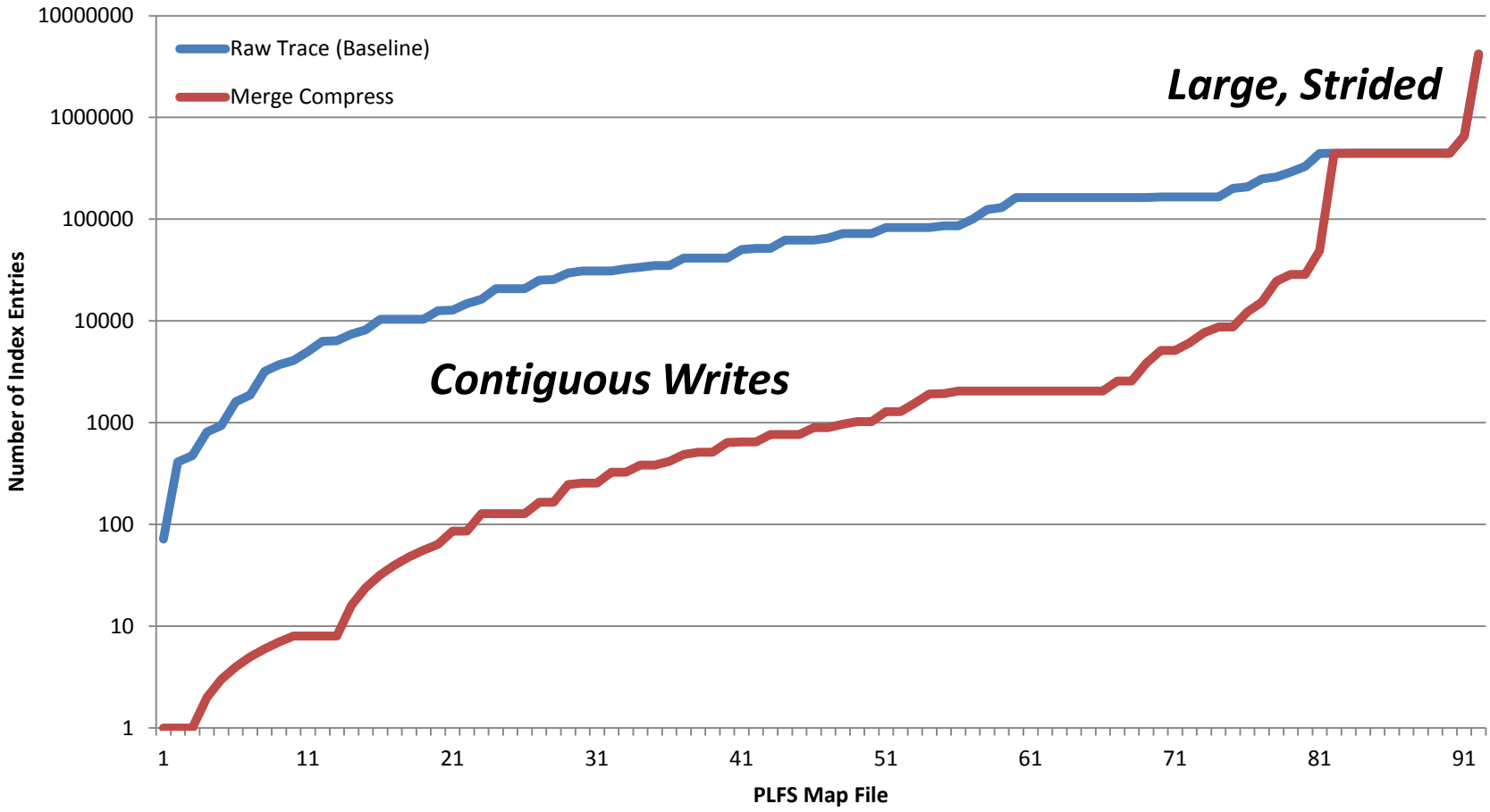
Offline Index Optimization

- Three stage pipeline
  - Incremental compression and consolidation
- Incremental compression
  1. **Merging** physically contiguous entries (in PLFS)
     - Not subject to buffer size limits



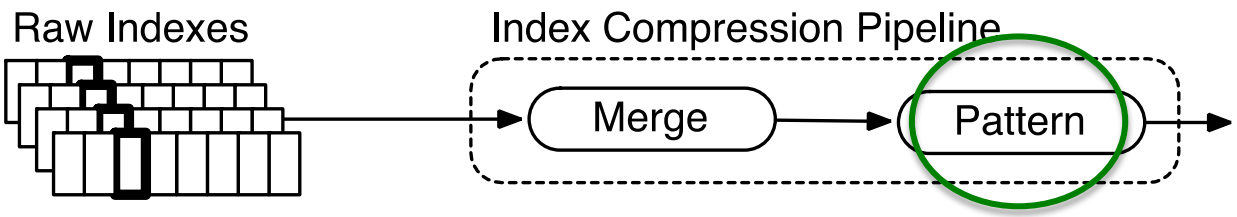
- Applied technique to 92 PLFS indexes published by LANL

Merging Reduces PLFS Index Size

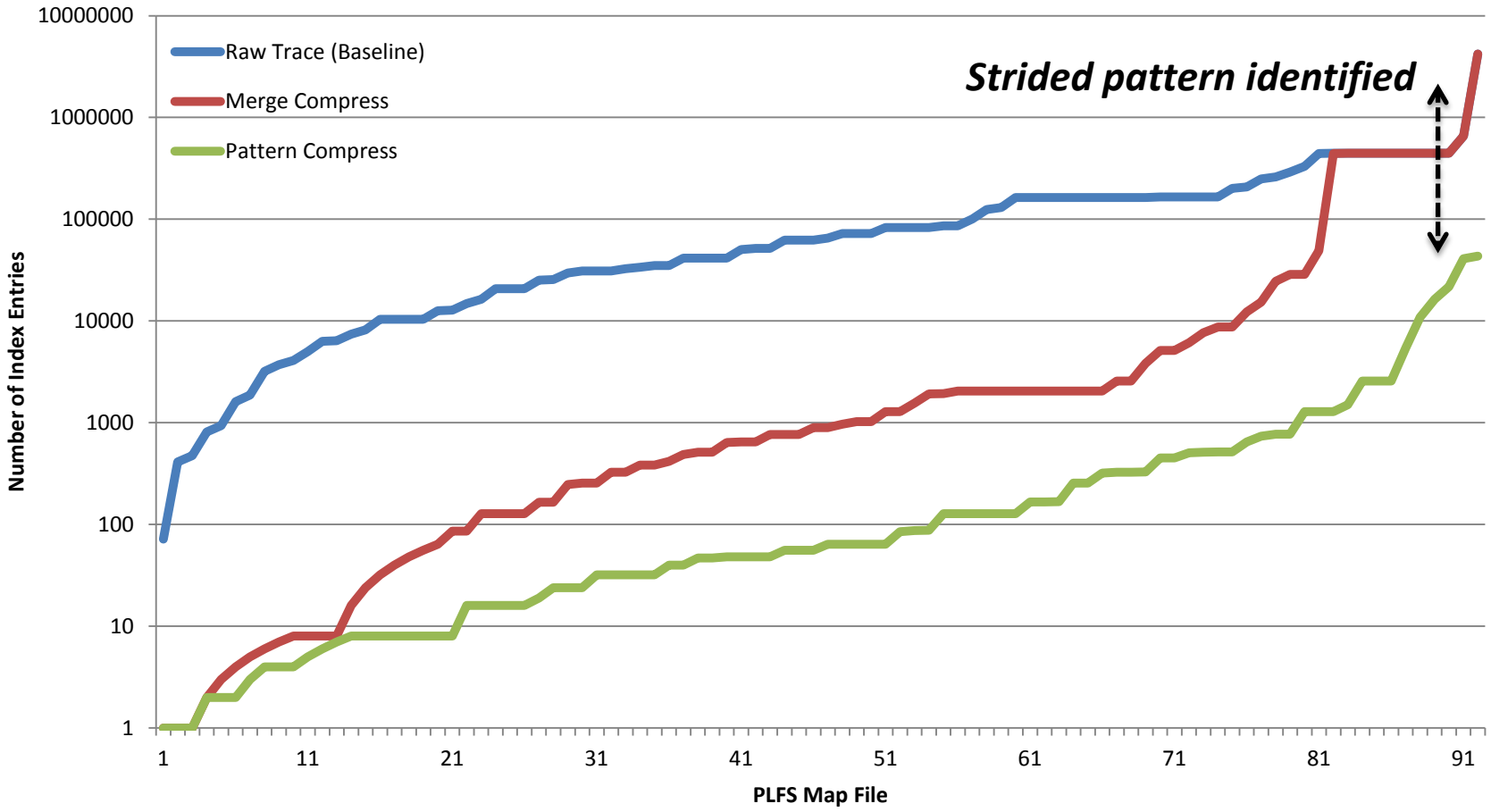


Index Compression: Pattern

- Compactly represent extents using patterns
- Example pattern template
  - offset + stride * i, low < i < high
- Fit data to this pattern to reduce index size
- Linear algorithm; parallel across logs

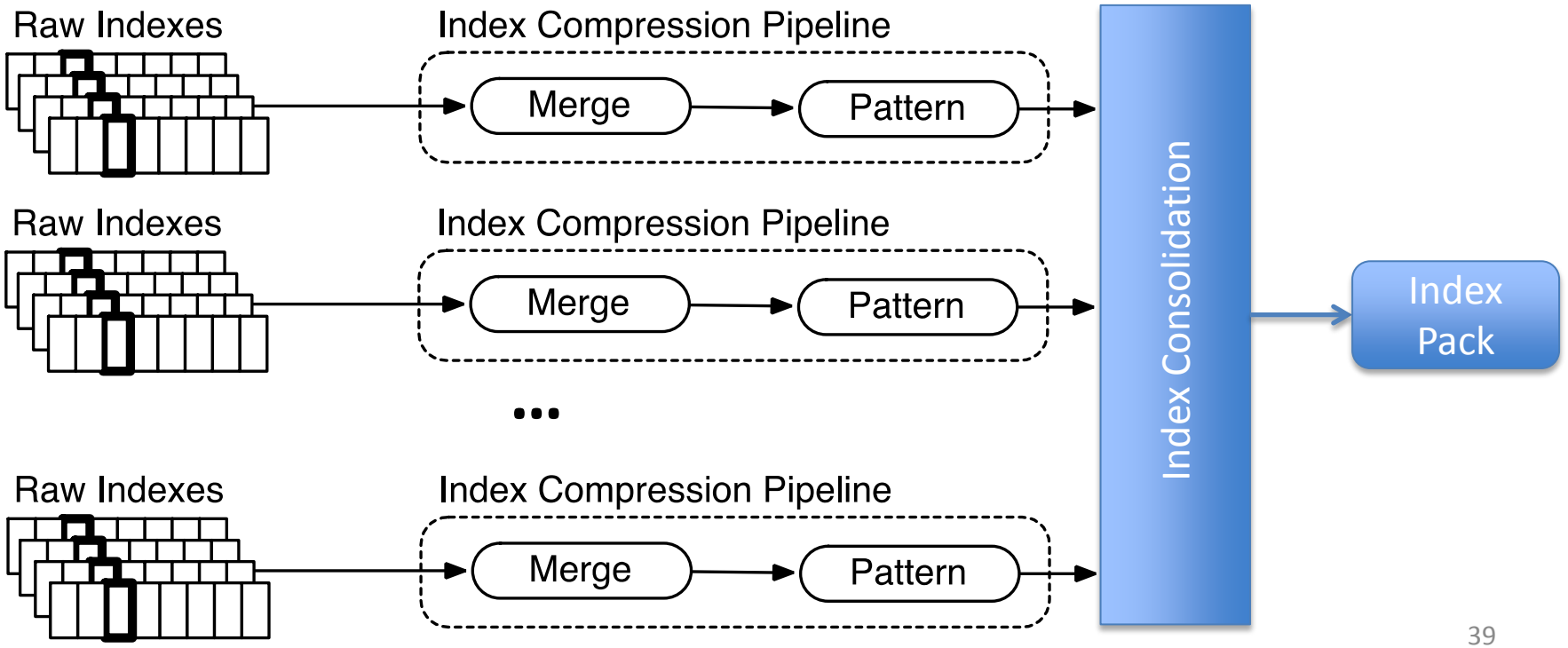


Pattern Compression Improves Over Merging



Index Consolidation

- Combines all logs together (in PLFS)
- Increases index read efficiency



Wrapping Up

- Implementing new data model plugins
  - Hadoop and Visualization
  - Refining API with more use cases
  - Constructing specification language
- Thank you to supporters
  - DOE funding (DE-SC0005428), Gary Grider John Bent, James Nunez
- **Questions? ---** jayhawk@cs.ucsc.edu
- Poster session

Extra Slides

Index Reduction Improvements