Fusing
Data Management Services with File Systems

(A Vision by Database & File System Researchers)

Scott Brandt, Carlos Maltzahn, Neoklis Polyzotis, Wang-Chiew Tan

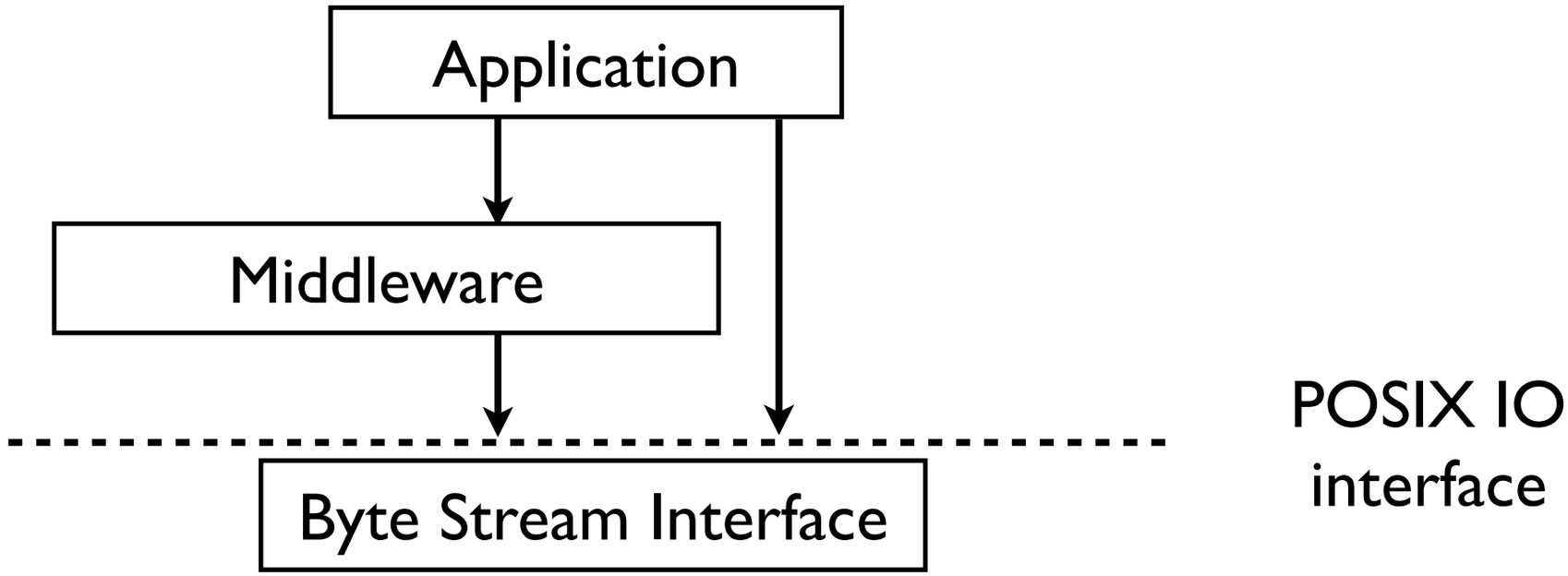
ISSDM, PDSI, Systems Research Lab, Database Systems Group, UC Santa Cruz

Introduction

- Data is growing fast
  - 10-100x faster than compute speeds
  - moving data becomes dominant
- POSIX IO does not scale
  - 45 years ago: 100MB of data at high end
  - Now: up to 1 billion times more data
  - Middleware tries to make up for limitations
- Performance price of POSIX IO is high
  - Workload-specific interposition layers:
    almost 1,000 x speed-up
- Middleware and Interposition layers
  - Either not fully exploit storage architecture, or
  - Statically encode workload-architecture mapping
  - Parallel to early database history

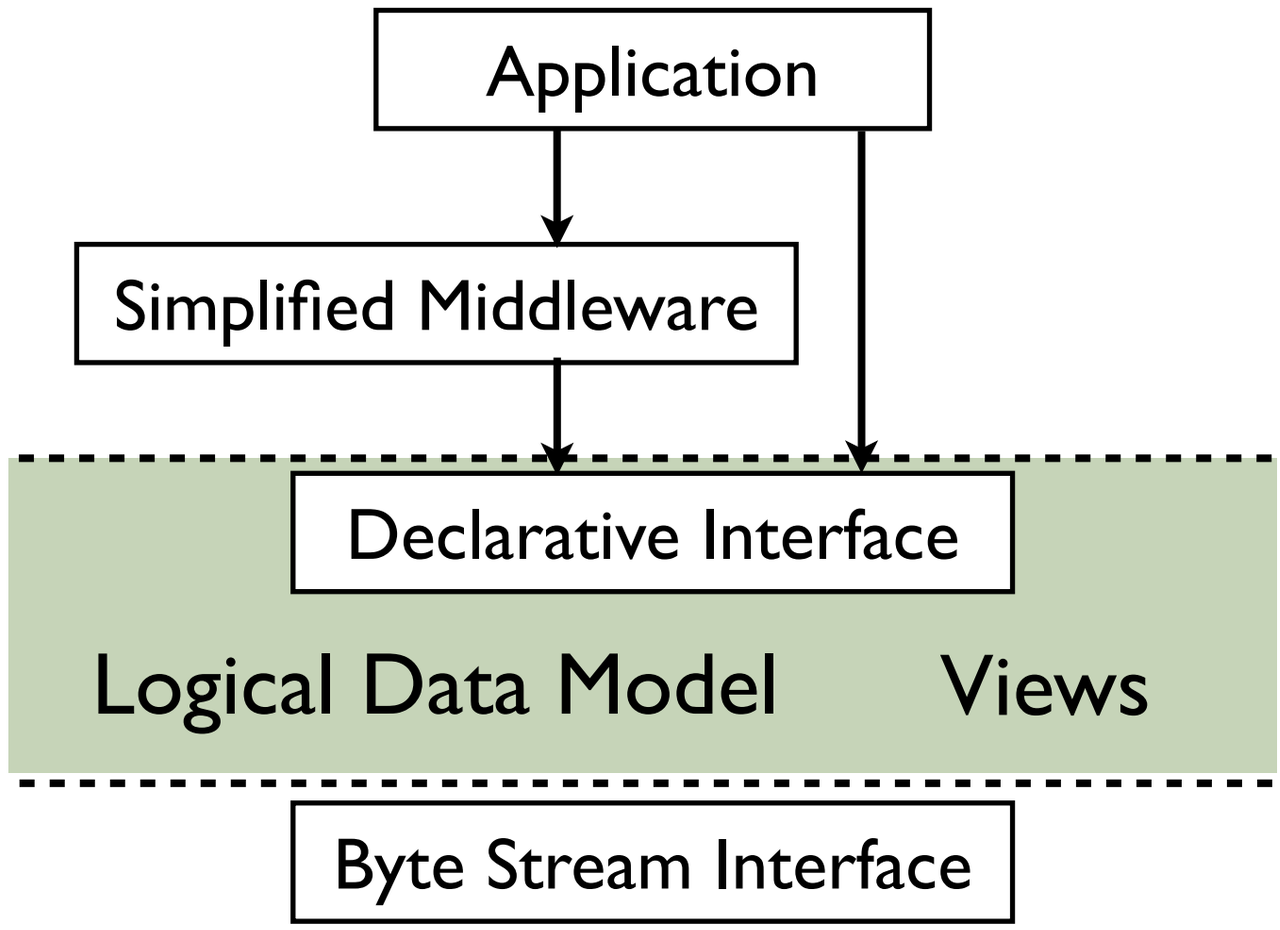
2

Damasc


Application
Middleware
Byte Stream Interface
POSIX IO
interface

3

Damasc



Damasc interface

POSIX IO interface

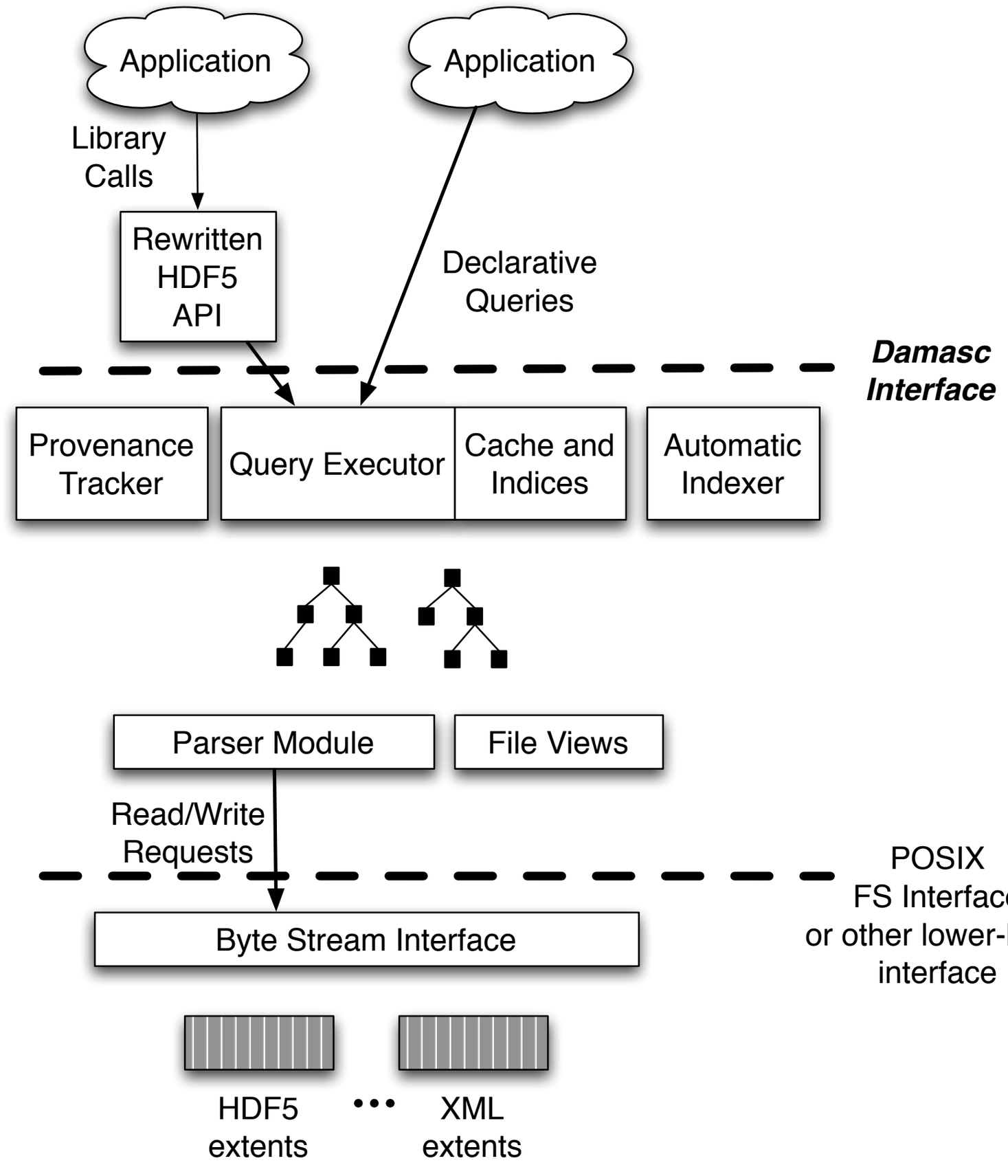
4

Why now?

1. Mature HPC Middleware APIs:
   - NetCDF, HDF5, MPI IO, ...
   - Established abstractions above POSIX IO
2. Big data increasingly forces out-of-core data management
3. Advances in auto-tuning of database systems
4. Advances in end-to-end performance management
   - Scheduling of competing data management activities
5. Advances in parallel file system scalability
   - via greater intelligence at storage nodes

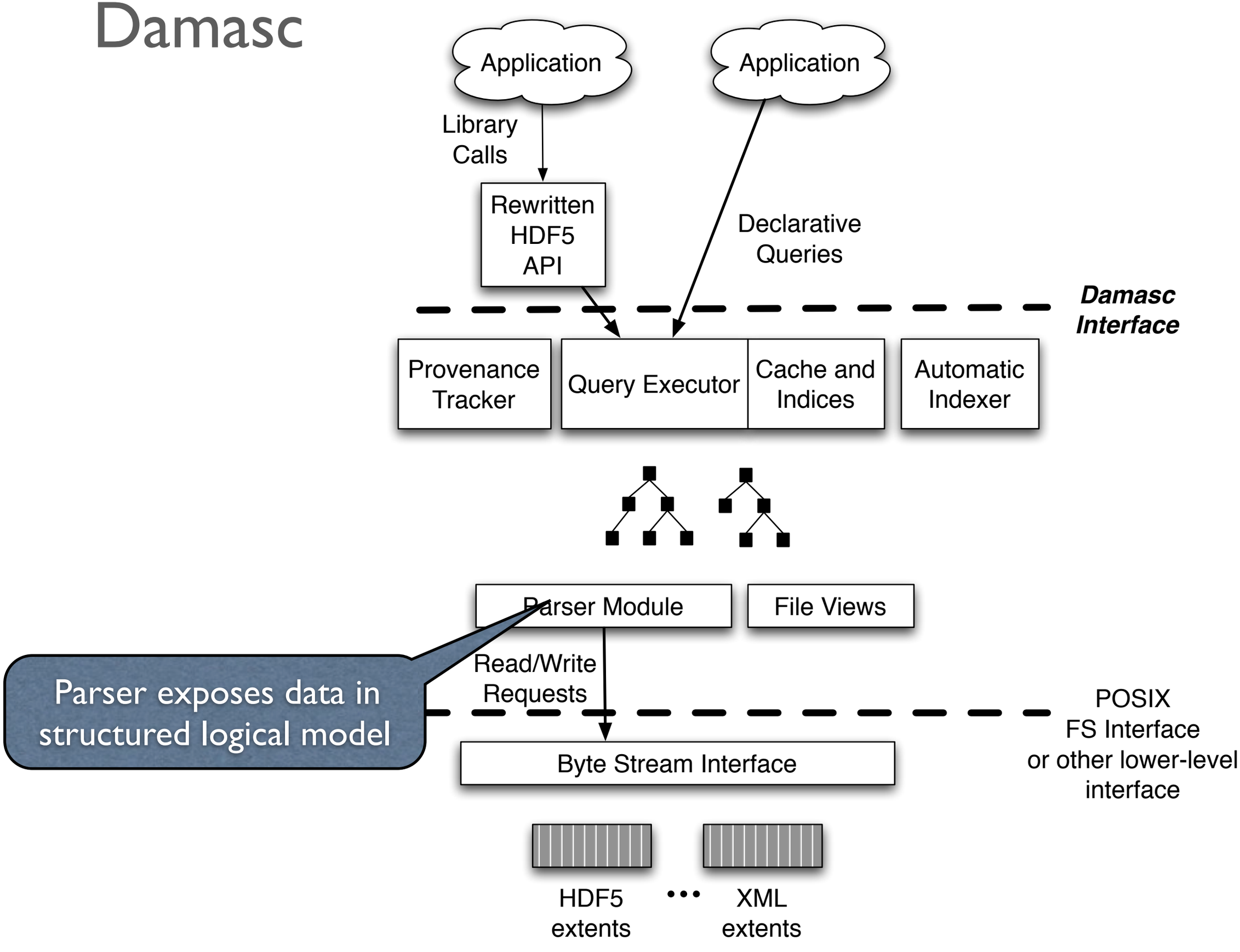
5

Damasc


Application
Application
Library
Calls
Rewritten
HDF5
API
Declarative
Queries
Damasc
Interface
Provenance
Tracker
Query Executor
Cache and
Indices
Automatic
Indexer
Parser Module
File Views
Read/Write
Requests
POSIX
FS Interface
or other lower-level
interface
Byte Stream Interface
HDF5
extents
XML
extents

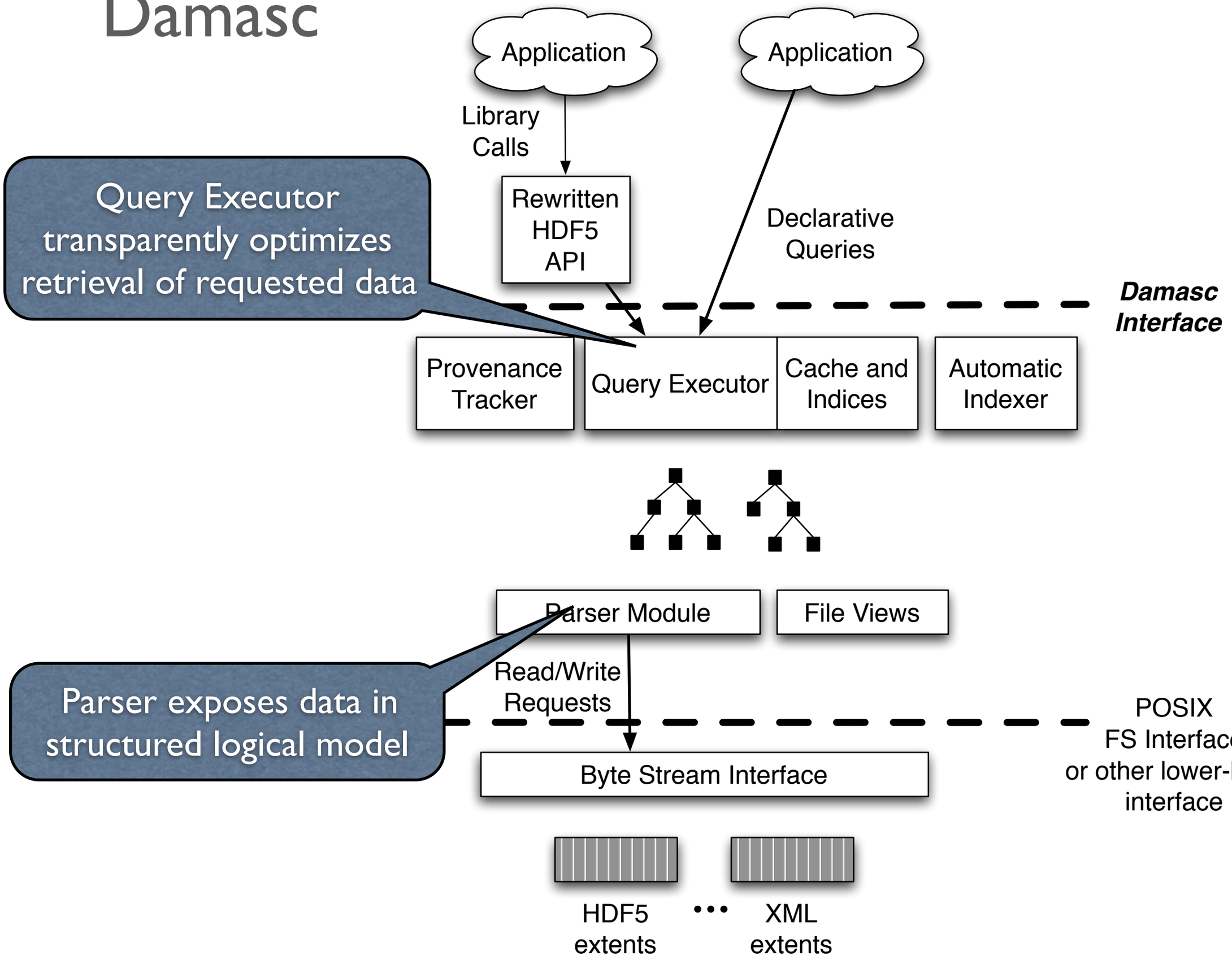
6

Damasc


Application
Application
Library Calls
Rewritten HDF5 API
Declarative Queries
Damasc Interface
Provenance Tracker
Query Executor
Cache and Indices
Automatic Indexer
Parser exposes data in structured logical model
Parser Module
File Views
Read/Write Requests
Byte Stream Interface
POSIX FS Interface or other lower-level interface
HDF5 extents
XML extents

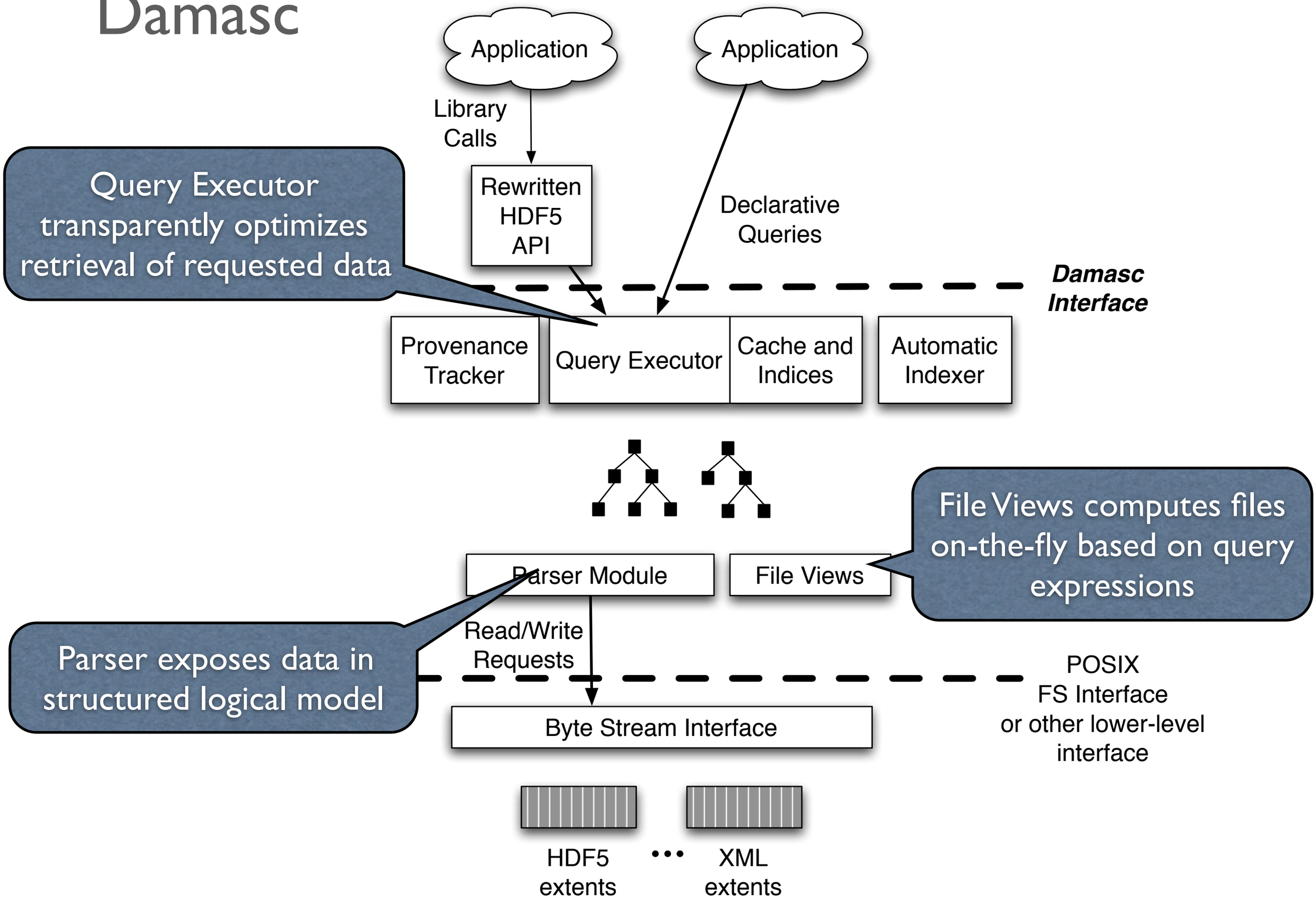
6

Damasc


Application
Application
Library Calls
Rewritten HDF5 API
Declarative Queries
Query Executor transparently optimizes retrieval of requested data
Damasc Interface
Provenance Tracker
Query Executor
Cache and Indices
Automatic Indexer
Parser Module
File Views
Read/Write Requests
Parser exposes data in structured logical model
POSIX FS Interface or other lower-level interface
Byte Stream Interface
HDF5 extents
XML extents

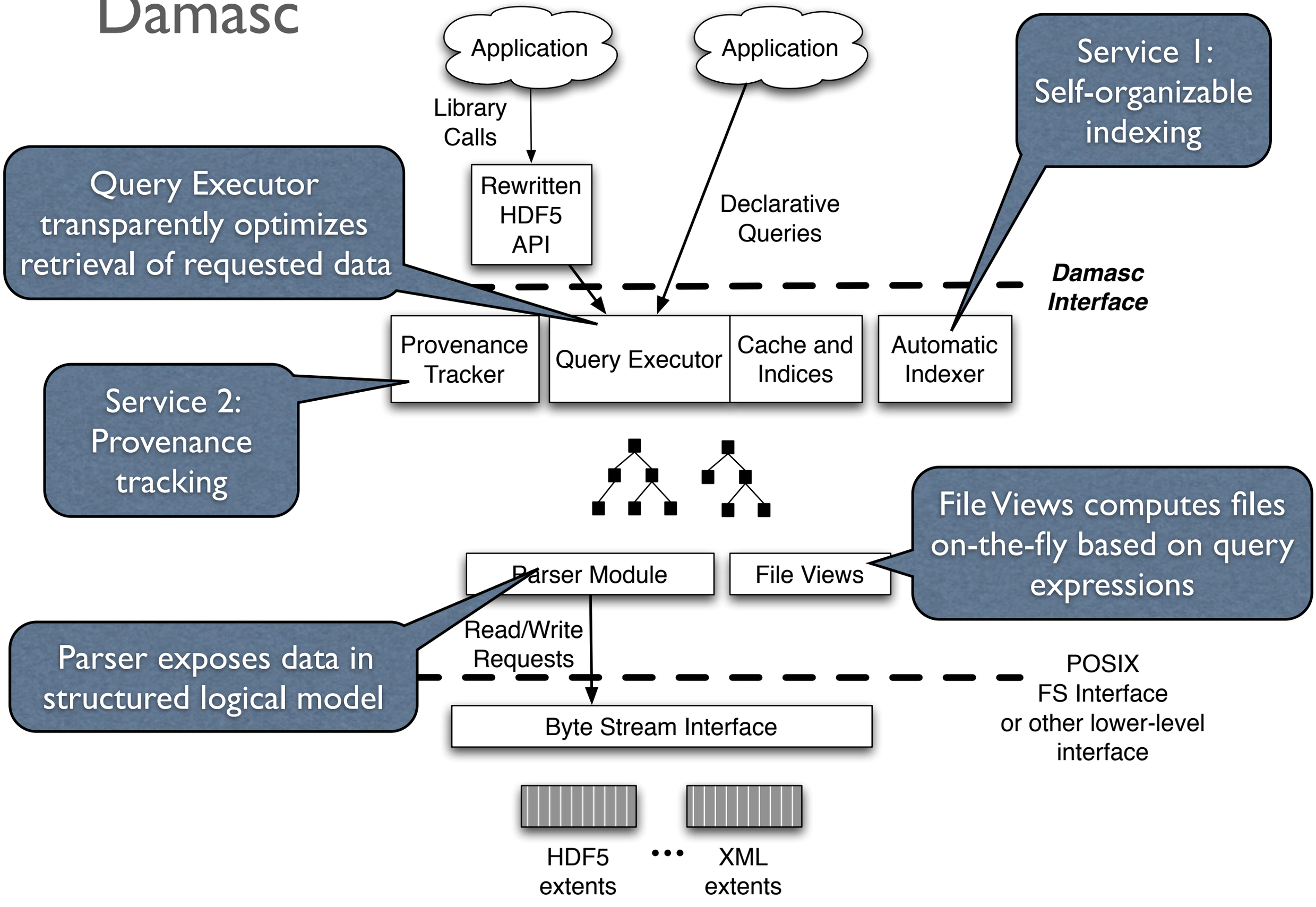
6

Damasc


Application
Application
Library Calls
Rewritten HDF5 API
Declarative Queries
Query Executor transparently optimizes retrieval of requested data
Damasc Interface
Provenance Tracker
Query Executor
Cache and Indices
Automatic Indexer
Parser Module
File Views
File Views computes files on-the-fly based on query expressions
Read/Write Requests
Parser exposes data in structured logical model
POSIX FS Interface or other lower-level interface
Byte Stream Interface
HDF5 extents
XML extents

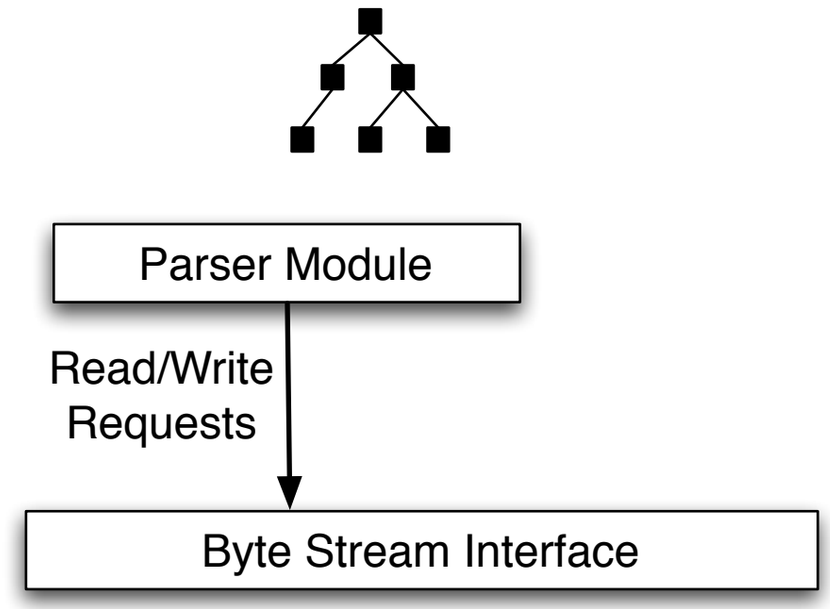
6

Damasc


Application
Application
Library Calls
Rewritten HDF5 API
Declarative Queries
Service 1: Self-organizable indexing
Query Executor transparently optimizes retrieval of requested data
Damasc Interface
Provenance Tracker
Query Executor
Cache and Indices
Automatic Indexer
Service 2: Provenance tracking
Parser Module
File Views
File Views computes files on-the-fly based on query expressions
Read/Write Requests
Parser exposes data in structured logical model
POSIX FS Interface or other lower-level interface
Byte Stream Interface
HDF5 extents
XML extents

6

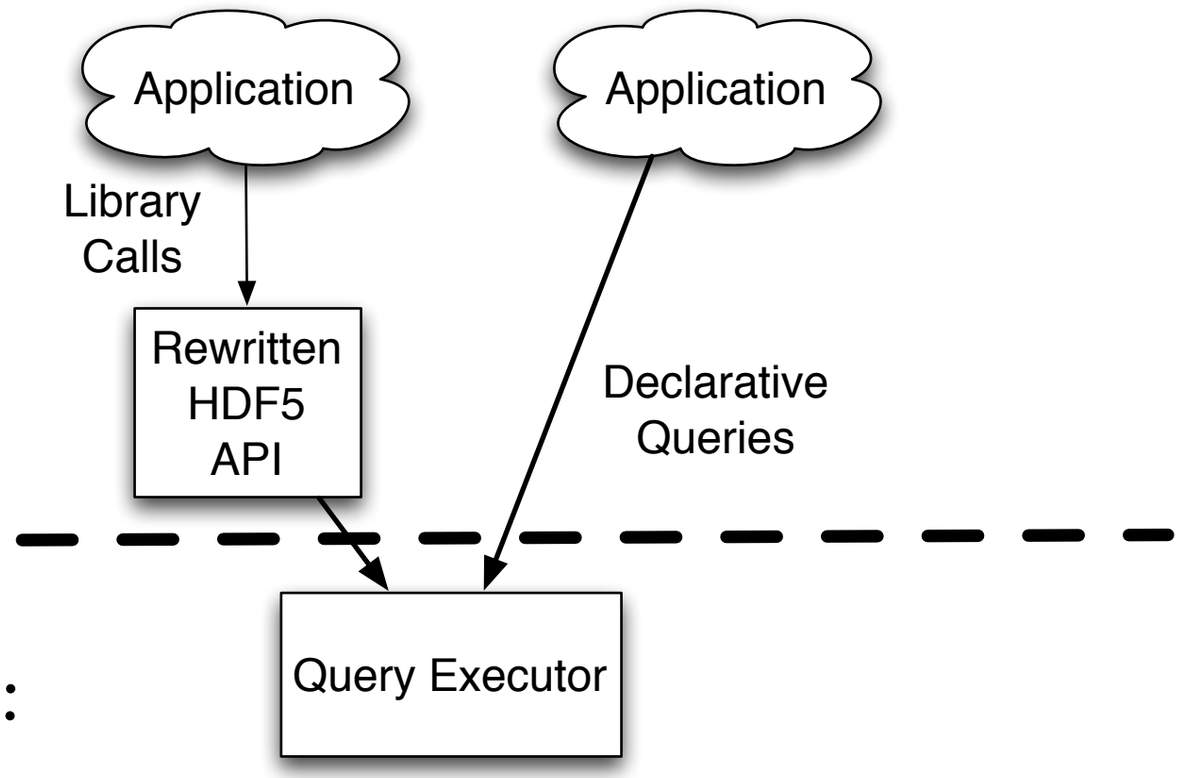
Parser Module



- Repository of format-specific parsers
- Parser creates trees
- Common semi-structured data model
  - Same abstract interface to different formats
  - Structural indices into byte stream files
- Formulation vs Evaluation of queries:
  - Formulation: based on semi-structured data model
  - Evaluation: lazy, parsing of entire file **not** required
- Optimized/scalable evaluation mechanisms ...

7

Declarative Queries



- Declarative "query" expression:
  - **What** to retrieve or update, not **how**
  - As opposed to procedural querying
- Algebraic operators:
  - Input and output is in semi-structured data model
  - Leverage existing query languages (XPath, ...)
  - Include operators of common formats (HDF5, NetCDF, ...)
- Complex expressions:
  - Retrieve/update combinations

8

Declarative Access: Example

- Path access operator applied on a bib file with

  publication[/title contains ‘file’ ‘systems’]/abstract

  - Return abstracts of publications that contain the word “file” or “system” in the title

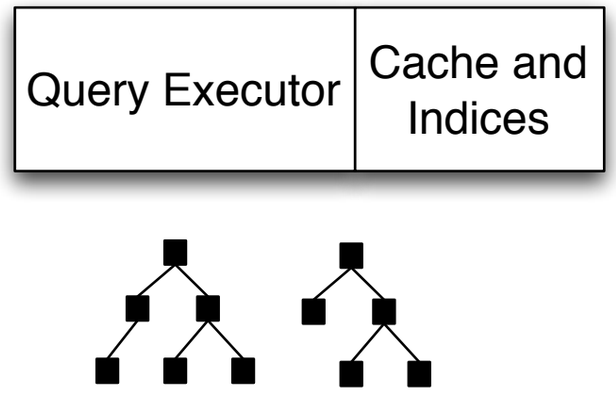
- Update operator applied on bib file with

  publication[/title contains ‘file’]/tags = ‘file’

  - Tag publications with “file” that contain the word “file” in the title

9

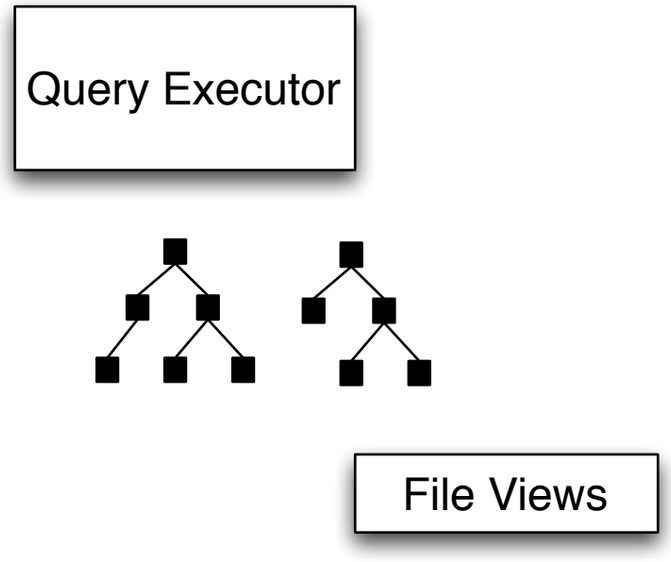
Query Optimization



- Query Executor:
  - Input: expression of logical operators
  - Output: best query plan using physical operators
  - Via: rewrite rules, cost model
- Cost Model: accounts for layout and statistics about data
- Fragmented Parsing:
  - Physical operators use file metadata to parse only what is needed
- Indexing of Keywords and File Structure
- Hybrid expressions: logical and physical operations
  - Trying to avoid need to bypass Damasc

10

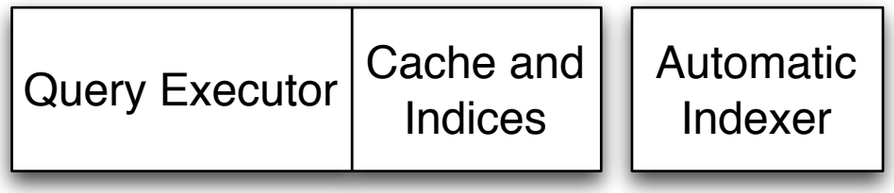
Views



- On-the-fly computation of file(s)
- First class citizen:
  - Views can be queried and/or updated
  - Views can be based on other views
- Query "fusing":
  - Optimize query against view + query defining view
- Logical Independence:
  - Views shield applications from physical format changes

11

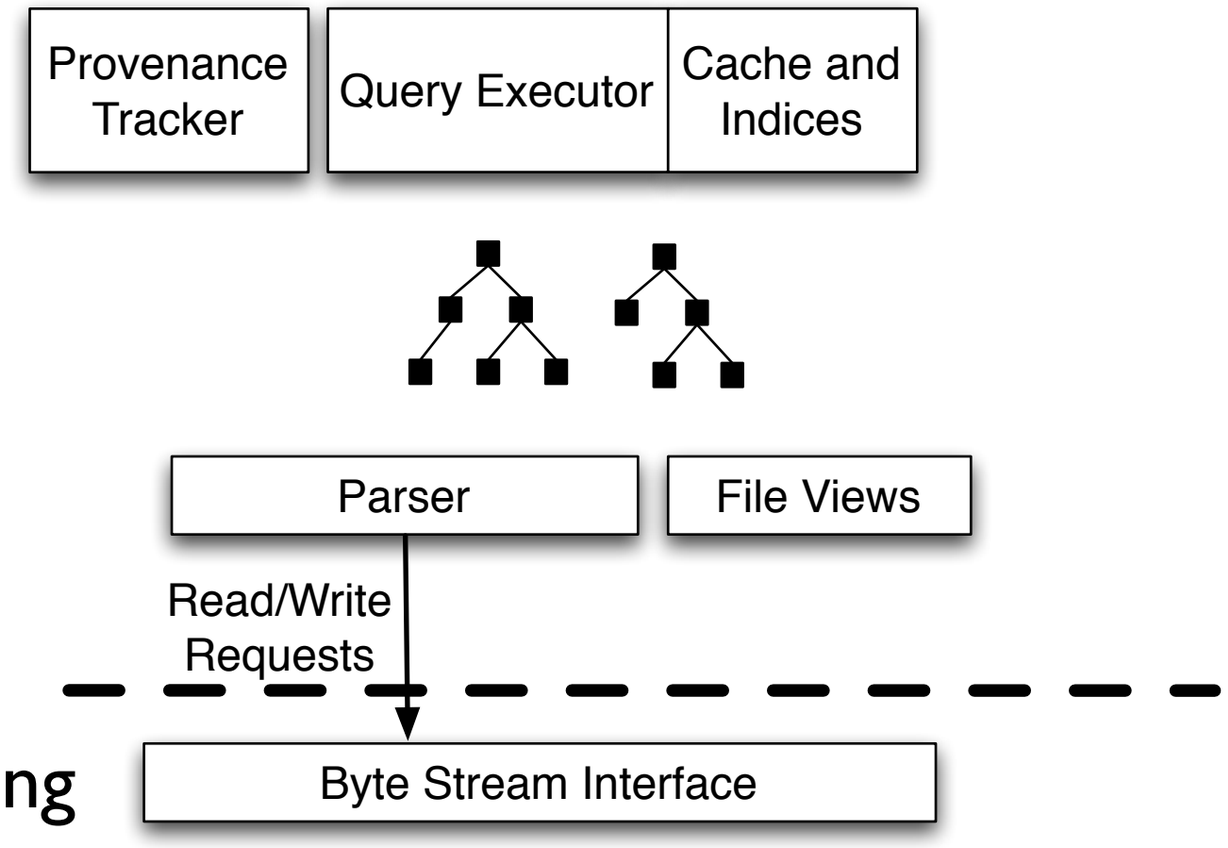
Service 1: Self-Organizable Indexing



- Index Cost vs Benefit:
  - Essential for performance
  - Cost of index maintenance
- Automatic index creation and destruction:
  - Based on recent query patterns
  - Based on index interactions
- Hybrid, partial indexing: e.g. one file has
  - inverted-list index for keywords in titles
  - path index
  - some lazy indexing

12

Service 2: Provenance Tracking



- End-to-end provenance tracking
  - From views to underlying files, and
  - back to (other) views
  - semantic objects (e.g. matrix)
- Automatic bridging of observed and disclosed provenance collection
  - Observed provenance collection on physical level
  - Disclosed provenance by applications on logical level

13

Realization in a Parallel File System

- 1st step: middleware
  - All global Damasc data structures shared via POSIX IO
  - Limited data movement savings
- 2nd step: Ceph extension
  - Distributed query executor, parser, indexer
  - Damasc spans storage clients and servers
  - Leveraging Ceph's intelligent OSDs
- Format-aligned striping
  - Striping strategy based on structure, not bytes
  - Adaptive mapping to distributed data structures
  - Leveraging Ceph's scalable metadata cluster

14

Applications & Damasc

- Applications rely on higher-level APIs:
  - NetCDF, HDF-5, ...
- Declarative Damasc interface
  - Simplification for middleware libraries
  - Cross-workload-adaptive optimization of the storage of particular data sets w/ multiple formats
- Views for overcoming bottlenecks
  - Example: View mapping one file to many files plus automatic indexing (similar to PLFS)

15

Conclusions

- Moving data becomes dominant overhead
- Middleware provides much-needed functionality but not performance
- Damasc adds data management layer to FS
  - Additional semantic information for storage layer
  - Facilitate in-place processing on storage nodes
  - Eventually: full-scale distributed processing

16

Thank you

- Questions and comments?

- Contact: carlosm@cs.ucsc.edu

- Please read our paper!

17