OrangeFS Overview Tutorial

Walt Ligon
Clemson University

Tutorial Goals

- Brief History
- Architecture Overview
- User Interfaces
- Important Features
- Installation/Configuration

Brief History

- **Brief History** ← *Starting here!*
- Architecture Overview
- User Interfaces
- Important Features
- Installation/Configuration

A Brief History

- First there was PVFS
  - V0
    - Aric Blumer (Clemson)
    - 1993
    - PVM User's Meeting Oak Ridge
  - V1
    - Rob Ross (Clemson)
    - 1994 – 2000
  - V2
    - Rob Ross (Argonne), Phil Carns (Clemson), et al.
    - 2000 - Present

PVFS Funding

- NASA
- NSF
  - PACI
  - HECURA
- DOE
  - Office of Science
  - SciDAC
- Government Agencies

PVFS Partnerships

- Clemson U
- Argonne NL
- Northwestern U
- Acxiom
- Ames NL
- U of Michigan
- Ohio St. U
- Ohio Supercomputer Center
- Carnegie Mellon U
- U of Heidleberg
- Sandia NL
- U of Oregon

Emergence of Orange

- Project started in 2007
  – Develop PVFS for "non-traditional" uses
    - Very large number of small files
    - Smaller accesses
    - Much more metadata
  – Robust security features
  – Improved resilience
- Began to see opportunities for broader area
  – Big Data Management
  – Clouds
  – Enterprise

Today and Beyond

- Clemson reclaims primary site
  - As of 2.8.4 began using name “OrangeFS”
  - Omnibond offers commercial support
  - Currently on 2.8.6
  - Version 2.9.0 soon to be released with new features
  - 2.10 is internal development version for …
- OrangeFS 3.0

What to Expect in 3.0

- Totally Redesigned Object Model
  - Replication
  - Migration
  - Dynamic configuration
  - Hierarchical Storage Management (tiers)
  - Metadata support for external devices (archive)
  - Rule-based security model
  - Rule-based policies

Architecture Overview

- Brief History
- **Architecture Overview** ← *You are here!*
- User Interfaces
- Important Features
- Installation/Configuration

Architecture Overview

- OrangeFS is designed for parallel machines
  - Client/Server architecture
  - Expects multiple clients and servers
- Two major software components
  - Server daemon
  - Client library
- Two auxiliary components
  - Client core daemon
  - Kernel module

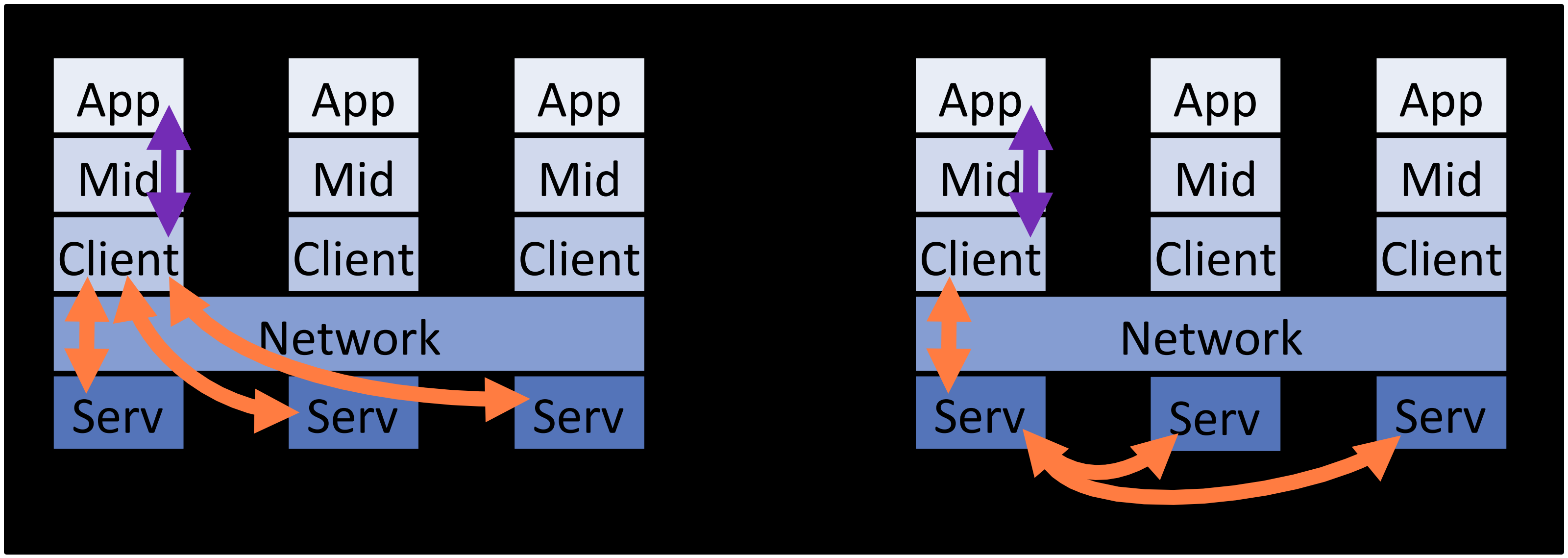
User Level File System

- Everything runs at the user level on clients and servers
  - Exception is the kernel module, which is really just a shim – more details later
- Server daemon runs on each server node and serves requests sent by the client library, which is linked to client code on compute nodes

Networking

- Client and server communicate using an interface called "BMI"
  - BMI supports tcp/ip, ib, mx, portals
  - Module design allows addition of new protocols
  - Requests use PVFS protocol similar to that of NFS but expanded to support a number of high performance features

Server-to-Server Communication



Traditional Metadata Operation

Create request causes client to communicate with all servers O(p)

Scalable Metadata Operation

Create request communicates with a single server which in turn communicates with other servers using a tree-based protocol O (log p)

Storage

- The OrangeFS server interacts with storage on the host using an interface layer named “trove”
  - All storage objects referenced with a “handle”
  - Storage objects consist of two components
    - Bytestreams – sequences of data
    - Key/Value Pairs – data items that are accessed by key
  - As currently implemented …
    - Bytestreams with the local file system (data)
    - Key/value pairs with BerkeleyDB (metadata)

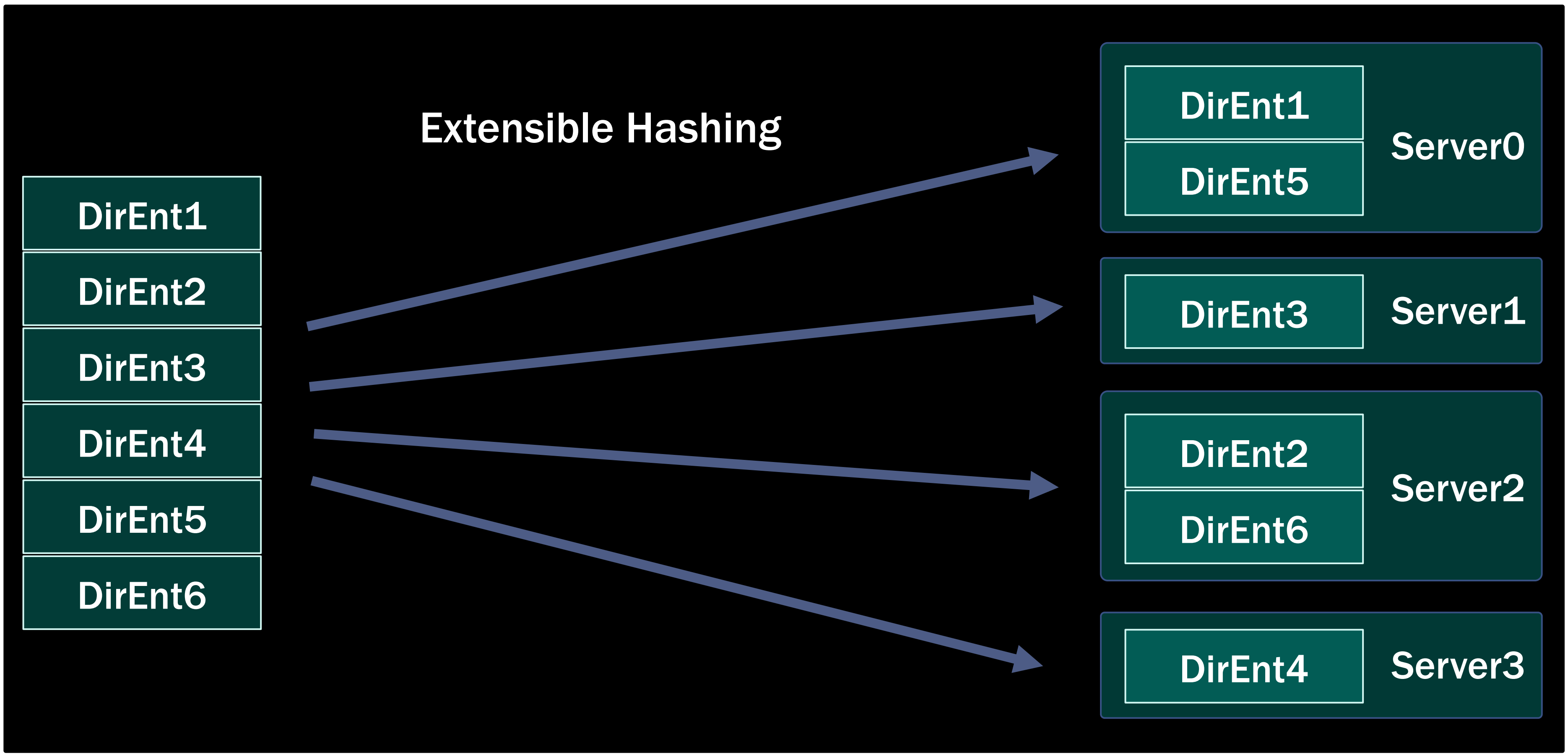
File Model

- Every file consists of two or more objects
  - Metadata object contains
    - Traditional file metadata (owner, permissions, etc.)
    - List of data object handles
    - FS specific items (layout, distribution, parameters)
    - User-defined attributes
  - Data objects contain
    - Contents of the file
    - Attributes specific to that object (usually not visible)

Directories

- Work just like files except
  - Data objects contain directory entries rather than data
  - Directory entries have entry name and handle of that entry's metadata object (like inode number)
  - Extendable hash function selects which data object contains each entry
  - Mechanisms based on GIGA+ manage consistency as directory grows or shrinks

Distributed Directories



Policies

- Objects of any type can reside on any server.
  - Random selection (default) used to balance
  - User can control various things
    - Which servers hold data
    - Which servers hold metadata
    - How many servers a given file or directory are spread across
    - How those servers are selected
    - How file data is distributed to the servers
  - Parameters can be set in configuration file, many on a directory, or for a specific file

User Interfaces

- Brief History
- Architecture Overview
- **User Interfaces** ← *Half way!*
- Important Features
- Installation/Configuration

User Interfaces

- System Interface
- VFS Interface
- Direct Access Library
- MPI-IO
- Windows Client
- Web Services Module
- Utilities

The System Interface

- Low-level interface of the client library
  - PVFS_sys_lookup()
  - PVFS_sys_getattr()
  - PVFS_sys_setattr()
  - PVFS_sys_io()
- Based on the PVFS request protocol
- Designed to have interfaces built on them
- Provide access to all of the features
- NOT a POSIX-like interface

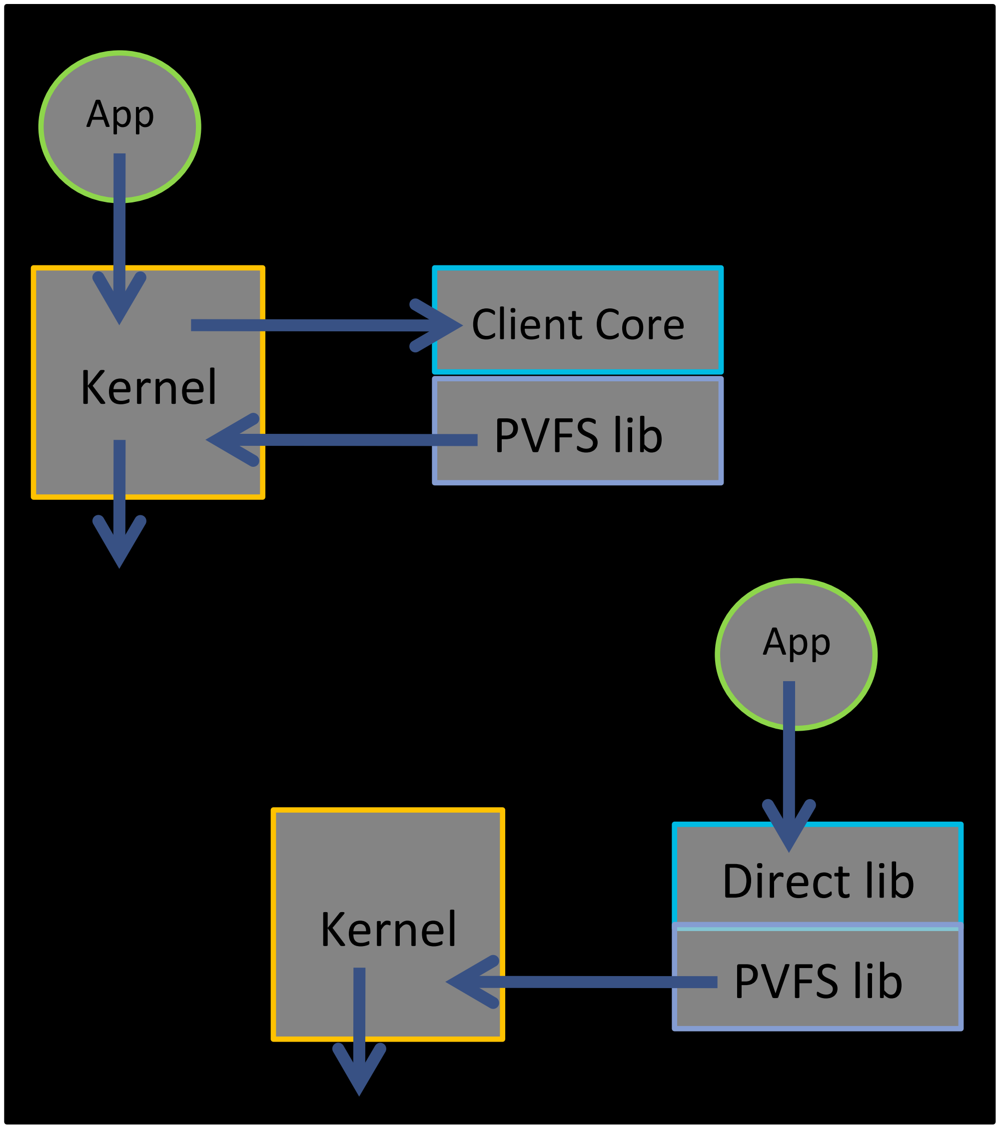
The VFS Interface

- Linux kernel module allows OrangeFS volumes to be mounted and used like any other
  - Limited mmap support
  - No client side cache
  - A few semantic issues
- Must run client_core daemon
  - Reads requests from the kernel then calls library
- The most common and convenient interface
- BSD and OS X support via FUSE

The Direct Library

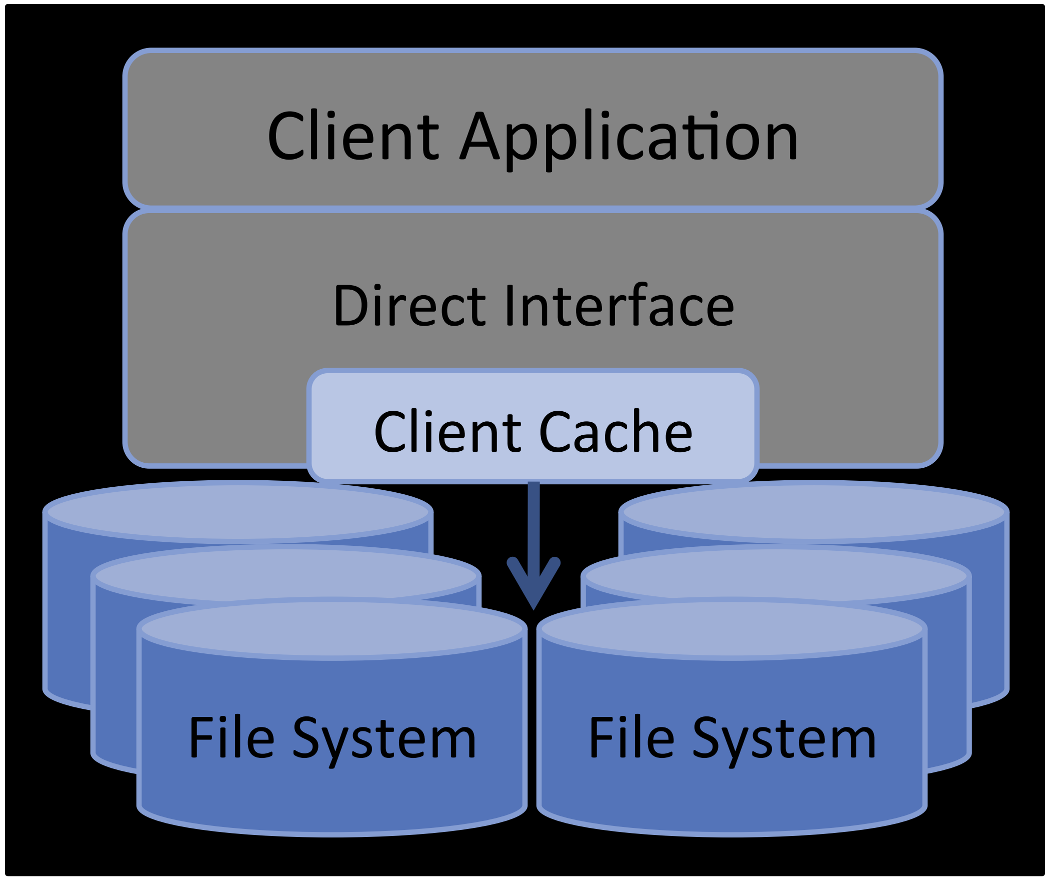
- Interposition library for file related stdio and Linux system calls
- Links programs directly to the client library
- Can preload the shared library and run programs without relinking
- Faster, lower latency, more efficient than VFS
- Configurable user-level client cache
- Best option for serious applications

Direct Access Library



- Implements:
  - POSIX system calls
  - Stdio library calls
- Parallel extensions
  - Noncontiguous I/O
  - Non-blocking I/O
- MPI-IO library

Direct Interface Client Caching

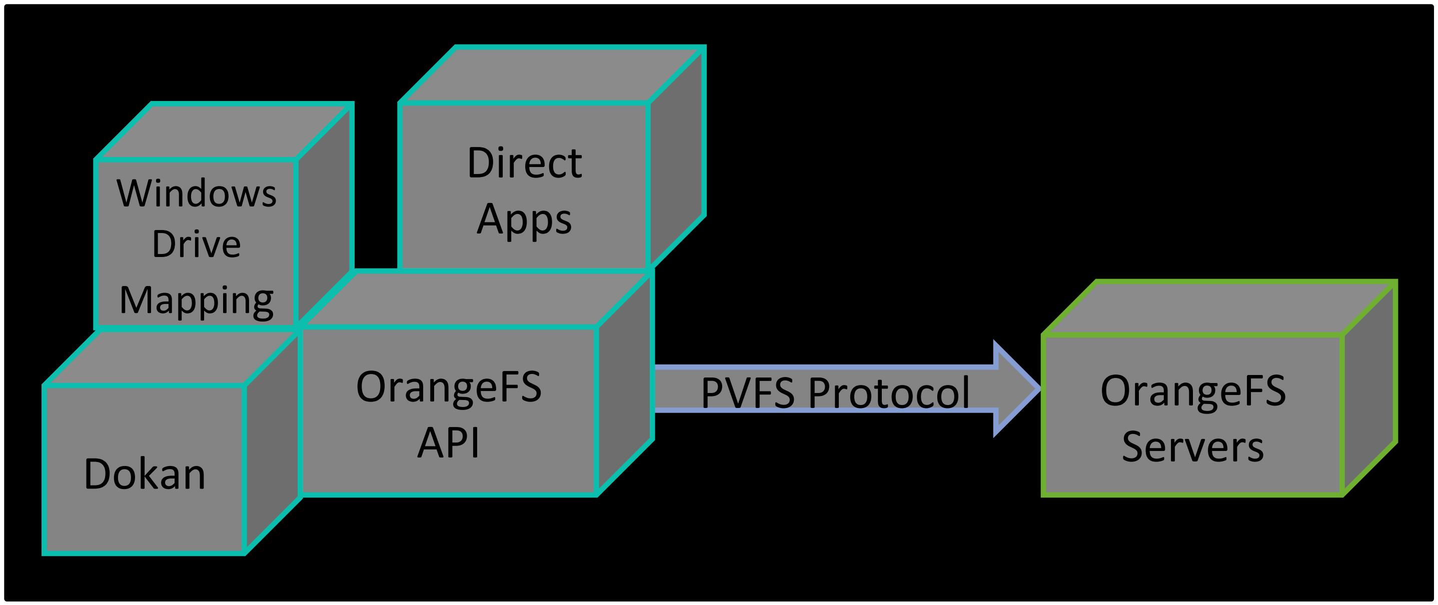


- Direct Interface enables Multi-Process Coherent Client Caching for a single client

MPI-IO

- Most MPI libraries provide ROMIO support, which has support for direct access to the client library
- A number of specific optimizations for MPI programs, especially for collective IO
  - Aggregators, data sieving, data type support
- Probably the best overall interface, but only for MPI program

Windows Client

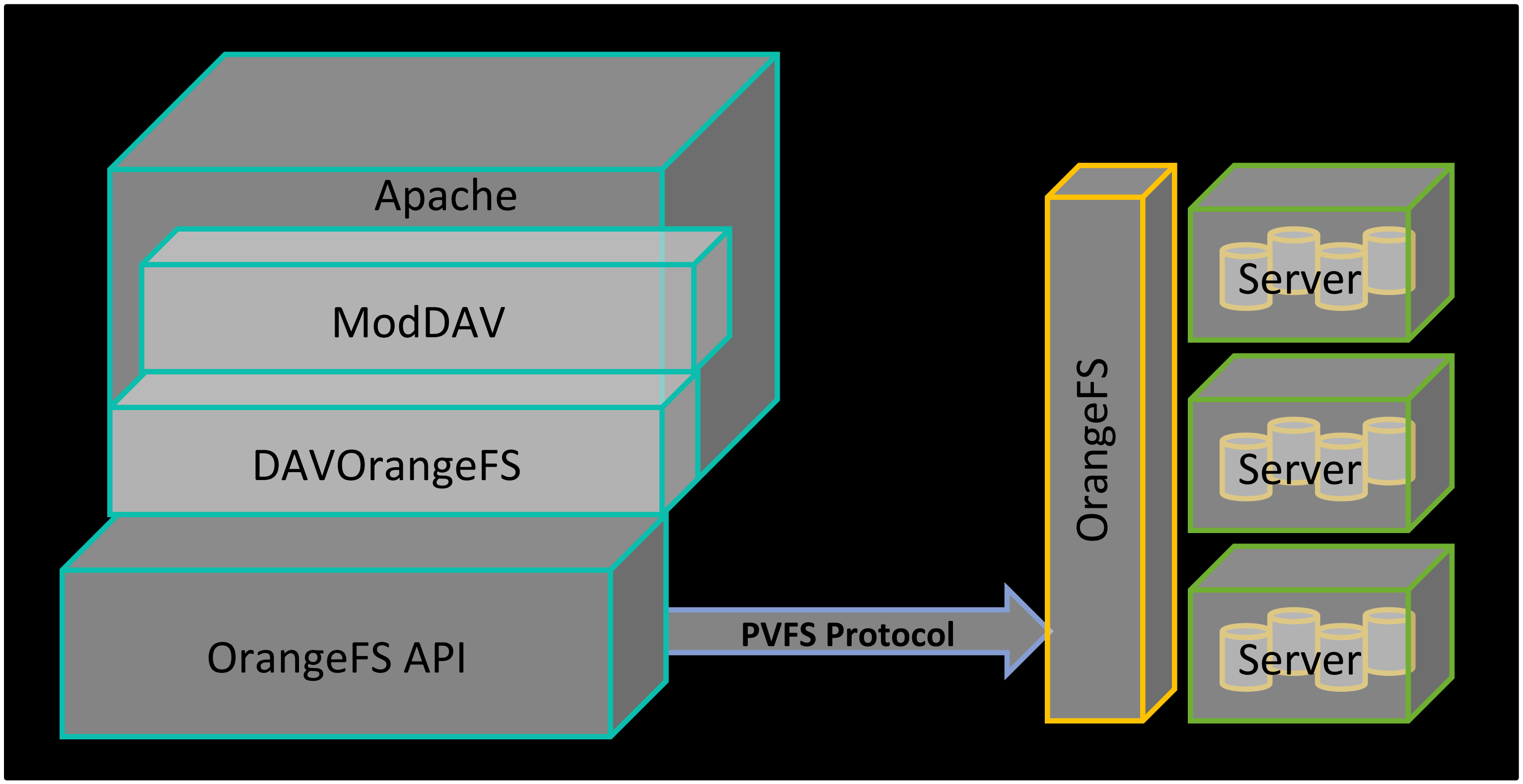


- Supports Windows 32/64 bit
- Server 2008, R2, Vista, 7

Web Services Module

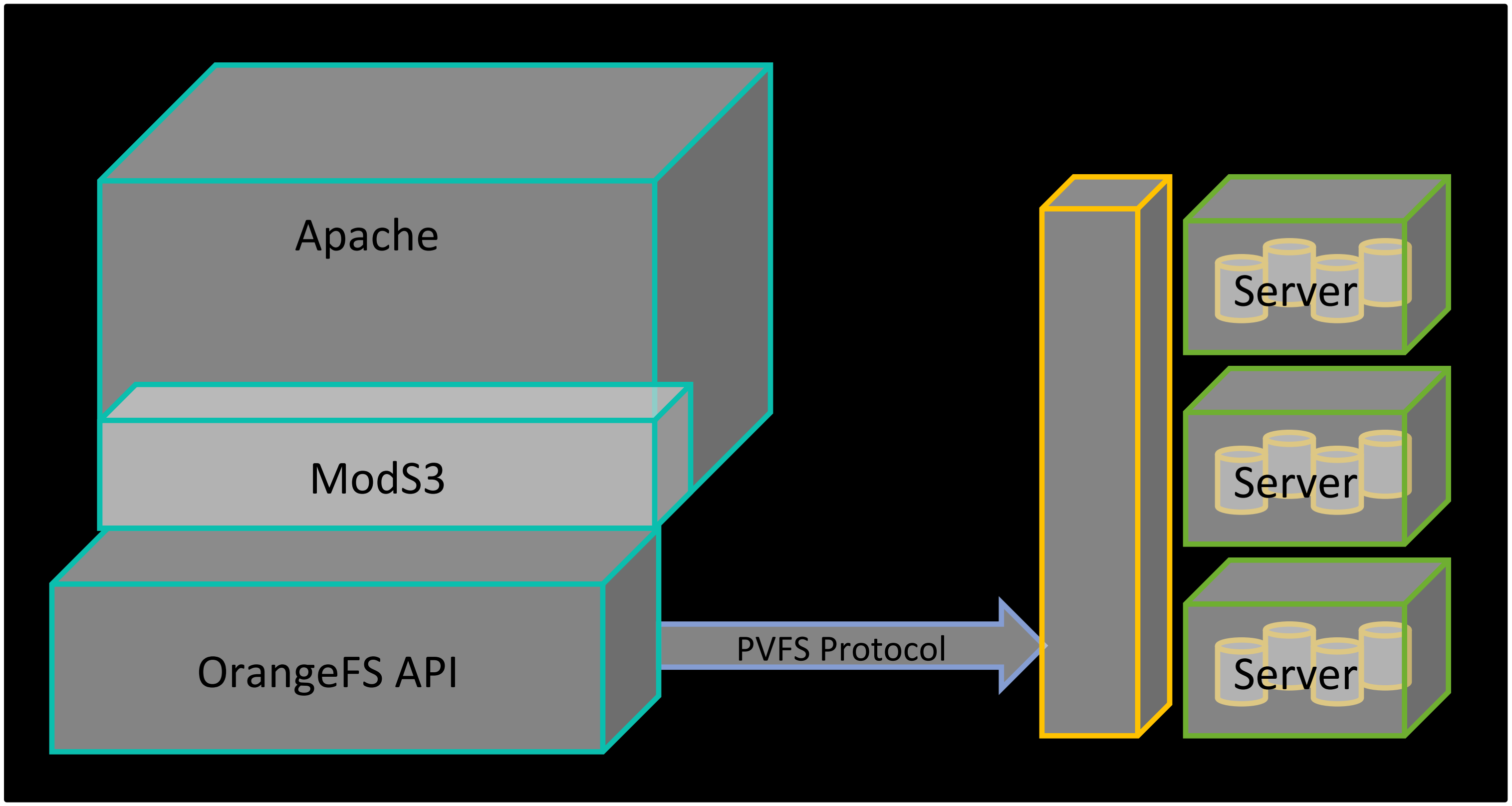
- WebDAV
- S3
- REST

WebDAV



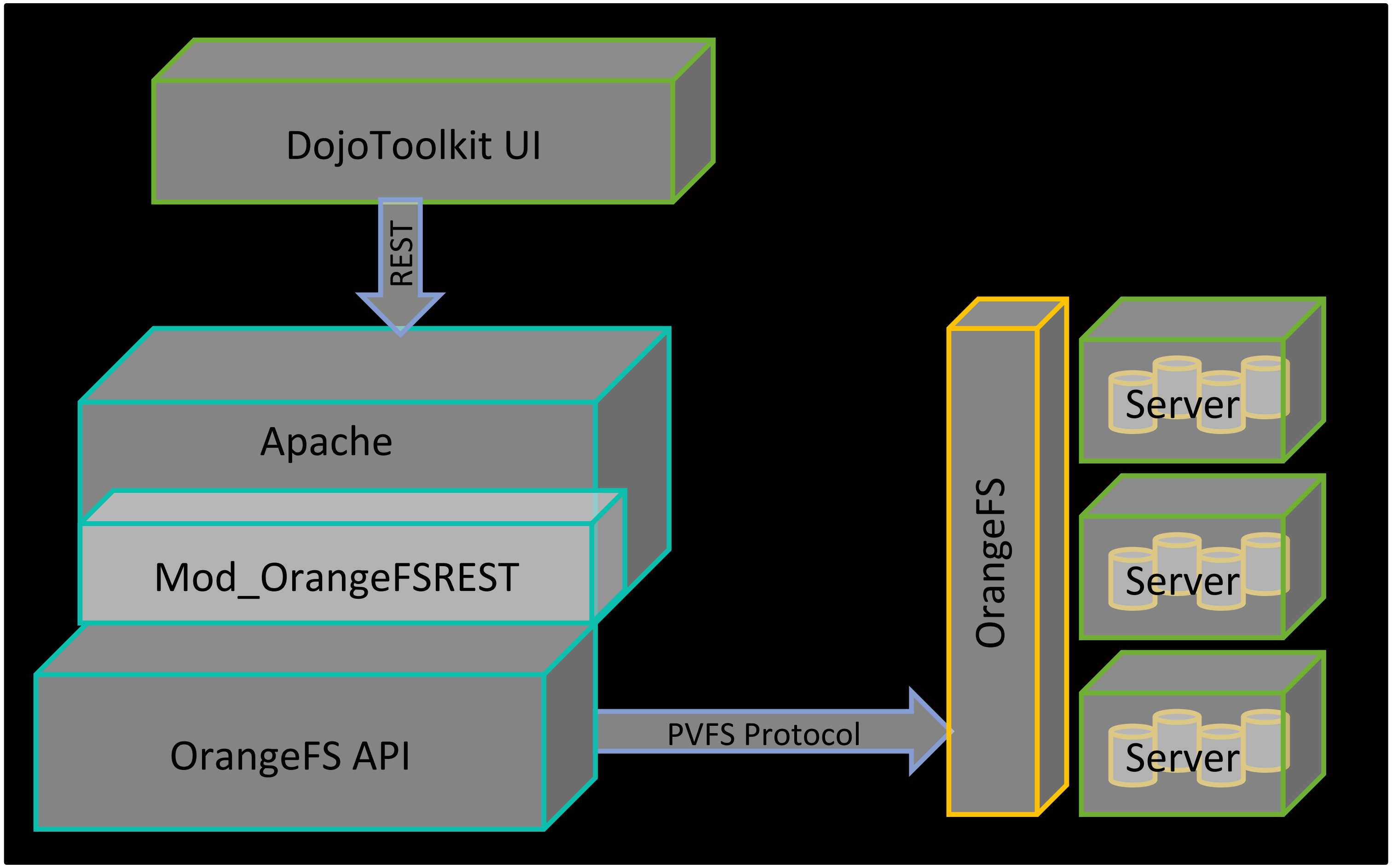
- Supports DAV protocol and tested with (insert reference test run – check with mike)
- Supports DAV cooperative locking in metadata

S3



Apache
ModS3
OrangeFS API
PVFS Protocol
Server
Server
Server

Admin REST Interface



Important Features

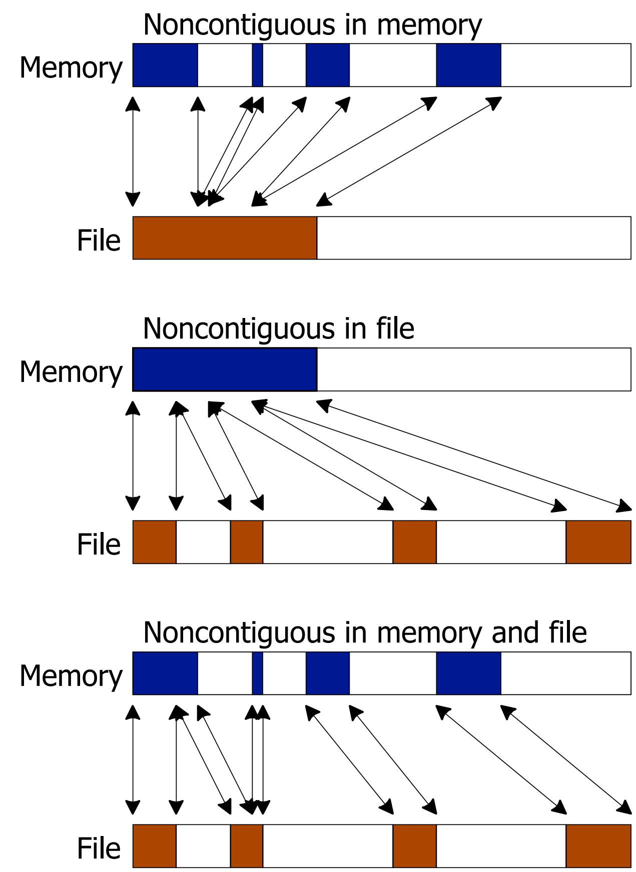
- Brief History
- Architecture Overview
- User Interfaces
- **Important Features** ← *Over the hump!*
- Installation/Configuration

Important Features

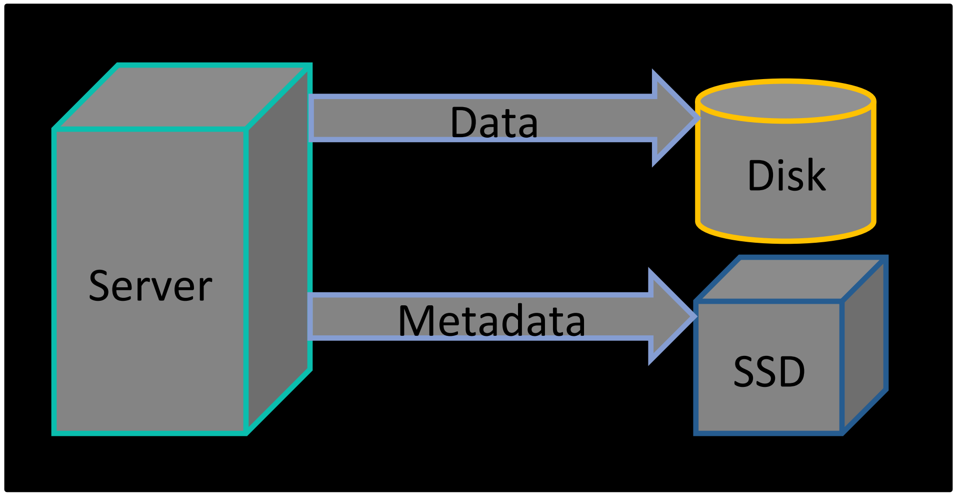
- Already Touched On
  - Parallel Files (1.0)
  - Stateless, User-level Implementation (2.0)
  - Distributed Metadata (2.0)
  - Extended Attributes (2.0)
  - Configurable Parameters (2.0)
  - Distributed Directories (2.9)
- Non-Contiguous I/O (1.0)
- SSD Metadata Storage (2.8.5)
- Replicate On Immutable (2.8.5)
- Capability-based Security (2.9)

Non-Contiguous I/O

- Noncontiguous I/O operations are common in computational science applications
- Most PFSs available today implement a POSIX-like interface (open, write, close)
- POSIX noncontiguous support is poor:
  - readv/writev only good for noncontiguous in memory
  - POSIX listio requires matching sizes in memory and file
- Better interfaces allow for better scalability

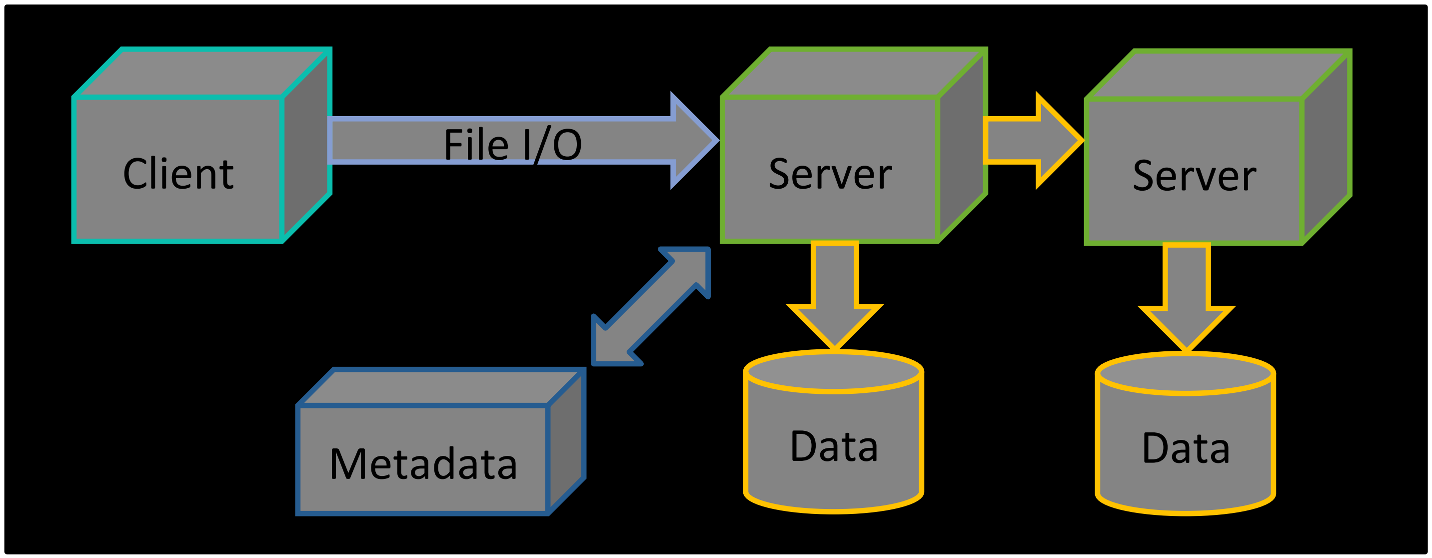


SSD Metadata Storage



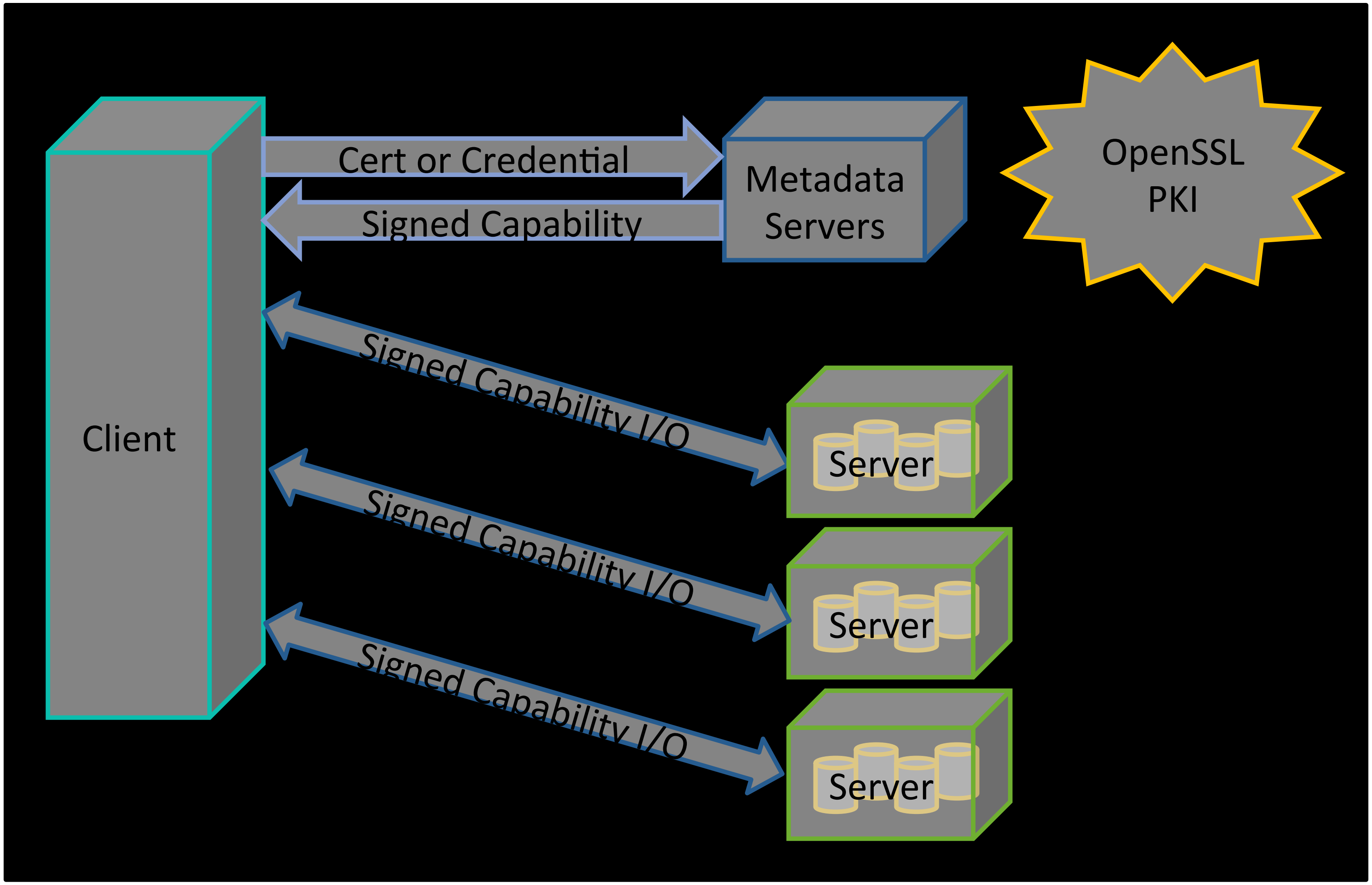
- Writing metadata to SSD
  - Improves Performance
  - Maintains Reliability

Replicate On Immutable



- First Step in Replication Roadmap
- Replicate data to provide resiliency
  - Initially replicate on Immutable
  - Client read fails over to replicated file if primary is unavailable

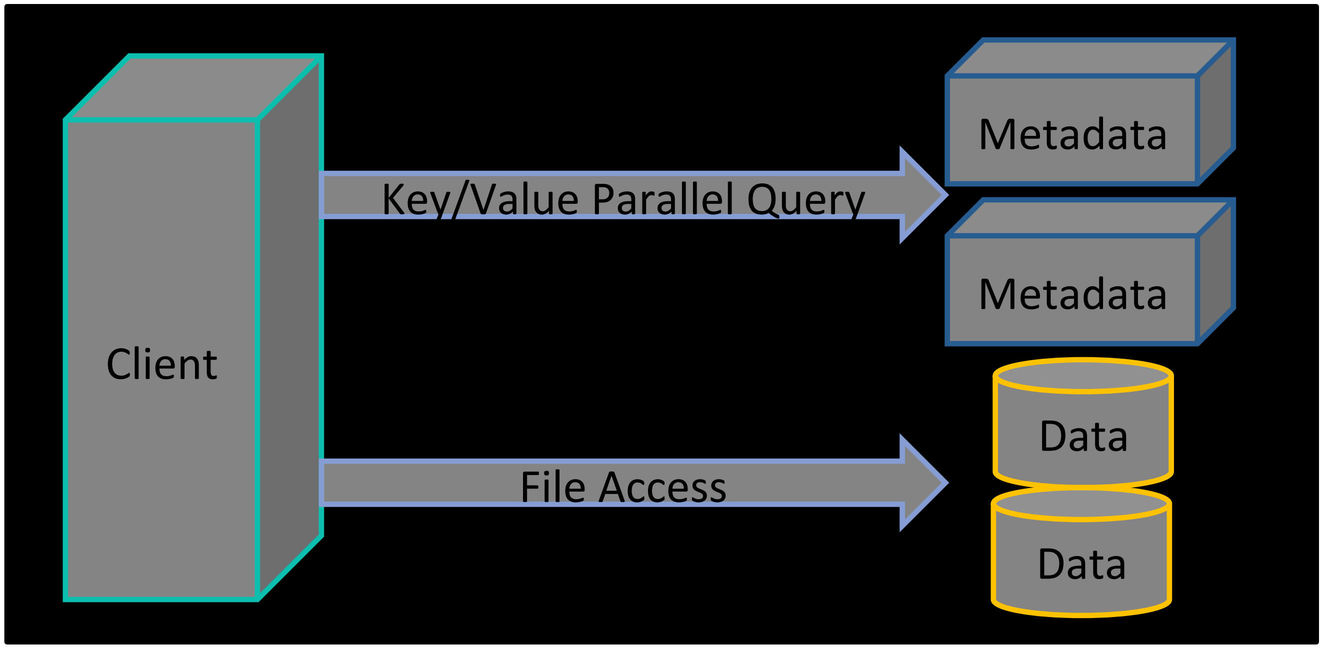
Capability Based Security



Future Features

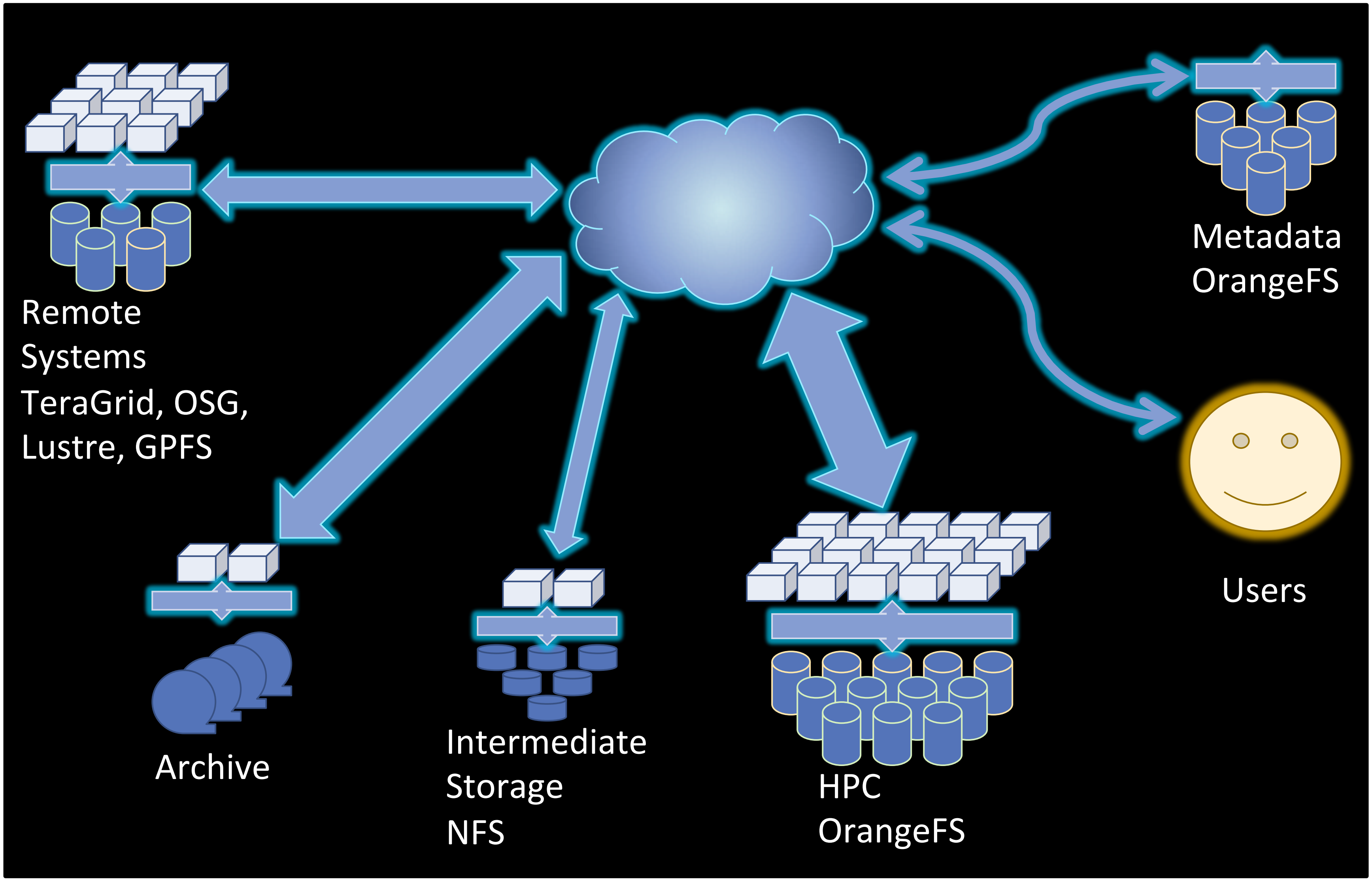
- Attribute Based Search
- Hierarchical Data Management

Attribute-Based Search



- Client tags files with Keys/Values
- Keys/Values indexed on Metadata Servers
- Clients query for files based on Keys/Values
- Returns file handles with options for filename and path

Hierarchical Data Management



Installation/Configuration

- Brief History
- Architecture Overview
- User Interfaces
- Important Features
- **Installation/Configuration** ← *Last One!*

Installing OrangeFS

- Distributed as a tarball
  - You need
    - GNU Development environment
      - GCC, GNU Make, Autoconf, flex, bison
    - Berkeley DB V4 (4.8.32 or later)
    - Kernel Headers (for kernel module)
    - A few libraries
      - OpenSSL, etc.

Building OrangeFS

```
# ./configure --prefix=<install_dir>
              --with-kernel=<kernel_dir>
              --with-db=<db_dir>
# make
# make install
# make kmod
# make kmod_install
```

Configuring OrangeFS

- Servers
  - Many installations have dedicated servers
    - Multi-core
    - Large RAM
    - RAID subsystems
    - Faster network connections
  - Others have every node as both client and server
    - Checkpoint scratch space
    - User-level file system
    - Smaller, more adaptable systems

Server Storage Modules

- AltIO
  - General-purpose
  - Uses Linux AIO to implement reads/writes
  - Typically utilizes Linux page cache
- DirectIO
  - Designed for dedicated storage with RAID
  - Uses Linux threads and direct IO
  - Bypasses Linux page cache

Parameters to Consider

- Default number of IO Servers
- Default strip size
- RAID parameters
  - Cache behavior
  - Block sizes
- Local file system layout (alignment)

Metadata Servers

- As many or as few as you need
  - Every server can be a metadata server
    - Metadata access load spread across servers
  - Dedicated metadata servers
    - Keep metadata traffic from data servers
    - Custom configured for metadata
  - Depends on workload and data center resources

Clients

- Minimum each client needs libpvfs2
  - Direct access lib (libofs)
  - MPI libraries
  - Precompiled programs
  - Web Services
- Most clients will want VFS interface
  - pvfs2-client-core
  - Modpvfs2
- Need a mount string pointing to a server (fstab)
  - Many clients, spread load across servers

Learning More

- www.orangefs.org web site
  - Releases
  - Documentation
  - Wiki
- pvfs2-users@beowulf-underground.org
  - Support for users
- pvfs2-developers@beowulf-underground.org
  - Support for developers

Commercial Support

- www.orangefs.com & www.omnibond.com
  - Professional Support & Development team

QUESTIONS & ANSWERS