OBFS: File Systems for Object-Based Storage Devices

<u>Feng Wang</u>, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long

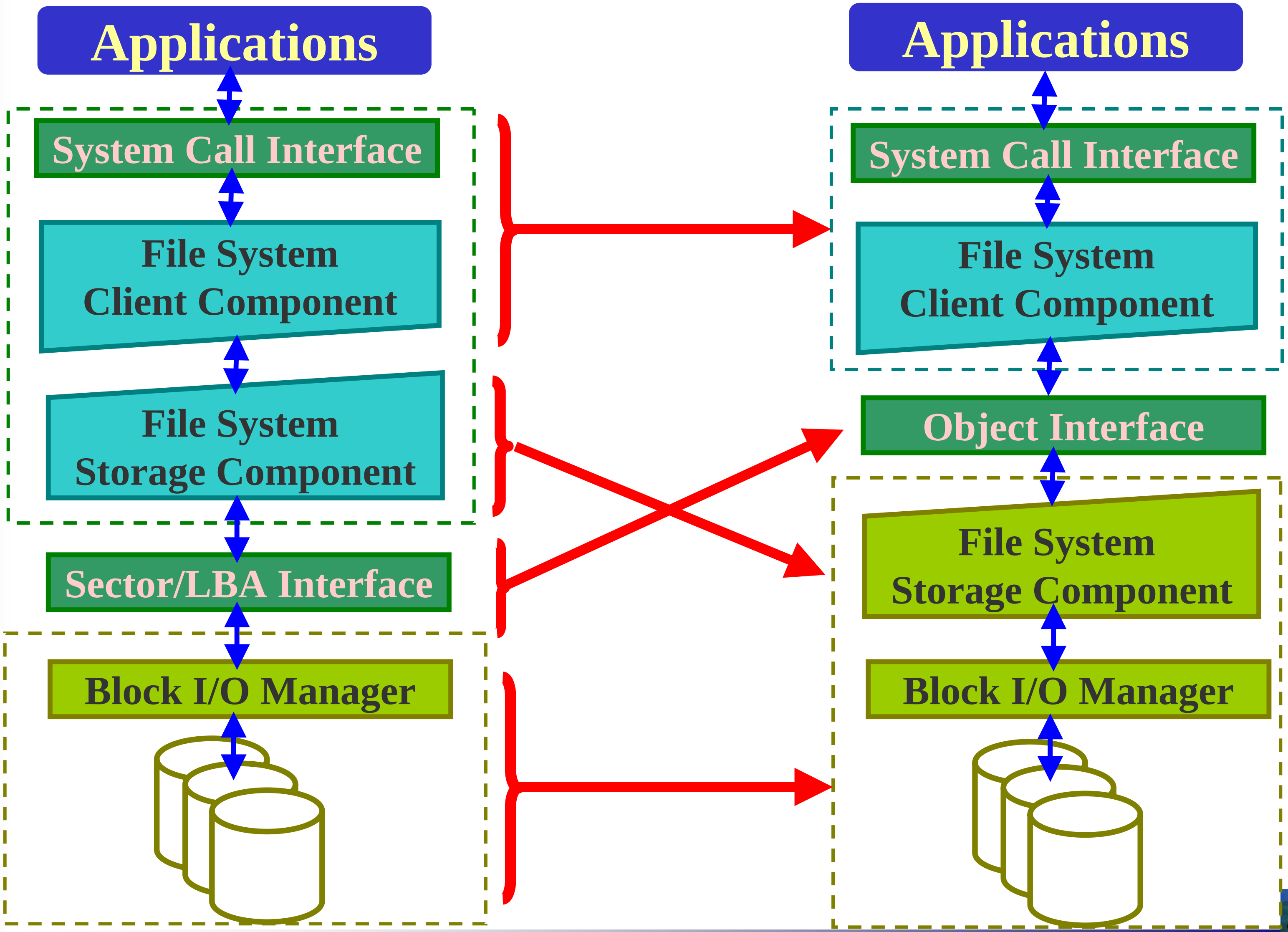
Storage Systems Research Center
University of California, Santa Cruz

NASA/IEEE MSST 2004
12th NASA Goddard/21st IEEE Conference on Mass Storage Systems & Technologies
The Inn and Conference Center
University of Maryland University College
Adelphi MD USA
April 13-16, 2004

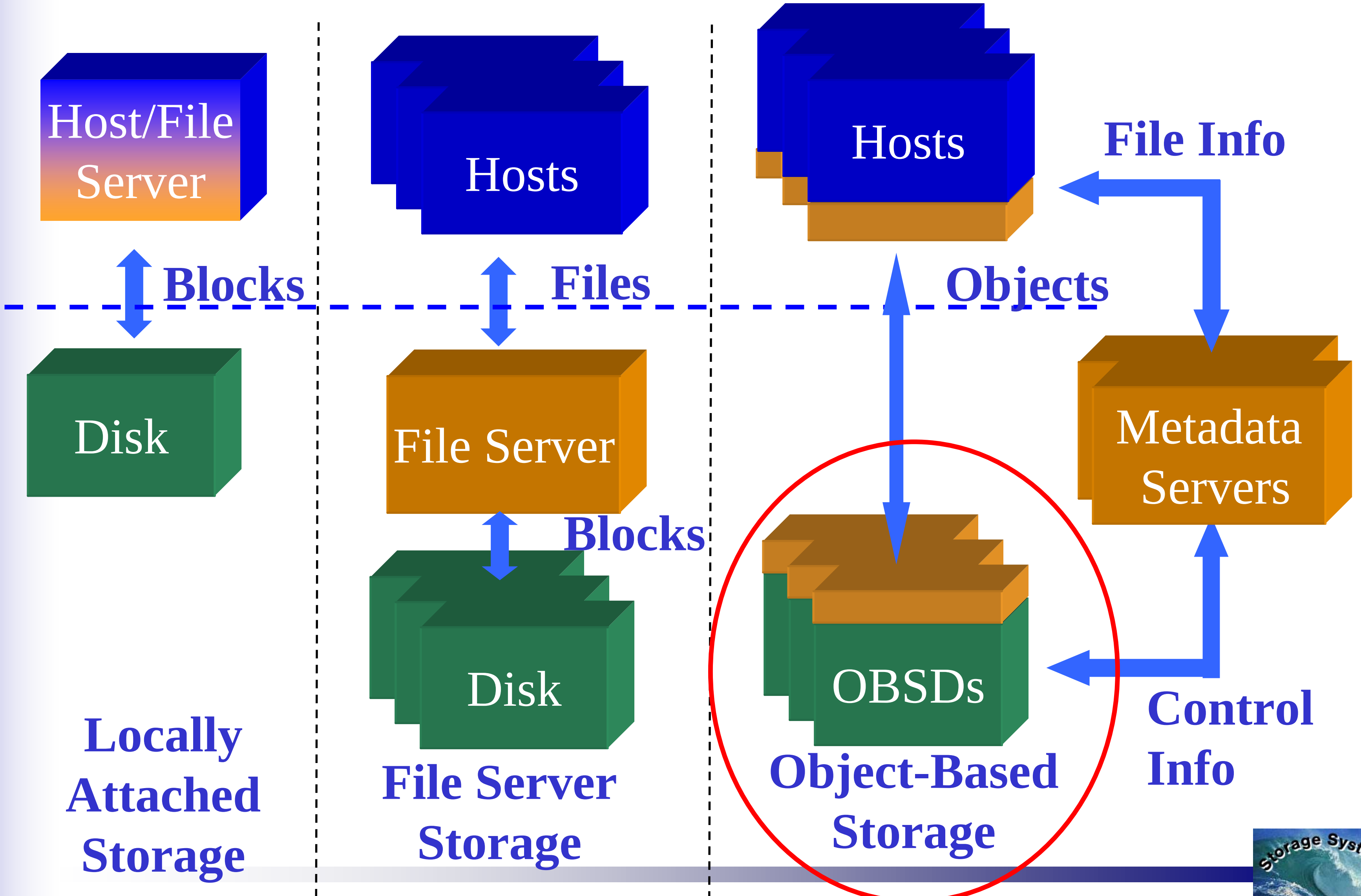
The Storage Model is Changing

- Extremely high storage capacity, bandwidth and scalability is desired
  - Scientific computing environment
  - Visualization system
- Existing storage systems cannot scale to this level
  - Bottlenecks caused by centralized control mechanisms
- Object-based storage is a promising alternative
  - Scalable
  - Supports parallel access
  - Highly distributed metadata and data management
- Storage management needs to adapt to this new model
  - File and object storage are fundamentally different
  - Need Object-Based File Systems (OBFS)

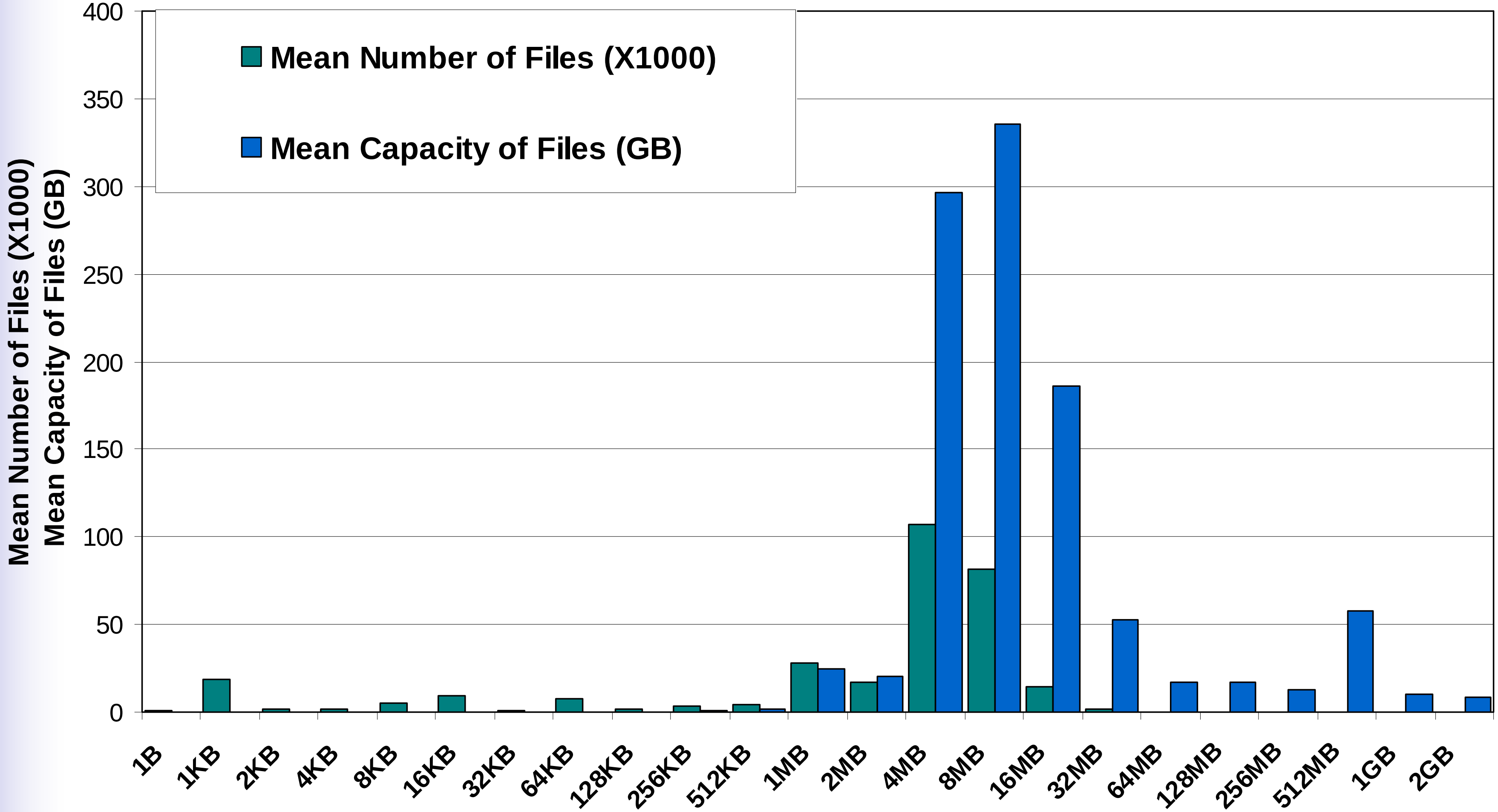
Object-Based Storage Model



Different Storage Models In Action



Workload Characteristics



- Data courtesy of Lawrence Livermore National Laboratory

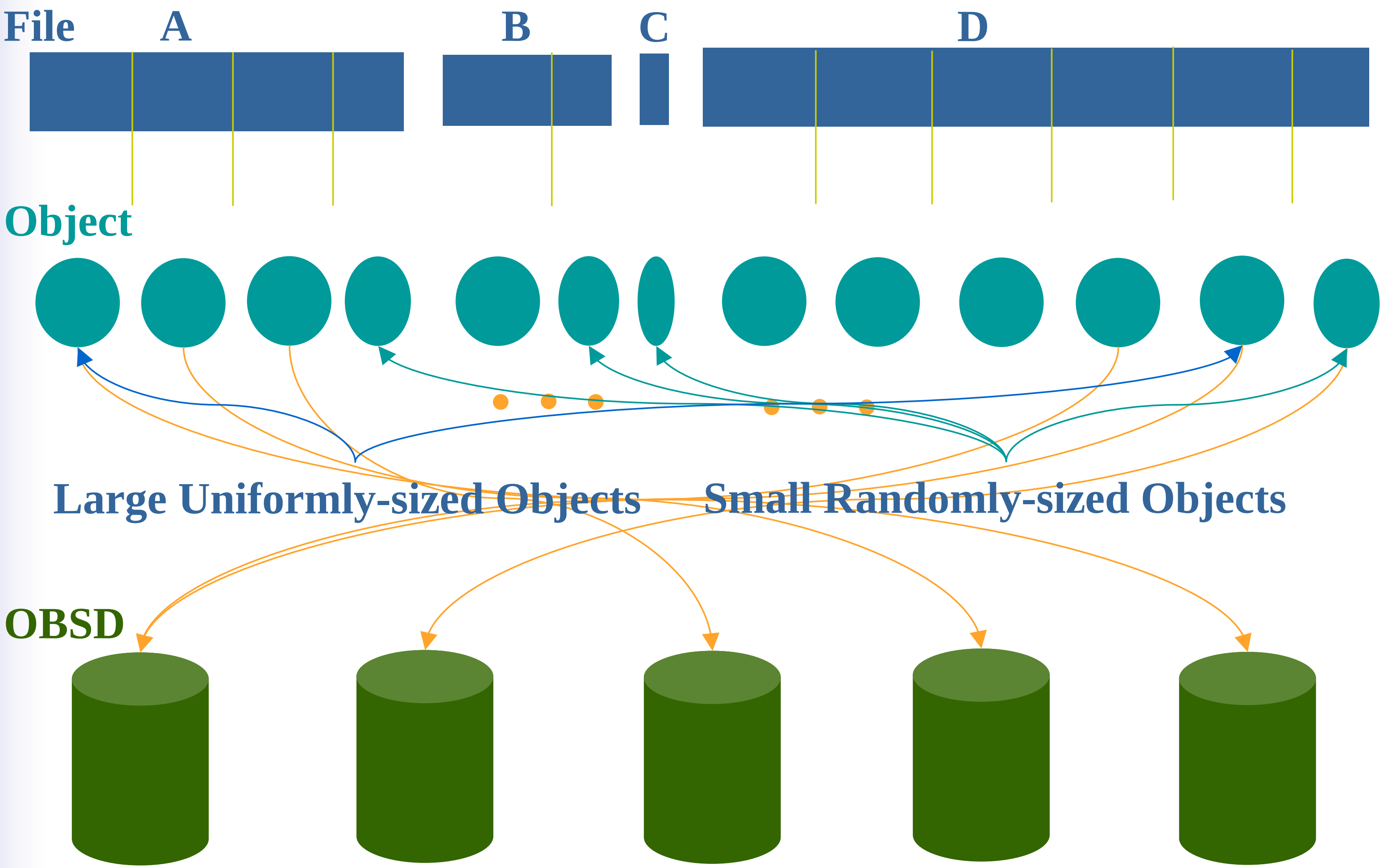
LLNL File Access Pattern

- Files are accessed in parallel by multiple clients
  - Up to 10,000 clients may access a single file
- Writes are deeply buffered at the client main memory
- Almost all data are transferred by large sequential requests
- File accesses switch between several typical patterns

- Simulation Stage
  - Write intensive
  - Very little read
  - Multiple clients access one file
  - Sequential and random accesses
  - Memory dump
- Post Analysis Stage
  - Read intensive
  - Very little write
  - Totally random accesses

Typical Data Access Scenarios

Object Workload



Object Workload Characteristics

- Objects are more uniformly sized than files
  - System stripe unit size provides upper bound.
- Large objects dominate
  - More than 80%
- Weak inter-object locality
  - Objects in a file tend to be distributed to different OBSDs
- Strong intra-object locality
  - Objects tend to be accessed as a whole

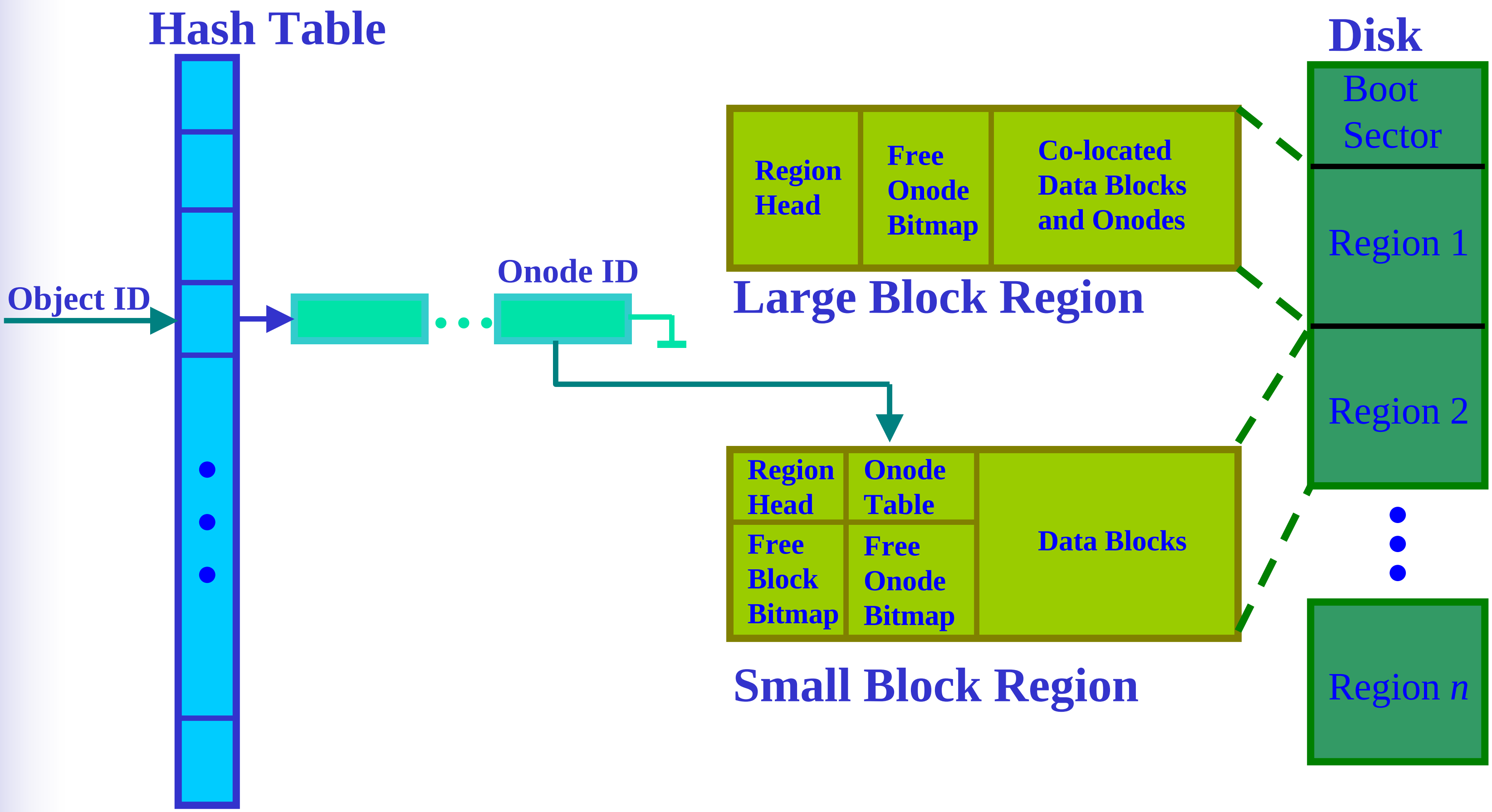
OBFS Design Principles

- Flat object name space
  - Fast mapping and retrieval of objects is essential
- Data layout optimization for object workload
  - Most objects are large and uniformly sized
- High throughput
- High reliability
- Simple

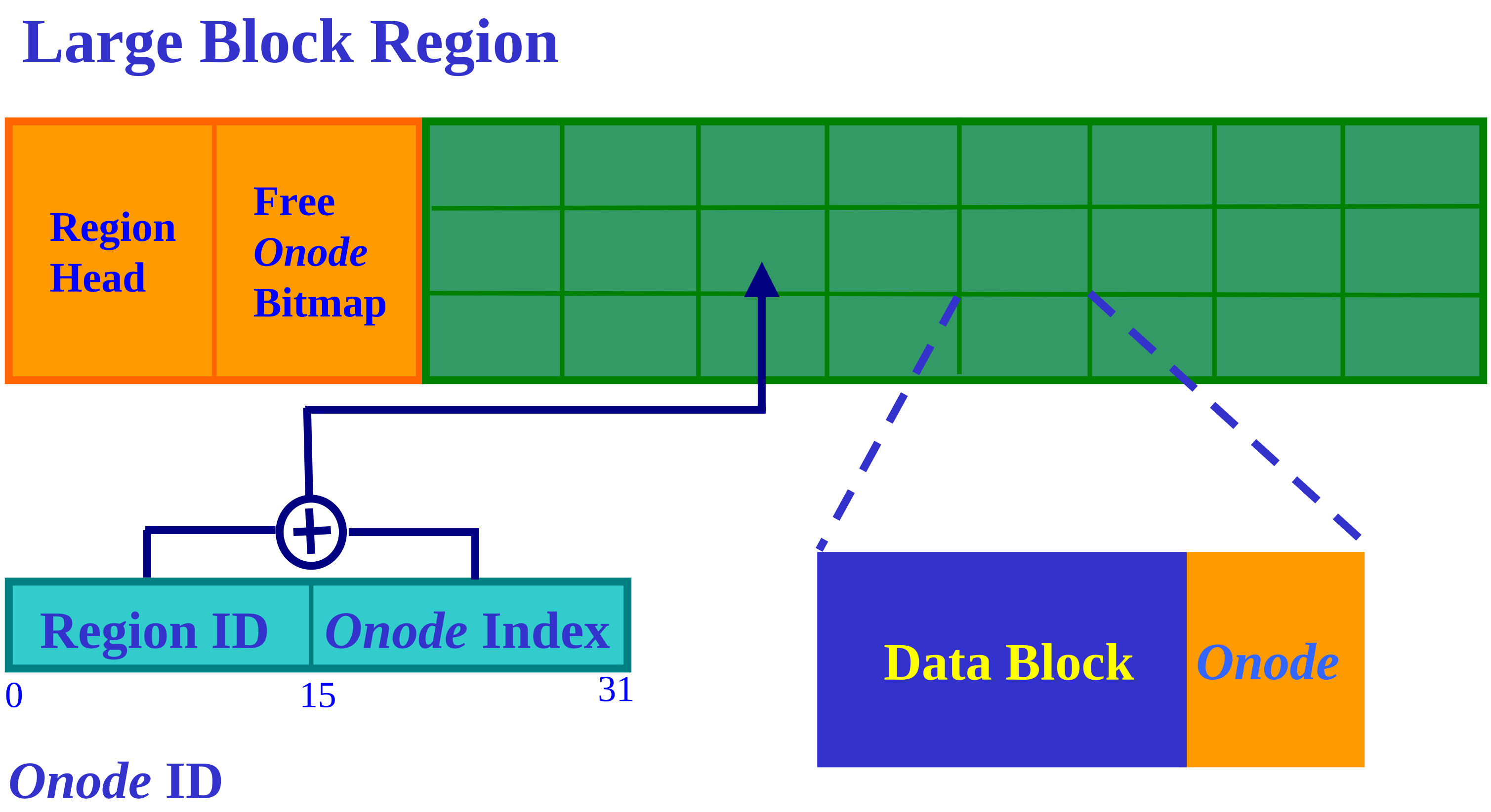
Preliminary Design

- Variably-sized blocks
  - Large blocks optimized for large objects
  - Small blocks guarantee efficient space usage
- Region
  - Keep blocks of the same size together
  - Each object lives in a single region
- Hash table to map and retrieve objects
- Collocated objects and their metadata (Onode)

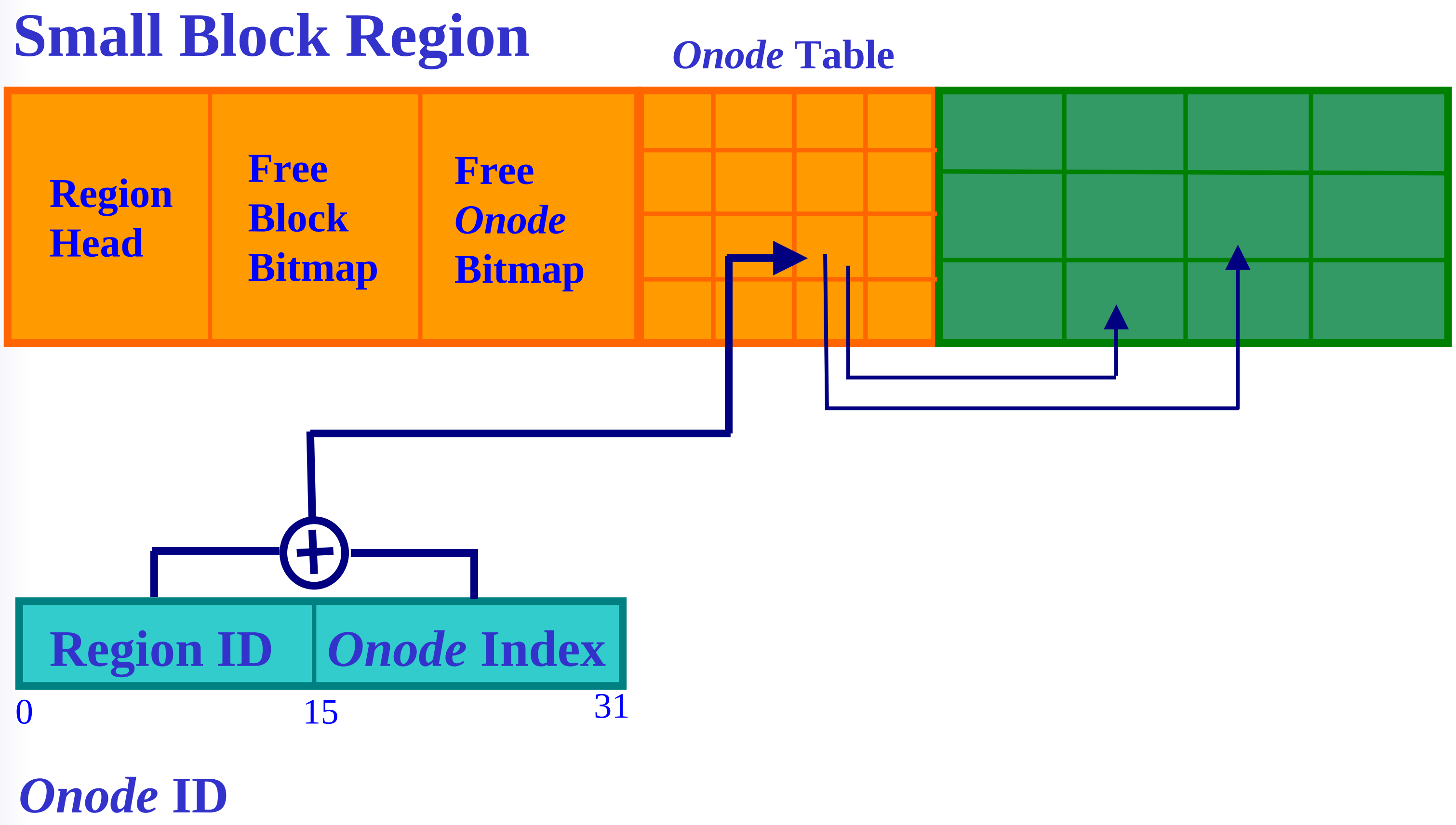
OBFS Design


Hash Table
Object ID
Onode ID
Region Head
Free Onode Bitmap
Co-located Data Blocks and Onodes
Large Block Region
Region Head
Onode Table
Free Block Bitmap
Free Onode Bitmap
Data Blocks
Small Block Region
Disk
Boot Sector
Region 1
Region 2
Region n

OBFS Design – Region Structure and Data Layout



OBFS Design – Region Structure and Data Layout



OBFS Design – Allocation Policy

- A large object is allocated to the nearest free large block or nearest free region
- A small object is allocated blocks from a single region
  - Start searching from the nearest small block region
  - Calculate the minimal number of block extents that are allocated to the object
  - If the number of extents is smaller than a pre-defined threshold, the object will be assigned to this region
  - Else, find another small block region and repeat this process

OBFS Design – File System Reliability, Consistency and Recoverability

- Synchronous object and object metadata writes
  - Improve data reliability
  - Simplify consistency checking scheme
  - Simplify recovery scheme
- Asynchronous file system data structure updates
  - Hash table, free ***onode*** bitmap and free block bitmap
- File system consistency
  - The object metadata stores redundant information of the file system data structures
  - File system can be brought back to consistent state by regenerating file system data structures through redundant information maintained in the object metadata

Performance Evaluation

- Experimental setup:
  - Red Hat Linux, kernel version 2.4.0
  - Executed on a PC with a 1 GHZ Pentium III CPU and 512 MB of RAM
  - A dedicated 80 GB Maxtor D740X-6L disk
  - Ext2, Ext3 and XFS synchronously mounted
- OBFS compared against Linux Ext2, Ext3, and XFS
- More experimental results in paper

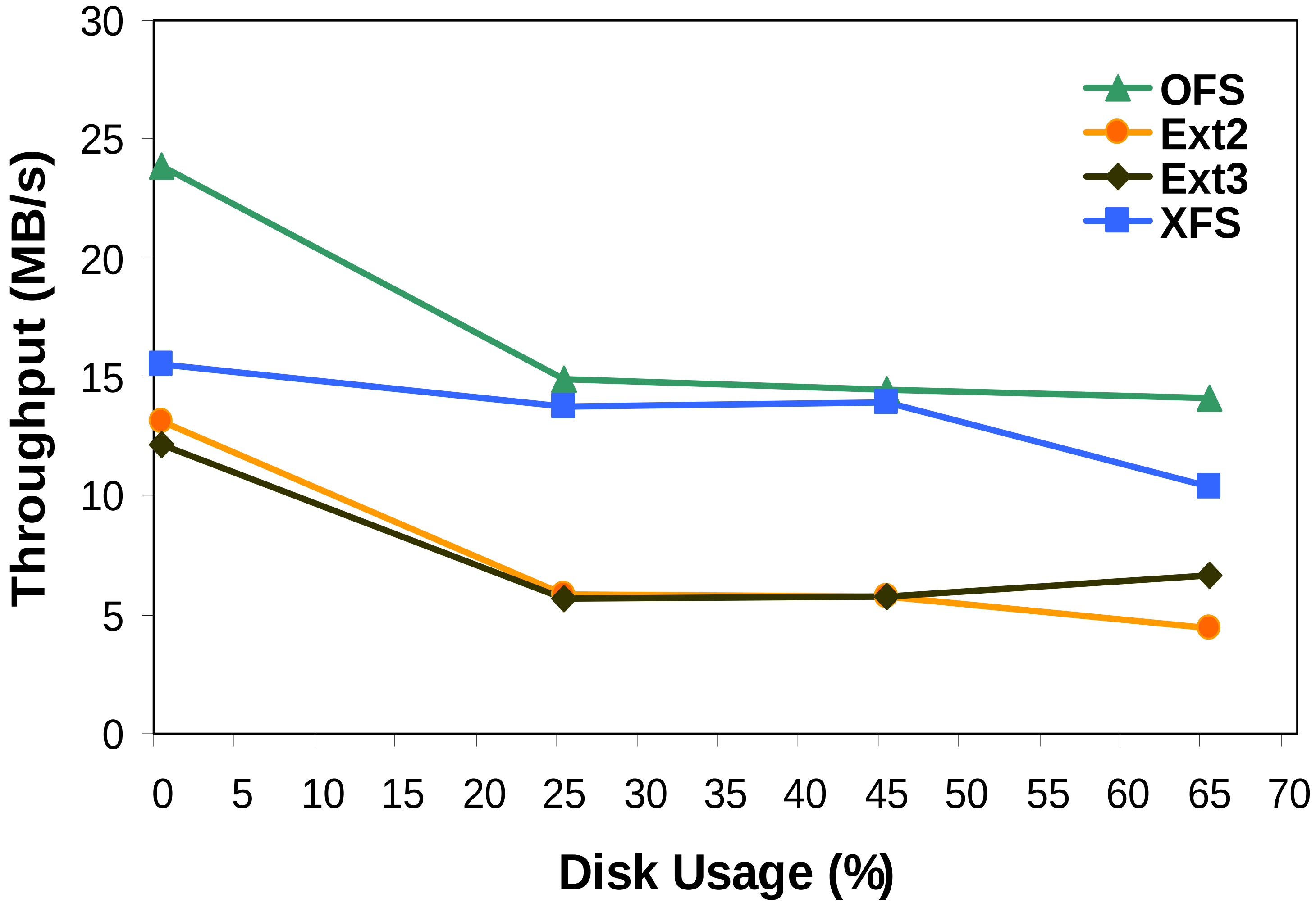
Performance Evaluation – Object Benchmarks

- Derived from LLNL workload
- Consist of sequence of object operations
  - 80% of all objects are large objects (512 KB)
  - Small object are uniformly distributed between 1KB and 512 KB
  - Read, write, rewrite, and delete account for 56%, 15%, 14% and 15% of all requests respectively

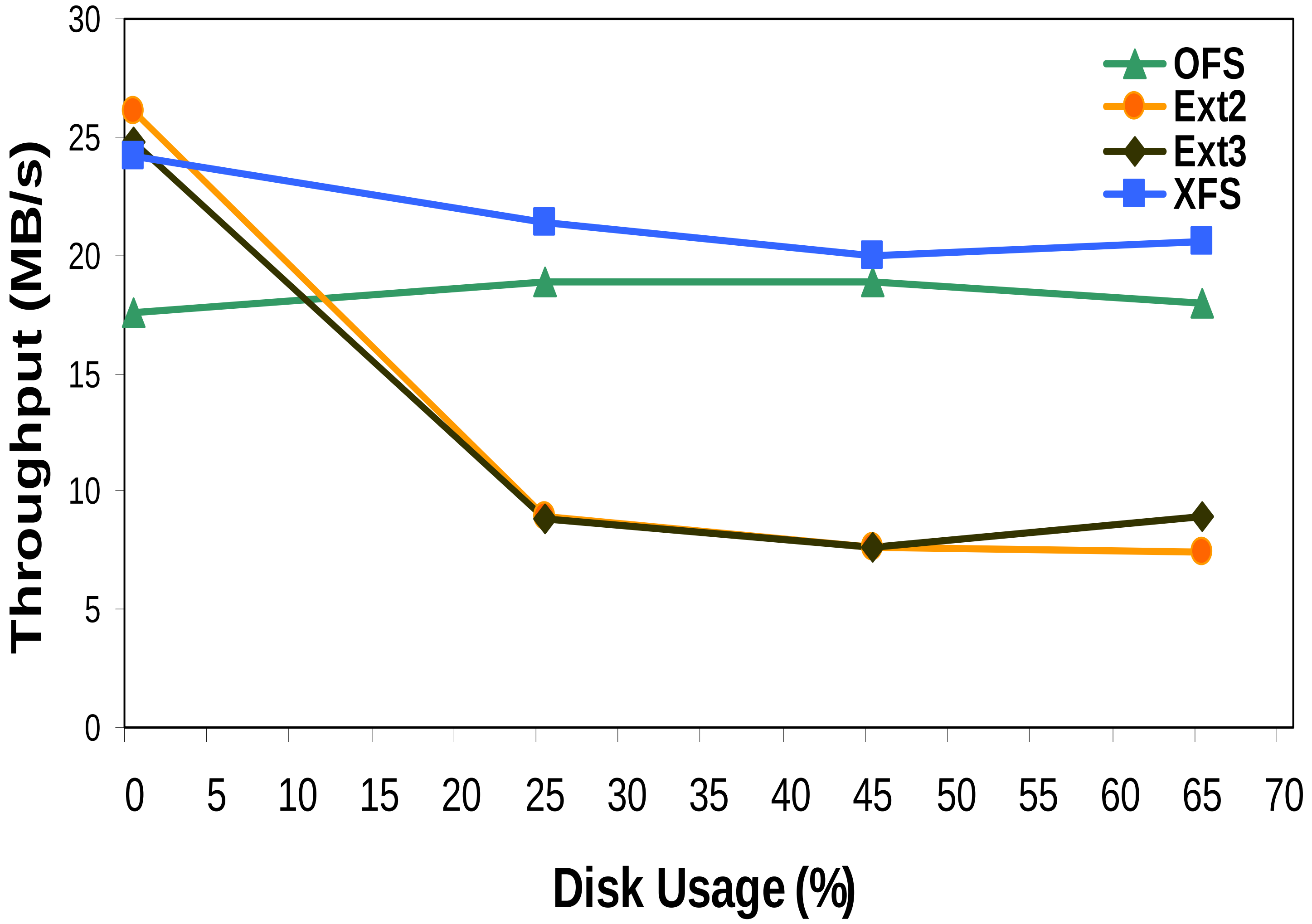
Performance Evaluation – File System Aging

- Make the results of the file system benchmarking more realistic
- Our aging workload
  - Sequence of write and delete requests
  - Write/delete ratio is dynamically adjusted based on disk usage
  - 80% of all objects are Large objects (512KB)
  - Small objects are uniformly distributed between 1KB and 512 KB
  - Delete requests are randomly generated from the current objects on disk

Benchmark Results – Write



Benchmark Results – Read



Conclusions

- Object workload characterization
  - Large objects dominate
  - Weak inter-object locality
- OBFS design and implementation
  - Variably-sized blocks
  - Region structure
  - Hash table for object naming space management
  - Collocated objects and their metadata
- Performance
  - Much better than Ext2/3
  - Comparable to XFS with 1/25 the code

Acknowledgements

- This research was supported by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory.

- We are also grateful to our sponsors: National Science Foundation, USENIX Association, Hewlett Packard Laboratories, IBM Research, Intel Corporation, Microsoft Research, ONStar, Overland Storage, and Veritas.

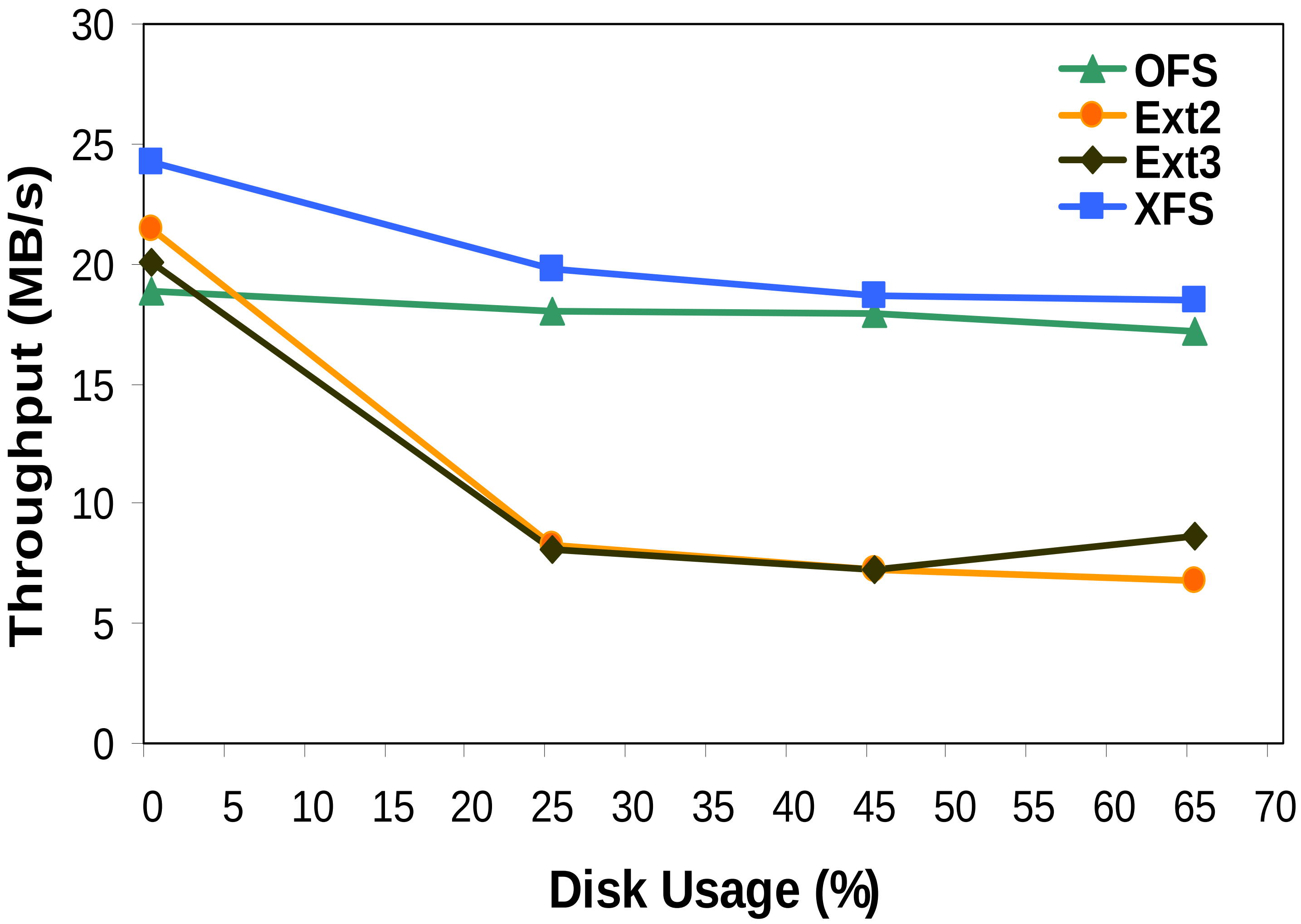
Thank You!

- More information
  - http://ssrc.cse.ucsc.edu
  - http://ssrc.cse.ucsc.edu/obsd.shtml
  - http://www.cse.ucsc.edu/~cyclonew
- Questions?

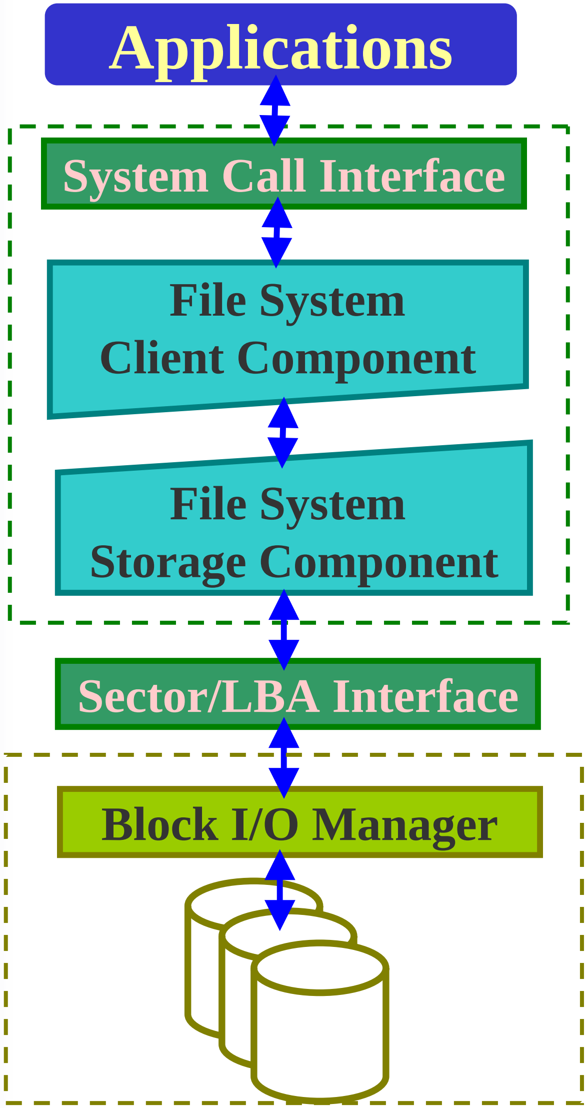
On-Going Work

- Object workload characterization
  - Parallel file workload collection
  - More general workload analysis
  - Policies study
    - File-object mapping
    - Object placement
    - Replication
    - Client-side cache management
  - Simulation approach

Benchmark Results – Overall

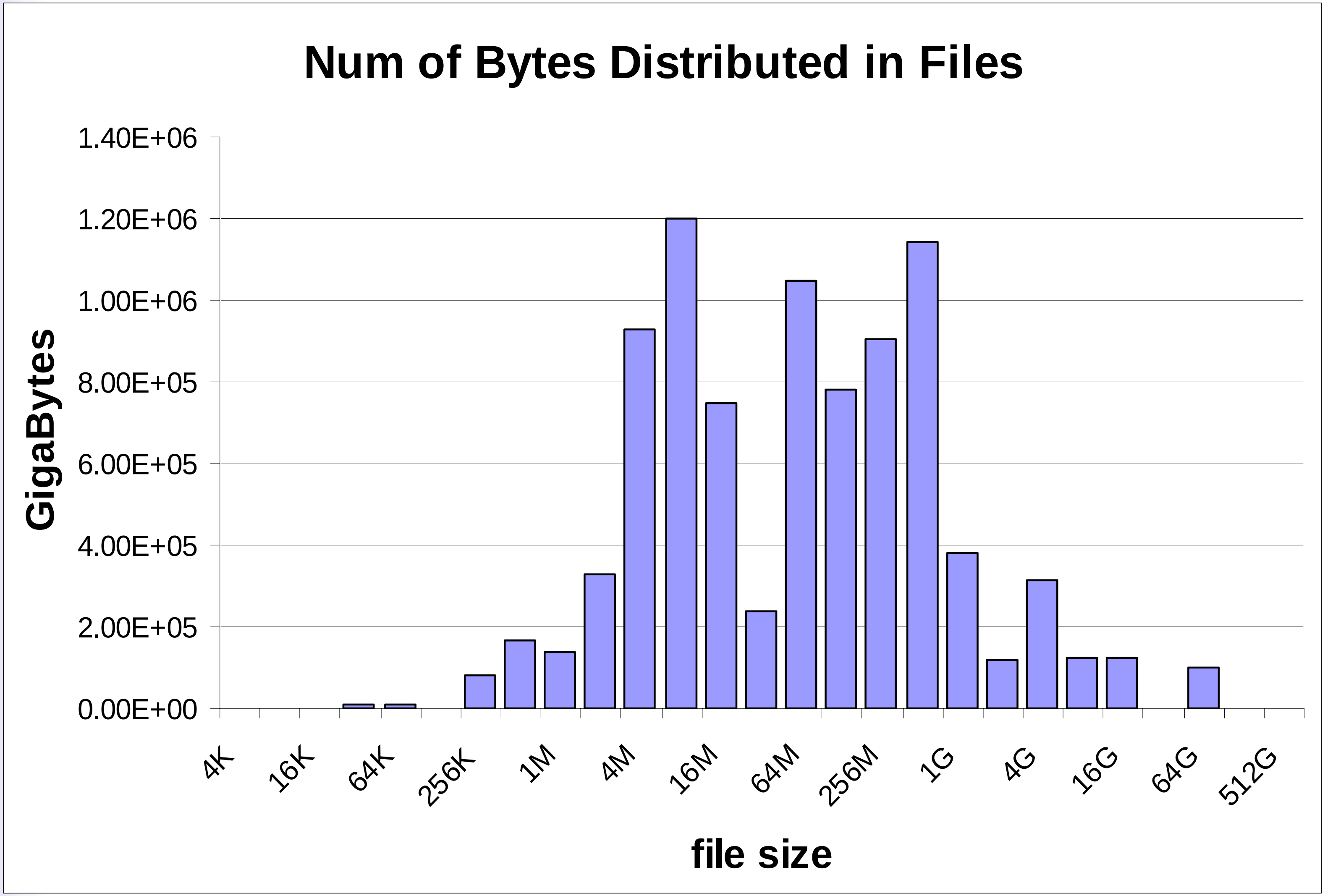


Traditional Storage Model



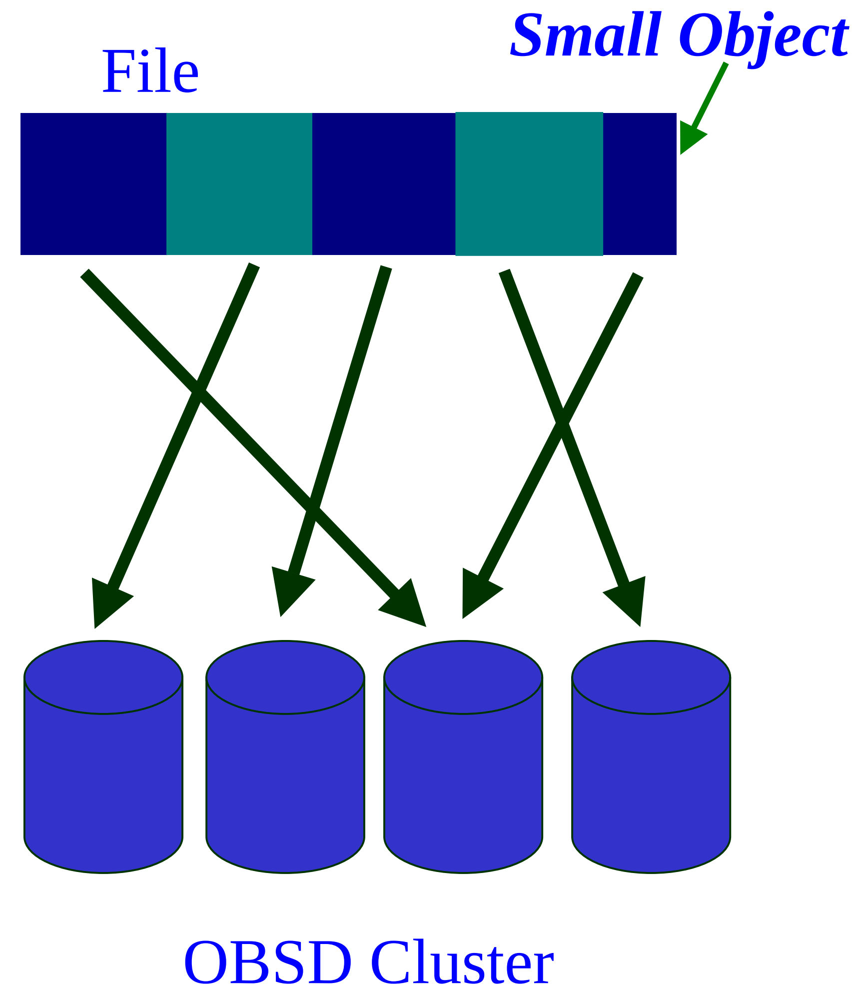
- File system functionality
  - Directory hierarchy management
  - Access control
  - Protection
  - Data allocation
  - Request Scheduling
  - "Data Switch"
- Sector/LBA interface
  - Low-level knowledge of disk characteristics and organization used in file system

Workload Characteristics



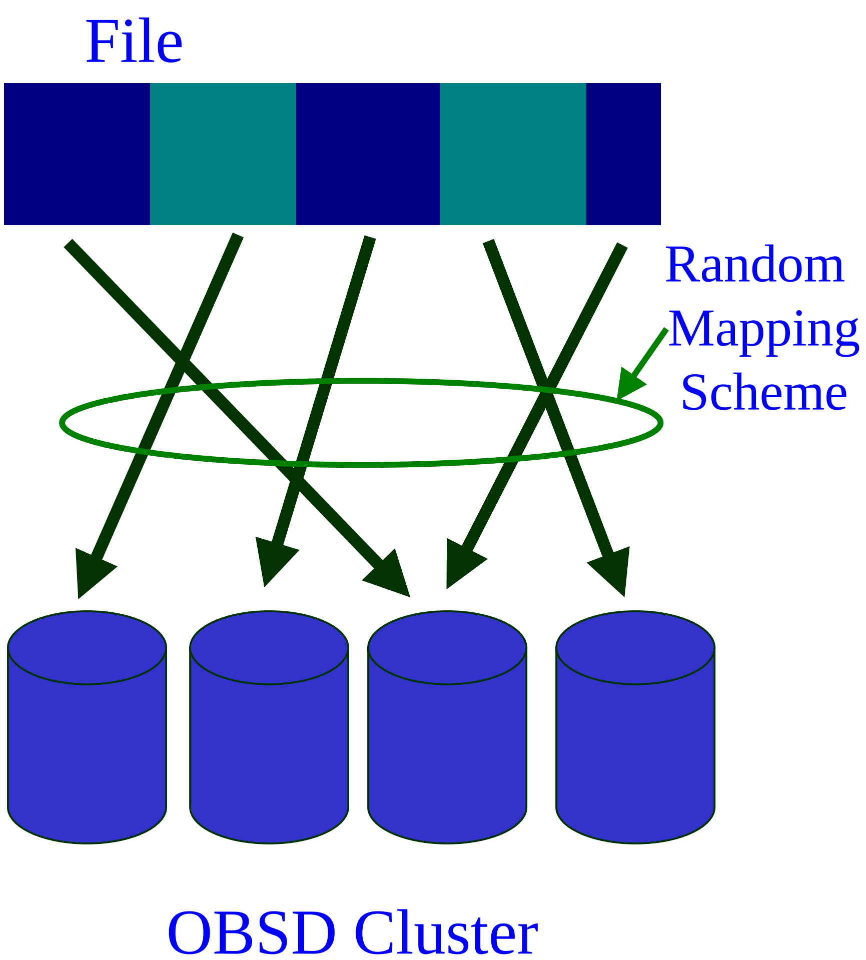
Object Workload

- Files are striped into objects (system-level design decision)
  - ***large objects***: the stripe-unit size objects
  - ***small objects***: all other objects
  - Objects are evenly distributed across the cluster of OBSDs
- Large objects dominate
- Weak inter-object locality



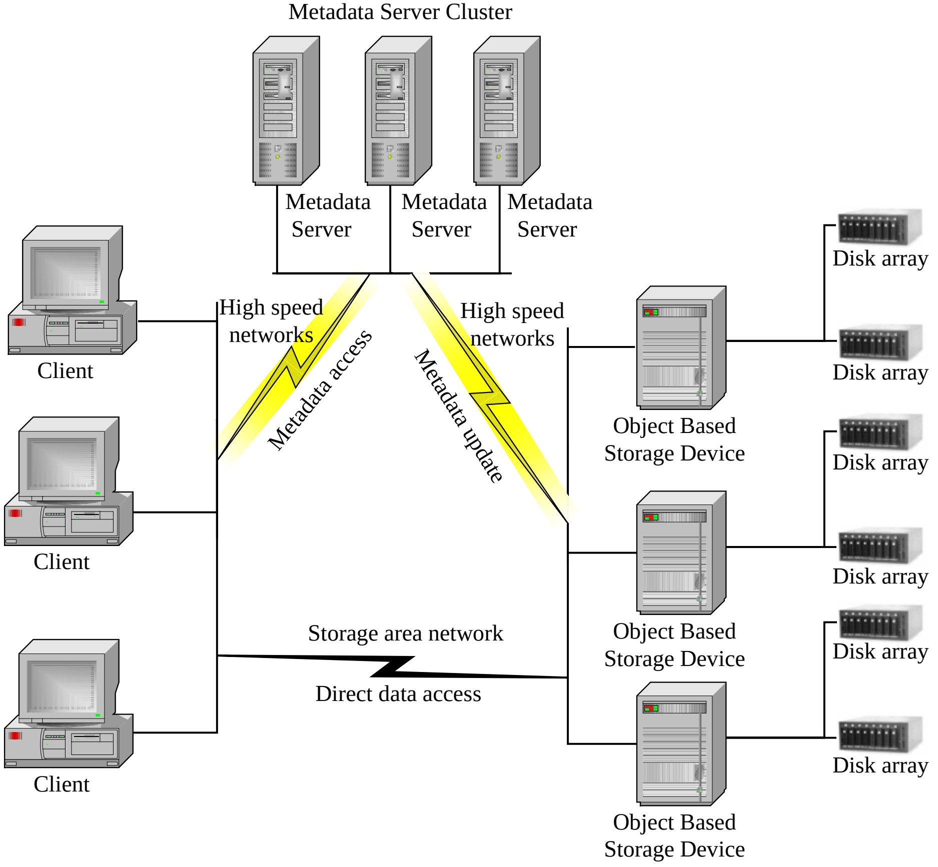
Object Workload

- Files are striped into objects (system-level design decision)
  - ***large objects***: the stripe-unit size objects
  - ***small objects***: all other objects
  - Objects are evenly distributed across the cluster of OBSDs
- Large objects dominate
- Weak inter-object locality



Large Scale Distributed Object-Based Storage System (LSDOSS)

- Currently being developed at Storage System Research Center (SSRC), UCSC
- Aim to scientific computing environment
- Build on Object-Based Storage Devices (OBSDs)
- Expect to deliver 100 GB/s throughput and 2 PB capacity



Current Status

- Object workload characterization
  - Preliminary analysis based on LLNL data
  - Random object placement policy (RJ's work)
  - Fixed-size data striping scheme
- Object interface
  - Identify the basic command sets
    - Read/partial read, write/partial write, delete
- OBFS design and implementation
  - Optimize the data layout and object mapping scheme based on the object workload analysis
  - User-level implementation in Linux

Hypothesis

- Significantly better OBSD system performance can be obtained through OFSes specifically designed for object workloads than can be obtained with general-purpose file systems
- Rationale
  - Workload is different: sizes, locality, access patterns
  - Interfaces are different: object-specific operations, lack of file metadata
  - Requirements are different: metadata, permissions, locking, reliability, caching

File Systems For Object-Based Storage Devices

Feng Wang
Scott Brandt
Ethan Miller
Darrell Long

Storage Systems Research Center
University of California, Santa Cruz