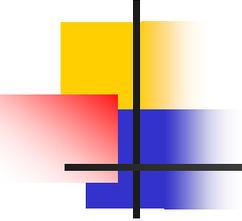


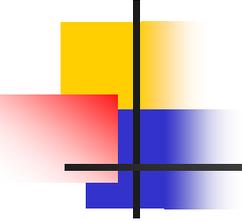
Object Caching for MPI-IO

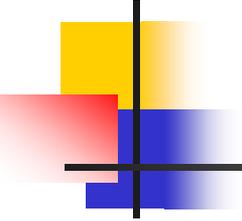
Phillip M. Dickens and Jeremy Logan
Department of Computer Science
University of Maine

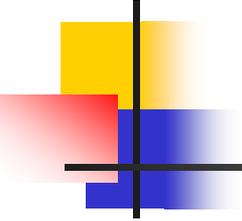


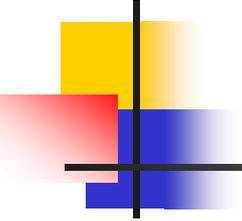
Project Overview

- A fundamental challenge in providing scalable, parallel I/O is that data-intensive applications, do not, in general, access their data in a manner consistent with how data is stored on disk.
 - Access patterns often lead to a large number of small I/O requests.
 - Well-known problem, techniques such as two-phase I/O, data sieving, Data Type I/O have been developed significantly improving I/O performance.

- 
-
- Problem is not with I/O access patterns, but rather with the legacy view of a file as a linear sequence of bytes (block-based file).
 - Applications do not access their data consistent with this file data model.
 - More accurately described as an object model.
 - Each object represents a file region in which the process operates.
 - File data is viewed as a collection of (perhaps) non-contiguous objects.

- 
-
- Object file: File that stores data as a contiguous set of objects rather than as a linear sequence of bytes.
 - More powerful file model because it “encodes” the access patterns of the application.
 - Ignoring contention (for the moment), if a process has all of its objects stored locally, and objects are stored contiguously on disk then:
 - Process can move its data to and from the file system in a single I/O operation.

- 
-
- Merging the power and flexibility of the parallel I/O interface (and implementations) with a more powerful object file model.
 - However, MPI does not operate in terms of objects, so we are developing an object caching system to provide an interface between MPI applications and object files.
 - Completely transparent to the application.



Objects

- Objects are based on MPI file views:
 - Maps relationship between regions of a file that a process will access and the way regions laid out on disk.
 - Process cannot “see” or access any regions outside of file view
 - Logically maps a contiguous view window onto the (perhaps) non-contiguous file regions in which the process will operate.

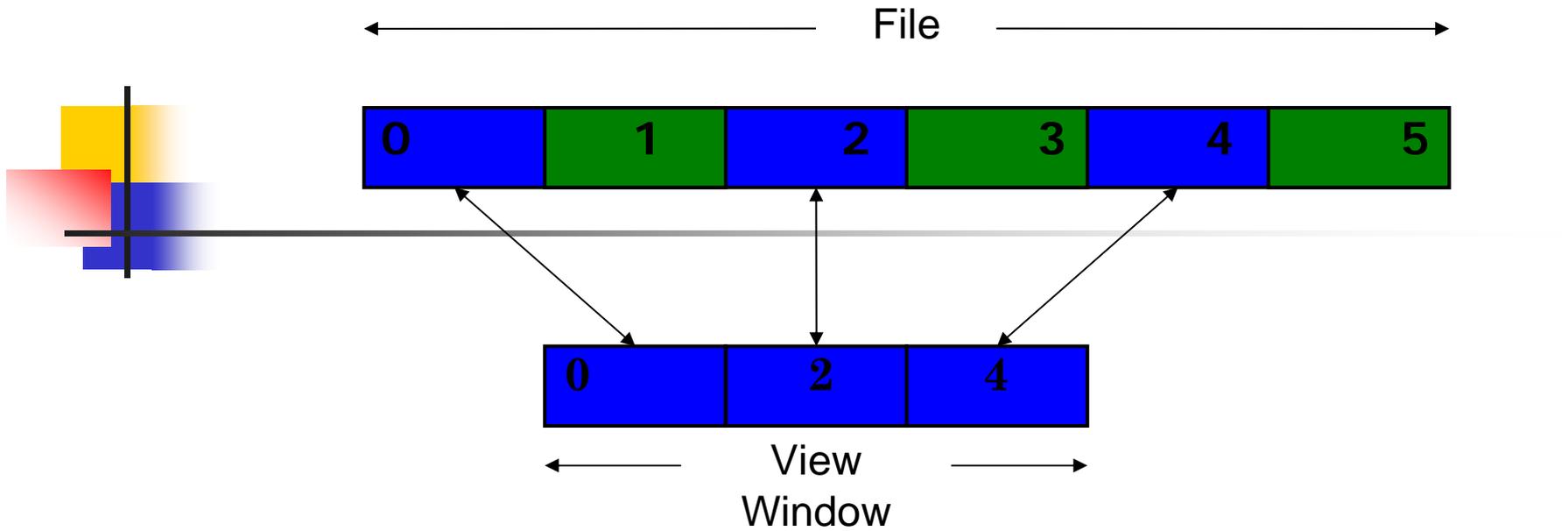
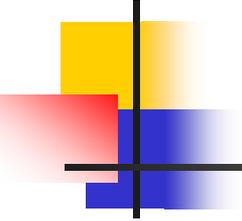


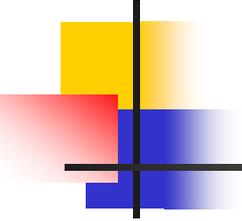
Figure shows file access patterns of two processes.

Store the contiguous “objects” on the process that will use them.



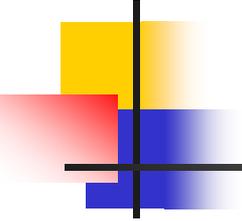
Object Creation

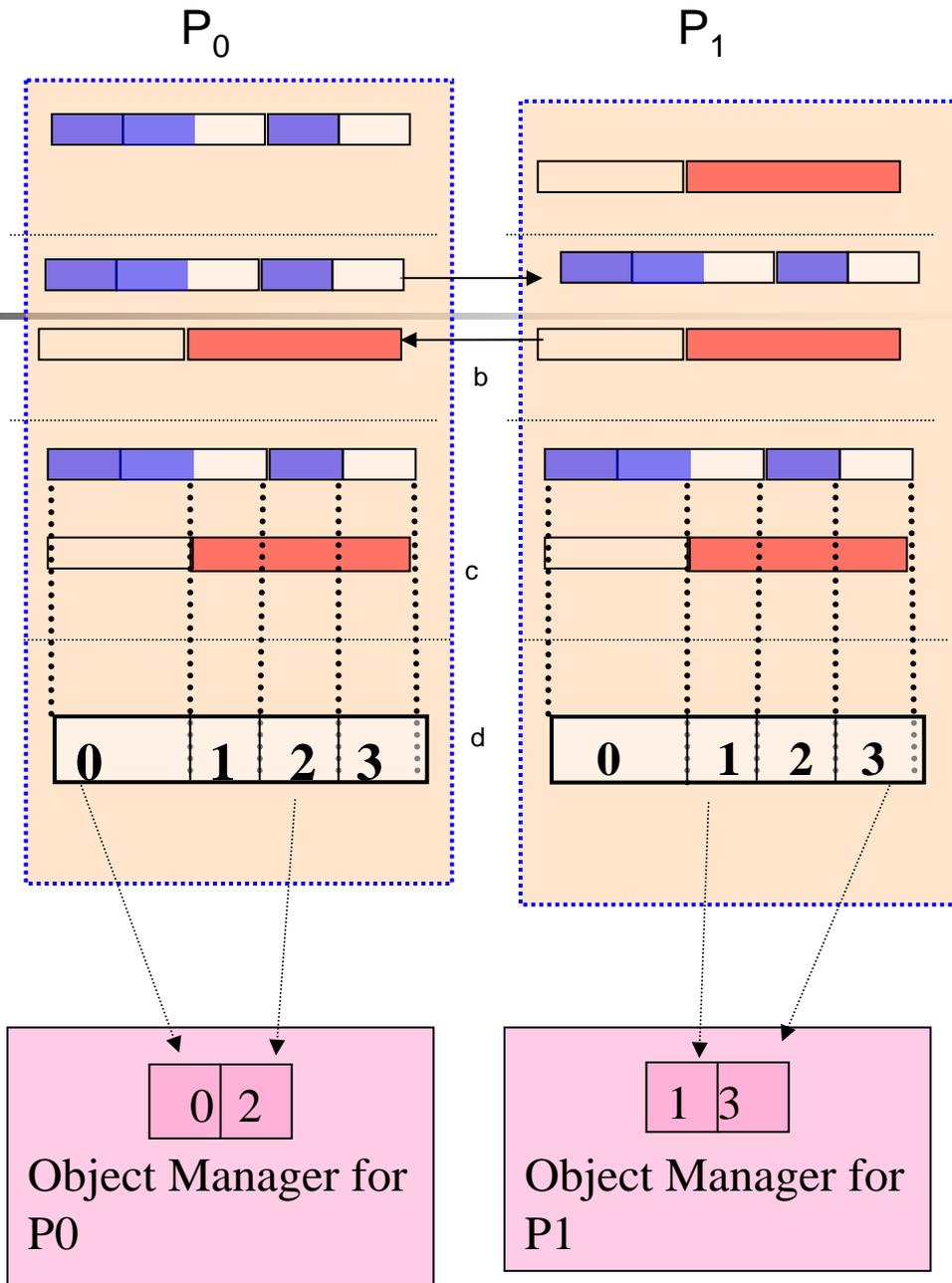
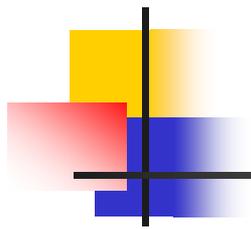
- Intersections of MPI file views.
 - Object created for each intersection.
- Objects created in this way encode all known information about processes access patterns.
- Identifies all file regions within which conflicting accesses are possible (shared objects) and all regions for which there can be no contention (private objects).
- This information can be utilized by runtime system to significantly enhance performance.

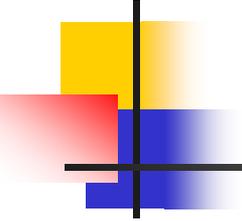


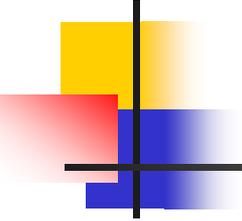
Cache Runtime System

- All processes that share a file participate in the object cache for that file.
 - Utilize process memory and any local disk space.
- Local object manager for each participating process.
- Once objects are created, distributed among local managers based on a cost model of assigning a given object to a given manager.
- Local manager controls meta-data and locking for all objects it controls.

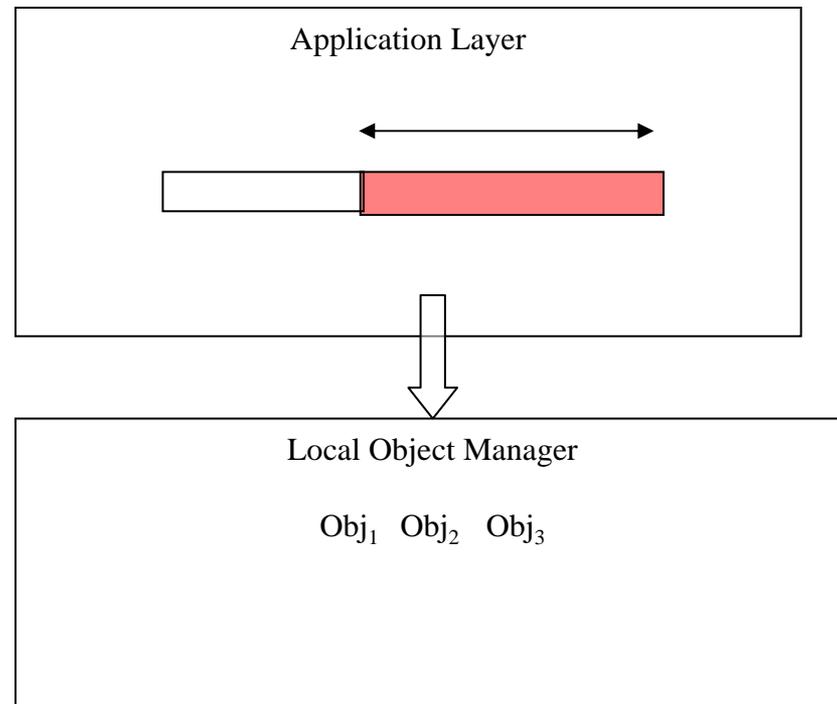
- 
-
- Once objects are created all subsequent I/O operations are performed in the cache (except in the case of `sync()` or `close()` operation).



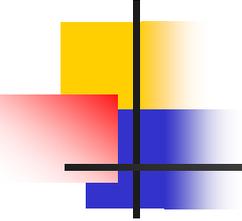
- 
-
- Once created, can determine reverse access set which is list of all processes that have access to one of their objects.
 - Distinction between shared and private objects has two important ramifications:
 - Only shared objects must be locked, and represents the minimum (known) overlap of shared file region.
 - Provides maximum possible concurrency for data access.
 - Simplify and increase performance of locking system

- 
-
- Essentially, each object manager acts as a centralized lock manager for those processes.
 - Contention for write locks is limited to subset of processes that can access objects.
 - Rather than a complex distributed lock manager (for the entire file), have a distributed set of centralized lock managers operating in parallel.

Dynamic Translation Between File Data Models

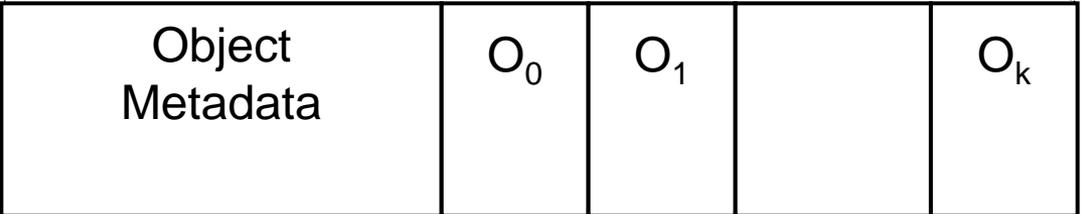
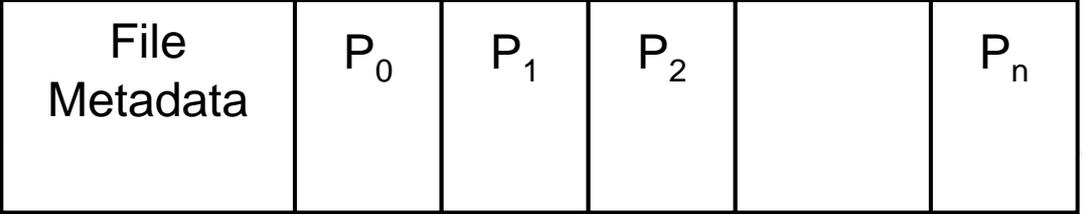


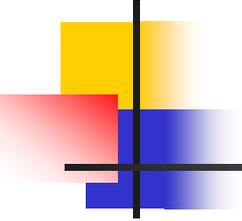
Request for
(shared) Object 2.



Object Files

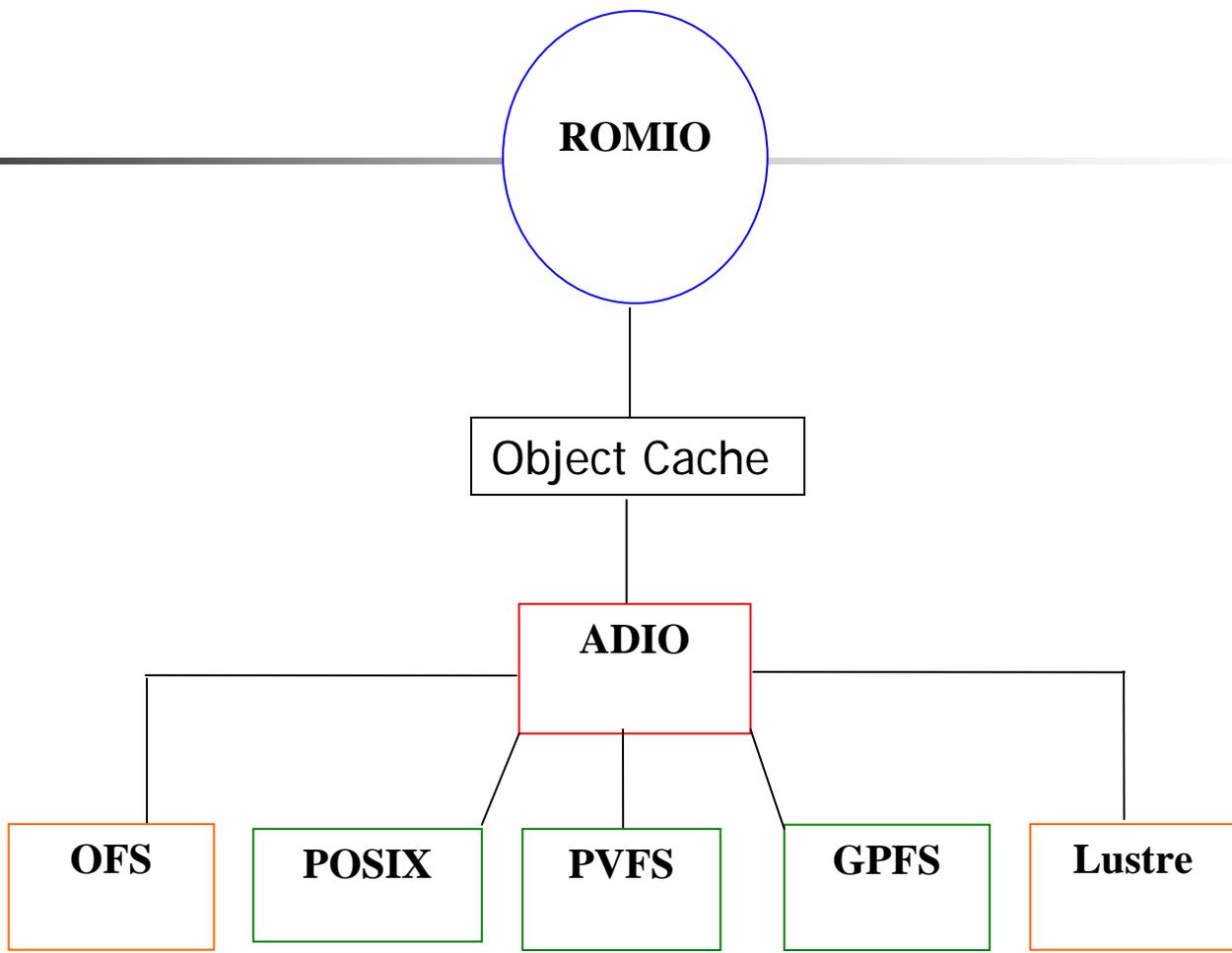
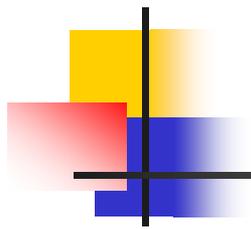
- Objects can be written to disk as either a block-based file or an object file.
- If written as an object file, meta-data needed to translate between file models.
- If written as block-based object managers perform operation similar to two-phase I/O to put objects back into linear sequence of bytes.
- Advantage: Each process can write all of its objects to disk in parallel.

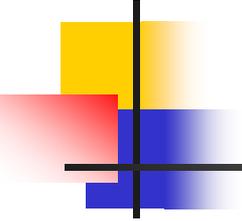




Integration with MPI

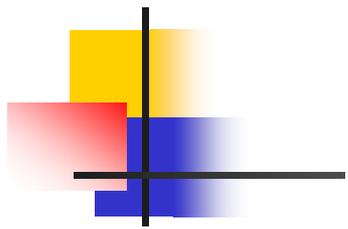
- Have integrated prototype caching system in MPICH2 developed at Argonne National Laboratory.
- Whole concept of object files can be completely transparent to the application.
- Object files are treated as simply another file system.
 - ADIO driver for object files.



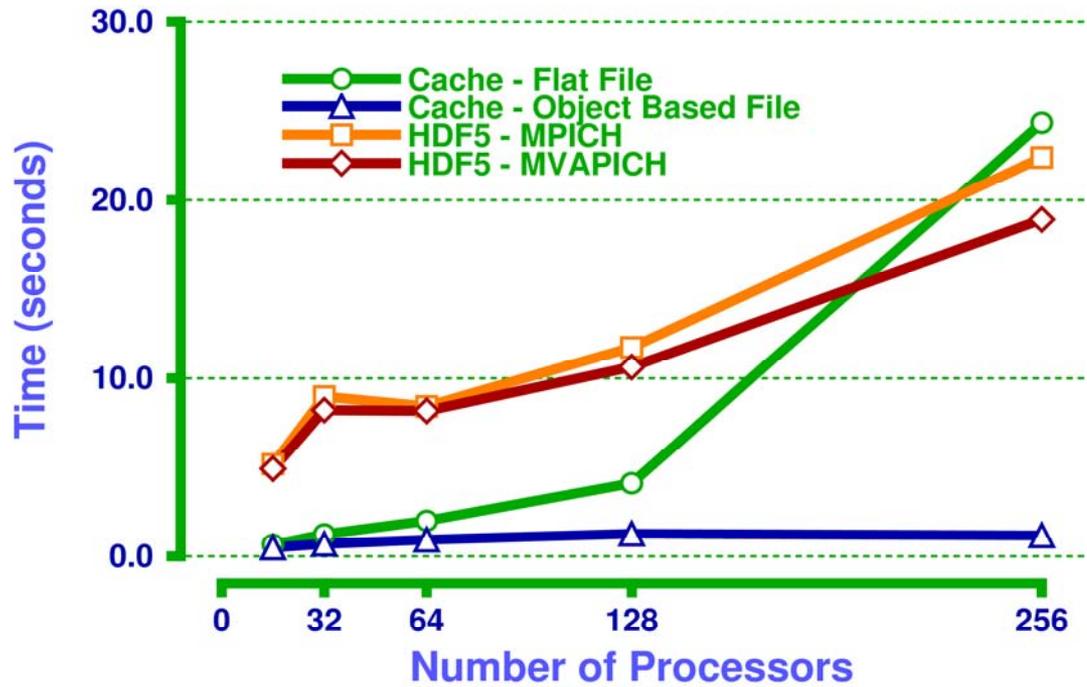


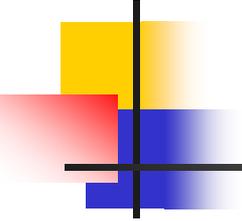
Experimental Evaluation

- Preliminary performance results using FLASH I/O benchmark.
- Simulated the I/O because we are not yet able to interface with HDF-5 or parallel NetCDF (work in progress).
 - Used the same access patterns and created same size file.
- Experiments performed on Ranch (TACC) on Lustre file system (Scratch) with 50 OSSs and 300 OSTs.
 - Compared our simulated approach using the object cache, and version of FLASH from ANL uses HDF5.
 - Compared using both object files and block-based files.



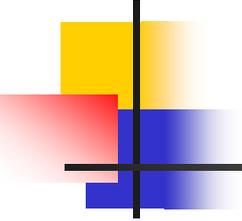
FLASH I/O Performance Checkpoint Write





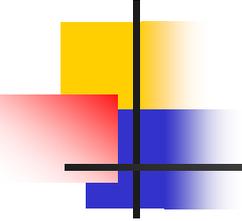
Issues

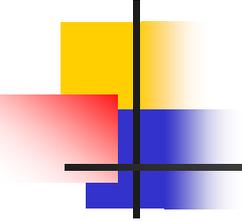
- Benefits obtained from this approach are a function of the quality of the file views provided by application.
- In the best case many advantages to this approach.
 - Simplified locking
 - Locality of cache objects to processes
 - Can buffer multiple
 - Contention for locks is reduced
 - Fewer file system accesses
 - Buffering can span multiple I/O calls.



Issues

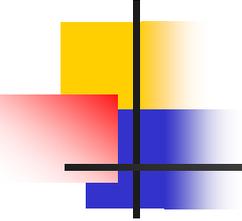
- In the “worst” case revert to a standard block-based caching system.
- Looking for other means of exposing file access patterns.

- 
-
- Currently investigating use of XML files.
 - Other possibilities include
 - Graphical tool to input object structure.
 - Object learning.
 - Develop information on user's behalf.
 - If application frequently changes file views then:
 - Try to convince them to change their approach.
 - Do not use object cache.



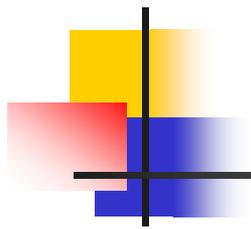
Conclusions

- Developing new model for file data based on objects.
 - Encode all known file access patterns.
 - Natural mechanism to capture information about shared/private regions of a file.
 - Can be used by runtime system to simplify and increase performance of the locking system (under construction).
 - Provides maximum concurrency possible based on current knowledge.
 - Can write object files in a single I/O operation in parallel.



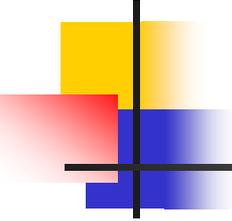
Conclusions (continued)

- We have integrated prototype caching system with MPICH2.
- Saving data as object files can significantly improve performance.
- Looking forward to experimenting with MPI Atomic mode.
- Investigate performance with very large number of small, unaligned objects.
 - Bread and butter of the object caching system.



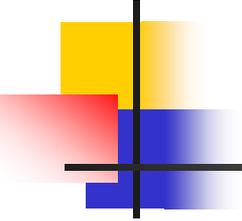
Thank you!

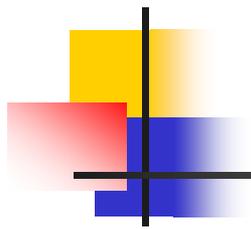
Questions?



Understanding the Performance of MPI-IO in Lustre File System Environment

- Important problem, reason for which is not well understood.
- We believe the fundamental problem is that assumptions underpinning most important parallel I/O optimizations do not hold in a Lustre environment.
- Most widely held assumption is that parallel I/O performance is maximized when aggregator processes perform large, contiguous I/O operations.

- 
-
- Problem is that this can easily cause an all-to-all communication pattern that creates contention at the network layer, the locking layer, OSS (OST) level.
 - In many cases, the overhead of such contention completely dominates performance advantage of performing fewer I/O operations.

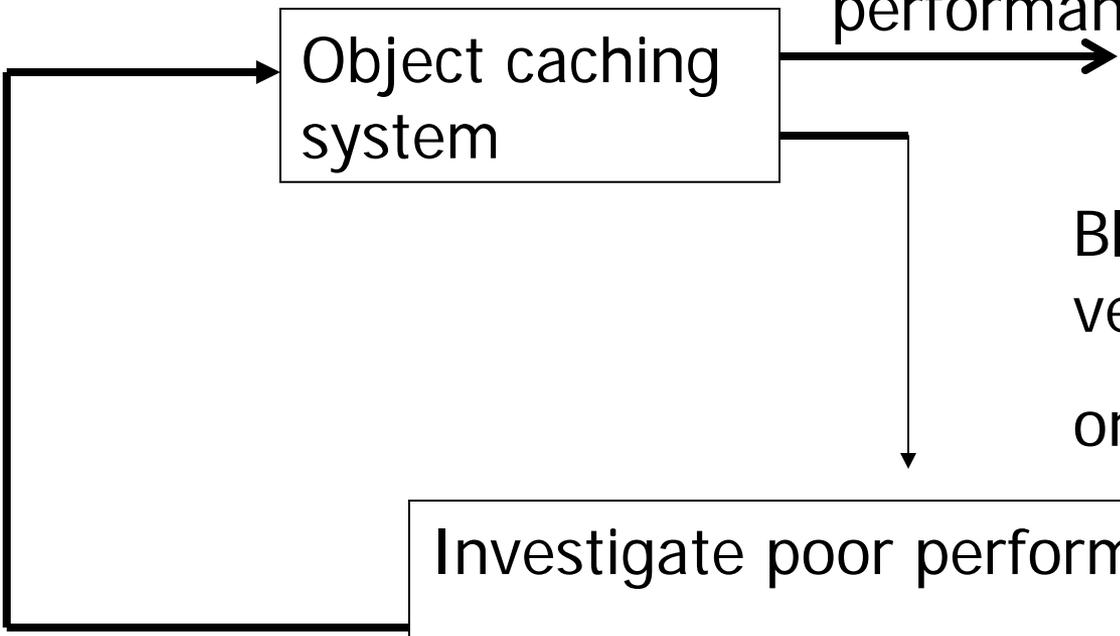


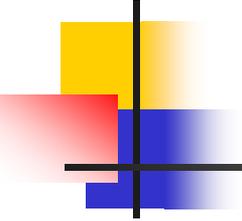
Object Files with Excellent performance

Object caching system

Block-based files with very poor performance on Lustre

Investigate poor performance.
Implement new algorithms more aligned with Lustre's storage model



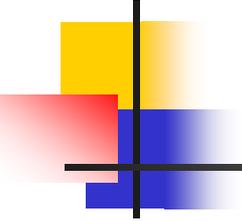


- Hypothesis:

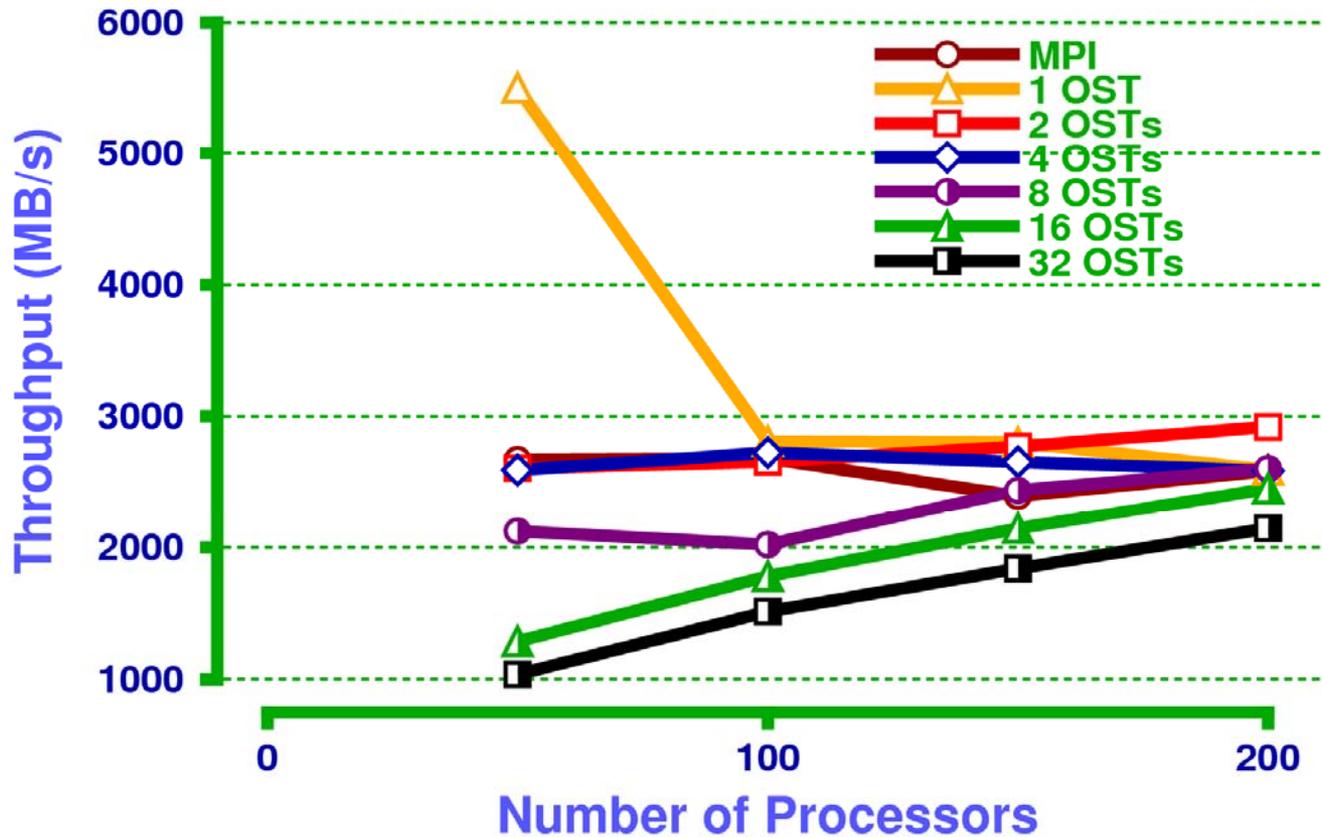
- Performance dominated by cost of communication pattern.

- Experiment:

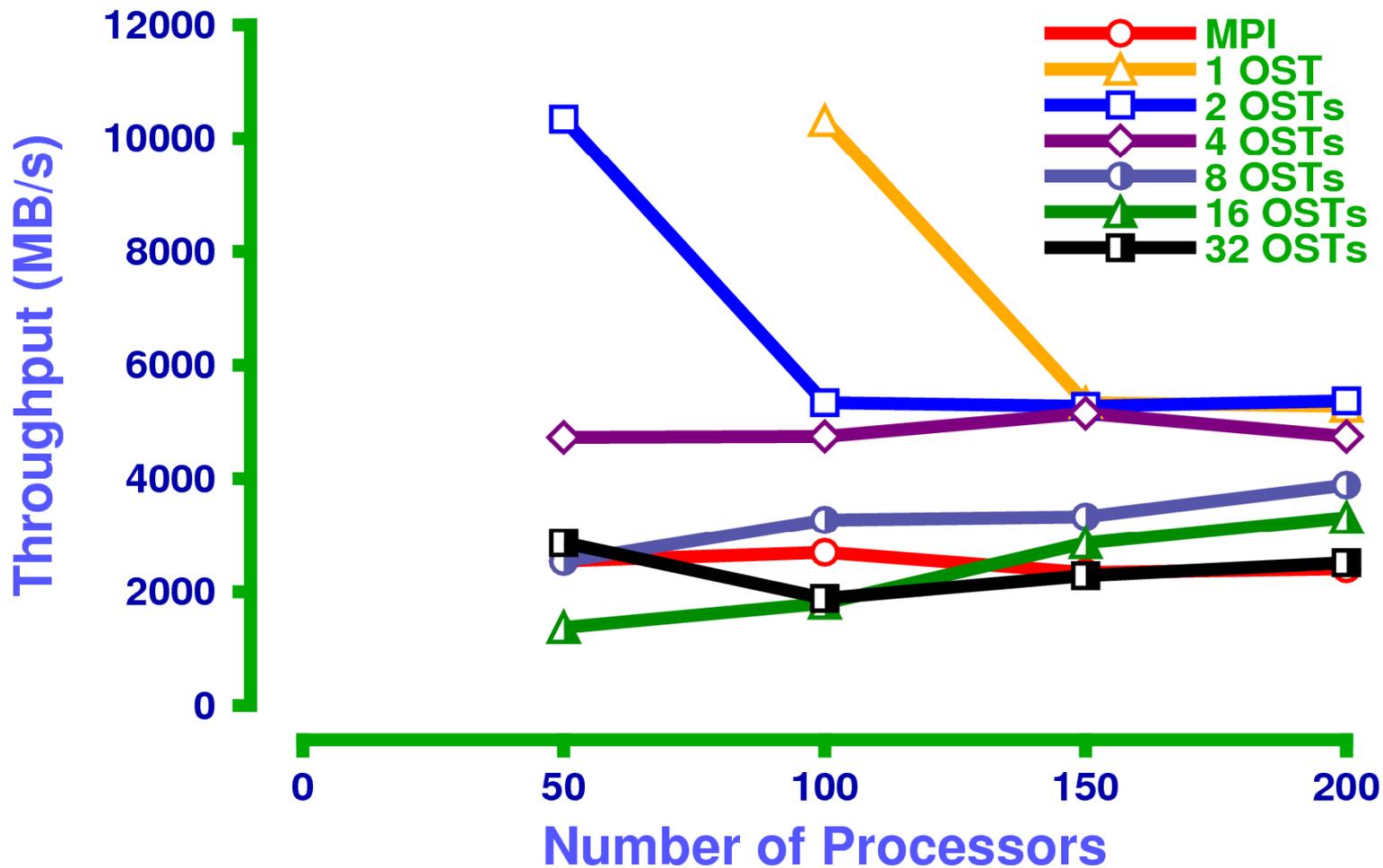
- Control number of OSTs (or OSSs) with which aggregator processes communicate.
- Can do this by controlling the block size for which each process is responsible.
- Tradeoff: Decreasing number of OSTs with which process communicates increases the number of separate I/O requests.

- 
-
- Identify different approaches by number of OSTs with which aggregator processes communicate.
 - Executed on large Lustre file system on Ranger.
 - 50 OSSs and 300 OSTs.
 - Writing 50 Gigabyte file.
 - Varied number of OSTs and number of aggregator processes.
 - Compared with native MPI-IO implementation.

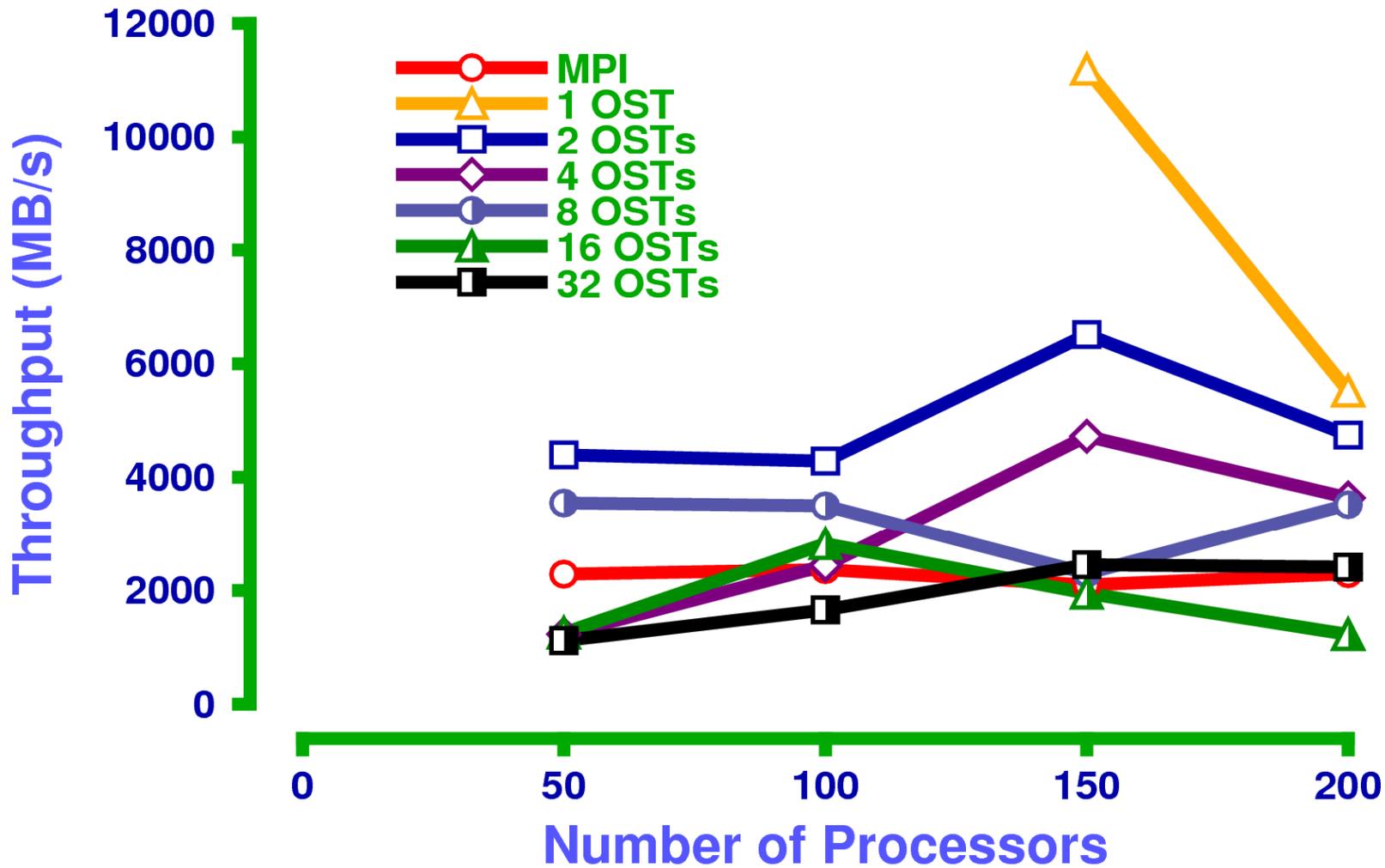
Average Throughput vs. Processor Count Ranger - 50 OSTs



Average Throughput vs. Processor Count Ranger - 100 OSTs



Average Throughput vs. Processor Count Ranger - 150 OSTs



Average Throughput vs. Processor Count Big Red

