# Optimizing Local File Accesses for FUSE-Based Distributed Storage


Shun Ishiguro[1], Jun Murakami[1],
Yoshihiro Oyama[1,3], Osamu Tatebe[2,3]

1. The University of Electro-Communications, Japan

2. University of Tsukuba, Japan

3. Japan Science and Technology Agency

# Background

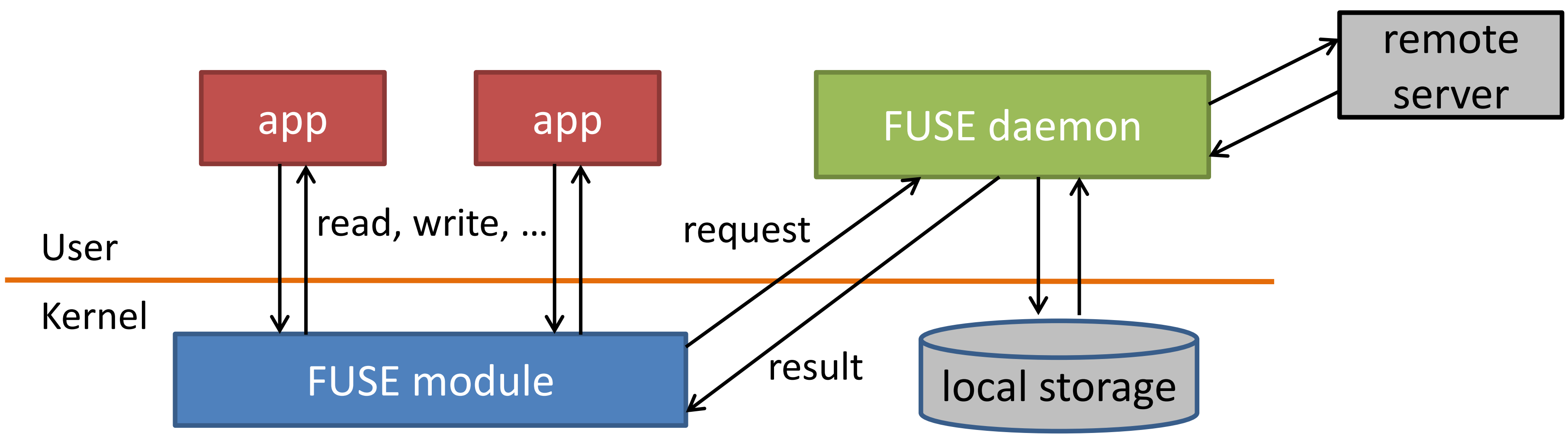
- Highly scalable distributed file systems are a key component of data-intensive science
  - Low latency, high capacity, and high scalability of storage are crucial to application performance
  - Many distributed file systems can manage numerous large files by federating multiple storage
    - Ex.: Lustre, GlusterFS, Ceph, Gfarm
- Several systems allow users to mount file systems on UNIX clients by using FUSE

# FUSE

- A framework for implementing user-level file systems
  - Once mounted, the distributed file systems can be accessed with standard system calls such as open
- Drawback: considerable overhead on I/O
  - It degrades overall performance of data-intensive HPC applications
  - Large overhead is reported in multiple papers
    - [Bent et al., SC09], [Narayan et al., Linux OLS ’10], [Rajgarhia et al., ’10]
    - Ex.: Bent et al. reported 20% overhead on the bandwidth of file systems

# Structure of FUSE

- Composed of a kernel module and a userland daemon
- Implementing a file system $\cong$ implementing a daemon

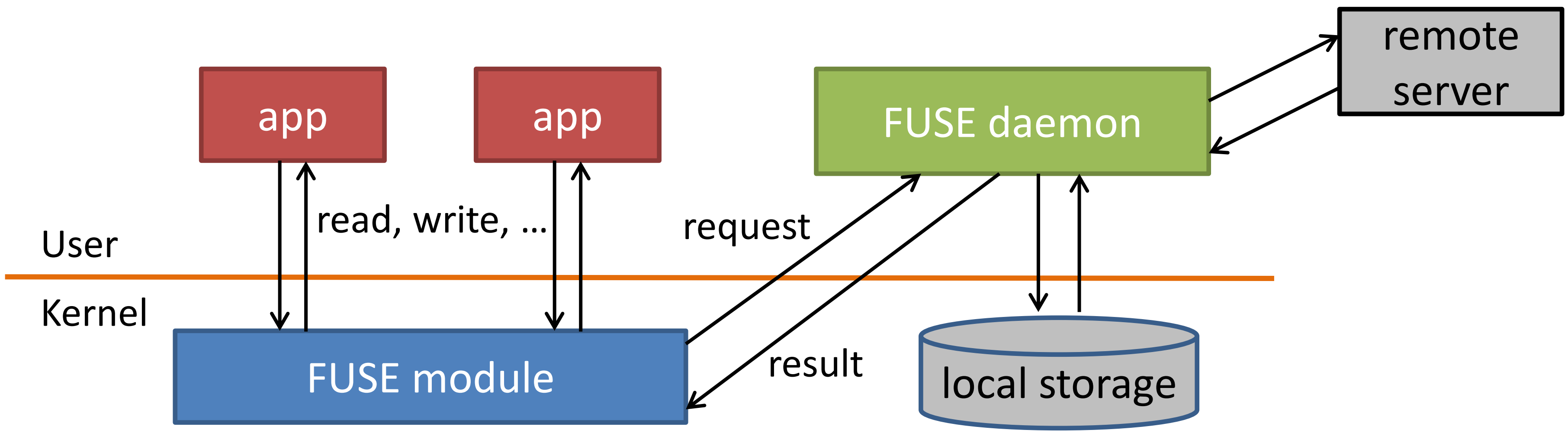


# Source of Overhead

- Context switches
  - Between processes
  - Between userland and kernel
- Redundant memory copies

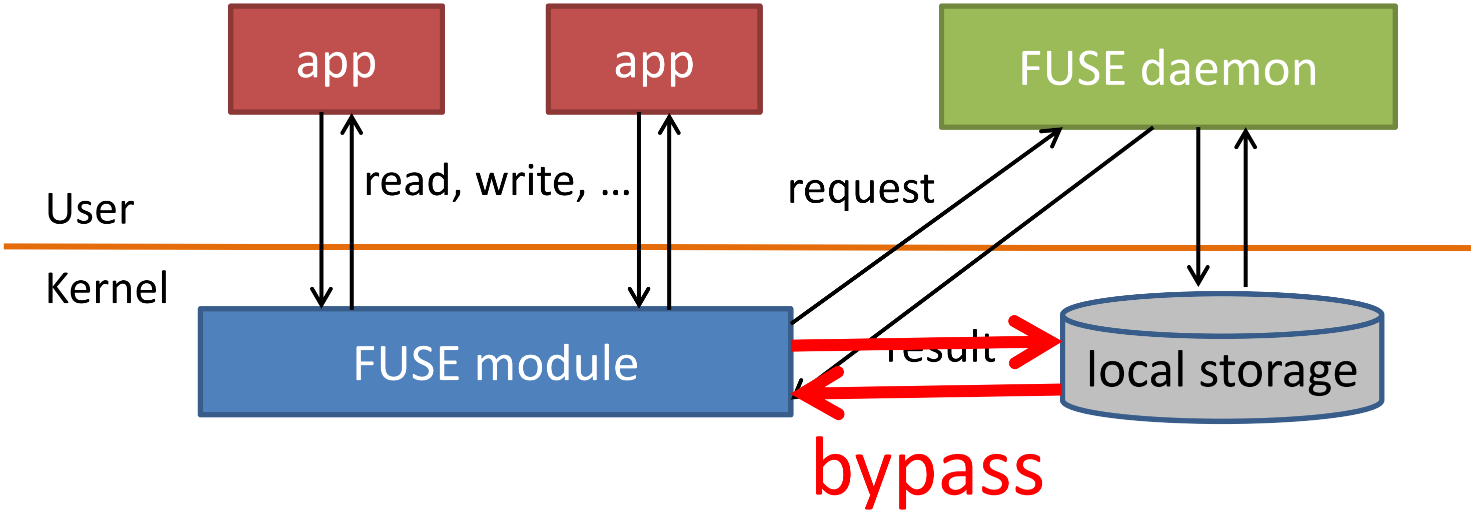
All of them occur even if the file is in local storage

Several distributed FSes such as HDFS utilize local storage



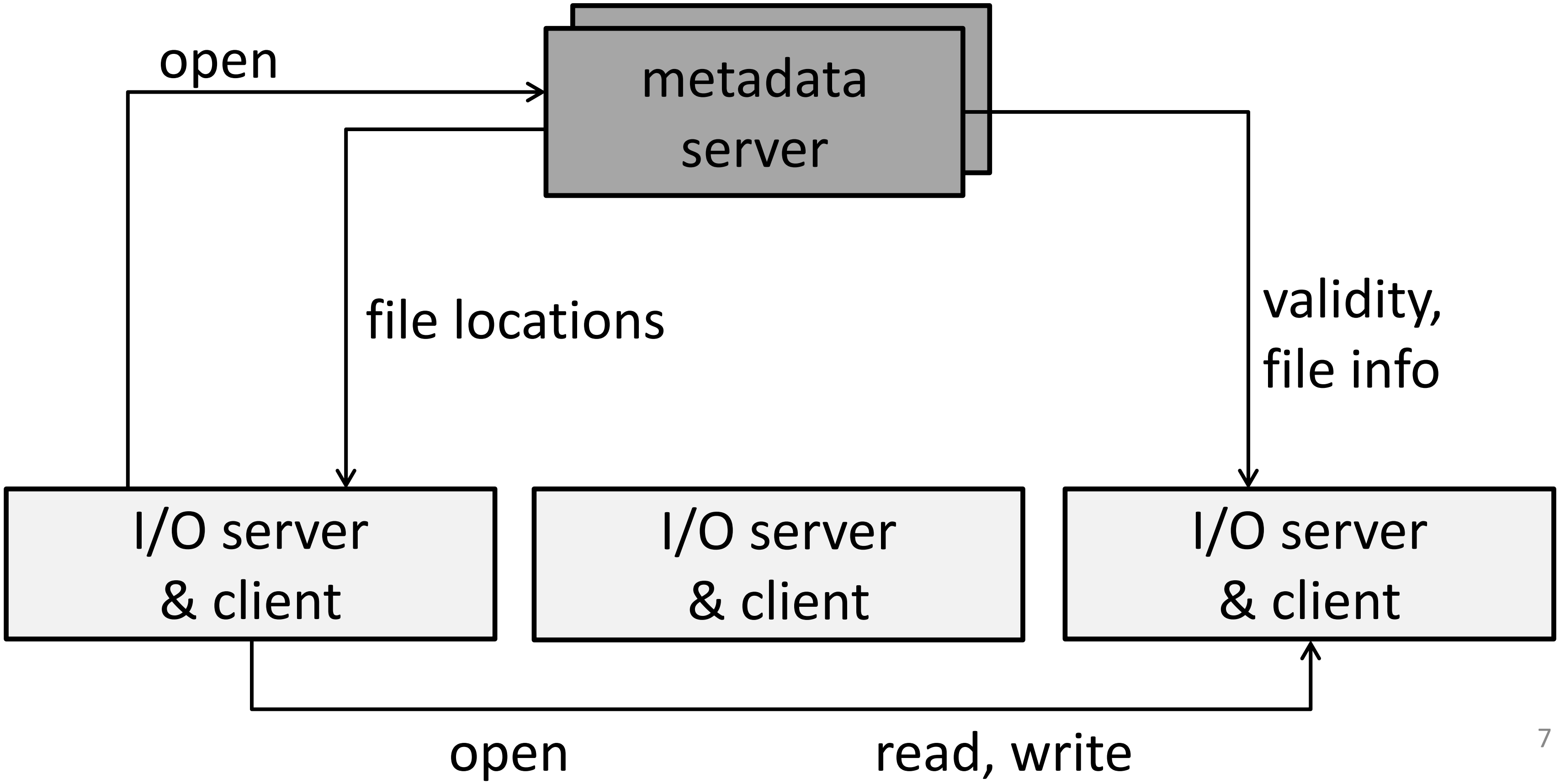
# Goal

- Propose a mechanism that allows to access local storage directly via FUSE
  - It “bypasses” redundant context switches and memory copies
  - Accesses to remote files are out of this work
- Demonstrate the effectiveness by using Gfarm distributed file system [Tatebe et al., ’10]



# Gfarm Architecture

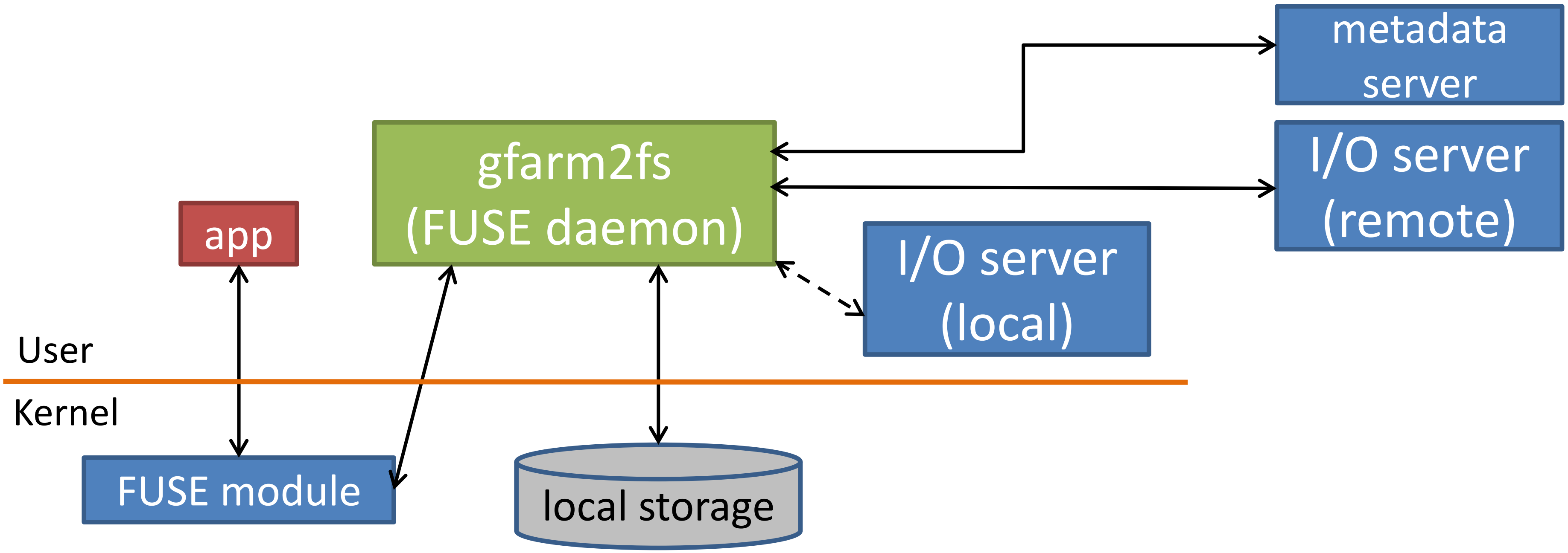
- Gfarm aggressively uses local storage
  - I/O server and client can exist in the same node



# Mounting a Gfarm File System

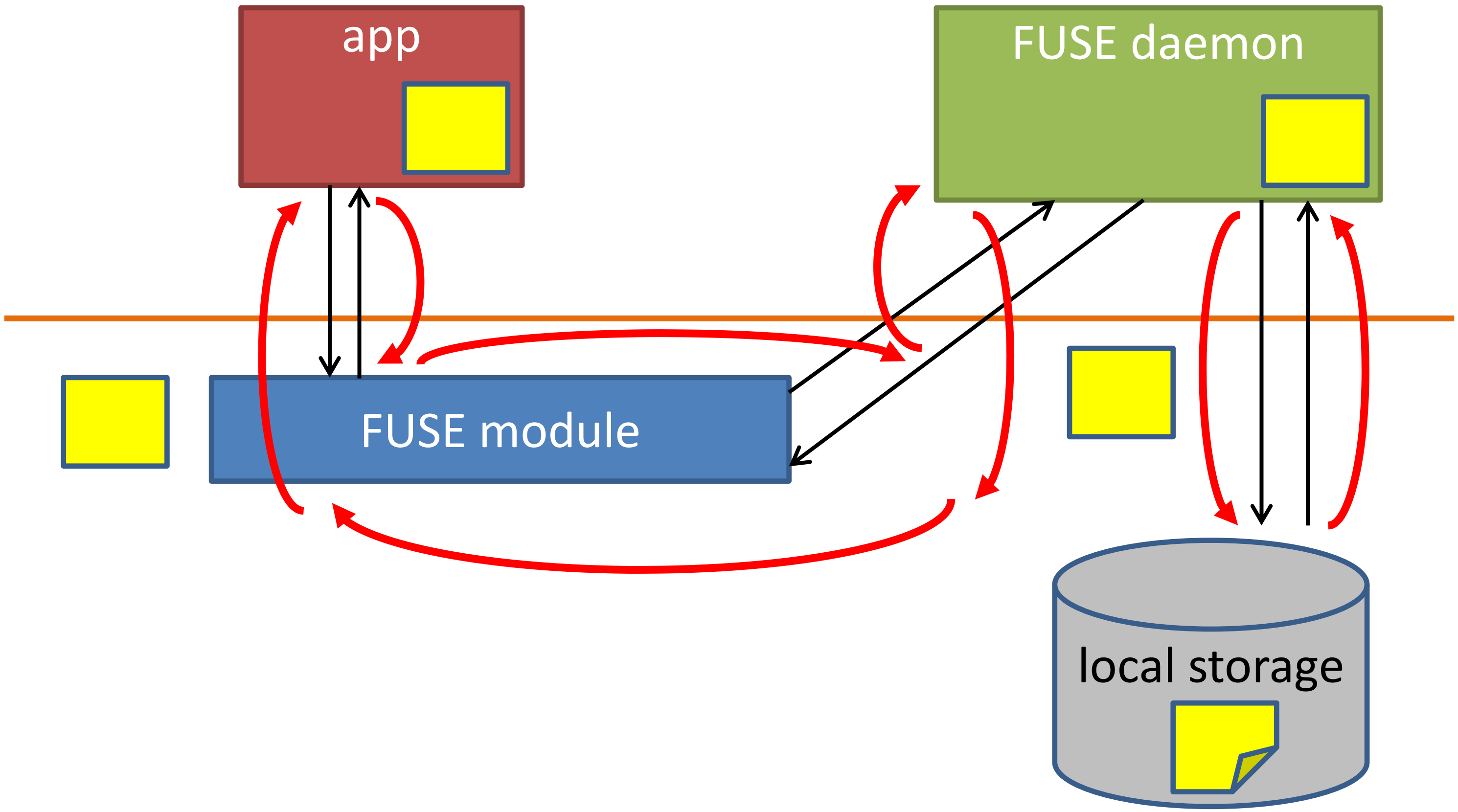
- gfarm2fs
  - Is FUSE daemon for mounting Gfarm file system
  - Communicates with metadata servers and I/O servers by invoking the API of Gfarm

# How to Access a Mounted Gfarm FS through gfarm2fs




- If the accessed file is managed by a remote I/O server, gfarm2fs receives the file from the remote I/O server
- If the accessed file is managed by the local I/O server, gfarm2fs bypasses the server
  - gfarm2fs itself executes system calls to access the file

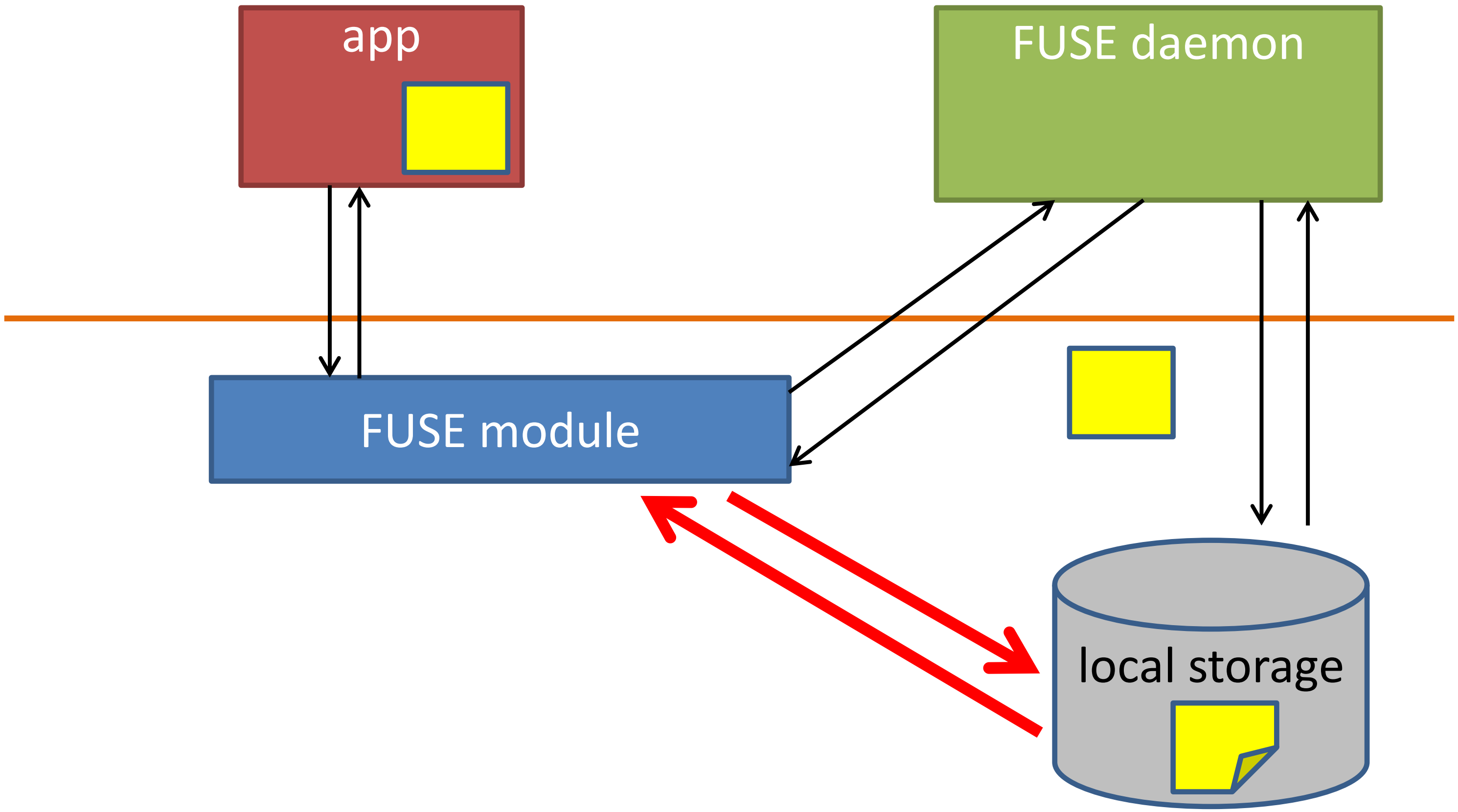
# Elaborating the Overhead of FUSE



# Proposed Mechanism

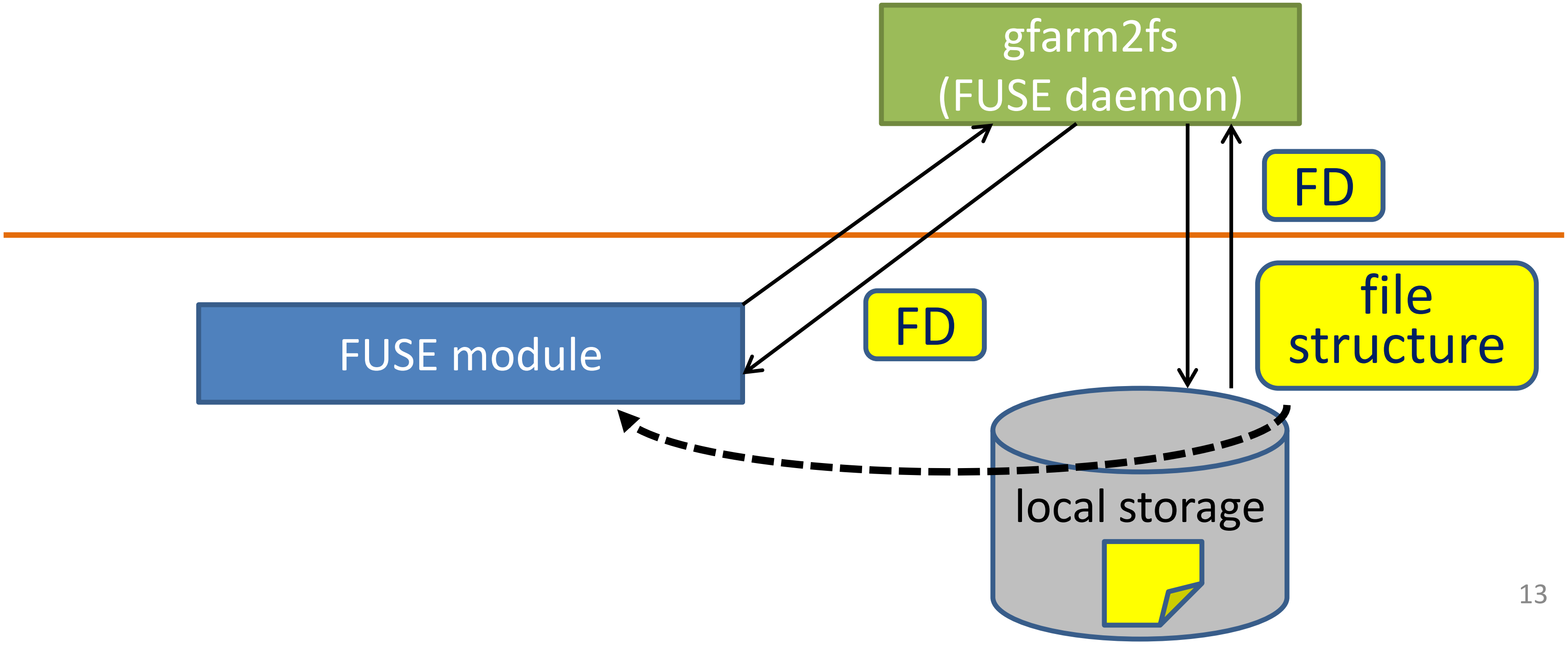
- Our modified FUSE module directly accesses local storage
  - No context switch between processes
    - Only one round-trip between kernel and user
  - Reduced memory copy
    - Page cache due to FUSE daemon is no longer created
    - Page cache is created only by local FS such as ext3
- No change in remote file accesses

# Overview of the Mechanism



# Design (1/2)

- Key idea
  - gfarm2fs passes the file descriptor (FD) of a locally opened file to the FUSE module
  - FUSE module obtains the file structure associated with the FD
  - FUSE module uses the file structure in the following read/write



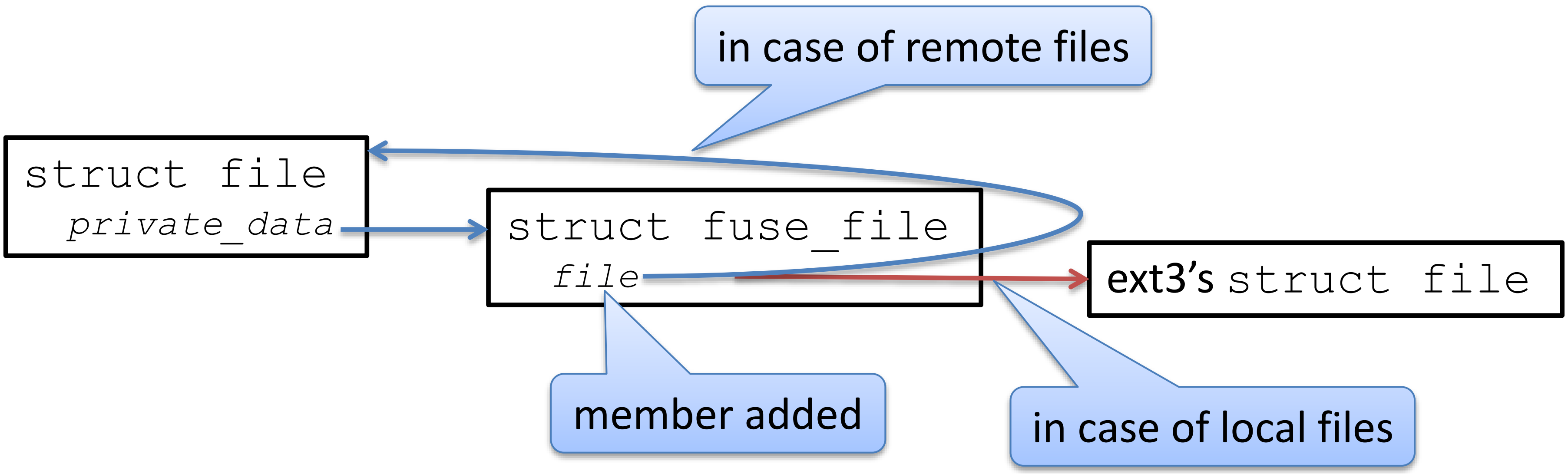
# Design (2/2)

- gfarm2fs sends -1 as FD for remote files
  - Thus, FUSE module can know the file is local or remote
  - FUSE module forwards, to gfarm2fs, read and write requests against remote files in the same way as before
- All operations other than read and write are executed through gfarm2fs
  - open, close, rename, truncate, stat, seek, …

# Implementation

- Modified programs
  - gfarm2fs
  - FUSE module
  - FUSE library
  - Gfarm library (minor modification)
- Modified parts
  - Part of communication between gfarm2fs and the FUSE module
    - We added a new member to `struct fuse_file_info` for passing a FD
  - Part of obtaining and managing file structures
  - Part of file operations of read/write

# Managing a File Structure




- In case of local files, file structure of ext3 can be obtained by referencing `file` of `struct fuse_file`
- In case of remote files, `file` points to the original file structure

# File Operations

- Both FUSE and ext3 (ext4) invoke the same functions for executing read/write
  - generic_file_read (ext3) / do_sync_read (ext4) generic_file_write (ext3) / do_sync_write (ext4)
  - These are default callback functions in FUSE framework
- We replace FUSE's callback functions for read/write with our interception functions
  - Thus read/write operations are intercepted

# Example of Interception Code

```
ssize_t
fuse_generic_file_read(struct file *filp,
                       char __user *buf,
                       size_t len,
                       loff_t *ppos){
    struct fuse_file *ff = filp->private_data;
    return generic_file_read(ff->file, buf, len, ppos);
}
```

Calls original function

In case of local files, it is a file structure of ext3.
In case of remote files, it is the same as filp (the original structure).

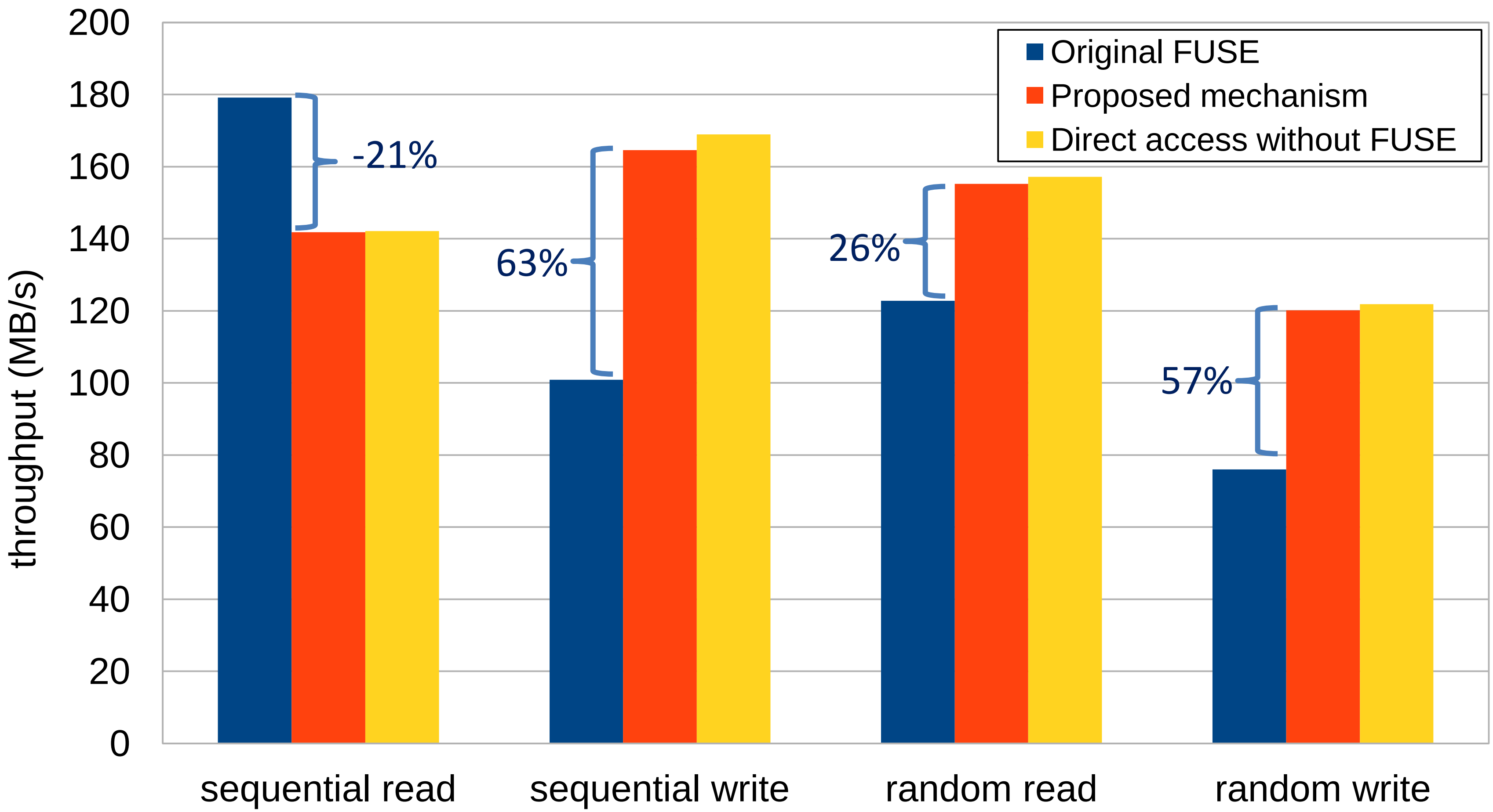
# Experiments

- Measured I/O throughput
  - Benchmarking with IOzone
    - File size: 100 GB
    - Record size: 8 MB
  - We compared:
    - Original FUSE
    - Proposed mechanism
    - Direct access without FUSE (ext3)

# Platform

- Cluster of 4 nodes connected with Infiniband QDR
  - 1 node: running metadata server
  - 3 nodes: running I/O server and client
- CPU: Intel Xeon E5645 2.40 GHz (6 cores) × 2
- memory: 48 GB
- HDD: SAS 600 GB 15,000 rpm
- OS: CentOS 5.6 64 bit (kernel-2.6.18-238.el5)
- File system: ext3
- Gfarm: version 2.4.2
- FUSE: version 2.7.4
- gfarm2fs: version 1.2.3

# Throughput



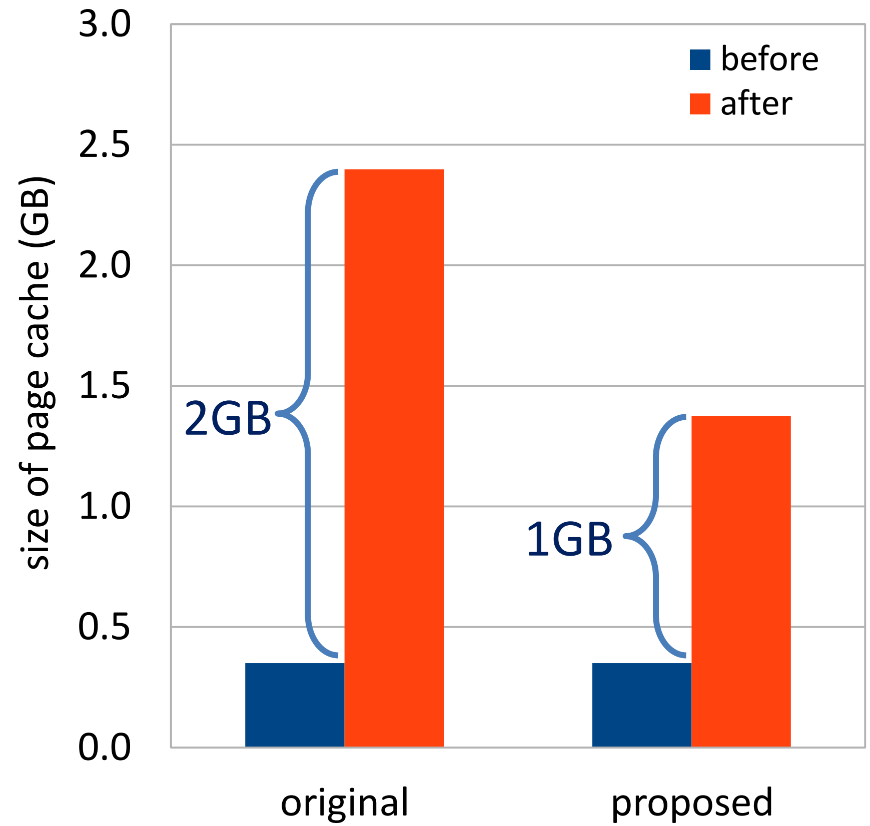

- 26-63% improvement when compared with original FUSE (except read)
- 97-99% throughput of direct access
- Read throughput is affected by OS kernel's read ahead mechanism

# Further Experiments

- Measured memory consumption for page cache
  - Page cache sizes before and after reading a 1 GB file
  - We compared:
    - Original FUSE
    - Proposed mechanism

# Memory Usage for Page Cache

- Before and after reading 1GB file
- Increase in page cache
  - Original FUSE: 2GB
  - Our mechanism: 1GB
- In original FUSE, both the FUSE module and ext3 create their own page cache
- In our mechanism, there exists ext3's cache only



# Related Work (1/2)

- LDPLFS [Wright et al., ’12]
  - Uses DLL to retarget POSIX file operations to functions of a parallel FS
  - Can accelerate file read and write without modification to application code
  - Does not work well with statically-linked binary code
- [Narayan et al. ’10]
  - Stackable FUSE module is used for reducing context switch overhead of FUSE
  - They apply the approach to encryption FS
  - They do not mention application to distributed FS

# Related Work (2/2)

- Ceph [Weil '06]
  - Distributed FS that provides highly scalable storage
  - Because the Ceph client is merged with the Linux kernel, Ceph file system can be mounted without FUSE
  - Client nodes are different from storage nodes
- Lustre
  - Distributed FS that consists of a metadata server and object storage targets, which store the contents of files
  - Lustre client nodes are assumed to be distinct from object storage targets

Gfarm aims to improve performance by using local storage effectively when a client and I/O server share the same node

# Summary and Future Work

- We proposed a mechanism for accessing local storage directly from a modified FUSE module, and adapted it to Gfarm
- Applications running with our mechanism installed can read and write a local Gfarm FS without the intervention of FUSE daemon
- We measured 26-63% increase in read and write throughput
  - In one sense, we show an “upper bound” of improvement in ideal case
- Future work
  - Develop a kernel module (driver) that also allows access to remote I/O servers without the intervention of FUSE daemon