An Active Storage Framework for Object Storage Devices

Michael Runde
Wesley Stevens
Paul Wortman
John Chandy

University of Connecticut

The goal:

To enable the execution of code directly on networked storage servers

Why Active Storage?

- Based on work by Riedel, Acharya
- Allows us to run applications directly on storage nodes
  - Storage nodes can now also compute
- Can dramatically reduce data traffic
  - Also possibly eliminate large network latencies
- Take better advantage of fast RAID arrays and SSDs
  - Drives bottle-necked by slow networks
- Run applications in parallel across multiple nodes
- Make use of unused processor time

Why Object Storage Devices (OSDs)

- Moves part of the filesystem to the disk
- Deals with objects instead of blocks
- Data and metadata are separated
  - Direct access to data once authorized
- Object attributes are flexible and directly modifiable
- Objects accessible by name on nodes, not just as bytes
- High throughput possible through striping data
- Included in the Linux kernel as of the 2.6.30 release

Where We Started

- Built on top of an open-source OSD stack
- OSD initiator and target developed originally by Ohio Supercomputer Center (OSC)
  - Implements the SCSI T10 OSD spec
- Both undergoing active development headed by Panasas who sells commercial OSD style systems
  - Plan future OSD spec compliance

Programming Model

- Acharya, Riedel, Xie(Oasis) - Stream based
- MapReduce - Hard to transform some problems
- Our model is Remote Procedure Call (RPC) based
  - Use executable objects
  - Added command to begin execution
  - Allow full access to all OSD functions
- Functions can be run sync or async
  - Due to iSCSI 30sec timeout
  - Working to allow queries for async
- Allow parallel execution using async
- Support multiple languages (C, Java, Python)

OSD Specification Changes

- Addition of the EXECUTE_FUNCTION() command
- Sent over iSCSI
- Triggers execution of an executable object
- Carries information including:
  - Object id
  - Arguments to the function
  - Return data from function

OSD Specification Changes

- Root information page appended to include:
  - List of supported virtual machines
    - Type (C, Java, etc.)
    - Min/max API supported
    - Engine implementation version
- Additional object attributes including:
  - Type(C, Java, etc)
  - Min/max API supported

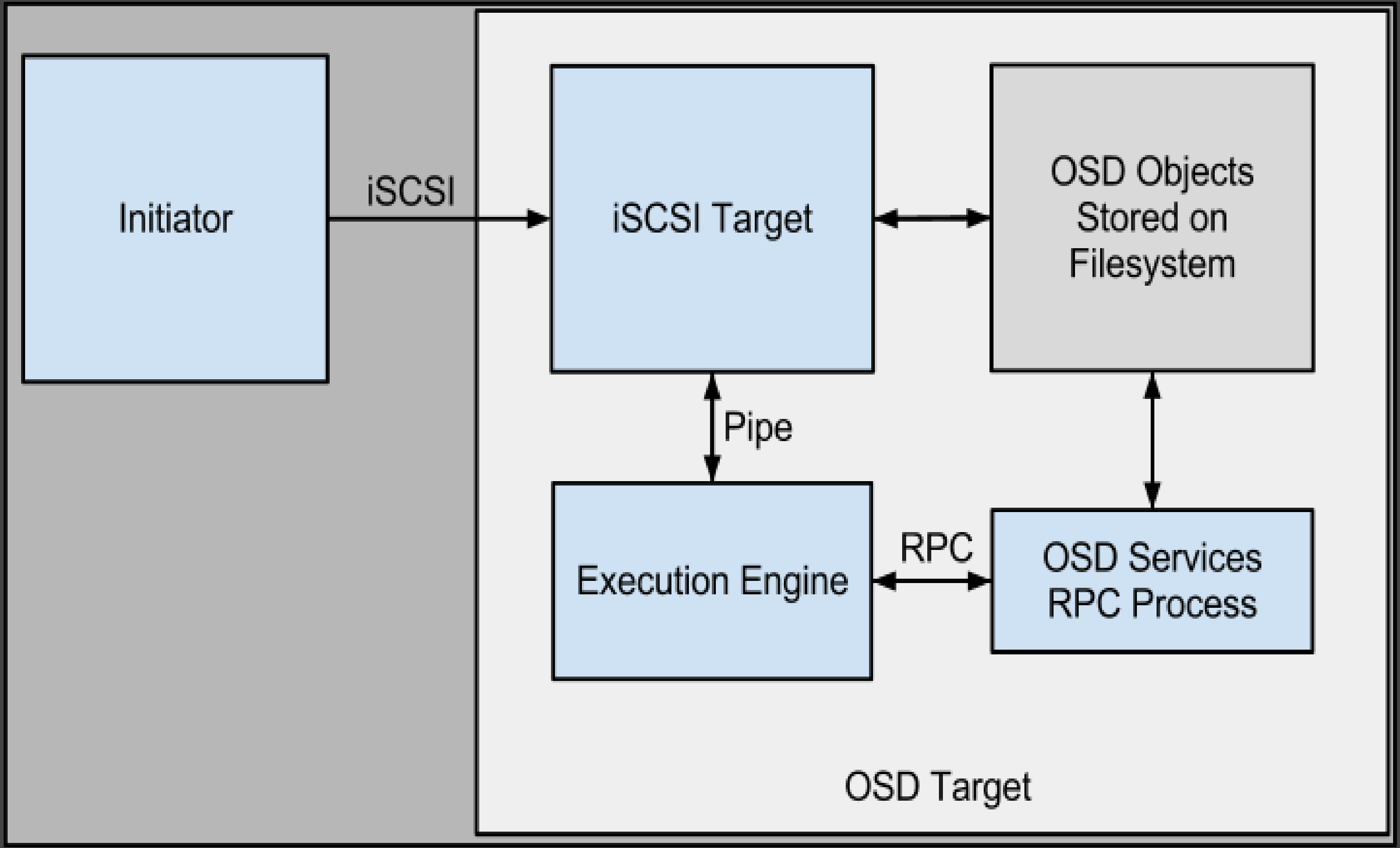
Security

- Multiprocess Implementation
  - Limits AS functions from directly accessing objects
  - Limits access to the OSD services library
    - Forces the use of RPC
  - Enforces the use of OSD security mechanisms
- Chroot Sandboxing
  - Applied to engines
  - Limits engines inside a single directory
  - Allows limiting of libraries
    - AS versions of libraries possible

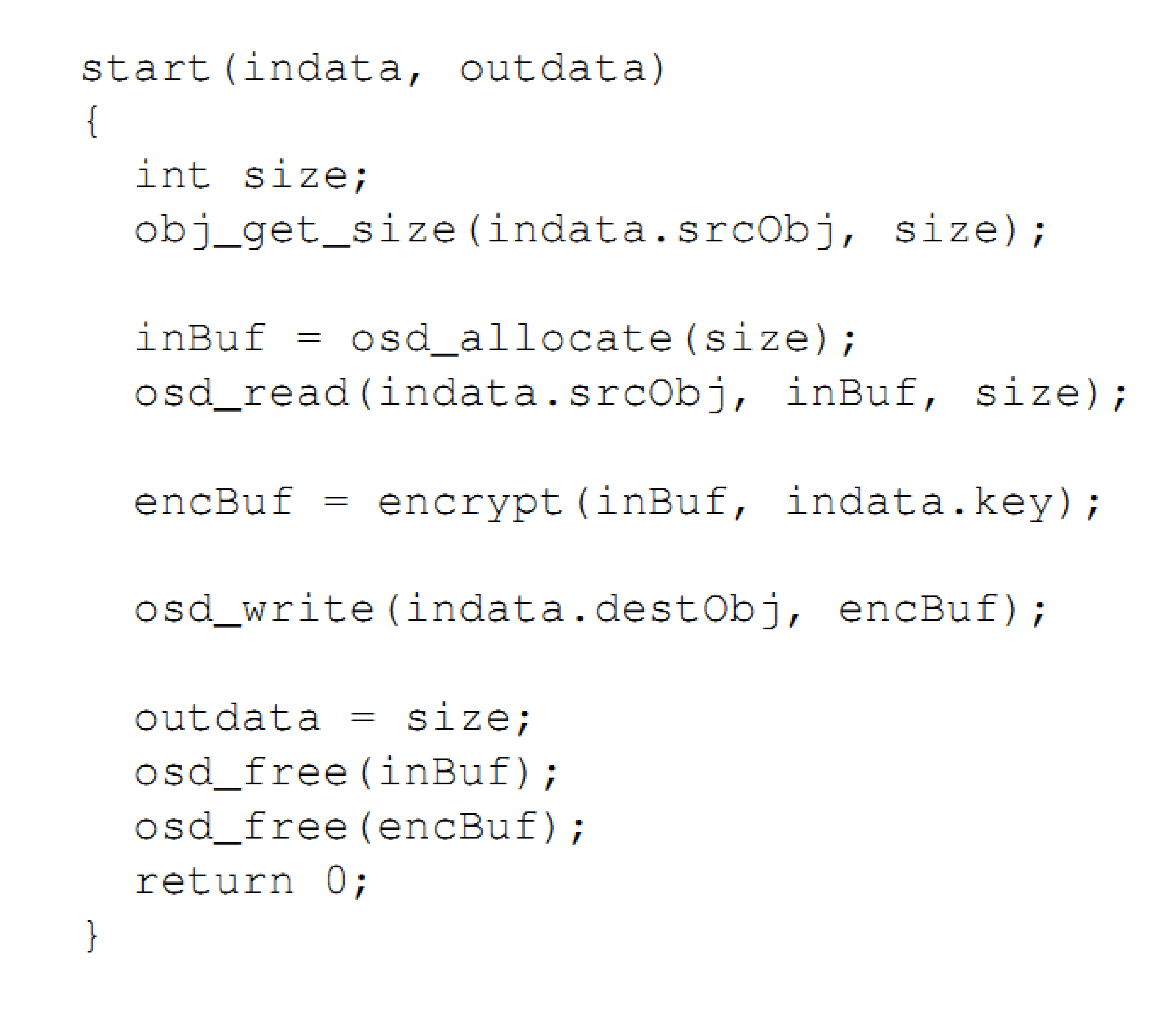
Active OSD Implementation

- Active storage engines are implemented on the target
  - Provide an API to allow access to storage objects
  - Sandboxed to limit AS code to this API
  - Currently support C and Java, Python forthcoming
    - C functions are shared libraries
    - Java functions are JAR archives
  - Supported by the OSD Services RPC Process
- Simultaneous execution supported
  - Includes long running functions

Active Storage OSD

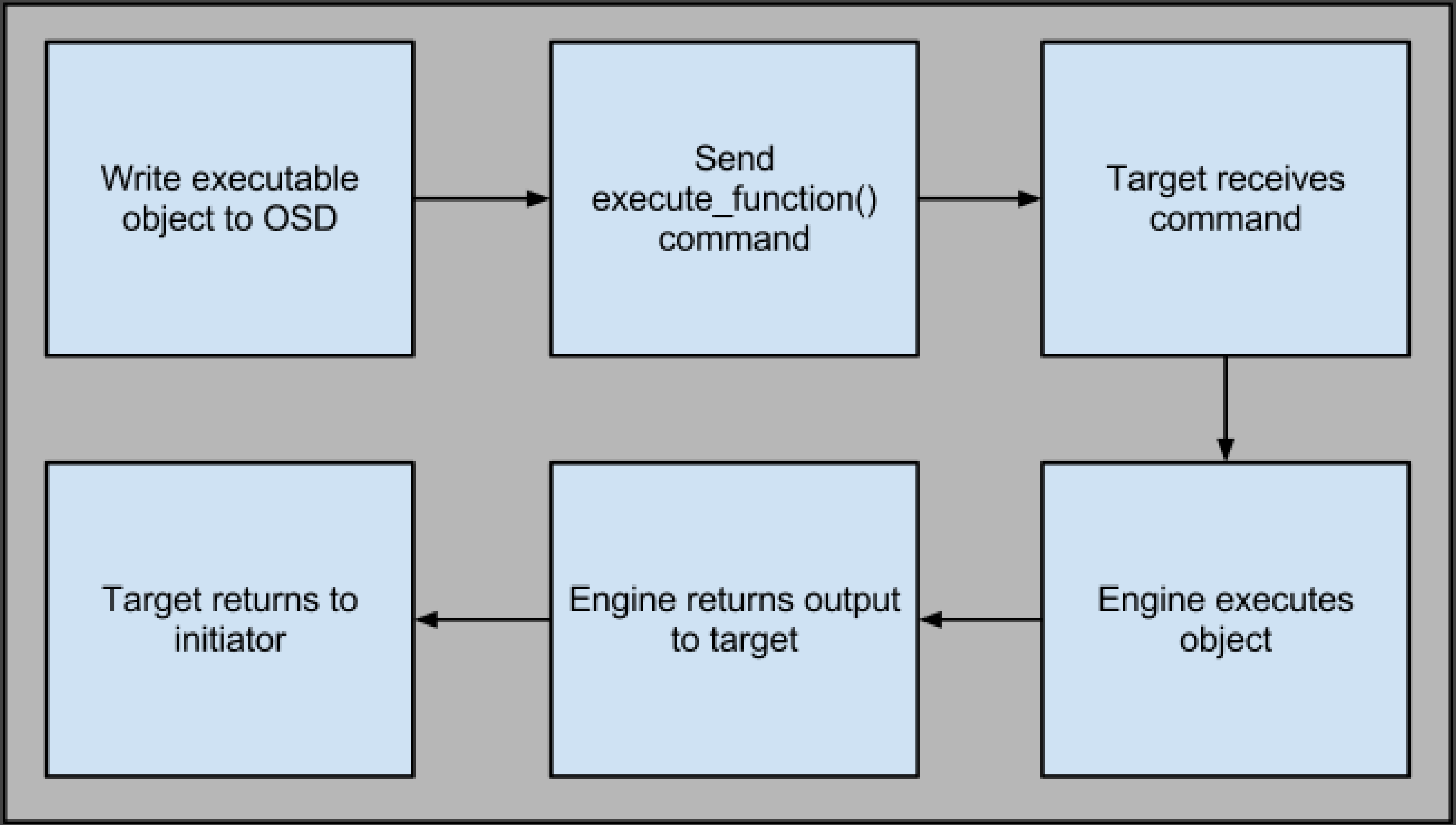


Active Storage Encrypt Code Example



```
start(indata, outdata)
{
  int size;
  obj_get_size(indata.srcObj, size);

  inBuf = osd_allocate(size);
  osd_read(indata.srcObj, inBuf, size);

  encBuf = encrypt(inBuf, indata.key);

  osd_write(indata.destObj, encBuf);

  outdata = size;
  osd_free(inBuf);
  osd_free(encBuf);
  return 0;
}
```

Active OSD Execution Flow



Write executable object to OSD

Send execute_function() command

Target receives command

Engine executes object

Engine returns output to target

Target returns to initiator

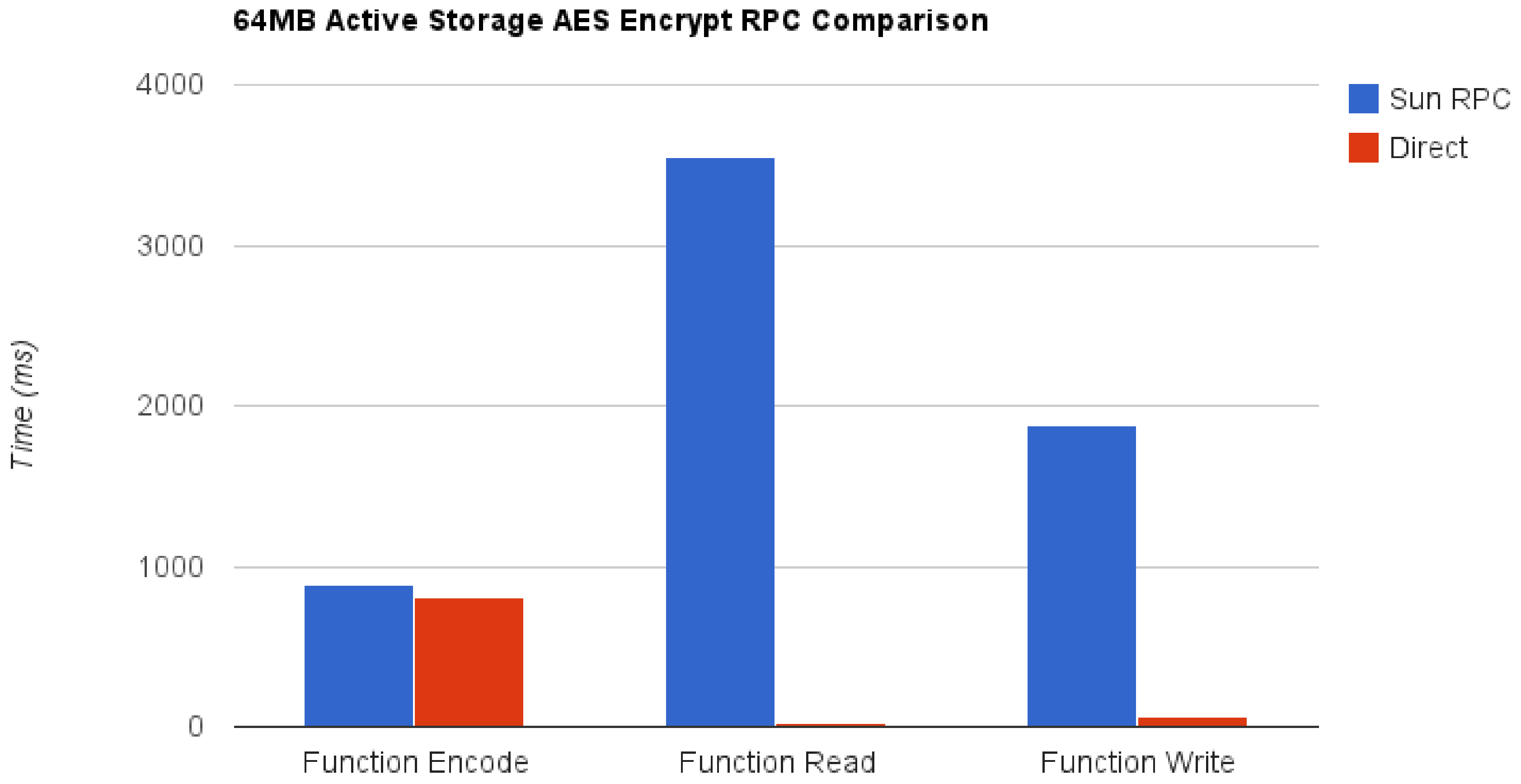
Active Storage Engines

- Take in arguments, execute function and return results
- All engines share common code written in C
- RPC and sandbox initialization
  - Initially used standard SunRPC
- OSD RPC calls
  - Used for all OSD function calls from application
- Communication to the iSCSI target through a pipe
- Only started on first execute_function() call
- One engine for each supported language

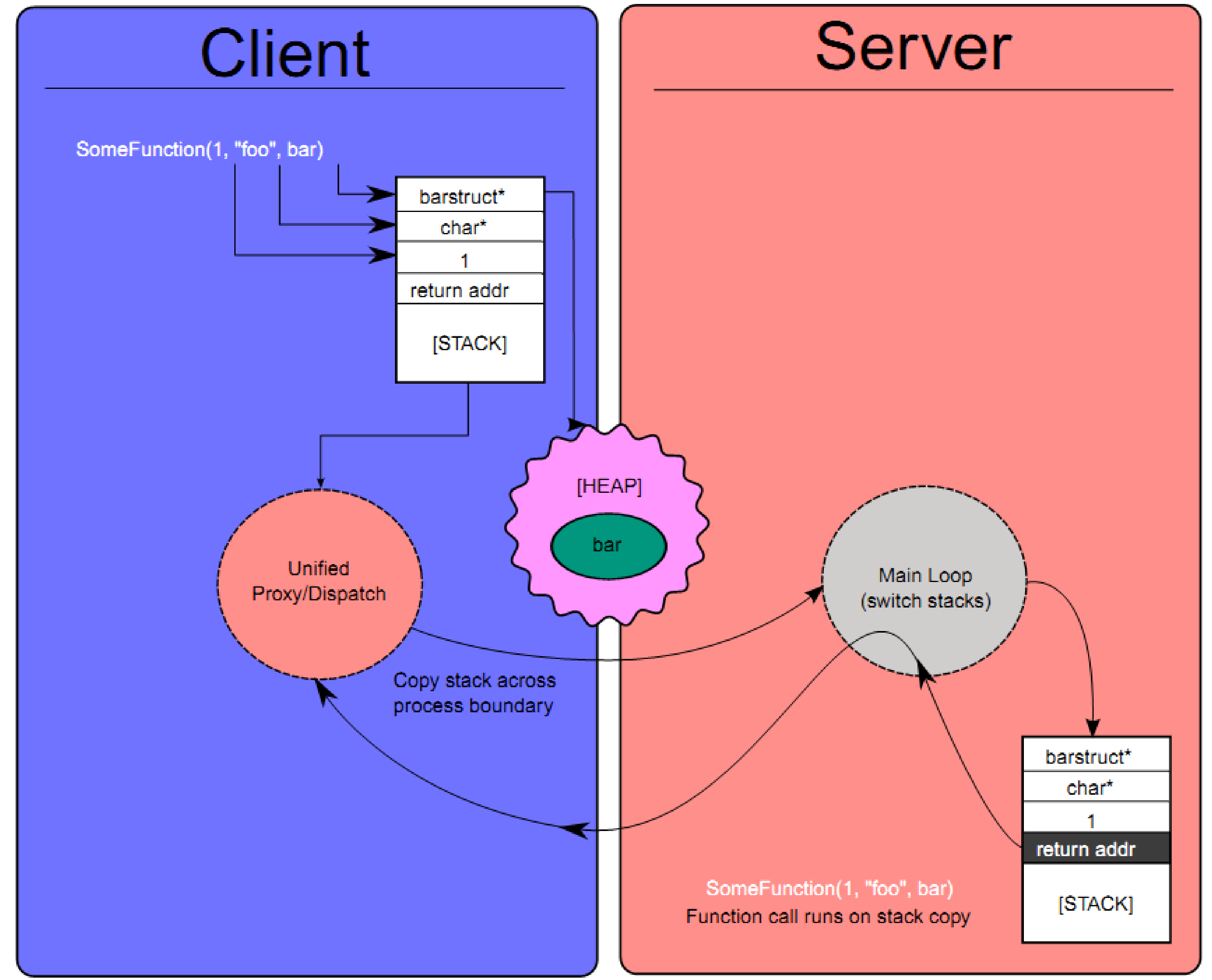
Active Storage Engines

- C-engine:
  - Opens objects as C shared libraries
  - Main "start" function then called like a local function
  - Header file defines start and other OSD prototypes
- Java-engine:
  - Opens JAR (Java Archives)
  - Creates a JVM
  - Uses JNI for Java -> C function calling
  - Helper Java applications handle loading and execution
  - Contains a separate API class to define OSD calls

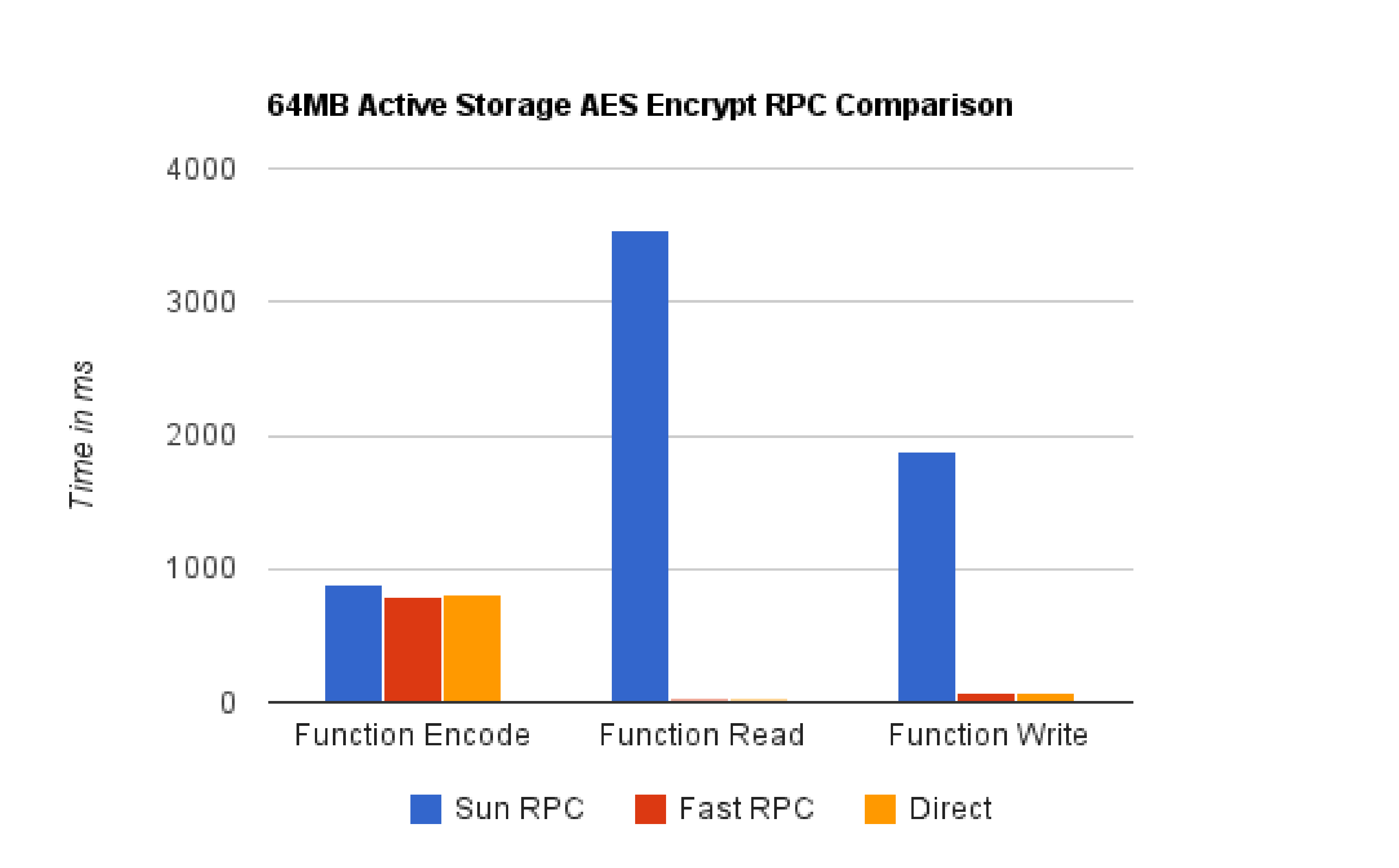
Early Testing: Not so good



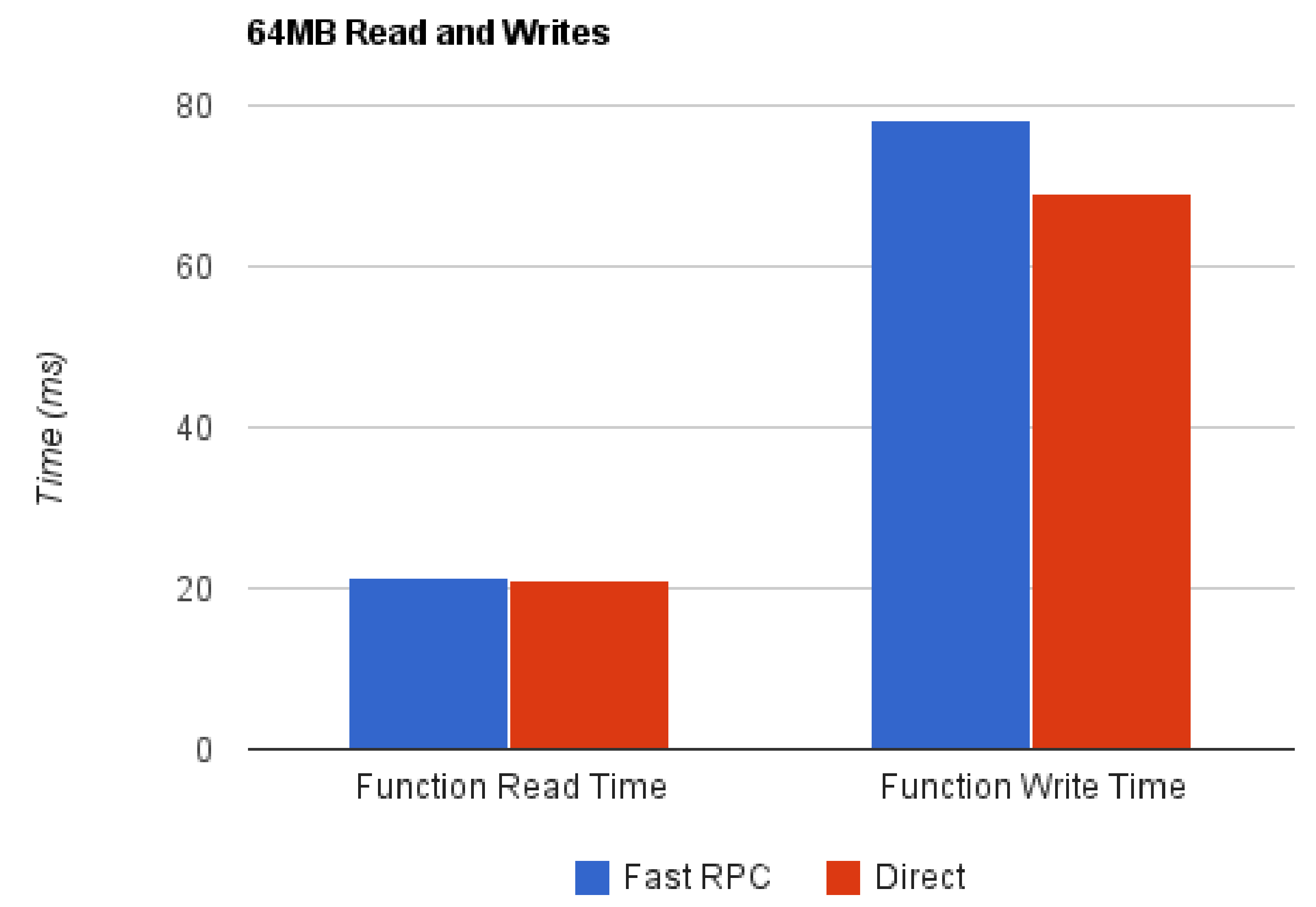
Fast RPC - Mike Hearn, Univ. of Durham


Client
Server
SomeFunction(1, "foo", bar)
barstruct*
char*
1
return addr
[STACK]
[HEAP]
bar
Unified Proxy/Dispatch
Main Loop (switch stacks)
Copy stack across process boundary
barstruct*
char*
1
return addr
[STACK]
SomeFunction(1, "foo", bar)
Function call runs on stack copy

Early Results: Enter fast RPC



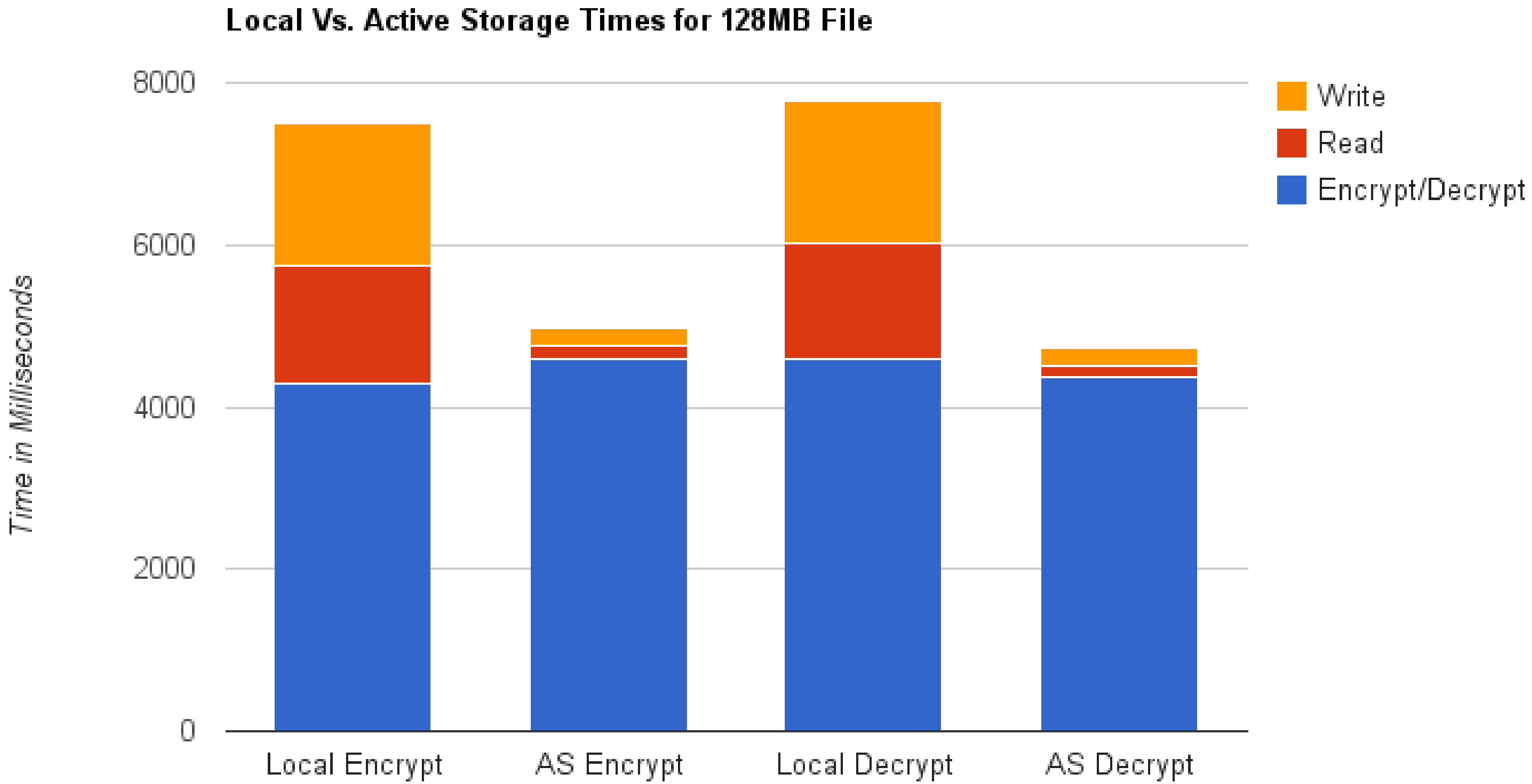
Early Results: Enter fast RPC



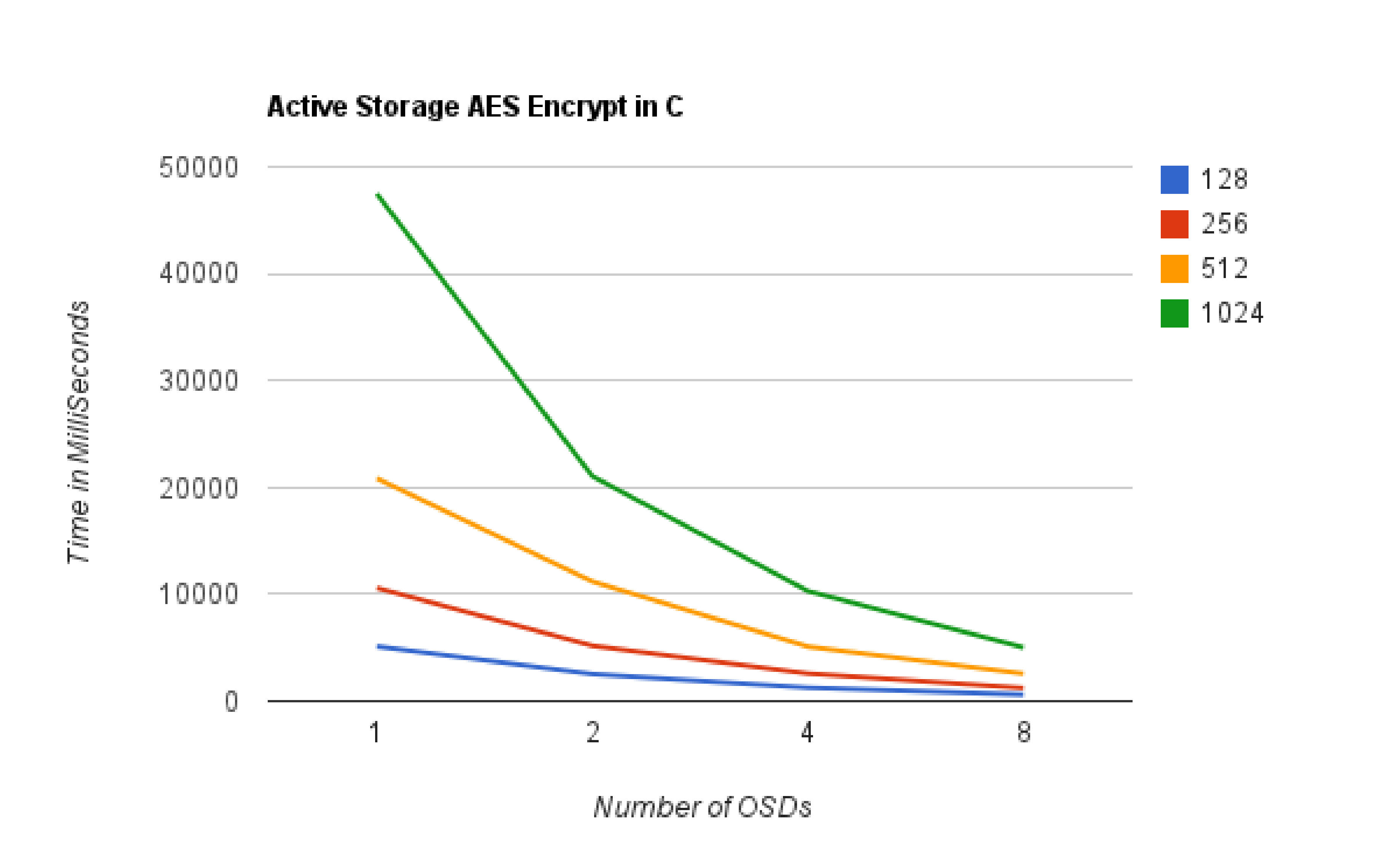
OSD Services RPC Process

- Essentially an RPC based OSD target
- Shares OSD code with the iSCSI target
- Exists outside of the sandbox
- Shares the metadata database and objects on disk
- Converted to FastRPC for performance improvements
    - RPC designed around local communications
    - Utilizes a single shared heap
        - Eliminates copying buffers
    - Only copies stack from engine
    - 100x speed improvement over Sun RPC

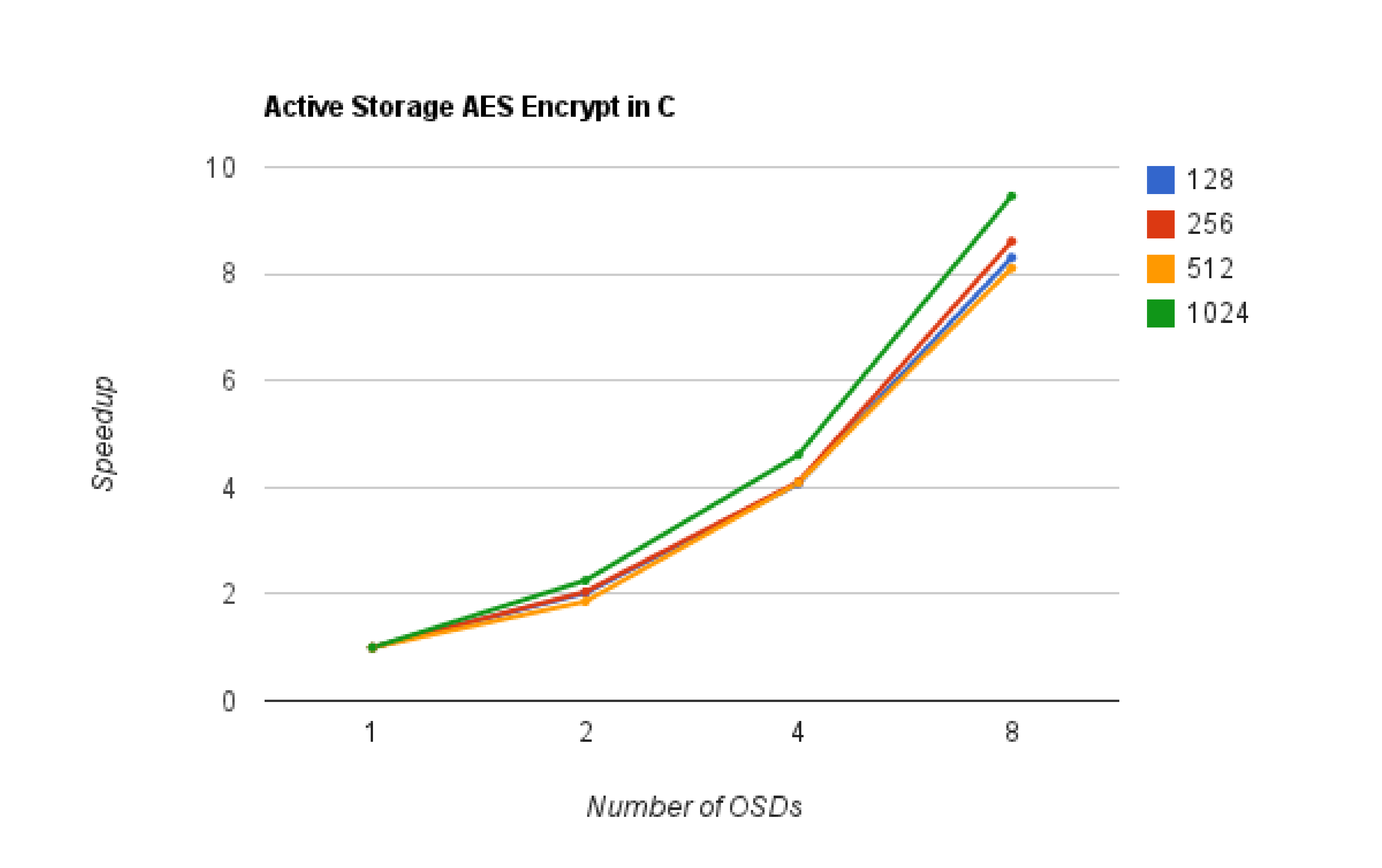
Results: AES Local vs. Active Storage



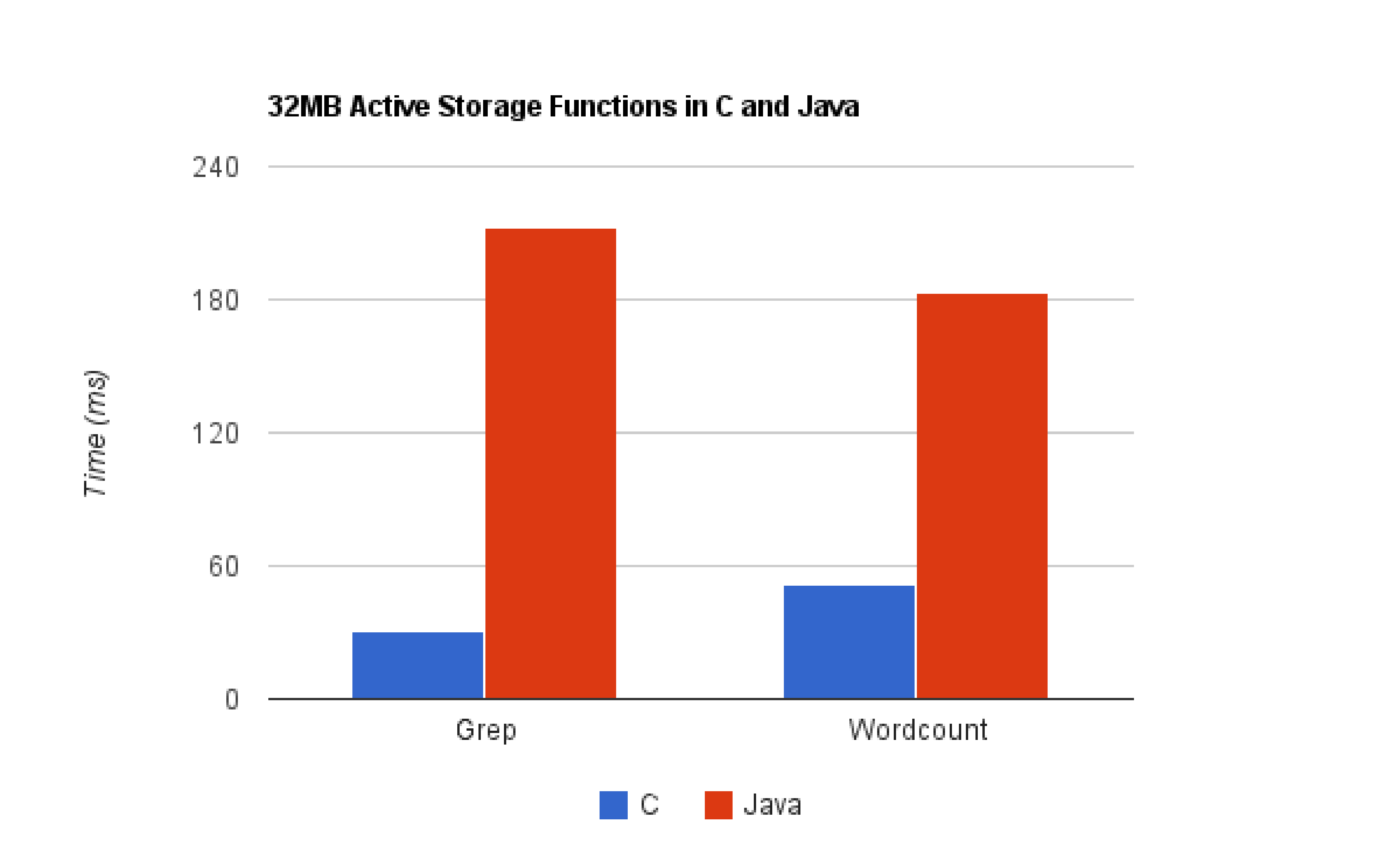
Results: Scaling with Multiple OSDs



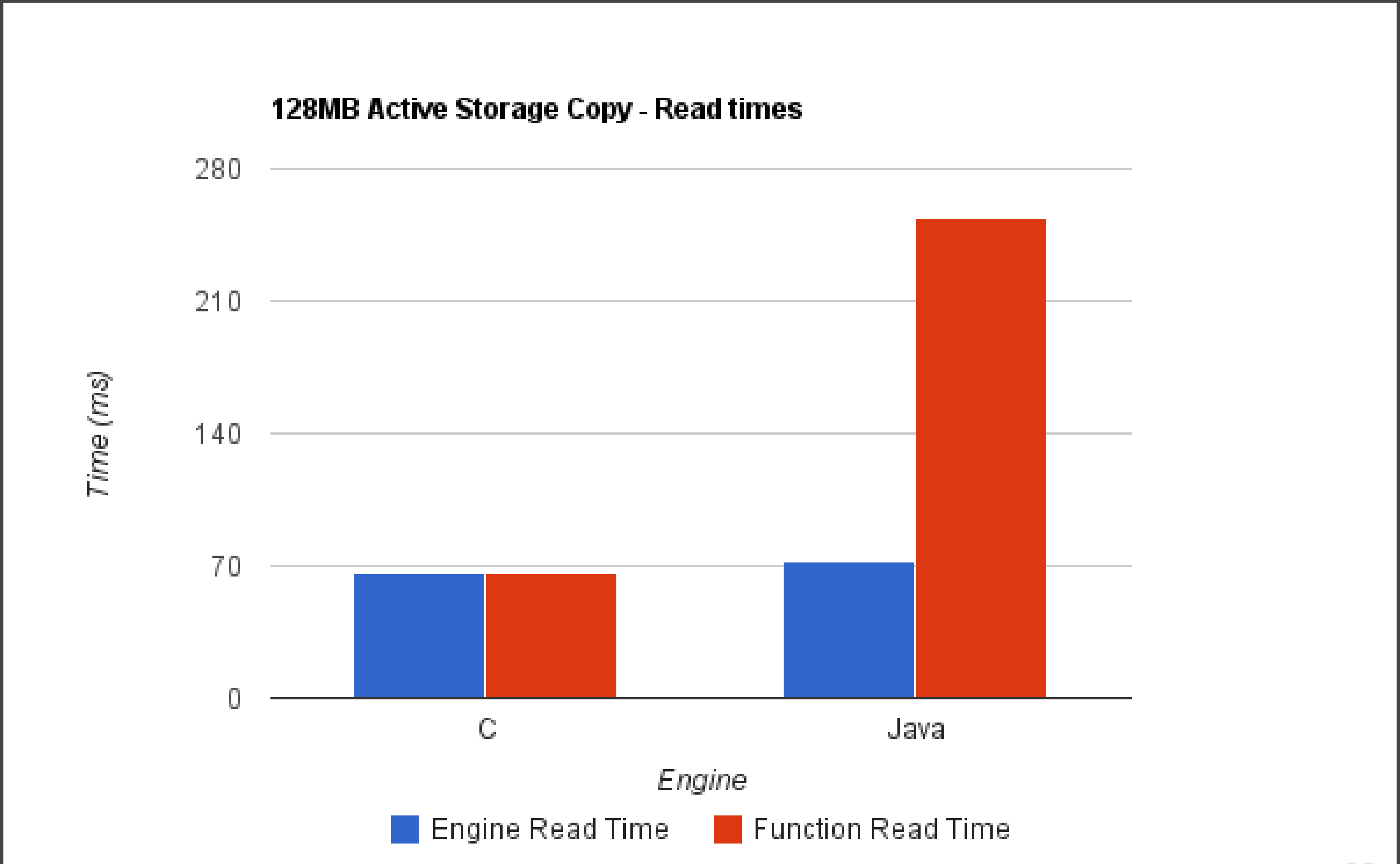
Results: Scaling with Multiple OSDs



Results: C vs. Java



Results: C vs. Java Read Times



Future Work

- Implement async queries for long running functions
- Multithread target to allow:
    - Access to OSD while executing objects
    - Multiple simultaneous engines
- Finish implementing Python engine
- Make code available for download
- Re-write engine RPC code in their native languages
    - eg. remove need for JNI interface for RPC
    - Possible performance improvements

Conclusion

- Active storage provides a way to:
  - Transfer data processing to a network storage device
  - Decrease transfer times
  - Provide scalability utilizing multiple OSDs
- This framework allows for:
  - Multiple execution engines
  - Security through the sandbox and separate proceses
  - Low overhead
    - Empty function takes ~500us
  - Scalability across multiple OSDs
- Thanks to NSF for generous support of this work