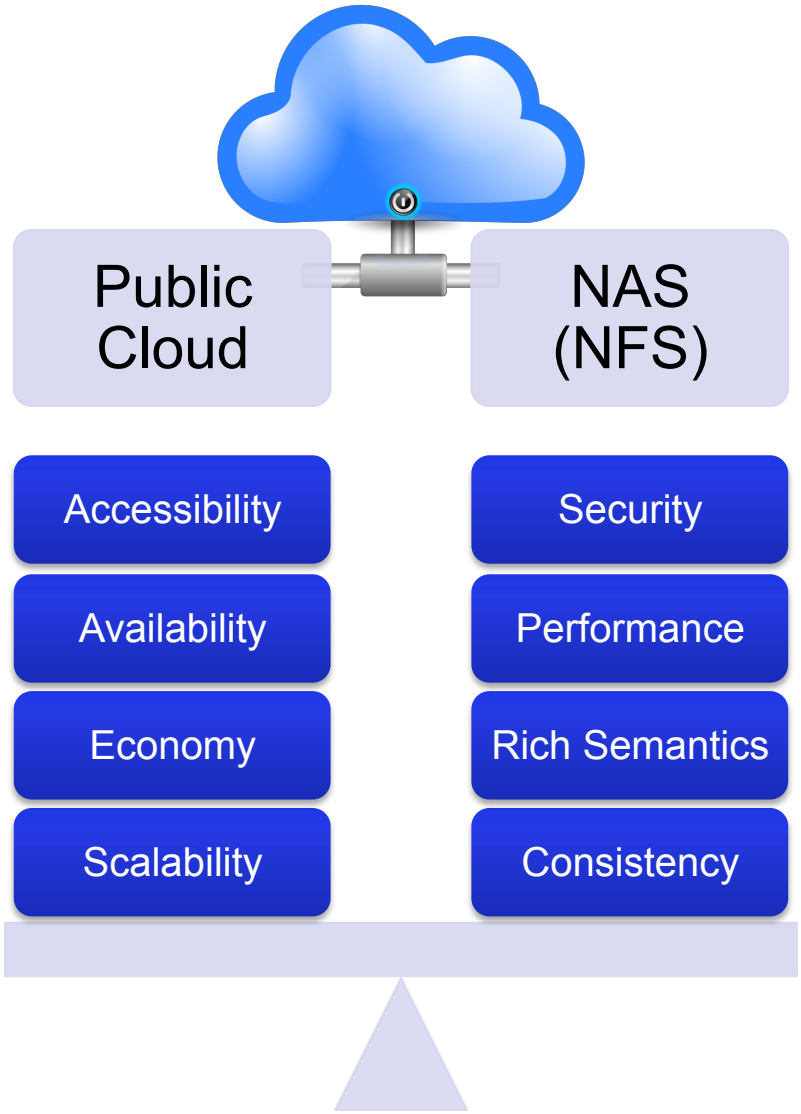
Kurma: Secure Geo-distributed Multi-cloud Storage Gateways

Ming Chen and Erez Zadok

Stony Brook University
File Systems and Storage Lab (FSL)

Cloud Storage Gateways



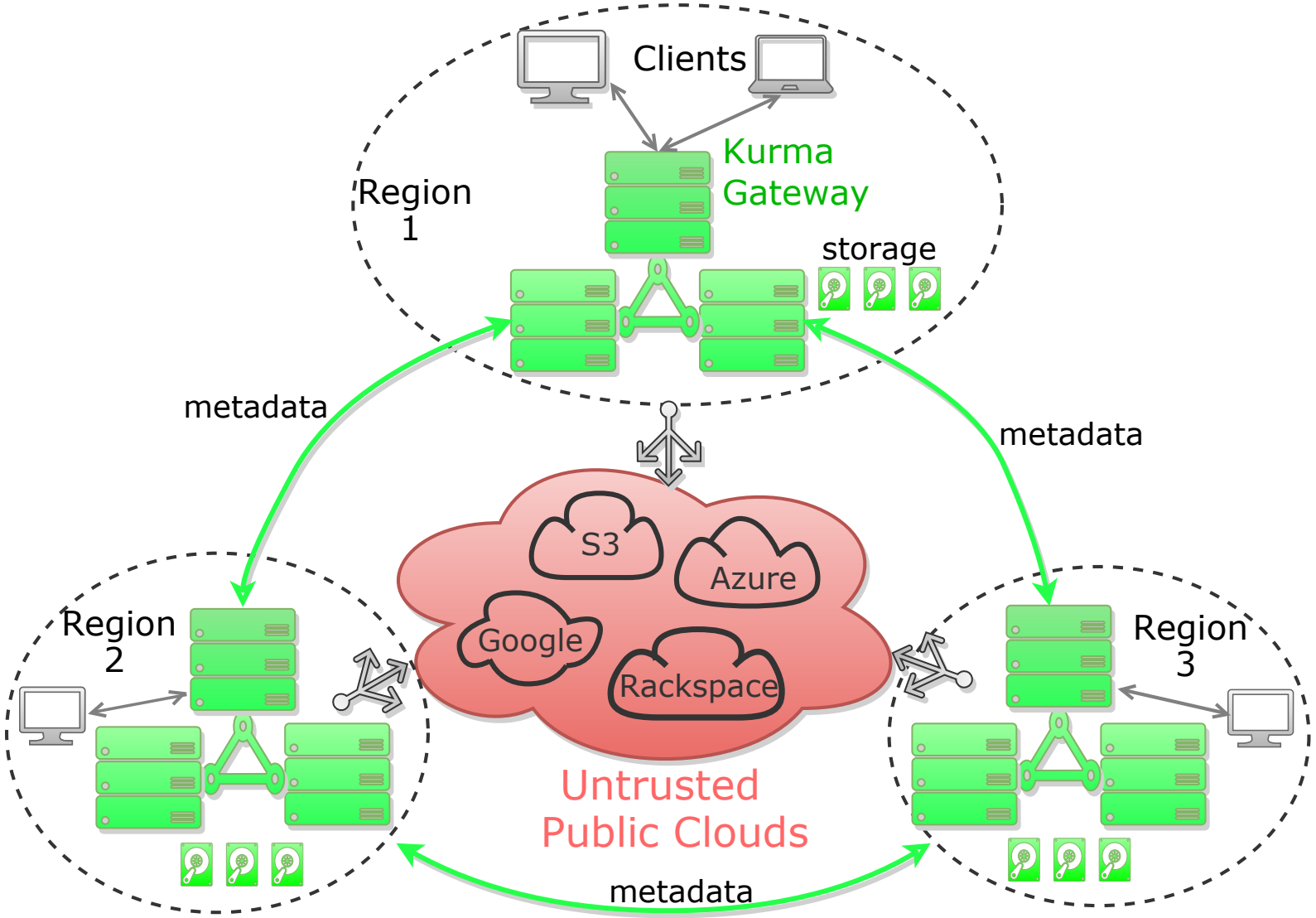
- Benefits of cloud gateways
  - Combine advantages of both clouds and traditional NAS
  - High security without relying on trusted third parties
  - Allow clients to use public clouds using network-attached storage (NAS) protocols but still share across regions

Stony Brook University

Kurma Design Goals

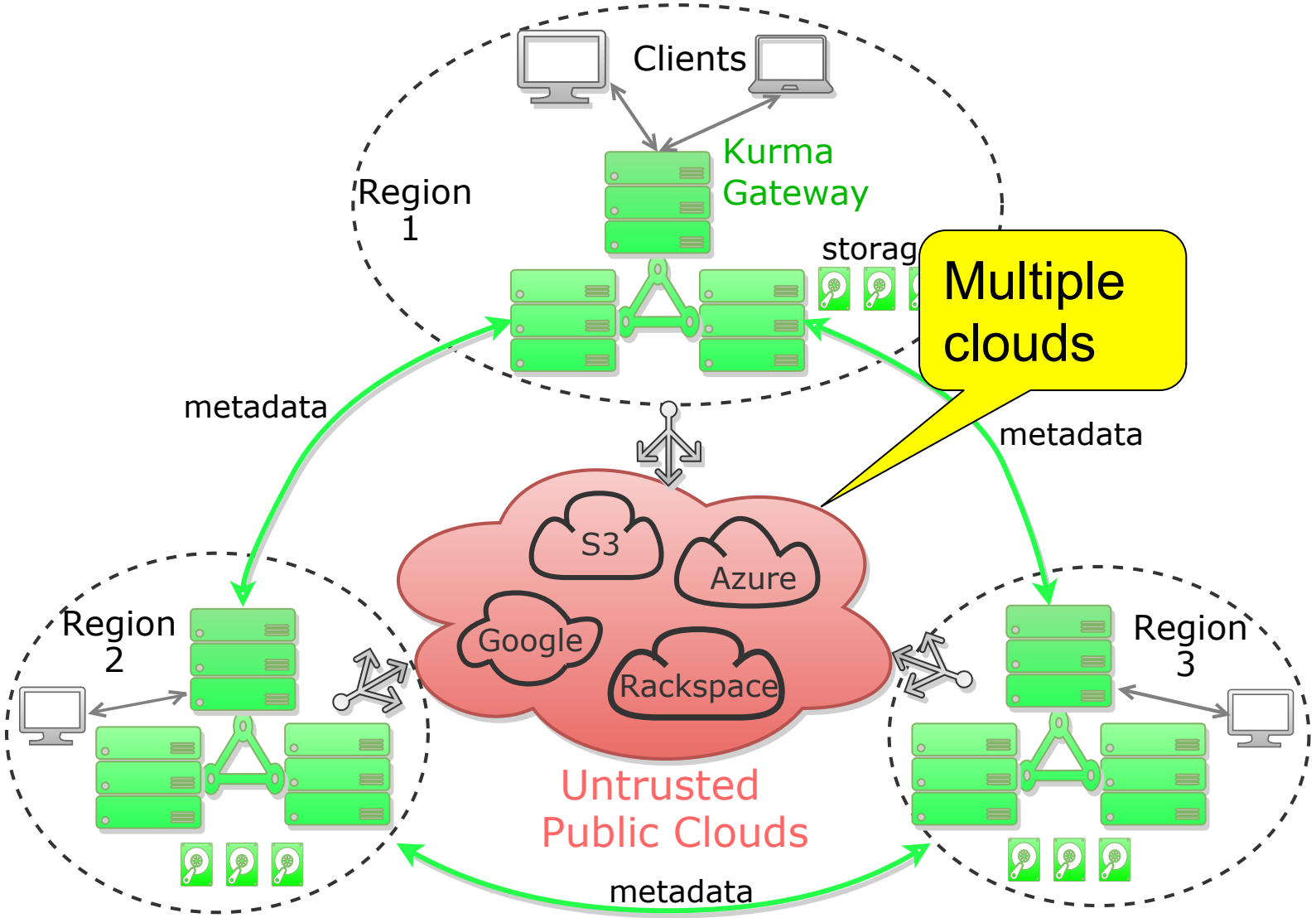
1. Strong Security
   - Use clouds to store only encrypted blocks
   - Share metadata directly among gateways
2. High availability
   - Use multiple public clouds
   - Each gateway is highly available (ZooKeeper)
3. High performance
   - Extensive caching for data and metadata
   - Asynchronous replication of metadata
4. High flexibility
   - Replication, erasure coding, and secret sharing

Kurma Architecture



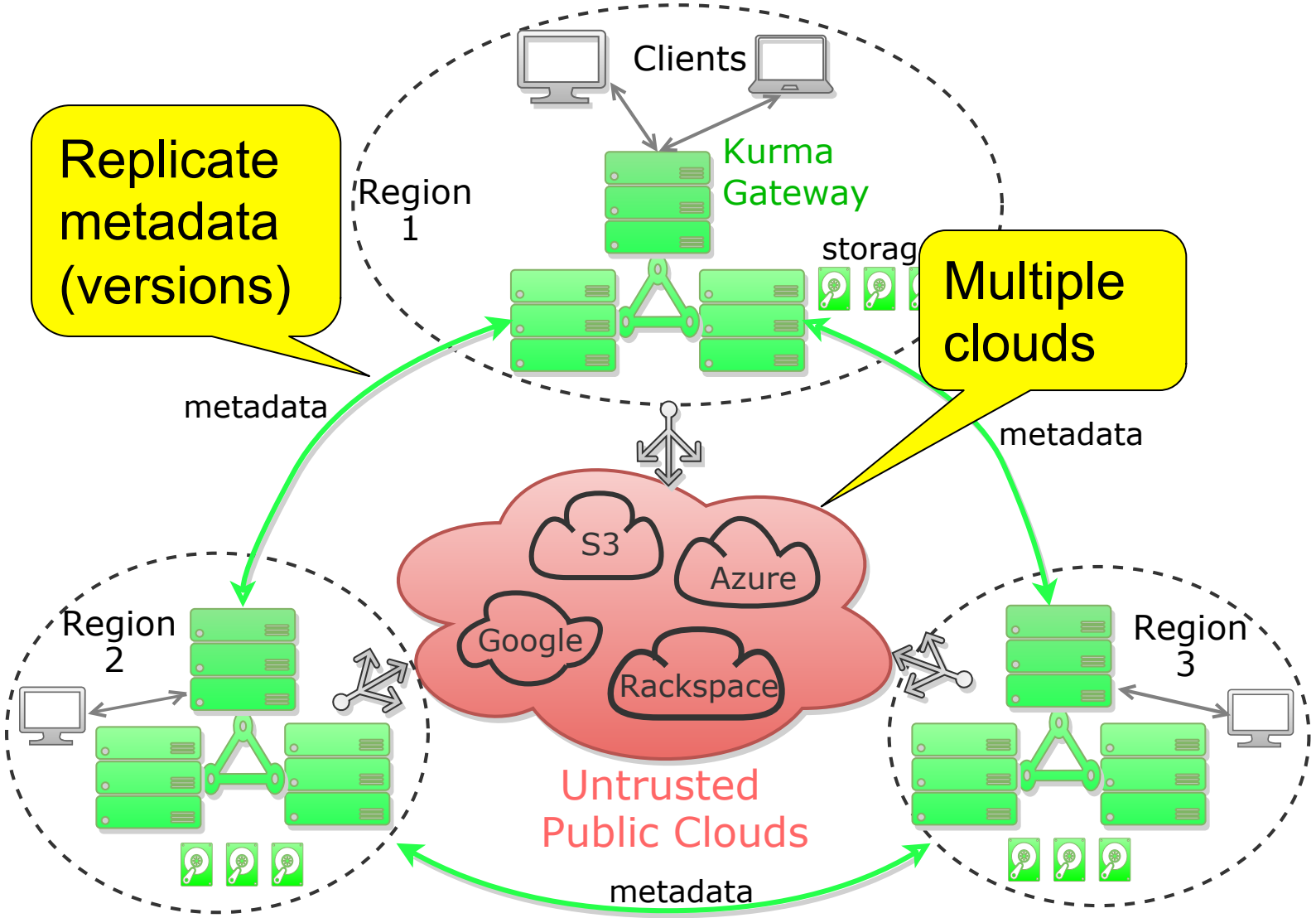
Stony Brook University

Kurma Architecture



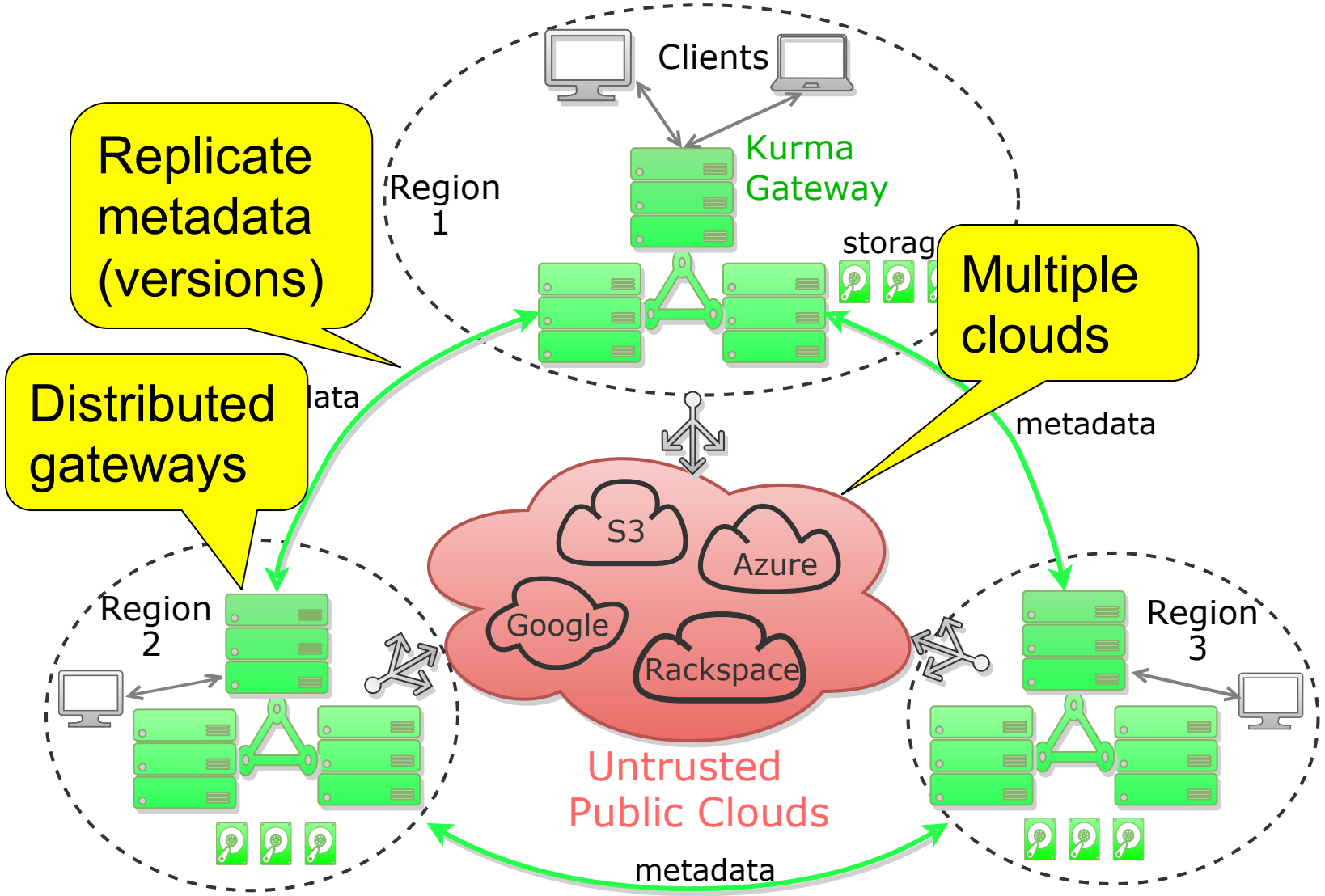
Stony Brook University

Kurma Architecture



Stony Brook University

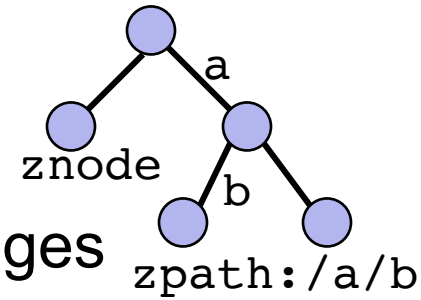
Kurma Architecture



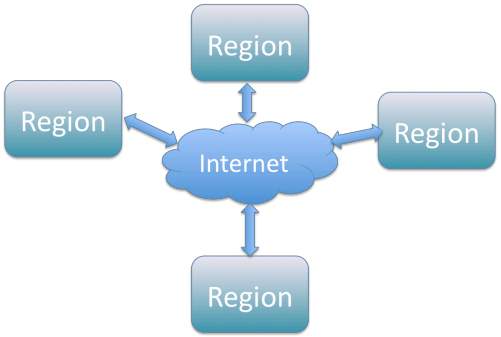
Stony Brook University

Background

- ZooKeeper: A distributed coordination service
  - Coordinate Kurma servers
  - Store Kurma FS metadata
  - Execute transactions of metadata changes



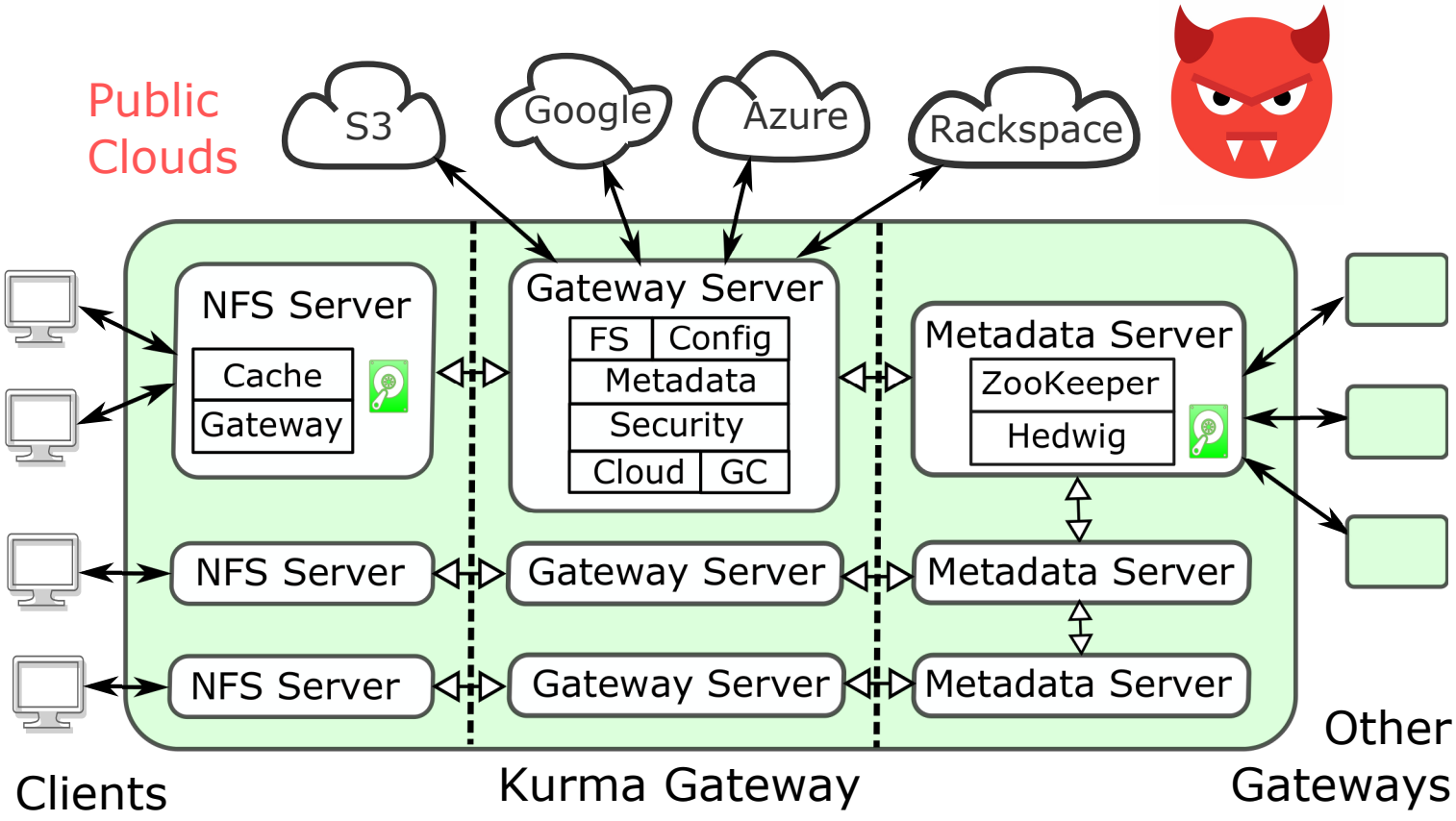
- Hedwig: A publish-subscribe system
  - Provide guaranteed delivery
  - Replicate Kurma metadata



- Thrift: A RPC framework
  - Define FS metadata format
  - RPC among Kurma servers

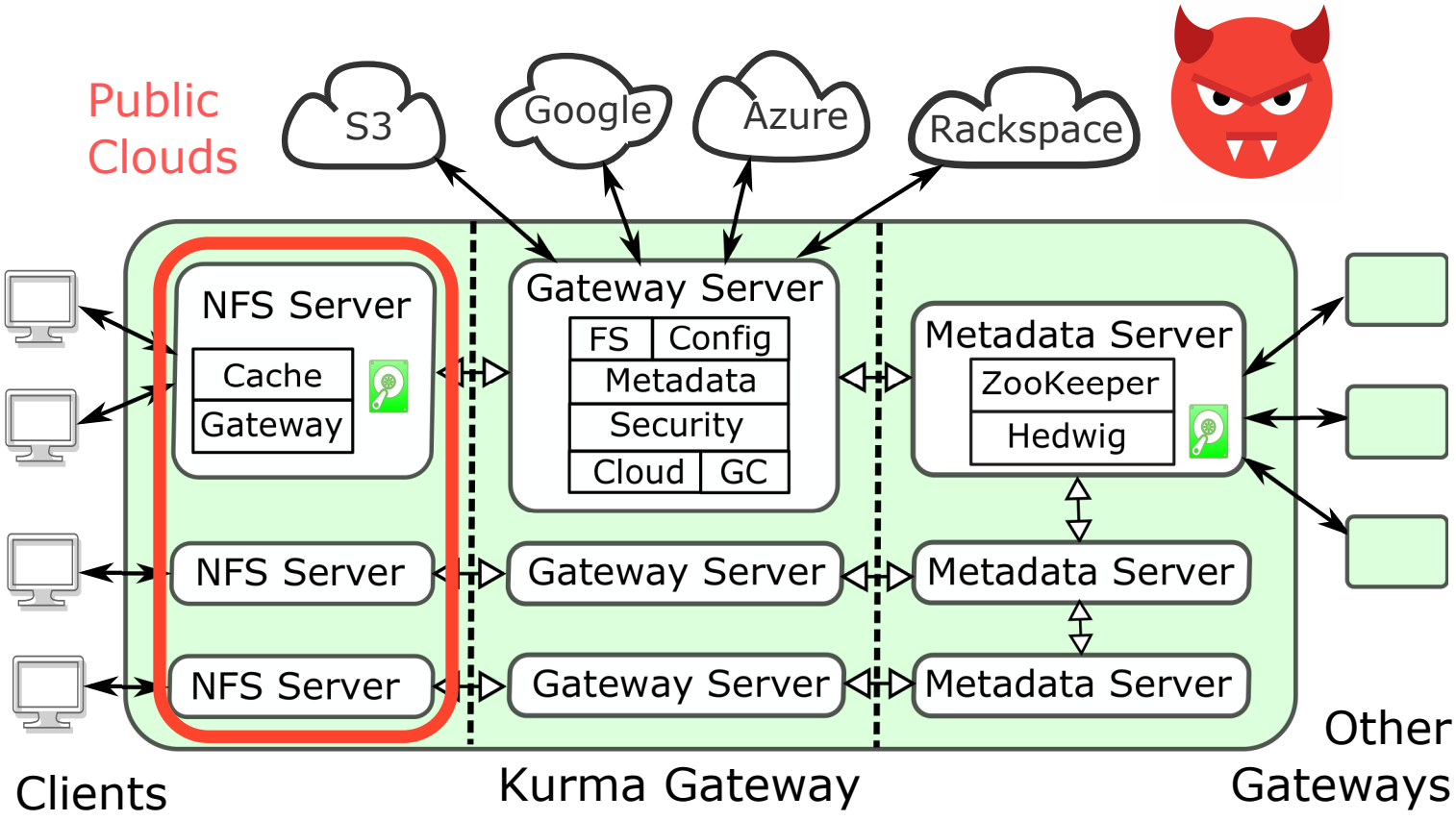
Stony Brook University

Components



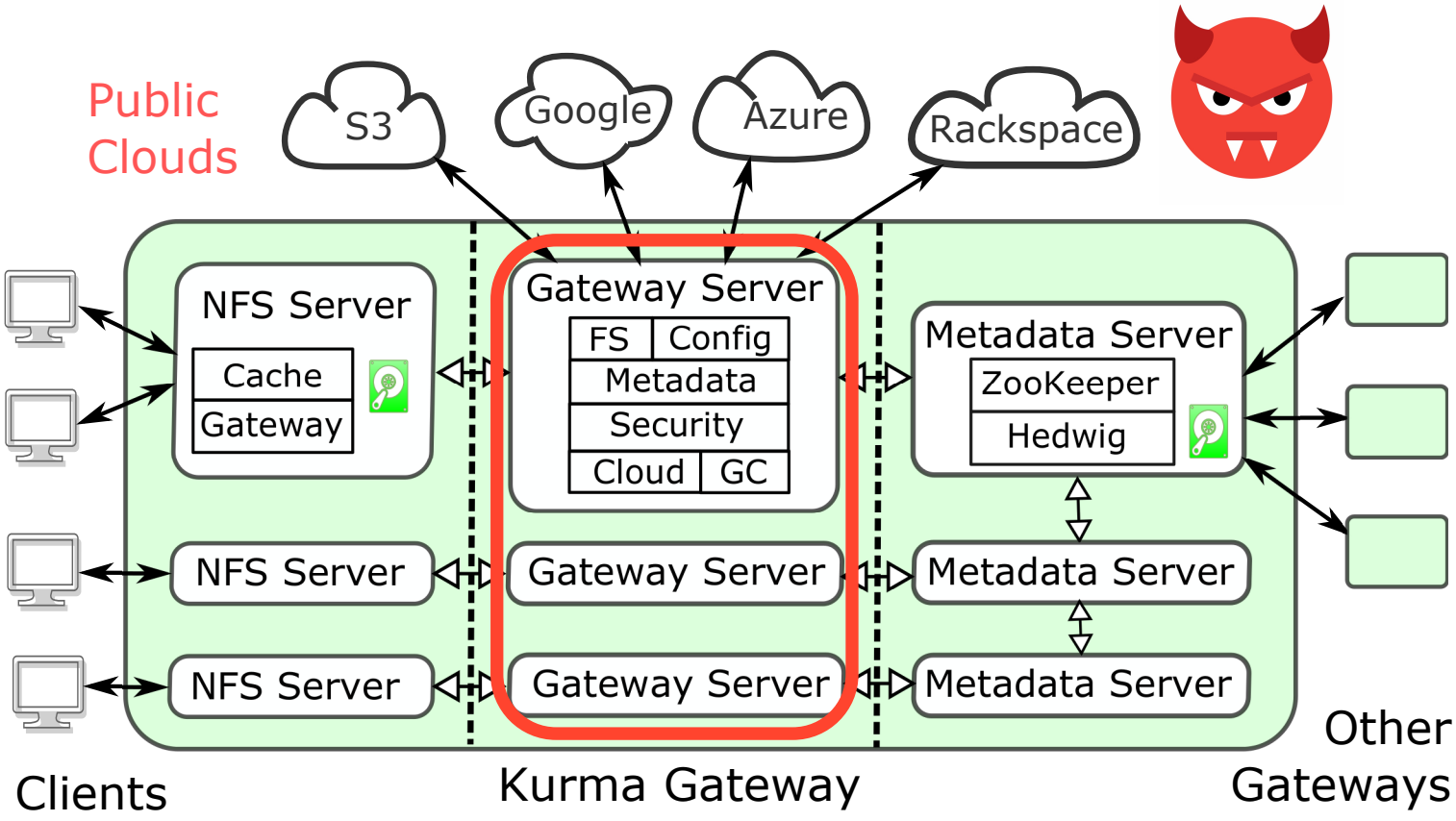
Stony Brook University

Components



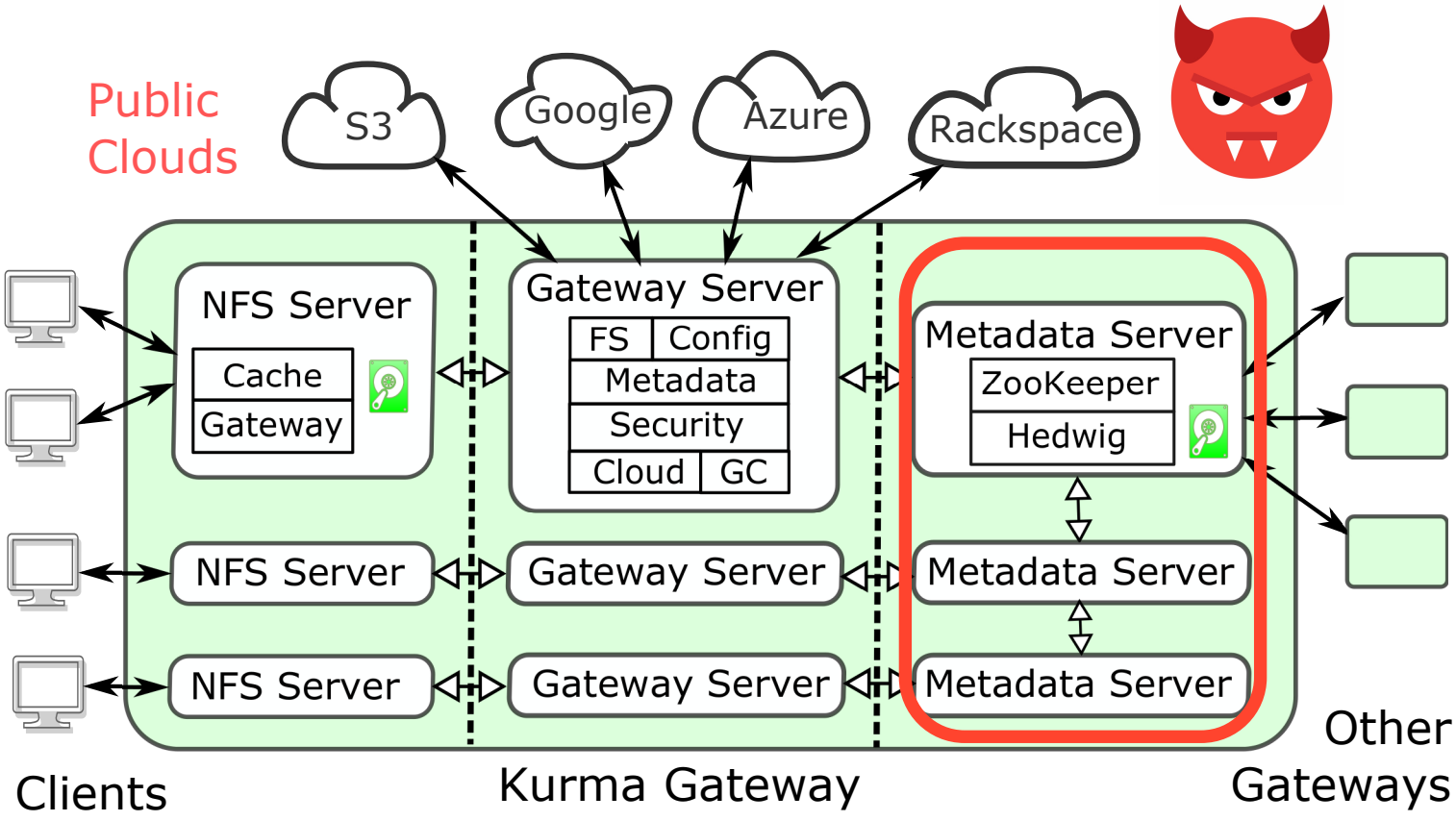
Stony Brook University

Components



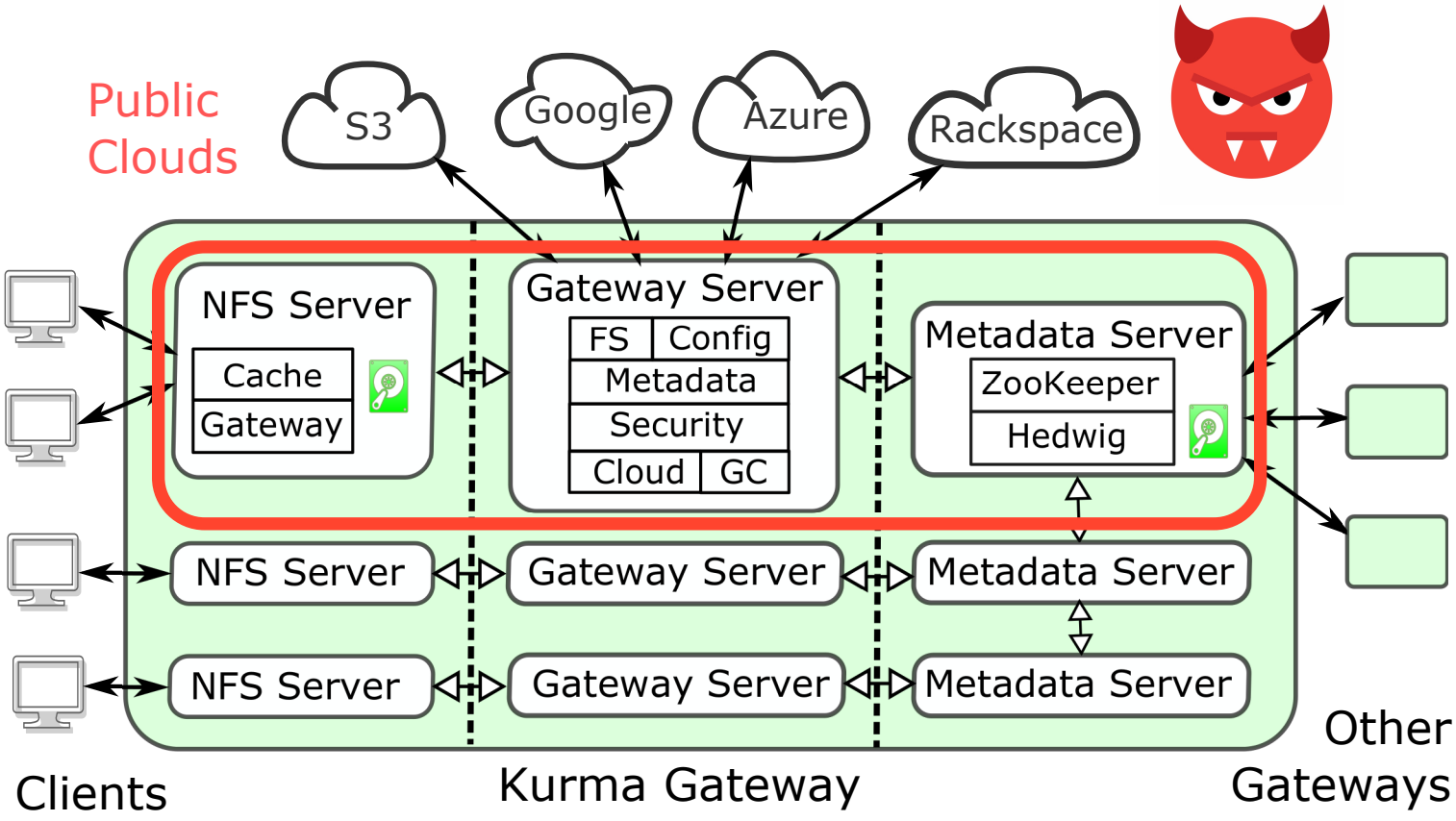
Stony Brook University

Components



Stony Brook University

Components



Stony Brook University

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                id;
   ObjectID                parent;
   Attributes              attrs;
   i32                     block_shift;
   list<i64>               block_versions;
   list<GatewayID>         block_creators;
   string                  redundancy;
   list<string>            cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                id;
   ObjectID                parent;
   Attributes              attrs;
   i32                     block_shift;
   list<i64>               block_versions;
   list<GatewayID>         block_creators;
   string                  redundancy;
   list<string>            cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef   i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                id;
   ObjectID                parent;
   Attributes              attrs;
   i32                     block_shift;
   list<i64>               block_versions;
   list<GatewayID>         block_creators;
   string                  redundancy;
   list<string>            cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                id;
   ObjectID                parent;
   Attributes              attrs;
   i32                     block_shift;
   list<i64>               block_versions;
   list<GatewayID>         block_creators;
   string                  redundancy;
   list<string>            cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                  id;
   ObjectID                  parent;
   Attributes                attrs;
   i32                       block_shift;
   list<i64>                 block_versions;
   list<GatewayID>           block_creators;
   string                    redundancy;
   list<string>              cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                id;
   ObjectID                parent;
   Attributes              attrs;
   i32                     block_shift;
   list<i64>               block_versions;
   list<GatewayID>         block_creators;
   string                  redundancy;
   list<string>            cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID                id;
   ObjectID                parent;
   Attributes              attrs;
   i32                     block_shift;
   list<i64>               block_versions;
   list<GatewayID>         block_creators;
   string                  redundancy;
   list<string>            cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Metadata Management

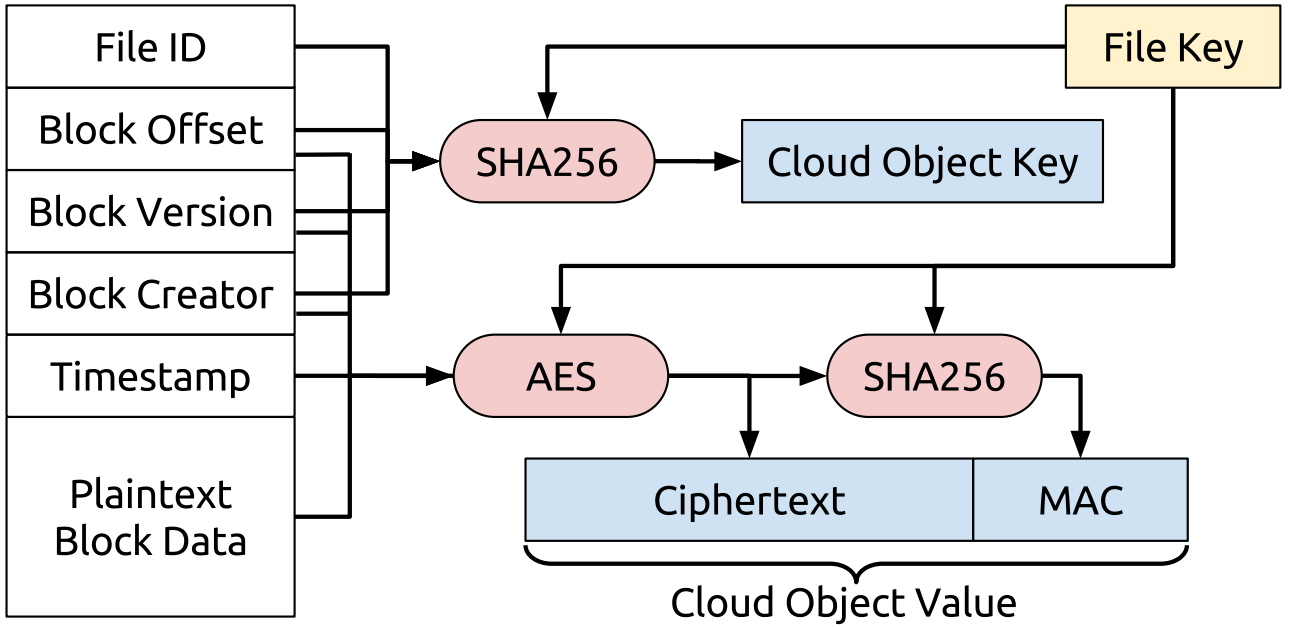
- Defined using Thrift
- Stored in ZooKeeper
- Replicated cross-regions using Hedwig

```
typedef  i16  GatewayID
struct ObjectID {
   i128              id;
   GatewayID         creator;
   byte              type;
}
struct Attributes {
   i64               size;
   i32               flags;
        ... ...
   i64       remote_ctime;
}
```

```
struct File {
   ObjectID              id;
   ObjectID              parent;
   Attributes            attrs;
   i32                   block_shift;
   list<i64>             block_versions;
   list<GatewayID>       block_creators;
   string                redundancy;
   list<string>          cloud_ids;
   map<GatewayID, binary> keymap;
}
```

Kurma Security

- Only file data blocks are saved in clouds
- Blocks are authenticated and encrypted
- Per-file secret key protected by gateway master keys
- Detect swap and replay attacks



Stony Brook University

Multi-Cloud Redundancy

	Replication	Erasure Coding	Secret Sharing
Parameters (e.g., 4 clouds)	$n=4$	$k=3, m=1$	$n=4, k=3, r=2$
Apply to a block	n identical 1MB blocks	$k+m$ non-identical $1/k$ MB block	n non-identical $1/k$ MB block
Write a block	$n \times 1\text{MB}$	$(k+m) \times 1/k\,\text{MB}$	$(k+m) \times 1/k\,\text{MB}$
Read a block	any 1 cloud	any k clouds	any k clouds
Tolerate failure of clouds	$n=f+1$	$m=f$	$n-k=f$
Write amplifications	$f+1$	$(f+1)/k$	$(f+1)/k$
Example	$2 \times 1\text{MB}$ blocks	$4 \times 340\text{KB}$ blocks	$4 \times 340\text{KB}$ blocks

Hybrid Consistency Model

- FIFO consistency across gateways
  - Updates made by a single gateway are seen by other gateways in the order they occur, but updates from different gateways may be seen in any interleaved order
  - FS metadata is asynchronously replicated among all regions using Hedwig which does not order message across gateways
  - Resolves inter-gateway conflicts as needed
- Region-level NFS consistency
  - Same as traditional NFS
  - Data freshness in the same region

Implementation

- NFS Servers built on top of NFS-Ganesha
  - FSAL_PCACHE
  - FSAL_KURMA
- Gateway Servers
  - File-System Module uses Thrift
  - Metadata Module uses Apache Curator (ZooKeeper)
  - Security Module uses Java 8 standard cryptographic library
  - Cloud Module uses cloud Java drivers
  - Redundancy uses Jerasure and CAONS-RS secret sharing

Components	Language	LoC
Kurma NFS Server	C/C++	15,802
Kurma Gateway Server	Java	27,976
Secret Sharing JNI	C/C++	2,480
RPC & Metadata Definition	Thrift	668

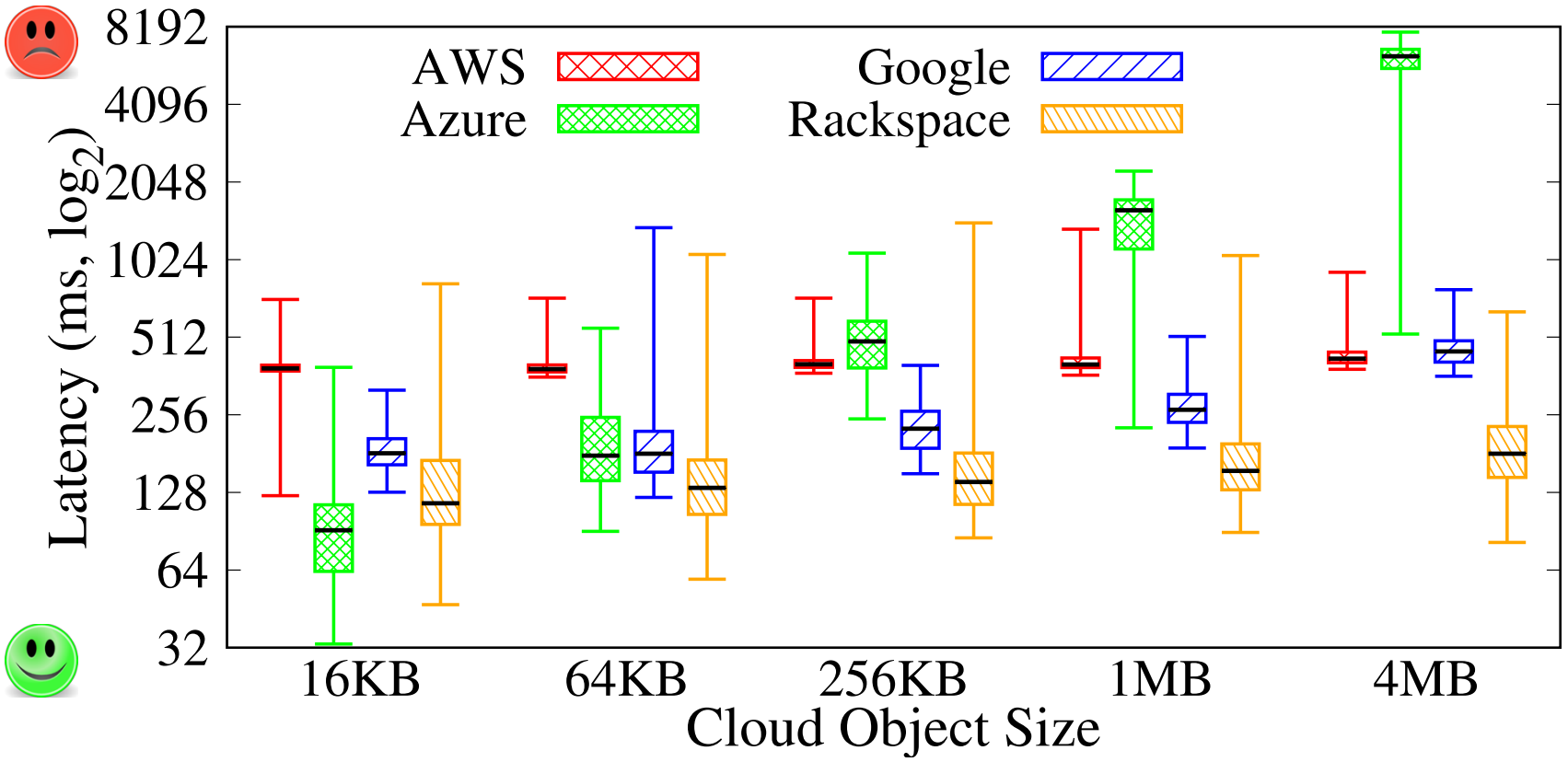
Optimizations

1. Avoid high-latency of ZooKeeper
   - Batch metadata changes using transactions
   - Use in-memory cache for hot znodes
2. Avoid performance variations of clouds
   - Sort clouds online every *N* seconds
3. Reduce metadata size
   - Compress file-system metadata
   - Use large block sizes

Evaluation

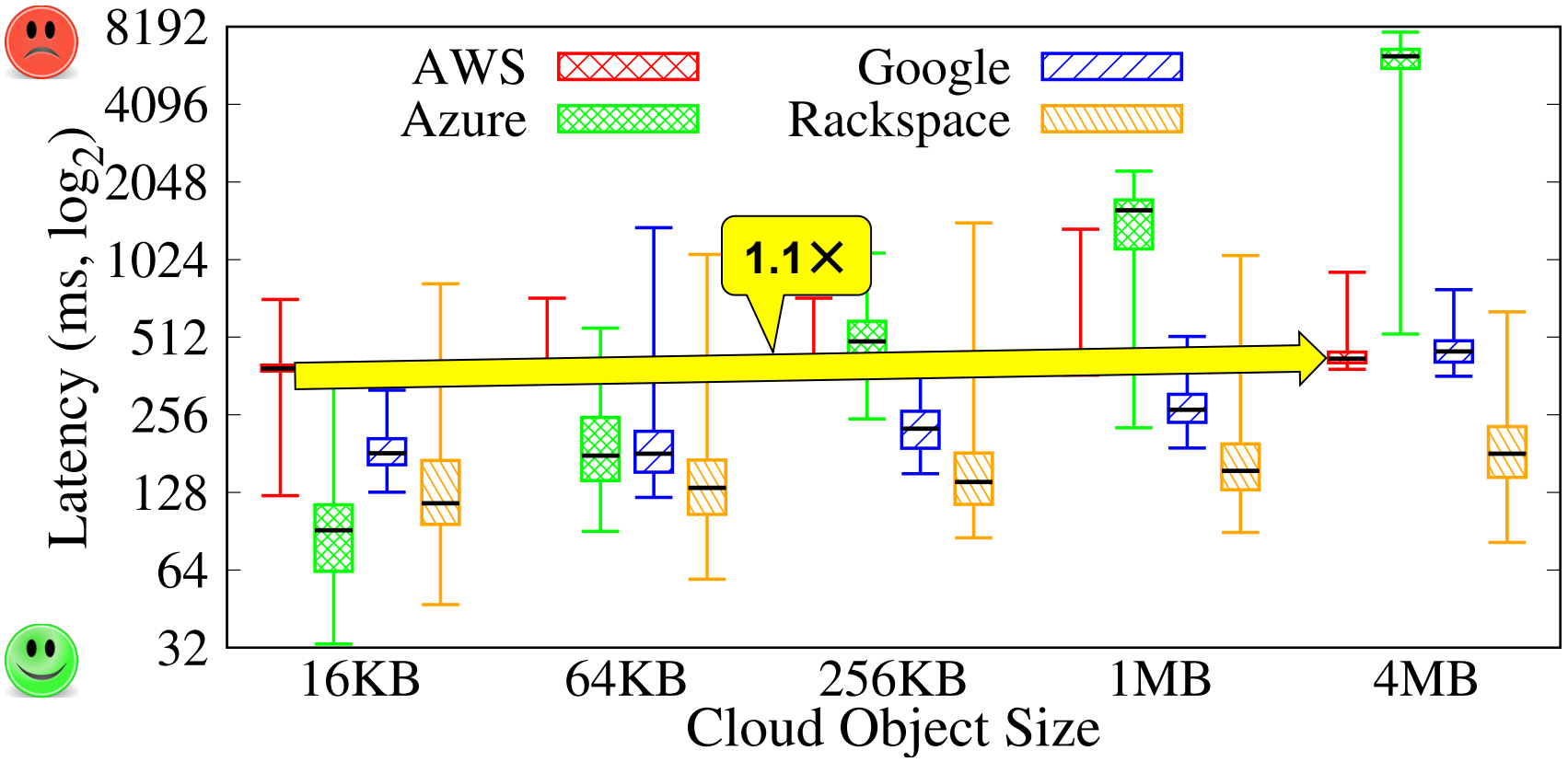
- Experimental setup
  - Two regions with a network RTT of 100ms
  - Each region contains VMs for
    - 3 Metadata Servers running ZooKeeper and Hedwig
    - 1 Gateway Server
    - 1 NFS Server with persistent cache on an Intel SSD
    - 1 NFS client
    - Each VM has two cores and 4GB of RAM running Fedora 25 with Linux 4.8.10 kernel
  - Baseline: traditional NFS server
    - Runs NFS-Ganesha FSAL_VFS
    - Uses an Intel SSD formatted with Ext4
- Security tests
  - Availability test
  - Integrity tests: swapping and replay attacks

Cloud Read Latency



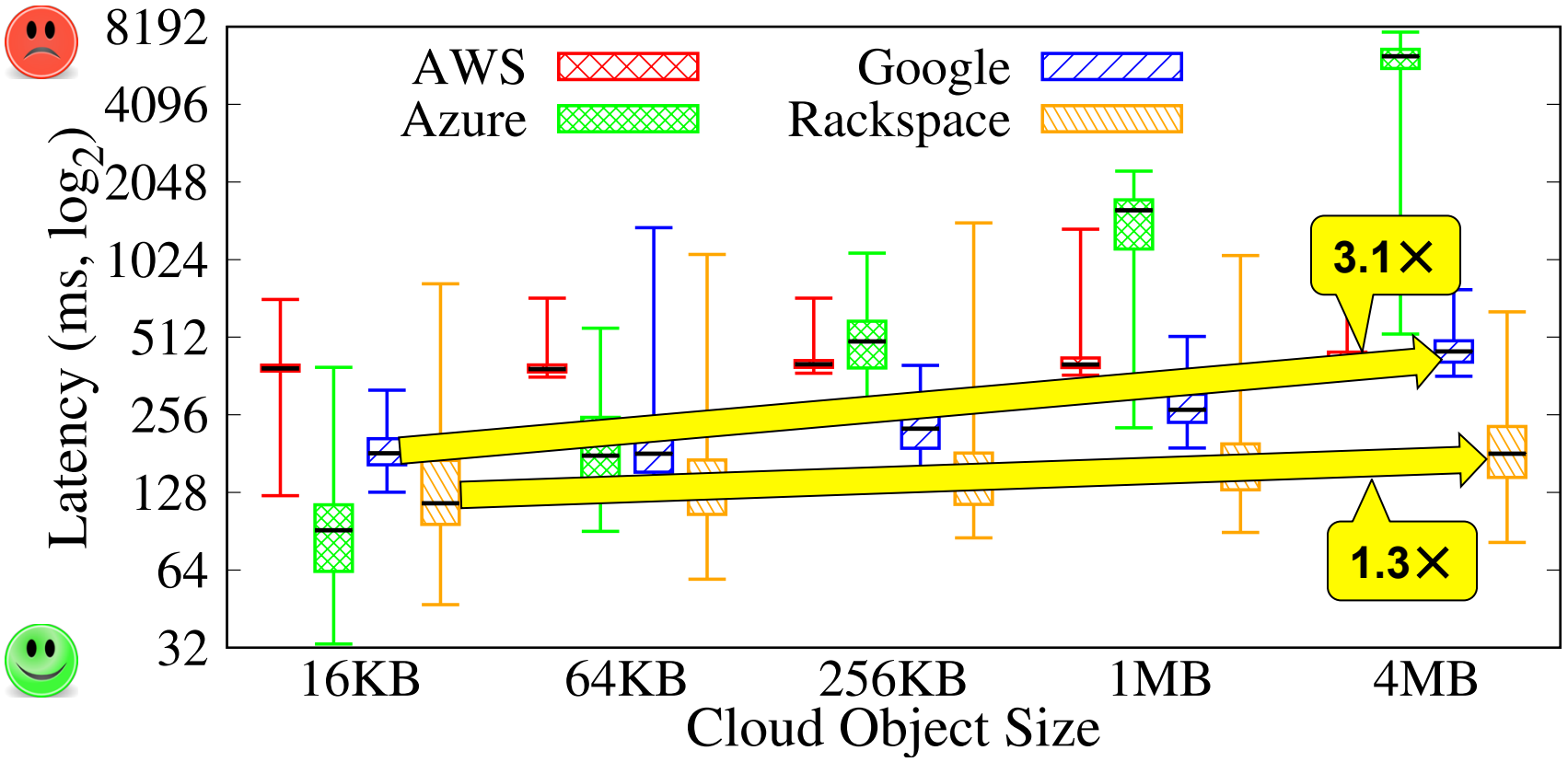
Stony Brook University

Cloud Read Latency



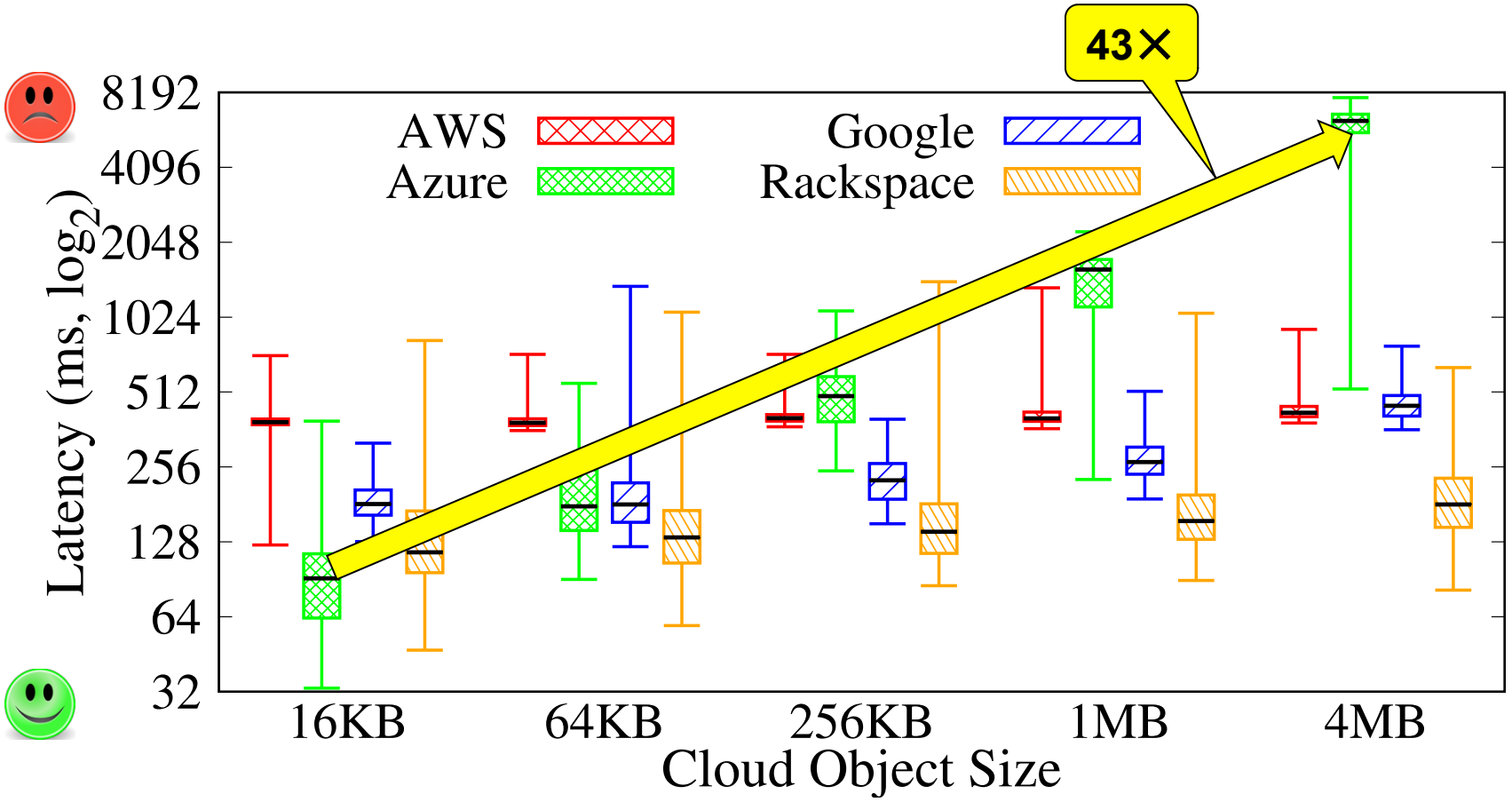
Stony Brook University

Cloud Read Latency



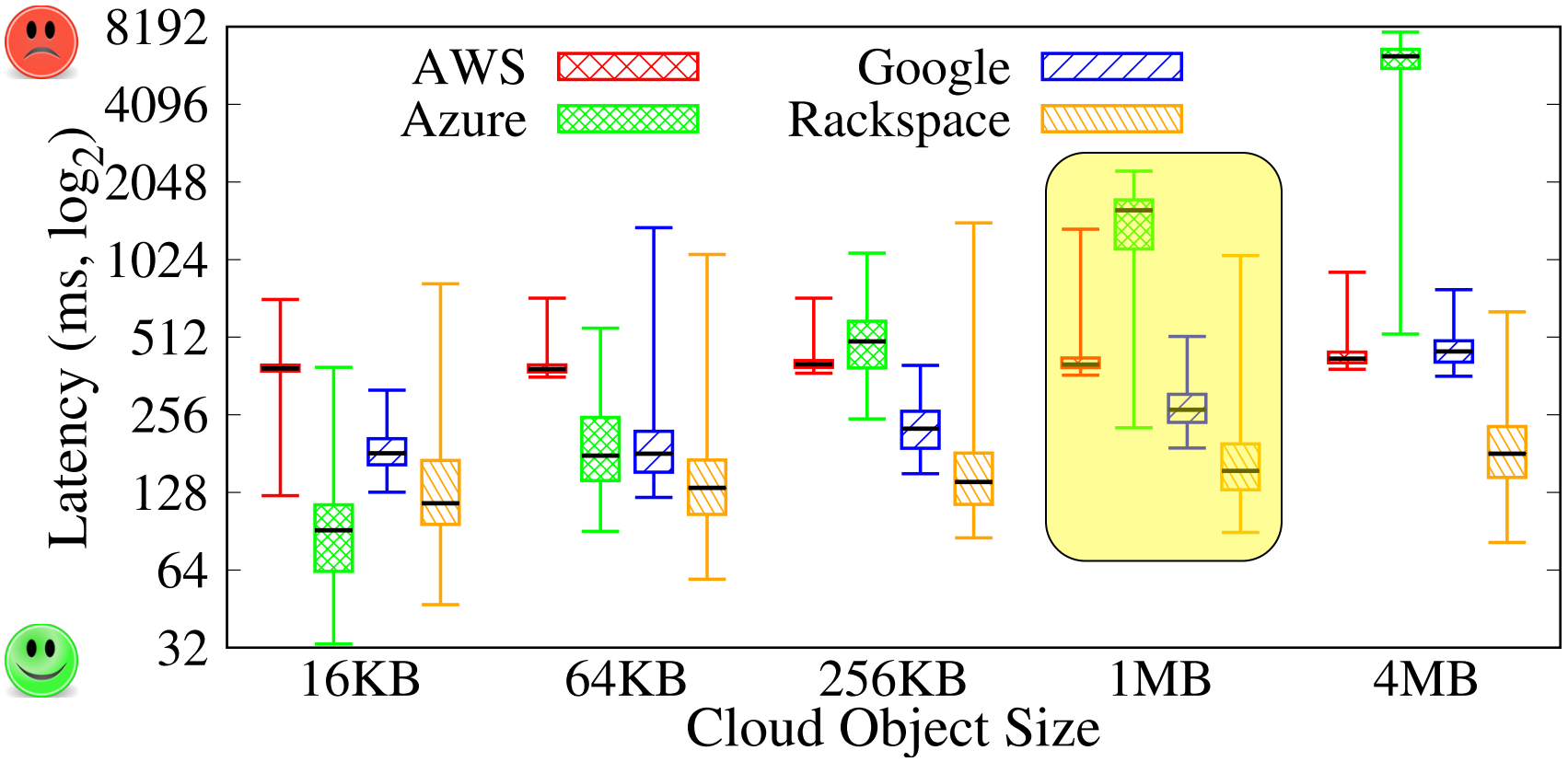
Stony Brook University

Cloud Read Latency



Stony Brook University

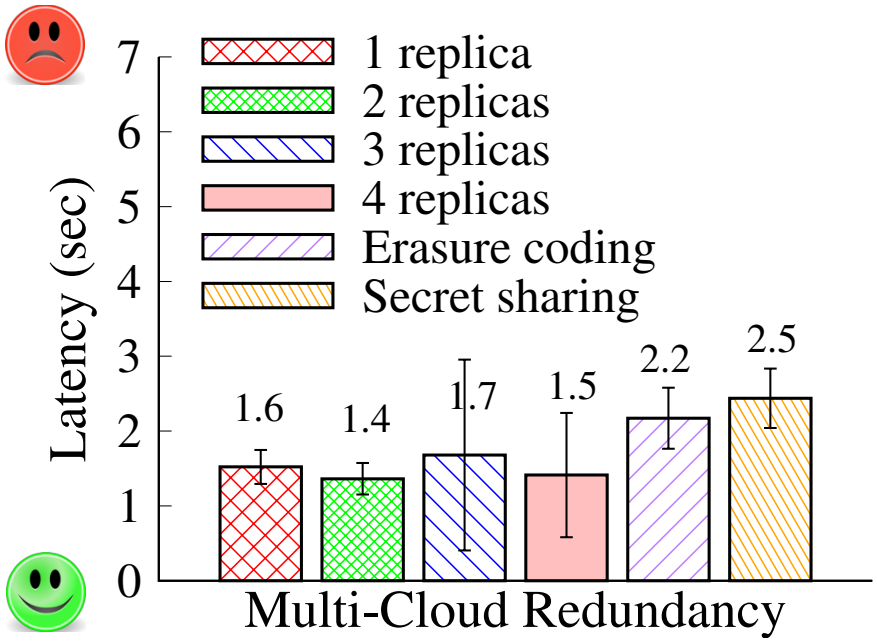
Cloud Read Latency



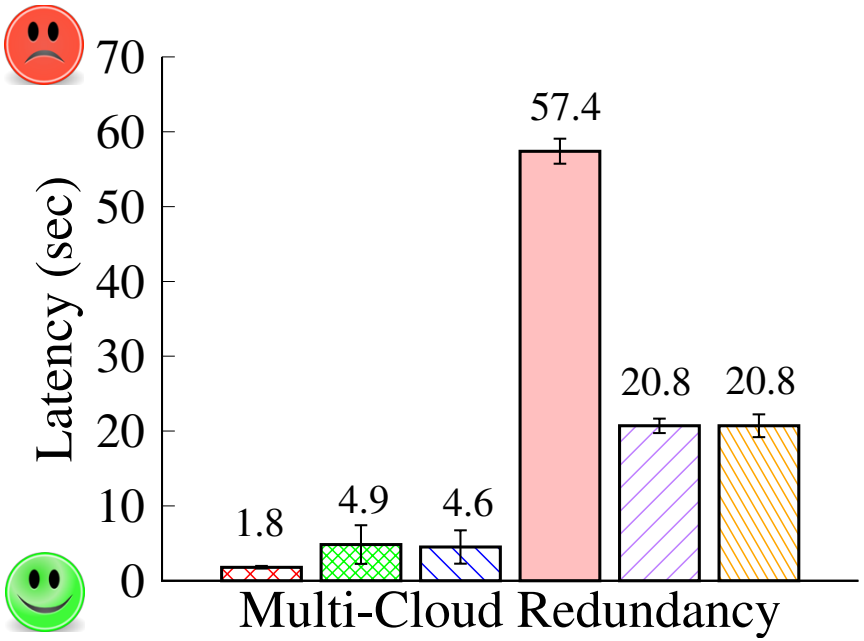
Stony Brook University

Multi-Cloud Redundancy

- N-replica: Save N identical replicas in N clouds
- Erasure coding: Reed-Solomon (*k = 3, m = 1*)
- Secret sharing: CAONS-RS (*n = 4, k =3, r = 2*)



(a) Read a 16MB file

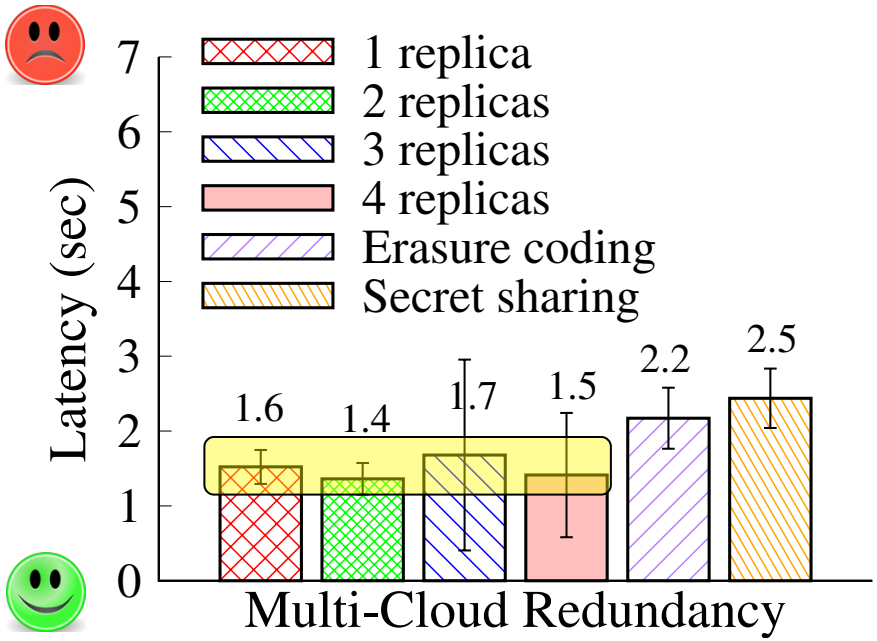


(b) Write a 16MB file

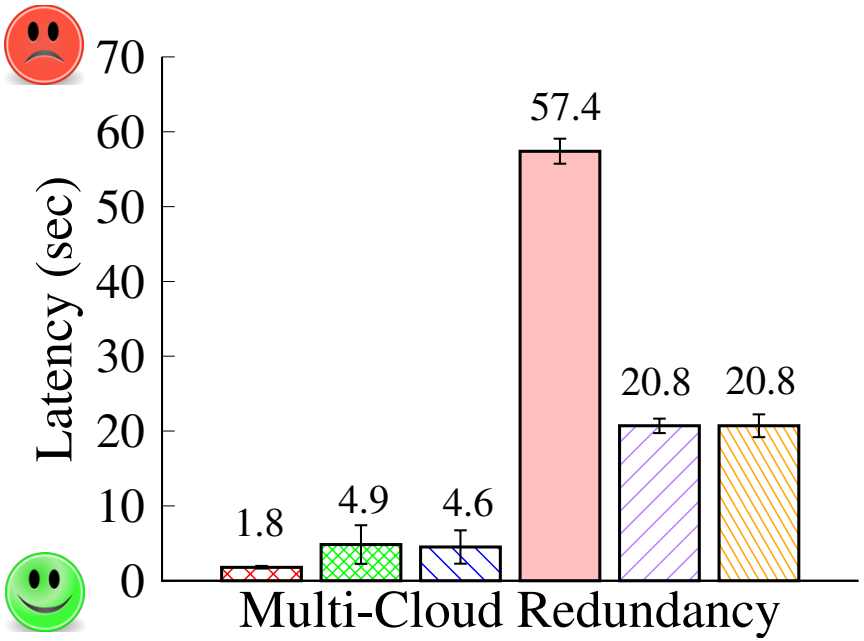
Stony Brook University

Multi-Cloud Redundancy

- N-replica: Save N identical replicas in N clouds
- Erasure coding: Reed-Solomon (*k = 3, m = 1*)
- Secret sharing: CAONS-RS (*n = 4, k =3, r = 2*)



(a) Read a 16MB file

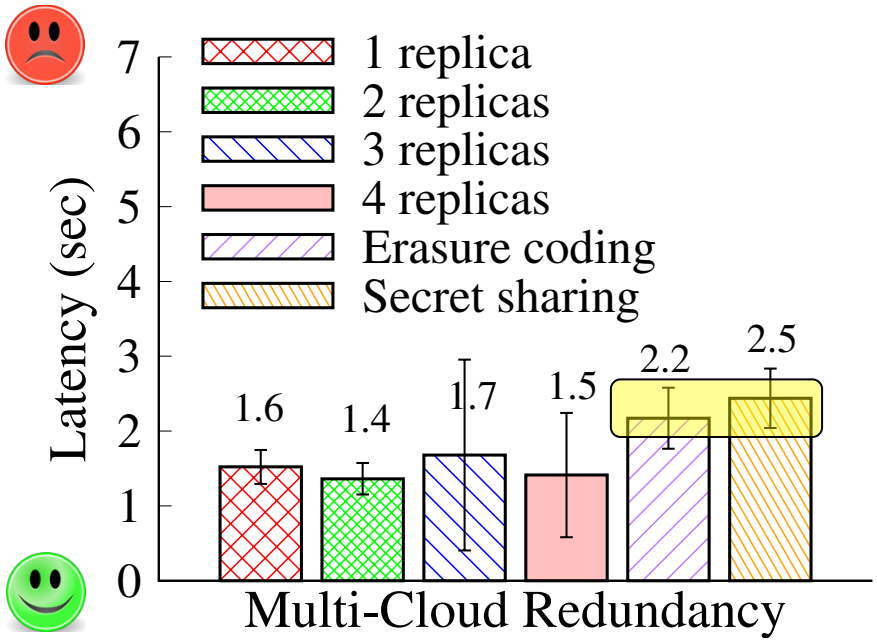


(b) Write a 16MB file

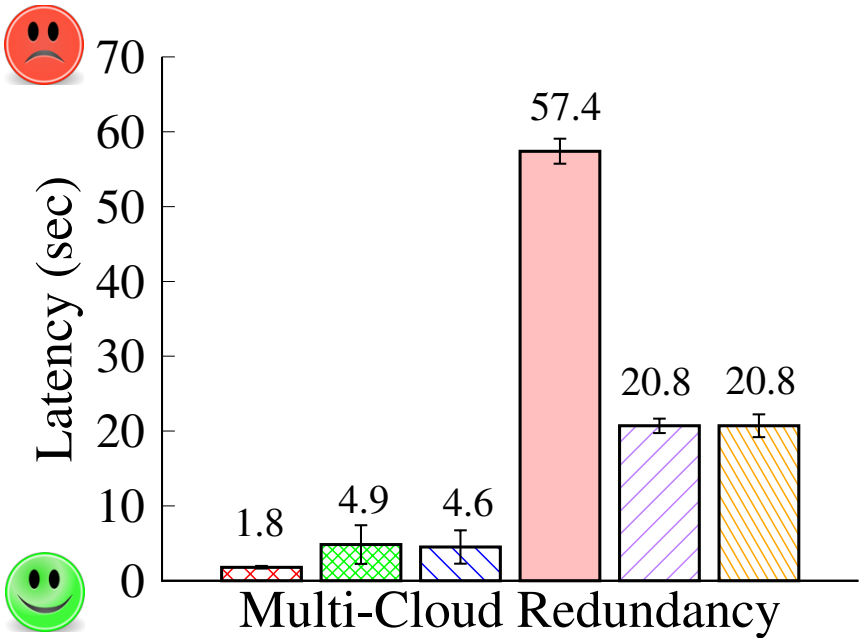
Stony Brook University

Multi-Cloud Redundancy

- N-replica: Save N identical replicas in N clouds
- Erasure coding: Reed-Solomon (*k = 3, m = 1*)
- Secret sharing: CAONS-RS (*n = 4, k =3, r = 2*)



(a) Read a 16MB file

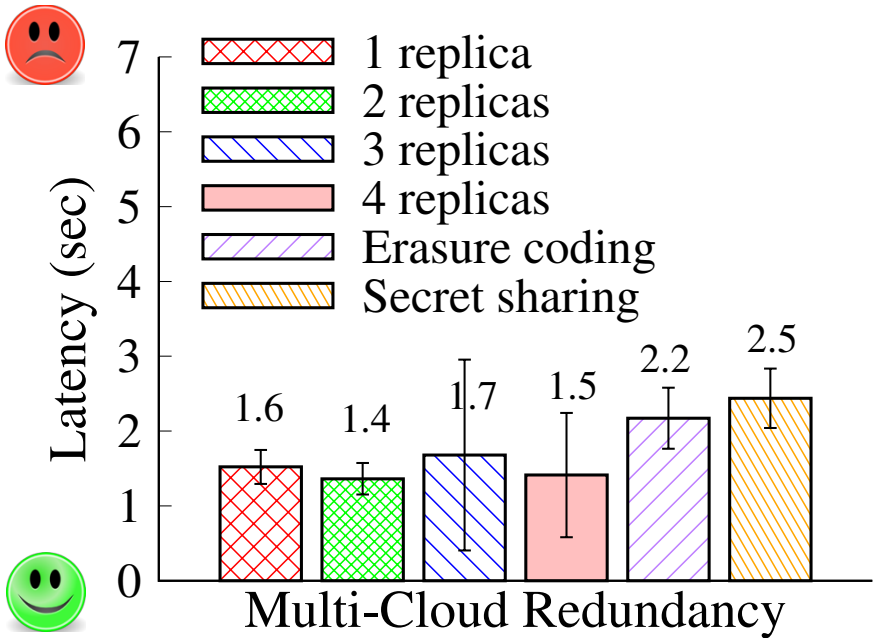


(b) Write a 16MB file

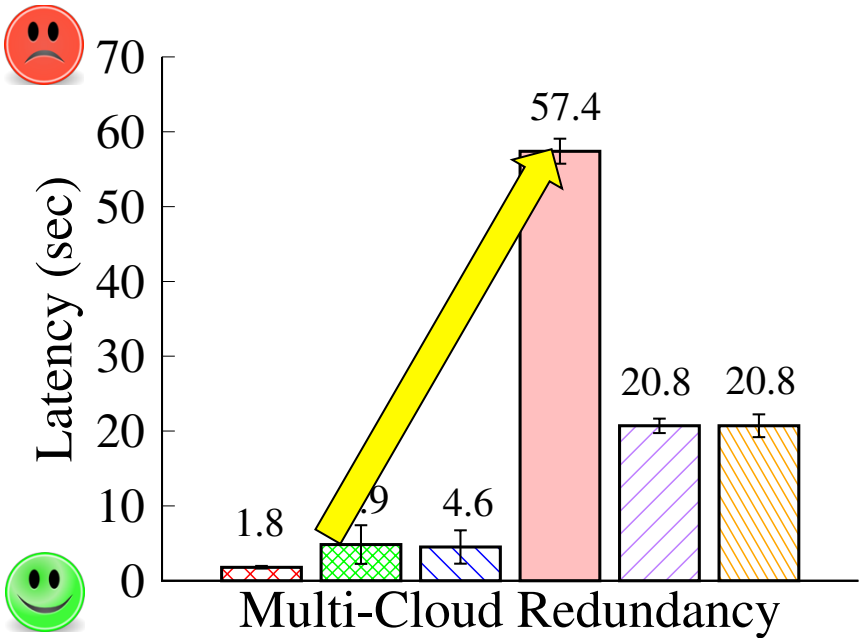
Stony Brook University

Multi-Cloud Redundancy

- N-replica: Save N identical replicas in N clouds
- Erasure coding: Reed-Solomon (*k = 3, m = 1*)
- Secret sharing: CAONS-RS (*n = 4, k =3, r = 2*)



(a) Read a 16MB file

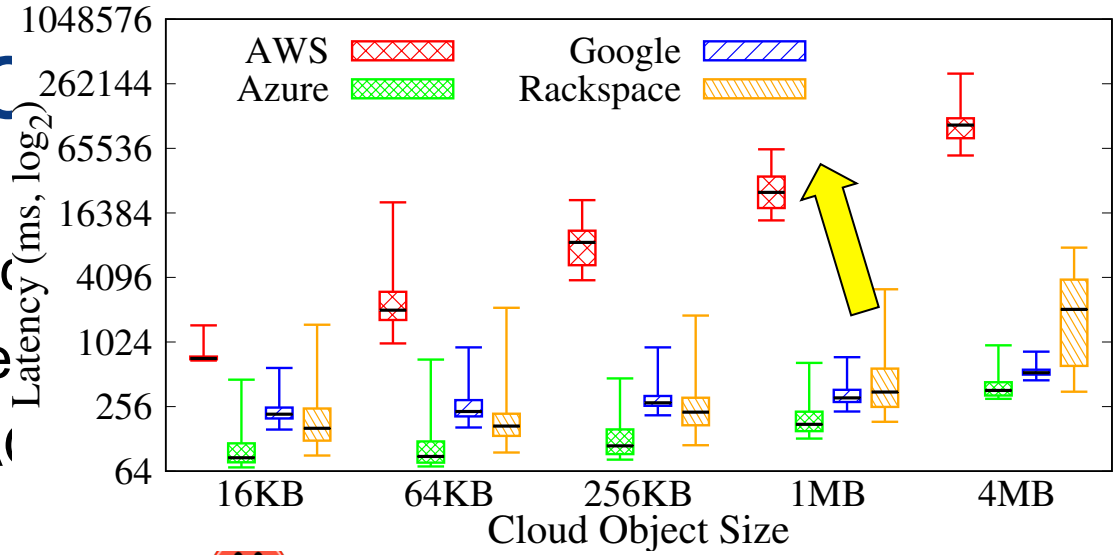


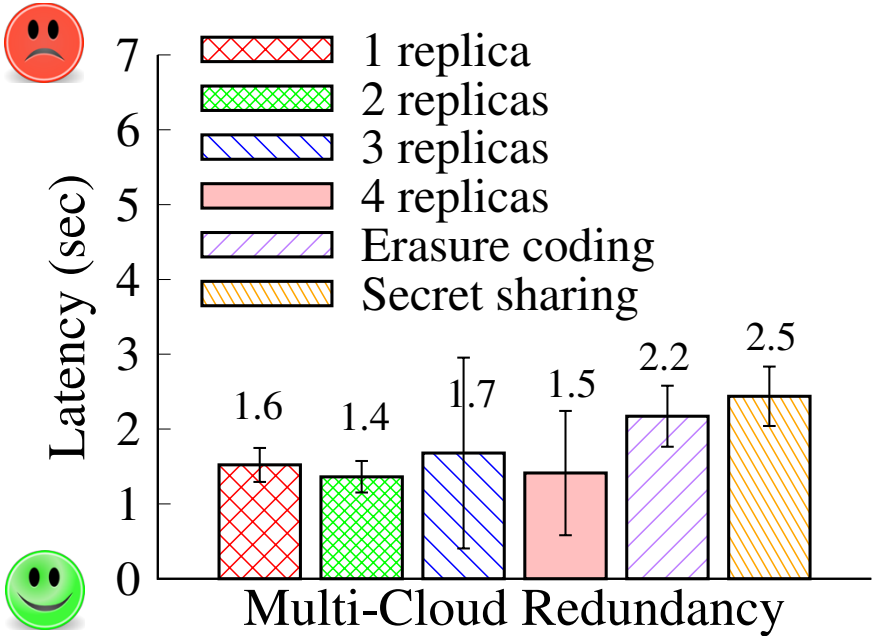
(b) Write a 16MB file

Stony Brook University

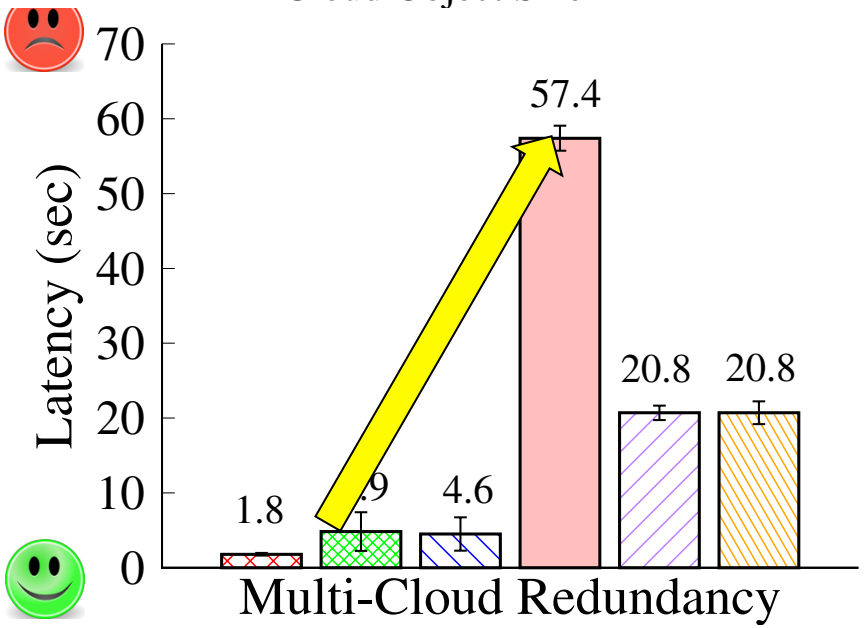
Multi-Clo

- N-replica: Save N i
- Erasure coding: Re
- Secret sharing: CA





(a) Read a 16MB file

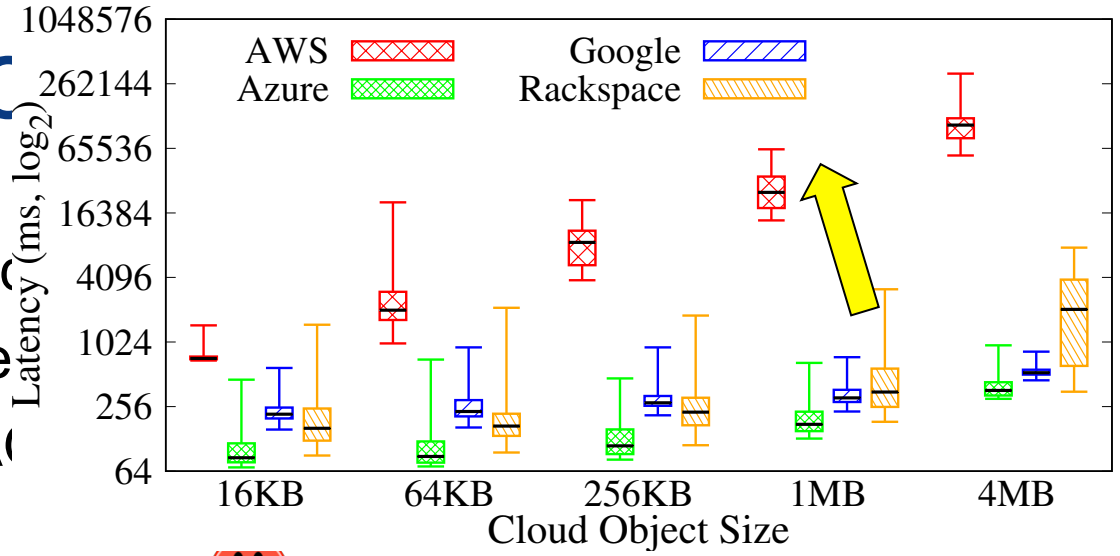


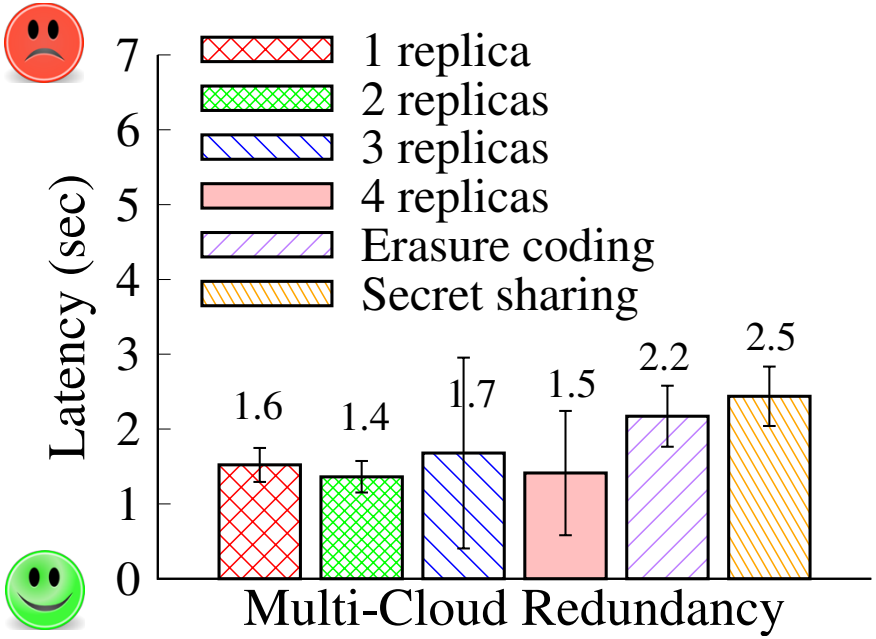
(b) Write a 16MB file

Stony Brook University

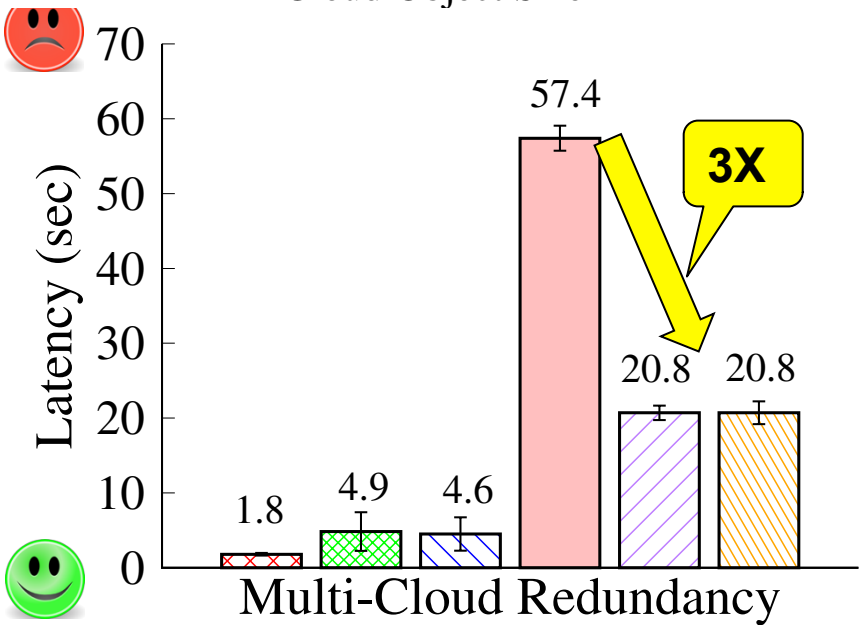
Multi-Clo

- N-replica: Save N i
- Erasure coding: Re
- Secret sharing: CA





(a) Read a 16MB file

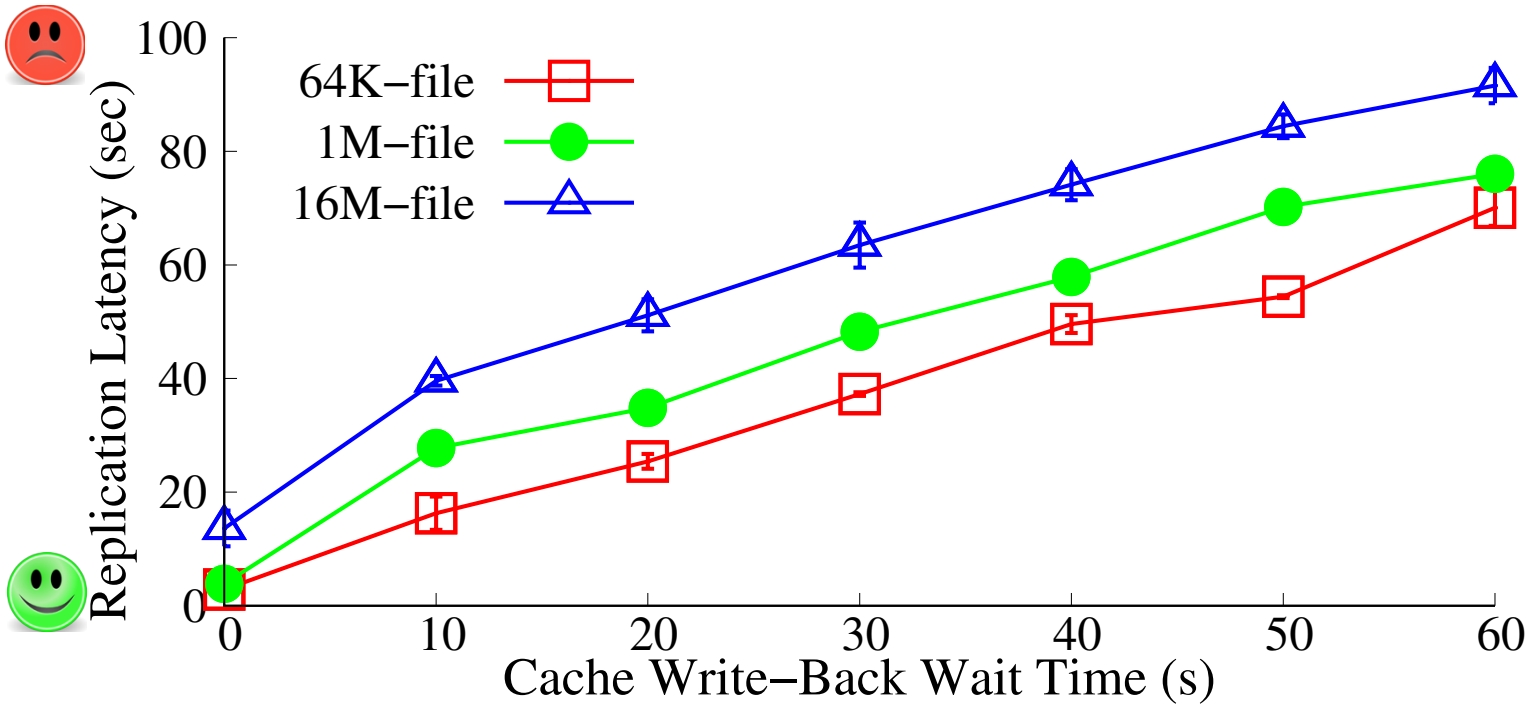


(b) Write a 16MB file

Stony Brook University

Cross-Gateway Replication

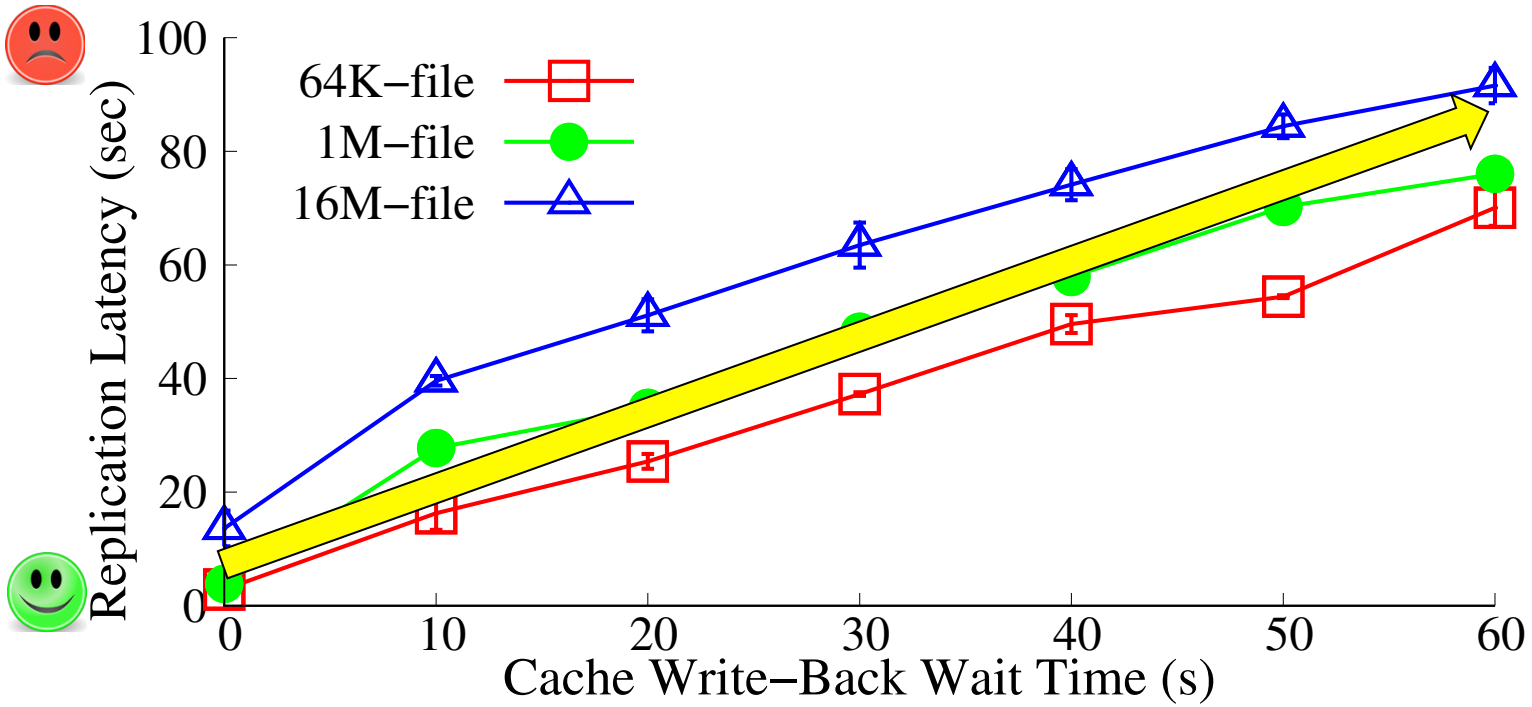
- Create a file in one gateway and read it in another gateway



Stony Brook University

Cross-Gateway Replication

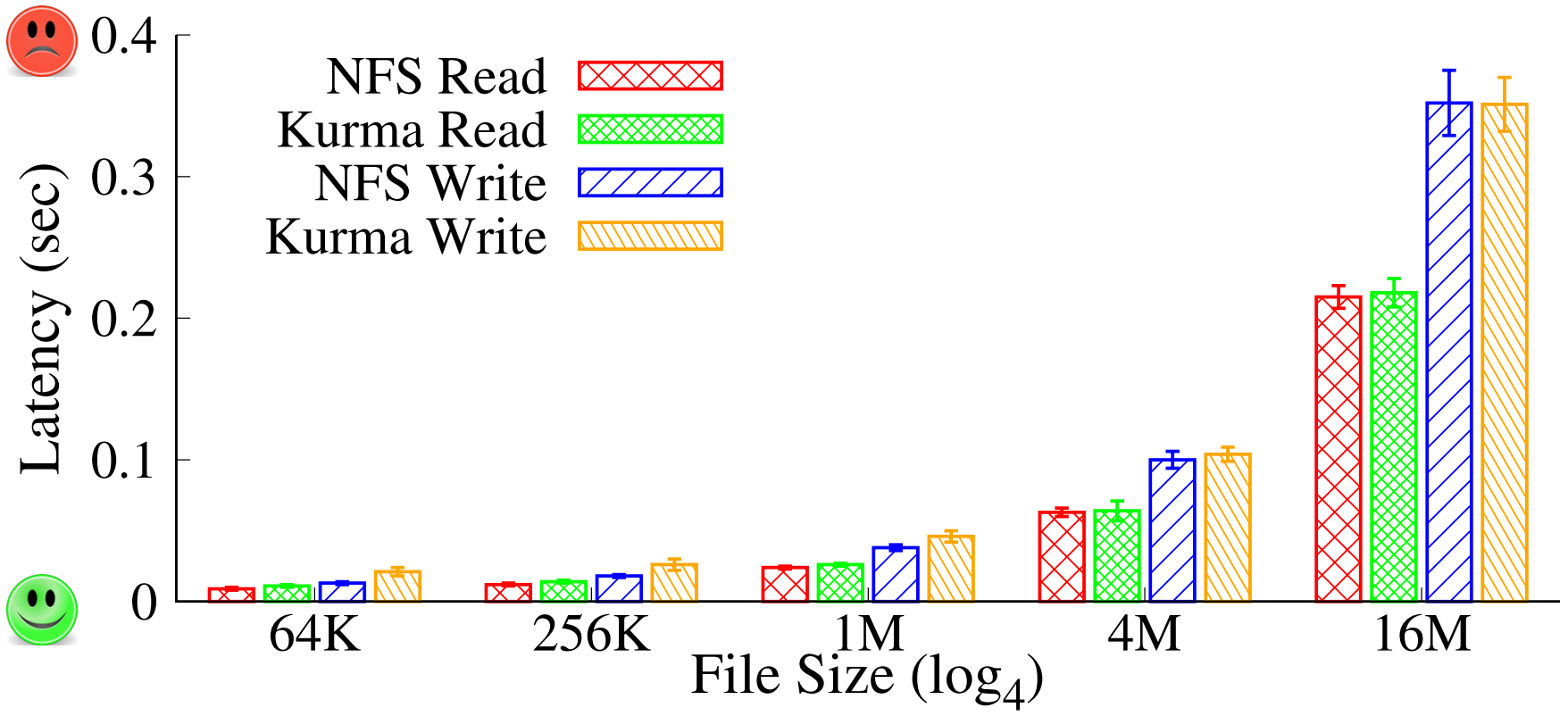
- Create a file in one gateway and read it in another gateway



Stony Brook University

Data Operations

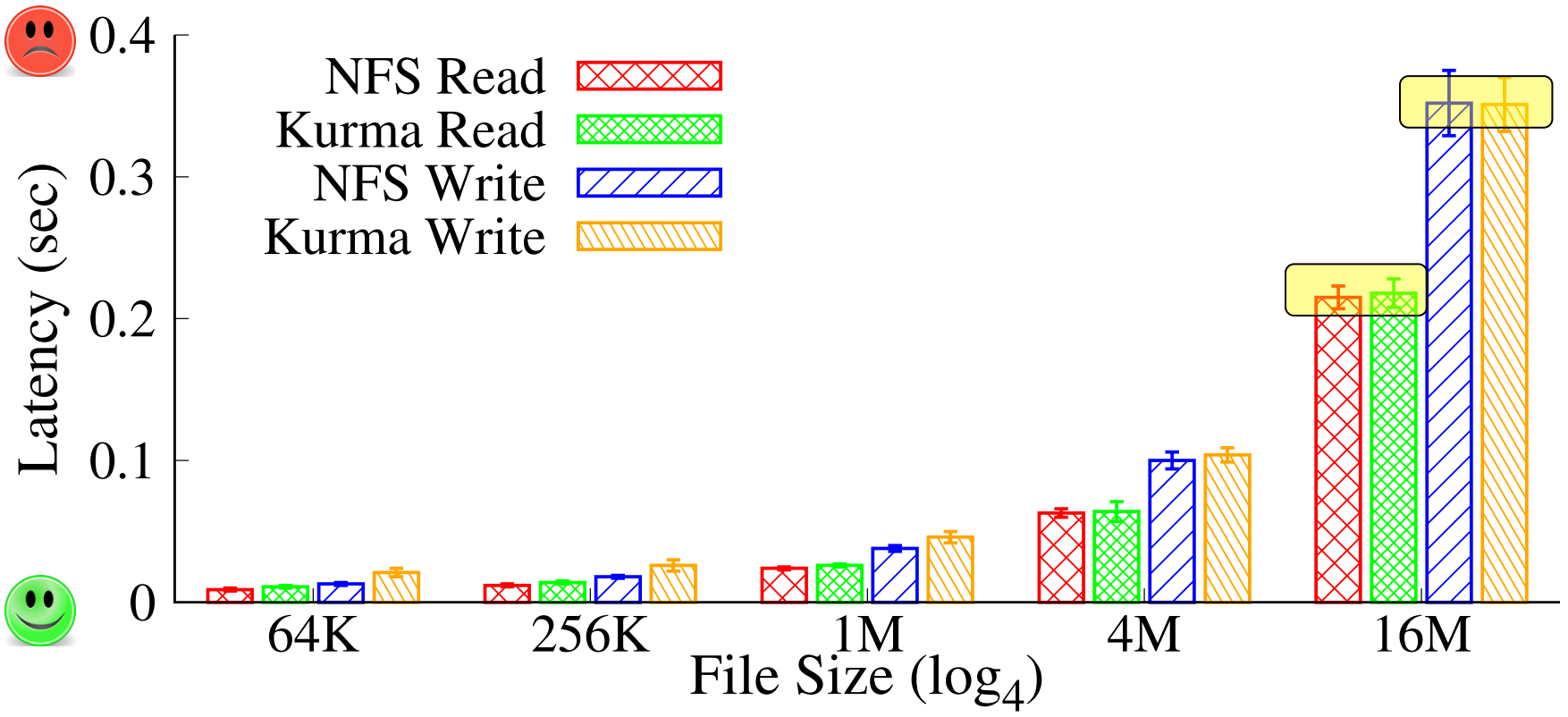
- Read and write files with a hot cache



Stony Brook University

Data Operations

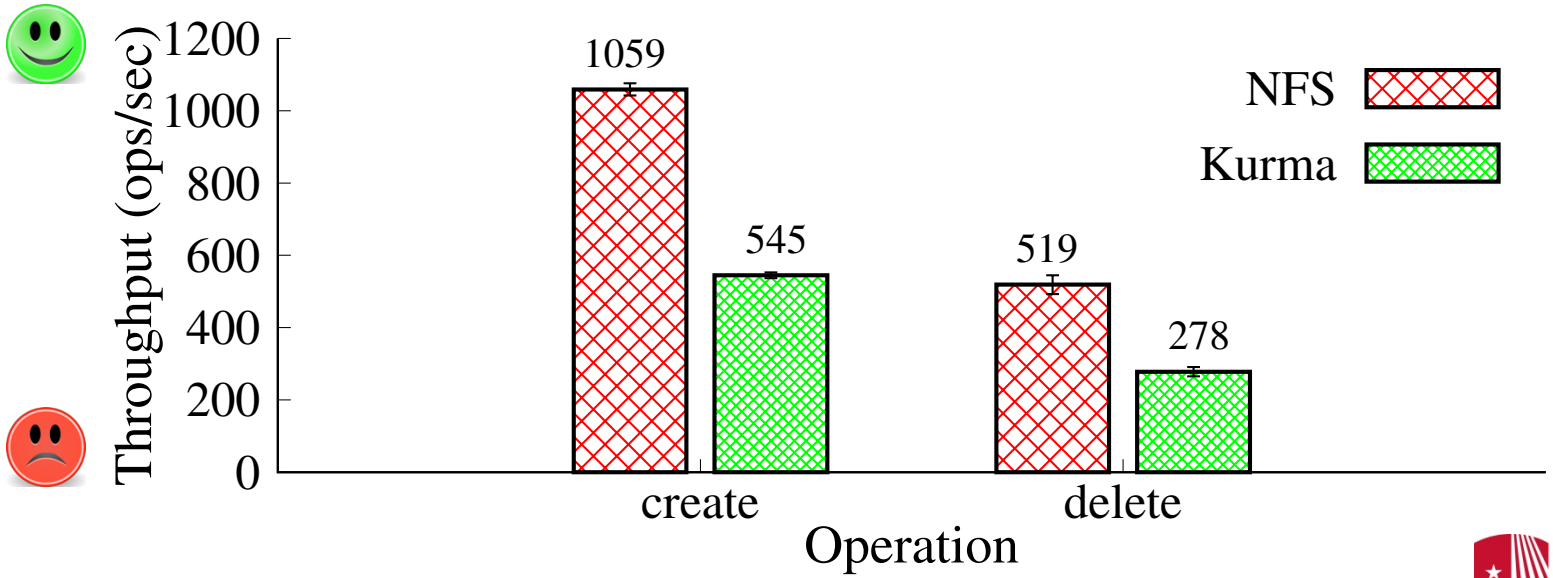
- Read and write files with a hot cache



Stony Brook University

Metadata Operations

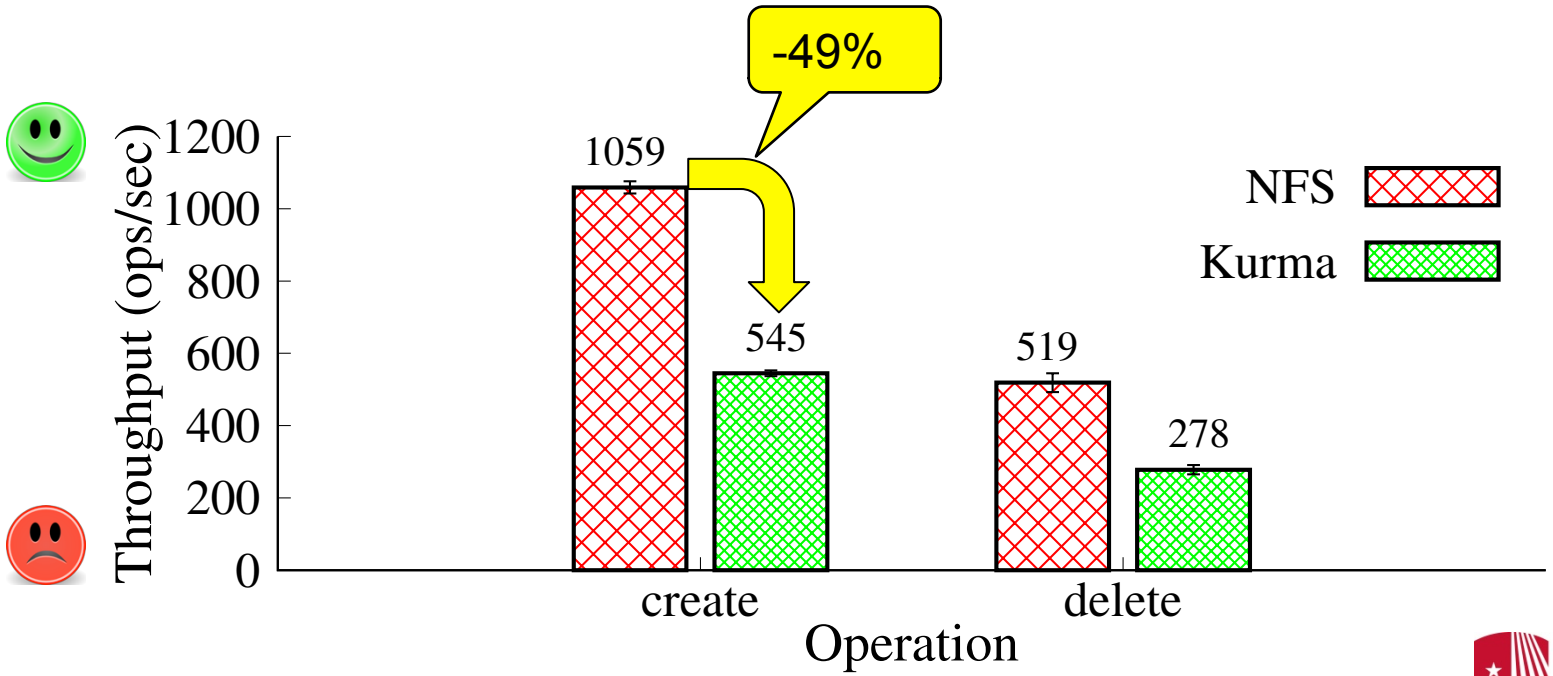
- Kurma metadata operations are slower than the baseline
  - A metadata operation requires changes to multiple ZooKeeper nodes (znodes). For example, create a file:
    1. create file znode
    2. create keymap znode
    3. update parent directory's znode
  - Each ZooKeeper change incurs multiple network hops.



Stony Brook University

Metadata Operations

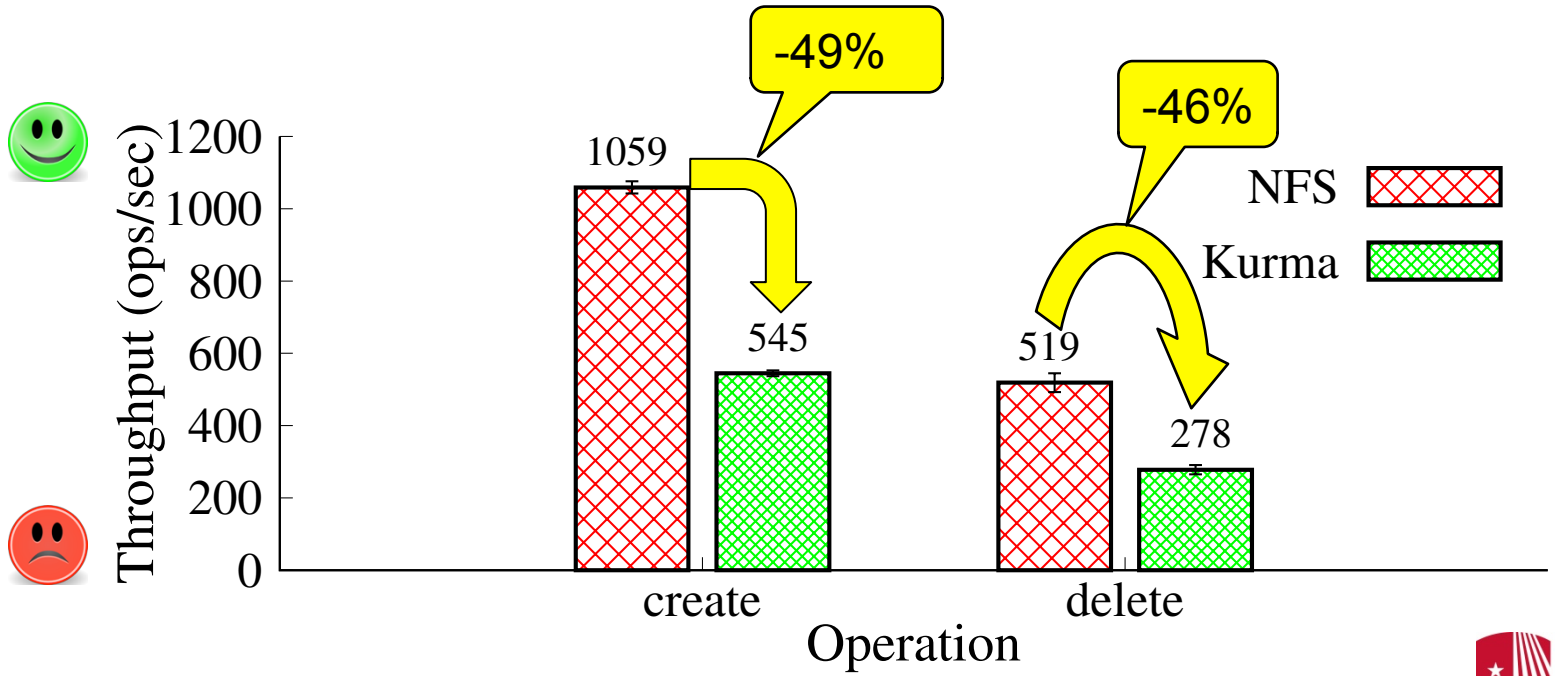
- Kurma metadata operations are slower than the baseline
  - A metadata operation requires changes to multiple ZooKeeper nodes (znodes). For example, create a file:
    1. create file znode
    2. create keymap znode
    3. update parent directory's znode
  - Each ZooKeeper change incurs multiple network hops.



Stony Brook University

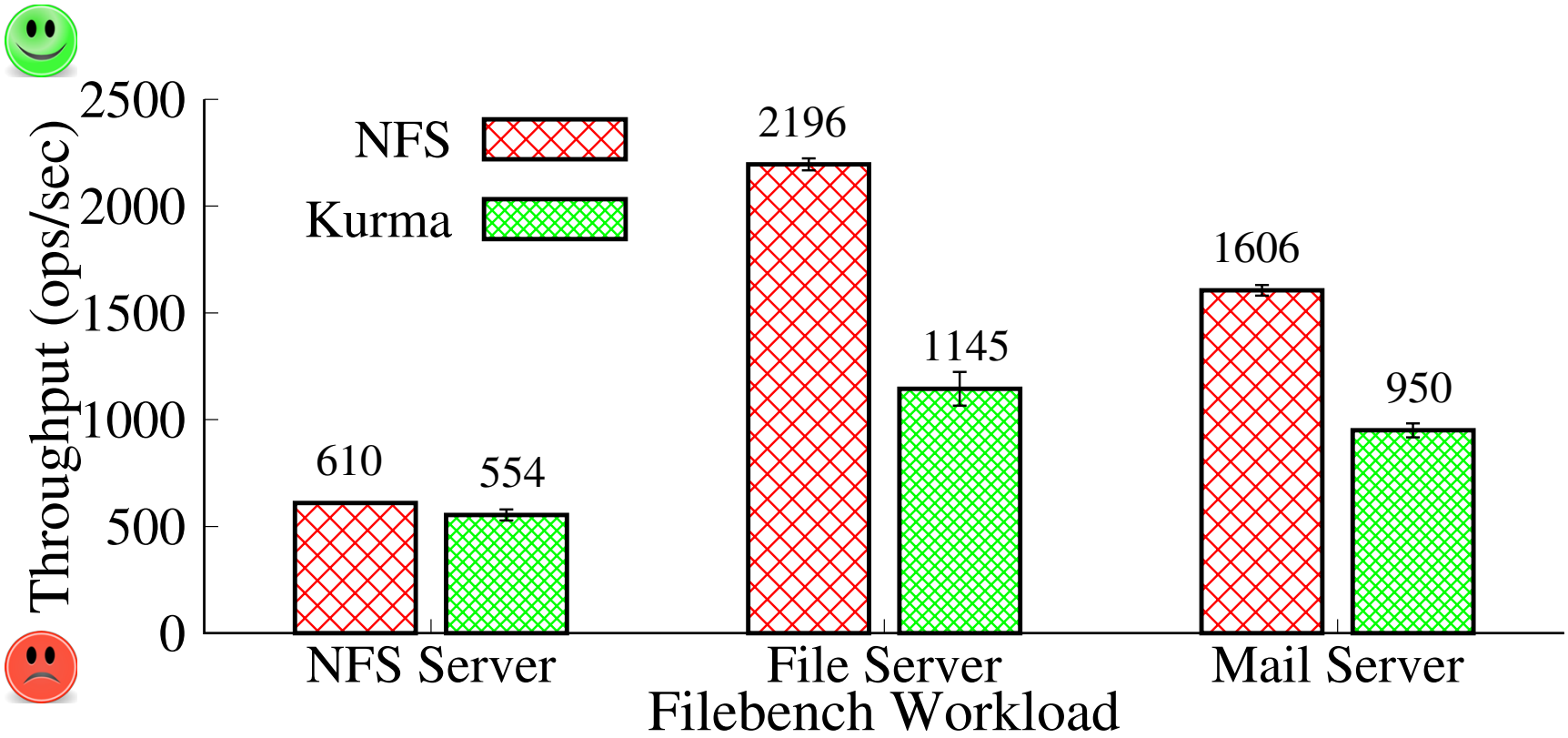
Metadata Operations

- Kurma metadata operations are slower than the baseline
  - A metadata operation requires changes to multiple ZooKeeper nodes (znodes). For example, create a file:
    1. create file znode
    2. create keymap znode
    3. update parent directory's znode
  - Each ZooKeeper change incurs multiple network hops.



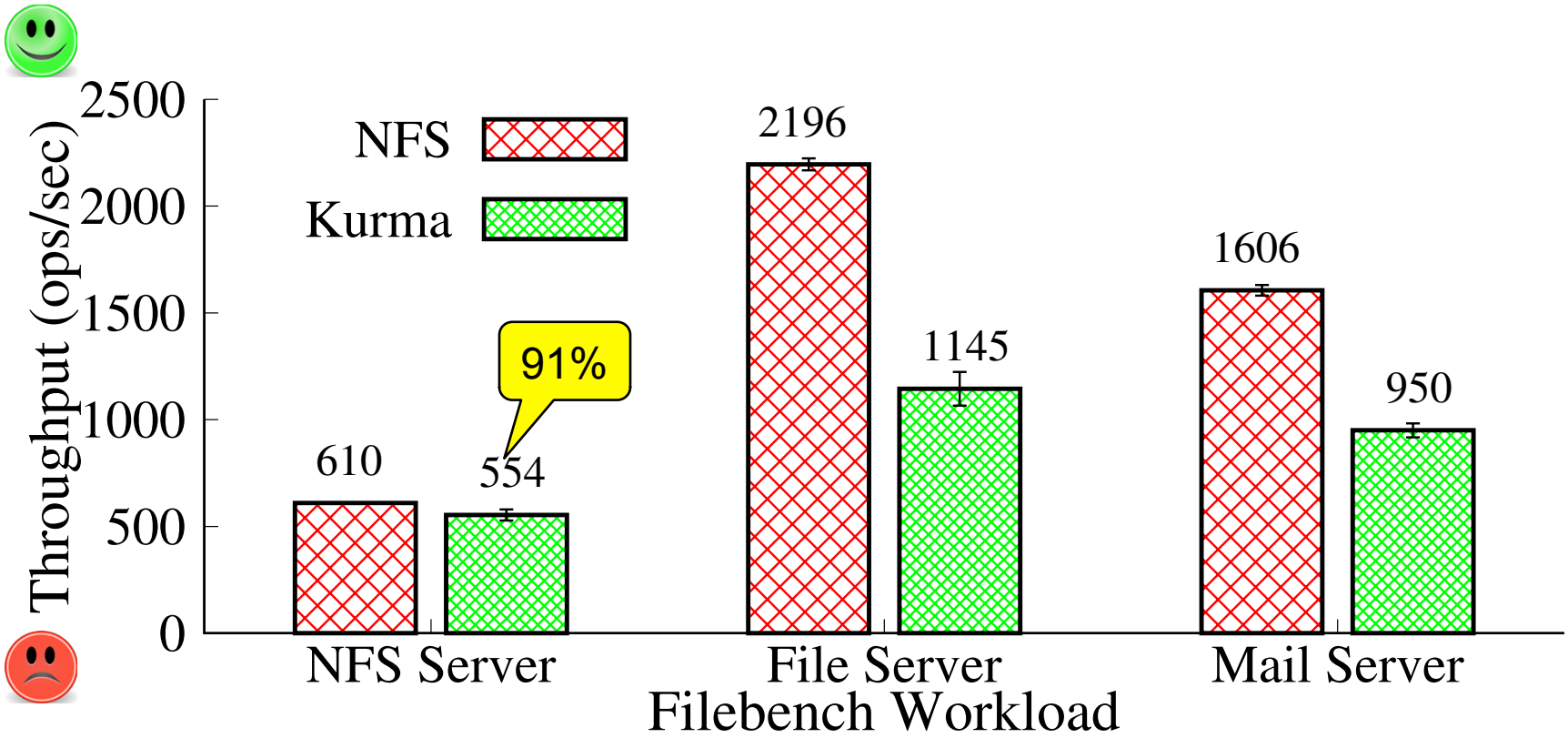
Stony Brook University

Filebench Workloads



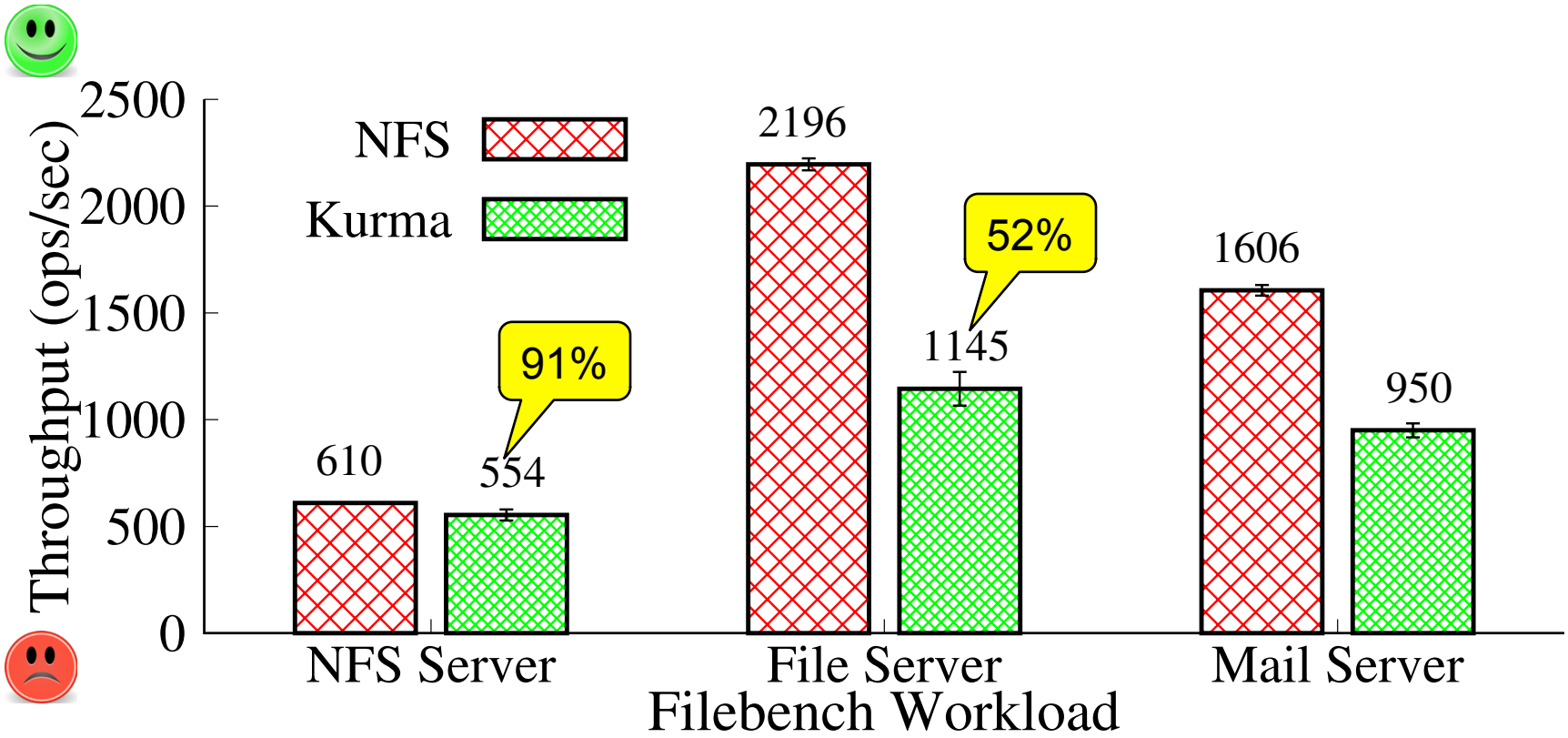
Stony Brook University

Filebench Workloads



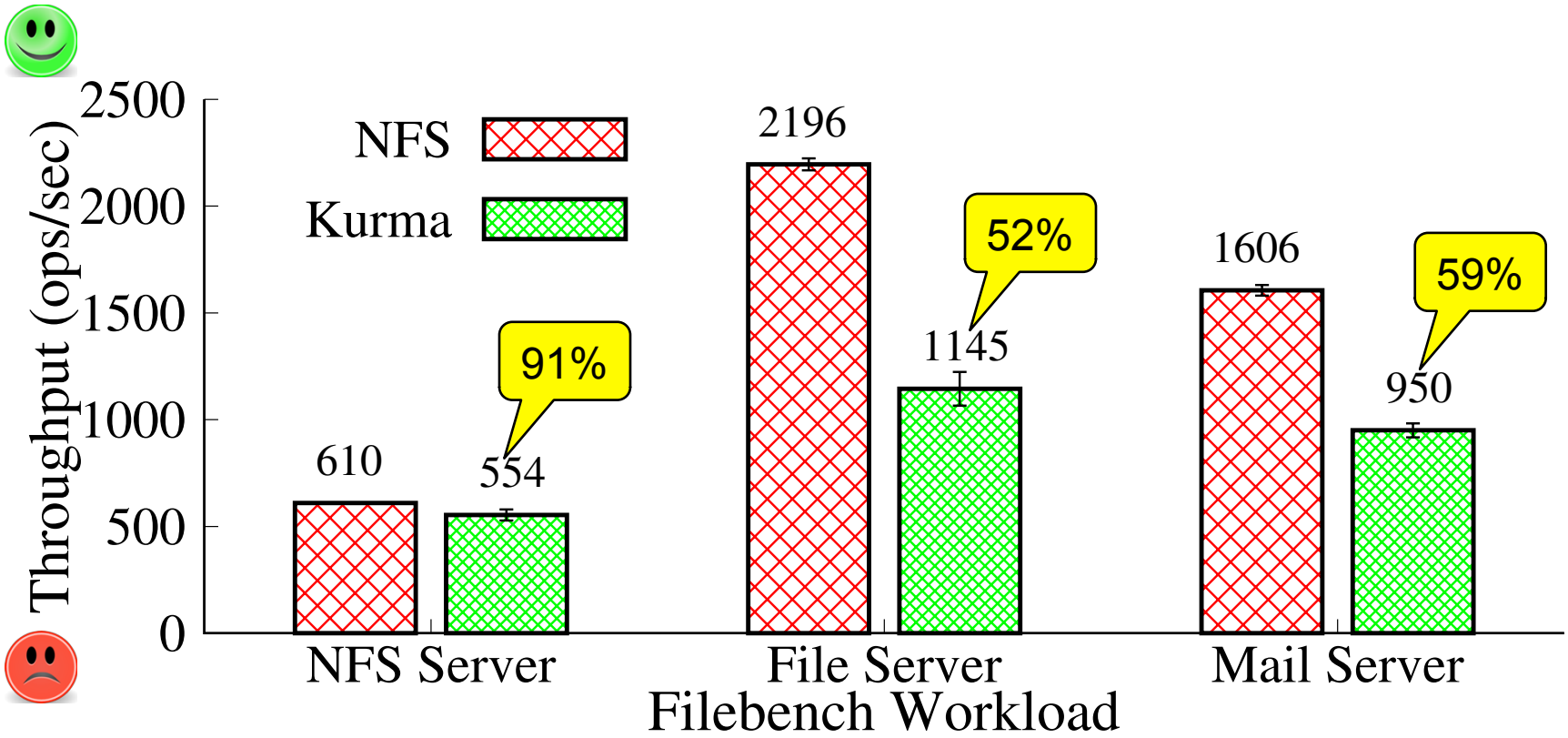
Stony Brook University

Filebench Workloads



Stony Brook University

Filebench Workloads



Stony Brook University

Conclusions

- Kurma: secure distributed multi-cloud gateways
  - Protect file data with authenticated encryption
  - Store file metadata on-premises ZooKeeper
  - Securely share data across regions
  - Multi-cloud: replication, erasure coding, secret sharing
- Implementation and evaluation
  - Data operations are as fast as traditional NFS
  - Metadata performance: 51-54%
  - Filebench workloads: 52-91%
  - http://github.com/sbu-fsl/kurma

Kurma: Secure Geo-distributed Multi-cloud Storage Gateways

Q&A

Stony Brook University