TiDedup: A New Distributed Deduplication Architecture for Ceph

Myoungwon Oh, Sungmin Lee, Samuel Just, Young Jin Yu, Duck-Ho Bae
Sage Weil, Sangyeun Cho, Heon Y. Yeom
Samsung Electronics IBM Ceph Foundation Seoul National University

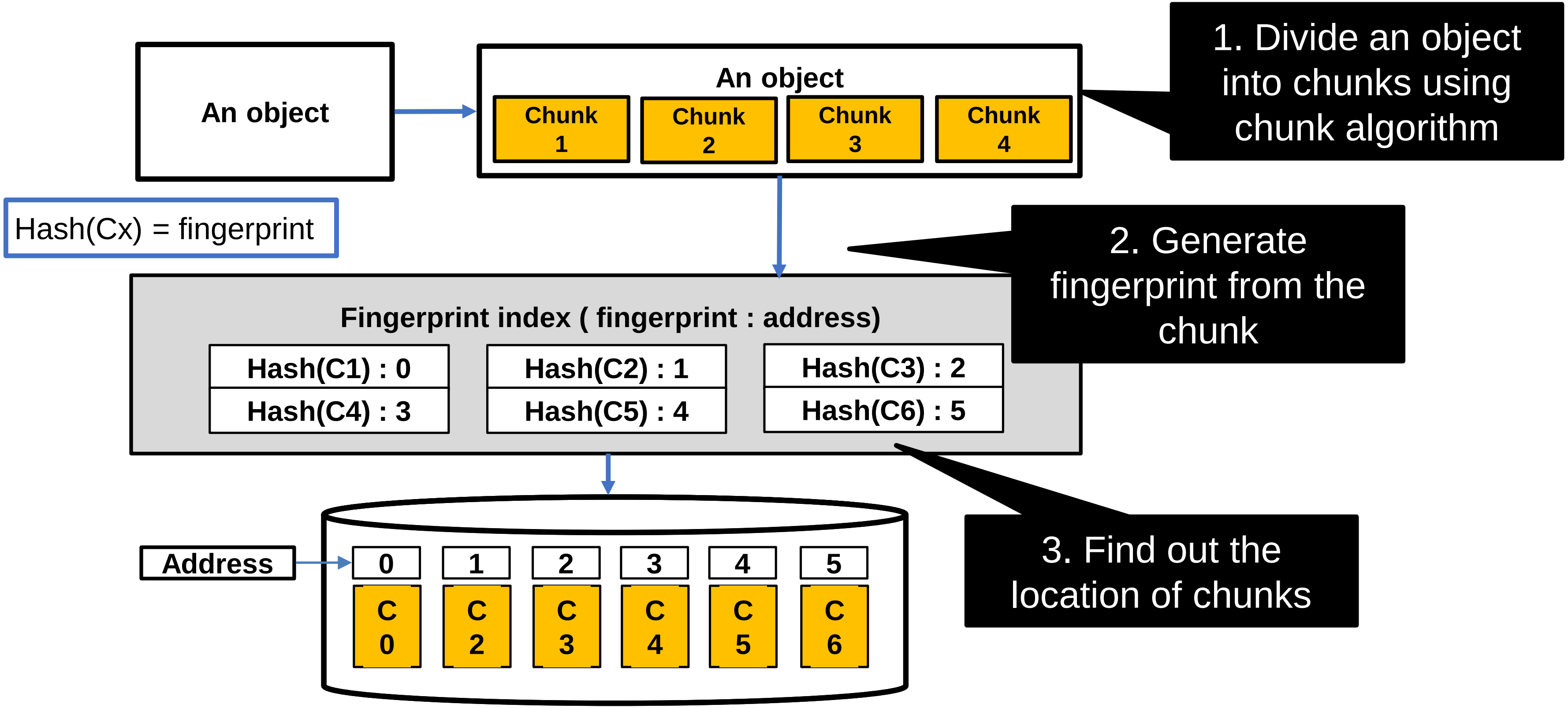
SOLUTION
Core Values

AGENDA

- Background
- Motivation
- Design
- Conclusion

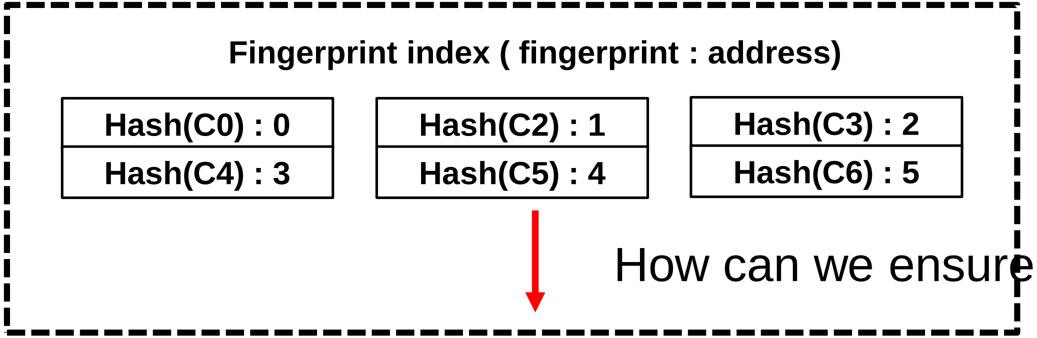
Background

- **Conventional (local) deduplication system**



Background

- **We had a discussion on how deduplication can be implemented on Ceph. The primary concern is how Ceph manage fingerprint index at scale**
  - How to look up a pair < fingerprint : address >
  - How to distribute fingerprint index entry evenly



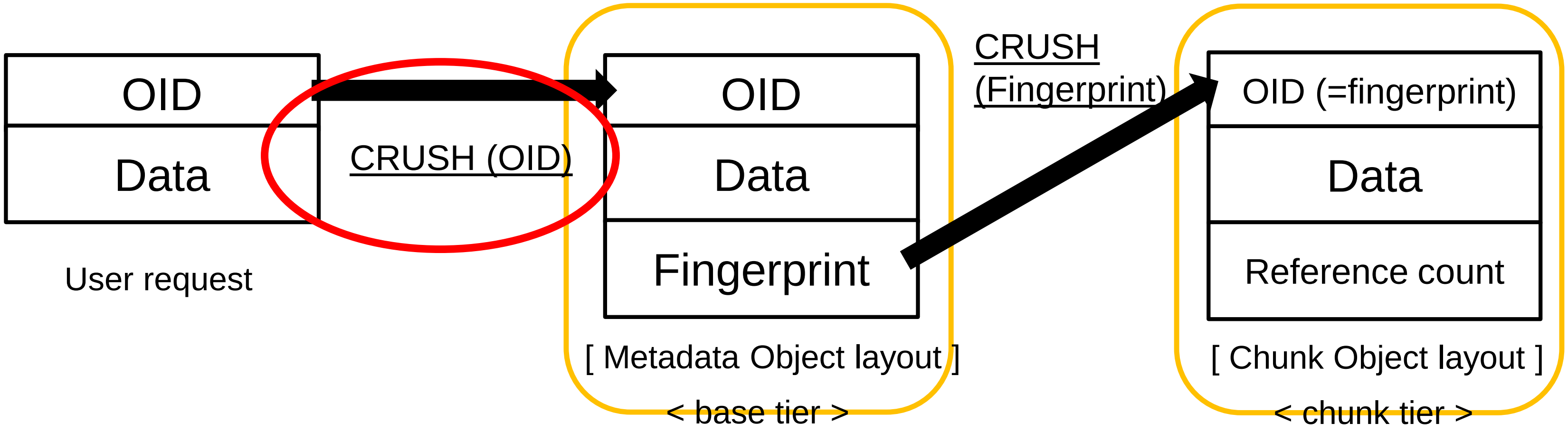
How can we ensure the scalability?

Our prior solution is double hashing !!

Background (prior work)

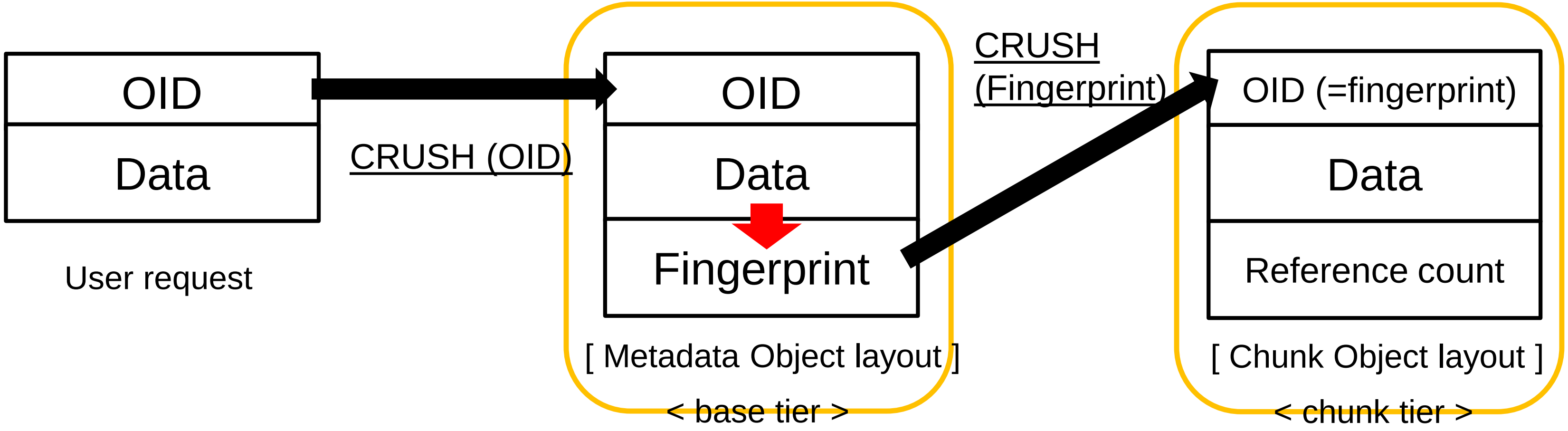
What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself



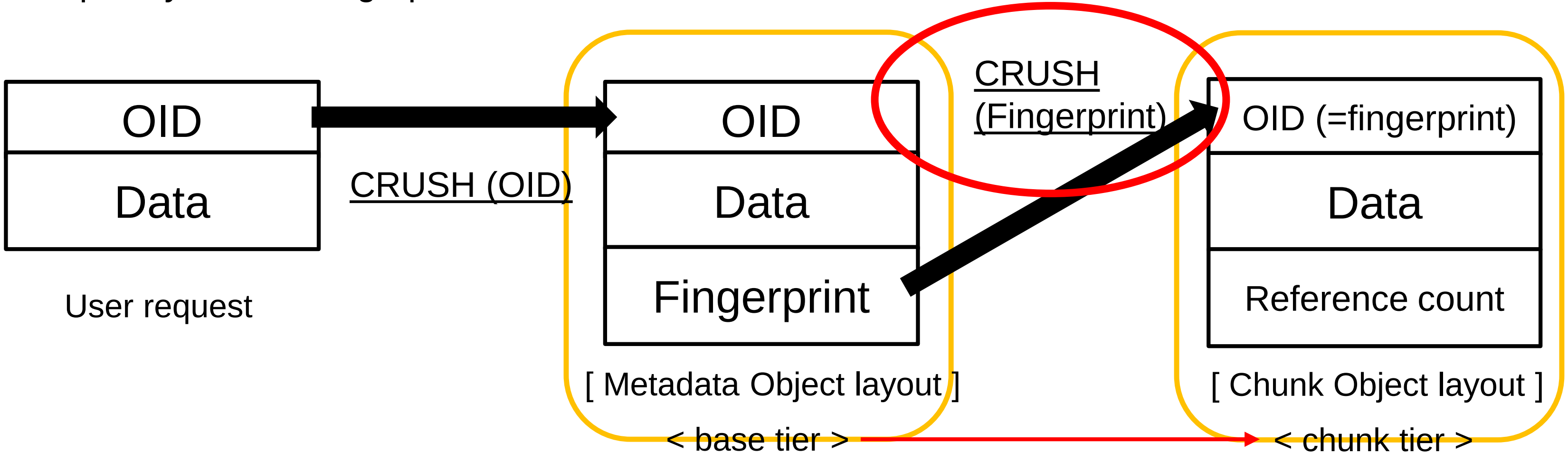
What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself



What is double hashing?

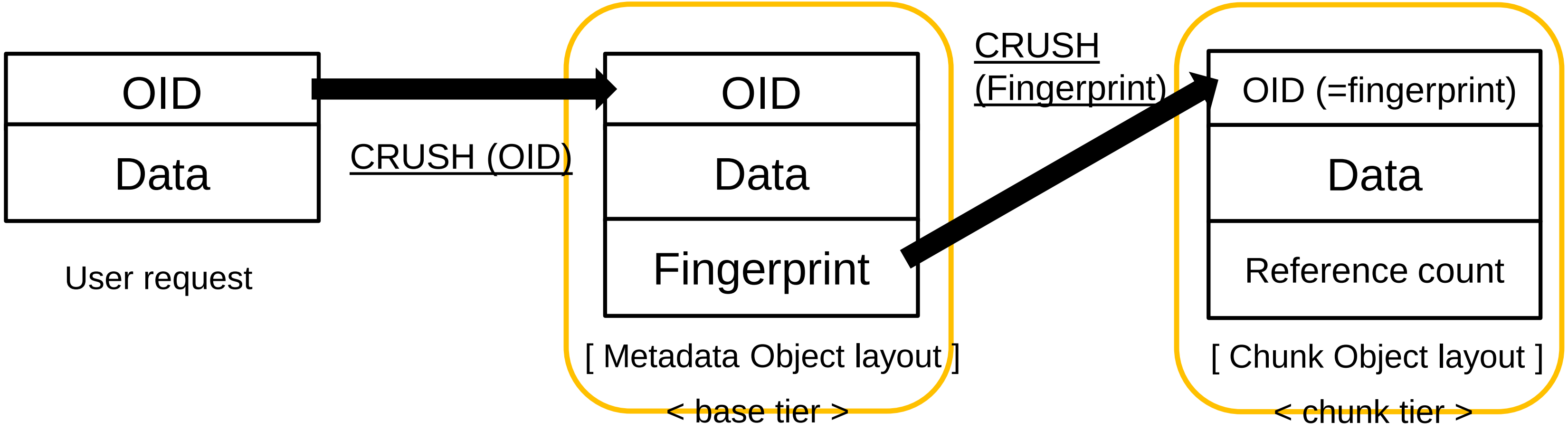
- Use existing data placement algorithm once again
- Completely remove fingerprint index itself



Background (prior work)

What is double hashing?

- Use existing data placement algorithm once again
- Completely remove fingerprint index itself



Metadata and chunk objects are equal to existing object
More details in the paper !

Motivation

- **We made a prototype version of cluster-level deduplication, but the following challenges arose**

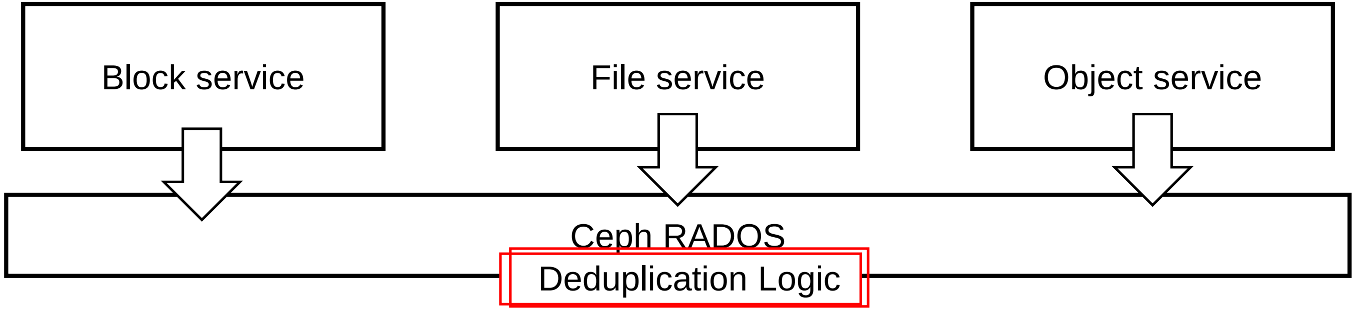
1. ***Is deduplication really helpful?***

2. ***Structure limitations: performance degradation and inefficient chunking***

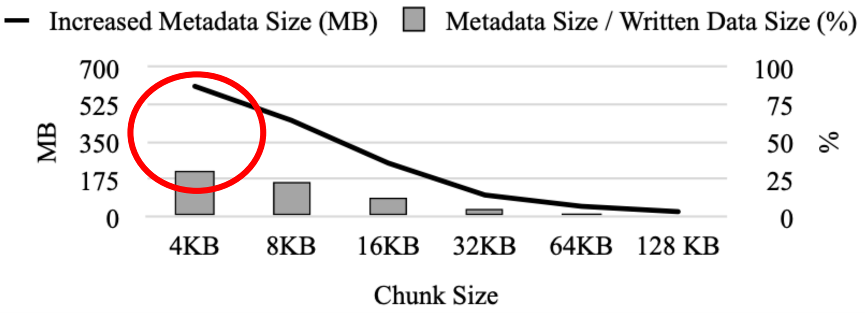
3. ***Snapshot and scrub (reference GC) overhead***

Is deduplication always helpful?

- **Ceph is a general-purpose storage system**

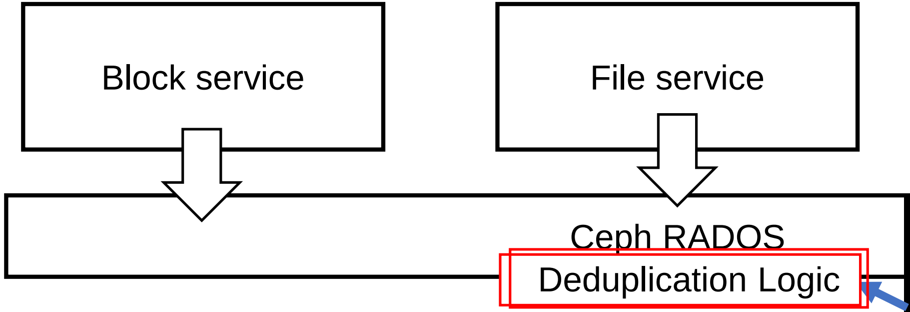


- **Deduplication penalty: If objects have unique content, the storage space is wasted, if anything.**



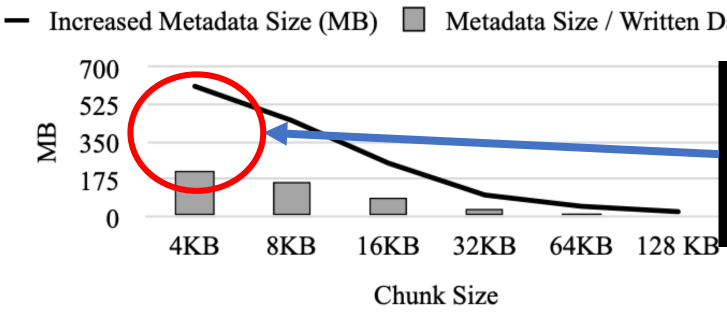
Is deduplication always helpful?

- **Ceph is a general-purpose storage system**



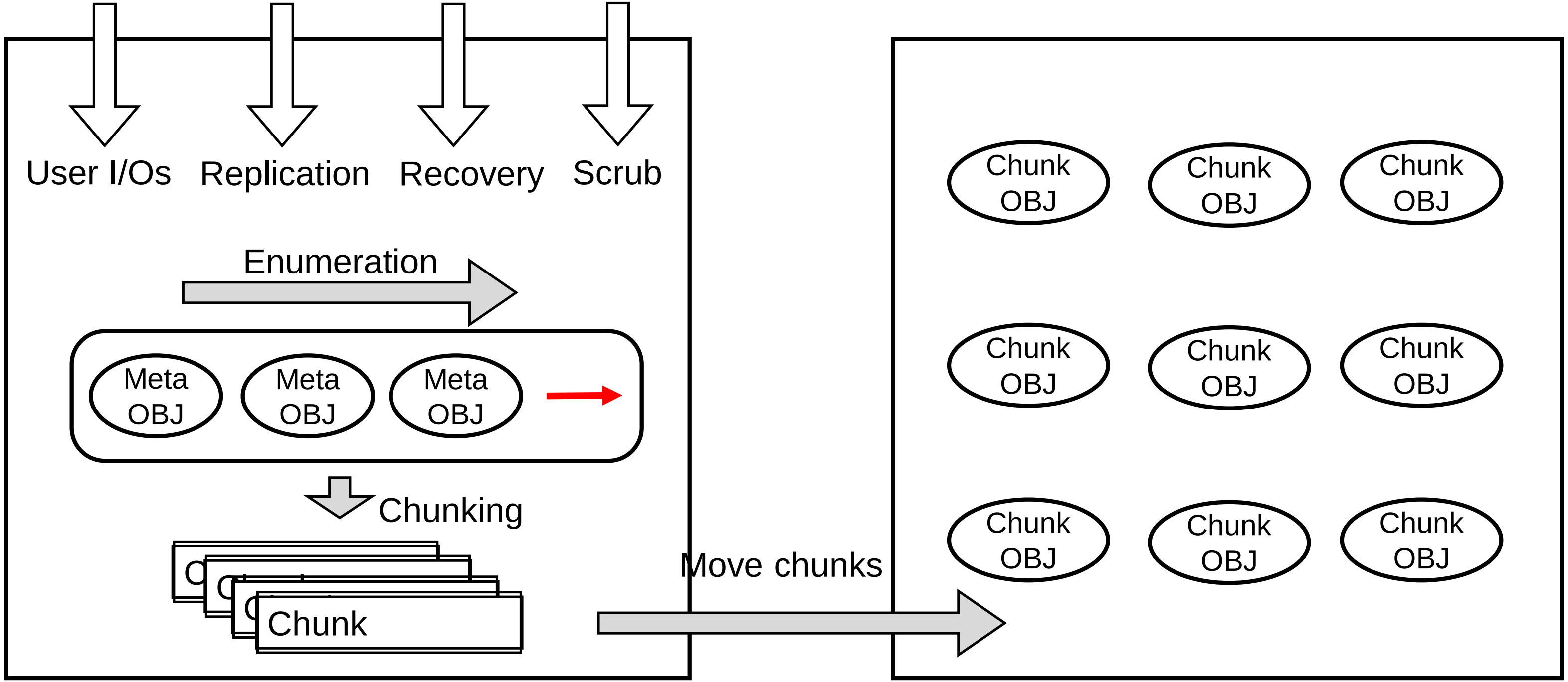
Workload differs in the services: Dedup could or could not be helpful

- **Deduplication penalty: If objects have unique content, the storage space is wasted, if anything.**



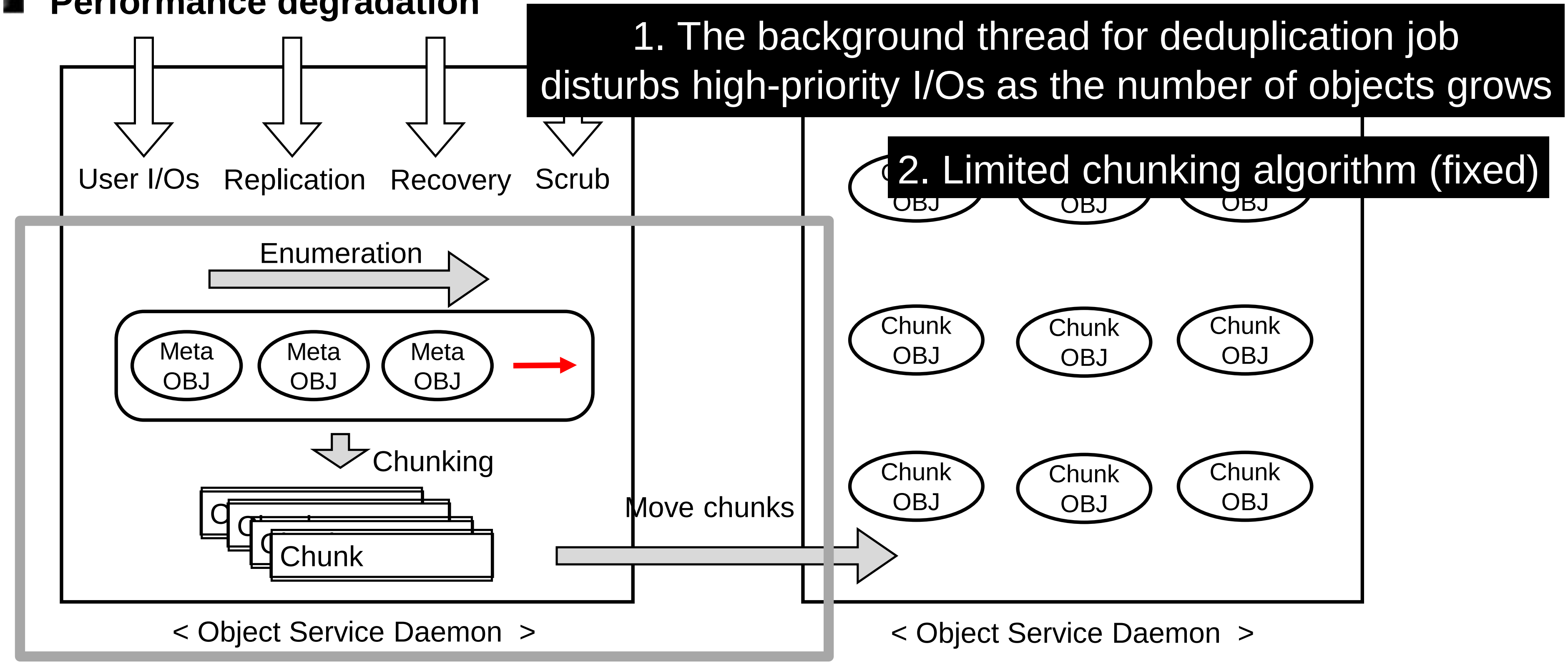
Metadata size added for deduplication increases significantly

- **Performance degradation**



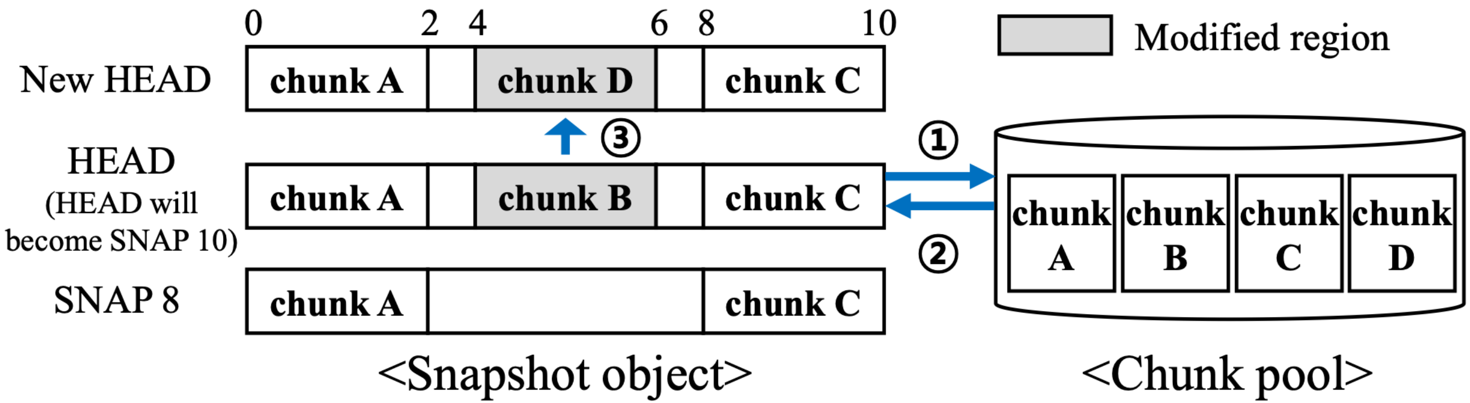
< Object Service Daemon >   < Object Service Daemon >

- **Performance degradation**



Snapshot support

- An naive approach can not work out because snapshot creation generates messages as much as the chunks the object contains (e.g., if SNAP 10 contains three chunks, the three messages are generated)

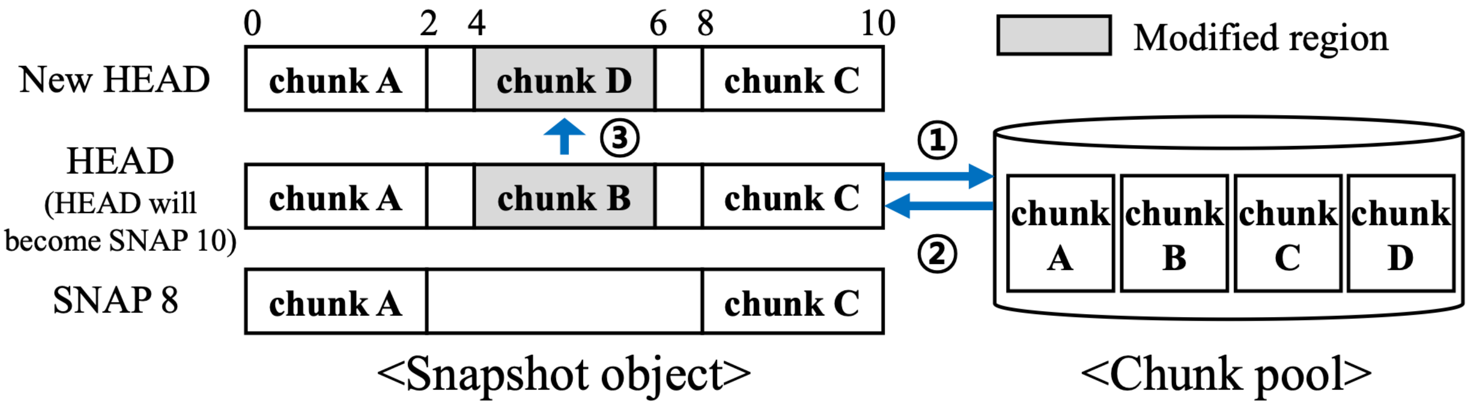


< 1. Send increment message for three chunks for snapshot creation, 2. Acks, 3. Snapshot creation is done >

Snapshot support

- An naive approach can not work out because snapshot creation generates messages as mu … he three me…

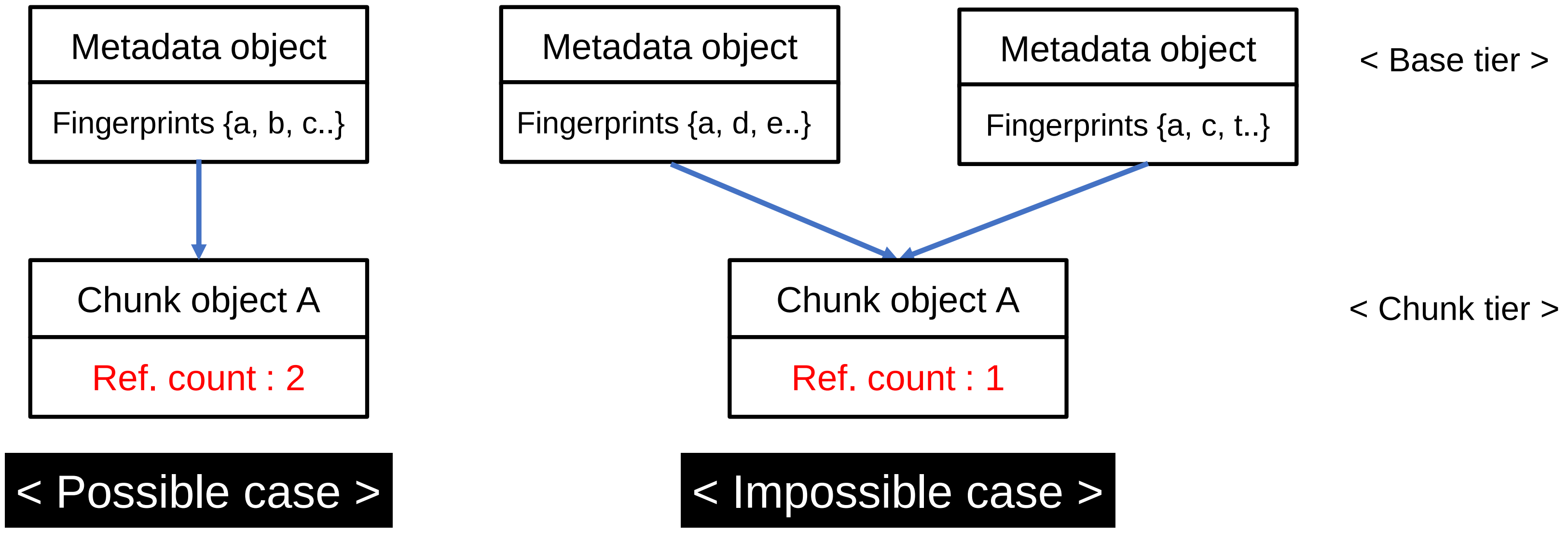
Must wait for all increment messages to complete snapshot creation (1, 2, and 3)



< 1. Send increment message for three chunks for snapshot creation, 2. Acks, 3. Snapshot creation is done >

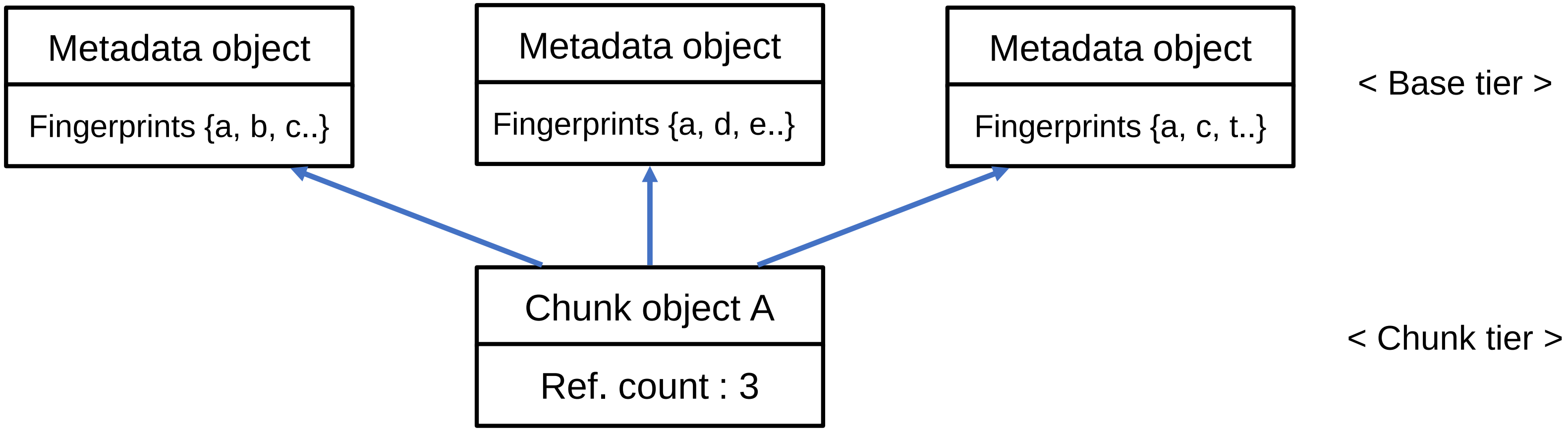
Scrub overhead

- Scrub is something like reference garbage collection that check the reference is valid
- Scrub is needed because reference leak (rarely) occur from false-positive design, as follows:



Scrub overhead

- Scrub requires a full search on the base tier. So, it takes a significant time to complete as the number of objects grows



Need to check all metadata objects to see if ref. count is valid
***O* (#chunk objects × #metadata objects).**

Design

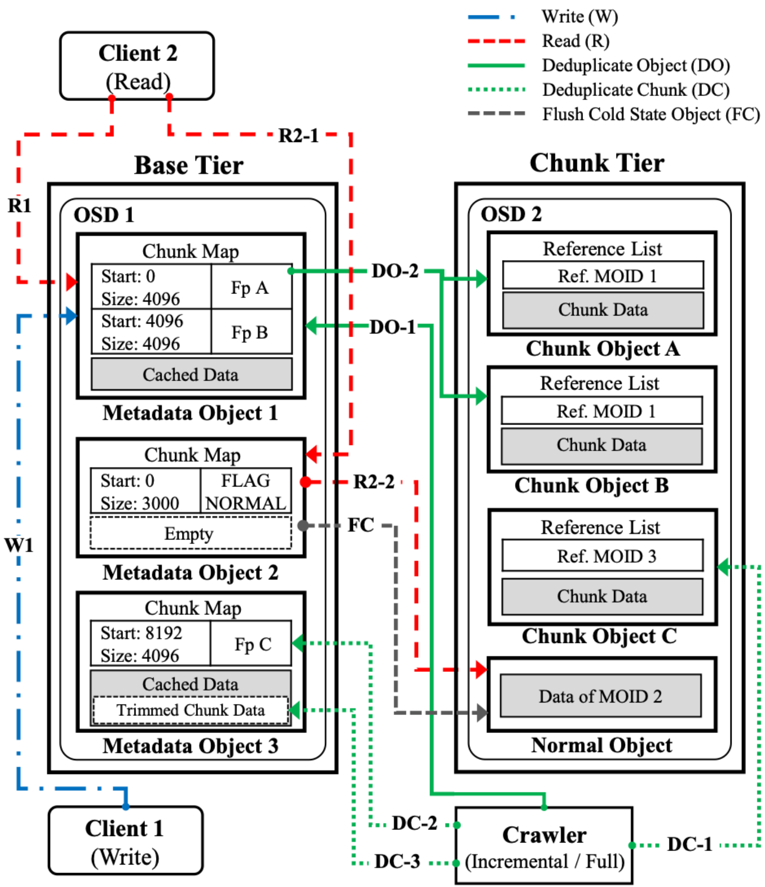
- Selective cluster-level crawling

- Event driven architecture

- OID shared reference scheme

System Overview

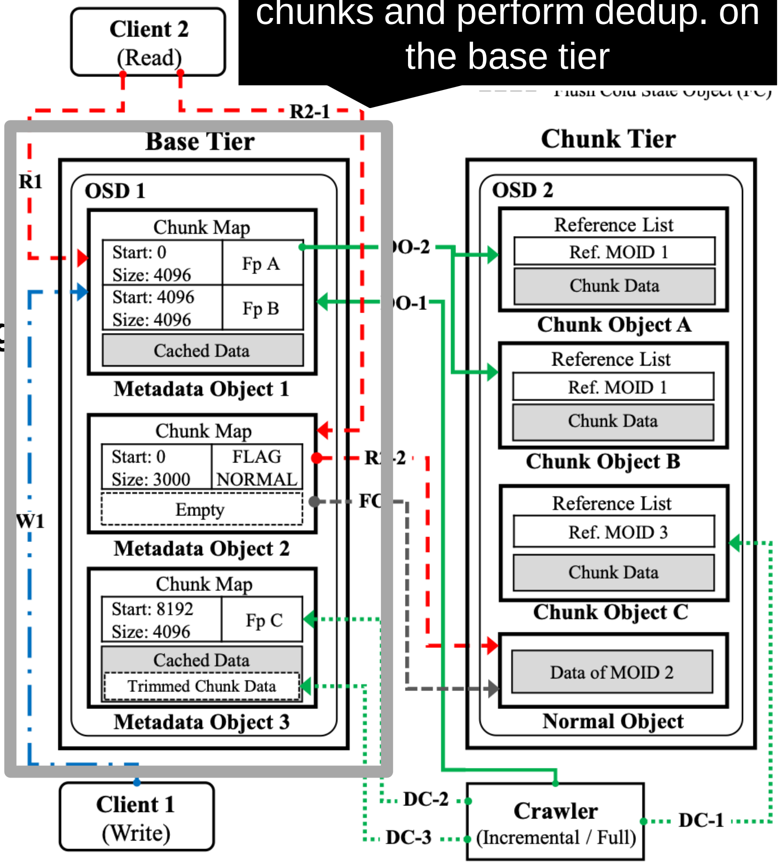
- **Selective dedup processing using crawler**
  - Perform deduplication if data is dedup-able
- **Event-driven architecture with content defined chunking**
  - Only act in the event of predefined APIs with CDC (content defined chunking)
- **OID-shared reference scheme**
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management



System Overview

- **Selective dedup processing using crawler**
  - Perform deduplication if data is dedup-able
- **Event-driven architecture with content defined chunking**
  - Only act in the event of predefined APIs with CDC (content defined chunking)
- **OID-shared reference scheme**
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management

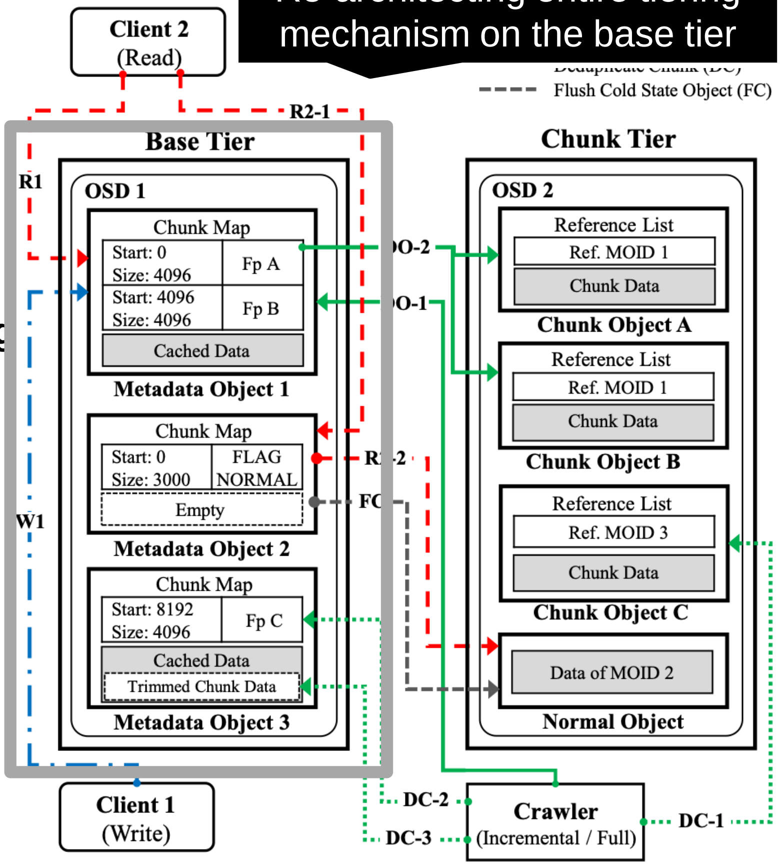
Crawler find out dedup-able chunks and perform dedup. on the base tier



System Overview

- **Selective dedup processing using crawler**
  - Perform deduplication if data is dedup-able
- **Event-driven architecture with content defined chunking**
  - Only act in the event of predefined APIs with CDC (content defined chunking)
- **OID-shared reference scheme**
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management

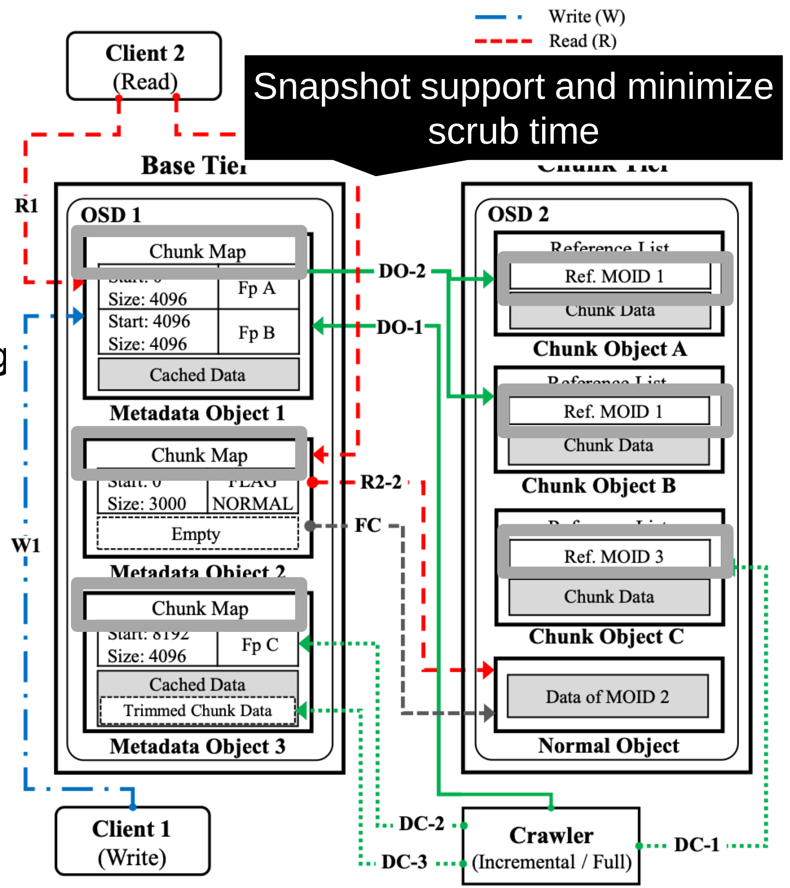
Re-architecting entire tiering mechanism on the base tier



System Overview

- **Selective dedup processing using crawler**
  - Perform deduplication if data is dedup-able
- **Event-driven architecture with content defined chunking**
  - Only act in the event of predefined APIs with CDC (content defined chunking)
- **OID-shared reference scheme**
  - Reduce the number of message for snapshot and scrub based on false-positive ref. management

Snapshot support and minimize scrub time



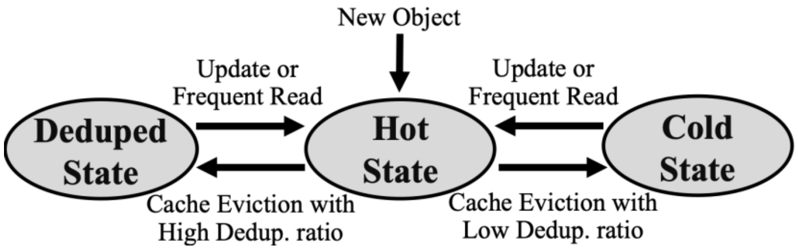
Crawler

- **Decoupled dedup controller scheme from server daemon**
  - Unlike prior work, TiDedup employ a separate process, called crawler
  - The crawler process searches objects, then makes a decision on deduplication if the chunk is dedup-able (> threshold value)

- **Perform deduplication if the chunk is found more than threshold times with two modes**
  - Incremental mode
    - At daytime, scan a small set of metadata object gradually to minimize resource usage
  - Full mode
    - At nighttime, scan all metadata object, then perform deduplication without consider resource utilization

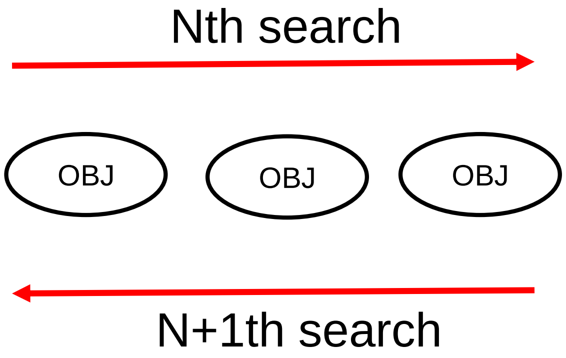
Object management

- Object is one of three states: hot or cold or deduped



Stateless

- Re-execution to overcome failures
- Repeat the loop in the reverse direction
- In-memory fingerprint store
  - Store fingerprints to check which fingerprint is dedup-able
  - Map < fingerprint : duplicate count >



Event-driven architecture

- **The goal is to react an action in the event of external APIs with CDC**
  - I/O path and APIs are designed (e.g., set_chunk, tier_flush, tier_evict, tier_promote)
  - Relevant metadata (e.g., chunk_info_t) is embedded into existing object metadata, called “object_info_t”
  - chunk_state in chunk_info_t can be one of <u>{MISSING, CLEAN}</u>
    - MISSING: no cached content, CLEAN: cached content
- **Read**
  - Find chunk_info_t which is associated with the requested offset
  - Forward read request to a chunk object (OID is the destination OID

    In chunk_info_t) if the chunk state is MISSING
- **Write**
  - Write contents, then clear the corresponding chunk_info_t---content define chunking calculates the different chunk boundary depending on the contents, so modification requires a recalculation

Existing metadata

```
object_info_t {          // object's metadata
   manifest_info_t {
     chunk_map <offset, chunk_info_t>;
   }
   version;
   object state;
}
```

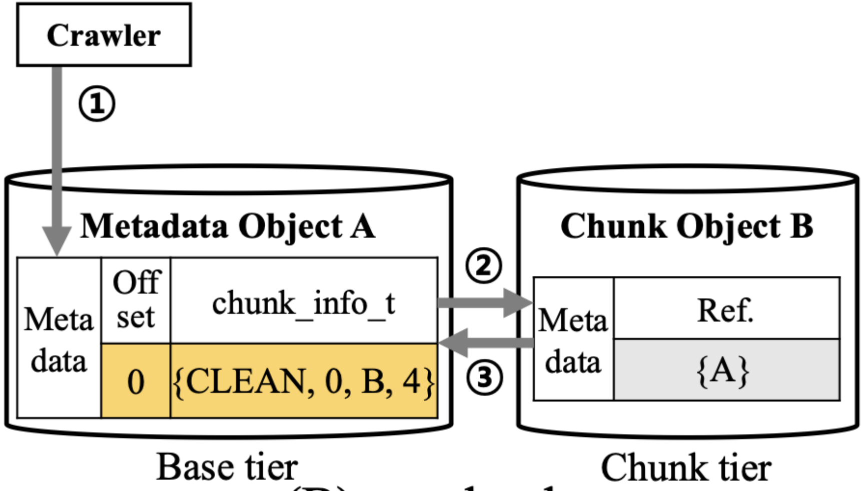
New metadata

```
chunk_info_t {
   chunk state;
   destination offset;
   length;
   destination OID;     // fingerprint value from
                         chunk's object
}
```

Event-driven architecture

Set_chunk()

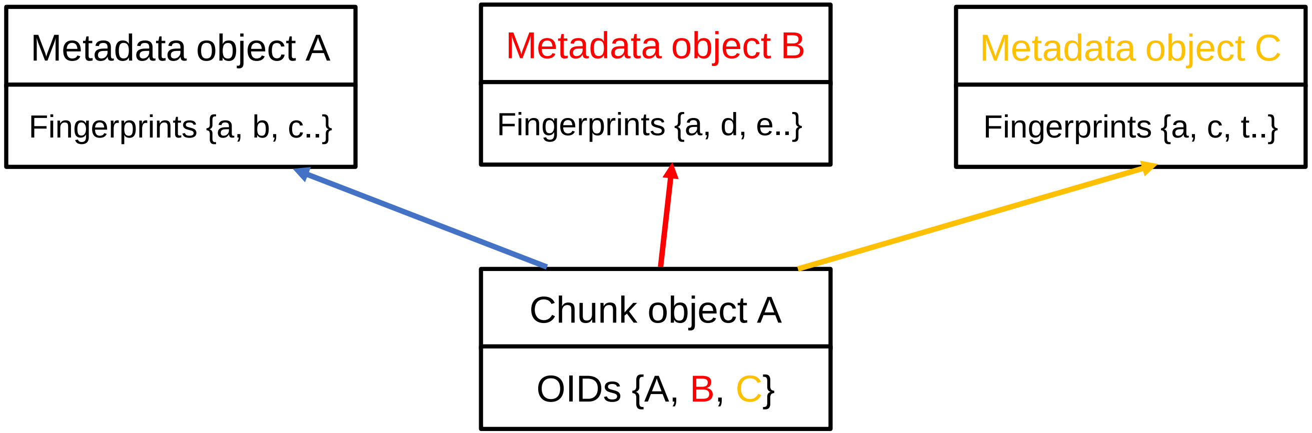
- **Set a part of the object to deduped**
- Synchronous call (wait for increment message’s ack, then reply the result to caller and update metadata)
- Transactional processing



(B) set-chunk

Scrub worker

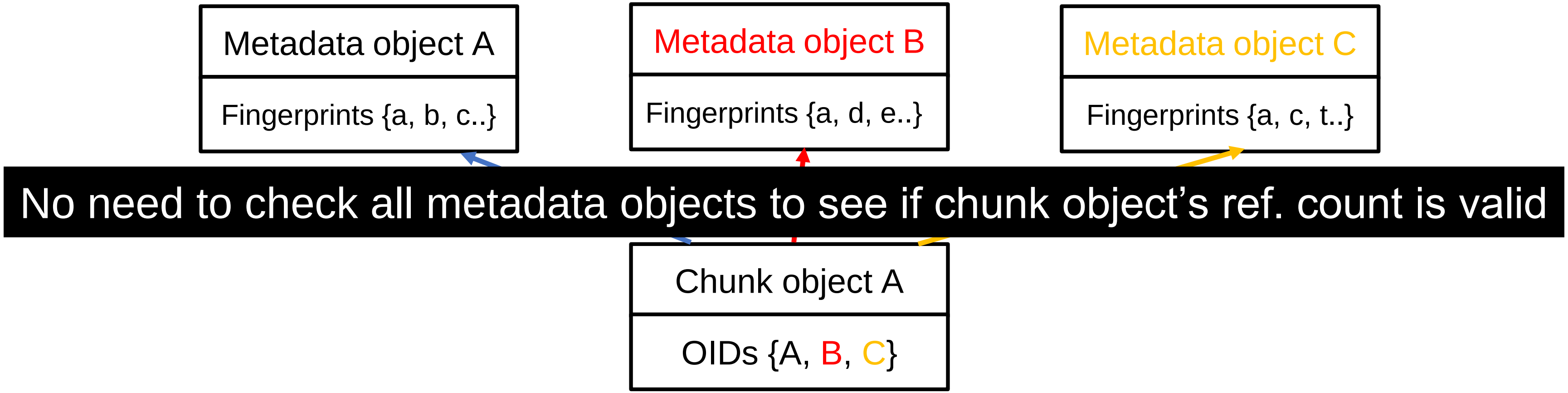
- Each chunk object has OIDs (like a back pointer) instead of ref. count
  - Ceph's OID format includes location information such as tier and object name
- To check if a reference is valid, TiDedup needs only two reads (chunk object's extended attributed and metadata object), unlike the prior work



OID shared reference

Scrub worker

- Each chunk object has OIDs (like a back pointer) instead of ref. count
  - Ceph's OID format includes location information such as tier and object name
- To check if a reference is valid, TiDedup needs only two reads (chunk object's extended attributed and metadata object), unlike the prior work

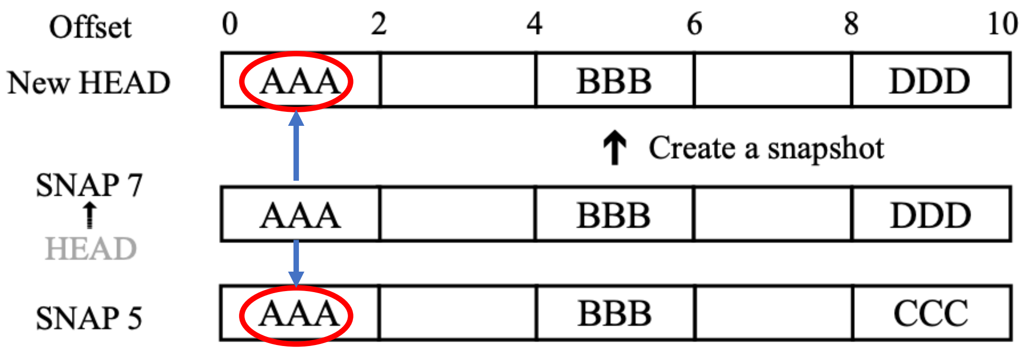


Objective

- Based on false-positive design, minimize the number of generated message

Snapshot creation

- Check adjacent snapshot's chunk first, then do not add chunk's reference if adjacent snapshot's chunk is identical when creating a new snapshot
- The number of chunk's reference:
  - ➢ AAA: 1, BBB: 1, DDD: 1, CCC: 1



Snapshot deletion

- If either adjacent snapshots has the same chunk, no messages needs to be generated
- TiDedup would send a delete reference message only if adjacent chunks and itself are unreferenced.

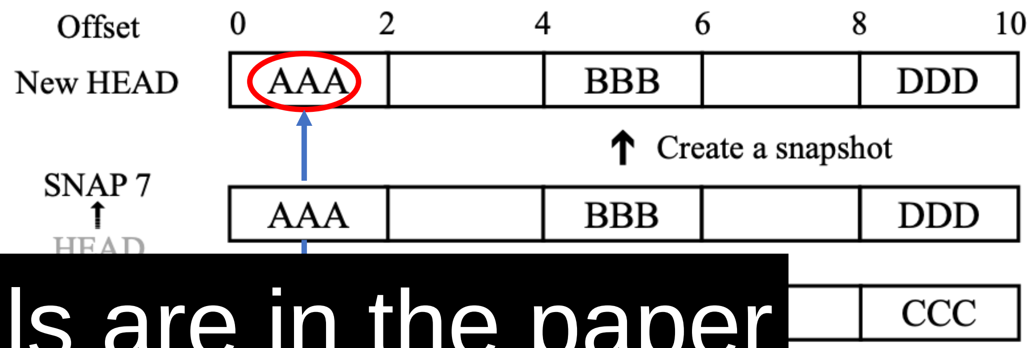
OID shared reference

Objective

- Based on false-positive design, minimize the number of generated message

Snapshot creation

- Check adjacent snapshot's chunk first, then do not add chunk's reference if adjacent snapshot's chunk is identical when creating a new snapshot
- The number of chunk's reference:
  - ➢ AAA: 1, BBB: 1, DDD: 1, CCC: 1



More details are in the paper

Snapshot del

- If either adjacent snapshots has the same chunk, no messages needs to be generated
- TiDedup would send a delete reference message only if adjacent chunks and itself are unreferenced.

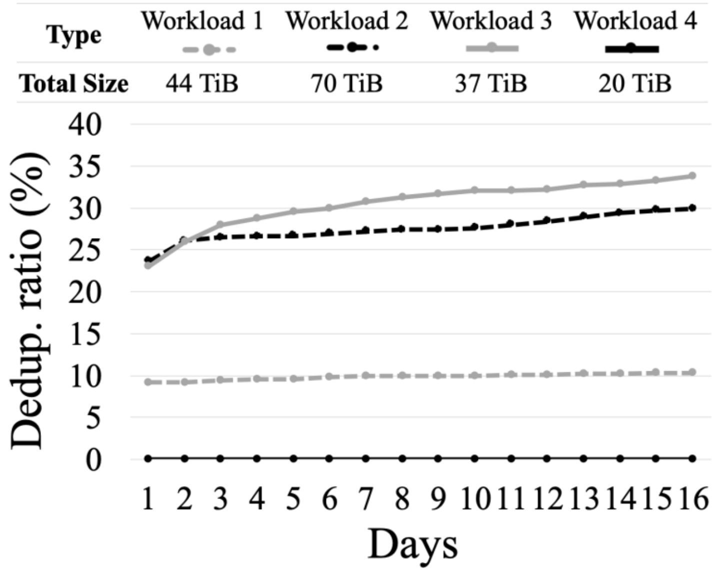
Evaluation

Test Environment

- 10 machines in total (a 2-way AMD EPYC 7543 32-cores)
- 512 GB DRAM for each machine
- QLC SSDs (4 TB) for each machine
- The latest version (Reef) of the Ceph
- Two replicas
- FastCDC [Wen Xia et al., USENIX ATC 2016]
- SHA1 is used to generate fingerprint value among the available fingerprint algorithm options (e.g., SHA1, SHA128, and SHA256).

Evaluation

- **The effect of deduplication in real world workload**



< Space saving on factory data >

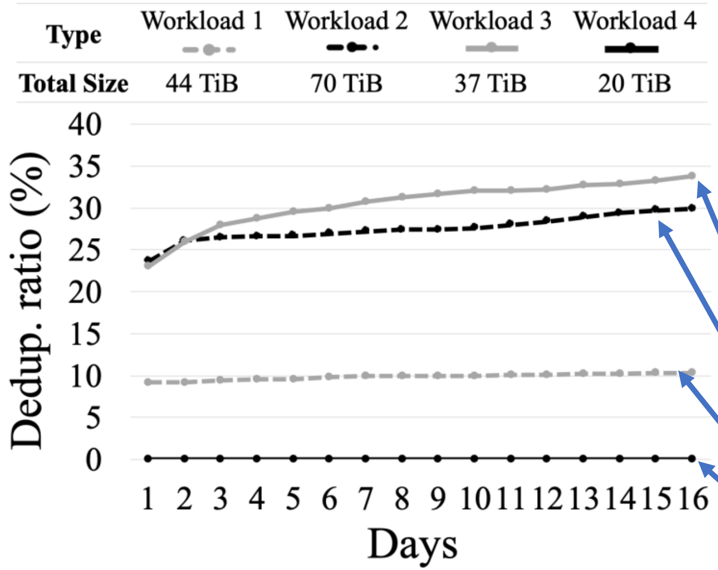
Workload 1: equipment status

Workload 2: chip information during manufacturing

Workload 3: logs for photo lithography

Workload 4: metrology and inspection image files

The effect of deduplication in real world workload



< Space saving on factory data >

Workload 1: equipment status

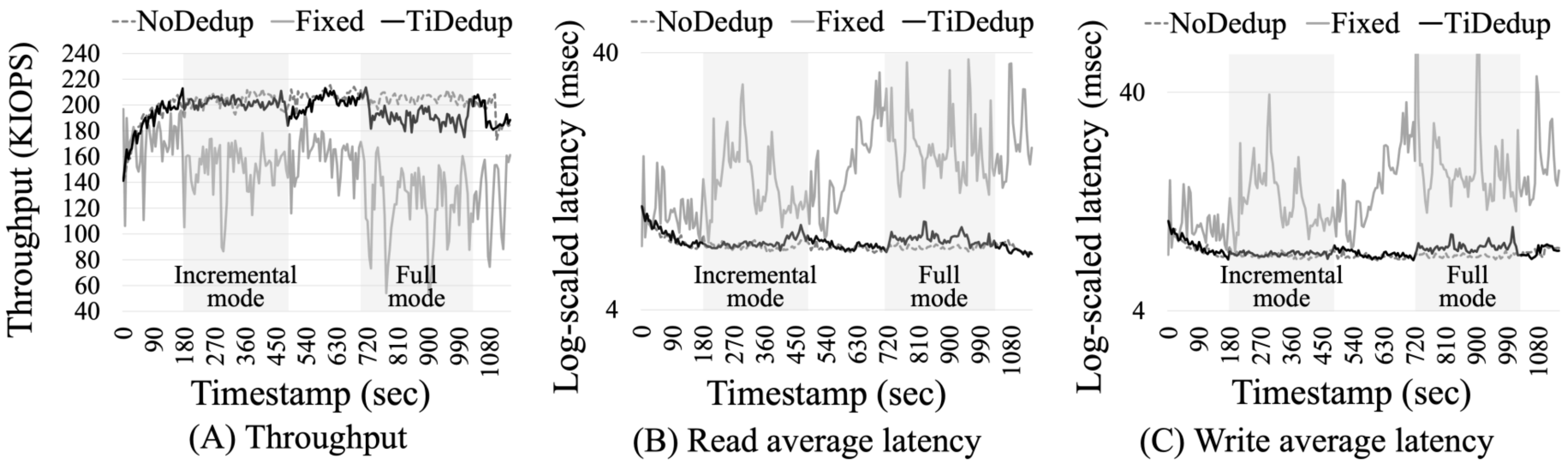
Workload 2: chip information during manufacturing

Workload 3: logs for photo lithography

Workload 4: metrology and inspection image files

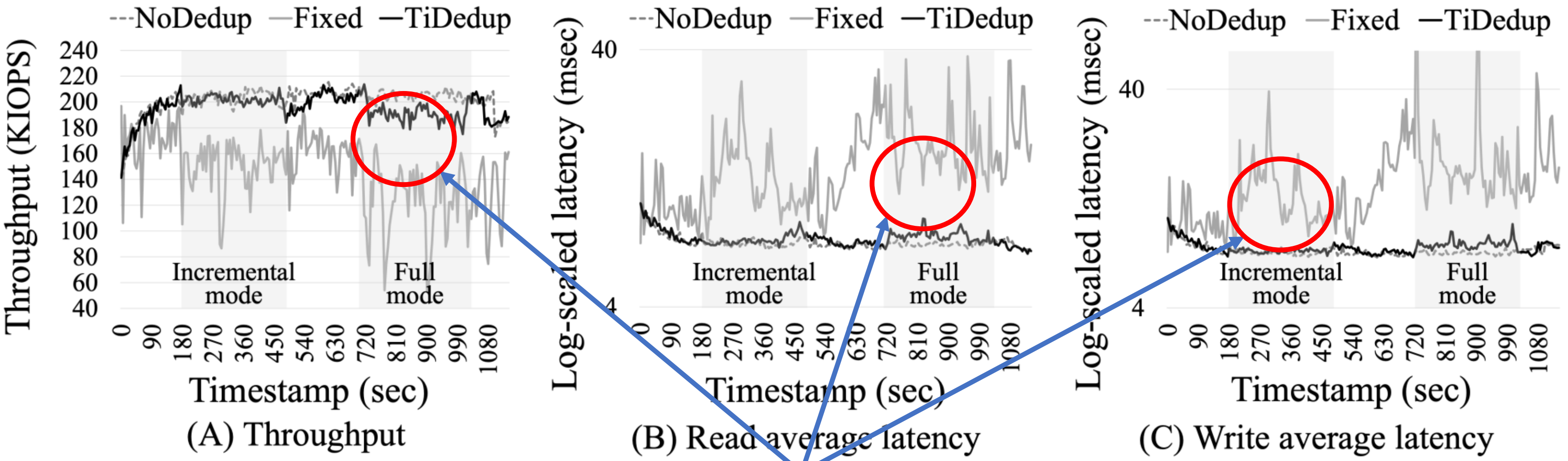
The effect of deduplication differs in the workloads

- **Throughput test**



< YCSB throughput (Workload a) >

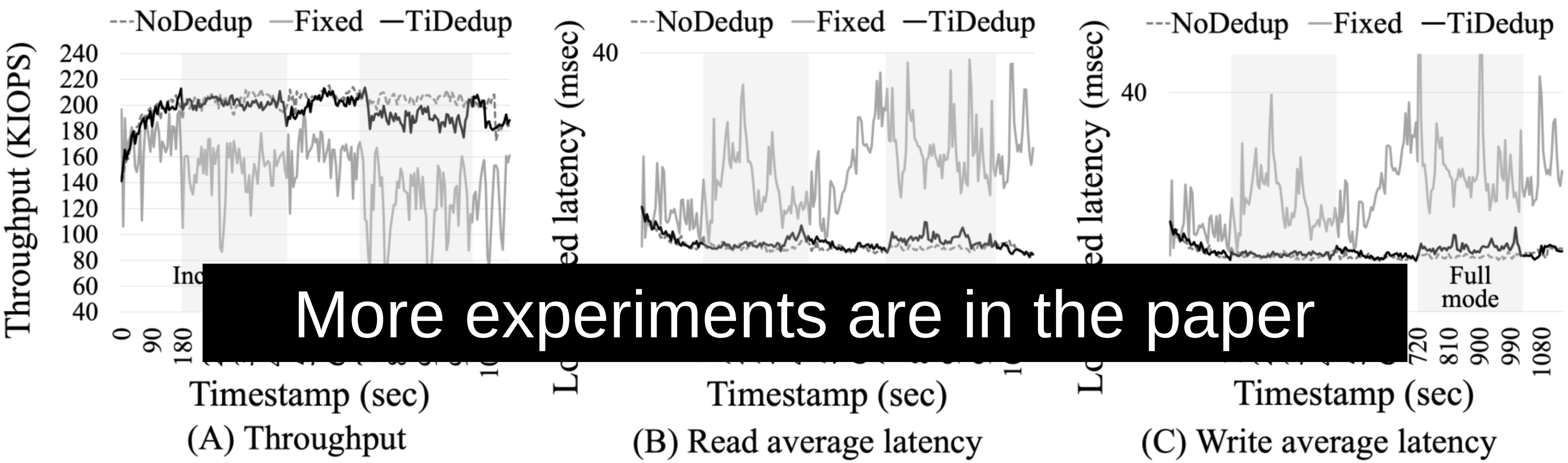
- **Throughput test**



(A) Throughput (B) Read average latency (C) Write average latency

Performance fluctuation disappear in TiDedup

- **Throughput test**



More experiments are in the paper

< YCSB throughput (Workload a) >

Conclusion

- **Prior work has a few limitations to be used in Ceph**
  - Storage saving, structure limitations, scrub and snapshot support
- **TiDedup proposed three key design to resolve observed problems**
  - Selective crawler, event-driven architecture, OID-shared reference
- **The source code is available at Ceph github**
  - https://docs.ceph.com/en/latest/dev/deduplication/
  - https://github.com/ssdohammer-sl/ceph/tree/tidedup
- **Welcome feedback**
  - GitHub, private email, mailing list, whatever


SOLUTION
CORE VALUES
Speciality
Ownership
Leadership
Upgrowth
Together
Integrity
Openness
Now
Speciality
Ownership
Leadership
Upgrowth
Together
Integrity
Openness
Now

Thank You

- **The effect of deduplication in real world workload**

	Virtual disks			Logs		
Chunk size	8K	16K	32K	8K	16K	32K
Fixed	21%	12%	10%	5.7%	5.4%	5.3%
TiDedup (CDC)	45%	36%	27%	18.5%	16%	12.6%

< Space saving on real-world datasets depends on the chunking algorithm and average chunk size. Virtual disks represents VMware vSphere images (10.1 TB) from a developer cloud service (67 users). Logs represents service logs (560 GB) for cloud Infrastructure including monitoring and device state.>

- **The effect of deduplication in real world workload**

	Virtual disks			Logs		
Chunk size	8K	16K	32K	8K	16K	32K
Fixed	21%	12%	10%	5.7%	5.4%	5.3%
TiDedup (CDC)	45%	36%	27%	18.5%	16%	12.6%

< Space saving on real-world datasets depends on the chunking algorithm and average chunk size. Virtual disks represents

CDC shows considerable improvement in terms of space saving in some workload
The both algorithms can be a compliment approach

infrastructure including monitoring and device state.>