10. Capacity Optimization

Special Topics in Computer Systems:
Modern Storage Systems
(IC820-01)

Instructor:

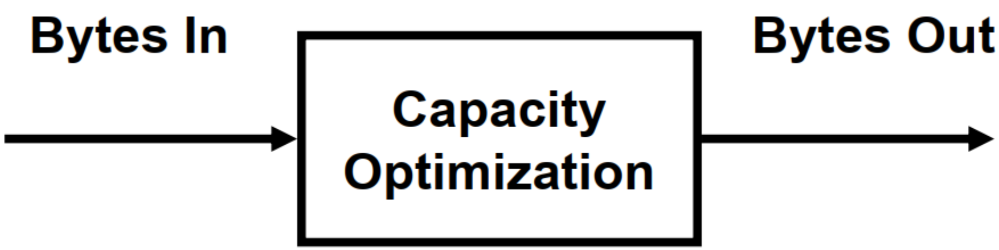
Prof. Sungjin Lee (sungjin.lee@dgist.ac.kr)

Capacity Optimization

- **There are two popular methods to improve storage capacity**
  - **Method 1: *Data Deduplication***
    - The replacement of multiple copies of data with references to a shared copy in order to save storage space
  - **Method 2: *Data Compression***
    - The encoding of data to reduce its storage requirements

- **Widely used in modern storage systems and devices**
  - Not only improve *storage capacity*, but *I/O performance* and *energy efficiency*

Space Reduction Ratio & Percent



$$\text{Ratio} = \frac{\text{Bytes In}}{\text{Bytes Out}} \qquad \% = \frac{\text{Bytes In} - \text{Bytes Out}}{\text{Bytes In}}$$

$$\text{Ratio} = \left(\frac{1}{1 - \%}\right) \qquad \% = 1 - \left(\frac{1}{\text{Ratio}}\right)$$

Space Reduction Ratio & Percent (Cont.)

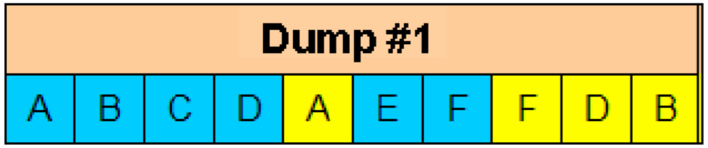
Space Reduction Ratio	Space Reduction Percent
2:1	1/2 = 50%
5:1	4/5 = 80%
10:1	9/10 = 90%
20:1	19/20 = 95%
100:1	99/100 = 99%
500:1	499/500 = 99.8%

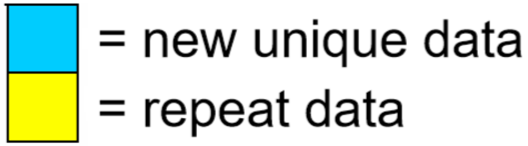
- Ratios can meaningfully be compared only under the same set of assumptions
- Relatively low space reduction ratios provide significant space savings

Outline

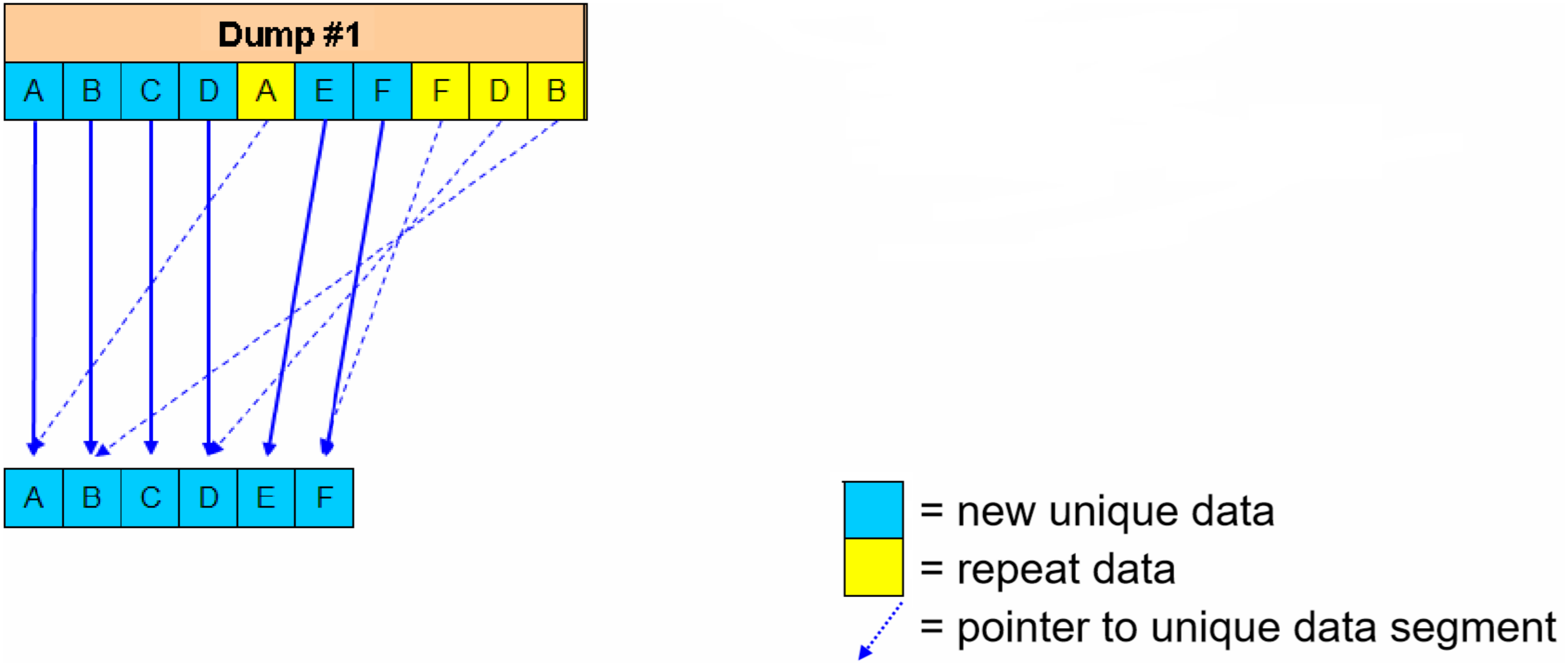
- **Data Deduplication**
- Data Compression
- Case Studies

Data Deduplication Simplified

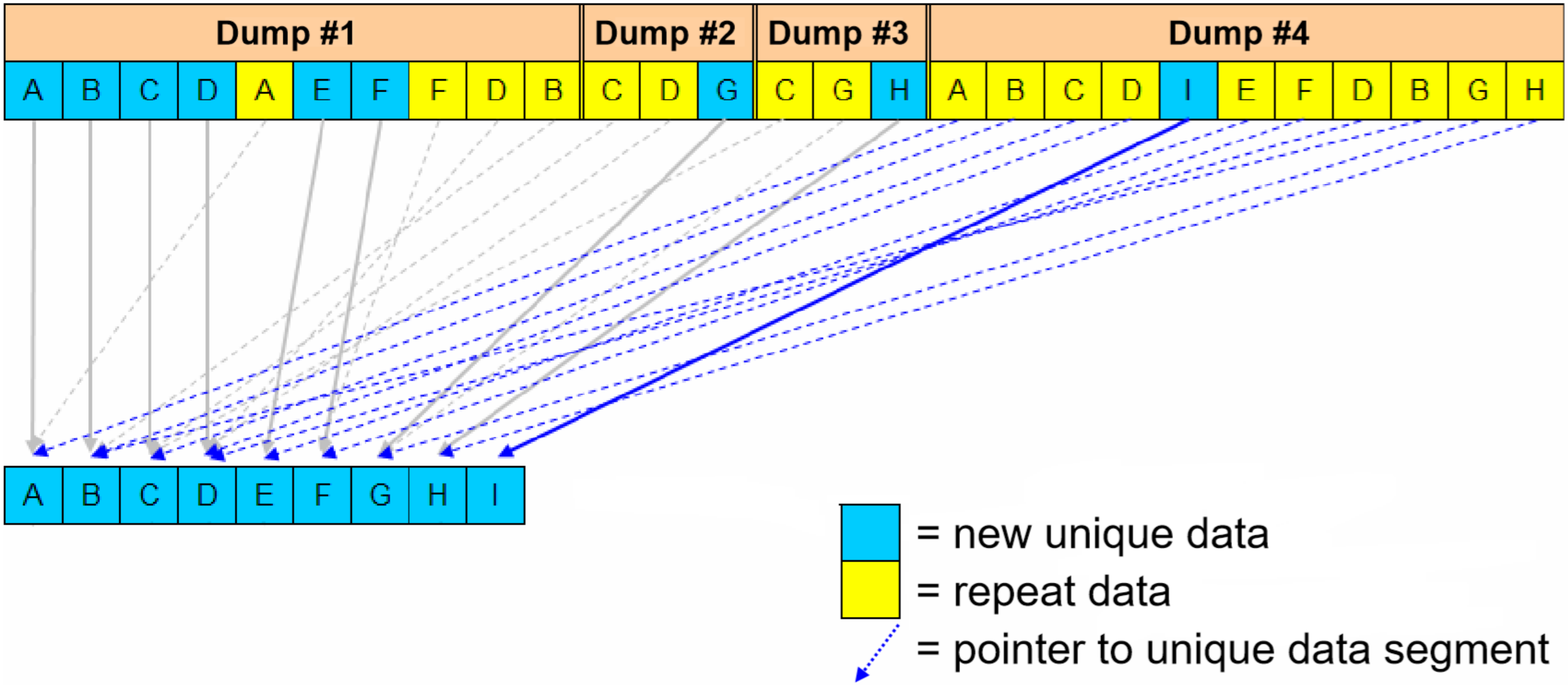




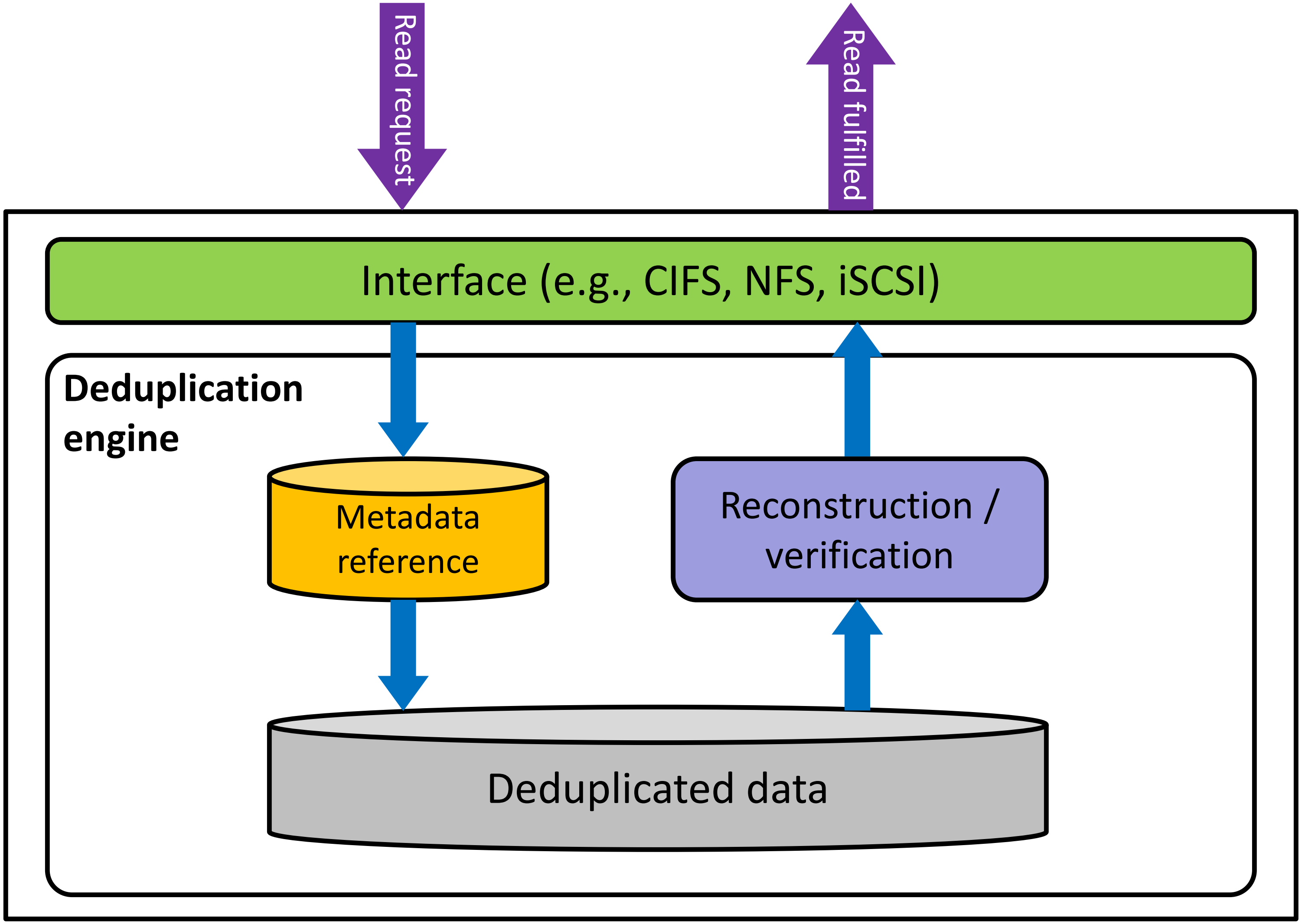
Data Deduplication Simplified (Cont.)



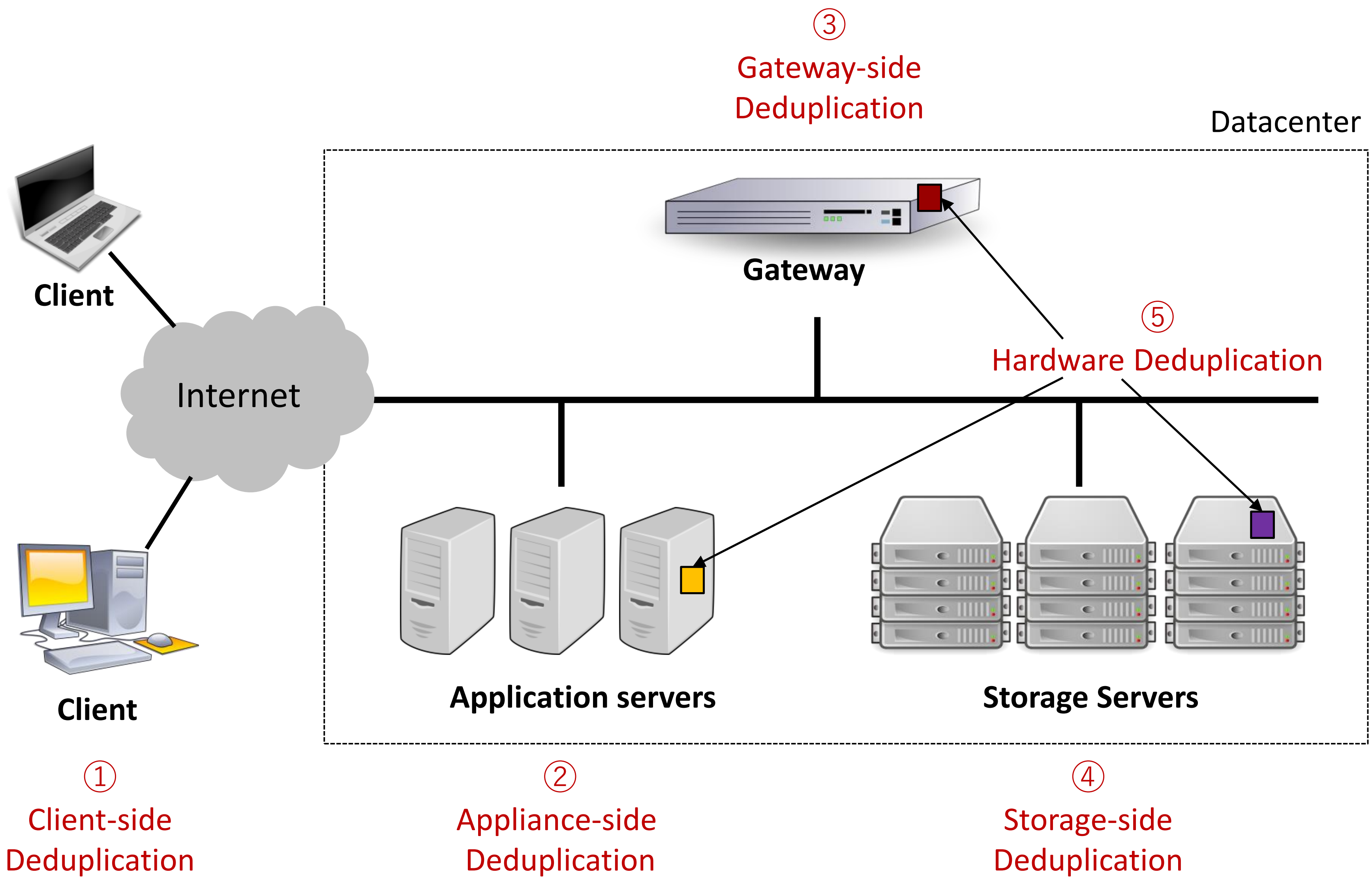
Data Deduplication Simplified (Cont.)



Reading Duplicated Data



Where is Dededupliation Deployed?



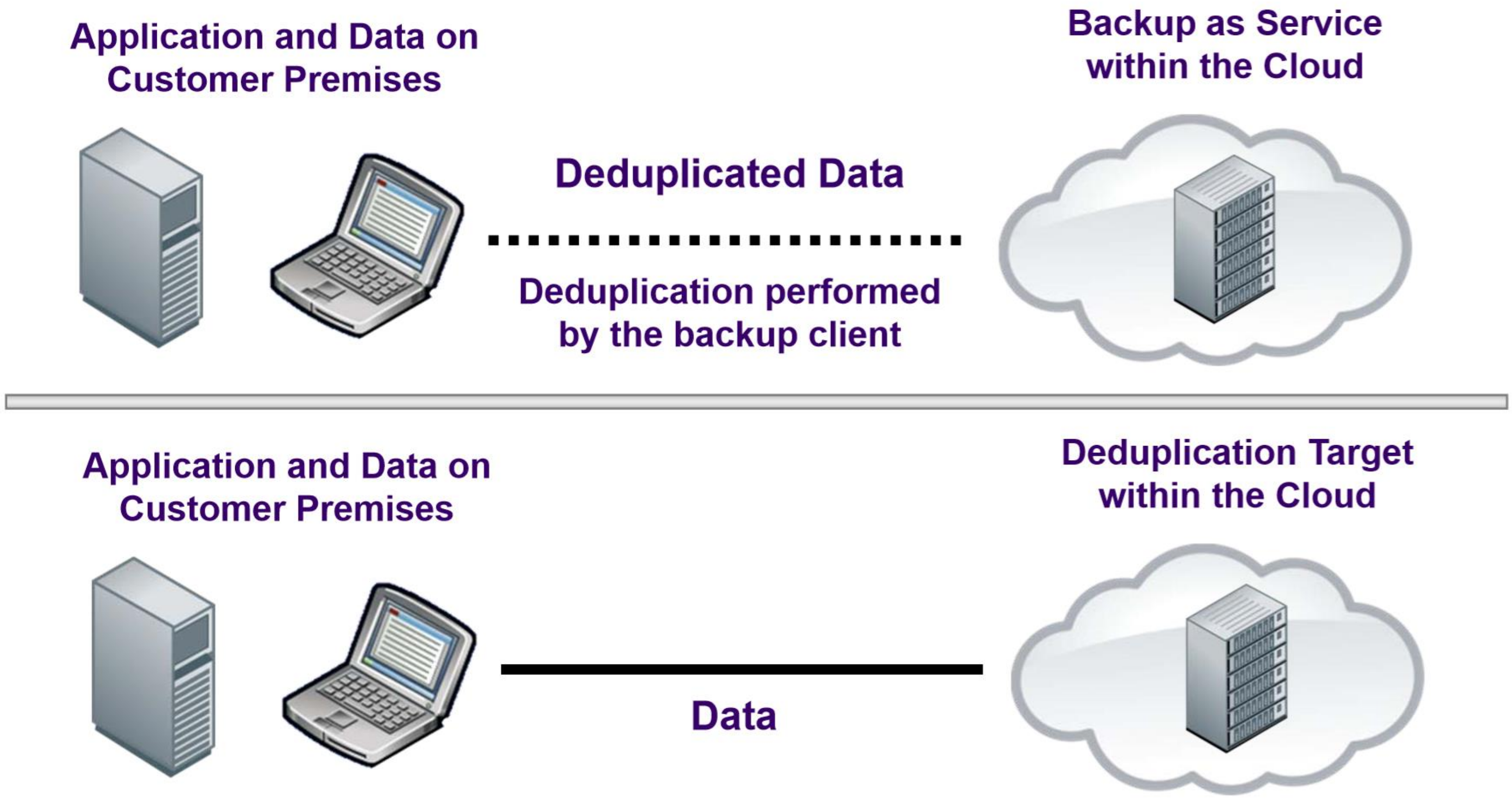
Source and Target

- ***Source Deduplication***
  - Identifies duplicate data at the client
  - Transfers unique segments to a central repository
  - Separate client and server components

- ***Target Deduplication***
  - Identifies duplicate data where the data is being stored
  - Stores unique segments
  - Standalone systems

- **Consideration**
  - Neither approach enables a greater or lesser space savings
  - Scope of data deduplication may vary by implementation

Source and Target (Cont.)



Inline or Post-Process

- ***Inline Deduplication***
  - Data deduplication performed *before* writing the duplicate data
  - May improve I/O performance
  - Low deduplication ratio

- ***Post-process Deduplication***
  - Data deduplication performed *after* the data to be deduplicated has been initially stored
  - May degrade I/O performance
  - High deduplication ratio

Fixed or Variable Size Segment

- ***Subfile Data Deduplication***
  - ***Fixed-length Segment Deduplication***
    - Evaluation of data includes a fixed reference window used to look at segments of data during deduplication process
    - Provide fixed granularity (e.g., 4 KB, 8 KB, or 128 KB)
  - ***Variable-length Segment Deduplication***
    - Evaluation of data uses a variable length window to find duplicate data in stream or volume of data processed
    - Provide variable granularity

- ***Single Instance Deduplication***
  - Operate at a granularity of an entire file or data object

Outline

- Data Deduplication
- **Data Compression**
- Case Studies

Data Compression

- **Lossless versus lossy compression**
  - Lossless compression means that no information is lost when a file is compressed and then uncompressed
  - Lossy compression usually results in better compression ratio, but some information is lost

- **For data storage, lossless compression is mainly used**
  - LZ1-Based algorithm combined with Huffman encoding
    - LZ1 algorithm to identify matches in a sliding window history buffer
    - A post encoder to Huffman encode the matches and literals
  - Hardware and software implementations

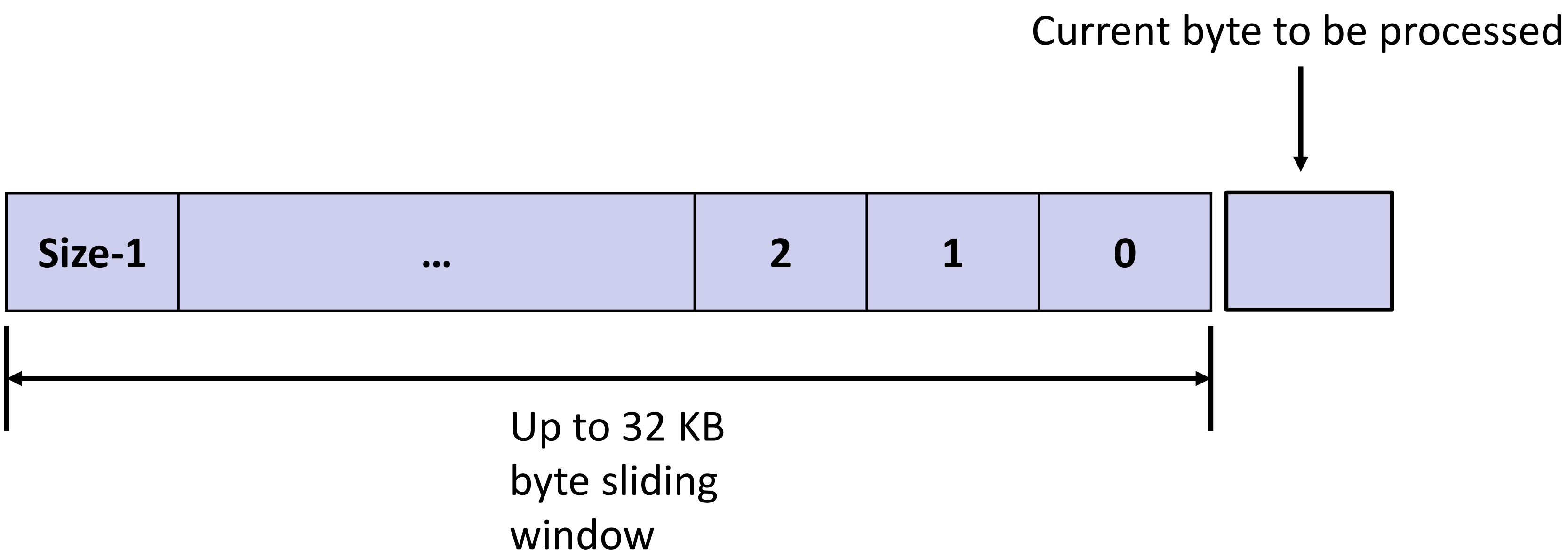
LZ1 Architecture

- A ***String-Matcher*** searches a History-Buffer to find repeating strings of bytes

- A sliding-window ***History-Buffer*** adds one new byte and drops off one byte from the back end of the buffer

- A ***Post-Encoder*** is a prefix encoder
  - Use statistics to encode the most common string matches with a smaller number of bits

LZ1 Algorithm and History Buffer



LZ1 String Matching

Input String: ABCDABCFCDAB…..

Kept in the history buffer

Input	Output
A	A
B	B
C	C
D	D
ABC	Distance=4, Length=3
F	F
CDAB	Distance=6, Length=4

Encoded string

Huffman Encoding

Input String: A B B C B A D B

Probability of Occurrence

Input character	Probability
A	0.25
B	0.5
C	0.125
D	0.125

Huffman Encoding (Cont.)

```
      /\
     0  1
    /     \
   B     / \
        0    1
       /       \
      A       / \
             0    1
            /       \
           C         D
```

Symbol	Code	Pr
A	10	0.25
B	0	0.5
C	110	0.125
D	111	0.125

- Reduction =

  $\frac{1}{2}[0.25(2) + 0.5(1) + 0.125(3) + .125(3)]$
  $= 0.875$

- Reduction in data size due to Huffman encoding.

Data Dependent

- Random data provides poor compression ratio performance
- Data with repeating byte strings, 2-byte or longer provides grater compression ratio performance
- Compression ratio greater than 100:1 are possible
- May expand if attempting to compress previously compressed data, but a system could detect this and send original data without compression

Algorithm Dependent

- Size of sliding window
- Static or dynamic Huffman encoding
- Number of matches tracked
- Length of matches the algorithm will search for

Hardware Implementation

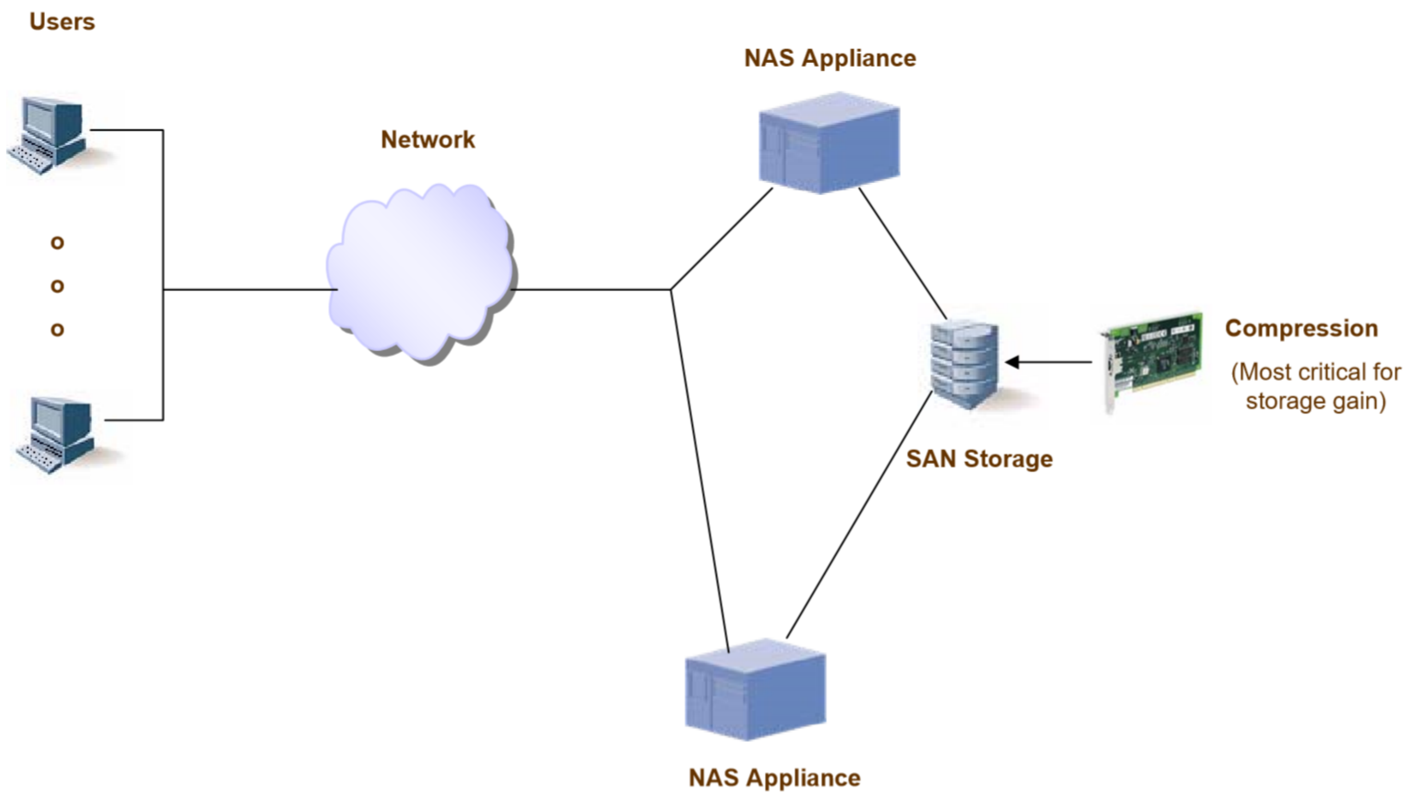
- **Software compression may result in the degradation of I/O throughput and latency as well as energy consumption**

- **Hardware compression**
  - (+) Higher data rate throughput (10x)
  - (+) Free up valuable CPU bandwidth
  - (+) reduce power consumption
  - (+) Speed up a network link by sending shorter files
  - (–) Lower compression ratio

- **Several hardware-specific compression algorithms are available**
  - Hardware implementation of GZIP algorithms
  - X-Match PRO: Real-time compression / decompression

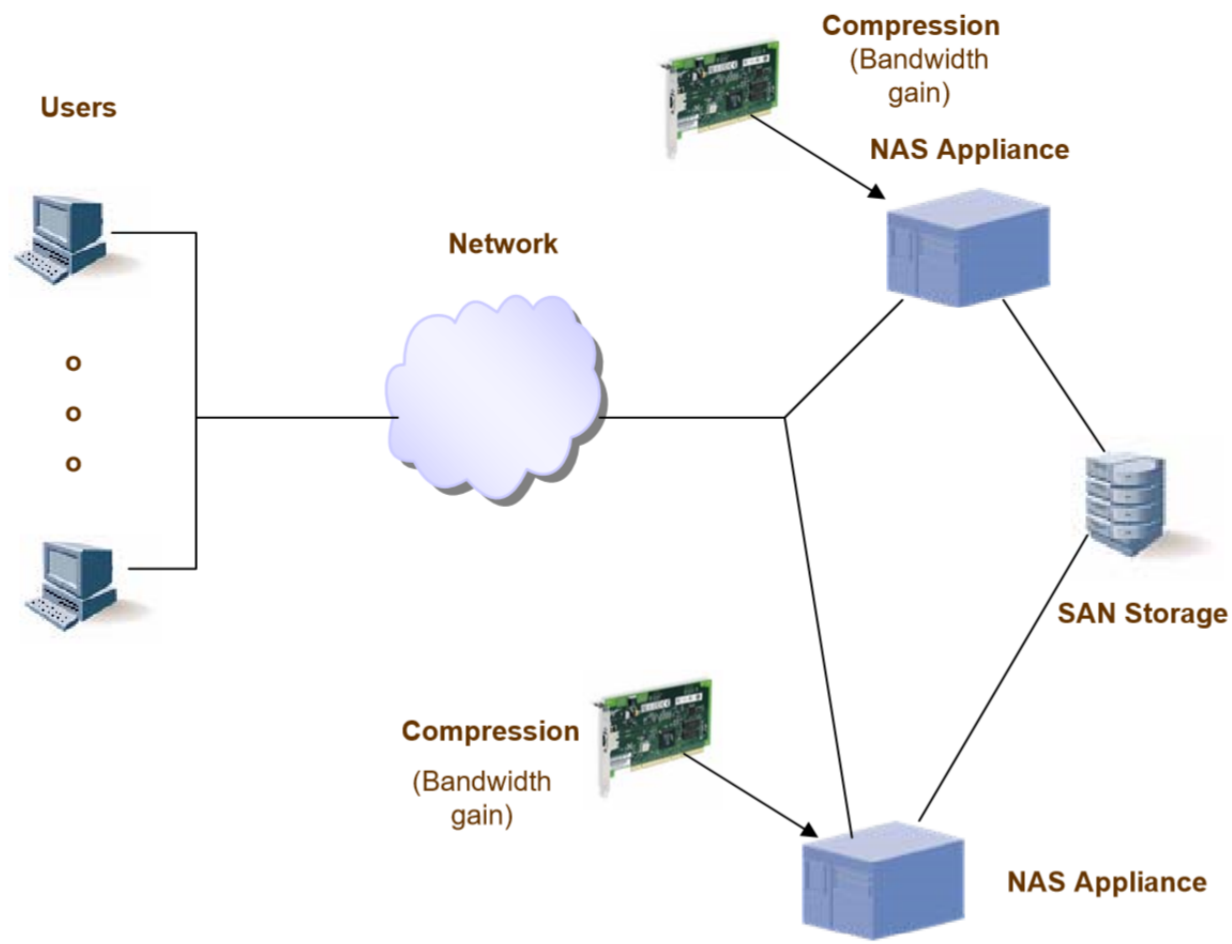
Deployment of Compression

- Similar to deduplication; combined with deduplication



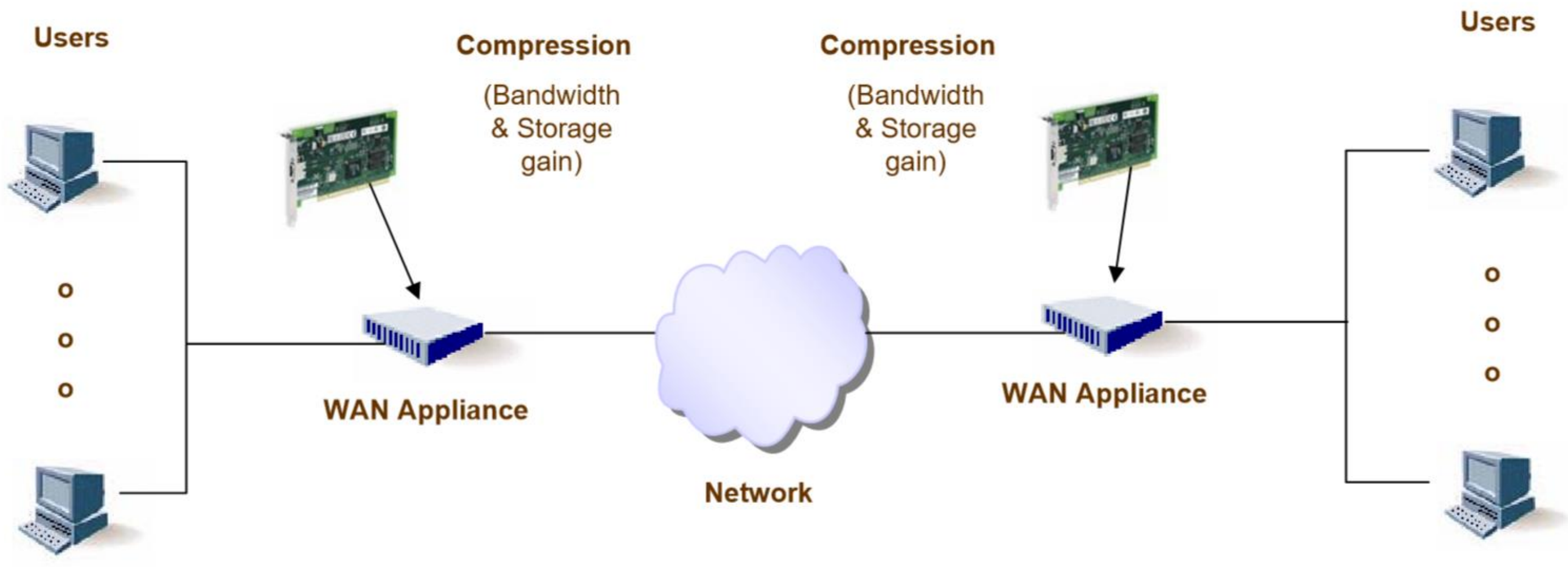
Deployment of Compression (Cont.)

- Similar to deduplication; combined with deduplication



Deployment of Compression

- Similar to deduplication; combined with deduplication



Data Deduplication vs Data Compression

- **Data deduplication**
  - (+) Large space saving is possible for exactly matched segments or files
  - (–) Even a single-bit difference makes deduplication ineffective
  - (–) Unable to get rid of repeated bit patterns of segments or files

- **Data compression**
  - (+) Effectively remove repeated bit patterns from input data streams
  - (–) Do not provide a high compression ratio all the times

- **Common design practice**
  - Use deduplication for input data streams
  - Use lossless compression for storage of duplicate data and remaining meta data

Outline

- Data Deduplication
- Data Compression
- **Case Studies**
  - **CA-SSD: Content-aware Solid-state Drives**
  - BlueZIP: Hardware-accelerated Compression
  - DAC: Dedup-assisted Compression

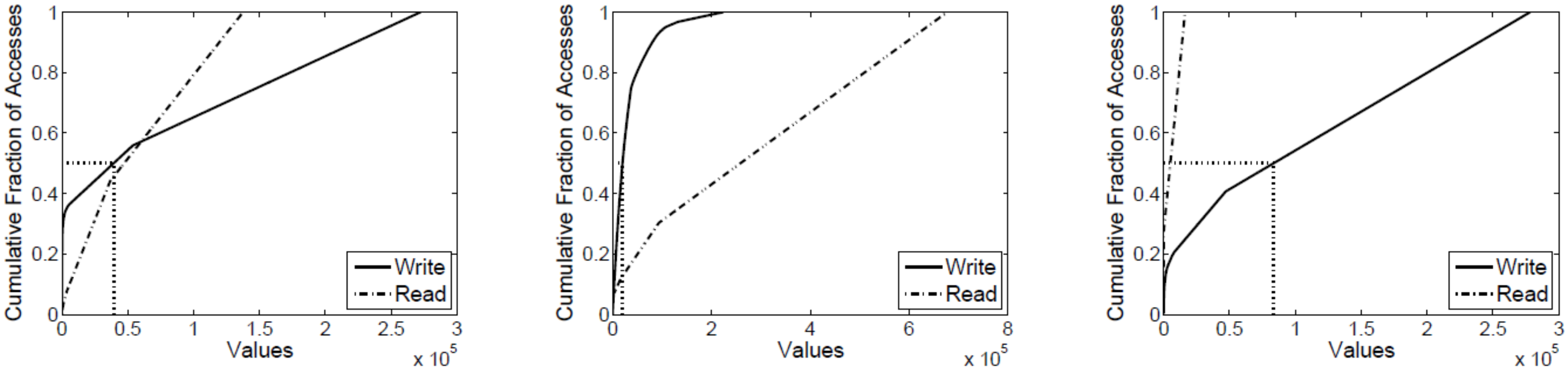
Value Locality

- **To improve the reliability and performance of SSD, many FTL schemes exploit temporal/spatial locality**
  - Temporal locality: buffering writes to eliminate duplicate writes
  - Spatial locality: coalescing multiple sub page writes into fewer page writes

- **However, another form of locality, *Value Locality*, has not been completely unexplored**
  - Value locality: certain content is accessed preferentially

Value Popularity (VP)

- Value popularity: the number of occurrences of each unique value
- Value popularity in real-word workloads



(a) web (b) mail (c) homes

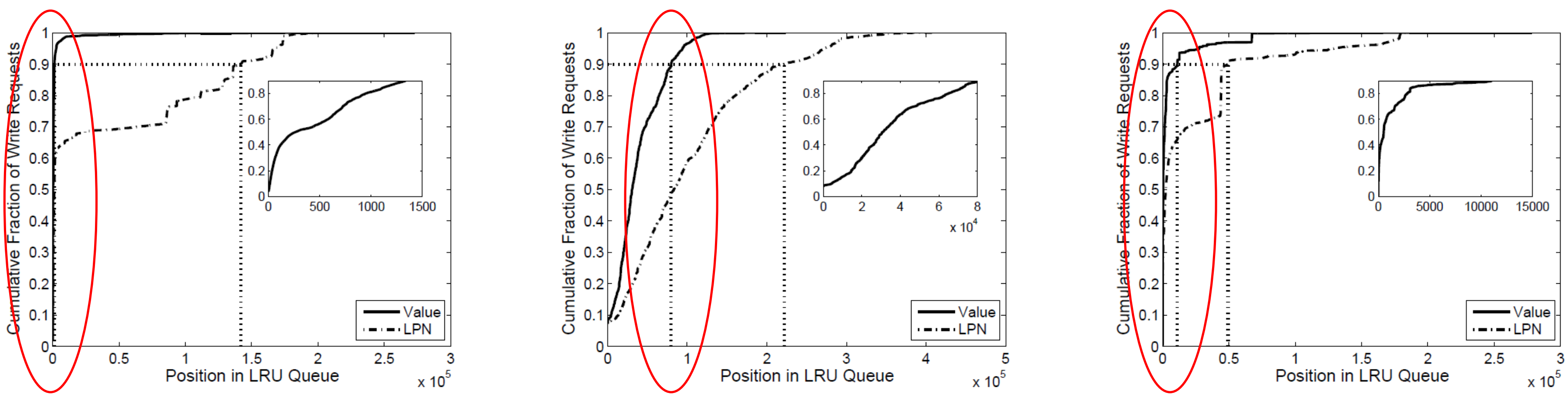
- Workload statistics

Workload	Size (GB)	% Writes	Req. (mill.)	Unique Request (%) Write	Unique Request (%) Read	Seq. %
web	1.95	77.01	3.8	42.35	32.05	83.8
mail	4.22	77.32	3.6	7.83	80.85	94.7
homes	3.02	96.76	4.4	66.37	80.75	70.8

Temporal Value Locality (TVL)

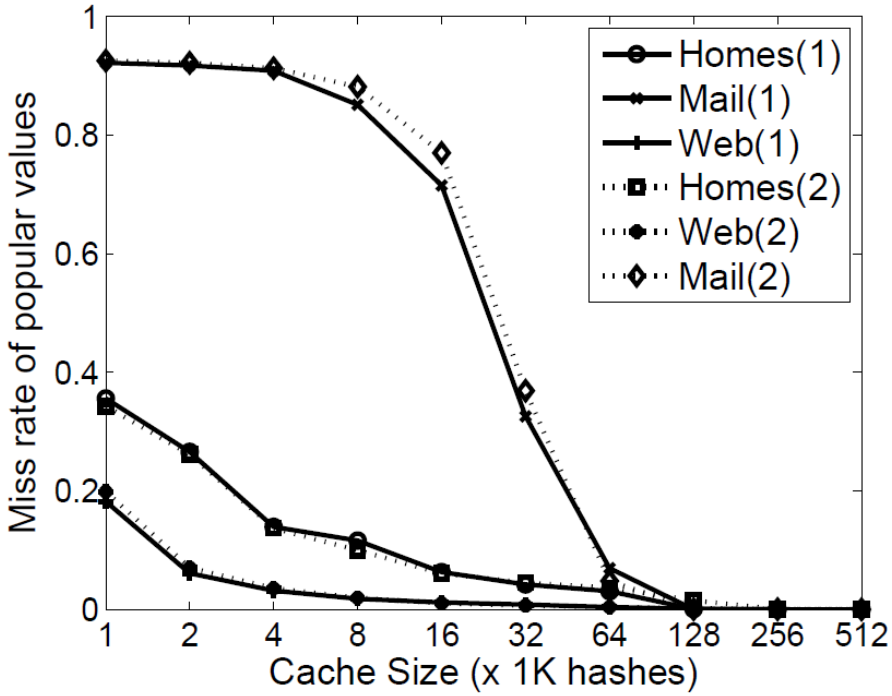
- **Temporal Value Locality**: If a certain value is accessed now, it is likely to be accessed again in the near future

- **Temporal value locality in real-world workloads**
  - The metadata cache is managed as a queue with an LRU eviction policy
  - CDFs of number of writes of the value at the (i+1)st location in the LRU queue



(a) web (b) mail (c) homes

Cache Miss Rate for Popular Values



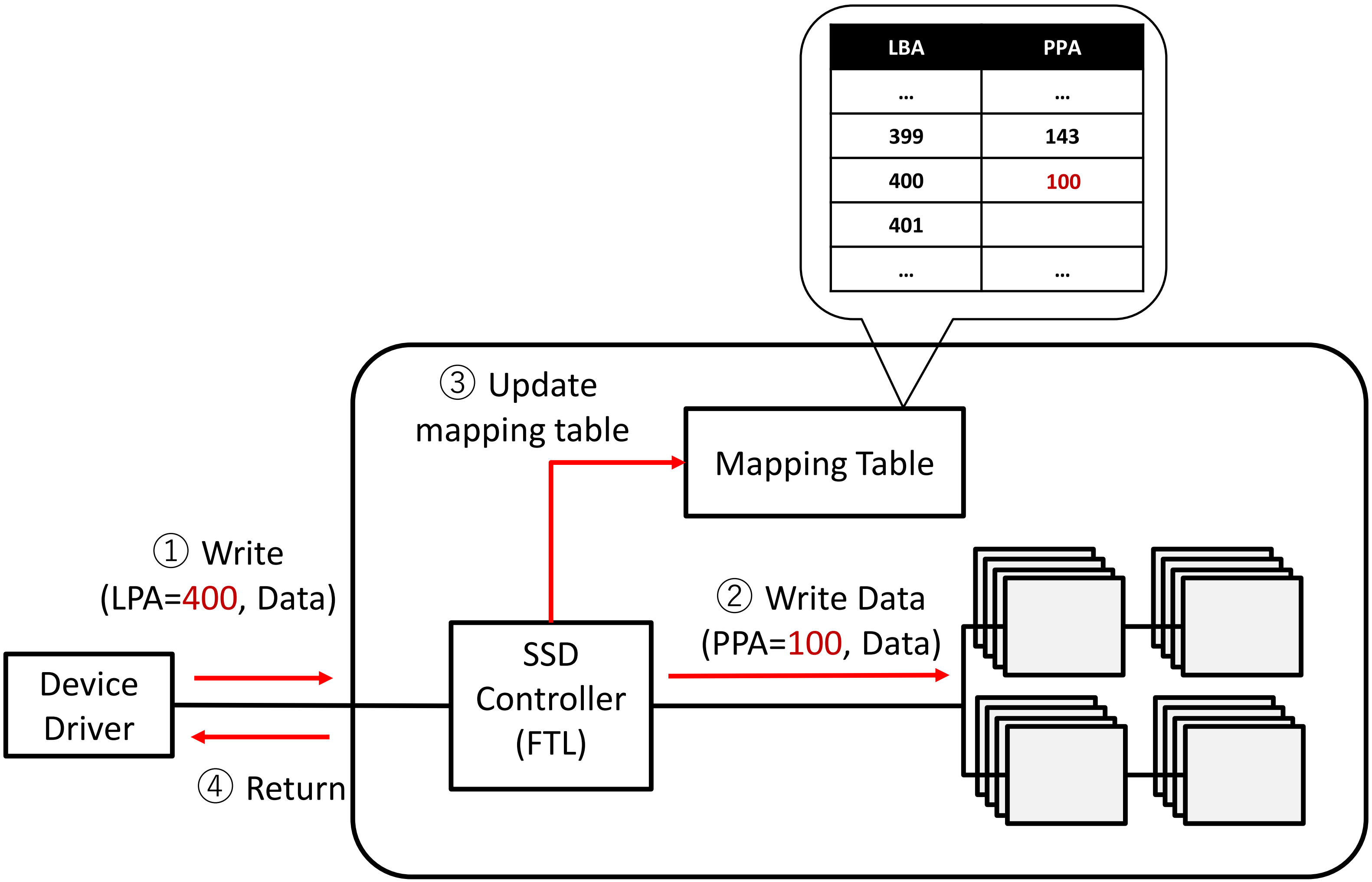
- Popular values represent the minimum number of values which account for 50% of accesses
- All the three traces achieve about 90% hit rate with 1.75 MB metadata cache (64 K hashes X 28 B = 1.75 MB)

Content-Aware SSD

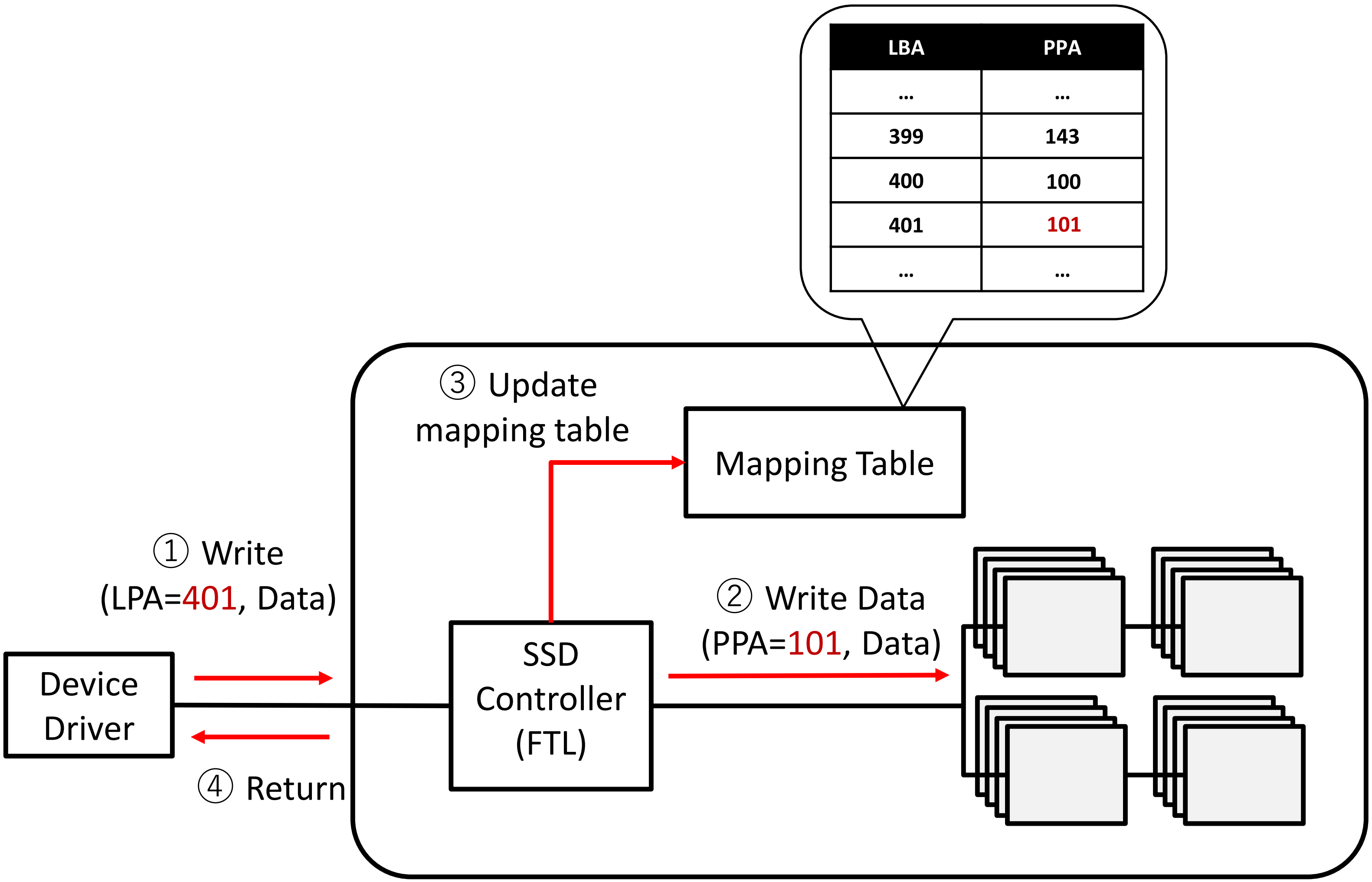
- **The presence of value locality in a workload means that it preferentially accesses certain content over others**

- **This property facilitates data de-duplication inside an SSD**
  - Store only a non-intersecting chunk having *a unique hash value*
  - Other data blocks with the same content point to the unique block

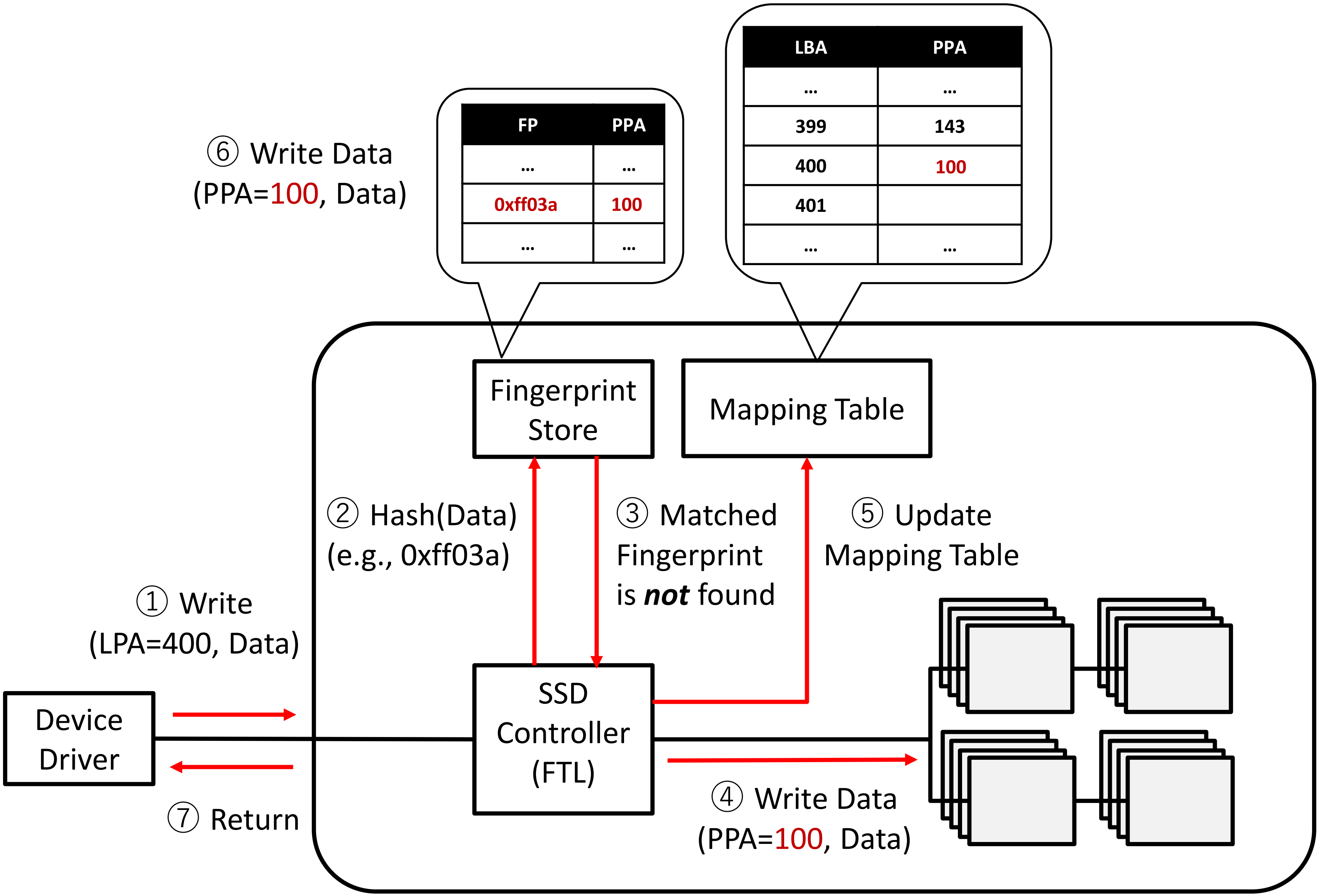
How a Conventional SSD Works?



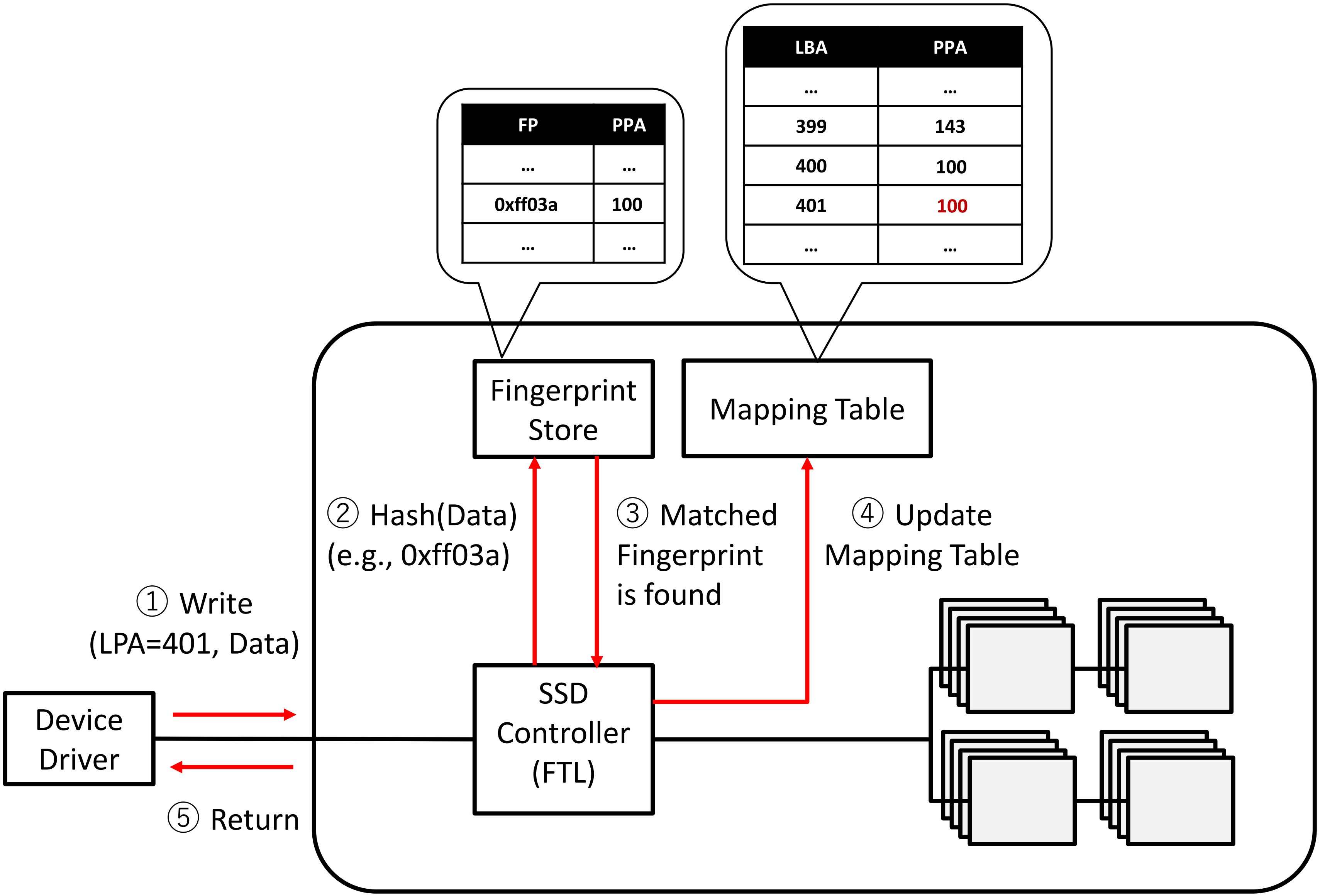
How a Conventional SSD Works?



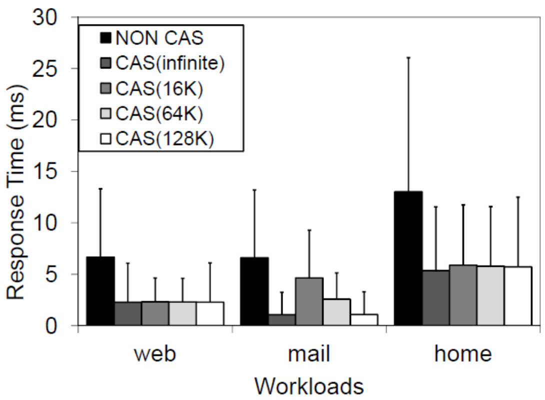
How CA-SSD Work? (Simplified)



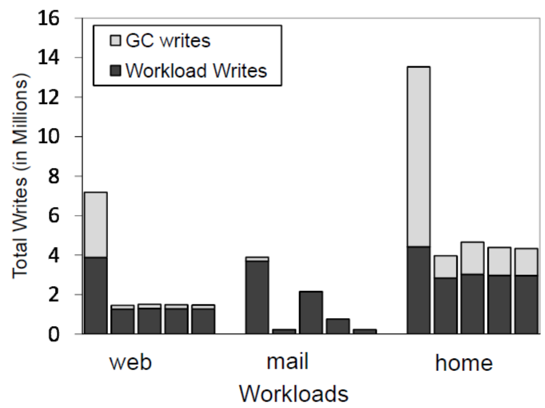
How CA-SSD Work? (Simplified) (Cont.)



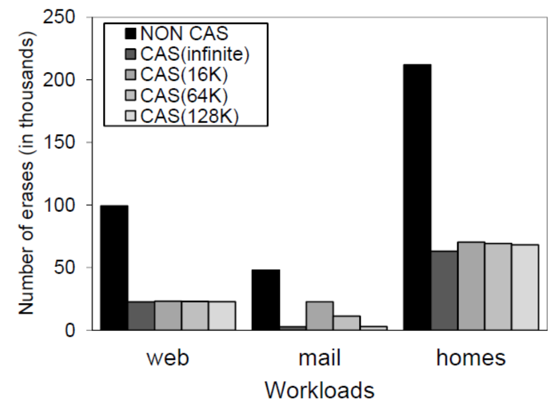
Experimental Results



(a) Response Time



(b) Total Writes



(c) Block Erases

- The reduction in write traffic: 77%, 93%, and 70% for web, mail, and home
- The write reduction benefits directly translate into the reduced response time and the reduced block erases
- CAS(128K) provides the performance close to the CA(infinite)
  - mail shows lower TVL and requires a larger metadata cache

Outline

- Data Deduplication
- Data Compression
- **Case Studies**
  - CA-SSD: Content-aware Solid-state Drives
  - **BlueZIP: Hardware-accelerated Compression**
  - DAC: Dedup-assisted Compression

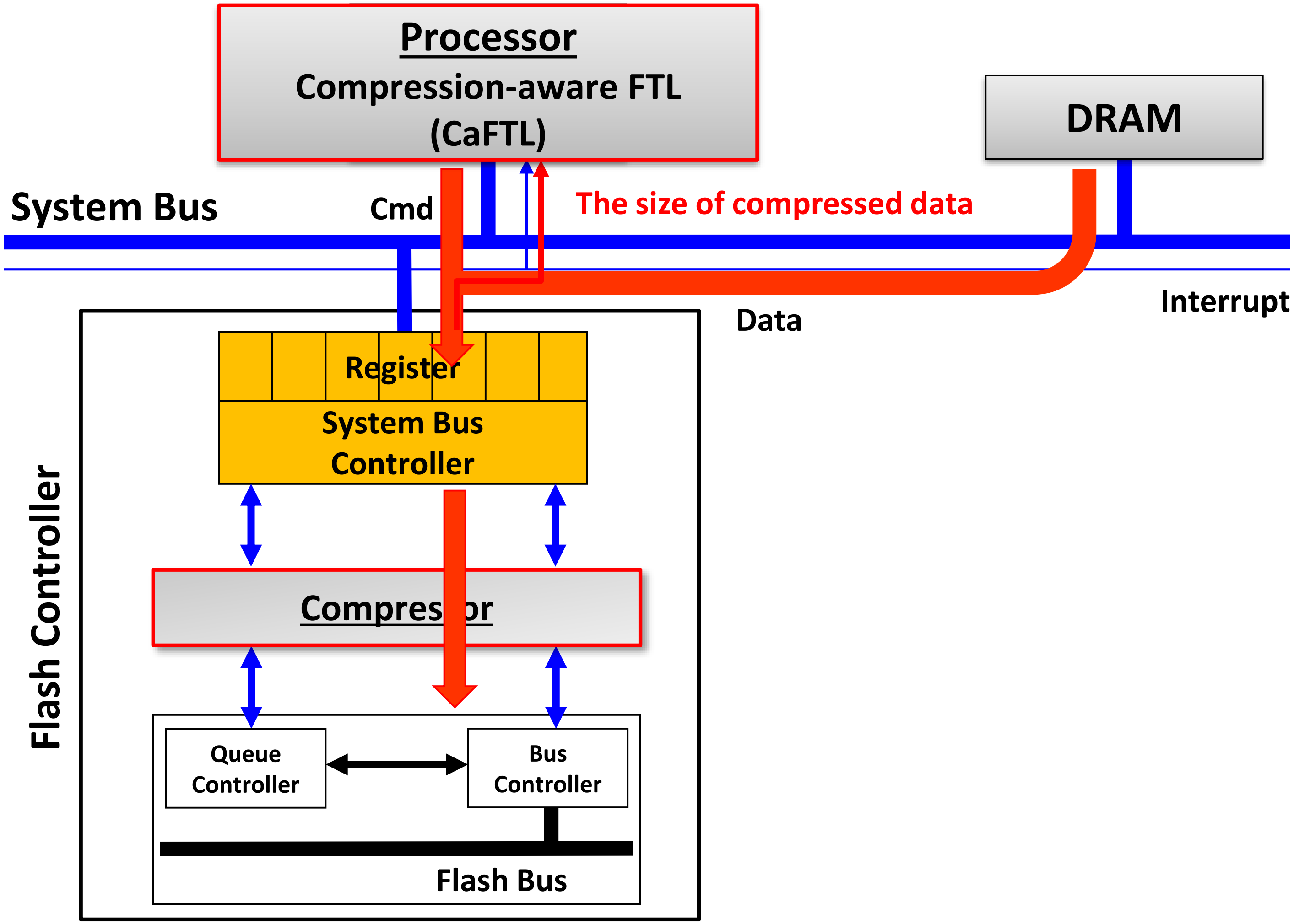
BlueZIP: Hardware-accelerated Compression

- **Requested data contain lots of repeated bit patterns**
  - Lossless compression helps to eliminate such repeated bit patterns.

- **BlueZIP: A hardware-accelerated compression technique,**
  - Compress requested data at runtime
    - Use a hardware-accelerated compression module
    - Provide software support for maximizing the benefits of hardware-accelerated compression
  - Improve the lifetime and performance of storage devices
    - Reduce the amount of data written to flash memory
    - Reduce the time taken to transfer data between a host system and flash memory

Overall Architecture of BlueZIP

- Implement the hardware compression module inside the flash controller to reduce system bus traffic

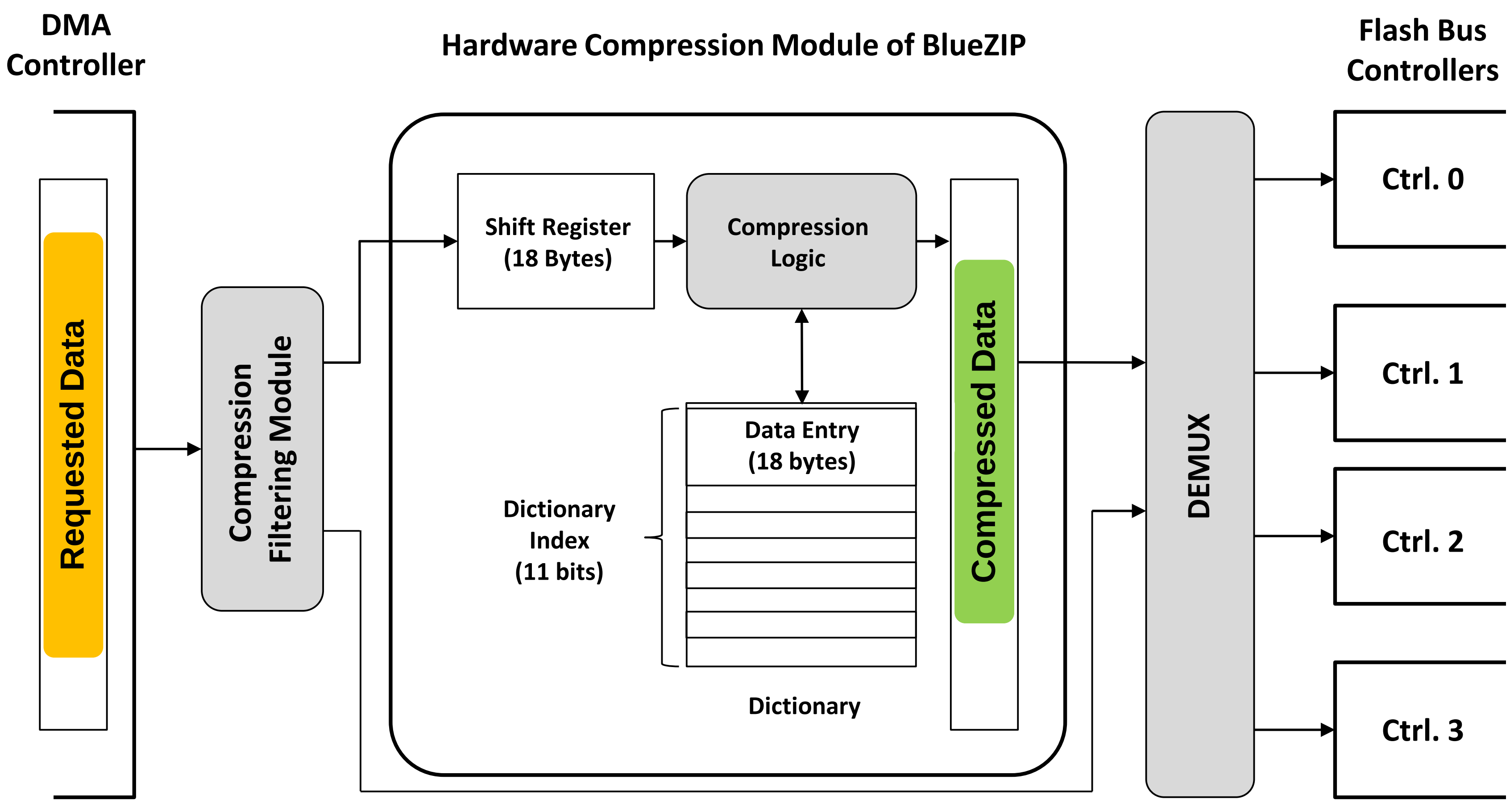


HW: Compression Algorithm

- **Use the LZRW3 compression algorithm**
  - Relatively high compression ratio
  - Easy hardware implementation

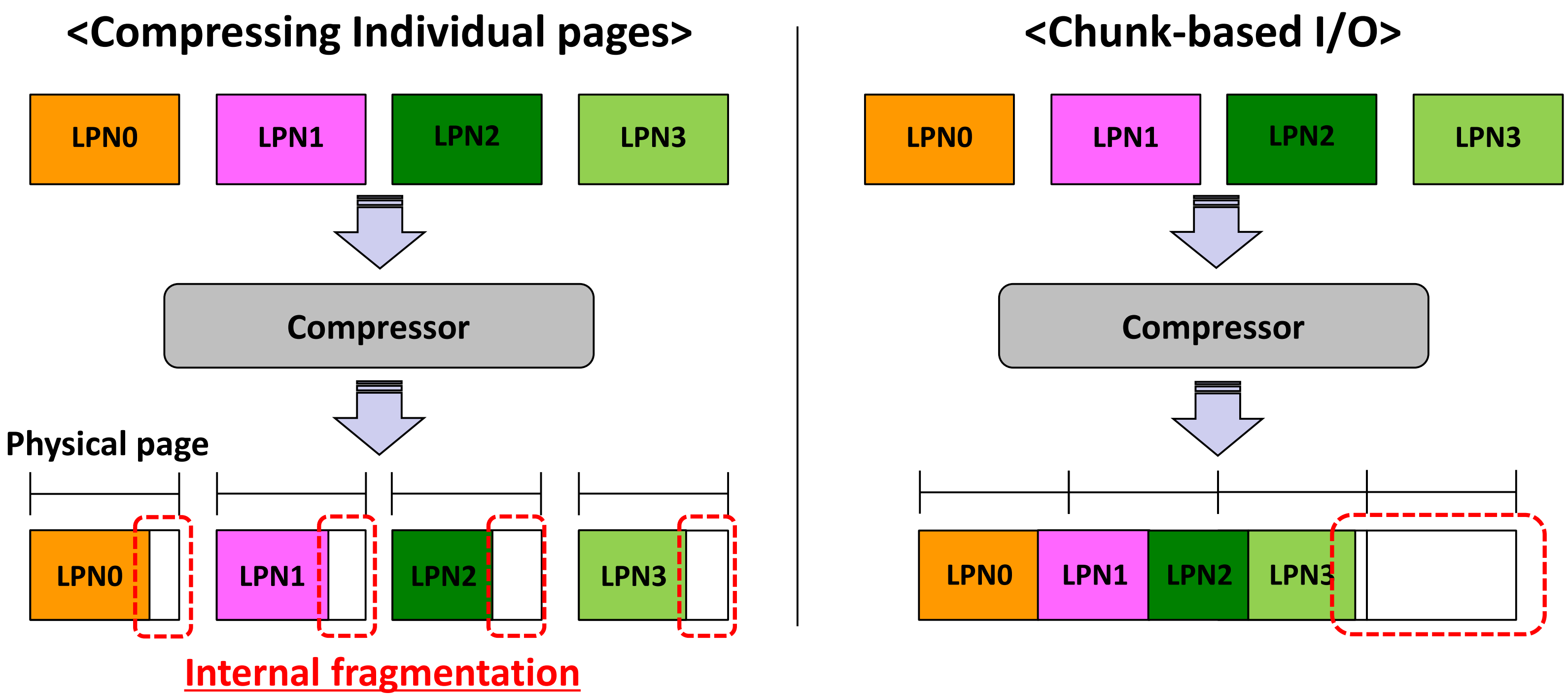
Compression Algorithm	Hardware Complexity	Performance (Cycles)	Compression Ratio	Description
X-Match	Low	1,024 cycles / 4 KB (20.48 us@ 50 MHz)	Low (e.g., 20%)	Hardware-Based Memory Compression
LZ77	High	4,096 cycles > / 4 KB	High (e.g., 50%)	File Compression
LZRW3	Middle	4,096 cycles / 4 KB (81.92 us@ 50 MHz)	High (e.g., 40%)	File Compression

HW: Implementation



HW-SW: Compression Unit

- **Chunk-based I/O**
  - Compress several logical pages into several physical pages
  - Mitigate the internal fragmentation problem
- **Read performance penalty**
  - Use a read buffer that holds previously decompressed pages



SW: Compression-Aware FTL

Write Buffer

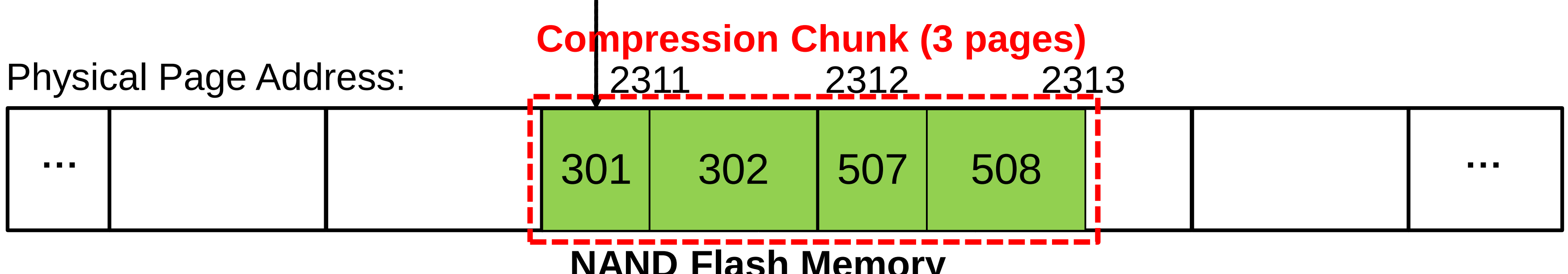
301	302	507	508	...

Page Mapping Table

Logical Page Address	Physical Page Address
⋮	⋮
301	2311
302	2311
507	2311
508	2311

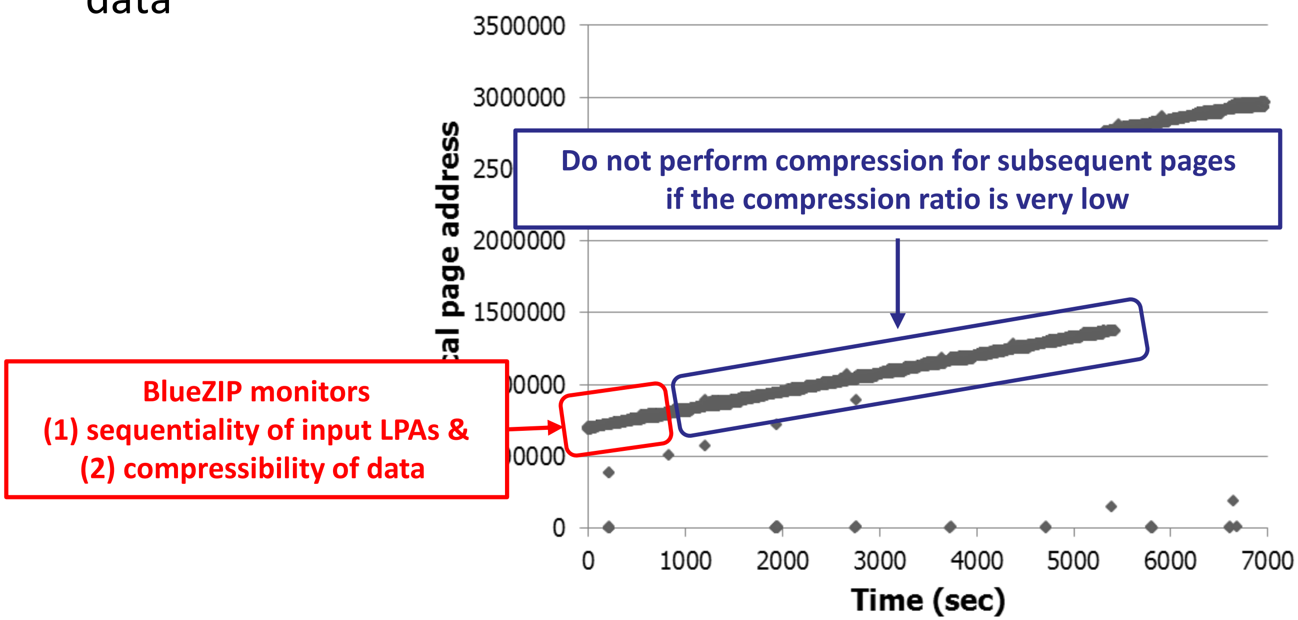
Data Chunk Table

Physical Page Address	Valid Page Counter	No. of Physical Pages	Compression Indicator
⋮	⋮	⋮	⋮
2311	4	3	1
2312	4	3	1
2313	4	3	1
	⋮	⋮	⋮



SW: Selective Compression

- **The data size expansion problem**
  - The size of compressed data > the size of original data
  - Degrade storage performance and lifetime
  - Usually observed in writing multimedia files
- **Selective compression**
  - Detect poorly compressed sequential writes in advance and do not compress those data



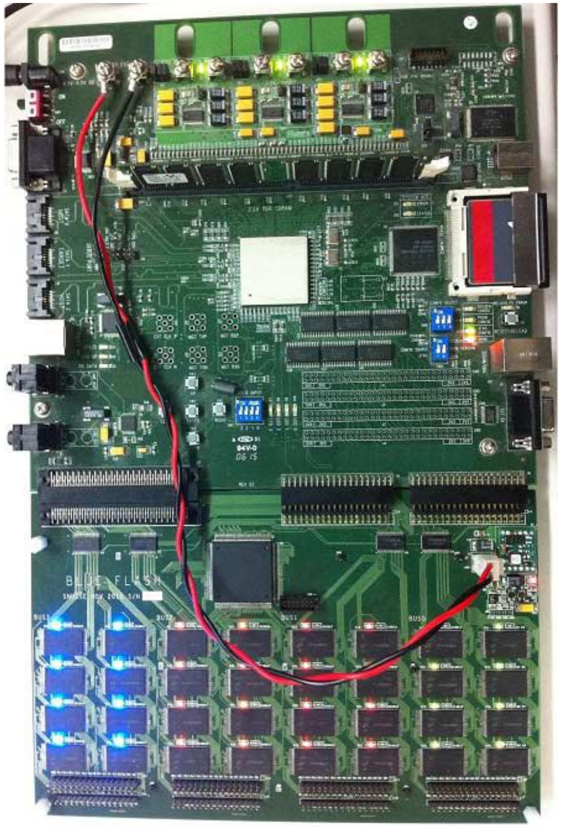
Experimental Settings

Benchmarks

Benchmark	Description	Compression Ratio
SENSOR	A set of sensor data files which were collected during a semiconductor fabrication process	Very high
LINUX	A subset of the Linux kernel 2.6.32 source files	high
DOCUMENT	A set of documents and image files	medium
MP3	A set of MP3 files already highly compressed	Very low

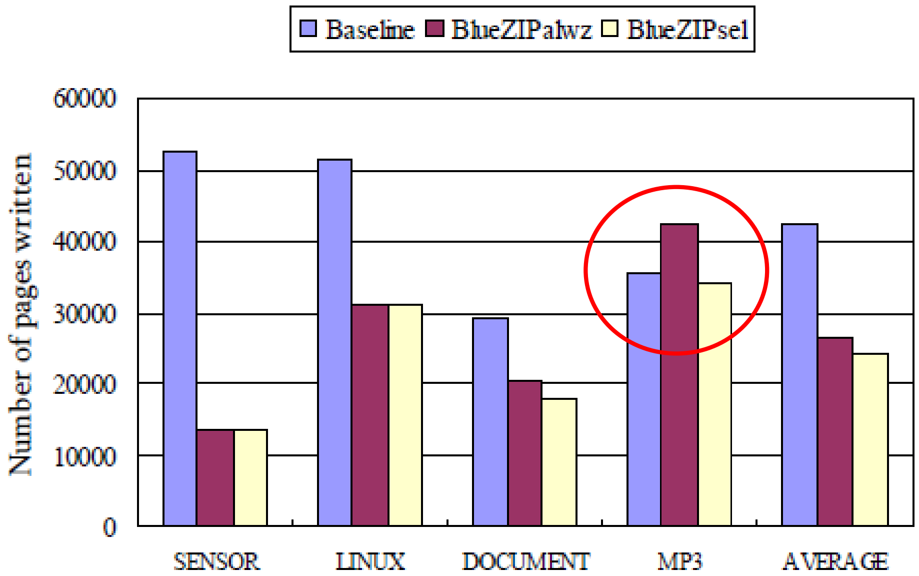
SSD Configurations

Configurations	Description
Baseline	Use no lossless compression
BlueZIPalwz	Use lossless compression all the time
BlueZIPsel	Use selective compression

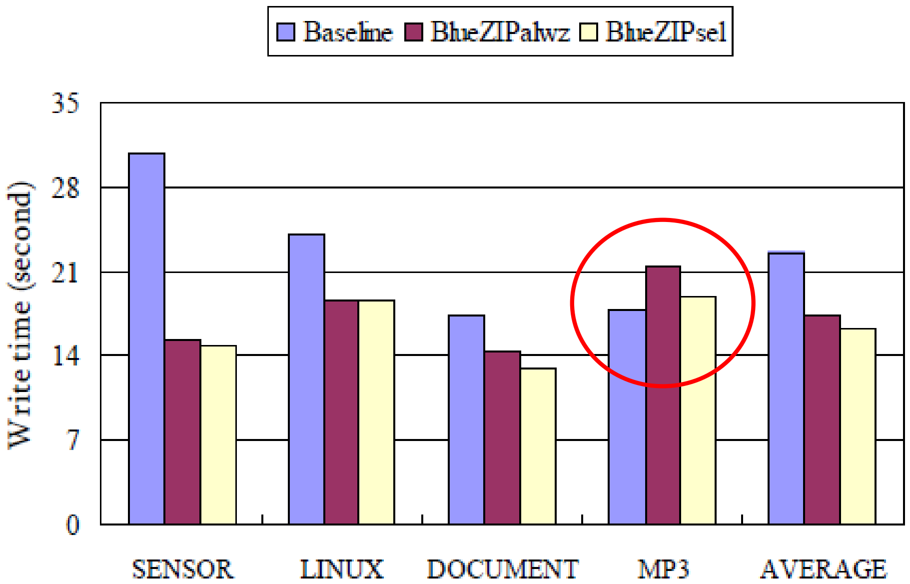


<BlueZIP prototype>

Experimental Results



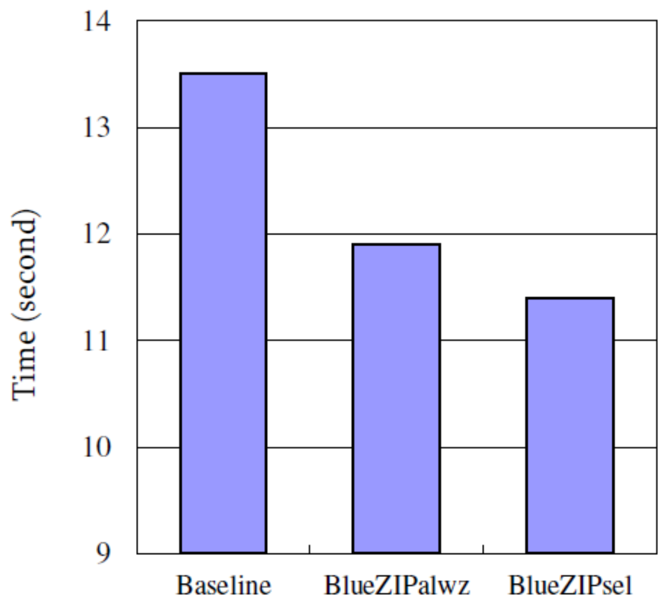
The number of pages written



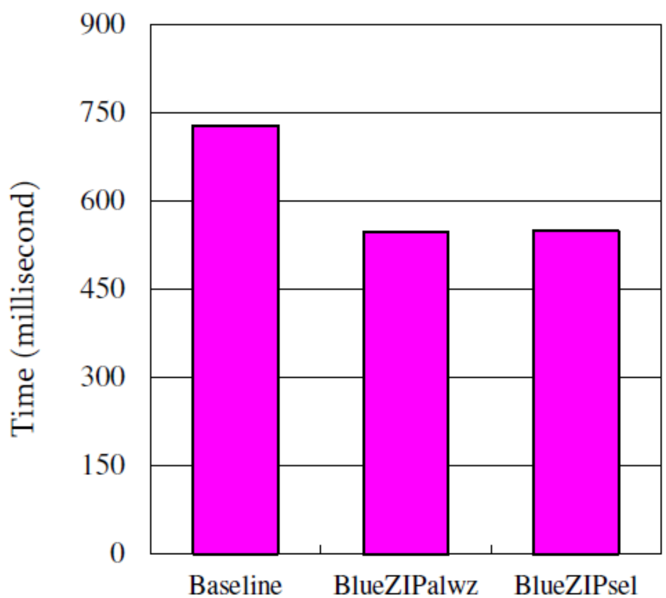
Time (second)

- **BlueZIPsel writes 38% less data over Baseline.**
  - The amount of written data is increased with BlueZIPalwz due to the data expansion problem of lossless compression.
- **BlueZIPsel achieves 17%-50% higher performance than Baseline**

Read Performance



Grep



Find

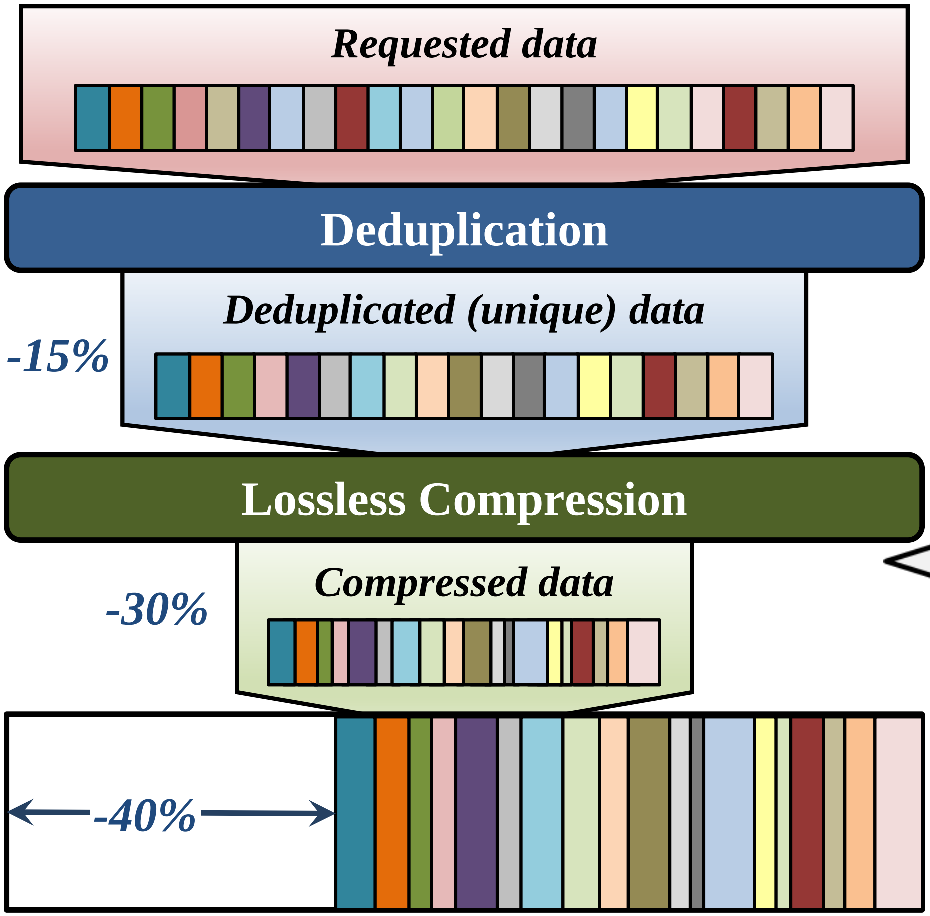
- BlueZIPsel improves the overall read performance by 20% on average.
- The reduction in the number of pages sufficiently offsets the decompression overhead.

Outline

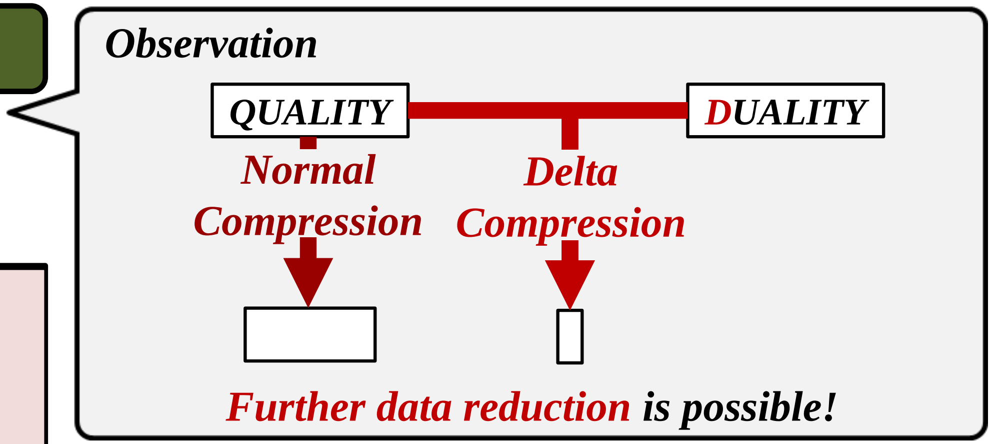
- Data Deduplication
- Data Compression
- Case Studies
  - CA-SSD: Content-aware Solid-state Drives
  - BlueZIP: Hardware-accelerated Compression
  - DAC: Dedup-assisted Compression

Integration of Capacity-optimization Techniques

- **Maximizing the data compression ratio**
  - Improvements can be accumulated

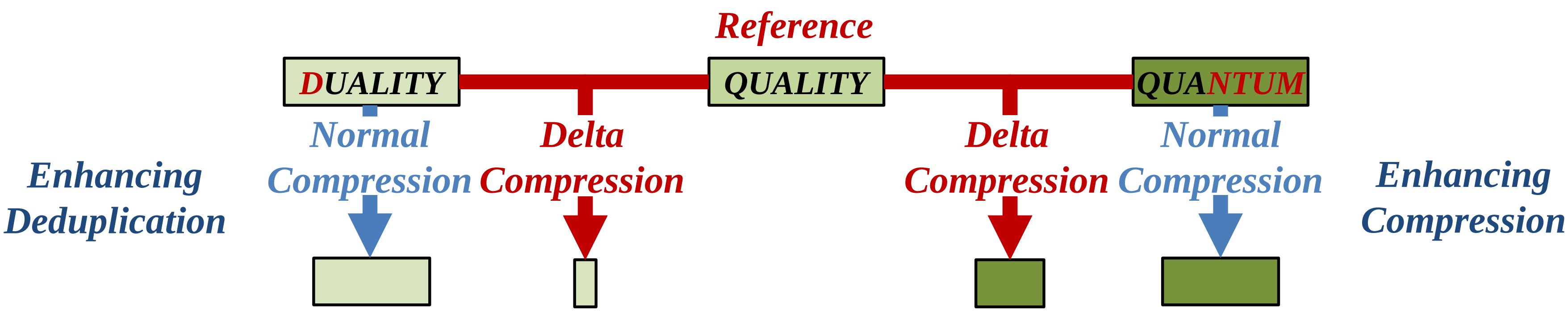


Q: Can we do better?

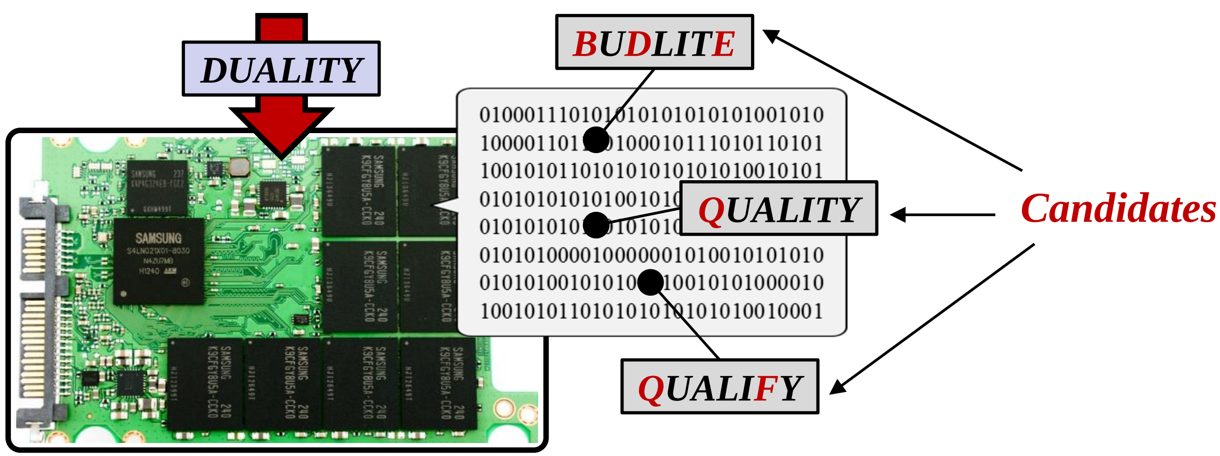


DAC: Dedup-Assisted Compression

- **A novel integrated data reduction technique**
  - Exploiting data similarity to further increase data reduction ratio

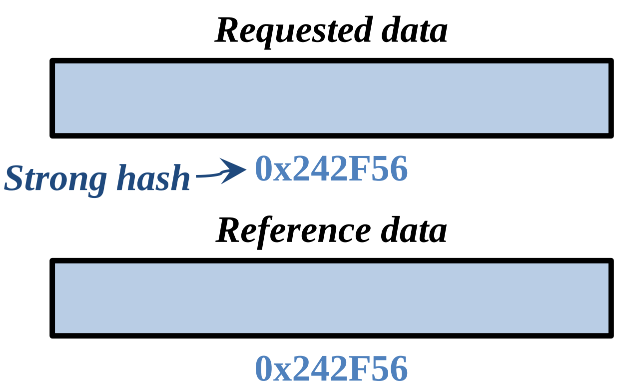


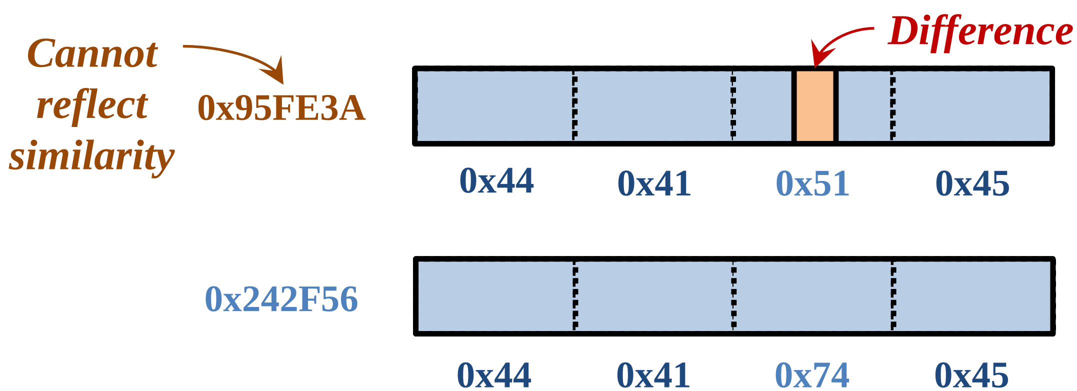
- **Reference search is not easy**



DAC: Reference Data Search

- **Exploiting partial fingerprints (hashes) of data**
  - Generated by the strong hash module for deduplication (Dedup-Assisted)





Fingerprint (FP) Store

Page FP	Page Addr
…	…
0x141250	0x022
0x242F56	0x0F1
0x44516F	0x011
0x6DFF25	0x1CB
…	…

Deduplication

Sub-Page Fingerprint Store

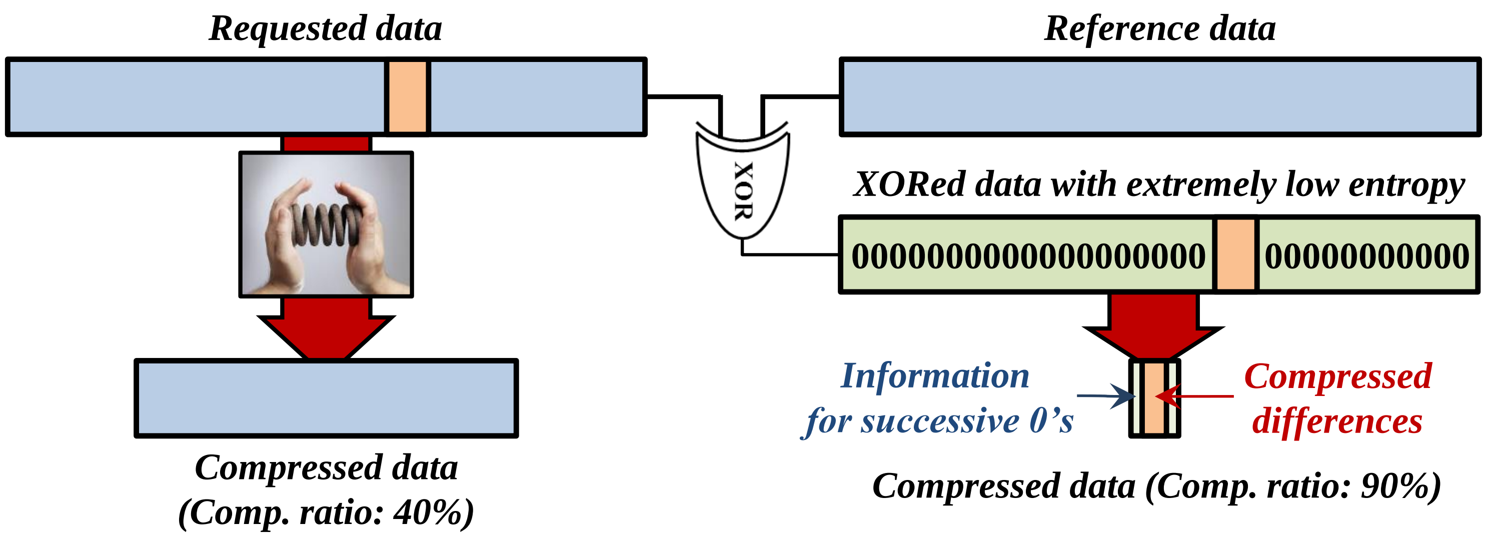
	1st FP	2nd FP	3rd FP	4th FP	Page Addr
	…	…	…	…	…
50%	0x44	0x2F	0x51	0x7E	0x02F
75%	0x44	0x41	0x74	0x45	0x0F1
25%	0x44	0x5E	0x3C	0x7A	0x012
0%	0x45	0x0D	0x45	0x6D	0x13B
	…	…	…	…	…

Dedup-Assisted Compression

Wide searching range, constant searching time, and quantified data similarity

DAC: Data Compression with a Reference

- **Exploiting XOR logic to extremely reduce data entropy**

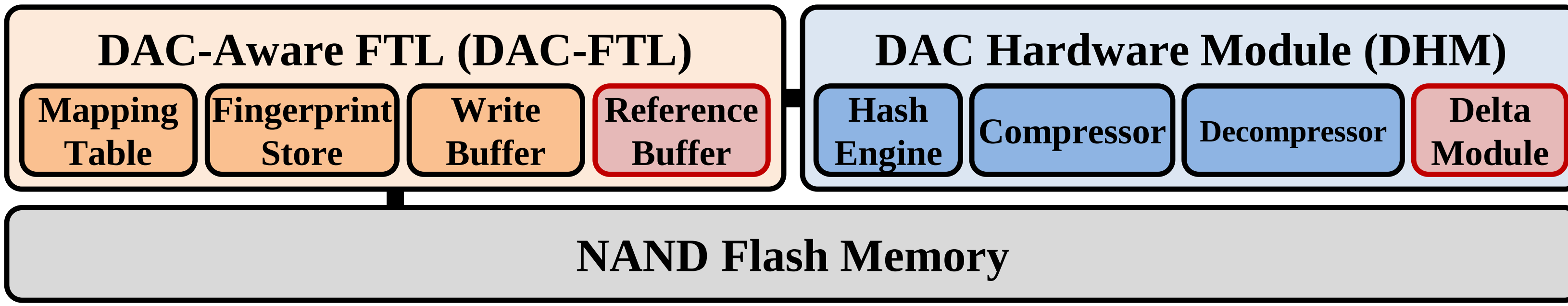


- **Lossless compression algorithm: XmatchPRO**
  - Employing both the run-length and dictionary-based data encoding
  - Easily implementable in H/W
  - High performance and comparable compression ratio

Higher compression efficiency w/o an additional delta-aware data encoding

Experimental Environment

- **Target SSD Architecture**
  - Recent high-end SSDs employing deduplication and compression

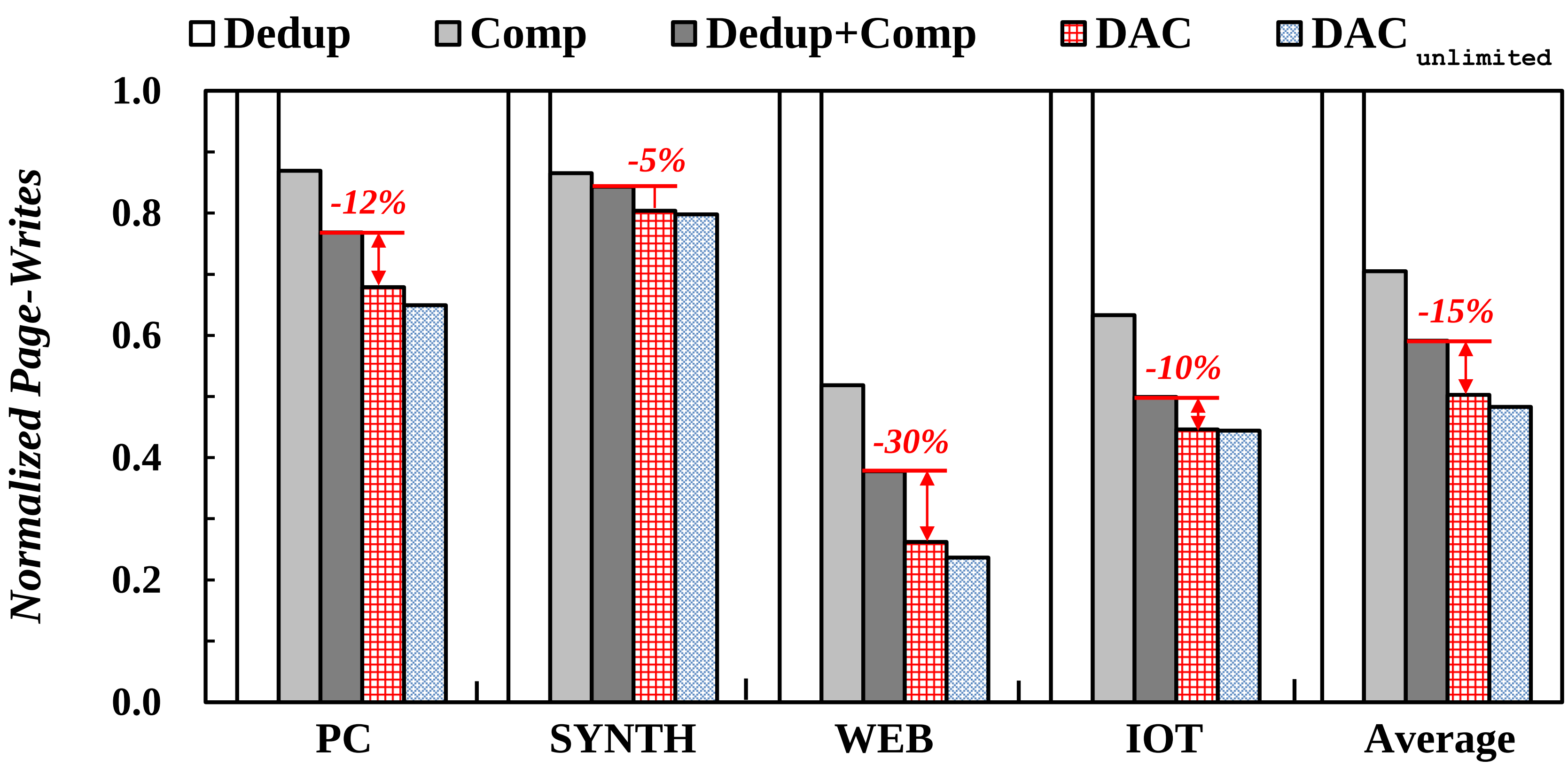


- **Evaluation on a storage emulation environment**
  - Hardware modules for a strong hash function (MD5) and lossless compression (XmatchPRO) were emulated by software.
- **Benchmarks: block I/O traces collected form a high-end PC**

Benchmark	PC	SYNTH	WEB	IOT
Data Redundancy	*Moderate*	*Low*	*Very High*	*High*
Data Compressibility	*Moderate*	*Low*	*Very High*	*High*

Evaluation Result

- Compared with a combination of deduplication and lossless compression, DAC reduces write traffic by up to 30% and 15% on average
- DAC can further increase data reduction whether data entropy is high or not



End of Chapter 10