CSCI4180 Tutorial-8: Assignment 3 Review (Part-1)

Zhao Jia

jzhao@cse.cuhk.edu.hk

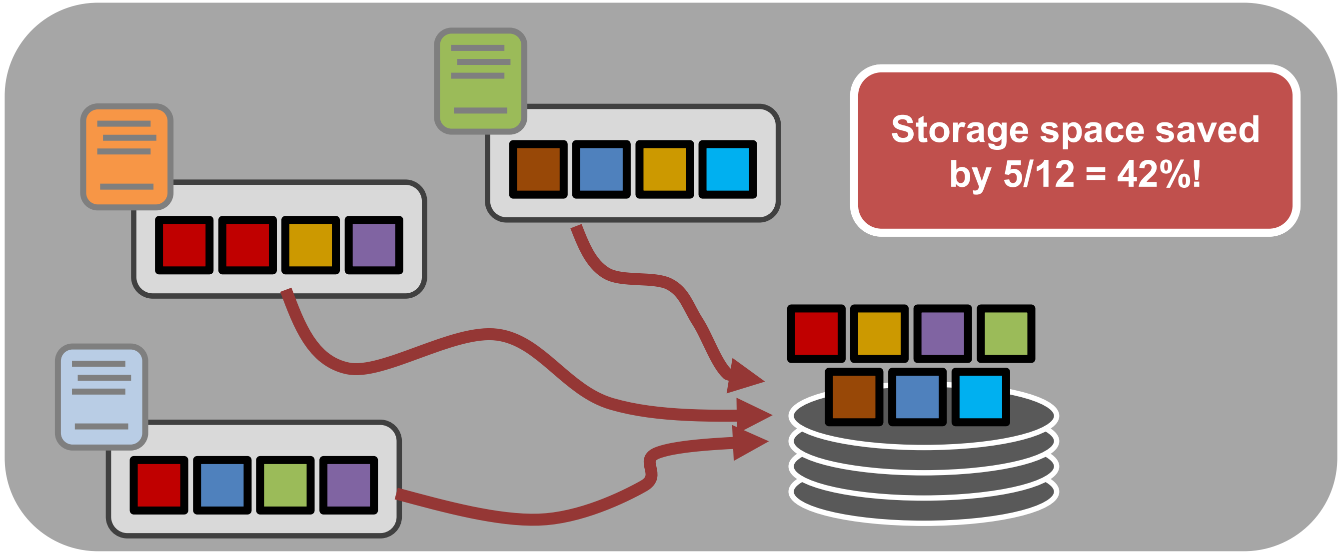
Nov. 23, 2022

Content

- Deduplication
- Variable-size chunking
- Checksum
- Indexing

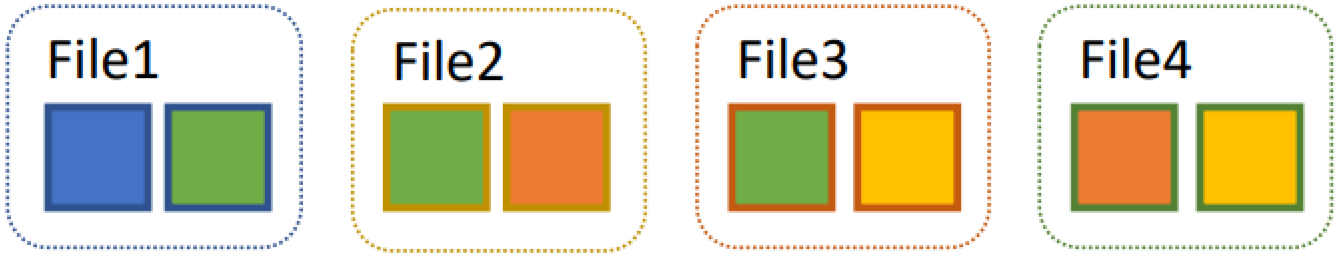
Deduplication

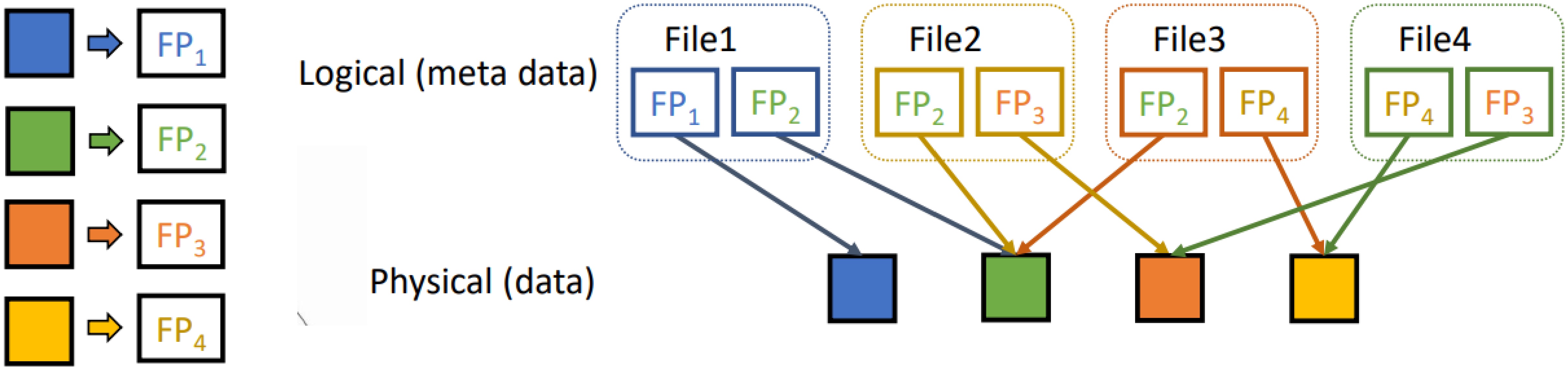
- Deduplication → Coarse-grained compression
- Unit: chunk: fixed-size or variable-size
  - compute a fingerprint
    - Same fingerprint → same content
- Store only one copy of chunks with same content; other chunks refer to the copy by refences (pointers)



Deduplication

➢ Deduplication → Coarse-grained compression





Deduplication (Cont.)

- Why “removing duplicate data”
  - Storage efficiency
    - Avoid storing the duplicated chunks in the storage backend
  - Reducing data transfer
    - Removing the duplicate data in the client side

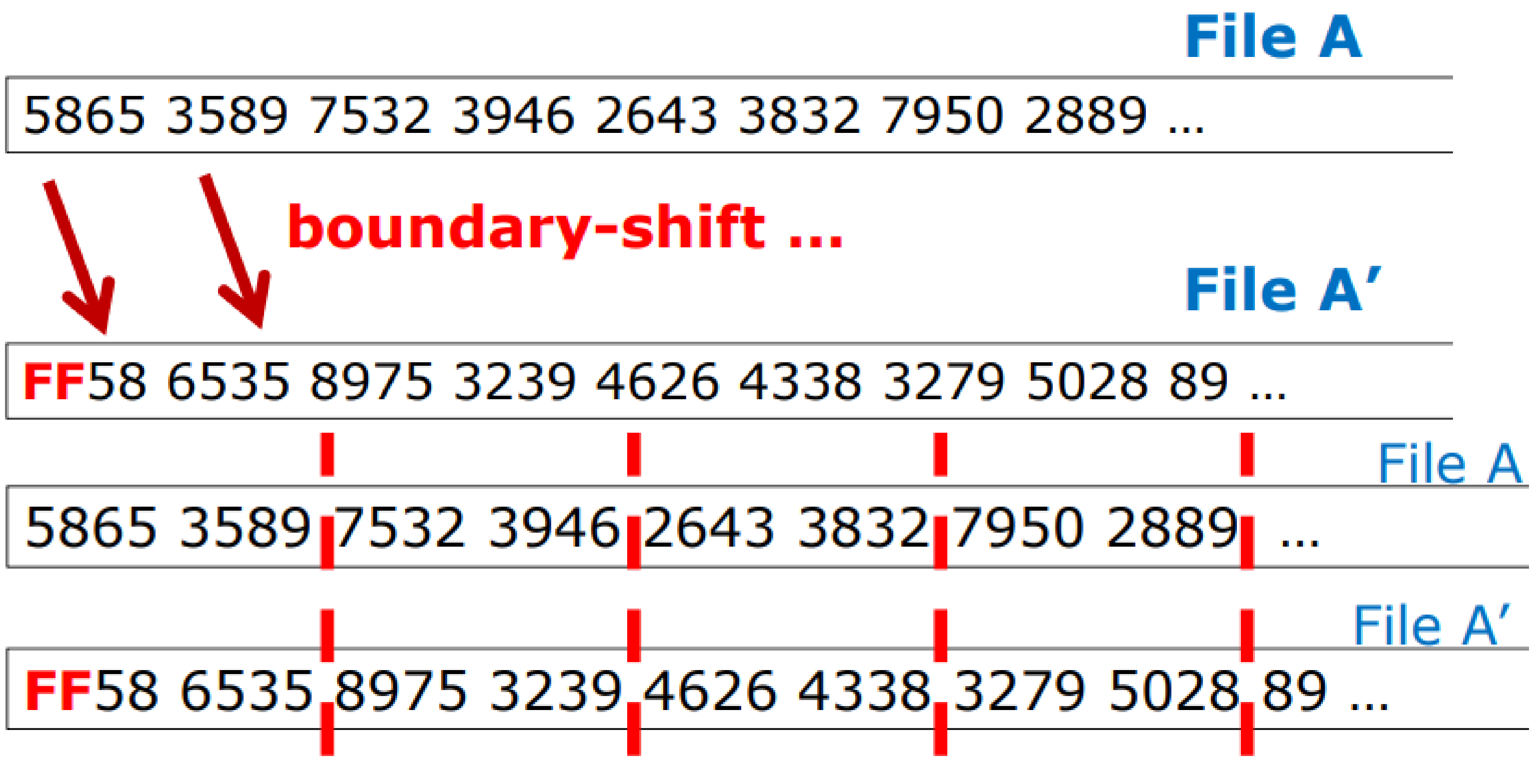
- Three main components
  - 1. Chunking: divide data into different chunks
  - 2. Checksum: compute checksum to identify each chunk
  - 3. Indexing: a search structure for efficient access of the information of chunks
    - Check whether a chunk is duplicated

Content

- Deduplication
- Variable-size chunking
- Checksum
- Indexing

Variable-size Chunking

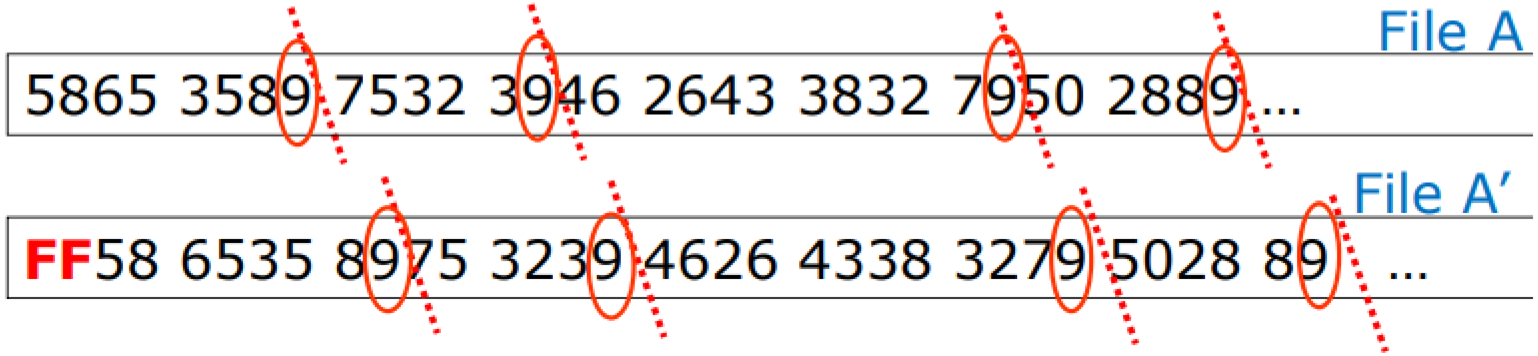
- Why not using fixed-size chunking (FSC)?
  - Boundary-shift problem
  - Vulnerable to data modification
    - Insert or delete



→→ ***FSC: No duplicates will be detected***

Variable-size Chunking

- Variable-size chunking
  - Content-defined chunking (CDC)
    - Identify each chunk according to its content
  - Rabin fingerprint algorithm
    - An efficient rolling hash



→→***CDC: Most duplicates will be detected***

https://en.wikipedia.org/wiki/Rabin_fingerprint

Rabin Fingerprint Algorithm

- What is fingerprint?
  - Fingerprint is the identifier of data
  - We use Rabin fingerprint algorithm to compute the fingerprint of the data
    - A method for implementing fingerprints using polynomials over a finite field.
  - Rabin fingerprint algorithm is a classical chunking algorithm, but it is not state-of-the-art.
    - Performance is not good
    - Faster chunking algorithm: FastCDC
    - But it is a good start point

https://en.wikipedia.org/wiki/Rabin_fingerprint

Rabin Fingerprint Algorithm (Cont.)

- Formula (How to compute Rabin fingerprint)

$$p_s(d,q) = \begin{cases} \left(\sum_{i=1}^{m} t_i \times d^{m-i}\right) \; mod \; q, & s = 0 \\ \left(d \times (p_{s-1} - d^{m-1} \times t_s) + t_{s+m}\right) \; mod \; q, & s > 0 \end{cases}$$

- Symbols
  - p_s: [result] The fingerprint we computes.
  - t_i: [data] Usually t_i is 1 byte of data.
  - m: [parameter] Window size (bytes).
  - d: [parameter] base.
  - q: [parameter] modulus

https://en.wikipedia.org/wiki/Rabin_fingerprint

Rabin Fingerprint Algorithm (Cont.)

- Example
  - Start of the file

1	**9**	**6**	**4**	**8**	**6**	**3**	**5**	**…**

  - Parameters
    - m = 4 (window size)
    - d = 10 (base)
    - q = 13 (modulus)

Rabin Fingerprint Algorithm (Cont.)

- Example
  - index = 0

1	**9**	**6**	**4**	**8**	**6**	**3**	**5**	**…**

  - Compute p_0

$p_0 = (t_1 * d^3 + t_2 * d^2 + t_3 * d^1 + t_4 * d^0) \bmod 13$

$p_0 = (1 * 10^3 + 9 * 10^2 + 6 * 10^1 + 4 * 10^0) \bmod 13$

$p_0 = 1$

Rabin Fingerprint Algorithm (Cont.)

- Example
  - index = 1

1	9	6	4	8	6	3	5	…

  - Compute p_1

$p_1 = (d * (p_0 – d^3 * t_1) + t_5) \bmod 13$

$p_1 = (10 * (1 – 10^3 * 9) + 8) \bmod 13$

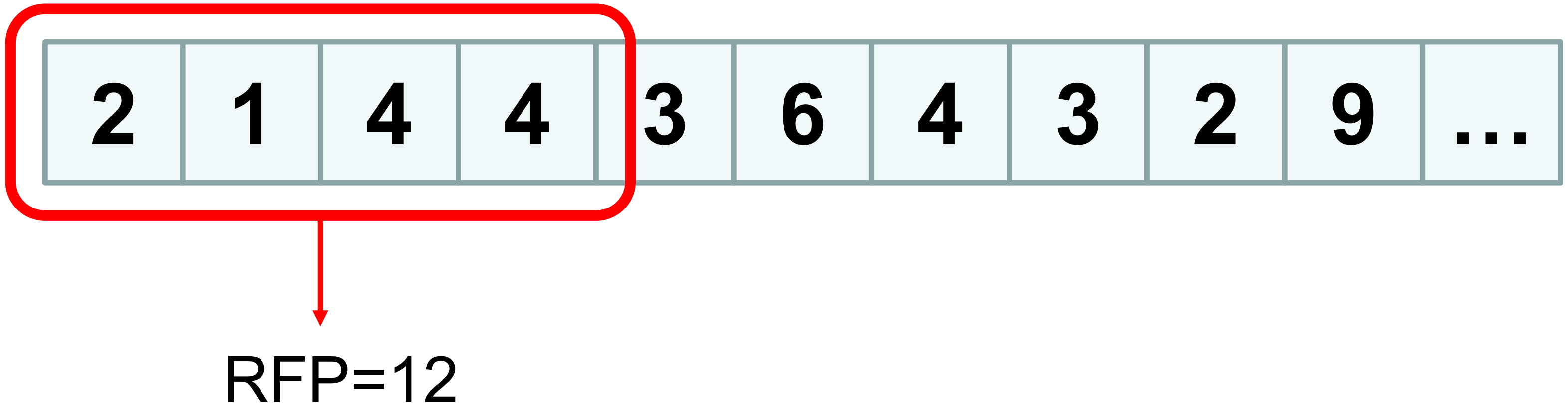
$p_1 = 2$

We can compute the following fingerprint in the same manner.

Chunking With RFP Algorithm (Cont.)

➢ How to do variable-size chunking via RFP algorithm?

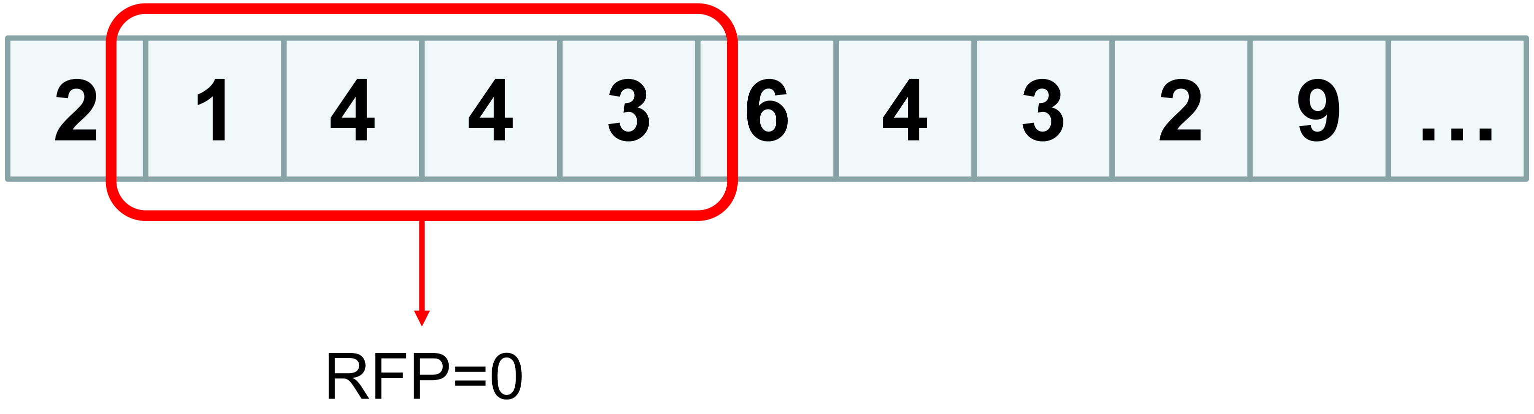
- Example: m=4, d=10, q=13, mask=$(1111)_2$=15



- RFP & mask != 0
- Think about: in which case will RFP & mask = 0 ?

Chunking With RFP Algorithm (Cont.)

- How to do variable-size chunking via RFP algorithm?
  - Example: m=4, d=10, q=13, mask=$(1111)_2$=15

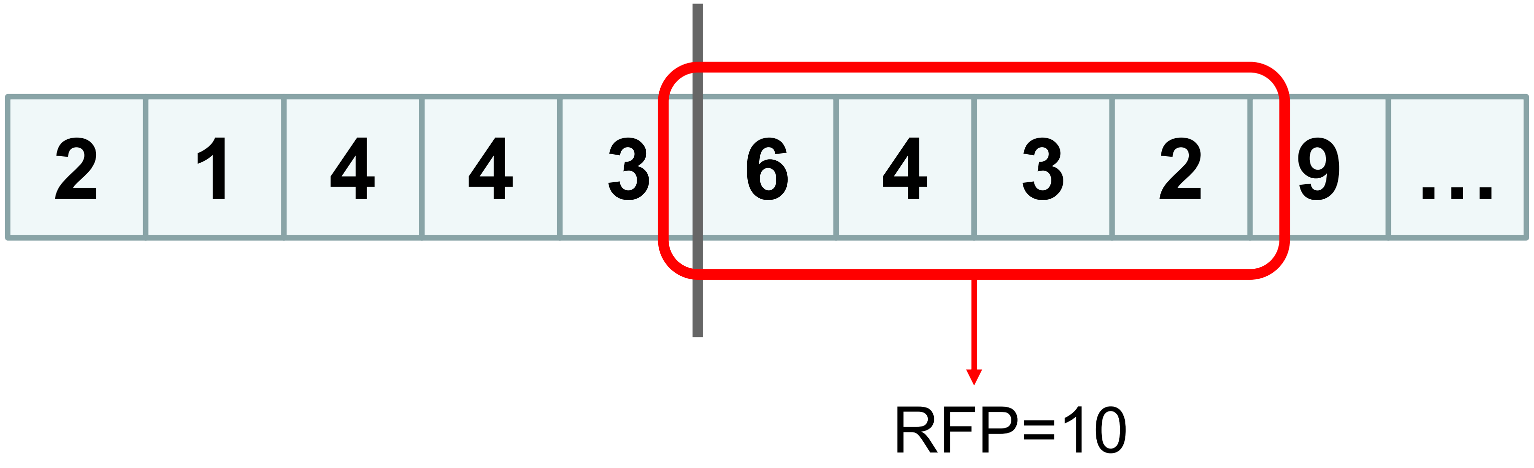


- RFP & mask = 0
- Set an anchor point at the end of current window

Chunking With RFP Algorithm (Cont.)

➢ How to do variable-size chunking via RFP algorithm?

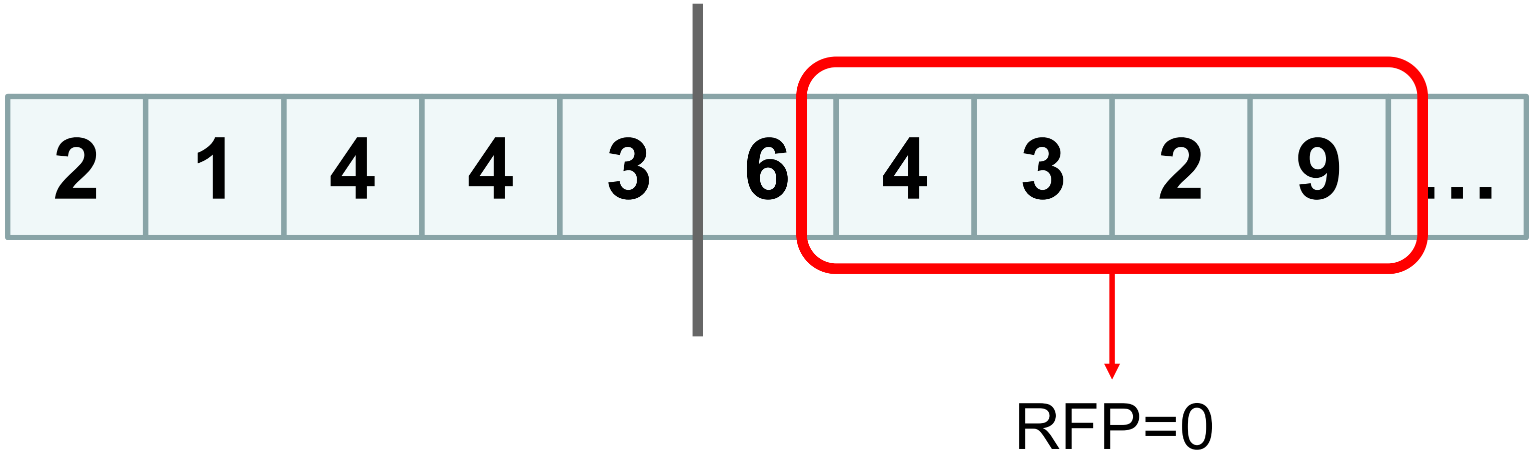
- Example: m=4, d=10, q=13, mask=$(1111)_2$=15



- RFP & mask != 0
- Continue

Chunking With RFP Algorithm (Cont.)

- How to do variable-size chunking via RFP algorithm?
  - Example: m=4, d=10, q=13, mask=$(1111)_2$=15

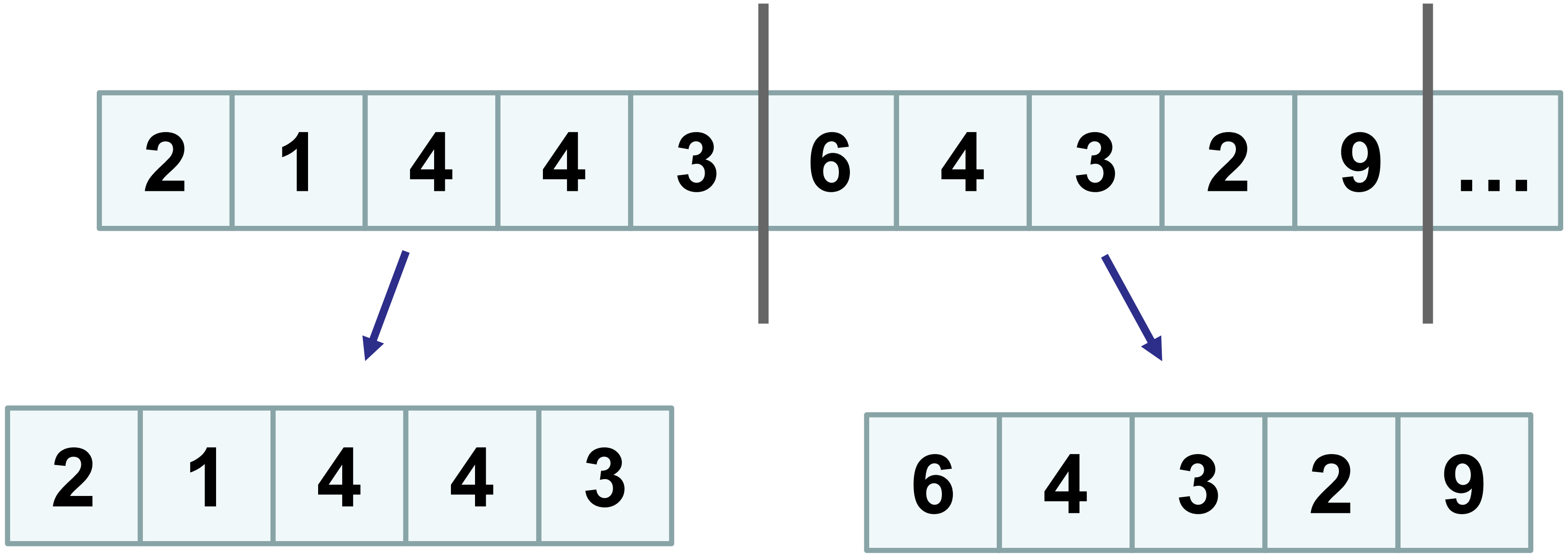


- RFP & mask = 0
- Set an anchor point

Chunking With RFP Algorithm (Cont.)

➢ How to do variable-size chunking via RFP algorithm?

- Example: m=4, d=10, q=13, mask=$(1111)_2$=15



- Up to now, we create 2 chunks with RFP algorithm

Chunking With RFP Algorithm

➢ How to do variable-size chunking via RFP algorithm?

- We need these parameters
  - For computing the rabin fingerprint
    - m: window size (min chunk size), e.g., 32
    - d: base, e.g., 257
    - p: modulus (avg chunk size), e.g., 2048
  - For chunking
    - mask: multiple 1-bits (111…1111)
      - we use the number same as (modulus – 1), e.g., 2047
    - max chunk size, e.g., 4096
      - to control the max chunk size

Summary of chunking

- Basically, we start from a buffer whose size is minimum size of a chunk, except for EOF.

- Computing the Rabin Fingerprint for each sliding window.

- Chunking
  - RFP & mask = 0
  - Maximum size of a chunk is reached

Summary of chunking

- The relationship between the efficiency of deduplication and chunk size [1]
  - Efficiency of deduplication: raw deduplication ratio = logical size / physical size
  - Smaller chunk size → finer-grained compression → higher raw deduplication ratio
  - In our assignment, when we refer to deduplication ratio, we mean raw deduplication ratio

Chunk size	Full backup	Incremental backup	Weekly-full backup
2KB	**218.5**	**13.6**	**42.8**
4KB	197.0	12.6	39.4
8KB	181.9	11.7	36.5
16KB	167.4	10.7	33.6
32KB	153.3	9.8	30.8
64KB	139.1	8.9	27.9
128KB	128.0	8.2	25.7
WFC	16.4	1.1	2.3

TABLE II. RAW DEDUPLICATION RATIOS FOR VARIOUS CHUNKING METHODS AND BACKUP STRATEGIES. WFC STANDS FOR WHOLE-FILE CHUNKING.

[1] MSST'16 A Long-Term User-Centric Analysis of Deduplication Patterns

Hints

- Attention
  - RFP algorithm might overflow before modular operation
    - (a + b) mod q ≡ (a mod q + b mod q) mod q
    - (a - b) mod q ≡ (a mod q – b mod q) mod q
    - (a * b) mod q ≡ (a modq * b mod q) mod q
- Performance
  - Fast modular exponentiation algorithm
  - Leverage multi-threading to pipelining the workflow

Content

- Deduplication
- Variable-size chunking
- Checksum
- Indexing

Checksum

- How to compute checksum of a data chunk?
  - We can use **SHA-256** algorithm to create checksum, which is available in **Java MessageDigest Library**
    - Note that: for our assignment, we choose SHA256 algorithm
  - Example:
    - There is a data buffer **data**, whose length is **len**

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
md.update(data, 0, len);
byte[] checksumBytes = md.digest();
```

- Checksum is used to identify a data chunk
  - An important component in deduplication

Content

- ➢ Deduplication
- ➢ Variable-size chunking
- ➢ Checksum
- ➢ Indexing

Indexing

- Two kinds of indexes
  - 1. Fingerprint indexing: to manage chunks
  - 2. File recipe: to manage files
- Fingerprint indexing
  - It is a data structure
  - Given a fingerprint value
    - Return whether corresponding chunk exists, and chunk information (e.g., chunk address)
- File recipe
  - Given a file recipe
    - Return all chunks' information of this file (chunk list)

Indexing

- As chunk sizes increase, a decrease in actual deduplication ratio is not obvious or guaranteed [1]
  - Smaller chunk size → larger metadata size
  - Actual deduplication ratio = logical data / (physical data + metadata)
  - In our assignment, when we refer to deduplication ratio, we mean raw deduplication ratio

- Metadata
  - 1. Fingerprint indexing
  - 2. File recipe

Chunk size	Full backup	Incremental backup	Weekly-full backup
2KB	50.9	11.1	25.8
4KB	79.3	**11.4**	30.2
8KB	107.9	11.1	**32.0**
16KB	127.2	10.5	31.6
32KB	**133.9**	9.7	29.9
64KB	130.5	8.9	27.6
128KB	124.3	8.2	25.7

TABLE III. EFFECTIVE DEDUPLICATION RATIOS AFTER ACCOUNTING FOR META-DATA OVERHEADS.

[1] MSST'16 A Long-Term User-Centric Analysis of Deduplication Patterns

Thank you
Q & A