Lecture 7: Deduplication (Part 1)

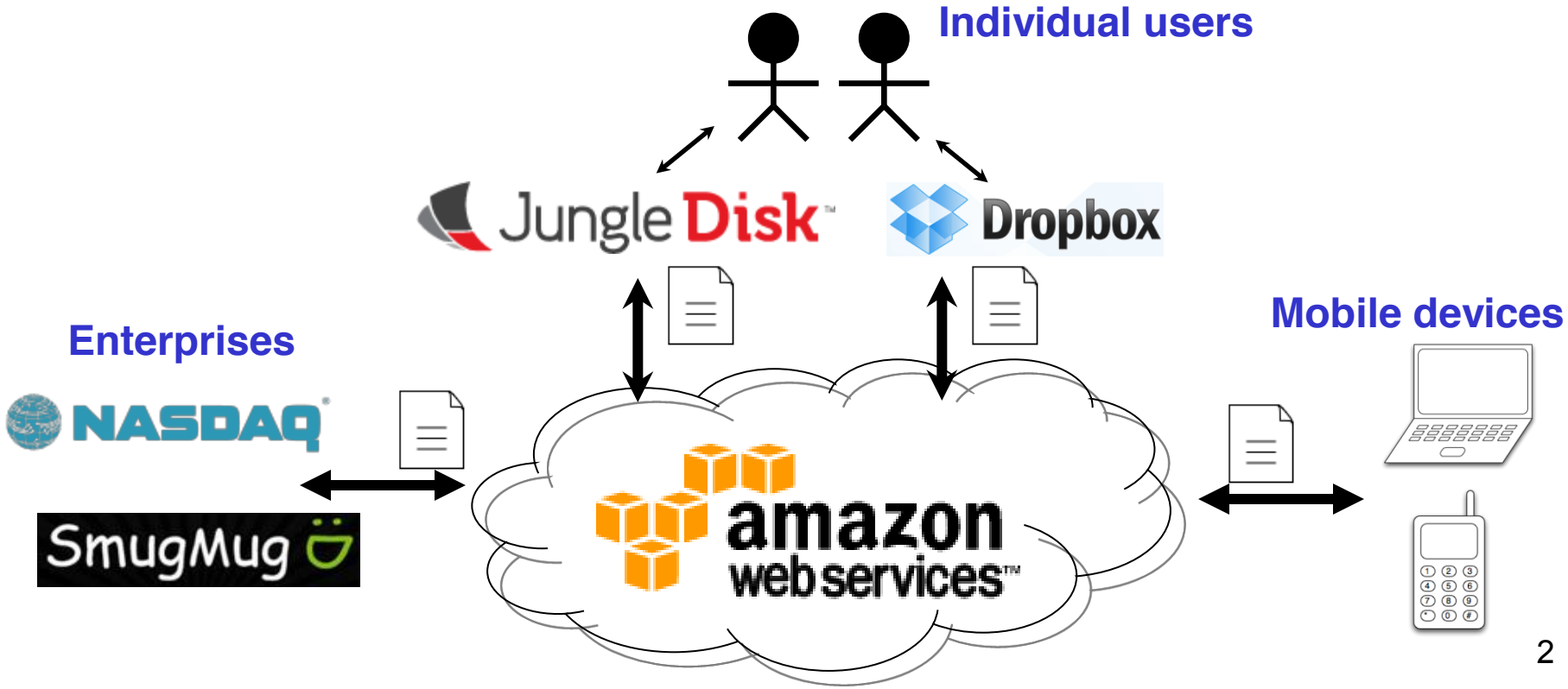
CSCI4180 (Fall 2013)

Patrick P. C. Lee

Most slides are adapted from Dr. T. Y. Wong's lecture in 2012 spring

Cloud Storage

- Cloud storage is now an emerging business model for data outsourcing
  - **Pay as you go**



Cloud Storage Cost Model

Amazon S3 Rackspace Windows Azure

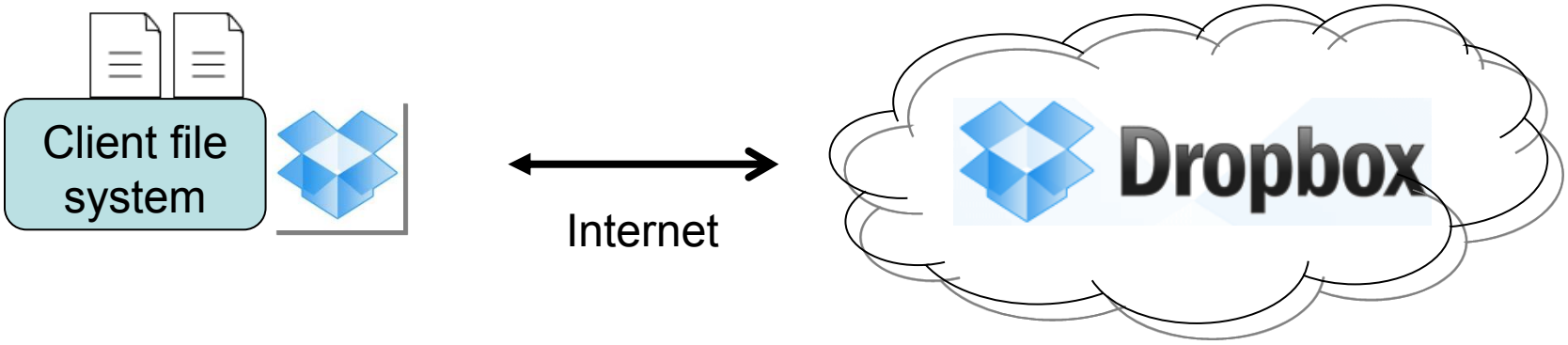
	S3	RS	Azure
Storage (per GB)	$0.14	$0.15	$0.15
Data transfer in (per GB)	free	free	free
Data transfer out (per GB)	$0.12	$0.18	$0.15
PUT,POST (per 10K requests)	$0.10	free	$0.01
GET (per 10K requests)	$0.01	free	$0.01

Monthly price plan as of Sep 2011

- Components of cloud storage cost:
  - Storage space
  - Data transfer (outbound from the cloud)
  - Number of requests

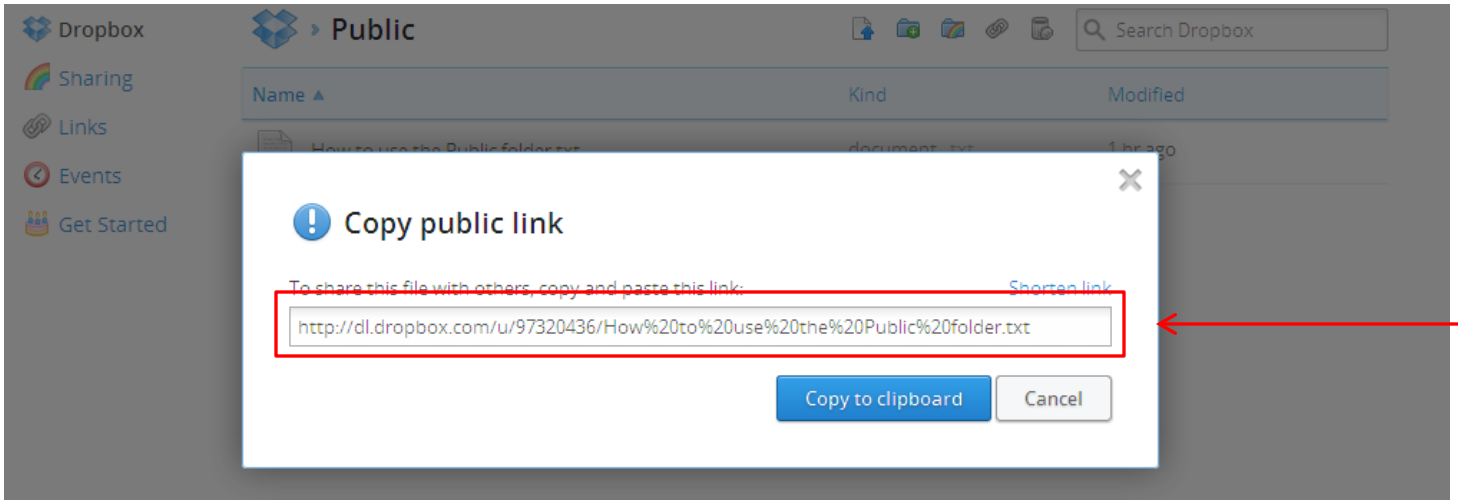
Motivation

- Many of us use **Dropbox** to store and share data?
- How to build a Dropbox service?
  - What is the Dropbox network?
  - How does Dropbox generate revenue?



Motivation

- Where is the Dropbox network?

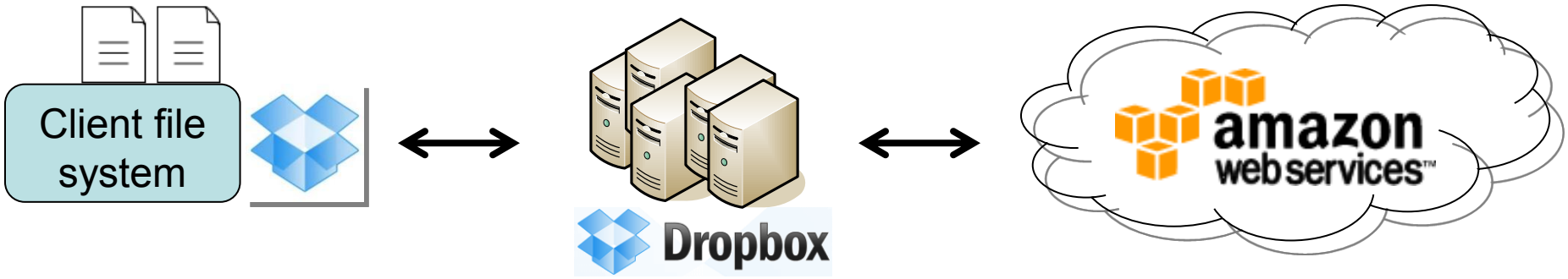



```
$ nslookup dl.dropbox.com

Non-authoritative answer:
Name:    dl-balancer3-985632286.us-east-1.elb.amazonaws.com
Addresses:  23.21.176.62, 23.21.251.228, 50.16.240.166, 107.20.133.134
          107.20.162.145, 107.22.210.127, 174.129.0.56, 174.129.197.250
Aliases:  dl.dropbox.com
```

Motivation

- AWS empowers Dropbox
  - EC2: elastic compute cloud
    - Web hosting, computing
  - S3: Simple storage service
    - Object storage



Dropbox gateway servers

Motivation

- Amazon charges:

http://aws.amazon.com/s3/#pricing

Storage Pricing

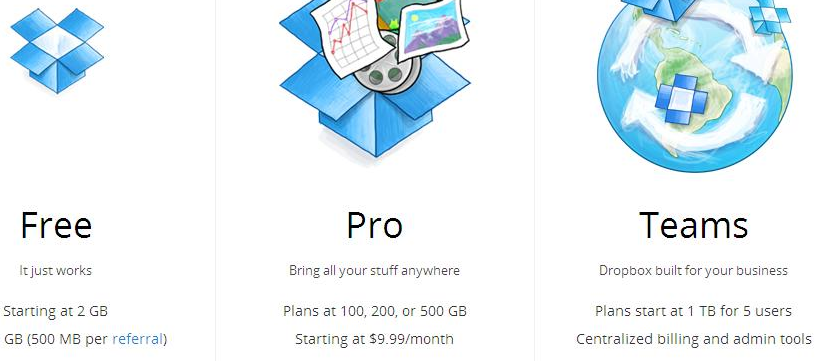
Region: US Standard

	Standard Storage	Reduced Redundancy Storage
First 1 TB / month	$0.125 per GB	$0.093 per GB
Next 49 TB / month	$0.110 per GB	$0.083 per GB
Next 450 TB / month	$0.095 per GB	$0.073 per GB

Pricing as of Aug 2012

- Dropbox also charges:

https://www.dropbox.com/pricing



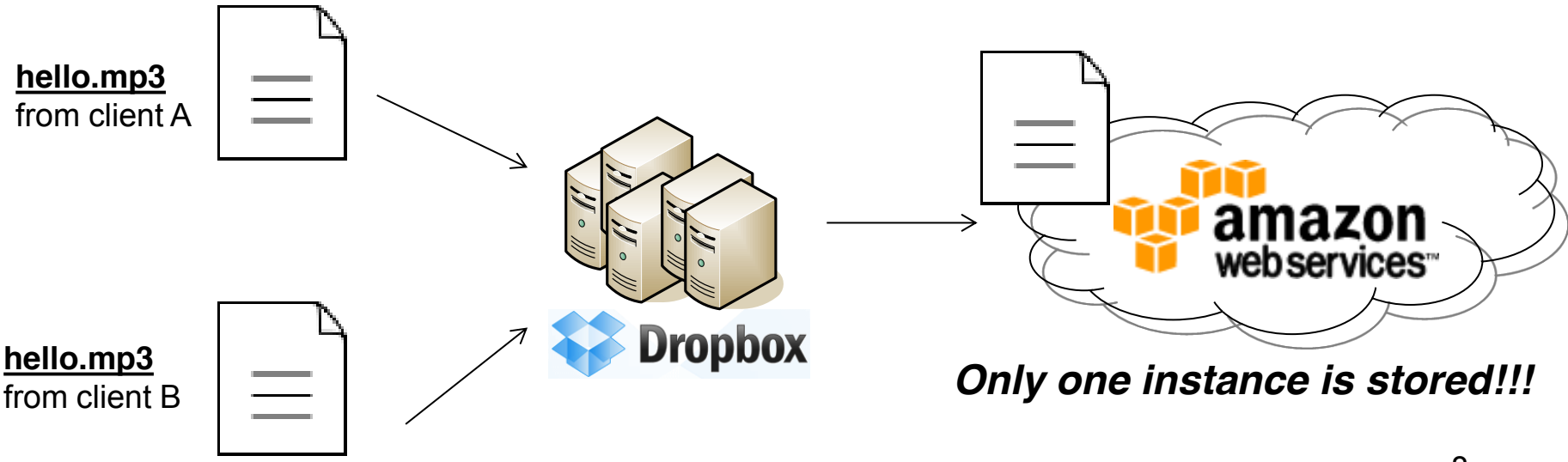
100GB storage/month:
Amazon: $12.5/month
Dropbox: $9.99/month

- Dropbox is cheaper. Then how can Dropbox make profit?

Motivation

- Most Dropbox users use free accounts
- A paid Dropbox account is also cheaper

- Dropbox implements **deduplication** over S3



Motivation

- Deduplication:
  - An approach that eliminates redundant data on storage. Instead of storing multiple copies of data of the same content, only one copy is stored.
- Many users upload similar data to the cloud. Dropbox exploits this feature to make profit.

Deduplication vs. Compressions

- Compression reduces the amount of stored bits compared to the original data
  - reorganize and reformat the data to reduce dedendency
  - Transforms data into new representation with higher entropy
  - Typically works on a single file (or a single batch of files)
- Deduplication
  - no need to zip and unzip
  - Detects identical data blocks / similarities between data blocks
  - Works across files (e.g., archives, backups, or collections of virtual machine images)

- In practice, compression has to be performed after deduplication
  - What about deduplication after compression?

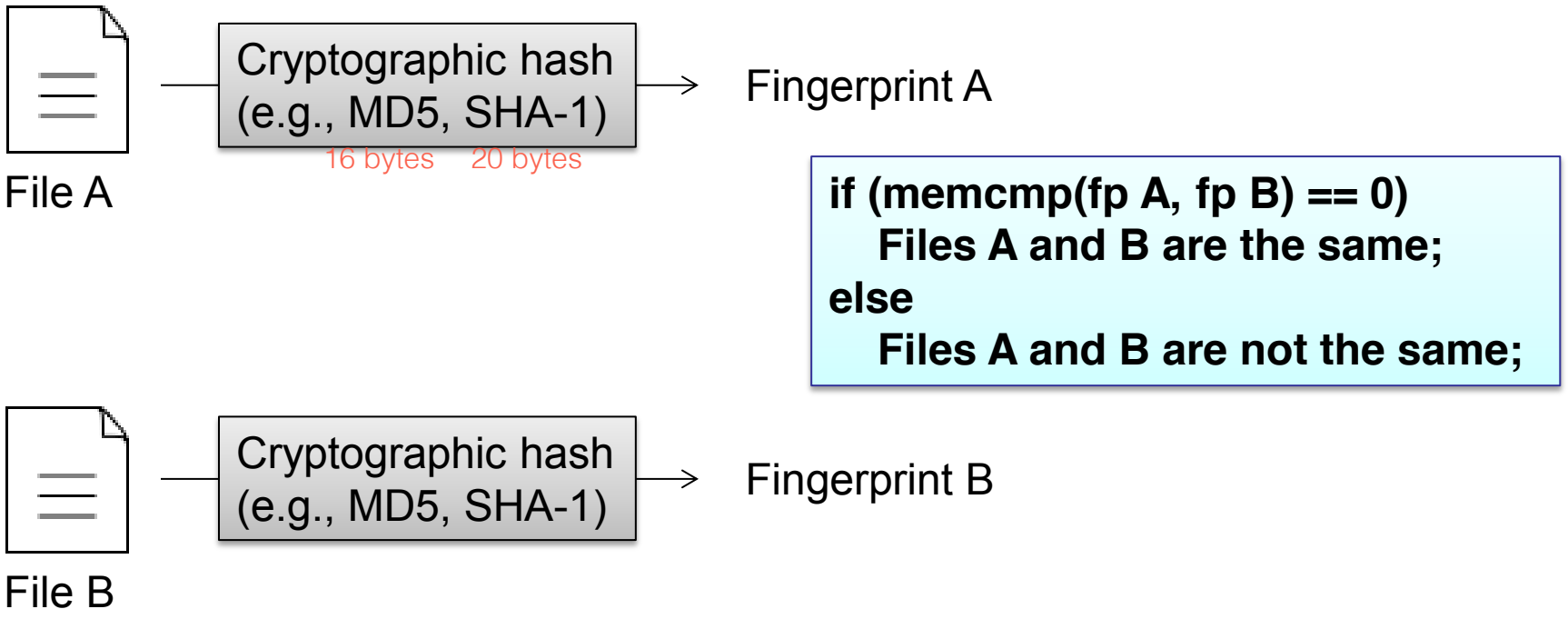
Deduplication Overview

- Deduplication on a stream of data:
  - Fingerprinting (compare by hash):
    - Generate identifiers of data based on content
  - Chunking:
    - Divide data stream into different chunks
    - Generate fingerprints for chunks
  - Indexing:
    - Maintain all fingerprints of existing chunks
    - *To be discussed in next lecture*.

Fingerprinting

reduce the size of byte-by-byte comparison to make sure two files are identical

- How to check if two copies of data are identical?
  - **Fingerprinting (compare-by-hash)**:
- Fingerprinting on a per-file basis:



Fingerprinting

- Why compare-by-hash?
  - Instead of reading data byte-by-byte, we can determine if two data copies are identical by comparing only few bytes

- V. Hensen disagrees: Problem: Hash collision
  - "*Use of compare-by-hash is justified by mathematical calculations based on assumptions that range from unproven to demonstrably wrong*"
    - From mathematical point of view, the probability that *"two different files give the same checksum*" is non-zero!
  - Ref: "An analysis of compare-by-hash", in Proceedings of the 9th conference on Hot Topics on Operating Systems, 2003.
- What happens if two hashes collide?

Fingerprinting

- J. Black advocates:
  - *We conclude that it is certainly fine to use a 160-bit hash function like SHA-1…The chance of an accidental collision is about 2^{-160}….*"
  - Ref: "Compare-by-hash: A Reasoned Analysis", In Proceedings of USENIX Annual Technical Conference, 2006

- *My view: It is more likely to see a hardware crash before seeing a hash collision. It's okay to use fingerprinting.*

Chunking

- What is the problem of *fingerprinting on a file*?
  - What happens if I update the first byte of a file?

- Dropbox implementation:
  - Doesn't use the concept of files
  - Instead, every file is split up into chunks of up to 4MB
  - Apply SHA-256 to each chunk
  - Hash values are sent to Dropbox servers, and are compared to the hashes already stored
  - If a chunk doesn't exist, upload it
    - That is, only updated chunks are uploaded

Chunking

- Why does Dropbox implement this chunking?
  - Storage saving
  - Network bandwidth saving
- Dropbox keeps track of fingerprints of all chunks in its database

- Identical chunks of multiple files (of multiple different users) can also be deduplicated
  - There may be security issues (discussed later)

Chunking

- Dropbox uses **fixed-size chunking** always 4MB
  - Each file is divided into equal-size chunks
- Any problem with the following?
  - You are writing a C program…
  - After you finished it, you save it to the Dropbox folder. [1st upload]
  - Yet, you cannot compile because you mistype a variable name "dummy" as "tummy". You update the variable and upload the file again [2nd upload]
  - Yet, you cannot compile it because you miss the statement "#include <stdio.h>"! You fix the problem and upload the file again. [3rd upload]
- Question: If there are n chunks in the 1st upload, then how many chunks are in the 2nd and 3rd upload?

Chunking

content-defined chunking

Variable-size chunking: enables adaptive boundaries on dividing data into chunks

Strategies	Parameters	Costs	Deduplication Effectiveness
Whole file	NIL	Fastest	Lowest
Fixed-size	Chunk size	Disk seeks Fingerprint calculations	Middle
Variable-size may need extra space to store the different content	Average chunk size	Disk seeks CPU time to determine chunk boundaries Fingerprint calculations	Highest

Comparison on chunking strategies

Chunking

- Fixed-size chunking:
  - Negligible work on determining chunk boundaries
- Variable-size chunking:
  - **Rabin-Karp Algorithm** (or **Rabin fingerprinting (RFP)**) is the standard
    - A Rabin fingerprint is the polynomial representation of data
    - Used for **string pattern recognition**
  - Applications of Rabin fingerprinting:
    - Storage deduplication
    - Network traffic redundancy elimination
    - Worm detection

Rabin Fingerprinting

- String pattern recognition

How to identify the pattern “OR” in the following string?

1. Define a window of size **m** bytes, e.g., **m = 2** in the above case.

2. Let the length of the string to be **n**. Then:

```
for (i = 0; i <= n - m; i++)
   if (strncmp(string + i, pattern, m) == 0 )
      printf("Pattern found at %d-th byte\n", i);
```

Rabin Fingerprinting

- String pattern recognition

Complexity: O(m(n-m+1)) if n >> m, it is linear
if m>> n, it depends on m

Can we do better than this?

n-m+1 steps

m steps

1. Define a window of size **m** bytes, e.g., **m = 2** in the above case.
2. Let the length of the string to be **n**. Then:

```
for (i = 0; i <= n - m; i++)
    if (strncmp(string + i, pattern, m) == 0 )
        printf("Pattern found at %d-th byte\n", i);
```

Rabin Fingerprinting

- RFP's idea is to reduce strncmp() operations!
- Transform a pattern into an integer value called the **fingerprint**

Example (m = 5):

Input data:

```
Dfefjfdls jf;ldafdjkf lksdjf
;sdjaf s fds j;ladkfj dlk fjdak
jf;lasdkj k

Dskafjd;safj kdsj ;fadkj eiawoi
q qwihrie oidafdaj lkjejef;a
```

```
fjfdl  ⟶ fingerprint f1
 j;la  ⟶ fingerprint f2
fjd;s  ⟶ fingerprint f3
afdaj  ⟶ fingerprint f4
```

RFP is a **many-to-one mapping**. Different patterns may be mapped to the same fingerprint (we resolve this later). But same pattern must have the same fingerprint.

RFP: Polynomial Representation

- Rabin fingerprinting is related to **polynomial** and modular arithmetic…

Let the "alphabets" in the universe contain {0-9} only.

Let the input be an array: $t = [t_1, t_2, \ldots, t_m, t_{m+1}, \ldots, t_n, \ldots]$, where $0 \leq t_m \leq 9$

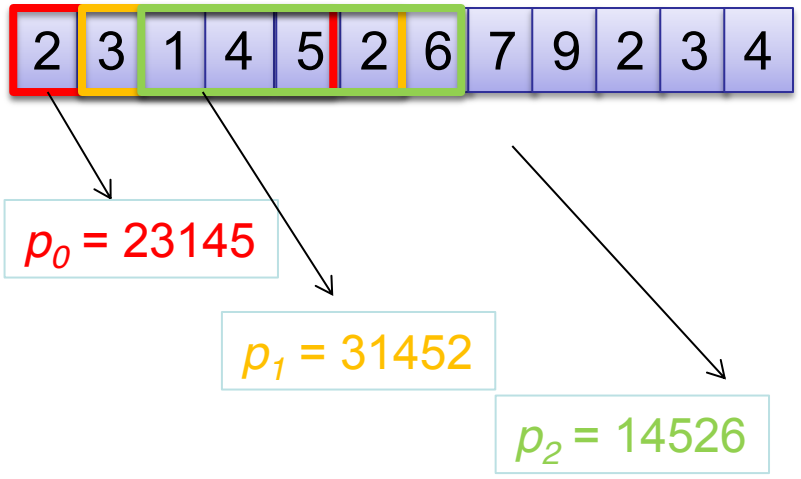
$$p_s = \sum_{i=1}^{m} t_{s+i} \times 10^{m-i}$$

where
(1) $s \geq 0$
(2) p_s is the fingerprint value that represents a window of data of length m.

RFP: Polynomial Representation

- Example:

Let the “alphabets” in the universe contain {0-9} only.



Sliding window representation!

RFP: $p_s = \sum_{i=1}^{m} t_{s+i} \times 10^{m-i}$

RFP: Polynomial Representation

- Can we do smarter?

Let the “alphabets” in the universe contain {0-9} only.

Let the input be an array: $t = [t_1, t_2, \ldots, t_m, t_{m+1}, \ldots, t_n, \ldots]$, where $0 \leq t_m \leq 9$

p_s can be rewritten as:

$$p_s = \begin{cases} \sum_{i=1}^{m} t_i \times 10^{m-i}, & s = 0 \\ 10 \times (p_{s-1} - 10^{m-1} \times t_s) + t_{s+m+1}, & s > 0 \end{cases}$$

RFP: Polynomial Representation

- Can we do smarter?

Let the input be an array: $t = [t_1, t_2, \ldots, t_m, t_{m+1}, \ldots, t_n, \ldots]$, where $0 \leq t_m \leq 9$

p_s can be rewritten as:

$$p_s = \begin{cases} \sum_{i=1}^{m} t_i \times 10^{m-i}, & s = 0 \\ 10 \times (p_{s-1} - 10^{m-1} \times t_s) + t_{s+m+1}, & s > 0 \end{cases}$$

Computed from previous p_s

<u>Question</u>: What is the new complexity?

RFP: Polynomial Representation

- A subtle improvement:
  - If the range of the possible values of an alphabet is known, then all the possible value of **this product term can be pre-computed and stored beforehand**!

no need to to multiplication

$$p_s = \begin{cases} \sum_{i=1}^{m} t_i \times 10^{m-i}, & s = 0 \\ 10 \times (p_{s-1} - 10^{m-1} \times t_s) + t_{s+m+1}, & s > 0 \end{cases}$$

RFP: Polynomial Representation

- Complexity:
  - O(m): to compute p_0
  - O(n-m+1): to compute p_s
- What if n and m are large?
  - n is large:
    - No problem! It is the price to pay for playing with a large file!
  - m is large:
    - Bad! We would produce a big fingerprint that is impractical to store.

RFP: Modular Arithmetic

- Rabin fingerprinting is related to polynomial and **modular arithmetic**…

Let the input be an array: $t = [t_1, t_2, \ldots, t_m, t_{m+1}, \ldots, t_n, \ldots]$, where $0 \leq t_m \leq 9$

The Rabin fingerprint can be described as a polynomial with base 10 ***modulo q***.

to limit the size of fingerprint

$$p_s = \begin{cases} \left(\sum_{i=1}^{m} t_i \times 10^{m-i}\right) \ mod\ q, & s = 0 \\ \left(10 \times (p_{s-1} - 10^{m-1} \times t_s) + t_{s+m+1}\right) \ mod\ q, & s > 0 \end{cases}$$

RFP: Modular Arithmetic

Properties of modular arithmetic:

further reduce the size

1. [(a mod n) + (b mod n)] mod n = (a + b) mod n
2. [(a mod n) - (b mod n)] mod n = (a - b) mod n
3. [(a mod n) x (b mod n)] mod n = (a x b) mod n

So, every operation in the equation can be calculated easily.

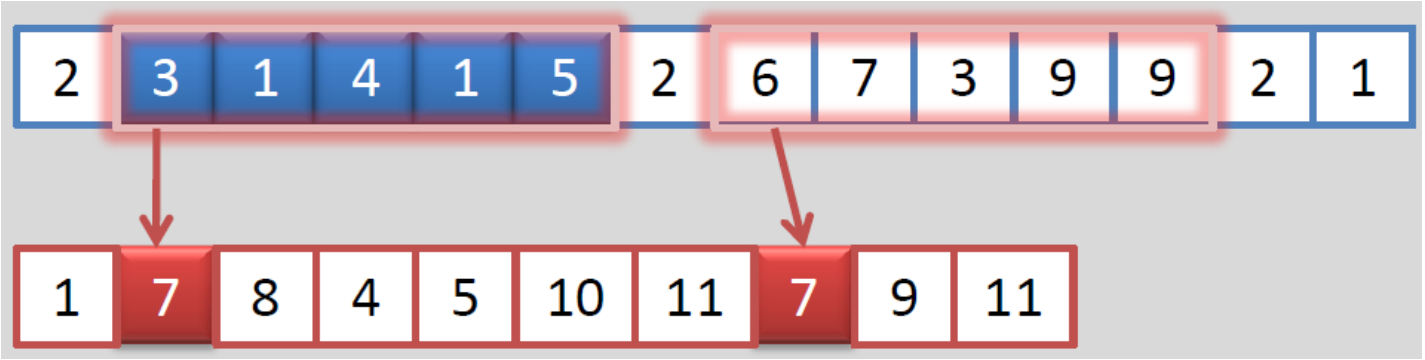
and, the size can be kept the same

$$p_s = \begin{cases} \left(\sum_{i=1}^{m} t_i \times 10^{m-i}\right) \ mod\ q, & s = 0 \\ \left(10 \times (p_{s-1} - 10^{m-1} \times t_s) + t_{s+m+1}\right) \ mod\ q, & s > 0 \end{cases}$$

q determine the size of fingerprint and collision rate

RFP: Modular Arithmetic

- With modular arithmetic, multiple string patterns may map to the same fingerprint
- Example: let q = 13



- 31415 and 67399 have the same RFP
- We need to call strncmp() to check if the string patterns are actually identical (can't simply reply on RFP)

RFP Summary

- RFP is a function of ***d*** and ***q***

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

- Parameter *d*
  - Practically speaking, '***d***' should not be 10.
  - '***d***' defines the range of the possible fingerprint values **before** taking the modulo operation. Typically, it should be a value that ***is larger than the largest value of the set of alphabets*** in order to **avoid unnecessary collisions**.
  - **Discussion**: Which '***d***' will you take?

257.
the number should be larger than 255.
256 is a number that easy to produce collision

RFP Summary

- RFP is a function of ***d*** and ***q***

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

- Parameter *q*
  - Again, it is better to choose a large value for ***q*** so as to minimize the number of collisions. But, don't forget our aim: **to have a reasonably large (and small) *q*** so that it is computationally convenient to operate on the fingerprints.
  - **Discussion**: Is the value 2^{31} is a good choice?

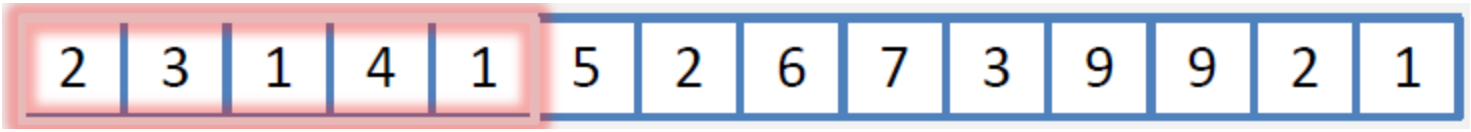
RFP on Deduplication

- How is RFP applied into **deduplication**?
- Flow:
  - (1) Select interested RFP values
  - (2) Select “m” as the parameter of the solution
  - (3) Use the interested RFP values to divide chunks
- Idea:
  - RFP values provide a guess of similar chunks
    - If two chunks have different RFP values, they must be different
  - Use cryptographic hash to decide if two similar chunks are actually identical

Reference: Udi Manber. *Finding Similar Files in a Large File System*, 1993.

RFP on Deduplication

- Step (1): Select interested RFP values
  - Why? Because storing all RFP values is expensive

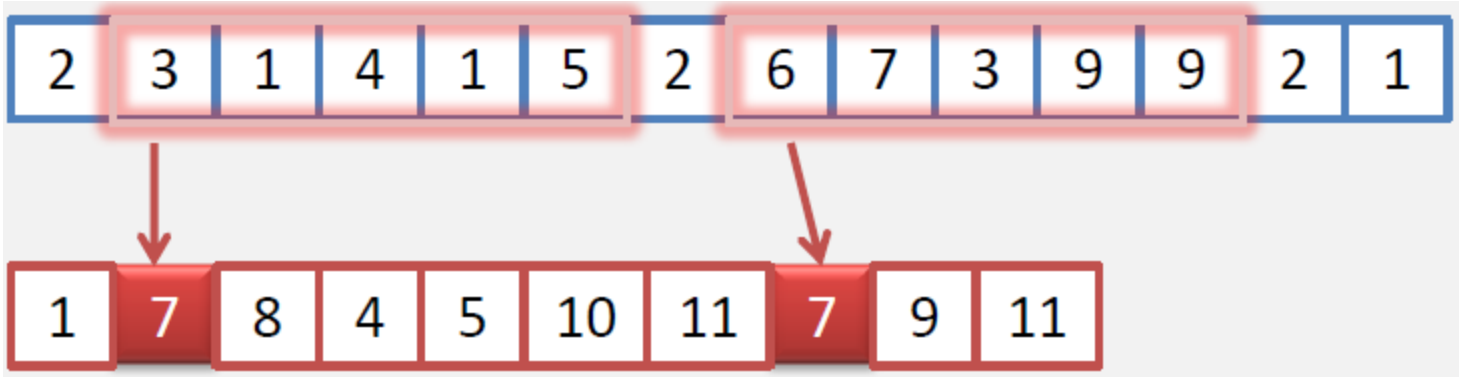


Number of RFP values for this stream: 10

- In general, number of RFP values is **n-m**.
- Saving and comparing all RFP values is expensive

RFP on Deduplication

- Step (1): Select interested RFP values
  - Solution: only interested in fingerprints that share the common properties
  - Example: we're only interested in RFP = 7



RFP on Deduplication

- Step (1): Select interested RFP values
  - Solution: only interested in fingerprints that share the common properties

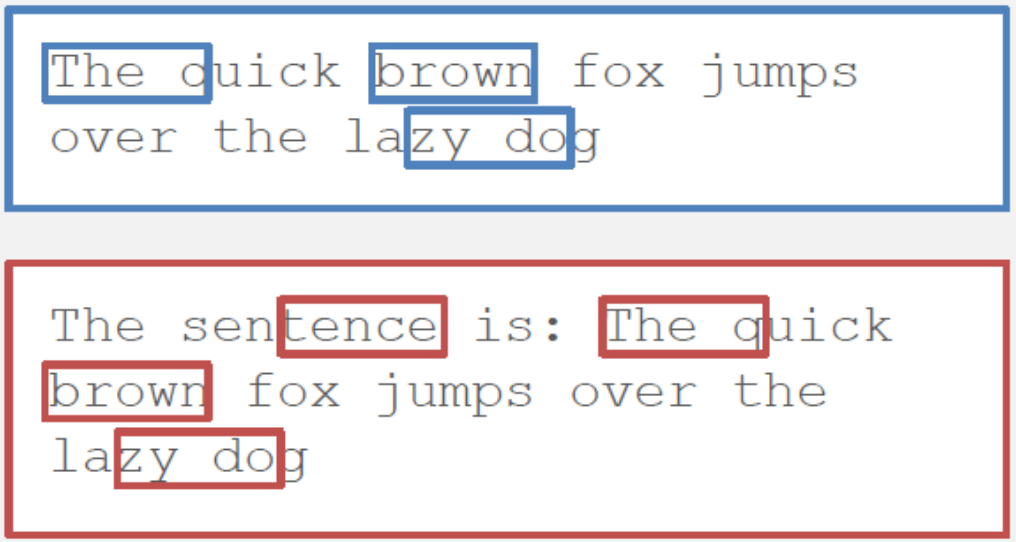
- To make the RFP computation fast, we can choose the property that we store fingerprints with ***least significant k bits equal to 0***.
  - On average, we'll store one RFP for each 2^k characters.

RFP on Deduplication

- Step (2): choose the value of **m**
  - Different **m** gives different sets of RFPs, and different locations of interested RFPs
  - **m** affects the number of RFP produced.
  - Also, it affects the minimum data length required

RFP on Deduplication

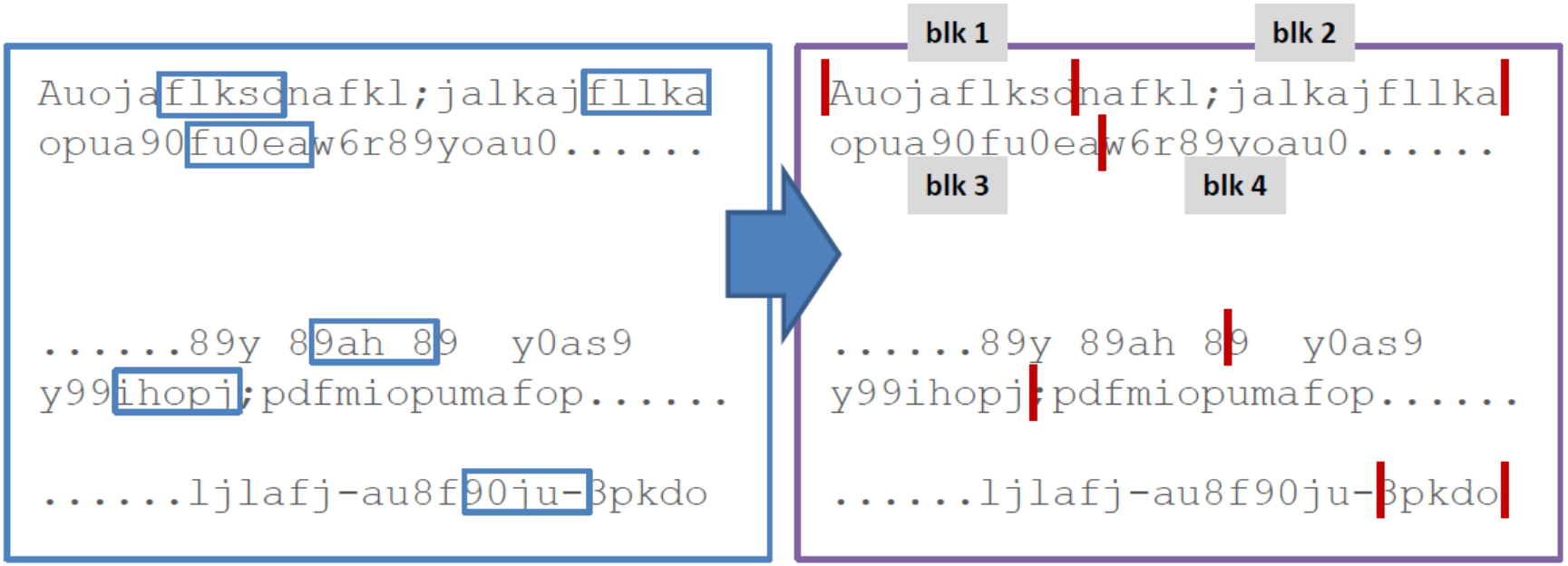
- Step (3): for each file, we record the chunks that match the interested RFPs



Note: a shift on the data will not affect the discovery of the target patterns.

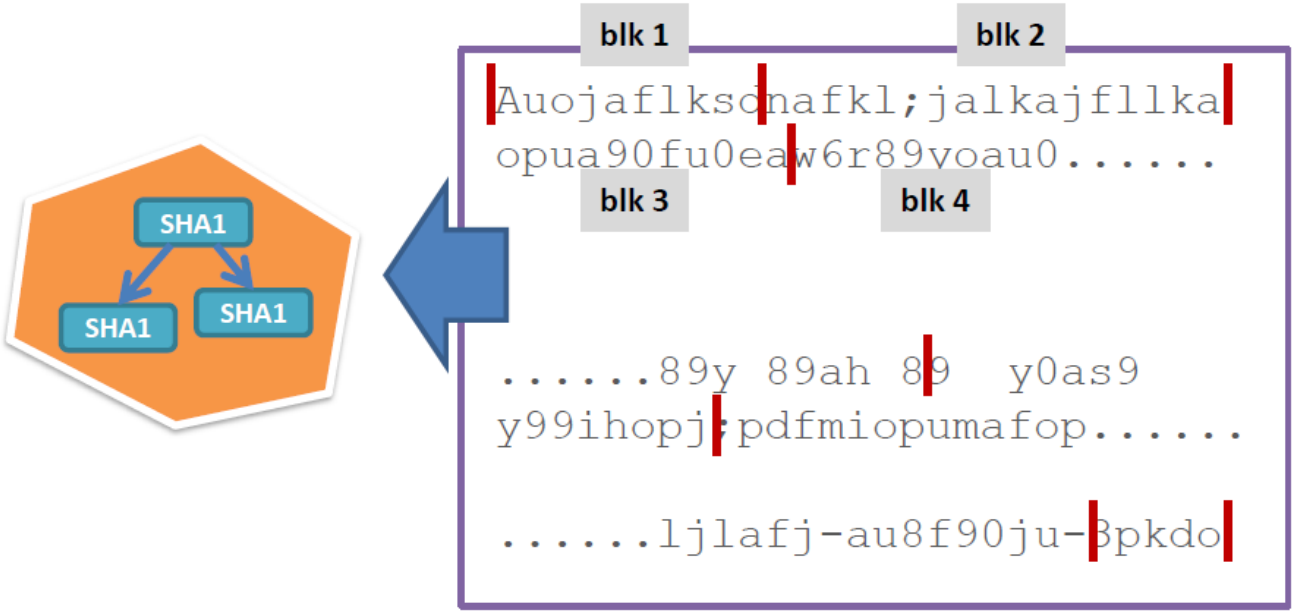
RFP on Deduplication

- RFP is used for determining block boundaries.
  - The chosen set of fingerprints creates **anchor points**.
  - It results in variable-sized blocks.
  - Most importantly, it is **content-dependent**.



RFP on Deduplication

- For each variable-sized chunk,
  - Compute a cryptographic hash value for each
  - With an indexing structure, we can find if duplicated contents exist in two different files



Summary

Window size, m	Determine the minimum length of each chunk.
Multiplier, d	Usually large than the number of possible inputs, e.g., 257 is a good choice for binary files. Why? Think about it: ***If 'd' is an even number***, then… - You will always produce even checksum values. - You are using only half of the usable range. - This implies a higher chance in obtaining the same fingerprint.
Modulo, q	Usually set it as a power of 2 number. Why? Because instead of using '%', we can use bitwise AND operator to achieve the modulo operation.
Anchor point selection	It affects how frequent an anchor point appears. Usually, choosing the few least significant bits as the selection criterion. e.g., `if ((RFP & 0xFF) == 0)` then produce one anchor point.

Homework Questions

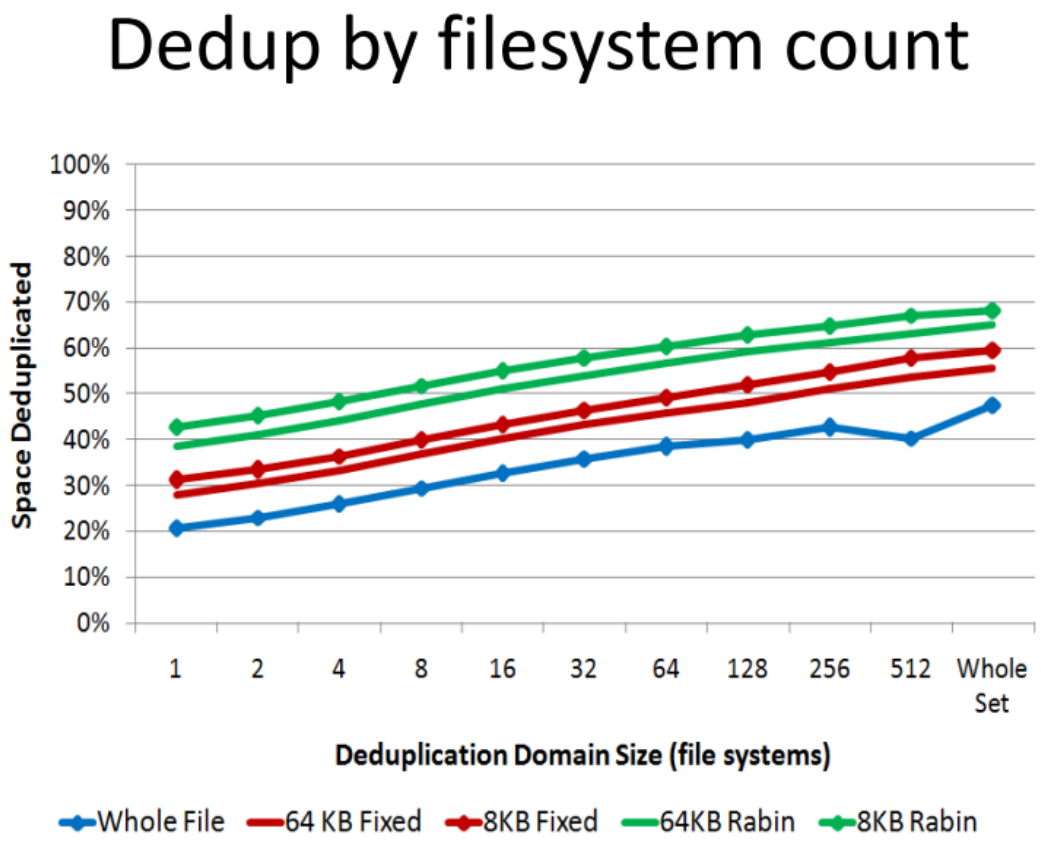
- What if the input contains many runs of zeros?
- What if the inputs are expected to be small, but have a very high similarity?

Case Study

- Real data analysis by **Microsoft Research**.
  - 857 desktop computers at Microsoft.
    - 40TB of data
    - 200M files
  - Experiment period: 4 weeks.
  - Deduplication workload: backups of the 875 filesystems.

Meyer et al., “A Study of Practical Deduplication”, FAST 2011 (best paper award)

Case Study



- Claim: the benefit of fine grained dedup is < 20%
  - Potentially just a fraction of that.

Other Applications of RFP

- Worm detection
  - Idea: use RFP to look for traffic patterns that appear like a worm attack
- Network optimization
  - Idea: use RFP to index traffic and eliminate redundancy
  - Instead of sending redundant data, send only smaller-size metadata

References

- Book Chapter
  - Chapter 34 in 1st edition; Chapter 32 in 2nd edition. Introduction to Algorithms. The MIT Press.

- Papers on storage deduplication
  - Muthitacharoen et.al. "***A Low-bandwidth Network File System***", in the Proceedings of 18th Symposium on Operating Systems Principles, 2001.
  - Benjamin Zhu et.al., "*Avoiding the Disk Bottleneck in the Data Domain Deduplication File System*", in Proceedings of FAST 2008