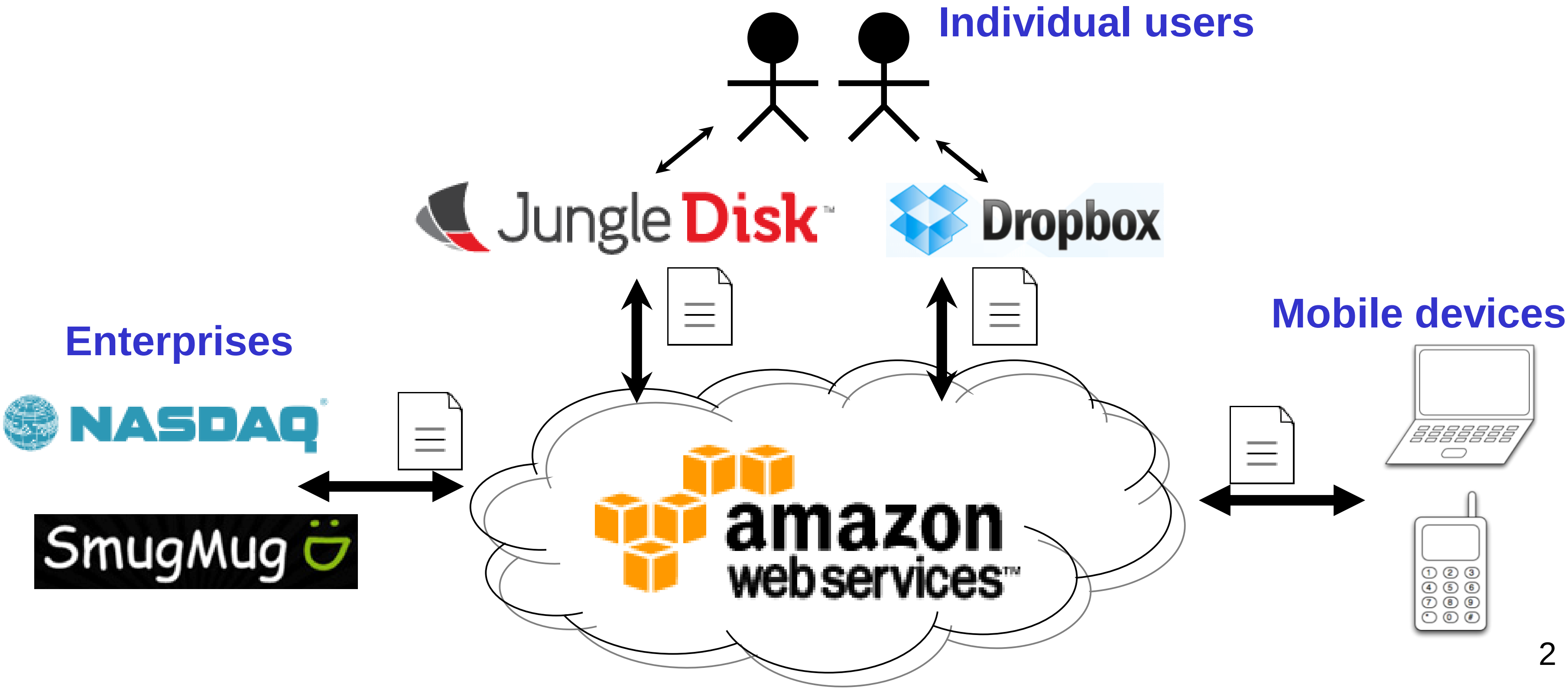
Lecture 8: Deduplication (Part 1)

CSCI4180

Patrick P. C. Lee

Cloud Storage

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



Cloud Storage Cost Model

- Commercial cloud storage providers charge by three components:
  - Storage space
  - Number of requests (e.g., PUT, GET, LIST, DELETE)
  - Data transfer (outbound from the cloud)

- Follows a **tiered** pricing model
  - e.g., 50TB storage is divided into two tiers:
    - Amazon S3 cost per month = 1024GB * $0.0300 per GB + 49 * 1024GB * $0.0295 per GB

Cloud Storage Cost Model

Storage (Standard)		Request		Transfer	
First 1 TB / month	$0.0300 per GB	PUT, COPY, POST, or LIST Requests	$0.005 per 1,000 requests	Transfer In	Free
Next 49 TB / month	$0.0295 per GB	Glacier Archive and Restore Requests	$0.05 per 1,000 requests	Transfer Out to Internet	
Next 450 TB / month	$0.0290 per GB	Delete Requests	Free	First 1 GB / month	$0.000 per GB
Next 500 TB / month	$0.0285 per GB	GET and all other Requests	$0.004 per 10,000 requests	Up to 10 TB / month	$0.120 per GB
Next 4000 TB / month	$0.0280 per GB	Glacier Data Restores	Free	Next 40 TB / month	$0.090 per GB
Over 5000 TB / month	$0.0275 per GB			Next 100 TB / month	$0.070 per GB

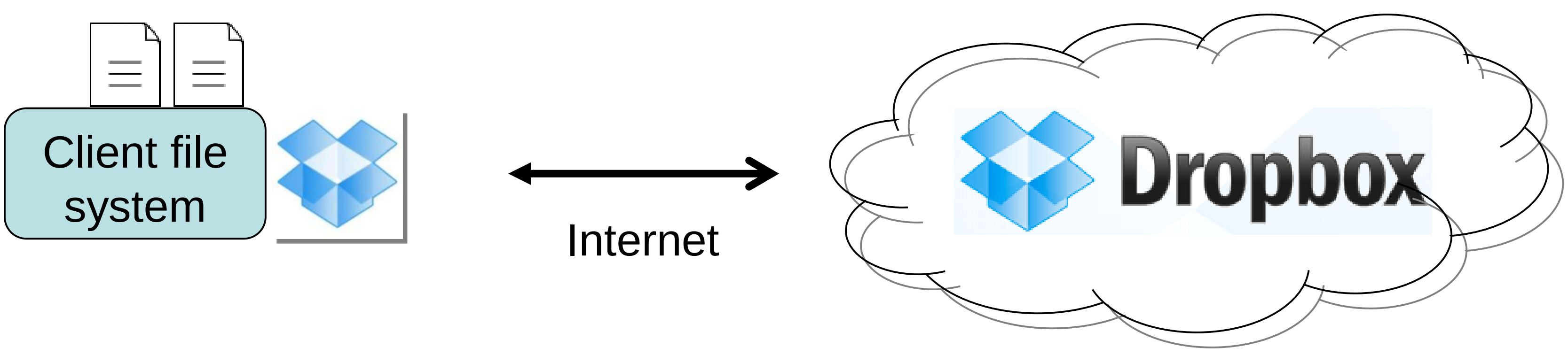
Amazon S3 Pricing in October 2014

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

Billing example: http://aws.amazon.com/s3/faqs/

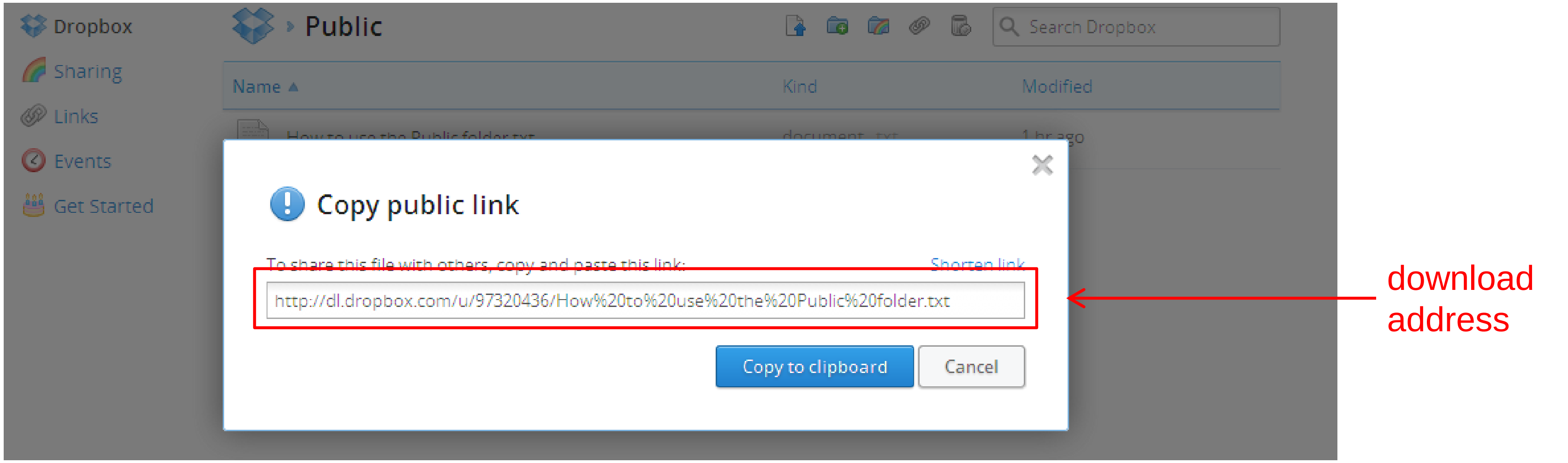
Motivation

- 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 was the Dropbox network (before 2016)?

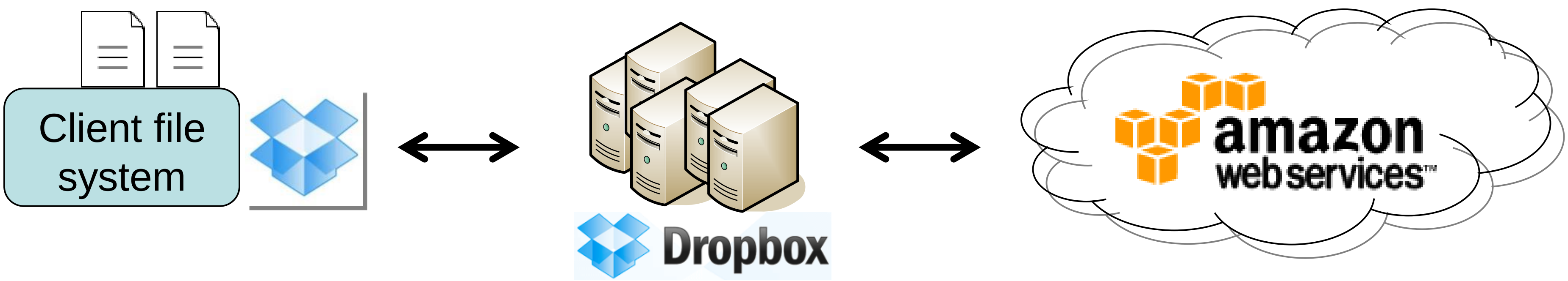


$ nslookup dl.dropbox.com

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

- Before 2016, AWS empowered 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

Pricing as of Oct 2014

	Standard Storage	Reduced Redundancy Storage	Glacier Storage
First 1 TB / month	$0.0300 per GB	$0.0240 per GB	$0.0100 per GB
Next 49 TB / month	$0.0295 per GB	$0.0236 per GB	$0.0100 per GB

- Dropbox also charges: **https://www.dropbox.com/plans**

Pricing as of Oct 2014

Dropbox Basic	Dropbox Pro	Dropbox for Business
Free	$9.99 / month	$15 / user / month
	Dropbox Basic plus:	**Dropbox Pro plus:**
2 GB of space	1 TB (1,000 GB) of space	Centralized admin controls
Safe, reliable backup	Additional sharing controls	Unlimited version history
Access from anywhere	Remote wipe	Comprehensive audit log
Simple file sharing		
Your current plan	Upgrade to Dropbox Pro	Learn more

1000GB storage/month:
Amazon: $30/month
Dropbox: $9.99/month

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

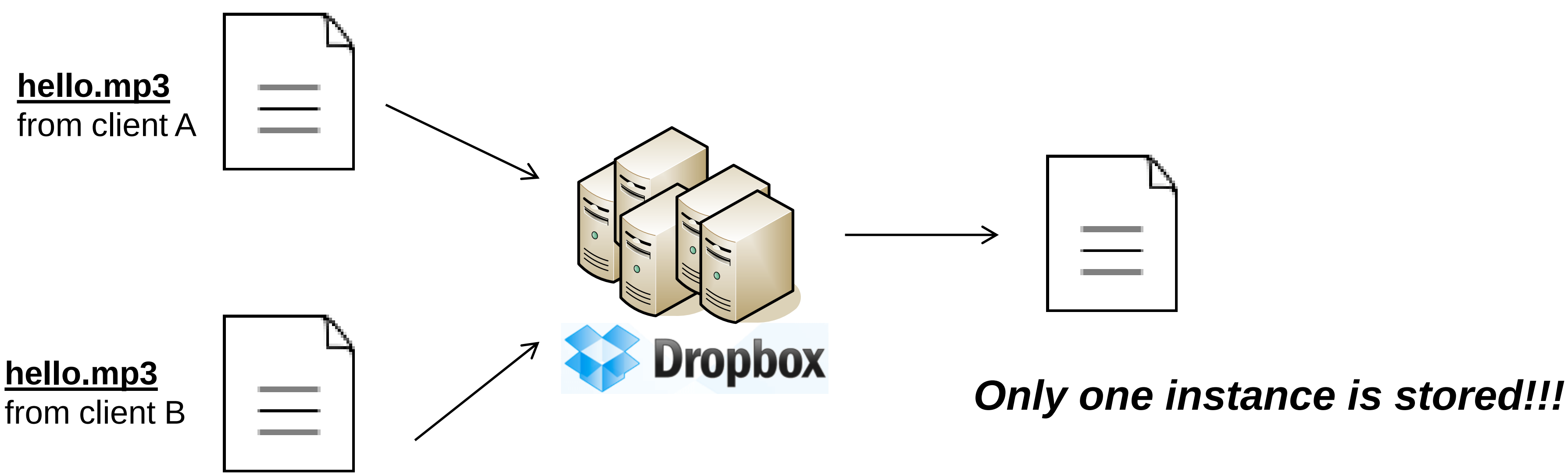
Motivation

- As of today (2017), Dropbox hosts files of over 500 million users in its own data centers
  - https://www.wired.com/2016/03/epic-story-dropboxs-exodus-amazon-cloud-empire/
  - https://techcrunch.com/2017/09/15/why-dropbox-decided-to-drop-aws-and-build-its-own-infrastructure-and-network/
- Still, a majority of users are free riders

Motivation

- Most Dropbox users use free accounts

- Dropbox implements **deduplication**

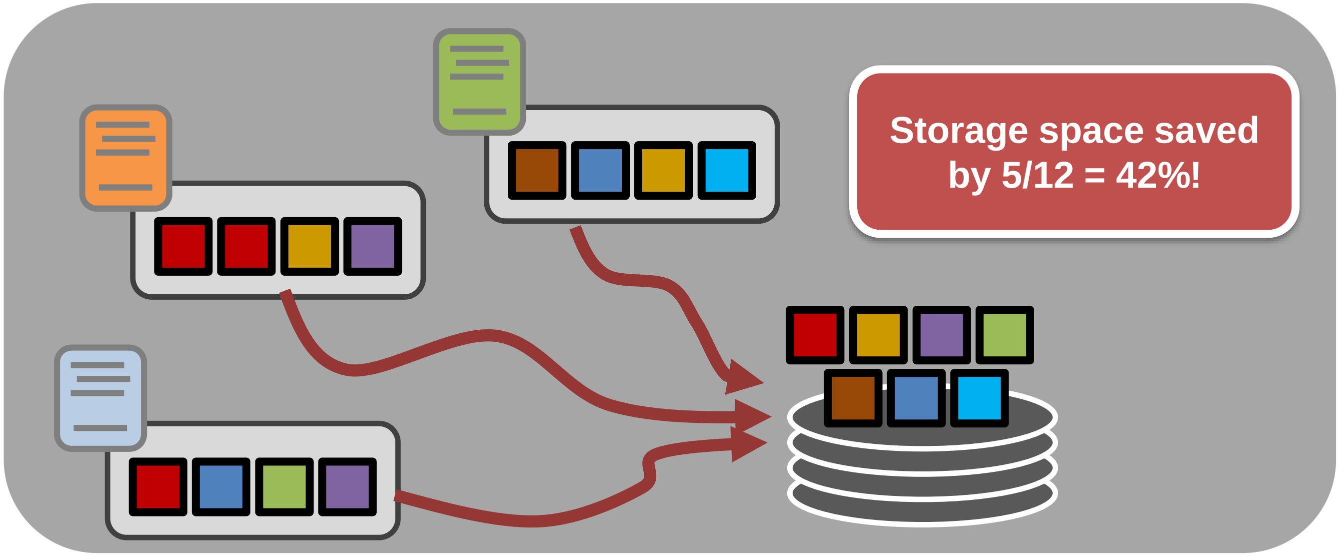


Motivation

- 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

- Deduplication → coarse-grained compression
- Unit: **chunk** (fixed- or variable-size)
- Store only one copy of chunks with same content; other chunks refer to the copy by references (pointers)



Deduplication vs. Compression

- Compression reduces the amount of stored bits compared to the original data
  - Transforms data into new representation with higher entropy
  - Typically works on a single file (or a single batch of files)
  - Byte-level, fine-grained compression
- Deduplication
  - Detects identical data chunks / similarities between data chunks
  - Works across files (e.g., archives, backups, or collections of virtual machine images)
  - Coarse-grained compression
- In practice, compression has to be performed after deduplication
  - What about deduplication after compression?

Data Reduction Workflow

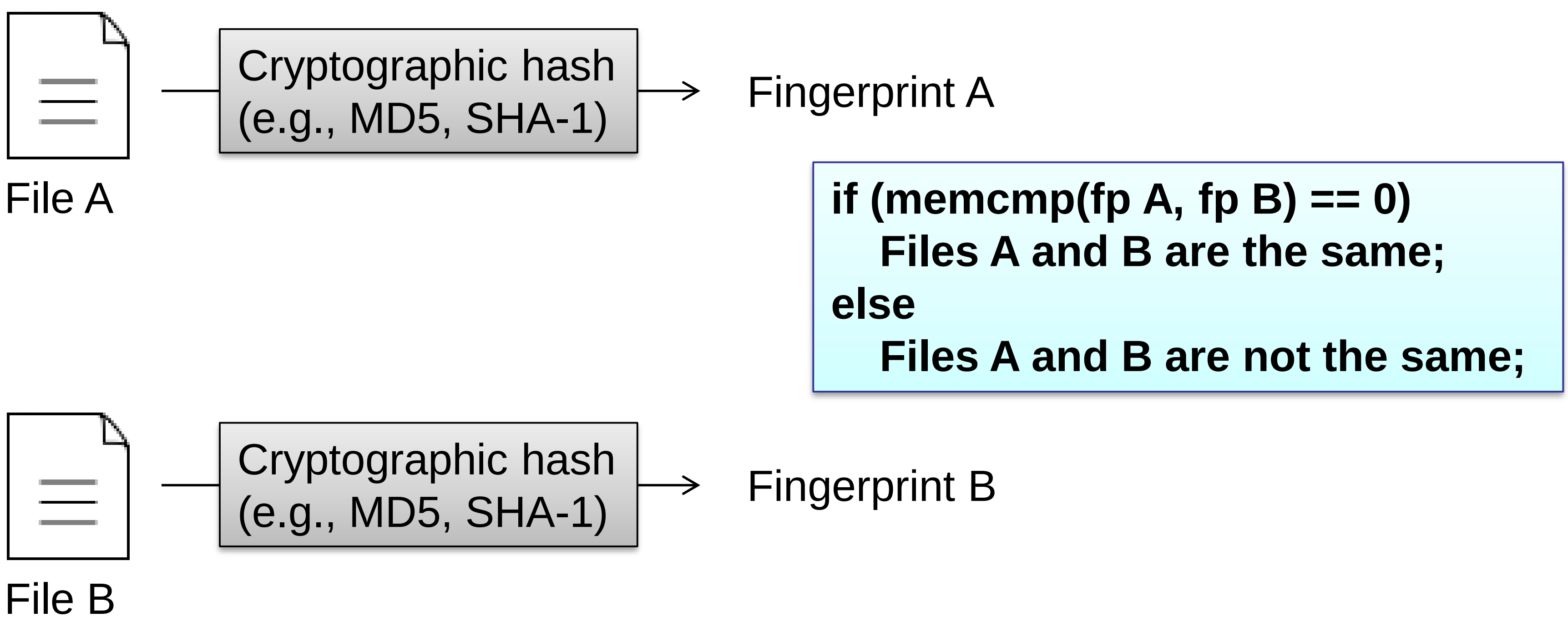
- **Deduplication**
  - Remove identical chunks
- **Delta compression** (or delta encoding)
  - Remove similar chunks by storing differences of two chunks
  - E.g., if chunk A has been stored and a new chunk A’ arrives, store “A’ – A”
- **Compression**
  - Remove byte-level redundancy within each chunk

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

- 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:
  - “*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 [USENIX Security'11]:
  - 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**
  - 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

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	Average chunk size	Disk seeks CPU time to determine chunk boundaries Fingerprint calculations	Highest

Comparison on chunking strategies

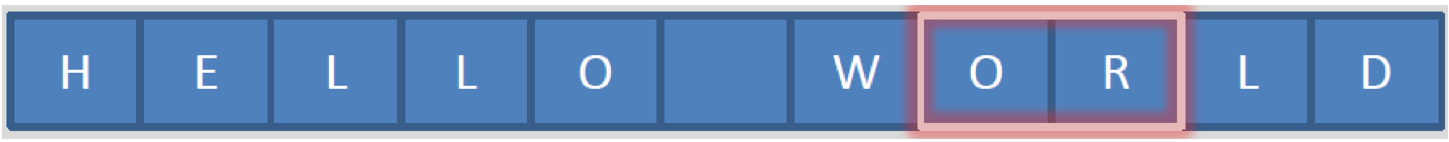
Chunking

- 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))

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 f_1

`j;la` ⟶ fingerprint f_2

`fjd;s` ⟶ fingerprint f_3

`afdaj` ⟶ fingerprint f_4

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 $0 \le t_m \le 9$

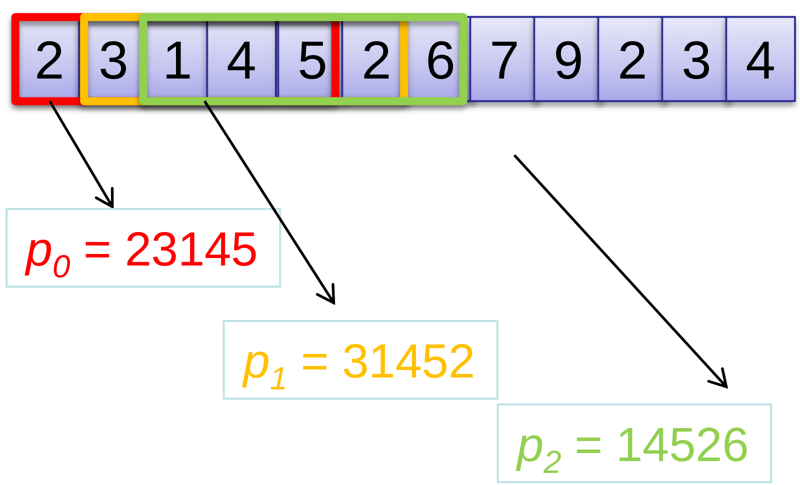
$$p_s = \sum_{i=1}^{m} t_{s+i} \times 10^{m-i}$$

where
(1) $s \ge 0$
(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!

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

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:

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

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 $0 \le t_m \le 9$

p_s can be rewritten as:

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

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**!

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

RFP: Polynomial Representation

- Complexity:
  - O(m): to compute p_0
  - O(1): to compute each 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***.

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

RFP: Modular Arithmetic

Properties of modular arithmetic:

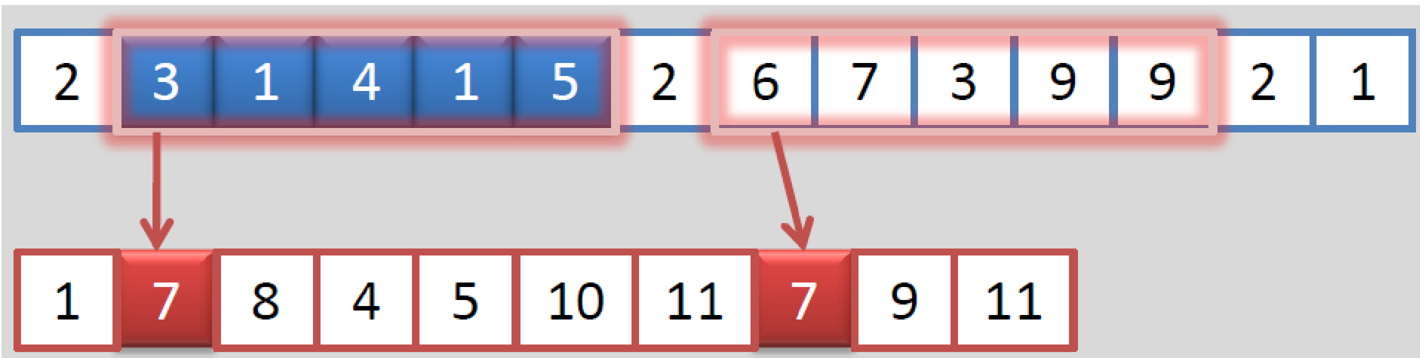
```
1.[(a mod n) + (b mod n)] mod n = (a + b) mod n
2.[(a mod n) - (b mod n)] mod n = (a - b) mod n
3.[(a mod n) x (b mod n)] mod n = (a x b) mod n
```

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

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

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***

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

- 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?

RFP Summary

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

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

- 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 $\mathbf{2^{31}}$ is a good choice?

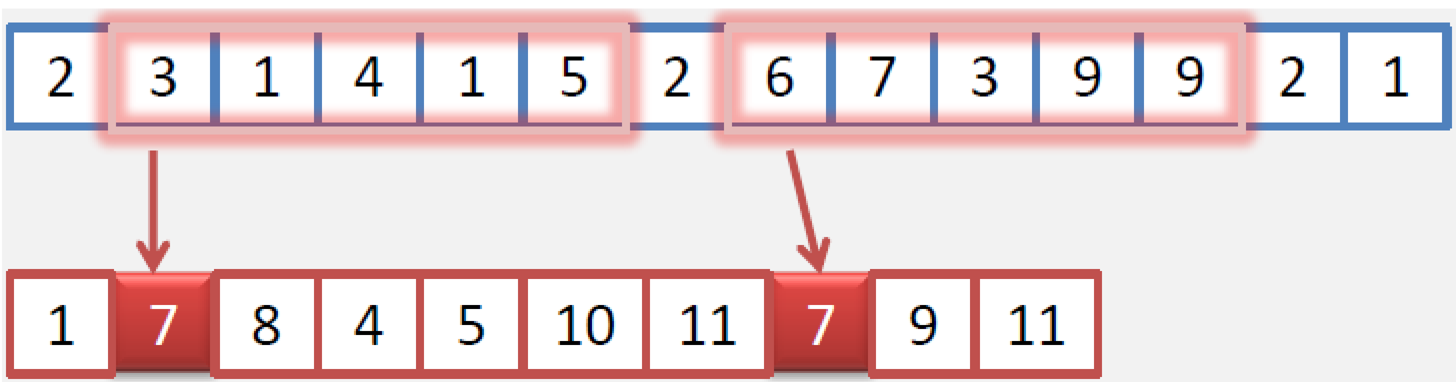
RFP on Deduplication

- How is RFP applied into **deduplication**?
- Flow:
  - (1) Select critical RFP value
  - (2) Select “m” as the parameter of the solution
  - (3) Use the critical RFP value to divide chunks
- Idea:
  - RFPs provide a guess of similar chunks
    - If two chunks have different RFPs, 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 critical RFP value
  - Solution: only interested in the RFPs that share the common properties
  - Example: we select the critical RFP value = 7



- Record the positions whose RFPs are equal to the critical RFP value

RFP on Deduplication

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

- To make RFP computation fast, we choose the critical RFP value whose ***least significant k bits equal to 0***
  - In fact, it doesn't matter to the correctness if we choose other critical RFP values, but the performance may not be as good as the above choice
  - On average, we'll store one RFP for each 2^k characters (assuming the bytes in the string are uniformly distributed)

RFP on Deduplication

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

RFP on Deduplication

- Step (3): for each file, we record the patterns that match the critical RFP value

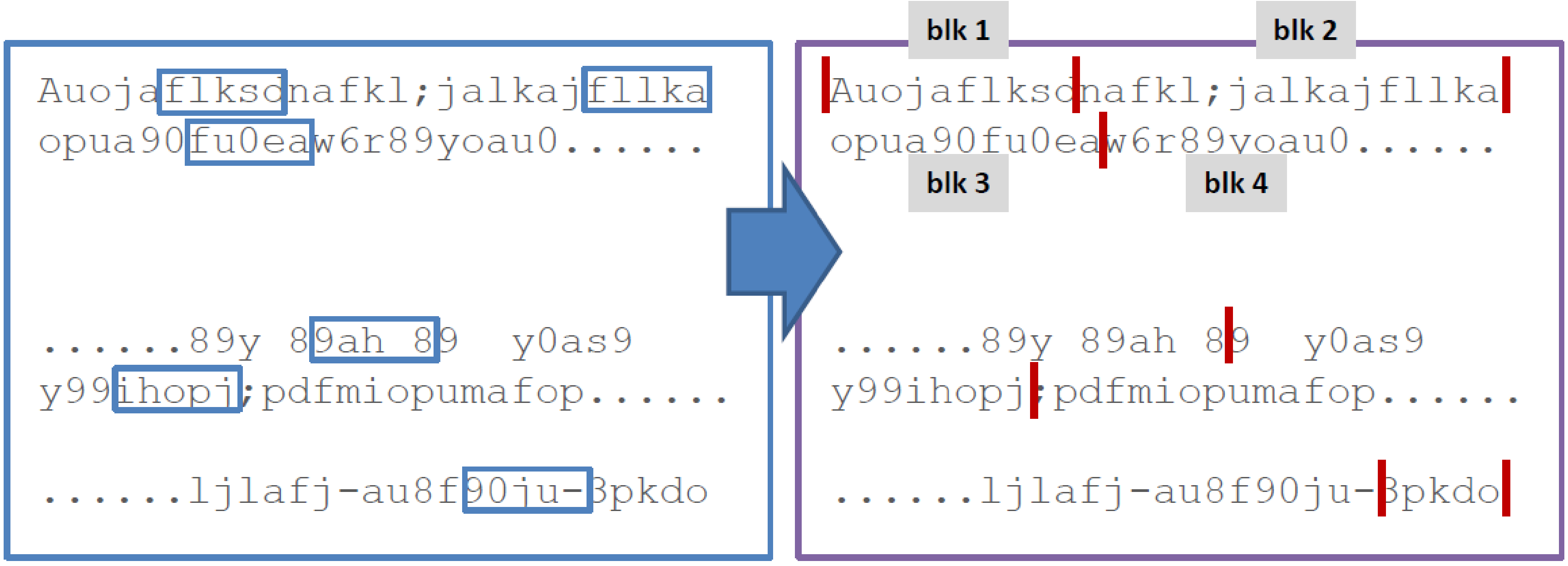
```
The quick brown fox jumps
over the lazy dog
```

```
The sentence is: The quick
brown fox jumps over the
lazy dog
```

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

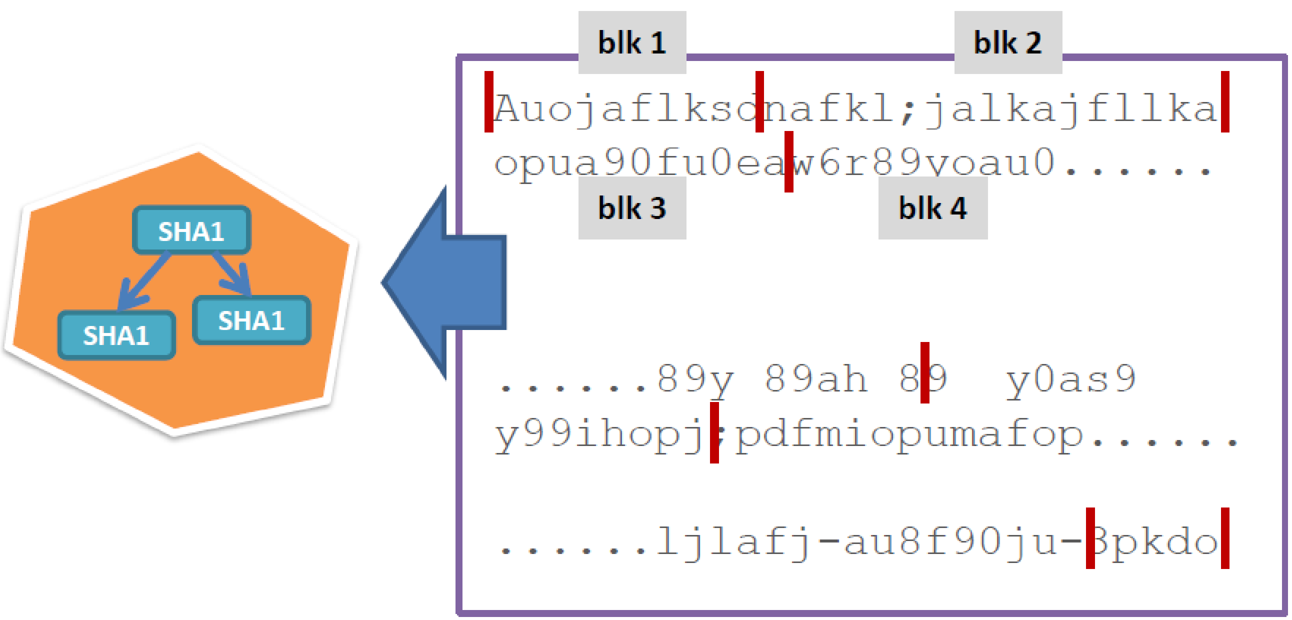
RFP on Deduplication

- RFP is used for determining chunk boundaries.
  - The chosen set of fingerprints creates **anchor points**.
  - It results in variable-sized chunks.
  - 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	Minimum chunk size
Multiplier, d	Usually large than the number of possible inputs, e.g., 257 is a good choice for binary files.
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. It tells you the average chunk size
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.
Max chunk size	Maximum chunk size

Homework Questions

- What if the input contains many runs of zeros?
- What if the inputs are expected to have very small size, even smaller than the minimum chunk size?
  - Chunking is infeasible

Case Study

- Analysis of deduplication effectiveness on 52 pre-made virtual disk images
- Fixed-size chunking is even more effective than variable-size chunking for VM image deduplication

Index	Name and version	Kernel	File system	Desktop	Image size	Disk type
1	Arch Linux 2008.06	Linux 2.6.25-ARCH	ext3	GNOME	3.5G	XS
2	CentOS 5.0	Linux 2.6.18-8.el5	ext3	None	1.2G	XS
3	CentOS 5.2	Linux 2.6.18-92.1.10.el5	ext3	GNOME	3.3G	MS
4	DAMP	Dragonfly 1.6.2-RELEASE	ufs	None	1.1G	MF
5	Darwin 8.0.1	Darwin 8.0.1	HFS+	None	1.5G	MF
6	Debian 4.0.r4	Linux 2.6.18-6-486	ext3	None	817M	XS
7	Debian 4.0	Linux 2.6.18-6-686	ext3	GNOME	2.5G	MS
8	DesktopBSD 1.6	FreeBSD 6.3-RC2	ufs	KDE	8.1G	XF
9	Fedora 7	Linux 2.6.21-1.3194.fc7	ext3	GNOME	2.9G	XS
10	Fedora 8	Linux 2.6.23.1-42.fc8	ext3	GNOME	3.4G	XS
11	Fedora 9 en-US	Linux 2.6.25-14.fc9.i686	ext3	GNOME	3.4G	XS
12	Fedora 9 fr	Linux 2.6.25-14.fc9.i686	ext3	GNOME	3.6G	XS
13	FreeBSD 7.0	FreeBSD 7.0-RELEASE	ufs	None	1.2G	XS
14	Gentoo 2008.0	Linux 2.6.24-gentoo-r8	ext3	Xfce	5.5G	XS
15	Gentoo 2008.0 with LAMP	Linux 2.6.25-gentoo-r7	ext3	None	8.1G	XF
16	Knoppix 5.3.1	Linux 2.6.24.4	ext3	KDE	13G	XS

A subset of VM images being studied

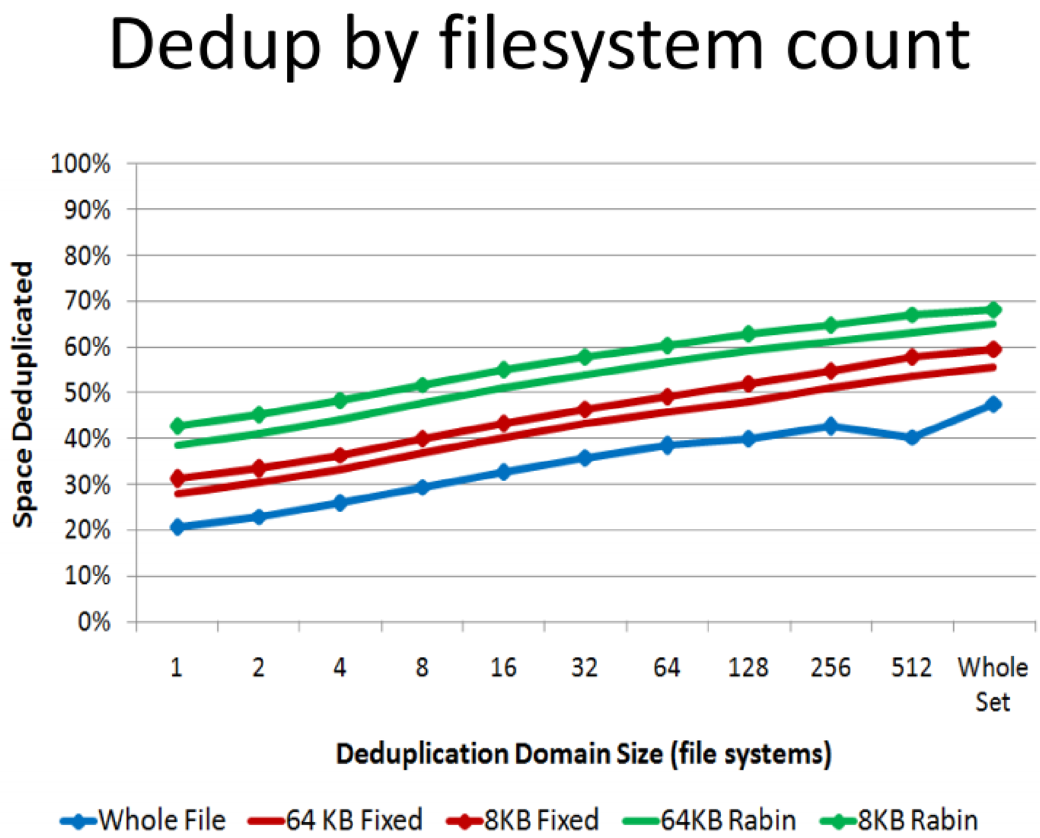
Jin and Miller, “The Effectiveness of Deduplication on Virtual Machine Disk Images”, SYSTOR 2009

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