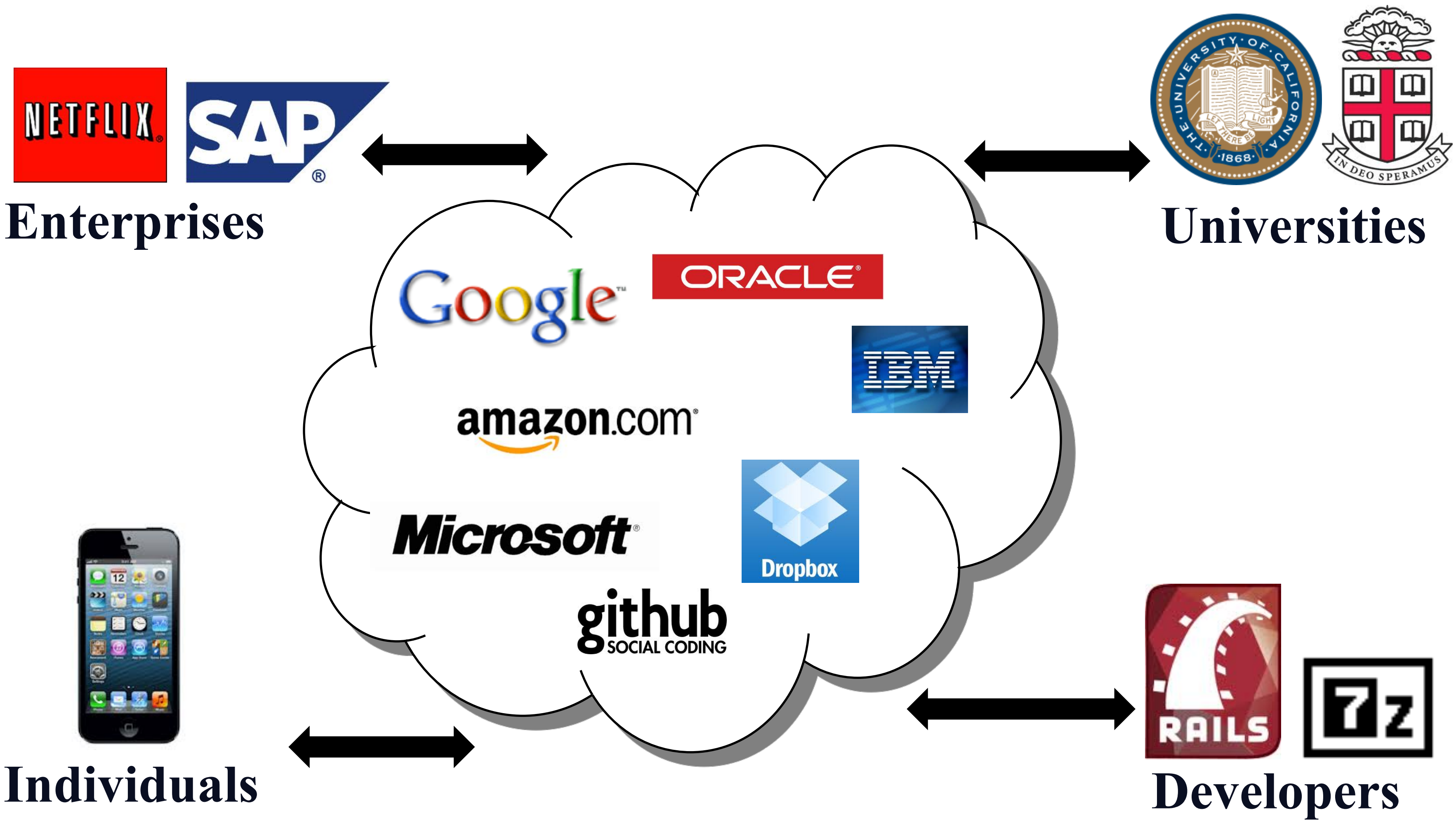
ENEE 457: Computer Systems Security
11/28/16

Lecture 23
Secure Storage

Charalampos (Babis) Papamanthou

Department of Electrical and Computer Engineering
University of Maryland, College Park

Cloud computing today

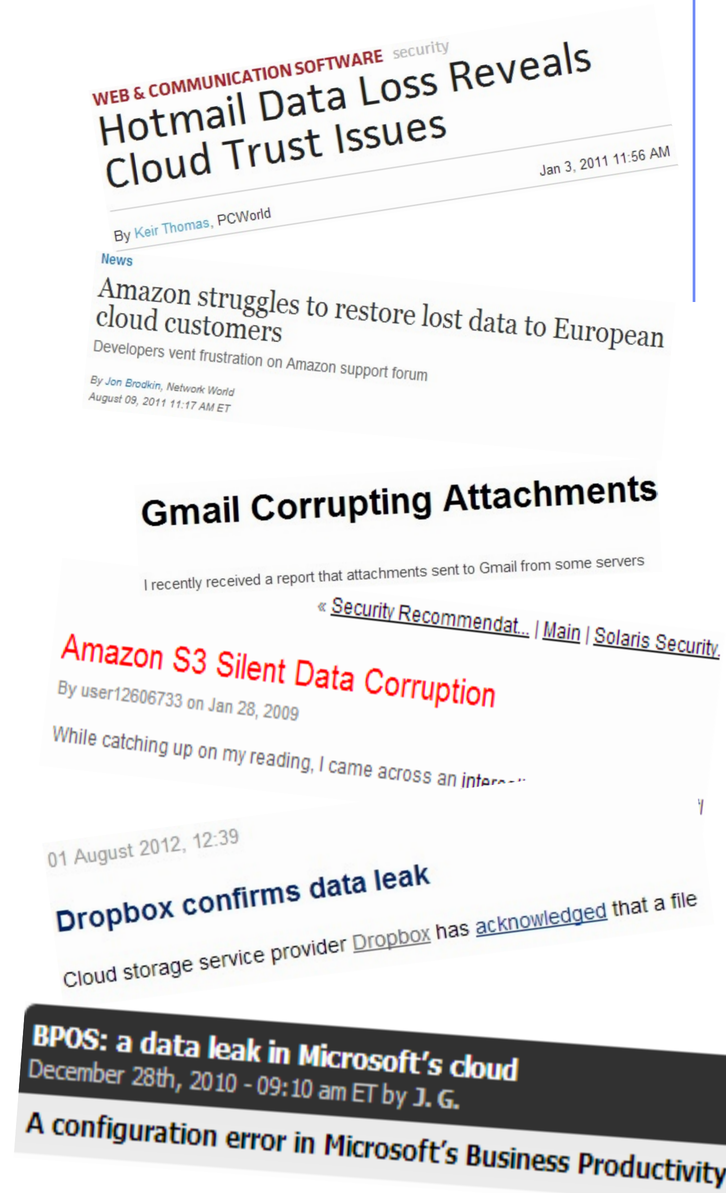


Are there any threats?

- Cloud providers are untrusted
  - Can **lose** data
  - Can return **corrupted** results
  - Can **leak** information

…we will have no liability to you for any unauthorized access or use, corruption, deletion, destruction or loss of any of your content or applications…

Amazon web service customer agreement
`http://aws.amazon.com/agreement/`



Do people care?

- Customers are paying for the services
  - They want **reliable** storage
  - They want **correctness** guarantees
  - They want to keep their **privacy**

*…**58%** of the public and **86%** of business leaders are excited about the possibilities of cloud computing. But more than **90%** of them are worried about security, availability, and privacy of their data as it rests in the cloud…*

Microsoft survey in 2010

http://news.cnet.com/8301-1009_3-10437844-83.html

What do I do?

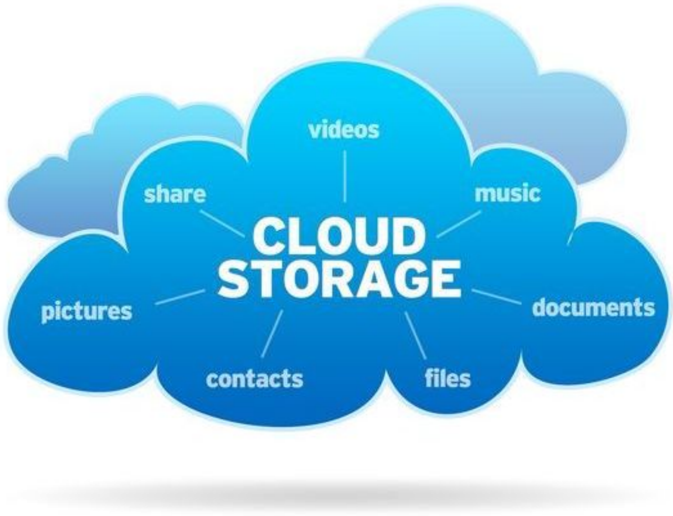
- **Enable people to use the cloud safely**
- **Verifiability in the cloud**
  - **Verify** that the cloud did the work correctly
- **Privacy in the cloud**
  - Use the cloud in a **privacy-preserving** manner

efficient both in theory and in practice

provably secure

no assumptions at the server

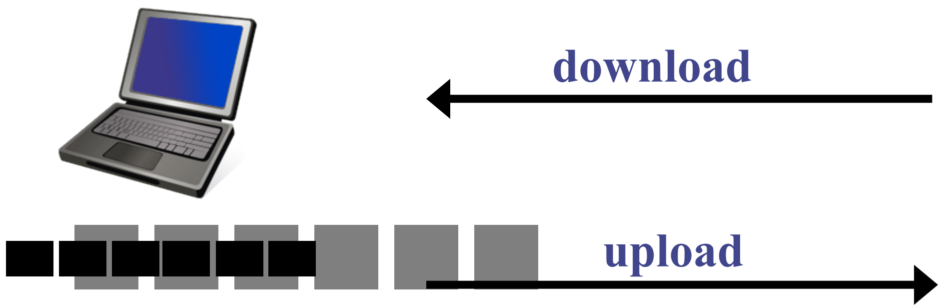
Secure Cloud Storage

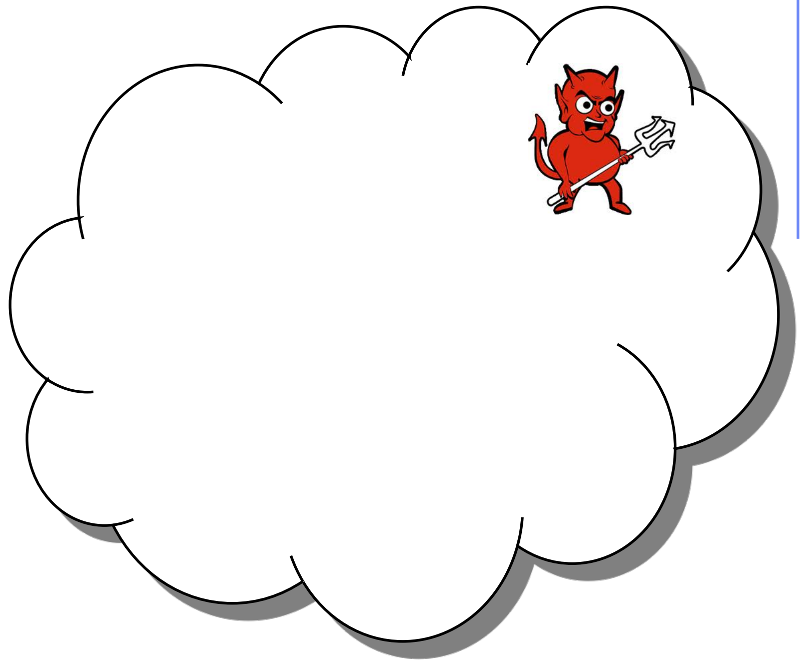


Security Framework

- We must make sure our files have not changed since they were uploaded
- We are going to ask the server that stores our files to compute a "proof" that he stores our files intact
- Central to the rest of the talk:
  - Cryptographic hash function, e.g., SHA256
  - **Let's try it out**

Storing your files in the cloud: Hash-based

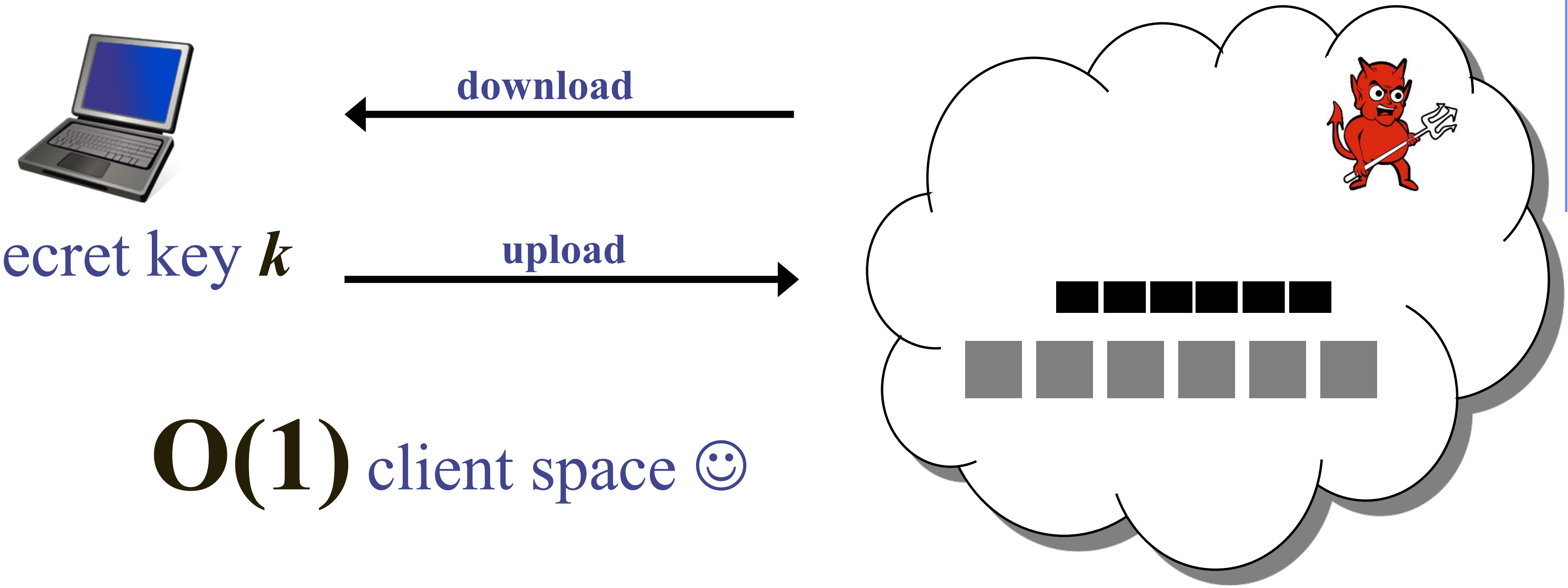




O(*n*) client space ☹

- How to verify that a file has **not** been corrupted?
  - Keep a hash (i.e., checksum) **locally** for each file
  - Download: **recompute** and **check**
  - Upload: **compute** and **store** new hash

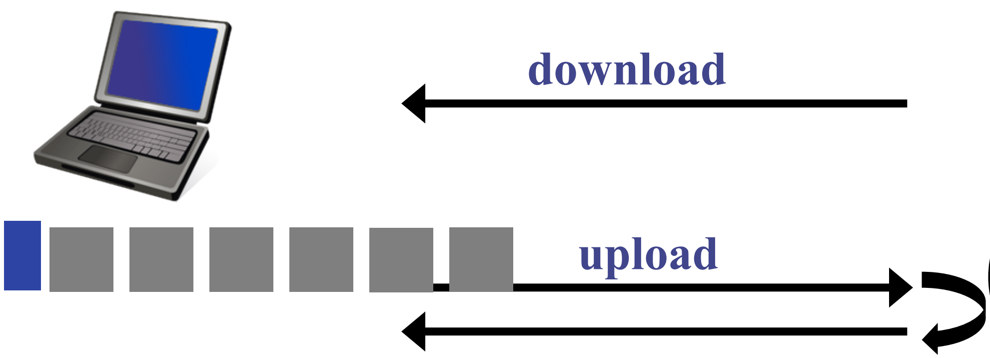
Constant space? MAC-based

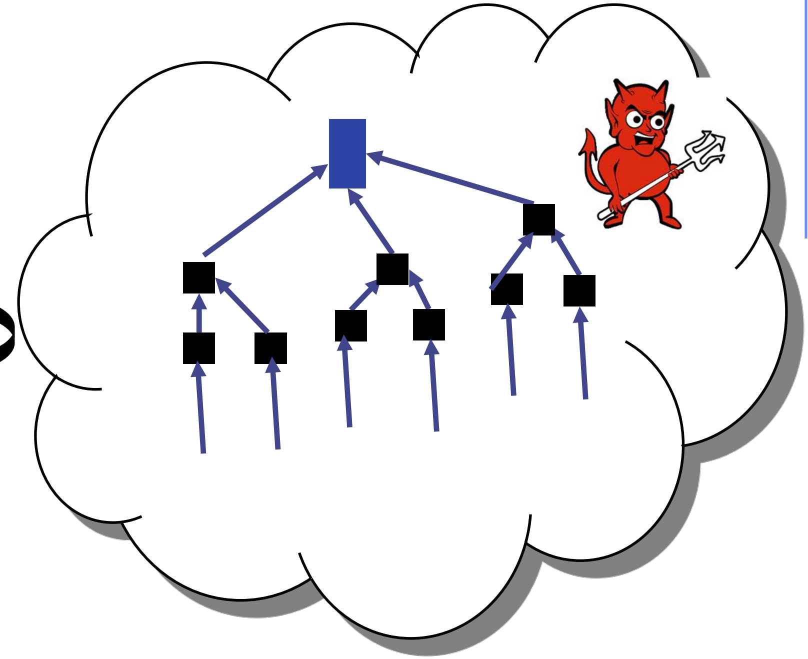


O(1) client space ☺

- How to verify that a file has **not** been corrupted?
- Compute a MAC for each file using a secret key
  - Store only the secret key!
  - Download: **recompute** and **check**
  - Upload: **?**

What about replay attacks? Tree-based



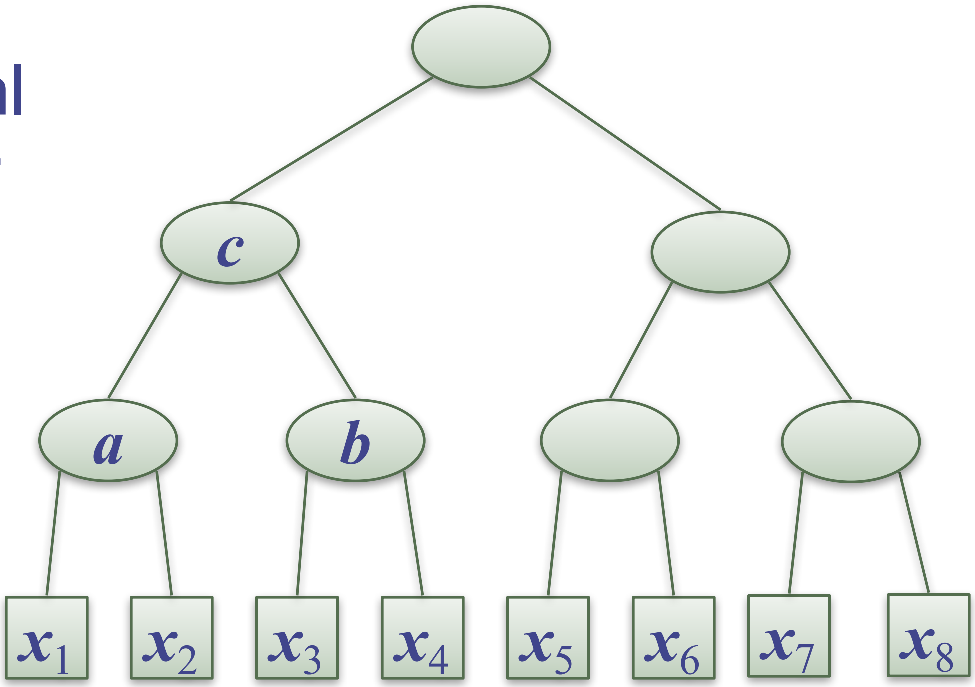


O(1) client space ☺

- Hashing over a tree and store only the roothash
  - Download: Fetch **O(log *n*)** hashes
  - Upload: An **interactive** protocol

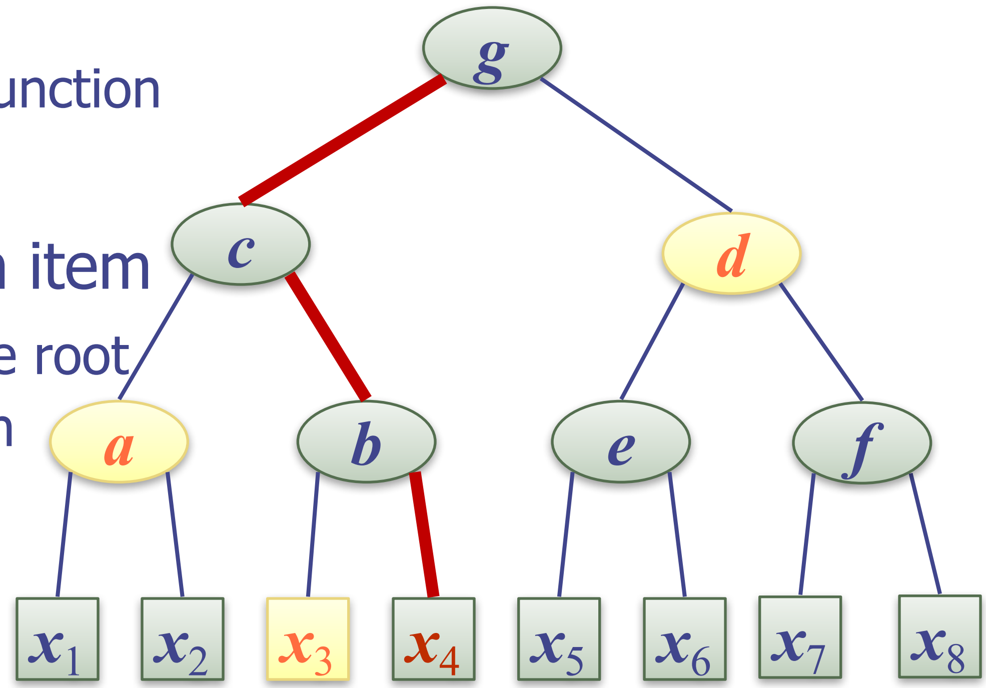
Hash Tree: Details

- Balanced binary tree defining a hierarchical hashing scheme over a set of items
  - $a = h(x_1, x_2)$
  - $b = h(x_3, x_4)$
  - $c = h(a, b)$
  - …
- The root hash is a hierarchical digest of entire set
- [Merkle]



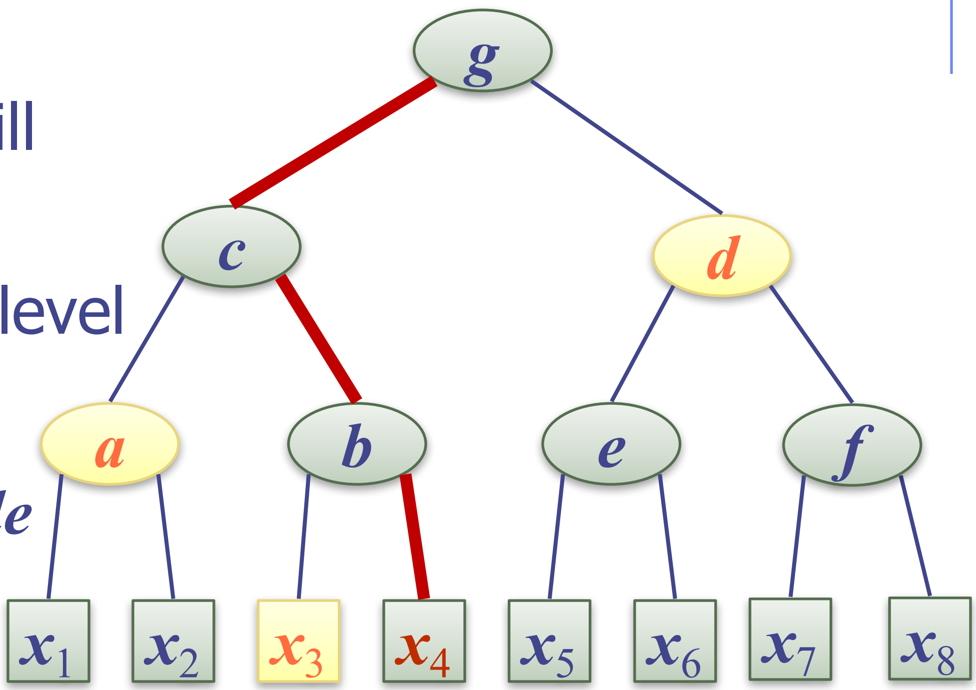
Hash Tree Verification

- Assumptions
  - Collision resistant hash function
  - Root hash is known
- Membership proof of an item
  - path from the item to the root (L/R sequence) plus hash values of sibling nodes
  - logarithmic size
  - logarithmic verification time
- Example
  - $g = h(h(a, h(x_3, x_4)), d)$



Proof intuition

- In order to provide a verifying proof for a different leaf element, the adversary will have to break collision resistance in at least one level of the tree
- ***This happens with negligible probability***



Recap:
Solutions to cloud based storage

Approach	Client Space	Proof Size	Verification	Proof Computation	Updates?
Hash all	O(n)	O(1)	O(1)	O(1)	O(1)
MAC	O(1)	O(1)	O(1)	O(1)	NO
Merkle tree	O(1)	O(log n)	O(log n)	O(log n)	O(log n)

- Can you make everything constant?
- ***Impossible under certain assumptions***

Other considerations on storage

How can you verify **all the files more efficiently**?

How do I make sure that server is storing all my data?

How do I make sure that server is storing all my data?

Idea 1: Download & check all blocks

How do I make sure that server is storing all my data?

Idea 1: Download & check all blocks

… but is expensive

How do I make sure that server is storing all my data?

- Idea 2: Probabilistically download and check a small subset of blocks

If server erases t out of n blocks, what is the probability of passing an audit?

Suppose k random blocks are checked during an audit

If server erases t out of n blocks, what is the probability of passing an audit?

Suppose k random blocks are checked during an audit

$$\Pr[\text{pass audit}] = (1-t/n)^k$$

If server erases t out of n blocks, what is the probability of passing an audit?

Suppose k random blocks are checked during an audit

$$Pr[\text{pass audit}] = (1-t/n)^k$$

- If $t = n/2$:

  $Pr[\text{pass audit}] = 2^{-k}$, i.e., negligible in k

If server erases t out of n blocks, what is the probability of passing an audit?

Suppose k random blocks are checked during an audit

Suppose t blocks have been tampered by the server

$$\Pr[\text{pass audit}] = (1-t/n)$$

- If $t = n/2$:

  $\Pr[\text{pass audit}] = 2^{-k}$, i.e., negligible in k

- If $t = 1$:

  Even if the client checks $n/2$ blocks,

  $\Pr[\text{pass audit}] = (1-1/n)^{n/2}$

  For n=1000, Pr[pass audit] = 0.6

How do I make sure that server is storing all my data?

Idea 2: Probabilistically download and check a small subset of blocks

Proof of data possession: fail to detect small number of erasures with significant probability.

Proofs of Retrievability

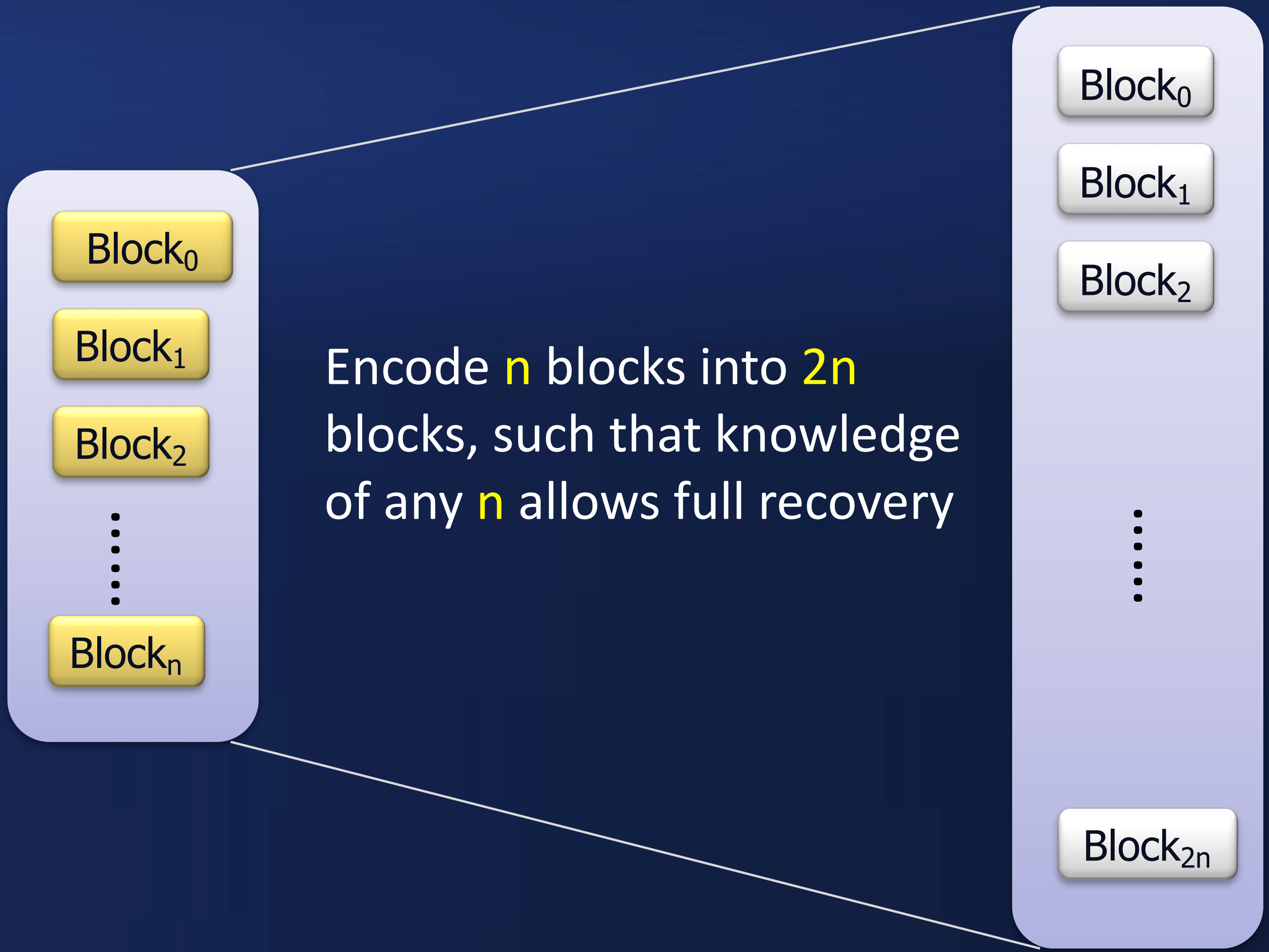
Even when a single block is lost, the client can detect with overwhelming probability.

Boosting the Probability of Detection?

Boosting the Probability of Detection?

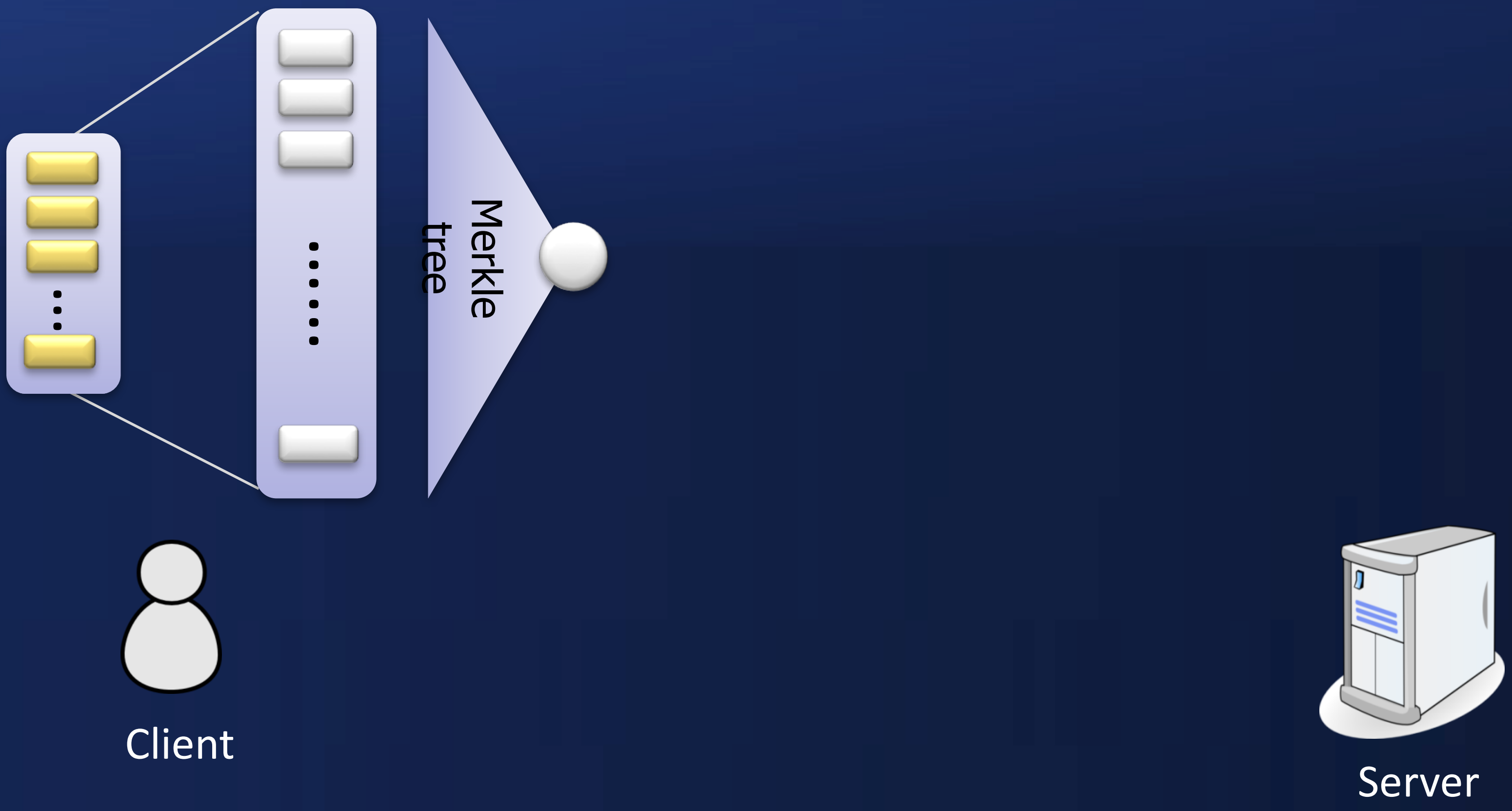
Use erasure coding, *s.t.* the server needs to delete at least n/2 blocks to cause actual data loss

Erasure Coding

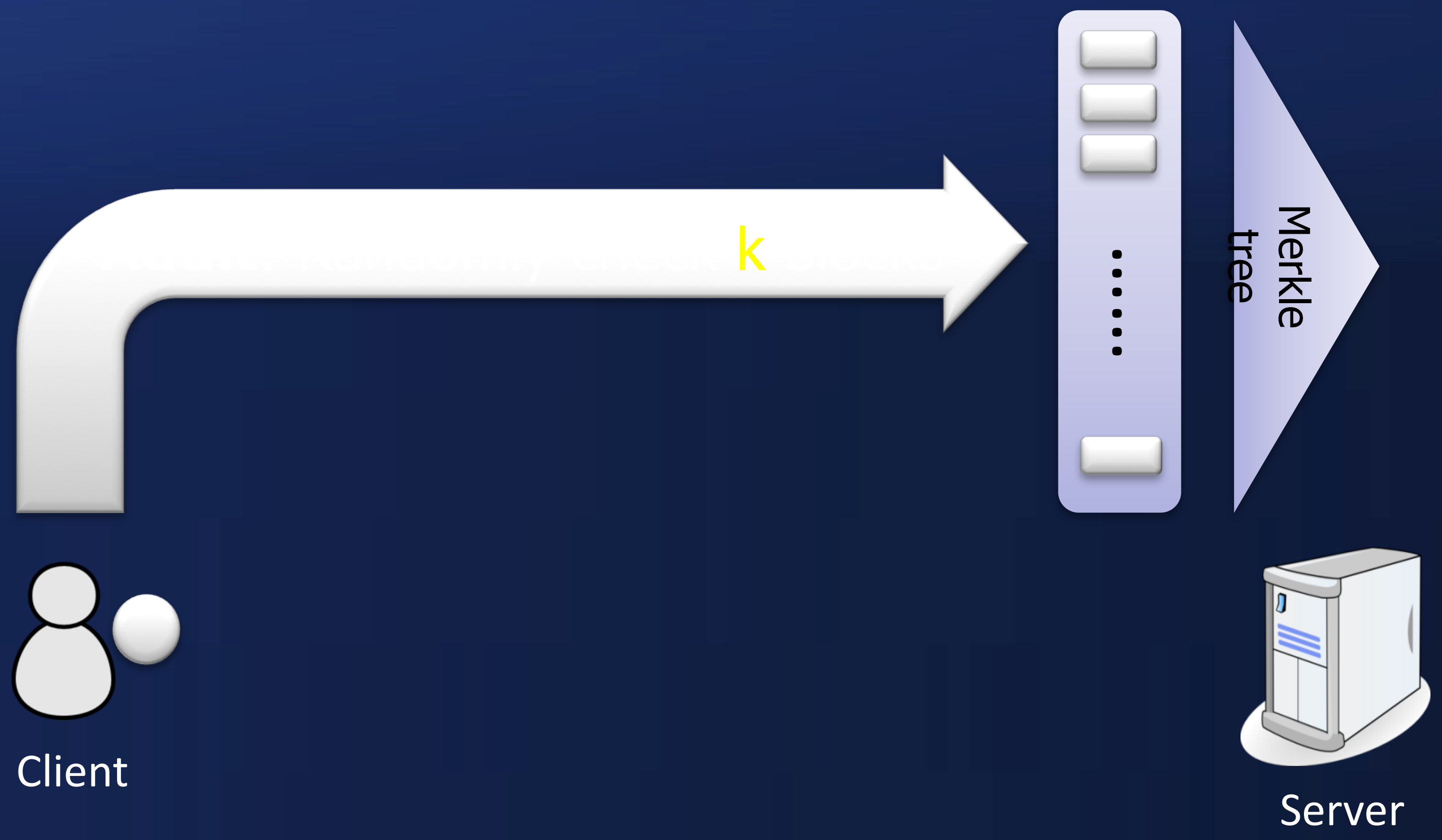


Encode n blocks into 2n blocks, such that knowledge of any n allows full recovery

Proofs of Retrievability



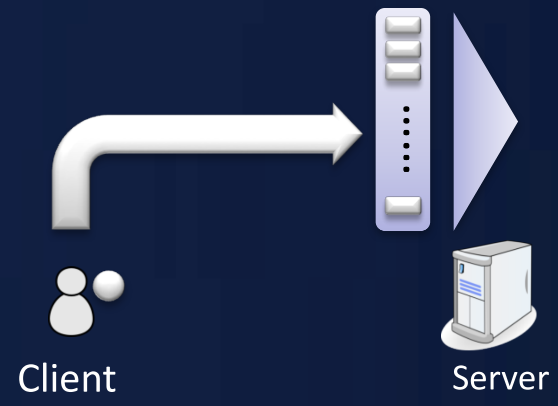
Proofs of Retrievability



Proofs of Retrievability

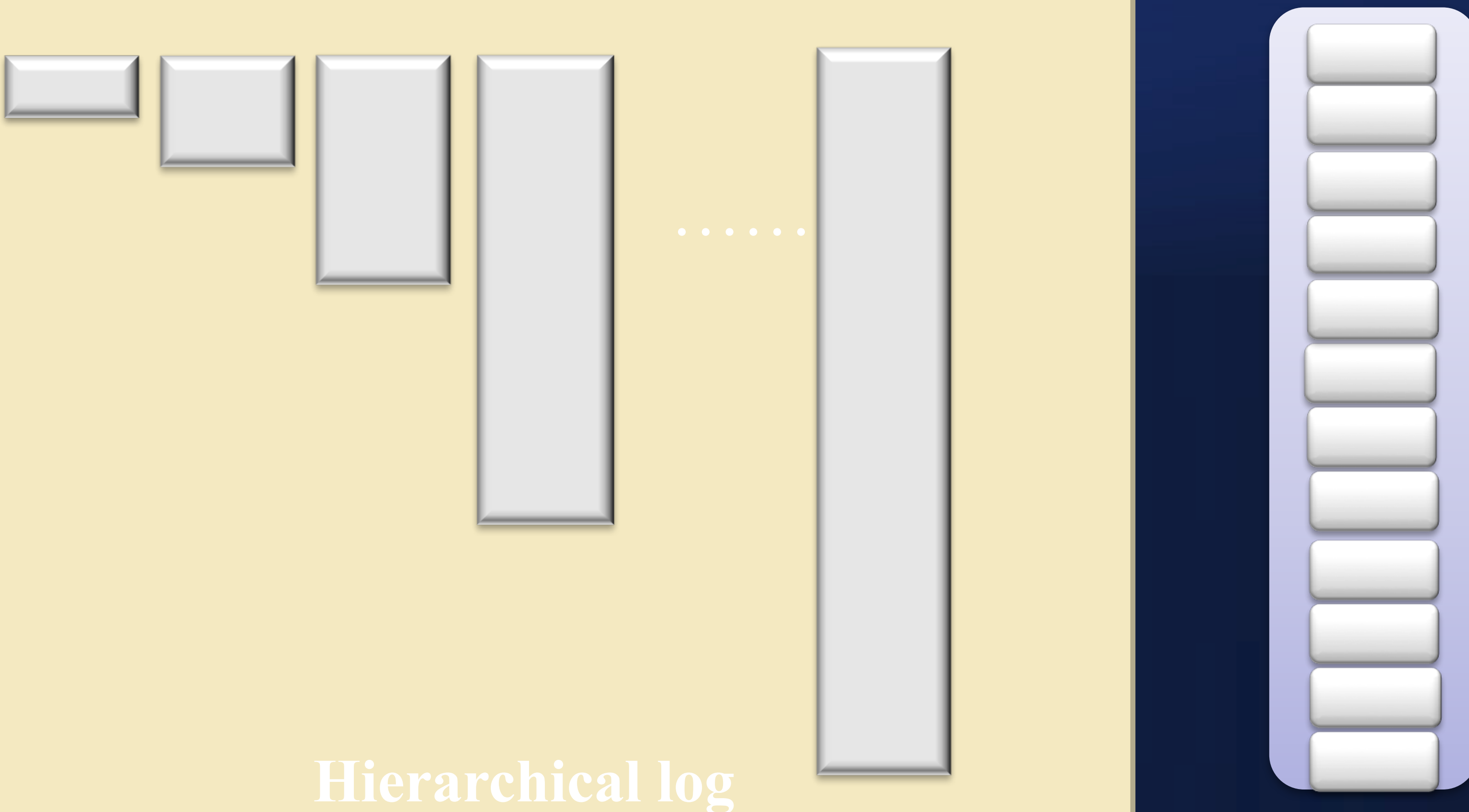
If data loss has occurred, then server must have erased more than n out of 2n blocks

Audit will detect this with probability **$1-2^{-k}$**

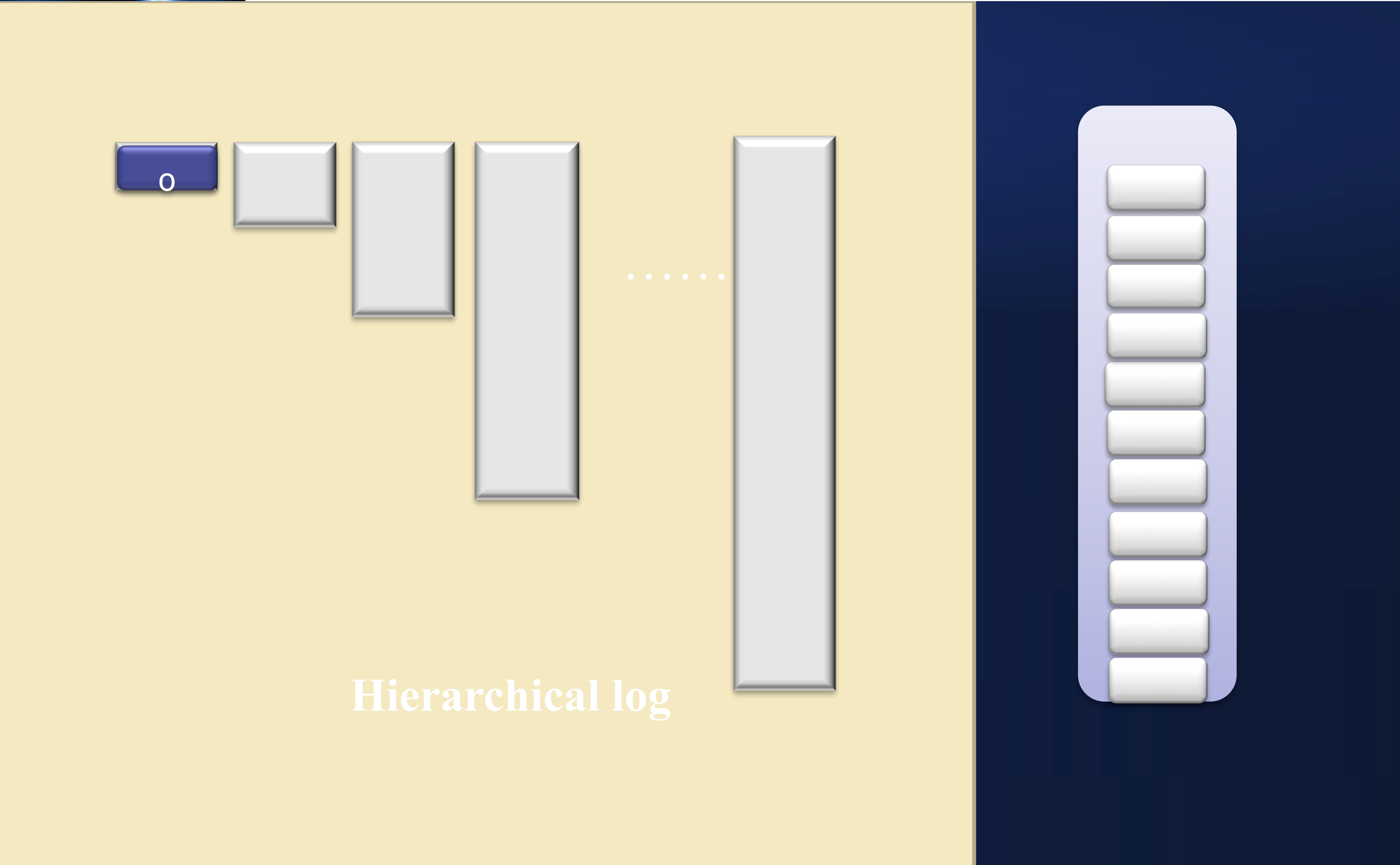


How to support updates efficiently?

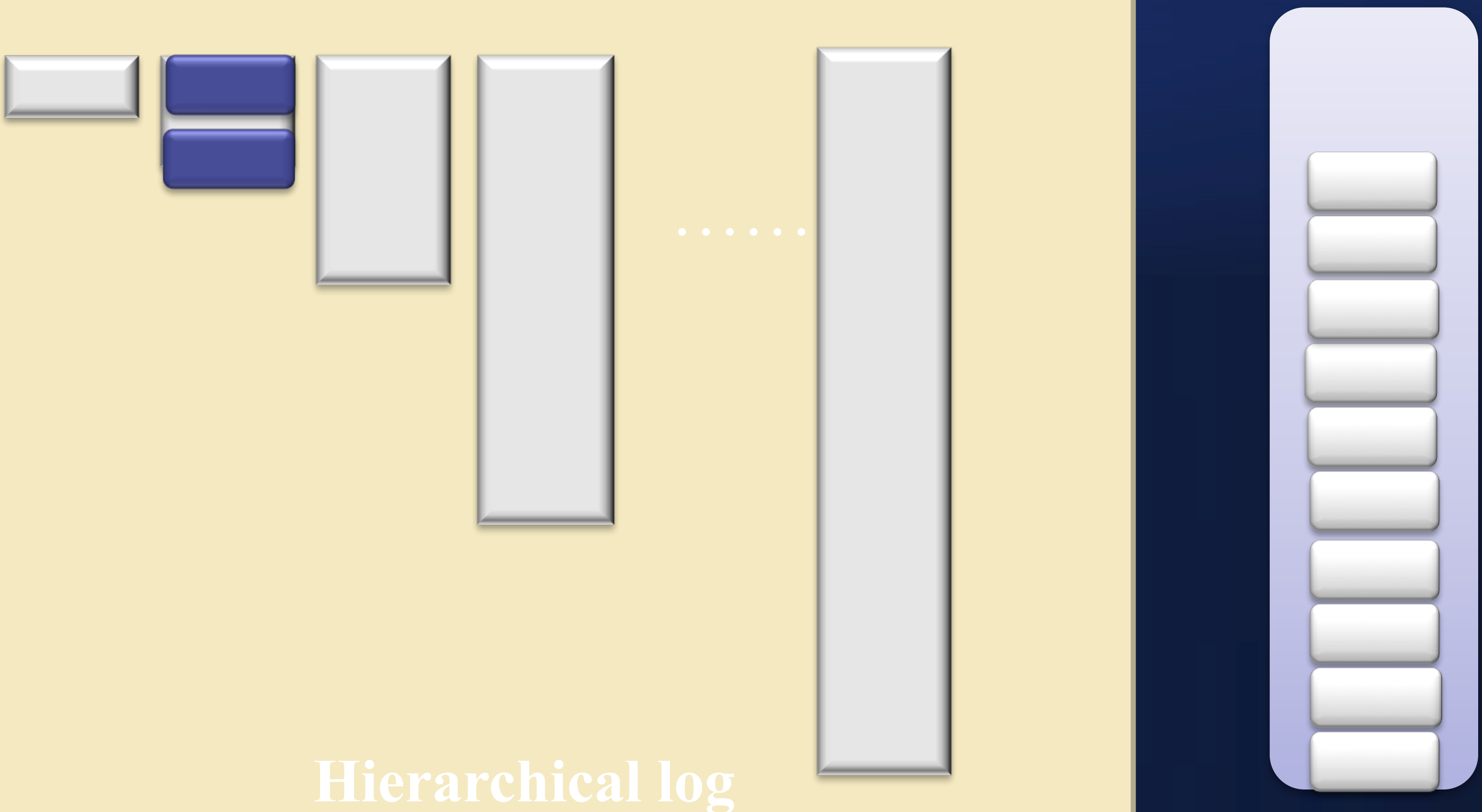
Update buffer



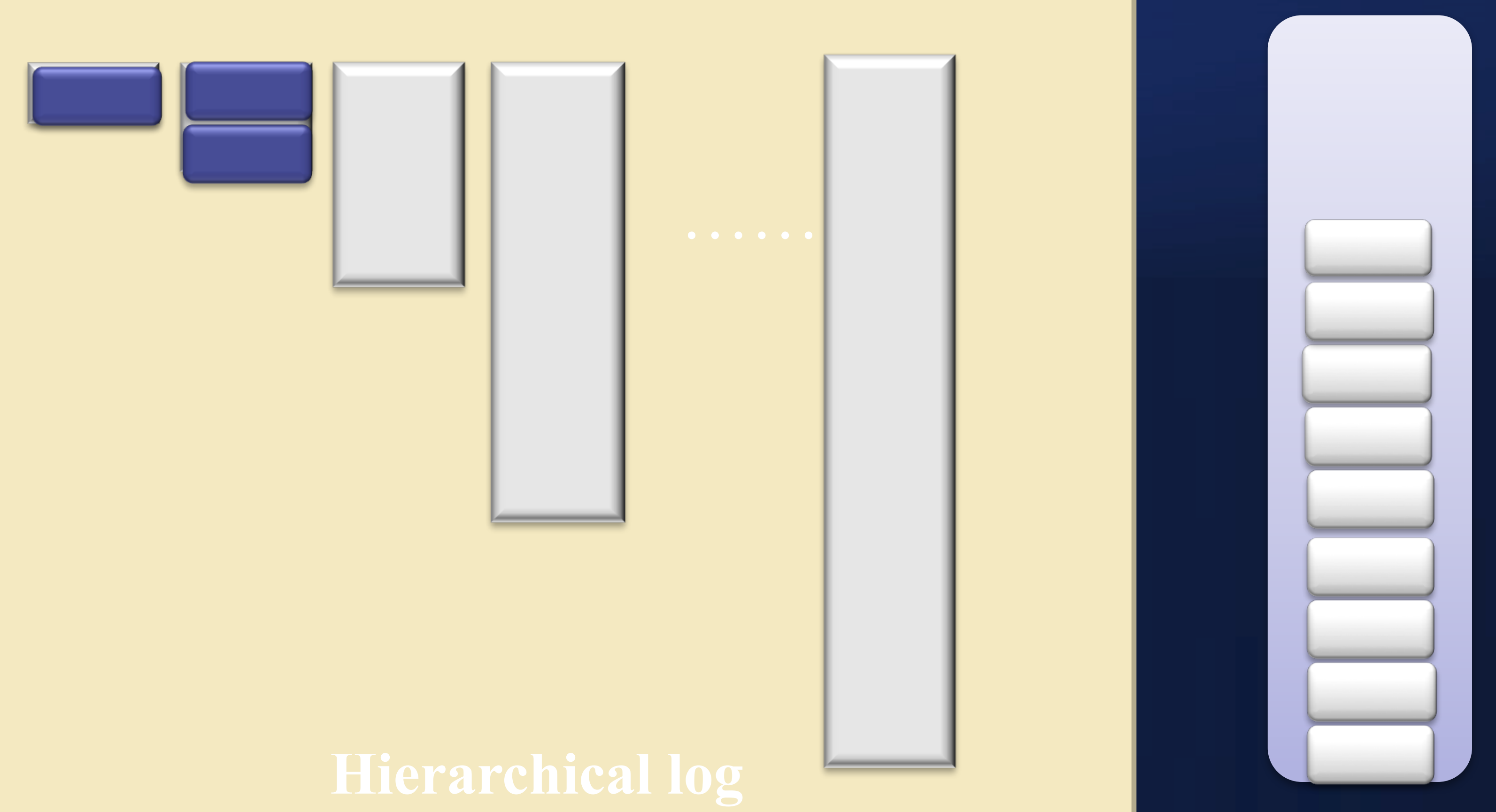
Update buffer



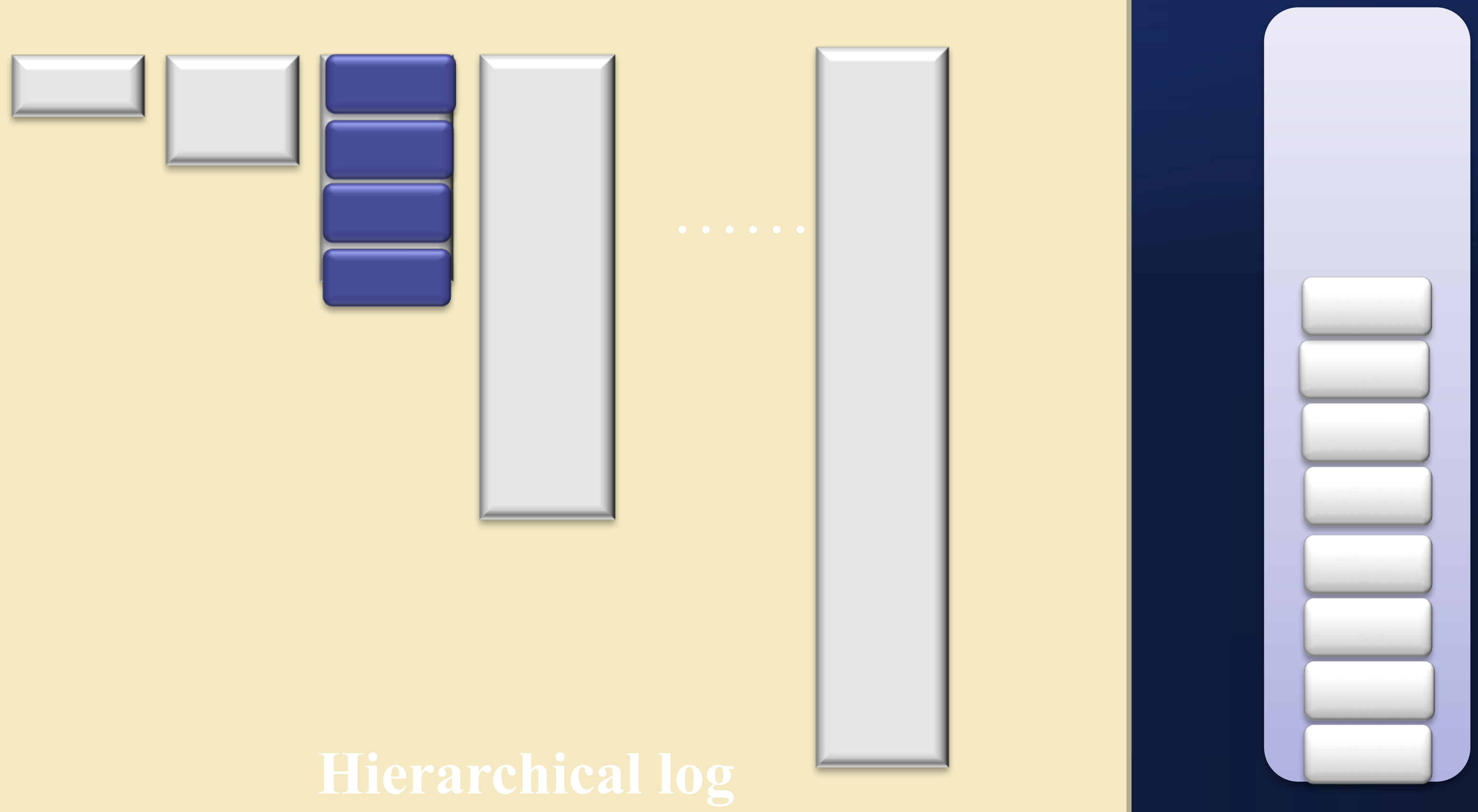
Update buffer



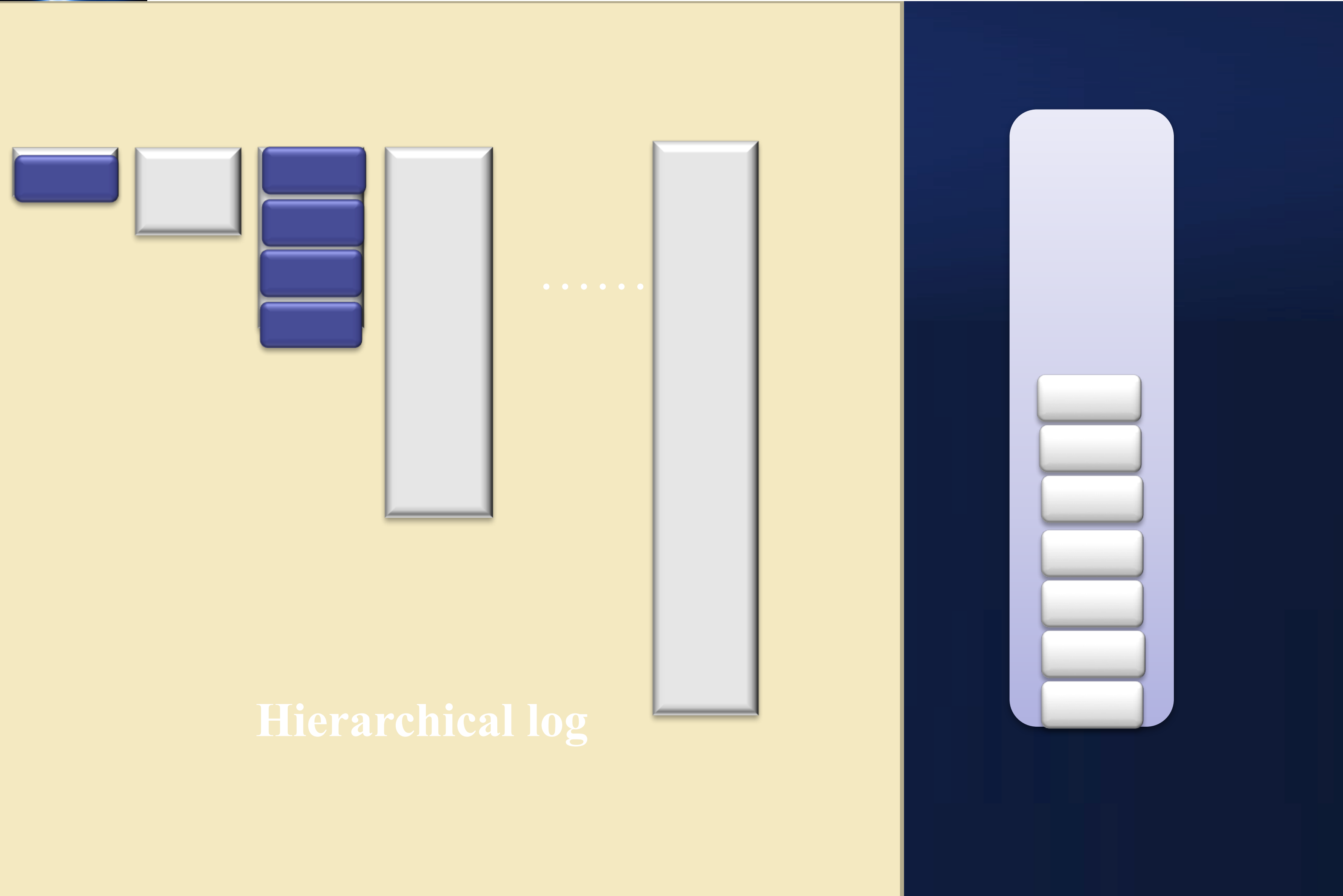
Update buffer



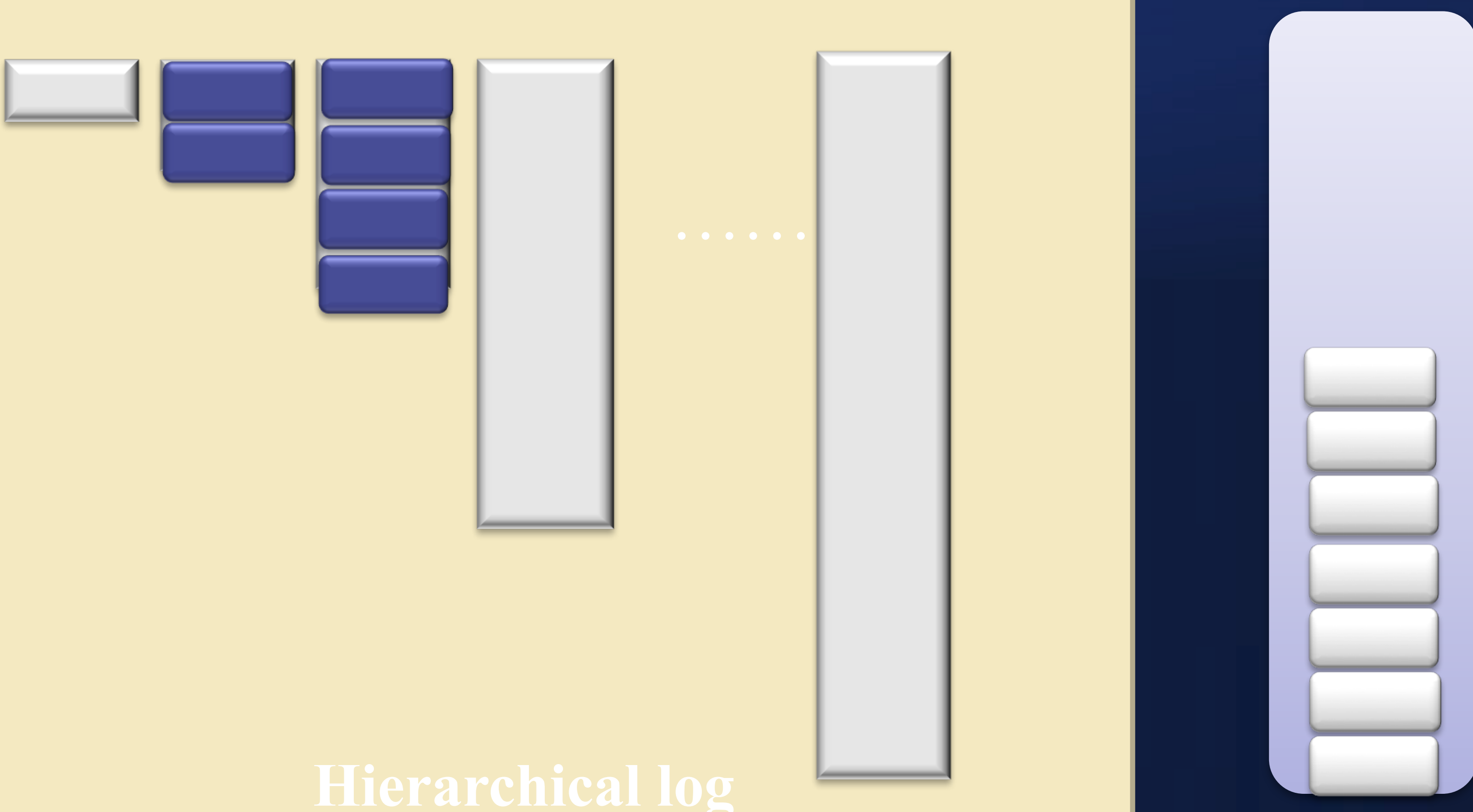
Update buffer



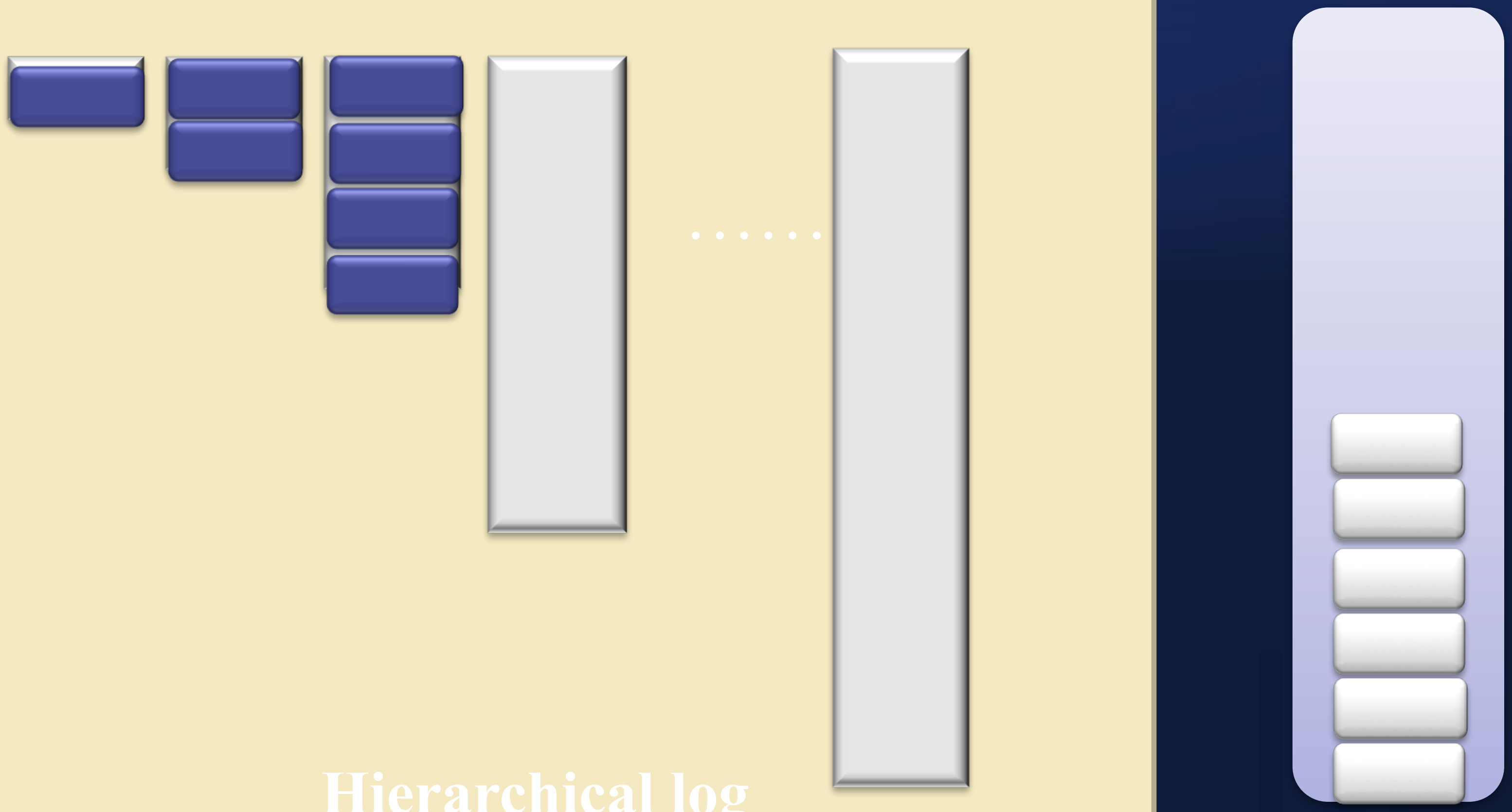
Update buffer



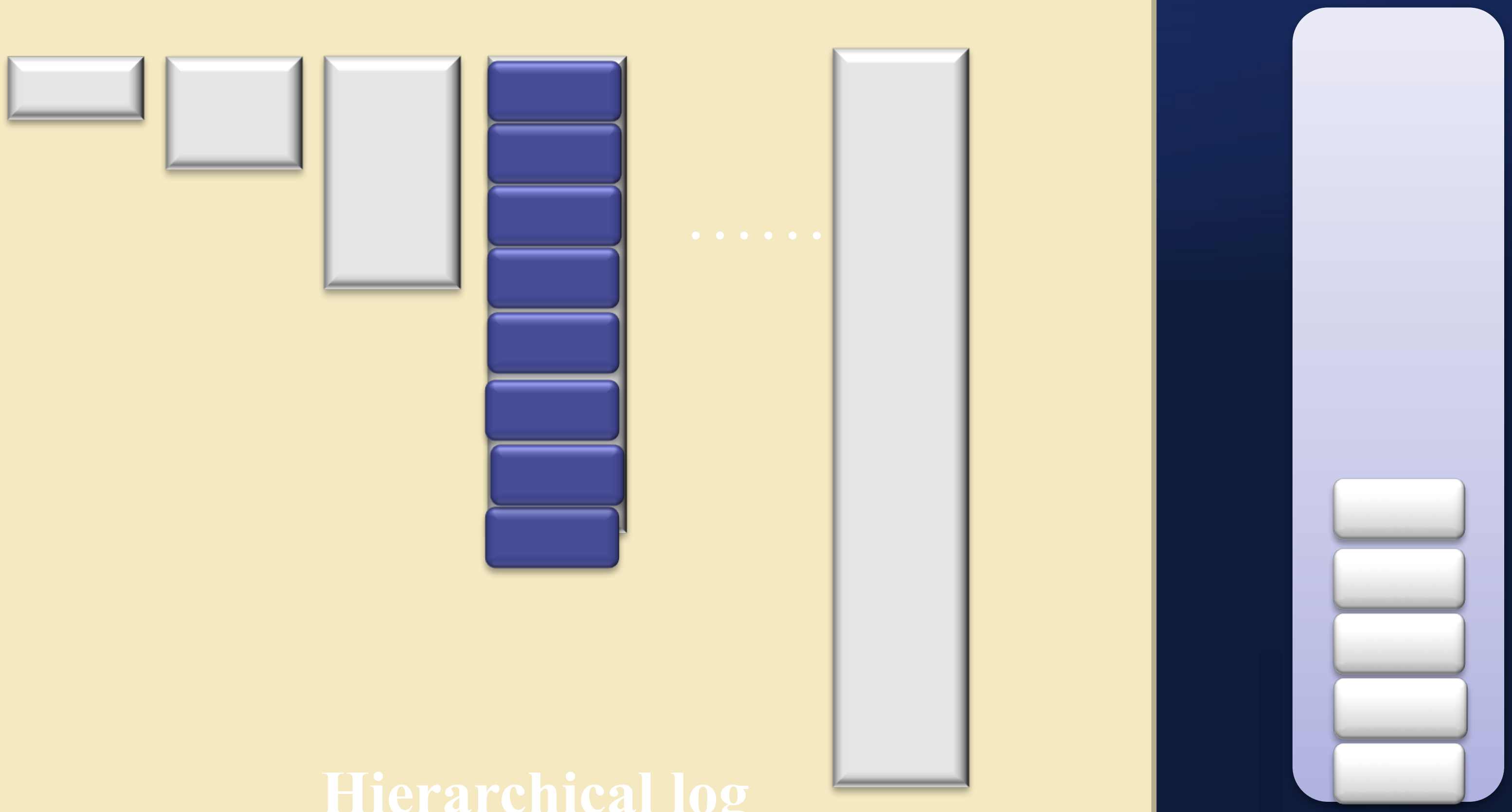
Update buffer



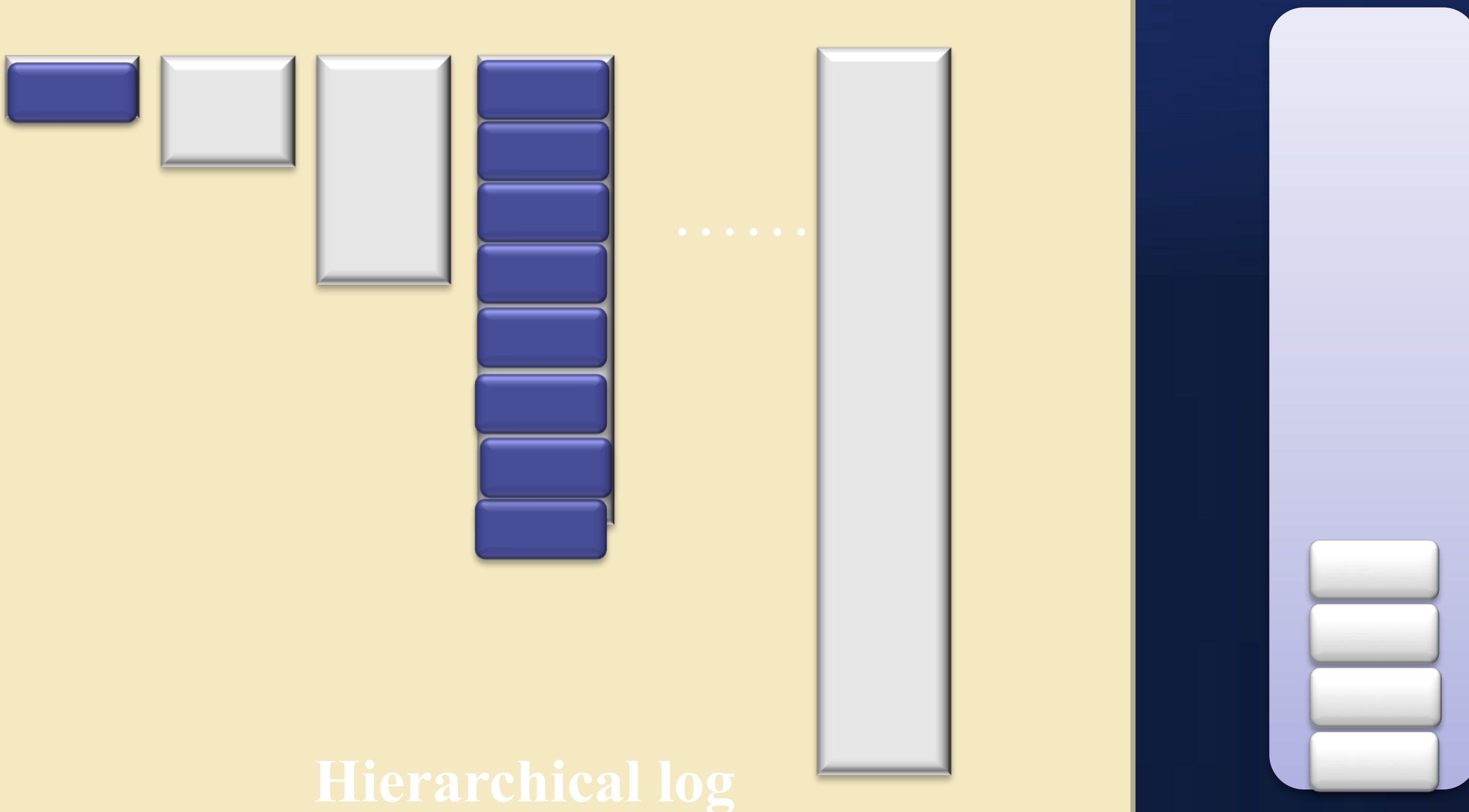
Update buffer



Update buffer



Update buffer



Update buffer