

Lethe: Secure Deletion by Addition

*Eugene Chou, Leo Conrad-Shah, Austen Barker
Andrew Quinn, Ethan L. Miller, Darrell D. E. Long
Center for Research in Systems and Storage
University of California, Santa Cruz*

Redesigning systems for data privacy

- Storage systems must provide data erasure
- Ideally, this is provided...
 - Within short timescales (timely)
 - With low overhead (efficient)
 - Without requirements of underlying storage medium (portable)
 - Without bespoke system solutions

Art. 17 GDPR
Right to erasure ('right to be forgotten')

1. The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay where one of the following grounds applies:

California Consumer Privacy Act (CCPA)

Home / Privacy / California Consumer Privacy Act (CCPA)

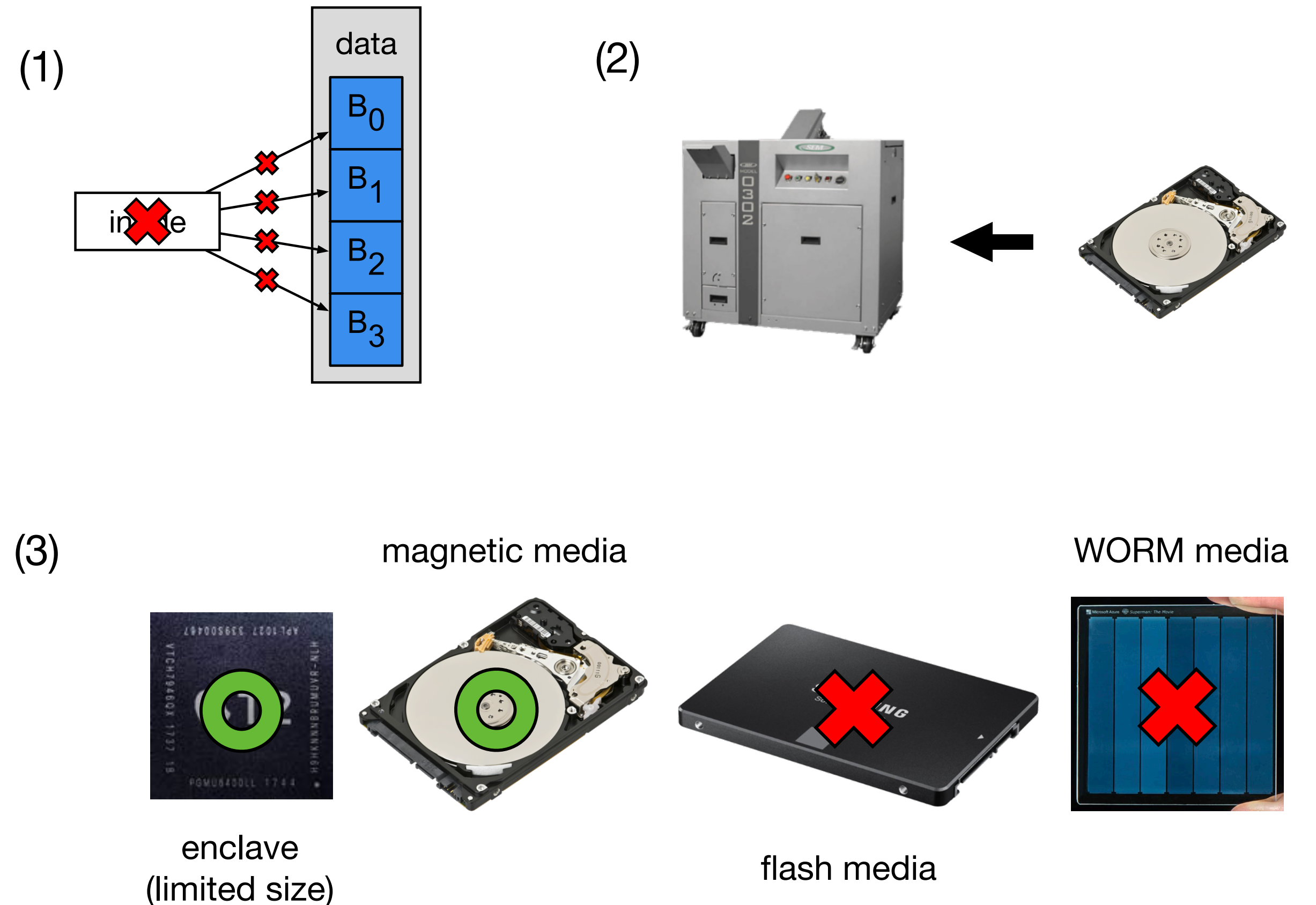
Updated on February 15, 2023

The [California Consumer Privacy Act of 2018 \(CCPA\)](#) gives consumers more control over the personal information that businesses collect about them and the [CCPA regulations](#) provide guidance on how to implement the law. This landmark law secures new privacy rights for California consumers, including:

- The [right to know](#) about the personal information a business collects about them and how it is used and shared;
- The [right to delete](#) personal information collected from them (with some exceptions);
- The [right to opt-out](#) of the sale or sharing of their personal information; and
- The [right to non-discrimination](#) for exercising their CCPA rights.

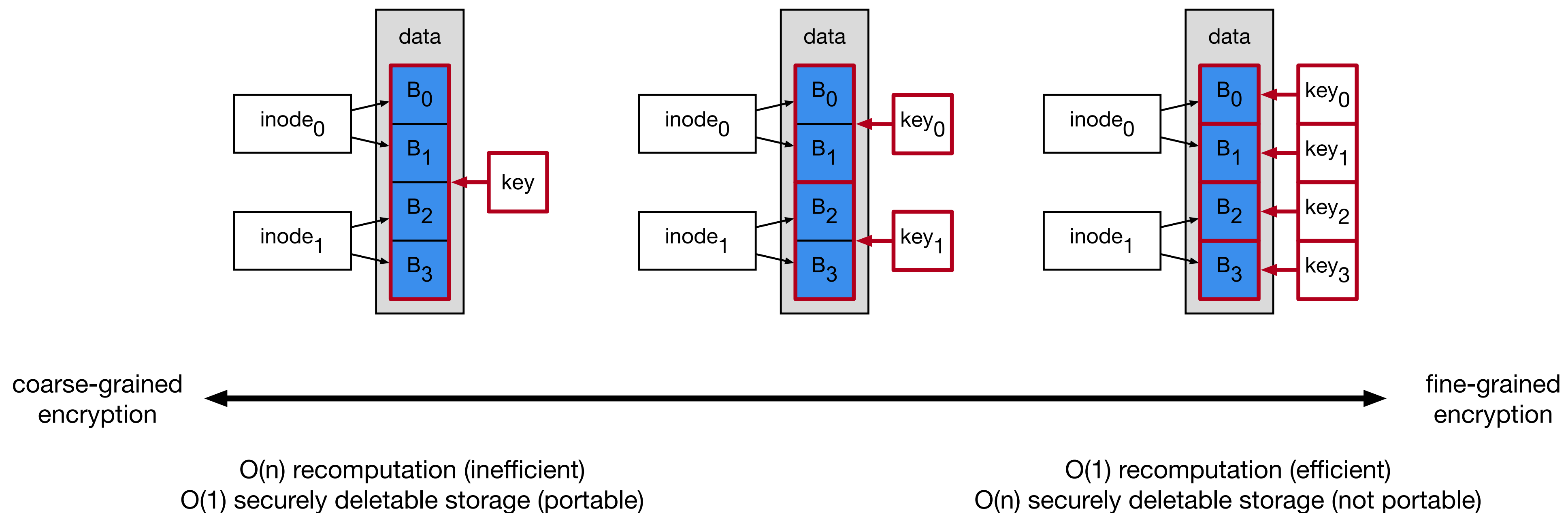
Existing approaches

1. `rm/unlink` doesn't securely delete data
2. Physical destruction is coarse-grained and wasteful (inefficient)
3. Overwrite erasure comes with impractical constraints
 - Requires in-place overwrite (not portable)
 - Often requires multiple passes
 - Degrades device durability



Cryptographic erasure

- Securely delete data by encrypting it and erasing the key
- Introduces the *key management problem*:
 - How do we balance the amount of securely deletable storage and computation?

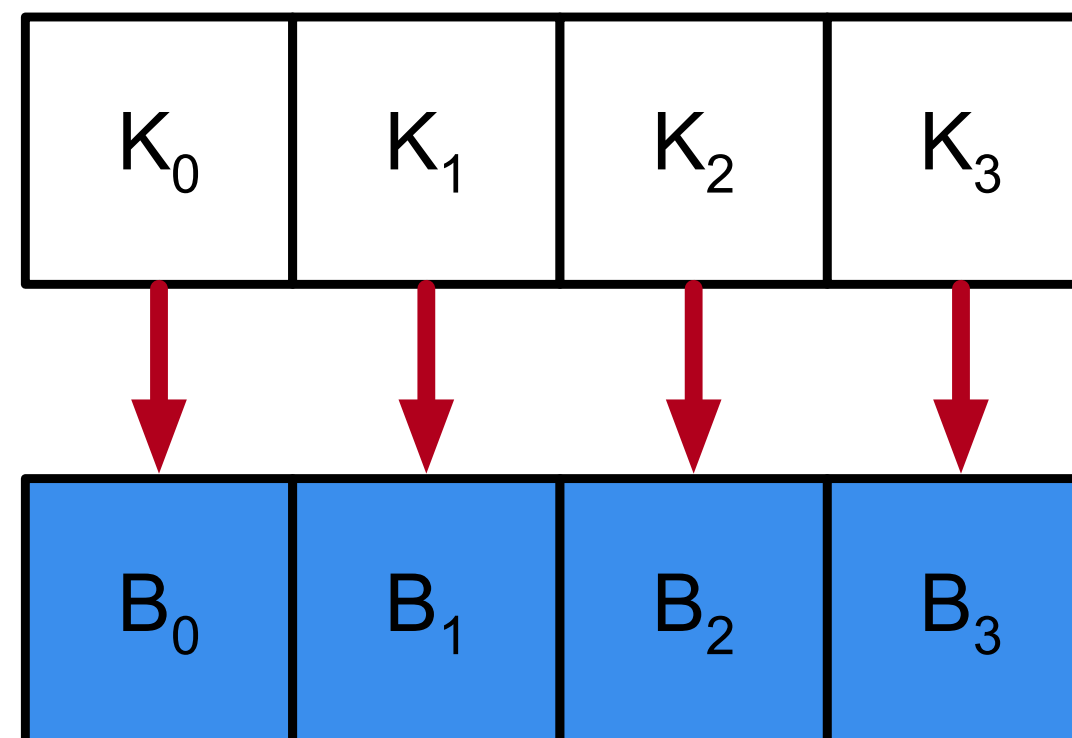


- Provides timely, fine-grained secure deletion on arbitrary storage media
 - Only requires a small, fixed amount of securely deletable storage
- Efficiently solves the key management problem
- All data is written as *append-only* using copy-on-write
 - Secure deletion isn't removing the data that is no longer wanted
 - It is adding data that only allows access to what remains valid
 - Portable since there is no requirement for in-place overwrite

The rest of the talk

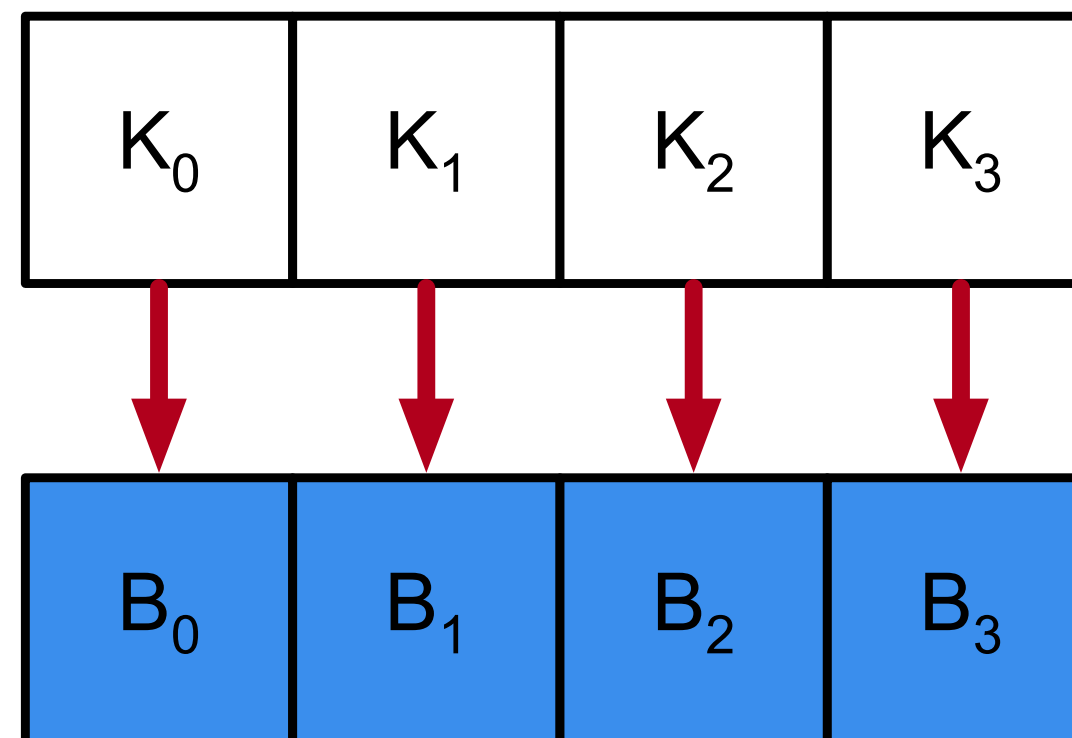
- Solution to the key management problem
- The design of Lethe
- Evaluation of the Lethe prototype

Solution to the key management problem

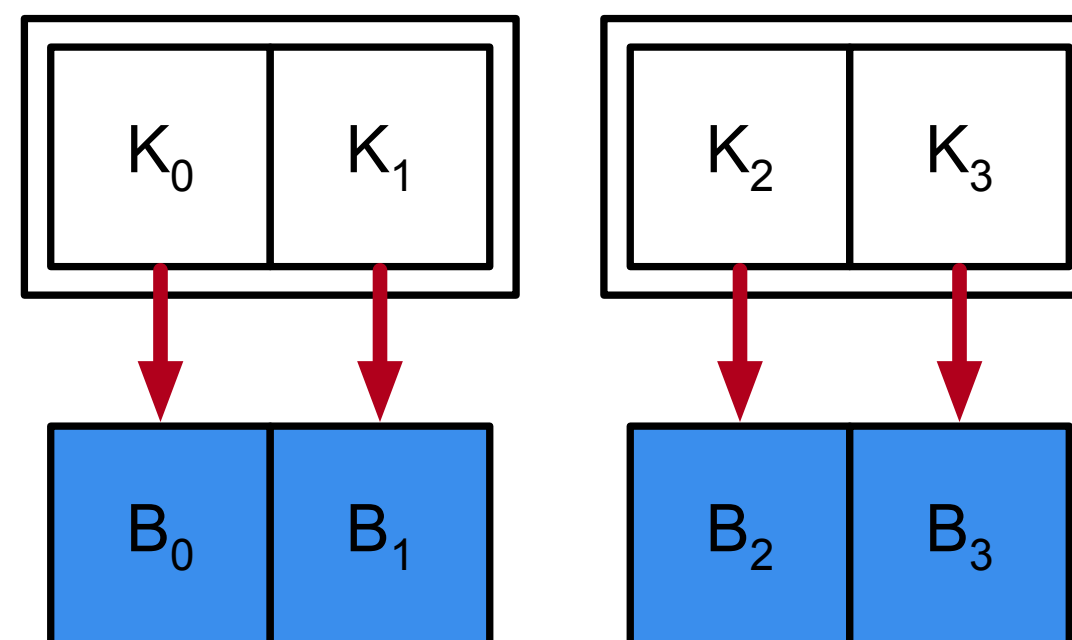


Solution to the key management problem

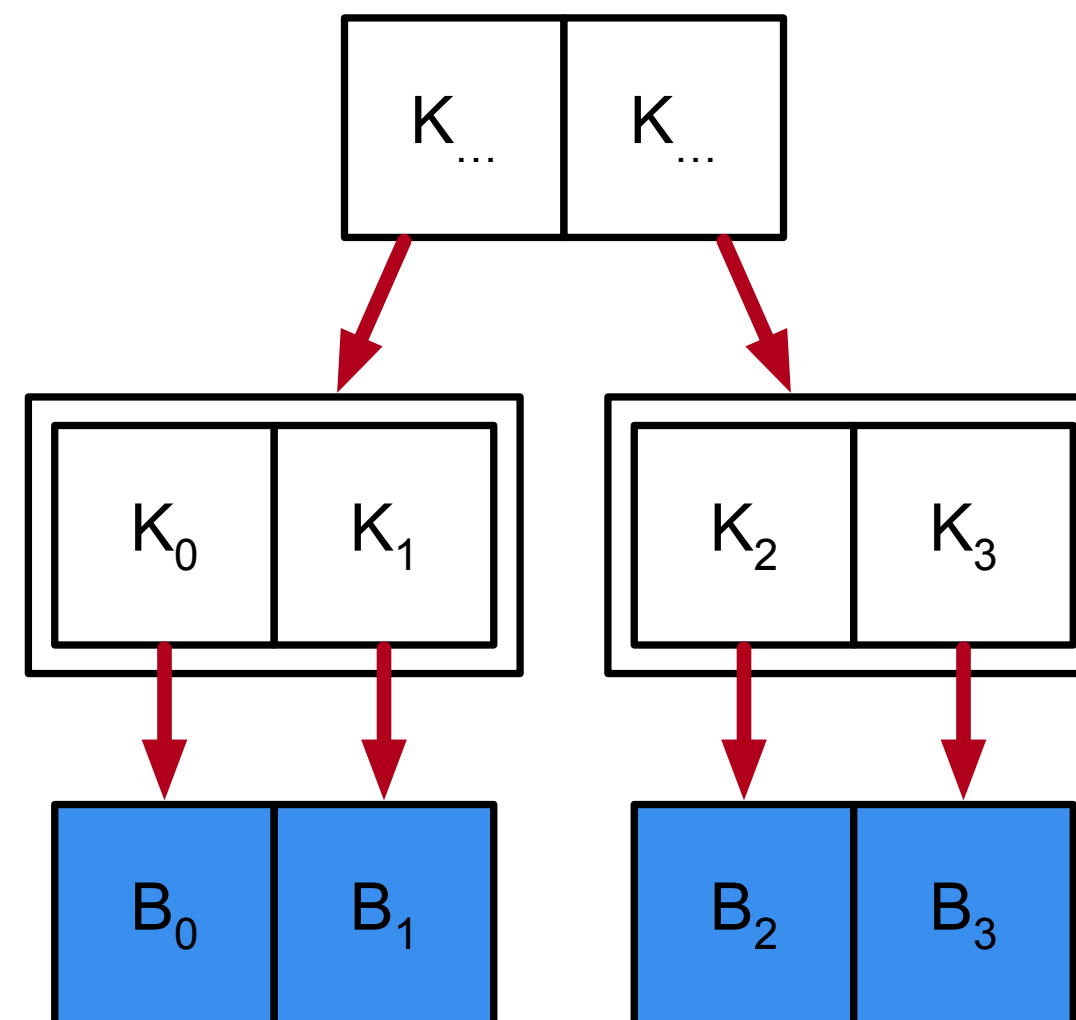
What if we applied cryptographic erasure recursively?



Solution to the key management problem

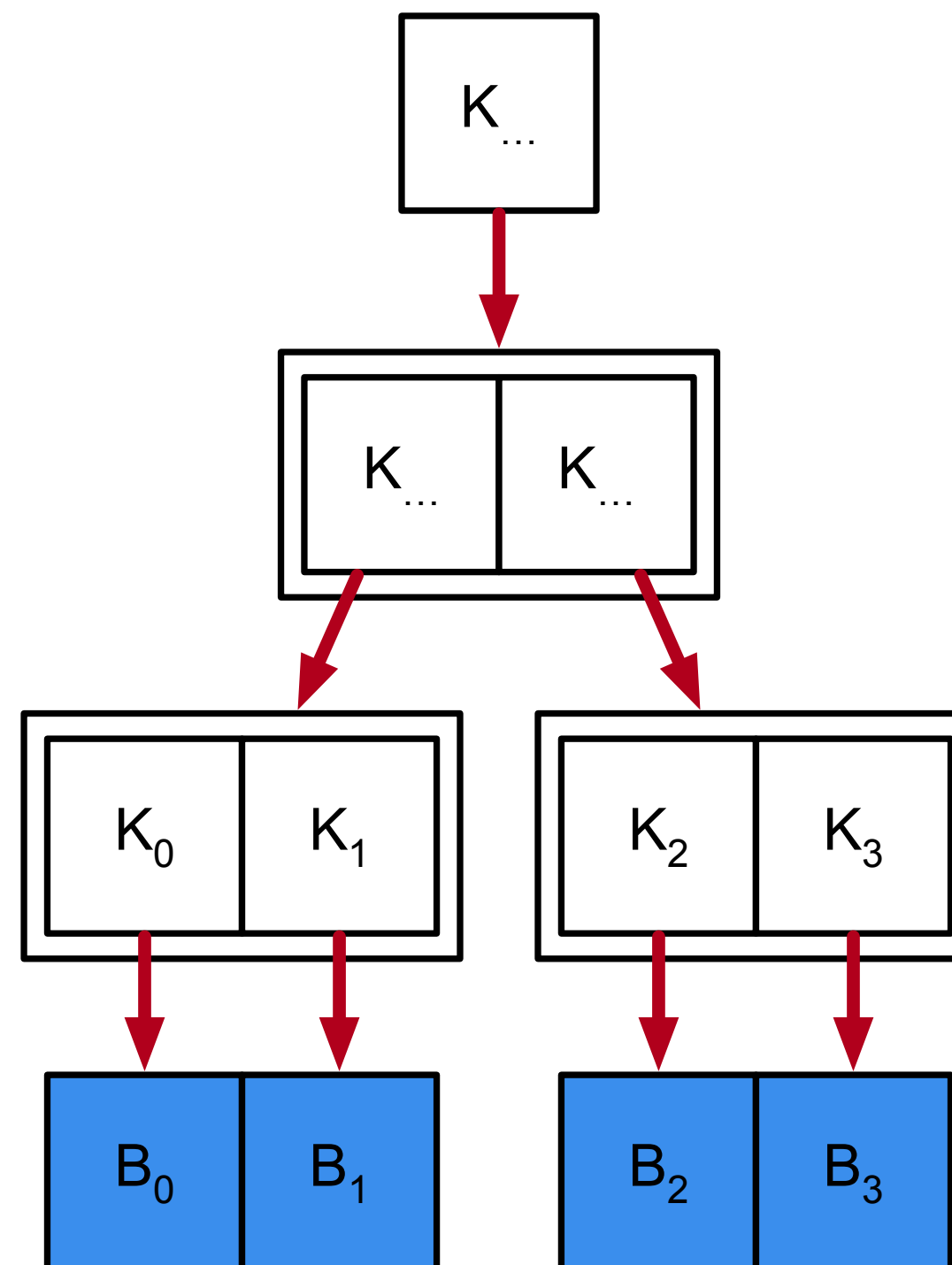


Solution to the key management problem



Solution to the key management problem

- Hierarchy reduces the $O(n)$ securely deletable storage down to $O(1)$: *a single key*
 - Portable since only the root key needs to be erased; key blocks are protected recursively

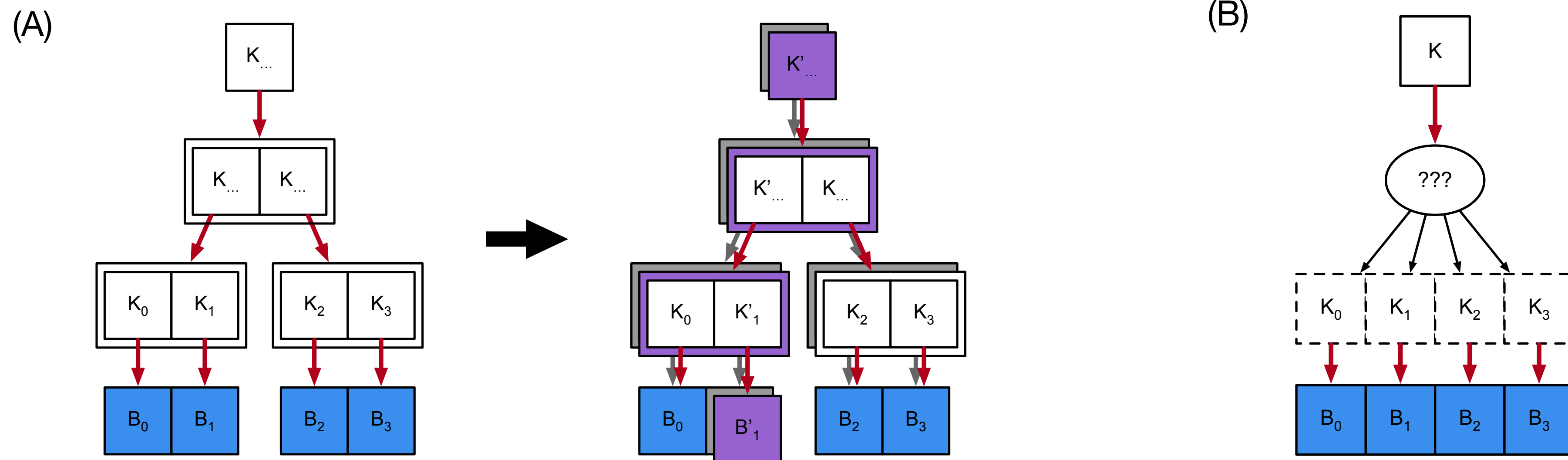


Solution to the key management problem

A. Key revocation requires $O(\log(n))$ re-encryption

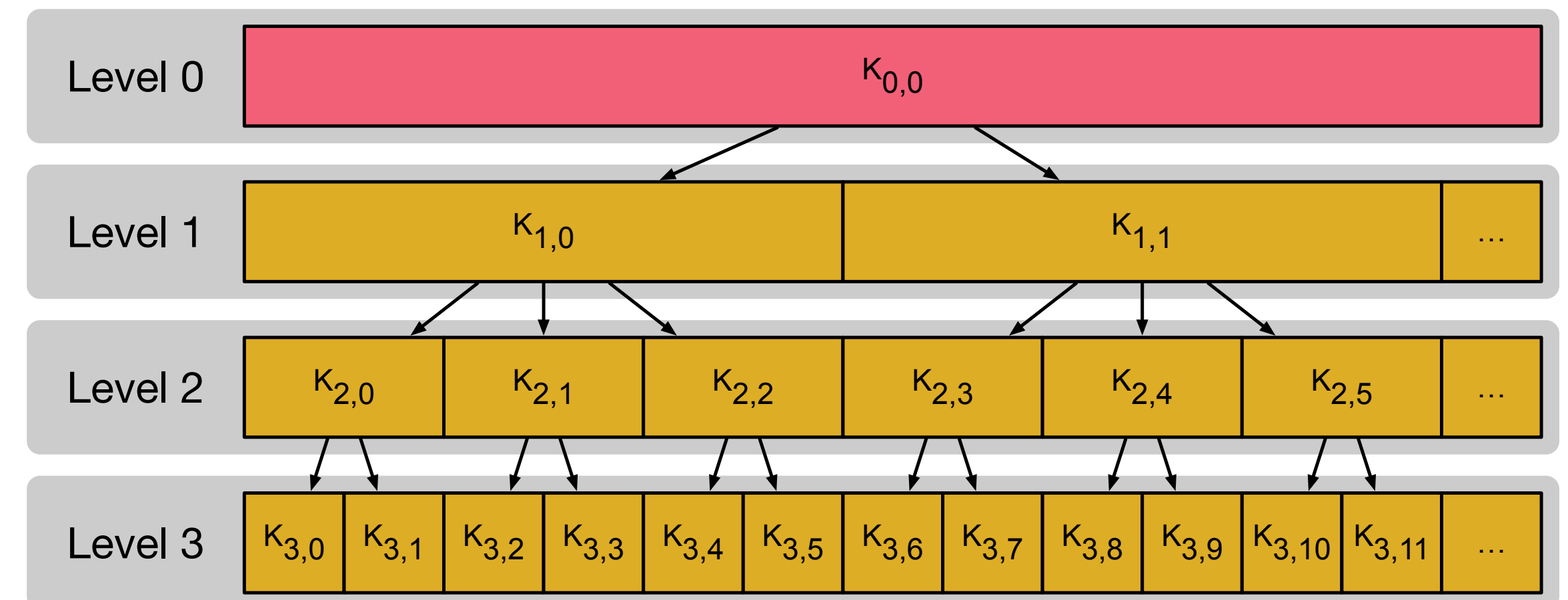
- *Roll forward* the keys to still-valid data, and replace the revoked keys
 - Revoked keys are securely deleted once old root is erased

B. We want to reduce the cost of the re-encryption by efficiently managing the tree of keys



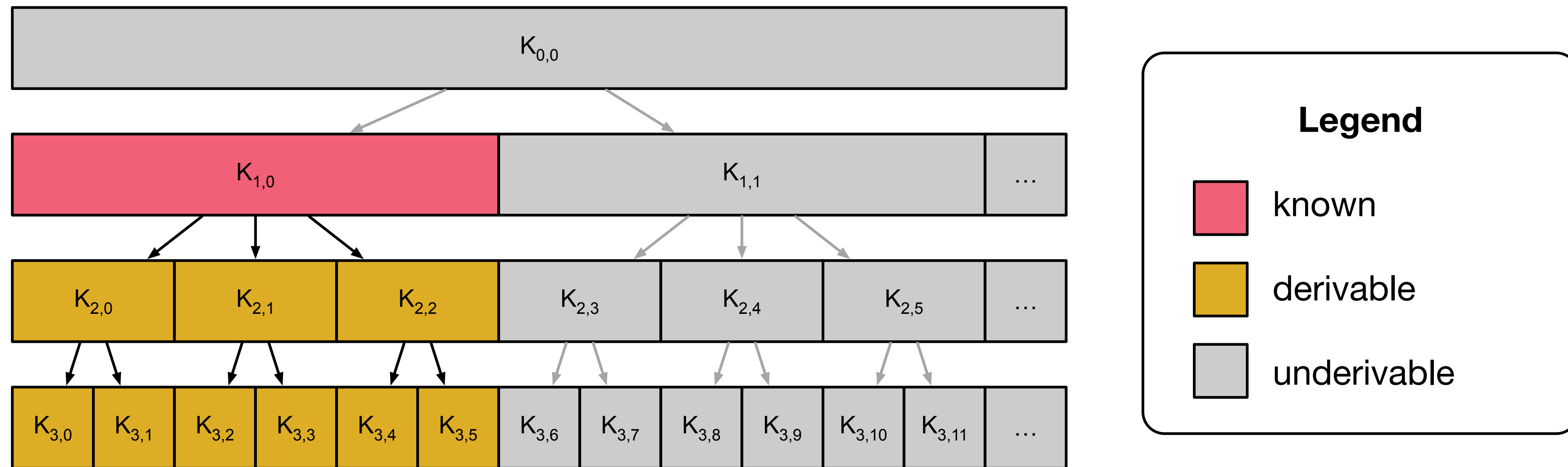
Keyed Hash Tree

- Used by Horus[1] for efficient key derivation
 - But not key revocation
- A node is a key identified by level and offset
- Descendant keys computed recursively
 - Parent m , child n , cryptographic hash H
 - $n_{\text{key}} = H(m_{\text{key}} || n_{\text{level}} || n_{\text{offset}})$
- Topology defined by L1 subtree topology
 - Can have an arbitrary number of L1 subtrees
 - Effectively infinite keys from a single root



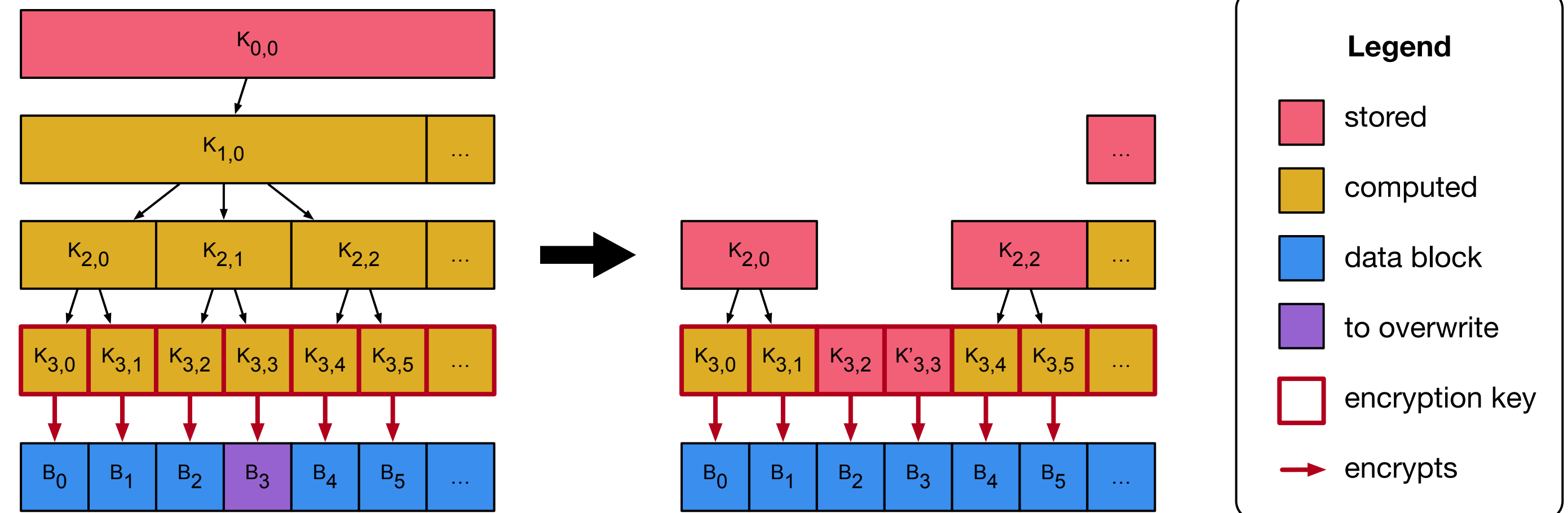
One-way relationships

- *Computationally intractable* to derive the value of an ancestor or sibling
 - Doing so would require “inverting” a cryptographic hash function



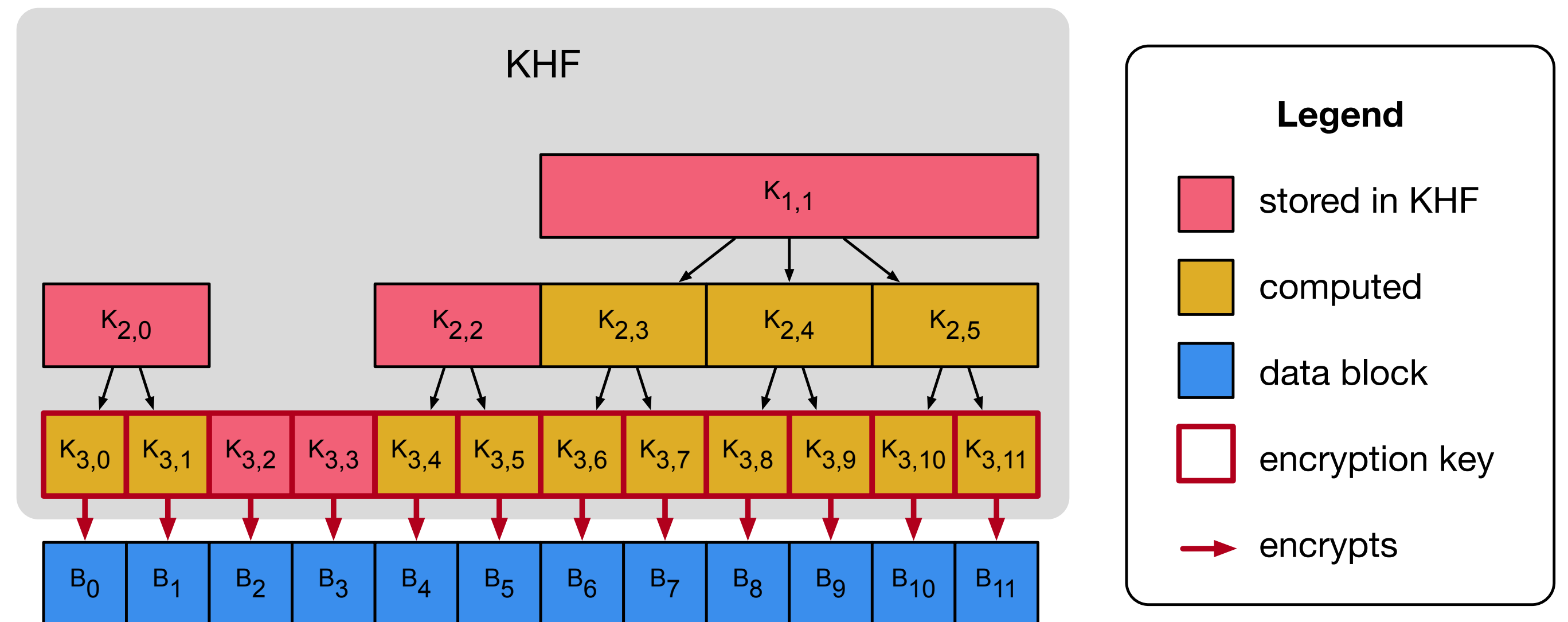
Fragmentation through key revocation

- Occurs when data is overwritten or deleted
- **Must keep roots that:**
 - Provide still-valid key
 - Provide replacement keys
- **Cannot keep roots that:**
 - Cover revoked keys
- Causes *fragmentation*



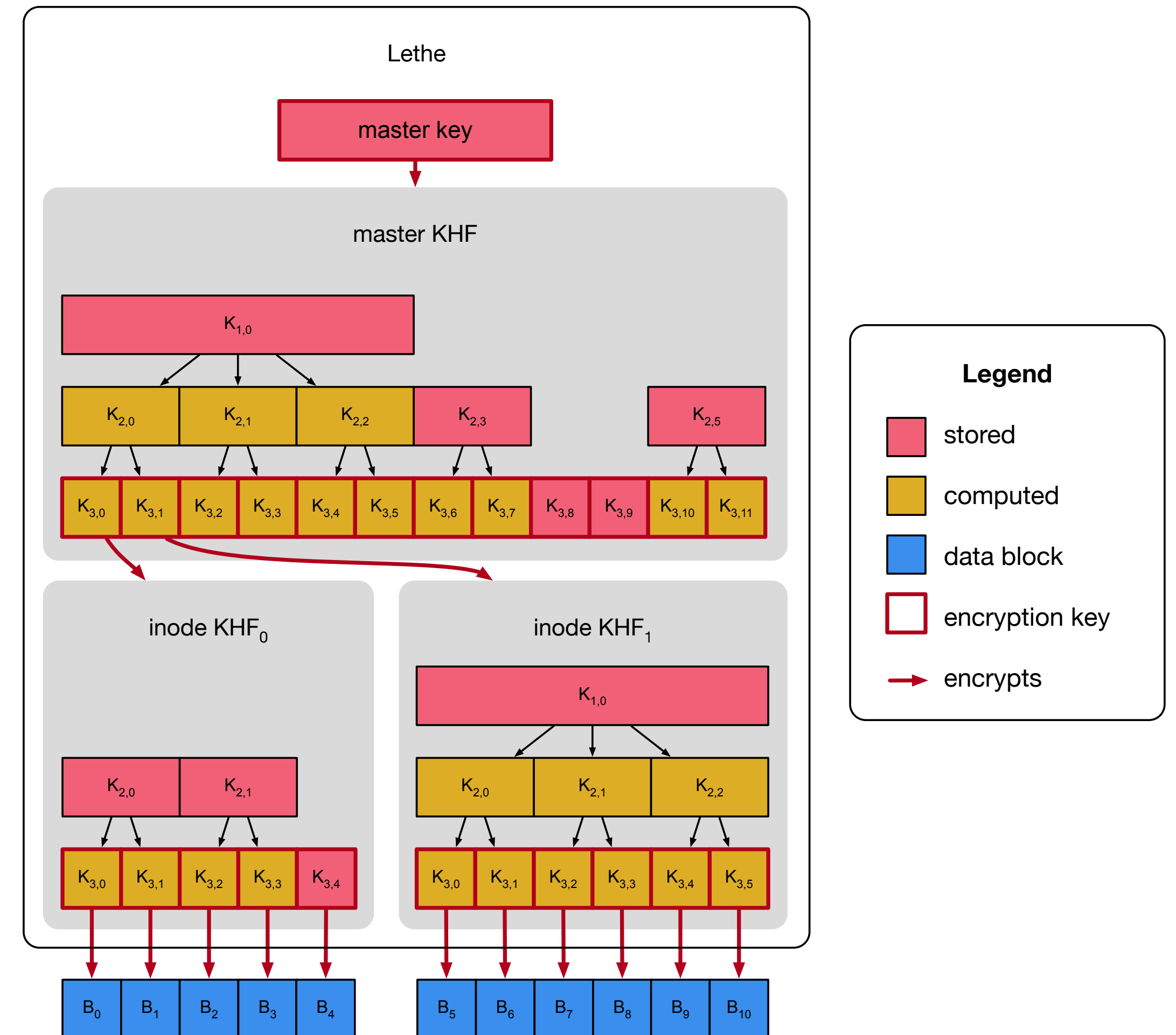
Keyed Hash Forest

- Describes a forest of KHTs
- Designed for efficient key derivation and key revocation
- Fragments as deletes occurs over time
- Degenerate KHF: each key is covered by its own root



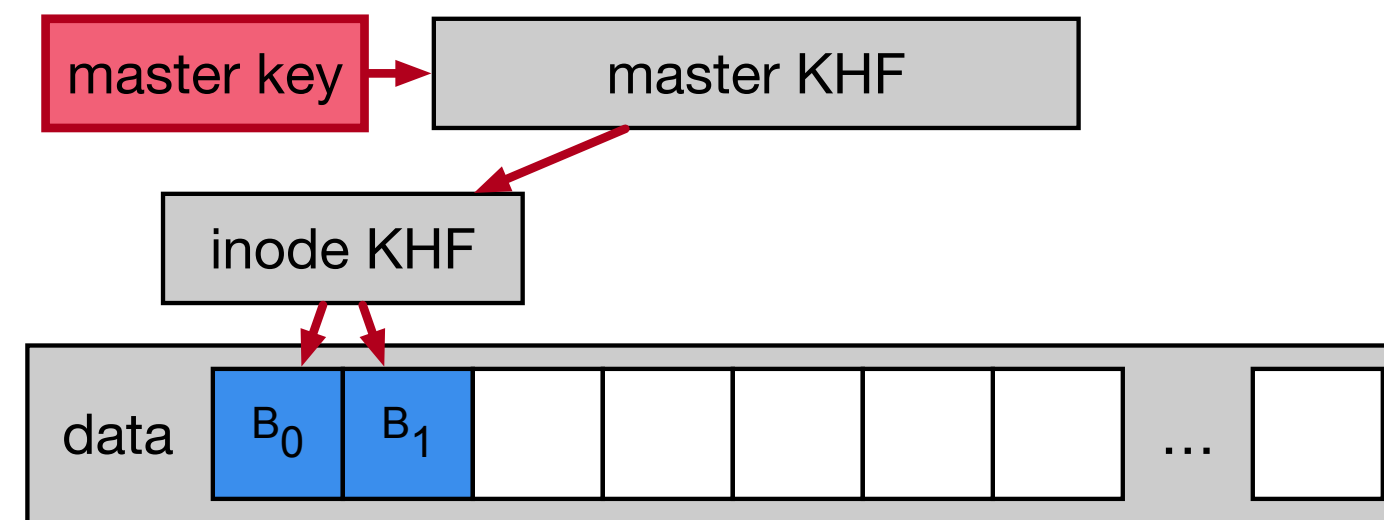
Lethe: Combining hierarchy and KHF

- Use more hierarchy to further reduce computation
 - Also helps in exploiting *locality*
- Lethe uses a two-level KHF hierarchy
 - Can use more levels if desired
- Inode KHF protects an inode's data blocks
- Master KHF protects inode KHF
- A master key protects the master KHF
 - Erasing the master key securely deletes the data it protects









Secure deletion by addition

time t

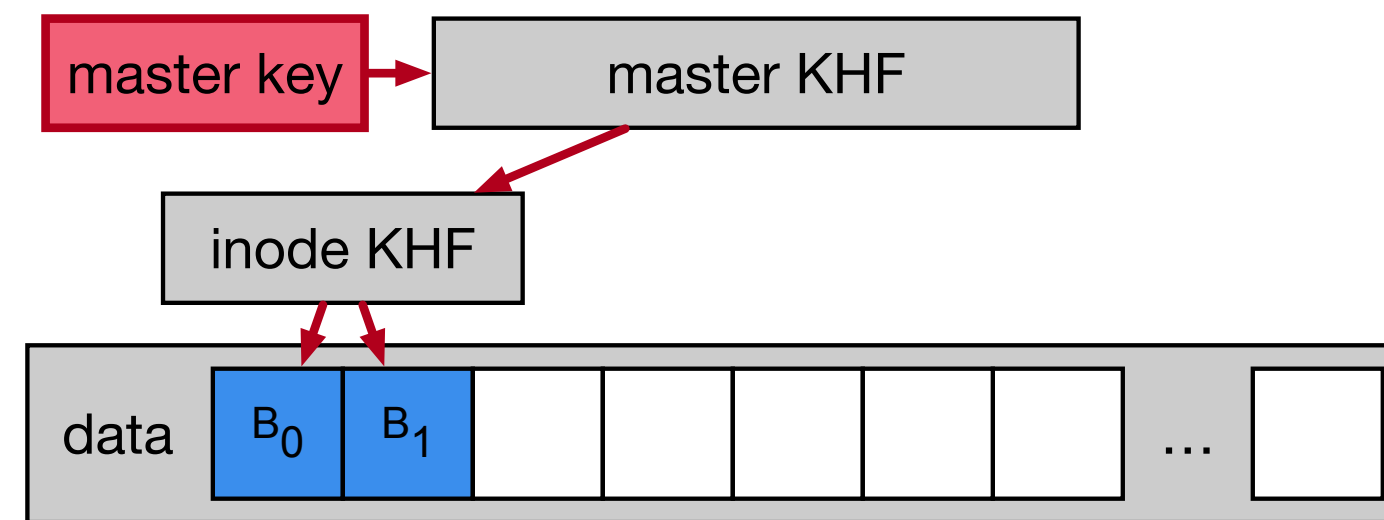


Legend

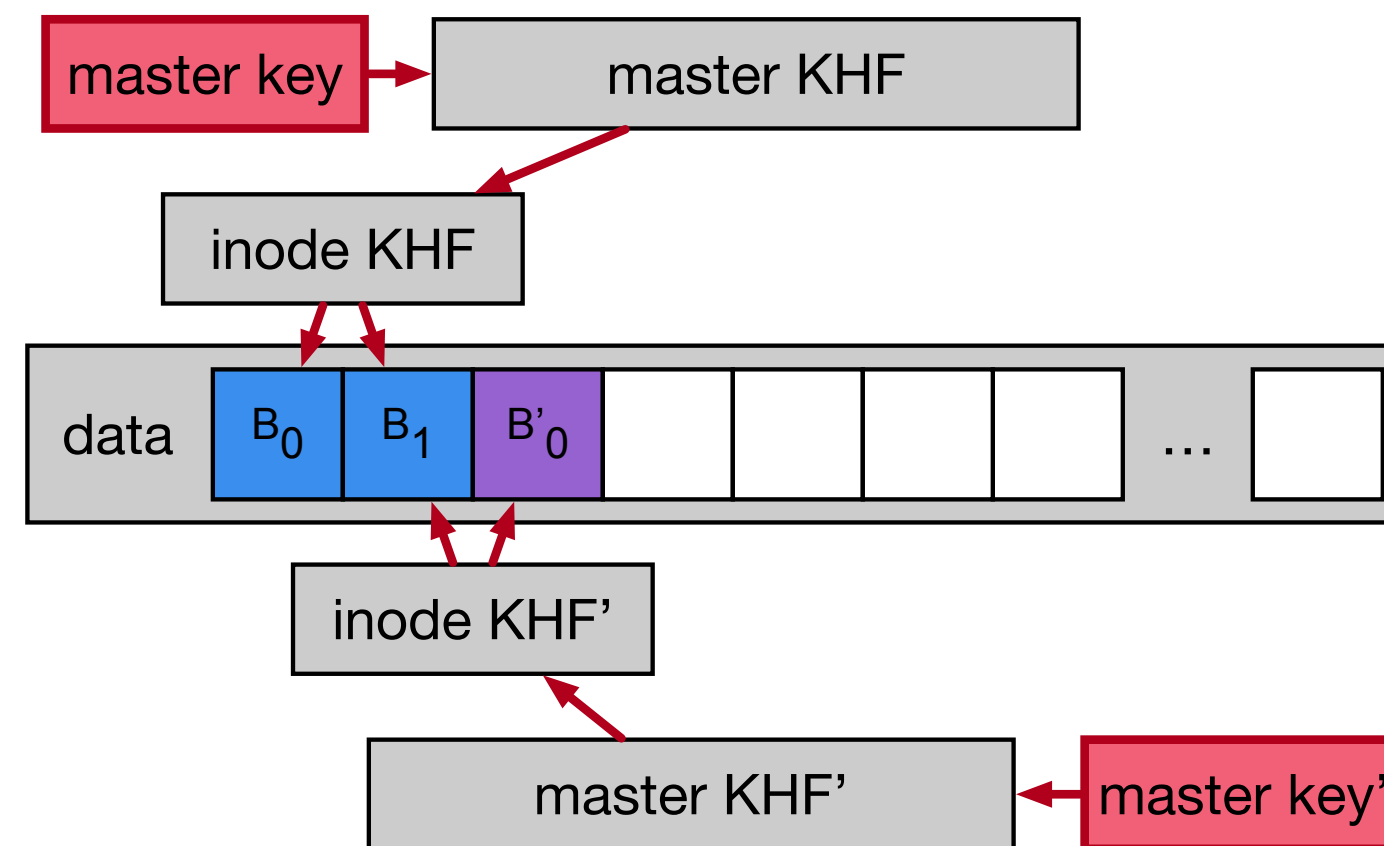
-  securely deletable
-  free block
-  used block
-  encryption key
-  encrypts
-  inaccessible

Secure deletion by addition


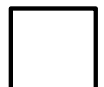




time t



time t'

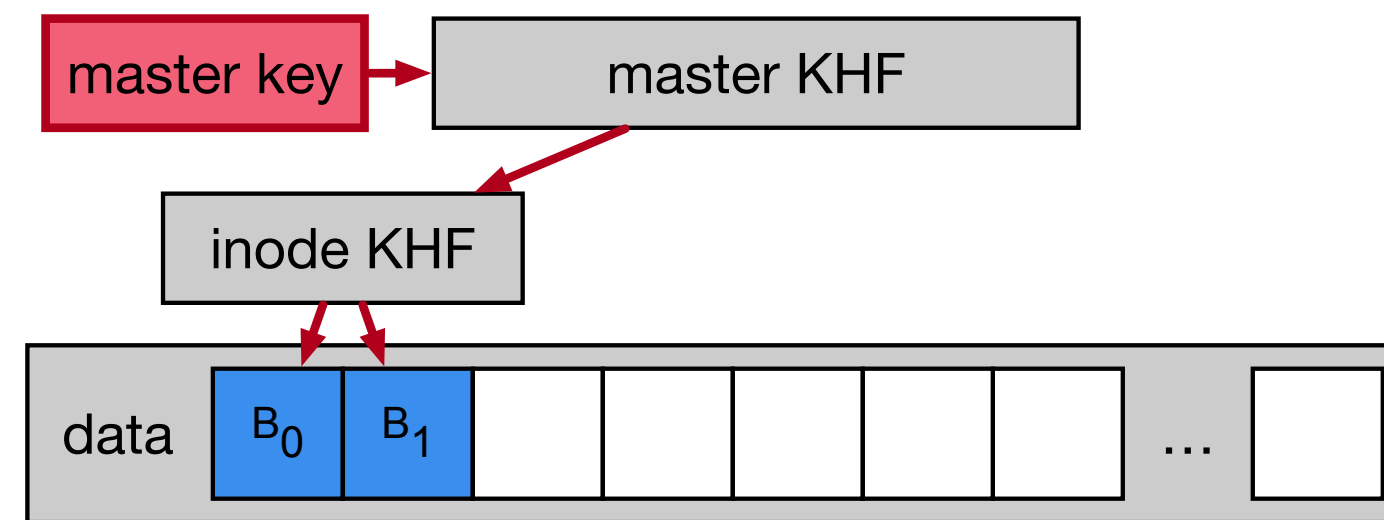


Legend

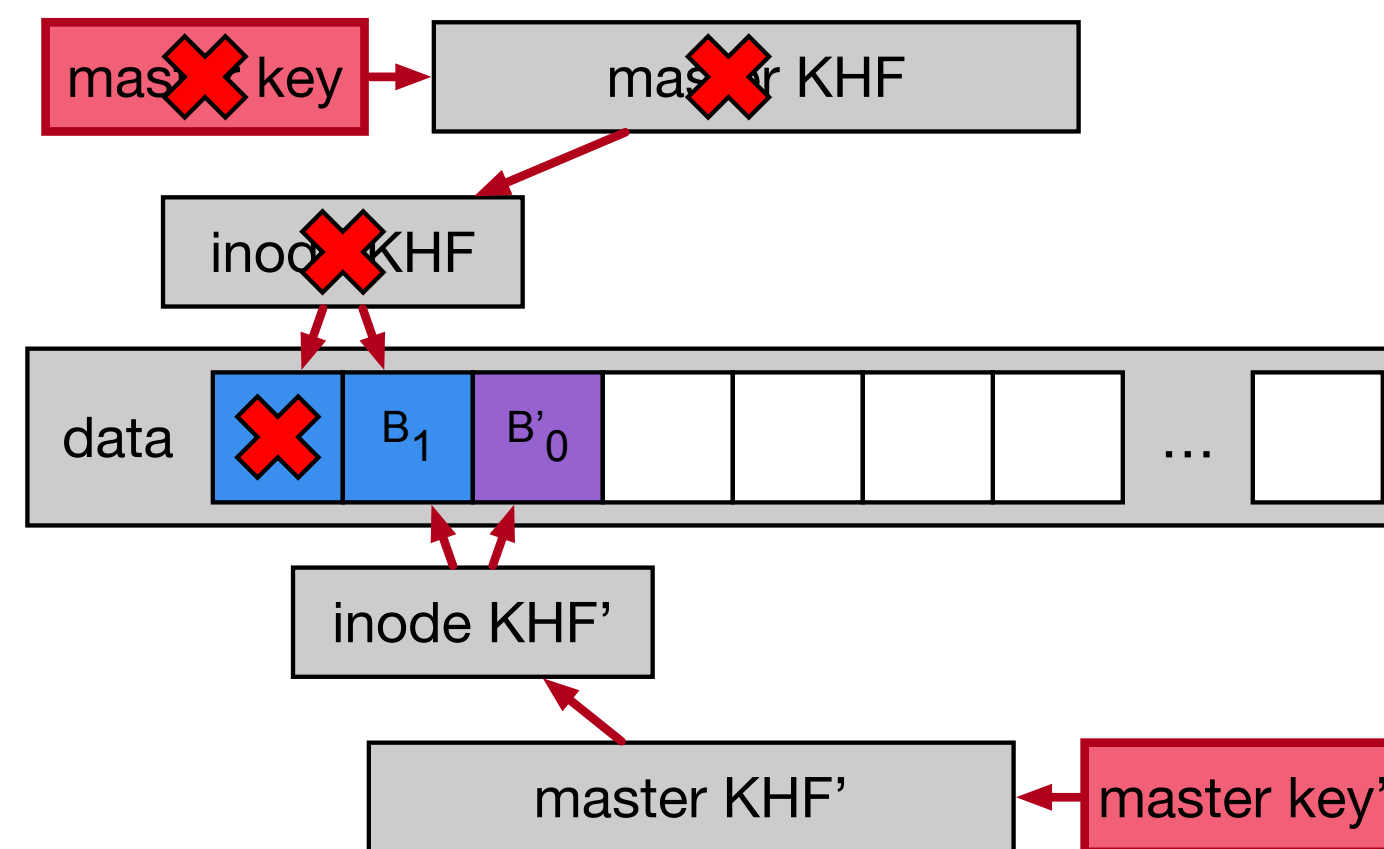
-  securely deletable
-  free block
-  used block
-  encryption key
-  encrypts
-  inaccessible

Secure deletion by addition







time t



time t'

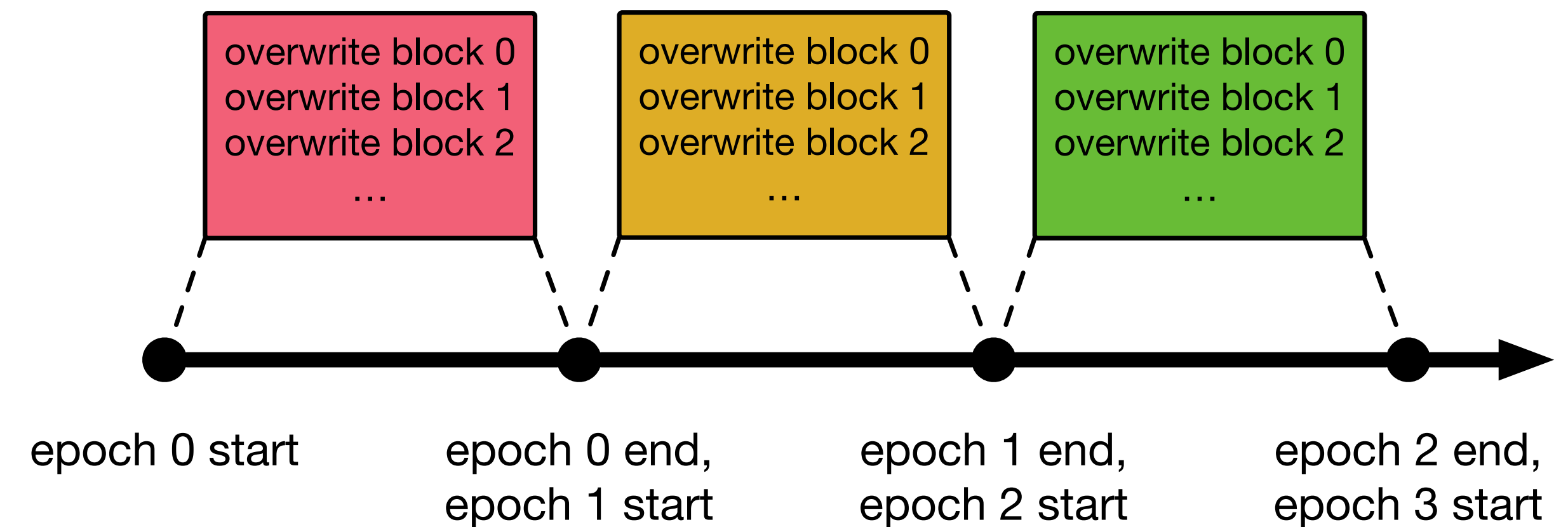


Legend

-  securely deletable
-  free block
-  used block
-  encryption key
-  encrypts
-  inaccessible

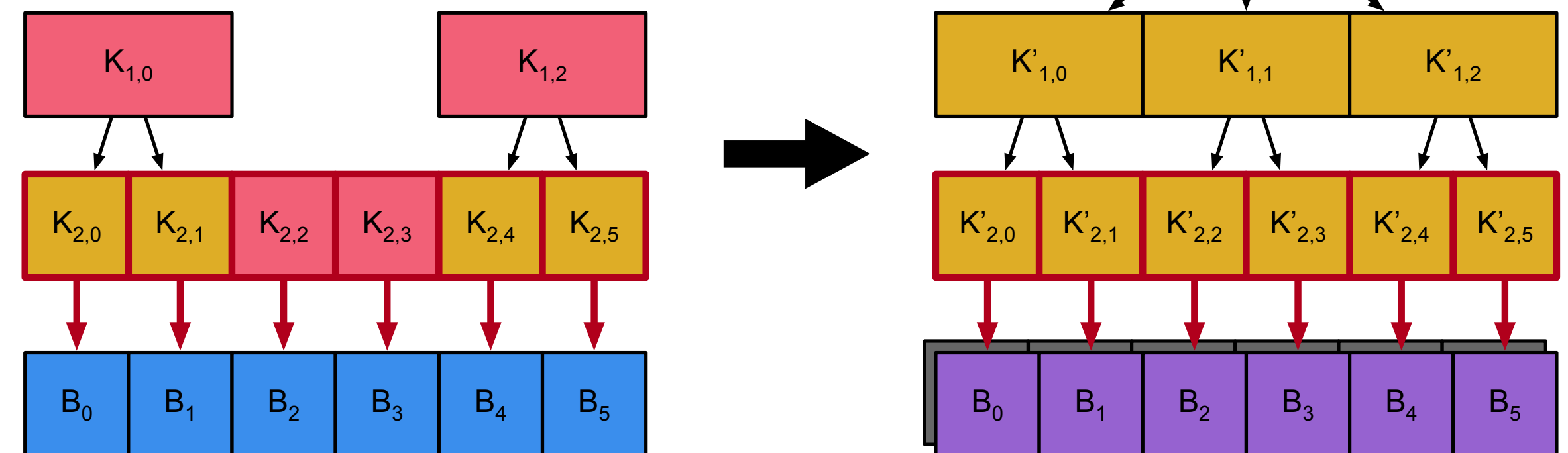
Epochs

- Intervals of time where write operations are batched together
 - Amortizes cost of updating hierarchy of KHF's on each write or delete
- Can be tuned for either...
 - Prompt secure delete: short epochs
 - Performance: longer epochs
- Can also be triggered manually



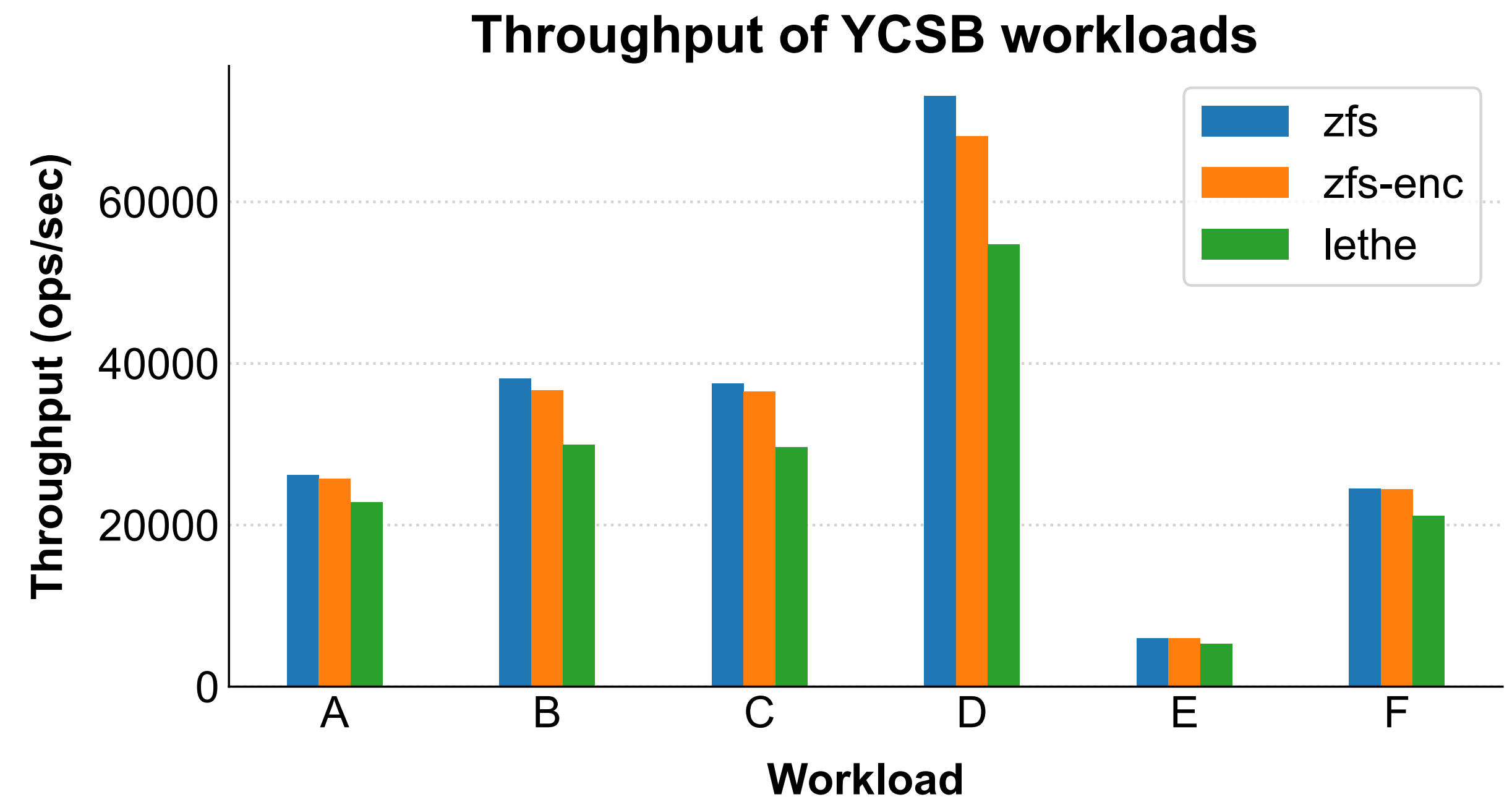
Consolidation

- KHF's can get quite large as fragmentation occurs over time
 - Persisting an extremely fragmented KHF is expensive
- Pay the penalty of re-encryption to defragment a KHF
 - Helpful for the master KHF, which needs to be re-written every epoch
- Not needed for security, but can improve performance in the long run



Evaluation

- Integrated Lethe into ZFS
- Compared against:
 1. Baseline ZFS (zfs)
 2. ZFS with native encryption (zfs-enc)
 - Per-block encryption
 - Doesn't support secure deletion
 3. ZFS with Lethe (lethe)
- YCSB (1M operations, 1M records, Zipfian)
 - Intel NUC (i5, 4 cores, 1.6GHz, 32GB RAM)
 - Samsung 970 EVO (500GB)



zfs to zfs-enc: 2.6% decrease in throughput
zfs to lethe: 17.63% decrease in throughput

Conclusion

- Lethe is the first system to provide secure deletion...
 - Within short timescales (timely)
 - With low overhead (efficient)
 - Without requirements of underlying storage medium (portable)
 - As long as there is access to a small, fixed amount of securely deletable storage
- Lethe reframes the art of secure deletion
 - Not by removing the data that is no longer wanted,
 - But by adding data that only allows access to what remains valid

Thanks for listening!

Questions?

email: euchou@ucsc.edu