CEASER

→ Sharing LLC is bad
→ Adversary can infer access patterns by doing evictions by doing cache conflicts.

CEASE:

Normal access address → Encryption engine → Encrypted address → Addressed that is different from the initial address

CEASER ⇒ same as CEASE but encryption key changes periodically.

→ Do all the above with less overhead

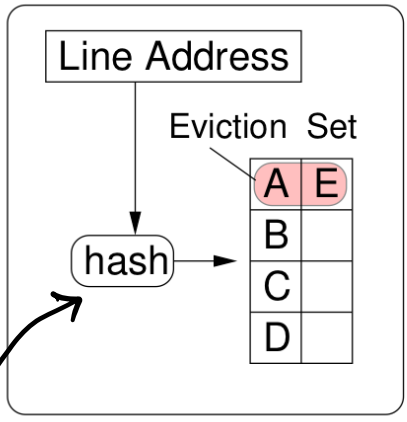
Solutions

Preservation
↓
The attacker is not allowed to evict victim
↓
Inefficient use of LLC

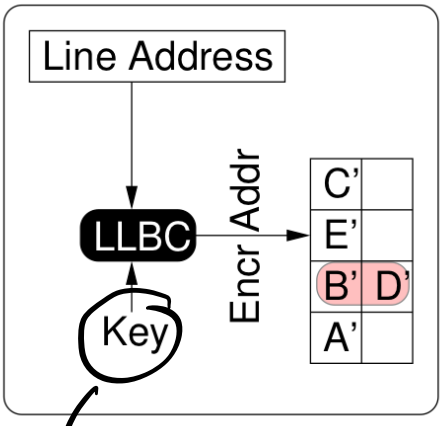
Randomization
↓
Randomised the address
↓
Should maintain large table → Impractical

Address space → Encrypted Address Space (EAS)
→ Physical Address Space (PAS)

Traditional



CEASE



PAS

Static as the mapping is dependent on the hash

EAS

Dynamic as the mapping is dependant on the key.

CEASE ⇒ Cache operated on Encrypted Address Space

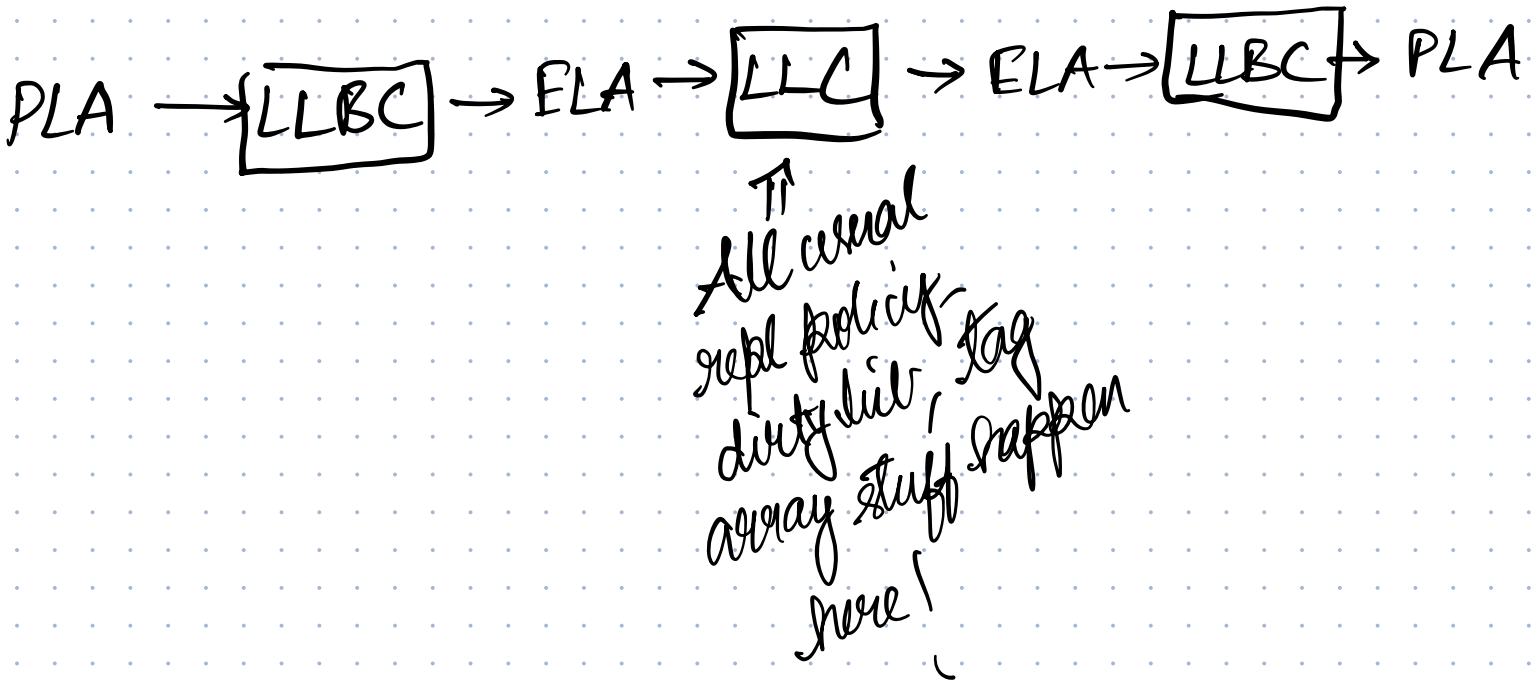
CEASER ⇒ Cache operated on Encrypted Address Space with Remapping

LLBC ⇒ Low Latency Block Cipher

LLBC ⇒ 4 stage Feistel Network ⇒ Encrypt / Decrypt in 2 cycles

Motive : Prevent Prime+Probe attack

Avalanche effect ⇒ change input slightly ⇒ change output significantly



LLBC:

→ We need a Block Cipher that operates with low latency. Less cycles!

→ Adversary has no access to the plaintext - ciphertext pair.

⇓

Because no one ELA access

⇓

So small width block is enough

Drawback of CEASE:

→ The key is generated by PRNG at power up
→ But the key remains constant for the timespan its powered up
→ The attacker would try all combinations to form an eviction set.

Solution : CEASER
↳ Changes keys overtime and remaps existing blocks
Types → Bulk remap
→ Gradual remap

Bulk remap:
→ At end of each epoch
RLA —oldkey→ PLA —newkey→ ELA
→ Impractical as all lines must be remapped simultaneously

Gradual Remap:
→ Set-Relocation Pointer (SPtr) : Which set to remap
→ Access-Counter (ACtr) : Decides when to trigger next re-map. W.R access!

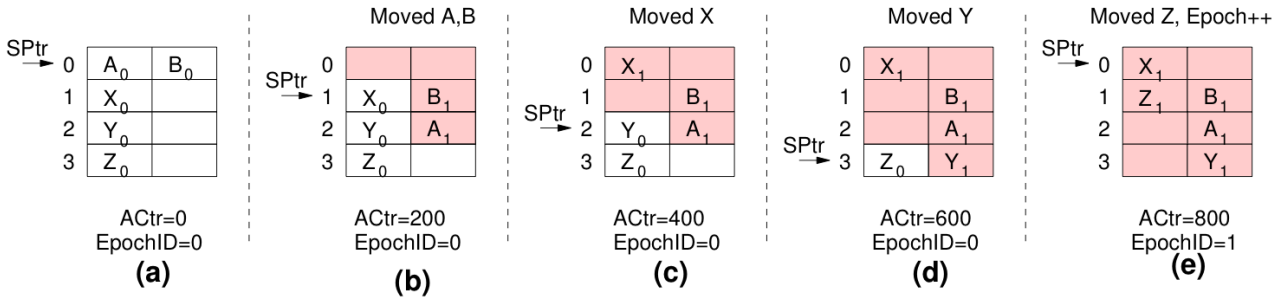
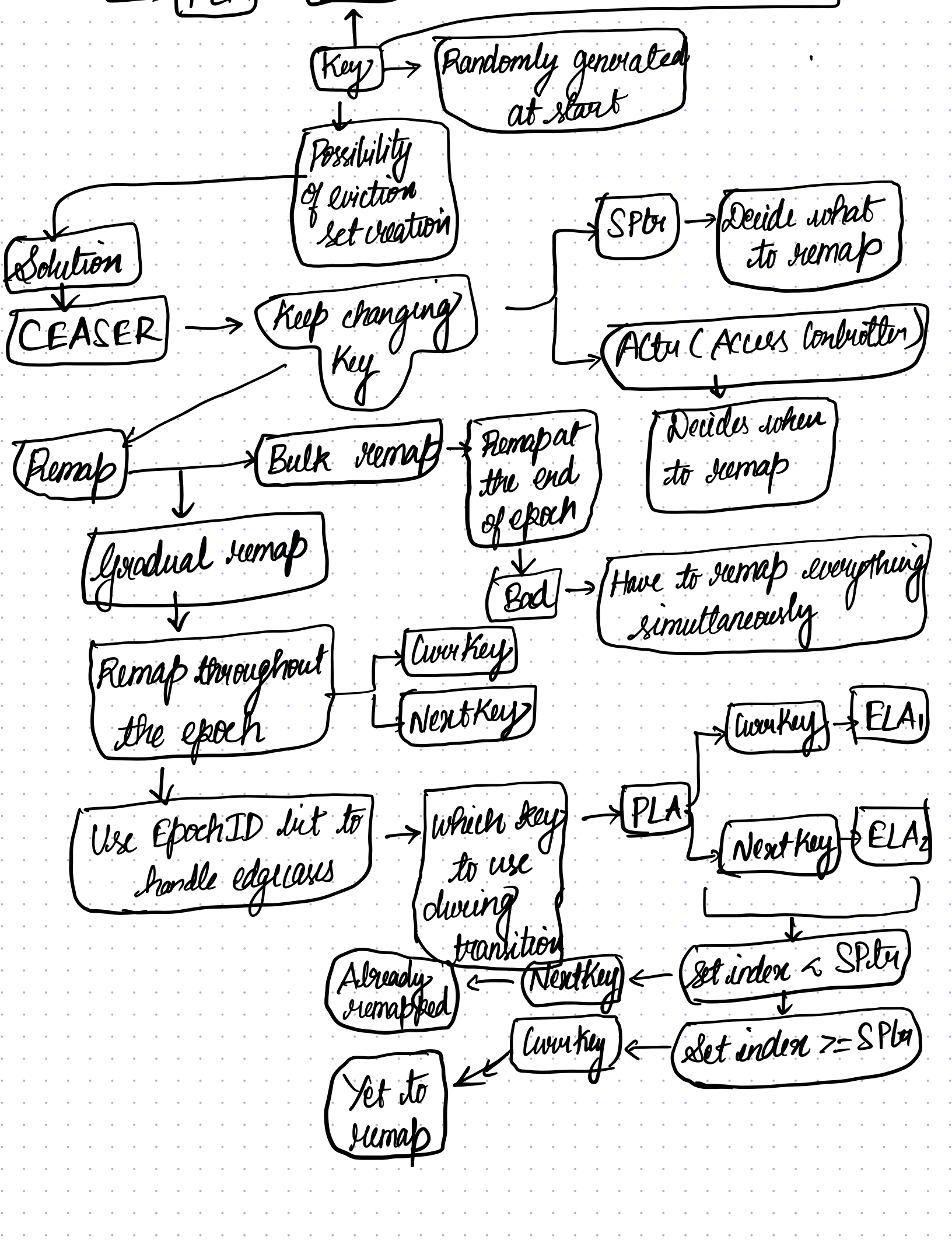


Fig. 8. Example of CEASER with gradual remapping for a cache with four sets (0-3). The subscript with the line denotes the EpochID with which the line was remapped. After every 200 access to the cache (tracked by the access counter, ACtr), all the lines of the set pointed by SPtr gets remapped based on the key of the next Epoch. The shaded area of the cache represents parts of the cache that have undergone remapping based on the key of the next epoch.

Edge case:
→ Two lines with same encrypt address based on different keys
↳ Solution: ~~Store~~ EID → Epoch ID.

Cache access during remapping:
→ Get set index based on both keys.
→ If set index >= SPtr → Yes ⇒ Current key
→ No ⇒ Next key.

Mindmap:
LLC sharing is bad → Covert channel attacks → Flush+Reload / Prime+Probe → Solution → Preserv / Random → Reduce adv.
But beneficial
CEASE → PLA → LLBC → ELA → LLC → ELA → LLBC → PLA


Key
Randomly generated at start
Possibility of eviction set creation
Solution
CEASER
Keep changing Key
SPtr
Decide what to remap
ACtr (Access Controller)
Decides when to remap
Remap
Bulk remap
Remap at the end of epoch
Bad
Have to remap everything simultaneously
Gradual remap
Remap throughout the epoch
CurrKey
NextKey
Use EpochID bit to handle edgecases
Which key to use during transition
PLA
CurrKey
ELA1
NextKey
ELA2
Set index < SPtr
Set index >= SPtr
NextKey
Already remapped
CurrKey
Yet to remap