Lifetime-Leveling LSM-Tree Compaction for Zoned Namespace (ZNS) Solid State Drives (SSD)

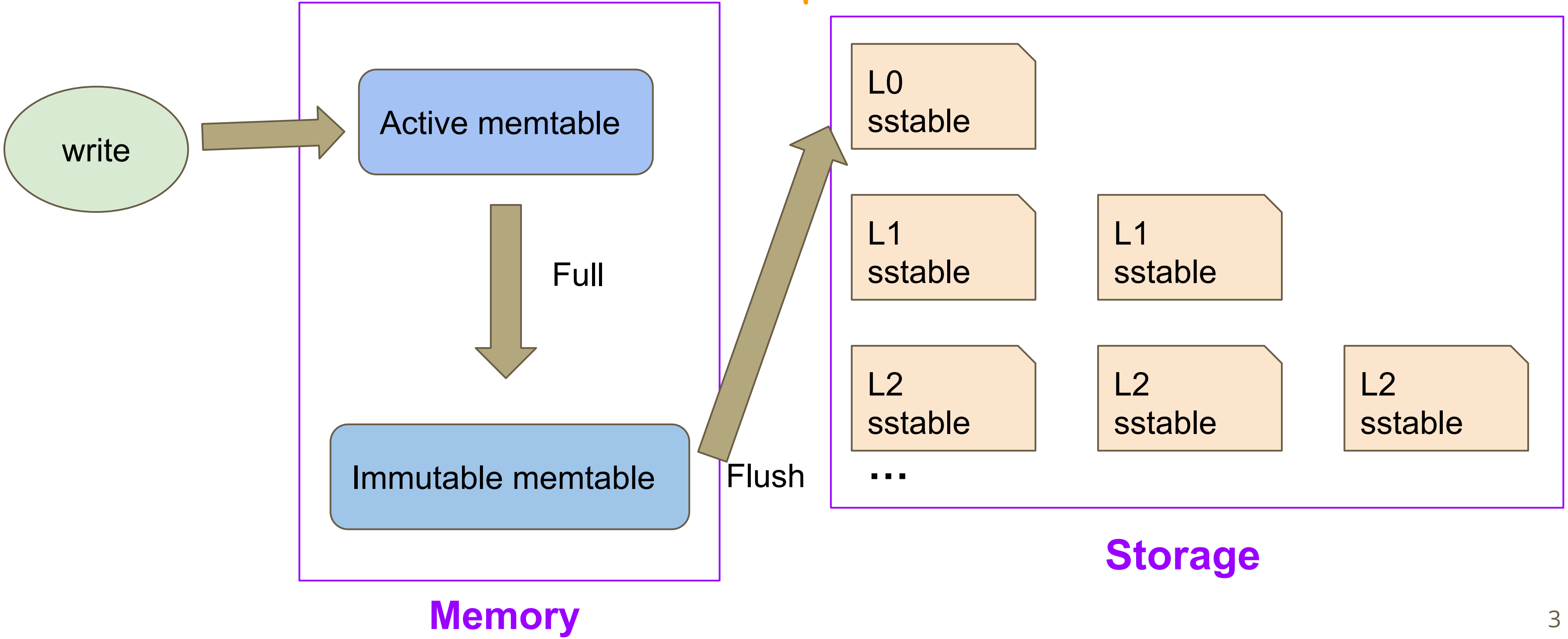
Wei-Tse Kao, Ming-Han Hsieh,
I-Ju Lin, Jingyi Li

Log-Structured Merge Tree (LSM-Tree) Compaction

LSM Tree (Log-Structured Merge Tree)

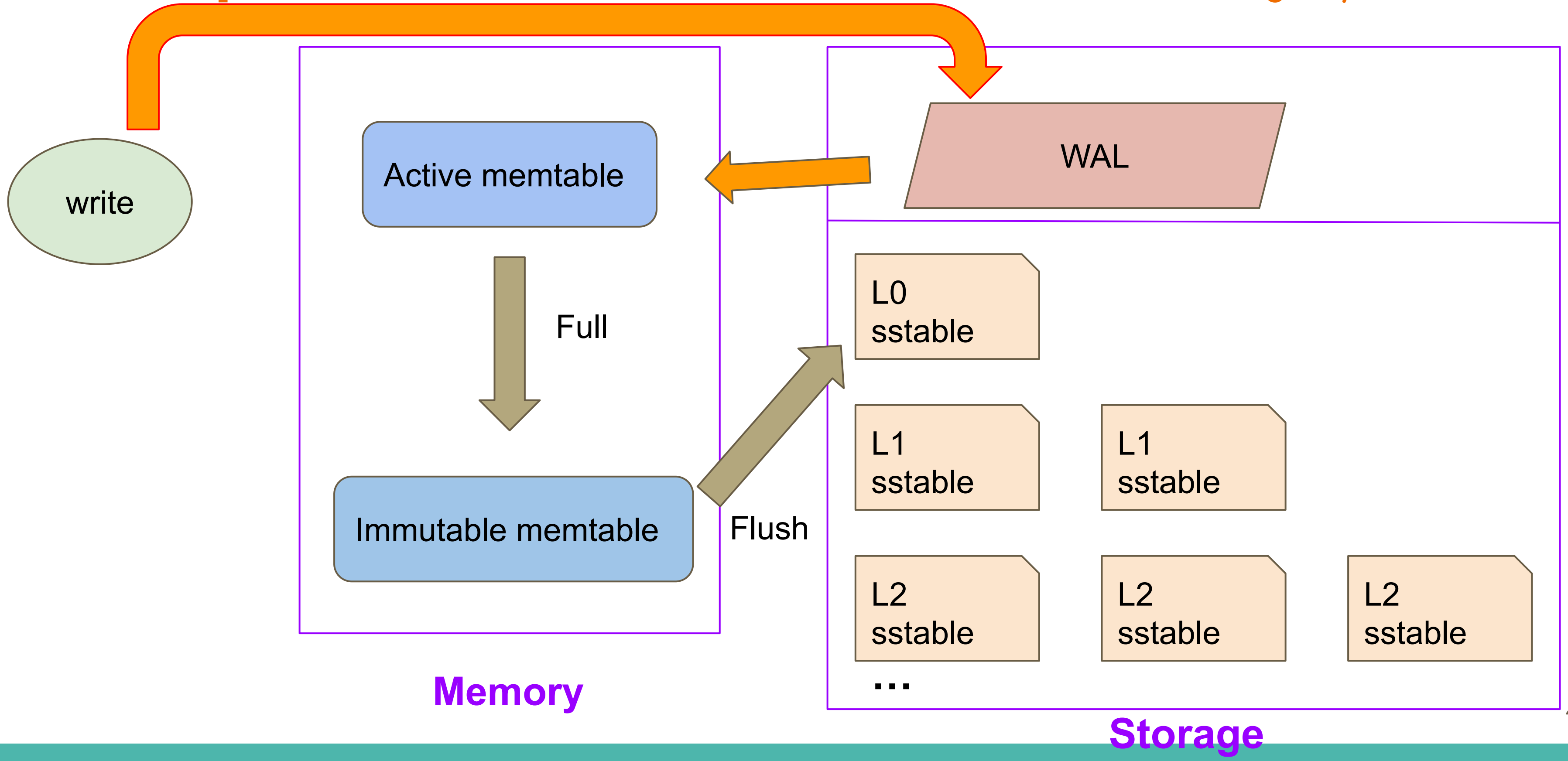
Write Operation

If system crashes, what problem can occur?

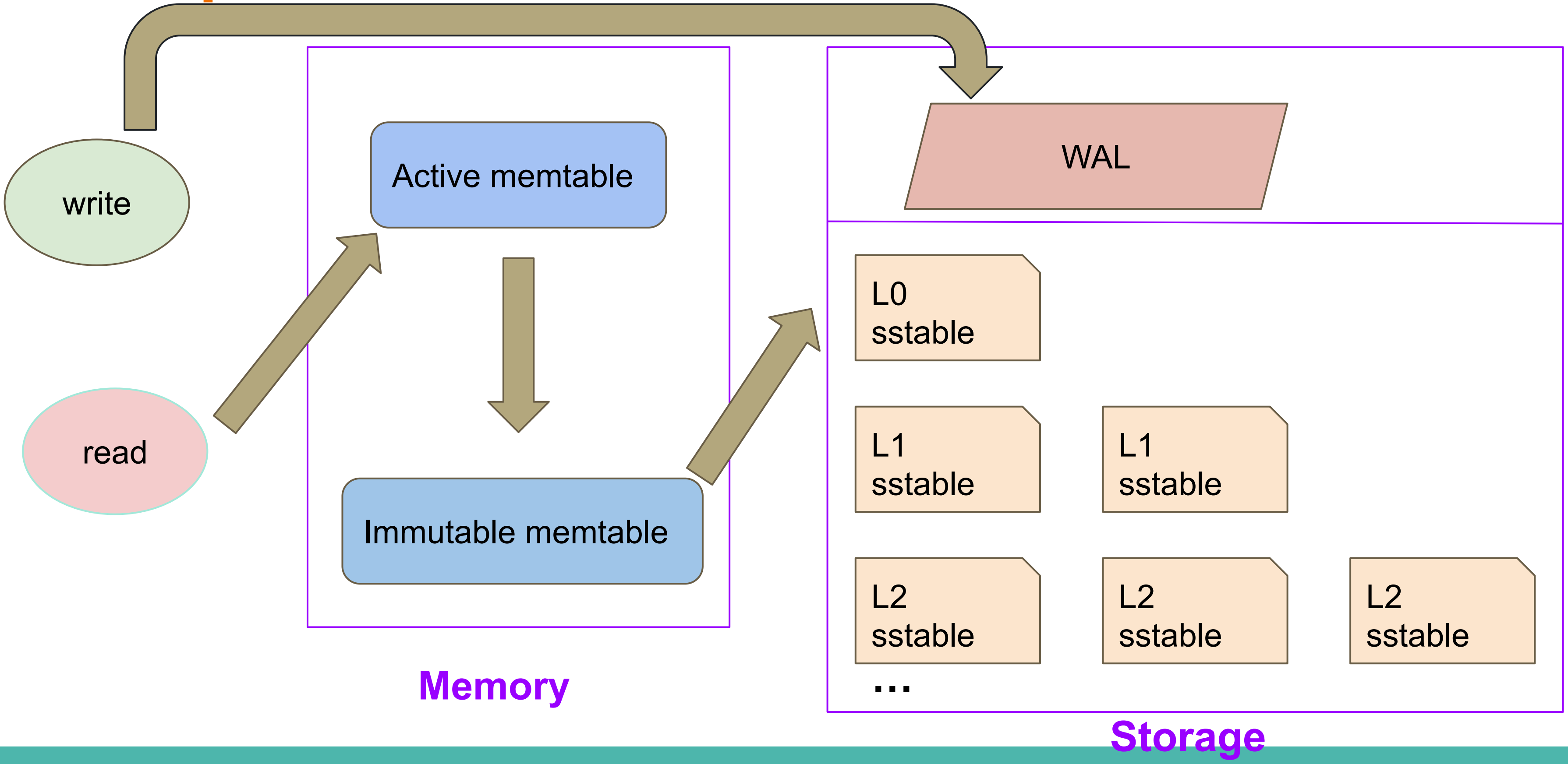


Write Operation

A logging technique used in database management systems and other data storage systems

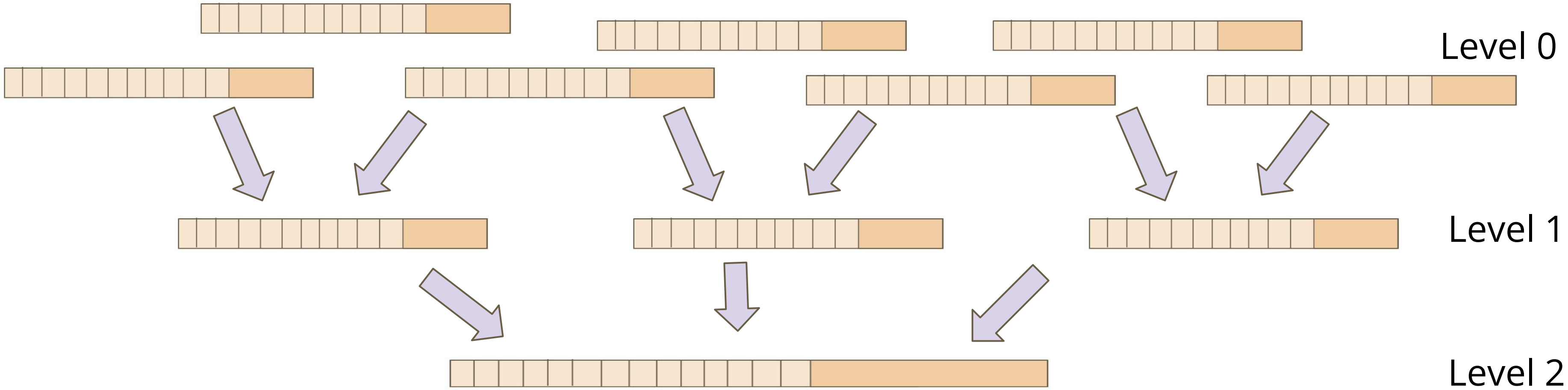


Read Operation



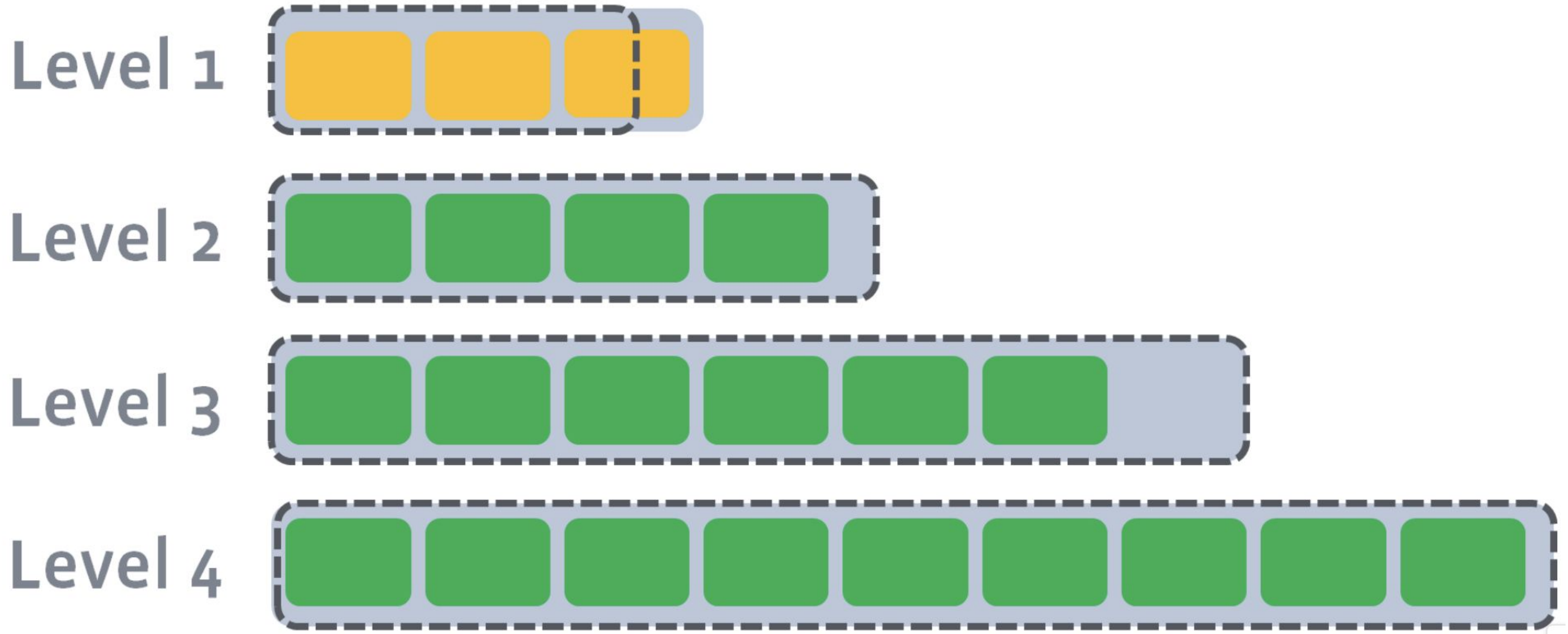
Sorted String Tables (SSTs)

- Each SST has key-value pairs in a sorted form
- SSTs in the same level have non-overlapping key ranges each other at all levels except $L0$
- All levels have thresholds

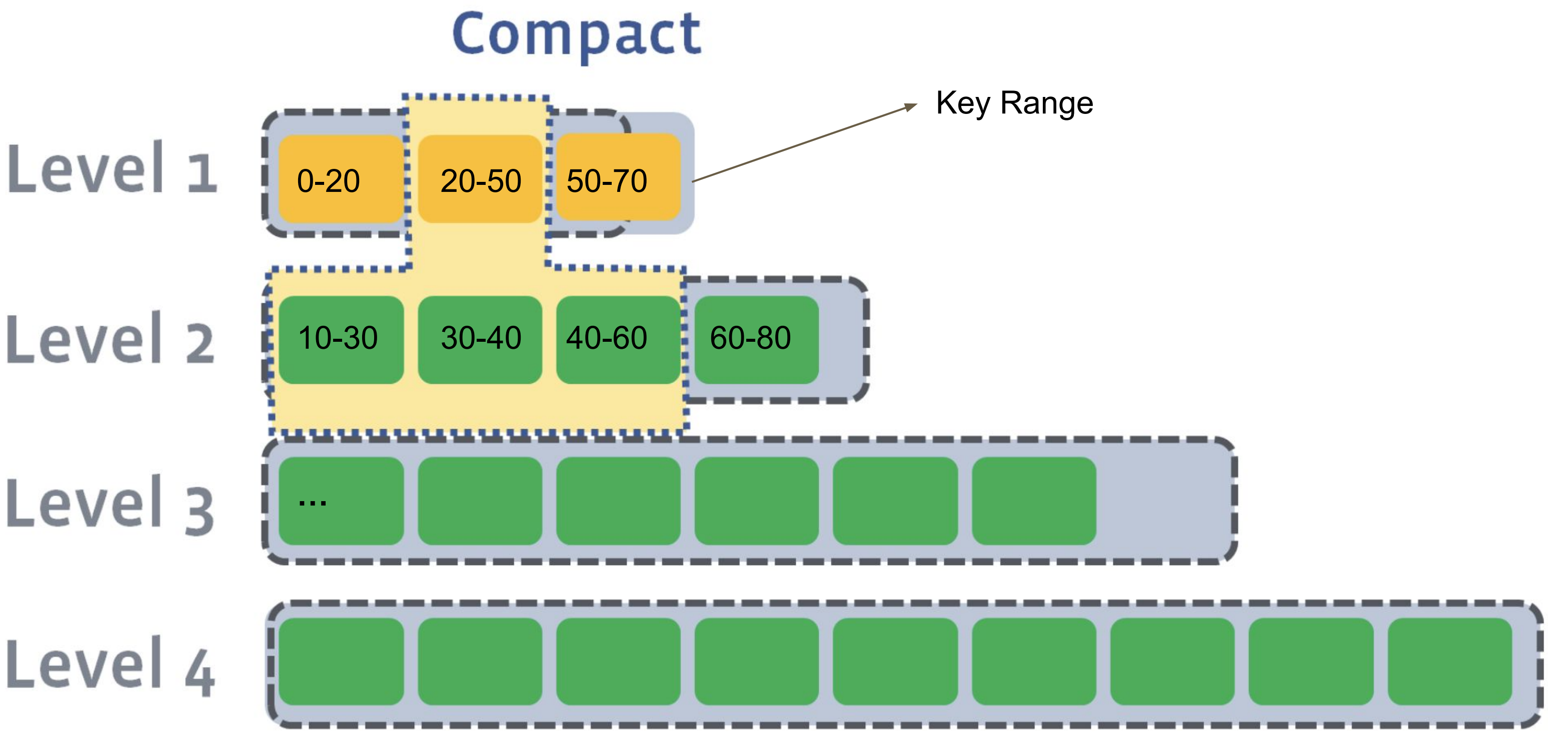


Compaction continues creating fewer, larger and larger files

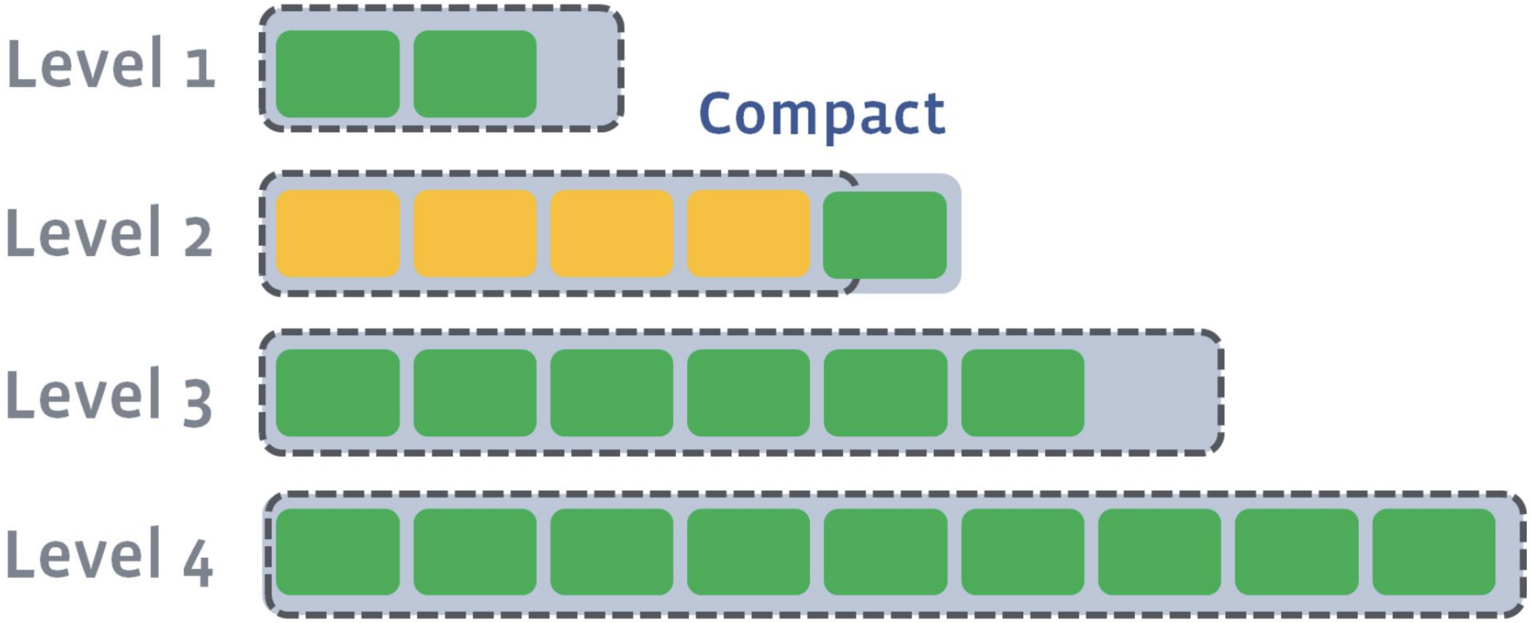
What will happen if the total size of SSTs in L1 exceeds its threshold?



Compaction


Compact
Key Range
Level 1
0-20
20-50
50-70
Level 2
10-30
30-40
40-60
60-80
Level 3
...
Level 4

Compaction



Space Amplification

- Space amplification is a phenomenon that occurs in LSM trees due to their write-optimized design

- Space used by the LSM tree > Actual size of the data being stored

Space Amplification

Occur due to the following reasons:

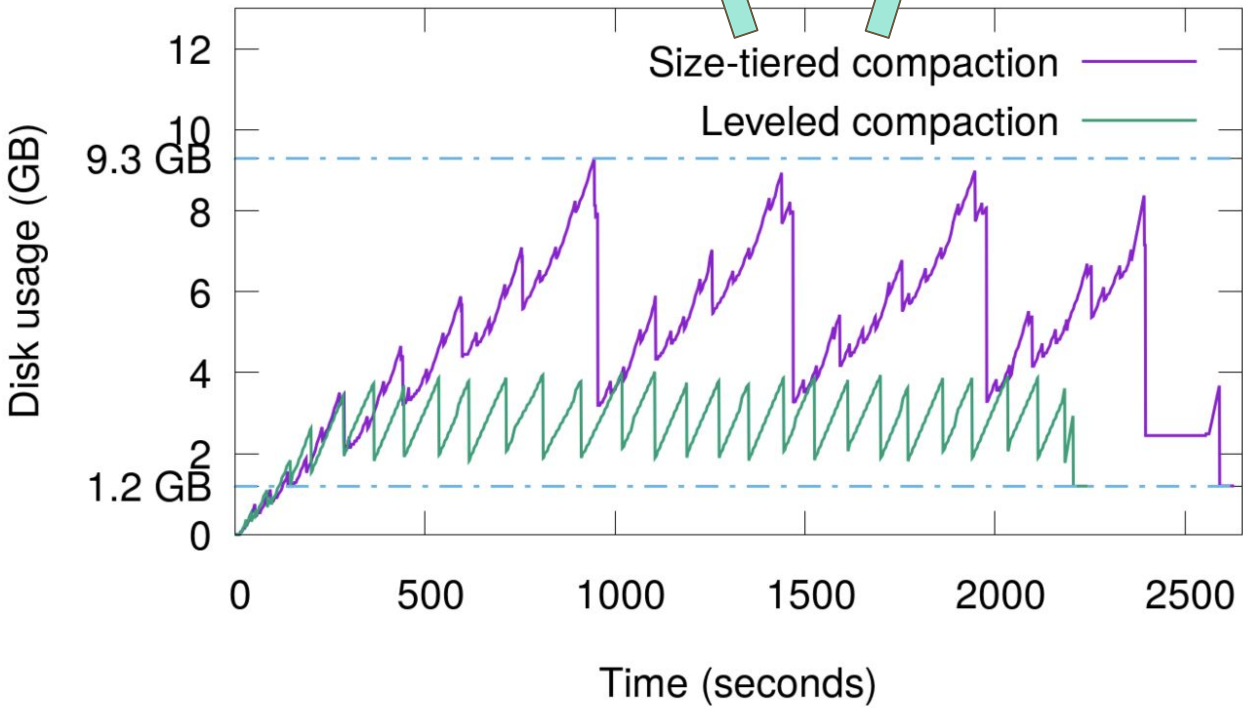
- **Updates**: multiple versions of the same data can coexist temporarily
- **Fragmentation**: When data is added, updated, and deleted, it can create gaps or holes
- **Tombstones**: When data is deleted, a tombstone is added and it consumes extra storage space until it is removed during compaction

Space Amplification

Why size-tiered compaction strategy has higher space amplification?

Overlapping data

Fragmentation



Write Amplification

An undesirable phenomenon associated with flash memory and solid-state drives where “Amount of data written to the system > Amount of data that the user intended to write”

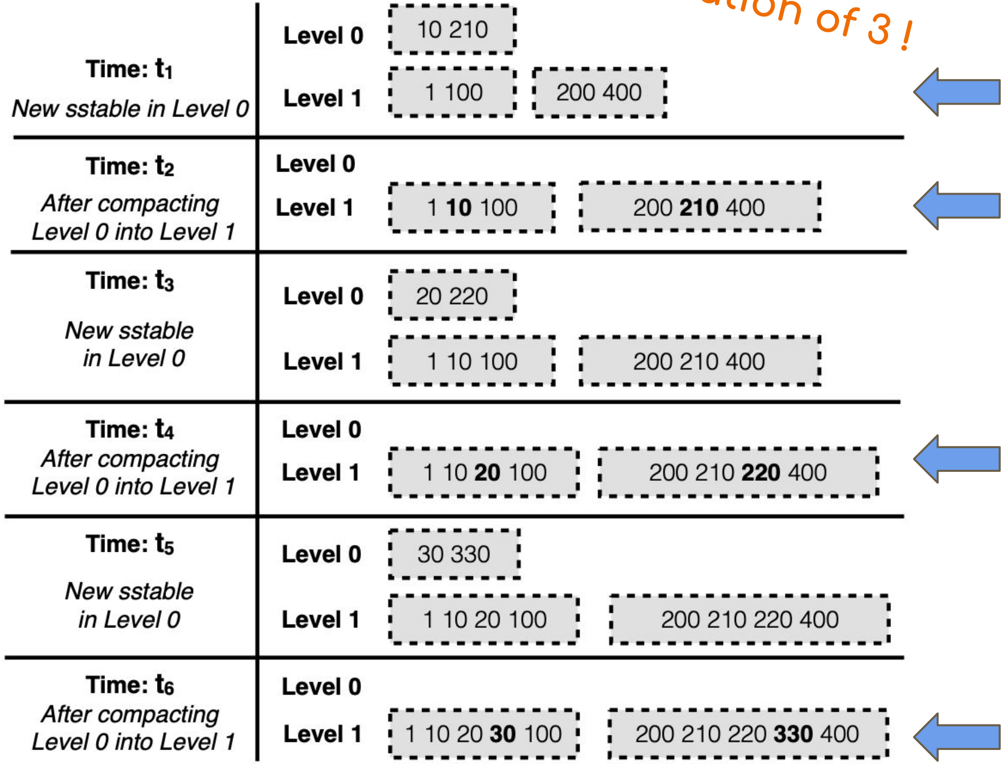
Write Amplification

High write amplification of LSM key-value stores can be traced to multiple rewrites of sstables during compaction

Overwriting of L logical pages of data

O(L) = 3

Measured write amplification of 3 !

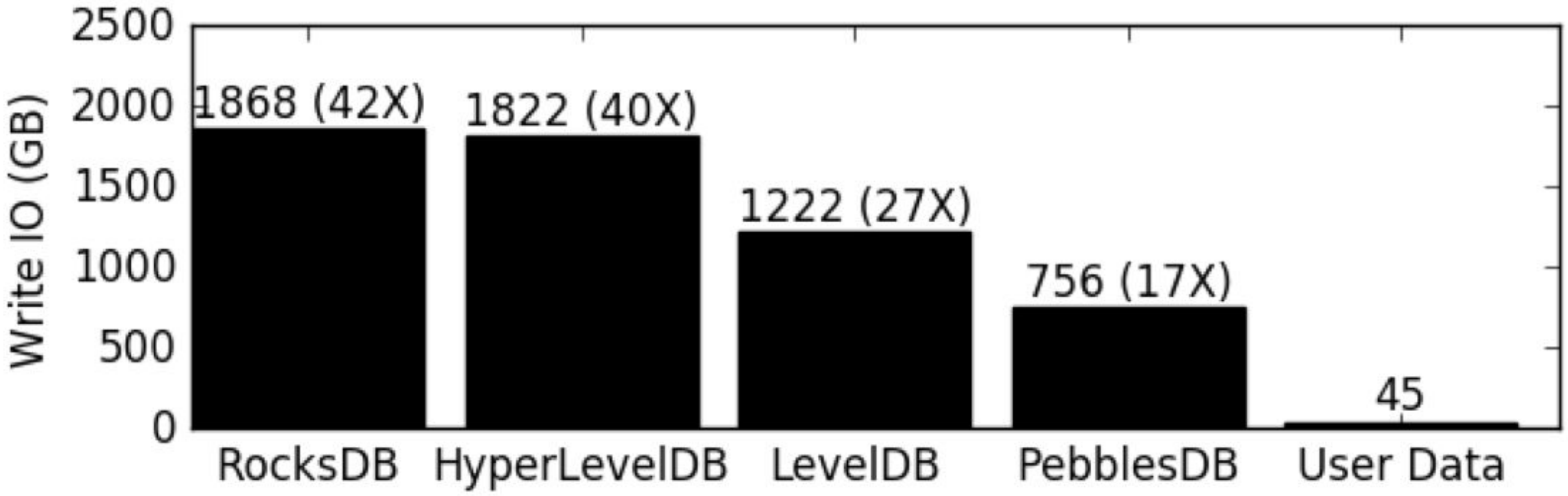


The Level 1 sstables are rewritten every time a Level 0 sstable is compacted !

PebblesDB: Building Key-Value Stores using Fragmented Log-Structured Merge Trees, SOSP 2017

Write Amplification

The total write IO (in GB) for different key-value stores when 500 million key-value pairs (totaling 45 GB) are inserted or updated.



PebblesDB: Building Key-Value Stores using Fragmented Log-Structured Merge Trees, SOSP 2017

Write Requests to LSM Tree is Sequential

LSM Tree —> Well-Suited to ZNS Storage Devices

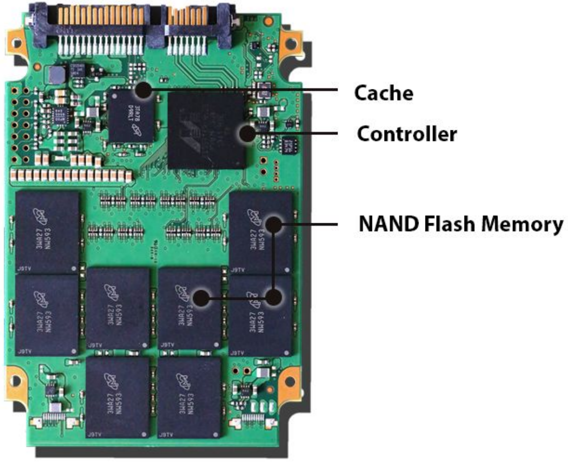
Zoned Namespace (ZNS)
Solid State Drives (SSD)

Solid State Drives (SSD)

HDD
3.5"

What's the difference?

SSD
2.5"



Black box vs White box SSD

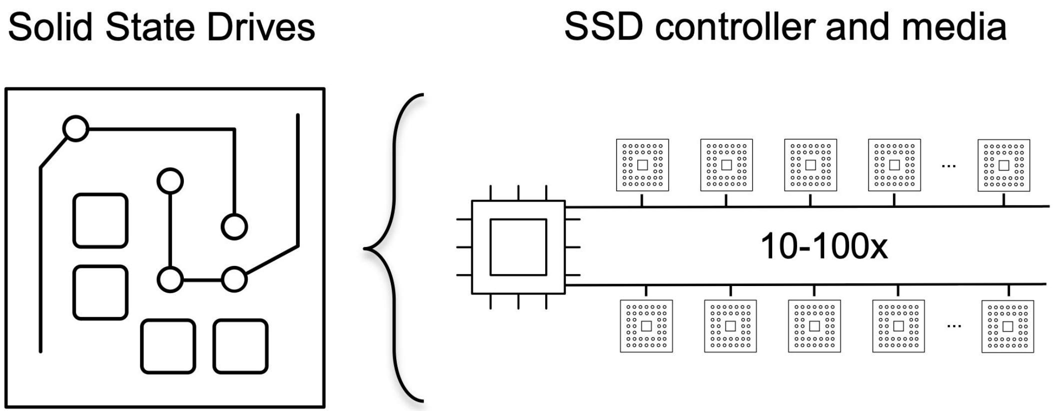
Black box SSD

White box SSD

Black box SSD	White box SSD
Traditional	Programmable
Computational	Open-channel
Streams	**ZNS**

Zoned Namespaces (ZNS) SSD

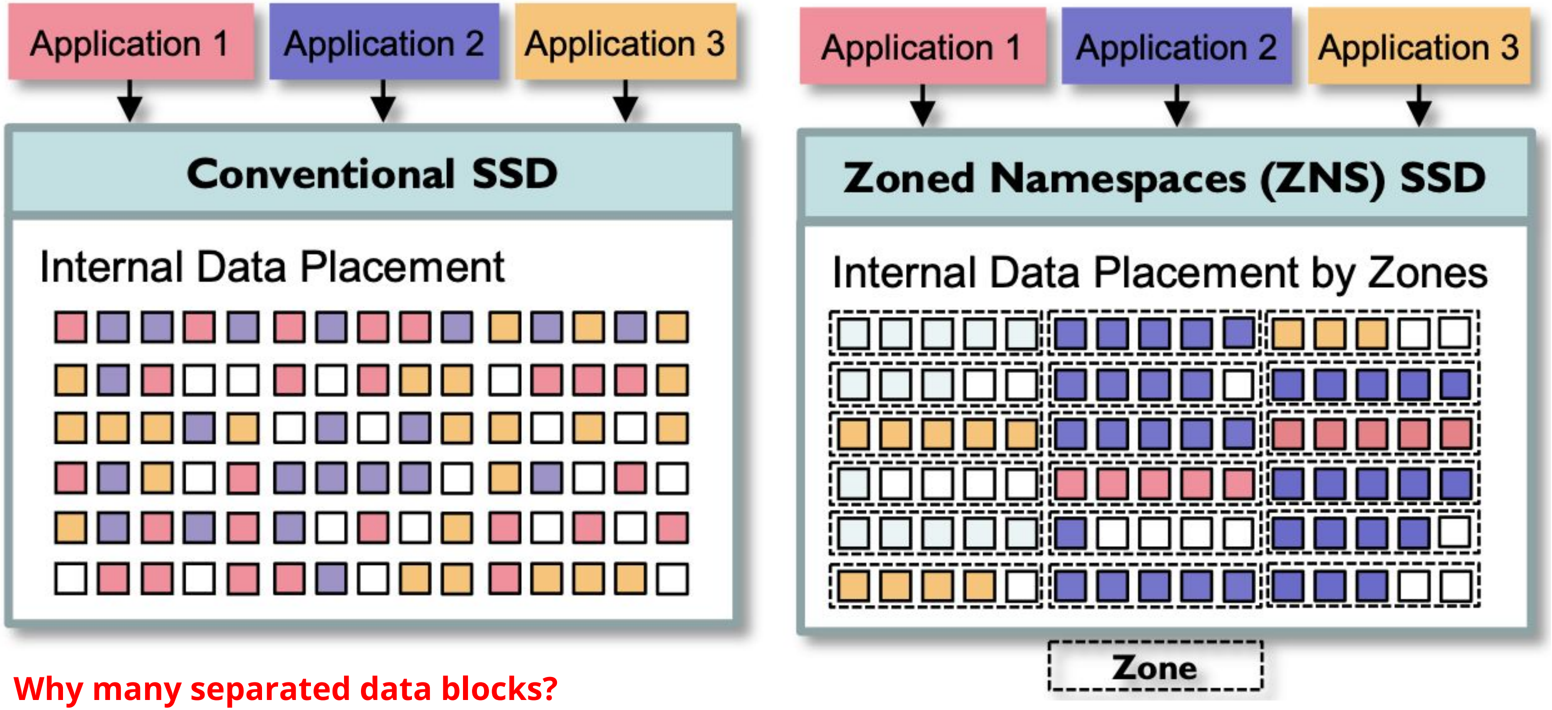
- The **logical block address space** is divided into **fixed-size zones**.
- Each zone must be **written sequentially** and **reset explicitly for reuse**



Zoned Namespaces (ZNS) SSDs: Disrupting the Storage Industry, SDC2020.

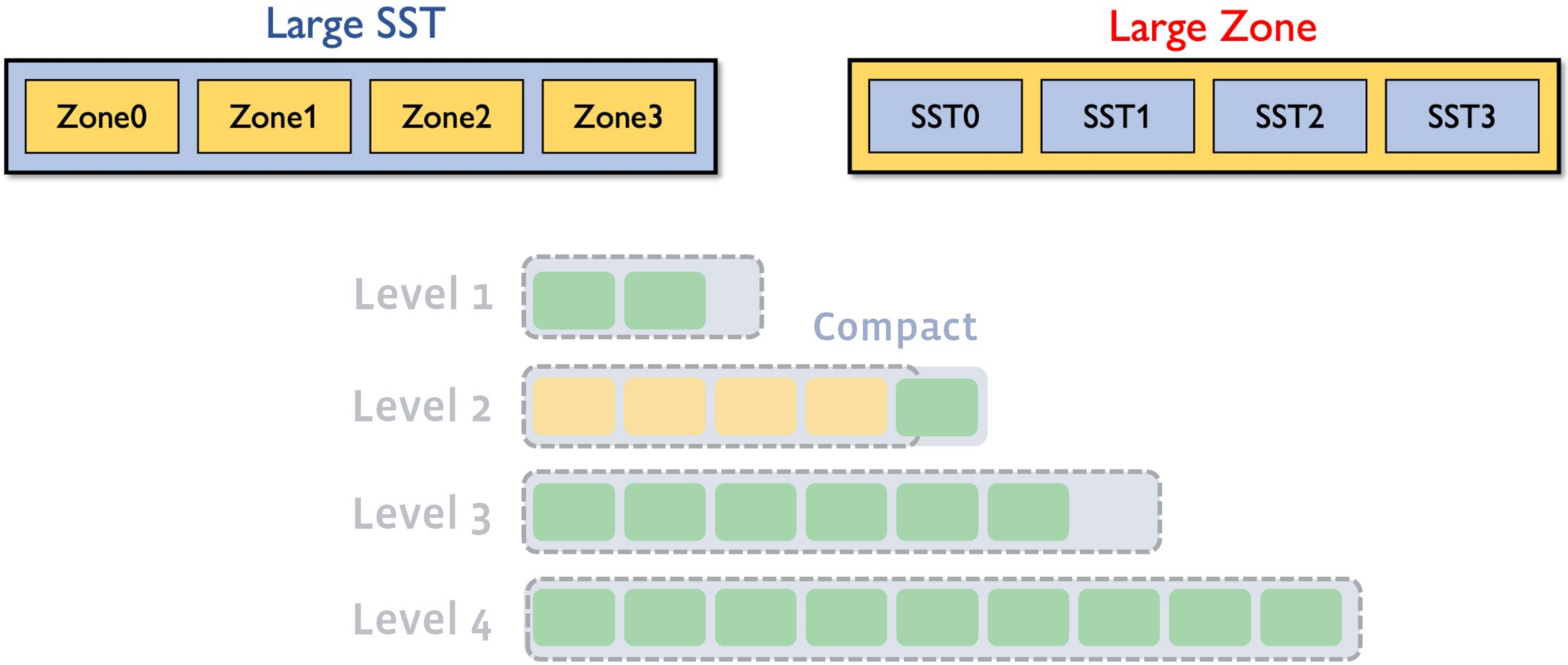
Zoned Namespaces (ZNS) SSD

What’s the difference? Any observation?



Why many separated data blocks?

LSM-tree Sorted String Tables (SSTs) at ZNS



Zone

Zone State	Description	Active Resources	Open Resources
Empty	Accepts writes. Reads return predefined data.	No	No
Implicitly Opened	Accepts writes. Reads before the WP returns valid data, else predefined.	Yes	Yes
Explicitly Opened	Accepts writes. Reads before the WP returns valid data, else predefined.	Yes	Yes
Closed	Accept writes. Reads before the WP returns valid data, else predefined.	Yes	No
Read Only	No writes. Reads before the WP returns valid data, else predefined. Intermediate state before offline state.	No	No
Offline	LBAs are neither writeable or readable.	No	No

State Machine

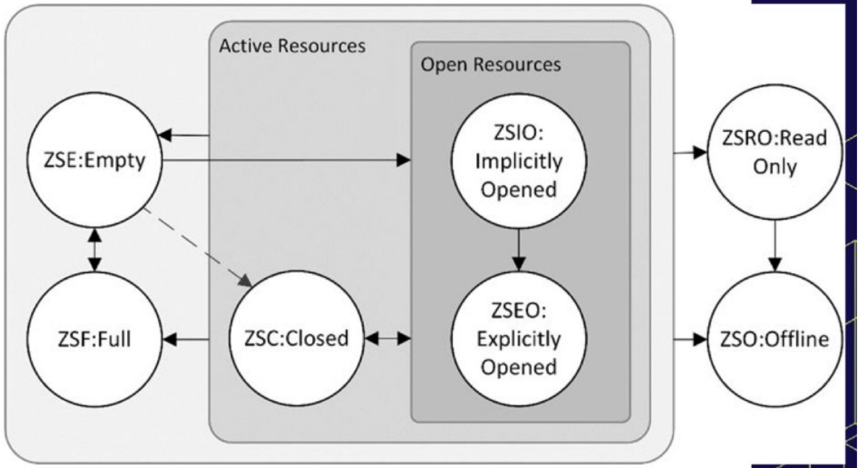
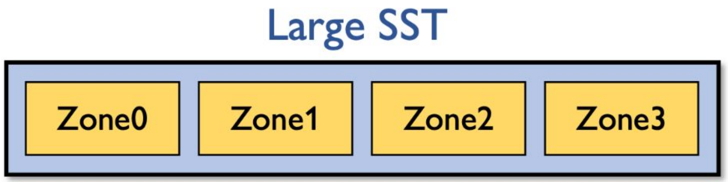
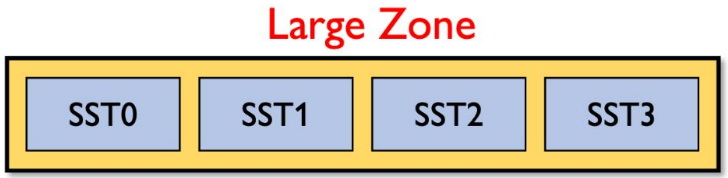


Figure source: The NVM Express Zoned

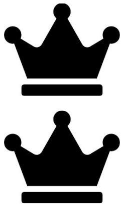
LSM-tree Sorted String Tables (SSTs) at ZNS





Compaction Cost

Memory Size

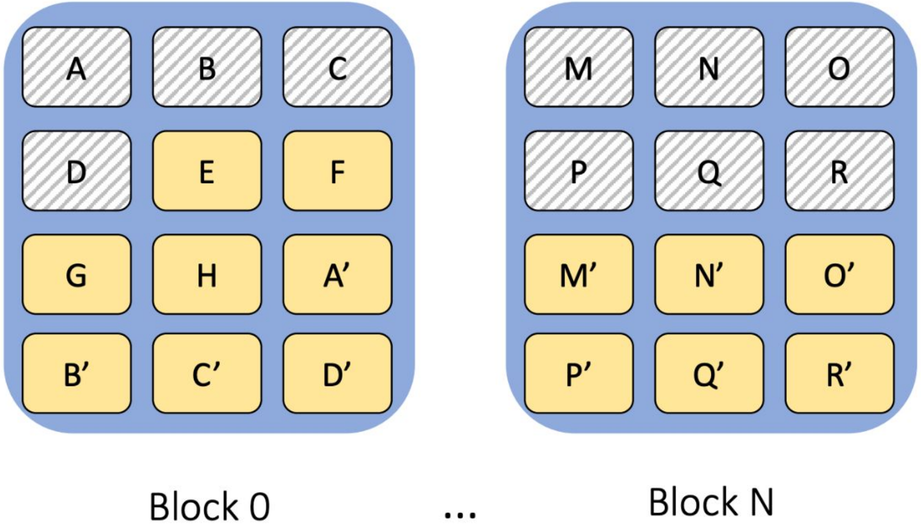


Updates in SSD

Mixed of valid and invalid pages in each block.

What if there is no space?

Garbage Collection!

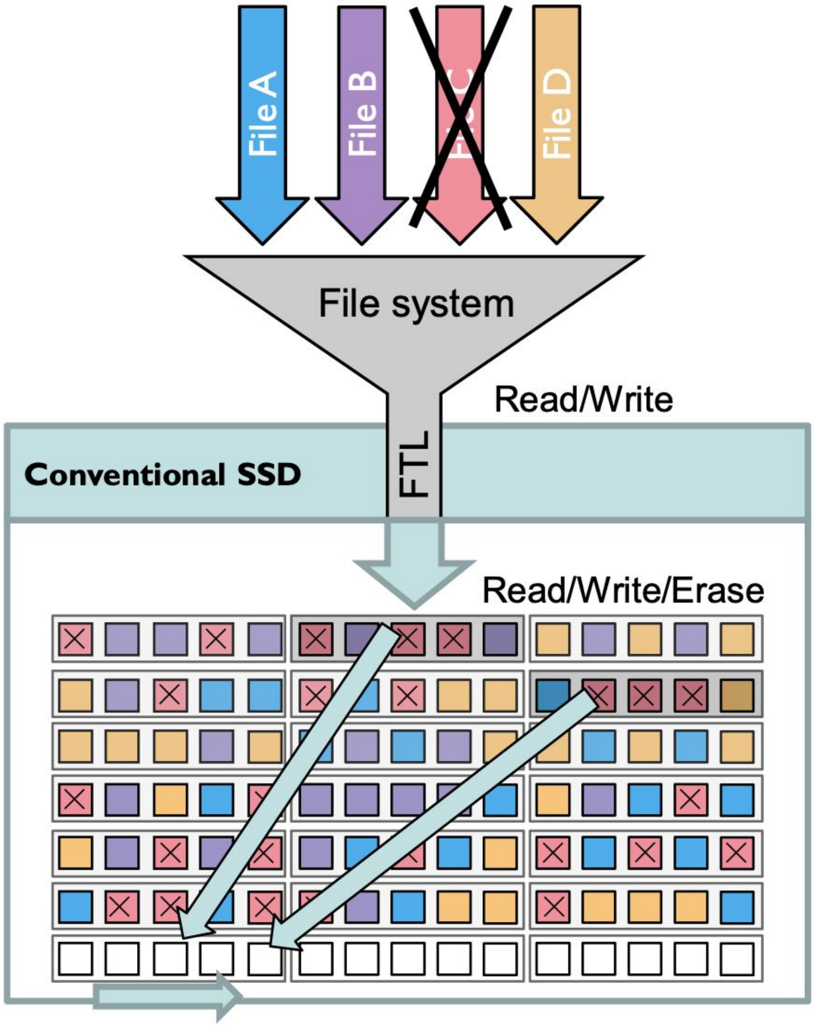


Deletes in SSD

The cost of Garbage Collection (GC):

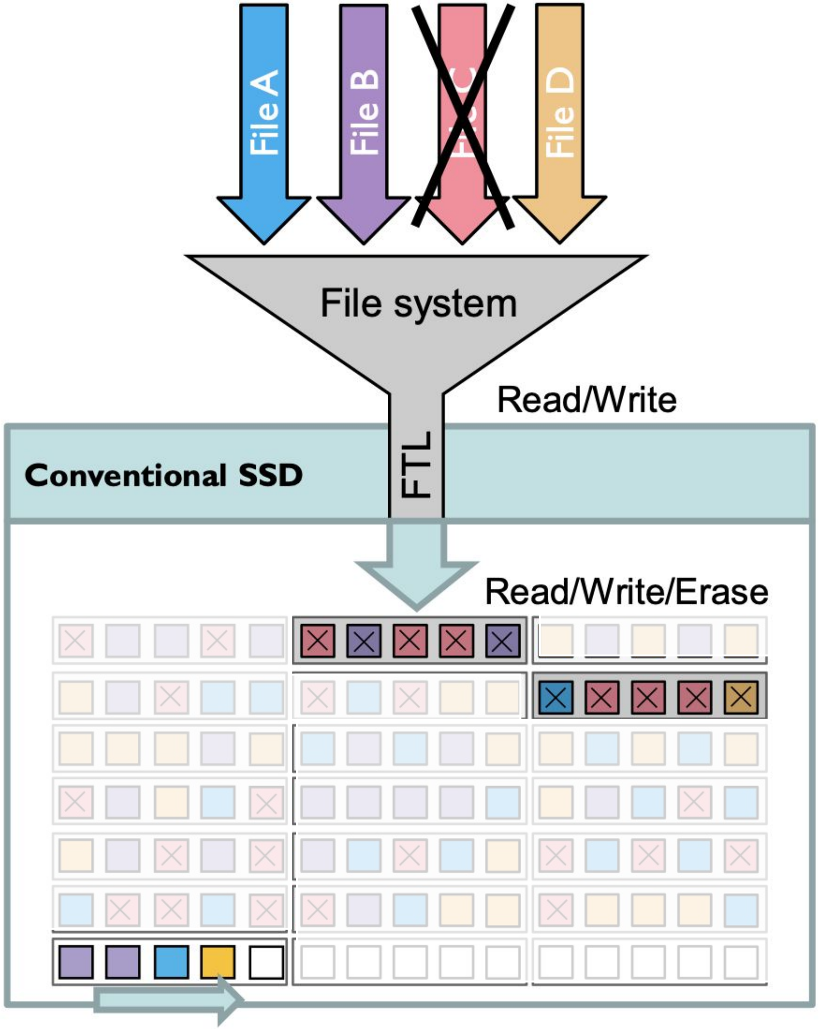
Write Amplification Factor (WAF)

How many rewrites here?



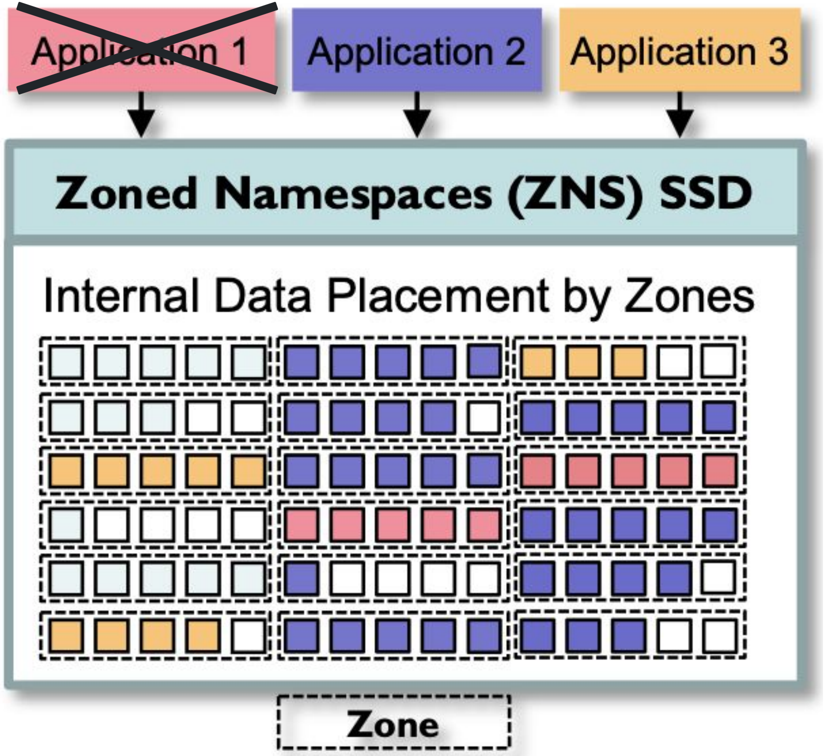
Deletes in SSD

- Lower Write Throughput
- Higher Write Latency
- Increase Device Cost
- Decrease Device Lifetime



Deletes in ZNS SSD

How many rewrites here?

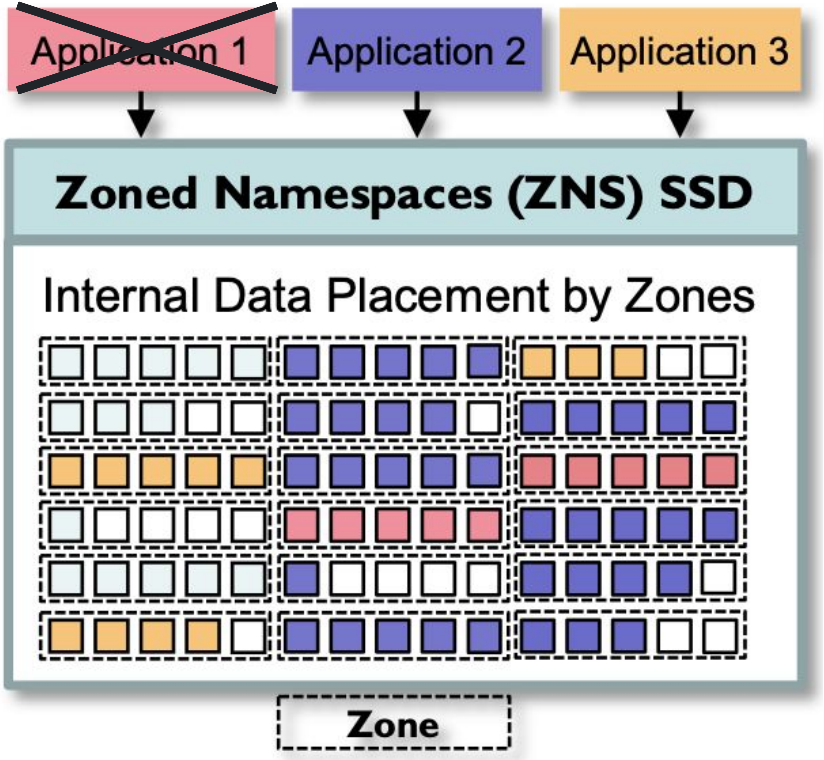


Zoned Namespaces (ZNS) SSDs: Disrupting the Storage Industry, SDC2020.

Deletes in ZNS SSD

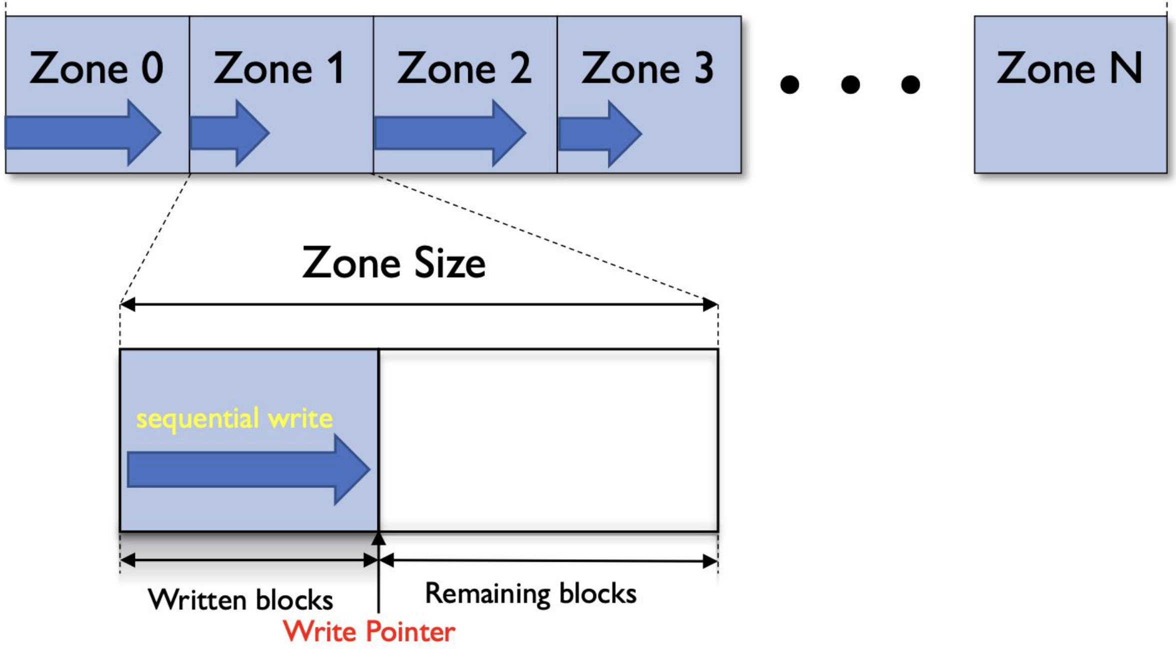
- ~~Lower~~ Write Throughput
- ~~Higher~~ Write Latency
- ~~Increase~~ Device Cost
- ~~Decrease~~ Device Lifetime

reset explicitly for reuse

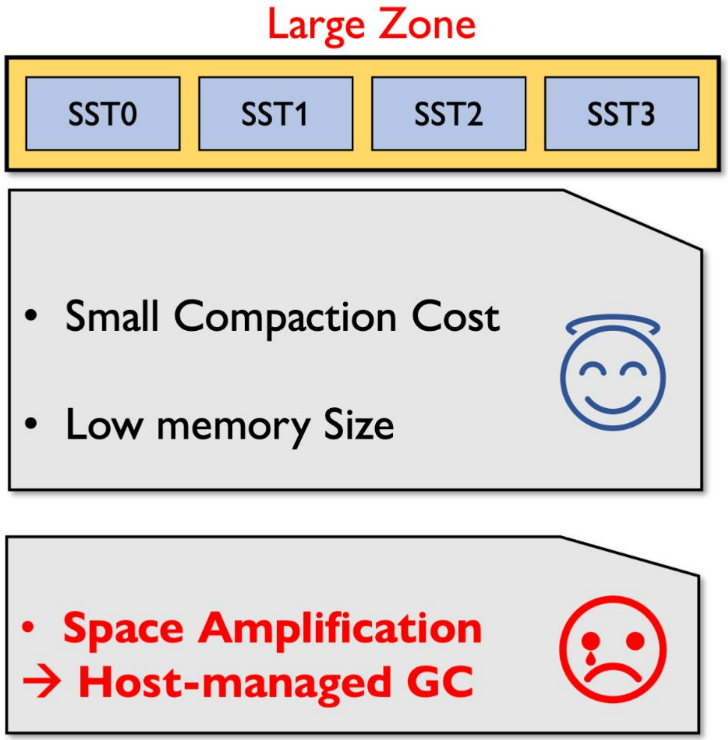


Zoned Namespaces (ZNS) SSDs: Disrupting the Storage Industry, SDC2020.

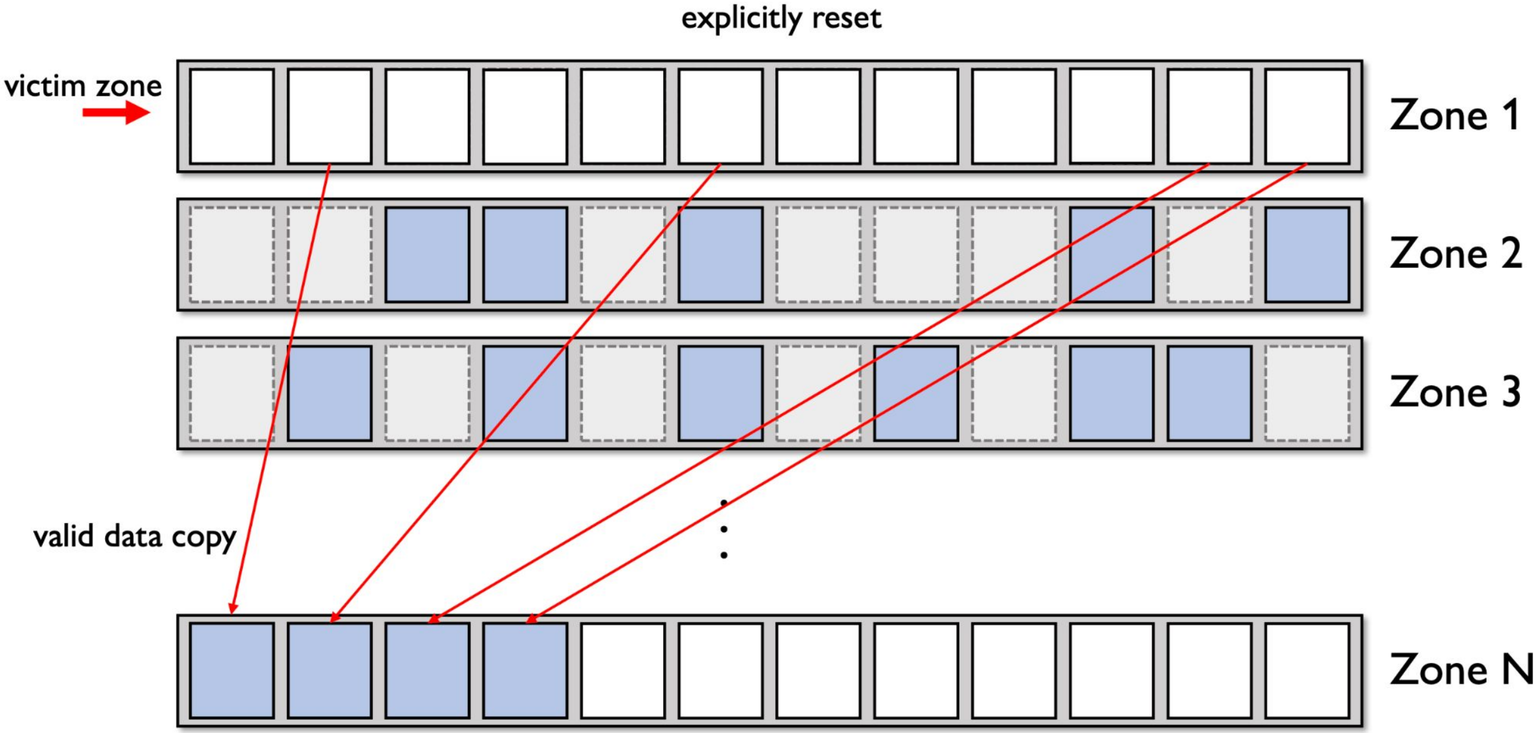
Space Amplification (SA)



LSM-tree Sorted String Tables (SSTs) at ZNS



Host-managed GC on ZNS



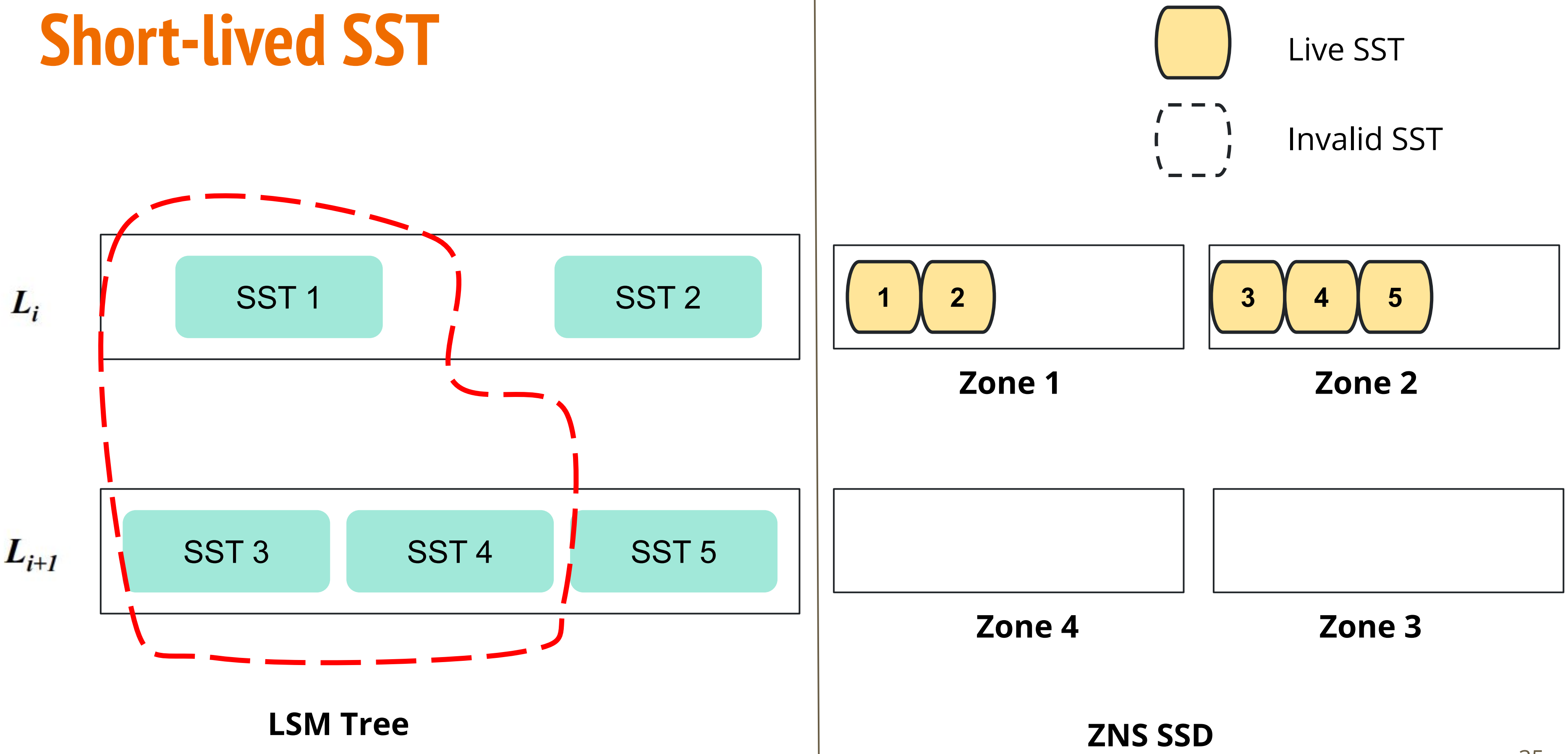
How SST works in ZNS?

Zoned Namespaces (ZNS) SSD

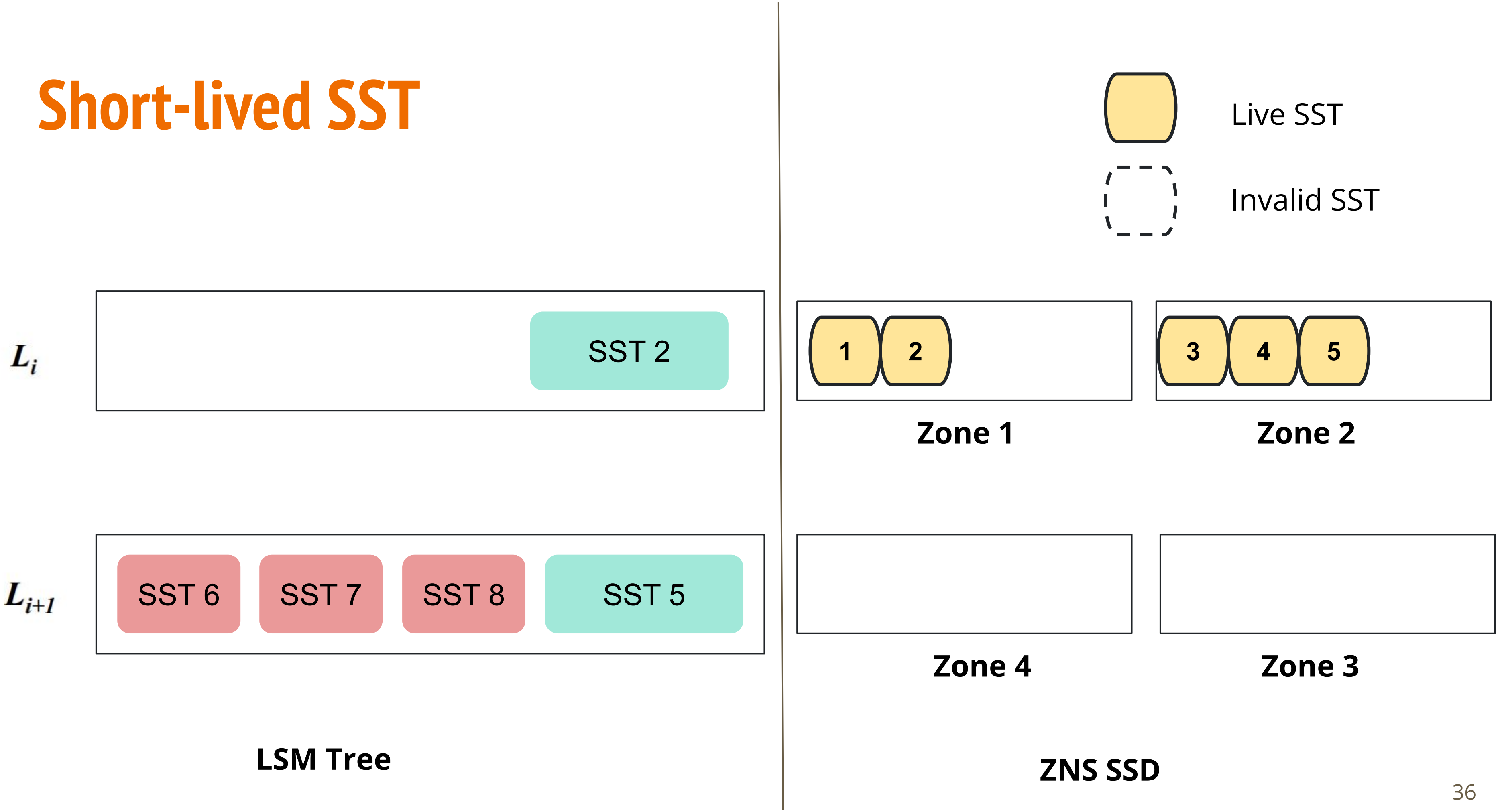
- The **logical address space** is divided into **fixed-size zones**.
- Each zone must be **written sequentially** and **reset explicitly for reuse**

Good for LSM tree

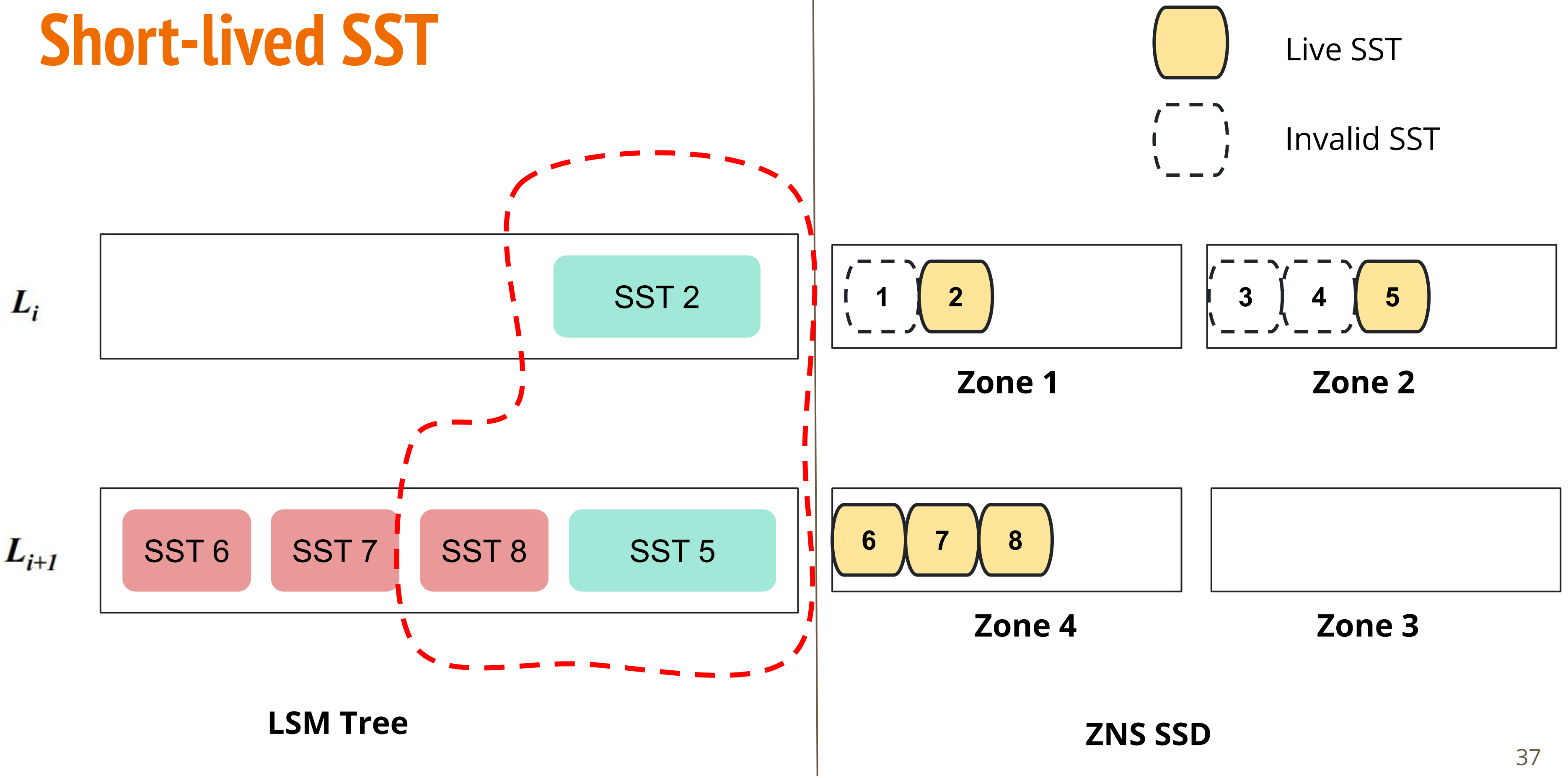
Short-lived SST



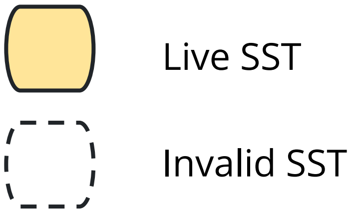
Short-lived SST

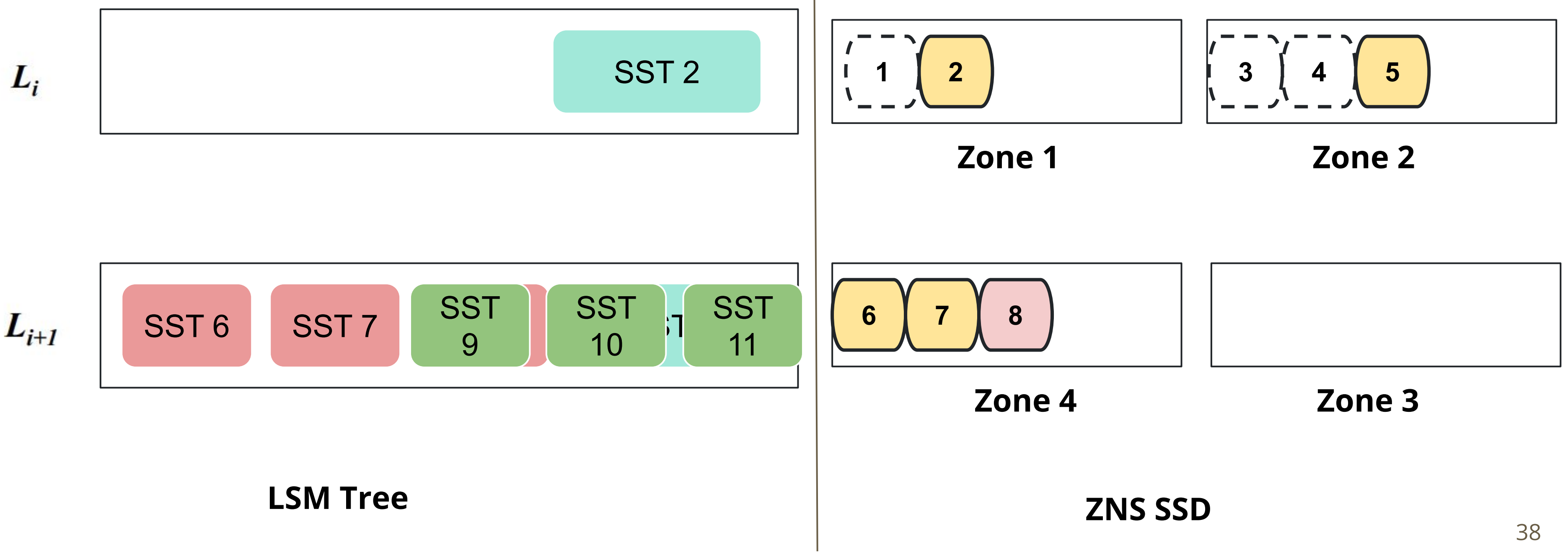


Short-lived SST

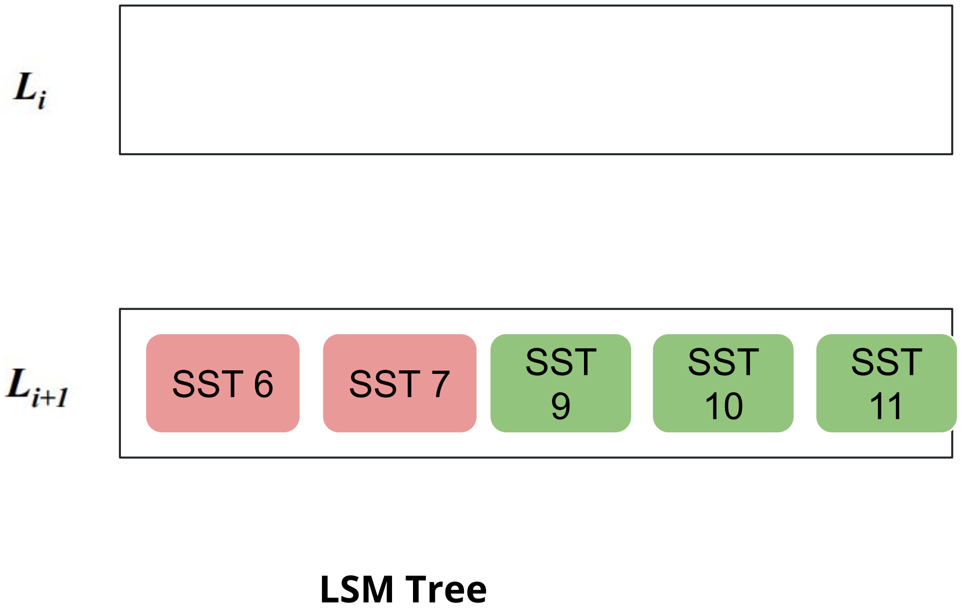


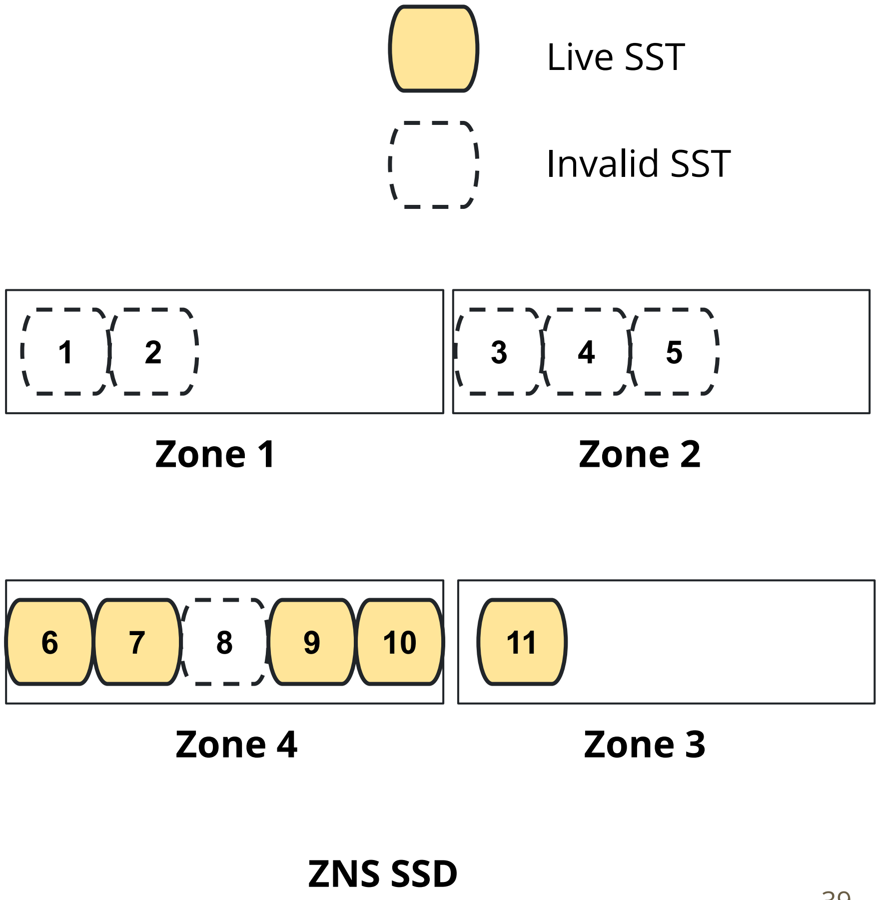
Short-lived SST



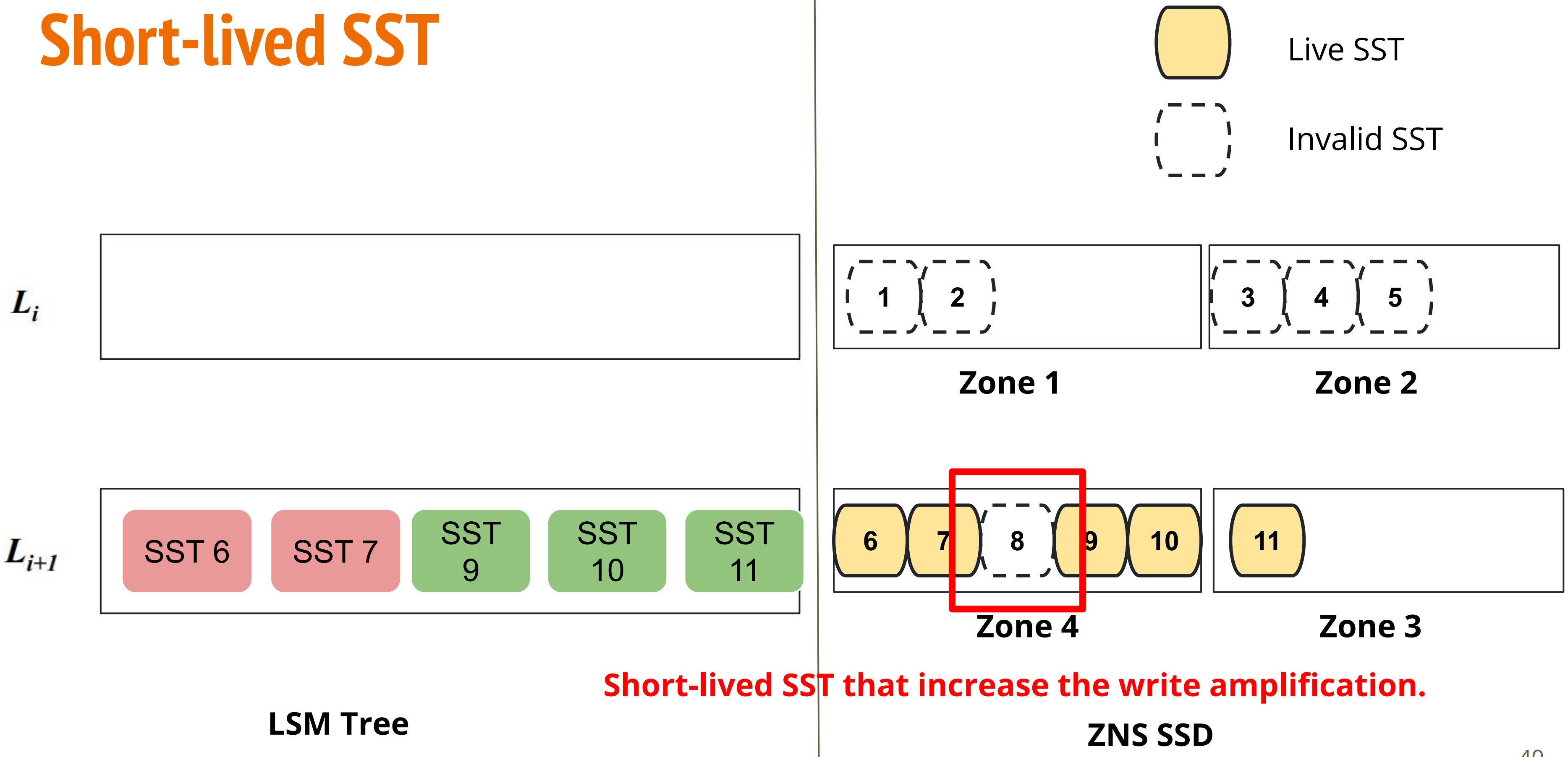


Short-lived SST



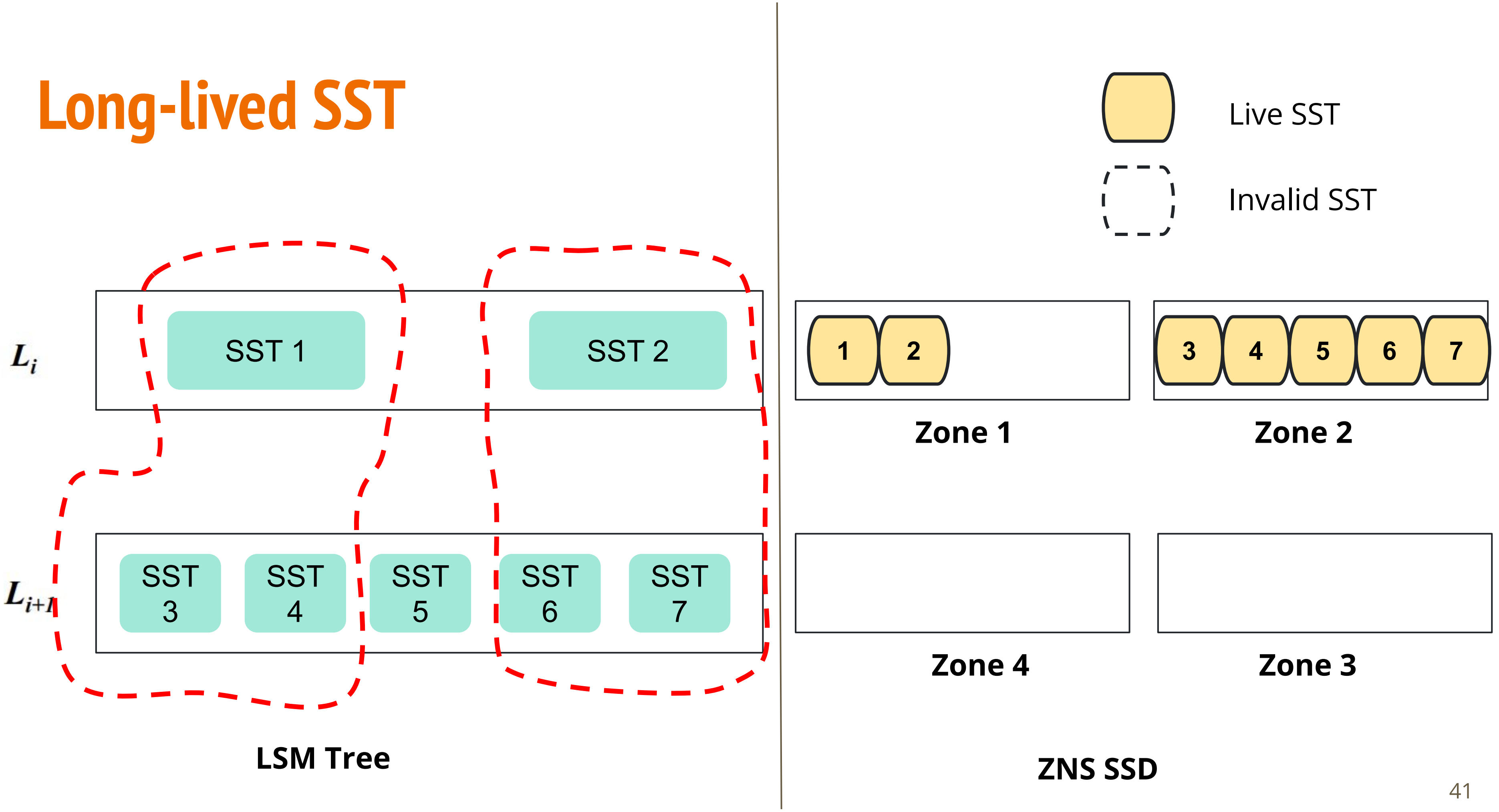


Short-lived SST

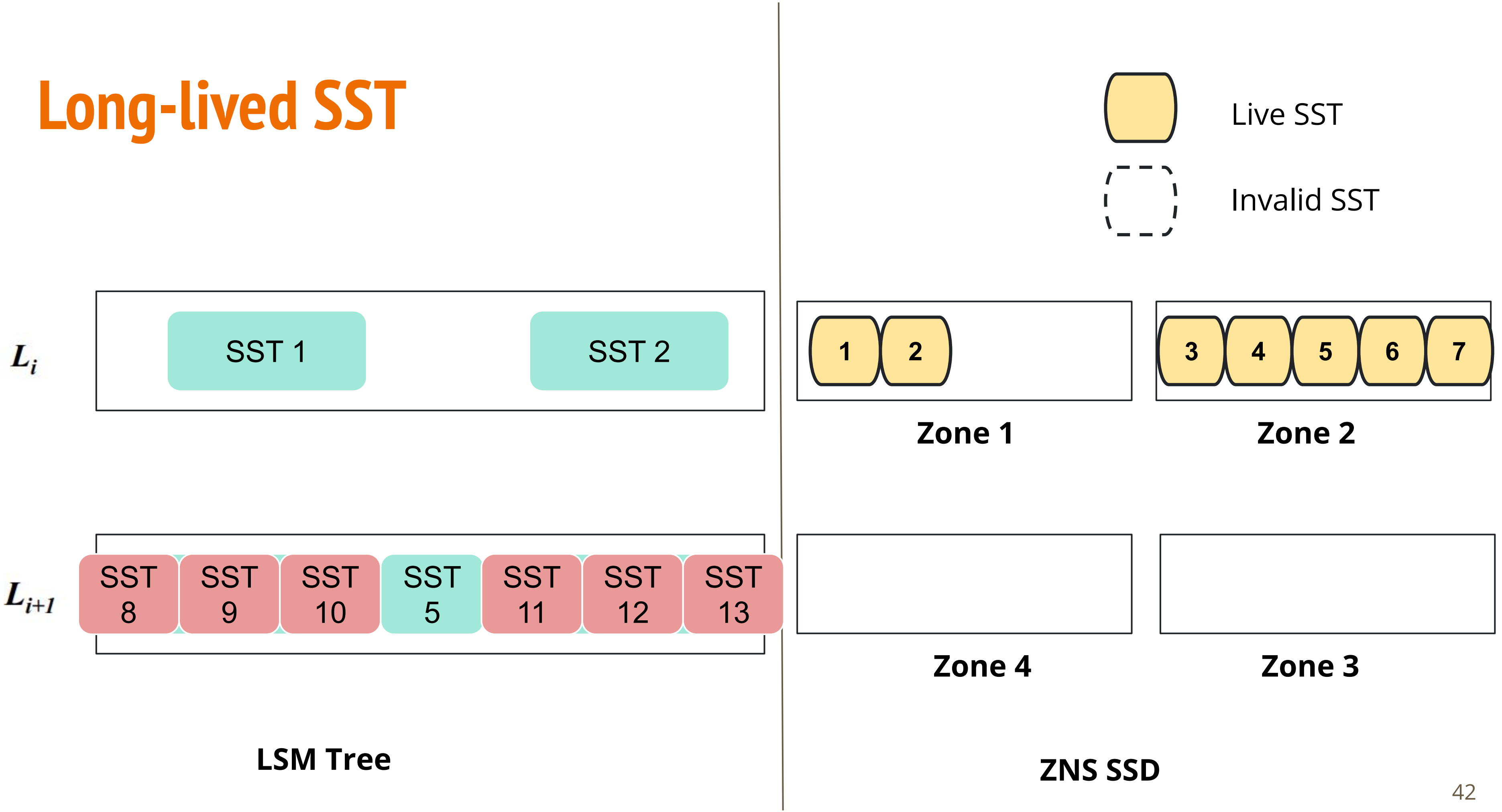


Short-lived SST that increase the write amplification.

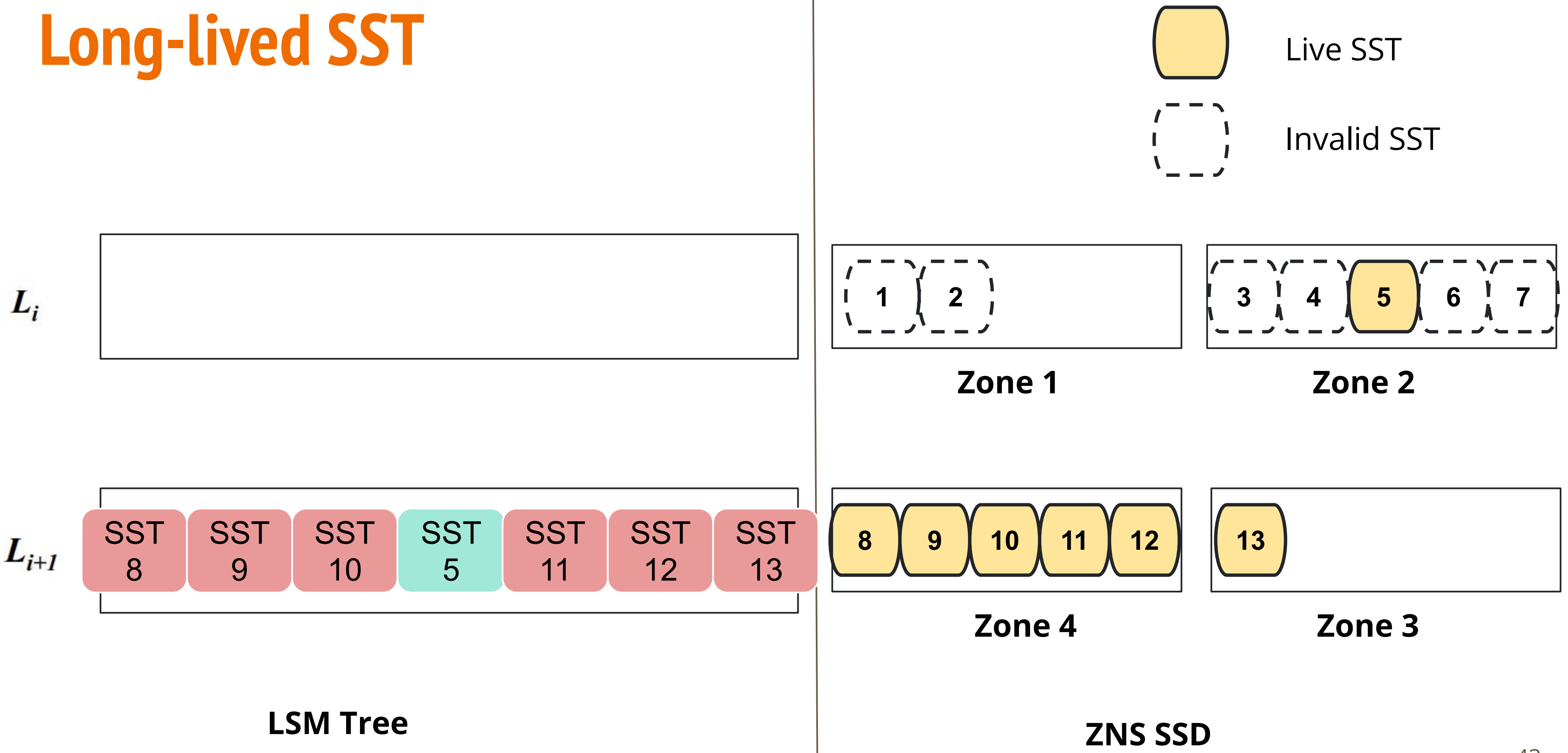
Long-lived SST



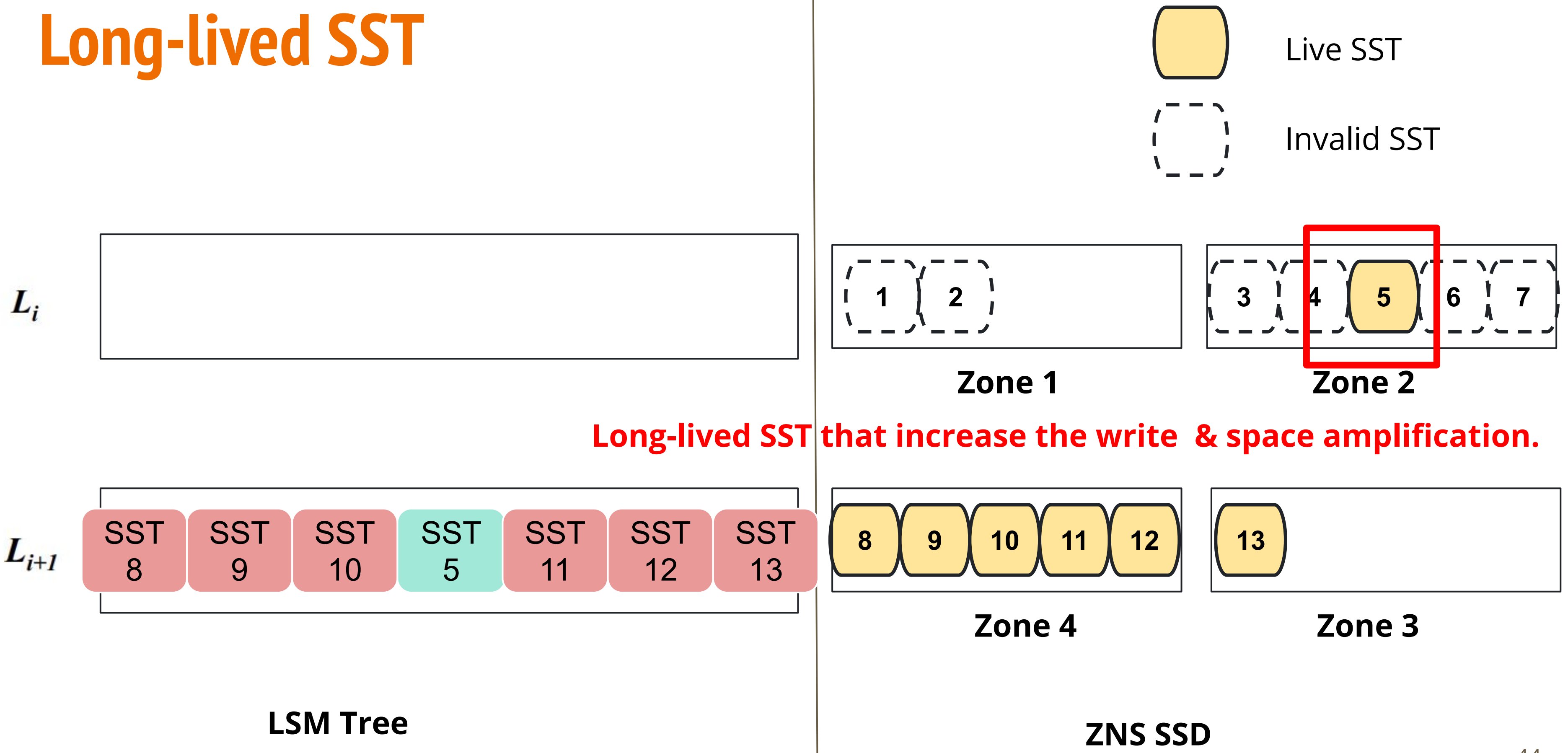
Long-lived SST



Long-lived SST



Long-lived SST



Long-lived SST that increase the write & space amplification.

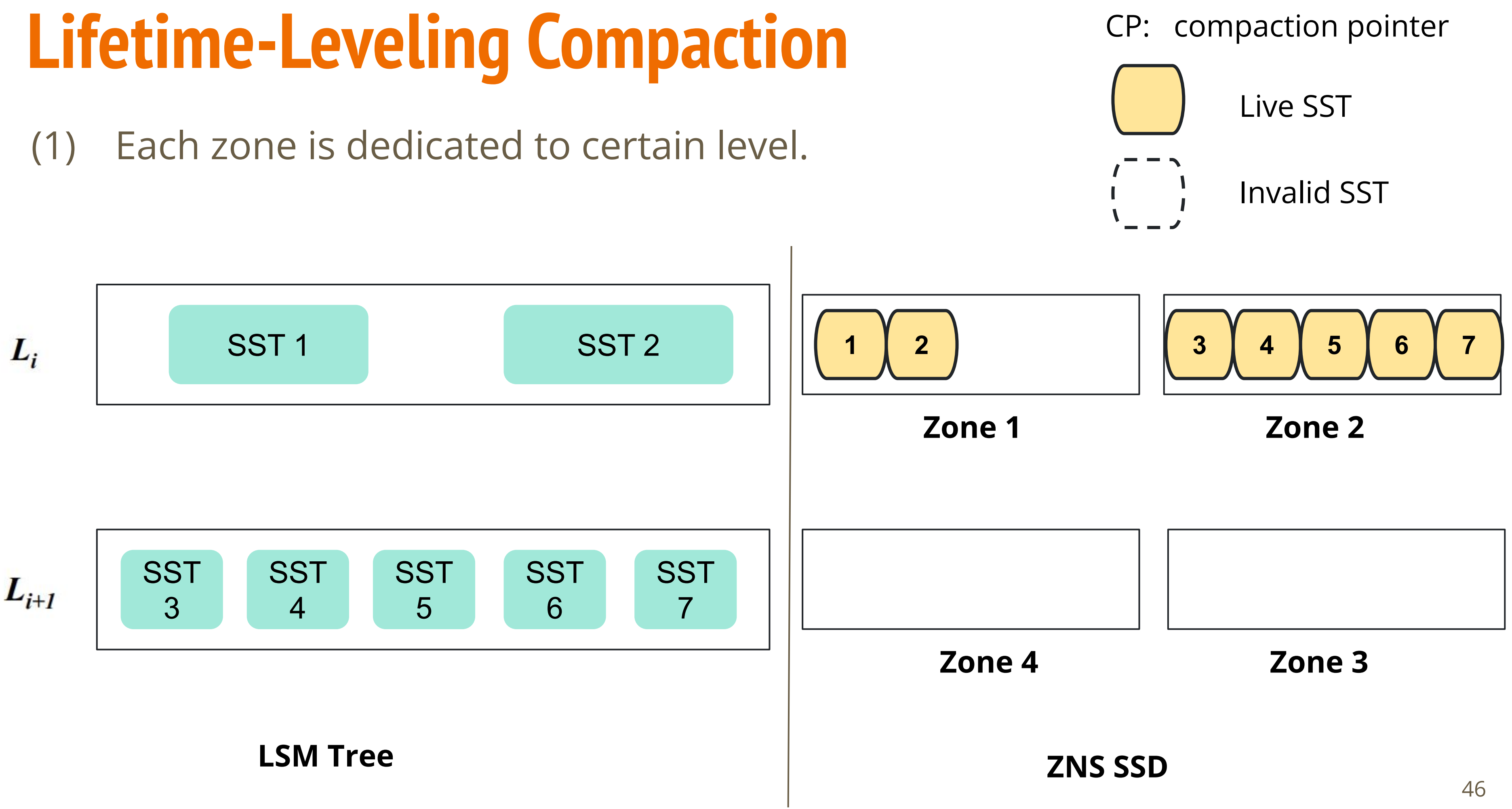
If we want to reduce the frequency of using GC:

1. **Reducing the number of short-lived SSTs.**
2. **Reducing the number of long-lived SSTs.**

⇨ **Lifetime-Leveling Compaction**

Lifetime-Leveling Compaction

(1) Each zone is dedicated to certain level.



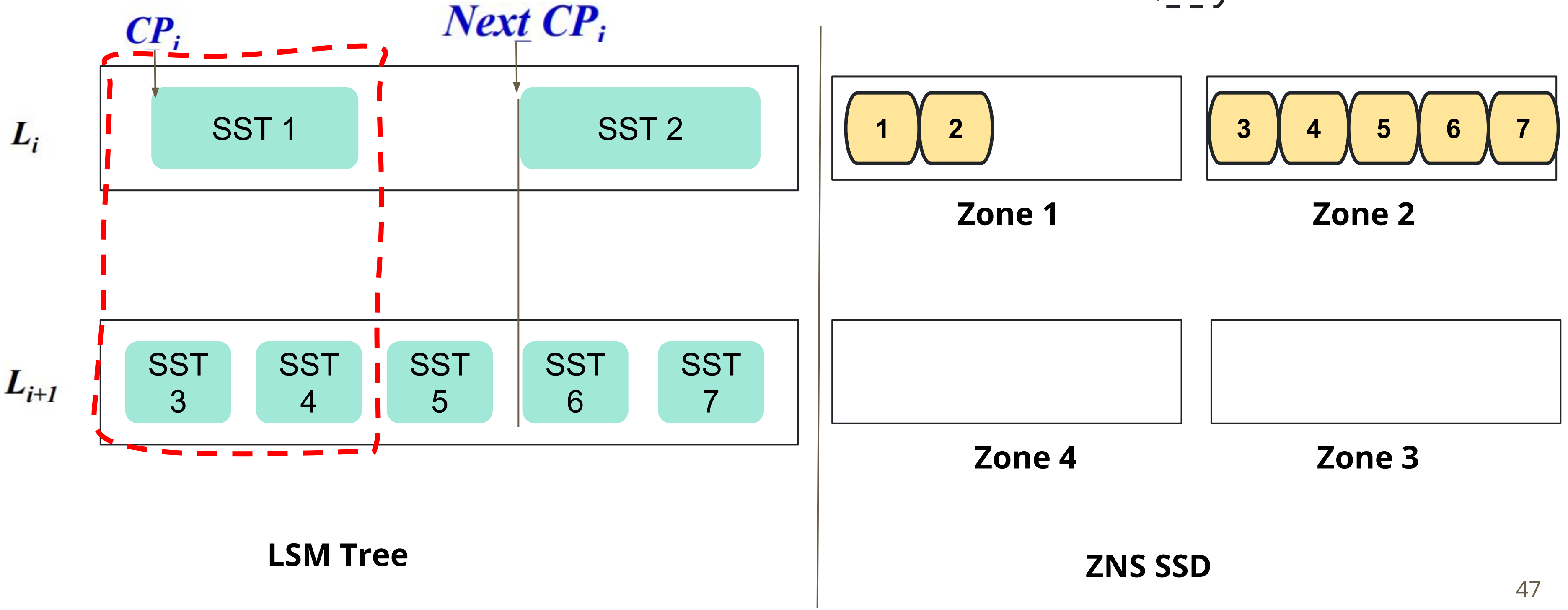
Lifetime-Leveling Compaction

CP: compaction pointer

Live SST

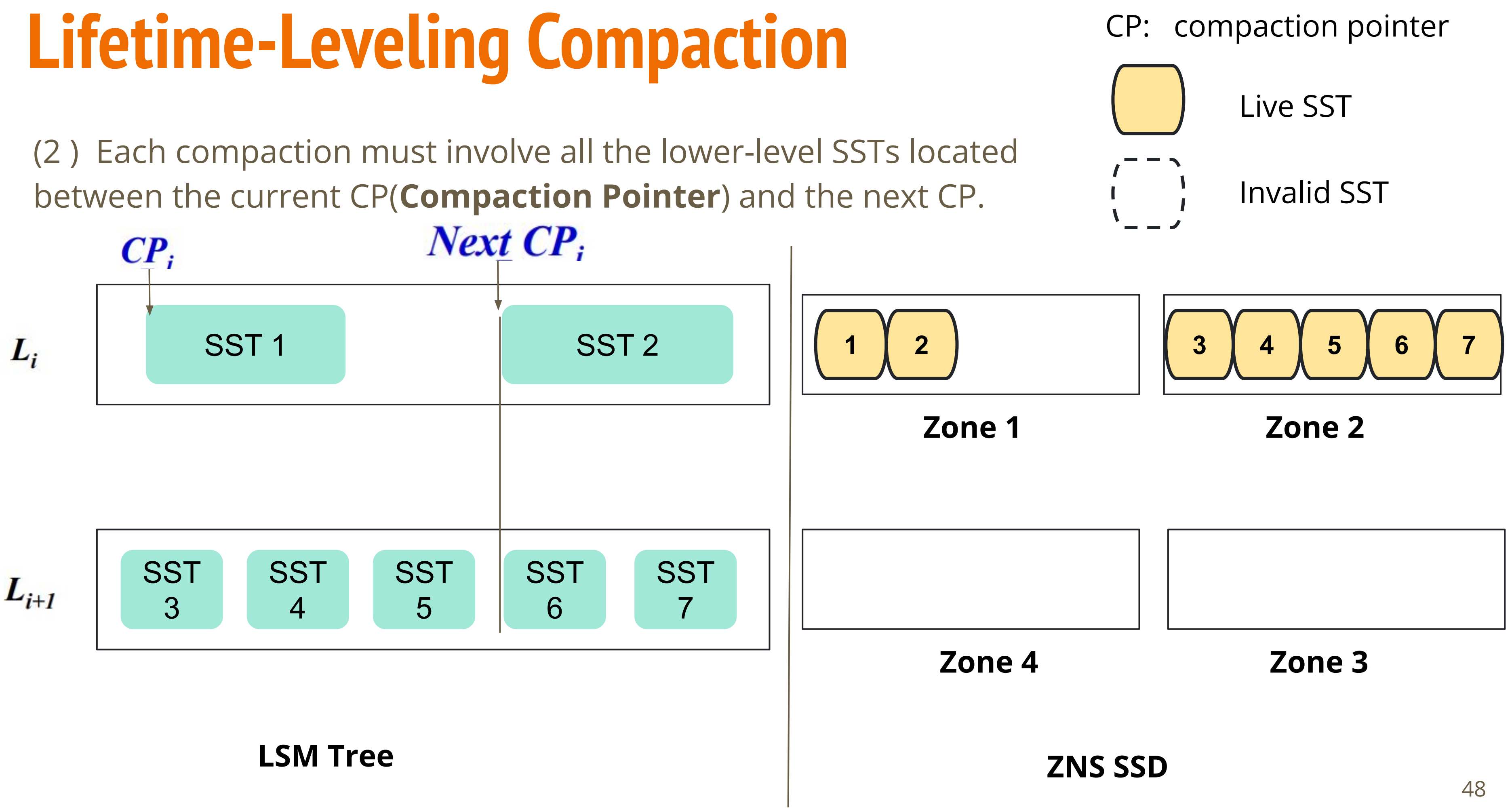
Invalid SST

(2) Each compaction must involve all the lower-level SSTs located between the current CP(**Compaction Pointer**) and the next CP.



Lifetime-Leveling Compaction

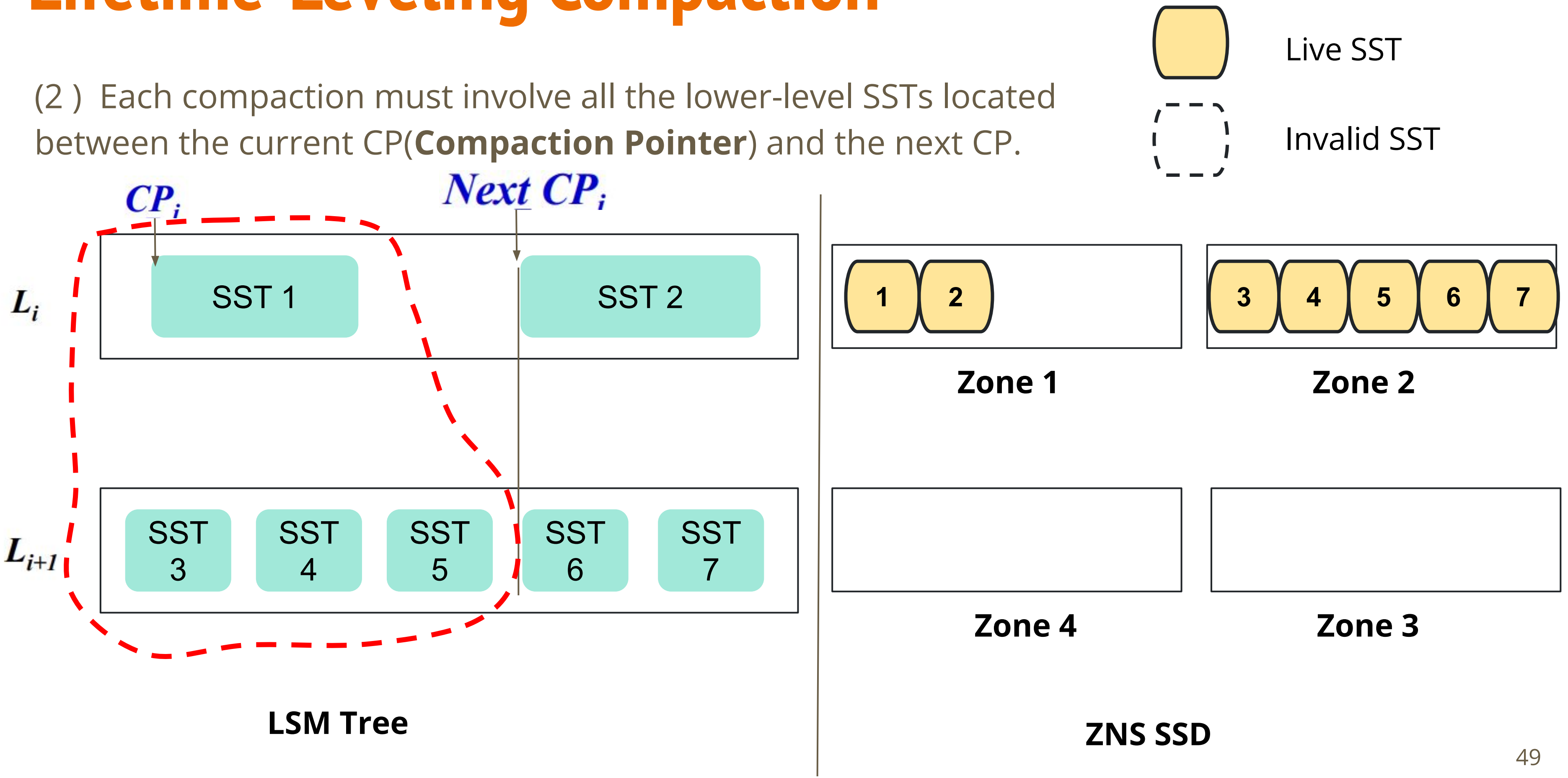
(2) Each compaction must involve all the lower-level SSTs located between the current CP(**Compaction Pointer**) and the next CP.



Lifetime-Leveling Compaction

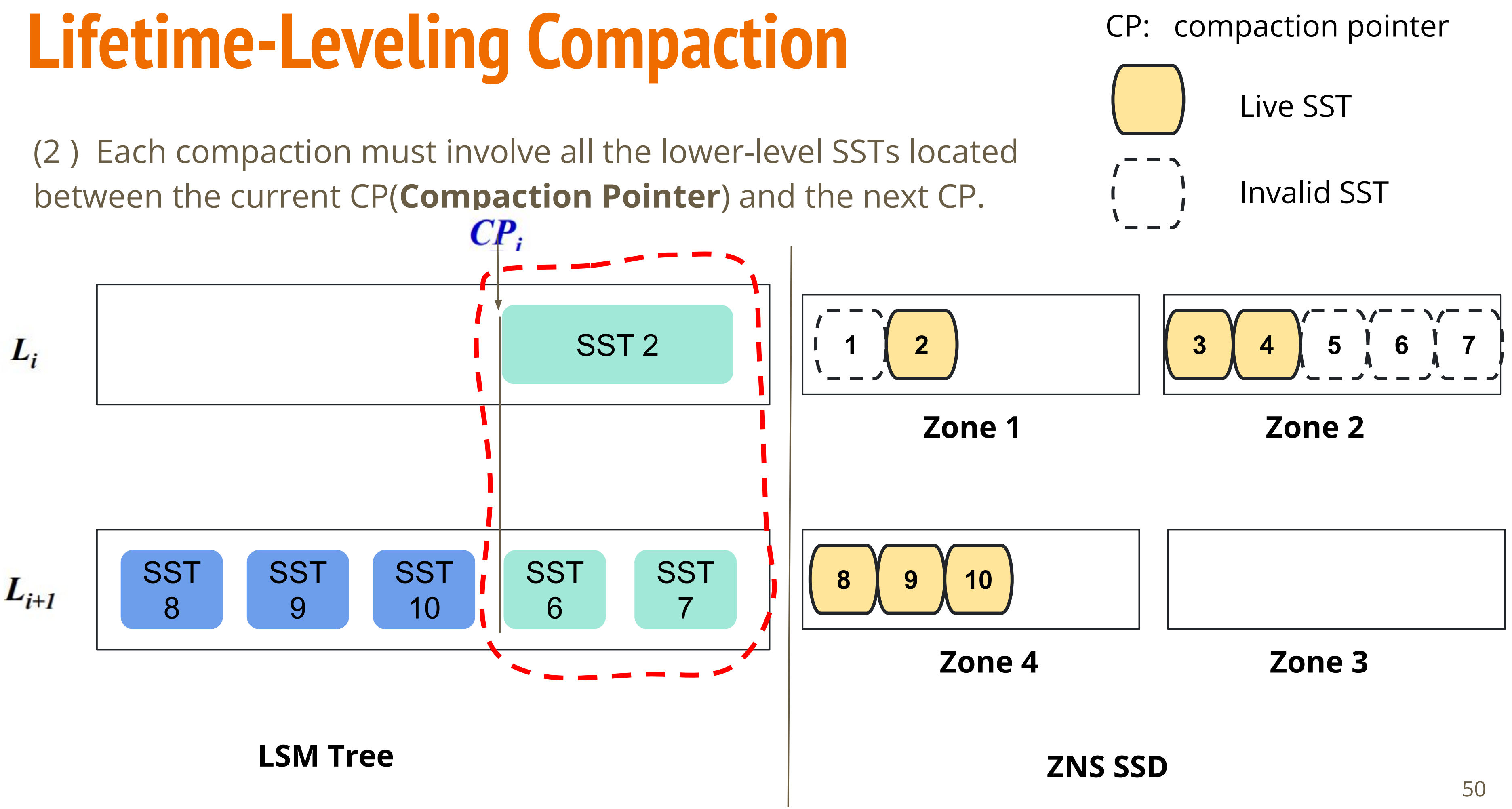
CP: compaction pointer

(2) Each compaction must involve all the lower-level SSTs located between the current CP(**Compaction Pointer**) and the next CP.



Lifetime-Leveling Compaction

(2) Each compaction must involve all the lower-level SSTs located between the current CP(**Compaction Pointer**) and the next CP.



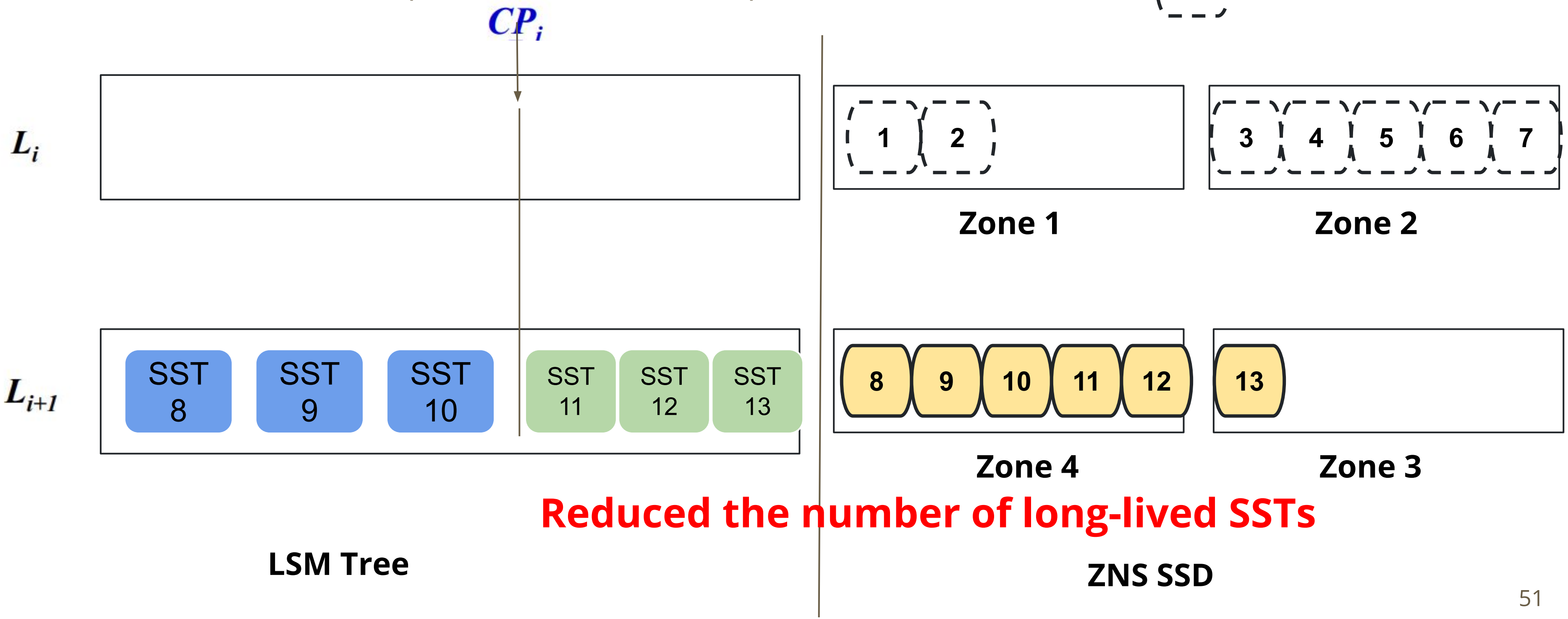
Lifetime-Leveling Compaction

CP: compaction pointer

Live SST

Invalid SST

(2) Each compaction must involve all the lower-level SSTs located between the current CP(**Compaction Pointer**) and the next CP.



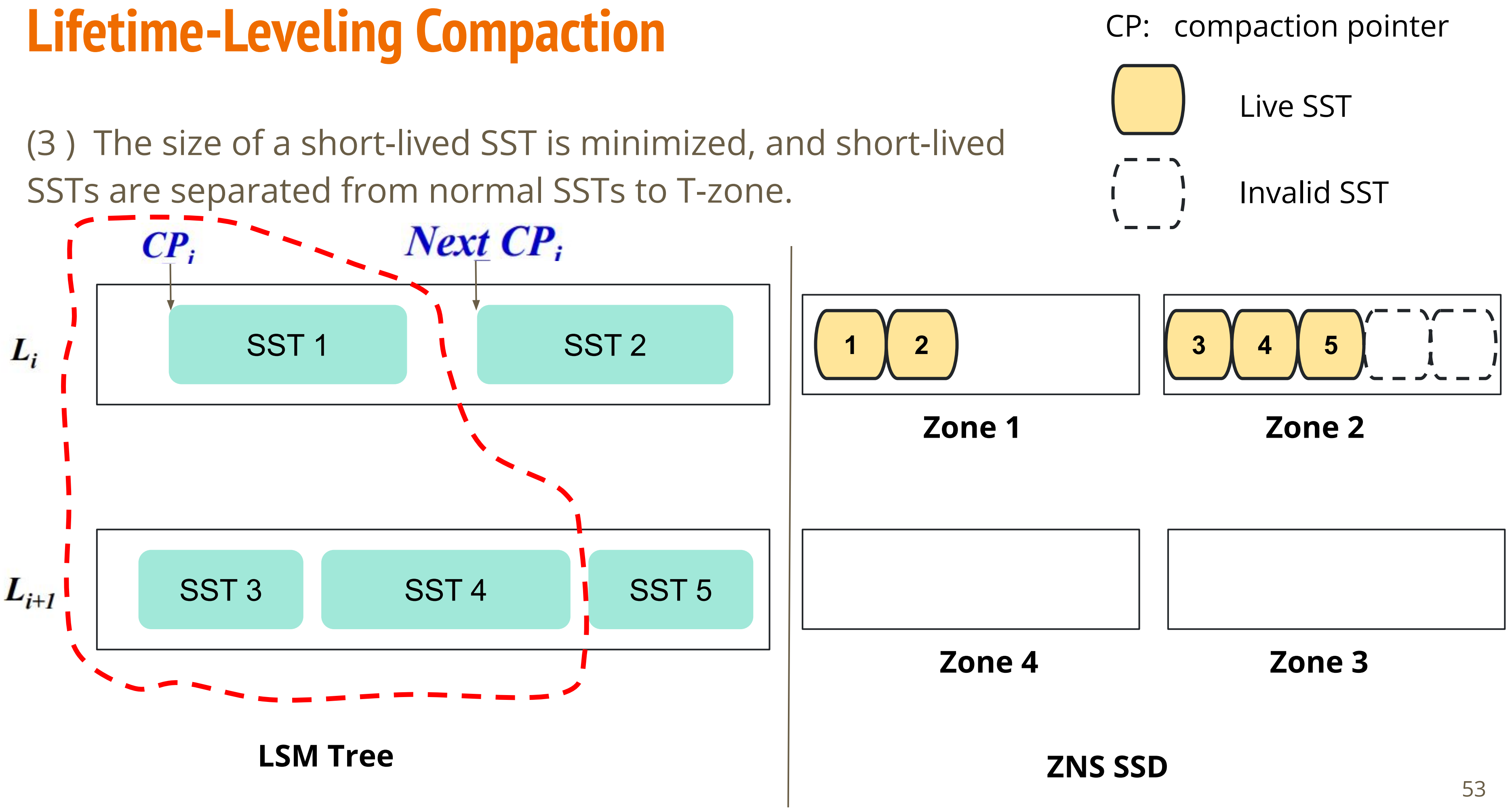
Reduced the number of long-lived SSTs

Pros and Cons?

	Long-lived SSTs	
Numbers	decrease	
Compaction Cost	increase	
Space Amplification	decrease	
Write Amplification	decrease	

Lifetime-Leveling Compaction

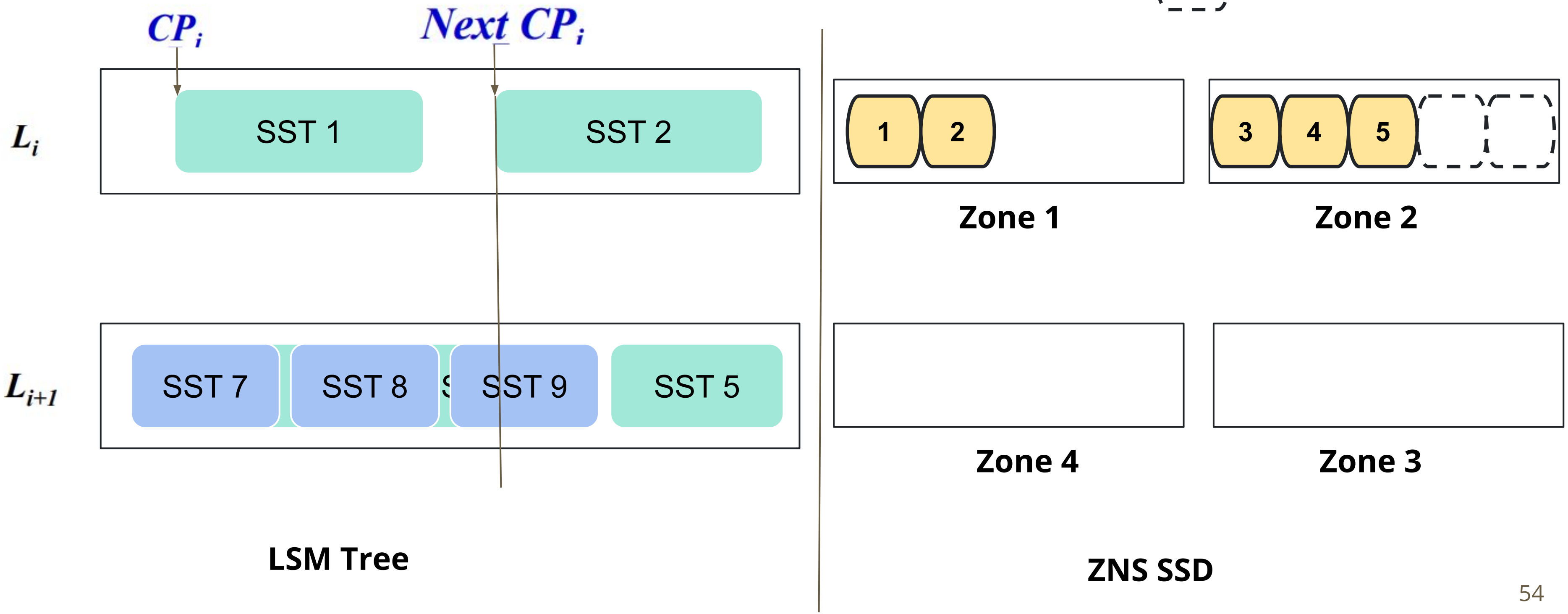
(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.



Lifetime-Leveling Compaction

CP: compaction pointer

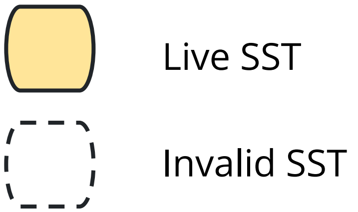
(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.

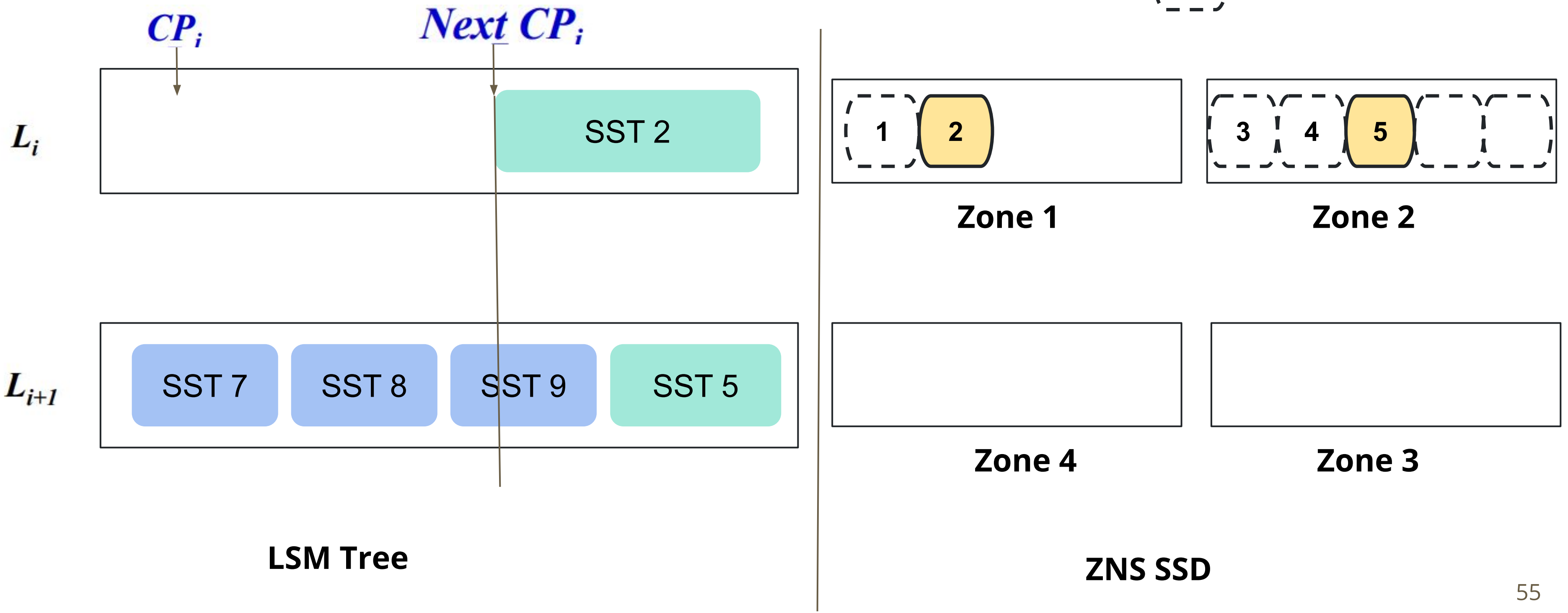


Lifetime-Leveling Compaction

CP: compaction pointer

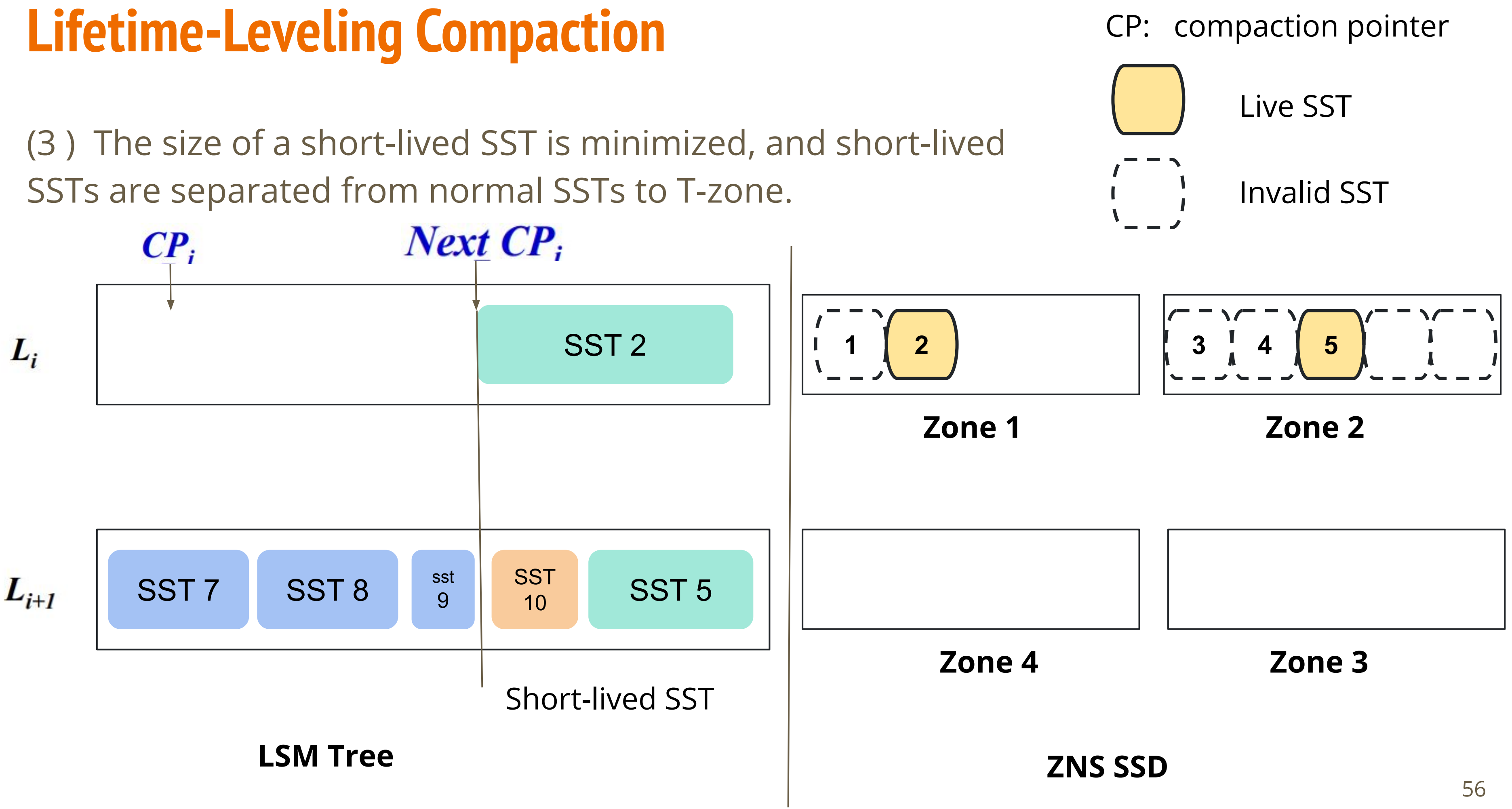
(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.





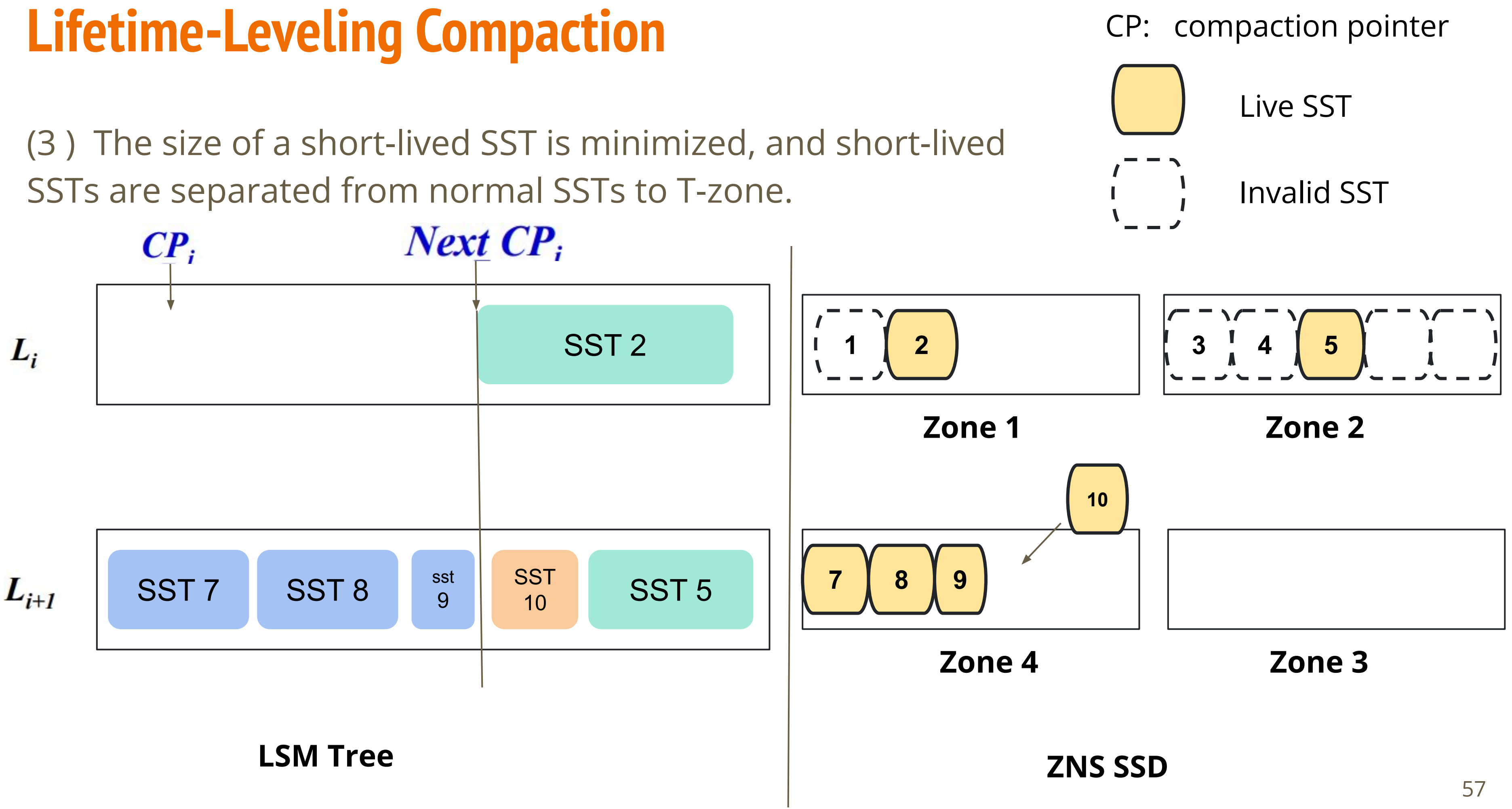
Lifetime-Leveling Compaction

(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.



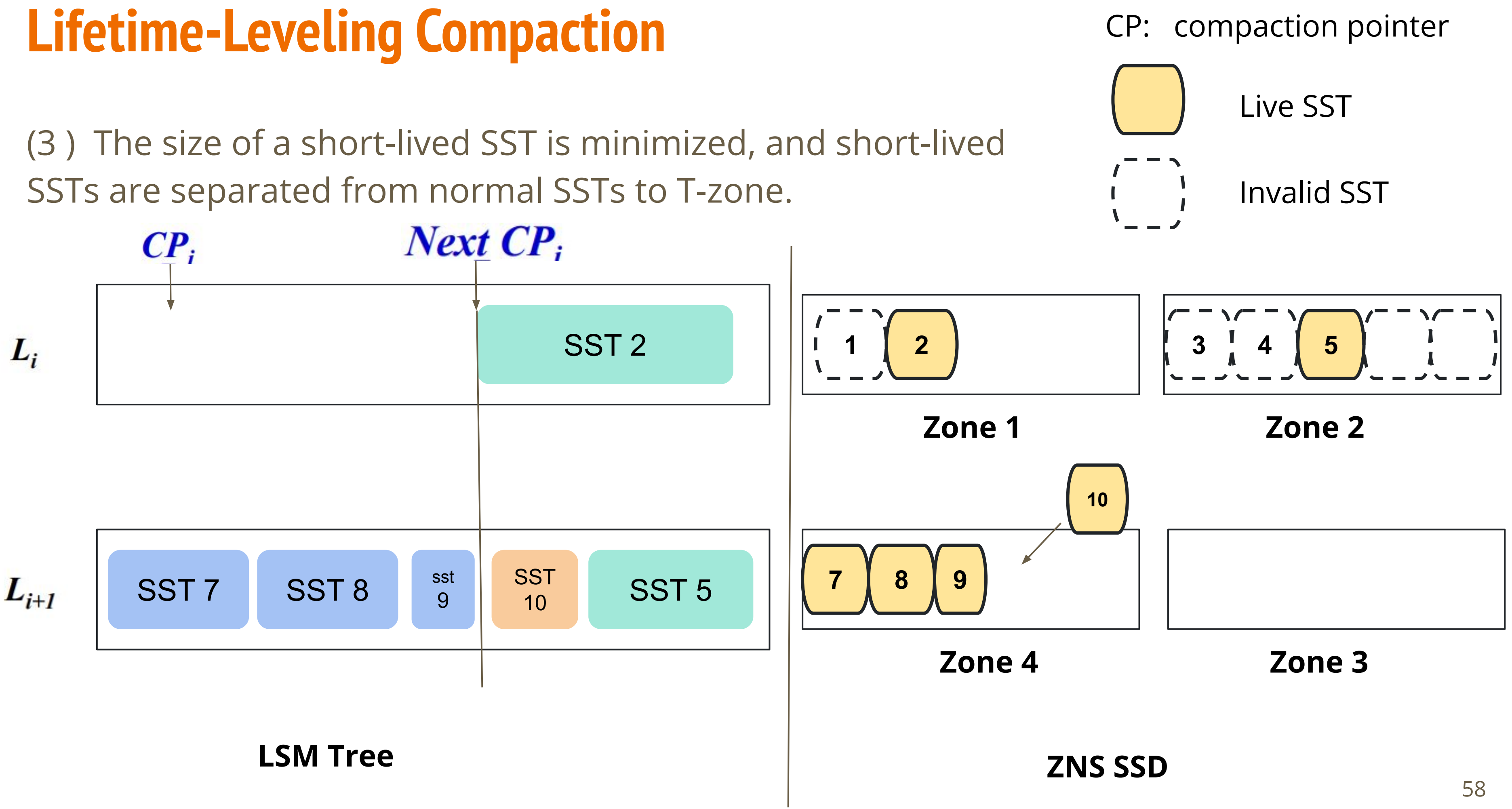
Lifetime-Leveling Compaction

(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.



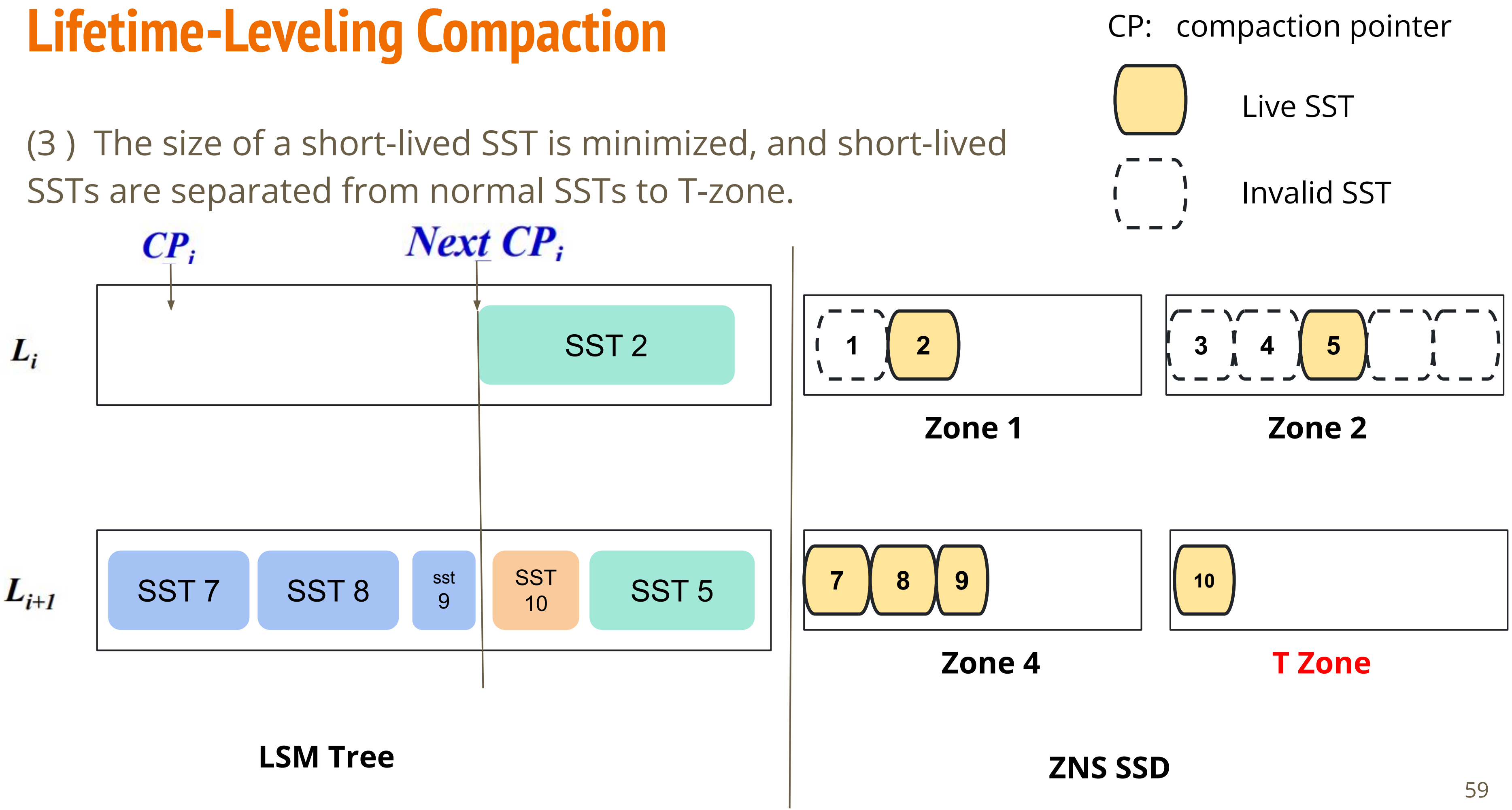
Lifetime-Leveling Compaction

(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.



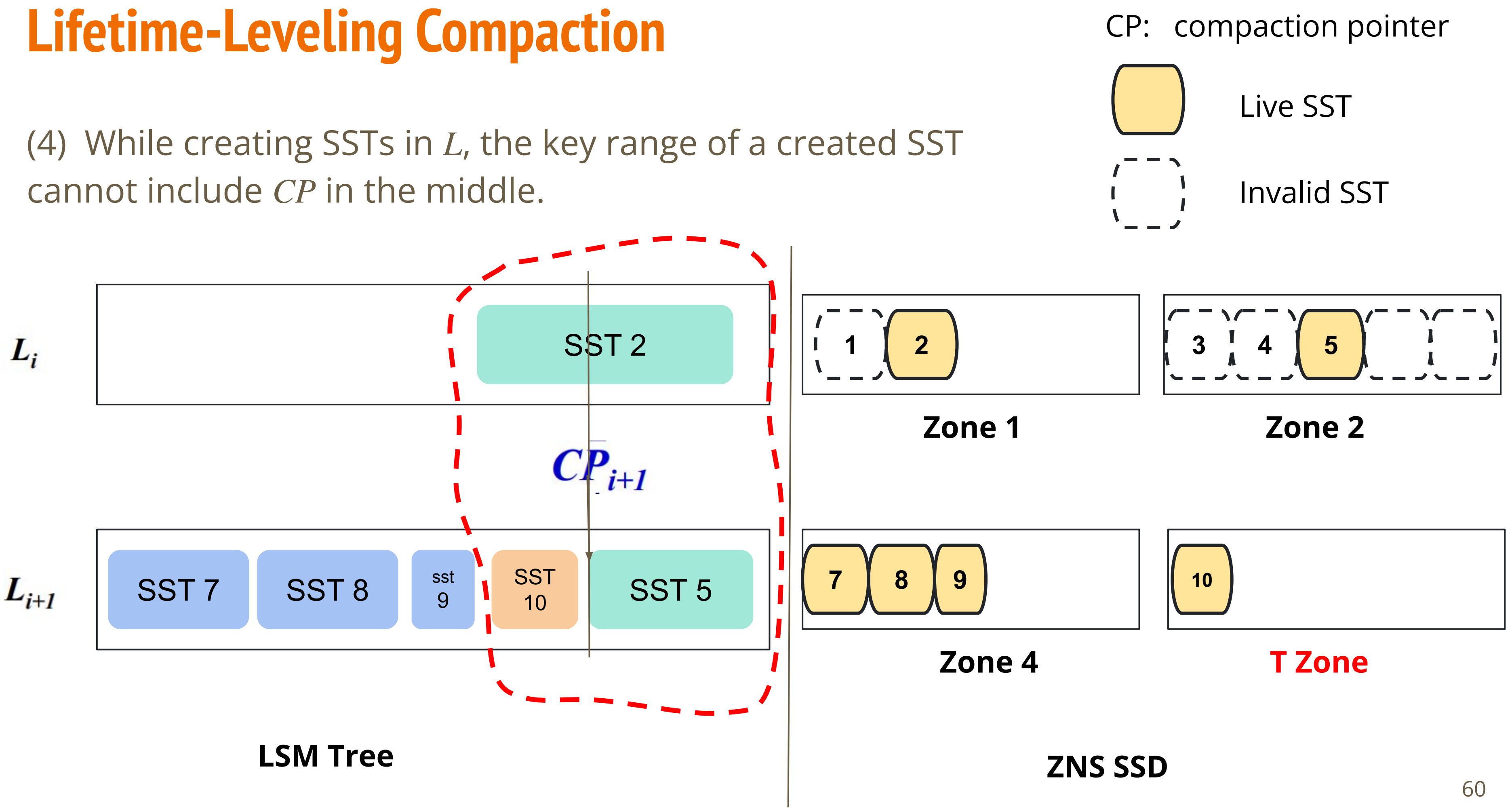
Lifetime-Leveling Compaction

(3) The size of a short-lived SST is minimized, and short-lived SSTs are separated from normal SSTs to T-zone.



Lifetime-Leveling Compaction

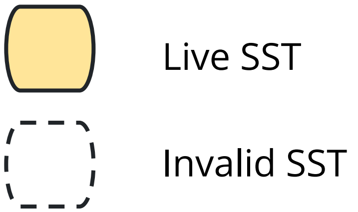
(4) While creating SSTs in L, the key range of a created SST cannot include CP in the middle.

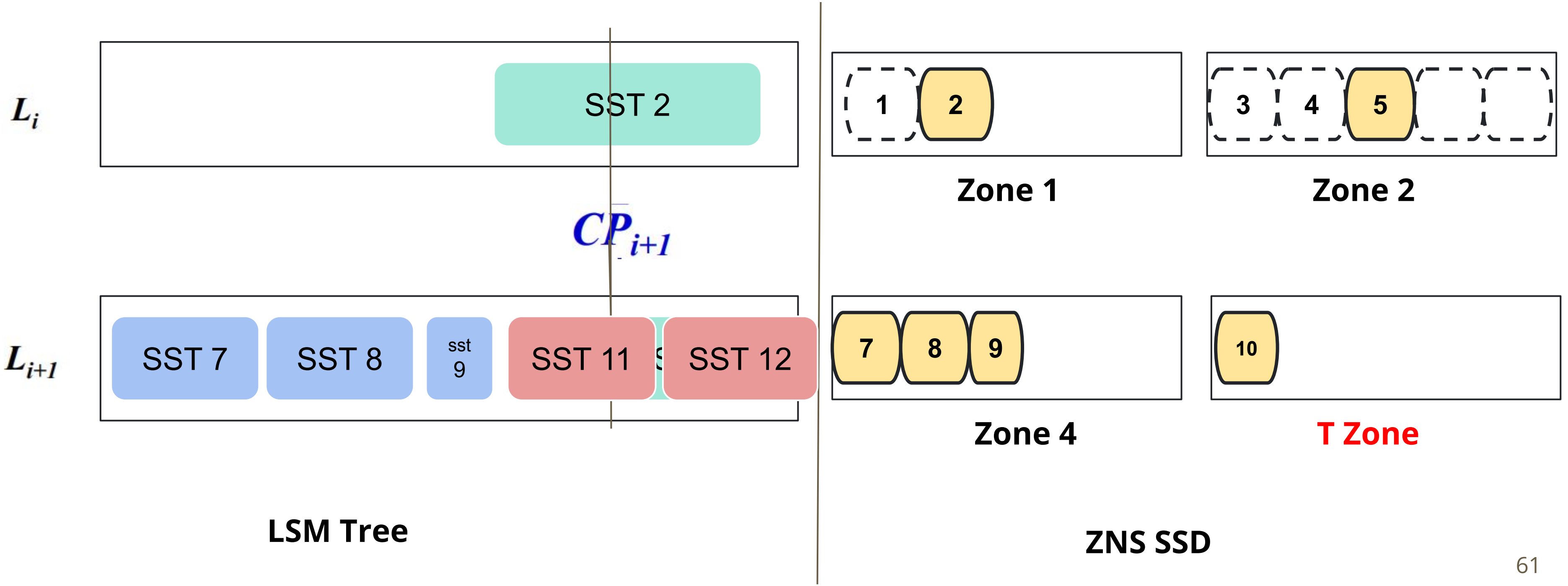


Lifetime-Leveling Compaction

(4) While creating SSTs in L, the key range of a created SST cannot include CP in the middle.

CP: compaction pointer

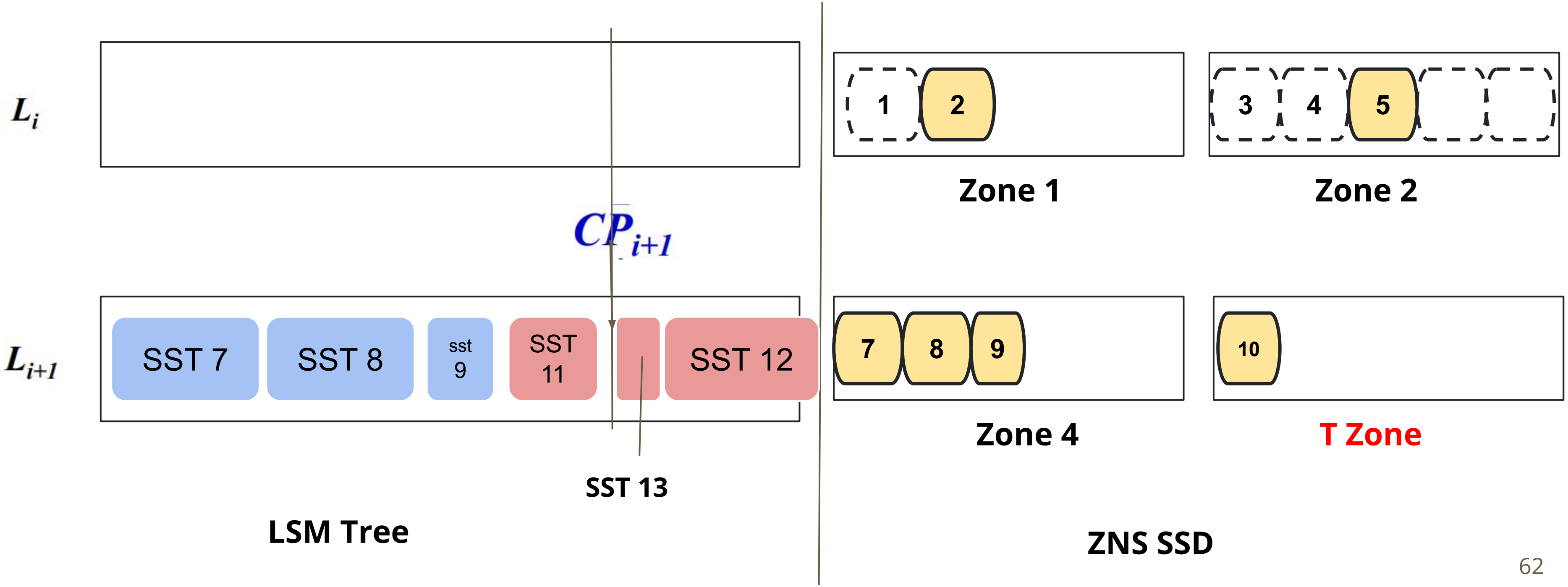




Lifetime-Leveling Compaction

CP: compaction pointer

(4) While creating SSTs in L, the key range of a created SST cannot include CP in the middle.



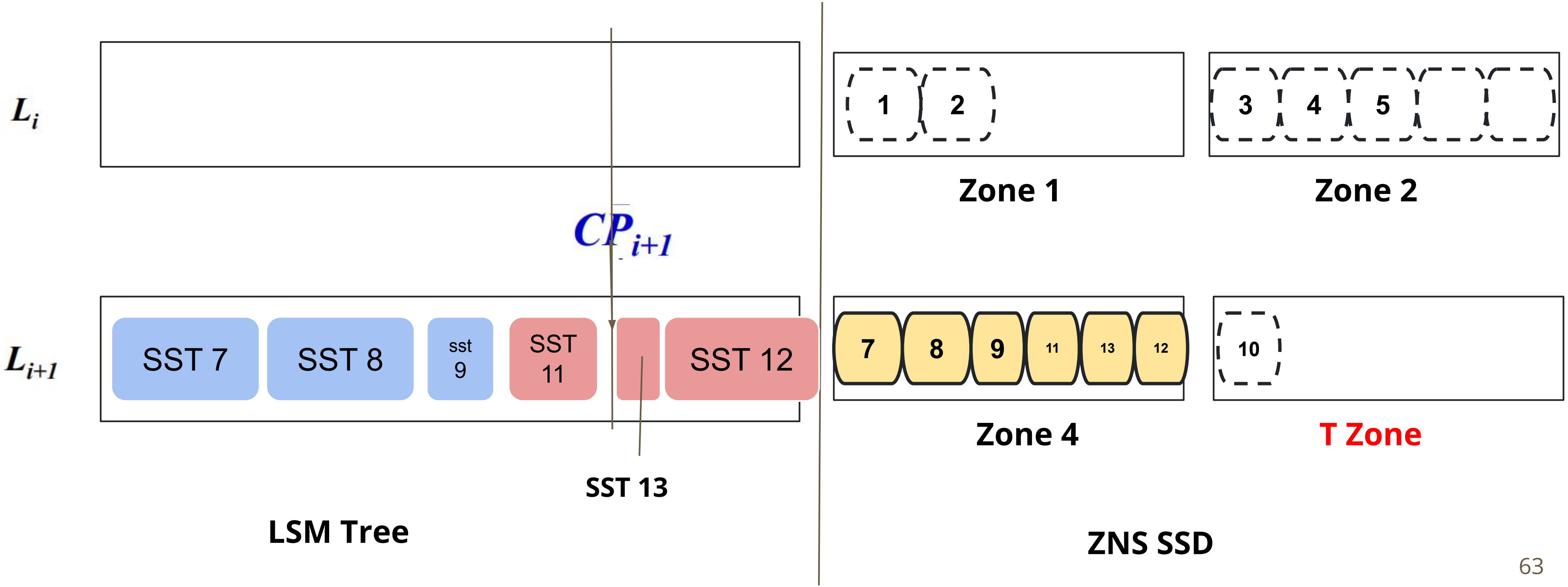
Lifetime-Leveling Compaction

CP: compaction pointer

Live SST

Invalid SST

(4) While creating SSTs in L, the key range of a created SST cannot include CP in the middle.



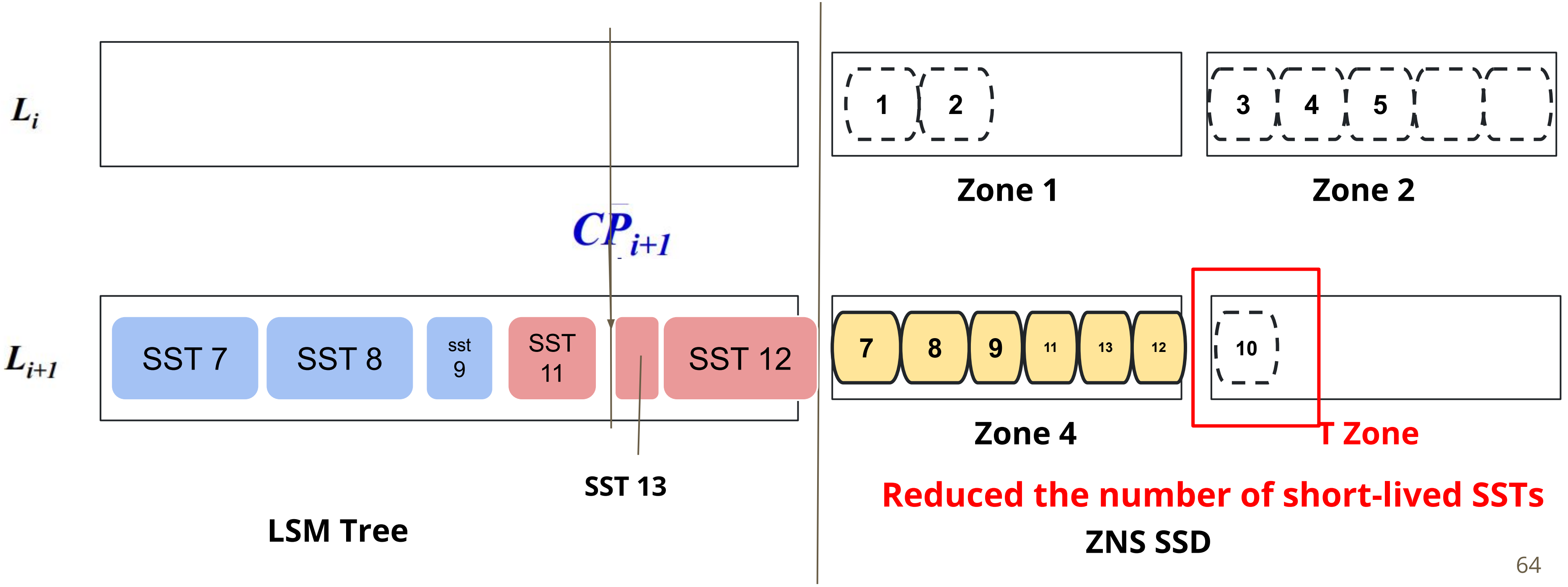
Lifetime-Leveling Compaction

CP: compaction pointer

Live SST

Invalid SST

(4) While creating SSTs in L, the key range of a created SST cannot include CP in the middle.



Pros and Cons?

	Long-lived SSTs	Short-lived SSTs
Numbers	decrease	decrease
Compaction Cost	increase	decrease
Space Amplification	decrease	decrease
Write Amplification	decrease	decrease

The amount of long-lived SSTs is generally small
=> 1.7x better performance by avoiding zone GCs

Evaluation

Evaluation

- Zone utilization
- Overall performance
- Write performance change
- Write breakdown

Testing Environment

Linux	64-bit Linux 5.11.1
CPU	4GHz quad-core Intel i7-4790K CPU
Memory	16GB DDR4, 1.5GB memory cache
ZNS	In-house ZNS SSD based on Cosmos+ OpenSSD
Default	LevelDB 1.19, key=16B, value=512B, SST size=4MB, GC=greedy, Zone=64MB

Comparisons

Compaction Technique:

BL	LevelDB
GC	LevelDB (GC-enable)
LS	LevelDB (GC-enable) + Level Separation
Gear	GearDB
LL (ours)	Lifetime Leveling Compaction

Gear Compaction

Pros:

- Automatically clean compaction windows during compaction
  - Eliminate garbage collection (GC)

Cons:

- Recursive compacting algorithm
  - Higher writing cost

ALGORITHM 1: Gear Compaction Algorithm

Input: V_i: victim SSTable in L_i

do
    DoGearComp $\leftarrow$ false;
    $O_{i+1} \leftarrow$ GetOverlaps (V_i); */*O_{i+1}: overlapped SSTables in L_{i+1}'s compaction window*/*
    result $\leftarrow$ merge-sort(V_i, O_{i+1});
    iter.key $\leftarrow$ MakeInputIterator(result);
    for *iter.first to iter.end* **do**
        if *key In_CW* L_{i+2} **then**
            write to buffer; */*wait in memory for the passive compaction*/*
        else
            if *key Out_CW* L_{i+2} **then**
                write to L_{i+1};
            else
                if *key Out* L_{i+2} **then**
                    write to L_{i+2};
                end
            end
        end
    end
    if *buffer* $!=$ *Null* **then**
        i++;
        $V_i \leftarrow$GetVictims(buffer);
        DoGearComp $\leftarrow$ true;
    end
while *DoGearComp* $==$ *true*;

Workload

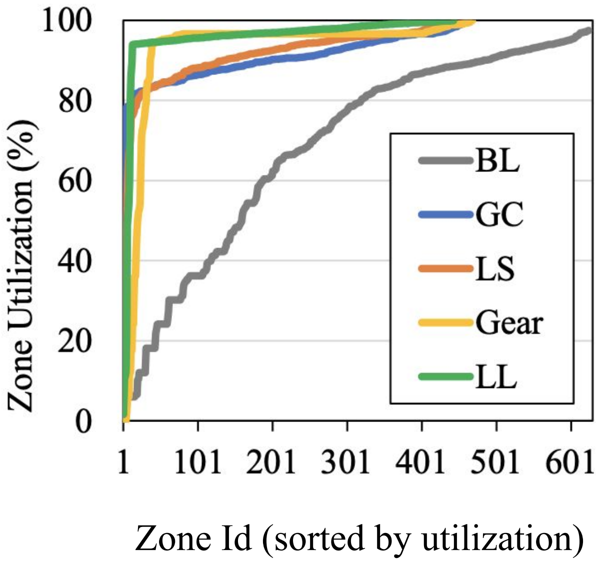
Workload:

- Fill-random benchmark of db_bench
- Key size: 16B
- Value size: 512B
- The total 27 GB of data were written by the workload.

Zone Utilization (Fill-random Workload)

Utilization of LL:

- Higher than **90%** at most zones

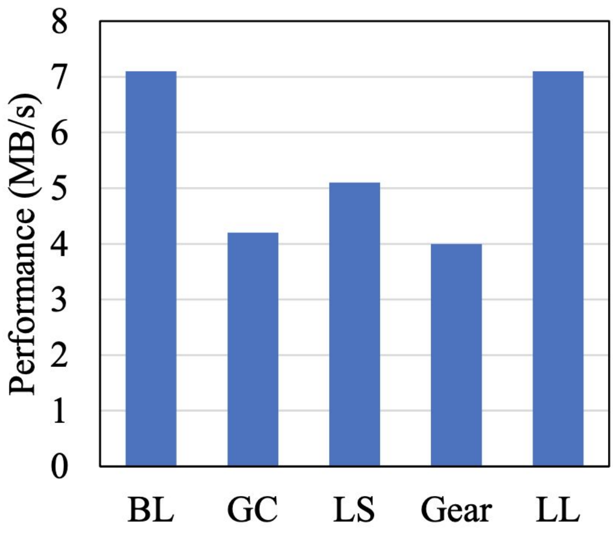


BL	LevelDB
GC	LevelDB (GC-enable)
LS	LevelDB (GC-enable) + Level Separation
Gear	GearDB
LL	Lifetime Leveling Compaction

Overall Performance (Fill-random Workload)

Performance of LL:

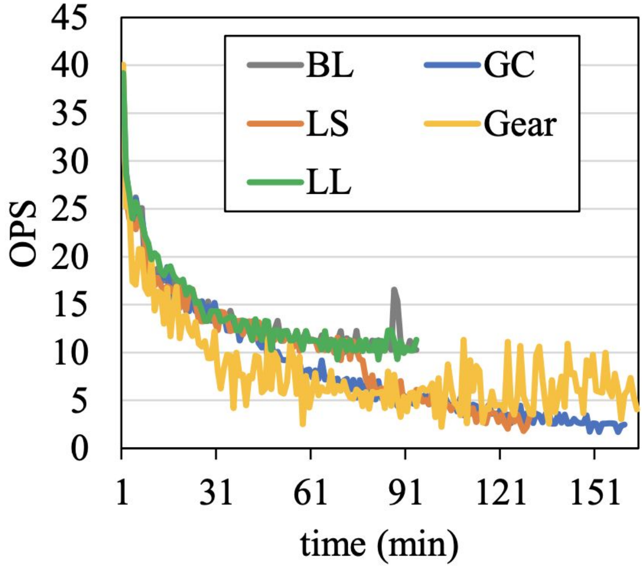
- **1.4X** speed up vs LS
- **1.8X** speed up vs Gear



BL	LevelDB
GC	LevelDB (GC-enable)
LS	LevelDB (GC-enable) + Level Separation
Gear	GearDB
LL	Lifetime Leveling Compaction

Write Performance Change (Fill-random Workload)

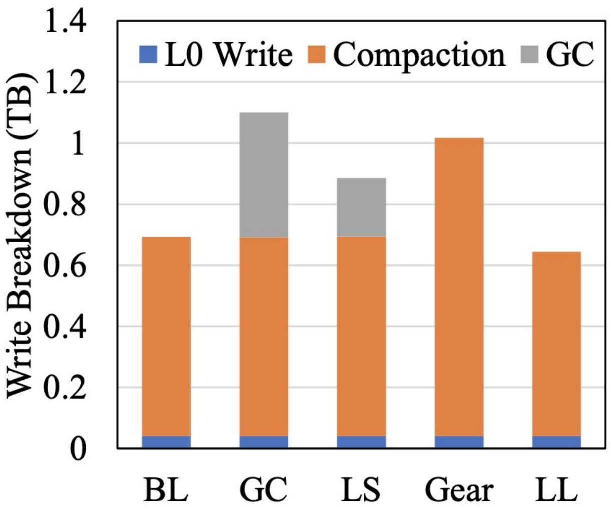
- BL & LL achieve stable and fast performance
- GC & LS degrade when garbage collection(GC) enabled
- Gear is unstable



BL	LevelDB
GC	LevelDB (GC-enable)
LS	LevelDB (GC-enable) + Level Separation
Gear	GearDB
LL	Lifetime Leveling Compaction

Write Breakdown (Fill-random Workload)

- GC-enable cause GC, LS write amplification
- LL achieves **less write cost** vs BL



BL	LevelDB
GC	LevelDB (GC-enable)
LS	LevelDB (GC-enable) + Level Separation
Gear	GearDB
LL	Lifetime Leveling Compaction

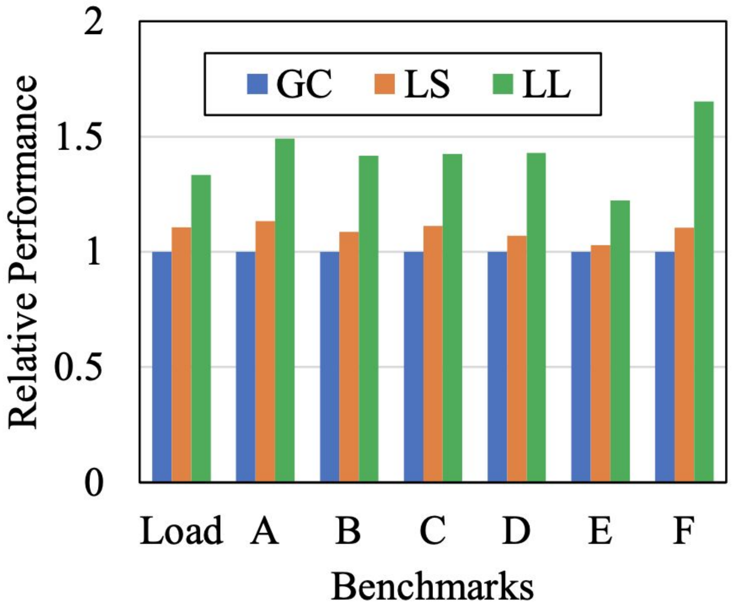
Performance (YCSB Workload)

YCSB (Yahoo! Cloud Serving Benchmark):

- Simulate real-world database usage

Performance of LL:

- **1.22~1.65X** speed up vs GC



Conclusion

- Current ZNS compaction suffers from **space amplification**
- Benefit of LL-Compaction:
  - Reduce space amplification without zone garbage collection(GC)
  - Performance: **1.22~1.65X speed up** vs GC
  - Utilization: **90%+** at most zones
- Limitation of LL:
  - Cannot use priority driven compaction algorithms (e.g., RocksDB)
- Future work:
  - Analyze the impact of priority-driven compaction algorithms on GC