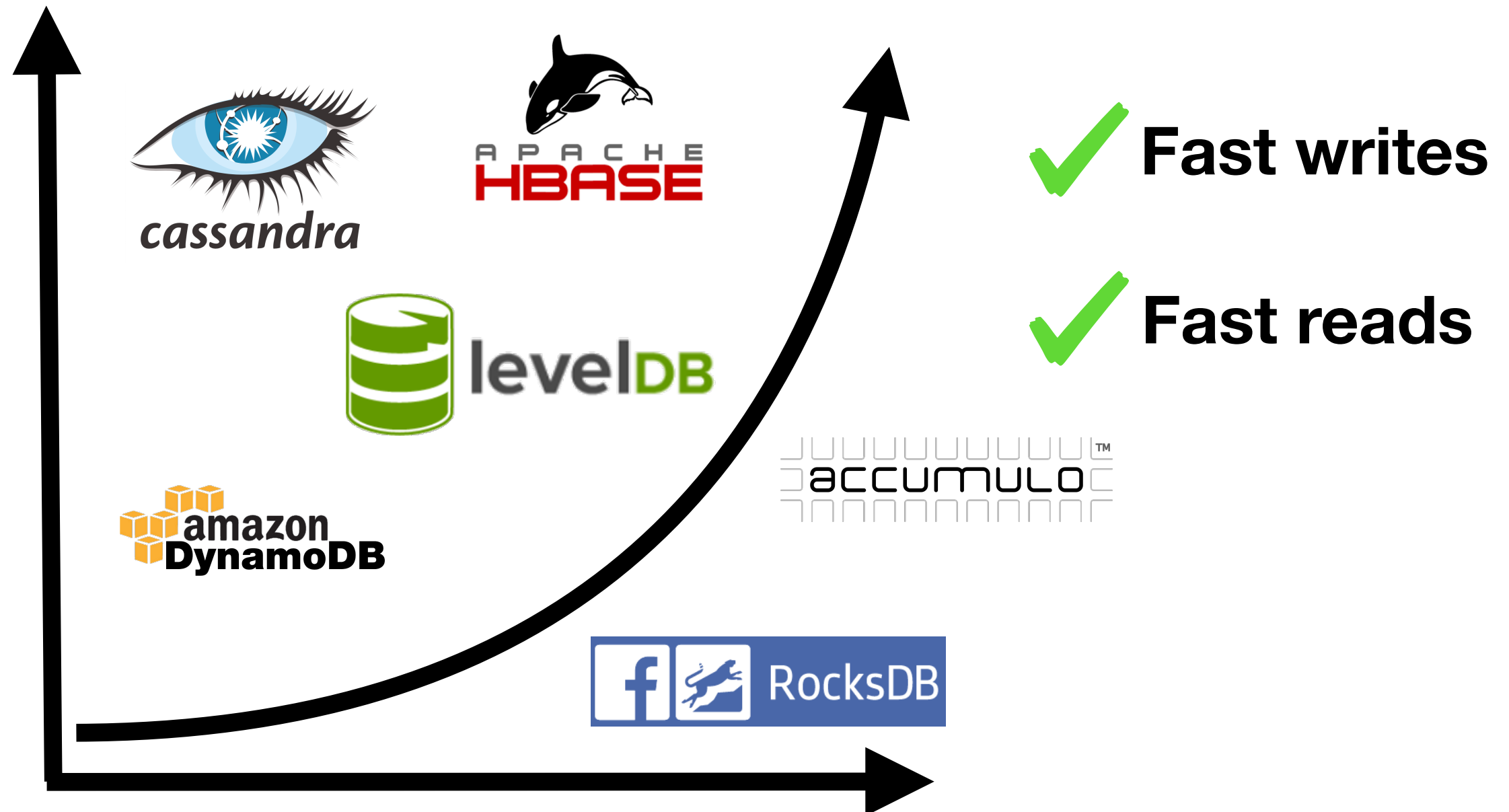


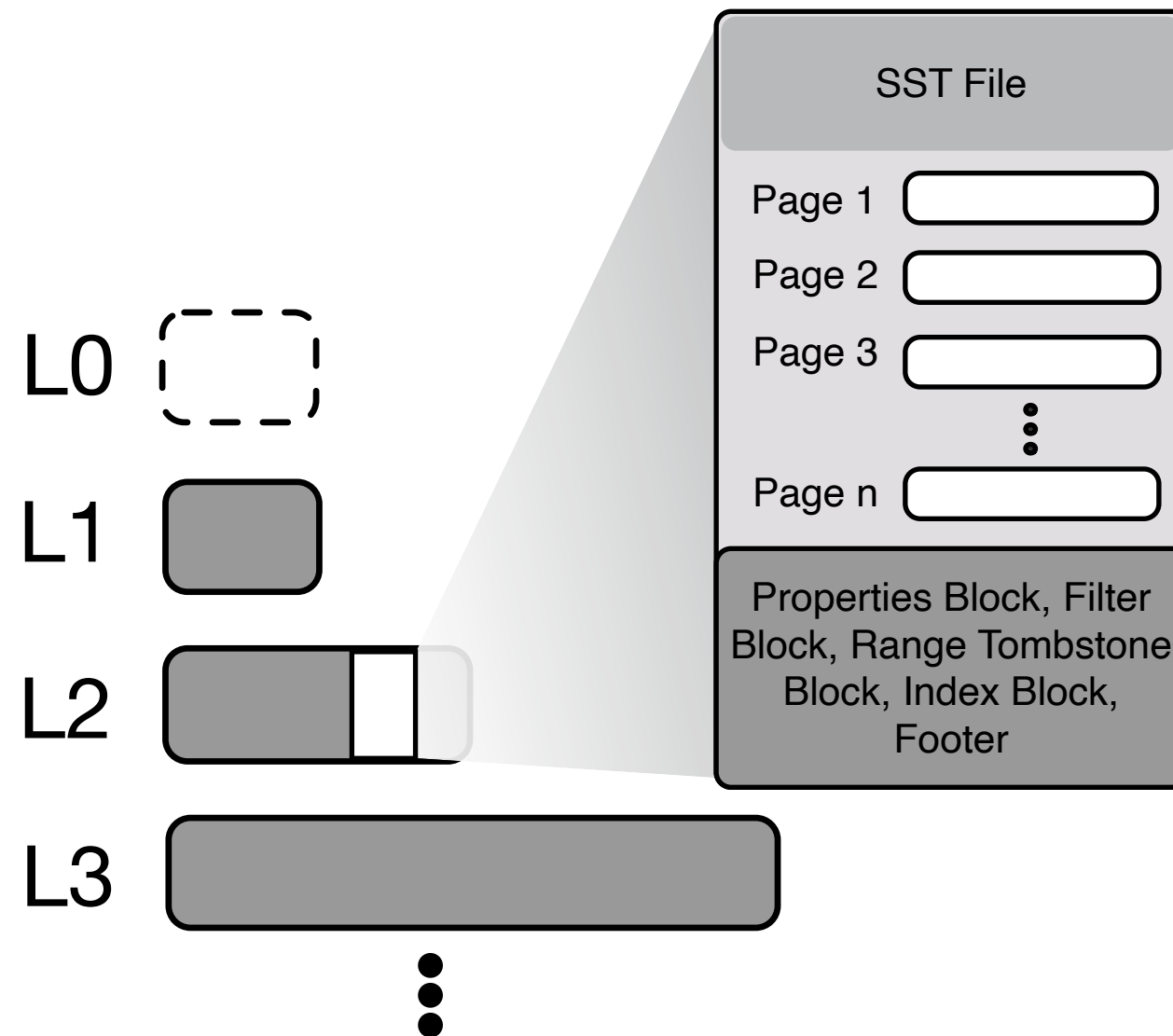
# Efficient Deletes in LSM Engines

Dimitris Staratzis

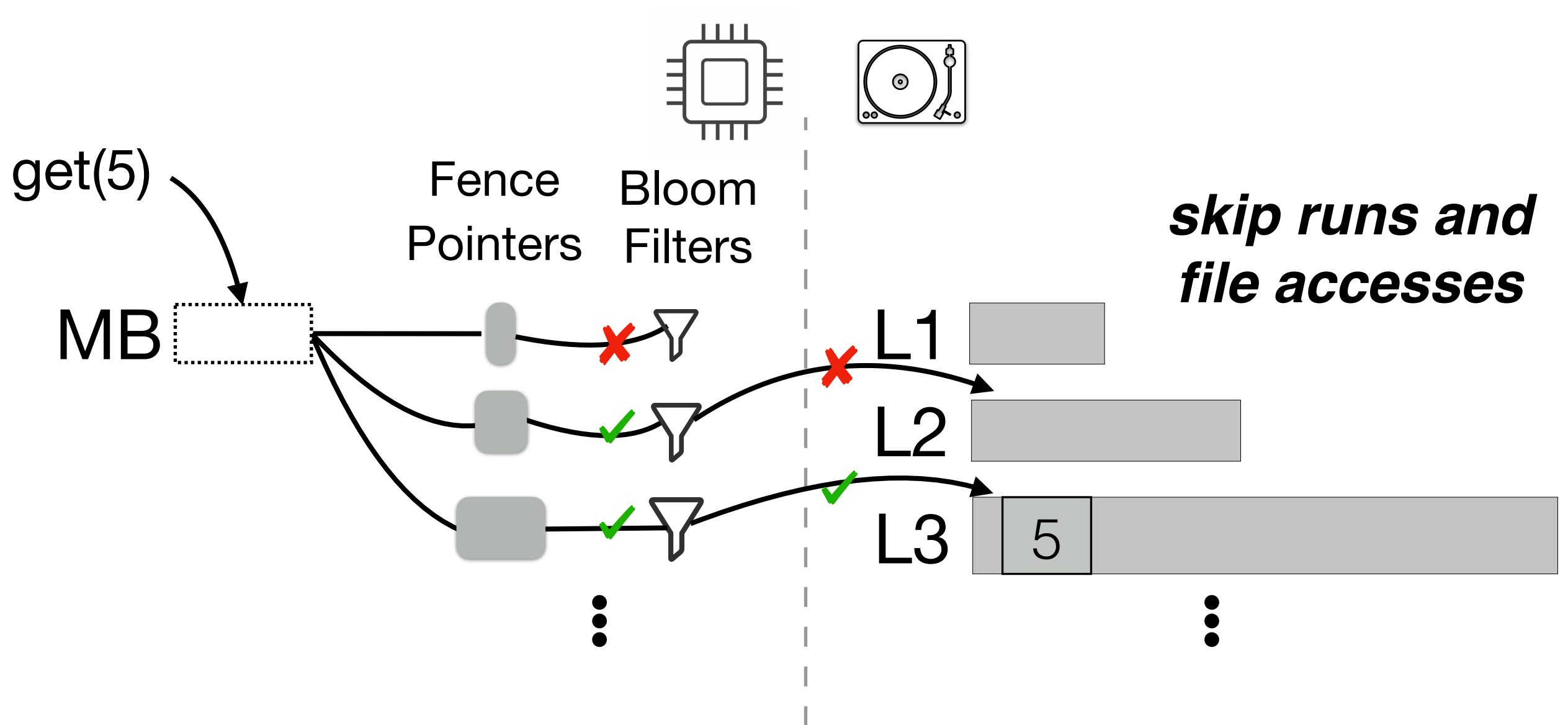
# LSM - trees are everywhere



# RocksDB structure



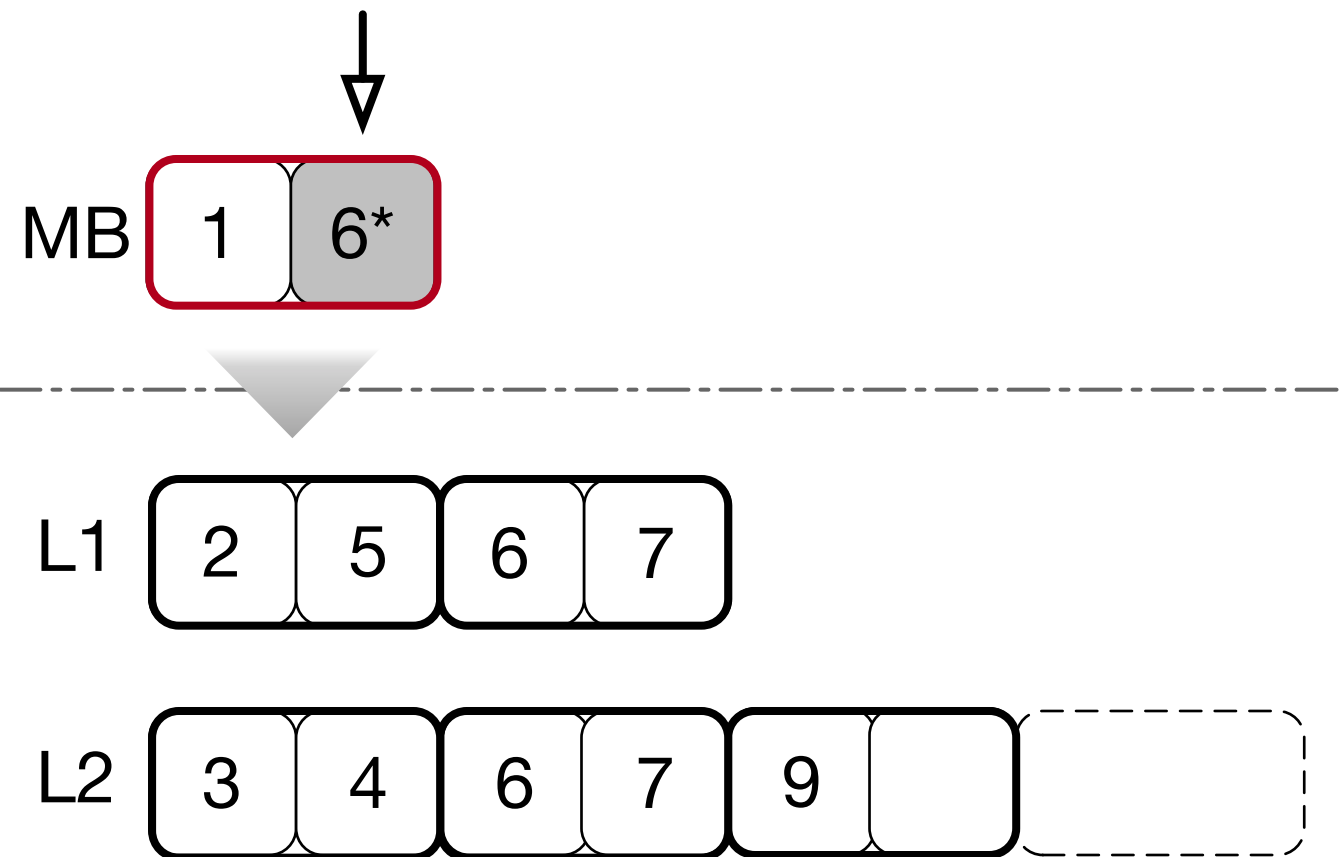
# Read-Path



# How to achieve fast deletes?

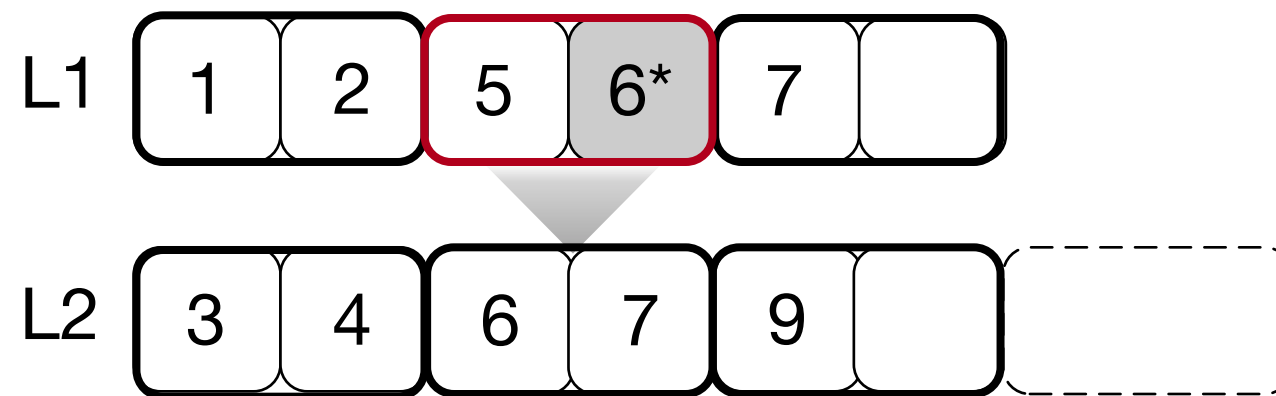
Out-of place!

# Deletes



# Deletes

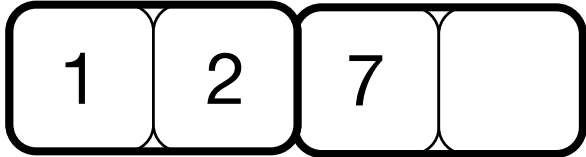
MB




# Deletes

MB 

---

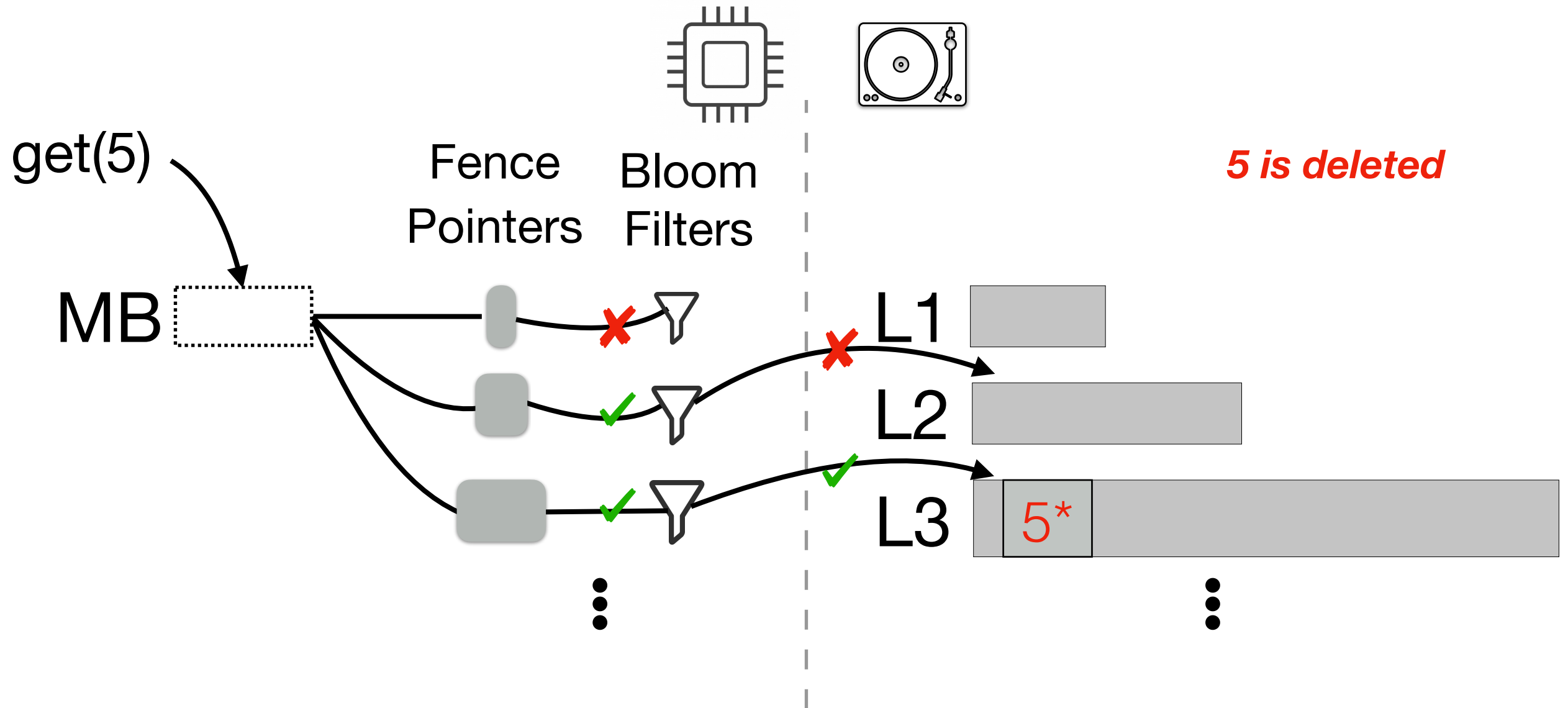
L1 

L2 

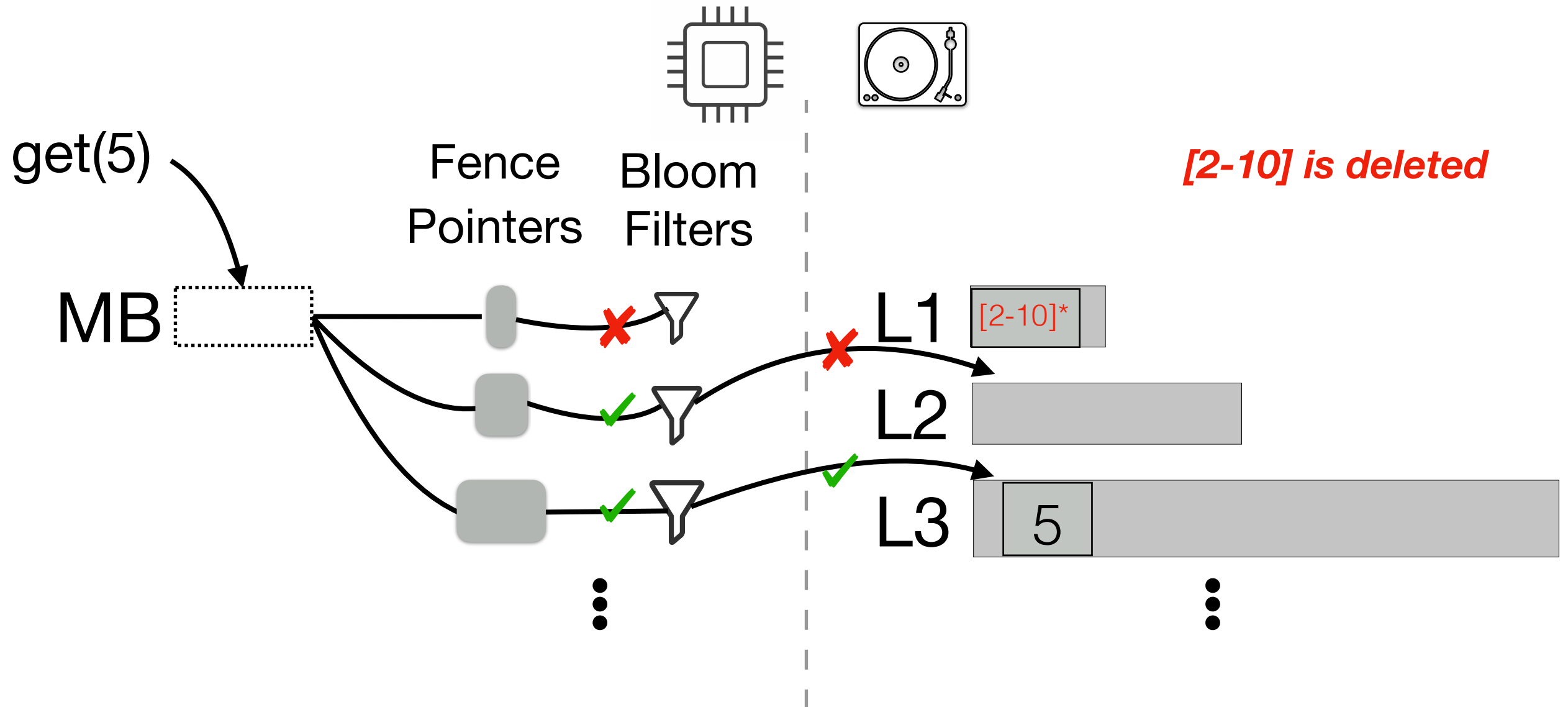
***Same process for both range and point deletes***



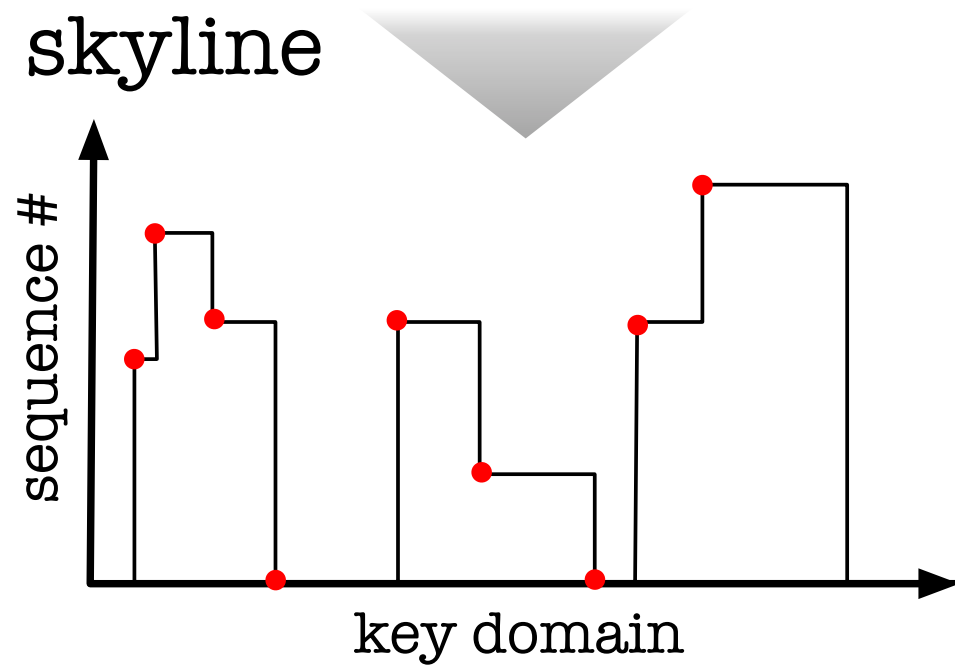
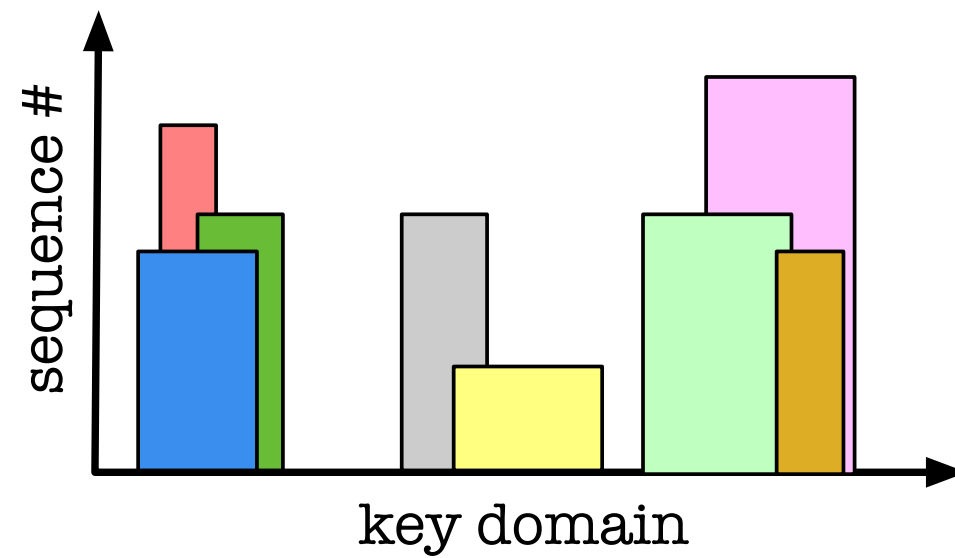
# Read-Path for Point Deletes



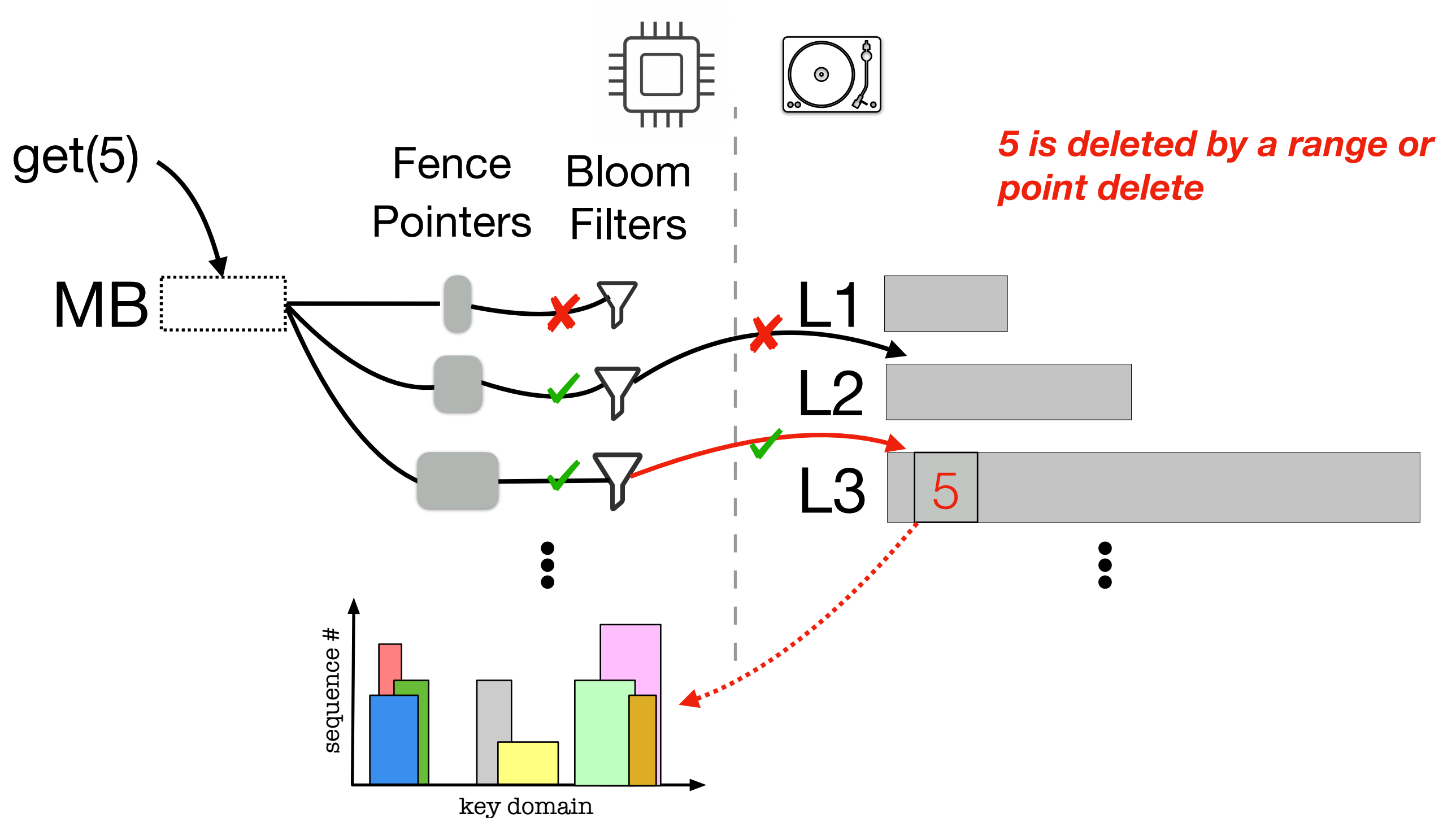
# Read-Path for Range Deletes



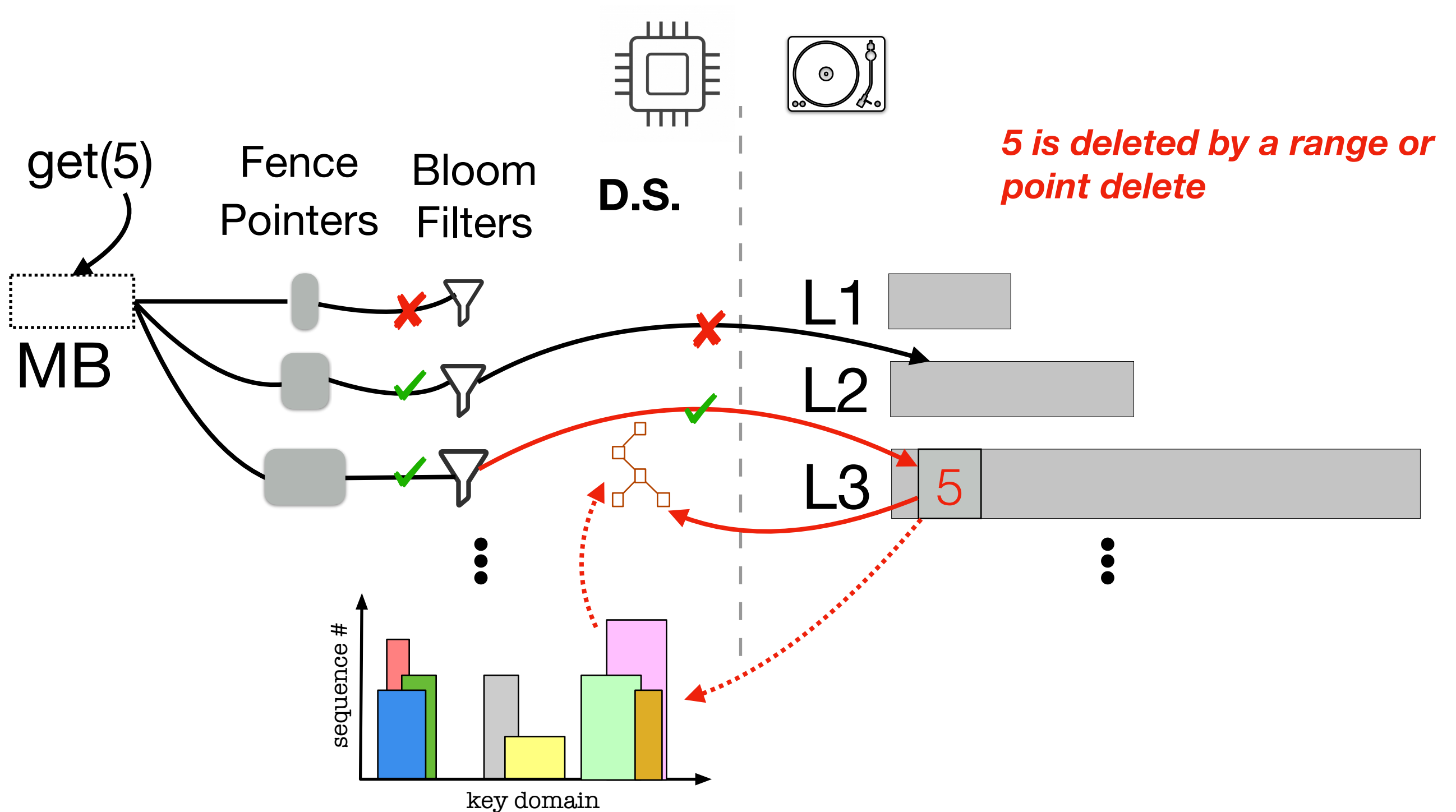
# The Skyline



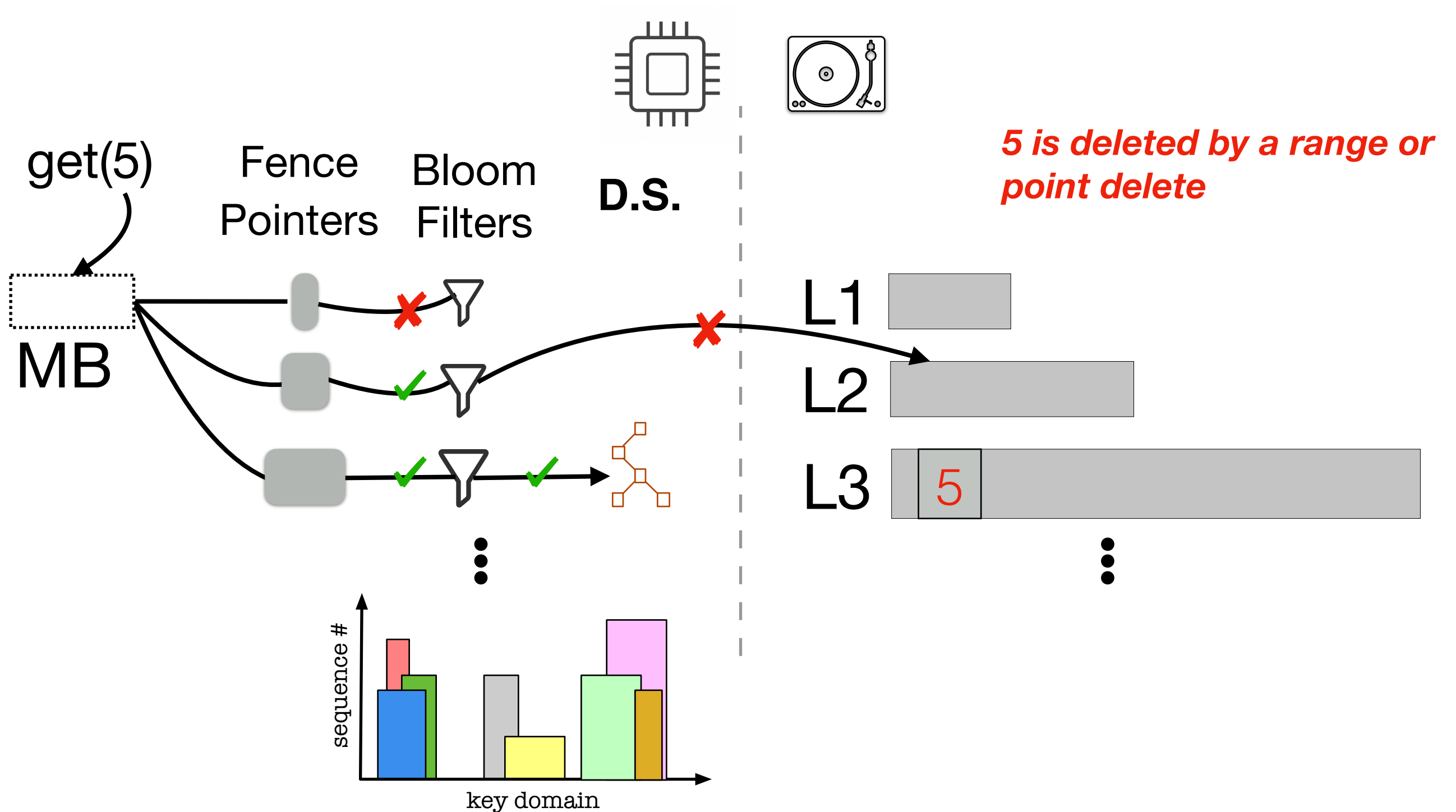
# Read - Path Overview



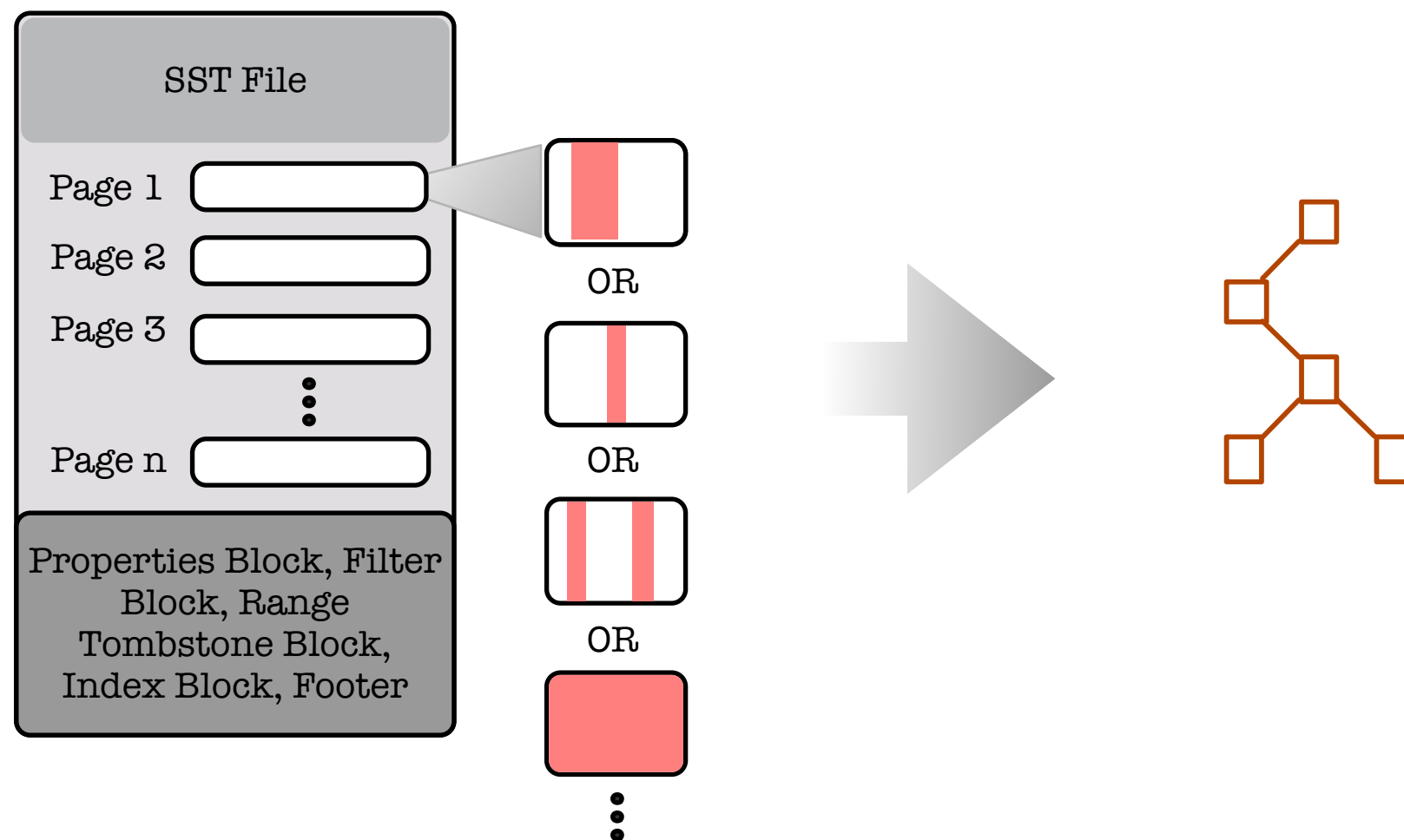
# Proposed solution



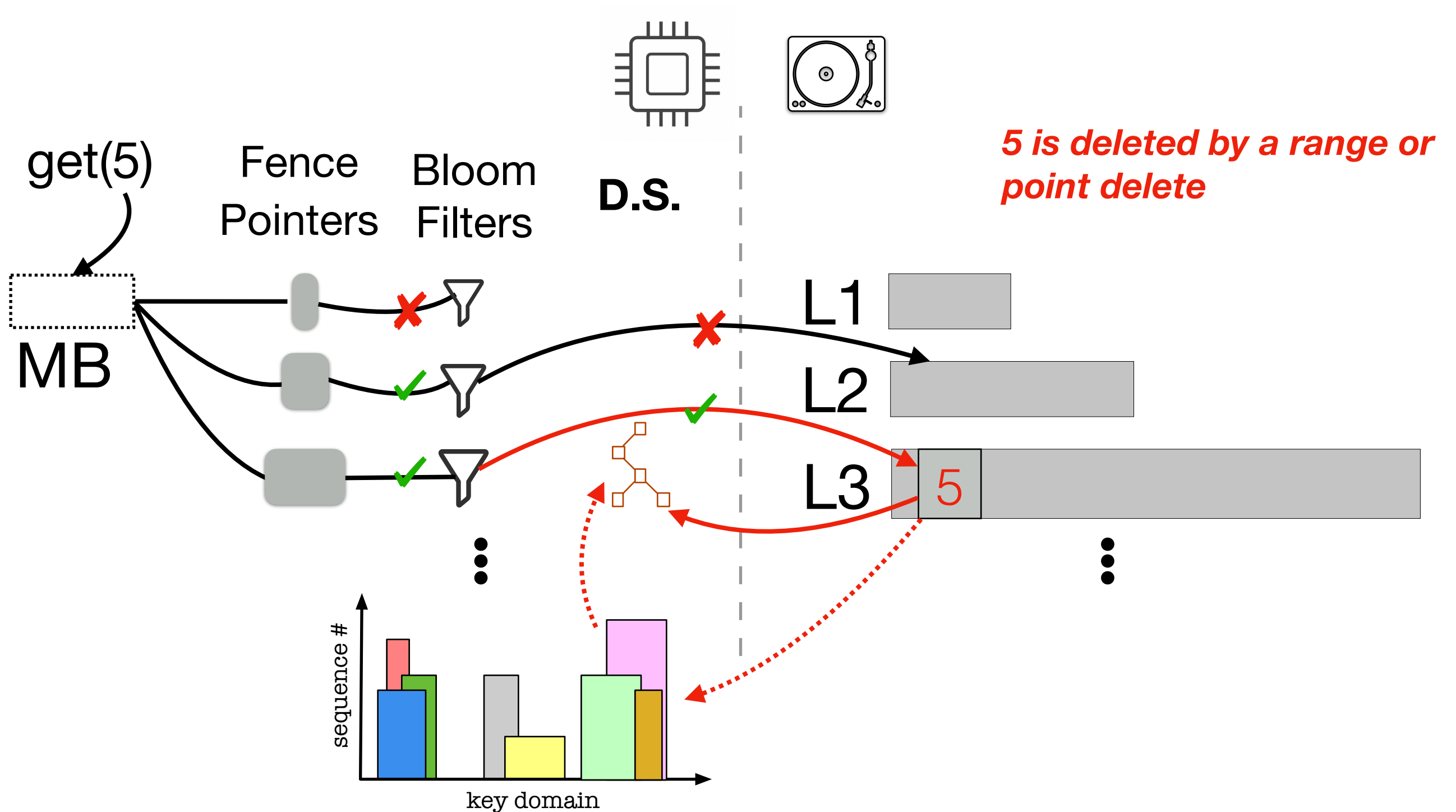
# Proposed solution in action



# Optimization

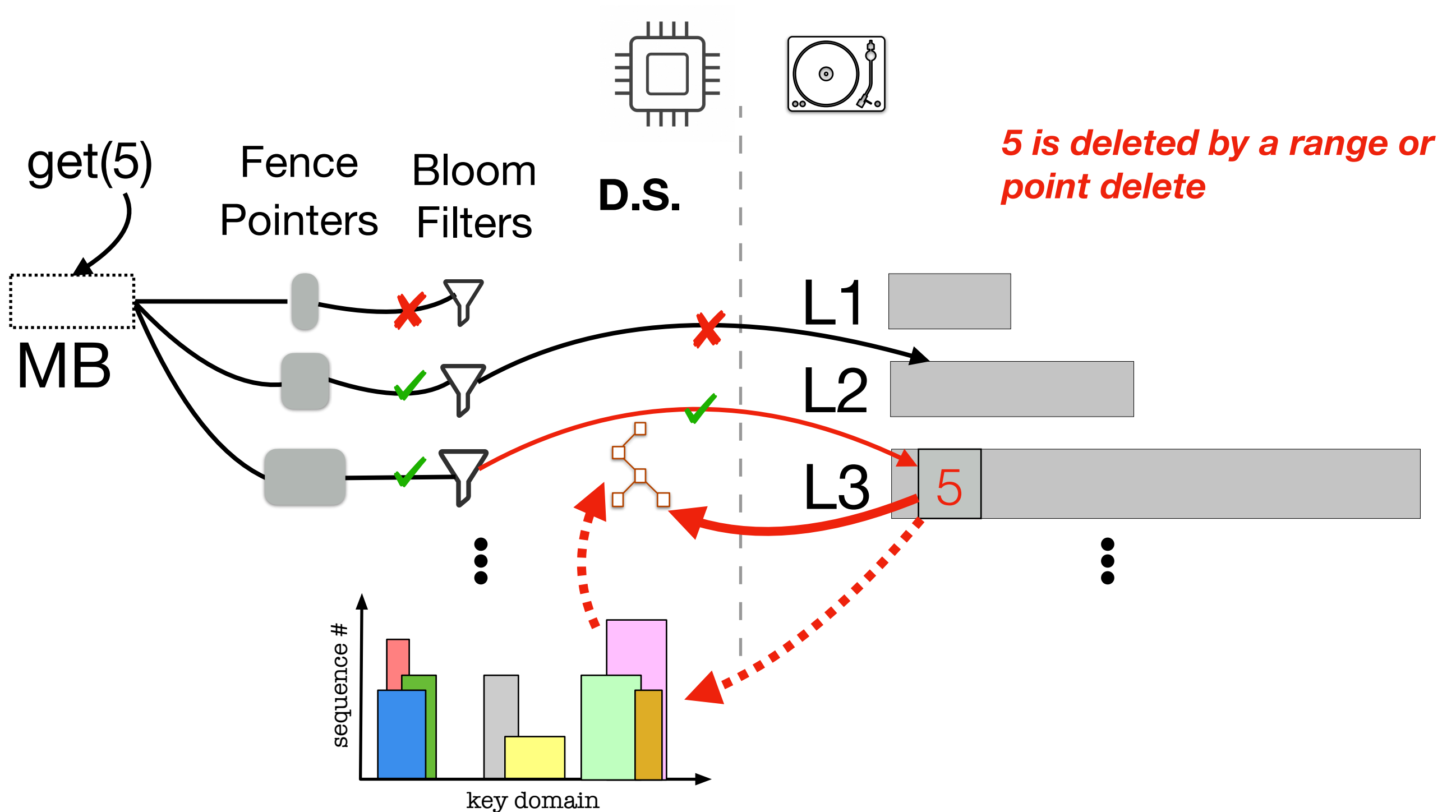


# Optimization benefit





# Optimization benefit



# D.S. desired properties

- **No false positives**
- **Small memory footprint**
- **Support for fast inserts and reads**

# Our method vs caching

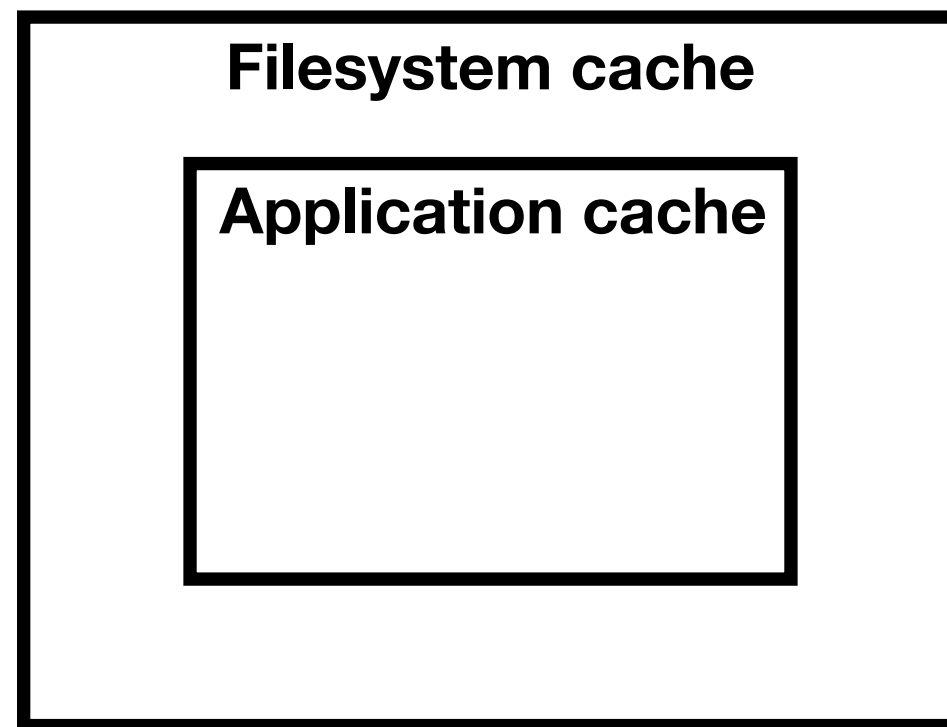
- **Better control of what we store**
- **Tunable memory footprint**
- **Both could work together**

# Experimentation

# Required Tools

- **Program that uses the RocksDB API**
- **Workload generator with multiple options**
- **Automated shell script to run experiments**

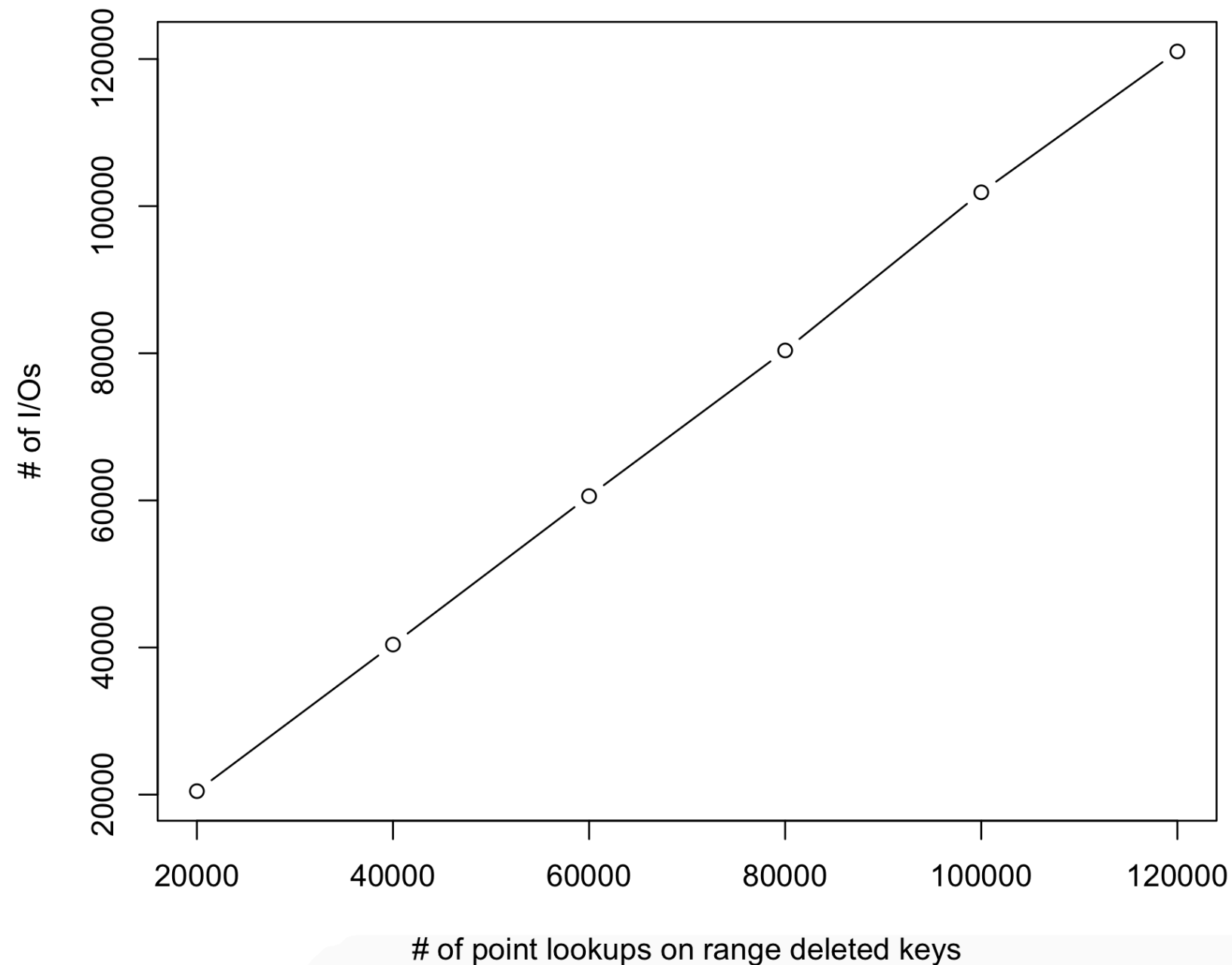
# Application vs Filesystem cache



**Disabled filesystem cache -> no OS interference**

# Repetitive queries on range deleted key

**Both Filesystem and Application cache disabled**



# Measuring range deletes I/Os

**Inserts: 1.000.000**

**Queries: 500.000**

**Repetitive Queries: 40%**

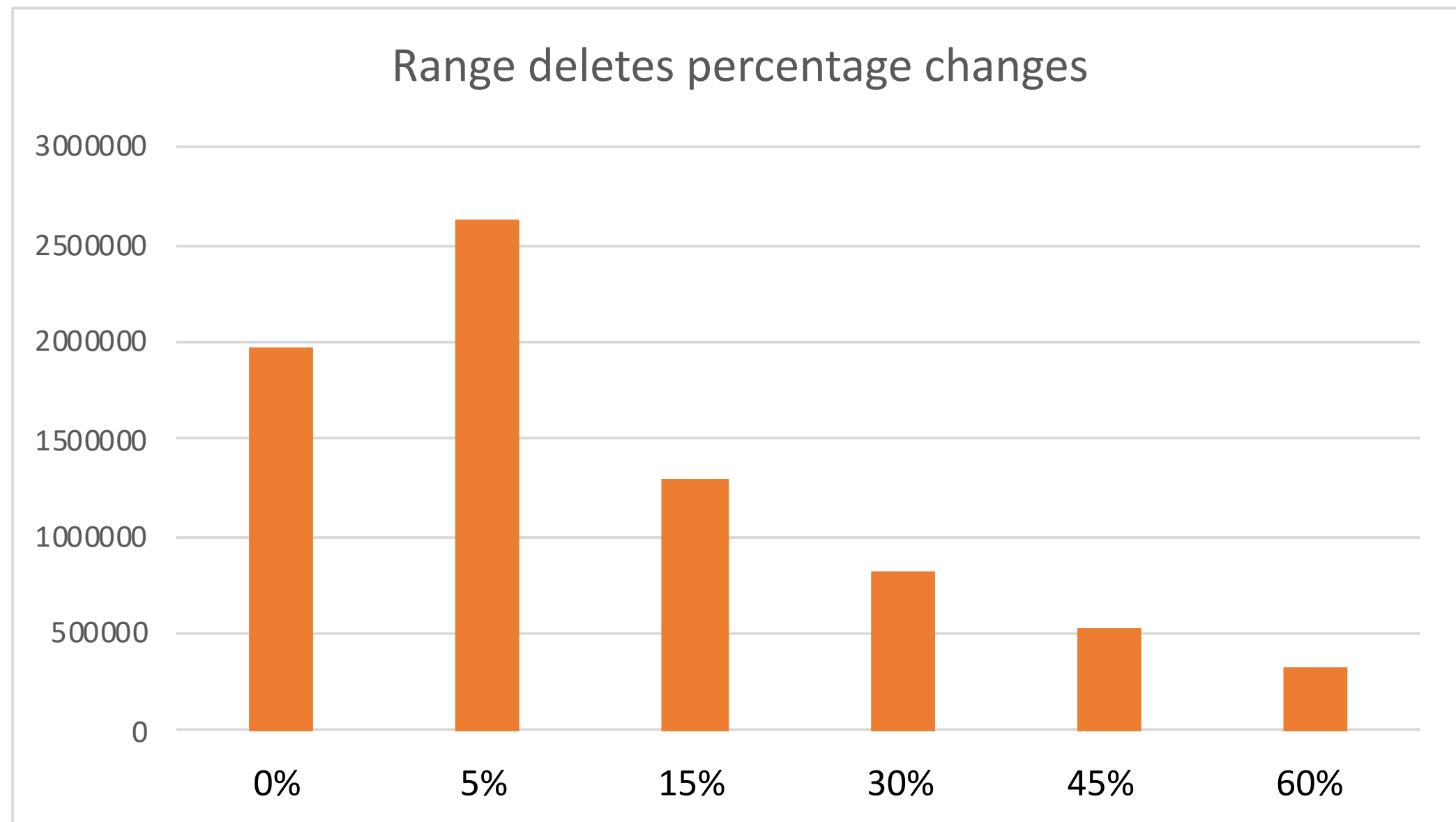
**Entry size: 1MB**

**Application Cache (Block Cache) size: 8MB**

**Variable: range deleted portion of the DB**

Range deletes percentage changes

**I/Os**





# Measuring range deletes I/Os

**Inserts: 1.000.000**

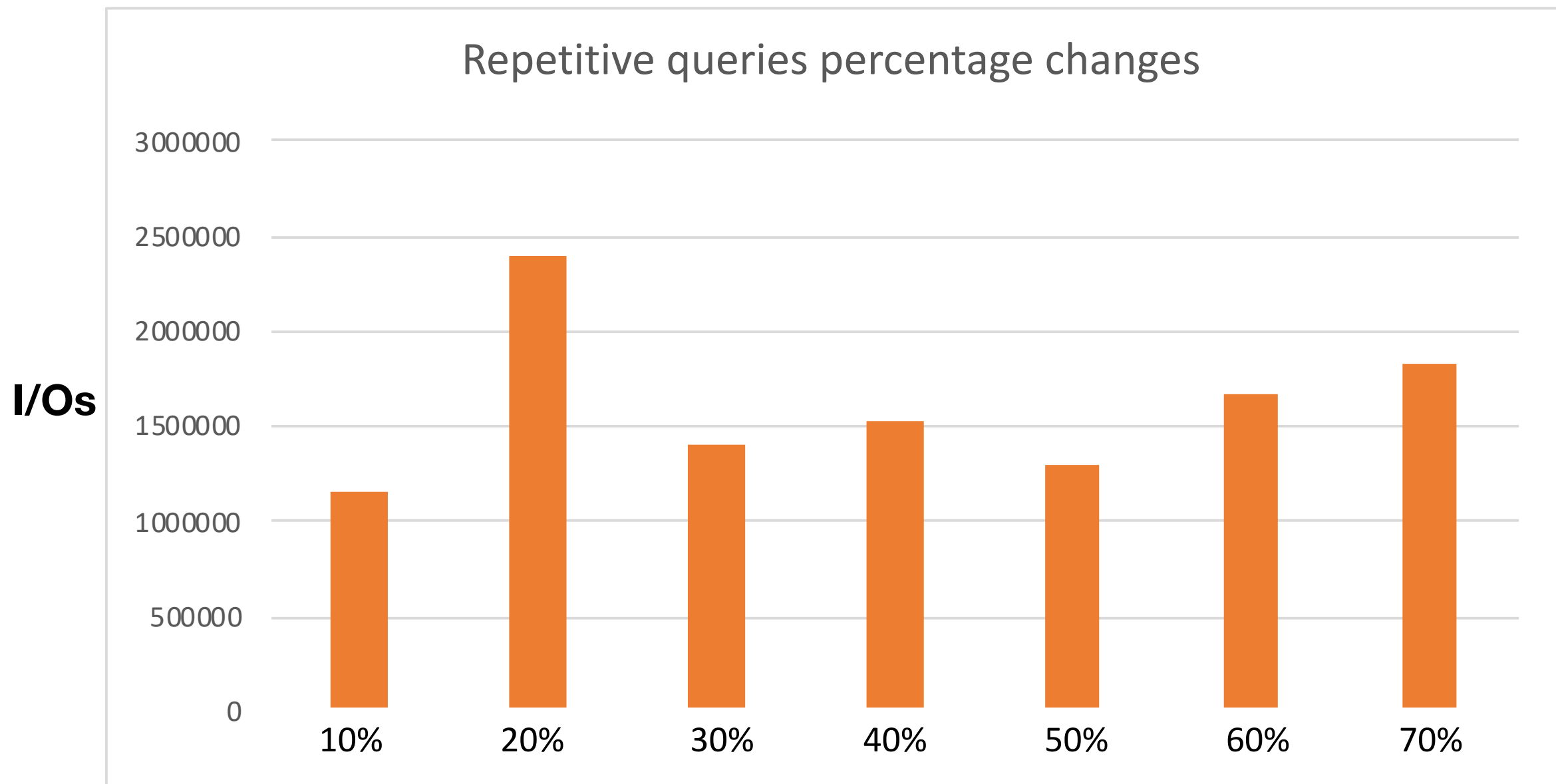
**Queries: 500.000**

**Range deleted portion of the DB: 15%**

**Entry size: 1MB**

**Application Cache (Block Cache) size: 8MB**

**Variable: percentage of repetitive queries**



**Thank you!**