Scheduling Problems in Write-Optimized Key-Value Stores

Prashant Pandey[1] Michael A. Bender[1] Rob Johnson[1,2]
[1]Stony Brook University, NY [2]VMware Research

Key-Value Stores are Ubiquitous

K1	Rob
K2	Michael
K3	Don
K4	Bill
K5	Jun
K6	Yang

- Can store and retrieve <key, value> pairs.
- KV stores are building blocks of databases, file systems, etc.
- Example: B-tree, Hash tables, etc.

Write-Optimized Key-Value Stores

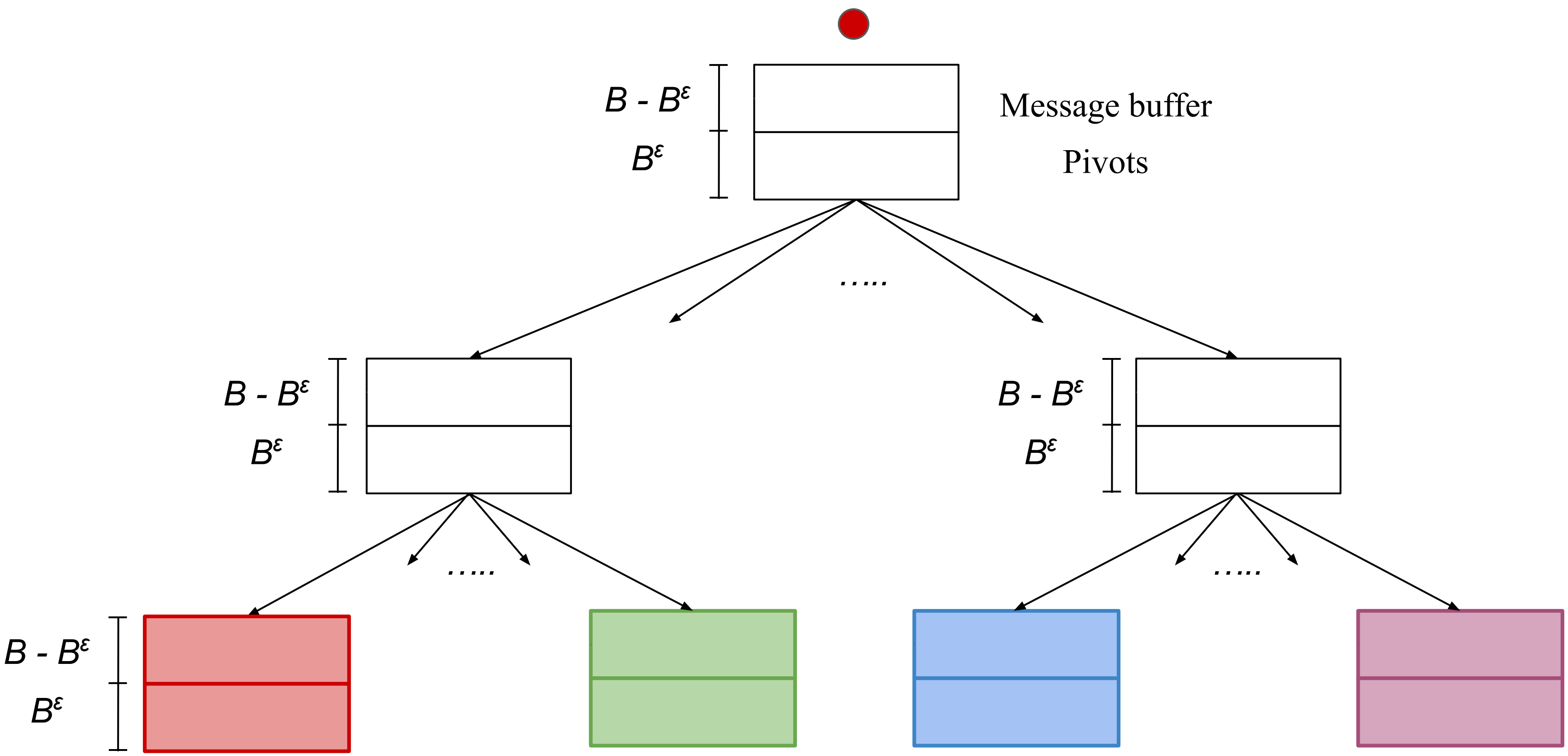
- State-of-the-art key-value stores are ***write optimized.***
- I.e. they **move data around in batches**.
- Batching amortizes the I/O cost of moving data.
- Write-optimized tree are designed for external memory.
- Examples: B^{ε}-trees or Log-structured merge trees.

Main idea of this talk:
how should we schedule
these batch data moves?

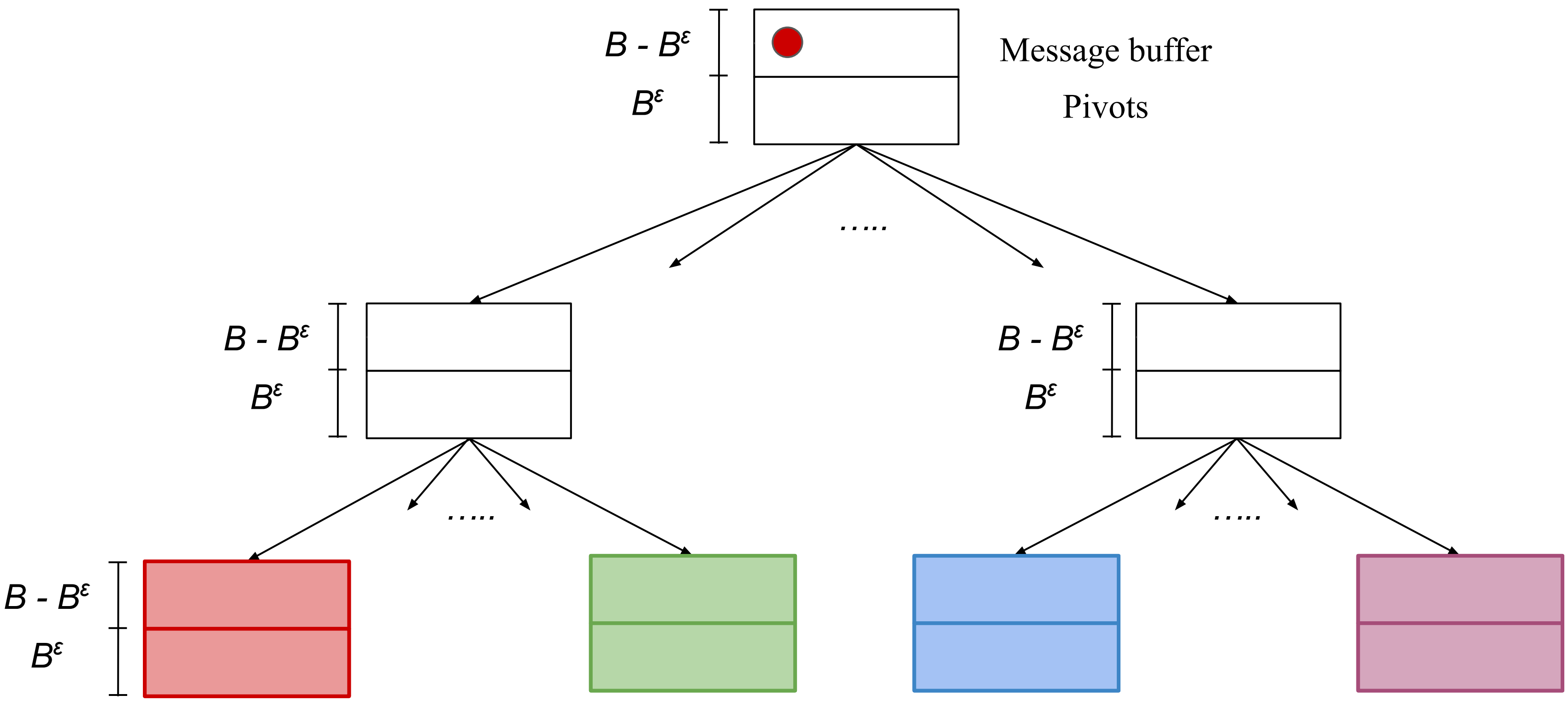
Outline

- B^{ε}-tree and operations
- Operations analysis
- Tradeoff between latency and I/O efficiency
- Scheduling problem in batch data moves

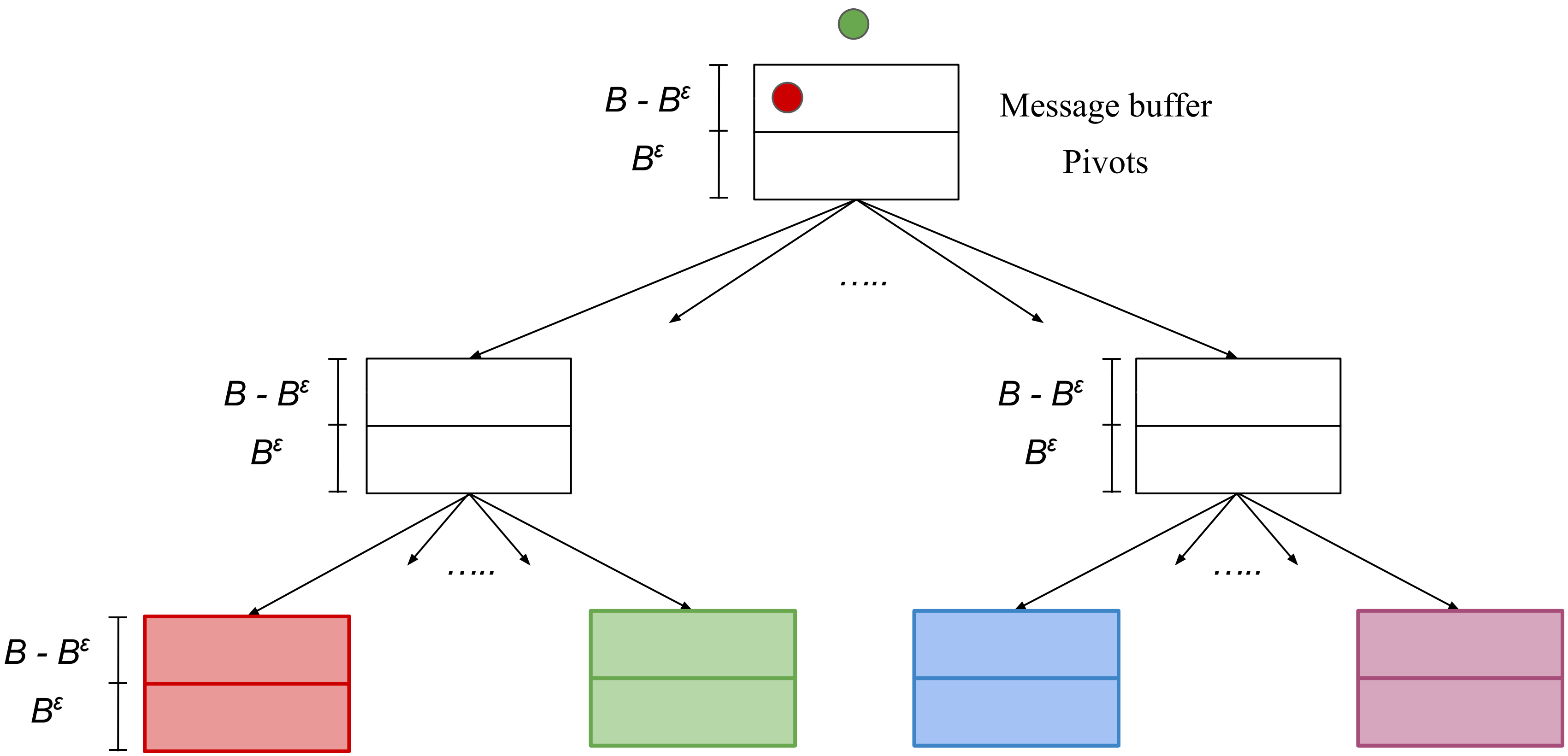
Insert Operation in a B^{ε}-tree



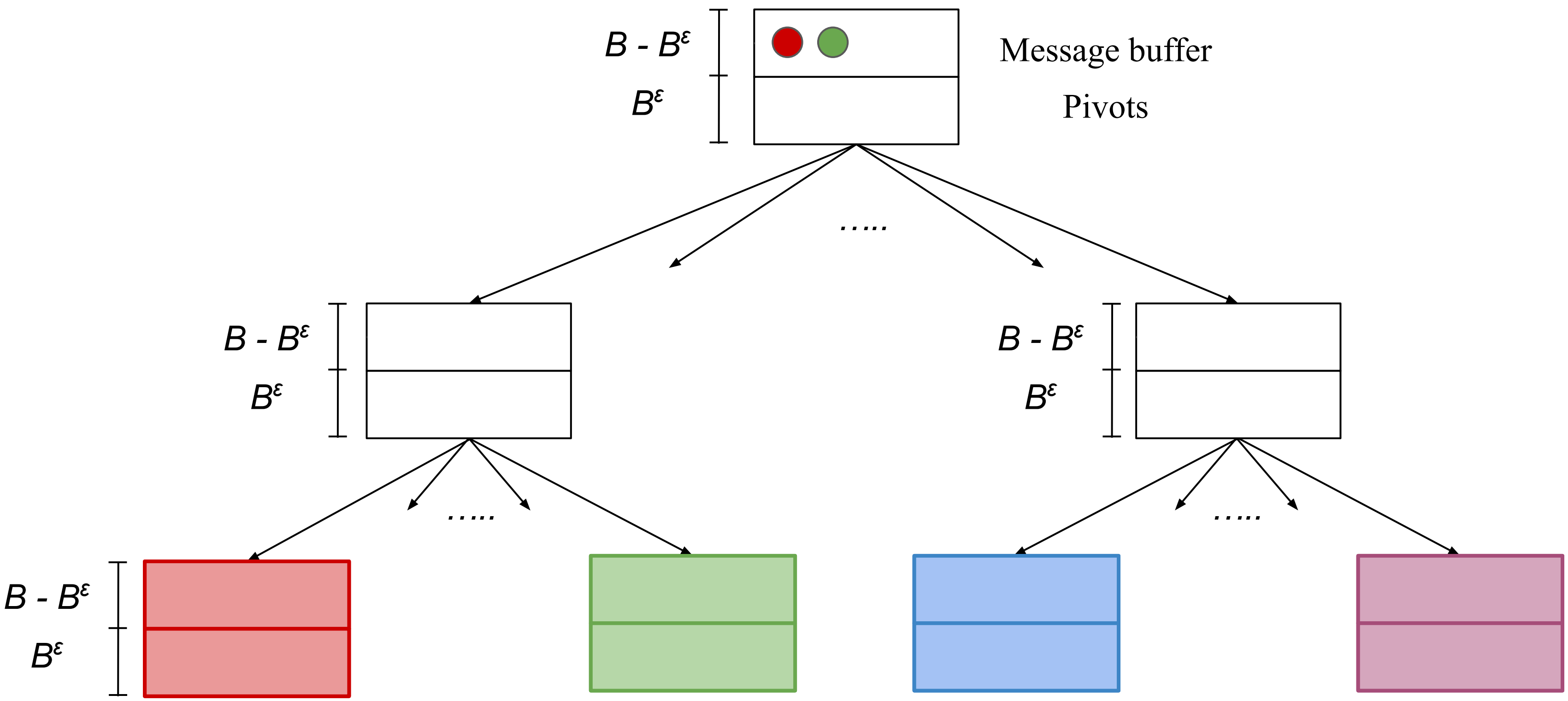
Insert Operation in a B^ε-tree



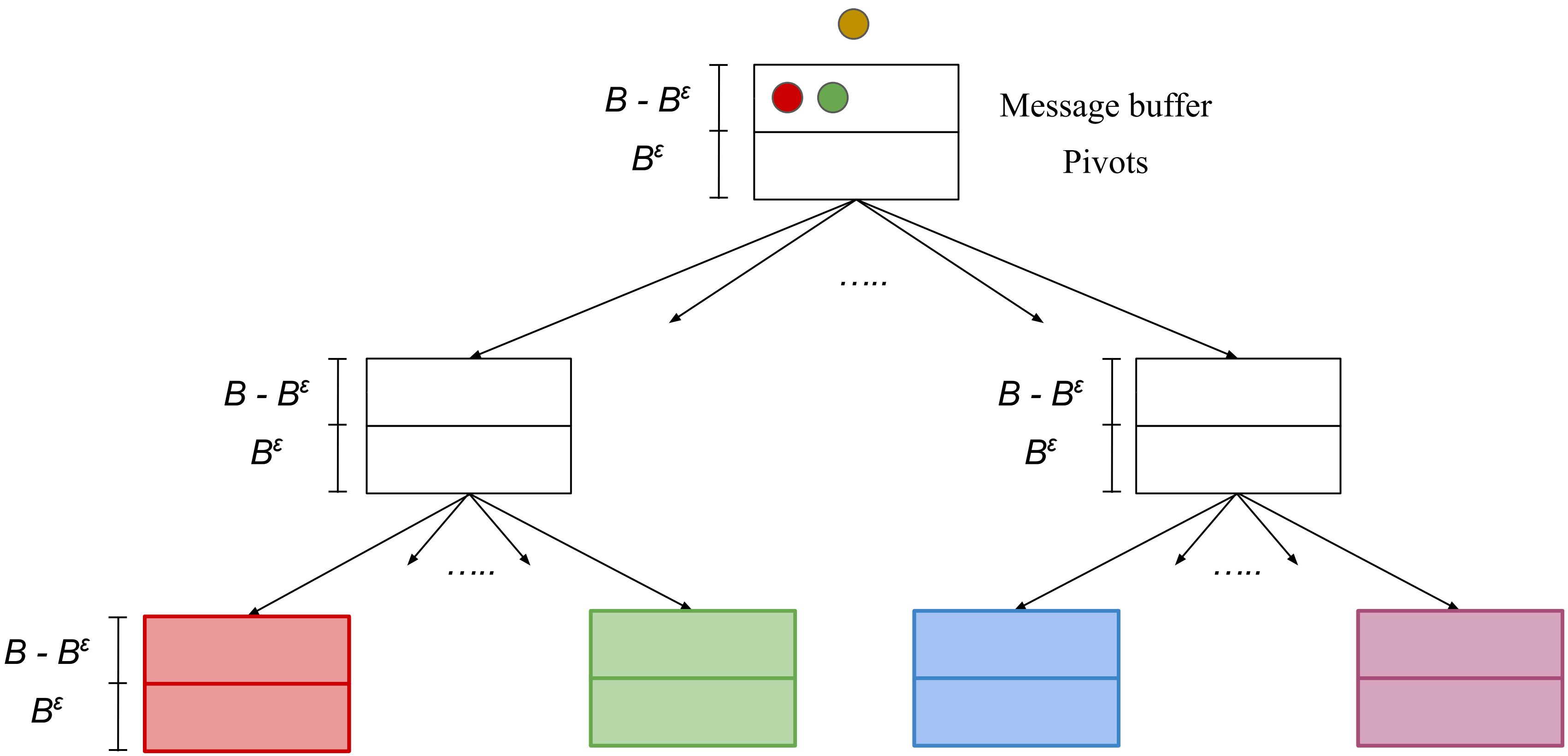
Insert Operation in a B^{ε}-tree



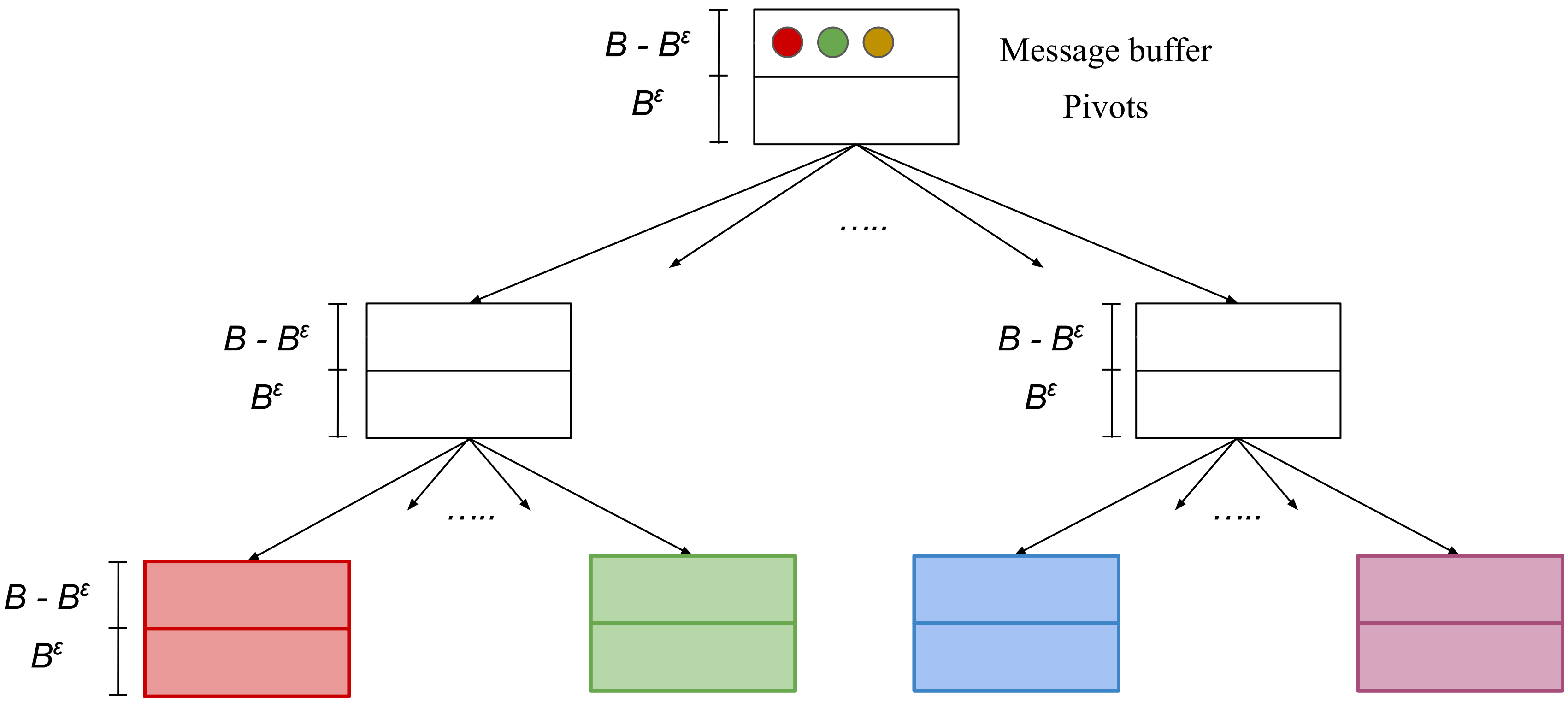
Insert Operation in a B$^{\varepsilon}$-tree



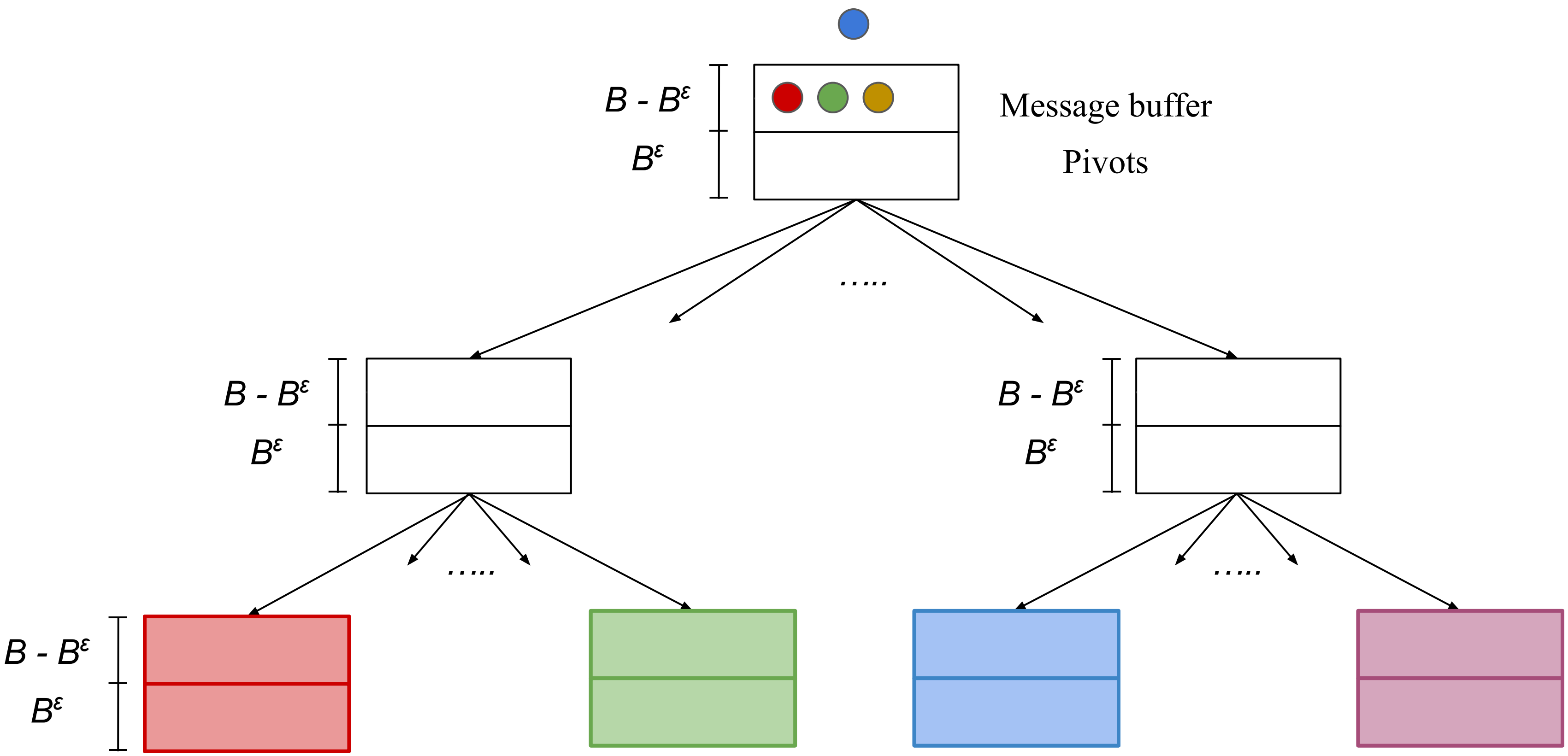
Insert Operation in a B^{ε}-tree



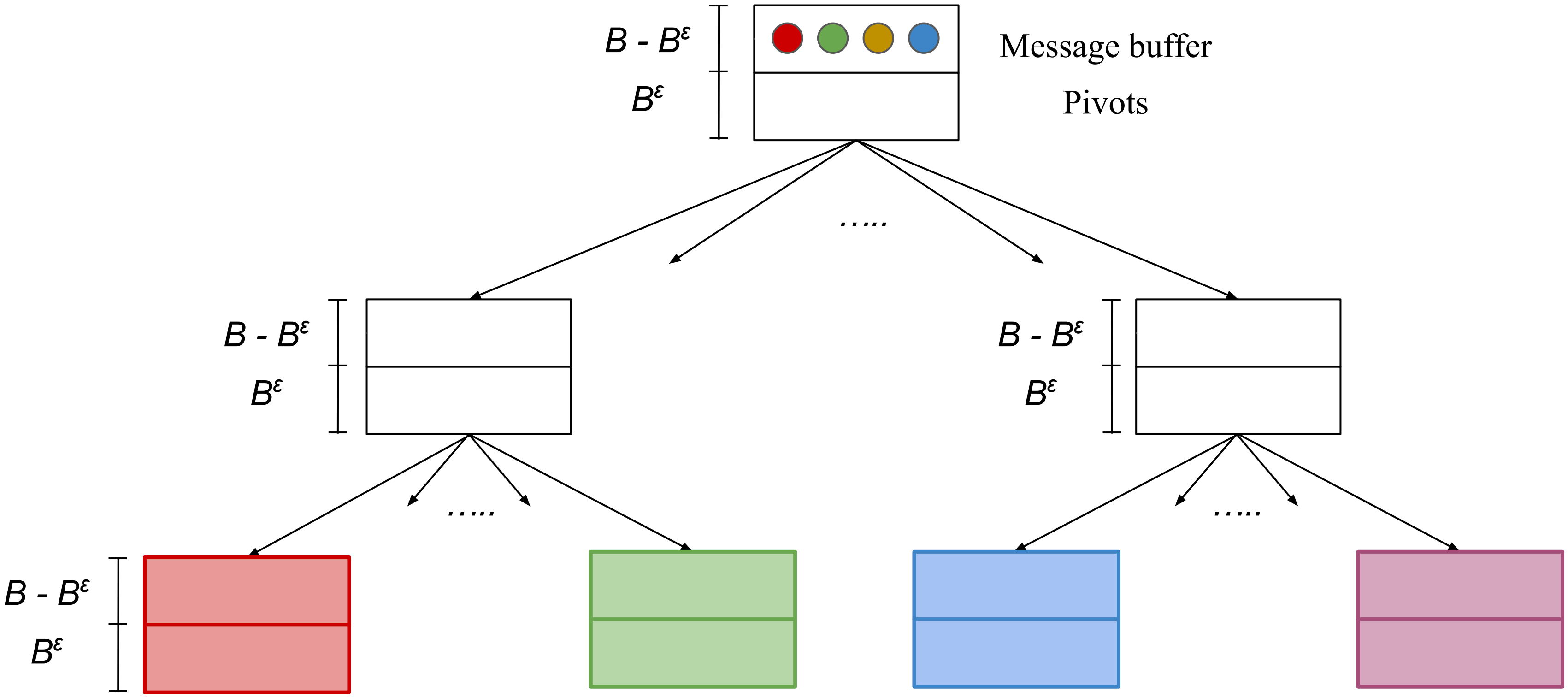
Insert Operation in a B$^\varepsilon$-tree



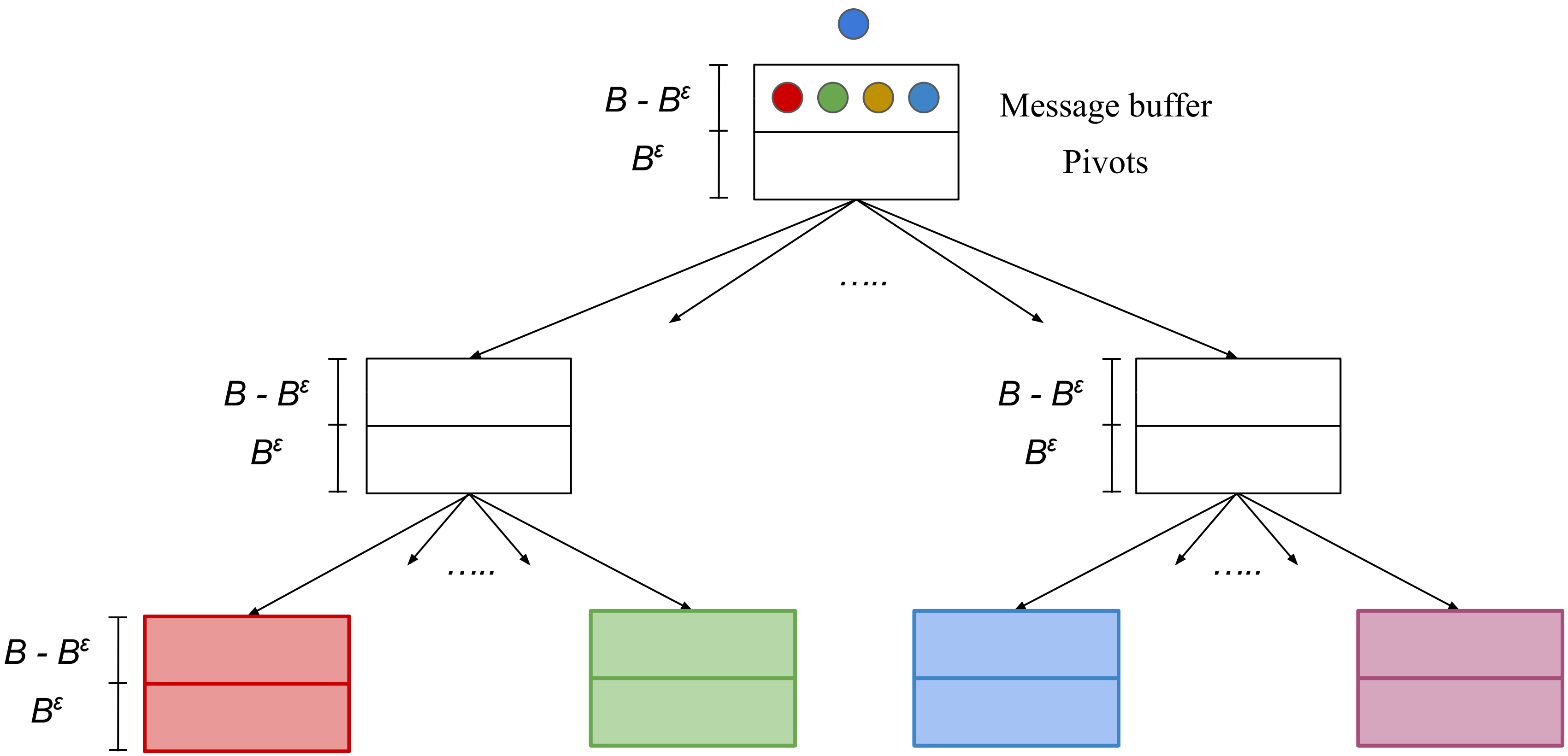
Insert Operation in a B$^{\varepsilon}$-tree



Insert Operation in a B^{ε}-tree

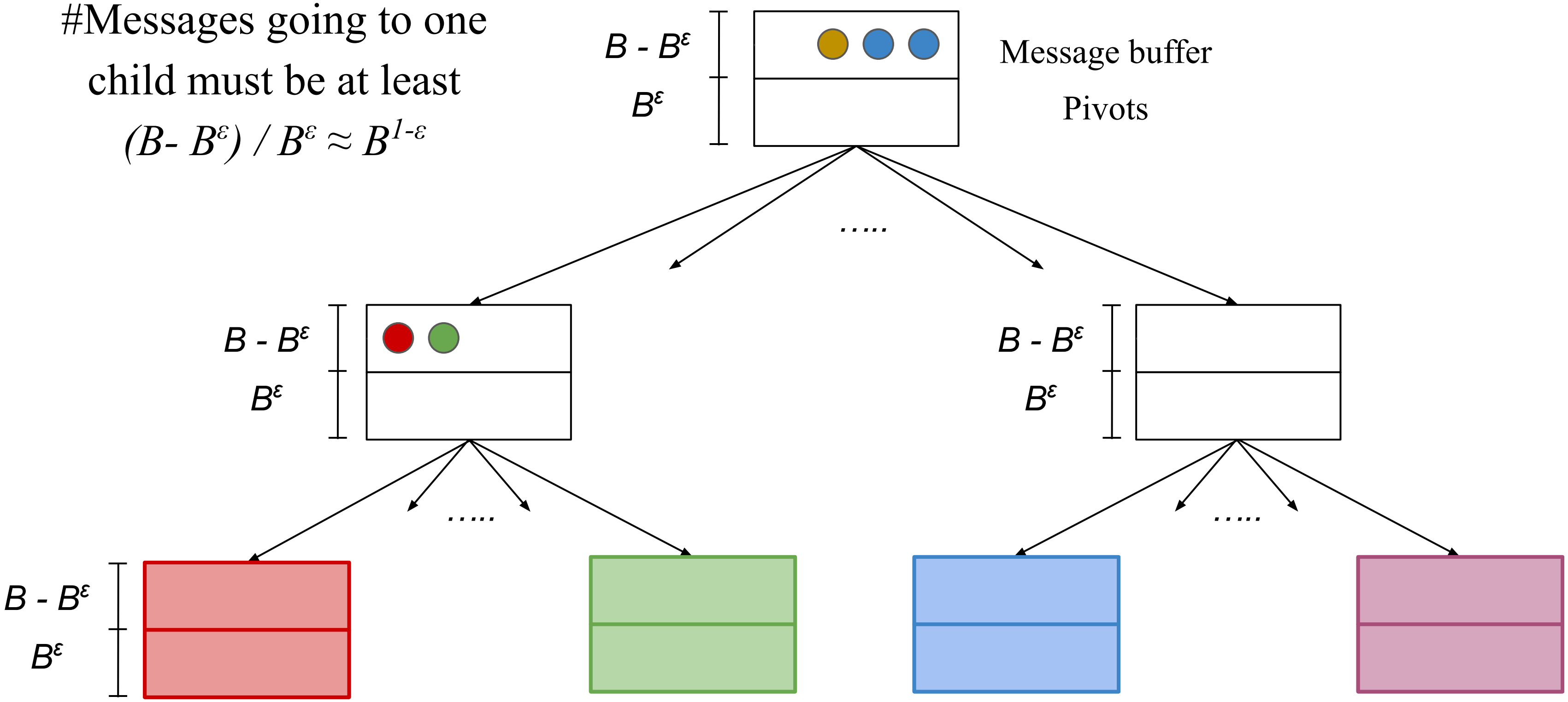


Insert Operation in a B^{ε}-tree

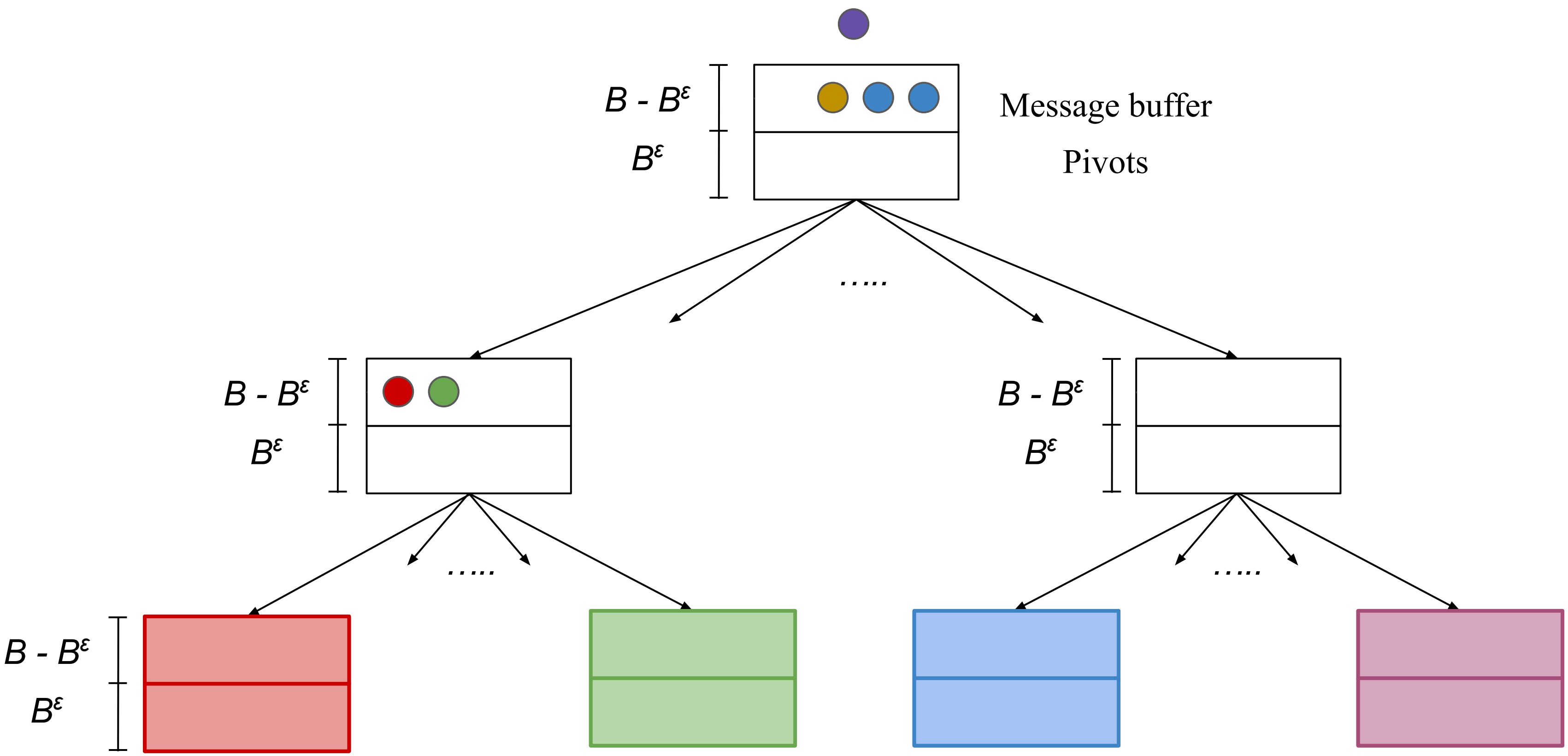


Insert Operation in a B^{ε}-tree

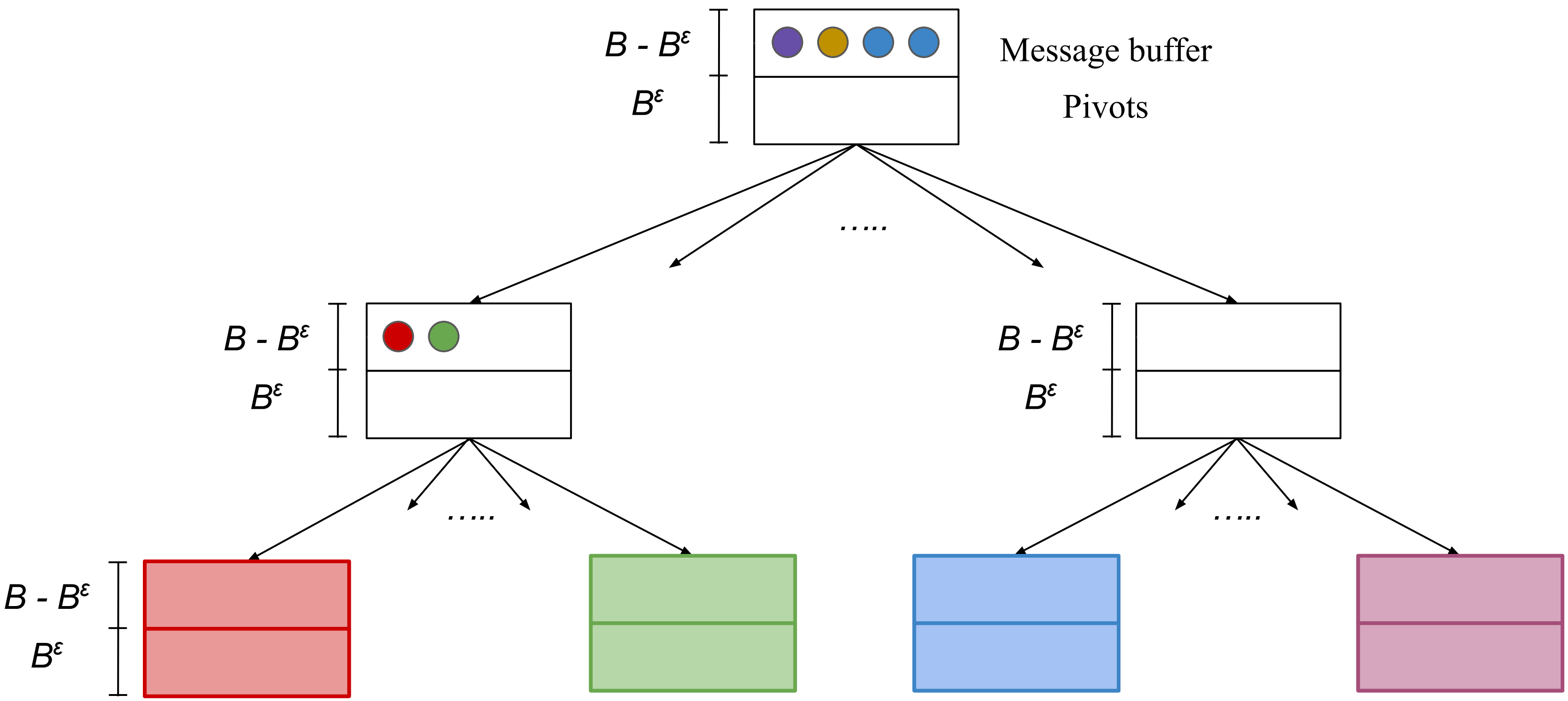
#Messages going to one child must be at least $(B - B^{\varepsilon}) / B^{\varepsilon} \approx B^{1-\varepsilon}$



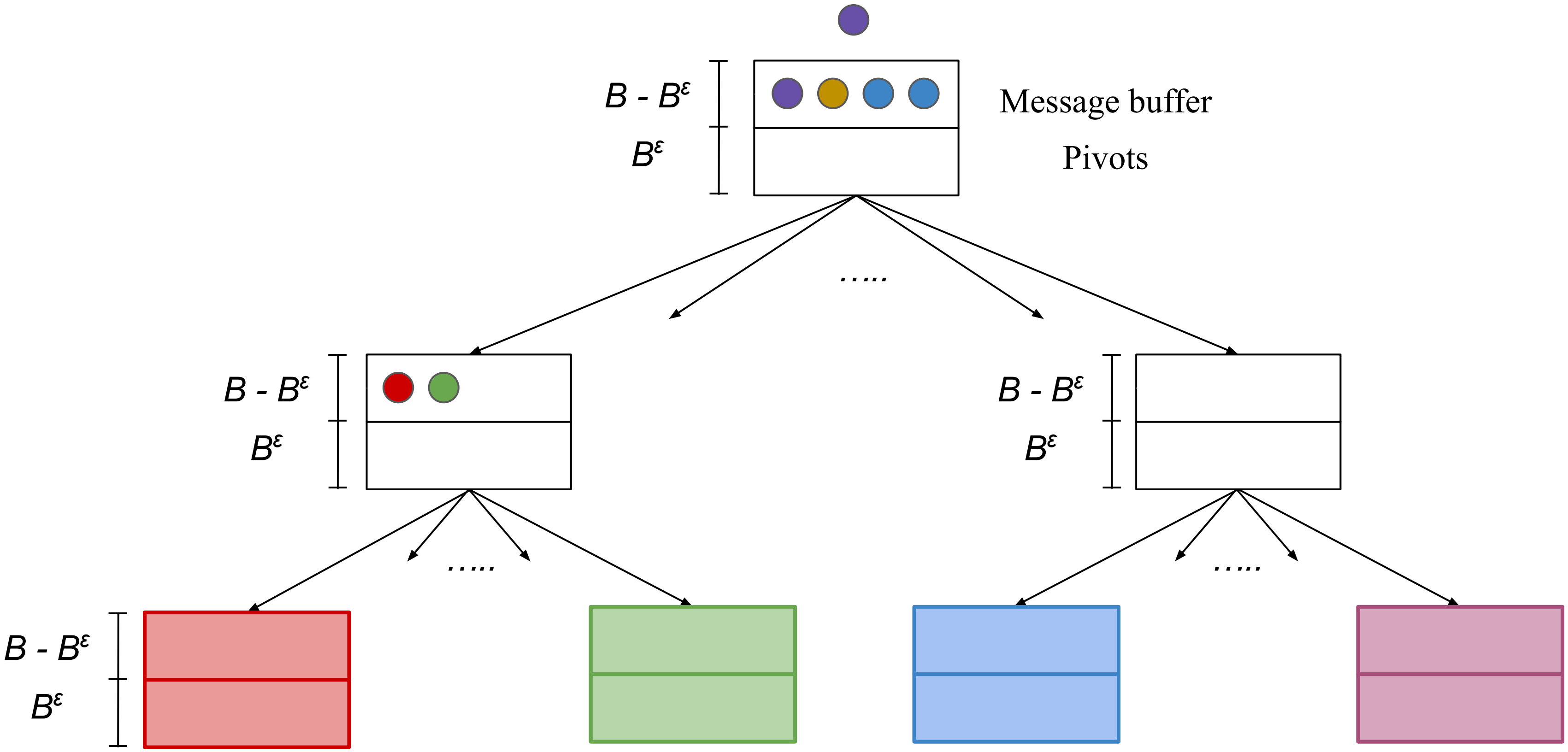
Insert Operation in a B$^\varepsilon$-tree



Insert Operation in a B^ε-tree

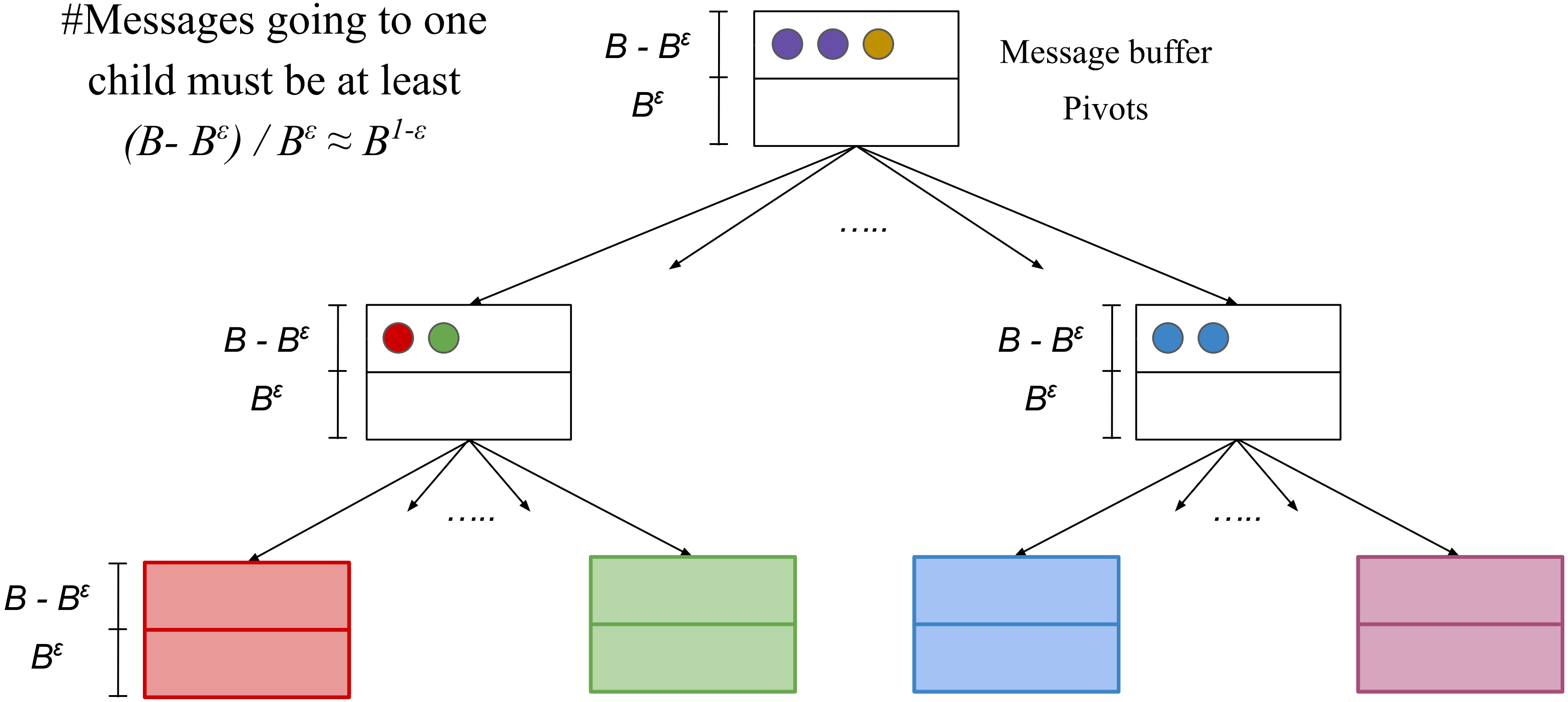


Insert Operation in a B^{ε}-tree

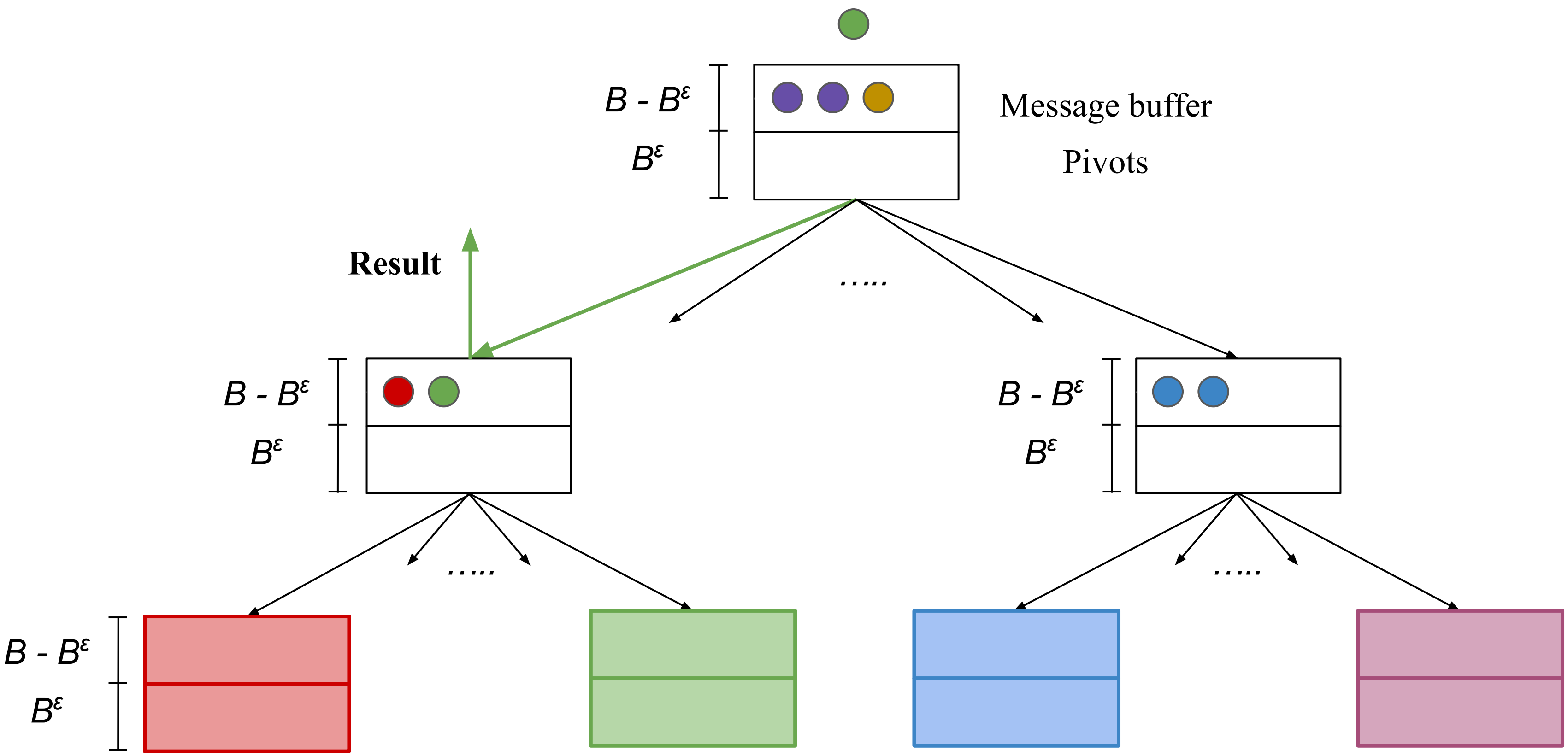


Insert Operation in a B^{ε}-tree

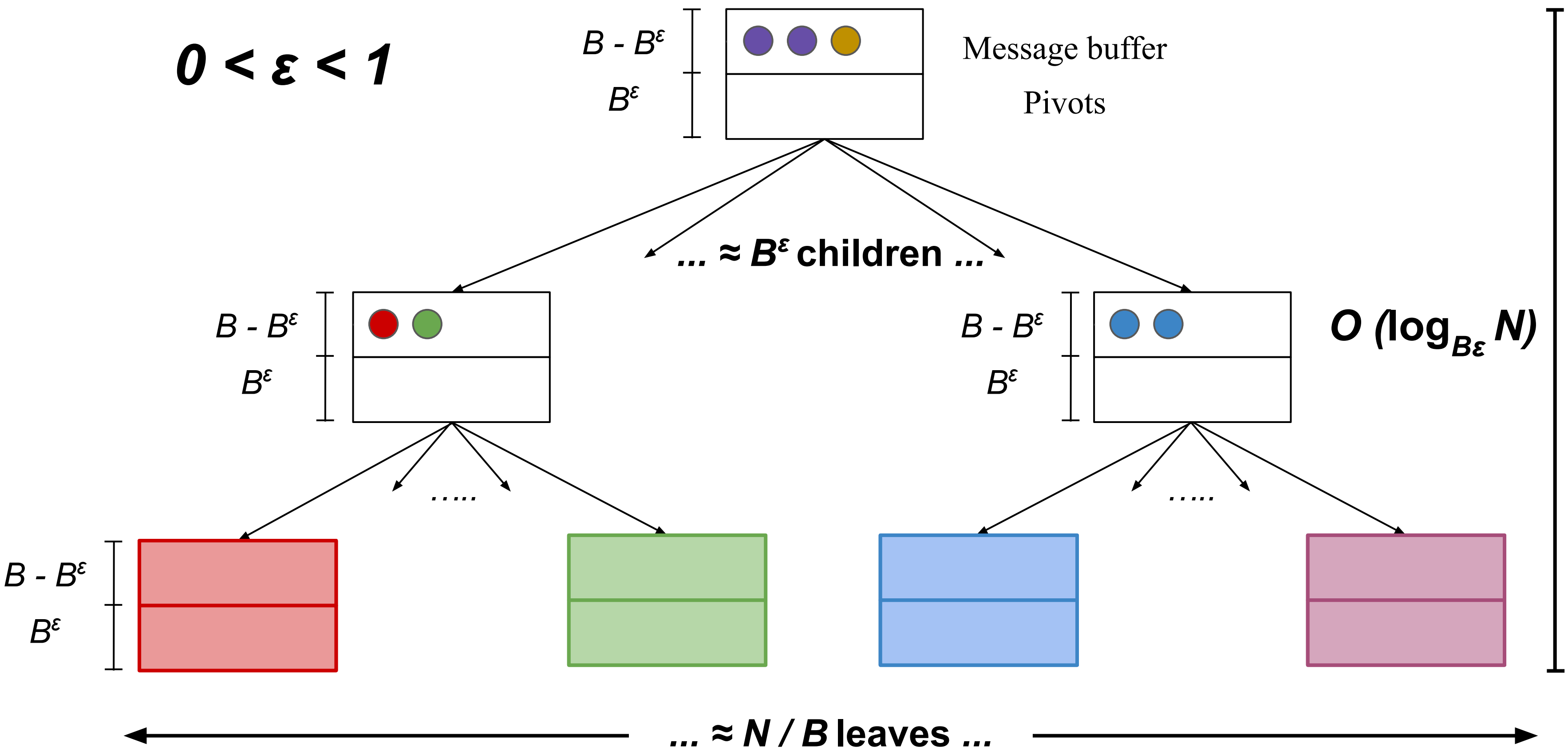
#Messages going to one child must be at least $(B - B^{\varepsilon}) / B^{\varepsilon} \approx B^{1-\varepsilon}$



Query Operation in a B^{ε}-tree

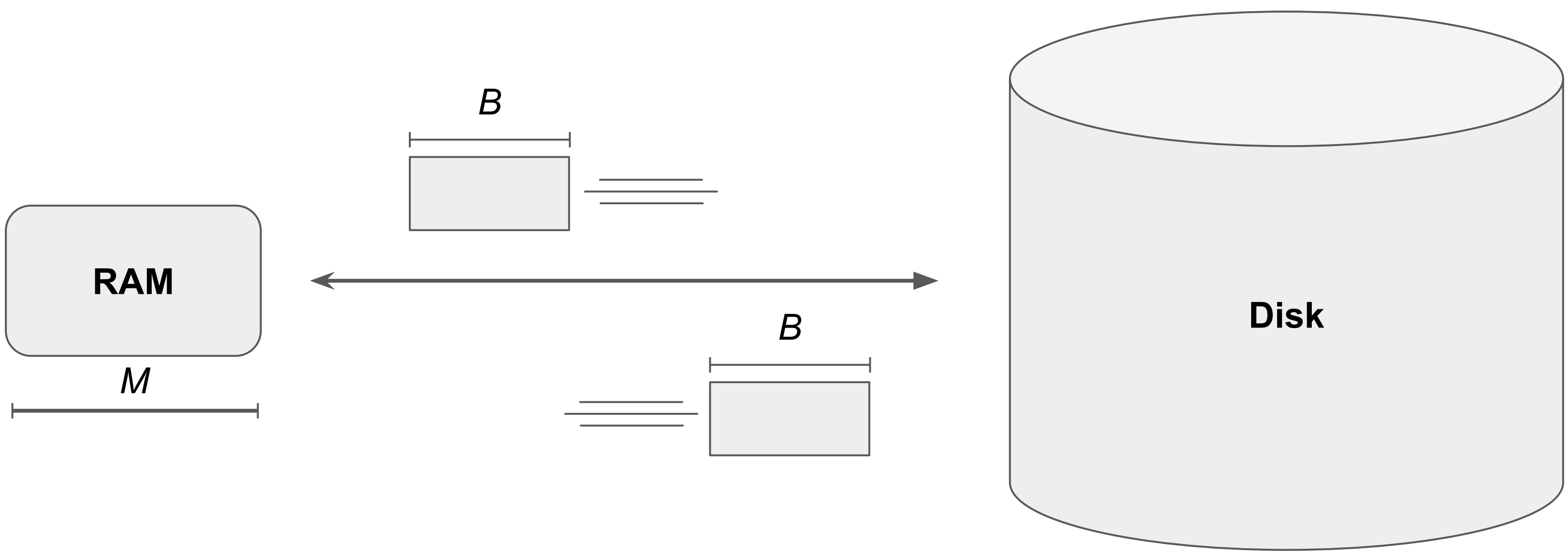


B^{ε}-tree



Performance Model

- How computation works
  - Data is transferred in blocks between RAM and disk.
  - The number of block transfers dominates the running time.
- Goal: minimize number of block transfers
  - Performance bounds are parameterized by block size *B*, memory size *M*, data size *N*.



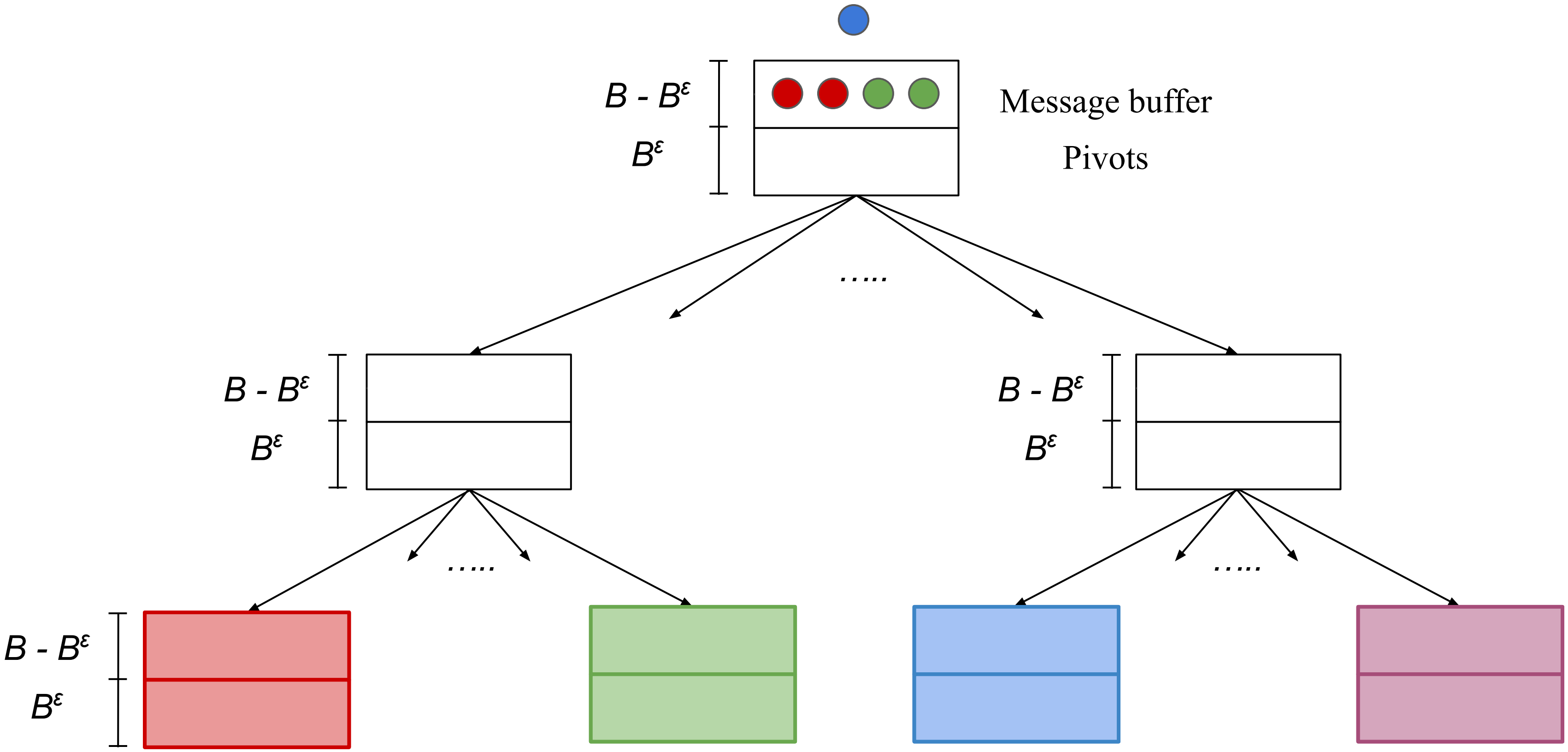
Operations

	Insert	query	Range query
B-tree	$\mathrm{Log}_B N$	$\log_B N$	$\log_B N + k/N$
B^{ε}-tree	$\mathrm{Log}_B N \,/\, \varepsilon B^{1-\varepsilon}$	$\log_B N \,/\, \varepsilon$	$\log_B N \,/\, \varepsilon + k/N$
B^{ε}-tree ($\varepsilon = 1/2$)	$\log_B N \,/\, \sqrt{B}$	$\log_B N$	$\log_B N + k/N$

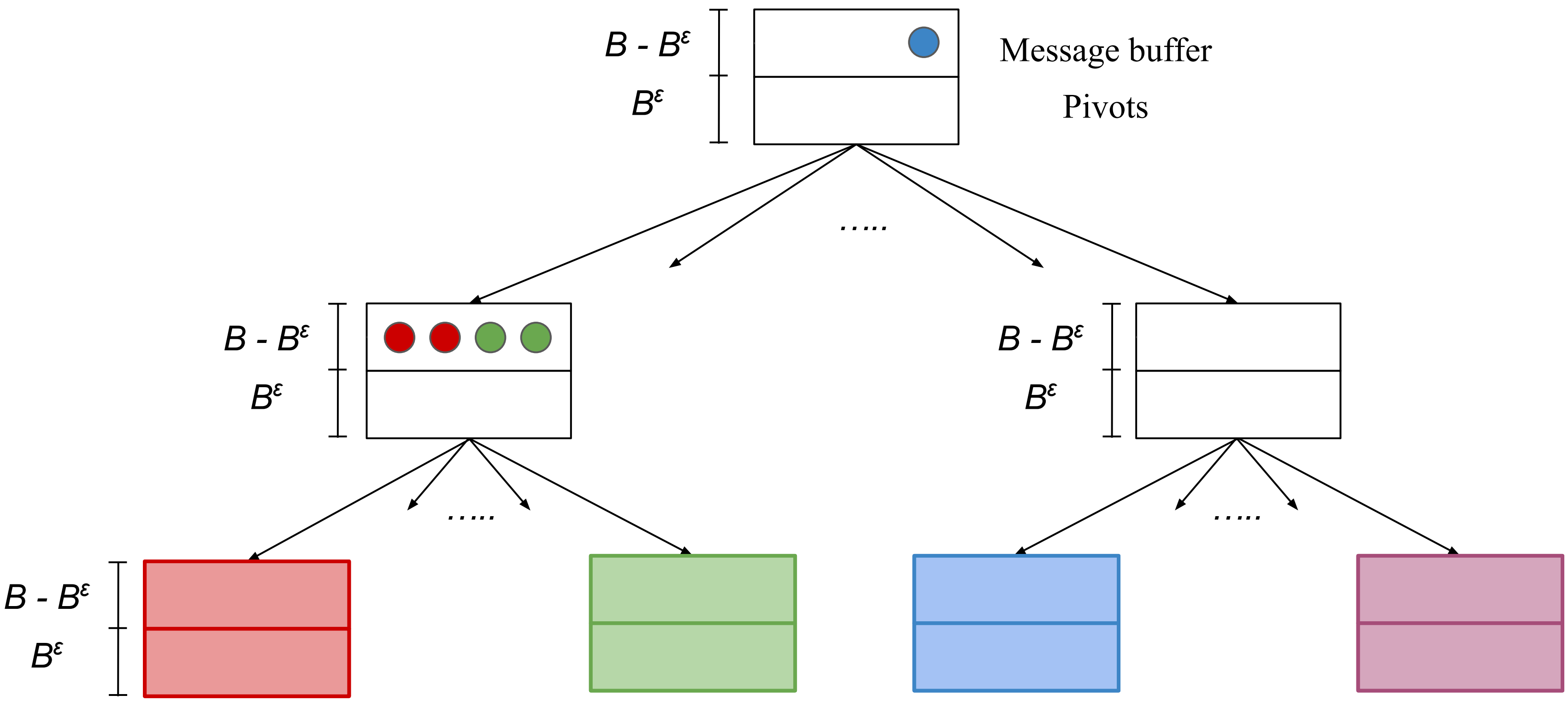
Operations

	Insert	query	Range query
B-tree	$\mathrm{Log}_B N$	$\log_B N$	$\log_B N + k/N$
B^{ε}-tree	$\boldsymbol{\mathrm{Log}_B N / \varepsilon B^{1-\varepsilon}}$	$\log_B N / \varepsilon$	$\log_B N / \varepsilon + k/N$
B^{ε}-tree ($\varepsilon = 1/2$)	$\log_B N / \sqrt{B}$	$\log_B N$	$\log_B N + k/N$

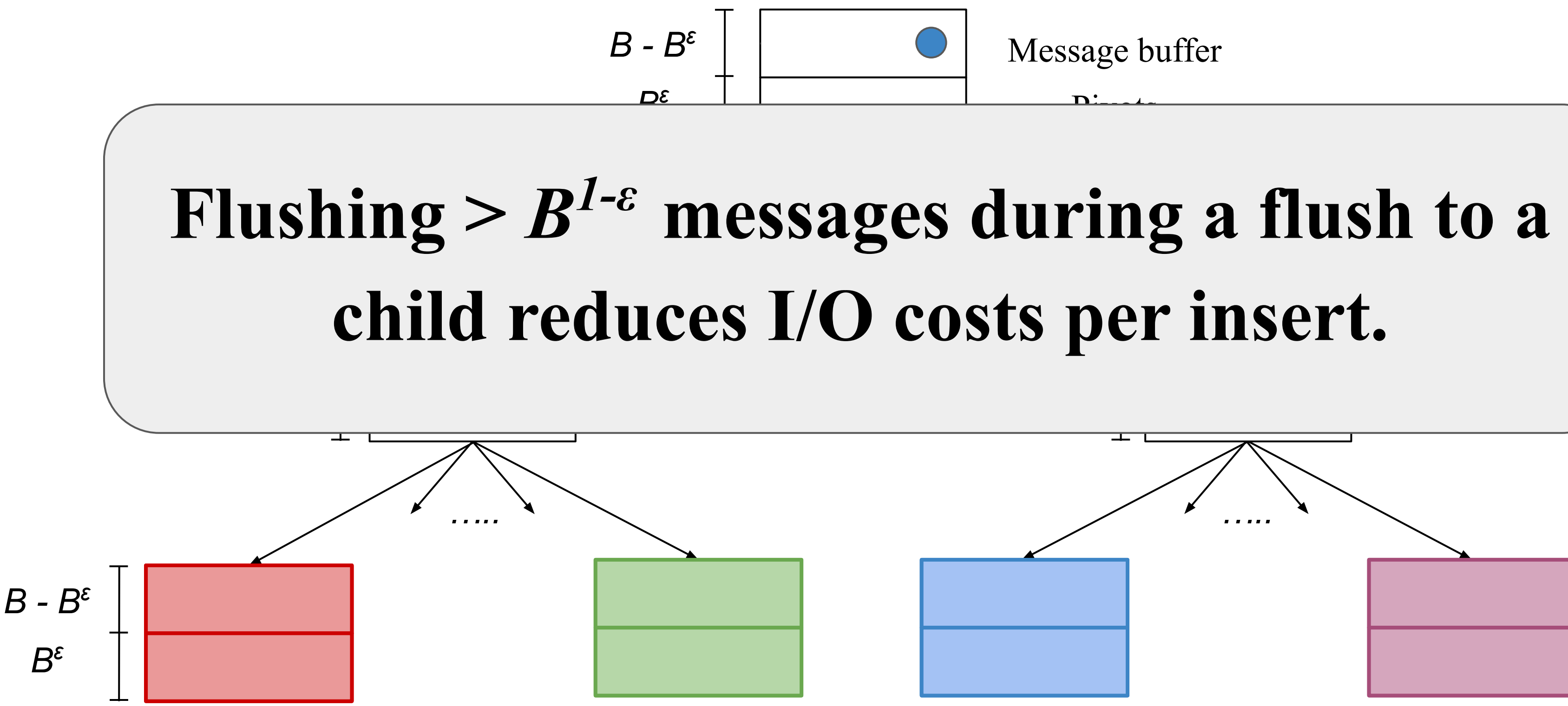
Moving More than $B^{1-\varepsilon}$ Messages in a Flush



Moving More than $B^{1-\varepsilon}$ Messages in a Flush

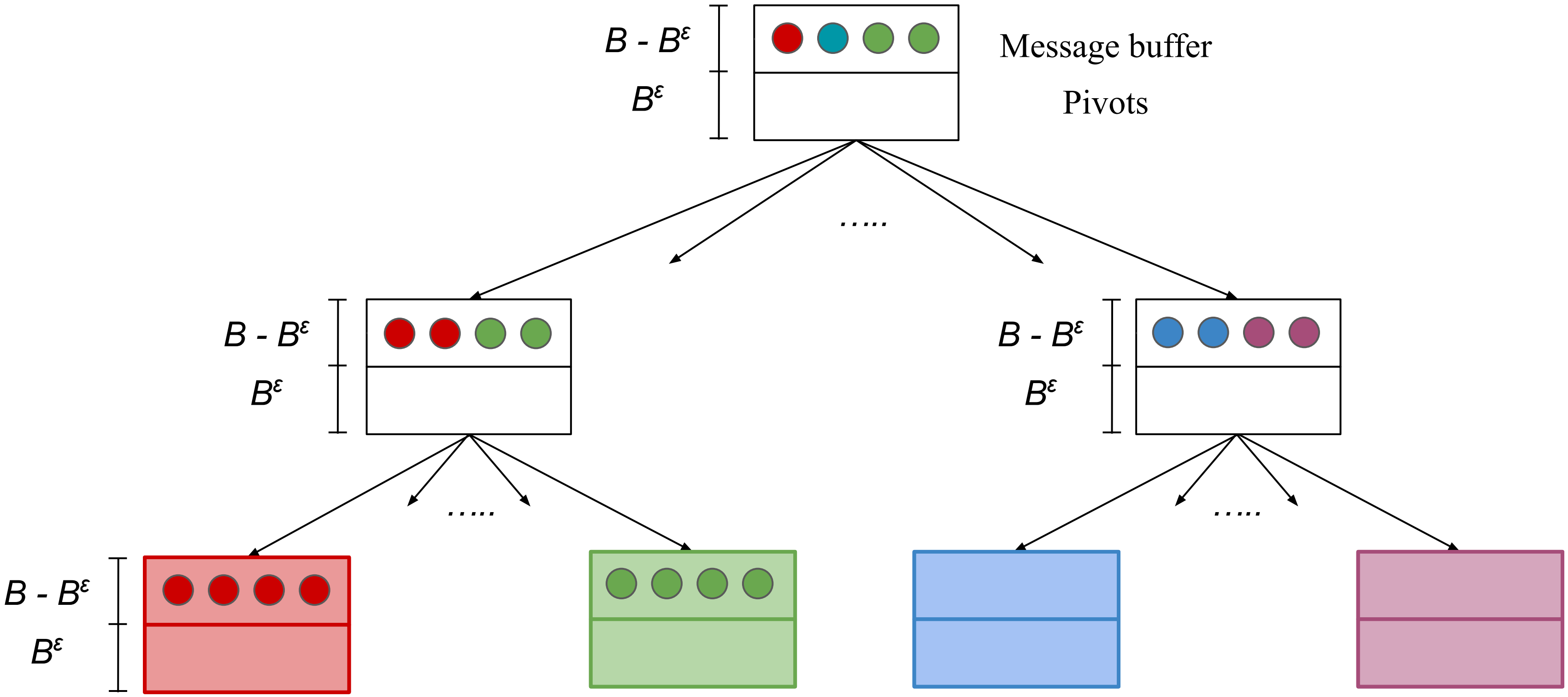


Moving More than $B^{1-\varepsilon}$ Messages in a Flush

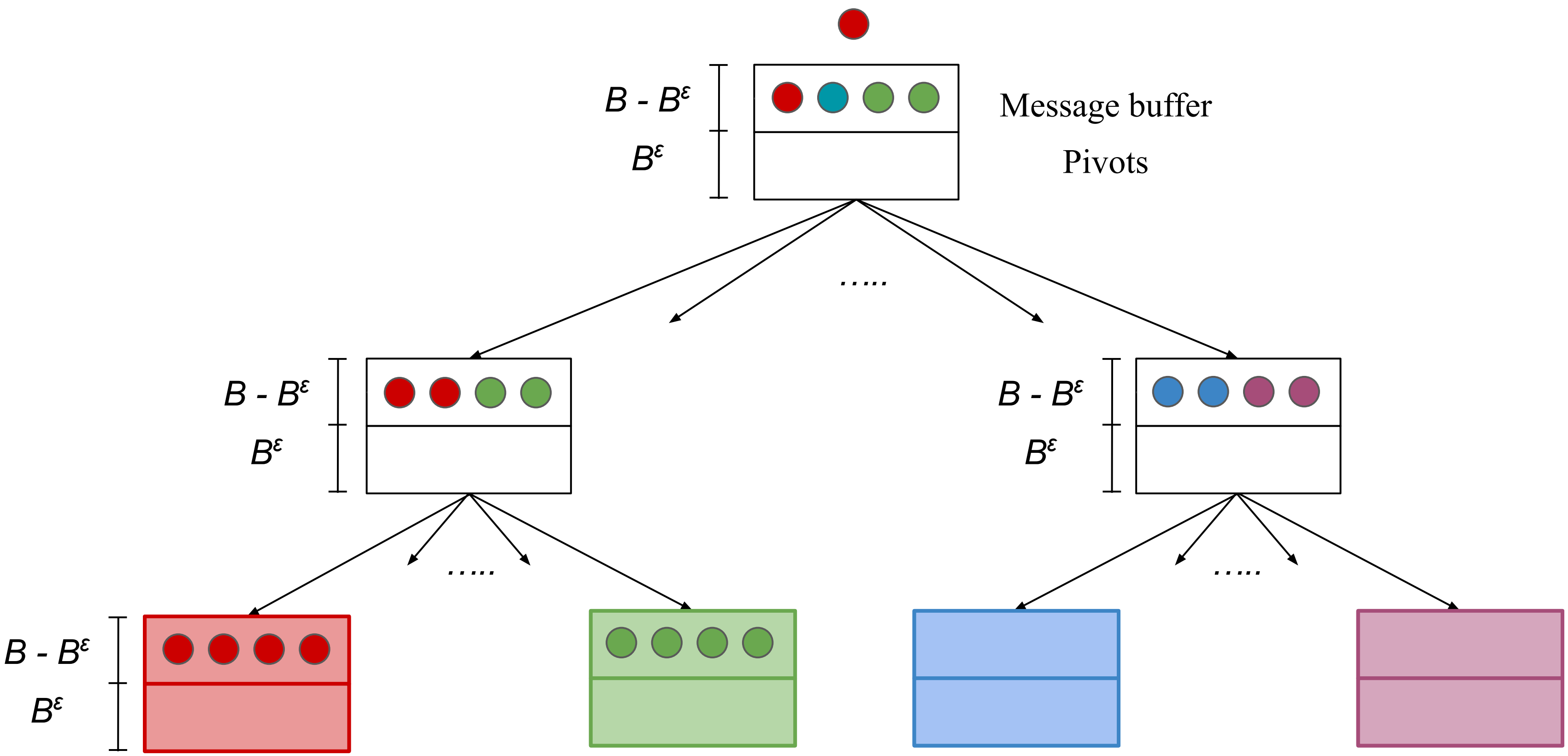


Flushing > $B^{1-\varepsilon}$ messages during a flush to a child reduces I/O costs per insert.

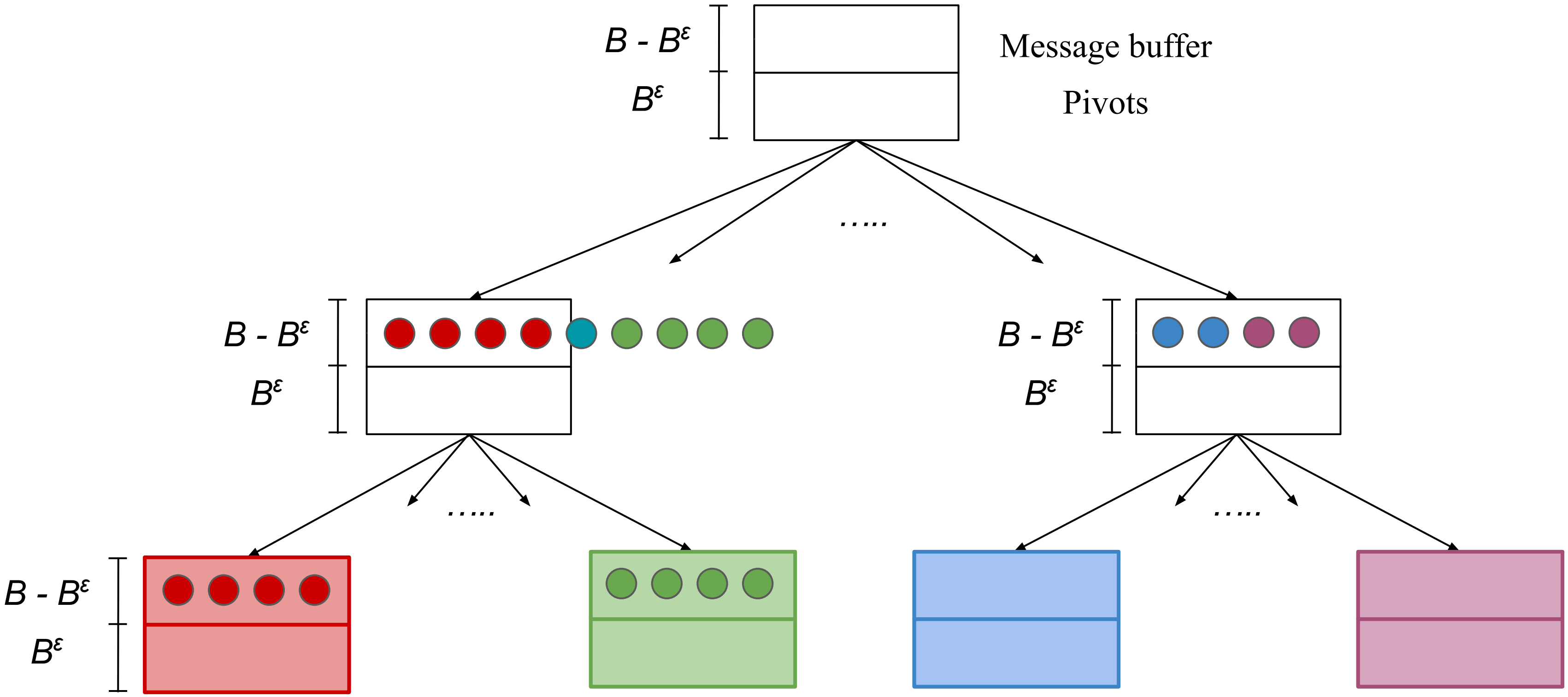
Avalanche



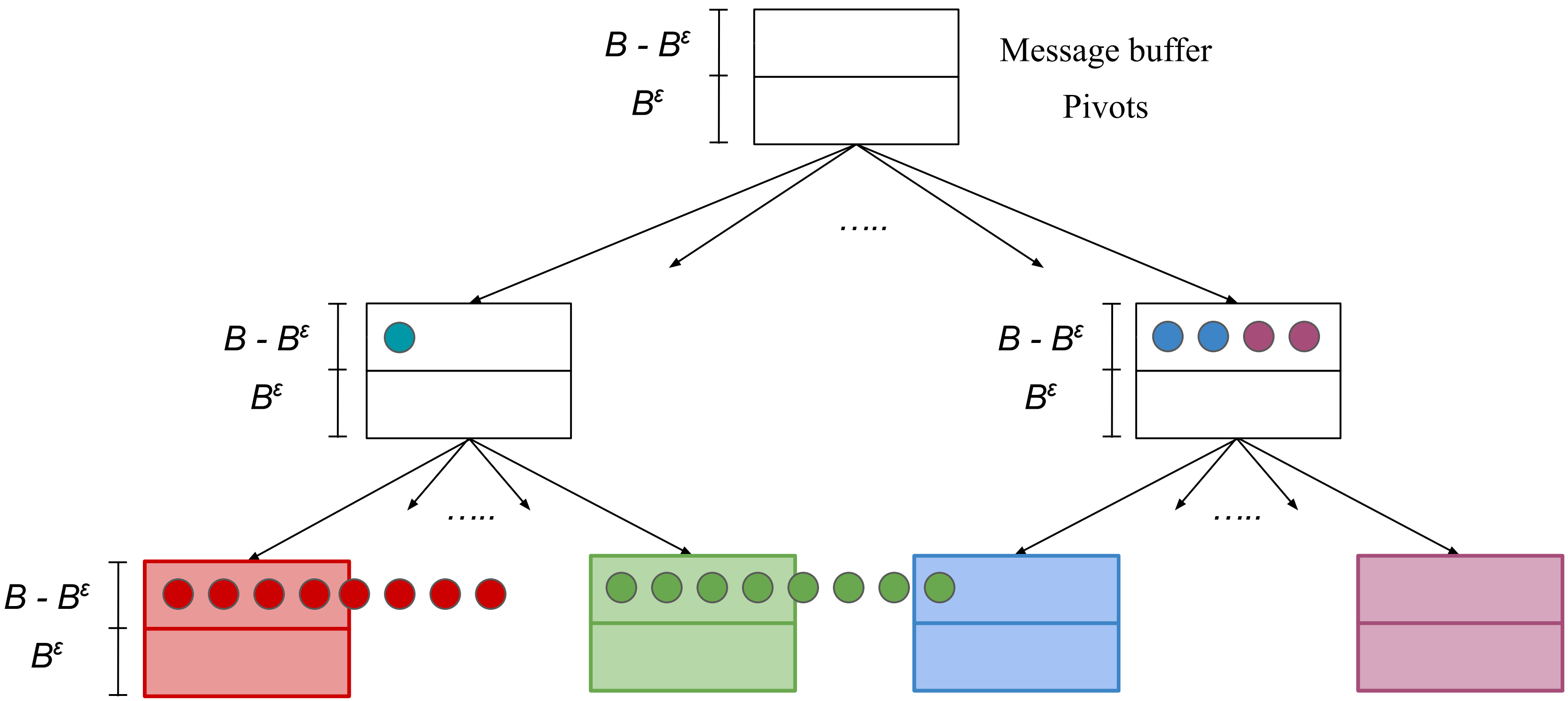
Avalanche



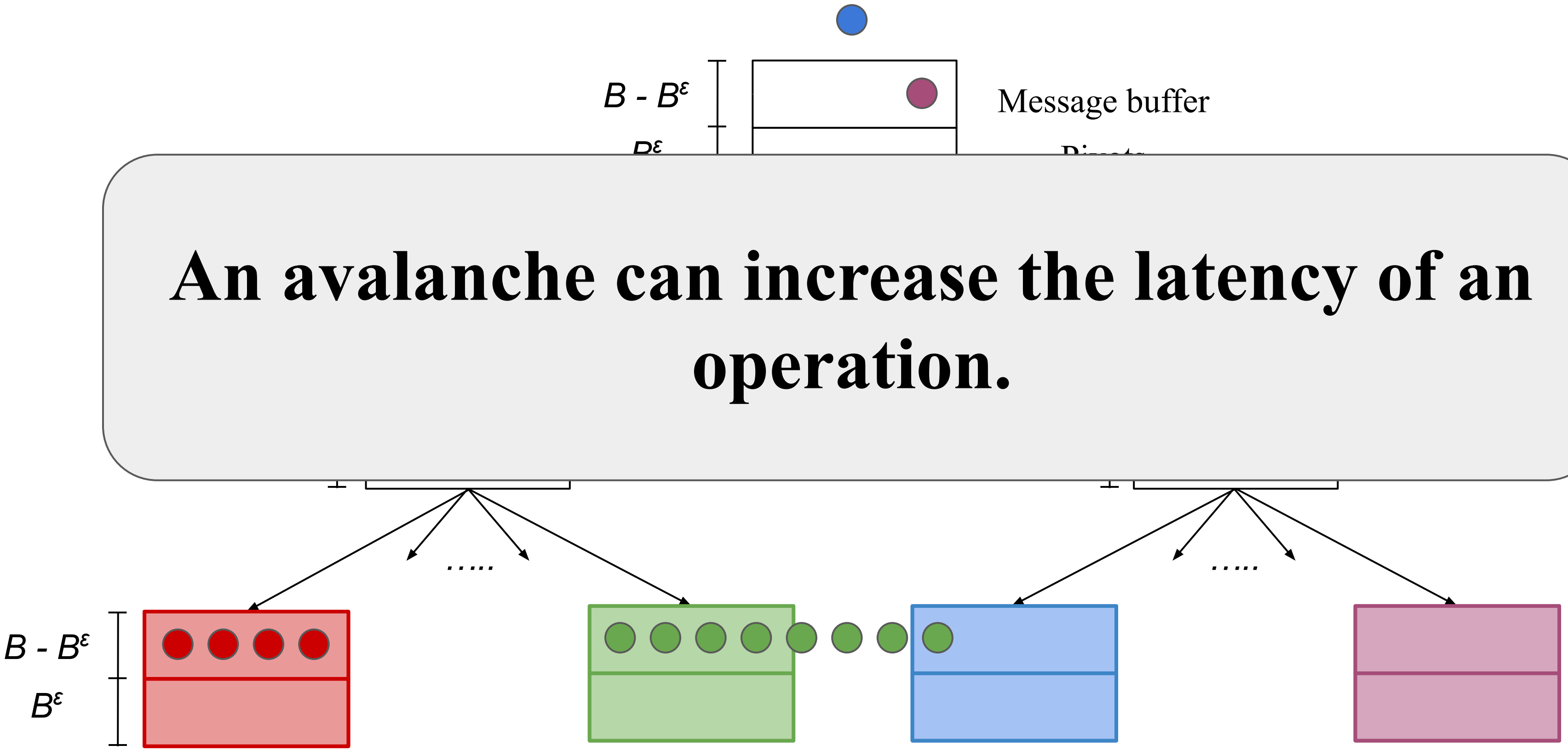
Avalanche



Avalanche

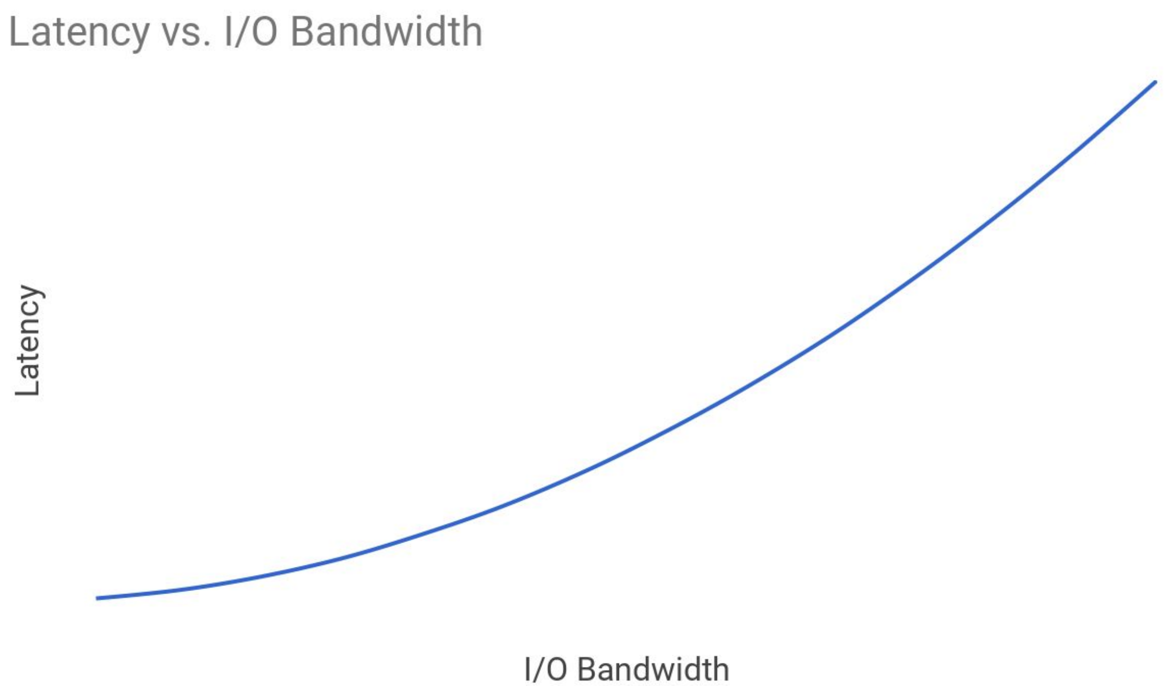


Avalanche



An avalanche can increase the latency of an operation.

Flushing tradeoff



- Flushing less number of messages to a child can result in sub-optimal I/O performance.
- Flushing a lot of messages to a child can cause an avalanche.

Scheduling Problem

- We now have a scheduling problem.
- Flushes are scheduled every $\varepsilon B^{1-\varepsilon} / \log_B N$ inserts.
- We can allow nodes to grow larger temporarily.

Is there a schedule in which if we pick a point and flush to a chosen child we can bound the maximum size of a node?

Possible Strategies to Pick the Child to Flush To?

- Pick the child to which you can flush the most number of messages.
- Pick the largest child such and find its sub-child where you can flush messages to resize the child without causing an avalanche.

References

- http://supertech.csail.mit.edu/papers/BenderFaJa15.pdf
- https://www.usenix.org/system/files/conference/fast15/fast15-paper-jannen_william.pdf
- https://www.usenix.org/system/files/conference/fast16/fast16-papers-yuan.pdf

Thank You!

Abstract

Write-optimized key-value stores, such as B^{ε}-trees, are the state-of-the-art key-value stores. B^{ε}-trees move data around in batches thereby amortizing the I/O cost of moving data.

During batch data moves in practice, we see an inherent tension between operation latency and I/O bandwidth utilization in B^{ε}-trees trees. This talk presents an open problem on how to schedule batch data moves in a B^{ε}-tree.