Buffered Partially-Persistent External-Memory Search Trees

— or, partially-persistent B-trees meet B^{ε}-trees

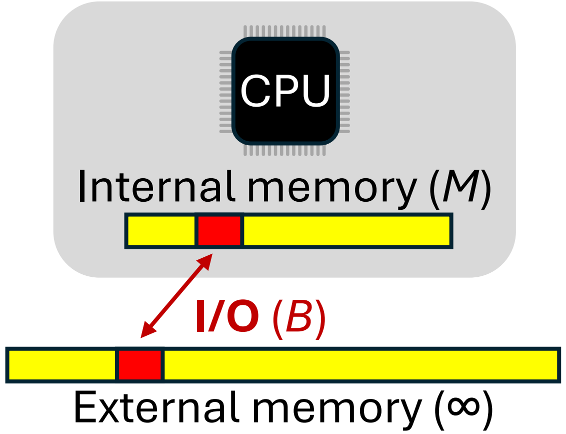
Gerth Stølting Brodal
Aarhus University

Casper Moldrup Rysgaard
Aarhus University

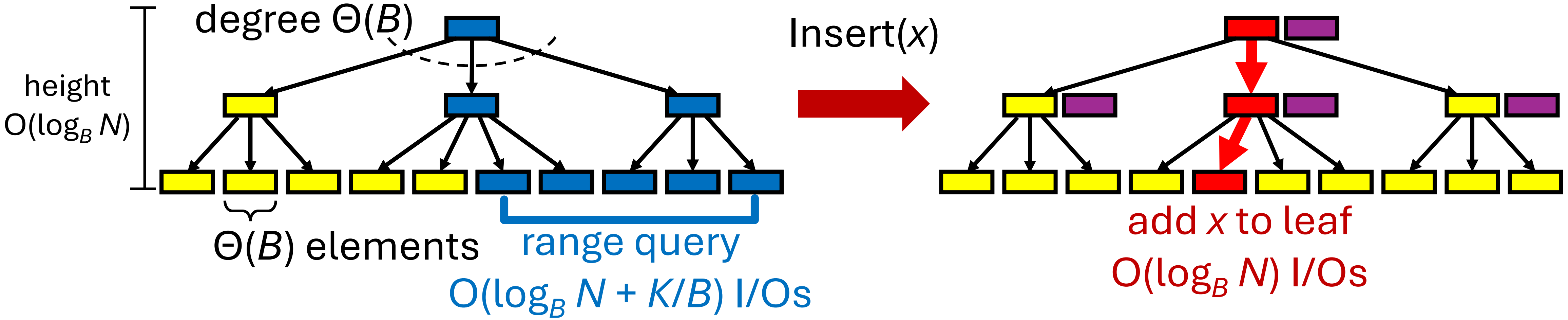
Rolf Svenning
Aarhus University

33rd Annual European Symposium on Algorithms (ESA), September 15, 2025, Warsaw, Poland

Buffered partially-persistent external-memory search trees



- Search tree (B-tree)



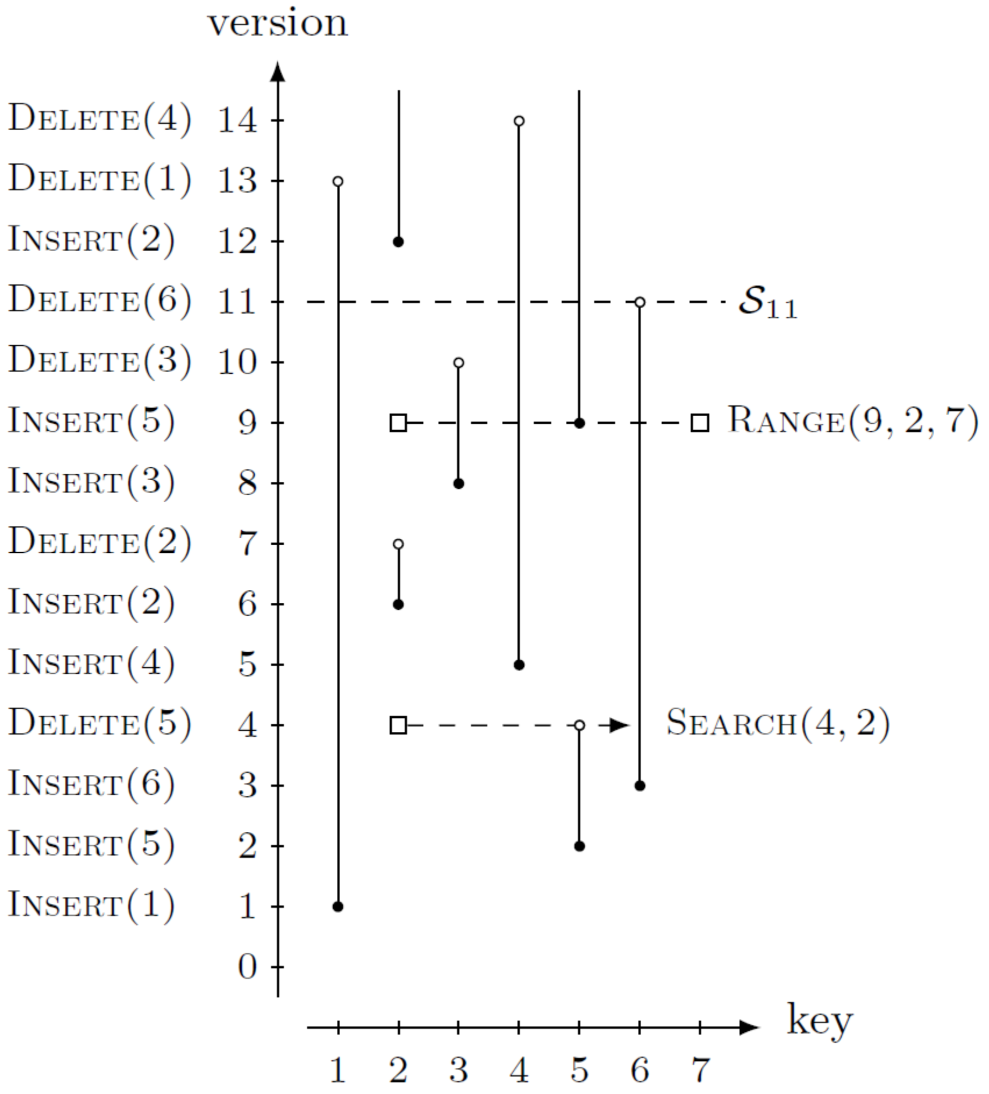
- **Partial persistence** = remember all previous versions
  - **Copy path** from root to updated nodes = space $O(\log_B N)$ blocks per update
  - Associate with each element and pointer a time interval where it is part of the structure (**fat node** and **node copying**)
- ***B^ε*-tree** – reduced degree B^ε, add **buffers**, blocks of updates

I/O results (all linear space)

Ephemeral	**Range query**	**Insert/Delete**	**Assumption**
B-tree Bayer, McCreight 1972	$O\left(\log_B N + \frac{K}{B}\right)$	$O(\log_B N)$	$M \geq 2B$
B^{ε}-tree a) Brodal, Fagerberg 2003 b) Bender et al. 2020 c) Das, Iacono, and Nekrich 2022	$O\left(\frac{1}{\varepsilon}\log_B N + \frac{K}{B}\right)$	$O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N\right)$	a) Amortized, $M \geq 2B$ b) Randomized, $B=\Omega(\log N)$ $M = \Omega(\max\{B^2, \log^{\Theta(1)} N, B^2\})$ c) Worst-case, $M = \Omega(B \log_B N)$
Partial persistence			
Partially-persistent B-tree Becker et al. 1996	$O\left(\log_B N + \frac{K}{B}\right)$	$O(\log_B N)$	$M \geq 2B$
This talk	$O\left(\frac{1}{\varepsilon}\log_B N + \frac{K}{B}\right)$	$O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N\right)$	a) Amortized, $M \geq 2B$ b) Worst-case, $M = \Omega(B^{1-\varepsilon} \log_2 N)$

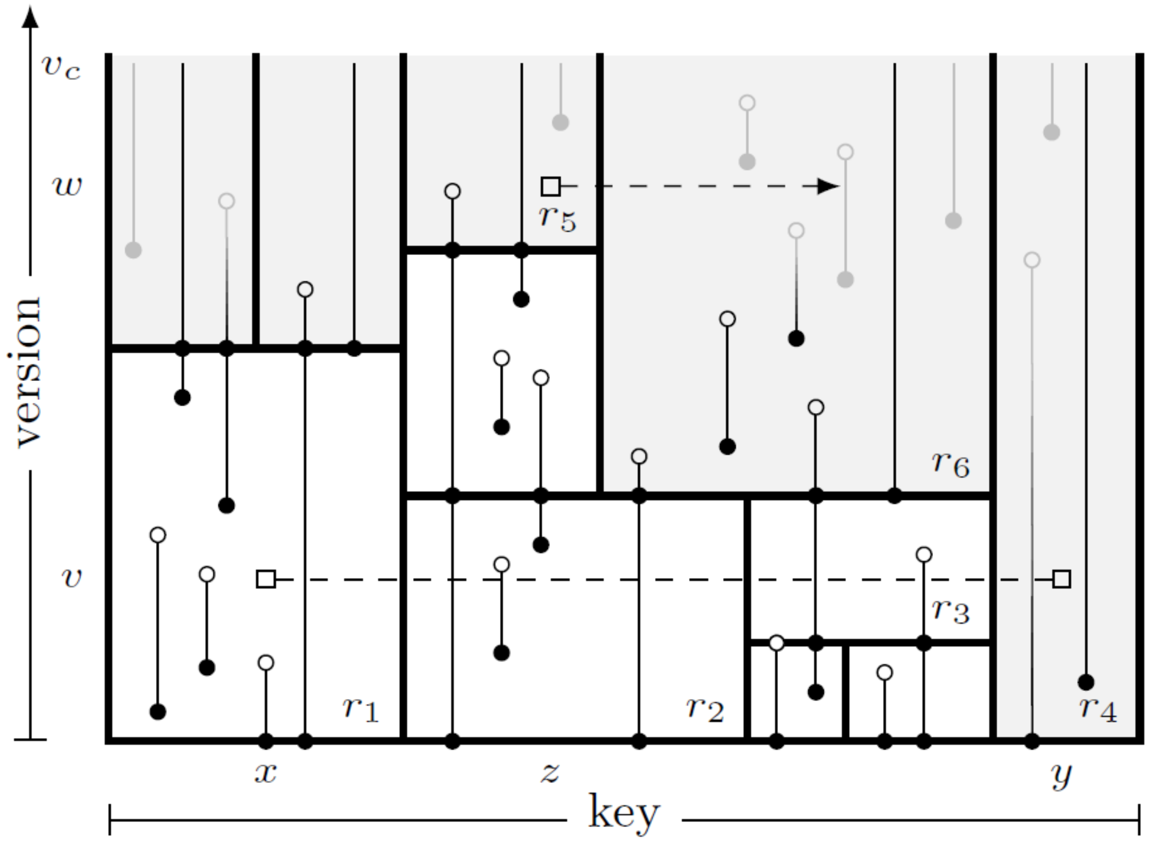
A geometric interpretation

- **Insertions** and **deletions** are endpoints of vertical segments
- **Search** and **range queries** are horizontal ray shooting and segment intersection queries



Rectangle partition

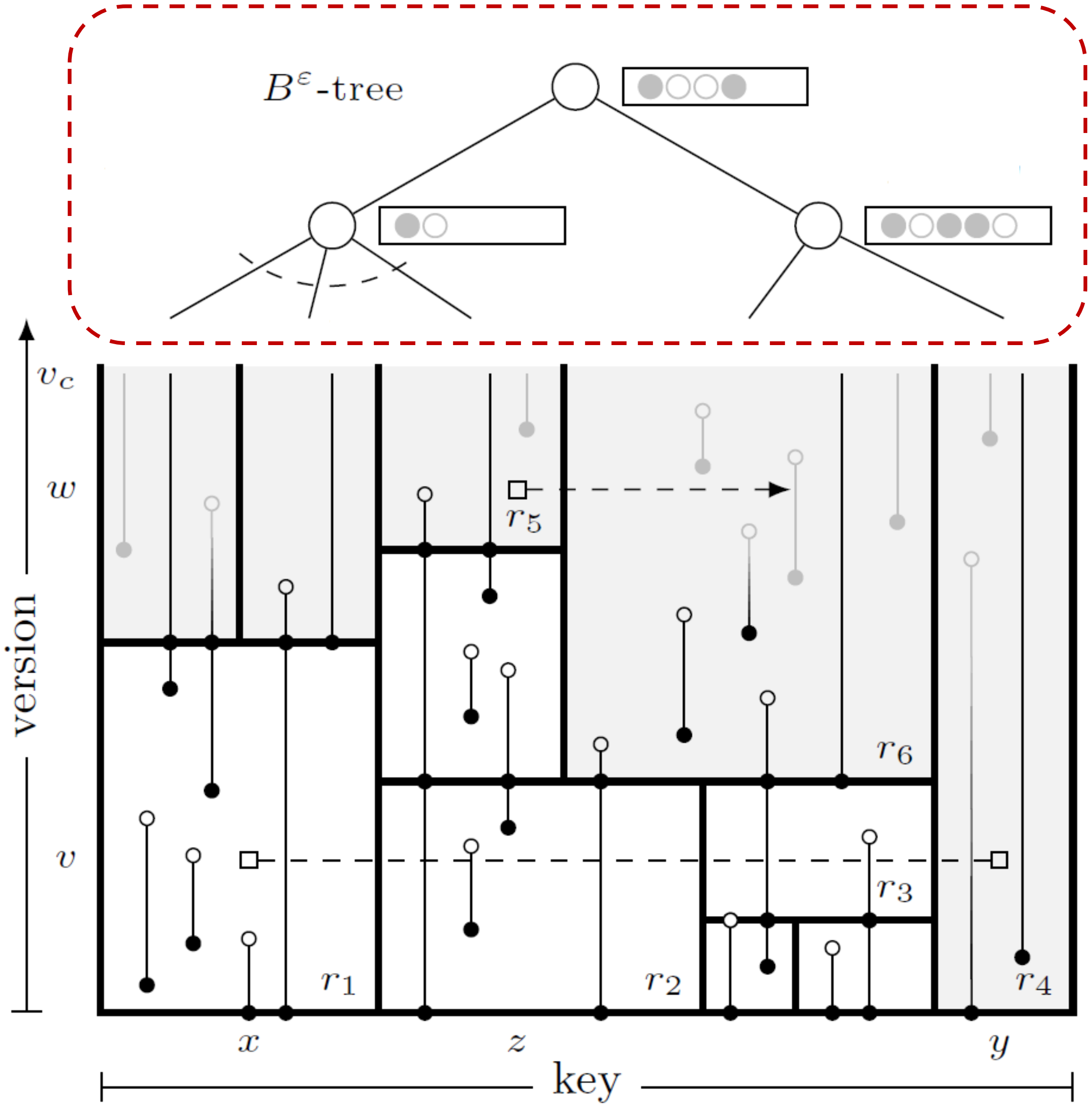
- Partition plan into **rectangles**
- Topmost rectangles are **open** (can received more updates)
- **Segments** crossing multiple rectangles are **split**
- Rectangles store **O($B \cdot \log_B N$) endpoints** where $\Omega(B \cdot \log_B N)$ segments **span** bottom-to-top
- **Global rebuild** when N doubles/halves



- Segments in a rectangle are **sorted by key**

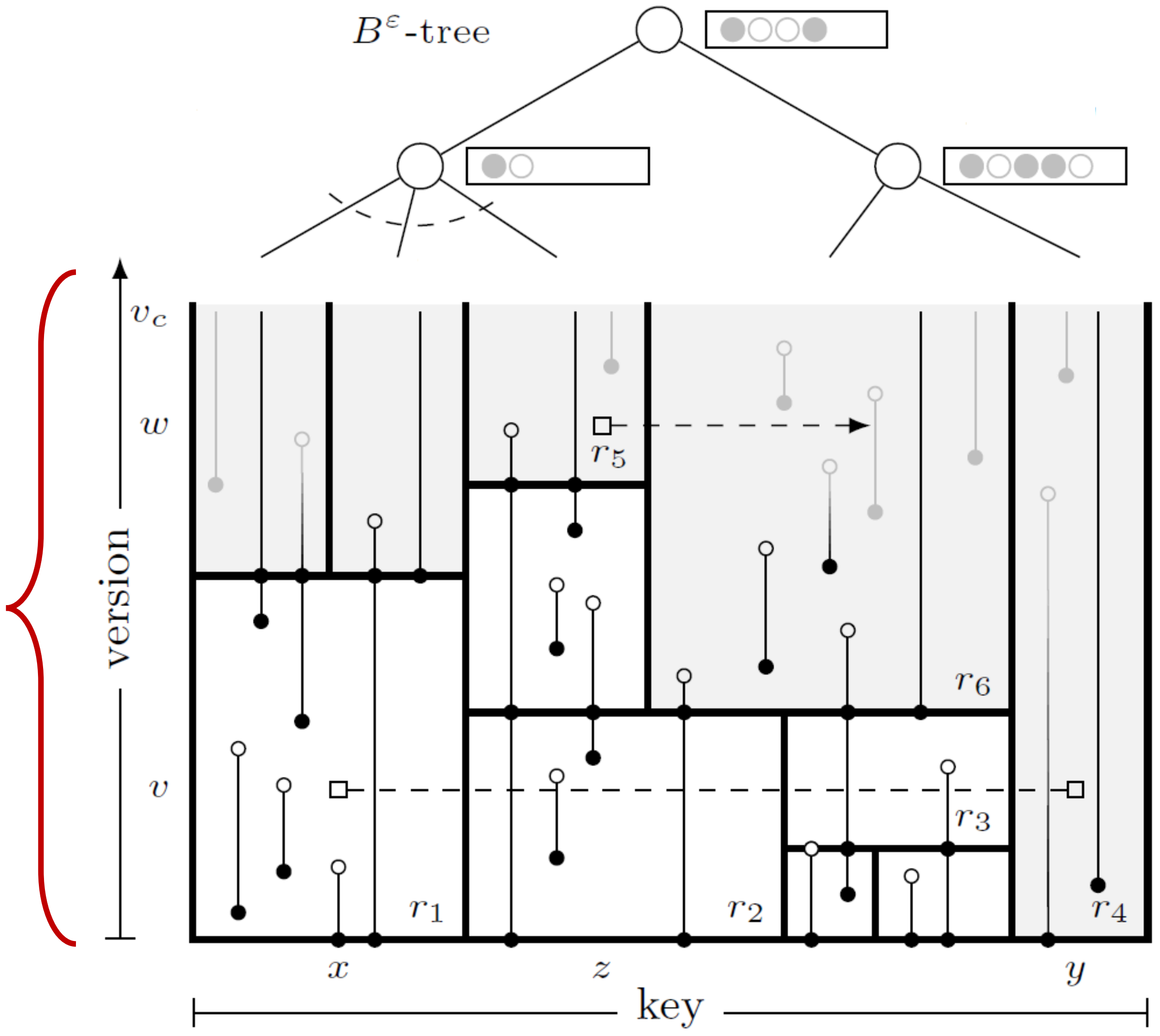
Top structure

- B^ε-tree with updates heading for the **open rectangles**
- Degree B^ε, buffer capacity B
- **Close** a rectangle when it has received $\Theta(B \log_B N)$ updates (move all buffered updates to it)
- **Join** and **split** open rectangles like in a B-tree (after closing them)

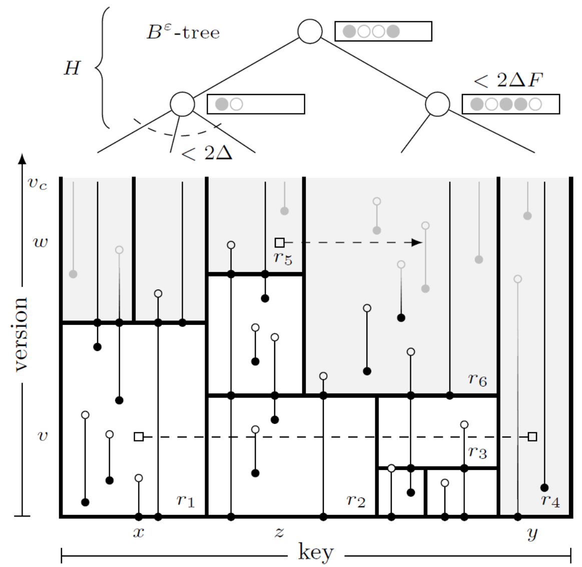


Point location

- **Queries** need to locate relevant rectangles
- For each version have a **B-tree** with the rectangles left-to-right intersection the version
- Make it partially-persistent using simple **path copying** for updates
- A **query** to an **open rectangle** moves all buffered updates to the rectangle



Achieving worst-case I/O bounds



- Buffer overflow = incremental **flush a single path**, pushing $B^{1\text{-}\varepsilon}$ updates to the child with most updates (ensures never more than $B^{1\text{-}\varepsilon} \cdot \log_2 B$ updates in a buffer heading for a single child)
- Incrementally **closing** an open rectangle, collect all buffered updates in internal memory, requires $\mathbf{M = \Omega(B^{1\text{-}\varepsilon} \cdot \log_2 N)}$
- **Incremental global rebuild** for changing N
- **Never merge internal nodes** (avoids flushing buffers, incremental global rebuilding bounds height)

Summary

- Linear space buffered partially-persistent B-trees, with I/O bounds

Range queries

$$O\left(\frac{1}{\varepsilon}\log_B N + \frac{K}{B}\right)$$

Insert/Delete

$$O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N\right)$$

a) Amortized, $M \geq 2B$
b) Worst-case, $M = \Omega(B^{1-\varepsilon}\log_2 N)$

- **Open problems**
  - Worst-case for $M \geq 2B$
  - Let fully-persistent B-trees meet B^{ε}-trees