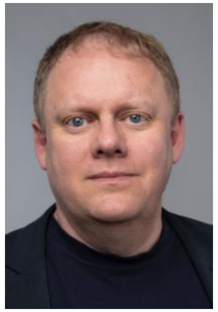
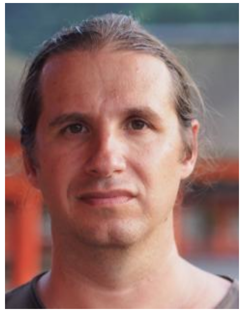
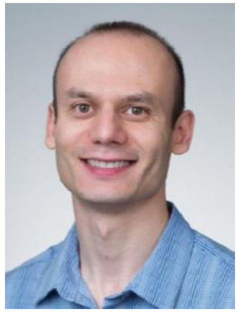
External-Memory Priority Queues and Persistent Search Trees

Gerth Stølting Brodal
Aarhus University





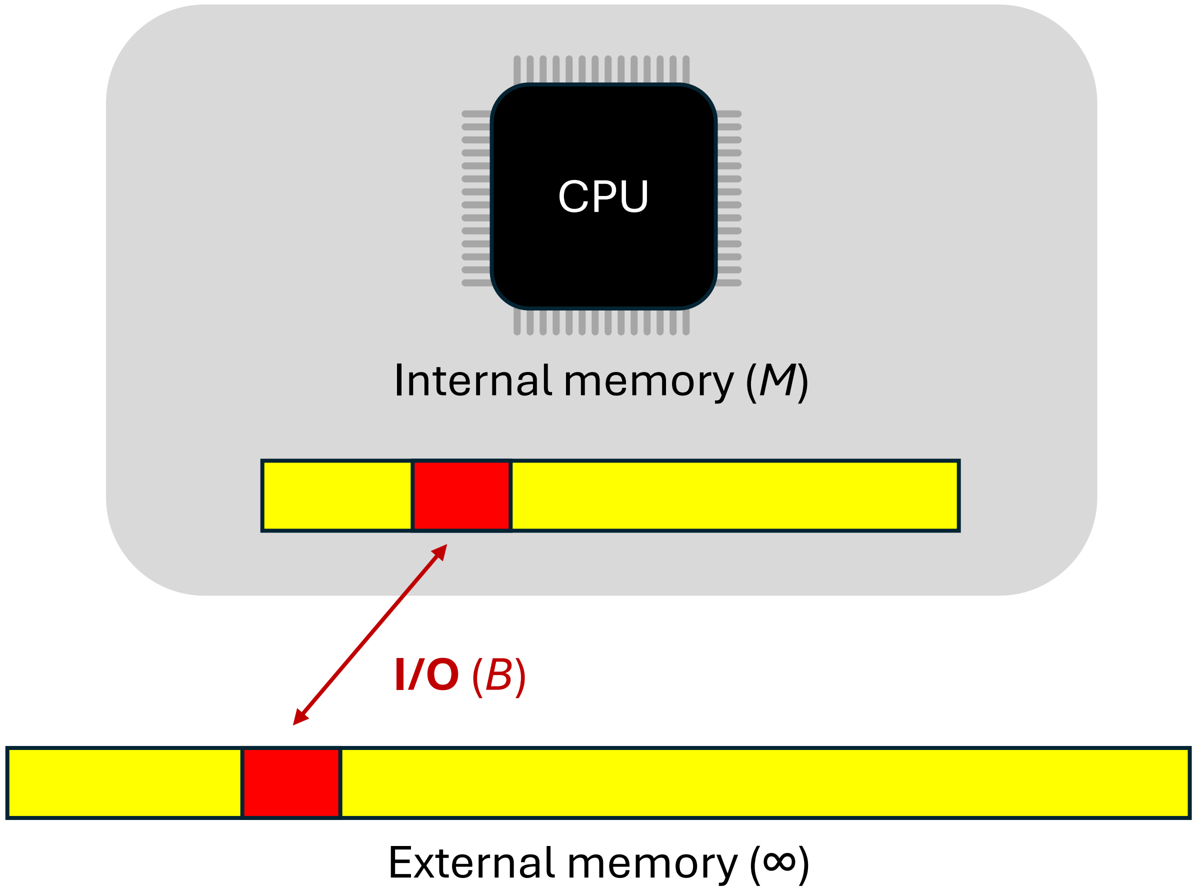


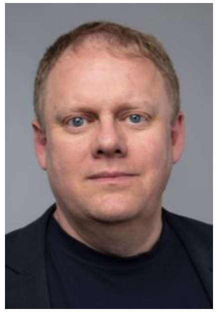

University of Primorska, Koper, Slovenia, December 8, 2025

External-Memory Model

Aggarwal, Vitter [CACM 1988]

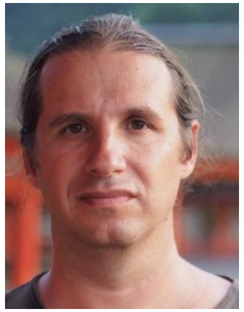


Part I
External-Memory Priority Queues with Optimal Insertions



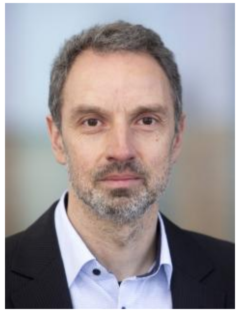
Gerth Stølting Brodal
Aarhus University

Michael Goodrich
University of California, Irvine



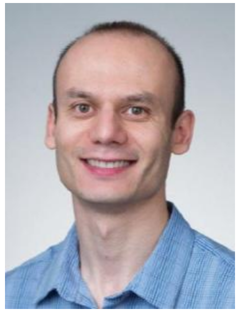
John Iacono
Université libre de Bruxelles

Jared Lo
University of Hawai'i at Mānoa



Ulrich Meyer
Goethe University Frankfurt am Main

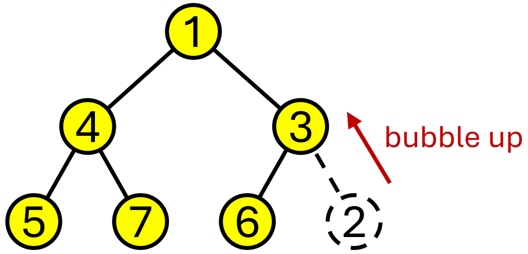
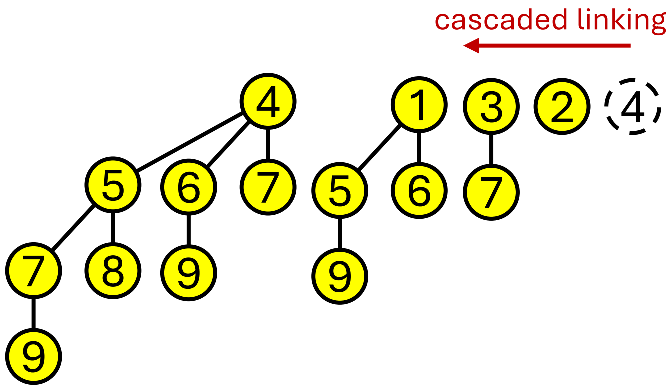
Victor Pagan
University of Hawai'i at Mānoa



Nodari Sitchinava
University of Hawai'i at Mānoa

Rolf Svenning
Aarhus University

Priority Queues

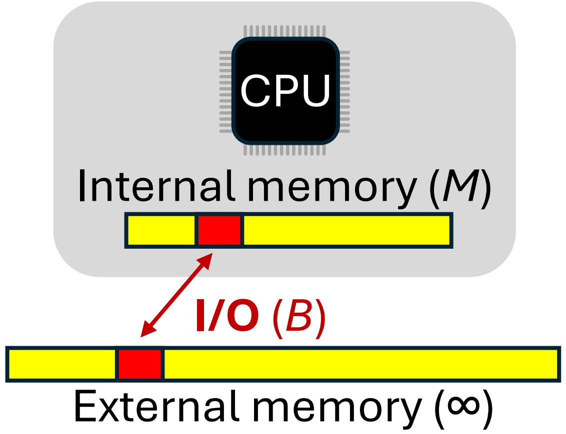
		INSERT(*e*)	DELETEMIN()
Binary heap (Williams 1964)		O(log *N*)	O(log *N*)
Binomial queue (Vuillemin 1978)		$O_A(1)$	O(log *N*)

Can we achieve a similar result in external memory ?

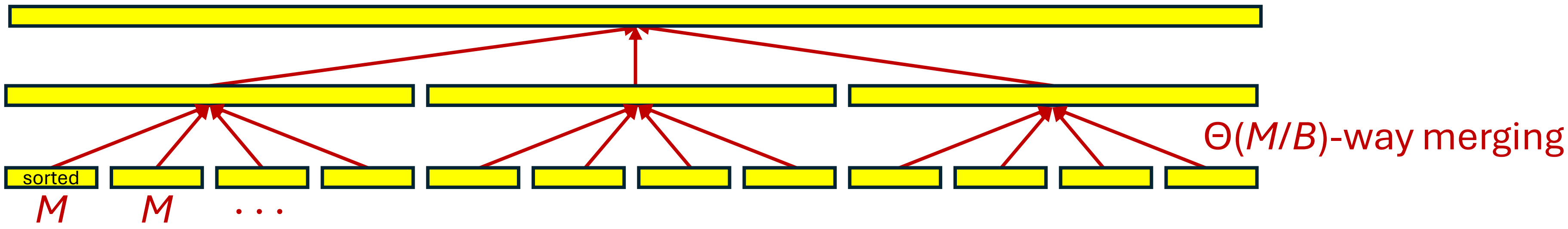
Sorting = *N* × INSERT + *N* × DELETEMIN ⇒ INSERT **or** DELETEMIN Ω(log *N*) comparisons

INSERT ≥ # DELETEMIN

External Memory

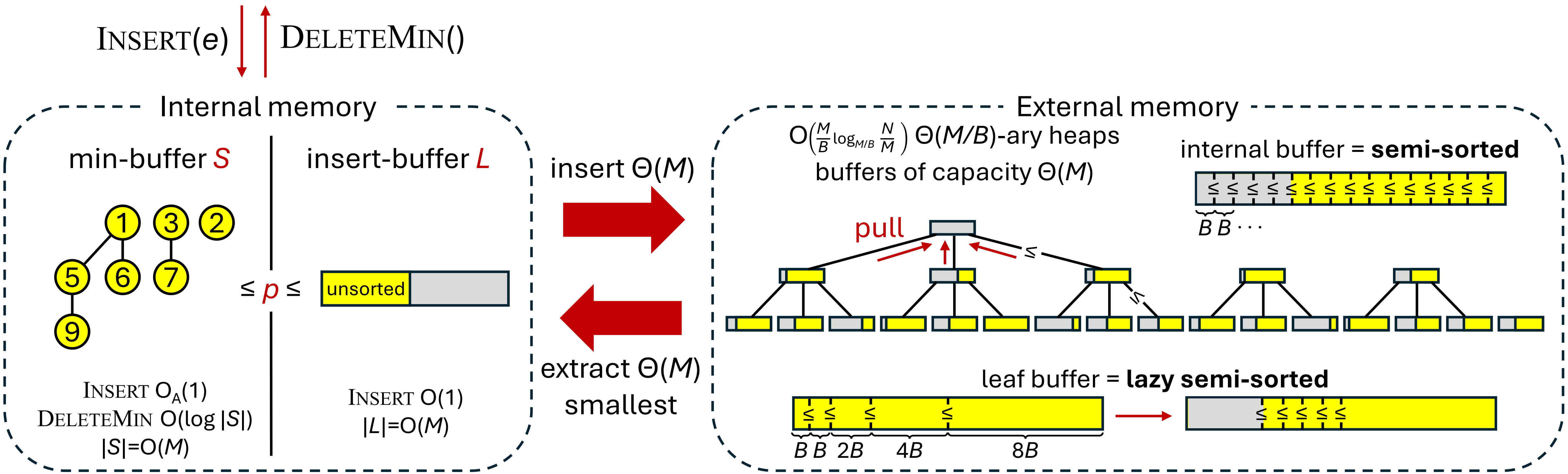


- External-memory model (Aggarwal, Vitter 1988)
- **Sorting** $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os and $O(N \cdot \log N)$ comparisons



- **Priority queue** $O_A\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os and $O_A(\log N)$ comparisons $\begin{cases}\textsc{Insert}\\\textsc{DeleteMin}\end{cases}$
  - **Buffer tree** (Arge 1995)
  - **Θ(*M*/*B*)-ary heap** (Kumar, Schwabe 1996; Fadel, Jakobsen, Katajainen, Teuhola 1997)
  - **Merging sorted lists** (Brengel, Crauser, Ferragina, Meyer 1999; Brodal, Katajainen 1998)
- **This talk** $\textsc{Insert}$ $O_A\left(\frac{1}{B}\right)$ I/Os and $O_A(1)$ comparisons

New External-Memory Priority Queue



Lemma Total $O\left(\frac{N/M}{M/B} + \frac{\#\,\text{deletions}}{M} \cdot \log_{M/B} \frac{N}{B}\right)$ pulls

Lemma Pulling M elements from M/B semi-sorted lists : $O(M/B)$ I/Os and $O\left(M \cdot \log_2 \frac{M}{B}\right)$ comparisons

Theorem Total $O\left(\frac{N}{B} + \frac{\#\,\text{deletions}}{B} \cdot \log_{M/B} \frac{N}{B}\right)$ I/Os and $O(N + \#\,\text{deletions} \cdot \log_2 N)$ comparisons

Summary – Part I

- **New optimal external memory priority queue**

	I/Os	Comparisons
INSERT	$O_A\left(\frac{1}{B}\right)$	$O_A(1)$
DELETEMIN	$O_A\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$	$O_A(\log N)$

- **Open problems**
  - Our data structure is inherently amortized – can it be made **worst-case** ?
  - Structure inherently cache-aware – is there a **cache-oblivious** data structure with matching I/O and comparison bounds ?

Frigo, Leiserson, Prokop, Ramachandran, *Cache-Oblivious Algorithms*, FOCS 1999

Part II
Buffered Partially-Persistent External-Memory Search Trees

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

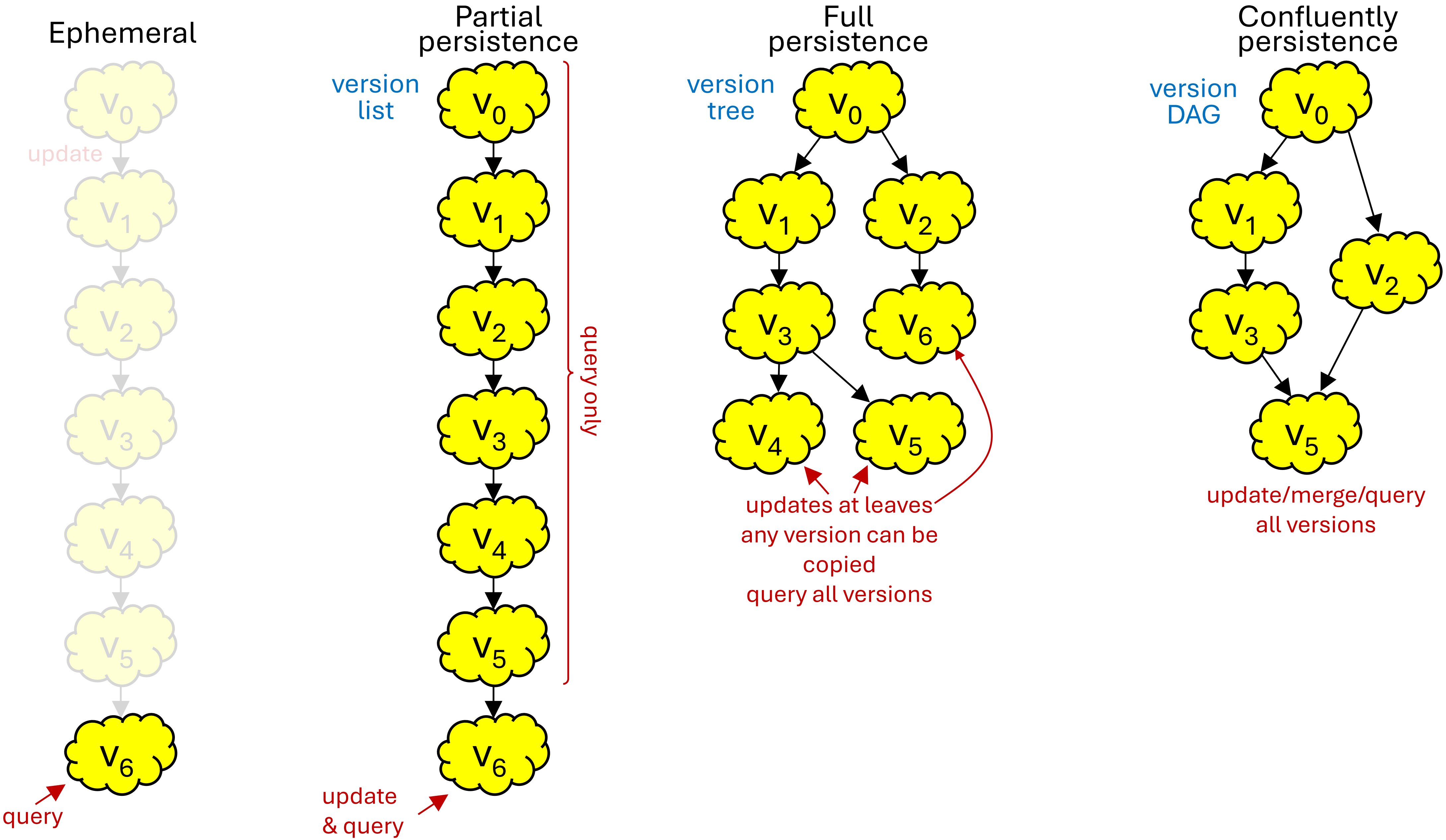
Gerth Stølting Brodal
Aarhus University

Casper Moldrup Rysgaard
Aarhus University

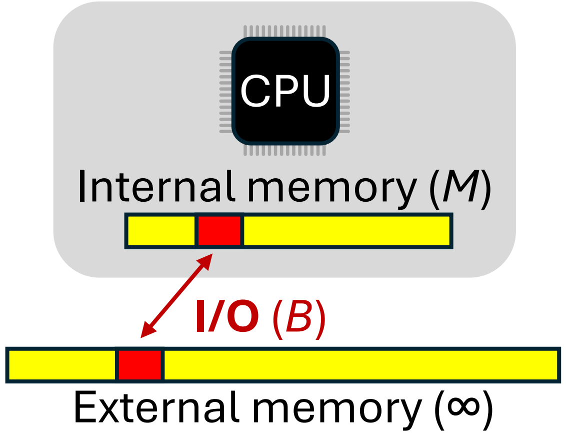
Rolf Svenning
Aarhus University

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

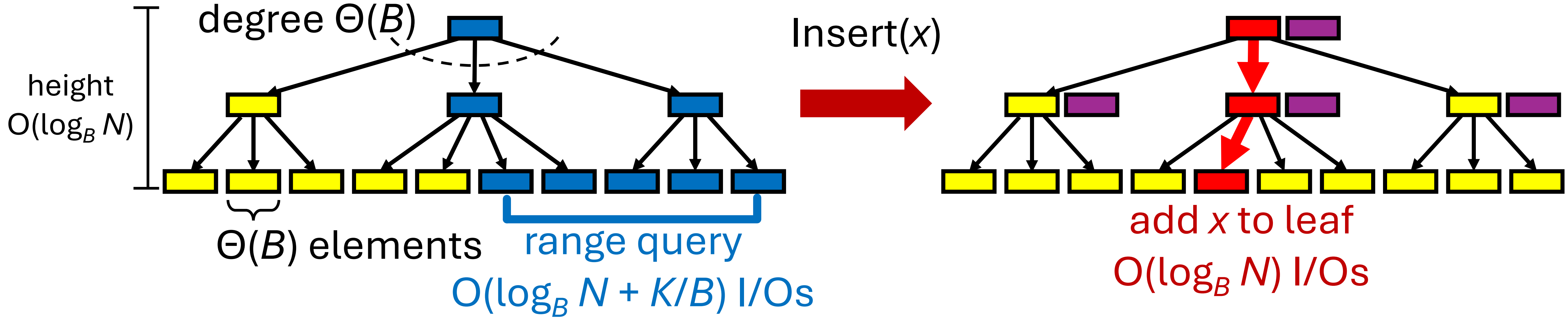
Persistent Data Structures



Buffered partially-persistent external-memory search trees



- Search tree (B-tree, Bayer & McCreight 1972)



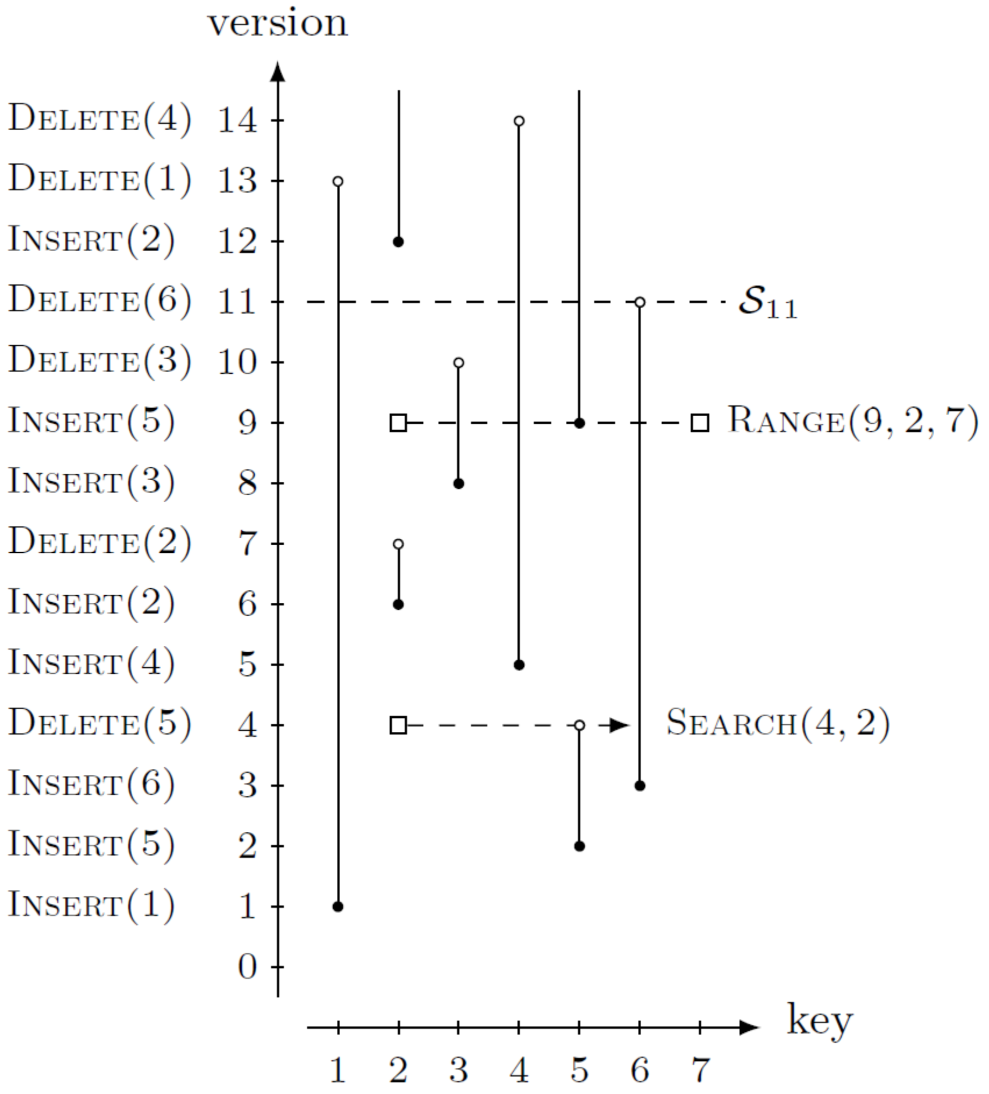
- **Partial persistence** = remember all previous versions
  - **Copy path** from root to updated nodes = space $O(\log_B N)$ blocks per update
  - Associate with each element and pointer a time interval where it is part of the structure (**fat node** and **node copying**)
- ***B*^ε-tree** – reduced degree B^ε, add **buffers** = blocks of delayed 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, Das, Farach-Colton, Johnson, and Kuszmaul 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, Gschwind, Ohler, Seeger, and Widmayer 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, $M = \Omega(B^{1-\varepsilon}\log_2 N)$

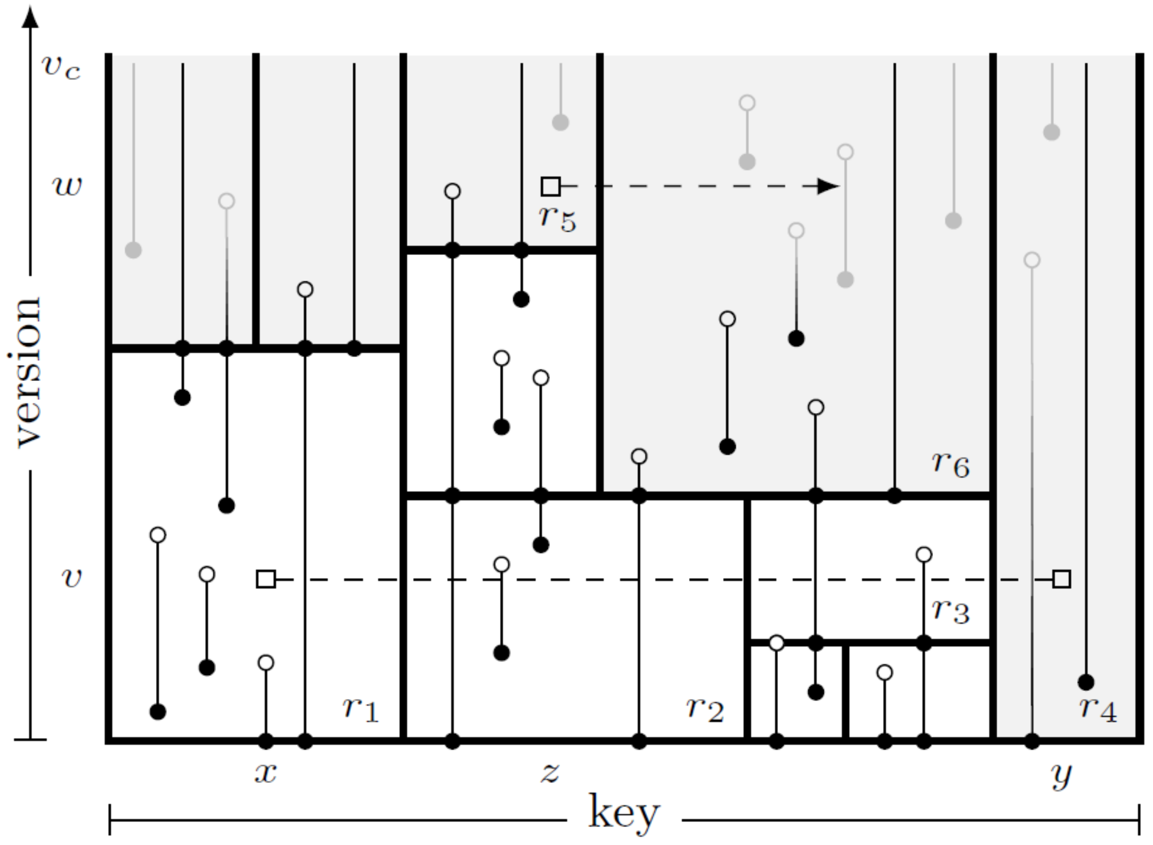
Geometric interpretation

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



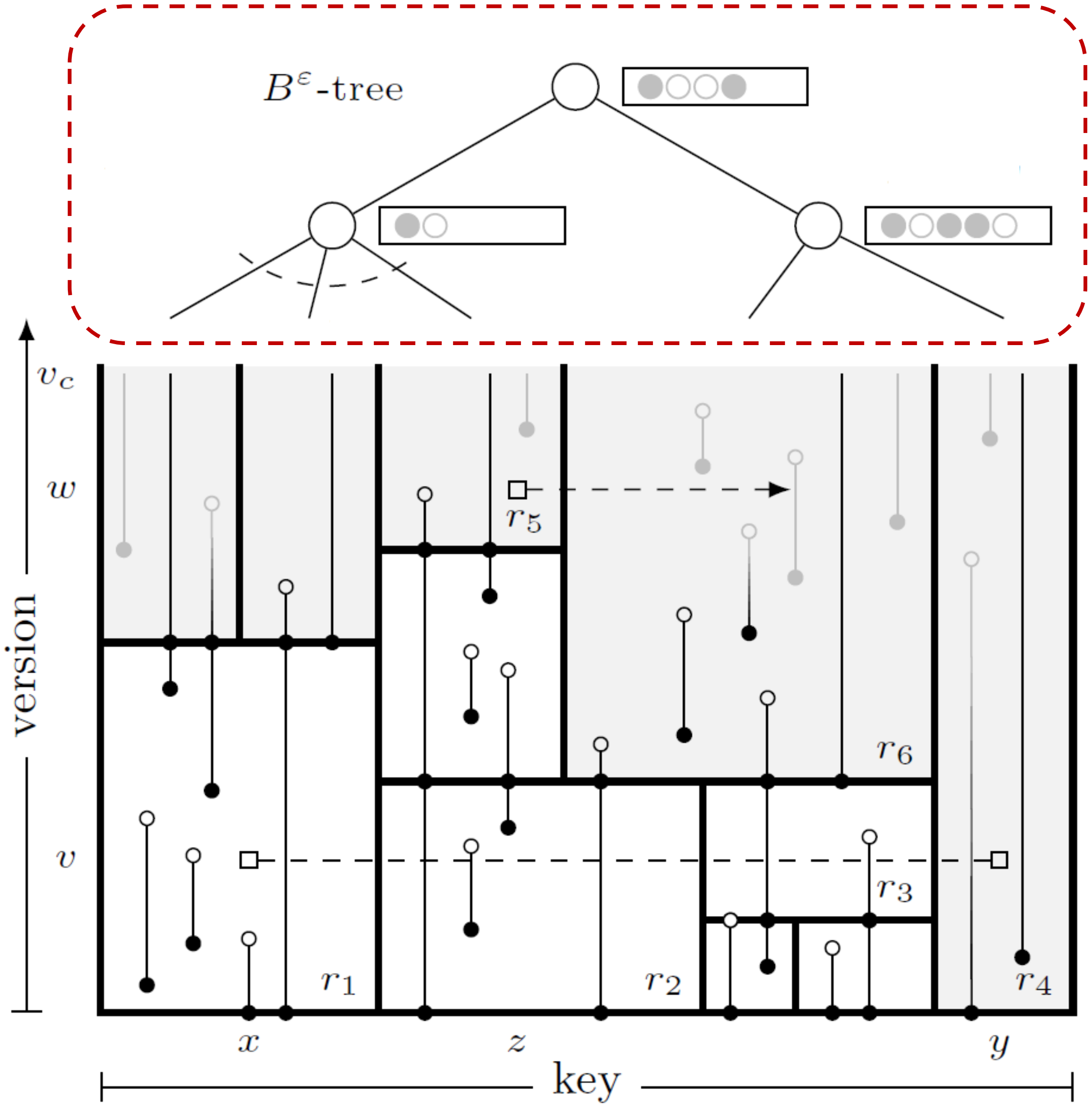
Rectangle partition

- Partition plan into **rectangles**
- Topmost rectangles are **open** (can received more updates)
- **Segments** crossing multiple rectangles are **split**
- Rectangles store **$\Theta(B \cdot \log_B N)$ 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**



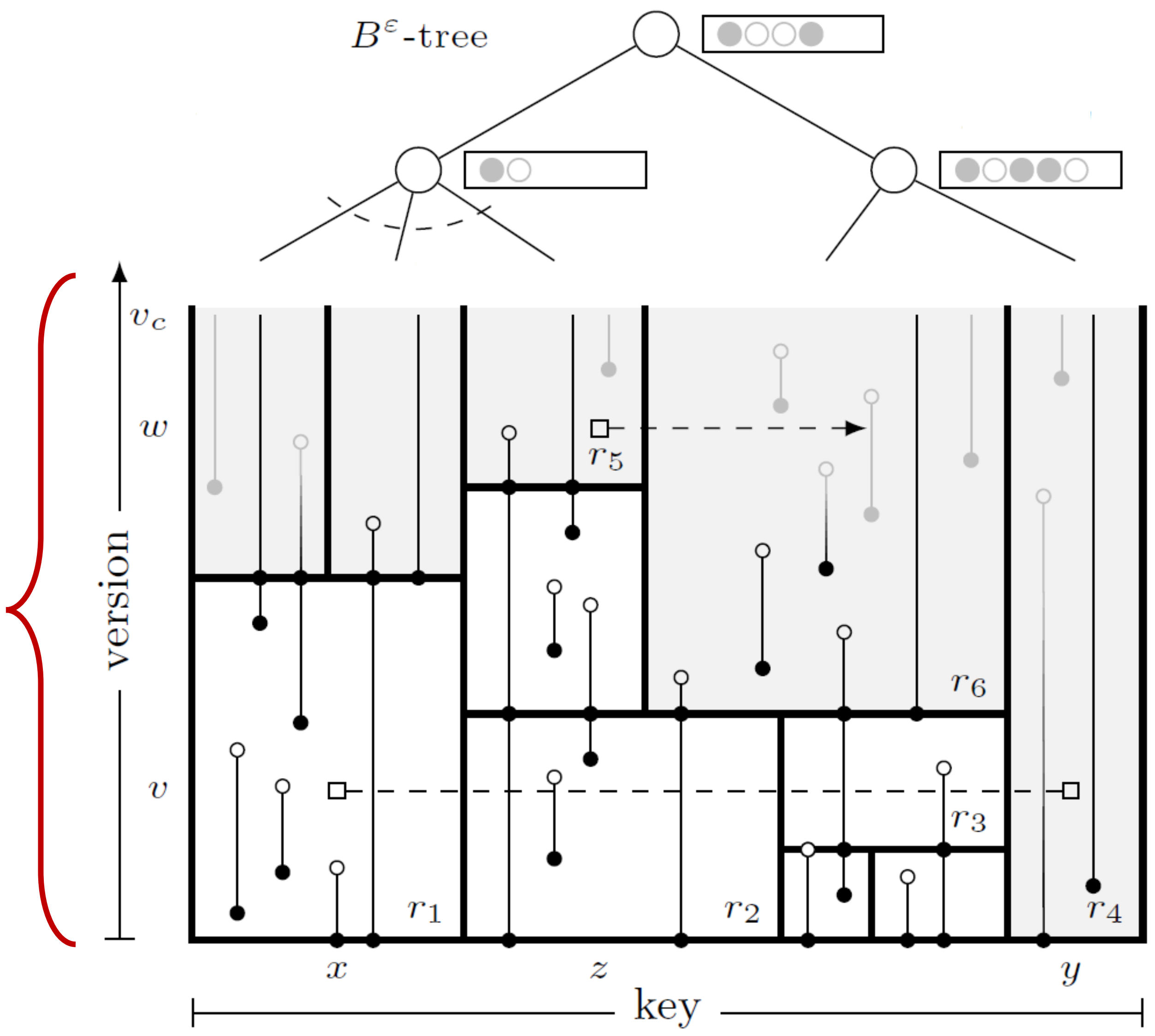
Top structure

- 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)

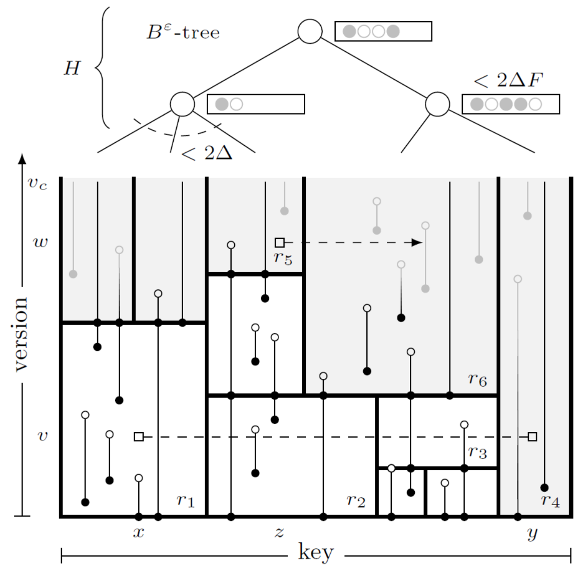


Point location

- **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** in top structure (avoids flushing buffers, incremental global rebuilding bounds height)

Summary – Part II

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

Range queries	**Insert/Delete**	
$O\left(\frac{1}{\varepsilon}\log_B N + \frac{K}{B}\right)$	$O\left(\frac{1}{\varepsilon B^{1-\varepsilon}}\log_B N\right)$	a) Amortized, $M \geq 2B$ b) Worst-case, $M = \Omega(B^{1-\varepsilon}\log_2 N)$

- **Open problems**
  - Worst-case for $M \geq 2B$
  - Let fully-persistent B-trees meet B^{ε}-trees
    (Brodal, Rysgaard, Svenning, STOC 2023, $O(\log_B N)$ I/Os queries & updates)

Thanks

Gerth Stølting Brodal
gerth@cs.au.dk

- Brodal, Goodrich, Iacono, Lo, Meyer, Pagan, Sitchinava, Svenning, *External-Memory Priority Queues with Optimal Insertions*, ESA 2025
- Brodal, Rysgaard, Svenning, *Buffered Partially-Persistent External-Memory Search Trees,* ESA 2025