UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

GreenBST: Energy-Efficient Concurrent Search Tree

Ibrahim Umar, Otto J. Anshus, Phuong H. Ha

Arctic Green Computing Lab
Department of Computer Science
UiT The Arctic University of Norway

Outline of the talk

- Background
- GreenBST: Energy-efficient concurrent search tree
- Evaluation
- Conclusion

BACKGROUND

Motivation

- The energy consumption of computing systems are mostly dominated by the **cost of data movement** [1]

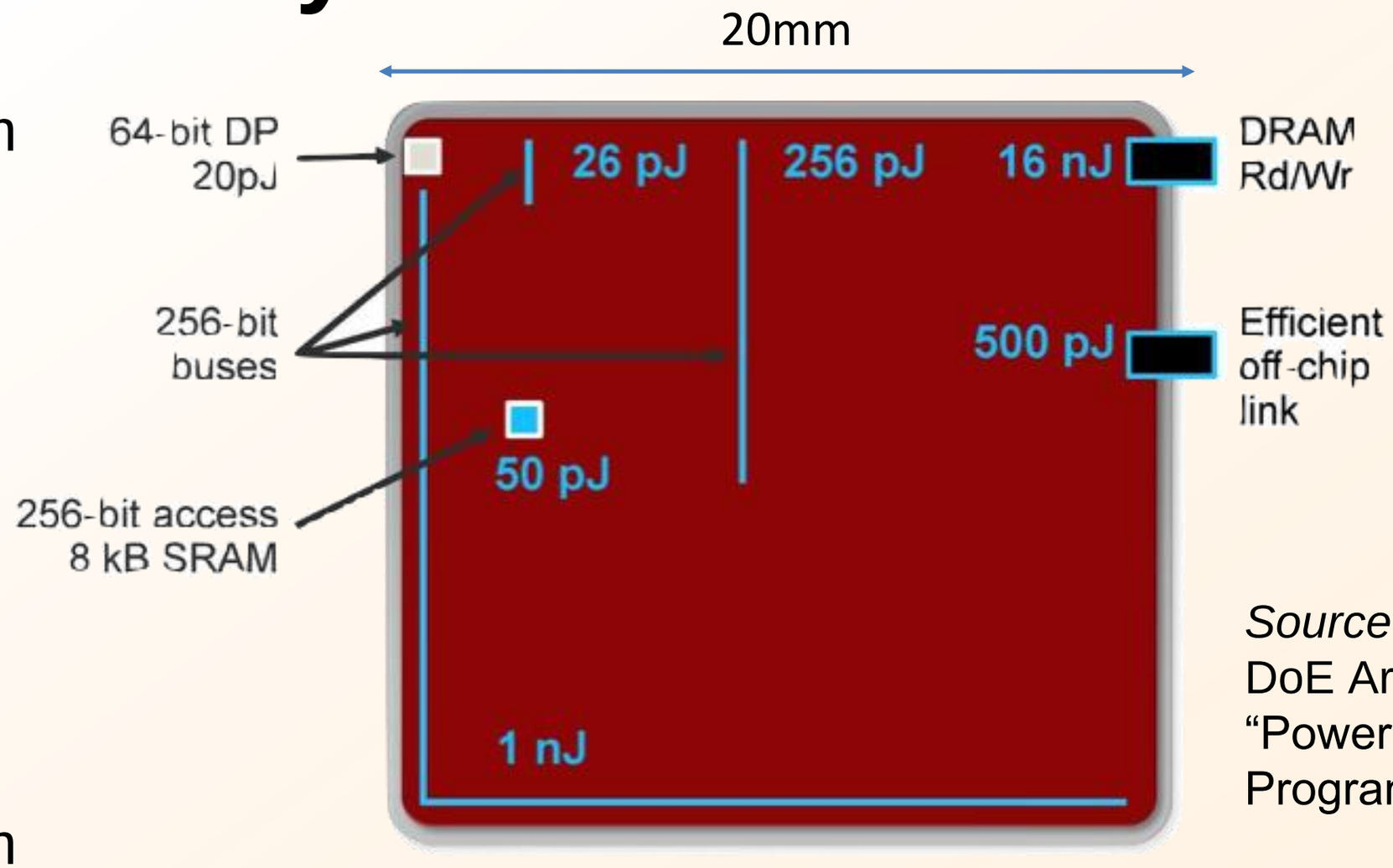
- Data locality in *finer-granularity* can bring greater energy savings to computing systems [2]
  - Fine grained locality: not only between CPU and RAM, but between memory hierarchies inside the CPU (L1 cache, L2C, L3C, ...)

- It is **important** for future data structures and algorithms to utilize **fine-grained data locality** and **concurrency**

[1] J. Choi, M. Dukhan, X. Liu and R. Vuduc, "Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks," Parallel and Distributed Processing Symposium, IPDPS 2014, pp. 447-457
[2] Dally, B.: Power and programmability: The challenges of exascale computing. In: DoE Arch-I presentation (2011)

Fine-grained data locality is an opportunity

- 28nm

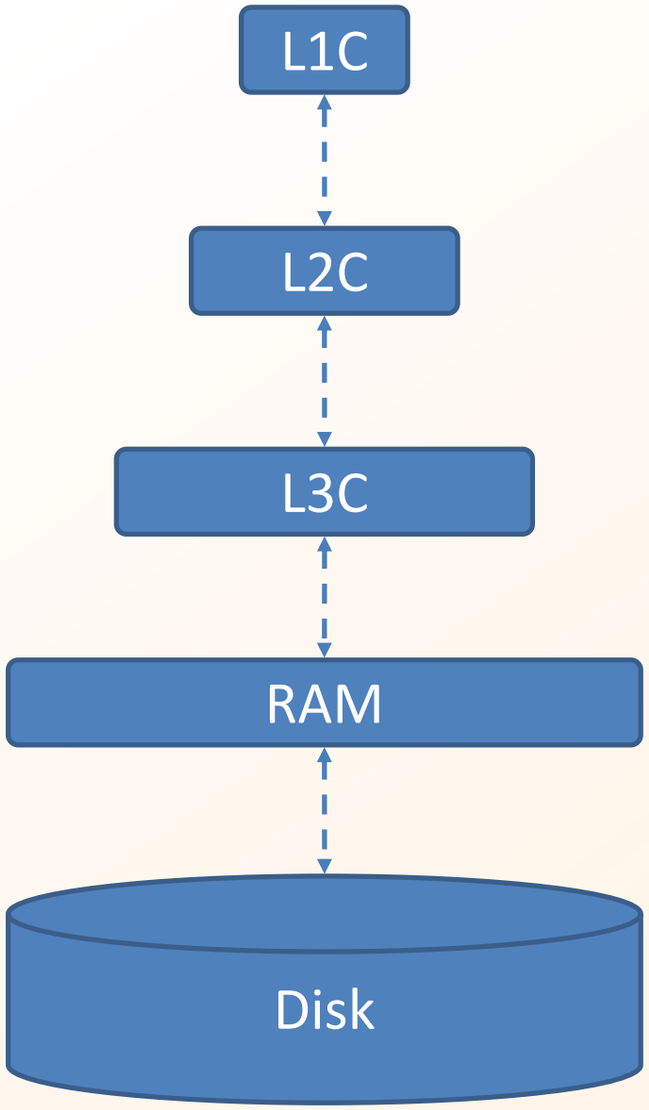


Source: Bill Dally, DoE Arch-I'11 "Power and Programmability"

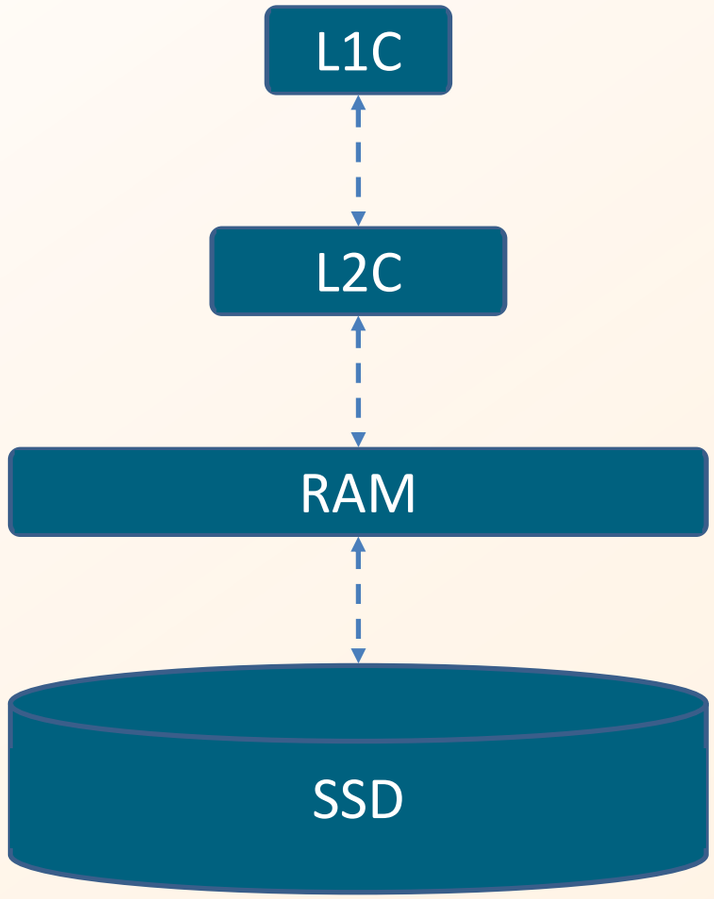
- 10nm

  Bulk of data should be accessed from **nearby memories (2pJ)**, not **across the chip (150pJ)**, **off chip (300pJ)** or **across the system (1nJ)** [2]

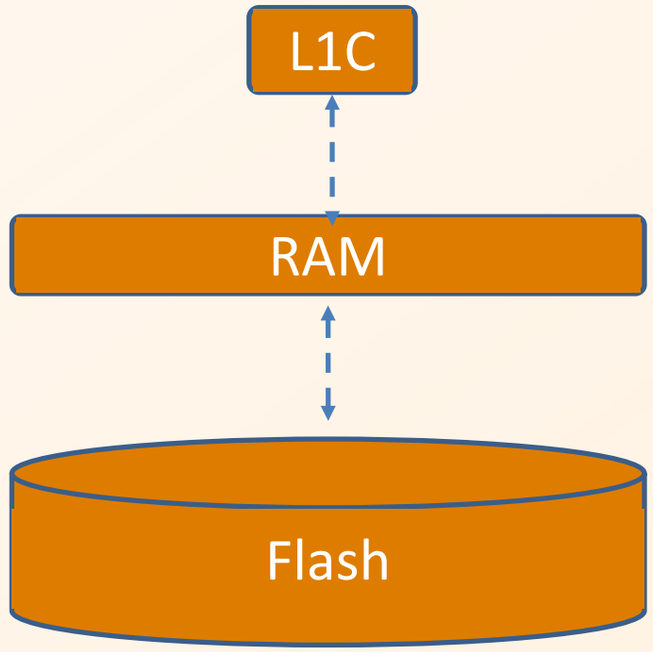
Fine-grained data locality on multiple platforms



Platform A



Platform B



Platform C

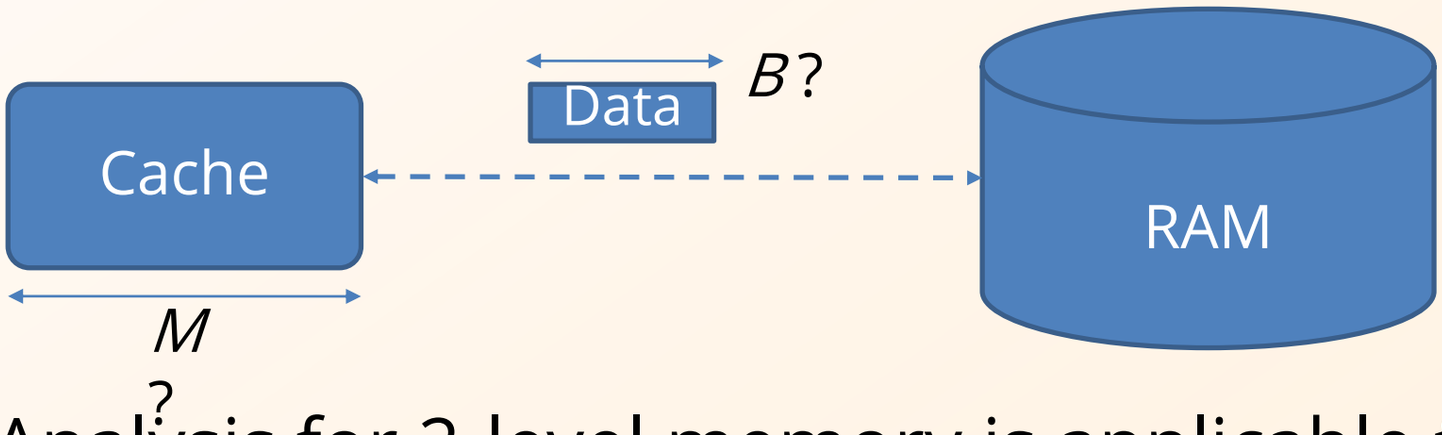
Be **oblivious** =

Cache oblivious!

The cache-oblivious model

- Block transfers dominates the execution time
  - Goal: minimize the number of data block transfers

- Cache-oblivious (CO) model [3]
  - Cache size M and block size B are unknown



- Analysis for 2-level memory is applicable for unknown multilevel memory (register, L1C, L2C, … ,LLC, memory).

[3] Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Annual Symp. Foundations of Computer Science. p. 285. FOCS ’99 (1999)

Search trees

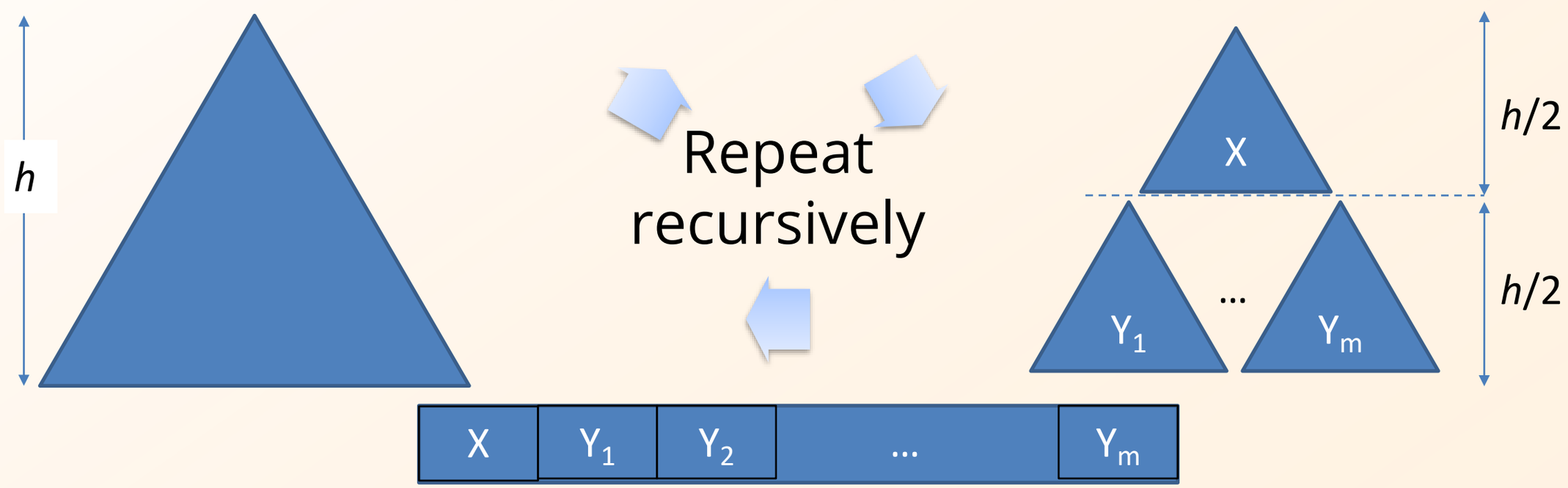
- Search trees are one of the important data structure for High Performance Systems (HPC)

- Example usage:
  - Databases (*PostgresSQL*, *CouchDB*)
  - Filesystems (*Btrfs*, *F2FS*)
  - Schedulers (the Completely Fair Scheduler (*CFS*))

- **Energy-efficient search tree** is a step towards an energy-efficient system

Cache-oblivious search trees: The van Emde Boas (vEB) layout

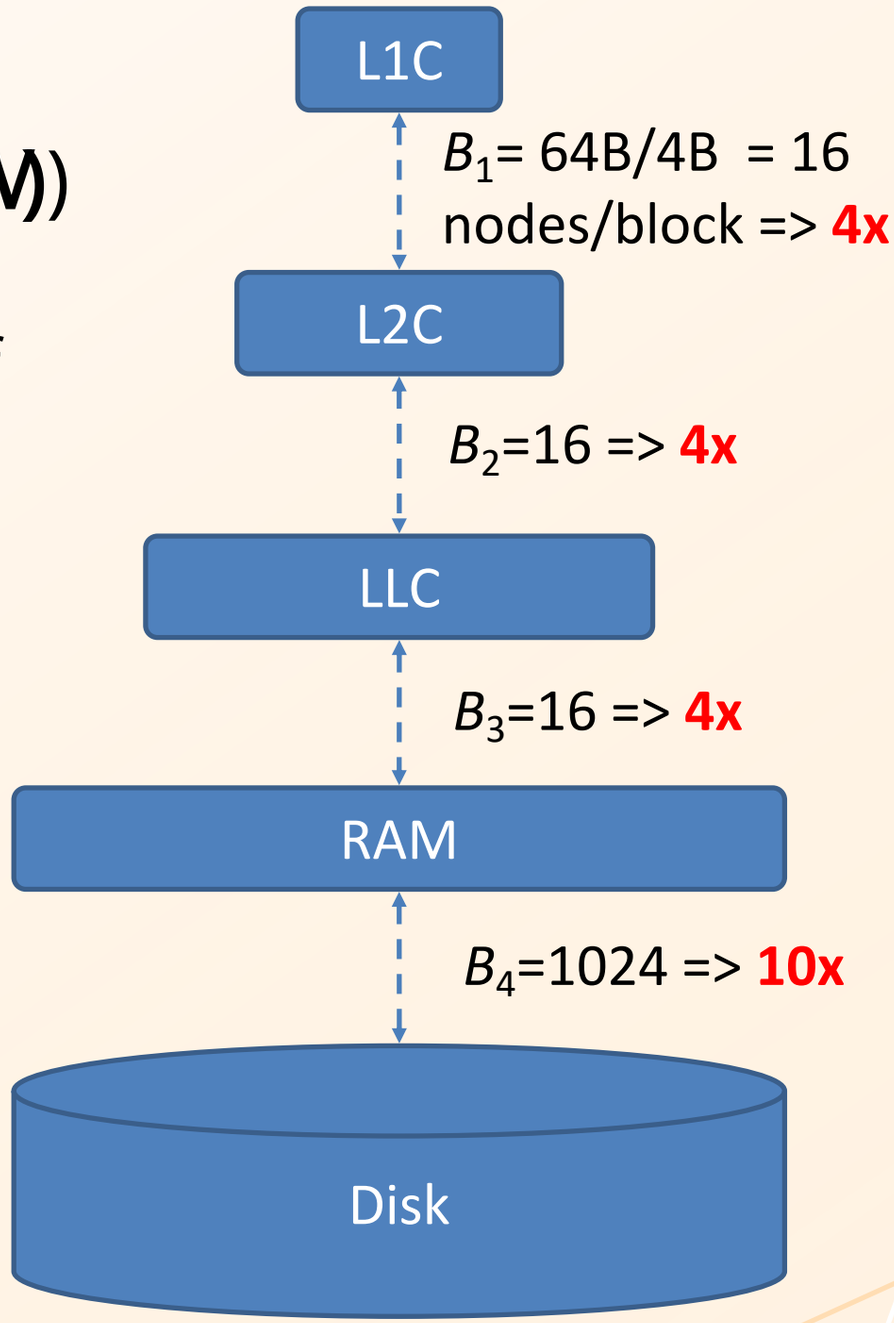
- CO model: van Emde Boas layout [4, 5]
- Search: **$O(\log_B N)$** data transfers (I/Os), where B is **unknown**
  - Cons: Inherently *sequential* during update, no fine-grained locking



[4] Prokop, H.: Cache-oblivious algorithms. Master's thesis, MIT (1999)
[5] van Emde Boas, P.: Preserving order in a forest in less than logarithmic time. In: Proc. 16th Annual Symp. Foundations of Computer Science. pp. 75–84. SFCS '75 (1975)

Fine-grained data locality:
multilevel memory benefits more

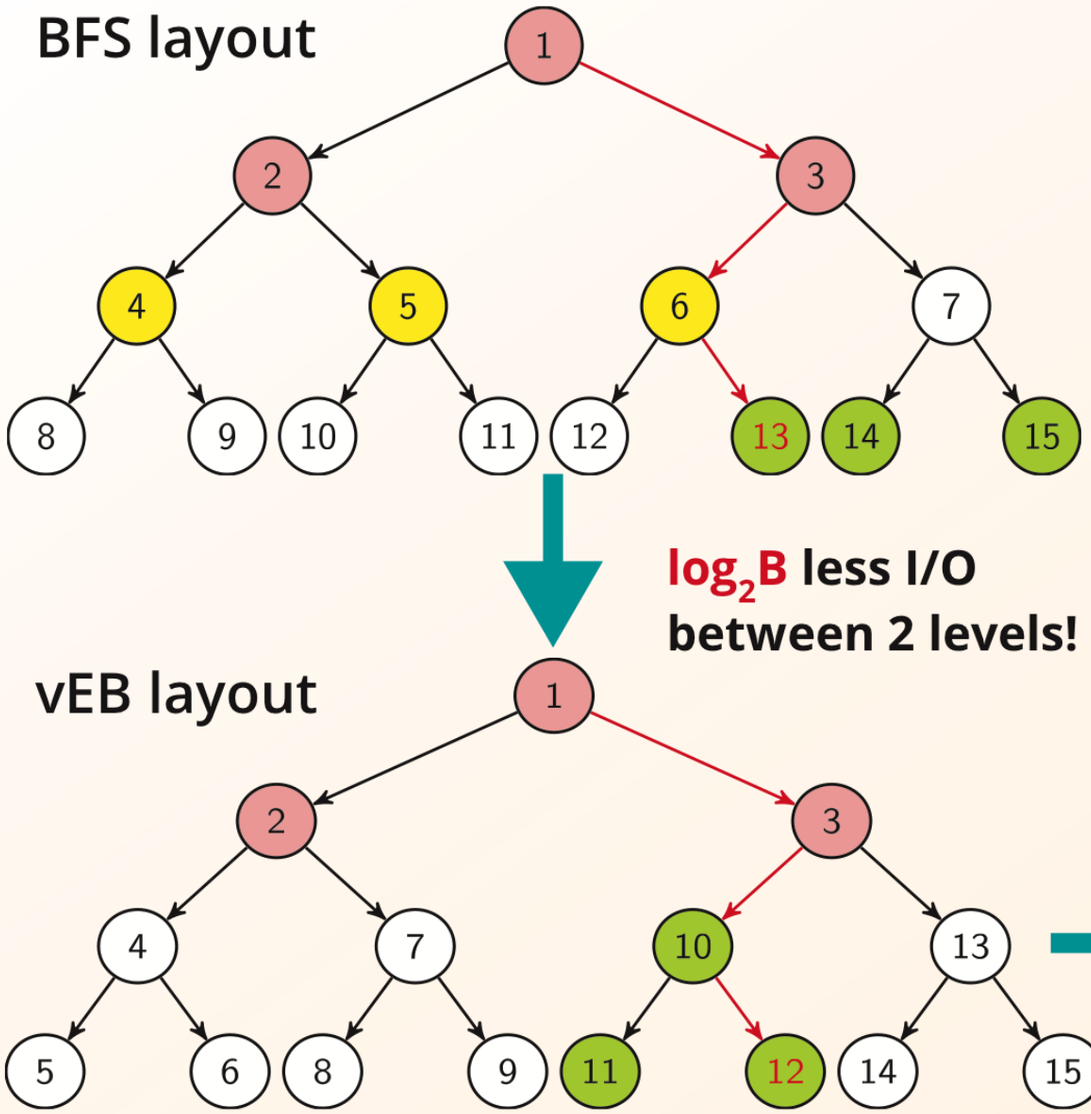
- The BFS layout tree has **O($\log_2 N$)** I/O complexity (*vs.* vEB w/**O($\log_B N$)**)
  - The vEB layout has $\log_2 B$ less I/O than BFS layout between 2 levels of memory
- Commodity machines, e.g.,
  - Tree node size: **4B**
  - Page size: **4KB**
  - Cache line: **64B**
- Maximum of **640x** less I/O for all levels (*intuitively*)



Locality-aware concurrent search tree: DeltaTree [Sigmetrics'15]

- A novel **relaxed cache-oblivious model** based on the cache-oblivious model, but suitable for high-concurrency algorithms
- We transform the van Emde Boas (vEB) layout for search trees into a novel **concurrency-aware vEB layout**
  - The layout benefits **concurrent updates,** unlike the original vEB layout
- We devise DeltaTree, a novel practical locality-aware concurrent search tree
  - DeltaTree search, Insert & Delete: **$O(\log_B M)$** I/O complexity, where B is unknown, but upper bound (UB) is known

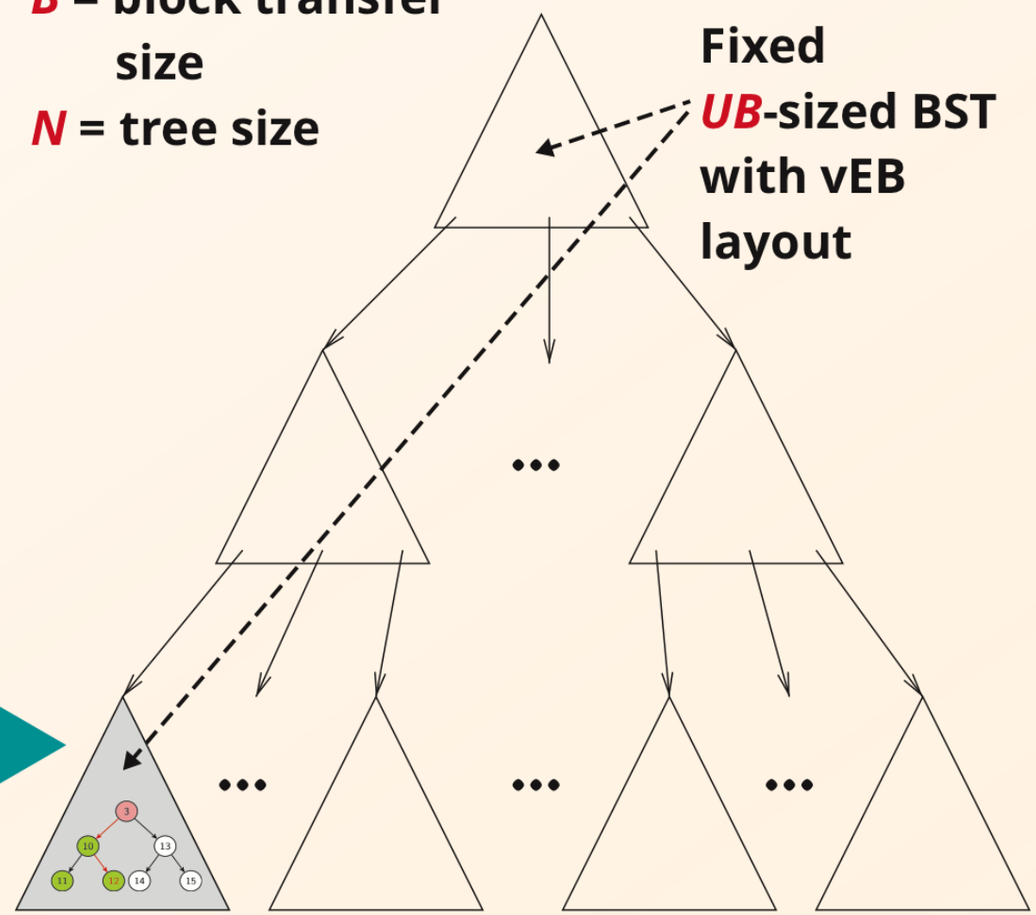
DeltaTree structure



DeltaTree search's data transfer (I/O) complexity is $O(\log_B N)$, where

B **= block transfer size**

N **= tree size**



DeltaTree is energy-efficient [PPoPP'16]

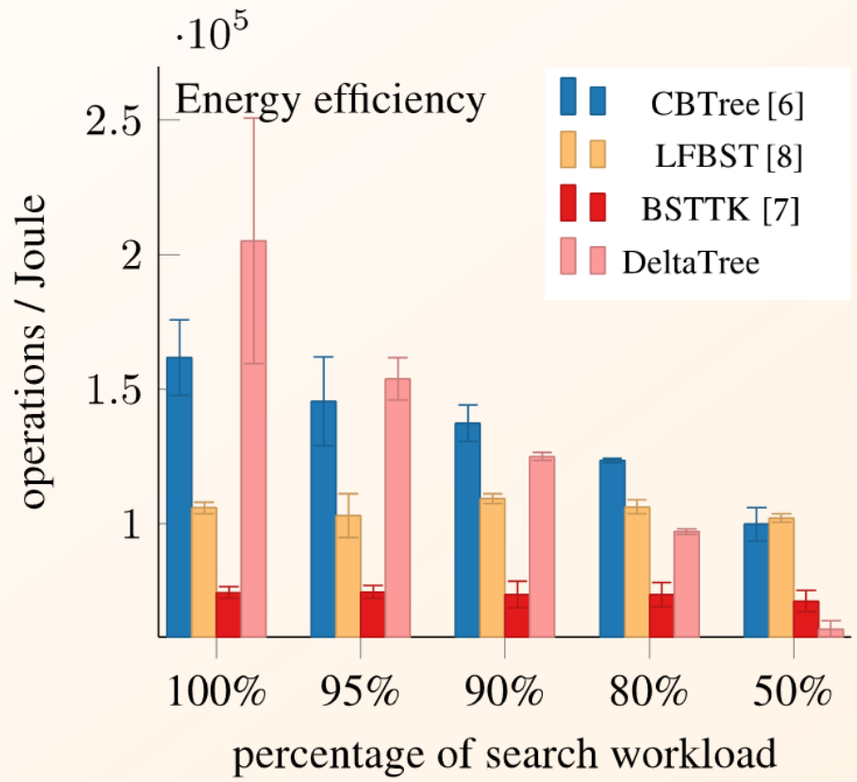
- Through experiments we documented the energy efficiency and throughput of DeltaTree and other state-of-the-art trees:
  1. **CBTree,** prominent locality-aware concurrent B+tree [6]
  2. **BSTTK**, portably scalable concurrent search tree [7]
  3. **LFBST,** non-blocking binary search tree [8]

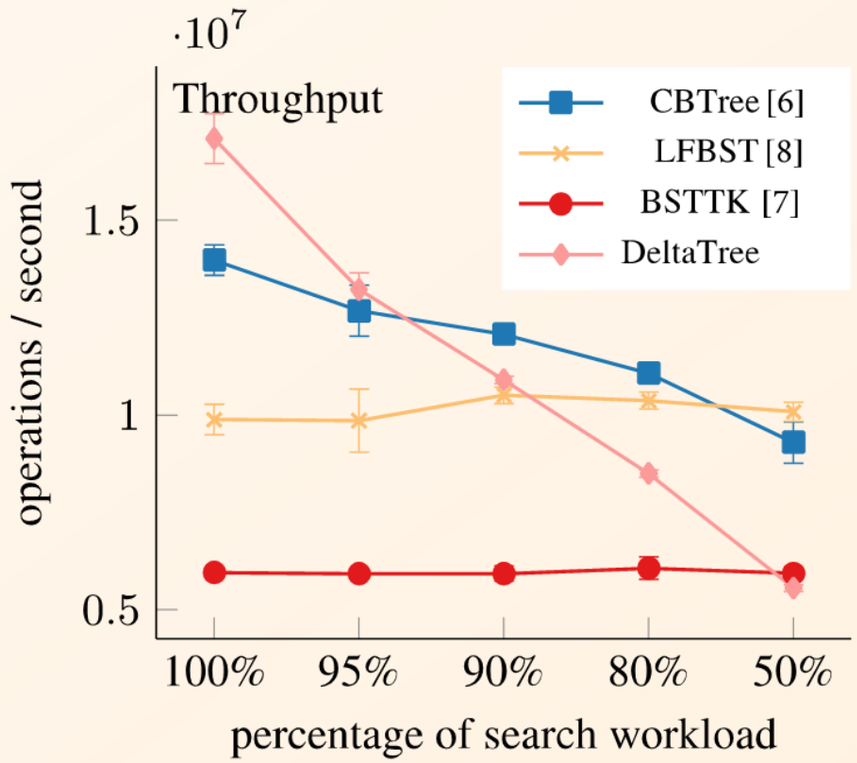
- DeltaTree energy-efficiency is better than state-of-the-art for the *search-intensive* workloads by up to **24%**

[6] Lehman, P.L., Yao, s.B.: Efficient locking for concurrent operations on b-trees. ACM Trans. Database Syst. 6(4), 650–670 (Dec 1981)
[7] David, T., Guerraoui, R., Trigonakis, V.: Asynchronized concurrency: The secret to scaling concurrent search data structures. In: Proc. 12th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems. pp. 631–644. **ASPLOS '15** (2015)
[8] Natarajan, A., Mittal, N.: Fast concurrent lock-free binary search trees. In: Proc. 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 317–328. **PPoPP '14** (2014)

However, …

- DeltaTree's energy efficiency and throughput is low in the *update-intensive* workloads



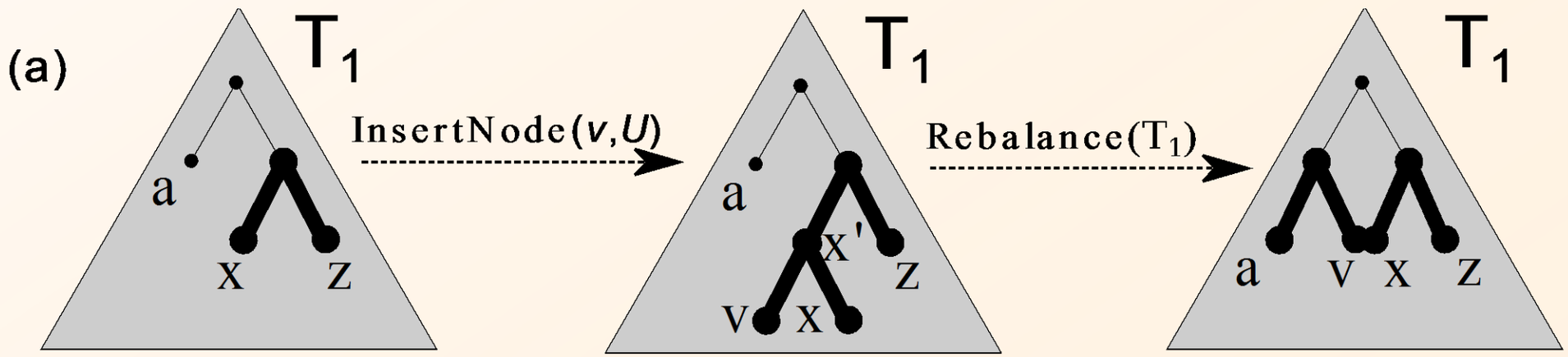


- Overhead of DeltaTree's **maintenance operations**

DeltaTree maintenance operation

- **Rebalance**, a maintenance operation that is required to keep DeltaTree in a good shape
  - Low height
  - Space saving

- However, this is DeltaTree's biggest operational overhead because it *rearranges the whole UB-sized tree* (DeltaNode)



GreenBST

GreenBST

- We devised **GreenBST**, a new fine-grained locality aware concurrent tree

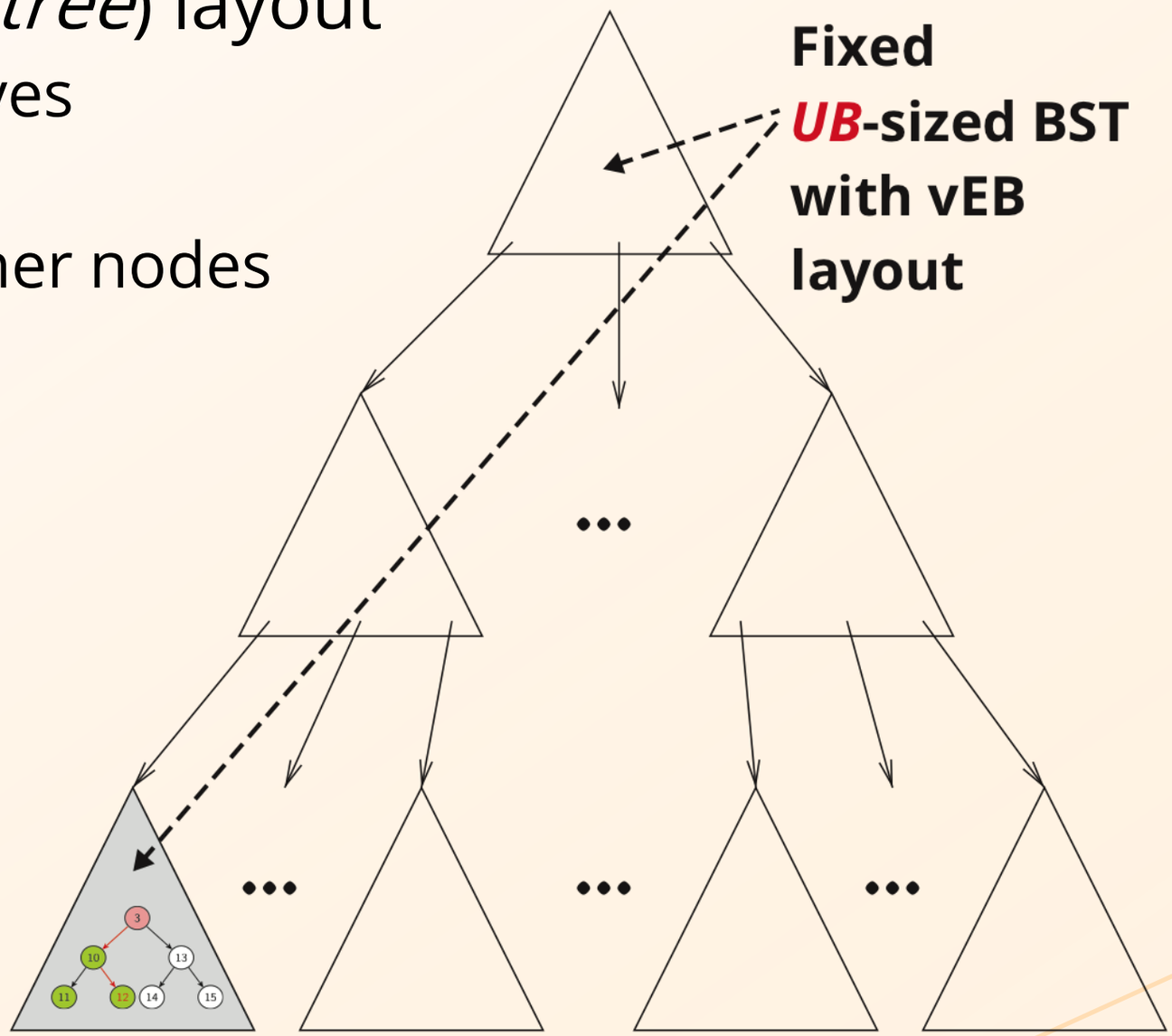
- GreenBST is based on DeltaTree with two significant improvements:

1. We reduce the DeltaTree memory footprint by using *heterogeneous* tree layout

2. We reduce the number of memory transfer in DeltaTree *maintenance* operations

1) Heterogeneous tree layout

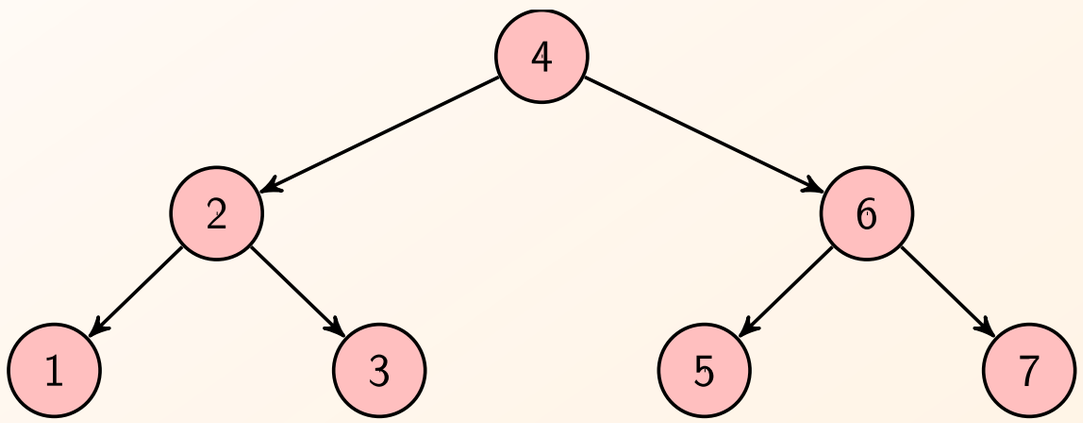
- All DeltaTree's UB-sized nodes are using the *leaf oriented* (or *external tree*) layout
  - All keys are at the leaves
  - Size is 2 x # of keys
  - Required to link to other nodes



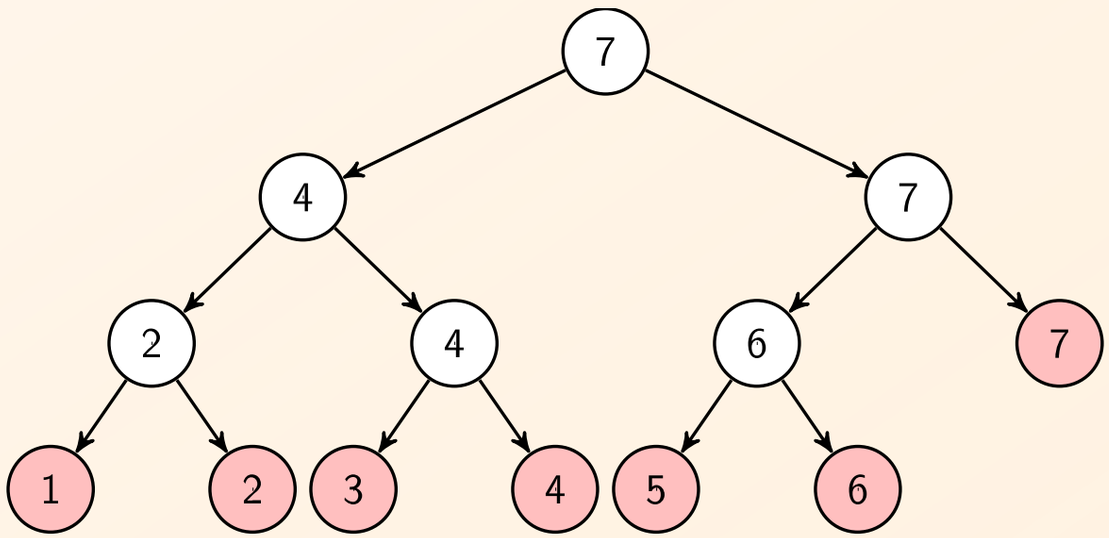
1) Heterogeneous tree layout (cont.)

Tree filled with 1, 2, … , 7 keys

- Non-leaf oriented / internal tree layout

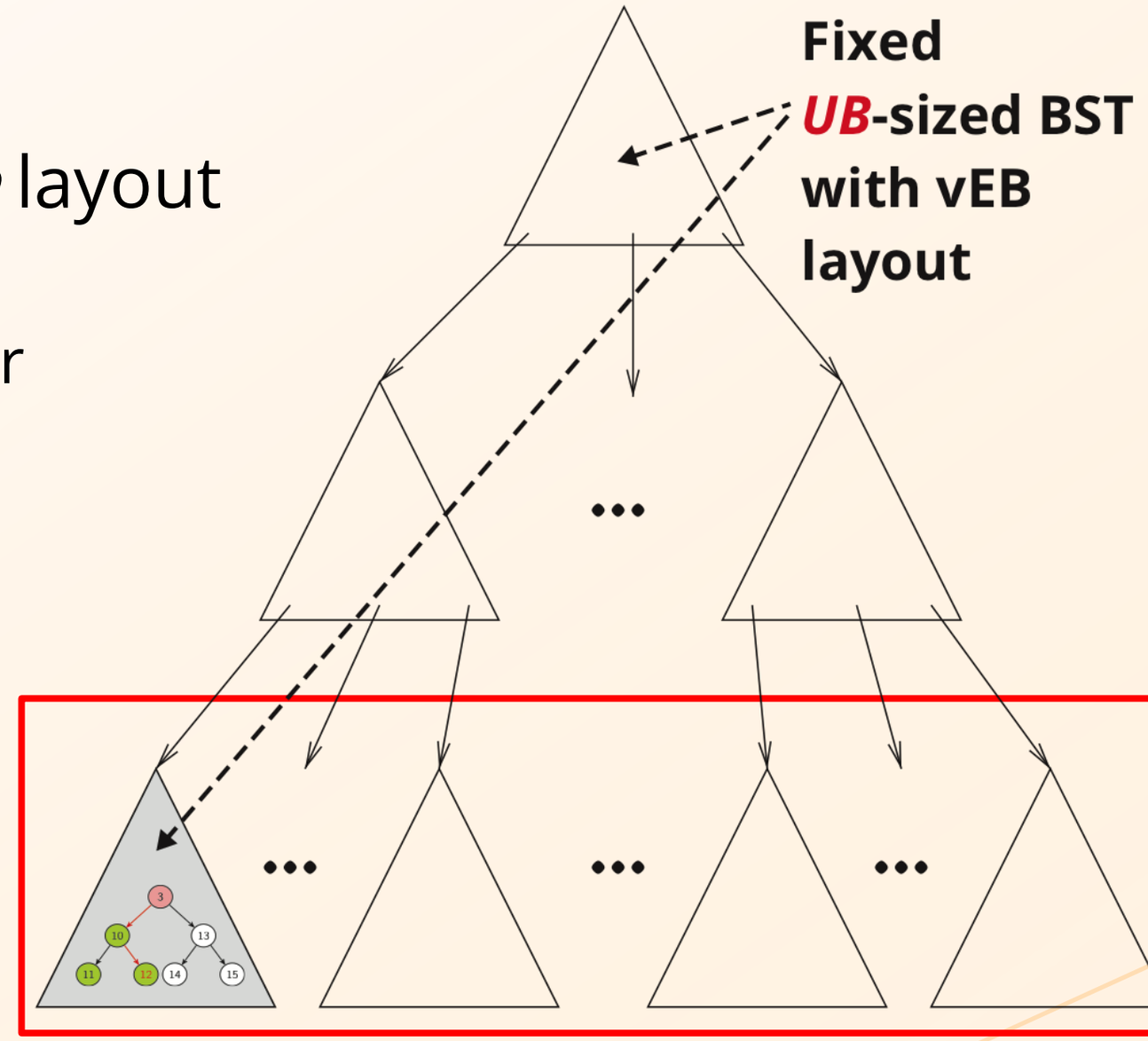


- Leaf oriented / external tree layout



1) Heterogeneous tree layout (cont.)

- However, the leaf *UB* nodes do not need to link to other nodes
- Use the *internal tree* layout
  - Less memory transfer during rebalancing
  - Save **25%** of space
    - Faster search

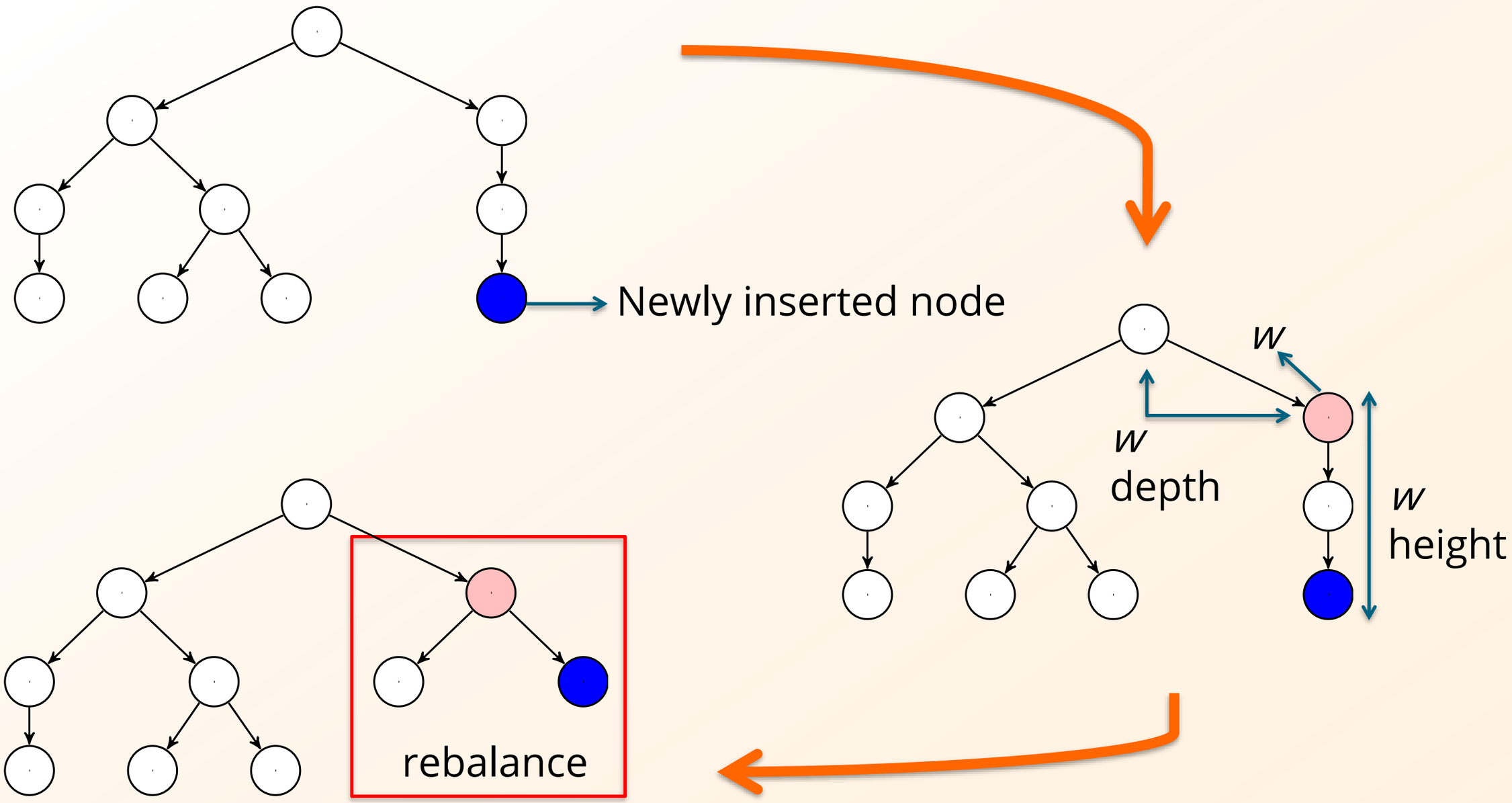


2) Incremental rebalance

- We define: `density(`*w*`)= #of keys inside subtree rooted at` ***w***`/max. keys inside the subtree`
  - Density is calculated after insertion and back-tracks to predecessor nodes
  - For example, a subtree **w** with height 3 and is only filled with 3 keys, then density(*w*) = $3/(2^3 - 1) =$ **0.42**
- There is also a density threshold $0 < \Gamma_1 < \Gamma_2 < \ldots < \Gamma_H$, where *H* is the tree height
- We only rebalance a subtree **w**, where density(*w*) ≤ $\Gamma_{\text{depth}(w)}$, following [9]

[9] Brodal, G.S., Fagerberg, R., Jacob, R.: Cache oblivious search trees via binary trees of small height. In: Proc. 13th ACM-SIAM Symp. Discrete algorithms. pp. 39–48. SODA '02 (2002)

2) Incremental rebalance (cont.)



EVALUATION

Evaluation setup

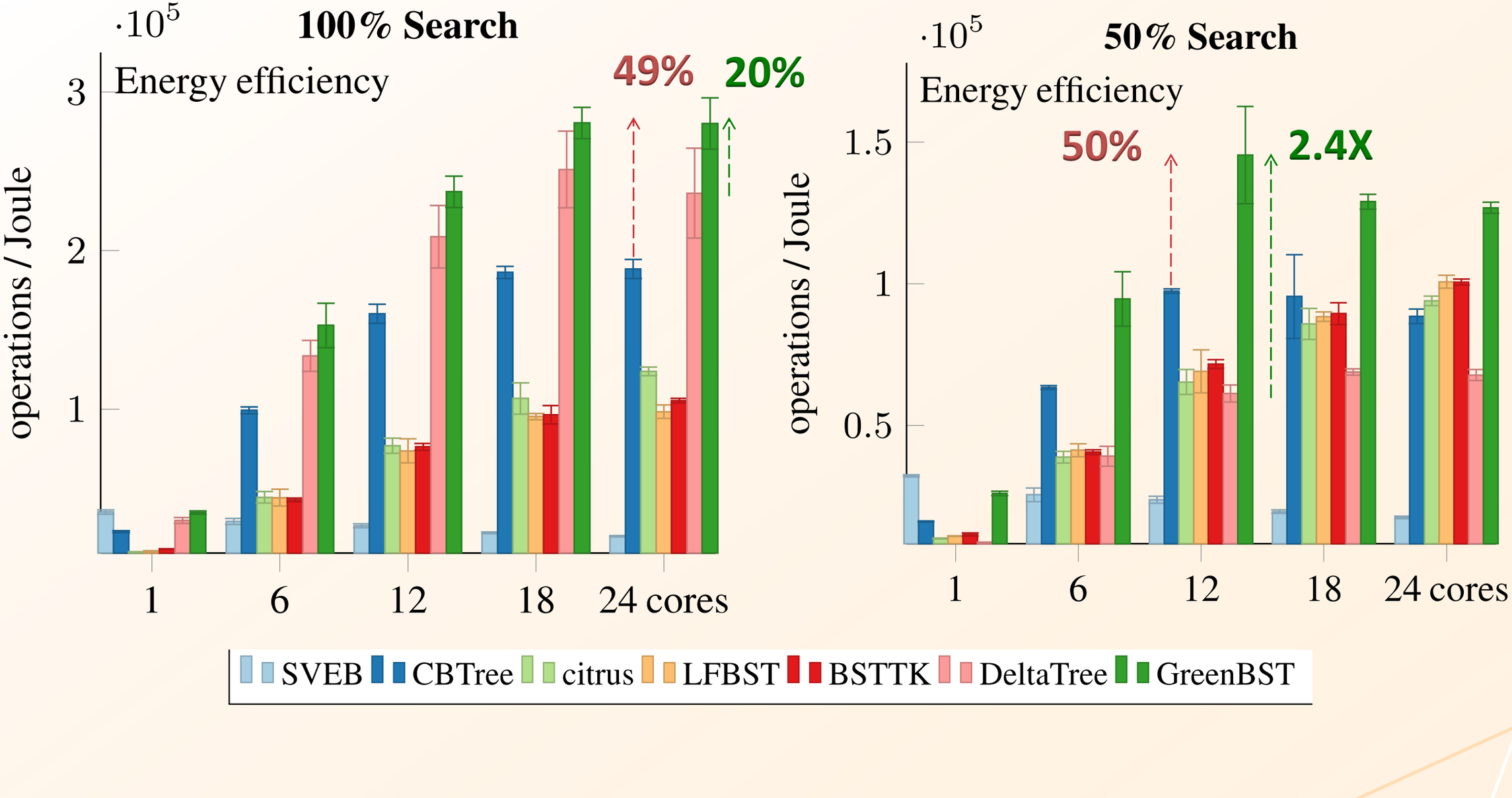
- We measured the energy efficiency and throughput of operations of several state-of-the art trees on multiple architectures

Algorithm	Description	Published
SVEB	Conventional vEB layout search tree	SODA'02
CBTree	Concurrent B-tree (B-link tree)	TODS'81
Citrus	RCU-based search tree	PODC'14
LFBST	Non-blocking binary search tree	PPoPP'14
BSTTK	Portably scalable concurrent search tree	ASPLOS'15
DeltaTree	Locality aware concurrent search tree	-
GreenBST	Improved locality aware concurrent search tree	-

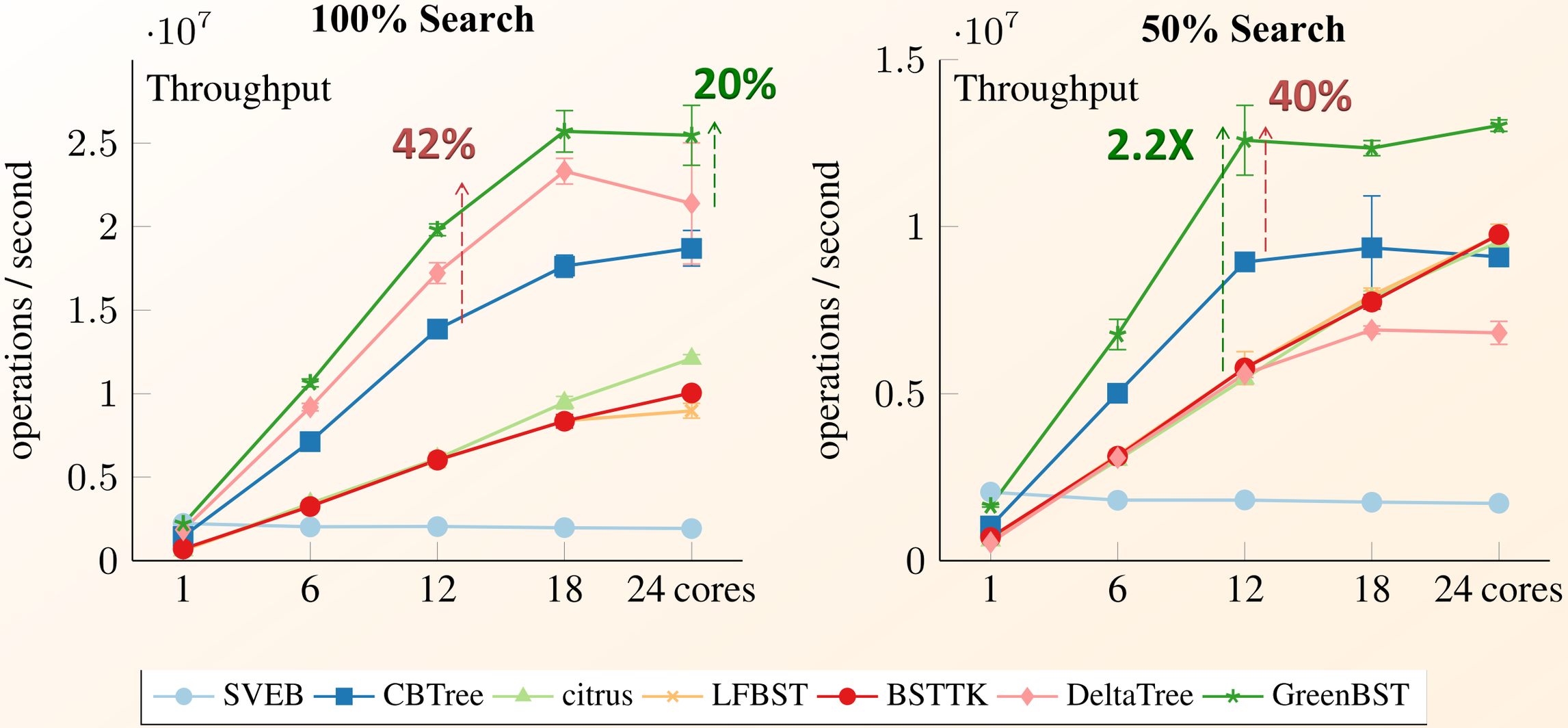
Evaluation setup (cont.)

- Platforms used:
  - **HPC platform** (24 core 2× Intel Xeon E5-2650Lv3 CPU with 64GB of RAM)
  - **ARM platform** (8 core Odroid XU+E, Samsung Exynos 5410 CPU with 2GB of RAM)
  - **MIC platform** (with 57 core Intel Xeon Phi 31S1P with 6GB of RAM)
- We run 5 million operations with **100%** and **50%** search after initial loading

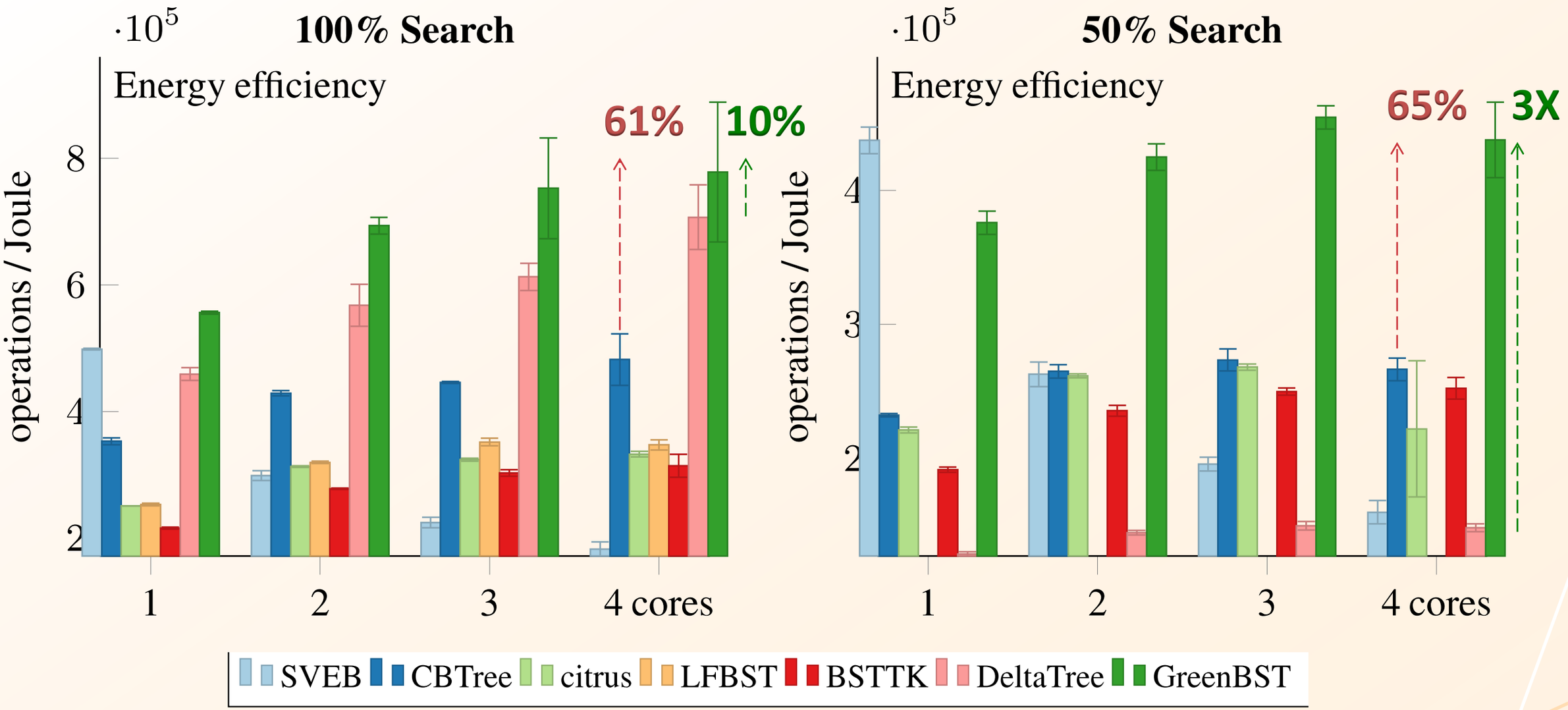
Energy efficiency (HPC platform)



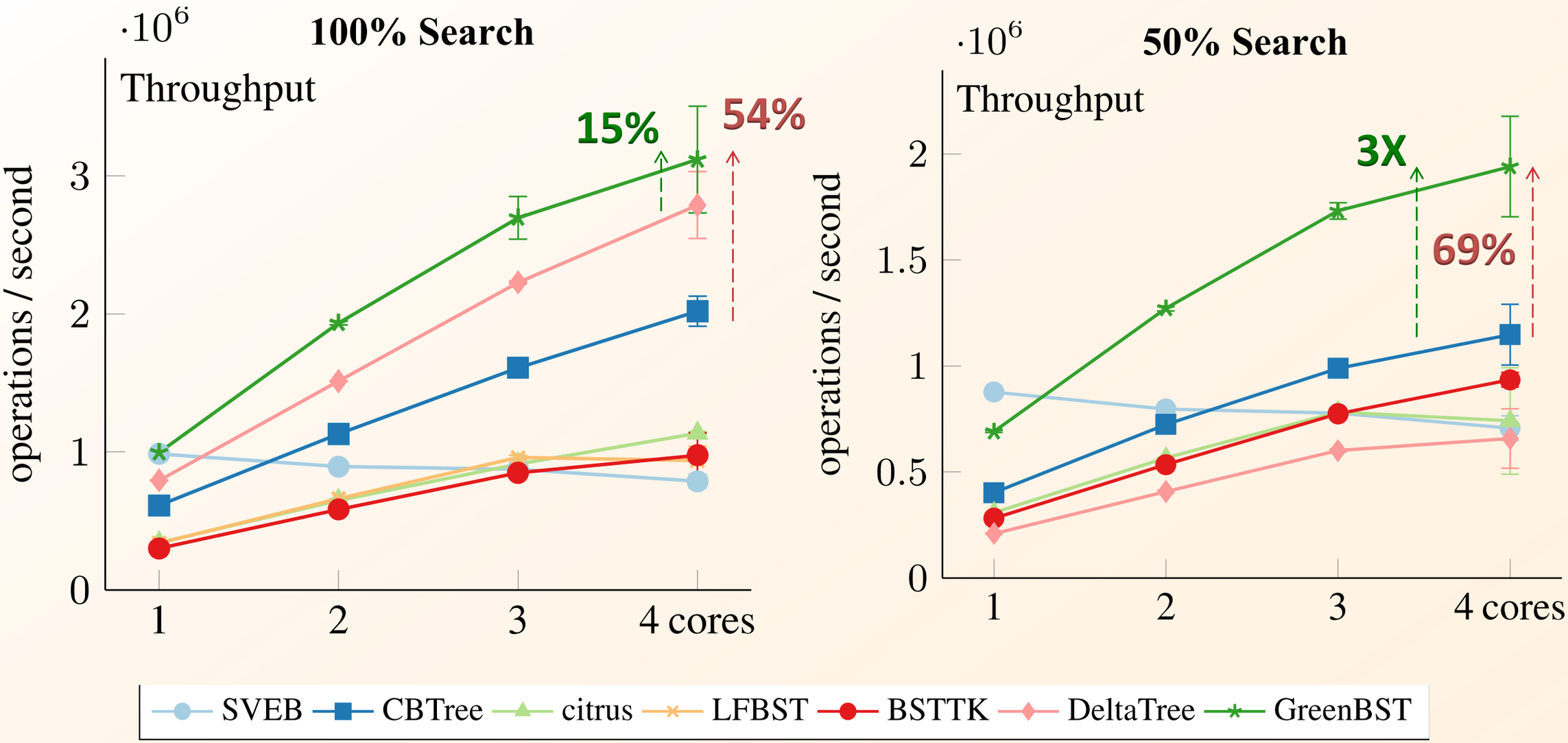
Throughput (HPC platform)



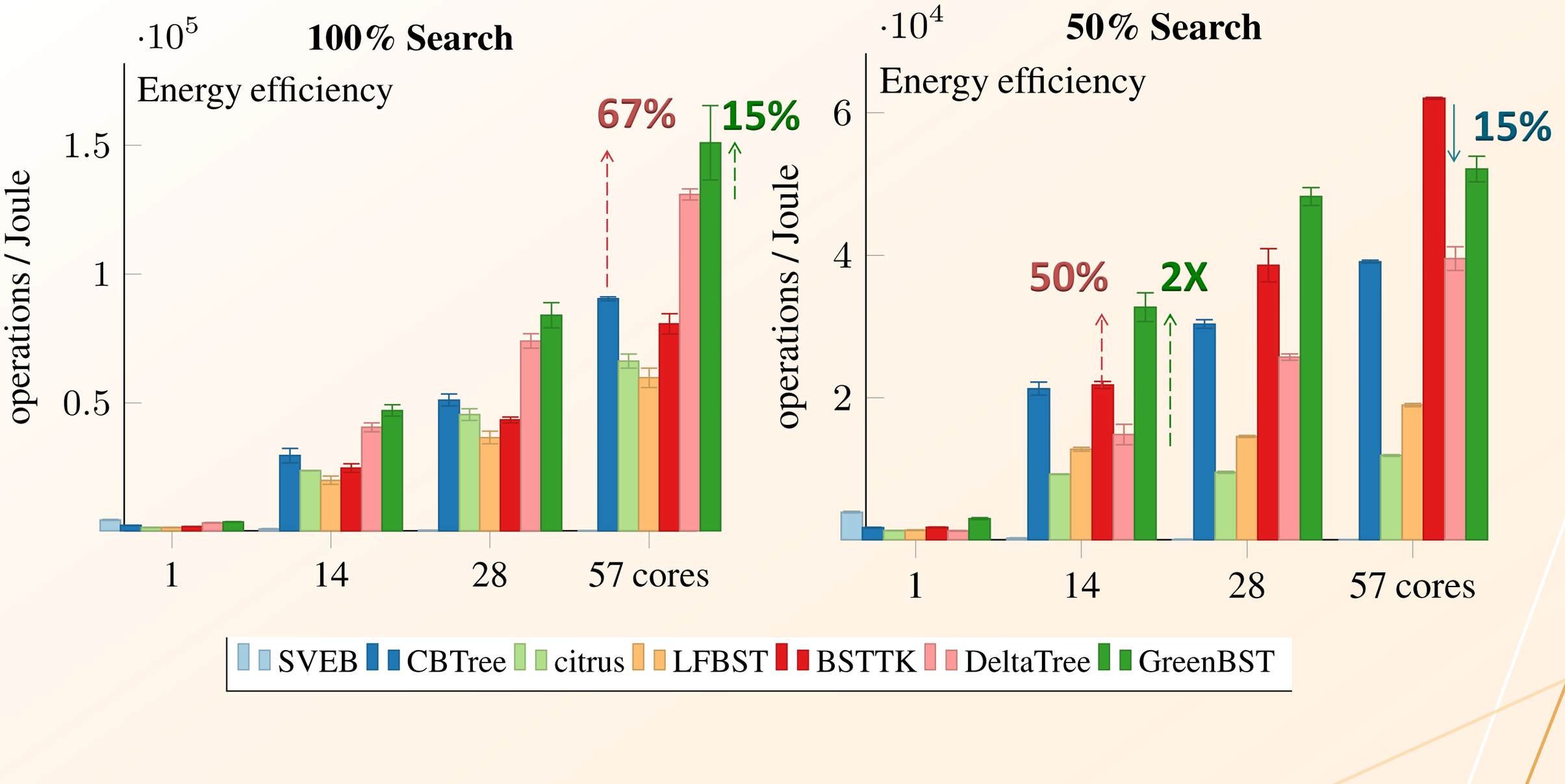
Energy efficiency (ARM platform)



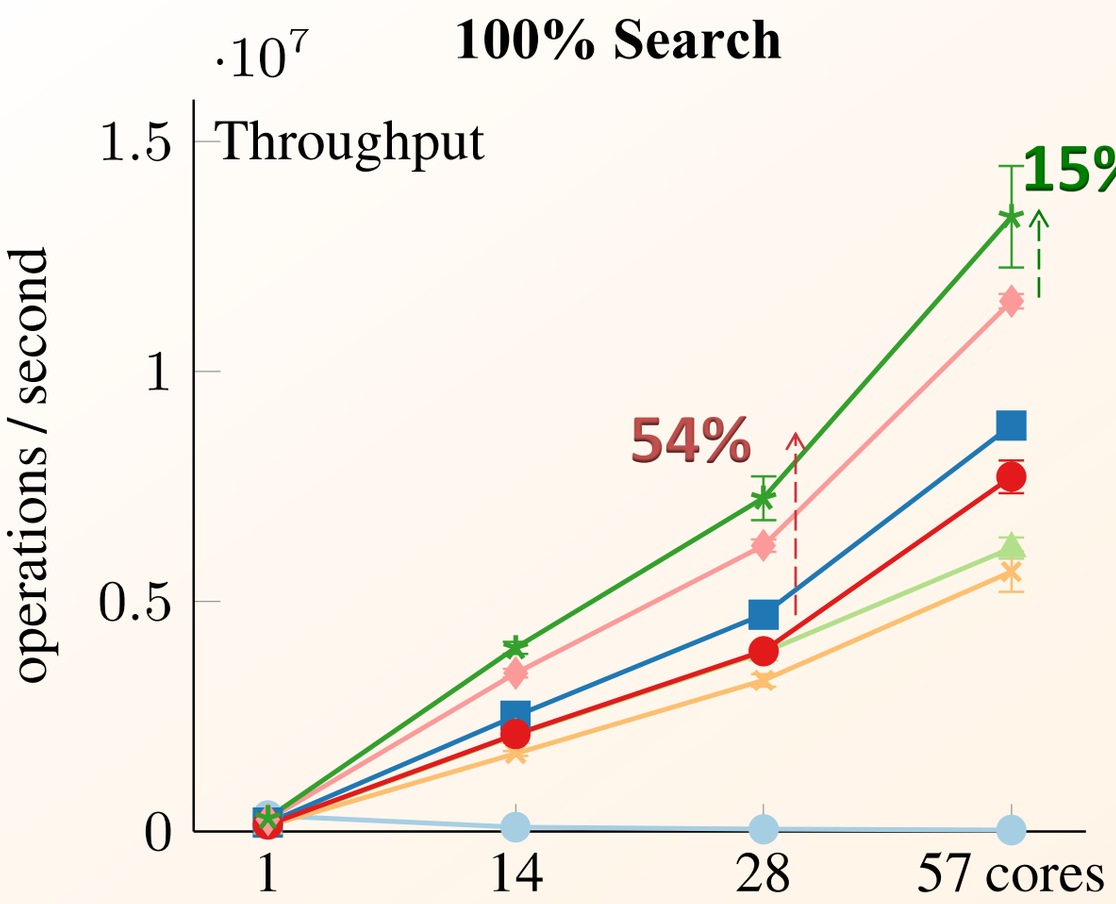
Throughput (ARM platform)

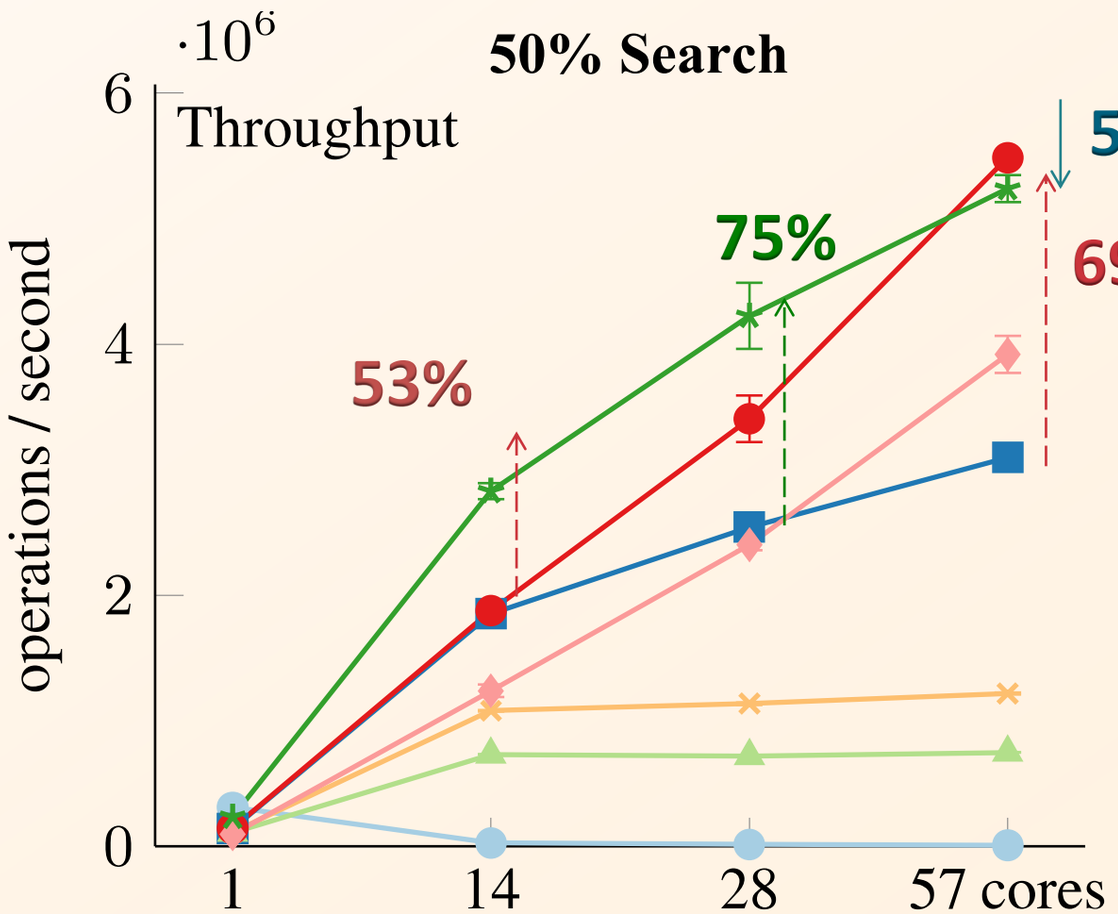


Energy efficiency (MIC platform)



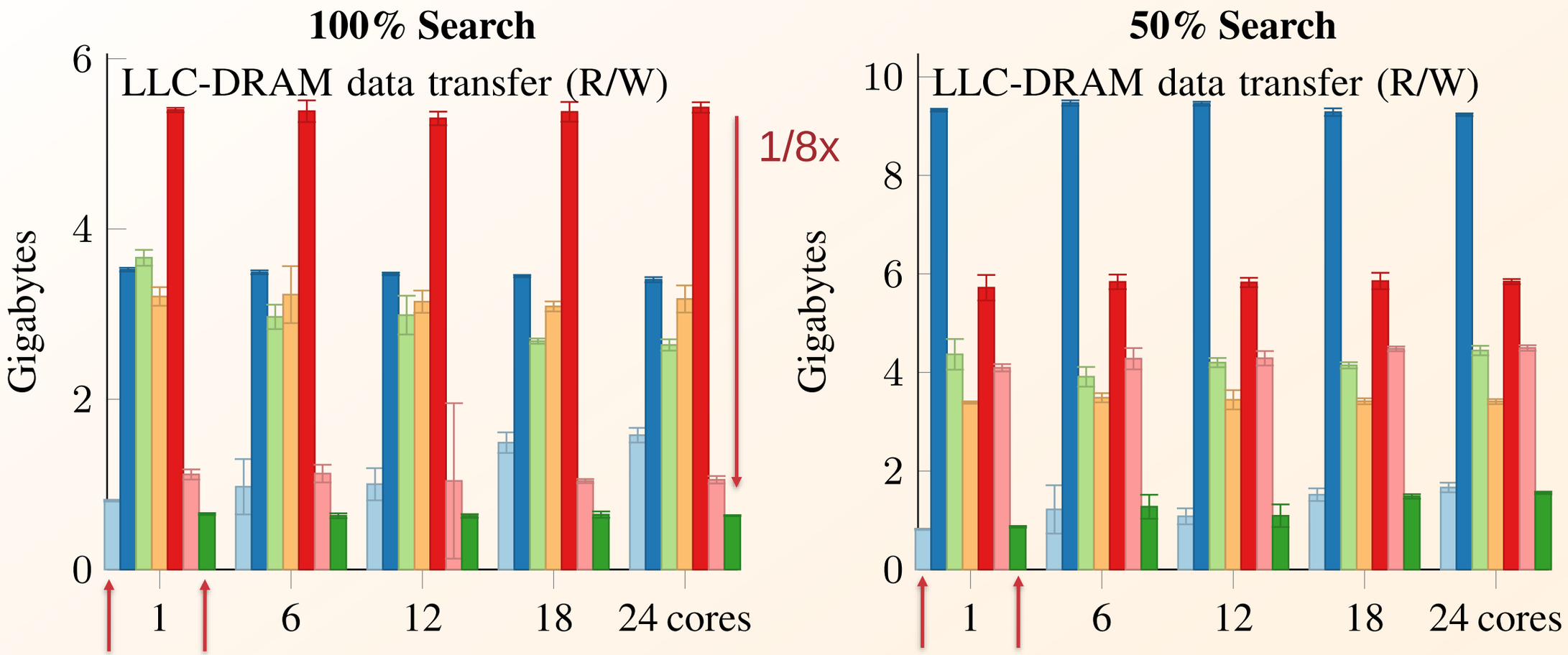
Throughput (MIC platform)





SVEB CBTree citrus LFBST BSTTK DeltaTree GreenBST

LLC-DRAM data transfer on the HPC platform



GreenBST transfers less data from/to DRAM than the other trees

SVEB CBTree citrus LFBST BSTTK DeltaTree GreenBST

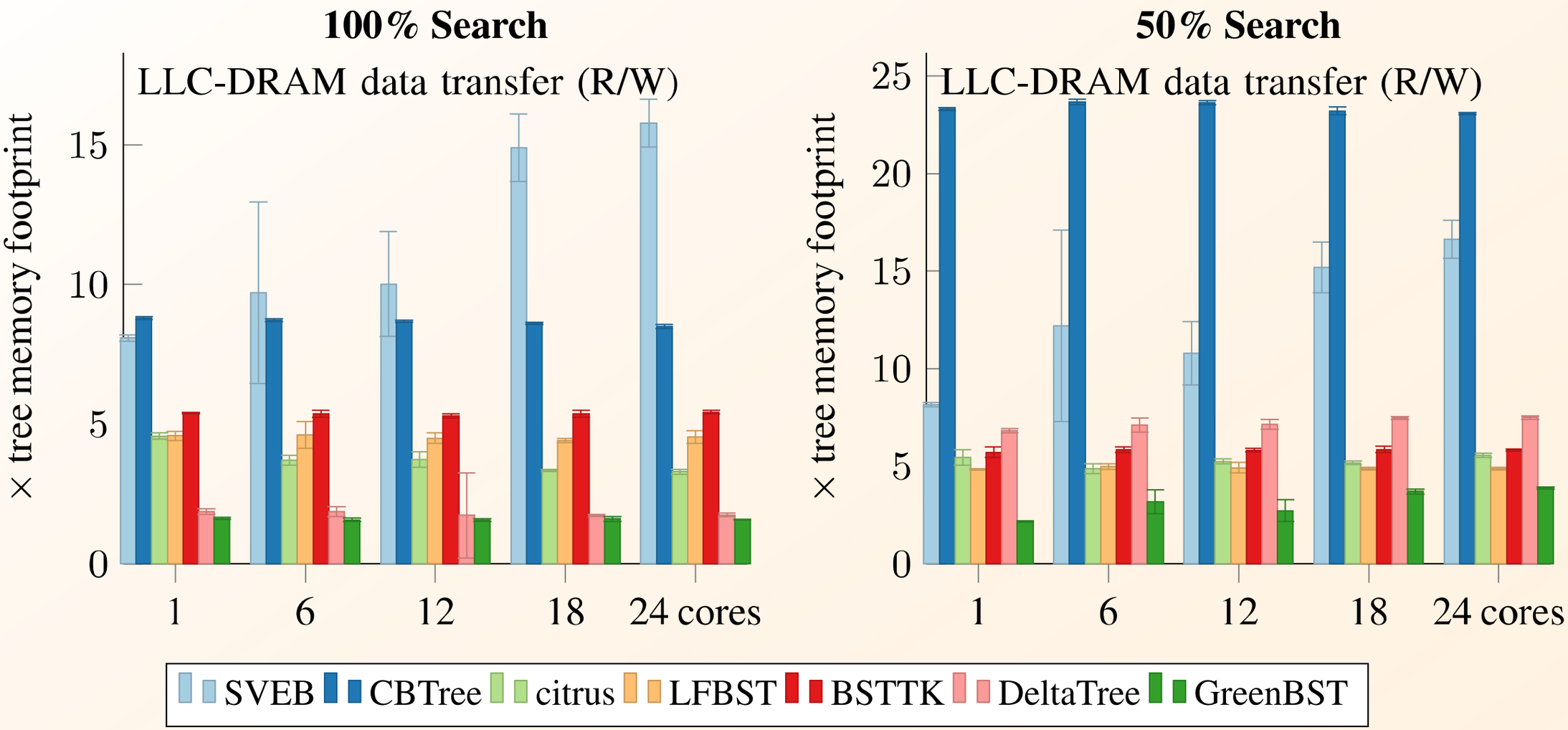
HPC Platform: The tree memory footprint after the initial loading into memory

Tree name	SVEB	CBTree	citrus	LFBST	BSTTK	DeltaTree	GreenBST
Memory used (in **GB**)	0.1	0.4	0.8	0.7	1.0	0.6	0.4

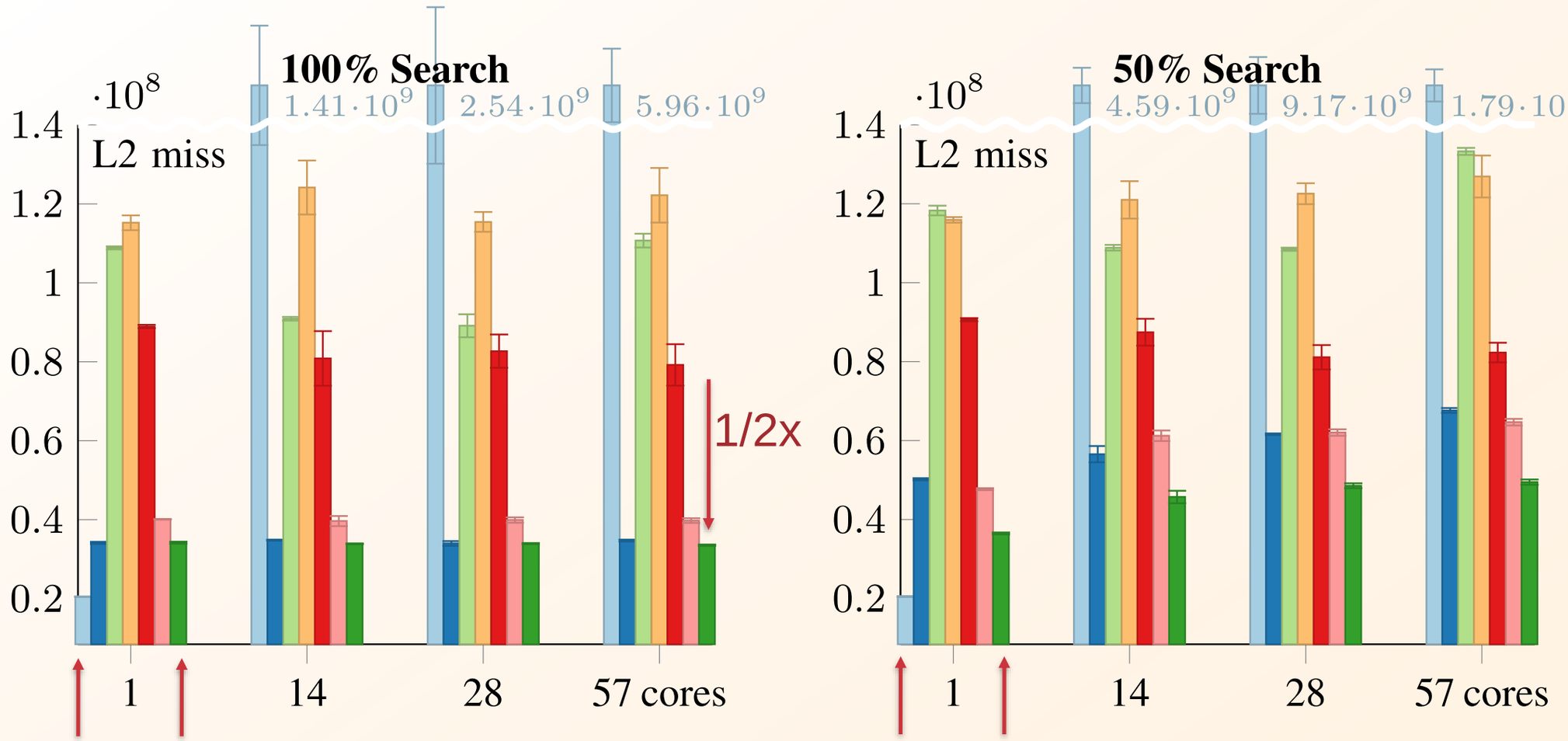
- GreenBST size 0.4x of BSTTK
- However, I/O can be 0.12x (i.e., GreenBST vs BSTTK in 100% search using 57 cores)

- GreenBST re-uses more data than the other trees

LLC-DRAM data transfer on the HPC platform (normalized, relative to the tree memory footprint)

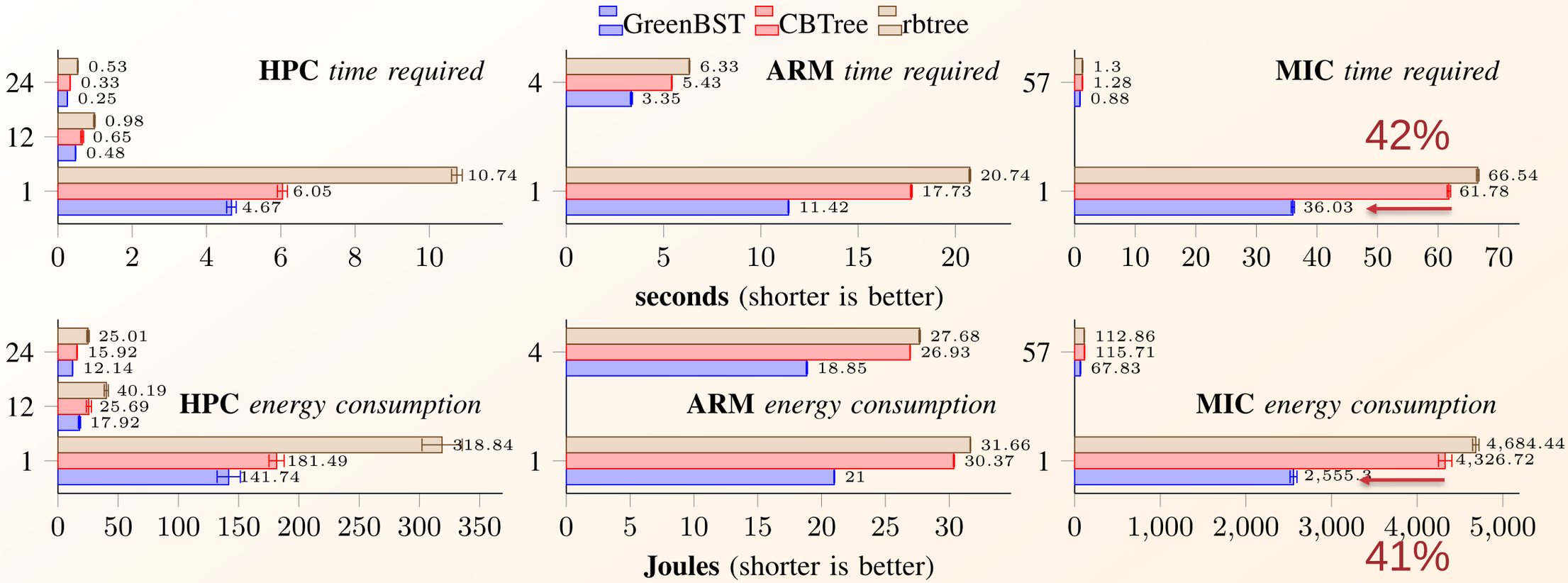


L2 cache miss on the MIC platform



GreenBST has fewer L2 misses than the other trees, except SVEB when using single core

Vacation benchmark from Stanford STAMP [12]



GreenBST needs 42% less time to finish the benchmark and 41% less energy to finish the benchmark

[12] Minh, C.C., Chung, J., Kozyrakis, C., Olukotun, K.: Stamp: Stanford transactional applications for multi-processing. In: Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on. pp. 35–46 (Sept 2008)

CONCLUSION

Conclusions

- GreenBST is the *first* portable energy-efficient concurrent search tree (see paper for the source code link)

- There are tradeoffs for using cache-obliviousness in data structures:

1. On multi-CPU and many cores systems, data-structures' locality-awareness can easily saturates the CPU interconnect bandwidth (e.g., Xeon's QPI and MIC's ring interconnect)

1. Higher interconnect bandwidth or novel data access pattern strategies for the cache-oblivious data structures for multi-CPU and many cores systems are needed

1. Otherwise, multi-CPU coherency mechanism energy overhead can exceed the energy saving obtained by fewer data movements.

THANK YOU