Pruning Masstree

Huanchen Zhang (CMU), David G. Andersen (CMU), Michael Kaminsky (Intel Labs)

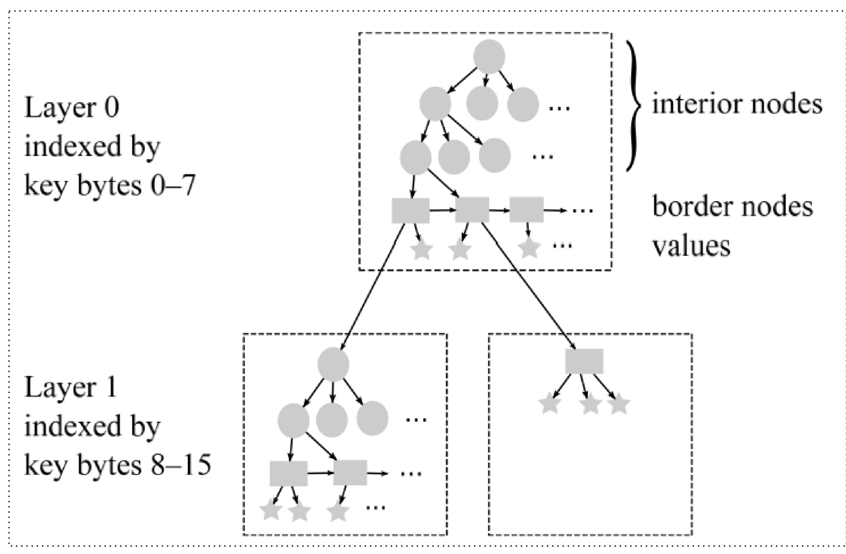
Overview

Motivation: Key-value stores are a critical building block behind many cloud and network services

Goal: Building a space-efficient, high-performance key-value store that also supports range queries

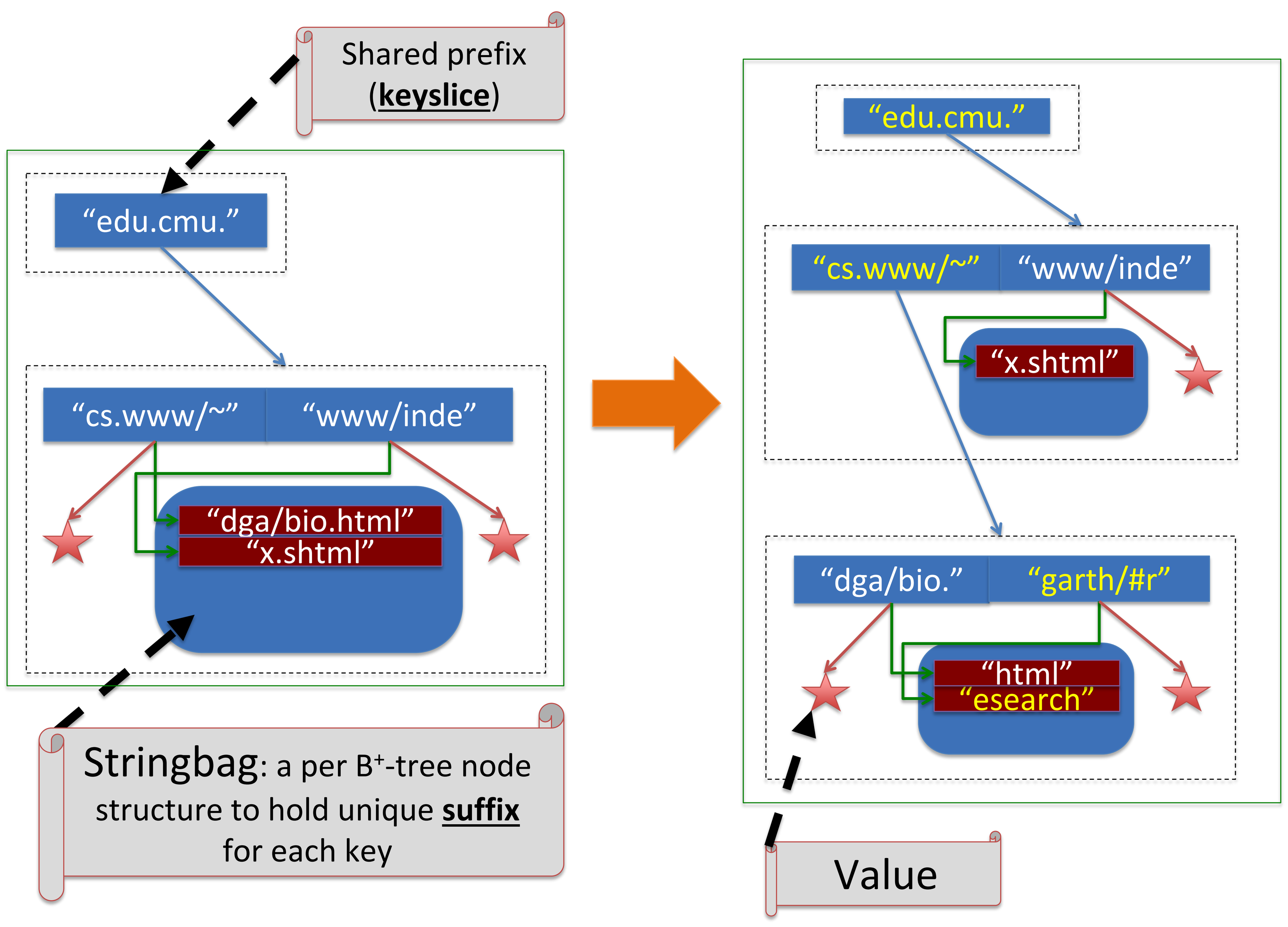
Baseline: Masstree

- The basic structure of Masstree is a concatenation of layers of B⁺-trees that conceptually form a trie [Masstree, Eurosys'12]



Example

- Initially, two URL keys "edu.cmu.cs.www/~dga/bio.html" and "edu.cmu.www/index.shtml" are stored in the Masstree
- Inserting a third key "edu.cmu.cs.www/~garth/#research" to the original 2-layer Masstree leads to a 3-layer Masstree



Improvement 1: Space-Efficient Masstree

Problem: high memory waste from Stringbags

- Aggressive coarse-grain memory allocation
- Internal fragmentation

Solution

- **More effective garbage collection**
  - Detect and reclaim unused Stringbags
  - Resolve internal fragmentation
- **More efficient memory allocation**
  - Conservative (invoke gc before granting new space)
  - Fine-grain to avoid over-allocation

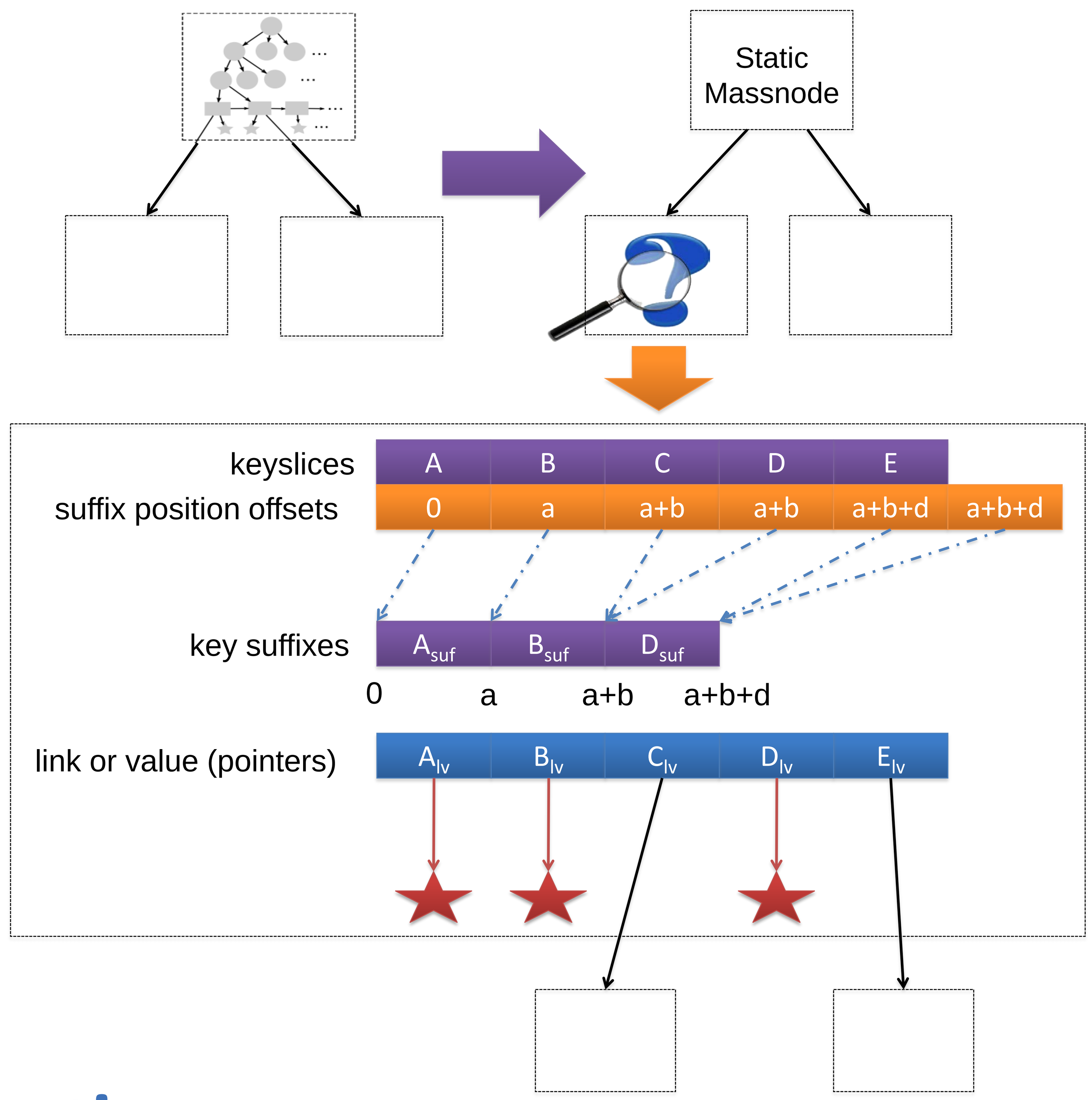
Improvement 2: Static Masstree

Problem: high structural overhead

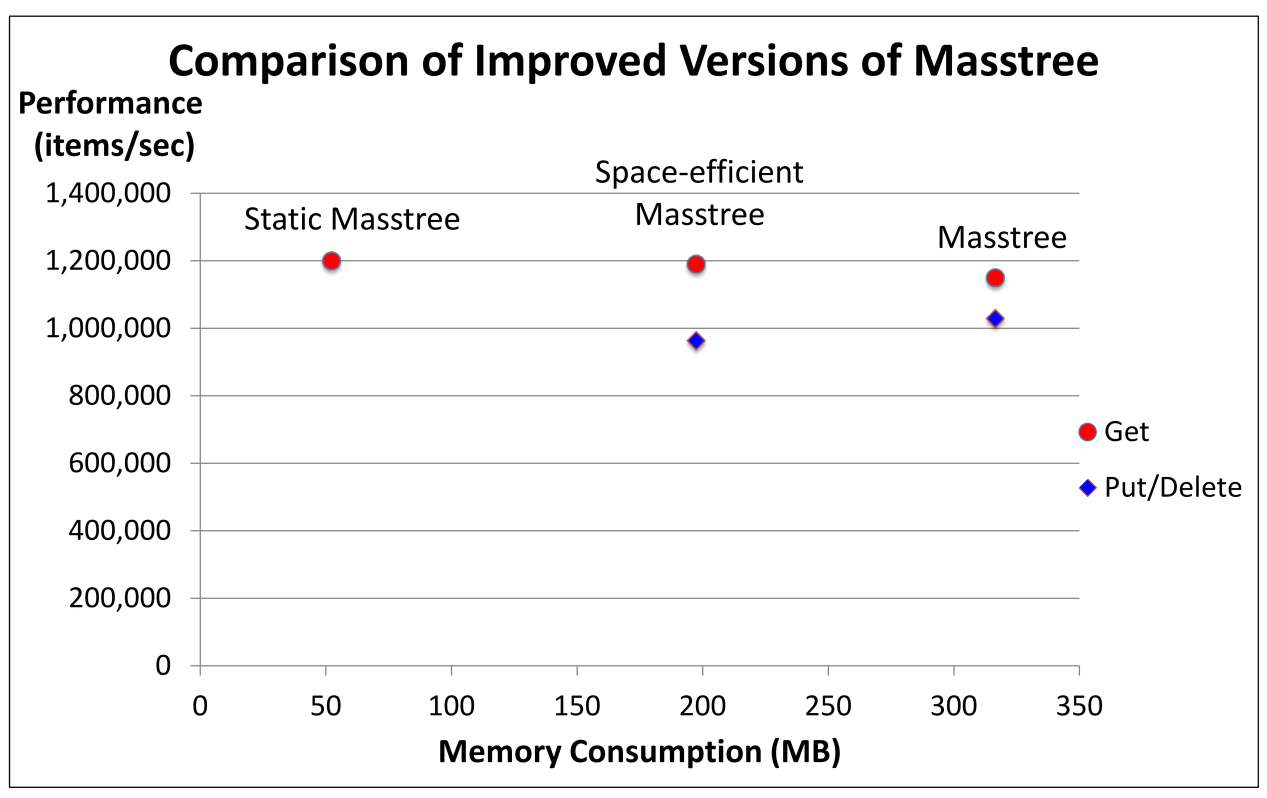
- Most B⁺-trees contain very few keys

Solution: treat "cold" keys as read only

- Preserve the trie structure for space-efficiency
- Serialize each B⁺-tree into a sorted array of keyslices and perform binary search on it for indexing
- Eliminate Stringbags and store key suffixes in place



Evaluation



Workload (based on YCSB)
- Key: URL
- Value: 64-bit integer
- 67% put, 33% delete; then 100% get
- Single thread

Workload (based on TPC-C)
- Key: 15-40B string
- Value: 64-bit integer
- Single thread

Intel Science & Technology Center for Cloud Computing

Carnegie Mellon University · Georgia Tech · intel · Princeton University · UC Berkeley · University of Washington