

Learned Index Structures: Foundations, Updatability, & Hardware Realities

- Kraska et al., *The Case for Learned Index Structures* (SIGMOD 2018)
- Ding et al., *ALEX: An Updatable Adaptive Learned Index* (SIGMOD 2020)
- Proposed design: Chunked Contiguous Learned Extents (CCLE)
- 60 minutes presentation, 20 minutes Q&A

The Bottleneck Flipped

- B-trees were designed for spinning disks, where high fanout minimized expensive block I/O at the cost of CPU comparison branches
- In-register arithmetic costs **1 to 2 CPU cycles**; execution is practically free
- A cache miss to DRAM stalls execution for **50 to 100 CPU cycles**
- Modern in-memory indexing throughput is bounded by memory stalls and cache misses, not arithmetic throughput

The Hardware Tax of Pointer B-Trees

- Main memory transfers data in **64-byte** lines; contiguous memory accesses exploit hardware prefetching automatically
- B-tree traversal is a pointer dependency chain: the CPU cannot fetch child node bytes until the parent comparison finishes
- Internal nodes load full cache lines of keys and metadata solely to route to the next child address
- Tree height directly determines the sequential latency tax of these pointer-chasing round-trips

The Math Budget vs. Tree Branching

- Traversing a 100-key B-tree node takes **~50 cycles** and narrows candidates by **100×** (e.g., 100M $\rightarrow$ 1M)
- In those same 50 cycles, a modern CPU SIMD pipeline can execute **~400 arithmetic operations**
- Evaluating a linear model ($\lfloor a \cdot x + b \rfloor$) takes only **~3 operations**
- If a model spends fewer than 400 operations to shrink the search window by more than 100×, math beats branching
- When cache misses hit B-tree pointer chases (50–100 cycles each), computation wins by an even wider margin

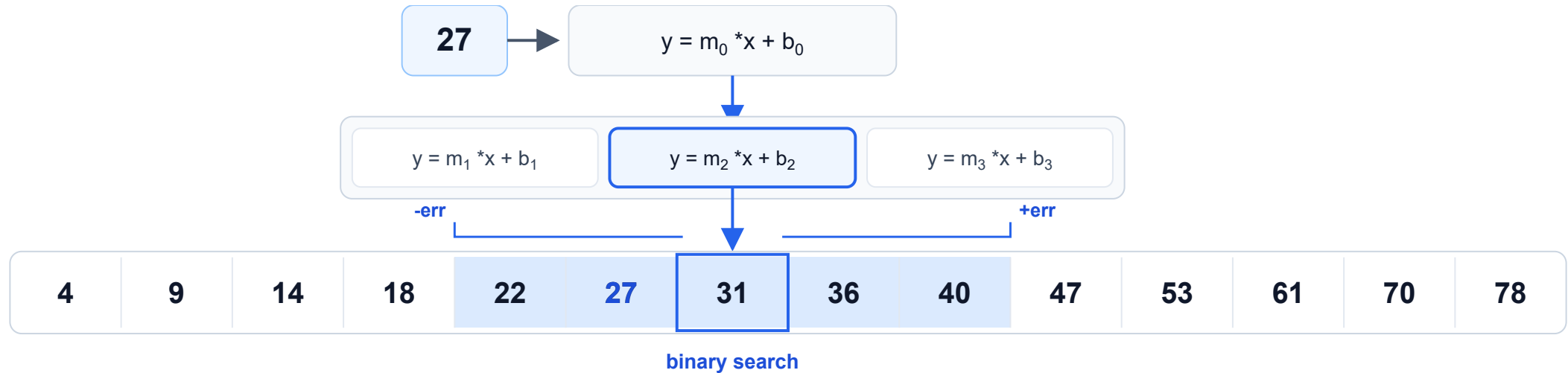
An Index Is a Function

- A range index maps a key to its rank/offset within a sorted contiguous array
- Offset approximates the scaled Cumulative Distribution Function (CDF): $y = F(\text{key}) \times N$
- Model evaluation ($y = \lfloor a \cdot x + b \rfloor$) replaces pointer chasing with direct arithmetic offset calculation
- If a constant-sized model captures the continuous distribution, prediction error scales as $O(\sqrt{N})$, compared to $O(N)$ for a fixed-size B-tree
- Local refinement around the predicted offset accesses contiguous memory, keeping prefetch pipelines saturated

The Naive Single-Model Index

- Baseline experiment: single 2-layer neural network (32 ReLU units/layer) trained on 200M weblog timestamps in TensorFlow
- Evaluation overhead: $\sim 80,000$ ns per inference (excluding search) vs. ~ 300 ns for B-tree traversal and ~ 900 ns for binary search
- In-engine C++ code generation reduces arithmetic execution to ~ 30 ns by eliminating framework dispatch latency
- A single monolithic model cannot learn irregular local CDF features without extreme parameter explosion; hierarchical models are needed

The Recursive Model Index (RMI)



- Hierarchical staging: root model evaluates key and selects a child model in the next stage
- Leaf models output a predicted slot and bound it within stored $[\text{min_err}, \text{max_err}]$ windows
- Final key resolution is bounded to a local search within the error interval

Training the Static RMI Hierarchy

- **Stage-Wise Top-Down Training:** The root model $f_0(x)$ is fit across all keys to predict scaled rank $[0, N)$
- **Key Partitioning:** Keys are fed into stage $l - 1$; the output routes key x into child partition $p = \lfloor M_l \cdot f_{l-1}(x) / N \rfloor$
- **Independent Child Fits:** Each model in stage l is trained strictly on its partitioned subset, minimizing squared error locally
- **Hybrid Fallback:** If a leaf model's empirical `max_err` exceeds an acceptable threshold, the model is swapped for a B-tree, strictly bounding worst-case latency

Stage Traversal Without Pointer Indirection

- In Kraska's static RMI, child selection is purely arithmetic: $\text{slot} = \lfloor M_l \cdot f_{l-1}(\text{key}) / N \rfloor$
- Direct array lookup: next model parameters live at $\text{base} + \text{slot} \times \text{stride}$
- Eliminates internal pointer chasing down the model hierarchy

Error Bounds and Monotonicity

- Static RMI tracks maximum empirical under-prediction (`min_err`) and over-prediction (`max_err`) across training keys
- Existing keys are guaranteed to lie within $[\hat{y} - \text{min_err}, \hat{y} + \text{max_err}]$ without false negatives
- Non-monotonic models break bounded search: lookups for missing keys can land outside the error window unless the search expands or uses exponential probing

Local Search Strategies

- **Model Biased Search:** Sets the initial pivot to the model's predicted position $\hat{y}$, then completes binary search within $[\hat{y} - \text{min_err}, \hat{y} + \text{max_err}]$
- **Biased Quaternary Search:** Probes $\hat{y} - \sigma$, $\hat{y}$, and $\hat{y} + \sigma$ concurrently to leverage memory prefetching, falling back to quaternary partitioning
- **Exponential Search (ALEX):** Probes exponentially increasing offsets ($2^0, 2^1, 2^2, \dots$) from $\hat{y}$ without precomputed error bounds

Learned Index vs. B-tree (Kraska et al.)

Comparison against stx::btree baseline (128 keys/page); lookup times include local search over records.

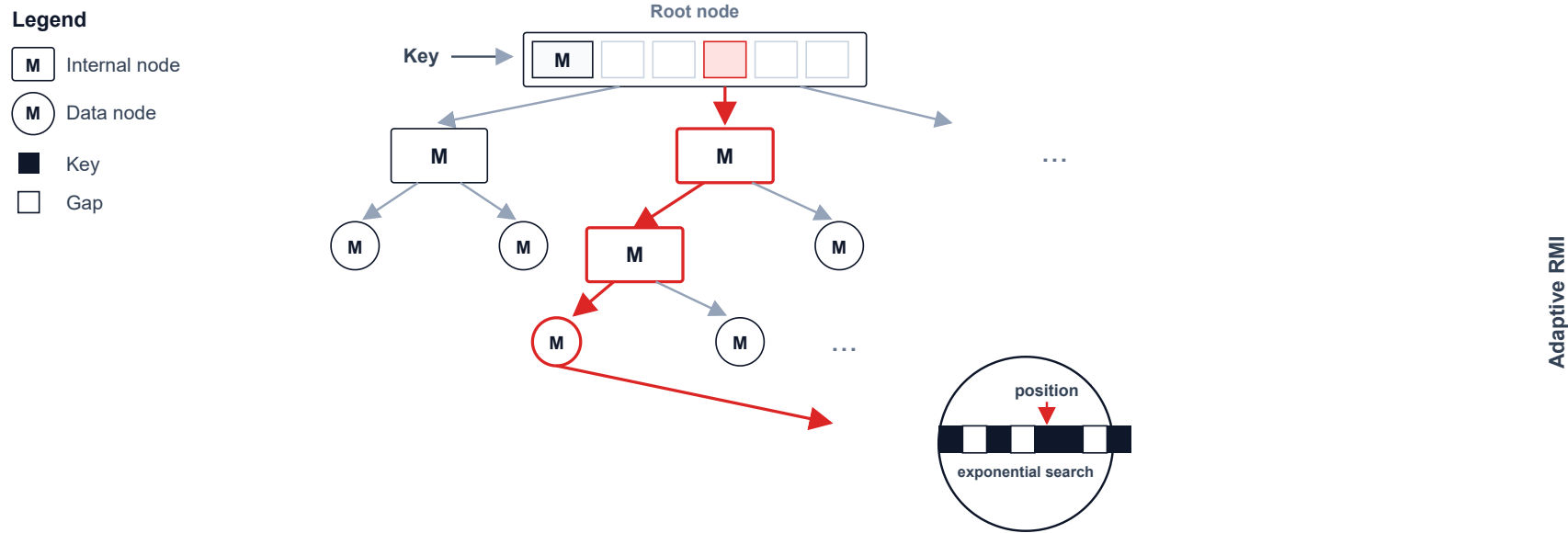
Dataset	B-tree Baseline	RMI (10k)	RMI (100k)	RMI (200k)
Maps	265 ns, 13.11 MB	98 ns (2.70×), 0.15 MB	82 ns (3.21×), 1.53 MB	86 ns (3.08×), 3.05 MB
Weblogs	260 ns, 12.98 MB	222 ns (1.17×), 0.15 MB	144 ns (1.81×), 1.53 MB	126 ns (2.07×), 3.05 MB
Lognormal	263 ns, 12.46 MB	178 ns (1.47×), 0.15 MB	152 ns (1.73×), 1.53 MB	146 ns (1.79×), 3.05 MB

- **Hardware sweet spot:** Maps peaks at 100k models (82 ns); scaling to 200k doubles the index footprint to 3.05 MB, and the added cache misses outweigh the marginal search window reduction.
- **Distribution complexity:** Smooth CDFs (Maps) maximize gains early, whereas irregular CDFs (Weblogs) require denser 2nd-stage model arrays to achieve substantial speedups.

Write Limitations in Static RMIs

- The static RMI design targets read-only analytical workloads
- In-place writes into a densely packed sorted array require $O(N)$ shift operations
- Shifting physical keys alters subsequent record ranks, invalidating downstream linear CDF models
- Kraska et al. address inserts using auxiliary delta-indexes merged in batches, incurring dual-lookup overhead and expensive background rebuilds

ALEX Architecture

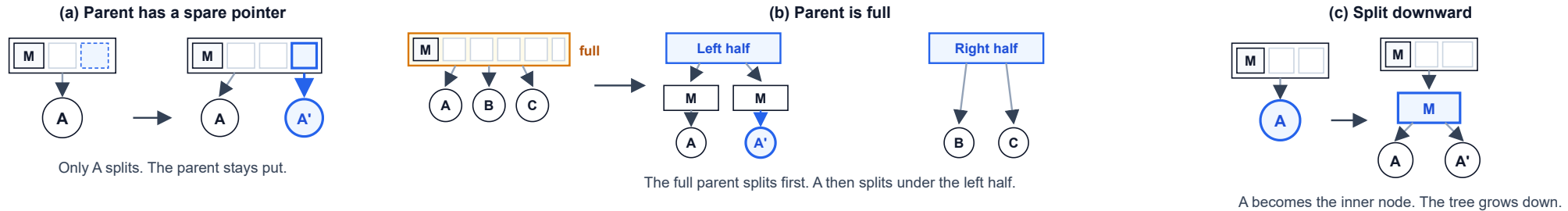


- Updatable, in-memory learned index (Ding et al., SIGMOD 2020)
- Leaf data nodes use **Gapped Arrays** with extra capacity to absorb inserts locally
- Key insertion uses **model-based placement**: keys are routed to predicted slots, shifting keys only to the closest available gap ($O(\log m)$ expected)
- Lookups evaluate the leaf model followed by unbounded **exponential search**

ALEX: Density Bounds & Intra-Node Cost Models

- Data nodes enforce density limits: lower limit $d_l = 0.6$, upper limit $d_u = 0.8$ (average space utilization $\approx 70\%$)
- A data node is marked full when the next insert causes fill density to exceed d_u
- ALEX tracks runtime statistics: exponential search iterations $S(N)$ and shift counts $I(N)$
- Linear intra-node cost model: $C_I(N) = w_s S(N) + w_i I(N) F(N)$, where $F(N)$ is the write fraction
- If empirical operation cost deviates by $> 50\%$ from expected cost, ALEX concludes the local linear CDF model has degraded

ALEX: Structural Maintenance & Range Scans



- **Node Expansion (No Cost Deviation):** Allocates a larger gapped array (n/d_l slots) and rescales model parameters without retraining
- **Node Splits (Cost Deviation $> 50\%$):**
 - *Split Sideways:* Allocates two sibling data nodes under the existing parent, retraining leaf models
 - *Split Downwards:* Converts the data node into an internal routing node with two child data nodes
- **Range Scans:** Each data node maintains an allocation bitmap; range queries scan the contiguous array while using the bitmask to skip empty gap slots

ALEX: Bulk Loading & Cost Model Breakdown

Minimizes expected operation latency across all leaf data nodes $\mathcal{D}$:

$$C(\mathcal{A}) = \frac{\sum_{N \in \mathcal{D}} (C_I(N) + C_T(N)) \cdot K(N)}{\sum_{N \in \mathcal{D}} K(N)}$$

- $\mathcal{A}$ & $\mathcal{D}$: $\mathcal{A}$ is the full index instance; $\mathcal{D}$ is the set of all active leaf data nodes.
- $K(N)$ (**Key Weight**): Number of keys in data node N ; weights cost so larger leaves dominate the metric.
- $C_I(N)$ (**Intra-Node Cost**): $w_s S(N) + w_i I(N) F(N)$.
 - $S(N)$: Expected exponential search iterations ($\approx \log_2$ error).
 - $I(N)$: Expected shifts to the closest gap; $F(N)$ is the workload write fraction.
- $C_T(N)$ (**Traverse-to-Leaf Cost**): $w_d D(N) + w_b B(\mathcal{A})$.
 - $D(N)$: Tree depth of node N (number of dependent pointer chases from root).
 - $B(\mathcal{A})$: Total size of all internal nodes and metadata in bytes.
- w (**Hardware Weights**): Fixed calibration constants converting algorithmic steps to nanoseconds:
 $w_s = 10$ ns, $w_i = 1$ ns, $w_d = 10$ ns, $w_b = 10^{-6}$ ns/byte (1 ns/MB cache penalty).

ALEX vs. Static Learned Indexes

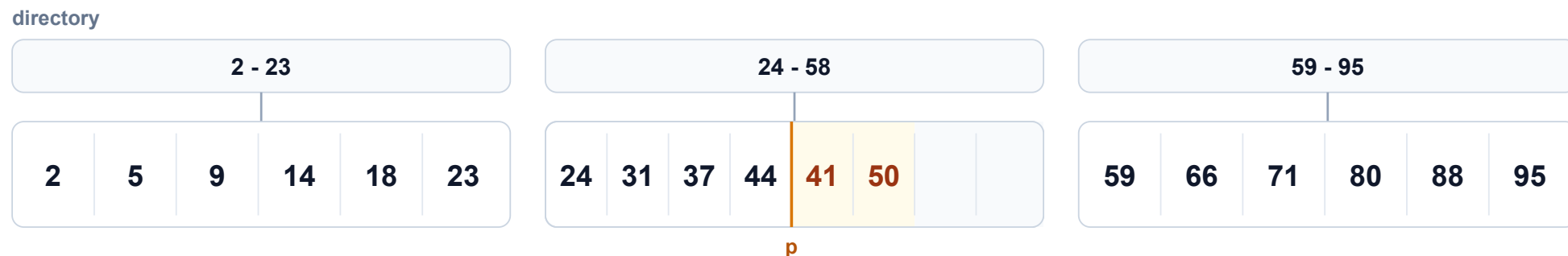
Read-only performance against two-level RMI (linear models, binary search):

- Throughput: up to **2.2×** higher than static RMI
- Index size: up to **15×** smaller index metadata footprint
- Uniform/linear key distributions (YCSB, longlat) yield comparable throughput; peak gains occur on clustered real-world distributions
- Leaf arrays maintain 70% space density to support fast, bounded inserts

ALEX's Structural Flaws: Breaking KISS & Regressing to Trees

- **Regression to Pointer B-Trees:** Learned indexes were invented to replace pointer chasing with arithmetic; ALEX reintroduces a dynamic inner pointer tree, reviving the dependent cache-miss chains learned models set out to eliminate.
- **The Gapped Array Tax:** Enforcing $\approx 30\%$ empty gap space permanently wastes DRAM; sequential scans dilute cache-line transfers with padding and pay bitmap-checking branch stalls on every element.
- **Concurrency & Latch Bottlenecks:** Structural splits (sideways and downwards) propagate mutations up the routing tree, crippling concurrency.
- **KISS Principle Breakdown:** Eliminates deterministic execution in favor of runtime cost counters, $> 50\%$ deviation heuristics, and Fanout Tree algorithms—trading structural simplicity for operational churn.

Chunked Contiguous Learned Extents (CCLE)



- Fixed-length block prefix of size n , appended by a dynamic, sorted contiguous tail
- Direct directory routing by key boundaries; avoids deep internal pointer hierarchies
- Packed data layout achieves $> 90\%$ density without empty gap padding or pointer trees
- Co-locates prefix and tail in a single slab allocation to maximize L1/L2 spatial locality

CCLE: Read & Write Execution Pipeline

- **Read Path:**

- **Two-Level Model Probe:** Block directory has no model, keeps a list of the ranges held by each block, eliminating the massive error rates caused by top level linear models. Block is found using binary search; block models search the area around the prediction via exponential search.
- **Tail Fallback:** On modeled a block miss, falls back to a binary search over the L1-resident sorted tail buffer.
- $O(1)$ **Maintenance:** Cached directory regression counters (S_x, S_y, S_{xx}, S_{xy}) enable instant top-model re-regression on splits.

- **Write Path:**

- **Tail Insertion:** Inserts binary-search and shift directly within the local tail buffer ($O(B)$ in-L1 shift).
- **In-Place Merge:** Tail saturation triggers a register-only backwards merge and closed-form OLS refit.
- **Slab Split:** Full slabs split at midpoint ($C/2$); pushes new entry to directory vector and updates running S -counters in $O(blocksize)$ time.

CCLE: In-Place Merge & Memory Mechanics

- **Zero Scratch Allocations:** When `tail == TailBudget` (128 keys), an in-place backwards merge runs directly within the slab's pre-allocated capacity using CPU register pointers.
- **Single-Pass Safety:** Right-to-left write cursor always stays ahead of read cursors; prevents overwriting unread records without an auxiliary scratch buffer.
- **Closed-Form OLS Fit:** Single-pass accumulation of $(n, S_x, S_y, S_{xx}, S_{xy})$ over at most 1,024 contiguous elements directly in L1/L2 cache.
- **Deterministic Bounds:** Zero cost-model branching, no leaf expansion heuristics, and no bitmap scans—amortized merge latency remains strictly bounded.

Architectural Comparison

Dimension	Kraska et al. (RMI)	ALEX	CCLE (Proposed)
Data Layout	Single contiguous packed array	Gapped array leaf nodes	Dense sorted prefix + sorted tail
Memory Density	100% packed	$\approx 70\%$ fill factor (30% gaps)	$\geq 90\%$ packed prefix, bounded tail
Hardware Parallelism	Embarrassingly parallel read queries; no pointer chains	Serialized pointer dependency hops; gaps break dense SIMD	Co-located slab enables MLP dual-issue; dense SIMD on tail
Index Tree	Multi-stage model grid	Inner routing tree with pointers	Flat extent directory
Write Handling	External delta-index / bulk rebuild	Shift to nearest gap; node split/expand	In-slab tail append; periodic 2-way merge
Model Retrain	Full index retraining	Cost-model triggered (local leaf retrain)	Closed-form regression during slab merge
Memory Alloc	Single static allocation	Dynamic leaf node allocation/splits	Slab allocation per chunk split

Trade-offs and Constraints of CCLE

- High tail insertion overhead: writes shift a sorted tail array, whereas ALEX shifts a few keys to an adjacent gap
- Point lookups must probe both the main extent list, and the active tail segment by binary search
- Dispersed writes across the key space trigger simultaneous tail merges across multiple slabs
- Well suited for monotonic append streams and batched updates; less optimal for uniformly distributed random point writes

Target System Architectures

- Columnar analytical systems: DuckDB, ClickHouse, Apache Arrow
- Stock screener time-series caches and multi-attribute evaluation engines
- High-ingest append-only primary keys: system logging, distributed tracing, and financial tick stores

Initial Benchmarks

- 4M `uint64` keys per file (**32 MB** of keys). Packed key–value data is **64 MB**, about five times this machine’s **12 MB** last-level cache. Rates are medians of 5 runs.
- **Density** is packed key–value bytes / (data + index). Speed cells are **reads / inserts**, CCLE divided by B+. Above 1× is faster. A dash means that side has no operations.
- B+ is `stx::btree_map`, 32 leaf slots and 32 inner slots (leaf node 536 B, inner node 528 B). That pair had the highest mean ops/sec on uniform-shuffled and clusters-shuffled (**3.73 million**). A key array larger than 256 B uses binary search; this leaf is exactly 256 B and uses the library’s linear scan.

Workload	CCLE density			B+ density	Reads / inserts		
	Uniform	Lognormal	Clusters		Uniform	Lognormal	Clusters
Read-only, shuffled	99.4%	99.4%	99.4%	92.7%	1.94× / —	1.32× / —	1.24× / —
Read-only, sorted	99.4%	99.4%	99.4%	92.7%	1.94× / —	1.31× / —	1.27× / —
Read-heavy, shuffled	88.7%	88.5%	88.6%	64.0%	1.45× / 1.04×	1.38× / 1.04×	1.40× / 1.02×
Read-heavy, sorted	99.1%	99.1%	99.1%	60.6%	2.07× / 1.40×	1.42× / 1.40×	1.39× / 1.35×
Mixed, shuffled	88.7%	88.5%	88.6%	64.0%	1.53× / 1.12×	1.37× / 1.10×	1.43× / 1.04×
Mixed, sorted	99.1%	99.1%	99.1%	60.6%	2.08× / 1.54×	1.34× / 1.49×	1.56× / 1.45×

- Read-only bulk-loads every key, then does 4M Zipf point lookups. Read-heavy is 95% Zipf lookups and 5% inserts; mixed is 50/50. Those two bulk-load the first half. Zipf draws and payloads use seed 1. Payload generation is outside the insert timer.
- **Initial benchmarks are promising on randomly generated data. Concurrency, real-world workloads and p99.9 remain to be proven.**

Experimental Setup & Evaluation Plan

- **Hardware & Runtime:** High-throughput benchmarking harness; unmanaged memory buffers ensuring zero runtime GC pause interference
- **Datasets:** 200M 8-byte keys across OSM Longitudes (linear), Lognormal (heavy-tailed), and synthetic weblog time series
- **Workload Spectrum:** Read-only (100R), read-heavy (95R/5W), balanced (50R/50W), and monotonic append-only streams
- **Key Metrics:** Operation throughput (M ops/sec), latency percentiles (p50, p99, p99.9 tail stalls), and physical memory footprint

Summary

- **Kraska et al. (RMI):** Proved learned CDF mappings beat tree branching in read-only workloads
- **ALEX:** Introduced gapped leaf storage, model-based inserts, and cost-driven splitting for mixed read/write workloads
- **CCLE:** Targets compact contiguous layouts by isolating write buffers into local tails and amortizing model retraining across slab merges

Discussion & Q&A

- Sizing the tail: trade-offs between tail lookup latency and slab merge frequency
- Benchmark verification: comparing zero-allocation implementations across CCLE, ALEX, and static RMI
- Open Questions (20 minutes)