nsdi '26

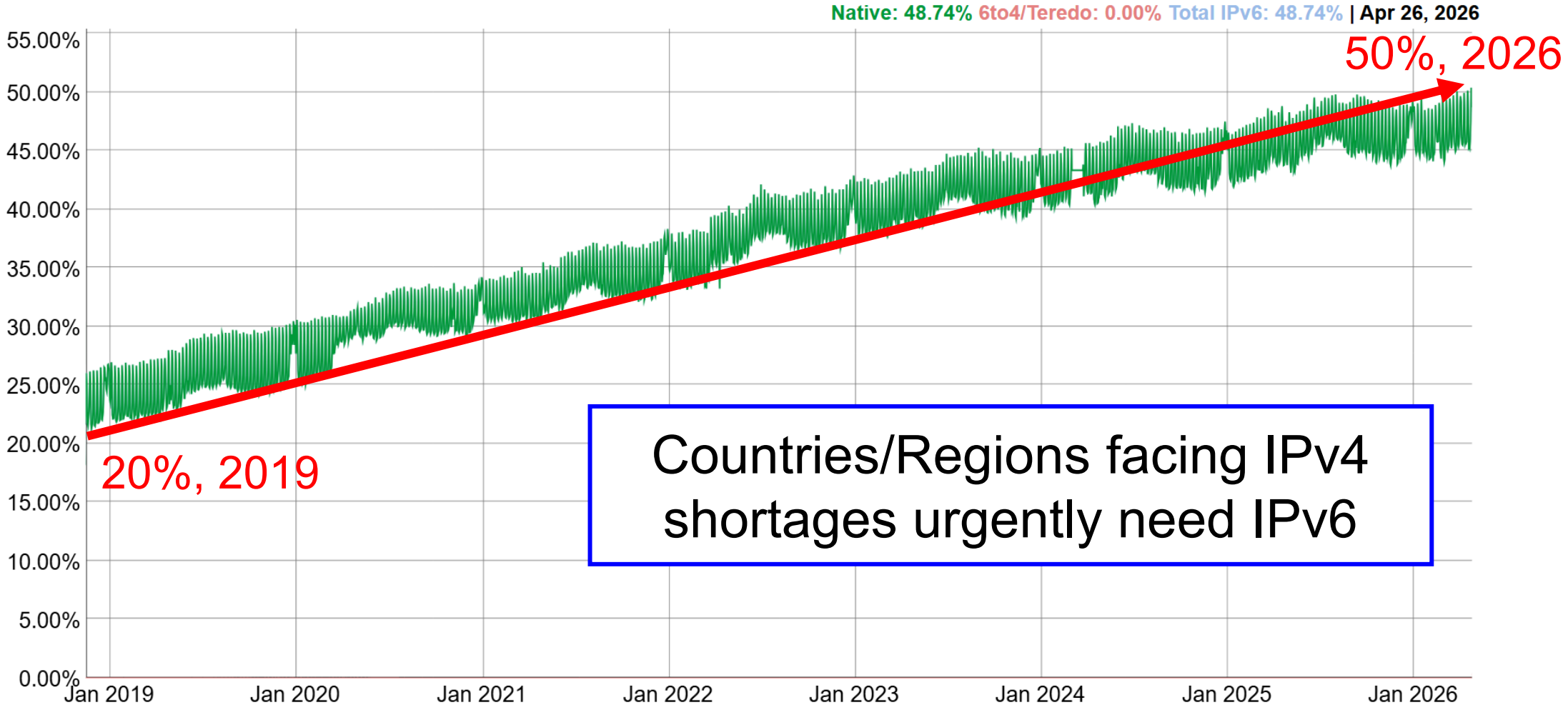
PlanB: Efficient Software IPv6 Lookup with Linearized B+-Tree

Zhihao Zhang[1,3,4], Lanzheng Liu[1], Chen Chen[1], Huiba Li[1]*,
Jiwu Shu[4], Windsor Hsu[1], Yiming Zhang[2,3]
[1]Alibaba Cloud, [2]NICE Lab SJTU, [3]NICE Lab XMU, [4]Tsinghua University

Alibaba Cloud

The Evolving IPv6 Landscape

- Global IPv6 traffic is surging



Google IPv6 Statistics

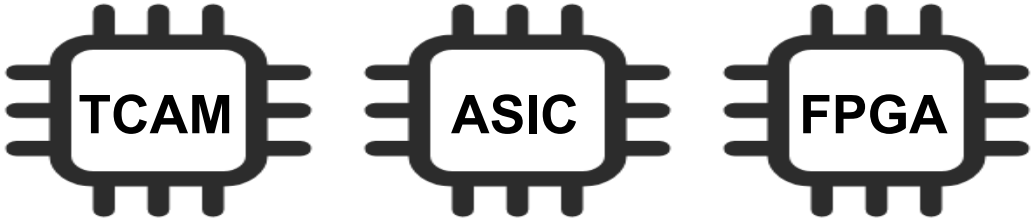
The Evolving IPv6 Landscape

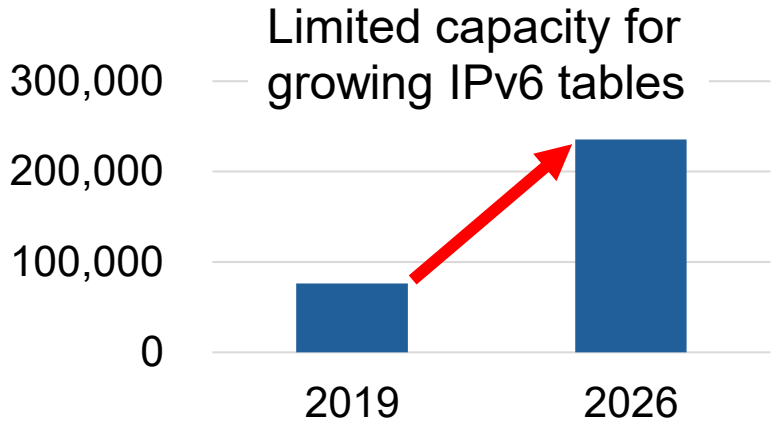
Rapid FIB Growth

Name	Time	Physical Location	IP Prefix Distribution (Top 5)	# of Prefixes
rrc00-19	20190801	Amsterdam, NL	48(46.52%),32(17.13%),44(6.13%),40(5.40%),36(3.97%)	76342
rrc00-20	20200801	/	48(47.87%),32(15.52%),44(6.14%),40(5.31%),36(4.15%)	95504
rrc00-21	20210801	/	48(41.82%),32(14.59%),44(8.73%),40(6.66%),64(4.81%)	143823
rrc	Highly Skewed Distribution		48(47.28%),32(13.31%),44(8.32%),40(7.73%),36(3.76%)	168572
rrc00-23	20230801	/	48(46.39%),32(11.83%),44(9.09%),40(7.89%),36(3.65%)	199801
rrc00-24	20240801	/	48(45.27%),32(11.19%),44(9.40%),40(9.01%),36(3.64%)	226222
rrc00-25	20250801	/	48(44.55%),32(11.00%),40(10.11%),44(9.69%),36(3.89%)	235466
rv1	20250801	Johannesburg, SA	48(44.78%),32(11.07%),40(10.00%),44(9.47%),36(3.76%)	235038
rv2	20250801	Singapore, SG	48(44.93%),32(10.92%),40(9.85%),44(9.56%),36(3.71%)	238498
rv3	20250801	Tokyo, JP	48(46.03%),32(11.42%),40(10.30%),44(9.19%),36(3.83%)	226893
rv4	20250801	New York, USA	48(44.88%),32(11.30%),40(10.17%),44(9.69%),36(3.82%)	229417
rv5	20250801	Rio de Janeiro, BR	48(45.37%),32(11.26%),40(9.36%),44(9.24%),36(3.86%)	227833
rv6	20250801	Sydney, AU	48(44.92%),32(11.21%),40(10.15%),44(9.53%),36(3.80%)	230672
rv7	20250801	Frankfurt, DE	48(44.56%),32(11.02%),40(10.02%),44(9.49%),36(3.82%)	234916

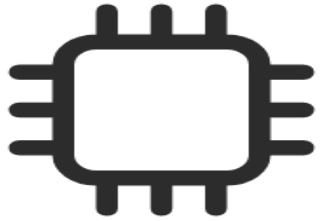
The Dilemma in IP Lookup

- **Hardware (TCAMs, ASICs, FPGAs):** High throughput but expensive, power-hungry, and lack NFV flexibility

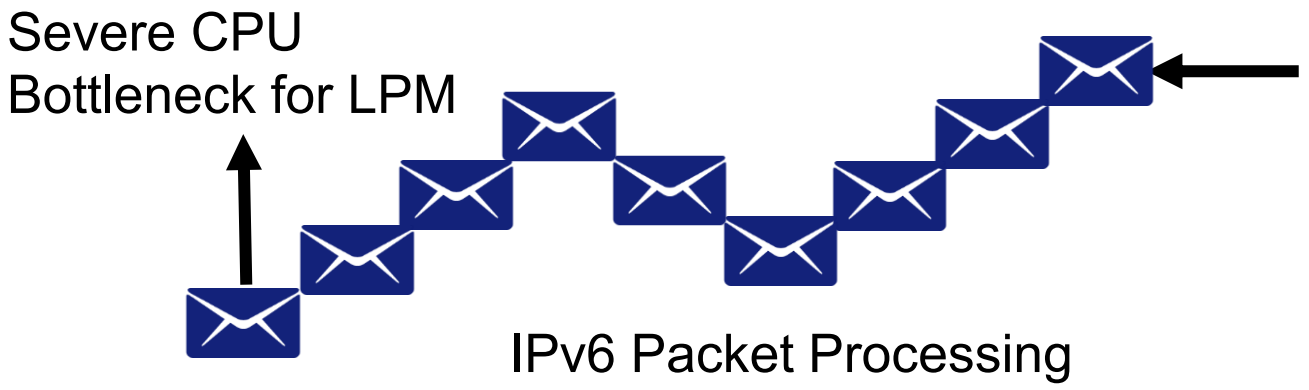




- **Commodity Software:** Flexible and cost-effective, but IP lookup (Longest Prefix Match - LPM) remains a severe CPU bottleneck

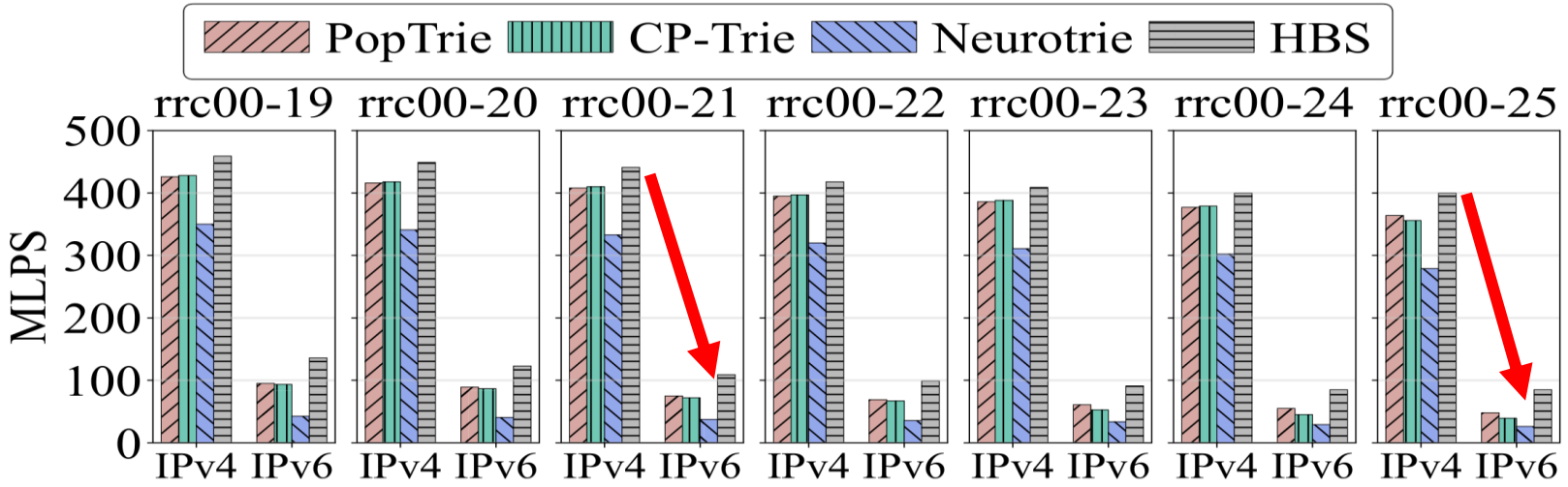


General CPU



The IPv6 Penalty in Software

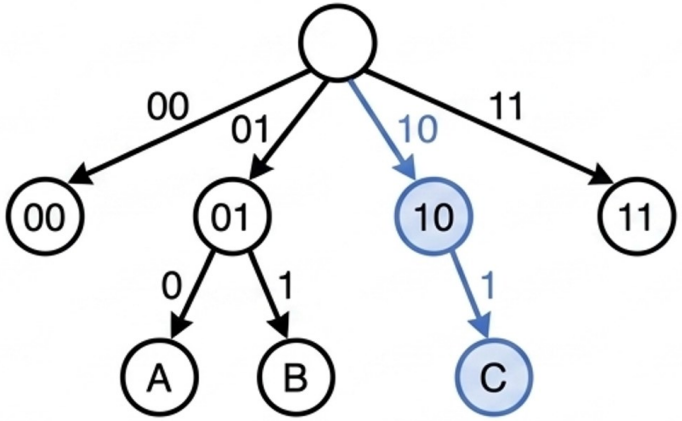
- Transitioning from IPv4 to IPv6 causes a massive performance drop in existing algorithms including Trie-based and Hash-based methods
  - Longer addresses (128-bit) expand memory footprints (10×–50×)
  - Data structures spill out of fast CPU caches (SRAM) into slow DRAM and Lookup cost increases 3×–10× vs. IPv4



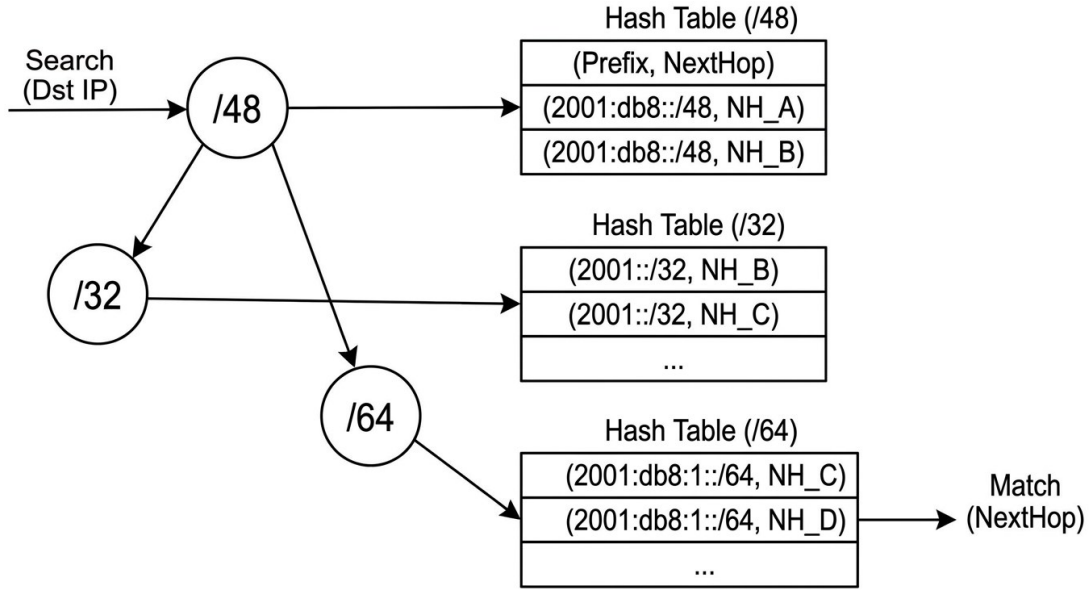
Why? Deeper tries, more memory accesses, and L3 cache misses.

The Core Issue: 2D Problem vs. 1D Solutions

- LPM is inherently a 2D search problem: matching both **prefix value** and **prefix length**
  - Trie-based: Traverses space bit-by-bit. Deep structures = extensive memory indirection and cache misses
  - Hash-based: Probes hash tables based on length. Susceptible to collisions and lacks strictly bounded worst-case times



Trie-based method



Hash-based method

PlanB At a Glance

- High-speed IPv6 lookup on commodity hardware

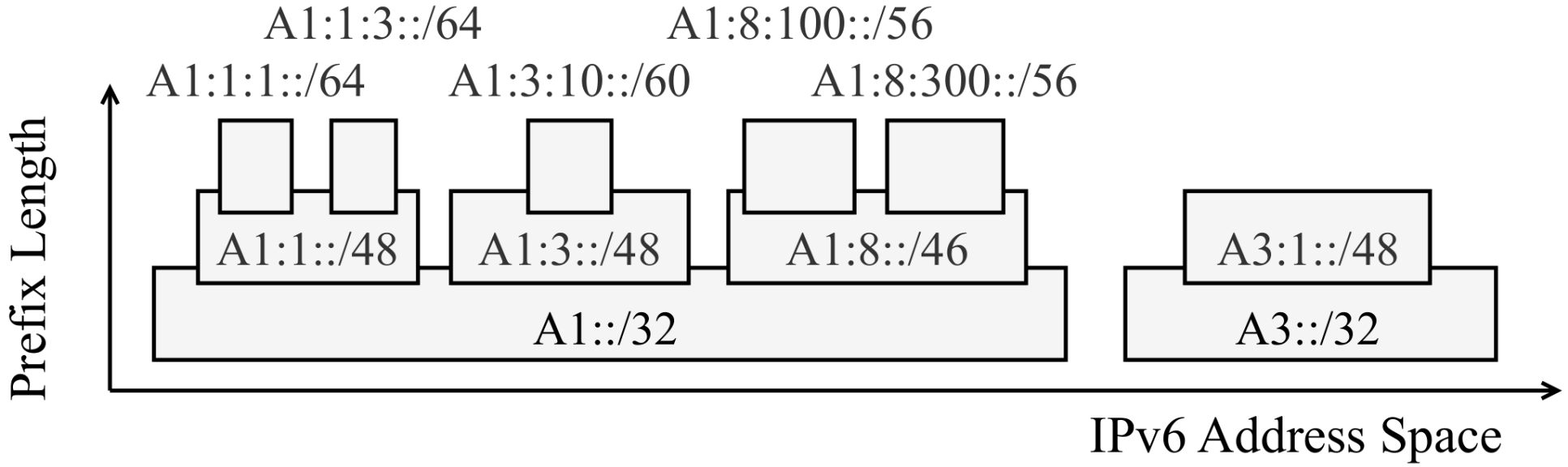
Dimensionality Reduction: Transform 2D LPM into a 1D elementary interval search

Cache-Aligned Data Structure: Linearized, pointer-less B^+-Tree

Hardware-Symbiotic Logic: Vectorization, batching, and branch-free/loop unrolling

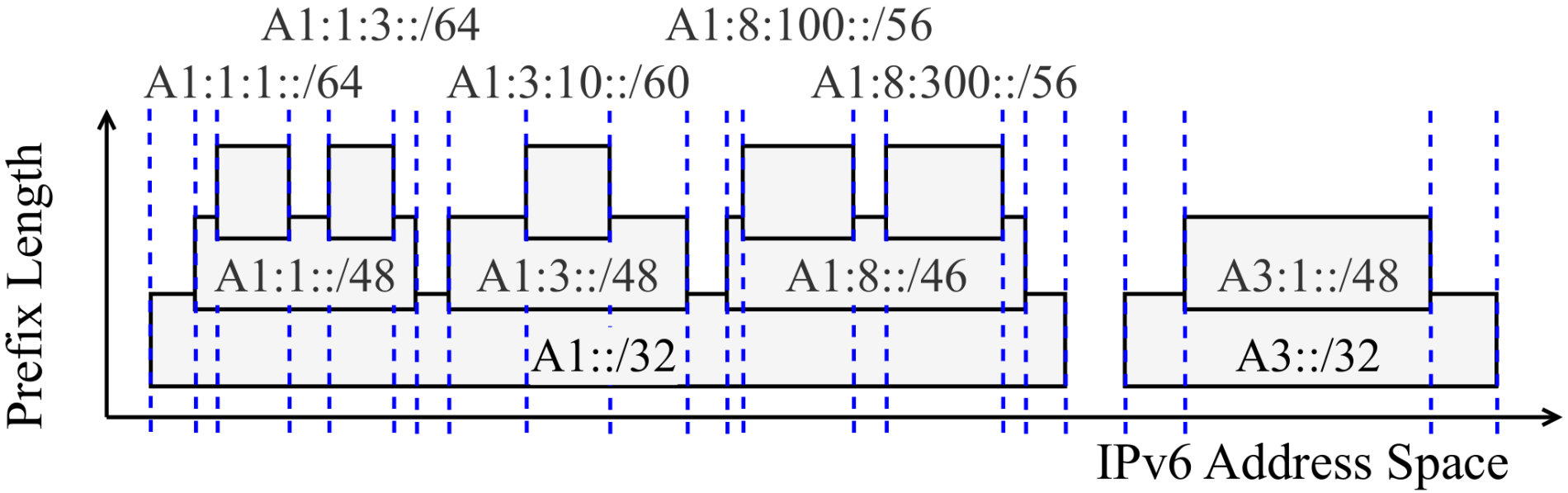
Step 1 - Range Conversion

- Flattening prefix complexities into deterministic boundaries
  - Convert every rule (Prefix + Length) into a closed IP address interval: [start_address, end_address]
  - Visually, prefixes become overlapping rectangles; height dictates priority (prefix length)



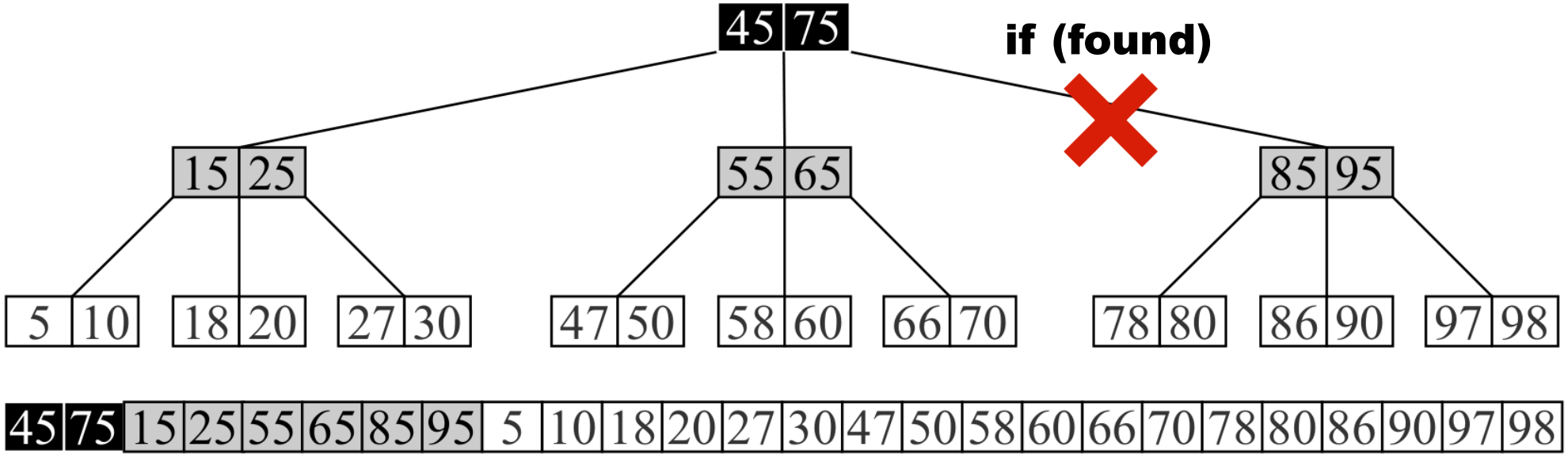
Step 1 - 1D Interval Partitioning

- Flattening prefix complexities into deterministic boundaries
  - Collect and sort all start/end boundaries to partition the 128-bit IPv6 space into a sorted array of **non-overlapping elementary intervals**
  - Result: **A 1D search problem**. Finding the matching prefix is now a simple array predecessor search



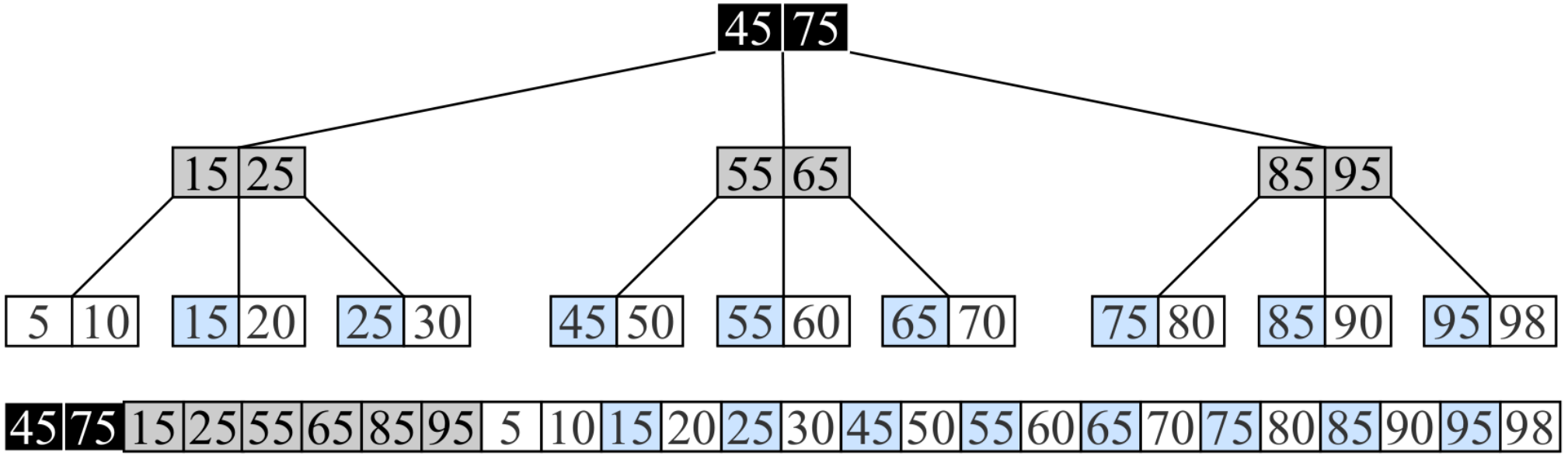
Step 2 - Why Not a Standard B-Tree?

- A Flat Array B-Tree conceptually organizes a search tree in a contiguous array
  - The Catch: Requires checking if the target is found at every internal node
  - **Introduces conditional branches**: kills CPU pipeline efficiency and makes SIMD batching difficult



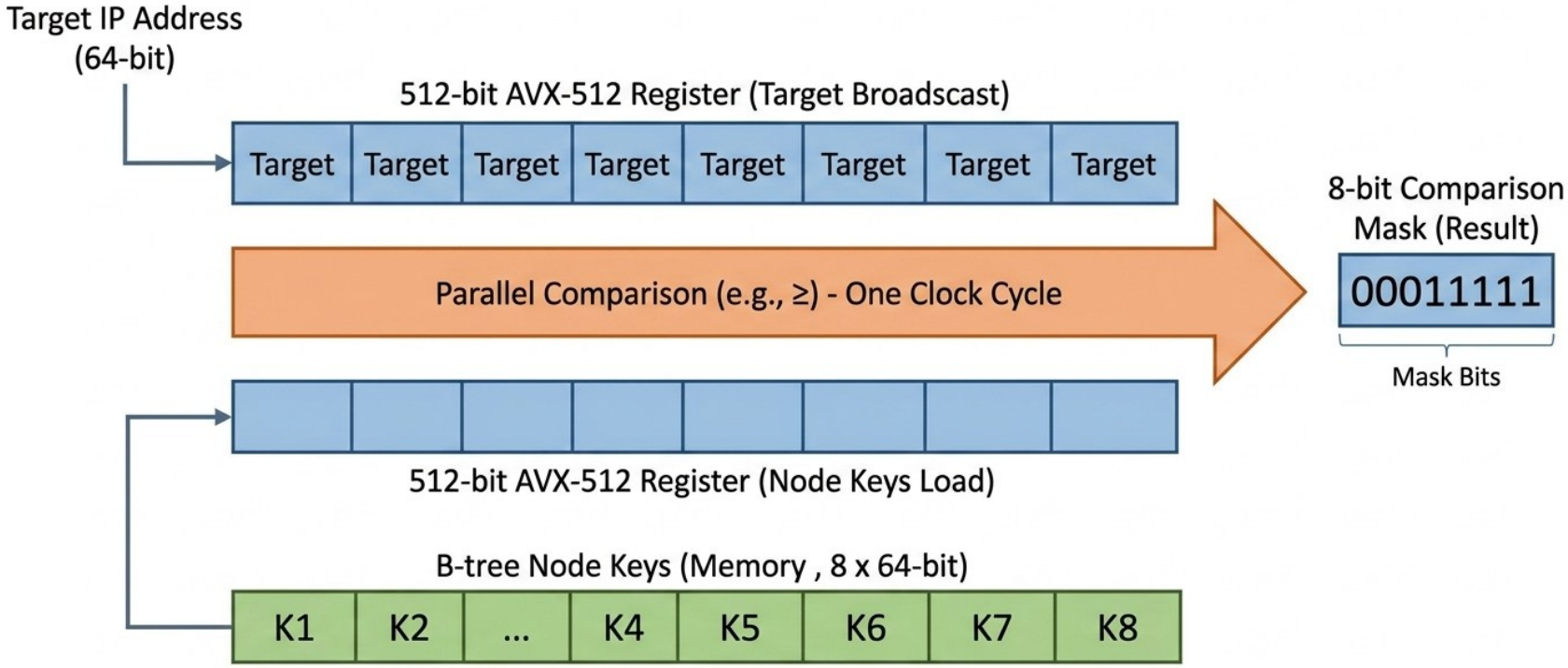
Step 2 - The Linearized B$^+$-Tree

- **Data Separation:** Internal nodes only guide the search; actual keys reside exclusively in leaves
- **Pointer-Less:** Children are found via simple arithmetic (child_start = level_start + offset)
- **Cache-Aligned:** Tuned each node fits perfectly into a 64-byte cache line (e.g., 8 keys = 9-ary tree)



Step 3 - Vectorization & SIMD

- Standard searches use sequential scalar comparisons
- PlanB uses AVX-512 (also supports other vectorization like AVX2 and ARM NEON):
  1. Broadcast target address to a 512-bit register
  2. Load 8 node keys into a second register
  3. Perform a single parallel comparison in one clock cycle



Step 3 - Branch-Free Traversal

- CPU branch mispredictions cost dozens of cycles
- PlanB eliminates if/else jumps entirely
  1. SIMD output generates a bitmask
  2. Uses popcnt (population count) on the bitmask to directly calculate the exact index of the next child branch

Listing 1 Vectorized Search on Linearized B$^+$-tree.

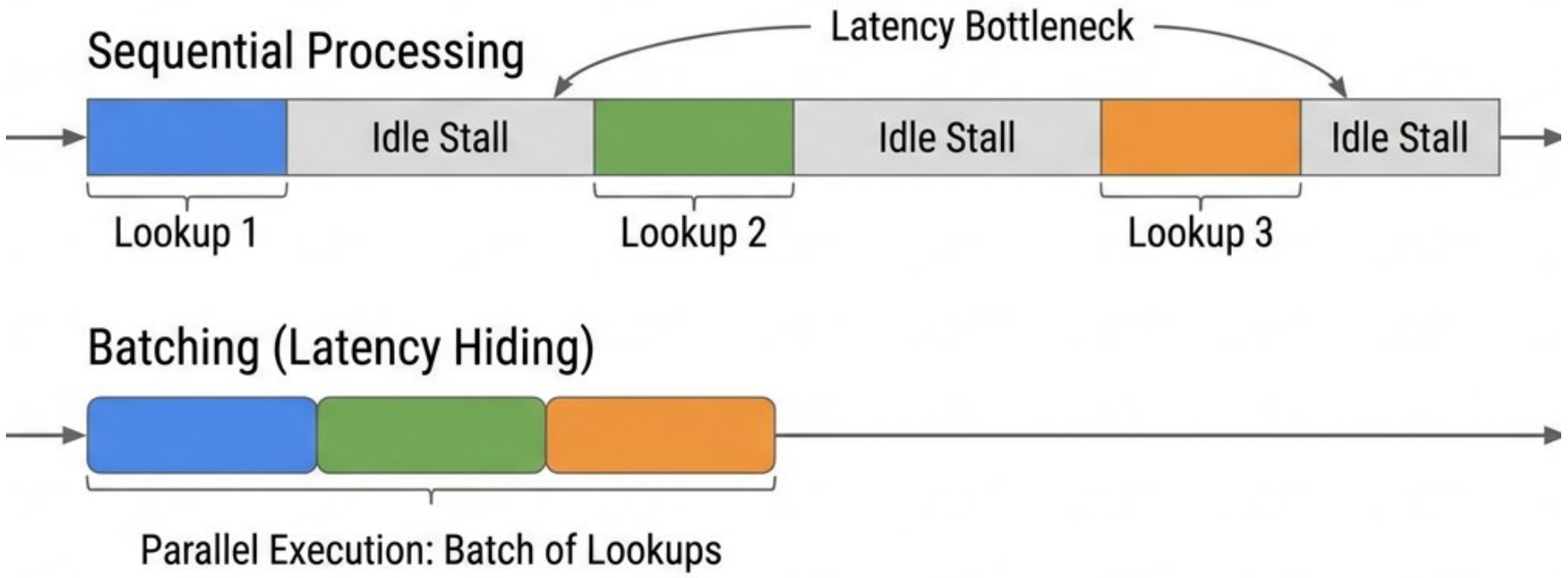
```
1  // k: keys per node, key: B+-tree keys array
2  long LBTree::vectorized_search(uint64_t prefix) {
3      long i = 0,c = -1, d = tree_depth;
4      __m512i vx = _mm512_set1_epi64(prefix);
5      do {
6          i = (k + 1) * i + (c + 1) * k;
7          __m512i node = _mm512_load_si512(&key[i]);
8          c = _mm512_cmpge_epu64_mask(vx, node);
9          c = __builtin_popcount(c);
10     } while (--d);
11     return i+c;
12 }
```

Vectorized Comparison

popcnt: Get Child Index

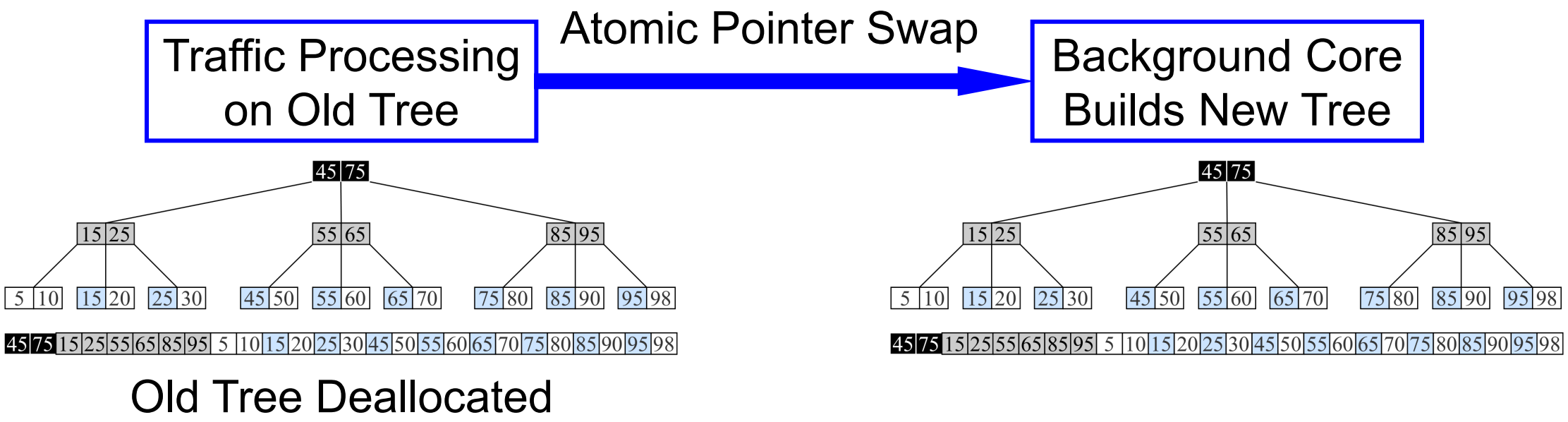
Step 3 - Batching & Loop Unrolling

- **Batching:** Processes lookups in groups. Hides SIMD multi-cycle latency by pipelining independent lookups
- **Loop Unrolling:** Because the B^+-tree depth is strictly bounded (e.g., 6-7 levels for 1M+ intervals), loops are fully unrolled at compile time



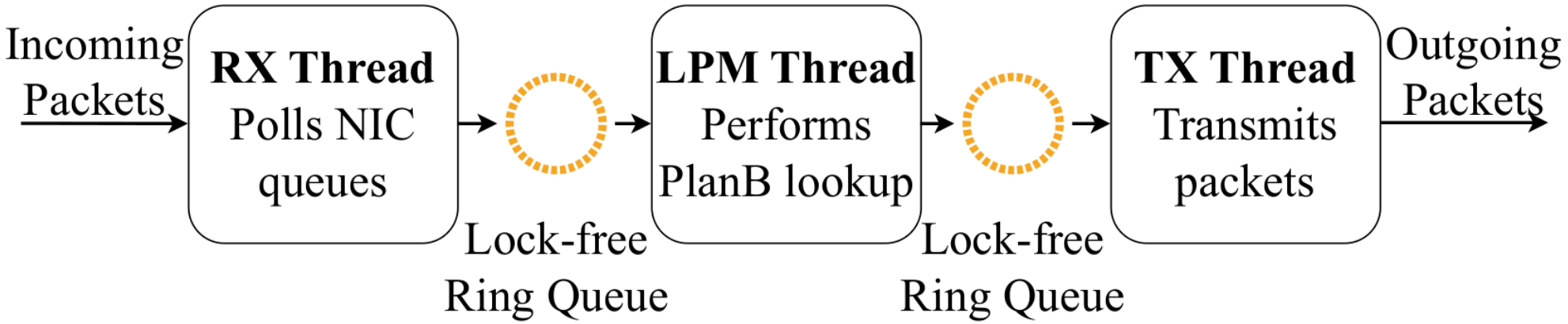
Dynamic FIB Updates

- **Background Rebuild:** Generate the new 2D-to-1D structure and Linearized B^+-Tree on a separate core
- **Atomic Swap:** Update a single pointer to redirect traffic to the new tree
  - Zero contention, zero locks, deterministic update times



DPDK Implementation

- Integrated into DPDK using an **RX-LPM-TX** Pipeline
  - Isolates LPM threads tightly to a single CPU cache group (e.g., AMD CCX)
  - Decouples packet I/O from memory-intensive IP routing to maintain strict L3 cache residency

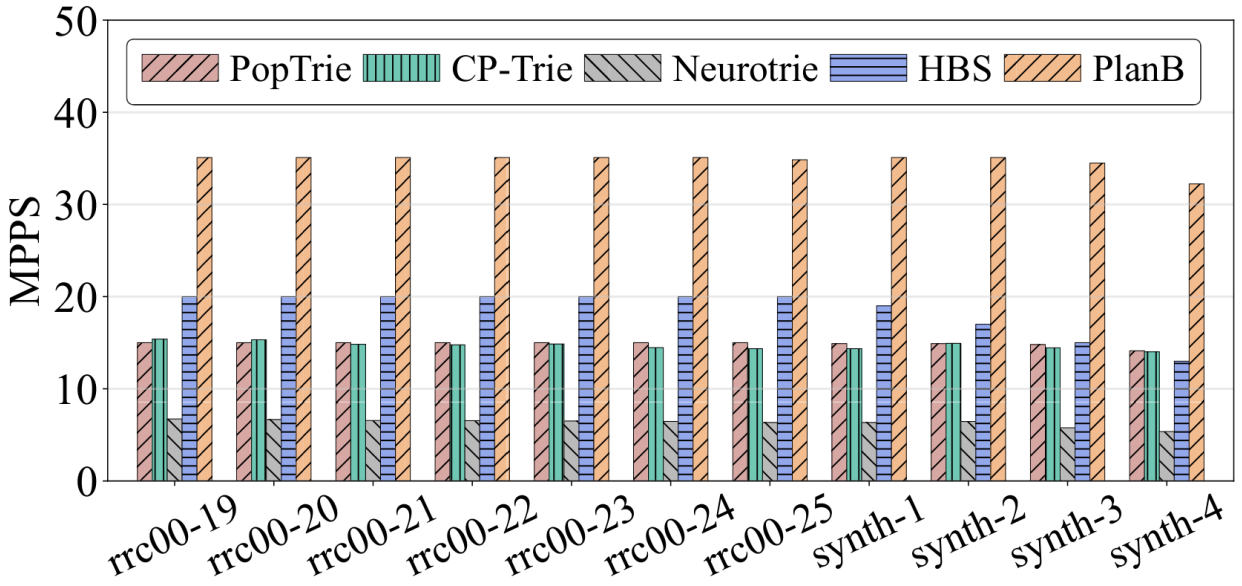


Evaluation Setup

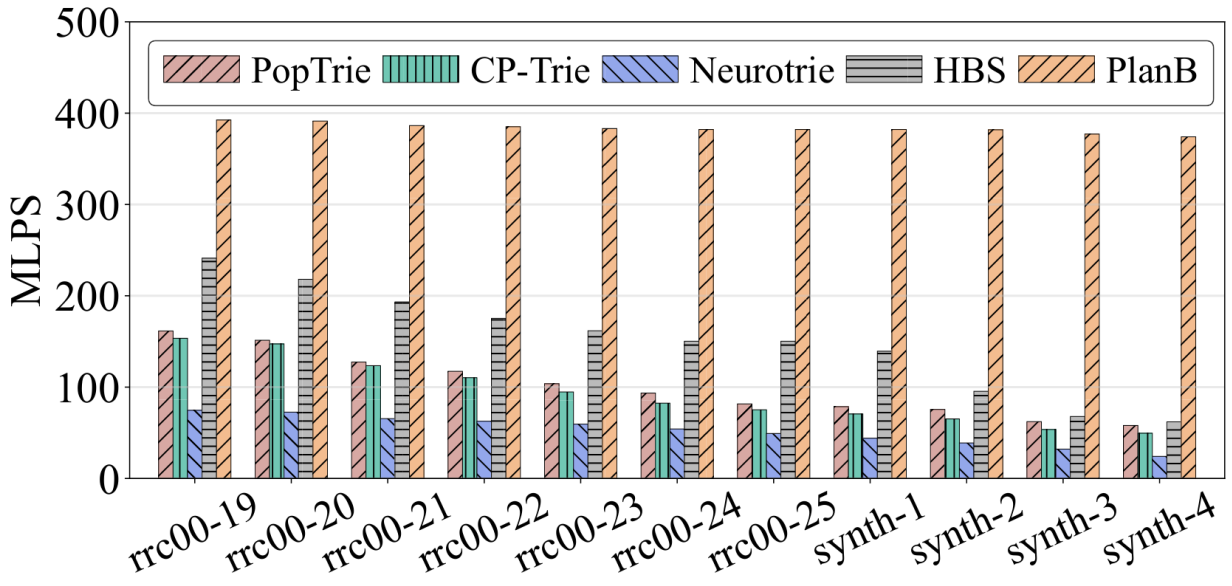
- Hardware
  - 24-core Intel Xeon (3GHz)
  - 12-core AMD Ryzen 9 (Zen5/5.1GHz, Zen5c/3.3GHz)
- Datasets
  - 14 real-world FIBs from RIPE/RouteViews (up to 235K prefixes)
  - 4 Synthetic FIBs (up to 1M prefixes)
- Baselines
  - PopTrie, CP-Trie, Neurotrie, HBS

System Throughput & Single-Core Speed

- **DPDK System Throughput:** Sustains high-speed L3 forwarding (1.7×–5.8× faster than baselines)
- **Single-Core Speed:** Peaks at 390 MLPS on AMD CPU (1.6×–12.5× faster than state-of-the-art baselines)



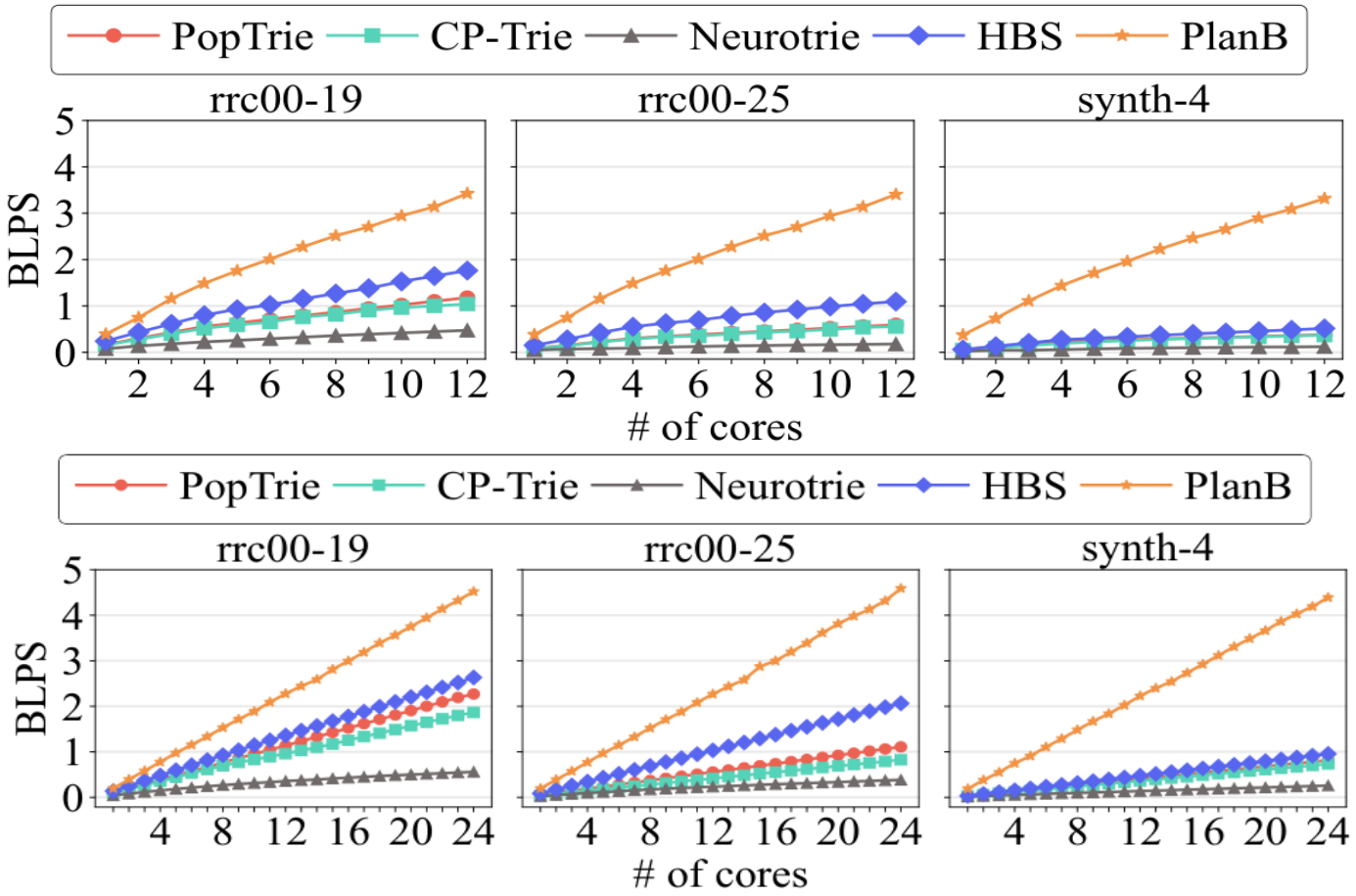
DPDK Throughput



Single-Core Speed on AMD CPU

Multi-Core Scalability

- Achieves near-linear multi-core scalability
  - AMD Platform: Up to 3.4 BLPS (1.8×–18.3× faster than baselines)
  - Intel Platform: Up to 4.6 BLPS (1.7×–14× faster than baselines)

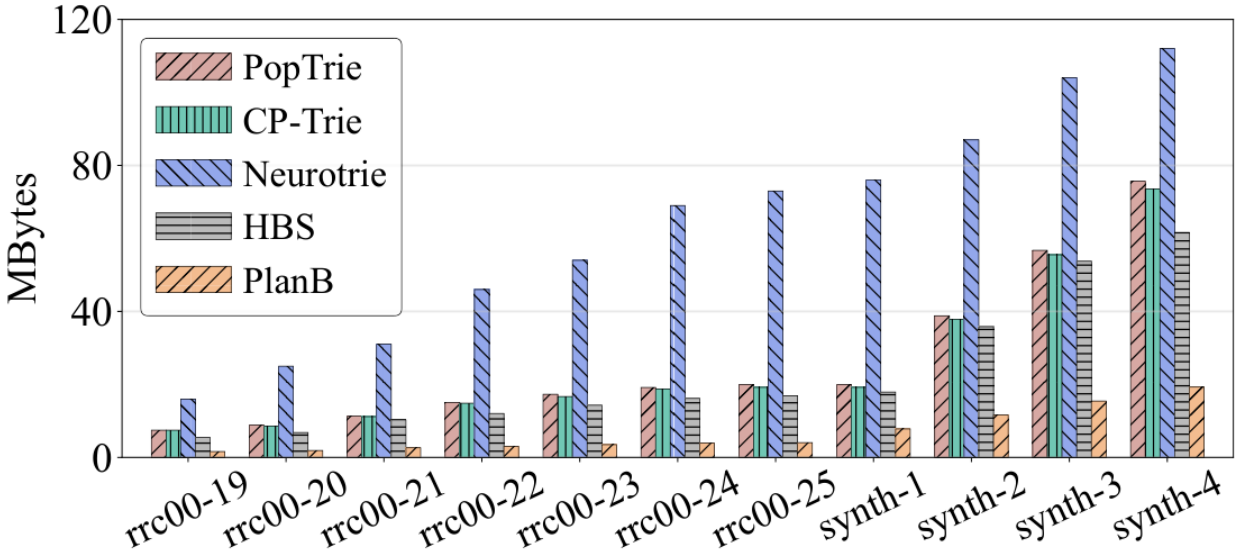


AMD Multi-Core Speed

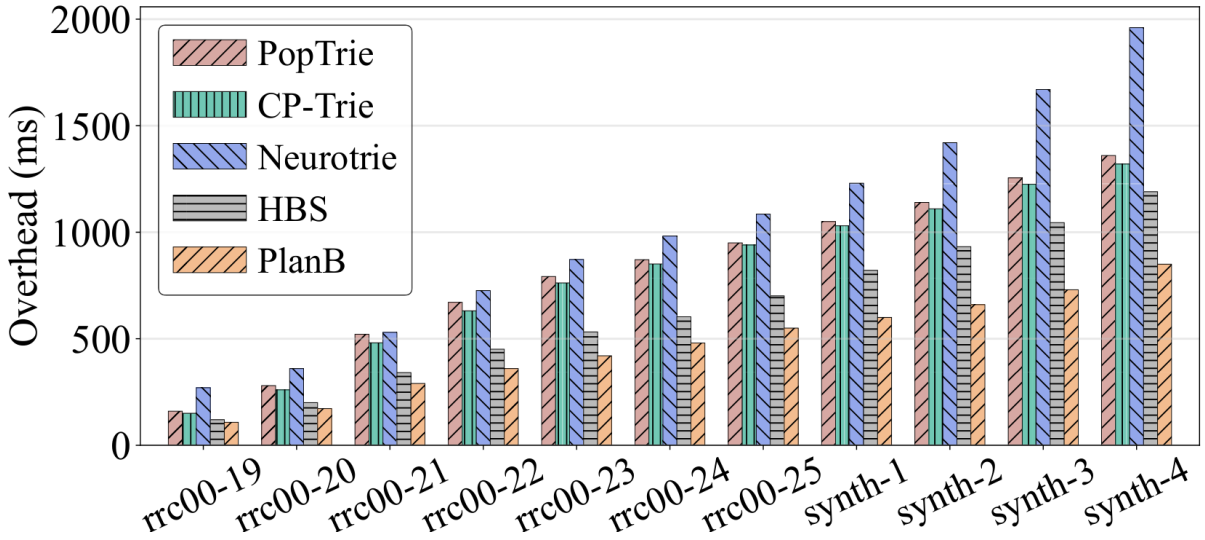
Intel Multi-Core Speed

Memory Overhead & Update Speeds

- Memory footprint reduced by 56.4%–92.5%. Fits entirely inside standard L3 CPU Cache (~36MB for 1M prefixes)
- Updates are faster because pointer-less arrays are highly compact. Rebuild overhead is heavily amortized



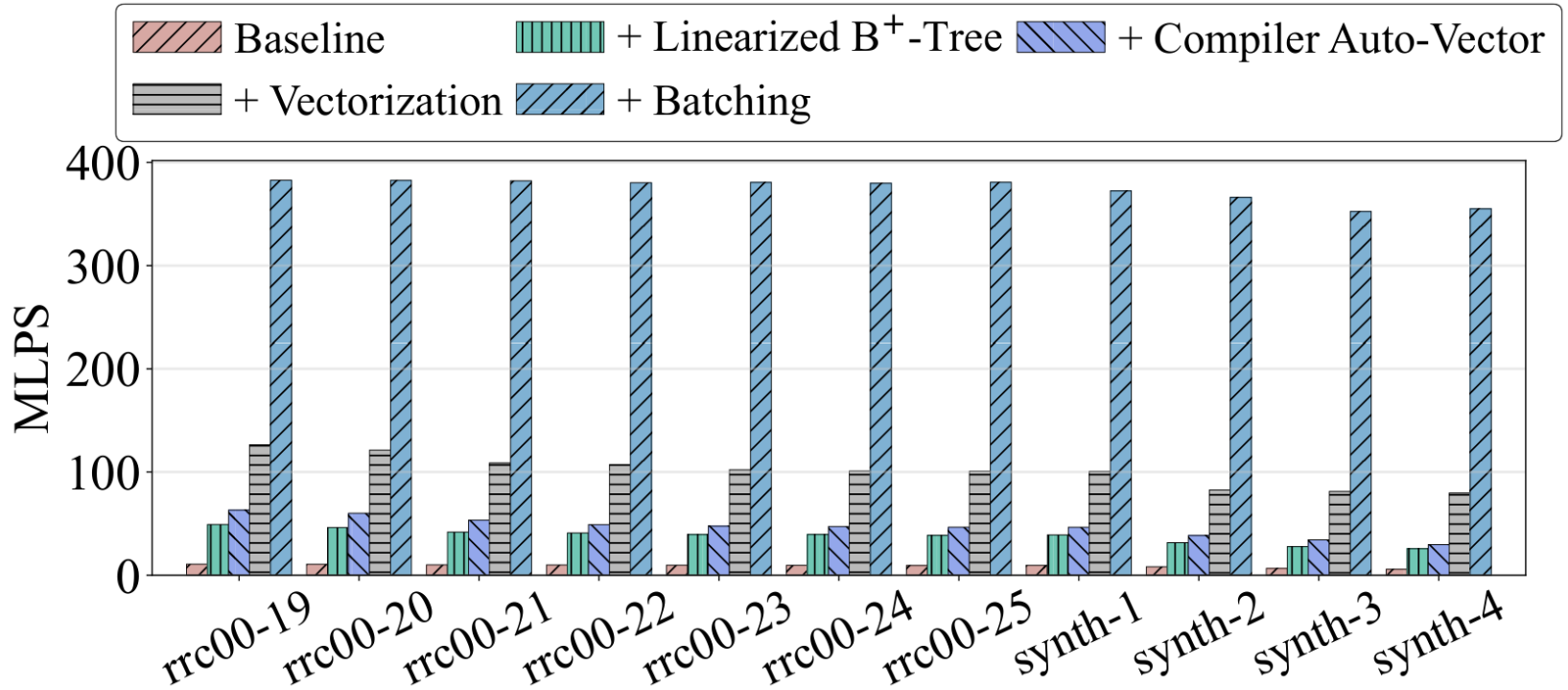
Memory Overhead



Update Overhead

Ablation Study

- Where does the performance come from?
  - Baseline (standard binary search): ~10 MLPS
  - Linearized B^+-Tree: 3.8×– 4.6× gain
  - AVX-512 SIMD: Additional 2×– 2.7× gain
  - Batching & Unrolling: Final 3×– 4.5× boost, reaching 390 MLPS



Discussion

- Hardware Adaptability
  - Diverse Platforms: Easily maps to hardware like FPGAs and ASICs
  - Memory Efficiency: Leverages standard on-chip SRAM
  - Flexibility: Functions efficiently without advanced SIMD

- Performance Stability
  - Near-Constant Throughput: Highly resilient to unpredictable traffic/malicious attacks. Sustains performance via a fixed, shallow search depth

- Hybrid Deployment Architecture
  - Control Plane (Software): Handles complex background tree reconstructions.
  - Data Plane (Hardware): Executes high-speed, accelerated lookups

Conclusion

- Problem
  - IPv6 software lookup is bottlenecked by treating 2D Longest Prefix Match as two inefficient 1D searches
- Insight
  - 2D prefix matching can be seamlessly unified into a single 1D search over non-overlapping address intervals
- Solution
  - PlanB transforms the 2D LPM into an equivalent 1D search problem over elementary intervals and designs the linearized B^+-tree with vectorization, batching, branch-free logic, and loop unrolling
- Result
  - PlanB improves lookup throughput by 1.6×–14× and reduces memory footprint by 56%–92% over state-of-the-art schemes

Thank you!

Github Repo: https://github.com/nicexlab/planb