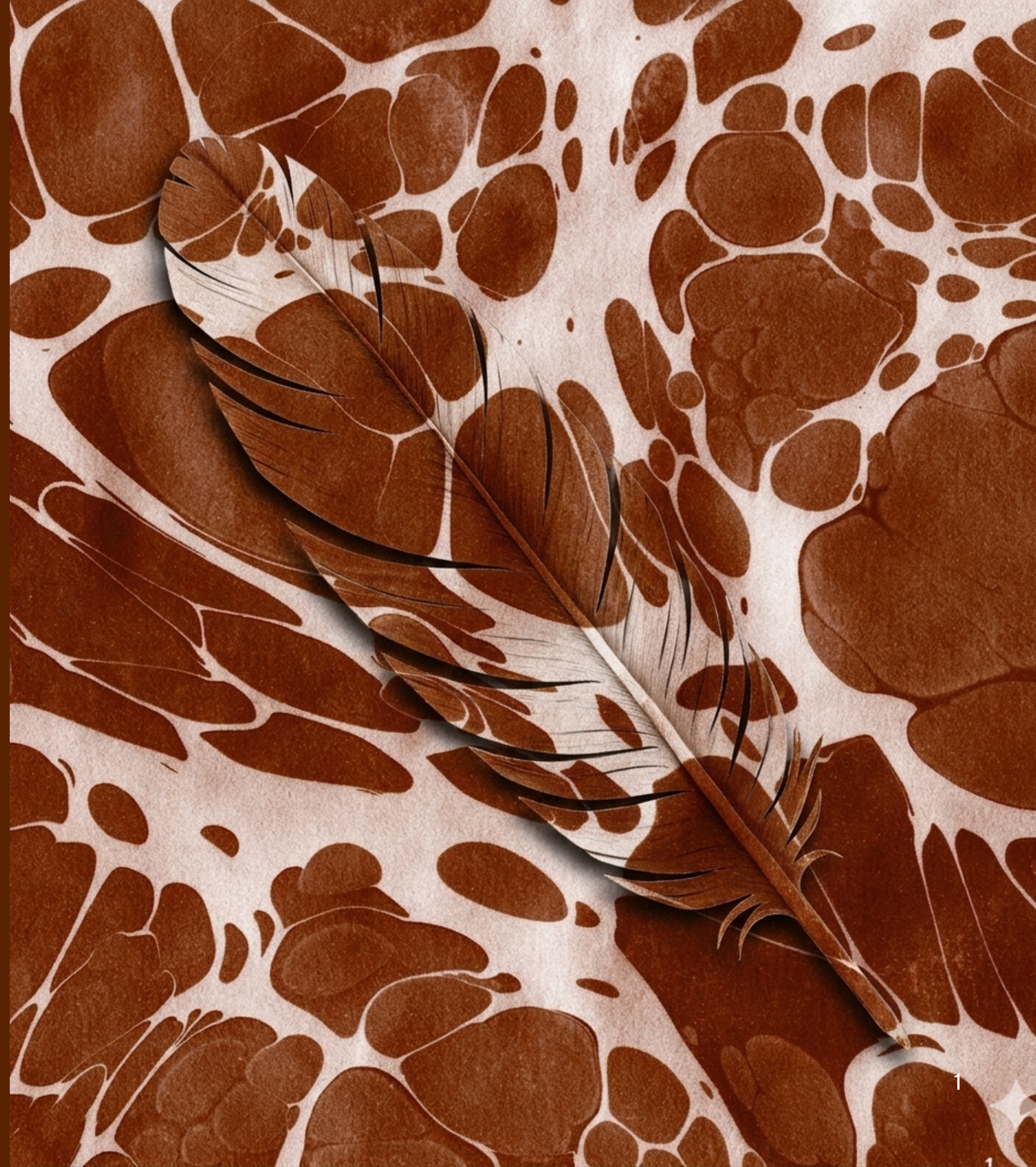
FEATHER
BTP-II Presentation

Saksham Rathi

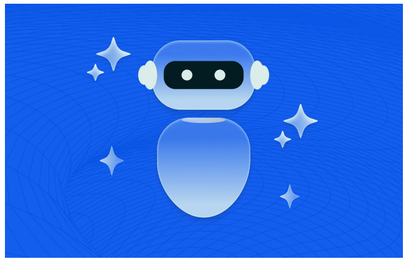
Under **Prof. Mythili Vutukuru**

Indian Institute of Technology Bombay
Spring 2025-26

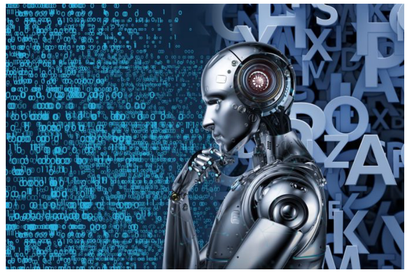


Introduction

- **LLMs** used everywhere



Chatbots (From here)



Code Generation (From: here)

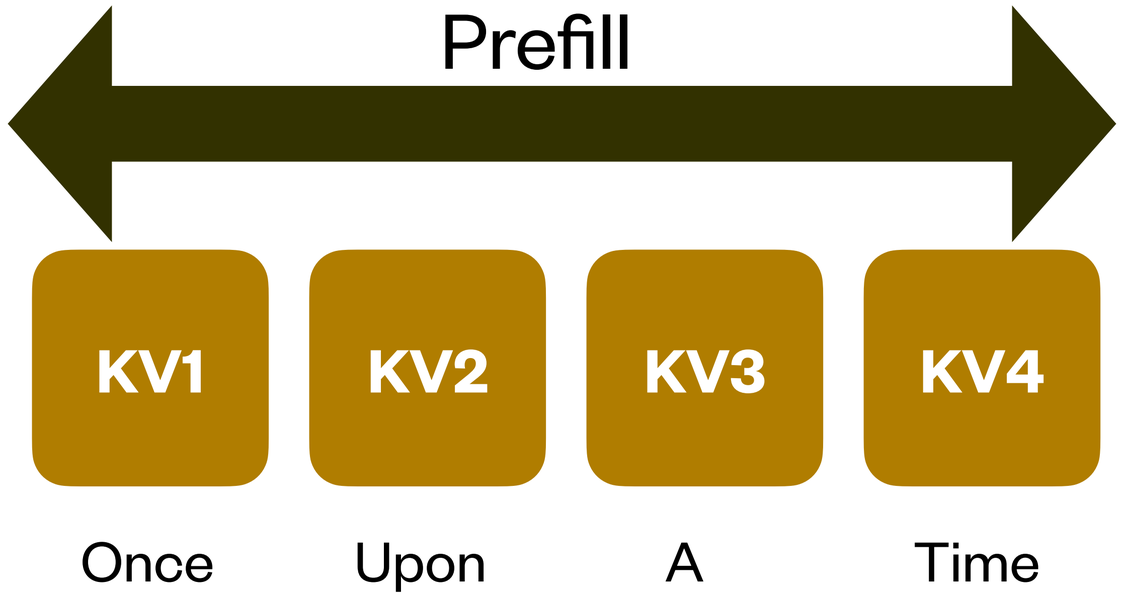
Financial Analysis (From: here)

- Recent Advancements

Schedulers **Parallelization** **Memory Management** **Attention Kernels**

- Still, serving at high throughput remains an **open** problem

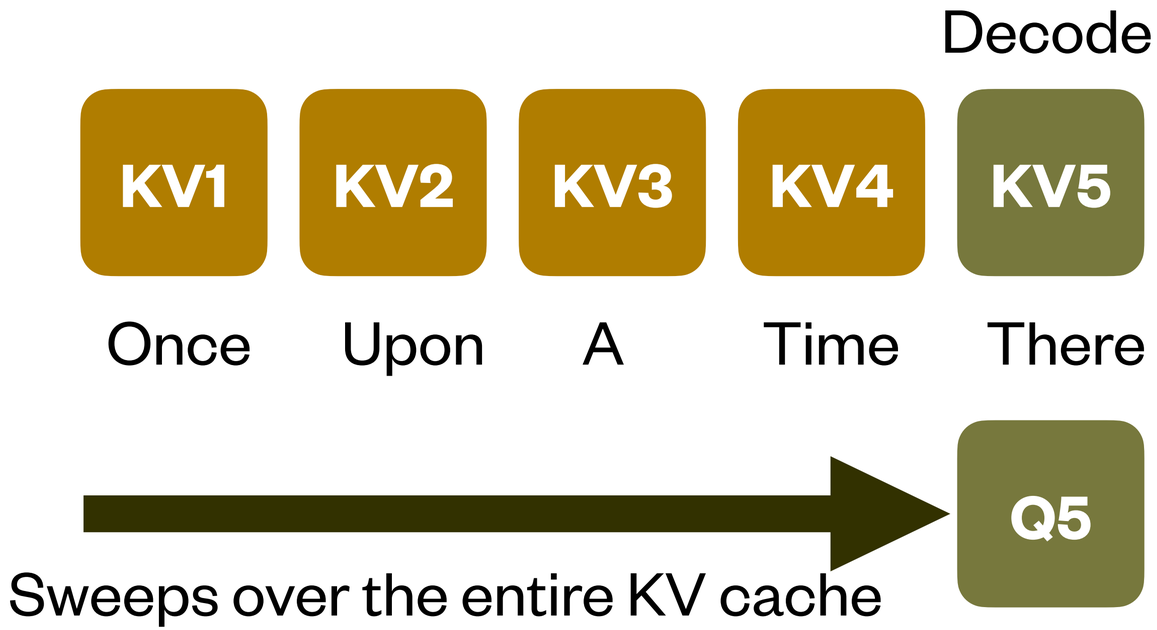
Prefill



Processes all tokens in parallel

Compute-bound

Decode



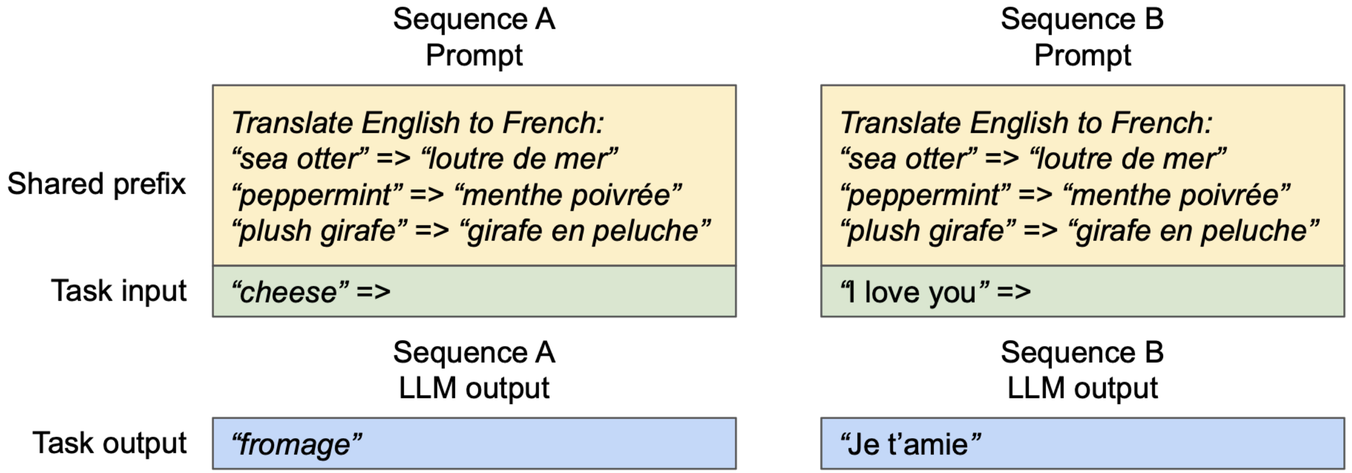
Auto-regressive

Memory-bound

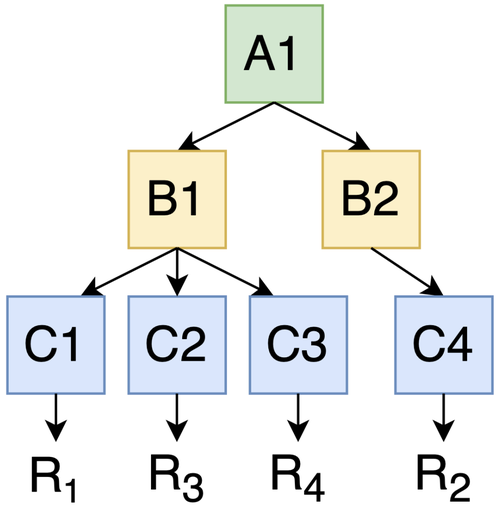
Batching

Prefix-sharing

Few-shot prompting | **System Prompts** | **RAG**

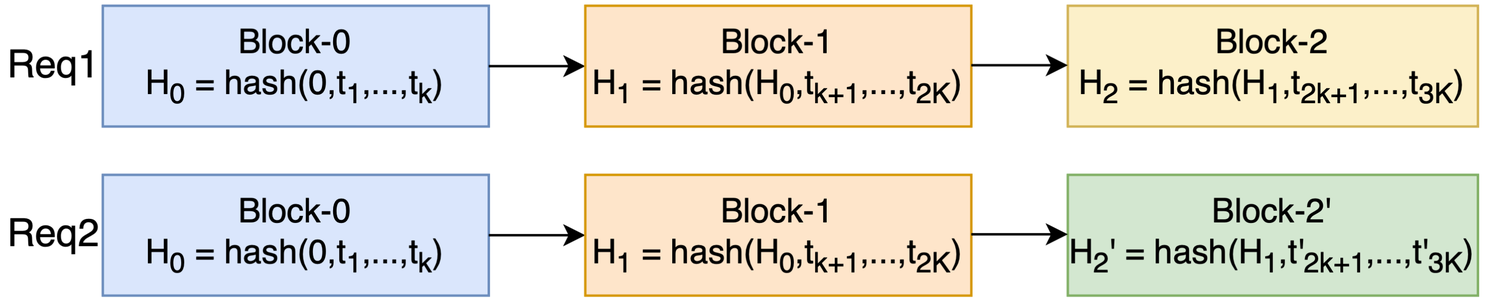


- Reuse KV via prefix caching



SGLang: Radix Tree

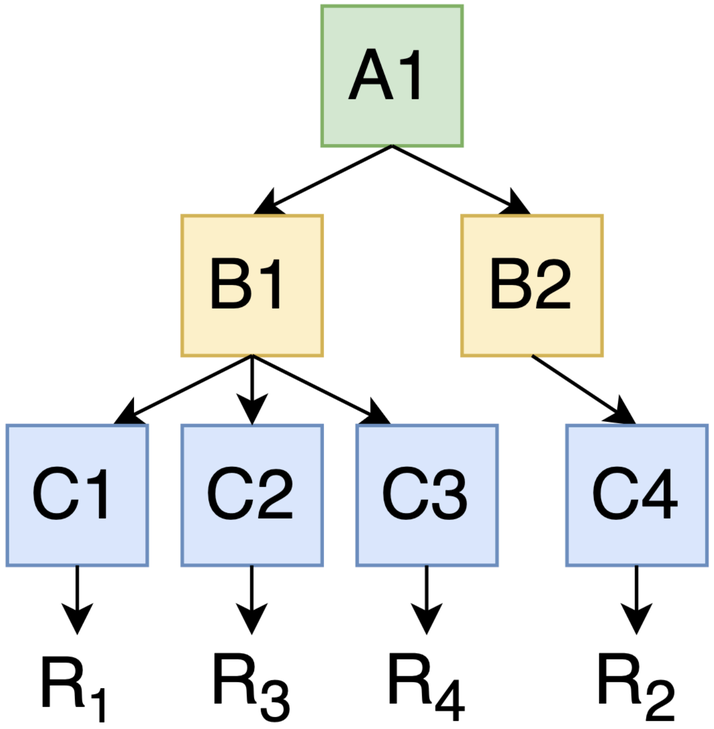
From vLLM



vLLM: Merkle Tree

Prefix-sharing Optimizations

- SGLang proposes DFS-based traversal of radix tree
- Prefix-aware attention kernels to reduce the number of memory accesses



FCFS: R_1, R_2, R_3, R_4

DFS: R_1, R_3, R_4, R_2

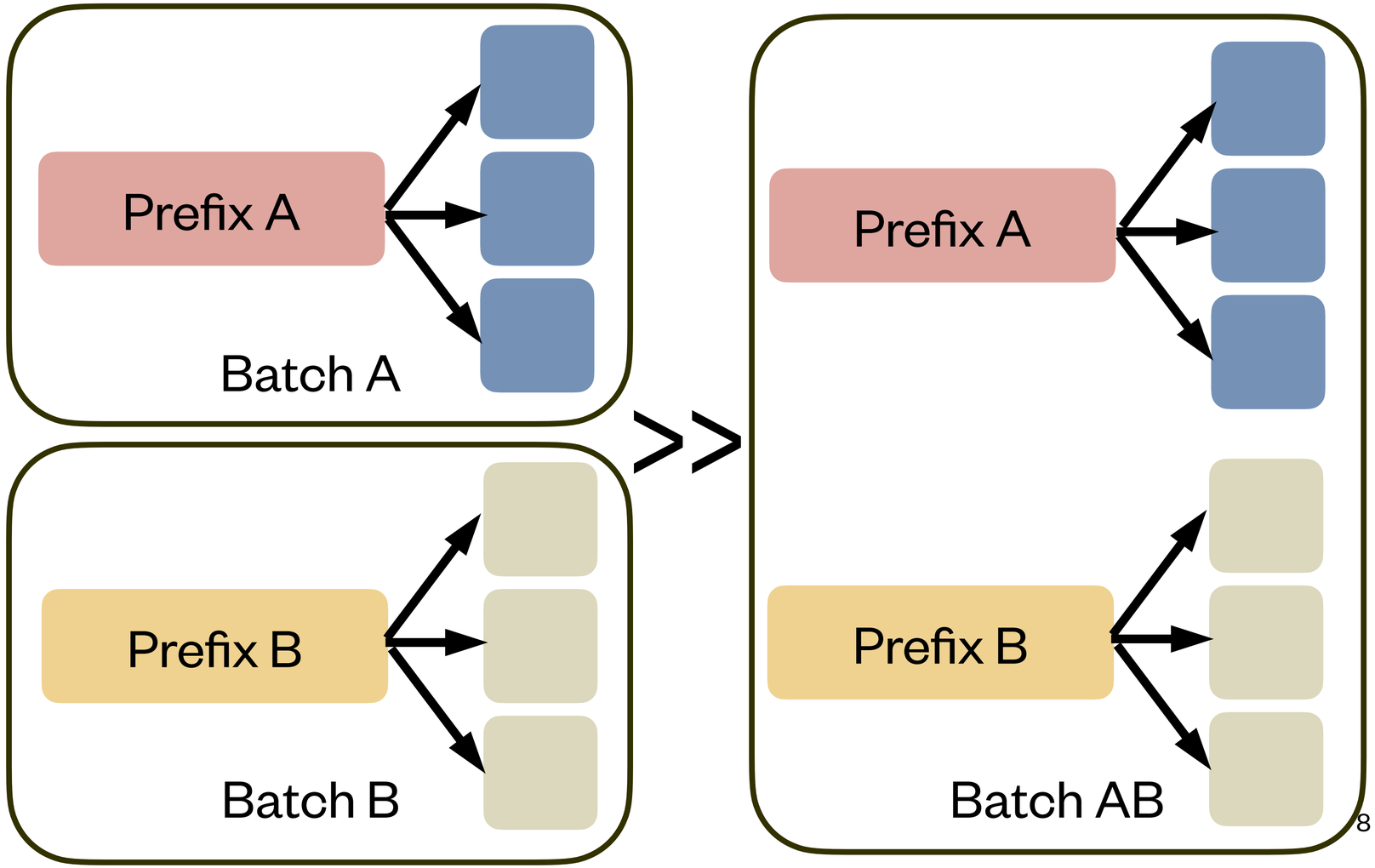
(B1 can be evicted once R_4 is done)

Experimental Setup

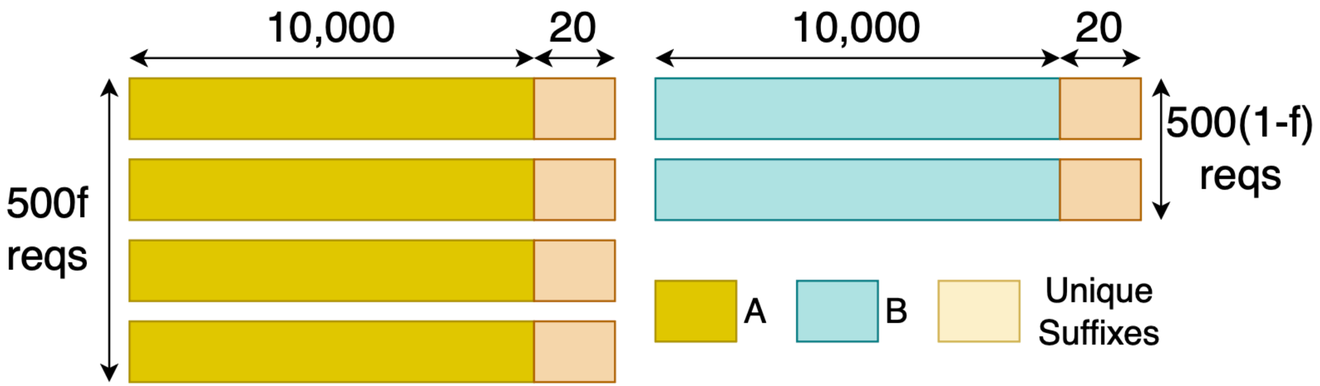
- Llama-3-8B
- RTX 6000 ADA GPU - 48 GB GDDR6, 96 MB L2 Cache
- vLLM, default batch size = 500
- 10k token prefix takes 1.2 GB KV Cache space
- L-Eval Dataset
- FlashAttention Backend
- Warm-up ensures KV cache present in GPU memory

Key Insight 1 - Prefix Homogeneity

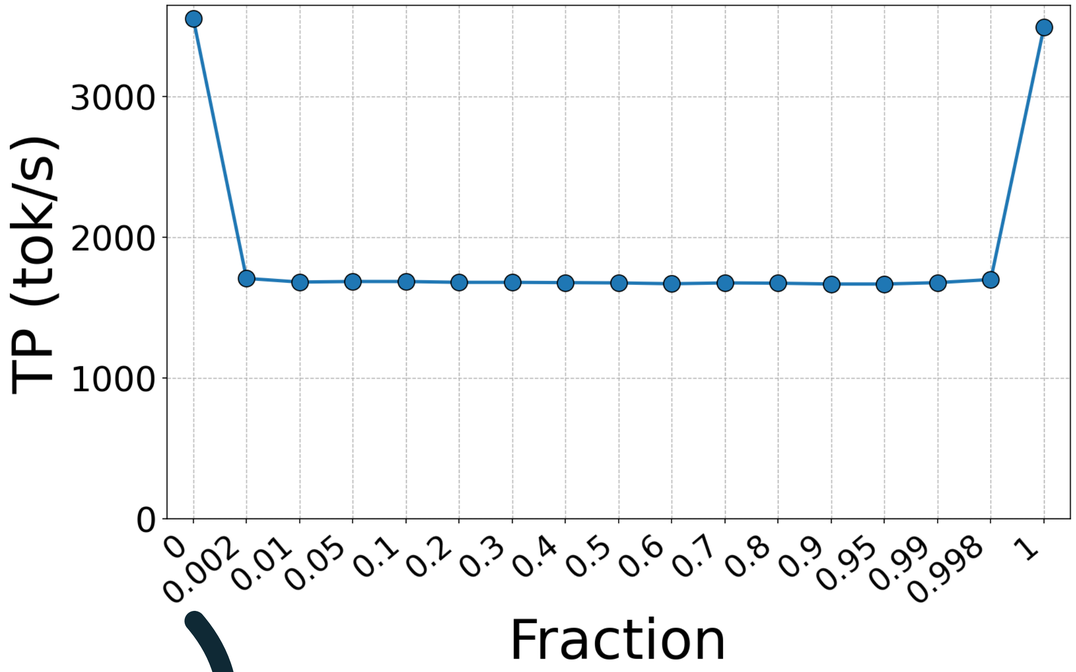
- **Prefix Homogeneity**
  The length of the prefix shared across **all** the requests of the batch
- Smaller prefix homogeneous batches outperform larger heterogeneous ones



Two prefix groups



For prefix-homogeneous batches, when all the requests of the batch share a single prefix, throughput is 2X higher

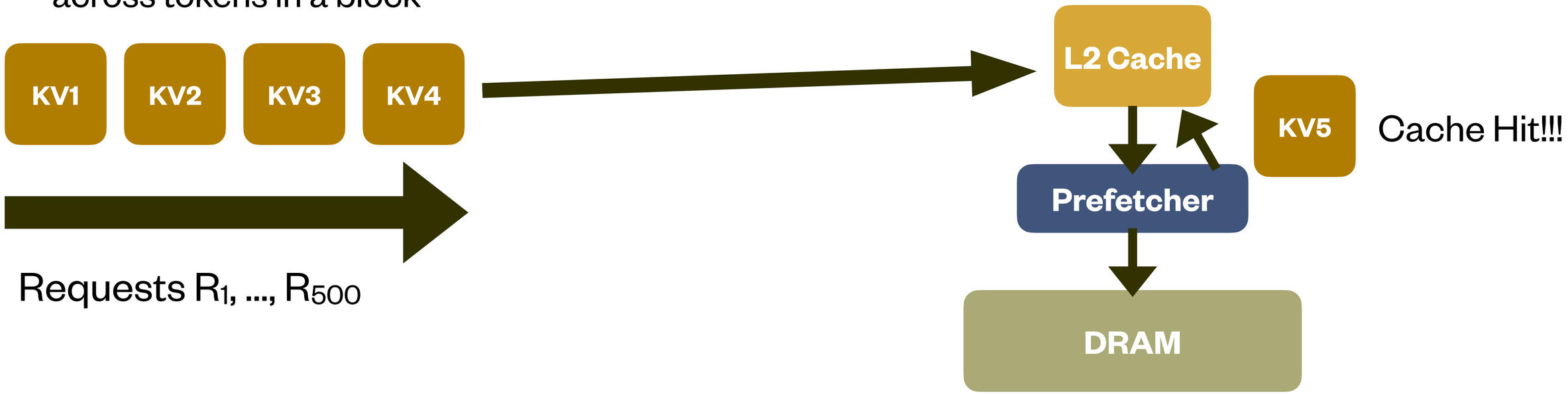


Prefix-A has one request
Prefix-B has 499 requests

Why?

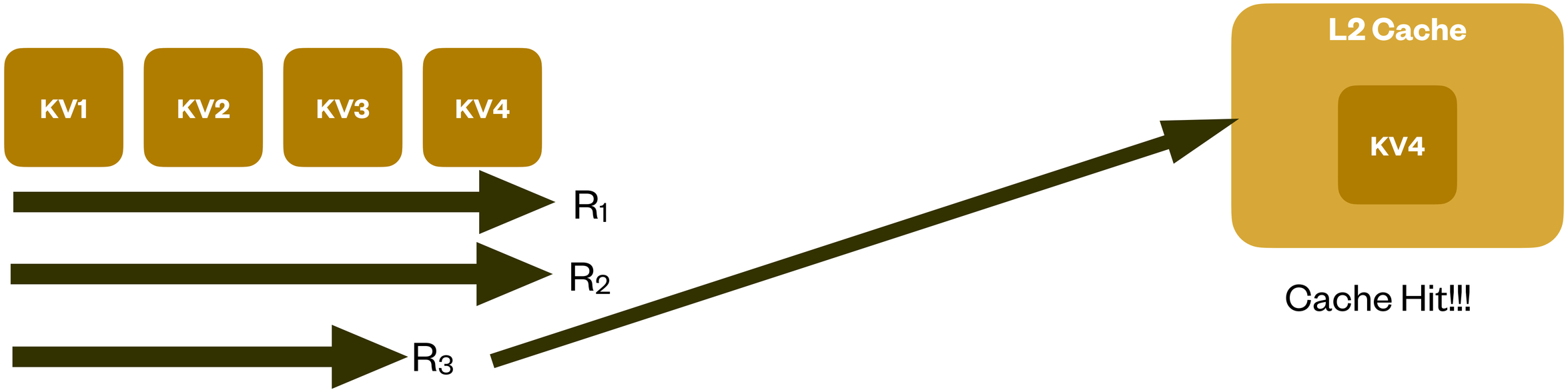
Spatial Locality

KV Cache contiguous
across tokens in a block



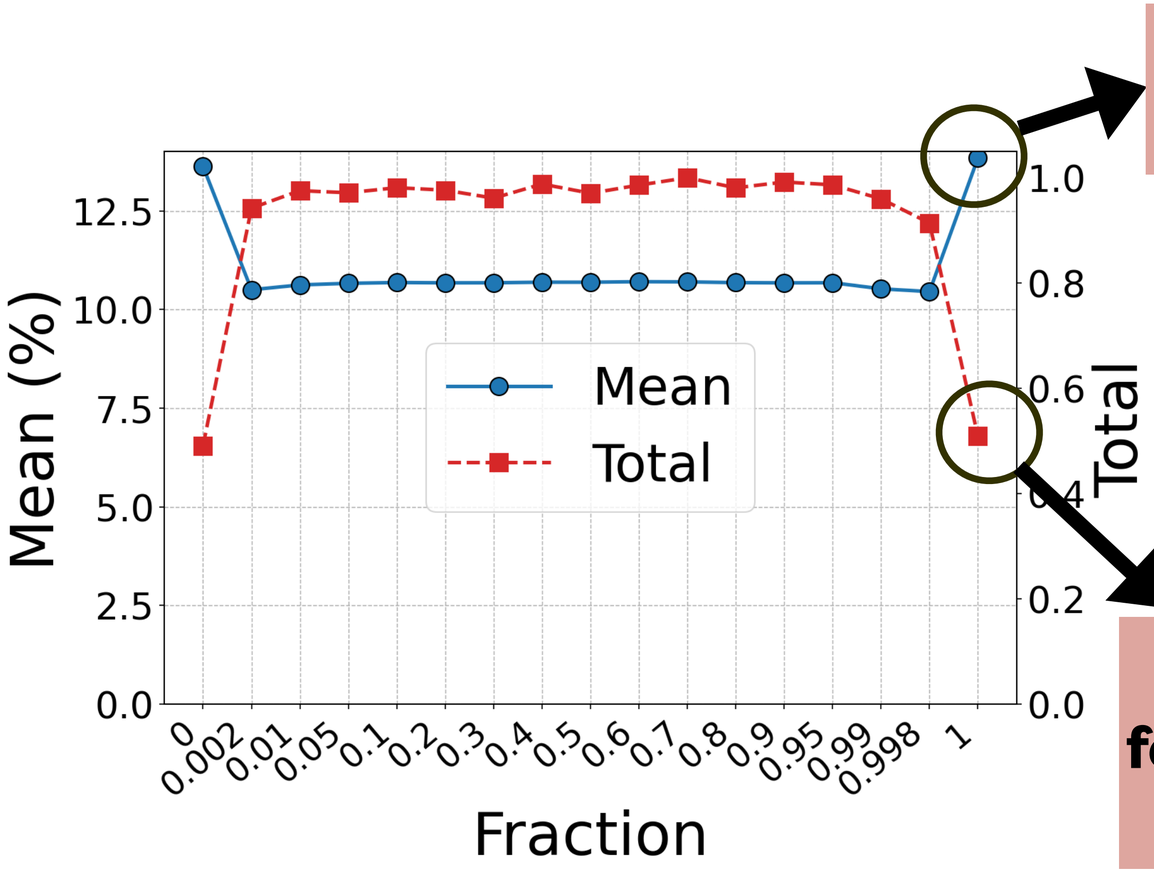
Sequential access to contiguous KV tensors enables the hardware prefetcher to fetch upcoming cache lines from DRAM into L2

Temporal Locality



Cache lines fetched for one request are reused by neighbouring requests before eviction

Two prefix groups - DRAM BW



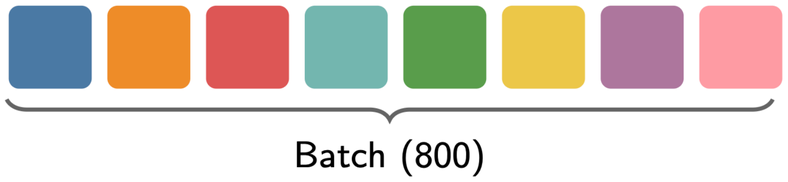
Mean DRAM BW

Total data fetched from DRAM

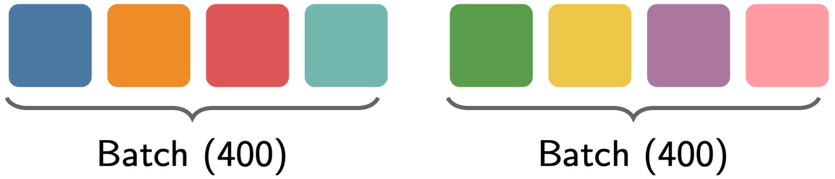
- Greater spatial locality leads to better hardware prefetching and higher BW utilization
- Temporal locality reduces the total data fetched from DRAM

Using DCGMI

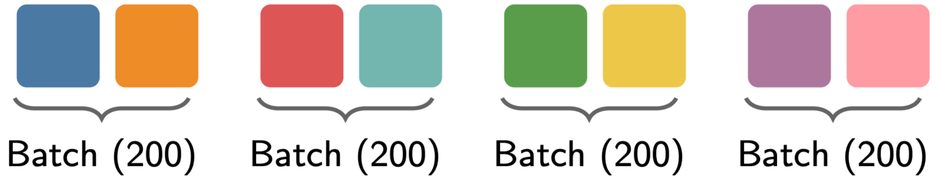
Batch Size vs. Prefix Homogeneity



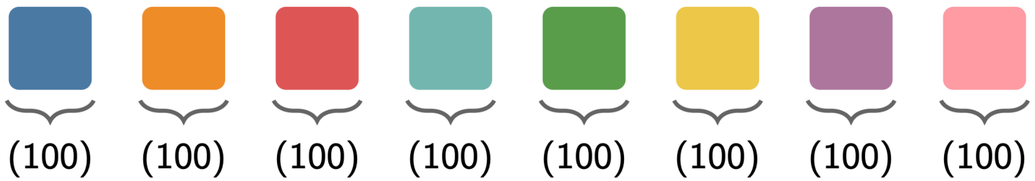
Batch Size = 800
1 batch × 8 prefix groups



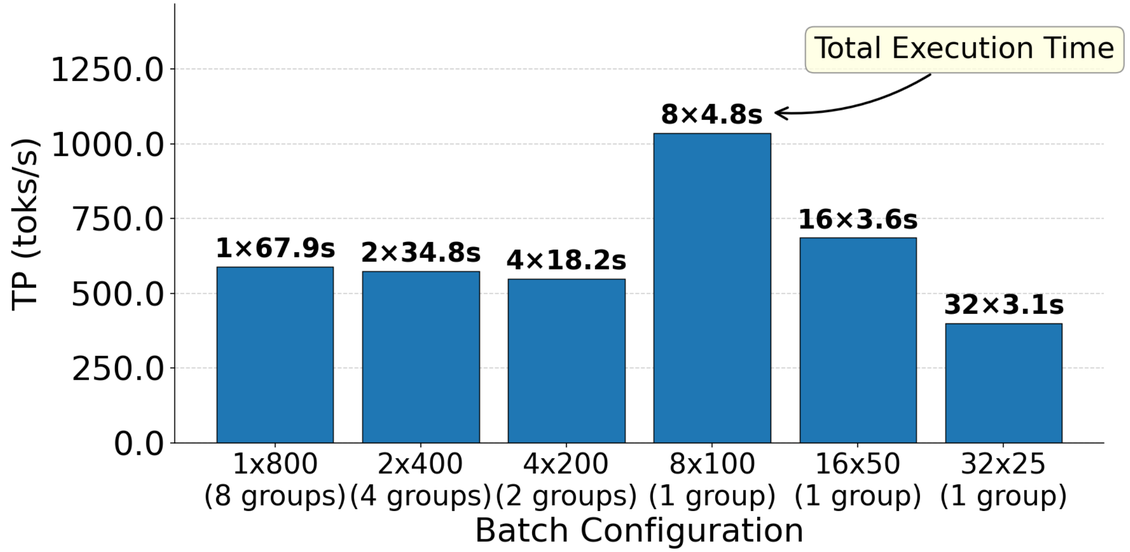
Batch Size = 400
2 batches × 4 prefix groups



Batch Size = 200
4 batches × 2 prefix groups

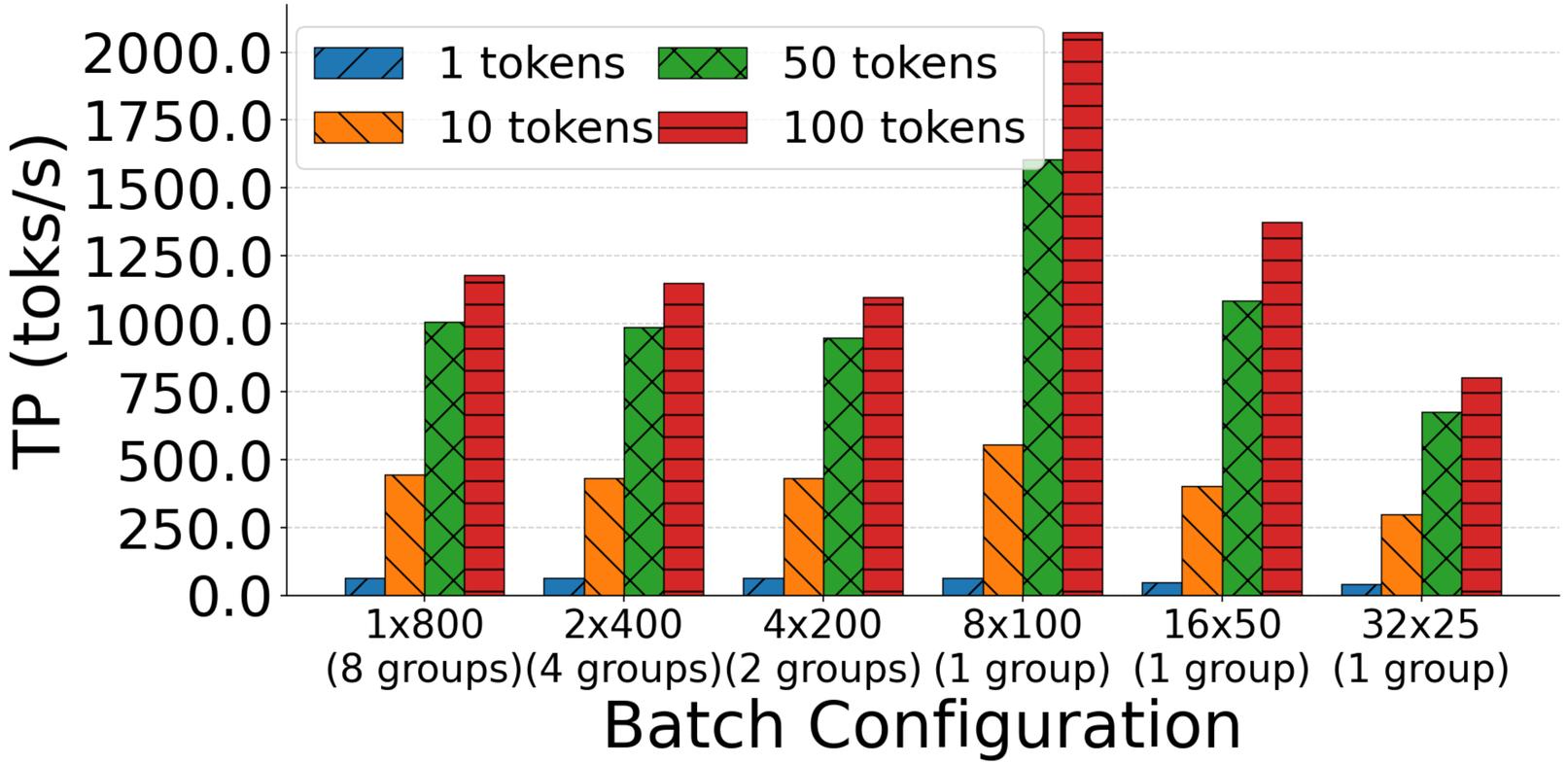


Batch Size = 100
8 batches × 1 prefix group



Small homogeneous batches perform better than larger heterogeneous ones

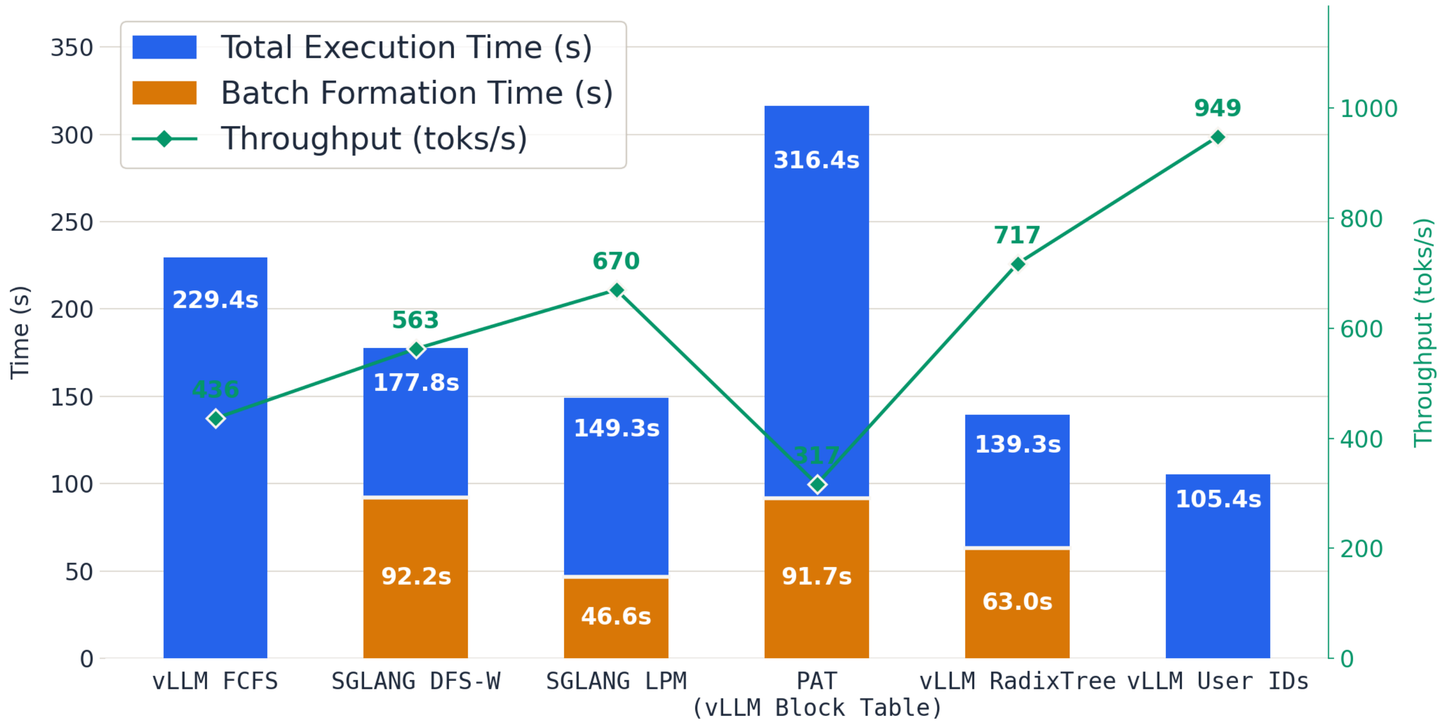
Batch Size vs. Prefix Homogeneity



- Larger decode lengths result in repeated KV cache sweeps
- Even 16X50 performs better than 1X800 for larger decode lengths

Key Insight 2 - Overhead of Dynamic Prefix Detection

- Existing schedulers use radix trees
- SGLang: DFS-W and LPM
- PAT: vLLM block table
- Overhead ~ GPU execution time



DFS-W = DFS-based tree traversal of radix tree

LPM = Longest matching cached prefixes

Key Insights and Ideas

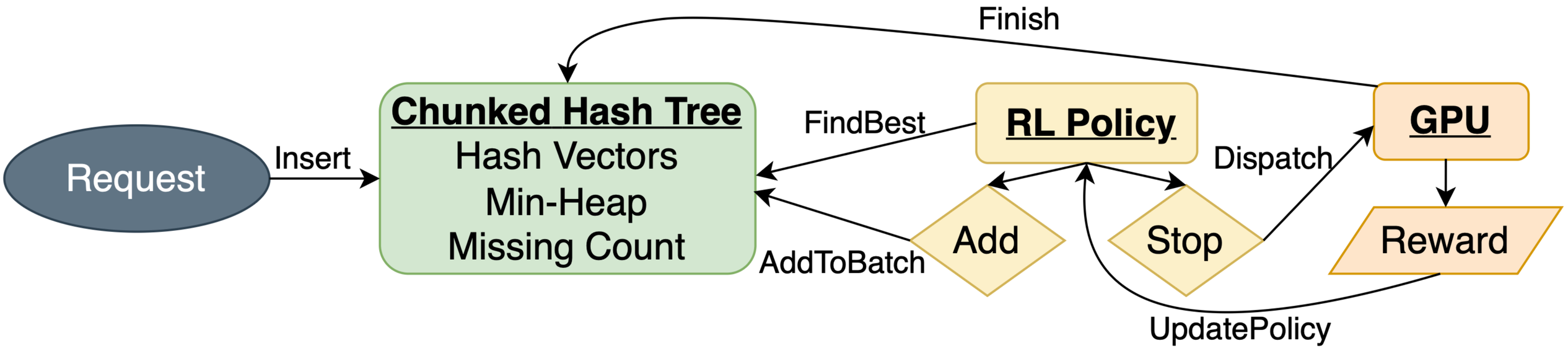
Key Insight 1: Increasing batch size utilizes GPU resources better, but might reduce prefix homogeneity

Key Insight 2: Prefix-aware schedulers incur large CPU overheads, often accounting for 50-90% of total latency

Key Idea 1: Learn the stopping decision via RL

Key Idea 2: Shared prefix detection via Chunked Hash Tree

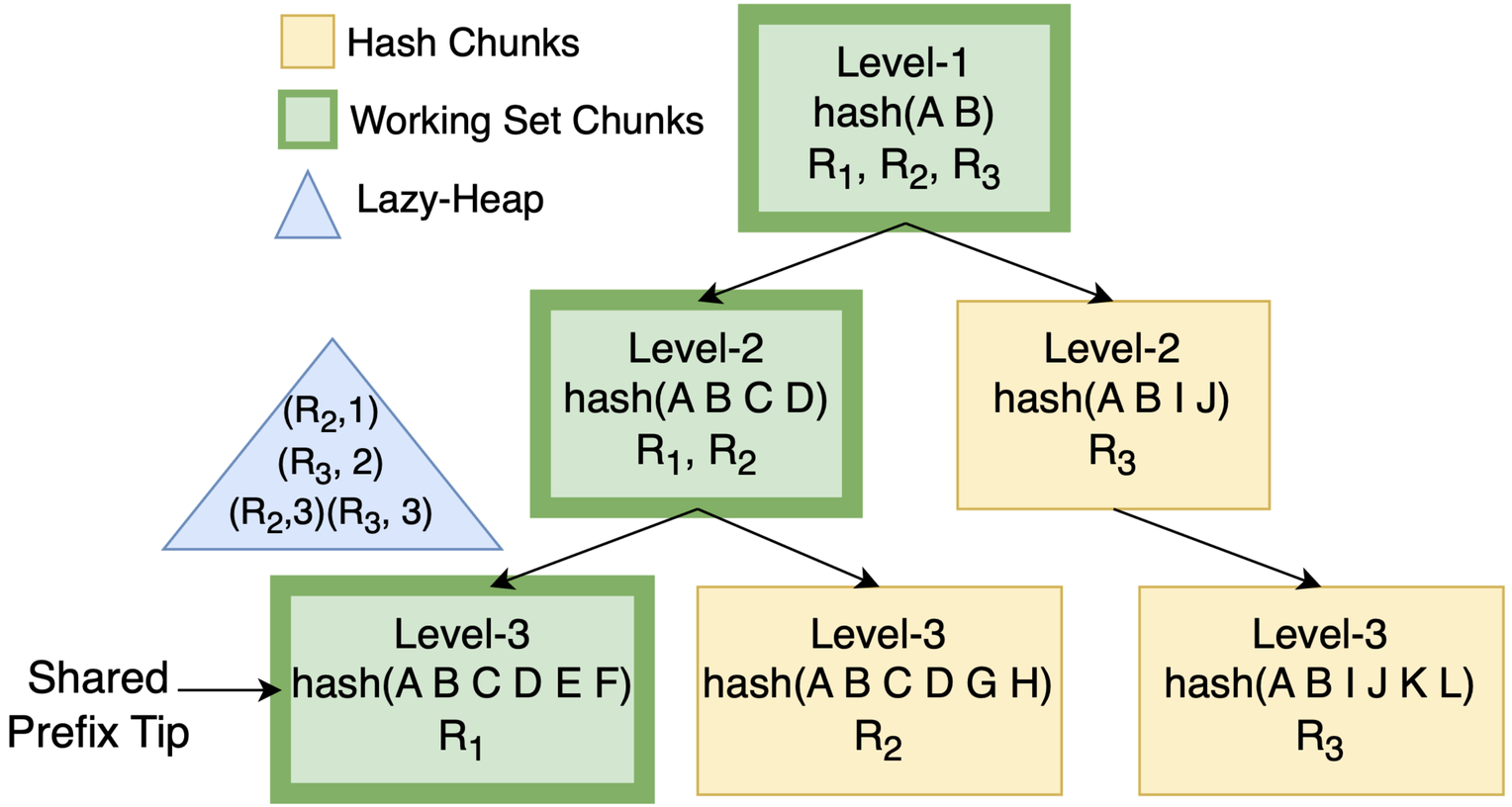
Key Ideas of the Design



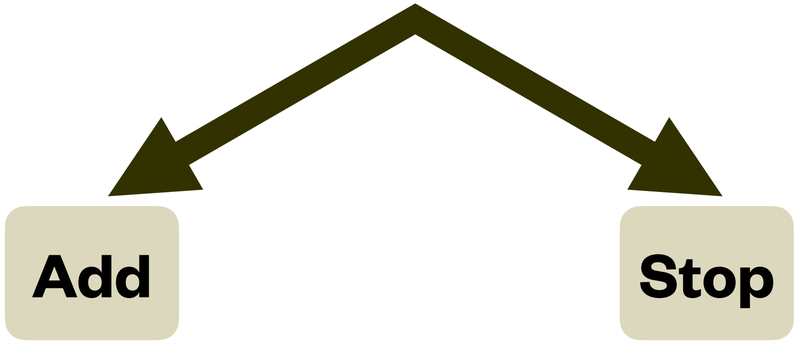
- Chunked Hash Tree - efficient data structure for shared prefix detection
- Reinforcement Learning - to stop batch formation at the right moment
- **Feather** = Chunked Hash Tree + RL

Chunked Hash Tree Overview

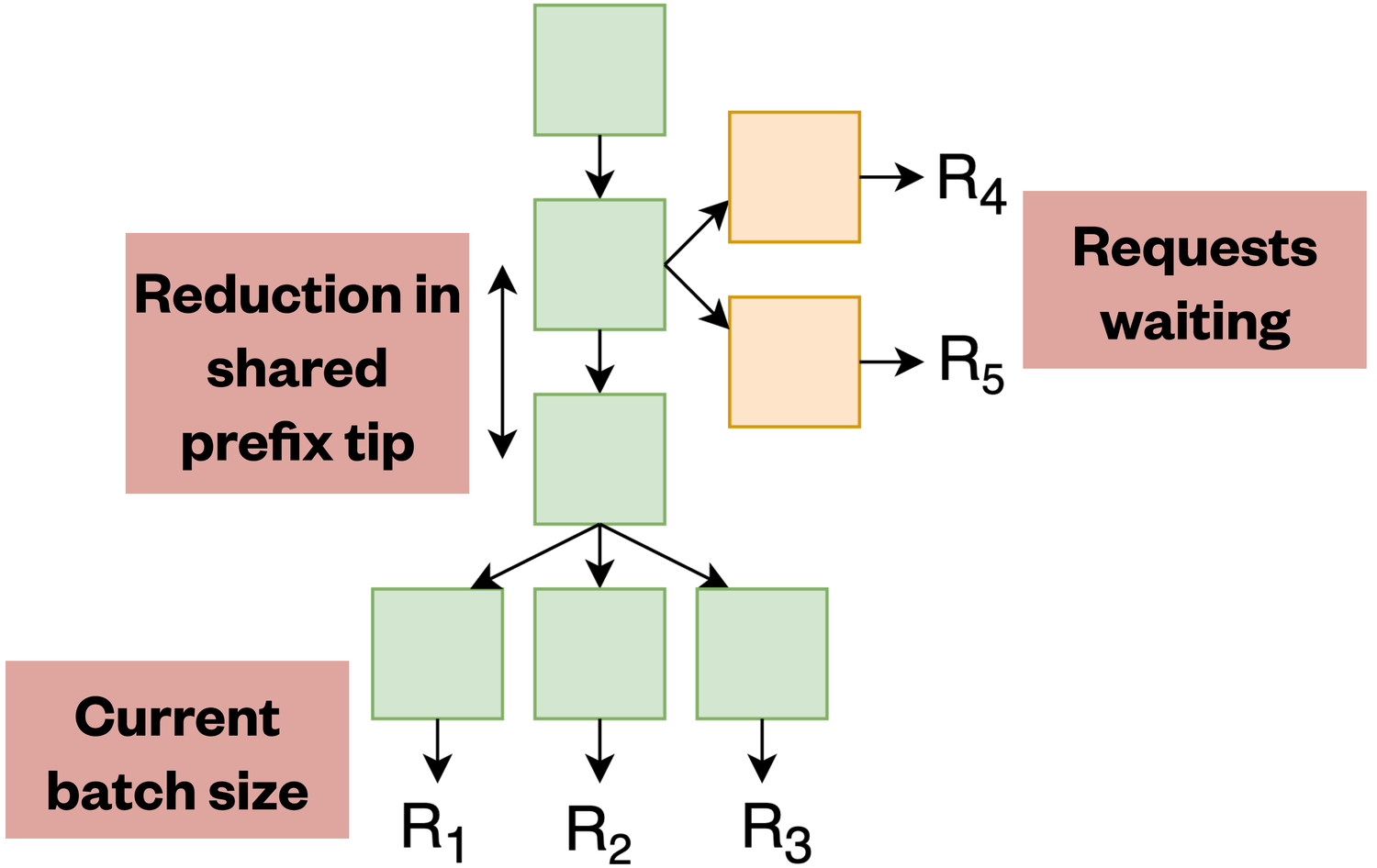
- Requests represented by cumulative prefix hashes
- Working set of chunks of active requests
- Lazy heap to store cost w.r.t working set
- Shared prefix tip denotes the point till everyone shares



Reinforcement Learning

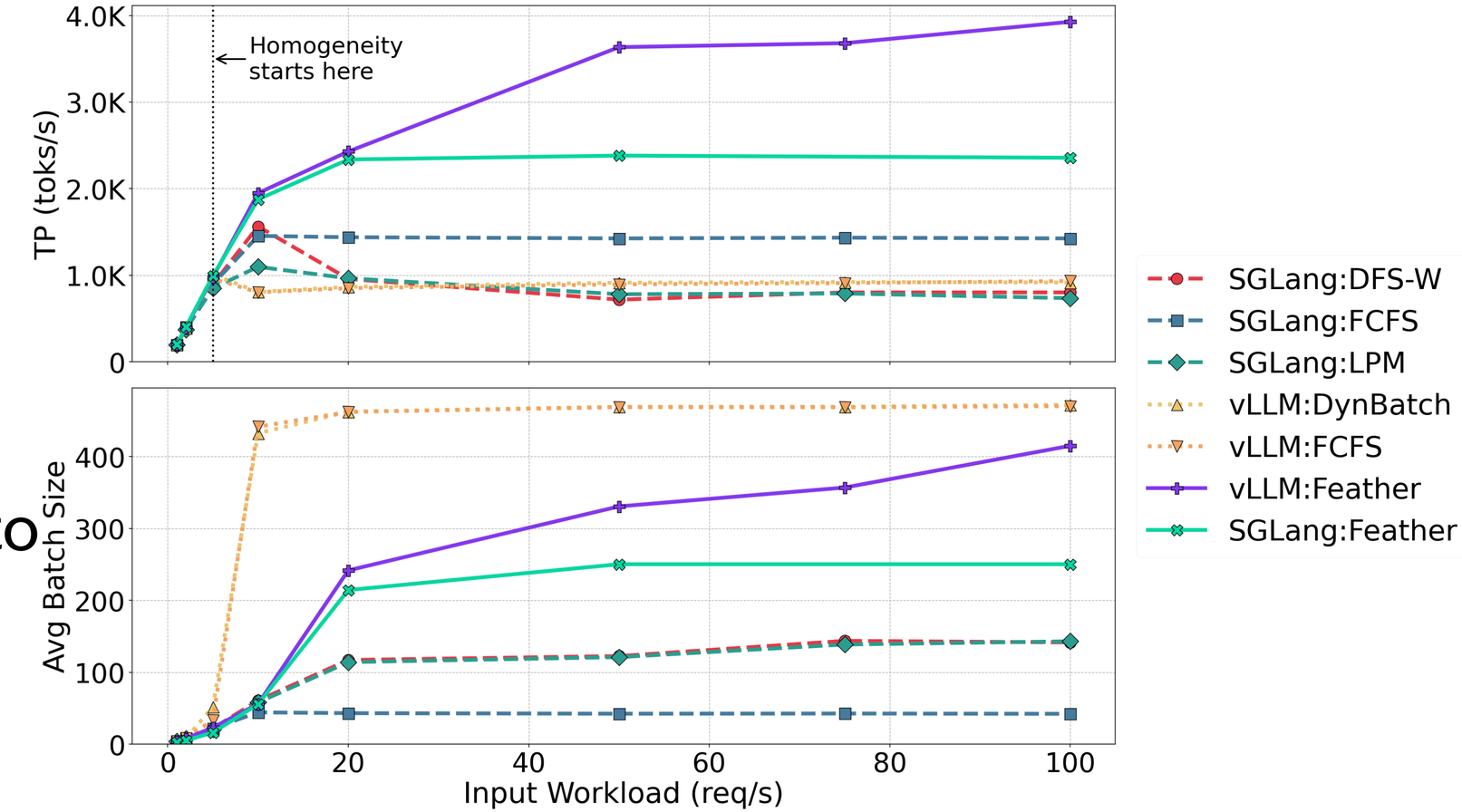


Reward = Decode throughput from the GPU



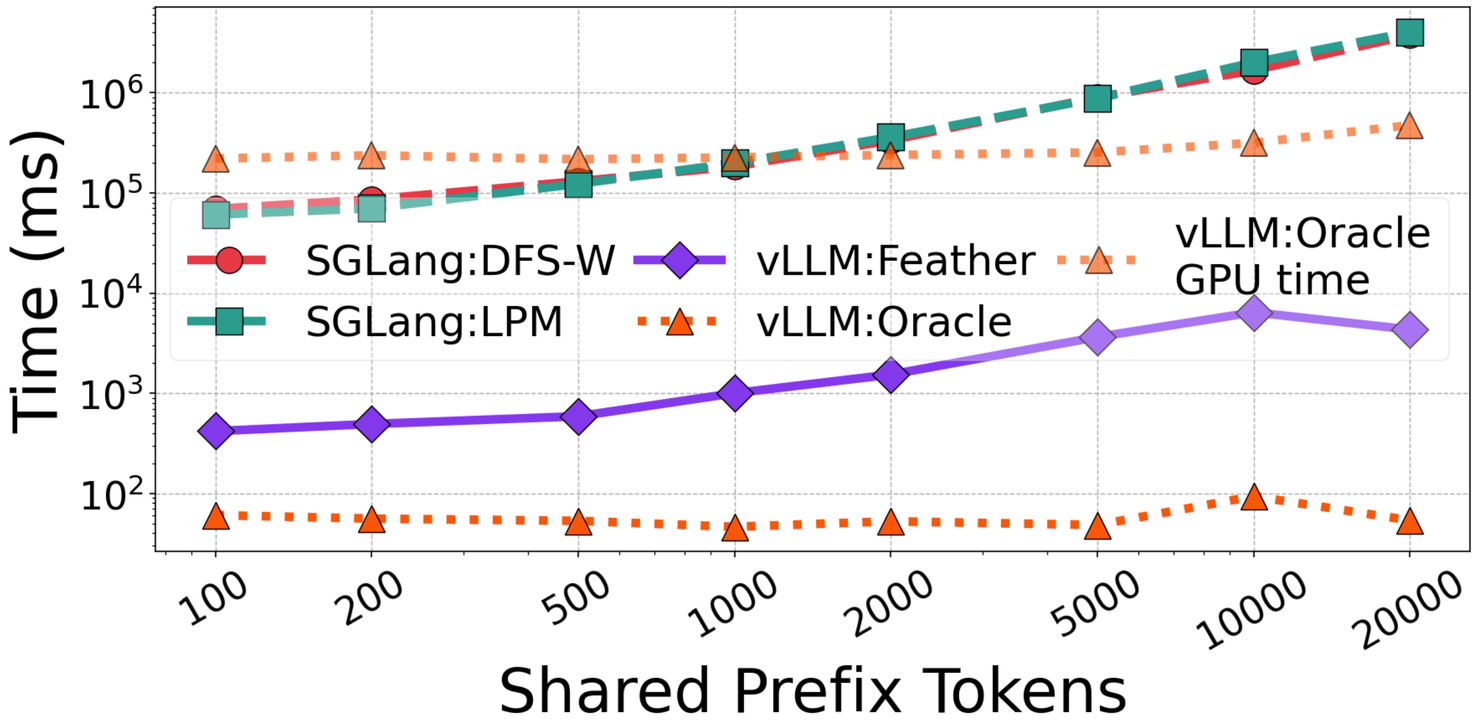
Evaluation across Input Request Rate

- 5 prefix groups, each of length 5000 tokens, generating 200 decode tokens
- Feather achieves 4X throughput as compared to vLLM despite a smaller batch size



CPU Scheduler Compute Overhead

- Oracle = user provides prefix group IDs, negligible compute
- SGLang's LPM and DFS-W incur 1000X higher overhead than Feather
- Feather less than 1% of GPU execution time



More Results

More Takeaways

- For prefix-homogeneous batches, throughput increases with the length of the shared prefix
- A very high number of prefix groups leads to GPU memory evictions
- Performance independent of number of radix tree levels
- Increasing a batch size beyond a point does not necessarily improve throughput

Feather Results

- Feather achieves 22X higher throughput than vLLM FCFS for LongChat-13B model
- Feather outperforms PAT, a prefix-aware attention kernel
- Feather leads to higher mean DRAM BW, but lower total number of memory accesses
- Chunk size of 1 is comparable in throughput to chunk size of 500

Conclusions and Future Steps

Conclusions

- Prefix homogeneity as a scheduling signal, as important as batch size
- CHT for fast shared prefix detection
- RL to stop the batch at the right moment
- Will push the changes to vLLM and SGLang repositories

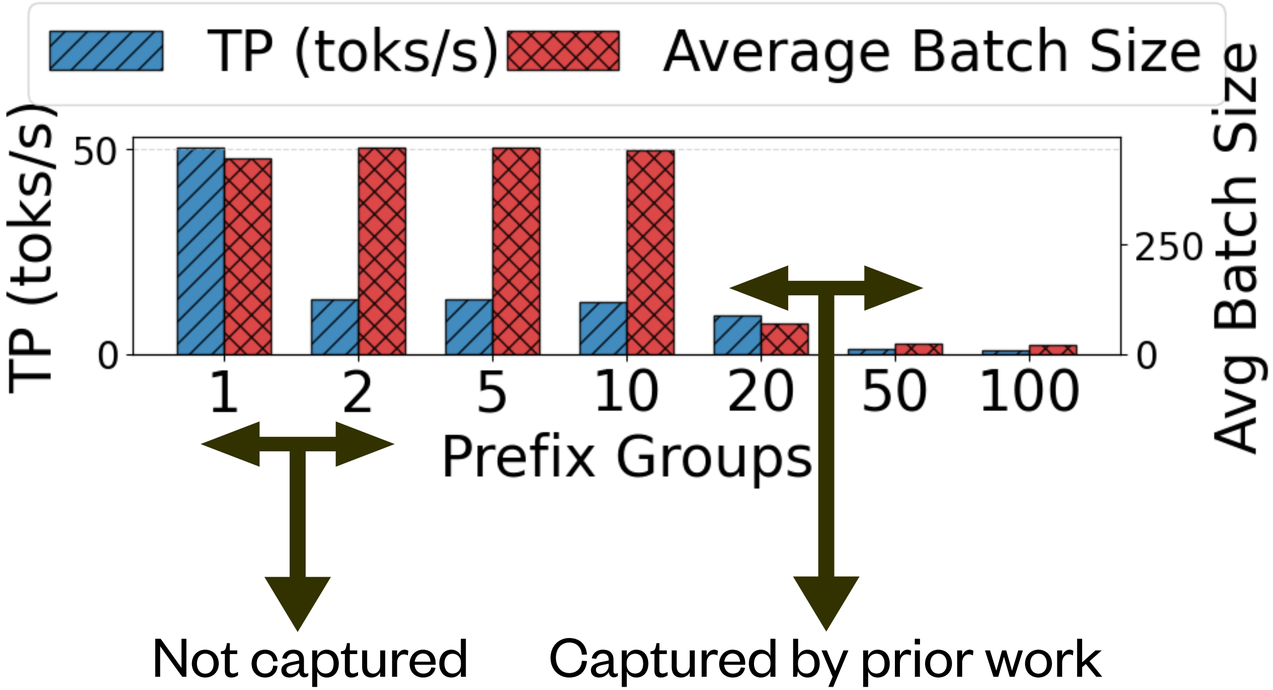
Future Steps

- Integration with prefix-aware attention kernels
- Extend to distributed and multi-GPU settings
- Better RL policies

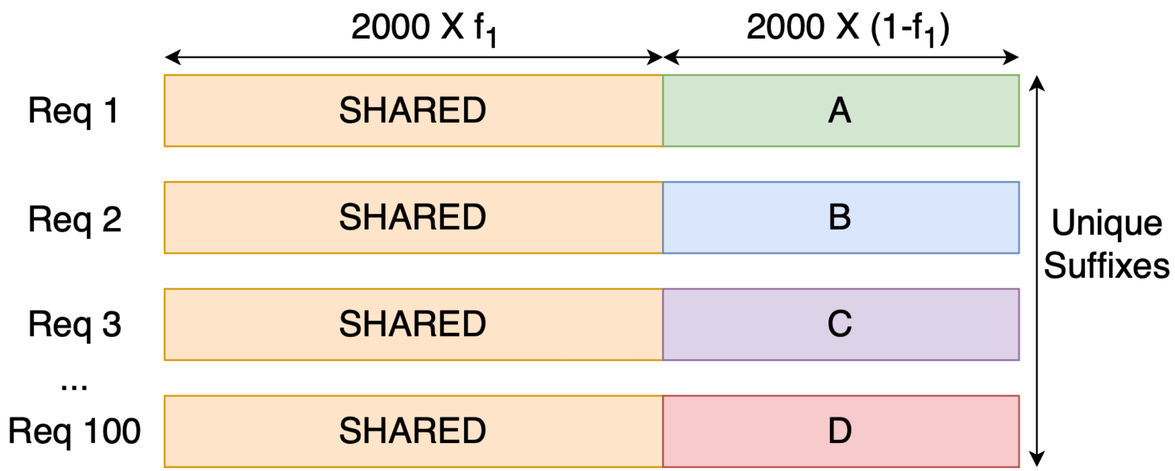
Thank You

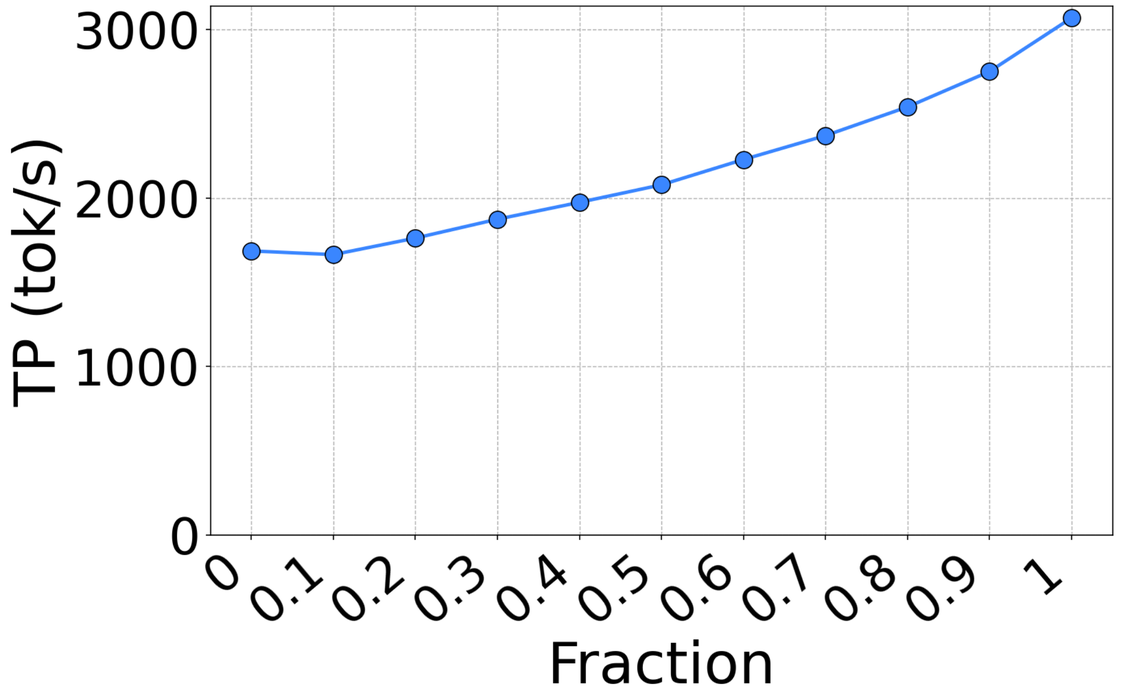
Number of Prefix Groups

- Drop from 1 to 2, because of heterogeneity
- Constant from 2 to 20 -> less impact of additional heterogeneity
- From 20 to 100, drastic drop because of KV cache evictions



Fraction of Prefix Shared

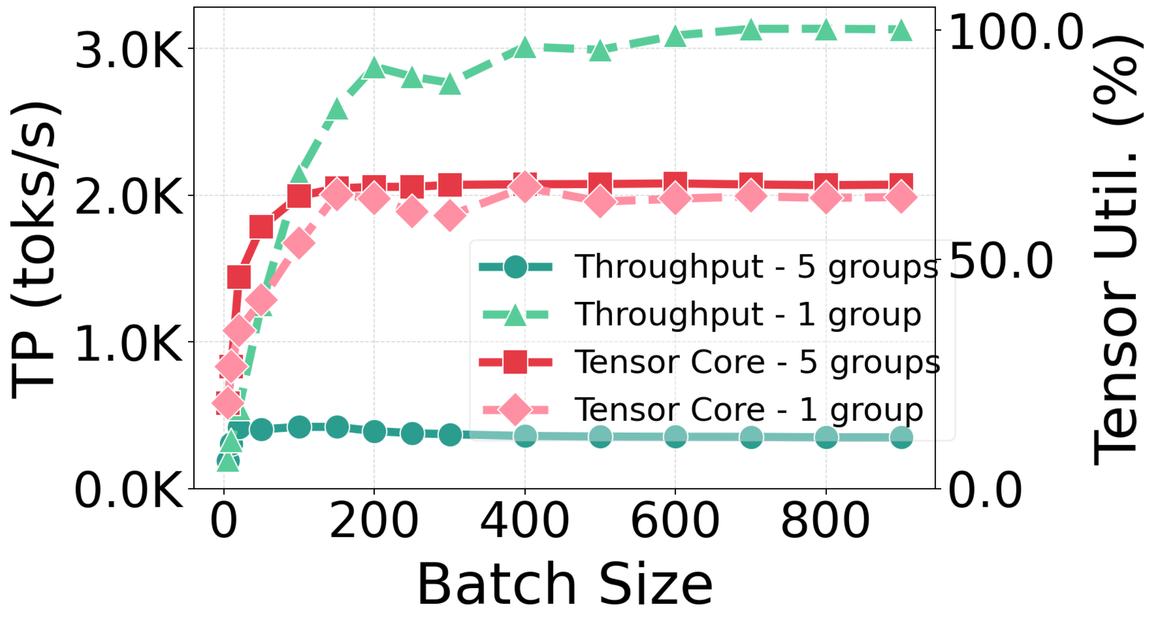




Throughput increases smoothly with the length of the shared prefix, because of higher locality benefits

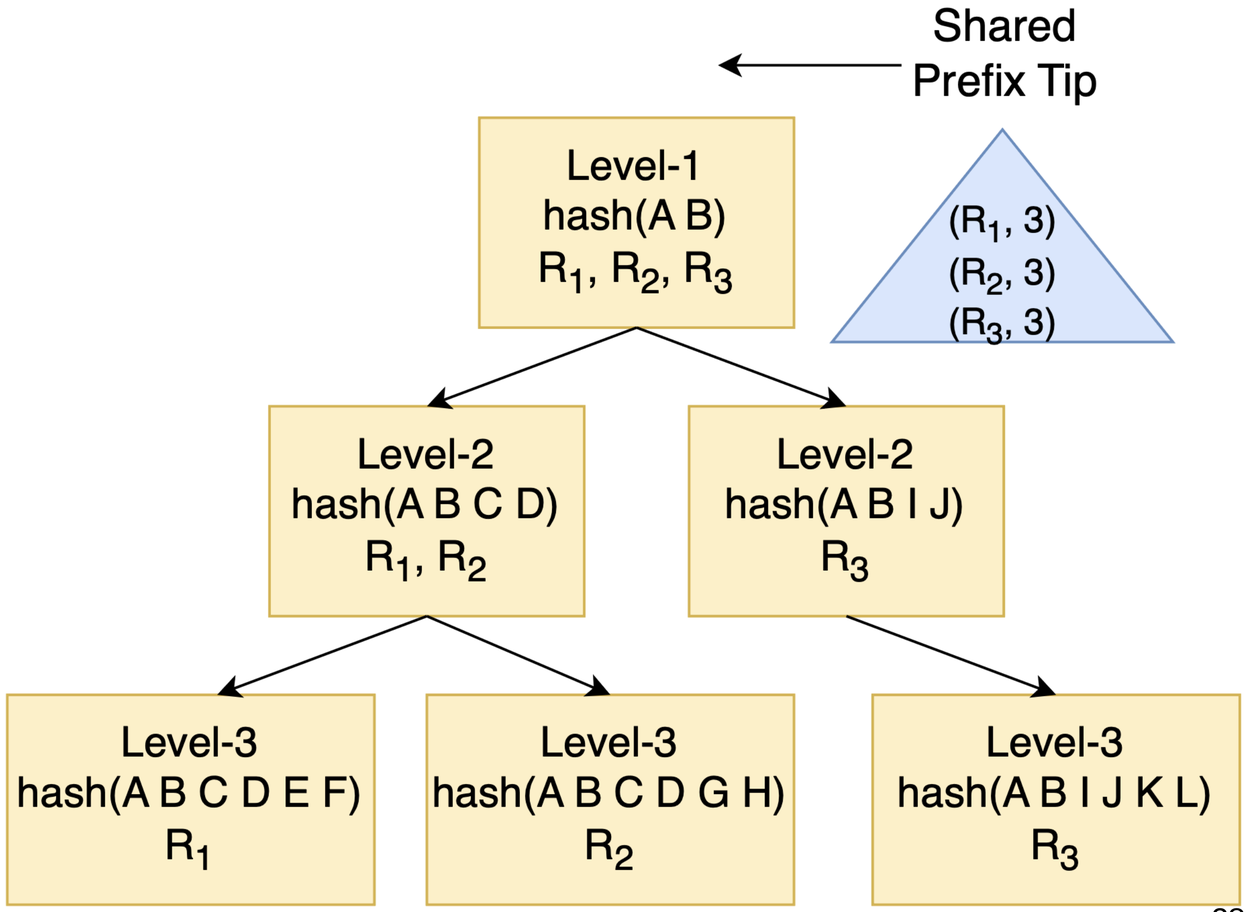
Across Batch Size

- Throughput plateaus before maximum batch size
- Saturation throughput higher for homogeneous batches
- Batch size at which homogeneous saturates is higher
- Compute utilization roughly the same



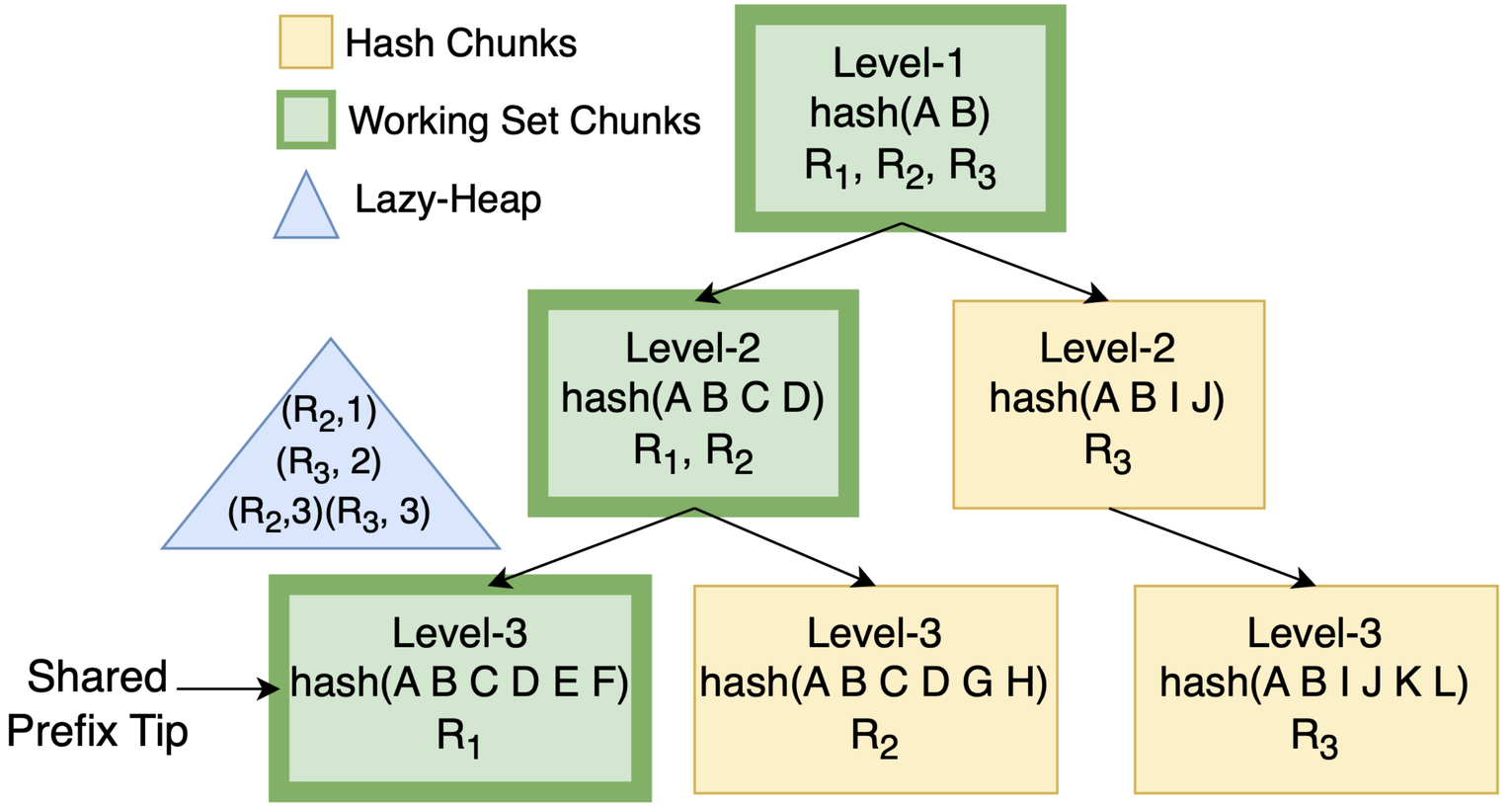
Chunked Hash Tree - Insertion

- Waiting requests inserted
- Token sequence divided into chunks
- Chunks represented by cumulative prefix hashes



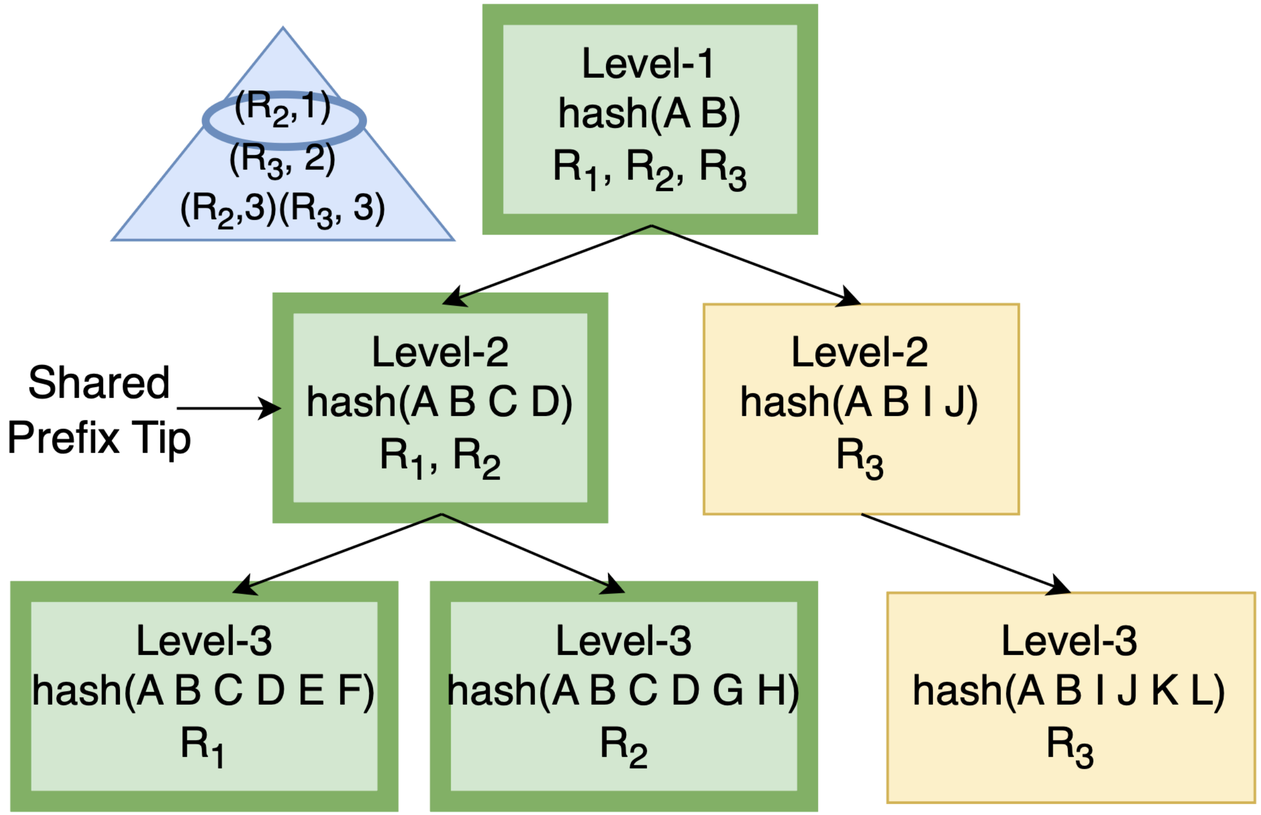
Chunked Hash Tree - AddToBatch

- R_1 added to the running batch - its chunks added to the working set
- Costs in the lazy heap represent distance from the working set
- Shared prefix tip represents the point till which the entire batch shares



Chunked Hash Tree - FindBest

- Pick the request with the minimum cost with respect to the working set
- Activating R_2 leads to reduction in shared prefix tip



Chunked Hash Tree - Finish

- R_1 finished, and removed from the tree
- Shared prefix tip again extended