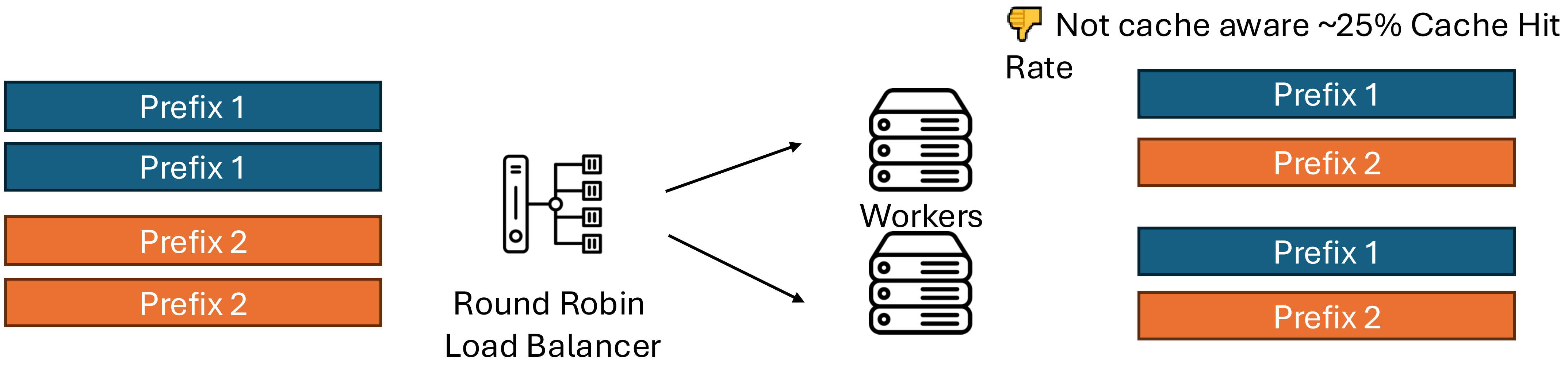
Cache-Aware Load Balancer in SGLang

Byron Hsu @ LMSYS Org

The Problems

- On a single inference engine, we can **schedule requests based on prefix cache hit rate**, so the requests tend to have high cache hit rate
- However, when scaling to multi engines under **multi-node or data parallelism settings**, the common load balancers (e.g. nginx) are not aware of prefix cache, so we lose much performance benefits.



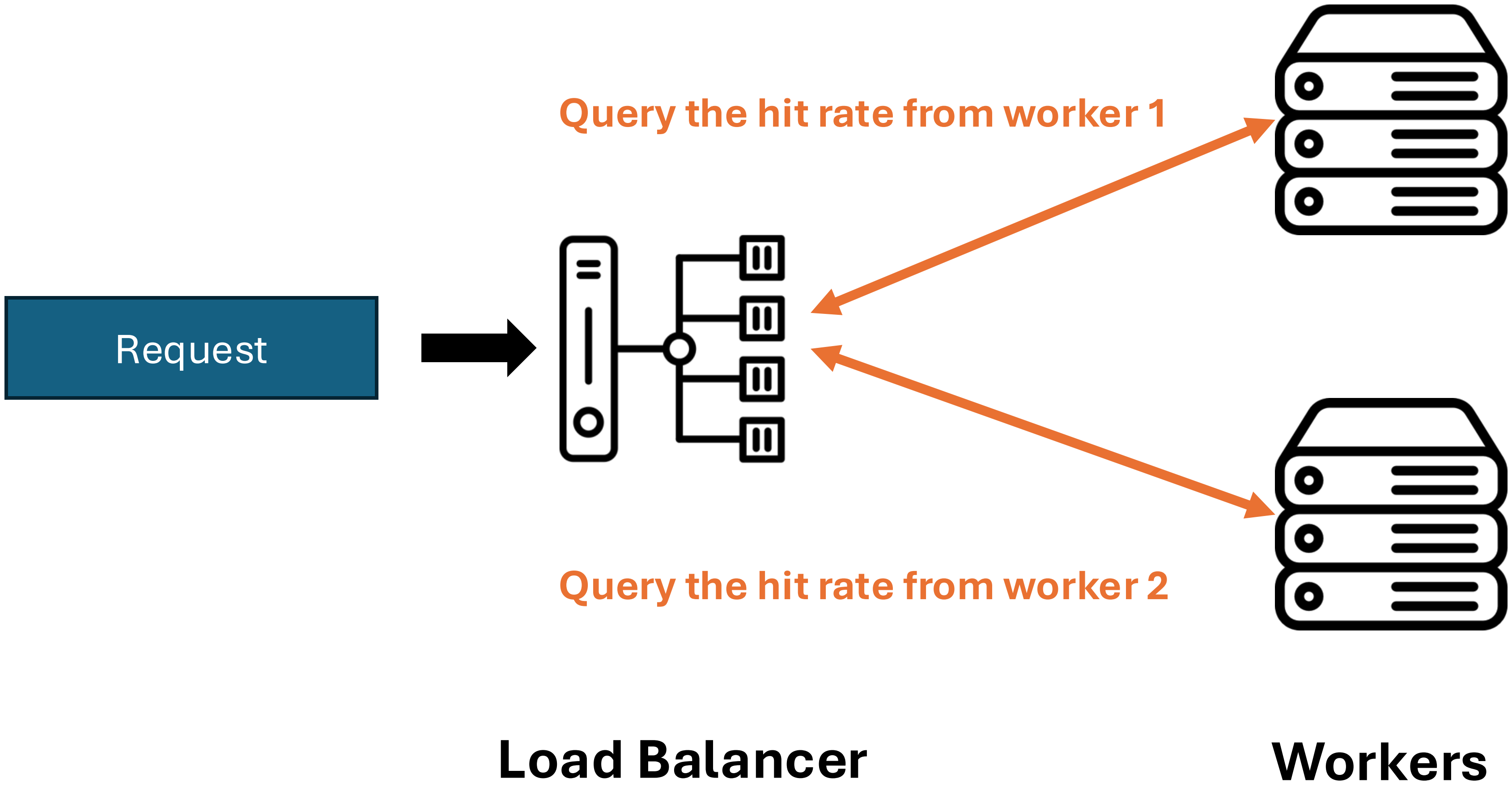
The Questions

- How can we design a load balancer to be **cache-aware** and **load balanced**?
- How can we ensure the load balancer is **fault tolerant** and **dynamically scalable**?

The Requirements

- **Cache Aware:**
  The requests are sent to the worker with higher cache hit rate
- **Load Balanced:**
  Workers are not overloaded or underloaded
- **Fault Tolerant:**
  Workers can be removed without affecting the availability
- **Dynamically Scalable:**
  Can dynamically add or remove workers

First Try: Ad-hoc Querying Cache Hit Rate



First Try: Ad-hoc Querying Cache Hit Rate

- **Latency Bottleneck:** The query of real-time cache hit rate is on the request critical path, resulting in latency bottleneck
- **Not Scalable:** The communication overhead will N times if there are N workers

Our Solution: Approximate Tree

- **Idea:** Construct **simulated radix trees** on the router (load balancer) which are very similar to the radix trees on the workers
- **Observation:** The accuracy of the simulated trees is not that important, but **making the right decision of routing is more important**

The Design of Approximate Tree

- **Tokenization Free**: We store characters in the tree so we can avoid tokenization overhead
- **Merged Radix Tree**: Instead of storing #worker trees, we merge them into one
- **Insertion:** Add the request into the tree when it is dispatched
- **Eviction:** Evict the nodes based on <u>Least Recently Used Leaves</u> policy every certain period

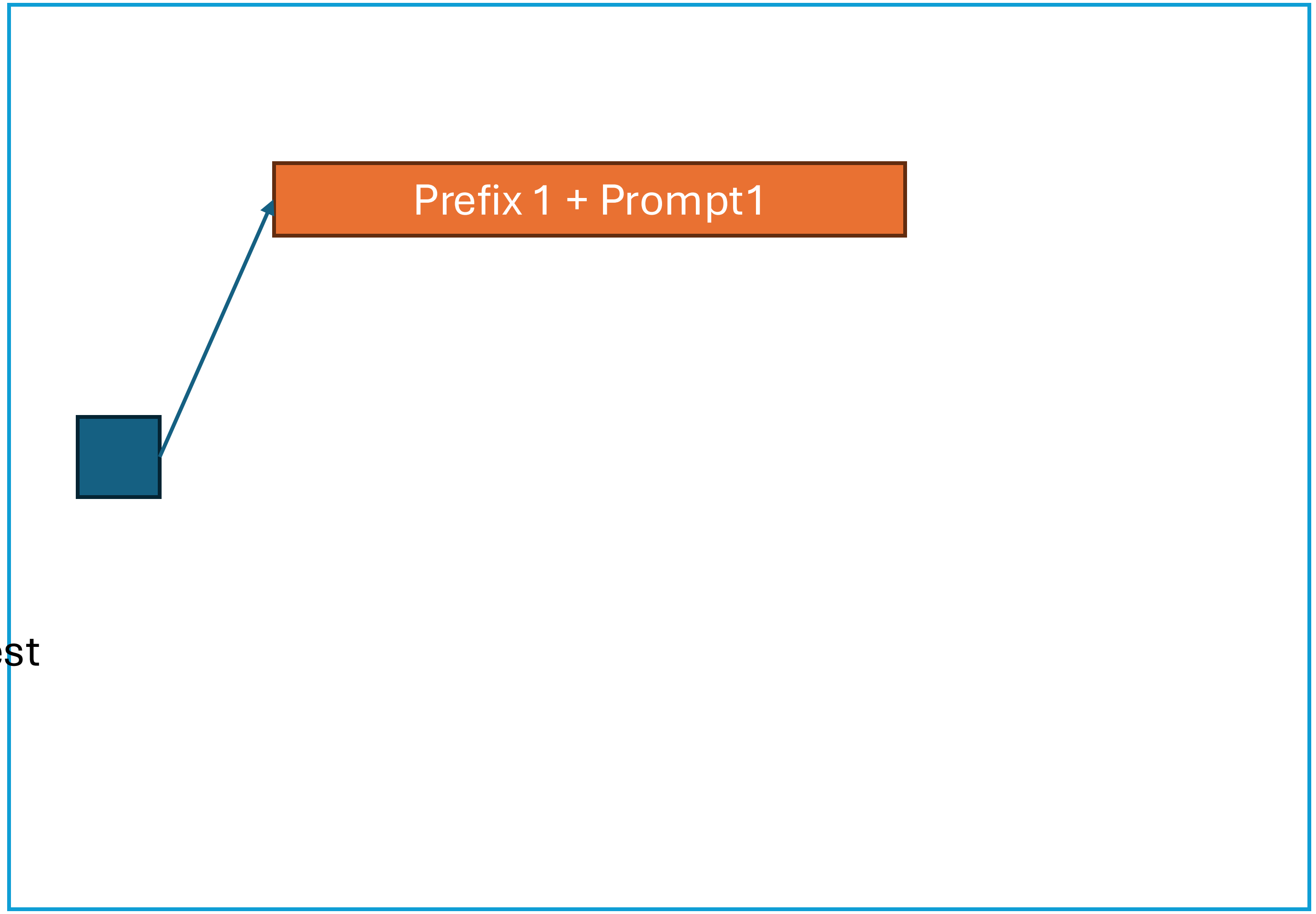
Constructing Approximate Tree

Prefix 1 + Prompt 1

User 1

Send Request

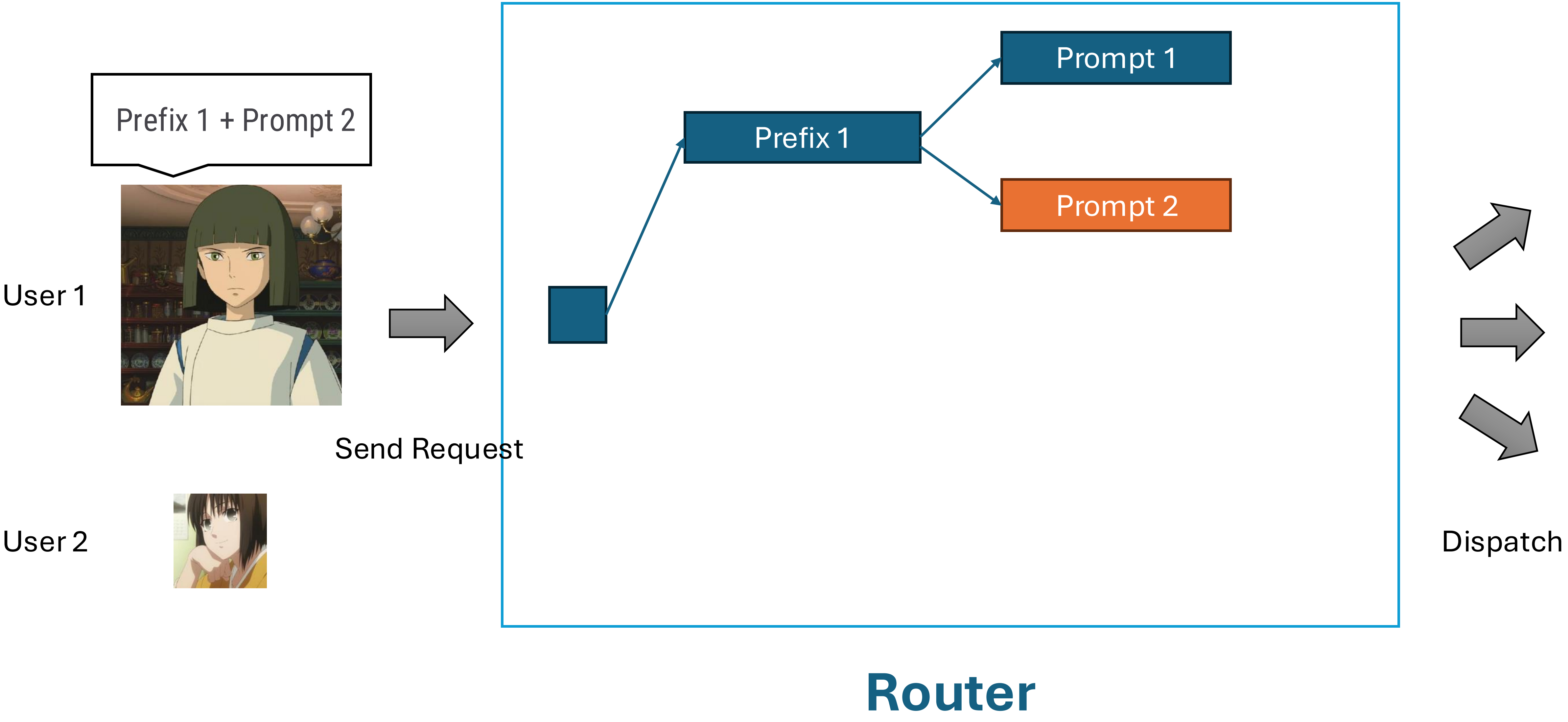
User 2



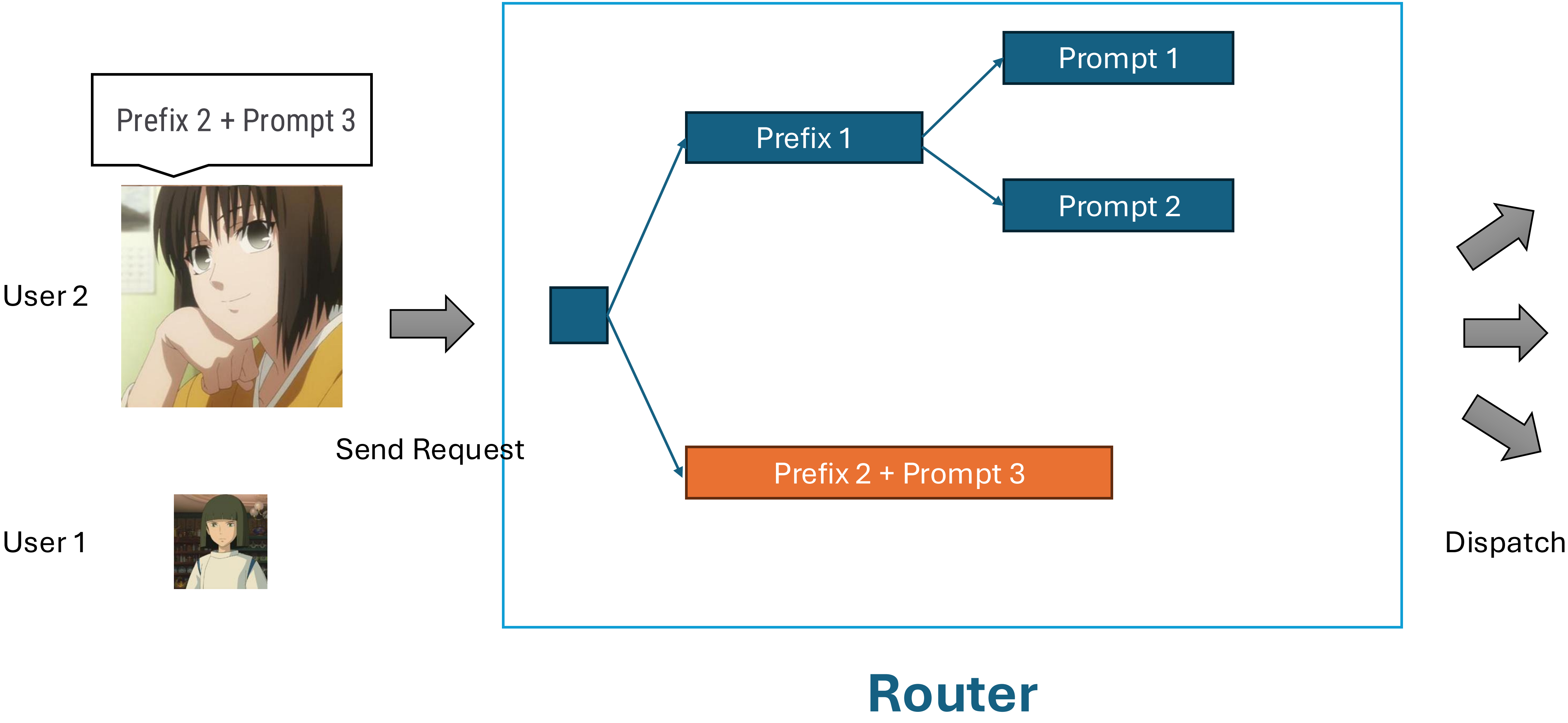
Dispatch

Router

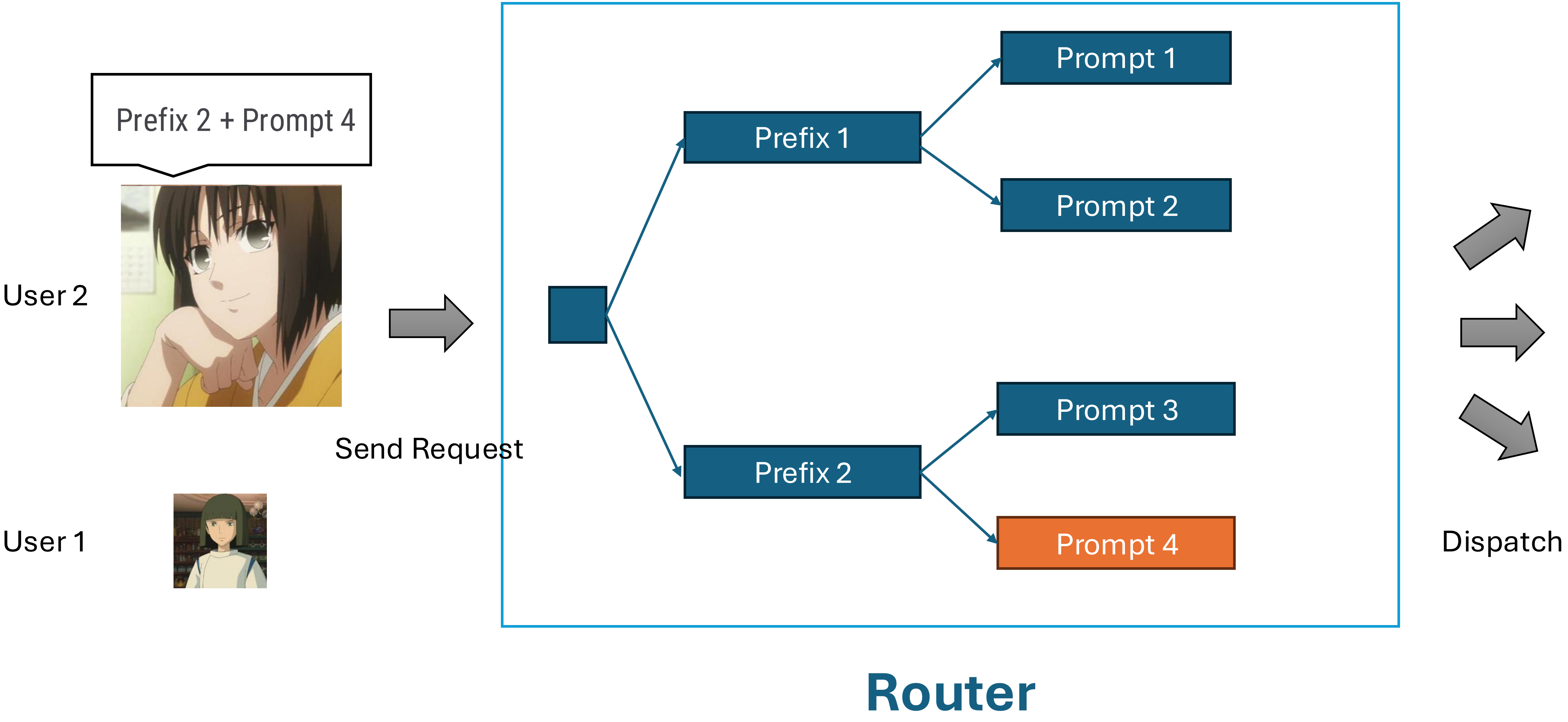
Constructing Approximate Tree



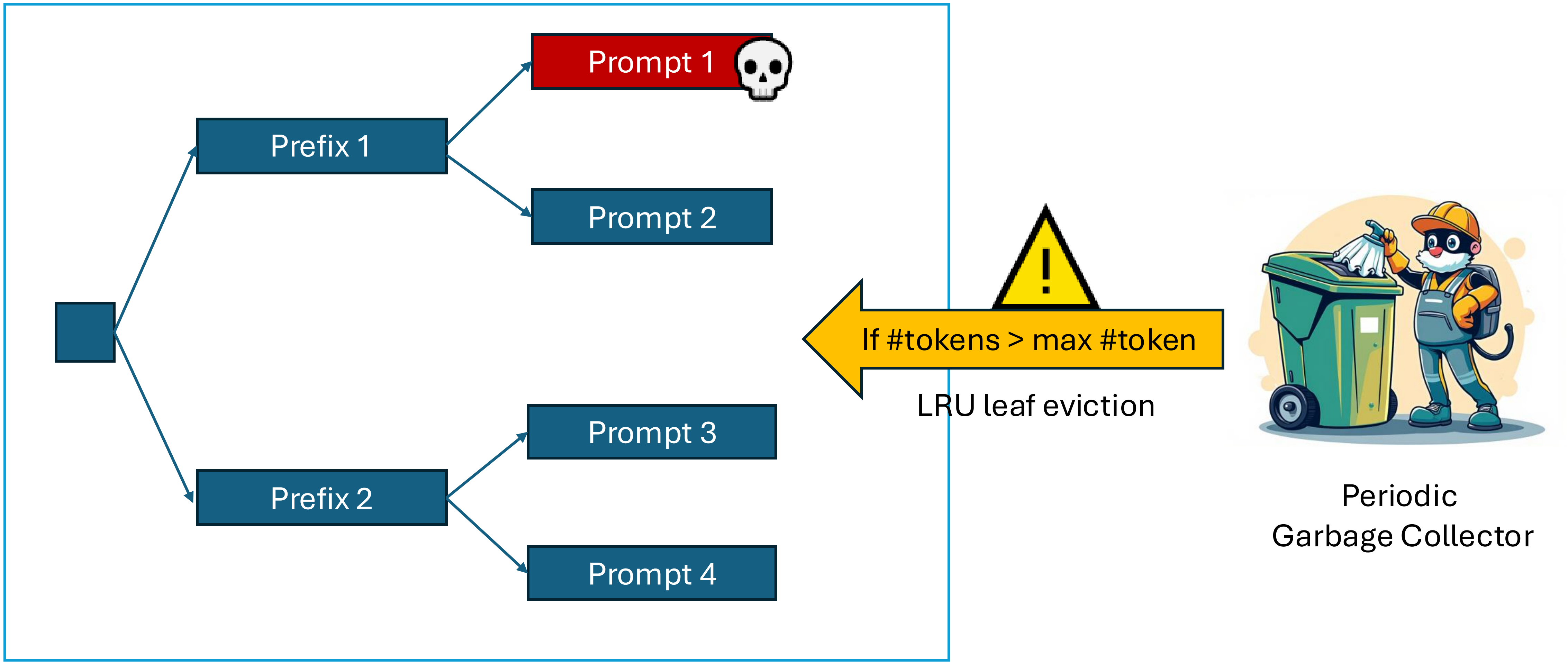
Constructing Approximate Tree



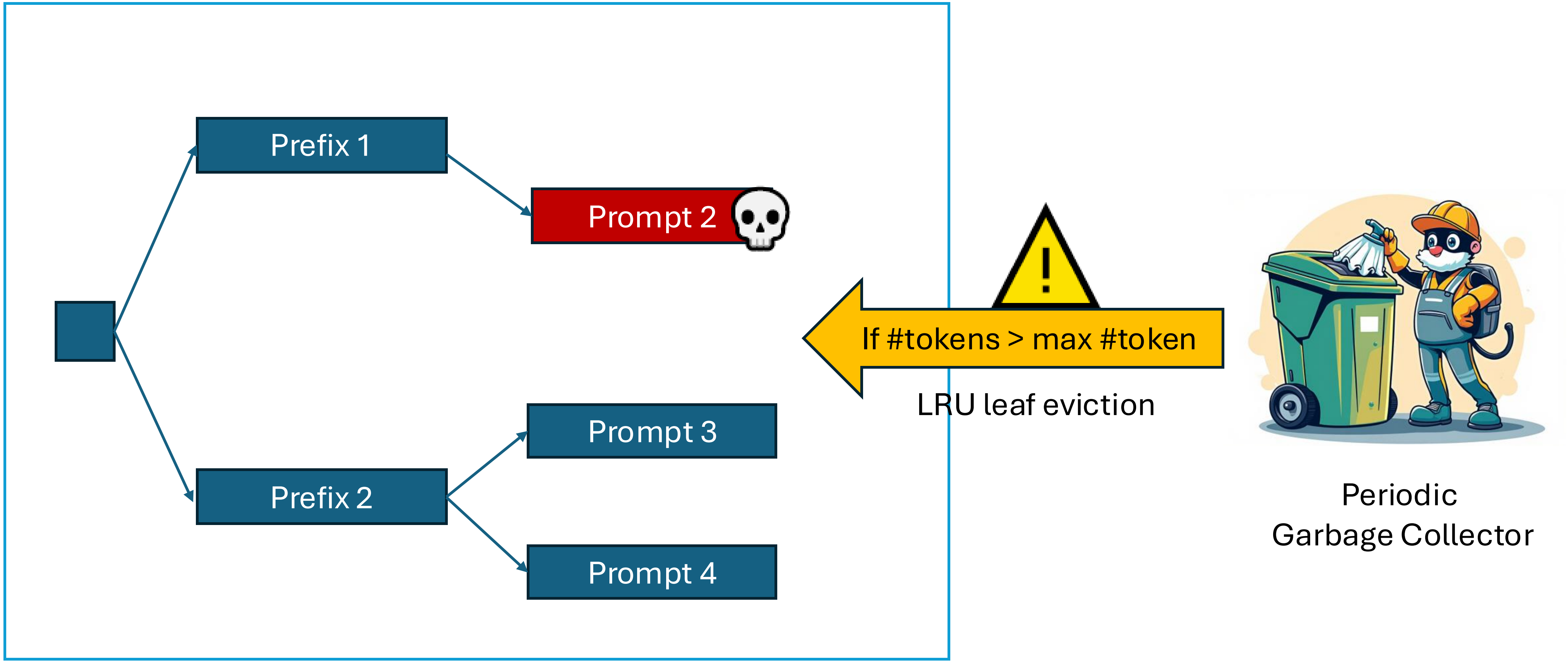
Constructing Approximate Tree



Evicting Approximate Tree (Lazy)

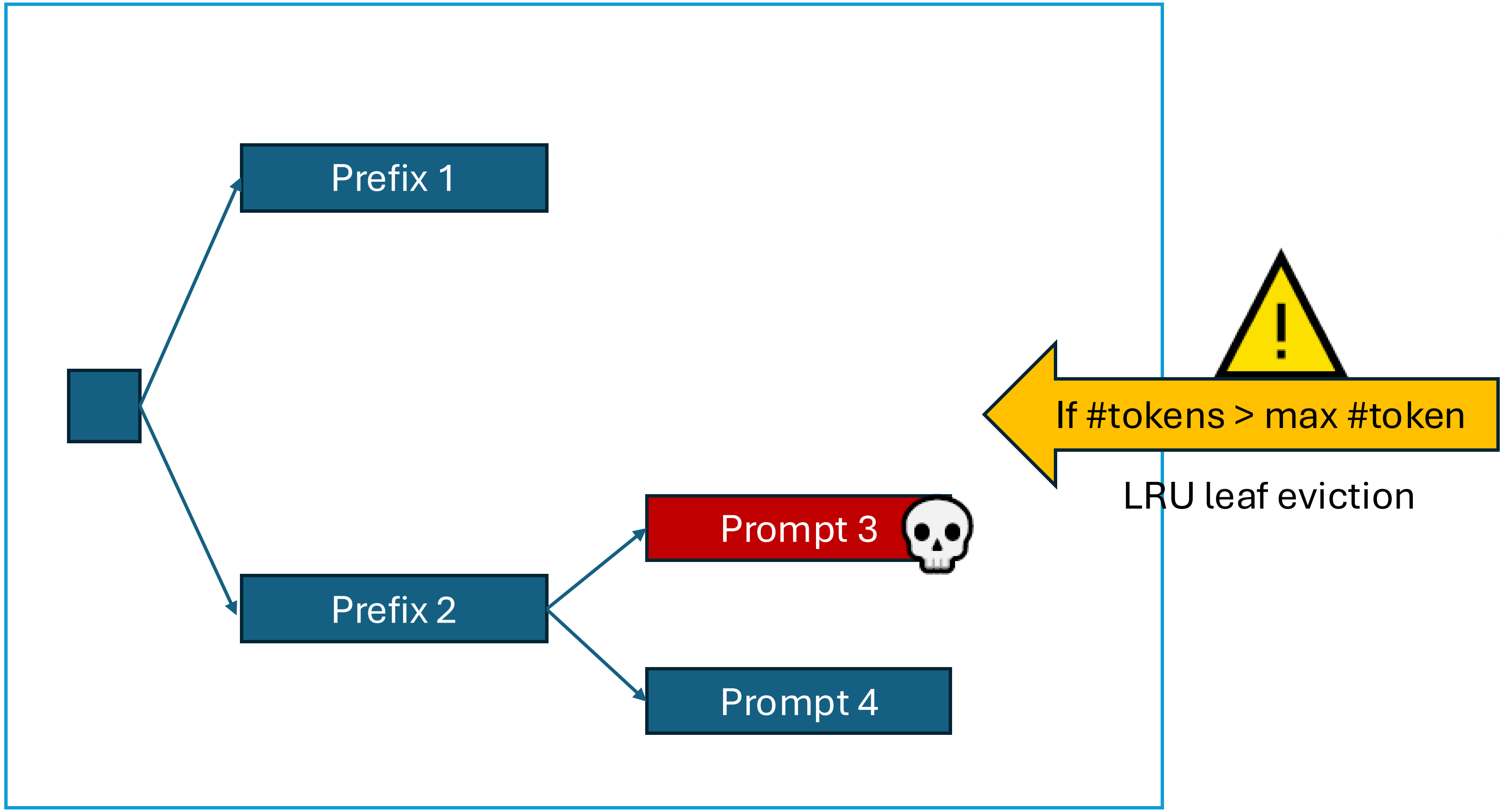


Evicting Approximate Tree (Lazy)



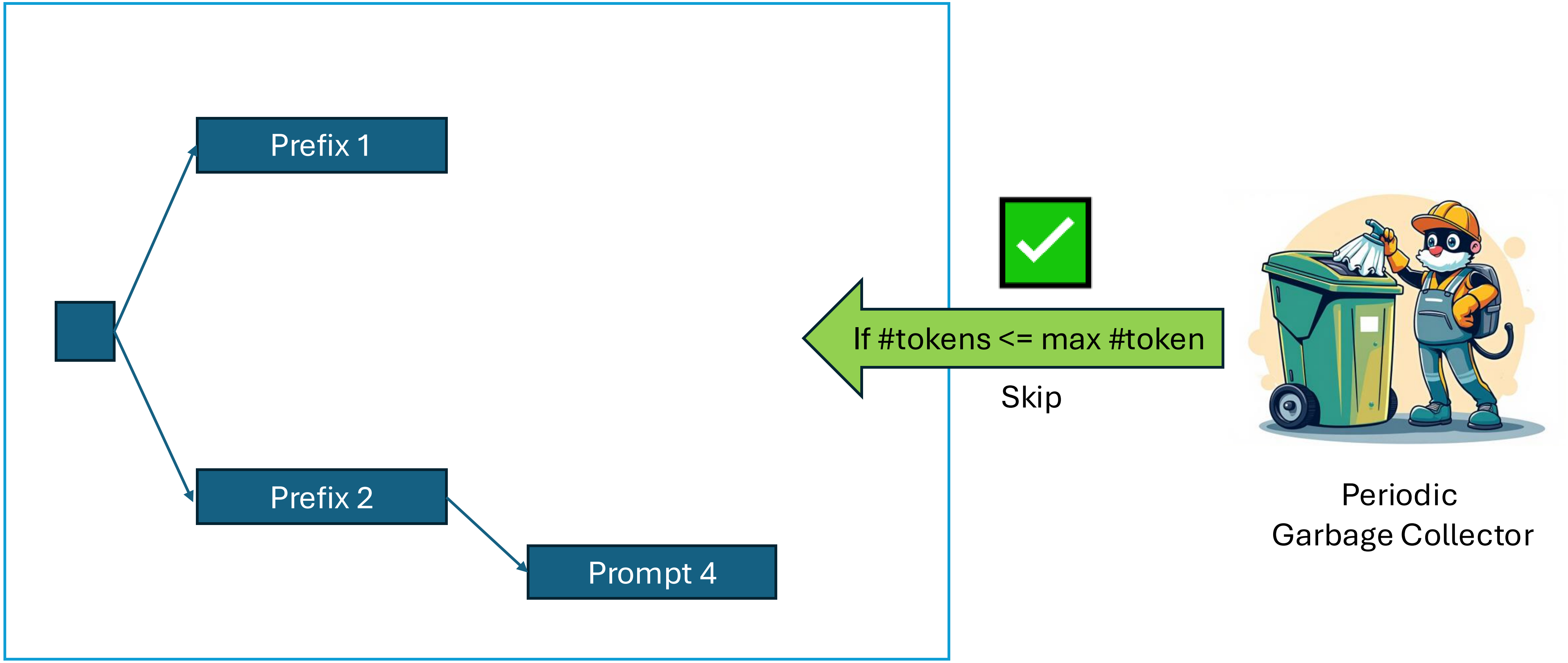
Periodic
Garbage Collector

Evicting Approximate Tree (Lazy)

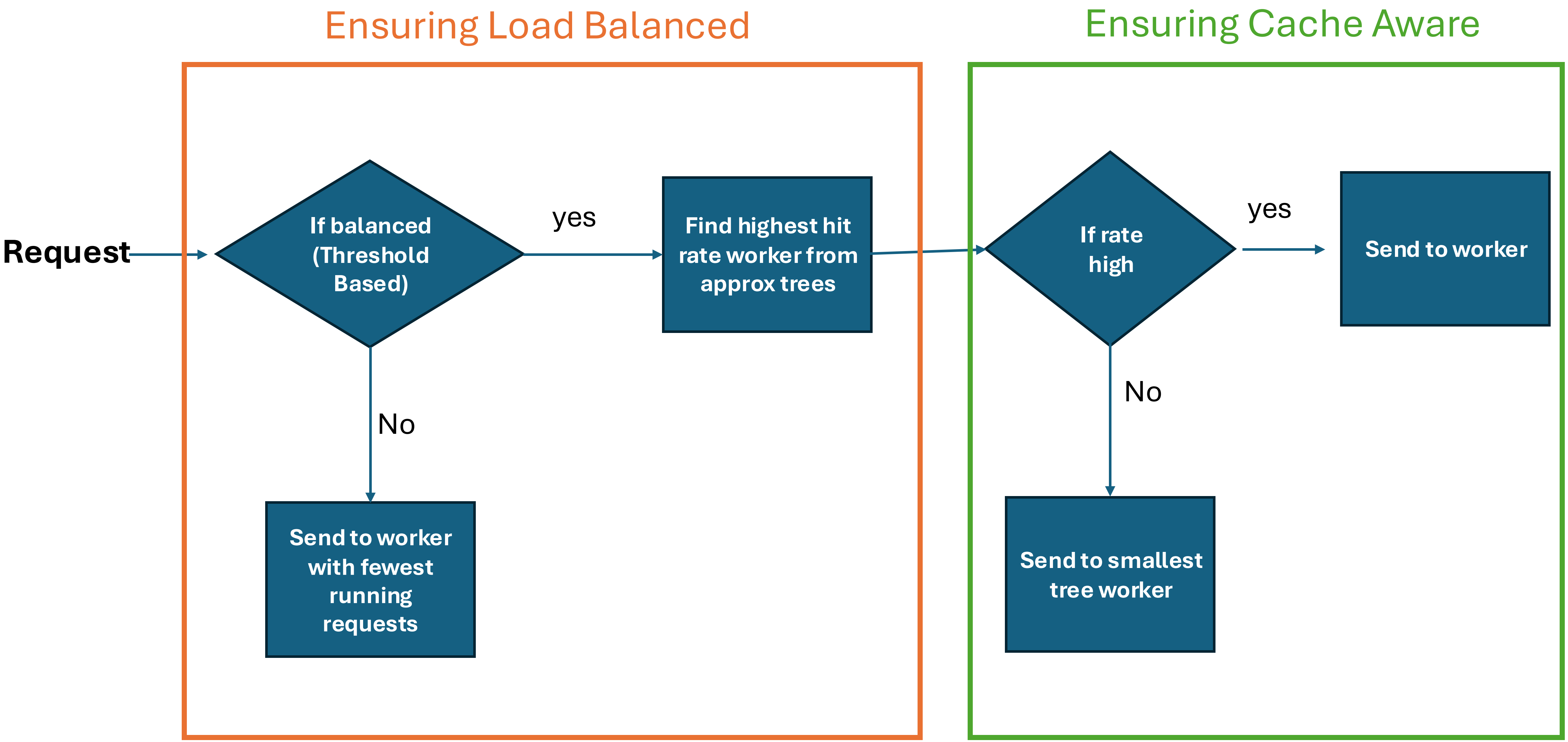


Periodic
Garbage Collector

Evicting Approximate Tree (Lazy)

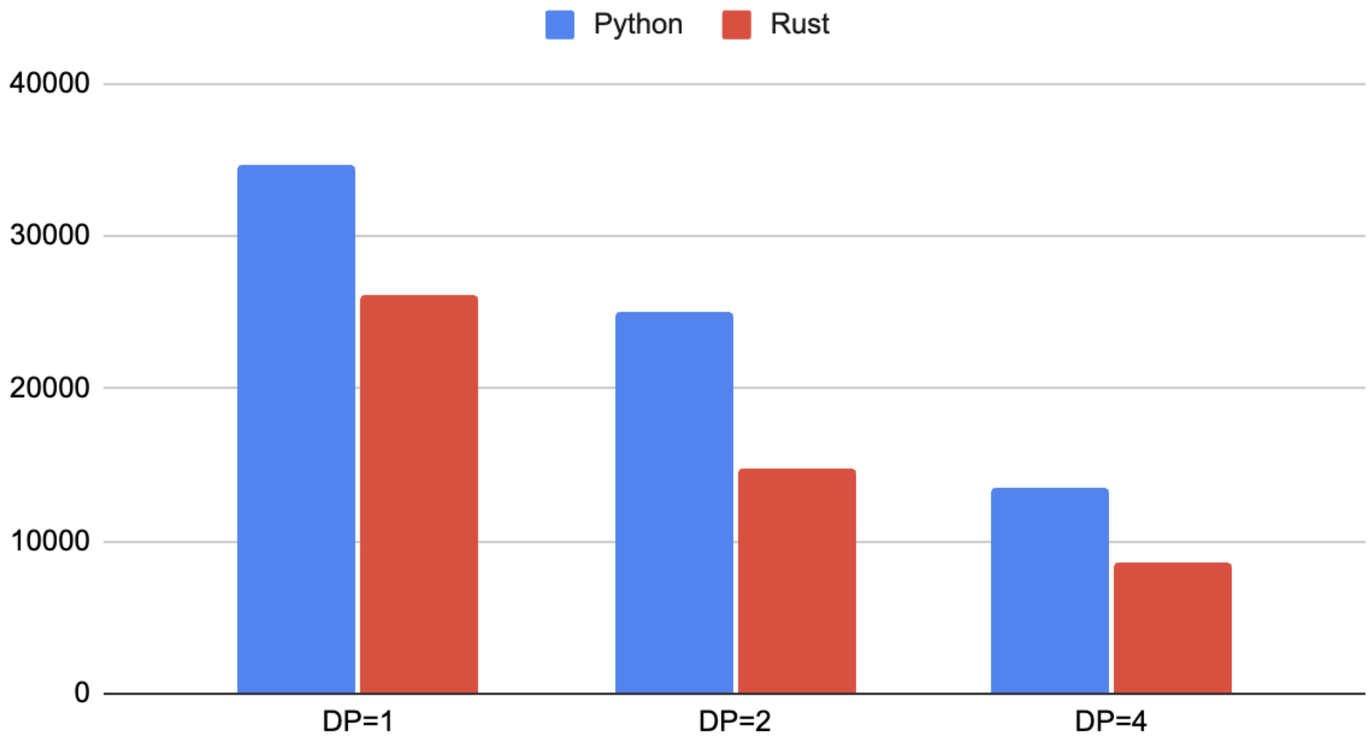


The Full Algorithm



Written in Rust

- Python produces significant overhead for forwarding requests
- Rust is extremely low overhead and provide good python binding support



Note: both are under round-robin

Dynamic Scaling

Provide HTTP endpoints to add and remove workers

- `/add_worker`
- `/remove_worker`

Usage:

```
$ curl -X POST http://localhost:30000/add_worker?url=http://worker_url_1
```

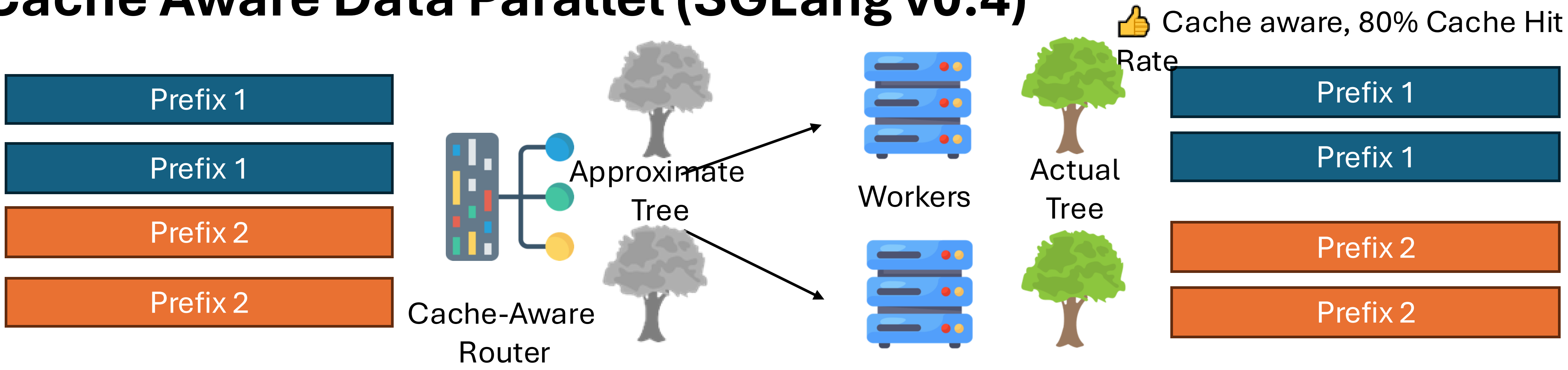
Example:

```
$ python -m sglang.launch_server --model-path meta-llama/Meta-Llama-3.1-8B-Instruct --port 3
$ curl -X POST http://localhost:30000/add_worker?url=http://127.0.0.1:30001
Successfully added worker: http://127.0.0.1:30001
```

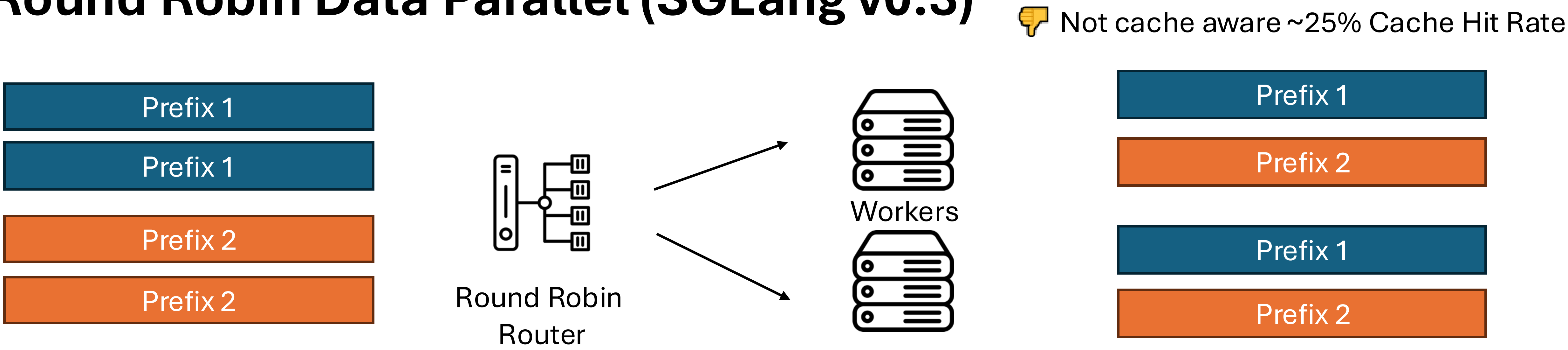
Fault Tolerance

1. If the request to a worker fails for **max_worker_retries** times, the router will remove the worker from the router and move on to the next worker.
2. If the total number of retries exceeds **max_total_retries**, the router will return an error.

Cache Aware Data Parallel (SGLang v0.4)



Round Robin Data Parallel (SGLang v0.3)



Benchmark

	SGLang v0.3	SGLang v0.4
Throughput (token/s)	82665	158596
Cache hit rate	20%	75%

The benchmark is conducted on a workload that has multiple long prefix groups, and each group is perfectly balanced.
The performance might vary based on the characteristics of the workload, but it should improve the cache hit rate significantly

- User A reported 25->70% hit rate improvement for llama3 70B on H100 TP=2
- User B reported 25->75% hit rate improvement for llama3 8B on A100 TP=1

LinkedIn 🤝 SGLang

- HTTP-free SGLang Engine (#1567)
- Cache-Aware Load Balancer (#1732)
- GRPC Server (#2478)