KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon

CloudNativeCon

THE LINUX FOUNDATION

China 2024

How to Increase the Throughput of Kubernetes Scheduler by Tens of Times

Yuquan Ren, Bing Li

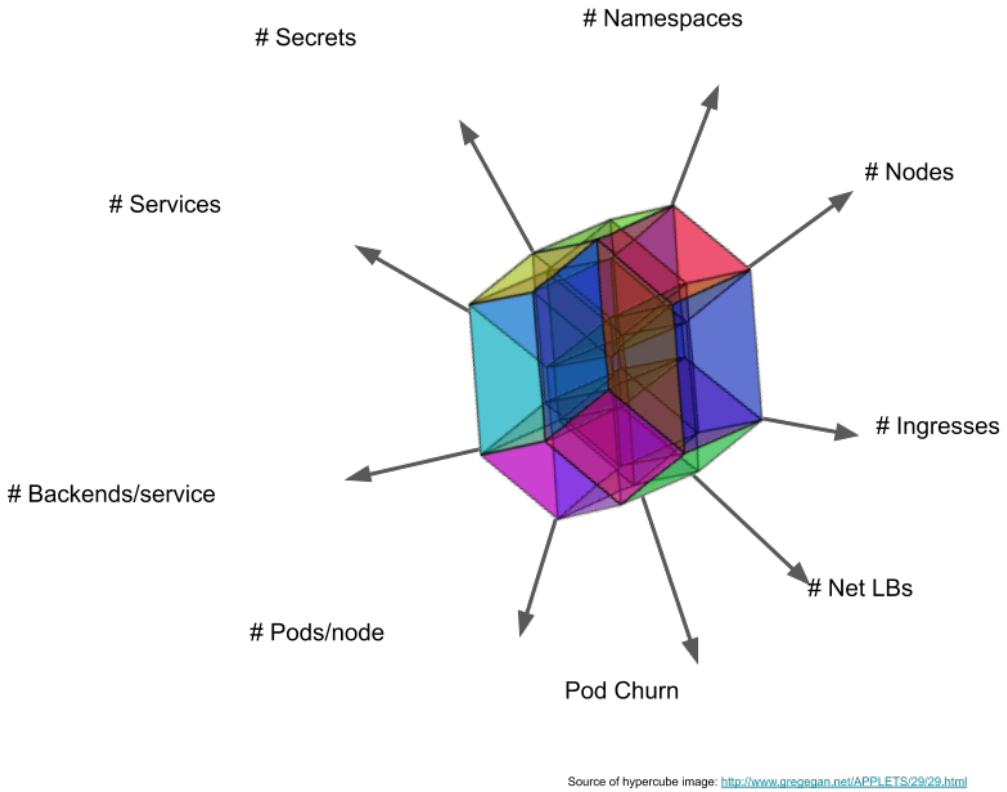
@ByteDance

Introduction

Content

- Background
- Scheduling performance optimizations
- Future work

Background - Kubernetes Scalability



Source of hypercube image: http://www.gregegan.net/APPLETS/29/29.html

Kubernetes Scalability Thresholds

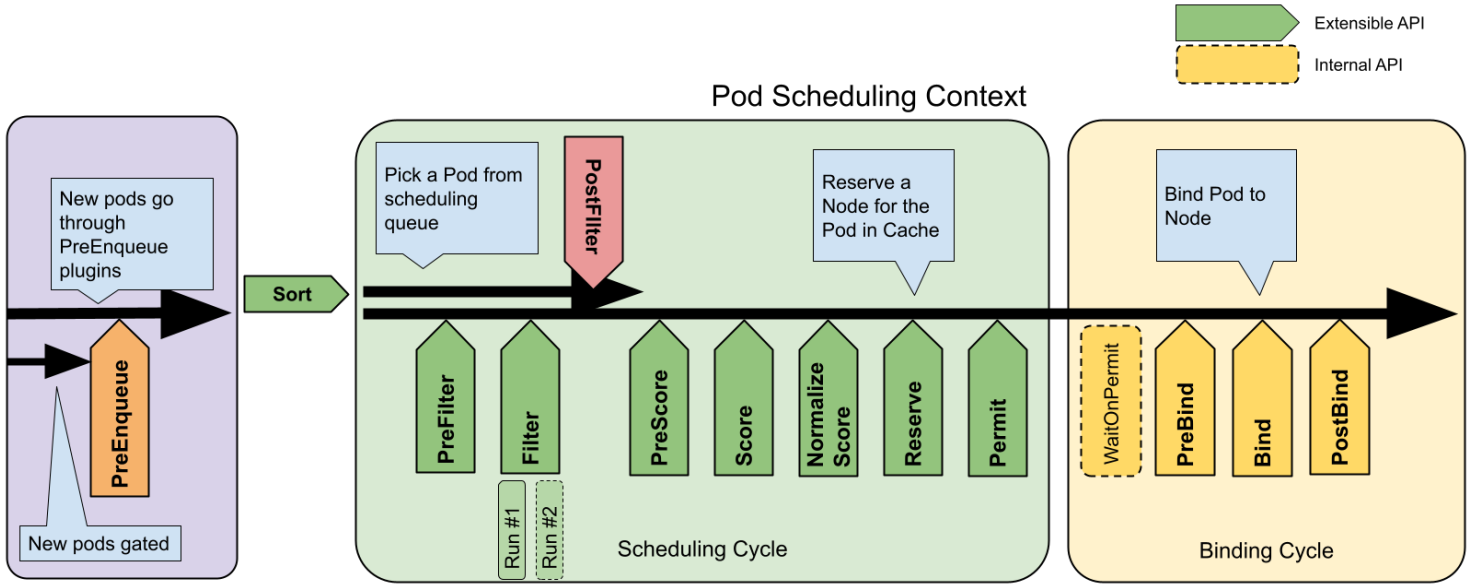
# Nodes	5000
# Pods	150000

No more than 100 Pods/s at ultra-large-scale cluster

-> Cannot support offline (stream and batch) workloads whose QPS usually exceed 1K+ Pods/s

-> Cannot further improve resource utilization by colocation in large-scale cluster

Background - Kubernetes Scheduler



Architecture: monolithic

Detailed implementation:
- queue
- cache
- snapshot (data syncing)
- scheduling
- preemption

Gödel Scheduling System

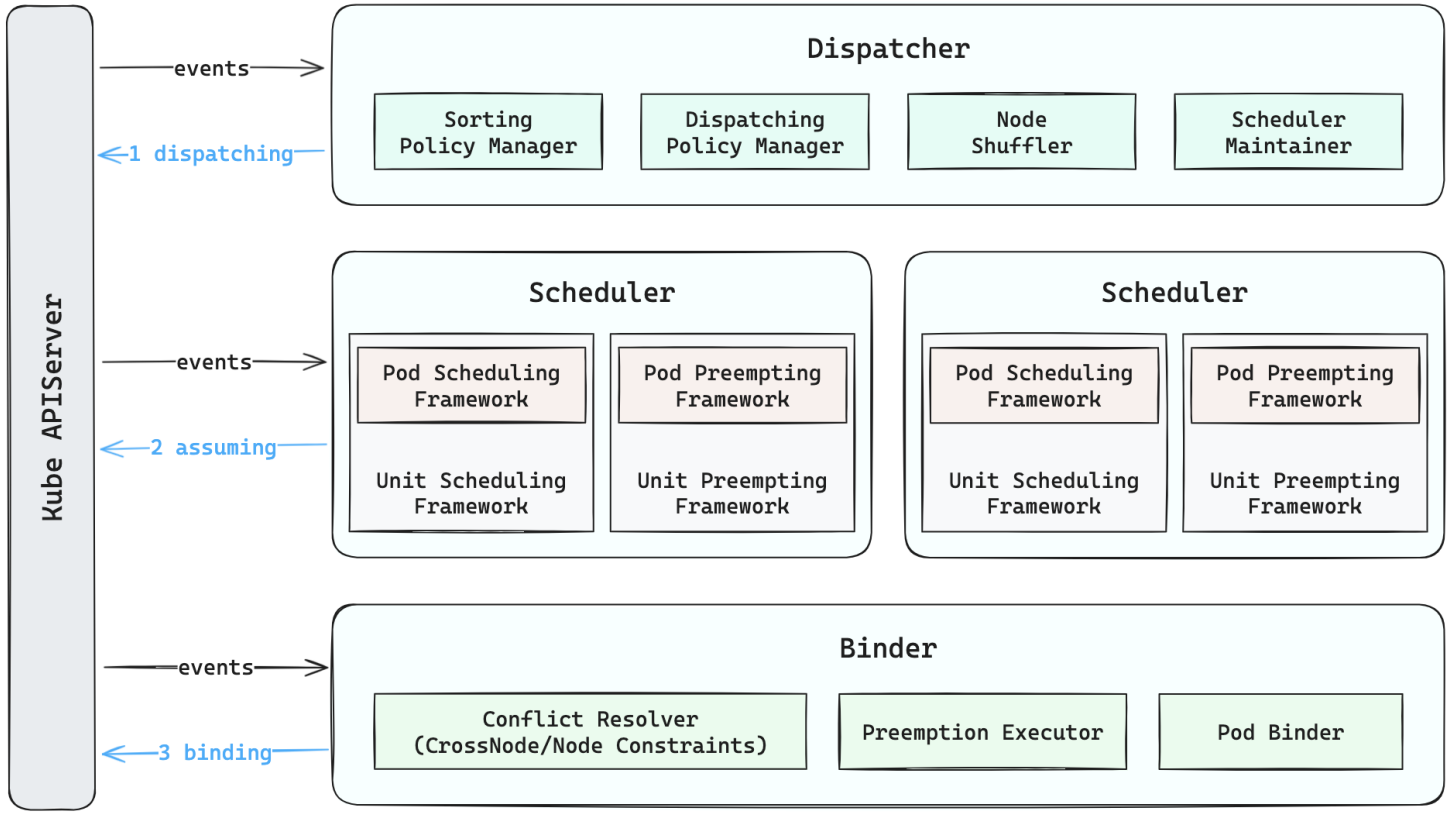
How to improve scheduling throughput ?

Gödel

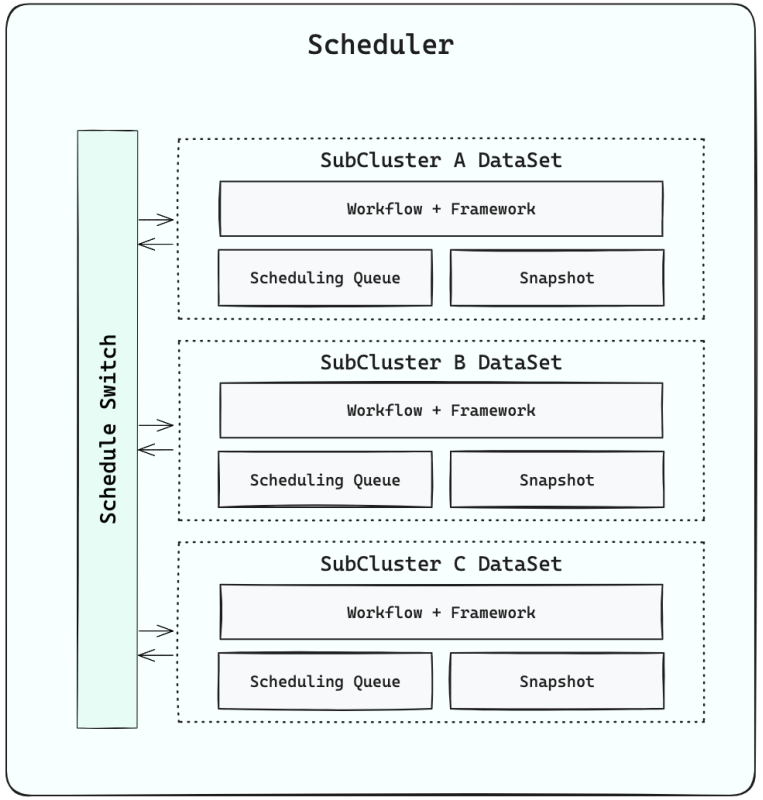
a unified scheduling system for both
online (micro service…) and offline (ml, stream, batch…) workloads

Gödel Scheduling System - Architecture

Optimistic Concurrency Architecture



SubCluster Concurrent Scheduling



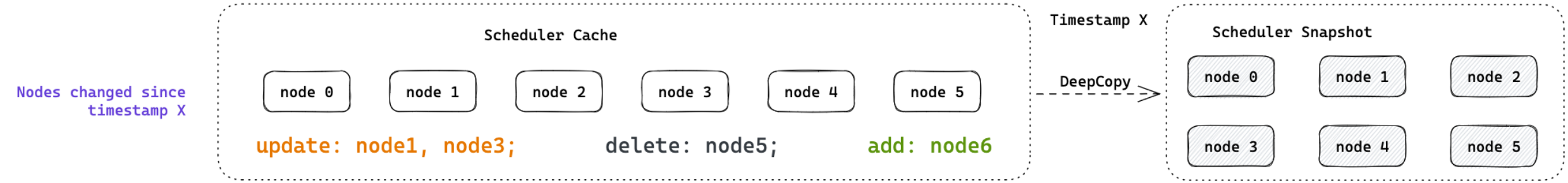
Gödel Scheduling System - Optimizations

Optimize single-shard performance to the utmost through

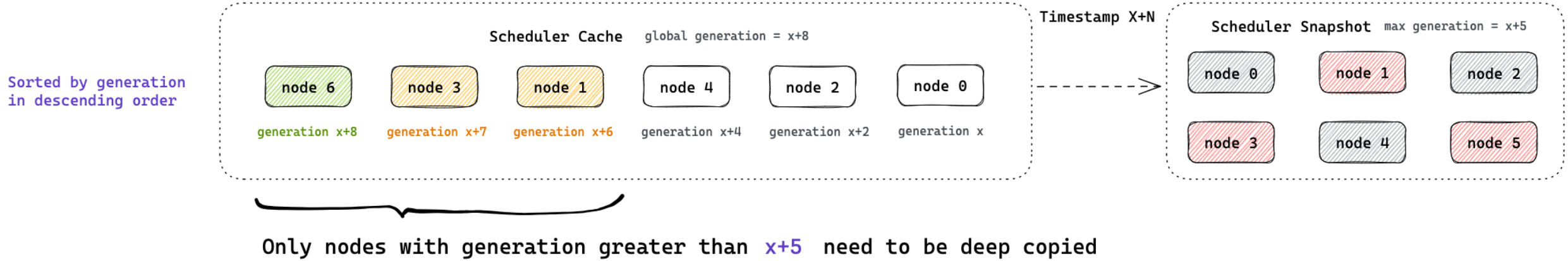
data structures & algorithms

Gödel Optimizations - Data Synchronization

How does Snapshot perceive changes in the Cache?



What if we maintain a generation that is time-sensitive?

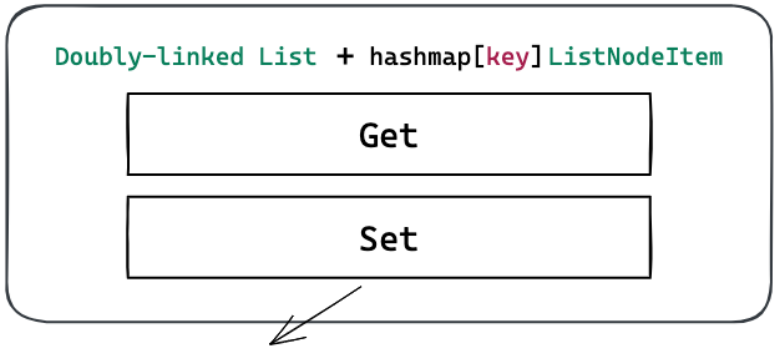


Only nodes with generation greater than x+5 need to be deep copied

Gödel Optimizations - Data Synchronization

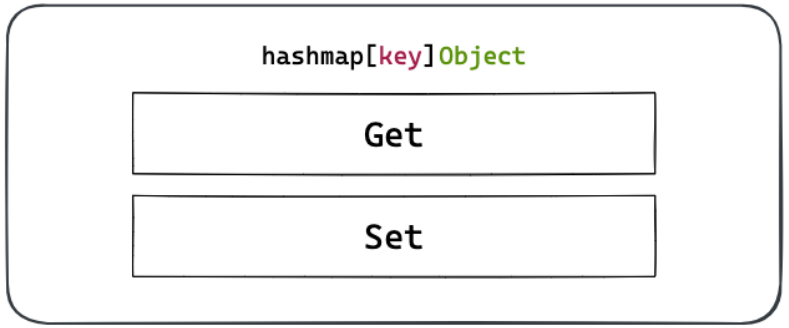
Generation Store

List Store: will be used in Cache



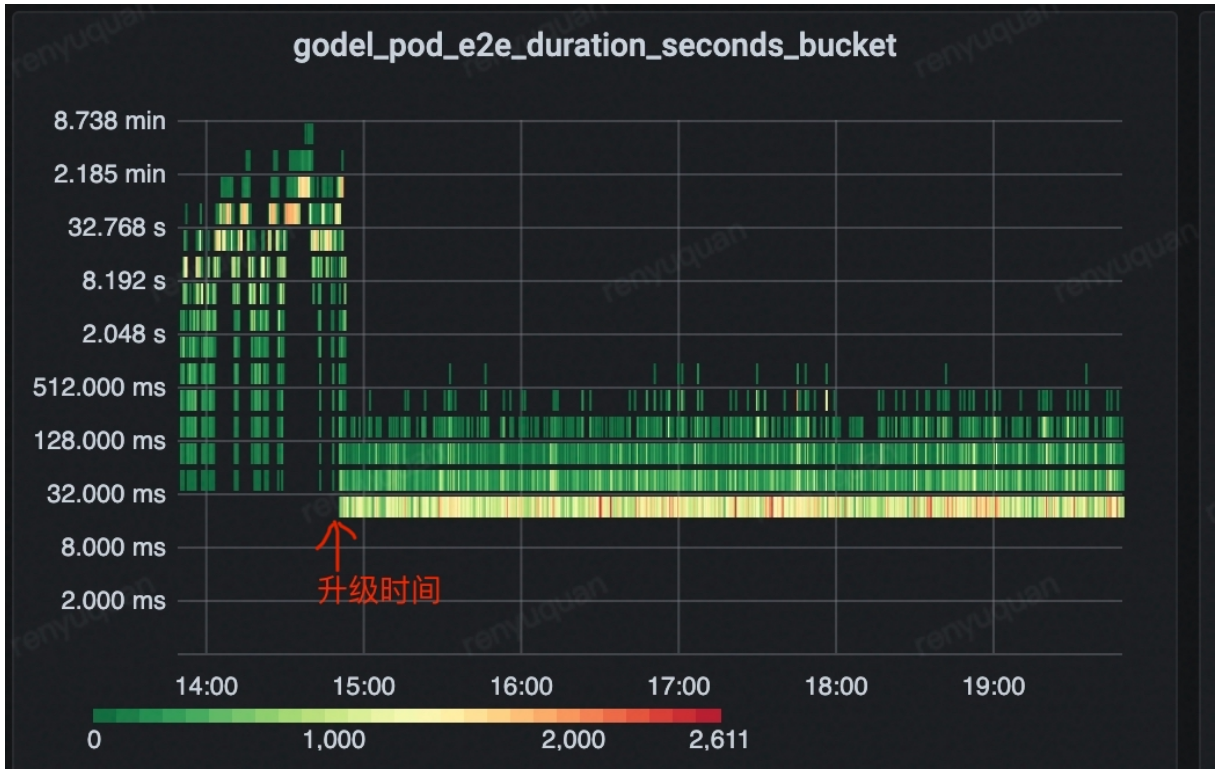
Set will maintain both generation changes and the orderliness of the doubly-linked list

Raw Store: will be used in Snapshot



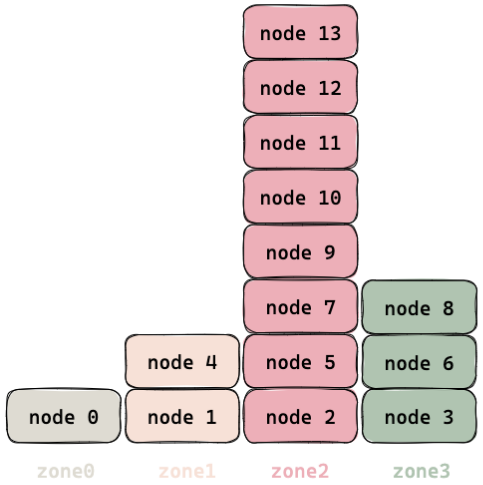
Abstract the underlying storage GenerationStore that supports incremental updates

Migrate all existing storage to GenerationStore

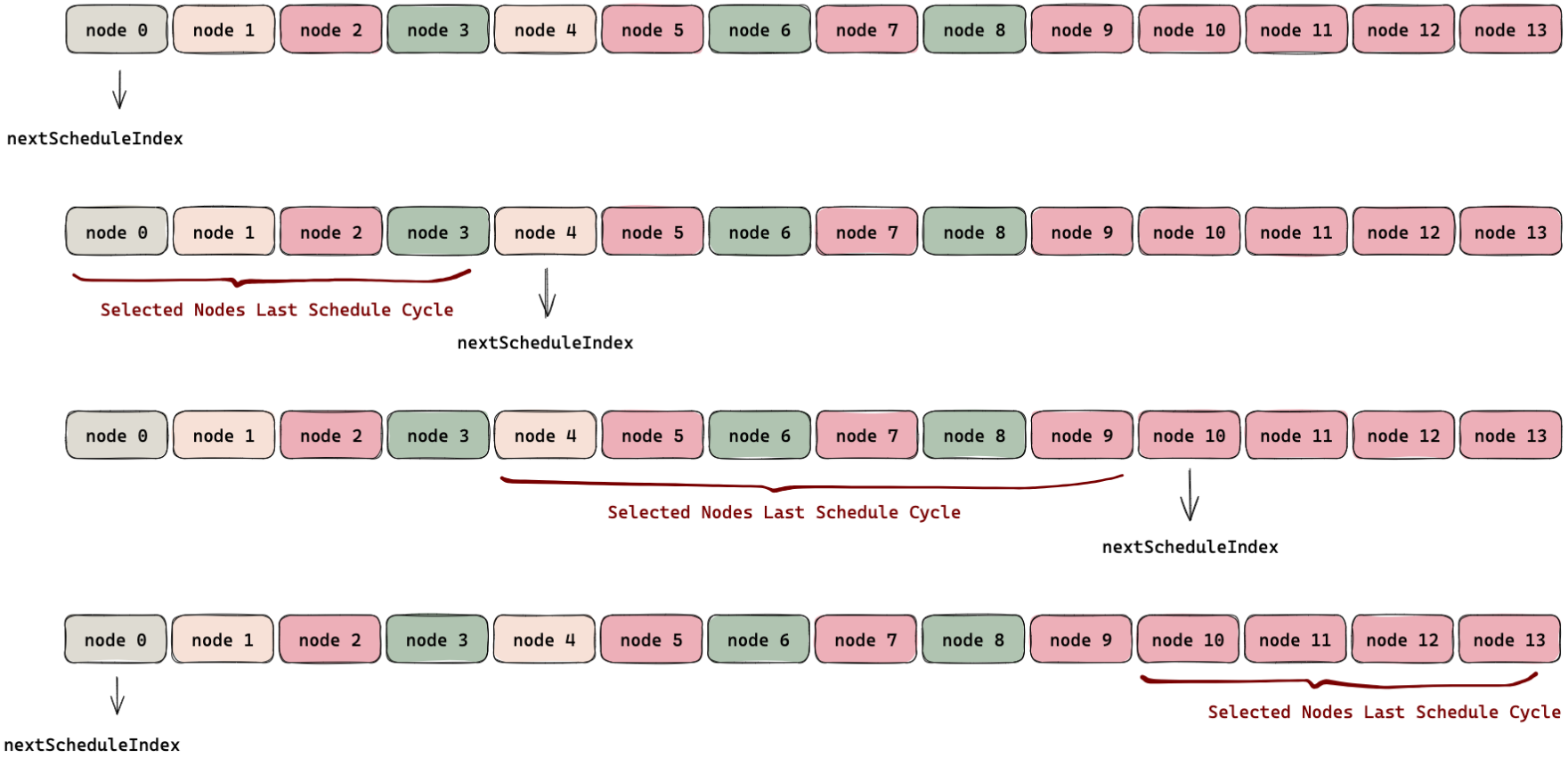


Gödel Optimizations - Data Synchronization

NodeTree in Cache



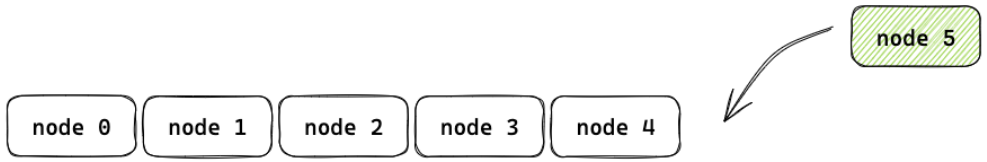
NodeList in Snapshot



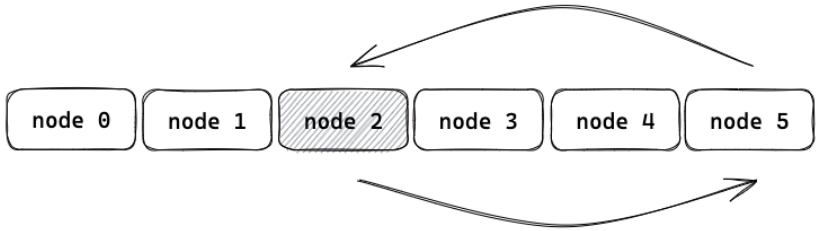
Gödel Optimizations - Data Synchronization

HashSlice in Snapshot

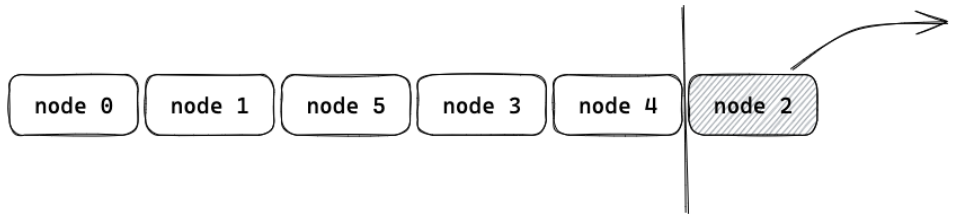
- hash: map[string]int
- slice: []NodeInfo



Place the new node directly at the end

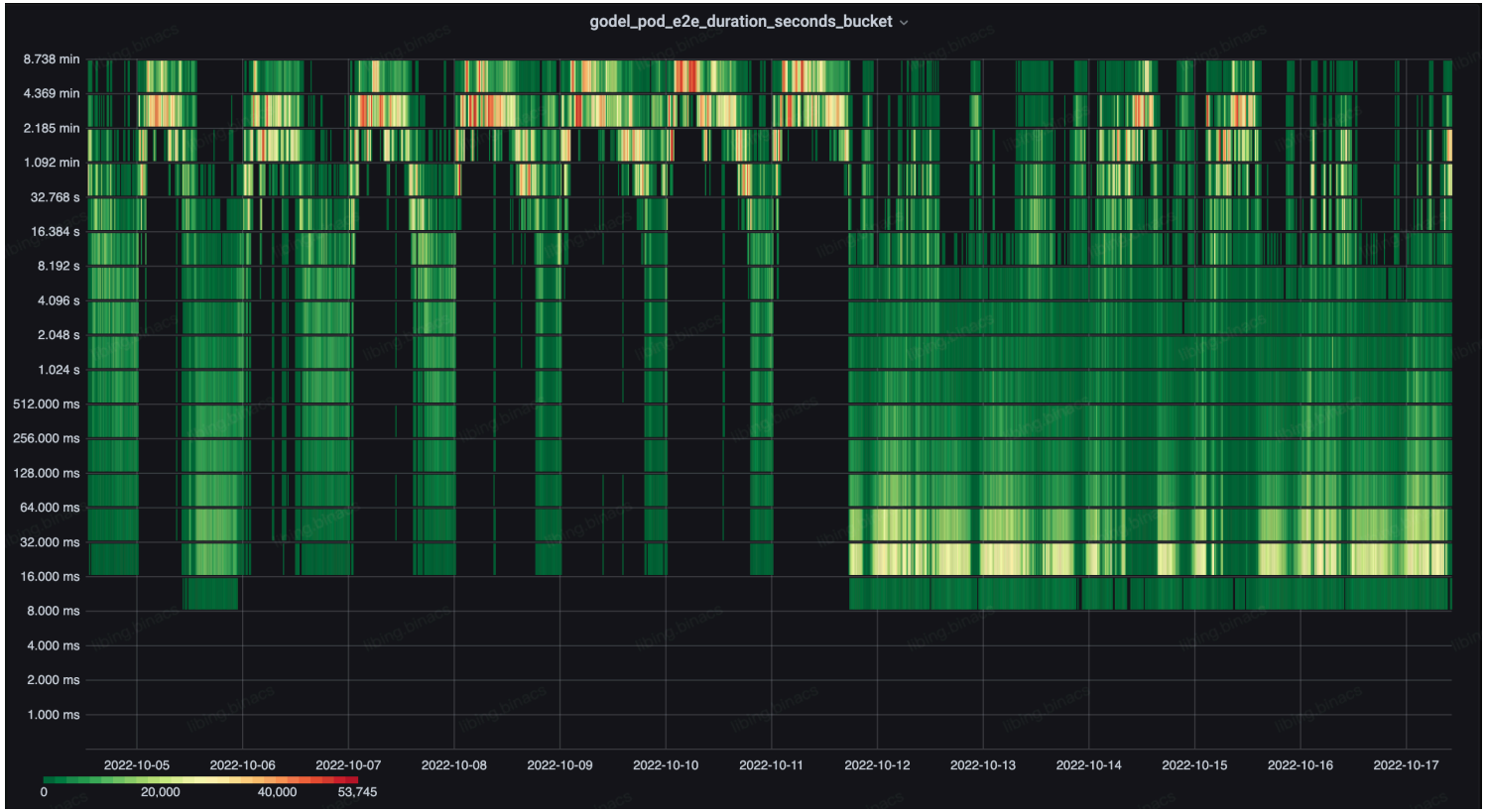


First, swap the node to be removed with the end node



Then, remove the end node directly

Gödel Optimizations - Data Synchronization

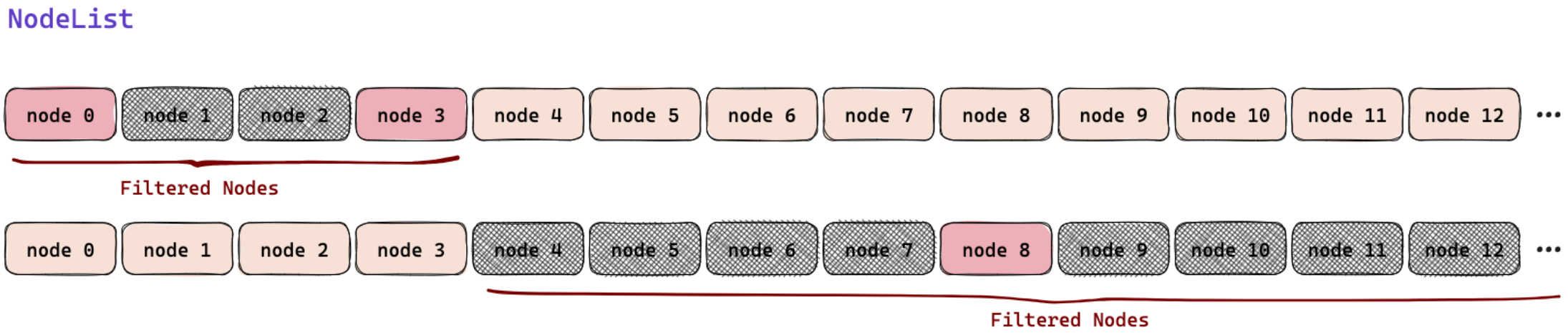


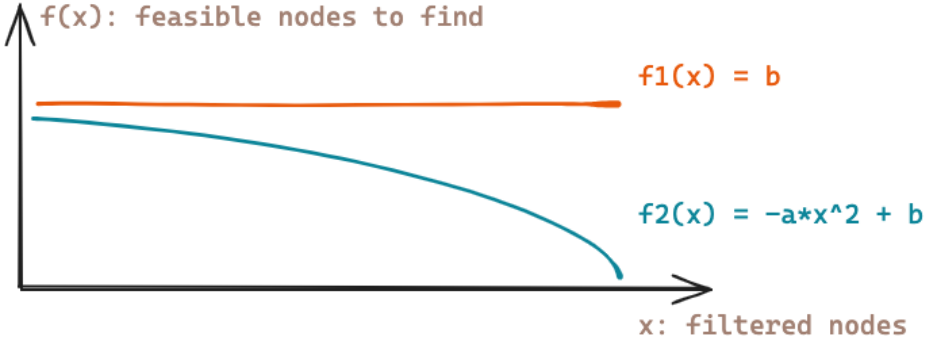
Ultra-large-scale cluster with 20K+ Nodes and 1M+ Pods, while 1K+ Incoming Pods/s

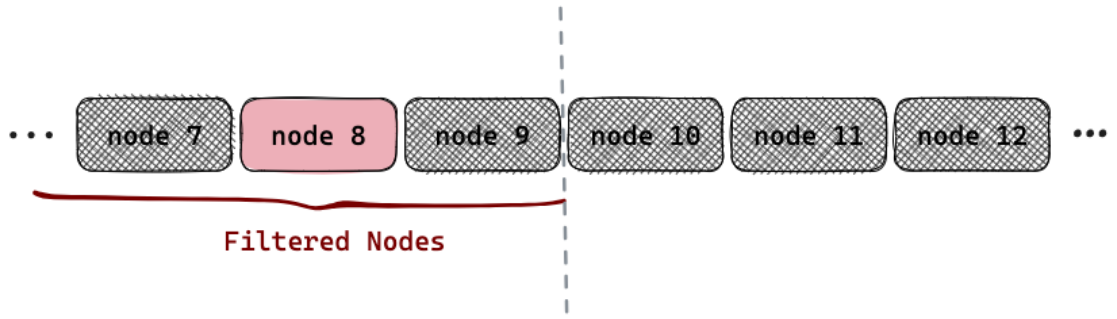
E2E latency distribution shift from minutes to milliseconds

Gödel Optimizations - Scheduling

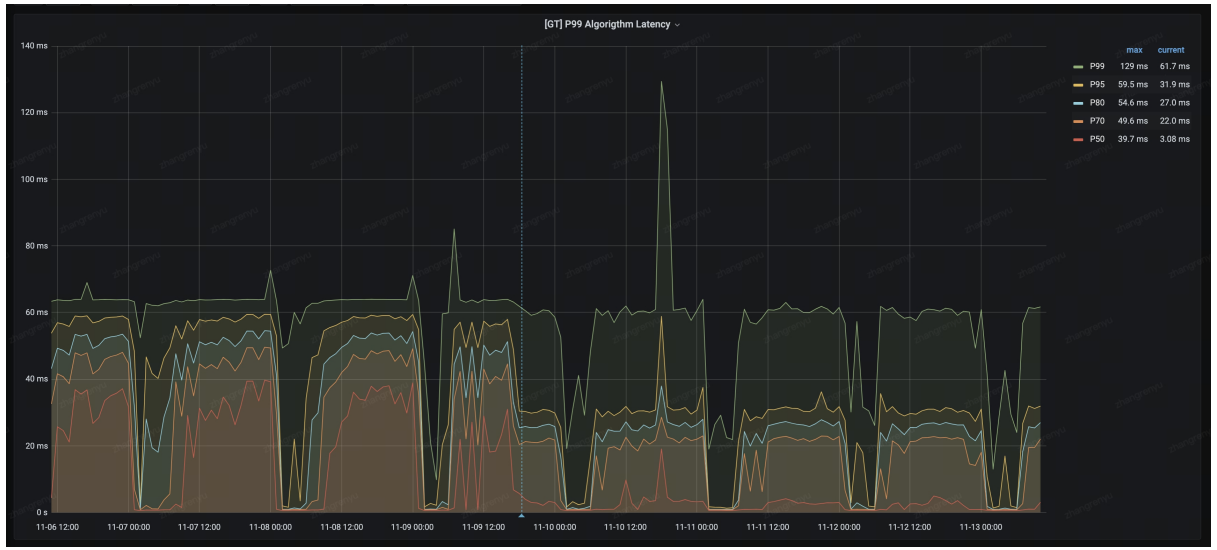
High Water Level





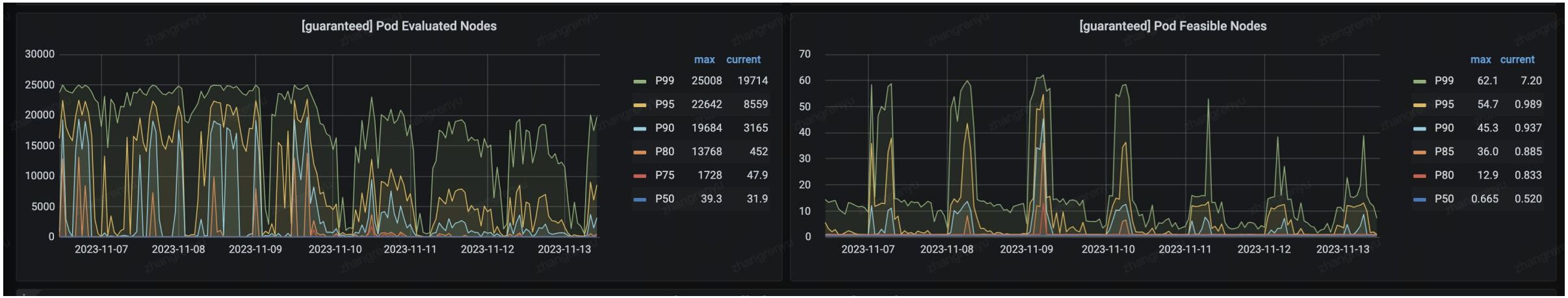


Gödel Optimizations - Scheduling



Algorithm Latency of almost all Pods is reduced by more than 50%

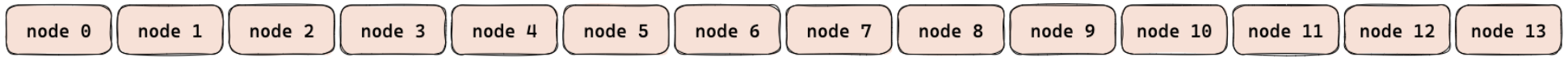
Reducing evaluated nodes while NOT significantly decrease feasible nodes

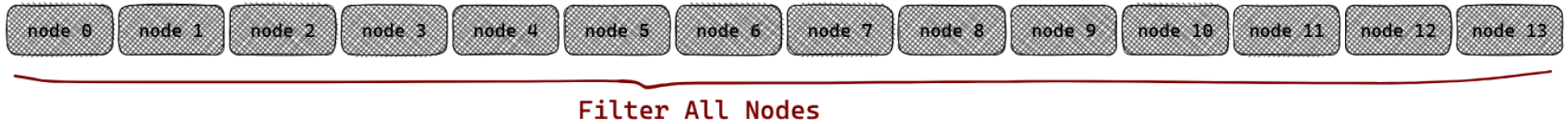


Gödel Optimizations - Scheduling

Unschedulable

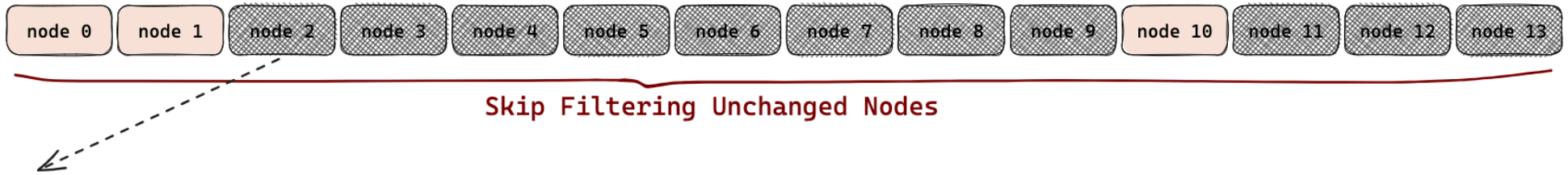
First Round






```
pod.LastSchedulingNodeGeneration = snapshot.GetGlobalMaxGeneration()
```

Second Round



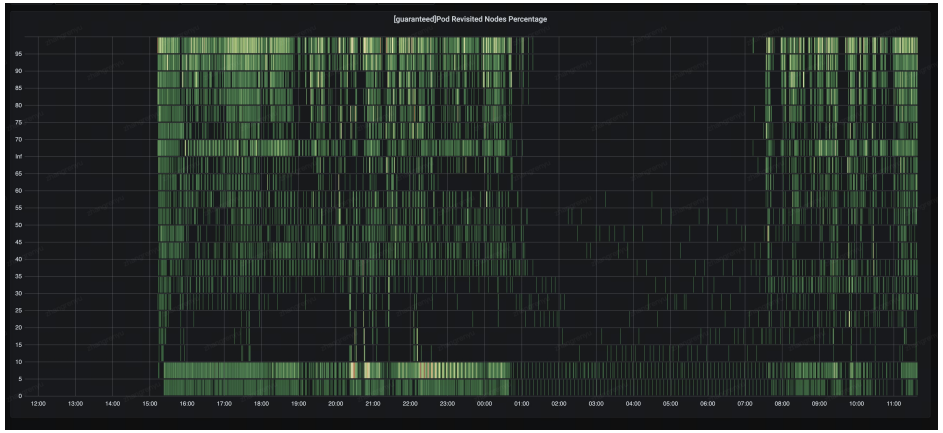

```
if pod.LastSchedulingNodeGeneration ≥ nodeInfo.GetGeneration():
    skip
```

Gödel Optimizations - Scheduling

tce.godel.trace :: scheduler::filter	-	26.64	19:30:17.032849
tce.godel.trace :: scheduler::preemptUnit	-	2.92	19:30:17.074290
tce.godel.trace :: scheduler::preemptPod	-	0.32	19:30:17.074329
tce.godel.trace :: scheduler::pendingInQueue	-	25508.28	19:30:17.077336
tce.godel.trace :: scheduler::updatePodInSchedulingQueue	-	0.05	19:30:17.096118
tce.godel.trace :: scheduler::schedule	-	9.54	19:30:42.586039
tce.godel.trace :: scheduler::scheduleUnit	-	8.79	19:30:42.586054
tce.godel.trace :: scheduler::schedulePod	-	8.52	19:30:42.586074
tce.godel.trace :: scheduler::getCachedNodes	-	0.02	19:30:42.586248
tce.godel.trace :: scheduler::filter	-	7.07	19:30:42.586293
tce.godel.trace :: scheduler::preemptUnit	-	0.68	19:30:42.594858
tce.godel.trace :: scheduler::preemptPod	-	0.28	19:30:42.594888
tce.godel.trace :: scheduler::pendingInQueue	-	20772.05	19:30:42.595627
tce.godel.trace :: scheduler::updatePodInSchedulingQueue	-	0.26	19:30:42.599404
tce.godel.trace :: scheduler::schedule	-	9.04	19:31:03.368229
tce.godel.trace :: scheduler::scheduleUnit	-	8.38	19:31:03.368246
tce.godel.trace :: scheduler::schedulePod	-	8.18	19:31:03.368271
tce.godel.trace :: scheduler::getCachedNodes	-	0.02	19:31:03.368438
tce.godel.trace :: scheduler::filter	-	7.06	19:31:03.368482
tce.godel.trace :: scheduler::preemptUnit	-	0.58	19:31:03.376645
tce.godel.trace :: scheduler::preemptPod	-	0.27	19:31:03.376675
tce.godel.trace :: scheduler::pendingInQueue	-	40864.01	19:31:03.377317
tce.godel.trace :: scheduler::updatePodInSchedulingQueue	-	0.05	19:31:03.381101
tce.godel.trace :: scheduler::schedule	-	1.46	19:31:44.241923
tce.godel.trace :: scheduler::scheduleUnit	-	1.38	19:31:44.241937
tce.godel.trace :: scheduler::schedulePod	-	1.35	19:31:44.241957
tce.godel.trace :: scheduler::getCachedNodes	-	0.02	19:31:44.242129
tce.godel.trace :: scheduler::filter	-	1.00	19:31:44.2421

Many nodes remained unchanged during multiple scheduling attempts

Filter duration from 27ms to 7ms



Gödel Optimizations - Preemption

Preemption Overview

Concurrent execution of the preemption process on different nodes

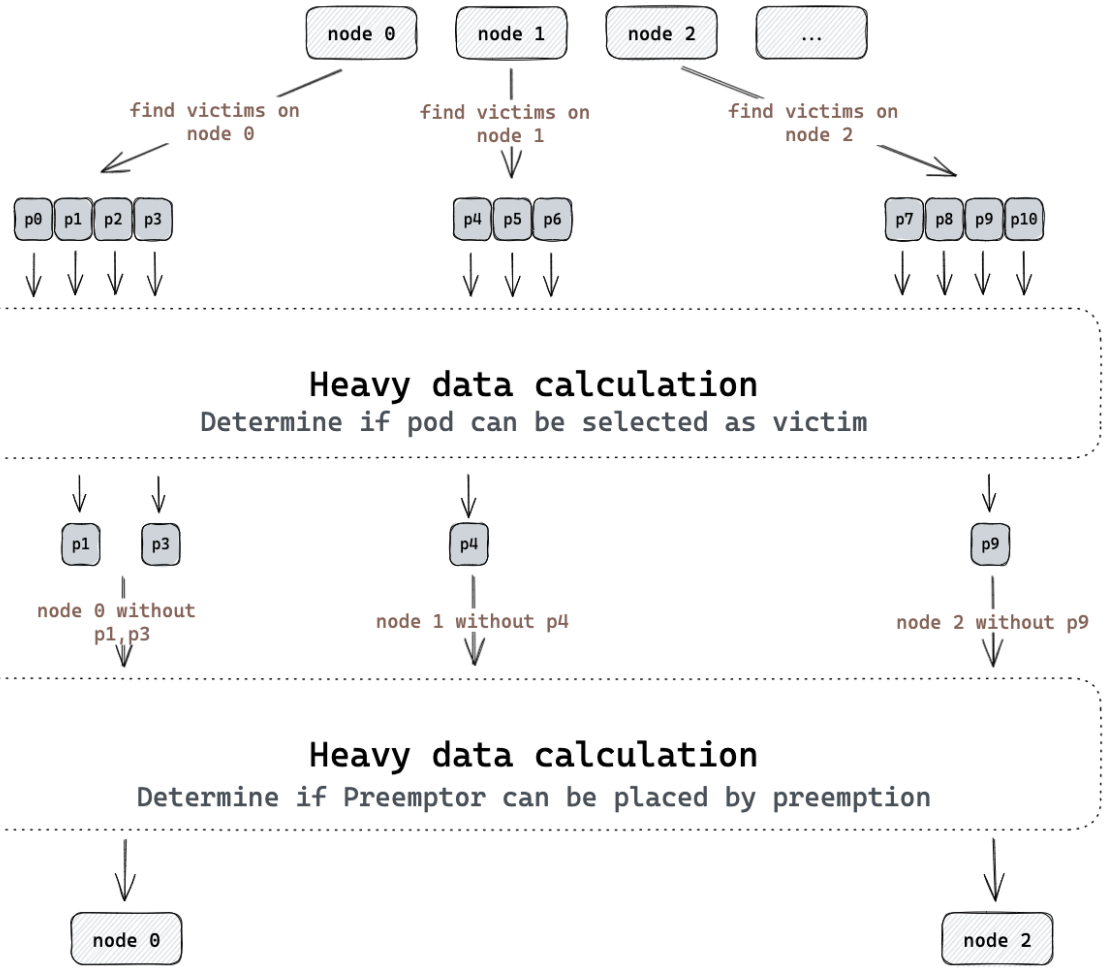
All pods enter the calculation process without discrimination

Pod VS Preemptor

Get a list of pods that can be preempted

Node VS Preemptor

Node Candidates



Gödel Optimizations - Preemption

How can we make fewer pods enter the data calculation process?

Maintain all pods on a node in the event handler

p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10

Filter out pods that can not be preempted

p0 p1 p2 p3 p4 p6 p7 p8 p9 p5 p10

Classify pods by resource request type

BE pods: p0 p2 p4 p8 p9

GT pods: p1 p3 p6 p7

Sort pods by priority from lowest to highest

BE pods: p4 p2 p0 p9 p8

GT pods: p6 p3 p1 p7

For a GT preemptor, only pods that also belong to GT and have a lower priority than the preemptor need to be involved.

Only two pods need to be involved in the data calculation process!

Gödel Optimizations - Preemption

How can we make fewer nodes enter the preemption process?

The basic principle of preemption is priority sorting

Sort pods by priority from lowest to highest

BE pods: p4 p2 p0 p9 p8

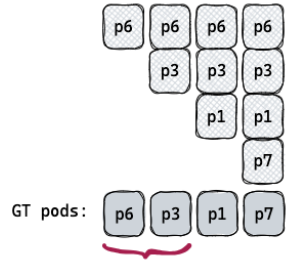
GT pods: p6 p3 p1 p7

For a GT preemptor, only pods that also belong to GT and have a lower priority than the preemptor need to be involved.

What if we could quickly determine if removing all these pods would successfully place the preemptor?

The essence of preemption is to release resources

Maintains the prefix sum of all pods resource requests



$$\text{prefixSum}[i] \sum_{j=1}^{i} A[i]$$

The resources released through preemption will not exceed the sum of p6 and p3

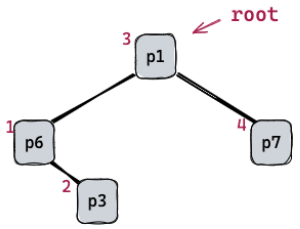
Quick fail: node remaining resources + releasable resources < preemptor request

---

How to maintain the prefix sum of resource requests while maintaining the pods order?

Sort pods by priority from lowest to highest in BST

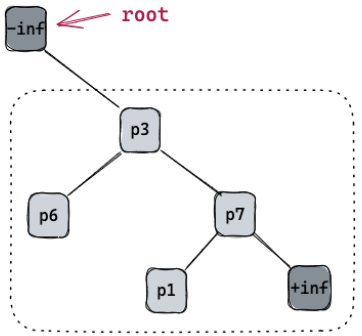
GT pods:



It is very difficult to maintain the prefix sum directly by the in-order traversal.
But maintaining the sum of requests for subtrees at each tree node is very simple.

Yes, Splay Tree!

Get the sum of resource requests from the first 4 pods



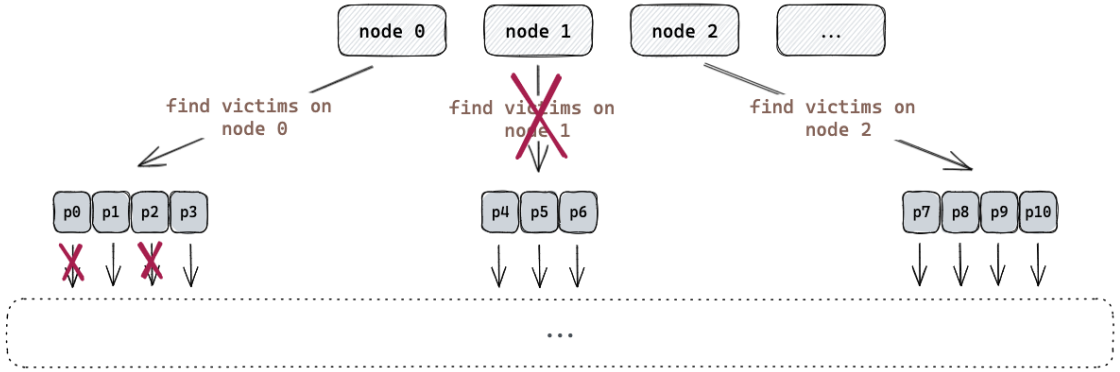
In the case of 200 tree nodes, no more than 8 rotation operations are required to obtain the sum of resource requests for a subtrees!

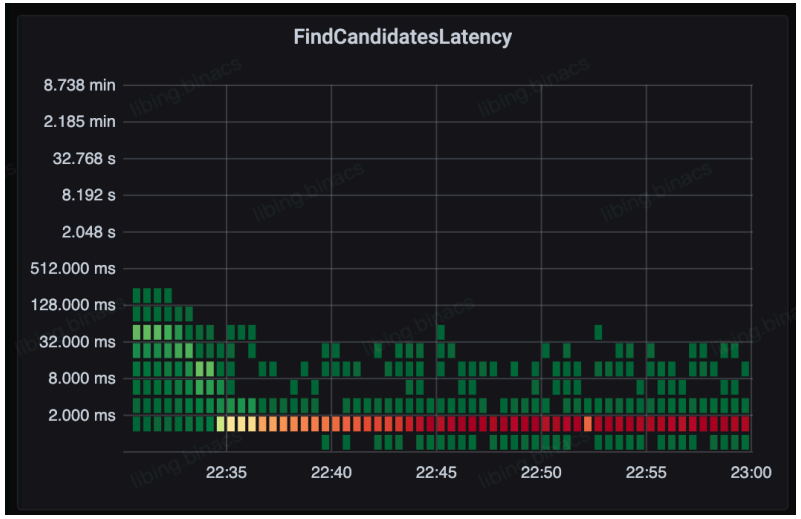
Gödel Optimizations - Preemption

Preemption Overview

Concurrent execution of the preemption process on different nodes
But some nodes will end quickly due to failing the heuristic check

Not All pods enter the calculation process ~~without discrimination~~

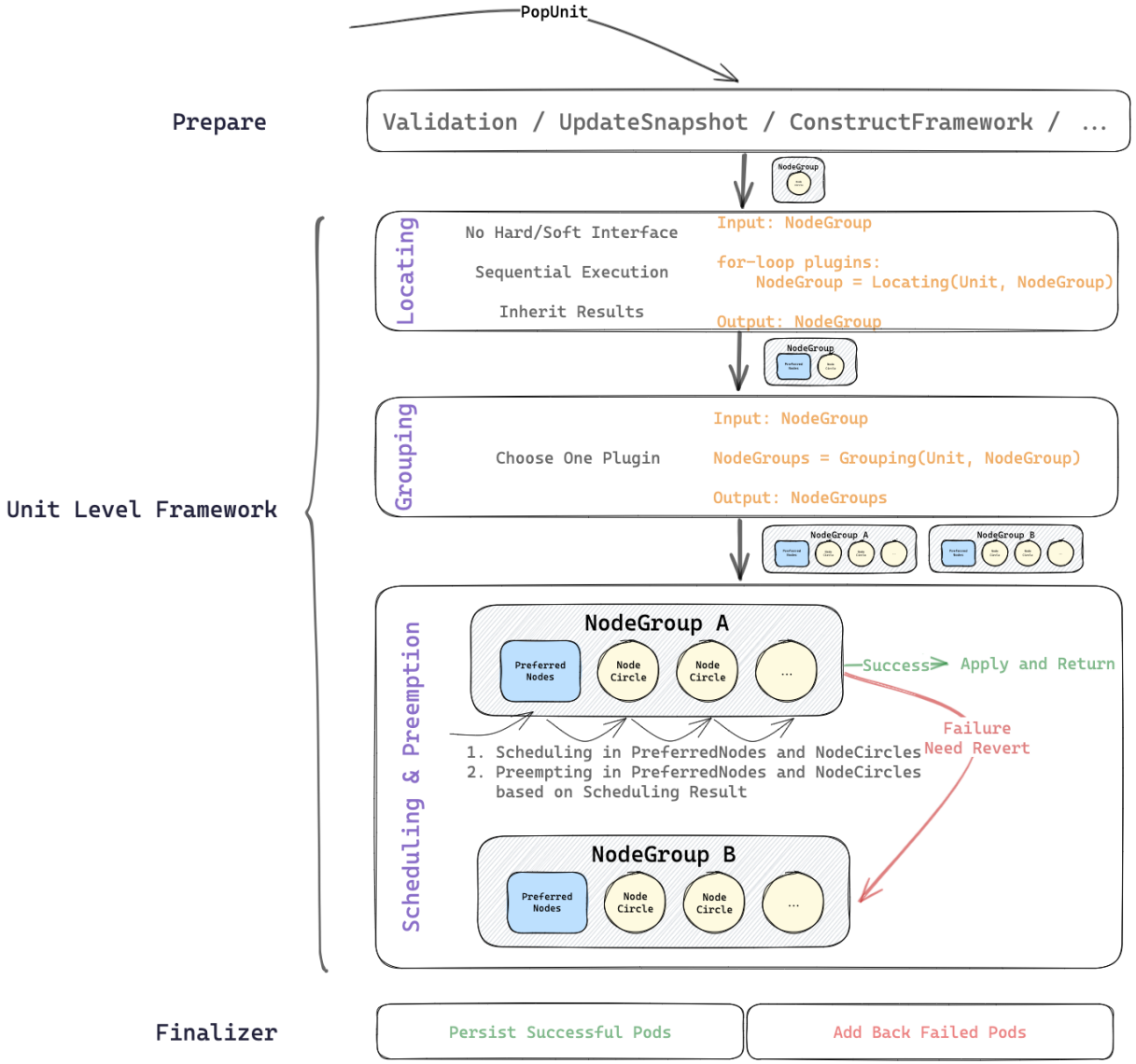




10x improvement in overall throughput

Filter out non-preemptable cases within 2ms by heuristic pruning

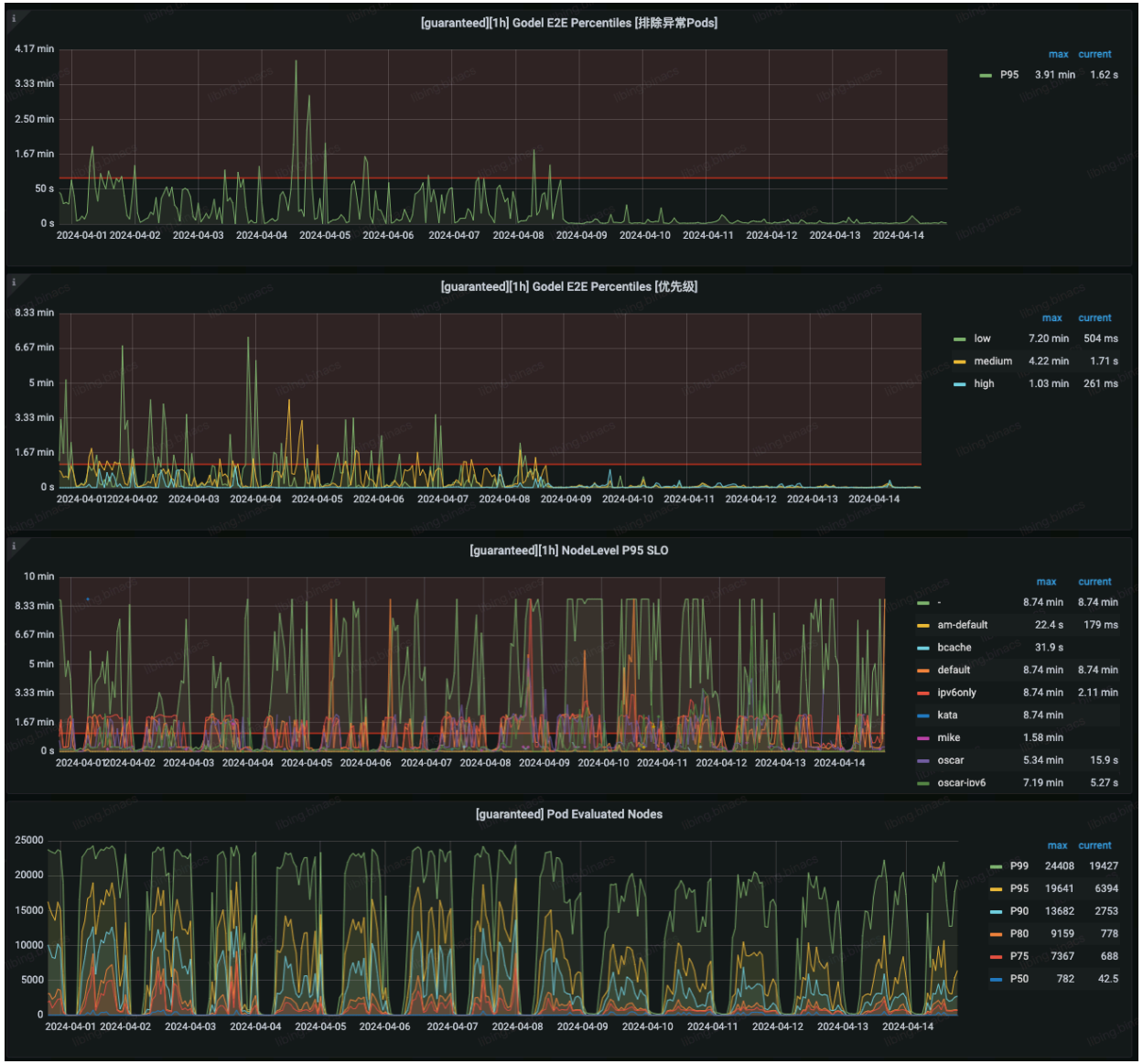
Gödel Optimizations - Unit Semantic & Unit Framework

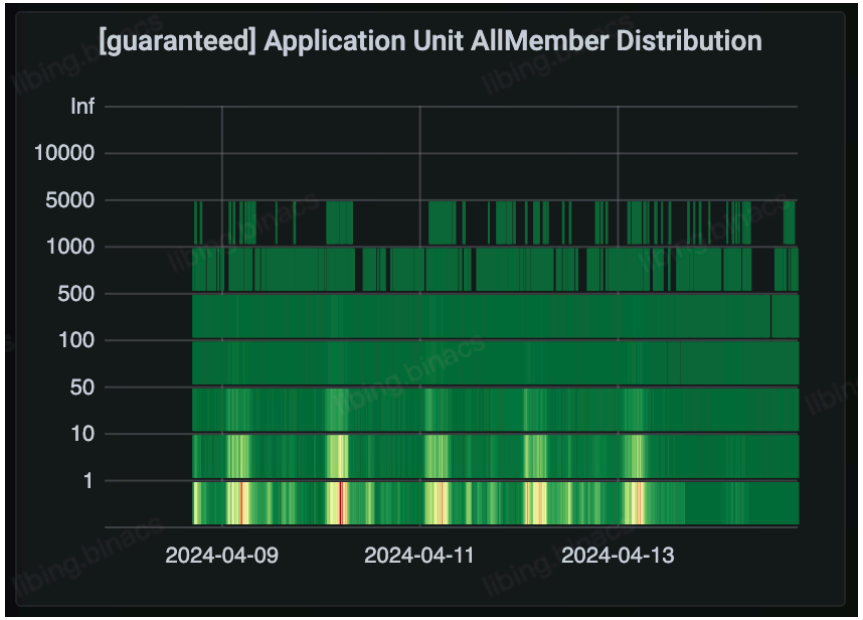


DaemonSet Locating:
30ms -> 0.3ms in a cluster of 20K+ Nodes

Support efficient scheduling through unit aggregation

Gödel Optimizations - Unit Semantic & Unit Framework

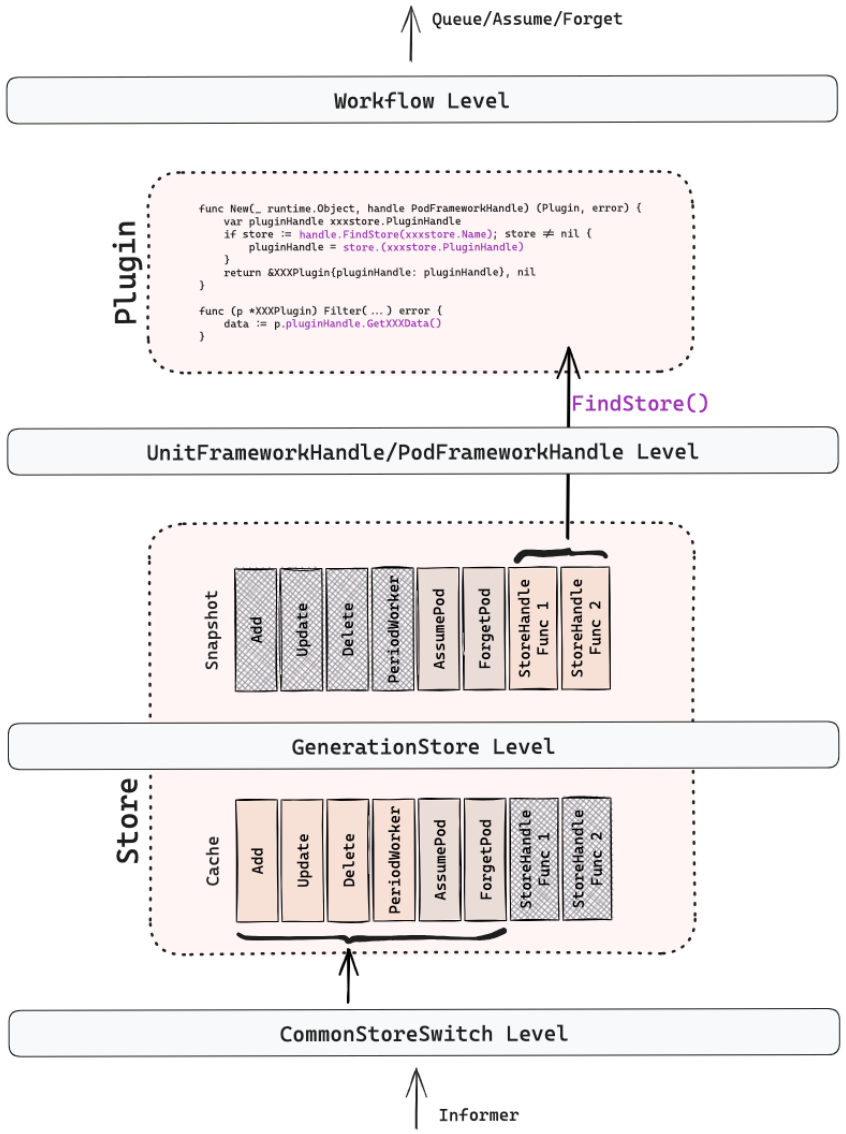




Aggregate thousands of Pods through ApplicationUnit

E2E Latency is stably suppressed to 1s in a cluster of nearly 25k Nodes

Gödel Optimizations - CommonStore Data Flow



Speed up plugin calculations by maintaining relevant data in the cache in advance

Gödel Optimizations - Achievements

TENs of times scheduling throughput

2K+ Pods/s single shard and 5K+ Pods/s multi-shards

Handle hyperscale cluster of
20k+ Nodes and 1M+ Pods
(colocation, CPU high utilization)

Gödel Scheduling System - Future Work

- Optimization of inter-component communication mechanisms
- More general batch scheduling
- More rational separation of plugin and storage implementation
- Intelligent queueing

…

THANK YOU

https://github.com/kubewharf/godel-scheduler