# Kamino: Efficient VM Allocation at Scale with Latency-Driven Cache-Aware Scheduling

**David Domingo**, Hugo Barbalho, Marco Molinaro, Kuan Lui, Abhisek Pan, David Dion, Thomas Moscriboda, Sudarsun Kannan, Ishai Menache



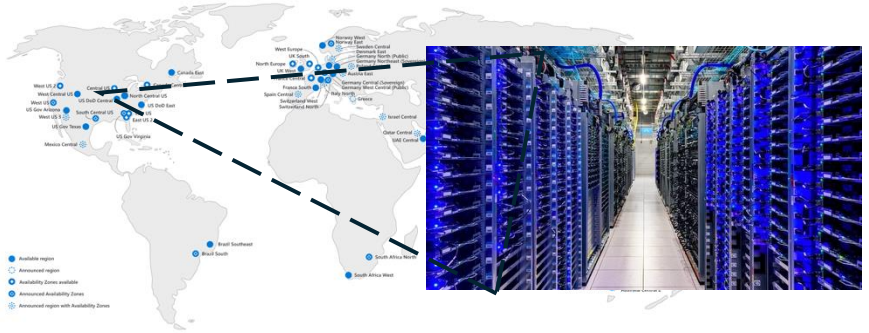






# Cloud Providers and Virtual Machines

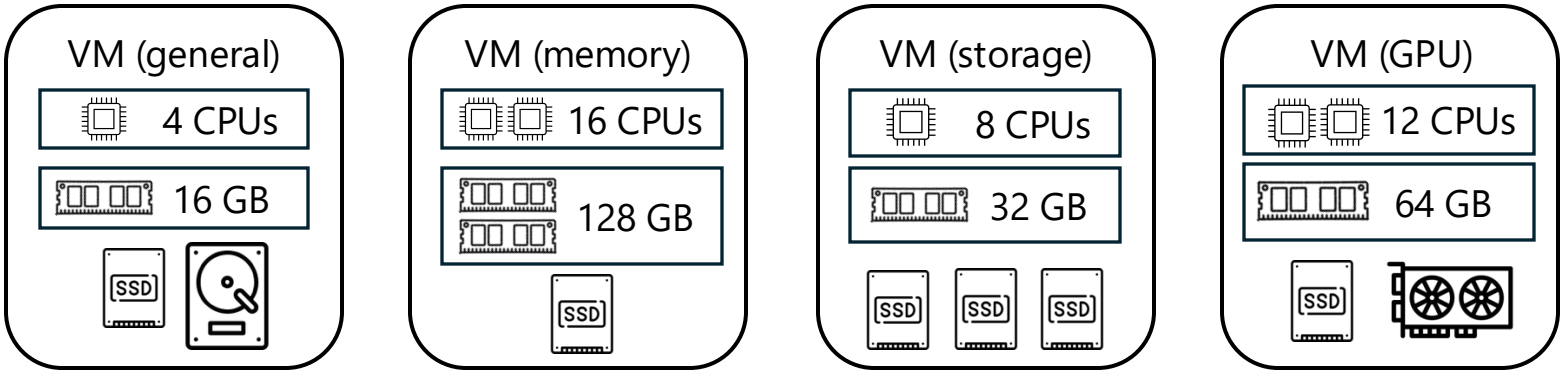
Microsoft Azure



- Cloud providers provide instant, online access to computing resources
- Made possible by operating data centers filled with readily available hardware
- Customers rent computing resources through virtual machines (VMs)
- VMs are allocated on physical machines accessible over the internet
- Enables flexible costs and scaling for customers

# VM Allocation Complexity

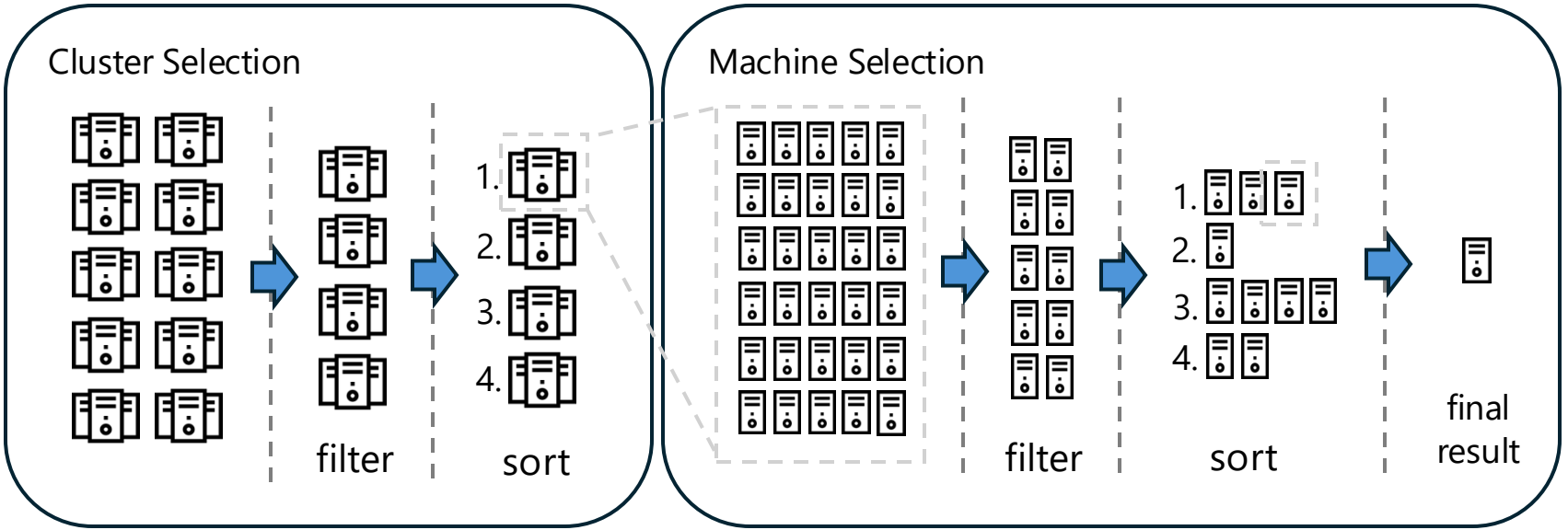
- Challenge: Placing VMs over 100k+ machines over multiple clusters
- Need to handle large variety of VM configurations
  - Varying amounts of CPU, memory, storage, networking, accelerators, location preferences




- Need to consider platform efficiency goals (eg. induce machine packing)

# VM Request Resolution

- Resolving VM placement is computationally intensive
  - Set manipulation to filter inventory, sorting to accommodate preferences
- Example of placement resolution:



# Managing VM Allocation Performance

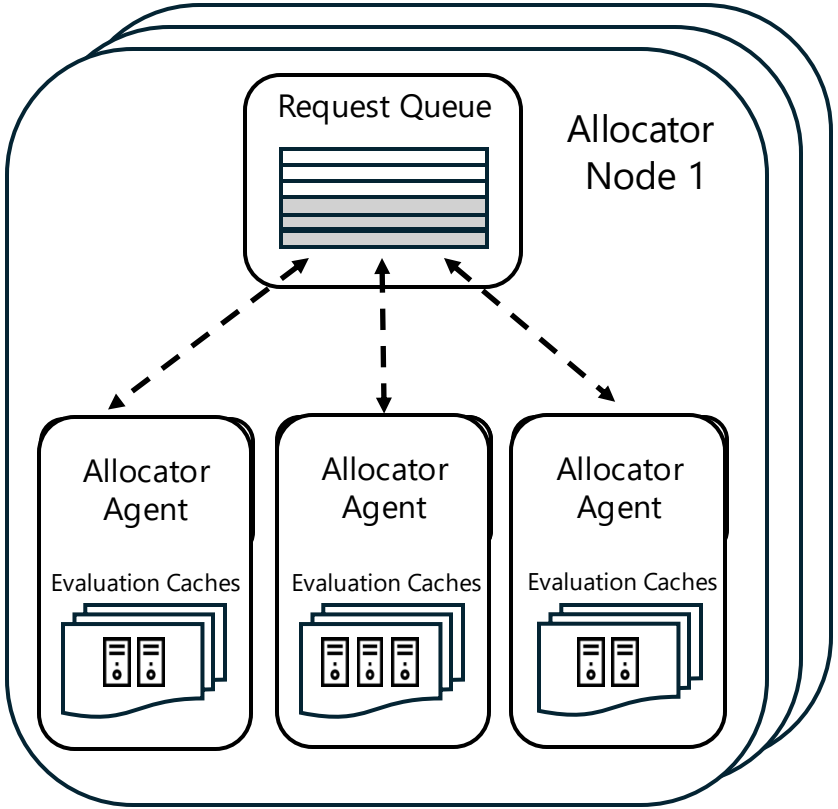
- Cloud providers must scale VM allocation to accommodate demand and expansion
- Need high VM request throughput
  - Many customer VM requests at any moment
- Need low VM request latencies
  - Enable fast deployments for customers
- Practical approaches to scaling allocations
  - Omega (EuroSys '13), Protean (OSDI '20), Twine (OSDI '20), VMWare DRS
- Efforts to maintain operational efficiency and service quality at scale
  - VM lifetime predictions (MLSys '23), Kerveros (OSDI '23)

Focus: Managing VM request latencies for Protean

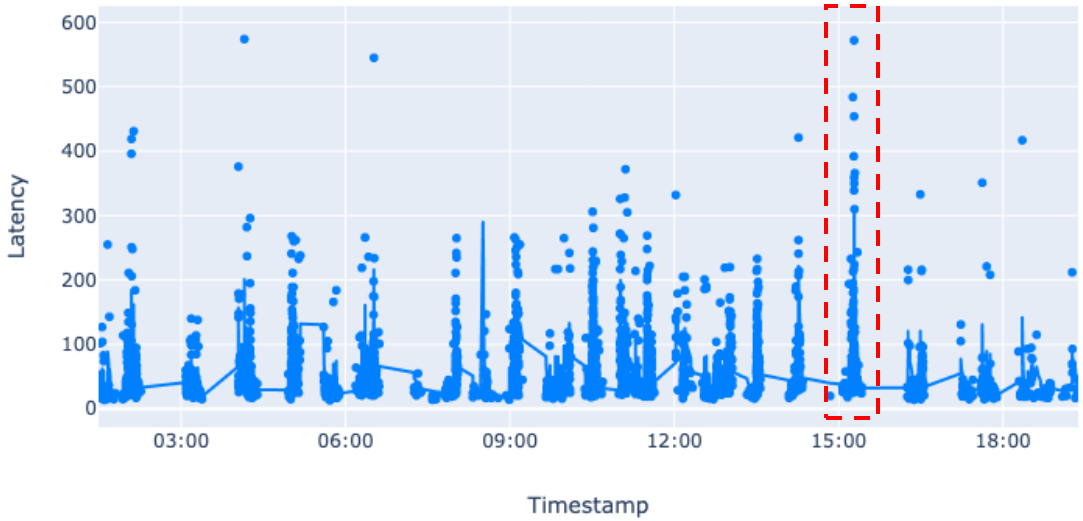
# Protean: Scaling VM Allocation

- **Protean** – Microsoft Azure's VM Allocator
- Multiple allocator agents
  - Parallelize VM request resolution
- Allocator agents pull from global queue
  - Effectively "load balance" requests
- Further scale with additional nodes
  - Scale past a single machine
- ***Request evaluation caches***
  - Cached sets of feasible inventory
  - Reuse results for future similar requests

Achieves high request throughput +
Latencies on order of tens of milliseconds



# Problem: Latencies Under Bursts

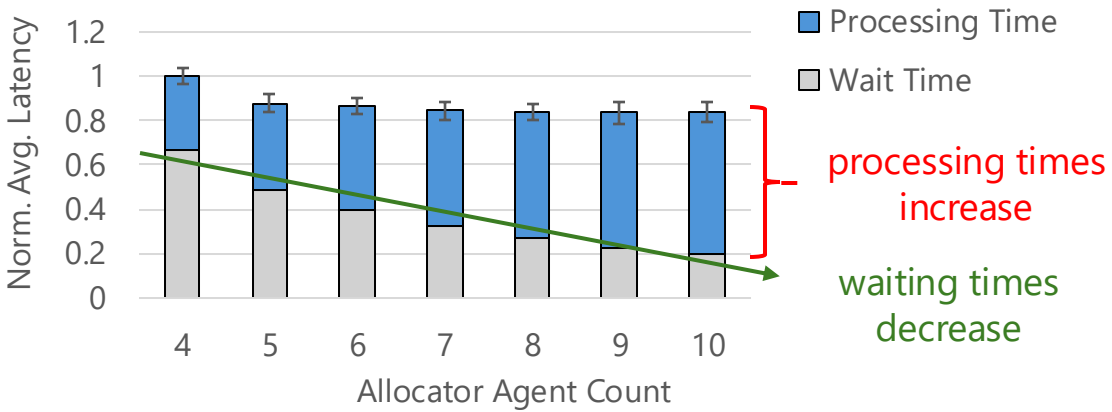



- VM request are bursty due to batch deployment, autoscaling, and capacity reservations
- High average and tail latencies suffered around bursts

How to improve latencies under bursts?

# Attempt #1: Increasing Parallelism

- Reduce latencies by increasing allocator agent count within a node




- Limited latency improvements
  - Degraded processing times cancel out improved wait times

Why do processing times increase?

# Memory Scaling Bottleneck

Resource usage of 4 popular allocator nodes:

| Node | Avg CPU Usage | Avg Memory Usage |
|---|---|---|
| Node 1 | 50% | 92% |
| Node 2 | 61% | 92% |
| Node 3 | 74% | 90% |
| Node 4 | 39% | 90% |

high memory use

- Allocator agents constrained by memory
- Result: scaling reduces memory efficiency and caching performance

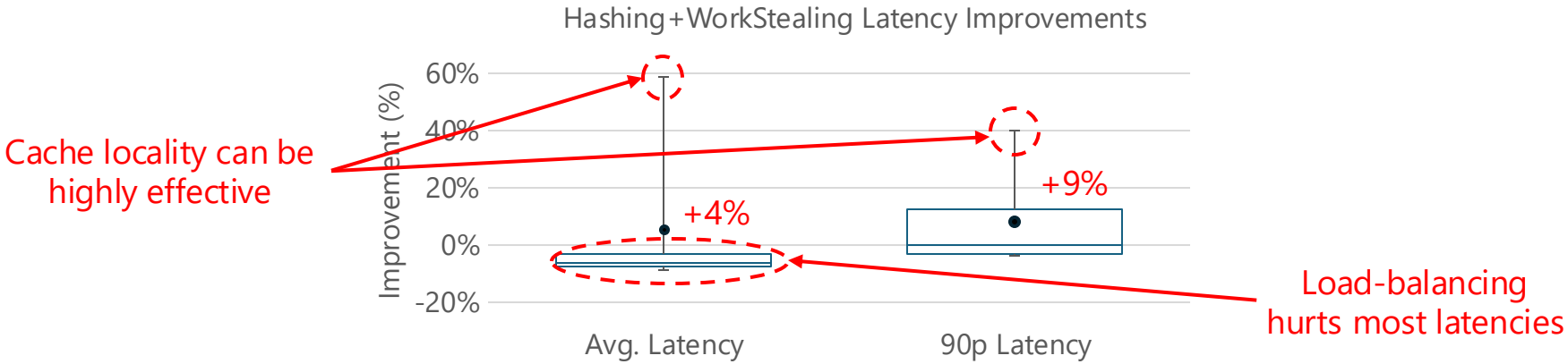
Increased cache duplication → Reduced ability to cache → Higher cache misses → Increased processing times

- Scaling without careful consideration would waste resources

Can memory efficiency be improved?

# Attempt #2: Maximizing Cache Locality

- Improve memory efficiency and latencies by maximizing cache locality
  - Potential load-imbalance must also be handled
- Induce locality with request hashing, load-balance with work stealing

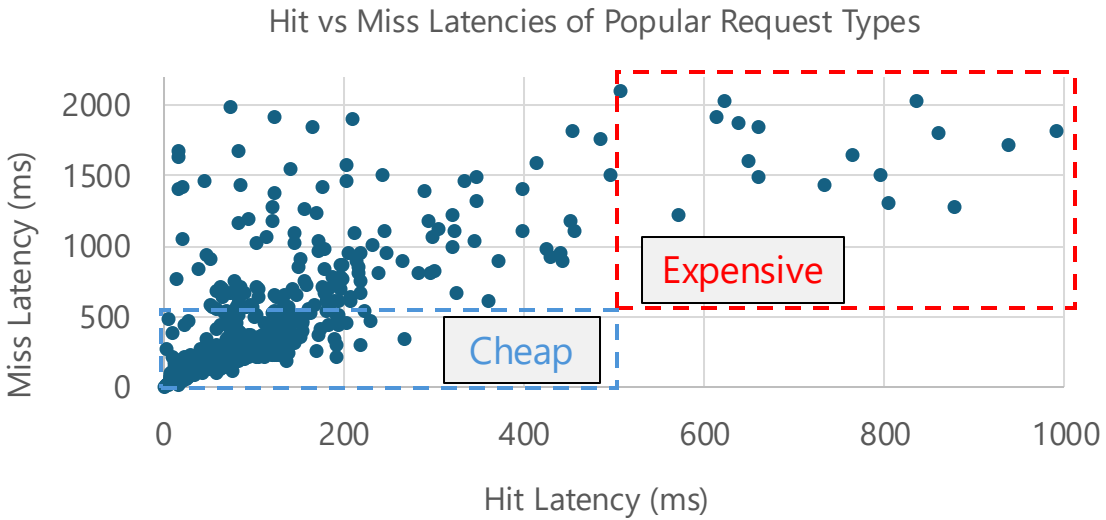



- Cache reuse and load-balancing conflict despite similar goal of lowering latencies

What makes cache reuse with load balancing challenging?

# Complicated Latency Variability

differences in constraints and computation
↓
varied latencies




hit times can be greater than miss times

- Focus on cache-hits causes unnecessarily delays for cheap requests
- Naïve load balancing causes expensive requests to miss, impacting overall latency

Need cache-aware, latency-driven scheduling

# Our Solution: Kamino

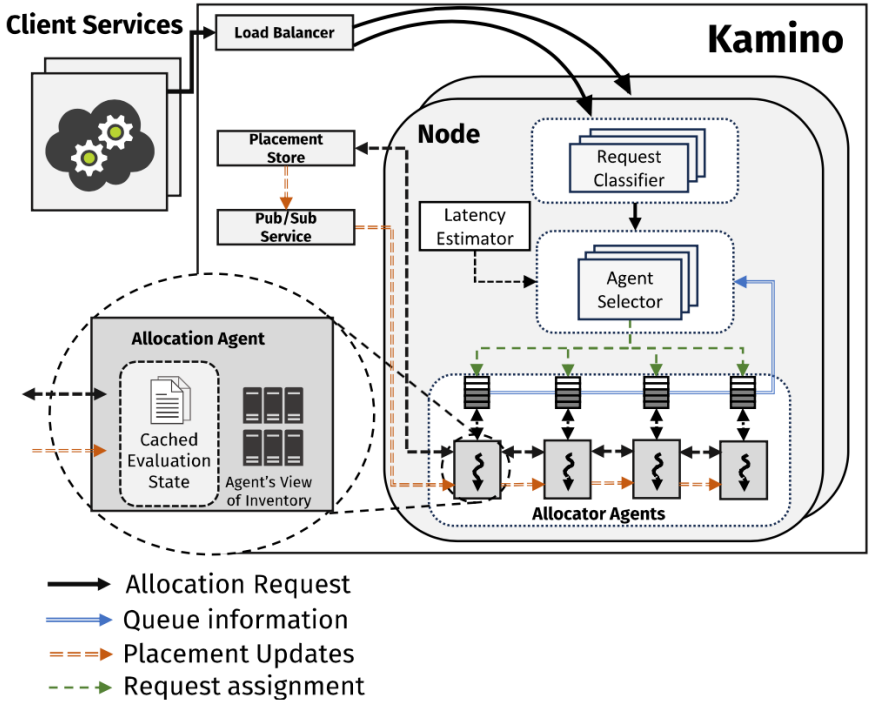
- Improve VM request latencies with…

## Cache-Aware, Latency-Driven Request Scheduling

- Considers cache state to improve cache locality
- Considers variable latencies to minimize latency impact from load balancing
- Able to reconcile cache-reuse with load balancing
  - Minimizes cache pollution and excessive delays
- Improves latencies while also improving memory usage

# Kamino Architecture

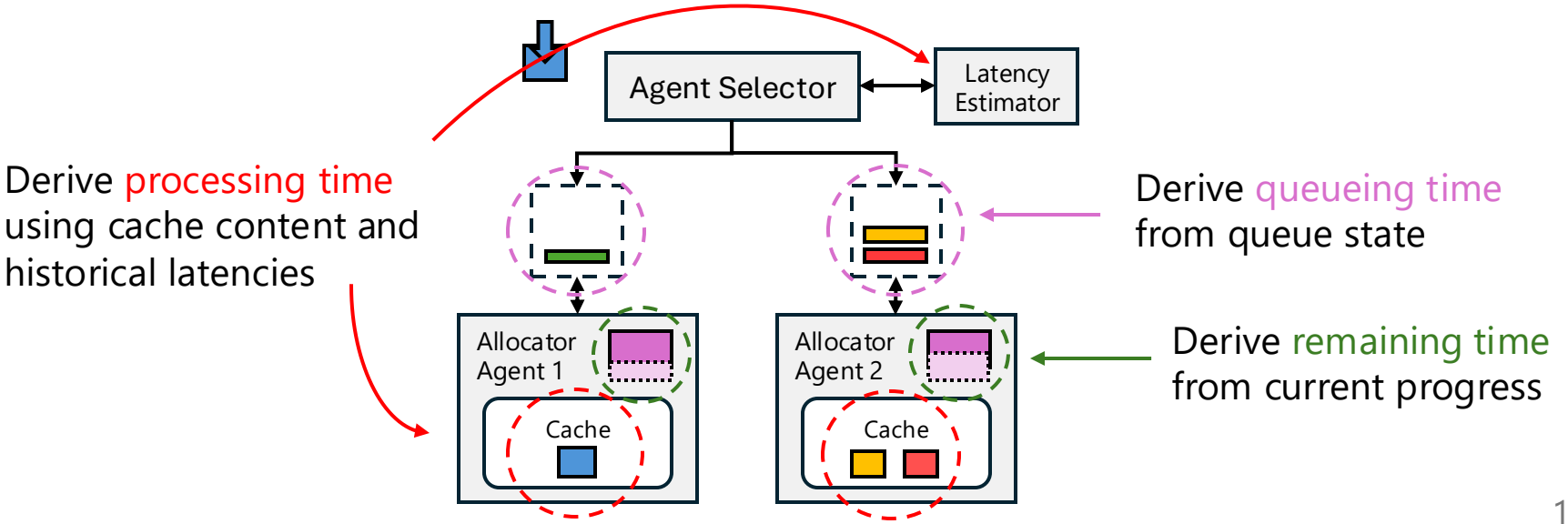
- Multi-Queue Design
  - Allows reordering of requests
  - Enables ability to induce cache affinity
- Request Classifiers
  - Parallelizes request precomputation
- Allocator Agent Selectors
  - Schedules requests across agents in parallel
  - Employs LatCache Algorithm
- Latency Estimator
  - Track historical request latencies



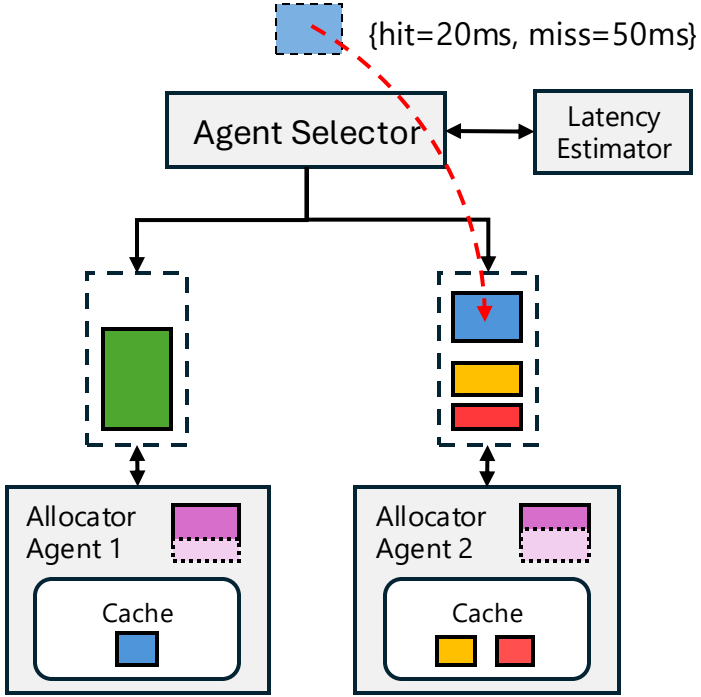
# LatCache Scheduling

- Key Idea: Schedule requests where latency is minimized
  - Ideal: Load balance away from cached result only if expected latency is better
- Schedule based on latency-driven model:

$$ExpectedTime(a_i, r_x) = ProcessingTime(r_x) + QueueingTime(a_i) + RemainingTime(a_i)$$



# LatCache In Action




Estimate Agent 1 Latency:

Processing Time = 20ms (*hit*)

Queueing Time = 60ms

Remaining Time = 20ms

Estimated Latency = 100ms

Estimate Agent 2 Latency:

Processing Time = 50ms (*miss*)

Queueing Time = 30ms

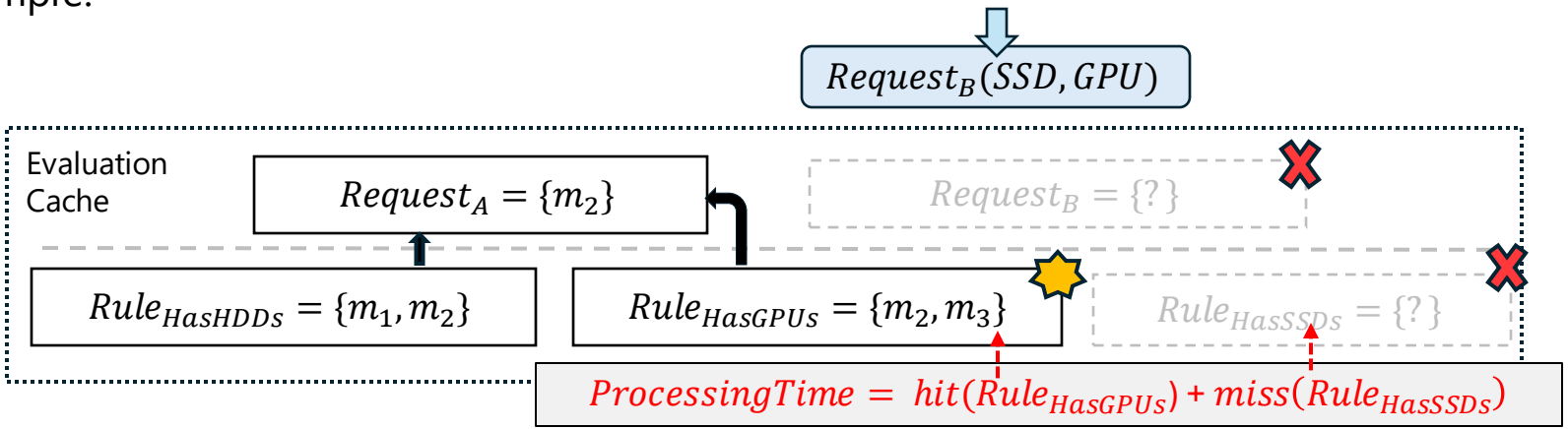
Remaining Time = 10ms

Estimated Latency = 90ms

Decision: Schedule to allocator agent 2 to minimize latency despite potential hit on allocator agent 1

# Handling Hierarchical Caches

- ***Top-level Request Caches*** store results for future identical requests
- ***Lower-level Rule Caches*** store intermediate results for subsets of constraints
- Example:




- LatCache-Rule variant tracks rule latencies to consider partial hits
- Processing time set to sum of expected rule latencies upon top-level miss

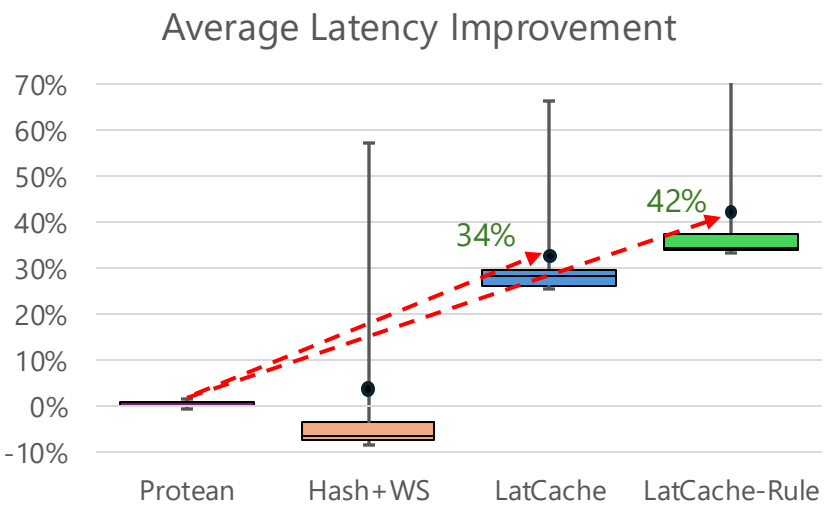
# Evaluation

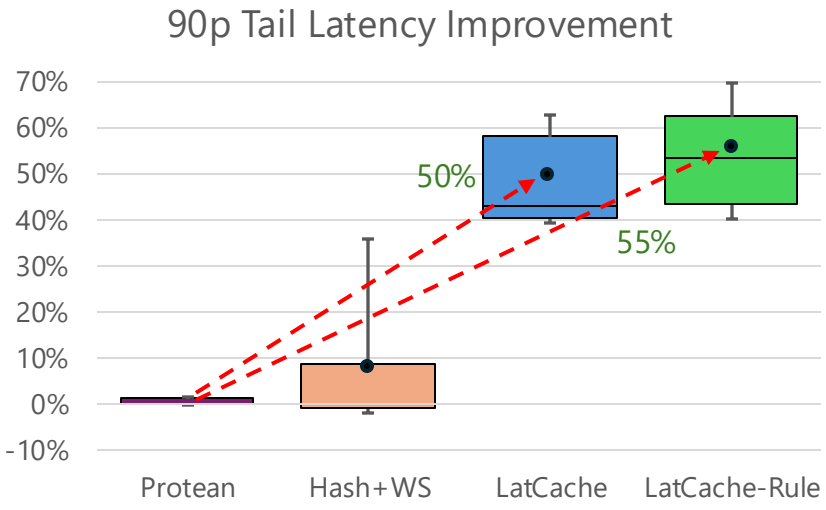
- Simulate performance against 6 real traces from high-traffic nodes across several zones
- Compare the following approaches:

| Algorithms | Description |
|---|---|
| **Protean** | Original shared-queue architecture for load-balancing |
| **Hash+WS** | Hash-based scheduling with work stealing for load balancing |
| **LatCache** | Kamino's latency-driven, cache-aware scheduling |
| **LatCache-Rule** | Kamino's LatCache variant with more precise rule latencies |

- Measure average and tail latency, memory usage, and cache hit rates

# Improvements



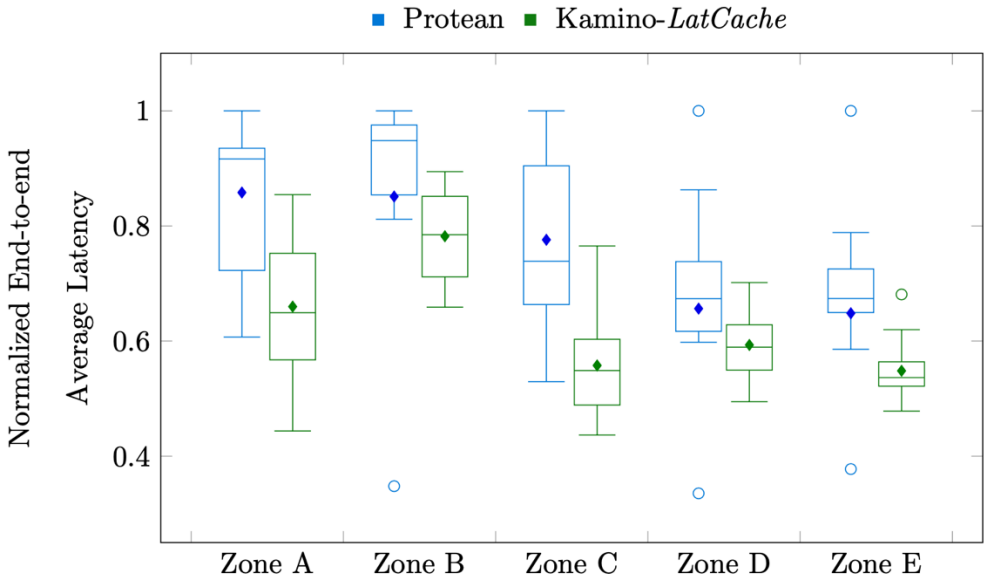




- Latency-awareness mitigates expensive misses and unnecessary delays
- Cache-awareness improves hit rates by ~13% and reduces memory by ~15%

# Evaluation: Production

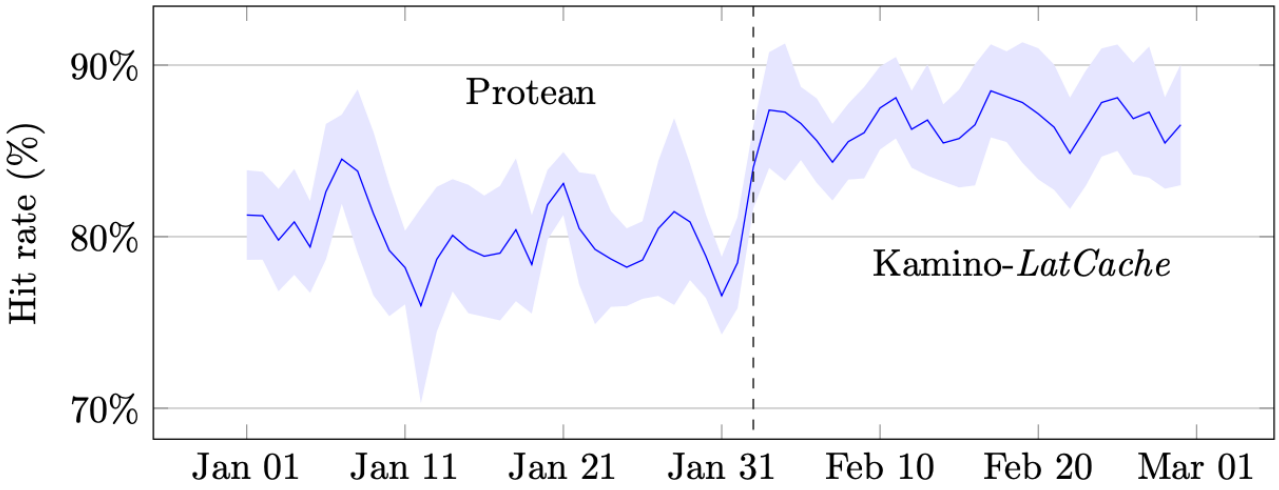
- How does Kamino's LatCache perform in practice?
- Deploy Kamino with LatCache across 5 production zones
- Compare performance before and after deployment
- Measure average latency, cache hit rates, and memory usage

# Improvements in Production: Latencies




- Improves average latencies by ~21% improves 90p latency and ~12% (not shown)
- Kamino's local-level scheduling optimization translates to zone-wide improvements

# Improvements in Production: Hit Rates & Memory




- Reduces cache misses by ~33%, improving cache hits by ~8%
- Reduces memory usage by ~17% on average across nodes (not shown)

# Conclusion

- Kamino – Cache-Aware, Latency-Driven VM request Scheduling
- Cache-Aware Latency Estimation
  - Adapts to variable request type latencies
  - Prioritizes cache locality for expensive requests
- Latency-Driven Scheduling
  - Reroutes requests only when beneficial
  - Reconciles cache-reuse with load balancing
- Improves overall latency performance and memory usage
- More efficient scaling against available hardware







