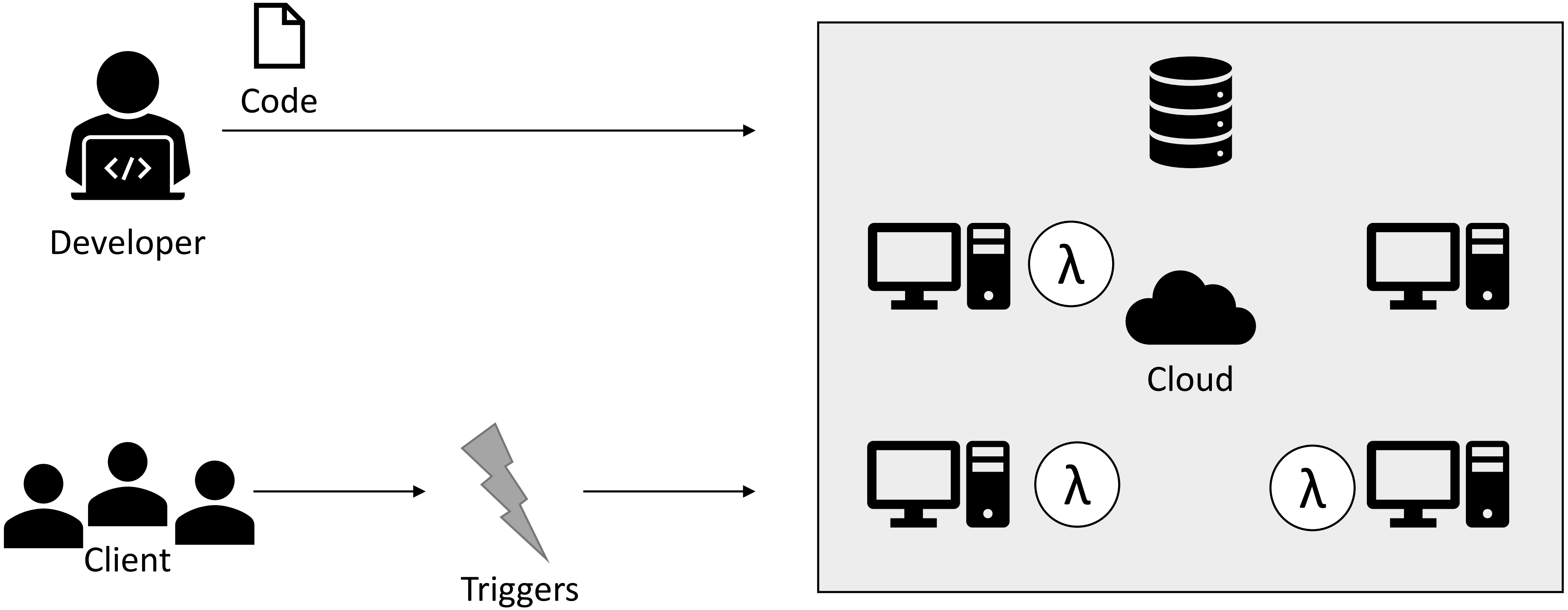
Memory Deduplication for Serverless Computing with Medes

Divyanshu Saxena, Tao Ji, Arjun Singhvi, Junaid Khalid, Aditya Akella

The University of Texas at Austin
Computer Science
College of Natural Sciences

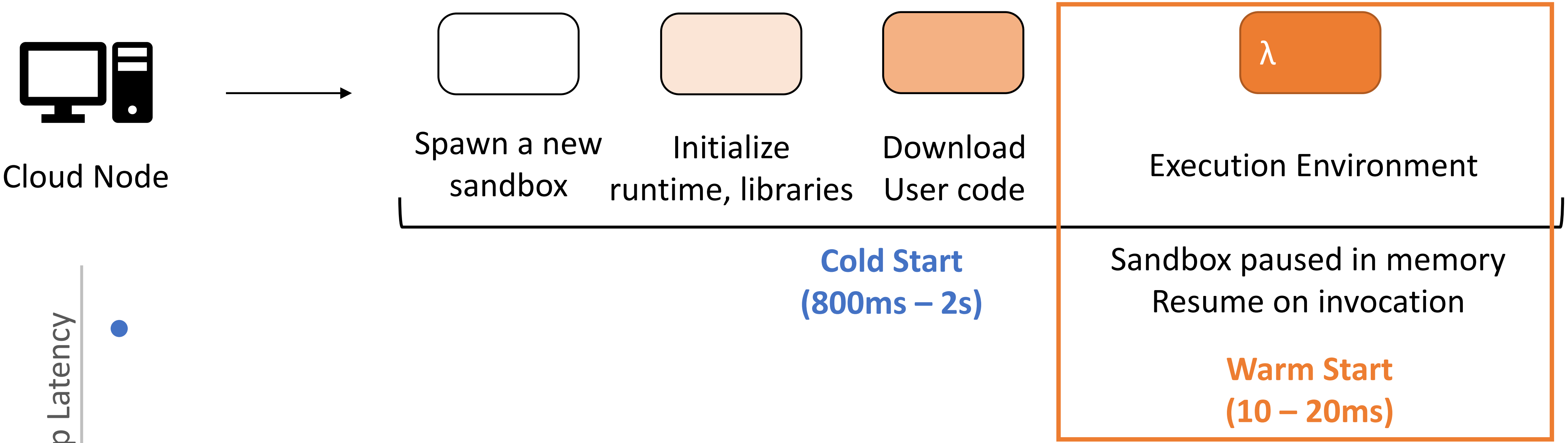
Computer Sciences
School of Computer, Data & Information Sciences

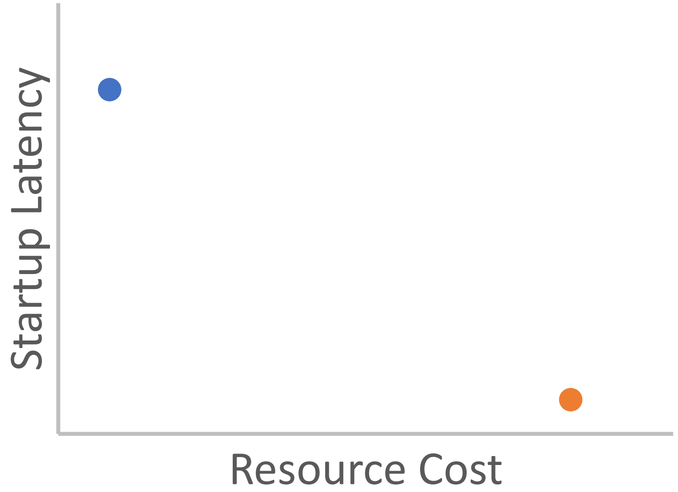
Serverless – Introduction



Serverless – Deep Dive

What exactly happens when a function is invoked on a serverless platform?





Keeping warm sandboxes is prohibitively memory expensive!

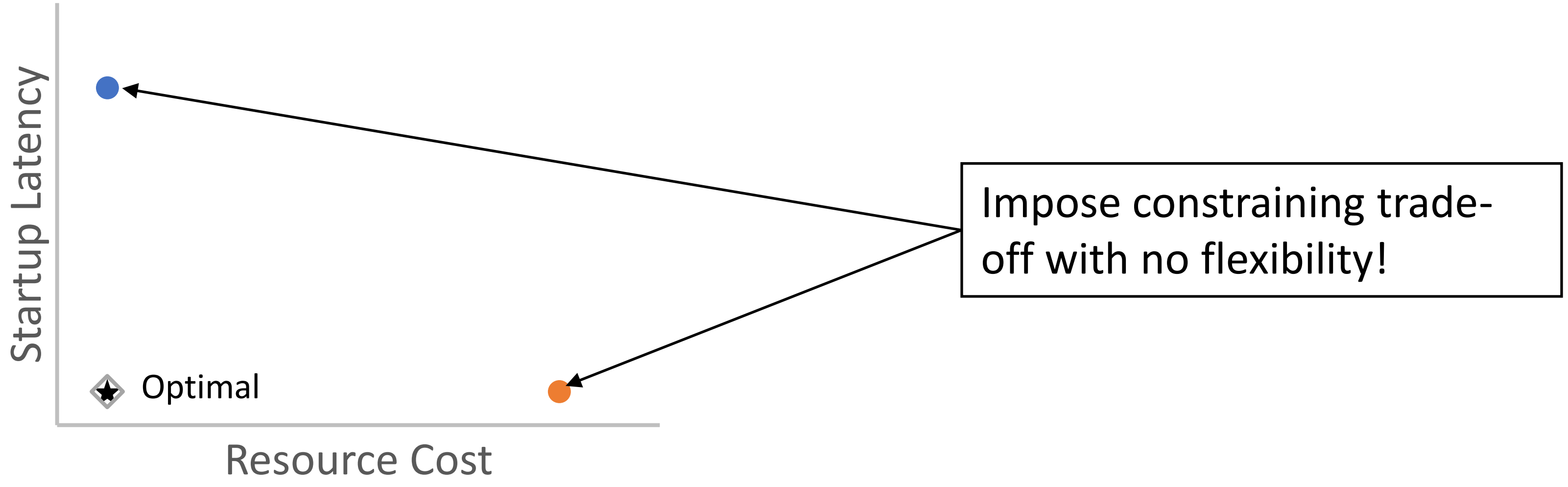
Serverless – Existing Techniques

- Fixed (AWS) or Adaptive (Hybrid Histogram$_{(ATC20)}$) keep-alive durations
- Provision sandbox resources in anticipation of future invocations $_{(ATC20,\ SoCC20)}$

Erratic serverless workloads are difficult to generalize!

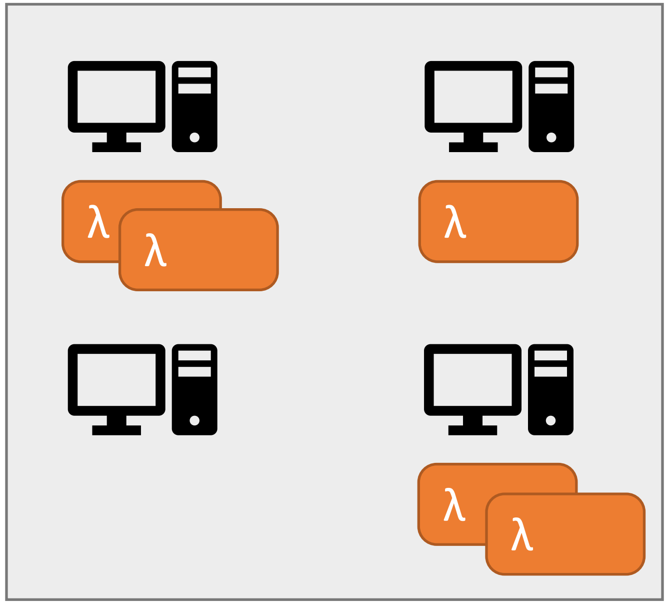
- Snapshot-restore techniques (Catalyzer$_{(ASPLOS20)}$, SEUSS$_{(EuroSys20)}$)

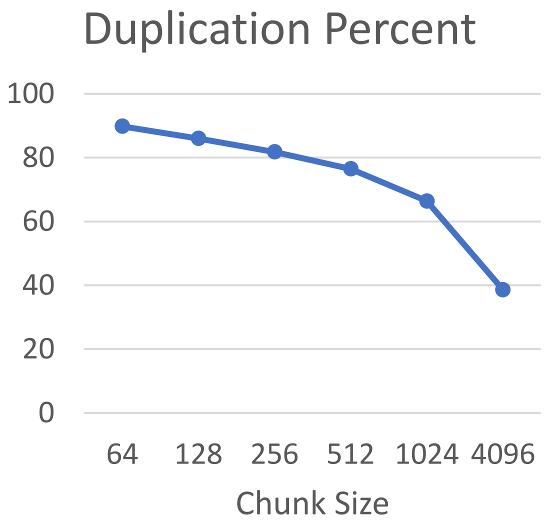
Cold starts still significantly slower than warm starts, or sacrifice isolation

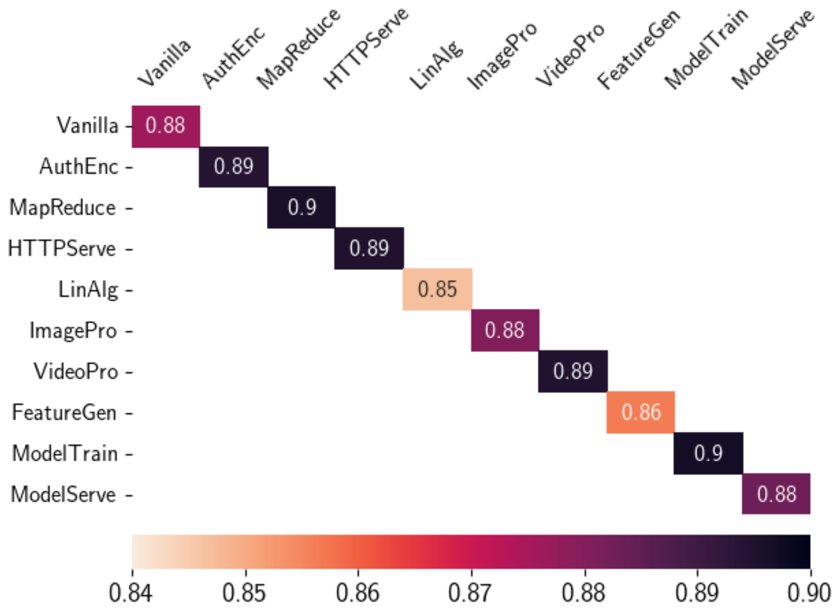


Key Insight

At any point, several "warm" sandboxes are in memory

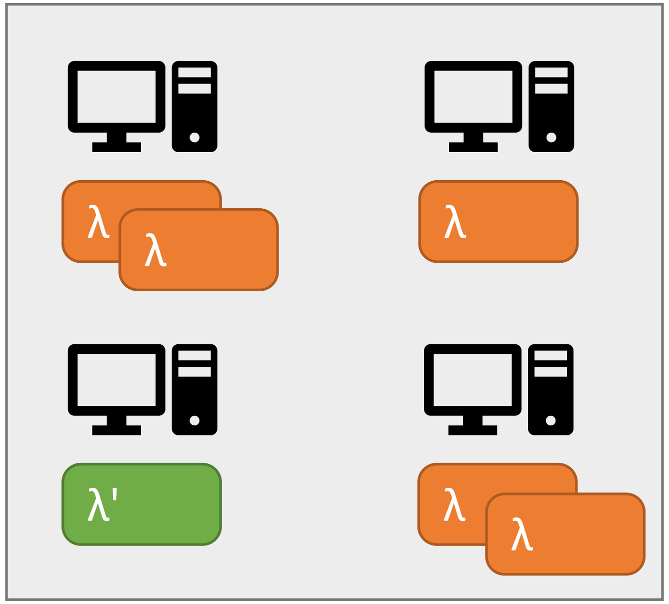


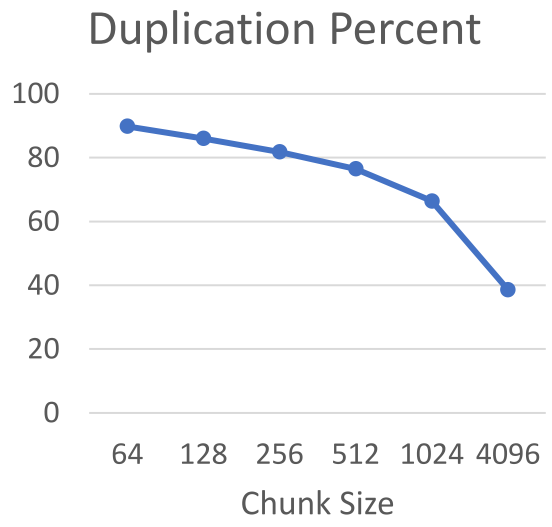


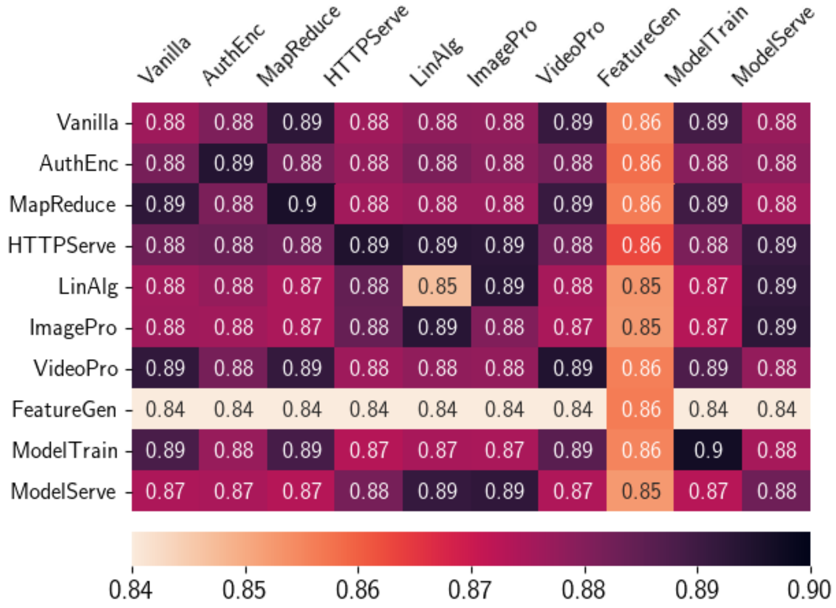


Key Insight

At any point, several “warm” sandboxes are in memory



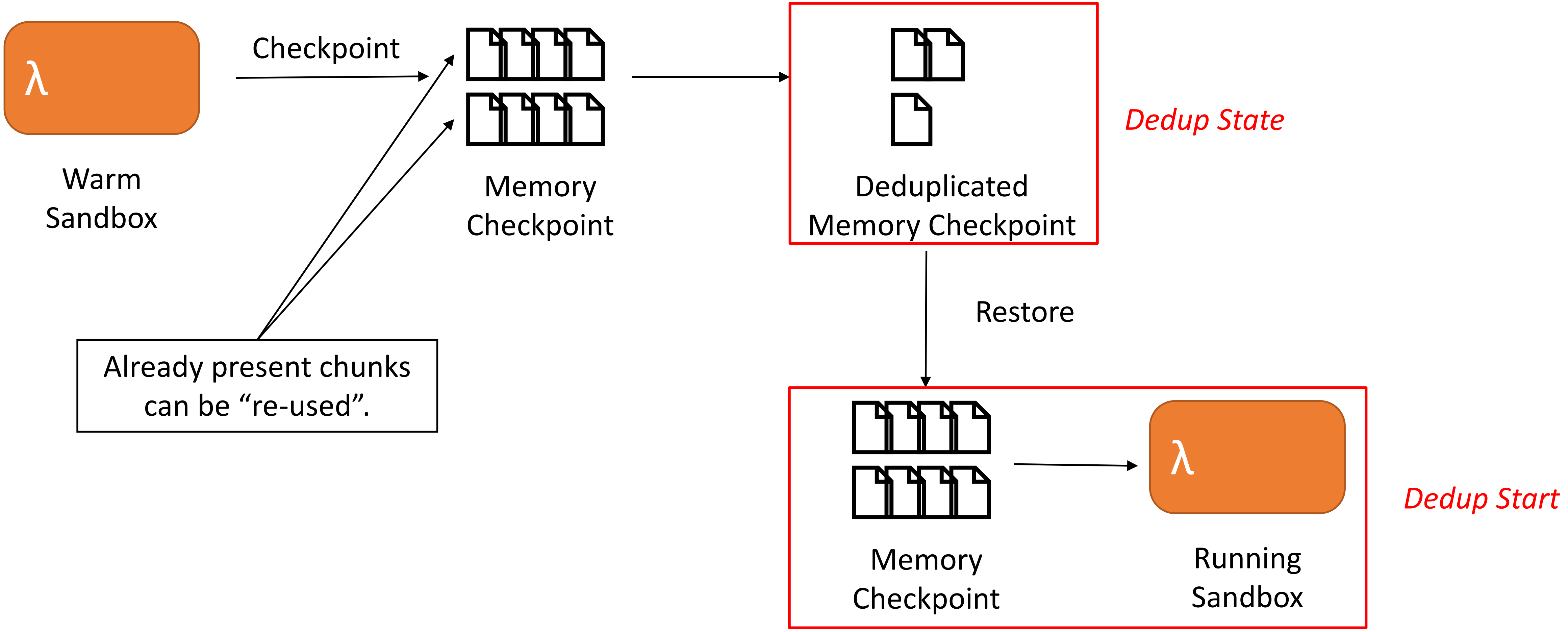




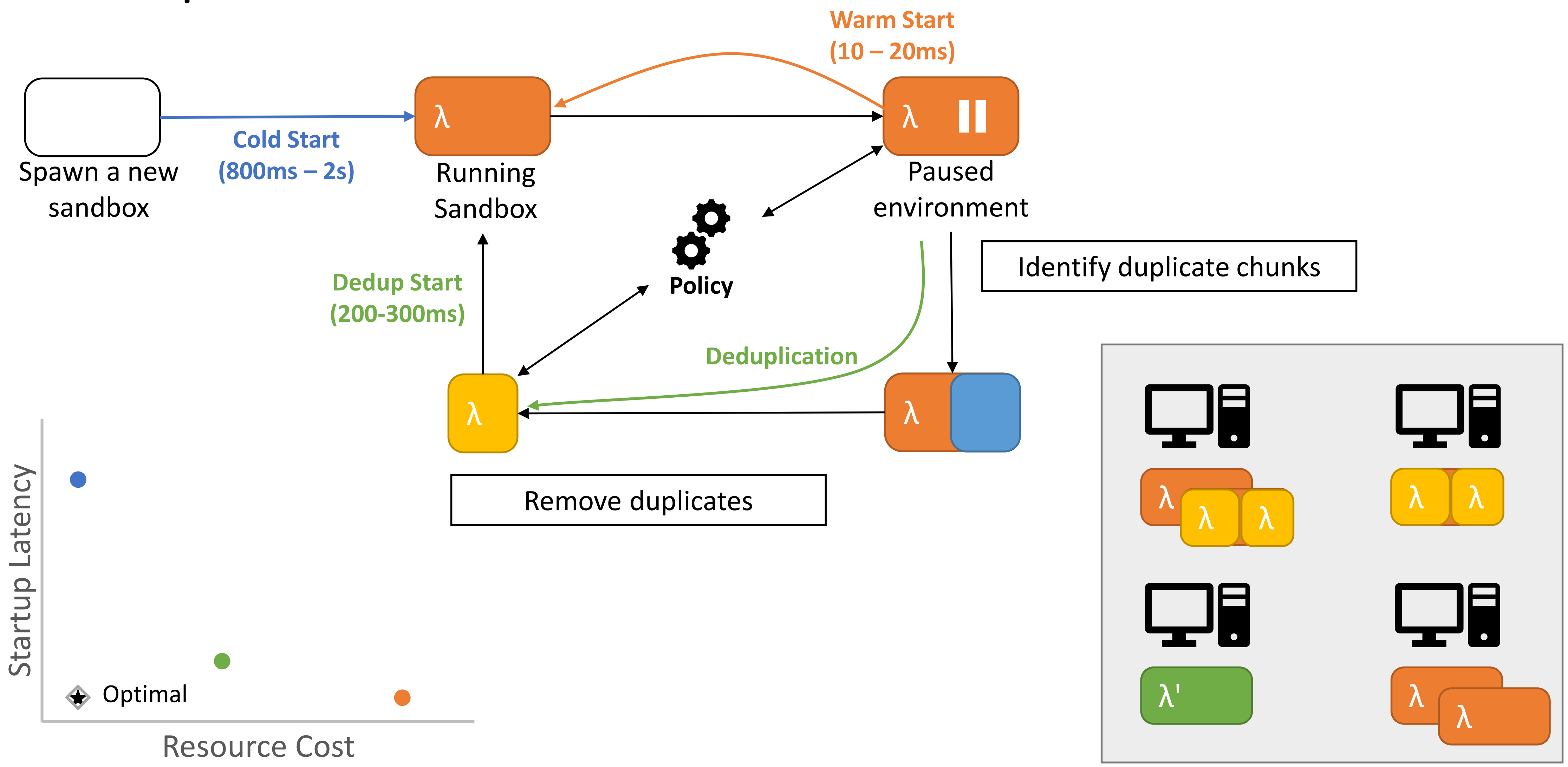
Duplication due to common runtime and libraries

Memory redundancy can be exploited to reduce sandbox footprints

Reusable Sandbox Chunks



Dedup Starts



Medes – System Design

Design Goals

Exploit redundancy across nodes

Efficient and scalable redundancy identification

Fast Dedup sandbox restores

Flexibility in navigating trade-offs

Medes – System Design

Design Goals

Exploit redundancy across nodes

Efficient and scalable redundancy identification

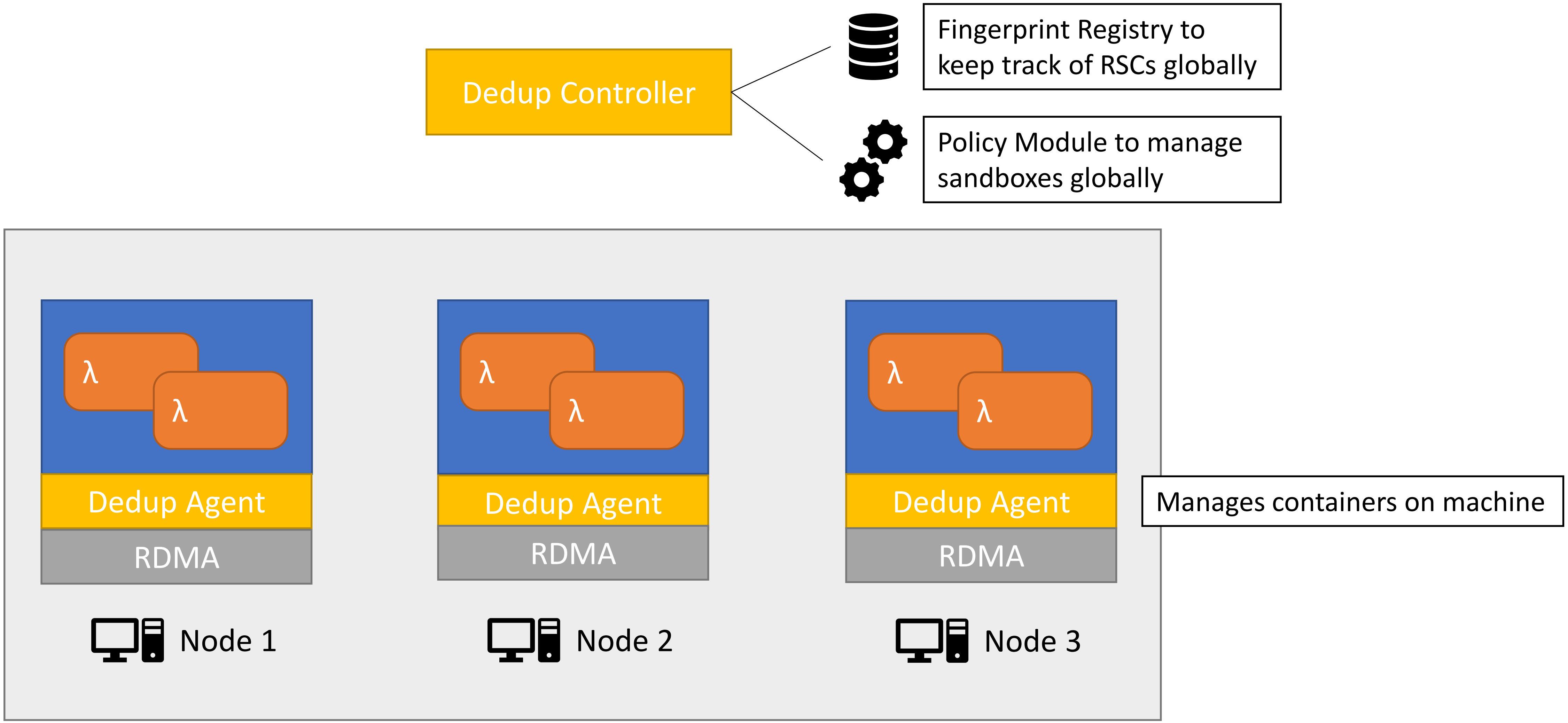
Fast Dedup sandbox restores

Flexibility in navigating trade-offs

Challenges

Maintain a global view of the cluster

Medes System Design – Global View



Medes – System Design

Design Goals

Exploit redundancy across nodes

Efficient and scalable redundancy identification

Fast Dedup sandbox restores

Flexibility in navigating trade-offs

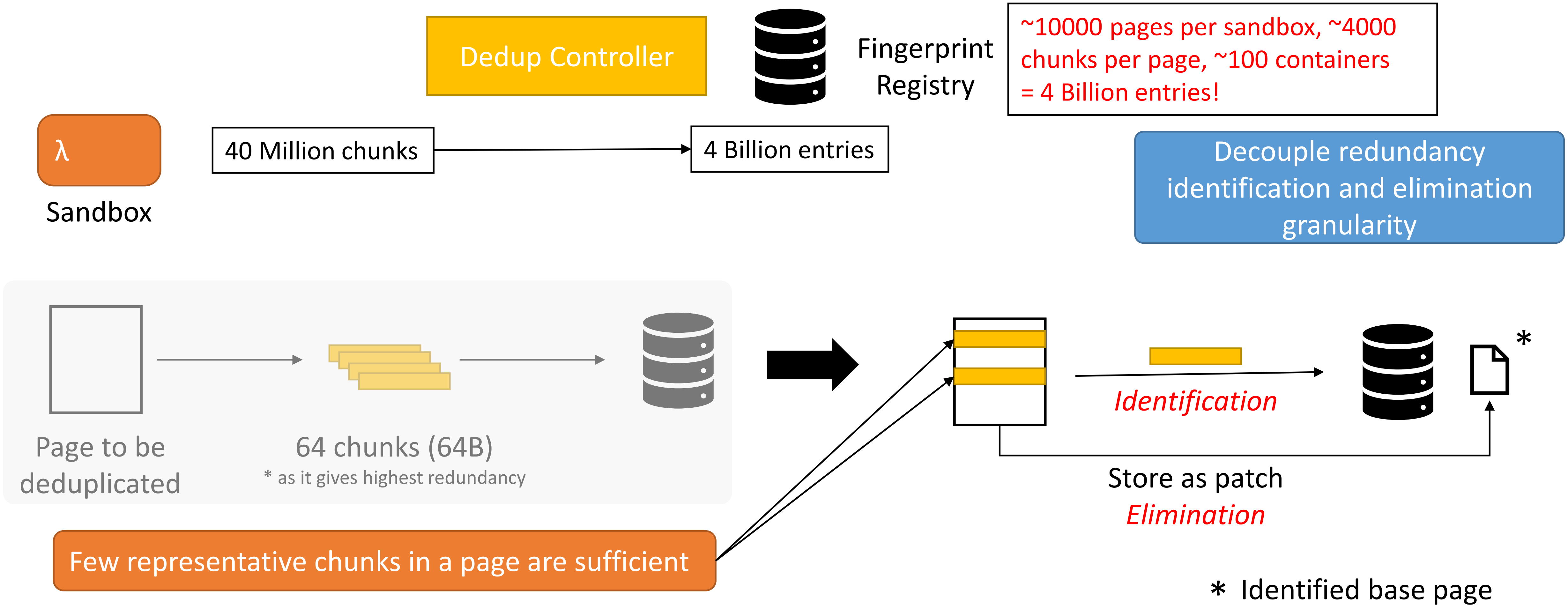
Challenges

Maintain global view of the cluster

Two Tier Architecture

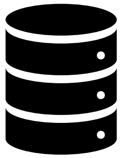
- Needs to be computationally lightweight
- And have low metadata overhead

Efficient Redundancy Identification

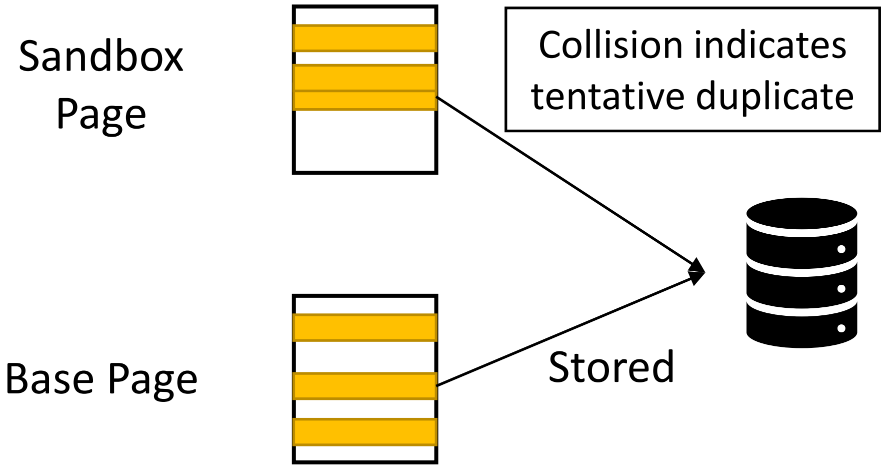


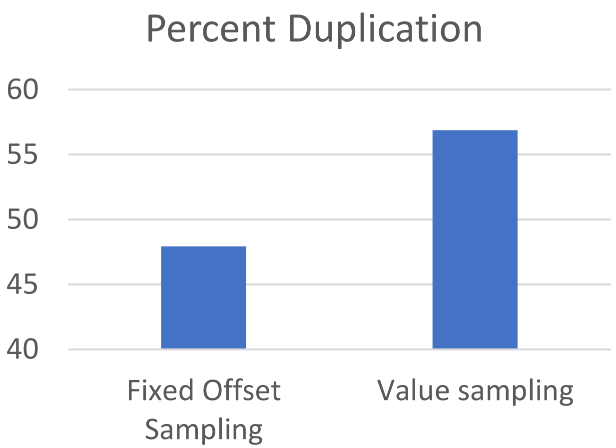
Efficient Redundancy Identification

Dedup Controller



Fingerprint Registry

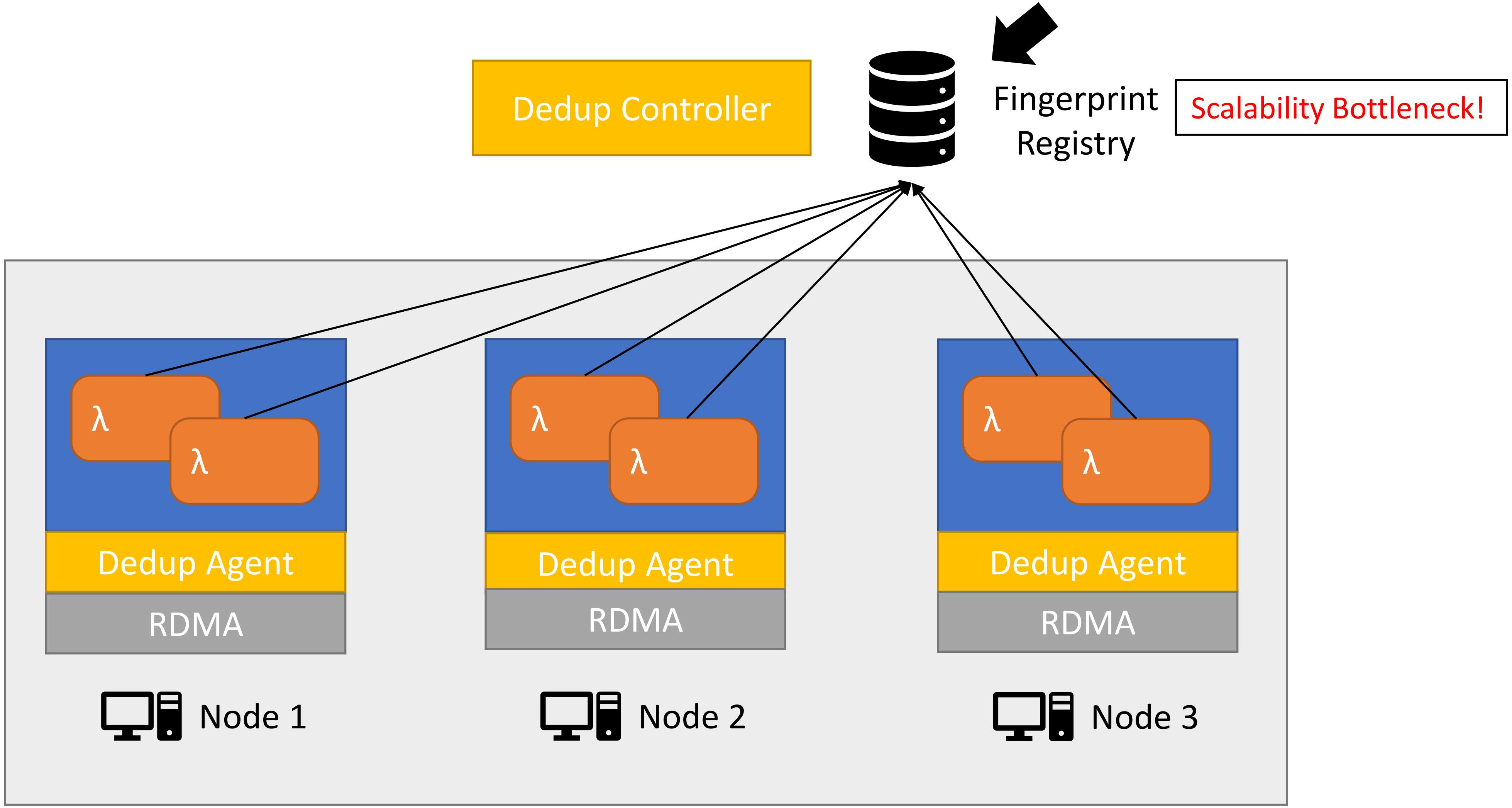




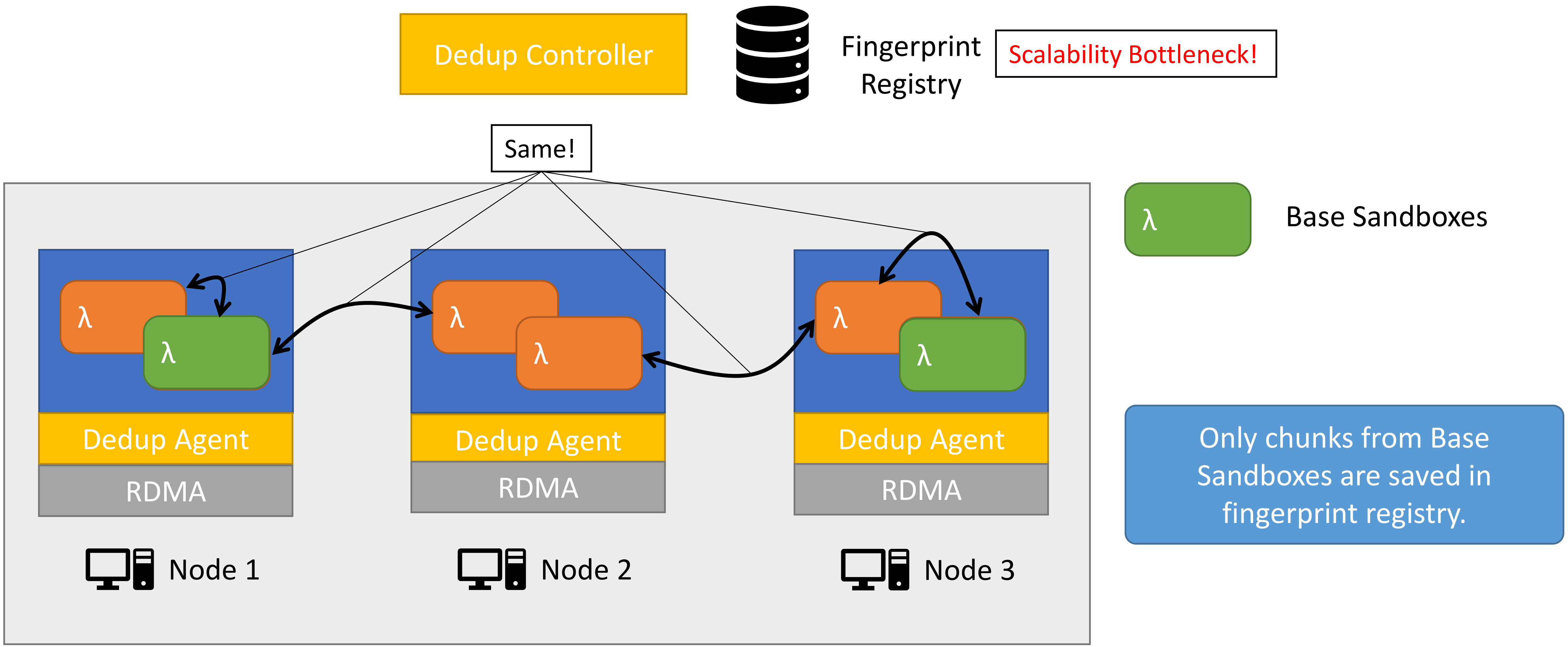
Value sampled fingerprinting

Use value sampling – choose only those chunks that have the last byte to be a particular marker.

Scalable Redundancy Identification



Scalable Redundancy Identification



Medes – System Design

Design Goals

Exploit redundancy across nodes

Efficient and scalable redundancy identification

Fast Dedup sandbox restores

Flexibility in navigating trade-offs

Challenges

Maintain global view of the cluster

Two Tier Architecture

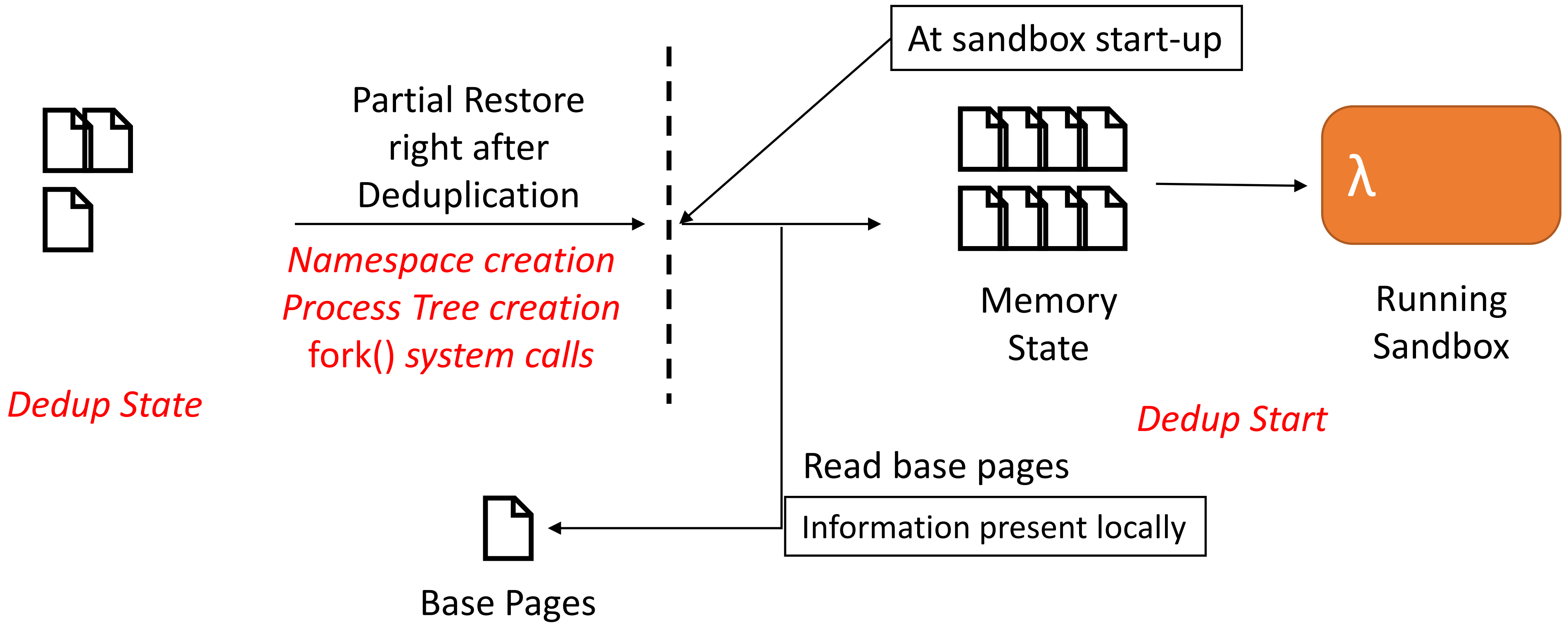
- Needs to be computationally lightweight
- And have low metadata overhead

Value sampled fingerprints, decouple redundancy granularity and select base containers

Dedup Starts should be much faster than cold starts

Fast Dedup Sandbox Restores

- Time consuming steps of sandbox restore performed right at deduplication [Catalyzer$_{(ASPLOS20)}$]



Medes – System Design

Design Goals

Exploit redundancy across nodes

Efficient and scalable redundancy identification

Fast Dedup sandbox restores

Flexibility in navigating trade-offs

Challenges

Maintain global view of the cluster

Two Tier Architecture

- Needs to be computationally lightweight
- And have low metadata overhead

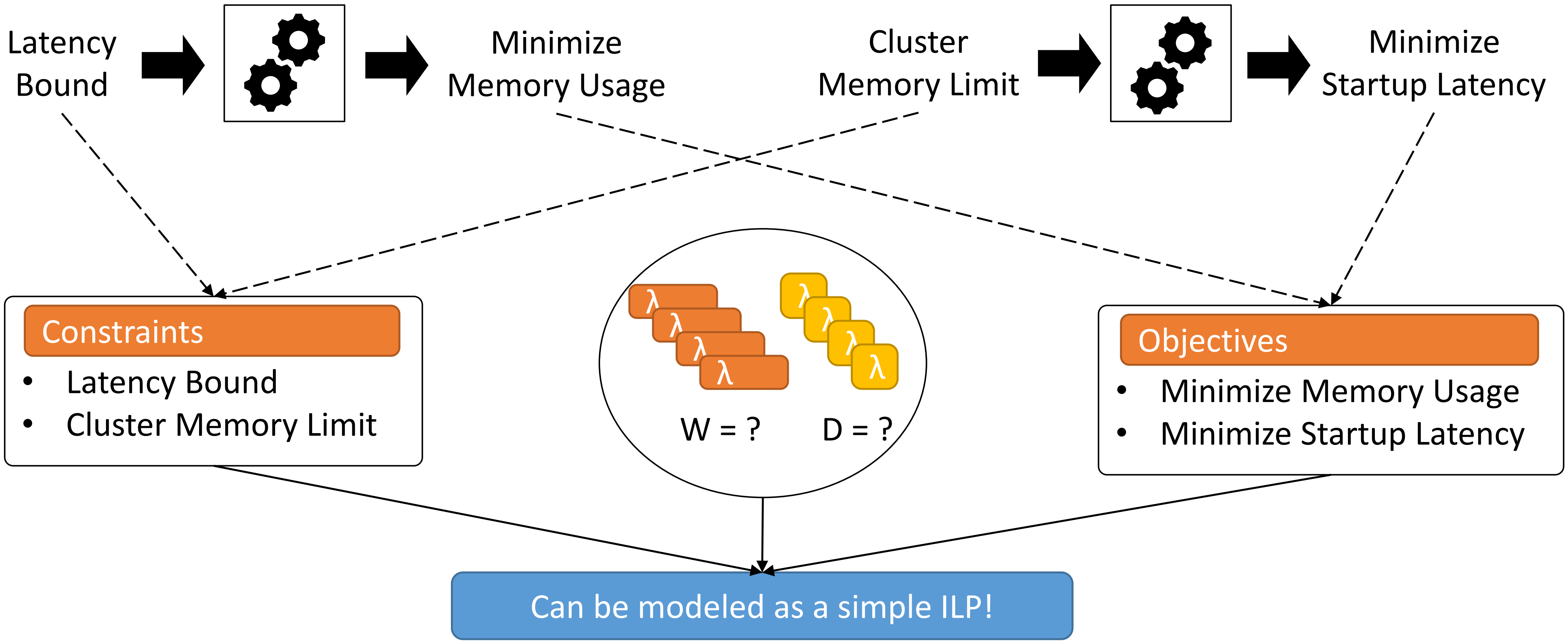
Value sampled fingerprints, decouple redundancy granularity and select base containers

Dedup Starts should be much faster than cold starts

Construct policy that can deduplicate containers as per requirement

Medes – Policy

- Allow operators run their platform in two configurations:



Evaluation

- Can Medes provide improved end-to-end latencies?
- How does Medes perform under memory pressure situations?
- Can Medes help reduce the overall memory footprints?
- Can tuning fixed keep-alive policies achieve the same performance-memory tradeoffs as Medes?

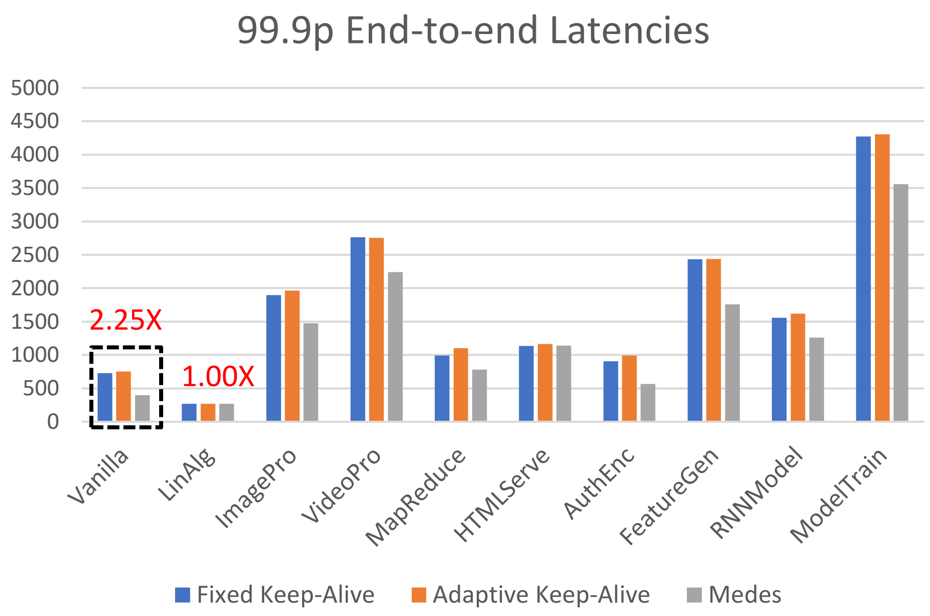
Evaluation - Setup

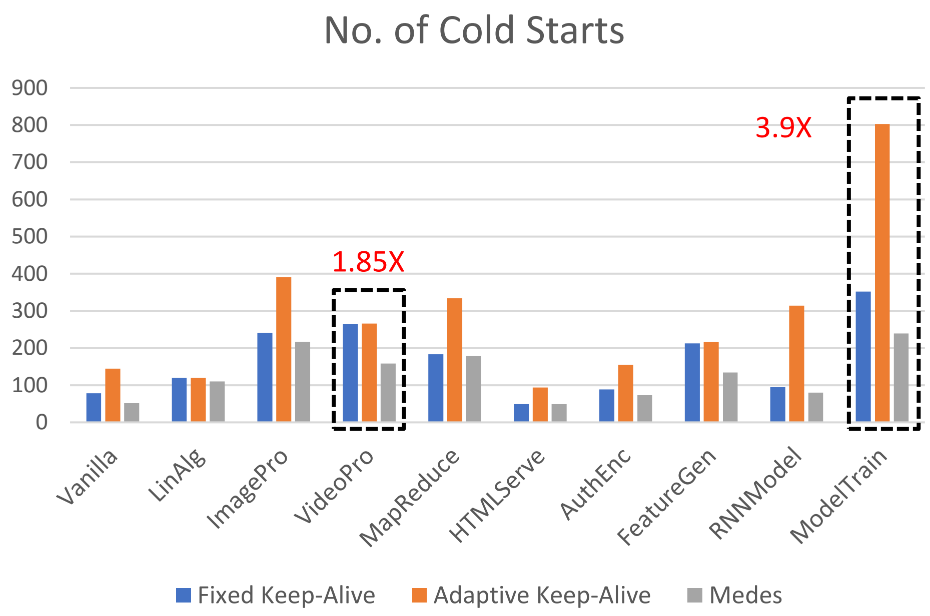
- Configuration
  - Controller – One xl170 node on Cloudlab (64GB RAM)
  - Nodes – Nineteen xl170 nodes, with 10Gbps NIC
    - Software memory constraint applied during experiments
- Workloads
  - One hour production traces of various applications released by Microsoft Azure
  - Functions from the FunctionBench benchmark suite

Evaluation

- Can Medes provide improved end-to-end latencies?
- How does Medes perform under memory pressure situations?
- Can Medes help reduce the overall memory footprints?
- Can tuning fixed keep-alive policies achieve the same performance-memory tradeoffs as Medes?

Evaluation – Improved end-to-end latencies



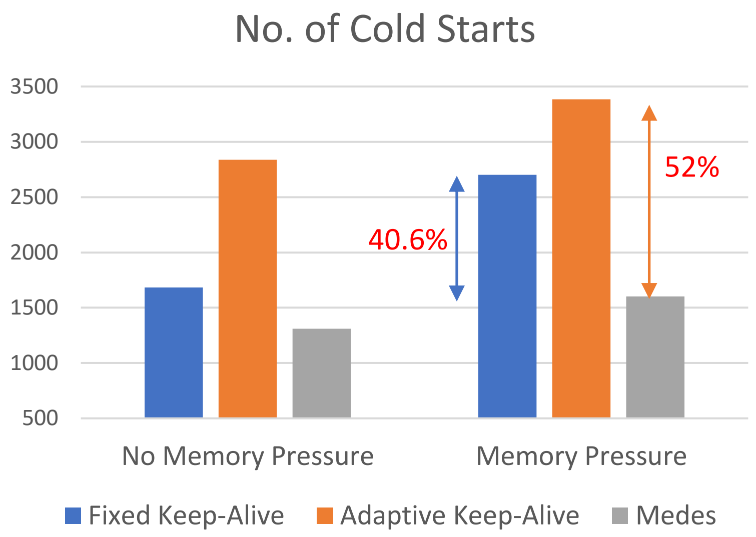


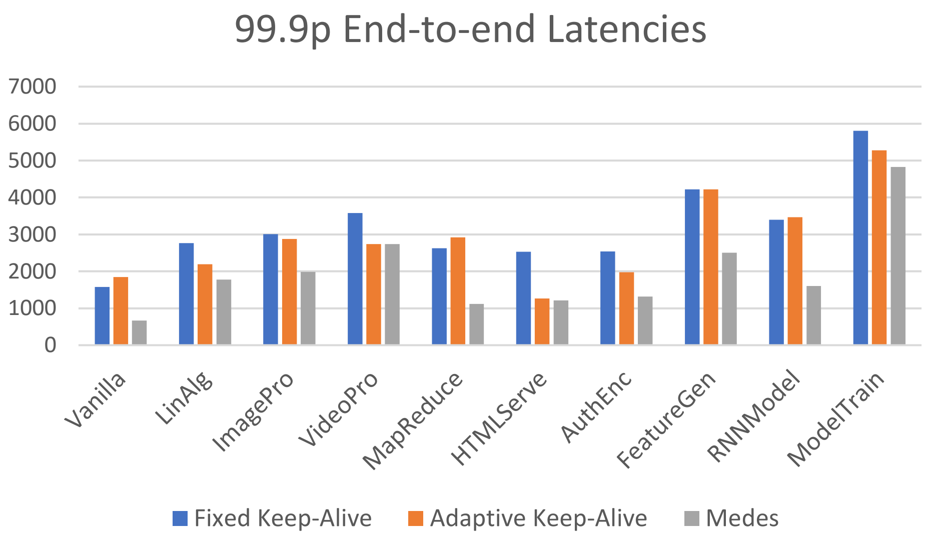
- Tail latencies are improved by a factor of 1-2.25X (1.3X on average).
- Cold starts are reduced up to 1.85X and 3.9X against fixed and adaptive policies.

Evaluation

- Can Medes provide improved end-to-end latencies?
- How does Medes perform under memory pressure situations?
- Can Medes help reduce the overall memory footprints?
- Can tuning fixed keep-alive policies achieve the same performance-memory tradeoffs as Medes?

Evaluation – Medes under Memory Pressure





The performance benefits increase under memory pressure.

- Cold starts reduced by 40.6% and 52% against fixed and adaptive keep-alive.
- Tail latencies are improved by a factor of up to 3.8X

Evaluation

- Can Medes provide improved end-to-end latencies?
- How does Medes perform under memory pressure situations?
- Can Medes help reduce the overall memory footprints?
- Can tuning fixed keep-alive policies achieve the same performance-memory tradeoffs as Medes?

Evaluation – Medes Memory Savings

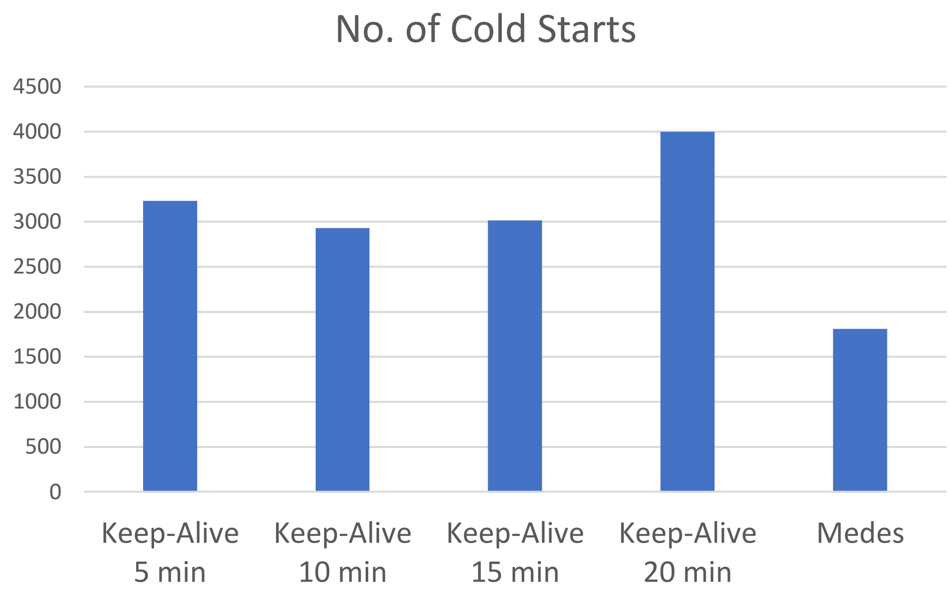
Function Environment	Percent Savings
Vanilla	$\frac{4.6MB}{17MB} = 27.06\%$
LinAlg	$\frac{10.5MB}{32MB} = 32.81\%$
ImagePro	$\frac{11.36MB}{26.4MB} = 43.03\%$
VideoPro	$\frac{12.22MB}{48MB} = 25.46\%$
MapReduce	$\frac{5.10MB}{32MB} = 15.94\%$
HTMLServe	$\frac{9.88MB}{22.3MB} = 44.30\%$
AuthEnc	$\frac{4.79MB}{22.3MB} = 21.48\%$
FeatureGen	$\frac{25.67MB}{66MB} = 38.89\%$
RNNModel	$\frac{52.23MB}{90MB} = 58.03\%$
ModelTrain	$\frac{26.33MB}{87.5MB} = 30.09\%$

- Medes can reduce 16-58% of the function memory state.

Evaluation

- Can Medes provide improved end-to-end latencies?
- How does Medes perform under memory pressure situations?
- Can Medes help reduce the overall memory footprints?
- Can tuning fixed keep-alive policies achieve the same performance-memory tradeoffs as Medes?

Evaluation – Flexibility achieved by Medes



Medes performs better than any keep alive

Best performing keep-alive

Large keep-alive leads to many evictions

Summary

- Medes can exploit sub-page level redundancy containers across a serverless platform.
- Medes uses:
  - Decoupling of redundancy identification and elimination granularities
  - Value-sampled fingerprints to identify similar pages
  - Fast sandbox restores
  - Flexible policy for sandbox management
- Medes can provide improved end-to-end latencies, memory usage.
- The benefits are enhanced for memory pressure scenarios.