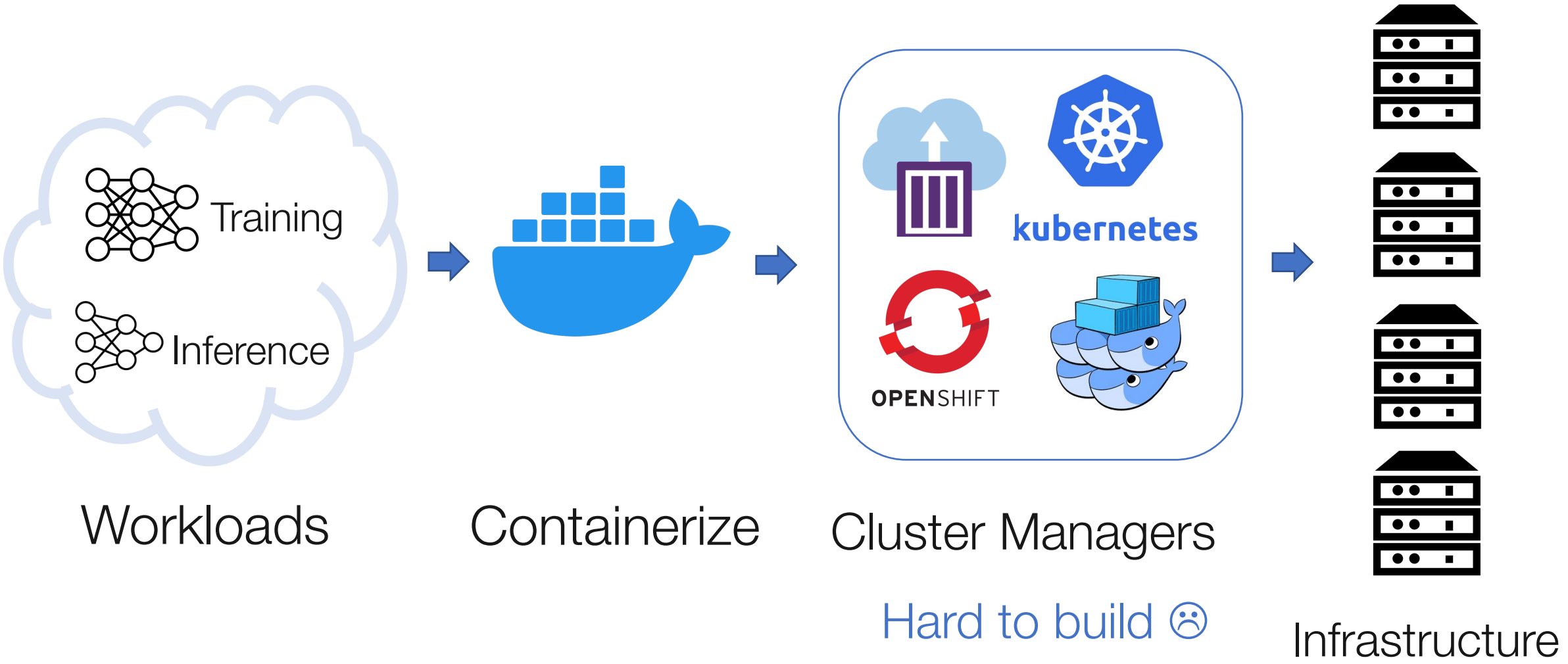
Scaling a Declarative Cluster Manager Architecture with Query Optimization Techniques

Kexin Rong[1,2], Mihai Budiu[3], Athinagoras Skiadopoulos[4],
Lalith Suresh[3], Amy Tai[5]

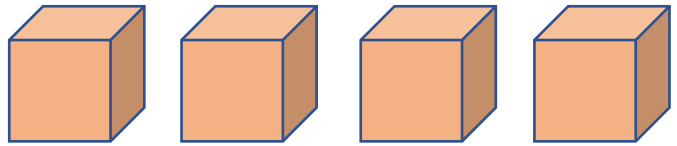
[1] Georgia Tech, [2] VMware Research, [3] Feldera*, [4] Stanford, [5] Google

*Work done while at VMware

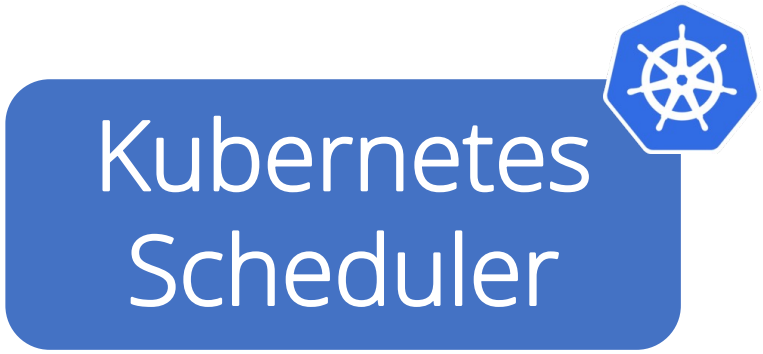
Cluster managers distribute workloads to resources



Running example: kubernetes scheduler



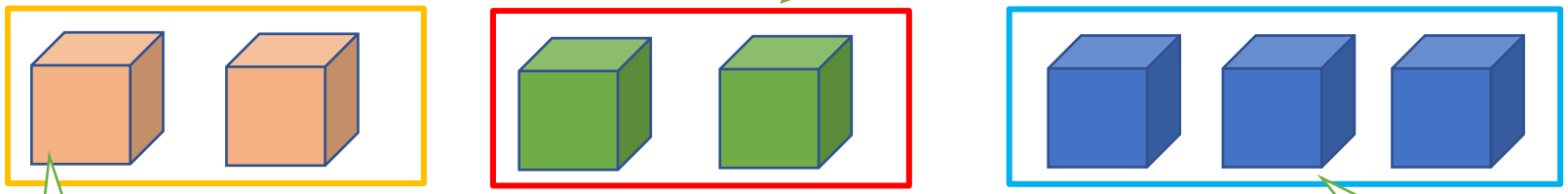
Pods



Nodes

Place us on the same rack!

Do NOT place us on the same rack!



POD
- 2GB RAM
- 16GB disk
- 1 core

Distribute us evenly!

Kubernetes Scheduler

30 types of hard and soft constraints

NP-Hard
Multi-dimensional bin-packing with constraints

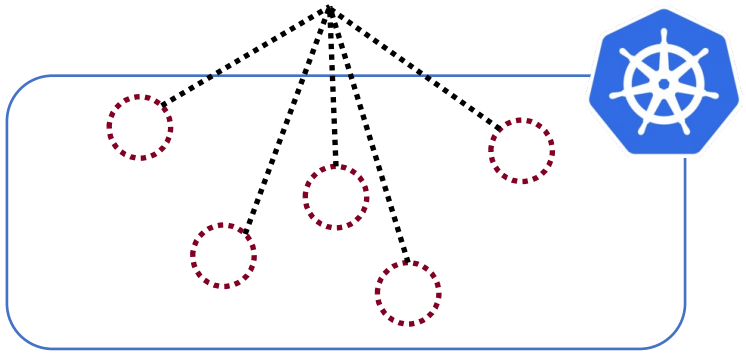
Nodes

How are cluster managers built?

Ad-hoc data structures for cluster states

+

Custom best-effort heuristics for decisions



Scalability?

Challenging with complex constraints

Decision quality?

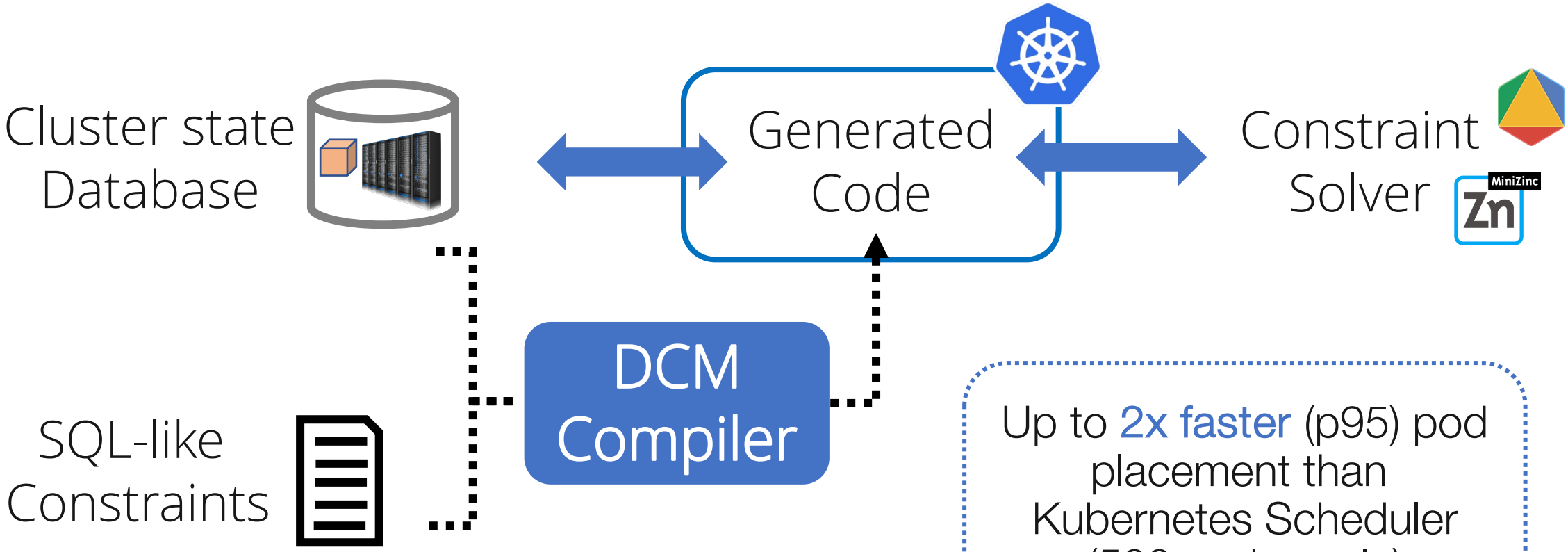
Can miss feasible solutions

Extensibility?

Hard to add new policies and features

DCM [HotOS'19, OSDI'20]
a declarative approach to cluster management

Developers specify what the cluster manager should achieve, not how

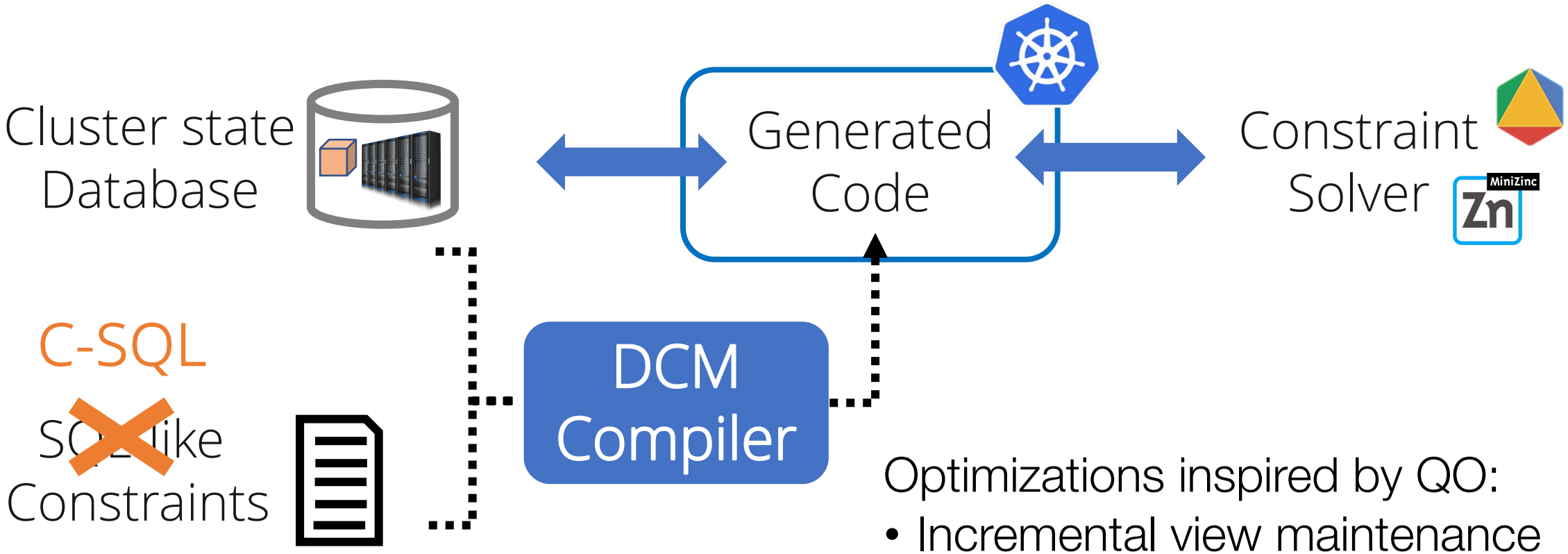


Up to 2x faster (p95) pod placement than Kubernetes Scheduler (500 node scale)

This work [VLDB'23]

From enterprise clusters (10^2-10^3 nodes) to hyperscale clusters (10^4 nodes)

With a formal language C-SQL and query-optimization techniques

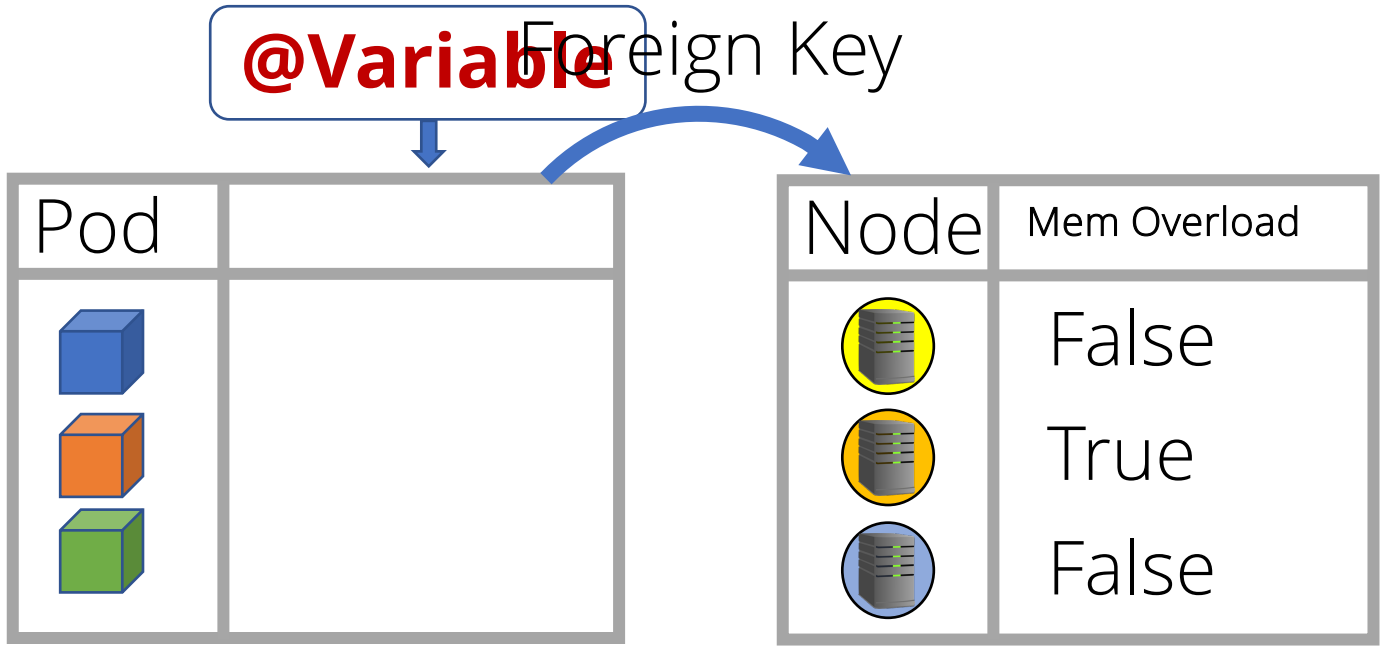


Optimizations inspired by QO:

- Incremental view maintenance
- Predicate Pushdown

C-SQL: SQL variant for constraint optimization

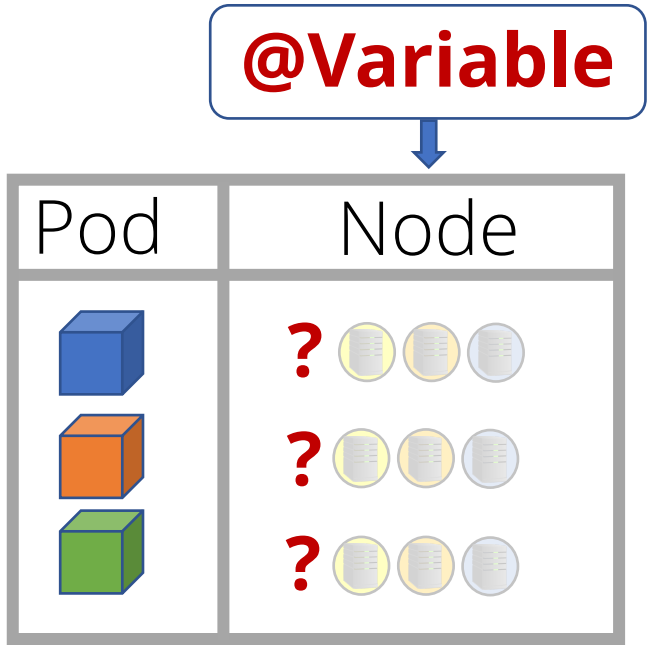
Variable Columns



Each row in the variable column is a variable in the constraint solver

C-SQL: SQL variant for constraint optimization

Hard Constraints



Node	Mem Overload
	False
	True
	False

```
CREATE CONSTRAINT avoid_mem_overload AS
CHECK pods.node IN
     (SELECT node
      FROM nodes
      WHERE nodes.mem_overload = false)
FROM pods
```

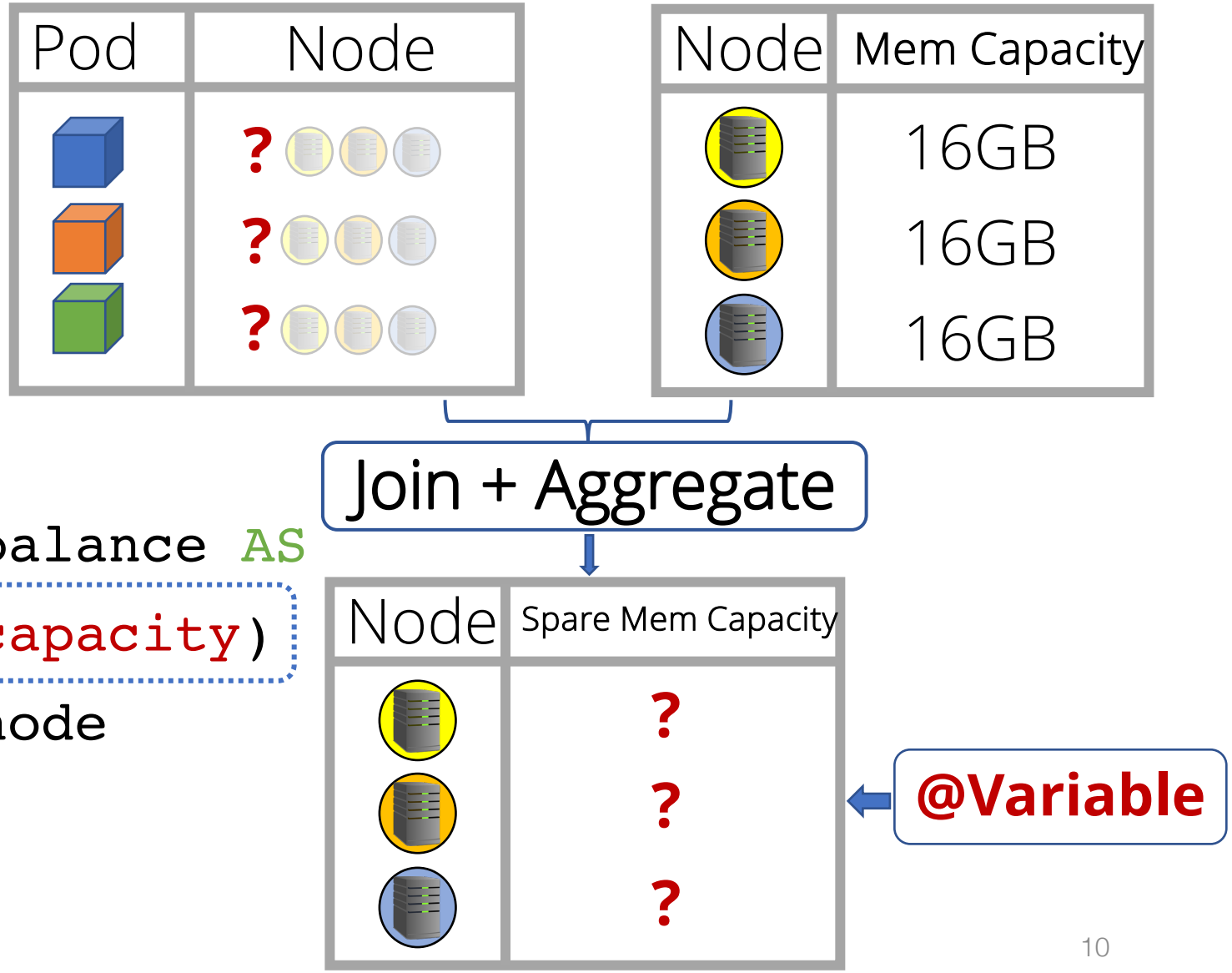
Constraint: conditions that each row satisfies

Relation: select some rows

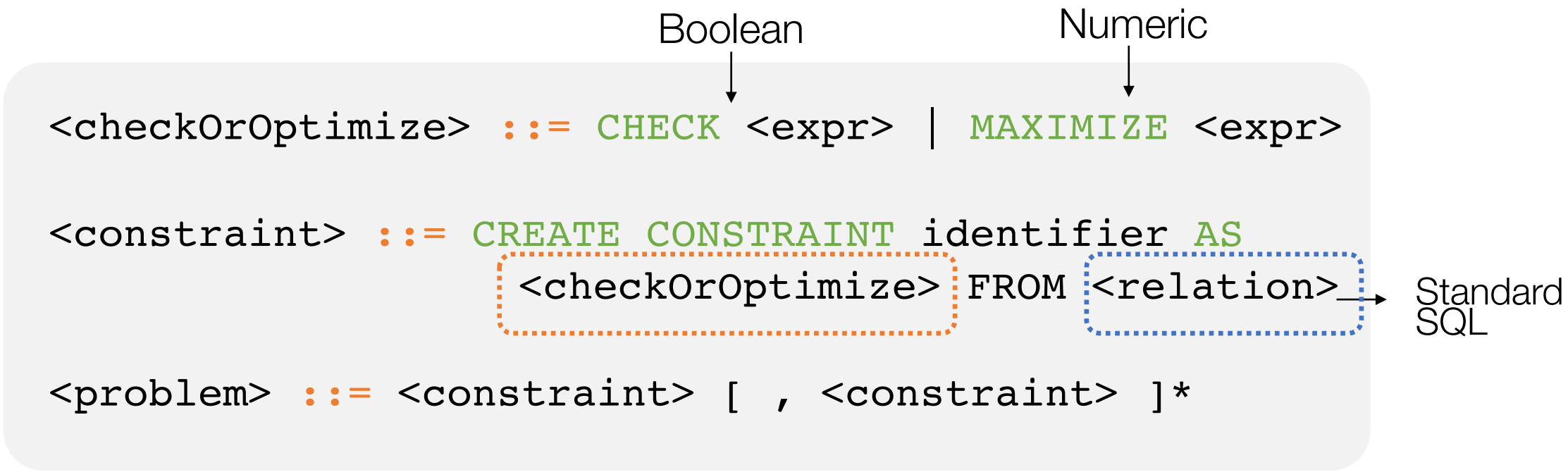
C-SQL: SQL variant for constraint optimization

Soft Constraints

```
CREATE CONSTRAINT load_balance AS
MAXIMIZE MIN(spare_mem_capacity)
FROM spare_capacity_by_node
```



C-SQL: SQL variant for constraint optimization



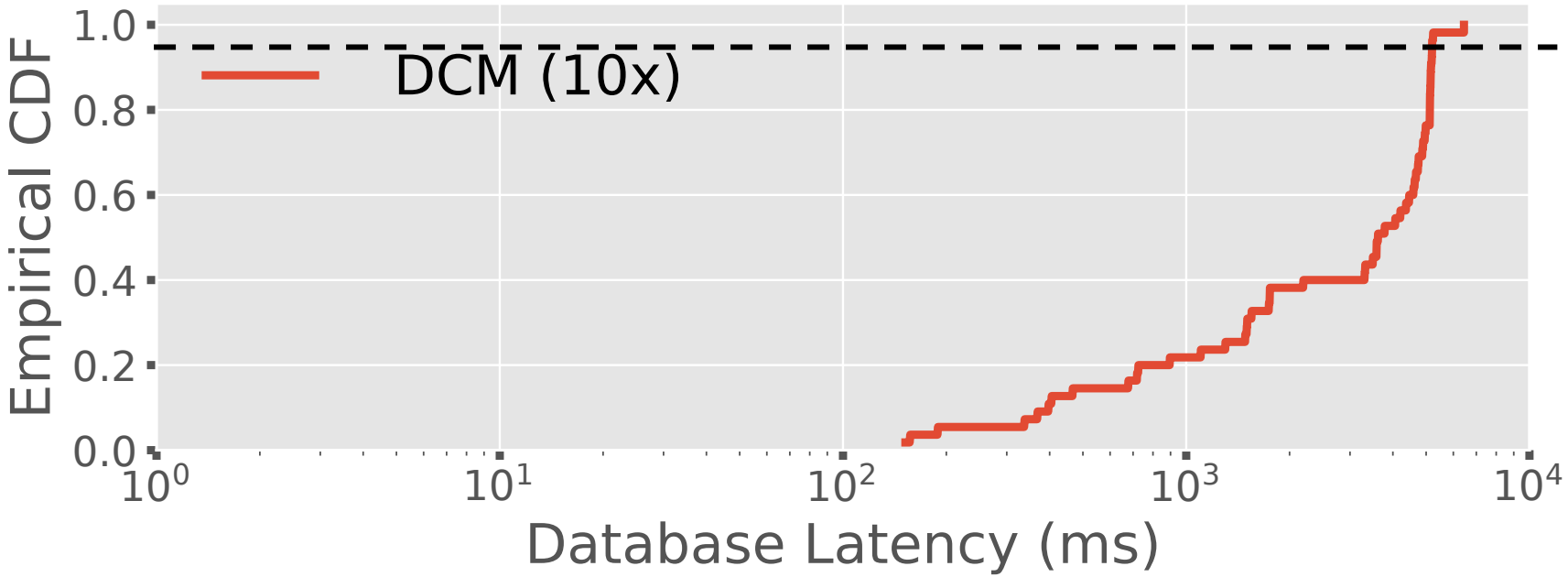
Constraint evaluation and relation evaluation are two bottlenecks

Will address with QO-inspired techniques

#1 Optimizing relation evaluation with IVM

```
<constraint> ::= CREATE CONSTRAINT identifier AS
                    <checkOrOptimize> FROM <relation>
```

Goal: sub-second overall latency for 50K-node cluster



p95 latency >5 sec
(500 node cluster)

Workload: 2019 Azure public trace, pod arrival rate sped up by 10x

#1 Optimizing relation evaluation with IVM

```
<constraint> ::= CREATE CONSTRAINT identifier AS
                    <checkOrOptimize> FROM <relation>
```

Opportunity: In a datacenter with O(100K) pods, a typical scheduling decision might only involve O(100) pods at a time, triggered by job arrivals or completions

=> make the work proportional to size of the changes, not the size of the databases

Solution: Automatically incrementalize the computation with IVM engine DDlog[1]

- Support SQL features such as joins, aggregates, GROUP BY, HAVING, OVER, and UNION
- Significant engineering efforts (11K LoC) to build a SQL-frontend and SQL-to-DDlog compiler [2]

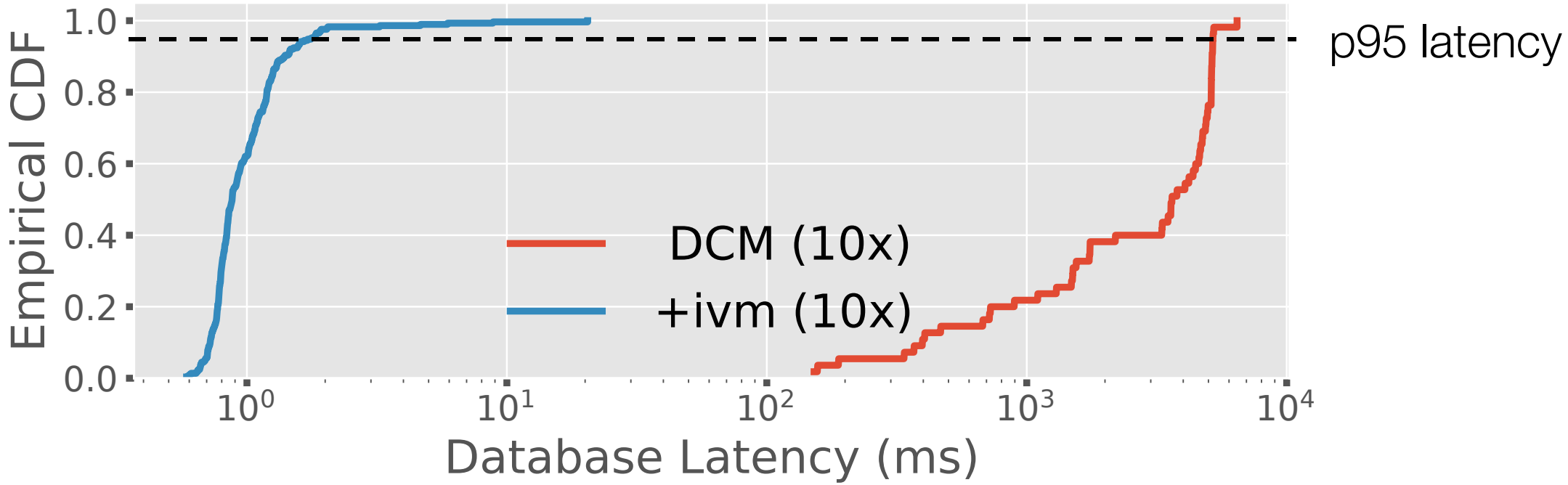
[1] Differential Datalog: https://github.com/vmware/differential-datalog
[2] Code available at: https://github.com/vmware/differential-datalog/tree/master/sql

Evaluation #1: Improved Performance

Workload: 2019 Azure public trace[1], pod arrival rate sped up by 10x

Environment: simulated cluster with 500 nodes

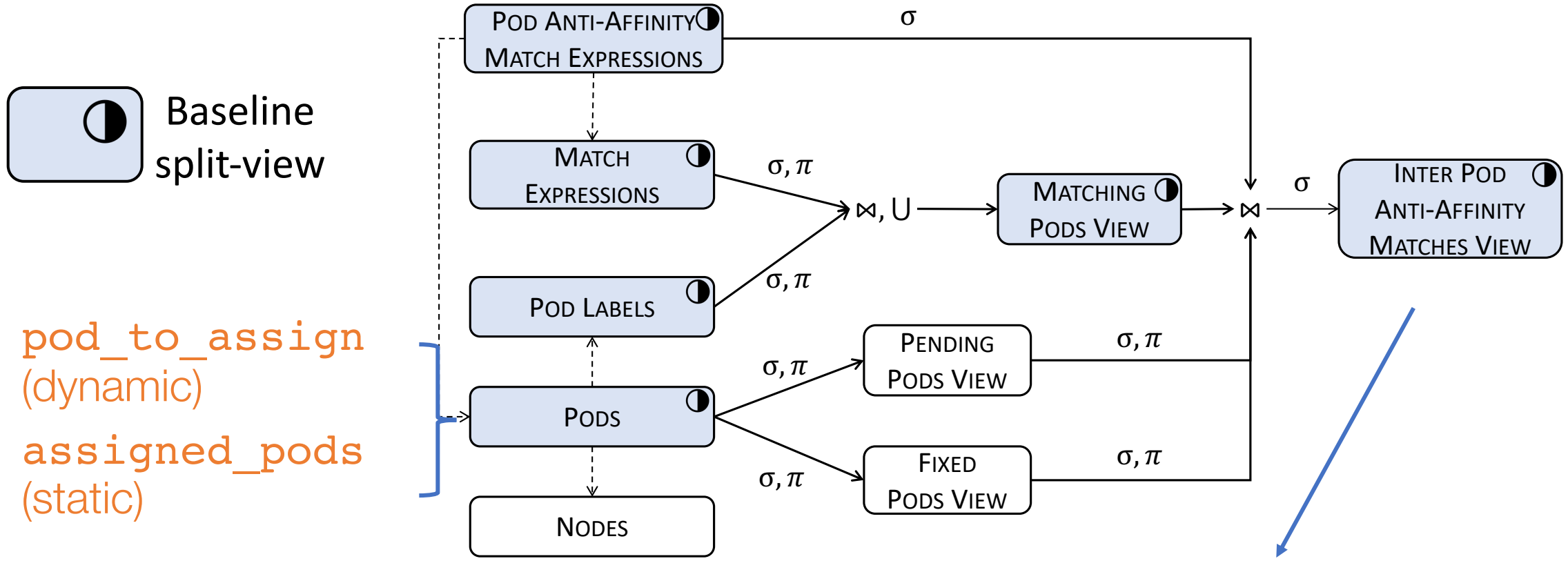
Result: p95 latency reduces from >5 seconds to 1.7ms (3000x speedup)



[1] https://github.com/Azure/AzurePublicDataset

Added benefit of IVM: code simplification

Before DDlog, users simulate IVM for performance via split views and triggers



~2.8× reduction in code size from ~185 lines per policy to ~64 lines

#2 Optimizing constraint evaluation with pushdowns

```
<constraint> ::= CREATE CONSTRAINT identifier AS
                 <checkOrOptimize> FROM <relation>
```

Opportunity: Certain constraints can be moved to relations without affecting correctness

Net effect: Reduce the optimization problem size

```
CREATE CONSTRAINT pod_affinity AS
CHECK (pods.has_pod_affinity = false) OR
      (pods.node NOT IN (…))
FROM pods
```

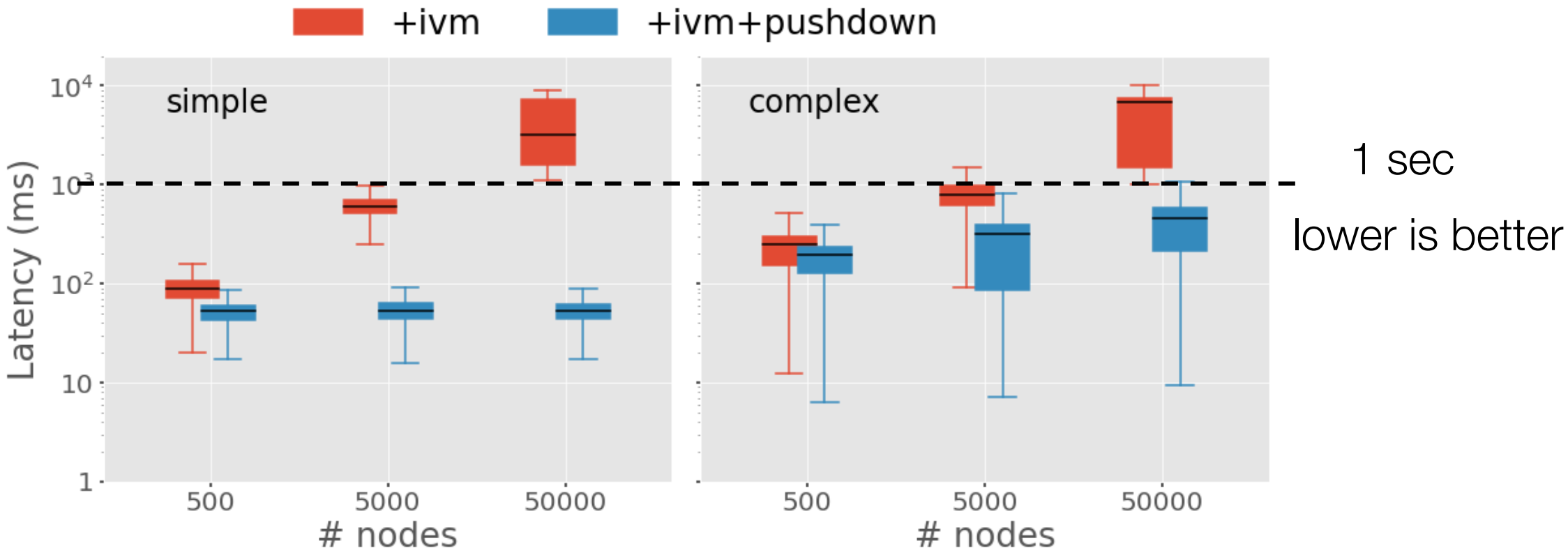
Feasibility-preserving constraint pushdown

```
CREATE CONSTRAINT pod_affinity AS
CHECK (pods.node NOT IN (…))
FROM pods
WHERE pods.has_pod_affinity = true
```

Evaluation #2: Effect of Constraint Pushdown

Environment: simulated cluster with 500, 5k and 50k nodes

Result: sub-second scheduling latency at 50K node cluster size (reduced optimization problem sizes by over 300x without affecting correctness)



Scaling DCM with Query Optimization Techniques

C-SQL: SQL-variant for constraint optimization

```
<constraint> ::= CREATE CONSTRAINT identifier AS
                 <checkOrOptimize> FROM <relation>
```

Two query optimization inspired techniques

- Incremental view maintenance for relation evaluation
  - Making work proportional to size of the changes, not the size of the databases
- Predicate Pushdown for constraint evaluation
  - Pushing down constraints to reduce optimization problem size

Net effect: sub-second scheduling latency on 50k-node clusters

Code available at: https://github.com/vmware/declarative-cluster-management/releases/tag/vldb23