Building Scalable and Flexible Cluster Managers Using Declarative Programming

github.com/vmware/declarative-cluster-management/

Lalith Suresh

VMware

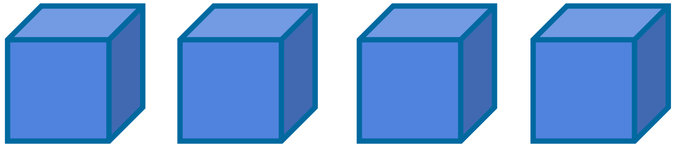
Hard to Build ☹

Cluster Managers

DCM ☺

Code-generate implementations from high-level specifications

Pods

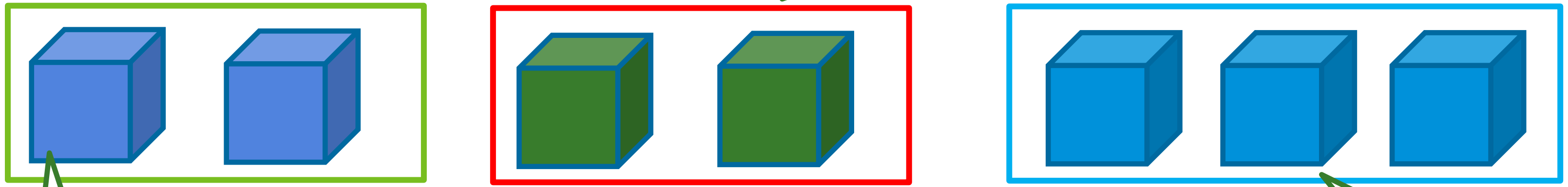


Kubernetes Scheduler

Nodes

Place us on the same rack!

Do **NOT** place us on the same rack!



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

Kubernetes Scheduler

Distribute us evenly!

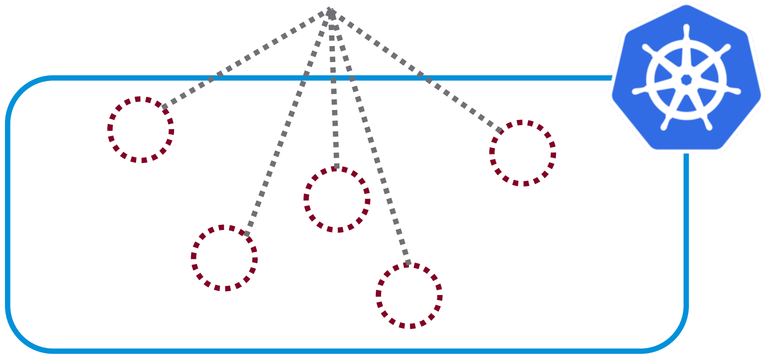
30 hard and soft constraints

NP-Hard
Multi-dimensional bin-packing with constraints

Nodes

Kubernetes Scheduler

Custom
Best-effort
Heuristics



Initial Placement

Filter-score approach

Queue

foreach ( )

.filter(H_1, ,)
.filter(H_2, ,)
…
.filter(H_n, ,)

.score(S_1, ,)
.score(S_2, ,)
…
.score(S_n, ,)

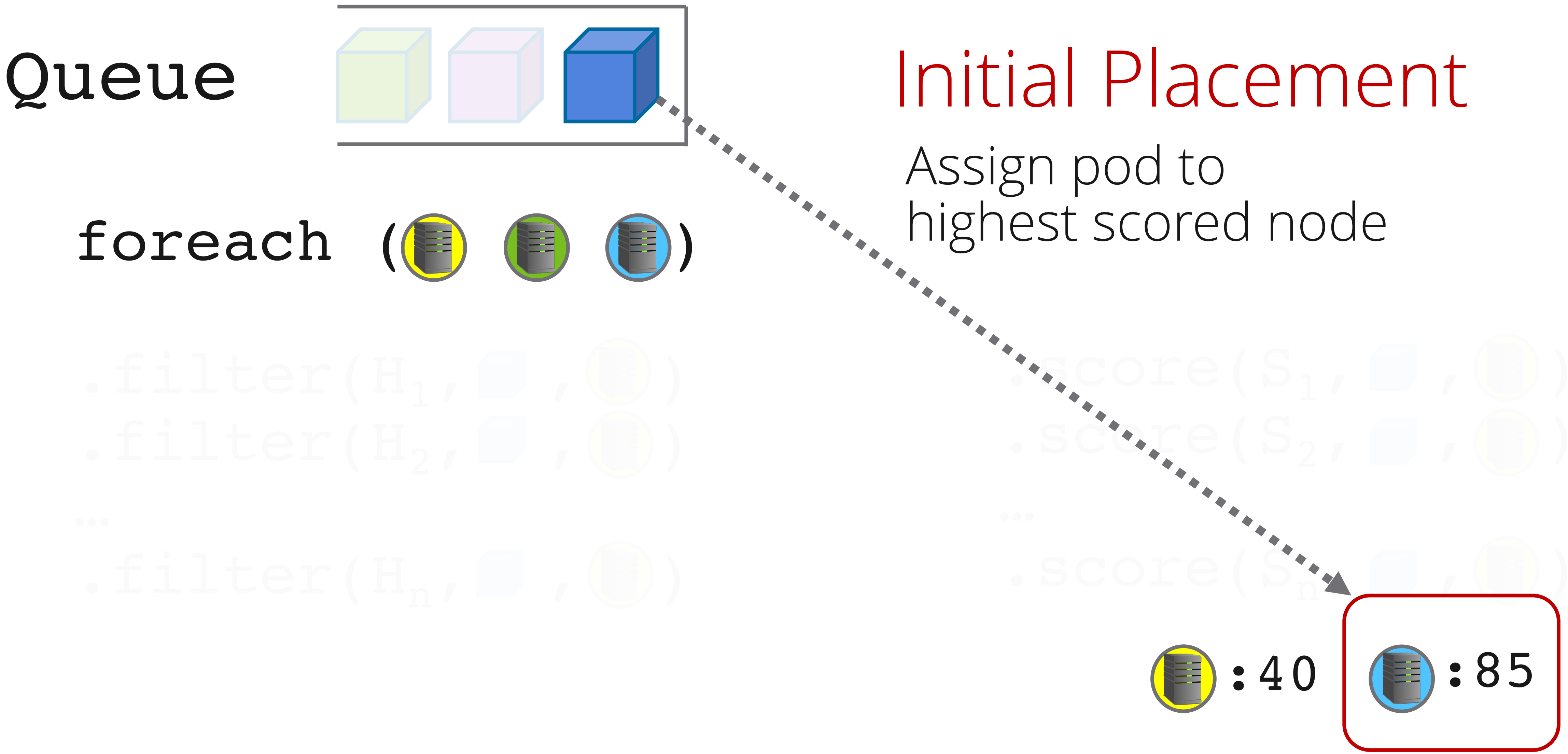
Hard constraints
Valid nodes, Capacity, Affinities, Anti-affinities…

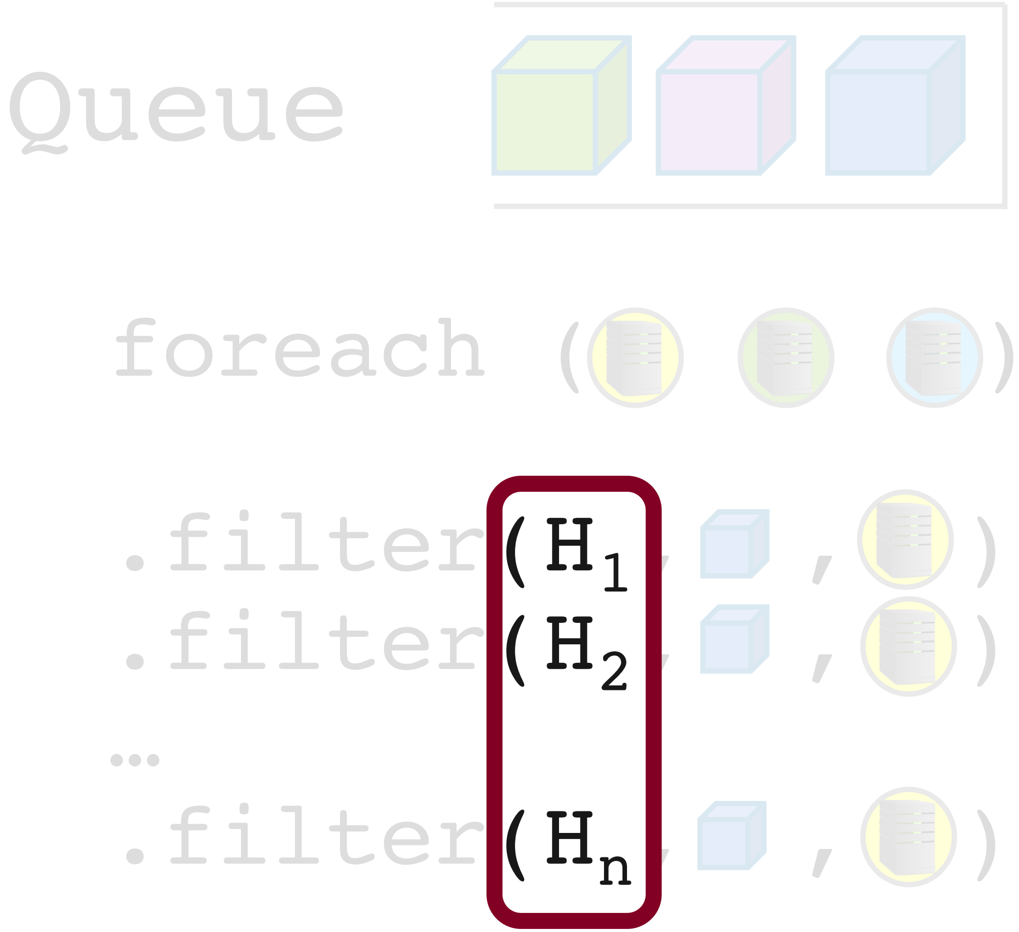
: 40

Soft constraints
Nodes with images, load balancing…

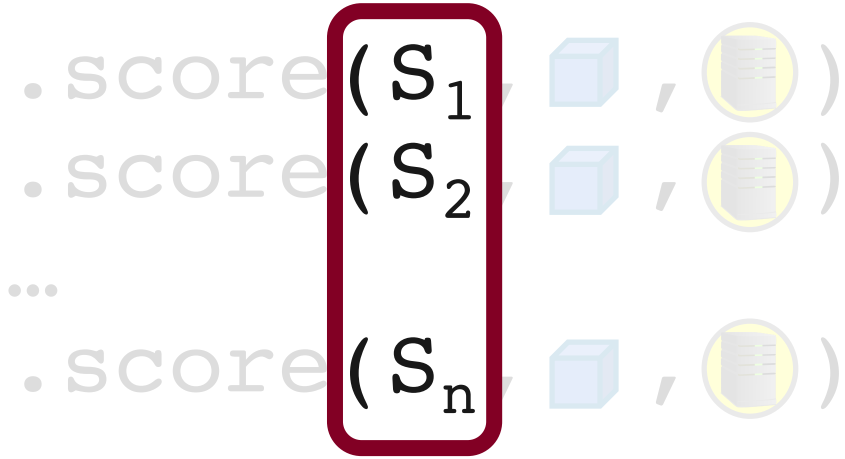
Initial Placement

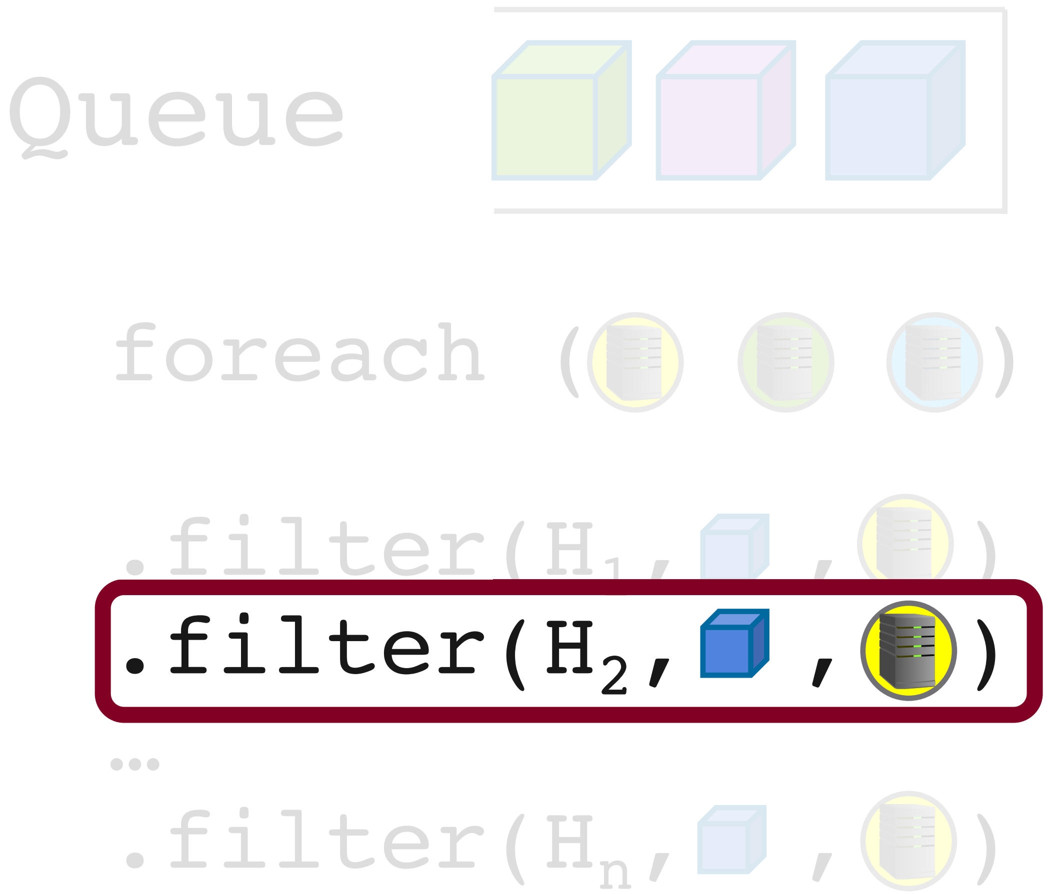
Assign pod to
highest scored node





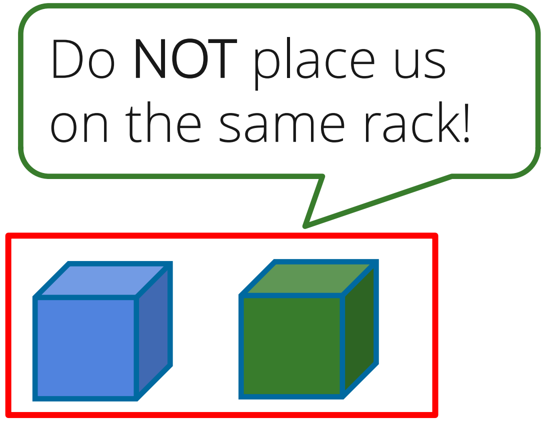
Each policy is implemented as a heuristic

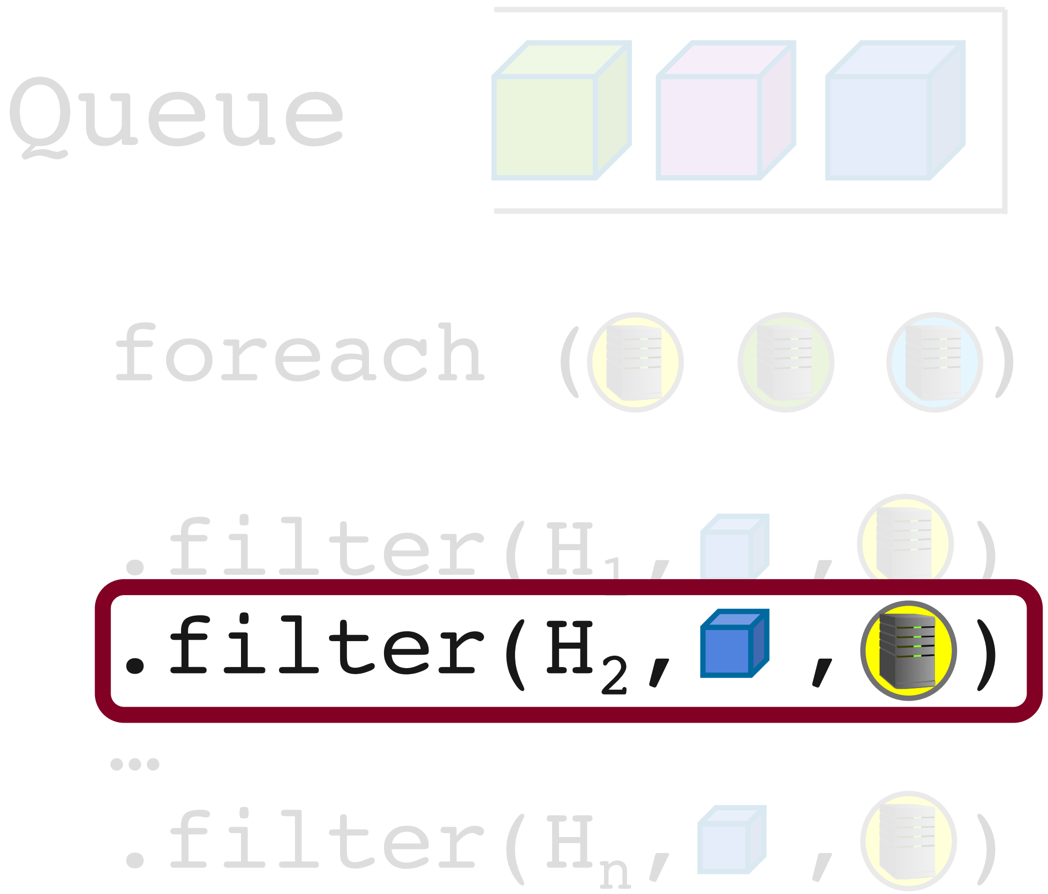




Global reasoning about groups of pods/nodes is hard

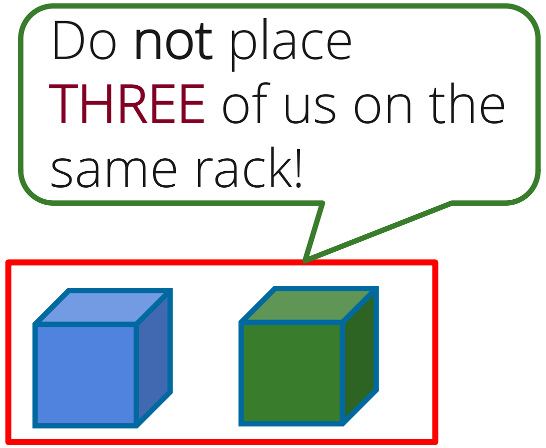
Requires brittle pre-computing and caching optimizations





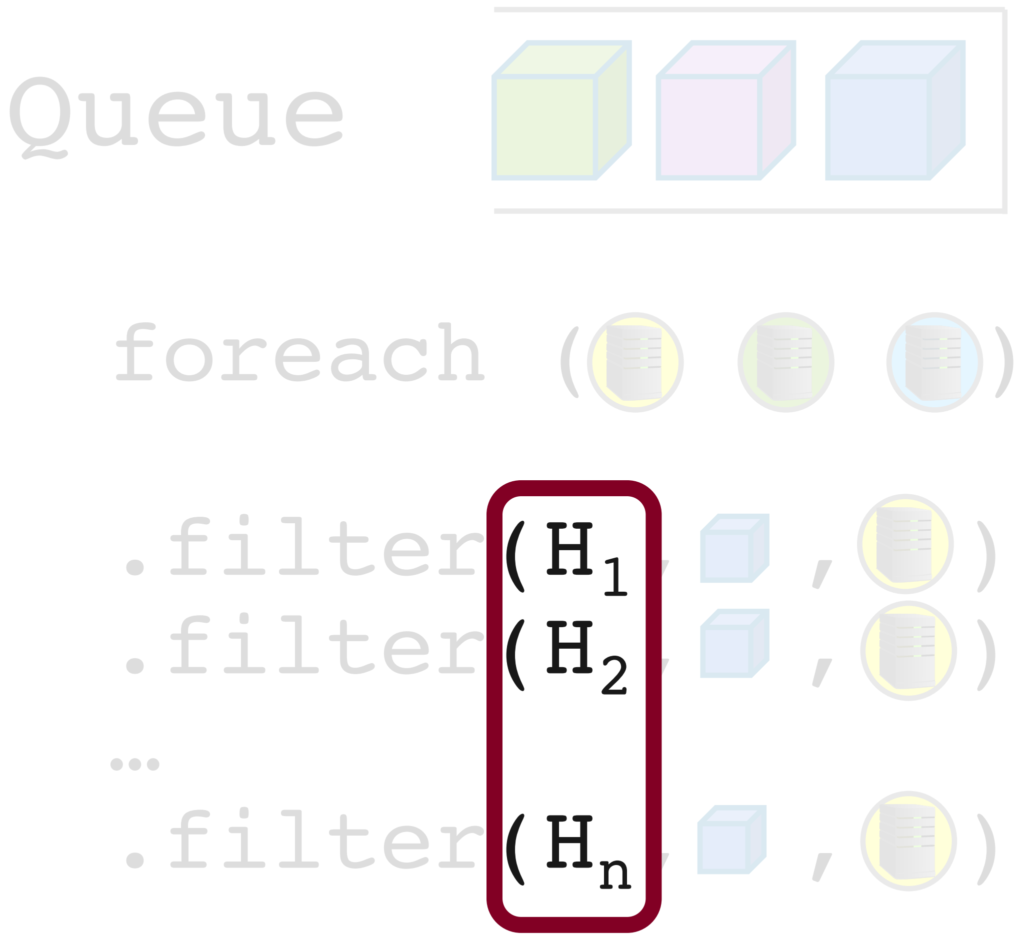
Global reasoning about groups of pods/nodes is hard

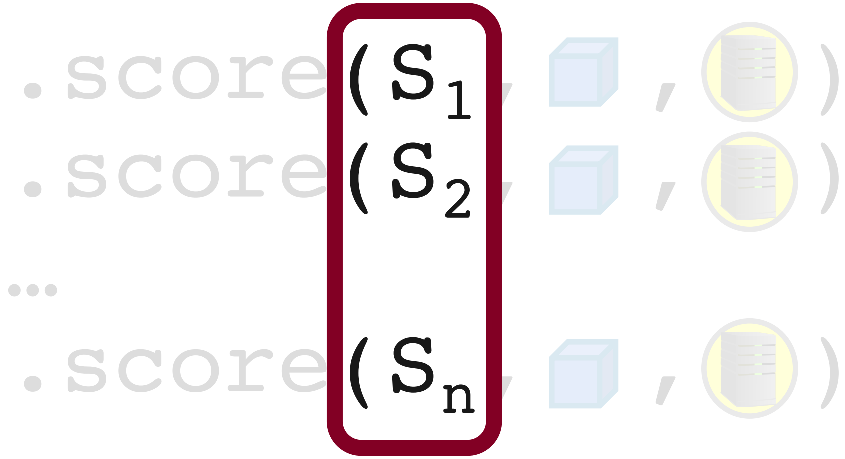
Optimizations break when requirements evolve



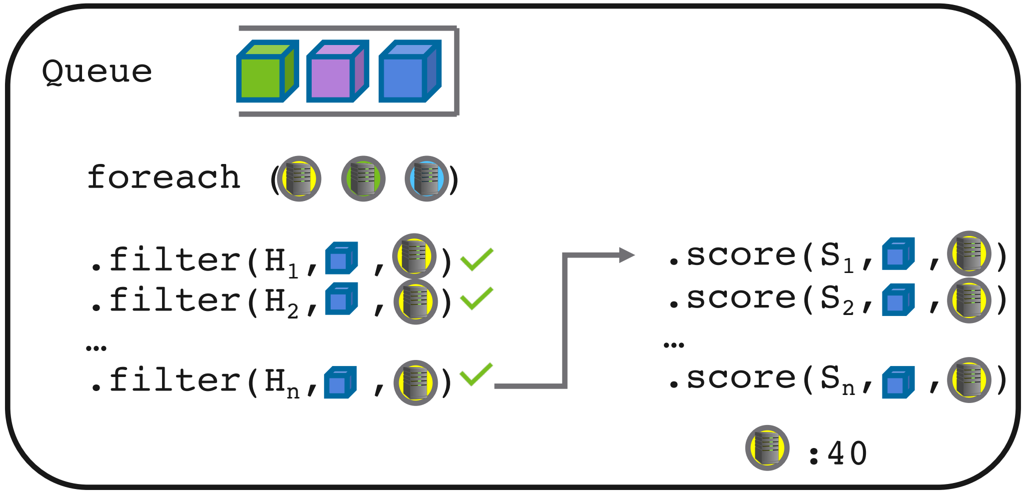
Performance?

Sampling, order dependent





Initial Placement



Pre-emption?

Make room by removing low-priority pods

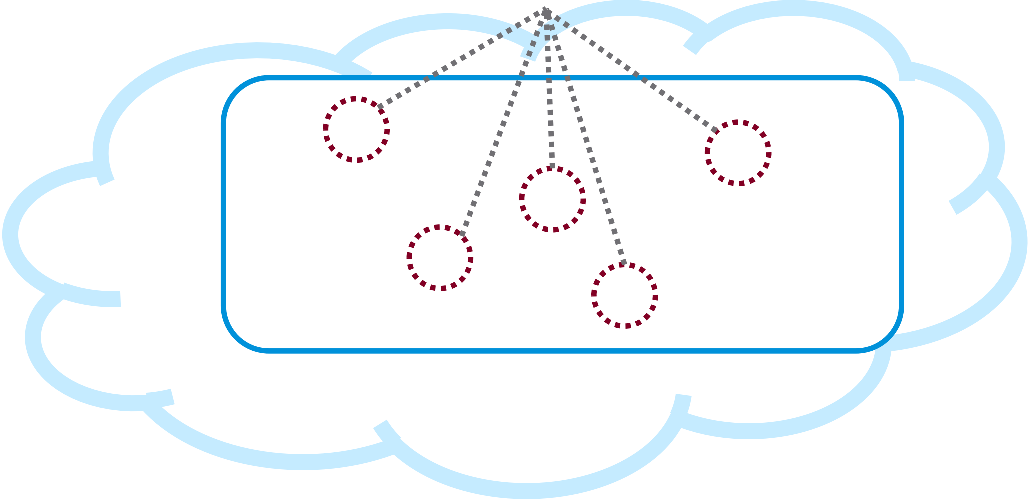
… with more ad-hoc heuristics to:

- pick potential nodes
- evaluate constraints
- retry scheduling assuming some pods are removed

De-scheduling?

Batch scheduling?

Custom
Best-effort
Heuristics



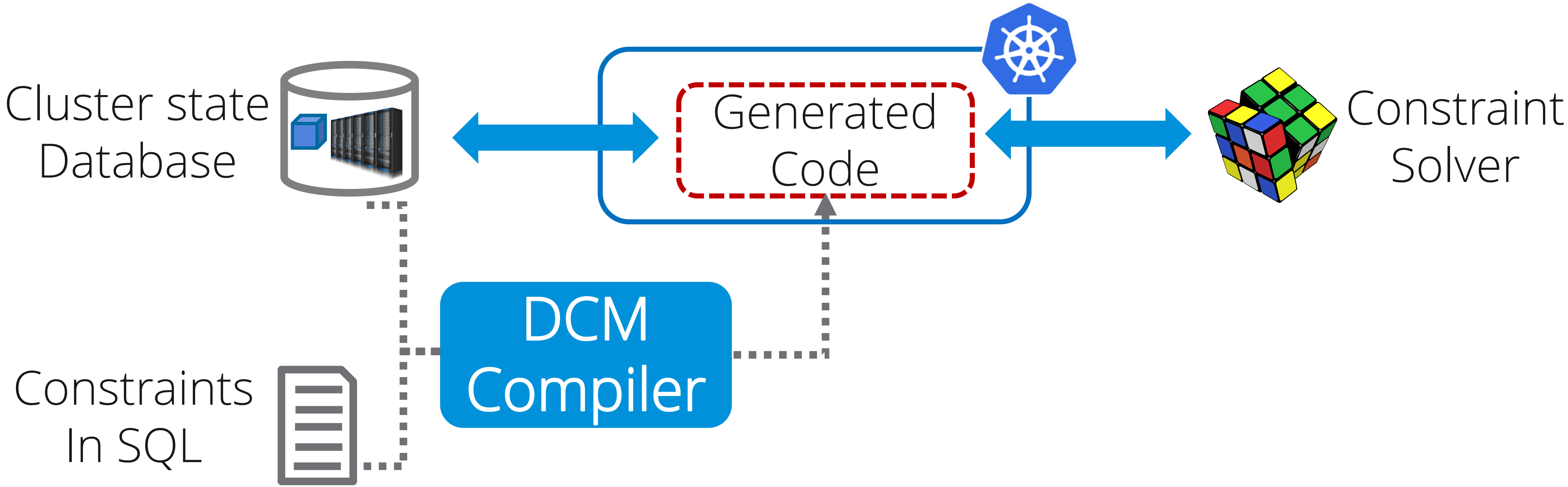
Scalability?
Challenging with complex constraints

Decision quality?
Can miss feasible solutions

Extensibility?
Hard to add new policies and features

Our approach

Declarative Cluster Managers (DCM)



Our approach

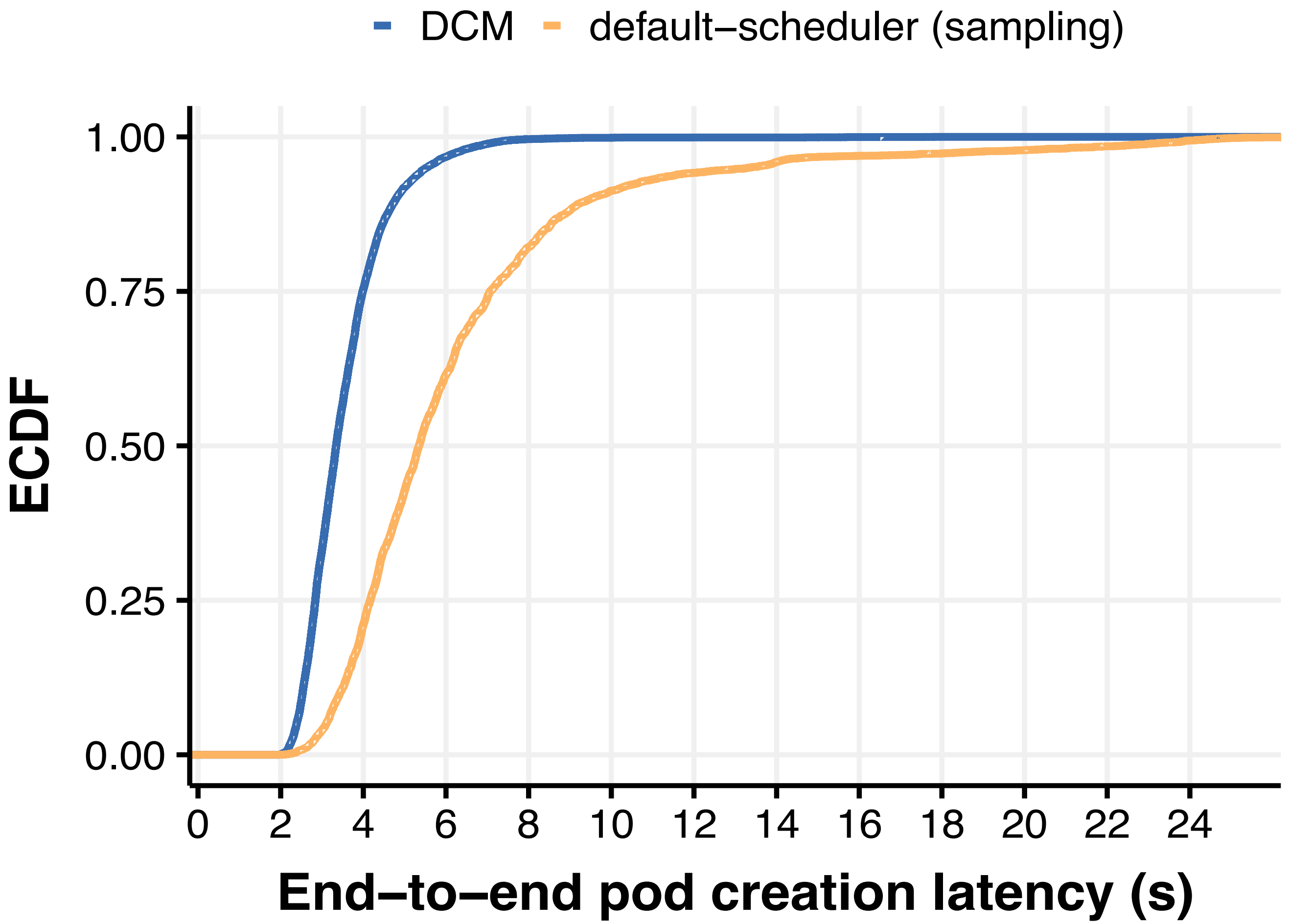
Declarative Cluster Managers (DCM)

Kubernetes Scheduler

Scalability

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

tig

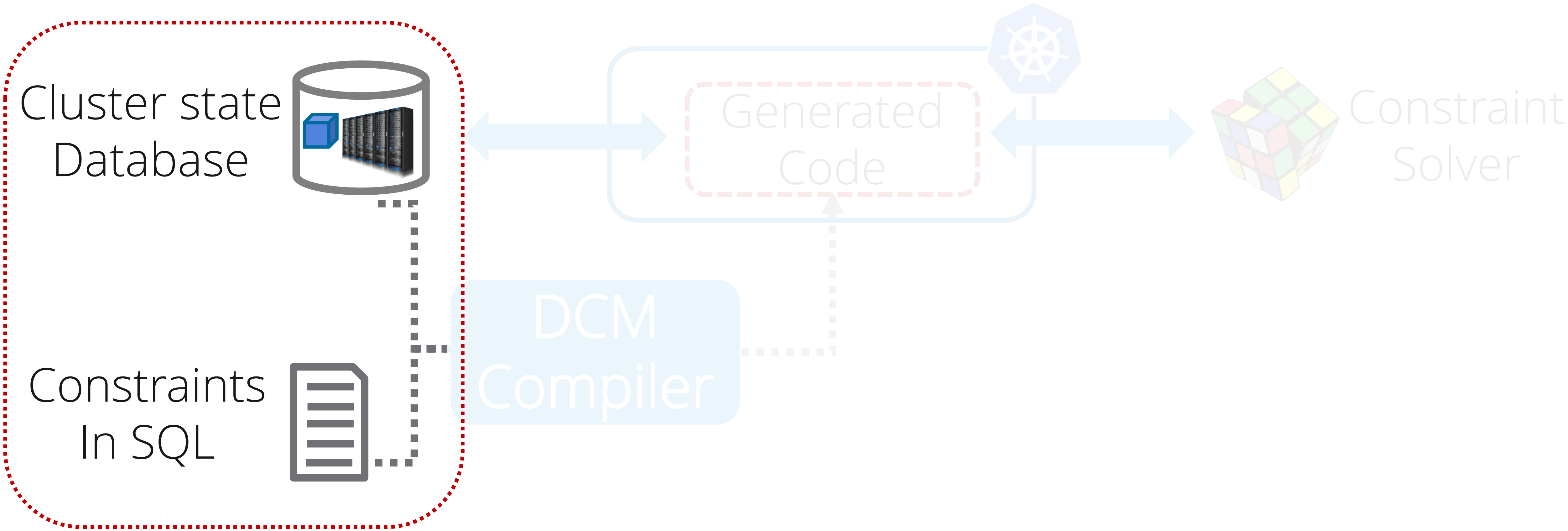


SQL

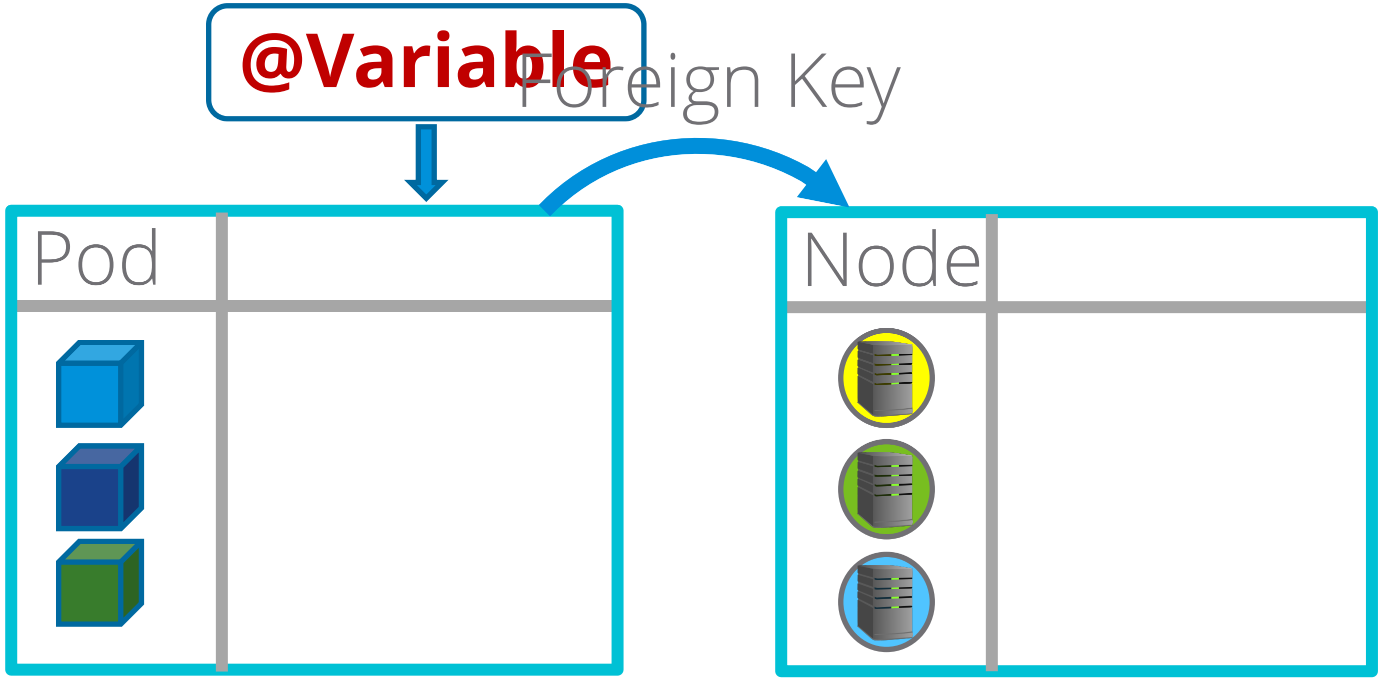
s

uling)

Programming Model



Variable Columns



Hard Constraints

@Variable

Pod	Node
	?
	?
	?

Node	Mem Overload
	False
	True
	False

@ hard constraint

```
CREATE VIEW avoid_mem_overload AS
```

Select some rows

Predicate

Soft Constraints

@Variable

Pod	Node
	?
	?
	?

Node	Mem Capacity
	16GB
	16GB
	16GB

@ soft constraint

```
CREATE VIEW load_balance AS
```

Numeric expressions to maximize

Programming Model

Express policies concisely using joins, aggregates, group bys, sub-queries, correlated sub-queries, arrays...

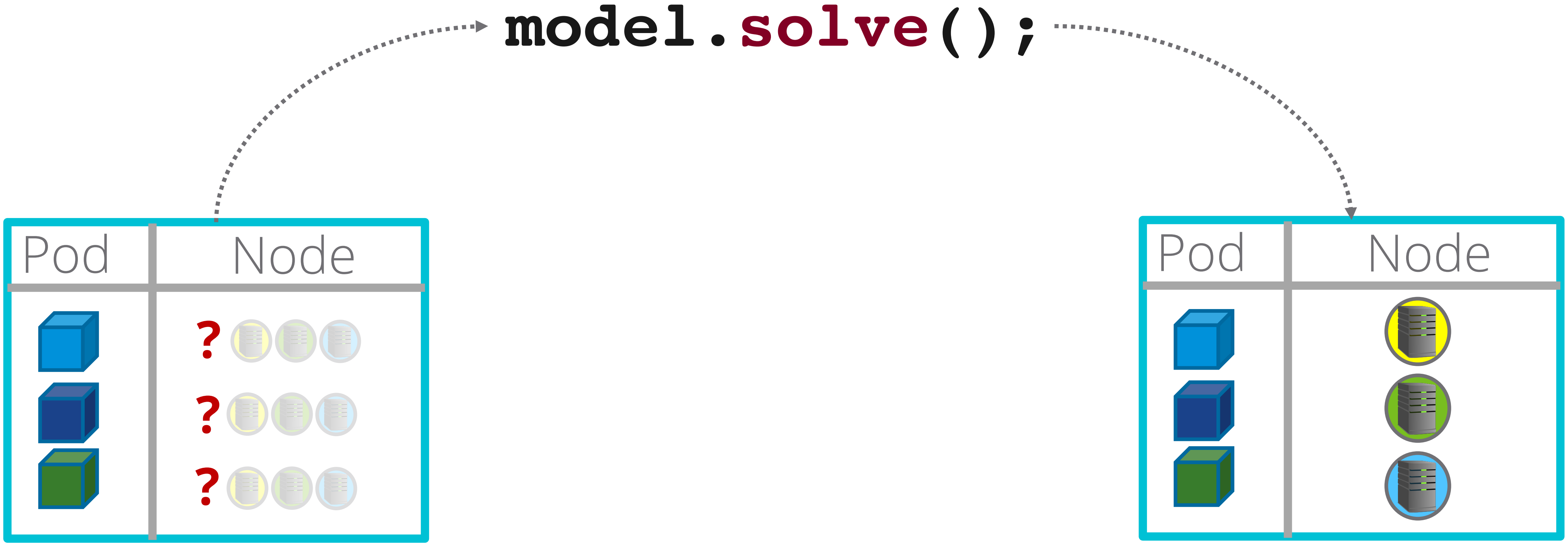
Programming Model

```
model = Model.create(dbConnection,
                    constraints.sql);

model.updateData();

model.solve();
```

Tables in, tables out



UNSAT cores

Which constraints failed?

```
model.solve(); // unsat ☹
```

```
solverException.core()
["load_constraint", "az_constraint"]
```

Cluster state
Database

Different models
Different tasks
Different timescales

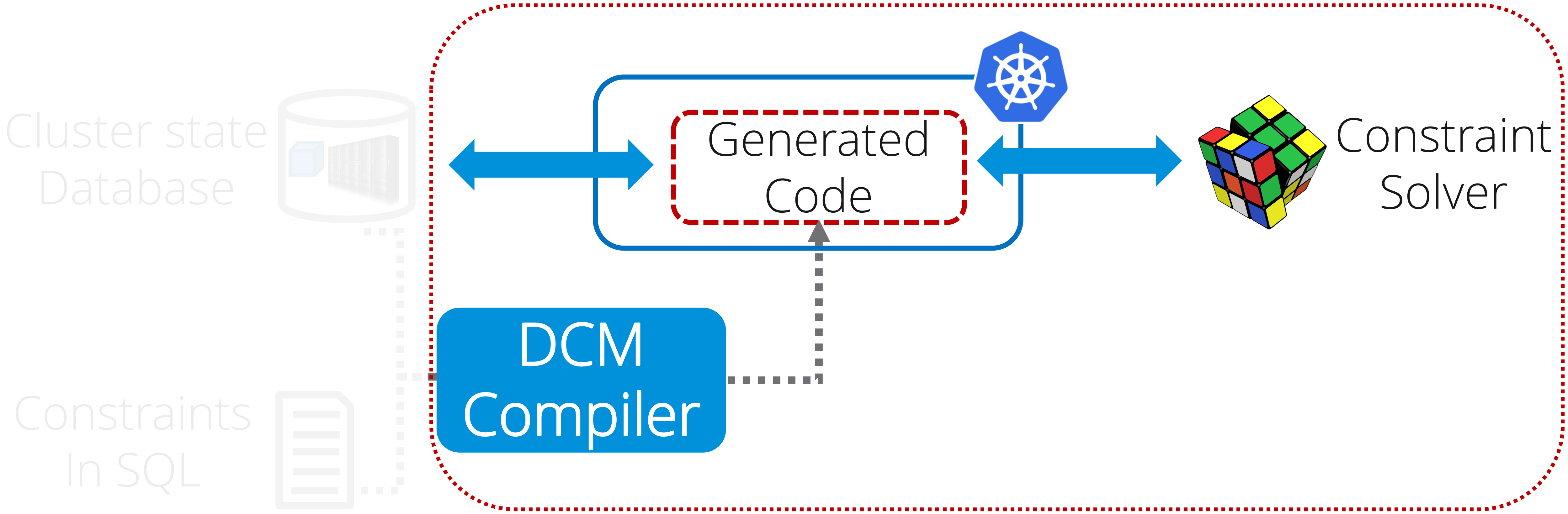
Cluster state
Database

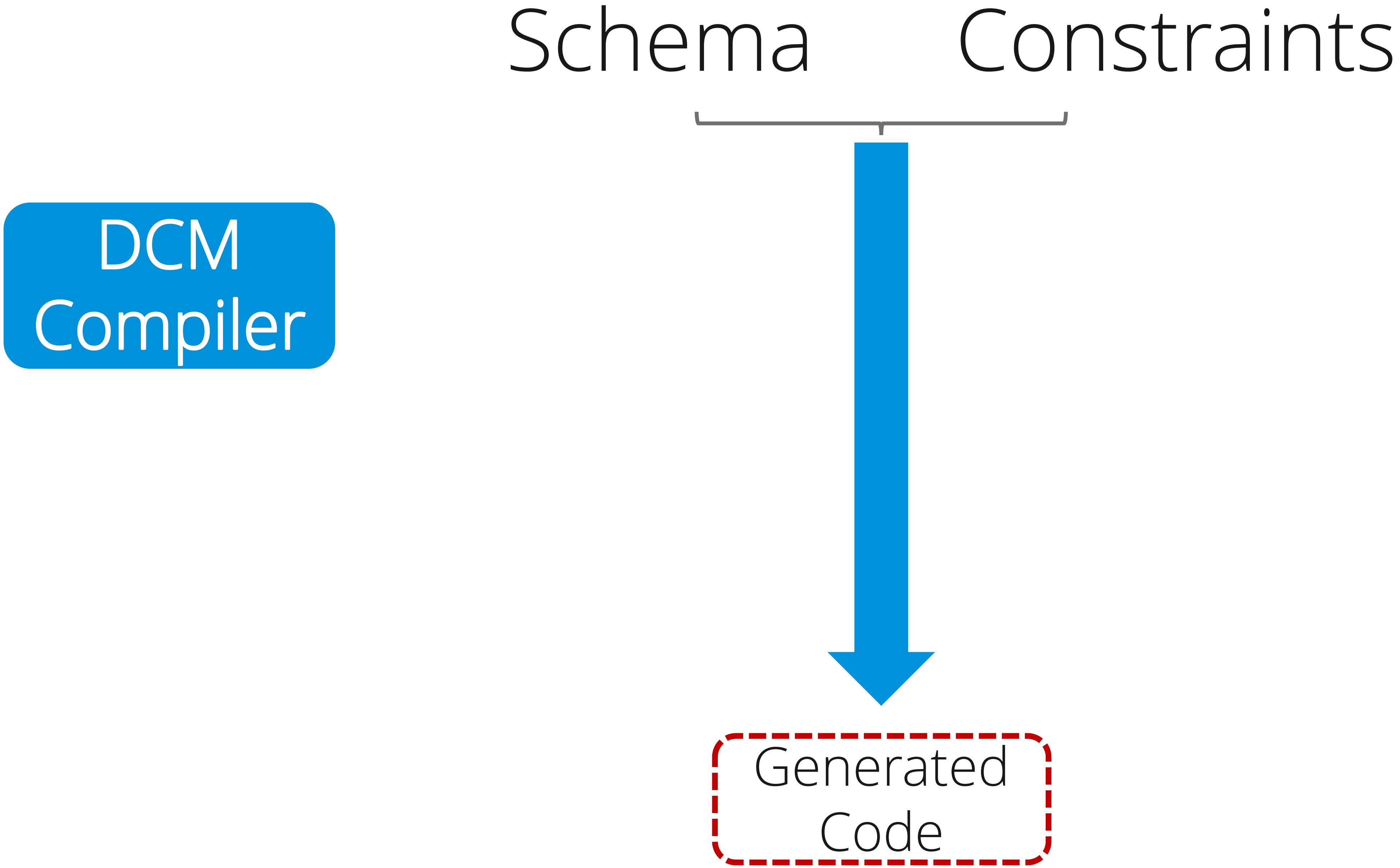
`initialPlacementModel`

`preemptionModel`

`deschedulingModel`

DCM Compiler




Schema
Constraints
DCM
Compiler
Generated
Code

```sql
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```sql
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

DCM Compiler

Generated Code

DCM Compiler

```
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

List comprehension Intermediate Repr.

Generated Code

DCM Compiler

```
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

```
[CHECK t1_c[i] * t2_d[i] == t2_c[j]
 | i -> RANGE(t1), j -> RANGE(t2),
  WHERE t1_b[i] == t2_b[j]
    AND t2_e[j] == 10]
```

Generated Code

DCM Compiler

```
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

```
[CHECK t1_c[i] * t2_d[i] == t2_c[j]
 | i -> RANGE(t1), j -> RANGE(t2),
  WHERE t1_b[i] == t2_b[j]
    AND t2_e[j] == 10]
```

Backend-specific Code generation

```
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

DCM Compiler

```
[CHECK t1_c[i] * t2_d[i] == t2_c[j]
 | i -> RANGE(t1), j -> RANGE(t2),
  WHERE t1_b[i] == t2_b[j]
    AND t2_e[j] == 10]
```

Flagship Backend
ORTools CP/SAT

```sql
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```sql
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

DCM Compiler

```
[CHECK t1_c[i] * t2_d[i] == t2_c[j]
 | i -> RANGE(t1), j -> RANGE(t2),
  WHERE t1_b[i] == t2_b[j]
    AND t2_e[j] == 10]
```

Generated Code

```java
for (int t1_it = 0; t1_it < t1.size(); t1_it++) {
  var t2_it = t2Index.get(t1.get(t1_it).getB());
  if (t2_it == null) continue;
  if (t2_e.get(t2_it).getE() == 10) {
     var i1 = o.prod(t1.get(t1_it).getCVar(),
                     t2.get(t2_it).getDVar());
     var i2 = o.eq(t1_c[t1_it], i1);
     model.addEquality(i1, i2);
  }
}
```

DCM Compiler

```
SELECT * FROM t1
```

Iterate over tables

```
for (int t1_it = 0; t1_it < t1.size(); t1_it++) {

}
```

DCM Compiler

```sql
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
```

Joins with indexes or nested for loops

```java
for (int t1_it = 0; t1_it < t1.size(); t1_it++) {
  var t2_it = t2Index.get(t1.get(t1_it).getB());
  if (t2_it == null) continue;

}
```

```sql
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
```

DCM Compiler

Remove irrelevant rows

```java
for (int t1_it = 0; t1_it < t1.size(); t1_it++) {
  var t2_it = t2Index.get(t1.get(t1_it).getB());
  if (t2_it == null) continue;
  if (t2_e.get(t2_it).getE() == 10) {


  }
}
```

DCM Compiler

```
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

Encode into constraints

```
for (int t1_it = 0; t1_it < t1.size(); t1_it++) {
  var t2_it = t2Index.get(t1.get(t1_it).getB());
  if (t2_it == null) continue;
  if (t2_e.get(t2_it).getE() == 10) {
     var i1 = o.prod(t1.get(t1_it).getCVar(),
                     t2.get(t2_it).getDVar());
     var i2 = o.eq(t1_c[t1_it], i1);
     model.addEquality(i1, i2);
  }
}
```

```
CREATE TABLE T1 (…);
CREATE TABLE T2 (…);
```

```
CREATE VIEW constraint_1 AS
SELECT * FROM t1
JOIN t2 ON t1.b = t2.b
WHERE t2.e = 10
CHECK t1.c * t2.d = t2.c
```

DCM Compiler

Generated Code

```
for (int t1_it = 0; t1_it < t1.size(); t1_it++) {
  var t2_it = t2Index.get(t1.get(t1_it).getB());
  if (t2_it == null) continue;
  if (t2_e.get(t2_it).getE() == 10) {
     var i1 = o.prod(t1.get(t1_it).getCVar(),
                     t2.get(t2_it).getDVar());
     var i2 = o.eq(t1_c[t1_it], i1);
     model.addEquality(i1, i2);
  }
}
```

Key technical challenge

Using the constraint solver effectively

Google OR-Tools CP/SAT solver

Google OR-Tools CP/SAT solver

“Search is dead”

CP solver

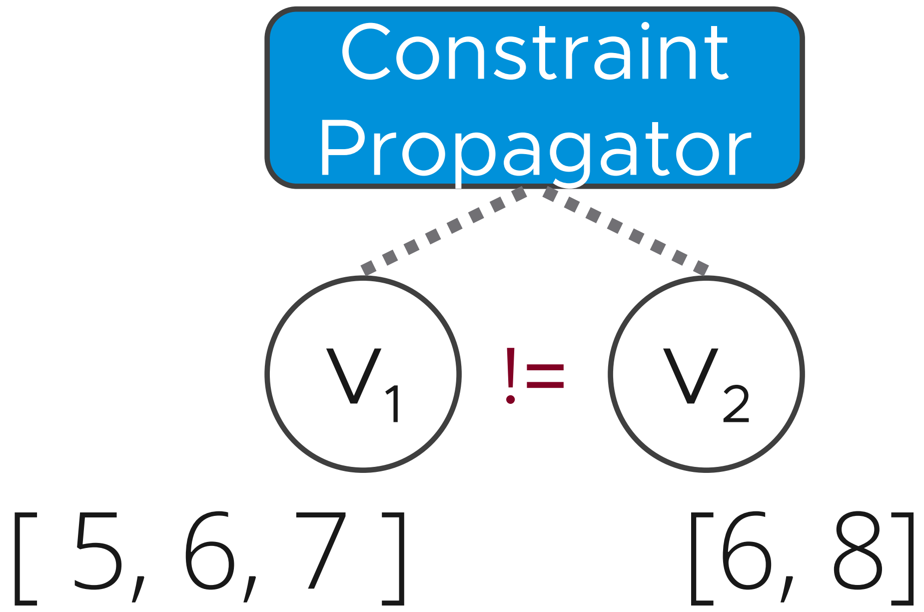
CP = Constraint Programming

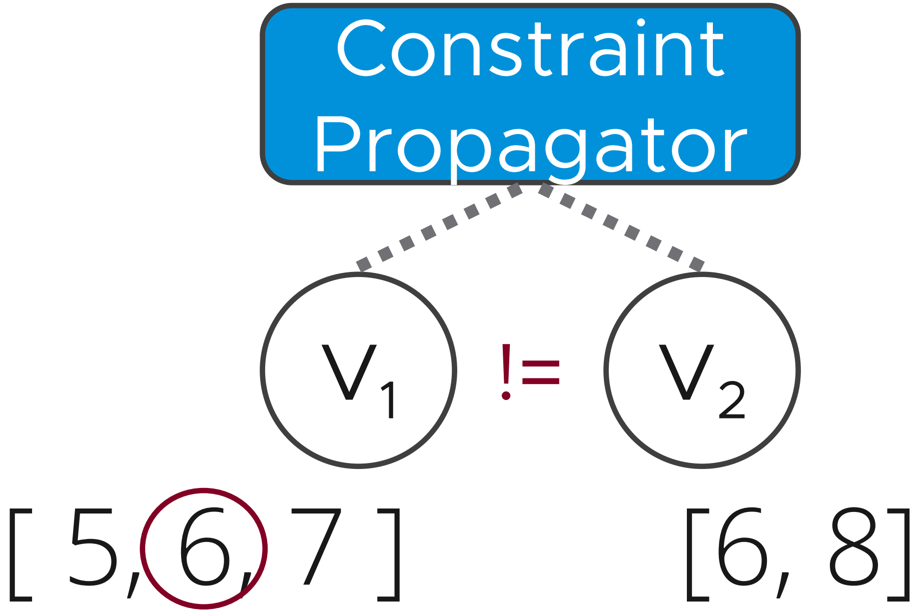
Variables + constraints

Variables V_1 != V_2

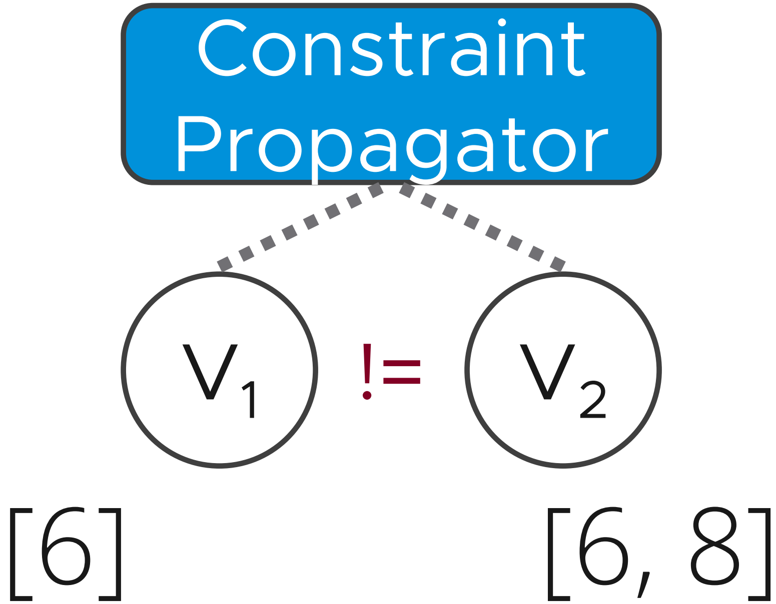
Domain [5, 6, 7] [6, 8]

Constraint

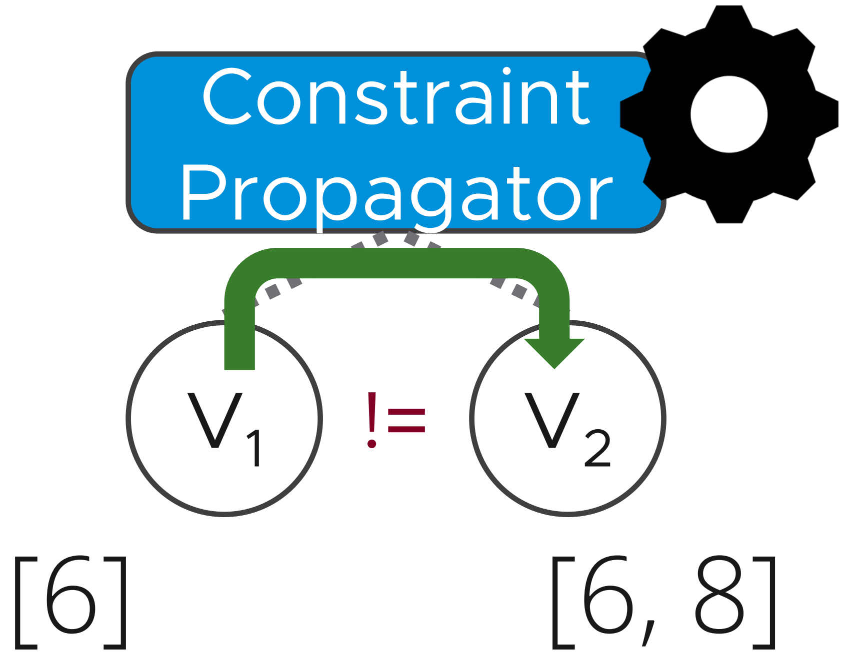

Constraint Propagator
V1
!=
V2
[5, 6, 7]
[6, 8]



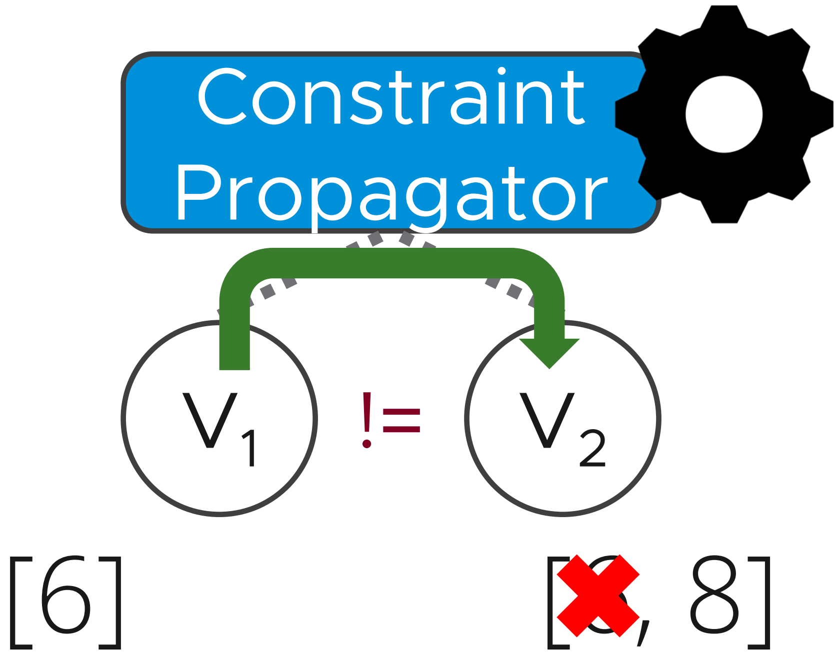
Solver fixes
V_1 to 6



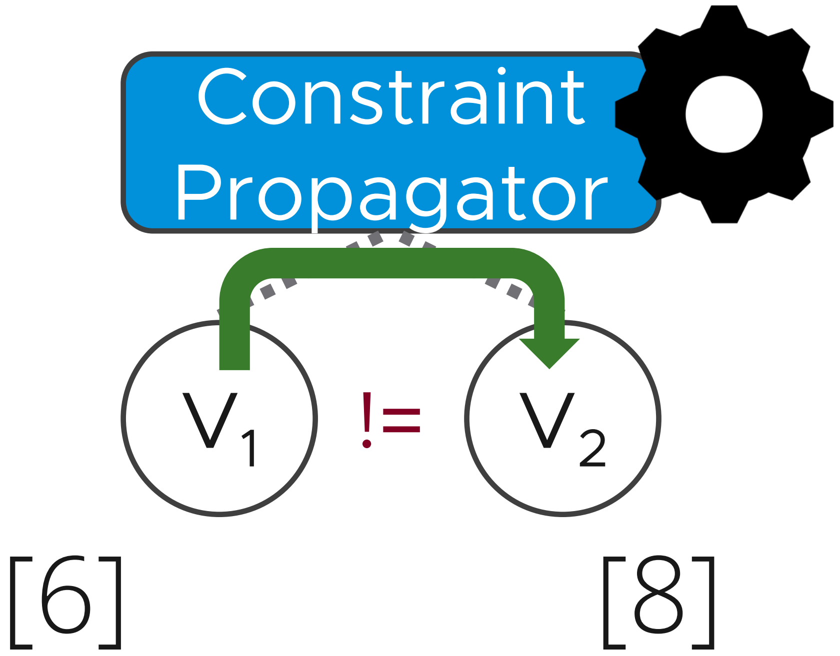
Solver fixes
V_1 to 6



Solver fixes
V_1 to 6



Propagation shrinks V_2 domain



A solution!

Input to CP solver

=

Graph of constraints and variables

(encoding)

Goal
Reduce the number of intermediate variables and constraints

Compiler features

Constant propagation
Common sub-expression elimination
Algebraic Identities
Global constraints

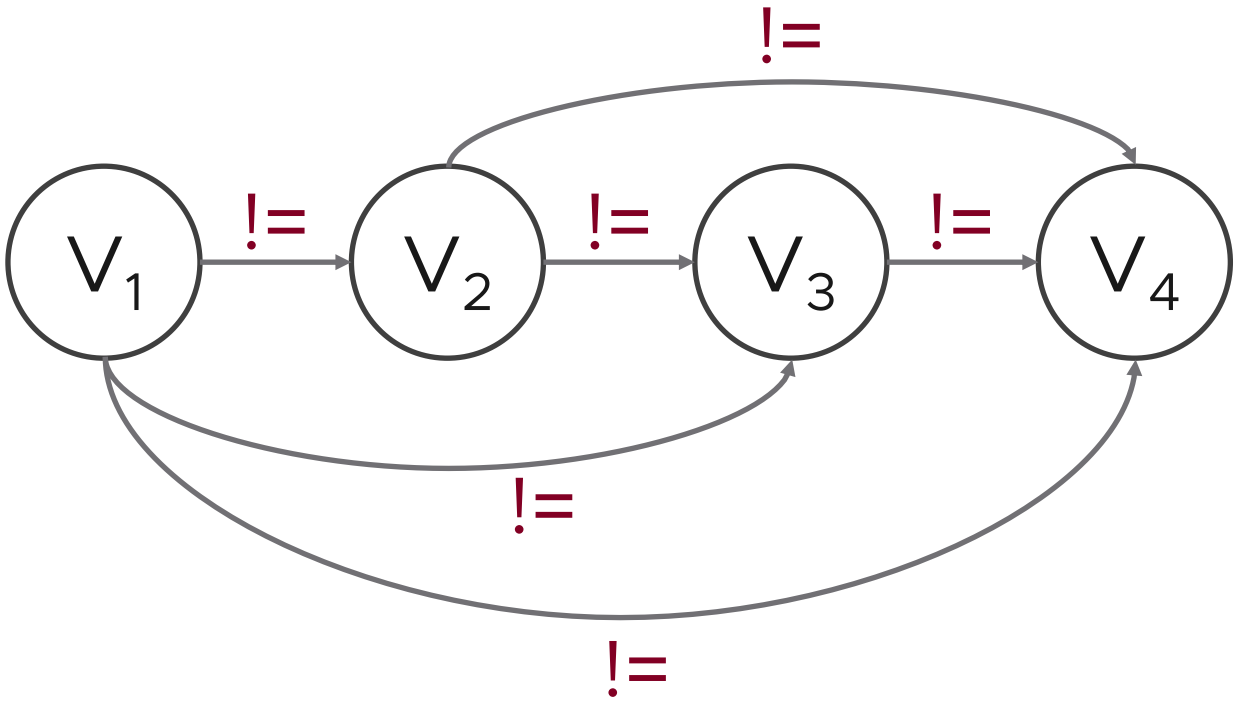
Global constraints

Constraints on groups of variables
Leverages specialized propagation algorithms

Global constraints

Example:

Constraint: ensure (V_1 ,V_2 , V_3, V_4) all take different values



Global constraints

Example:

Constraint: ensure (V_1 ,V_2 , V_3,V_4) all take different values

AllDifferent (V_1 ,V_2 , V_3,V_4)

Other examples:

Cumulative(), NoOverlap(),...

Solver performance is highly sensitive to the encoding

- Reduce number of introduced variables and constraints
- Leverage global constraints

Benchmark
Assign 50 tasks to 1000 workers

Naïve: 25 seconds

With optimizations: 85 ms!

Use Case: Kubernetes scheduler

Building a Kubernetes scheduler

Subscribe to Event Notifications

Without DCM

State in custom data-structures

Reason about single pod, single node at a time

Policies as filter-score passes (imperative)

With DCM

State in 

Reason about multiple pods, multiple nodes at a time

Policies as constraints in SQL (declarative)

Cluster state
Database

```
initialPlacementModel

preemptionModel

deschedulingModel
```

Use database for its strengths

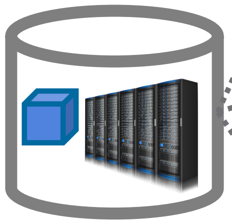
Cluster state
Database

Filter early

(e.g., nodes that are unavailable,
have no capacity)

Preferences

(e.g., least loaded nodes are more preferred)



`initialPlacementModel`

Compute expensive aggregates

(e.g., spare capacity per node,
groups of pods that are affine/anti-affine)

`initialPlacementModel`

Hard and soft constraints that use views computed in DB

Lessons learnt

Most time spent understanding Kubernetes semantics, not writing SQL

Performance engineering: most time spent on views computed in the DB

Incremental view maintenance

UNSAT cores were valuable during development

Evaluation

Use cases

Kubernetes Scheduler	VM Load Balancing Tool	Distributed Transactional Datastore

Scalability | **Decision quality** | **Extensibility**

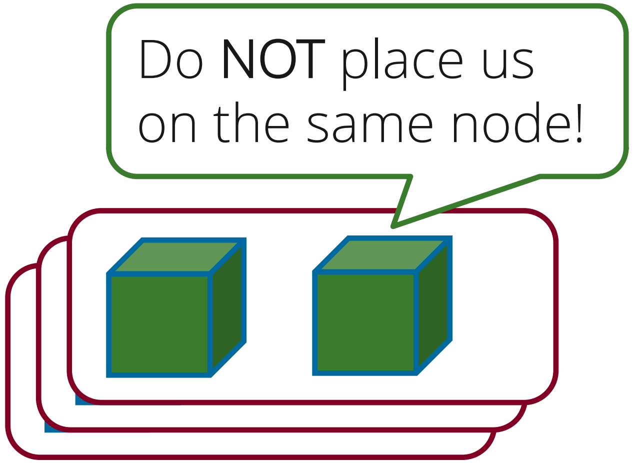
Evaluation

Kubernetes Scheduler

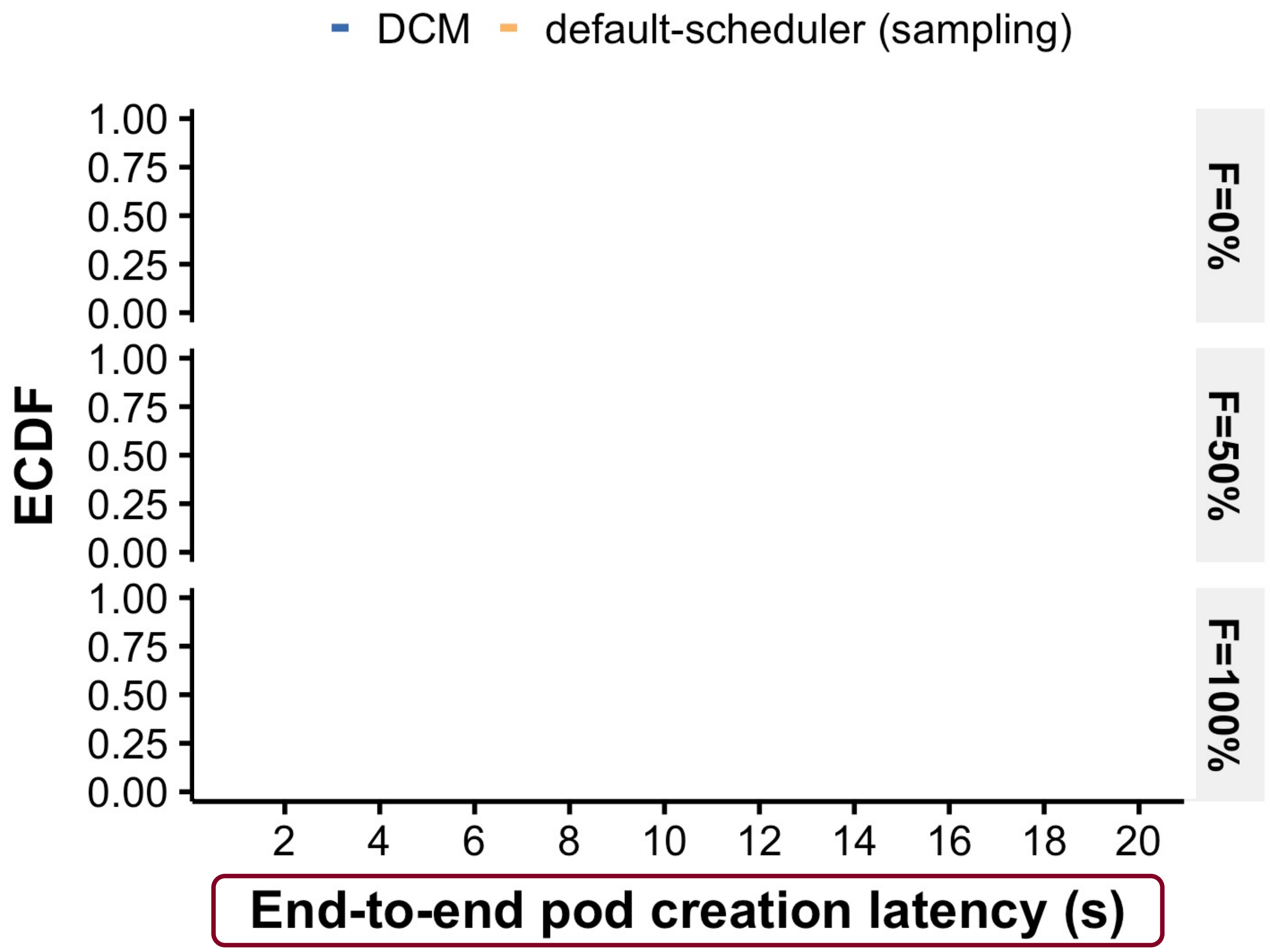
Scalability

- 500 node Kubernetes cluster
- Deploy a series of apps in an open-loop
- Azure 2019 trace
- Inter-pod anti-affinity constraint

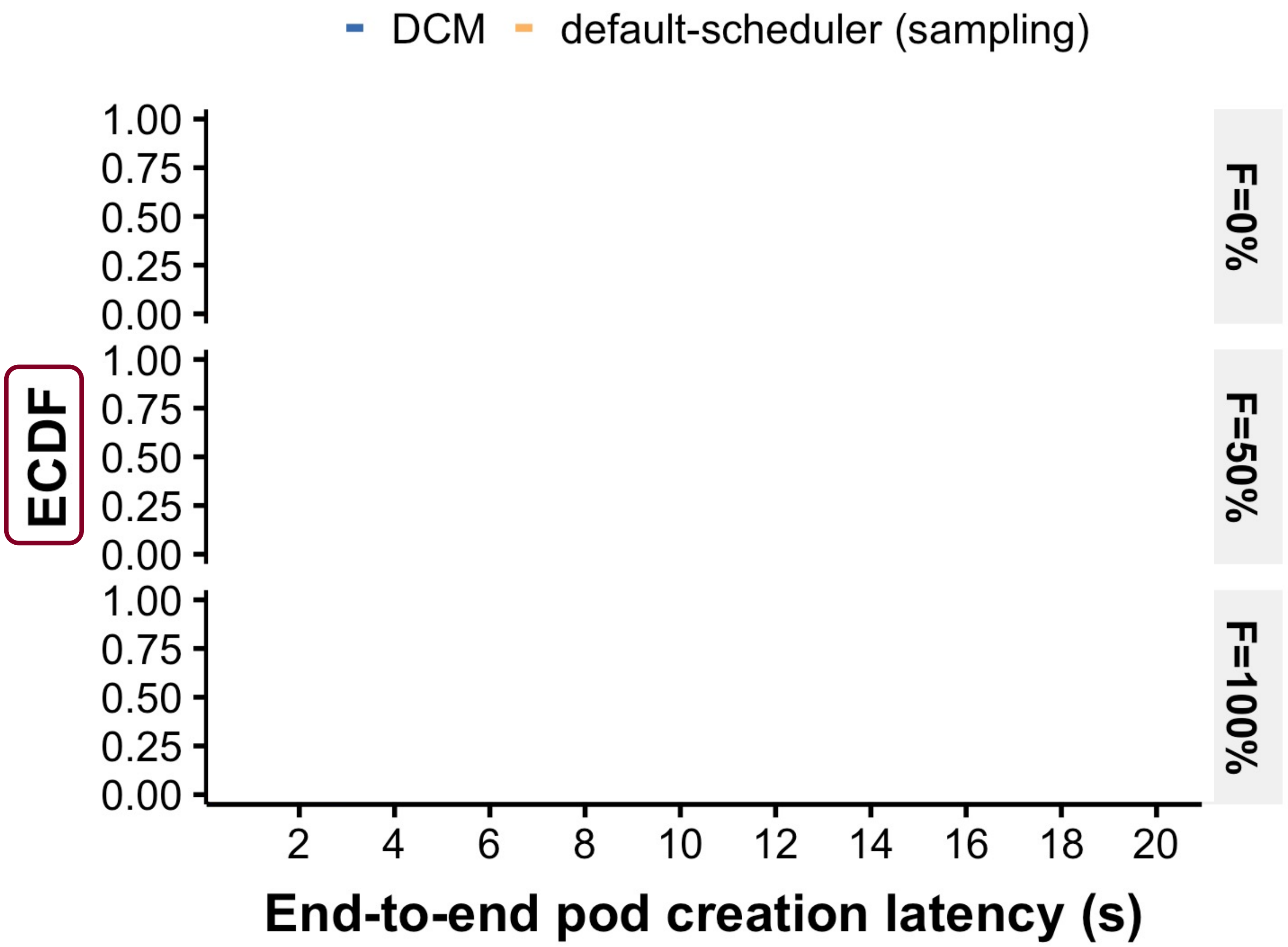
Recommended best practice,
but a challenging constraint



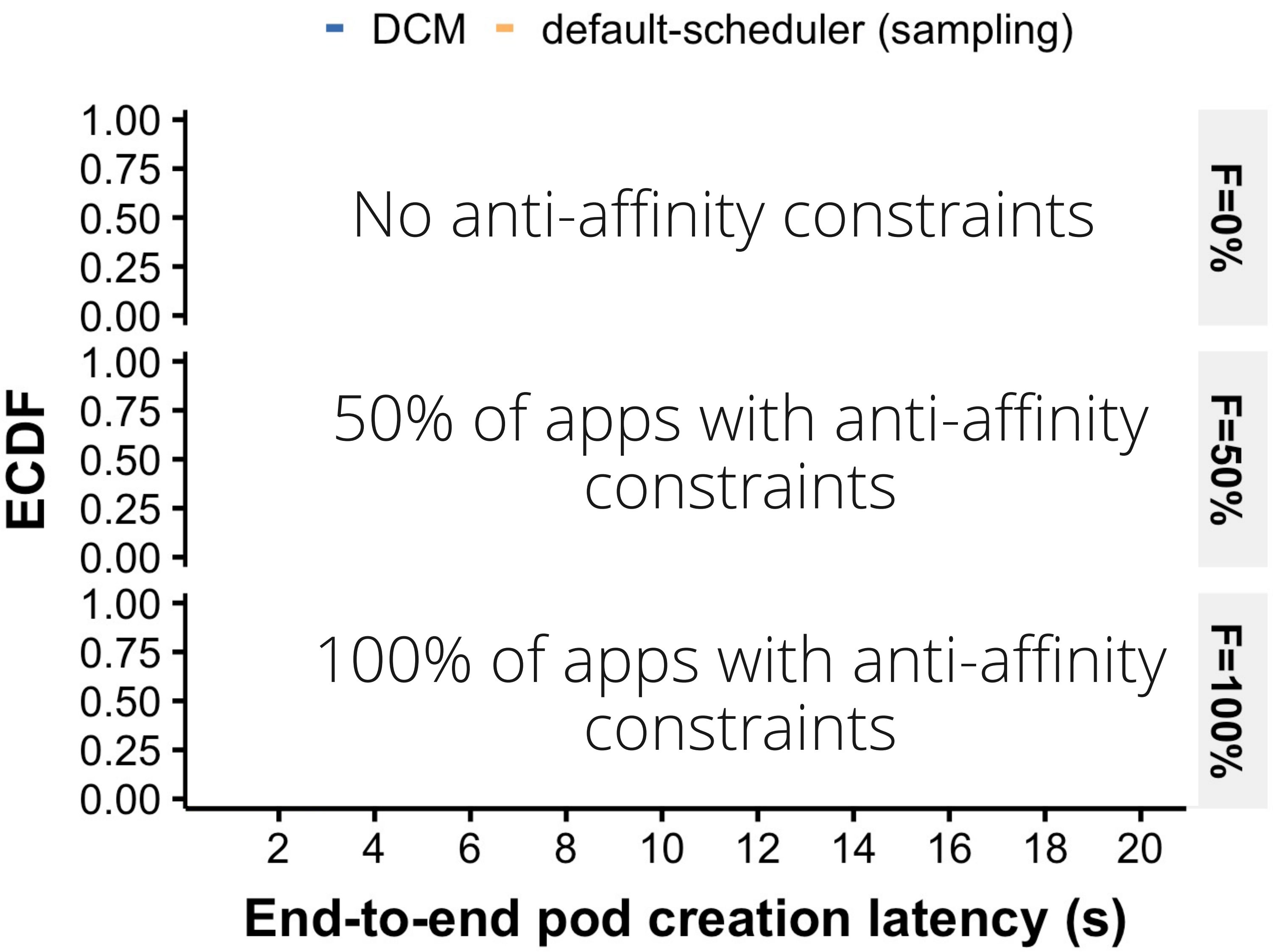
Kubernetes Scalability Evaluation



Kubernetes Scalability Evaluation



Kubernetes Scalability Evaluation

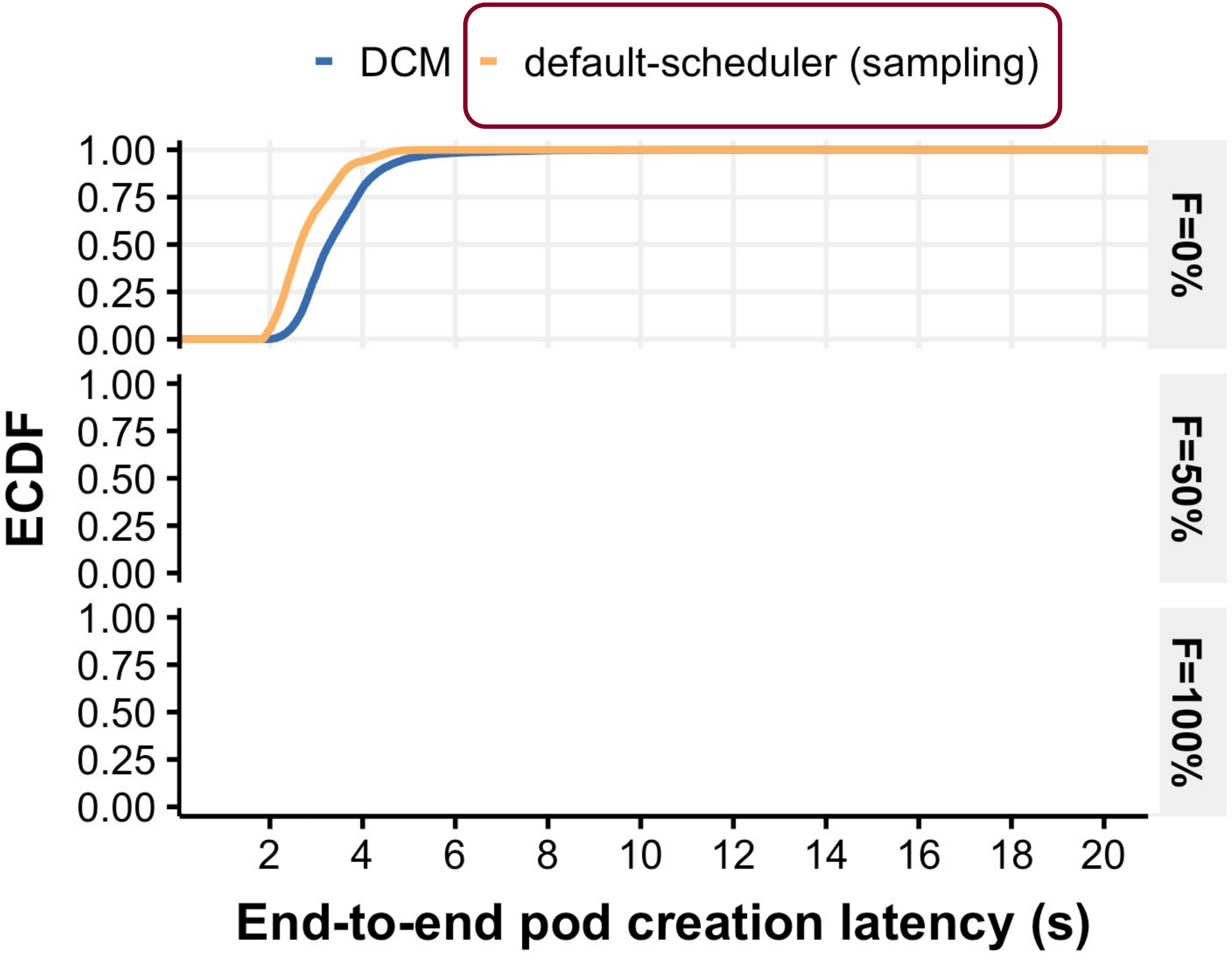


Kubernetes Scalability Evaluation

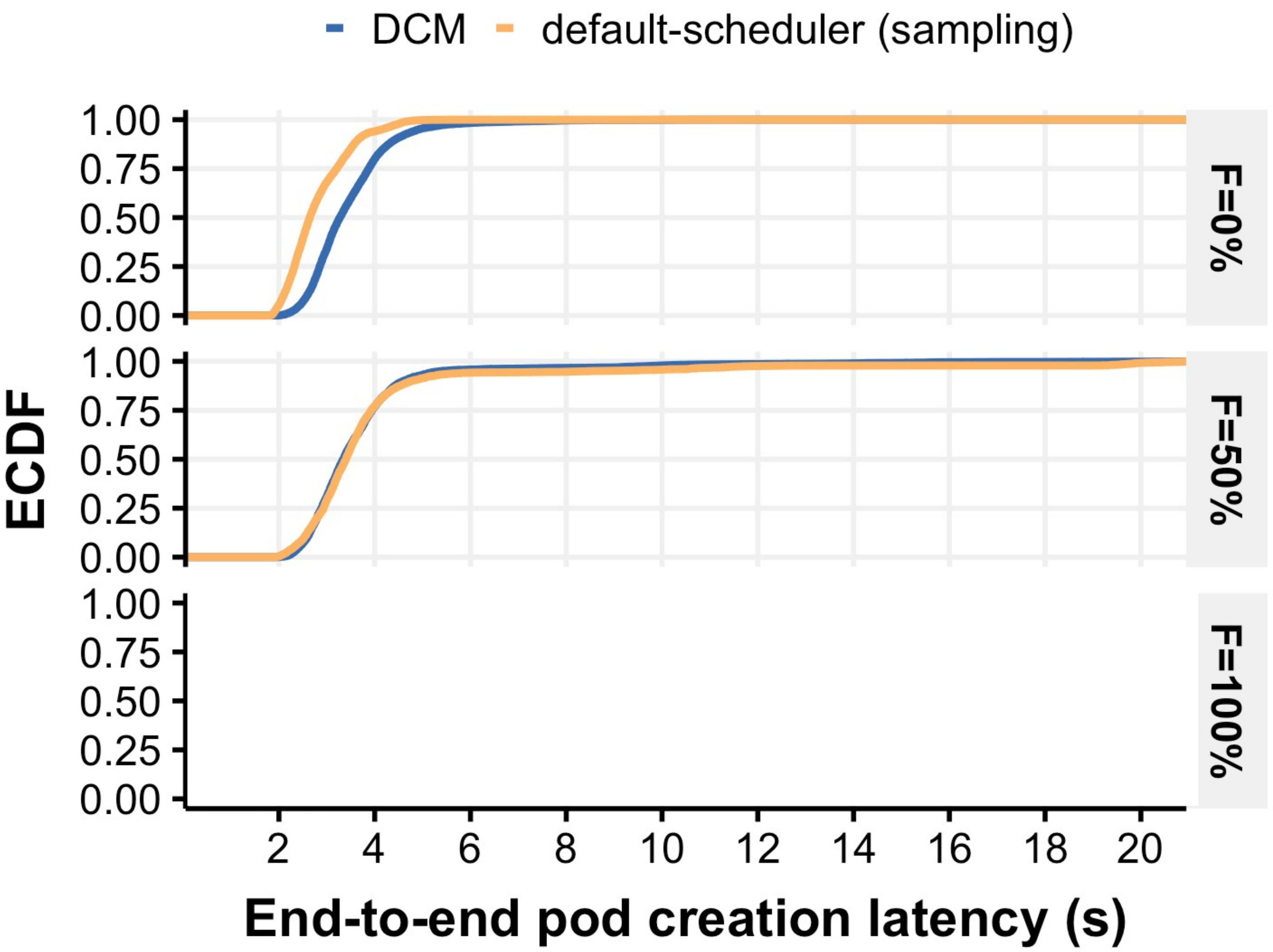
Baseline samples only 50% of nodes

p95 latency

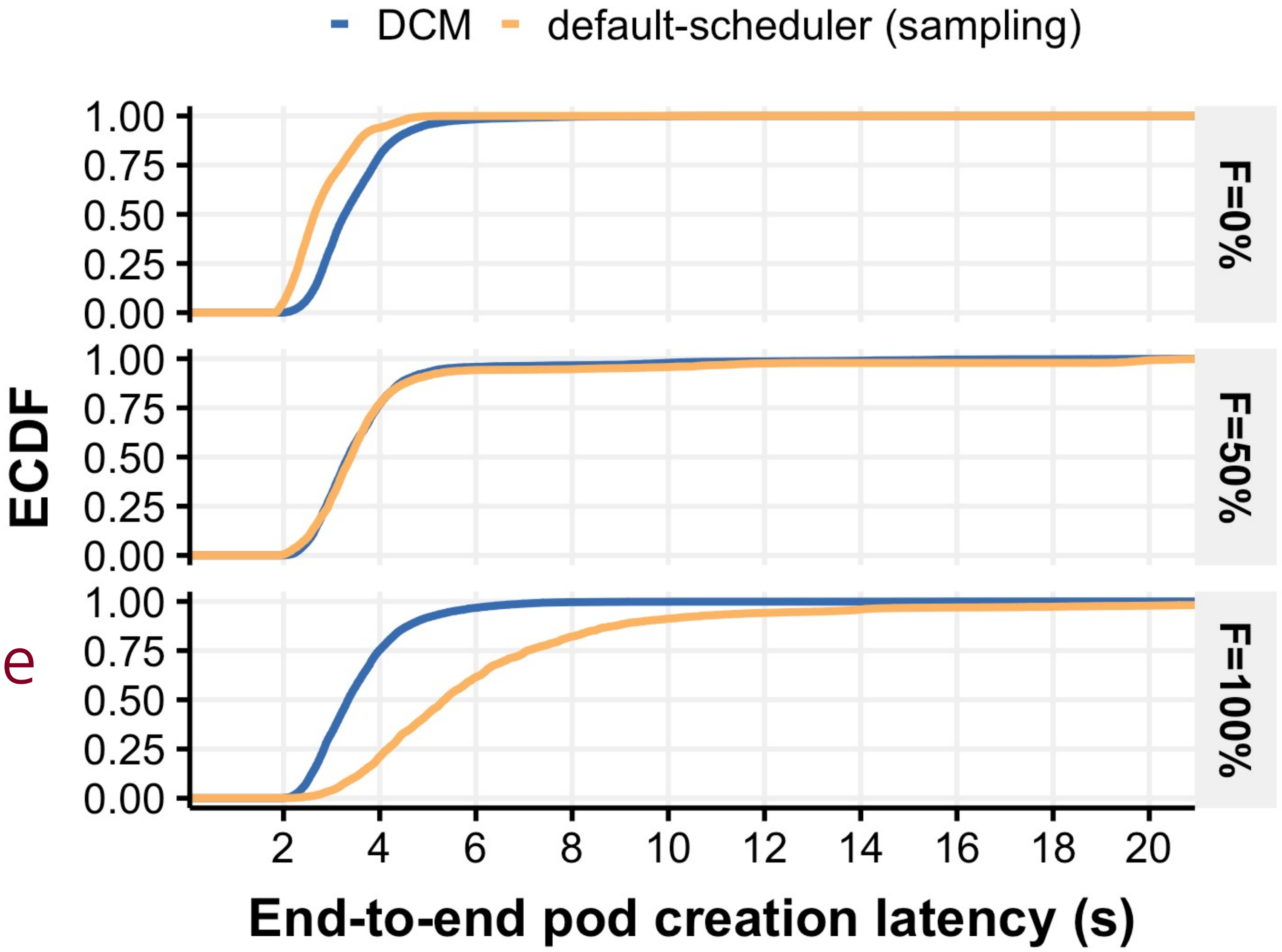
DCM = 5.33s
Baseline = 4.13s



Kubernetes Scalability Evaluation



Kubernetes Scalability Evaluation



DCM cuts 95th percentile latency in half

More details in the paper

Experiments with up to 10K nodes
Latency breakdown
More compiler details

Building Scalable and Flexible Cluster Managers Using Declarative Programming

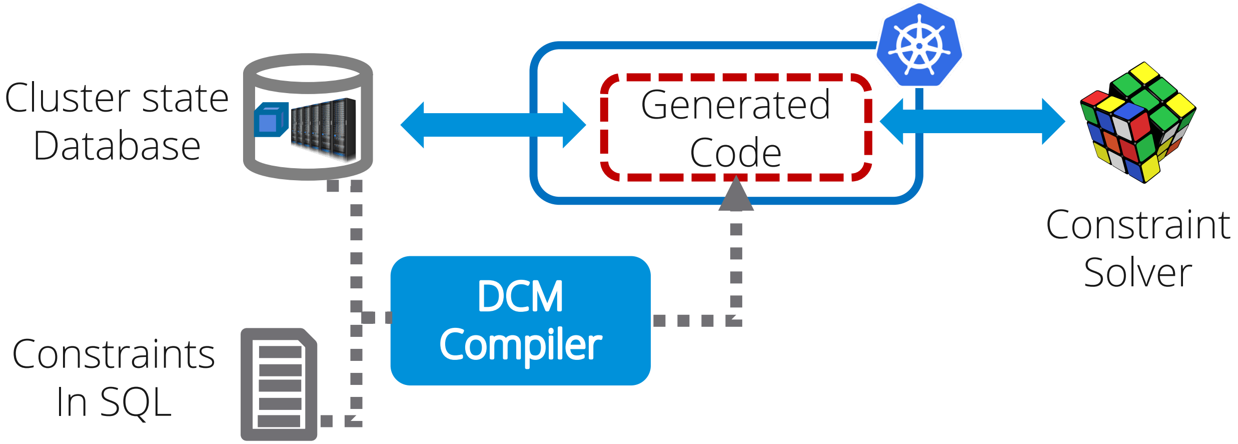
Lalith Suresh, João Loff[1], Faria Kalim[2], Sangeetha Abdu Jyothi[3], Nina Narodytska, Leonid Ryzhyk,
Sahan Gamage, Brian Oki, Pranshu Jain, Michael Gasch
VMware, [1]IST (ULisboa) / INESC-ID, [2]UIUC, [3]UC Irvine and VMware

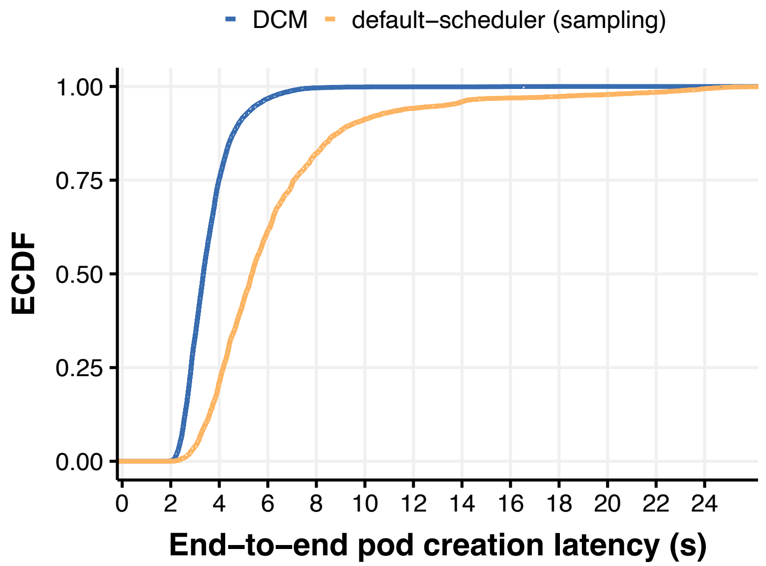
Abstract

Cluster managers like Kubernetes and OpenStack are notoriously hard to develop, given that they routinely grapple with hard combinatorial optimization problems like load balancing, placement, scheduling, and configuration. Today, cluster manager developers tackle these problems by developing

Despite the complexity of the largely similar algorithmic problems involved, cluster managers in various contexts tackle the configuration problem using custom, system-specific best-effort heuristics—*an approach that often leads to a software engineering dead-end* (§2). As new types of policies are introduced, developers are overwhelmed by having to write code to solve arbitrary combinations of increasingly

Kubernetes Scheduler	VM Load Balancing Tool	Distributed Transactional Datastore
Scalability	Decision quality	Extensibility





Open source under a BSD-2 license

https://github.com/vmware/declarative-cluster-management/