Placing Databases @ Uber

Casper Kejlberg-Rasmussen,
Software Developer, Uber

UBER

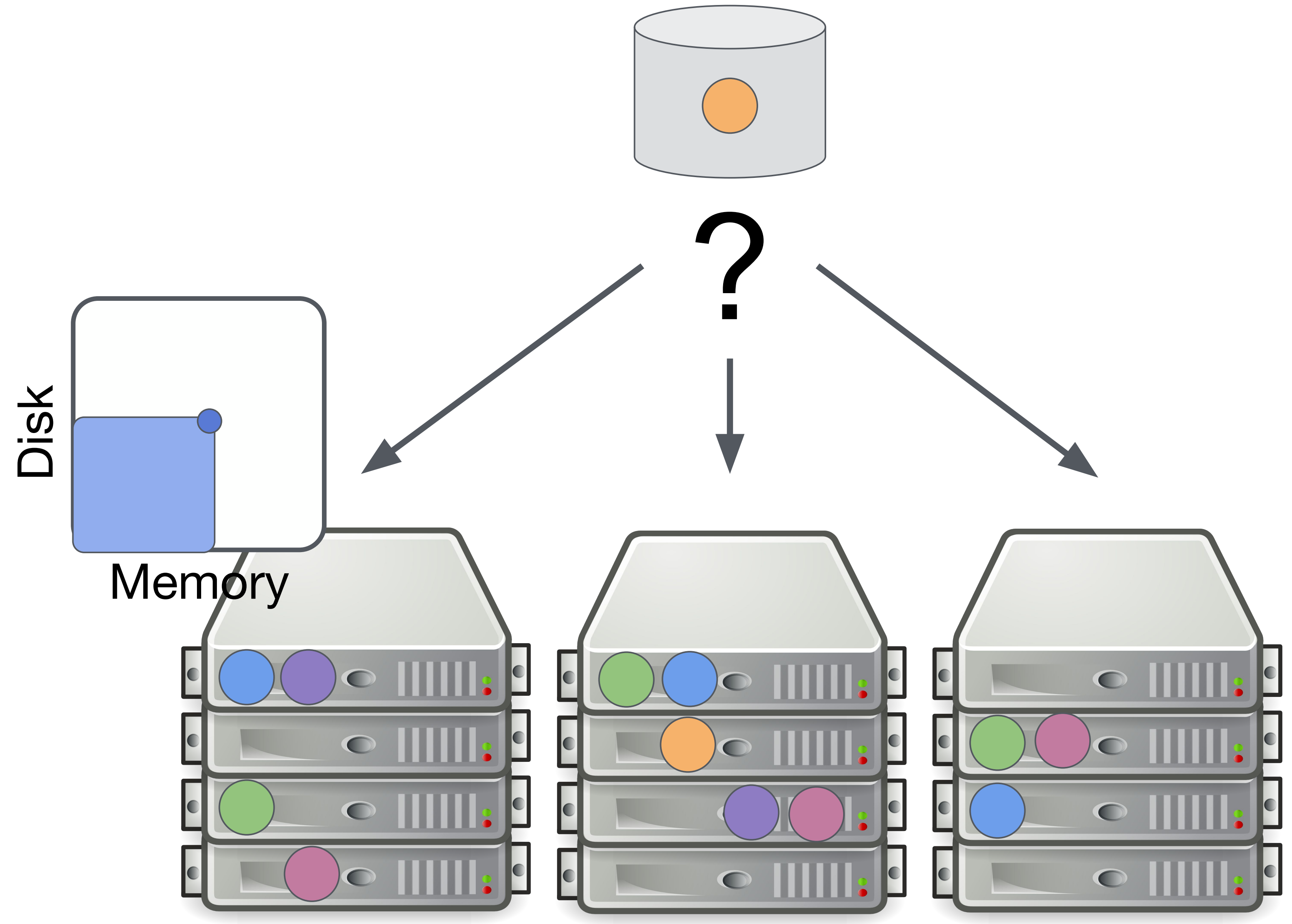
April, 2017

What do you mean by “placing databases”?

Why is it important where you place your databases?

Things we need to consider when deciding where to run a database:

- Does the host have enough resources to run the database?
- Should the host be avoided because of maintenance or some other issue?
- What other databases are already running on the host? Will using the host affect the cluster reliability?



Circles of the same color represents databases belonging to the same replication topology

Overview

What I will cover today

- Motivation and Background
- Core Problem
- Architecture
- Constraint Modelling
- Placement Algorithm
- Relocation Algorithm
- Simulations

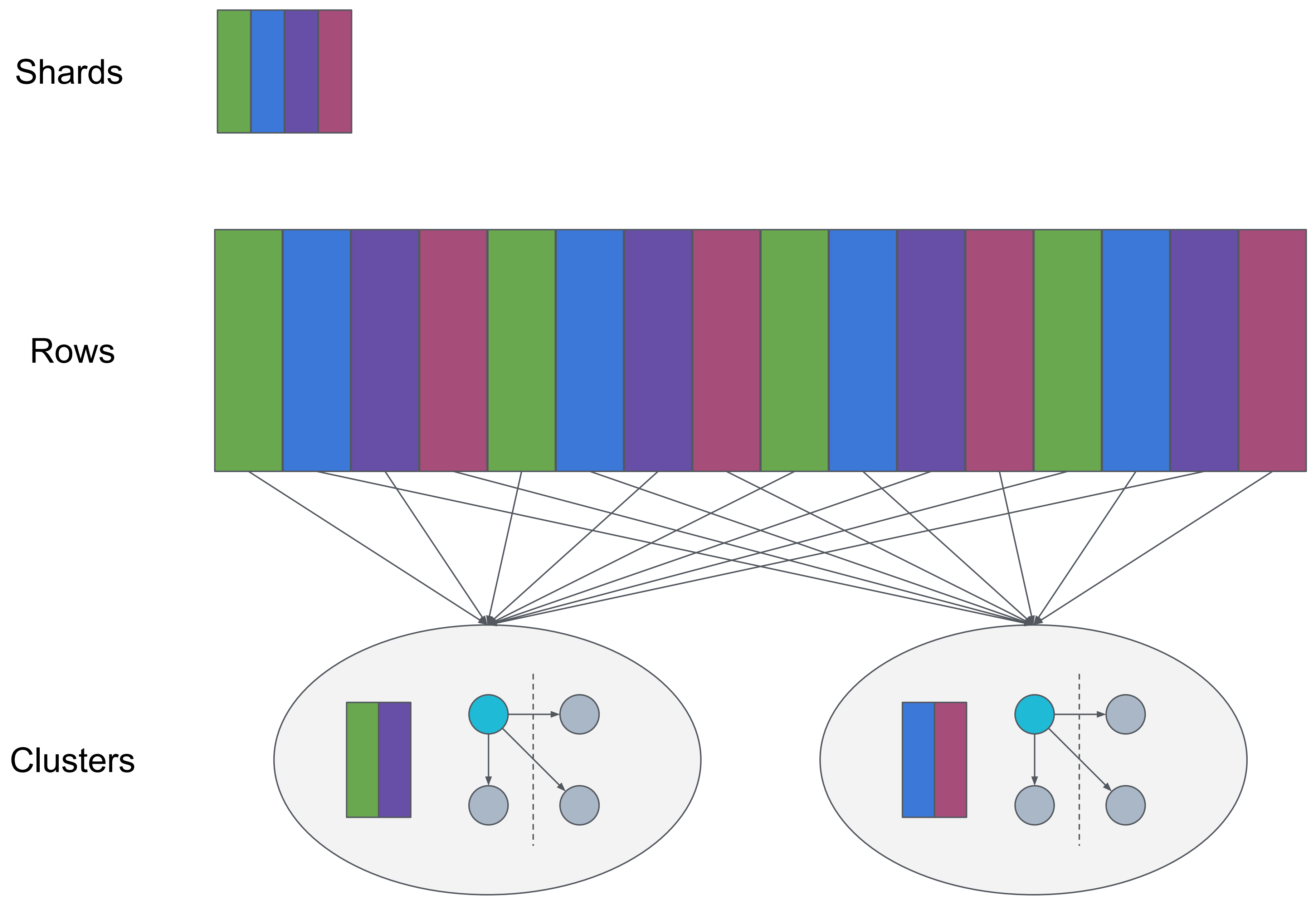
Motivation and Background

An overview of Schemaless and Opsless

Schemaless a NoSQL database

Overview and Database Topology

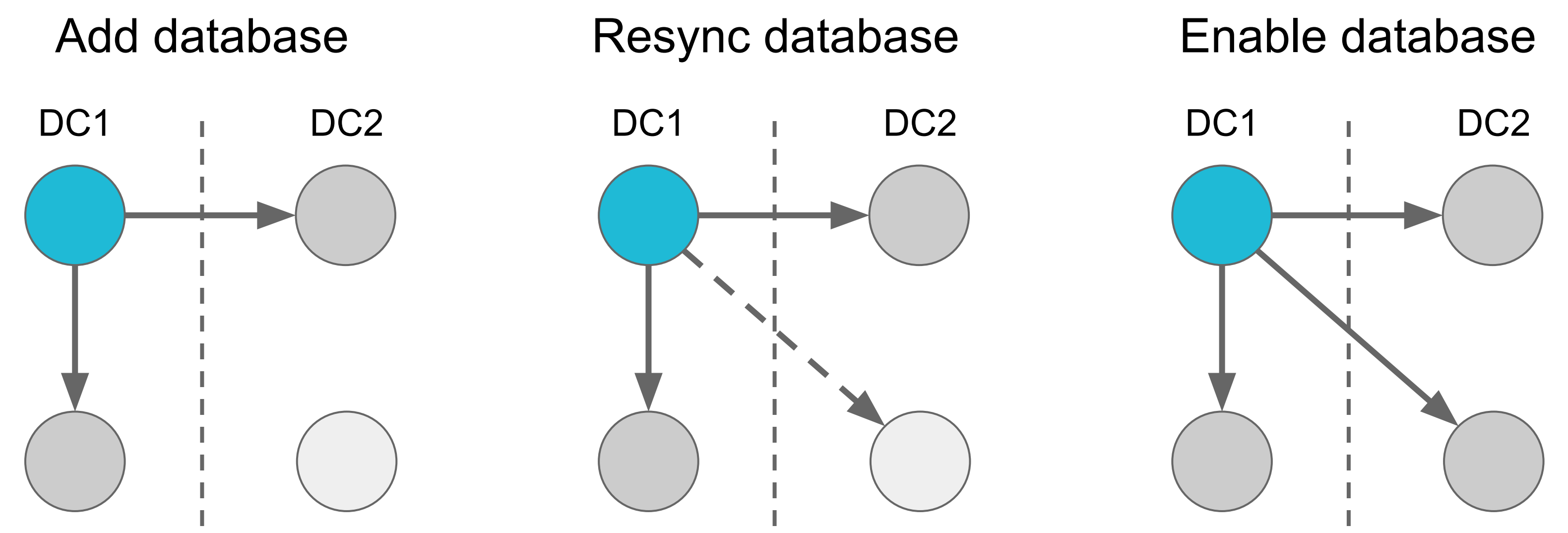
- Schemaless an in-house NoSQL key-value database with secondary indexes
- Simple API
    - String UUID String Timestamp JSON
      Insert(datastore, row-key, column-key, ref-key, cell)
    - String UUID String Timestamp
      Get(datastore, row-key, column-key, ref-key)
    - String String FieldConstraints
      Query(datastore, column-key, constraints)
- Topology of a Schemaless instance
- Schemaless is Ubers most popular storage technology
- Popularity means many databases to manage :(



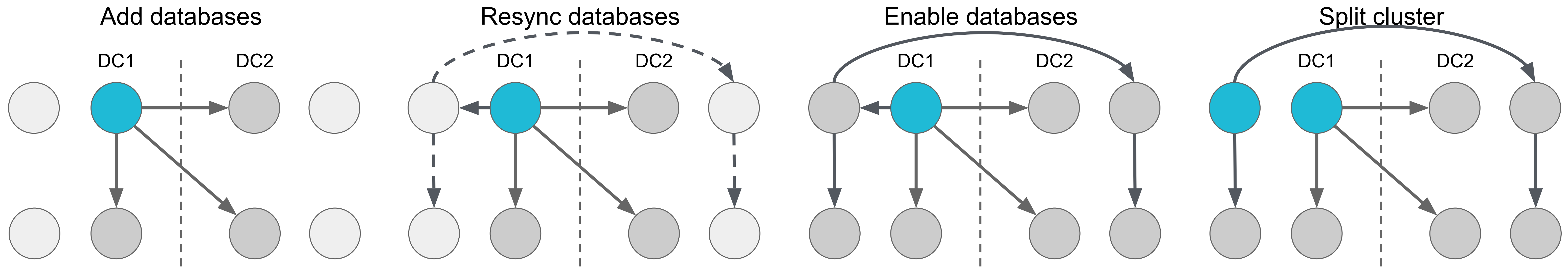
Opsless for Managing Schemaless

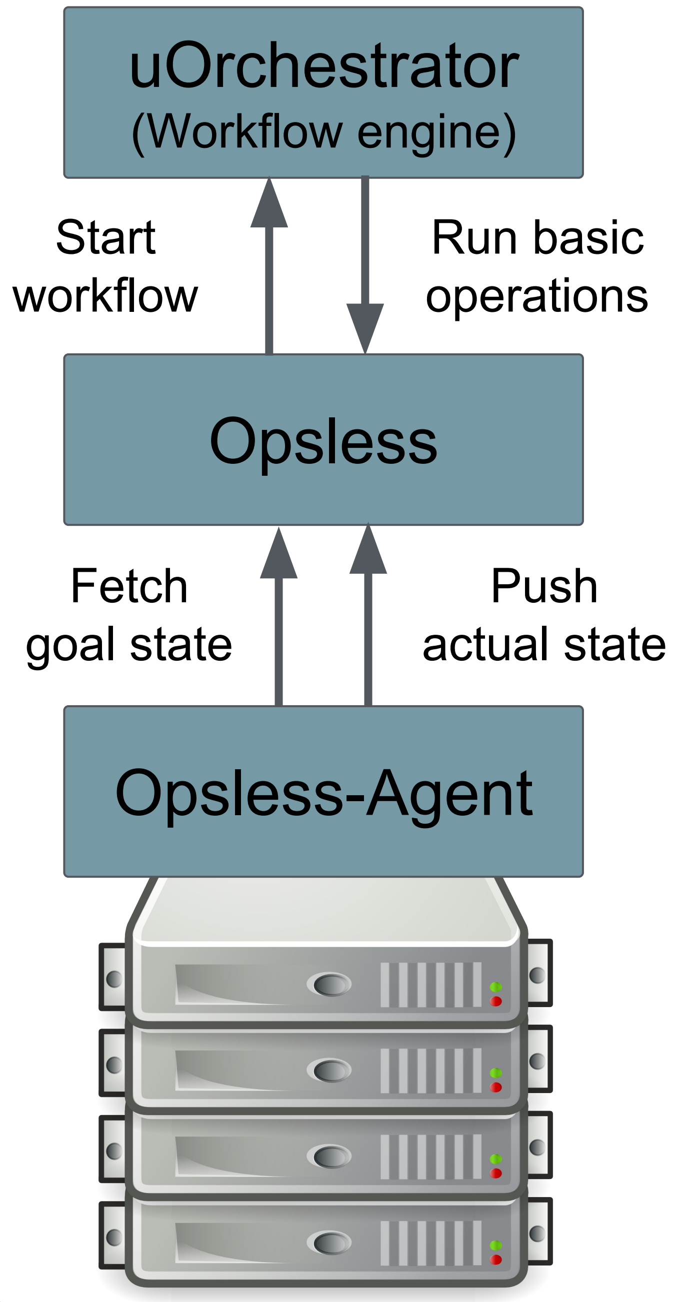
Overview of how Opsless manages Schemaless

- Containerized databases using Docker
- Goalstate driven operations
- Operations performed through workflows that build upon idempotent basic REST operations
- Example: adding a database



- Example: splitting a cluster





Core Problem

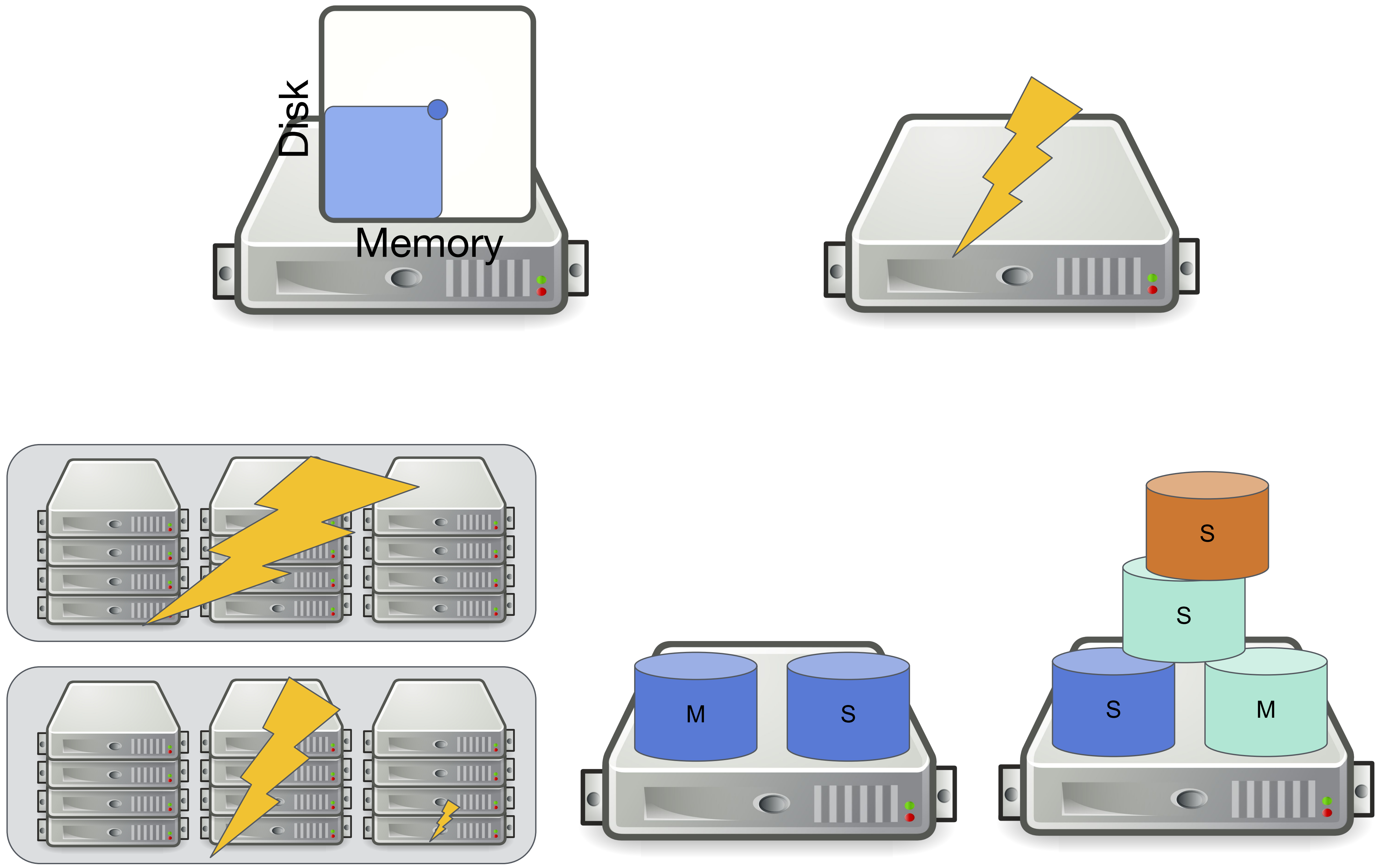
How do we characterize good placement?

Core Problem

How do we characterize good placement?

When deciding to place a database we have some “hard” and some “soft” requirements:

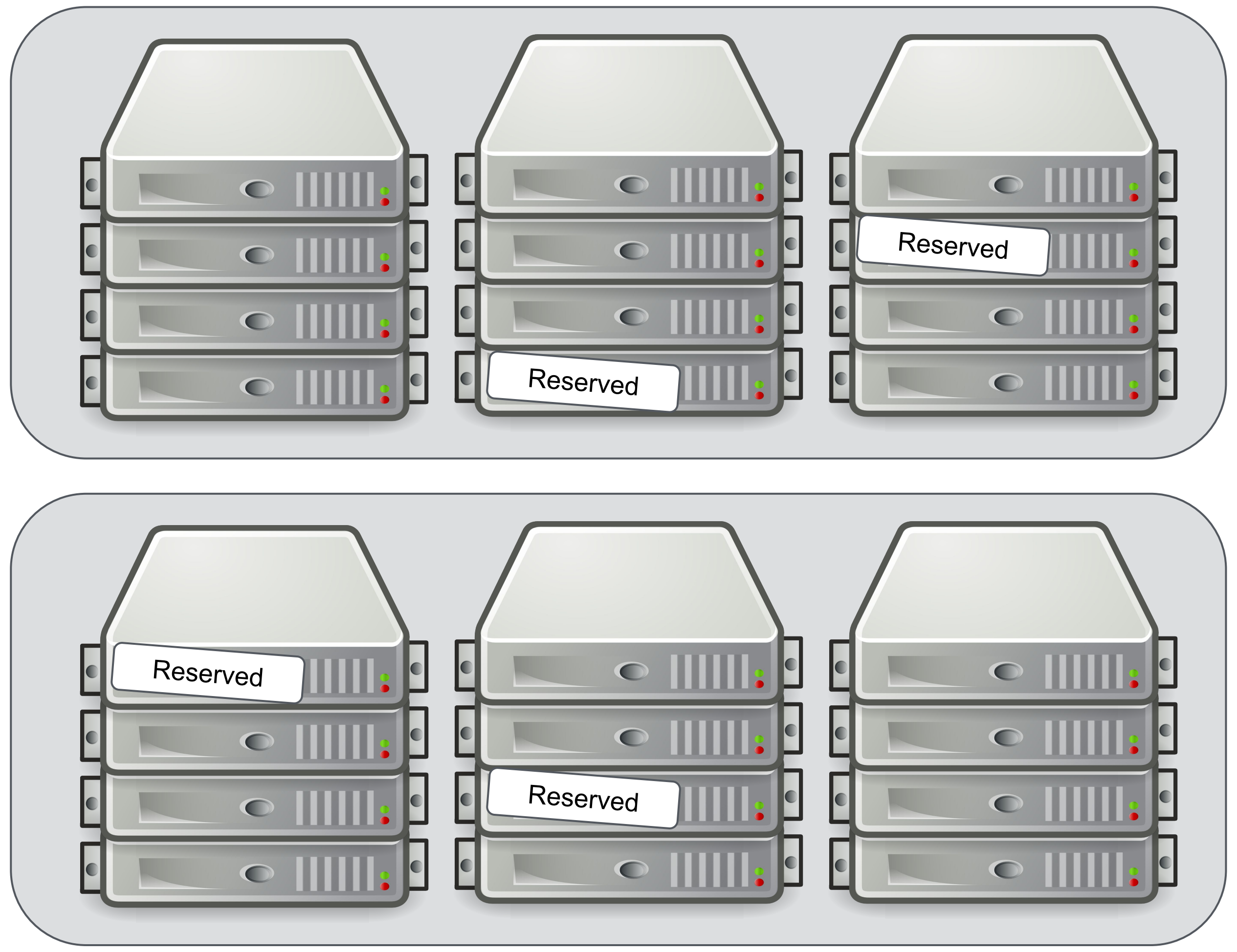
- Hard:
  - Resource constraints: memory, disk, etc.
  - Host issues: bad disk, wrong OS version, etc.
- Soft:
  - Databases spread out to minimize harm from host, rack or datacenter failures
  - Limit same instance databases on a host
  - Limit databases on a host



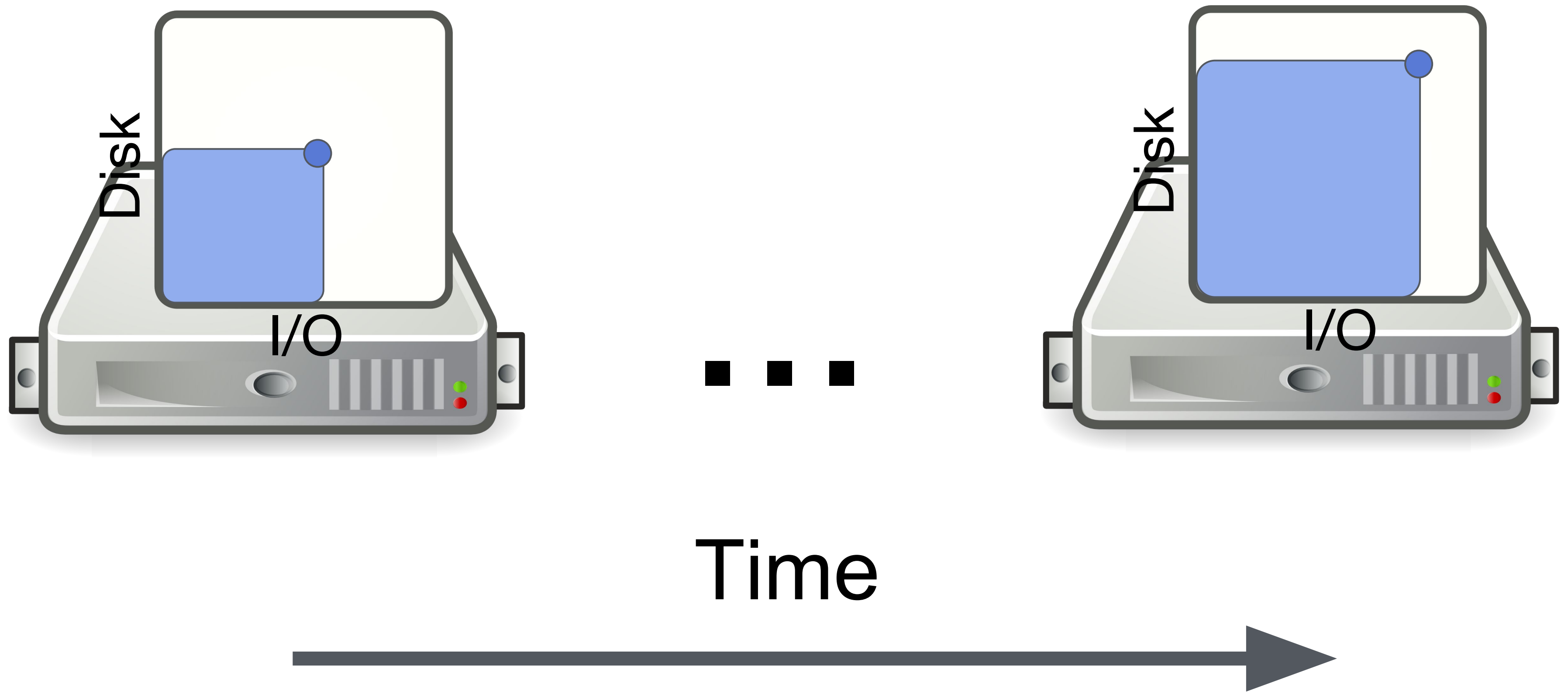
Core Problem

How do we characterize good placement?

- When we have placed a database we need to keep a reservation of its placement



- As time goes on databases consume more disk, I/O operations, etc. so we probably want to relocate/move them

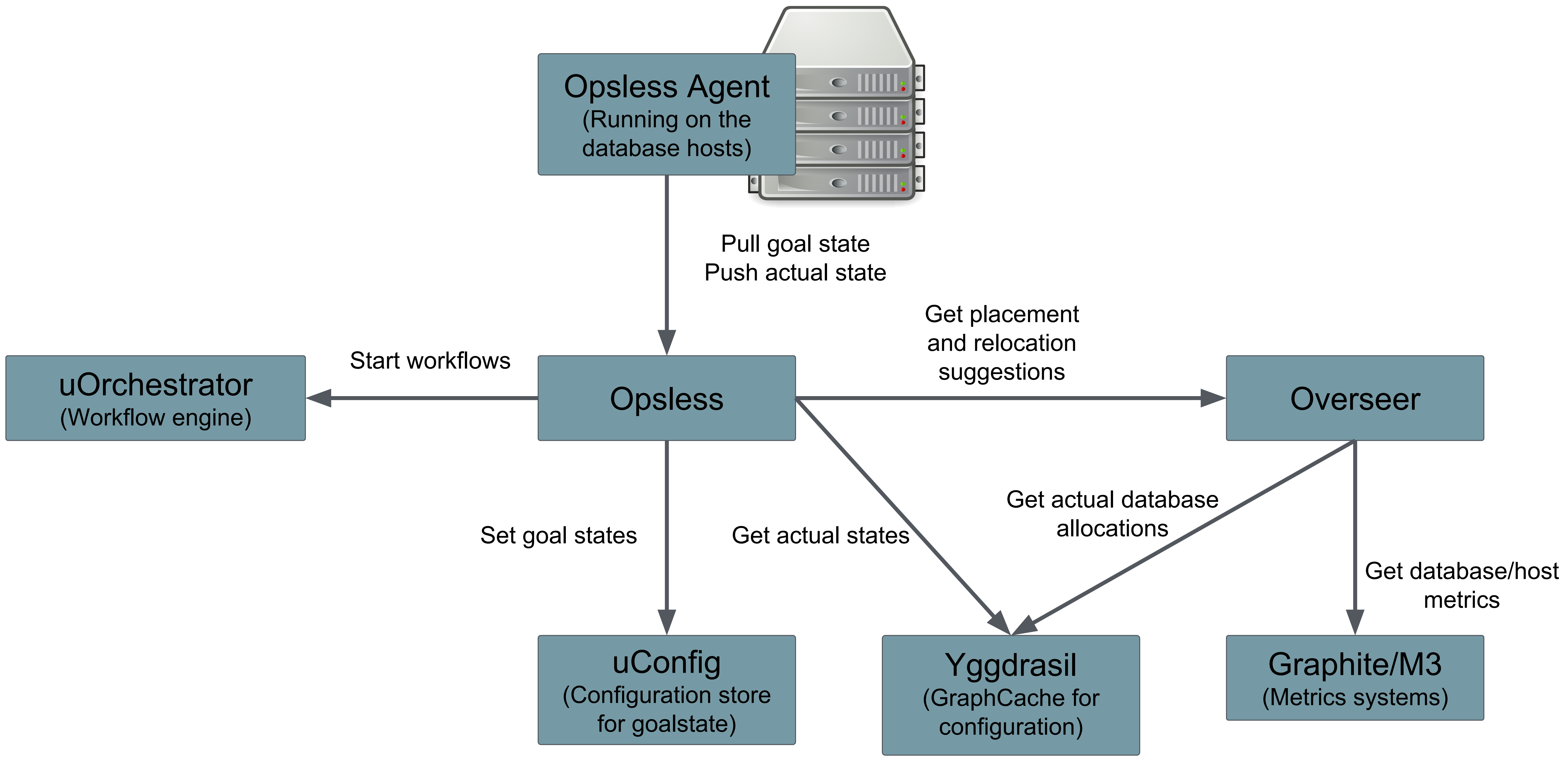


Architecture

The big picture of placing databases

Architecture

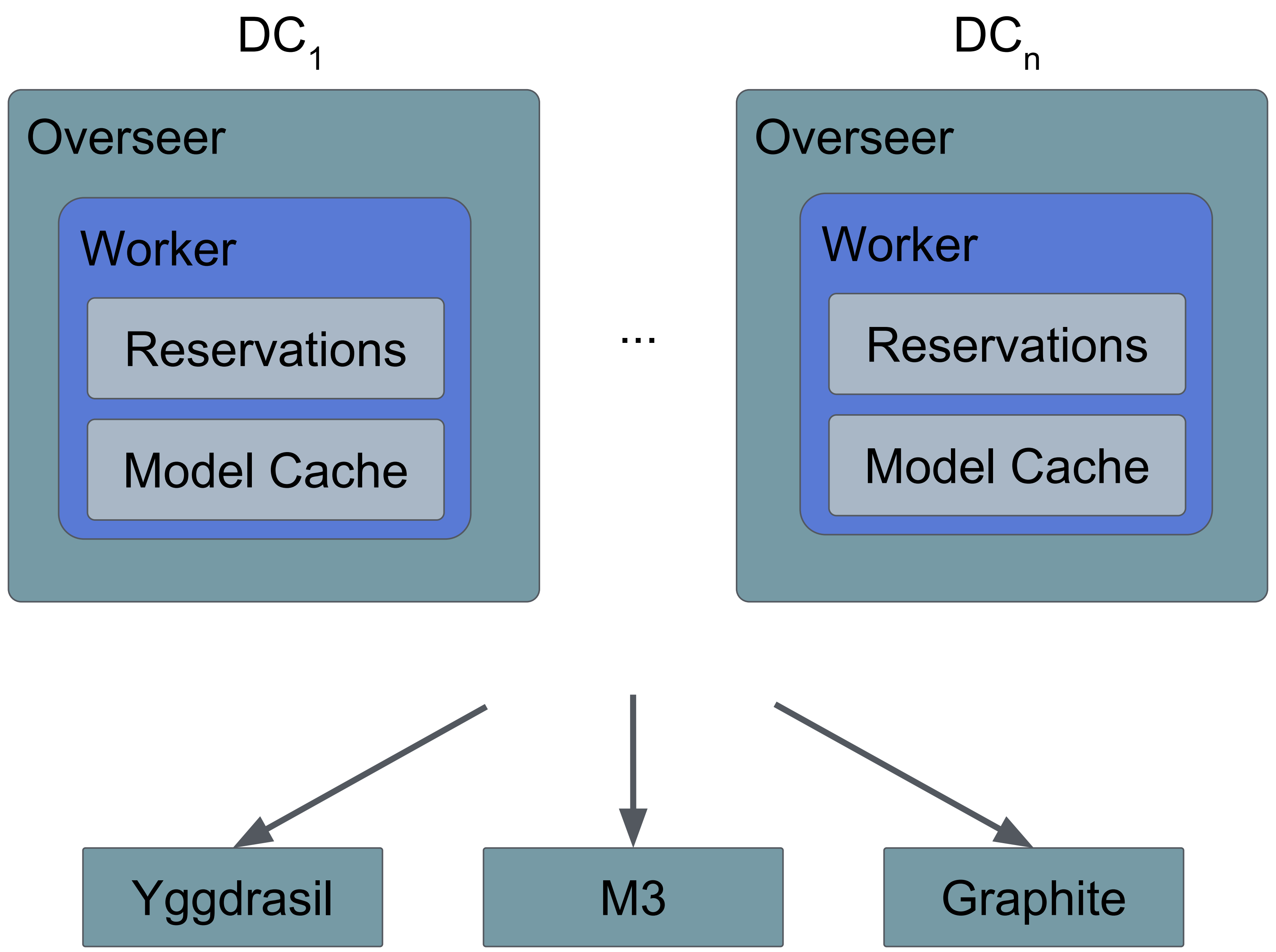
The big picture of placing databases



Architecture

The different components of Overseer

- Written in Go
- Exposes a HTTP-REST API
- Depends on the services
  - Yggdrasil - for finding datacenters, racks, hosts and databases that are currently in production
  - M3 & Graphite - for fetching metrics about hosts and databases used in the placement algorithms
- Keeps a cache of internal data models build using the data from the depencies
- Keeps a set of reservations of databases placed on hosts until changes are visible in Yggdrasil



Architecture

The different components of Overseer

- The HTTP-REST API supporting
  - Suggest-Placement

```json
{
  "databases": [
    {
      "name": "somestore-us1-cluster5-db3",
      "creation_time": "2017-01-01T12:00:00Z",
      "relations": {...},
      "resource_requirements": [...],
      "placement_requirements": [...],
      "relation_requirements": [...]
    }
  ],
  "skip_reservations": false,
  "skip_transcripts": true
}
```

Architecture

The different components of Overseer

- The HTTP-REST API supporting
  - Suggest-Placement
  - Suggest-Relocation

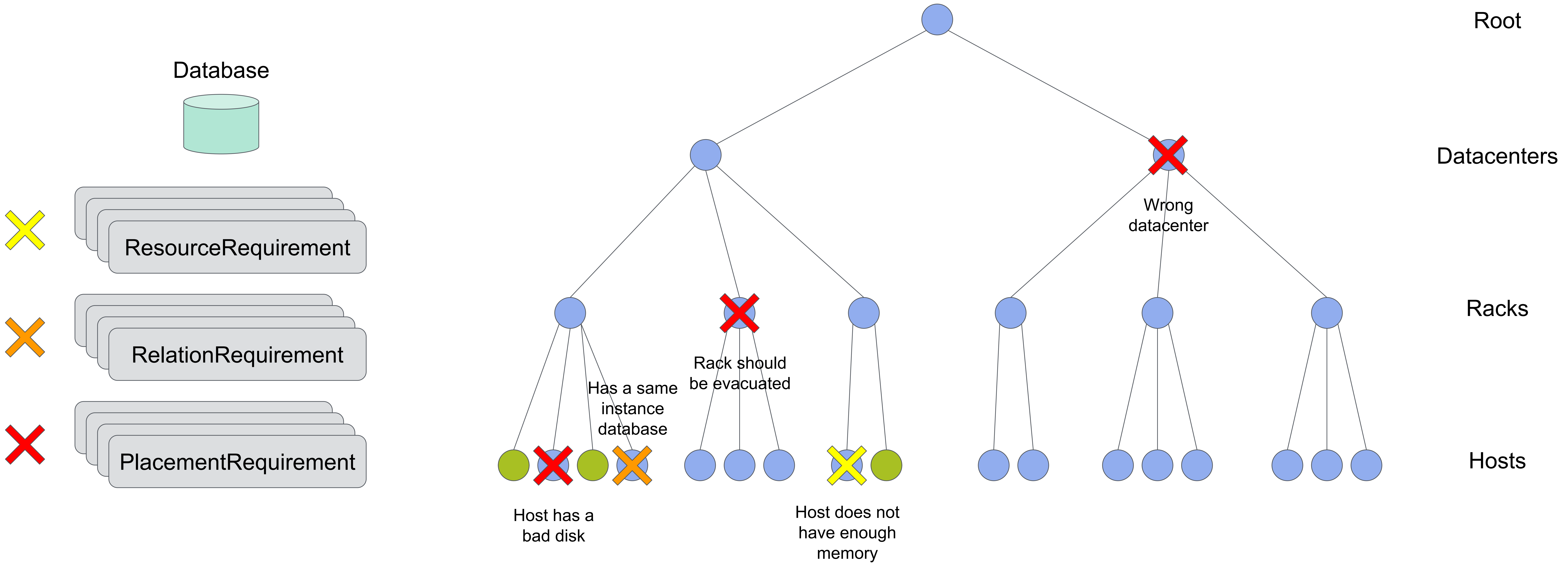

```json
{
 "group_name": "dc1",
 "minimum_rank": 1,
 "minimum_age": "168h"
}
```

Placement Algorithm

How the placement of databases is done

Placement Algorithm

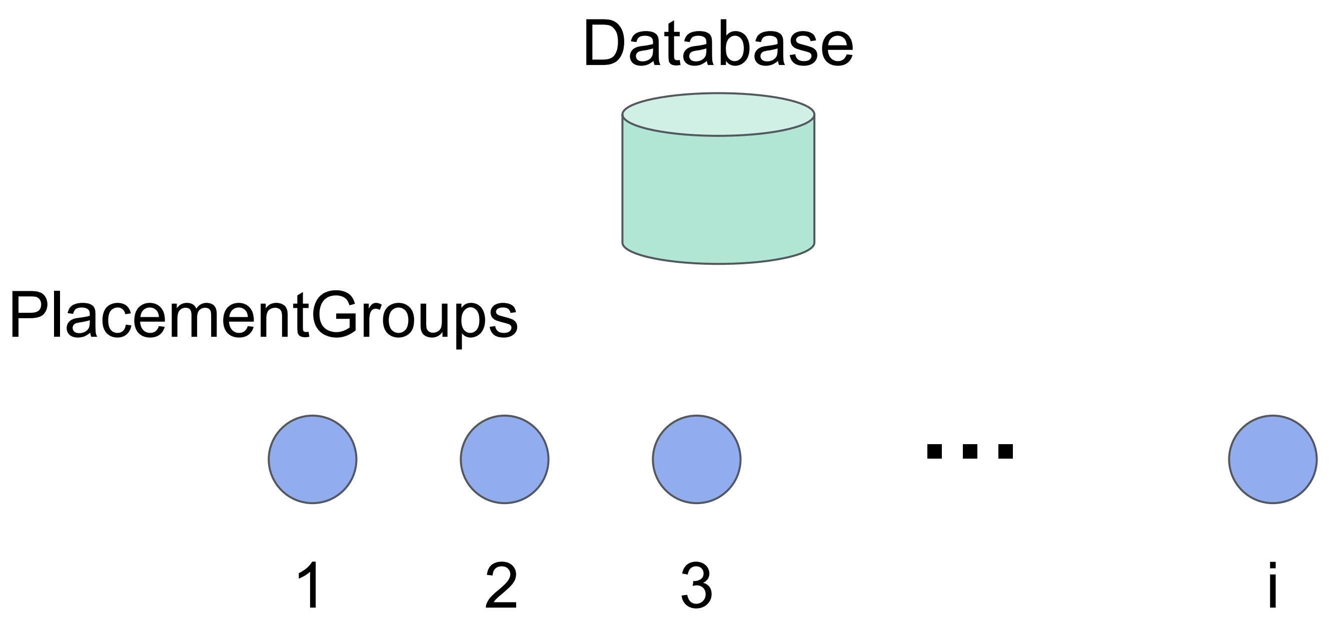
Filtering out PlacementGroups that pass the requirements



Placement Algorithm

Ordering the PlacementGroups that passed the requirements

- A PlacementOrdering is a ternary-relation:
  - PlacementGroup × PlacementGroup × Database →bool



- It must define a Strict Total Order:
  - Irreflexivity: a < a is always false
  - Transitivity: If a < b and b < c then a < c

```
type PlacementOrdering interface {
  Less(pg1, pg2 *PlacementGroup, db *Database) bool
}
```

Compare tuples for each group:

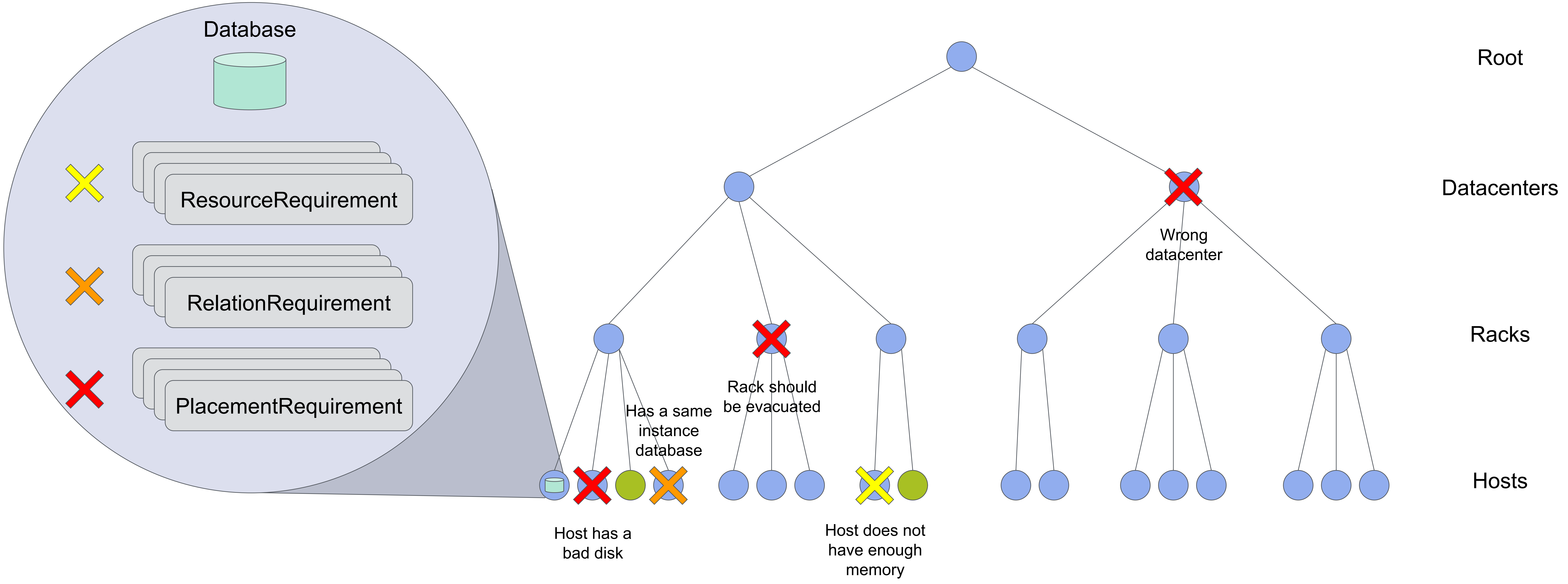
1. # Same cluster databases on rack
2. - Free disk space
3. - Used disk space
4. # Same instance databases on rack
5. # Databases on rack
6. - Free memory
7. - Used memory

Relocation Algorithm

How the relocation of databases is done

Relocation Algorithm

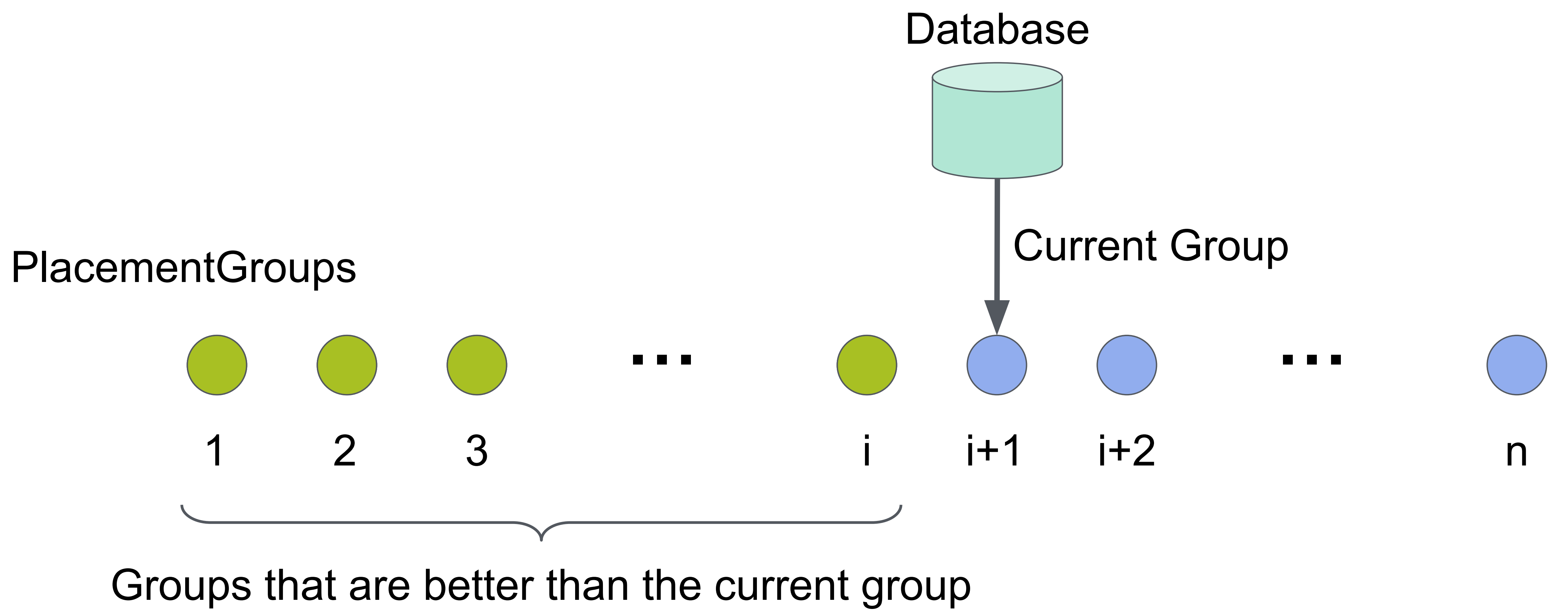
Filtering out PlacementGroups that pass the requirements



Relocation Algorithm

Ranking the current PlacementGroup against the others

- We again use the PlacementOrdering
  - PlacementGroup × PlacementGroup × Database →bool



- The relocation rank of the database is then *i* as there are *i* other groups that are better than the current

Modelling

The central constraint concepts in Overseer

Modelling

Overview of the constraint types

PlacementGroup

represents a datacenter, rack, host or other physical entity which can potentially contain a hierarchy of other physical entities.

```go
type PlacementGroup struct {
  Name      string
  Parent    *PlacementGroup
  Subgroups []*PlacementGroup
  Labels    *LabelSet
  Relations *LabelSet
  Resources *MetricSet
  Usage     *MetricSet
  Databases []*Database
}
```

Modelling

Overview of the constraint types

MetricType

represents a type of information, it can be cpu usage, memory usage, disk usage, etc.

```
{
 "aggregation": "sum",
 "unit": "bytes",
 "name": "disk_free"
}
```

```
{
 "aggregation": "sum",
 "unit": "bytes",
 "name": "memory_free"
}
```

```
type MetricType struct {
  Name        string
  Unit        string
  Aggregation AggregationType
}
```

Modelling

Overview of the constraint types

Database

represents a database which belongs to given cluster in a given instance in a given pipeline, this is captured by the relations field, which stores a set of labels capturing the these relations.


```json
{
 "name": "somestore-us1-cluster5-dbbabe1",
 "creation_time": "2017-01-01T12:00:00Z",
 "relations": {...},
 "resource_requirements": [...],
 "placement_requirements": [...],
 "relation_requirements": [...]
}
```


```go
type Database struct {
    Name                    string
    CreationTime            time.Time
    PlacementRequirements   []*PlacementRequirement
    RelationRequirements    []*RelationRequirement
    ResourceRequirements    []*ResourceRequirement
    Relations               *LabelSet
    Usage                   *MetricSet
}
```

Modelling

Overview of the constraint types

PlacementRequirement

represents a requirement in relation to a specific PlacementGroup having a specific label, i.e. we want to be placed in a given datacenter or we do not want to be placed on a specific rack, etc.

```go
type PlacementRequirement struct {
  AppliesTo     *LabelSet
  ConditionType ConditionType
  Required      *LabelSet
}
```

```json
{
 "applies_to": {
   "set": [
     {
       "name": ["datacenter"]
     }
   ]
 },
 "condition_type": "non_empty_intersection",
 "required": {
   "set": [
     {
       "name": ["dc2"]
     }
   ]
 }
}
```

```json
{
 "applies_to": {
   "set": [
     {
       "name": ["host"]
     }
   ]
 },
 "condition_type": "empty_intersection",
 "required": {
   "set": [
     {
       "name": ["issue"]
     }
   ]
 }
}
```

```json
{
 "applies_to": {
   "set": [
     {
       "name": ["host"]
     }
   ]
 },
 "condition_type": "empty_intersection",
 "required": {
   "set": [
     {
       "name": ["pools", "schemadock-testing"]
     }
   ]
 }
}
```

Modelling

Overview of the constraint types

RelationRequirement

represents a requirement in relation to a specific PlacementGroup having a specific relation, i.e. we want avoid a certain other type of databases.

```json
{
 "applies_to": {
   "set": [
     {
       "name": [
         "host"
       ]
     }
   ]
 },
 "label": {
   "name": [
     "schemaless", "instance", "somestore"
   ]
 },
 "comparison": "less_than_equal",
 "occurrences": 0
}
```

```go
type RelationRequirement struct {
  AppliesTo   *LabelSet
  Label       *Label
  Comparison  ComparisonType
  Occurrences int
}
```

Modelling

Overview of the constraint types

ResourceRequirement

represents a hard requirement for placing a Database in a given PlacementGroup which should have certain requirements for a specific metric.

```go
type ResourceRequirement struct {
  MetricType MetricType
  BoundType  BoundType
  Value      float64
}
```

```json
{
 "metric_type": {
   "aggregation": "sum",
   "unit": "bytes",
   "name": "disk_free"
 },
 "bound_type": "lower",
 "value": 841687164784
}
```

```json
{
 "metric_type": {
   "aggregation": "sum",
   "unit": "bytes",
   "name": "memory_free"
 },
 "bound_type": "lower",
 "value": 68719476736
}
```

Simulations

How to run simulations to answer questions about placements

Simulations

How to run simulations to answer questions about placements

We can run simulations on a snapshot of the hosts and databases in production. This is useful for

- Deciding if we have enough capacity to create new instances
- Deciding if we have enough capacity to split instances

```json
{
 "operations": [
   {
     "type": "placement",
     "placement_request": {
       "databases": [
         {
           "name": "somestore-us1-cluster5-db3",
           "creation_time": "2017-01-01T12:00:00Z",
           "relations": {...},
           "resource_requirements": [...],
           "placement_requirements": [...],
           "relation_requirements": [...]
         }
       ],
       "skip_reservations": false,
       "skip_transcripts": true
     },
     "time": "2017-03-30T10:46:49.336639221-07:00"
   }
 ]
}
```

Simulations

How to run simulations to answer questions about placements

We can run simulations on a snapshot of the hosts and databases in production. This is useful for

- Deciding if we have enough capacity to create new instances
- Deciding if we have enough capacity to split instances
- Experimenting with new placement orderings to see how it affects host utilization


```json
{
 "operations": [
   {
     "type": "relocation",
     "relocation_request": {
       "group_name": "dc1",
       "minimum_rank": 1,
       "minimum_age": "0m"
     },
     "time": "2017-03-30T10:46:49.336639221-07:00",
     "relocations": 1000
   },
   {
     "type": "relocation",
     "relocation_request": {
       "group_name": "dc2",
       "minimum_rank": 1,
       "minimum_age": "0m"
     },
     "time": "2017-03-30T10:46:49.336639221-07:00",
     "relocations": 1000
   }
 ]
}
```

Thank you

Casper Kejlberg-Rasmussen
Software Development Lead at Uber
kejlberg@uber.com

- Project Mezzanine: The Great Migration at Uber Engineering
  - http://eng.uber.com/mezzanine-migration/
- Designing Schemaless, Uber Engineering's Scalable Datastore Using MySQL
  - https://eng.uber.com/schemaless-part-one/
- The Architecture of Schemaless, Uber Engineering's Trip Datastore Using MySQL
  - https://eng.uber.com/schemaless-part-two/
- Using Triggers On Schemaless, Uber Engineering's Datastore Using MySQL
  - https://eng.uber.com/schemaless-part-three/
- Dockerizing MySQL at Uber Engineering
  - https://eng.uber.com/dockerizing-mysql/

UBER