



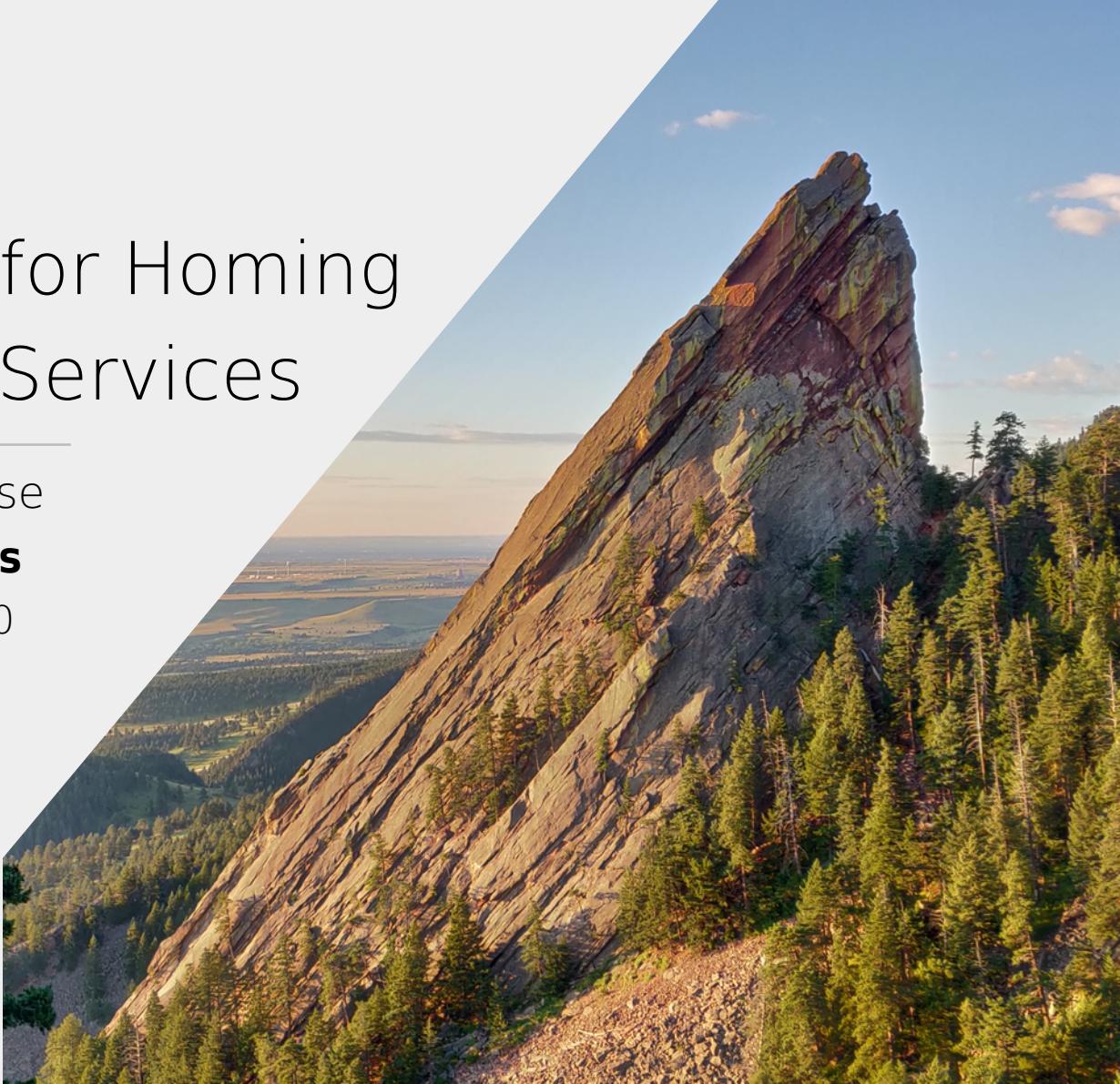
Efficient Approaches for Homing Complex Network Services

Dissertation Defense

by **Azzam Alsudais**

November 11th 2020

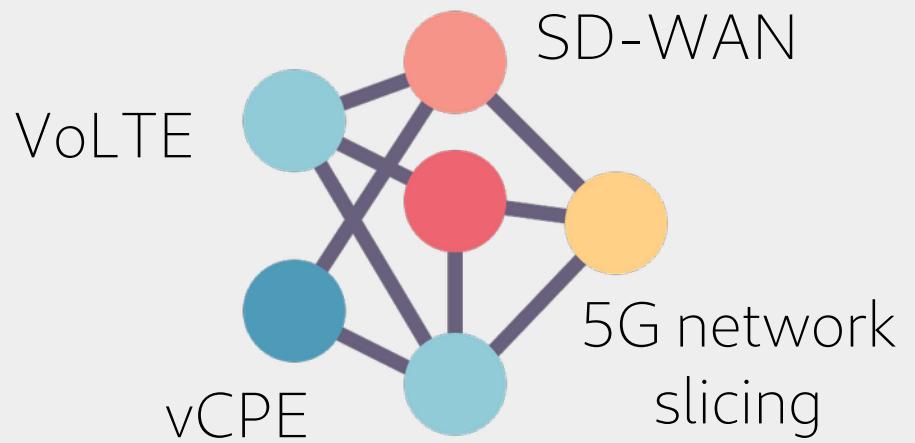
Committee	Eric Keller Shivakant Mishra Sangtae Ha Eric Rozner Shankar P. Narayanan
-----------	--



NSP Services



verizon[✓]



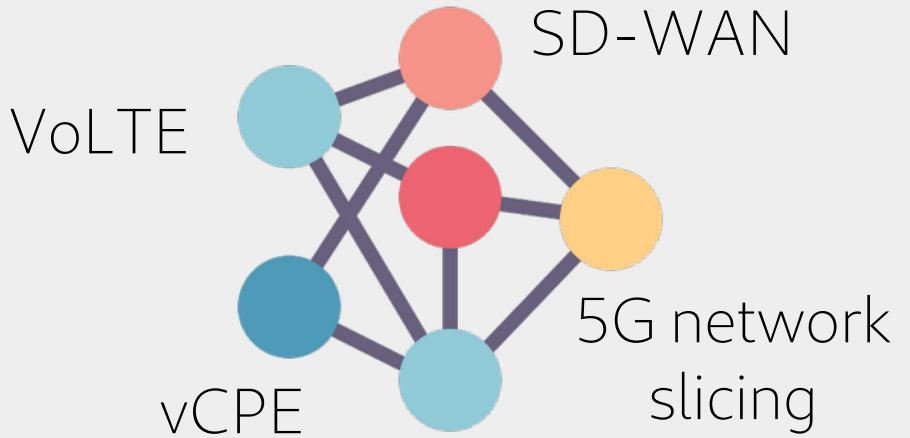
NSP Services



verizon[✓]



NSPs provide tens of network services to their corporate and individual customers



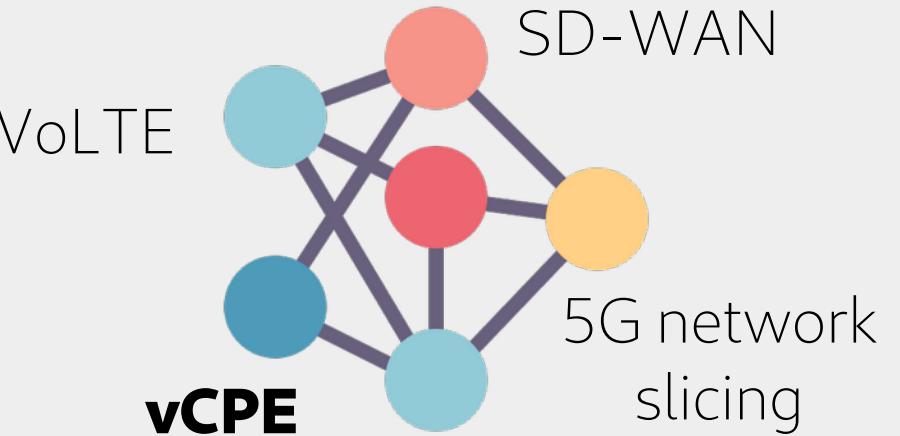
NSP Services



verizon[✓]



NSPs provide tens of network services to their corporate and individual customers



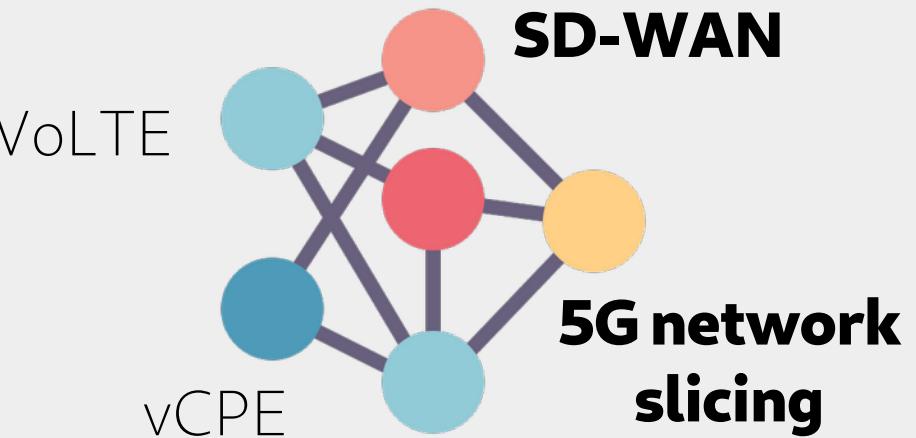
NSP Services



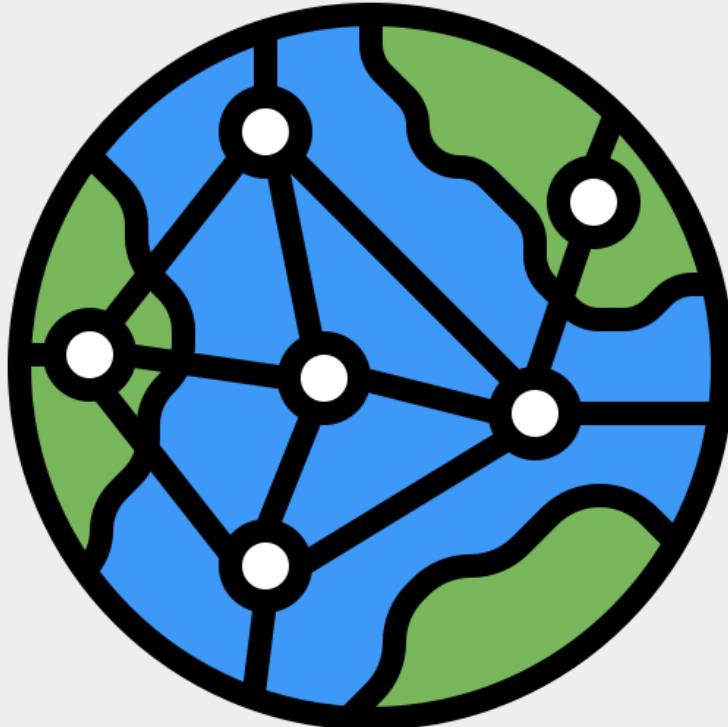
verizon[✓]



NSPs provide tens of network services to their corporate and individual customers

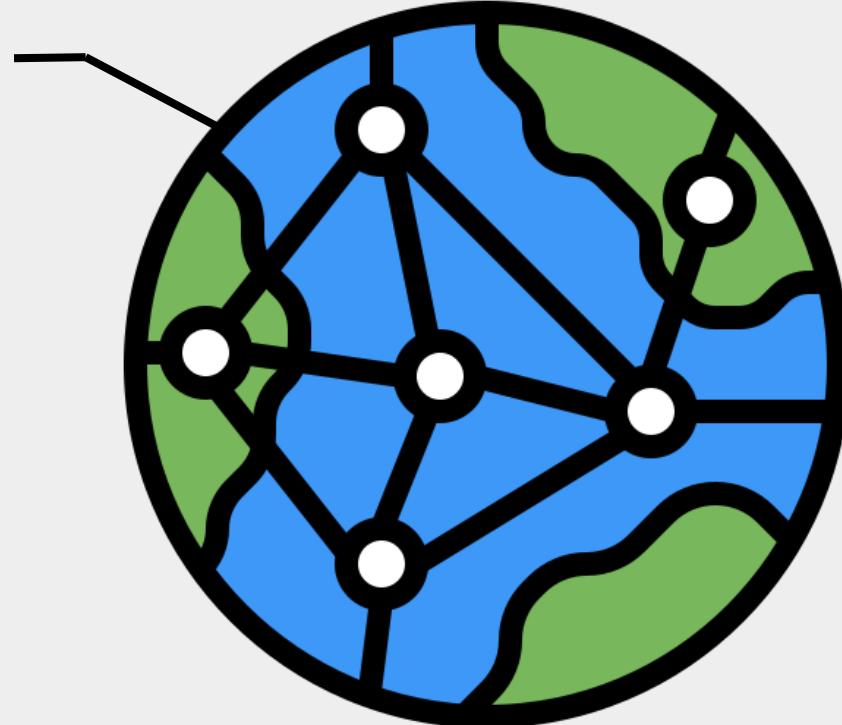


Global-scale Infrastructure

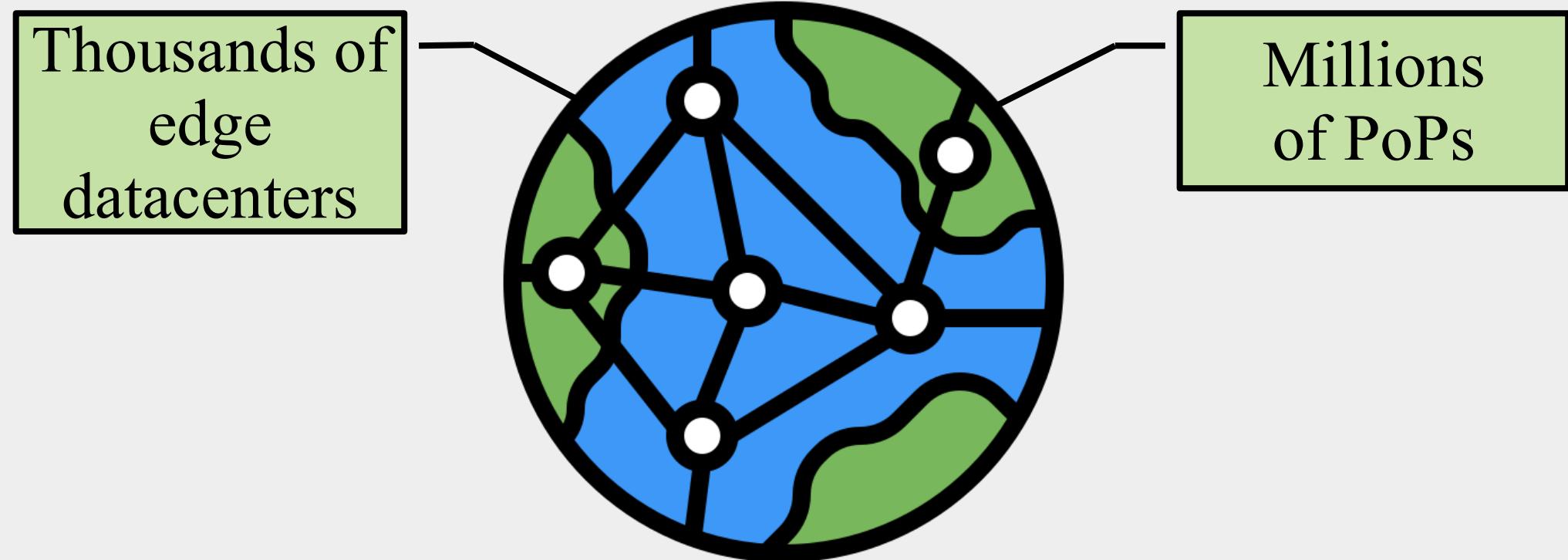


Global-scale Infrastructure

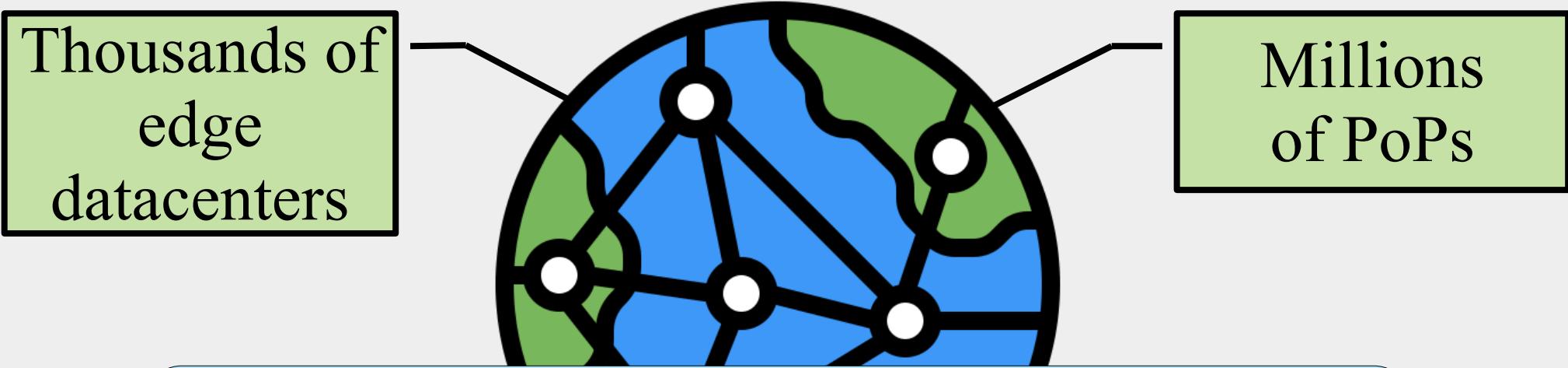
Thousands of
edge
datacenters



Global-scale Infrastructure



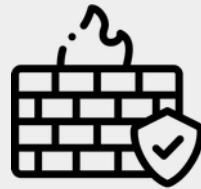
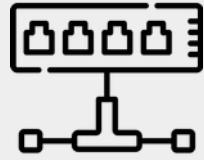
Global-scale Infrastructure



Servicing **1000s** of enterprise customers
and **hundreds of millions** of individual customers

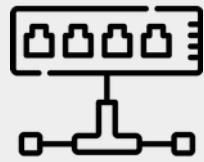
Legacy Network Functions

Legacy HW-centric



Legacy Network Functions

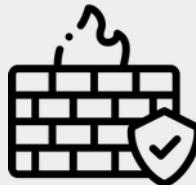
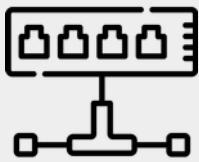
Legacy HW-centric



Hardware-centric can no longer cope
with the ever-evolving and dynamic
nature of today's demands

The move to software

Legacy HW-centric



virtualization

Virtualized Networking

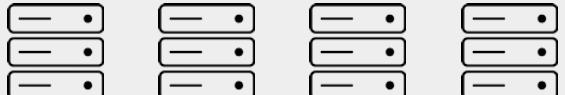
Orchestration & Control



VNFs



Commodity Servers



The move to software

Software-centric networking (or NFV) enabled better ways to **manage, configure, and orchestrate** network services



VIRTUALIZATION

Virtualized Networking

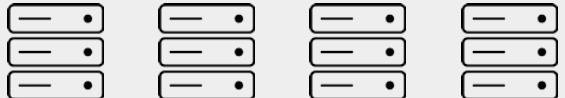
Orchestration & Control



VNFs



Commodity Servers



The move to software

Software-centric networking (or NFV) enabled better ways to **manage, configure, and orchestrate** network services

Homing of VNFs is an essential part in orchestrating network services for NSPs

Virtualized Networking

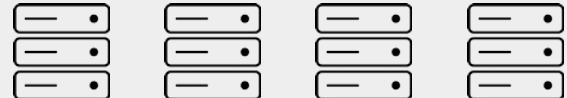
Orchestration & Control



VNFs



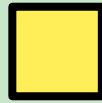
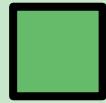
Commodity Servers



Homing as a service

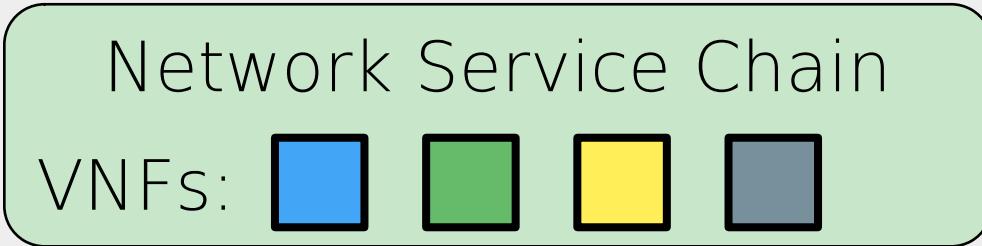
Network Service Chain

VNFs:



Homing as a service

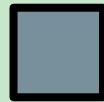
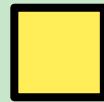
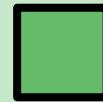
**Network
Demands**



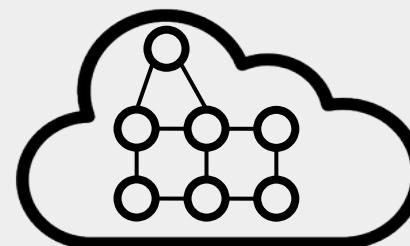
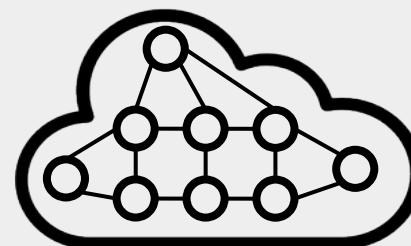
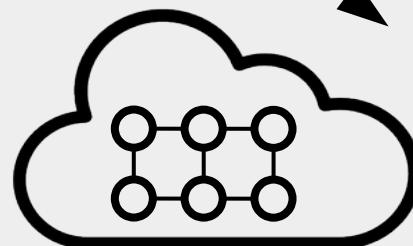
Homing as a service

Network Service Chain

VNFs:



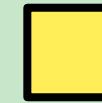
Geo-distributed
Compute/network
Resources



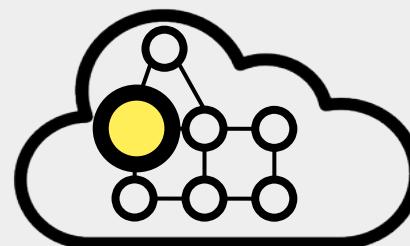
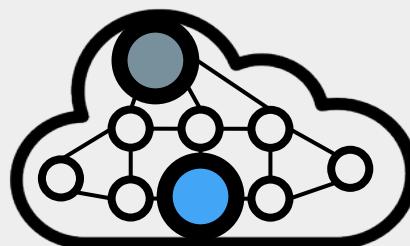
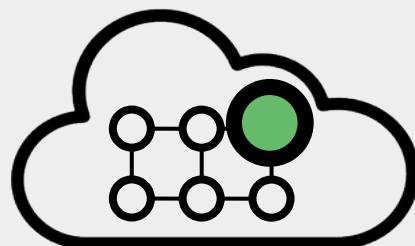
Homing as a service

Network Service Chain

VNFs:



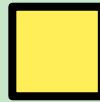
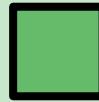
Find best feasible
resources to *home* VNFs



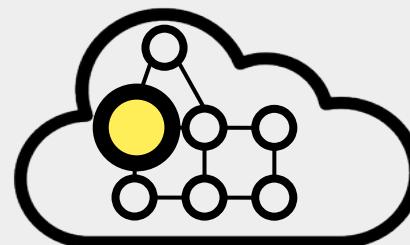
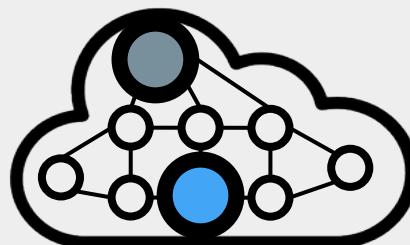
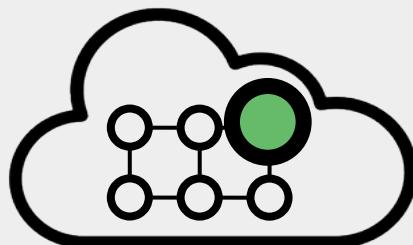
Homing as a service

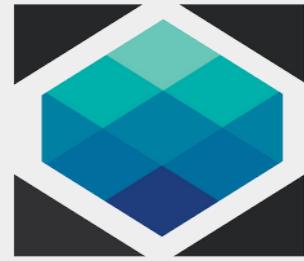
Network Service Chain

VNFs:



Homing is an essential service
in many network management
and automation platforms





ONAP

OPEN NETWORK AUTOMATION PLATFORM

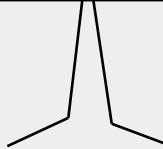
Orchestration, lifecycle management, and
automation of network services



ONAP
OPEN NETWORK AUTOMATION PLATFORM

Orchestration, lifecycle management, and automation of network services

Organizations: AT&T, IBM
Intel, Orange
400+ developers



Over 5M LoC
6 releases over 3 years

Bird's Eye View of How Homing works

Homing Service

Homing Service



Homing Service



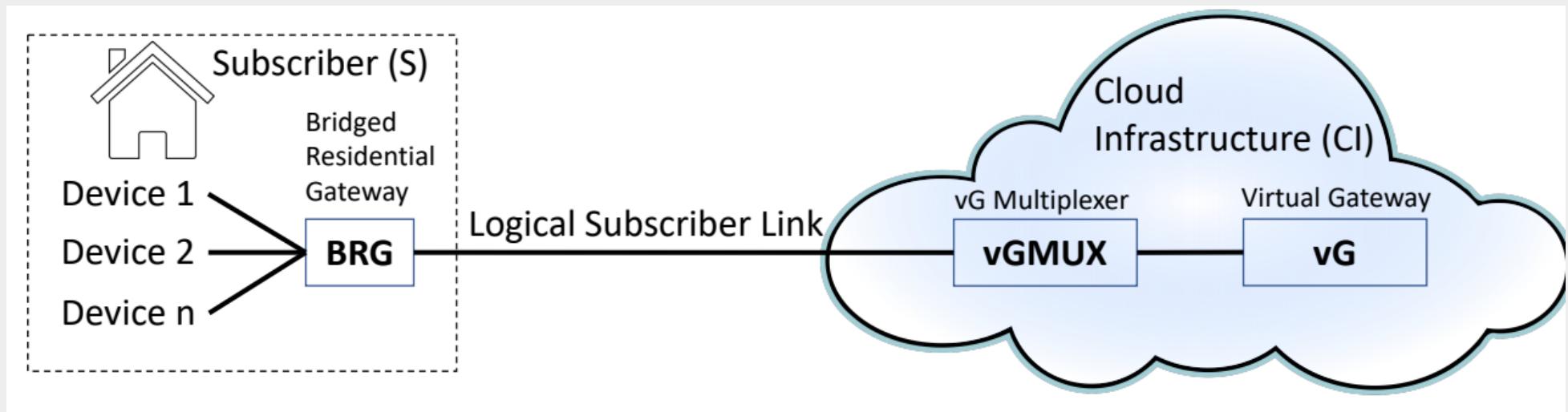
Submitted by NSP customers describing required service
such as: **SD-WAN, vCPE, 5G** network slicing, etc.

Homing Service

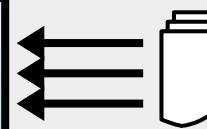


Example Service

Virtual Customer Premises Equipment (vCPE)
to provide dedicated broadband service



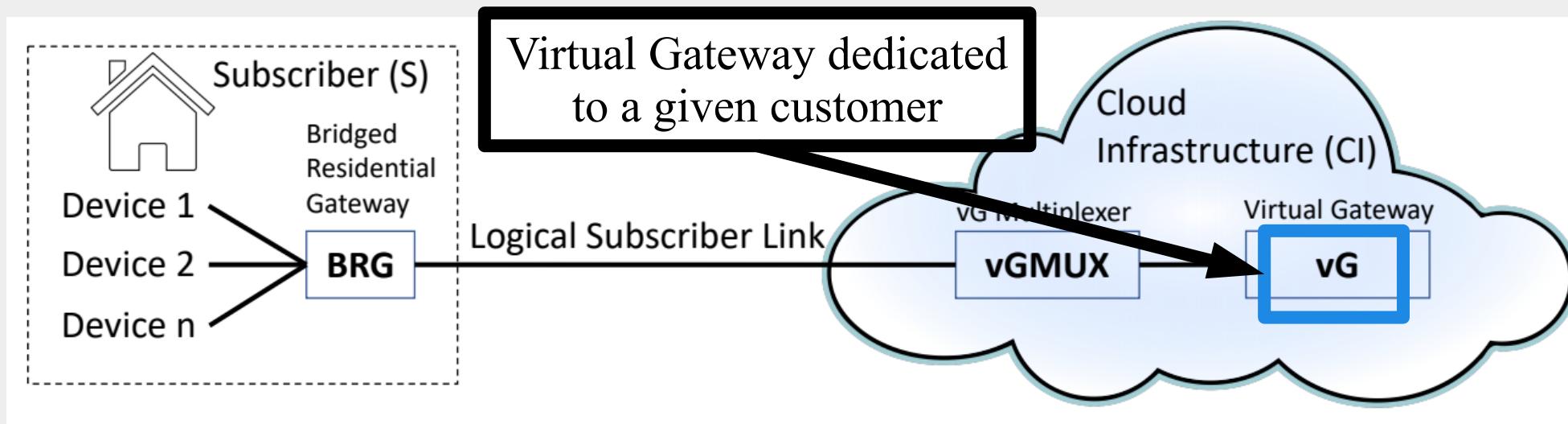
Homing Service



Homing
Requests

Example Service

Virtual Customer Premises Equipment (vCPE)
to provide dedicated broadband service

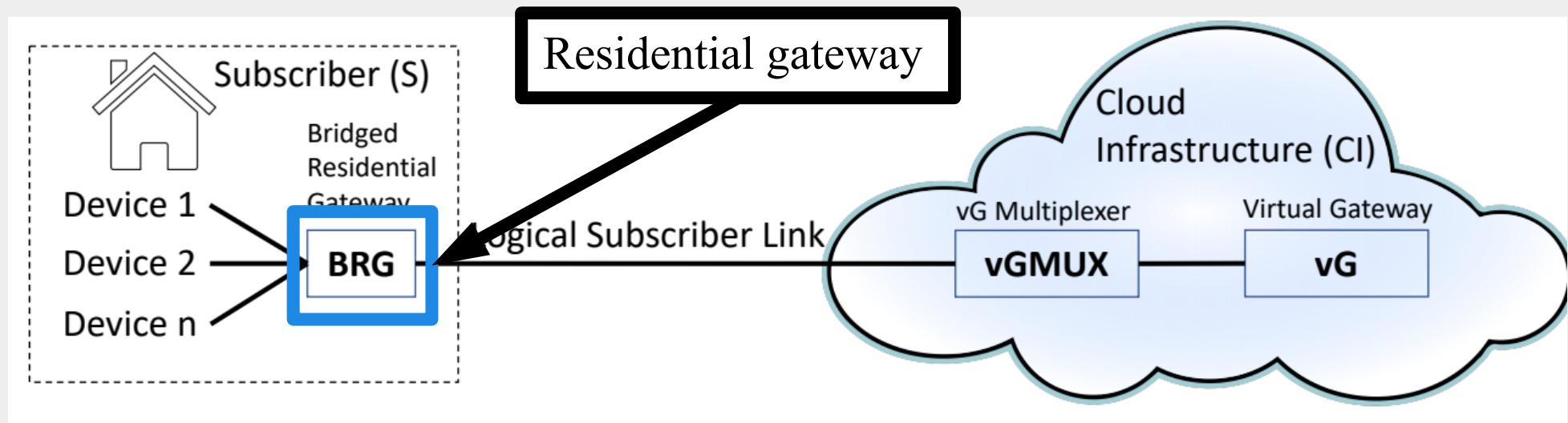


Homing Service

Homing Requests

Example Service

Virtual Customer Premises Equipment (vCPE)
to provide dedicated broadband service

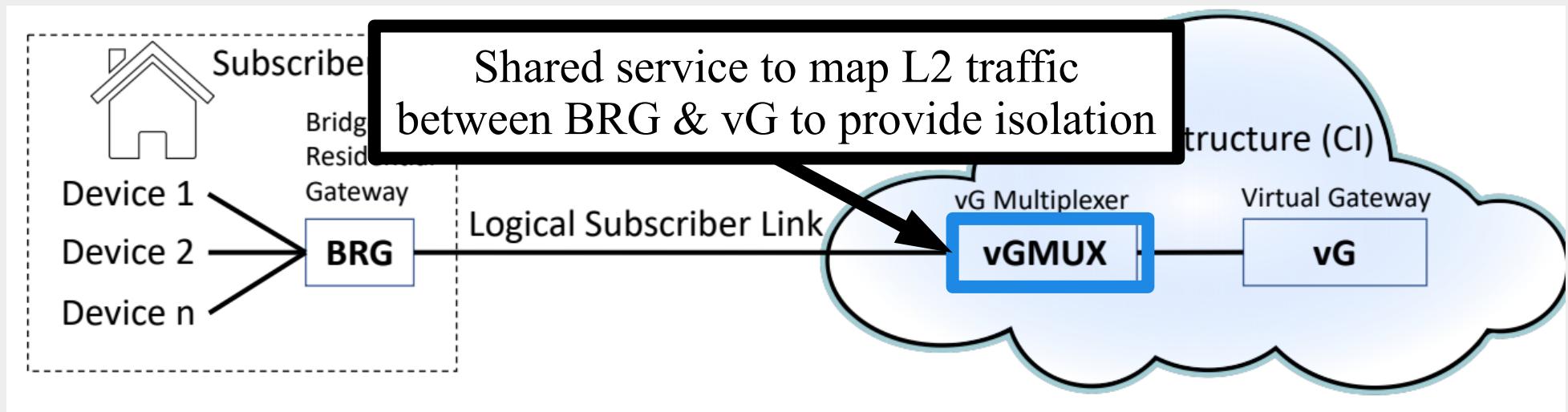


Homing Service



Example Service

Virtual Customer Premises Equipment (vCPE)
to provide dedicated broadband service

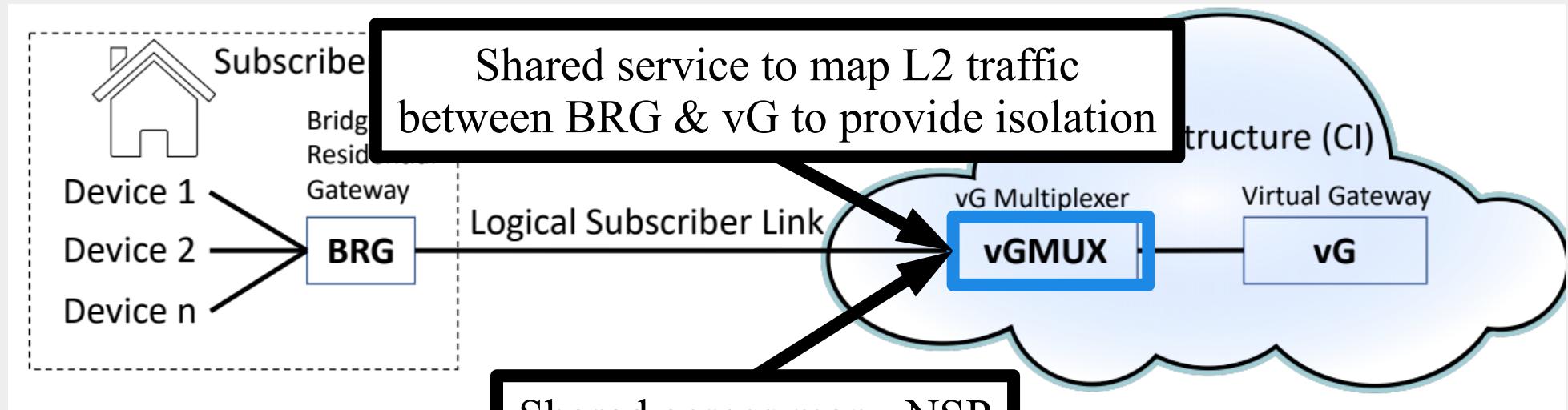


Homing Service

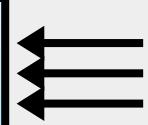


Example Service

Virtual Customer Premises Equipment (vCPE)
to provide dedicated broadband service



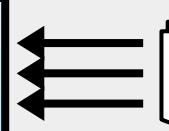
Homing Service



Homing
Requests

Homing Request Blocks

Homing Service



Homing Requests

Homing Request Blocks

Demands (VNFS)

- List of VNFS (FW, loadbalancer, gateway, etc)
- Type of requested resources (cloud or service)



Homing Request Blocks

Demands (VNFS)

- List of VNFS (FW, loadbalancer, gateway, etc)
- Type of requested resources (cloud or service)

Constraints

Define what feasible candidates are (enough Capacity, distance to customer, etc)



Homing Request Blocks

Demands (VNFS)

- List of VNFS (FW, loadbalancer, gateway, etc)
- Type of requested resources (cloud or service)

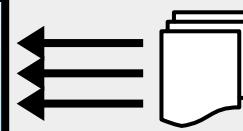
Constraints

Define what feasible candidates are (enough Capacity, distance to customer, etc)

Objective Function(s)

Optimize for certain objectives (distance, latency, provisioning cost, resource utilization, etc)

Homing Service



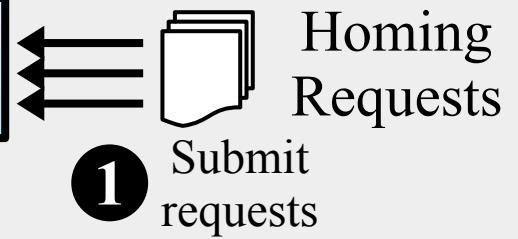
Homing Requests

Homing Request Example (vCPE)

```
Demands: [ vGMux, vG ],  
Customer-id: "c123",  
vpn-key: "c123.vpn1",  
customer-loc: "Boulder",  
Constraints:{  
    "C1": {"distance-to-loc": {  
        "demands": [ vGMux ],  
        "loc": customer-loc,  
        "dist": "< 1500mi" }},  
    "C2": {"zone": {  
        "demands": [ vGMux, vG ],  
        "qualifier": "same",  
        "category": "region" }}},  
.....
```

```
..... "C3": {"cloud-capacity": {  
    "demands": [ vG ],  
    "resources": {"vCPUs": "10",  
        "RAM": "4GB",  
        "DISK": "100GB", }}}},  
Objective-functions:{  
    "minimize": { "sum": {  
        "distance-vGMux": {  
            "demands": [ vGMux ],  
            "location": customer-loc },  
        "distance-vG": {  
            "demands": [ vG ],  
            "location": customer-loc } } } }  
}}
```

Homing Service



Homing Service



Solve each homing request

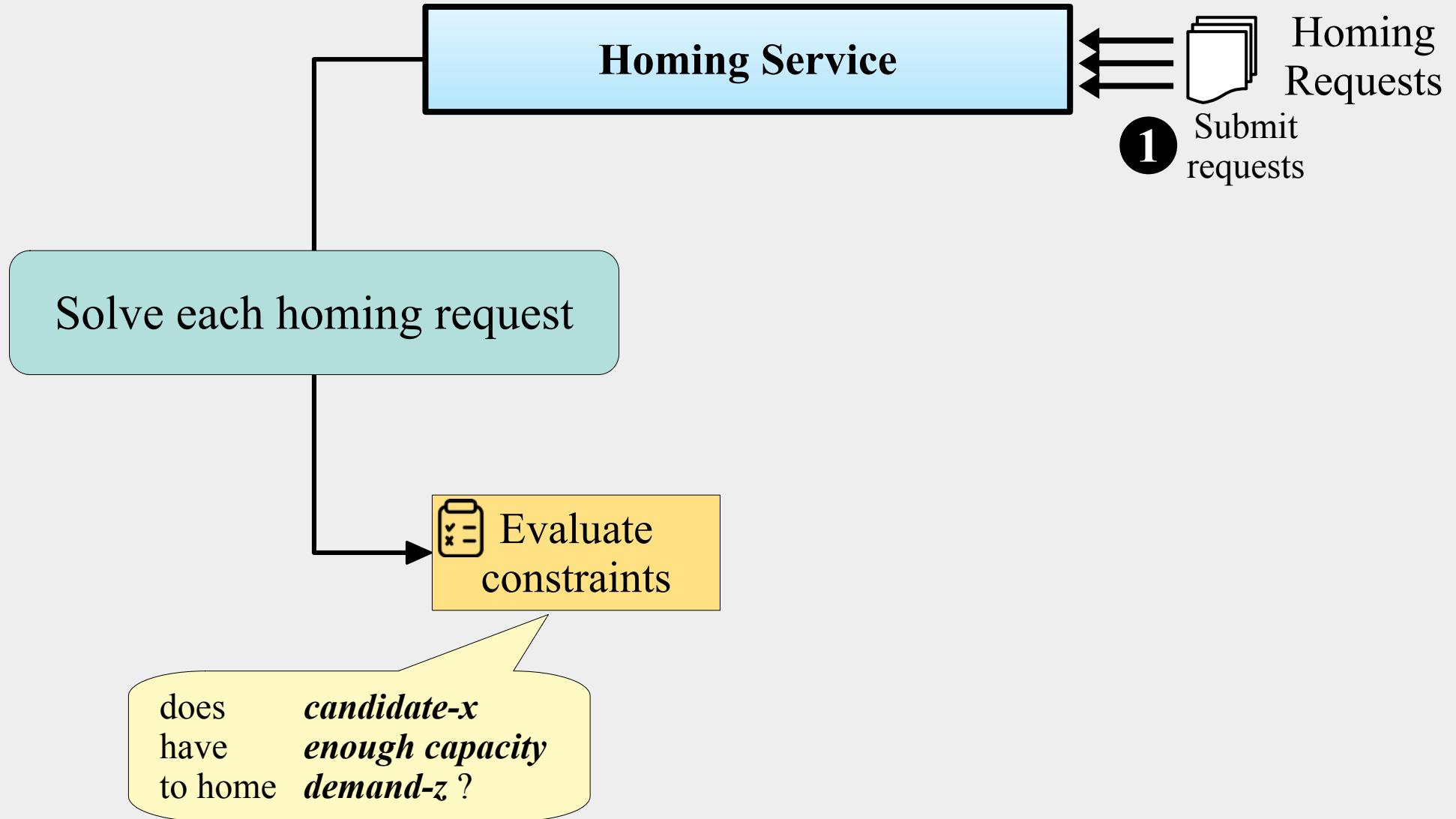
Homing Service

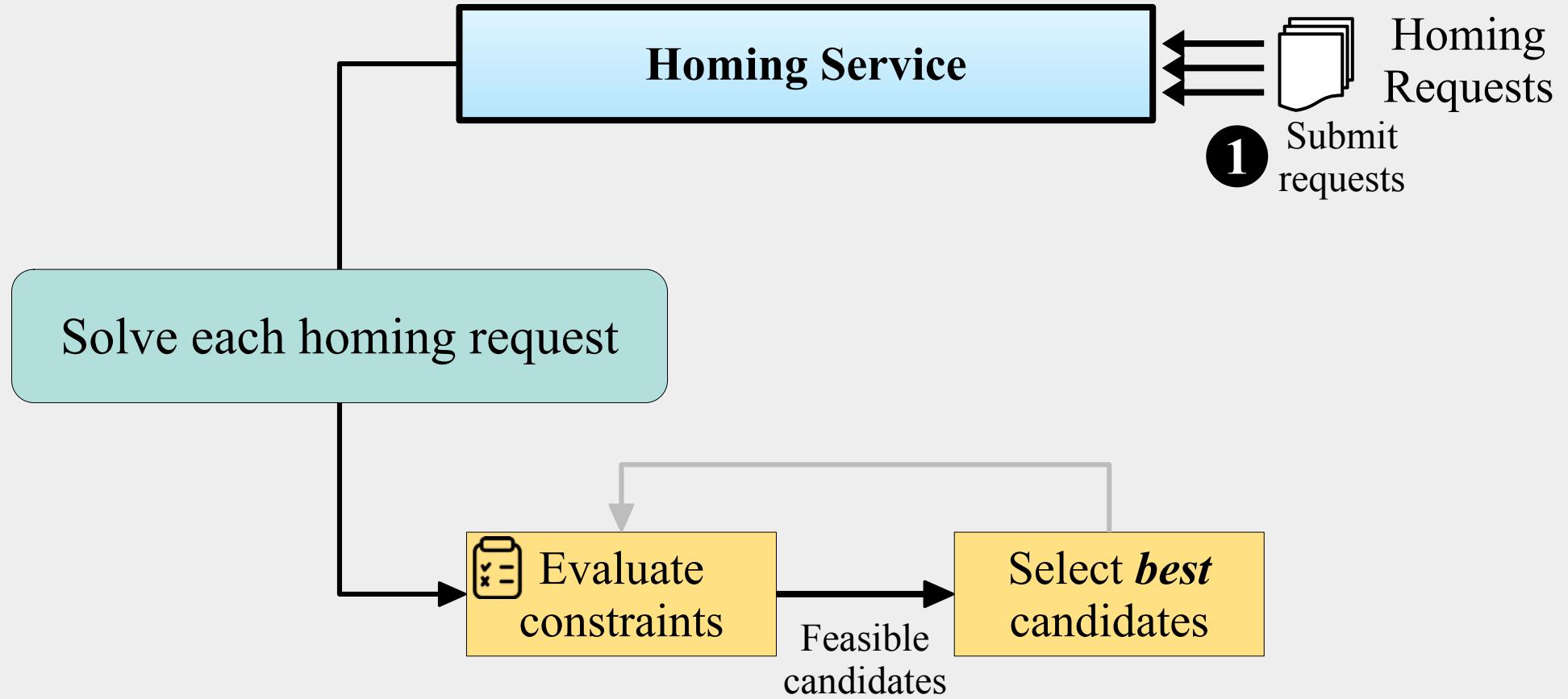


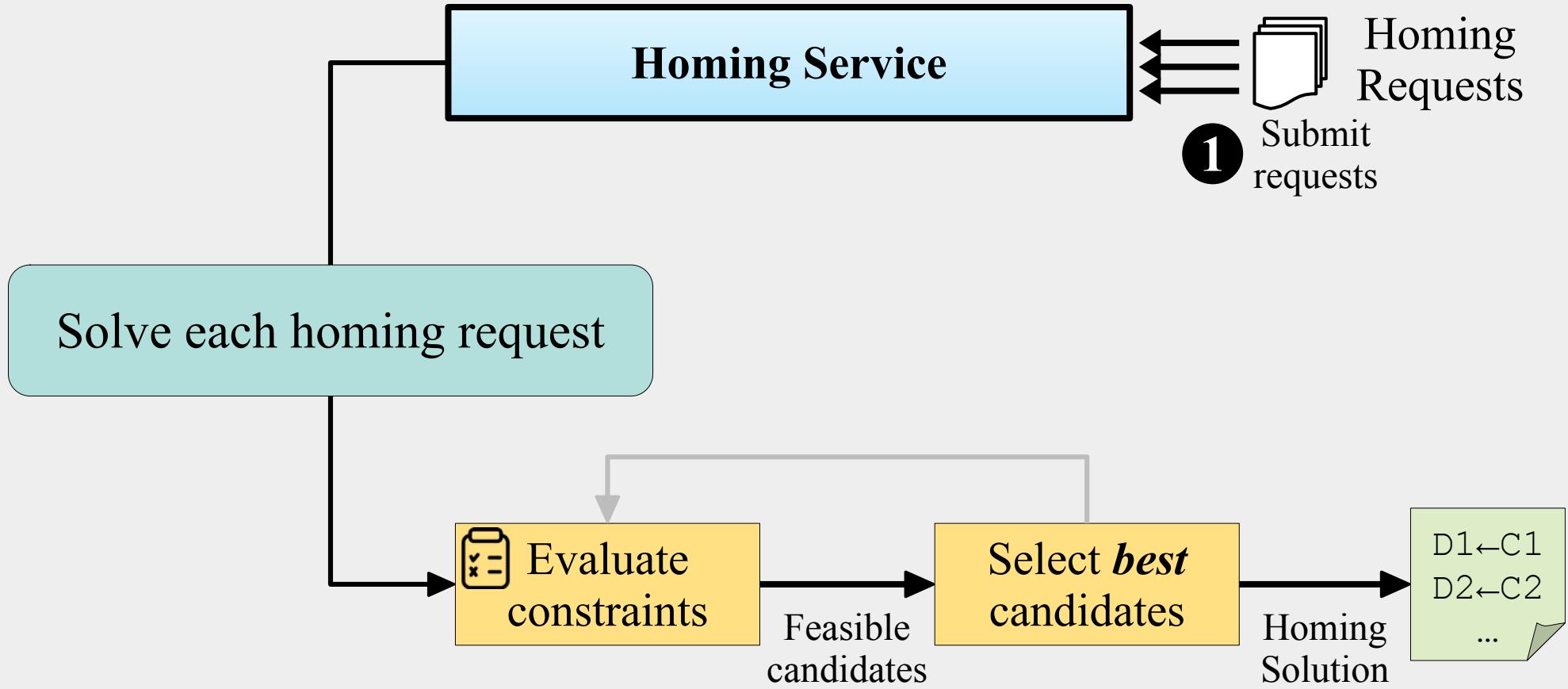
Solve each homing request

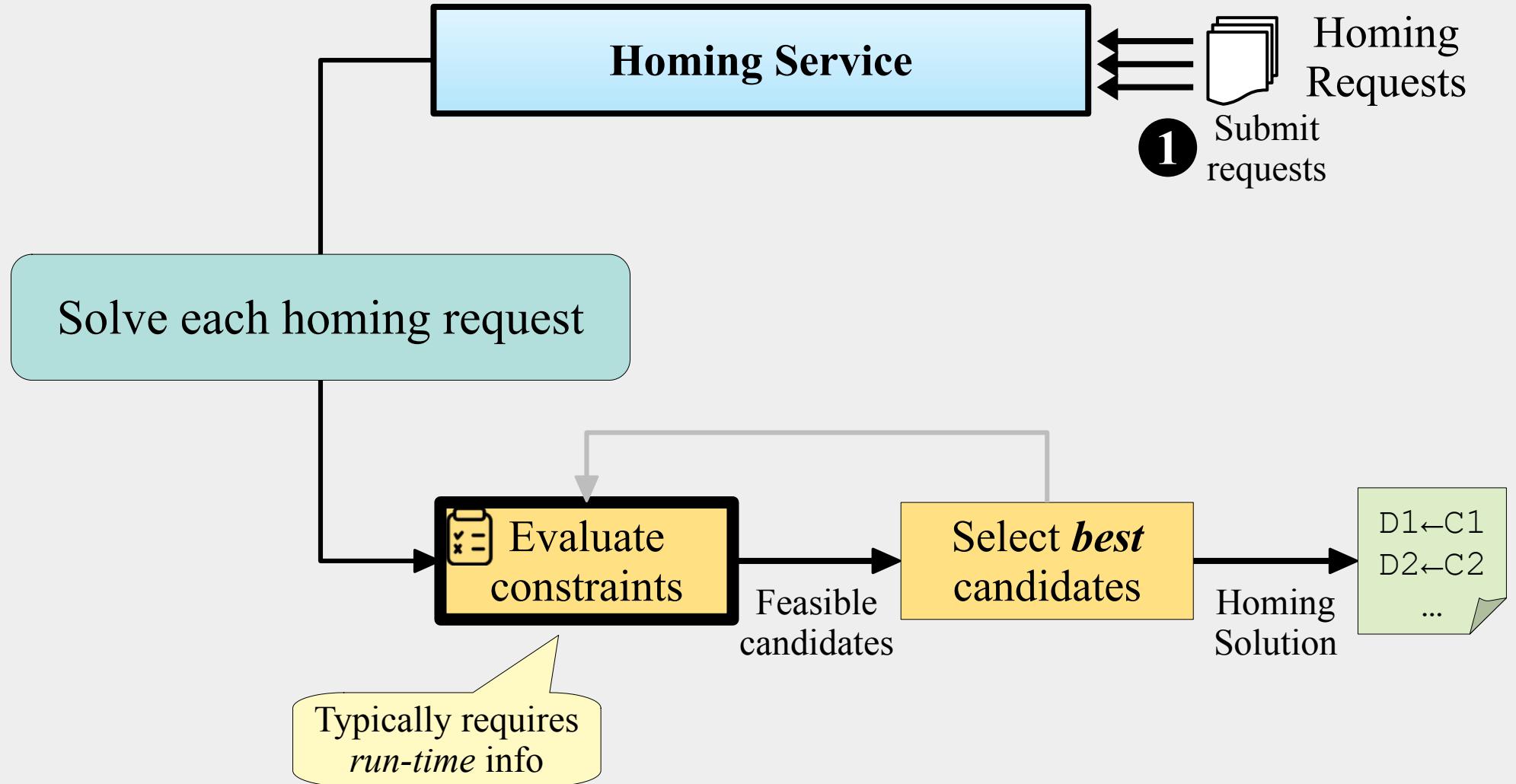


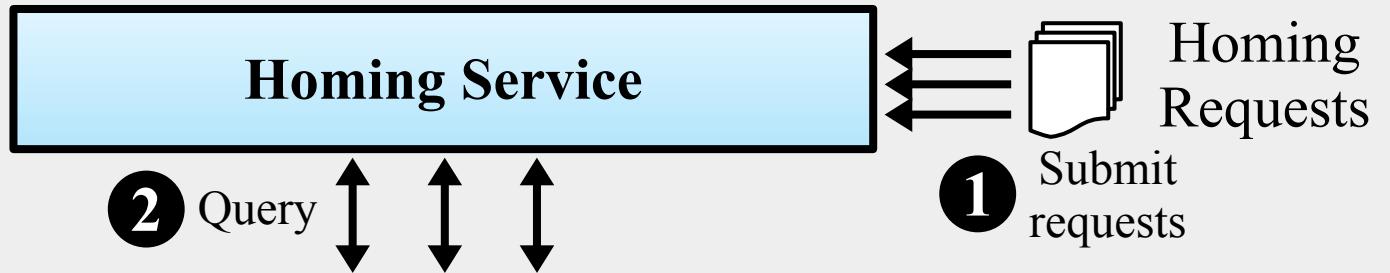
Evaluate
constraints

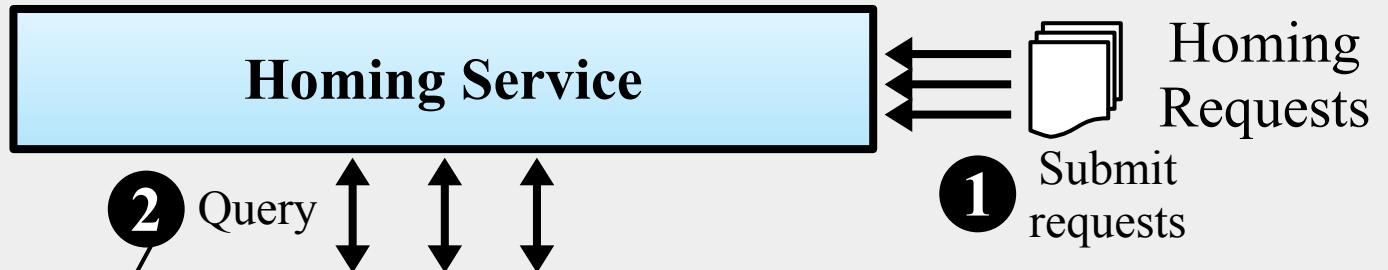




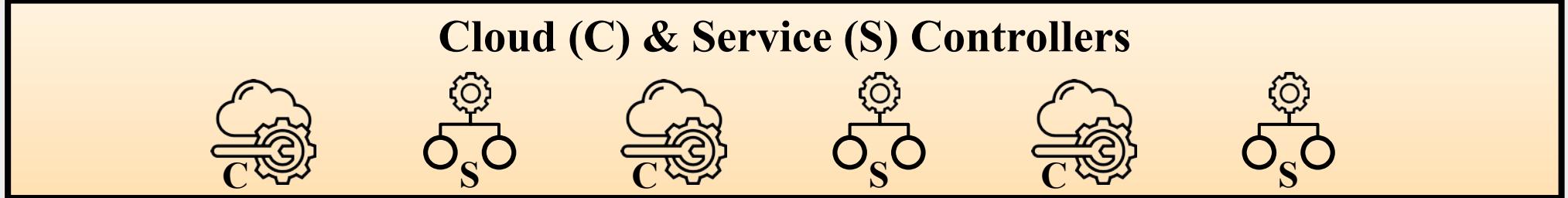
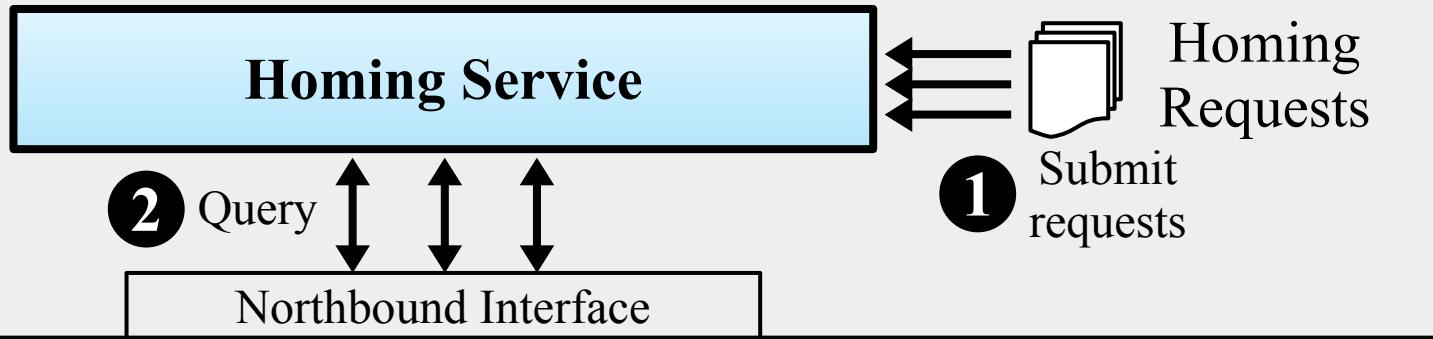


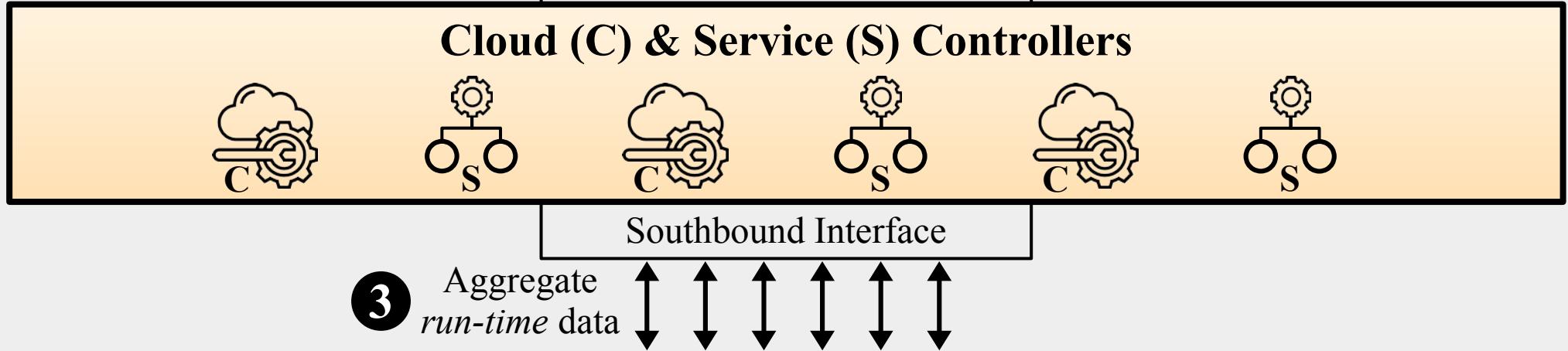
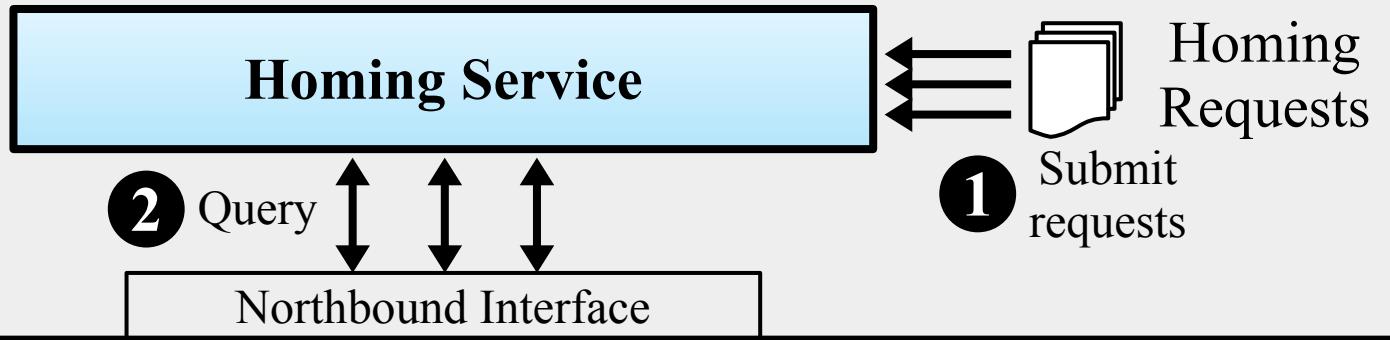


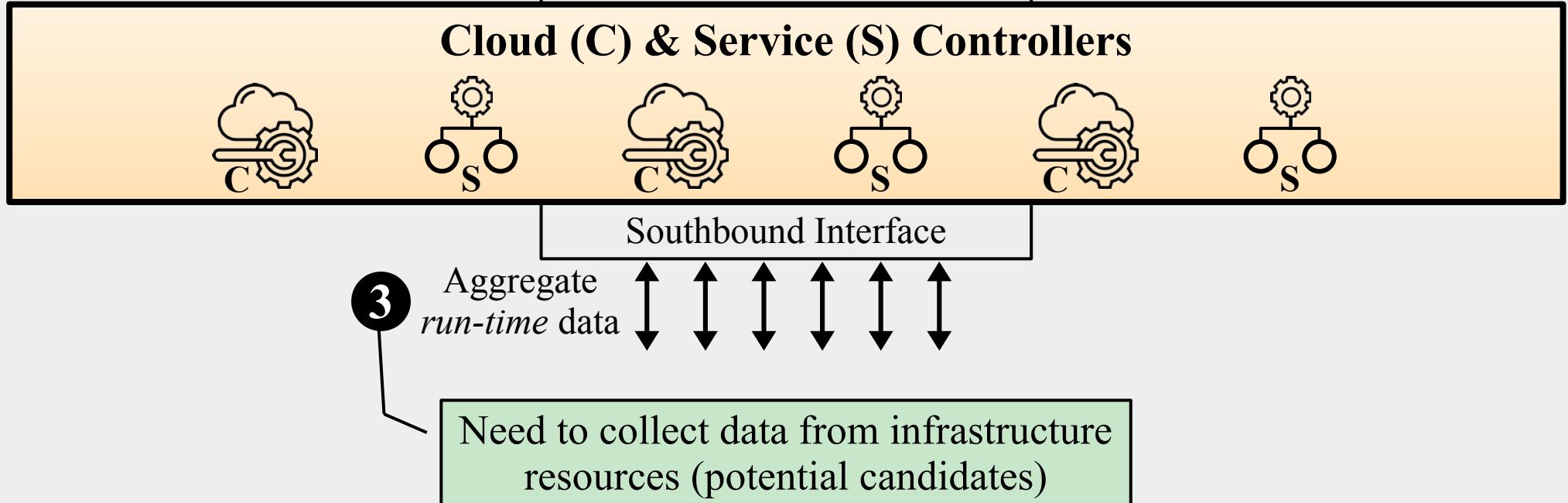
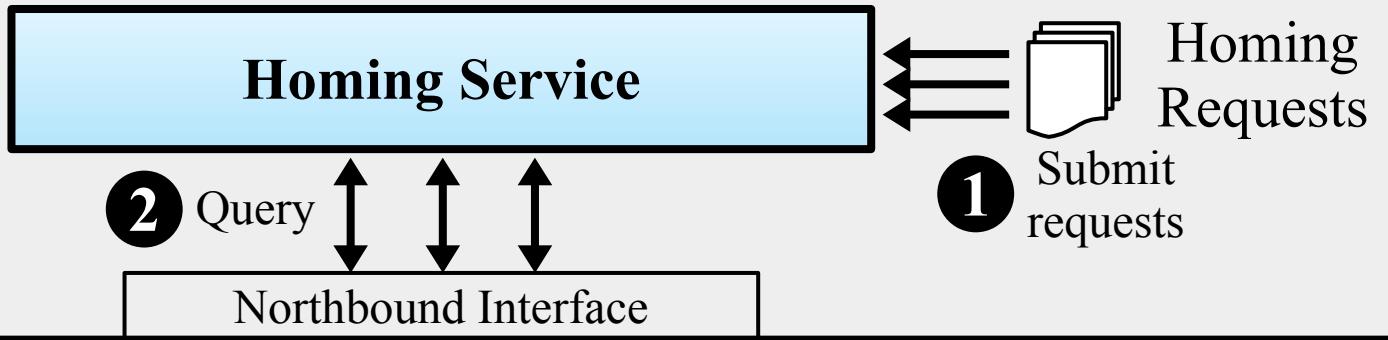


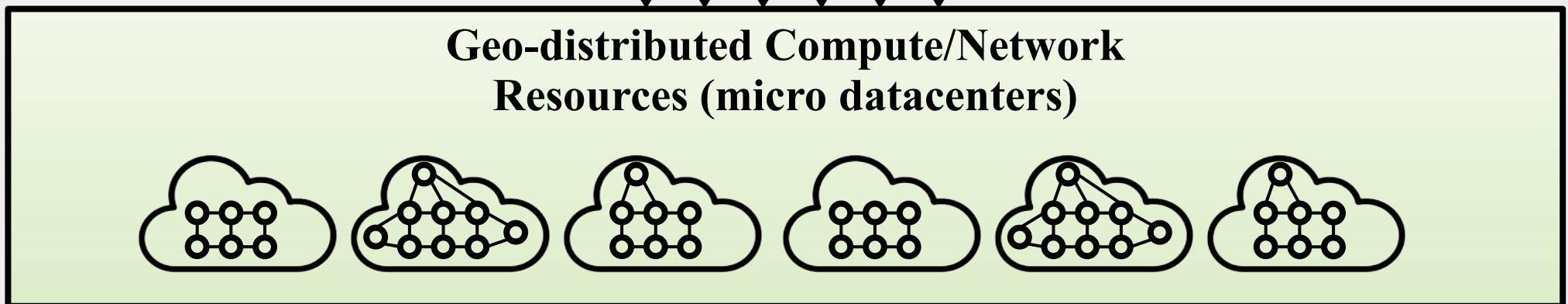
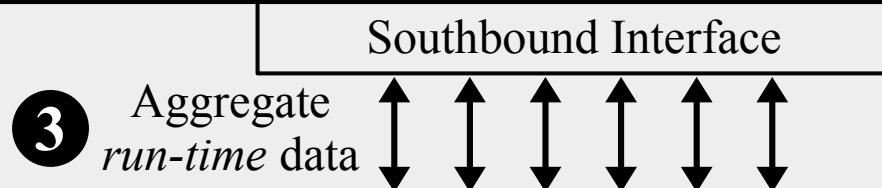
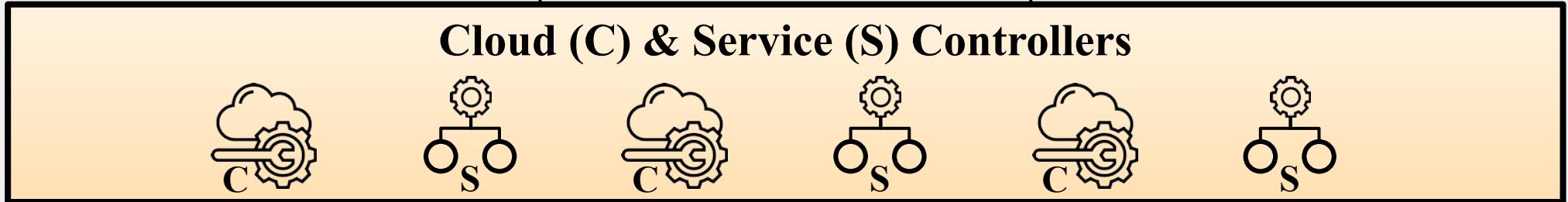
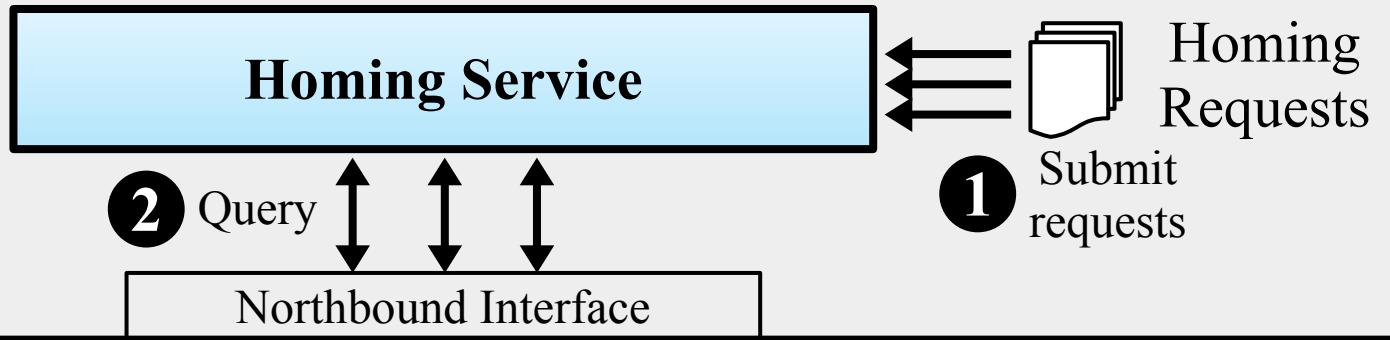


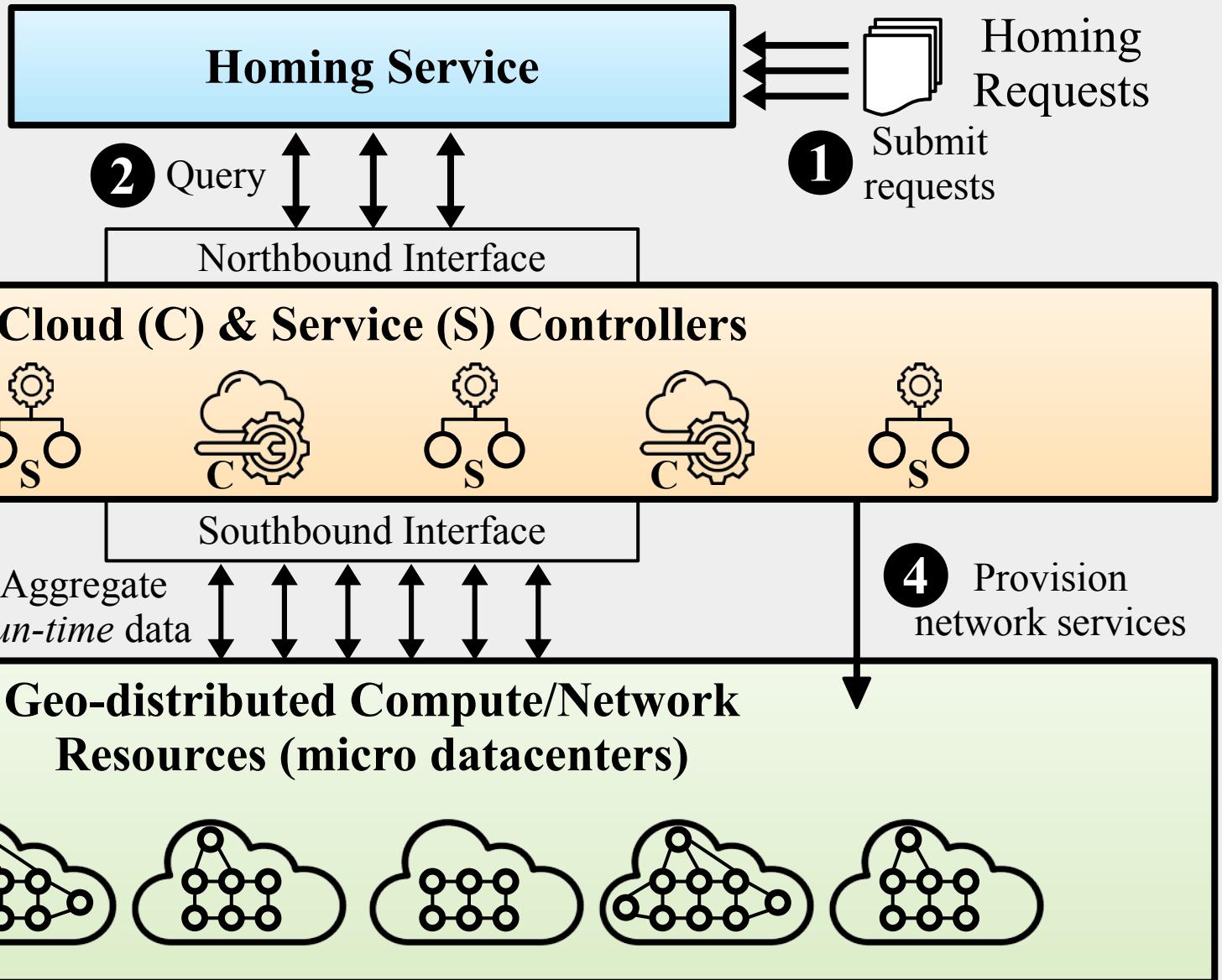
Need to query underlying infrastructure to make informed decisions

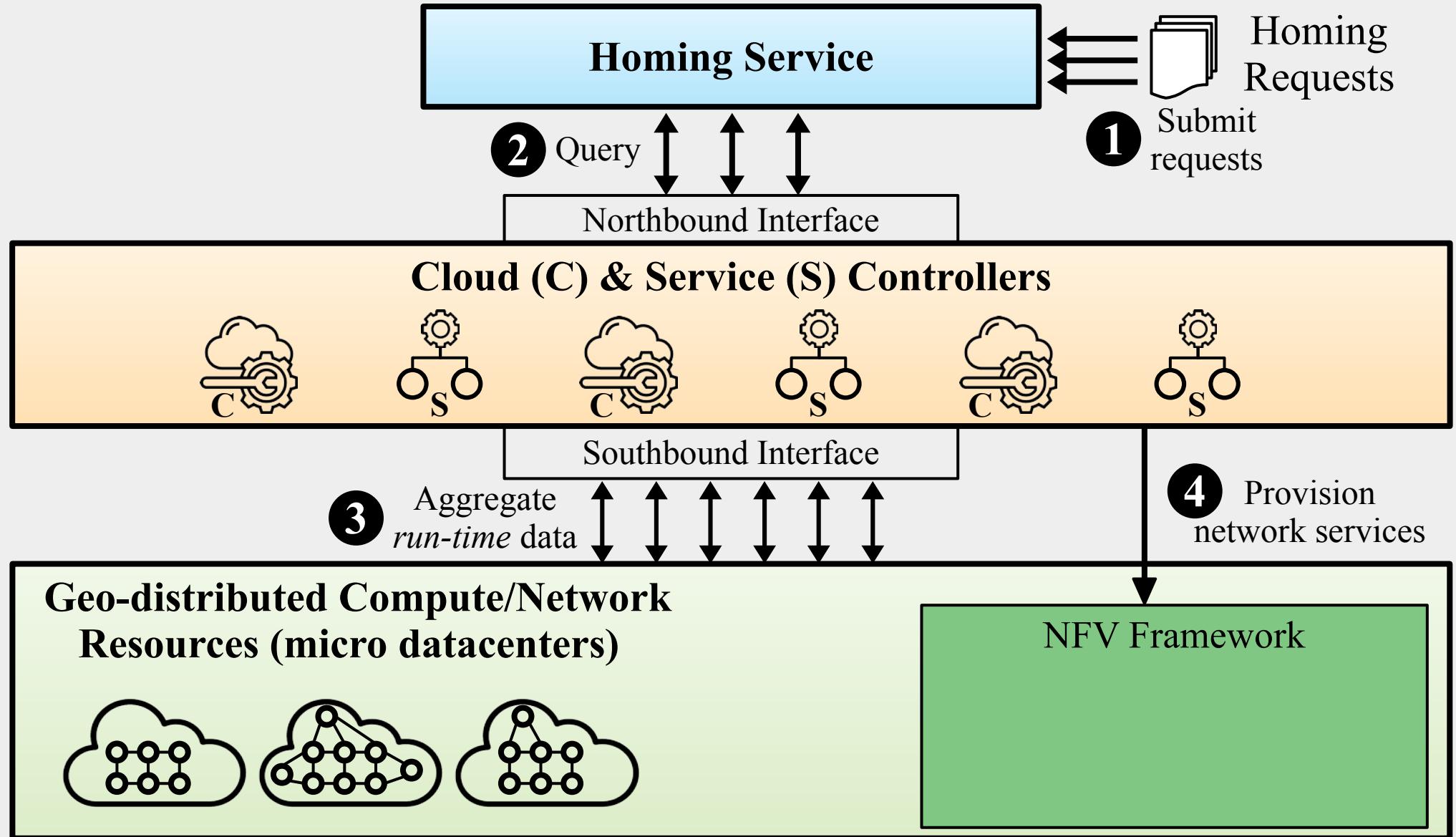


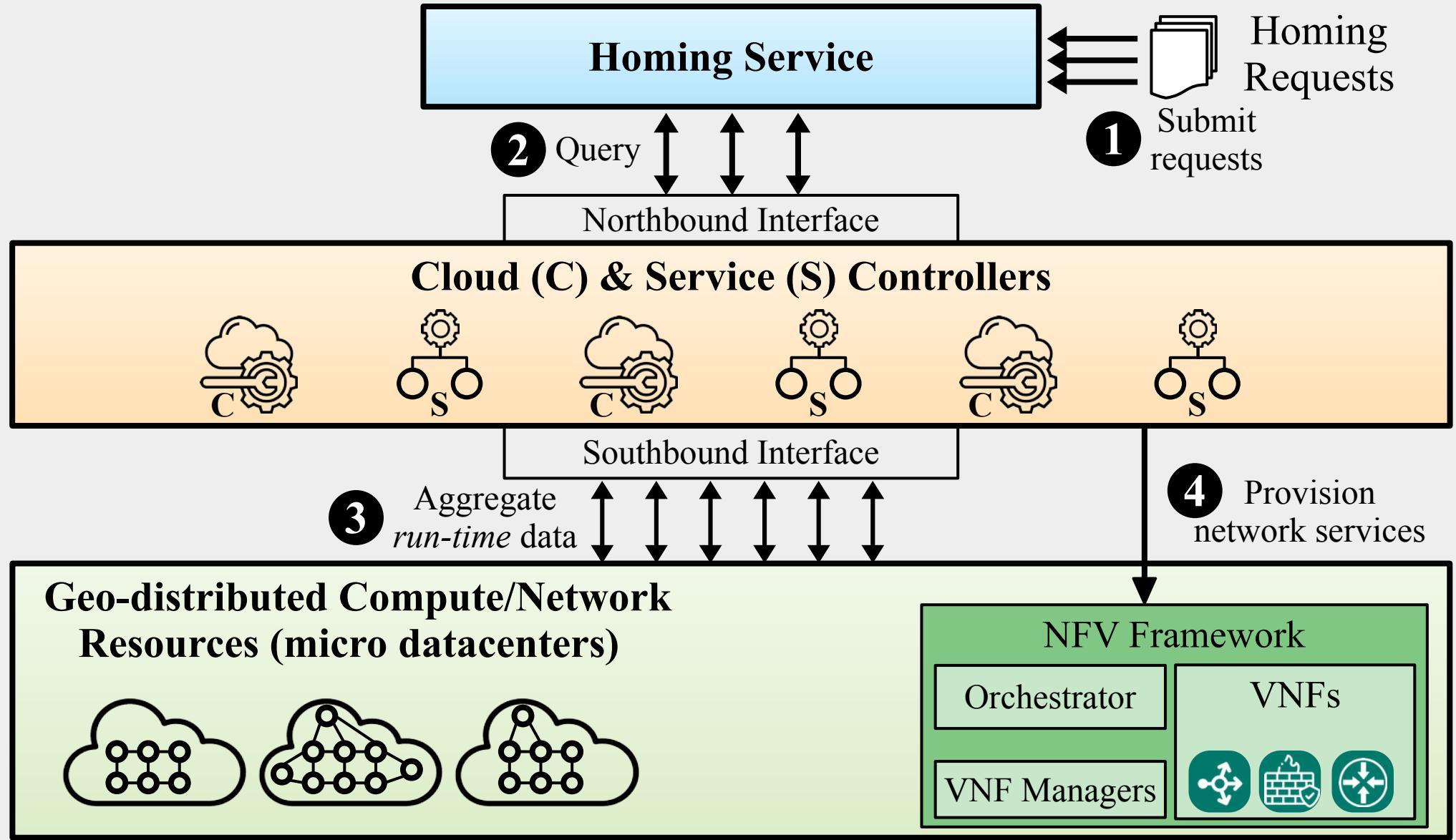


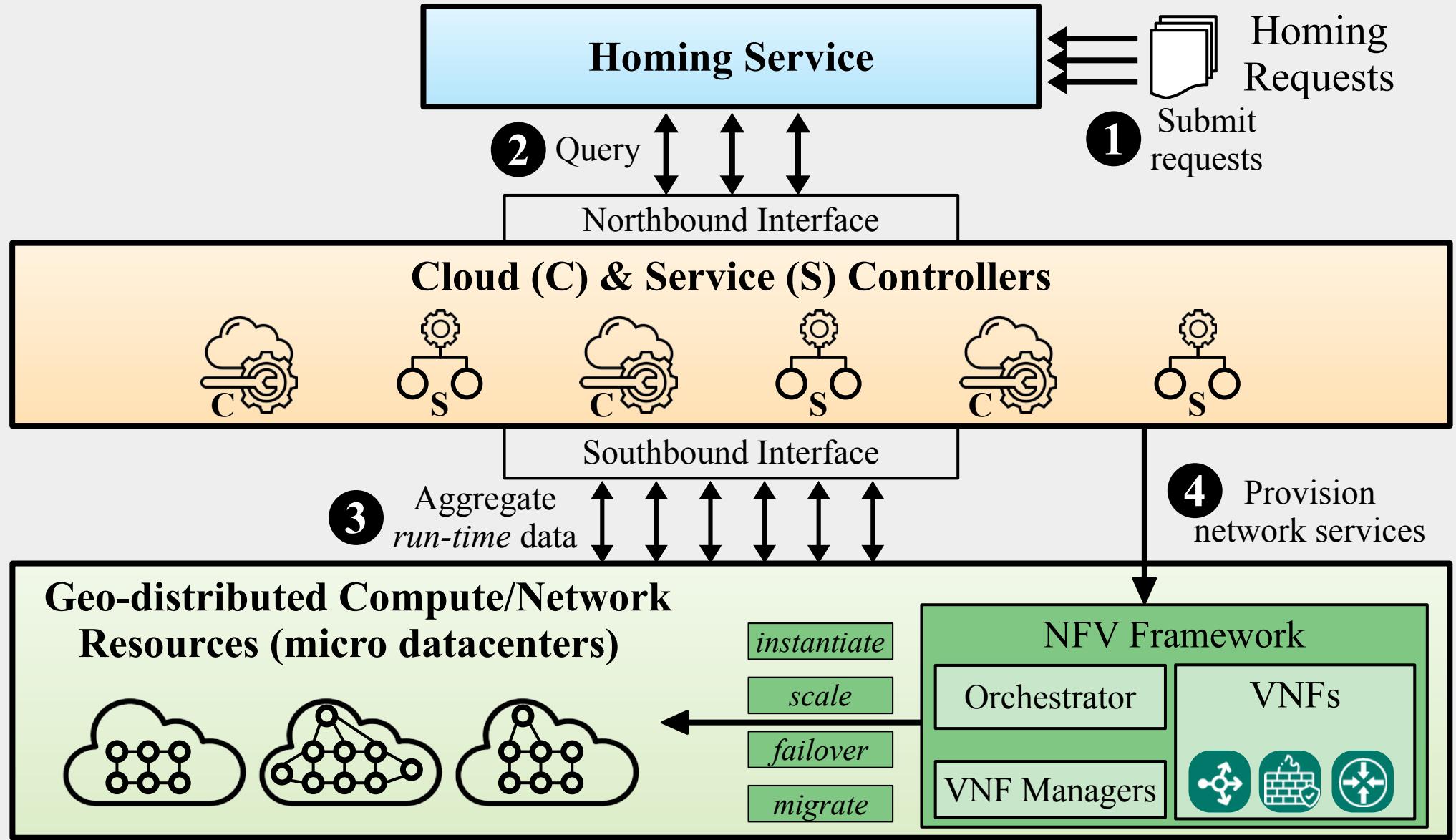












Challenges & Problems



Cloud (C) & Service (S) Controllers

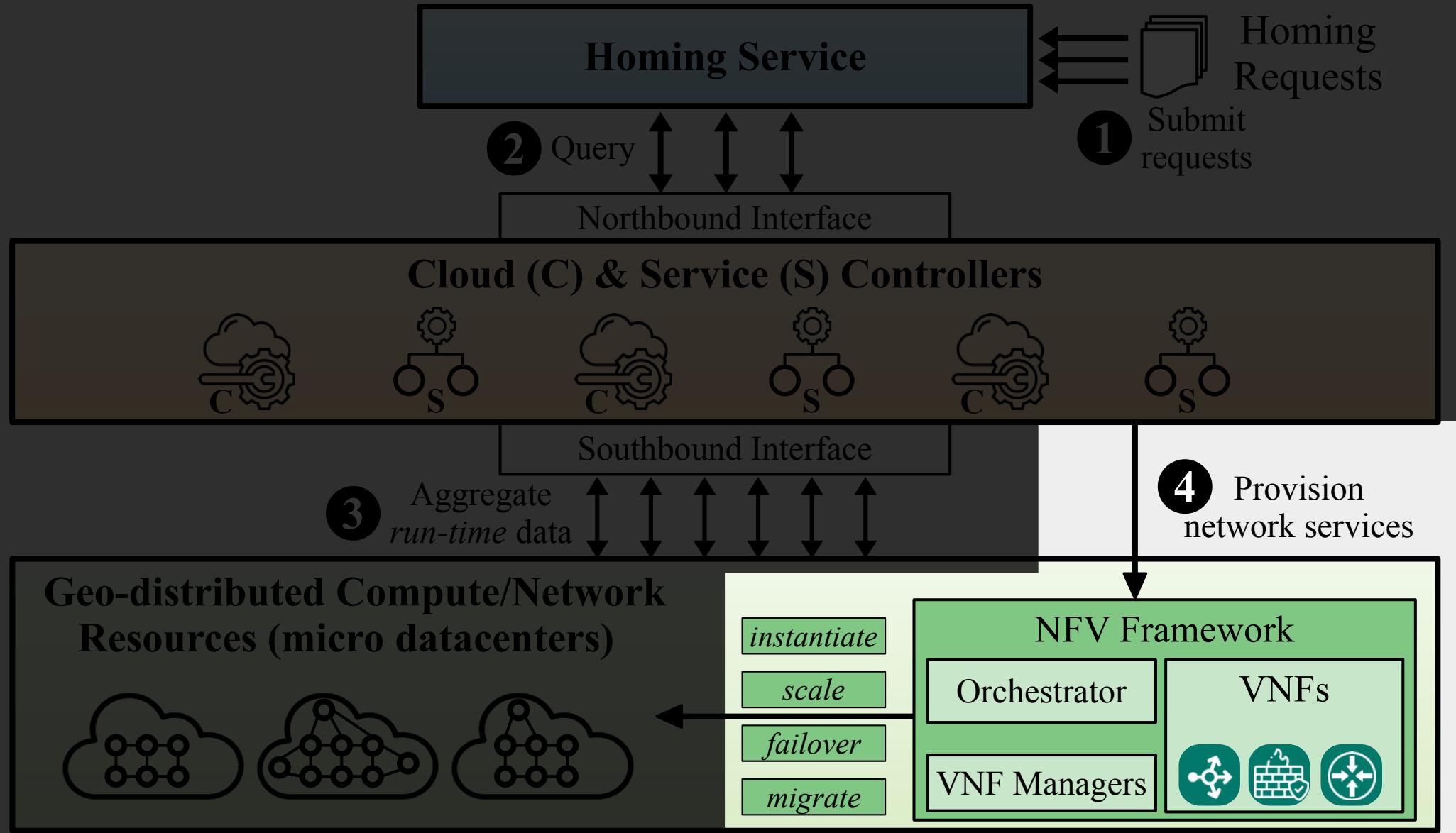


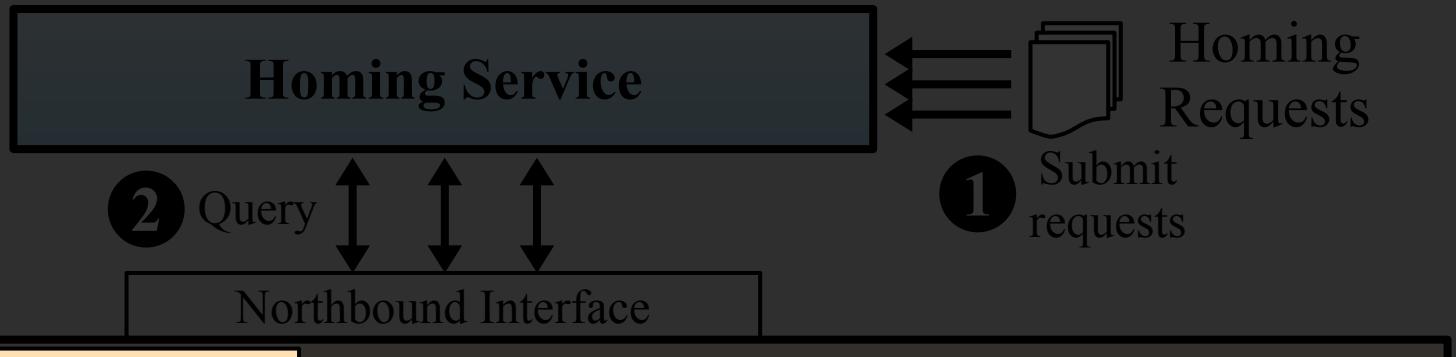
4 Major Challenges

4 Provision network services

Geo-distributed Compute/Network Resources (micro datacenters)







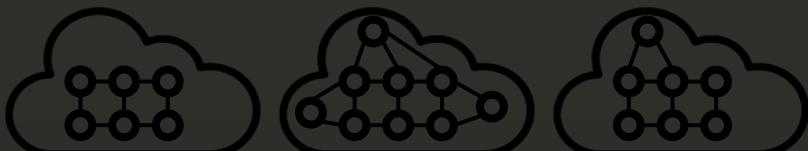
Challenge-1

How can we provision truly *elastic* and *agile* network services?

3 Aggregate
run-time data

4 Provision
network services

Geo-distributed Compute/Network Resources (micro datacenters)



instantiate

scale

failover

migrate

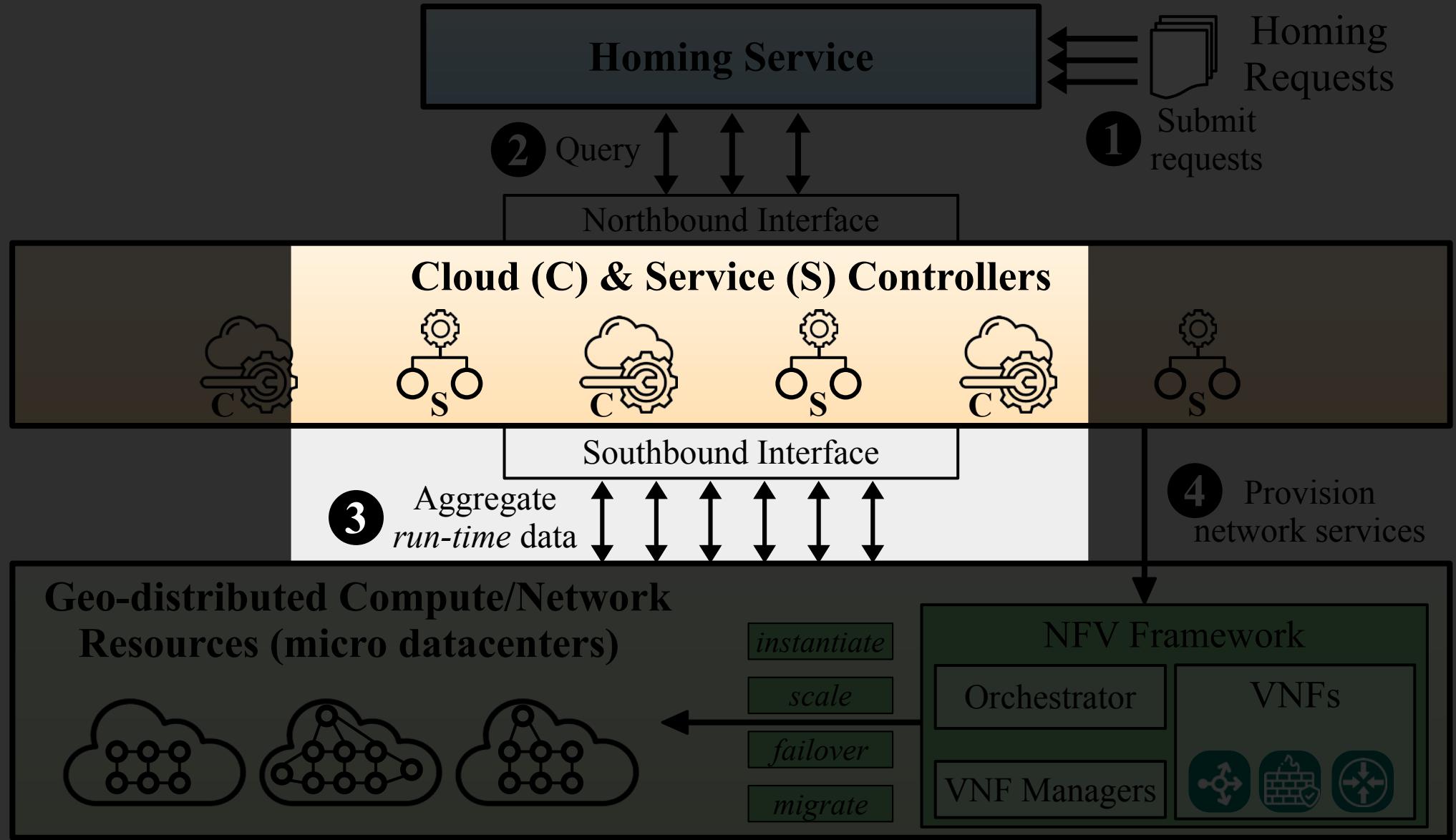
NFV Framework

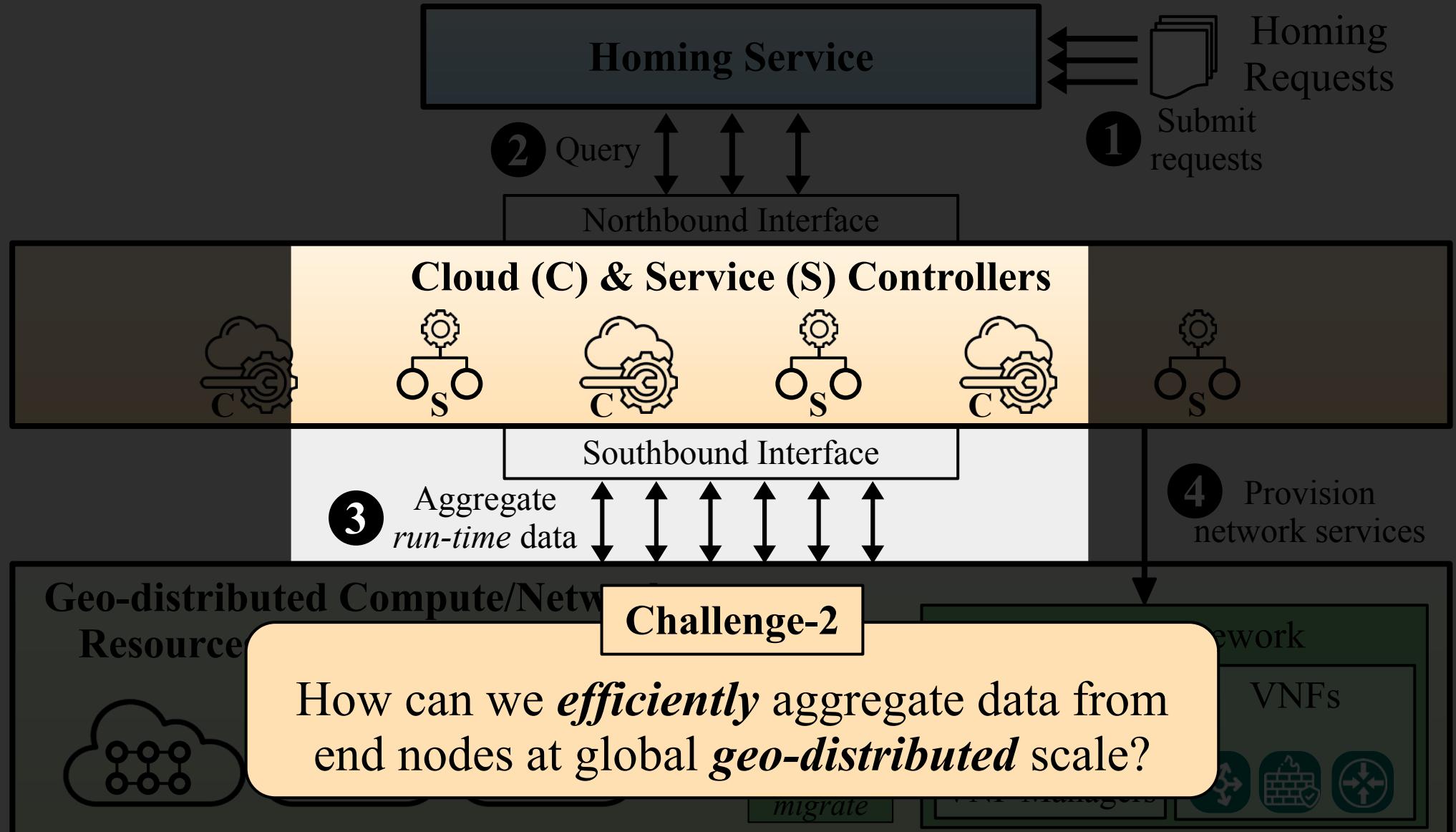
Orchestrator

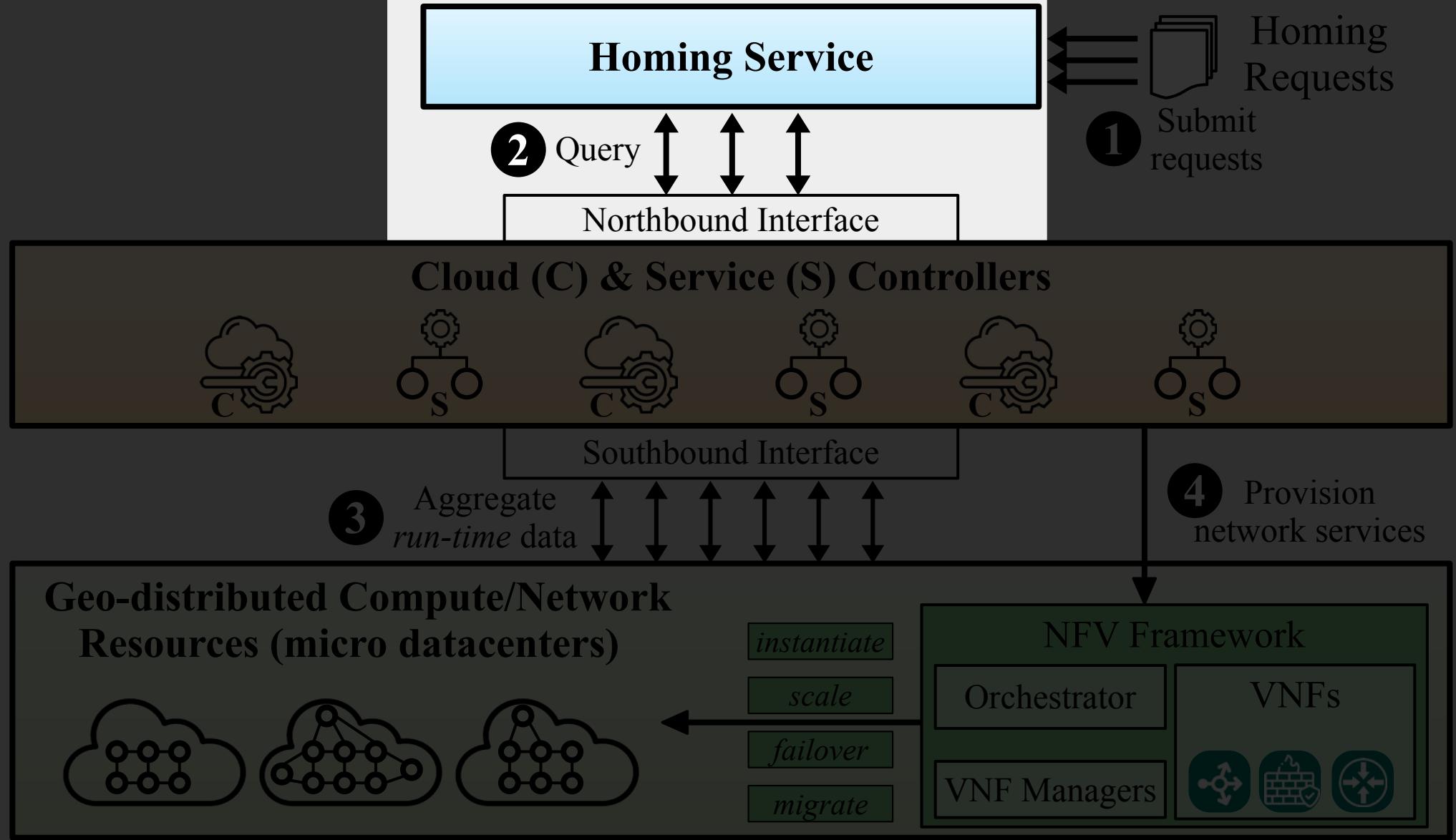
VNFs

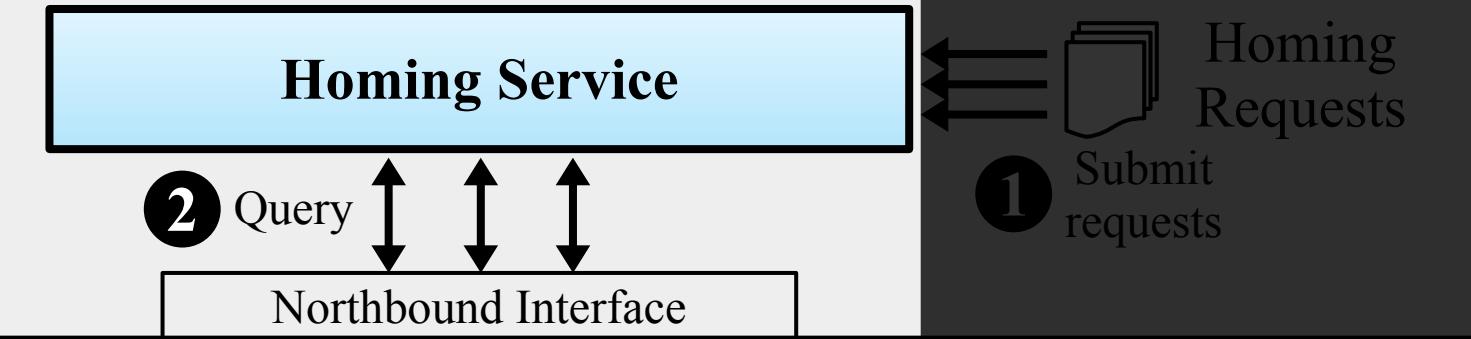
VNF Managers



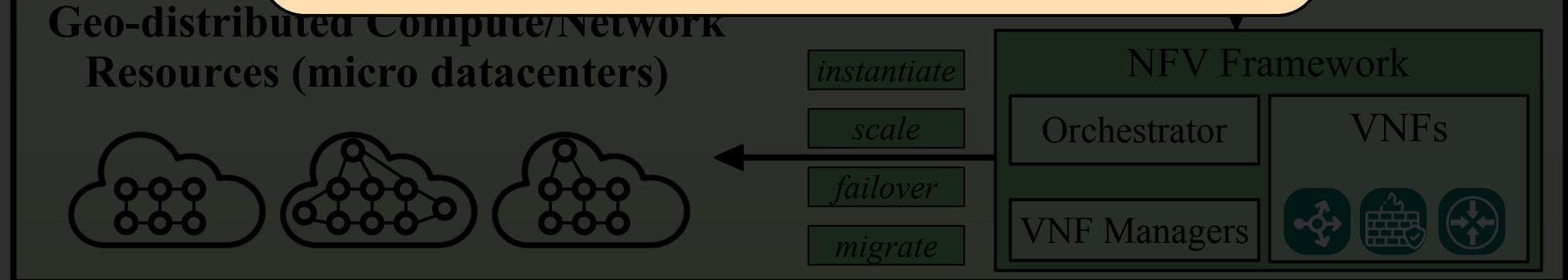


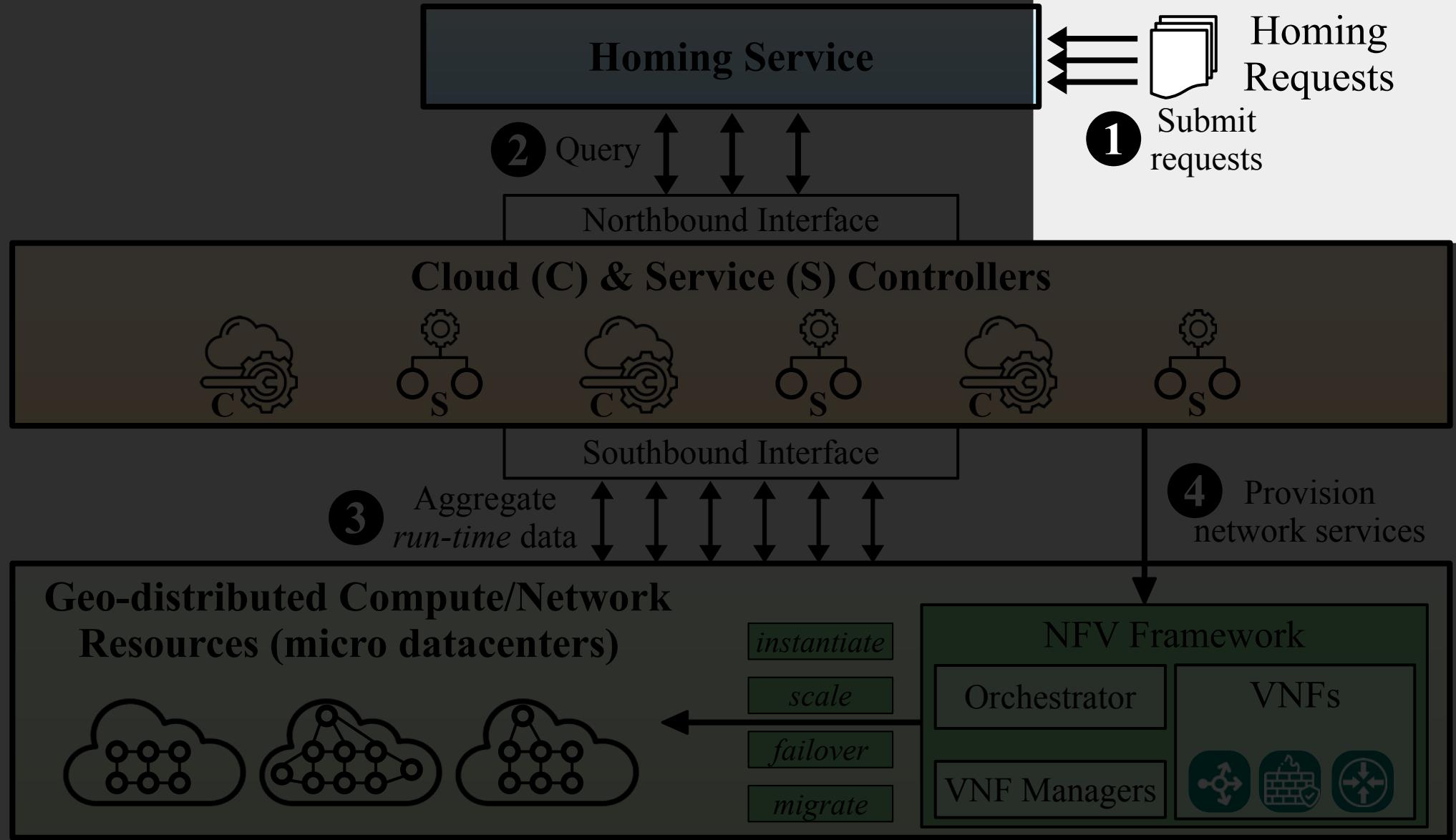


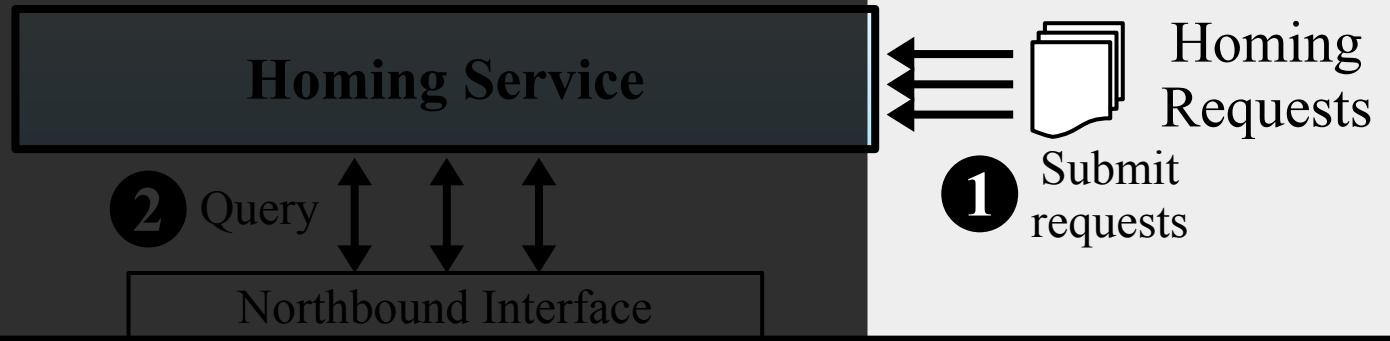




How can we *gain info* to make informed homing decisions without placing significant (querying) *load* on controllers?





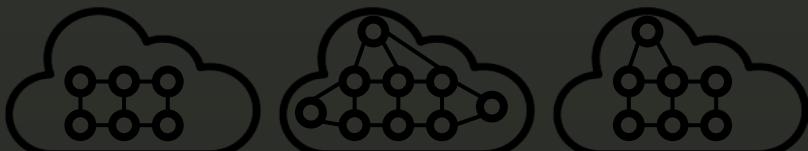


Cloud (C) & Service (S) Controllers

Challenge-4

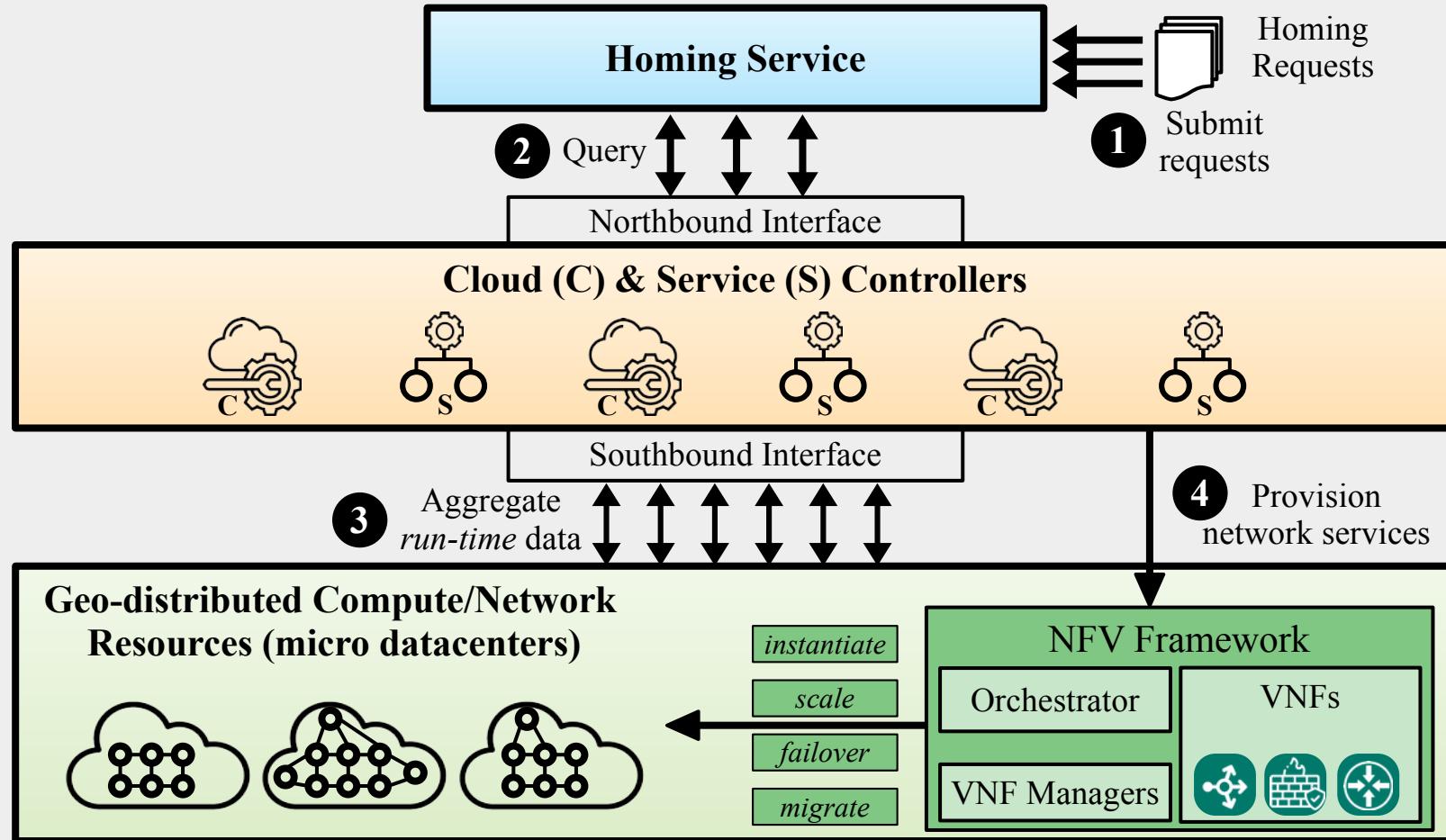
How can we account for *dependencies* between homing requests to make the homing process even *more efficient*?

Geo-distributed Compute/Network Resources (micro datacenters)

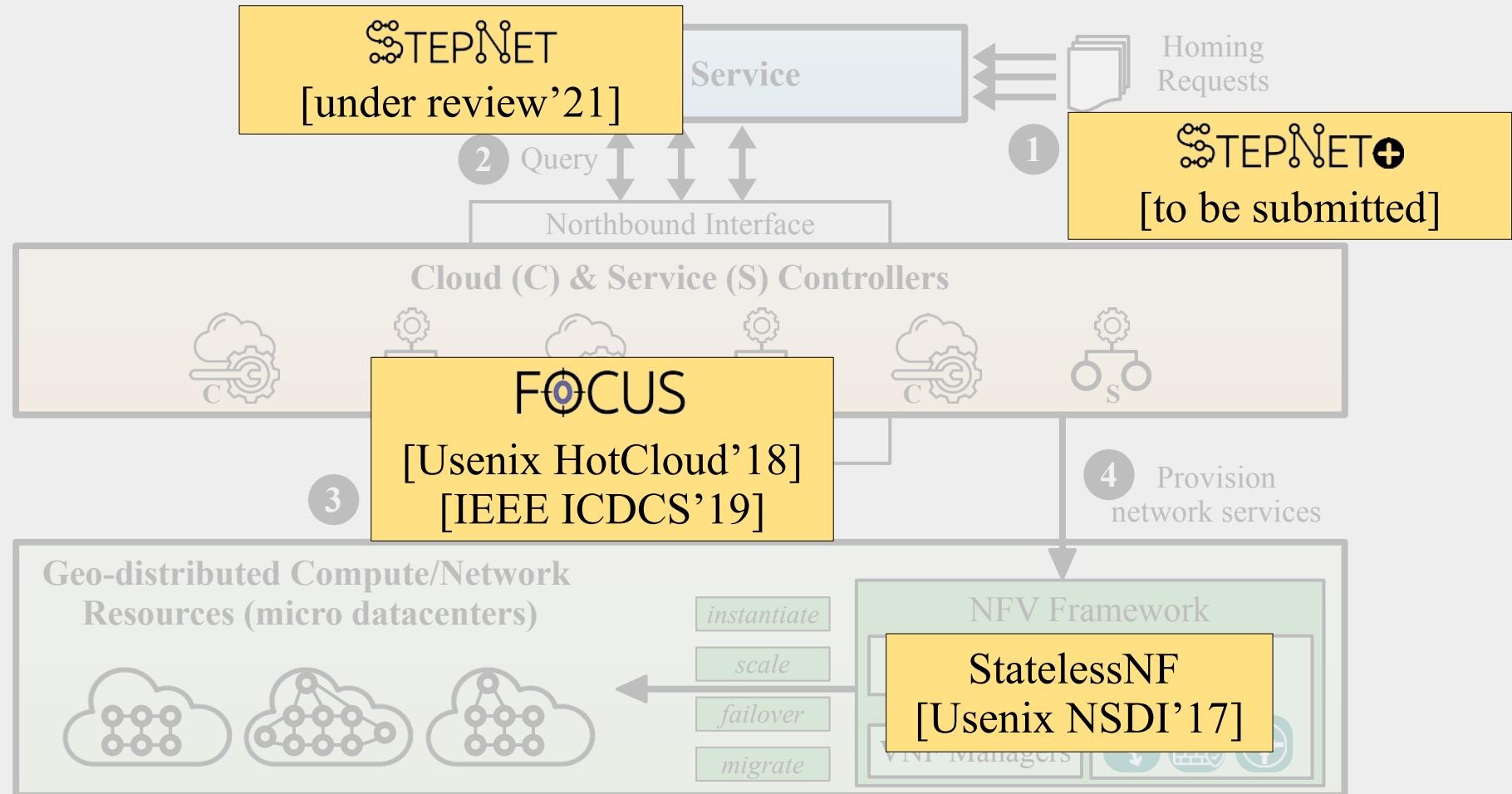


Contributions / Publications

Contributions / Publications

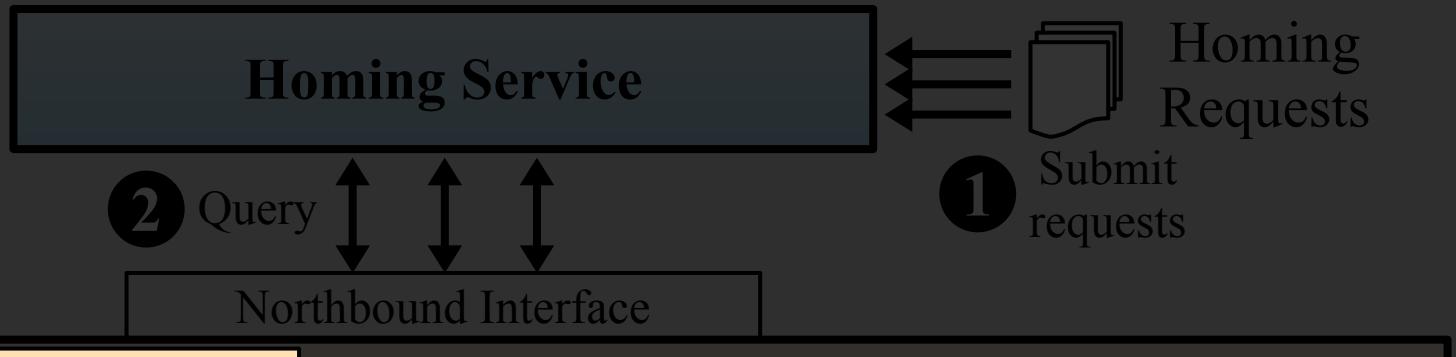


Contributions / Publications



Proposed Solutions

High Level View



Challenge-1

How can we provision truly *elastic* and *agile* network services?

- 3 Aggregate
run-time data

- 4 Provision
network services

Geo-distributed Compute/Network Resources (micro datacenters)



instantiate
scale
failover
migrate

NFV Framework

Orchestrator

VNFs

VNF Managers



Stateless Network Functions

Breaking the tight coupling of state and processing

Overview

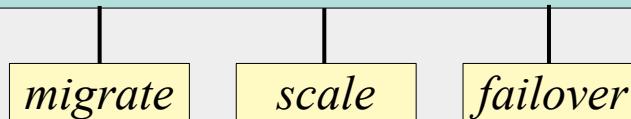
Virtualization enables us to package and provision
VNFS in *VMs* or docker *containers*

Overview

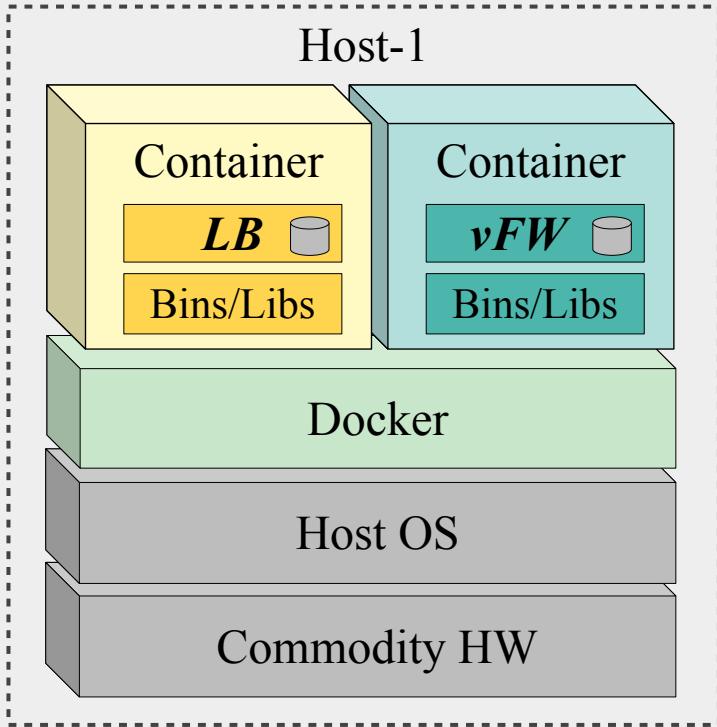
Virtualization enables us to package and provision
VNFS in *VMs* or docker *containers*



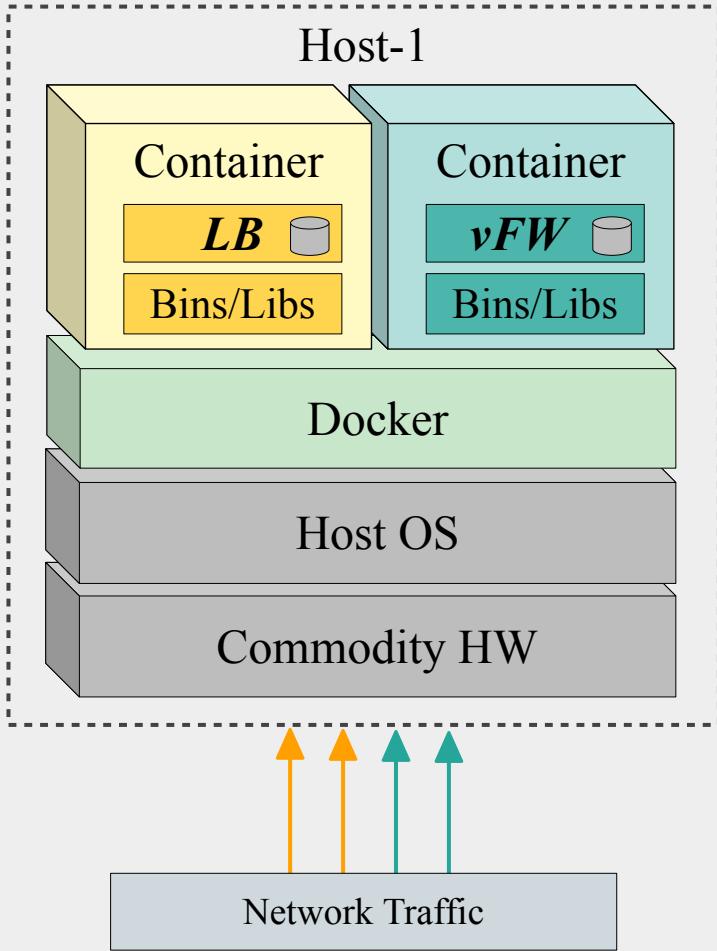
We can leverage cloud tech



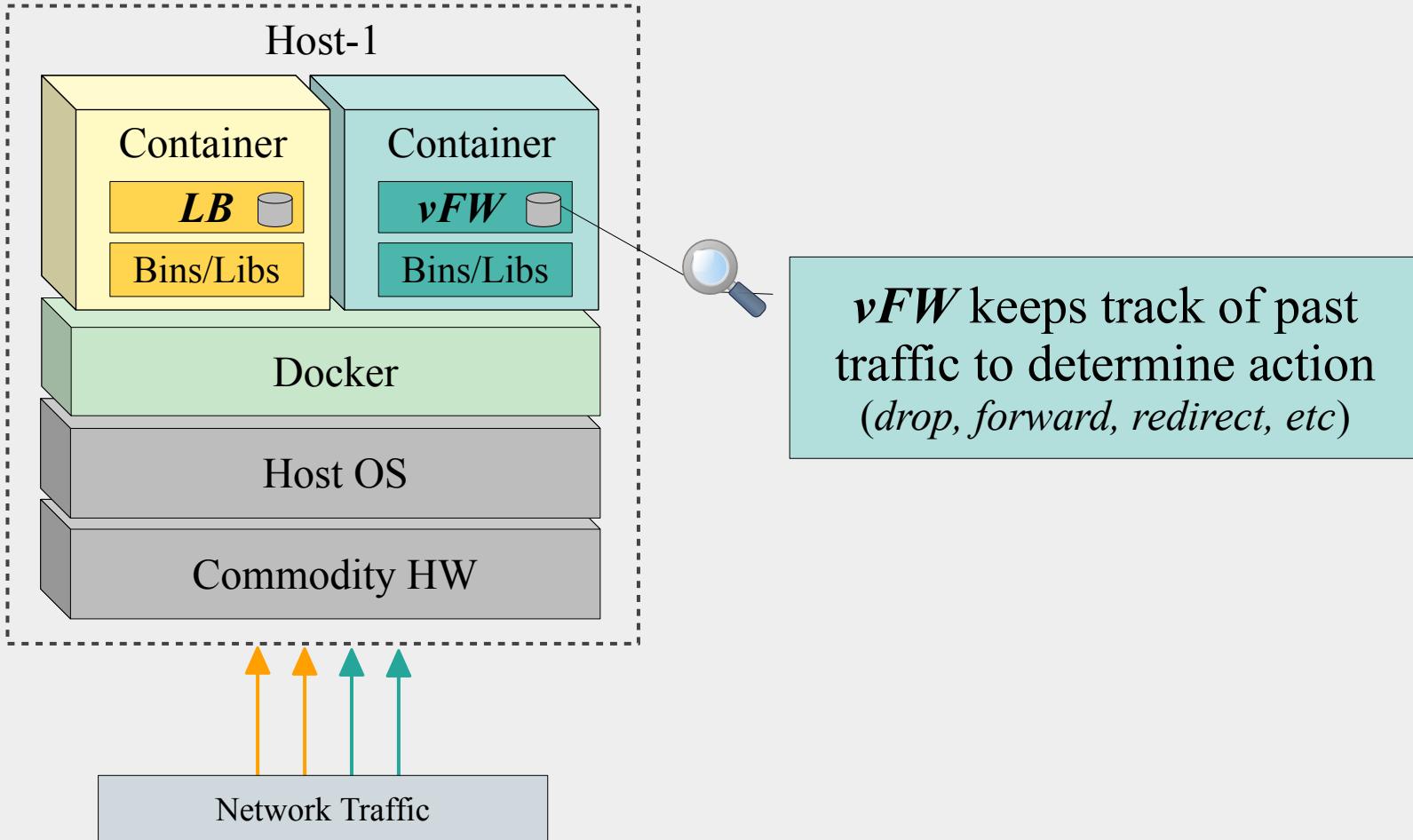
Overview



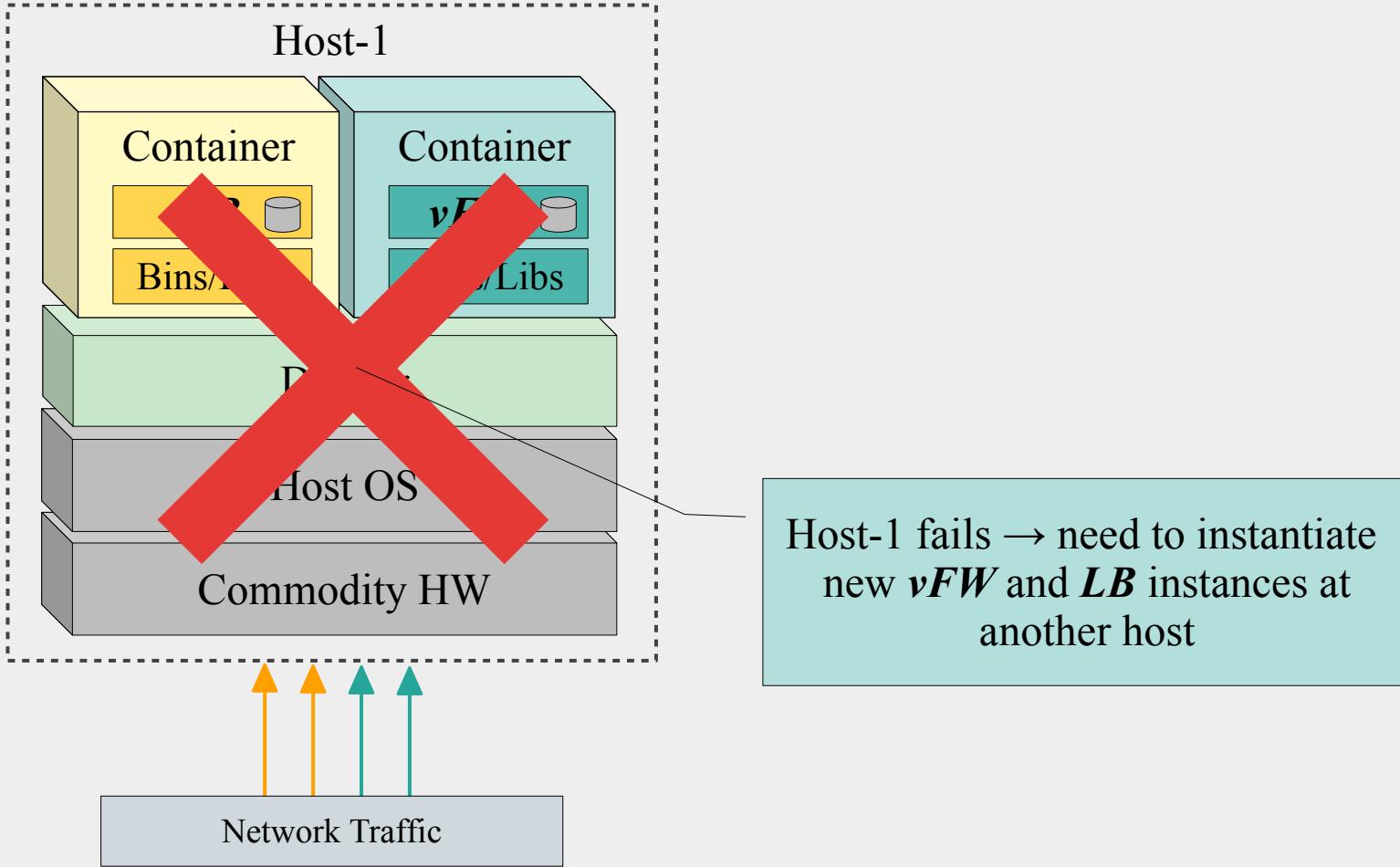
Overview



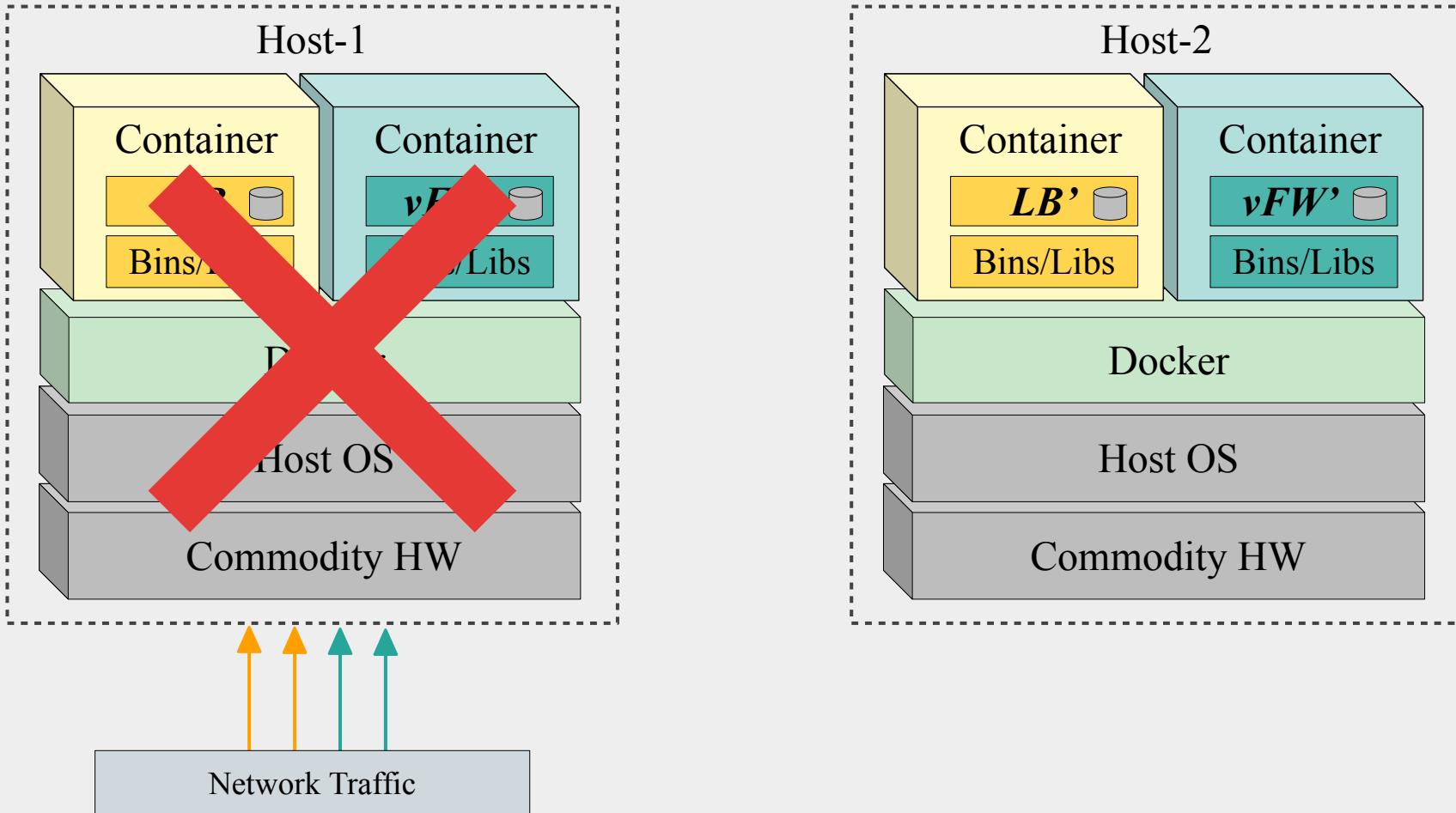
Overview



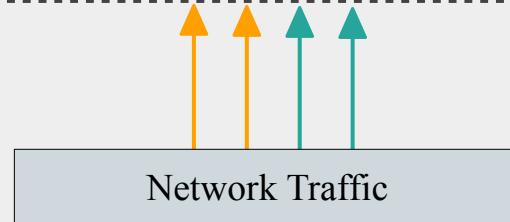
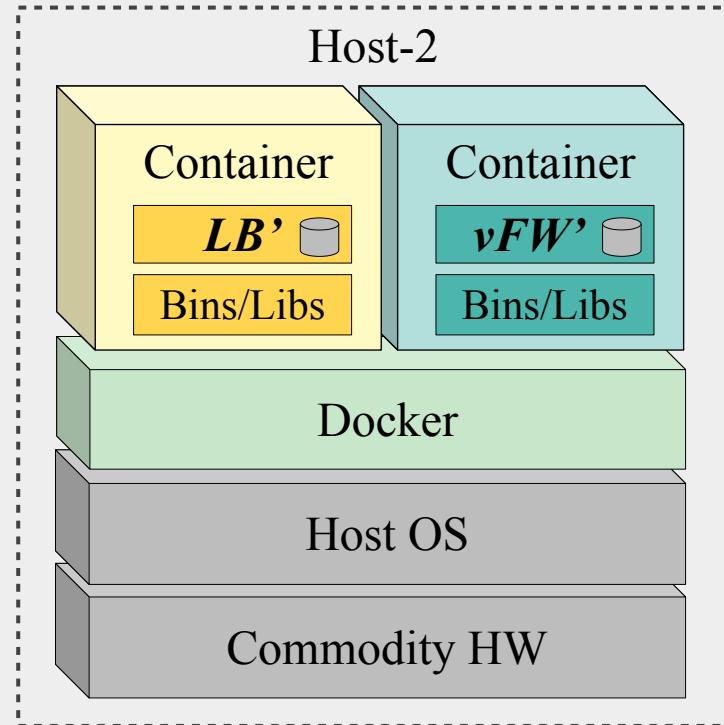
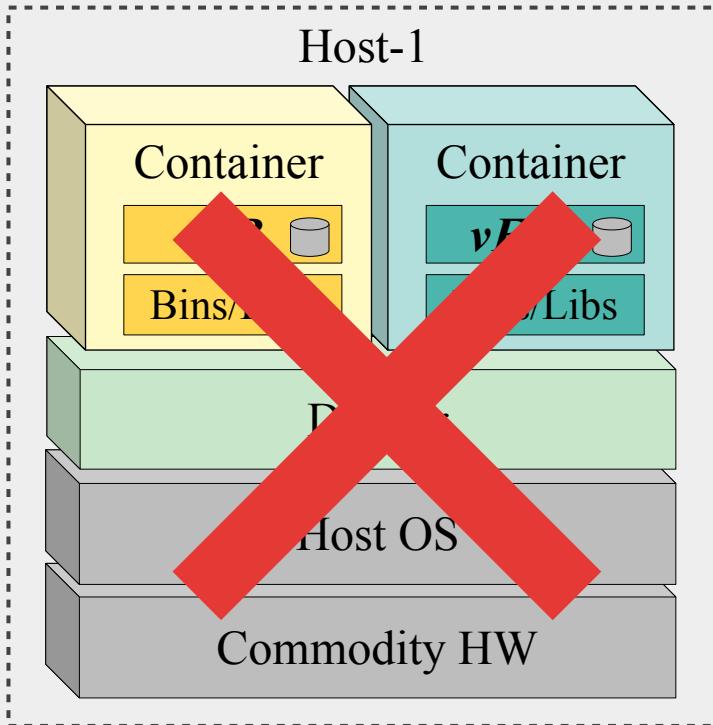
Overview



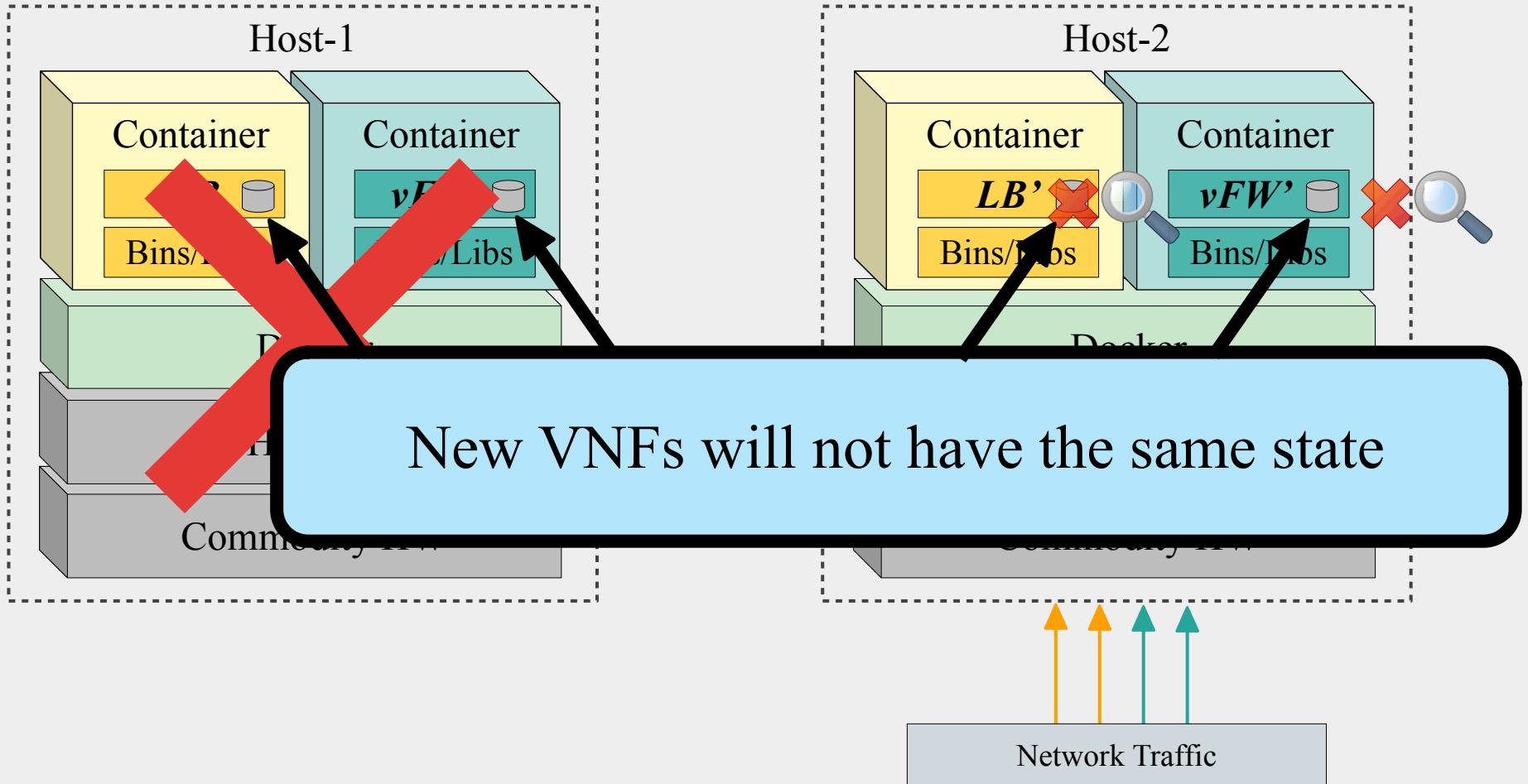
Overview



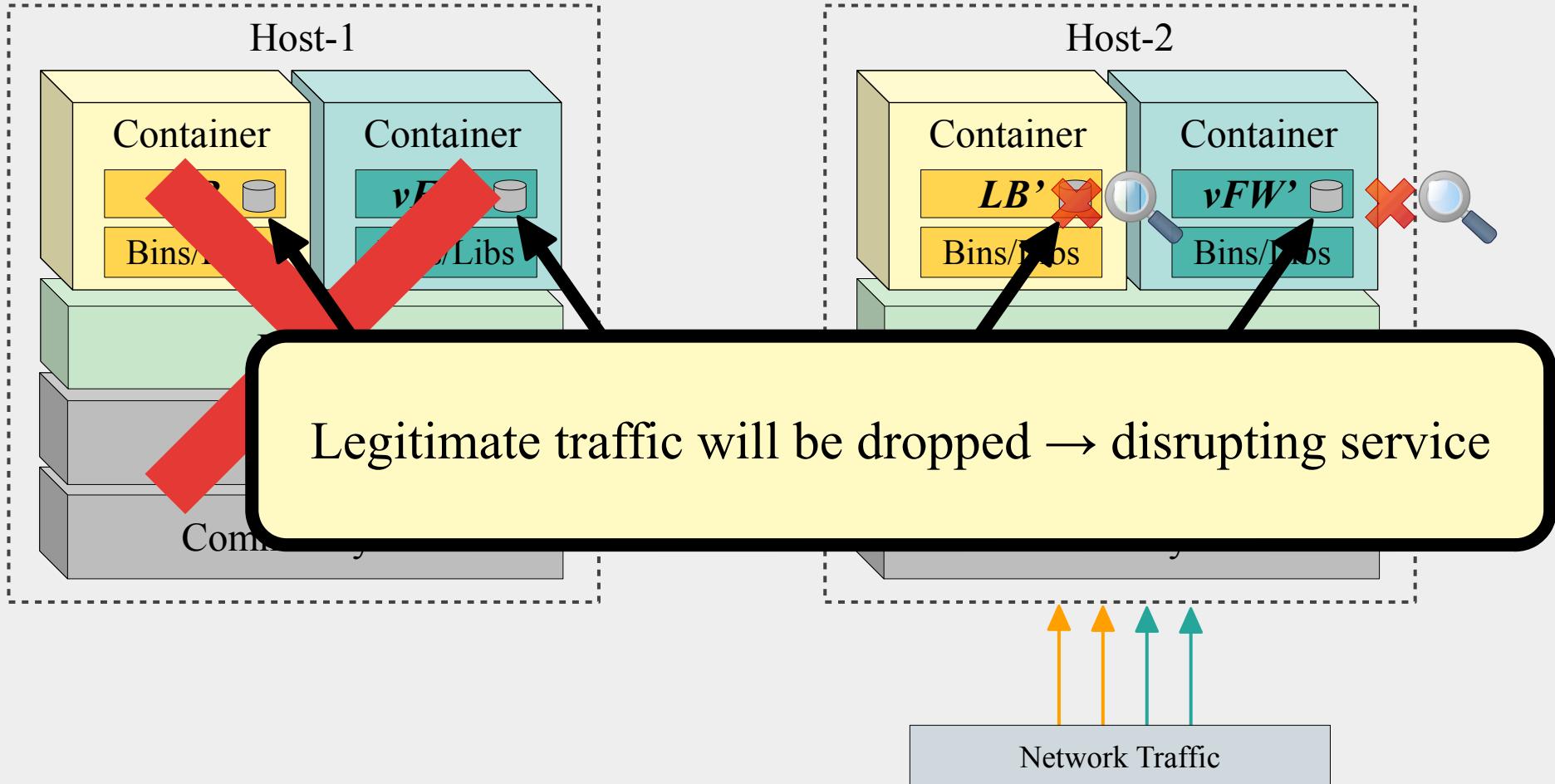
Overview



Challenge



Challenge



Other Solutions

State Checkpointing &
Replication [*Pico-SoCC'13*]

Packet Logging &
Replay [*FTMB-SIGCOMM'15*]

Other Solutions

State Checkpointing &
Replication [*Pico-SoCC'13*]

Packet Logging &
Replay [*FTMB-SIGCOMM'15*]

Frequently checkpoint
and replicate state

High latency overhead (10ms)



Other Solutions

State Checkpointing &
Replication [*Pico-SoCC'13*]

Frequently checkpoint
and replicate state

High latency overhead (10ms)

Packet Logging &
Replay [*FTMB-SIGCOMM'15*]

Log incoming packets
and replay upon failure

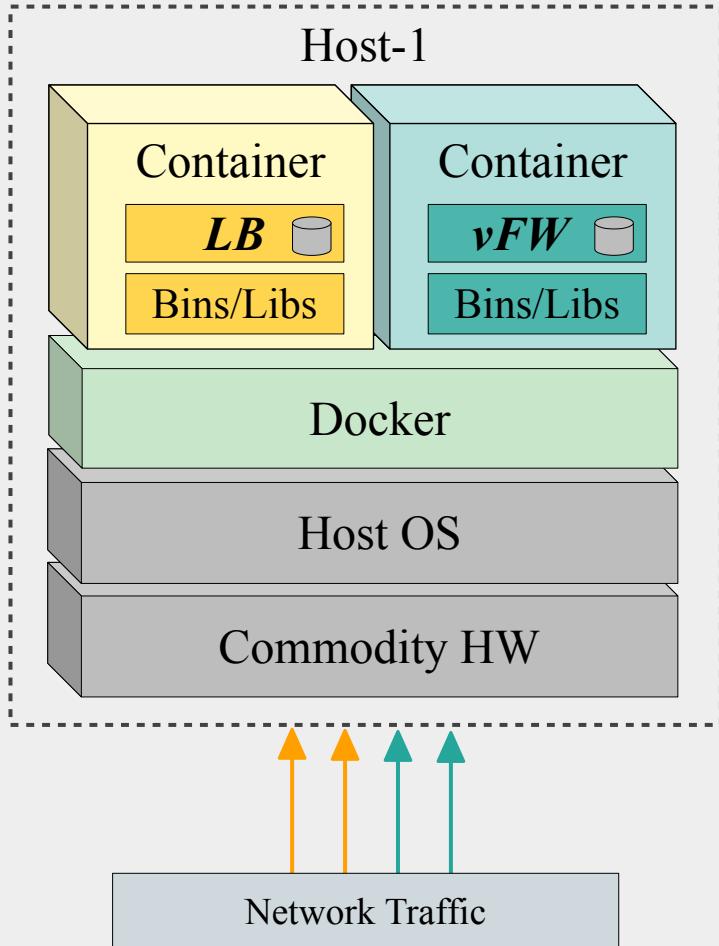
Long mean time to recover (MTTR)

Stateless Network Functions

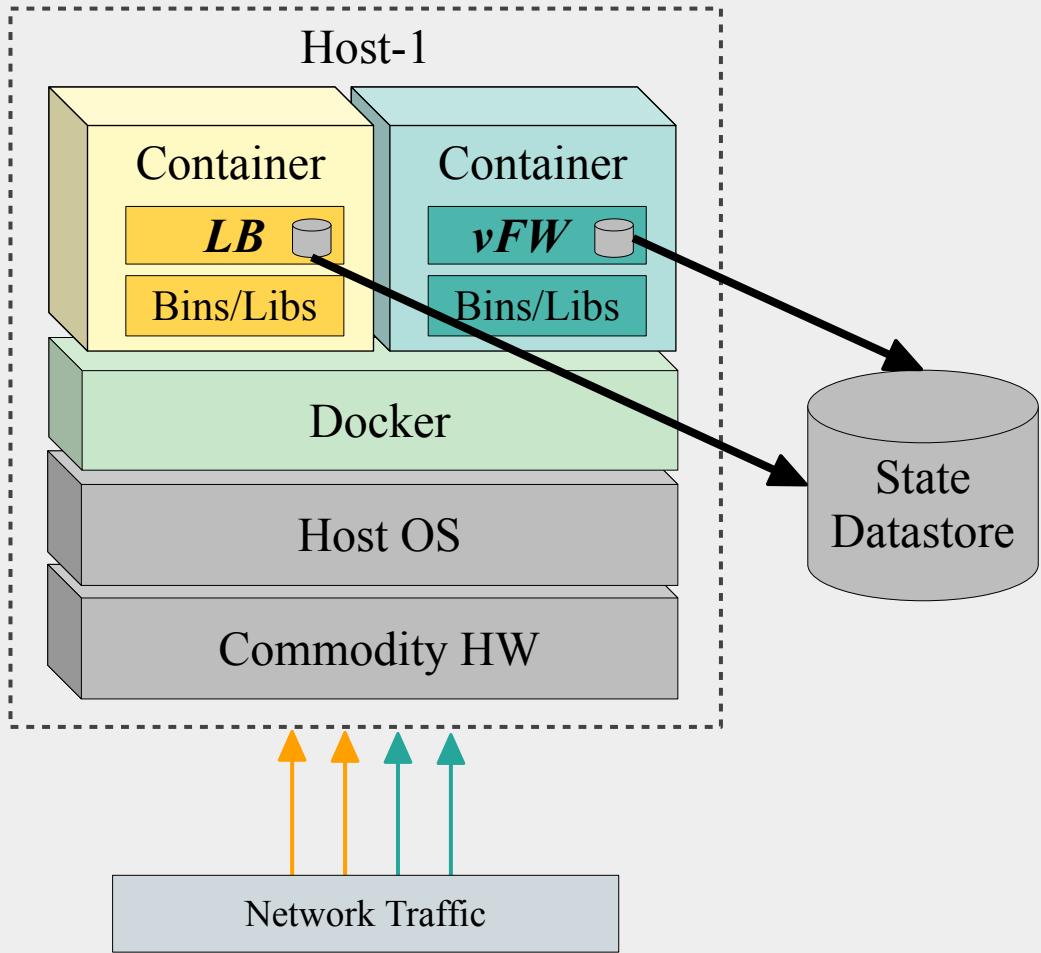
i

Deal with the **root** of the problem:
break tight coupling of *state* and *processing*

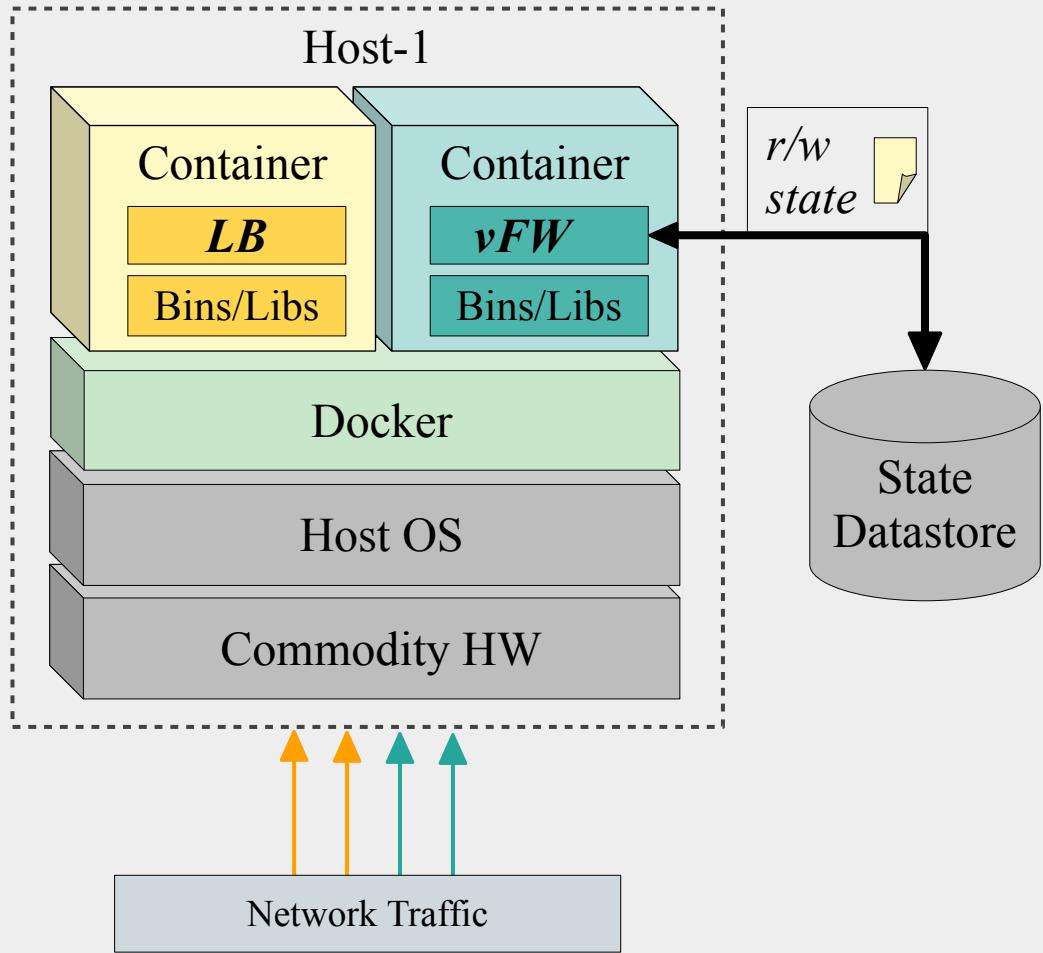
Stateless Network Functions



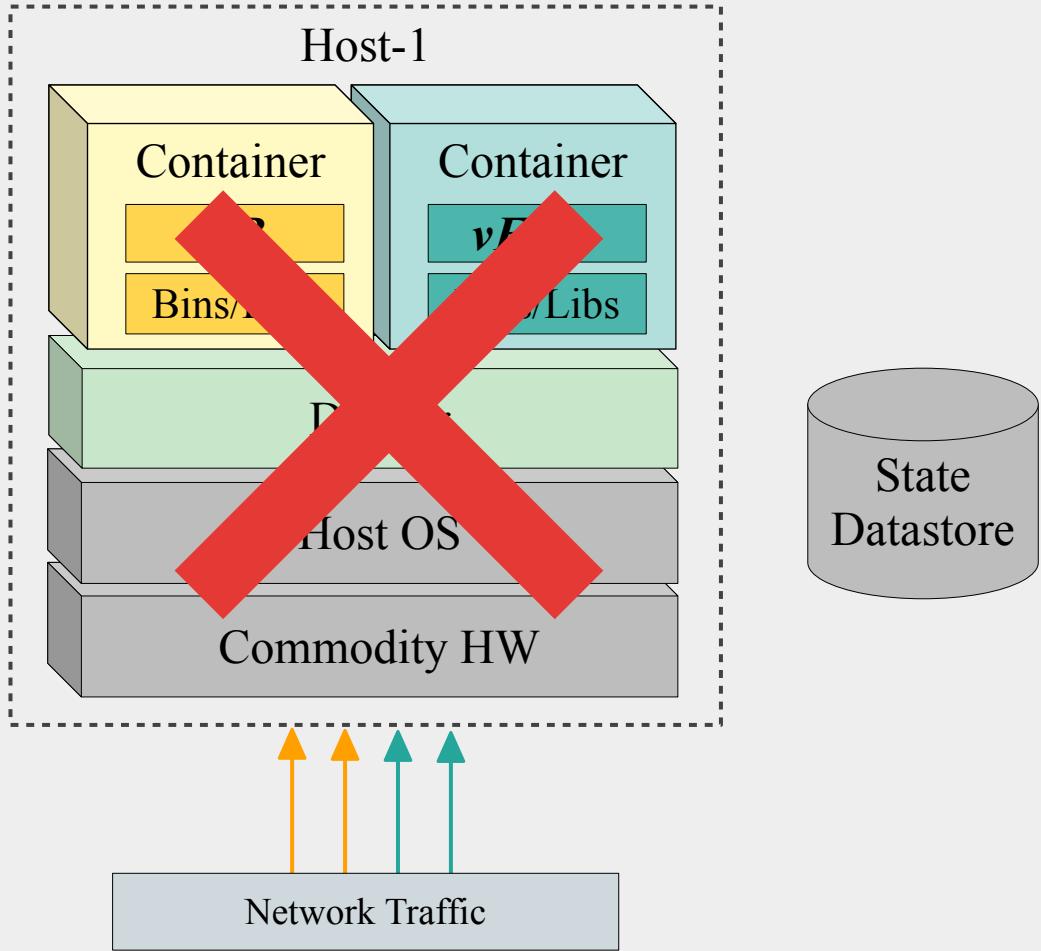
Stateless Network Functions



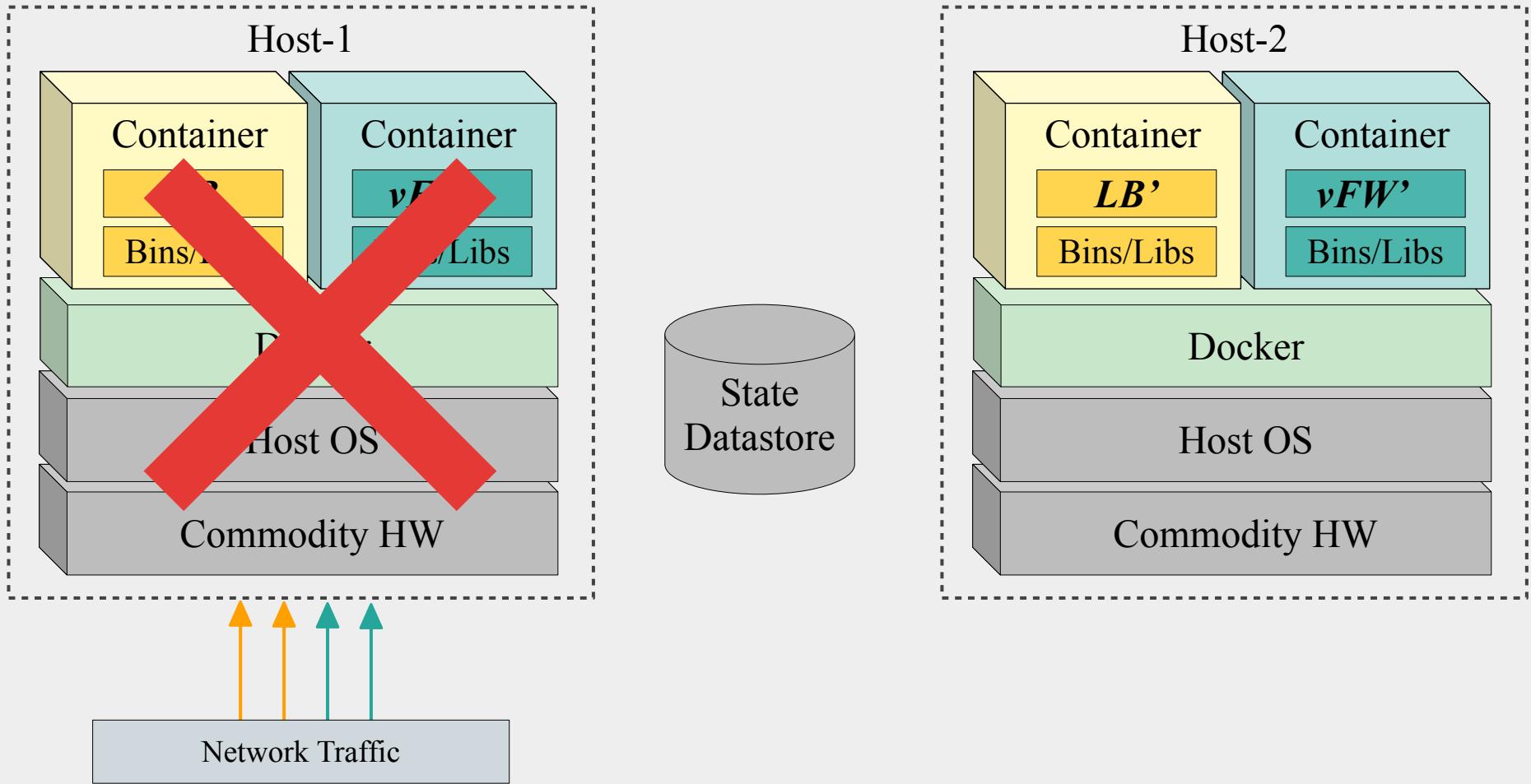
Stateless Network Functions



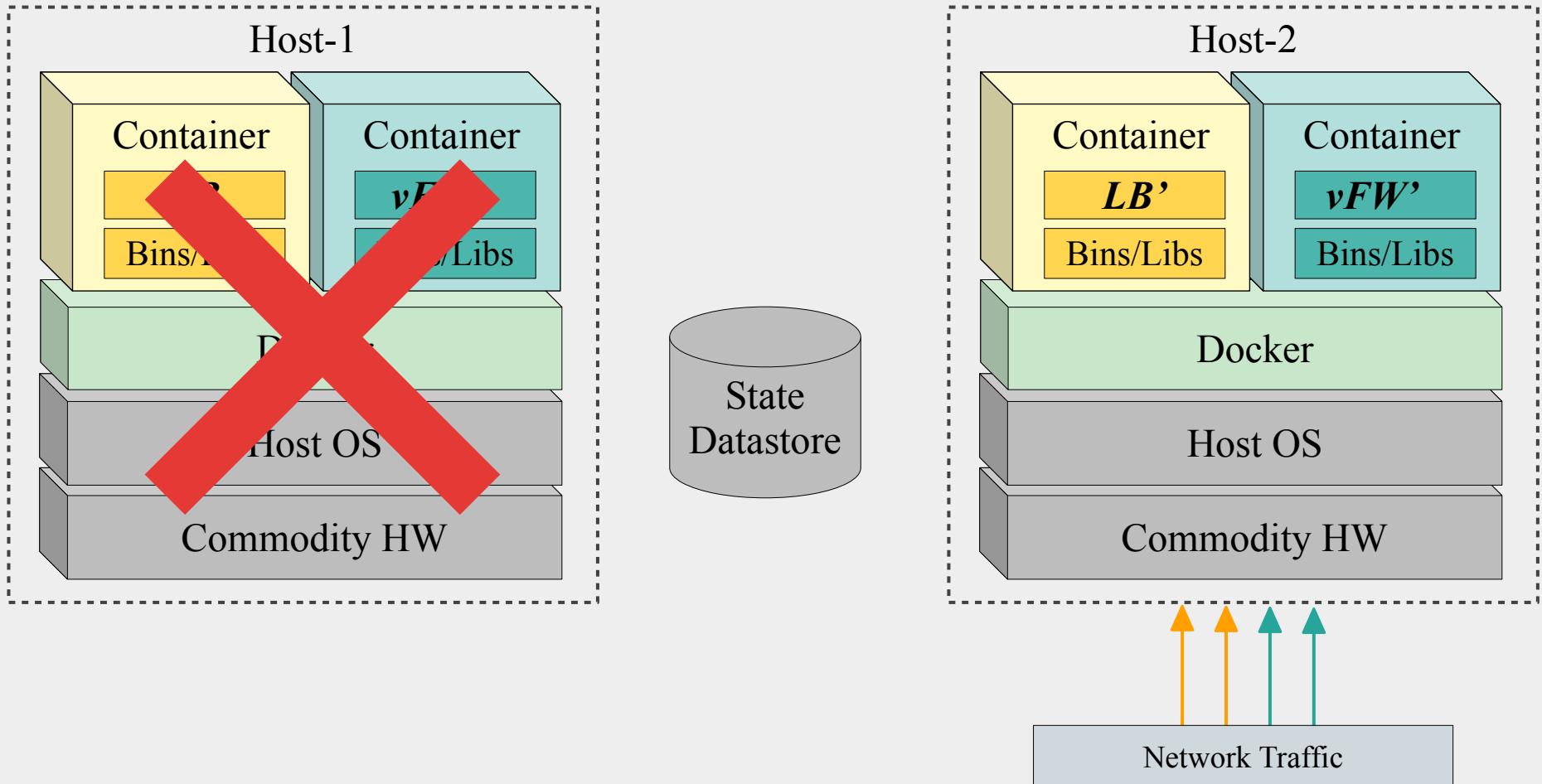
Stateless Network Functions



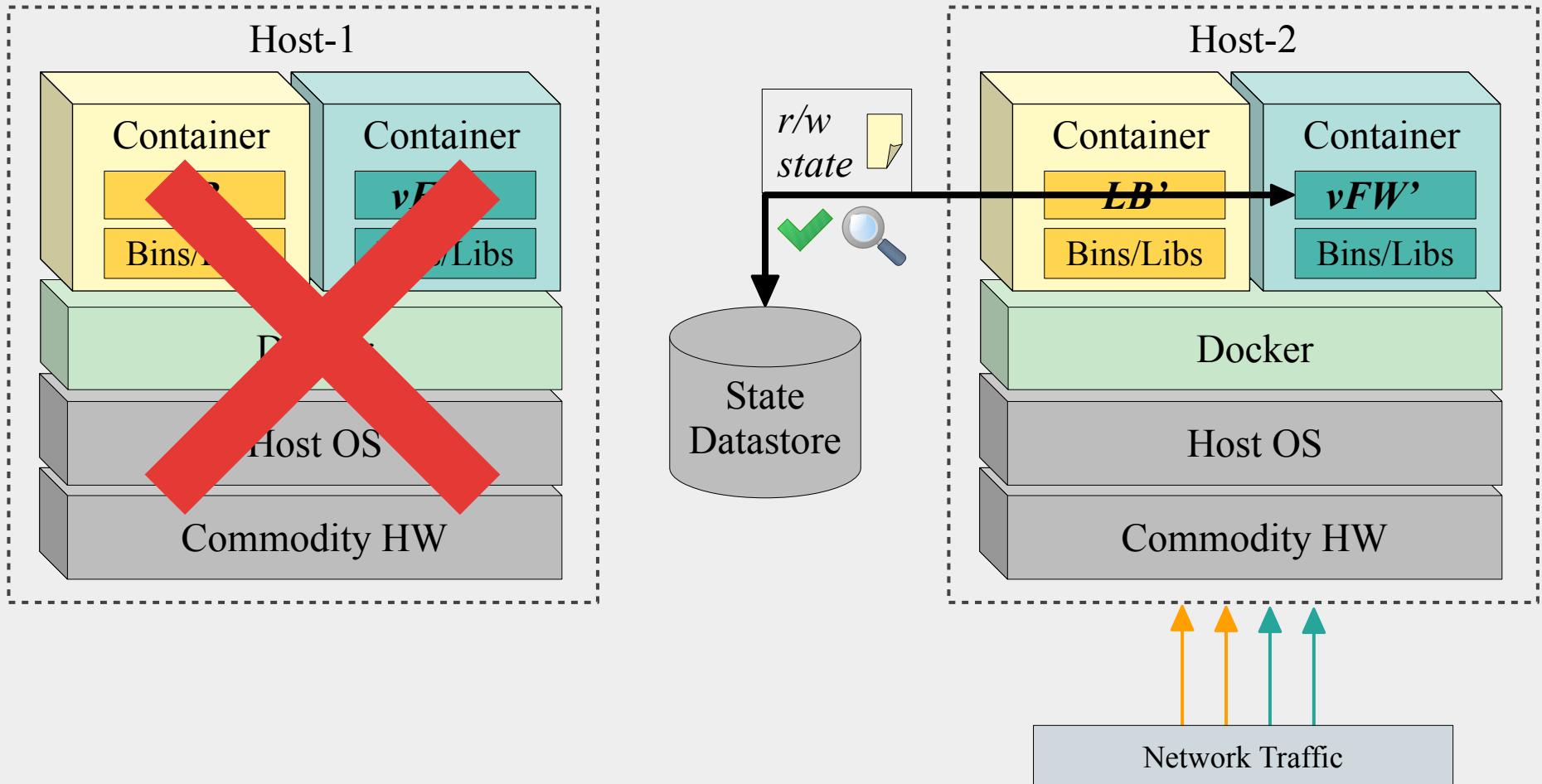
Stateless Network Functions



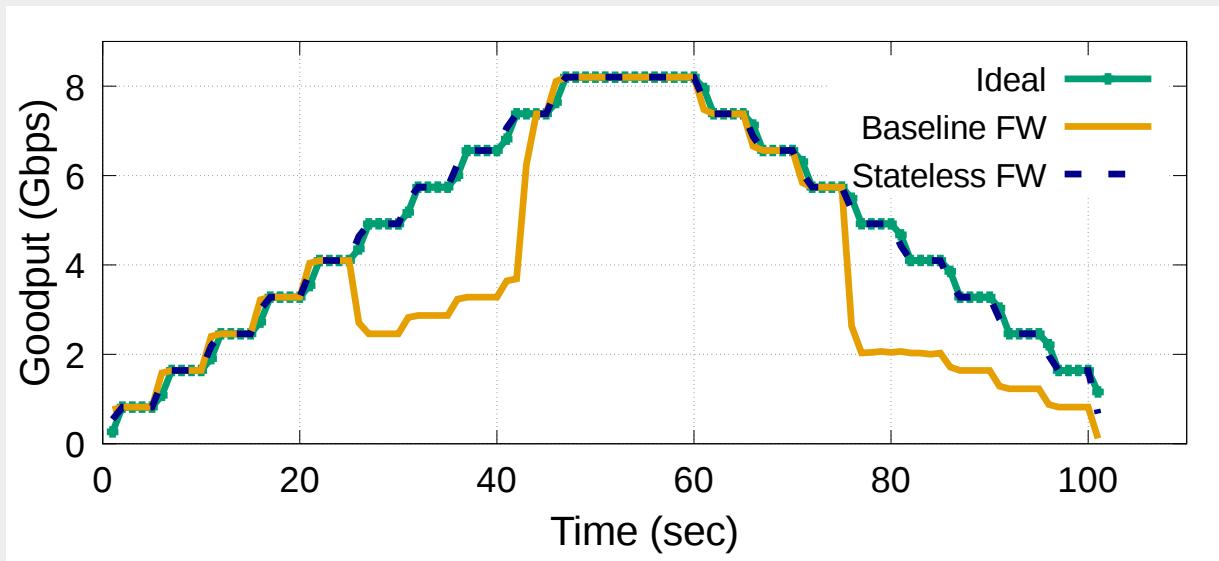
Stateless Network Functions



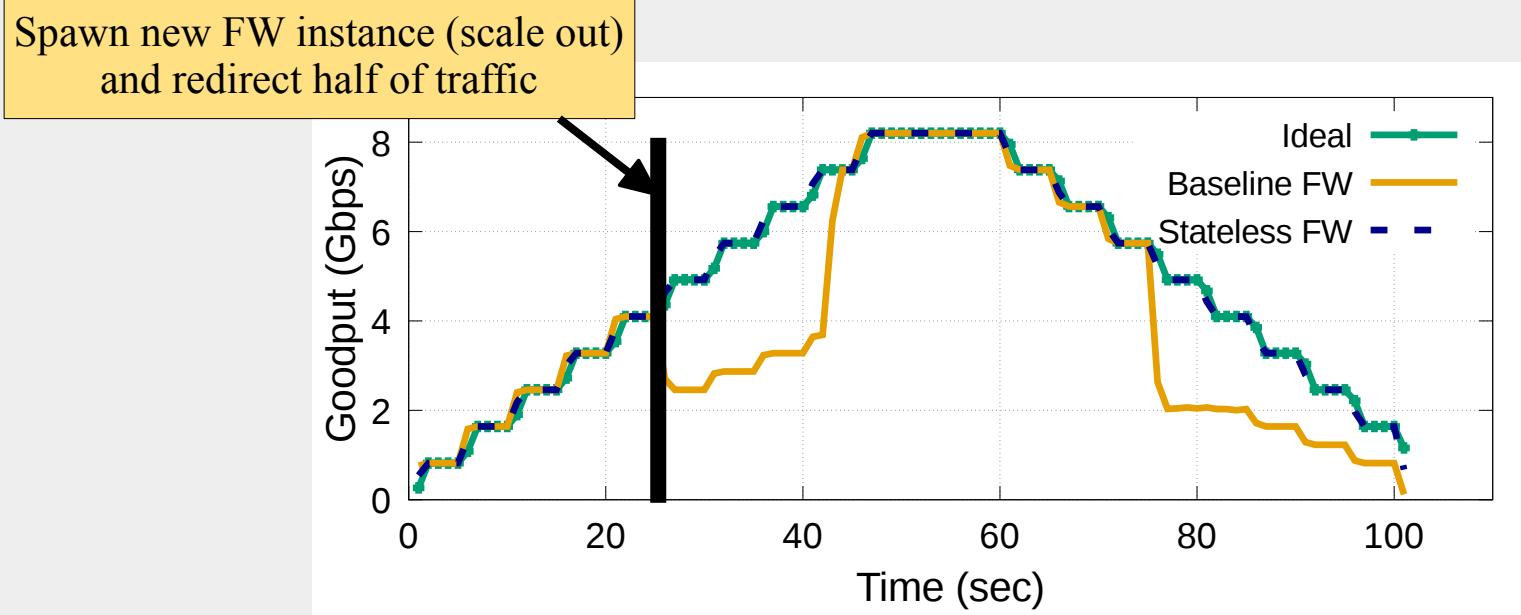
Stateless Network Functions



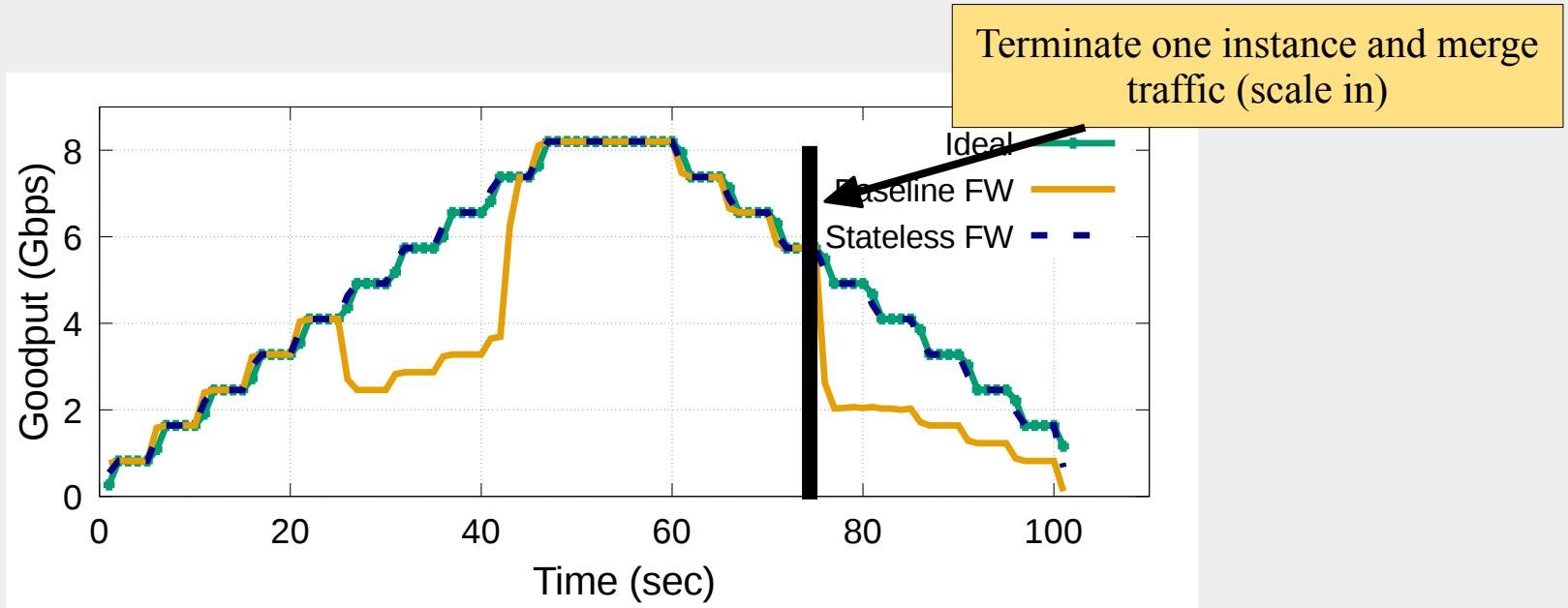
Stateless Network Functions



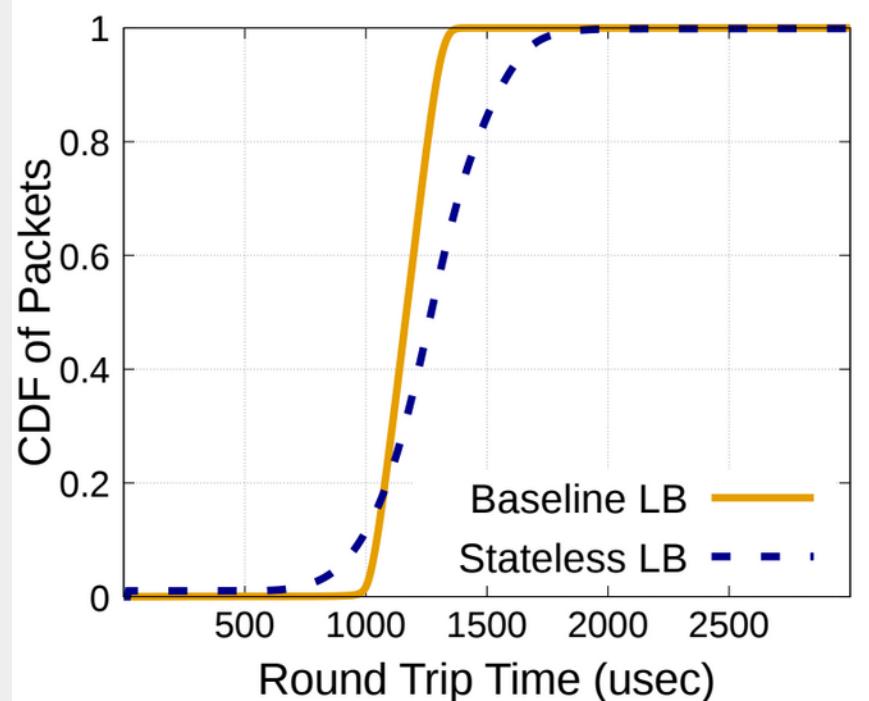
Stateless Network Functions



Stateless Network Functions



Stateless Network Functions



Stateless Network Functions

With stateless network functions, NSPs can seamlessly manage (failover, scale, migrate, etc) homed network services

Stateless Network Functions

Contributions

1

Designed a novel architecture for virtual network functions that provides true elasticity and agility

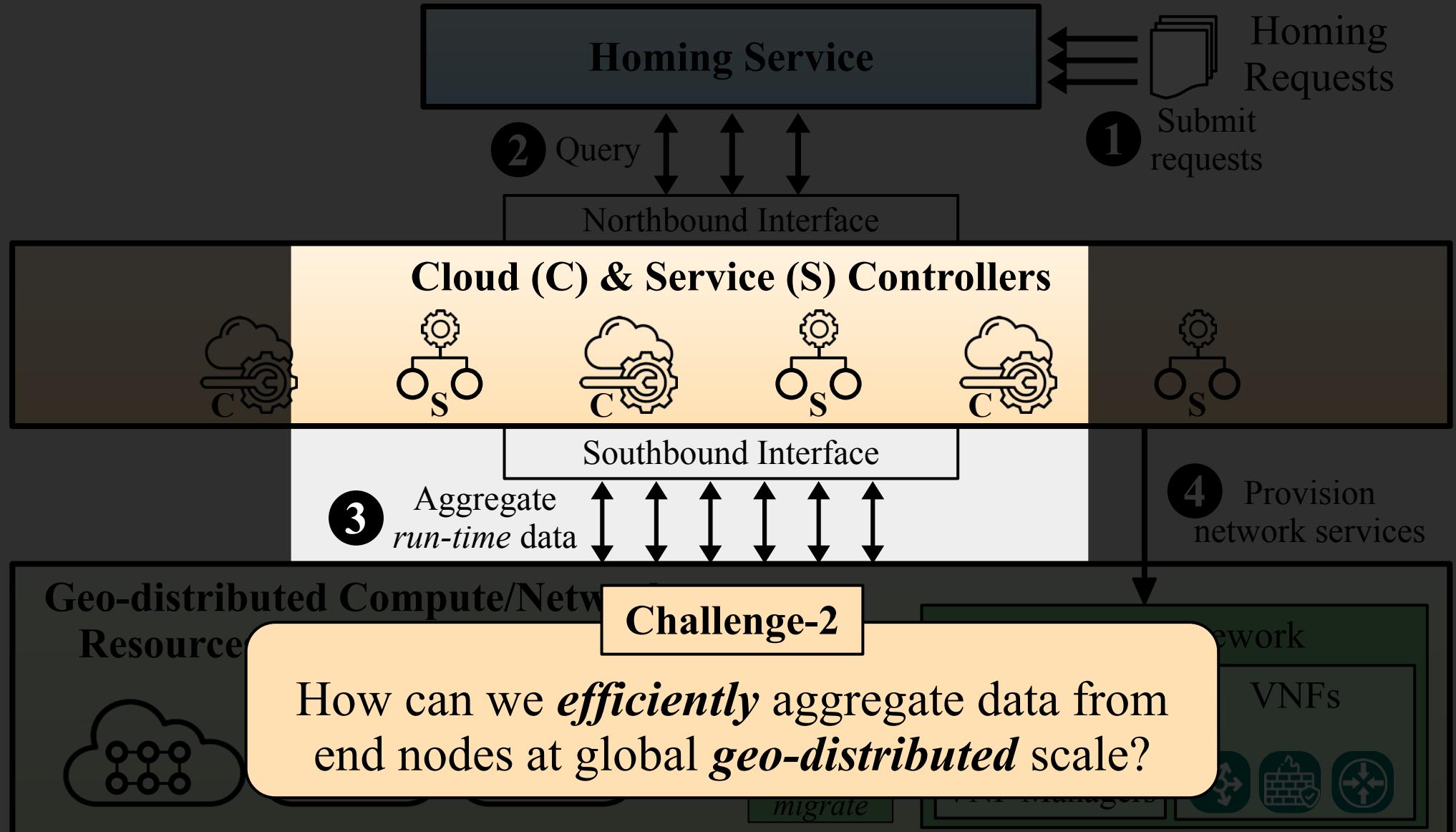
2

Introduced various optimizations to improve the performance of stateless VNFs

3

Evaluated and demonstrated StatlessNF's ability to provide seamless elasticity actions (failover, scale, etc)

Publications: [Usenix NSDI'17]



FOCUS

Scalable search service for highly
dynamic geo-distributed state

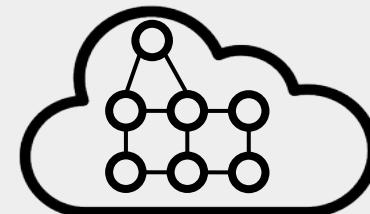
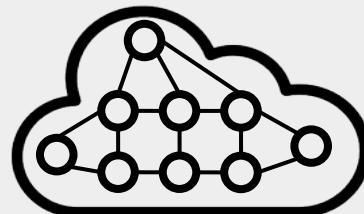
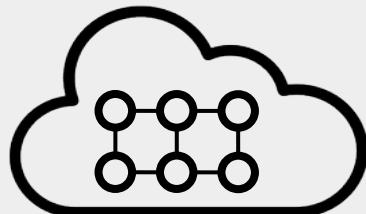
Overview

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Controller



Overview

Q

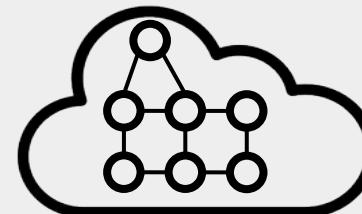
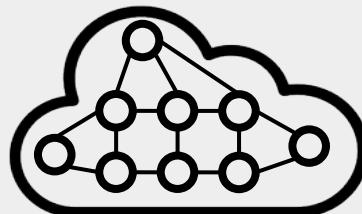
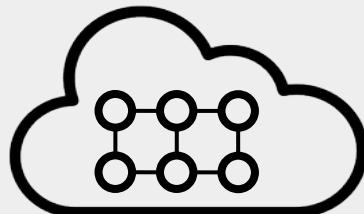
Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Controller



How do we know
which nodes satisfy
this query?



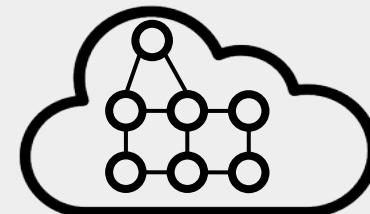
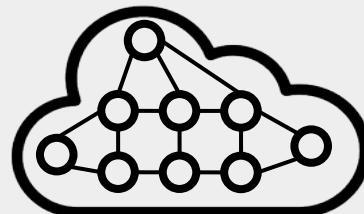
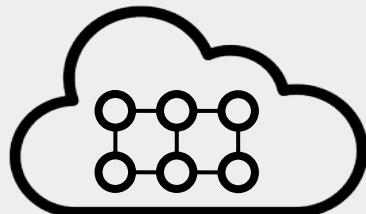
Overview

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Controller



Approach-1

Frequently Push Updates

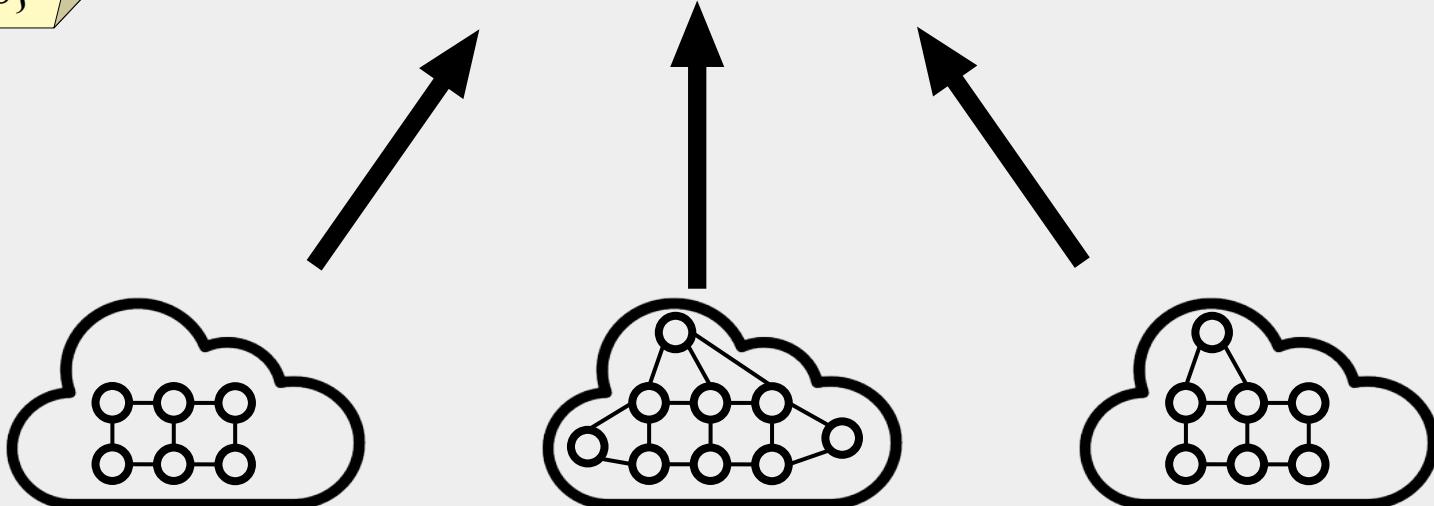
Overview

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Controller



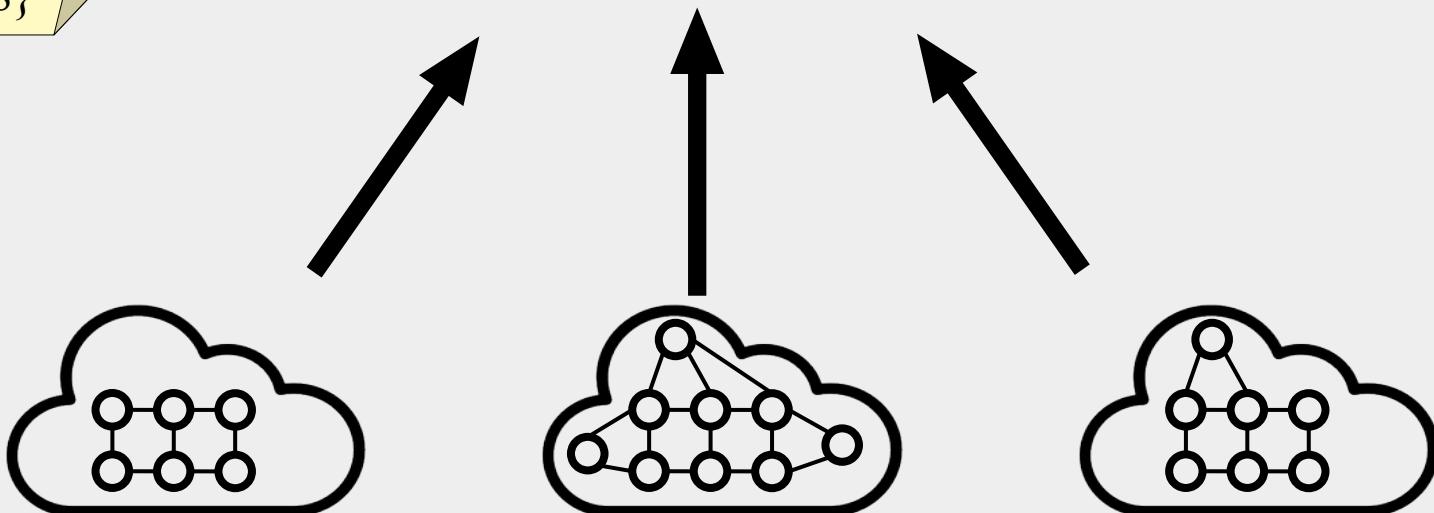
Approach-1

Frequently Push Updates

Overview

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



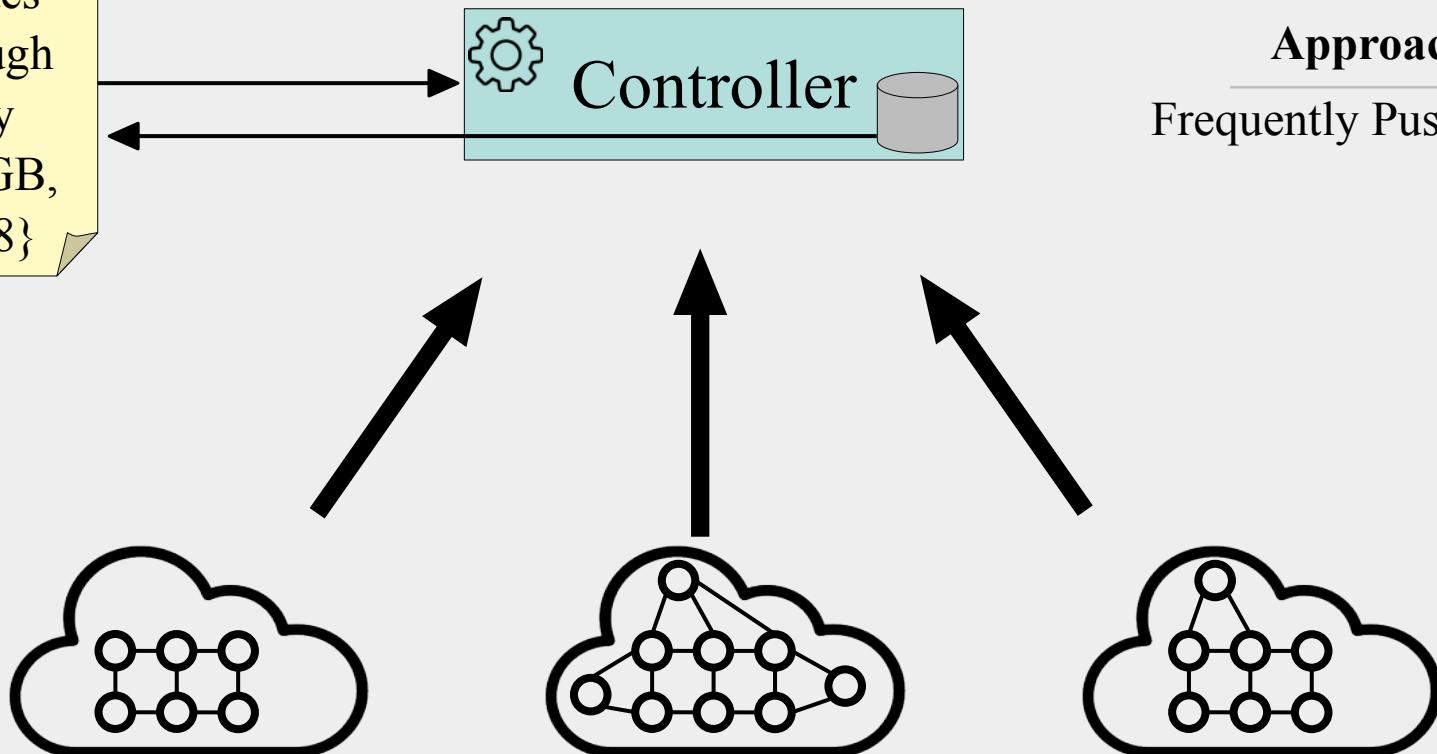
Approach-1

Frequently Push Updates

Overview

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



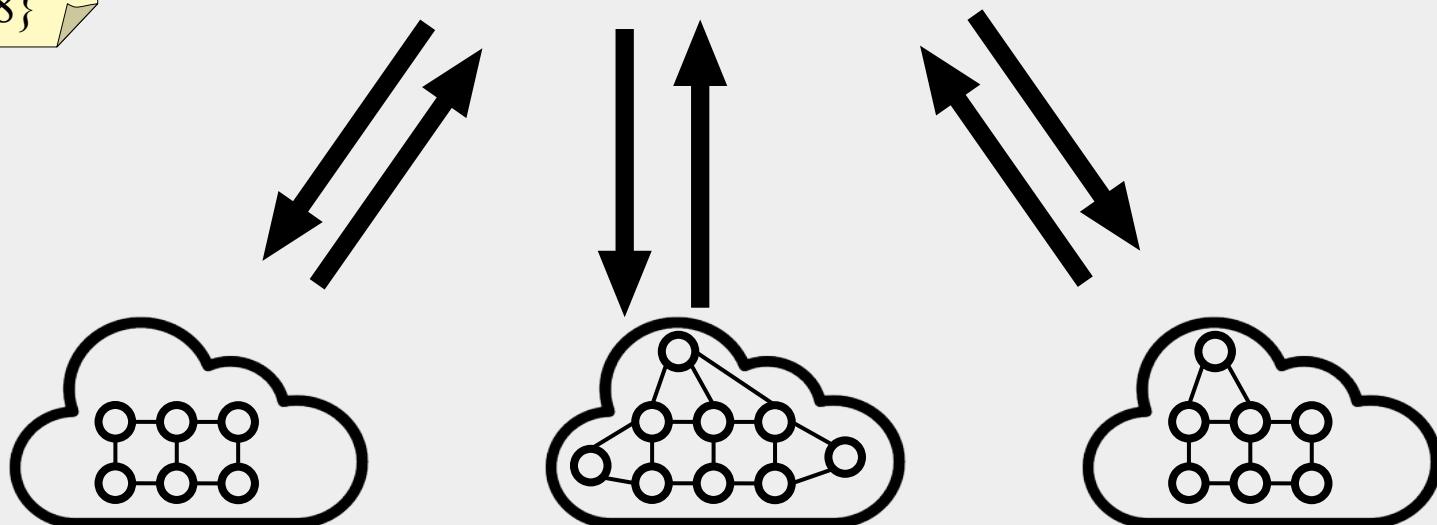
Approach-1

Frequently Push Updates

Overview

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}

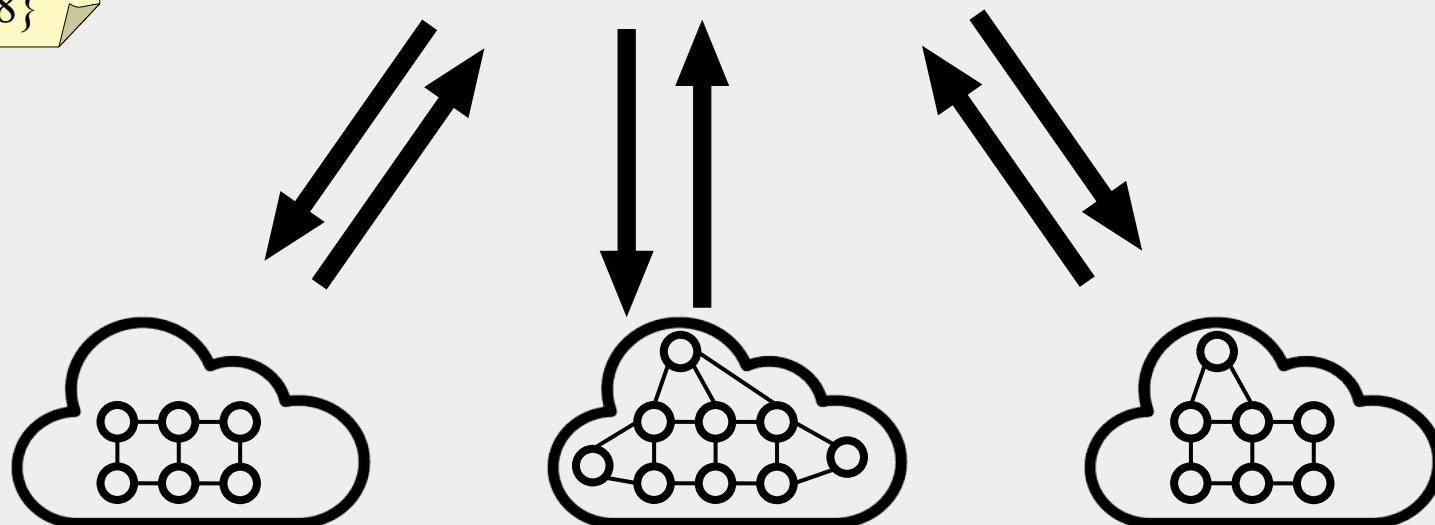


Approach-2
On-demand Pull

Challenges

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



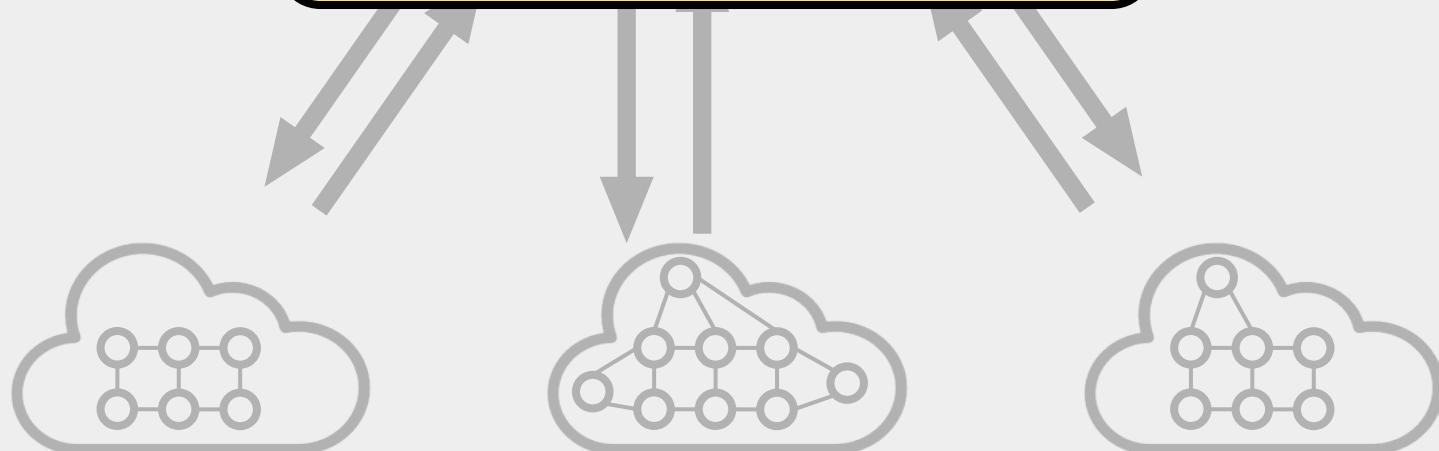
Challenges

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}

Push-based

Information freshness vs. scalability



Challenges

Q

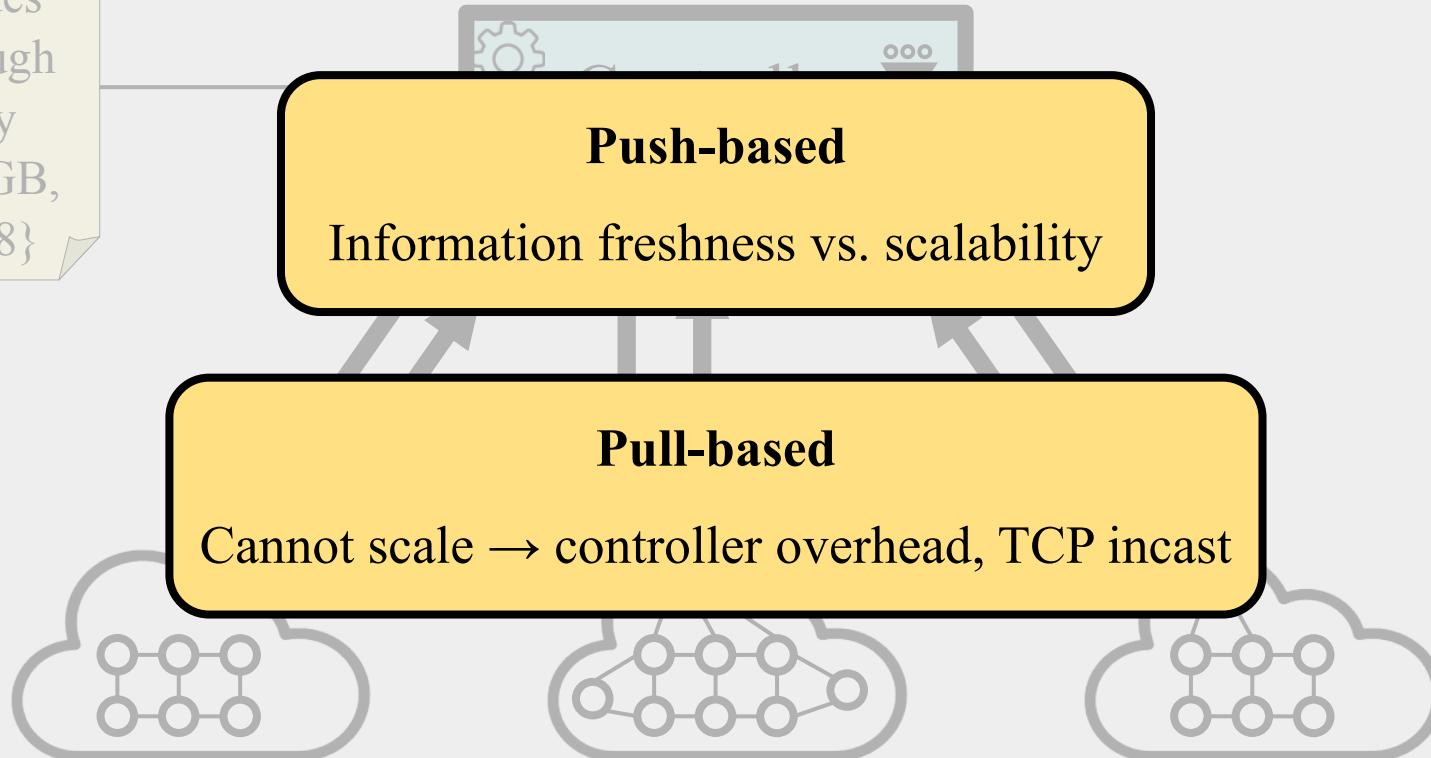
Find nodes
with enough
capacity
 $\{RAM:4GB,$
 $vCPUs: 8\}$

Push-based

Information freshness vs. scalability

Pull-based

Cannot scale → controller overhead, TCP incast



Challenges

Q

Find nodes
with

Existing systems (e.g., OpenStack) suffer from scalability limitations due to frequent push updates → cannot scale well beyond a few 100s of nodes

{Ra
vC



Challenges

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Existing solutions* throw more
resources at the problem



*OpenStack, Google Borg, Kafka, etc

Challenges

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Existing solutions* throw more
resources at the problem

!

We need a more efficient solution

FOCUS



Do we need to send a query to all nodes?

FOCUS

?

Do we need to send a query to all nodes?

!

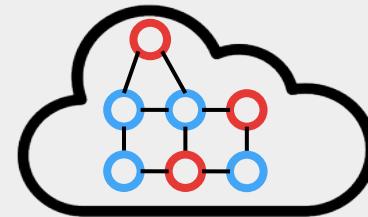
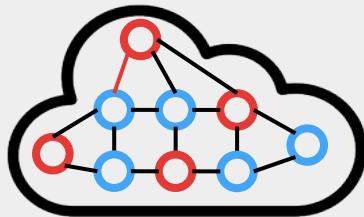
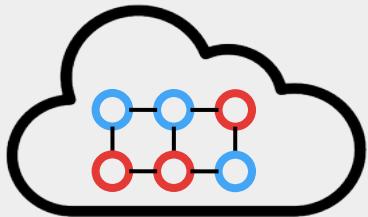
What if we could send the query
to nodes who have more potential?

FOCUS

1

Divide end nodes into *gossip* p2p groups based on their attribute values

FOCUS



Group-1: RAM (2-4]

Group-2: RAM (4-6]

FOCUS

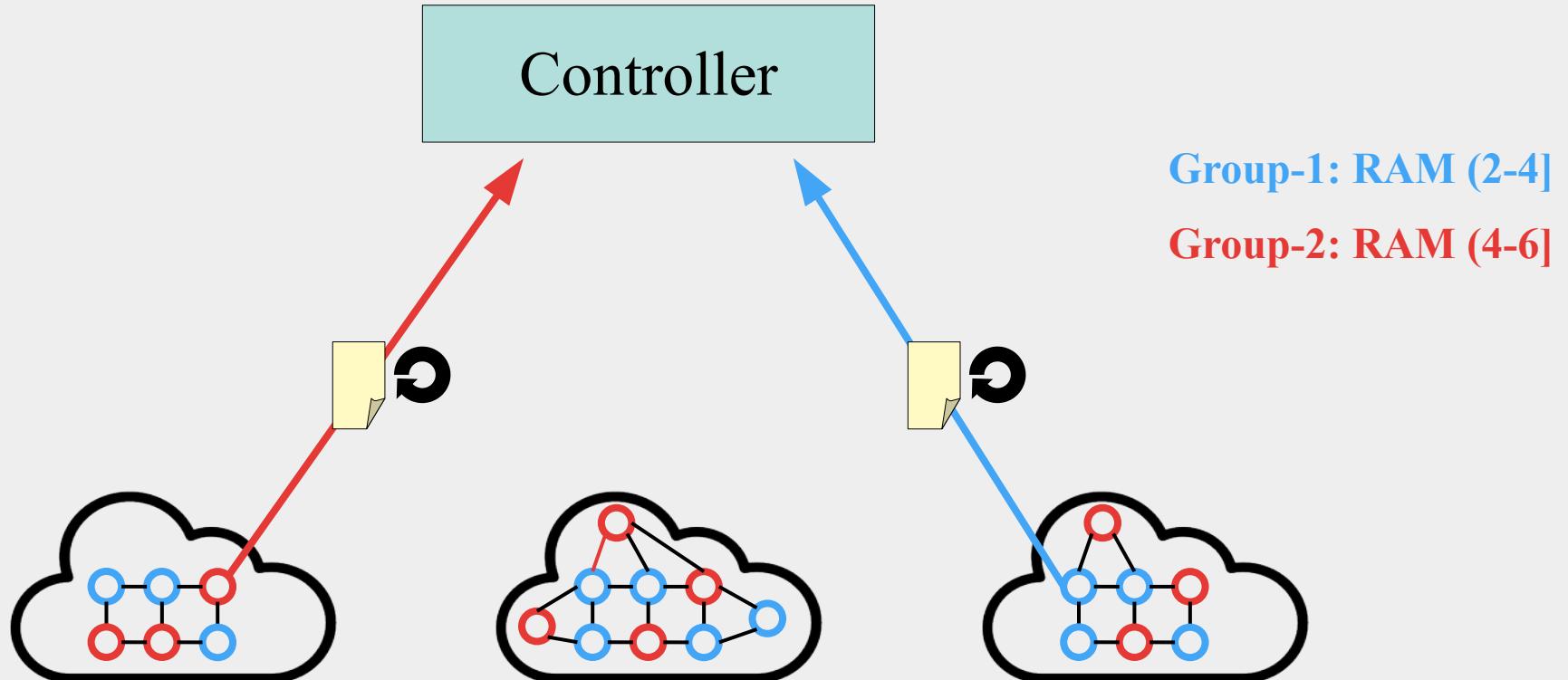
1

Divide end nodes into *gossip* p2p groups based on their attribute values

2

Push only meta-data about members of each attribute-based group

FOCUS



FOCUS

1

Divide end nodes into *gossip* p2p groups based on their attribute values

2

Push only meta-data about members of each attribute-based group

3

Route queries to corresponding groups to get real-time information

FOCUS

Q

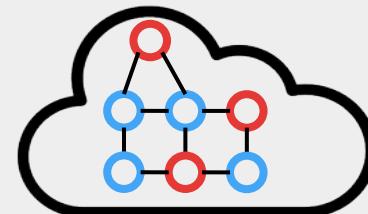
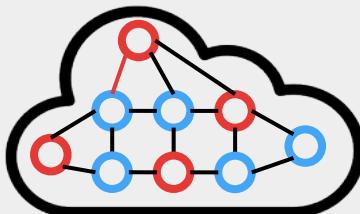
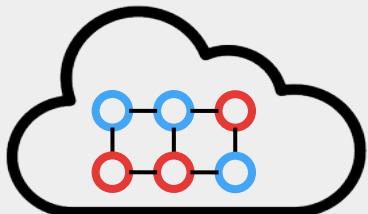
Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



Controller

Group-1: RAM (2-4]

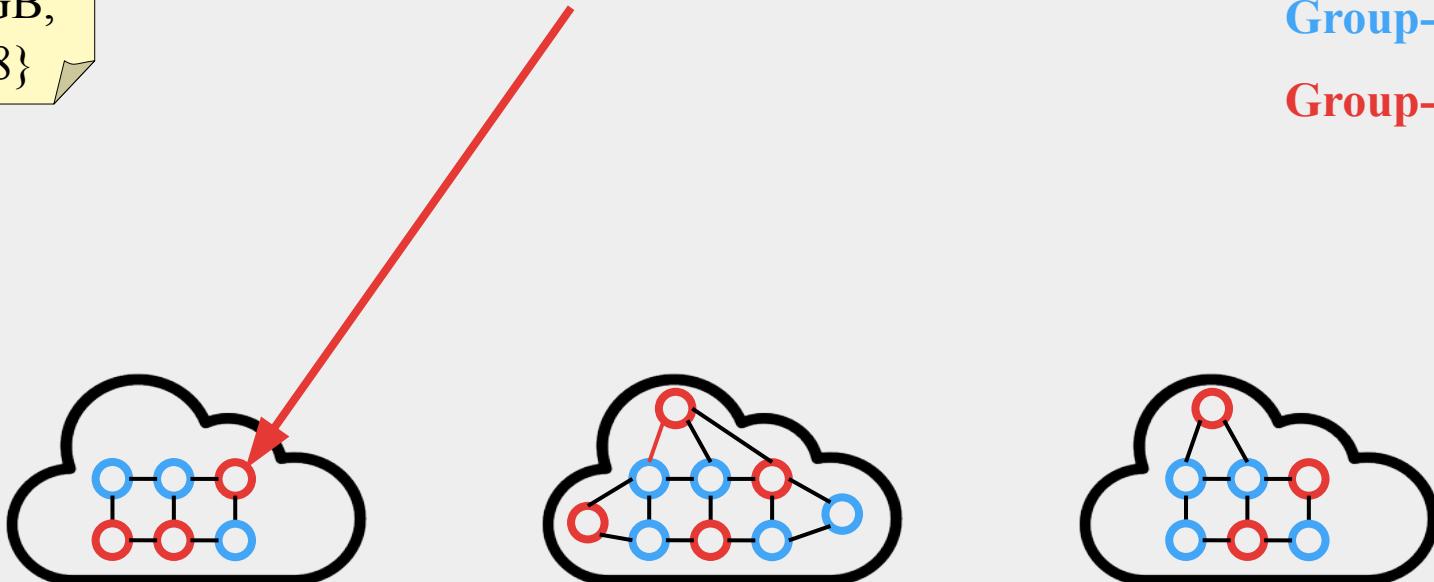
Group-2: RAM (4-6]



FOCUS

Q

Find nodes
with enough
capacity
{RAM:4GB,
vCPUs: 8}



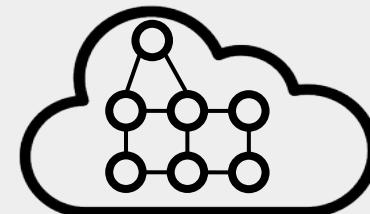
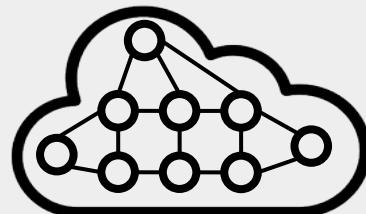
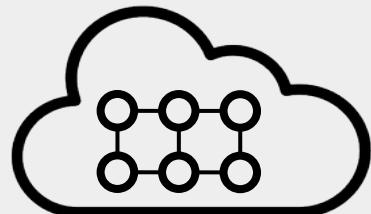


P2P Group Management

FOCUS

P2P Group Management

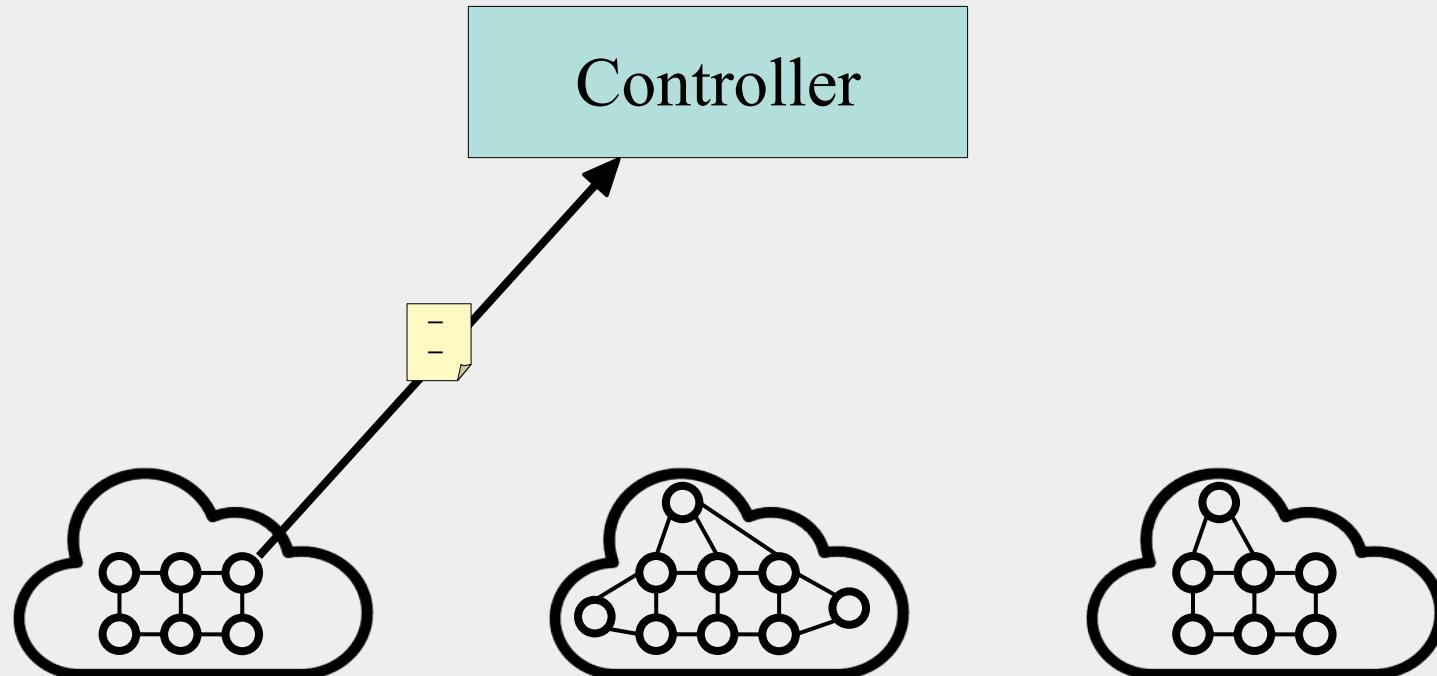
Controller



FOCUS

P2P Group Management

Bootstrapping



FOCUS

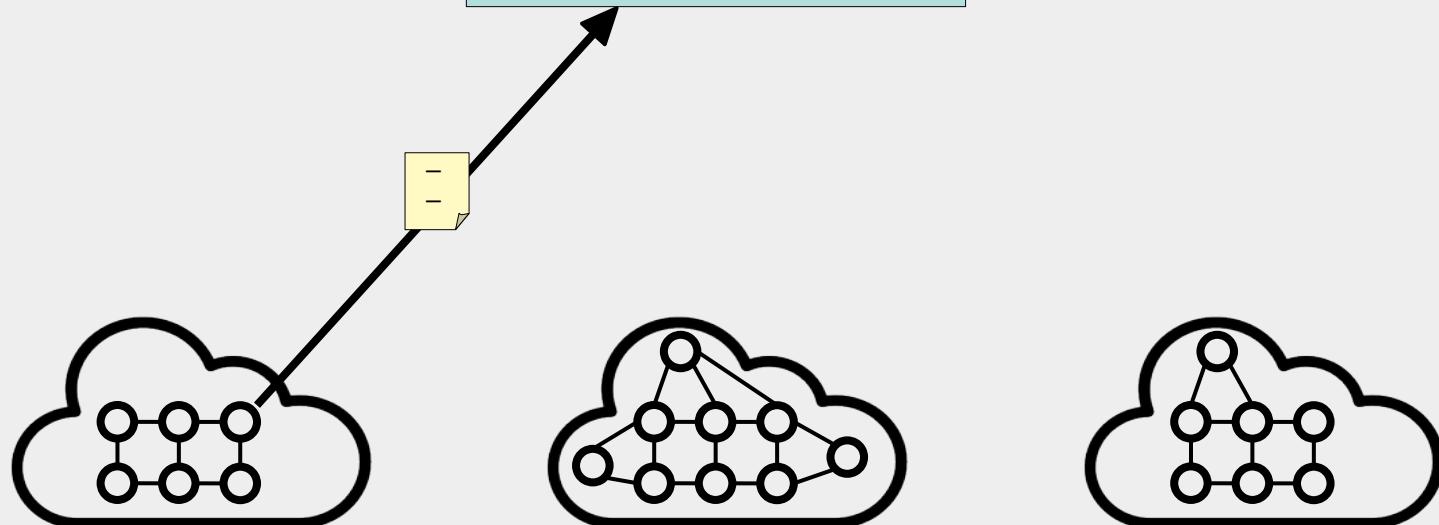
P2P Group Management

Bootstrapping

1

Node registers with controller
and attaches its current attributes

Controller



FOCUS

P2P Group Management

Bootstrapping

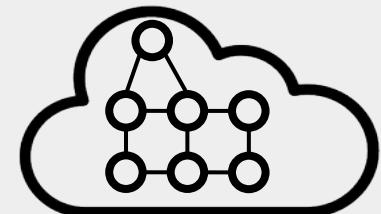
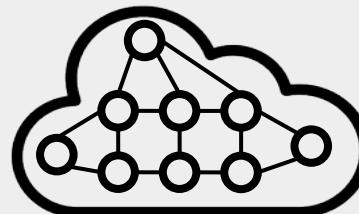
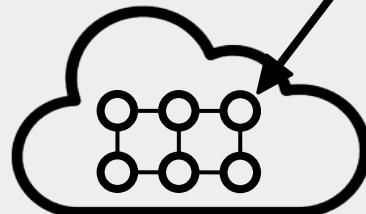
1

Node registers with controller
and attaches its current attributes

2

Controller tells node to create
new group and become entry-point

Controller



FOCUS

P2P Group Management

Bootstrapping

1

Node registers with controller
and attaches its current attributes

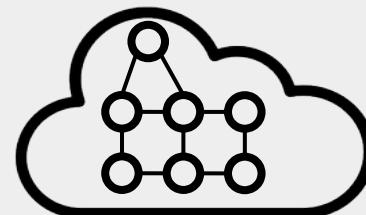
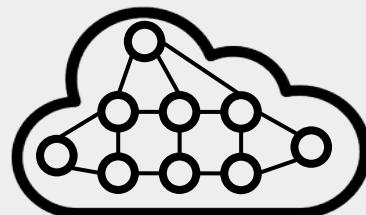
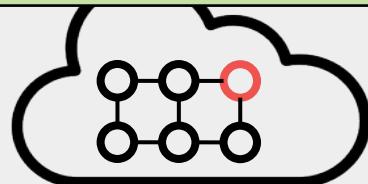
2

Controller tells node to create
new group and become entry-point

3

Node starts new gossip p2p group
and waits for future nodes to join

Controller

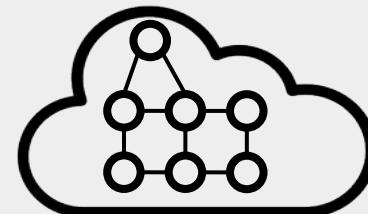
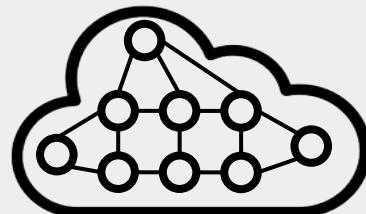
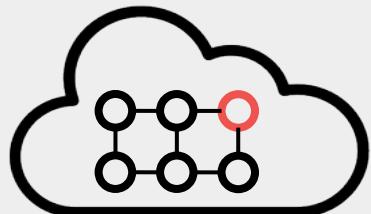


FOCUS

P2P Group Management

Joining Groups

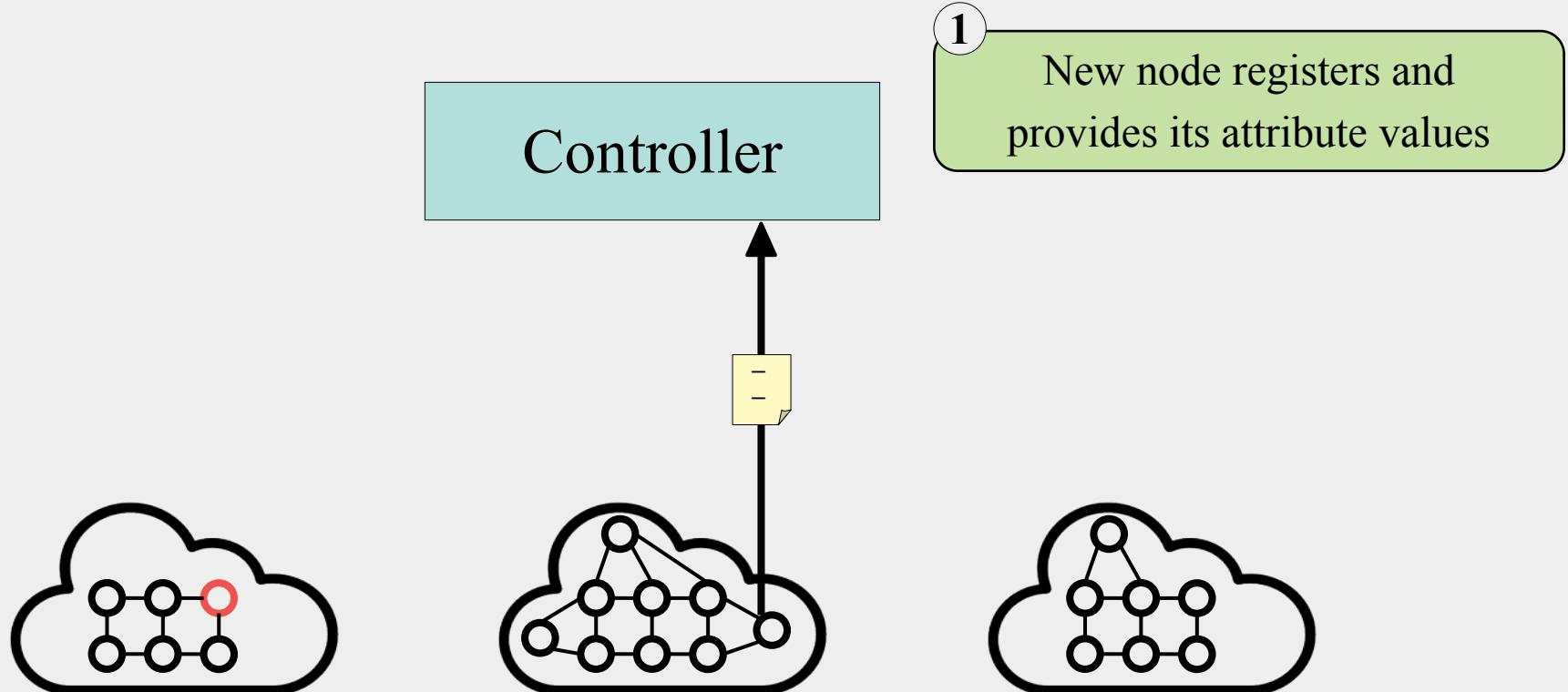
Controller



FOCUS

P2P Group Management

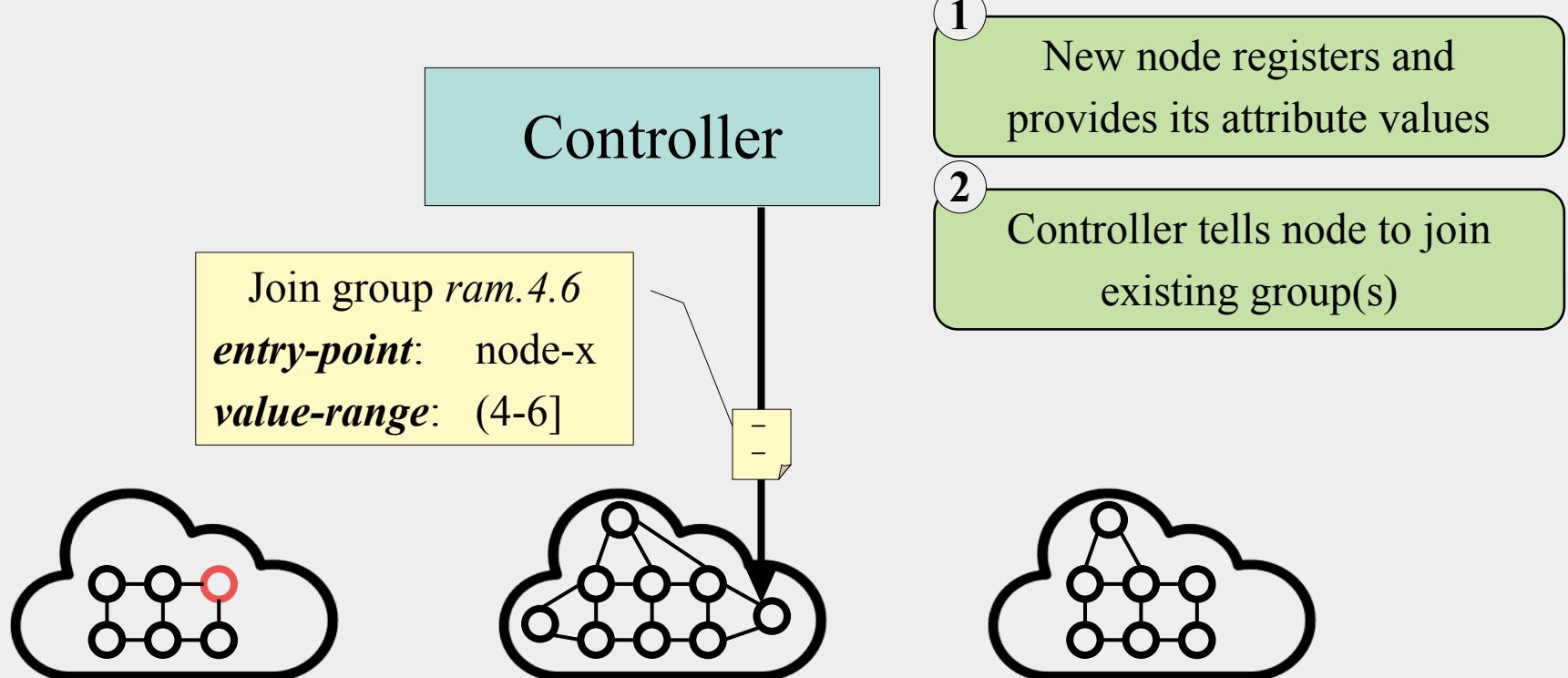
Joining Groups



FOCUS

P2P Group Management

Joining Groups

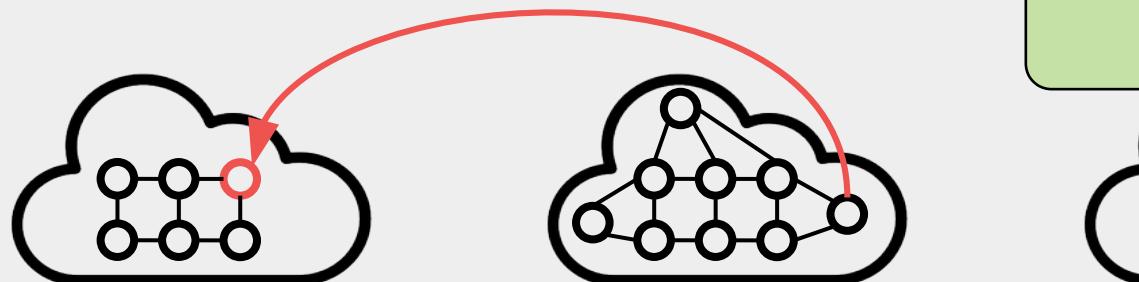


FOCUS

P2P Group Management

Joining Groups

Controller



1

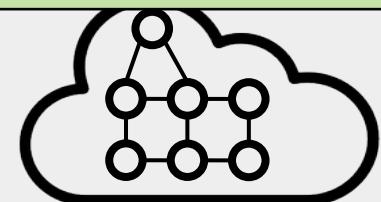
New node registers and provides its attribute values

2

Controller tells node to join existing group(s)

3

Node joins group



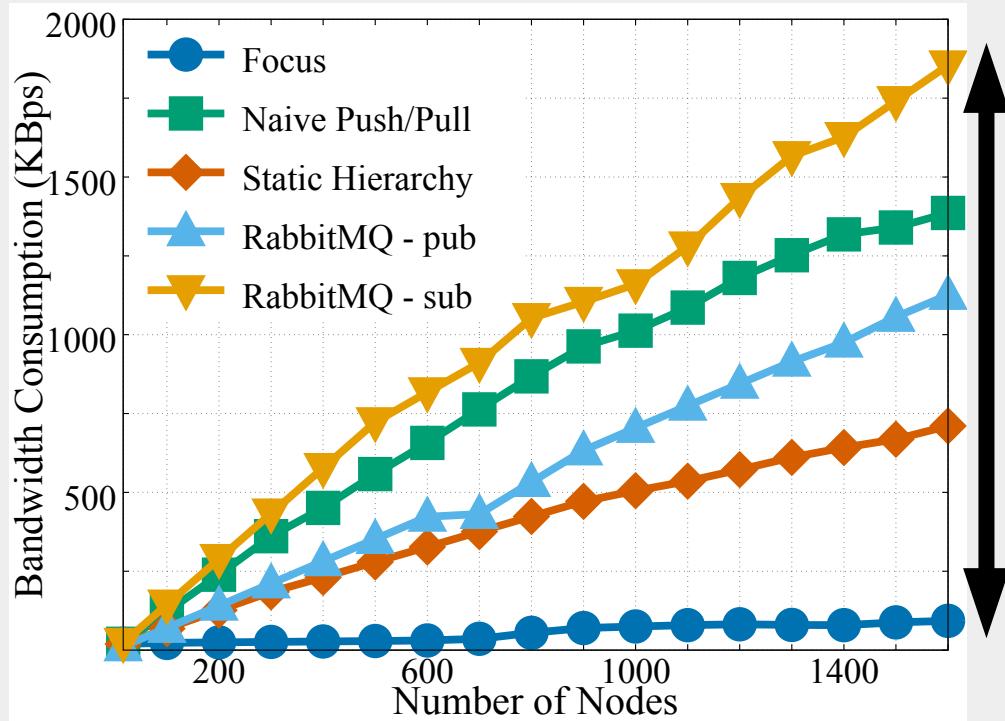
FOCUS

Implemented in *Java* with 3.3K LoC

Scales *w.r.t.*, number of *groups* instead
of number of *nodes*

Easily integrated with existing
systems  openstack™

FOCUS



Reduced 95% of
load on controller

FOCUS

Using FOCUS, controllers can *efficiently* aggregate highly dynamic information in real-time



Contributions

1

Proposed a novel gossip-based approach that is efficient and scalable replacing legacy solutions

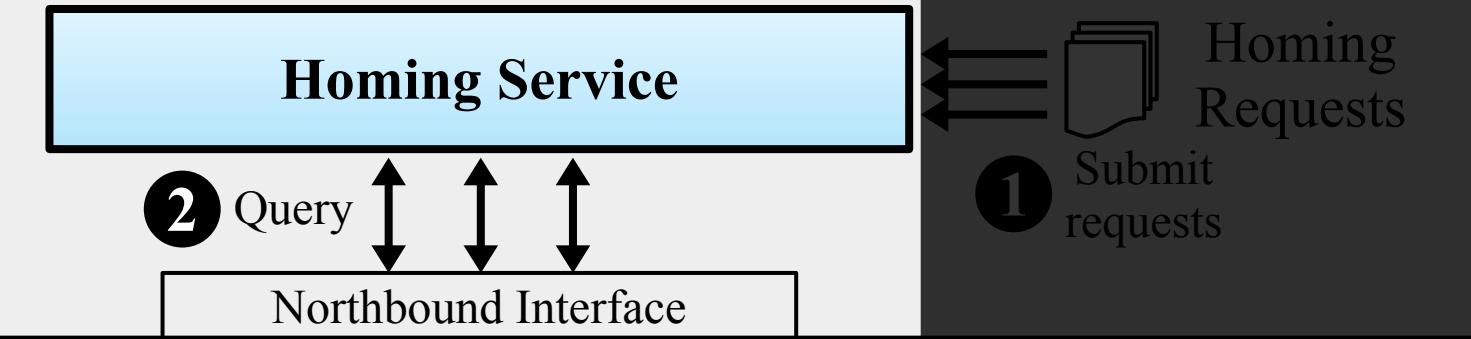
2

Designed a query interface that is easy to integrate into existing systems and supports complex queries

3

Implemented and evaluated FOCUS in a large-scale geo-distributed setting and demonstrated its efficiency

Publications: [Usenix **HotCloud**'18, IEEE **ICDCS**'19]



Cloud (C) & Service (S) Controllers

Challenge-3

How can we *gain info* to make informed homing decisions without placing significant (querying) *load* on controllers?

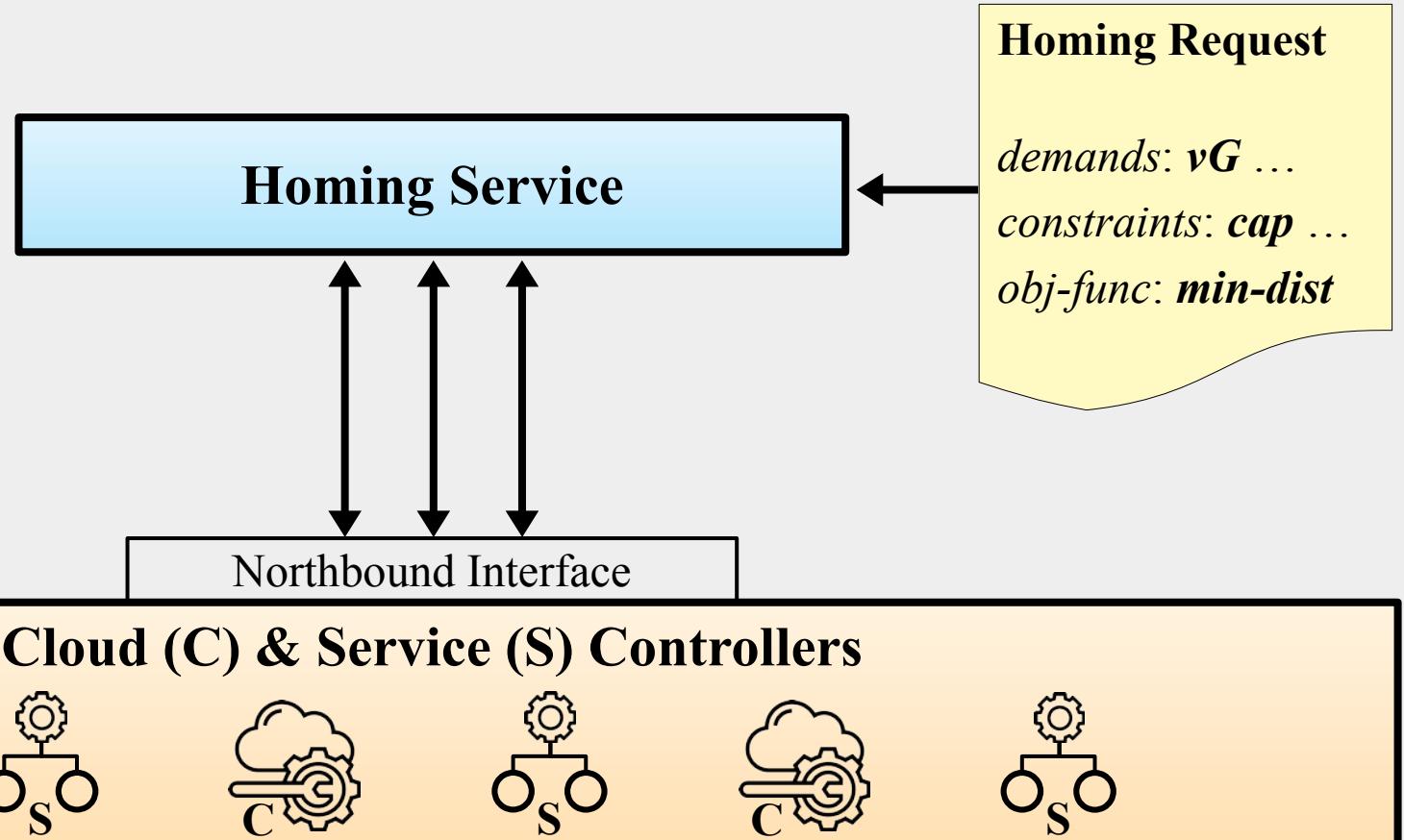
Geo-distributed Compute/Network Resources (micro datacenters)



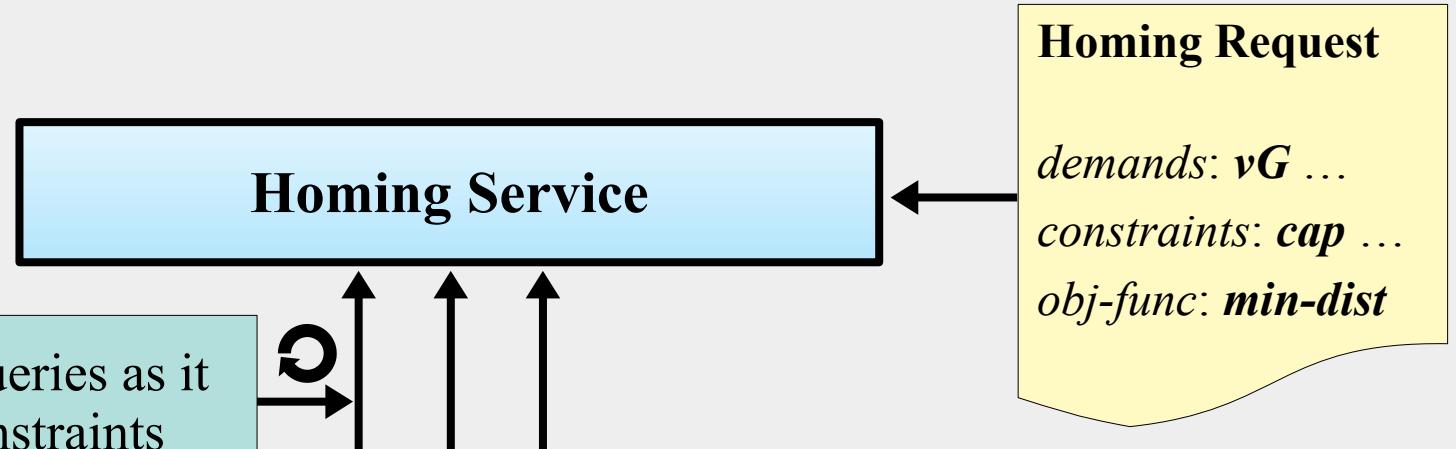


Efficient incremental approach to
homing network services

Overview



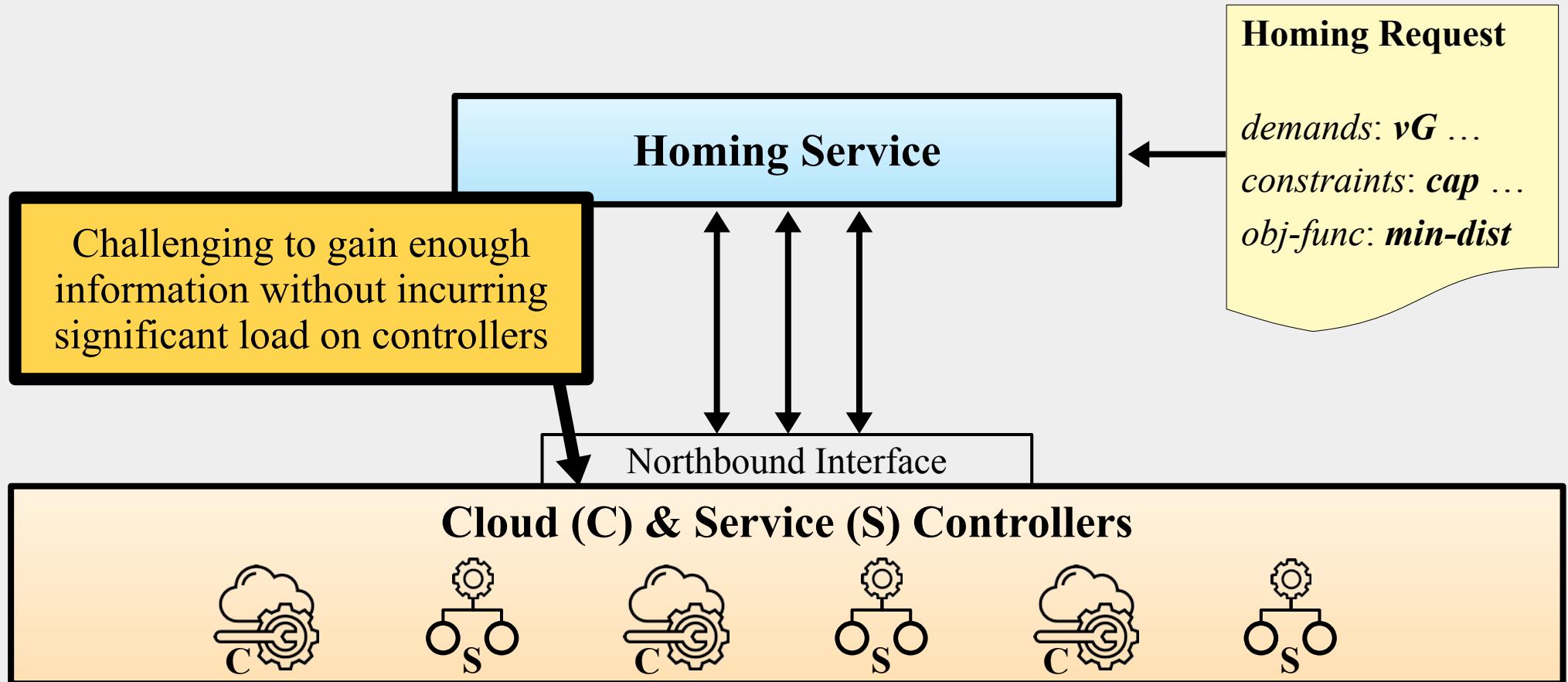
Overview



Cloud (C) & Service (S) Controllers



Challenge



Challenge



Why do we need to
reduce query load?

Challenge



Why do we need to
reduce query load?

Controllers top priority is life-cycle management of cloud
and service instances (monitoring, BGP routes, etc)

Controllers aren't designed for
large-scale query processing

Challenge



Why do we need to
reduce query load?

We analyzed 1-week logs for a homing service
running in production of a large NSP network

Challenge

Observations from analysis of a homing service running in production

1

The homing service sends 1000s
of queries for each request

Challenge

Observations from analysis of a homing service running in production

1

The homing service sends 1000s
of queries for each request



Controllers place a limit on # of
queries they allow for homing

Challenge

Observations from analysis of a homing service running in production

1

The homing service sends 1000s
of queries for each request



!

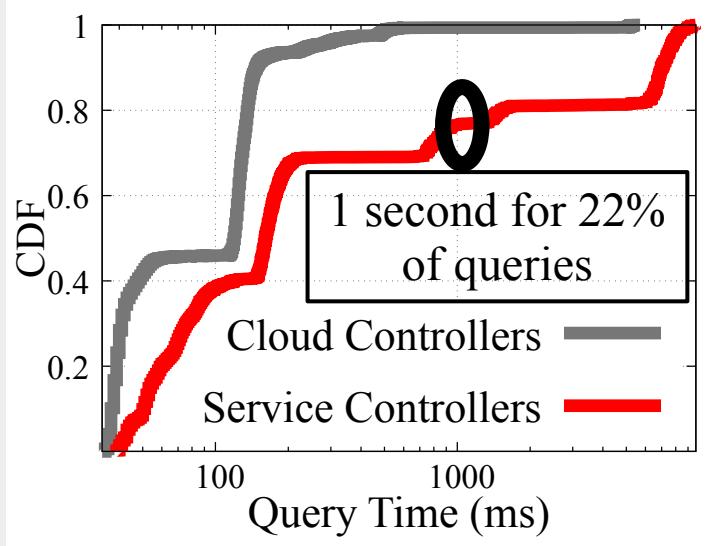
Homing service sends twice # of queries as allowed 44% of the time

!

Controllers *queue* homing queries → unwanted delays

Challenge

Observations from analysis of a homing service running in production



2

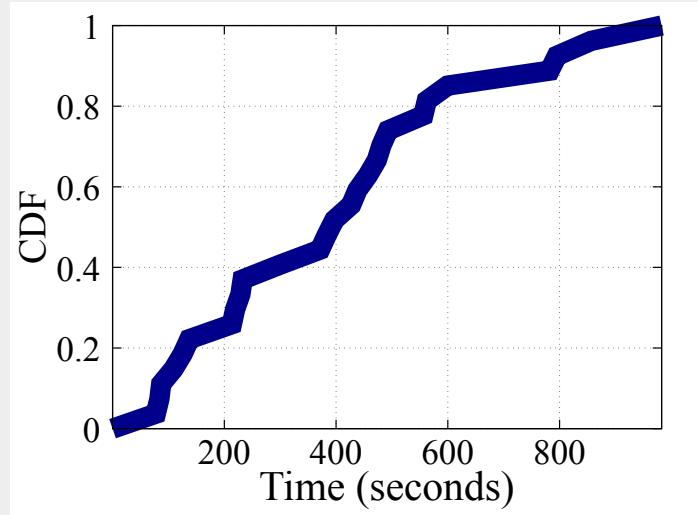
Controllers take a lot of time responding to homing queries

Challenge

Observations from analysis of a homing service running in production

3

Heavy *query processing* and *query queuing* → long time to process *each* homing request!



Challenge

Key Question

How can we reduce query load (*i.e.*, visibility) while maintaining *high-quality* homing solutions?

Challenge

Key Question

How can we reduce query load (*i.e.*, visibility) while maintaining *high-quality* homing solutions?







Goal

Reduce number of queries while
maintaining solution quality



Goal

Reduce number of queries while
maintaining solution quality

But, how?

Goal

Reduce number of queries while
maintaining solution quality

But, how?

i

Incremental Approach



?

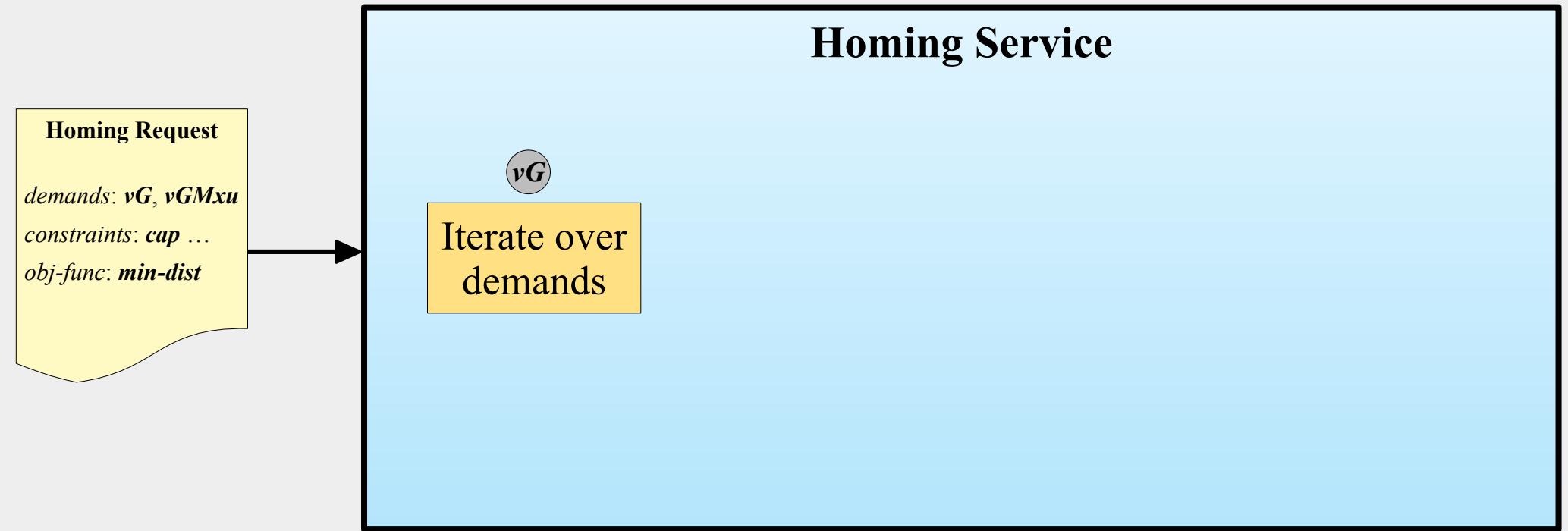
How/why the homing service
needs to send queries?

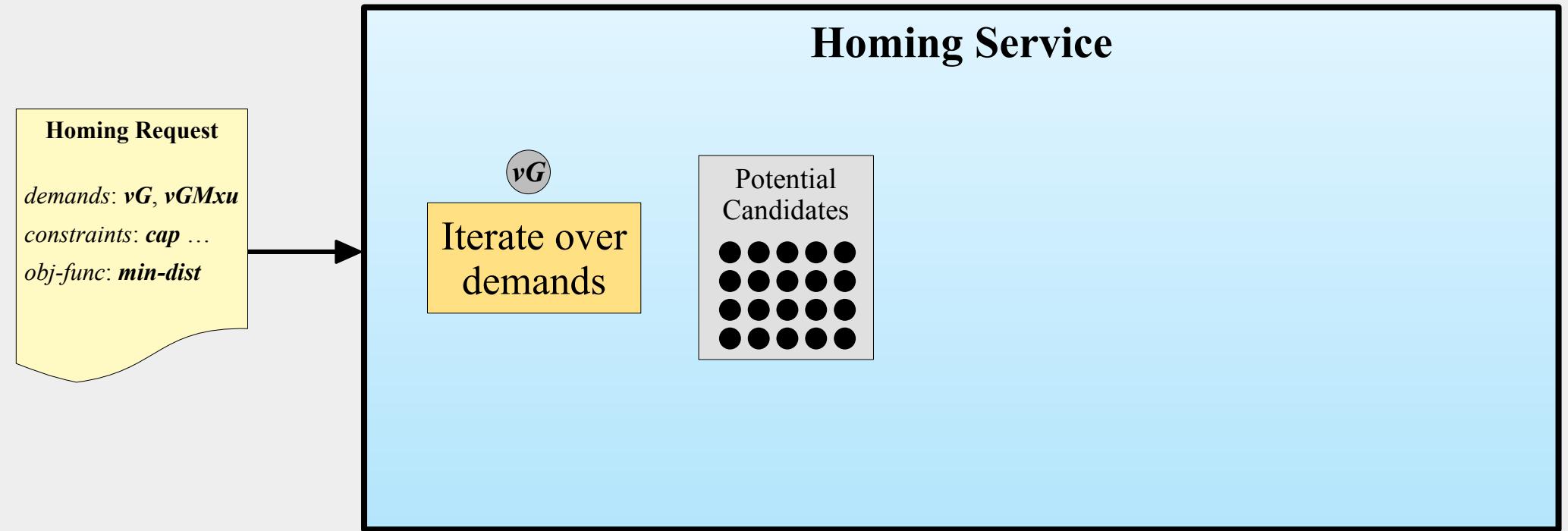
Homing Service

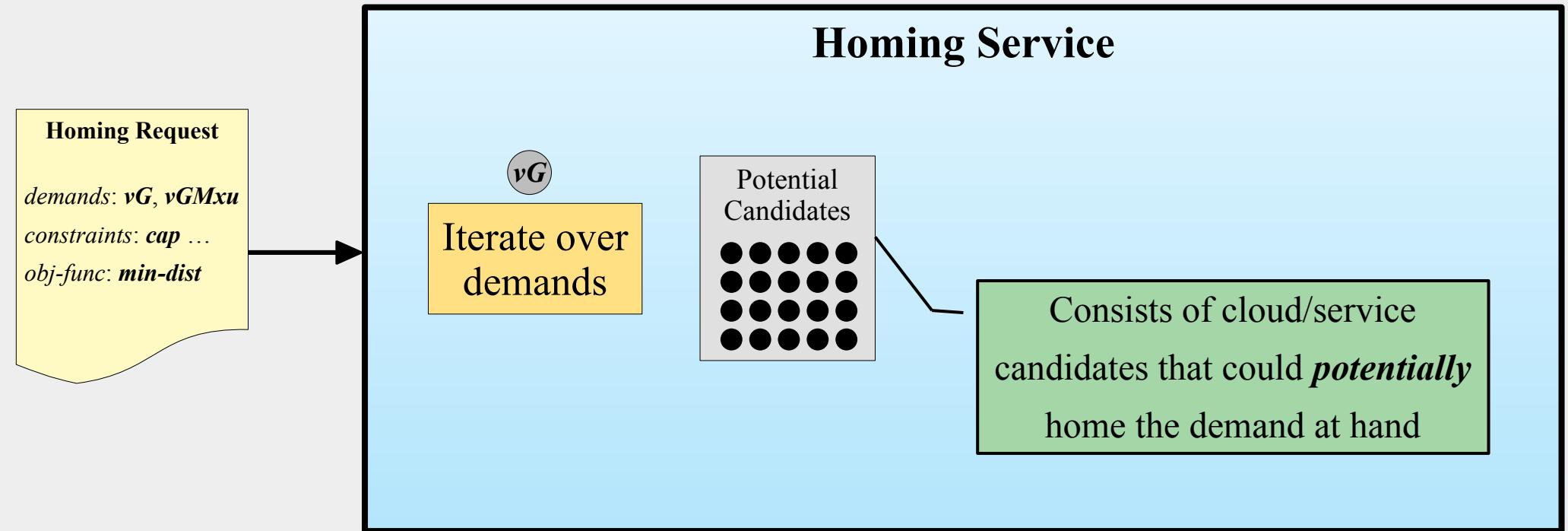
Homing Request

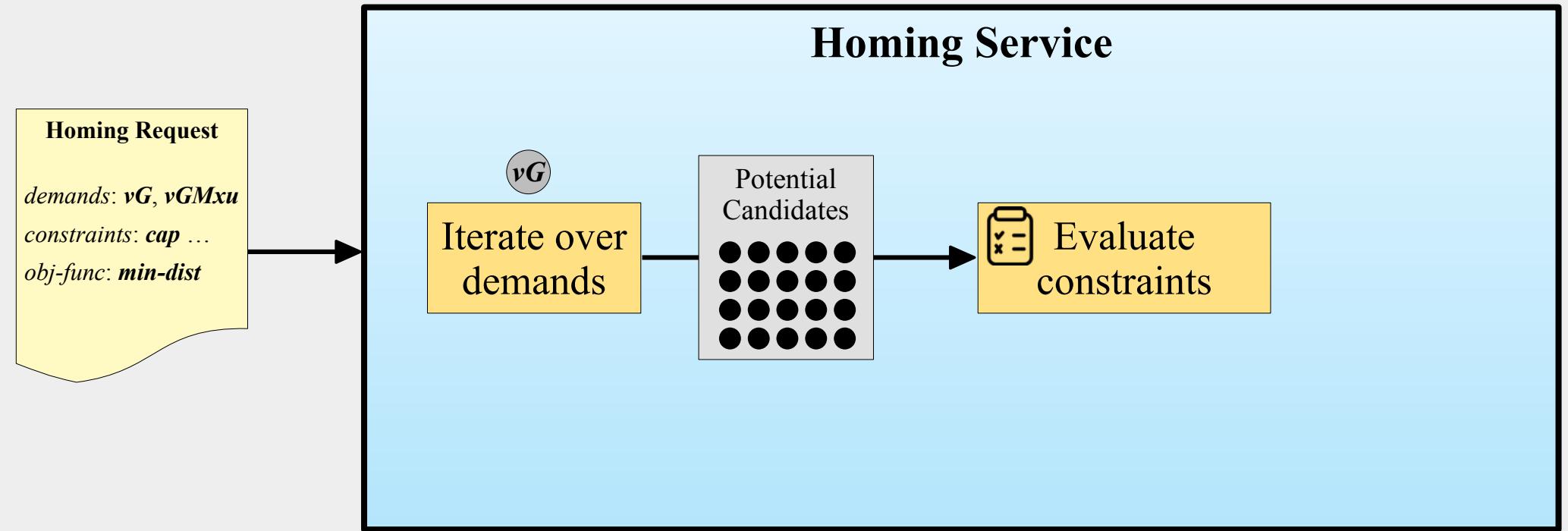
demands: $vG, vGMxu$
constraints: $cap \dots$
obj-func: $min-dist$

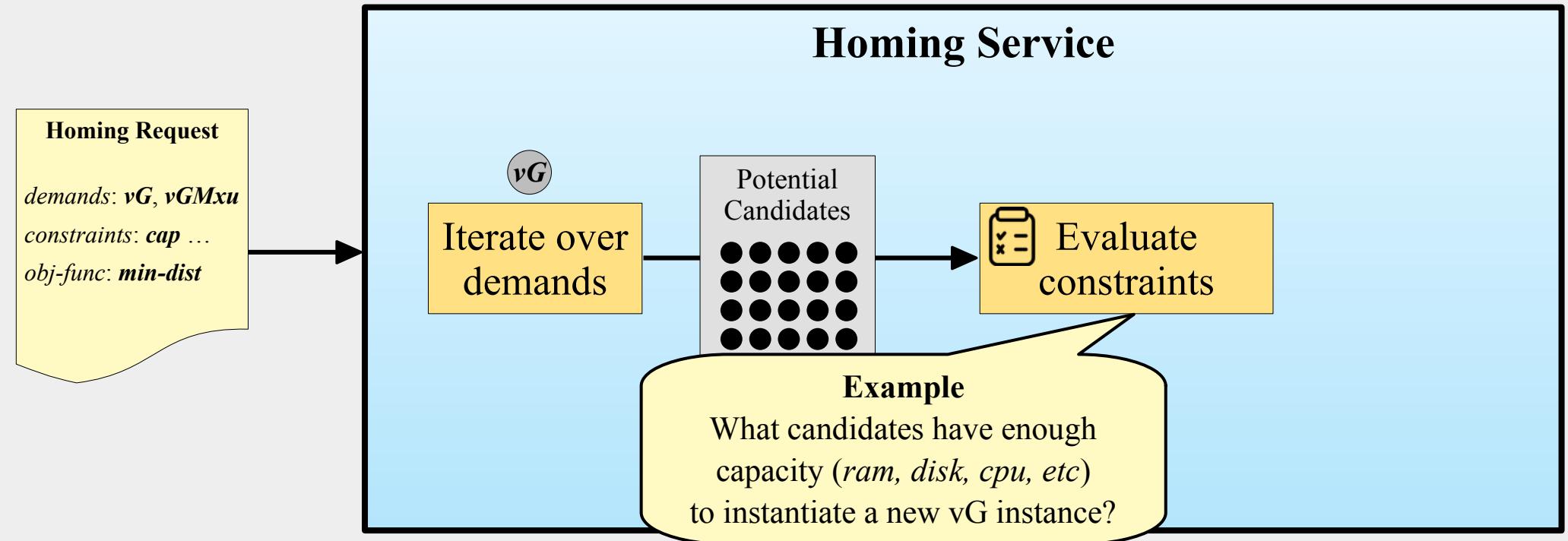


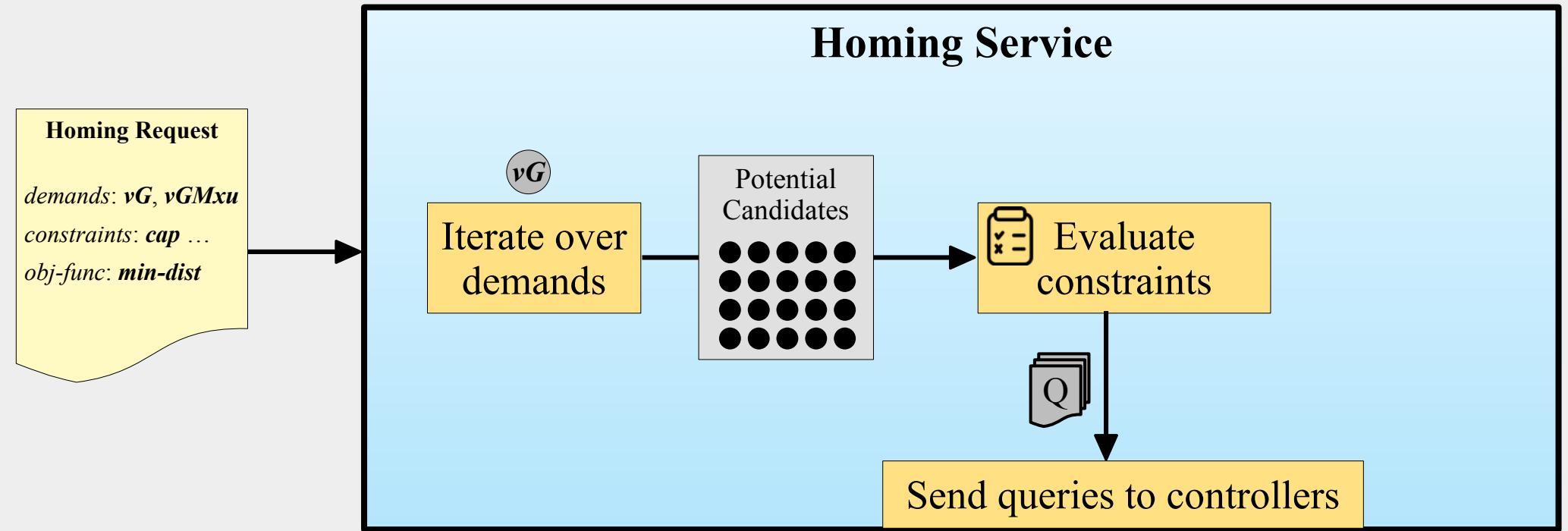


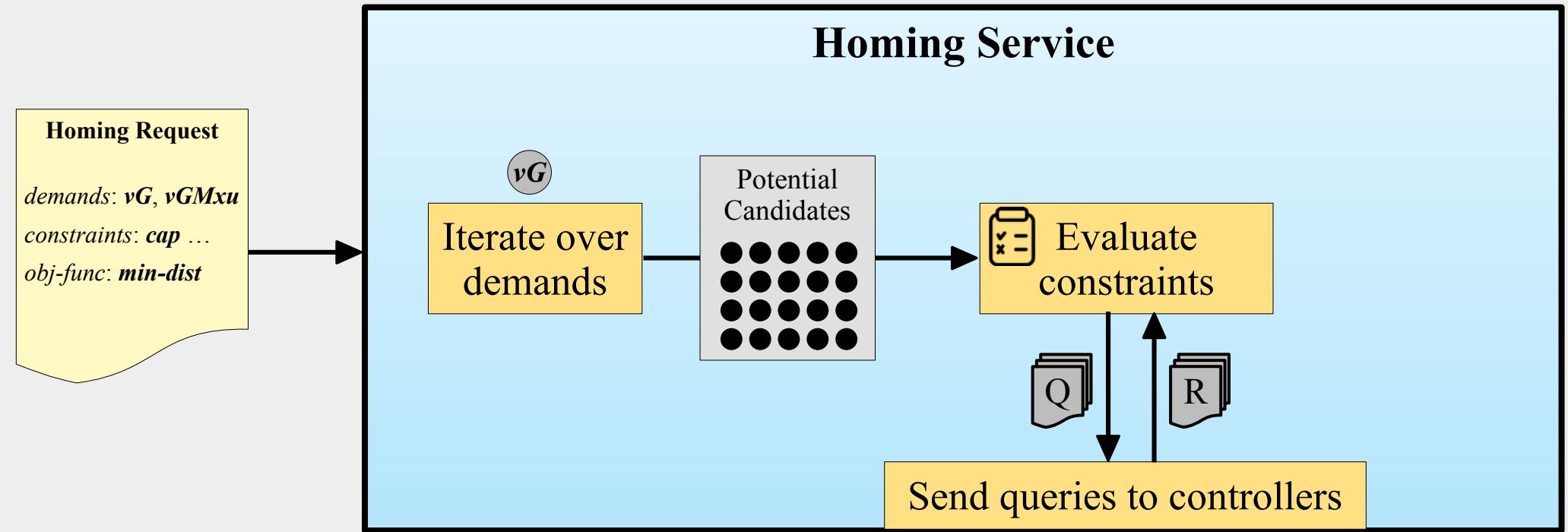


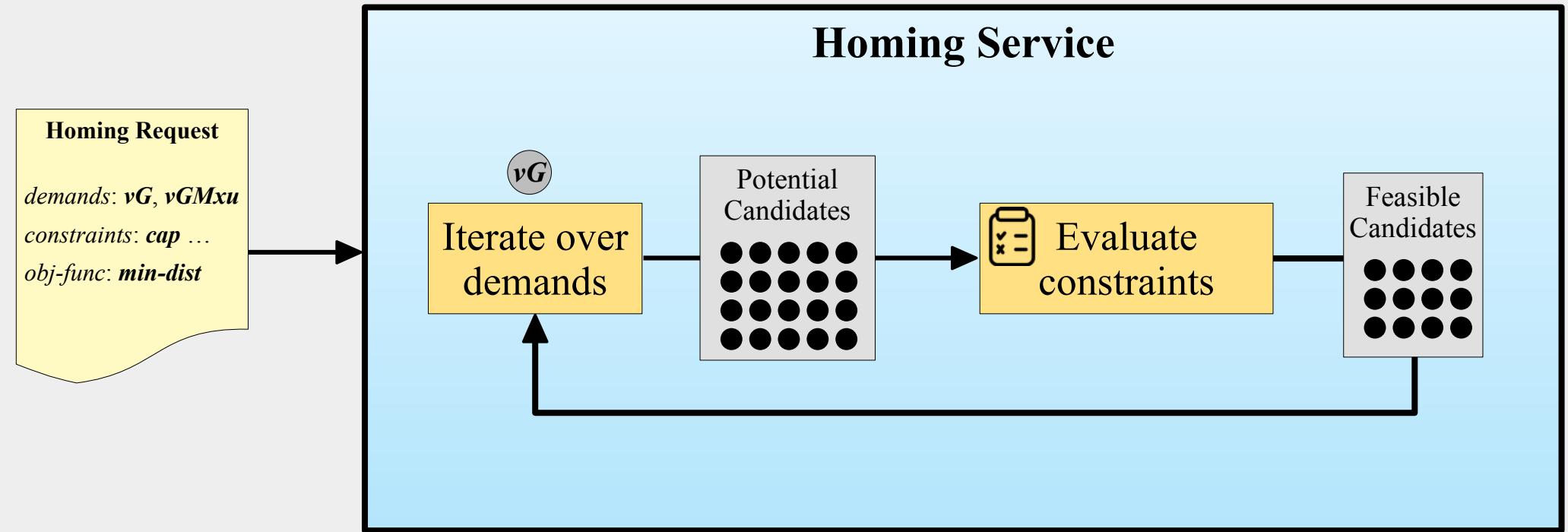


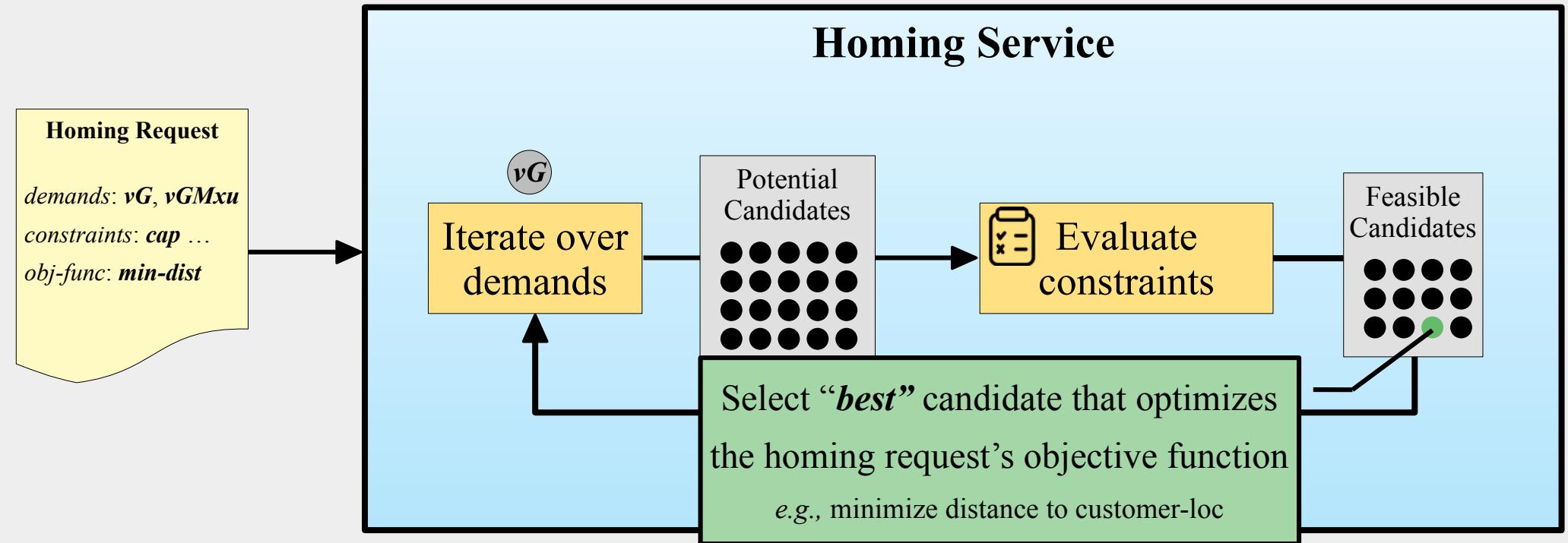


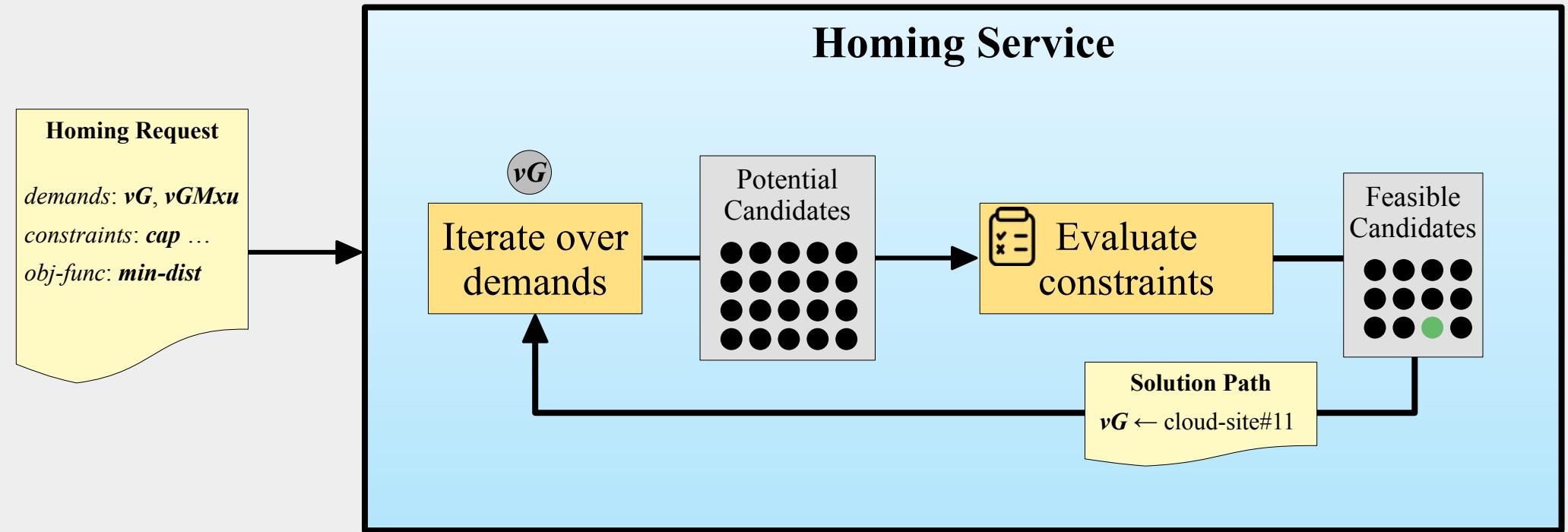


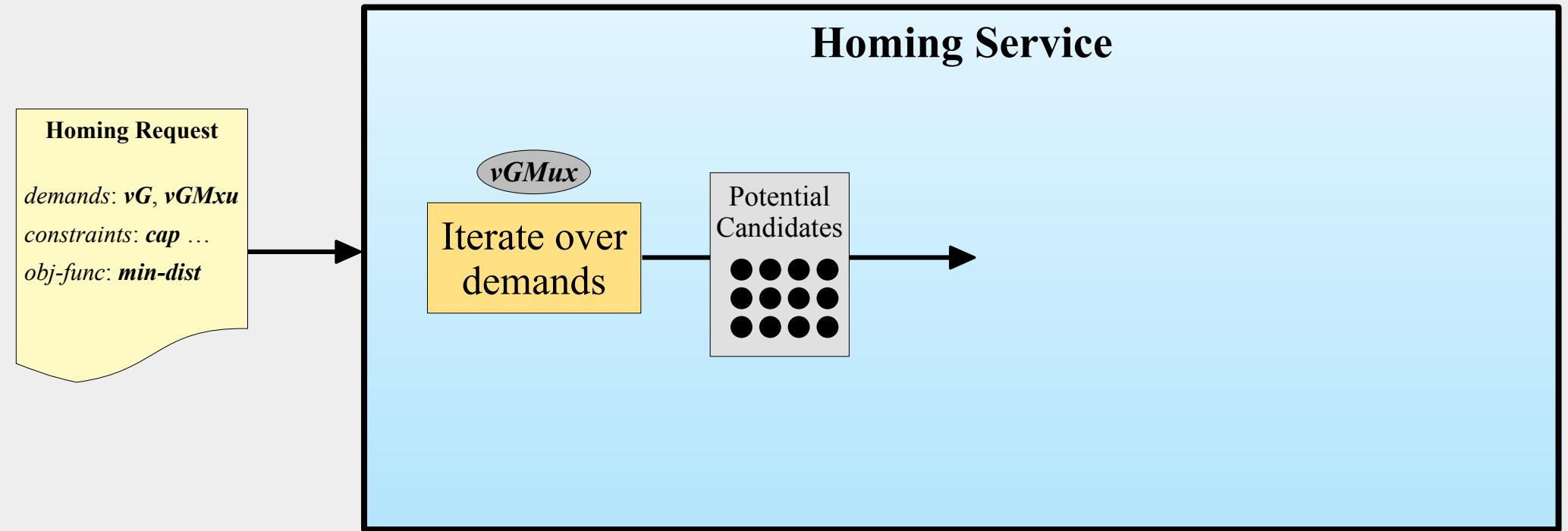


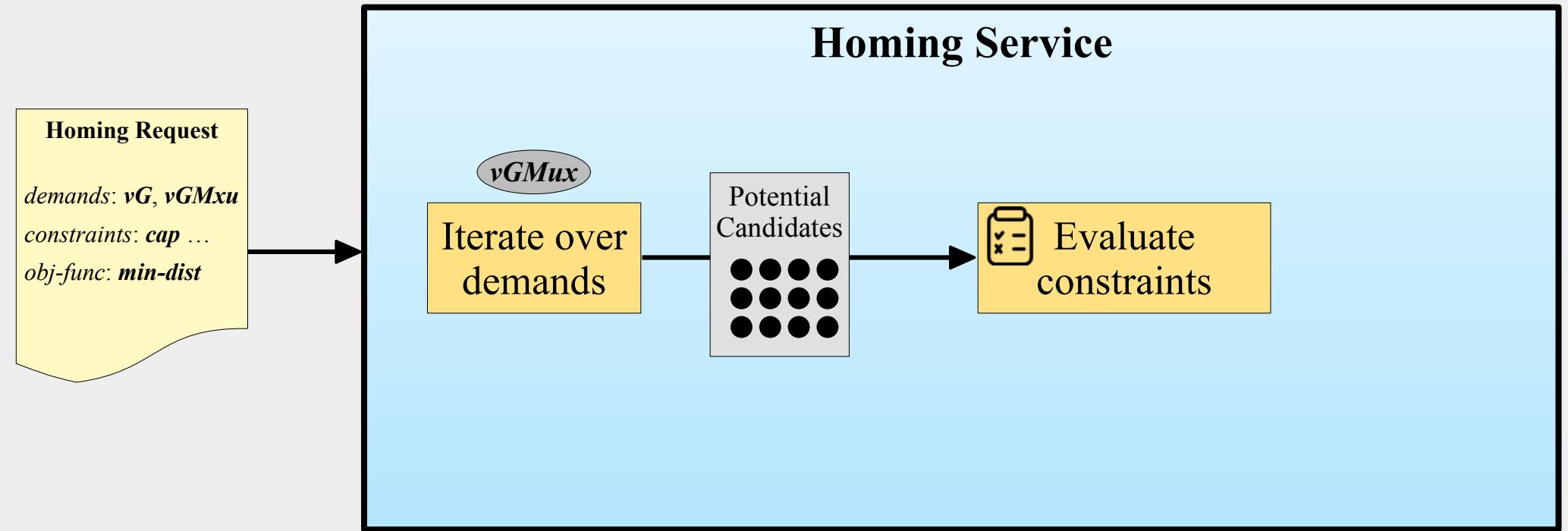


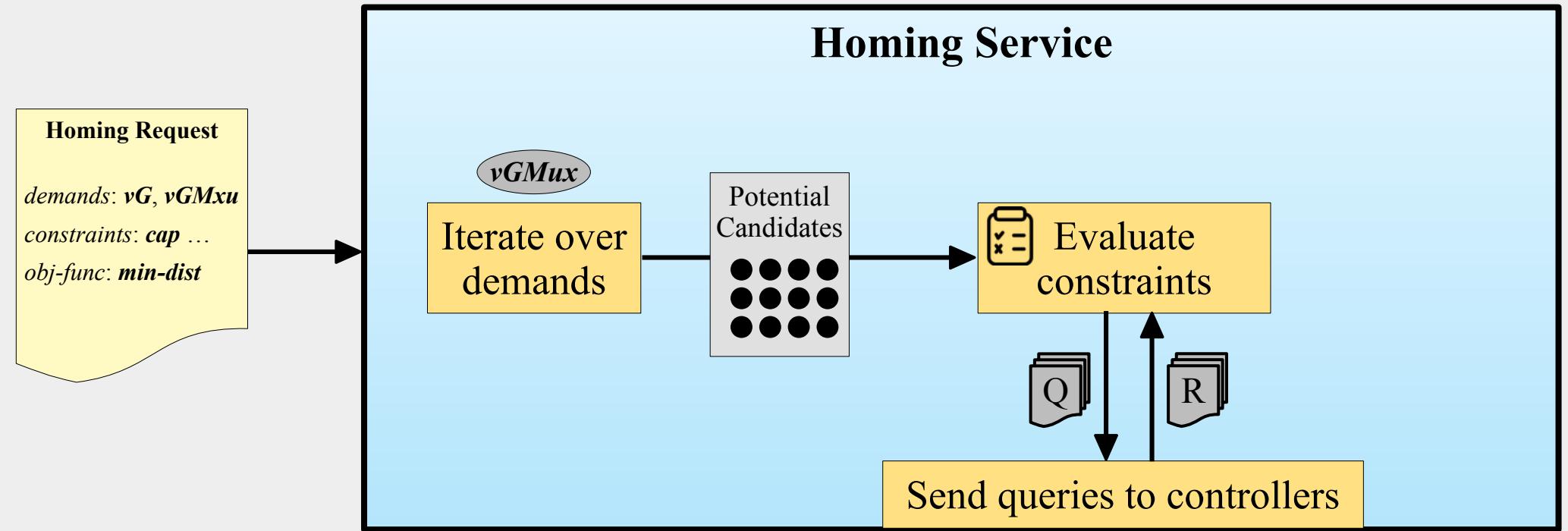


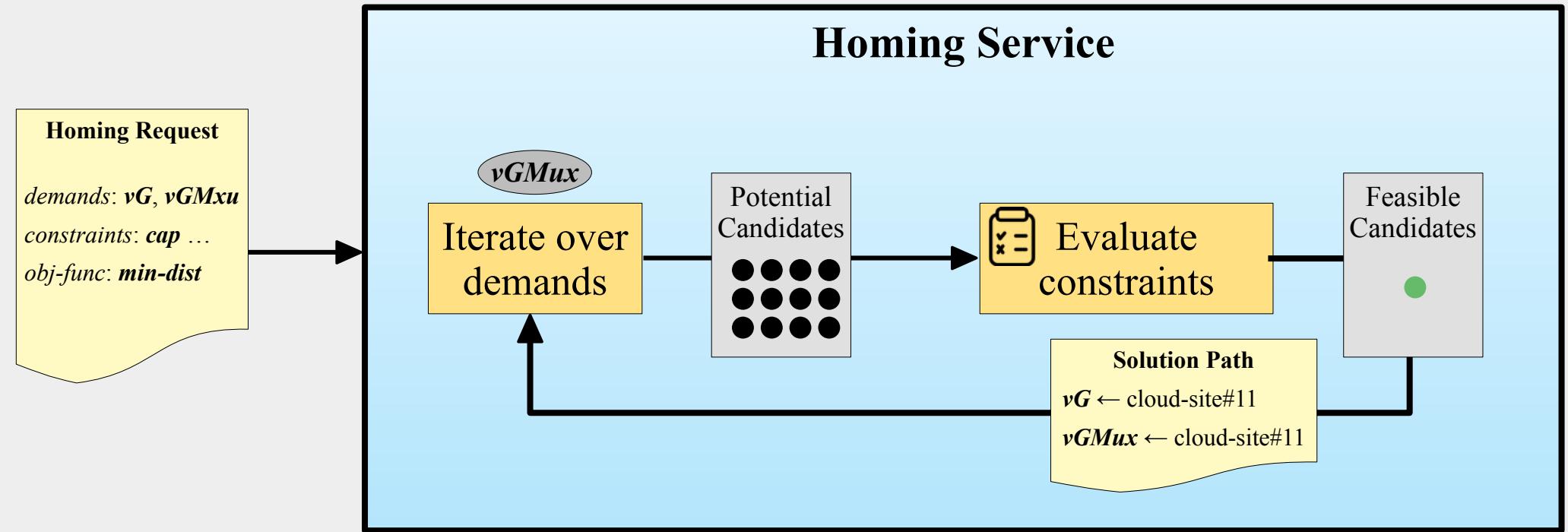












Homing Request
demands: $vG, vGMxu$
constraints: cap
obj-func: $min-dis$

Homing Service

vGMux

Potential
candidates

Key Observation

For many constraints, the number of issued queries is proportional to the number of candidates to be evaluated



Evaluate
constraints



Q



R

Send queries to controllers

Key Observation

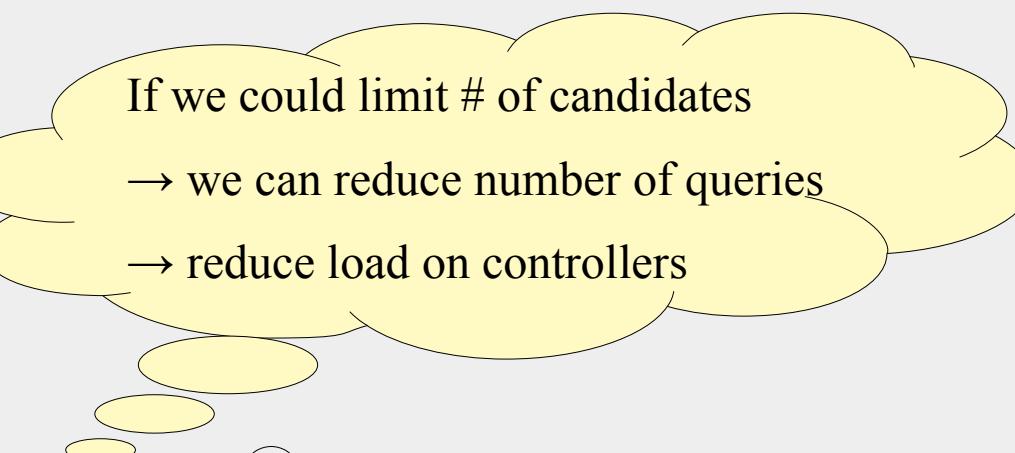
For many constraints, the number of issued queries is proportional to the number of candidates to be evaluated

If we could limit # of candidates

- we can reduce number of queries
- reduce load on controllers

Key Observation

For many constraints, the number of issued queries is proportional to the number of candidates to be evaluated

A thought bubble composed of several overlapping yellow rounded rectangles of varying sizes, arranged to resemble a thought or speech bubble.

If we could limit # of candidates
→ we can reduce number of queries
→ reduce load on controllers



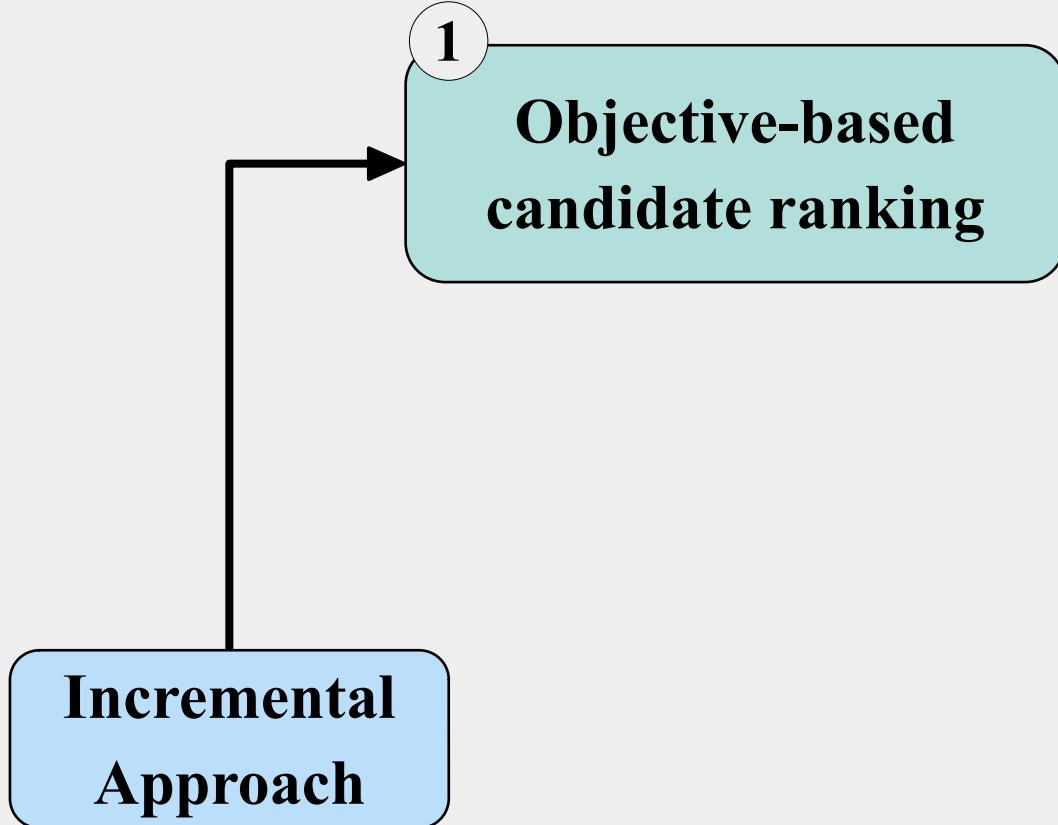
But, which candidates should we include?

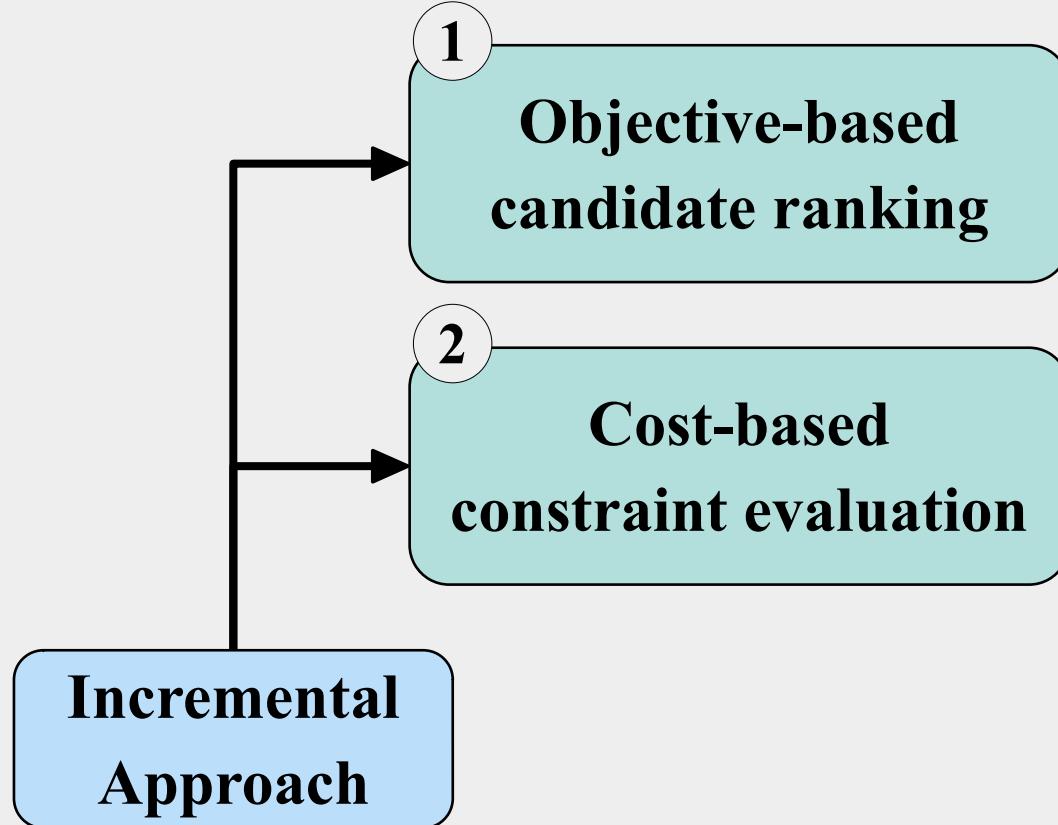
Leverage the **objective function** of each homing request to know which candidates are “*good*”

Incremental Approach



But, which candidates should we include?







1

Objective-based candidate ranking

a

Compute objective
value for all potential
candidates

1

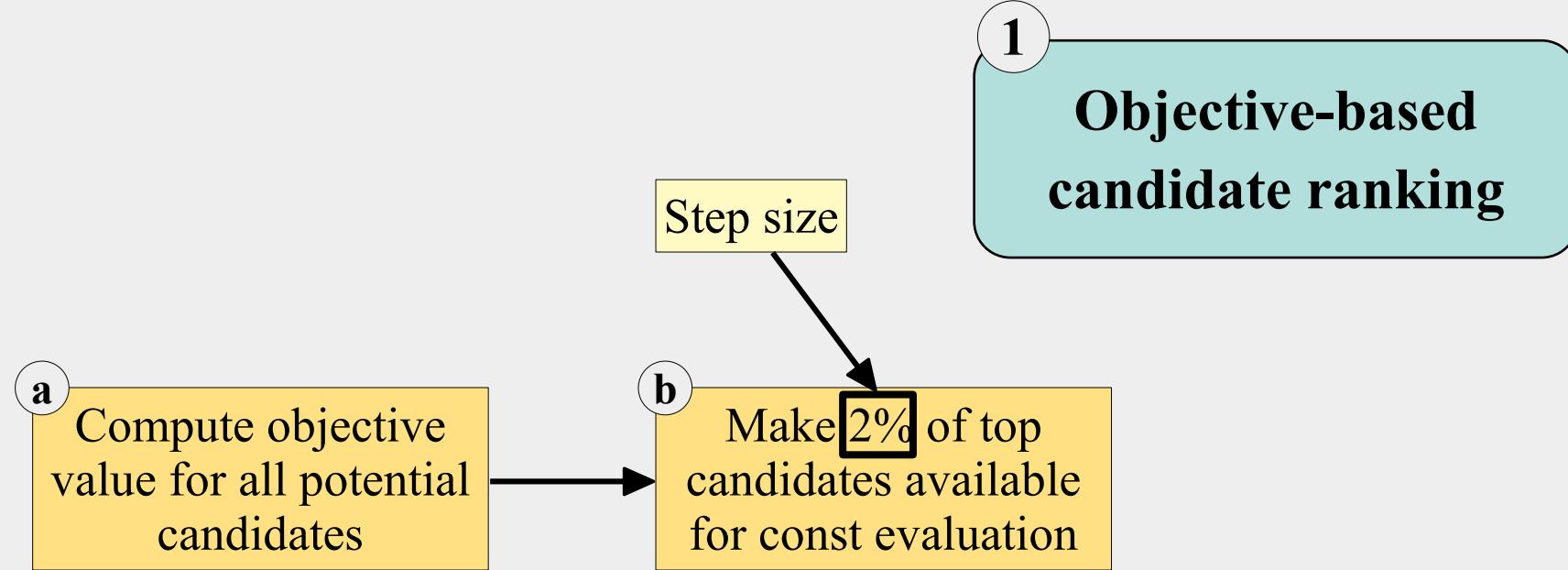
Objective-based candidate ranking

a

Compute objective
value for all potential
candidates

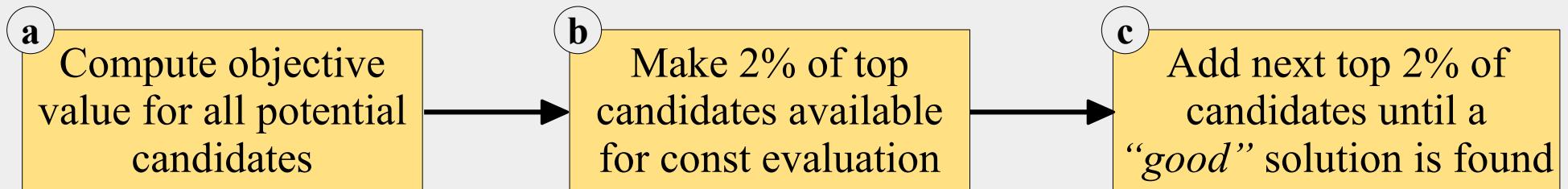
b

Make 2% of top
candidates available
for const evaluation



1

Objective-based candidate ranking



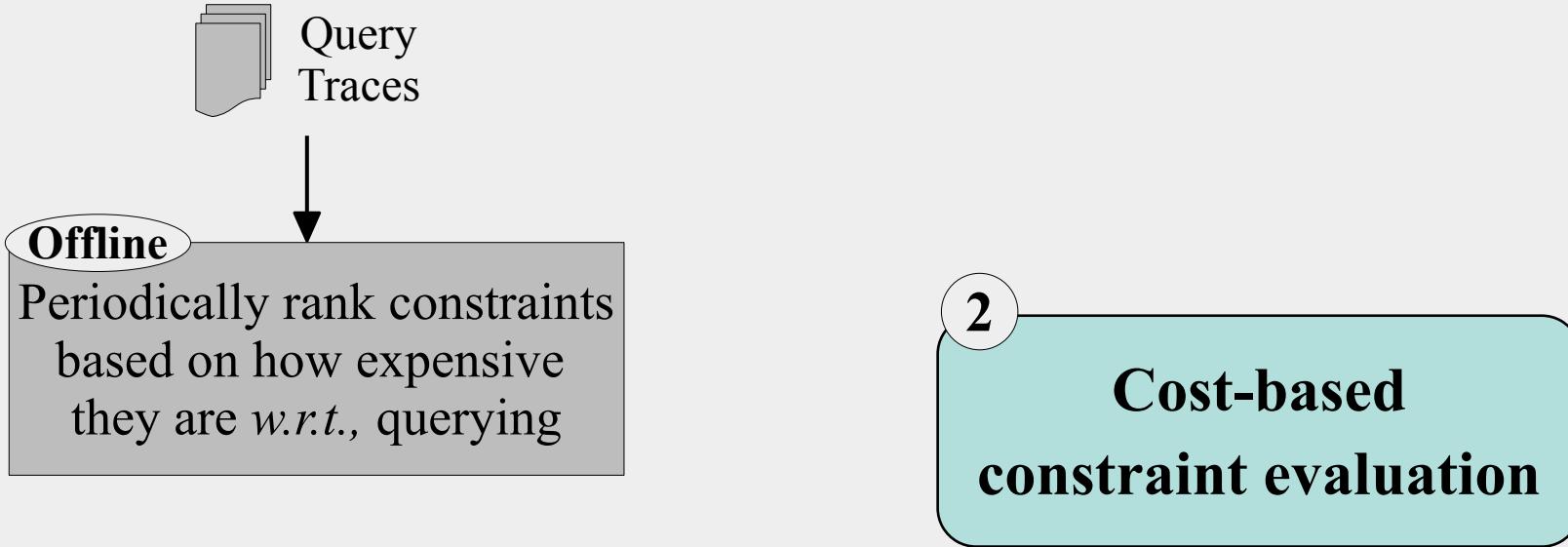


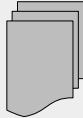
1

**Objective-based
candidate ranking**

2

**Cost-based
constraint evaluation**





Query
Traces



Offline

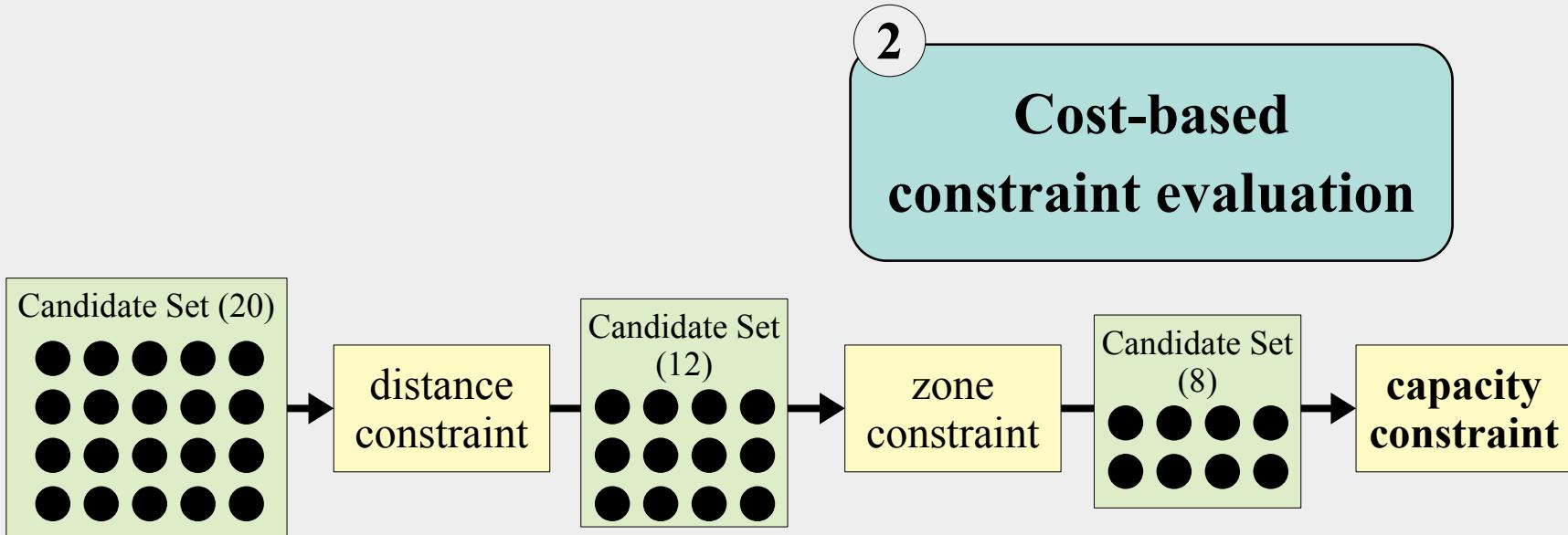
Periodically rank constraints
based on how expensive
they are *w.r.t.*, querying

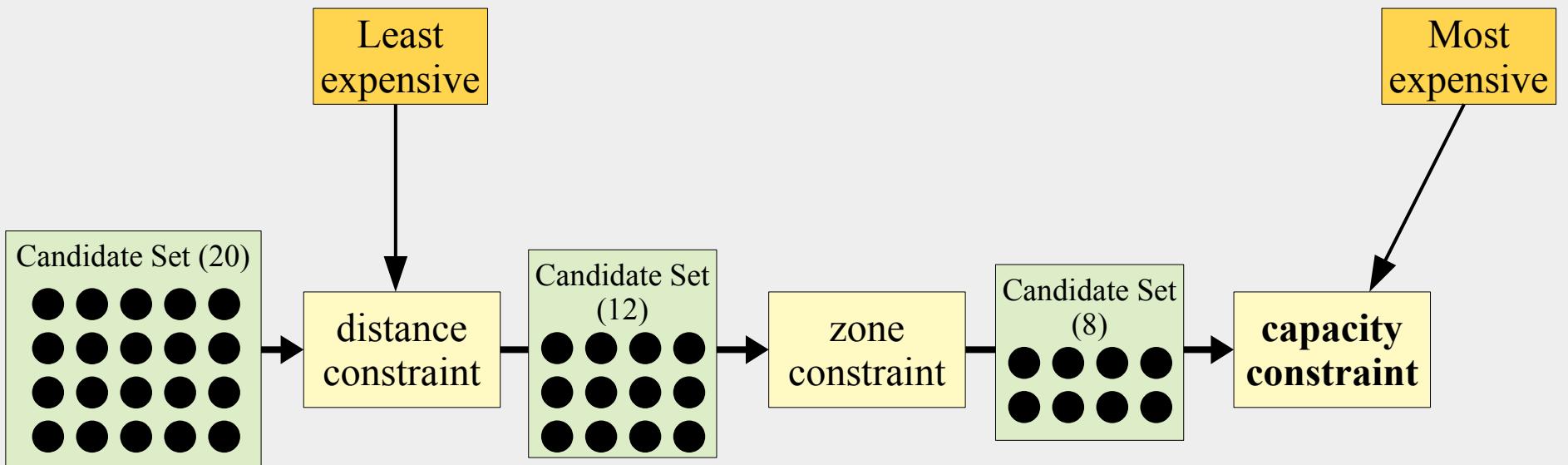
2

Cost-based constraint evaluation

Online

Evaluate least-expensive
constraints first





**Input:**

r : instance of a homing request consisting of: demands (VNFs), constraints, objective function(s), and a set of initial candidates for each demand
p : instance of a heuristic algorithm (e.g., best-fit, exhaustive)
step : incremental step (candidate subset) size
tol_thresh : local solution tolerance threshold

```
1: procedure FINDSOLUTION(r, p, step, tol_thresh)
2:   RANKCANDIDATES(r)
3:   best = null                                ▷ keep track of best solution
4:   tolerance = 0                             ▷ used for stopping_condition
5:   Start with empty available-candidates for each demand
6:   while True do
7:     if stopping_criteria then
8:       break
9:     for each d  $\in$  r.demands do
10:      increment available-candidates[d] by step
11:      sol = p.solution(r, available-candidates)           ▷ triggers
           constraints evaluation
12:      if sol = null then
13:        step = step * 2
14:        go to 6
15:      step = original size
16:      if first solution or sol is better than best then
17:        best  $\leftarrow$  sol
18:        tolerance = 0
19:      else                               ▷ sol did not improve solution quality
20:        tolerance ++
21:   return best
```

**Input:**

r : instance of a homing request consisting of: demands (VNFs), constraints, objective function(s), and a set of initial candidates for each demand
p : instance of a heuristic algorithm (e.g., best-fit, exhaustive)
step : incremental step (candidate subset) size
tol_thresh : local solution tolerance threshold

```
1: procedure FINDSOLUTION(r, p, step, tol_thresh)
2:   RANKCANDIDATES(r)
3:   best = null                                ▷ keep track of best solution
4:   tolerance = 0                               ▷ used for stopping_condition
5:   Start with empty available-candidates for each demand
6:   while True do
7:     if stopping_criteria then
8:       break
9:     for each d ∈ r.demands do
10:      increment available-candidates[d] by step
11:      sol = p.solution(r, available-candidates)           ▷ triggers
          constraints evaluation
12:      if sol = null then
13:        step = step * 2
14:        go to 6
15:      step = original size
16:      if first solution or sol is better than best then
17:        best ← sol
18:        tolerance = 0
19:      else                                         ▷ sol did not improve solution quality
20:        tolerance ++
21:   return best
```

**Input:**

r : instance of a homing request consisting of: demands (VNFs), constraints, objective function(s), and a set of initial candidates for each demand
p : instance of a heuristic algorithm (e.g., best-fit, exhaustive)
step : incremental step (candidate subset) size
tol_thresh : local solution tolerance threshold

```
1: procedure FINDSOLUTION(r, p, step, tol_thresh)
2:   RANKCANDIDATES(r)
3:   best = null                                ▷ keep track of best solution
4:   tolerance = 0                               ▷ used for stopping_condition
5:   Start with empty available-candidates for each demand
6:   while True do
7:     if stopping_criteria then
8:       break
9:     for each d in r.demands do
10:      increment available-candidates[d] by step
11:      sol = p.solution(r, available-candidates)           ▷ triggers
          constraints evaluation
12:      if sol = null then
13:        step = step * 2
14:        go to 6
15:        step = original size
16:        if first solution or sol is better than best then
17:          best  $\leftarrow$  sol
18:          tolerance = 0
19:        else                                     ▷ sol did not improve solution quality
20:          tolerance ++
21:   return best
```

**Input:**

r : instance of a homing request consisting of: demands (VNFs), constraints, objective function(s), and a set of initial candidates for each demand
p : instance of a heuristic algorithm (e.g., best-fit, exhaustive)
step : incremental step (candidate subset) size
tol_thresh : local solution tolerance threshold

```
1: procedure FINDSOLUTION(r, p, step, tol_thresh)
2:   RANKCANDIDATES(r)
3:   best = null                                ▷ keep track of best solution
4:   tolerance = 0                               ▷ used for stopping_condition
5:   Start with empty available-candidates for each demand
6:   while True do
7:     if stopping_criteria then
8:       break
9:     for each d in r.demands do
10:      increment available-candidates[d] by step
11:      sol = p.solution(r, available-candidates)           ▷ triggers
          constraints evaluation
12:      if sol = null then
13:        step = step * 2
14:        go to 6
15:        step = original size
16:        if first solution or sol is better than best then
17:          best  $\leftarrow$  sol
18:          tolerance = 0
19:        else                                     ▷ sol did not improve solution quality
20:          tolerance ++
21:      return best
```



Implementation



Implemented in *python* on top the homing and allocation (HAS) service of ONAP

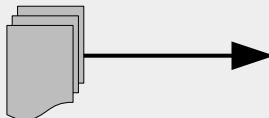


Implementation



Implemented in *python* on top the homing and allocation (HAS) service of ONAP

Production
Homing
Traces



Built an extensive trace-driven emulation framework to emulate controllers' behavior in production setting



Evaluation

Mimics production settings
→ *1000s* of candidates



Evaluation

Mimics production settings
→ ***1000s*** of candidates

Evaluated ***1200*** homing requests
spanning ***12*** NSP production services



Evaluation

Mimics production settings
→ ***1000s*** of candidates

Evaluated ***1200*** homing requests
spanning ***12*** NSP production services

Evaluated 3 optimization heuristics



Evaluation

Mimics production settings
→ **1000s** of candidates

Evaluated **1200** homing requests
spanning **12** NSP production services

Evaluated 3 optimization heuristics

Backtracking best-fit (BACBF)

Random

Shortest Path First (**SPF**)



Evaluation

Two configurations for each heuristic:

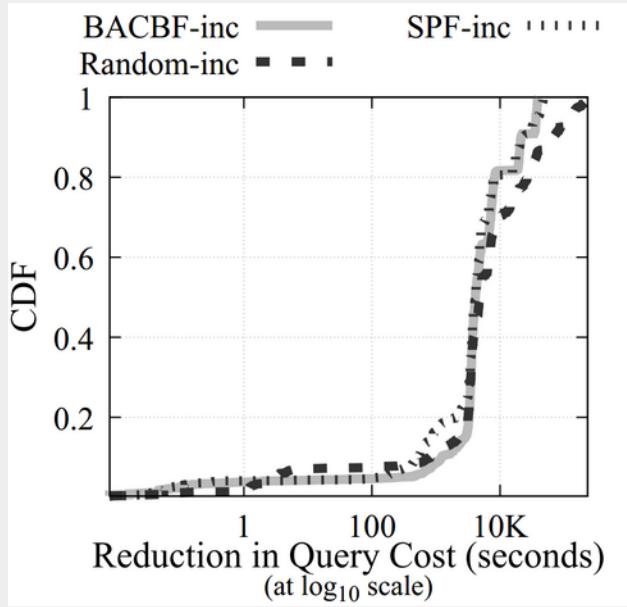
- **all** → all candidates are evaluated
- **inc** → only top candidates are evaluated

Backtracking best-fit (BACBF)

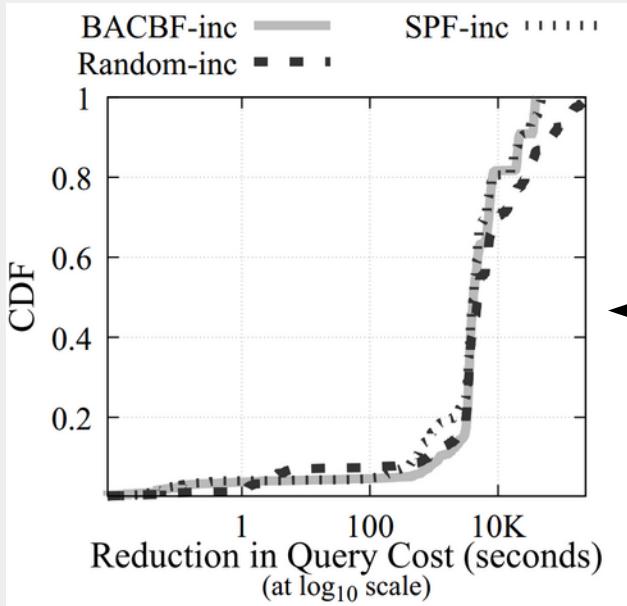
Random

Shortest Path First (**SPF**)

Evaluation

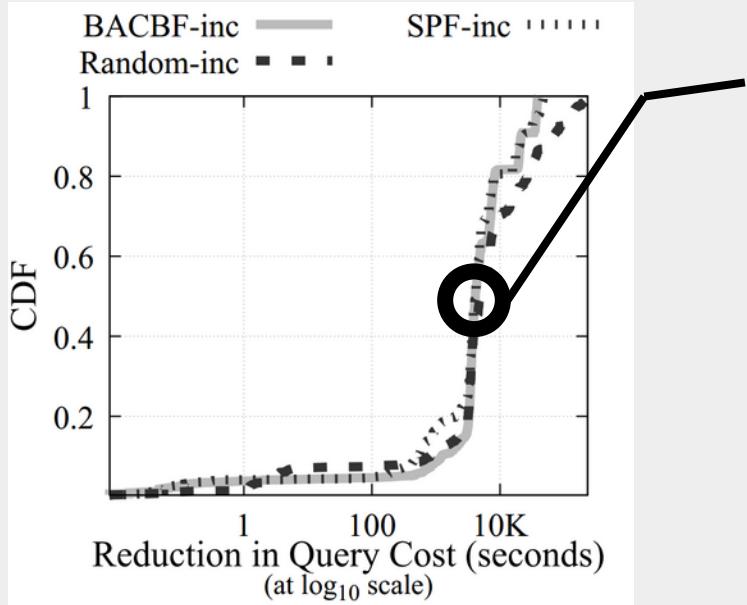


Evaluation



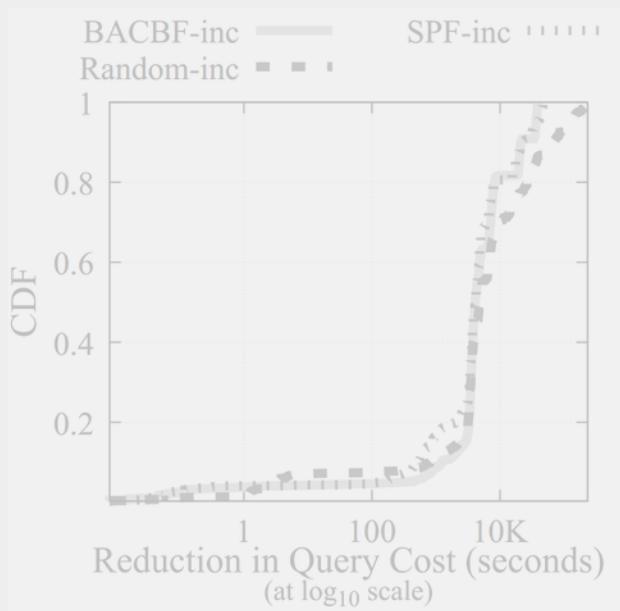
Query cost reduction the incremental approach was able to provide over non-incremental versions for **1200** requests

Evaluation

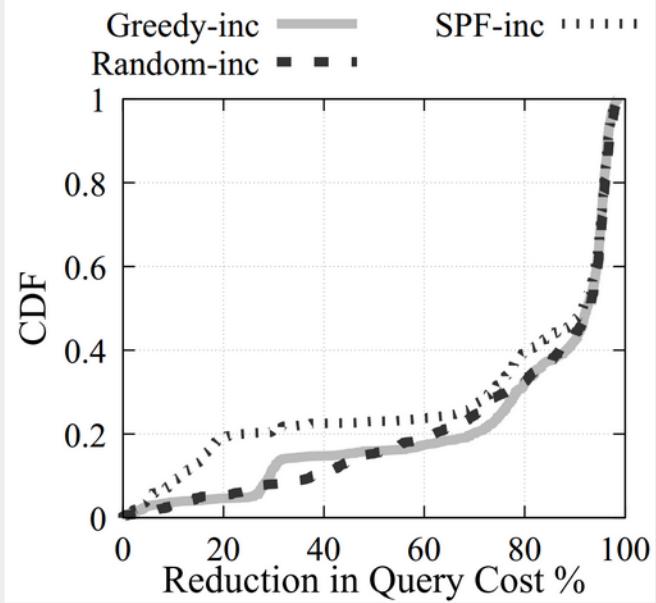


The incremental approach saved
~5K seconds in *per-request* query cost for
50% of requests across all 3 heuristics

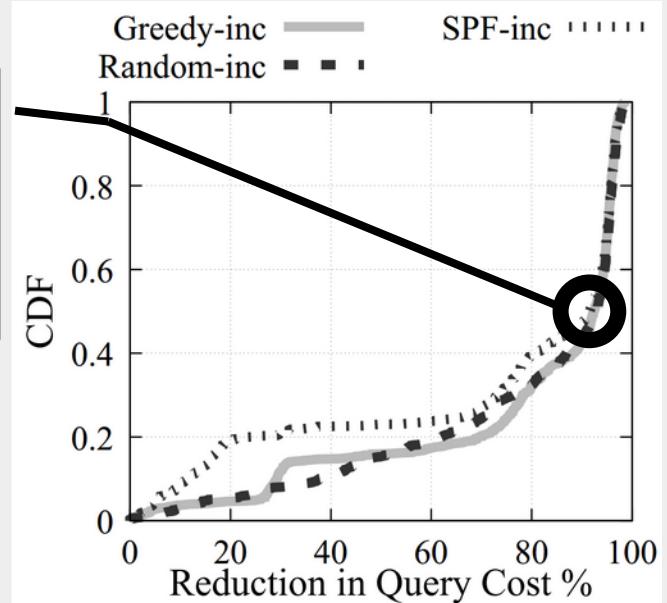
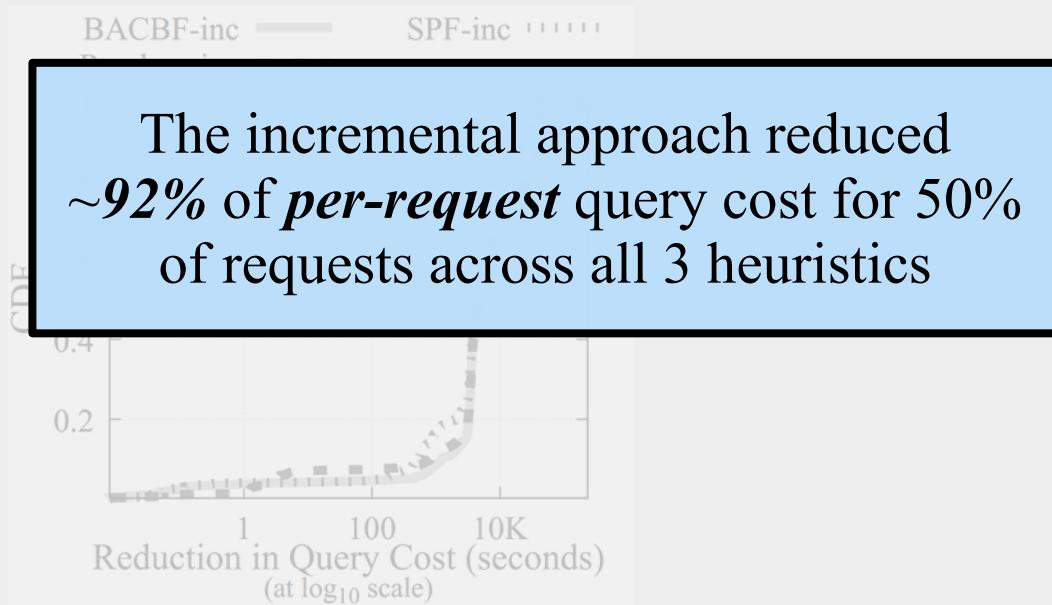
Evaluation



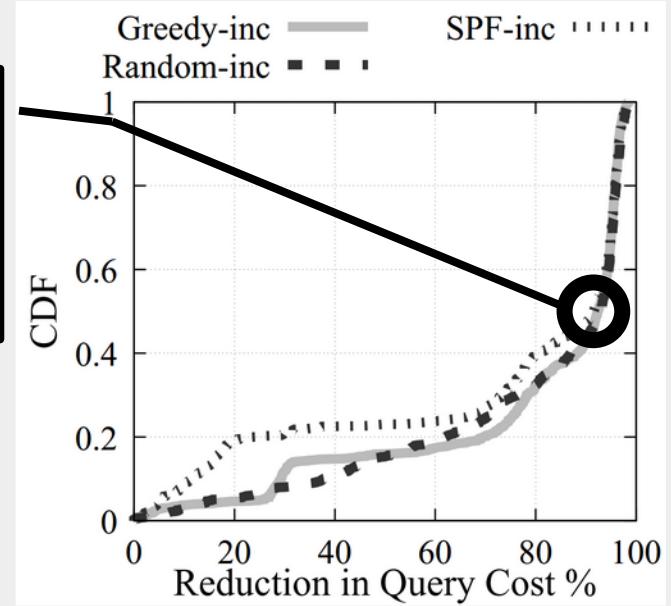
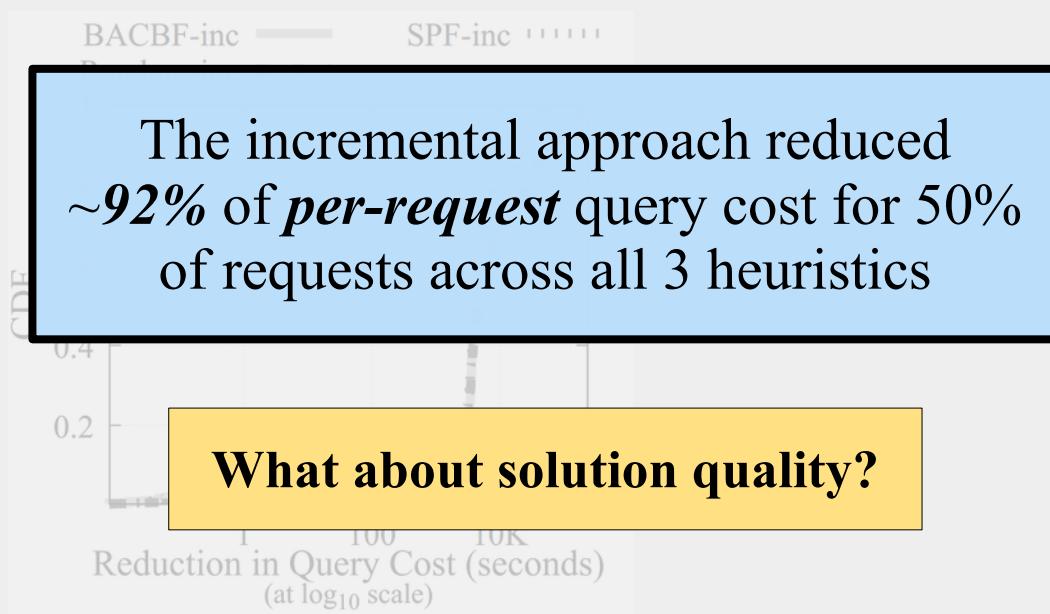
In %



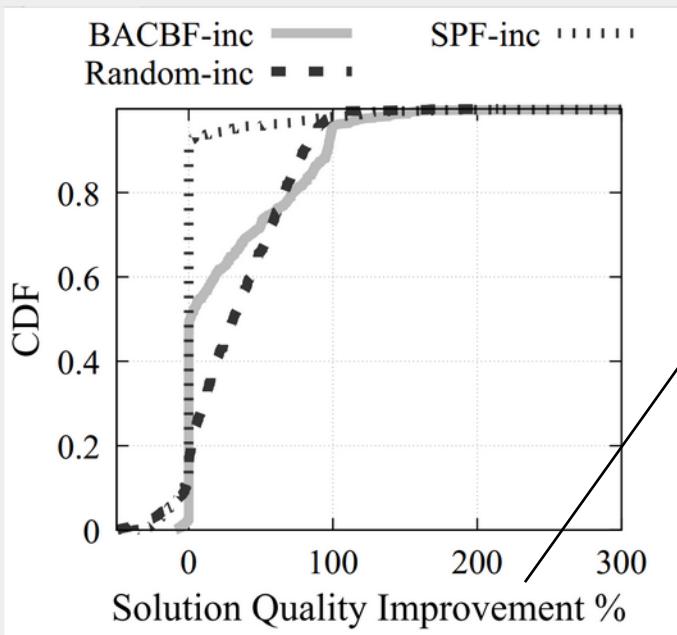
Evaluation



Evaluation



Evaluation

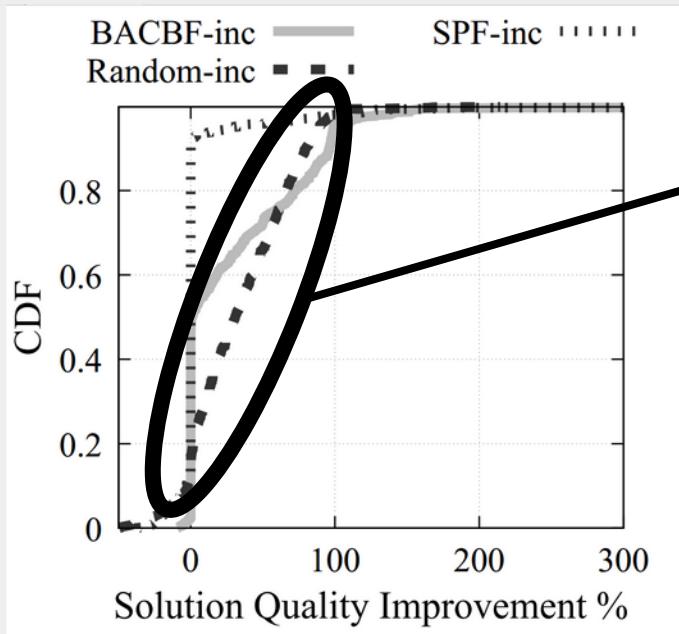


Improvement of solution quality that the incremental approach was able to bring to each of these three heuristics (same experiment)

Value > 0 means incremental approach was able to improve solution quality

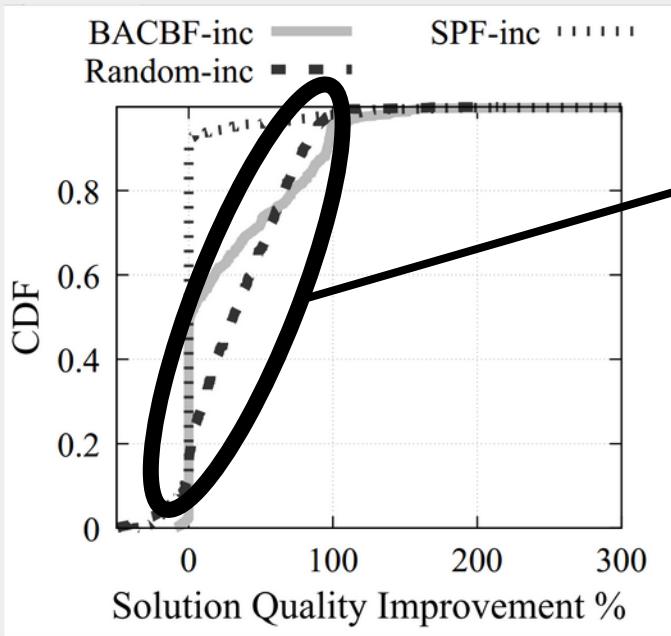
Value < 0 means incremental approach produced lower quality solutions

Evaluation

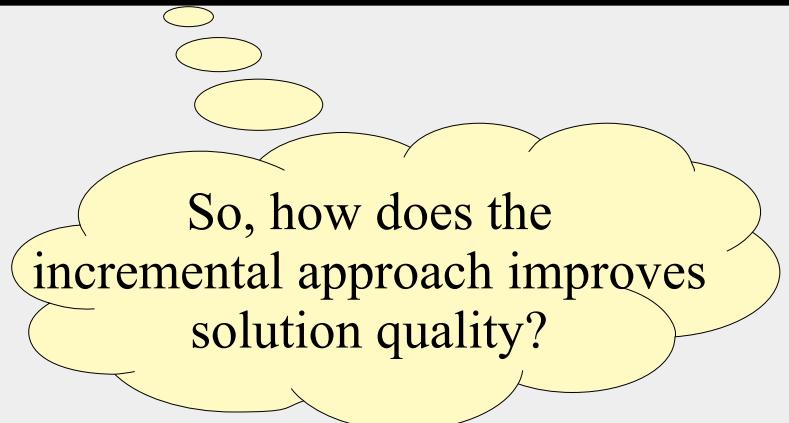


Random-inc provides much better homing solutions than *Random-all* for at least 75% of requests

Evaluation

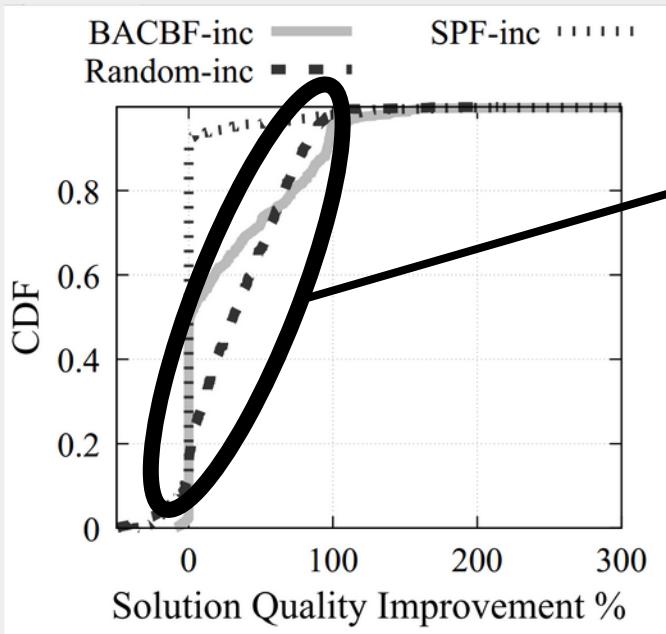


Random-inc provides much better homing solutions than ***Random-all*** for at least 75% of requests



So, how does the incremental approach improves solution quality?

Evaluation



Random-inc provides much better homing solutions than ***Random-all*** for at least 75% of requests

Incremental approach increases chances of choosing “*good*” candidates thanks to objective-based candidate ranking



Reduced (and more focused) visibility of the underlying infrastructure can actually improve solution quality



Evaluation

Takeaway

Intelligently reducing the number of queries to be sent (lower visibility) can actually improve solution quality since we're increasing the chance for a given heuristic to get stuck with a better local optimum.



Contributions

1

Analyzed and identified challenges and limitations facing the homing service in production systems

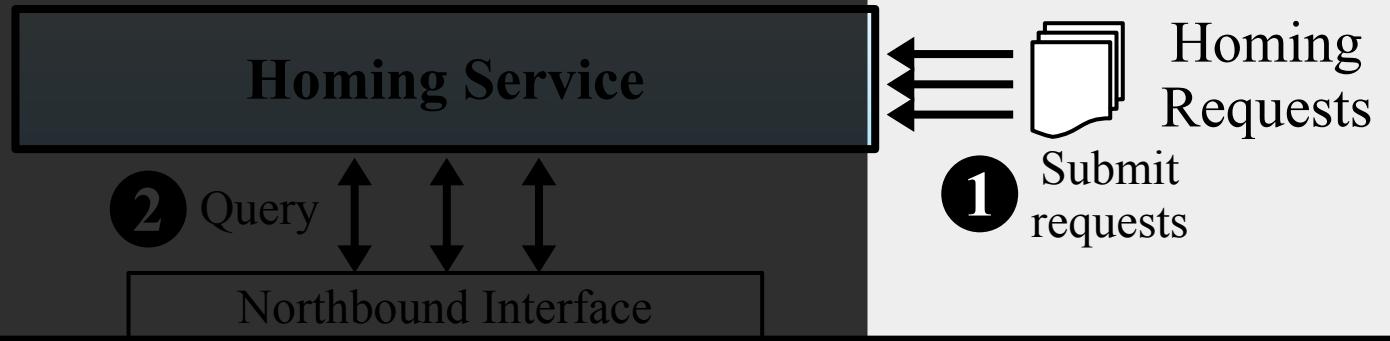
2

Proposed a novel incremental approach that significantly reduces load and improves quality

3

Designed and built an extensive trace-driven emulation framework resembling production behavior

Publications: [under review'21]



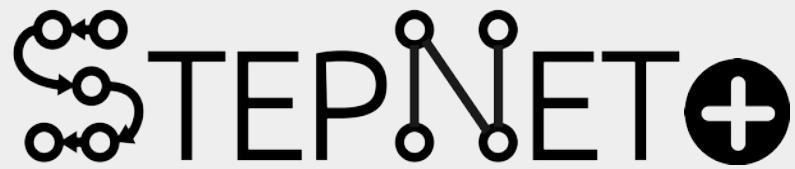
Cloud (C) & Service (S) Controllers

Challenge-4

How can we account for *dependencies* between homing requests to make the homing process even *more efficient*?

Geo-distributed Compute/Network Resources (micro datacenters)





Accounting for homing requests dependencies

Overview

?

Why do dependencies occur?

Overview

?

Why do dependencies occur?

A typical homing service receives
1000s of requests on a daily basis

Overview

?

Why do dependencies occur?

A typical homing service receives
1000s of requests on a daily basis

+

Each request takes minutes to solve

Overview

?

Why do dependencies occur?

A typical homing service receives
1000s of requests on a daily basis

+

Each request takes minutes to solve

!

NSPs deploy distributed instances
of the homing service to increase
overall throughput

Challenges

1

Redundant querying

2

Resource contention

3

Unoptimized resource sharing

Homing
Requests

R1

R2

R3

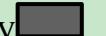
R4

Homing Requests

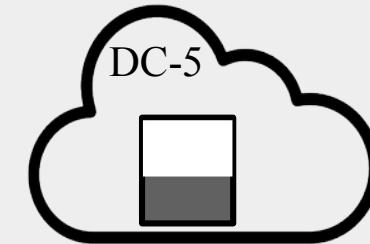
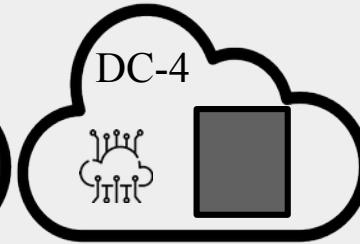
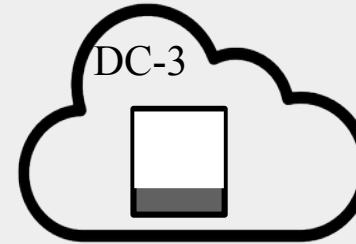
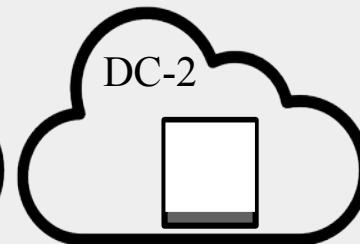
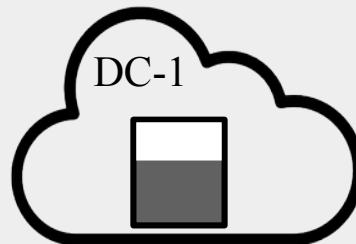
R1 Demands: vG , $vGMux$
Requested capacity 

R3 Demands: VPN
Requested capacity 

R2 Demands: vG , $vGMux$
Requested capacity 

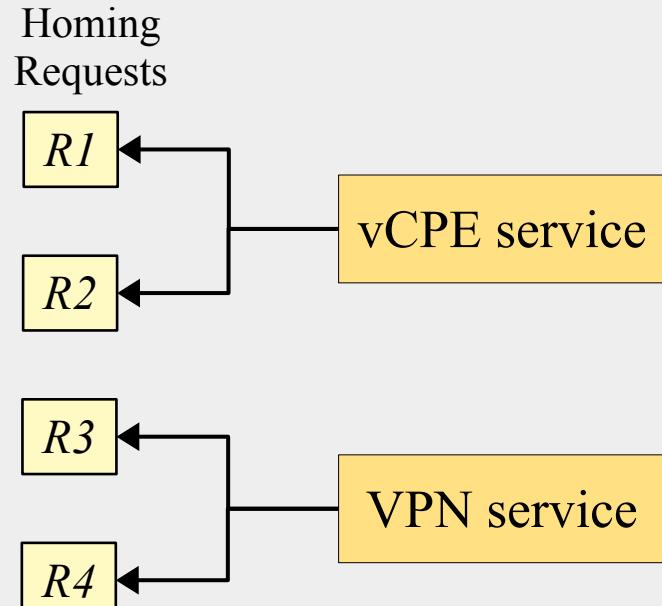
R4 Demands: VPN
Requested capacity 

Pool of Cloud & Service Candidates



Shared $vGMux$ 
Instance

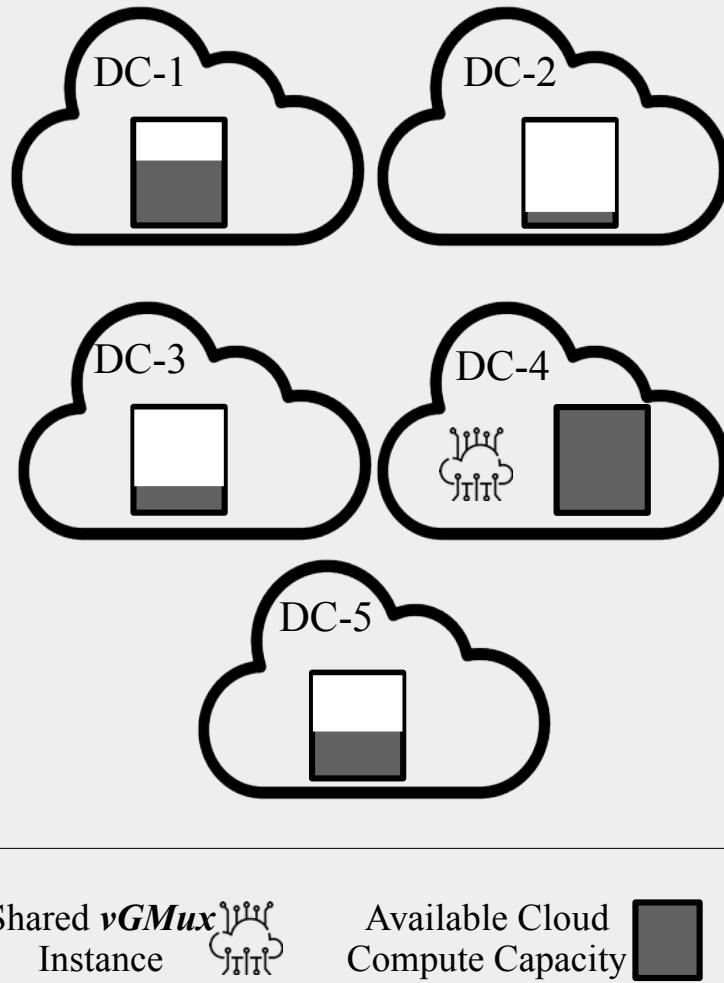
Available Cloud
Compute Capacity 



Homing Requests

$R1$	Demands: vG , $vGMux$
	Requested capacity
$R2$	Demands: vG , $vGMux$
	Requested capacity
$R3$	Demands: VPN
	Requested capacity
$R4$	Demands: VPN
	Requested capacity

Pool of Cloud & Service Candidates



Homing Requests

R1

R2

R3

R4

Homing Requests

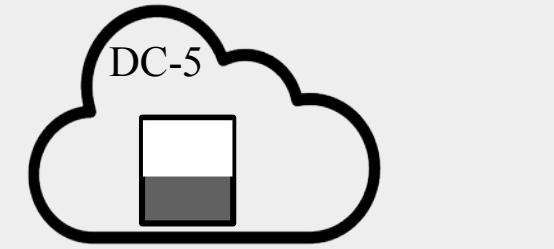
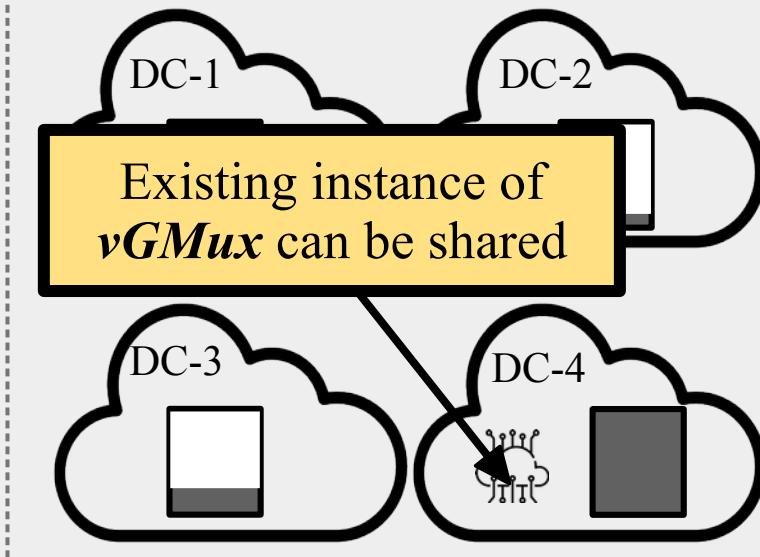
R1 Demands: *vG*, *vGMux*
Requested capacity 

R3 Demands: *VPN*
Requested capacity 

R2 Demands: *vG*, *vGMux*
Requested capacity 

R4 Demands: *VPN*
Requested capacity 

Pool of Cloud & Service Candidates



Homing Service Optimization Instances

Homing Requests

$R1$

$S1$

$R2$

$S2$

$R3$

$S3$

$R4$

$S4$

Homing Requests

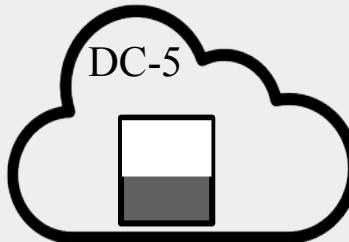
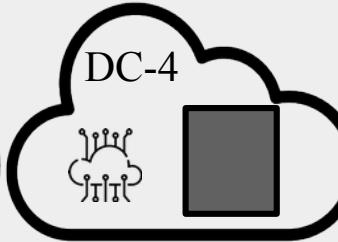
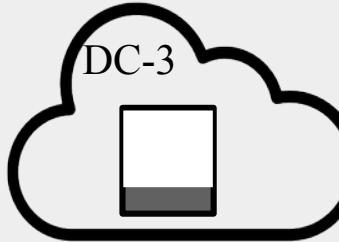
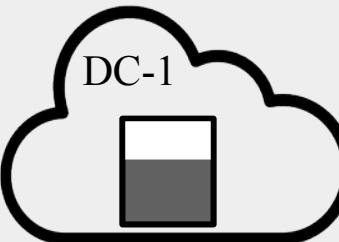
$R1$ Demands: vG , $vGMux$
Requested capacity 

$R3$ Demands: VPN
Requested capacity 

$R2$ Demands: vG , $vGMux$
Requested capacity 

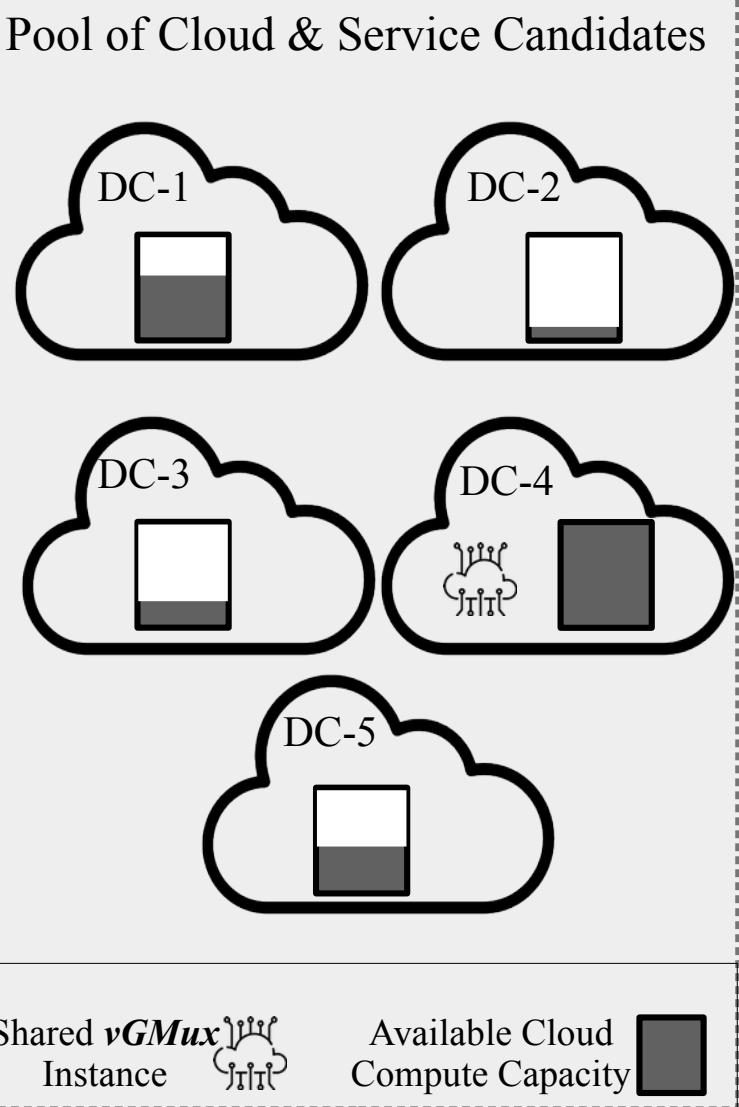
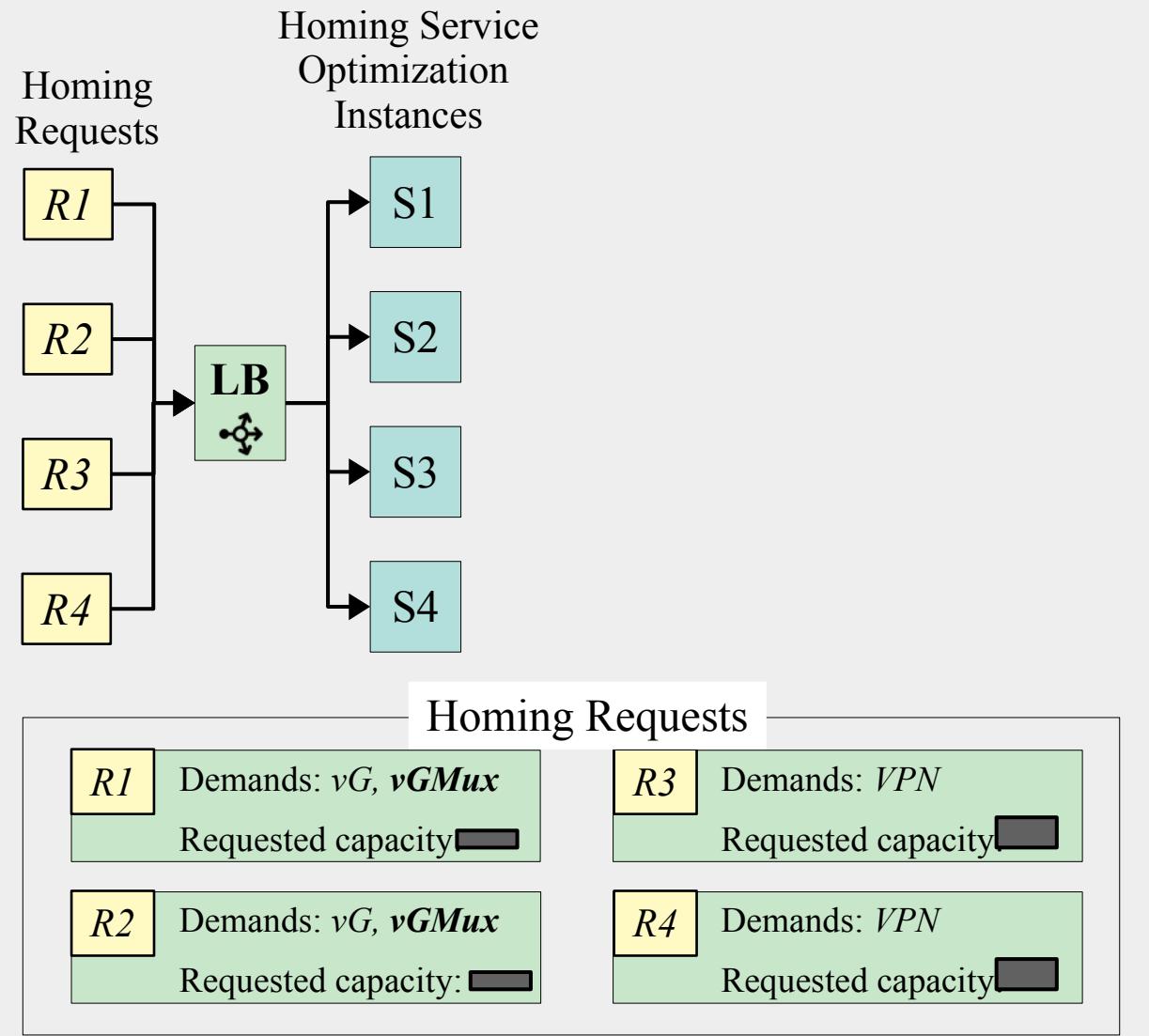
$R4$ Demands: VPN
Requested capacity 

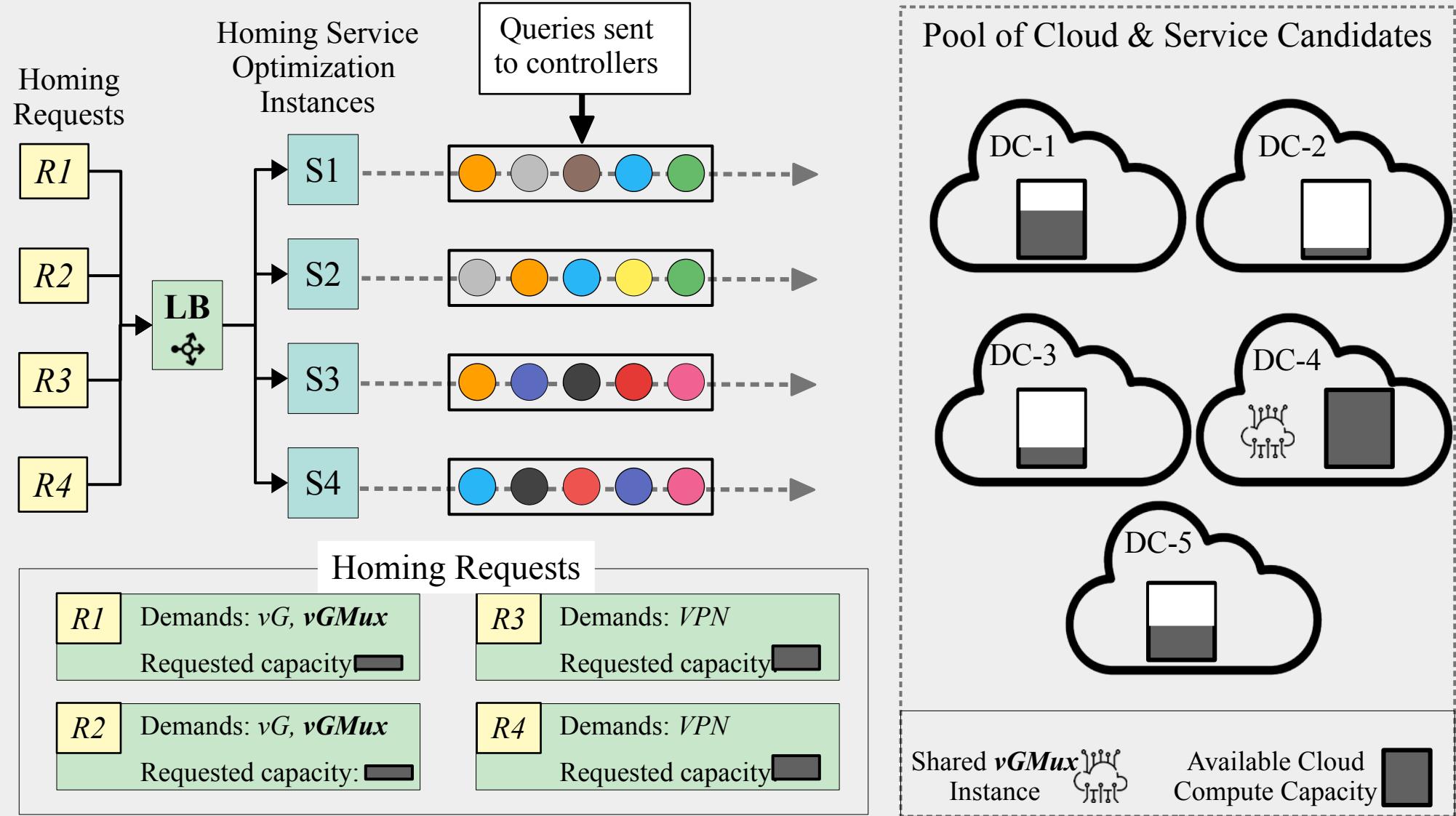
Pool of Cloud & Service Candidates

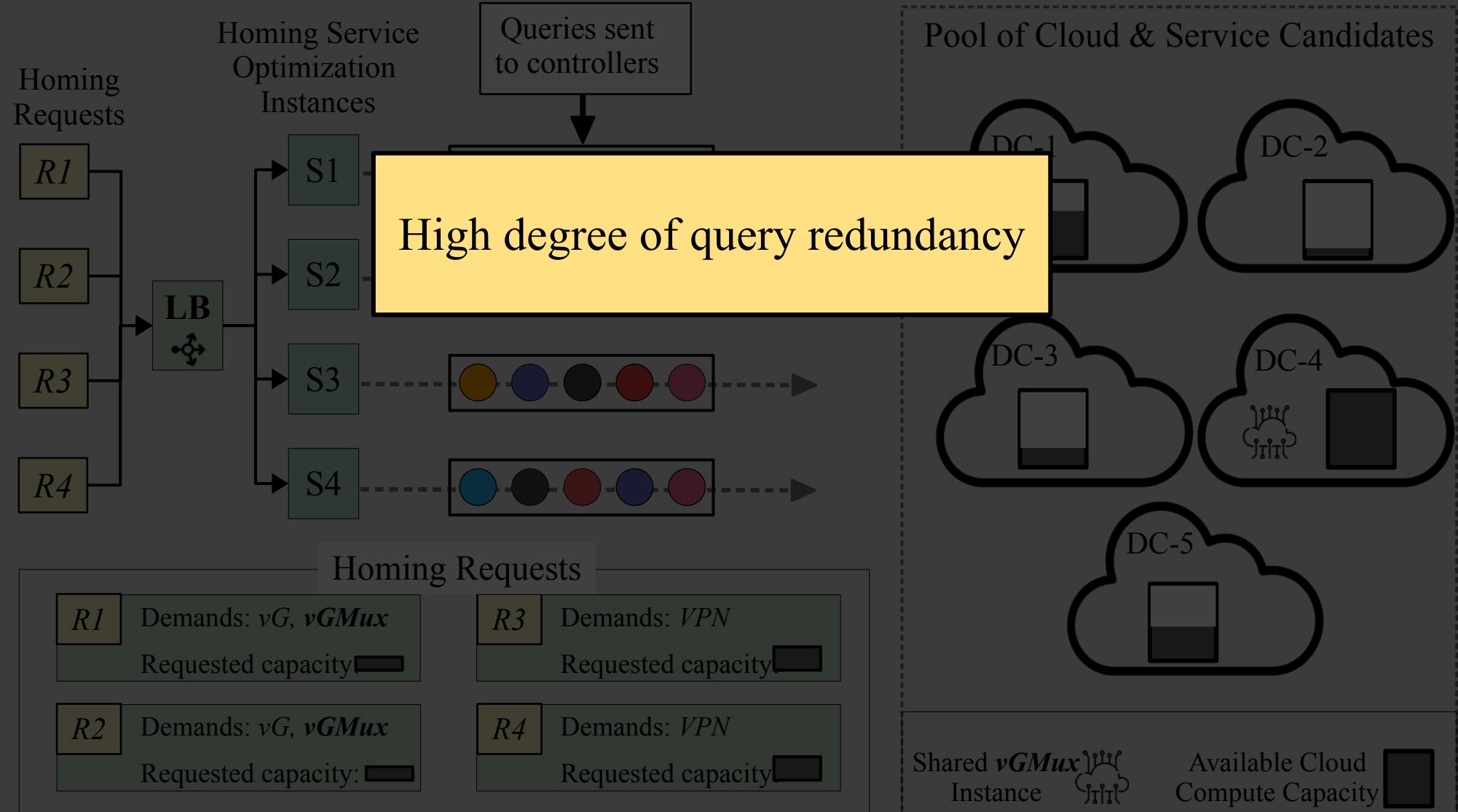


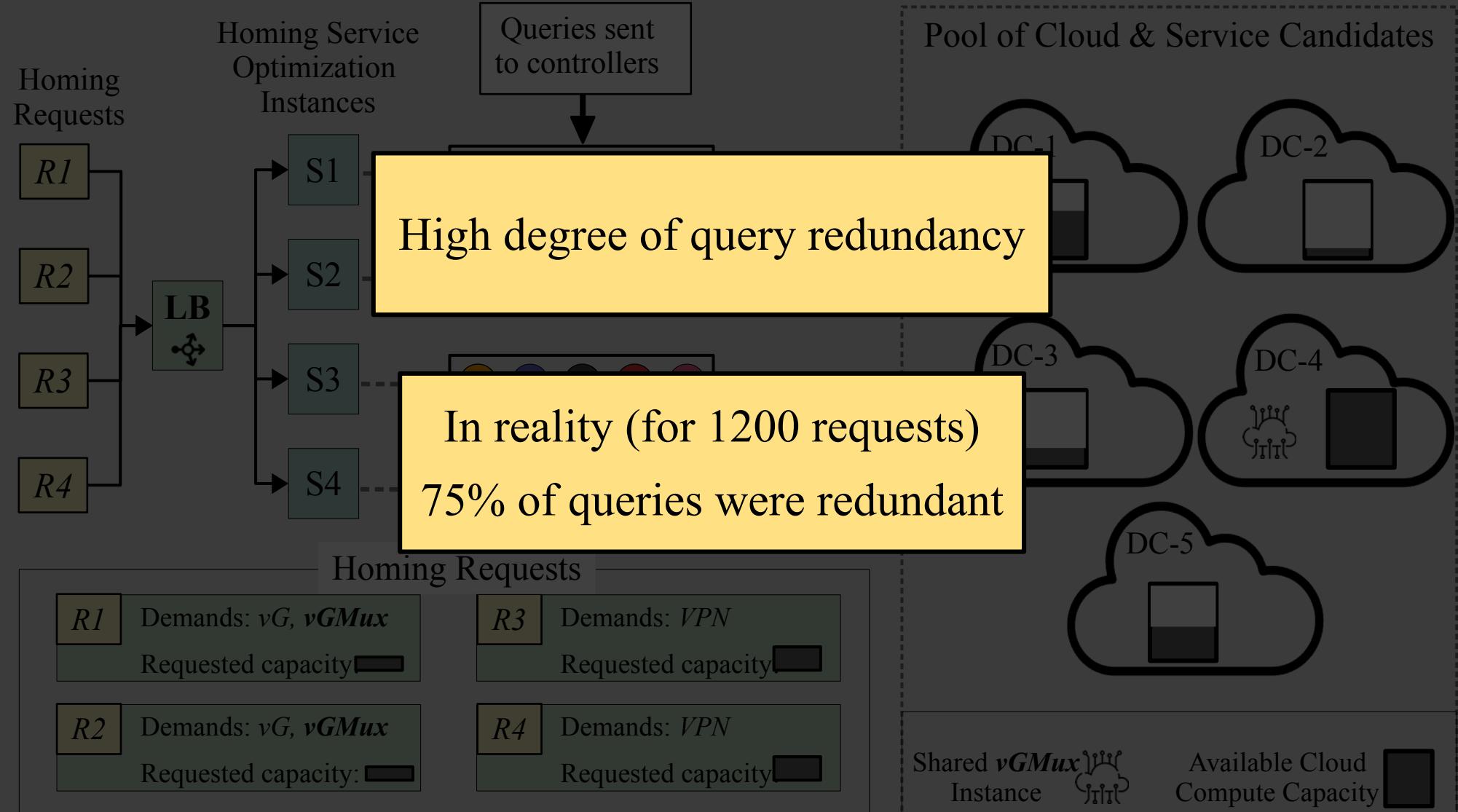
Shared $vGMux$ 
Instance

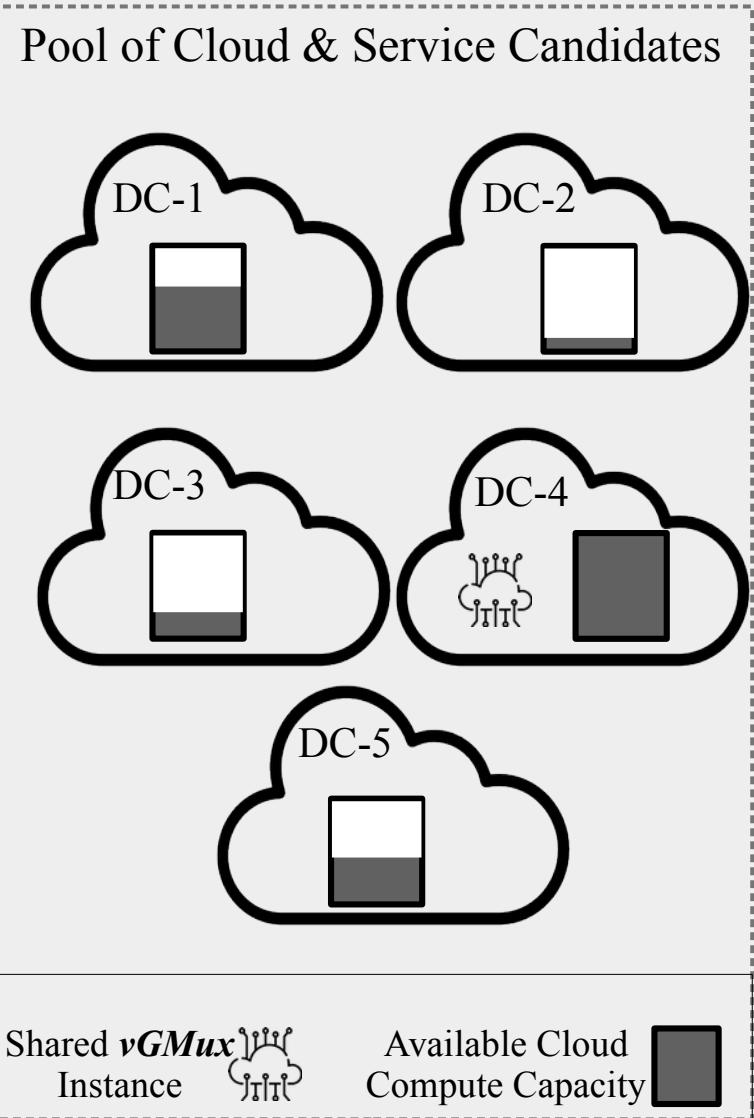
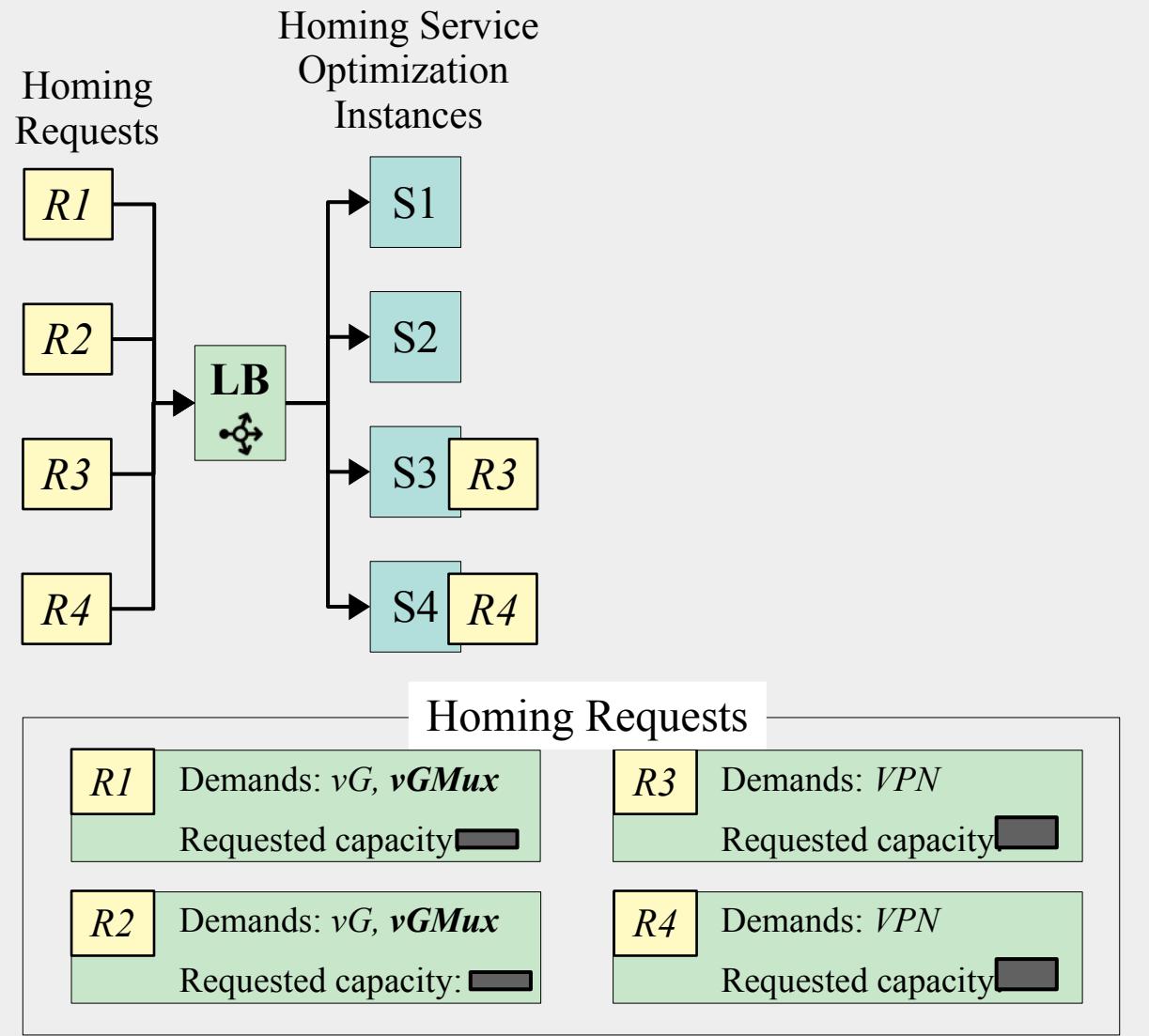
Available Cloud Compute Capacity 

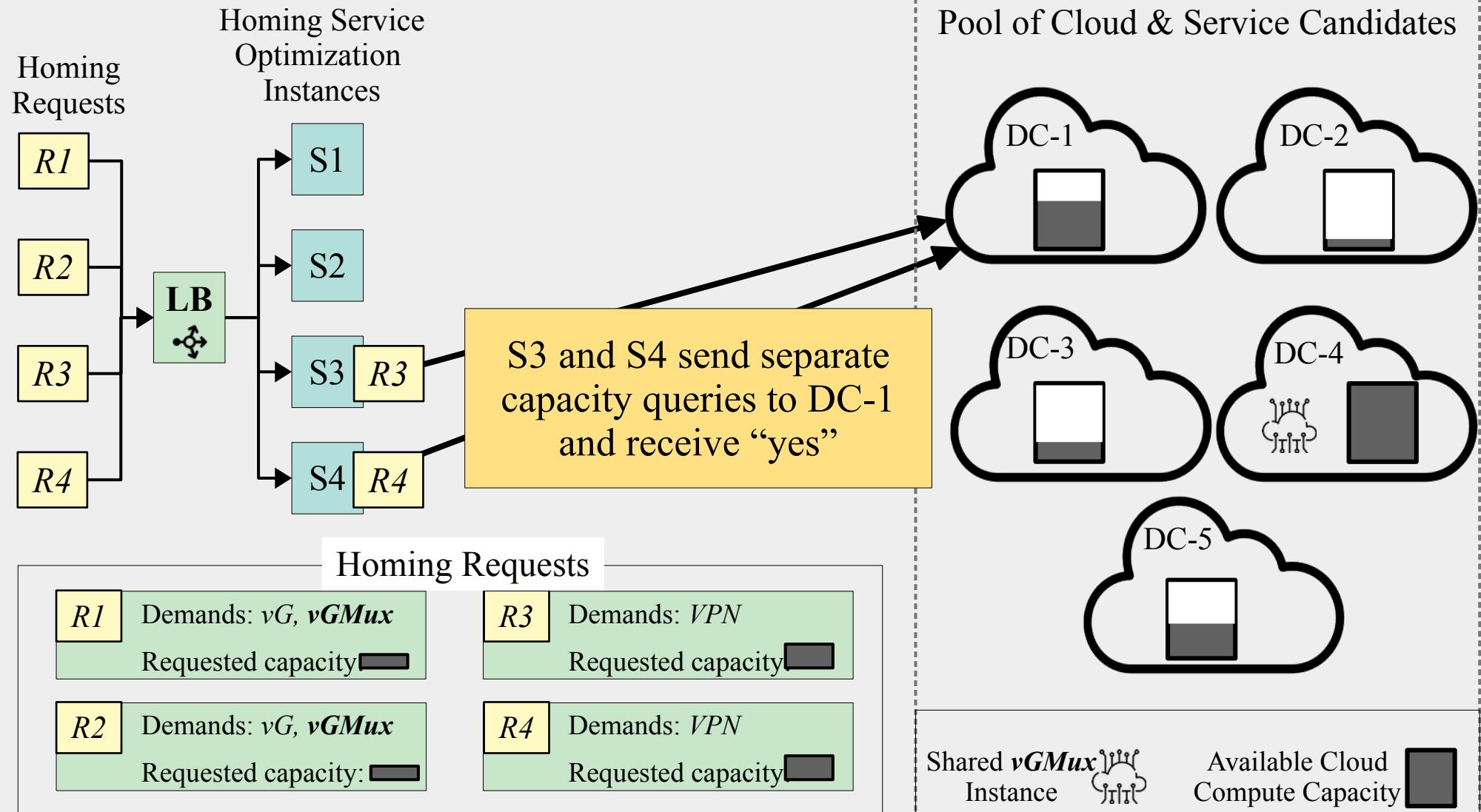


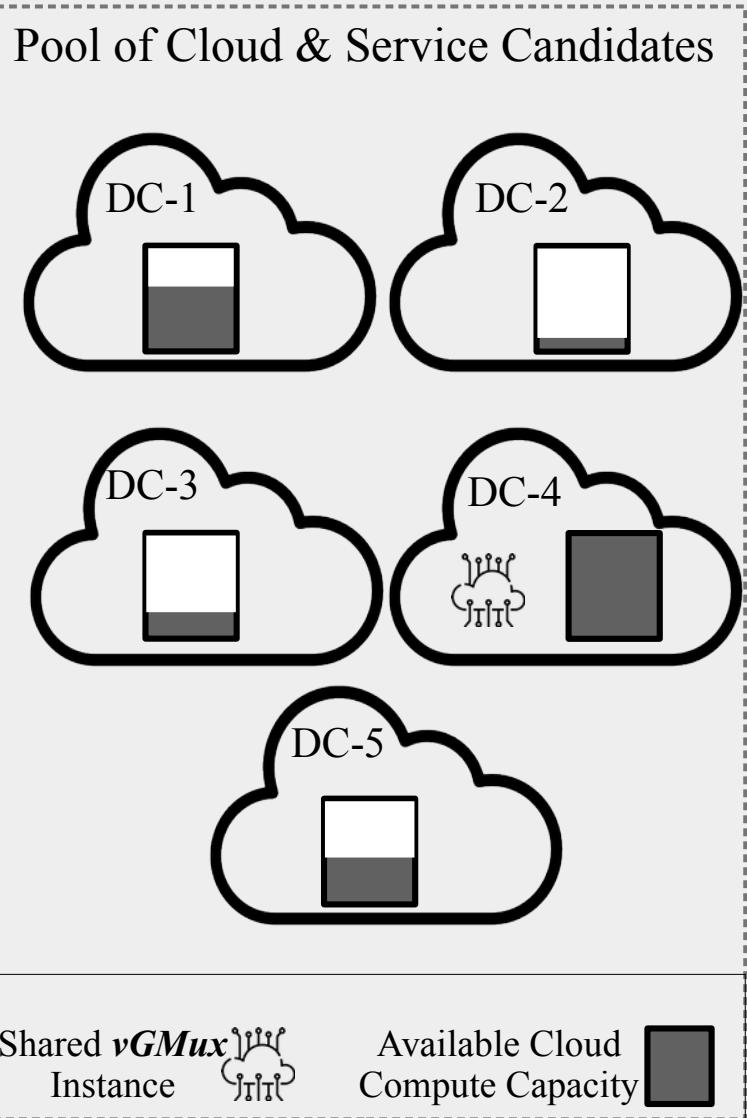
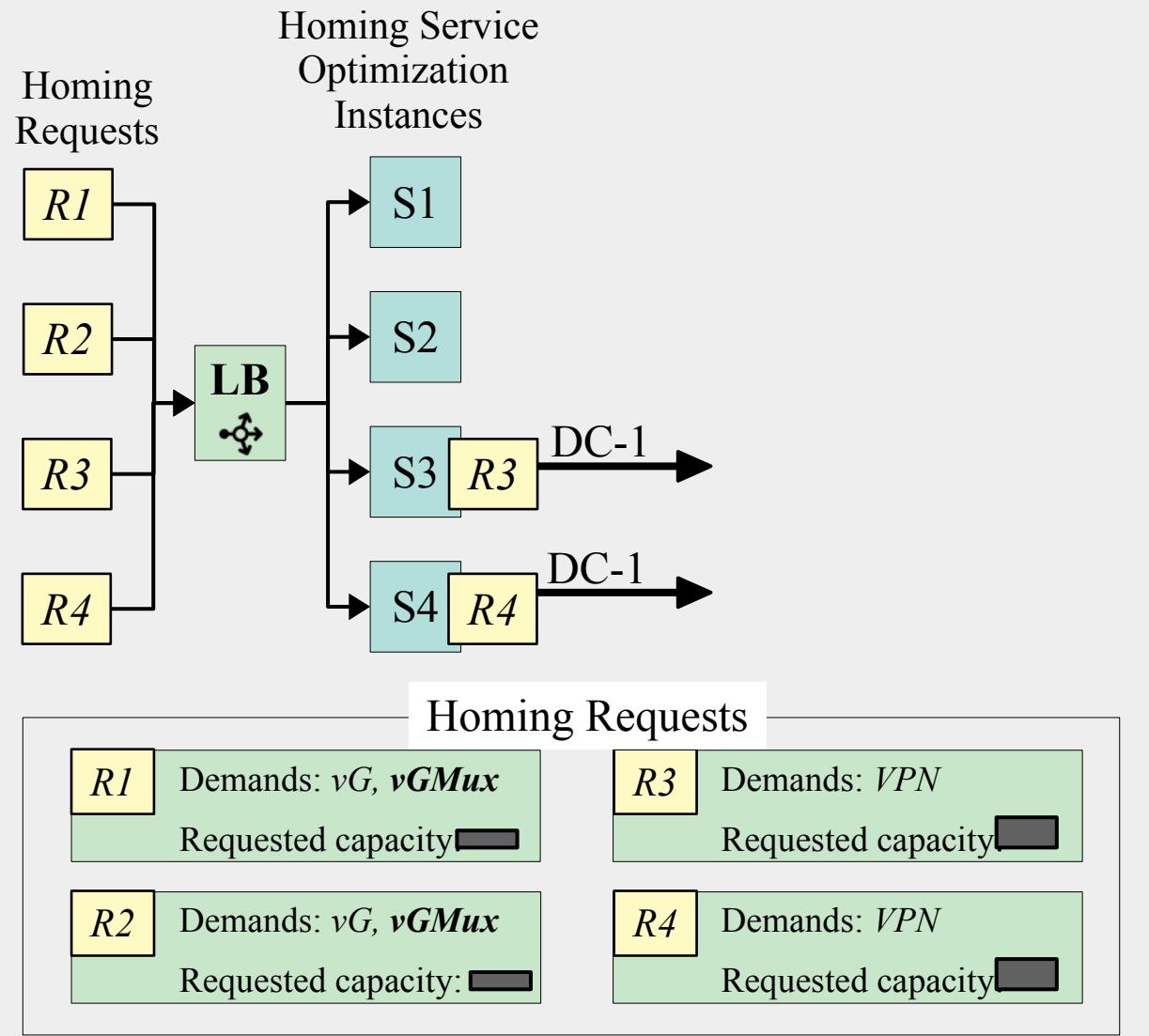


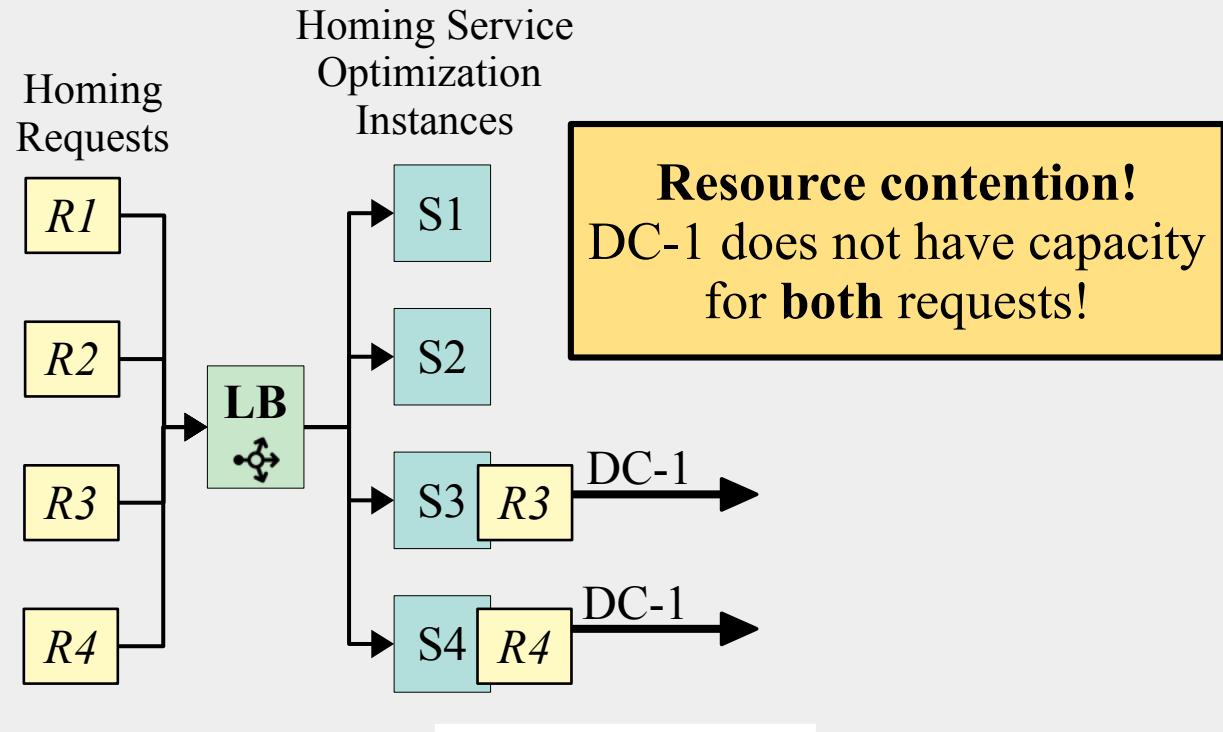






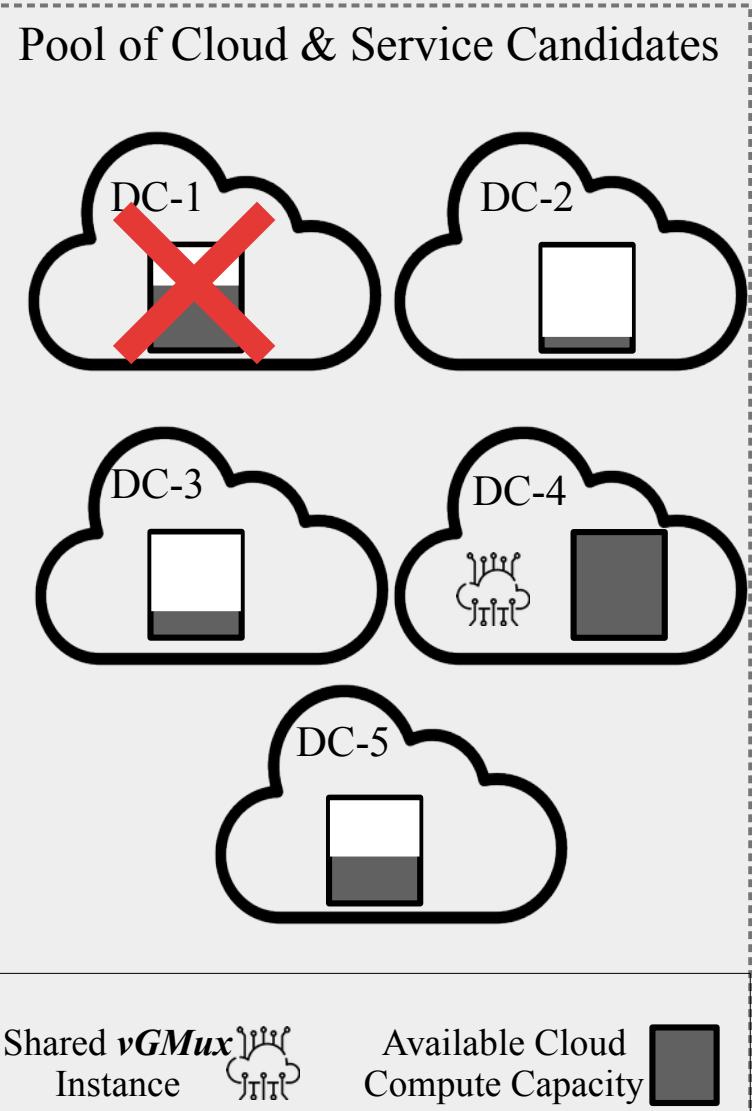


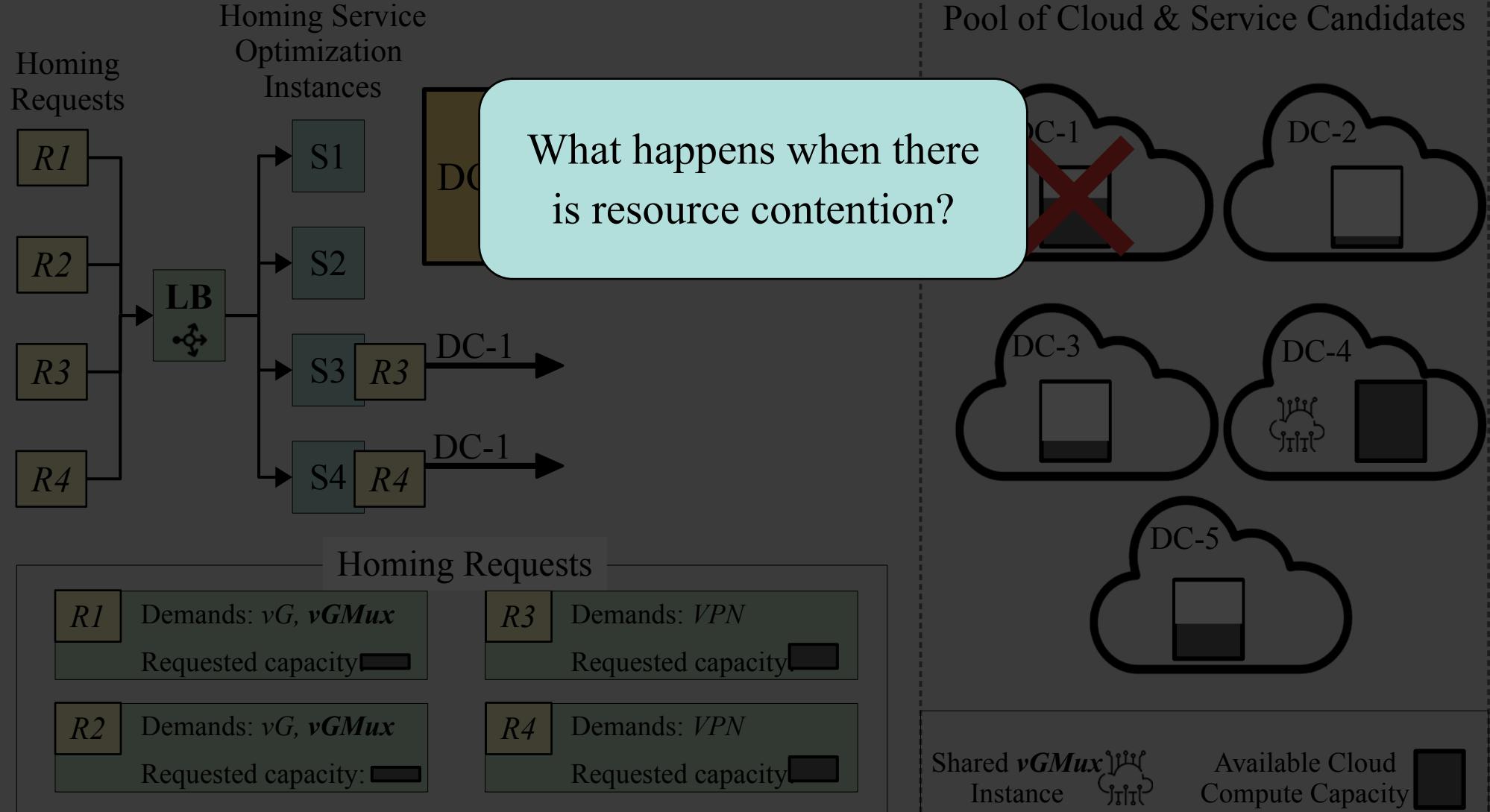


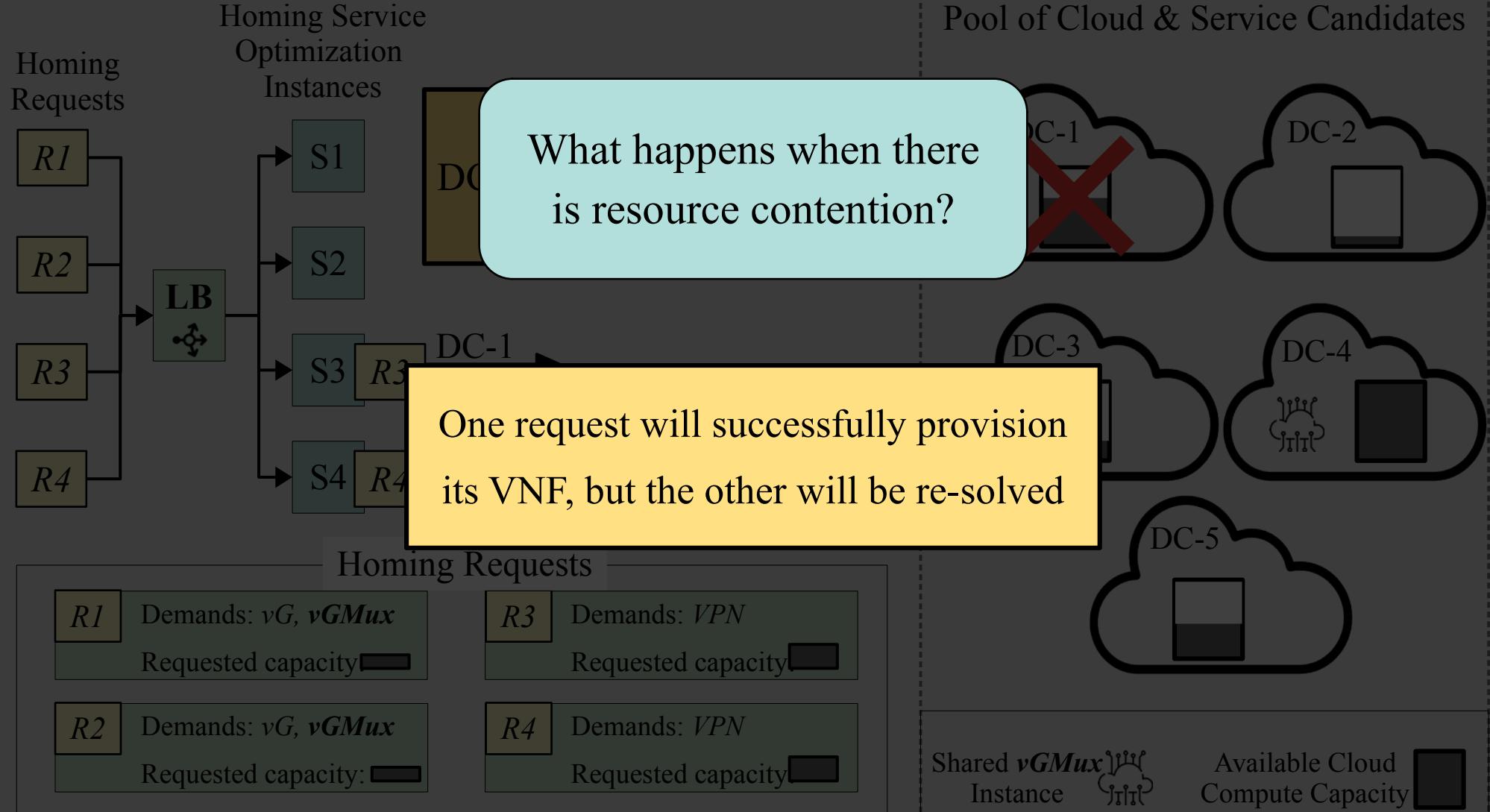


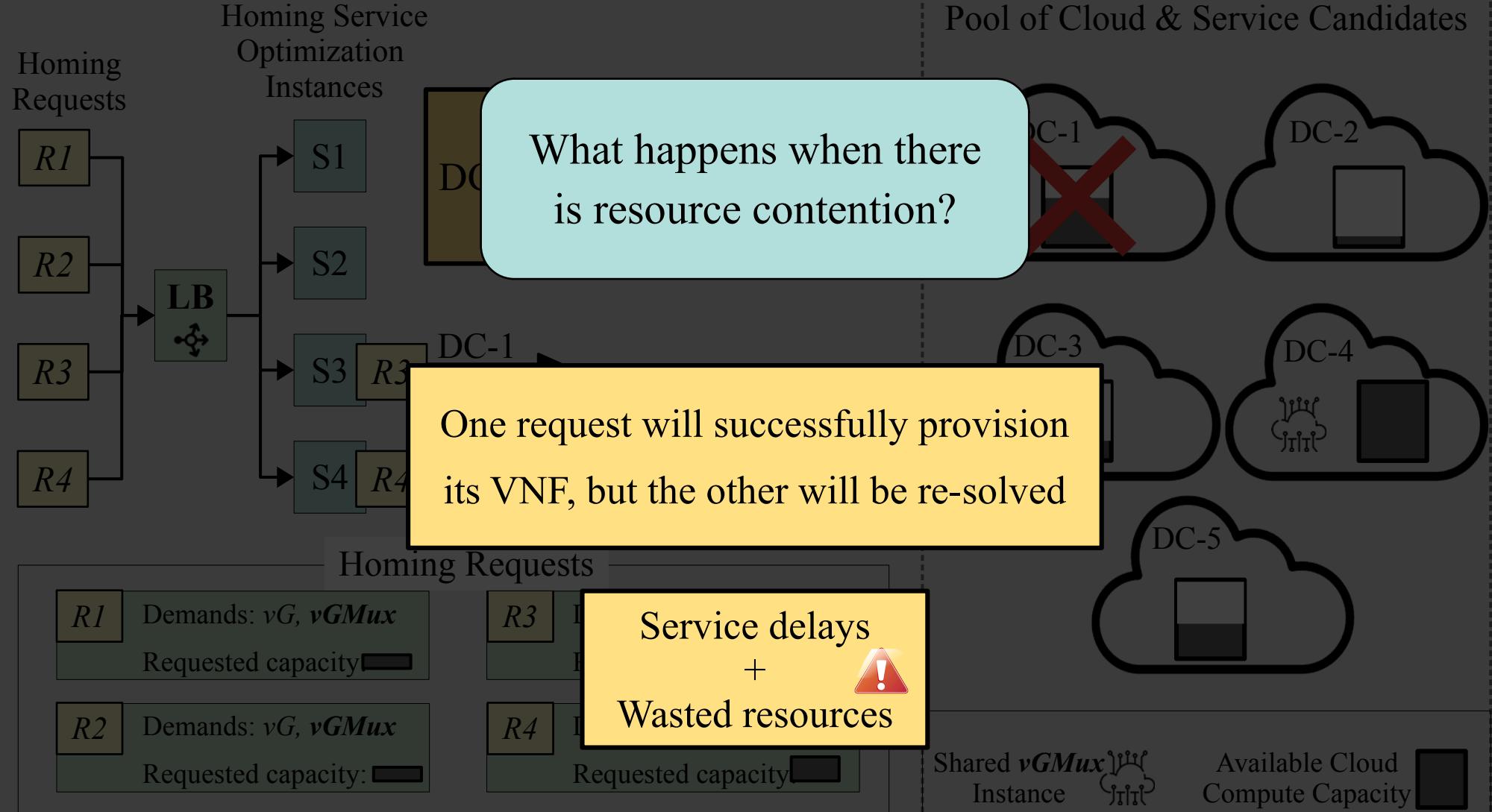
Homing Requests			
R1	Demands: vG , $vGMux$	R3	Demands: VPN
	Requested capacity		Requested capacity

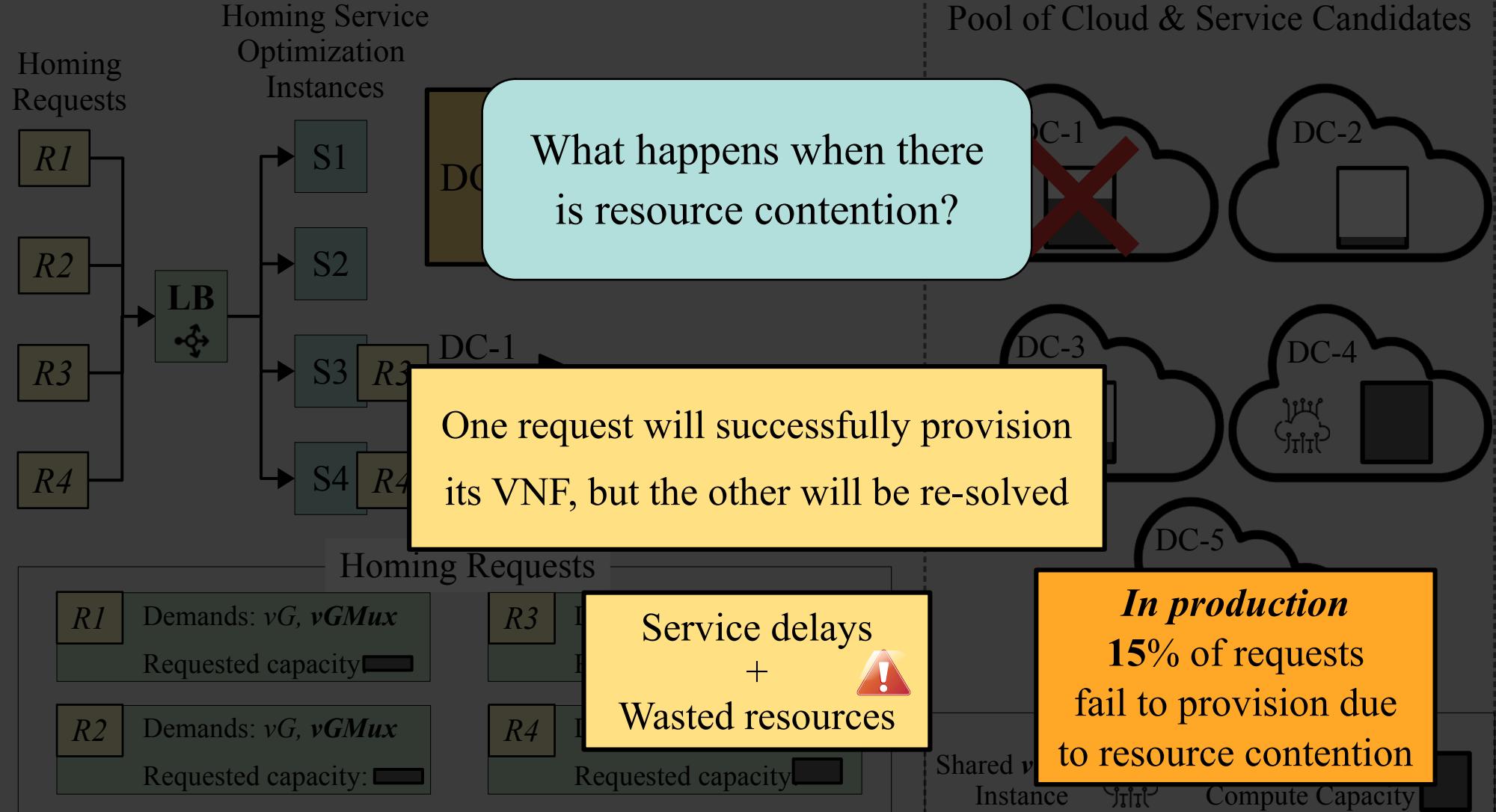
Homing Requests			
R2	Demands: vG , $vGMux$	R4	Demands: VPN
	Requested capacity		Requested capacity

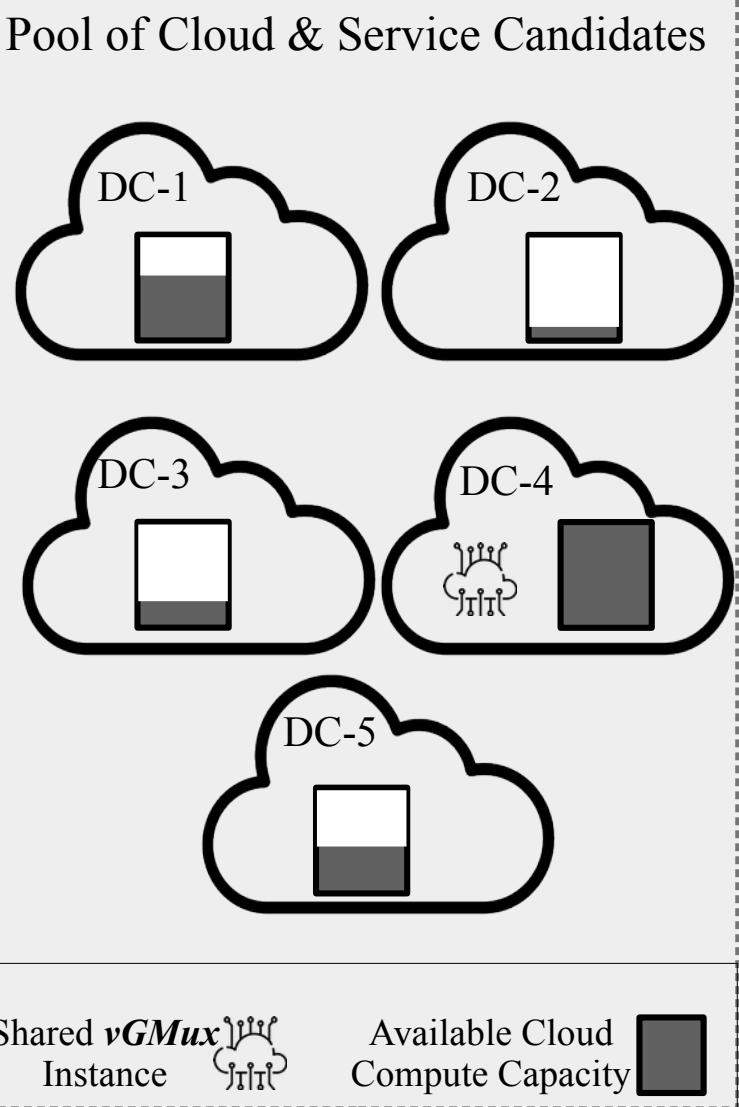
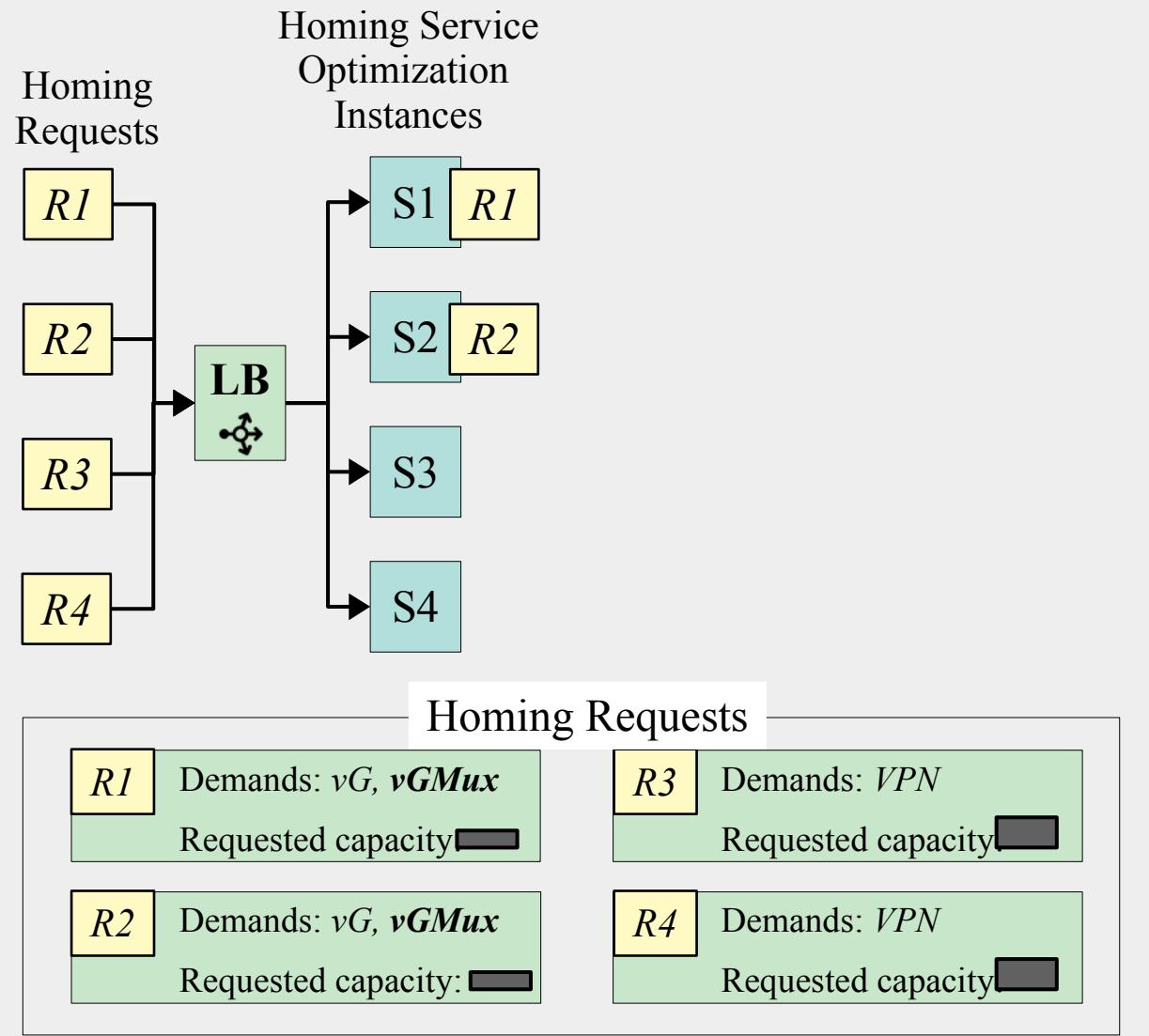


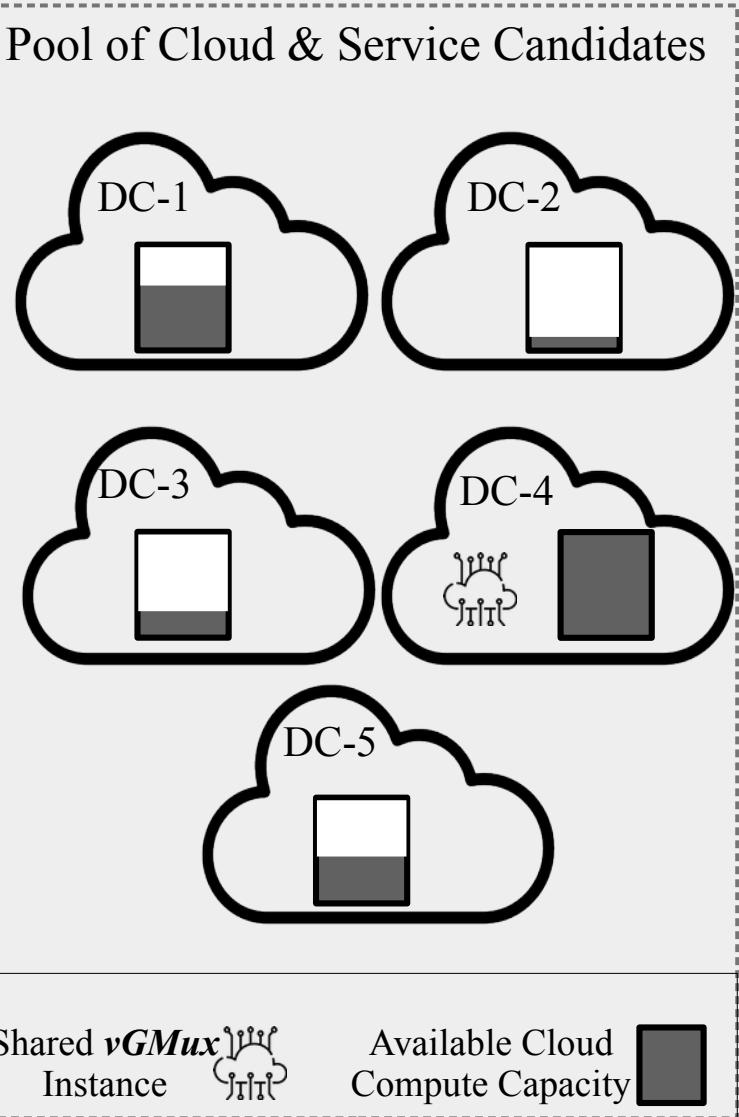
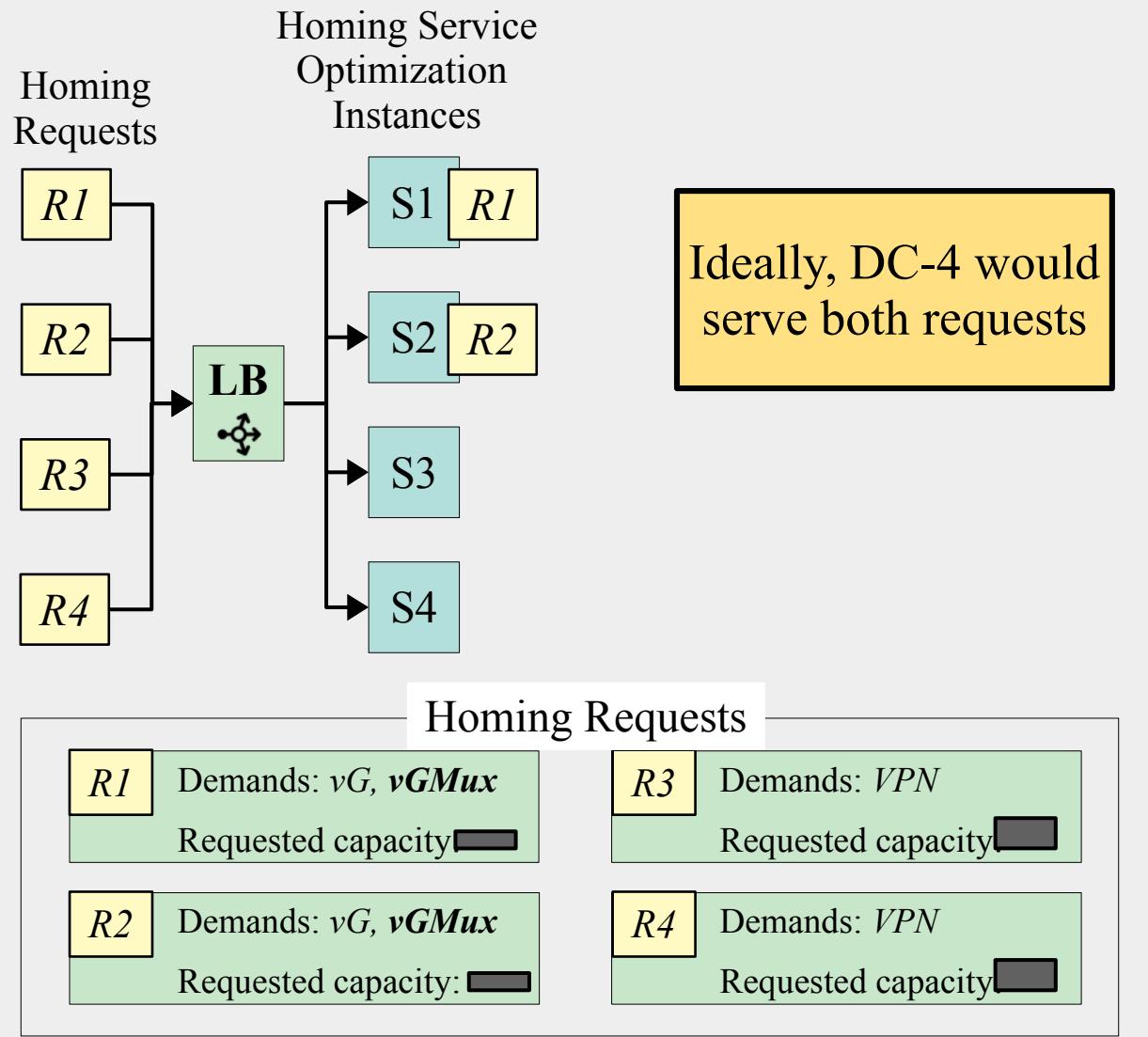




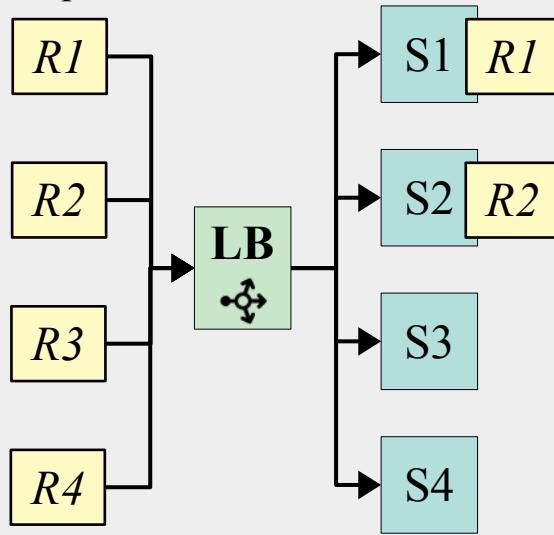








Homing Service Optimization Instances



Ideally, DC-4 would serve both requests

But it does not have enough capacity

Homing Requests

R1 Demands: vG , $vGMux$

Requested capacity

R2 Demands: vG , $vGMux$

Requested capacity

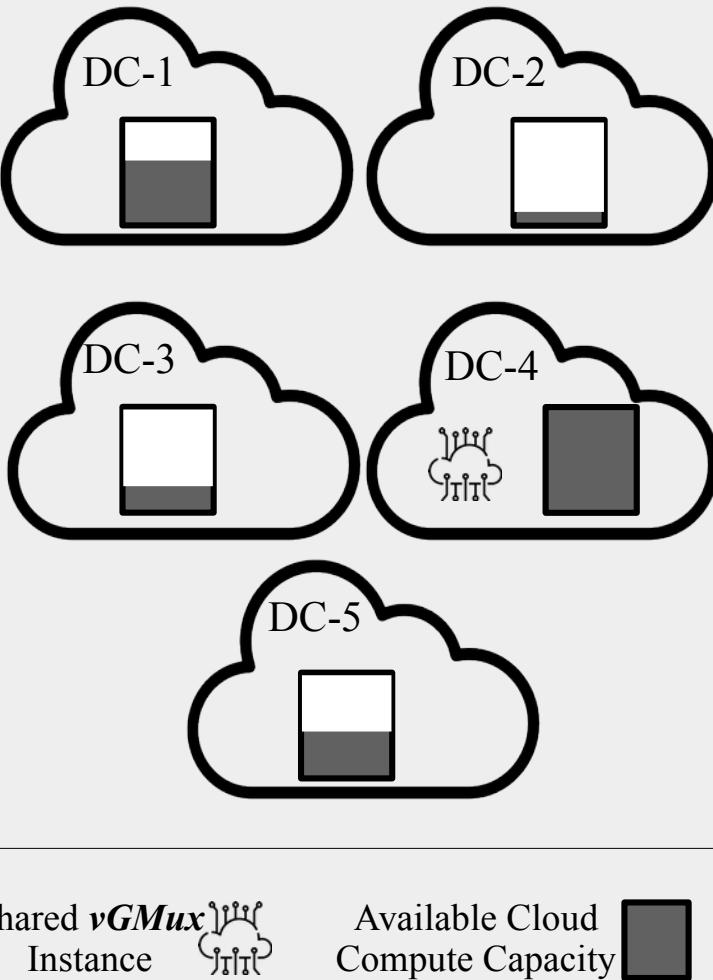
R3 Demands: VPN

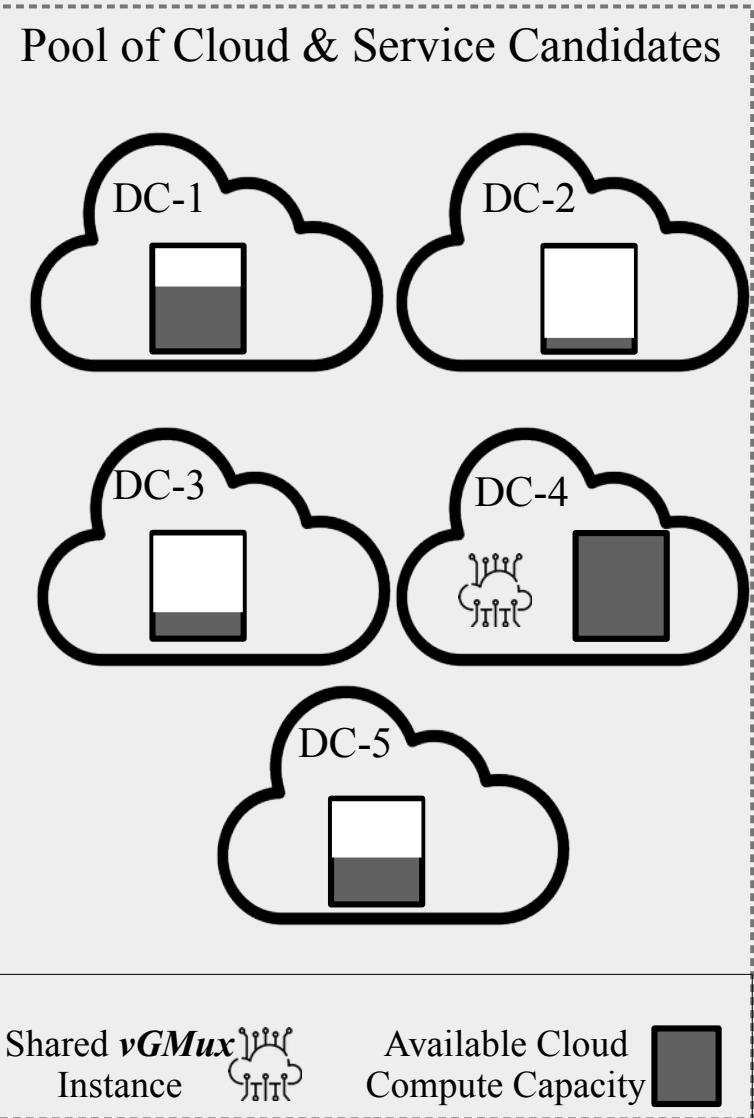
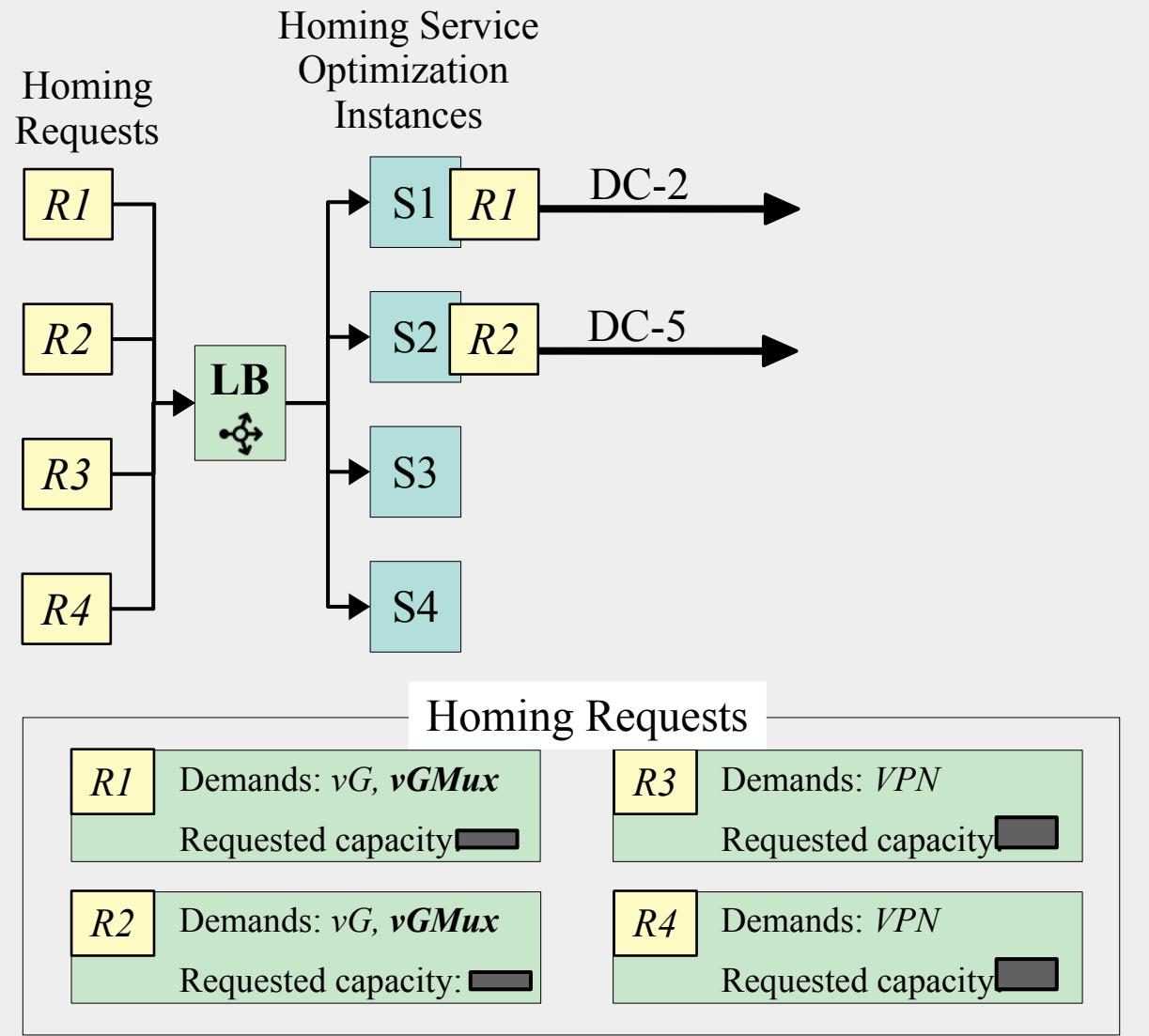
Requested capacity

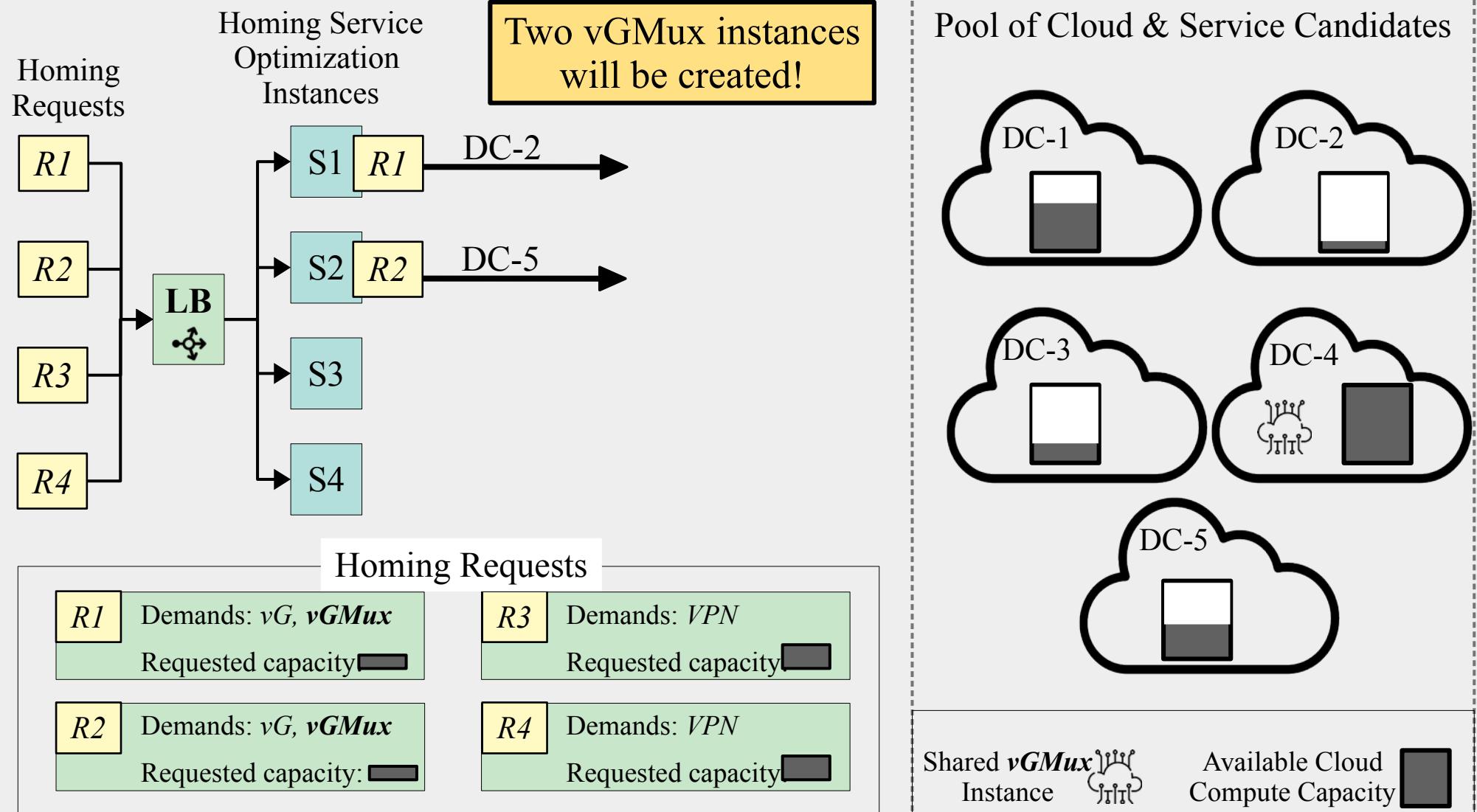
R4 Demands: VPN

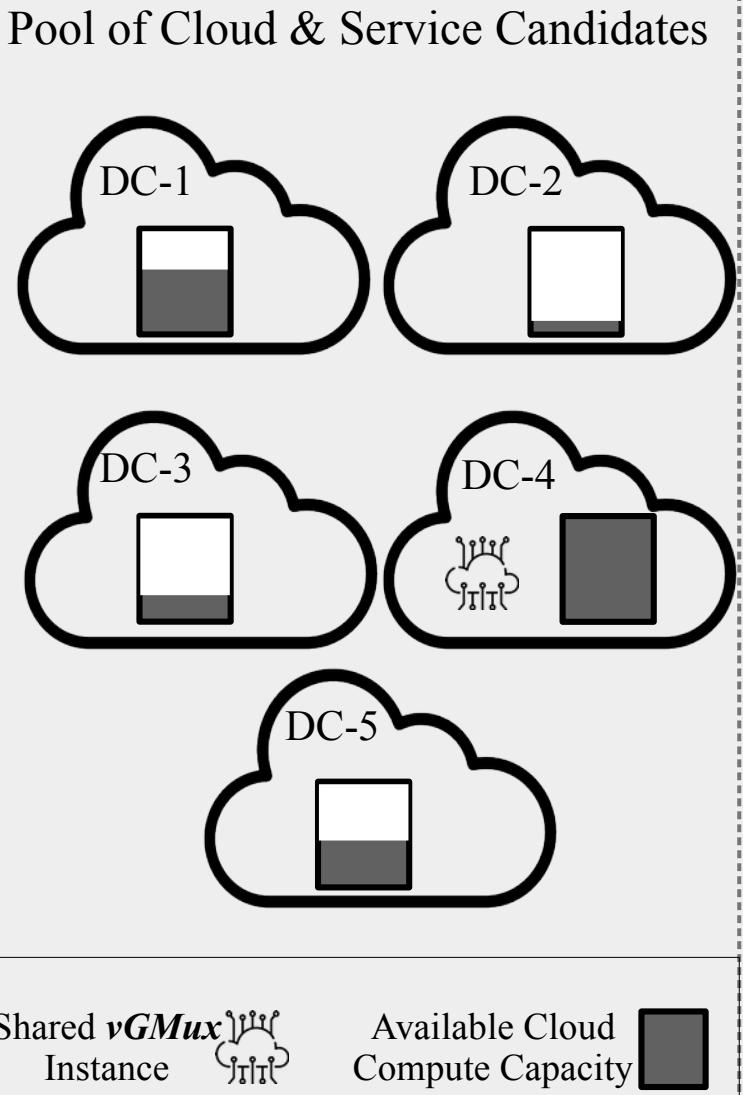
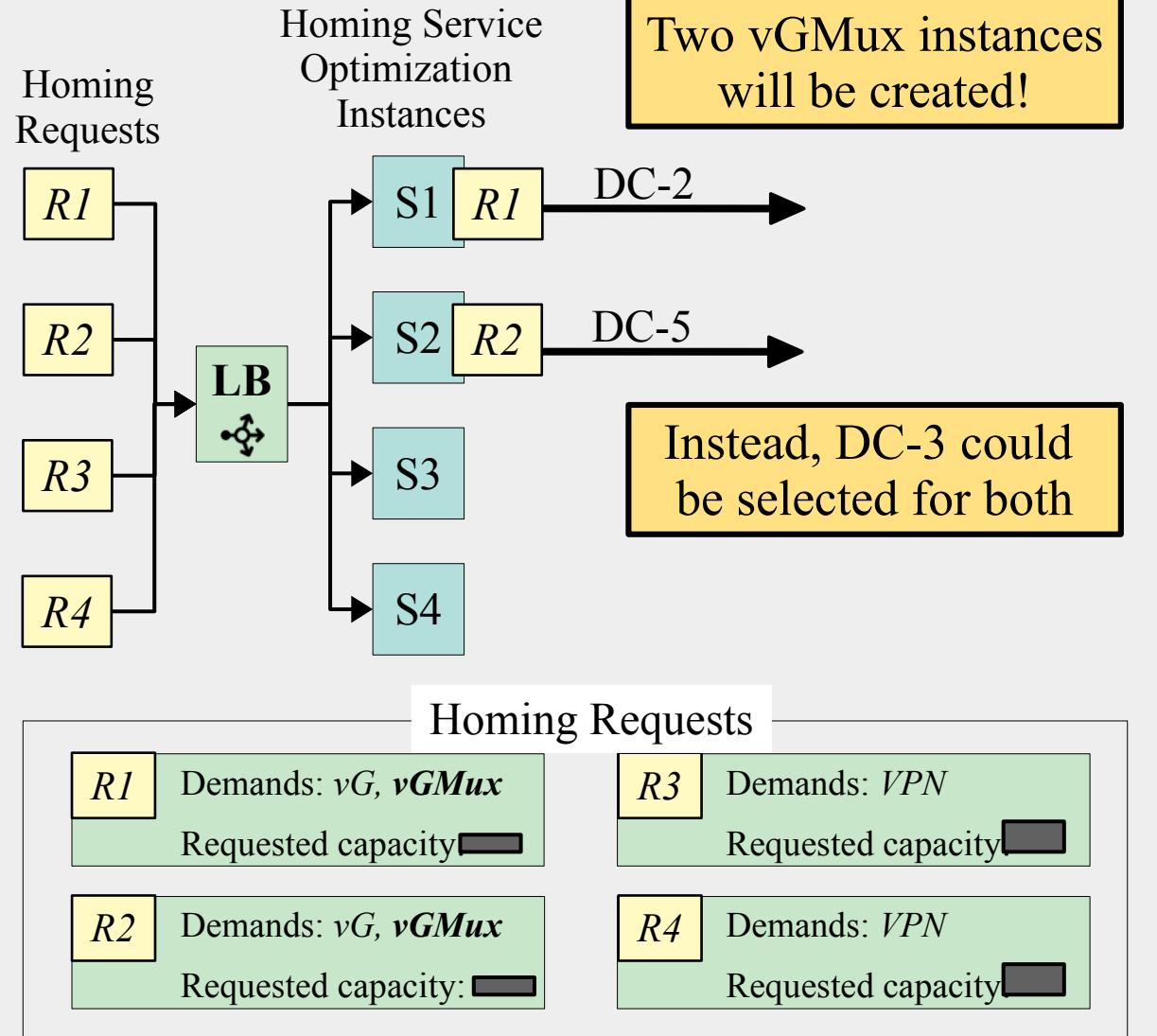
Requested capacity

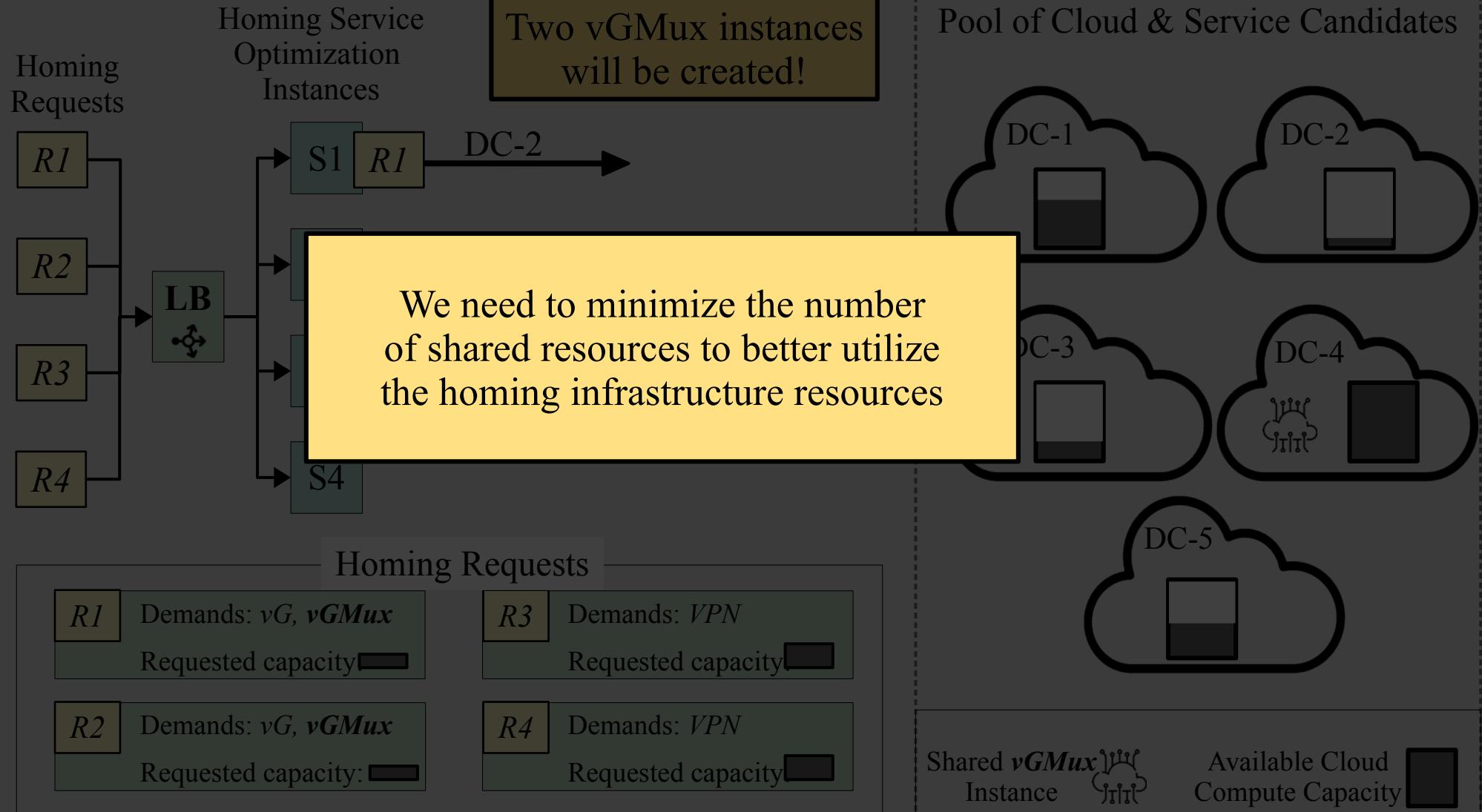
Pool of Cloud & Service Candidates

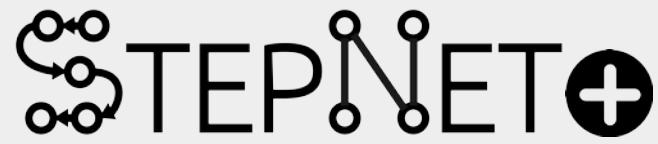




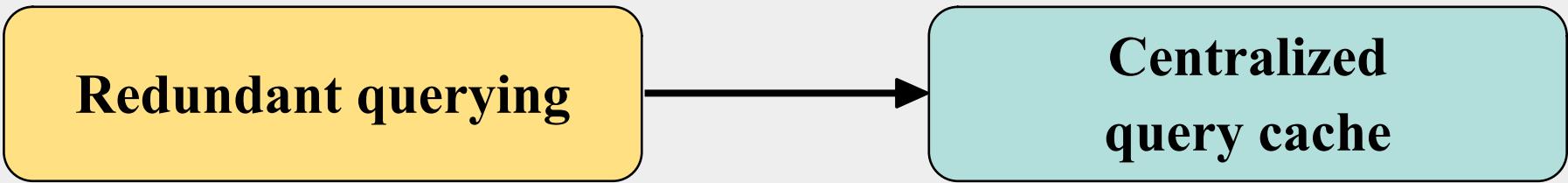
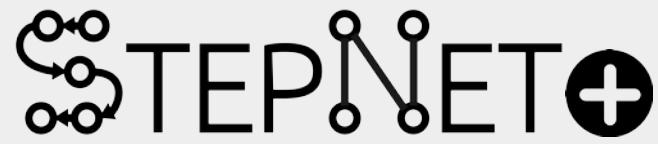


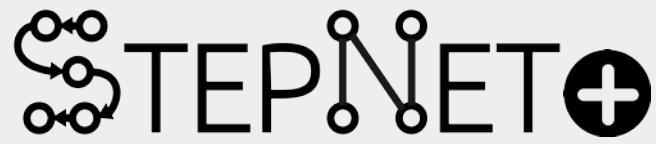






Redundant querying

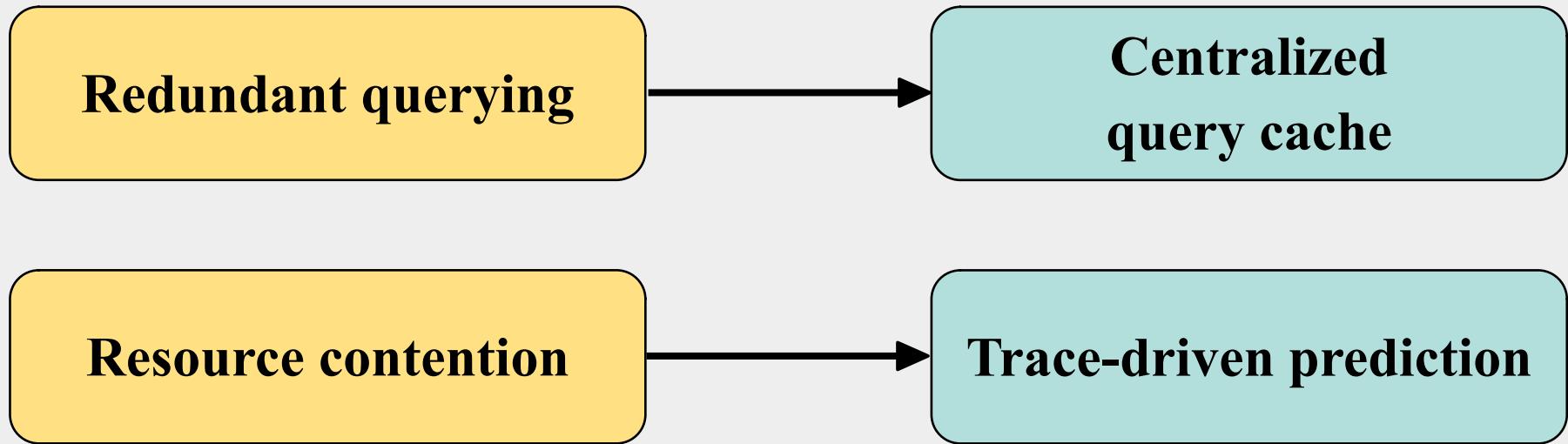




Redundant querying

**Centralized
query cache**

Resource contention



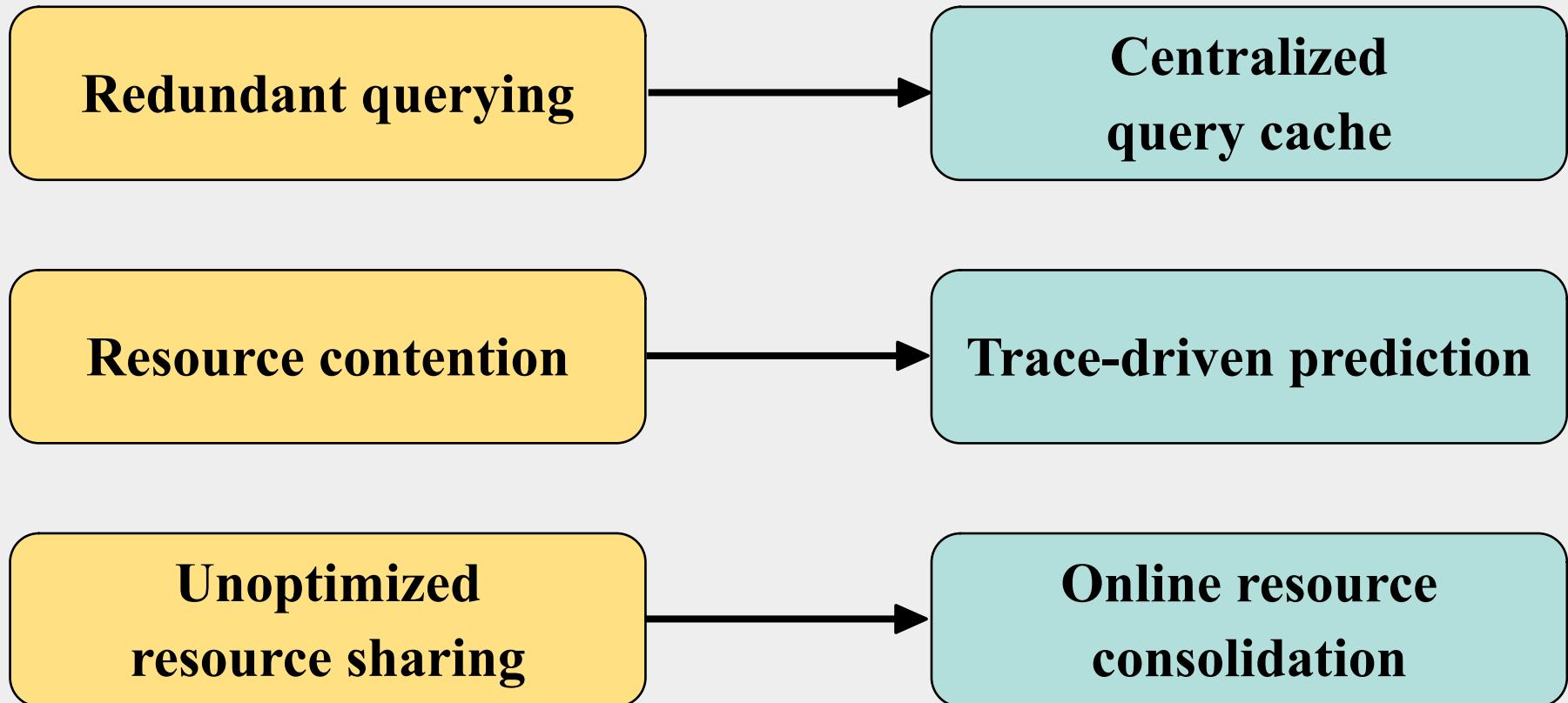
Redundant querying

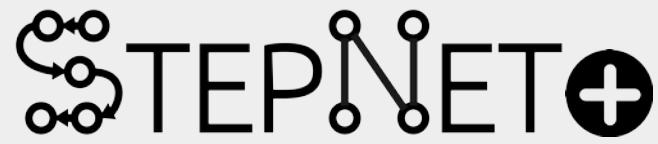
Centralized query cache

Resource contention

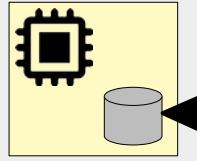
Trace-driven prediction

**Unoptimized
resource sharing**





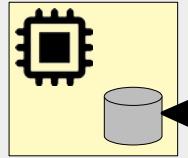
Homing
Optimization
Instances



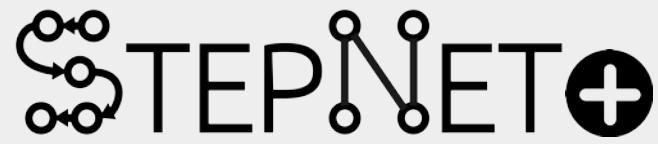
Per-request local
query cache

To avoid information staleness → evict cache entries after solving each request

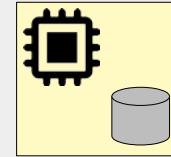
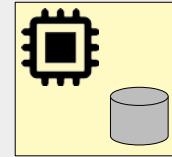
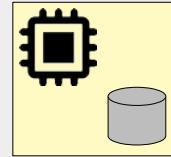
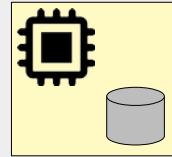
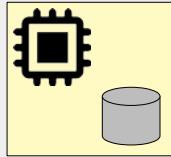
Homing
Optimization
Instances

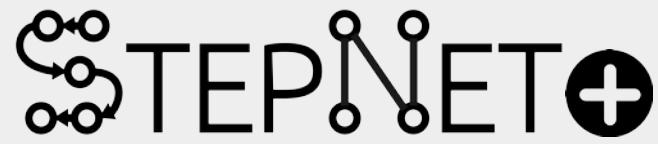


Per-request local
query cache

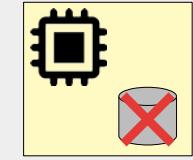
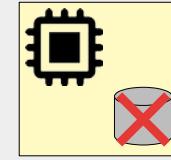
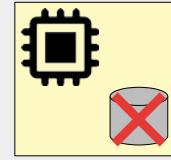
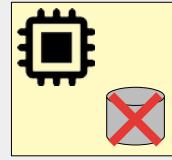
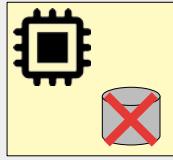


Homing
Optimization
Instances

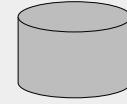




Homing
Optimization
Instances



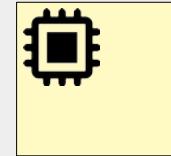
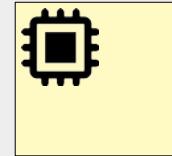
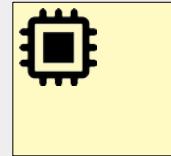
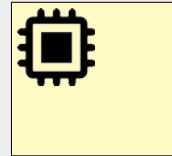
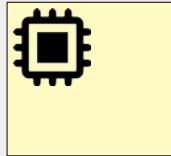
Centralized
query cache



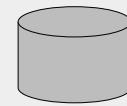


How to avoid stale information now?

Homing
Optimization
Instances



Centralized
query cache



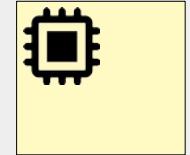
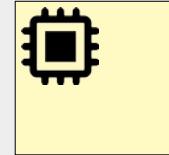
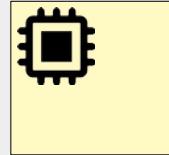
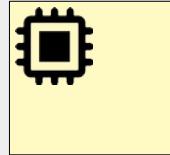
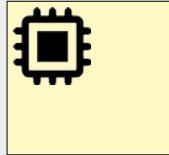


How to avoid stale information now?

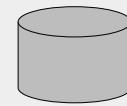


Batch requests and clear cache after each batch

Homing
Optimization
Instances



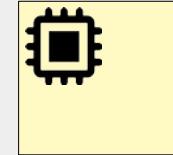
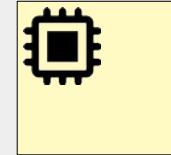
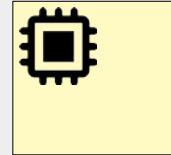
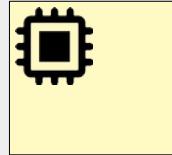
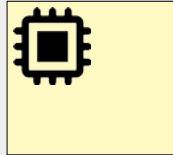
Centralized
query cache



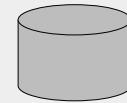


How to address *resource contention*
and *unoptimized resource sharing*?

Homing
Optimization
Instances

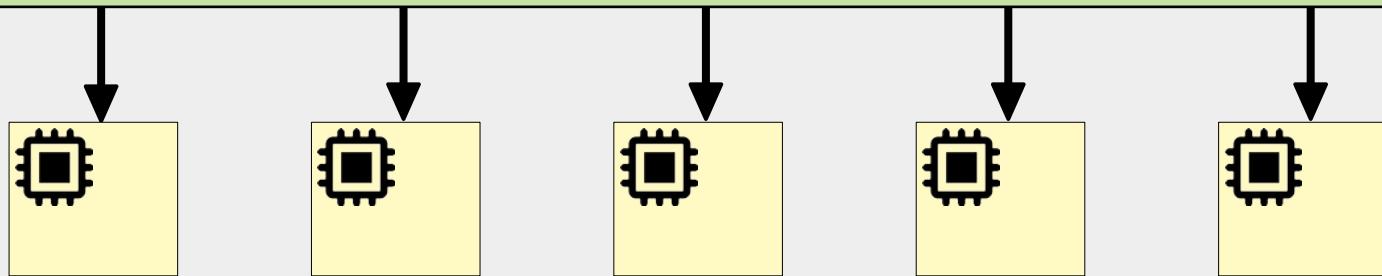


Centralized
query cache

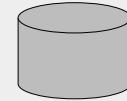


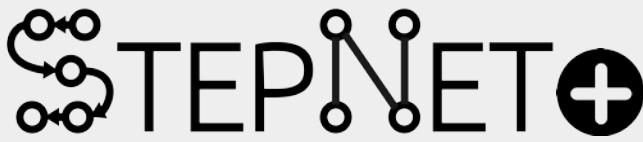
The reason these two problems occur is because homing decisions are left entirely to each local instance

Homing
Optimization
Instances



Centralized
query cache

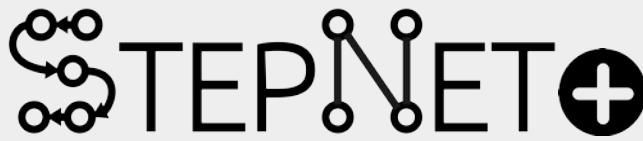




Input:

r : interpreted homing request instance
demands : list of demands (constraints and set of potential candidates are mapped into each demand).
solution_path : currently assigned candidates for demands.

```
1: procedure SOLUTION(r, demands, solution_path)
2:   if demands is empty then
3:     return solution_path
4:   d = demands.pop()
5:   valid_cands = evaluate_constraints(d, solution_path)
6:   sorted_cands = r.obj_func.compute(valid_cands)
7:   best_candidate = sorted_cands.pop()
8:   if best_candidate is null then
9:     //Backtrack: force previous demand to select different candidate
10:    demands.add(d)
11:    return null
12:   solution_path[d] = best_candidate
13:   solution = SOLUTION(r, demands, solution_path)
14:   if solution is null then
15:     remove best_candidate from candidate lists
16:   else
17:     return solution
18:   go to 7
```



Input:

r : interpreted homing request instance
demands : list of demands (constraints and set of potential candidates are mapped into each demand).
solution_path : currently assigned candidates for demands.

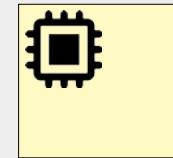
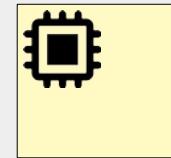
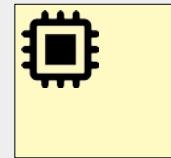
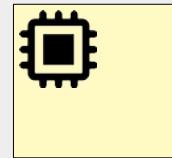
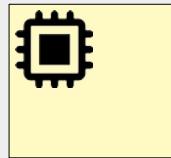
```
1: procedure SOLUTION(r, demands, solution_path)
2:   if demands is empty then
3:     return solution_path
4:   d = demands.pop()
5:   valid_cands = evaluate_constraints(d, solution_path)
6:   sorted_cands = r.obj_func.compute(valid_cands)
7:   best_candidate = sorted_cands.pop()
8:   if best_candidate is null then
9:     //Backtrack: force previous demand to select different candidate
10:    demands.add(d)
11:    return null
12:    solution_path[d] = best_candidate
13:    solution = SOLUTION(r, demands, solution_path)
14:    if solution is null then
15:      remove best_candidate from candidate lists
16:    else
17:      return solution
18:    go to 7
```



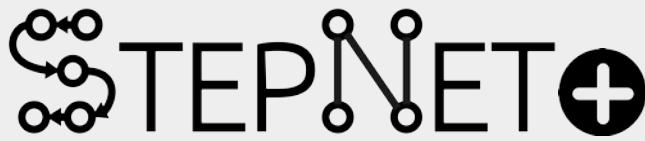
Homing Service Controller

Coordinate
Homing
Decisions

Homing
Optimization
Instances



Centralized
query cache



Input:

r : interpreted homing request instance
 $demands$: list of demands (constraints and set of potential candidates are mapped into each demand).
 $solution_path$: currently assigned candidates for demands.

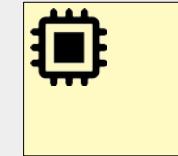
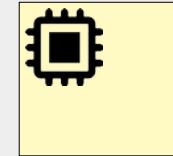
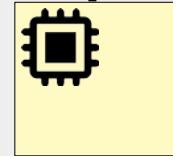
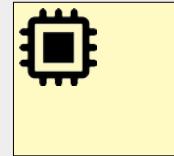
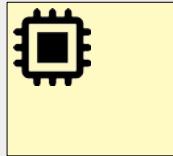
```
1: procedure SOLUTION( $r, demands, solution\_path$ )
2:   if  $demands$  is empty then
3:     return  $solution\_path$ 
4:    $d = demands.pop()$ 
5:    $valid\_cands = evaluate\_constraints(d)$ 
6:    $sorted\_cands = r.obj\_func.compute(valid\_cands)$ 
7:    $best\_candidate = sorted\_cands.pop()$ 
8:   if  $best\_candidate$  is null then
9:     //Backtrack: force previous demand to select different candidate
10:     $demands.add(d)$ 
11:    return null
12:    $solution\_path[d] = best\_candidate$ 
13:    $solution = \text{SOLUTION}(r, demands, solution\_path)$ 
14:   if  $solution$  is null then
15:     remove  $best\_candidate$  from candidate lists
16:   else
17:     return  $solution$ 
18:   go to 7
```

Offload decision to controller via API:
validateCandidate()



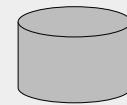
Homing Service Controller

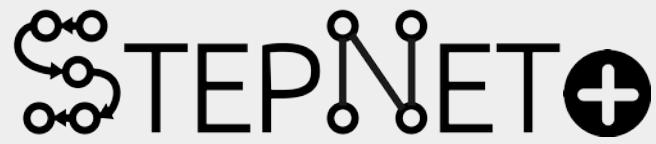
Homing
Optimization
Instances



*Can I assign
candidate to
demand?*

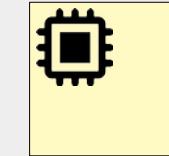
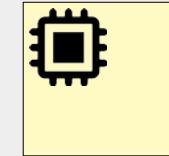
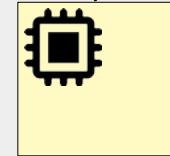
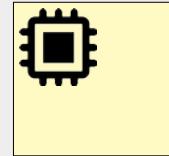
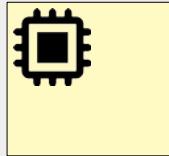
Centralized
query cache





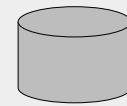
Homing Service Controller

Homing
Optimization
Instances



Yes

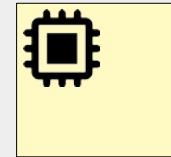
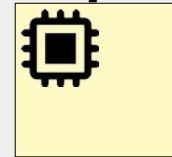
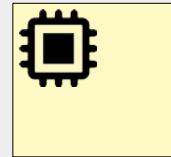
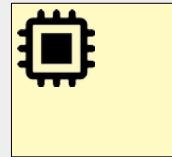
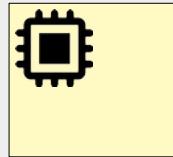
Centralized
query cache





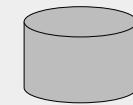
Homing Service Controller

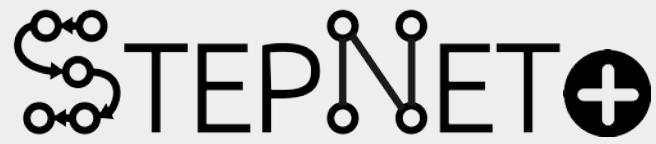
Homing
Optimization
Instances



*Can I assign
candidate to
demand?*

Centralized
query cache

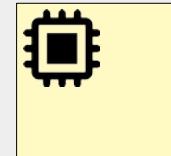
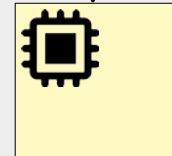
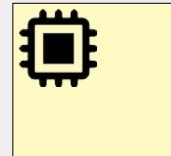
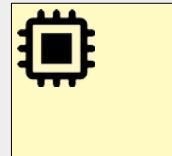
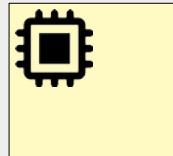




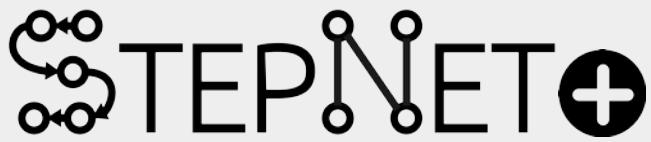
Homing Service Controller

*e.g., to prevent
resource contention*

Homing
Optimization
Instances



Centralized
query cache



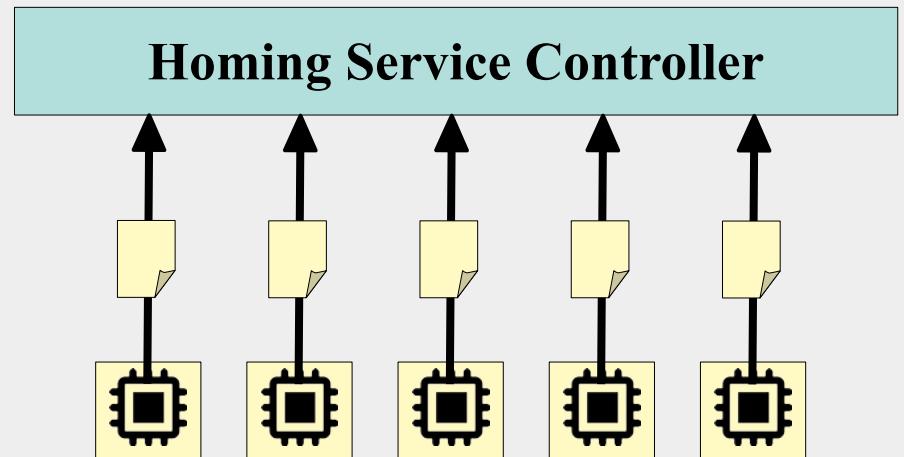
Homing Service Controller



So, how does the controller prevent resource contention or consolidate shared resources?

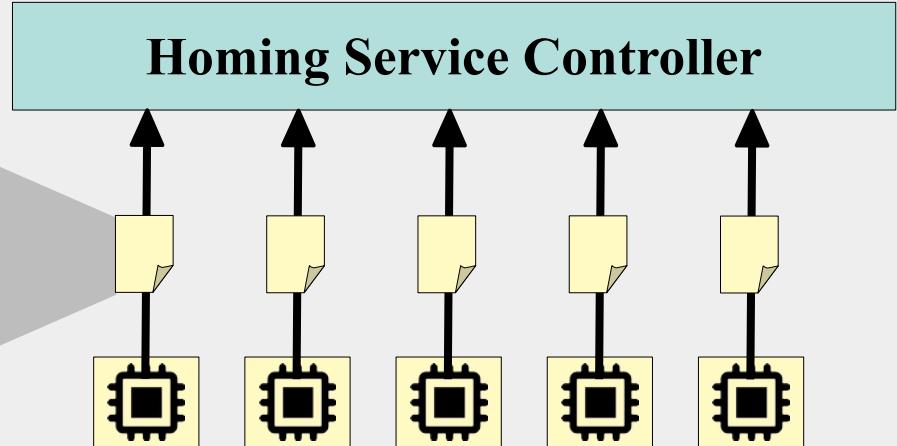


Coordinated Homing Decisions



Coordinated Homing Decisions

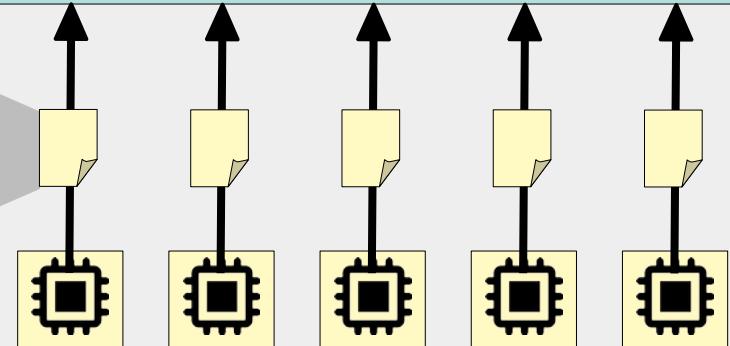
```
validateCandidate()  
Parameters:  
    demand  
    best_cand  
    feasible_cands
```



Coordinated Homing Decisions

```
validateCandidate()  
Parameters:  
demand  
best_cand  
feasible_cands
```

Homing Service Controller

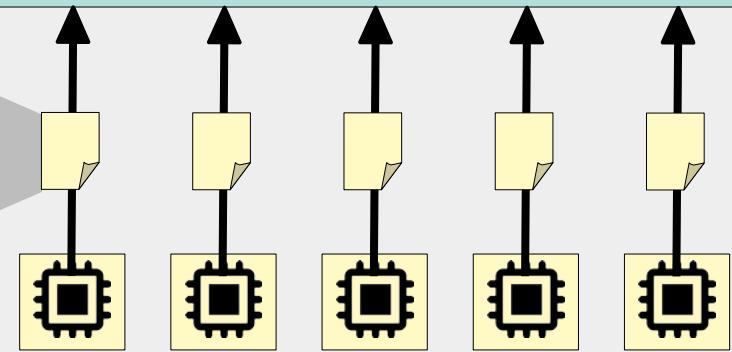


The current demand being processed

Coordinated Homing Decisions

```
validateCandidate()  
Parameters:  
    demand  
    best_cand  
    feasible_cands
```

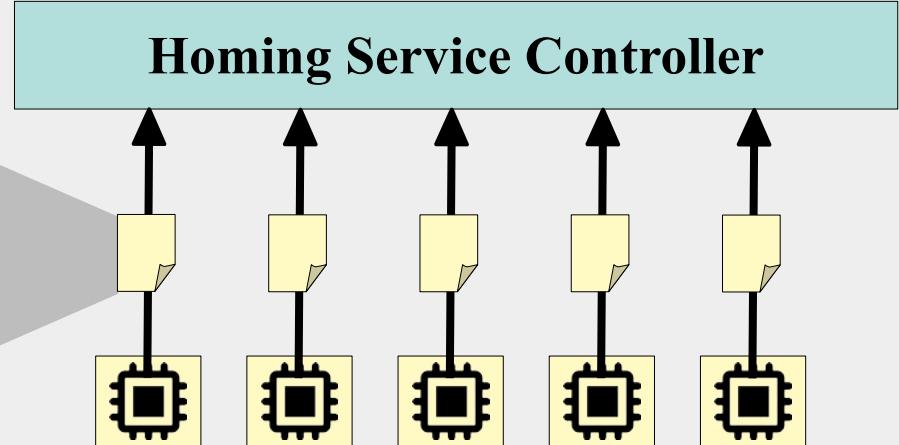
Homing Service Controller



The best candidate this instance
wants to assign to demand

Coordinated Homing Decisions

```
validateCandidate()  
Parameters:  
    demand  
    best_cand  
feasible_cands
```



List of feasible candidates for this demand

Coordinated Homing Decisions

```

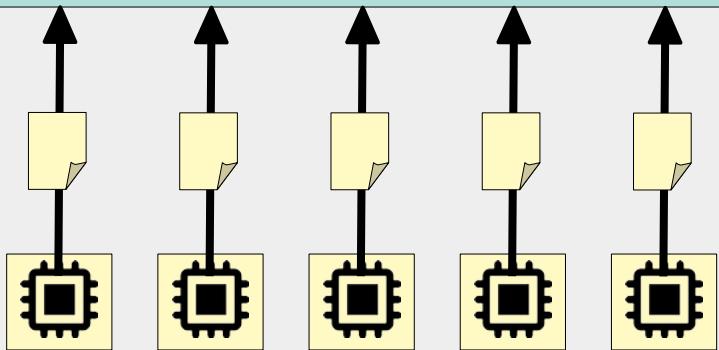
def validateCandidate(demand, best_cand, feasible_cands):
    if demand.type("shared_resource") and \
        best_cand is not service_instance:
        wait_for_other_requests(timeout)
        # for all other in-flight requests (in current batch)
        # that are interested in this shared_resource or until timeout
        chosen_candidate = consolidate_shared_resources(feasible_cands)
        # consolidates shared resources with other
        # requests in the current batch
        return chosen_candidate

    elif best_cand in requested_candidates:
        predictor = predict(best_cand)
        # predicts whether this candidate can
        # also accommodate this demand
        if predictor:
            return best_cand
        else:
            feasible_cands.remove(best_cand)
            return feasible_cands

    else:
        requested_candidates.increment(best_cand)
        return best_cand

```

Homing Service Controller



If no other requests are interested in same candidate, then allow it (no resource contention)

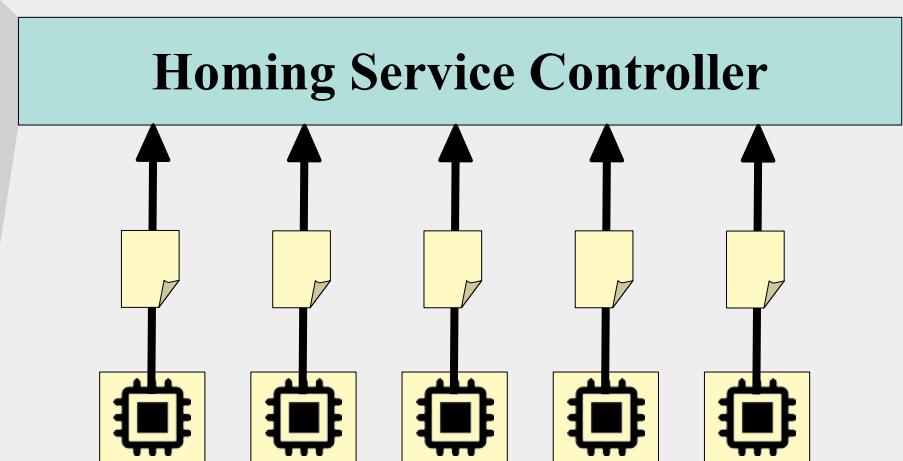
Coordinated Homing Decisions

```

def validateCandidate(demand, best_cand, feasible_cands):
    if demand.type("shared_resource") and \
        best_cand is not service_instance:
        wait_for_other_requests(timeout)
        # for all other in-flight requests (in current batch)
        # that are interested in this shared_resource or until timeout
    chosen_candidate = consolidate_shared_resources(feasible_cands)
        # consolidates shared resources with other
        # requests in the current batch
    return chosen_candidate

    elif best_cand in requested_candidates:
        predictor = predict(best_cand)
        # predicts whether this candidate can
        # also accommodate this demand
        if predictor:
            return best_cand
        else:
            feasible_cands.remove(best_cand)
            return feasible_cands
    else:
        requested_candidates.increment(best_cand)
        return best_cand

```



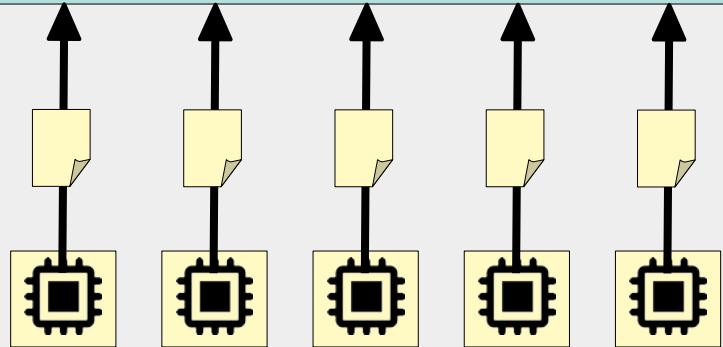
If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

Trace-driven prediction for resource contention

requested_resources	
cand-id	requests

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

Trace-driven prediction for resource contention

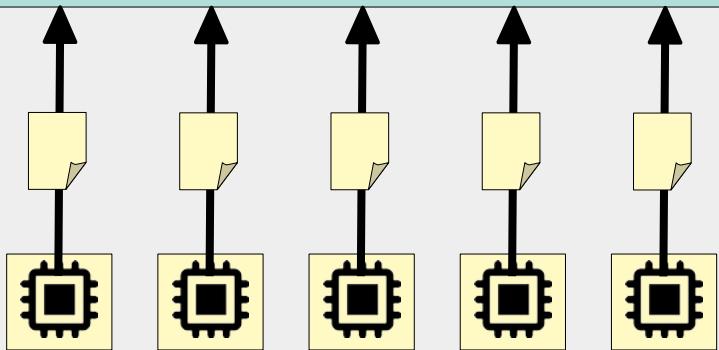
Request (R1)

best_cand: DC-1

requested_resources

cand-id	requests

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

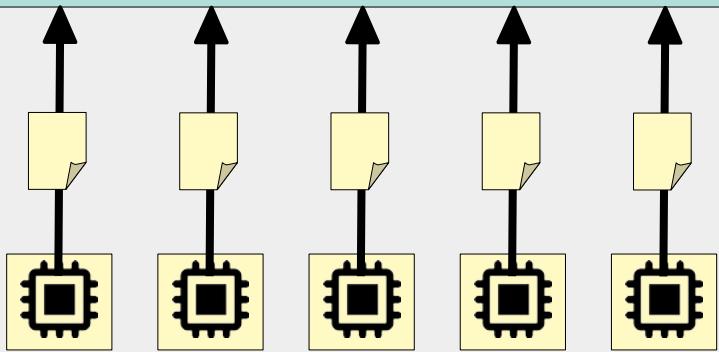
Trace-driven prediction for resource contention

Request (R1)

best_cand: DC-1

requested_resources	
cand-id	requests
	

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

Trace-driven prediction for resource contention

Request (R1)

best_cand: DC-1

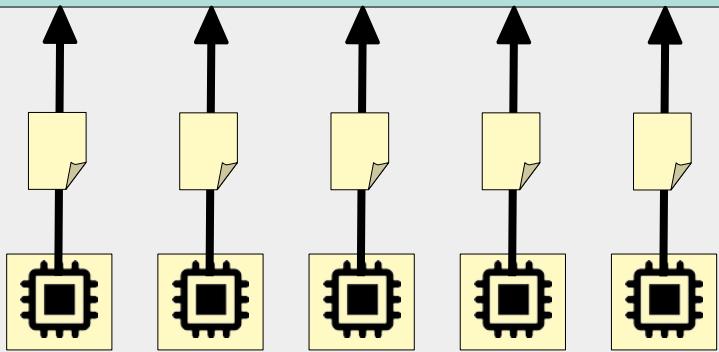
requested_resources

cand-id	requests
DC-1	[R1]

Response to R1

Yes, you can use DC-1

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

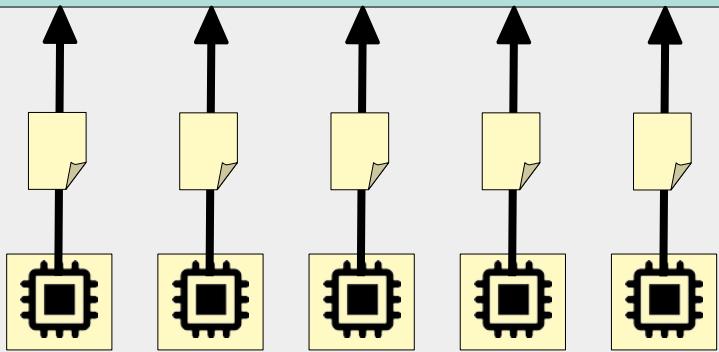
Trace-driven prediction for resource contention

requested_resources	
cand-id	requests
DC-1	[R1]

Request (R2)

best_cand: DC-1

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

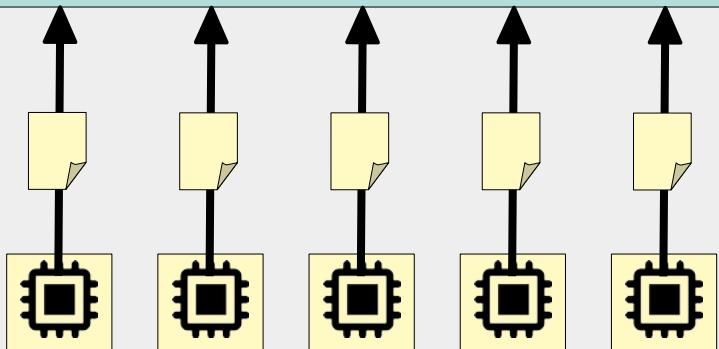
Trace-driven prediction for resource contention

Request (R2)

best_cand: DC-1

requested_resources	
cand-id	requests
DC-1	[R1]

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

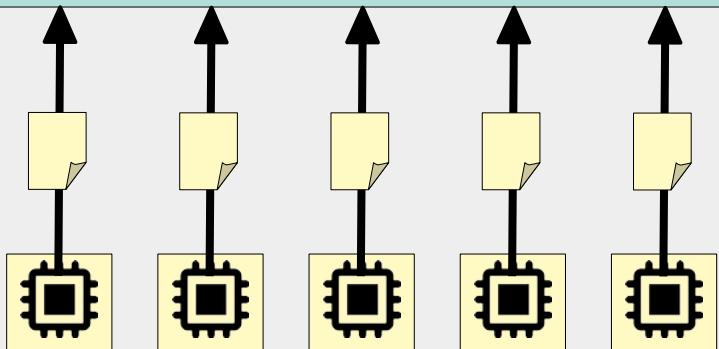
Trace-driven prediction for resource contention

Request (R2)
best_cand: DC-1

requested_resources	
cand-id	requests
DC-1	[R1]

Look at past logs for
DC-1 queries & predict

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

Trace-driven prediction for resource contention

Request (R2)

best_cand: DC-1

requested_resources	
cand-id	requests
DC-1	[R1]

Look at past logs for
DC-1 queries & predict

e.g., if 70% of capacity responses are “yes”
for DC-1 in past 2 days, then predict “yes”

If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

Coordinated Homing Decisions

Trace-driven prediction for resource contention

Request (R2)

best_cand: DC-1

requested_resources	
cand-id	requests
DC-1	[R1]

Look at past logs for
DC-1 queries & predict

e.g., if **70%** of capacity responses are “yes”
for DC-1 in past 2 days, then predict “yes”

Increase prediction threshold as more
requests accumulate for this candidate

If there are other requests interested in this
candidate, then *predict* whether this candidate
can be used to home more than one request

Coordinated Homing Decisions

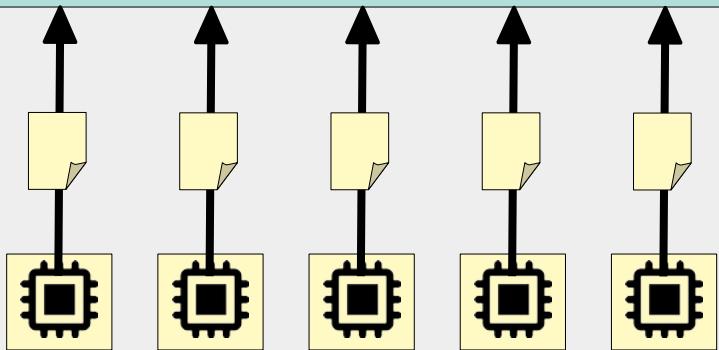
Trace-driven prediction for resource contention

Request (R2)
best_cand: DC-1

requested_resources	
cand-id	requests
DC-1	[R1]

Remove requests that
are provisioned (or fail)

Homing Service Controller



If there are other requests interested in this candidate, then *predict* whether this candidate can be used to home more than one request

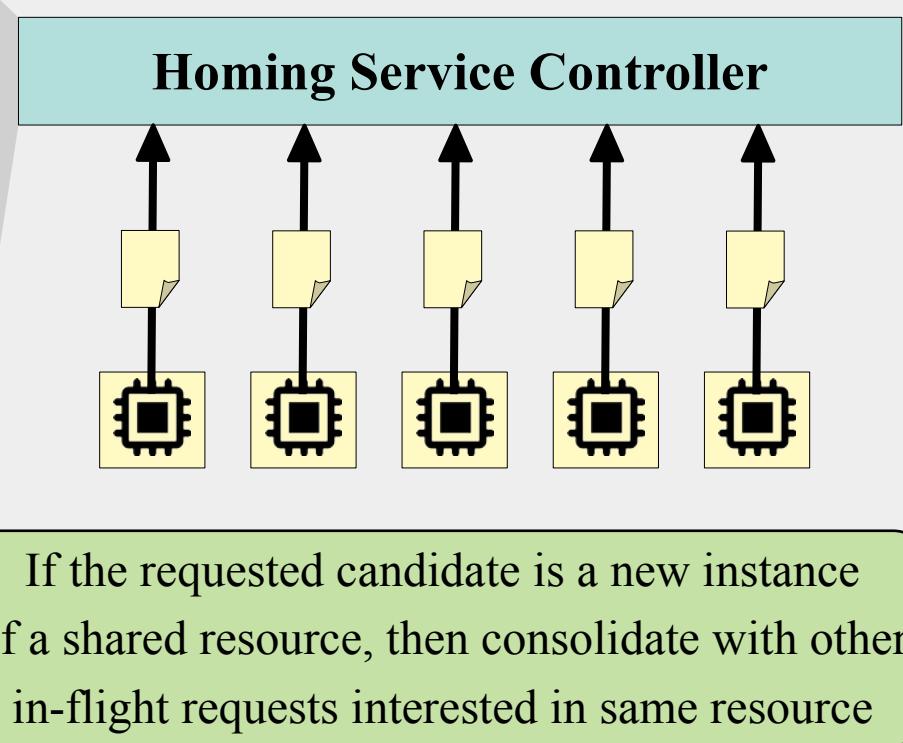
Coordinated Homing Decisions

```

def validateCandidate(demand, best_cand, feasible_cands):
    if demand.type("shared_resource") and \
        best_cand is not service_instance:
        wait_for_other_requests(timeout)
        # for all other in-flight requests (in current batch)
        # that are interested in this shared_resource or until timeout
        chosen_candidate = consolidate_shared_resources(feasible_cands)
        # consolidates shared resources with other
        # requests in the current batch
        return chosen_candidate

    elif best_cand in requested_candidates:
        predictor = predict(best_cand)
        # predicts whether this candidate can
        # also accommodate this demand
        if predictor:
            return best_cand
        else:
            feasible_cands.remove(best_cand)
            return feasible_cands
    else:
        requested_candidates.increment(best_cand)
        return best_cand

```



Coordinated Homing Decisions

Online consolidation of shared resources

R1

Demand: vGMux

best_cand: DC-2

feas_cands: [DC-3, DC-5]

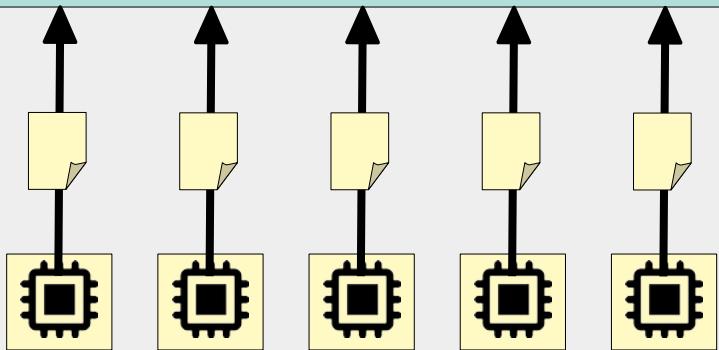
R2

Demand: vGMux

best_cand: DC-3

feas_cands: [DC-1, DC-5]

Homing Service Controller



If the requested candidate is a new instance of a shared resource, then consolidate with other in-flight requests interested in same resource

Coordinated Homing Decisions

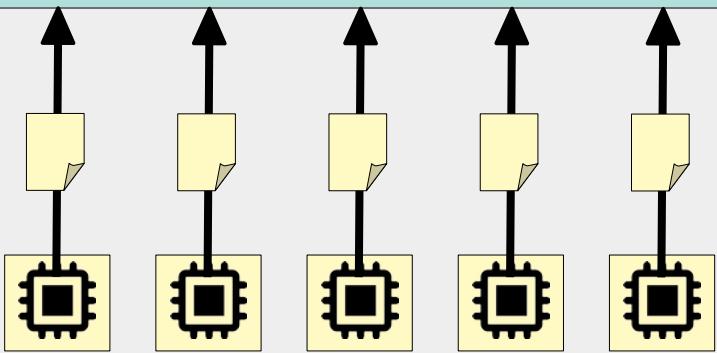
Online consolidation of shared resources

R1
Demand: vGMux
best_cand: DC-2
feas_cands: [DC-3, DC-5]

R2
Demand: vGMux
best_cand: DC-3
feas_cands: [DC-1, DC-5]

We want to minimize number
of new vGMux instances

Homing Service Controller



If the requested candidate is a new instance
of a shared resource, then consolidate with other
in-flight requests interested in same resource

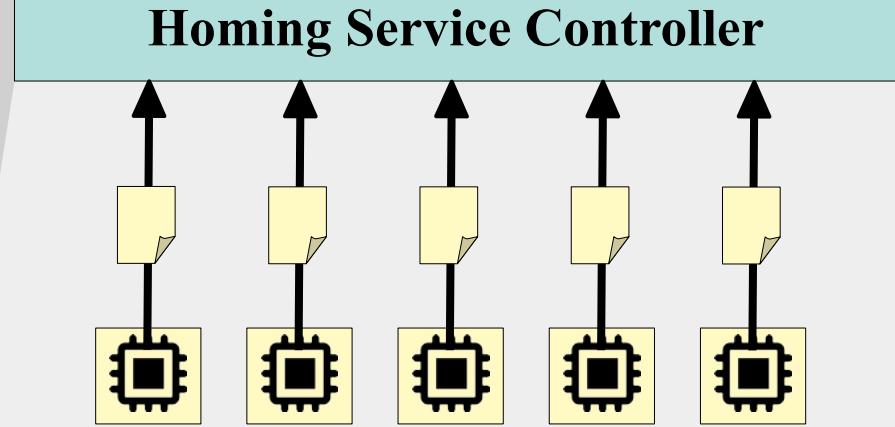
Coordinated Homing Decisions

Online consolidation of shared resources

R1
Demand: vGMux
best_cand: DC-2
feas_cands: [DC-3, **DC-5**]

Consolidate using candidates in feas_cands

R2
Demand: vGMux
best_cand: DC-3
feas_cands: [DC-1, **DC-5**]



If the requested candidate is a new instance of a shared resource, then consolidate with other in-flight requests interested in same resource

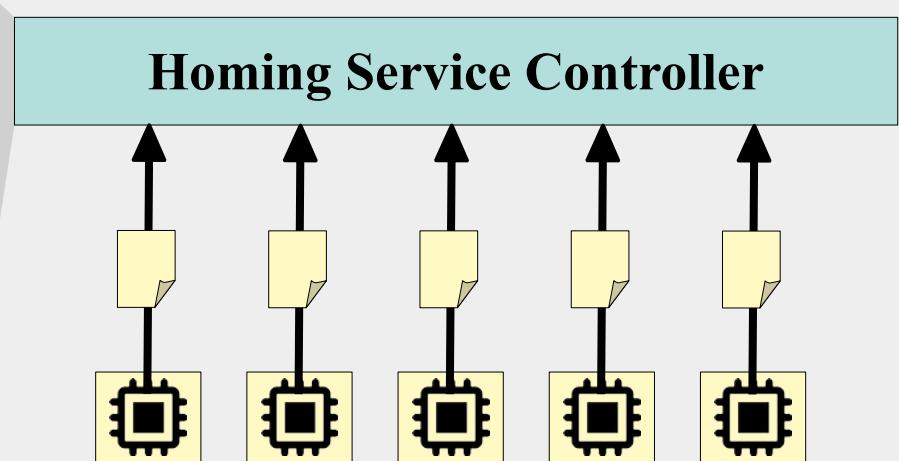
Coordinated Homing Decisions

Online consolidation of shared resources

R1
Demand: vGMux
best_cand: DC-2
feas_cands: [DC-3, **DC-5**]

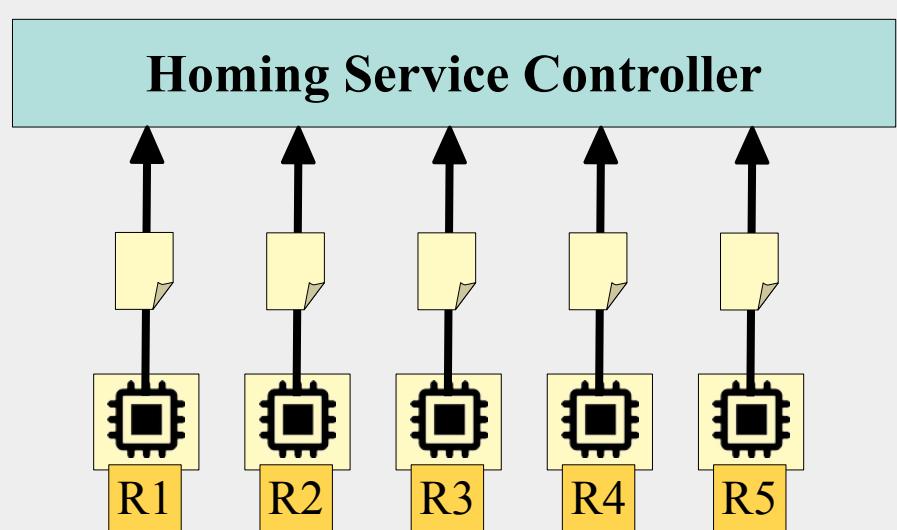
Let R1 and R2 select DC-5 as their *best_cand* for vGMux

R2
Demand: vGMux
best_cand: DC-3
feas_cands: [DC-1, **DC-5**]



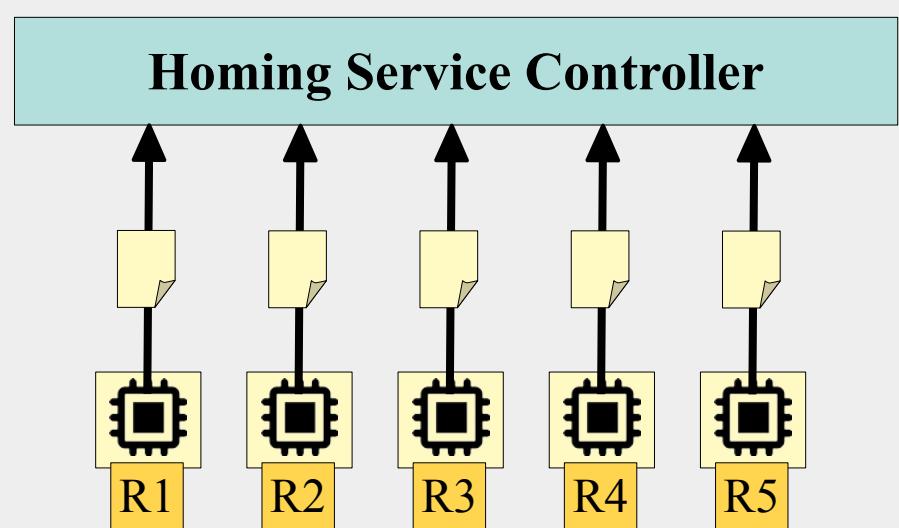
If the requested candidate is a new instance of a shared resource, then consolidate with other in-flight requests interested in same resource

How do we know if these requests are related?



How do we know if these requests are related?

Batching requests as they arrive could undermine benefits of caching as well as consolidation of shared requests

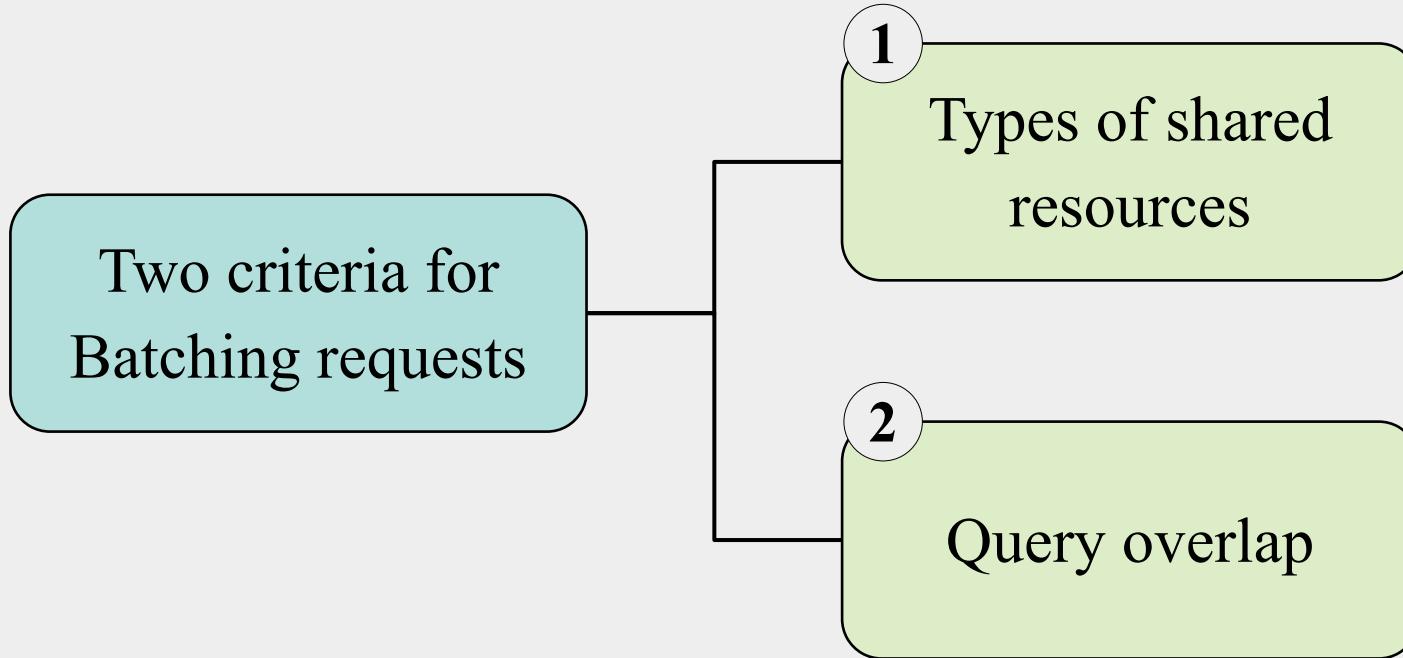




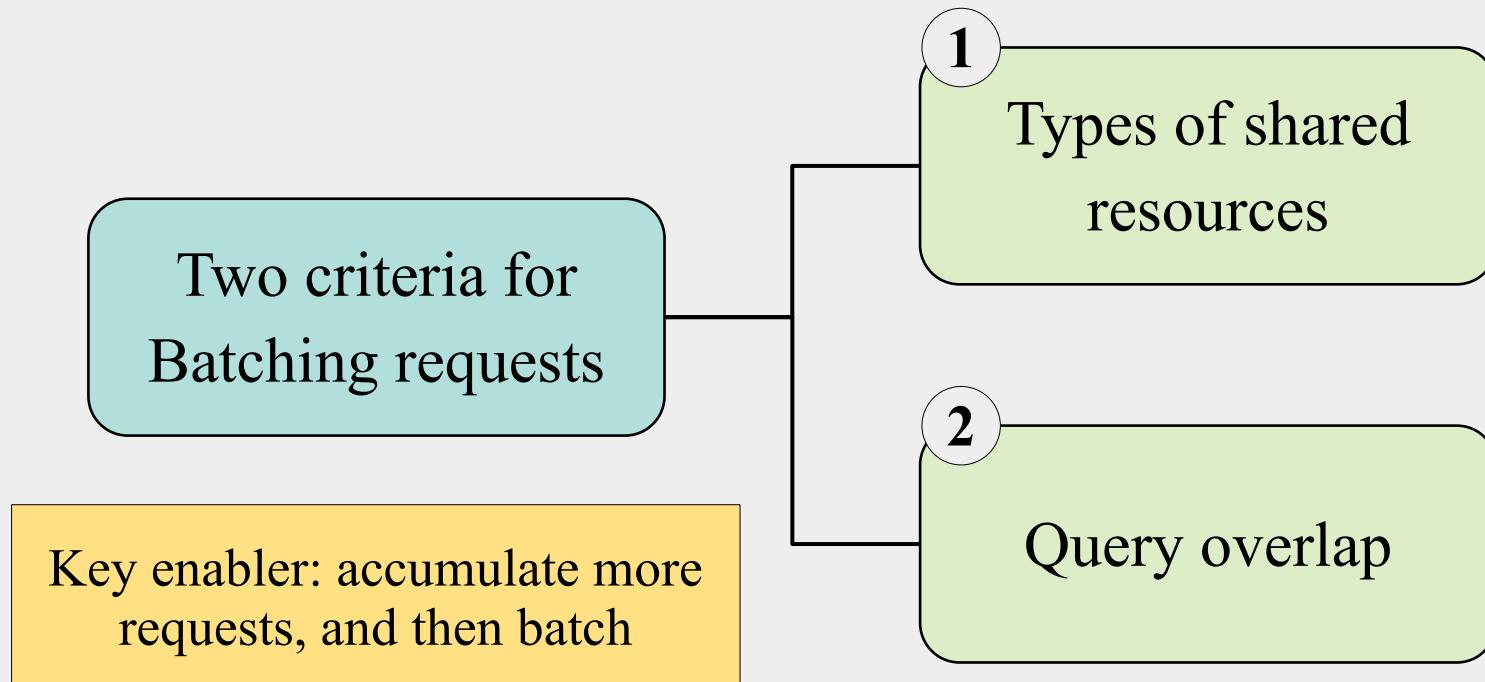
Multi-criteria Batching

Two criteria for
Batching requests

Multi-criteria Batching

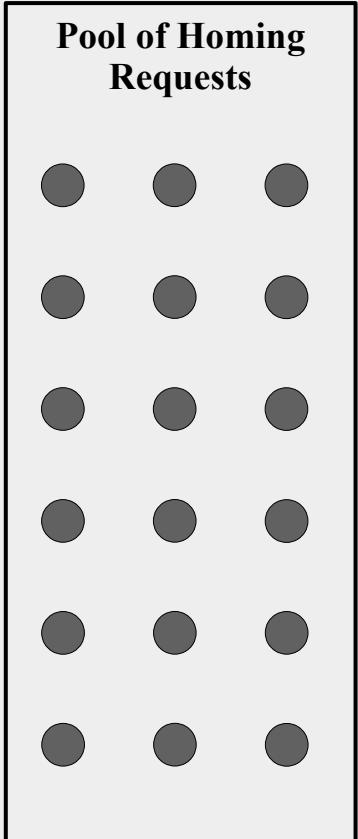


Multi-criteria Batching

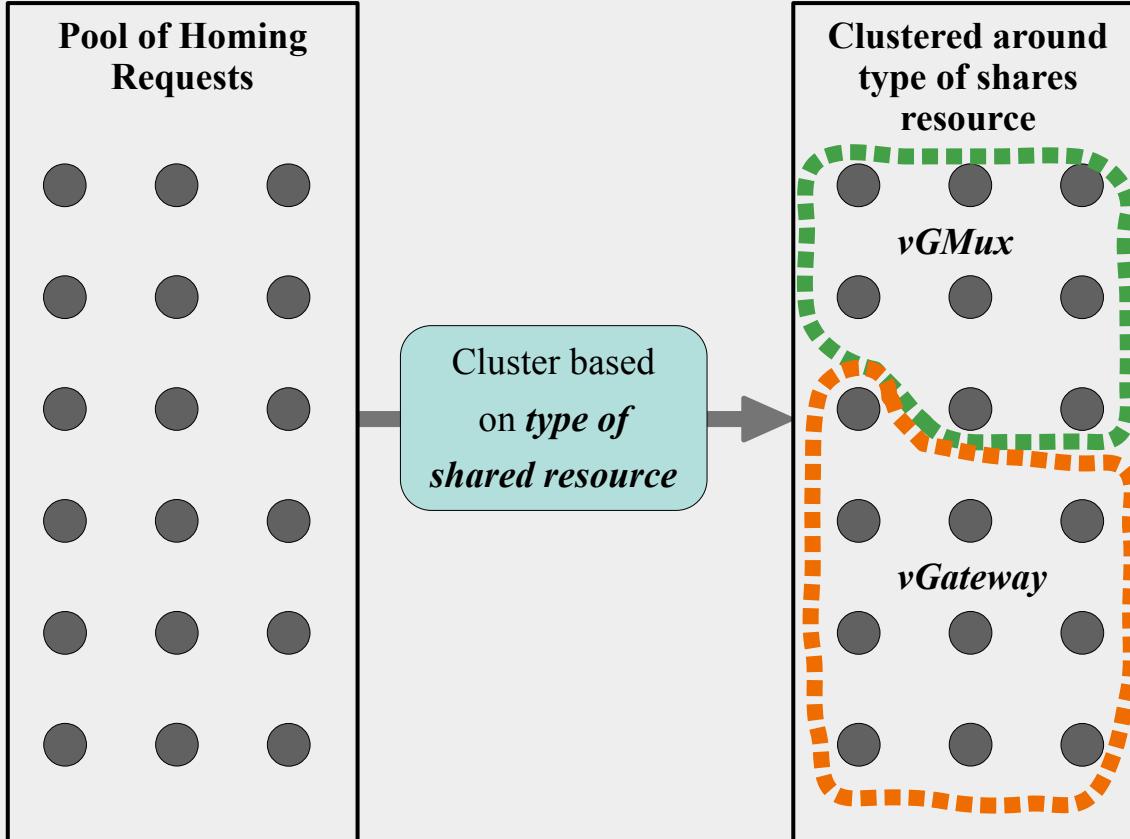




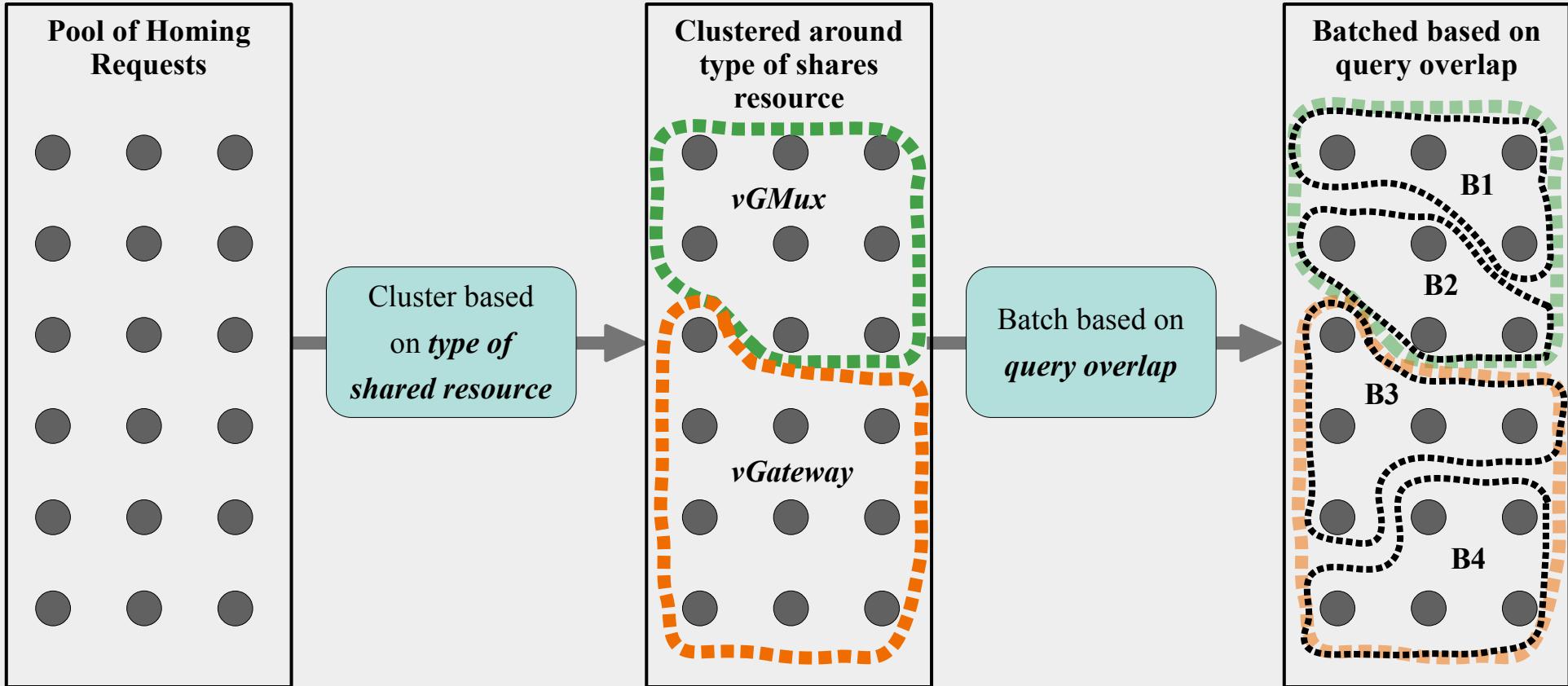
Multi-criteria Batching



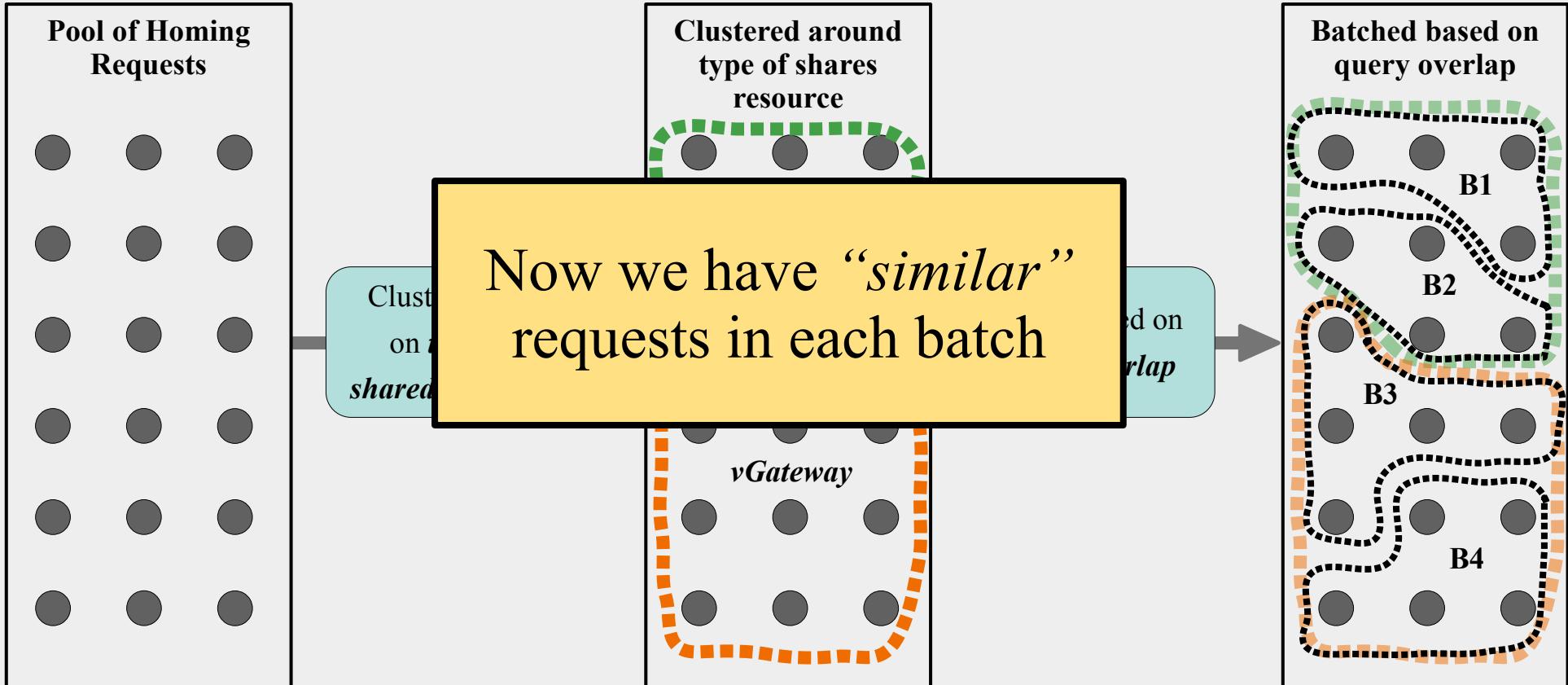
Multi-criteria Batching



Multi-criteria Batching



Multi-criteria Batching





Evaluation

Three approaches

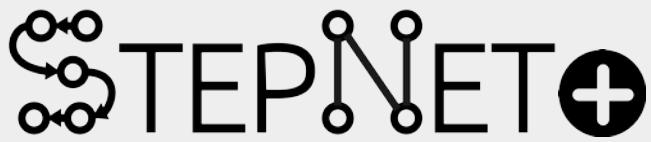


Evaluation

1

Baseline (no batching): STEPNET

Three approaches



Evaluation

Three approaches

1

Baseline (no batching): STEPNET

2

Strawman (time-based batching)



Evaluation

Three approaches

1

Baseline (no batching):

2

Strawman (time-based batching)

3

: multi-criteria batching



Evaluation

Experiment setup

Heuristic optimizer: *BACBF-inc*

Number of requests: 67

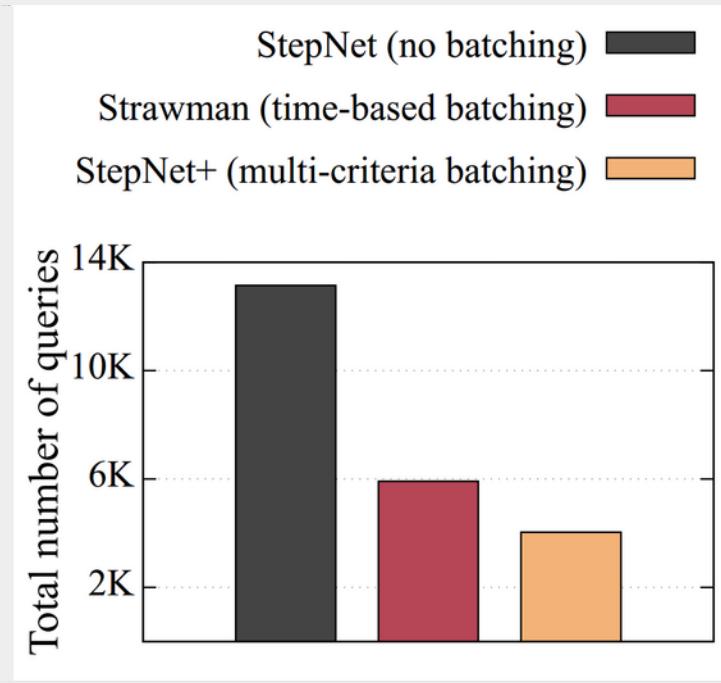
Batching window: 1-hour

Batch size: 5

Baseline uses per-request query cache

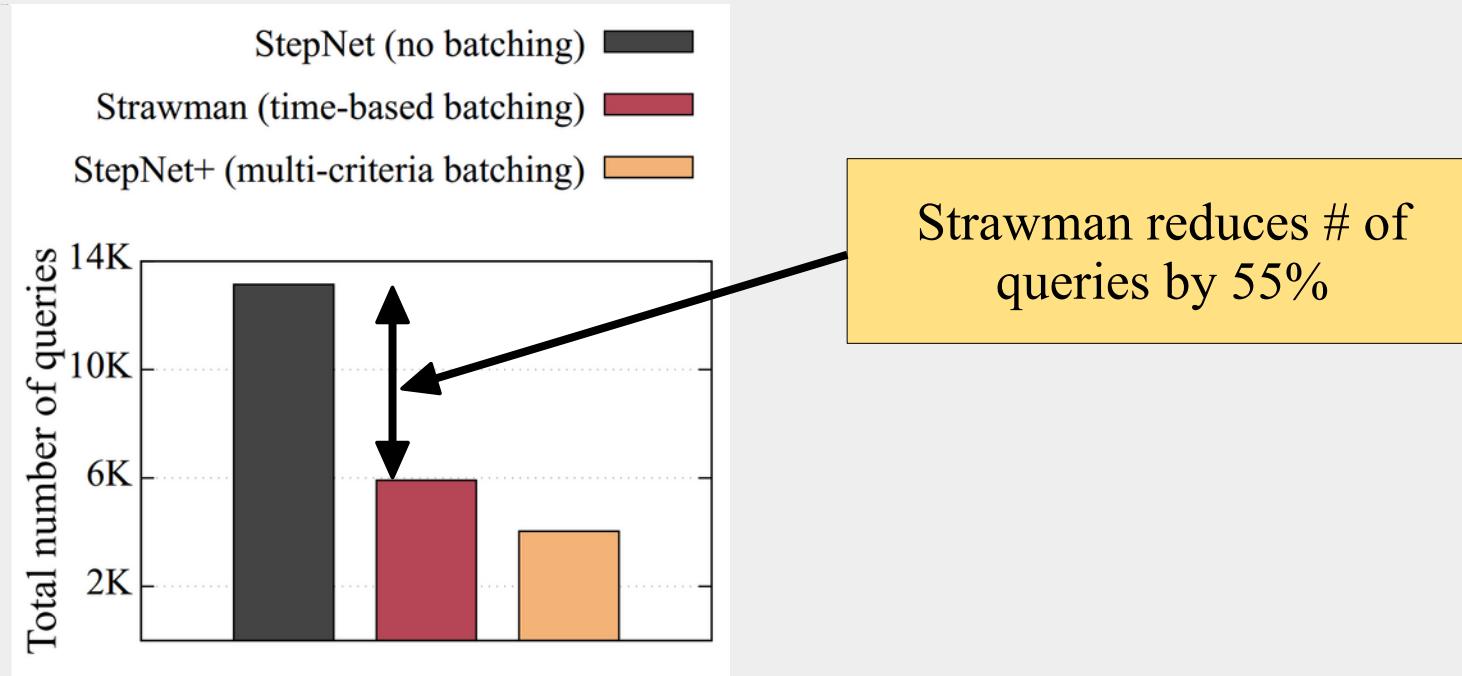
Other two configs use per-batch cache

Evaluation

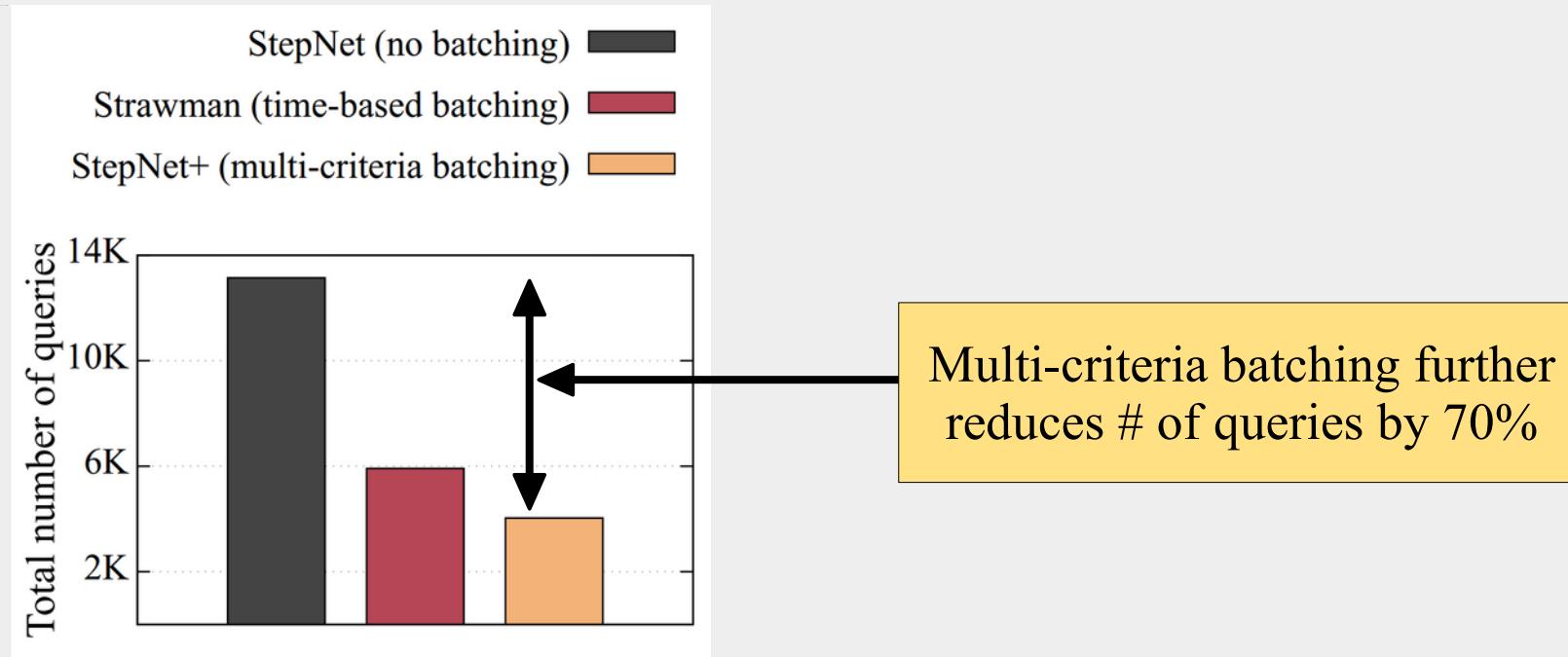


Reducing query redundancy

Evaluation

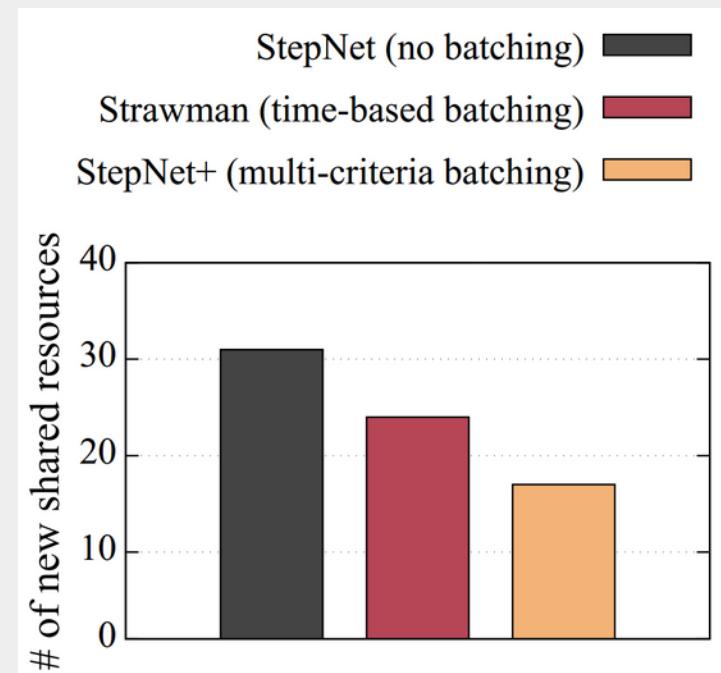


Evaluation



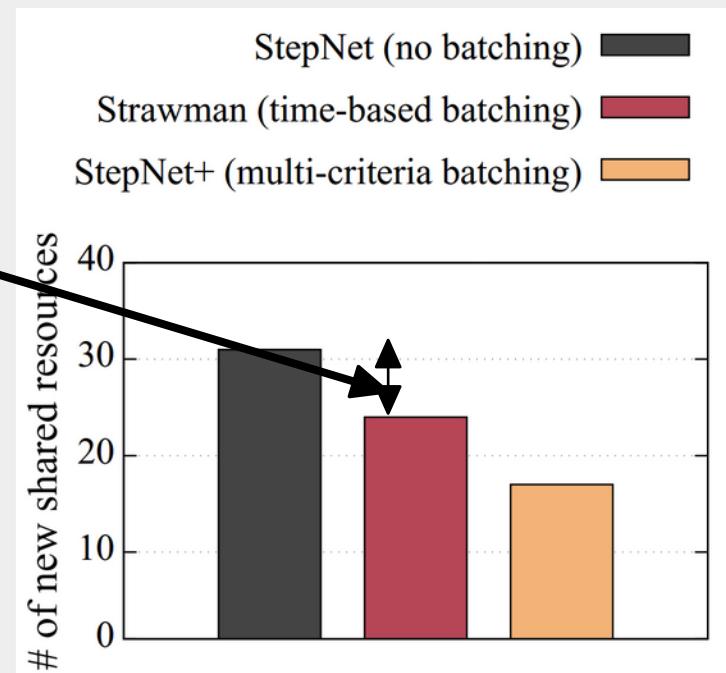
Evaluation

Consolidating shared resources



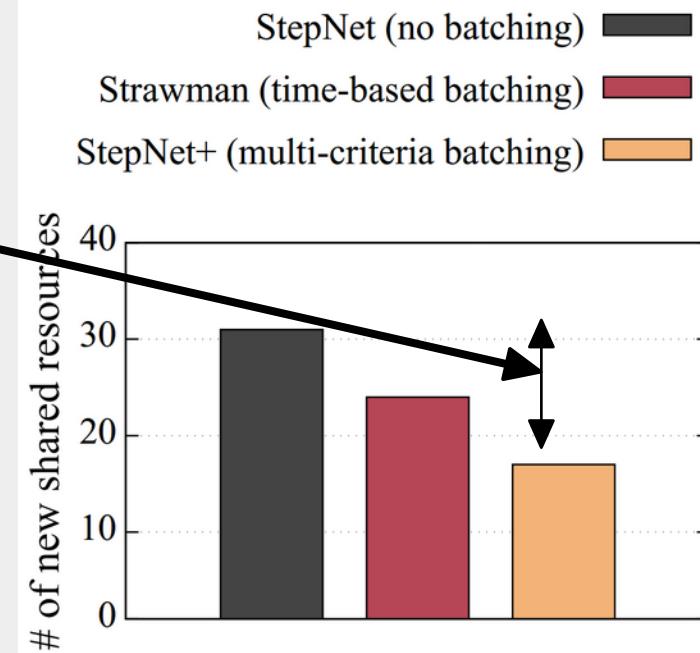
Evaluation

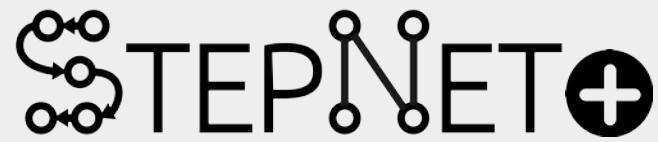
Strawman reduces # of Shared resources by 23%



Evaluation

StepNet+ reduces # of Shared resources by 45%





Evaluation

Preventing resource contention

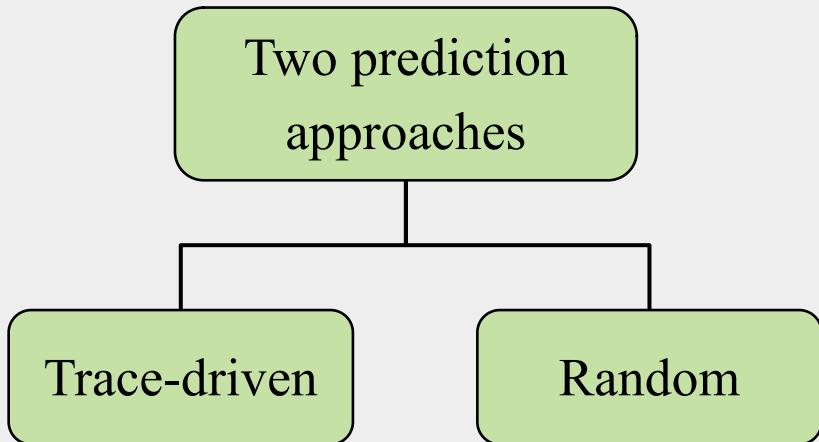


Evaluation

Preventing resource contention

Since the prediction mechanism doesn't depend on requests being batched, we measure the percentage of prevented resource contention across all requests

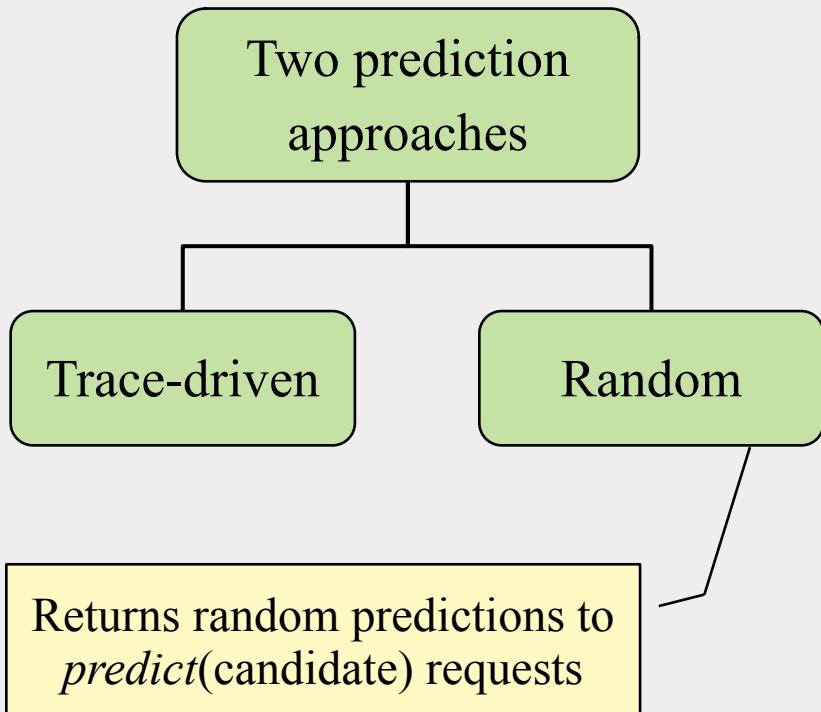
Evaluation



Preventing resource contention

Since the prediction mechanism doesn't depend on requests being batched, we measure the percentage of prevented resource contention across all requests

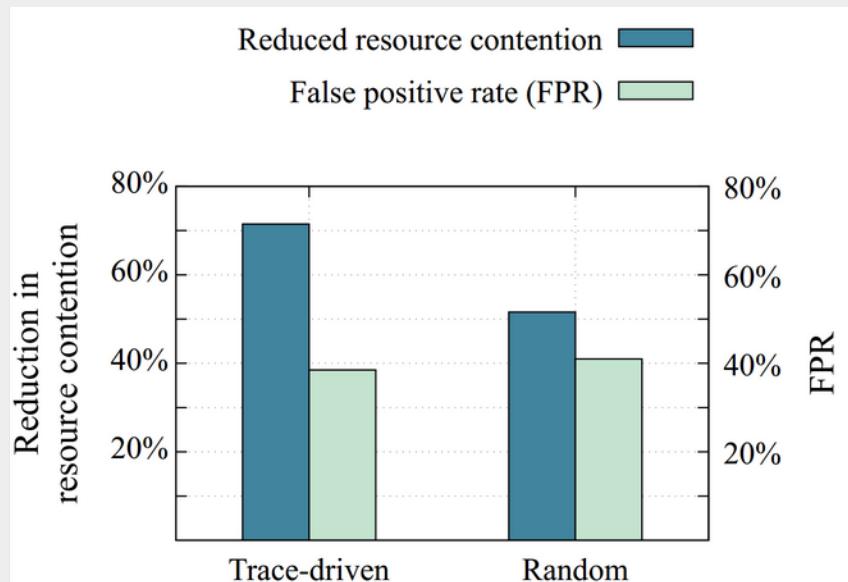
Evaluation



Preventing resource contention

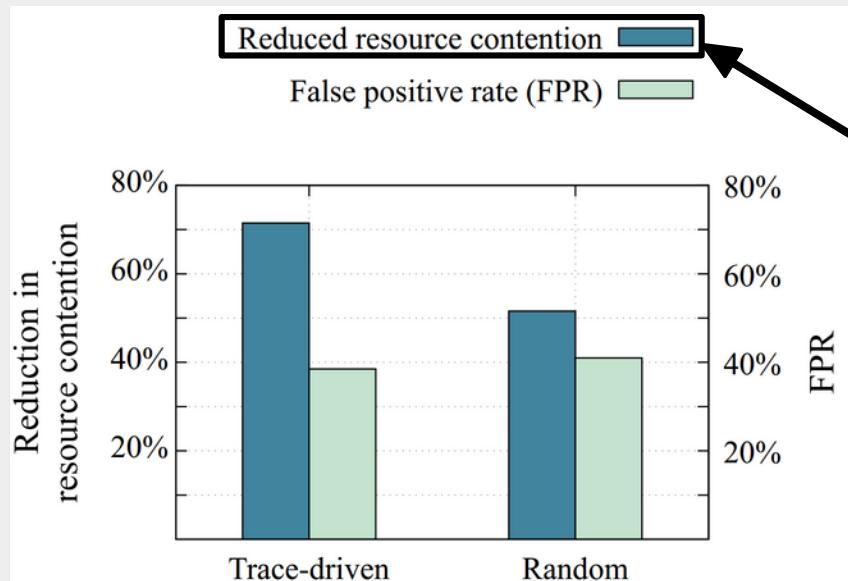
Since the prediction mechanism doesn't depend on requests being batched, we measure the percentage of prevented resource contention across all requests

Evaluation



Preventing resource contention

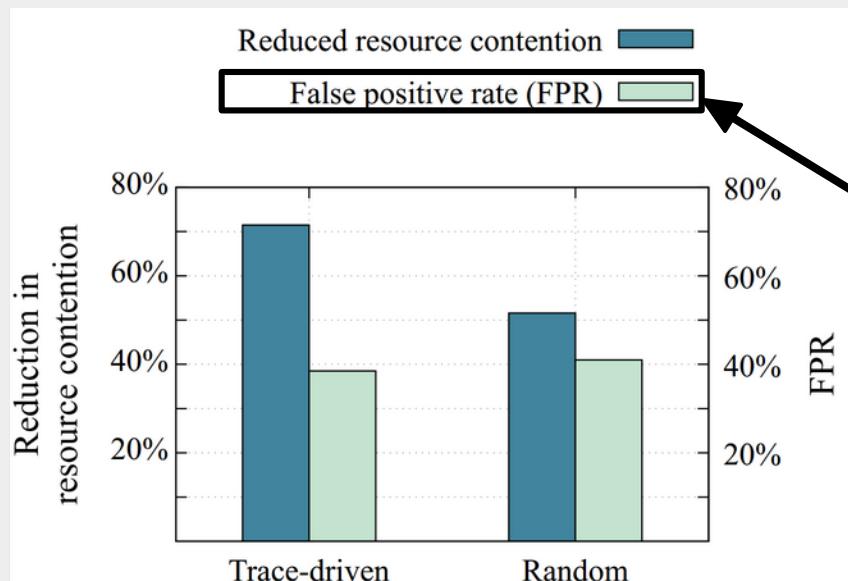
Evaluation



Preventing resource contention

Correctly predicting contention

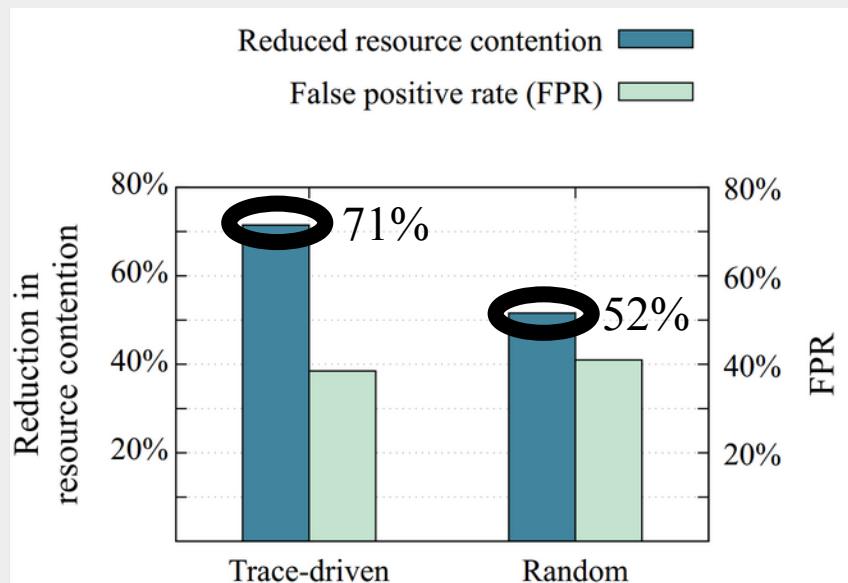
Evaluation



Preventing resource contention

Incorrectly predicting contention

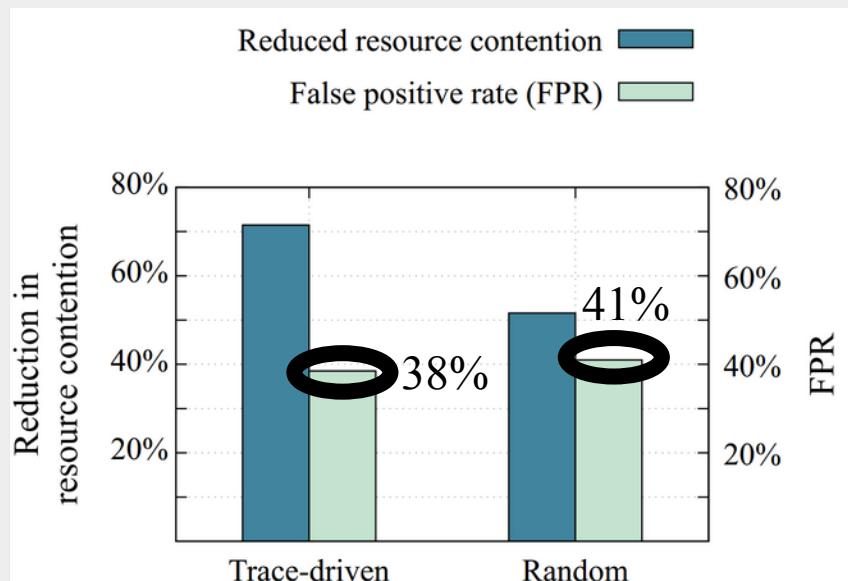
Evaluation



Preventing resource contention

Trace-driven prediction performs significantly better than random

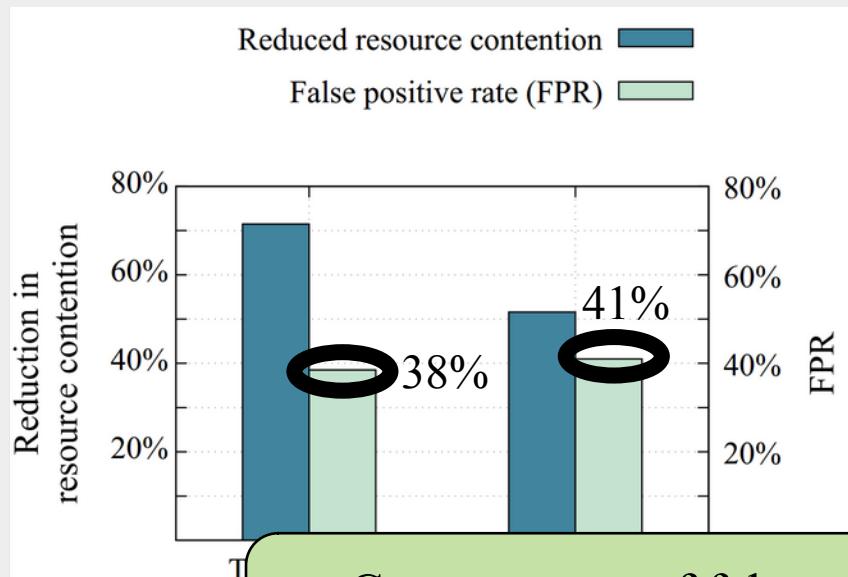
Evaluation



Preventing resource contention

False positive rate is
high for both approaches

Evaluation



Preventing resource contention

False positive rate is high for both approaches

Consequence of false positive is forcing the request to select a different candidate (may not be as good *w.r.t.*, objective value)



Contributions

1

Analyzed and identified dependency problems in logs of a homing service running in production

2

Proposed techniques to coordinate decision making across distributed instances of the homing service

3

Introduced multi-criteria batching to maximize benefits of query caching and consolidated resources

Publications: [to be submitted]

Future Work

Future Work

1

Using ML-based approaches to enhance accuracy
of prediction and batching in STEPNET+

Future Work

1

Using ML-based approaches to enhance accuracy
of prediction and batching in STEPNET+

2

Improving human-network interaction to
facilitate efficient network management

Future Work

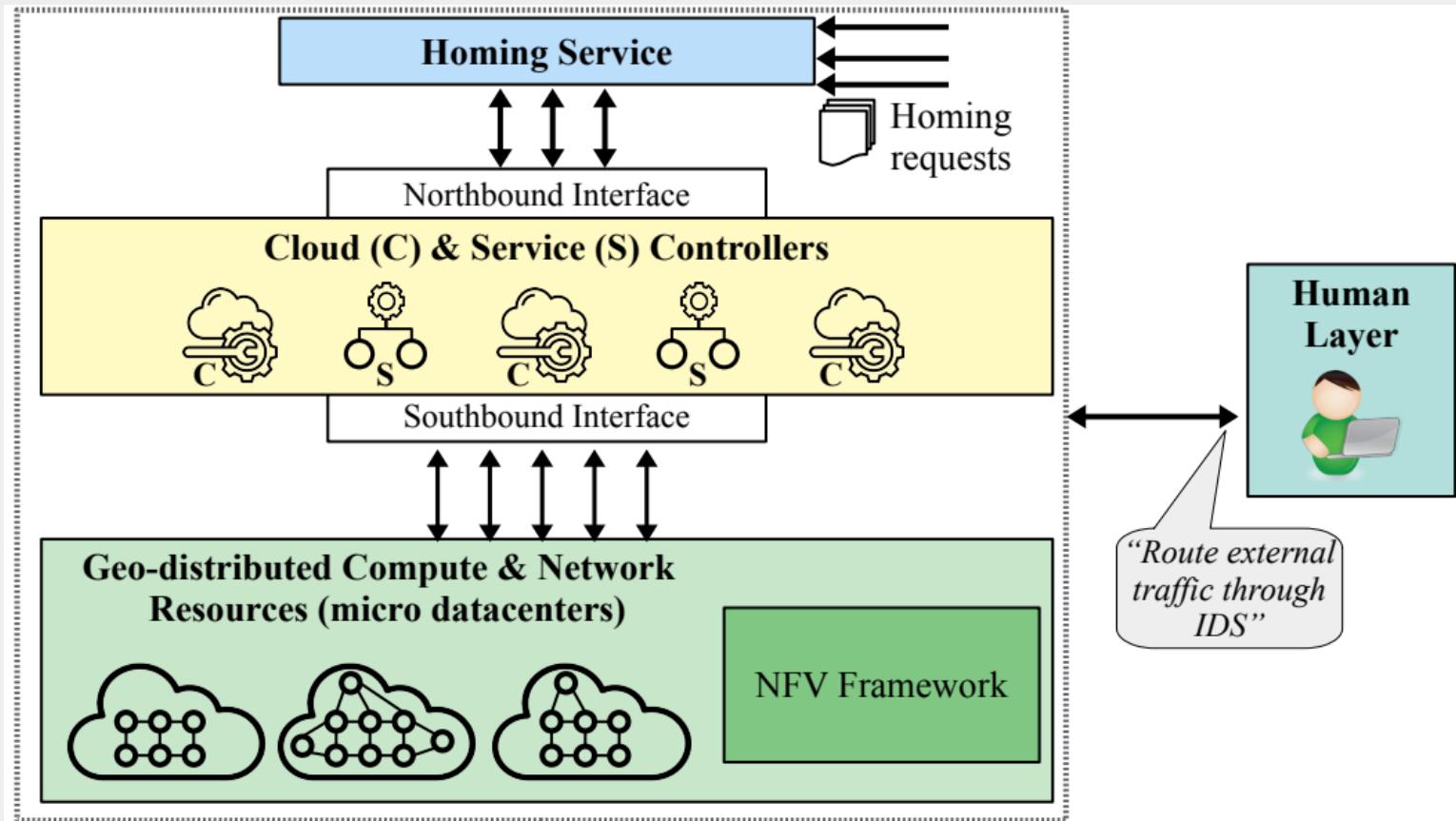
1

Using ML-based approaches to enhance accuracy
of prediction and batching in STEPNET+

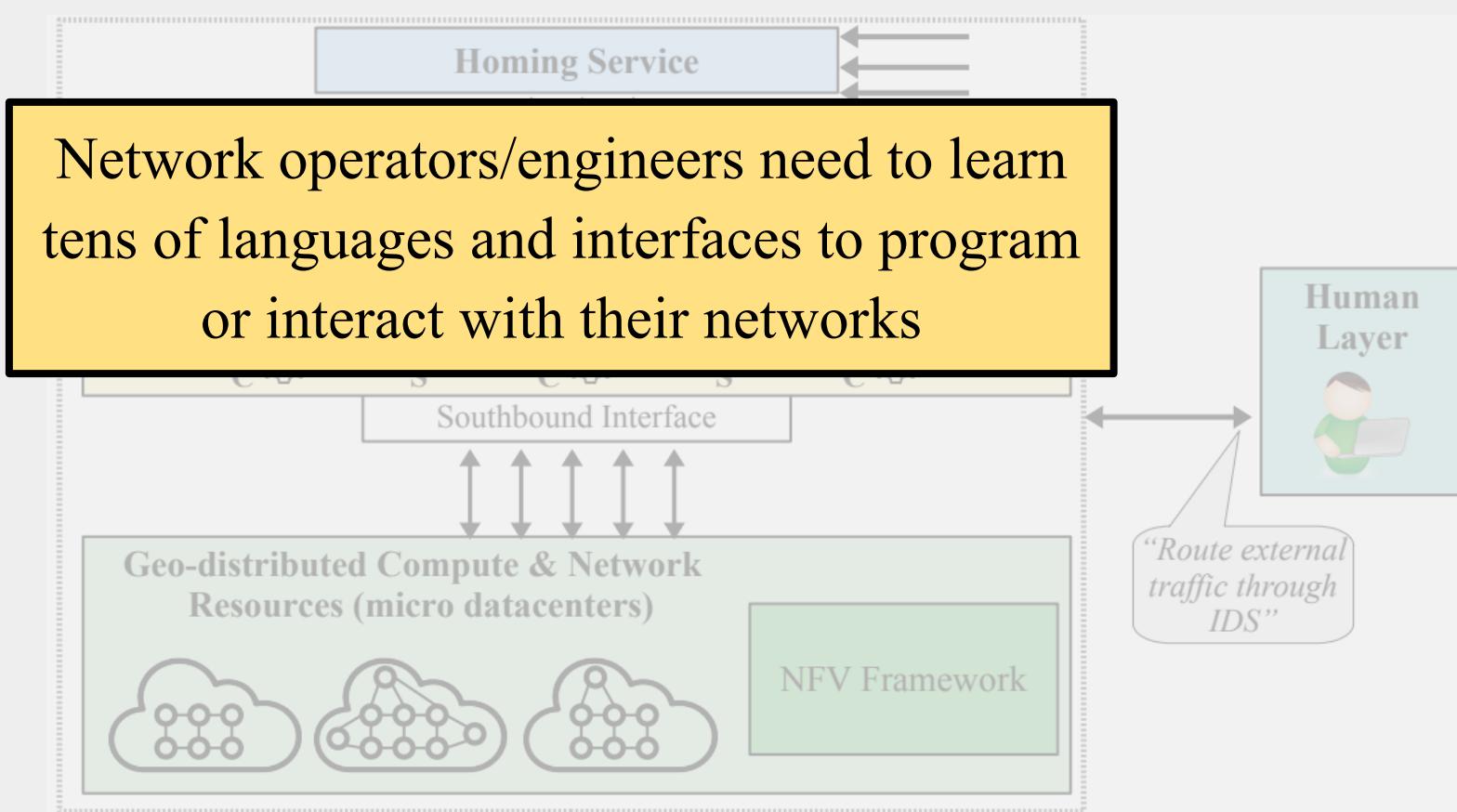
2

Improving human-network interaction to
facilitate efficient network management

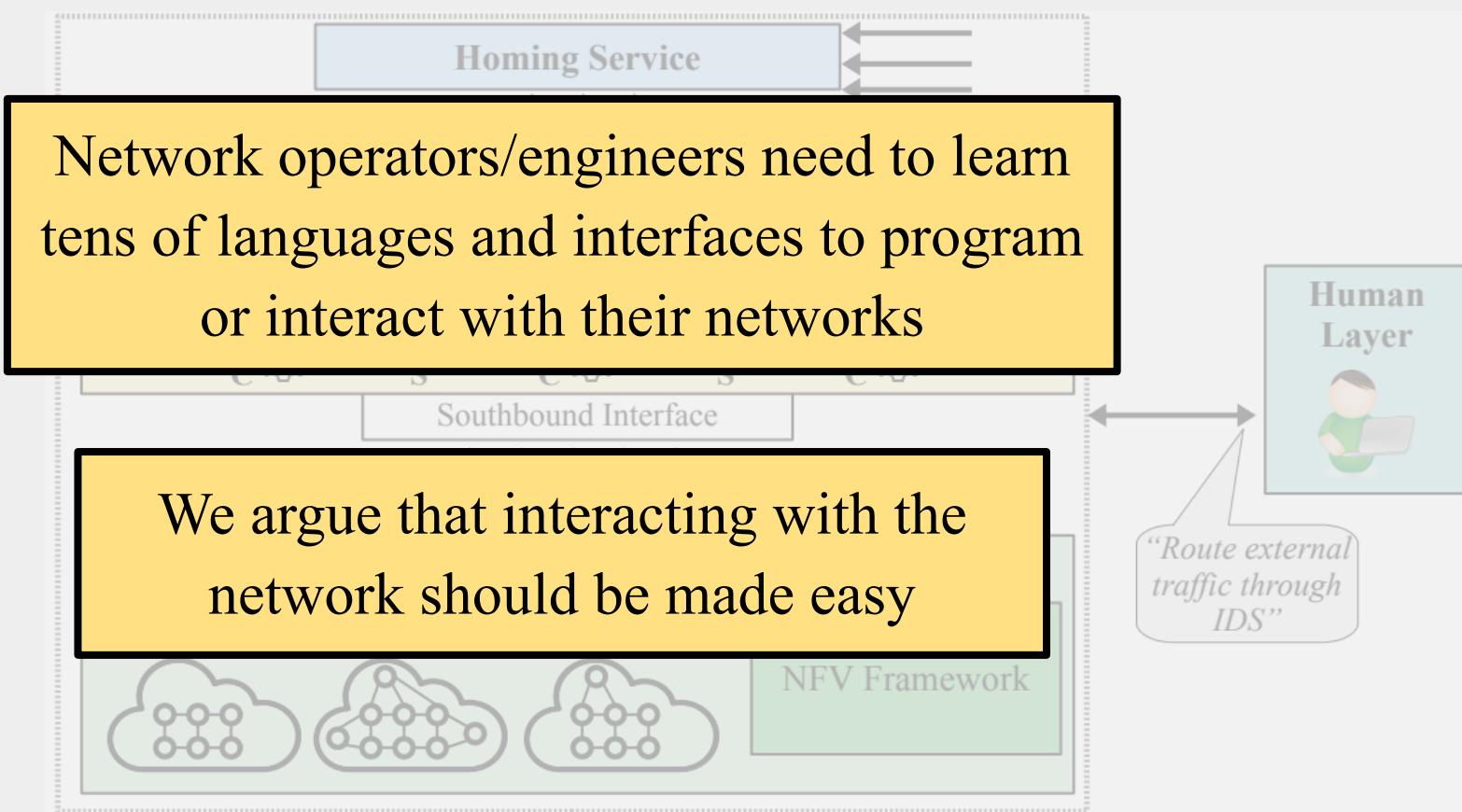
Future Work



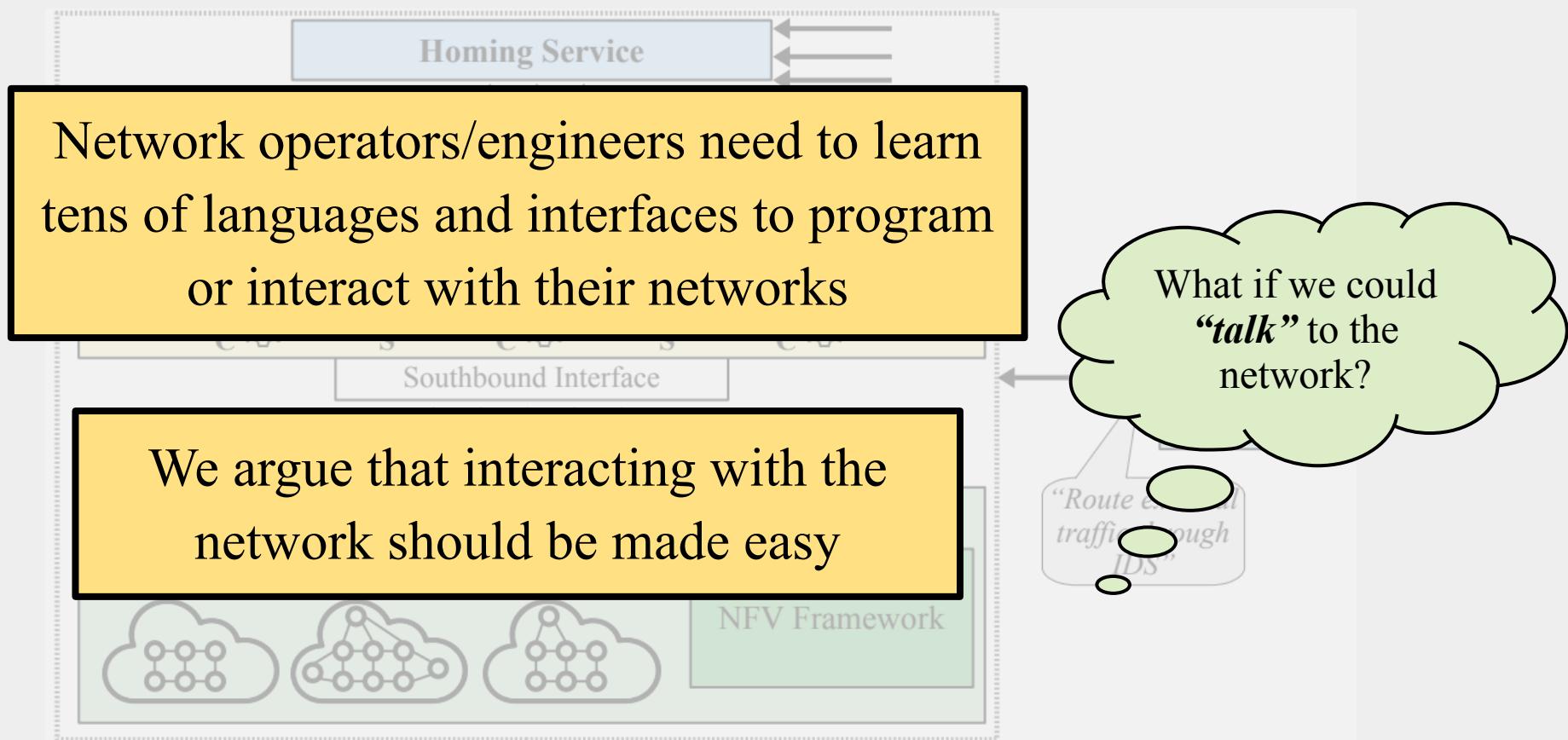
Future Work



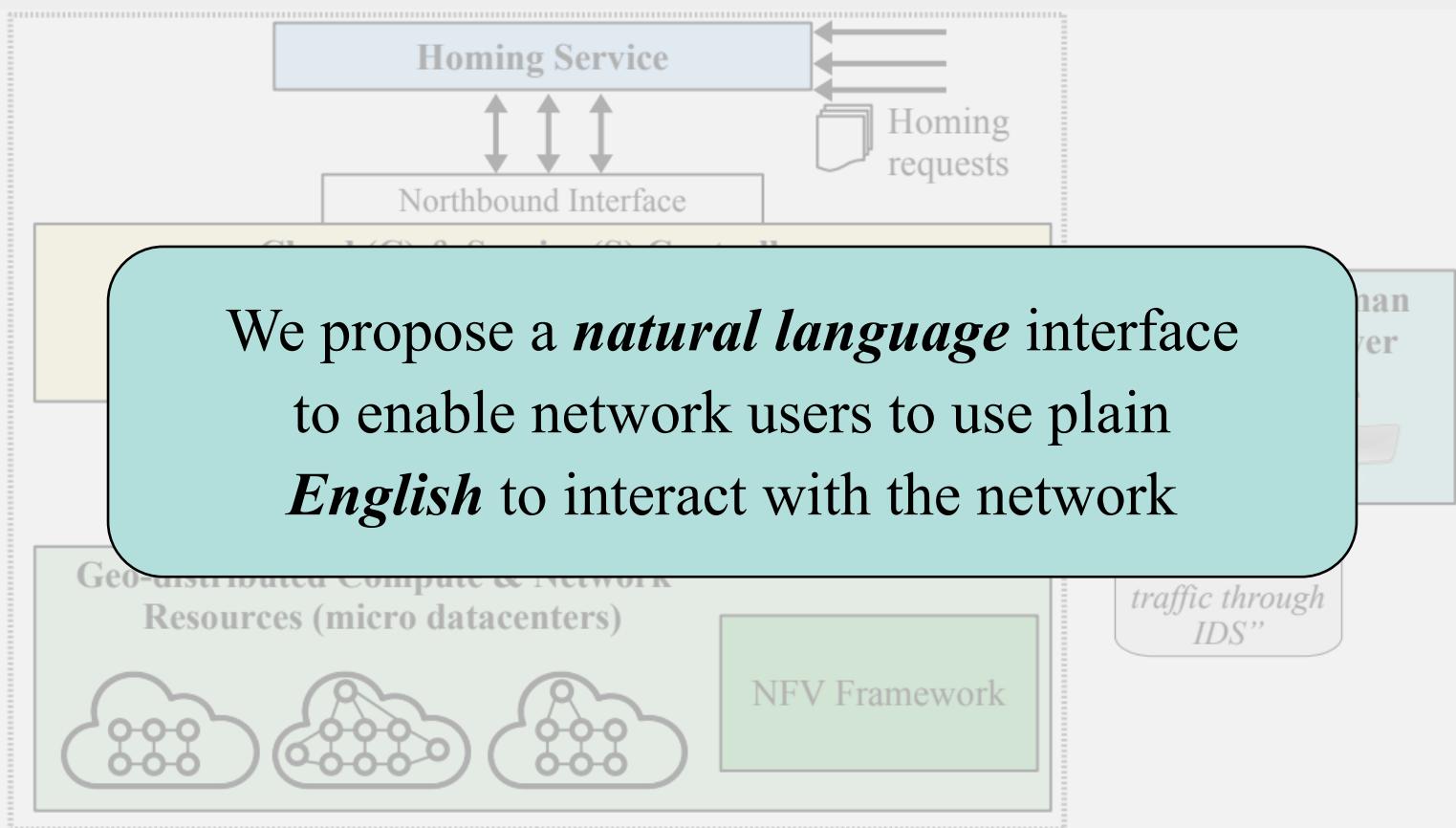
Future Work



Future Work



Future Work



Future Work

Homing Service

Northbound Interface

Homing
requests

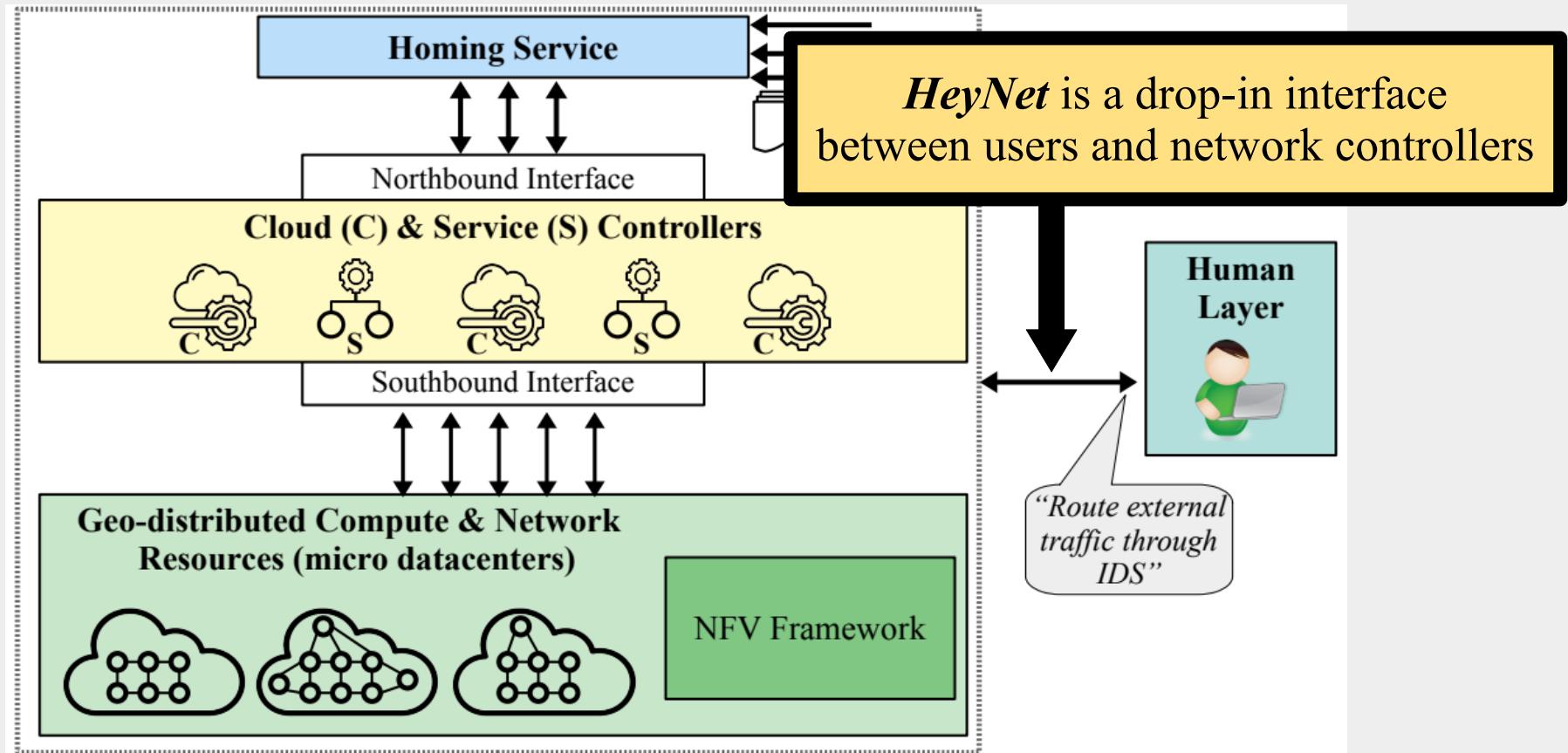
We propose a *natural language* interface
to enable network users to use plain
English to interact with the network

Geo-distributed Compute & Network
Resources (micro datacenters)

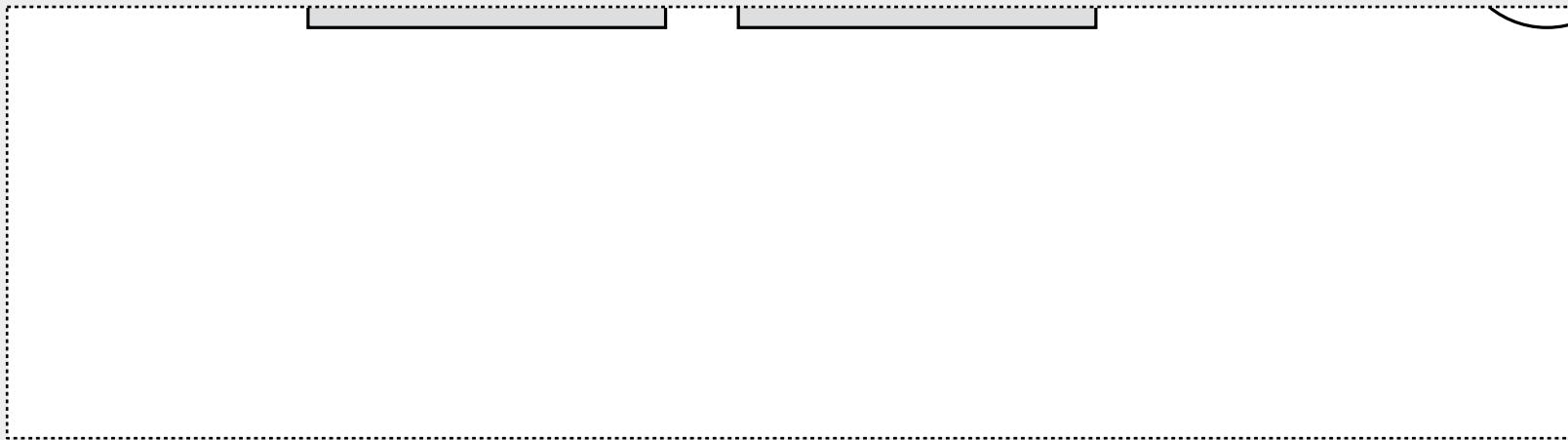
“traffic through
IDS”

Taking intent-based networking (INB) one step further!

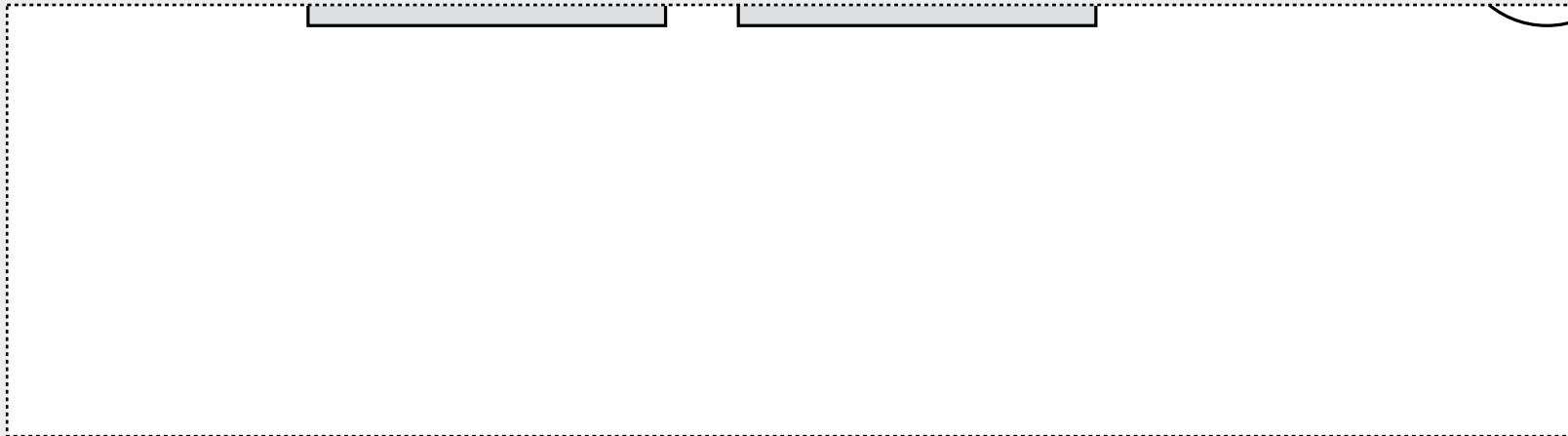
HeyNet



HeyNet

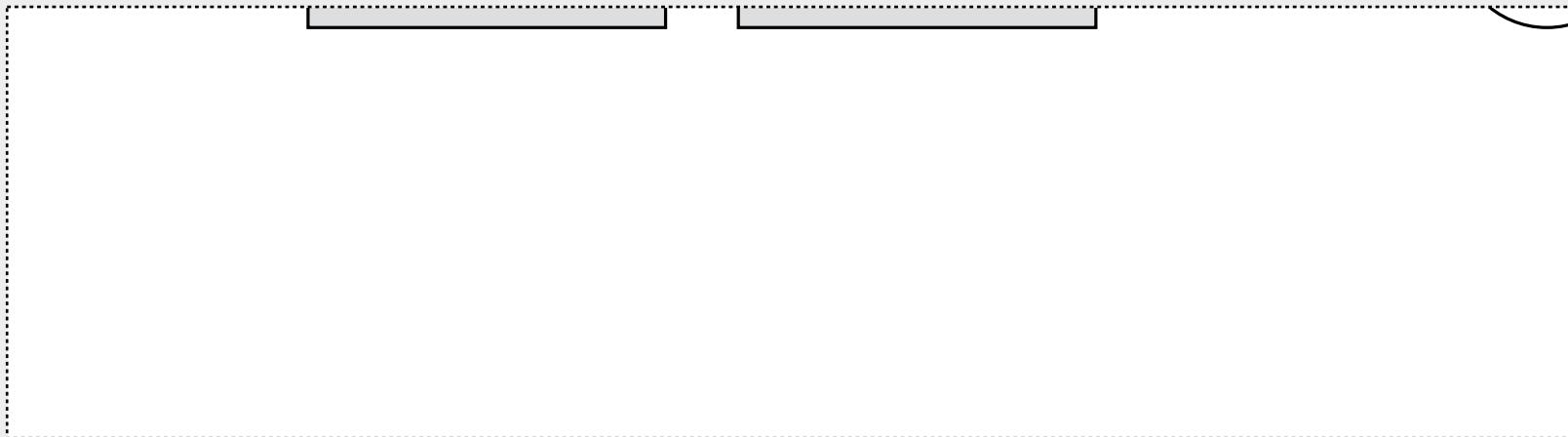


HeyNet



HeyNet parses plain English and constructs network “tasks” that are then converted into queries/configs the controller can understand

HeyNet



HeyNet [INFOCOM WKSHPS 2017] is a first step towards enabling the use of natural language to control and interact with network systems

Contributions

Analyzed production logs of large NSP homing service
and identified various challenges across the homing stack

Contributions

Analyzed production logs of large NSP homing service and identified various challenges across the homing stack

Design and evaluation of Stateless Network Functions to enable efficient management of VNFs in NSP/cloud infrastructures

Contributions

Analyzed production logs of large NSP homing service and identified various challenges across the homing stack

Design and evaluation of Stateless Network Functions to enable efficient management of VNFs in NSP/cloud infrastructures

Design and evaluation of FOCUS, a scalable search service for efficiently processing large-scale geo-distributed queries

Contributions

Analyzed production logs of large NSP homing service and identified various challenges across the homing stack

Design and evaluation of Stateless Network Functions to enable efficient management of VNFs in NSP/cloud infrastructures

Design and evaluation of FOCUS, a scalable search service for efficiently processing large-scale geo-distributed queries

Design and evaluation of StepNet (and StepNet+), a homing service that processes homing requests in an efficient manner

Contributions / Publications

