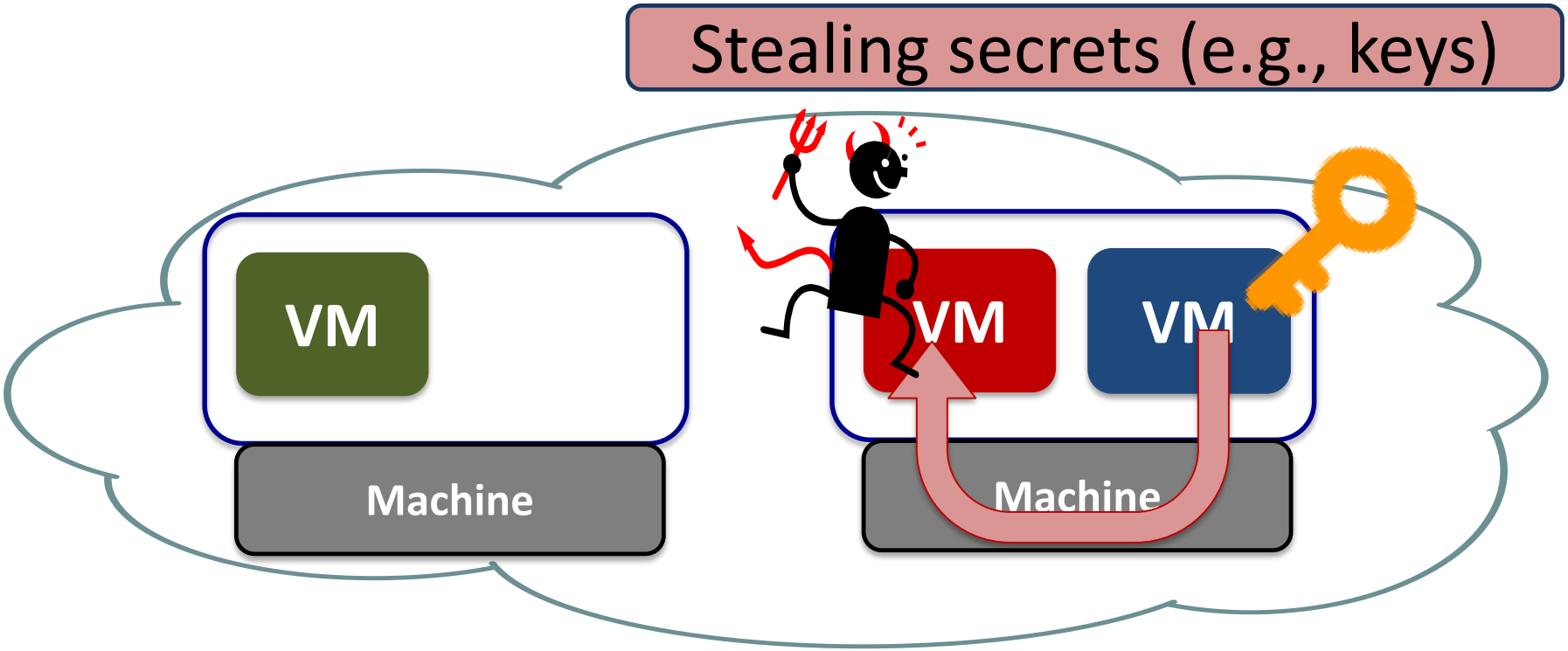
Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration

Soo-Jin Moon, Vyas Sekar Michael K. Reiter

Carnegie Mellon University

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Co-residency side-channel attacks in clouds



- Many different vectors
  (e.g., L2/L3 cache, storage, main memory)

Demonstrated side-channel attacks are not limited to:
Y. Zhang et al., CCS2012; T. Ristenpart et al., CCS2009; F. Liu et al., Oakland 2015

Limitations of Current Defenses

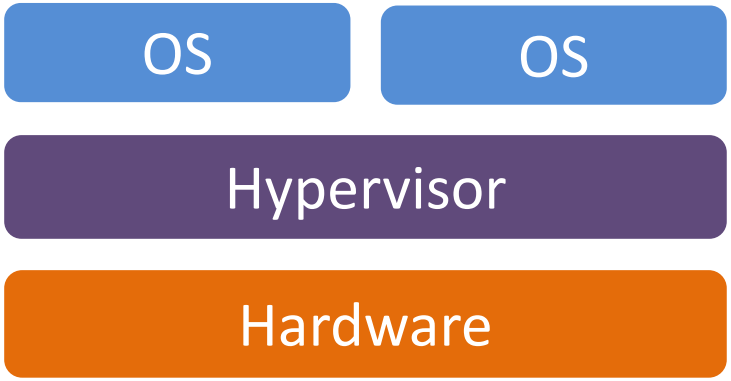
1. Requires significant/detailed upgrades

2. Attack-specific

Proposed defense includes but not limited to: Y. Zhang et al., CCS2013; T. Kim et al., USENIXSec 2012; F. Liu and R. Lee, Micro 2014

Limitations of Current Defenses

1. Requires significant/detailed upgrades



e.g., Noise injection

e.g., Deterministic execution

e.g., New cache design

2. Attack-specific

What about future side-channel attacks?

Proposed defense includes but not limited to: Y. Zhang et al., CCS2013; T. Kim et al., USENIXSec 2012; F. Liu and R. Lee, Micro 2014

Ideal Properties

1) General

2) Immediately deployable

Ideal Properties

1) General

2) Immediately deployable

Single-tenancy?

Ideal Properties

1) General

2) Immediately deployable

~~Single-tenancy?~~

Nomad Ideas

1) General

2) Immediately deployable

Nomad Ideas

1) General

Tackle root-cause

→ Minimize co-residency

2) Immediately deployable

Nomad Ideas

1) General

Tackle root-cause

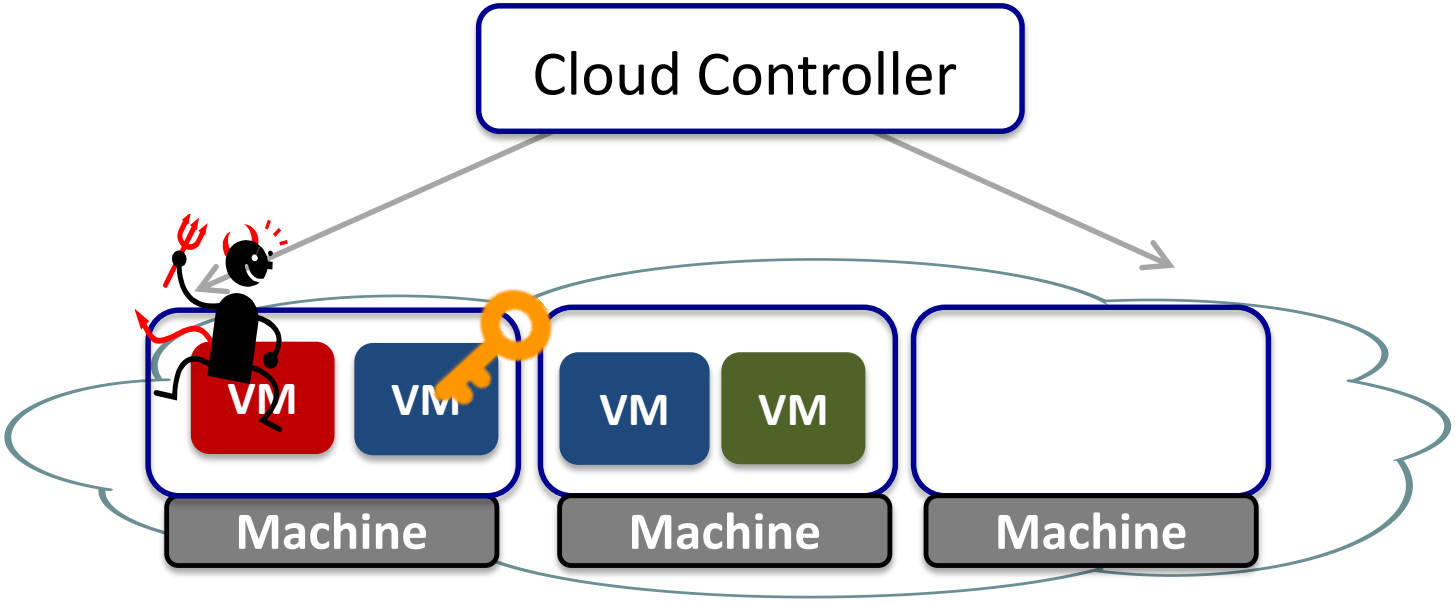
→ Minimize co-residency

2) Immediately deployable

Migration

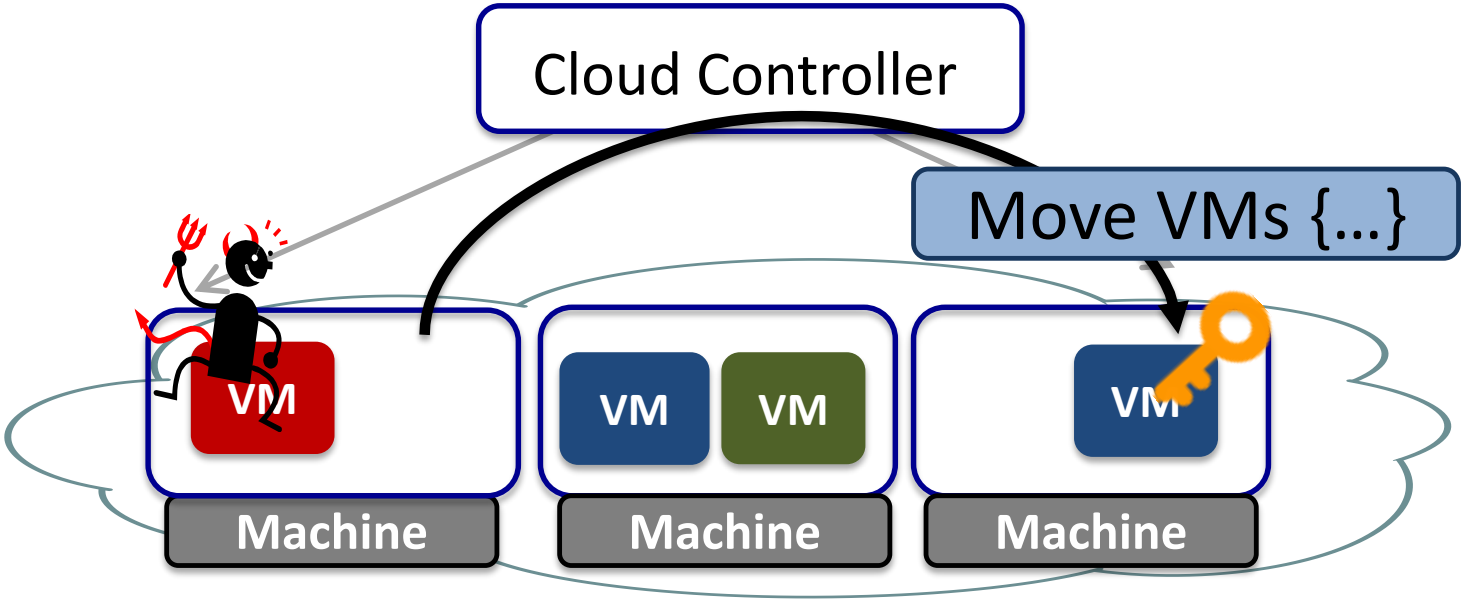
Nomad Vision: Migration-as-a-Service

- Provider-assisted



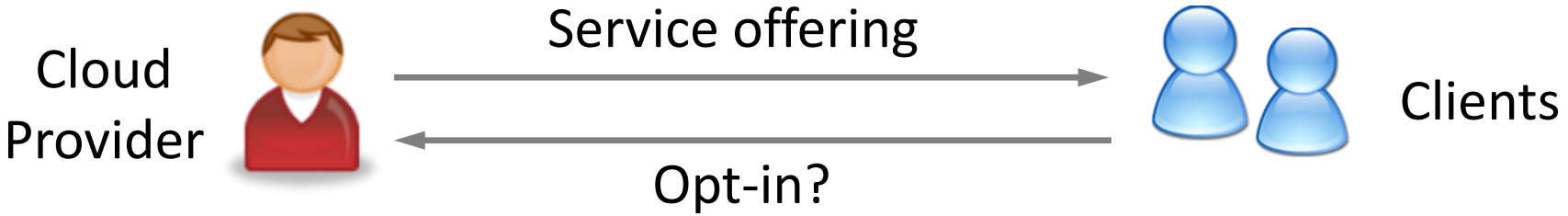
Nomad Vision: Migration-as-a-Service

- Provider-assisted

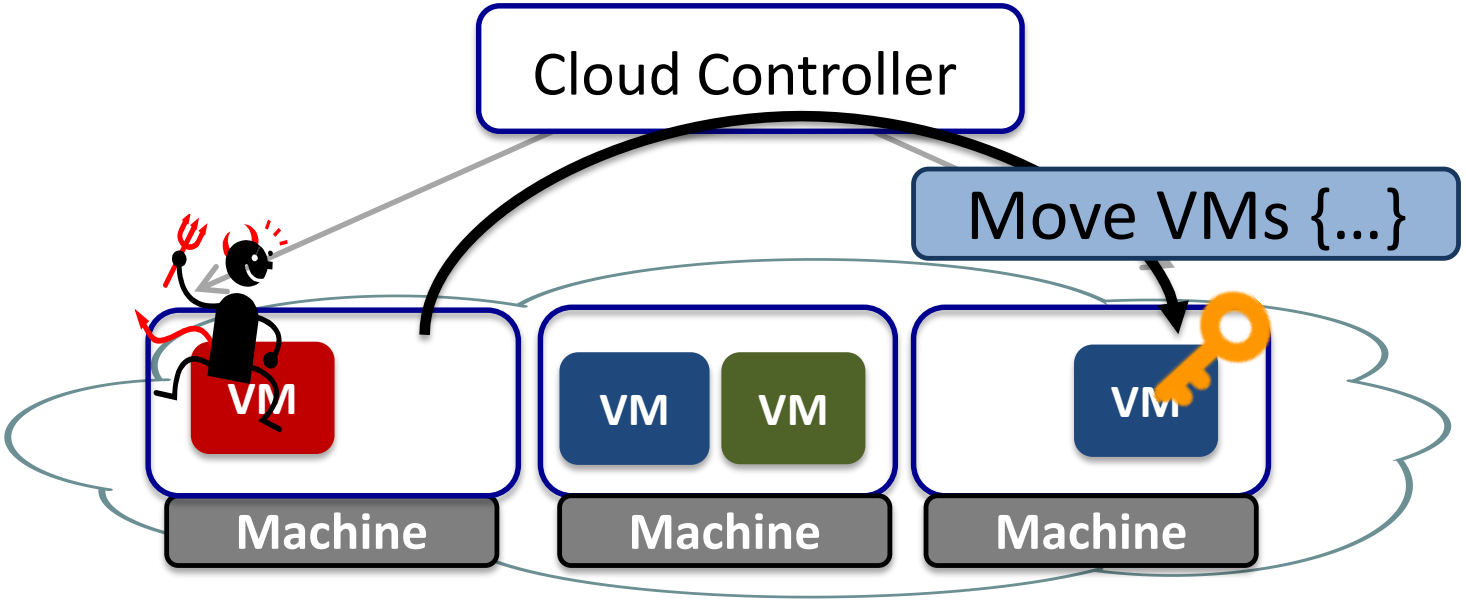


Nomad Vision: Migration-as-a-Service

- Opt-in Service



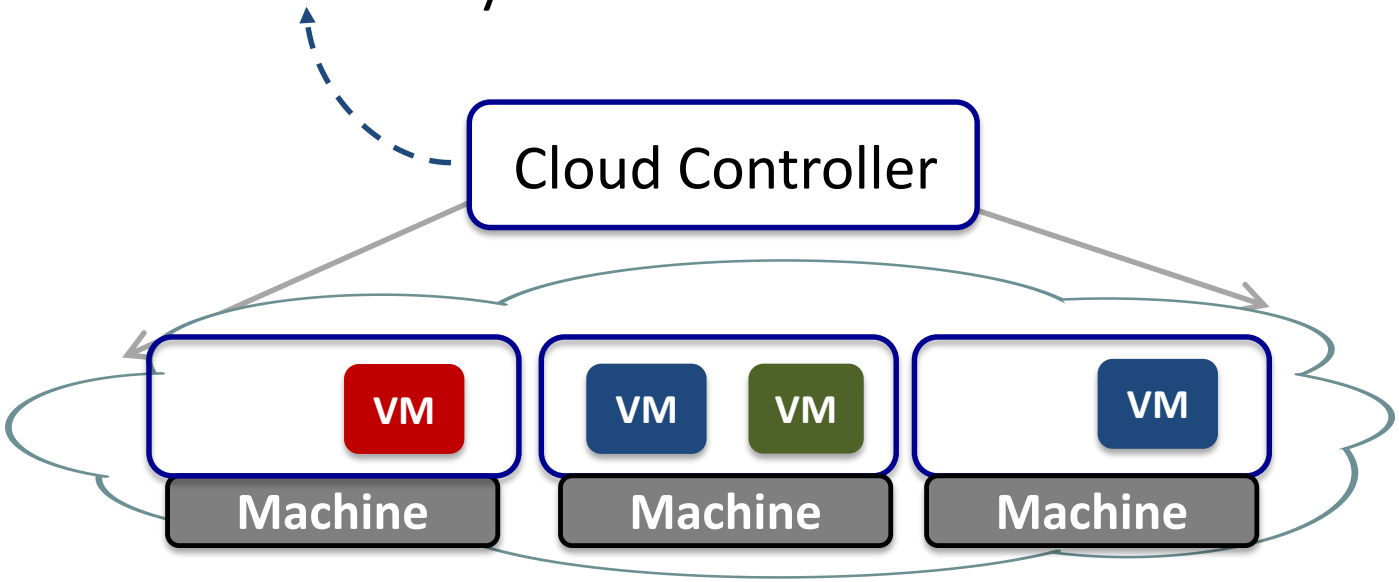
- Provider-assisted



Nomad Practical Challenges

Logic

Characterize information leakage due to co-residency



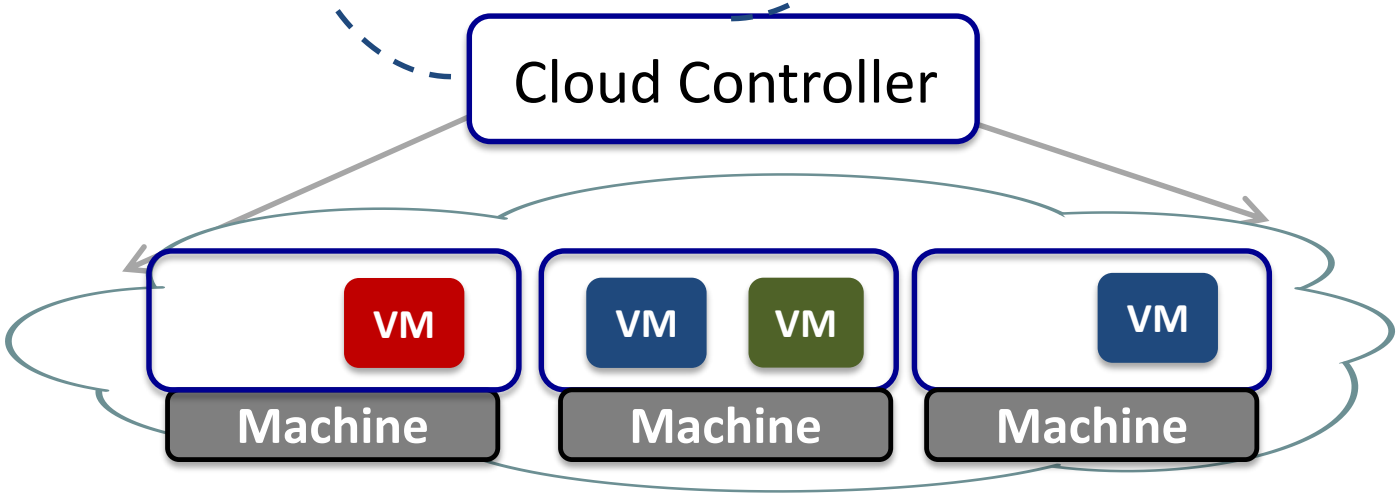
Nomad Practical Challenges

Logic

Characterize information leakage due to co-residency

Scalable Design

e.g., can Amazon EC2 run this?



Nomad Practical Challenges

Logic

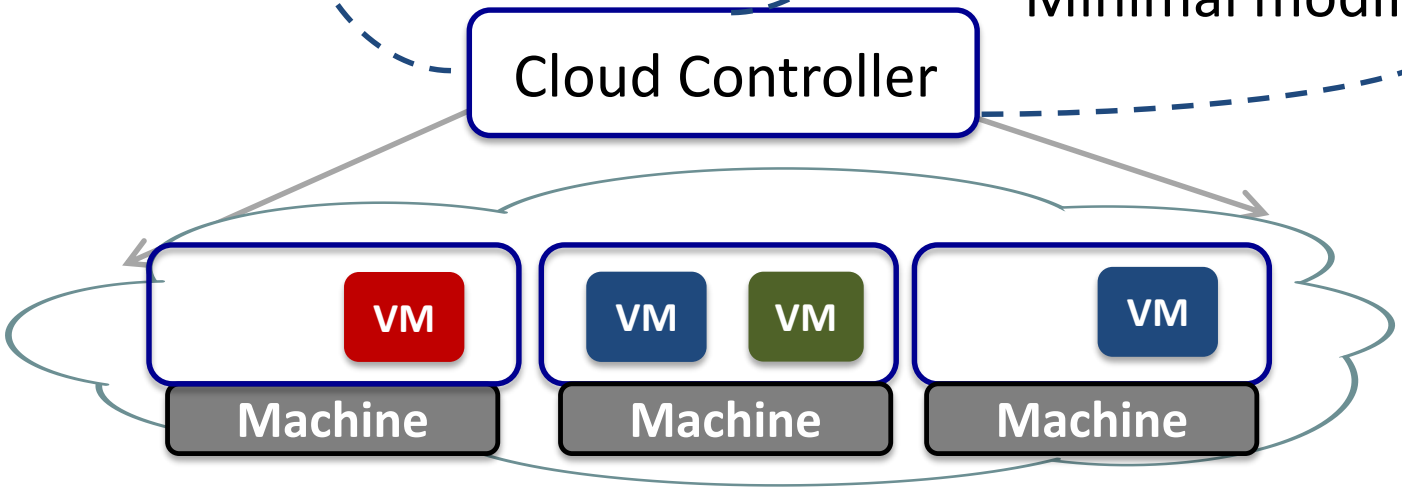
Characterize information leakage due to co-residency

Scalable Design

e.g., can Amazon EC2 run this?

Practical Impact (cloud)

Minimal modifications?



Nomad Practical Challenges

Logic

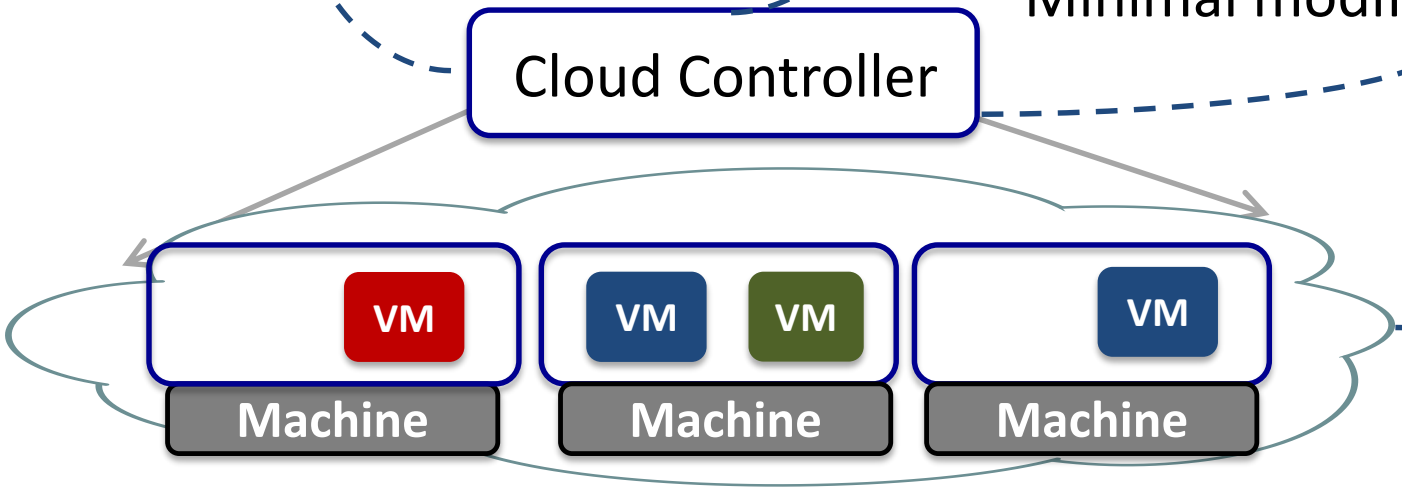
Characterize information leakage due to co-residency

Scalable Design

e.g., can Amazon EC2 run this?

Practical Impact (cloud)

Minimal modifications?



Practical Impact (applications)

1) Advancement of VM migration techniques
2) Many cloud workloads with in-built resilience to migration

Our Work

1. Idea

General side-channel defense via migration

Our Work

1. Idea

General side-channel defense via migration

→

2. Logic

Characterize information leakage due to co-residency

Our Work

1. Idea

General side-channel defense via migration

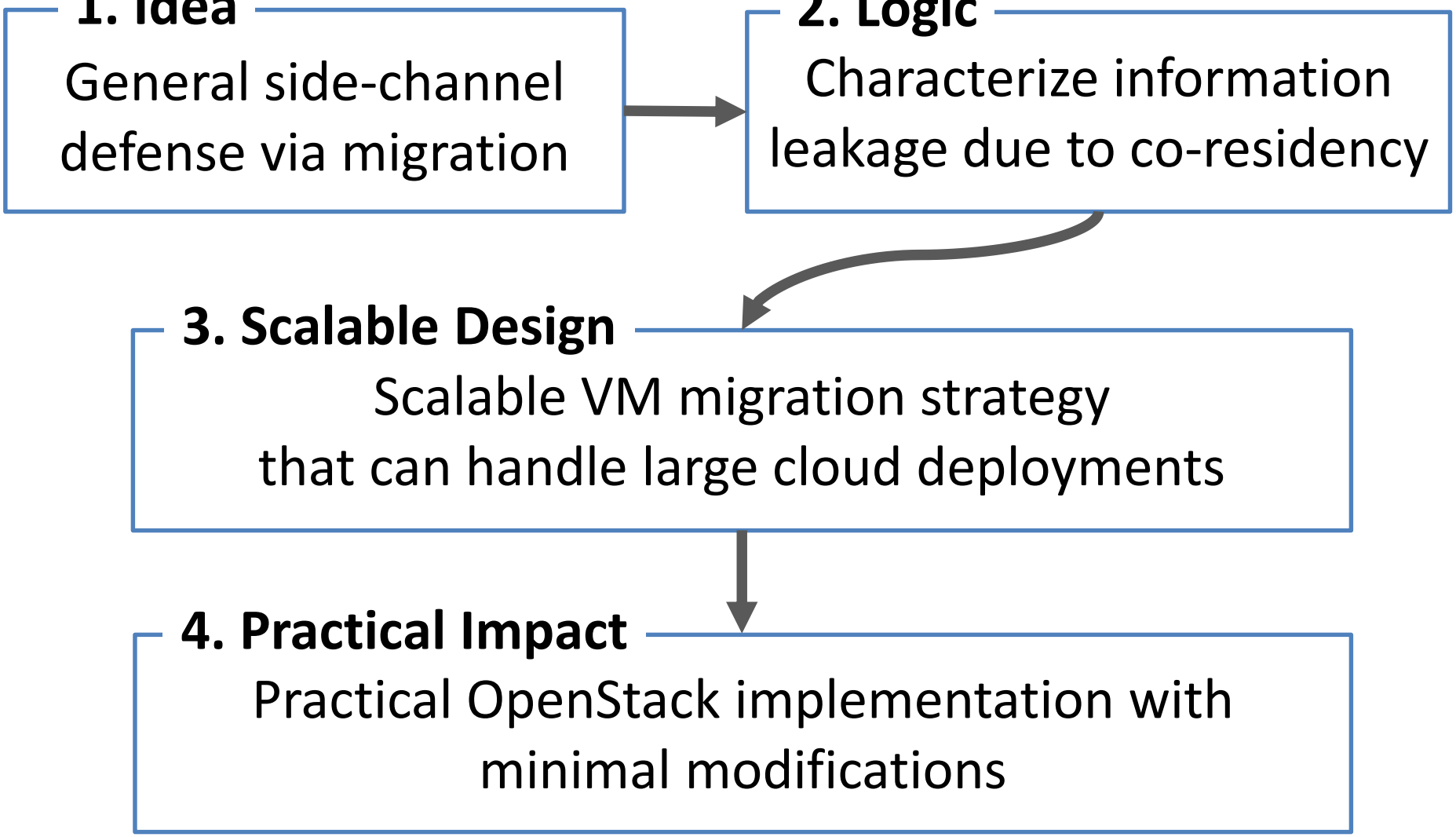
2. Logic

Characterize information leakage due to co-residency

3. Scalable Design

Scalable VM migration strategy that can handle large cloud deployments

Our Work


1. Idea
General side-channel defense via migration
2. Logic
Characterize information leakage due to co-residency
3. Scalable Design
Scalable VM migration strategy that can handle large cloud deployments
4. Practical Impact
Practical OpenStack implementation with minimal modifications

Our Work

1. Idea

General side-channel defense via migration

2. Logic

Characterize information leakage due to co-residency

3. Scalable Design

Scalable VM migration strategy that can handle large cloud deployments

4. Practical Impact

Practical OpenStack implementation with minimal modifications

Threat Model

Objective: Extract secrets via co-residency

- Can use any kind of resource
- Can launch/terminate VMs at will
- VMs of a given client can collaborate

Threat Model

Objective: Extract secrets via co-residency

- Can use any kind of resource
- Can launch/terminate VMs at will
- VMs of a given client can collaborate

- Cannot control VM placement
- No info. sharing across distinct clients

Threat Model

Objective: Extract secrets via co-residency

- Can use any kind of resource
- Can launch/terminate VMs at will
- VMs of a given client can collaborate

- Cannot control VM placement
- No info. sharing across distinct clients

Provider

- Don’t know which other clients are malicious

Information Leakage (InfoLeak) Model

Clients

Information Leakage (InfoLeak) Model

Clients

VM-level view

Replicated? (R or NR)

R

Information Leakage (InfoLeak) Model

Clients

VM-level view

Replicated? (R or NR)

R

NR

Information Leakage (InfoLeak) Model

Clients

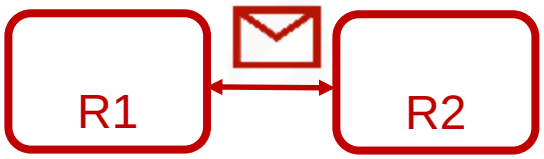
VM-level view

Replicated? (R or NR)

Collaborating? (C or NC)

R

C



NR

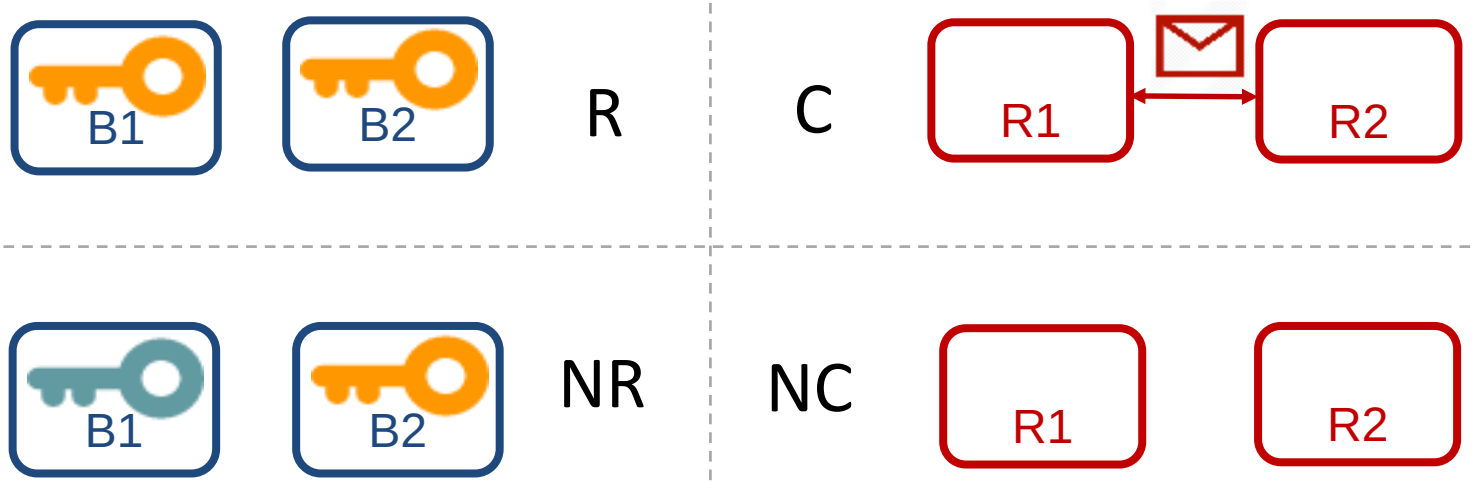
Information Leakage (InfoLeak) Model

Clients

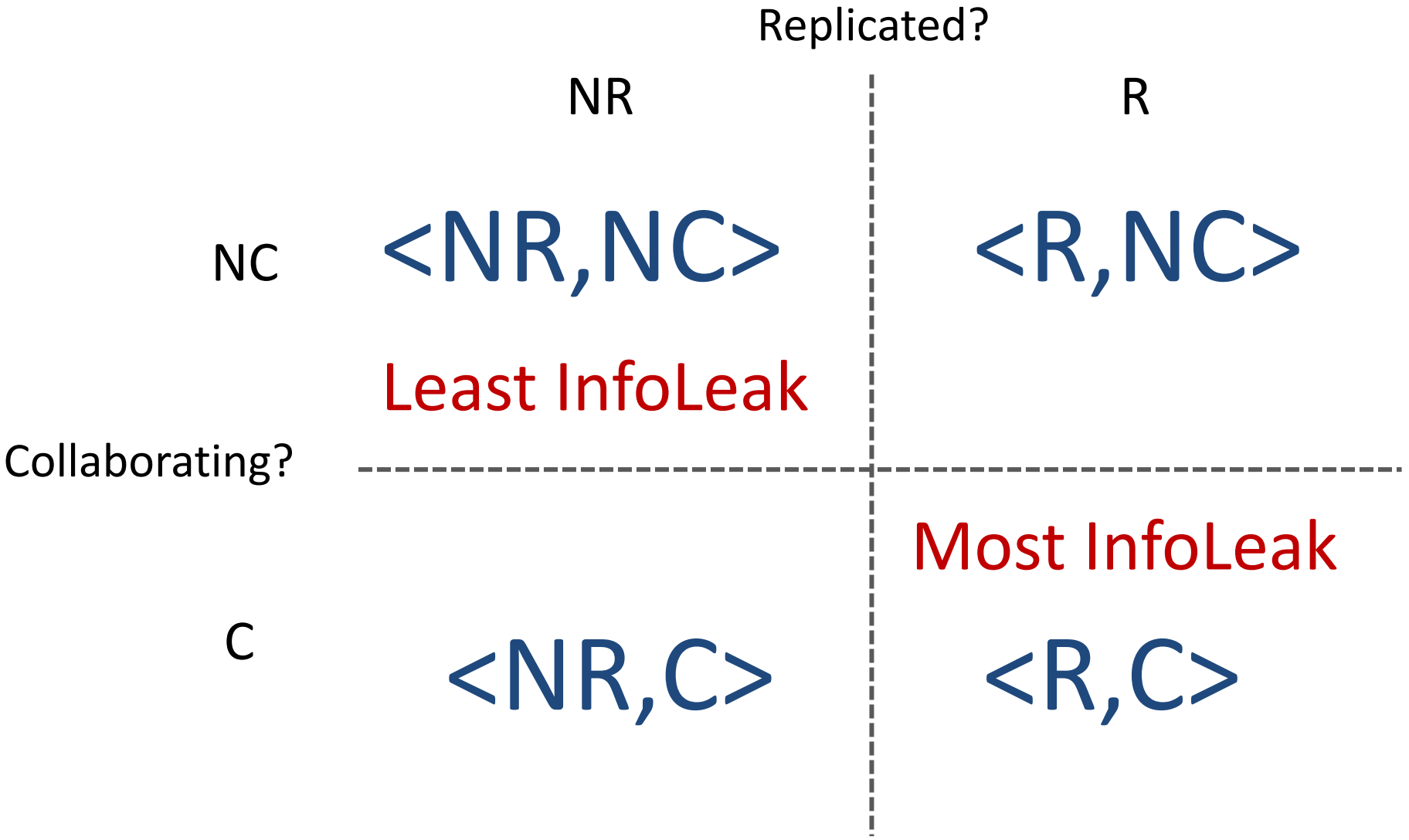
VM-level view

Replicated? (R or NR)

Collaborating? (C or NC)



Information Leakage (*InfoLeak*) Model



Our Work

1. Idea

General side-channel defense via migration

2. Logic

Characterize information leakage due to co-residency

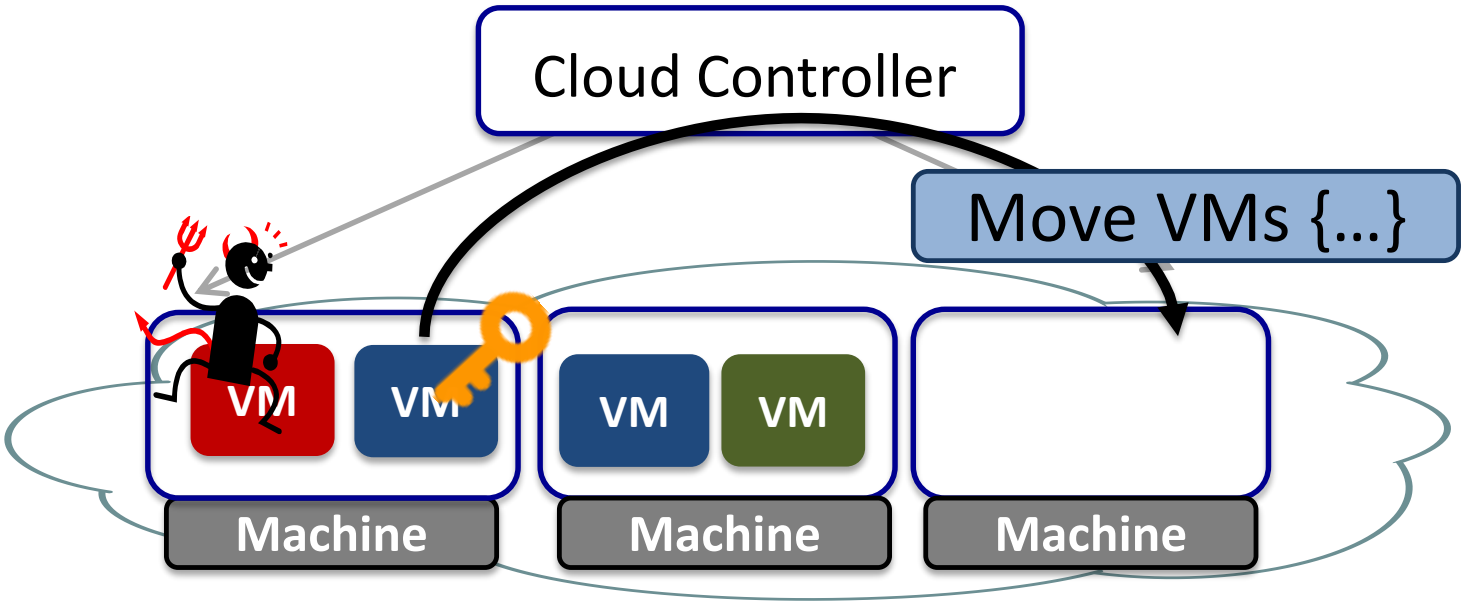
3. Scalable Design

Scalable VM migration strategy that can handle large cloud deployments

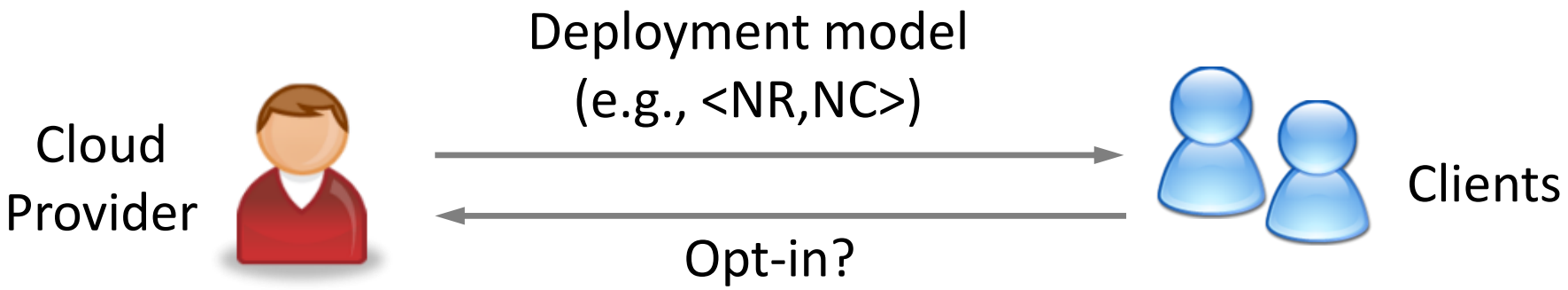
4. Practical Impact

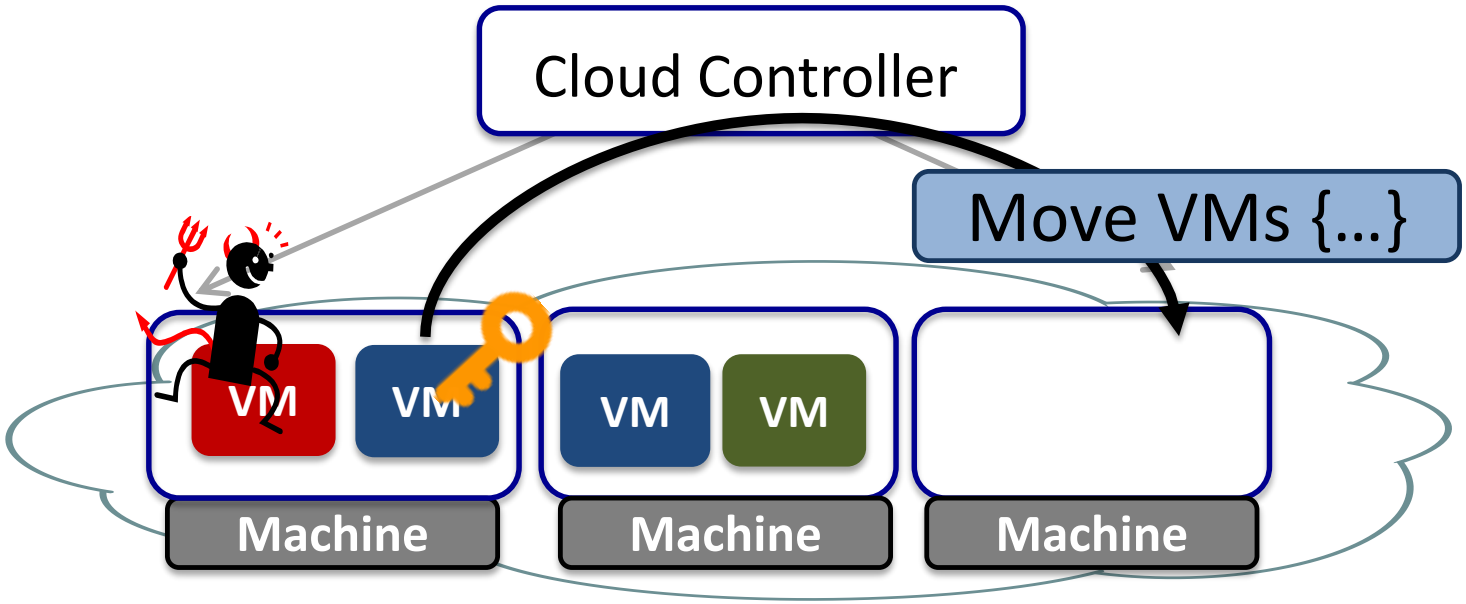
Practical OpenStack implementation with minimal modifications

System Overview

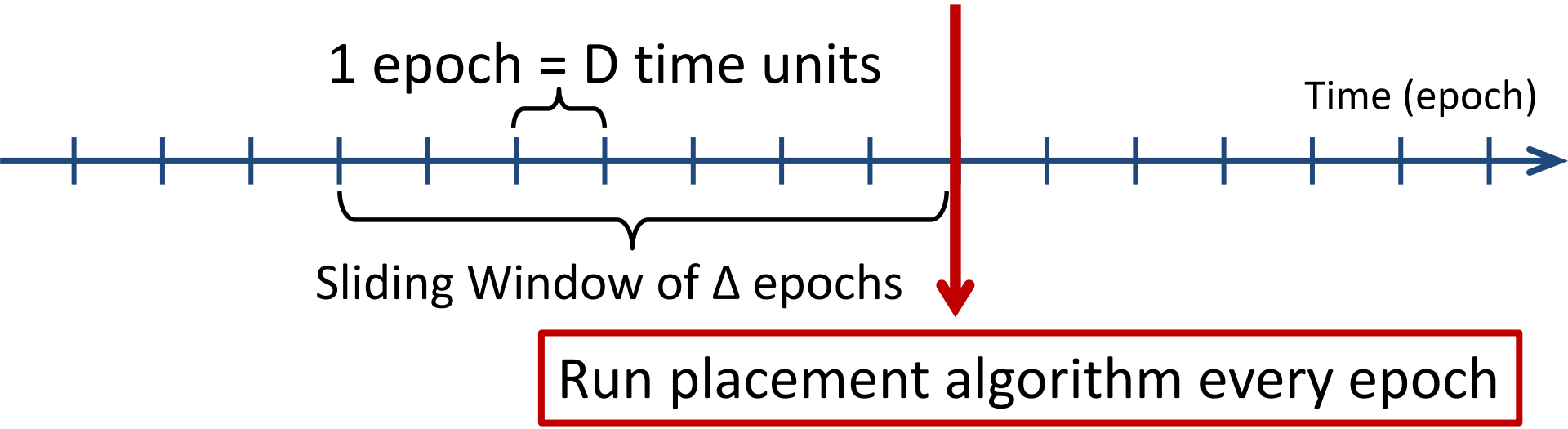


System Overview



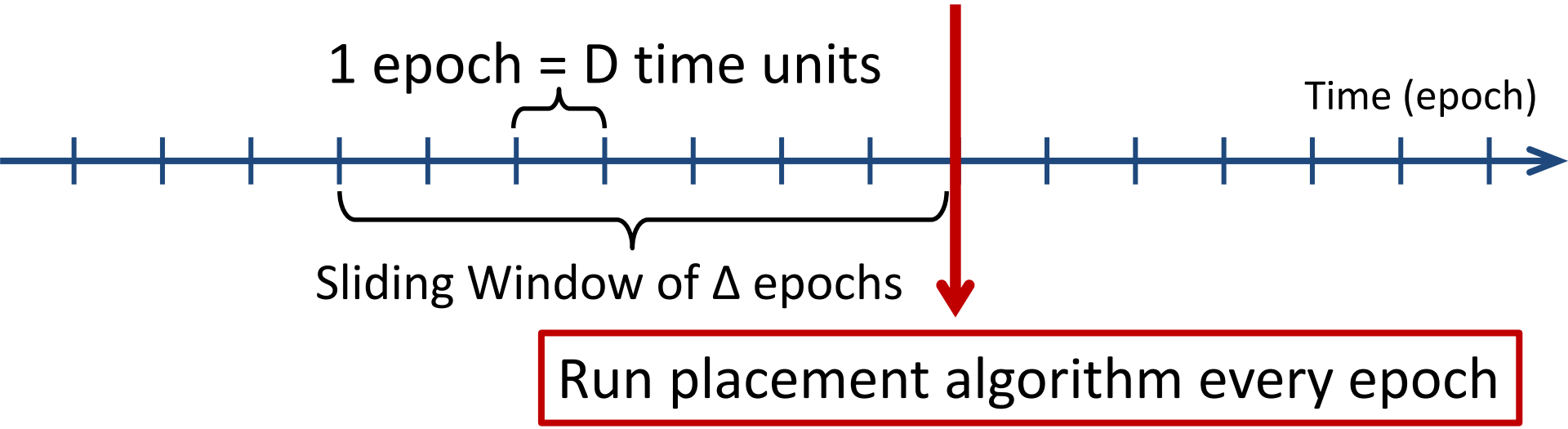


Operational Timeline



Run placement algorithm every epoch

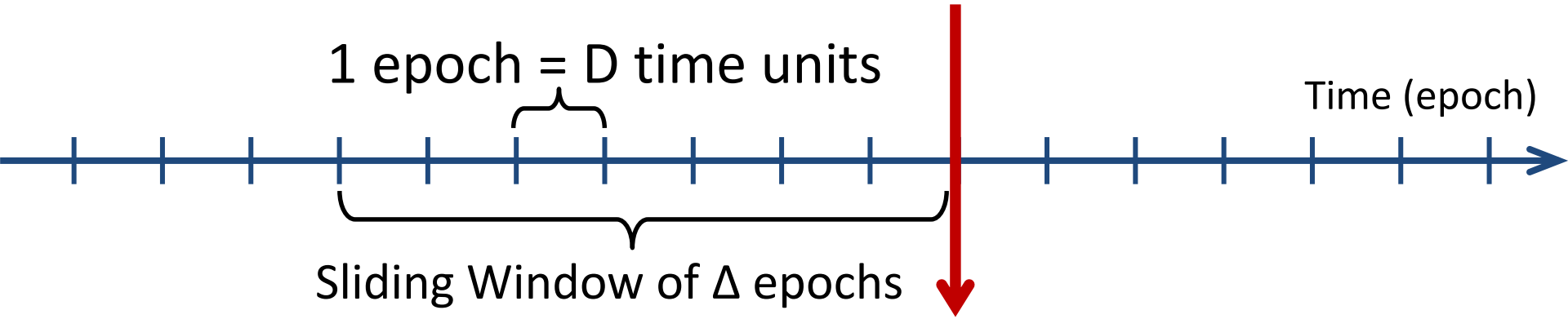
Operational Timeline



Side-channel Parameters:

- K: Information leakage rate (i.e., bits per time unit)
- P: secret length (i.e., bits)

Operational Timeline



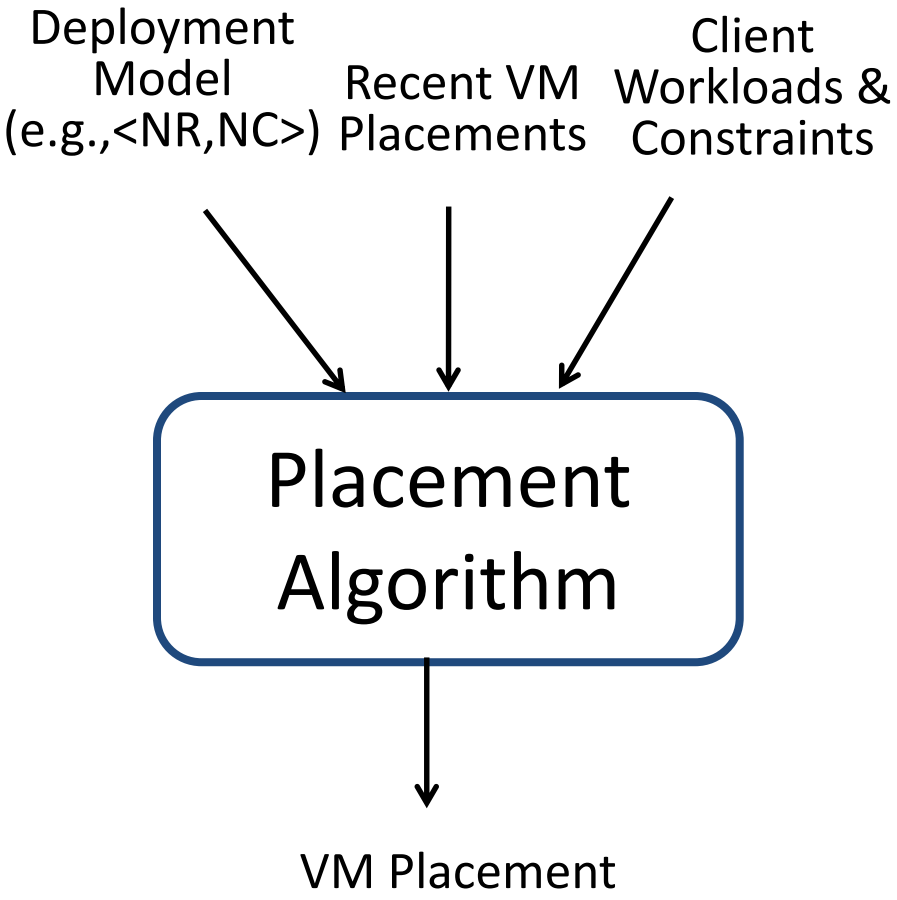
Run placement algorithm every epoch

Extracted secret (bits) if two VMs are co-resident for Δ epochs

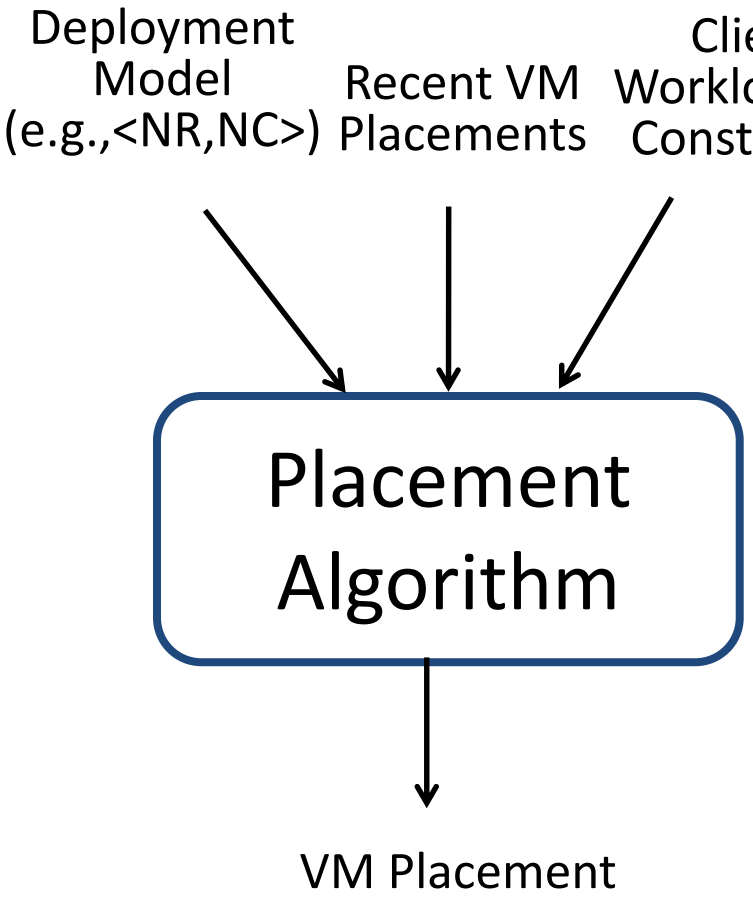
Provider chooses D and Δ to AT LEAST satisfy:

$$D * \Delta * K < P$$

Placement Algorithm



Placement Algorithm



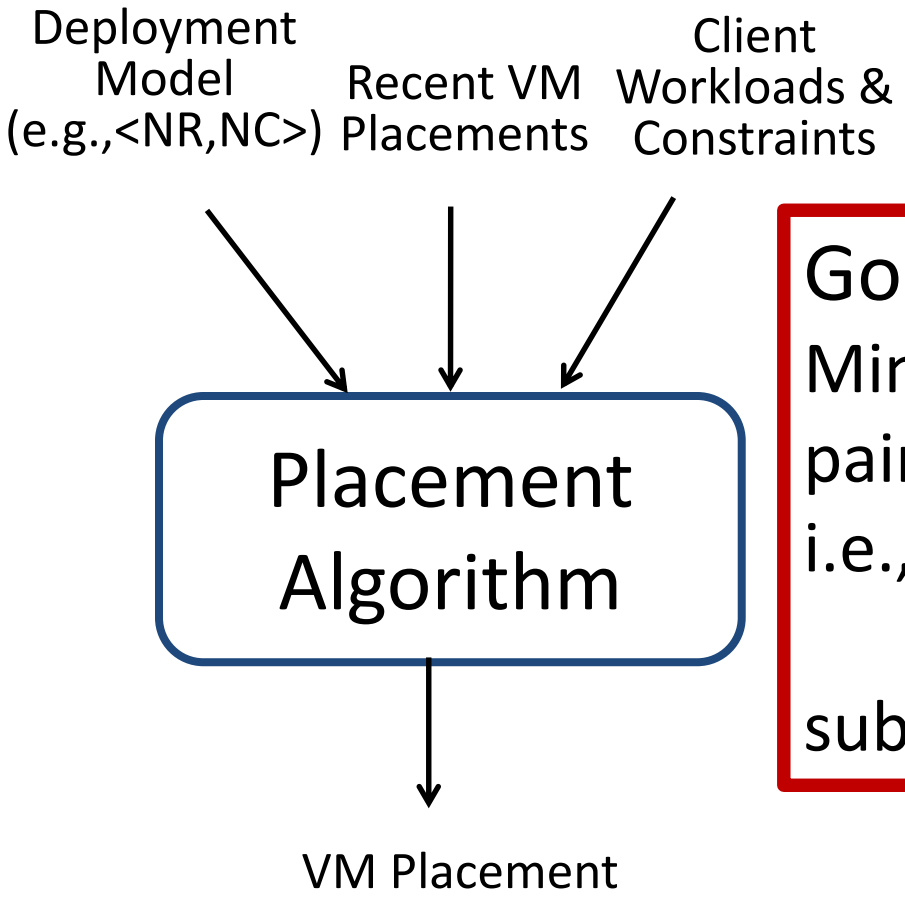
Goal (per epoch):

Minimize a *global* sum of a client-pair InfoLeak across past Δ epochs

i.e., $\sum_{c,c'} InfoLeak_{c \to c'}([t-\Delta, t])$

subject to a fixed migration budget

Placement Algorithm



F(Deployment Model)

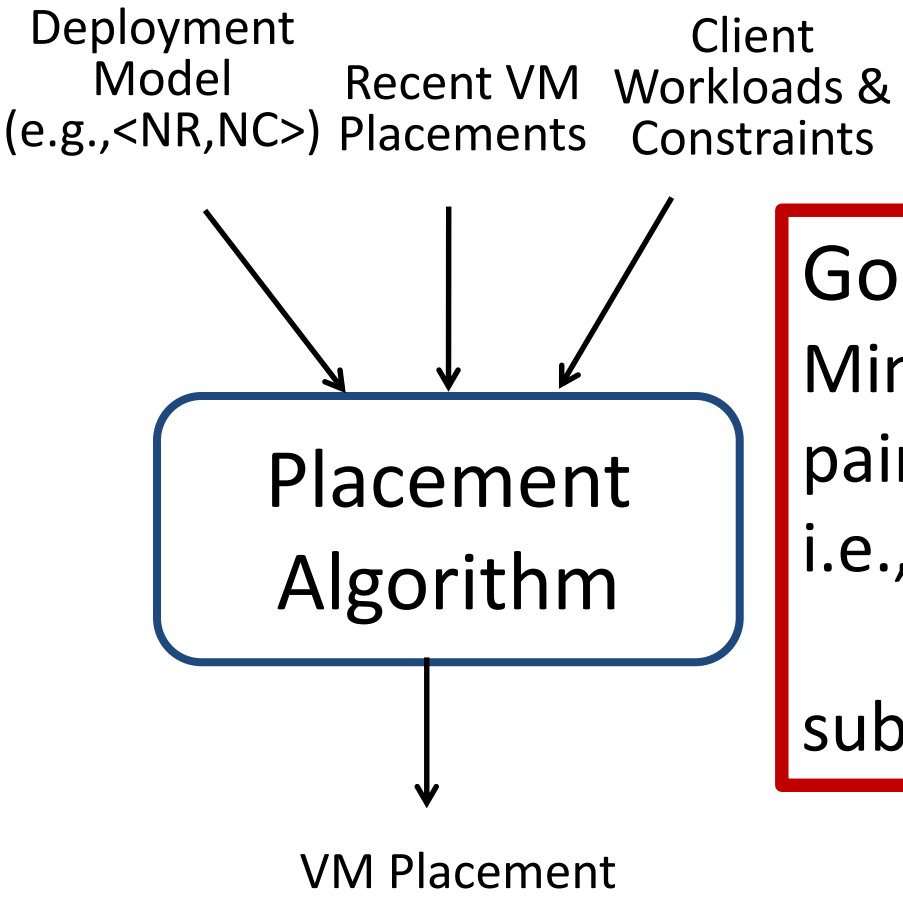
Goal (per epoch):

Minimize a *global* sum of a client-pair InfoLeak across past Δ epochs

i.e., $\sum_{c,c'} InfoLeak_{c \to c'}([t-\Delta, t])$

subject to a fixed migration budget

Placement Algorithm



F(Deployment Model)

Goal (per epoch):

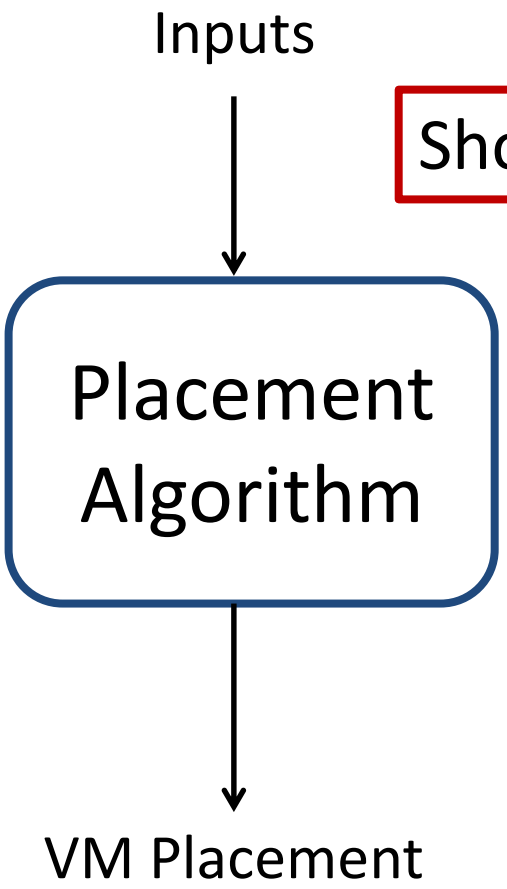
Minimize a *global* sum of a client-pair InfoLeak across past Δ epochs

i.e., $\sum_{c,c'} InfoLeak_{c \to c'}([t-\Delta, t])$

subject to a fixed migration budget

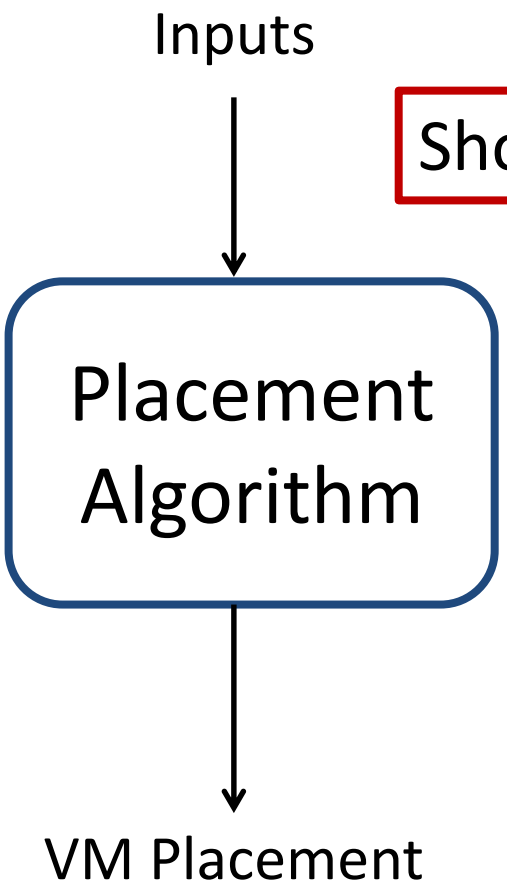
F(Network Capacity)

Challenge: Scalability



Should handle tens of thousands of servers

Challenge: Scalability

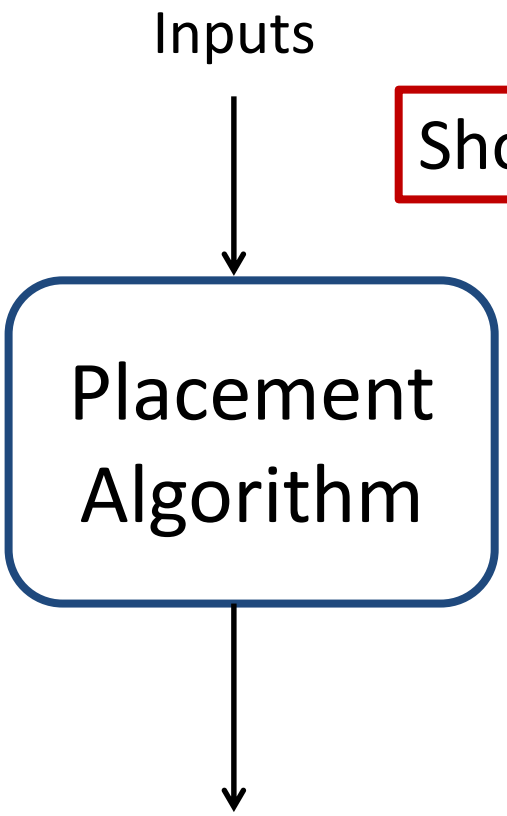


Should handle tens of thousands of servers

- ILP (Integer Linear Programming)

For 40 machines, D > 1 day

Challenge: Scalability

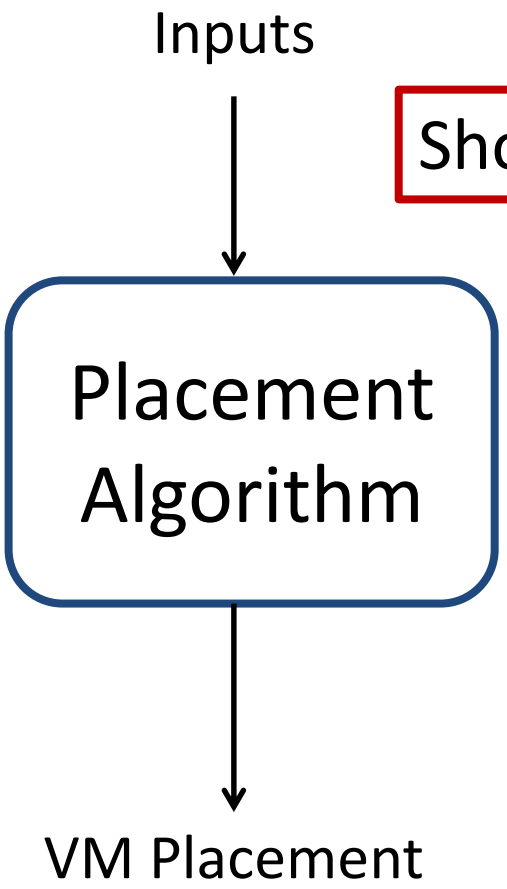


Should handle tens of thousands of servers

- ~~ILP (Integer Linear Programming)~~

~~For 40 machines, D > 1 day~~

Challenge: Scalability



Should handle tens of thousands of servers

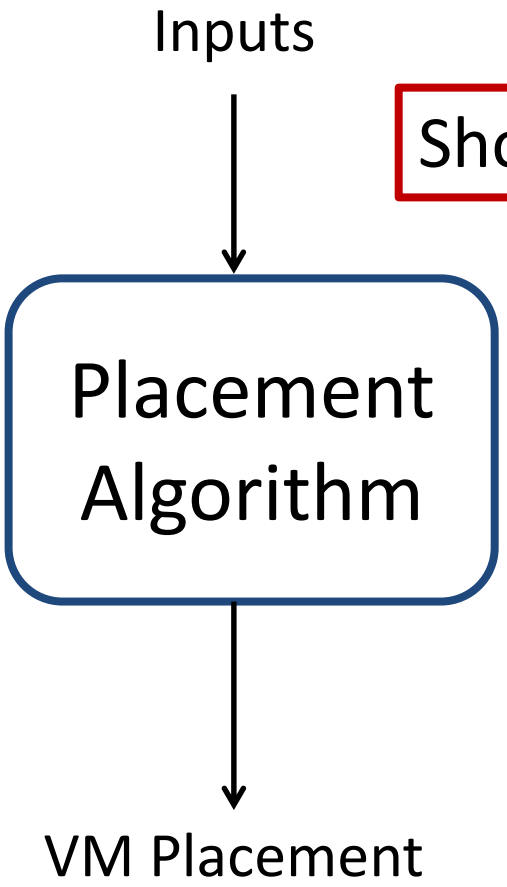
- ~~ILP (Integer Linear Programming)~~

~~For 40 machines, D > 1 day~~

- Basic Greedy

For 400 machines, D > 1 day

Challenge: Scalability



Should handle tens of thousands of servers

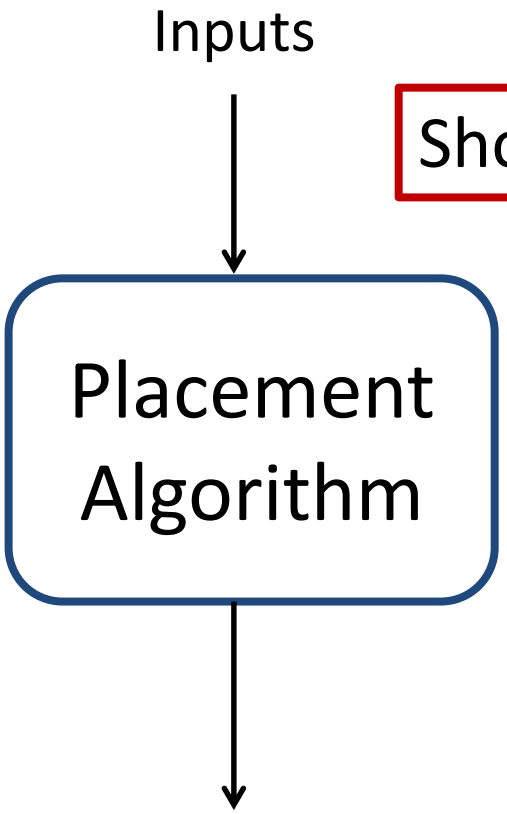
- ~~ILP (Integer Linear Programming)~~

~~For 40 machines, D > 1 day~~

- ~~Basic Greedy~~

~~For 400 machines, D > 1 day~~

Challenge: Scalability



Should handle tens of thousands of servers

- ~~ILP (Integer Linear Programming)~~

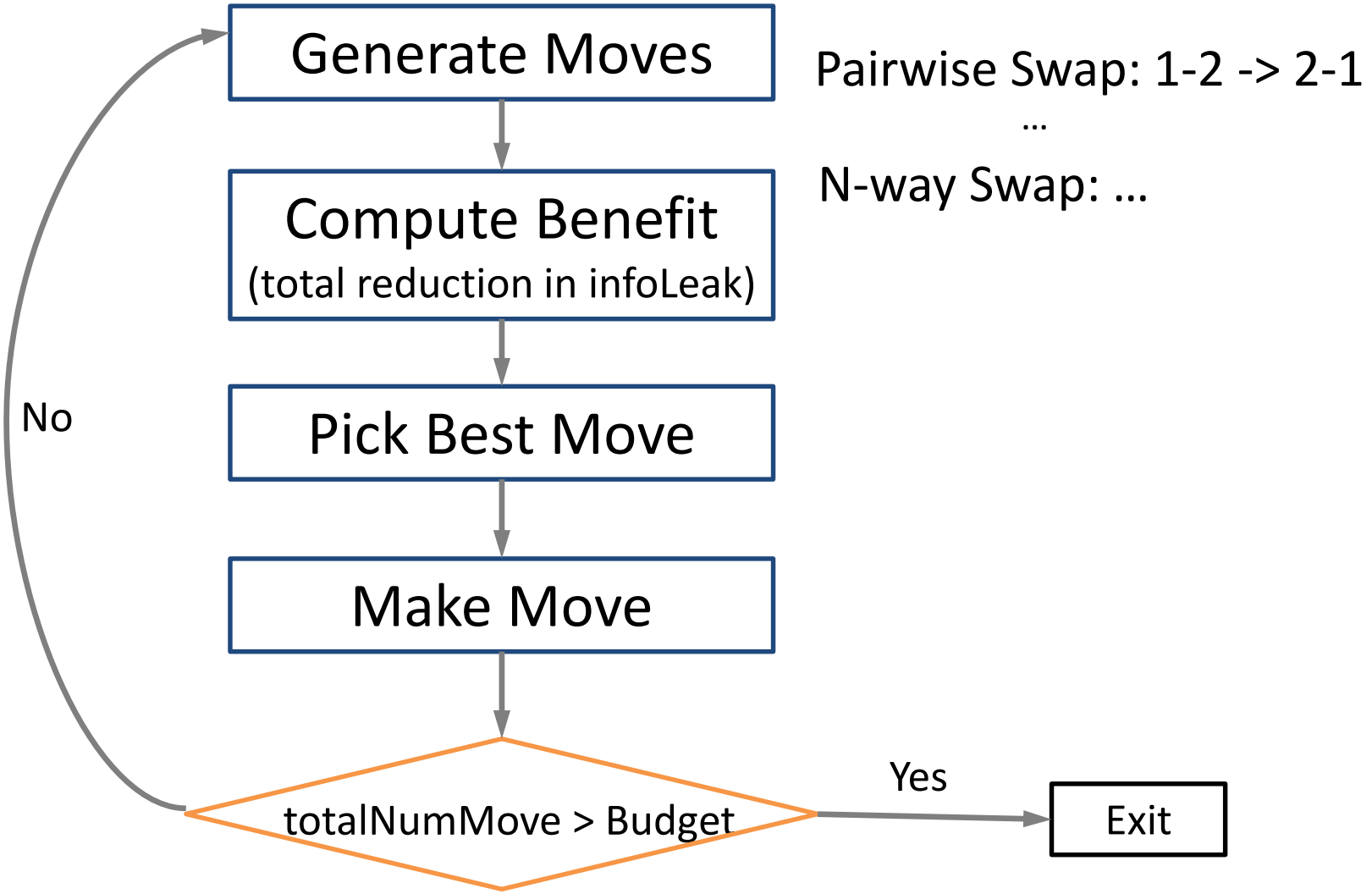
~~For 40 machines, D > 1 day~~

- ~~Basic Greedy~~

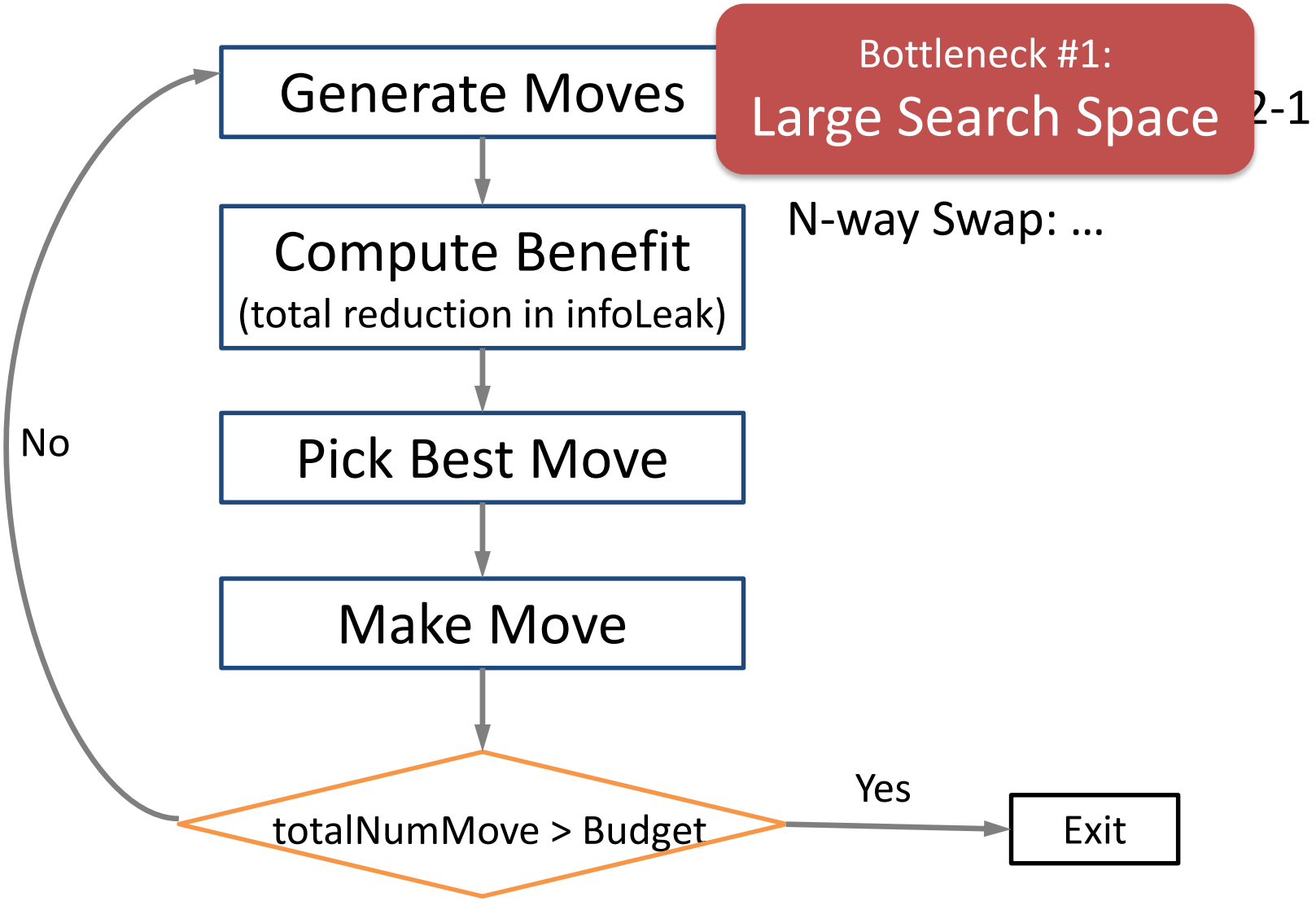
~~For 400 machines, D > 1 day~~

- Basic Greedy with our optimizations

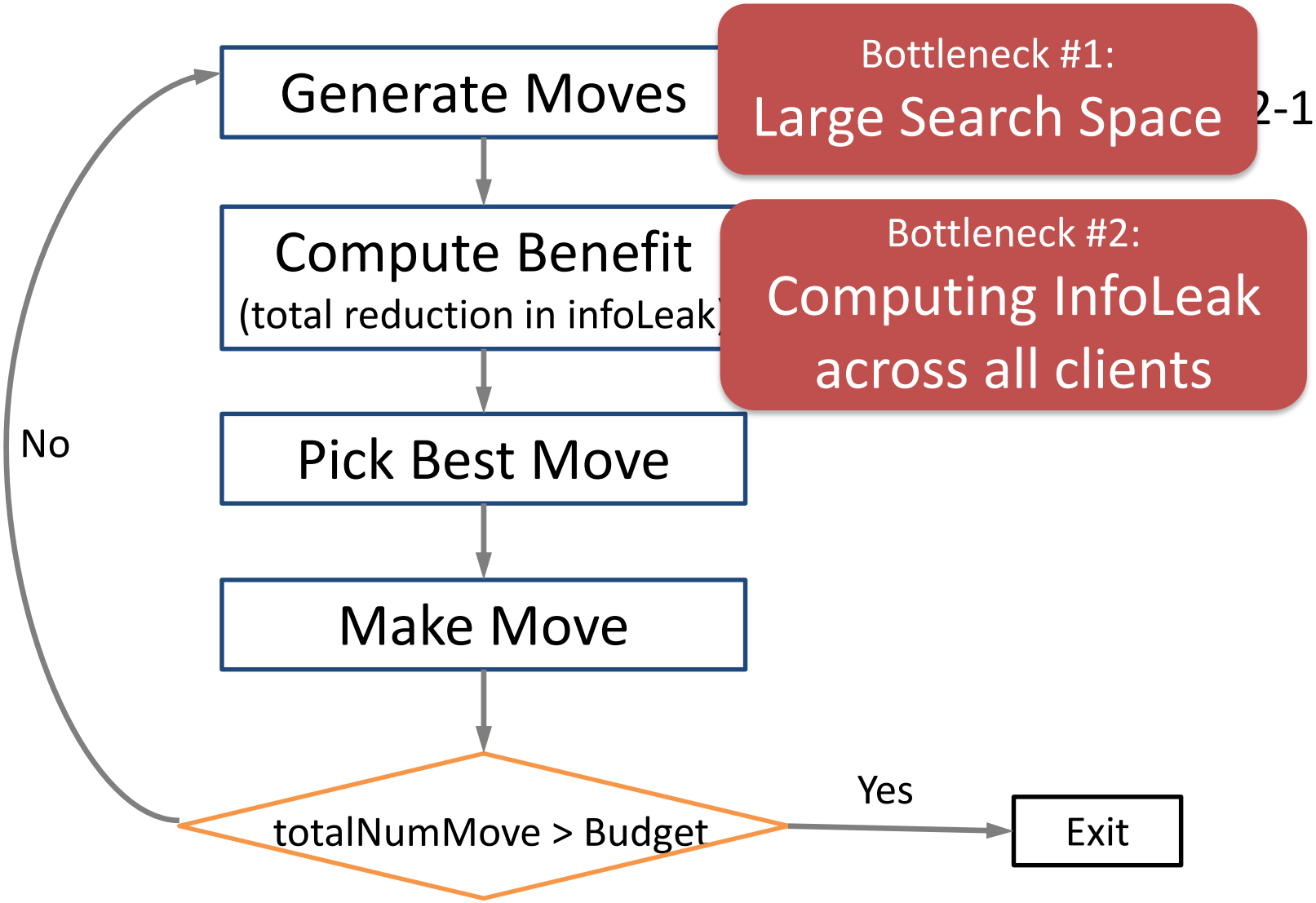
Why is Basic Greedy not scalable?



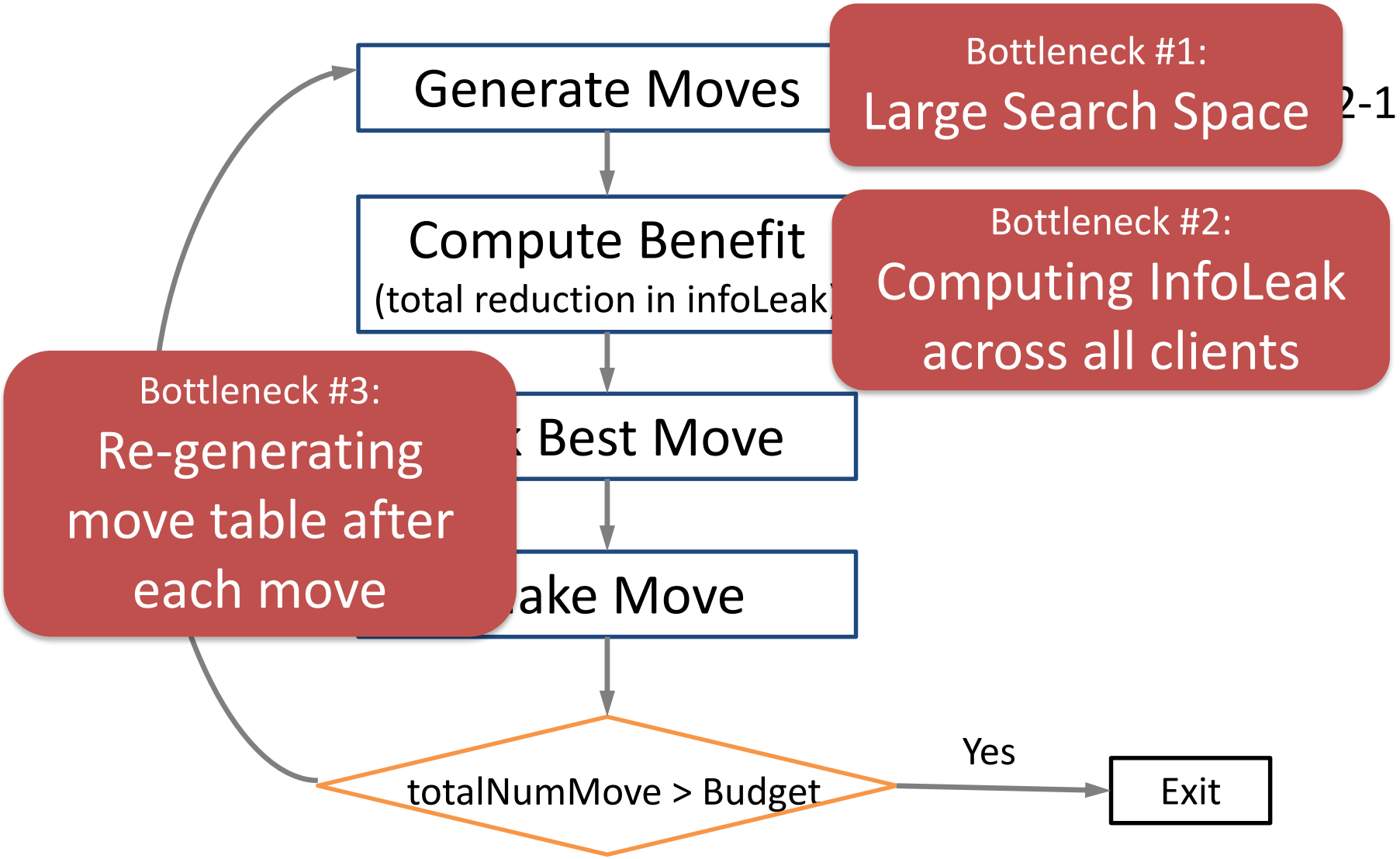
Why is Basic Greedy not scalable?



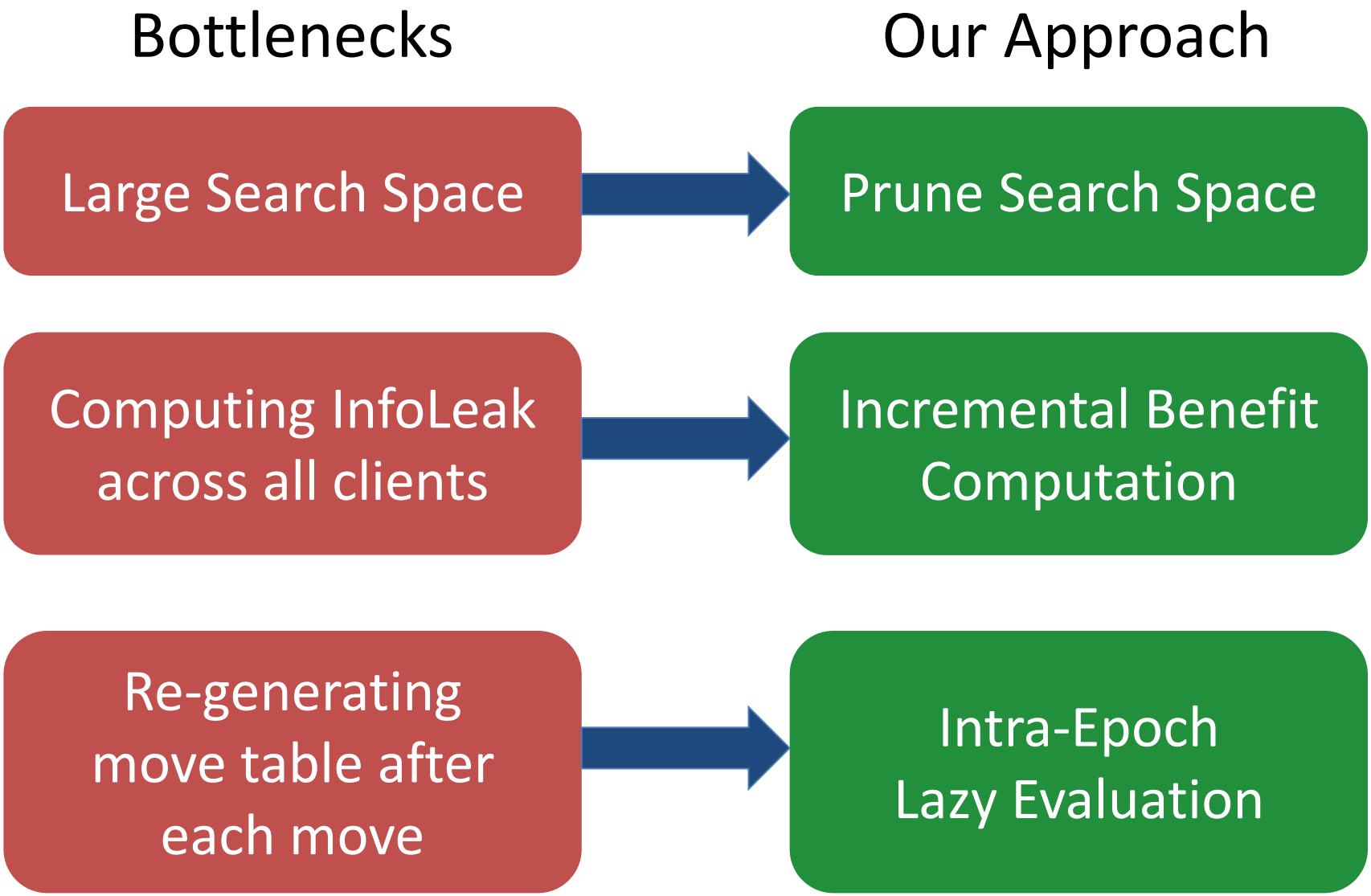
Why is Basic Greedy not scalable?



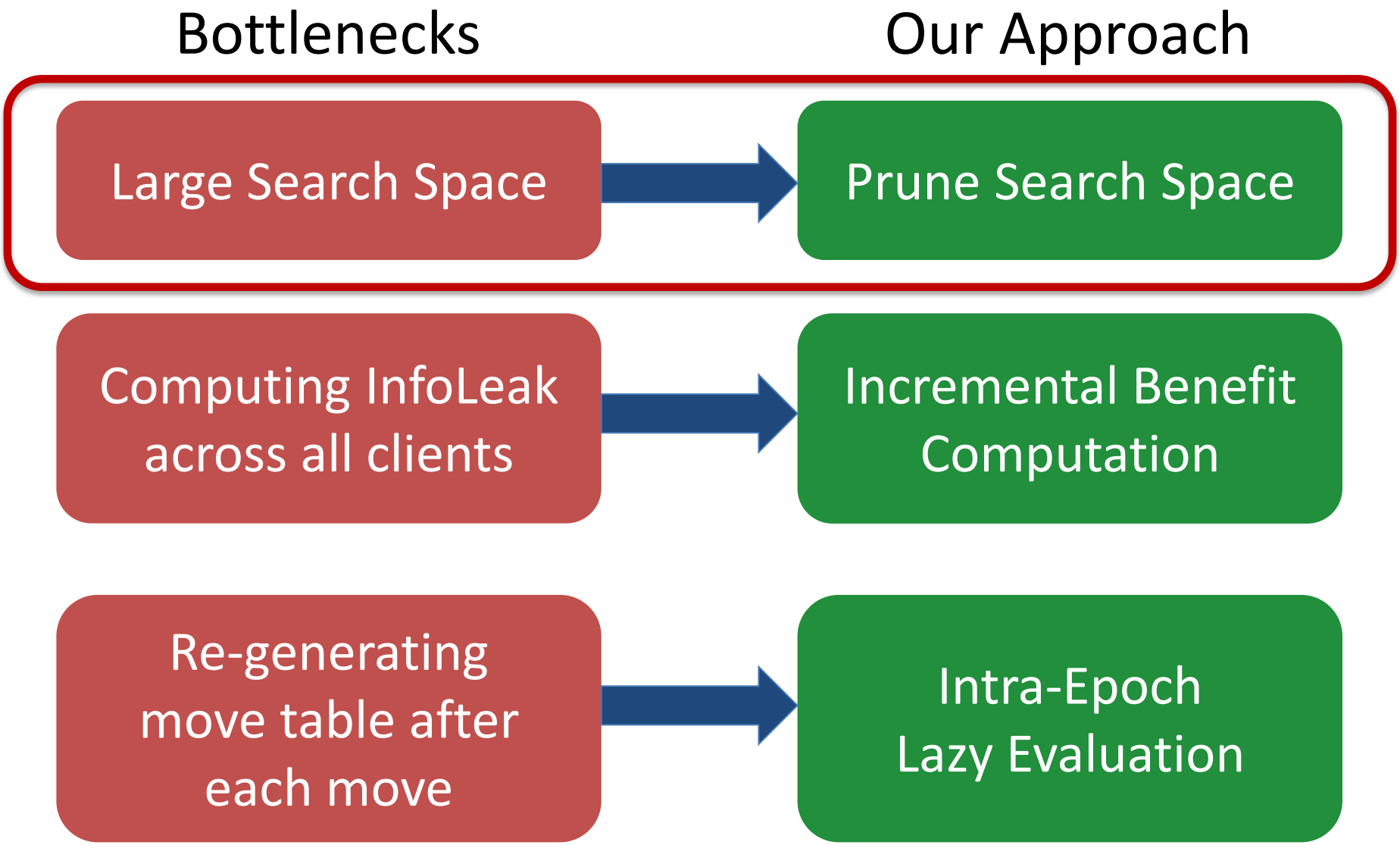
Why is Basic Greedy not scalable?



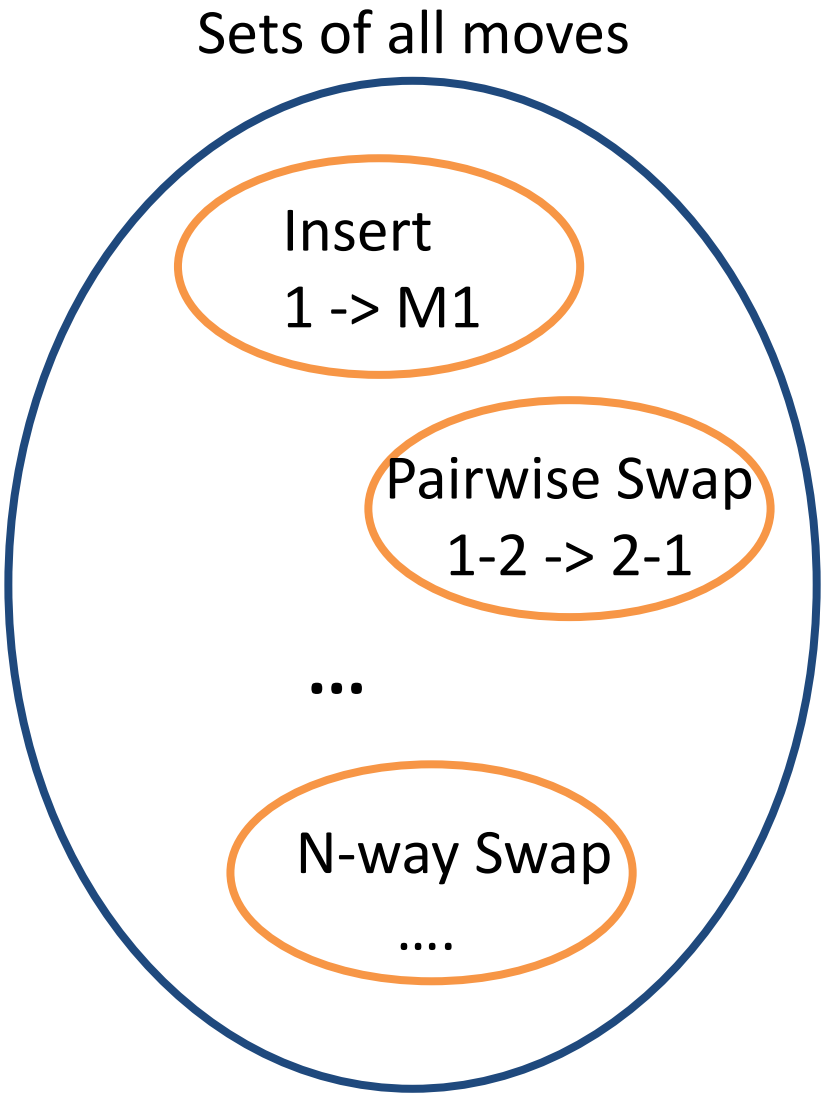
Our Approach



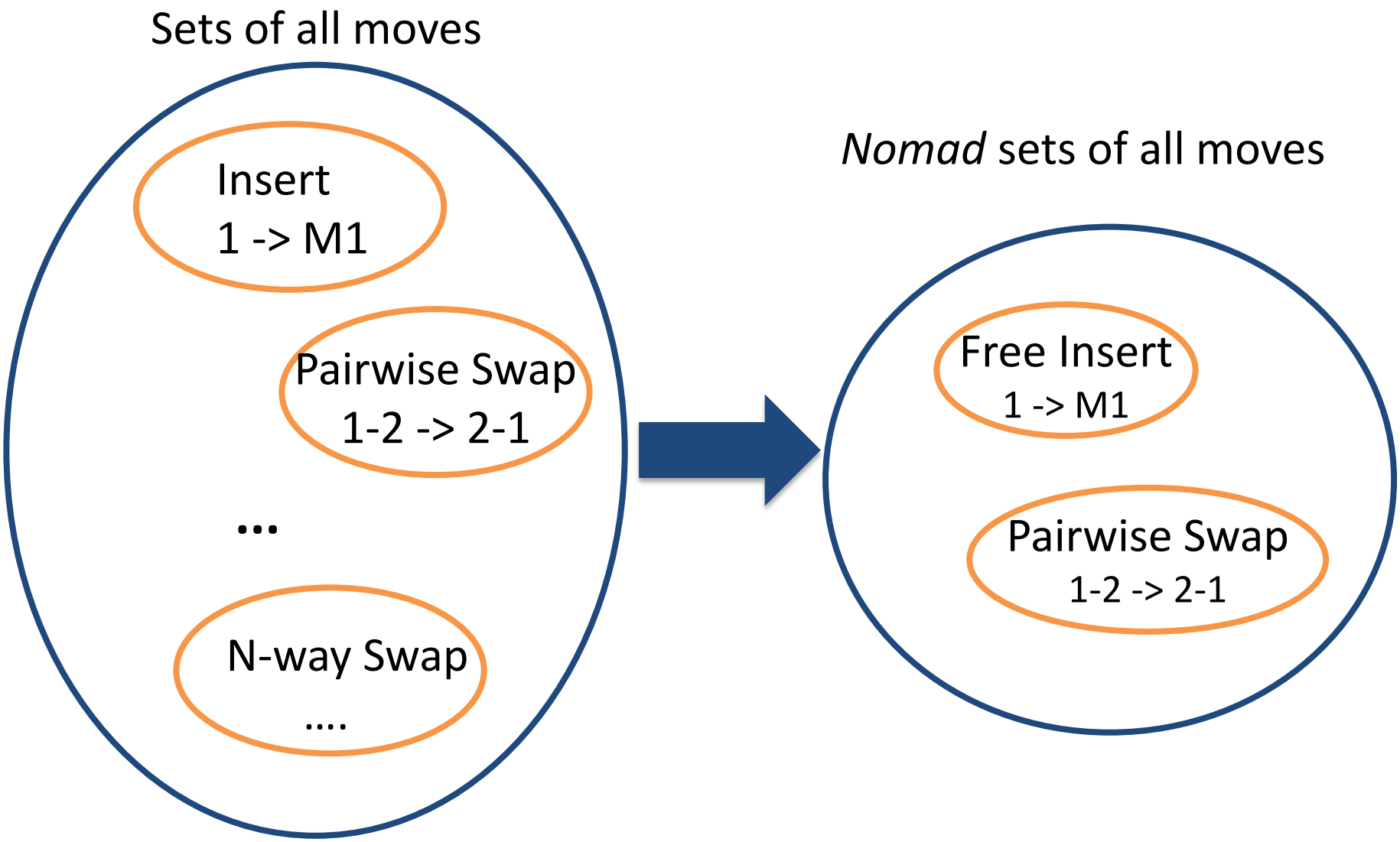
Our Approach



Prune #1: Pruning Move Space

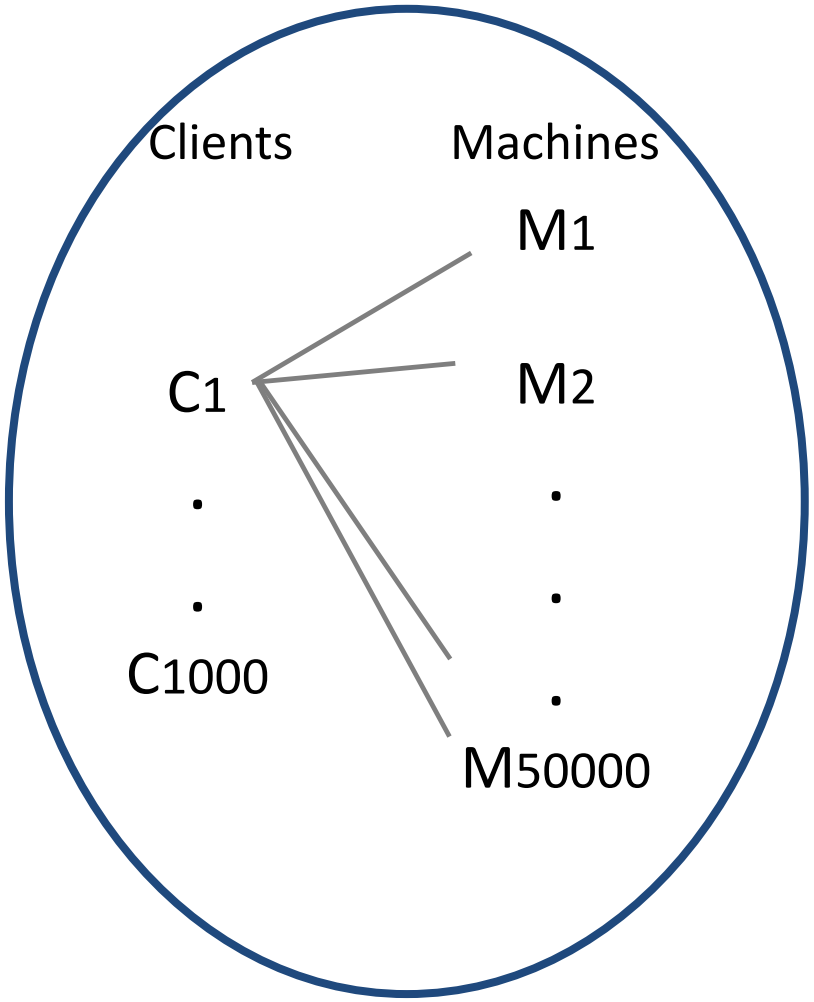


Prune #1: Pruning Move Space



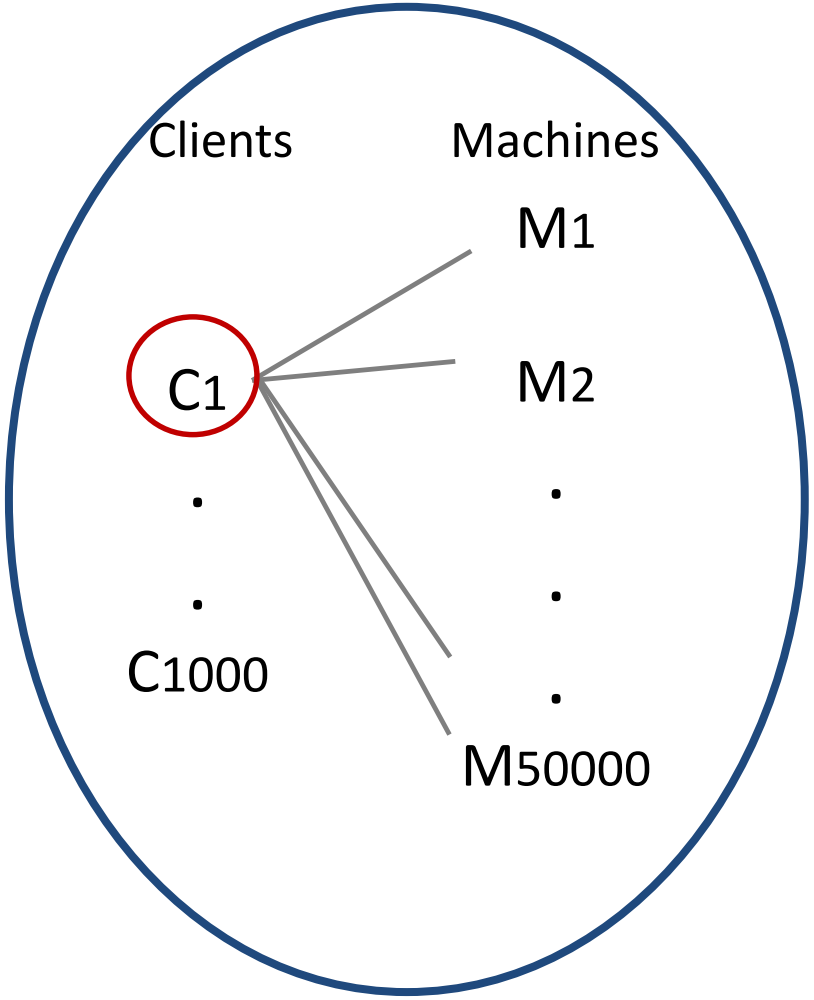
Prune #2: Hierarchical Decomposition

Sets of all free inserts



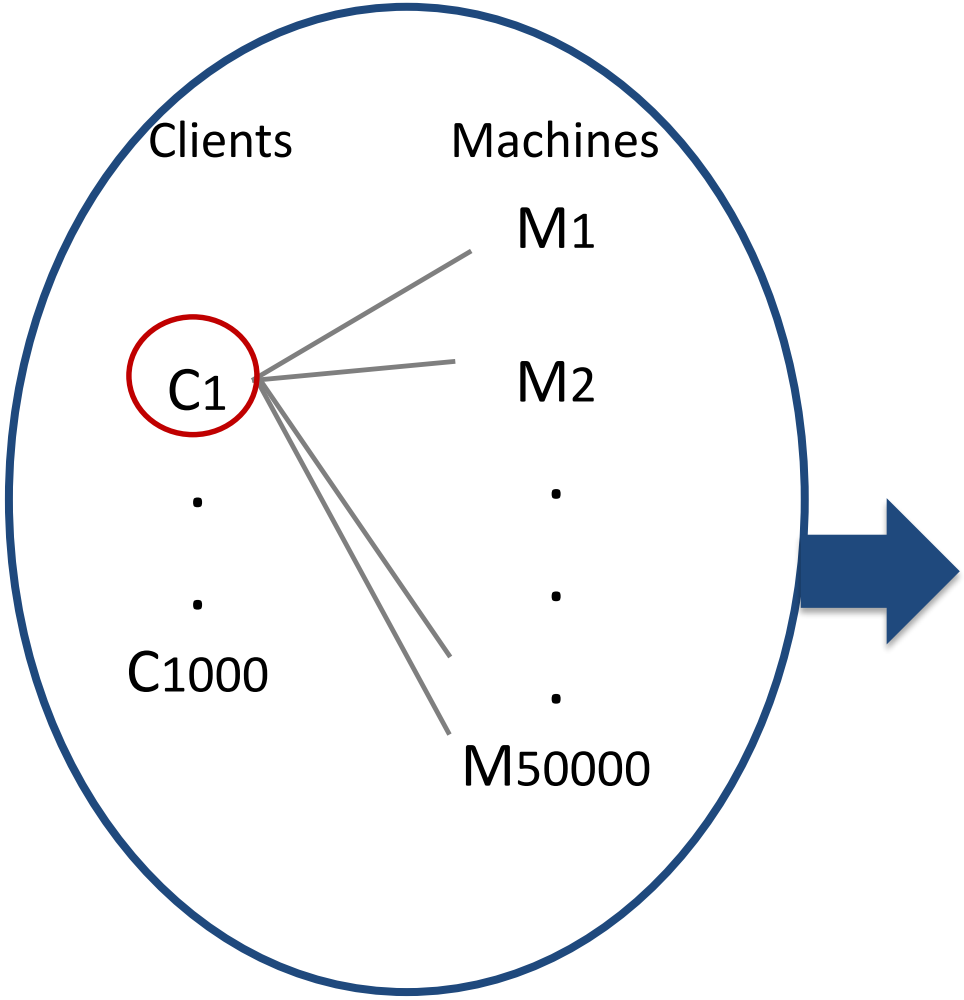
Prune #2: Hierarchical Decomposition

Sets of all free inserts

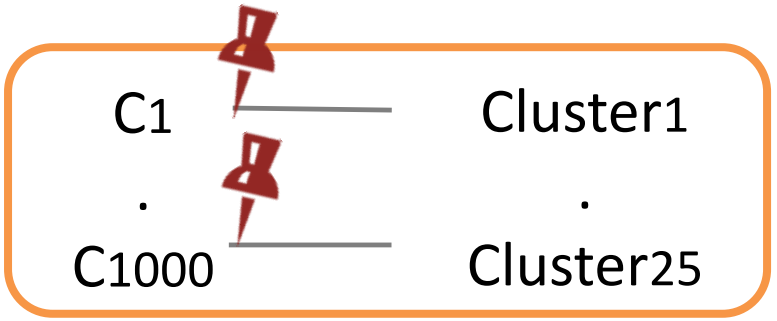


Prune #2: Hierarchical Decomposition

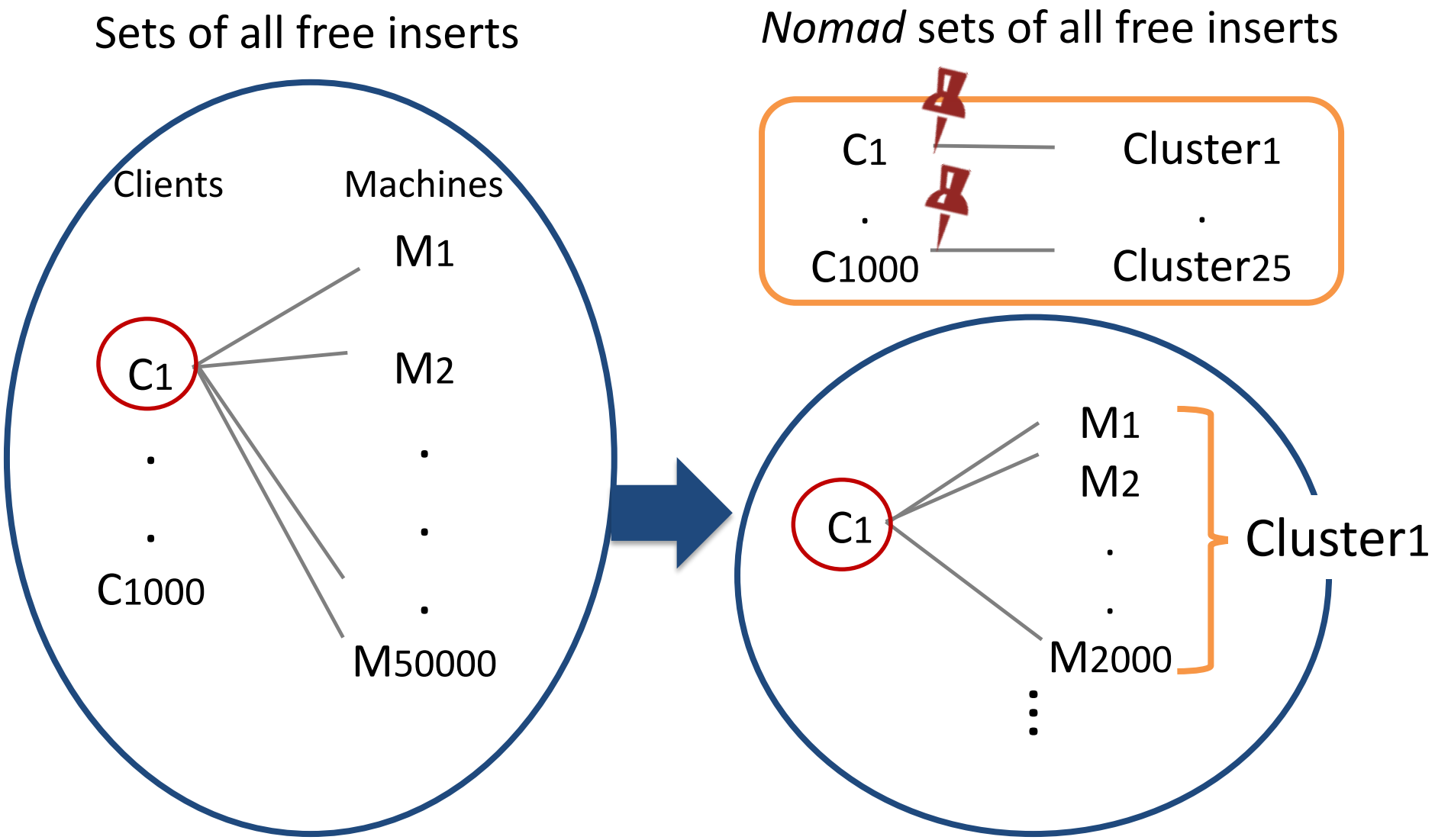
Sets of all free inserts



Nomad sets of all free inserts



Prune #2: Hierarchical Decomposition



Our Work

1. Idea

General side-channel defense via migration

2. Logic

Characterize information leakage due to co-residency

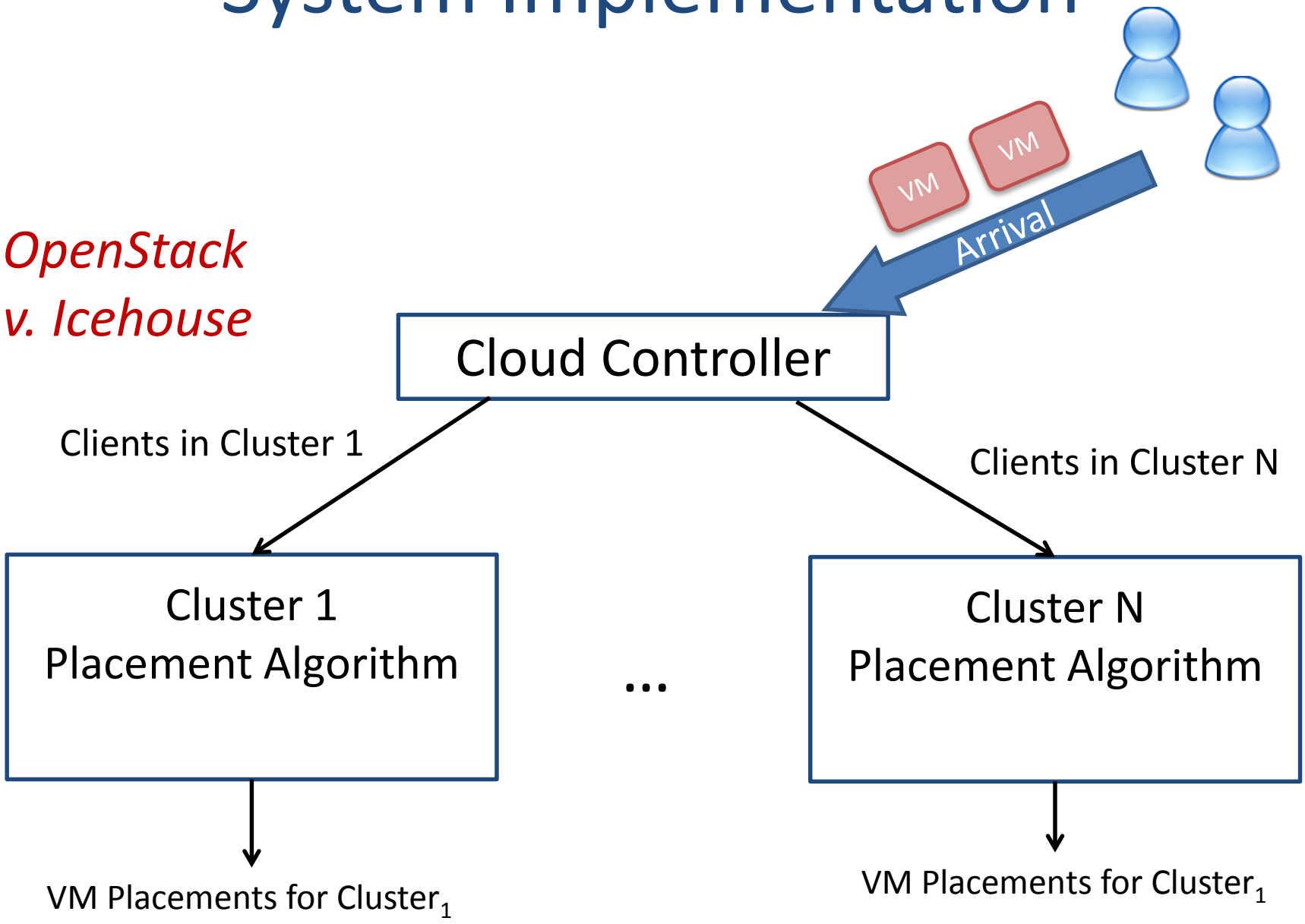
3. Scalable Design

Scalable VM migration strategy that can handle large cloud deployments

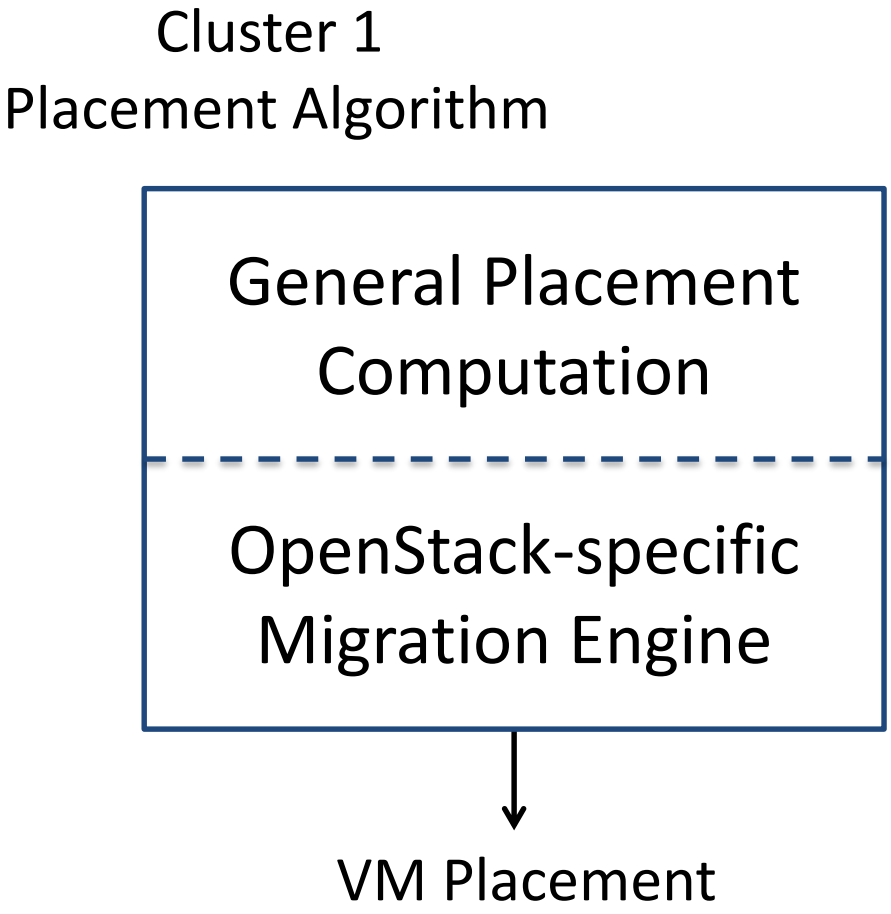
4. Practical Impact

Practical OpenStack implementation with minimal modifications

System Implementation



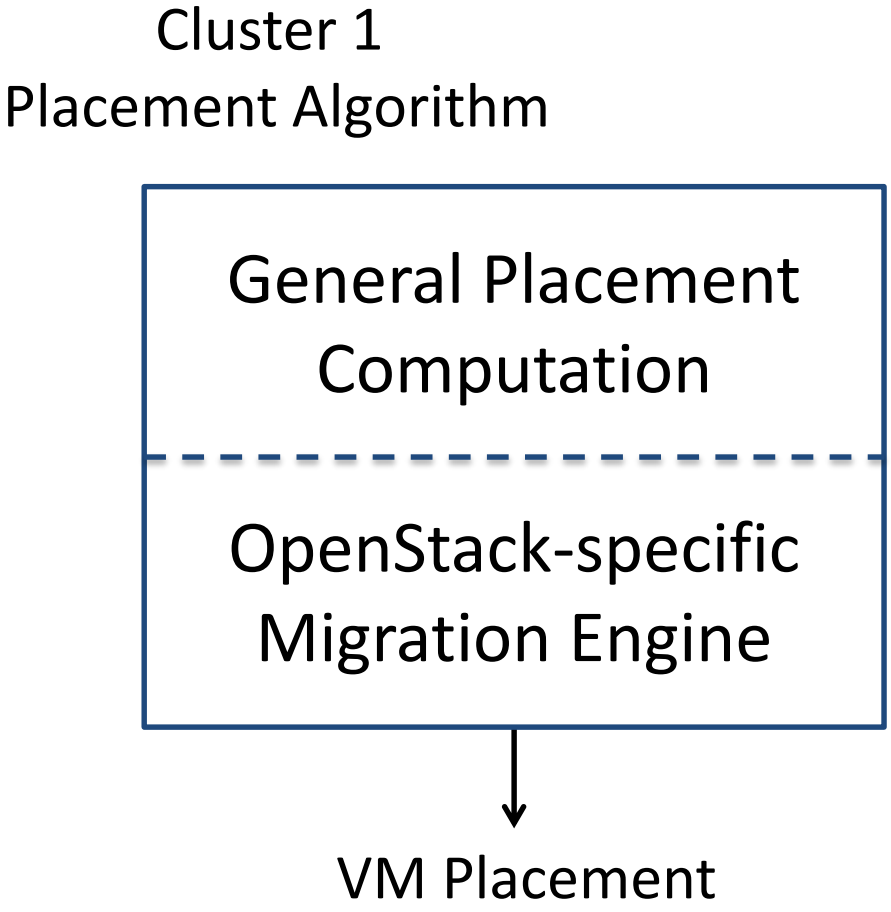
System Implementation (One Cluster)



Custom C++
~2000 LOC

OpenStack Icehouse:
~200 LOC in Controller Scheduler code

System Implementation (One Cluster)



Custom C++
~2000 LOC

OpenStack Icehouse:
~200 LOC in Controller Scheduler code

Requires minimal modifications to existing deployments

Key Evaluation Questions

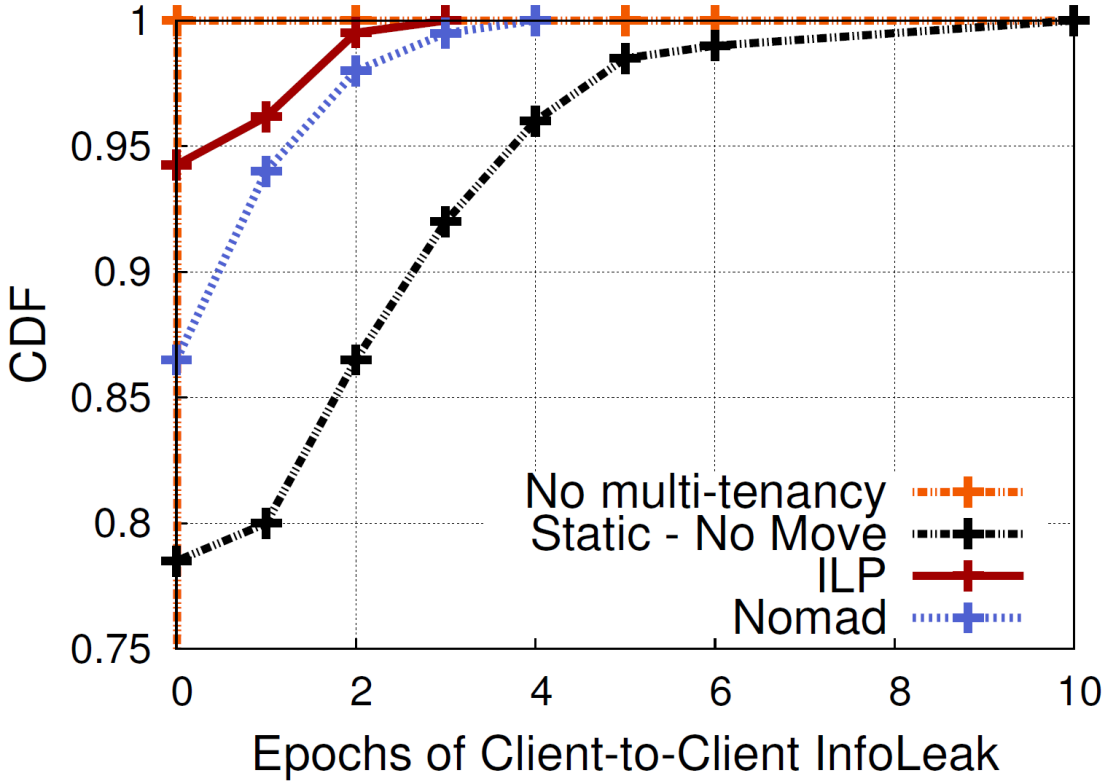
- Information leakage resilience
- Scalability
- Impact on cloud applications
- Benefit/Cost of each design idea
- Resilience to strategic adversary

Information Leakage resilience

<R,C>: Problem size of 20-machines

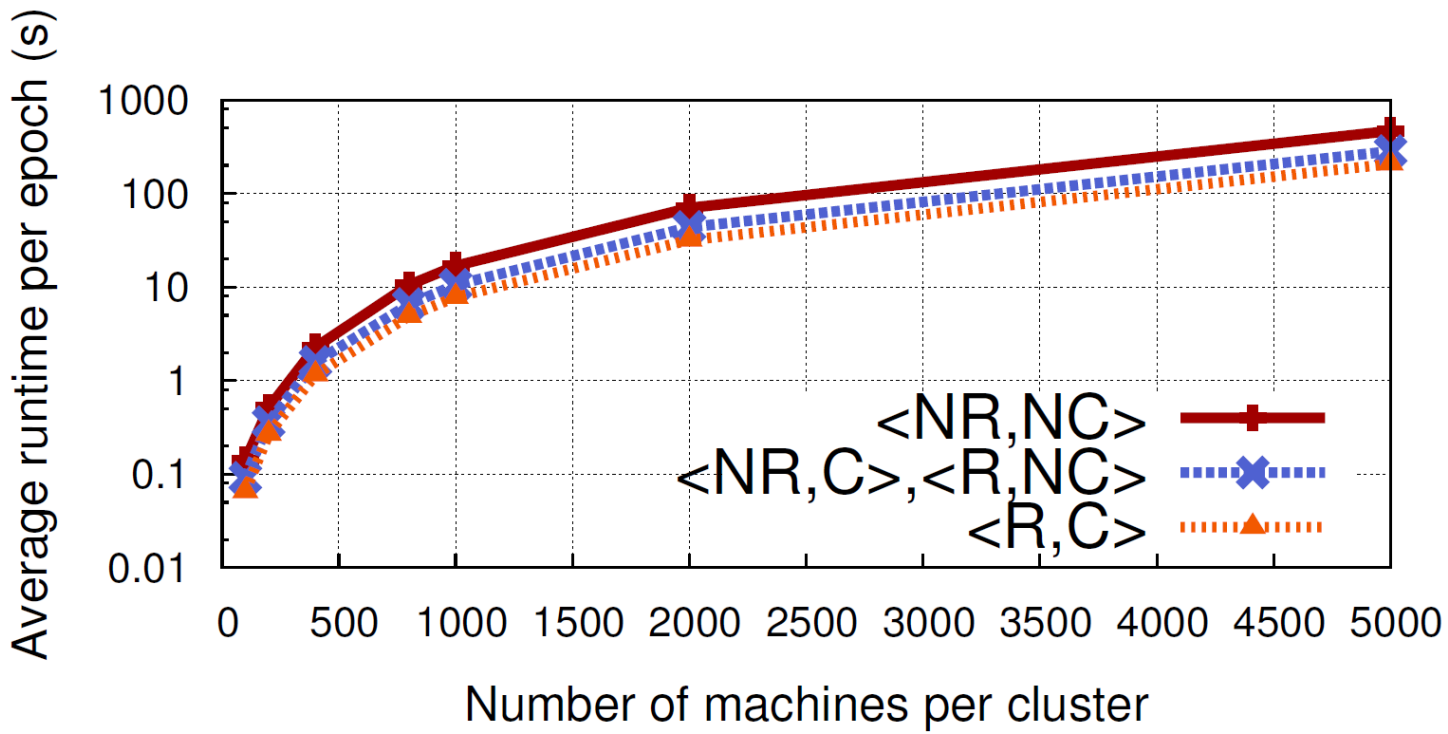
Metric:

$InfoLeak_{c \rightarrow c'}([t-\Delta, t])$



Nomad brings ~4.5x reduction in InfoLeak for 98th percentile compared to static w.r.t. ILP.

Scalability



Nomad placement algorithm is scalable to large deployments

Impact on cloud applications

Replicated web-server (Wikibench)

- Each client : 3 replicated web servers, 1 worker
  - In one epoch, *at least* 1 server migrates

$$Norm.Throughput\ (Norm.T) = \frac{T_{w/o} - T_w}{T_{w/o}} x\ 100$$

- Overhead (Norm. T)
  - ~0% for 95th Norm T.
  - 0.096% for 50th (median) Norm. T.
  - 1.8% for 5th Norm. T

Discussion

- Fast side-channel attacks
  - Need out-of-band defense
  - e.g., introduce cache noise, refresh secret
- Network Impact
  - With techniques like incremental diffs, the transfer size is much less than base VM image
- Incentives for adoption
  - Security-conscious clients opt-in
  - Providers have new revenue streams
- More opportunities
  - Fairness across clients

Conclusions

- Co-residency side-channel attacks: real/growing threats

Current World : 

No Migration

1. Per-attack fixes
2. Require significant upgrades

Nomad: 

"Migration-as-a-Service"

1. General solution
2. Needs minimal changes

- *Nomad* achieves:
  - Information leakage resilience close to the ILP
  - Scalable VM placement algorithm
  - Practical system atop OpenStack with minimal modifications