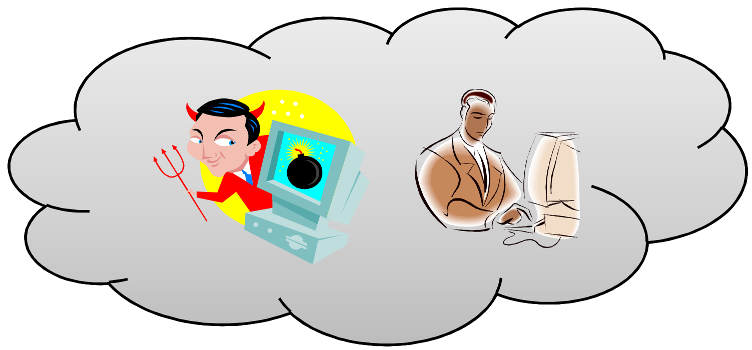
Hey, You, Get Off of My Cloud!

Exploring Information Leakage in Third-Party Clouds



Thomas Ristenpart, Eran Tromer, Hovav Shacham, Stefan Savage

UCSD MIT UCSD UCSD

Today's talk in one slide

Third-party clouds: amazon web services™

"cloud cartography" to map internal infrastructure

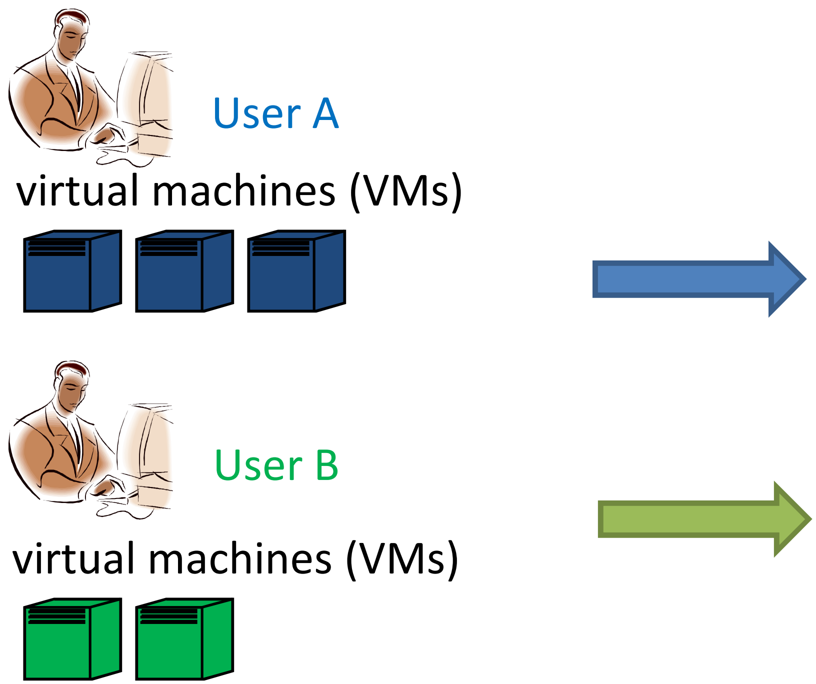
get malicious VM on same physical server as victim

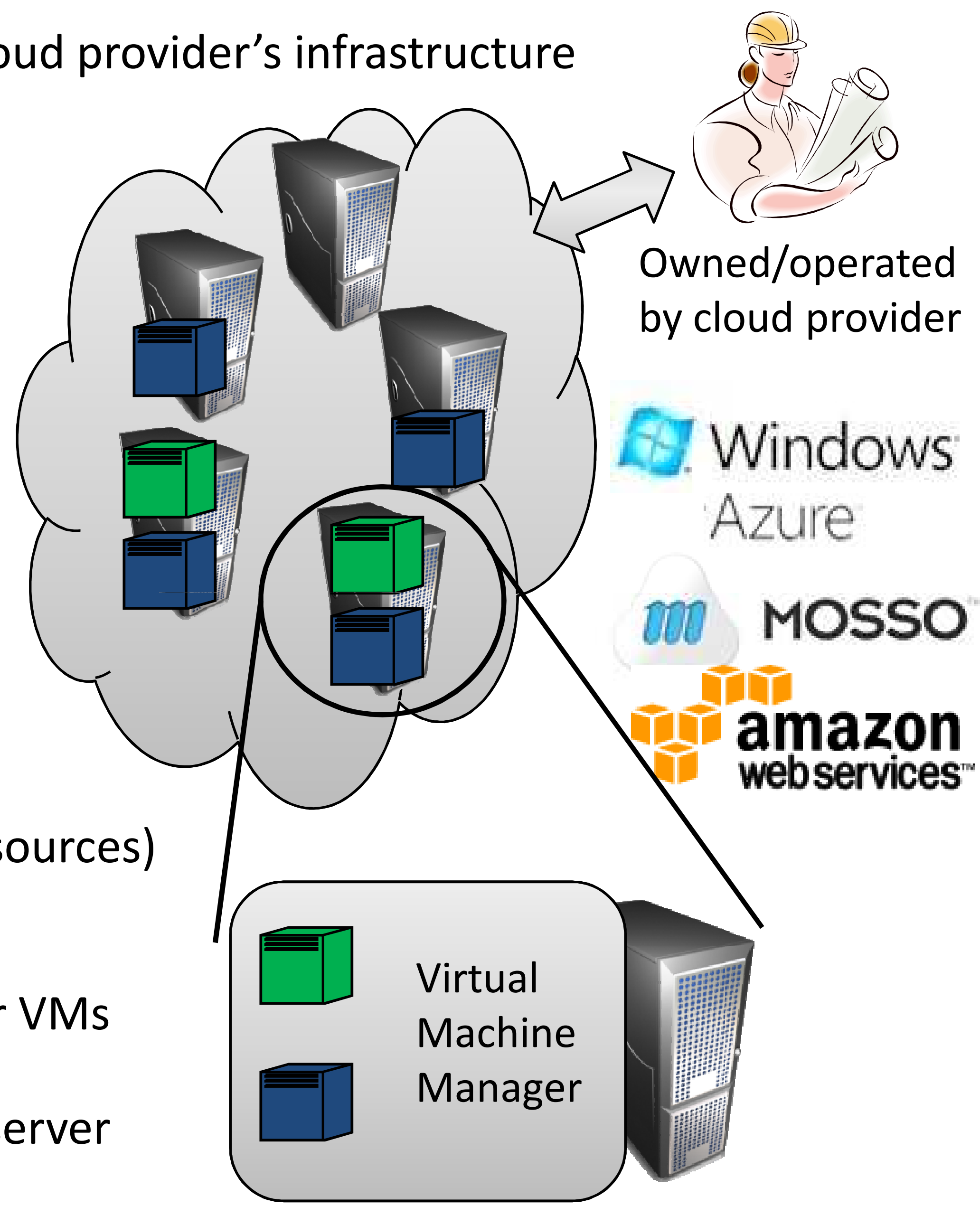
side-channels might leak confidential data of victim

Exploiting a placement vulnerability: only use cloud-provided functionality

A simplified model of third-party cloud computing

Users run Virtual Machines (VMs) on cloud provider's infrastructure



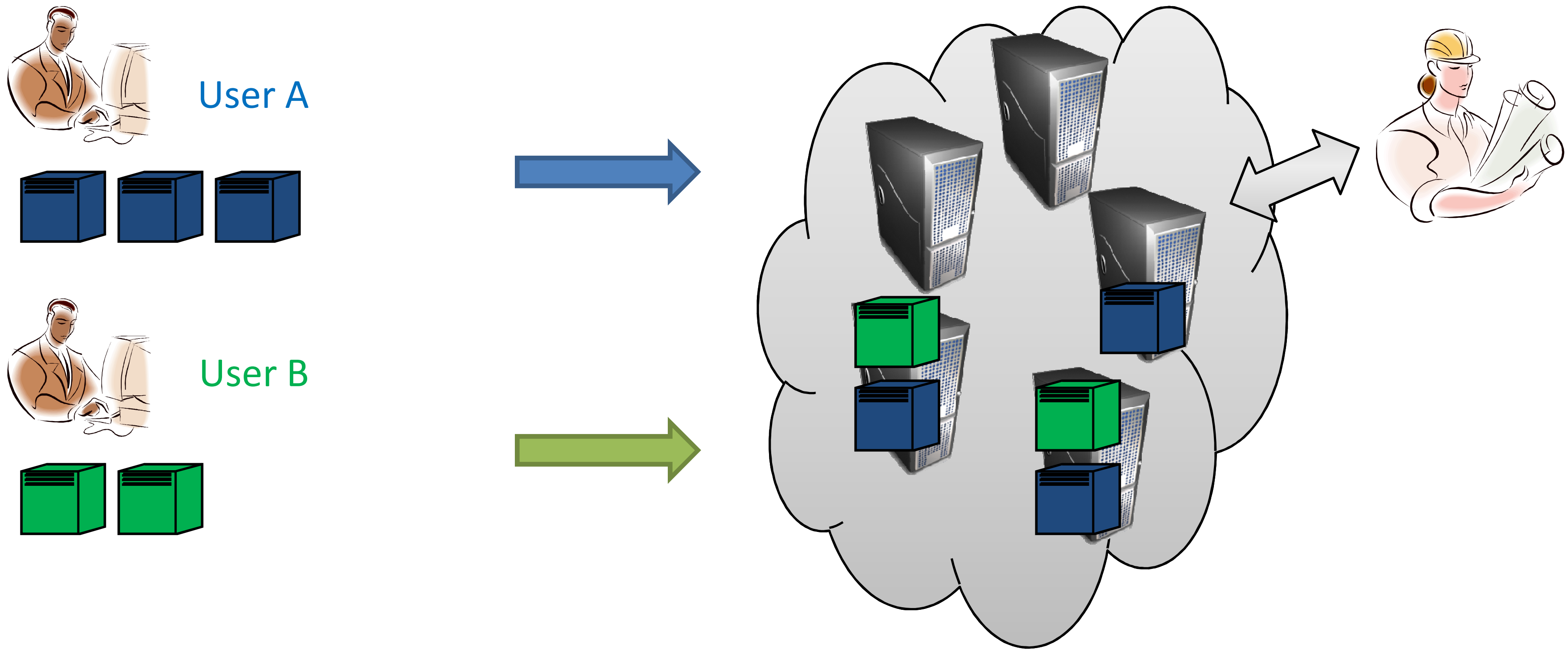


Multitenancy (users share physical resources)

Virtual Machine Manager (VMM)
manages physical server resources for VMs

To the VM should look like dedicated server

Trust models in cloud computing



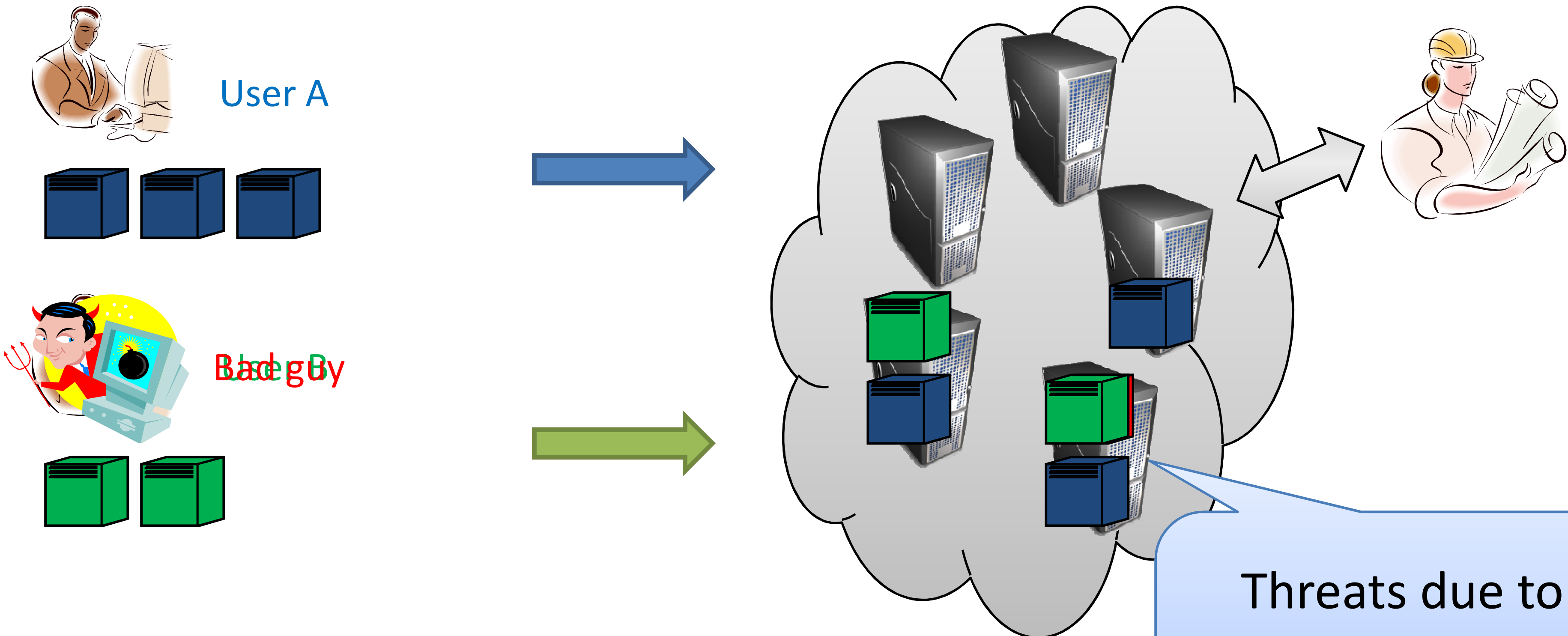
Users must trust third-party provider to

not spy on running VMs / data

secure infrastructure from external attackers

secure infrastructure from internal attackers

Trust models in cloud computing

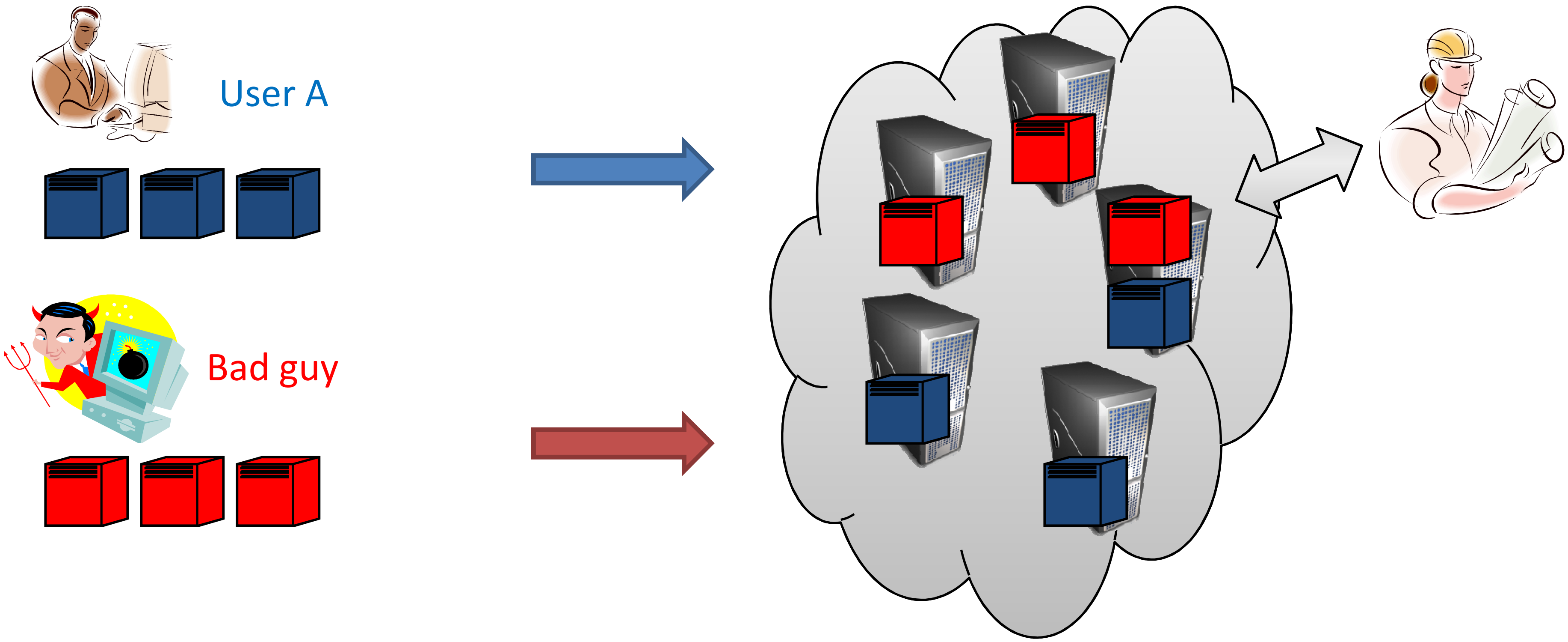


Threats due to sharing of physical infrastructure ?

Users must trust third-party provider to

- not spy on running VMs / data
- secure infrastructure from external attackers
- secure infrastructure from internal attackers
  - Your business competitor
  - Script kiddies
  - Criminals
  - …

We explore a new threat model:



Attacker identifies one or more victims VMs in cloud

1) Achieve advantageous placement

- Attacker launches VMs
- VMs each check for co-residence on same server as victim

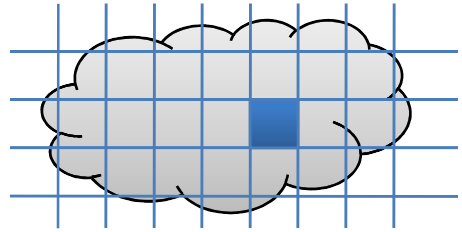
2) Launch attacks using physical proximity

Exploit VMM vulnerability — DoS — Side-channel attack

Using Amazon EC2 as a case study:

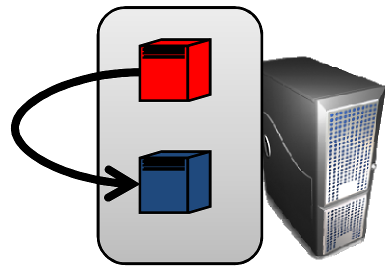
1) Cloud cartography

map internal infrastructure of cloud
map used to locate targets in cloud



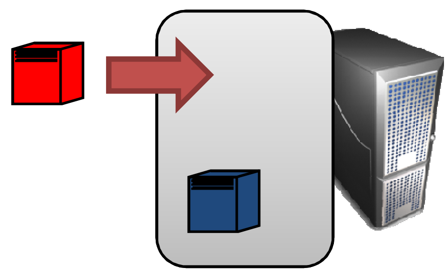
2) Checking for co-residence

check that VM is on same server as target
- network-based co-residence checks
- efficacy confirmed by covert channels



3) Achieving co-residence

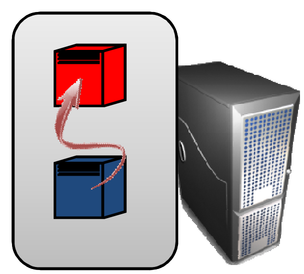
brute forcing placement
instance flooding after target launches



Placement vulnerability: attackers can knowingly achieve co-residence with target

4) Side-channel information leakage

coarse-grained cache-contention channels
might leak confidential information



What our results mean is that

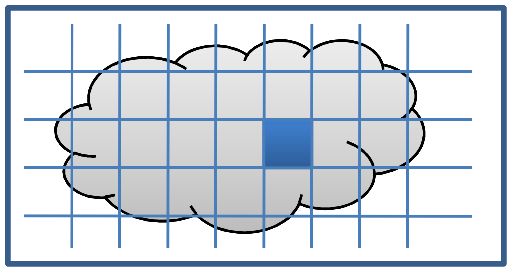
1) given no insider information

2) restricted by (the spirit of) Amazon's acceptable use policy (AUP)

(using only Amazon's customer APIs and very restricted network probing)

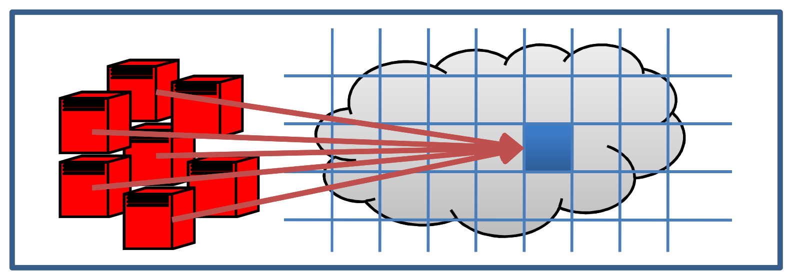
we can:

Pick target(s)

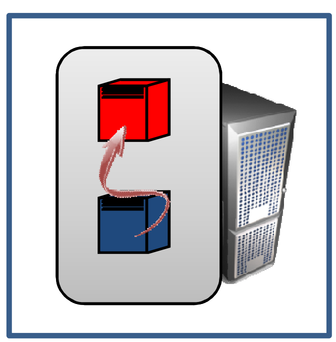


Choose launch parameters for malicious VMs

Each VM checks for co-residence



Given successful placement, spy on victim web server's traffic patterns via side channels



Before we get into details of case study:

Should I panic?

No. We didn't show how to extract cryptographic keys

But:

We exhibit side-channels to measure load across VMs in EC2

Coarser versions of channels used to extract cryptographic keys

Other clouds?

We haven't investigated other clouds

Problems only in EC2?

EC2 network configuration made cartography and co-residence checking easy

But:

These don't seem critical to success

Placement vulnerabilities seem inherent issue when using multitenancy

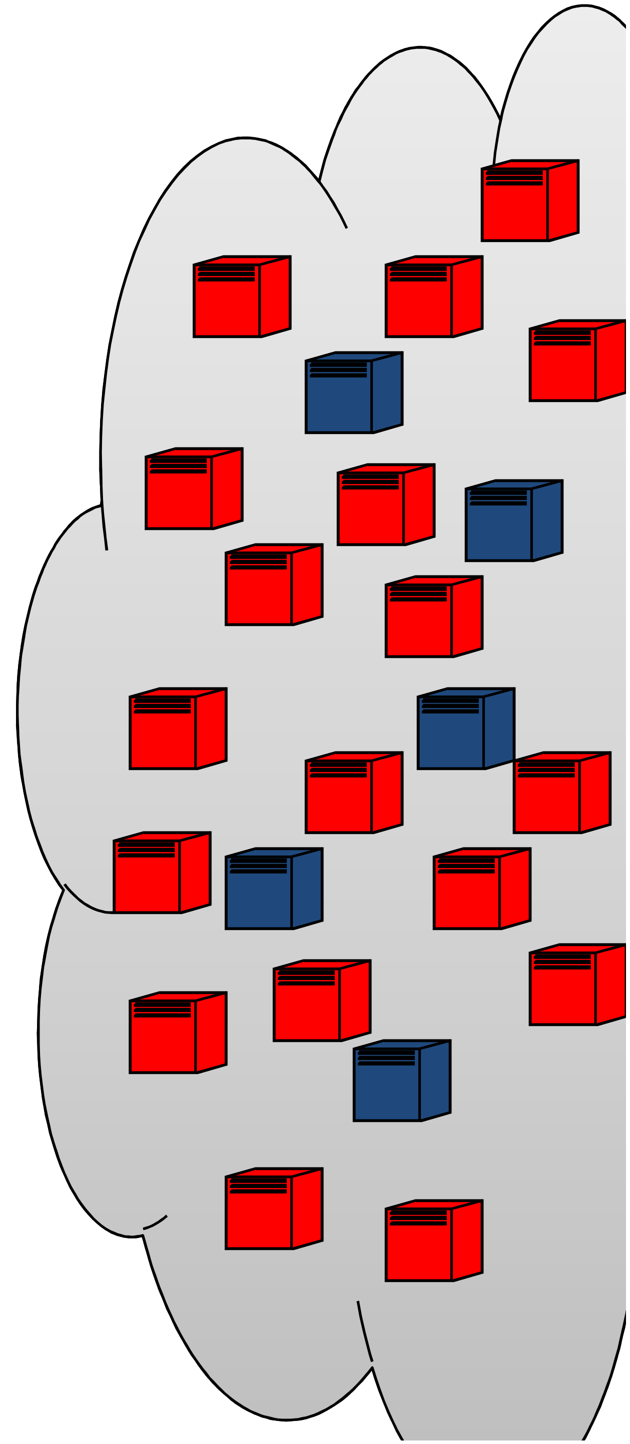
1 or more targets in the cloud and we want to achieve co-resident placement with any of them

Suppose we have an oracle for checking co-residence (we'll realize it later)

Launch lots of instances (over time), each asking oracle if successful

If target set large enough or adversarial resources (time & money) sufficient, this might already work

In practice, we can do much better than this



Some info about EC2 service (at time of study)

Linux-based VMs available
Uses Xen-based VM manager

launch parameters:

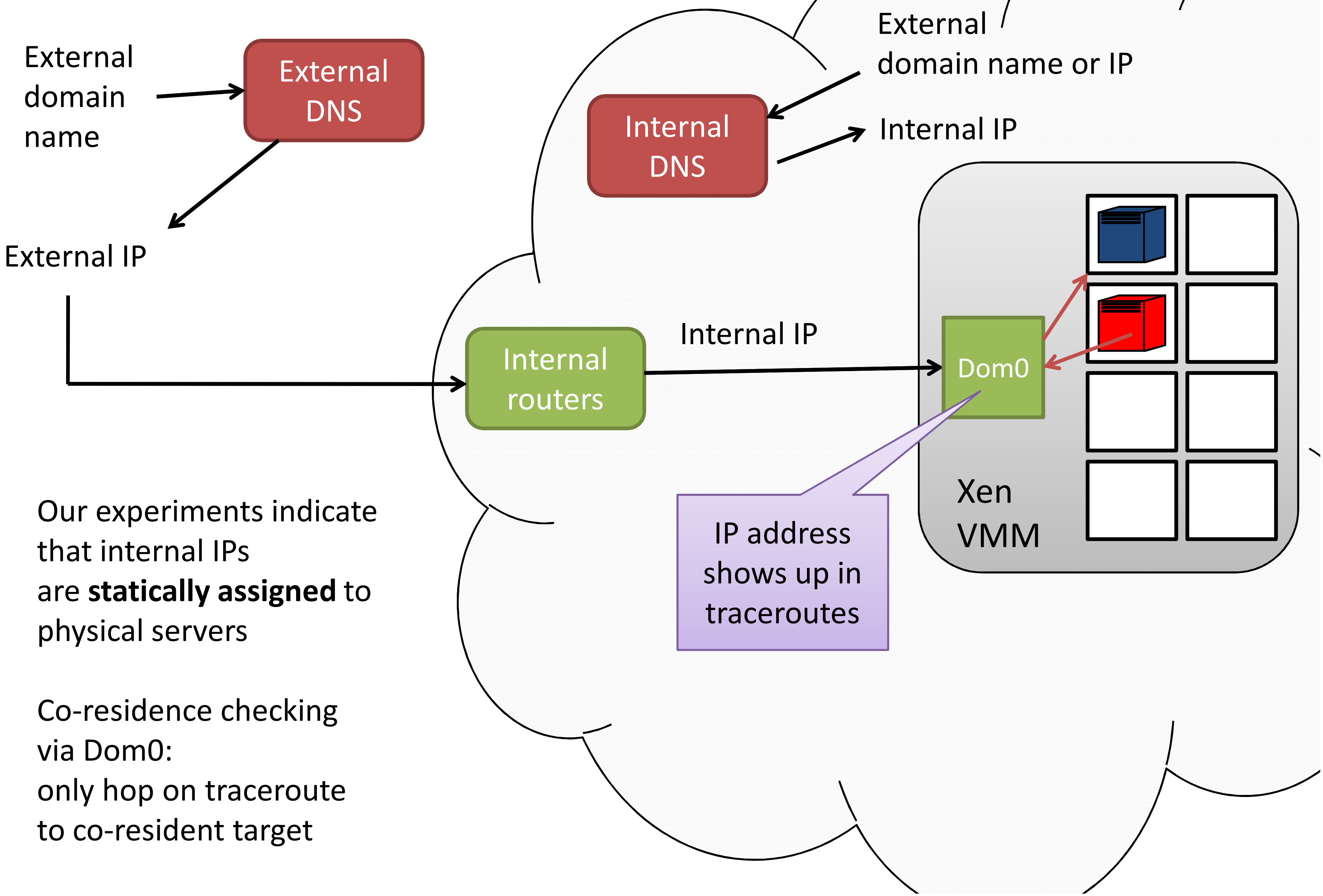
- User account
- 3 “availability zones” (Zone 1, Zone 2, Zone 3)
- 5 instance types (various combinations of virtualized resources)

Type	gigs of RAM	EC2 Compute Units (ECU)
m1.small (default)	1.7	1
m1.large	7.5	4
m1.xlarge	15	8
c1.medium	1.7	5
c1.xlarge	7	20

1 ECU = 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor

Limit of 20 instances at a time per account.
Essentially unlimited accounts with credit card.

(Simplified) EC2 instance networking



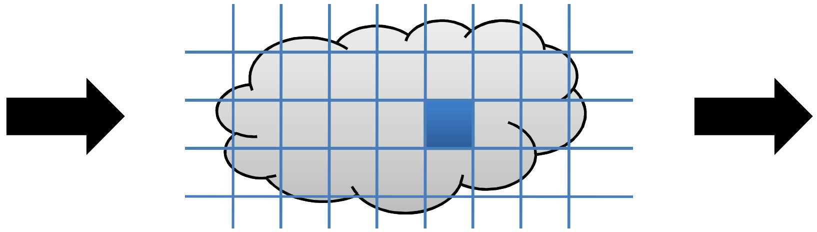
Our experiments indicate that internal IPs are **statically assigned** to physical servers

Co-residence checking via Dom0:
only hop on traceroute to co-resident target

Cloud cartography

Map internal cloud structure to locate targets

Target VM



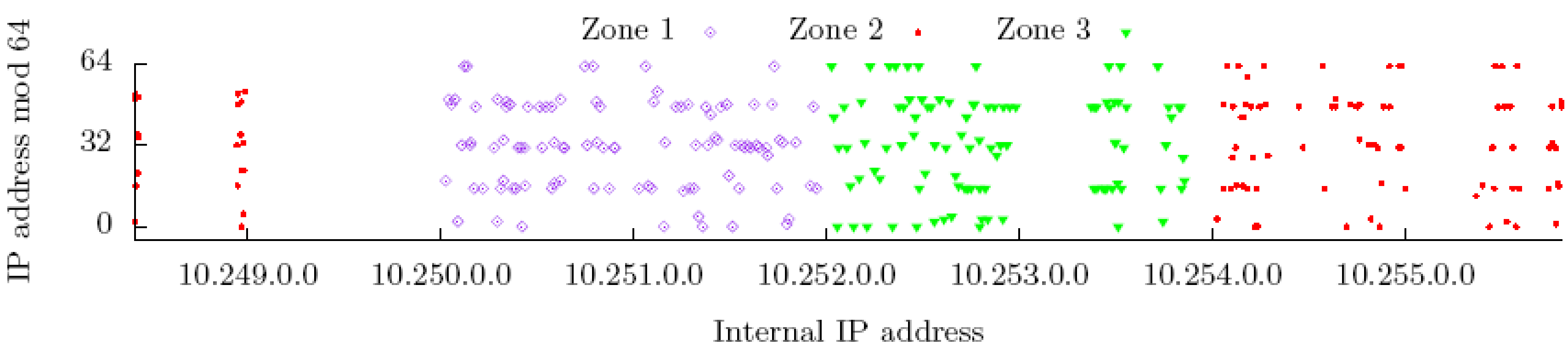
Launch parameters
to achieve co-residence

Towards generating a map, we want to understand affects of launch parameters:

Availability zone   Instance type   Account

From “Account A”:   launch 20 instances of each type in each availability zone

20 x 15 = 300 instances launched



Clean partition of internal IP address space among availability zones

Cloud cartography

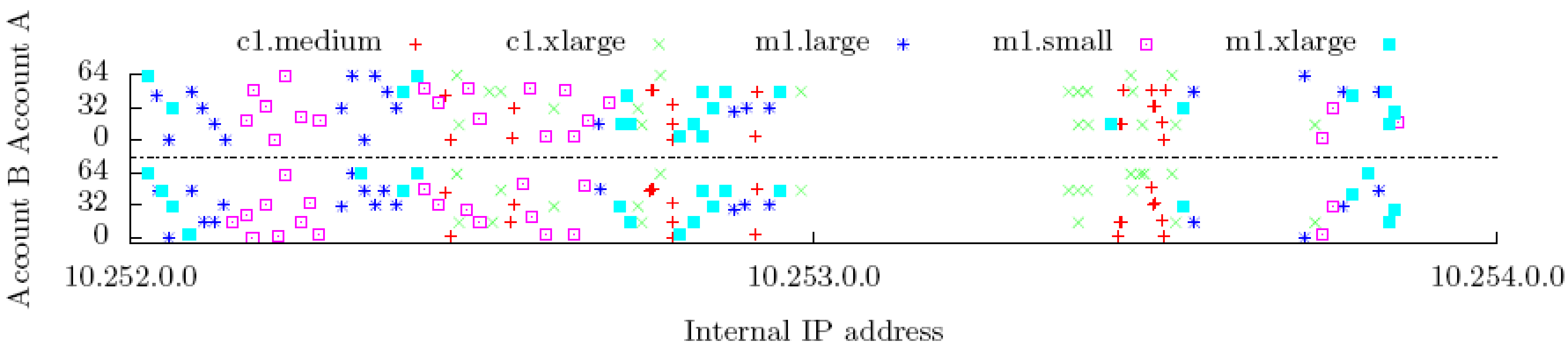
From "Account A": launch 20 instances of each type in each availability zone

20 x 15 = 300 instances launched

From "Account B": launch 20 instances of each type in Zone 3

20 x 5 = 100 instances launched

39 hours apart



55 of 100 Account B instances had IP address assigned to Account A instance

Seems that user account doesn't impact placement

Most /24 associated to single instance type and zone

Associate each /24 with Zone & Type

...
10.251.238.0 zone1 m1.large (ip)
10.251.239.0 zone1 m1.large (scan)
10.251.241.0 zone1 m1.xlarge (scan)
10.251.242.0 zone1 m1.xlarge (ip)
10.251.243.0 zone1 m1.xlarge (scan)
10.252.5.0 zone3 m1.large m1.xlarge (scan)
10.252.6.0 zone3 m1.large m1.xlarge (ip)
10.252.7.0 zone3 m1.large m1.xlarge (scan)
10.252.9.0 zone3 m1.large (ip)
10.252.10.0 zone3 m1.large (ip)
10.252.11.0 zone3 m1.large (scan)
10.252.13.0 zone3 m1.large m1.xlarge (scan)
10.252.14.0 zone3 m1.large (ip)
10.252.15.0 zone3 m1.xlarge (ip)
10.252.21.0 zone3 m1.large (scan)
10.252.22.0 zone3 m1.large (ip)
10.252.23.0 zone3 m1.large (ip)
10.252.25.0 zone3 m1.large (scan)
10.252.26.0 zone3 m1.large (ip)
10.252.27.0 zone3 m1.large (ip)
10.252.29.0 zone3 m1.large (scan)
10.252.30.0 zone3 m1.large (scan)
10.252.31.0 zone3 m1.large (ip)
10.252.33.0 zone3 m1.large (scan)
10.252.34.0 zone3 m1.large (ip)
10.252.35.0 zone3 m1.large (ip)
10.252.37.0 zone3 m1.small (ip)
10.252.38.0 zone3 m1.small (ip)
10.252.39.0 zone3 m1.small (ip)
...

Data from 977 instances with unique internal IPs

+

simple heuristics based on EC2 network configuration

=

Ability to label /24's with zone & instance type(s)

To locate a target in the cloud:

1) DNS lookup maps External IP to Internal IP

2) Check /24 to see what zone & instance type

Our map provides sufficiently precise estimate to use for mounting attacks.

Mapping might have other applications, as well (inferring types of instances used by a company)

Achieving co-residence

"Brute-forcing" co-residence

Attacker launches many VMs over a relatively long period of time in target's zone and of target type

Experiment:

1,686 public HTTP servers as stand-in "targets" running m1.small and in Zone 3 (via our map)

1,785 "attacker" instances launched over 18 days

Each checked co-residence against all targets

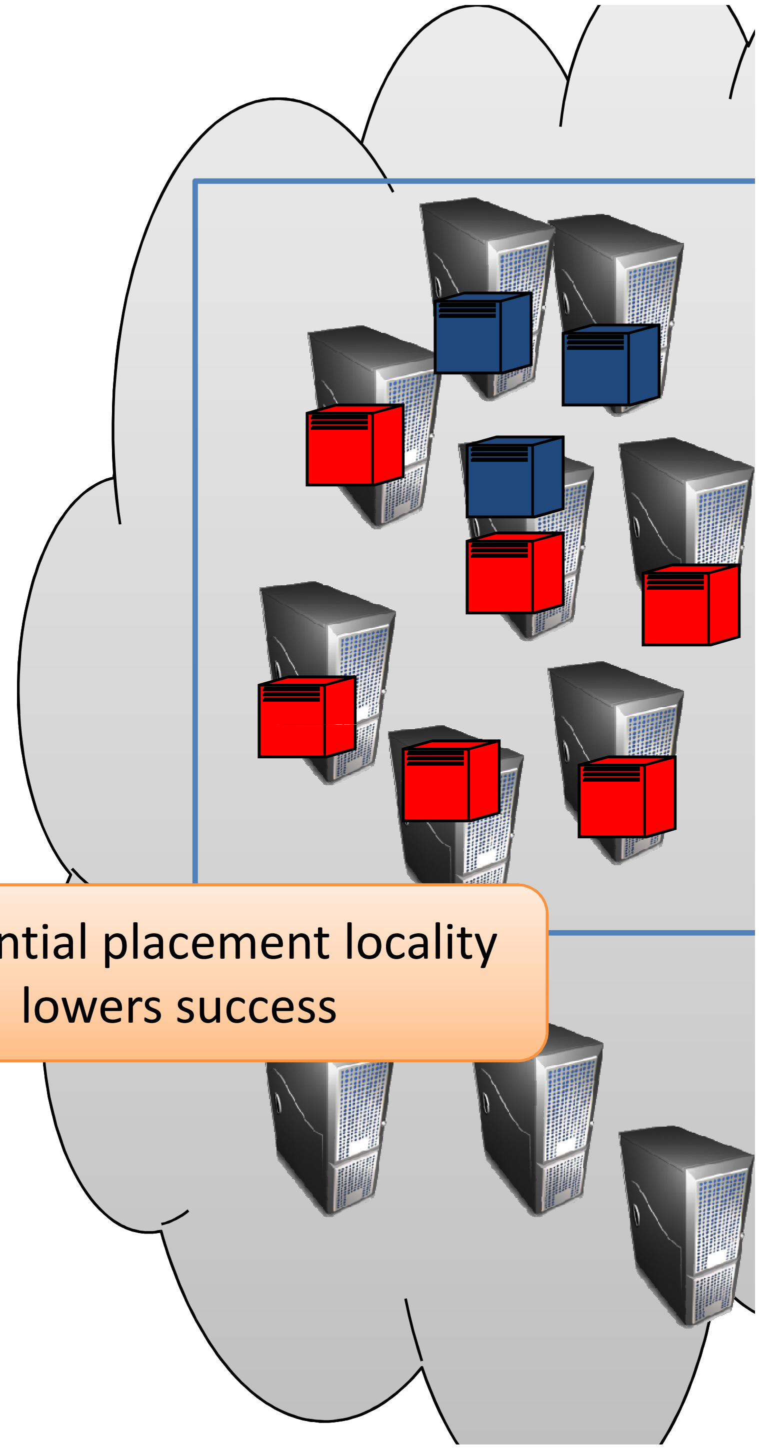
Results:

78 unique Dom0 IPs

141 / 1,686 (8.4%) had attacker co-resident

Sequential placement locality lowers success

Lower bound on true success rate



Achieving co-residence

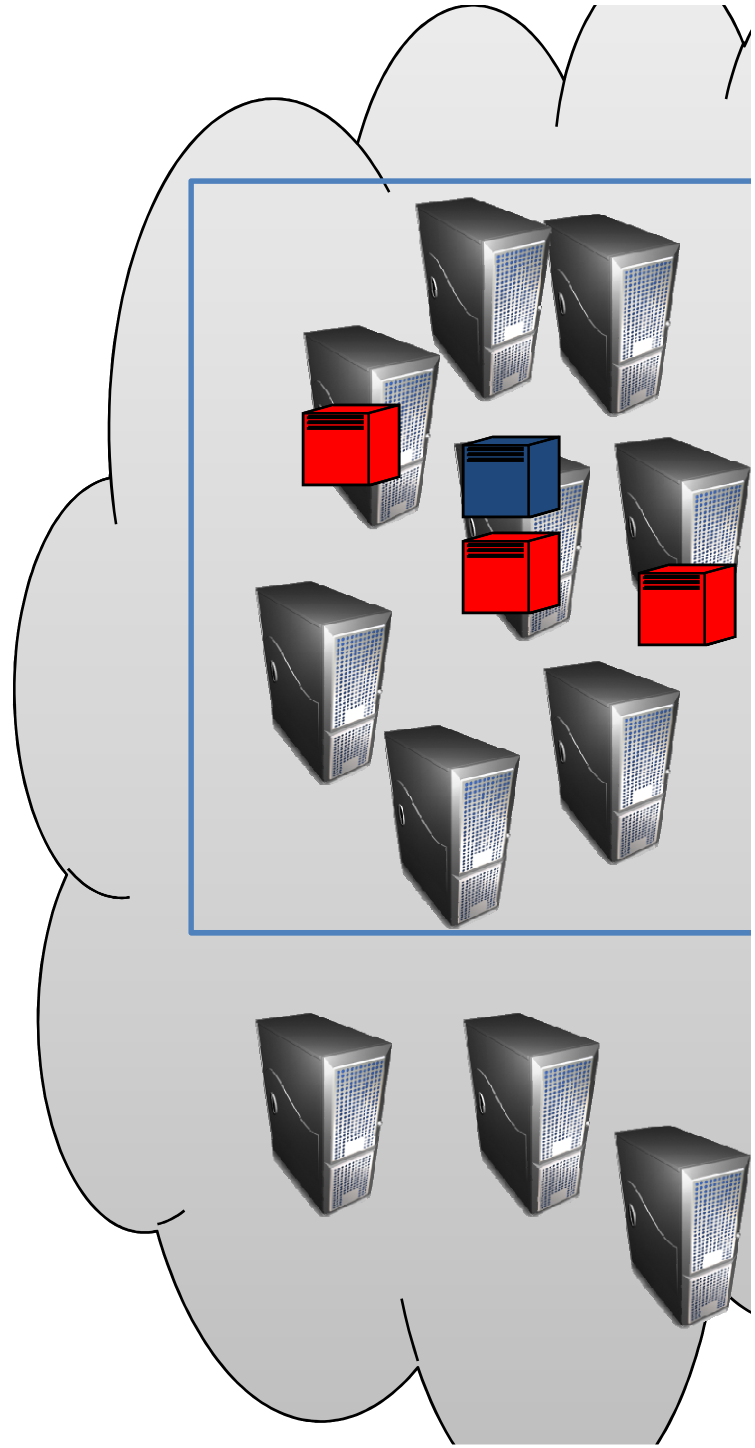
Can an attacker do better?

Launch many instances in parallel near time of target launch

Exploits parallel placement locality

Dynamic nature of cloud helps attacker:

- Auto-scaling services (Amazon, RightScale, …)
- Cause target VM to crash, relaunch
- Wait for maintenance cycles
- …



Achieving co-residence

Can an attacker do better?

Launch many instances in parallel near time of target launch

Exploits parallel placement locality

Experiment:

Repeat for 10 trials:

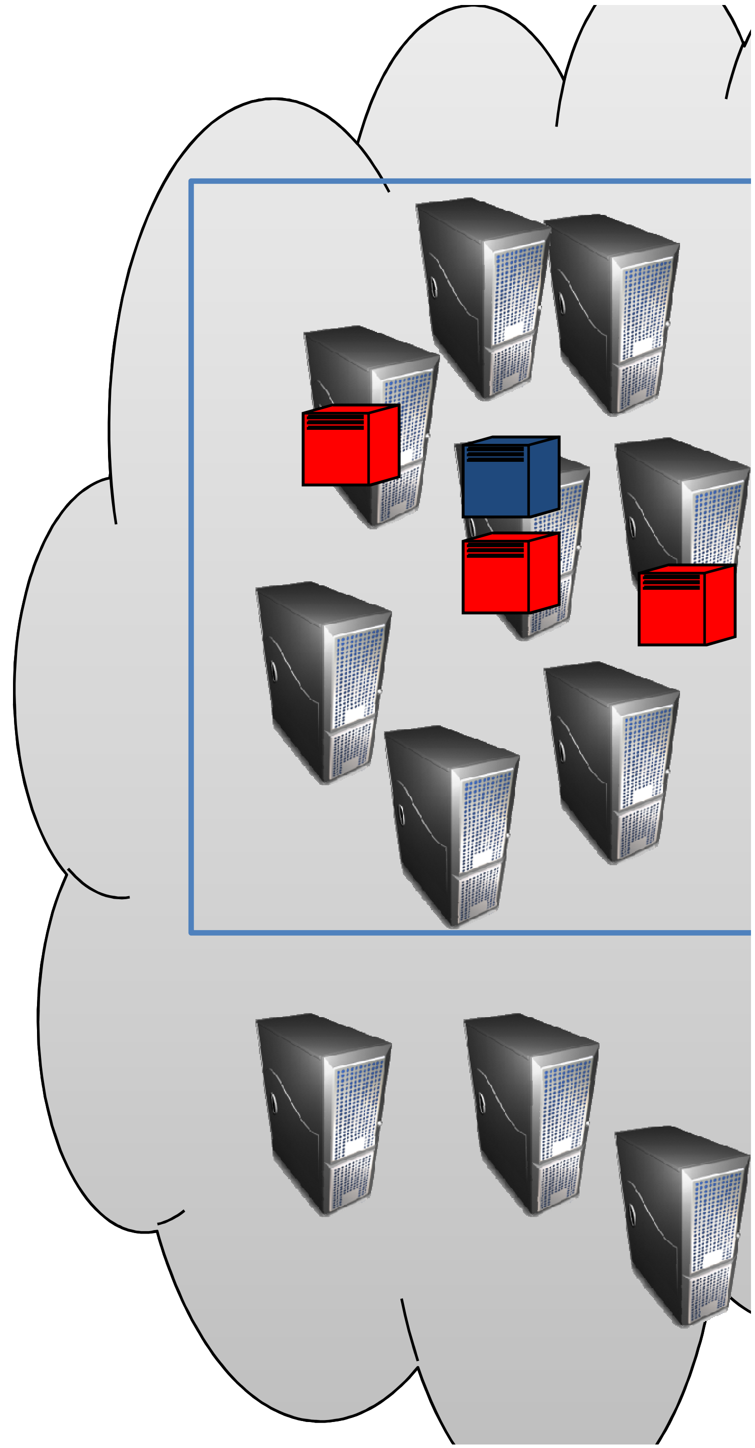
1) Launch 1 target VM (Account A)

2) 5 minutes later, launch 20 "attack" VMs (alternate using Account B or C)

3) Determine if any co-resident with target

4 / 10 trials succeeded

In paper:
parallel placement locality good for >56 hours
success against commercial accounts

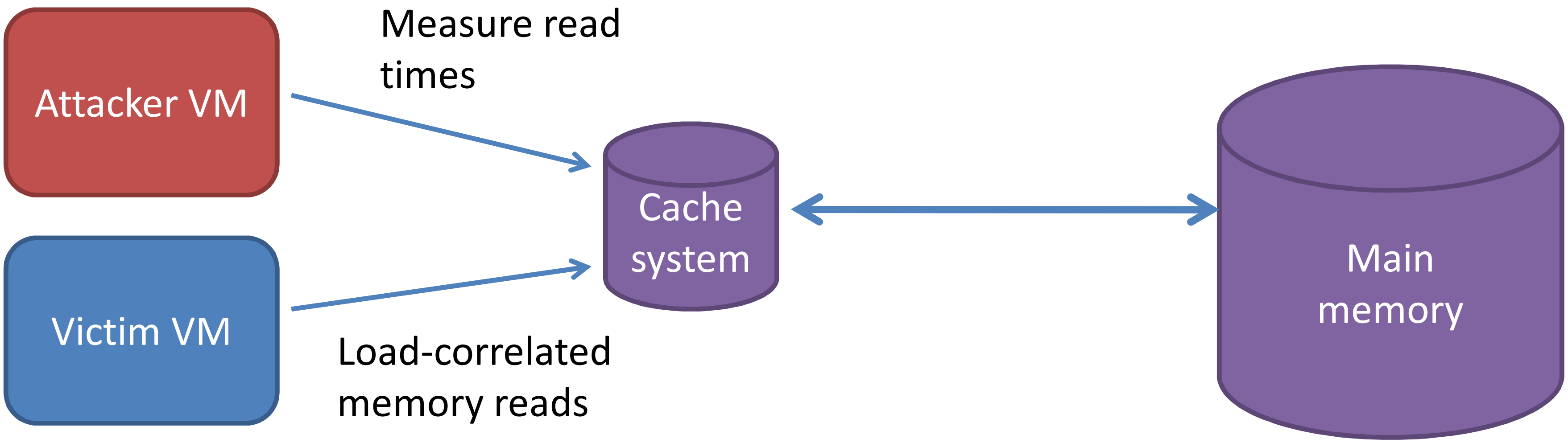


Attacker has uncomfortably good chance at achieving co-residence with your VM

What can the attacker then do?

Side-channel information leakage

Cache contention yields cross-VM load measurement in EC2



Attacker measures time to retrieve memory data

Read times increase with Victim's load

Extends [OST05] Prime+Probe technique

Measurements via Prime+Trigger+Probe :

1) Read an array to ensure cache used by attacker VM (Prime)

2) Busy loop until CPU's cycle counter jumps by large value (Trigger)

3) Measure time to read array (Probe)

Load measurement uses
coarse-grained side channel

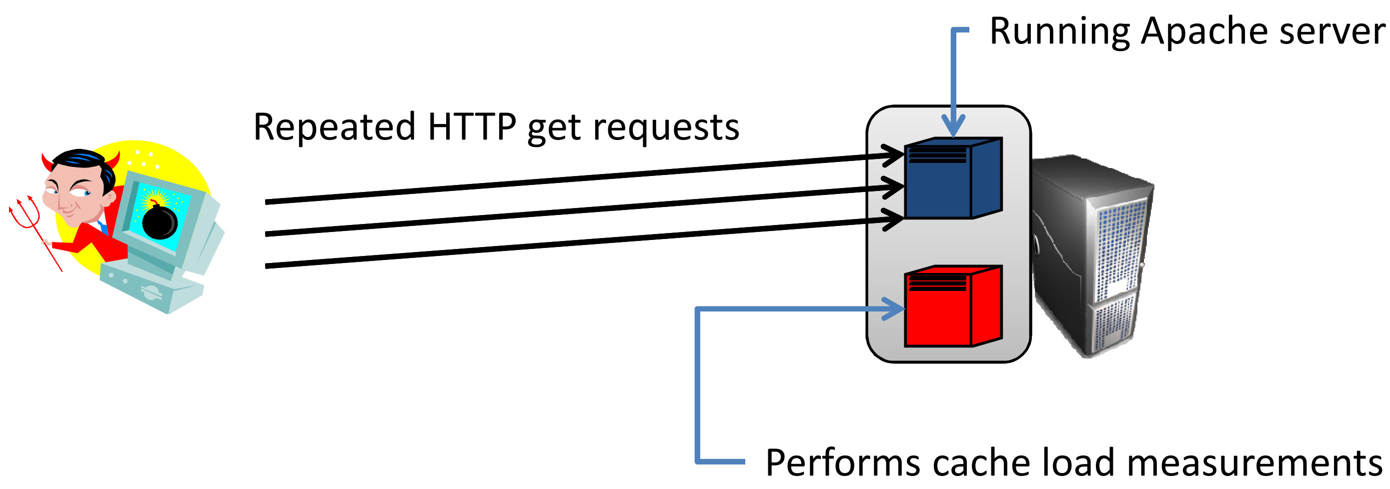
Simpler to mount

More robust to noise

Extract less information

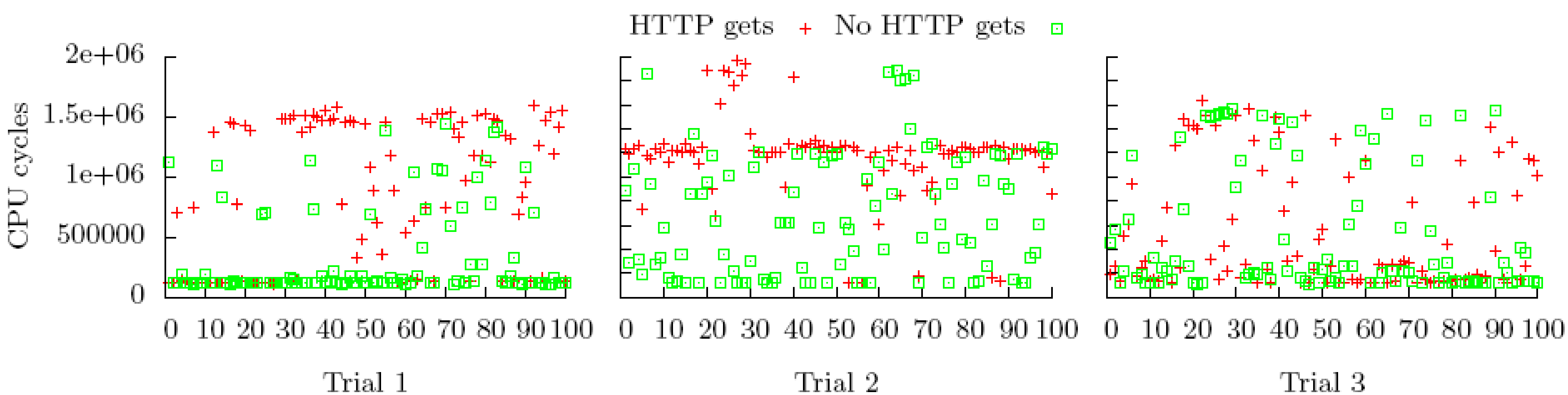
coarse side channels could be damaging
in hands of clever attackers

Cache-based load measurement to determine co-residence



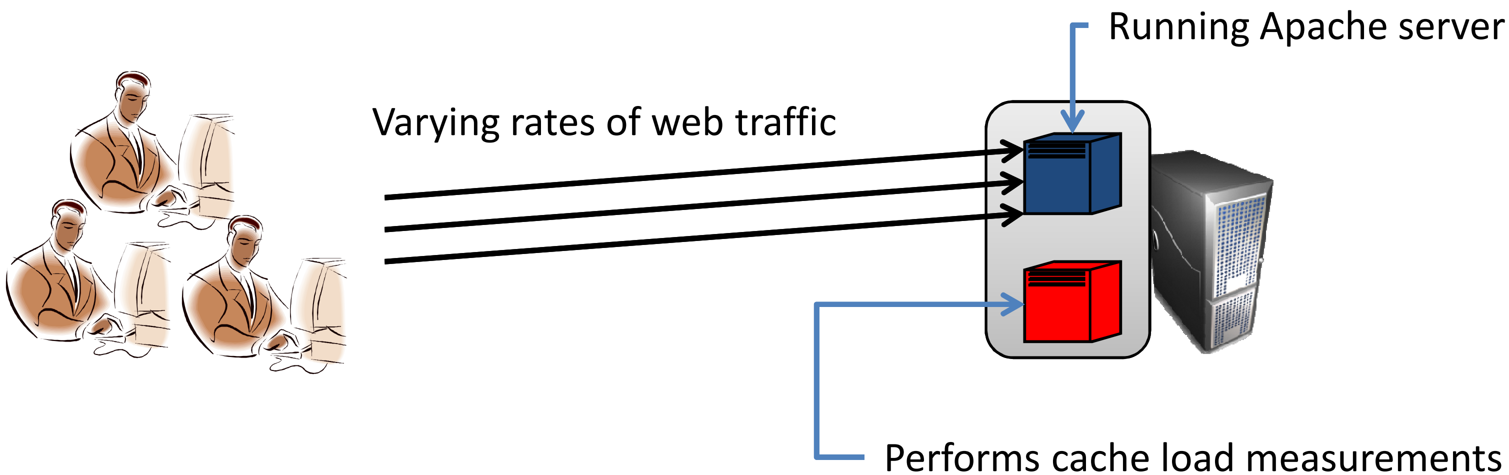
3 pairs of instances, 2 pairs co-resident and 1 not

100 cache load measurements during **HTTP gets** (1024 byte page) and with **no HTTP gets**

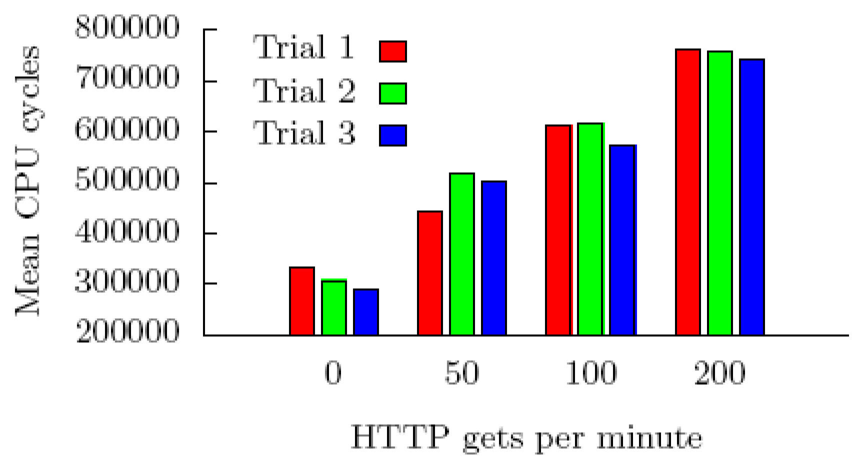


Instances co-resident | Instances co-resident | Instances NOT co-resident

Cache-based load measurement of traffic rates



3 trials with 1 pair of co-resident instances:
1000 cache load measurements during
0, 50, 100, or 200 **HTTP gets** (3 Mbyte page) per minute for ~1.5 mins

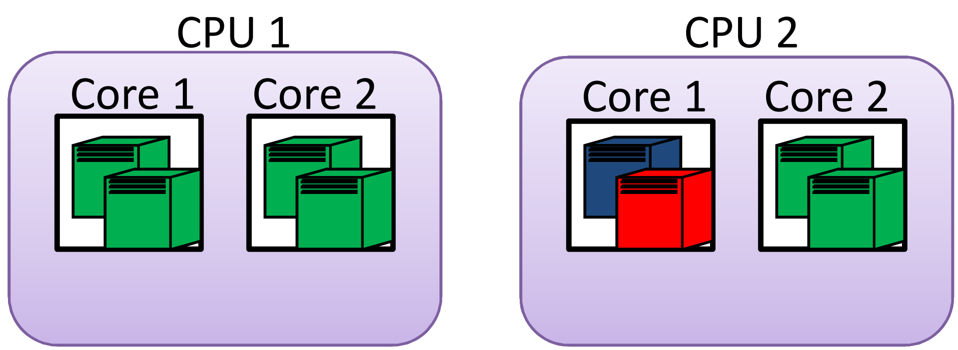


More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances

AMD Opterons

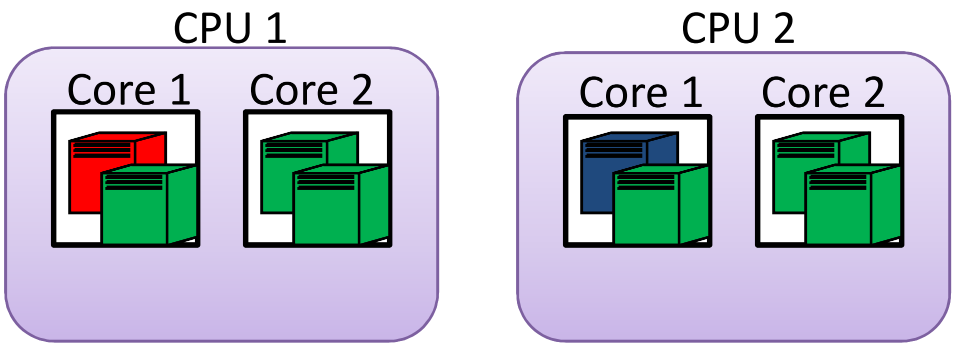


More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances

AMD Opterons

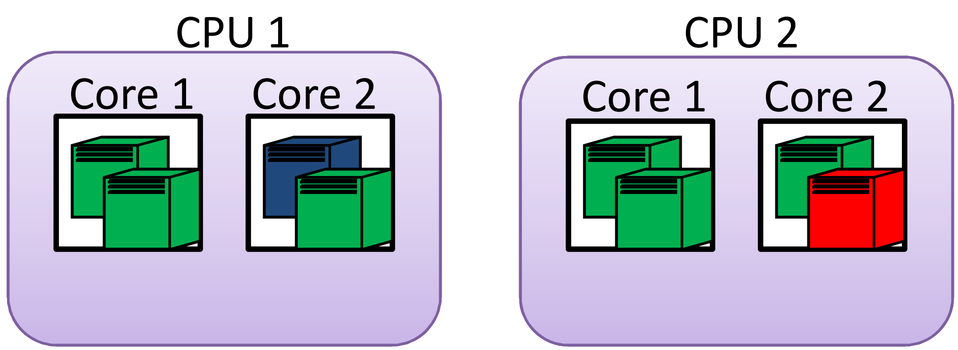


More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances

AMD Opterons



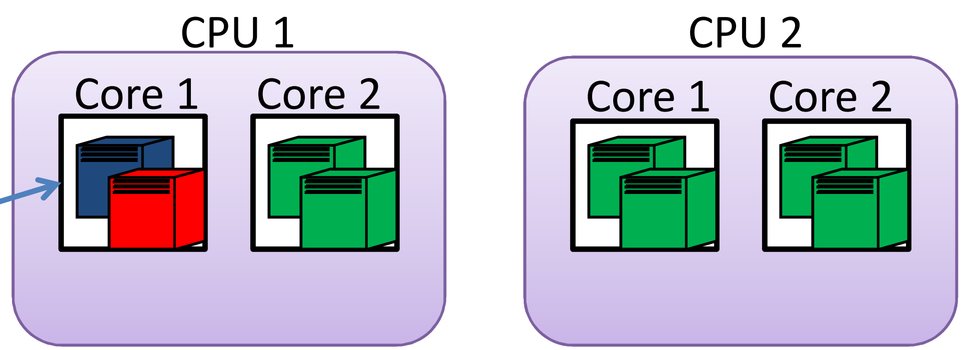
More on cache-based physical channels

Prime+Trigger+Probe combined with differential encoding technique gives high bandwidth cross-VM covert channel on EC2

Keystroke timing in experimental testbed similar to EC2 m1.small instances

AMD Opterons

VMs pinned to core



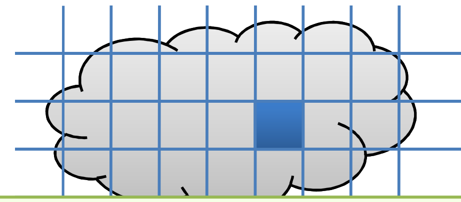
We show that cache-load measurements enable cross-VM keystroke detection

Keystroke timing of this form might be sufficient for the password recovery attacks of [Song, Wagner, Tian 01]

What can cloud providers do?

Possible counter-measures:

1) Cloud cartography

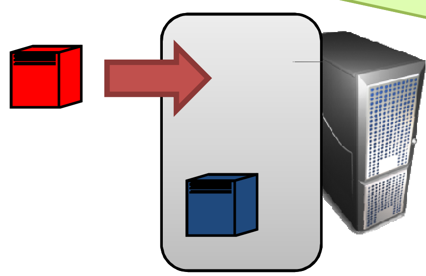


- Random Internal IP assignment
- Isolate each user's view of internal address space

Customers can pay the (slight) extra operational costs to avoid multitenancy

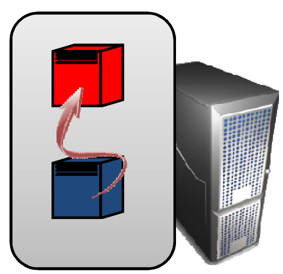
- Hide Dom0 from traceroutes
- Random Internal IP assignment

3) Achieving co-residence

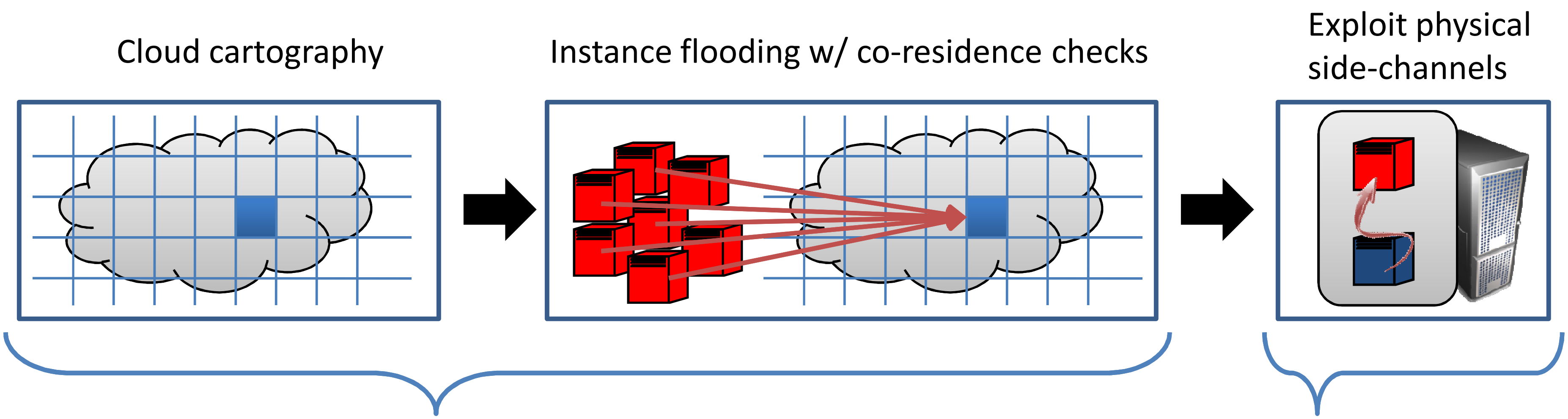


- Allow users to opt out of multitenancy

4) Side-channel information leakage



- Hardware or software countermeasures to stop leakage [Ber05,OST05,Page02,Page03, Page05,Per05]



Placement vulnerability: attackers can knowingly achieve co-residence with target

Load measurement via side channels

Security threat seems inherent to any third-party cloud with multitenancy

More demands on virtual isolation due to multitenancy

Coarse-grained side channels already of use to some attackers