Lecture 12: Cloud Security

Professor Adam Bates
Fall 2019

Goals for Today

- Learning Objectives:
  - Explore the security of different kinds of cloud infrastructures
- Announcements, etc:
  - **Midterm October 9th, 7pm, 1404 Siebel**
  - Grade distributions for MP1 checkpoints will be released after regrade requests are processed.
  - Schedule has been updated

Reminder: Please put away devices at the start of class

Midterm Details

- October 9th, 7-9pm
  - Here, 1404 Siebel
- Multiple choice + short answer
- **Closed book.**
- *No electronic devices permitted (or necessary)!*
- **Content**: All lectures prior to Oct 7, MP1 and MP2.
- We will have a review session, Q&A on October 7th
- Sample exams available (midterm only)! https://courses.engr.illinois.edu/cs461/fa2019/schedule.html

Early Amazon: Jan-Nov, Every Year

*"What do I do with all of these servers when they're not needed for online shopping??"**

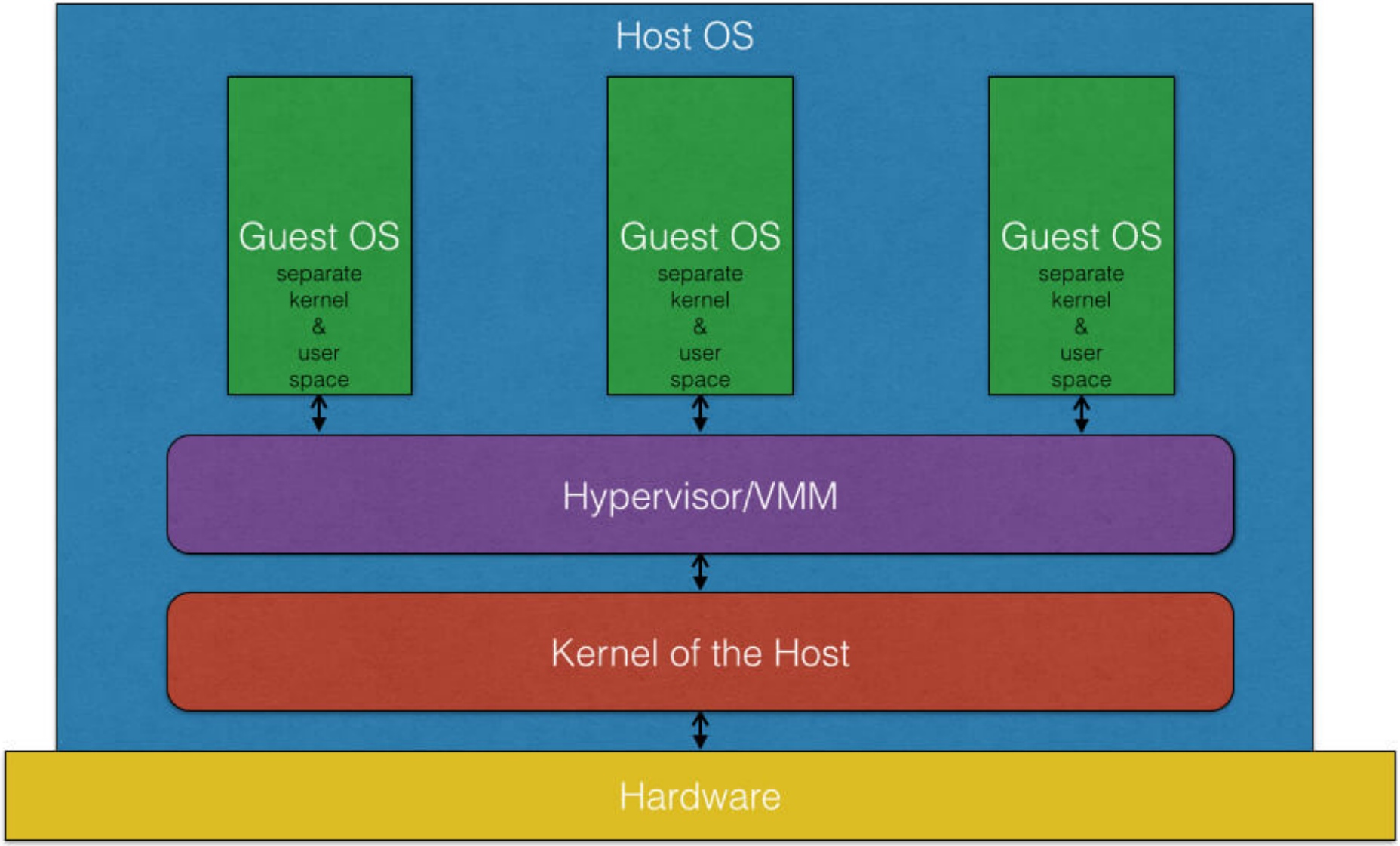
- Jeff Bezos, 2006

* completely fabricated

Cloud Computing (Gen 1)

- Dominated by Infrastructure-as-a-Service clouds
- Big winner was Amazon EC2
- Hypervisors that virtualized the hardware-software interface
- Customers were responsible for provisioning the software stack from the kernel up

Hypervisors (Preview)



Hypervisors (Preview)

- Strong isolation between different customer's virtual machines
- VMM is 'small' compared to the kernel… less LoC means less bugs means (~)more security.

Xen: ~300k LoC. AWESOME!
VMware: ~6 million LoC… mini OS
KVM: 10,000 LoC for virtualization…
… but 25+ million lines total in kernel

Economic Model

- Third-party cloud computing represents the promise of outsourced computation.
- It allows customers to purchase just the capacity they require, just when they require it.
- Cloud providers are able to maximize utilization of their capital investments by multiplexing many customer VMs across a shared physical infrastructure.
- It is a given that we need to be able to trust cloud providers to respect our private data…
  - *Note:* Trusted Execution Environments, conceivably, could be changing that.

… can we trust other users?

- We already know that 3rd Party cloud providers make their $$$ by multiplexing the machines in their monstrously large datacenters.
- Cloud computing creates threats of multi-tenancy, multiplexing the virtual machines of disjoint customers upon the same physical hardware.
- Could a customer be assigned to the same physical server as their adversary?
- Could the adversary exploit co-residency to extract confidential information?

Co-Residency Threat Model

- We trust the provider, its infrastructure and its employees.
- Adversaries are non-provider-affiliated malicious parties.
- Victims are running confidentiality-requiring services in the cloud.
- Everyone is a customer; both groups can all run and control many instances.
- We are not concerned with traditional threats and exploits here, even though they are alive and well in the cloud environment.
- 3rd Party Cloud Providers give attackers ***novel abilities*** , implicitly expanding the ***attack surface*** of the victim.
- Two kinds of attackers
  1. Casts a wide net in an attempt to attack somebody
  2. Focuses on attacking a particular victim service

Hey! You! Get Off of My Cloud!

1. Use Amazon EC2 as a case study.
   - U.S. Region
   - Linux Kernel
2. Achieve **PLACEMENT** of their malicious VM on the same physical machine as that of a target customer.
   - Determine where in the cloud an instance is likely to be located.
   - Determine if two instances are co-residents.
   - Intentionally launch an instance to achieve co-residence with another user.
3. Proceed to **EXTRACT** information and/or perpetrate all kinds of assorted nastiness.

[Ristenpart et al., CCS'09]

Hey! You! Get Off of My Cloud!

"Cloud Cartography"

- ***Hypothesis: different availability zones (and possibly instance types) are likely to correspond to different internal IP address ranges.***
- Since we already know that it's possible to infer the internal IP address of an instance associated with a public IP through the EC2's DNS service…
- If this hypothesis holds, an adversary can use a map of EC2 to determine the instance type and availability zone of their target, dramatically reducing the number of instances needed to achieve co-residence.

[Ristenpart et al., CCS'09]

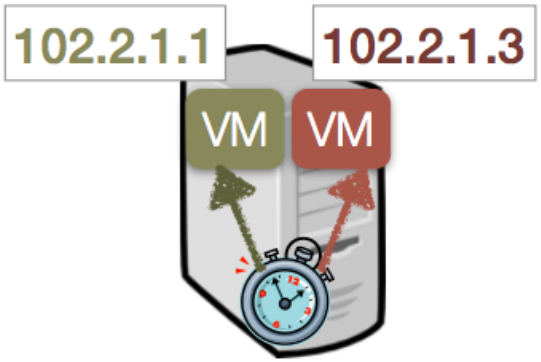
Co-Residency Detection

… but how does the adversary know they've been placed next to the victim?

Co-Residency Detection

1. Read shared state on two VMs

e.g., private IP addresses, shared TSC counters.



- Worked well in early days; was as simple as checking dom0 IP address.
- Less common now; many shared state channels have been patched.

2. Correlate performance of shared resources

e.g., network round-trip times, cache-based covert-channels.

- Early effective techniques used L2 cache, i.e., "prime and probe."
- Sharing is intrinsic to cloud computing; difficult to fix

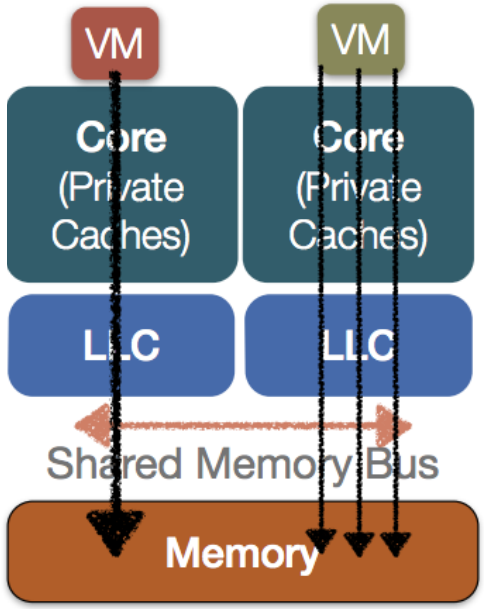
[Ristenpart et al., CCS'09]

Cooperative Detection

- In one class of co-residency detection schemes, VMs can collude to infer their placement.
  - (Works well for measurement studies but not attacks)
- Wu et al.'s memory locking covert channel does the trick!

Sender:

```
// allocate memory multiples of 64 bits
char_ptr = allocate_memory((N+1)*8)
//move half word up
unaligned_addr = char_ptr + 2
loop forever:
  loop i from (1..N):
    atomic_op(unaligned_addr + i, some_value)
  end loop
end loop
```



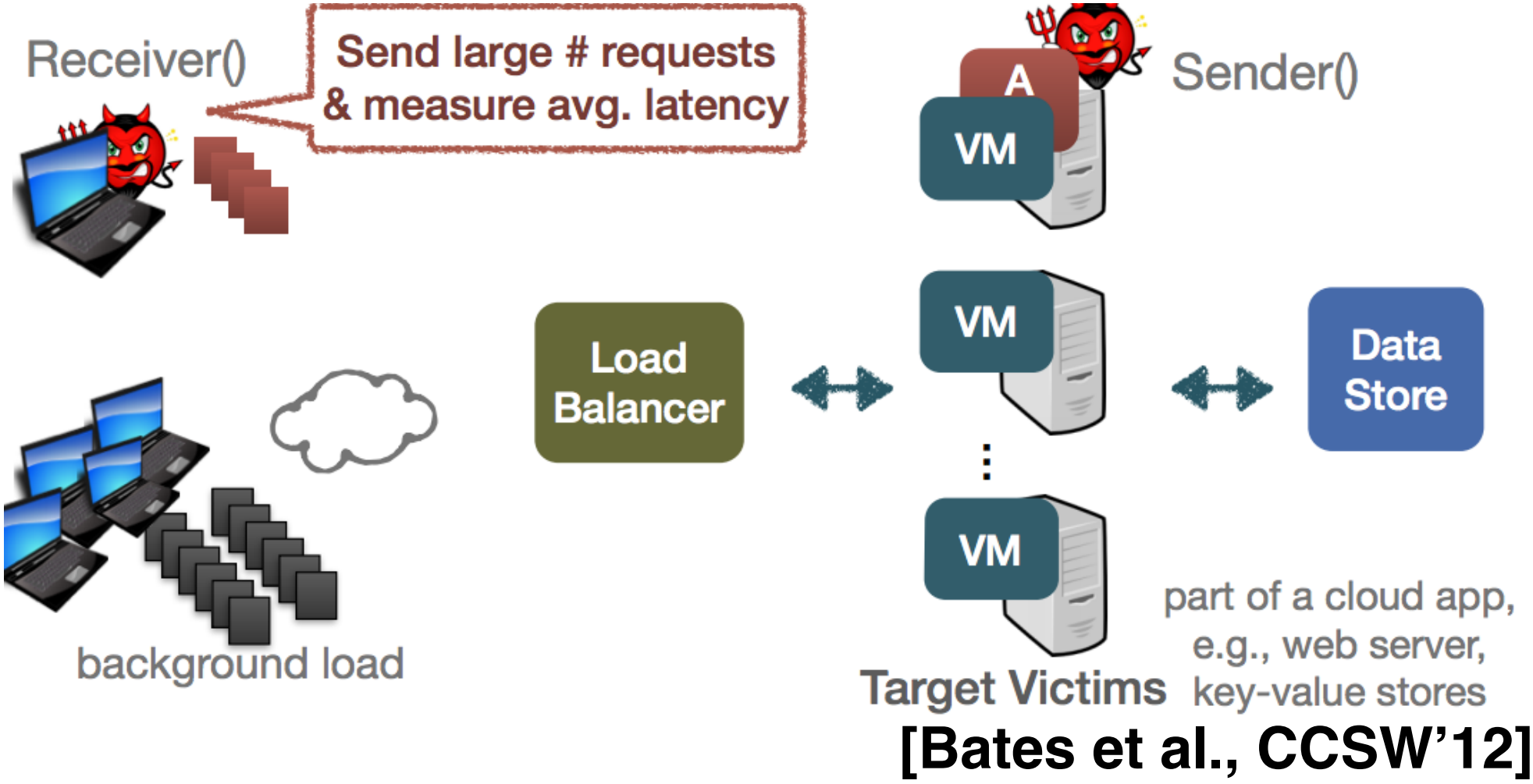
Receiver:

```
Observe() {
  s = start_time
  repeat N
    mem_access()
  done
  e = end_time
  bw = N/(e-s)
}
```

[Wu et al., Security'12]

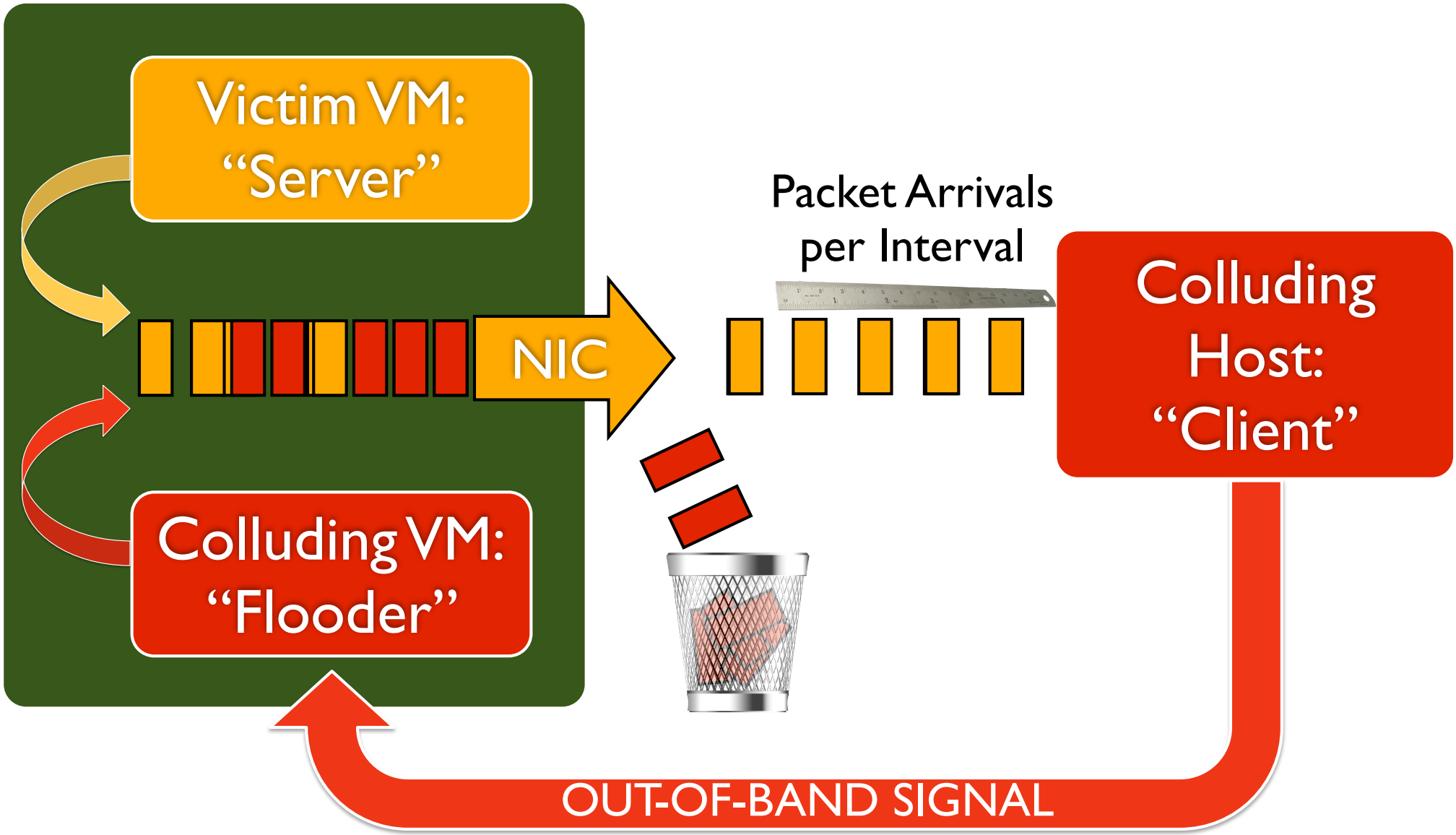
Cooperative Detection

- ***What about on an un-cooperative (victim) VM?***
- One possibility — embed a beacon into network activity!



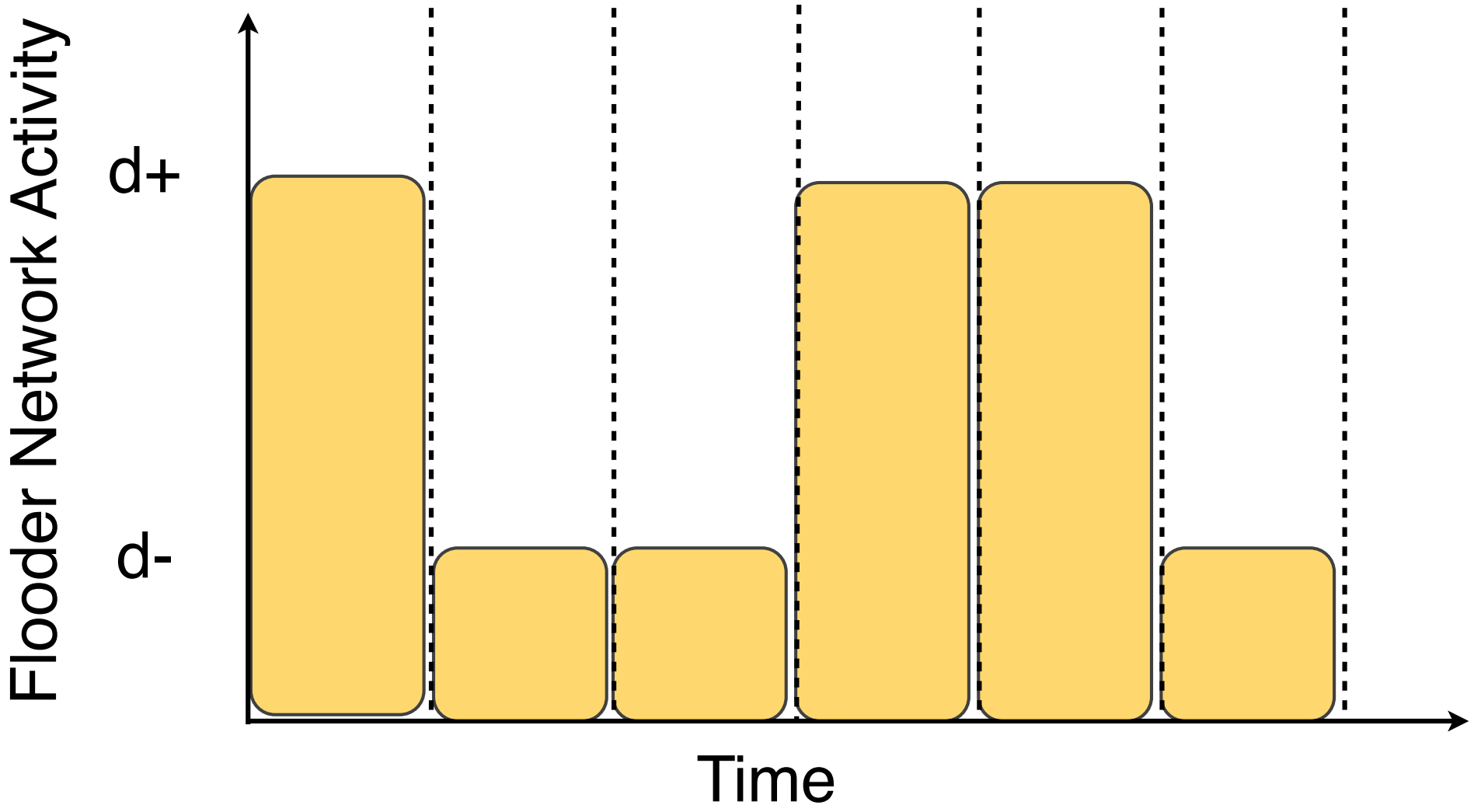
[Bates et al., CCSW'12]

Co-Resident Watermarking



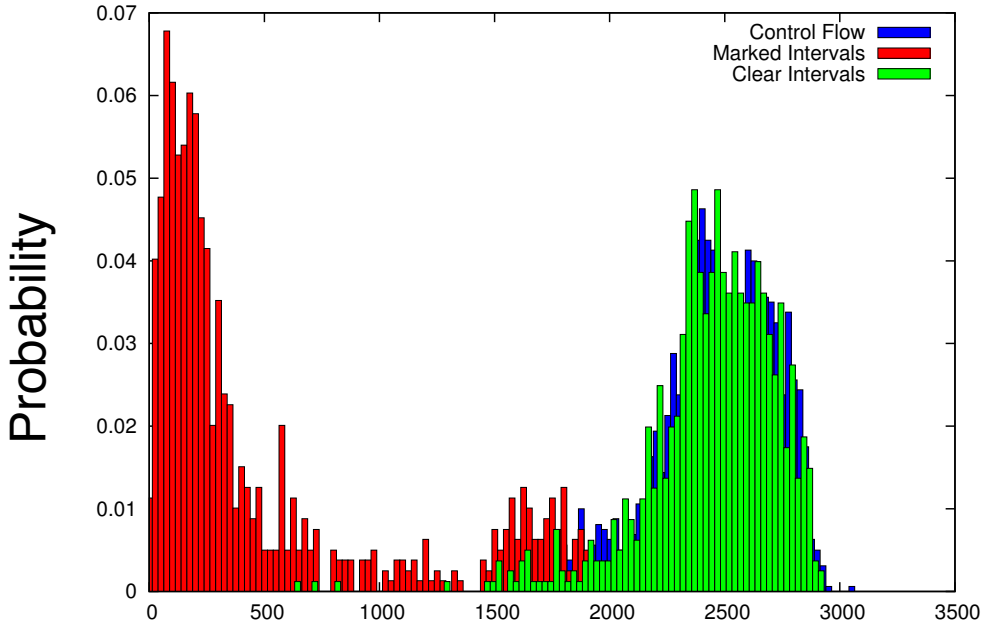
[Bates et al., CCSW'12]

Co-Resident Watermarking

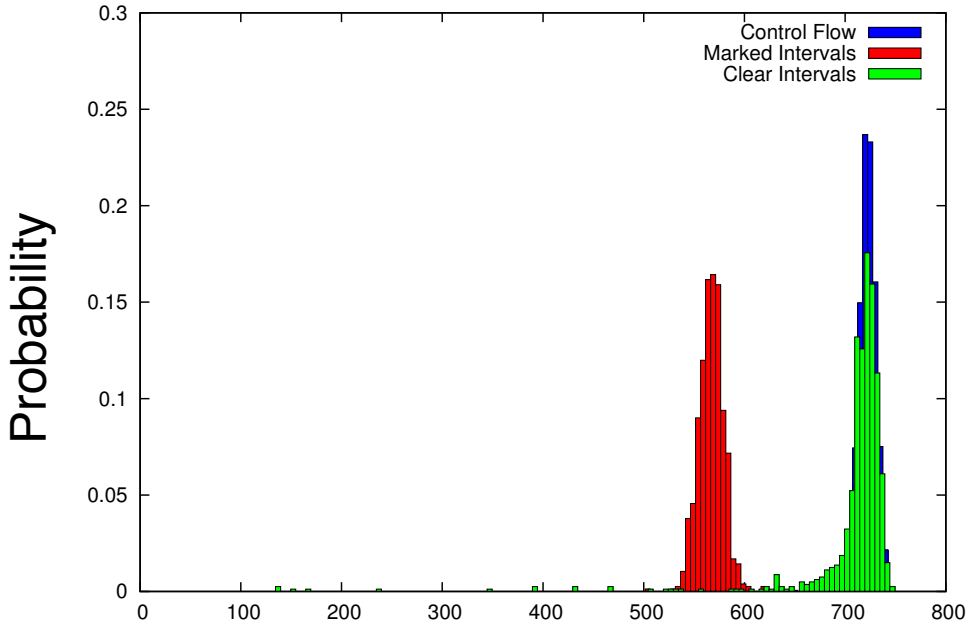


[Bates et al., CCSW'12]

Co-Resident Watermarking



ACISS (KVM)

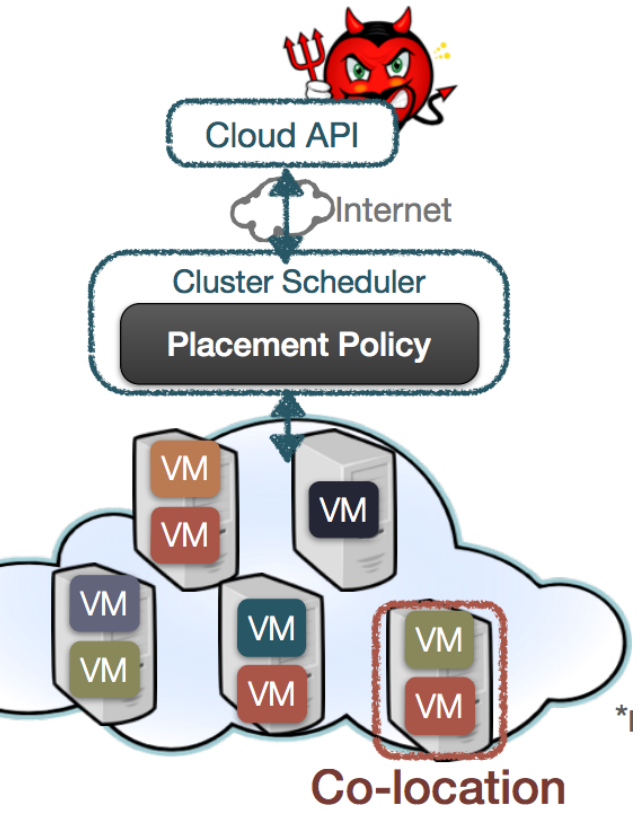


Futuregrid (Xen)

- *Co-resident watermarking* is a viable attack in production cloud environments.

[Bates et al., CCSW'12]

How hard *should* it be to co-locate?



If a truly random placement policy was used…

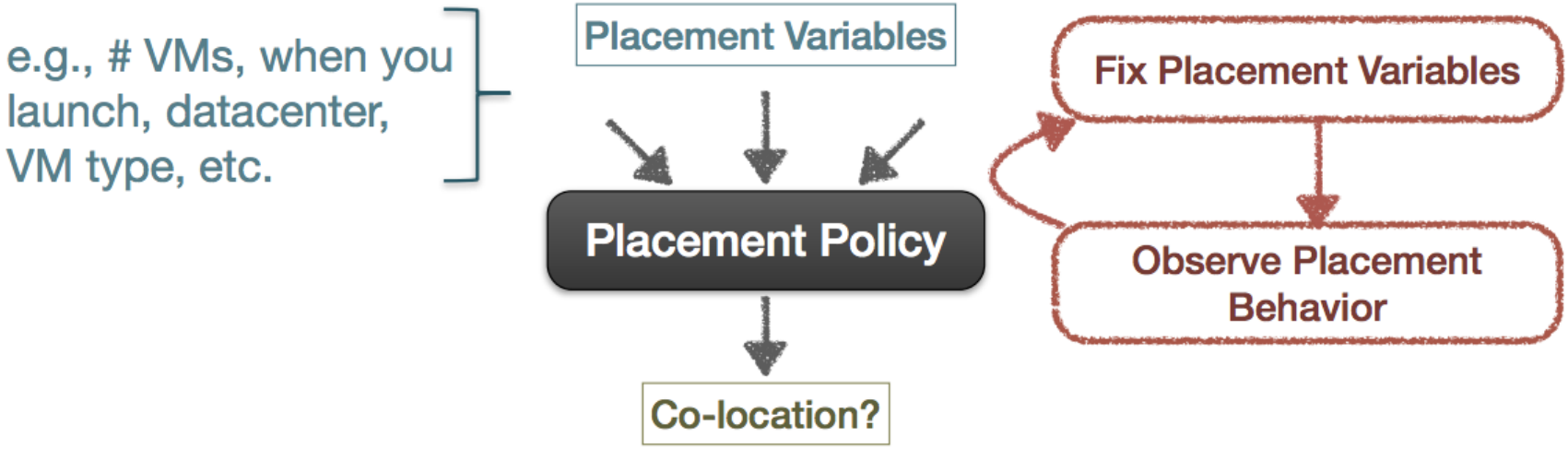
- N = 50,000 machines
- v victim VMs and a attacker VMs
- Probability of Collision:

$$P_c = 1 - (1 - \frac{v}{N})^a$$

v	a = ln(1– P_c)/ln(1–v/N); P_c= 0.5
10	3466
20	1733
30	1155

[Varadarajan et al., Security'15]

Placement Study



- 6 placement variables: # victim & attacker VMs, delay b/w launches, time of day, day of week, datacenter, cloud provider Small instance type
- 9 samples per strategy with 3 runs per time of day and 2 days of week (weekday/weekend).

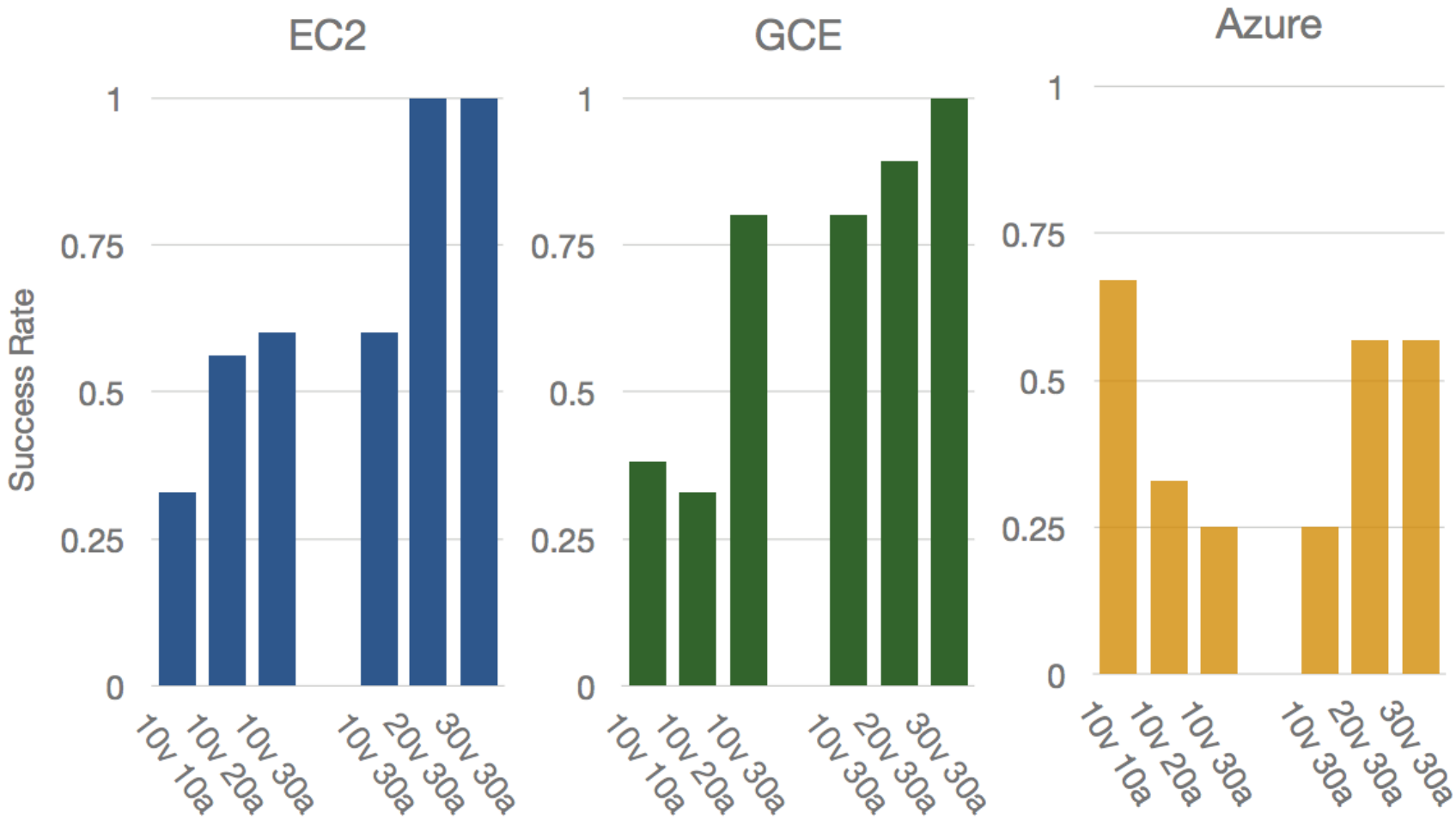
Google Compute Engine

amazon web services EC2

Microsoft Azure

[Varadarajan et al., Security'15]

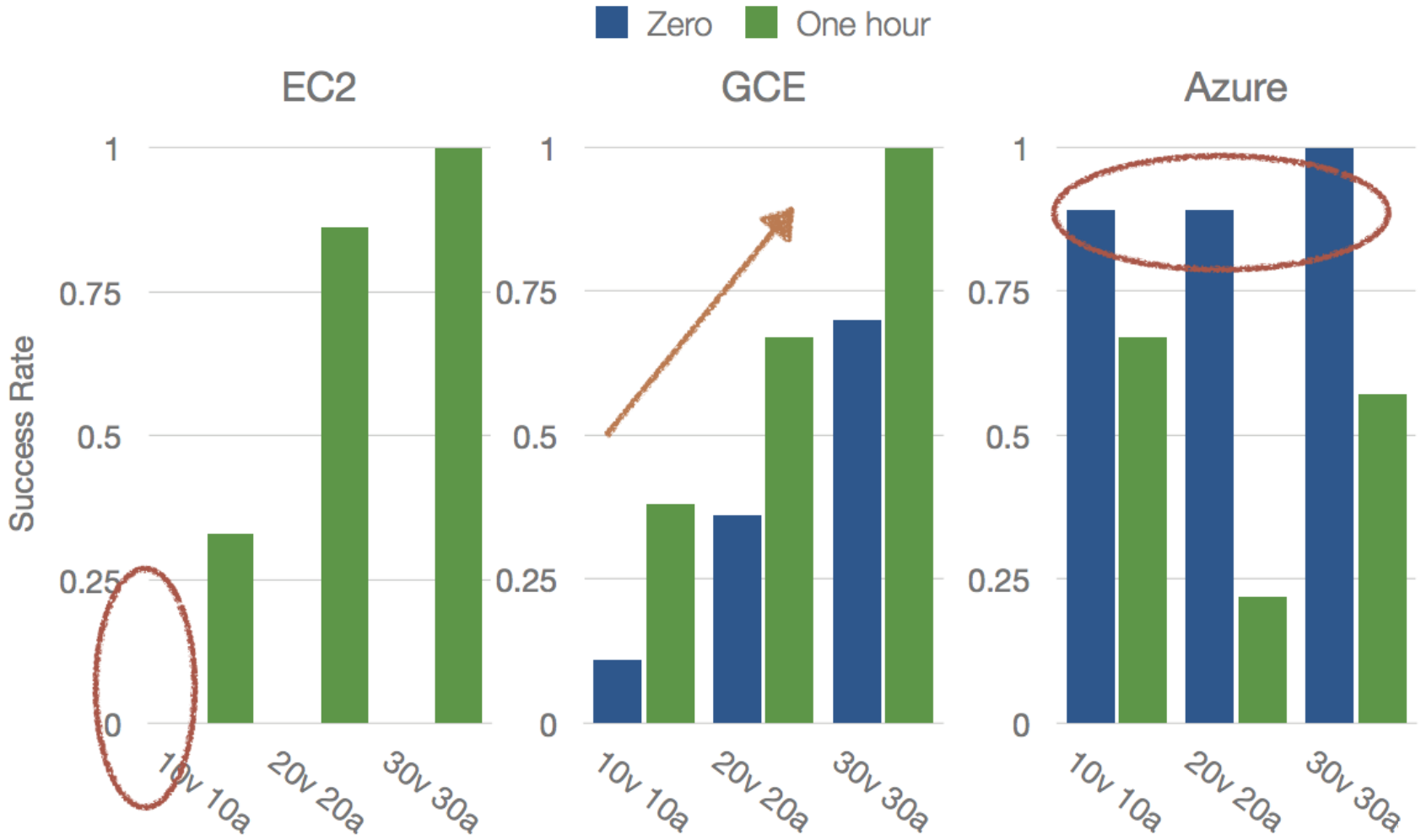
Variable # of VMs



Co-location is possible with as low as 10 VMs and always achieves co-location with 30 VMs

[Varadarajan et al., Security'15]

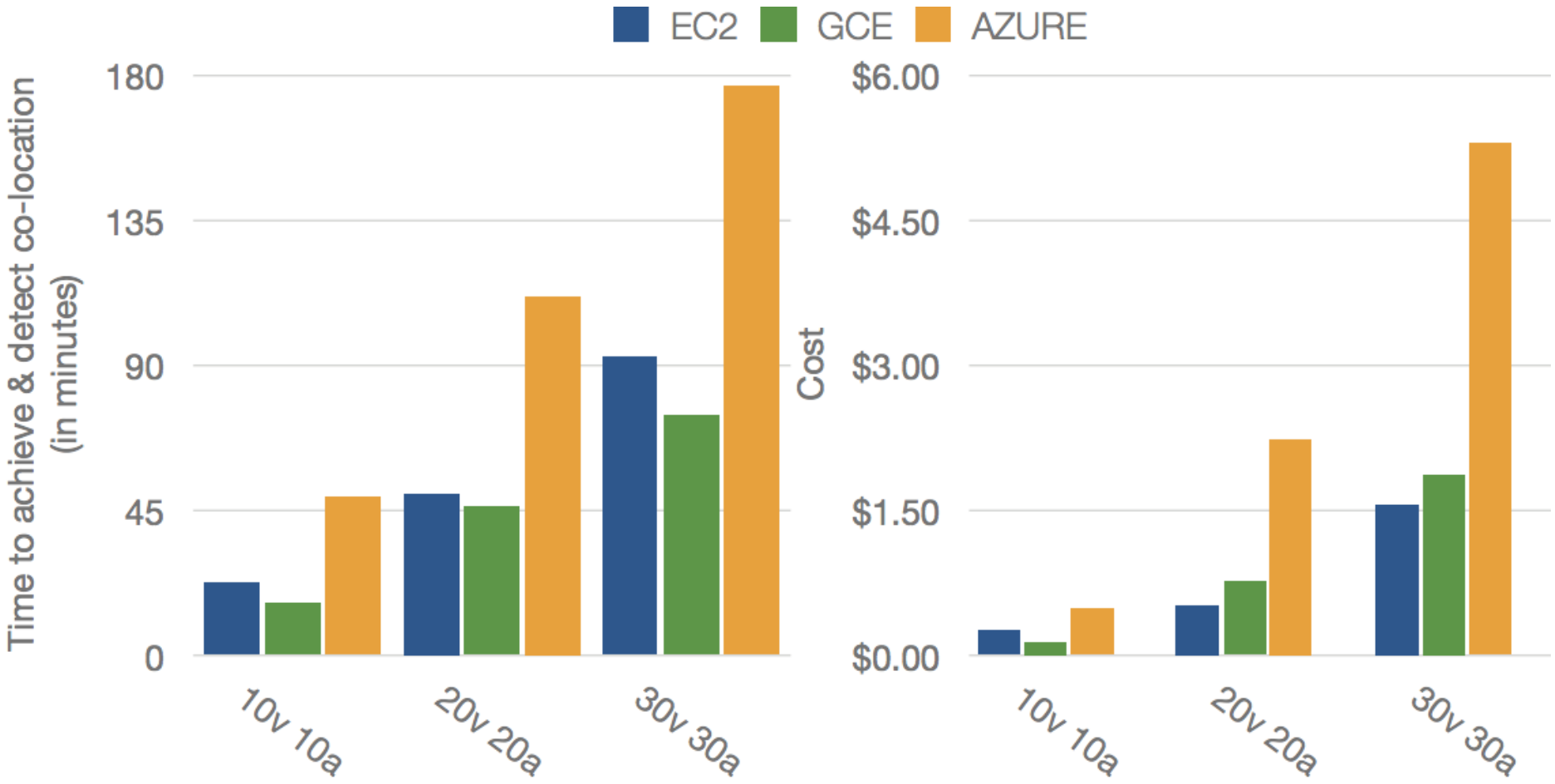
Variable Delay between launches



Different clouds have wildly different temporal placement strategies

[Varadarajan et al., Security'15]

Attack Cost



Successful co-location as affordable as 14 cents.

[Varadarajan et al., Security'15]

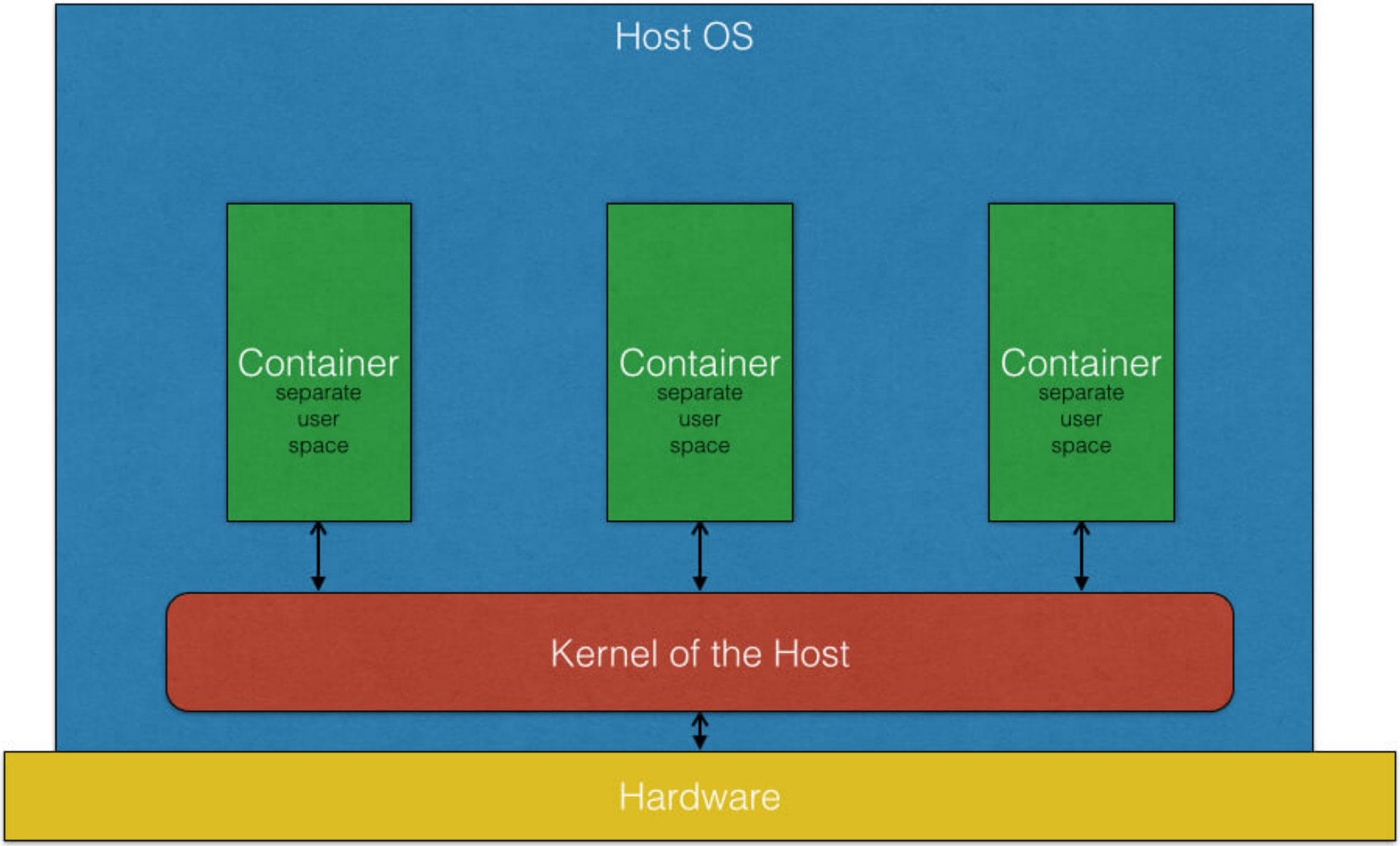
IaaS vs. *

- Strong isolation between different customer's virtual machines
- VMM is 'small' compared to the kernel… less LoC means less bugs means (~)more security.
- High degree of flexibility… but did most customers really need it?

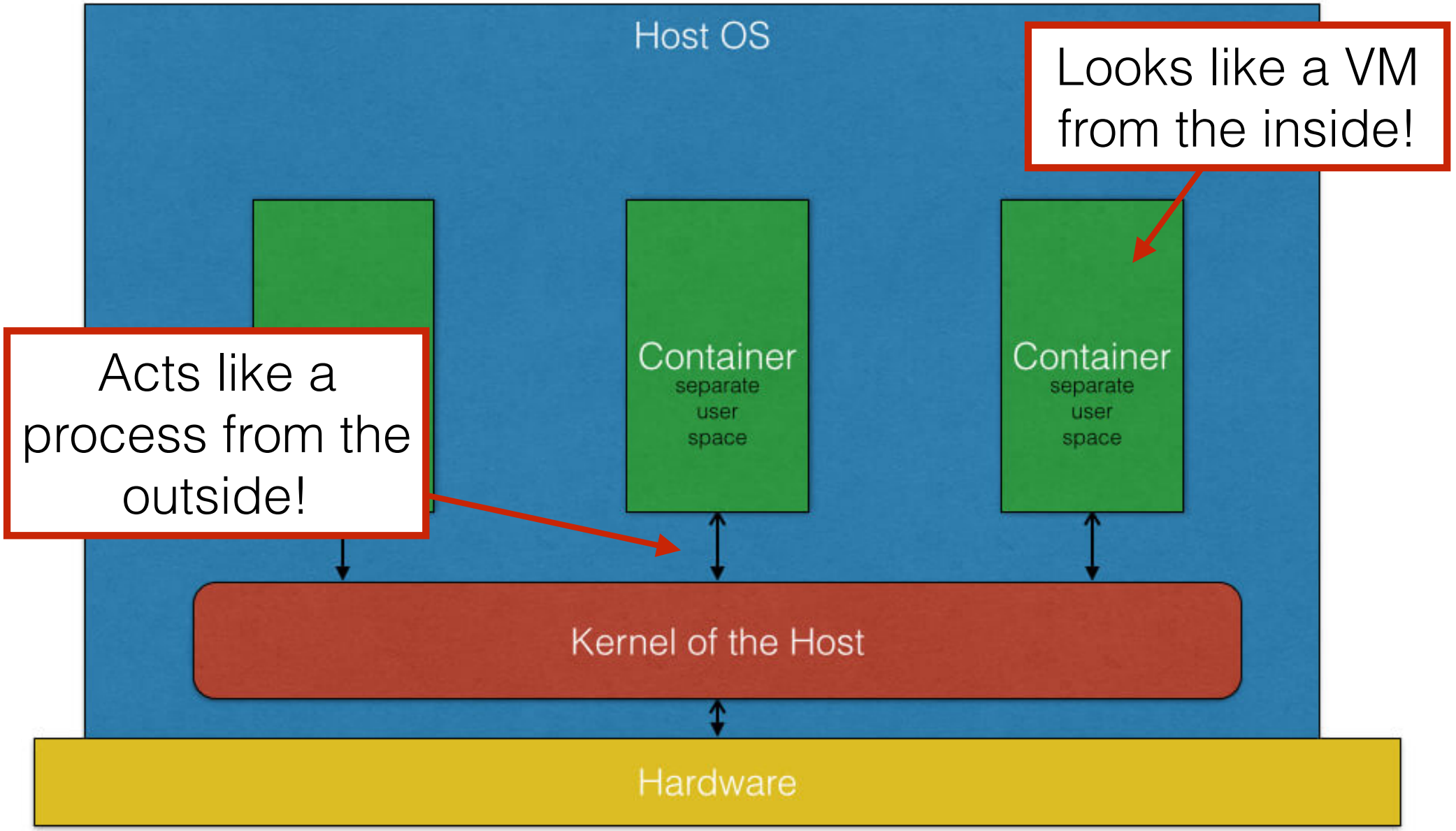
Enter Containers

- Rather than virtualize both user space and kernel space… why not just 'virtualize' user space?
- Meets the needs of most customers, who don't require significant customization of the OS.
- Sometimes called 'operating system virtualization,' which is highly misleading…
- Running natively on host, containers enjoy bare metal performance without reliance on advanced virtualization support from hardware.

Enter Containers



Enter Containers



OS Support for Containers

- Linux Containers (LXC):
  - chroot
  - Kernel Namespaces
    - PID, Network, User, IPC, uts, mount
  - cgroups for HW isolation
  - Security profiles and policies
    - Apparmor, SELinux, Seccomp

containers = chroot on steroids

- `chroot` changes the apparent root directory for a given process and all of its children
- An old idea! POSIX call dating back to 1979
- Not intended to defend against privileged attackers… they still have root access and can do all sorts of things to break out (like `chroot`'ing again)
- Hiding the true root FS isolates a lot; in *nix, file abstraction used extensively.
- Does not completely hide processes, network, etc., though!

Namespaces

- The key feature enabling containerization!
- Partition practically all OS functionalities so that different process domains see different things
- Mount (mnt): Controls mount points
- Process ID (pid): Exposes a new set of process IDs distinct from other namespaces (i.e., the hosts)
- Network (net): Dedicated network stack per container; each interface present in exactly 1 namespace at a time.
- ….

Namespaces

- The key feature enabling containerization!
- Partition practically all OS functionalities so that different process domains see different things
- Interprocess Comm. (IPC): Isolate processes from various methods of POSIX IPC.
    - e.g., no shared memory between containers!
- UTS: Allows the host to present different host/domain names to different containers.
- There's also a User ID (user) and cgroup namespace

User Namespace

- Like others, can provide a unique UID space to the container.
- More nuanced though — we can map UID 0 inside the container to UID 1000 outside; allows processes inside of container to think they're root.
- Enables containers to perform administration actions, e.g., adding more users, while remaining confined to their namespace.

cgroups

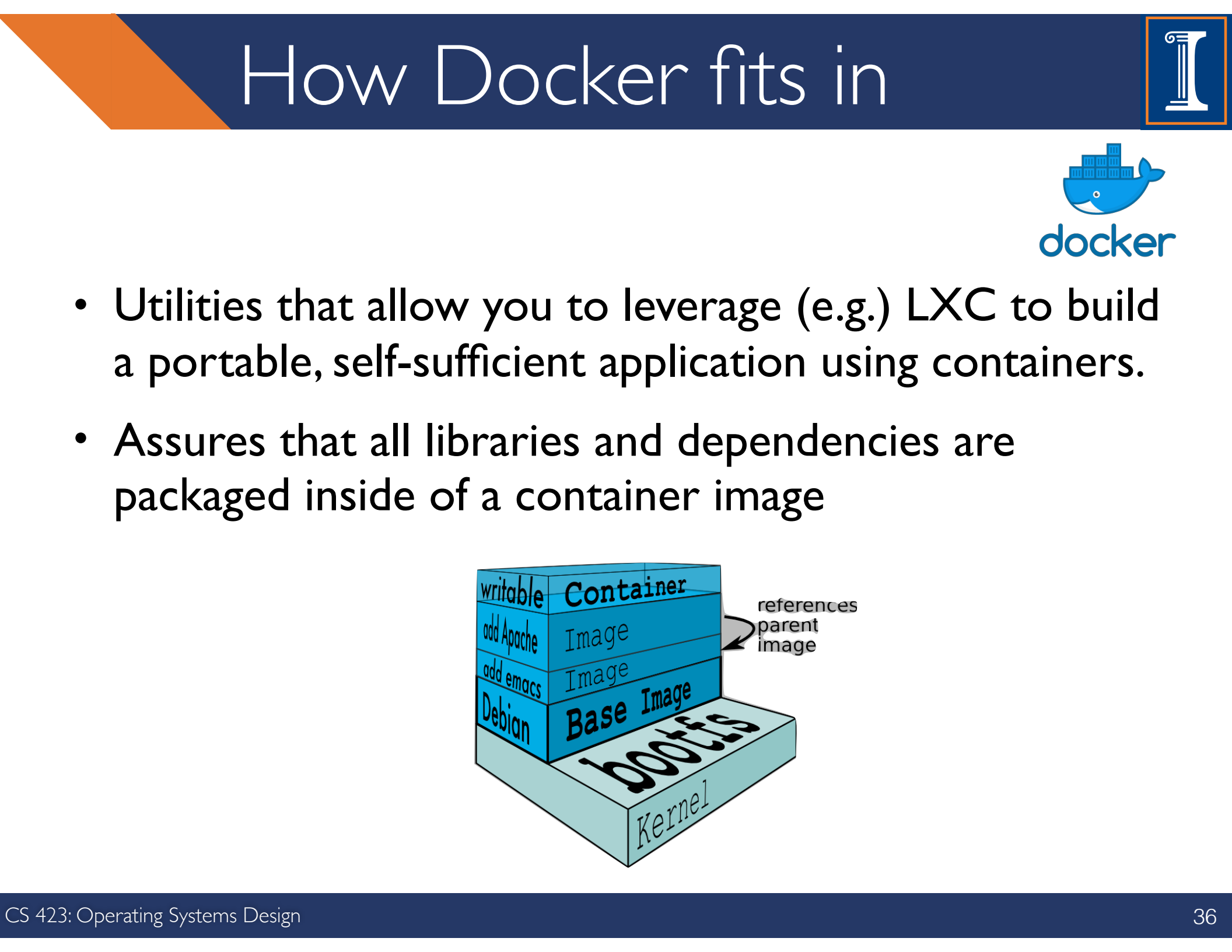
- Limit, track, and isolate utilization of hardware resources including CPU, memory, and disk.
- Important for ensuring QoS between customers! Protects against bad neighbors
- Operate at the namespace granularity, not per-process
- Features:
  - Resource limitation
  - Prioritization
  - Accounting (for billing customers!)
  - Control, e.g., freezing groups
- The cgroup namespace prevents containers from viewing or modifying their own group assignment

Container Security?

"Containers do not contain." - Dan Walsh (SELinux contributor)

- In a nutshell, it's *real hard* to prove that every feature of the operating system is namespaced.
  - /sys? /proc? /dev? LKMs? kernel keyrings?
- Root access to any of these enables pwning the host
- Solution? Just don't forget about MAC; at this point SELinux pretty good support for namespace labeling.
- SELinux and Namespaces actually synergize nicely; *much* easier to express a correct isolation policy over a coarse-grained namespace than, say, individual processes

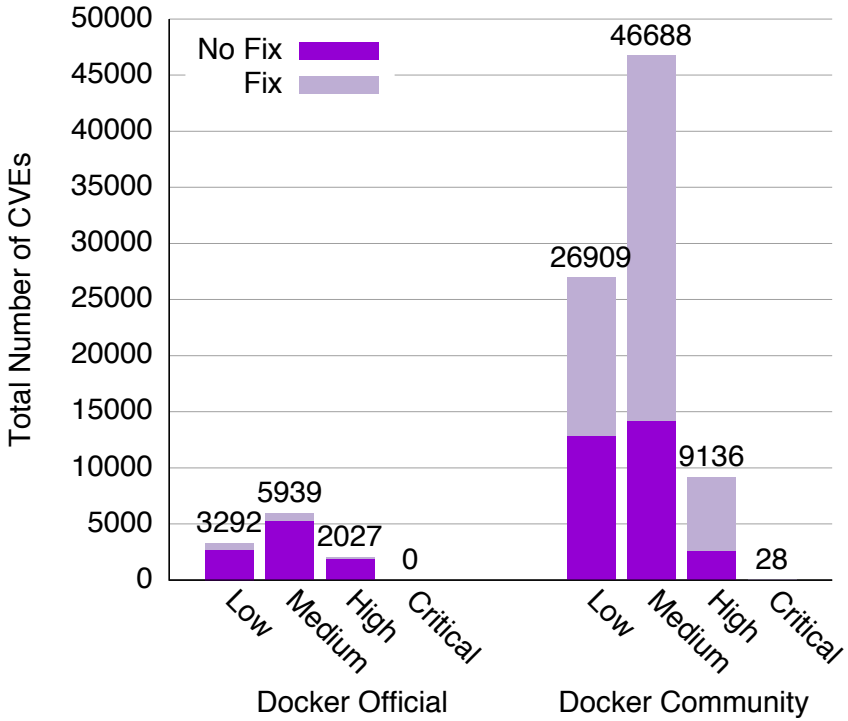
How Docker fits in

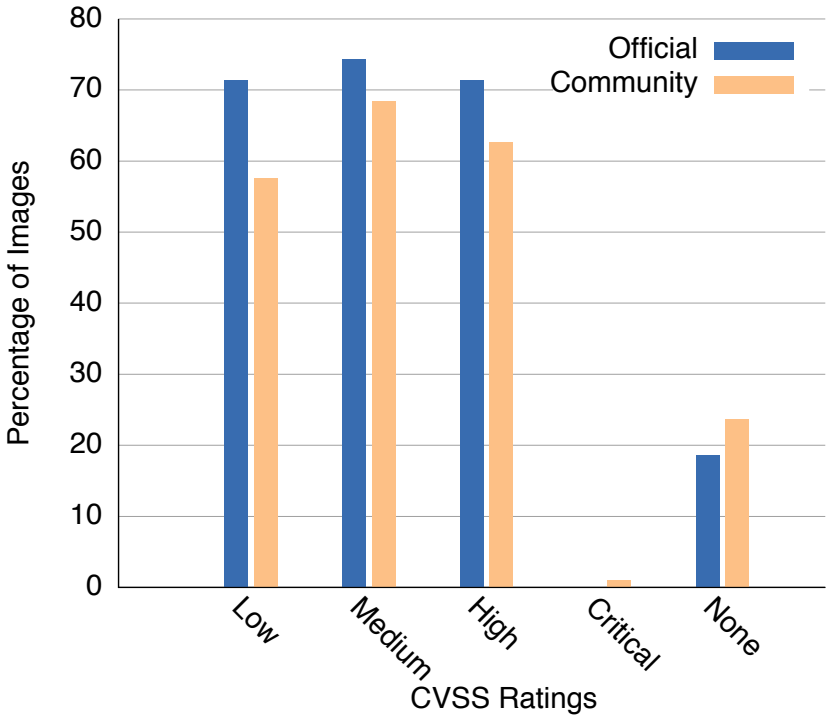


- Utilities that allow you to leverage (e.g.) LXC to build a portable, self-sufficient application using containers.
- Assures that all libraries and dependencies are packaged inside of a container image

Docker Attack Surface?

- Docker provides something analogous to an app market for building containers.
- Are the container images 'secure'?





[Hassan et al., NDSS'18]

Above the clouds…

- Container (~PaaS clouds) are strictly easier to manage than traditional IaaS VMs.
- The era of Container hype has somewhat come and gone… containers still expose more flexibility than most users need!!
- Th hype now is about Function-as-a-Service cloud; individual programs/functions executed by invocation, great for event-driven stuff.
- Enabled by containers

FaaS Threat Vectors

- Functions execution inside a container
- For performance 'warm' containers are potentially re-used across multiple functions/customers
  - i.e., numerous *explicit* data channels (unlike IaaS)
- Attack Strategy:
  - Exploit Function A, Instance 1 (Attack Request)
  - Write malware to a /tmp used by Function A
  - Malware exploits Function A, Instance 2 (Victim Request)
  - Attacker quickly exfiltrates victim data