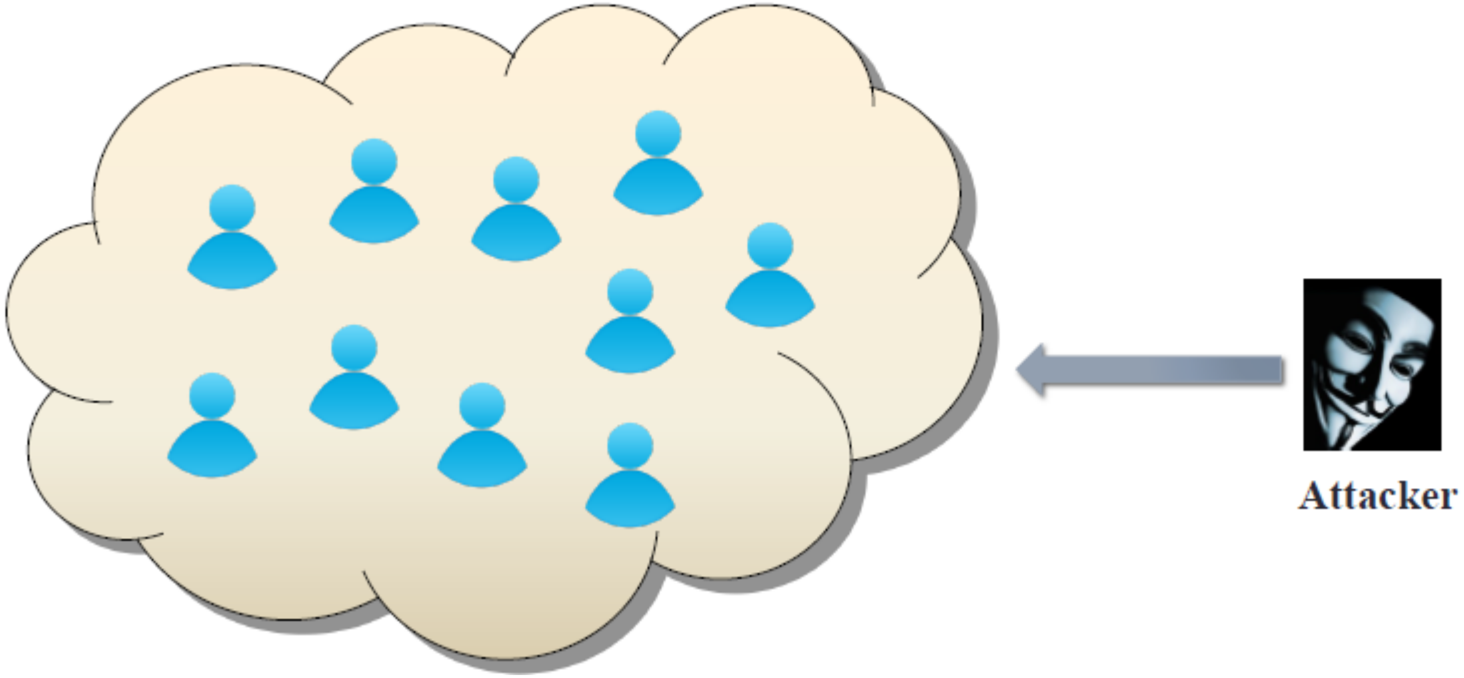
# Week 11 Cloud Security

Dr Zhi Zhang

# Public cloud becomes tempting target



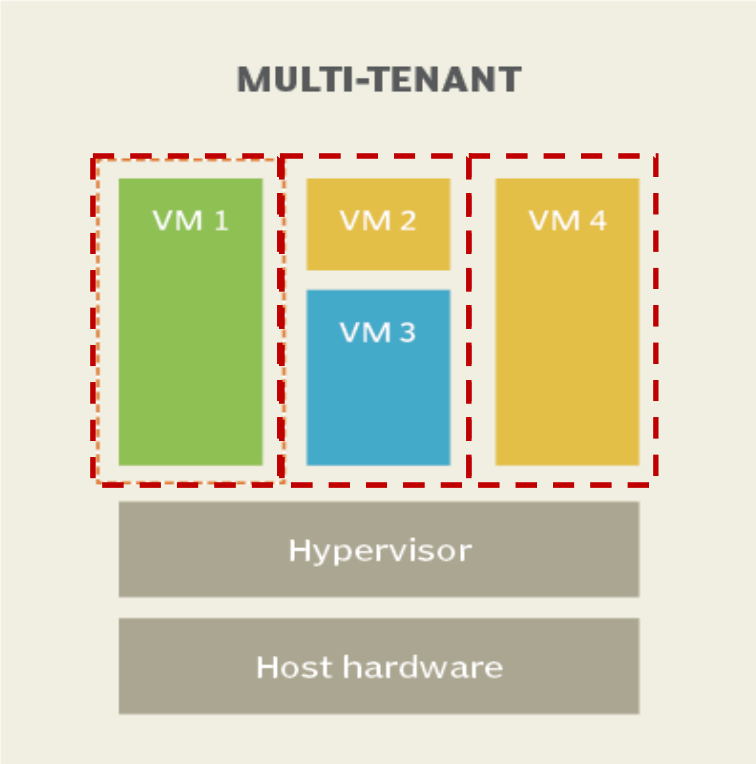

- Valuable targets:
  - banking, medical information, DNN models

# Overview

- Introduction to key terms in public cloud security
- Co-residence attacks and mitigations
- Cross-VM attacks and mitigations

# Key Terms

- **Multi-tenancy**:
  - IaaS clouds multiplex virtual machines (VMs) of disjoint customers upon the same physical machines.
- **Co-residency**:
  - A malicious VM is co-resident with a target VM upon the same physical machine.




https://www.techtarget.com/whatis/definition/multi-tenancy

# Key Terms

- Multi-tenancy:
- Co-residency:
- **Placement**:
  - The adversary manages to place their malicious VM onto the same physical machine as their target VM.

# Key Terms

- Multi-tenancy:
- Co-residency:
- Placement:
- Cross-VM attack :
  - The adversary from the malicious VM breaks data confidentiality or integrity against the target VM, so-called cross-VM attack.
    - e.g., side-channel, hardware fault
  - **Side-channel attacks:** the adversary attempts to infer a secret from a victim by gathering information about security-critical operations performed by the victim.
    - e.g., power consumption based side channel
  - **Hardware-fault attacks:** the adversary disrupts the normal operations of the computer hardware, compromising the integrity of the victim or the whole system.
    - e.g., rowhammer

# From the Attacker’s perspective

- **Step 1:**
  - map the internal cloud infrastructure (cloud cartography)
- **Step 2:**
  - check if two instances are co-resident on the same physical machine
- **Step 3:**
  - launch a VM that is highly likely to be co-resident with the target VM
- **Step 4:**
  - launch a cross-VM attack

# Attacker’s capabilities

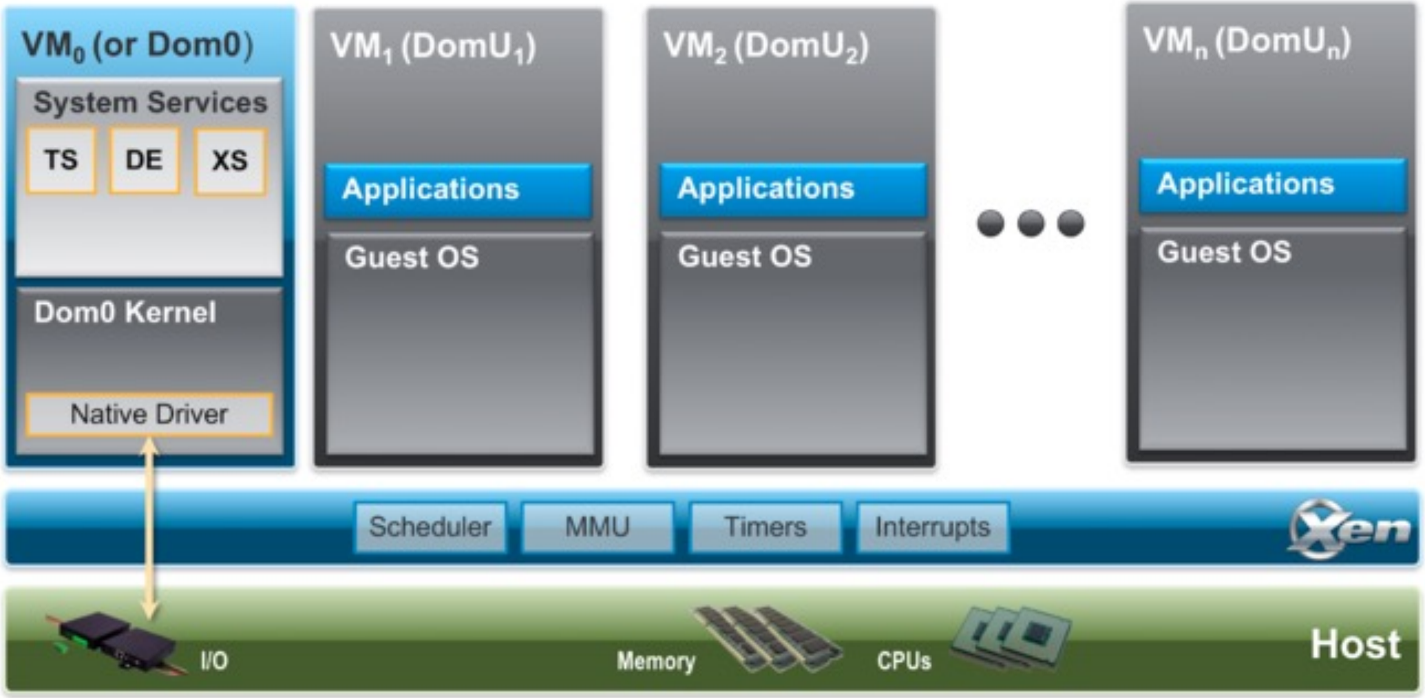
- Not affiliated with public cloud provider
- Can run many VMs
- Their VMs can be placed on the same physical hardware as victim VMs

# Case study: Amazon EC2

- **Clients:**
  - specify region, availability zone and instance type.
- **Instances:**
  - placed onto single physical machine
  - assigned external IP addresses and internal IP addresses.

# Case study: Amazon EC2

- Clients:
- Instances:
- **Xen hypervisor:**
  - domain0 (dom0): routes packets and reports itself as a hop.



# Step 1: map the internal cloud infrastructure

- A challenge: Instance placement is not disclosed by Amazon but needed by the attacker
  - Map the EC2 service to understand where instances are placed
- A hypothesis (Ristenpart et al. [CCS'09]) :
  - Different availability zones and instance types correspond to different internal IP address ranges (a private IP address indicates the physical location of a given instance)
- Survey public servers on EC2

# Survey public servers on EC2

- Create a set of public EC2-based web services (Ristenpart et al. [CCS’09])
  - Use *WHOIS* to identify distinct public IP address prefixes associated with instances.

```
cits1003@cits1003-virtualbox:~$ whois 216.239.32.10

#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy_reporting/
#
# Copyright 1997-2023, American Registry for Internet Numbers, Ltd.
#


NetRange:       216.239.32.0 - 216.239.63.255
CIDR:           216.239.32.0/19
NetName:        GOOGLE
NetHandle:      NET-216-239-32-0-1
Parent:         NET216 (NET-216-0-0-0-0)
NetType:        Direct Allocation
OriginAS:
Organization:   Google LLC (GOGL)
RegDate:        2000-11-22
Updated:        2012-02-24
Ref:            https://rdap.arin.net/registry/ip/216.239.32.0
```

# Survey public servers on EC2

- Create a set of public EC2-based web services (Ristenpart et al. [CCS’09])
  - Use *WHOIS* to identify distinct public IP address prefixes associated with EC2
  - Use external networking probing to find responsive IPs
    - Tools used to probe ports (e.g., 80 and 443), e.g., *nmap, hping* and *wget*

nmap: performs TCP connect probes

```
cits1003@cits1003-virtualbox:~$ nmap -sT -p 80,443,22 216.239.32.10
Starting Nmap 7.80 ( https://nmap.org ) at 2023-10-09 15:02 AWST
Nmap scan report for ns1.google.com (216.239.32.10)
Host is up (0.061s latency).

PORT     STATE    SERVICE
22/tcp   filtered ssh
80/tcp   open     http
443/tcp  open     https

Nmap done: 1 IP address (1 host up) scanned in 2.96 seconds
```

# Survey public servers on EC2

- Create a set of public EC2-based web services (Ristenpart et al. [CCS'09])
  - Use *WHOIS* to identify distinct public IP address prefixes associated with EC2
  - Use external networking probing to find responsive Ips
    - Tools used to probe ports (80 and 443), e.g., *nmap, hping* and *wget*

**hping:** performs TCP traceroutes

```
cits1003@cits1003-virtualbox:~$ sudo hping3 --traceroute -S  -p 443 216.239.32.10
HPING 216.239.32.10 (enp0s3 216.239.32.10): S set, 40 headers + 0 data bytes
hop=1 TTL 0 during transit from ip=10.0.2.2 name=_gateway
hop=1 hoprtt=7.5 ms
len=46 ip=216.239.32.10 ttl=64 id=28958 sport=443 flags=SA seq=1 win=65535 rtt=119.2 ms
```

# Survey public servers on EC2

- Create a set of public EC2-based web services (Ristenpart et al. [CCS'09])
  - Use *WHOIS* to identify distinct public IP address prefixes associated with EC2
  - Use external networking probing to find responsive Ips
    - Tools used to probe ports (80 and 443), e.g., *nmap, hping* and *wget*

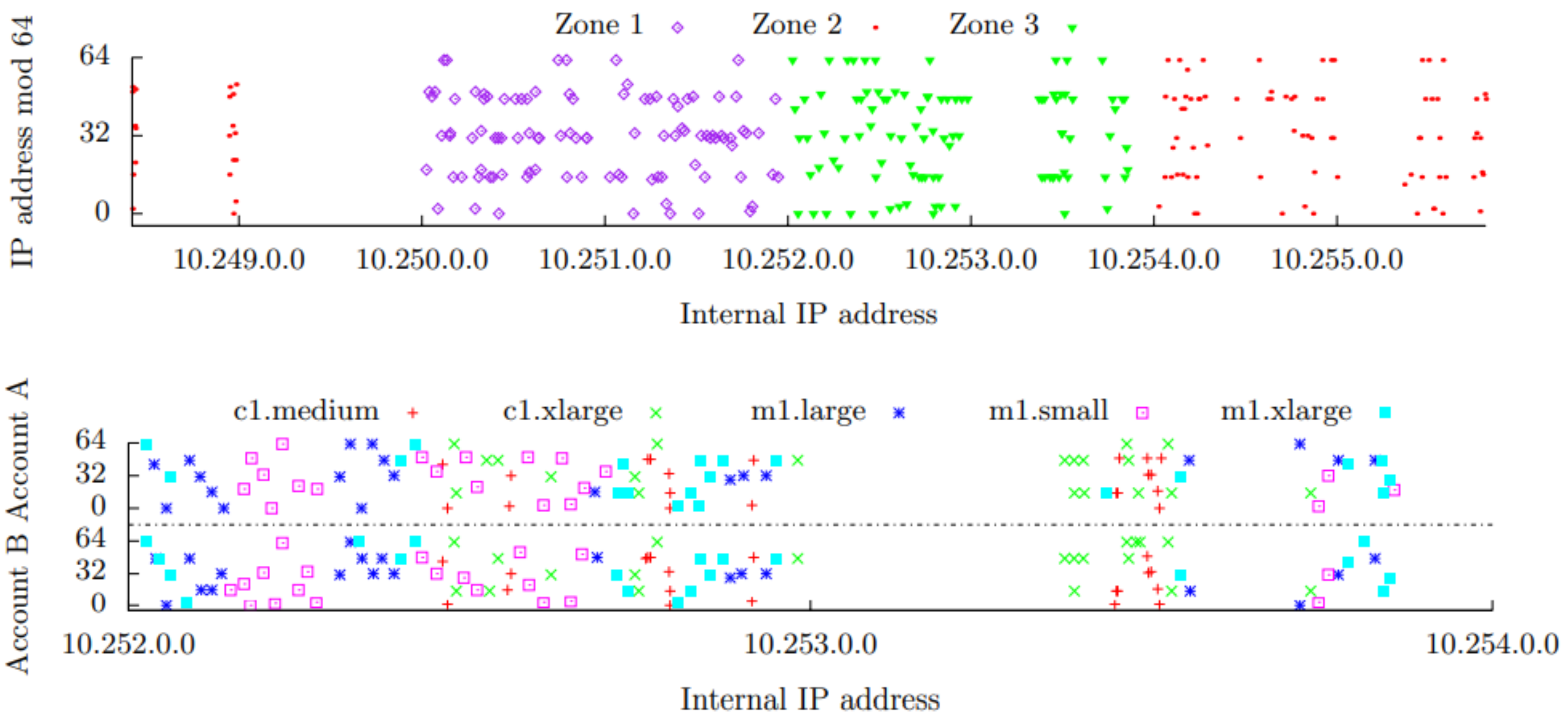
**wget:** retrieves web pages

```
cits1003@cits1003-virtualbox:~$ wget https://www.google.com
--2023-10-09 15:20:46--  https://www.google.com/
Resolving www.google.com (www.google.com)... 142.250.70.228, 2404:6800:4015:803::2004
Connecting to www.google.com (www.google.com)|142.250.70.228|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html'

index.html                   [ <=>                                ]  17.47K  --.-KB/s    in 0.03s

2023-10-09 15:20:46 (574 KB/s) - 'index.html' saved [17891]
```

# Survey public servers on EC2

- Create a set of public EC2-based web services (Ristenpart et al. [CCS'09])
  - Use *WHOIS* to identify distinct public IP address prefixes associated with EC2
  - Use external networking probing to find responsive IPs
    - Tools used to probe ports (80 and 443), e.g., *nmap, hping* and *wget*
  - Map all responsive public IPs to private IPs using the EC2's internal DNS

# EC2 Instance placement properties




Ristenpart et al. [CCS'09]

# Step 2: check if two instances are co-resident on the same physical machine

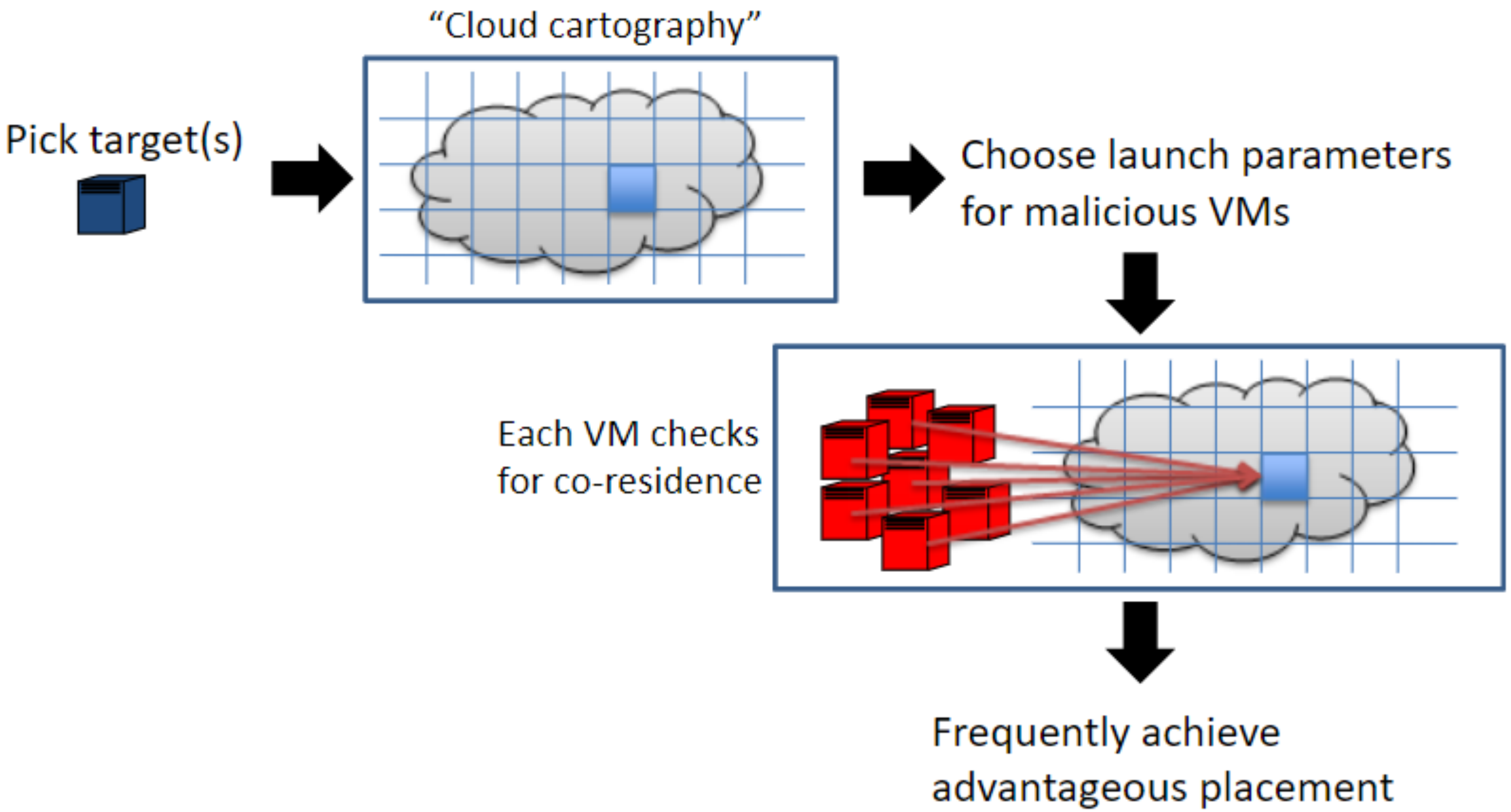
- Use the set of created instances.
- Check for co-residence (network-based): (Ristenpart et al. [CCS'09])
  - Match Dom0 public IP address,
  - Show small packet round-trip time, or
  - Exhibit numerically close internal IP address (e.g., within 7)

# Step 3: launch a VM that is highly likely to be co-resident with the target VM

- Brute-force placement
- Temporal locality in placement
  - instances that are launched at the same time are more likely to be assigned to the same physical machine

Ristenpart et al. [CCS'09]

# Steps 1-3: achieving co-residence with a target VM




Ristenpart et al. [CCS'09]

# Mitigating co-residence attacks

- Updating VM placement policy
- Updating networking management
- Deploying VPC

# Updating VM placement policy

- Larger pool of physical machines
  - More machines host the same type of instances.

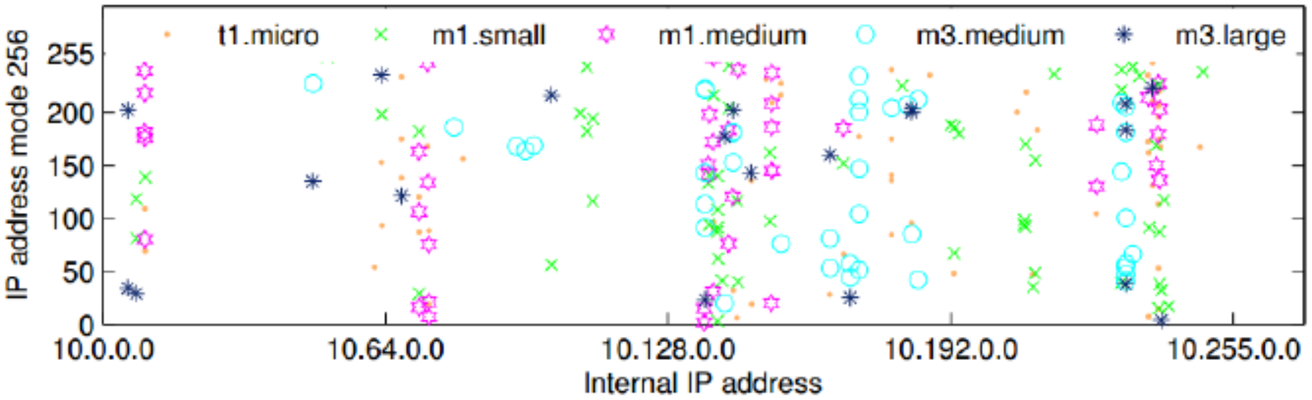
# Updating VM placement policy

- Larger pool of physical machines
- Reduced placement locality

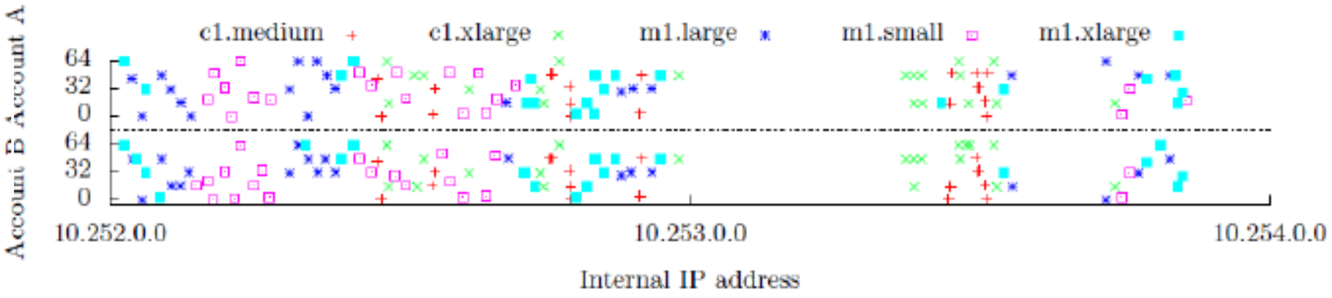
# Updating VM placement policy

- Larger pool of physical machines
- Reduced placement locality
  - Reduced type locality

recent study



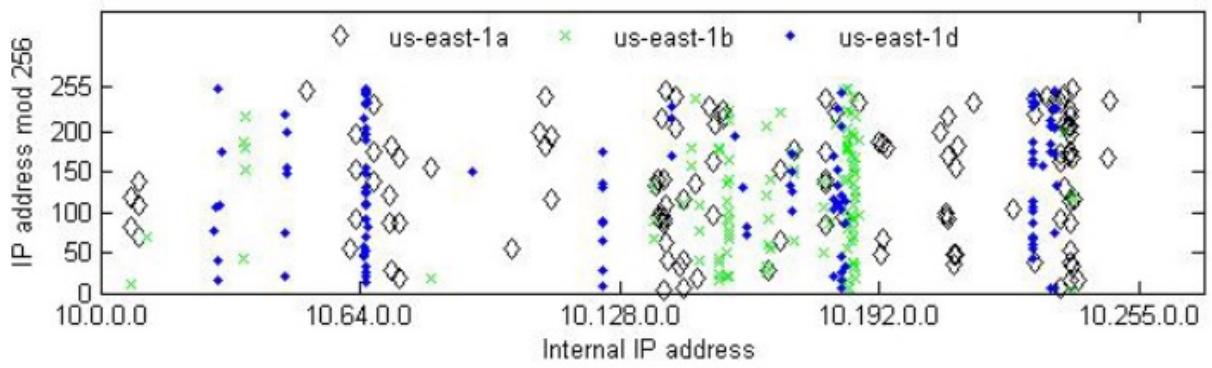

previous study



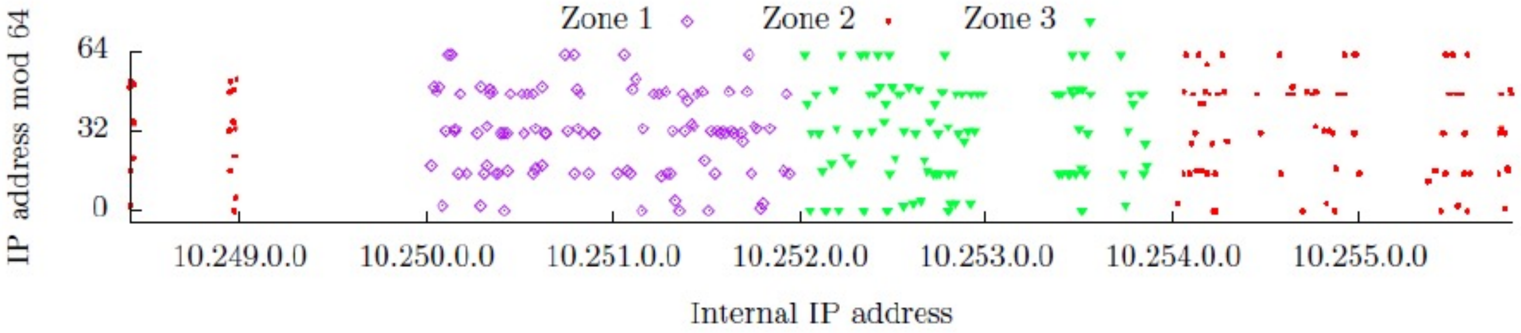
# Updating VM placement policy

- Larger pool of physical machines
- Reduced placement locality
  - Reduced zone locality

recent study




previous study



# Updating VM placement policy

- Larger pool of physical machines
- Reduced placement locality
- More dynamic assignment of placing virtual machines
  - Public-private IP address mapping changes frequently
  - Different types of instances can be placed in same physical machine

# Updating networking management

- Hide Domain0
  - Domain0 is the identifier of a physical machine
  - Domain0 is hidden in a network routing path
- Hide information of specific hops in the routing path
  - Sensitive routers and switches do not appear in the trace-routing path

```
Traceroute 54.88.197.86 (54.88.197.86), 30 hops max, 60 byte packets
1  10.210.136.3 (10.210.136.3) 1.248 ms  1.303 ms  1.501 ms
2  ip-10-1-14-17.ec2.internal (10.1.14.17)  0.529 ms  0.653 ms  0.781 ms
3  ip-10-1-172-2.ec2.internal (10.1.172.2)  0.492 ms  0.604 ms  0.729 ms
4  *  *  *
5  *  *  *
6  ec2-54-88-197-86.compute-1.amazonaws.com (54.88.197.86)  1.048 ms  0.883 ms
```

# Deploying VPC

- VPC is a service widely used in EC2 to enhance cloud security
  - Provides an isolated networking environment to host instances
  - Private IP address is invisible to outsiders

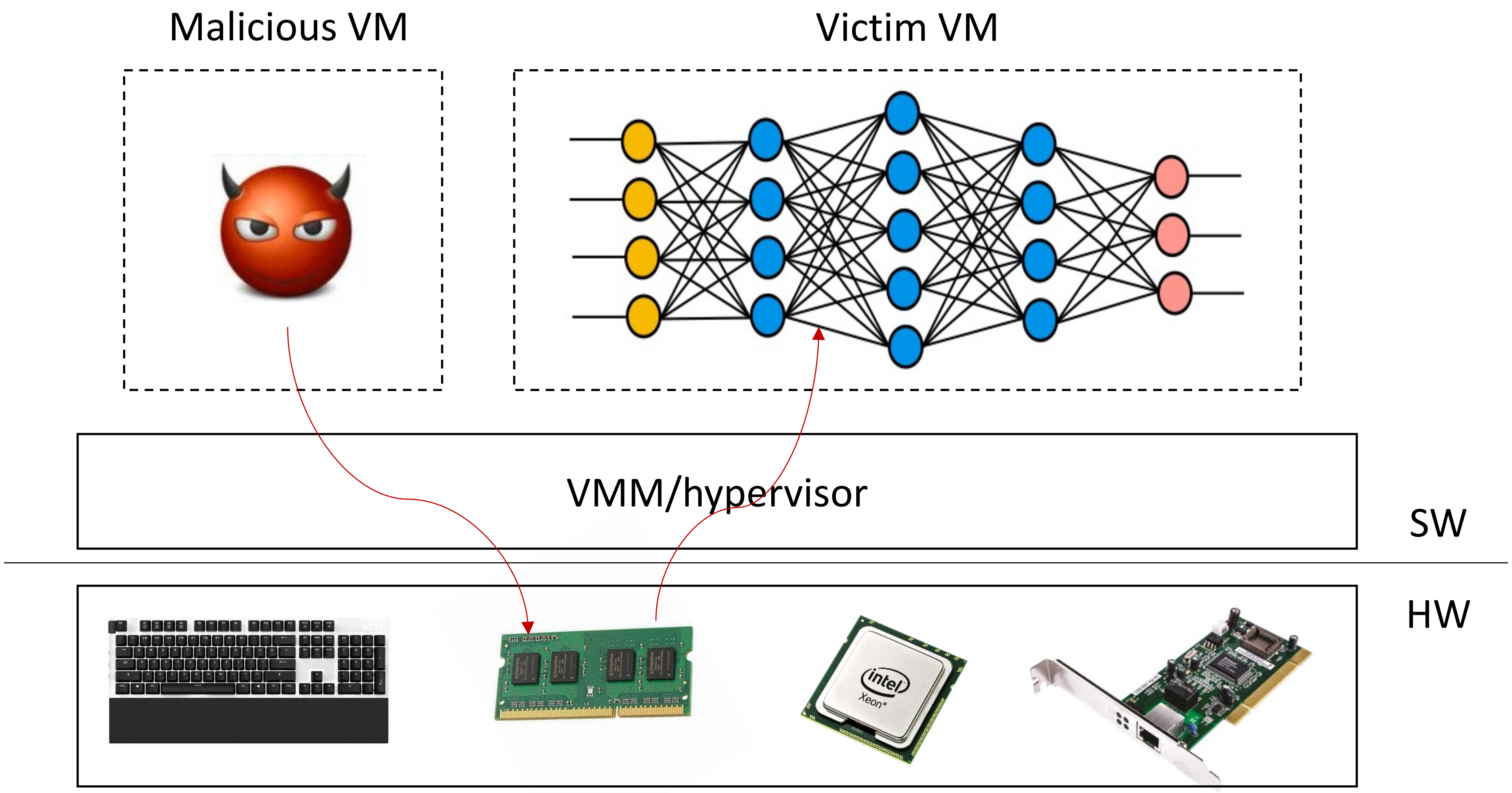
Is it possible to achieve co-residence in the presence of the countermeasures?

Yes!

# Step 4: launch cross-VM attacks

- **hardware fault attacks:**
  - e.g., rowhammer-induced accuracy depletion in deep-learning models
- **side channel attacks:**
  - e.g., power consumption-based deep-learning model stealing

# Rowhammer-induced accuracy depletion in deep-learning models




DRAM memory is prone to rowhammer faults.

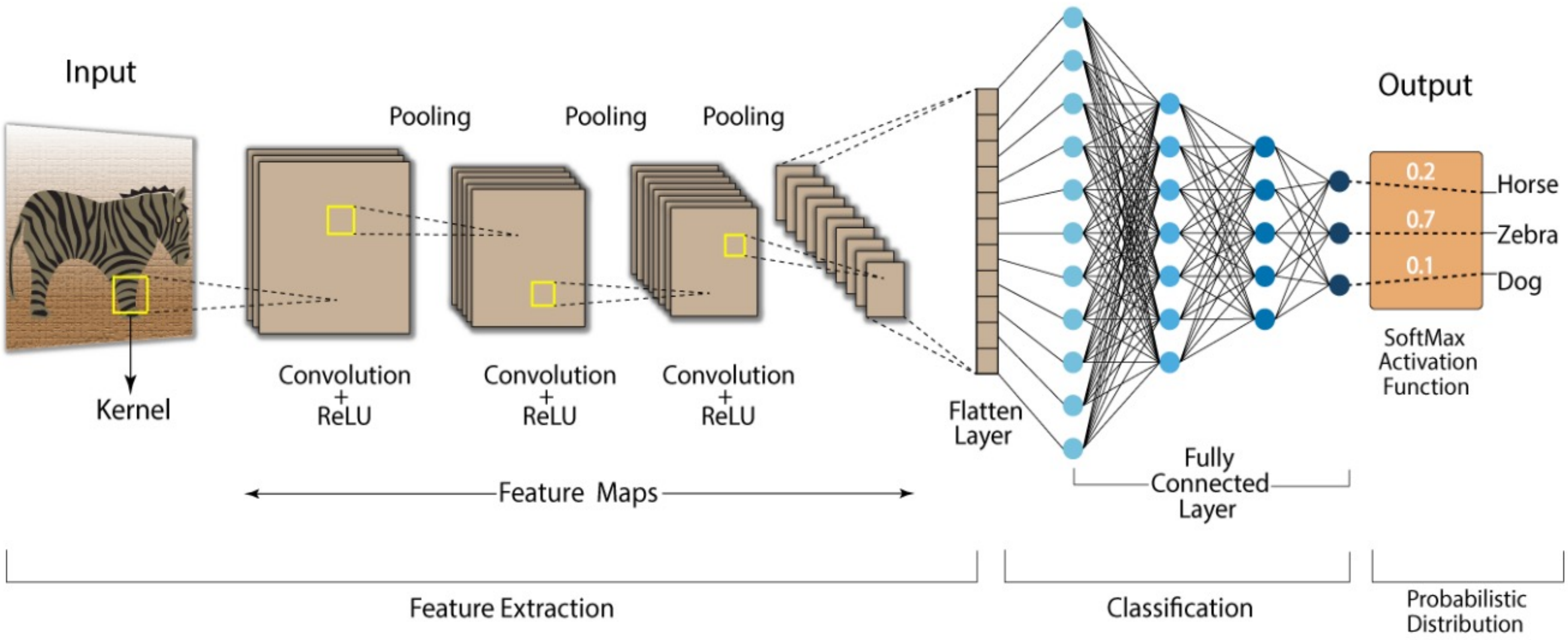
# DNN key terms

- Training: feed-forward and back-propagation
  - For given input data, it goes through from the input layer to the output layer (feed-forward). The resultant output is compared against ground truth via a loss function. To minimize the loss, the weights of the network are updated accordingly (back-propagation).
- Inference: a well-trained model is used to infer real-world unseen data

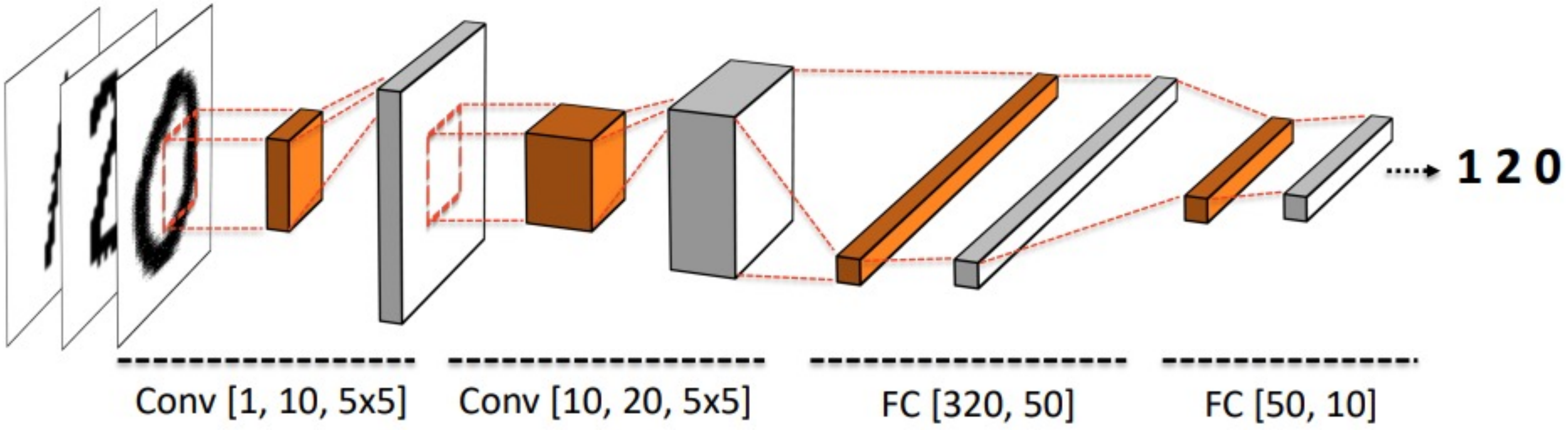
# DNN key terms

- Training: feed-forward and back-propagation
- Inference: a well-trained model is used to infer real-world unseen data
- **DNN architecture**: the whole set of processing layers.
- **Hyperparameters**: decides a DNN architecture.
  - layers and layer-wise hyperparameters
    - layers: the number of layers and the type of each layer, e.g., convolutional layer, pooling layer, and fully-connected layer.
    - layer-wise hyperparameters, e.g., the neuron number for a fully-connected layer.
- **Parameters**: configuration variables (i.e., weights and bias) of a specific model and their values are decided via training.

# An example: CNN



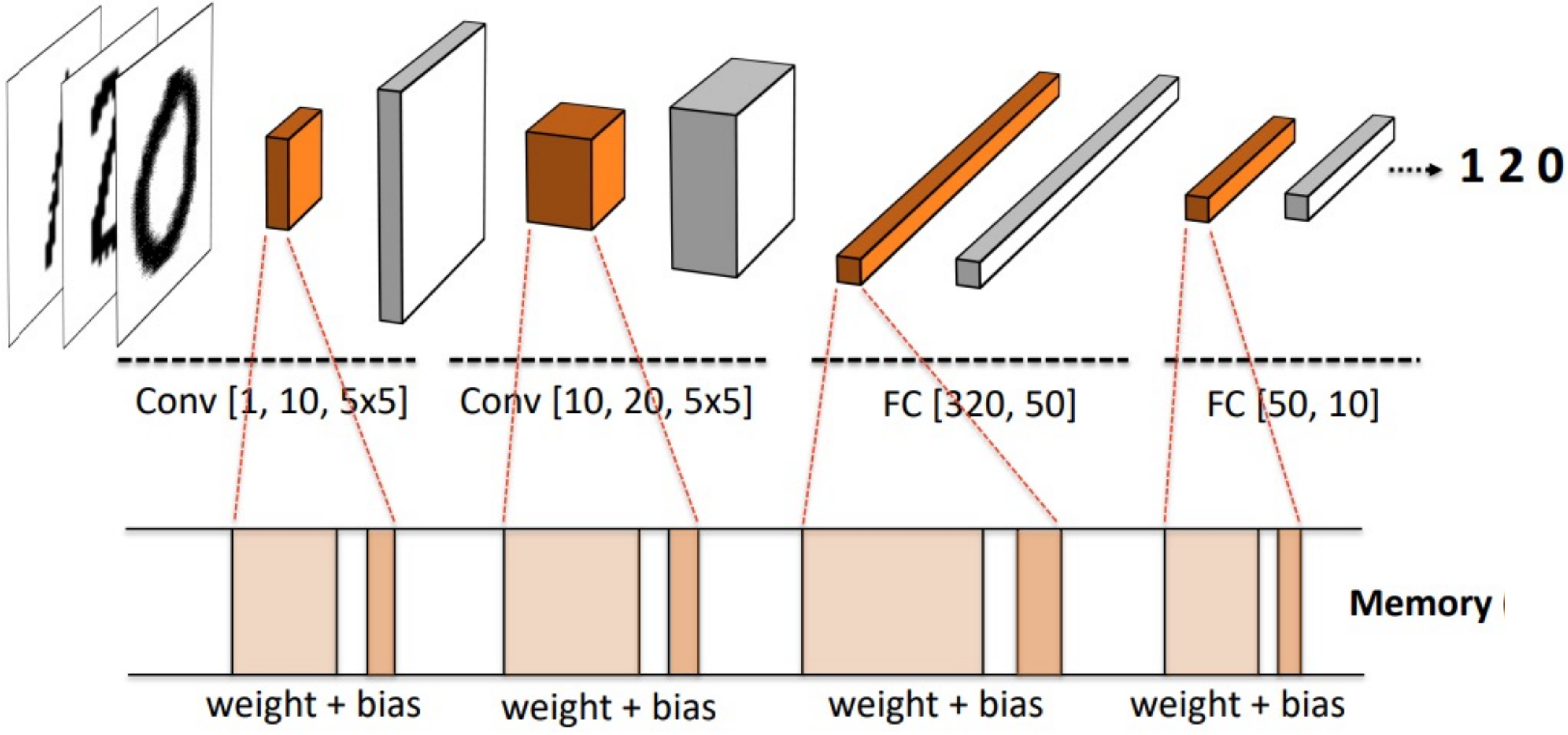
# How DNN Computes




Prediction accuracy: 98.53%

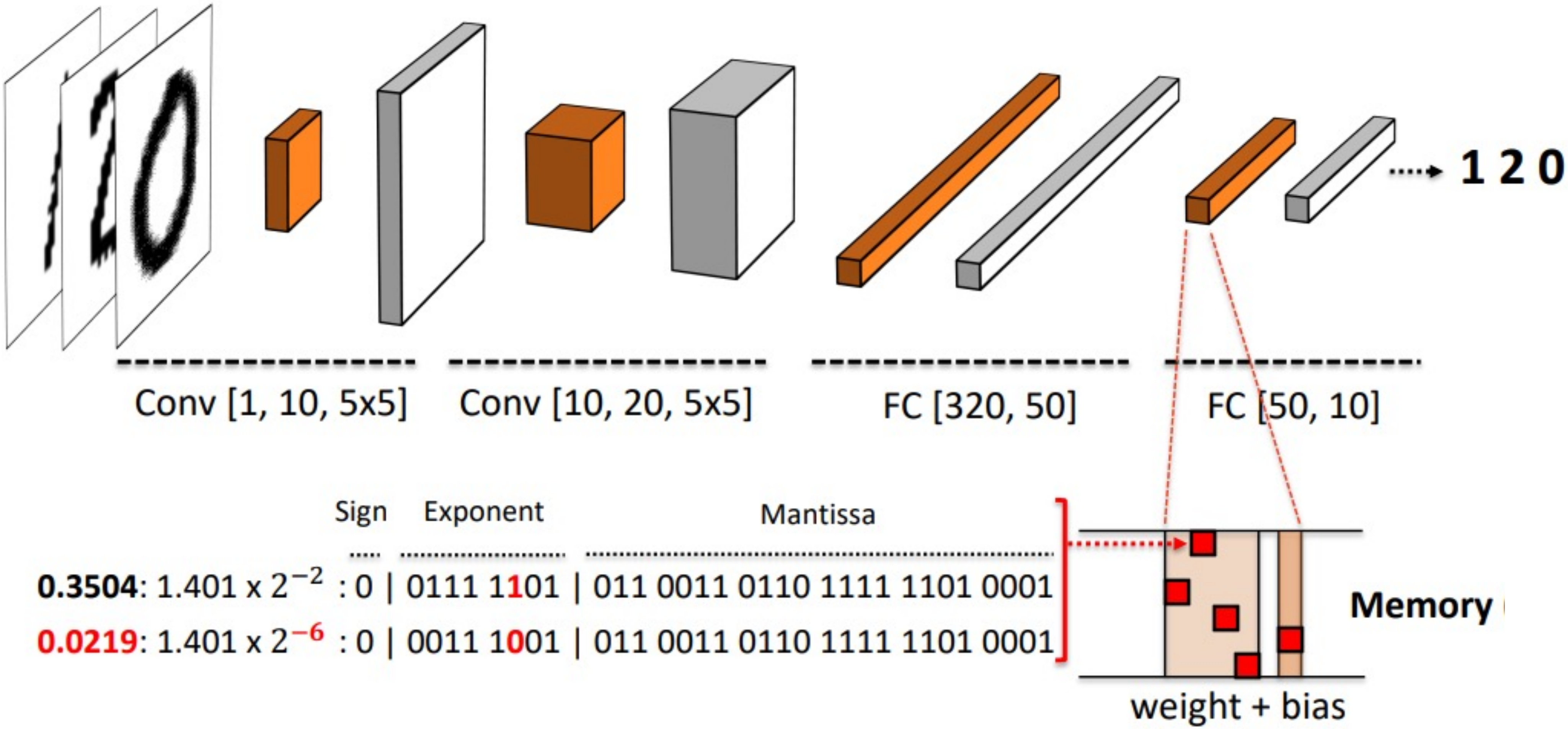
https://www.usenix.org/conference/usenixsecurity19/presentation/hong

# DNN parameters are stored in memory




https://www.usenix.org/conference/usenixsecurity19/presentation/hong

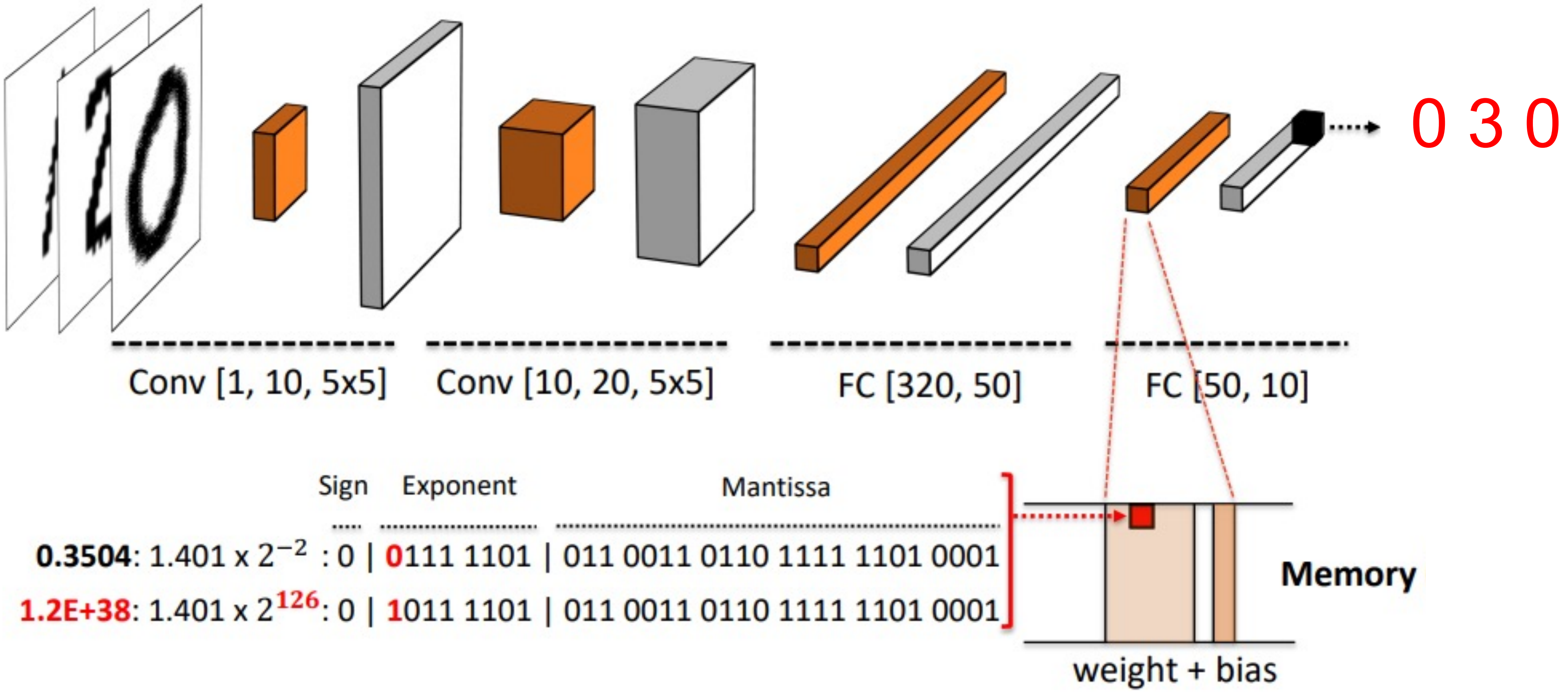
# Faulting unimportant parameters




Prediction accuracy: 93.53% (5% drop)

https://www.usenix.org/conference/usenixsecurity19/presentation/hong

# Faulting important parameters → Accuracy depletion



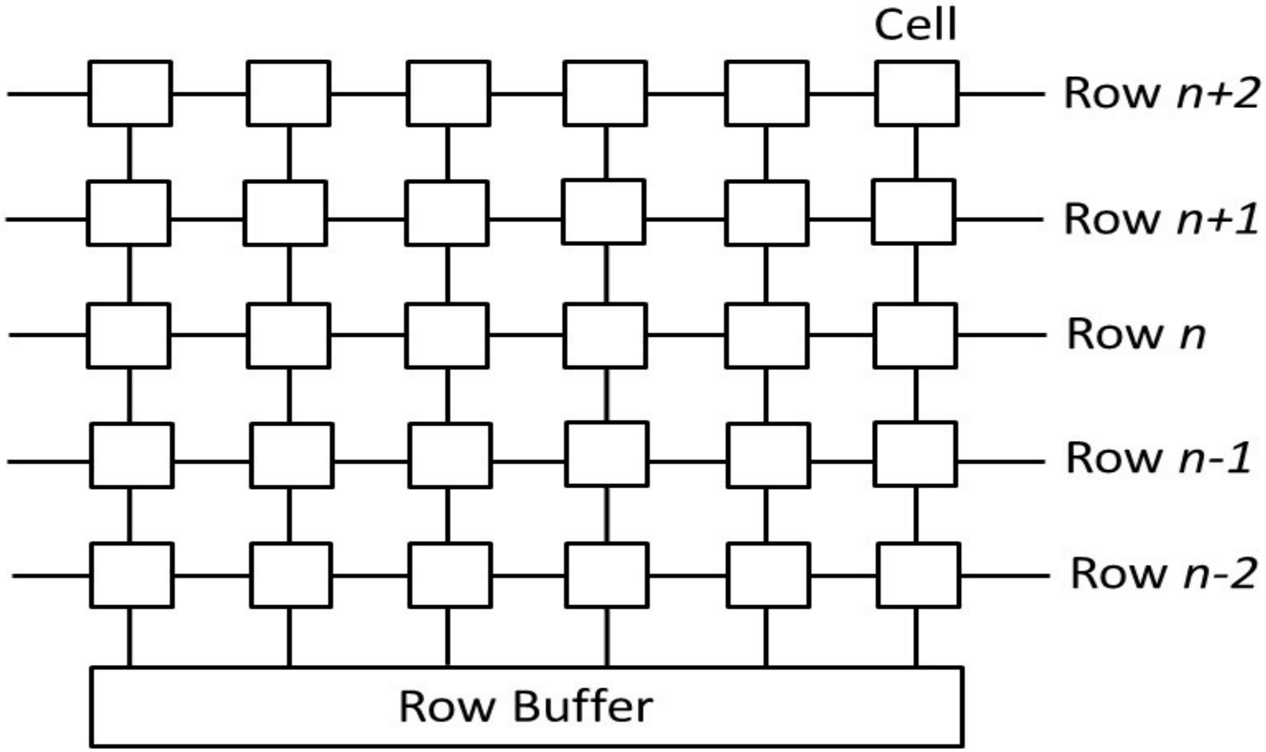

Prediction accuracy: 57.53% (41% drop)

https://www.usenix.org/conference/usenixsecurity19/presentation/hong

What is Rowhammer ?

Rowhammer: Frequently accessing DRAM rows

- A memory access opens a DRAM bank
- A bank has rows of cells
- A cell is either charged or discharged to store bit 0 or 1
- Row buffer facilitates memory access

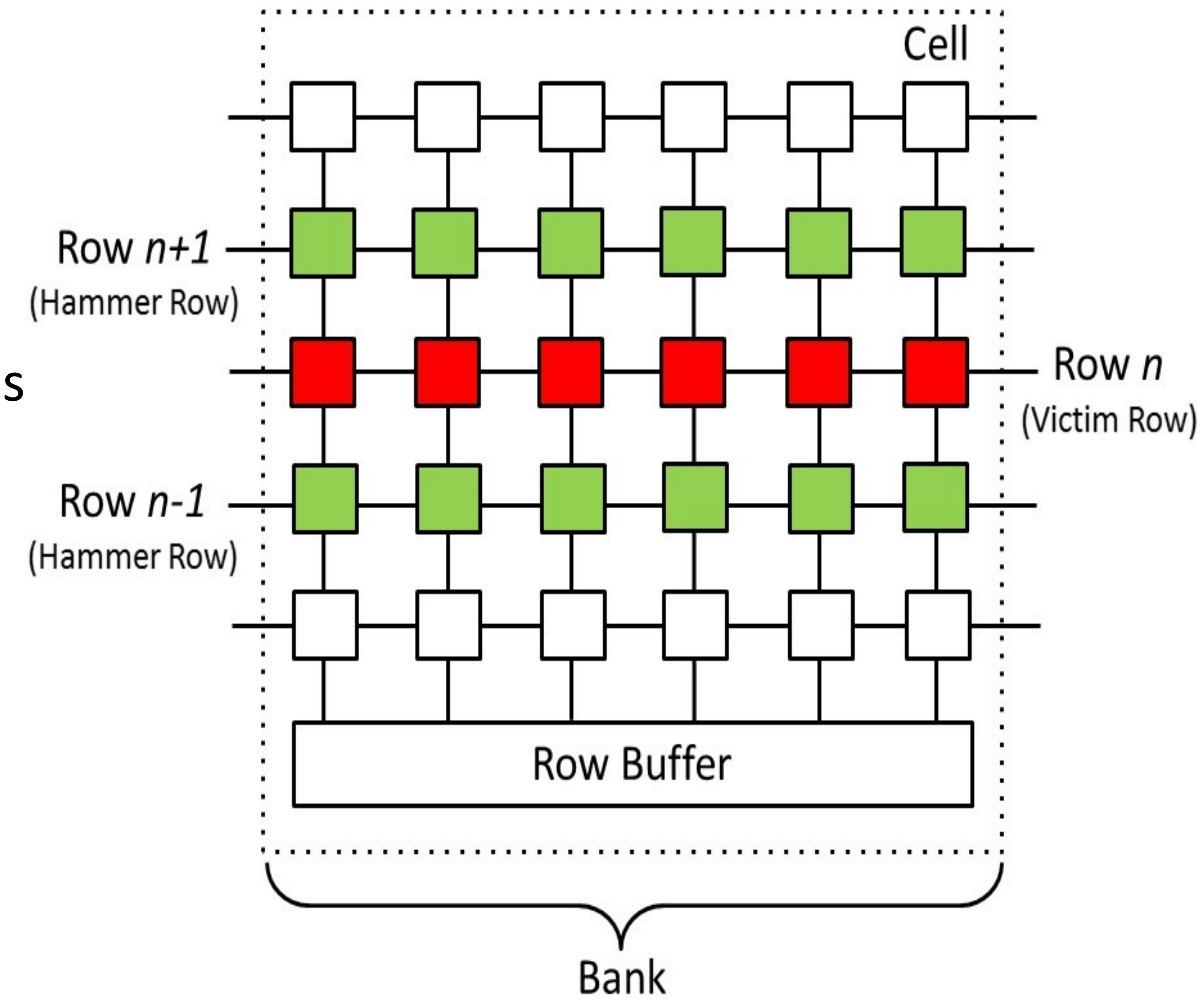



- Alternating row access clears row buffer

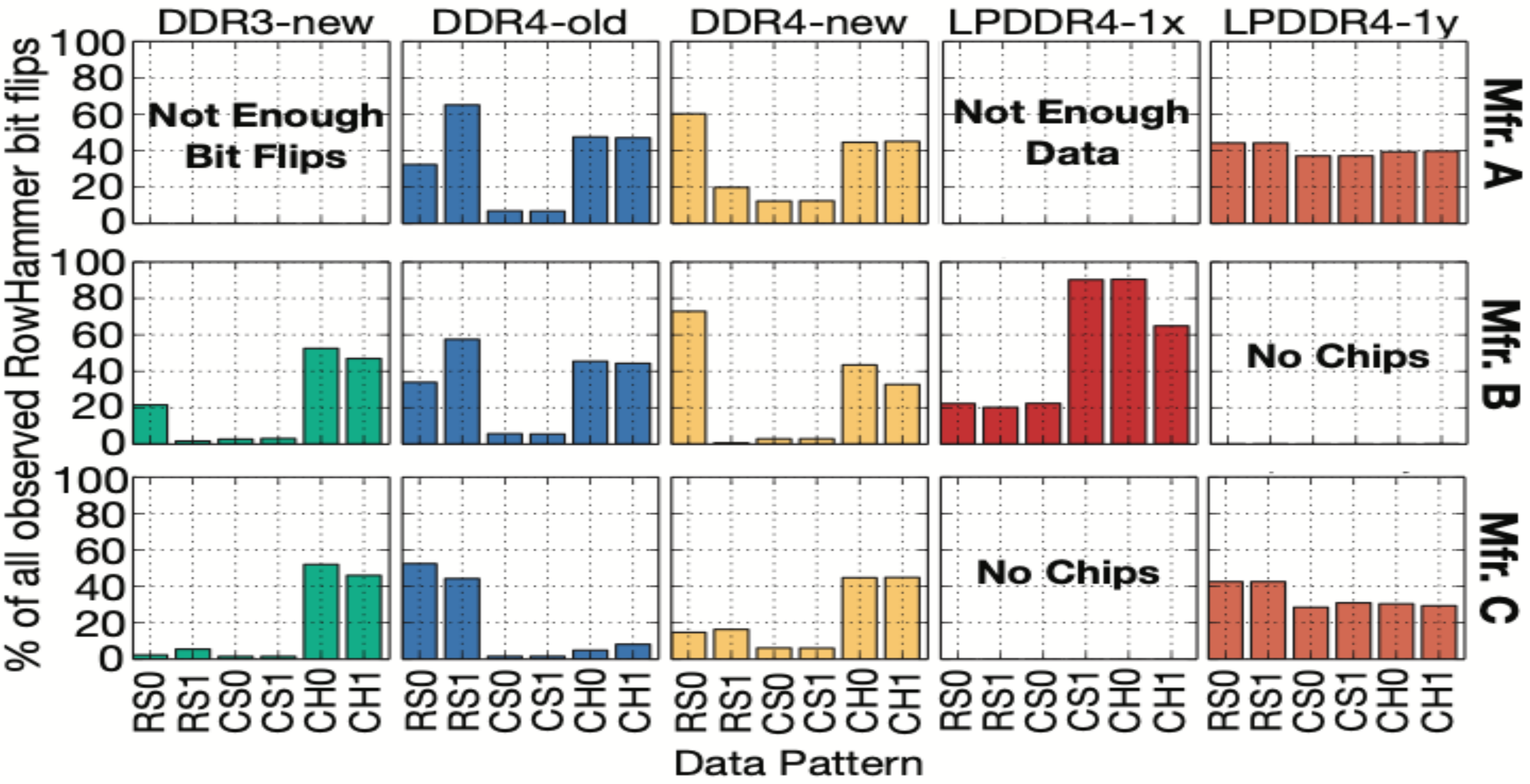
# Rowhammer: a DRAM fault

Kim et al. (ISCA'14)

frequently opening rows *n+1* & *n-1* causes charge leakage (bit flips) in row *n*



# Modern DRAM is prone to rowhammer bit flips




Kim et al. (ISCA'20) have induced rowhammer bit flips on 1580 DRAM chips from 300 DRAM modules and from the three major DRAM manufacturers.

# Rowhammer code

What is purpose of lines 2-3?

- Clear row buffer (e.g., alternate access)

```
1 loop:
2     mov (X), %eax
3     mov (Y), %ebx
4     clflush (X)
5     clflush (Y)
6     jmp loop
```

# Rowhammer code

What is the purpose of lines 4-5?

- flush CPU cache (e.g., clflush)

```
1 loop:
2     mov (X), %eax
3     mov (Y), %ebx
4     clflush (X)
5     clflush (Y)
6     jmp loop
```

# Hammer pattern

```
loop:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    jmp loop
```

Double-sided hammer

- X and Y are adjacent to the target row

Single-sided hammer

- Either X or Y is adjacent to the target row

Common

- X and Y must be in the same bank to clear row buffer

# Hammer pattern

```
loop:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    jmp loop
```

**Double-sided hammer**

- X and Y are adjacent to the target row

**Single-sided hammer**

- Either X or Y is adjacent to the target row

**Common**

- X and Y must be in the same bank to clear row buffer

Requirements

Mapping between a virtual address and its physical address

Mapping between a physical address and its DRAM location

# Mapping between a virtual address and its physical address

```c
uint64_t get_physical_addr(uintptr_t virtual_addr) {
    size_t page_size = 0x1000;
    int fd = open("/proc/self/pagemap", O_RDONLY);
    assert(fd >= 0);

    off_t pos = lseek(fd, (virtual_addr / page_size) * 8, SEEK_SET);
    assert(pos >= 0);
    uint64_t value;
    int got = read(fd, &value, 8);
    assert(got == 8);
    int rc = close(fd);
    assert(rc == 0);

    // Check the "page present" flag.
    assert(value & (1ULL << 63));
    uint64_t frame_num = value & ((1ULL << 54) - 1);
    return (frame_num * page_size) | (virtual_addr & (page_size - 1));
}
```

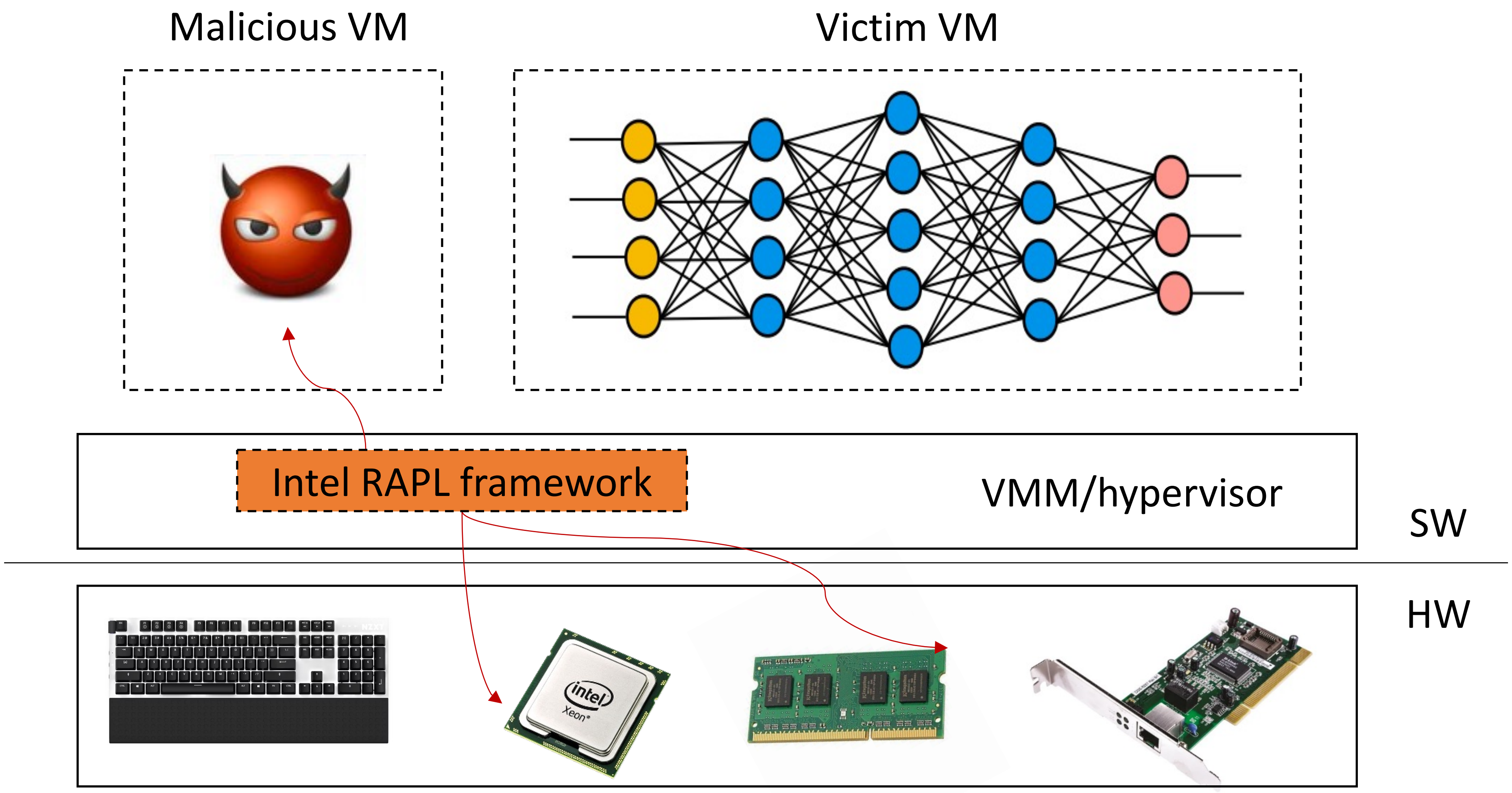
# Mapping between a physical address and its DRAM location

1. bank is decided by certain bits of a physical address.

```c
size_t get_dram_mapping(void* phys_addr_p) {
  uint64_t phys_addr = (uint64_t) phys_addr_p;
  static const size_t h0[] = { 14, 18 };
  static const size_t h1[] = { 15, 19 };
  static const size_t h2[] = { 16, 20 };
  static const size_t h3[] = { 17, 21 };
  static const size_t h4[] = { 17, 21 };
  static const size_t h5[] = { 6 };
  size_t hash = 0; size_t count = sizeof(h0) / sizeof(h0[0]);
  for (size_t i = 0; i < count; i++) {
    hash ^= (phys_addr >> h0[i]) & 1;
  }
  size_t hash1 = 0; count = sizeof(h1) / sizeof(h1[0]);
  for (size_t i = 0; i < count; i++) {
    hash1 ^= (phys_addr >> h1[i]) & 1;
  }
  size_t hash2 = 0; count = sizeof(h2) / sizeof(h2[0]);
  for (size_t i = 0; i < count; i++) {
    hash2 ^= (phys_addr >> h2[i]) & 1;
  }
  size_t hash3 = 0; count = sizeof(h3) / sizeof(h3[0]);
  for (size_t i = 0; i < count; i++) {
    hash3 ^= (phys_addr >> h3[i]) & 1;
  }
  size_t hash4 = 0; count = sizeof(h4) / sizeof(h4[0]);
  for (size_t i = 0; i < count; i++) {
    hash4 ^= (phys_addr >> h4[i]) & 1;
  }
  size_t hash5 = 0; count = sizeof(h5) / sizeof(h5[0]);
  for (size_t i = 0; i < count; i++) {
    hash5 ^= (phys_addr >> h5[i]) & 1;
  }
  return (hash5 << 5) | (hash4 << 4) | (hash3 << 3) | (hash2 << 2) | (hash1 << 1) | hash;
}
```

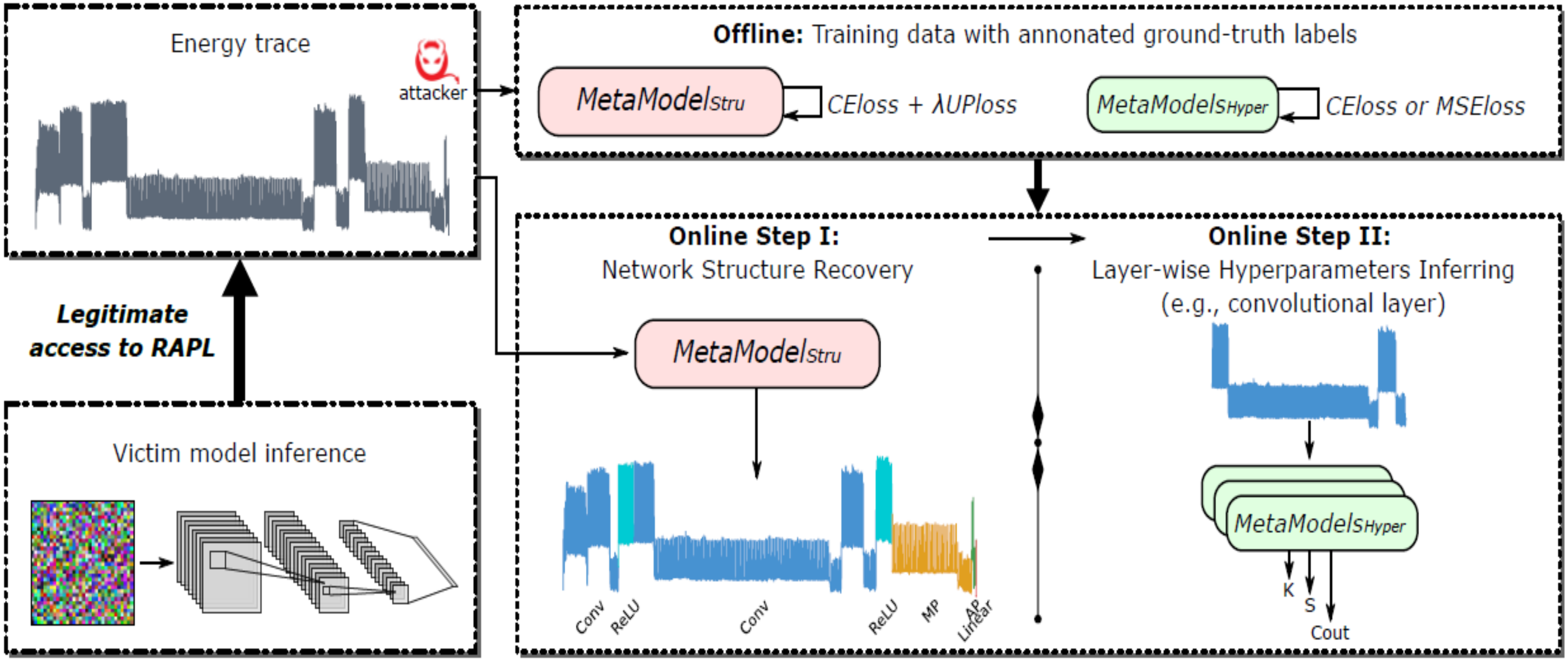
2. Row index is decided by bits 18-32 of a physical address.

Power consumption-based deep-learning model stealing




CPU and DRAM power consumption are available to VMs via Intel RAPL (Running Average Power Limit) framework.

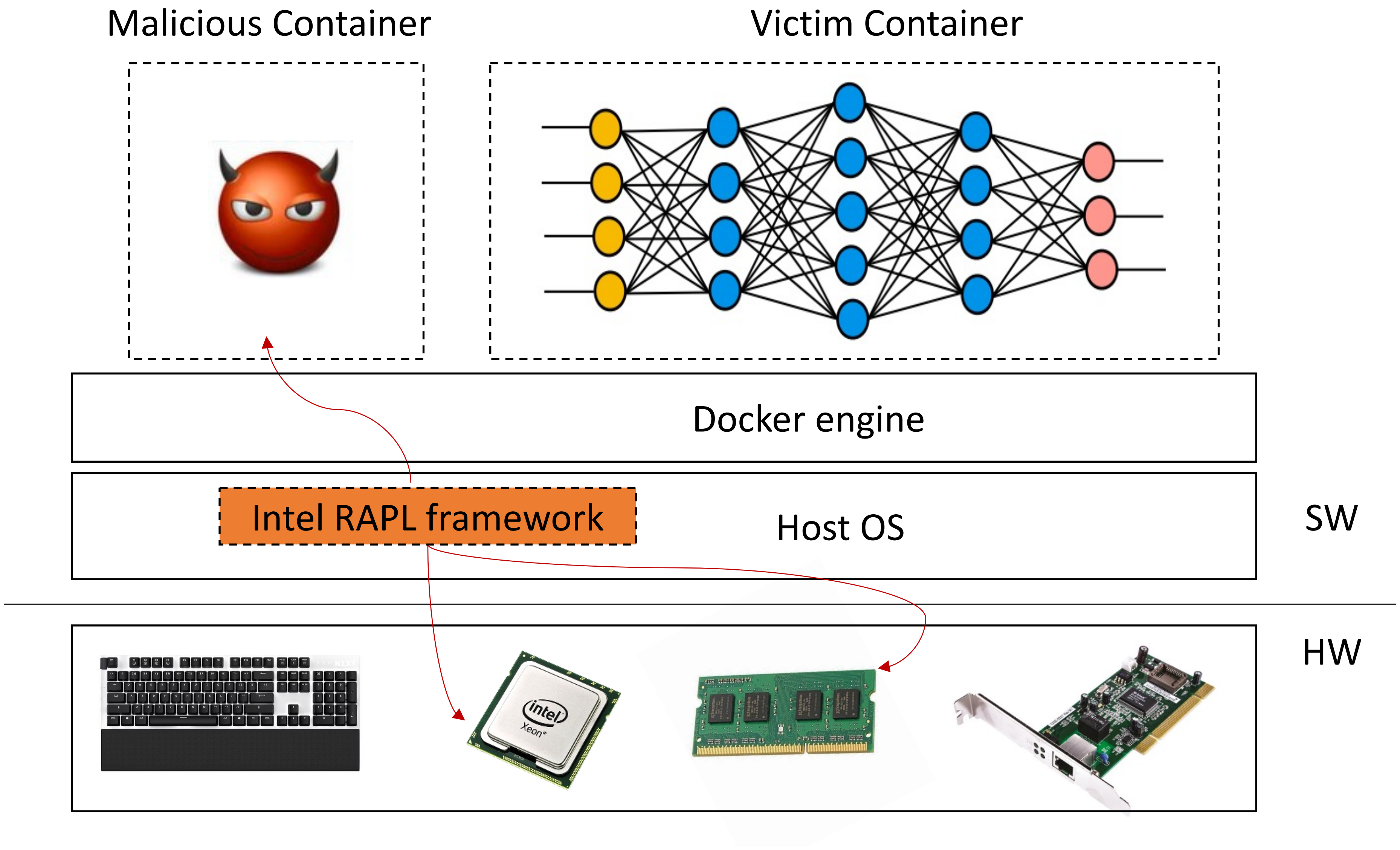
# The flow of model stealing




Zhang et al. (S&P'24)

Based on our observation, some (not all) EC2 instances have provided access to RAPL

| AWS EC2 | Enabled | Disabled |
|---|---|---|
| Instance Types | c4.8xlarge, r4.8xlarge, d2.8xlarge, i3.8xlarge, x1e.8xlarge, h1.8xlarge | t1.micro, t2.xlarge, t3.xlarge, c3.2xlarge, c4.2xlarge, c5.2xlarge |




A proof-of-concept (PoC) attack has been demonstrated in a recent docker environment, i.e., version 20.10.1

Zhang et al. (S&P'24)

# Docker container

**Gabriela Georgieva** <gabriela.georgieva@docker.com>
to Security, me, G

Jul 14, 2023, 12:46 AM

Hi Zhi and Yansong,

We have reviewed your report and agree that this is a security issue. Thank you for bringing it to our attention. We are planning to address it by adding the relevant [redacted] be already covered as they require [redacted] We aim patch this for all of moby, containerd and runc.

We will keep you updated on the status of the issue. Please let us know if you have any questions or comments!

Best regards,
Security Team @ Docker, Inc.

**CVE-2023-5453** has been reserved tentatively.

What is CVE?

short for Common Vulnerabilities and Exposures. The CVE list is a National Vulnerability Database (NVD) maintained by the U.S. government.