



# Bifrost: Analysis and Optimization of Network I/O Tax in Confidential Virtual Machines

**Dingji Li**, Zeyu Mi, Chenhui Ji, Yifan Tan, Binyu Zang,  
Haibing Guan, and Haibo Chen

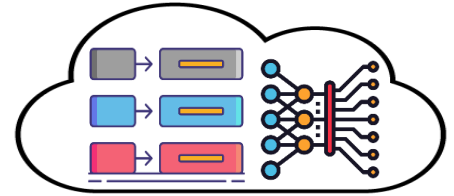
*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University  
Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China  
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University  
Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University*



# Data Security is Crucial for Cloud Computing

- **Applications are processing sensitive data in the cloud**

- For example, key-value store, AI inference, financial service, etc.



KV-store & AI in the Cloud

- **A compromised hypervisor → steal VM data**

- All VM memory is accessible to and controlled by the hypervisor
- Many VM escape CVEs are disclosed in recent years
  - CVE-2019-6778, CVE-2019-14835, CVE-2019-18389, CVE-2021-29657, etc.

- **Regulations have been established to enforce data security**

- E.g., General Data Protection Regulation (GDPR)



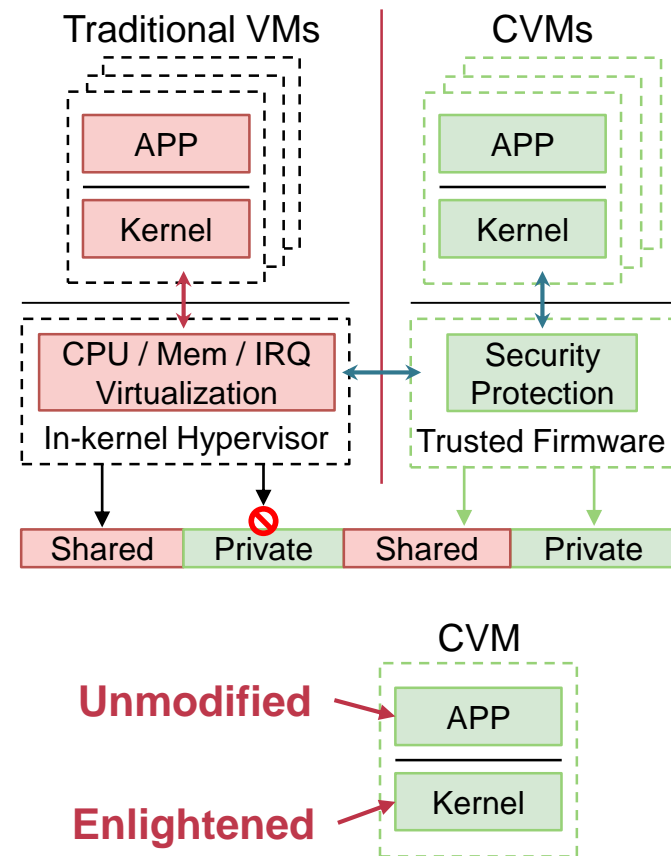
# CVM is Becoming Popular in Data Centers

- **Good security**

- OS-level confidential computing
  - Each guest OS is hardware-isolated from outside
- CPU states are protected during VM exits
- Memory isolation with hardware encryption
  - **Private type**: only accessible to CVM
  - **Shared type**: also accessible to hypervisor

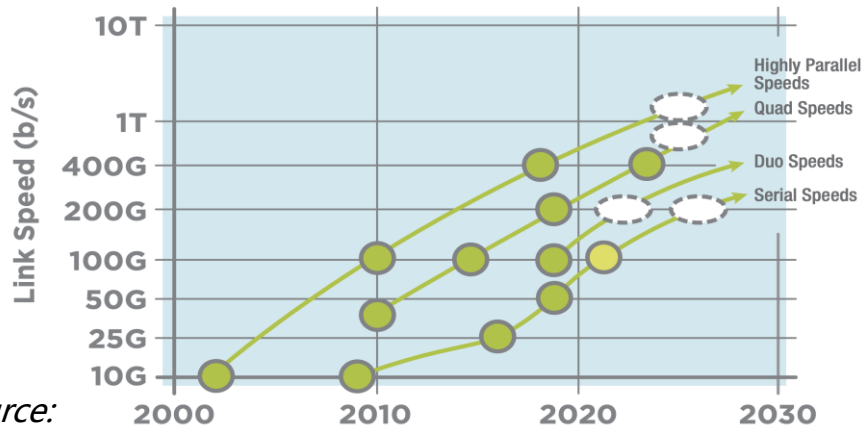
- **Good compatibility**

- Ease of integration with existing IaaS
- Transparent to application workloads



# Terabit Ethernet is Approaching

- **The speed of modern network devices continues to grow**
  - For example, NVIDIA ConnectX-7 400GbE SmartNIC
- **CPUs become performance bottleneck**
  - Both application logic and I/O processing consume significant CPU resources



Source:

[ethernetalliance.org](http://ethernetalliance.org)

Standard Completed

● Ethernet Speed ● Speed in Development ○ Possible Future Speed

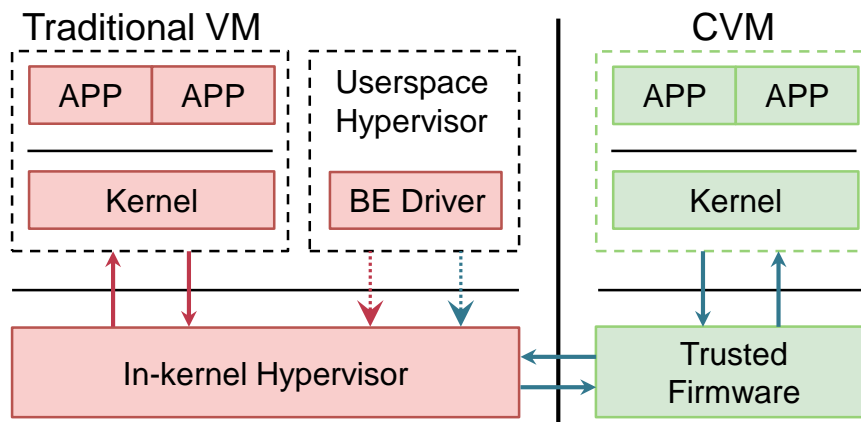
APP logic I/O processing



CPUs are fully loaded

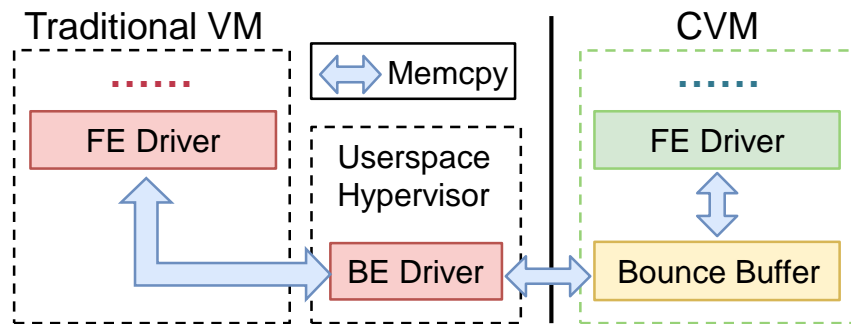
# Paravirtual I/O Networking in CVMs

- **A primary I/O virtualization choice for modern cloud providers**
  - Typical usage: a polling-based userspace I/O backend for high performance
- **I/O event notification**
  - Hypervisor notifies VMs via vIRQs, which may trigger VM exits
  - CVMs' exits have higher latency vs. traditional VMs



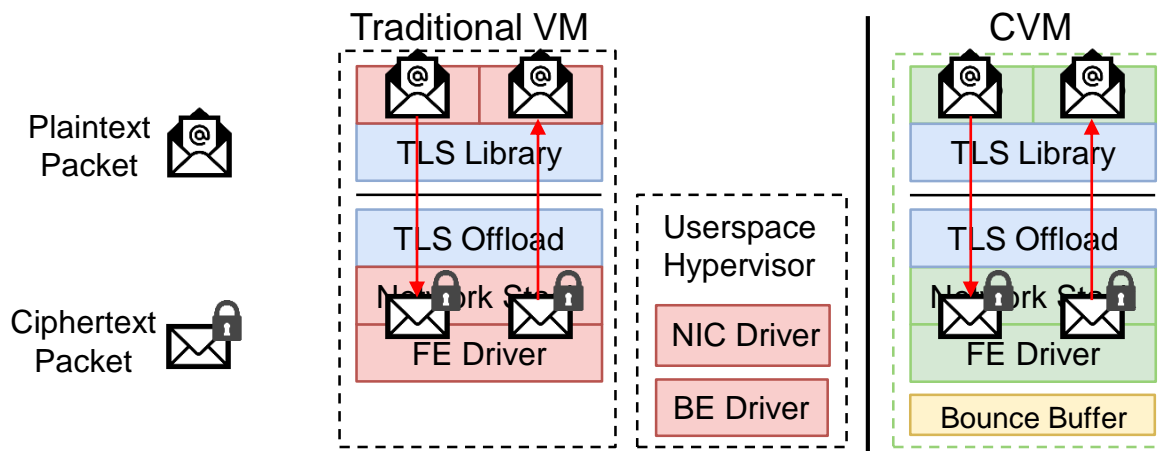
# Paravirtual I/O Networking in CVMs

- **A primary I/O virtualization choice for modern cloud providers**
  - Typical usage: a polling-based userspace I/O backend for high performance
- **I/O data transfer**
  - Hypervisor copies I/O data to/from VM memory
  - CVMs require **bounce buffer** because hypervisor cannot access private memory



# Paravirtual I/O Networking in CVMs

- **A primary I/O virtualization choice for modern cloud providers**
  - Typical usage: a polling-based userspace I/O backend for high performance
- **I/O data protection**
  - End-to-end encryption, such as transport layer security (TLS)
  - In-kernel TLS support for enhanced performance and extended features



# The Network Performance of CVM

- **End-to-end evaluation**
  - Baseline: traditional VMs
  - Benchmark: network-intensive applications
- **Testbed configuration**
  - **CVM**: AMD SEV-ES/SNP, w/o posted IRQ\*
  - **CVM+PI**: simulated Intel TDX, w/ posted IRQ
  - More details in the *Evaluation* part

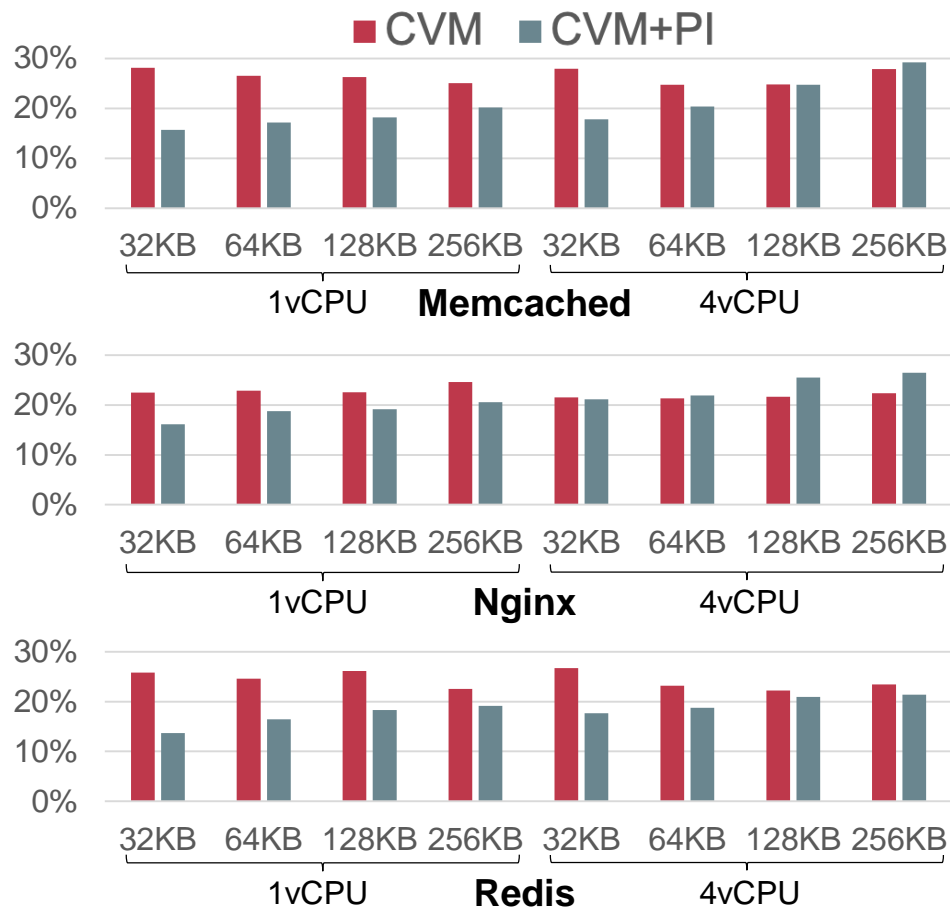
\*Posted IRQ: eliminate VM exits during vIRQ deliveries



# The Network Performance of CVM

- **End-to-end evaluation**
  - Baseline: traditional VMs
  - Benchmark: network-intensive applications
- **Testbed configuration**
  - **CVM**: AMD SEV-ES/SNP, w/o posted IRQ\*
  - **CVM+PI**: simulated Intel TDX, w/ posted IRQ
  - More details in the *Evaluation* part
- **Summary: poor network performance**
  - **CVM**: 21% - 28% overhead vs. baseline
  - **CVM+PI**: 13% - 29% overhead vs. baseline

\*Posted IRQ: eliminate VM exits during vIRQ deliveries



# CVM-IO Tax Greatly Impacts Performance

- **CVM-IO tax & application workloads share limited CPU execution time**
- **CVM-IO tax**
  - CPU time spent on *security protections* and *intrinsic network I/O procedures*
  - **More** CPU time → **Worse** performance
- **Application workloads**
  - CPU time spent on *business logic* and *payload processing*
  - **More** CPU time → **Better** performance

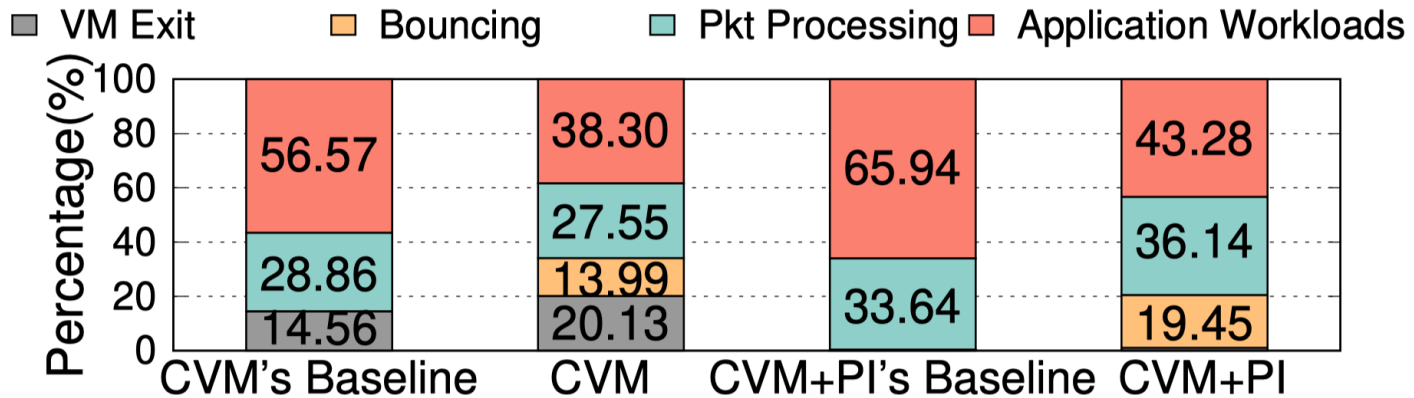
# CVM-IO Tax Greatly Impacts Performance

- CVM-IO tax & application workloads share limited CPU execution time
- CVM-IO tax
  - CPU time spent on *security protections* and *intrinsic network I/O procedures*
  - More CPU time → **Worse** performance
- Application workloads
  - CPU time spent on *business logic* and *payload processing*
  - More CPU time → **Better** performance

CVM-IO tax type	Factors that determine CPU time of each tax type
VM exits	The frequency of guest-host interactions
Bounce buffer	The size of I/O data transferred
Packet processing	The number of network packets processed

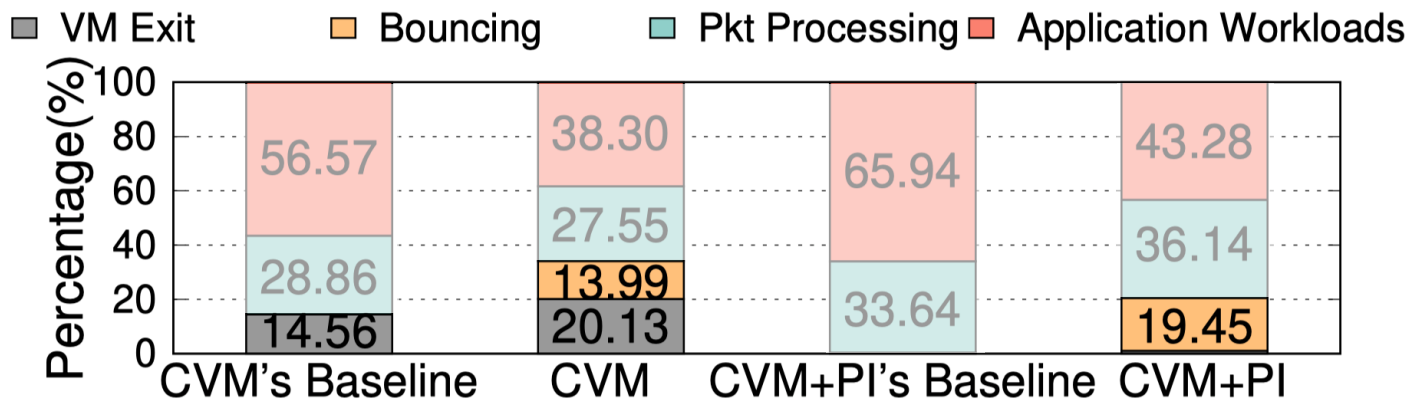
# CVM-IO Tax Breakdown

- **Example: a 4-thread Memcached server in a 4-vCPU VM**
  - *memtier\_benchmark*: 8 clients, 32 concurrent requests and 256KB data size
- **CVM-IO tax consumes > 50% CPU time**



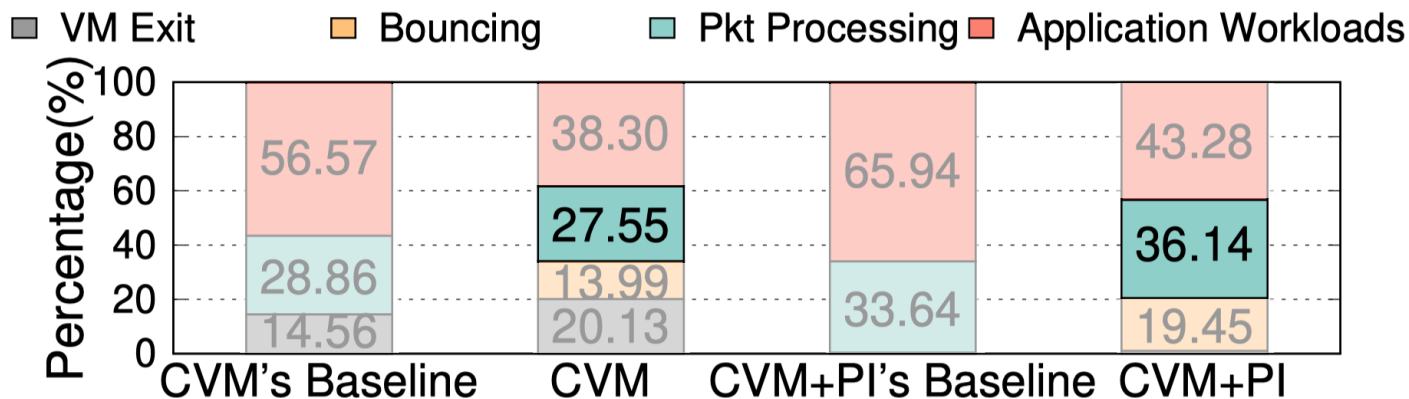
# CVM-IO Tax Breakdown

- **Example: a 4-thread Memcached server in a 4-vCPU VM**
  - *memtier\_benchmark*: 8 clients, 32 concurrent requests and 256KB data size
- **CVM-IO tax consumes > 50% CPU time**
  - Overhead vs. baseline: **VM exits** and **bounce buffer**



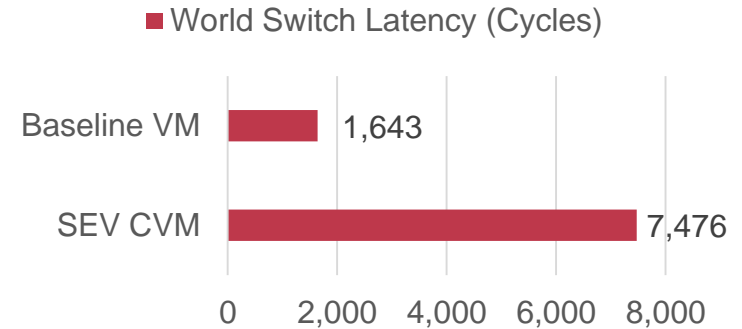
# CVM-IO Tax Breakdown

- **Example: a 4-thread Memcached server in a 4-vCPU VM**
  - *memtier\_benchmark*: 8 clients, 32 concurrent requests and 256KB data size
- **CVM-IO tax consumes > 50% CPU time**
  - Overhead vs. baseline: **VM exits** and **bounce buffer**
  - **Packet processing** consumes a large portion of CPU time



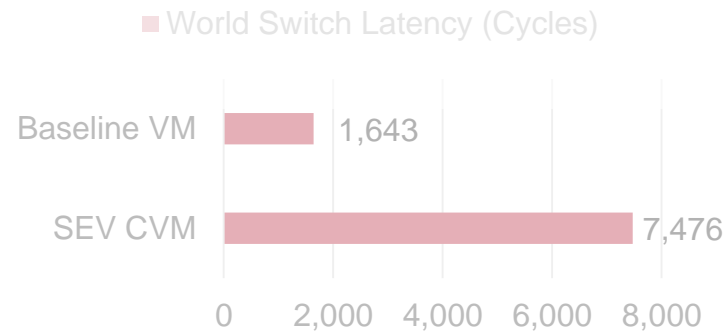
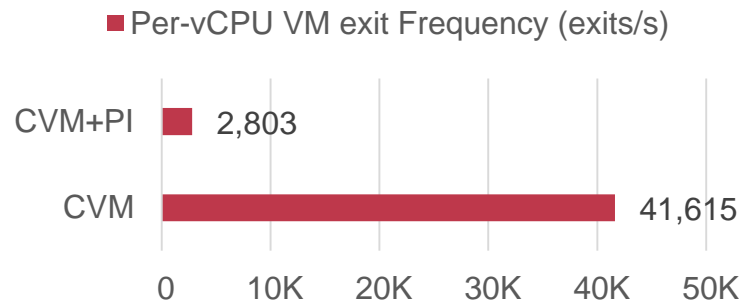
# Lengthy VM Exits

- **CVMs introduce protection on CPU states**
  - For instance, when a VM exit happen, the trusted firmware may save and clear registers
- **The protection add large CPU cycles to guest-host world switches**
  - **CVM** consumes 5,833 cycles more than its baseline



# Lengthy VM Exits

- **CVMs introduce protection on CPU states**
  - For instance, when a VM exit happen, the trusted firmware may save and clear registers
- **The protection add large CPU cycles to guest-host world switches**
  - CVM consumes 5,833 cycles more than its baseline
- **If without posted IRQ, vIRQ → frequent VM exits**
  - CVM triggers ~20x more VM exits/s than CVM+PI





# Lengthy VM Exits

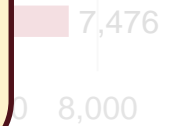
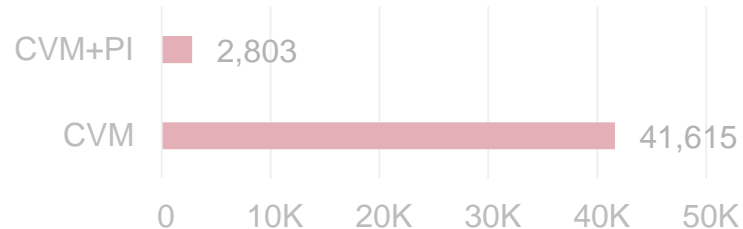
- CVMs introduce protection on CPU states

- **VM exits can take up a large portion of the CPU time of CVMs**

- Up to 20% of CVM's CPU cycles → Significant performance impact
- Increased latency & High frequency → Large overhead vs. baseline VMs

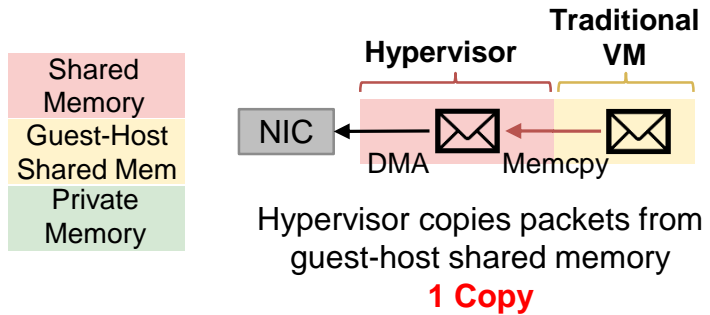
- If ➤ **Posted IRQ minimizes the performance impact of VM exits tax**

- At most 1% of CVM+PI's CPU cycles
- Will be supported soon on next-generation CVM hardware platforms



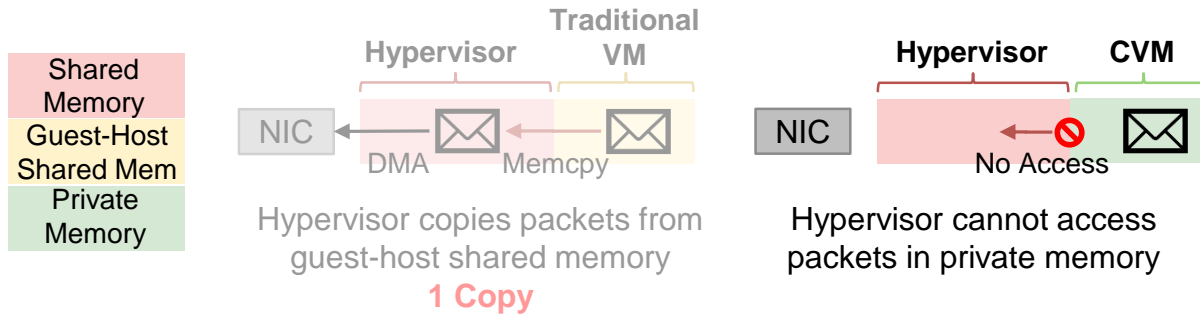
# Bounce Buffer

- Traditional VM memory is shared with the hypervisor



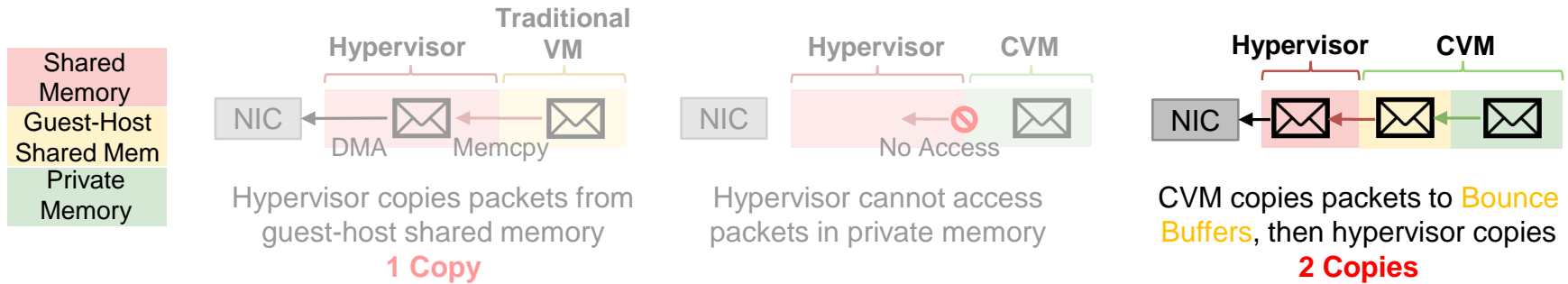
# Bounce Buffer

- CVM memory is set to private by default



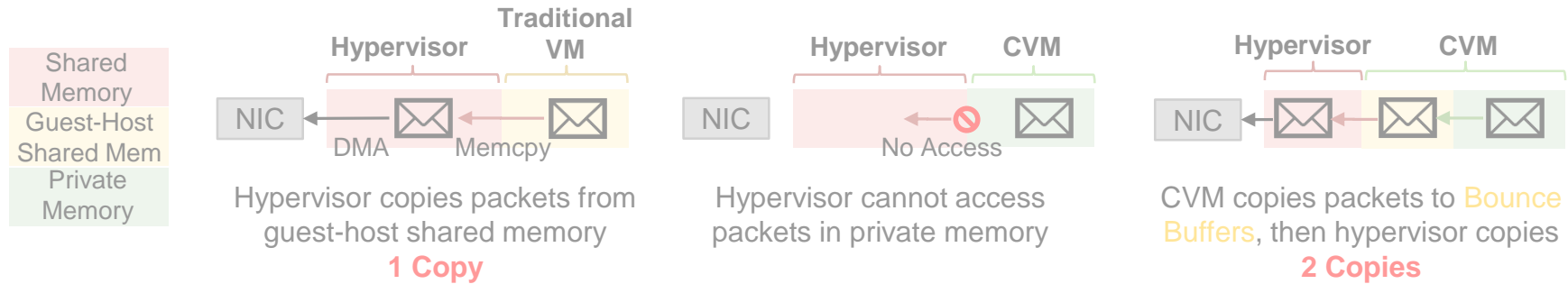
# Bounce Buffer

- CVM memory is set to private by default



# Bounce Buffer

- CVM memory is set to private by default



- **1 more copy → Larger I/O data size leads to more CPU time cost**

# Bounce Buffer

- CVM memory is set to private by default

Traditional

VM

Envelope icon

Bounce copies

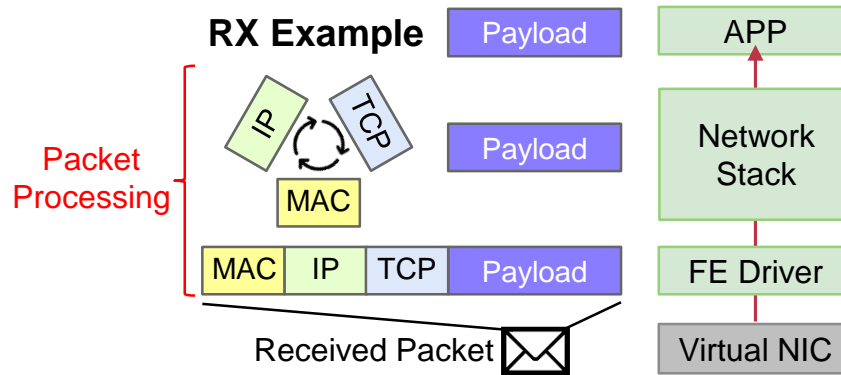
➤ **Bounce buffers can consume a large portion of the CPU time of CVMs**

- Up to 20% of CVM+PI's CPU cycles → Significant performance impact
- Both copying I/O data & maintaining metadata are costly
- CVM-specific & Large data size → Large overhead vs. baseline VMs

- 1 ➤ **For performance, it is necessary to avoid bouncing large-size I/O data**

# Packet Processing

- Comprise operations such as header parsing, encap and decap

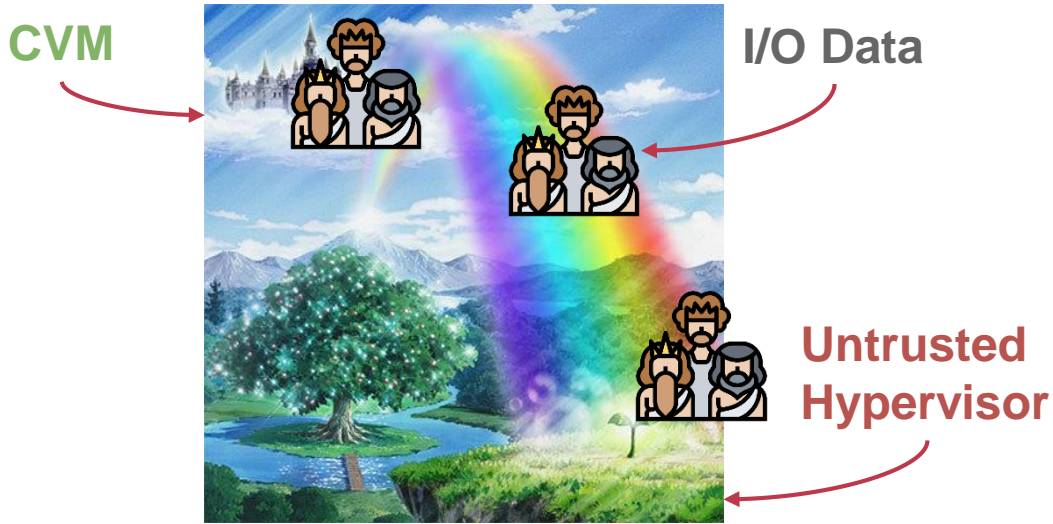


- Process each packet → more packets leads to more CPU time

# Packet Processing

- Comprise operations such as header parsing, encap and decap
  - **Packet processing consumes a large fraction of the CPU time of CVMs**
    - Up to 36% of CVM+PI's CPU cycles → Significant performance impact
    - There are a large number packets in network-intensive scenarios
  - **Reducing the number of packets can mitigate its performance impact**
- Process each packet → more packets leads to more CPU time





# Our Design: Bifrost



CPU and firmware



Side-channel  
Denial-of-Service

**ASSUMPTION**

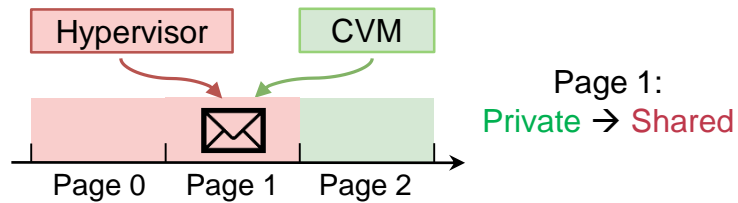
CVMs protect I/O data with end-to-end encryption, and do not voluntarily leak data

# Design Goals

- **Performance**
  - **Bounce buffer tax**: avoid bouncing large-size I/O data
  - **Packet processing tax**: reduce the number of packets to be processed
- **Security**
  - Maintain the same level of security guarantees as existing CVMs
- **Universality**
  - Applicable to diverse platforms (e.g., x86, ARM), guest/host OSes (e.g., Linux, FreeBSD)
- **Practicality**
  - Transparent to applications & Non-intrusive and minor modifications

# Challenges

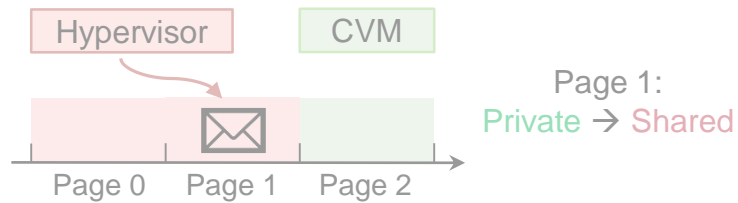
## C1. Out-of-place hardware memory encryption and decryption



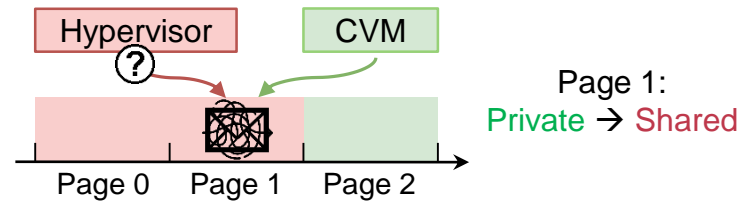
**Ideal:** eliminate bounce buffer via **zero copy**  
by keeping data in the same page

# Challenges

## C1. Out-of-place hardware memory encryption and decryption



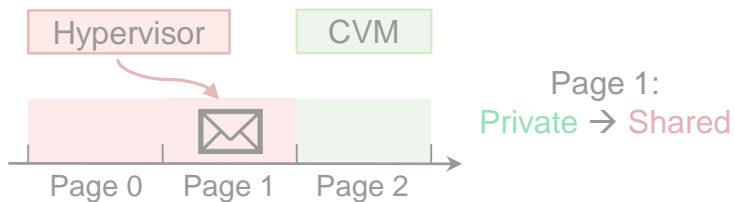
**Ideal:** eliminate bounce buffer via **zero copy**  
by keeping data in the same page



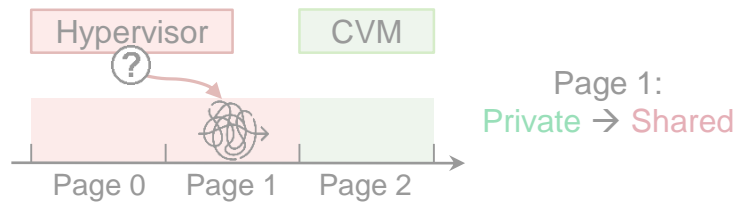
**Reality:** **out-of-place** hardware encryption / decryption  
→ data loss after memory type conversion

# Challenges

## C1. Out-of-place hardware memory encryption and decryption

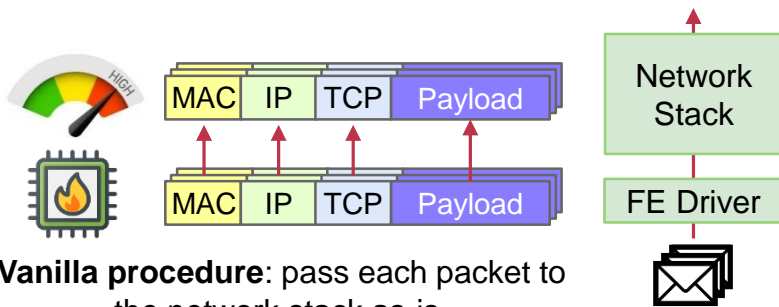


**Ideal:** eliminate bounce buffer via **zero copy** by keeping data in the same page



**Reality:** **out-of-place** hardware encryption / decryption → data loss after memory type conversion

## C2.

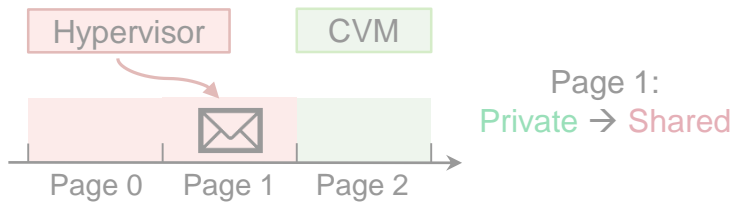


**Vanilla procedure:** pass each packet to the network stack as-is.

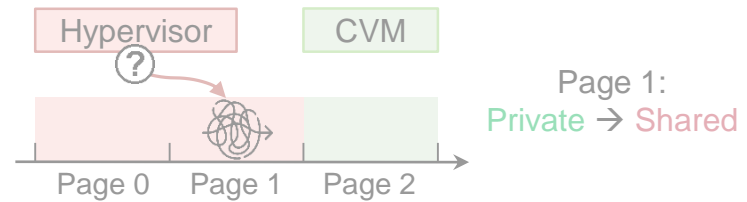
**Too many** packets to be processed.

# Challenges

## C1. Out-of-place hardware memory encryption and decryption

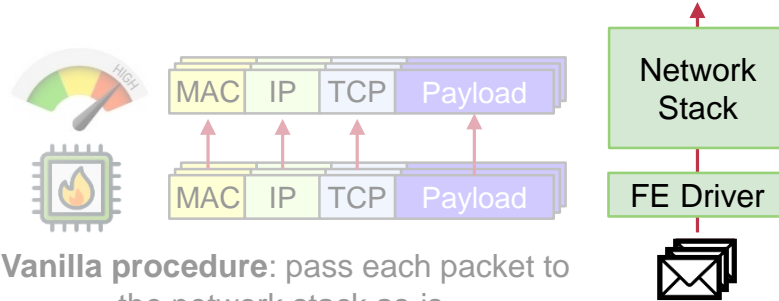


**Ideal:** eliminate bounce buffer via **zero copy** by keeping data in the same page

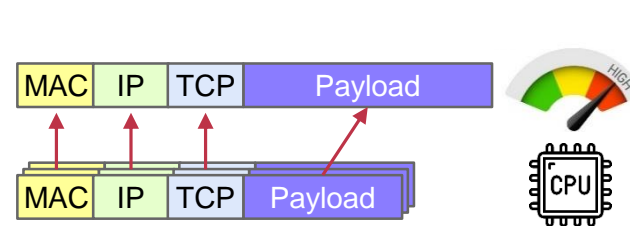


**Reality:** **out-of-place** hardware encryption / decryption → data loss after memory type conversion

## C2. Costly packet pre-processing in the device driver



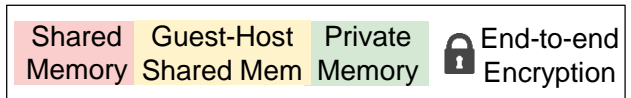
**Vanilla procedure:** pass each packet to the network stack as-is.  
**Too many** packets to be processed.



**Existing optimizations (e.g., GRO):** reassemble packets before the network stack.  
**Reduced** packets to be processed.

# Observations and Insights

## O1. Either end-to-end encryption or private memory is sufficient to assure data security



Only end-to-end encryption. **Secure**

The icon consists of a pink box, a yellow box, and a green box, with a black padlock icon over the yellow box and a shield with a checkmark to the right.

Only private memory protection. **Secure**

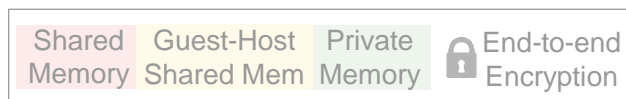
The icon consists of a green box with a black envelope icon containing an '@' symbol and a shield with a checkmark to the right.

Both protections. **Secure, but Redundant**

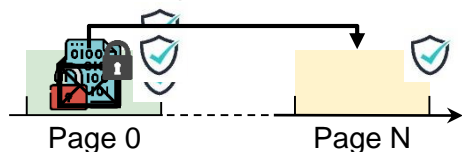
The icon consists of a green box with a black padlock icon over an envelope icon, followed by two shields with checkmarks.

# Observations and Insights

O1. Either private memory or end-to-end encryption is sufficient to assure data security



O2. End-to-end encryption has the side effect of moving the payload location

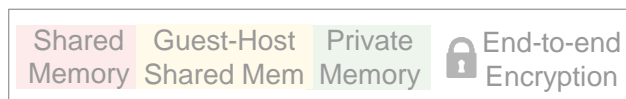


**Typical usage:** in-place encryption then bouncing, and bouncing then in-place decryption

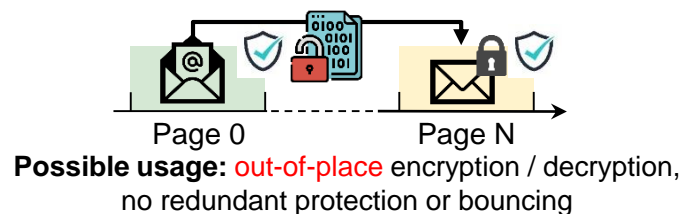
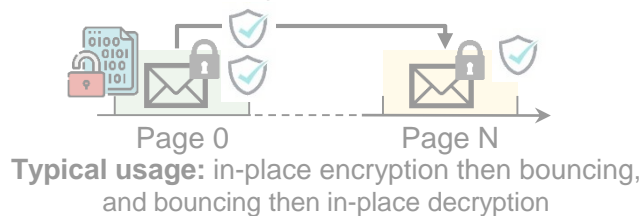


# Observations and Insights

## O1. Either private memory or end-to-end encryption is sufficient to assure data security

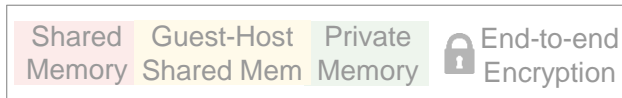


## O2. End-to-end encryption has the side effect of moving the payload location

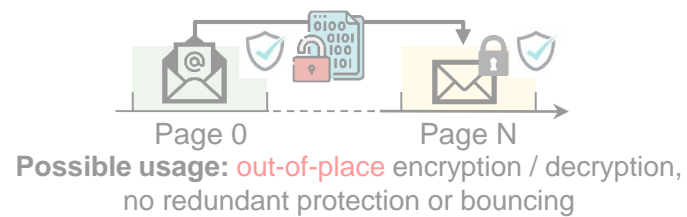
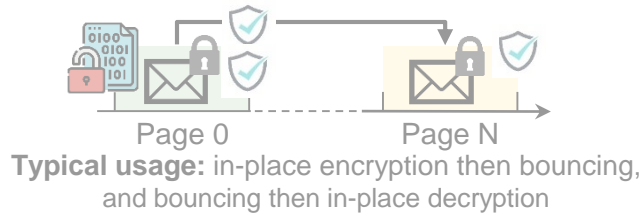


# Observations and Insights

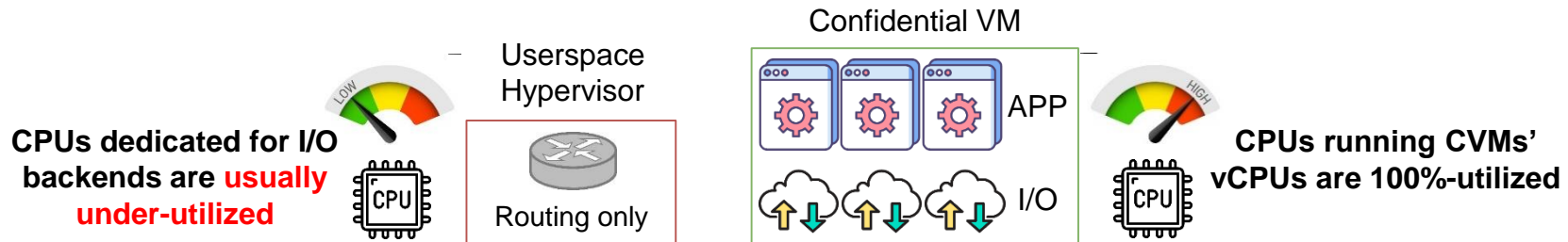
O1. Either private memory or end-to-end encryption is sufficient to assure data security



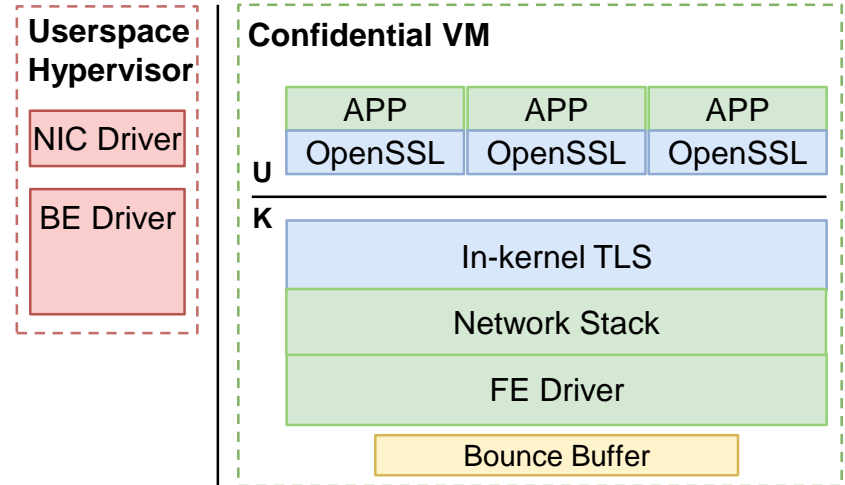
O2. End-to-end encryption has the side effect of moving the payload location



O3. I/O backends usually have plenty of residual CPU resources available





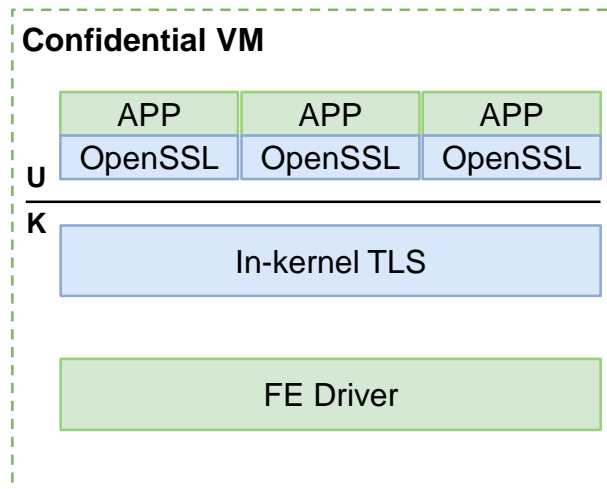
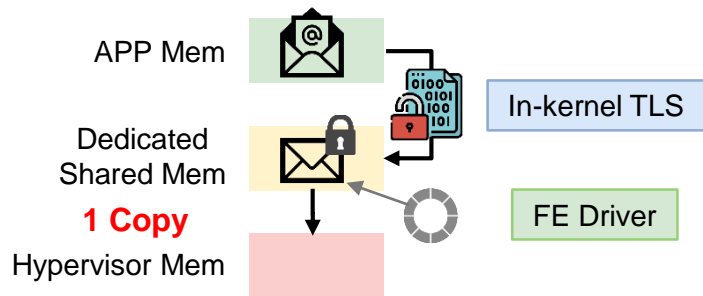
# Architecture and High-Level Design



# Architecture and High-Level Design


## D1. Zero-copy encryption deduplication (ZCED)

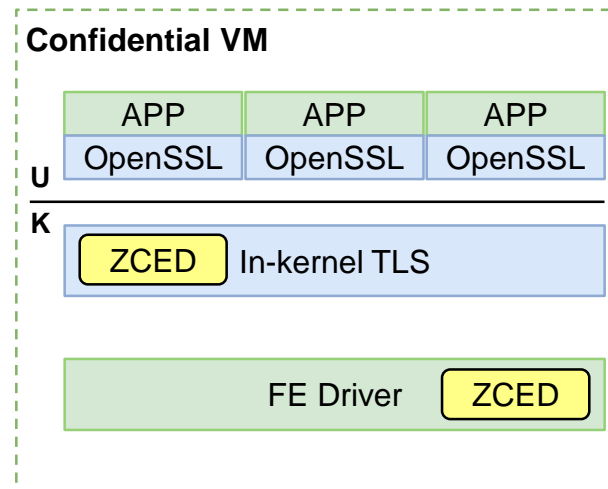
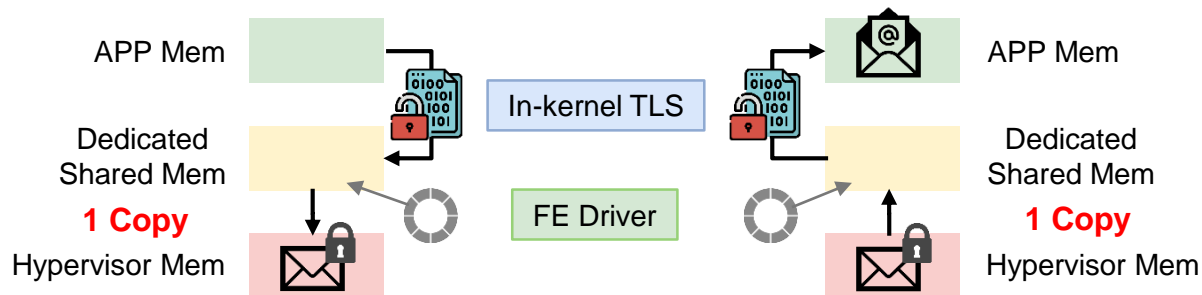
- Eliminate payload bouncing tax by leveraging O1&O2
  - APP memory   → guest-host shared memory



# Architecture and High-Level Design


## D1. Zero-copy encryption deduplication (ZCED)

- Eliminate payload bouncing tax by leveraging O1&O2
  - APP memory  $\leftarrow$    $\rightarrow$  guest-host shared memory
- Minimize modification & reuse CVM allocators
  - Dedicated NUMA nodes (called **ZCED NUMA**)



# Architecture and High-Level Design

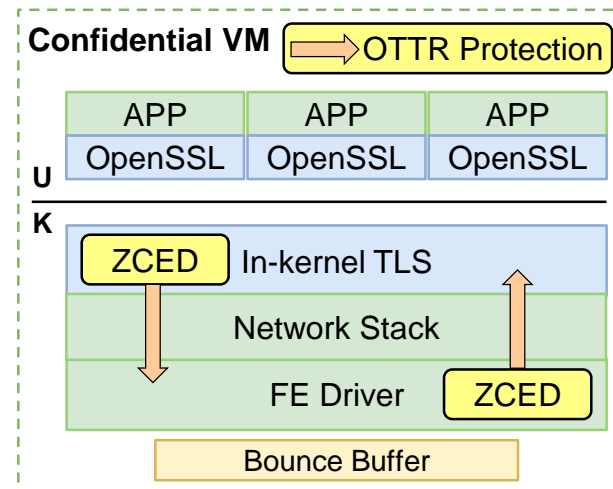
## D1. Zero-copy encryption deduplication (ZCED)

- Eliminate payload bouncing tax by leveraging O1&O2
  - APP memory  guest-host shared memory
- Minimize modification & reuse CVM allocators
  - Dedicated NUMA nodes (called **ZCED NUMA**)

## D2. One-time trusted read (OTTR)


- Defend against TOCTTOU attacks on ZCED NUMA
  - Principle: only trust the 1<sup>st</sup> read from the shared memory
  - Enforce protections in both TX and RX directions

\*Please refer to our paper for more details about OTTR



# Architecture and High-Level Design

## D1. Zero-copy encryption deduplication (ZCED)

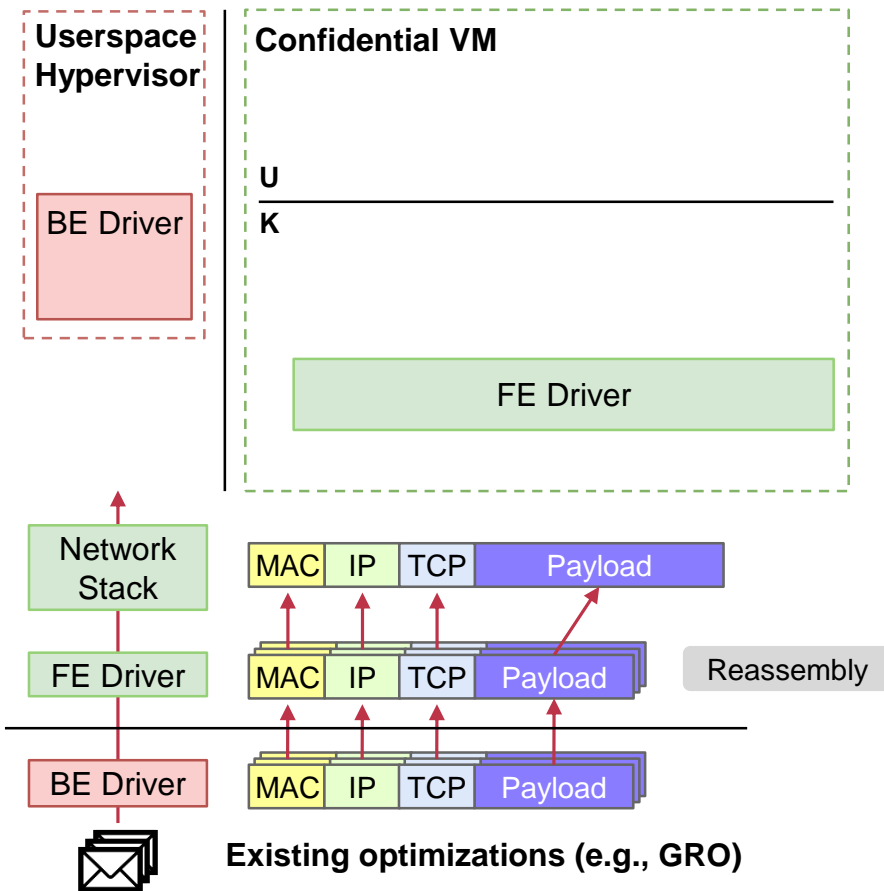
- Eliminate payload bouncing tax by leveraging O1&O2
  - APP memory  → guest-host shared memory
- Minimize modification & reuse CVM allocators
  - Dedicated NUMA nodes (called **ZCED NUMA**)

## D2. One-time trusted read (OTTR)

- Defend against TOCTTOU attacks on ZCED NUMA
  - Only trust the 1<sup>st</sup> read from the shared memory


## D3. Pre-receiver packet reassembly (PRPR)

- Reduce packet processing tax by leveraging O3
  - Offload reassembly to the network I/O backend



# Architecture and High-Level Design

## D1. Zero-copy encryption deduplication (ZCED)

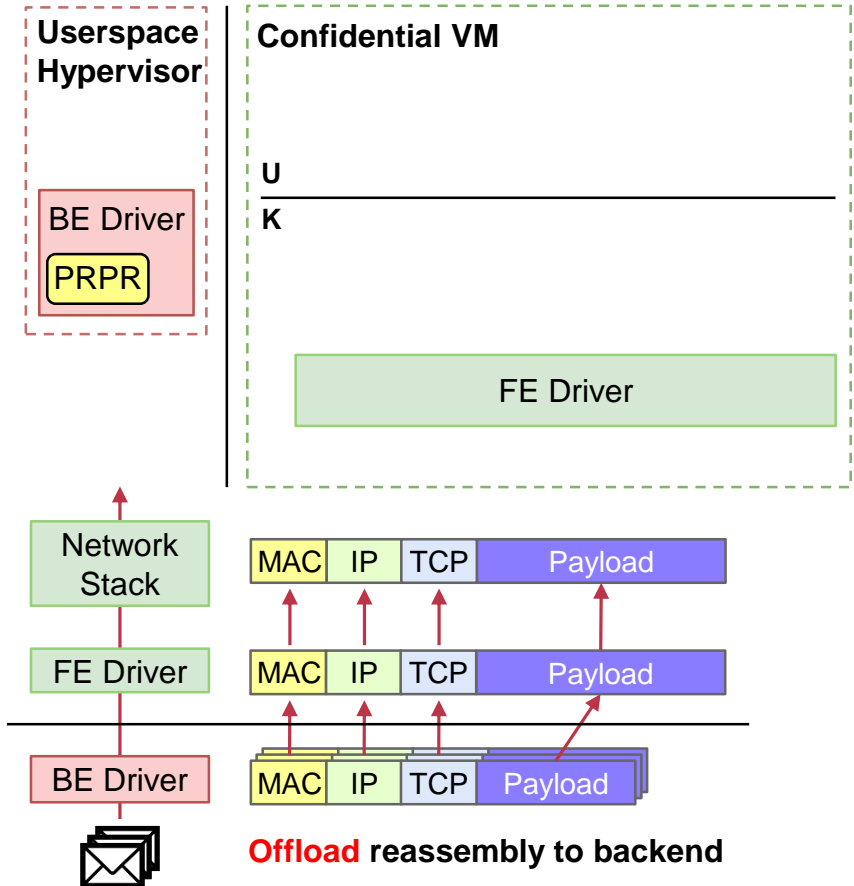
- Eliminate payload bouncing tax by leveraging O1&O2
  - APP memory  → guest-host shared memory
- Minimize modification & reuse CVM allocators
  - Dedicated NUMA nodes (called **ZCED NUMA**)

## D2. One-time trusted read (OTTR)

- Defend against TOCTTOU attacks on ZCED NUMA
  - Only trust the 1<sup>st</sup> read from the shared memory

## D3. Pre-receiver packet reassembly (PRPR)

- Reduce packet processing tax by leveraging O3
  - Offload reassembly to the network I/O backend

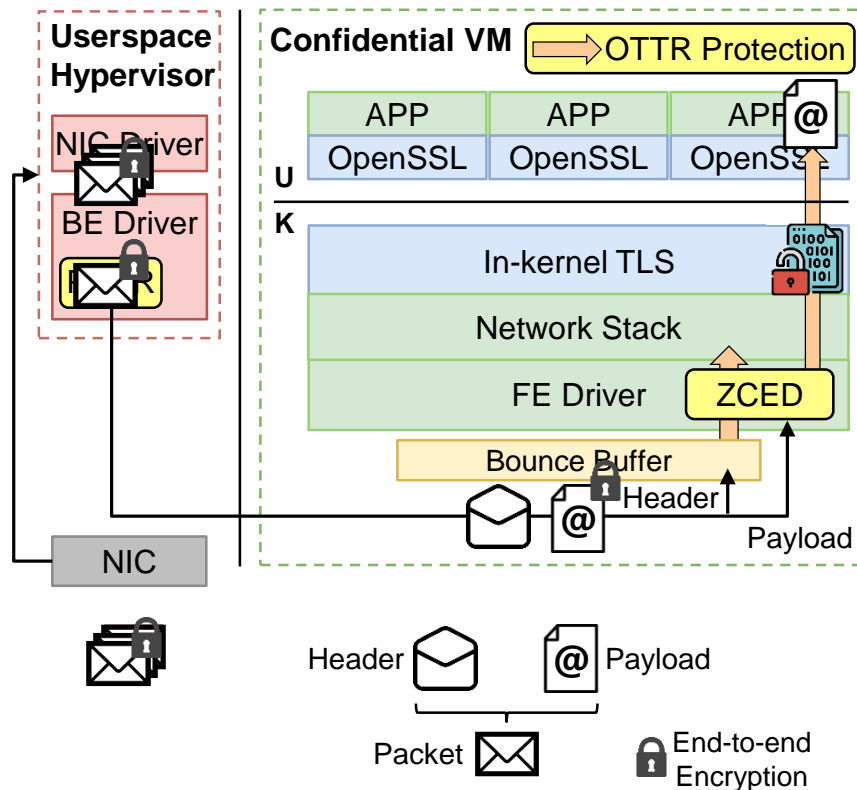




# Packet Receiving Workflow

Incoming packets arrive at the backend driver

1. **PRPR** reassembles same-flow packets
2. The backend driver flushes the packet to CVM
  - Encrypted payload: kept in ZCED NUMA memory
  - Header: bounced to private memory by **OTTR**
3. The TLS layer decrypts payload for APPs
  - Directly from **ZCED** NUMA memory to APP memory
  - **OTTR** eliminates TOCTTOU issues during decryption



# Packet Sending Workflow

APPs start to send out payload for network I/O

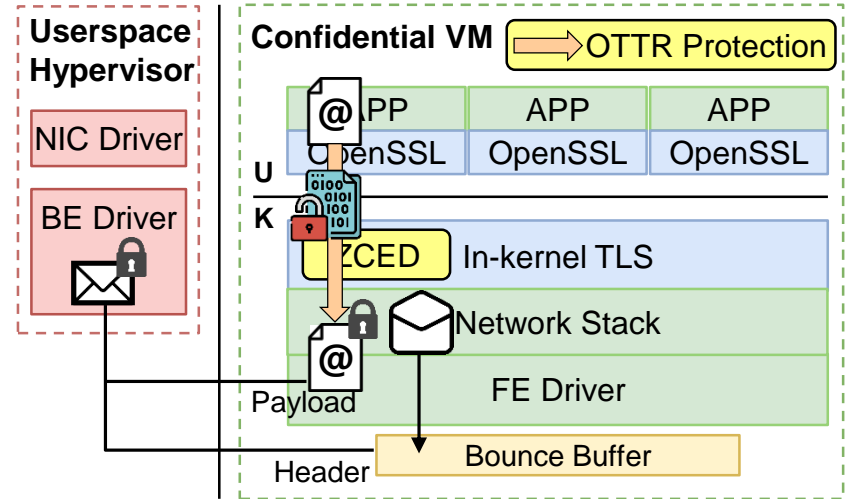
## 1. The TLS layer encrypts payload

- Directly from APP memory to **ZCED** NUMA memory
- **OTTR** eliminates TOCTTOU issues during encryption

## 2. The CVM prepares packets for the backend driver

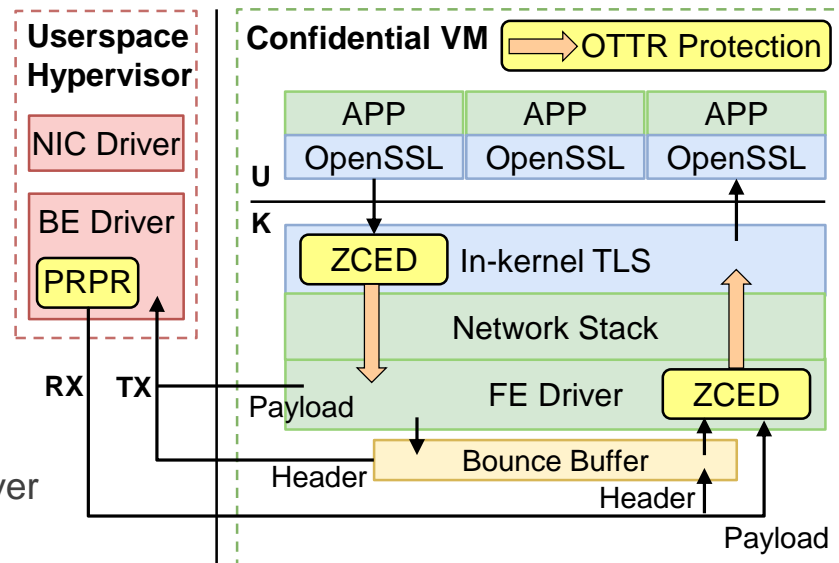
- Encrypted payload: kept in shared memory
- Header: built in private mem, bounced to shared mem

## 3. The backend driver obtains packets from shared memory



# Bifrost Summary

- **VM exits tax**
  - ✓ Solved by posted interrupt on next-gen CVM hardware
- **Bounce buffer tax**
  - Challenge: out-of-place hardware memory encryption
  - ✓ **Zero-copy encryption deduplication (ZCED)**
    - Eliminate payload bouncing
    - TOCTTOU defense: **One-time trusted read (OTTR)**
- **Packet processing tax**
  - Challenge: costly packet pre-processing in the device driver
  - ✓ **Pre-receiver packet reassembly (PRPR)**
    - Offload packet reassembly to network I/O backend



\*Please refer to our paper for more details



# Prototype and Evaluation

# Prototype and Testbed Setup

Component	LoC
Guest Linux v6.0-rc1	815
Open vSwitch v2.17.3	175
DPDK v21.11.2	541

## Implementation Complexity

Naming	Description
<b>CVM+RIF</b>	<b>CVM</b> + Reduced IRQ Frequency
<b>+ZC</b>	Only apply ZCED & OTTR
<b>+PRPR</b>	Only apply PRPR

## Additional Naming Convention

Component	Configuration
CPU	<b>CVM+RIF</b> : 2x AMD EPYC 7T83, 128 cores <b>CVM+PI</b> : 2x Intel Xeon Gold 5317, 24 cores
DDR4 DRAM	AMD 500GB; Intel 188GB
NIC	NVIDIA Connect-X6 200Gbps, back-to-back
Host OS	Ubuntu 20.04.4 LTS <b>CVM+RIF</b> : Linux v5.19.0-rc6 (SEV enabled) <b>CVM+PI</b> : Linux v5.4.0
Network Backend	Open vSwitch v2.17.3 DPDK v21.11.2
Guest OS	Ubuntu 20.04.4 LTS Linux v6.0-rc1 1 or 4 vCPUs, 16GB memory, 2 virtqueues

## Testbed Configuration

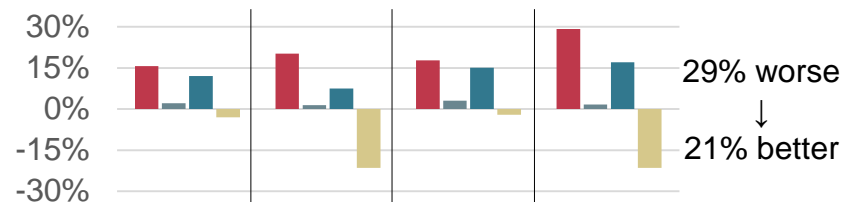
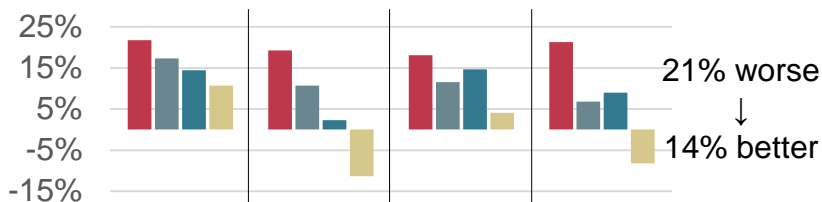
\*Please refer to our paper for more details

# Application Benchmarks

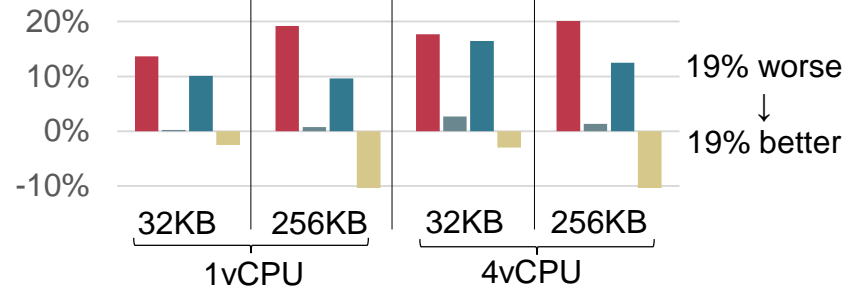
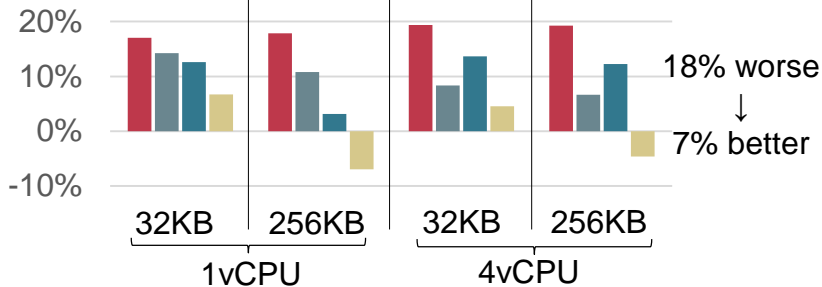
Y-axis: Overhead vs. traditional VMs  
Lower is better, negative → improvement

■ Vanilla ■ +ZC ■ +PRPR ■ Bifrost

Memcached



Redis



CVM+RIF

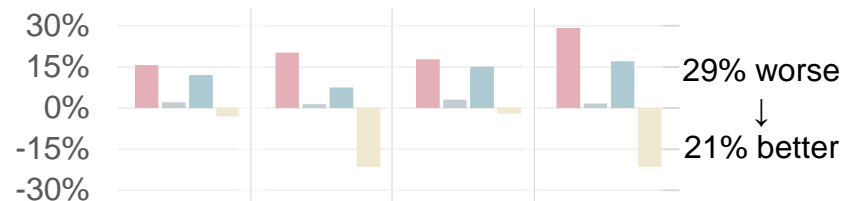
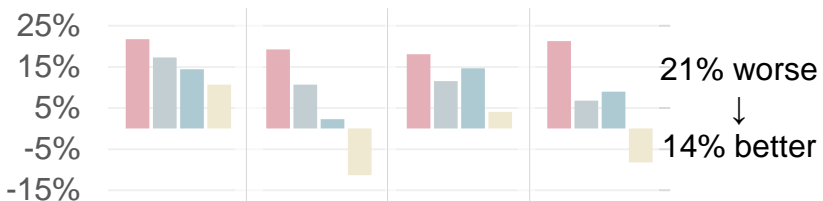
CVM+PI

# Application Benchmarks

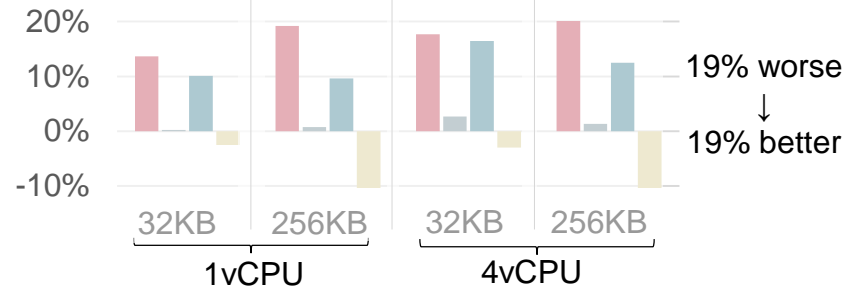
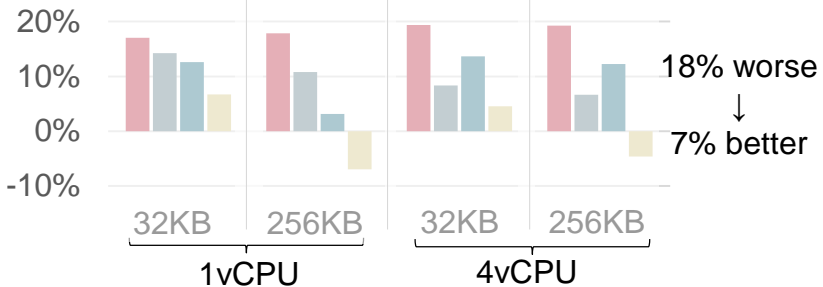
Y-axis: Overhead vs. traditional VMs  
Lower is better, negative → improvement

■ Vanilla ■ +ZC ■ +PRPR ■ Bifrost

**Memcached**  
Improvement increases  
as the data size grows



**Redis**



**CVM+RIF**

**CVM+PI**

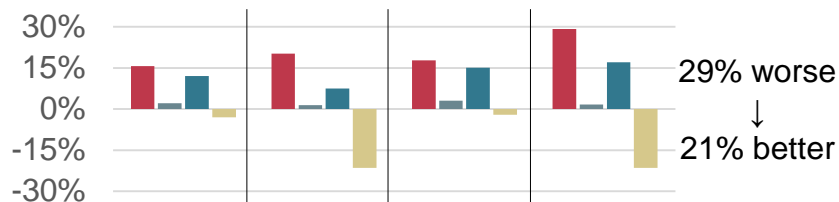
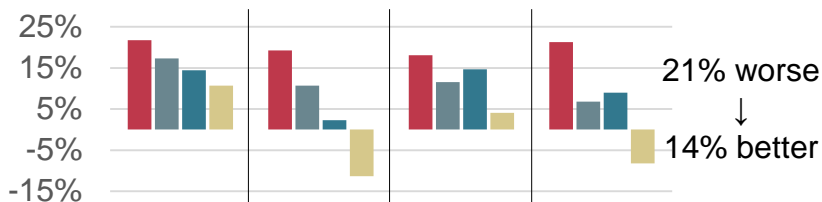
# Application Benchmarks

Y-axis: Overhead vs. traditional VMs  
Lower is better, negative → improvement

■ Vanilla ■ +ZC ■ +PRPR ■ Bifrost

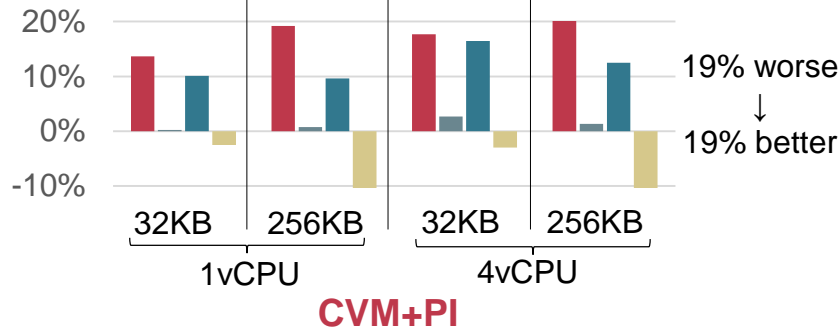
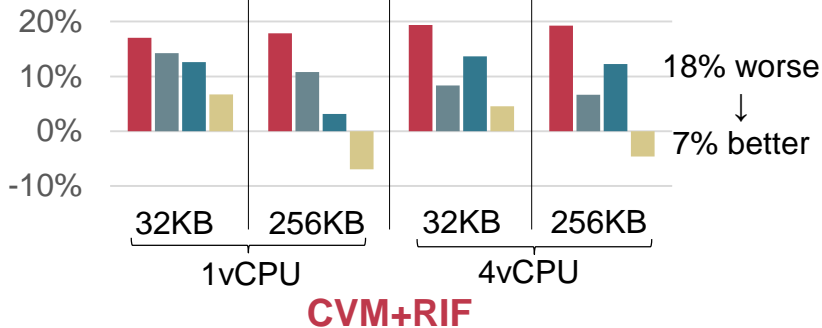
## Memcached

Improvement increases as the data size grows



## Redis

CVM+RIF w/o PI still have much VM exits tax → Smaller improvement





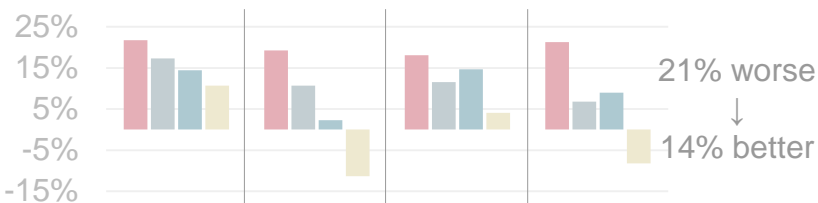
# Application Benchmarks

Y-axis: Overhead vs. traditional VMs  
Lower is better, negative → improvement

■ Vanilla ■ +ZC ■ +PRPR ■ Bifrost

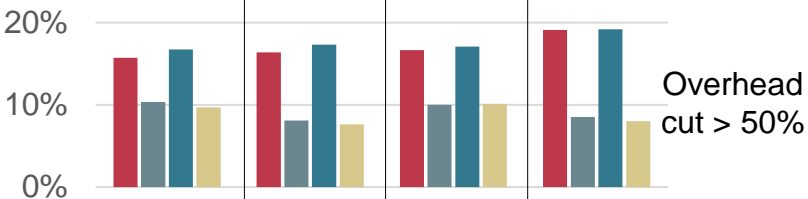
## Memcached

Improvement increases as the data size grows



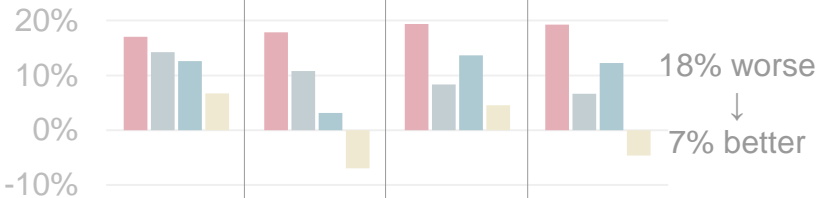
## Nginx

TX traffic is dominant  
PRPR has little effect



## Redis

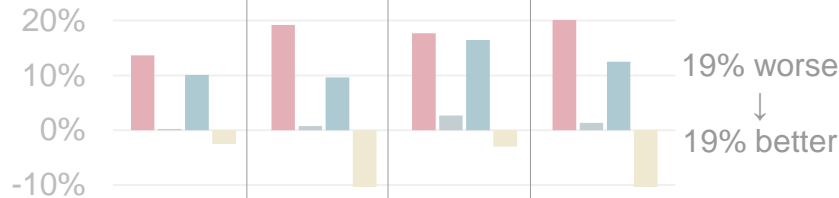
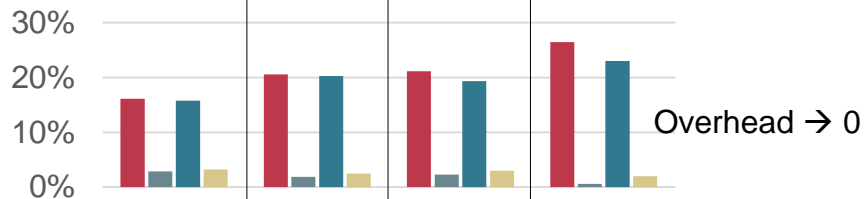
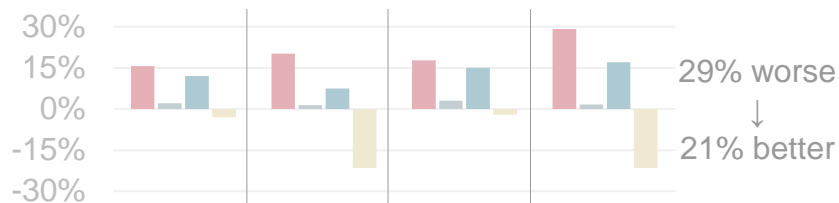
CVM+RIF w/o PI still have much VM exits tax  
→ Smaller improvement



1vCPU

4vCPU

CVM+RIF



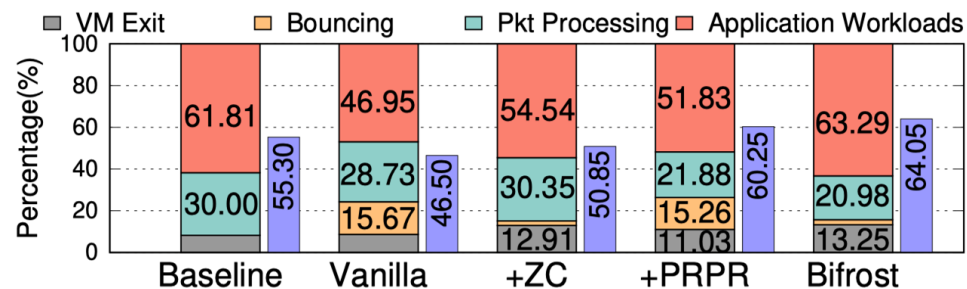
1vCPU

4vCPU

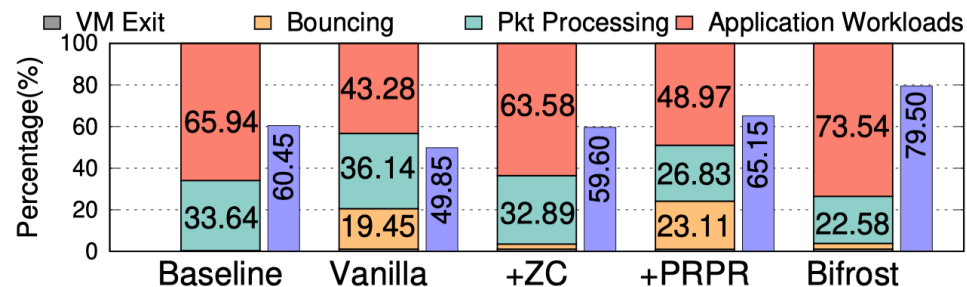
CVM+PI

# CPU Utilization Breakdown

- To explain the aforementioned performance improvements
- Breakdown examples
  - Memcached experiments
  - 4-vCPU VM and 256KB data size



(a) Breakdown of *CVM+RIF* and its baseline & Backend CPU utilization

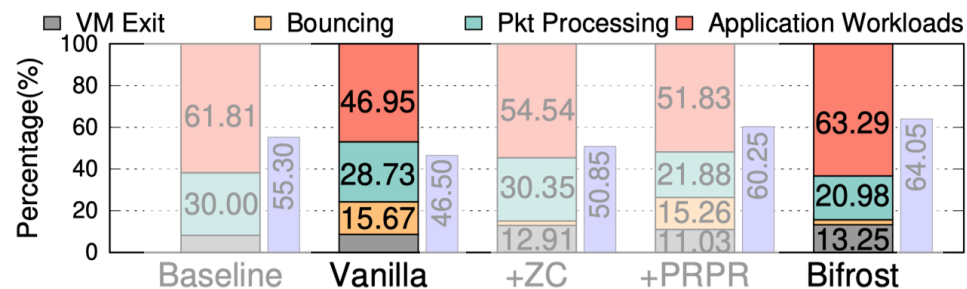


(b) Breakdown of *CVM+PI* and its baseline & Backend CPU utilization

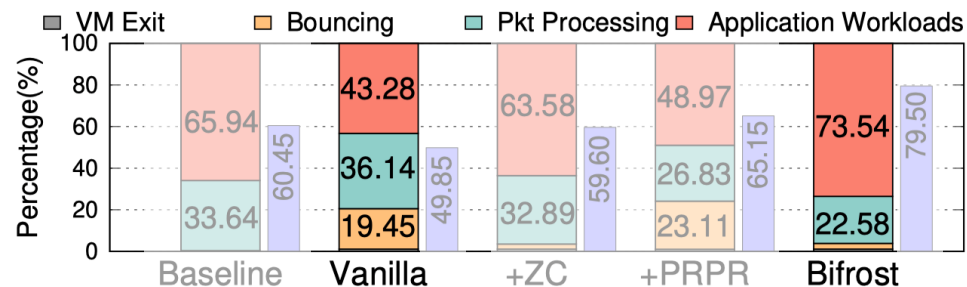
# CPU Utilization Breakdown

- **Where does the improvement come from?**

- Application workloads CPU time: 43% → 74%
- **Bounce buffer tax**: 23% → 2%
- **Packet processing tax**: 36% → 22%



(a) Breakdown of *CVM+RIF* and its baseline & Backend CPU utilization

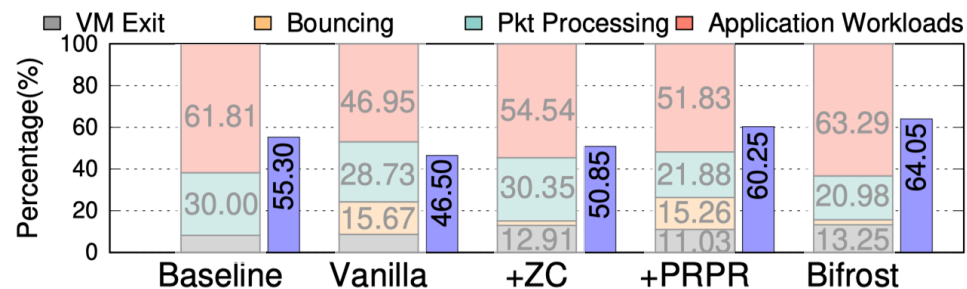


(b) Breakdown of *CVM+PI* and its baseline & Backend CPU utilization

# CPU Utilization Breakdown

- **Where does the improvement come from?**

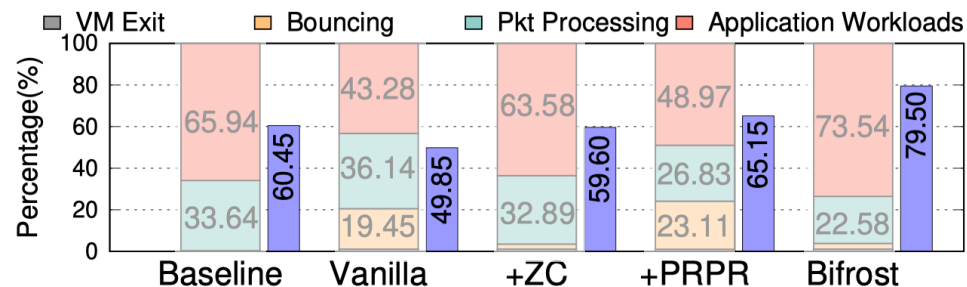
- Application workloads CPU time: 43% → 74%
- **Bounce buffer tax**: 23% → 2%
- **Packet processing tax**: 36% → 22%



(a) Breakdown of *CVM+RIF* and its baseline & Backend CPU utilization

- **How is backend CPU utilization impacted?**

- Spend at most 19% more CPU time than the baseline (i.e., traditional VM)
- Still not fully loaded, no negative impact on backend processing



(b) Breakdown of *CVM+PI* and its baseline & Backend CPU utilization

\*Please refer to our paper for more details

# Conclusion



- The **1<sup>st</sup>** systematic CVM-I/O tax analysis for network-intensive CVMs
- A new paravirtual I/O design: **Bifrost**
  - Eliminate redundant bounces for network packets (*Design 1: ZCED*)
  - Maintain the same level of security guarantees as existing CVMs (*Design 2: OTTR*)
  - Greatly reduce packet processing cost in CVMs (*Design 3: PRPR*)
- **Significantly improve the network I/O performance of CVMs**
  - Outperform traditional VMs by **up to 21.50%**



Bifrost prototype available:

<https://github.com/IPADS-Bifrost>

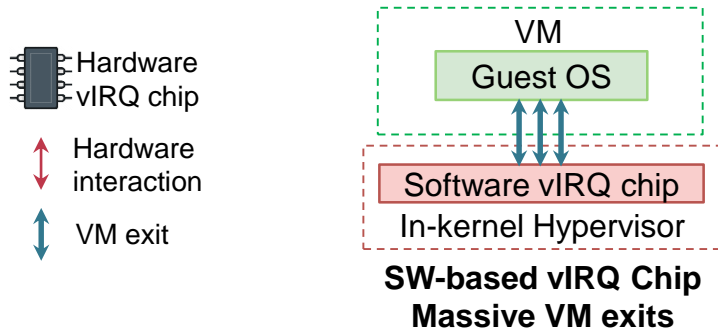
Thanks!



# Backup Slides

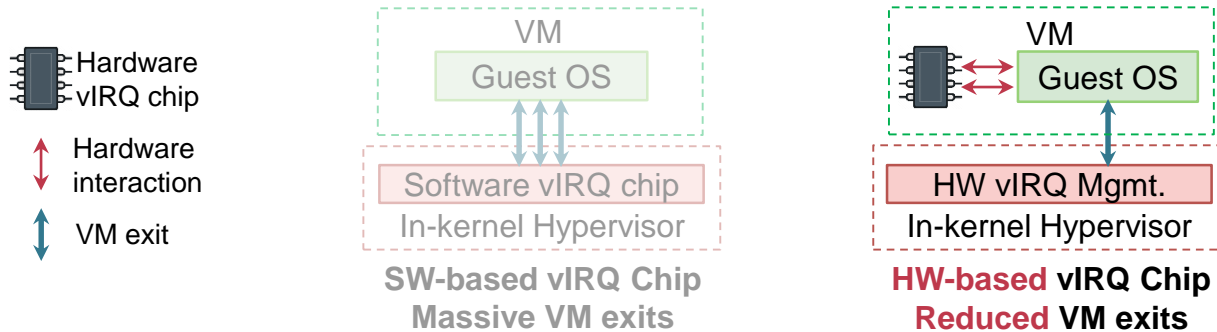
# Frequent VM Exits without Posted IRQ

- I/O-intensive workloads → frequent vIRQ due to I/O notifications



# Frequent VM Exits without Posted IRQ

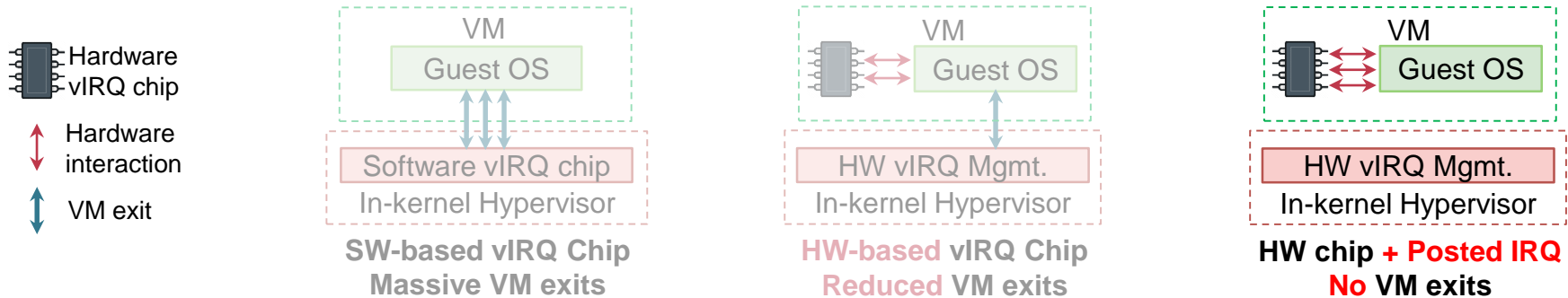
- I/O-intensive workloads → frequent vIRQ due to I/O notifications





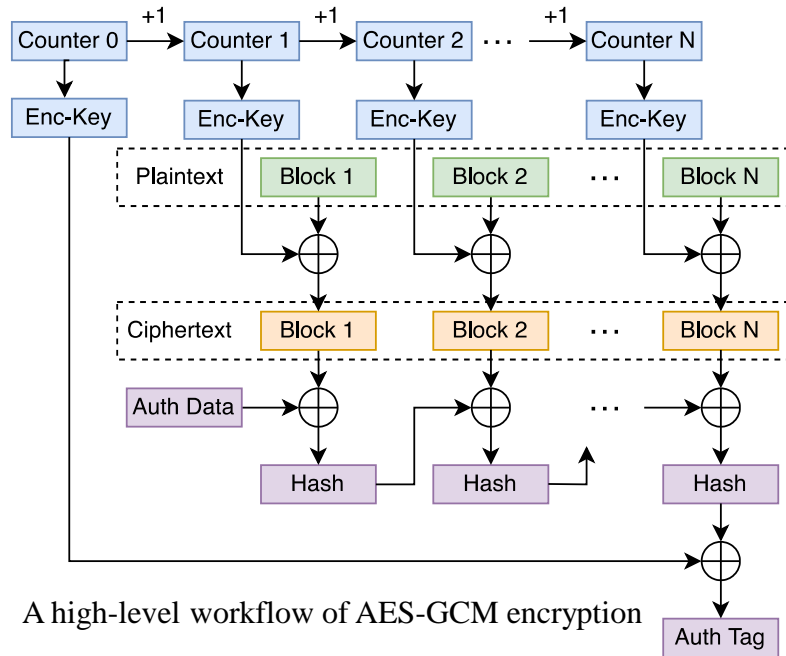
# Frequent VM Exits without Posted IRQ

- I/O-intensive workloads → frequent vIRQ due to I/O notifications



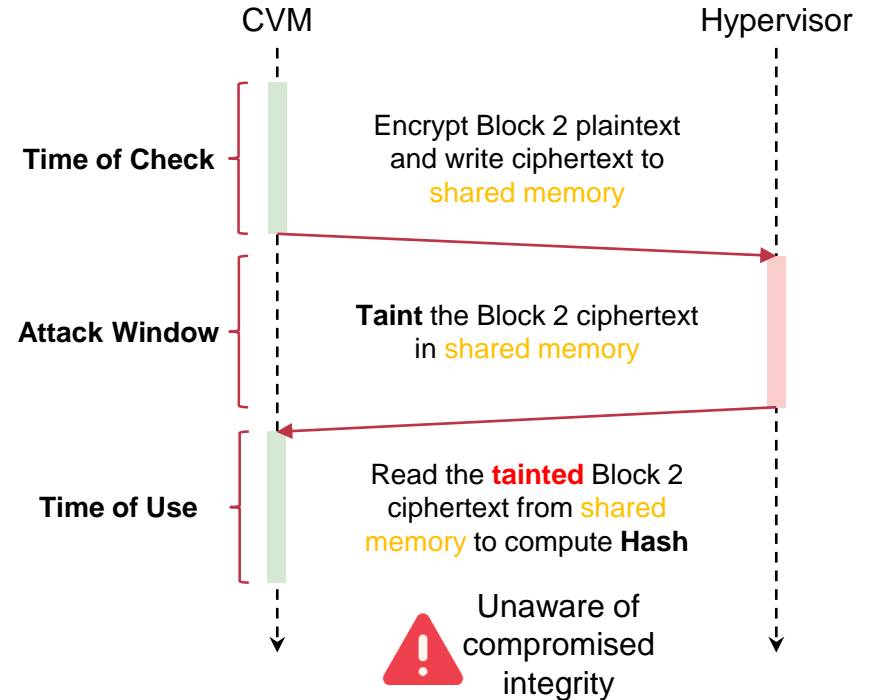
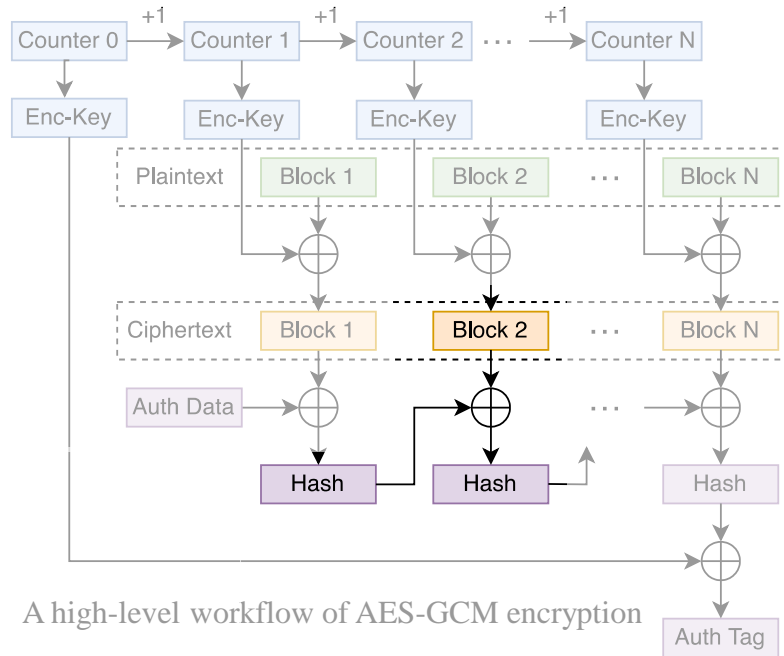
# A TOCTTOU Example in AES-GCM

- For integrity, an Auth Tag is computed iteratively based on each block of ciphertext
- The correctness of the Auth Tag depends on the correctness of each block of ciphertext



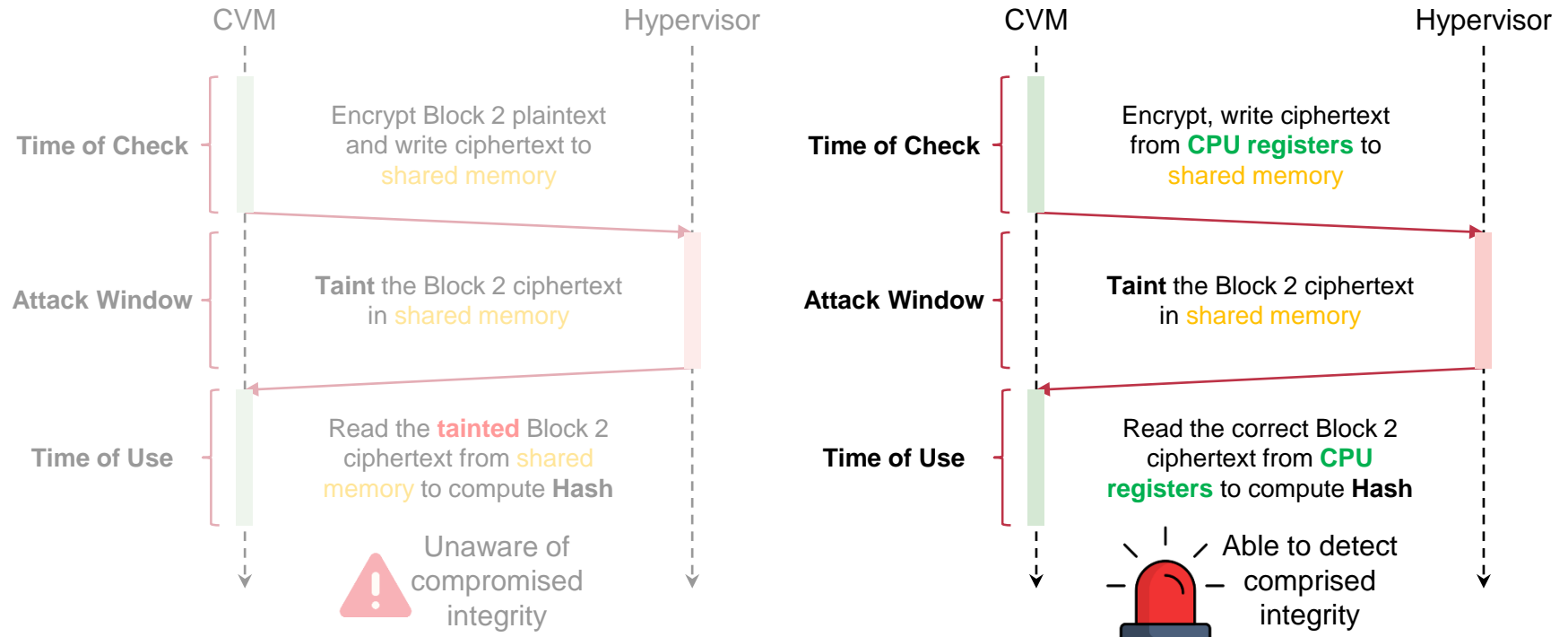
# A TOCTTOU Example in AES-GCM

- For integrity, an Auth Tag is computed iteratively based on each block of ciphertext
- The correctness of the Auth Tag depends on the correctness of each block of ciphertext



# One-Time Trusted Read (OTTR)

- Payload: keep encryption/decryption processing in **CPU registers**
- Header: bounced to private memory before processing



# Side-Channel Attacks on AES Encryption

- **Memory access in AES implementations**
  - Precomputed lookup tables (for performance), S-box (for security)
  - Access patterns on these tables reveal information about encryption keys
- **Cache-based side-channel (though out-of-scope in CVM's threat model)**
  - Access patterns on tables + Known precomputed tables → Possible key leakage
- **ZCED does NOT leak access patterns on tables in private memory**
  - Though attackers may be able to know when encryption is happening
    - Already mitigated in previous work [1,2,3], such as dynamic table storage

[1] Cache Attacks and Countermeasures: the Case of AES (CT-RSA'06)

[2] Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds (CCS'09)

[3] Cross-VM side channels and their use to extract private keys (CCS'12)