Translation Pass-Through for Near-Native Paging Performance in VMs

Shai Bergman[*], Mark Silberstein[*], Takahiro Shinagawa[§],

Peter Pietzuch[†], Lluis Vilanova[†]

[*] TECHNION Israel Institute of Technology

[§] 東京大学 THE UNIVERSITY OF TOKYO

[†] Imperial College London

shaiberg1@tx.technion.ac.il

Background: Virtual Machines

A compute resource that runs programs in its own virtual environment

Used in the cloud for:

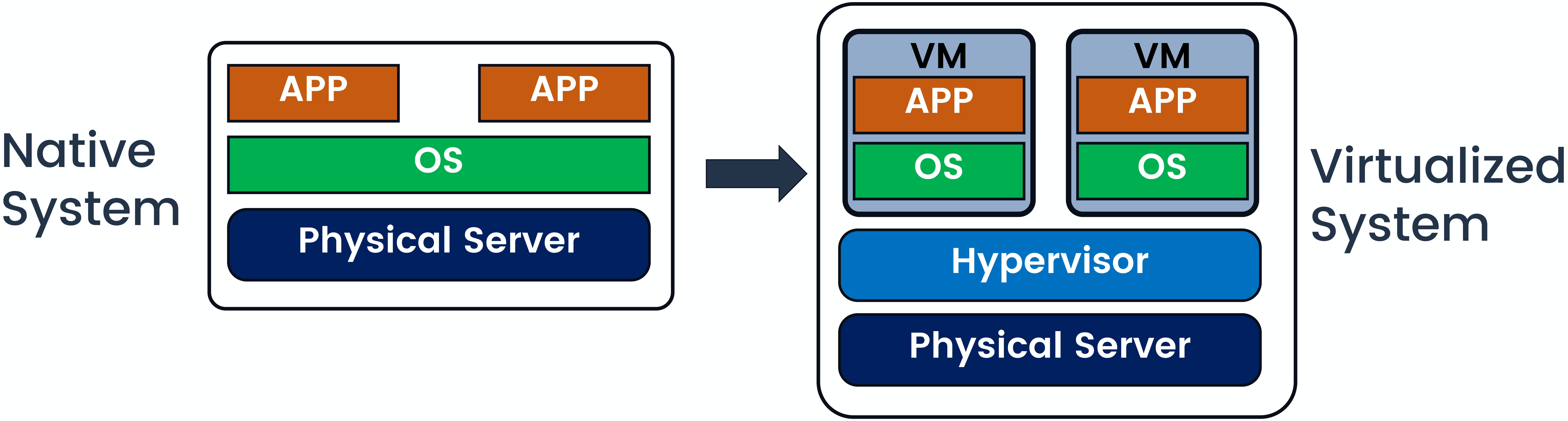
- Consolidation of resources -> <u>Efficiency</u>
- Isolation -> <u>Security</u>
- Resource provisioning -> <u>Flexibility</u>

VMs improve resource utilization, reducing hardware costs and energy consumption.

Background: Virtual Machine Overheads

However, virtualization comes at a cost:

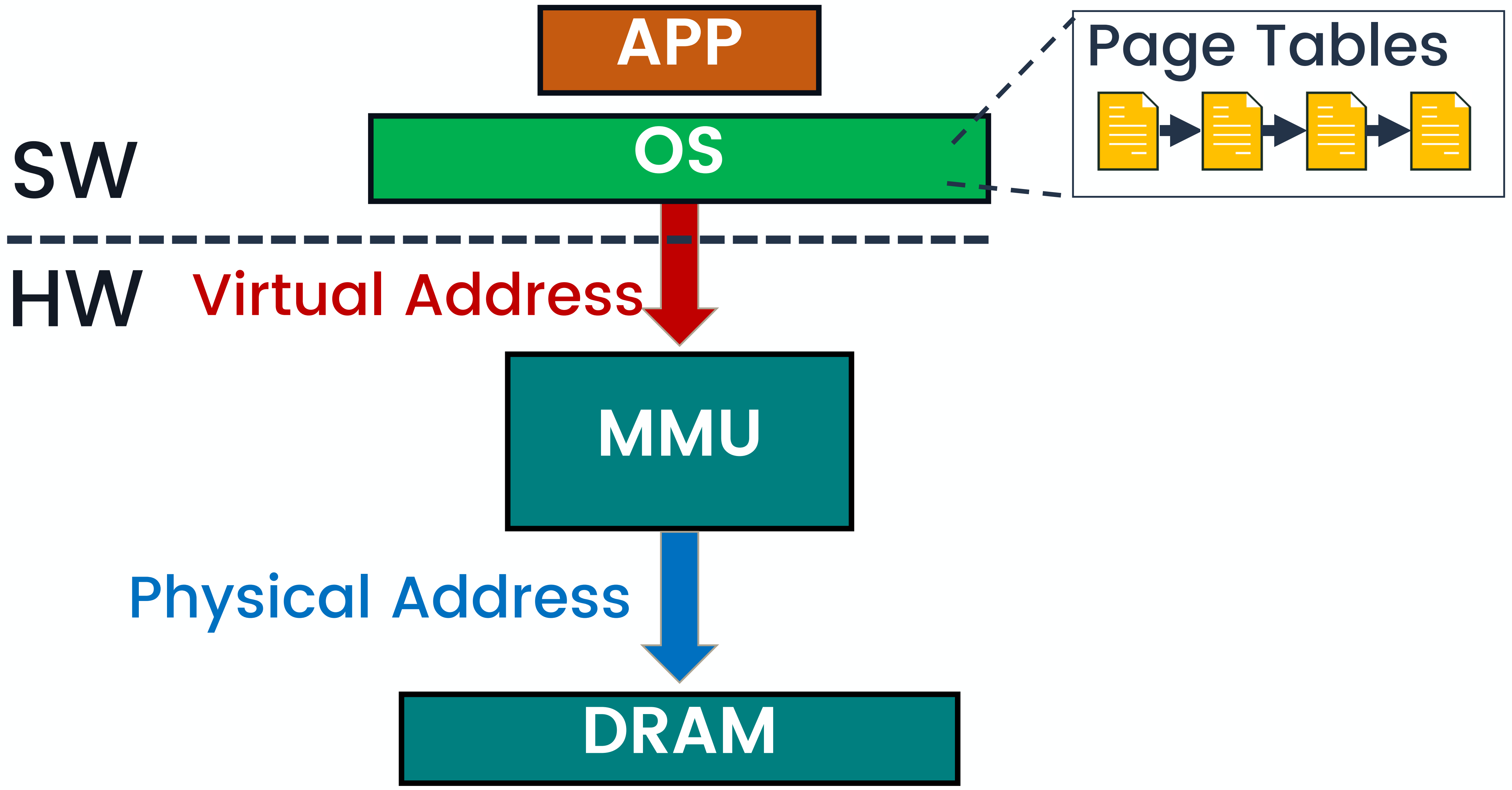
Isolation requirements + VM abstractions impact performance



Memory translation overheads alone cause workload performance slowdown of up to 2.4X*

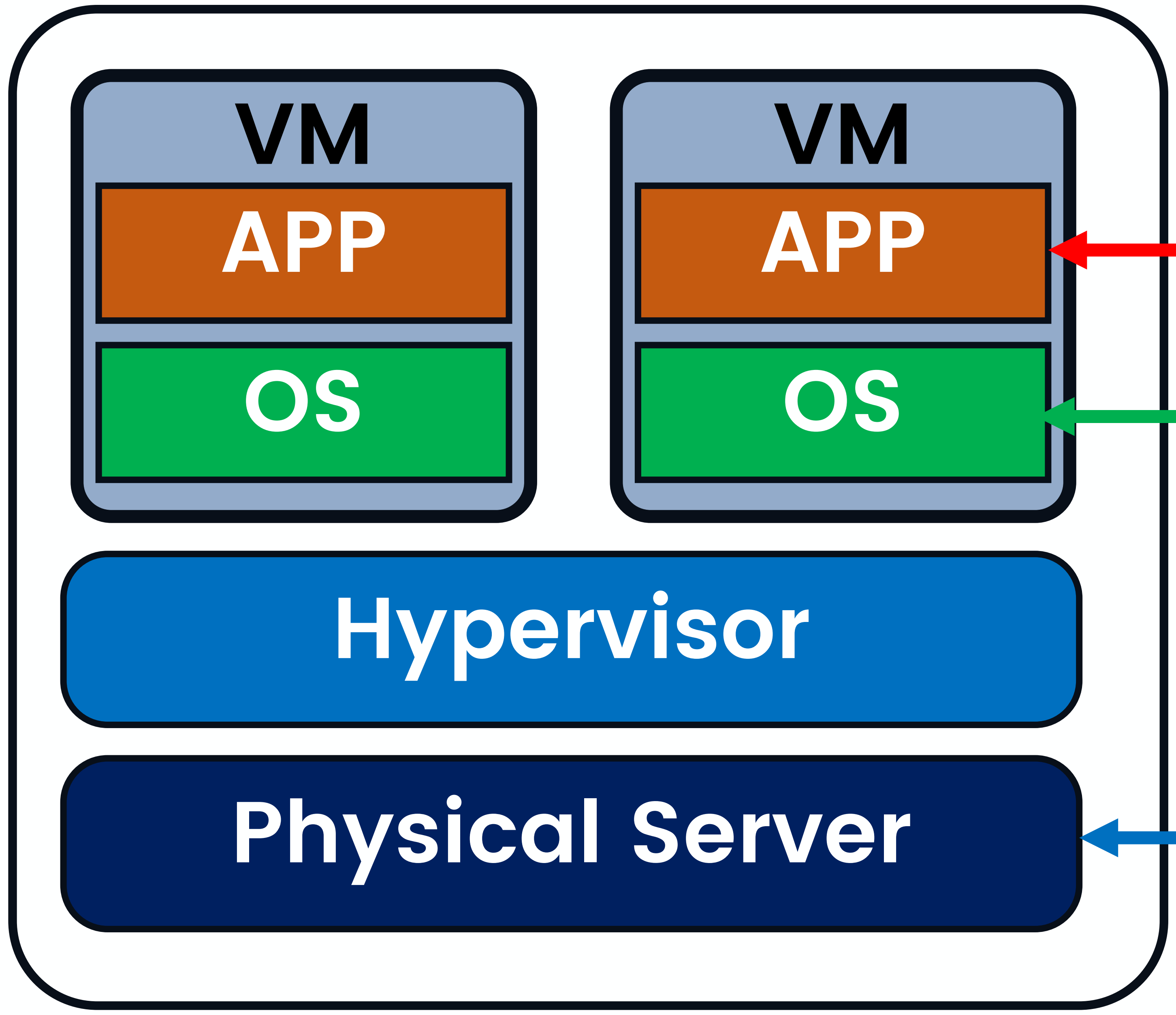
*GUPS: The HPC Challenge (HPCC) Benchmark Suite

Background: Bare Metal Memory Translation



4 memory accesses for translation

Background: VM Memory Translation

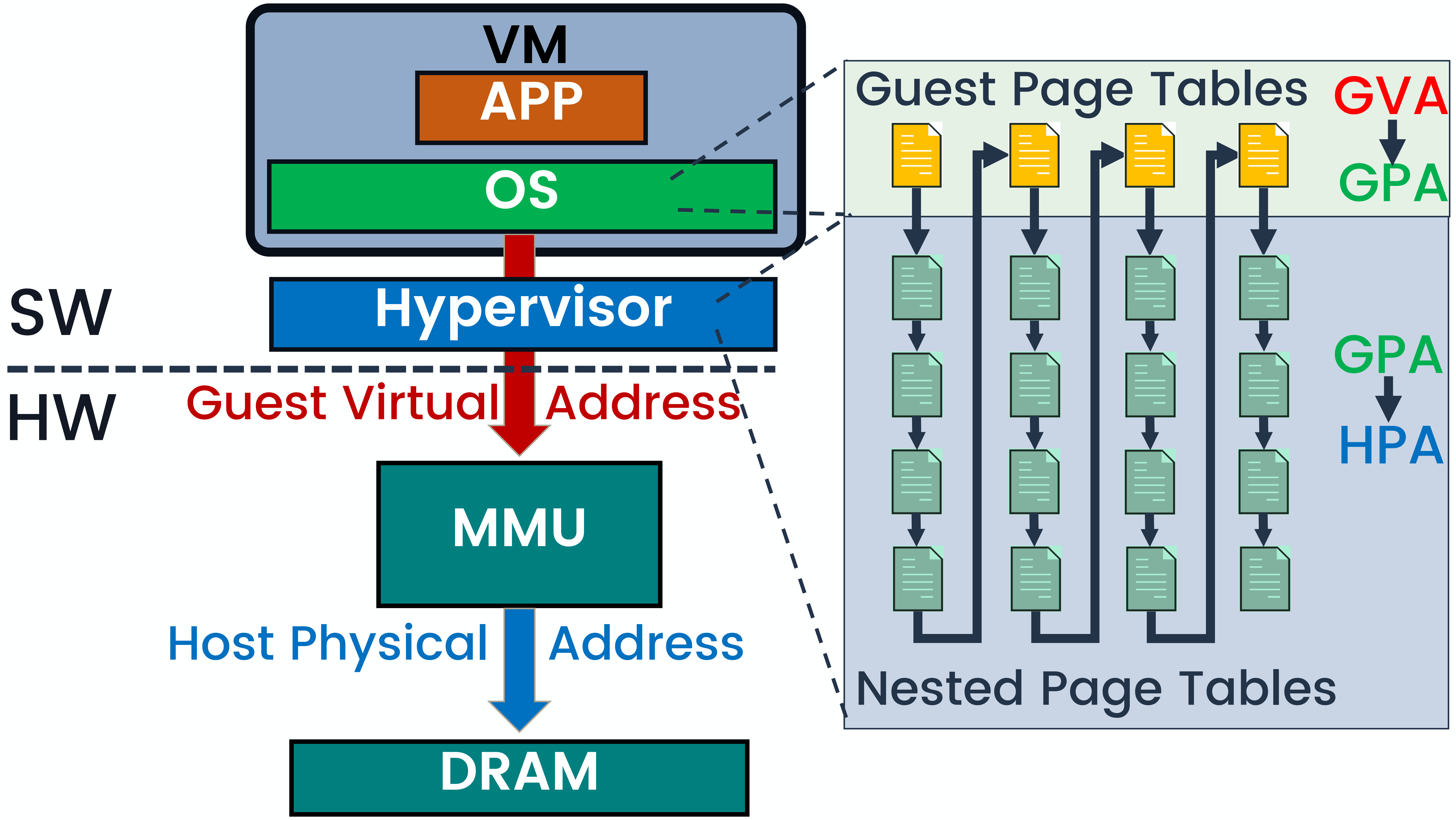


Applications within VMs utilize Guest Virtual Addresses (GVA).

VMs maintain their own Guest Physical Address space (GPA).

Memory is eventually accessed utilizing Host Physical Addresses (HPA).

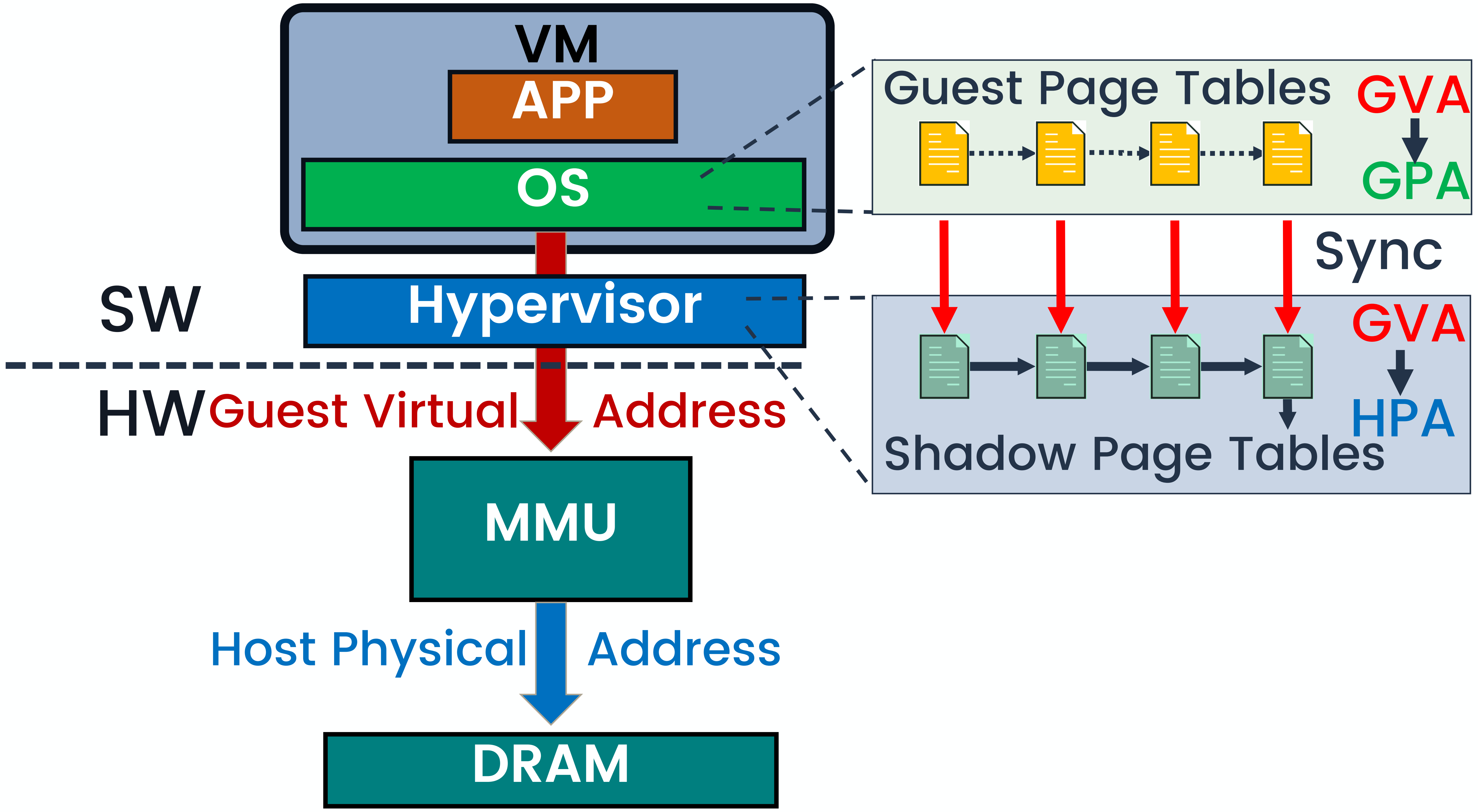
Problem: VM Memory Translation – Nested Paging



Hypervisor maintains NPT

– Up to 24 page walk accesses + No hypervisor intervention per update

Problem: VM Memory Translation – Shadow Paging



Hypervisor virtualizes changes to SPT

+ Up to 4 page walk accesses – Hypervisor intervention per update

VM Memory Translation – Challenges

Both translation and management are important

Current solutions optimize one at the expense of the other

Nested paging:

– Translation: up to 24 page walk accesses on TLB miss

+ Management: no hypervisor intervention per update

Shadow paging:

+ Translation: up to 4 page walk accesses

– Management: hypervisor intervention per page table update (VMEXITs)

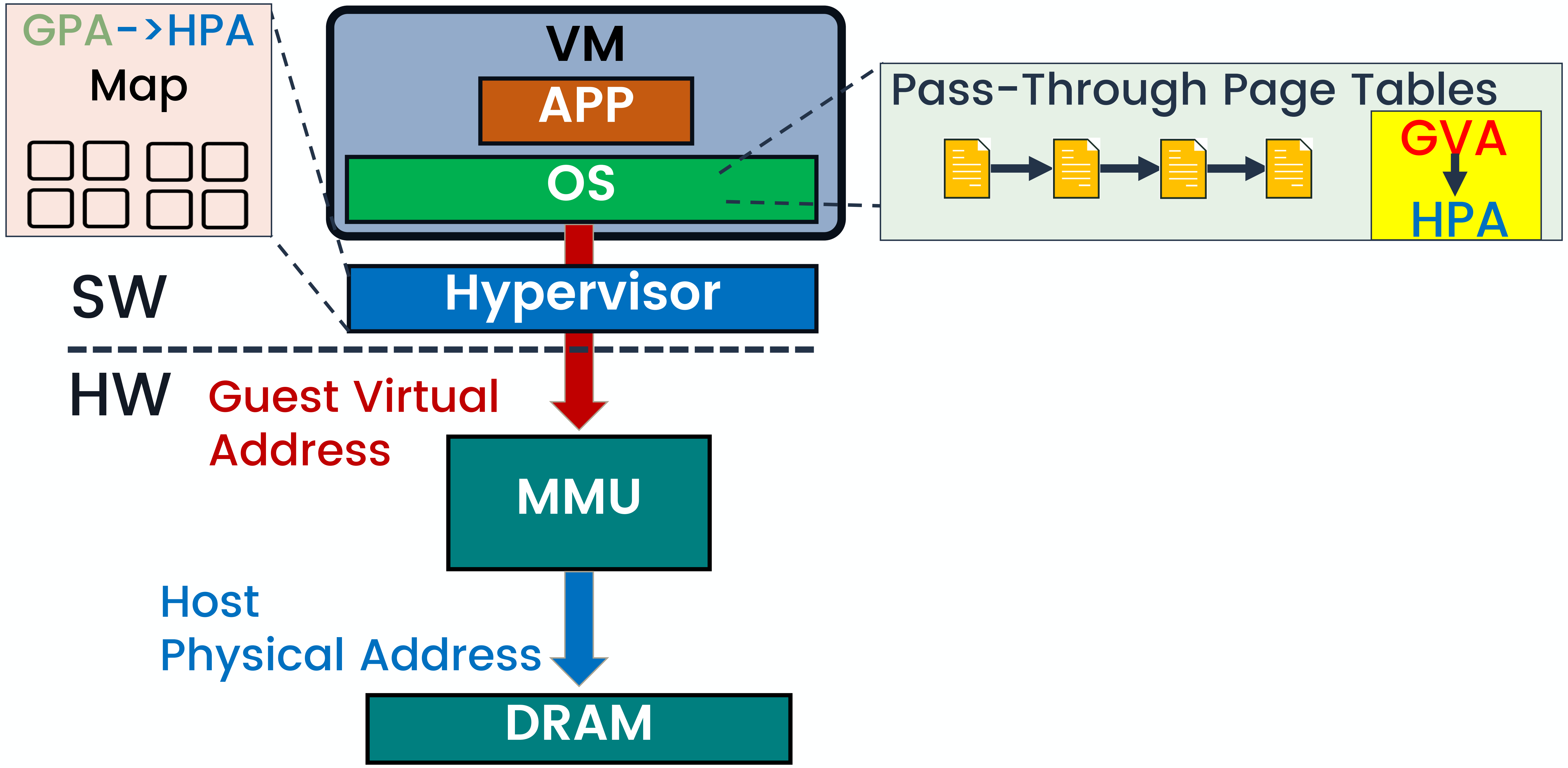
Our main goal: efficient translation and management in virtualized systems

Translation Pass-Through: Design Goals

1. Self-managed, direct guest VM to host memory translation
   - ➢ For native performance for translation and management
2. Efficiently maintain protection between VMs
   - ➢ Without hypervisor intervention on page table updates
3. Ease of integration and maintenance
   - ➢ For fast adoption and backwards compatibility

1. Self-managed, Direct Guest VM to Host Memory Translation

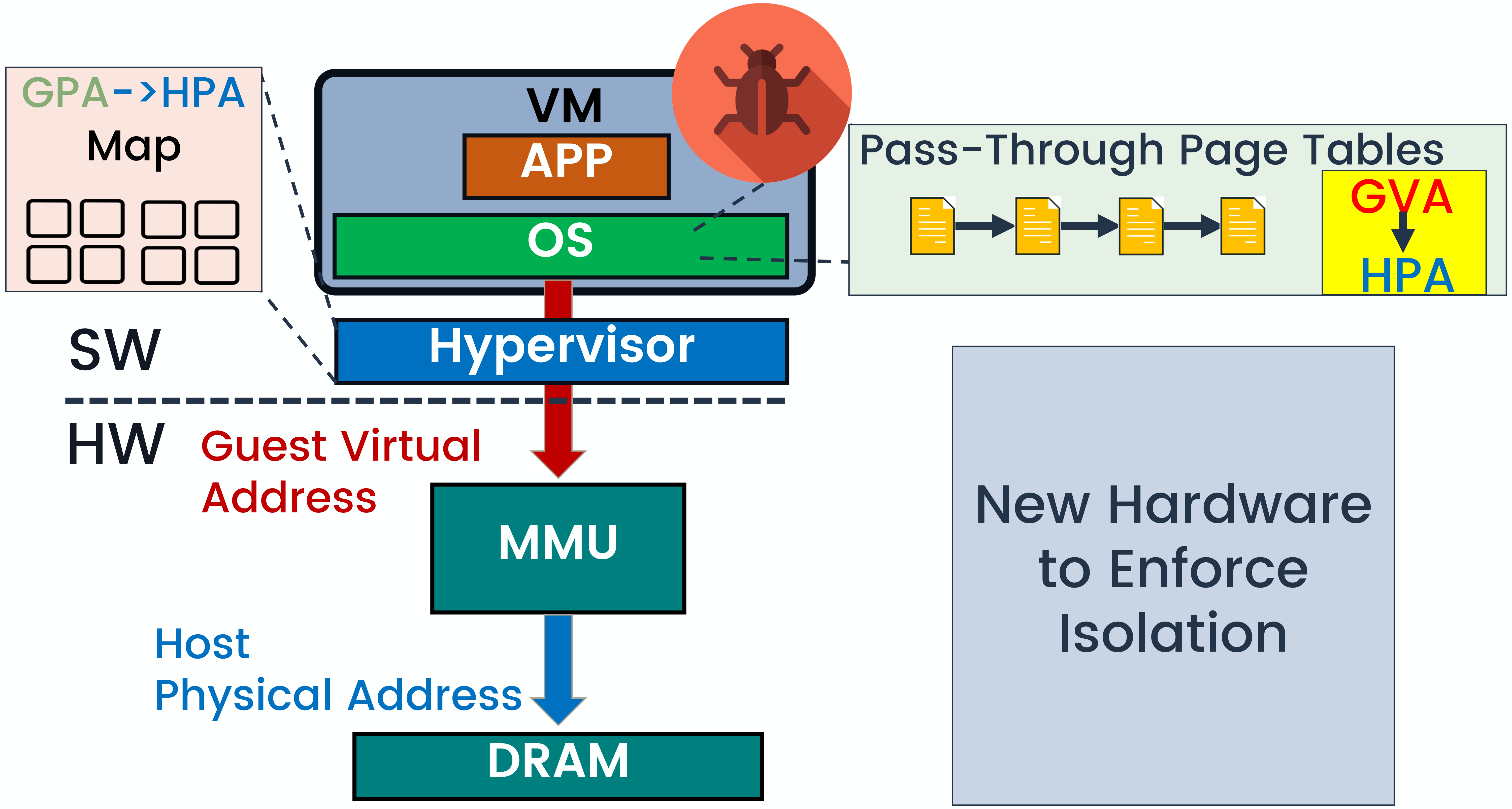
- Guest VM directly manages and translates GVA->HPA



Translation:

- Guest managed page tables
- Translate directly to HPA
- Utilized directly by MMU

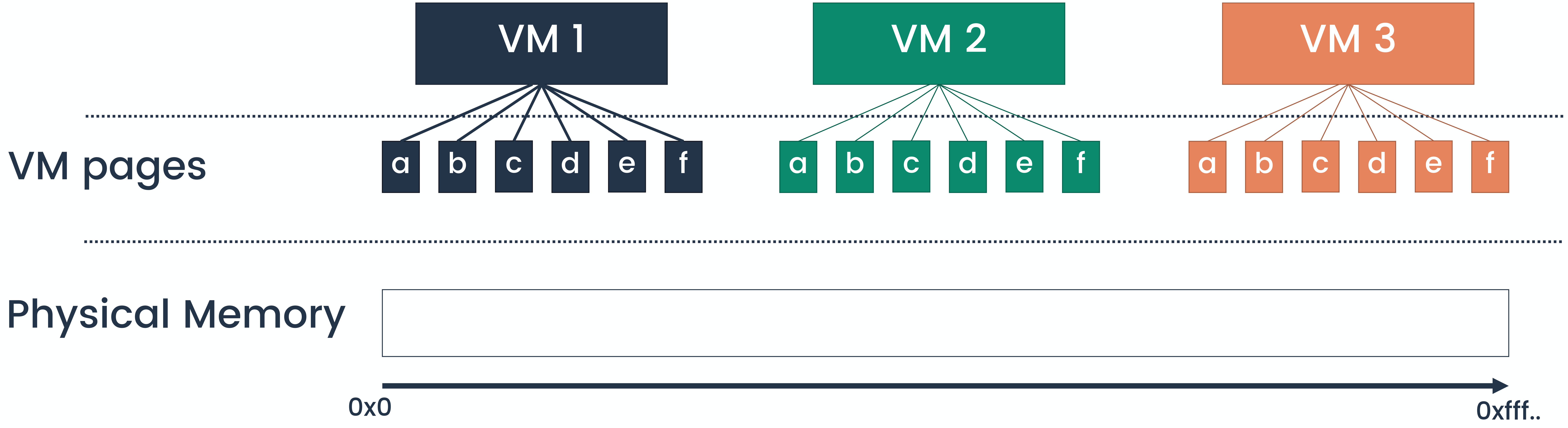
2. Maintain Protection Between VMs



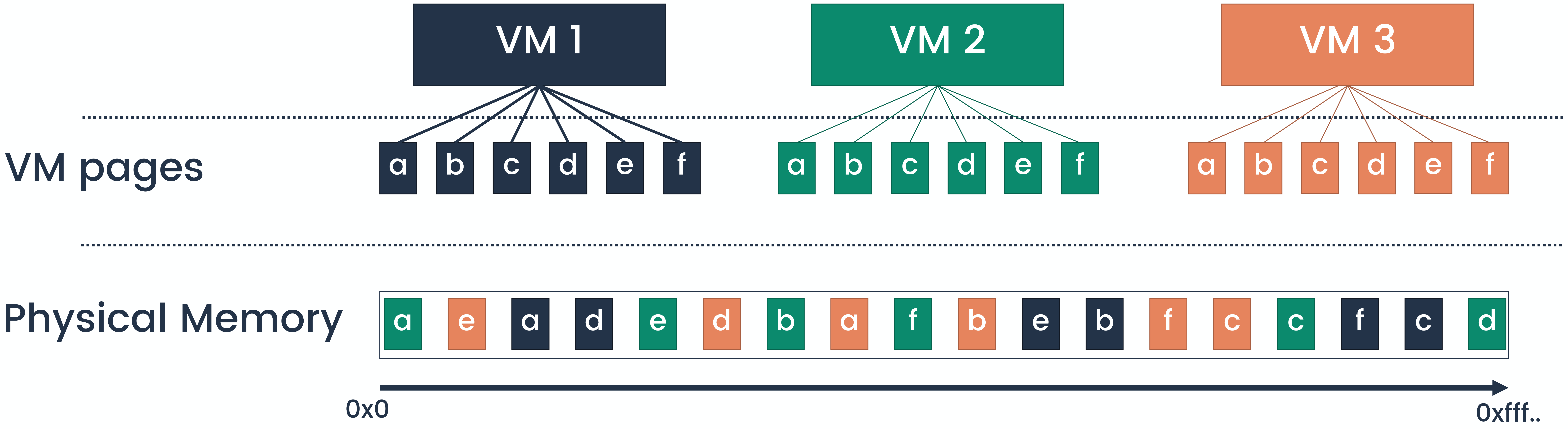
Translation:

- Guest managed page tables
- Translate directly to HPA
- Utilized directly by MMU

Proposed Hardware: Physical Page Tags

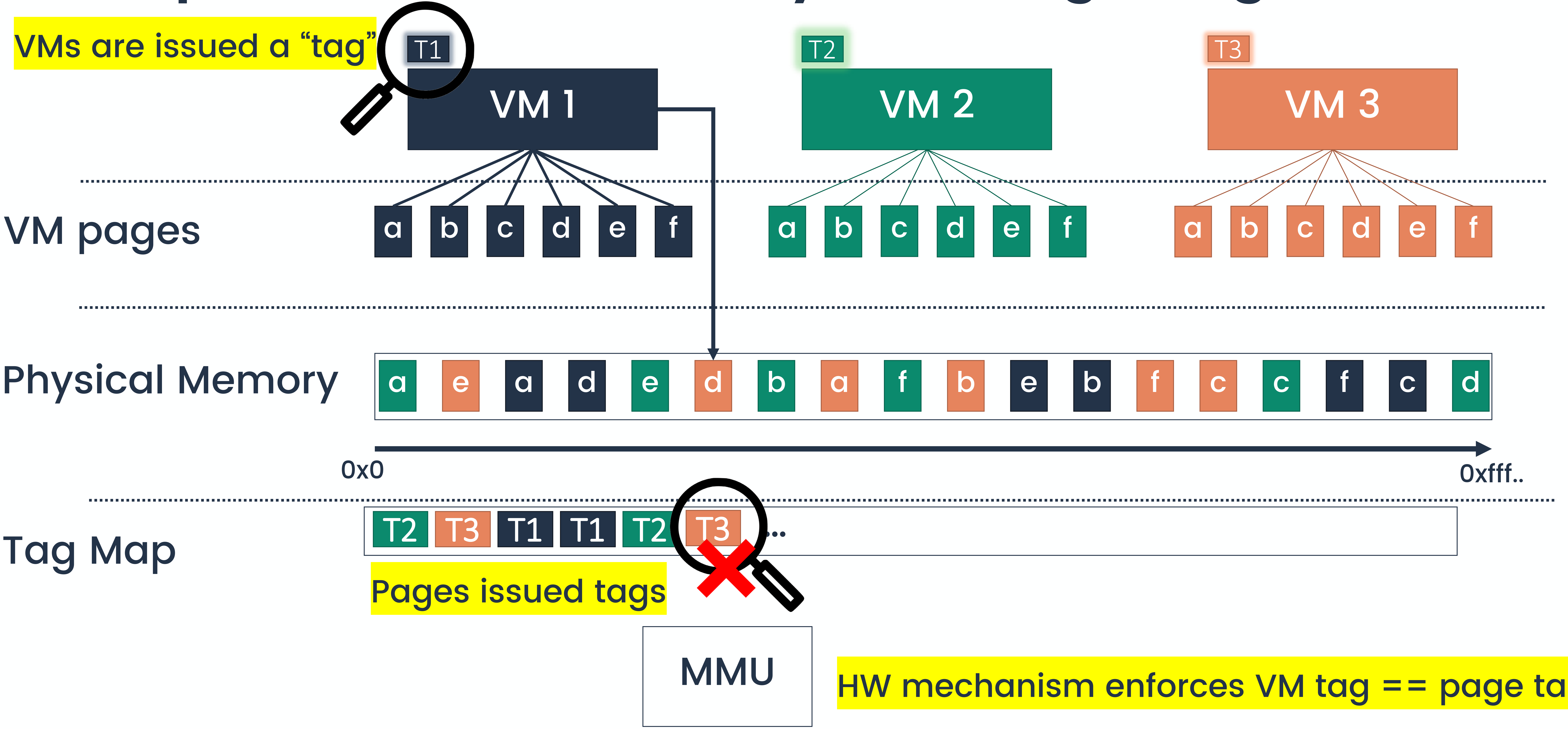


Proposed Hardware: Physical Page Tags



Proposed Hardware: Physical Page Tags

VMs are issued a "tag"



Pages issued tags

HW mechanism enforces VM tag == page tag.

Proposed Hardware: Physical Page Tags

Hardware is available today (with minor modifications)

AMD
SEV-SNP

Other mechanisms that could also be used

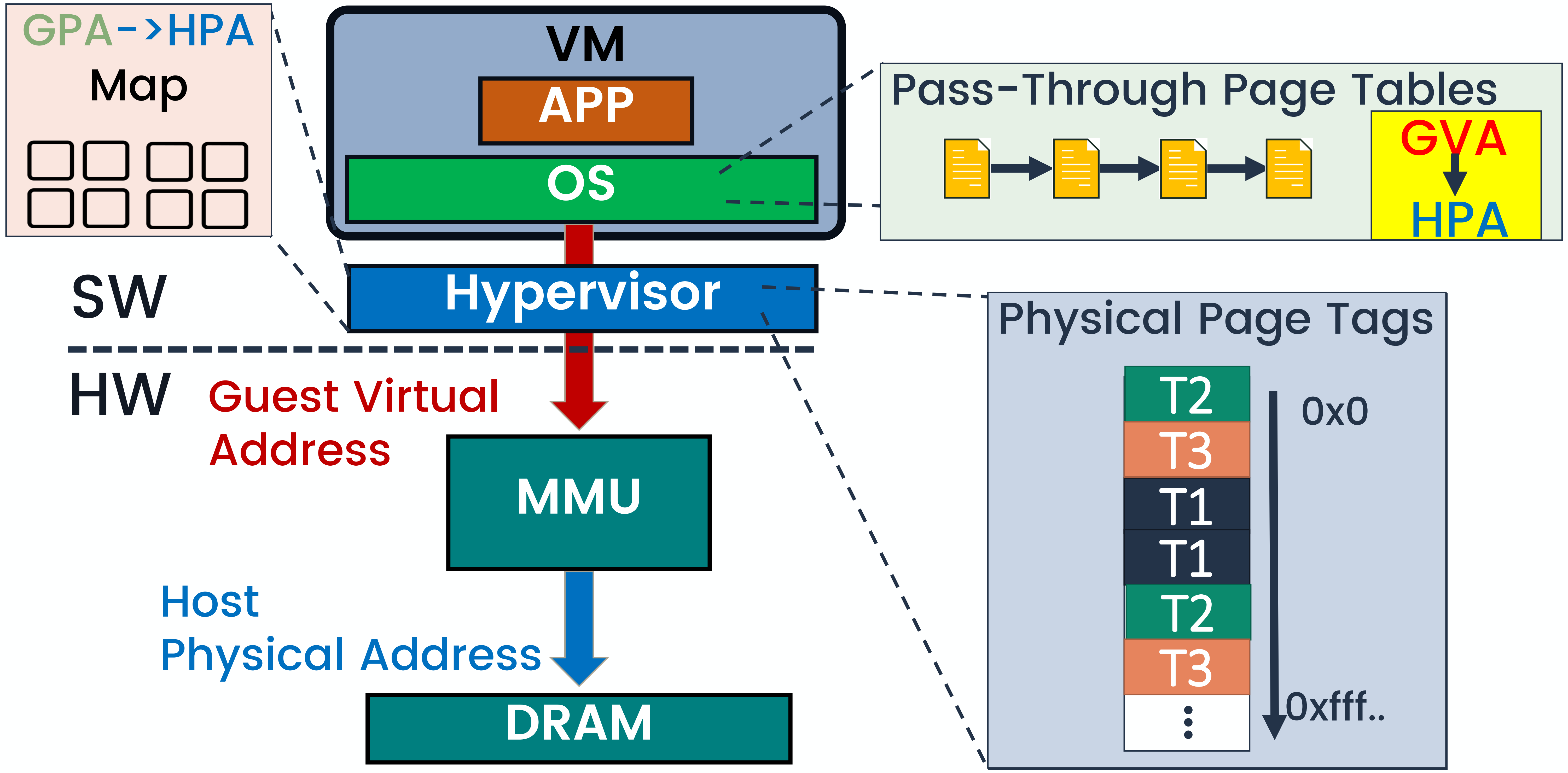
arm
CHERI

RISC-V®
PMP

Physical page tag checks can overlap memory translation

2. Maintain Protection Between VMs

➢ Hypervisor assigns MMU tags to VMs and HPAs



Translation:

- Guest managed page tables
- Translate directly to HPA
- Utilized directly by MMU

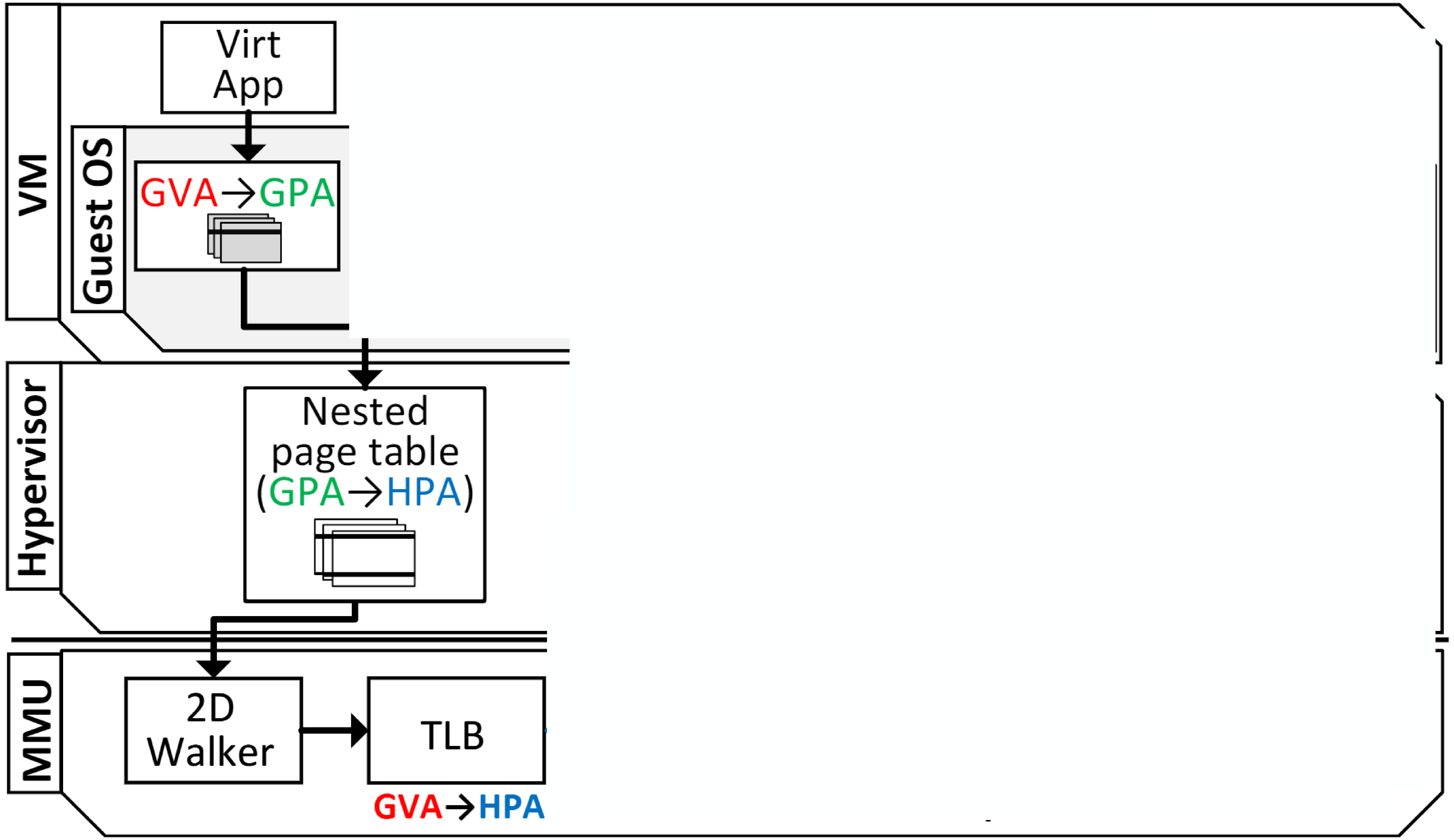
Isolation:

- Hypervisor managed Page Tags
- Enforce inter-VM isolation.

Decouple translation and isolation

Overlap page table traversal and tag lookup

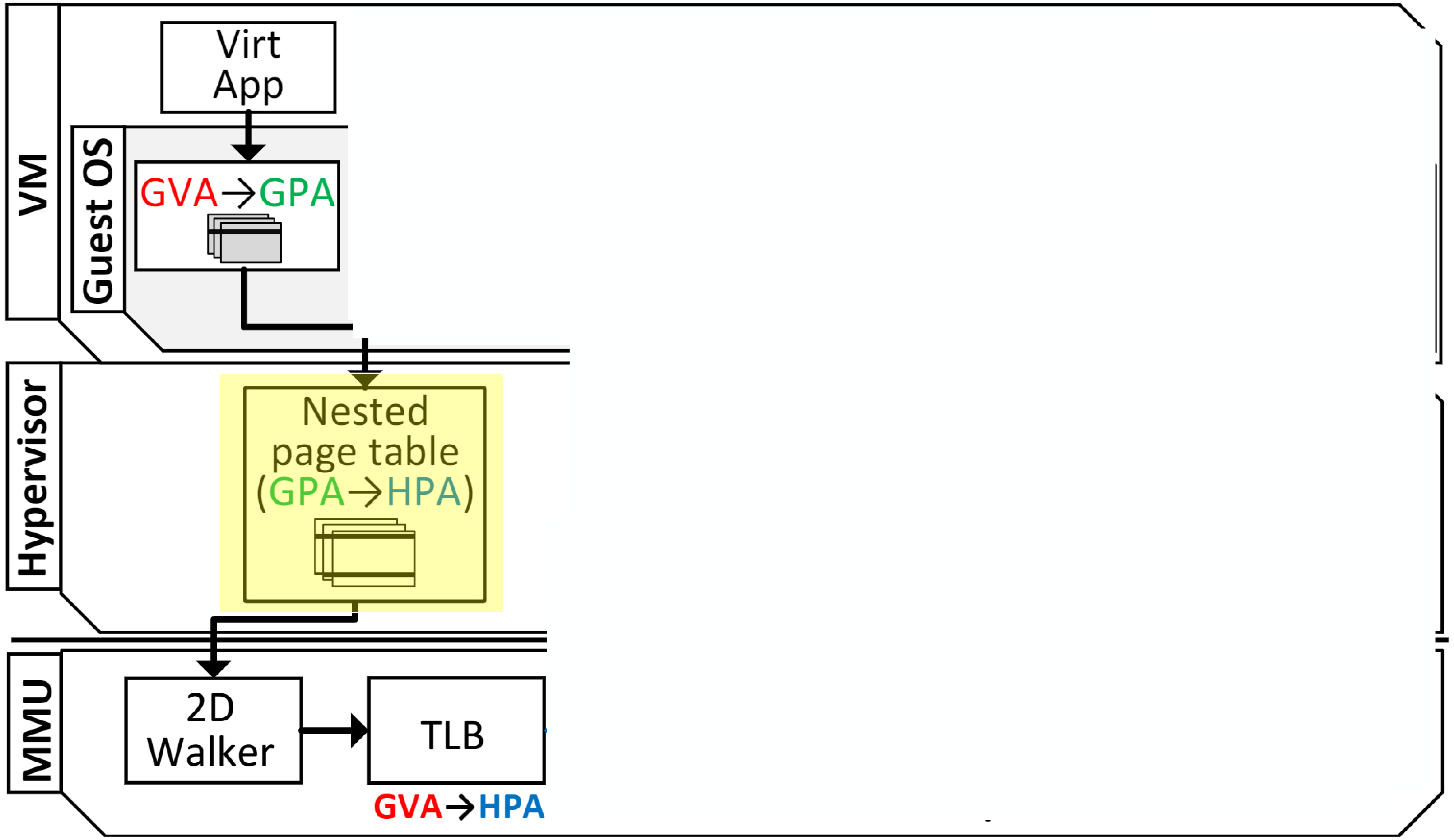
3. Ease of Integration and Maintenance



Translation Pass-Through for Near-Native Paging Performance in VMs. USENIX ATC 23

3. Ease of Integration and Maintenance

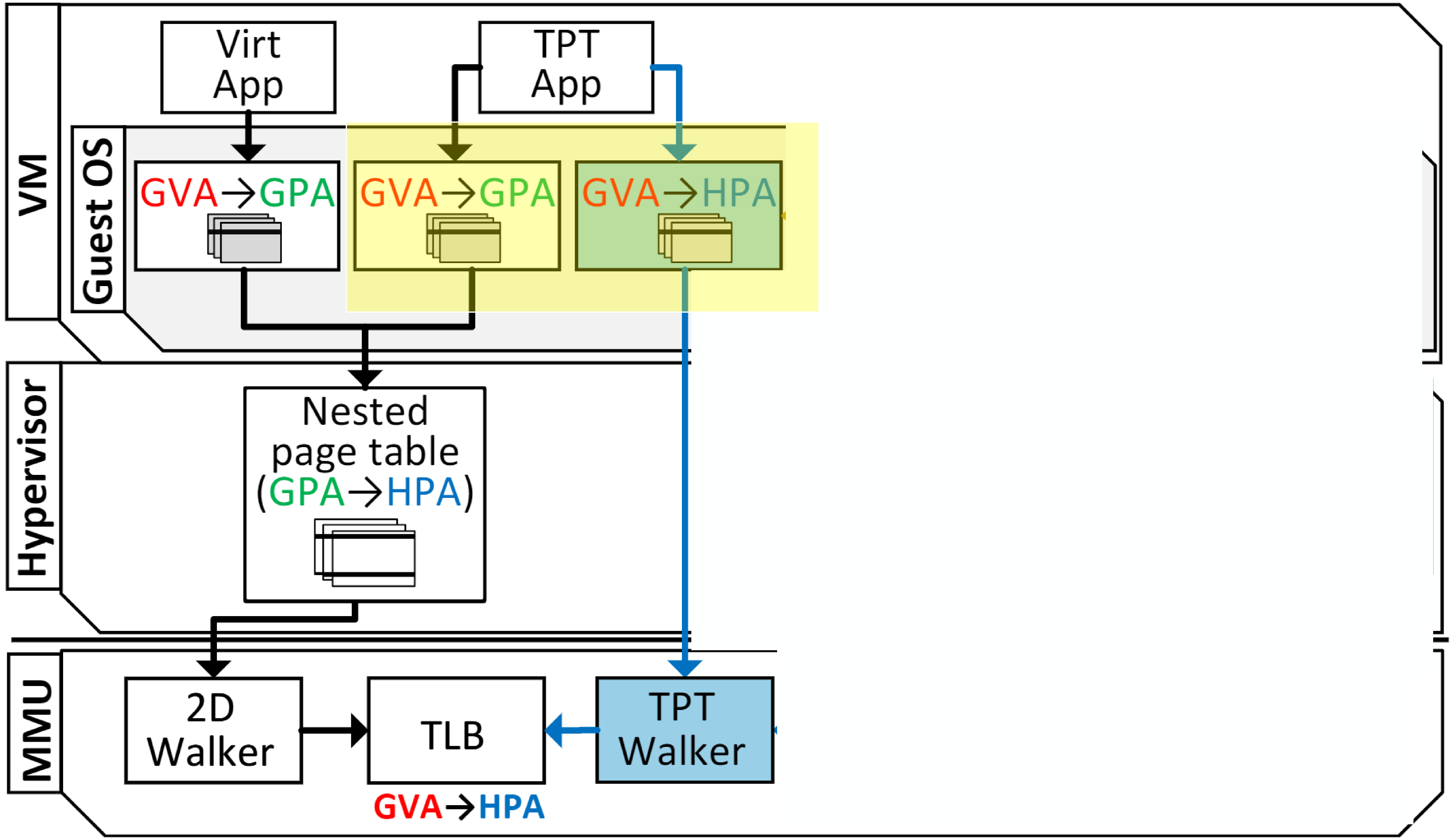
- Uses NPT by default
  For backwards compat.
  (e.g., boot)



3. Ease of Integration and Maintenance

- Uses NPT by default
  For backwards compat. (e.g., boot)

- Dual page tables
  for native translation performance & hypervisor fallback



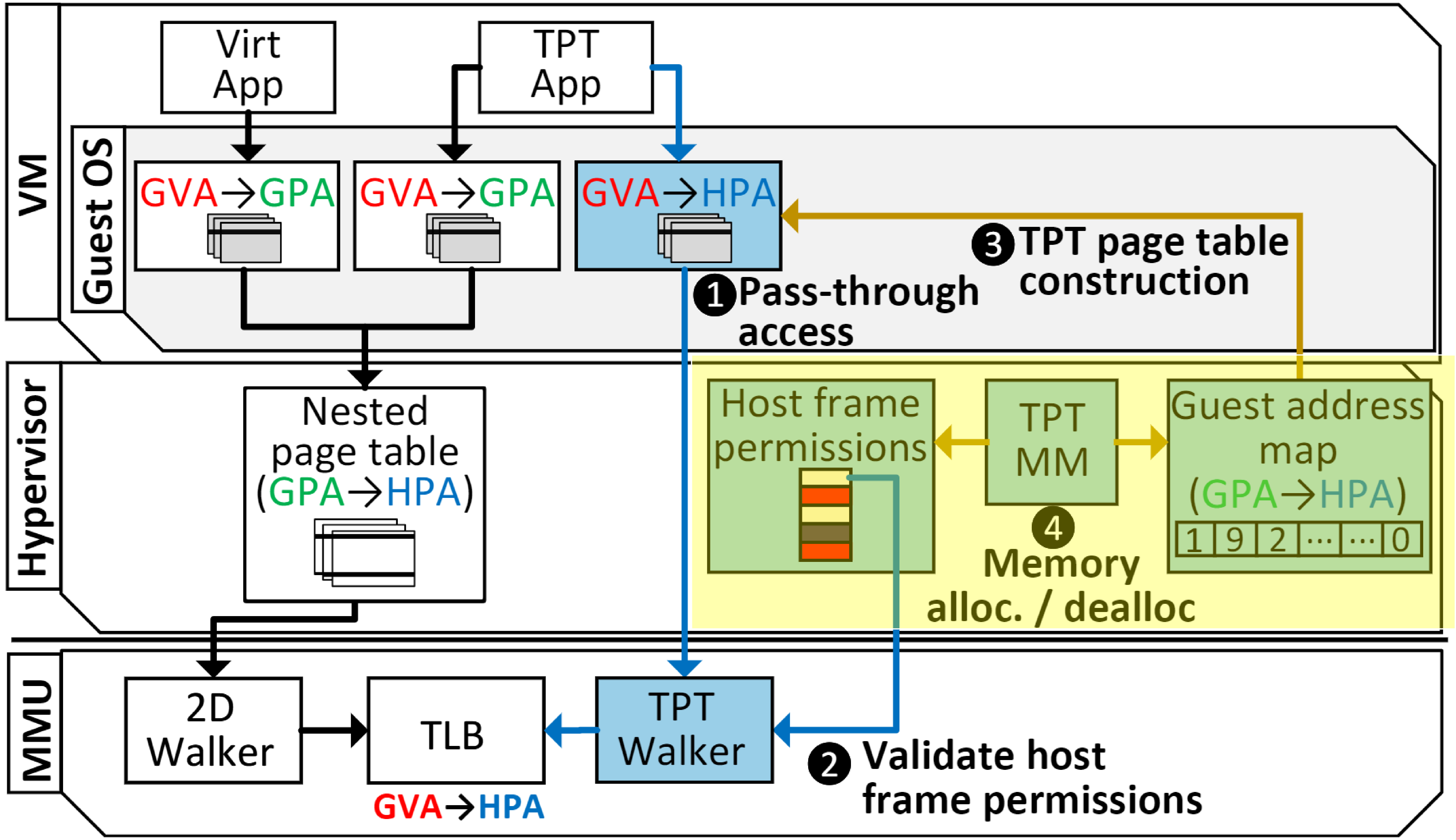
Translation Pass-Through for Near-Native Paging Performance in VMs. USENIX ATC 23

3. Ease of Integration and Maintenance

- Uses NPT by default
  For backwards compat. (e.g., boot)

- Dual page tables
  for native translation performance & hypervisor fallback

- TPT construction
  for guest-managed GVA->HPA page tables

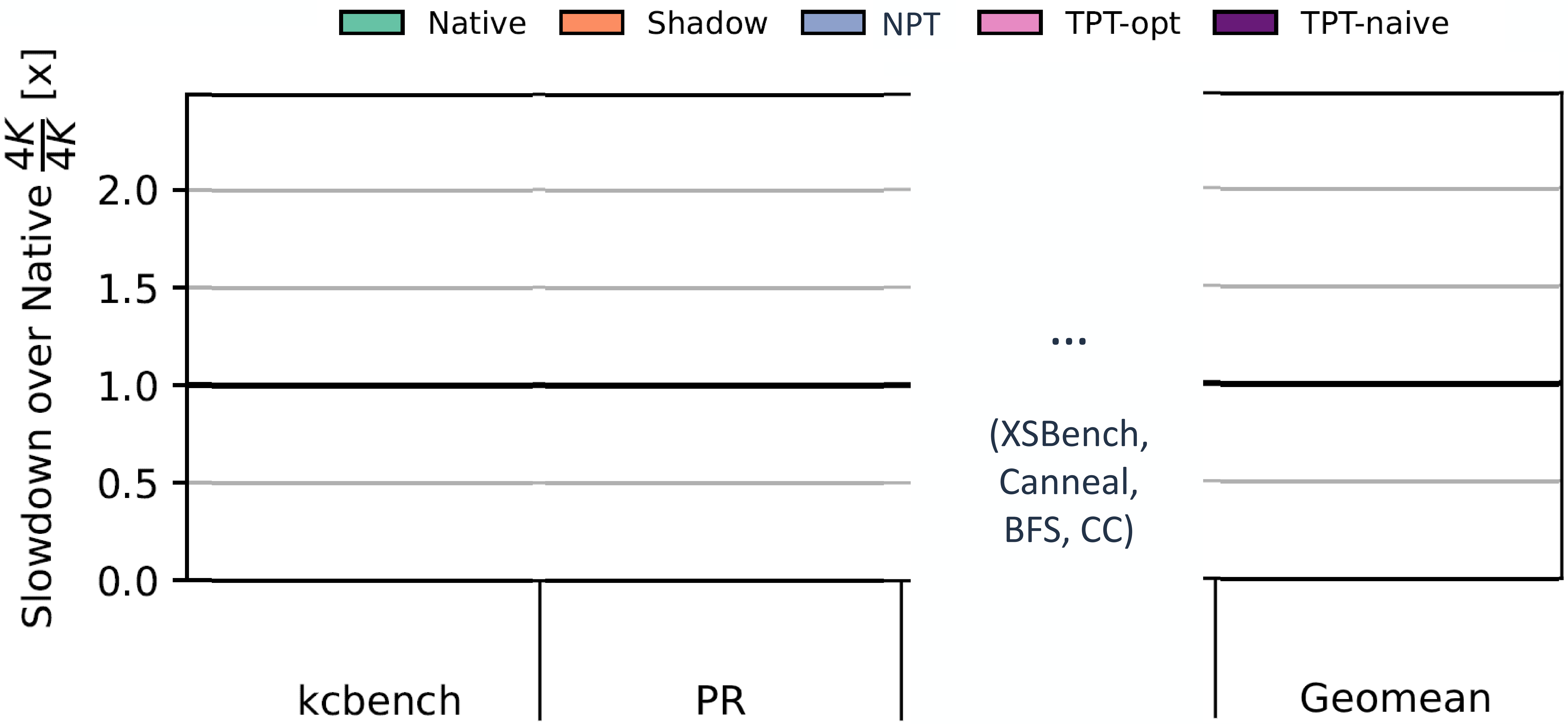


TPT Evaluation: Setup

Configurations:

- Native
- VM + Shadow Paging
- VM + NPT (nested paging)
- TPT-opt (full tag-check overlap)
- TPT-naïve (no tag-check overlap)

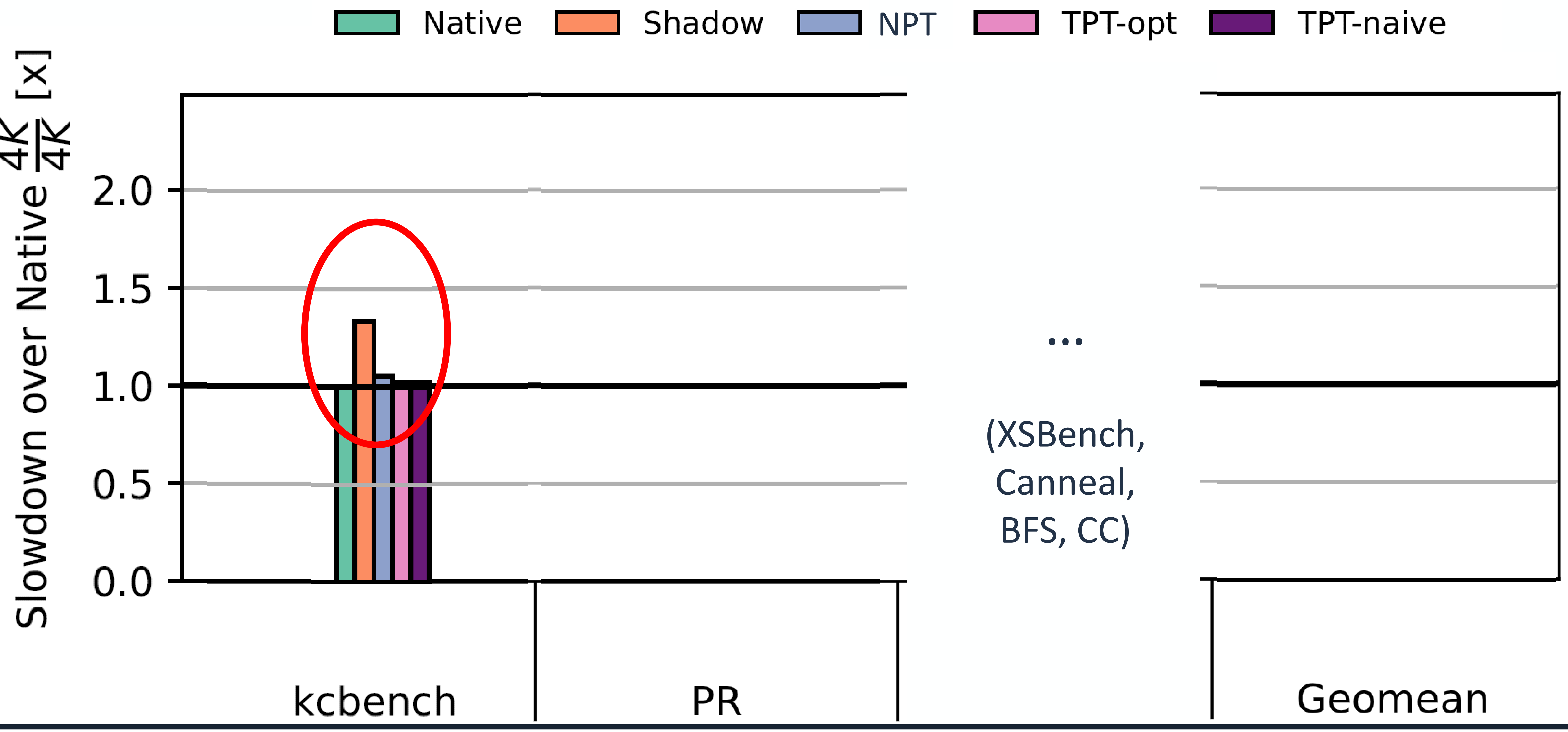
TPT Evaluation: Application performance



…

(XSBench, Canneal, BFS, CC)

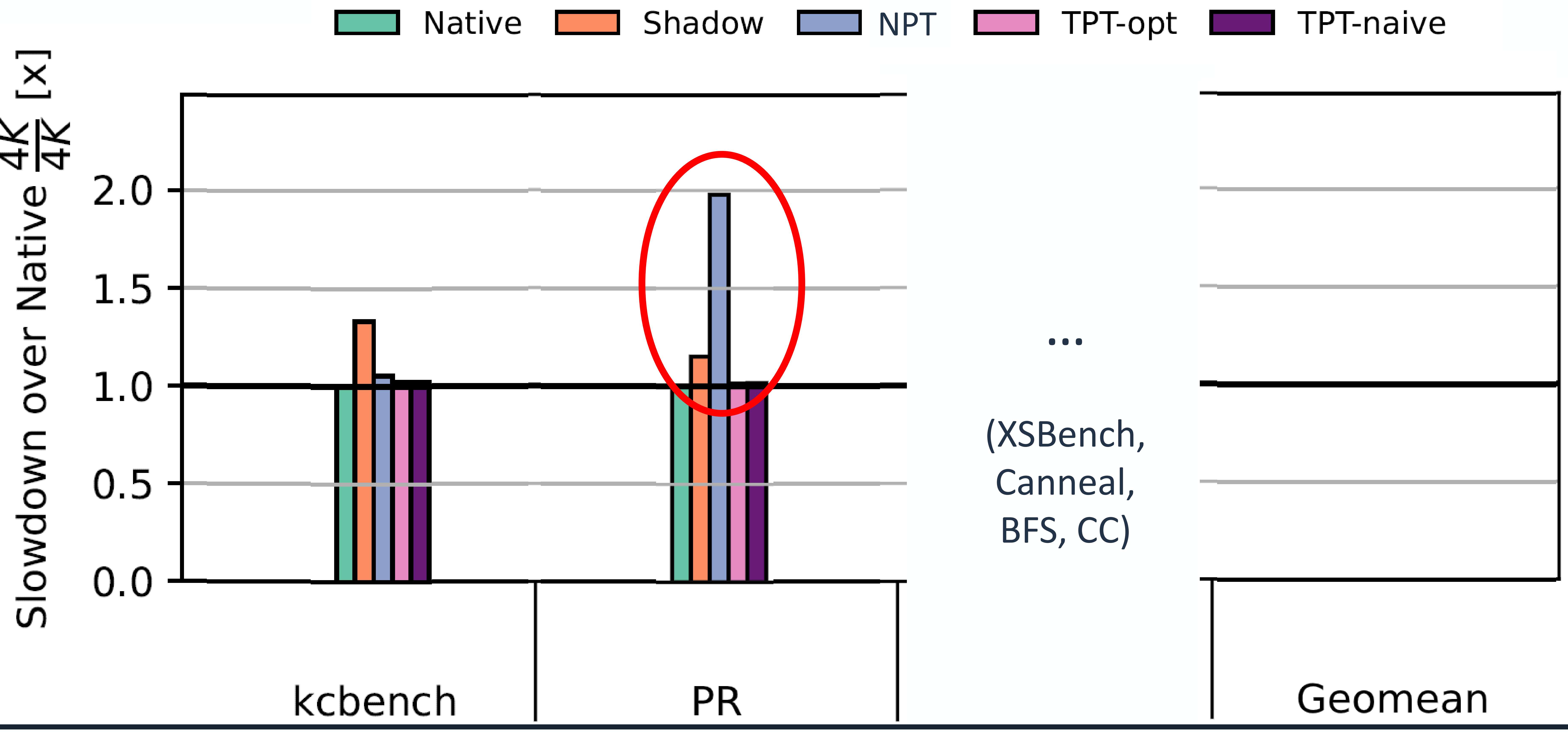
TPT Evaluation: Application performance



kcbench spawns processes and constructs new page tables

TPT has no page table manipulation overheads due to self-managed page tables

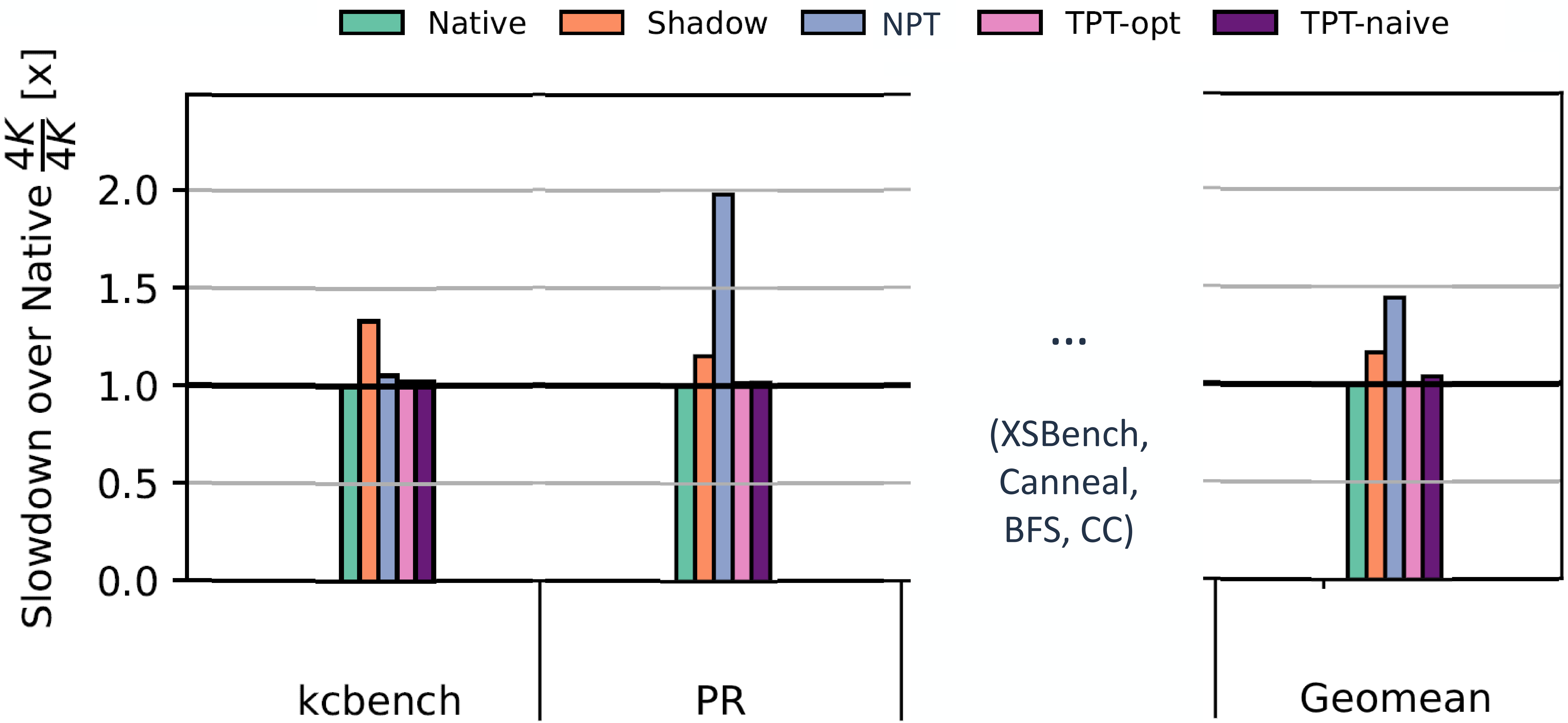
TPT Evaluation: Application performance



PR access pattern causes high TLB miss rates

TPT has no translation overheads due to pass-though translations

TPT Evaluation: Application performance

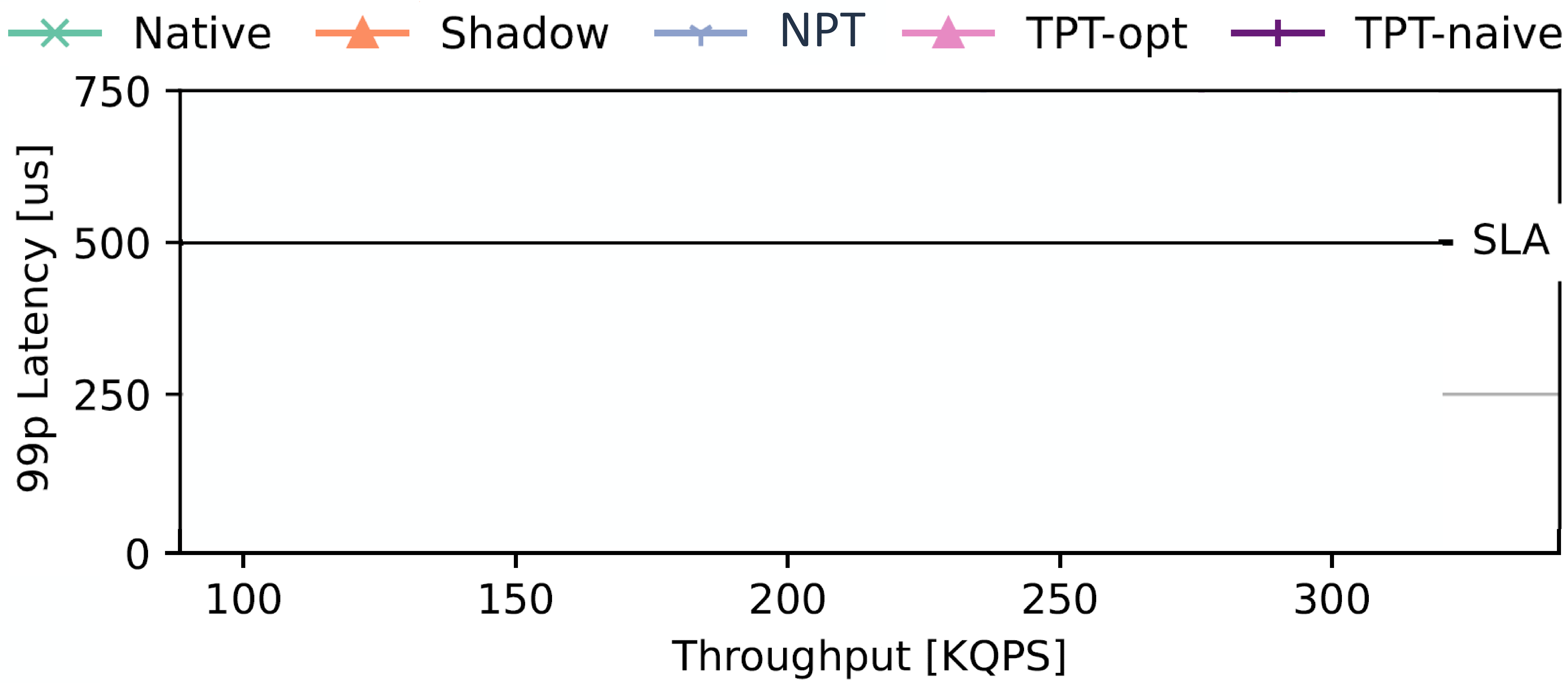


(XSBench, Canneal, BFS, CC)

Translation Pass-Through for Near-Native Paging Performance in VMs. USENIX ATC 23

TPT Evaluation: memcached

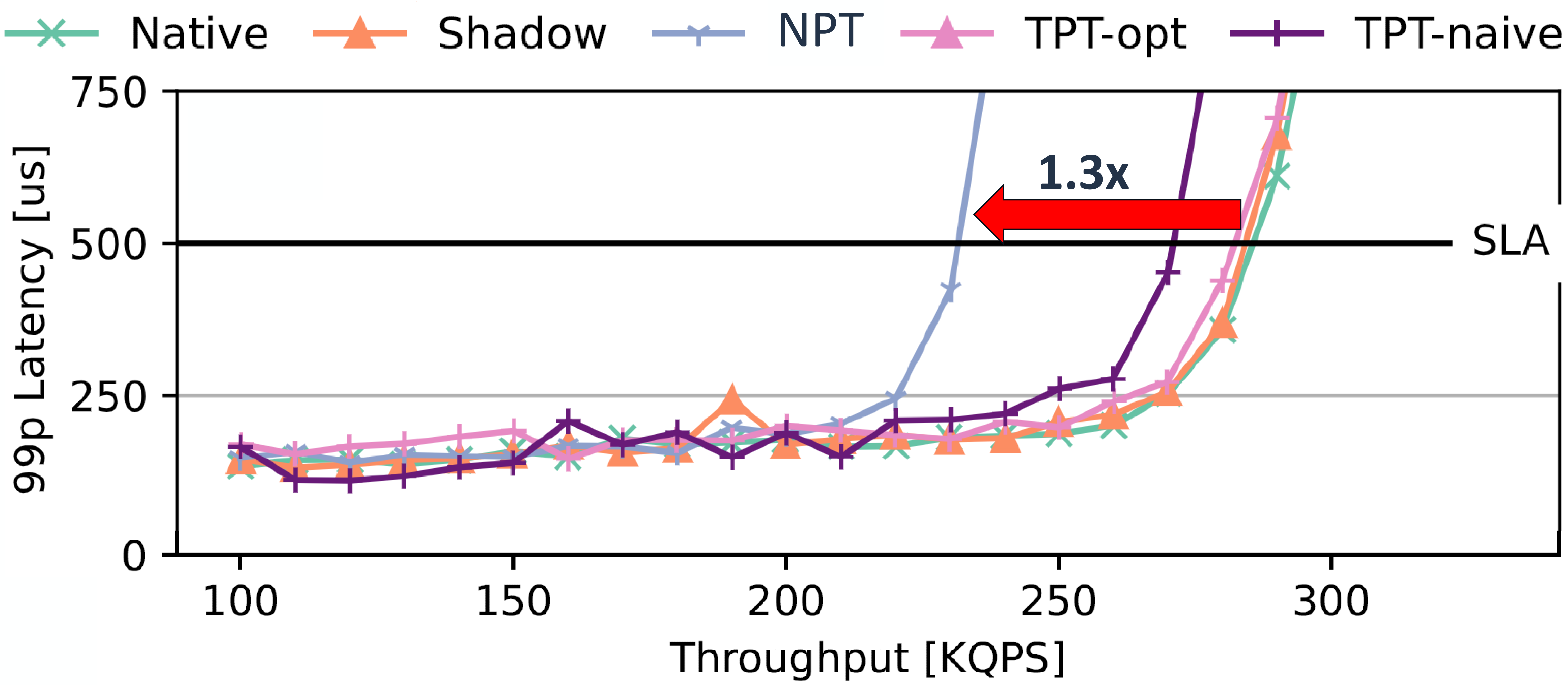
Serving Facebook ETC workload



Translation Pass-Through for Near-Native Paging Performance in VMs. USENIX ATC 23

TPT Evaluation: memcached

Serving Facebook ETC workload



Memcached tail latency is very sensitive to TLB misses

TPT with translation and tag checking overlap matches native performance

Conclusions

- Virtualization has high memory management overheads
  - Due to translation <-> isolation coupling
  - Worse with larger data sets and address spaces (e.g., 5-level pg. table)
- TPT eliminates translation and page table management overheads
  - VM address translation in parallel with inter-VM isolation via tagging
  - Supported with minor software changes + backwards compat. (e.g., boot)
  - Hardware support is almost all there (e.g., AMD SEV-SNP)

Thank you!
Questions?

shaiberg1@tx.technion.ac.il github.com/acsl-technion/TPT