MICRO 2026
59th IEEE/ACM International Symposium on Microarchitecture

Available Reviewed Reproducible

SagePTE

Co-Locating Guest and Host PTEs to Accelerate Virtual Memory Translation

Amir Noohi
University of Edinburgh

Mostafa Derispour
University of Edinburgh

Rodrigo C. O. Rocha
Huawei

Antonio Barbalace
University of Edinburgh

Artemiy Margaritov
Huawei

Athens, Greece | 31 October – 4 November 2026

Virtualization

Virtualization is ubiquitous

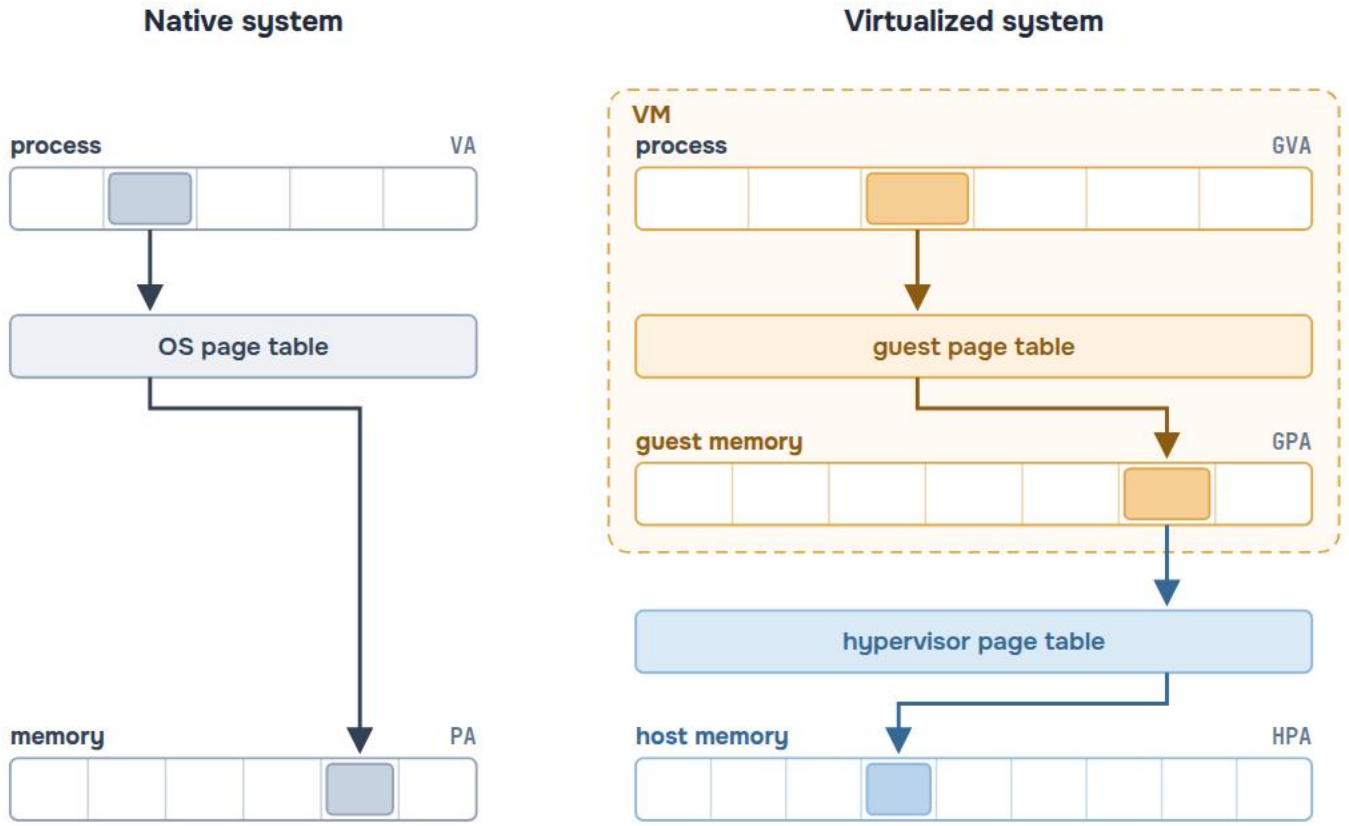
Every cloud server runs many virtual machines, each with its own operating system, isolated from the others by a hypervisor.

Native: one translation

The OS's page table maps each page of a process to a page of memory.

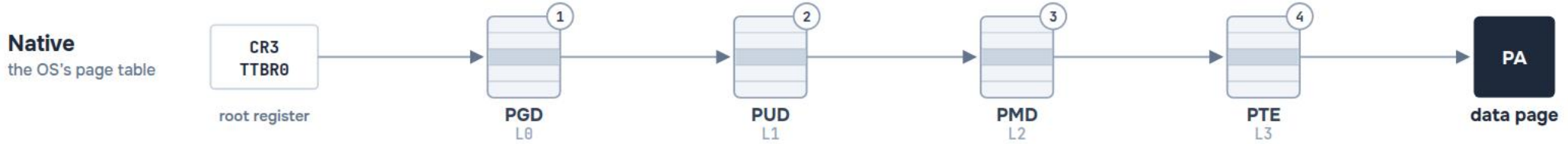
In a VM: two translations

The guest's page table maps the page only to guest memory; the hypervisor's page table maps that to host memory. Every memory access needs both.



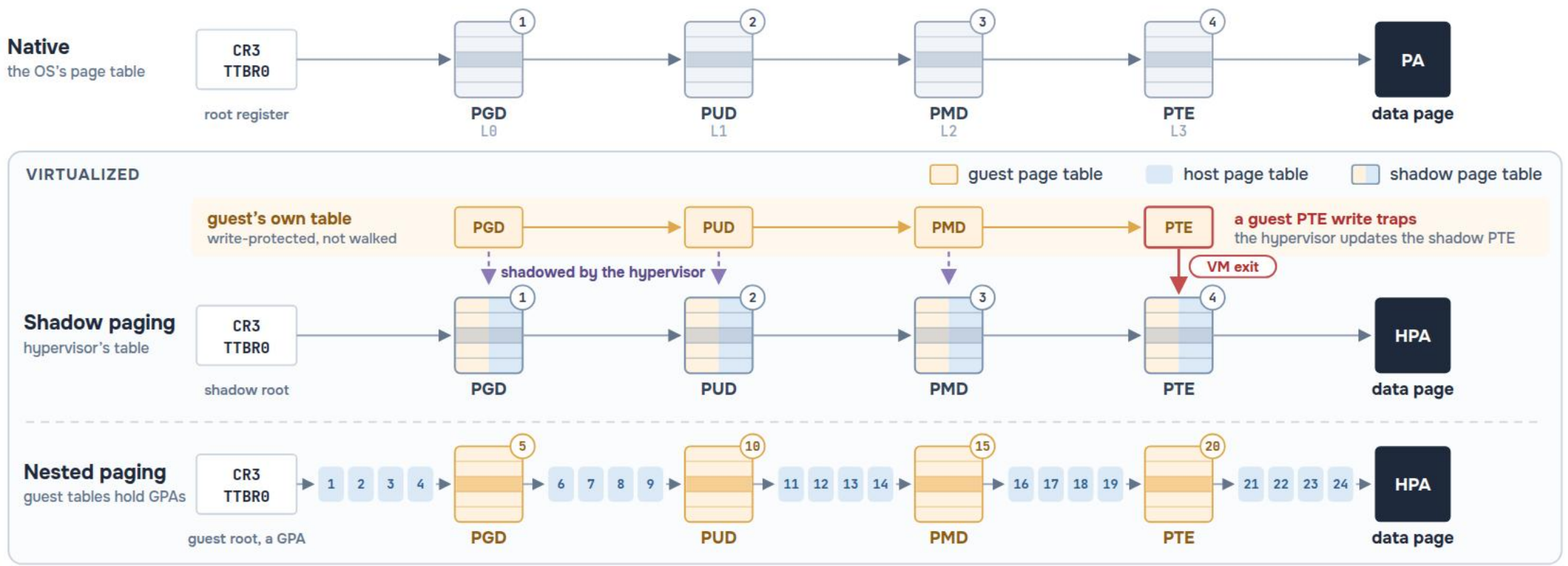
Amir Noohi amir.noohi@ed.ac.uk

Address translation in virtualization

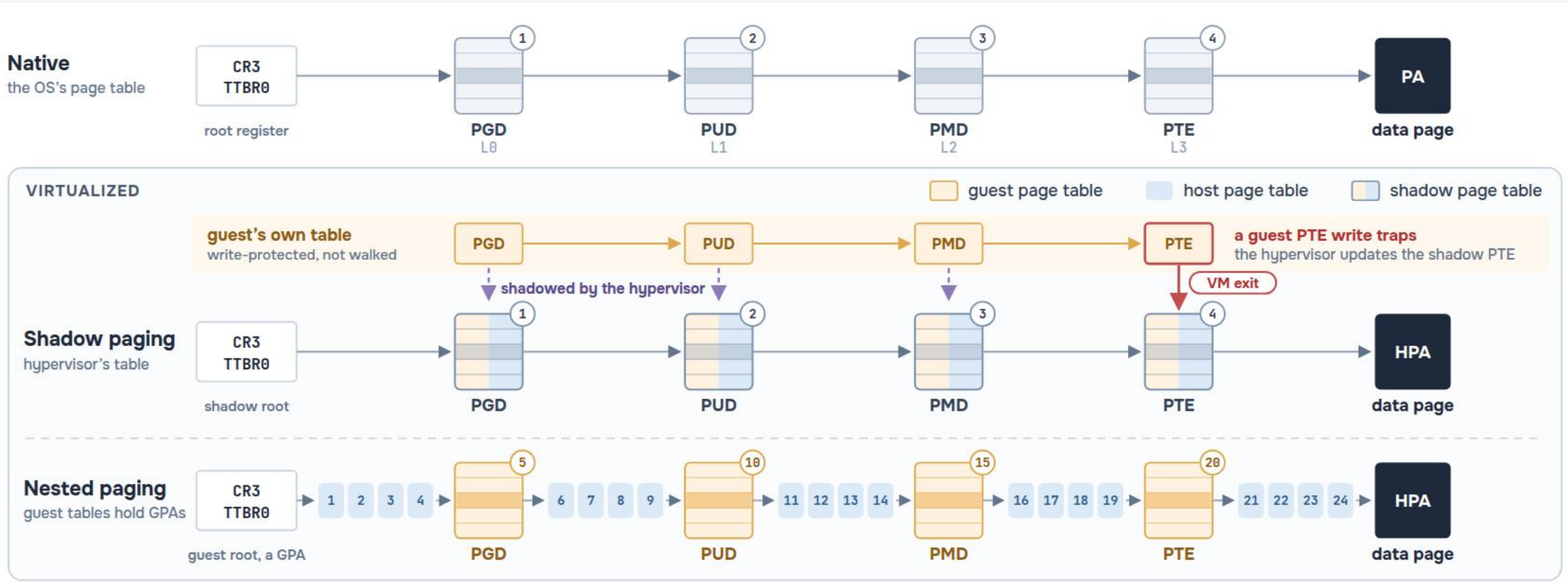


Amir Noohi amir.noohi@ed.ac.uk

Address translation in virtualization



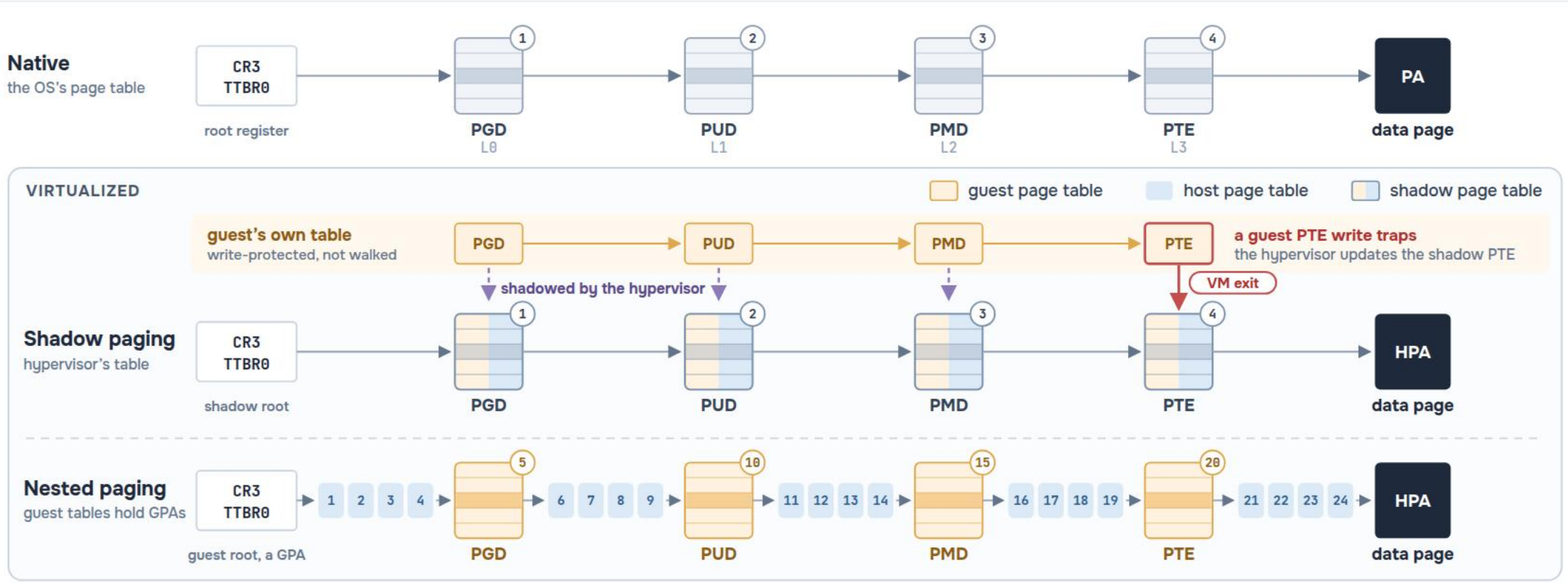
Address translation in virtualization



	Shadow paging the hypervisor merges both tables into one	Nested paging Intel EPT · AMD NPT · Arm Stage 2
TLB miss one page walk	4 memory accesses	24 memory accesses
Guest PTE update mmap, fork, page fault	**VM exit** on every update	**no VM exit**

Amir Noohi amir.noohi@ed.ac.uk

Address translation in virtualization

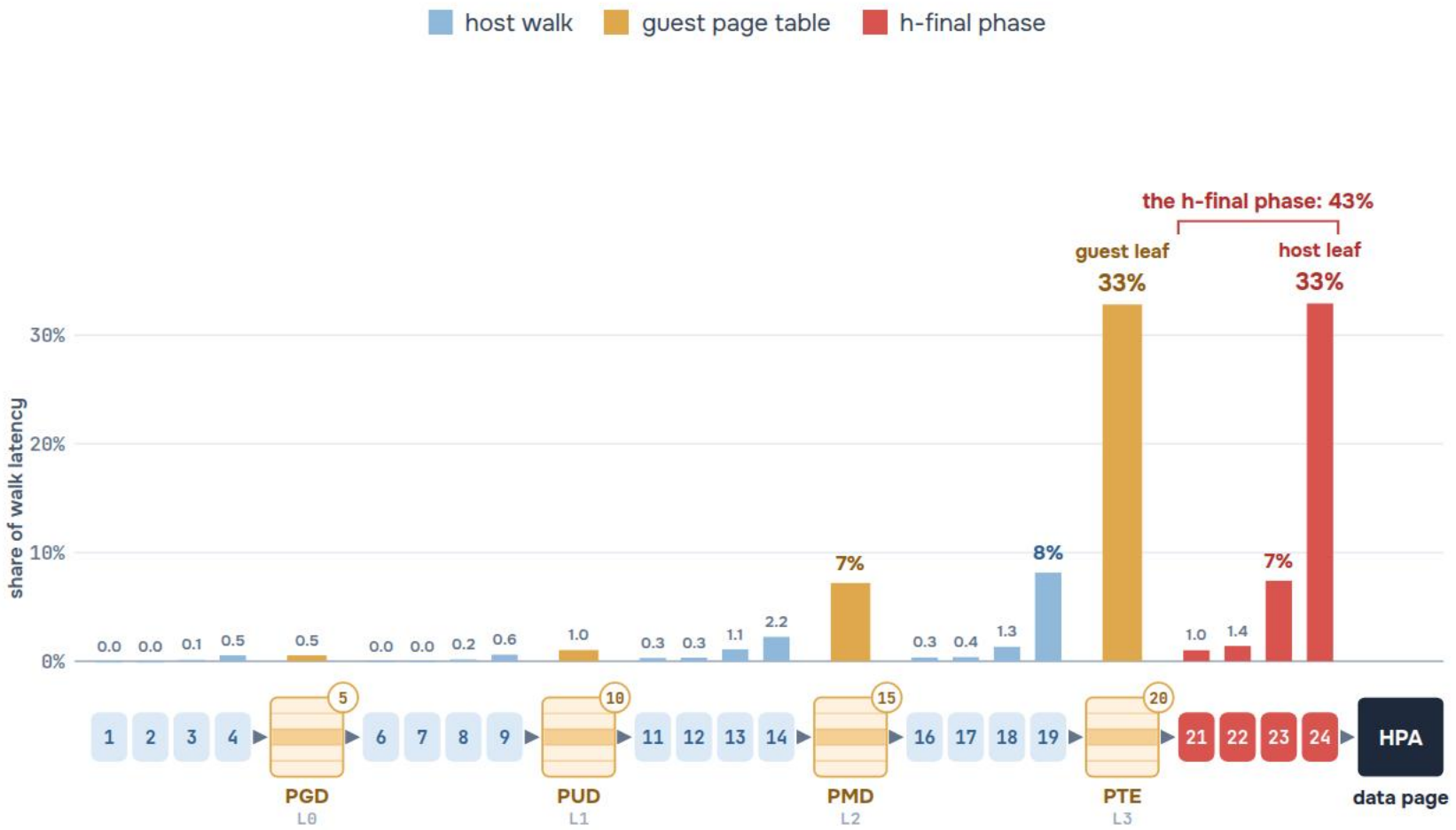


	Shadow paging the hypervisor merges both tables into one	THE GAP both at once?	Nested paging Intel EPT · AMD NPT · Arm Stage 2
TLB miss one page walk	4 memory accesses	fewer accesses	24 memory accesses
Guest PTE update mmap, fork, page fault	VM exit on every update	no VM exit	no VM exit

Amir Noohi amir.noohi@ed.ac.uk

Nested page walk latency

Where one nested walk spends its time



Workload

Application	Redis
Memory	155 GB, 2^{29} keys
Requests	uniform SETs, GETs
Pages	4 KB

Simulated system

L1 DTLB	64 entries
L2 TLB	1,536 entries
PWCs	2, 4, 32 entries
L1D	32 KiB
L2	1 MiB
LLC	22 MiB
Latency	PWC 1, L1 4, L2 14, LLC 54, DRAM 200 cycles

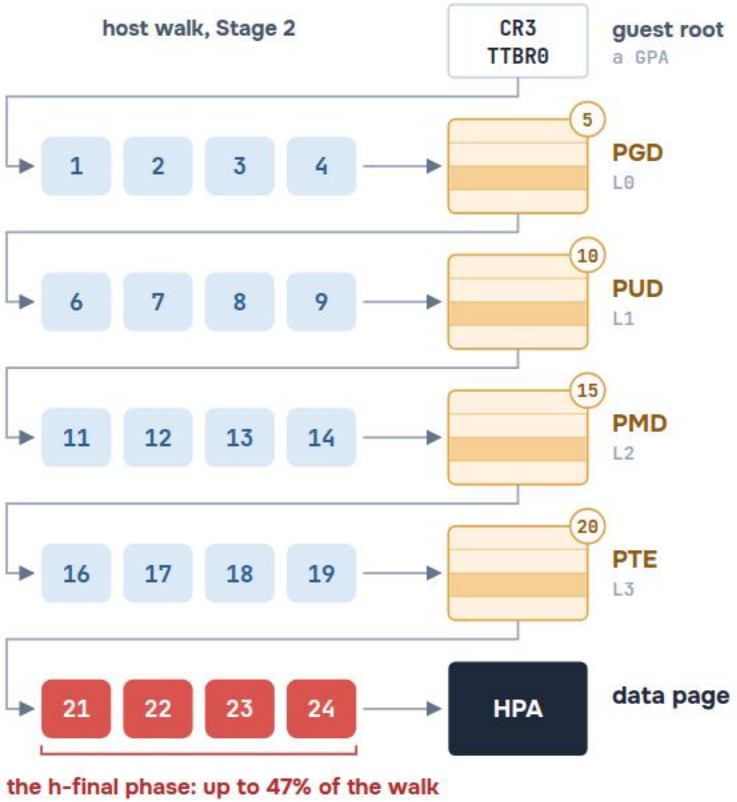
Amir Noohi amir.noohi@ed.ac.uk

Colocation of PTEs

host walk | guest page table | h-final phase

Nested paging today

24 dependent accesses



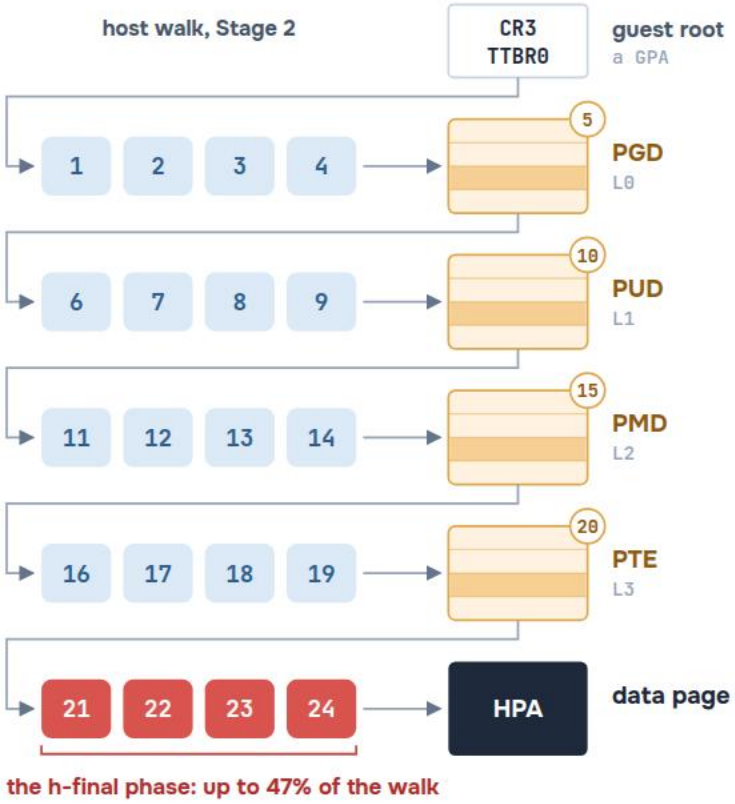
Amir Noohi amir.noohi@ed.ac.uk

Colocation of PTEs

host walk | guest page table | h-final phase

Nested paging today

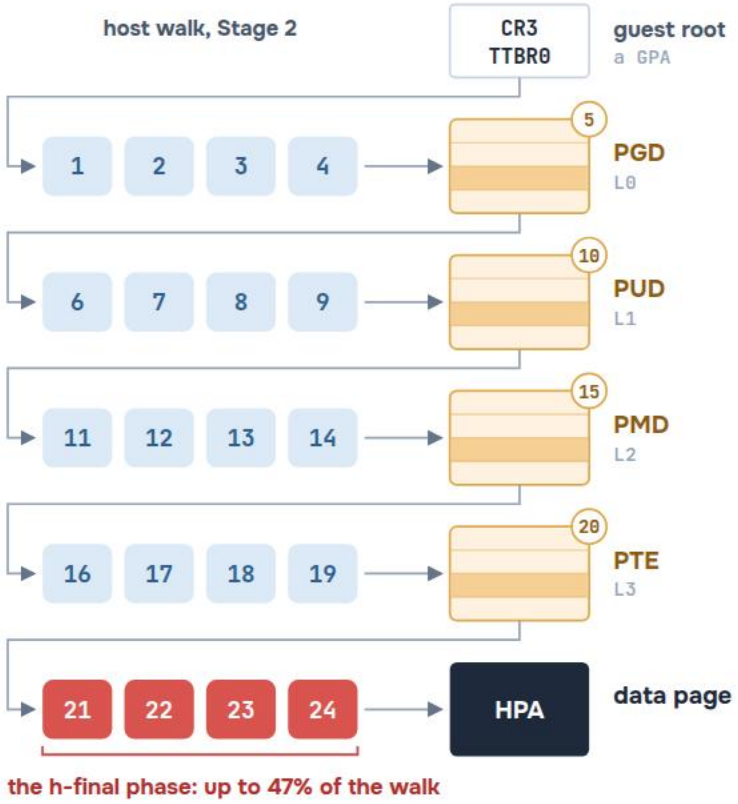
24 dependent accesses



the h-final phase: up to 47% of the walk

Colocated

the host PTE sits next to the guest PTE



the h-final phase: up to 47% of the walk

Amir Noohi amir.noohi@ed.ac.uk

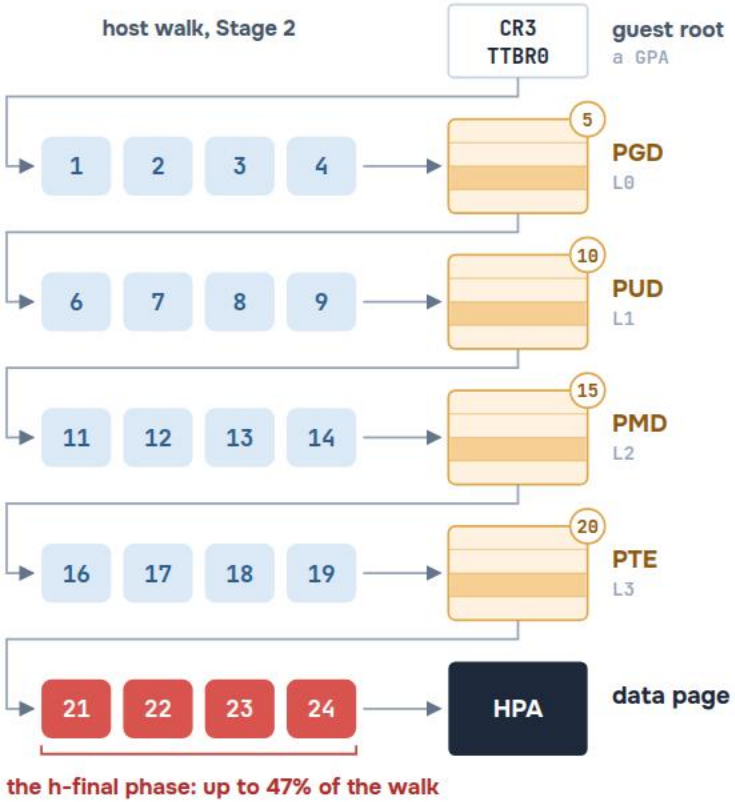
IDEA

Colocation of PTEs

host walk | guest page table | h-final phase

Nested paging today

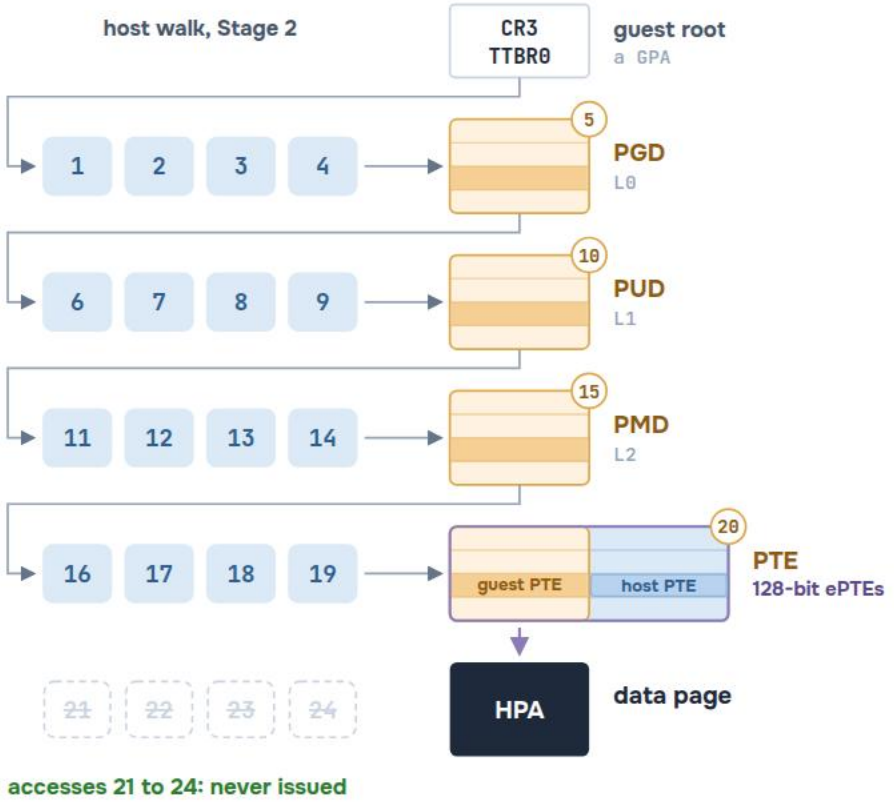
24 dependent accesses



the h-final phase: up to 47% of the walk

Colocated

the host PTE sits next to the guest PTE



accesses 21 to 24: never issued

NO NEW ATOMICS

two 16-byte operations that cores already have

128-bit atomic read

returns both PTEs in one access

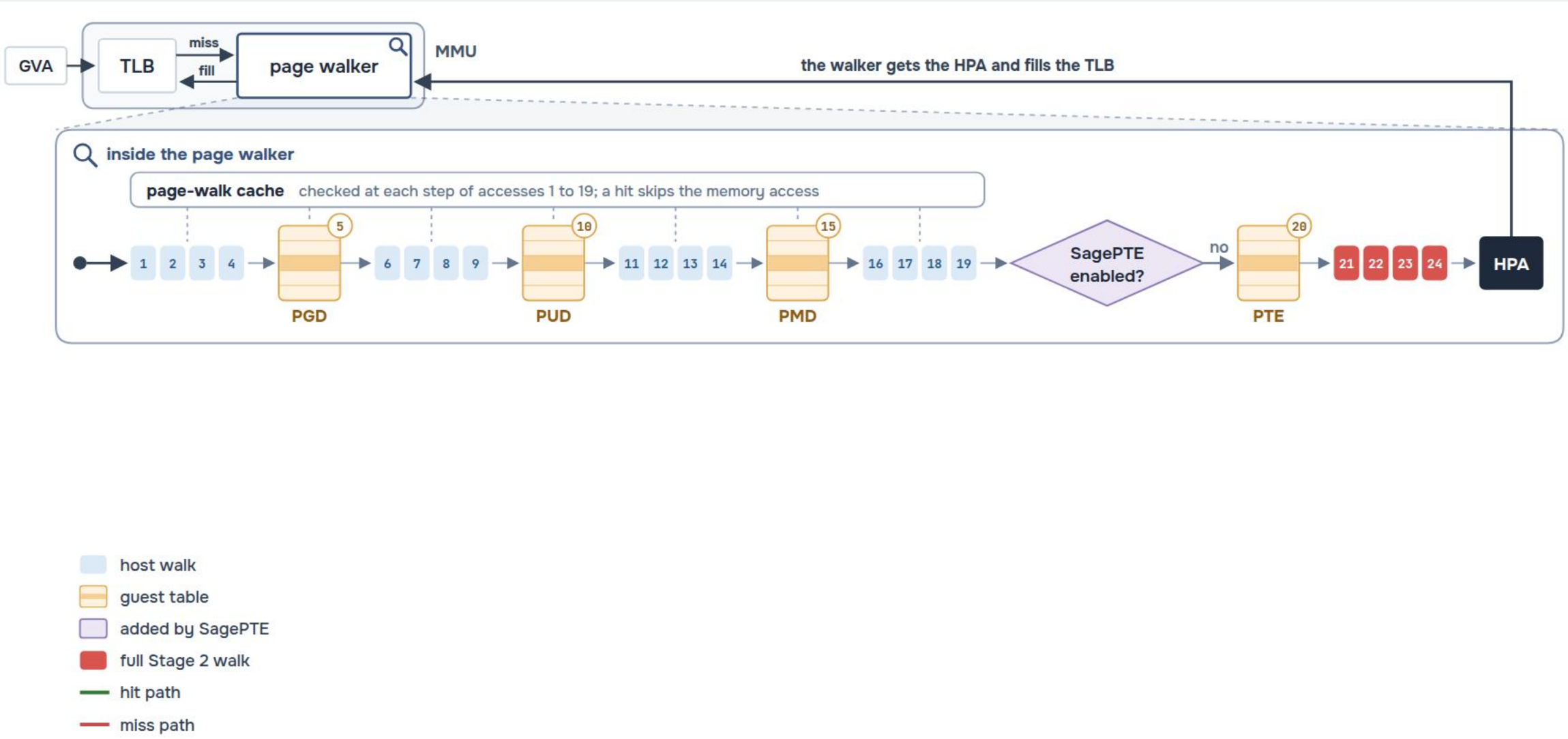
128-bit compare-and-swap

fills the host PTE if the guest PTE is unchanged

Arm	`FEAT_LSE128`
x86	`CMPXCHG16B`

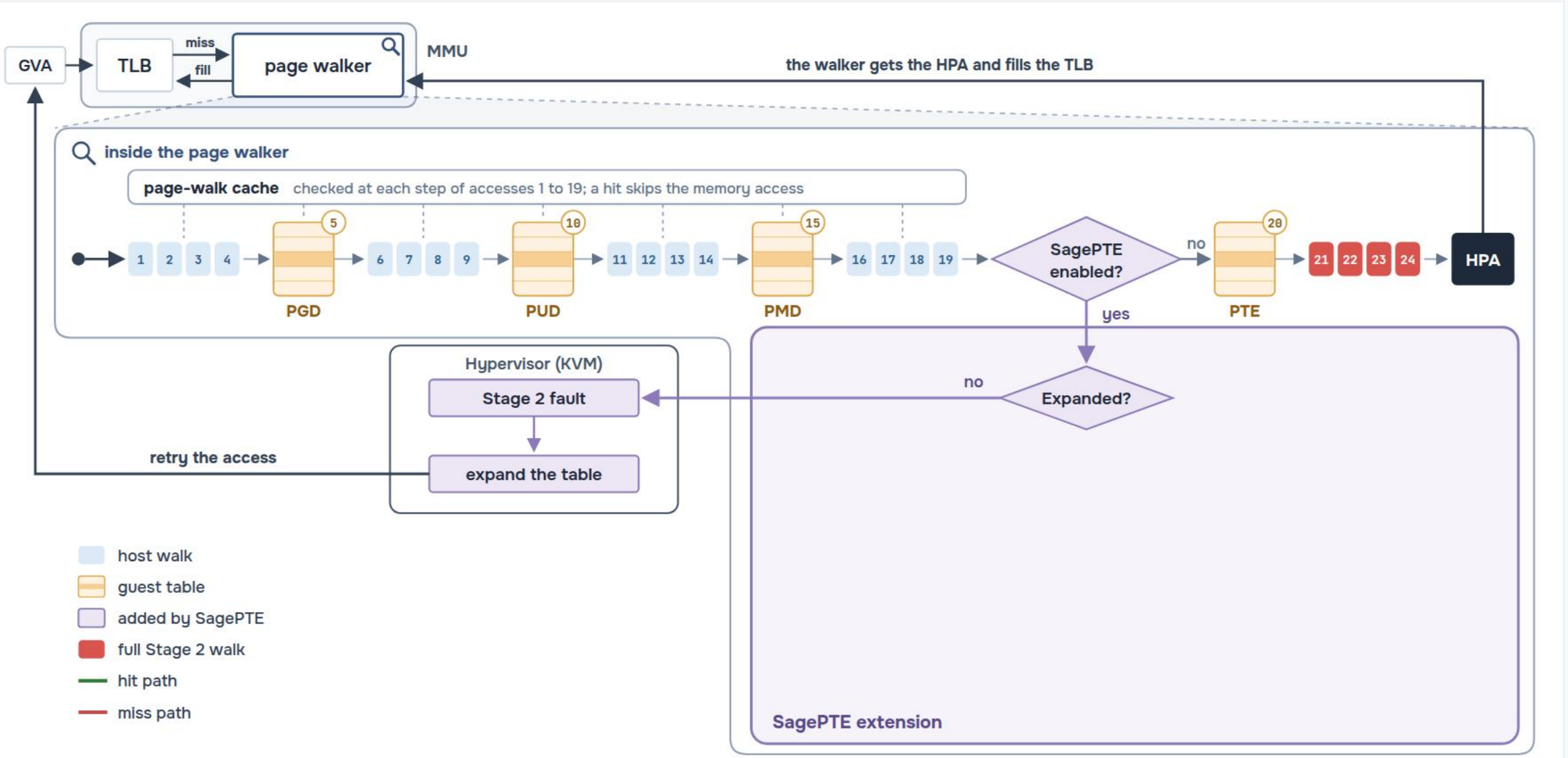
Amir Noohi amir.noohi@ed.ac.uk

SagePTE in action



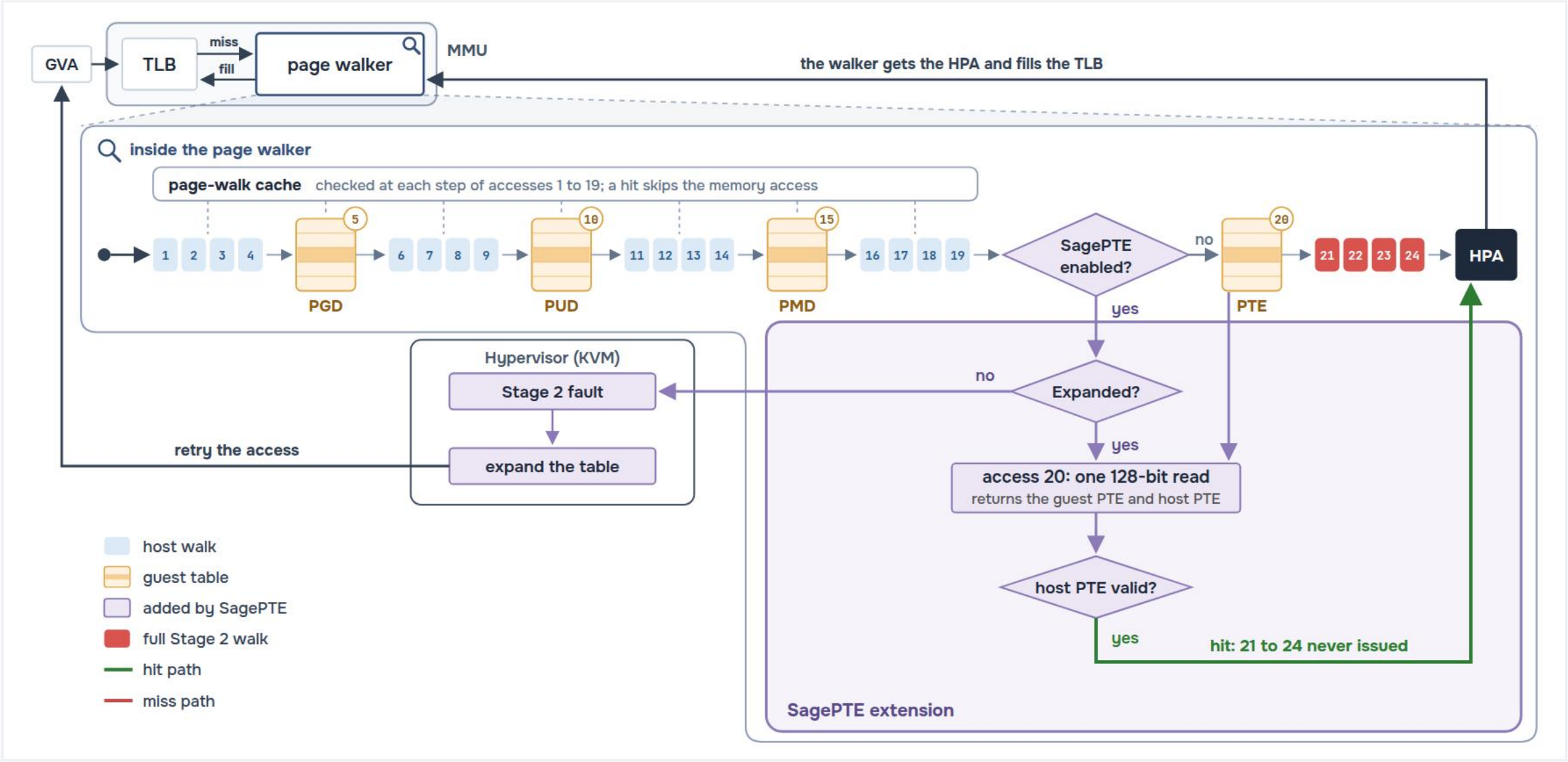
Amir Noohi amir.noohi@ed.ac.uk

SagePTE in action



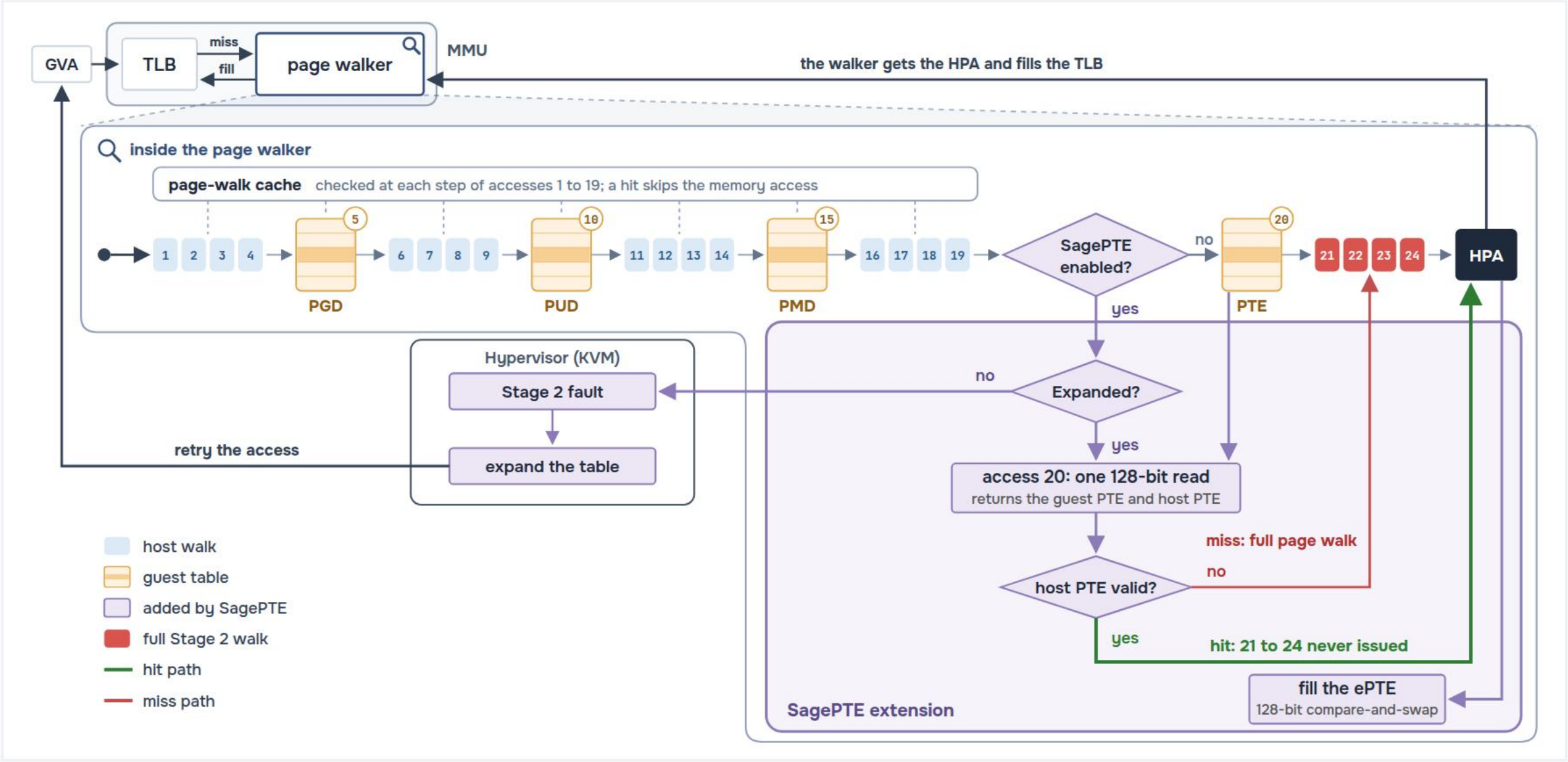
Amir Noohi amir.noohi@ed.ac.uk

SagePTE in action



Amir Noohi amir.noohi@ed.ac.uk

SagePTE in action

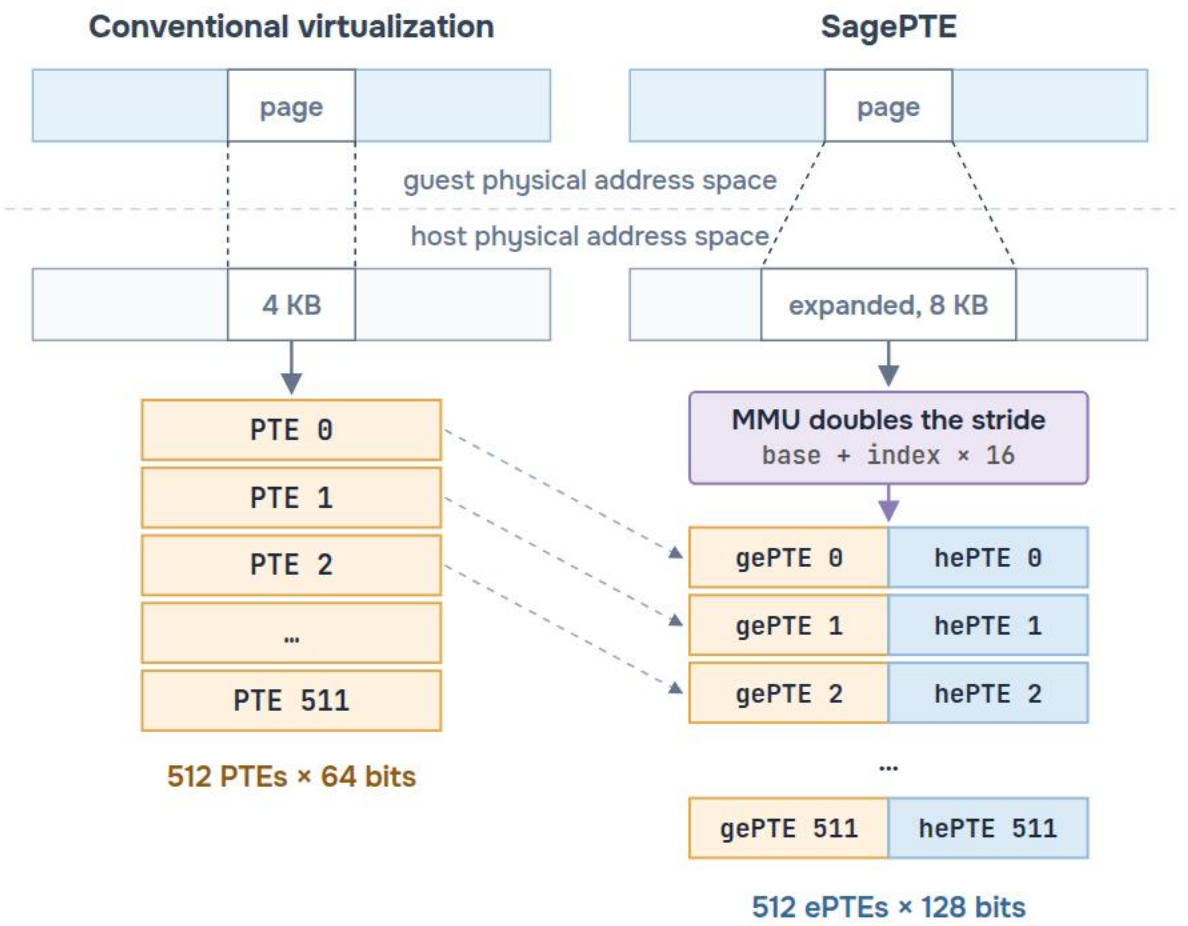


Amir Noohi amir.noohi@ed.ac.uk

The extended PTE

Layout of a guest leaf table

what the guest sees, what memory holds

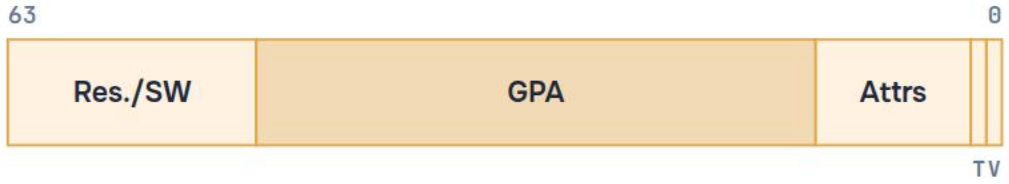


The ePTE format

128 bits



hePTE: the host's half, hidden from the guest



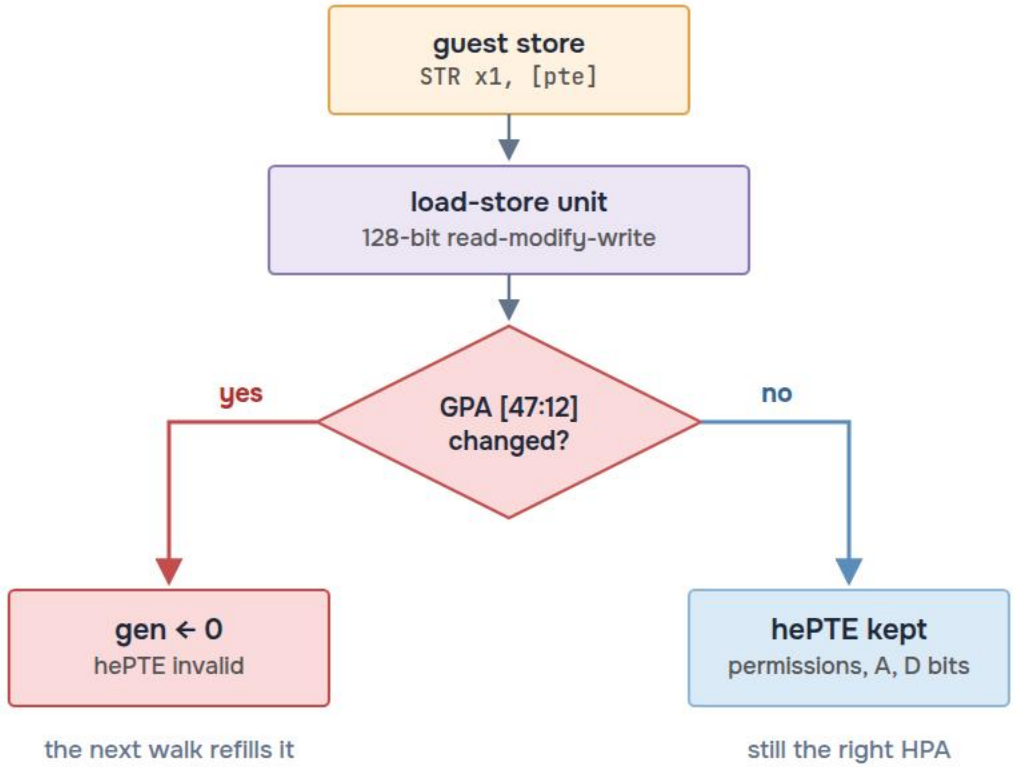
gePTE: the guest's PTE, standard Armv8 format

`Gen`	generation, 0 = invalid	`Res./SW`	reserved, software bits
`Res.`	reserved, PA [51:48]	`GPA`	guest physical [47:12]
`Cached HPA`	Stage 2 output [47:12]	`Attrs`	permissions, attributes
`Attr.`	cached host attributes	`T, V`	descriptor type, valid

Amir Noohi amir.noohi@ed.ac.uk

Invalidation

Guest write to a PTE — the hardware compares the old GPA with the new one



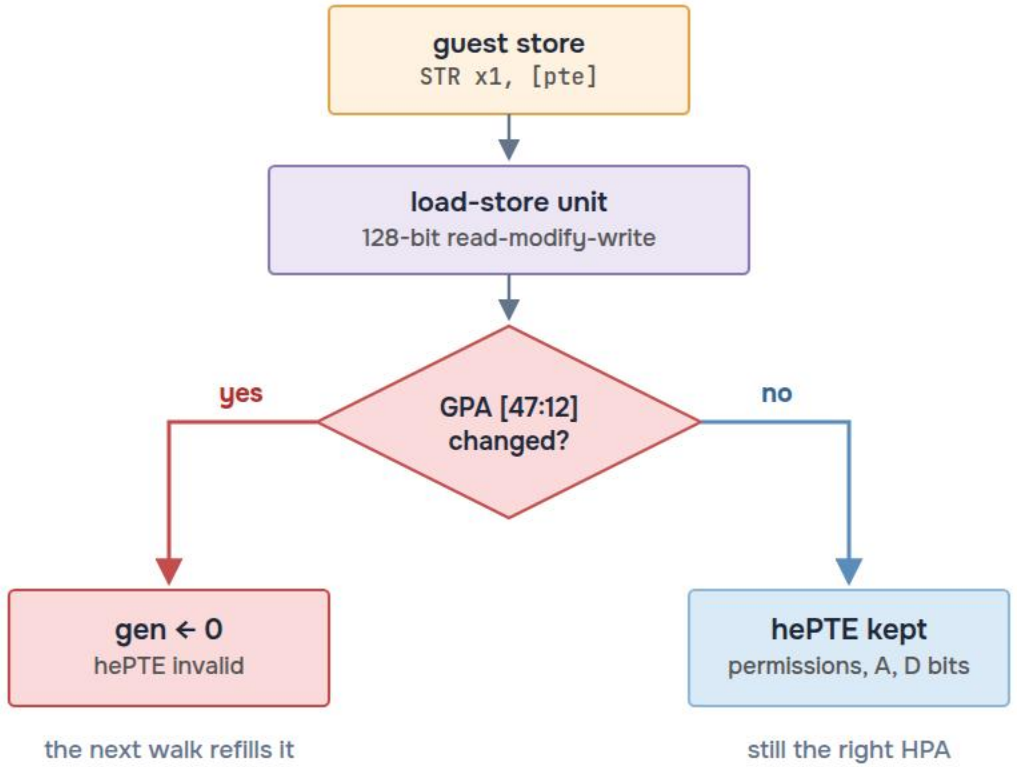
no VM exit, no trap: the store completes in the load-store unit

Amir Noohi amir.noohi@ed.ac.uk

Invalidation

Guest write to a PTE

the hardware compares the old GPA with the new one



no VM exit, no trap: the store completes in the load-store unit

Host-wide invalidation

page migration, KSM, remaps

host epoch
6 → 7

`ACTLR_EL1[24:21]`, written by the hypervisor

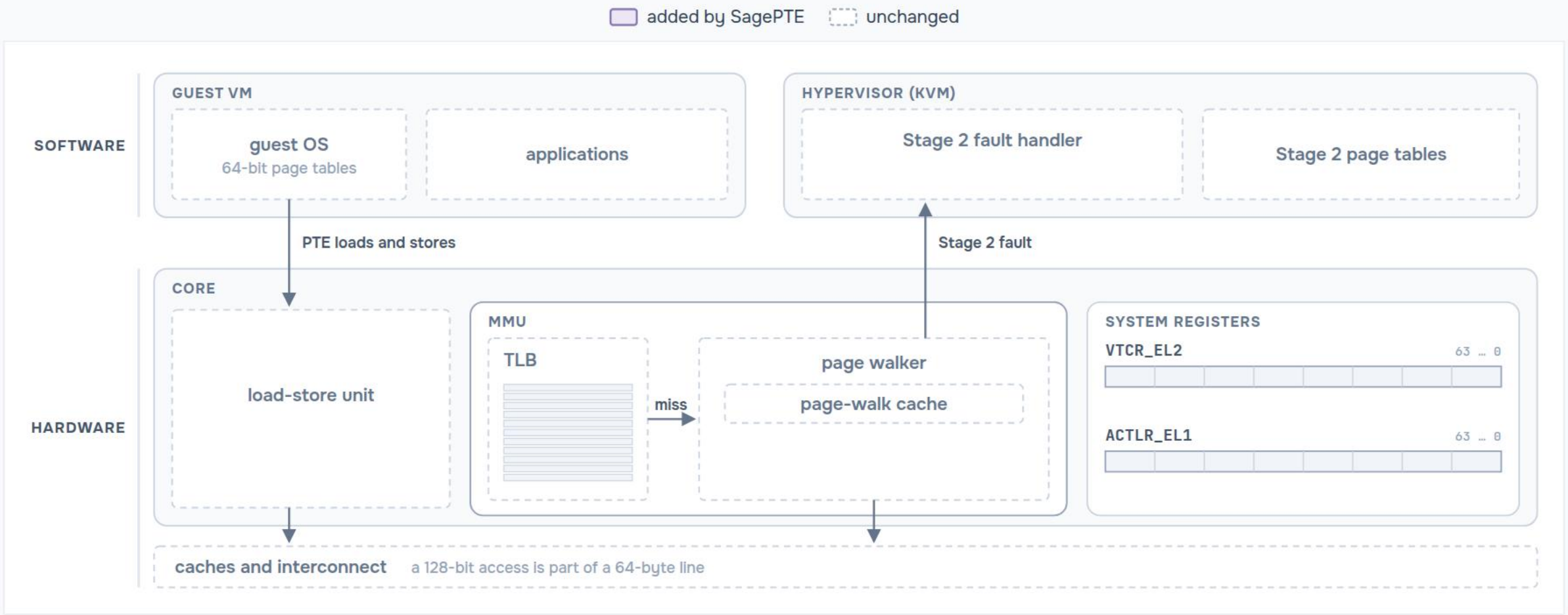
a leaf table in memory

gePTE	HPA, gen 6	stale: 6 ≠ 7
gePTE	HPA, gen 6	stale: 6 ≠ 7
gePTE	HPA, gen 7	valid, refilled after the bump
gePTE	HPA, gen 0	invalid, after a guest write

no scan: the walker checks each hePTE's generation against the epoch when it reads it, and refills stale ones lazily

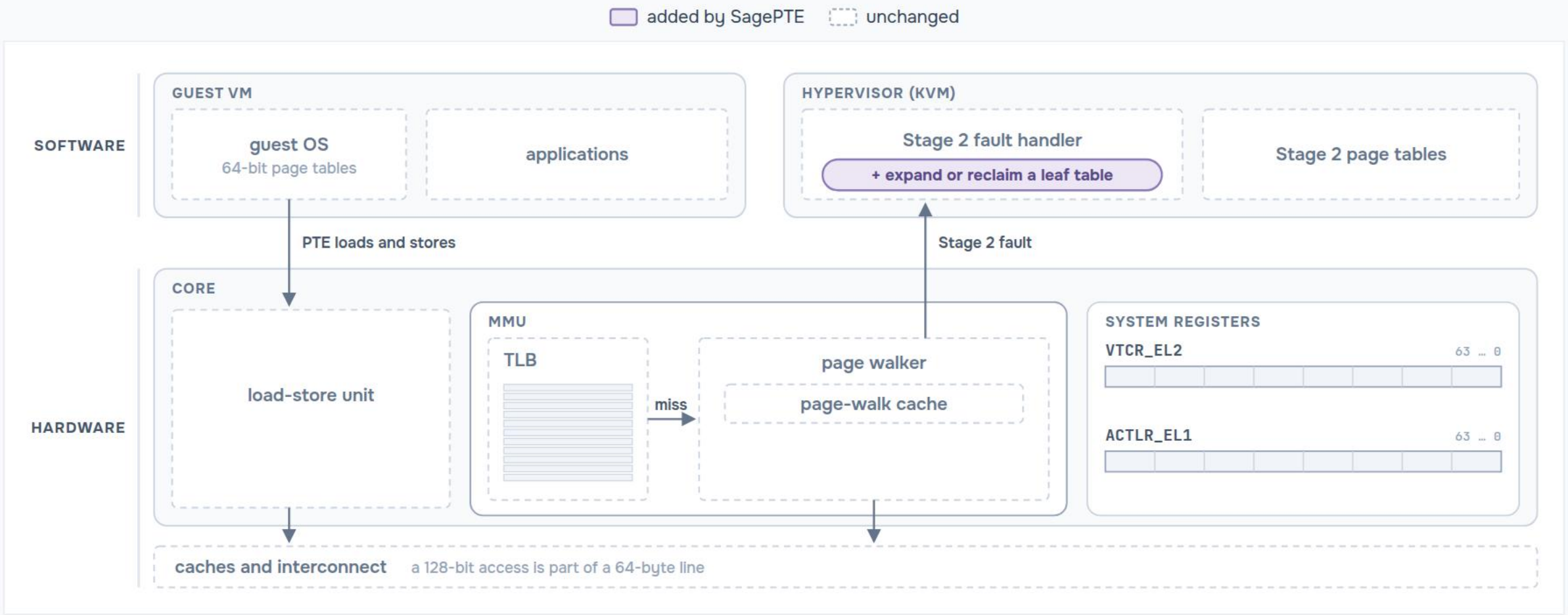
Amir Noohi amir.noohi@ed.ac.uk

Requirements



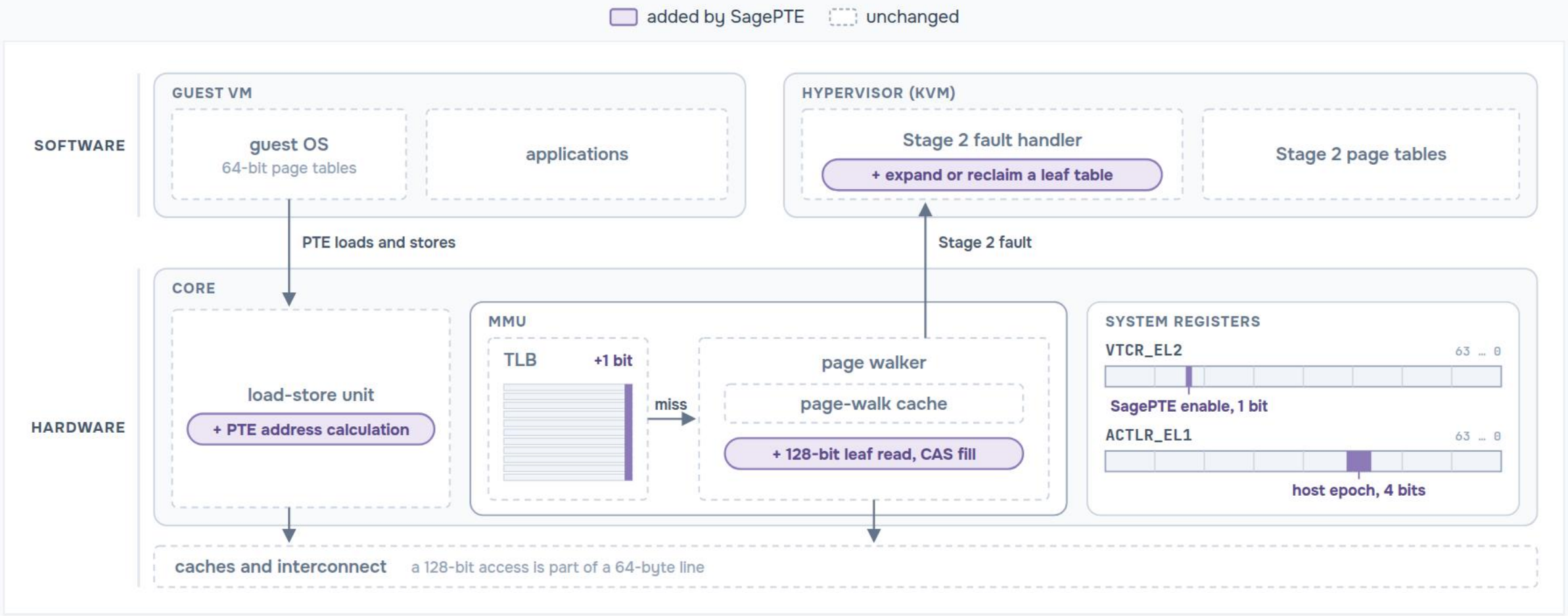
Amir Noohi amir.noohi@ed.ac.uk

Requirements



Amir Noohi amir.noohi@ed.ac.uk

Requirements



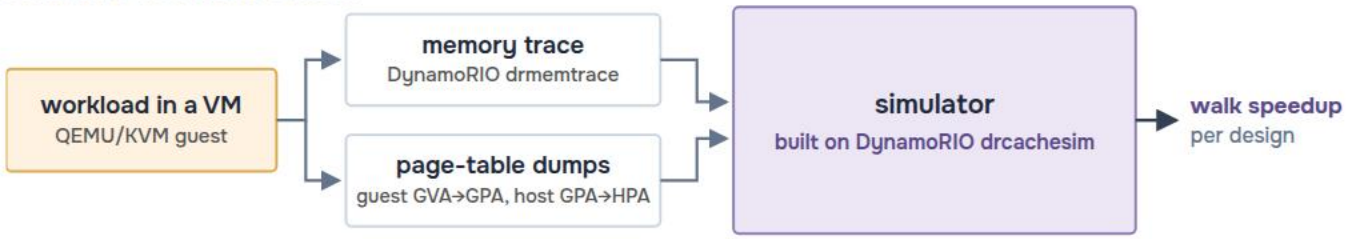
Amir Noohi amir.noohi@ed.ac.uk

Methodology

Measure the walk time, simulate the walk speedup

as in ASAP and DMT

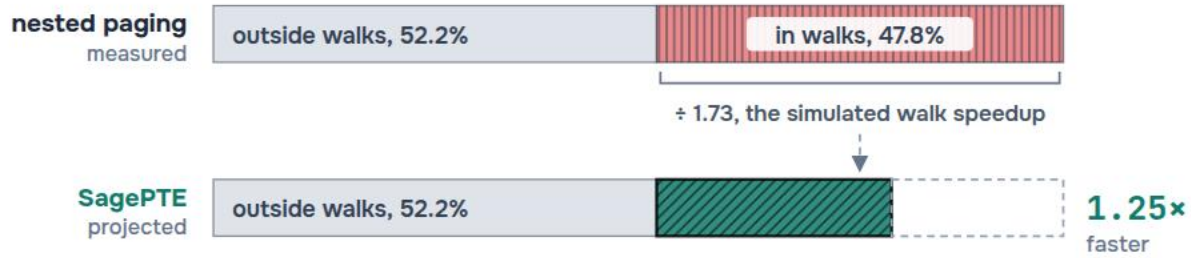
SIMULATED FOR EVERY DESIGN



MEASURED ON REAL HARDWARE



REDIS, 4 KB PAGES



Simulated system

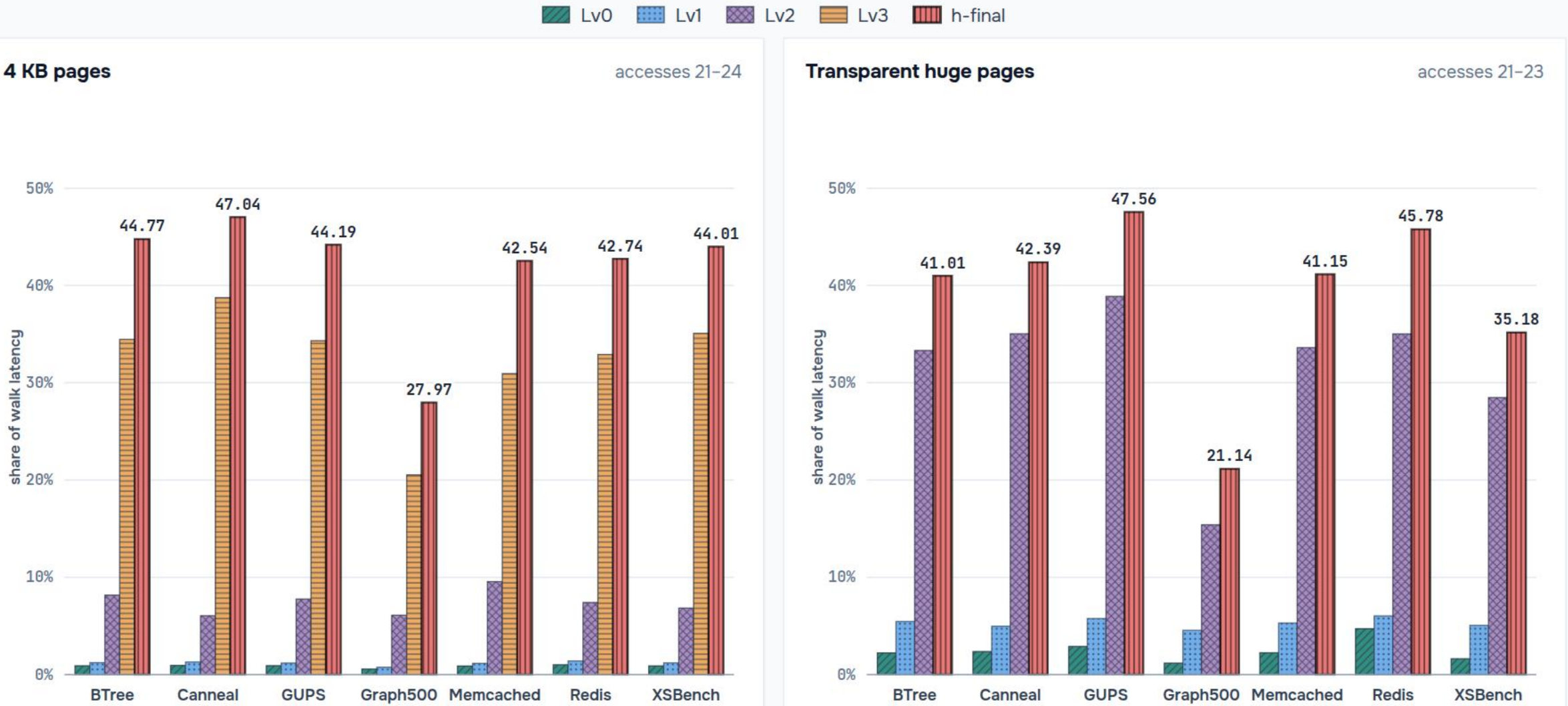
as in the Ampere Altra Max M96-30

Core	Arm Neoverse N1, 96 cores
L1 TLBs (I, D)	48 entries, fully associative
L2 TLB	1,280 entries, 5-way
Guest, host PWCs	2, 4, 32 entries for L0, L1, L2
L1I, L1D	64 KiB, 4-way
L2	1 MiB, 8-way
System-level cache	16 MiB, 16-way, shared
Cache line	64 B
Latency, cycles	PWC 1, L1 4, L2 14, SLC 54, DRAM 200
Pages	4 KB, or THP in guest and host
Trace	2 B references, 300 M warm-up

functional prototype: QEMU, Linux and KVM

Amir Noohi amir.noohi@ed.ac.uk

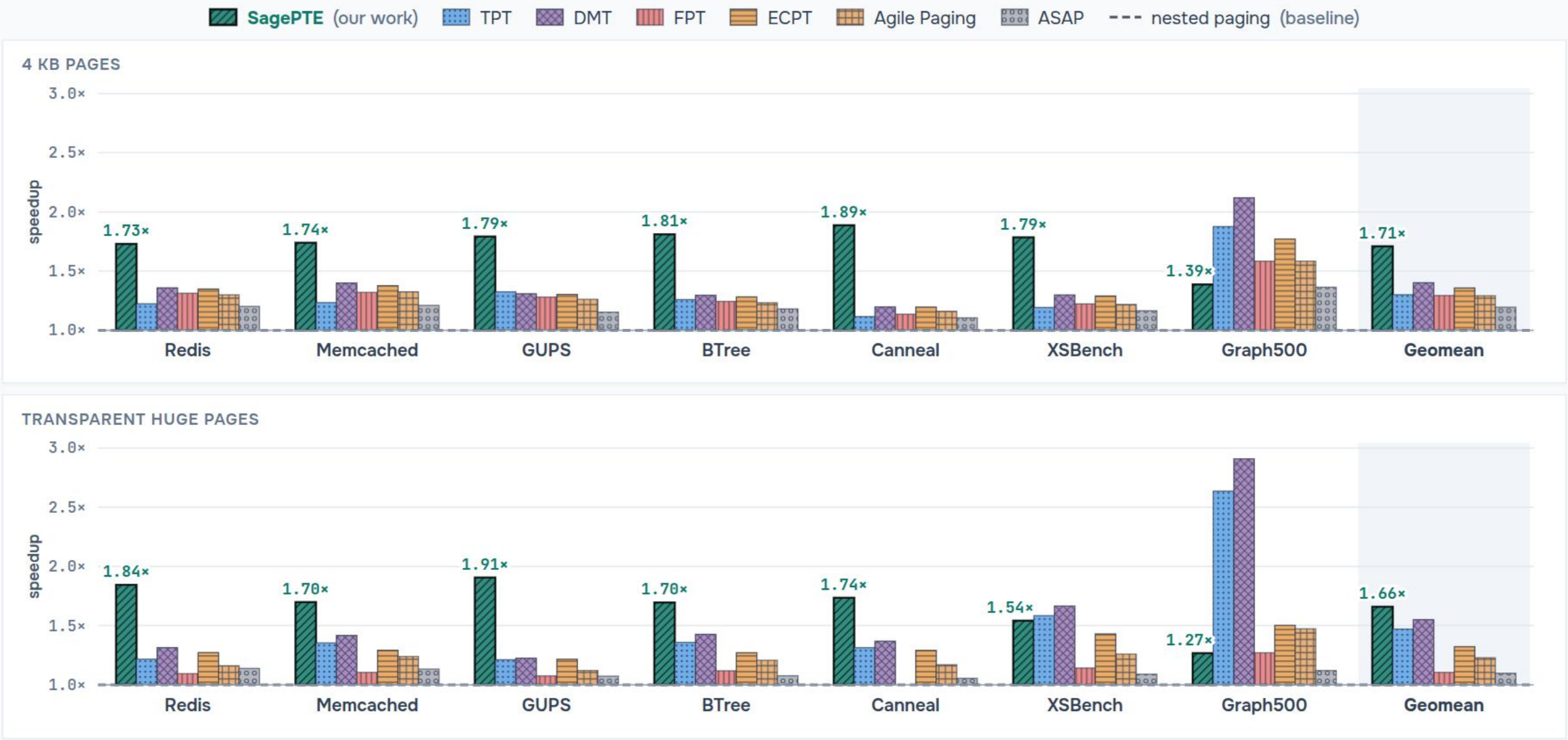
Last-level PTE latency



Share of nested page-walk latency spent in each host level of the h-final phase (Lv0 to Lv3), and in the whole phase. With huge pages the h-final walk resolves the 2 MB data page in three accesses, Lv0 to Lv2.

Amir Noohi amir.noohi@ed.ac.uk

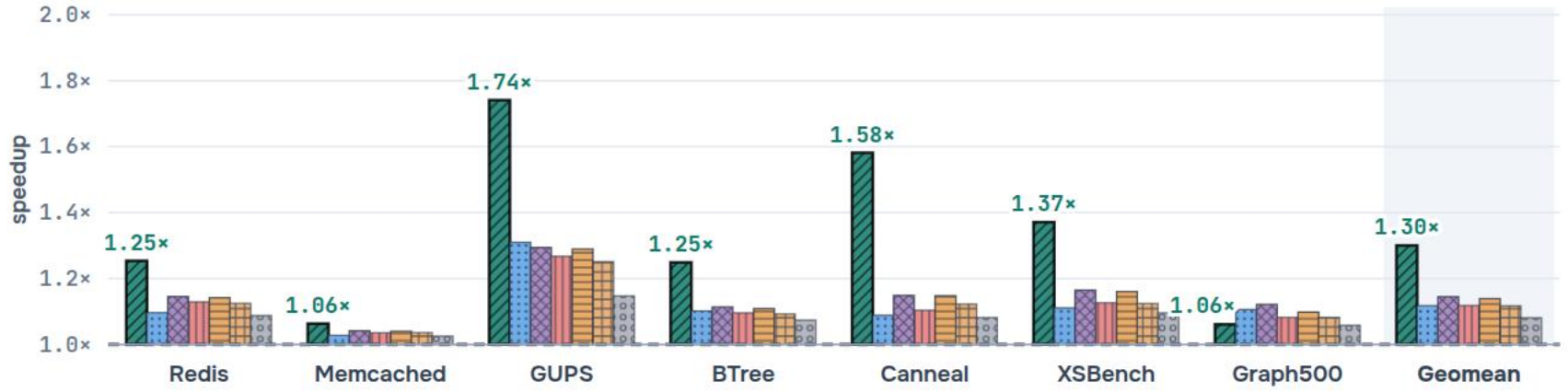
Page-walk speedup

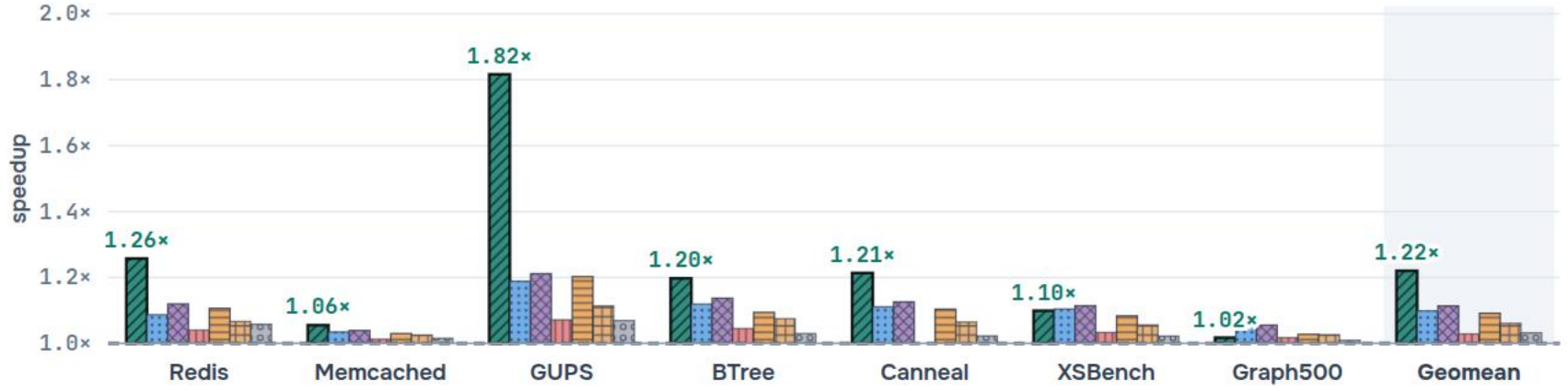


Geomean speedup: **1.71×** with 4 KB pages and **1.66×** with huge pages; DMT, the best prior design, reaches 1.40× and 1.55×. SagePTE trails where the leaf is warm: Graph500, and XSBench with huge pages.

End-to-end speedup

SagePTE (our work) TPT DMT FPT ECPT Agile Paging ASAP --- nested paging (baseline)





TIME IN PAGE WALKS	4 KB	THP
Redis	47.8%	44.7%
Memcached	13.9%	12.7%
GUPS	96.3%	94.5%
BTree	44.5%	40.1%
Canneal	78.2%	41.5%
XSBench	61.4%	25.6%
Graph500	20.4%	7.9%

SAGEPTE OVER	4 KB	THP
TPT	1.17×	1.11×
DMT	1.15×	1.09×
FPT	1.17×	1.18×
ECPT	1.15×	1.11×
Agile Paging	1.17×	1.15×
ASAP	1.21×	1.18×

Geomean speedup: **1.30×** with 4 KB pages and **1.22×** with huge pages. The gain follows the time each workload spends in page walks.

Amir Noohi amir.noohi@ed.ac.uk

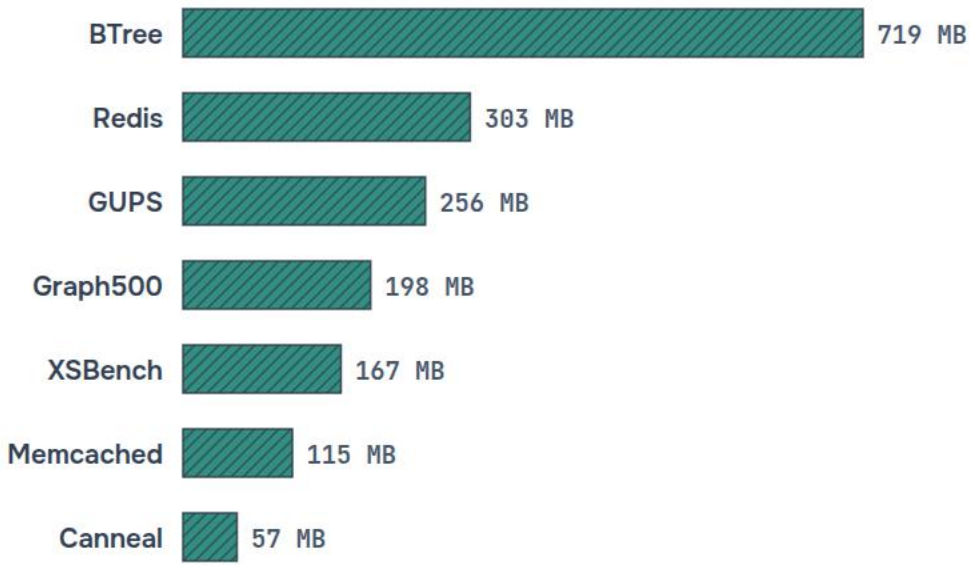
Overheads

Memory overhead

only the leaf page tables double, 4 KB → 8 KB

< 0.27% of VM memory

GUEST PAGE TABLES PER WORKLOAD



Expansion latency

one-time, per leaf table: 3.01 µs

< 0.12% of execution



EXPANSIONS PER RUN

WORKLOAD, 4 KB PAGES	EXPANSIONS	TIME, MS	OF RUNTIME	THP
BTree	183,835	553.5	0.089%	361
Redis	77,657	233.8	0.117%	158
GUPS	65,554	197.4	0.070%	132
Graph500	50,720	152.7	0.060%	111
XSBench	42,724	128.2	0.012%	87
Memcached	29,561	89.0	0.008%	58
Canneal	14,581	43.9	0.002%	31

THP: expansions with huge pages. Expansions trap to the hypervisor; guest PTE updates never do.

Amir Noohi amir.noohi@ed.ac.uk

Takeaways

1 The cost is concentrated at the leaf of the page walk



accesses 21 to 24

the h-final phase: up to 47% of page-walk latency

2 Colocate the host and guest PTEs

gePTE	hePTE
guest, 64 bits	host, 64 bits

one 128-bit ePTE, one read

no guest OS changes, no VM exits, the 128-bit atomics cores already have

3 Faster walks, efficient virtualization

	4 KB PAGES	HUGE PAGES
page walks	1.71×	1.66×
end to end	1.30×	1.22×

geomean speedup over nested paging

Thank you.

Questions?

ARTIFACT

`github.com/amirmnoohi/SagePTE-AE`

`doi:10.5281/zenodo.21876937`

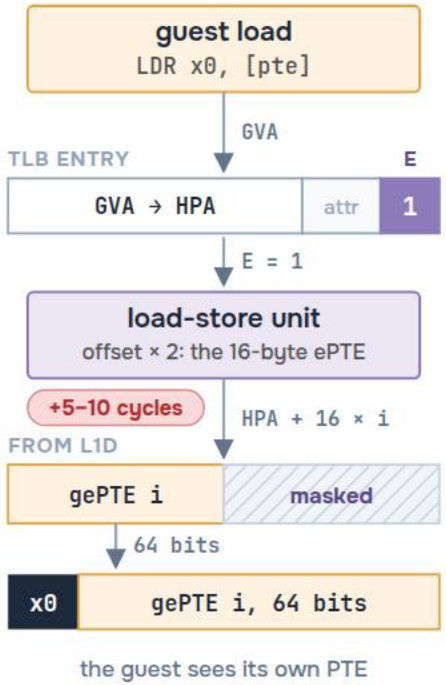
Available · Reviewed · Reproducible

Amir Noohi `amir.noohi@ed.ac.uk`

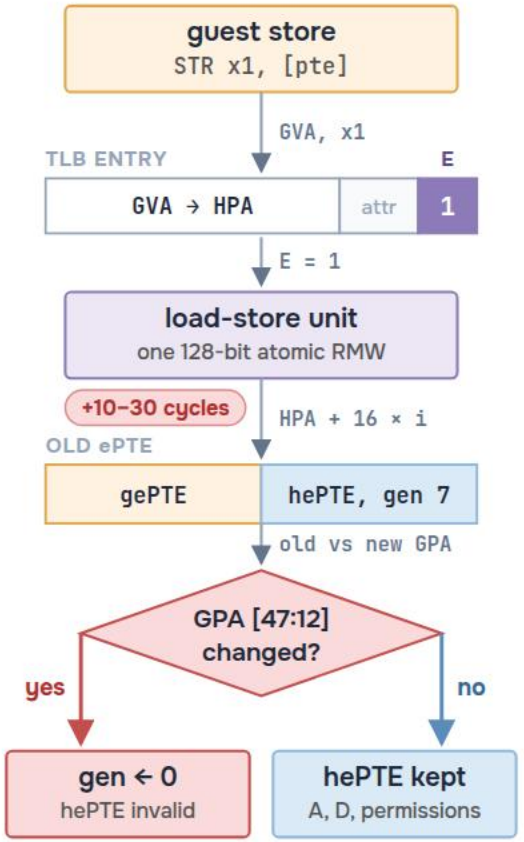
Guest and host accesses

Guest accesses to its own PTEs

steered by one TLB bit, E



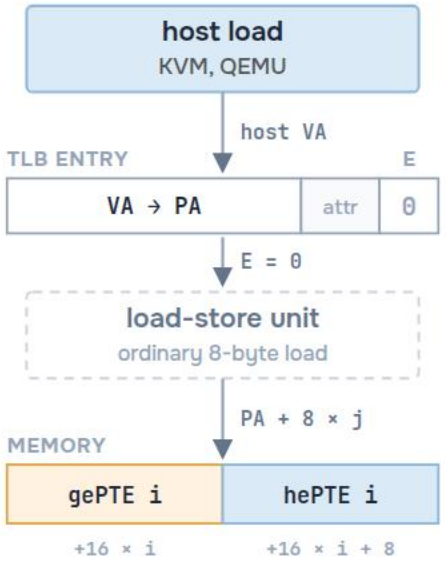
the guest sees its own PTE



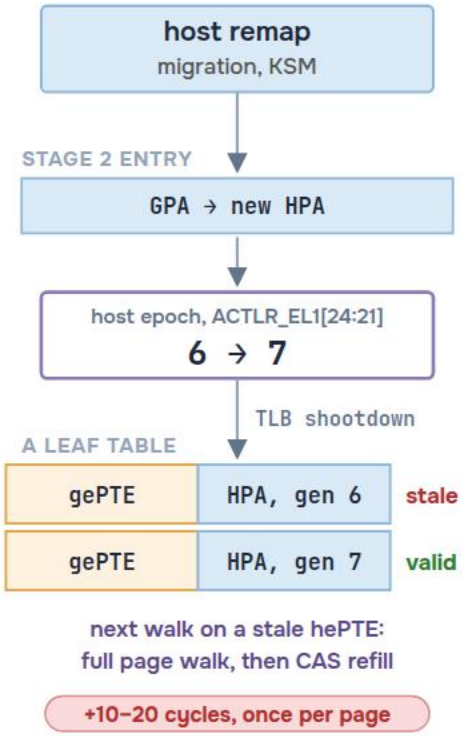
no VM exit, no trap: both complete in the load-store unit

Host accesses

its own mappings never set E



hardware unchanged
software sees both halves,
16 bytes apart



next walk on a stale hePTE:
full page walk, then CAS refill

+10–20 cycles, once per page

no scan: each hePTE is checked against the epoch when it is read

Cycles are estimates for an N1-class core; SagePTE was not synthesized. Data loads and stores (E = 0) add nothing; a page walk adds about 1 cycle.

Amir Noohi amir.noohi@ed.ac.uk

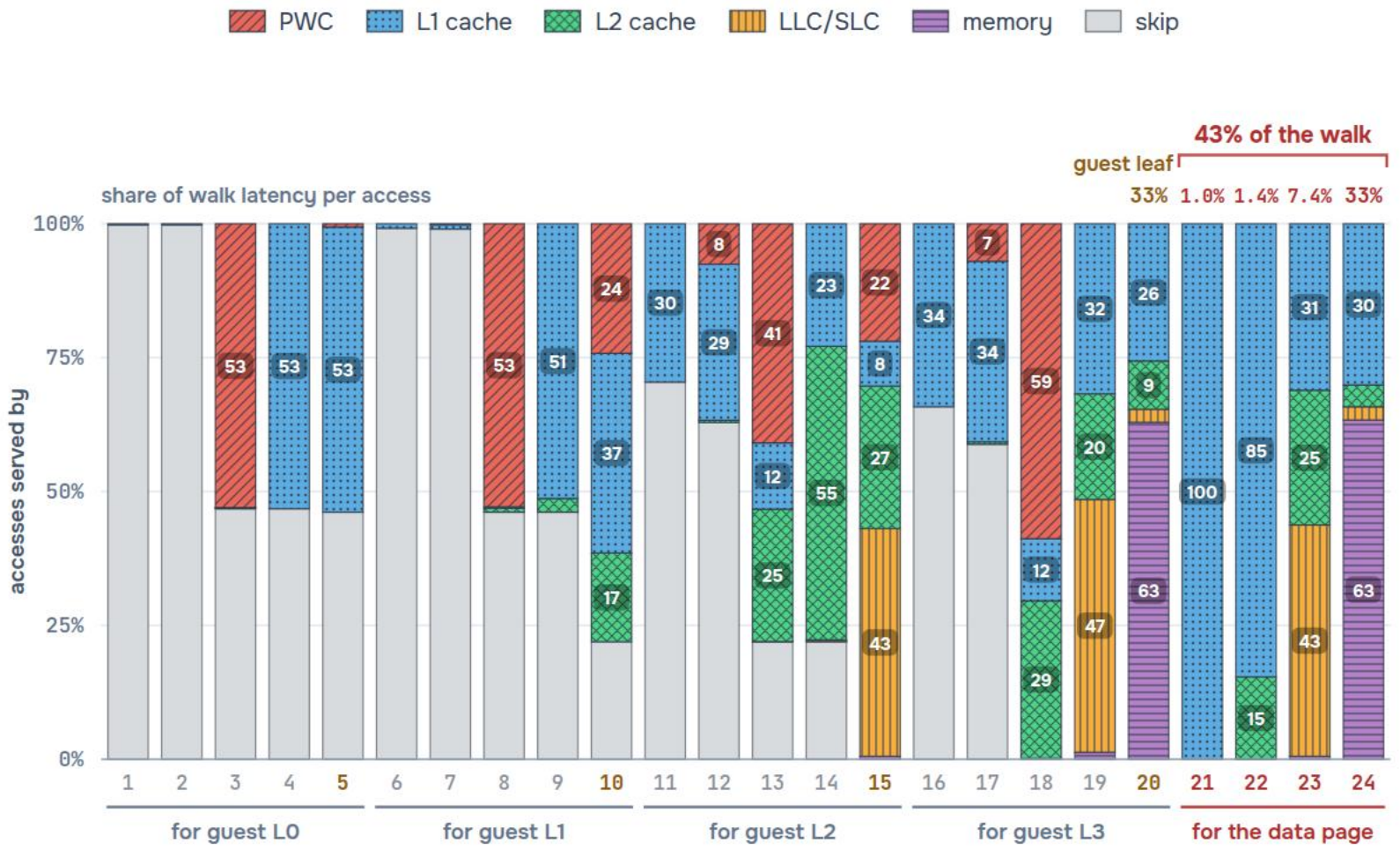
What each design removes



Each design's accesses as the evaluation's simulator models them, 4 KB pages. ASAP issues all 24 and prefetches the two deepest levels of both dimensions (its Figure 7). Walk speedup: geomean over the seven workloads.

Amir Noohi amir.noohi@ed.ac.uk

Which level serves each access

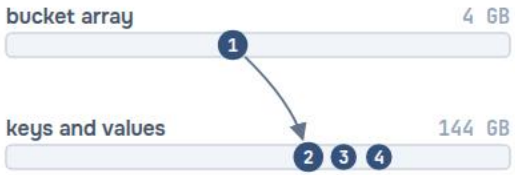


Amir Noohi amir.noohi@ed.ac.uk

Memory access patterns

Redis 155 GB — 48%

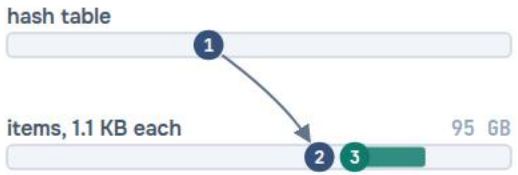
Random lookup, then a short chase



one GET: a random bucket, then the entry, its key and its value, side by side

Memcached 95 GB — 14%

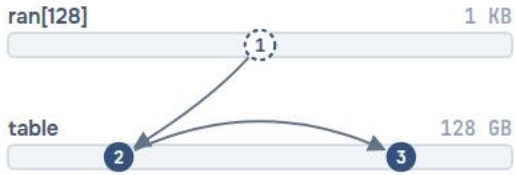
Random item, short sequential read



one GET: a random bucket, then one item: its key, then its value in order

GUPS 128 GB — 96%

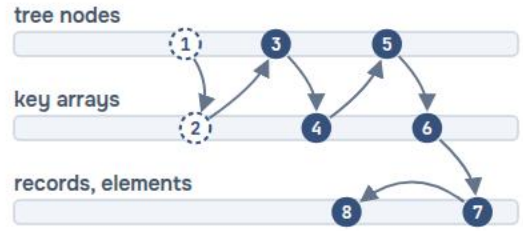
Random updates, no locality



one update: ran[j], then table[x] and its mirror, table[N - x]

BTree 125 GB — 44%

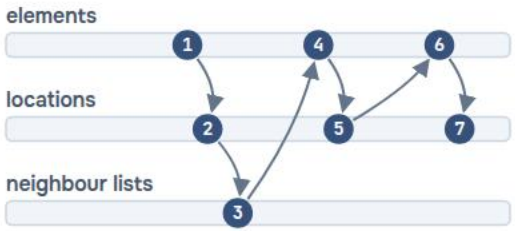
Pointer chase down the tree



one lookup: about 10 nodes, each then its keys; the top levels stay cached

Canneal 62 GB — 78%

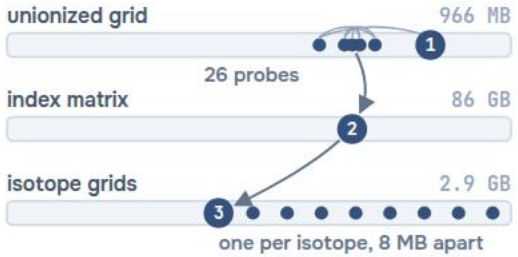
Random element, two hops each



one move: an element, its location and its neighbours, then two hops per neighbour

XSBench 84 GB — 61%

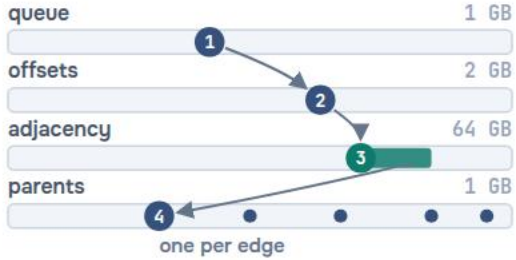
Binary search, then strided reads



one lookup: a binary search, one row of the index, then one point in each isotope's grid

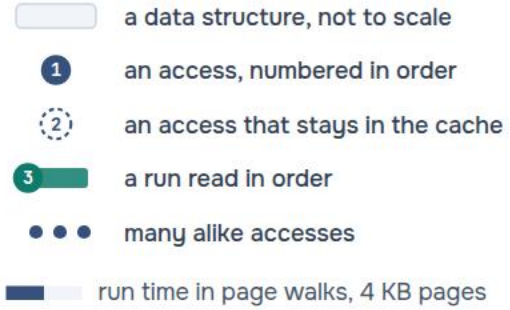
Graph500 123 GB — 20%

Sequential scans, random checks



one vertex: from the queue, its offsets and its edges in order, then each neighbour's parent

How to read



One operation of each workload, as its source code does it at the inputs it runs with: its main data structures, and its accesses in order.

Amir Noohi amir.noohi@ed.ac.uk