香港科技大学（广州）
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

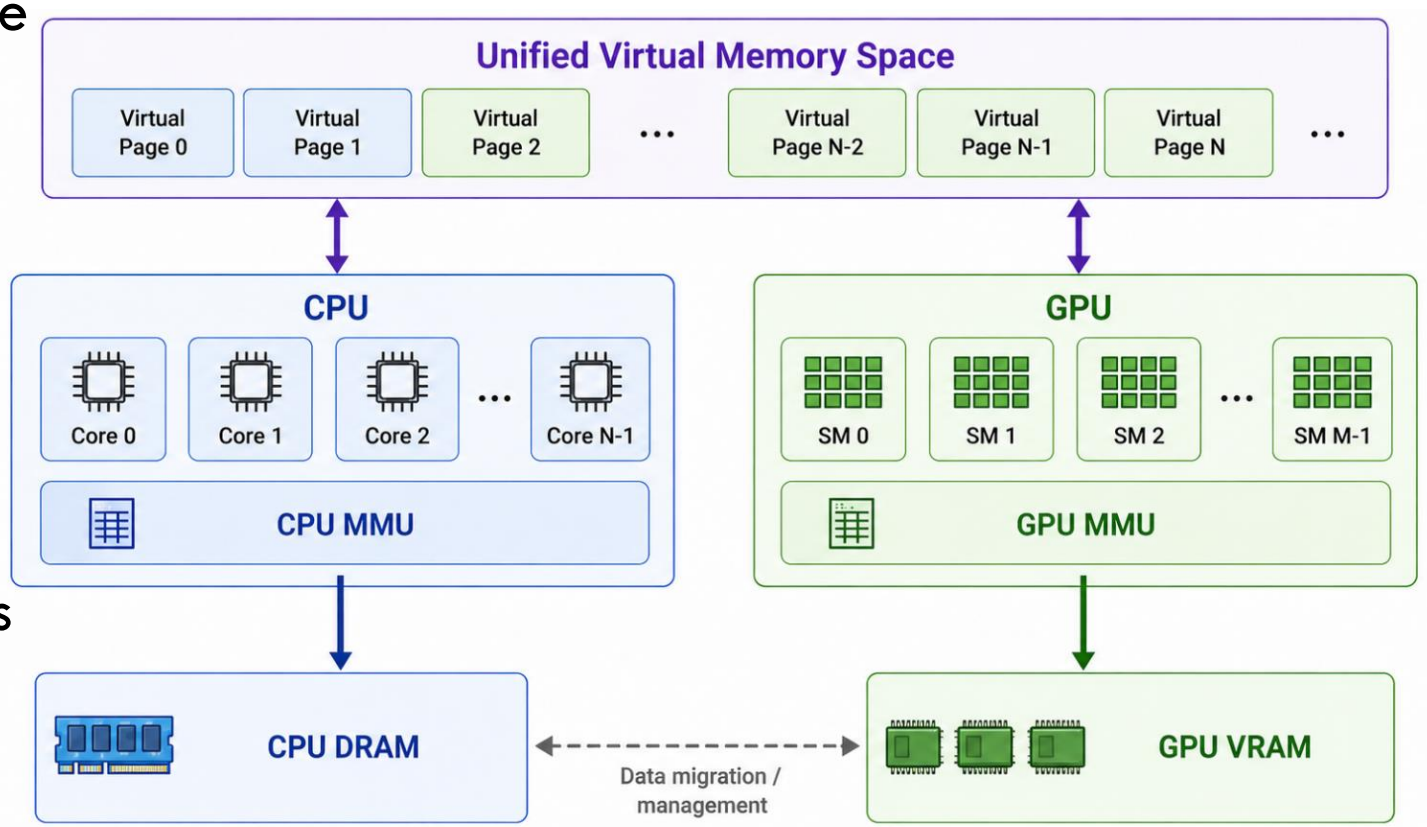
cuPTW: Leveraging Idle Compute Units for Massively Parallel GPU Page Table Walks

Zihang Chen[1], Tianao Ge[1], Lieven Eeckhout[2]
Hongyuan Liu[3], Jiayi Huang[1]

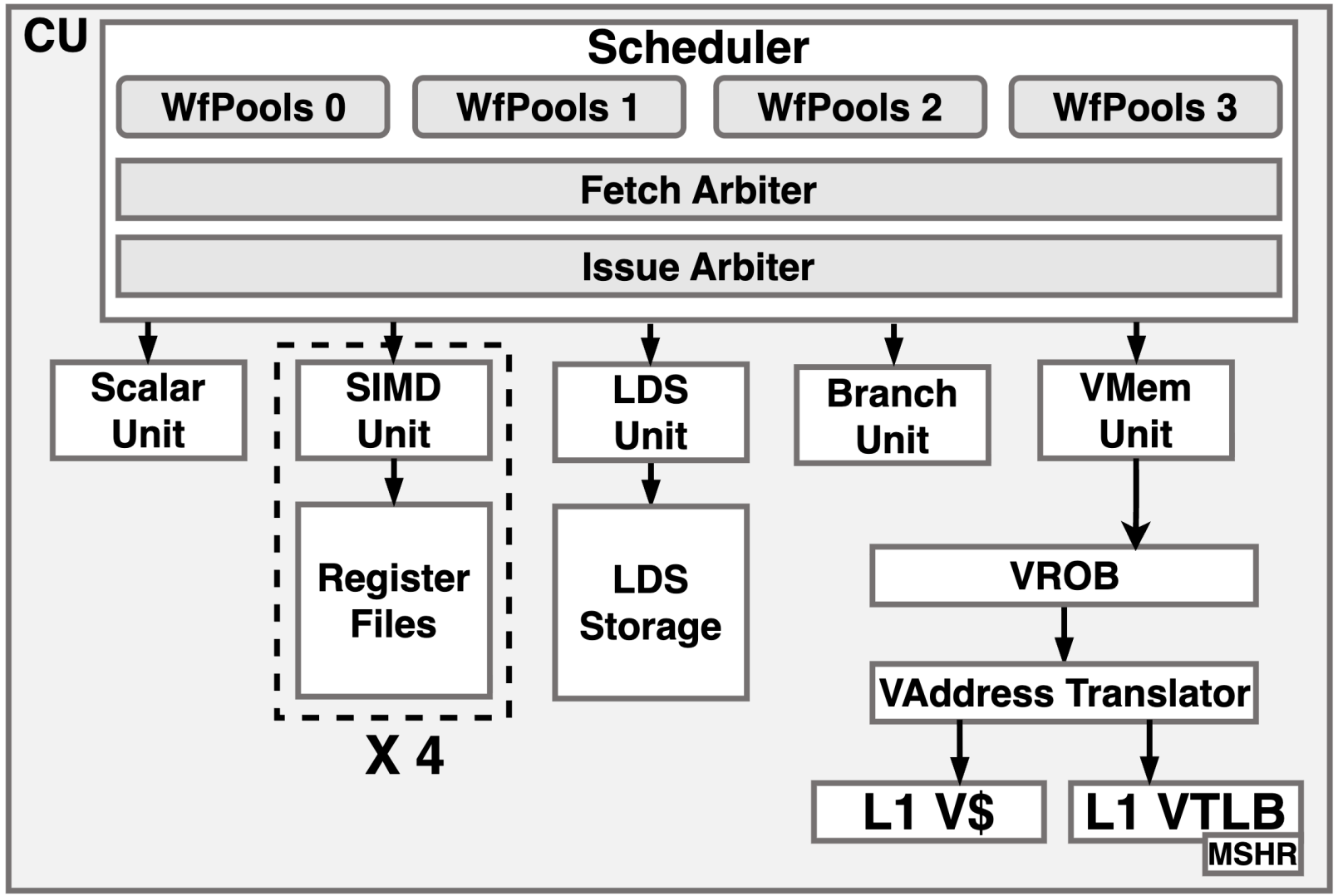
[1] The Hong Kong University of Science and Technology (Guangzhou)
[2] Ghent University
[3] Stevens Institute of Technology

Why Address Translation important in GPUs

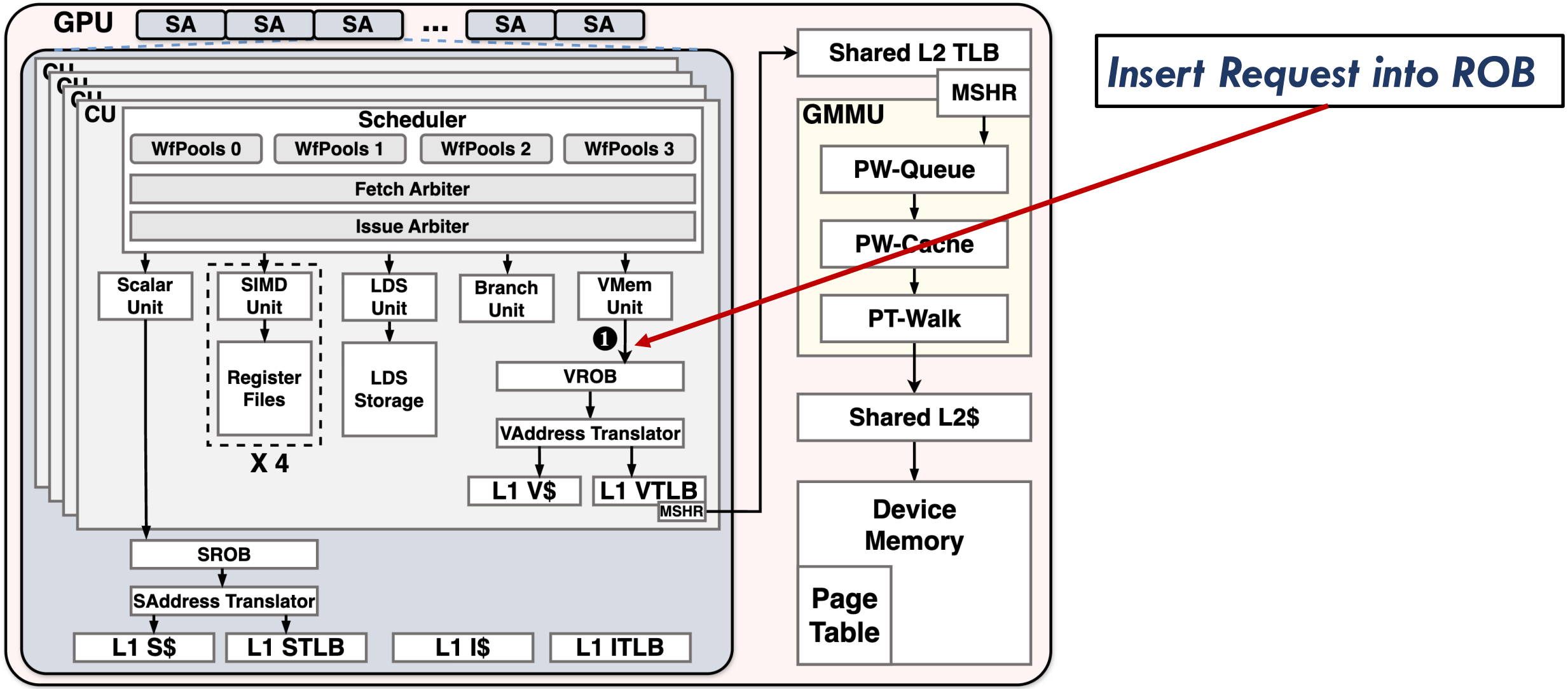
- Unified Memory / Unified Virtual Space
  - Simplify programming
  - i.e., cudaMemManaged()
- Memory Oversubscription
- Protection &Isolation
  - Prevent unauthorized memory access



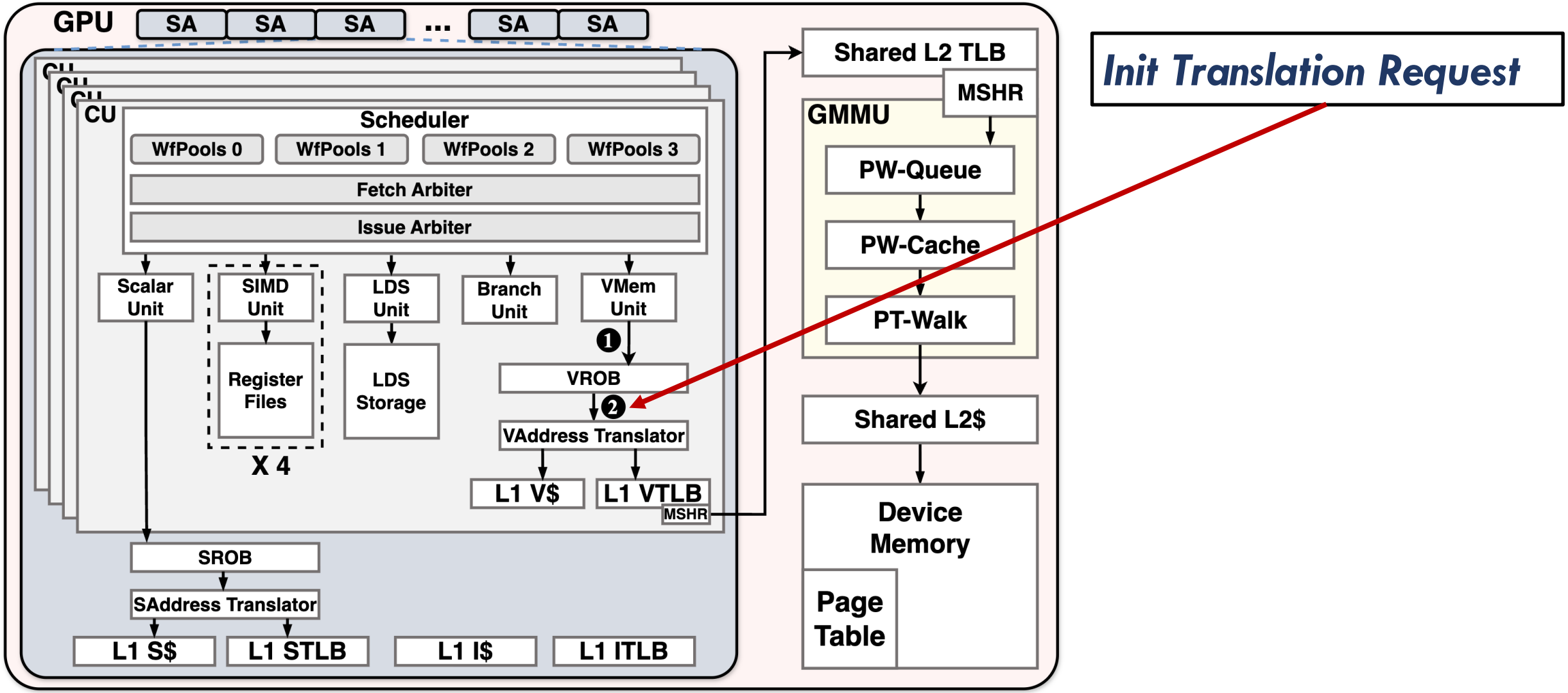
AMD Compute Unit Microarchitecture



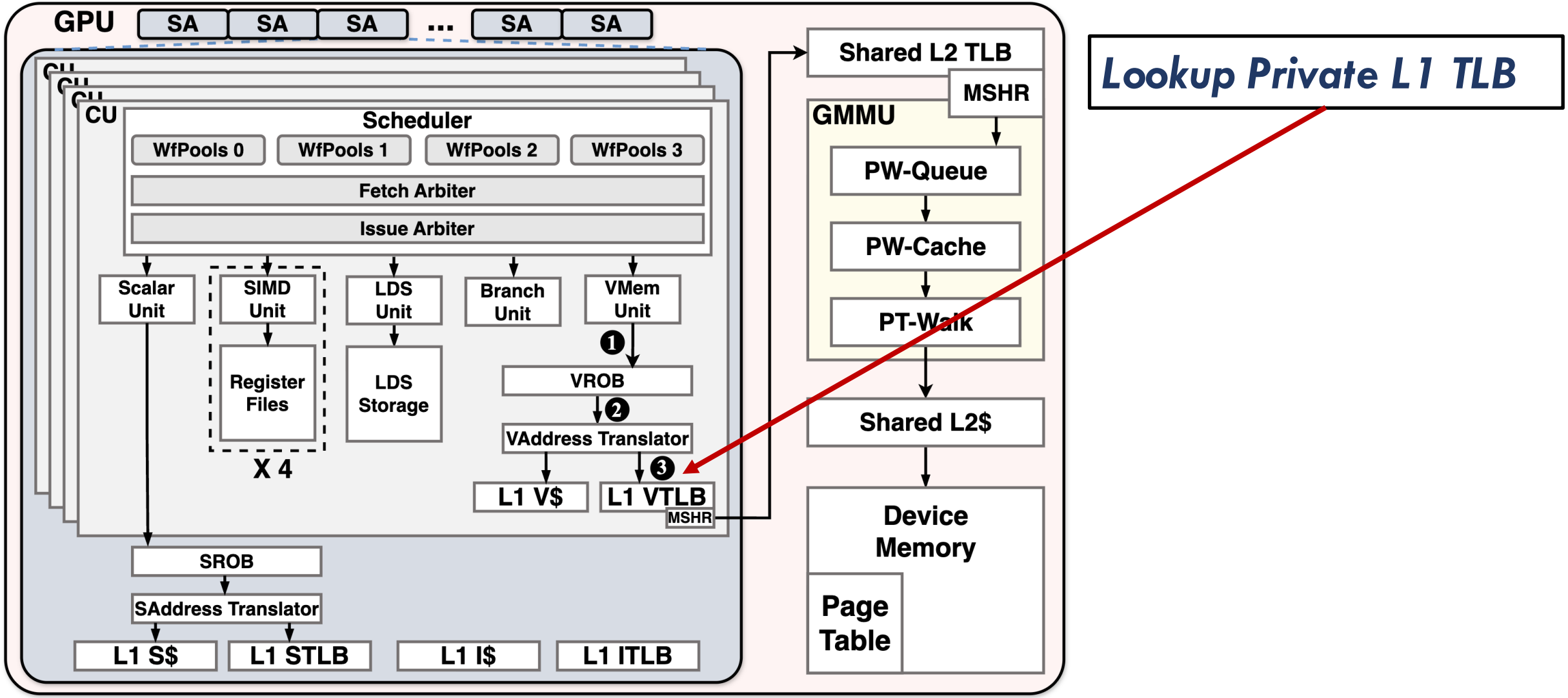
GPU Address Translation Workflow



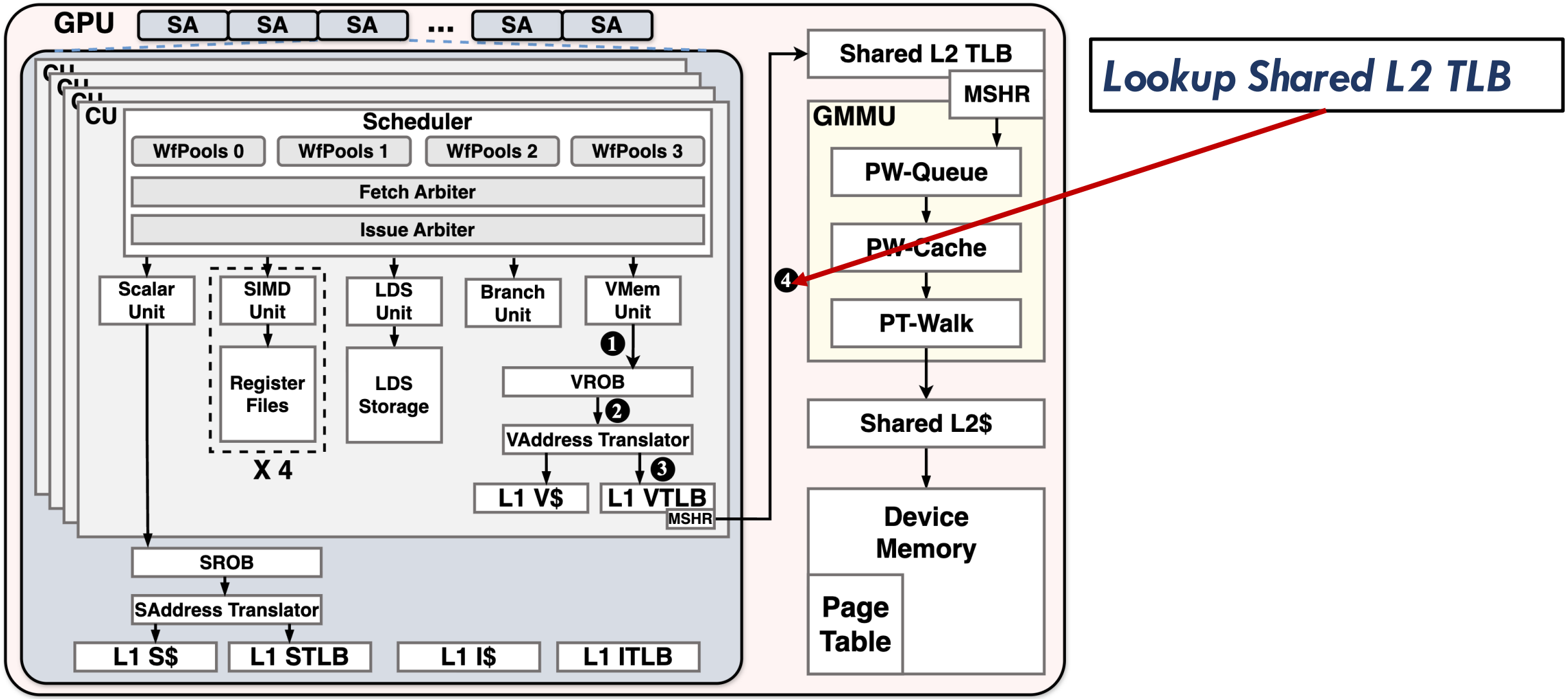
GPU Address Translation Workflow



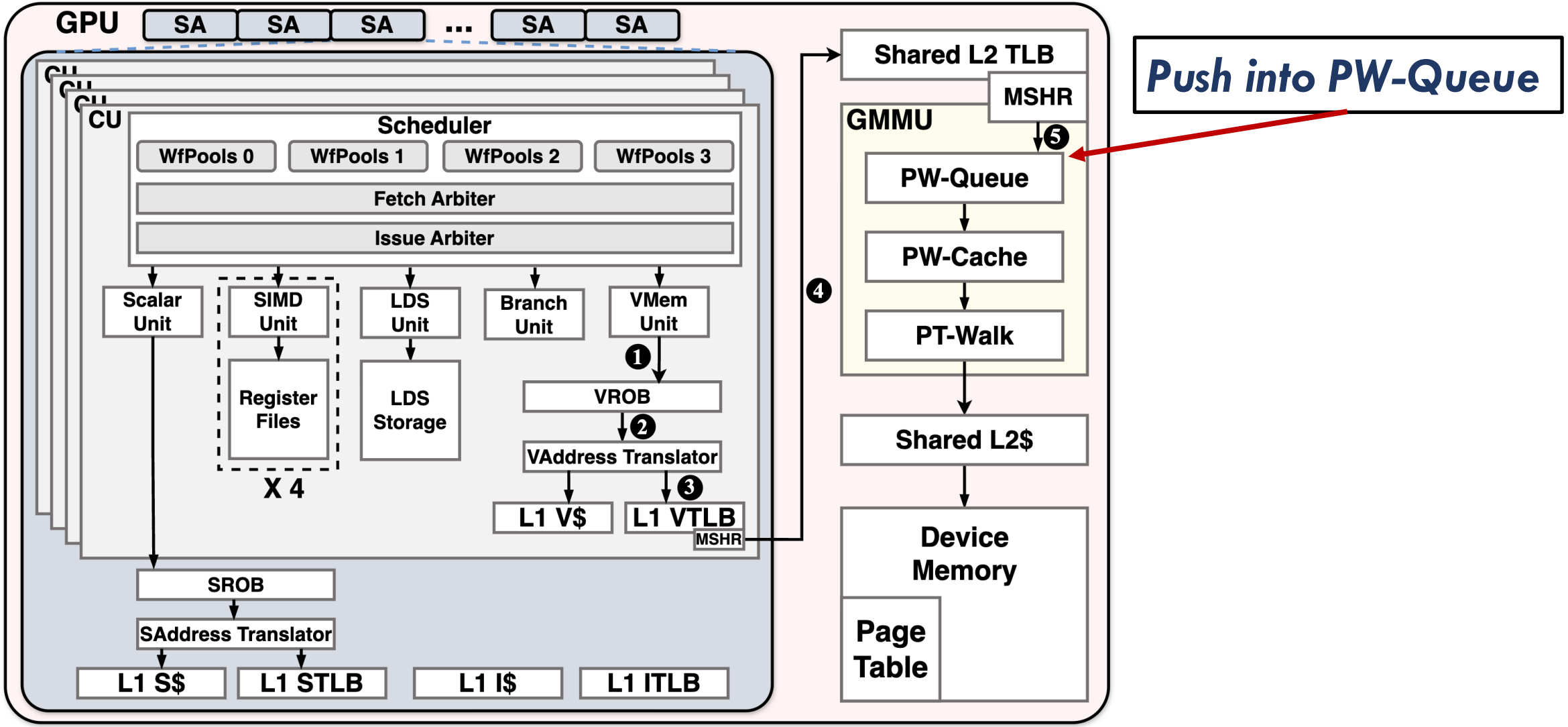
GPU Address Translation Workflow



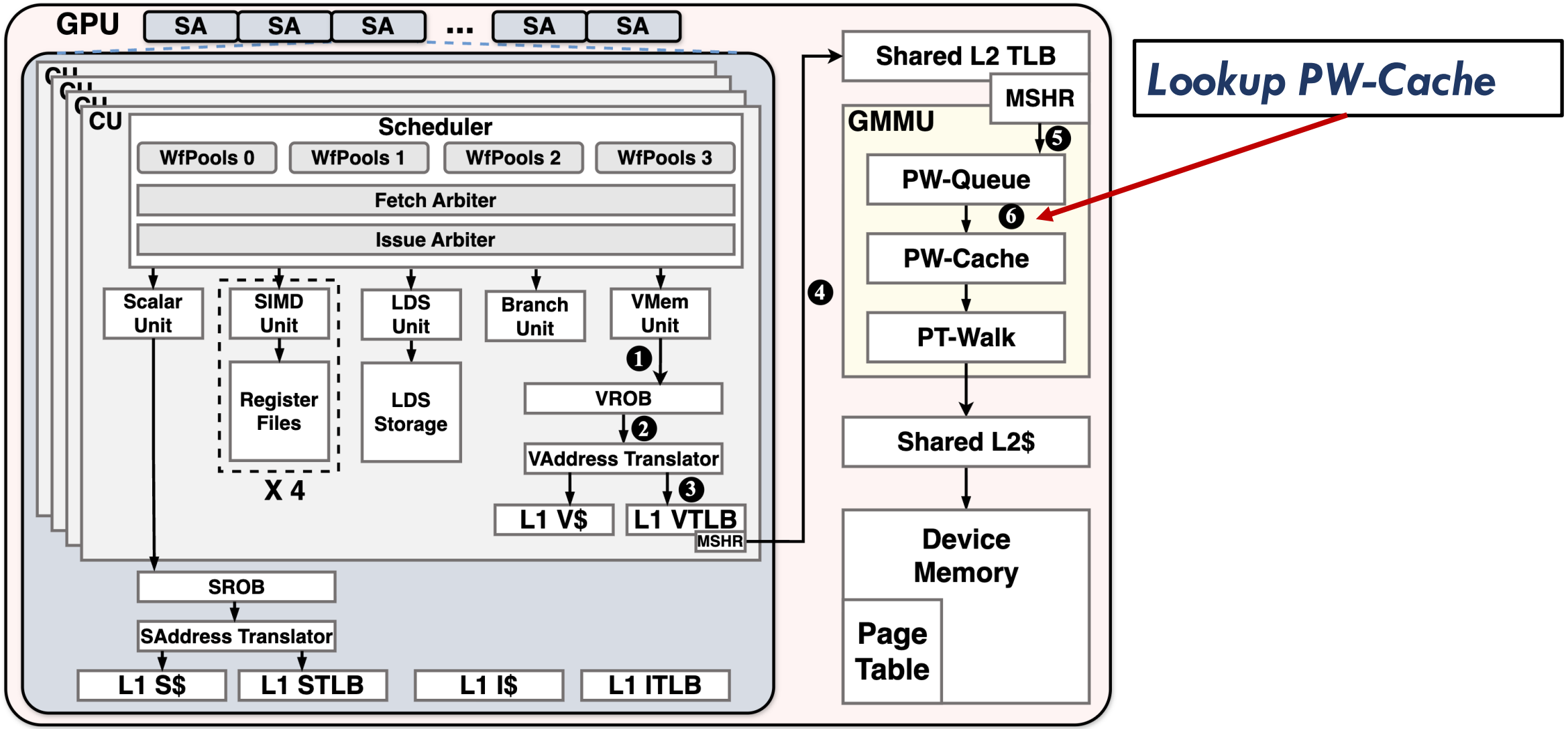
GPU Address Translation Workflow



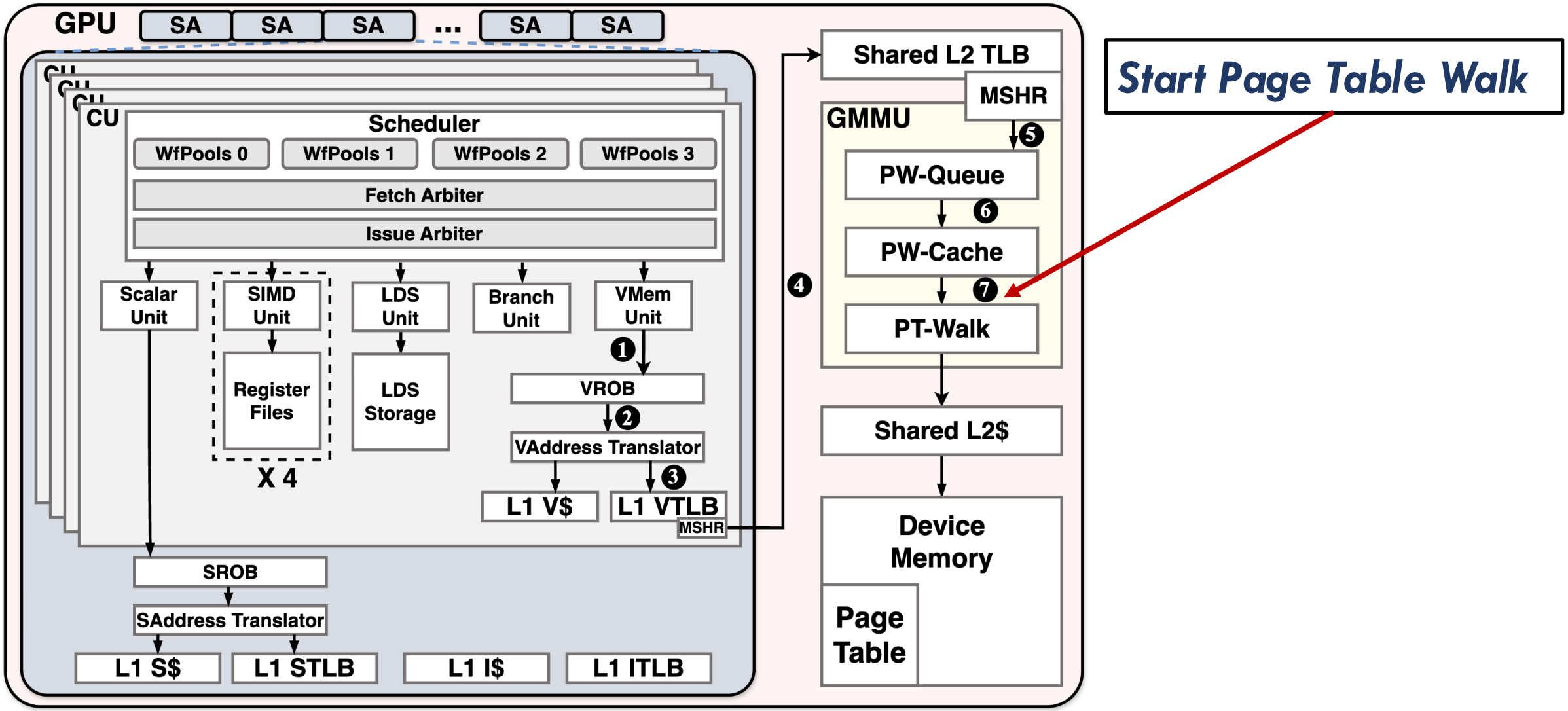
GPU Address Translation Workflow



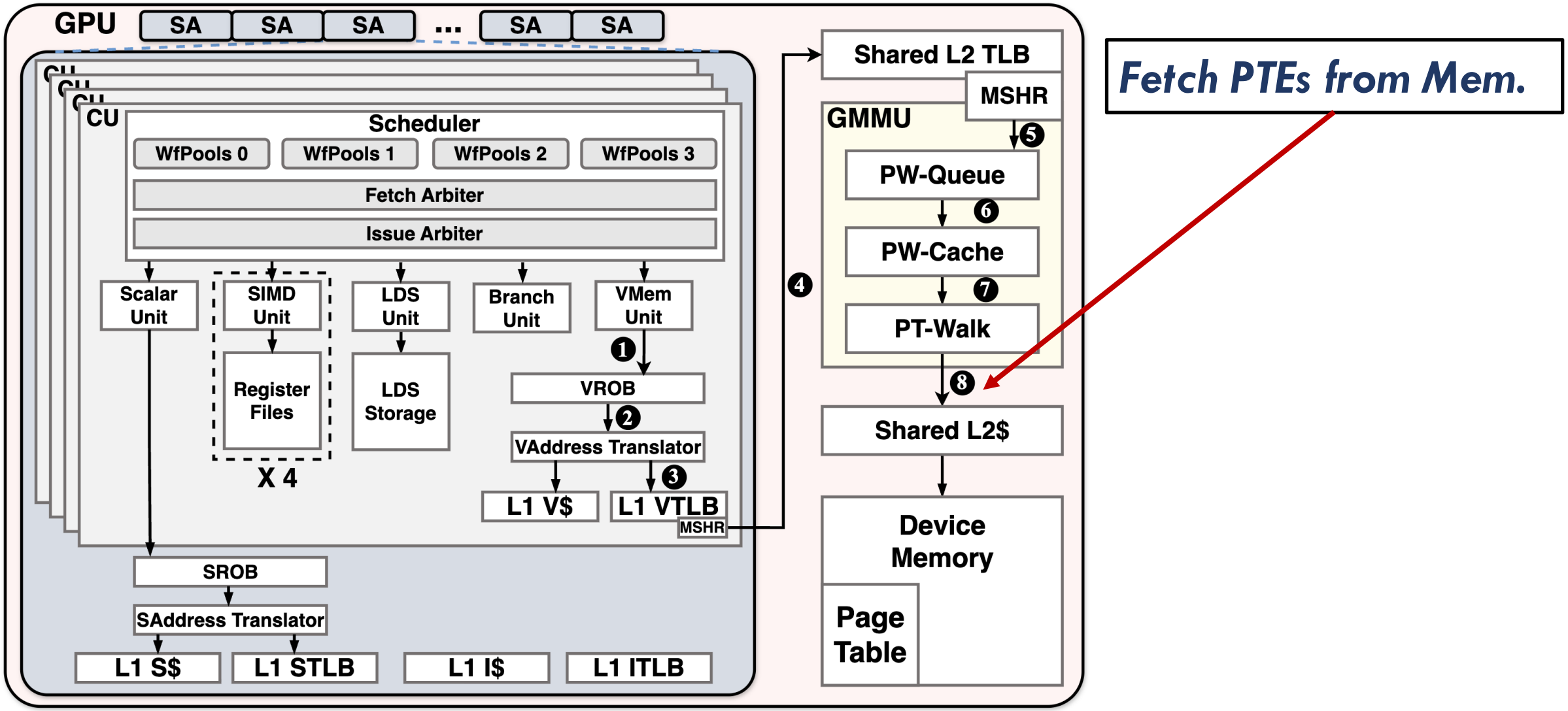
GPU Address Translation Workflow



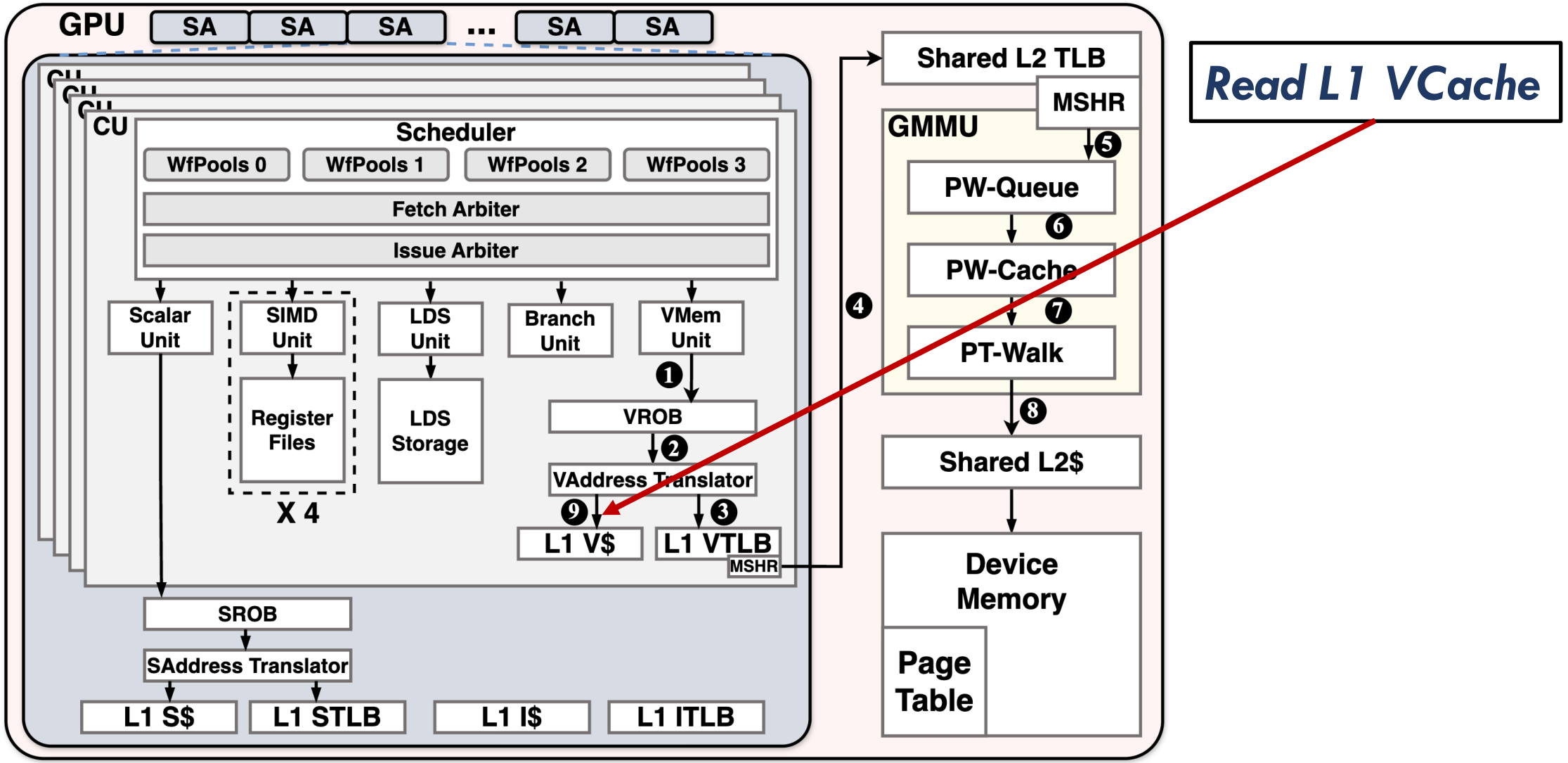
GPU Address Translation Workflow



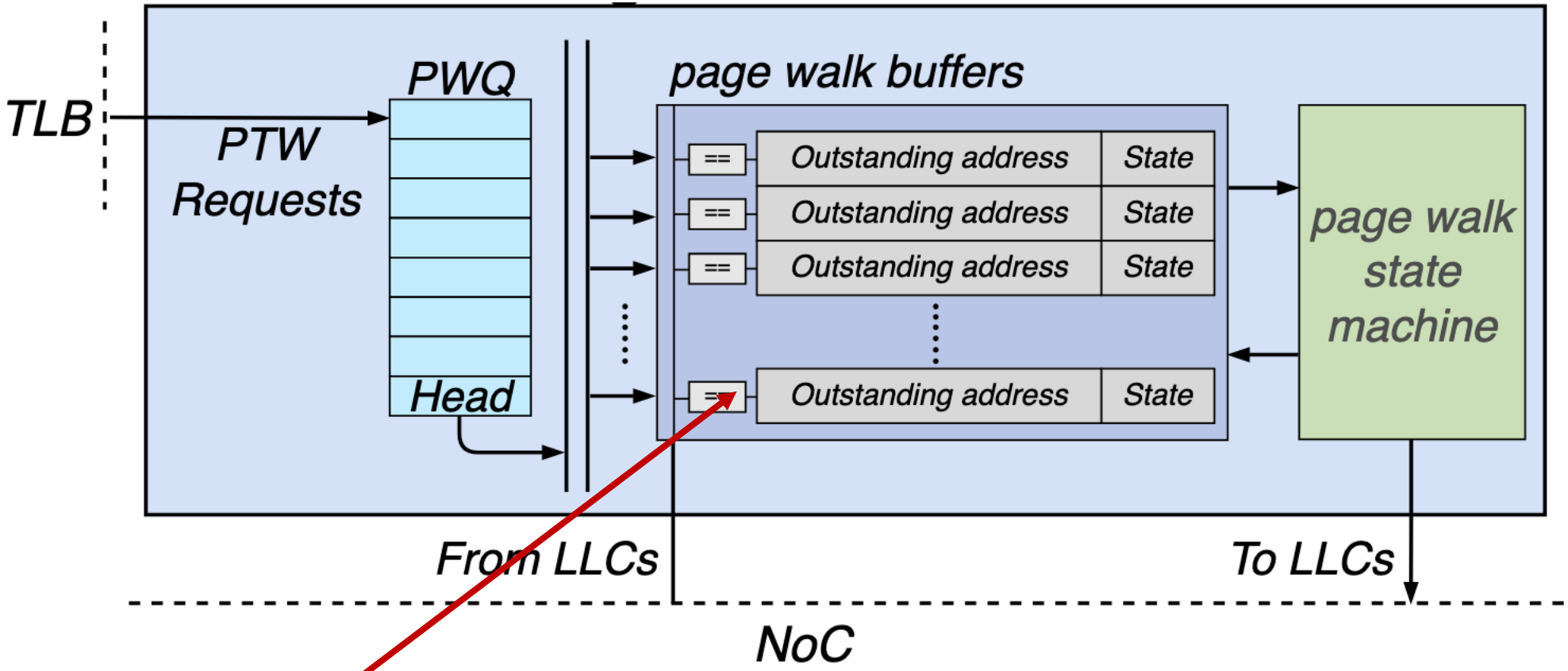
GPU Address Translation Workflow



GPU Address Translation Workflow



How Page Walkers Work



Content-addressable memory-Based Lookup

Motivation #1

Translation Dominates Memory Latency

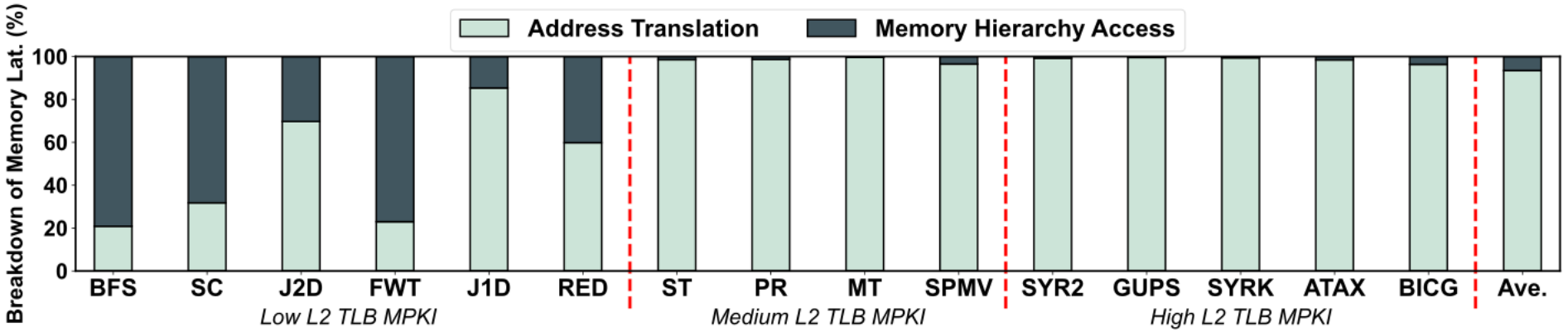


Figure. Breakdown of memory request latency on the baseline GPU.

92.1 % Address Translation **7.9 % Data Access**

Takeaway: TLB misses and page walks, not data cache/DRAM access, dominate GPU memory stall latency.

Motivation #2

Translation stalls waste valuable compute resources

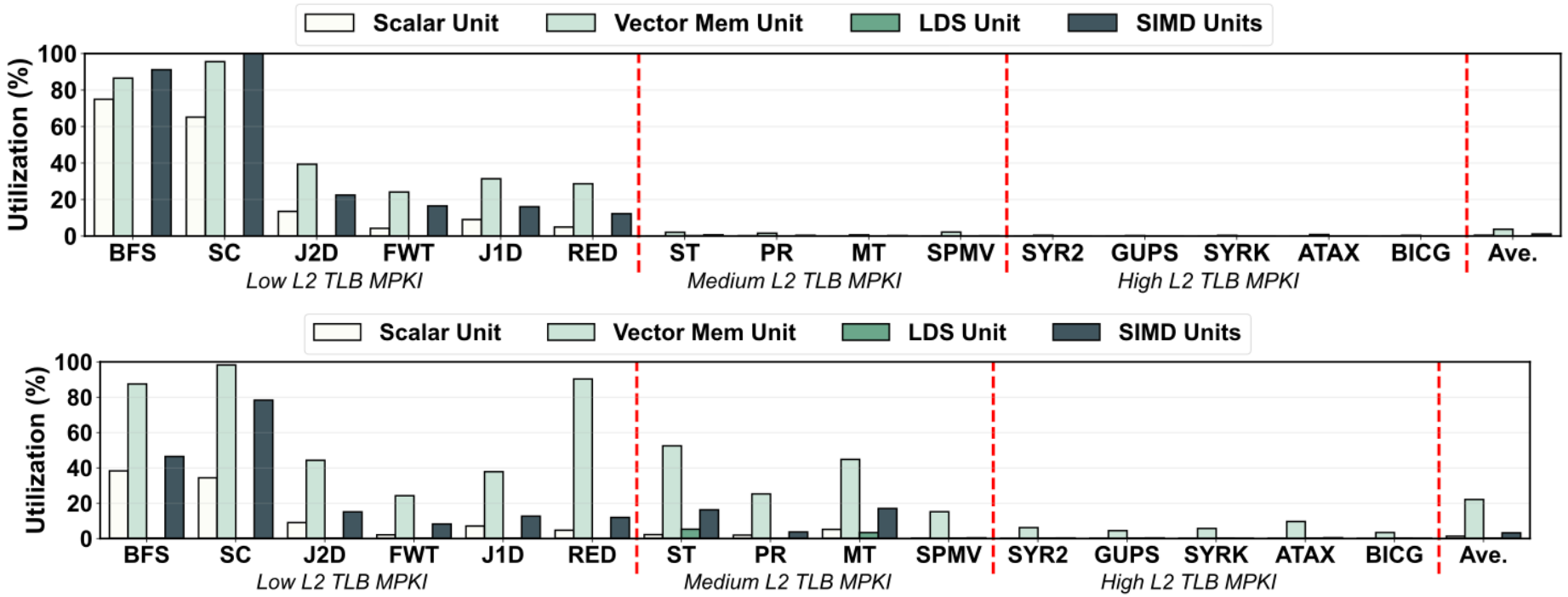


Figure. Overall utilization of functional units for the baseline GPU with (upper) / without (bottom) translation overhead.

Takeaway: Due to the inefficiency of the address translation system, computational and memory resources are under-utilized.

Motivation #3

The number of page walkers influence performance

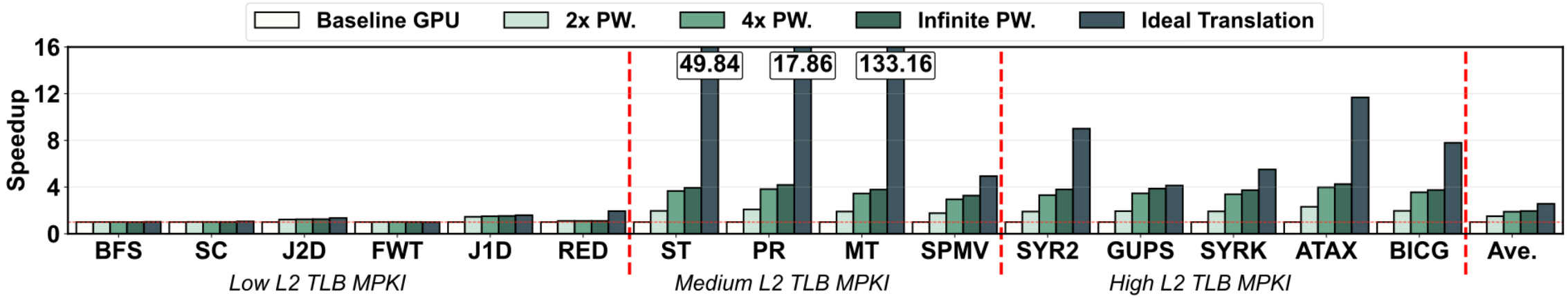


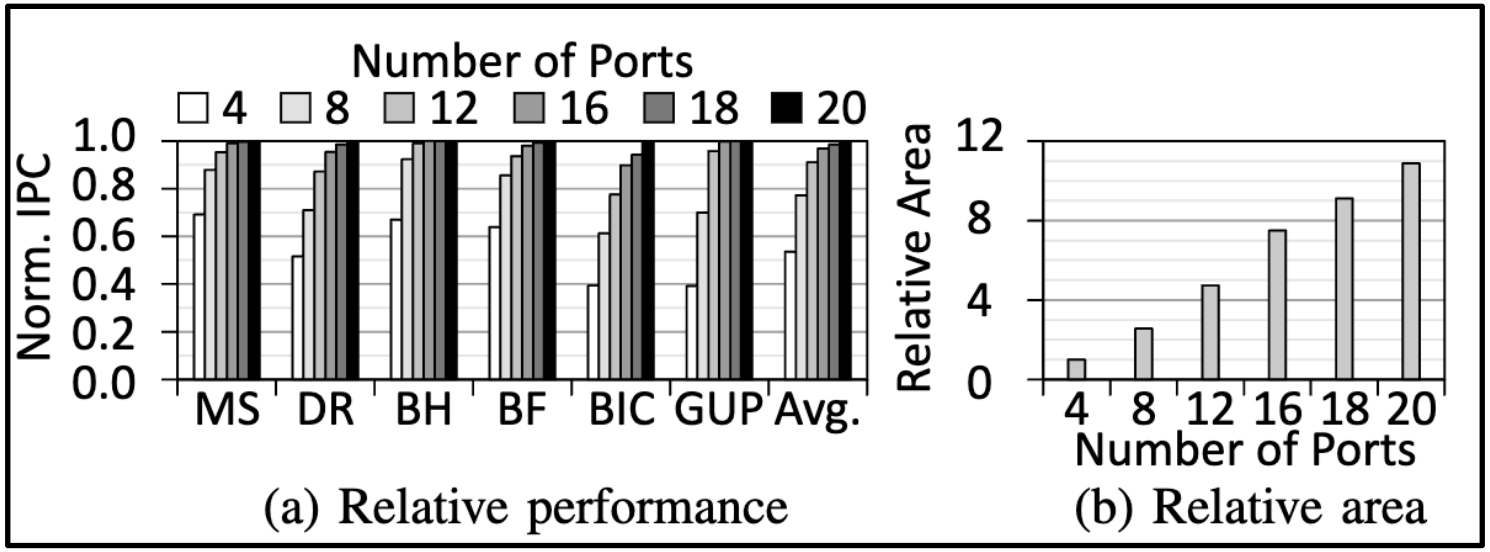
Figure. Speedup the number of page walkers versus ideal translation, normalized to the baseline GPU.

Takeaway: The number of page walkers significantly influences GPU translation efficiency.

Motivation #3

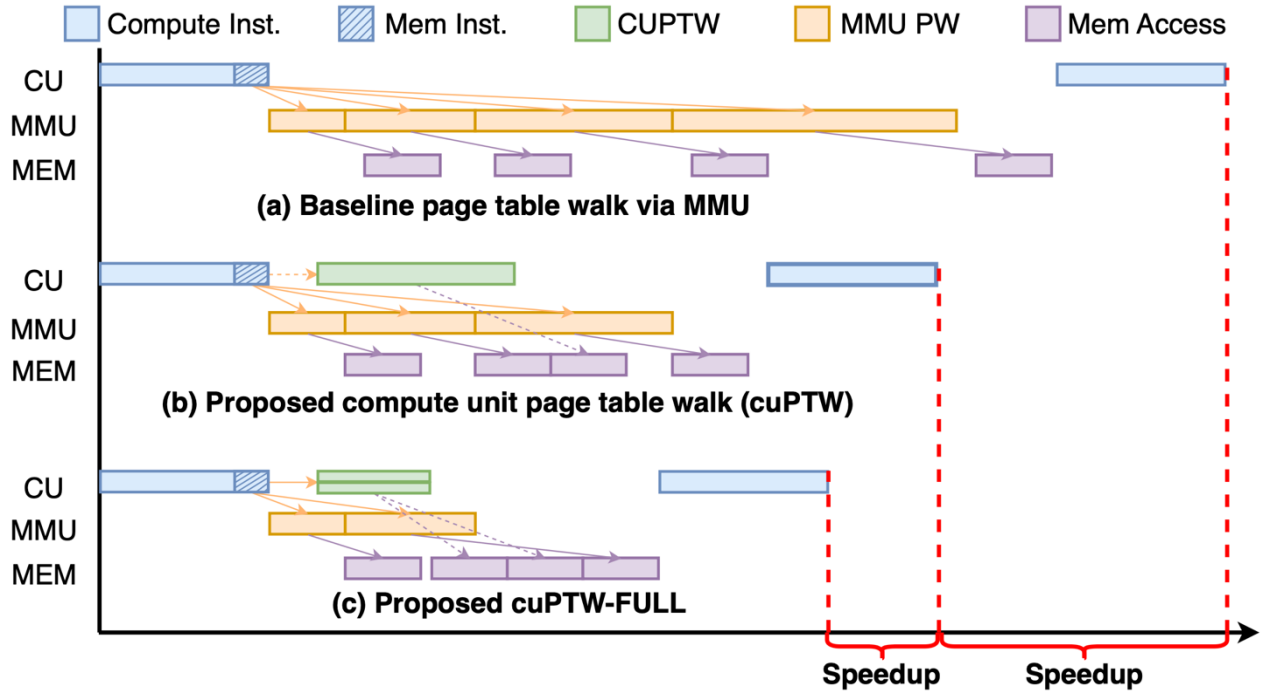
More page walker, More Area

2 x Speedup → 9 x Area Overhead [HPCA 2025]



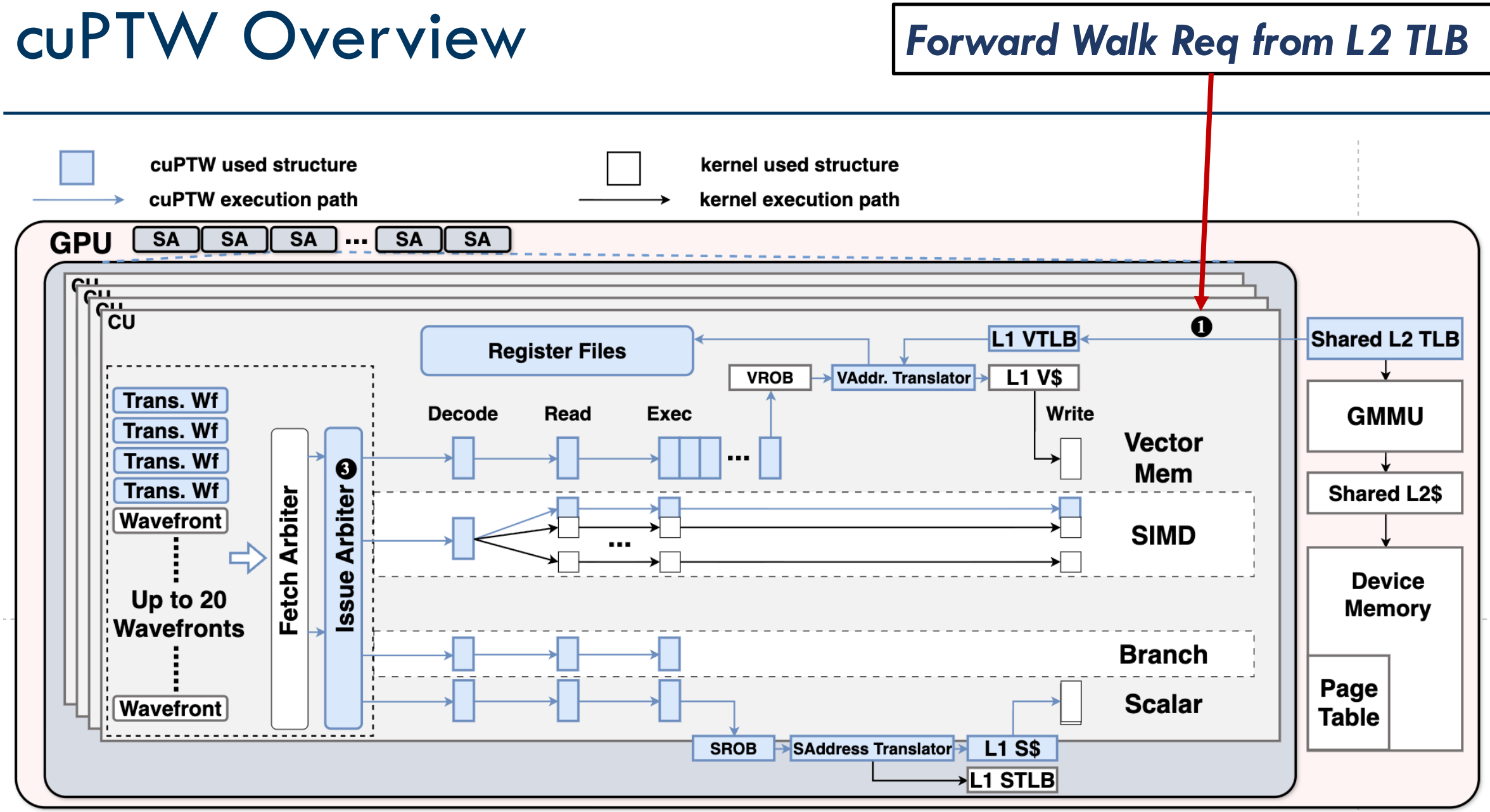
Key Idea

Turn page table walks into massively parallel computation.



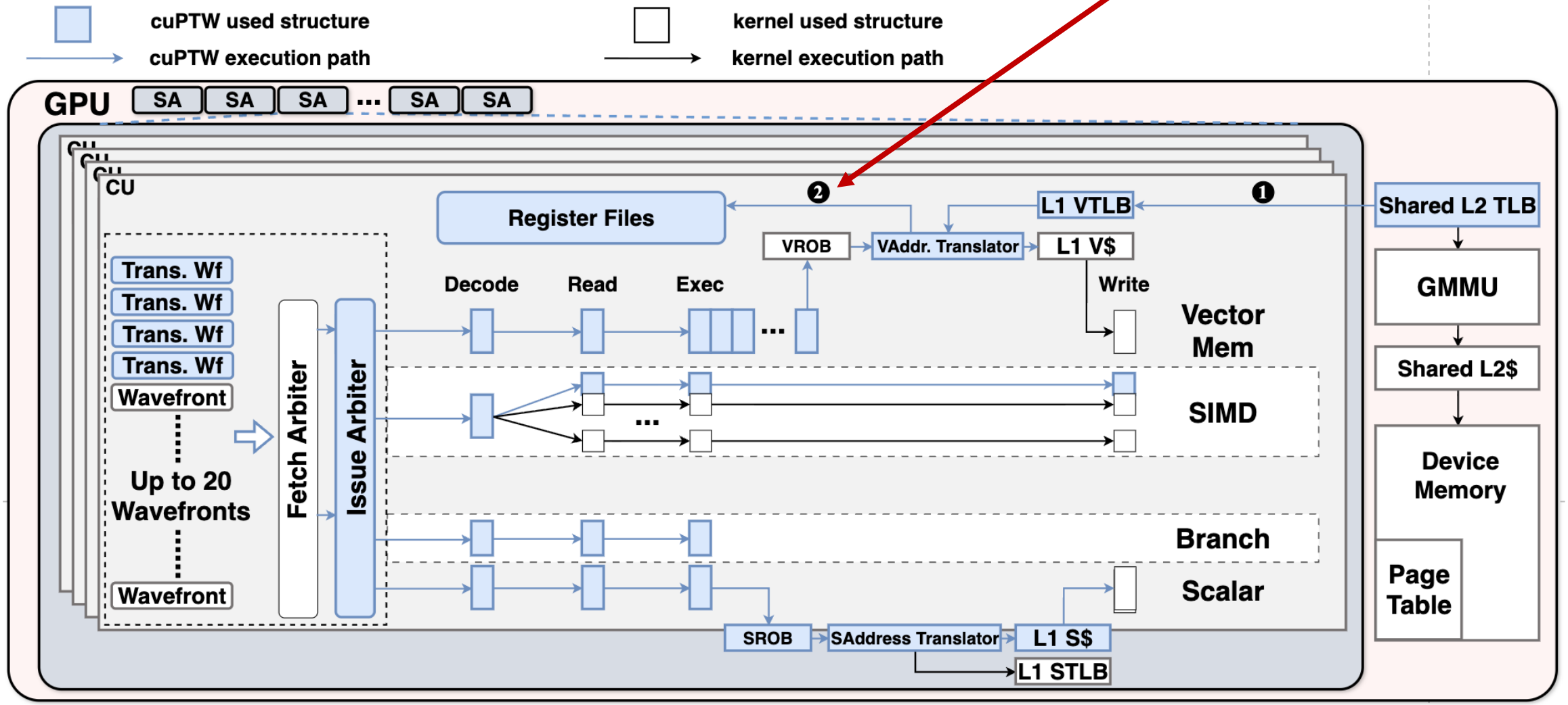
Key Idea: Instead of adding many hardware walkers, cuPTW repurposes idle CUs for massively parallel page walks.

cuPTW Overview



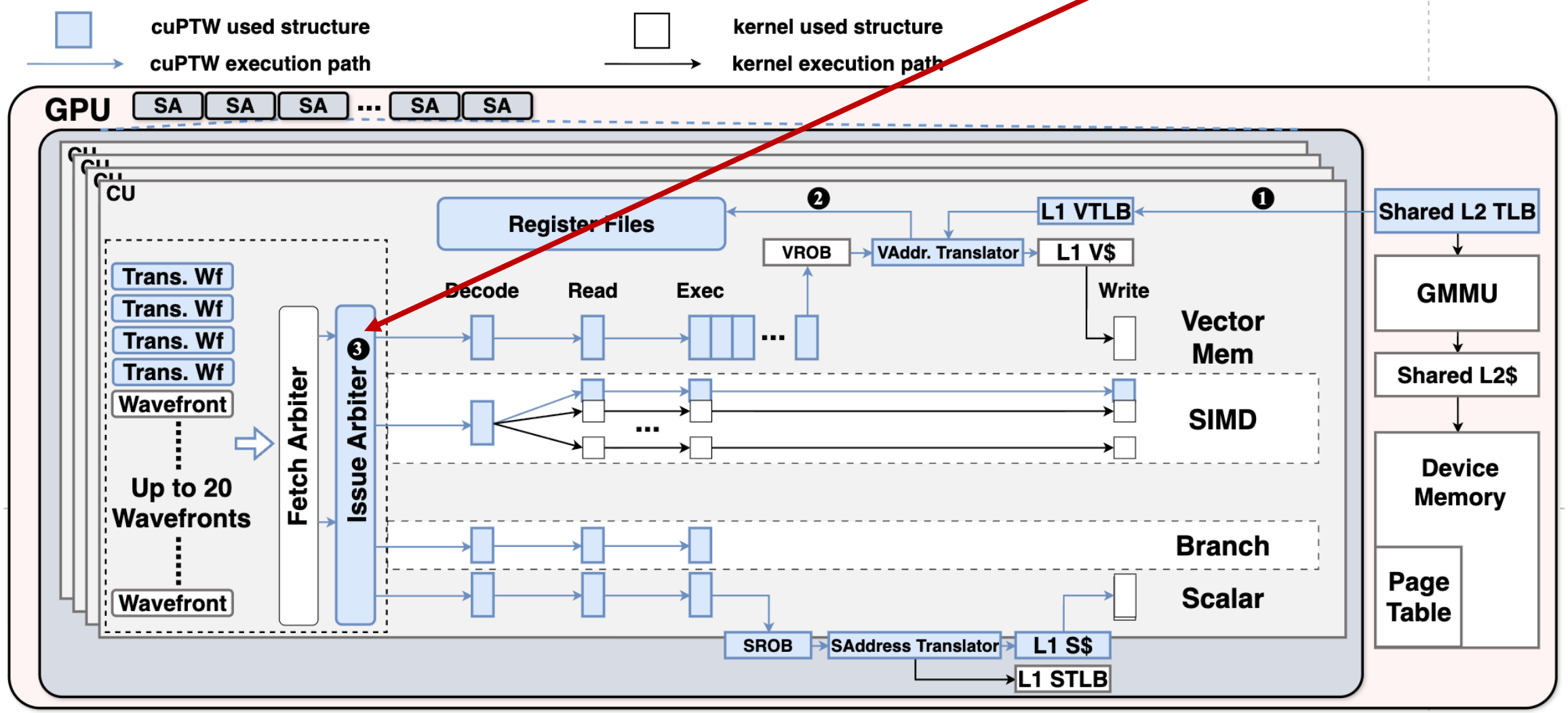
cuPTW Overview

Request Setup: Write Metadata to Regs



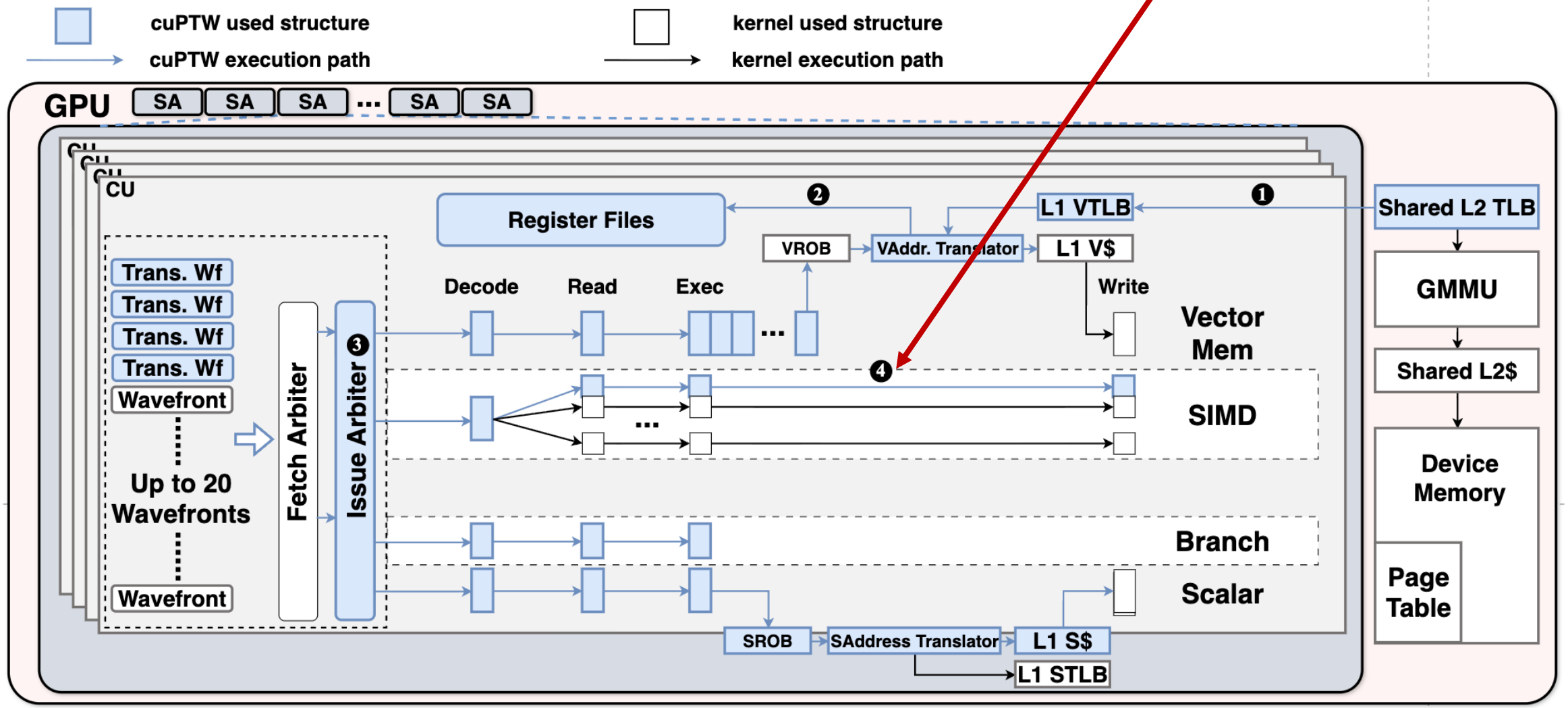
cuPTW Overview

Dispatch and Issue Translation Wave.

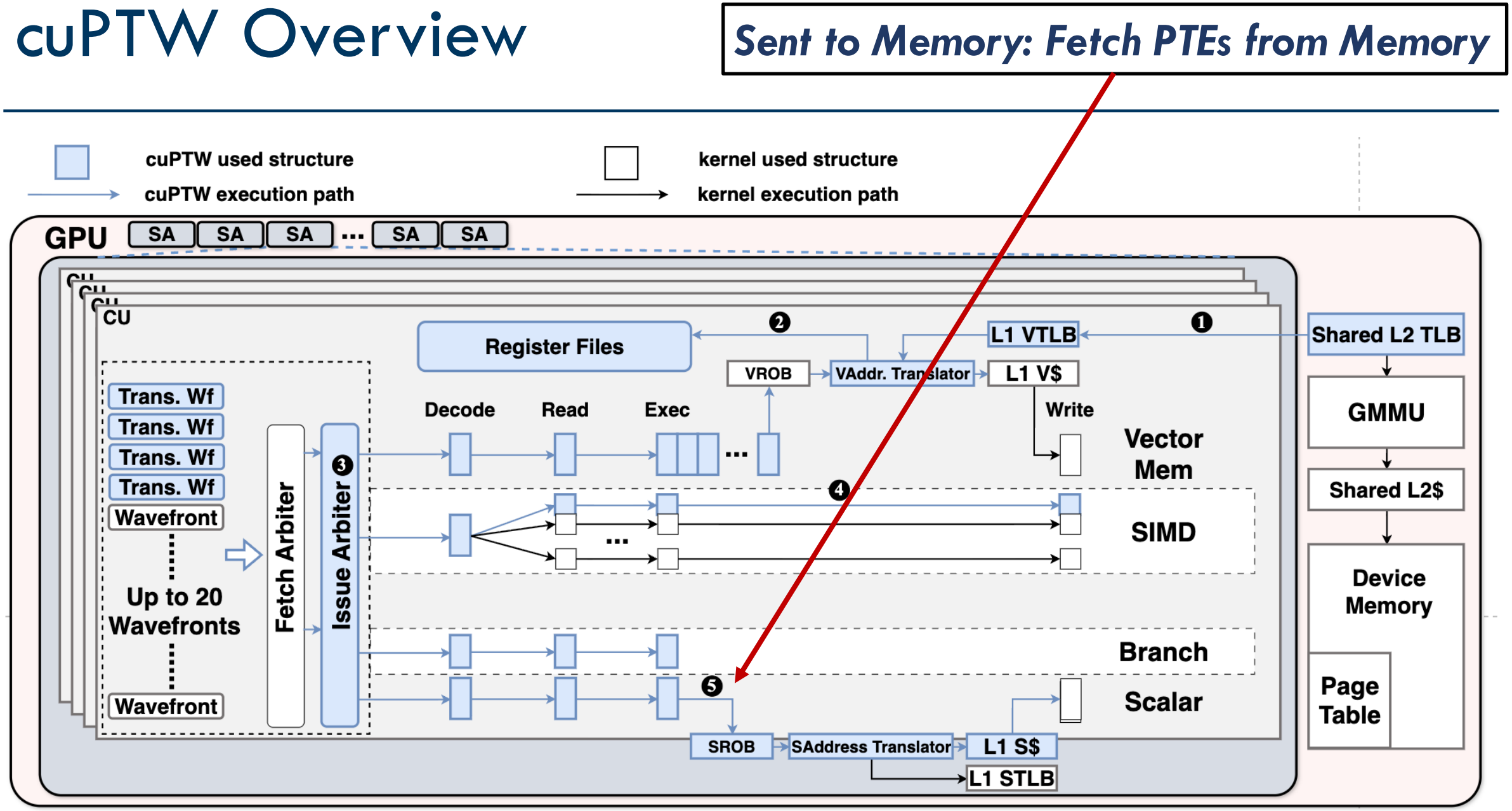


cuPTW Overview

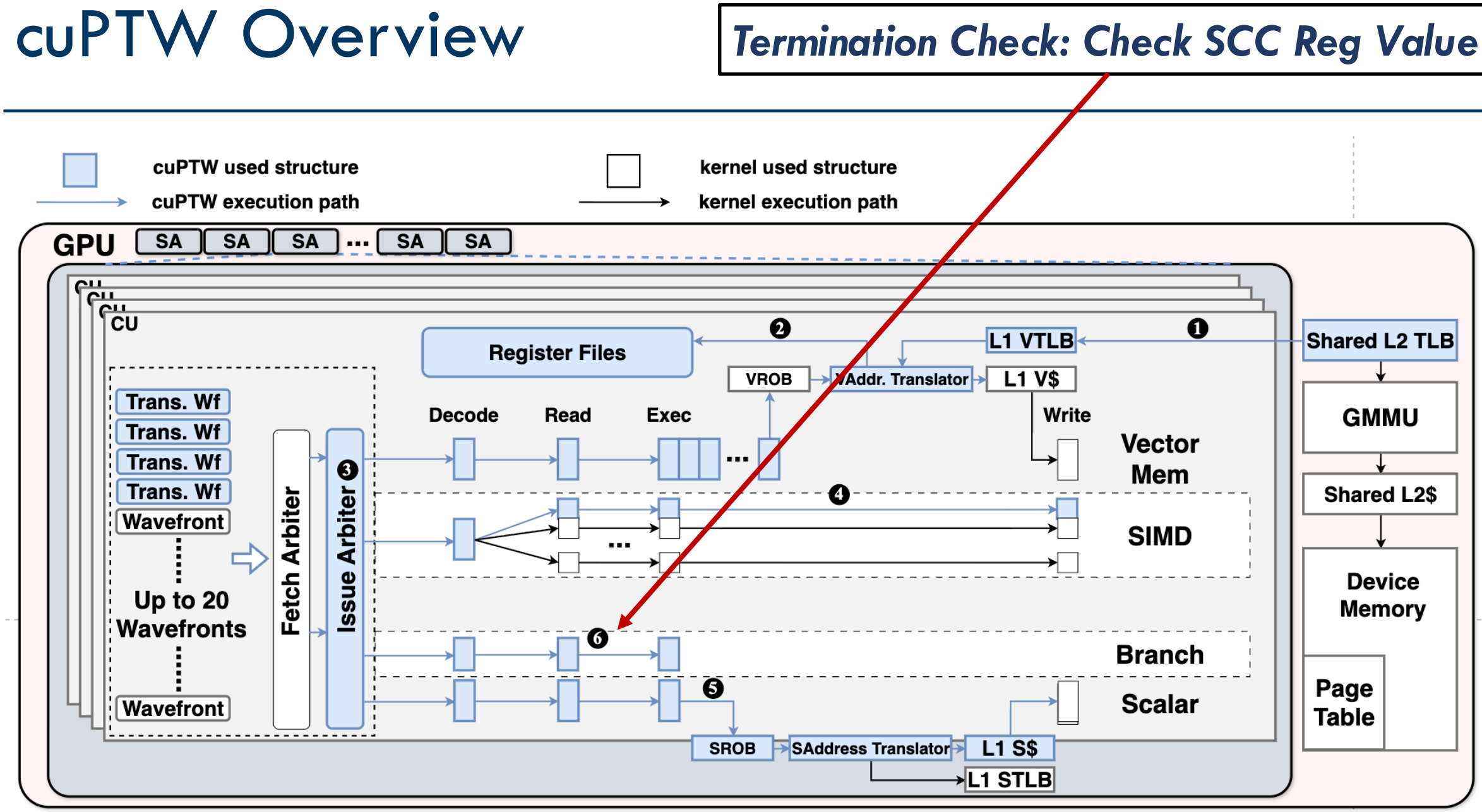
Page Offset Calculation: Compute offset for the PTE address and Set SCC Register



cuPTW Overview

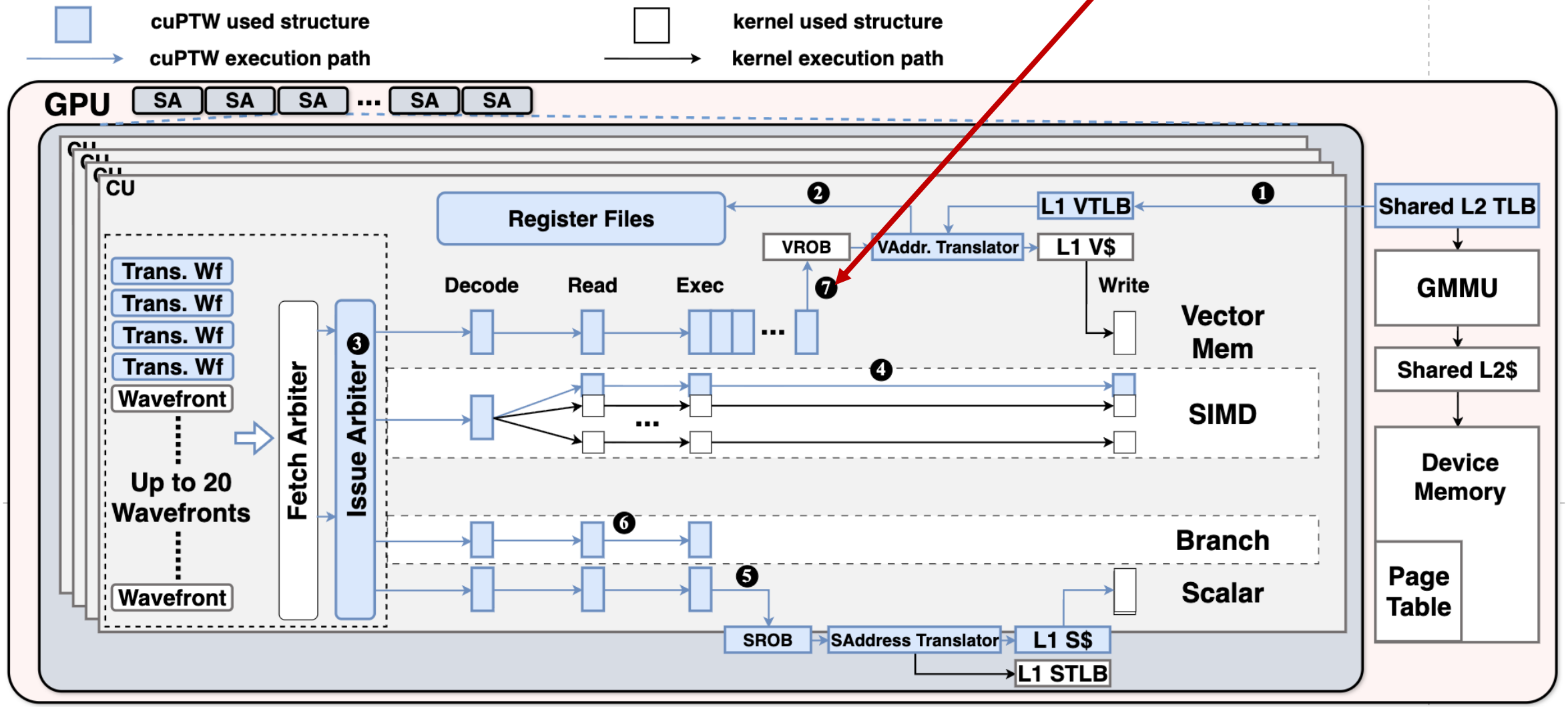


cuPTW Overview

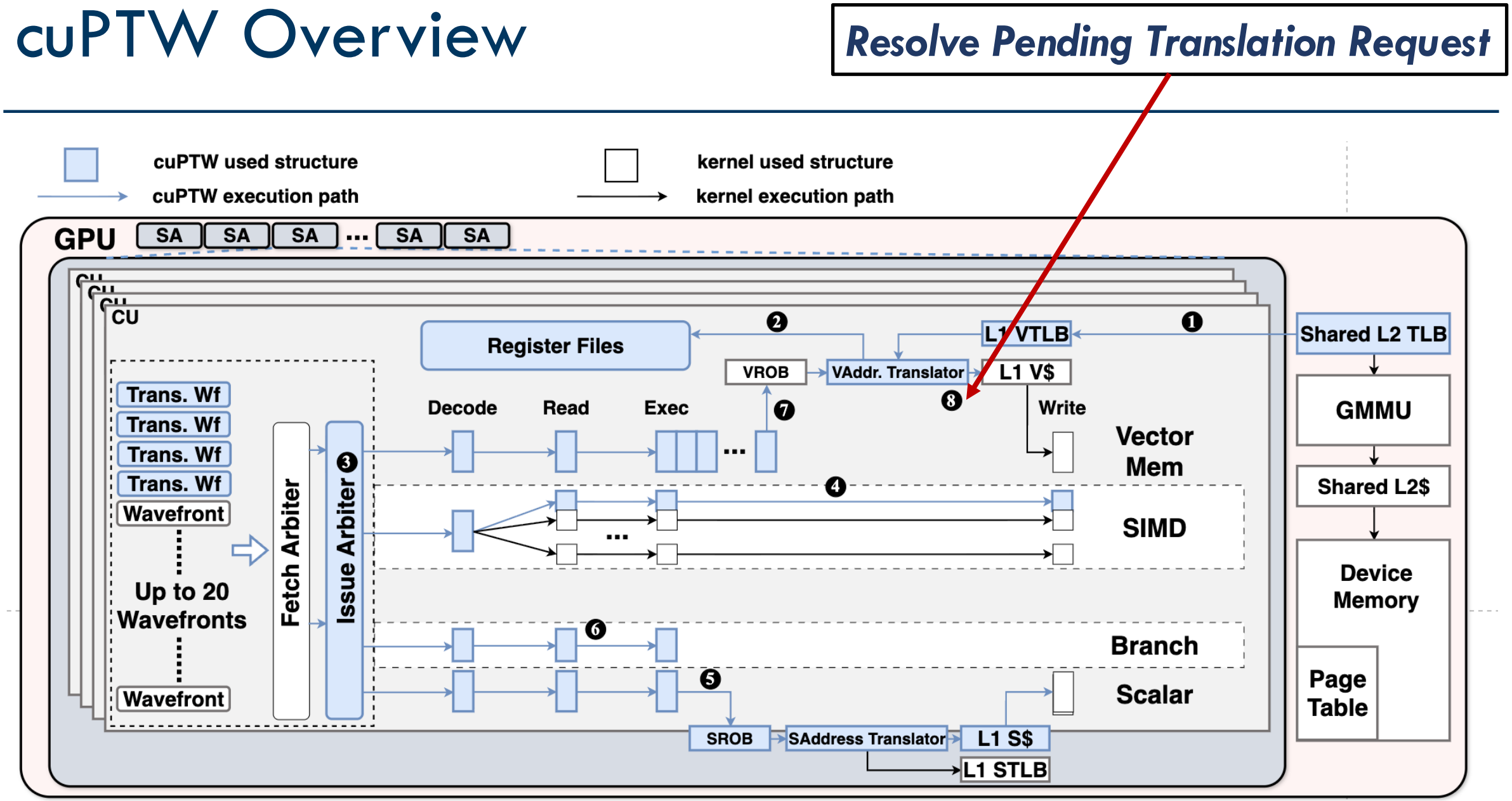


cuPTW Overview

Translation Done: Send PTE via VMem Unit



cuPTW Overview



Translation Wavefront Overview

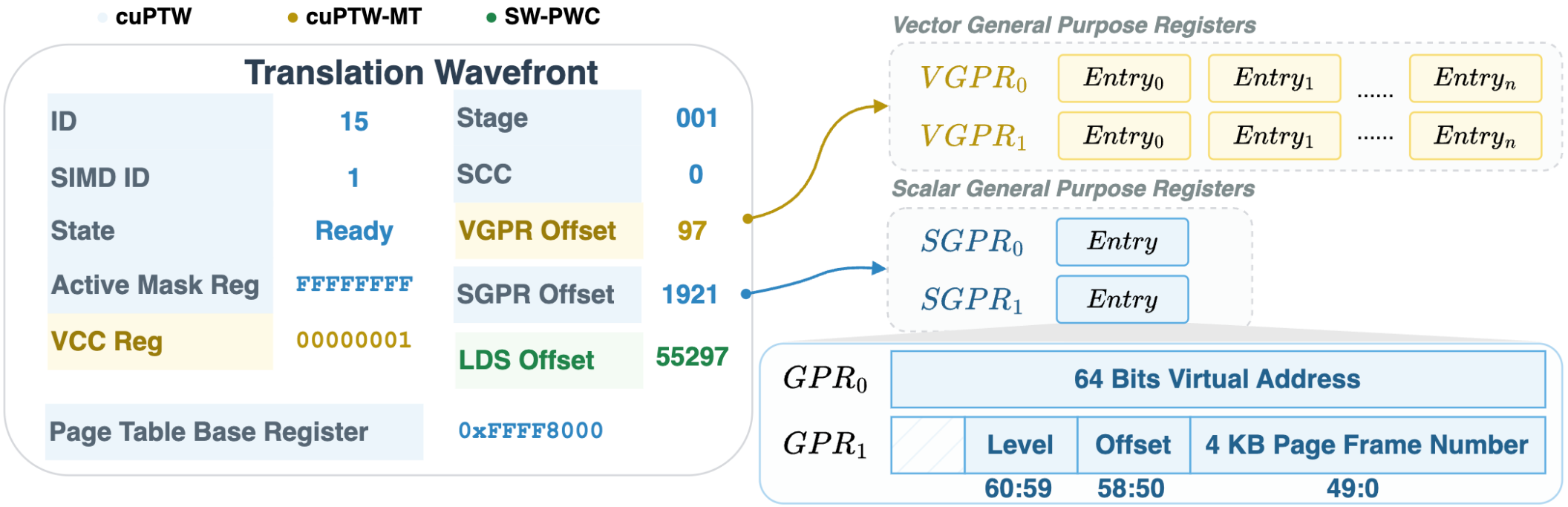
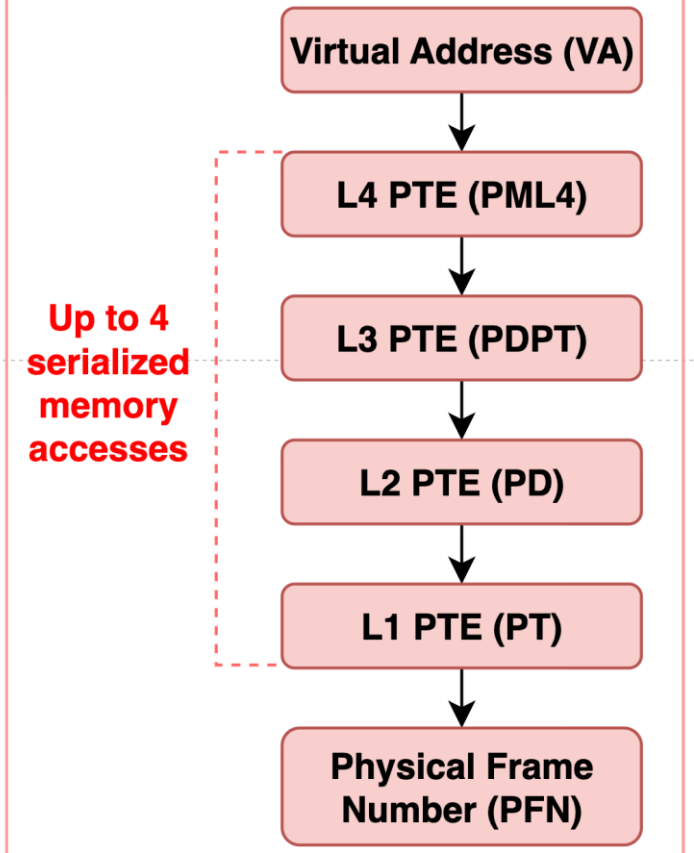


Figure. Overview of the translation wavefront.

Software-Managed Page Walk Cache

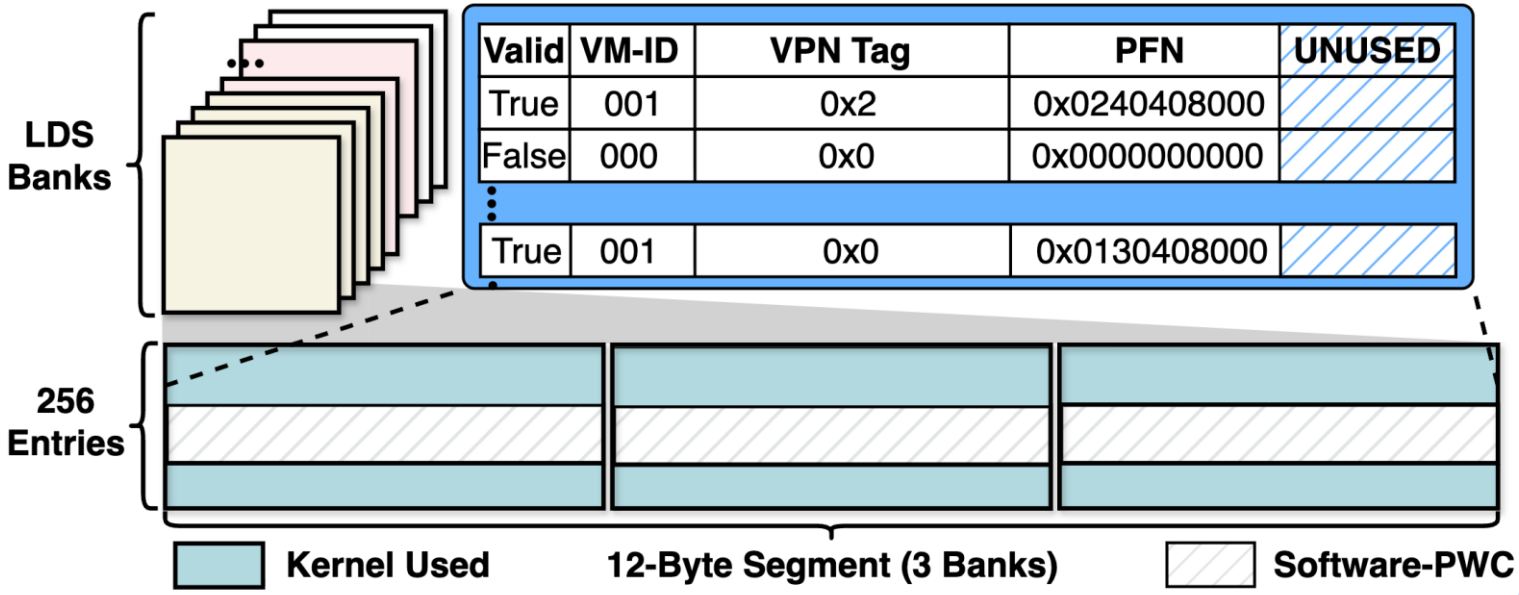
Use idle LDS as a software-managed page walk cache

Problem: Full 4-level Page Walk



Our Solution: Software-Managed Page Walk Cache in Idle LDS Memory

Lookup the SW-PWC before walking the page table



1.81 x

Lower cuPTW walk latency

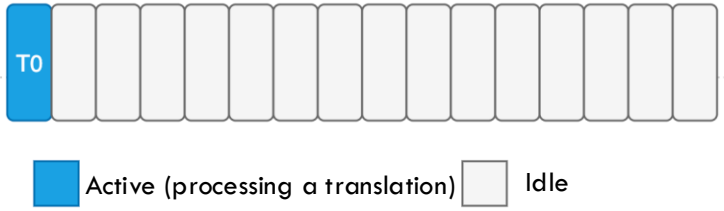
Multi-Threaded cuPTW

Use multiple SIMD lanes to process multiple translations in parallel

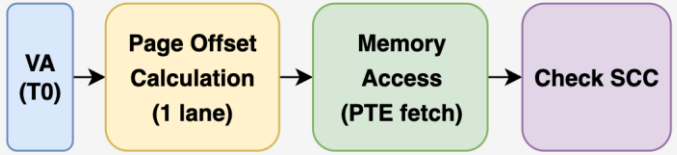
Problem: Full 4-level Page Walk

Each translation wavefront uses only one lane

Only 1/16 lanes active (6.25% utilization)



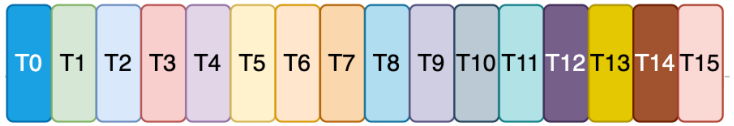
Example: One translation per wavefront



Most SIMD lanes are idle -> Low page-walk throughput

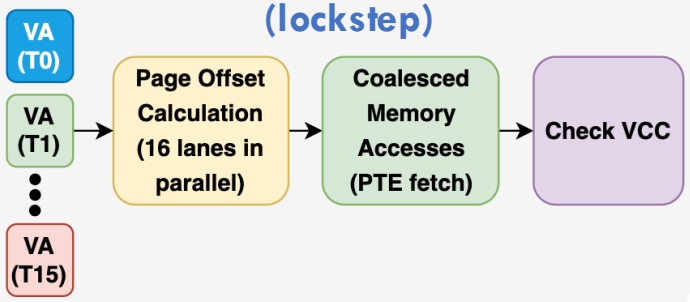
cuPTW-MT (Multi-Threaded)

Pack multiple translation request into one wavefront



Each colored lane handles one translation request

Example: 16 translations per wavefront (lockstep)



More lanes active -> Higher page-walk throughput

Performance gain (vs. cuPTW)

Up to 5.63 x speedup

1.15 x geometrics mean

Methodology

- Simulator: MGPUSim [ISCA 2019]

Module	Configuration
# of CU	1.0 GHz, 128 in total
CU	4 exec. units per CU, 64 threads per wavefront
DRAM	1 TBps, 100 ns latency
L1 V-cache	64 KB, 16-way set associative, 28 cycles, CU private
L1 S-cache	64 KB, 16-way set associative, 28 cycles, shared b/w 4 CUs
L1 I-cache	64 KB, 16-way set associative, 28 cycles, shared b/w 4 CUs
LDS	32 KB, 32 banks, 22 cycles (5 cycles bank conflict penalty), CU private
L2 cache	8 MB, 16-way set associative, 160 cycles, GPU shared
L1 TLB	32 entries, 8 MSHR entries, fully associative, 20-cycle lookup latency, CU private
L2 TLB	2048 entries, 256 MSHR entries, 8-way, 80-cycle lookup latency, GPU shared
Page table walk	16 page table walkers, GPU shared [60, 61, 65]
Page walk cache	32 entries fully associative, 10-cycle lookup latency [16, 58]

Methodology

- Workloads
  - Application classified by L2 TLB MPKI into ***L***ow, ***M***edium, ***H***igh

Abbr.	Application	L1TLB MPKI	L2TLB MPKI	Memory footprint	L2 TLB MPKI
BFS	Breadth First Search	8.5	0.04	80 MB	Low
SC	Simple Convolution on Matrix	7.6	0.4	2048 MB	Low
J2D	2-D Jacobi Solver	95.7	2.2	2048 MB	Low
FWT	Fast Warshall Transform	4.5	2.3	256 MB	Low
J1D	1-D Jacobi Solver	13.6	3.2	2048 MB	Low
RED	Reduction Kernel	11.9	5.9	2048 MB	Low
ST	Stencil Operation on 2-D Matrix	48.2	64.7	512 MB	Medium
PR	Page Rank Algorithm	94.4	95.5	1024 MB	Medium
MT	Matrix Transpose	184.4	196.4	512 MB	Medium
SPMV	Sparse Matrix Vector Multiplication	2,012.8	416.2	360 MB	Medium
SYR2	Rank-2k of a Symmetric Matrix	1,265.2	1,044.9	192 MB	High
GUPS	Multi-thread, Random Access	1,399.9	1,147.1	1025 MB	High
SYRK	Rank of a Symmetric Matrix	864.2	1,324.8	512 MB	High
ATAX	Matrix Transpose and Vector Multiplication	2,225.8	1,890.8	64 MB	High
BICG	Sub Kernel of BiCGStab Linear Solver	2,173.6	2,127.9	64 MB	High

Methodology

- Configuration
  - **Baseline**: A conventional GPU that performs page table walks
  - **MPW**: a SOTA hardware page walker design [HPCA 2024]
  - **cuPTW**:
    - **cuPTW**: basic proposal
    - **cuPTW-SW**: enhanced with software-managed page walk cache
    - **cuPTW-MT**: batching translation requests
    - **cuPTW-Full**: combines cuPTW-SW and cuPTW-Full

Evaluation

Overall performance across different configurations

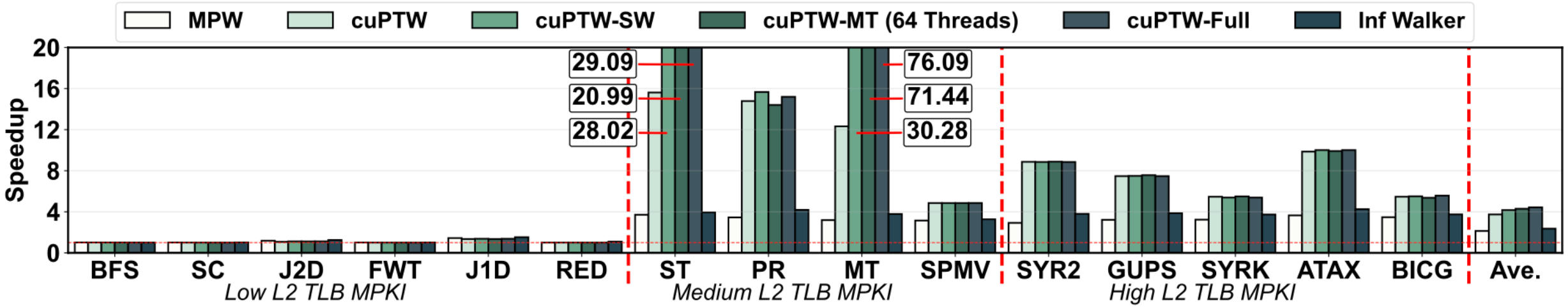


Figure. Speedup for MPW, the cuPTW variants and infinite page walkers, normalized to baseline GPU.

Key Idea: cuPTW, cuPTW-SW, cuPTW-MT and cuPTW-Full achieve a 3.74×, 4.17×, 4.30× and 4.43× geometric mean speedup over baseline GPU, respectively.

Evaluation

Overall performance in a Multi-Chiplet Module GPU setup.

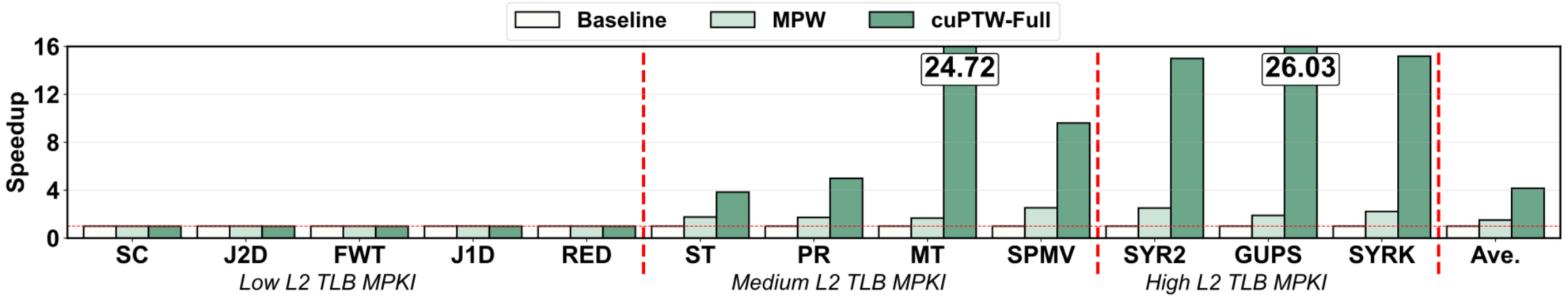


Figure. Speedup for MPW and cuPTW-Full in a Multi-Chiplet Module GPU setup.

Key Idea: cuPTW-Full delivers 4.16× and 2.76× higher performance than the baseline and MPW, respectively in MCM-GPU system.

Evaluation

Compared with super-page schemes

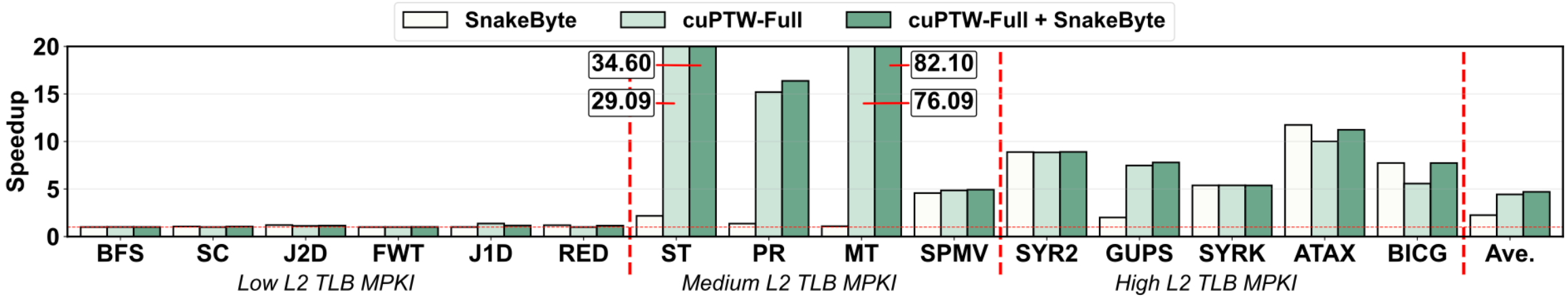


Figure. Speedup for SnakeByte [HPCA 2023], cuPTW-Full, and the combination of cuPTW-Full and SnakeByte, normalized to a baseline GPU with 4 KB pages.

Key Idea: cuPTW-Full outperforms SnakeByte by 1.97×. Furthermore, we evaluated cuPTW on top of SnakeByte, resulting in 2.09× speedup over SnakeByte.

More Results

- Performance Breakdown
  - 3.74x (cuPTW)
  - 4.17x (cuPTW-SW)
  - 4.30x (cuPTW-MT)
  - 4.43x (cuPTW-Full)

More Details in Paper

- Profiling Results
  - 9.92x higher translation throughput
  - 3.38x lower page walk latency

Conclusion

- **Improving GPU page table walk throughput**

- **Key Techniques:** Offloading page table walk requests to idle compute units
  - Leverage underutilized compute units to execute walk request
  - Skip repeated upper-level page table walk by SW-PWC
  - Parallelize translation tasks across multiple SIMD lanes

- **Significant Performance Gains**
  - 4.43x and 2.08x geometric mean speedup over state-of-art page walker design and baseline hardware page walker

Thanks

cuPTW: Leveraging Idle Compute Units for Massively Parallel GPU Page Table Walks

Zihang Chen <zchen097@connect.hkust-gz.edu.cn>,
Tianao Ge, Lieven Eeckhout, Hongyuan Liu, Jiayi Huang

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)