Evaluating Warp Schedulers for Graphics Workloads

Aurora Tomás[1], Juan L. Aragón[2], and Antonio González[1]

[1] Universitat Politècnica de Catalunya

[2] Universidad de Murcia

8th International Workshop on Domain Specific System Architecture (DOSSA-8) @ MICRO 2026

November 1, 2026

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Cofinanciado por la Unión Europea

AGENCIA ESTATAL DE INVESTIGACIÓN

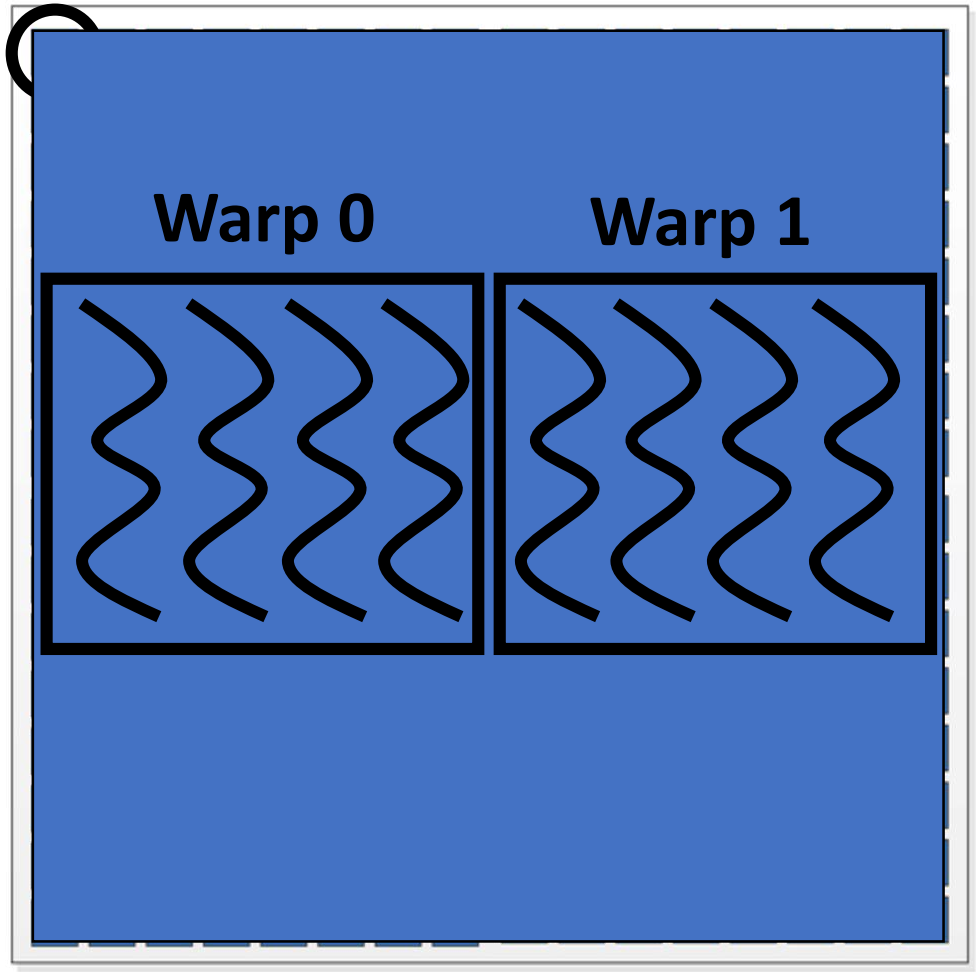
Outline

- Motivation
- Evaluation & Analysis
- Potential Directions
- Conclusion

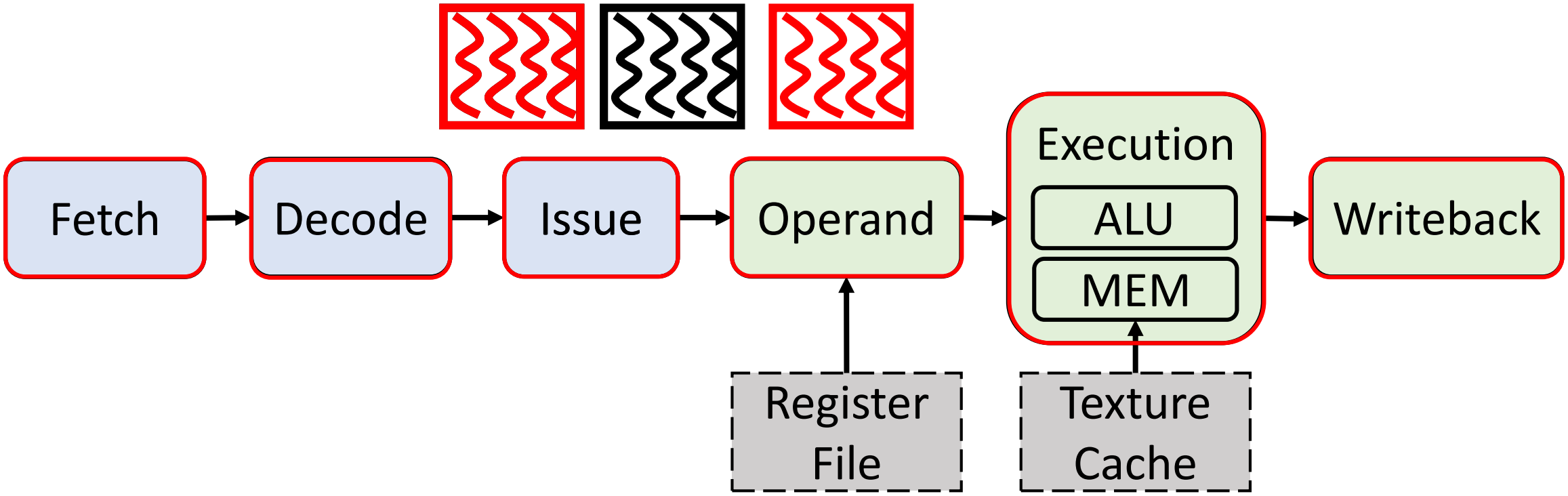
GPU

- Many cores exploit the parallelism of graphics workloads
  - Pixel computations are largely independent
- Large numbers of concurrent threads maximize throughput and hide memory latency (SIMT)
- Threads are grouped into warps, executing in lock-step on SIMD hardware

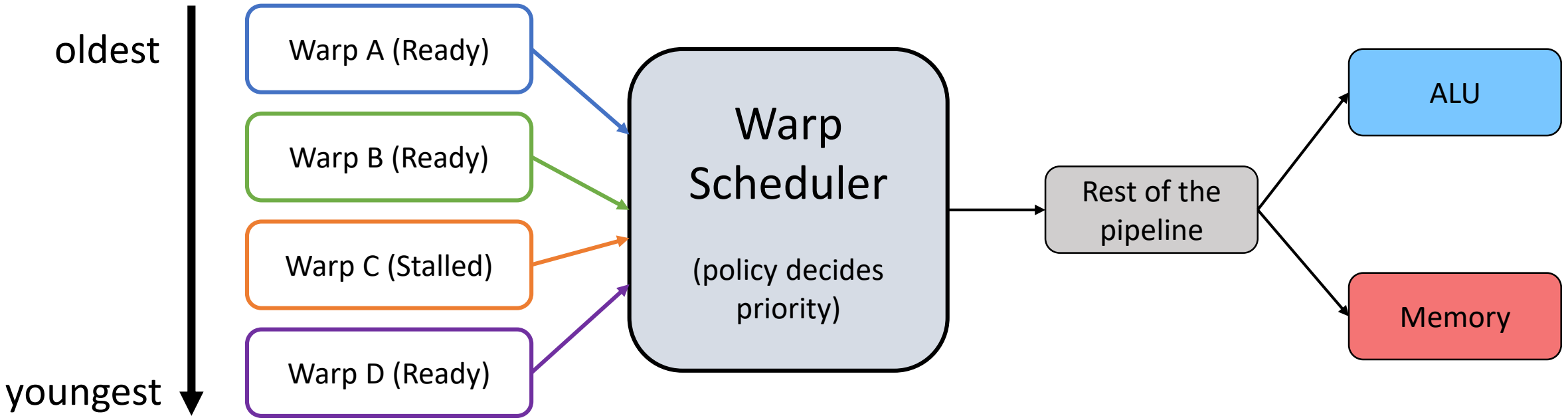
Many cores



Shader Core Microarchitecture



What does the warp scheduler do?



Different policy → Different execution order

Importance of Warp Scheduling

- Scheduling decisions strongly affect:
  - Operand readiness
  - Cache locality
  - Memory pressure
  - Latency hiding
  - Overall executing time

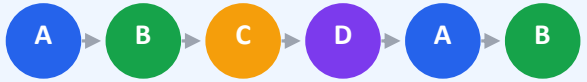
- Significant effort devoted developing novel warp schedulers to improve GPU efficiency



Fixed Policies: LRR, GTO, GTY, OLD

LRR **Loose Round-Robin**

Rotate through ready warps



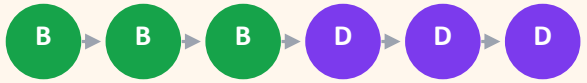
GTO **Greedy-Then-Oldest**

Keep current warp; on stall → oldest



GTY **Greedy-Then-Youngest**

Keep current warp; on stall → youngest



OLD **Oldest-Ready**

Always choose the oldest ready warp



Same situation, different decision

Current warp: B

Ready warps:

A oldest	Ready
B	**Current**
C	Ready
D youngest	Ready

If B can still issue…

LRR	chooses	**C**
GTO	chooses	**B**
GTY	chooses	**B**
OLD	chooses	**A**

If B stalls…

LRR	chooses	**C**
GTO	chooses	**A**
GTY	chooses	**D**
OLD	chooses	**A**

Takeaway

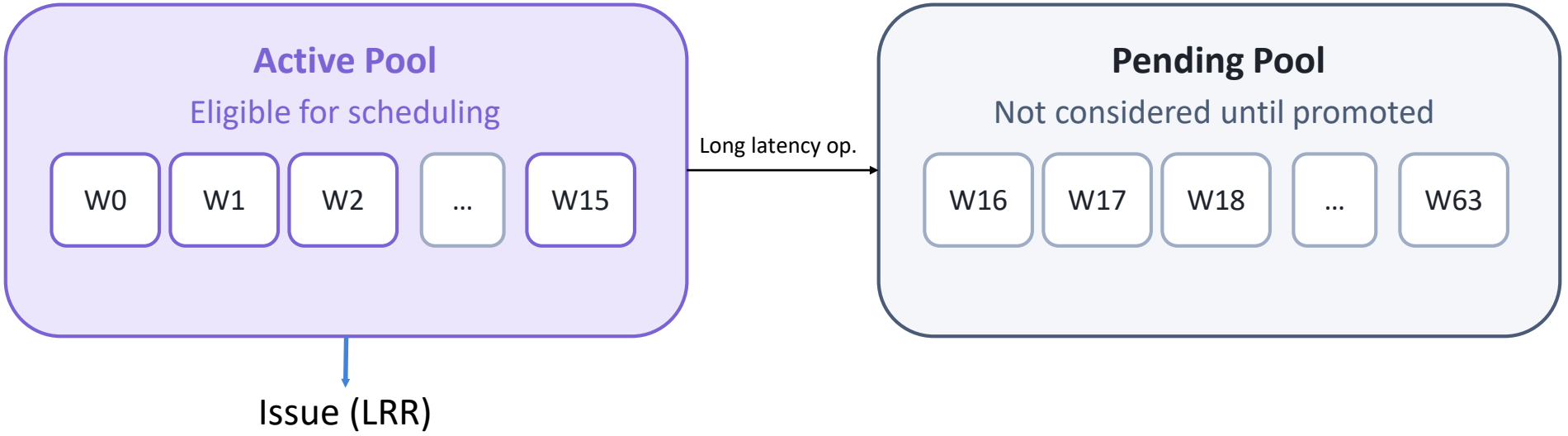
LRR = Fairness

GTO = Intra-warp locality

GTY = Intra-warp locality

OLD = Large waiting-time gaps between warps

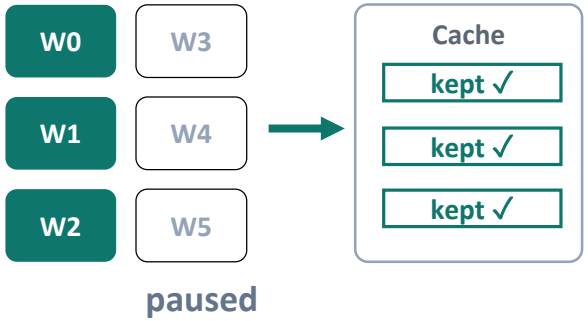
Hierarchical Policy: TLAS



Adaptive Policies: CCWS, CAWS, MASCAR

CCWS Intra-warp locality

Less warps for less eviction



Uses cache feedback to favor warps that preserve intra-warp reuse

CAWS Warp criticality

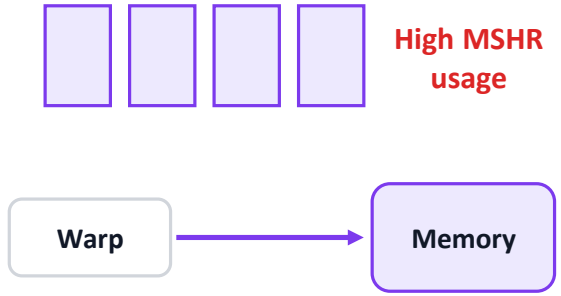
Accelerate lagging warps



Prioritizes critical warps to reduce completion-time disparity

MASCAR Memory pressure

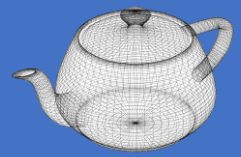
React to memory saturation



When congested, prioritizes one warp's memory requests to make progress and the rest keep computing

Evaluation Methodology

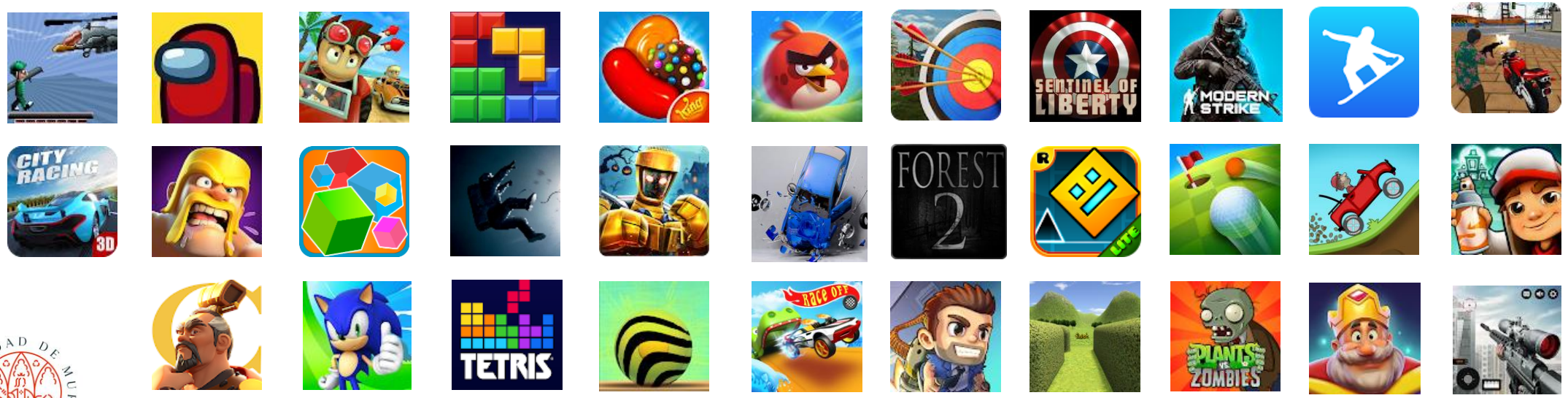
Teapot



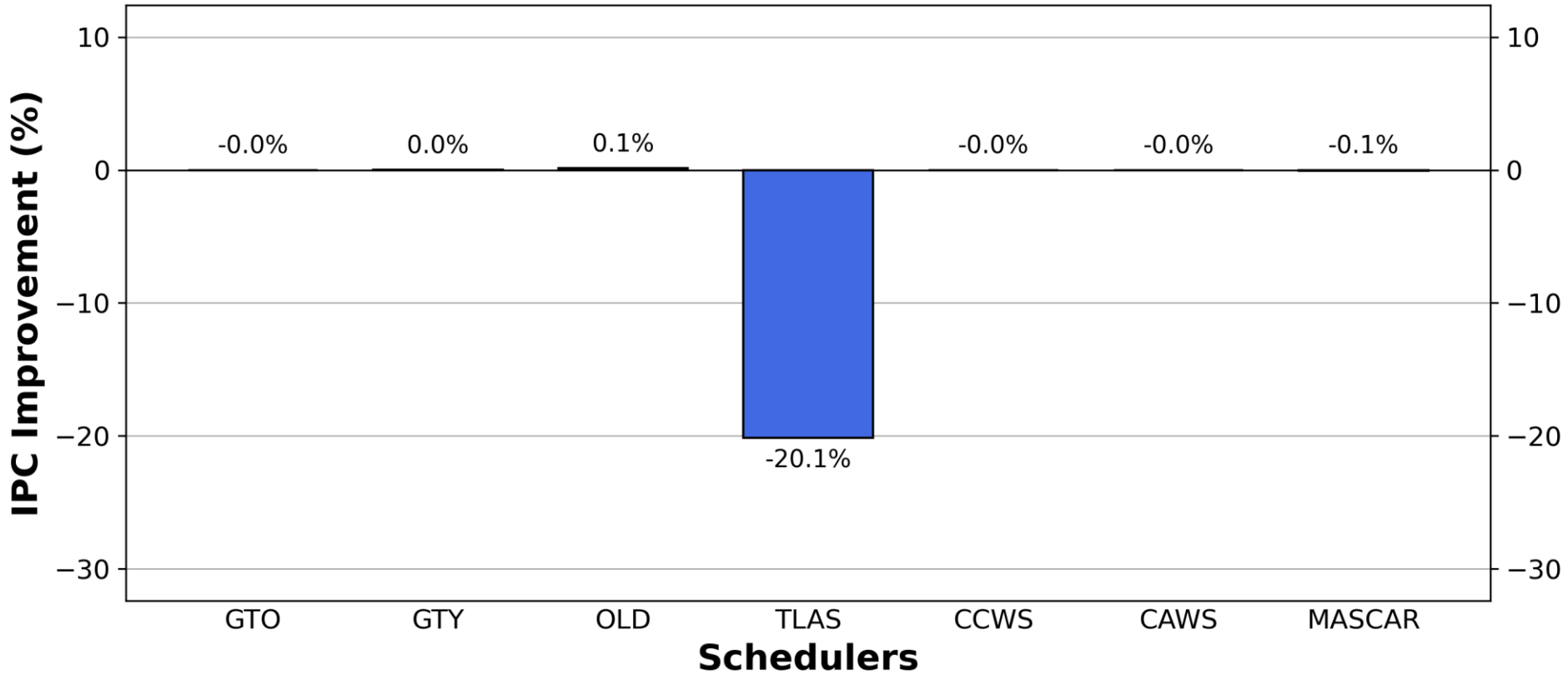
- Cycle-accurate simulator
- Mobile GPUs
- ARM Valhall-like

Evaluation Methodology

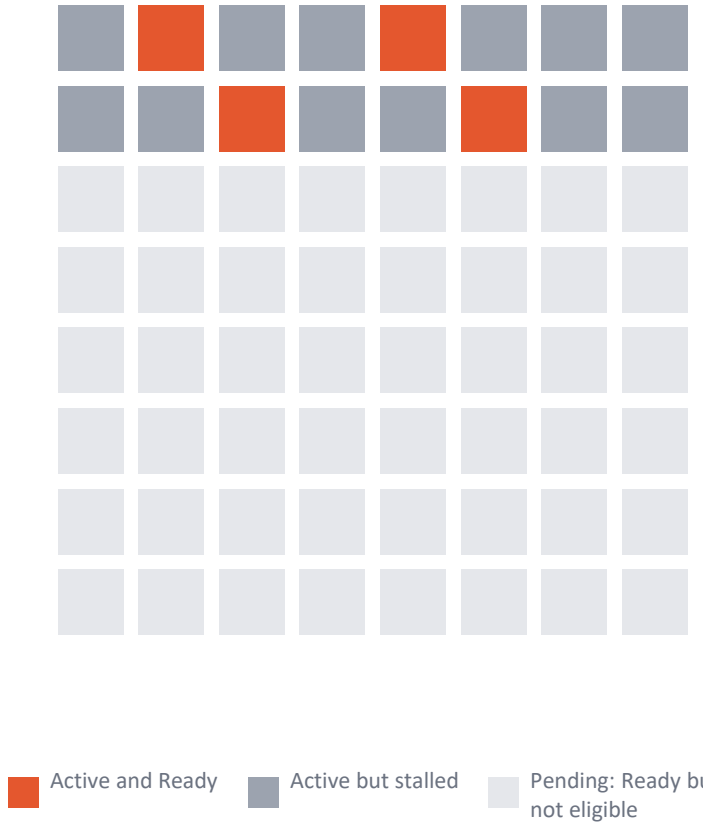
- Benchmarks
  - Scenes from 32 commercial Android graphics applications
  - 2D/3D, different genres, different requirements
  - Significant memory footprint per frame ~ 2X L2 capacity



Performance: No scheduler beats LRR on avg



Why TLAS fails: Too few eligible warps

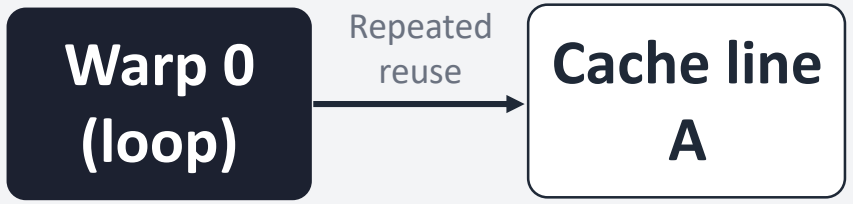


TLAS assumes that a small subset can provide enough ILP/TLP to keep pipelines busy

- Only 1/4 of warps may issue
- Frequent dependencies and memory stalls
- Too few ready warps

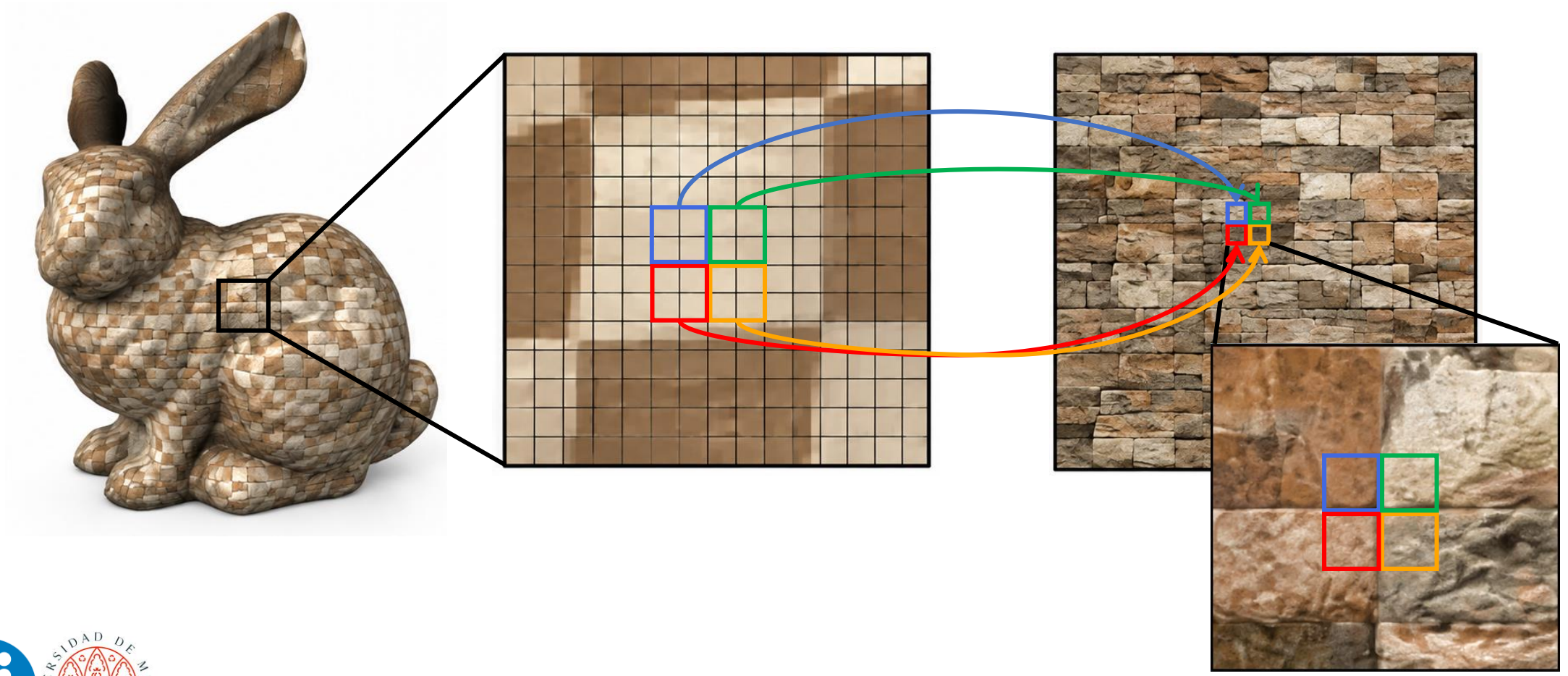
Why GTO, GTY, CCWS fail: Wrong Locality

GPGPU: Intra-warp Reuse



- Keep this warp running (GTO, GTY)
- Throttle others to protect it (CCWS)

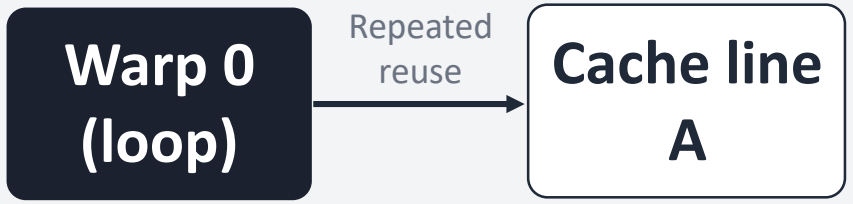
Why GTO, GTY, CCWS fail: Wrong Locality



Same cache line

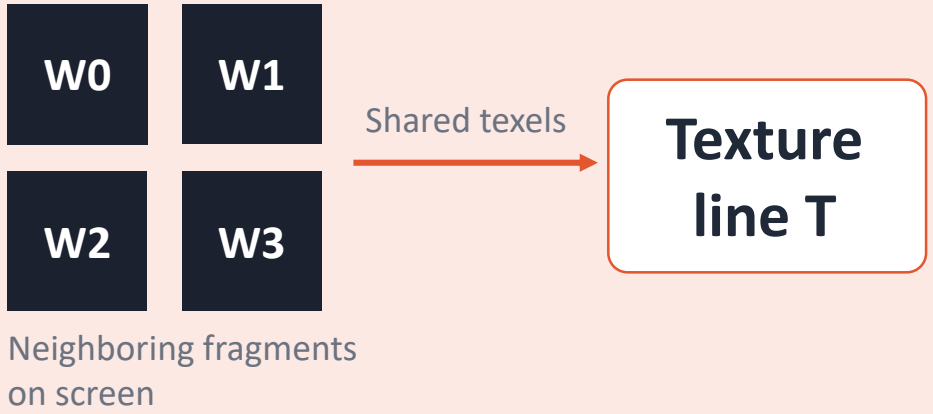
Why GTO, GTY, CCWS fail: Wrong Locality

GPGPU: Intra-warp Reuse



- Keep this warp running (GTO, GTY)
- Throttle others to protect it (CCWS)

Graphics: Inter-warp Reuse



Neighboring fragments share the same (or nearby) texture data

Their locality target does not match the dominant reuse pattern in graphics

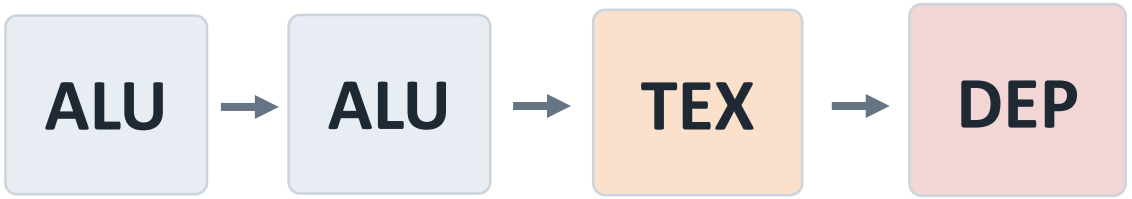
Why OLD and CAWS fail: Uncommon Patterns

- They target imbalance patterns less frequent in graphics workloads
  - OLD: Large disparity in waiting time between eligible warps
  - CAWS: Large completion-time disparities among warps within the same CTA

Why MASCAR fails: Instruction Dependencies

- Relies on a warp issuing multiple memory requests until reaching a load-dependent instruction

Typical shader program pattern



Texture use is often followed immediately by a dependent instruction

GPGPU Kernels vs. Graphics Shaders

	Typical GPGPU	Graphics (this study)
Program	Fewer, long kernels, often with loops	**Many, short warps (~ 13 instructions on average)**
Memory ins / warp	Many; can be issued in bursts	**Few**
Dependencies	Consumer can be far away	**Next instruction depends on it**
Locality opportunity	Intra-warp (warp revisits its lines)	**Inter-warp (neighbor fragments share texels)**

Potential Directions

- Graphics: **Enormous** number of very **short-lived warp**s, with locality driven by **inter-warp** reuse
- GPGPU: **Fewer**, **long-lived warps**, with locality driven by **intra-warp** reuse
- Future schedulers may incorporate information about recently accessed memory regions to identify short-term reuse opportunities across warps

Conclusion

- Existing warp schedulers fail to improve LRR for graphics workloads
- GPGPU assumptions do not match graphics
  - Short shaders, immediate dependencies and inter-warp reuse
- **Exploiting inter-warp** awareness for scheduling remains unexplored in graphics applications

More in the paper

Evaluating Warp Schedulers for Graphics Workloads

Aurora Tomás[1], Juan L. Aragón[2] and Antonio González[1]

[1]Universitat Politècnica de Catalunya
[2]Universidad de Murcia

Evaluating Warp Schedulers for Graphics Workloads

Aurora Tomás[1], Juan L. Aragón[2], and Antonio González[1]

[1] Universitat Politècnica de Catalunya

[2] Universidad de Murcia

8th International Workshop on Domain Specific System Architecture (DOSSA-8) @ MICRO 2026

November 1, 2026

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Cofinanciado por la Unión Europea

AGENCIA ESTATAL DE INVESTIGACIÓN