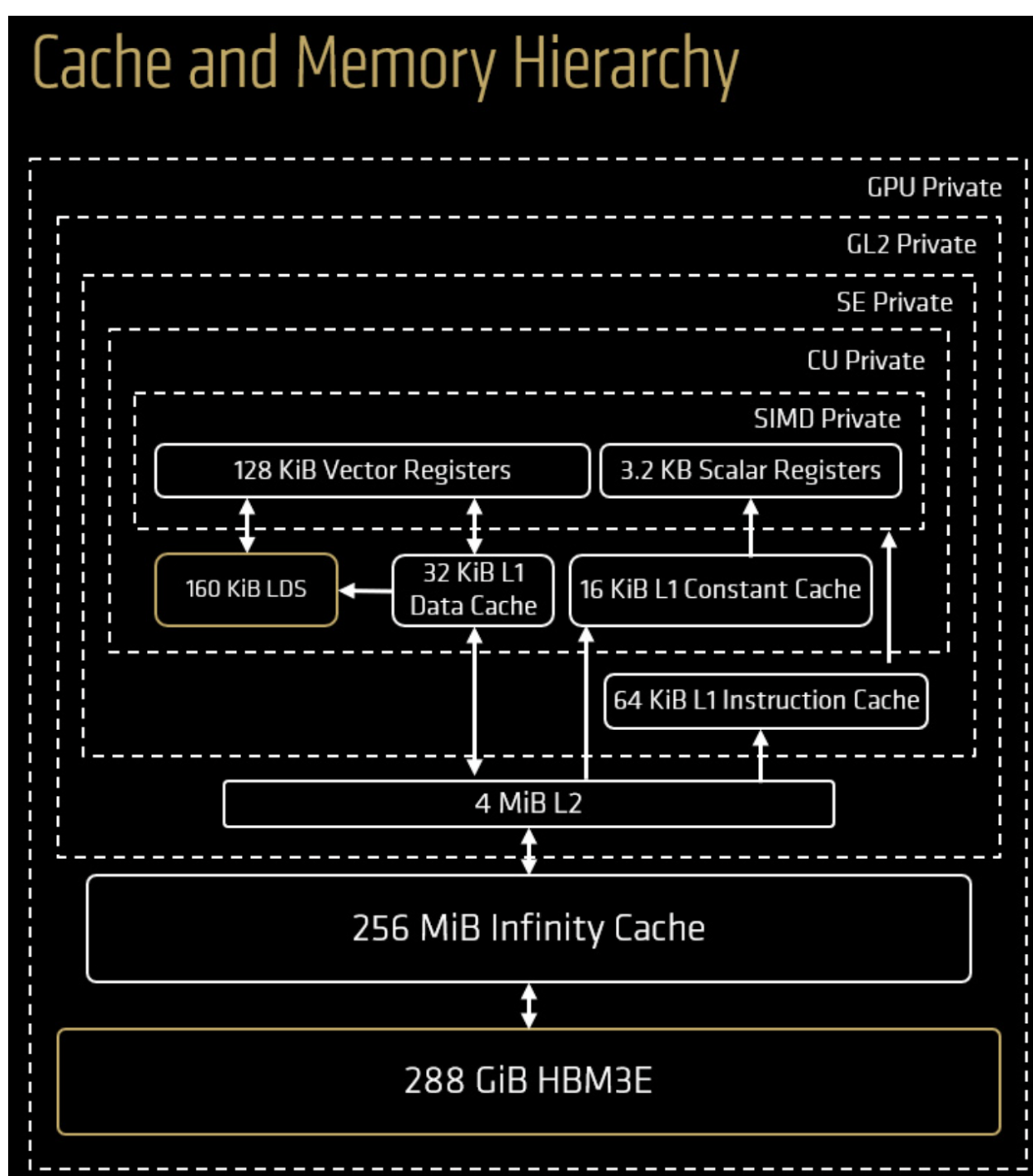


Introduction

Composable Kernel (CK) is AMD's open-source (MIT), header-only HIP C++ template library for high-performance GPU operators — the production backend behind ROCm's GEMM, convolution, FlashAttention, fused-MoE etc.



Two layers, one source tree. Legacy CK exposes a deep template hierarchy (DeviceGemm* → GridwiseGemm* → BlockwiseGemm*) parameterized by tile sizes and MFMA instruction. `ck_tile` is the newer tile-programming DSL: *distributed tensors* plus explicit `load_tile / store_tile / shuffle_tile` primitives, organized into four composable tiers

Tier	Scope	Owens
warp	1 wavefront (64 lanes)	a single MFMA shape; per-wave A/B/C register distribution
block	1 workgroup	tiling the warp op; LDS staging and barriers
pipeline	Block over the K-loop	software pipelining, prefetch, double-buffering, MFMA/global overlap
kernel	The grid	tile→workgroup scheduling, epilogue fusion, launch config

Why this matters for HSTU. The four tiers map one-to-one onto the degrees of freedom our kernel needs and Triton does not expose: warp-tier control over WarpGemm register layouts (register-resident Q/S/O, fused two-GEMM chaining with no LDS round-trip), block-tier LDS control (XOR-swizzled four-deep K/V ring), pipeline-tier scheduling (transposed V loads + prefetch), and kernel-tier partitioning (mask-range skipping, XCD balancing, split-KV).

	Triton	Composable Kernel
Model	Python DSL	C++ template
Register layout	Compiler-chosen	Explicit, per-warp
LDS layout & swizzle	Implicit	explicit
GEMM chaining	Via LDS/shared round-trip	Register-resident, matched WarpGemm layouts
Pipeline schedule	limited	Intrawave (compute-bound)/ Interwave (mem-bound)
Tile -> CU mapping	Program-id arithmetic	TilePartitioner: 2D / 1D / SpatiallyLocal + RemapXCD
Arch-specific ISA	partial	First-class

HSTU and Semi-local Attention

ULTRA-HSTU is fully deployed at scale, serving billions of users daily, and delivers 4%–8% consumption and engagement improvements in production A/B tests. Generative recommender models the user's interaction history as a *sequence* and transduces it, e.g. what a person watched, liked and skipped.

HSTU is the attention primitive at its core. It encodes a user-interaction-history sequence of thousands to tens of thousands of events, then lets candidate items attend to that encoded history. Histories run to 16K events, so the attention operator dominates inference cost.

HSTU attention is not softmax attention

$$\text{Attn}(Q, K, V) = (\text{SiLU}(\alpha \cdot Q K^T) \odot M) V$$
$$\text{SiLU}(x) = x \cdot \sigma(x)$$

- No row normalization** – no running max/sum to carry across K tiles
- Purely pointwise** – no reduction along the K axis. Score computation, activation, masking and the V multiply fuse into one register-resident loop.

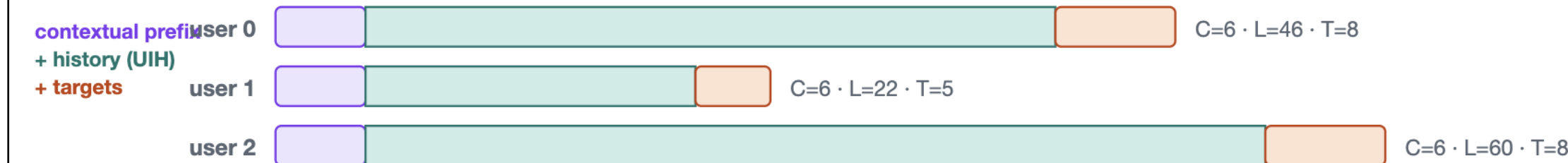
Semi-local attention: $O(n^2) \rightarrow O(n)$

Doubling the history quadruples attention compute. Scaling histories from 8K to 16K events pushed inference past the point of being servable.

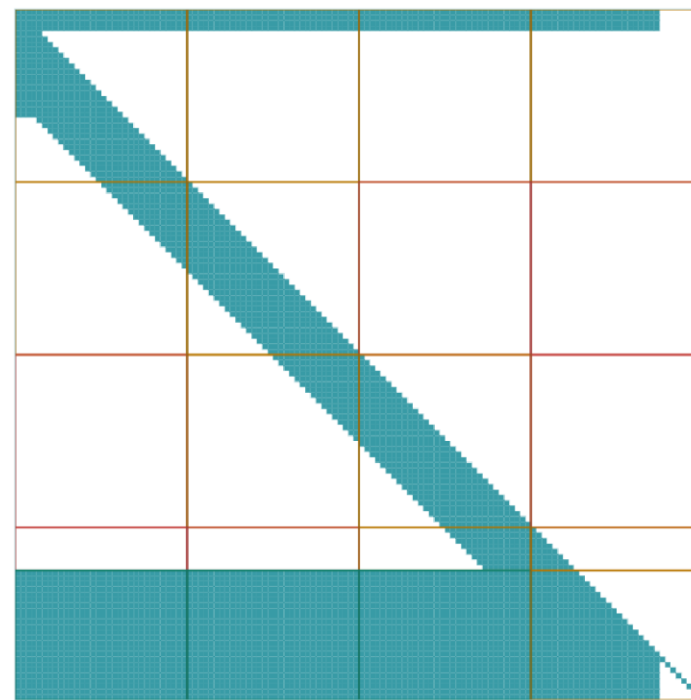
Semi-local attention (SLA) is a hybrid of local sliding-window and sparse global attention. Three parameters compose into one functional mask over jagged inputs:

- Max_attn_len: Sliding window.** Query i attends only to keys in $(i-W, i]$. Makes long histories cheap.
- min_full_attn_seq_len: Dense recent tail.** The most recent N queries get full causal attention regardless of the window — recent behavior is where the signal is.
- contextual_seq_len: Global prefix.** Every query may attend to the contextual tokens.

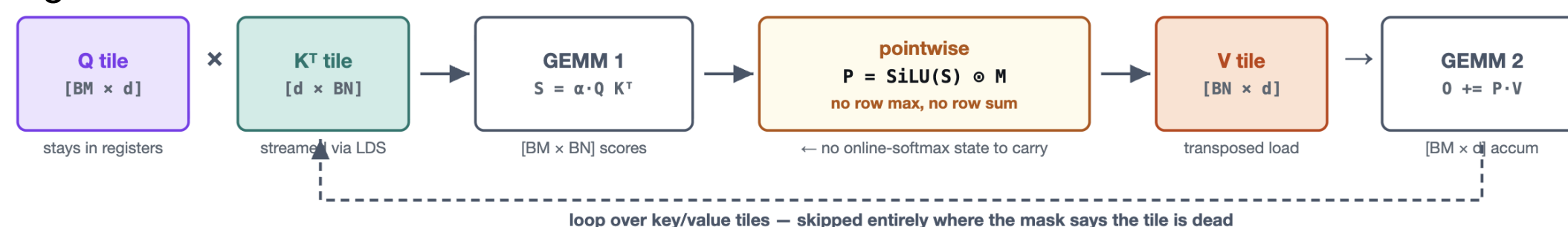
Jagged input - every user a different length. no padding



Semi-local mask

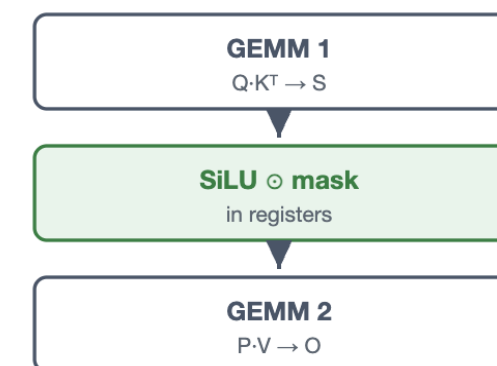


Tiling and the two chained GEMMs



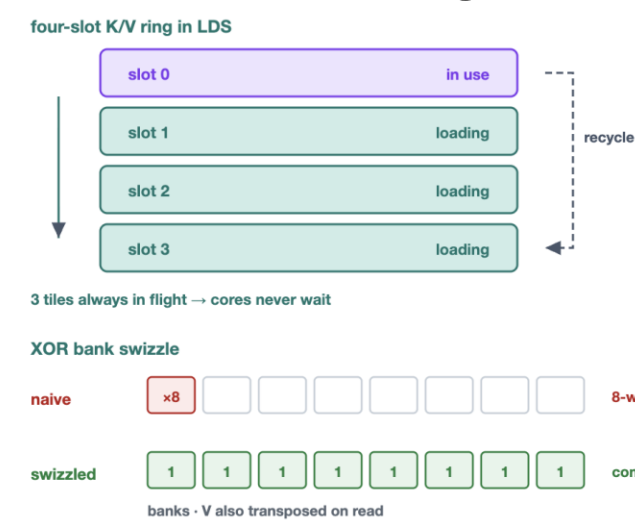
Optimizations and Results

1. Register-resident operands



2nd GEMM's operand layout is chosen to match the 1st GEMM's output layout, so output all live in VGPR for the duration of the KV loop.

2. Swizzled LDS ring + prefetch



K/V move through a four-slot ring in LDS. Addresses are XOR-swizzled to avoid bank conflict, and overlaps VMEM loads with MFMA matrix-core issue.

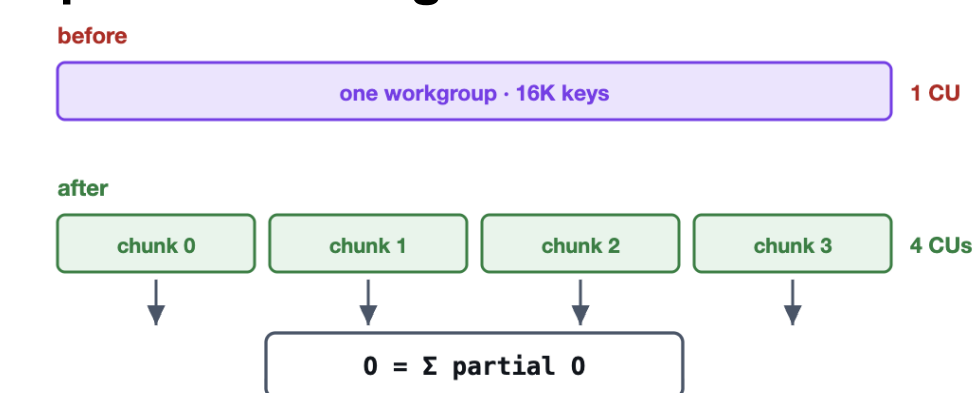
3. Sparsity- and occupancy-aware scheduling

Mask-range skipping and whole-tile fast paths exploit local/causal sparsity; runtime tile selection and grid ordering balance XCD workload and maximize L2 reuse

4. Explicit register placement of thread indices

Through purpose-built tile-window interfaces we force the `warp_id` into a vector register (VGPR) rather than a scalar register (SGPR) when addressing LDS. Since the LDS store/load offsets of a distributed tile are inherently per-lane, keeping the `warp_id` in a VGPR lets the whole index computation run in the vector path — avoiding repeated cross-lane scalarization and SGPR↔VGPR moves inside the hot loop.

5. Split-KV for long-context cross attention



6. Full PyTorch integration

Tile shapes, MFMA instruction choice are selected per architecture at compile time. The kernel is registered as a PyTorch operator compatible with PT2 and AOTInductor, so produc

