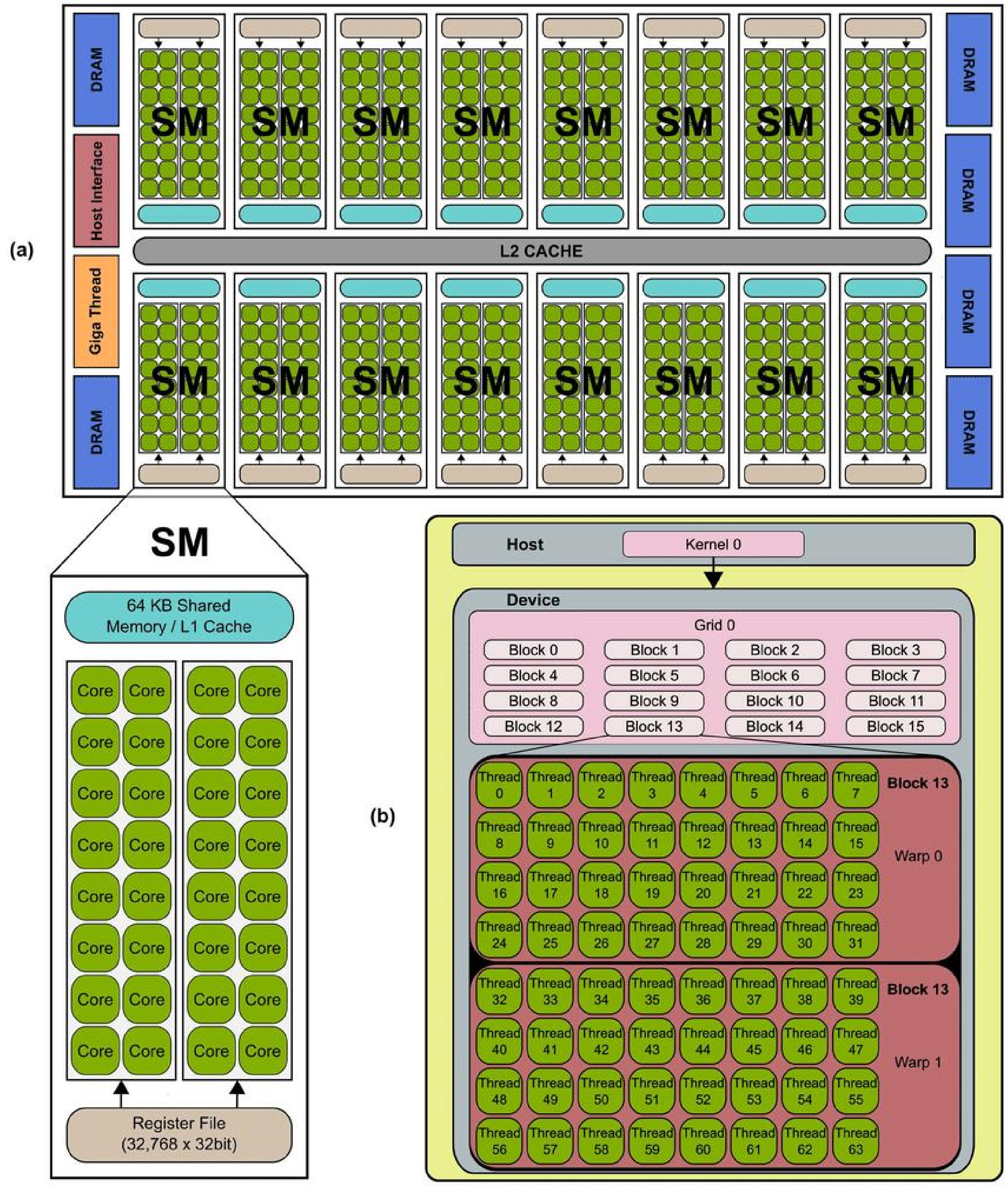
vLLM
KV cache & profile

Chenye Wang

Nov 26, 2024

GPU Architecture

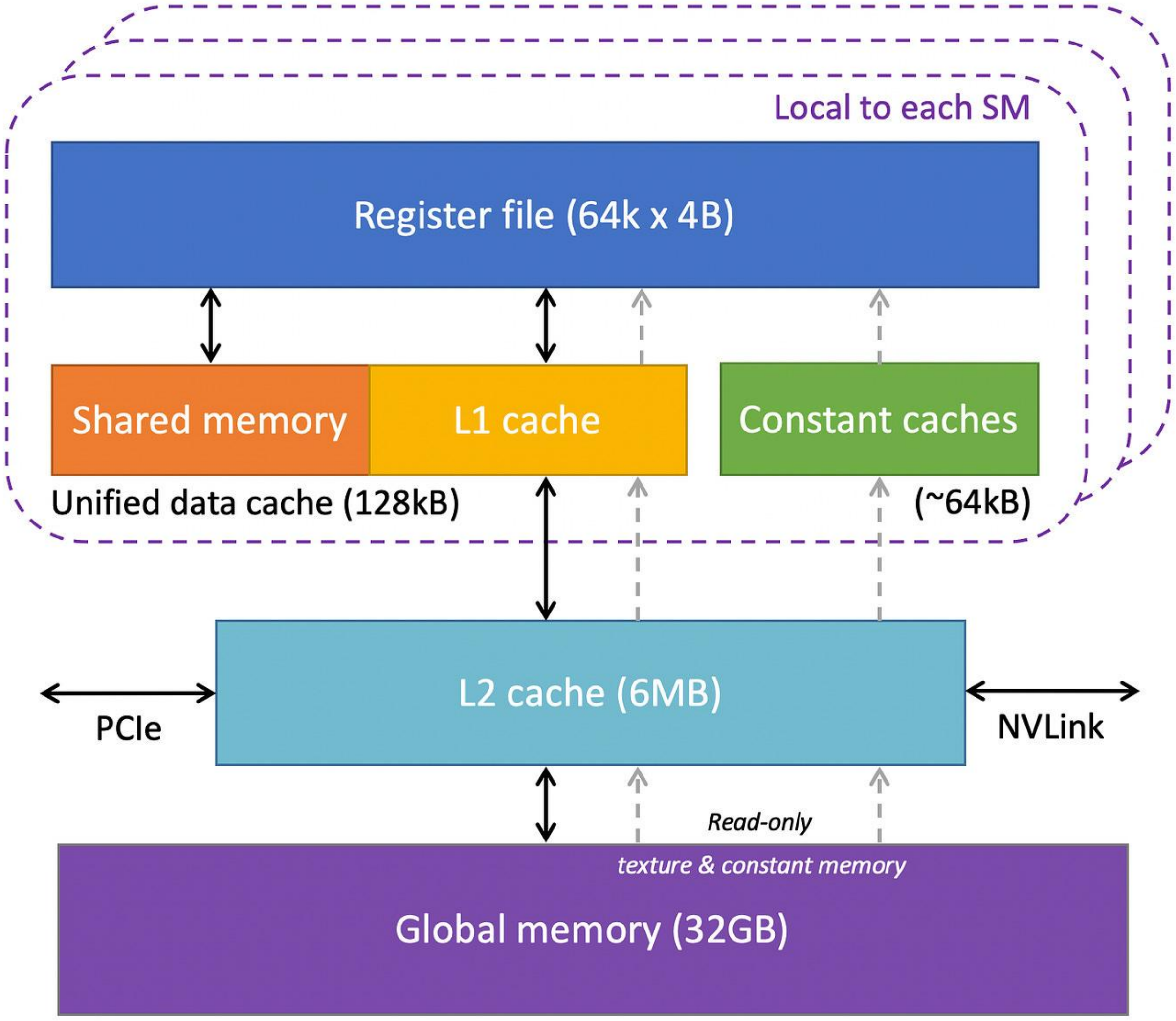
- Streaming Multiprocessor (SM)
  - Stream Processor (SP) core
  - registers
  - shared memory / L1 cache
  - thread block slot
  - thread slot
- CUDA programming model
  - grid
    - thread block
      - warp
        - thread



[1] https://agenda.infn.it/event/7260/contributions/66624/attachments/48346/57204/GPU_NMR_notes_2.pdf

Memory Architecture

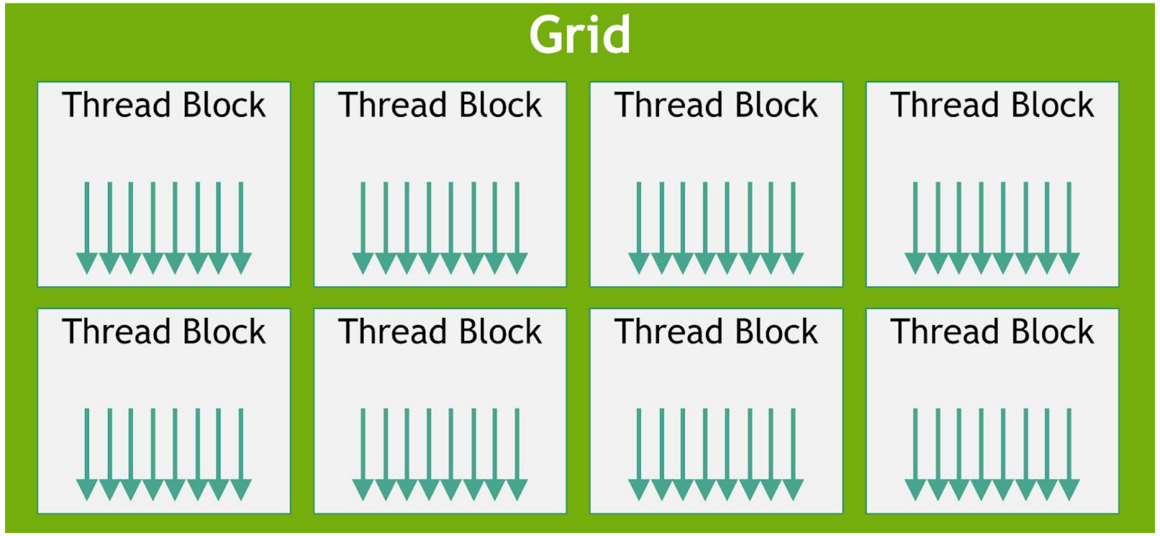
- Registers
  - allocated to threads dynamically
  - zero-overhead scheduling
- Shared Memory
  - only load shared data once
  - threads synchronization
- L2 Cache
  - shared by all SMs
- Global Memory
  - copy data from CPU

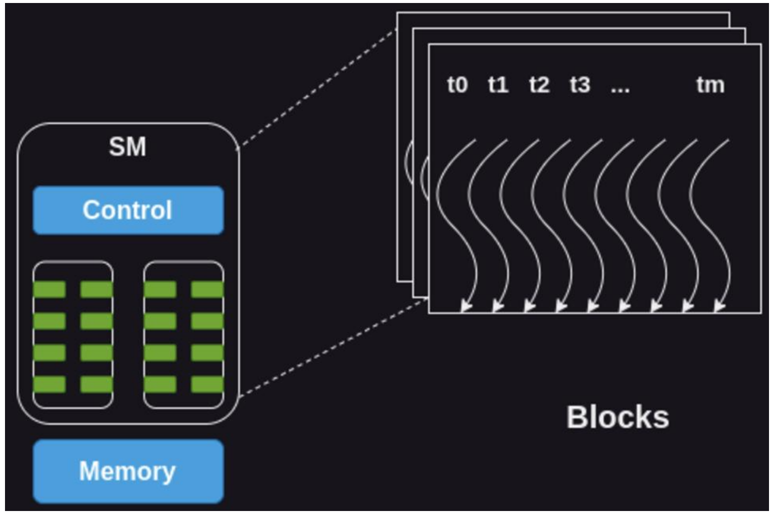


[1] https://cvw.cac.cornell.edu/gpu-architecture/gpu-memory/memory_levels

Compute Architecture

- To execute a kernel on GPU
  - launch a grid with threads
- The number of thread blocks and threads depends on
  - size of the data
  - amount of parallelism
  - limit of SM resources
- Warp -- 32 threads
  - execute on a set of cores
  - Single Instruction Multiple Threads (SIMT)

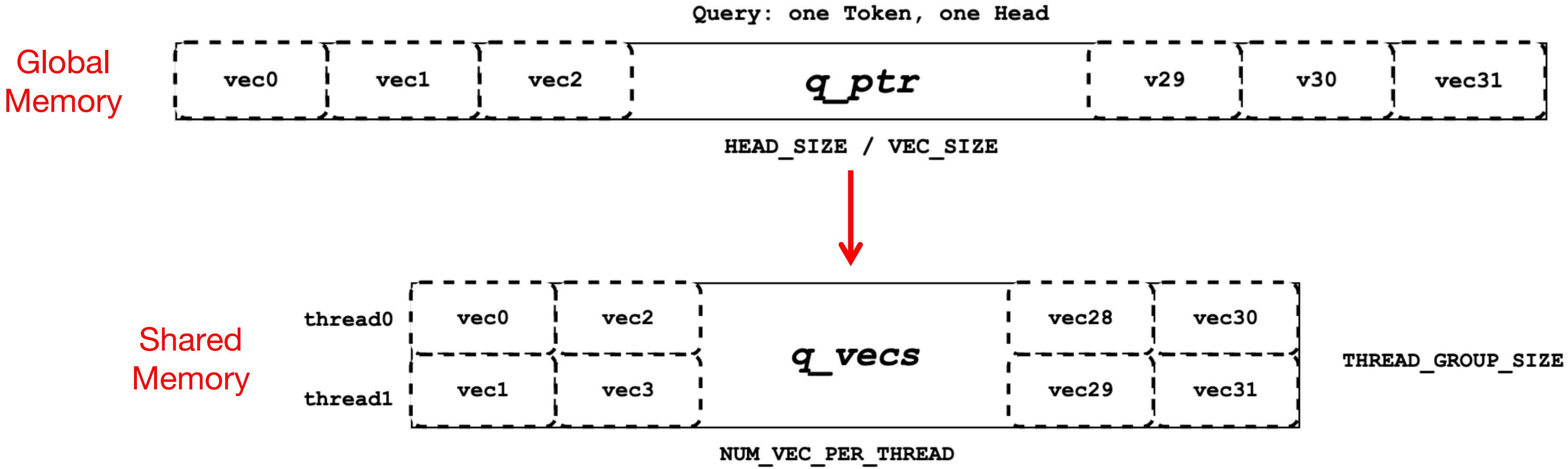




[1] https://blog.codingconfessions.com/p/gpu-computing

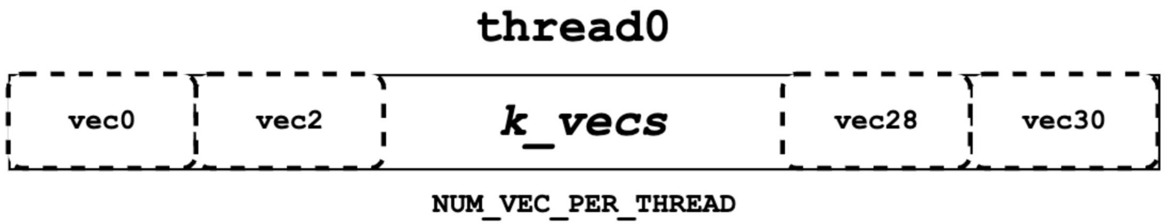
Query memory layout and assignment

- Thread group: one query token and one key token
- HEAD_SIZE = 128
- VEC_SIZE = 4

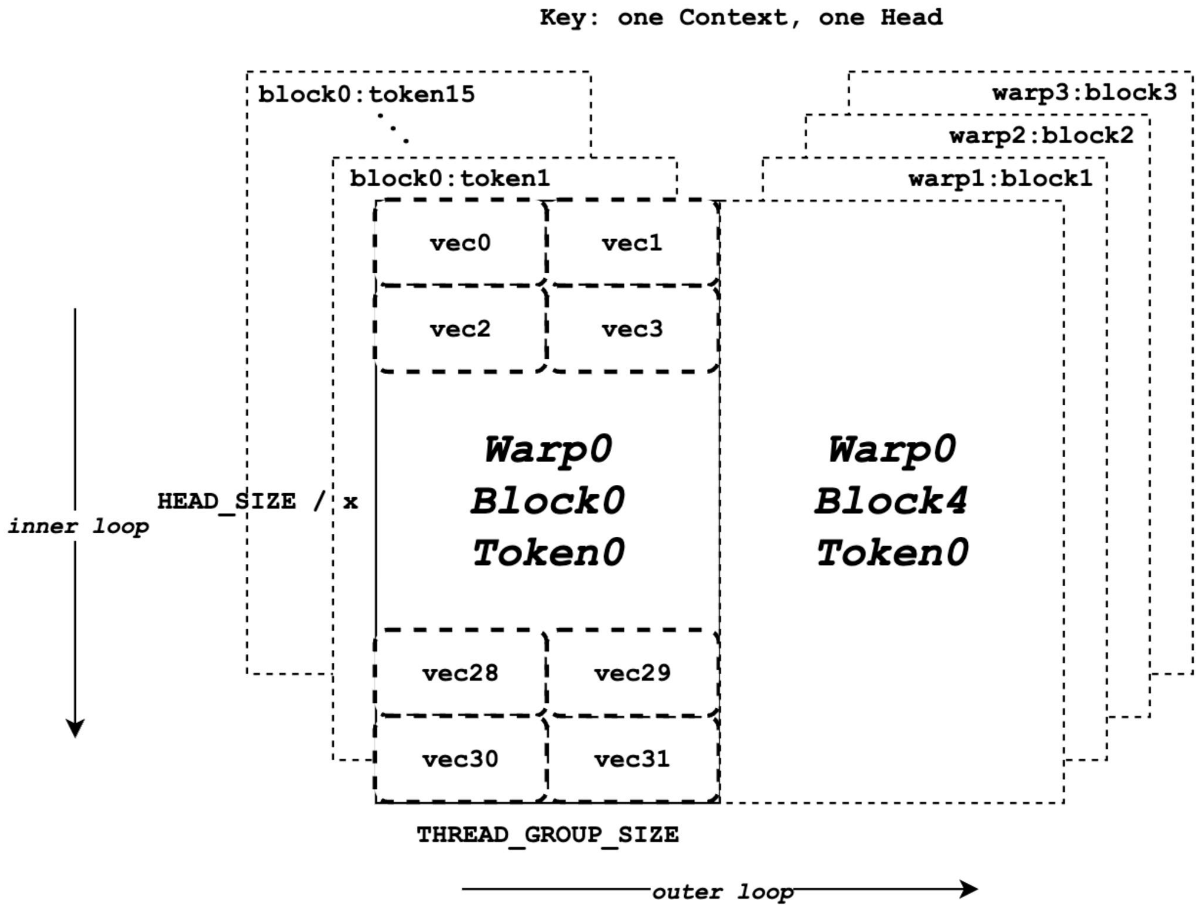


Key memory layout and assignment

- Warp: one query token and a block of key tokens
- BLOCK_SIZE = 16
- x = 8



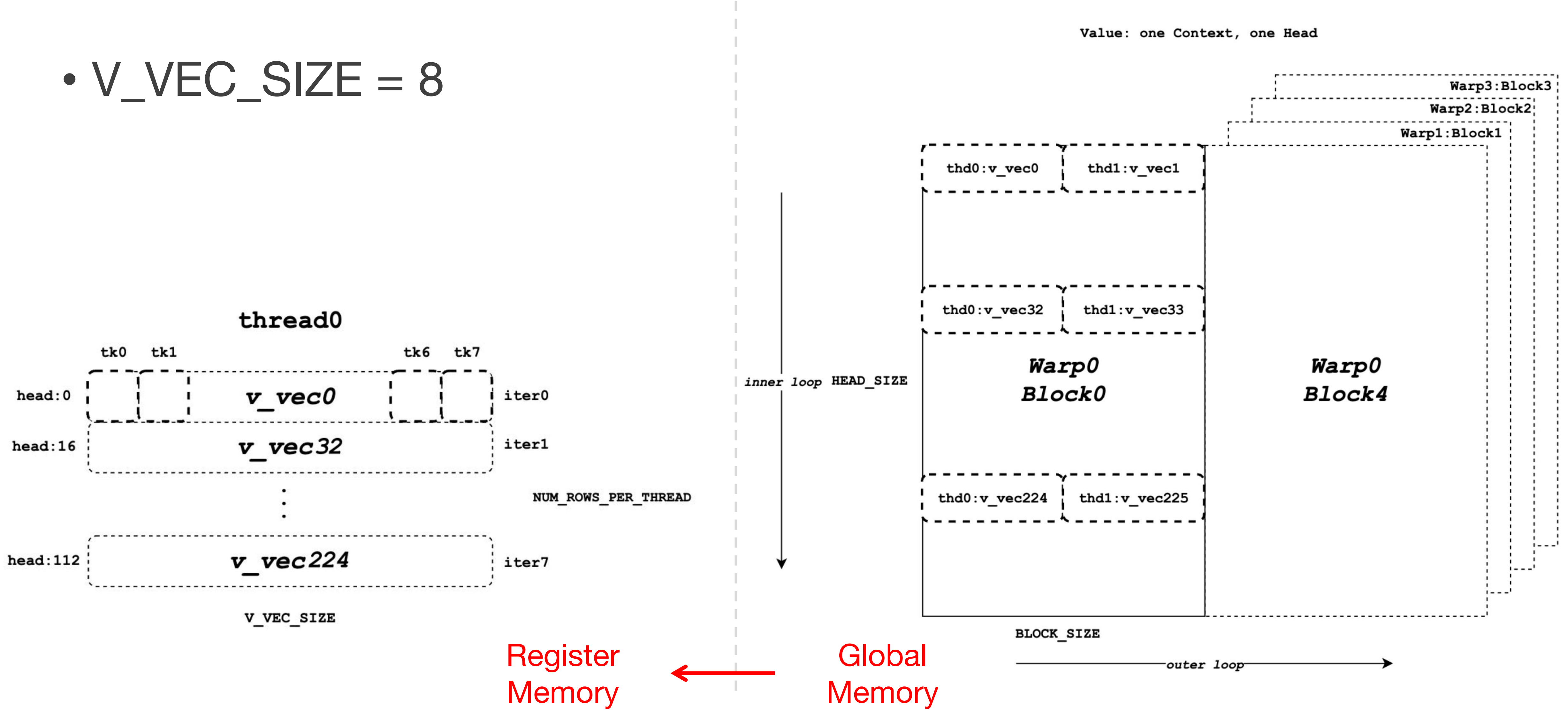
Register Memory



Global Memory

Value memory layout and assignment

- V_VEC_SIZE = 8



How much space does the cache block need?

- one token
  - key and value
  - multiple layers
  - multiple heads
  - head size of elements

```python
# vllm/worker/cache_engine.py
class CacheEngine:
    def get_cache_block_size(
        ...
    ) -> int:
        head_size = model_config.get_head_size()
        num_heads = model_config.get_num_kv_heads(parallel_config)
        num_attention_layers = model_config.get_num_attention_layers(
            parallel_config)

        key_cache_block = cache_config.block_size * num_heads * head_size
        value_cache_block = key_cache_block
        total = num_attention_layers * (key_cache_block + value_cache_block)
        ...
        dtype_size = get_dtype_size(dtype)
        return dtype_size * total
```

How much memory is available for KV blocks?

- Profile peak memory usage other than KV cache
- Allocate remaining free memory for KV cache

```python
# vllm/worker/worker.py
class Worker(LocalOrDistributedWorkerBase):
    def determine_num_available_blocks(self) -> Tuple[int, int]:
        torch.cuda.empty_cache()
        self.model_runner.profile_run()
        torch.cuda.synchronize()

        free_gpu_memory, total_gpu_memory = torch.cuda.mem_get_info()
        peak_memory = self.init_gpu_memory - free_gpu_memory
        ...
        cache_block_size = self.get_cache_block_size_bytes()
        num_gpu_blocks = int(
            (total_gpu_memory * self.cache_config.gpu_memory_utilization -
             peak_memory) // cache_block_size)
        num_cpu_blocks = int(self.cache_config.swap_space_bytes //
                             cache_block_size)
        num_gpu_blocks = max(num_gpu_blocks, 0)
        num_cpu_blocks = max(num_cpu_blocks, 0)
        ...

        torch.cuda.empty_cache()
        return num_gpu_blocks, num_cpu_blocks
```

Profile the memory usage of the model

```python
# vllm/worker/model_runner.py
class GPUModelRunnerBase(ModelRunnerBase[TModelInputForGPU]):
    def profile_run(self) -> None:
        ...
        max_num_batched_tokens = self.scheduler_config.max_num_batched_tokens
        max_num_seqs = self.scheduler_config.max_num_seqs
        ...
        # Profile memory usage with max_num_sequences sequences and the total
        # number of tokens equal to max_num_batched_tokens.
        seqs: List[SequenceGroupMetadata] = []
        ...
        for group_id in range(max_num_seqs):
            seq_len = (max_num_batched_tokens // max_num_seqs +
                       (group_id < max_num_batched_tokens % max_num_seqs))
            ...
            seq = SequenceGroupMetadata(
                ...
                block_tables=None,
                ...
            )
            seqs.append(seq)
```

```python
        # Run the model with the dummy inputs.
        num_layers = self.model_config.get_num_layers(self.parallel_config)
        kv_caches = [
            torch.tensor([], dtype=torch.float32, device=self.device)
            for _ in range(num_layers)
        ]
        finished_requests_ids = [seq.request_id for seq in seqs]
        model_input = self.prepare_model_input(
            seqs, finished_requests_ids=finished_requests_ids)
        ...
        self.execute_model(model_input, kv_caches, ...)
        torch.cuda.synchronize()
        return
```

Summary

- Parameters that affect the number of KV blocks
  - model config
    - head size
    - number of heads
    - number of attention layers
  - cache config
    - gpu memory utilization
  - scheduler config
    - max_num_batched_tokens
    - max_num_seqs