Cache Refill/Access Decoupling for Vector Machines

Christopher Batten, Ronny Krashinsky, Steve Gerding, Krste Asanović

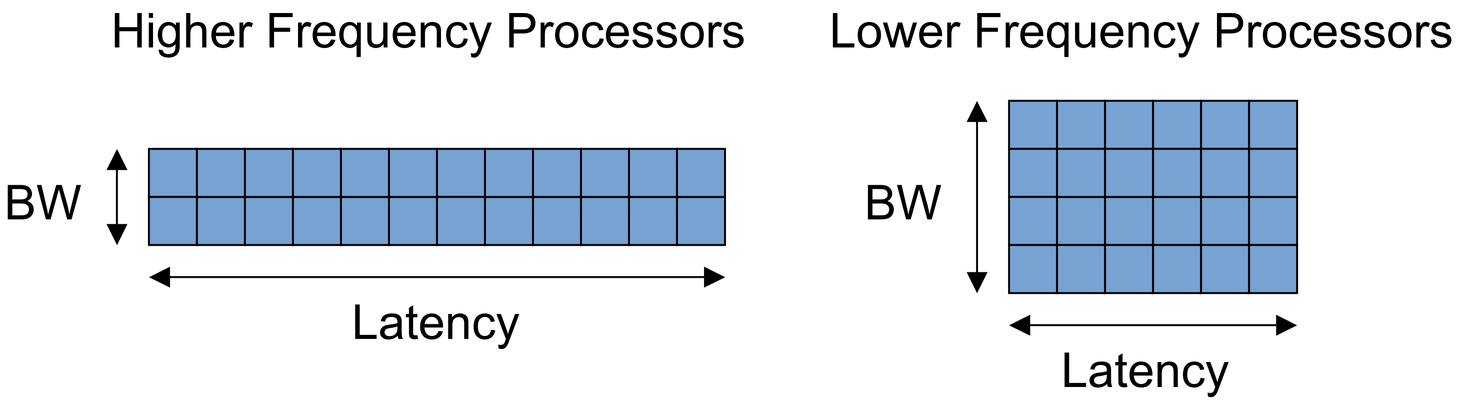
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
September 23, 2004

Outline

- Motivation
  - Large bandwidth-delay product memory systems
  - Access parallelism and resource requirements
- The SCALE memory system
  - Baseline SCALE memory system
  - Refill/access decoupling
  - Vector segment accesses
- Evaluation
- Conclusions

Bandwidth-Delay Product

- Modern memory systems
  - Increasing latency: Higher frequency processors
  - Increasing bandwidth: DDR, highly pipelined, interleaved banks
- These trends combine to yield very large and growing bandwidth-delay products
  - Number of bytes of memory bandwidth per processor cycle times the number of processor cycles for a round trip memory access
  - To saturate such memory systems, processors must be able to generate and manage **many hundreds** of outstanding elements



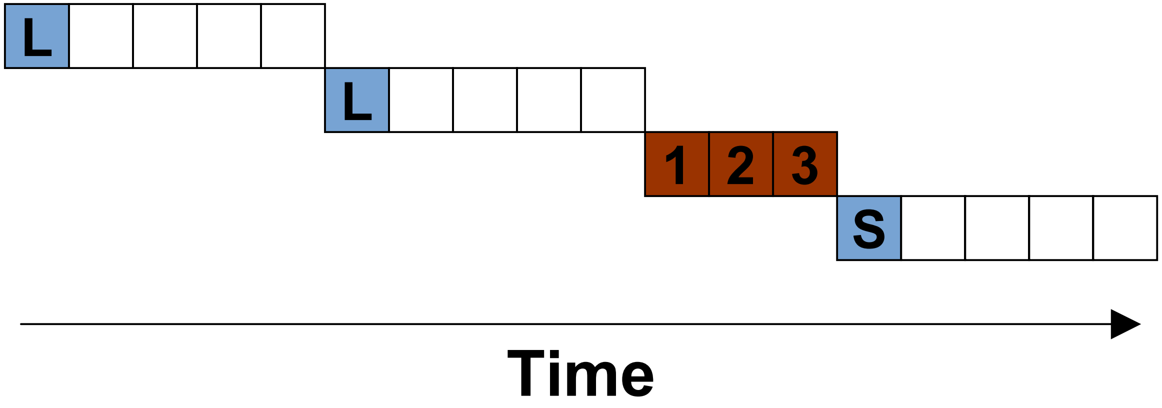
Higher Frequency Processors

Lower Frequency Processors

Access Parallelism

- Memory accesses which are independent and thus can be performed in parallel exhibit **access parallelism**
- The addresses of such accesses are usually known well in advance
- We can exploit access parallelism to saturate large bandwidth-delay memory systems

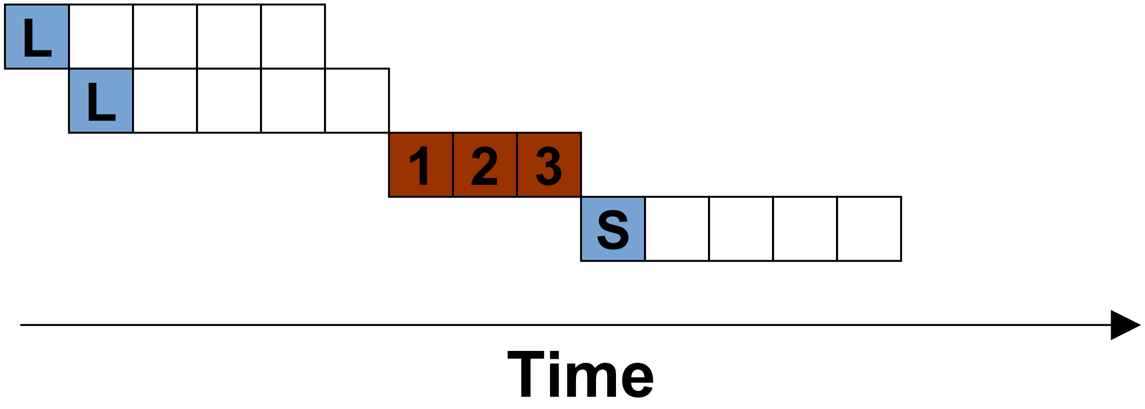
```
loop
  load
  load
  compute
  store
end
```



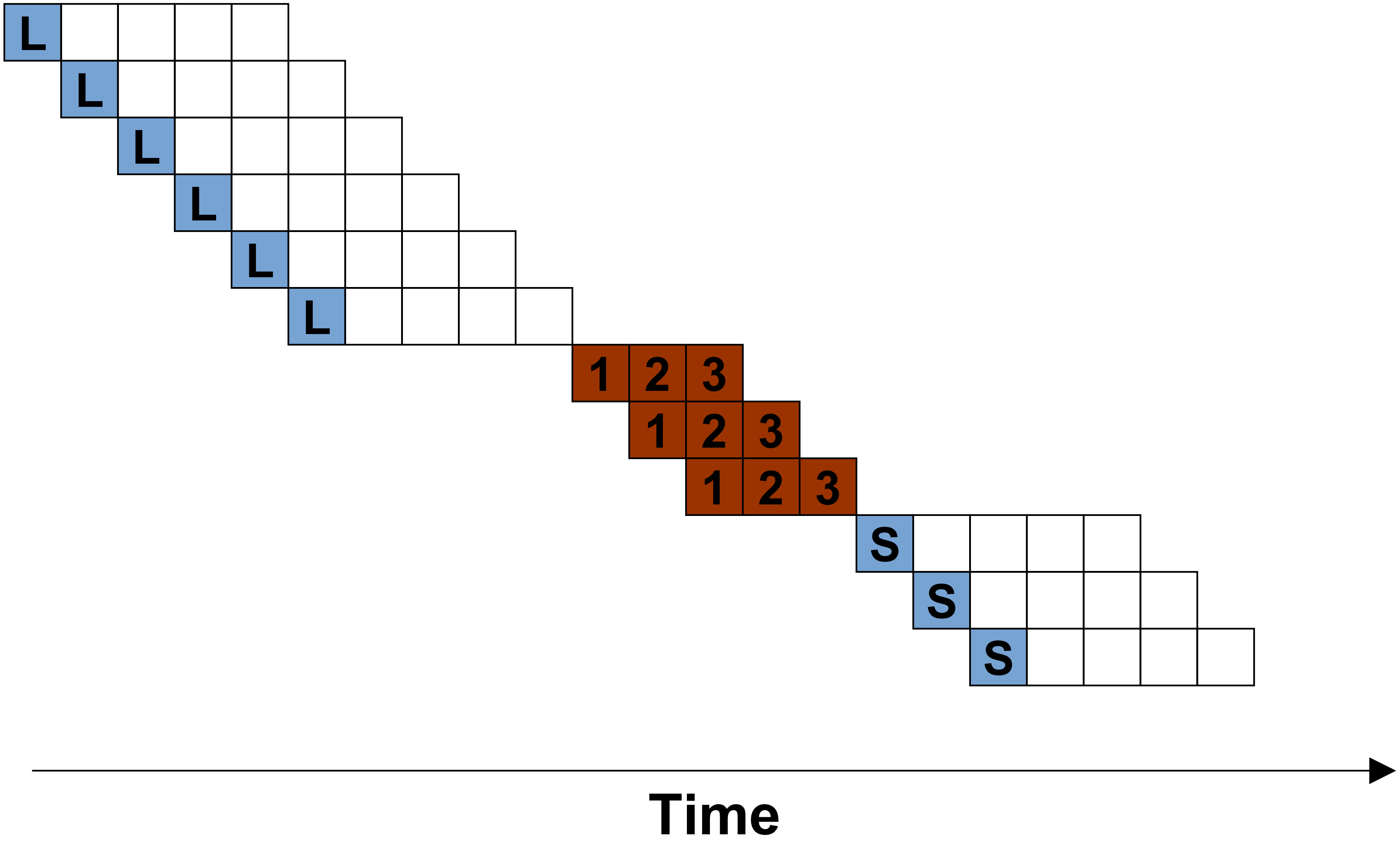
Access Parallelism

- Memory accesses which are independent and thus can be performed in parallel exhibit **access parallelism**
- The addresses of such accesses are usually known well in advance
- We can exploit access parallelism to saturate large bandwidth-delay memory systems

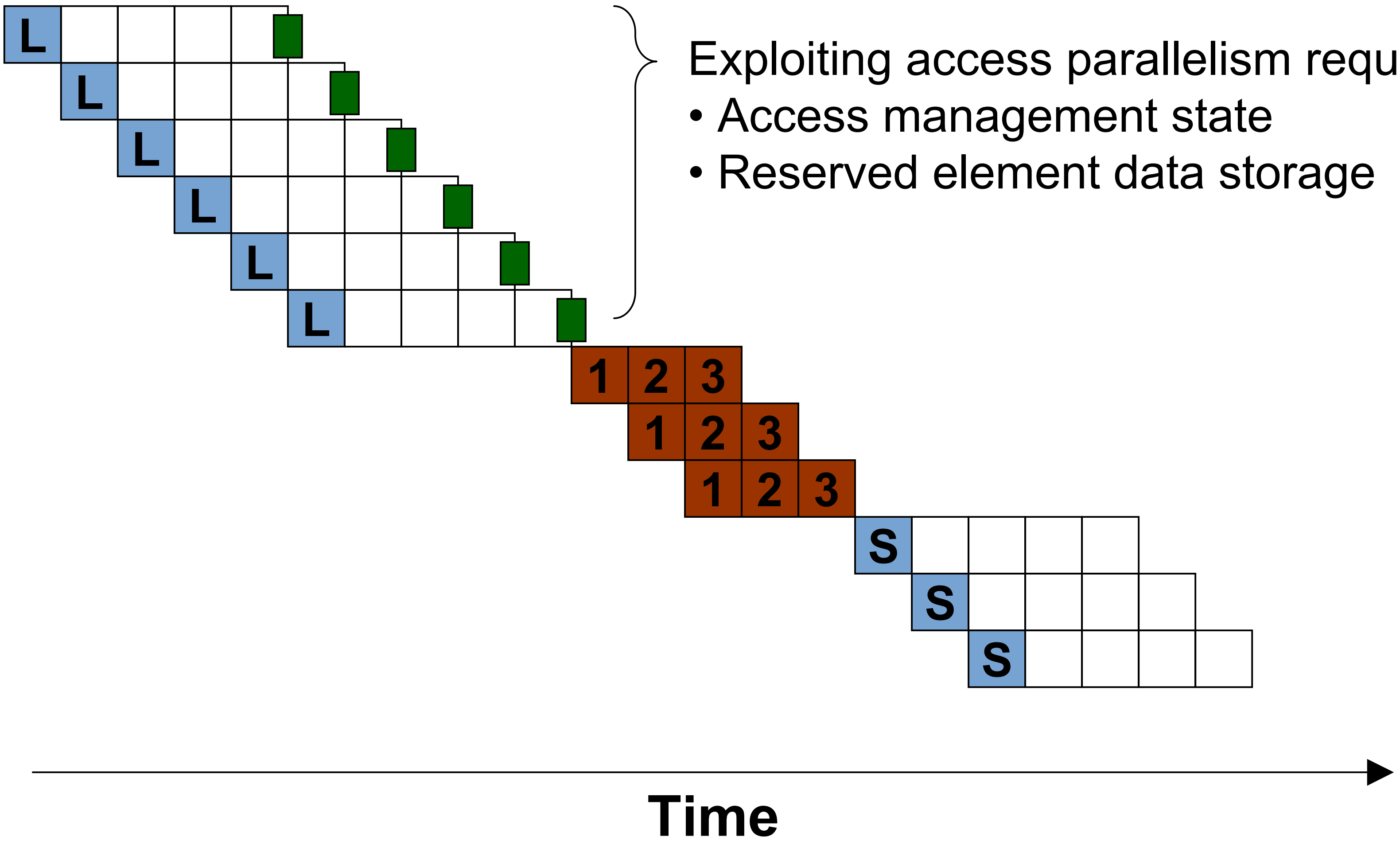
```
loop
  load
  load
  compute
  store
end
```



Access Parallelism



Access Parallelism



Exploiting access parallelism requires

- Access management state
- Reserved element data storage

Structured Access Parallelism

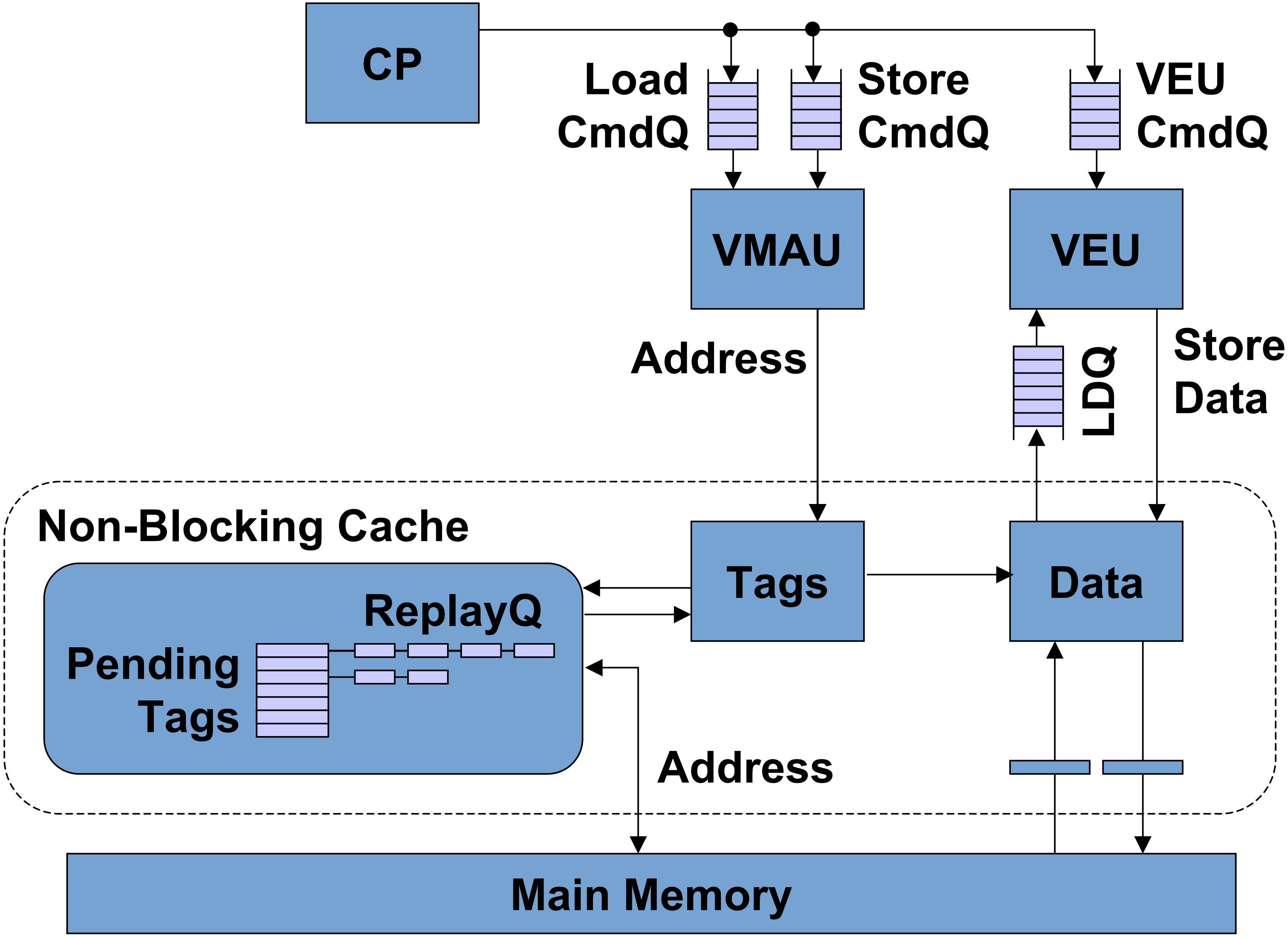
- The amount of required access management state and reserved element data storage scales roughly linearly with the number of outstanding elements
- Structured access parallelism is when the addresses of parallel accesses form a simple pattern such as each address having a constant offset from the previous address

Goal: Exploit structured access parallelism to saturate large bandwidth-delay product memory systems, while efficiently utilizing the available access management state and reserved element data storage

Access Parallelism in SCALE

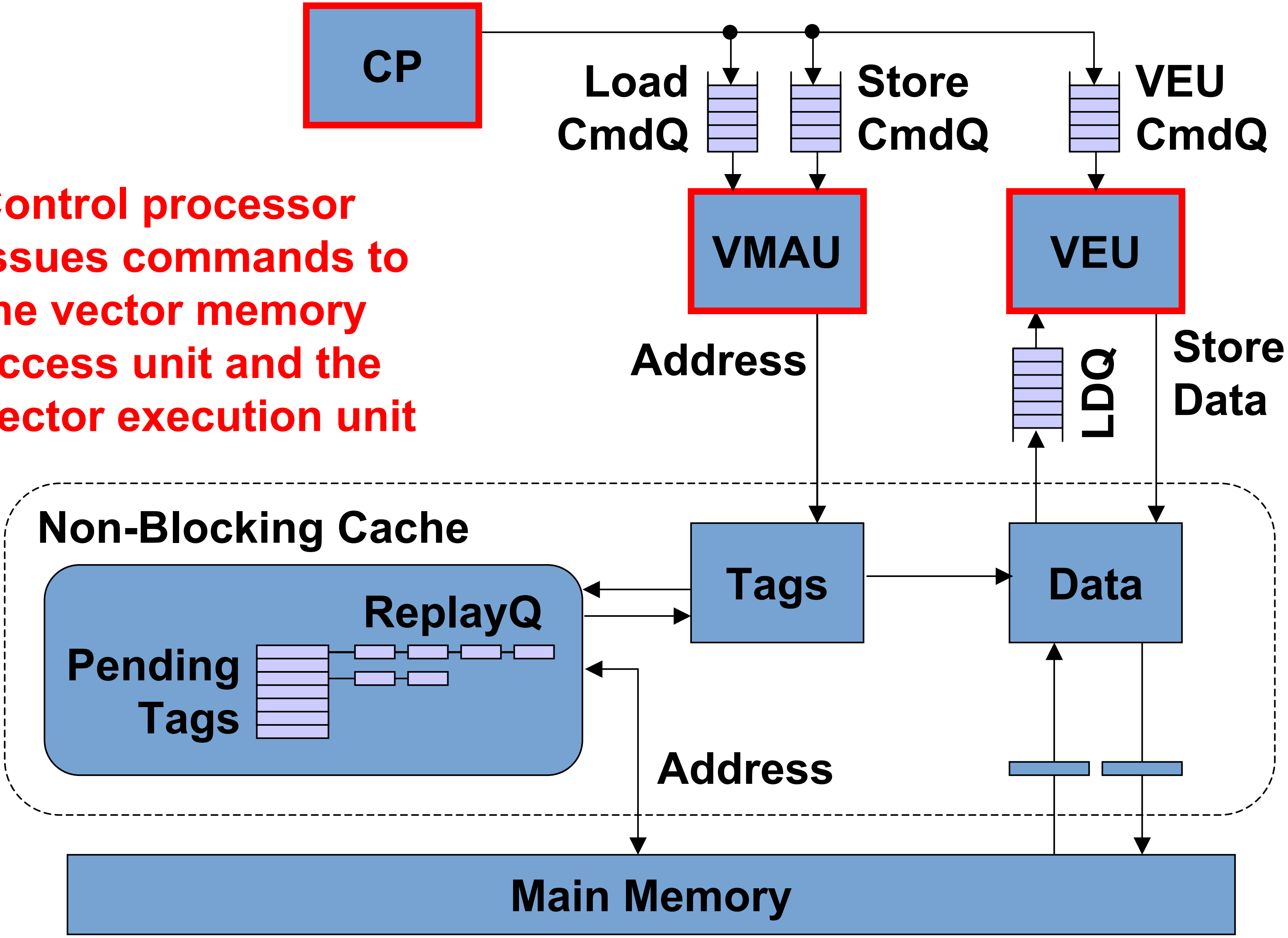
- SCALE is a highly decoupled vector-thread processor
  - Several parallel execution units effectively exploit data level compute parallelism
  - A vector memory access unit attempts to bring whole vectors of data into vector registers as in traditional vector machines
  - Includes a unified cache to capture the temporal and spatial locality readily available in some applications
  - Cache is non-blocking to enable many overlapping misses
- We introduce two mechanisms which enable the SCALE processor to more efficiently exploit access parallelism
  - **Vector memory refill unit** provides refill/access decoupling
  - **Vector segment accesses** represent a common structured access pattern in a more compact form

The SCALE Memory System



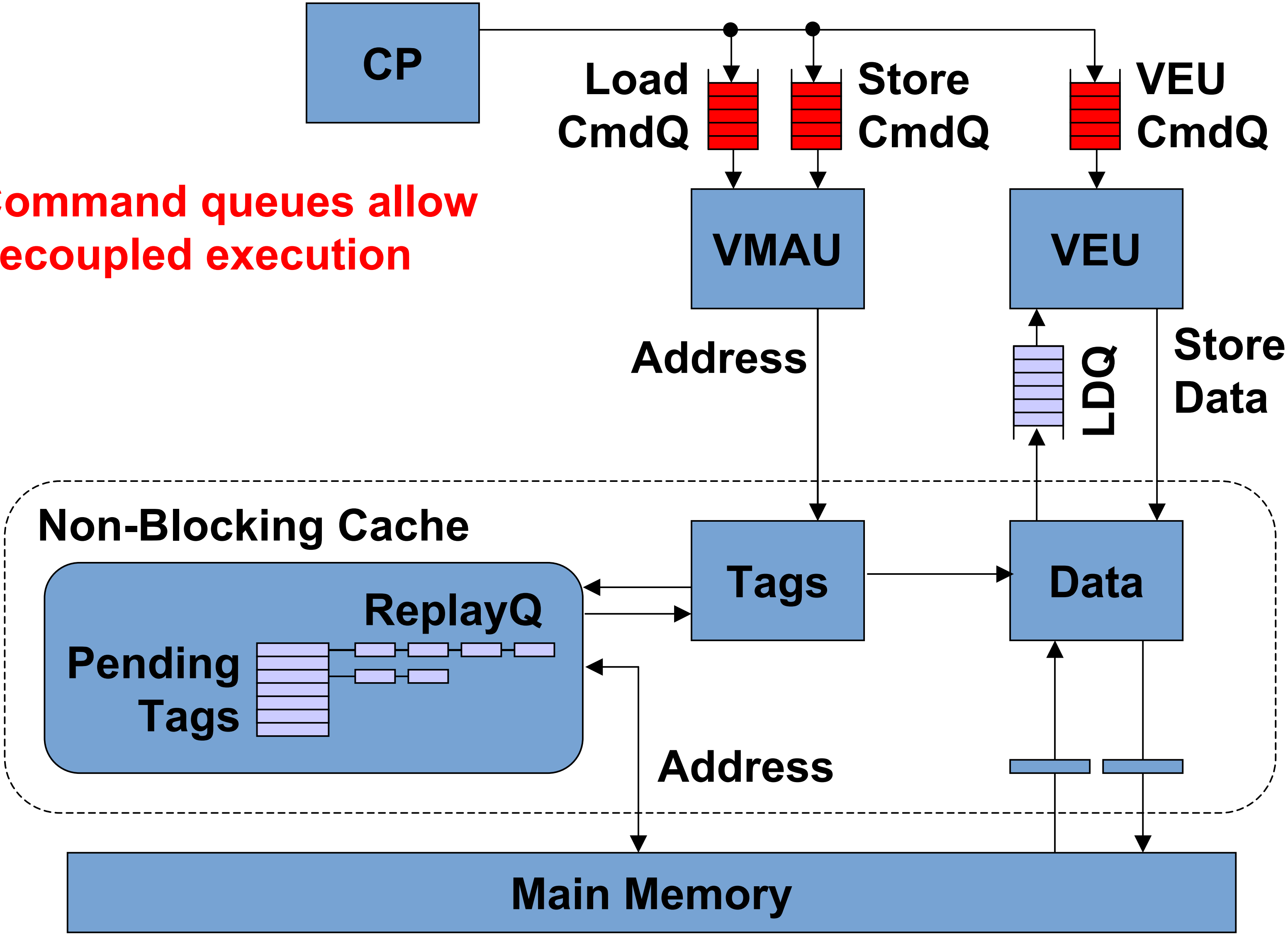
The SCALE Memory System

Control processor issues commands to the vector memory access unit and the vector execution unit

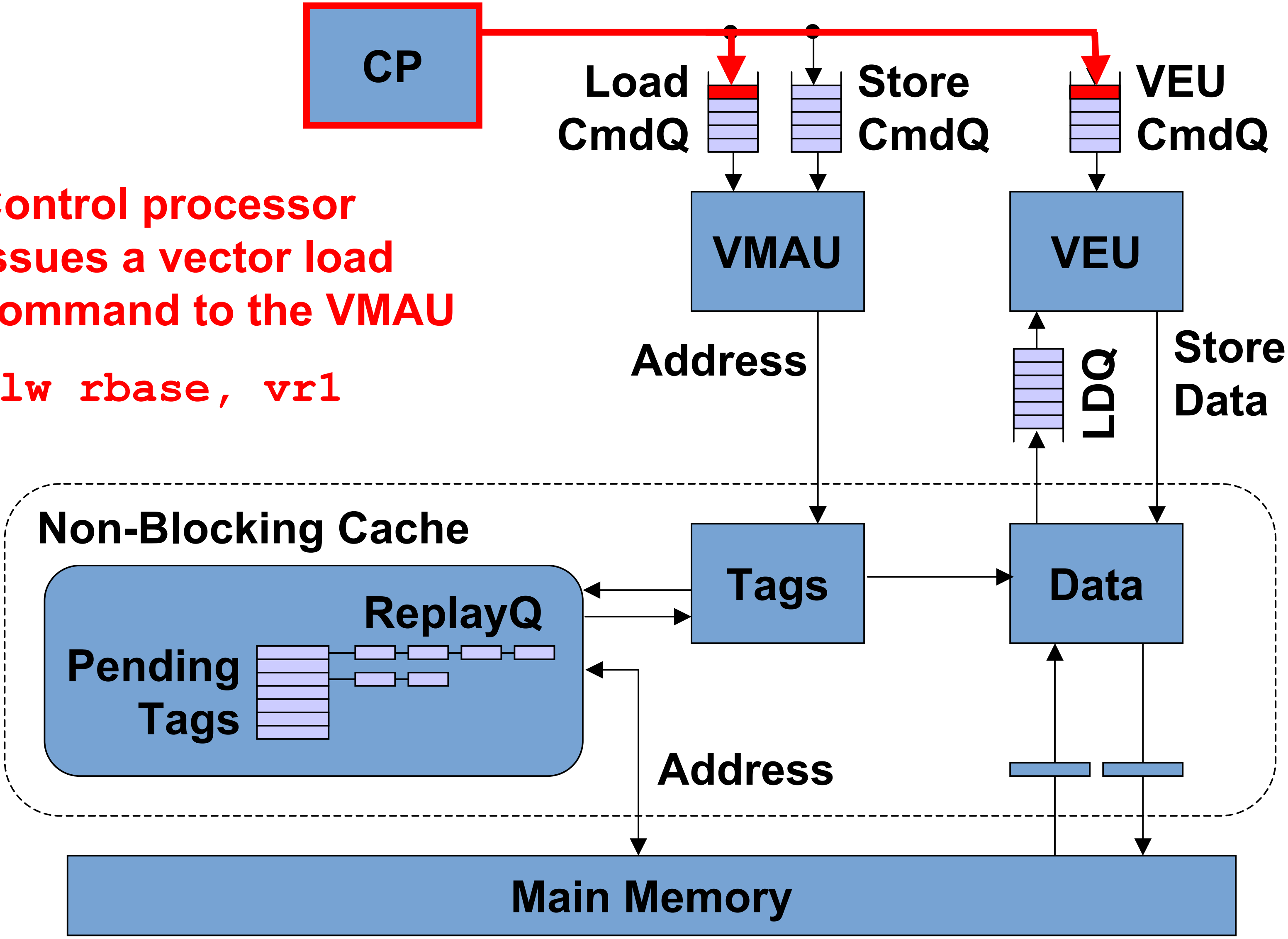


The SCALE Memory System

Command queues allow decoupled execution



Tracing a Vector Load

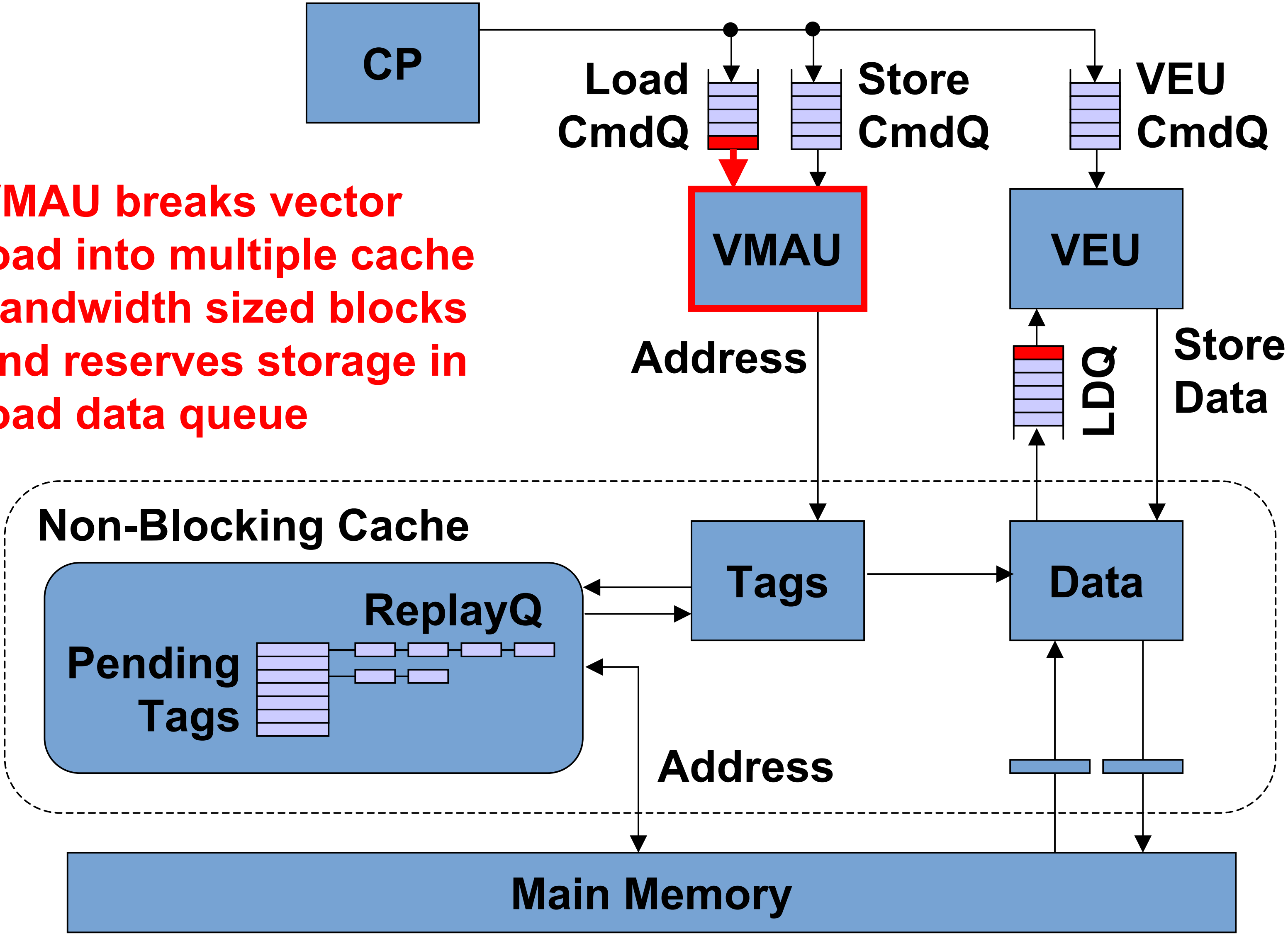


Control processor issues a vector load command to the VMAU

```
vlw rbase, vr1
```

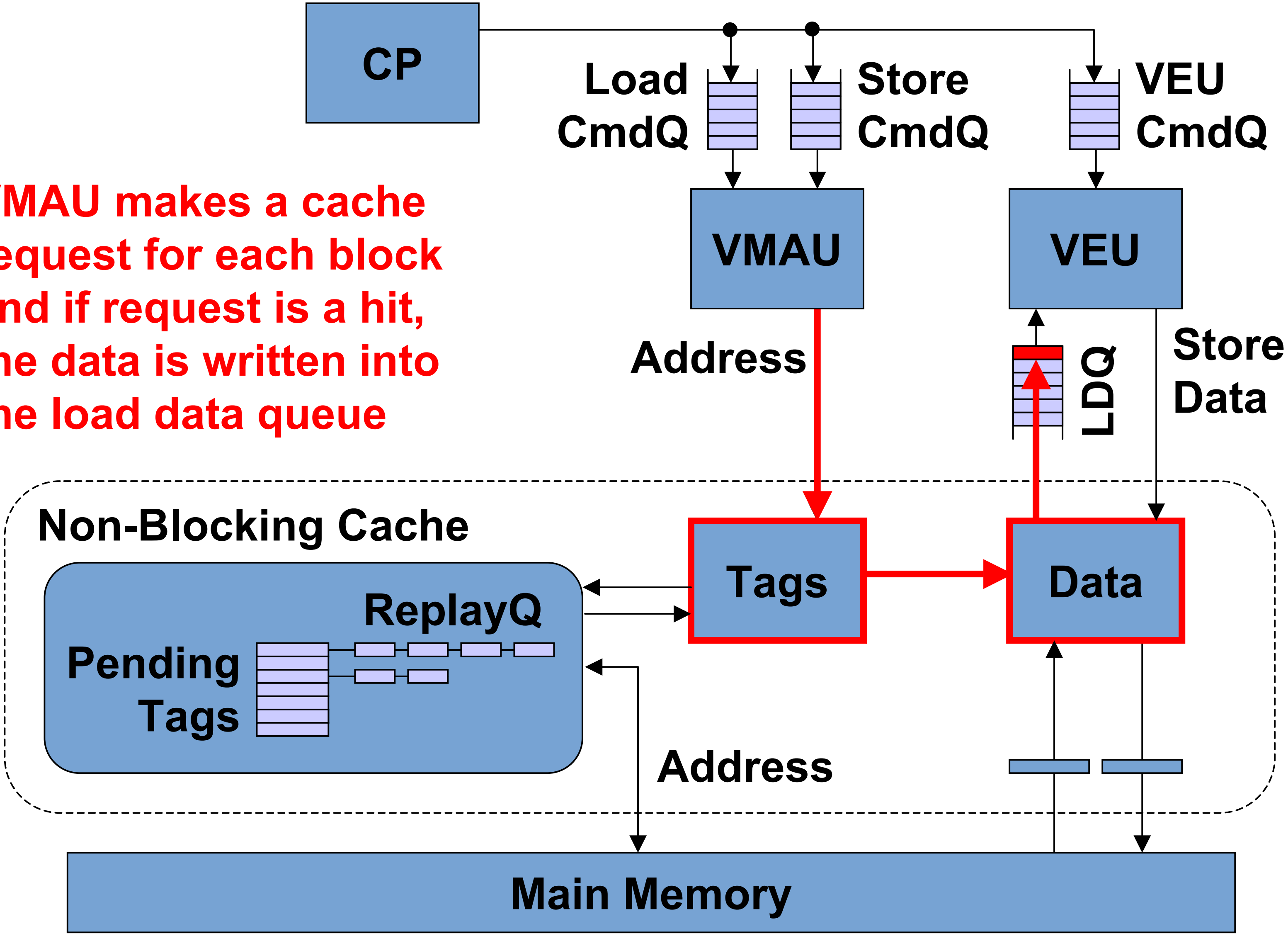
Tracing a Vector Load

VMAU breaks vector load into multiple cache bandwidth sized blocks and reserves storage in load data queue



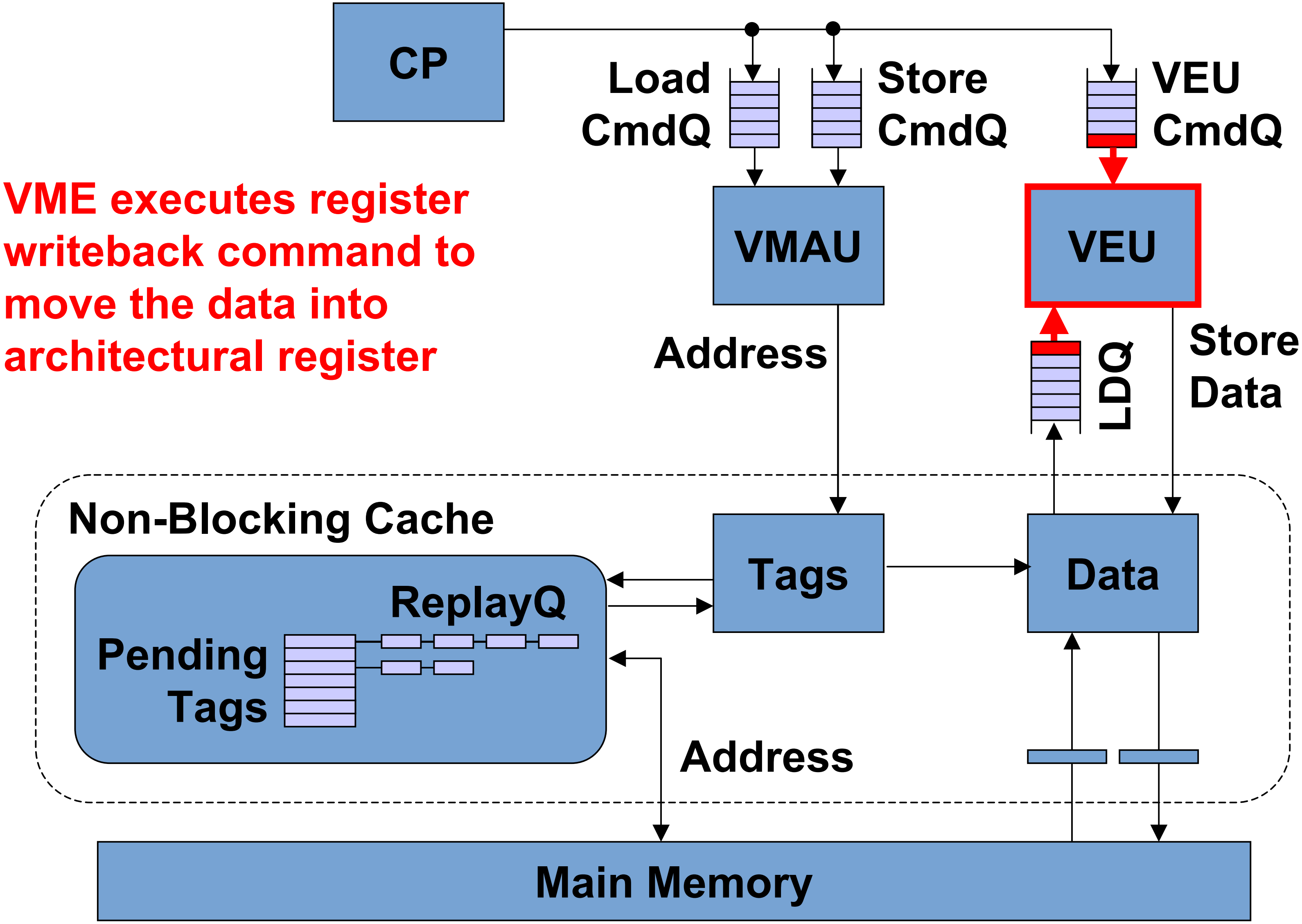
Tracing a Vector Load

VMAU makes a cache request for each block and if request is a hit, the data is written into the load data queue



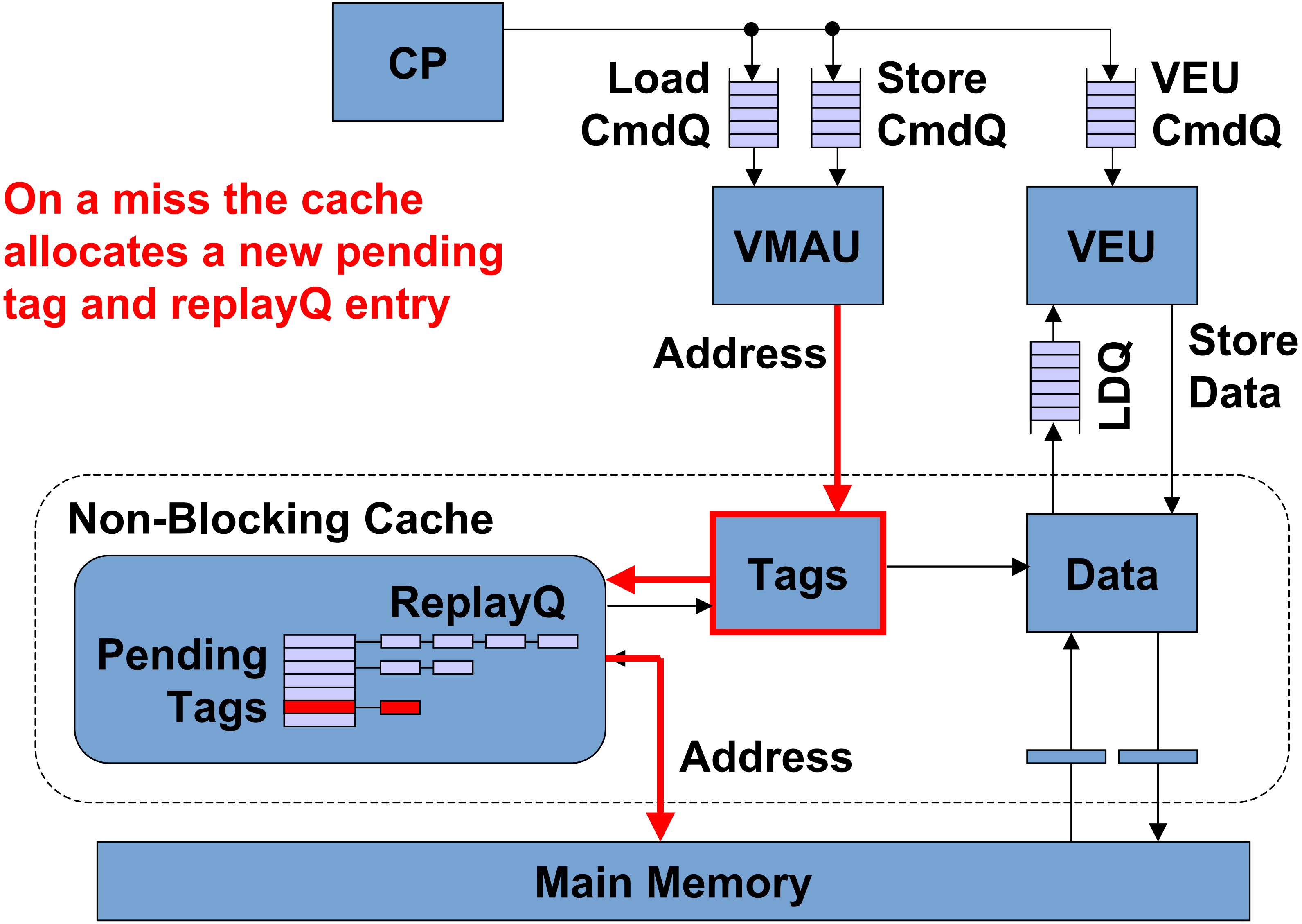
Tracing a Vector Load

VME executes register writeback command to move the data into architectural register



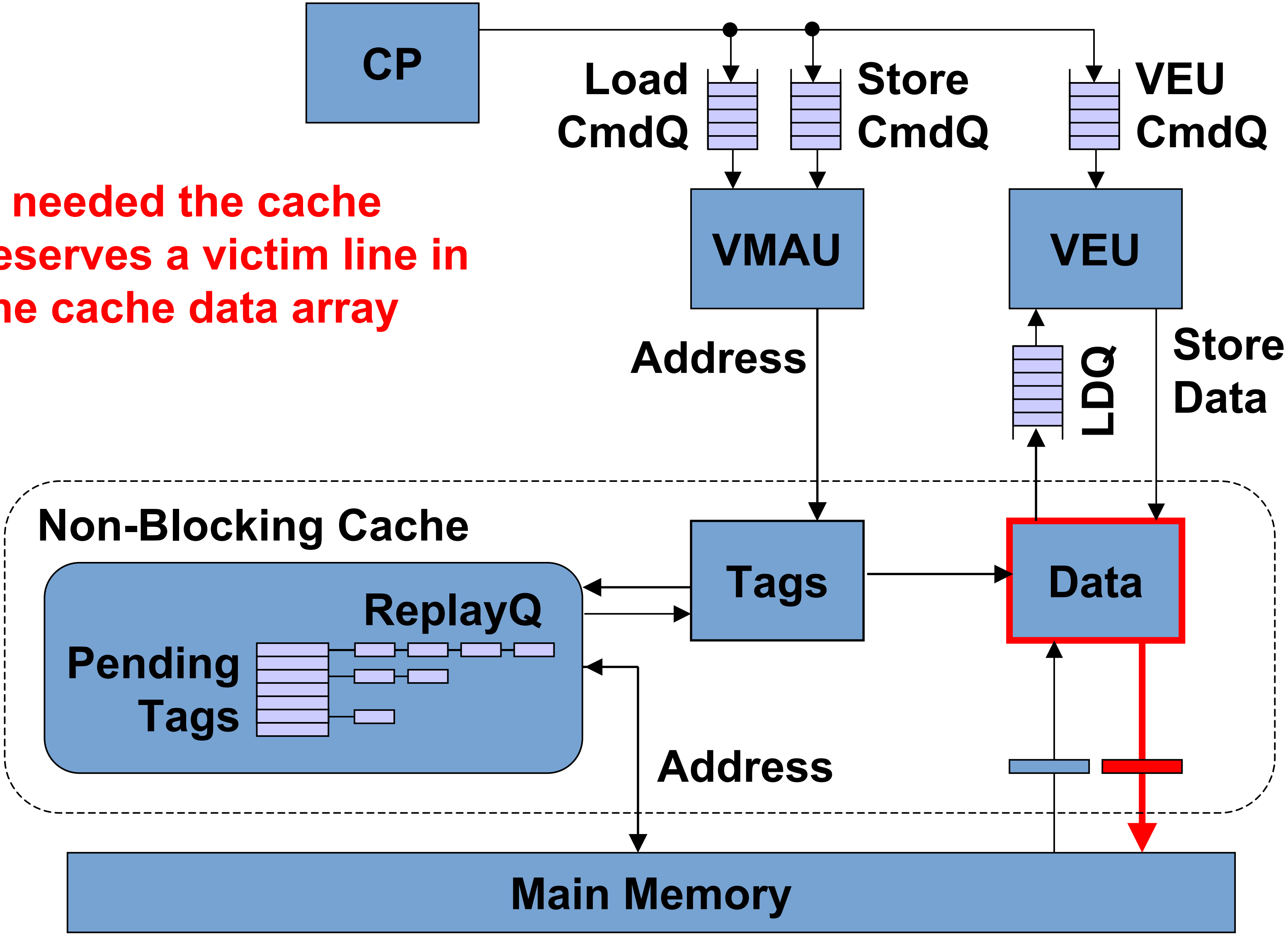
Tracing a Vector Load

On a miss the cache allocates a new pending tag and replayQ entry



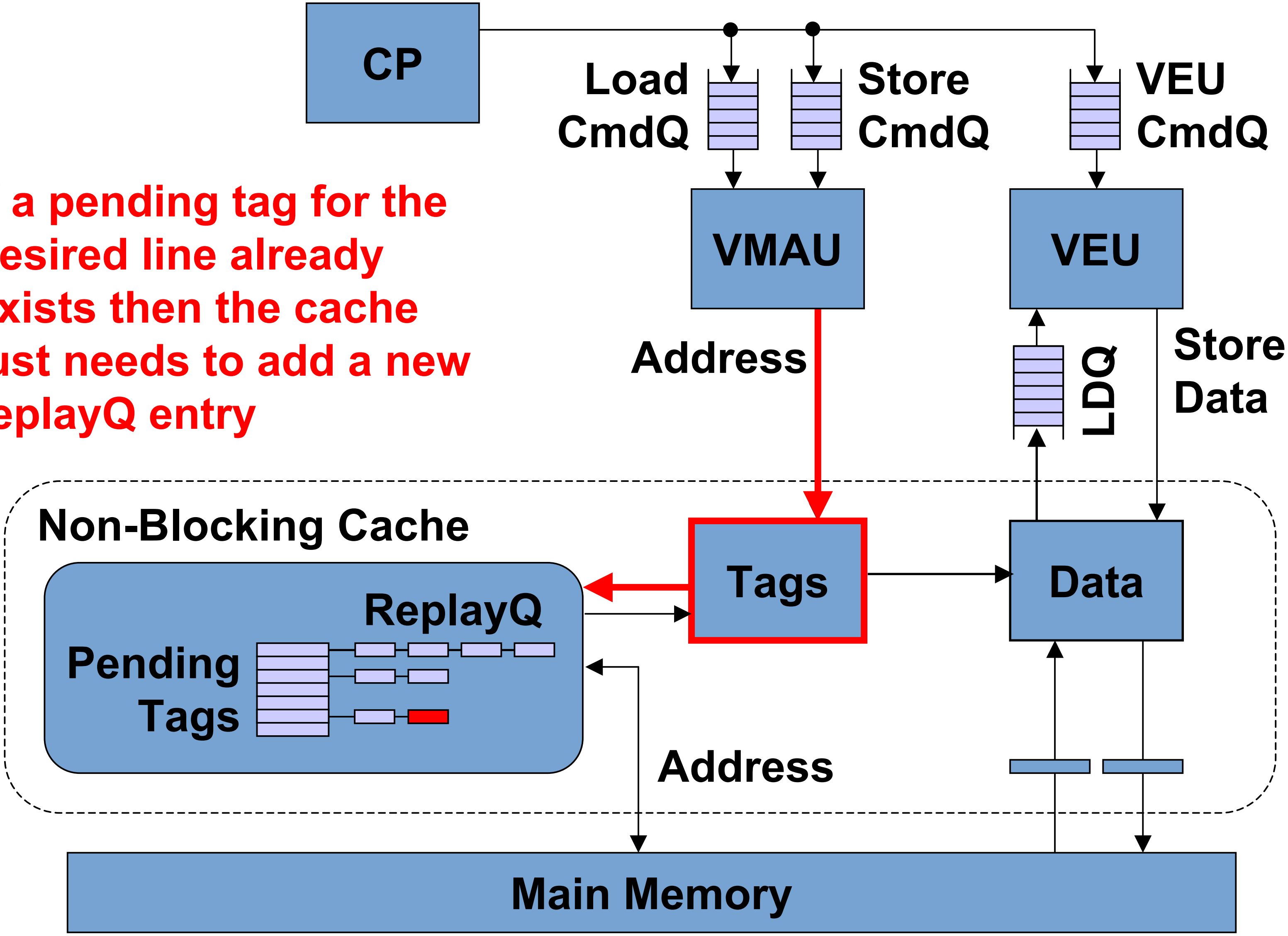
Tracing a Vector Load

If needed the cache reserves a victim line in the cache data array



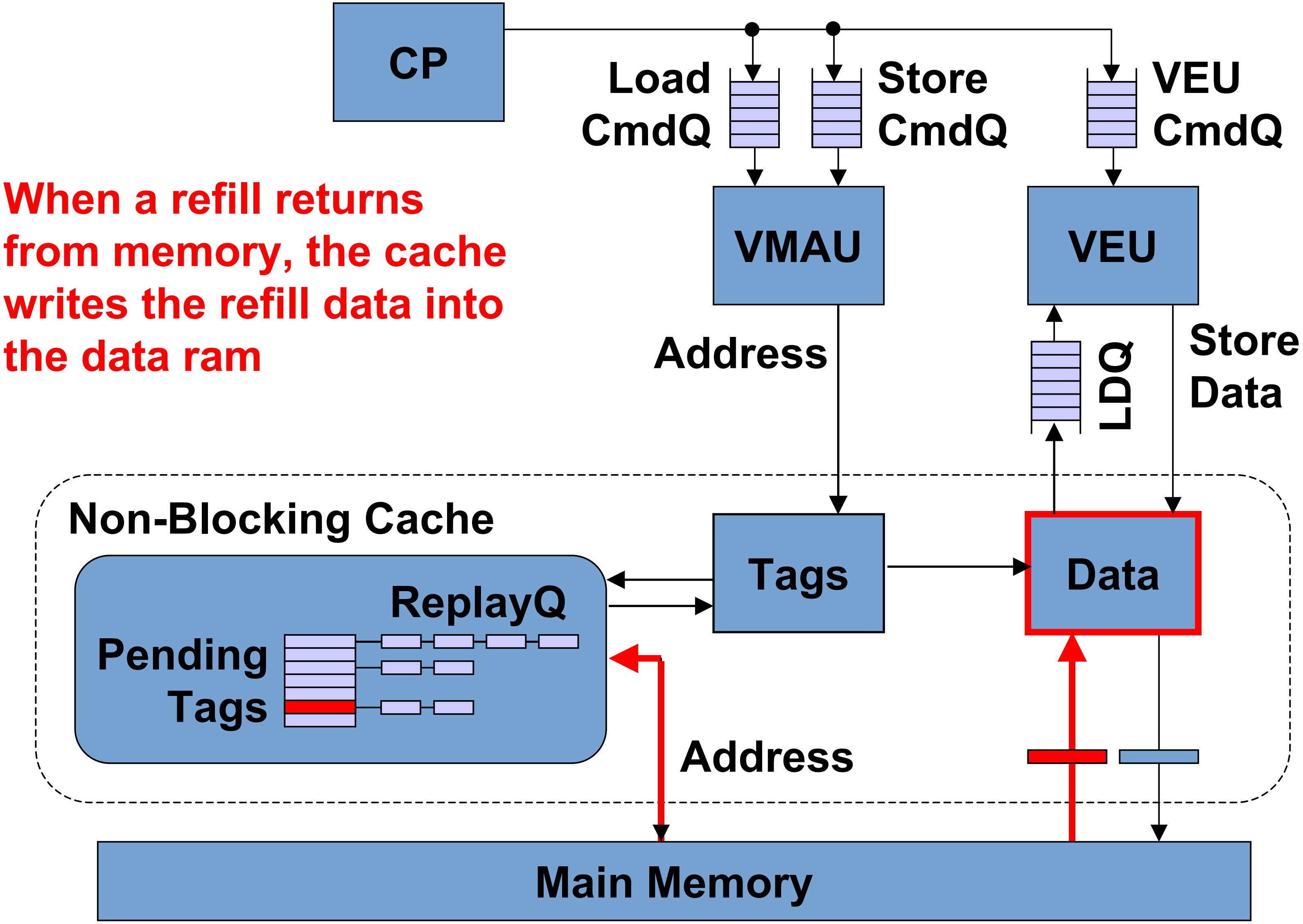
Tracing a Vector Load

If a pending tag for the desired line already exists then the cache just needs to add a new replayQ entry



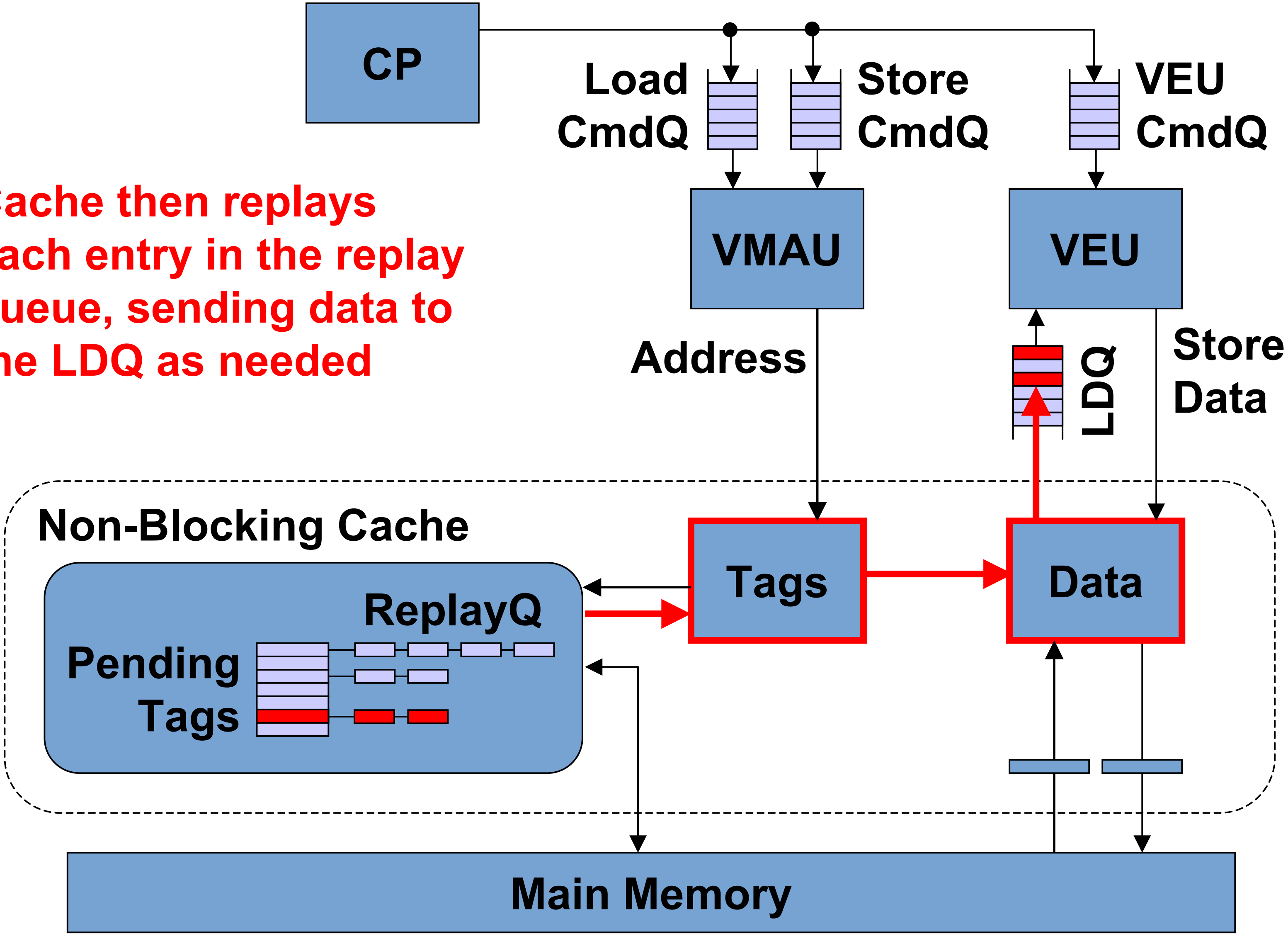
Tracing a Vector Load

When a refill returns from memory, the cache writes the refill data into the data ram



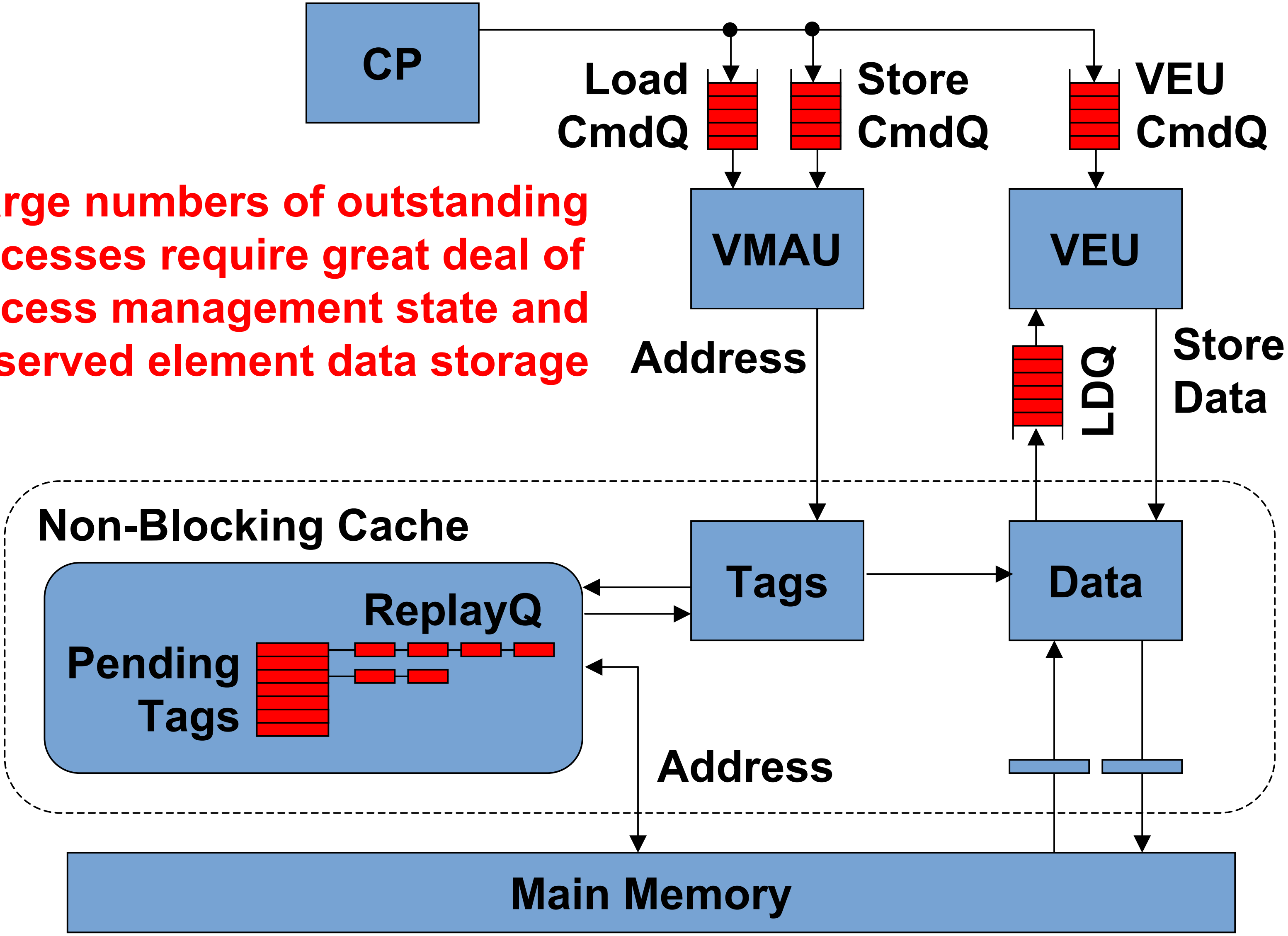
Tracing a Vector Load

Cache then replays each entry in the replay queue, sending data to the LDQ as needed

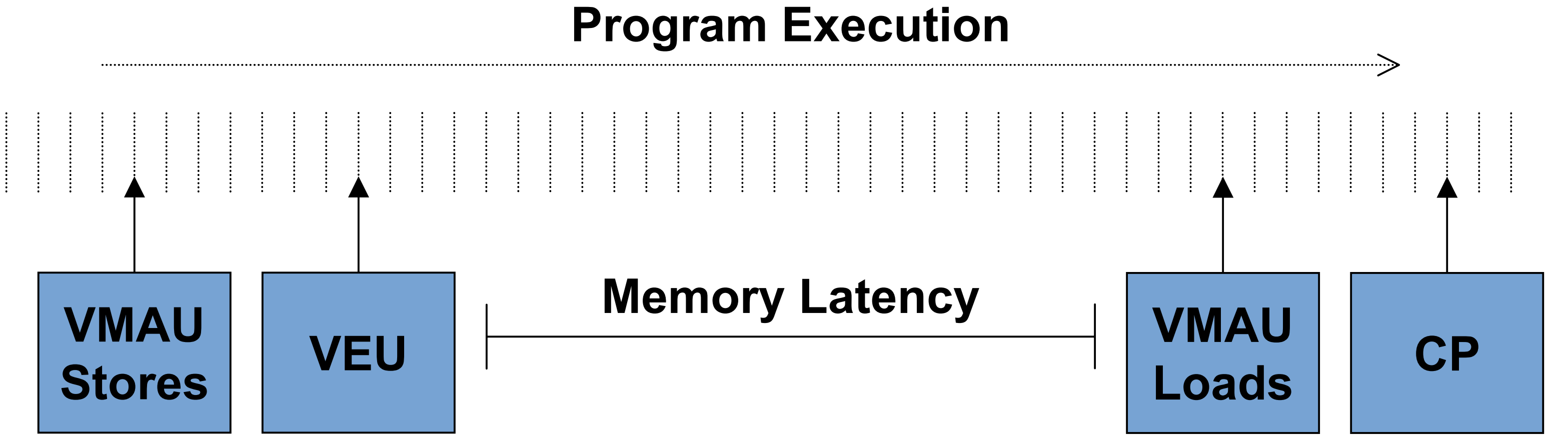


Tracing a Vector Load

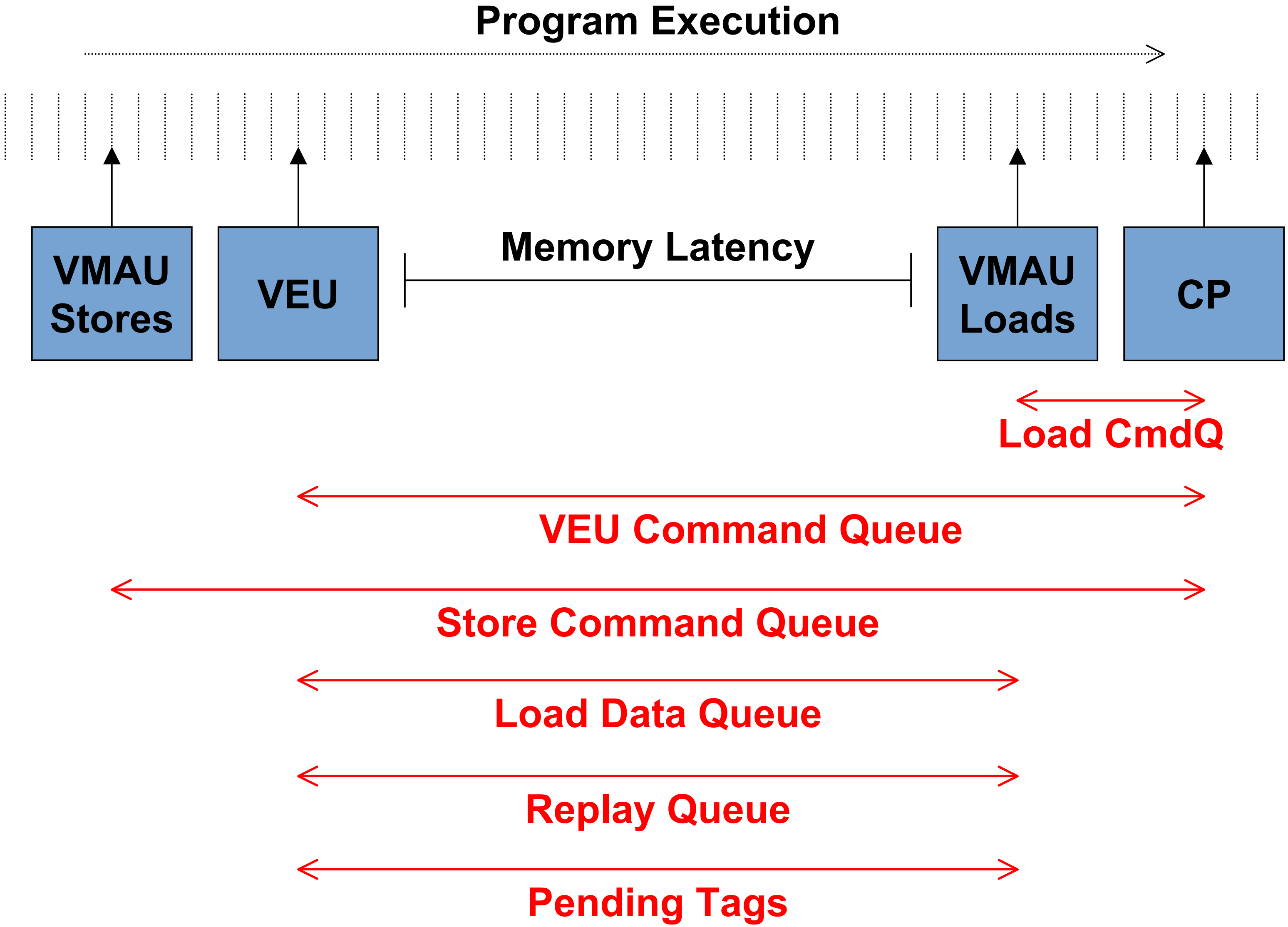
Large numbers of outstanding accesses require great deal of access management state and reserved element data storage



Required Queuing Resources

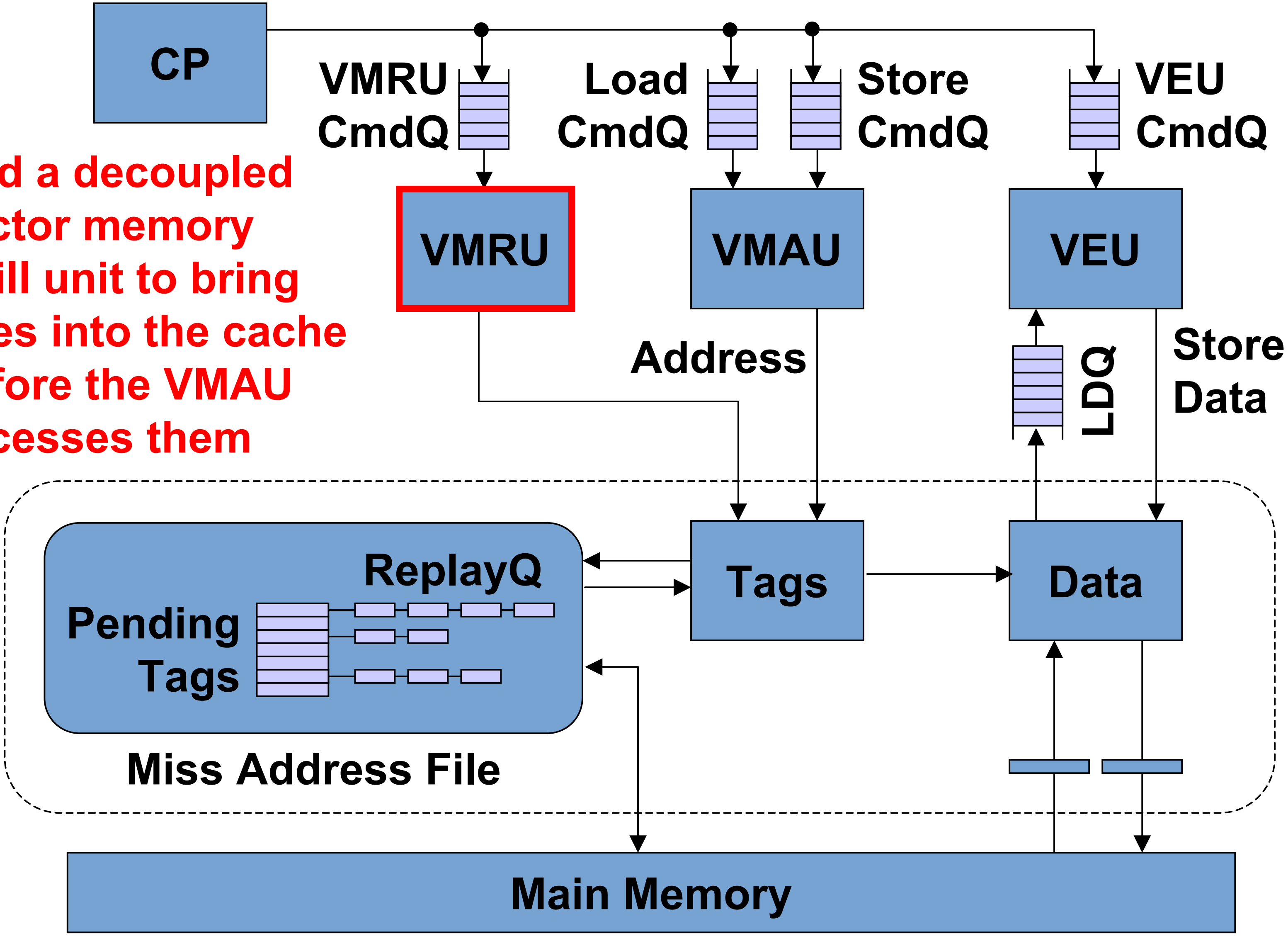


Required Queuing Resources



Vector Memory Refill Unit

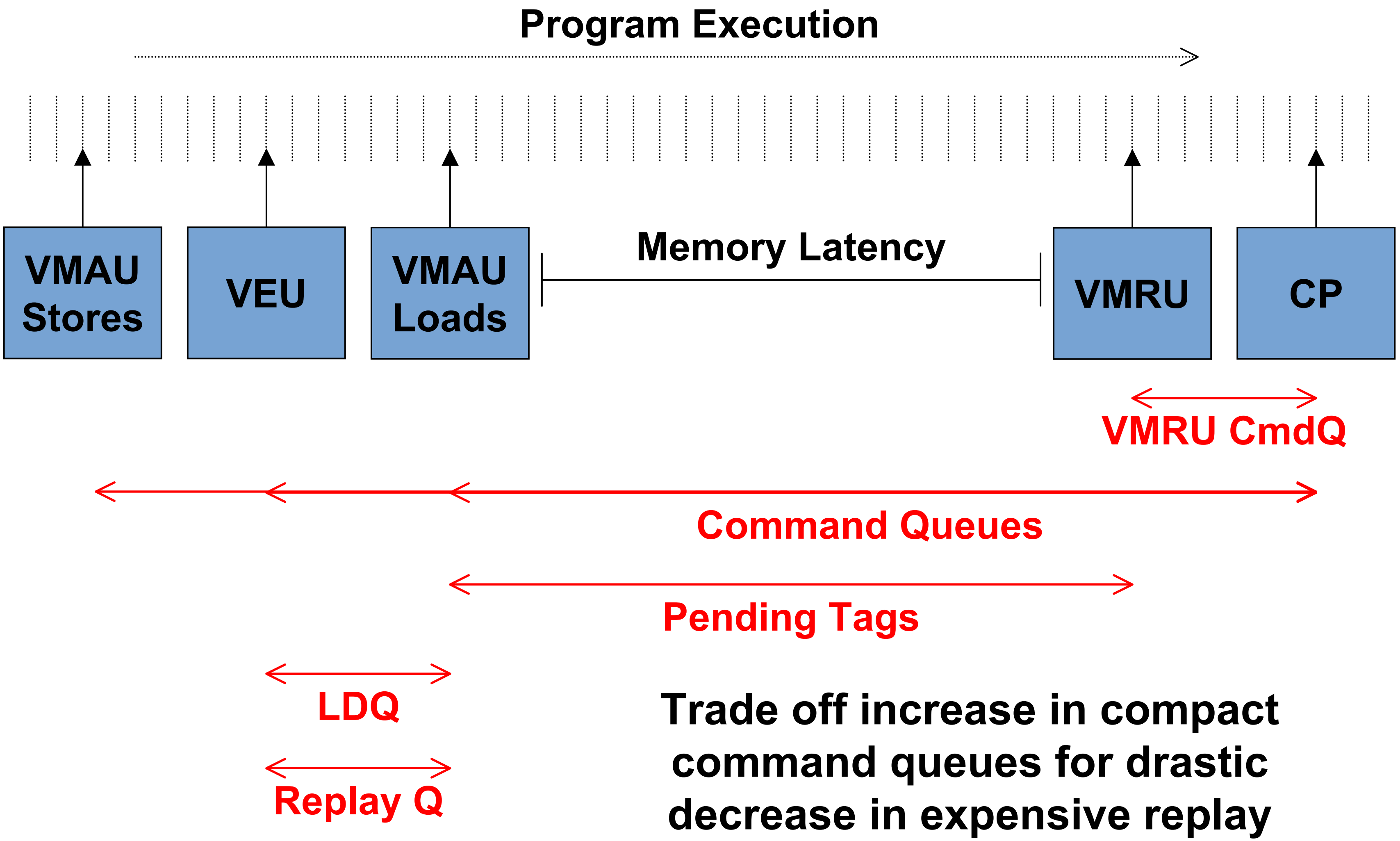
Add a decoupled vector memory refill unit to bring lines into the cache before the VMAU accesses them



Vector Memory Refill Unit

- VMRU runs ahead of the VMAU and pre-executes vector load commands
  - Issues refill requests for each cache line the vector load requires
  - Uses cache as efficient prefetch buffer for vector accesses, but because it is a cache, the buffer also captures reuse
  - Ideally the VMRU is far enough ahead that VMAU always hits
- Key implementation concerns
  - Throttling the VMRU to prevent evicting out lines which have yet to be used by the VMAU
  - Throttling the VMRU to prevent it from using up all the cache miss resources and blocking the VMAU
  - Throttling the VMAU to enable the VMRU to get ahead for memory bandwidth limited applications
  - Interaction between VMRU and cache replacement policy
  - Handling vector stores: allocating versus non-allocating

Required Queuing Resources



Program Execution

VMAU Stores

VEU

VMAU Loads

Memory Latency

VMRU

CP

VMRU CmdQ

Command Queues

Pending Tags

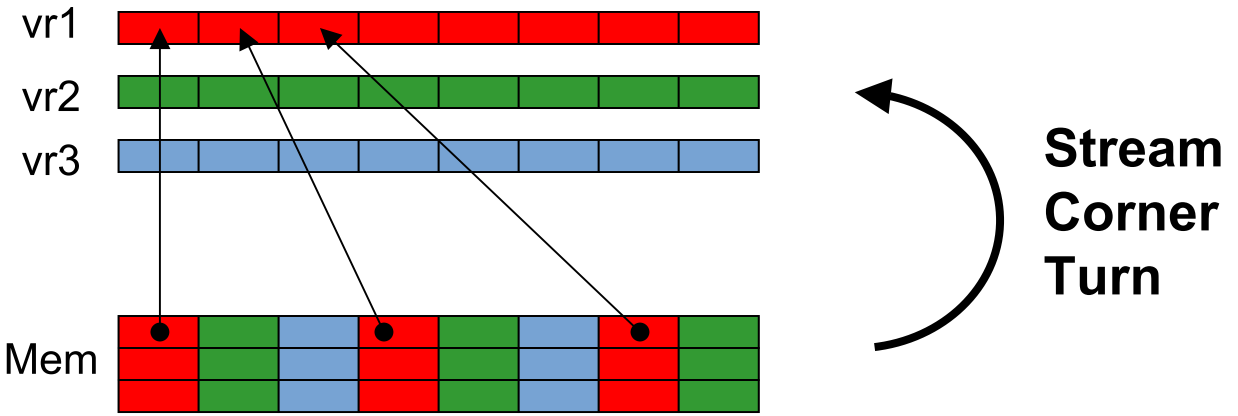
LDQ

Replay Q

Trade off increase in compact command queues for drastic decrease in expensive replay and load data queues

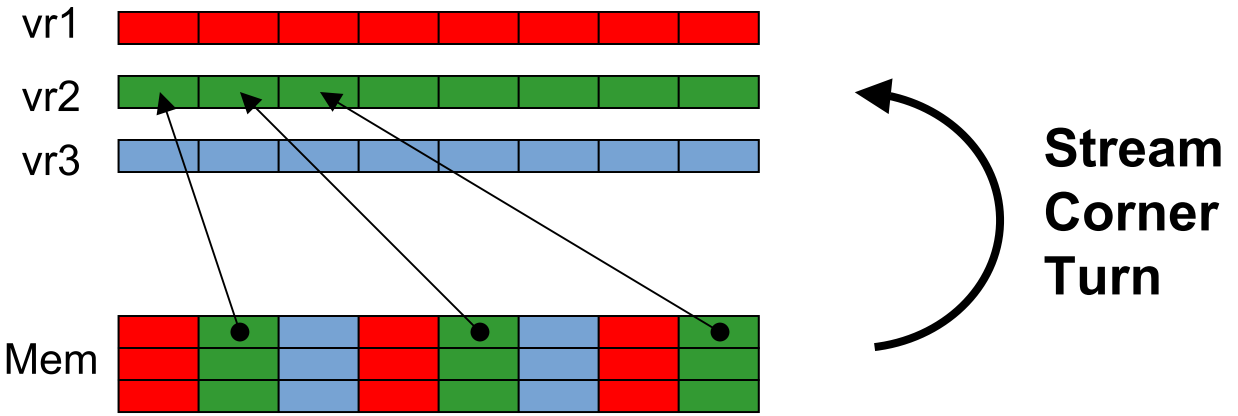
Vector Segment Accesses

- Vector processors usually use multiple strided accesses to load stream-of-records or groups of columns into vector registers



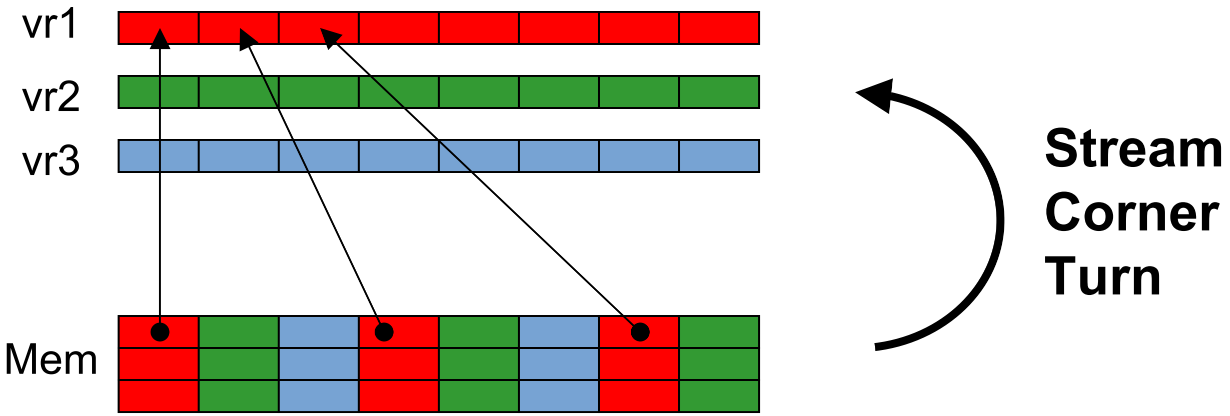
Vector Segment Accesses

- Vector processors usually use multiple strided accesses to load stream-of-records or groups of columns into vector registers



Vector Segment Accesses

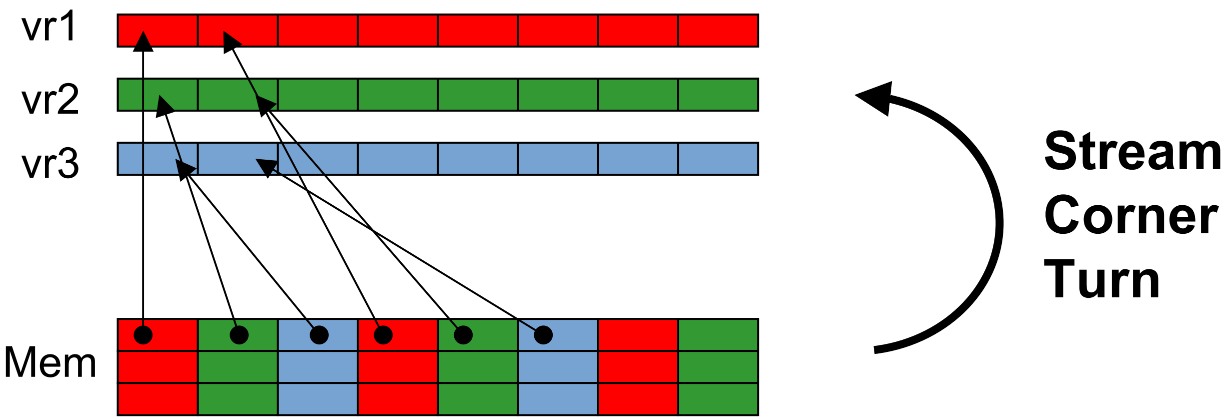
- Vector processors usually use multiple strided accesses to load stream-of-records or groups of columns into vector registers



- Several disadvantages
  - Increases bank conflicts in banked caches or memory systems
  - Ignores spatial locality in the application
  - Makes inefficient use of access management state

Vector Segment Accesses

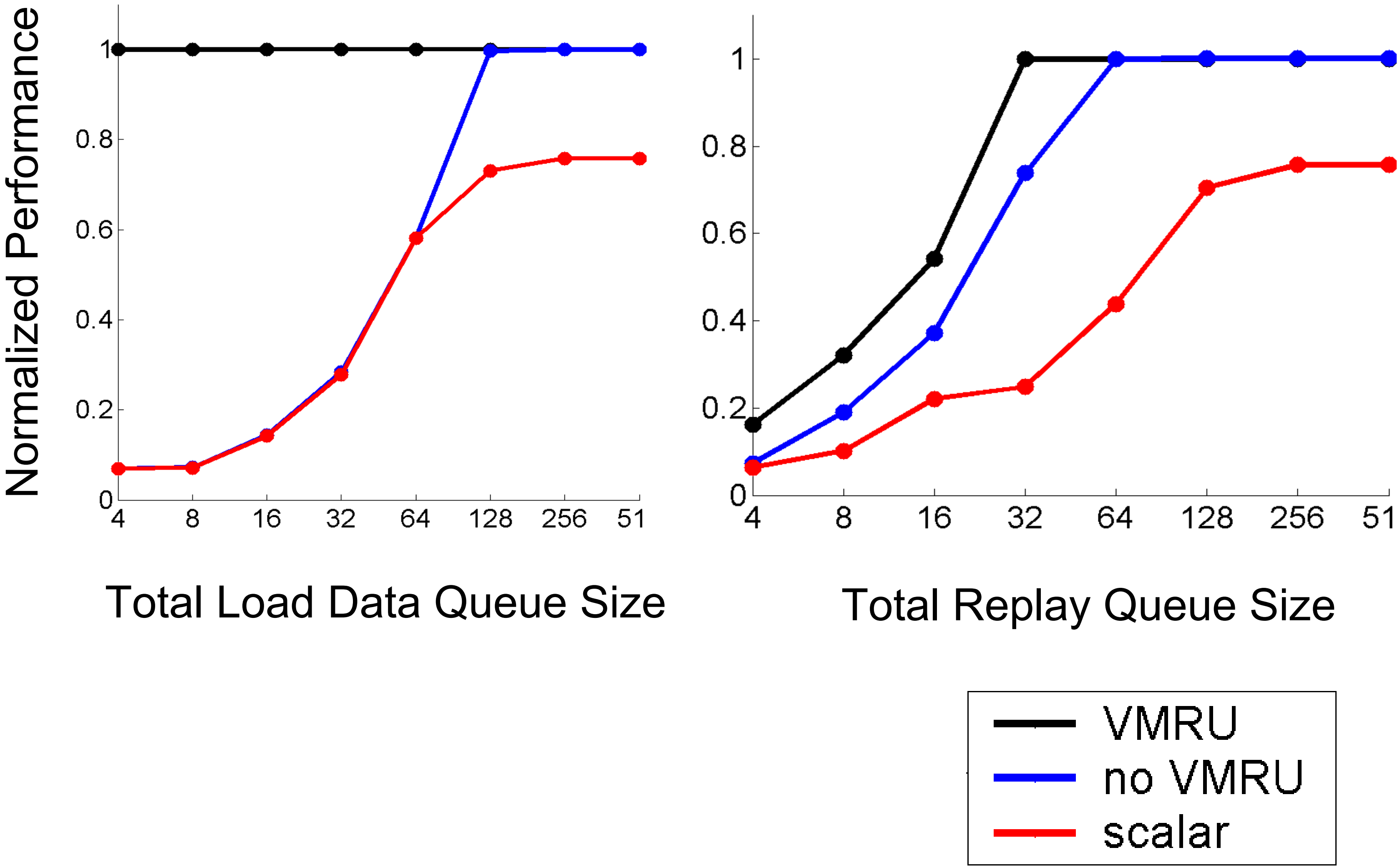
- Segment accesses uses efficient buffering under the existing data crossbar to perform the stream corner turn in hardware with a single vector memory command
- Reads data from the cache in a unit stride fashion and then writes data into the vector register file one element at a time



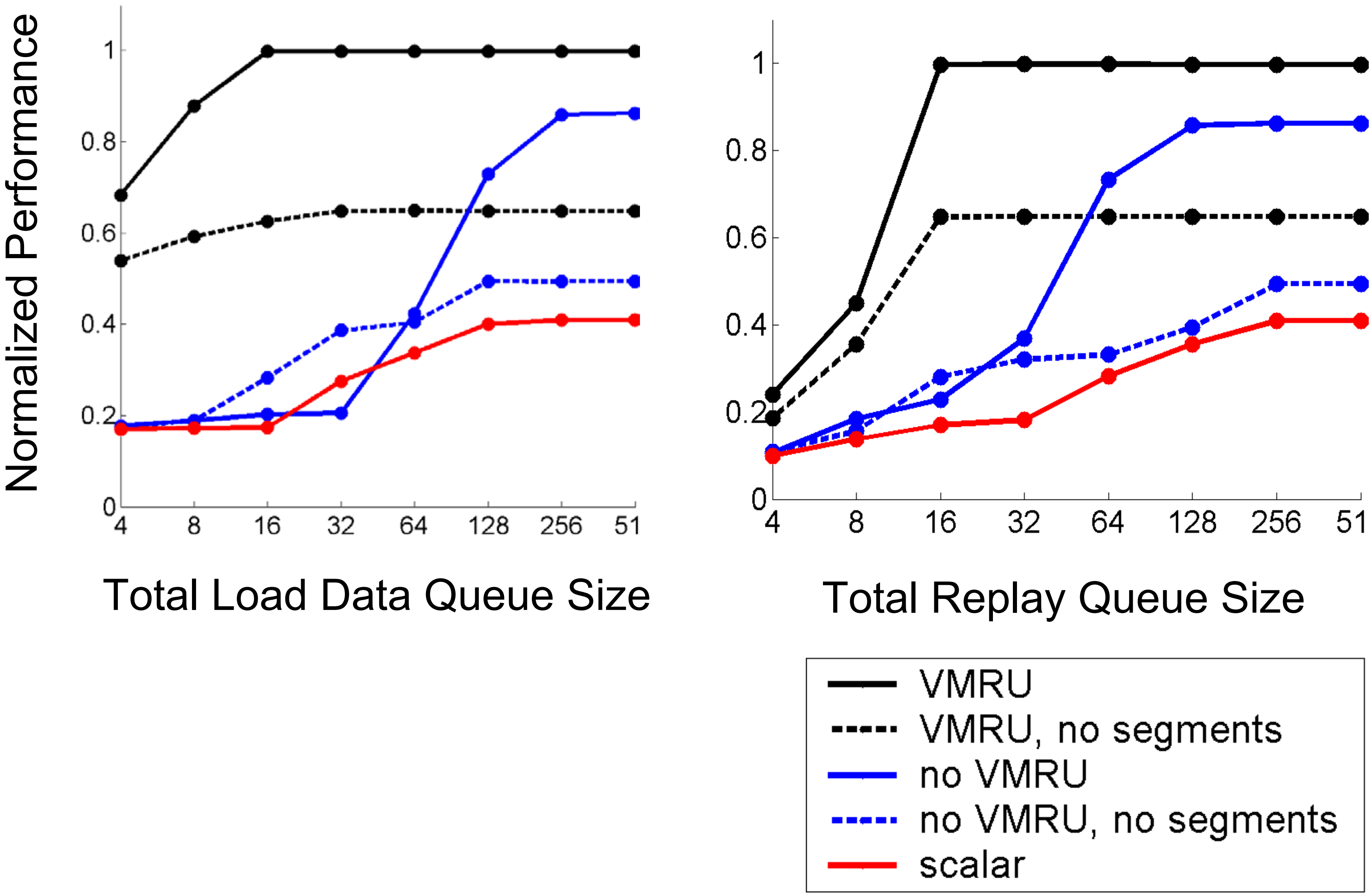
Evaluation

- Microarchitectural C++ model of control processor, VMRU, VMAU, VAE, and non-blocking cache
- "Magic" main memory with a latency of 100 cycles and bandwidth of 8 bytes/cycle to model the planned SCALE prototype system
  - Bandwidth delay product is 800 bytes = 25 cache lines
- A selection of kernels and microkernels which exhibit a wide variety of characteristics
  - Cooley-Tukey Fast Fourier Transform
  - Inverse Discrete Cosine Transform
  - Vertex 3D to 2D projection
  - Matrix transpose
  - Color conversion

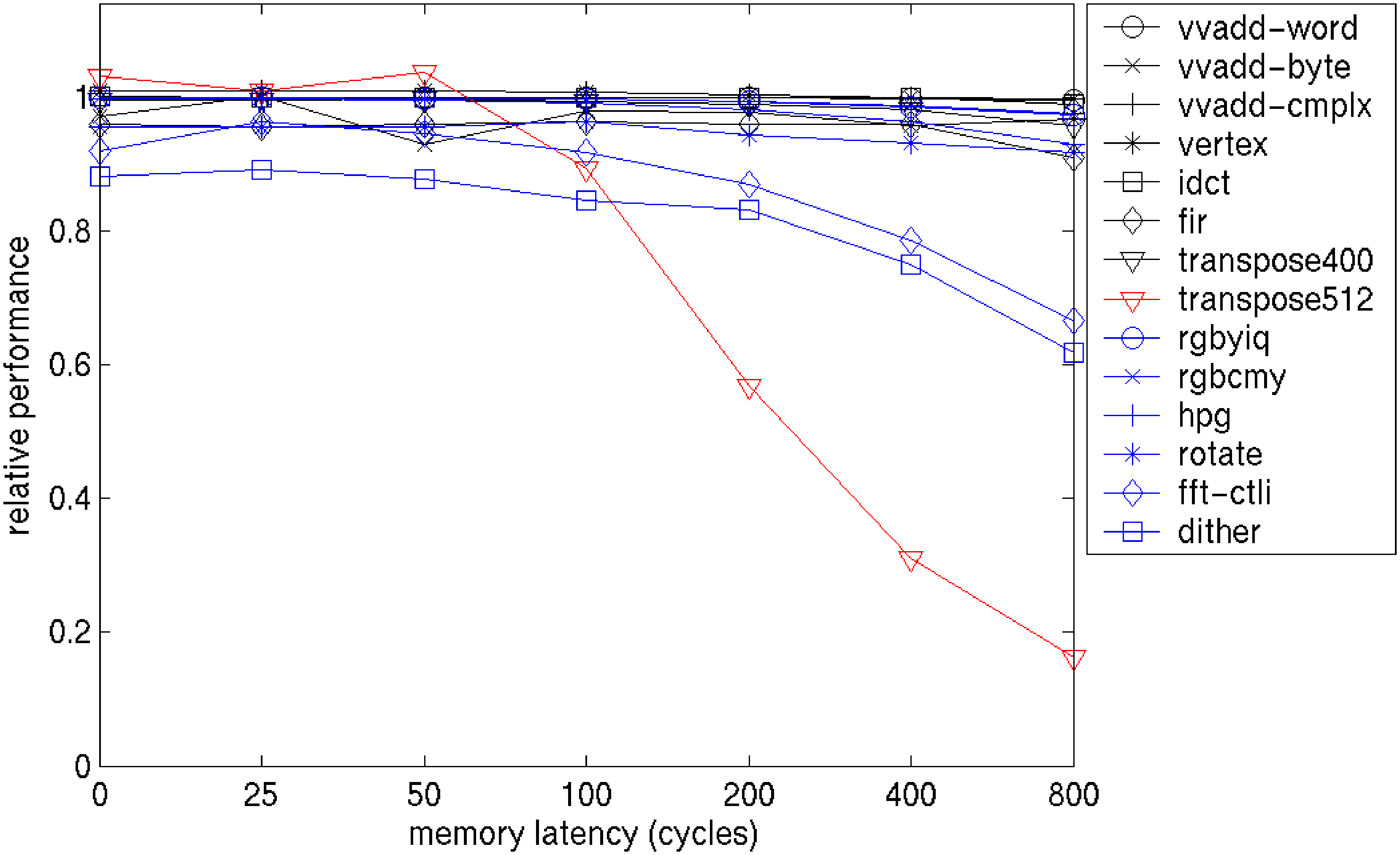
VVAdd Word Microkernel



RGBYIQ Kernel



Performance vs. Mem Latency



Conclusions

- Saturating **large bandwidth-delay memory systems** requires many in-flight elements and thus a great deal of access management state and reserved element data storage
- The SCALE processor uses **refill/access decoupling** and **vector segment accesses** to efficiently saturate its memory system with hundreds of outstanding accesses

Paper to appear in *37th International Symposium on Microarchitecture*, December 2004