Hwacha V4: Decoupled Data Parallel Custom Extension

Colin Schmidt, Albert Ou, Krste Asanović

UC Berkeley

Berkeley Architecture Research

Introduction

- Data-parallel, custom ISA extension to RISC-V with a configurable ASIC-focused implementation
- Developed to research energy-efficient implementation of vector architecture
- Several versions over the years with this the most recent being V4
- Rocket-chip based accelerator with binutils, llvm-backend, FireSim, and ASIC VLSI support
- Open-sourced now!

Key Ideas in Vector Architectures

Berkeley Architecture Research

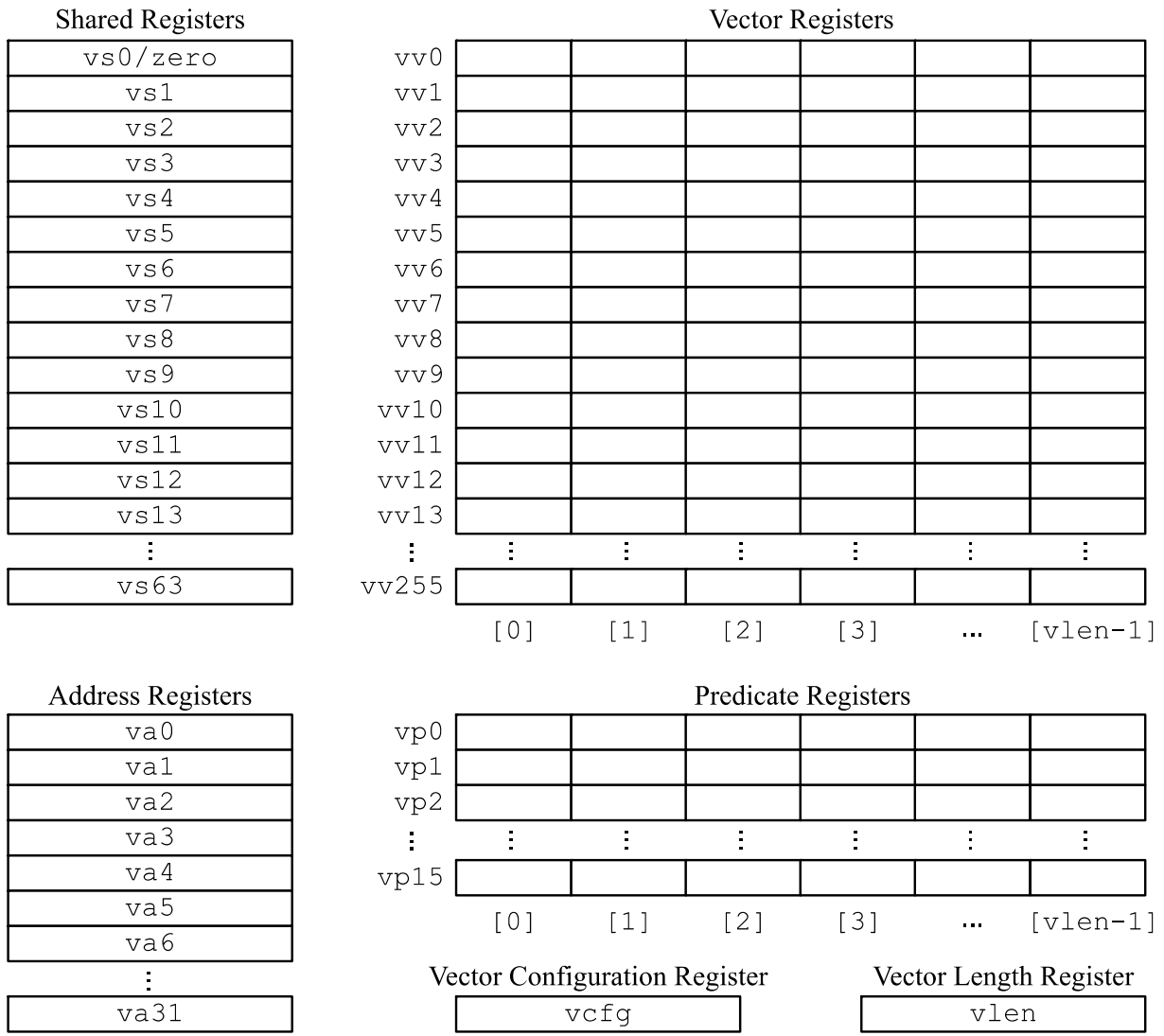
- Runtime-variable vector length register
  - Compact, portable code
  - Hardware implementation scalability
  - Reasonable encoding space usage
- Long vectors with temporal and spatial execution
- Scalar-vector computation
- Configurable register file
  - Trade architectural registers for longer vector lengths

Hwacha Key Ideas

- Goal: Maximize efficiency of in-order vector microarchitecture
- Access/execute decoupling
- Precision-reconfigurable vector RF to increase vector lengths
- Predication and consensual branches to enable control flow
- Mixed scalar-vector computation
- Mixed-precision registers and datapaths
- OS support with unified virtual memory and restartable exceptions

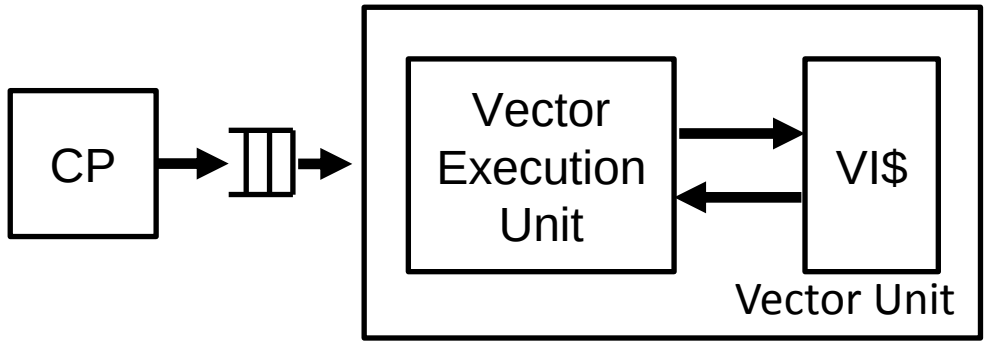
Hwacha Architecture Overview

- Configurable vector data and predicate register files
- Fixed number of address and scalar registers
- Unit-stride, constant-stride, indexed load/stores
- Full set of half, single, double FP, integer, and predicate operations



Vector Fetch Programming model

- Separate vector instructions from scalar control thread
- Vector-fetch instruction, `vf`, sends address from control thread to vector unit
- Vector unit fetches and decodes separate instruction stream
- Control processor runs ahead enqueuing more work, hiding latency



Hwacha Vector-Fetch Architectural Paradigm

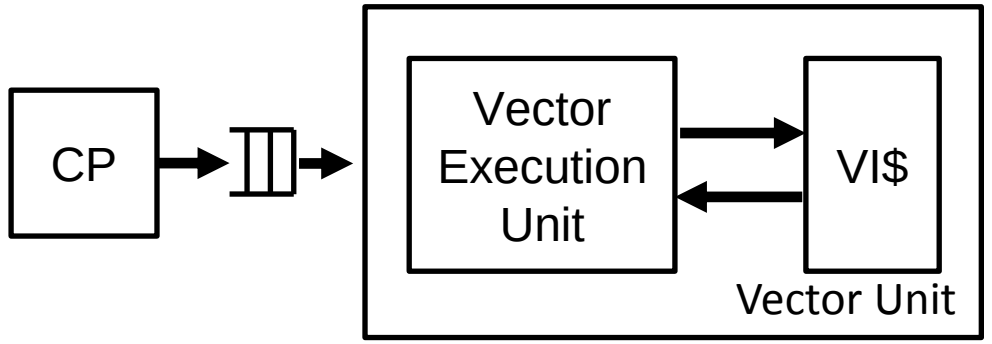
```
a0: n, a1: a,
a2: *x, a3: *y

  vmcs vs0, a1
stripmine:
  vsetvl t0, a0
  vmca va0, a2
  vmca va1, a3
  vf saxpy
  slli t1, t0, 2
  add a2, a2, t1
  add a3, a3, t1
  sub a0, a0, t0
  bnez a0, stripmine
```

Control Thread

```
saxpy:
  vlw vv0, va0
  vlw vv1, va1
  vfma.svv vv1, vs0, vv0, vv1
  vsw vv1, va1
  vstop
```

Worker Thread




```
for (i=0; i<n; i++) {
  y[i] = a*x[i] + y[i];
}
```

Hwacha Vector-Fetch Architectural Paradigm

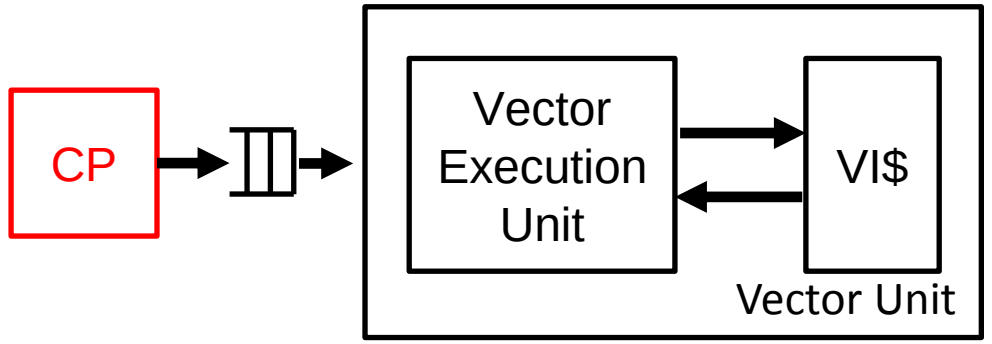
```
a0: n, a1: a,
a2: *x, a3: *y

  vmcs vs0, a1
stripmine:
  vsetvl t0, a0
  vmca va0, a2
  vmca va1, a3
  vf saxpy
  slli t1, t0, 2
  add a2, a2, t1
  add a3, a3, t1
  sub a0, a0, t0
  bnez a0, stripmine
```

Control Thread

```
saxpy:
  vlw vv0, va0
  vlw vv1, va1
  vfma.svv vv1, vs0, vv0, vv1
  vsw vv1, va1
  vstop
```

Worker Thread




```
for (i=0; i<n; i++) {
  y[i] = a*x[i] + y[i];
}
```

Hwacha Vector-Fetch Architectural Paradigm

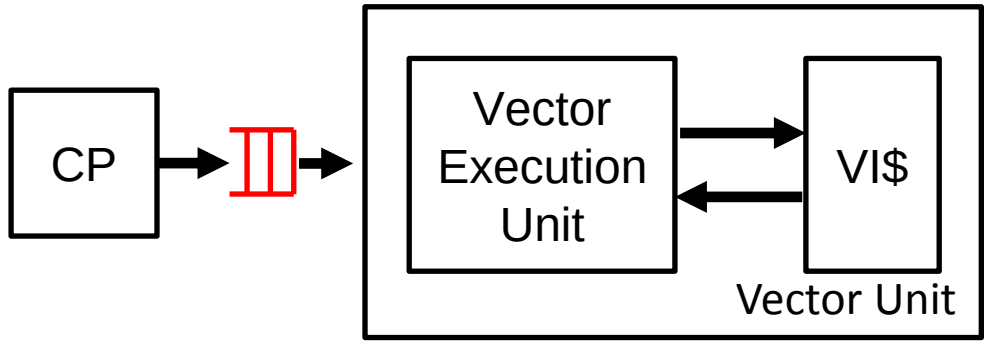
```
a0: n, a1: a,
a2: *x, a3: *y

  vmss vs0, a1
stripmine:
  vsetvl t0, a0
  vmsa va0, a2
  vmsa va1, a3
  vf saxpy
  slli t1, t0, 2
  add a2, a2, t1
  add a3, a3, t1
  sub a0, a0, t0
  bnez a0, stripmine
```

Control Thread

```
saxpy:
  vlw vv0, va0
  vlw vv1, va1
  vfma.svv vv1, vs0, vv0, vv1
  vsw vv1, va1
  vstop
```

Worker Thread




```
for (i=0; i<n; i++) {
  y[i] = a*x[i] + y[i];
}
```

Hwacha Vector-Fetch Architectural Paradigm

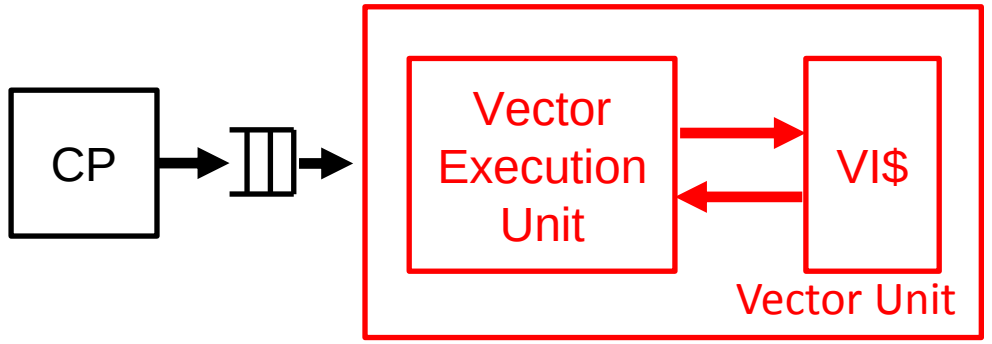
```
a0: n, a1: a,
a2: *x, a3: *y

  vmcs vs0, a1
stripmine:
  vsetvl t0, a0
  vmca va0, a2
  vmca va1, a3
  vf saxpy
  slli t1, t0, 2
  add a2, a2, t1
  add a3, a3, t1
  sub a0, a0, t0
  bnez a0, stripmine
```

Control Thread

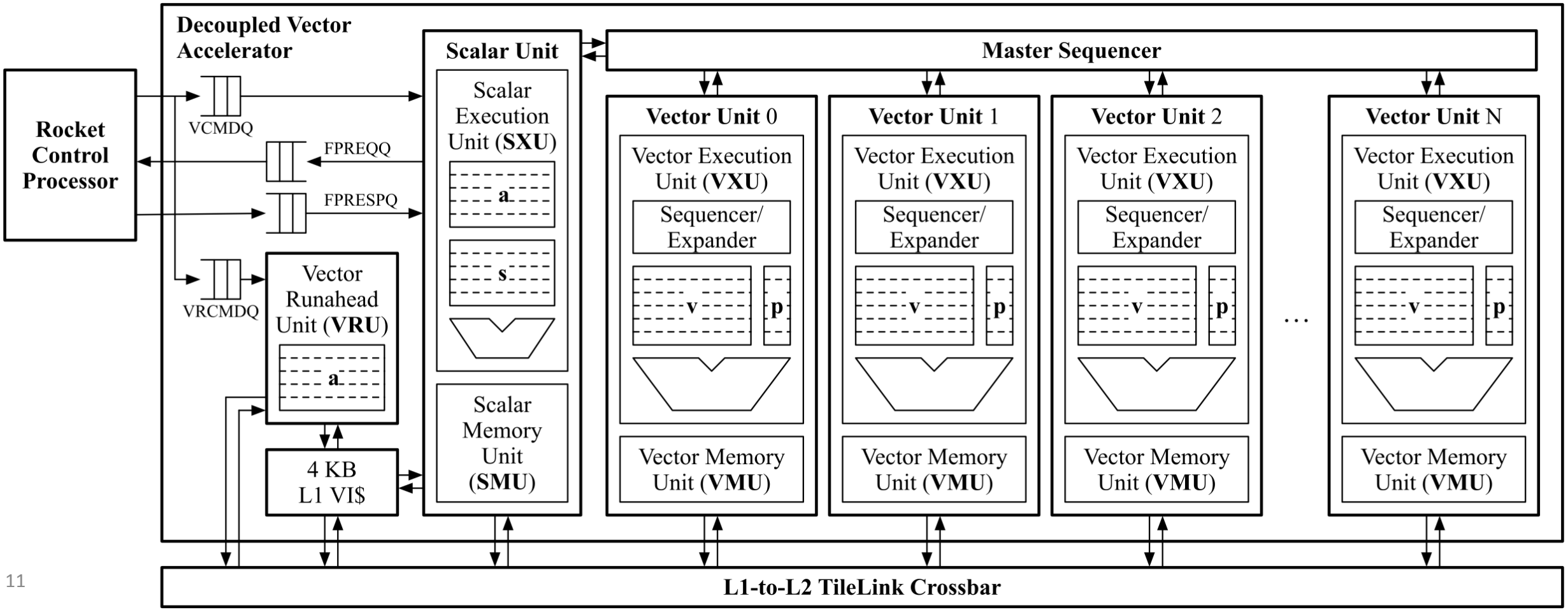
```
saxpy:
  vlw vv0, va0
  vlw vv1, va1
  vfma.svv vv1, vs0, vv0, vv1
  vsw vv1, va1
  vstop
```

Worker Thread

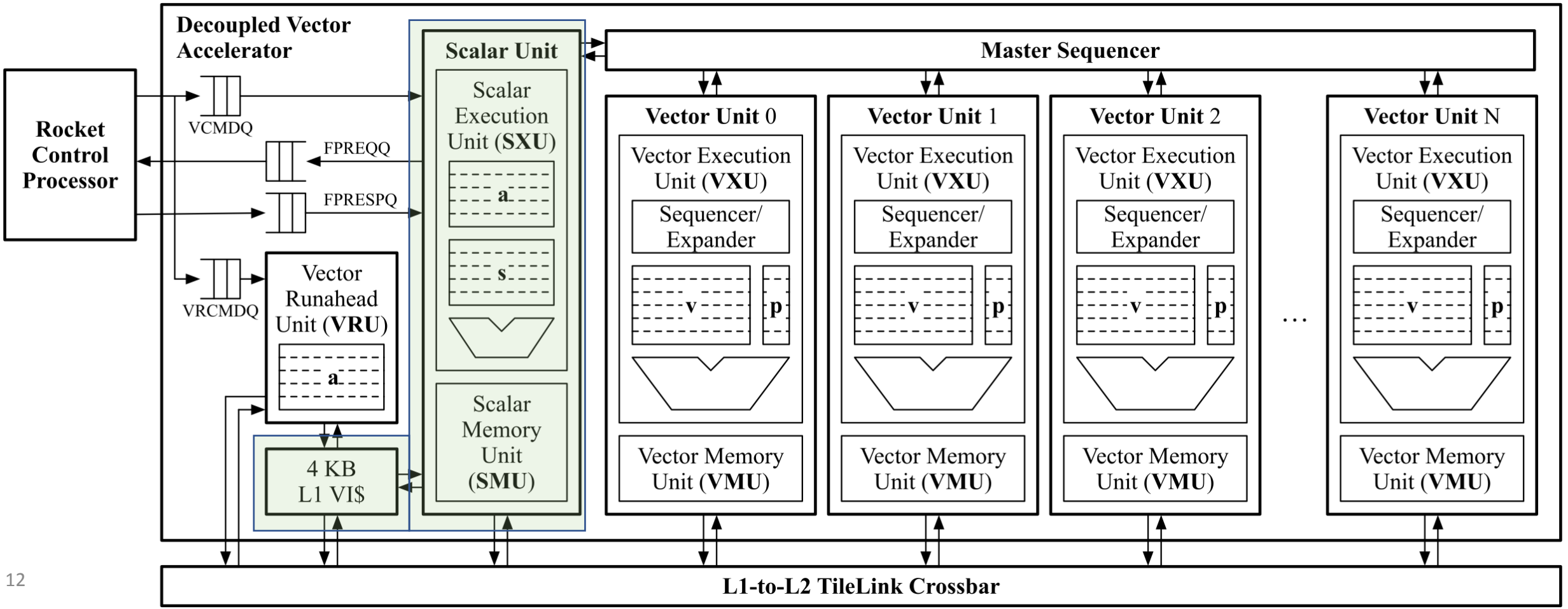



```
for (i=0; i<n; i++) {
  y[i] = a*x[i] + y[i];
}
```

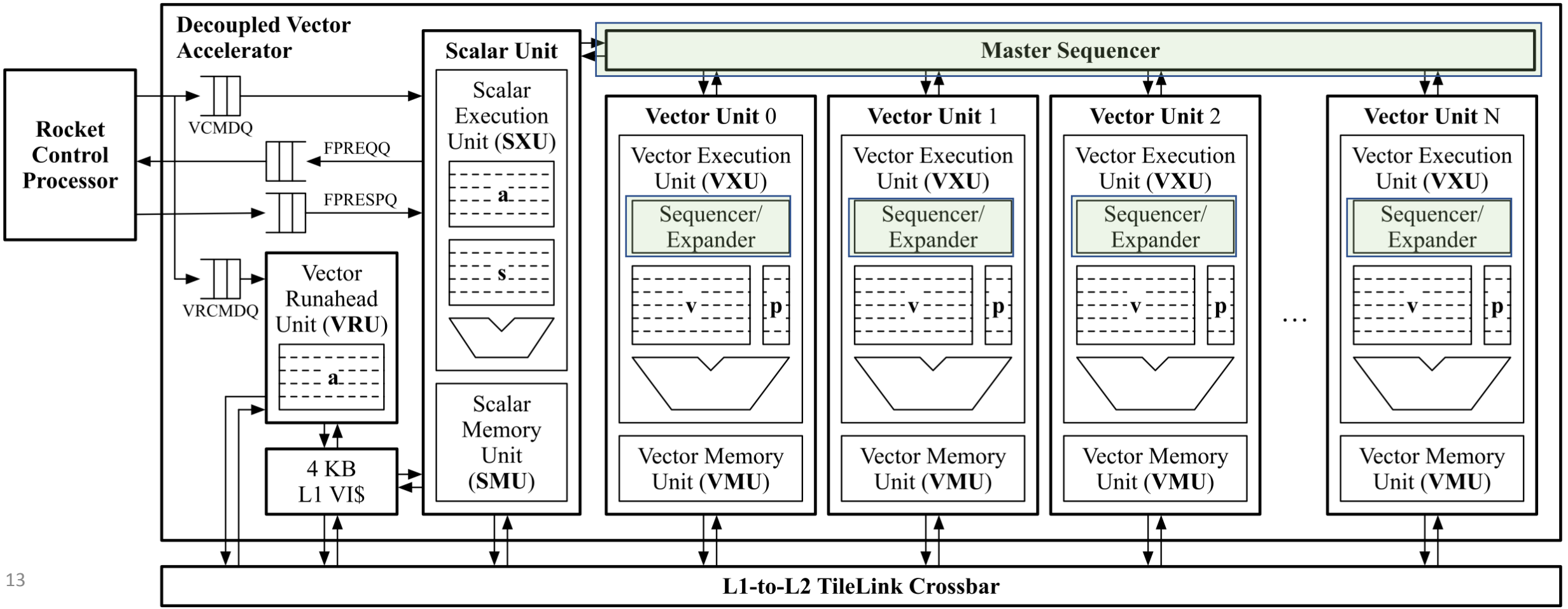
Hwacha Microarchitecture



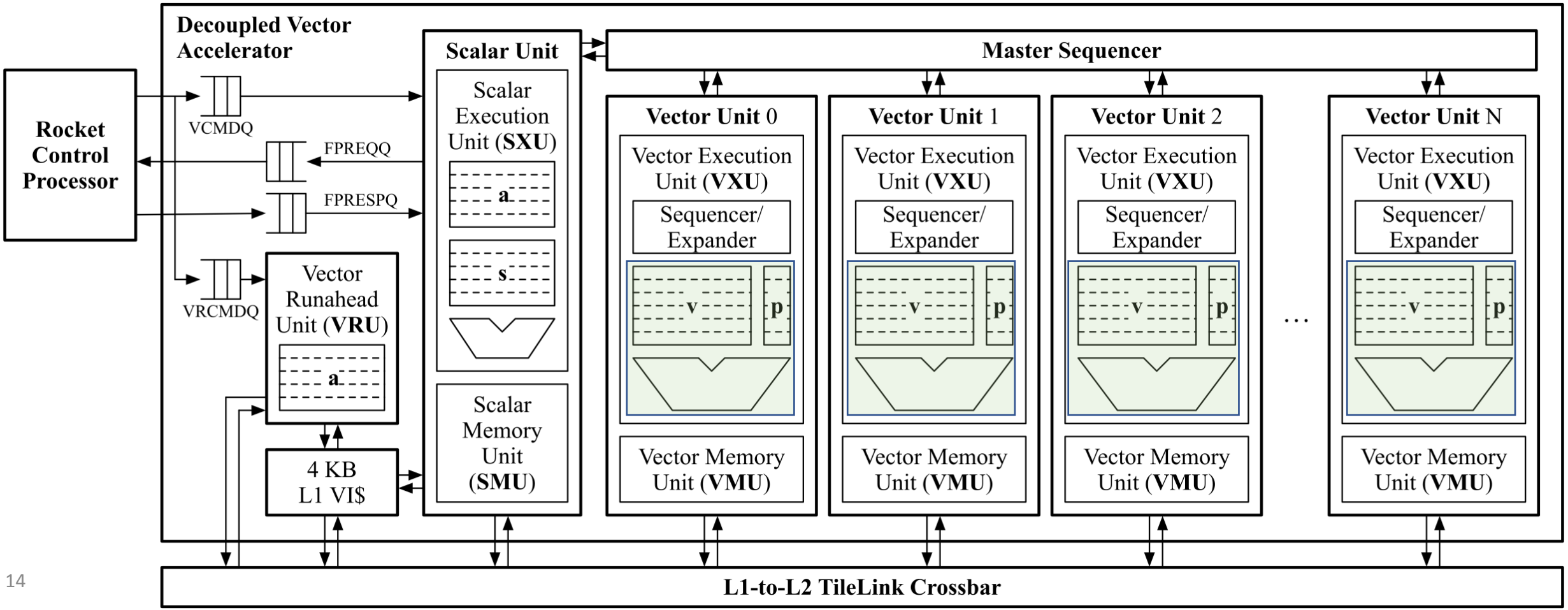
Hwacha Microarchitecture



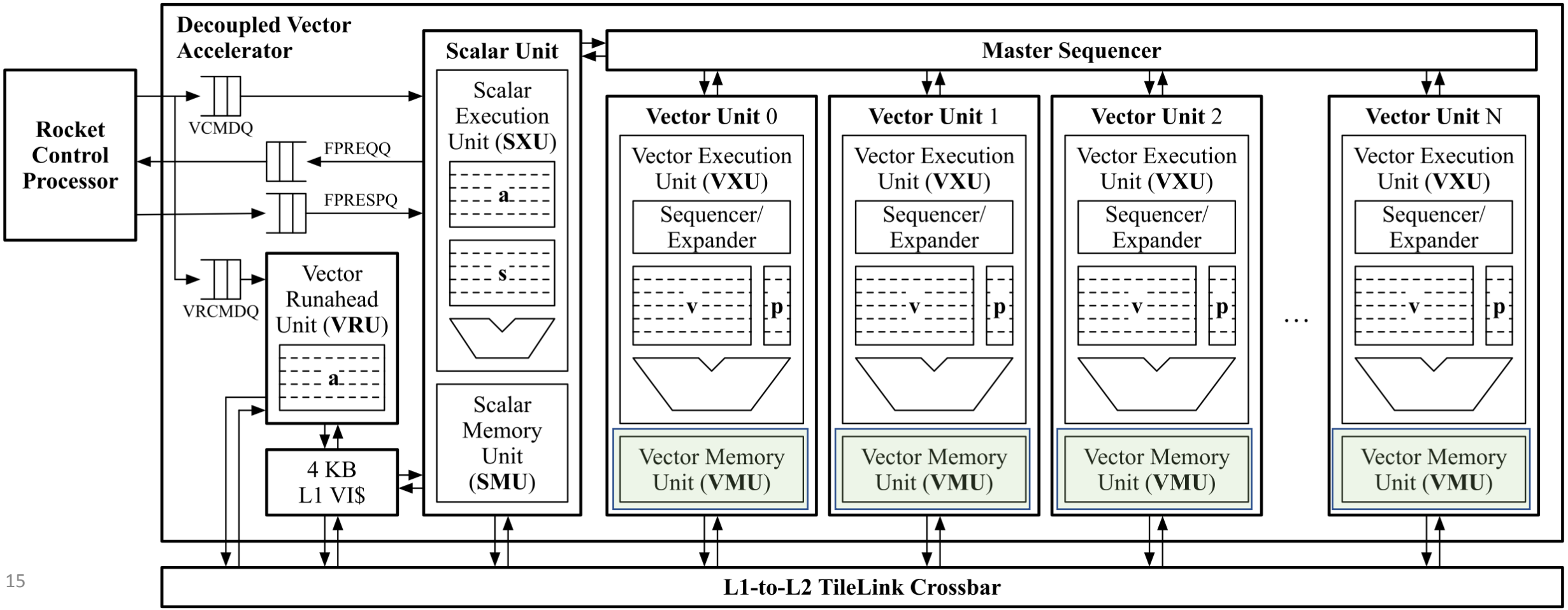
Hwacha Microarchitecture



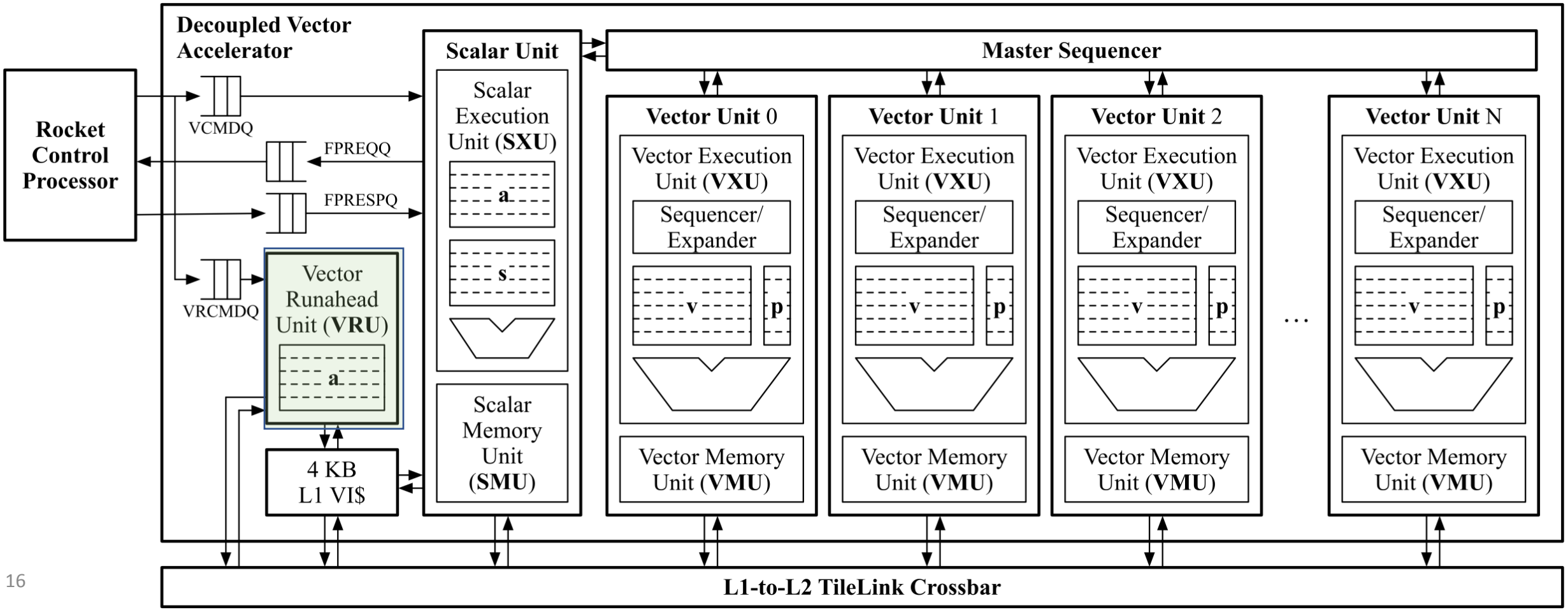
Hwacha Microarchitecture



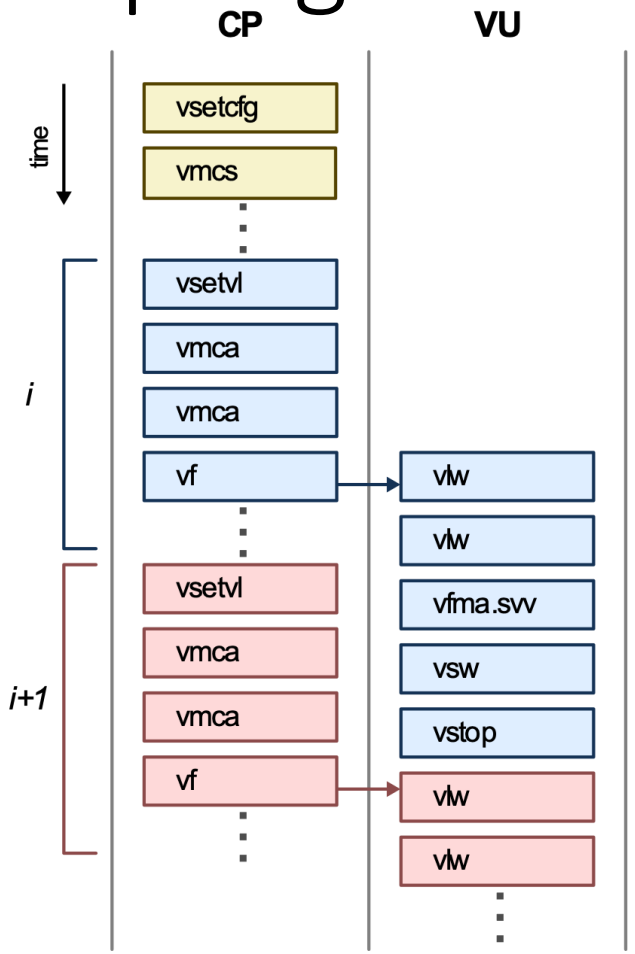
Hwacha Microarchitecture

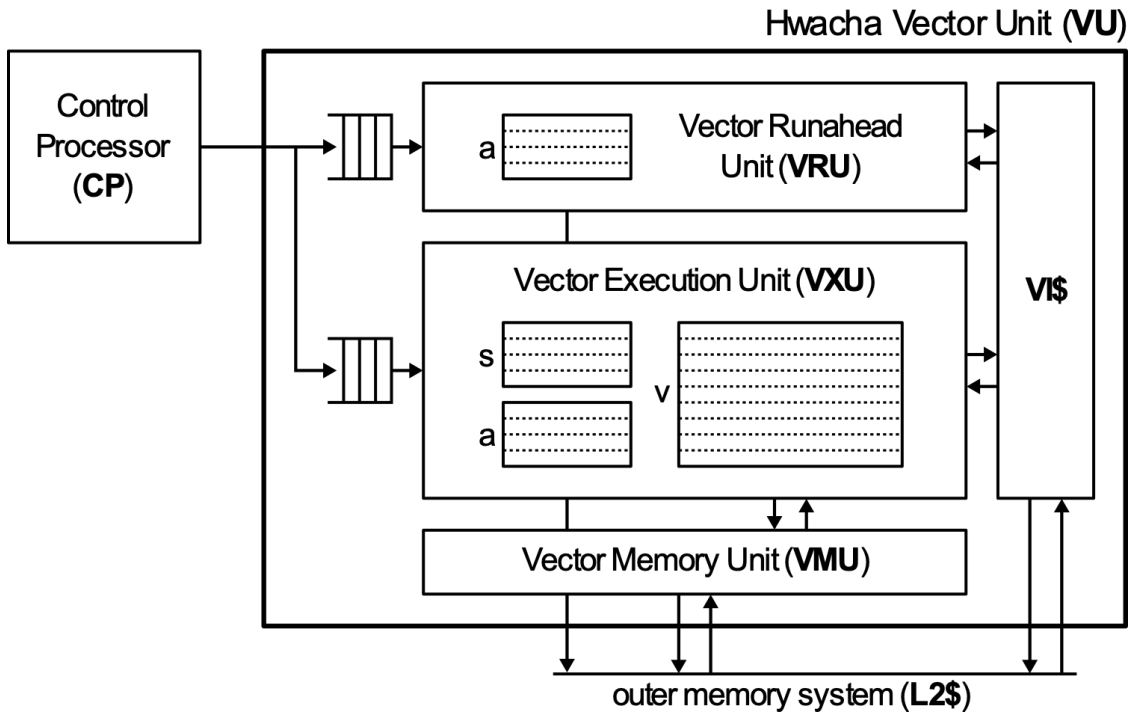


Hwacha Microarchitecture



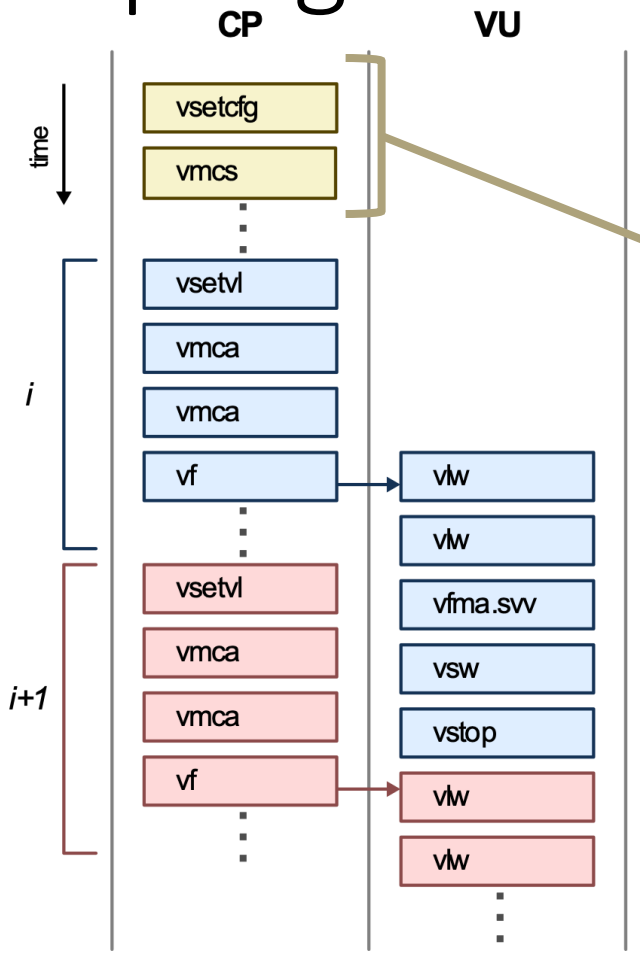
Decoupling

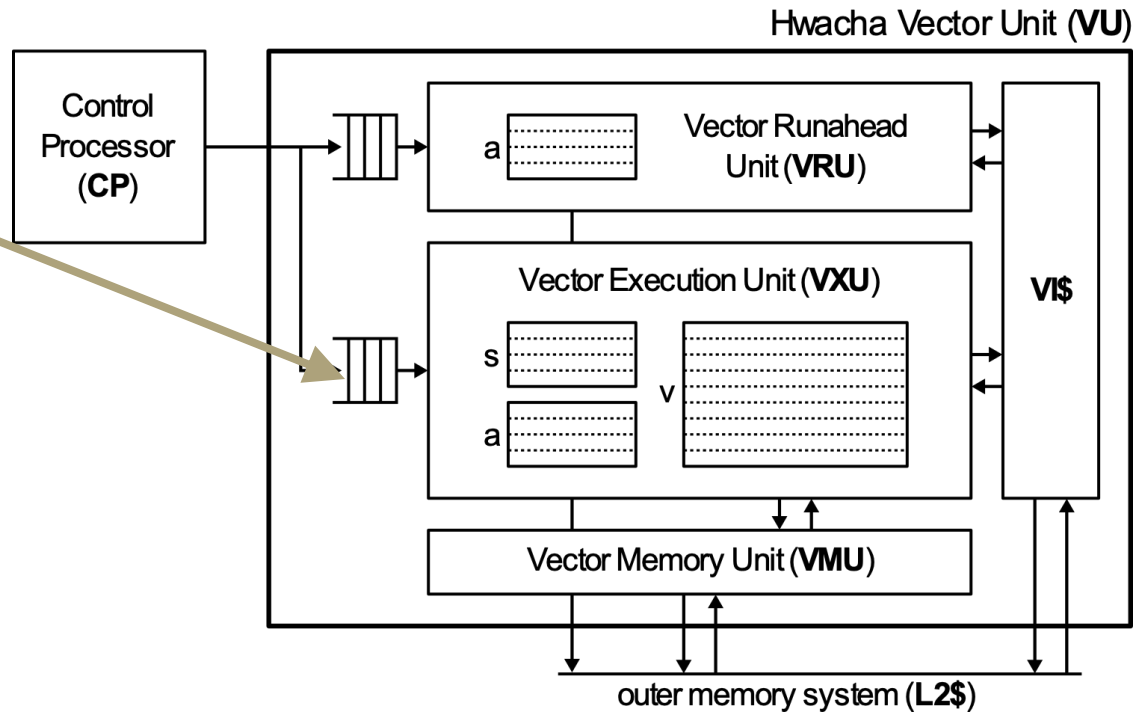




- Exploit regularity of constant-stride vector memory accesses for aggressive yet accurate prefetching
- Extensive decoupling in microarchitecture to tolerate latency and resolve memory references as early as possible

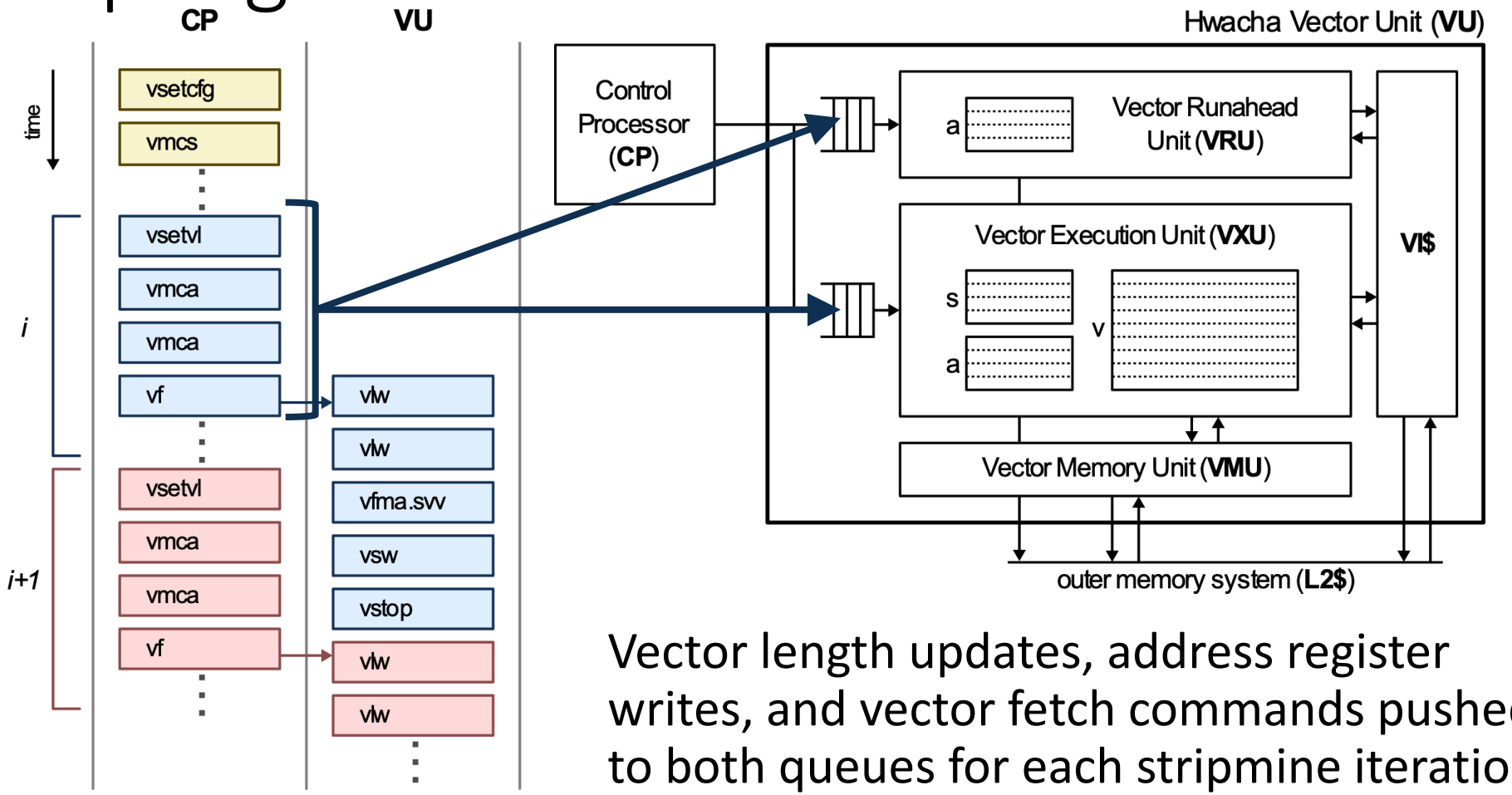
Decoupling





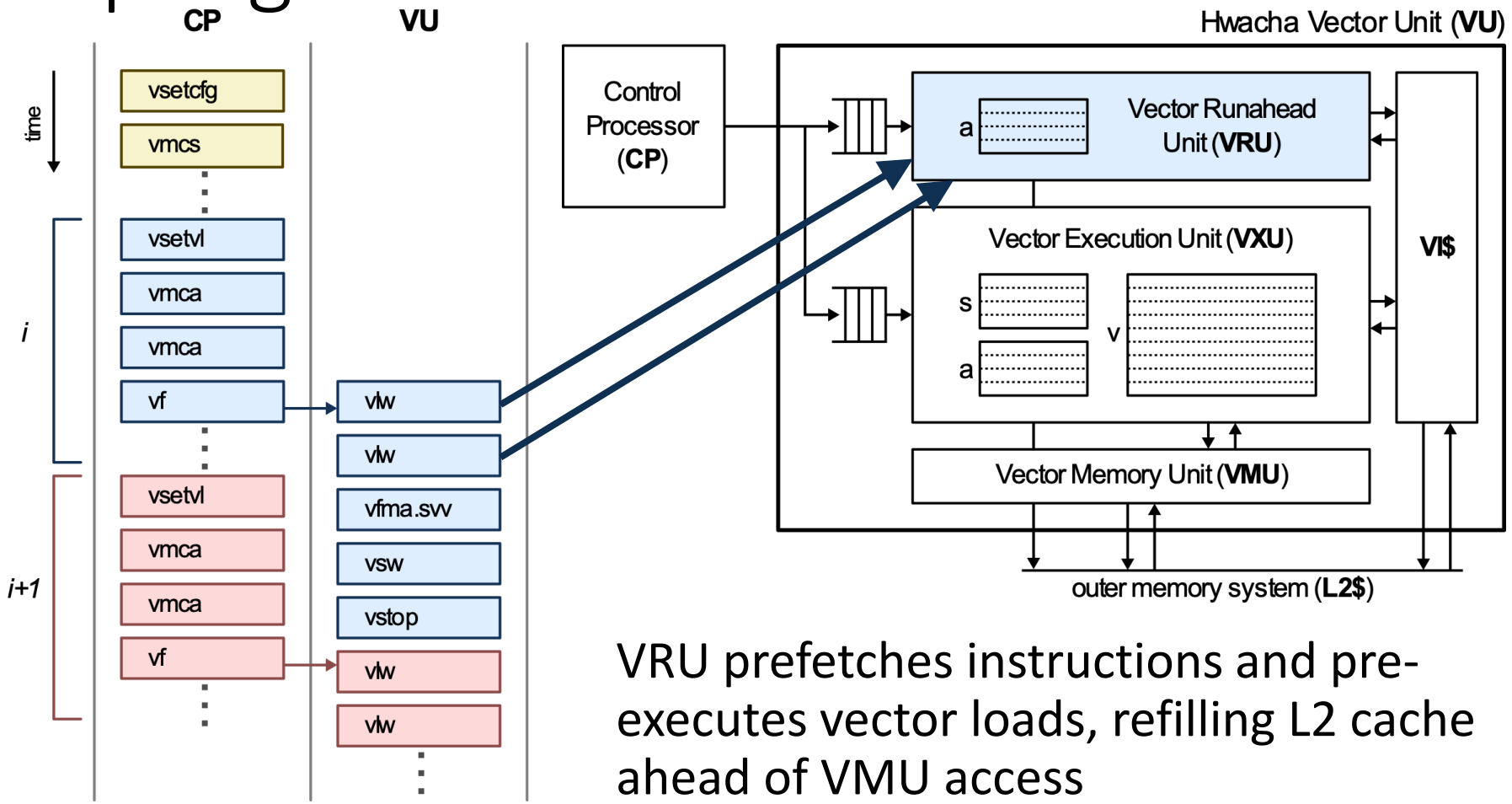
Before stripmine loop, control thread pushes scalar register write to VXU command queue

Decoupling



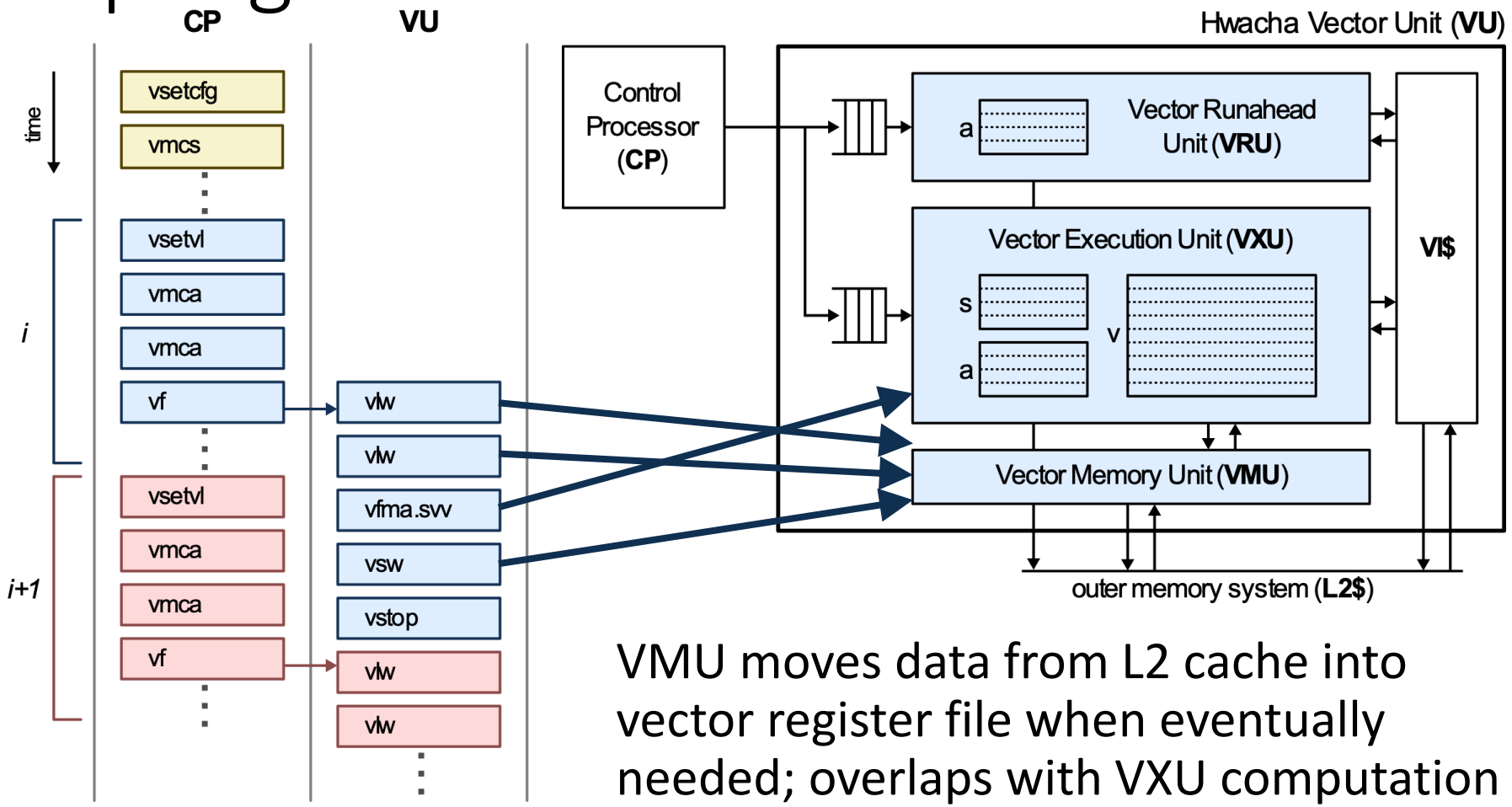
Vector length updates, address register writes, and vector fetch commands pushed to both queues for each stripmine iteration

Decoupling



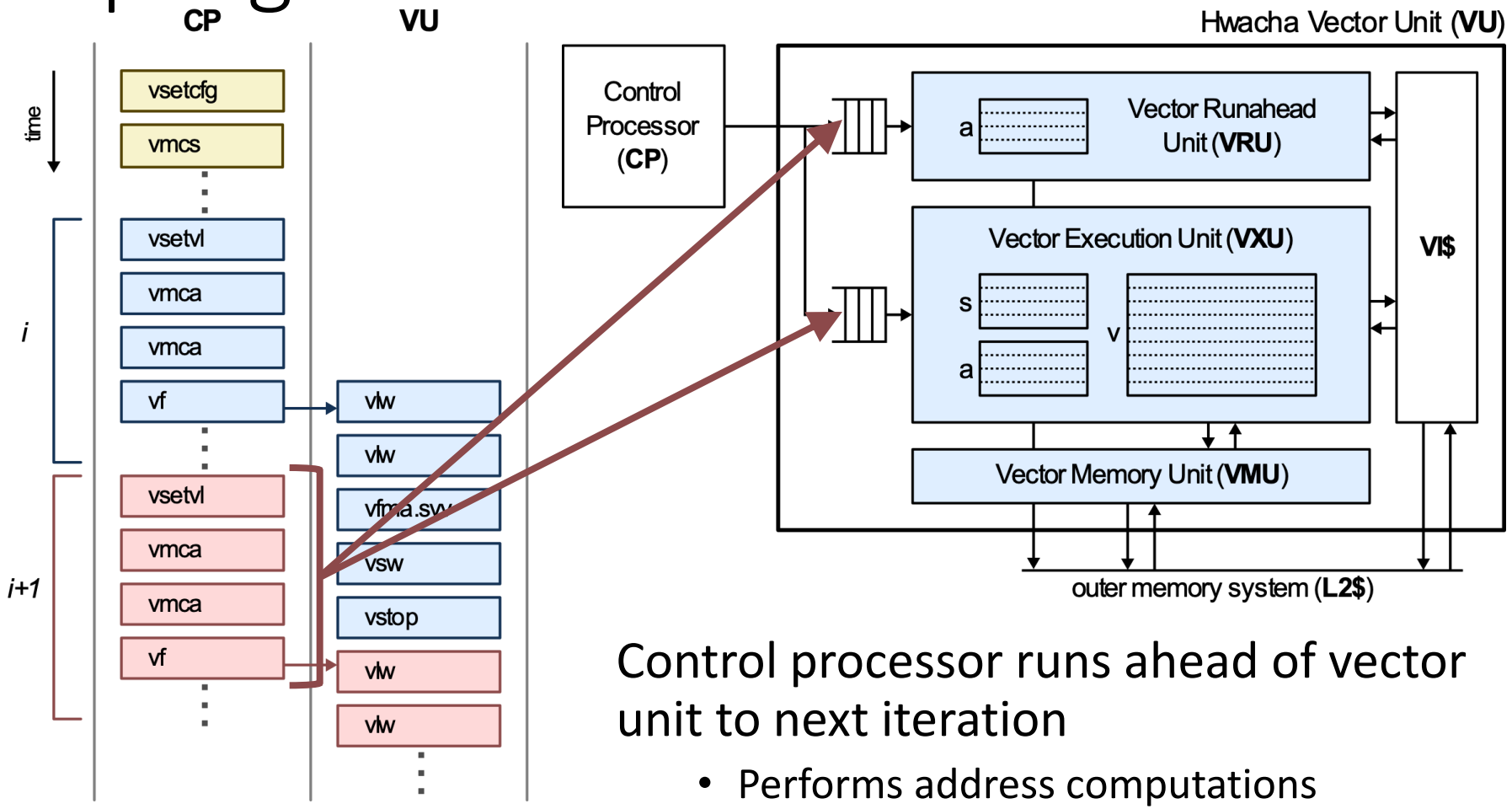
VRU prefetches instructions and pre-executes vector loads, refilling L2 cache ahead of VMU access

Decoupling



VMU moves data from L2 cache into vector register file when eventually needed; overlaps with VXU computation

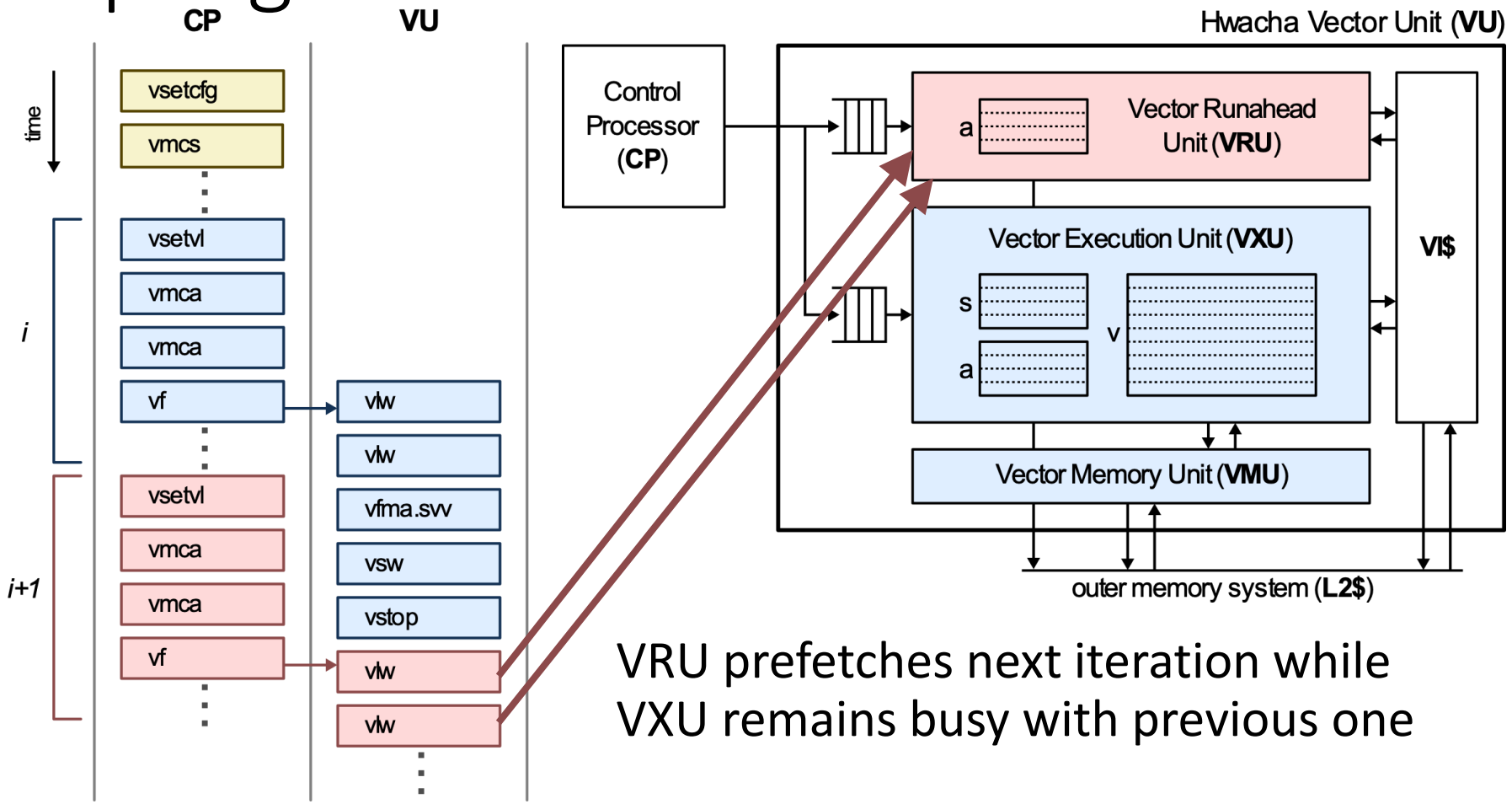
Decoupling



Control processor runs ahead of vector unit to next iteration

- Performs address computations
- Pushes next vector fetch command

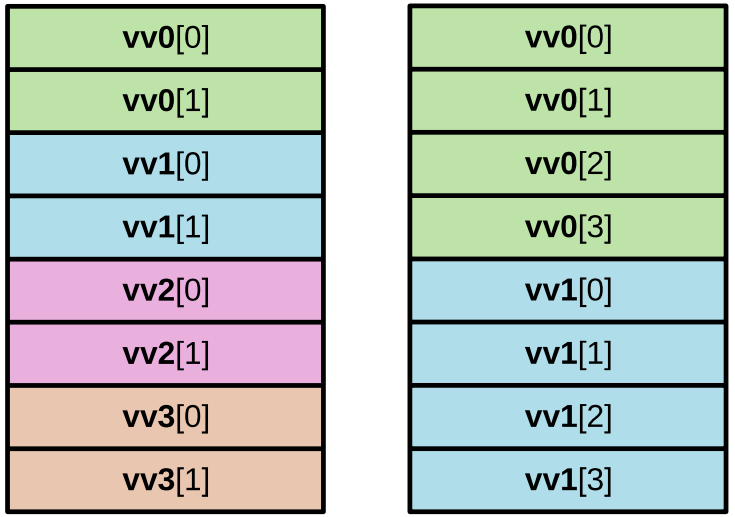
Decoupling



VRU prefetches next iteration while
VXU remains busy with previous one

Reconfigurable Vector Register File

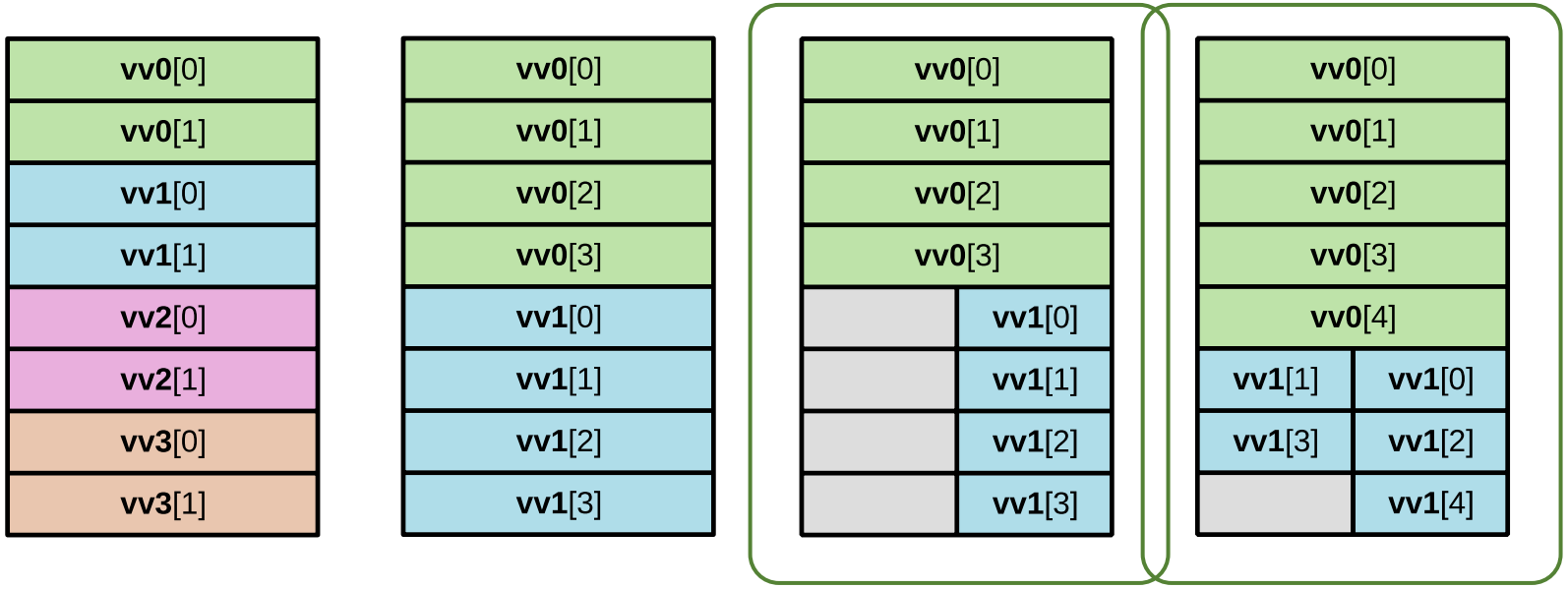
- Software specifies number of vector data and predicate registers
- Hardware dynamically remaps elements to physical storage



- Maximum hardware vector length automatically extends to fill available capacity
- Exchange unused architectural registers for longer hardware vectors

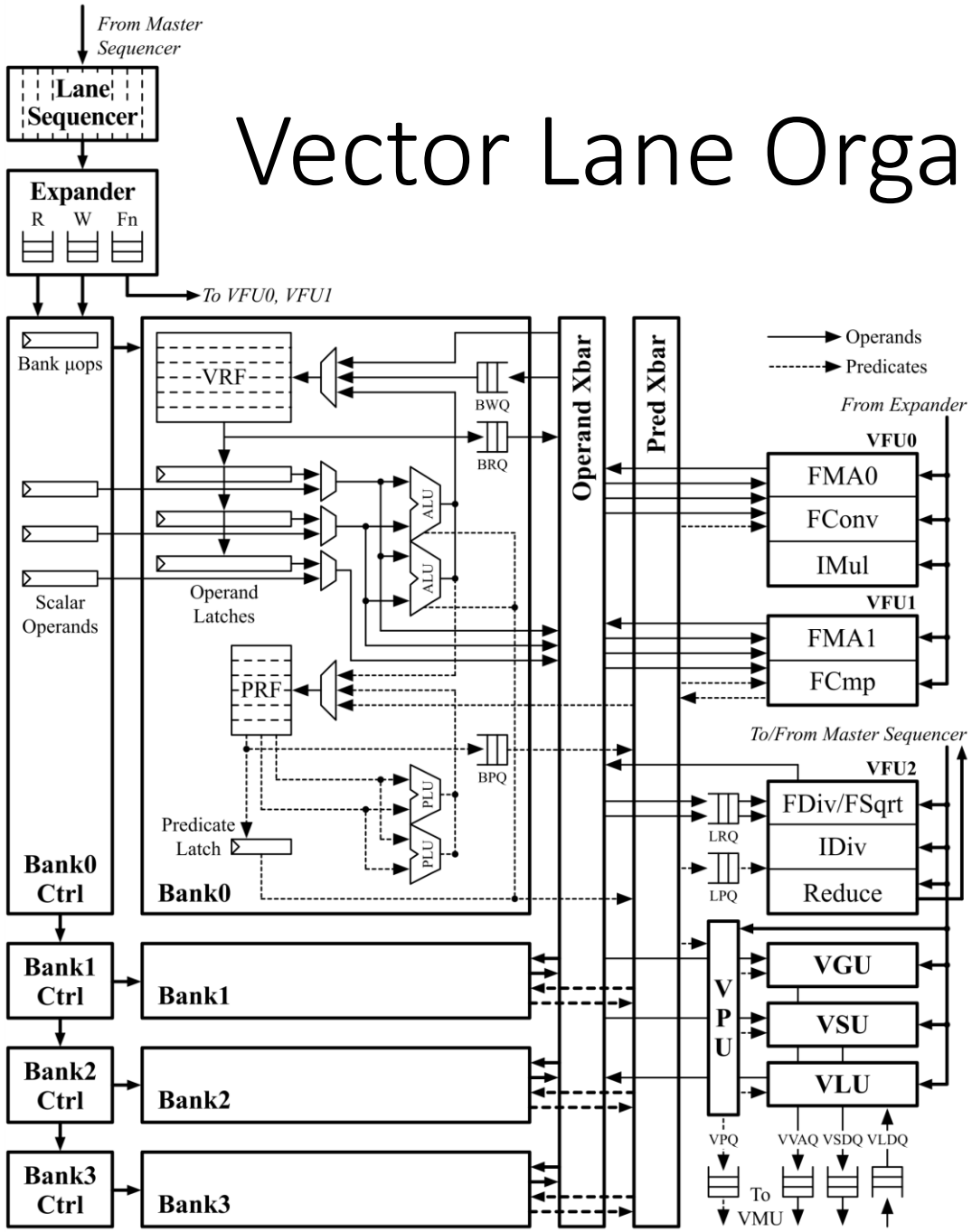
Mixed-Precision Support

- Configurable element widths for individual vector registers
- Subword register packing and alignment of mixed-precision operands are managed by hardware – software remains fully oblivious



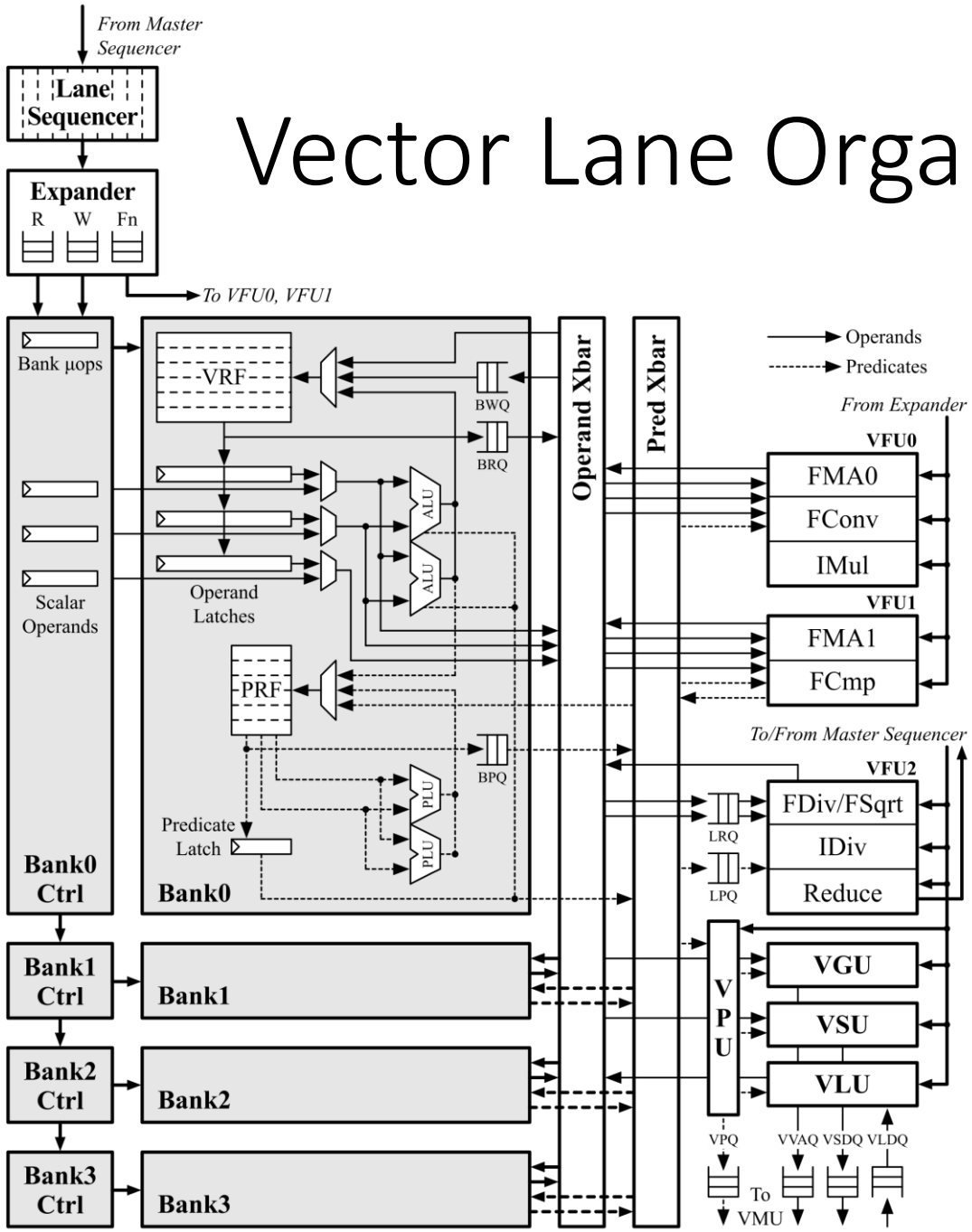

```
vsetcfg 1, 1
   vlen = 5
```

Vector Lane Organization



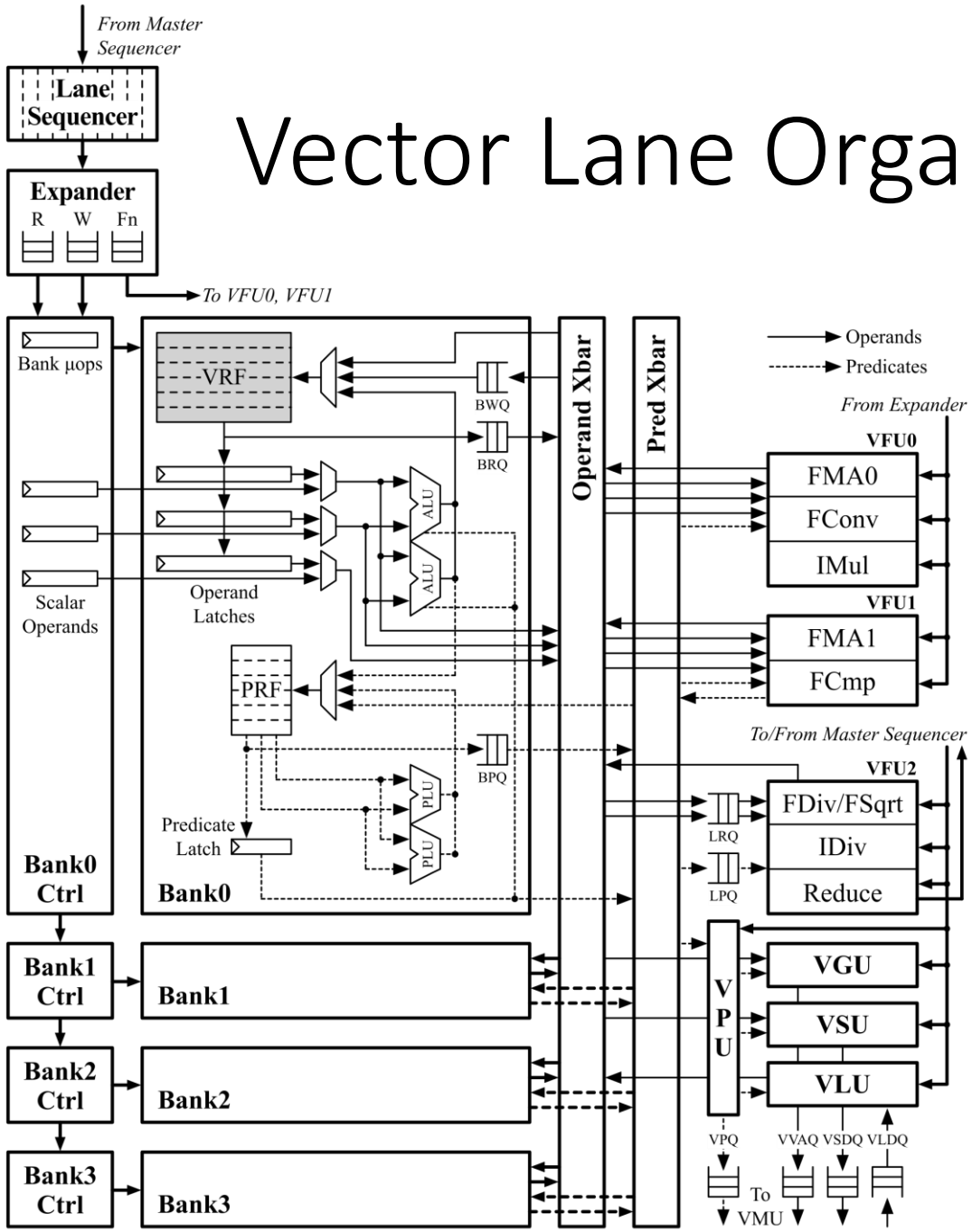
- Compact register file of four 1R/1W SRAM banks
- Per-bank integer ALU/PLU
- Two independently scheduled FMA clusters
  - Total of four double-precision FMAs per cycle
- Pipelined integer multiplier
- Variable-latency decoupled functional units
  - Integer divide
  - Floating-point divide with square root
  - Reduction

Vector Lane Organization



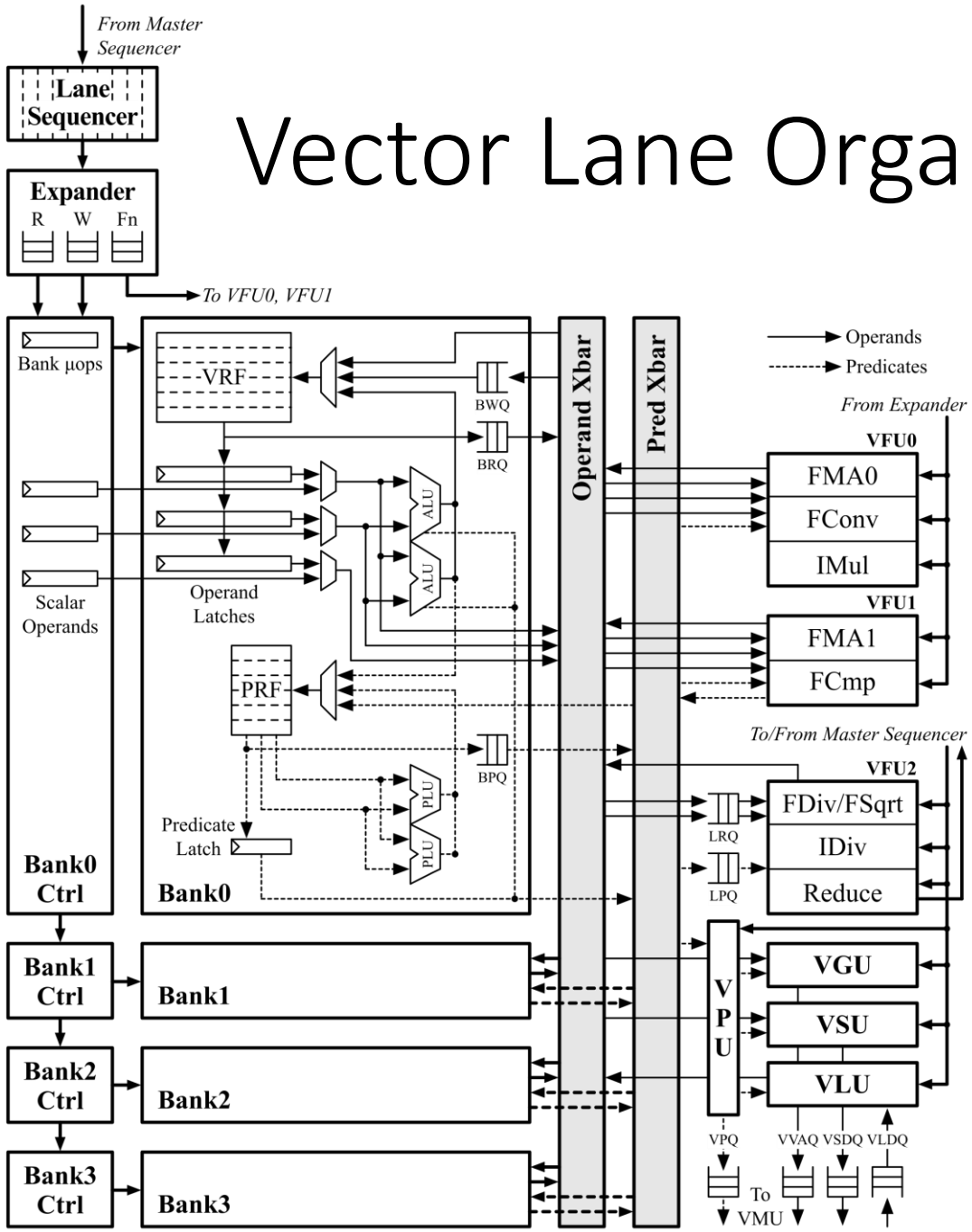
- Compact register file of four 1R/1W SRAM banks
- Per-bank integer ALU/PLU
- Two independently scheduled FMA clusters
  - Total of four double-precision FMAs per cycle
- Pipelined integer multiplier
- Variable-latency decoupled functional units
  - Integer divide
  - Floating-point divide with square root
  - Reduction

Vector Lane Organization



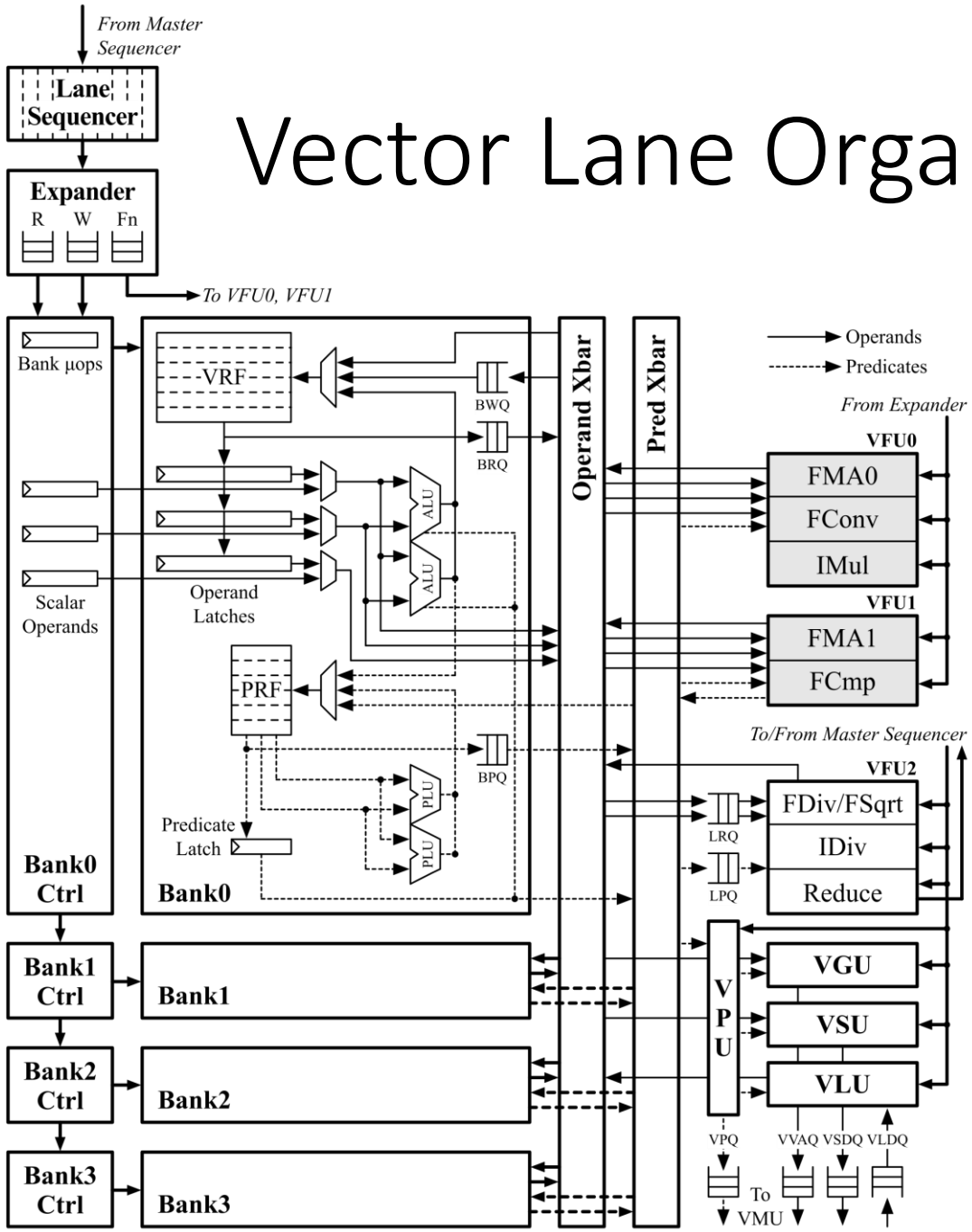
- Compact register file of four 1R/1W SRAM banks
- Per-bank integer ALU/PLU
- Two independently scheduled FMA clusters
  - Total of four double-precision FMAs per cycle
- Pipelined integer multiplier
- Variable-latency decoupled functional units
  - Integer divide
  - Floating-point divide with square root
  - Reduction

Vector Lane Organization



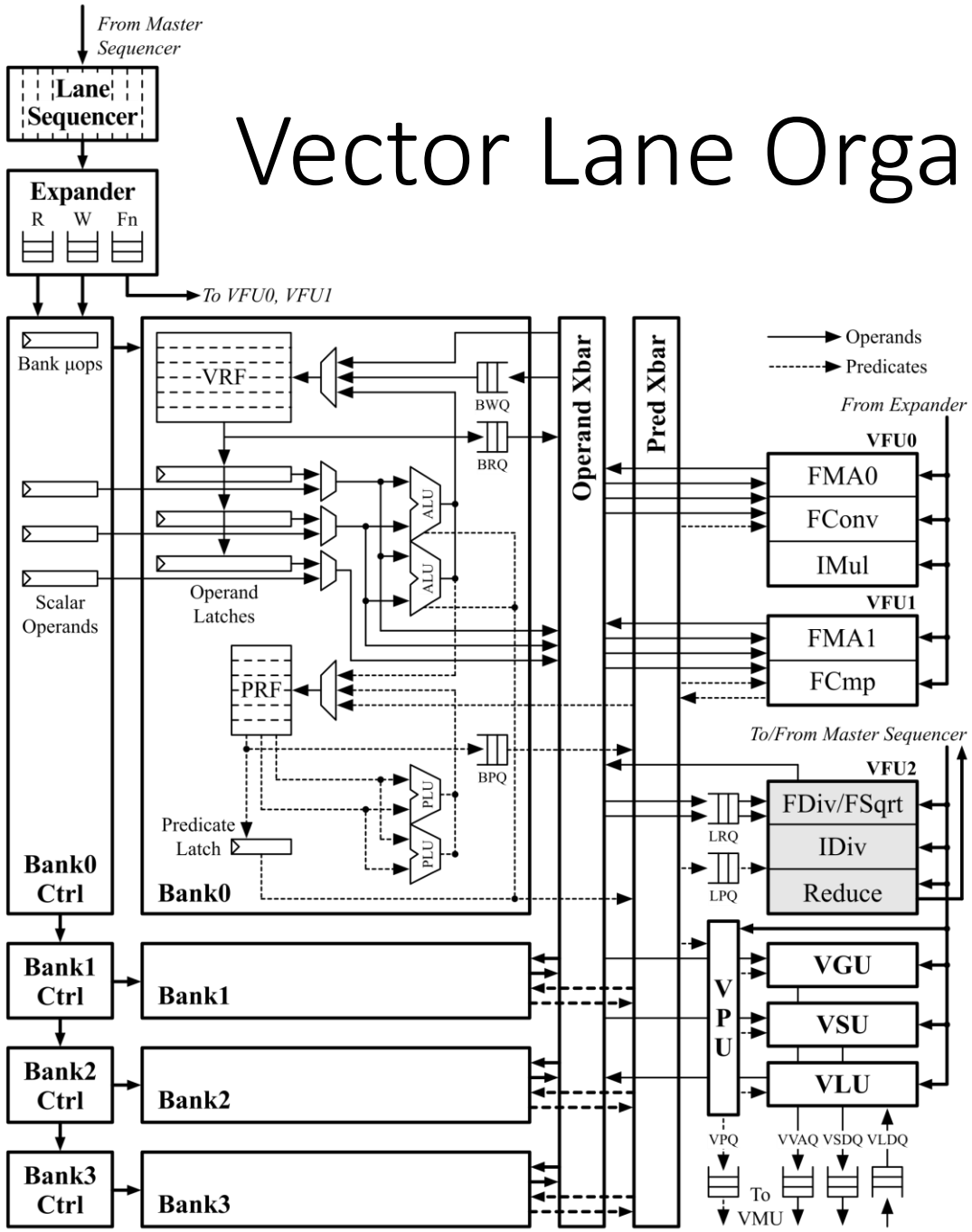
- Compact register file of four 1R/1W SRAM banks
- Per-bank integer ALU/PLU
- Two independently scheduled FMA clusters
  - Total of four double-precision FMAs per cycle
- Pipelined integer multiplier
- Variable-latency decoupled functional units
  - Integer divide
  - Floating-point divide with square root
  - Reduction

Vector Lane Organization



- Compact register file of four 1R/1W SRAM banks
- Per-bank integer ALU/PLU
- Two independently scheduled FMA clusters
  - Total of four double-precision FMAs per cycle
- Pipelined integer multiplier
- Variable-latency decoupled functional units
  - Integer divide
  - Floating-point divide with square root
  - Reduction

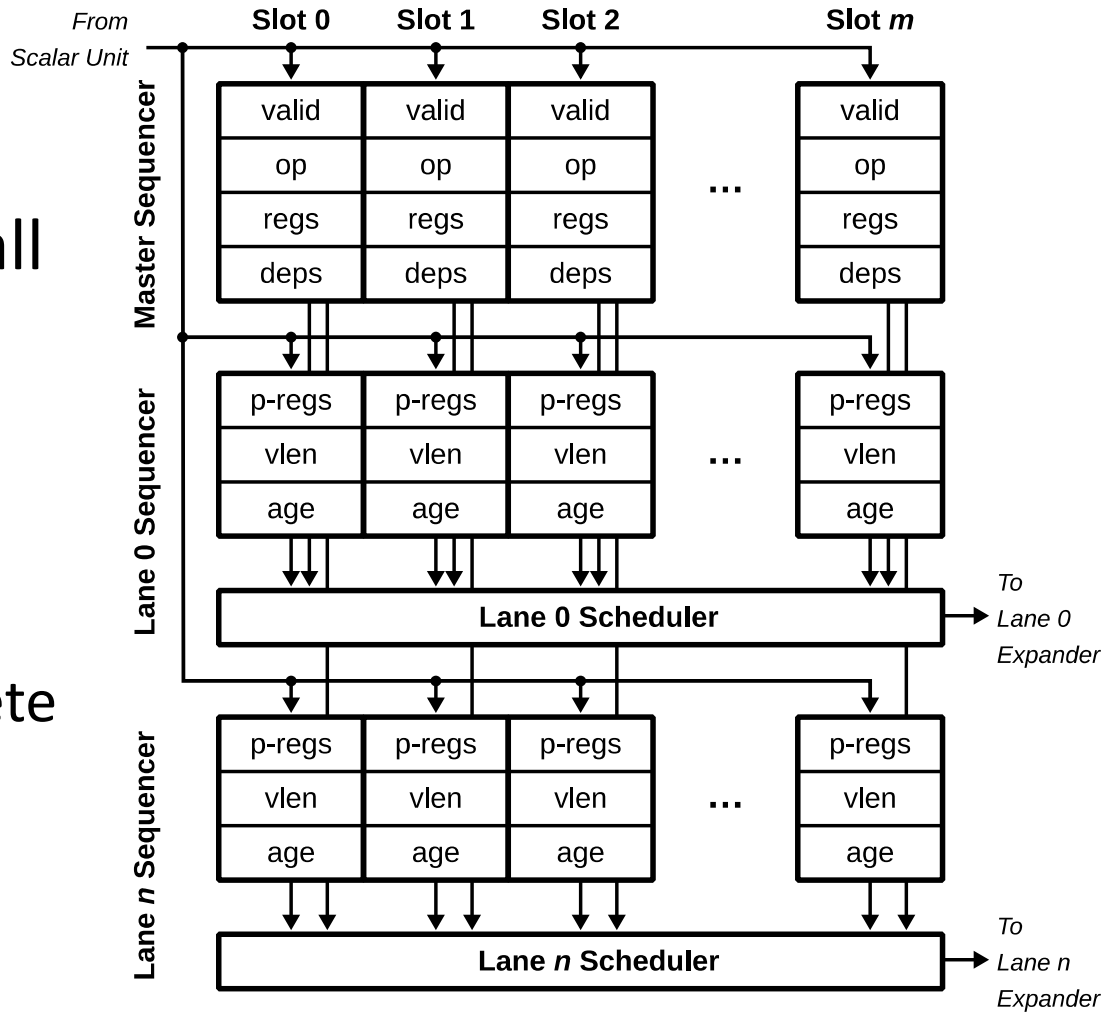
Vector Lane Organization



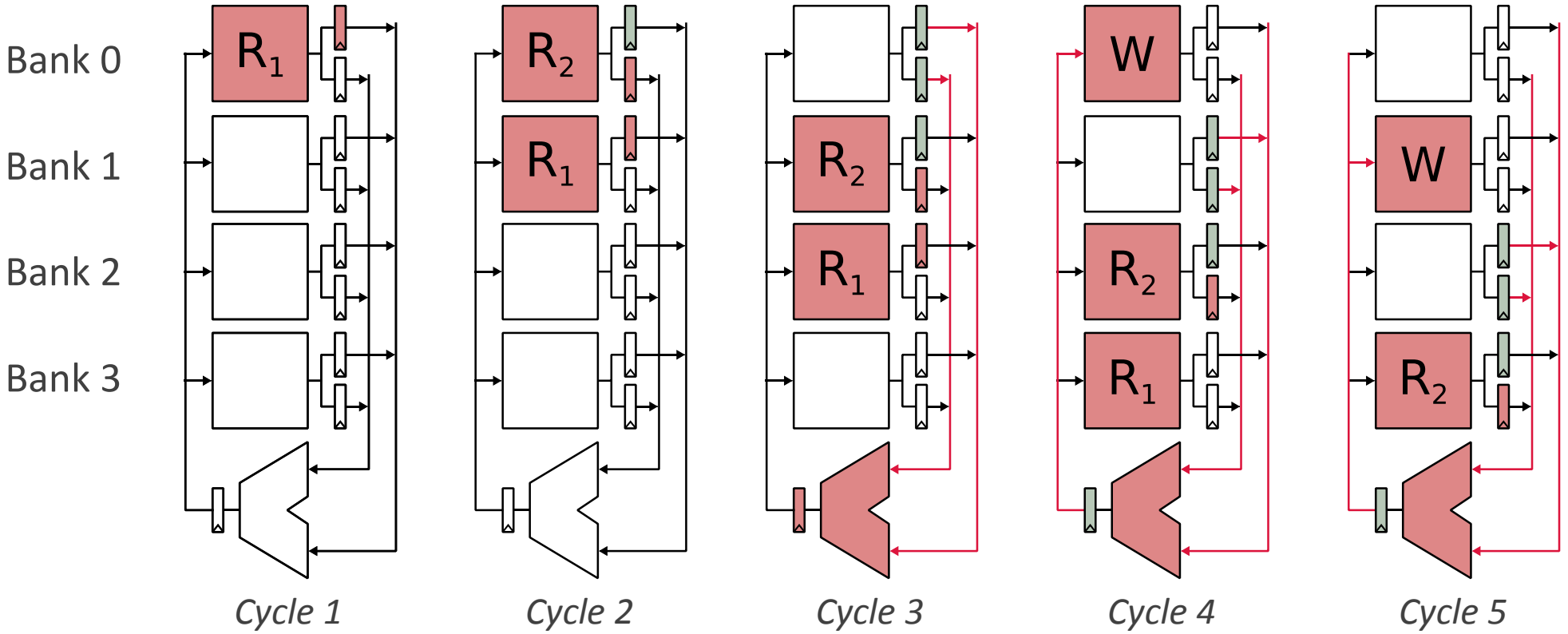
- Compact register file of four 1R/1W SRAM banks
- Per-bank integer ALU/PLU
- Two independently scheduled FMA clusters
  - Total of four double-precision FMAs per cycle
- Pipelined integer multiplier
- Variable-latency decoupled functional units
  - Integer divide
  - Floating-point divide with square root
  - Reduction

Sequencer

- Instruction window
- Group of 8 elements striped across all four banks forms atomic unit of scheduling within lane
- Issues to expander after all hazards resolved for element group
  - Vector ILP: Element groups from different instructions can issue/complete out-of-order
  - Expander converts sequencer ops to µops – low-level control signals that drive lane datapath



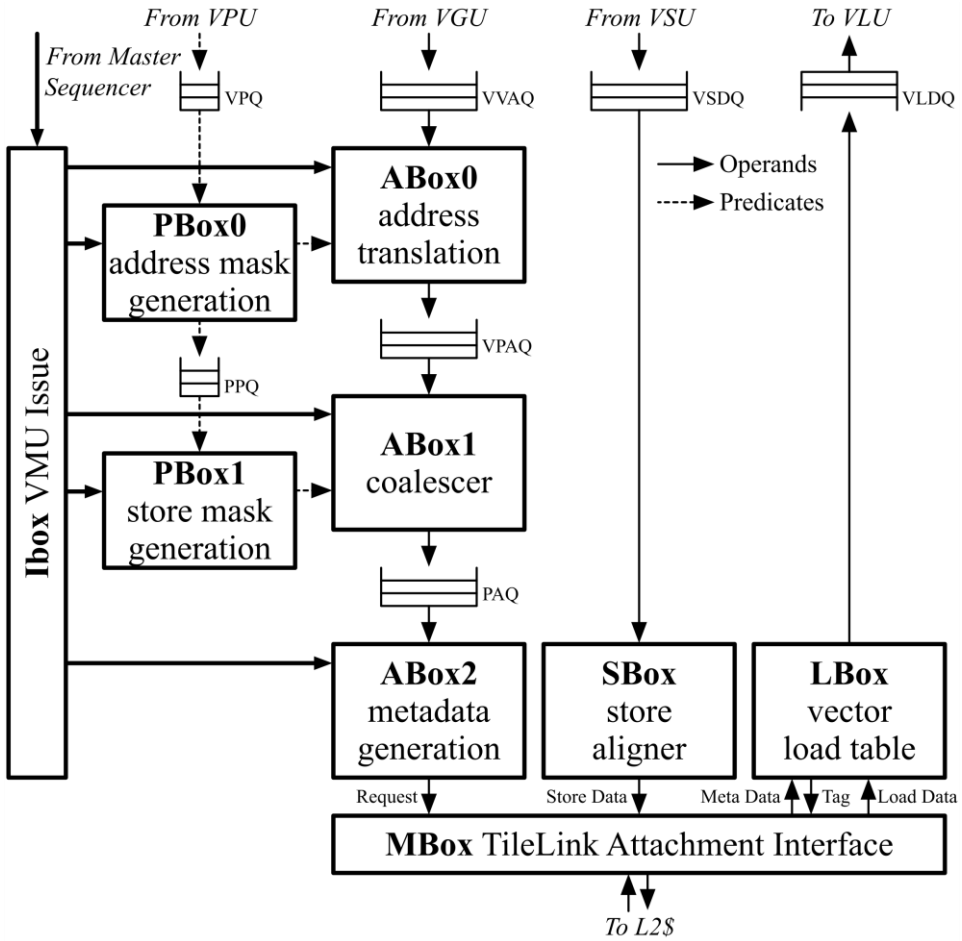
Systolic Bank Execution



- Sustains *n* operands/cycle after *n*-cycle initial latency
- Stall-free cascade of µops (“fire and forget”) after clearing hazards
- Striped element mapping avoids bank conflicts

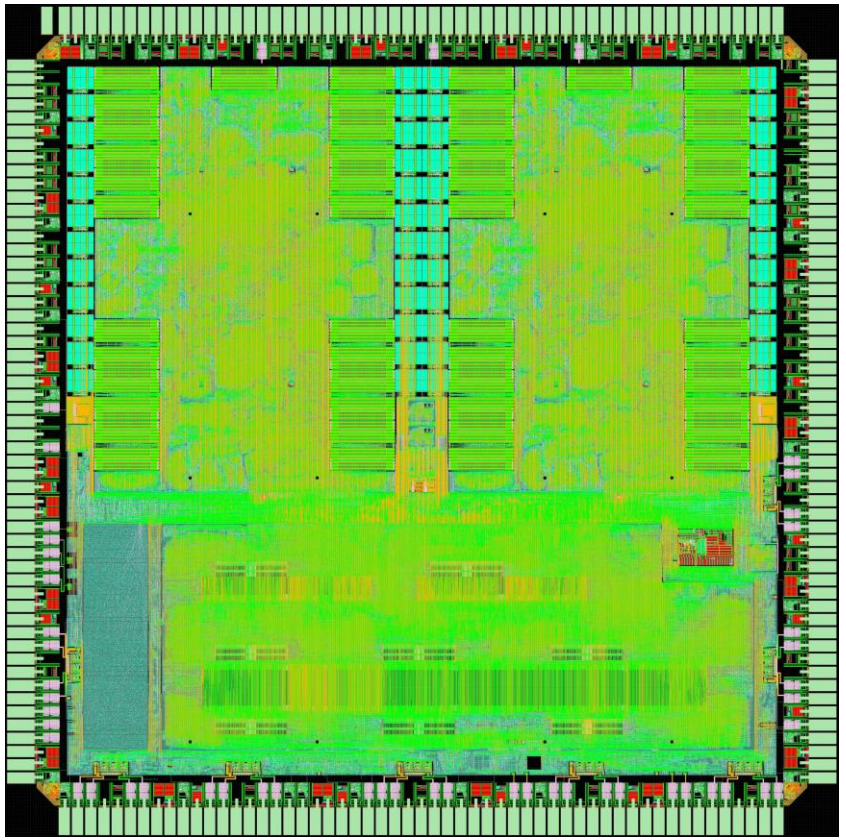
Vector Memory Unit

- 128-bit TileLink2 interface to L2 cache
- Coalesced accesses for unit-stride memory operations
- Optimizations to handle response reordering for loads:
  - Out-of-order writeback mechanism to minimize buffering
  - Support for “tailgating” overlapped load operations



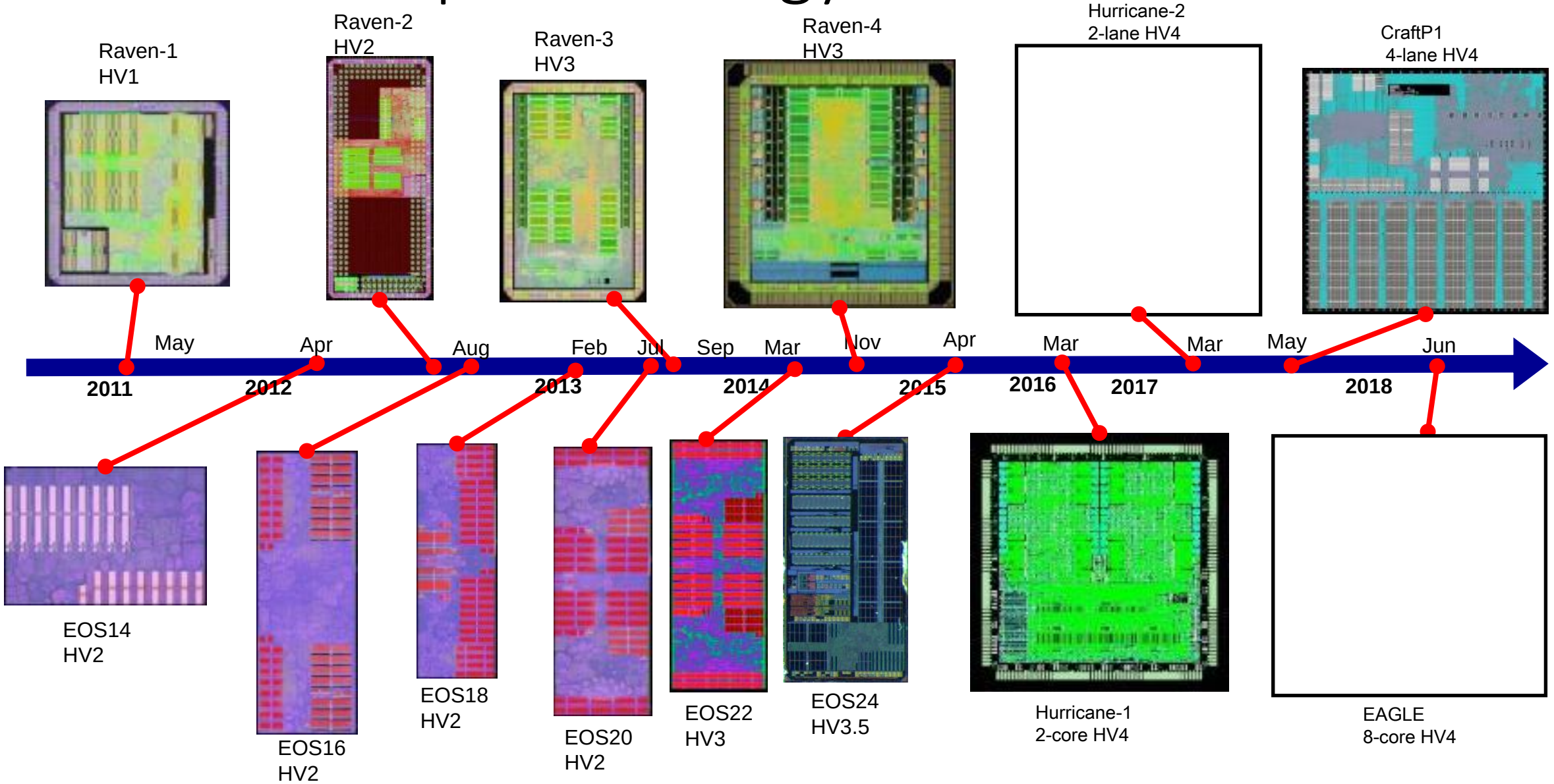
Benchmarks/Tape-outs

- Silicon-proven: 10+ tape-outs
- Peak power efficiency: 40+ DP-GFLOPS/W in ST 28nm FD-SOI
- Peak floating-point performance: 64+ DP-GFLOPS, 128+ SP-GFLOPS, 256+ HP-GFLOPS in TSMC 16nm 5x5mm
- 95+% of peak performance on DGEMM
- 10X acceleration over Rocket on Caffe NN with single lane



2.8x2.8mm
ST 28nm FD-SOI

Hwacha Chip Chronology



Raven/Hurricane: ST 28nm FD-SOI; EOS: IBM 45nm SOI; CRAFT/EAGLE:TSMC16nm

Current/Future Work

- Move frontend to the standard RISC-V Vector extension
  - Retain lane structure
- Extensions to RVV for polymorphic instructions
- Explore domain-specific extensions

- Draft of RVV spec at github.com/riscv/riscv-v-spec

Open Source Release

Berkeley Architecture Research

- Full parameterized Chisel3 RTL
  - Multi-lane, Sequencer slots, queue depths
- ISA assembly tests, hand-coded micro-benchmark suite
- Random torture test generator for verification
- Spike extension for simple software simulation
- FireSim support for ~90MHz FPGA simulation
- GNU binutils assembler
- Preliminary OpenCL compiler based on LLVM+pocl
- Documentation: technical reports

Acknowledgements

Berkeley Architecture Research

- Yunsup Lee
- Alon Amid, Sagar Karandikar, Quan Nguyen, Stephen Twigg, Huy Vo, Andrew Waterman, Jerry Zhao
- All students who worked on these chips as part of the following Berkeley labs and centers: ParLab, ASPIRE, ADEPT, BWRC

Sponsors:

Research partially funded by DARPA CRAFT HR0011-16-C-0052 and PERFECT HR0011-12-2-0016; Intel Science and Technology Center for Agile Design; and ADEPT Lab industrial sponsors and affiliates Intel, Google, Huawei, Siemens, SK Hynix, Seagate.

Questions and Links

RTL: github.com/ucb-bar/hwacha

Example Project: github.com/ucb-bar/hwacha-template

Toolchain+spike+tests: github.com/ucb-bar/esp-tools

FireSim: github.com/firesim/firesim/tree/hwacha

OpenCL+LLVM: github.com/ucb-bar/esp-llvm

Docs: hwacha.org