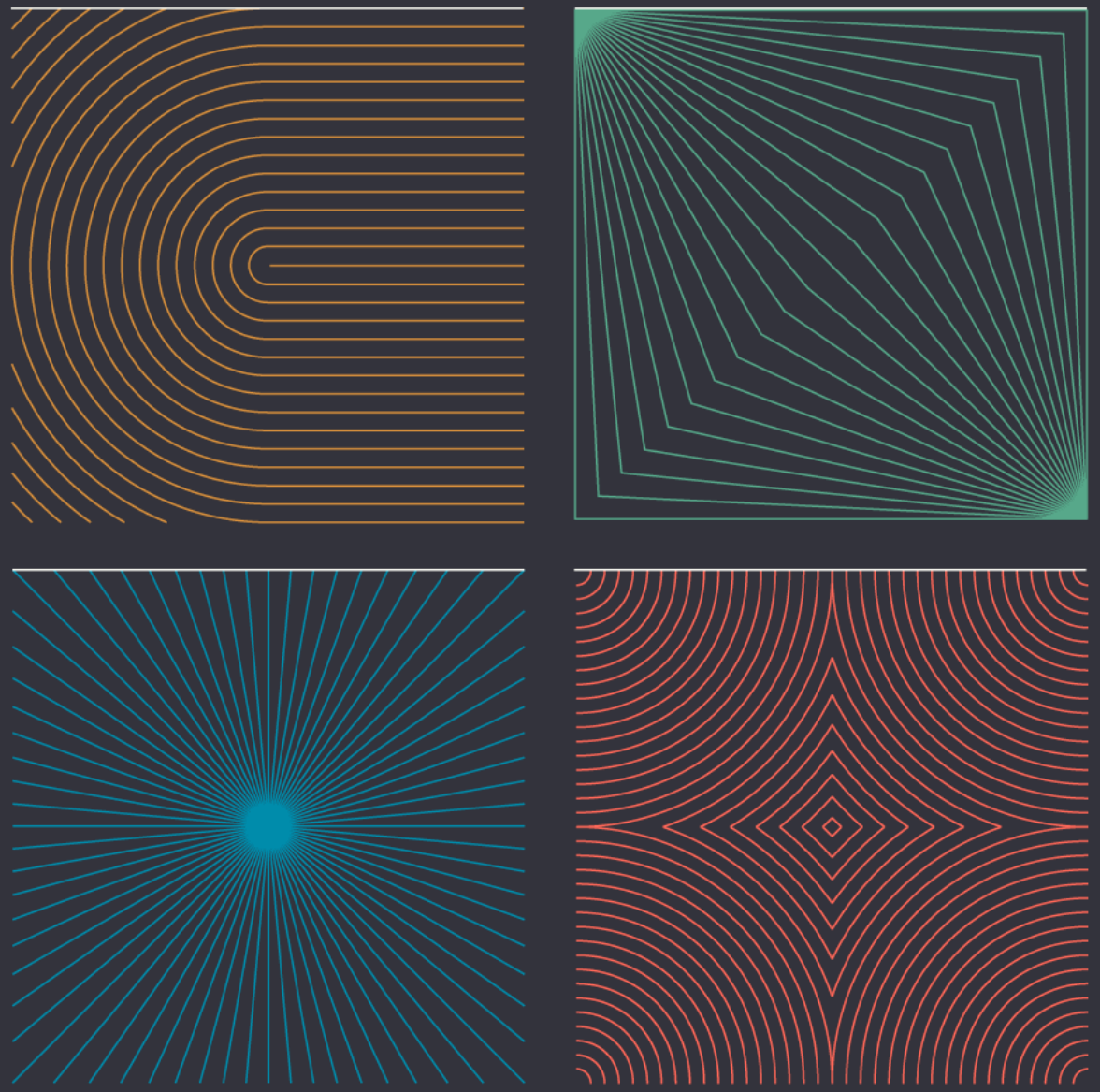
Ocelot2

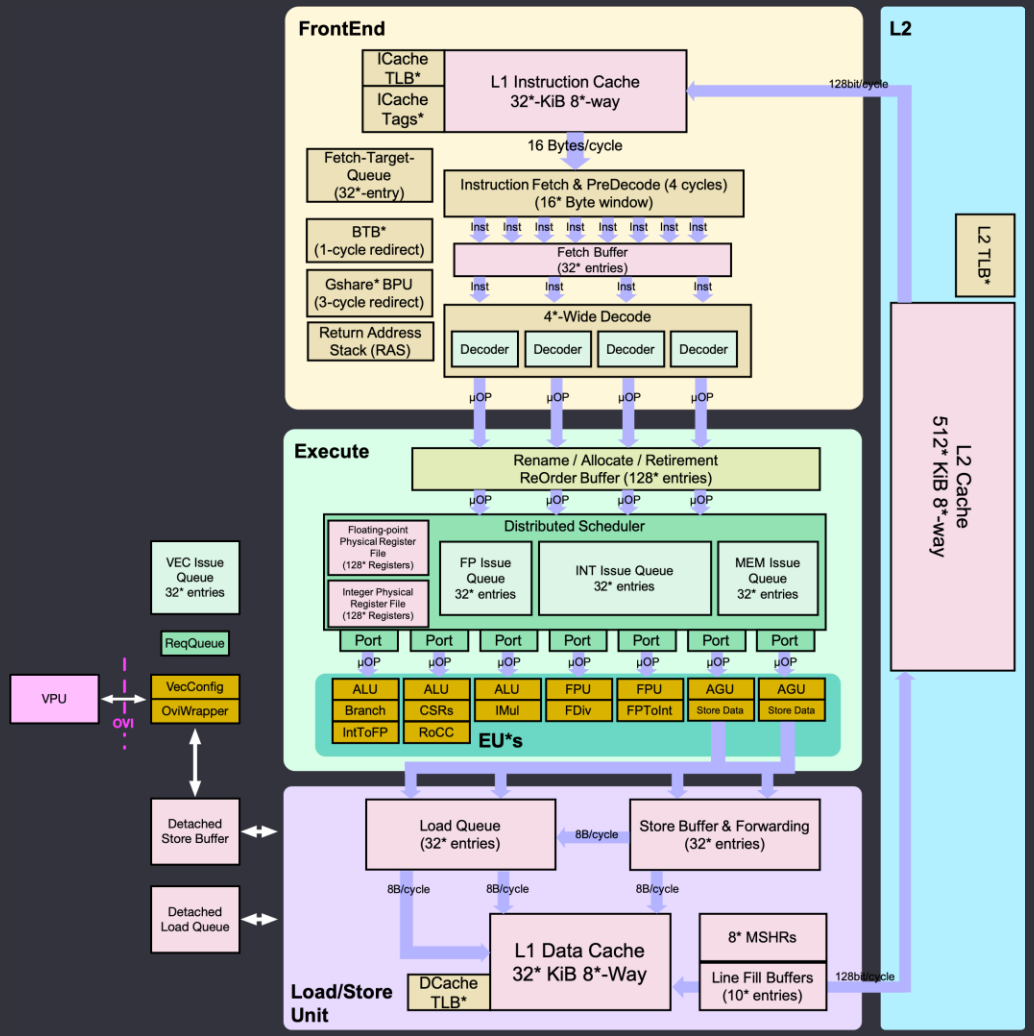
Kuan-Yu Chen

Yavuz Selim Tozlu

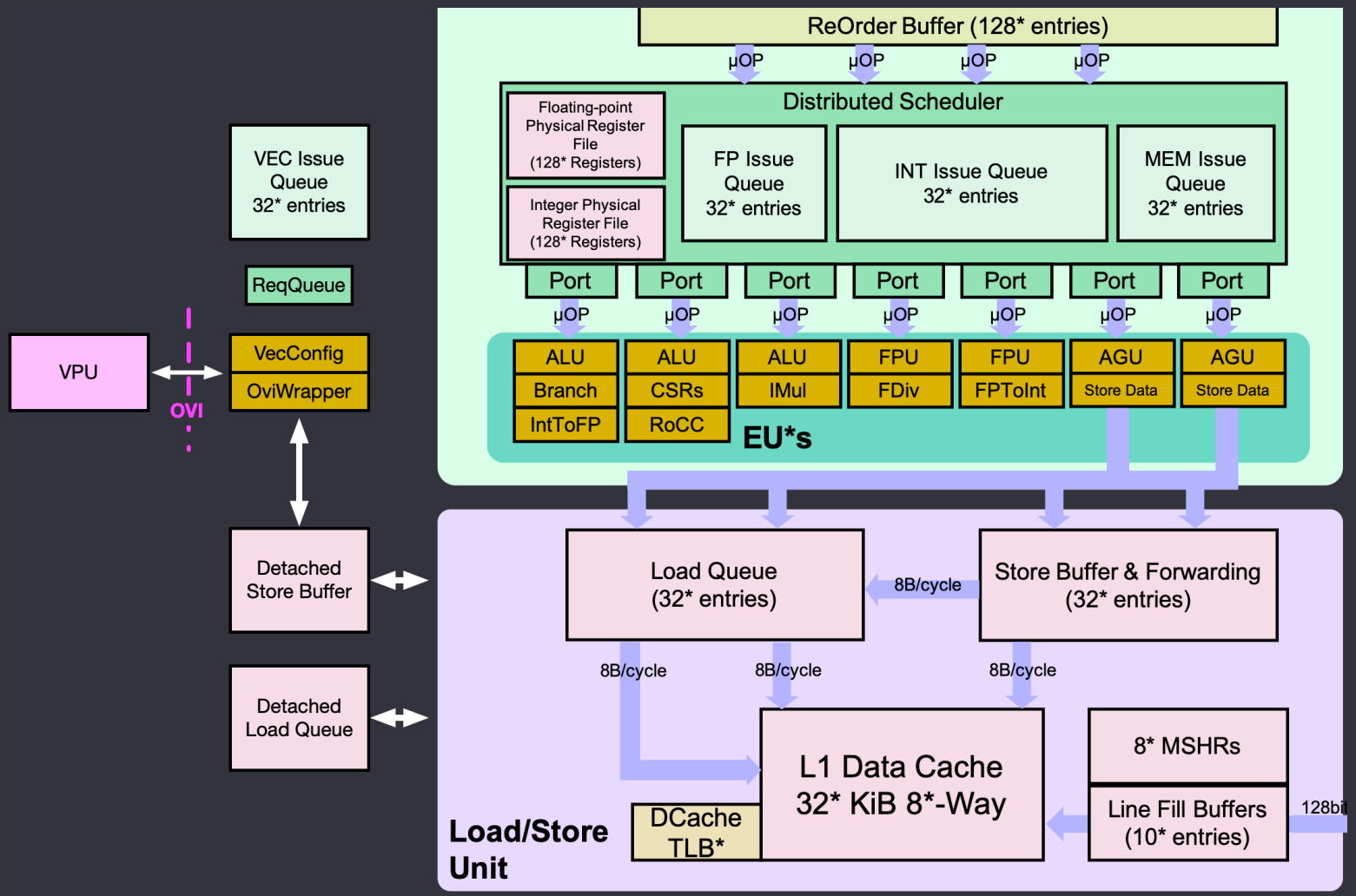


tenstorrent

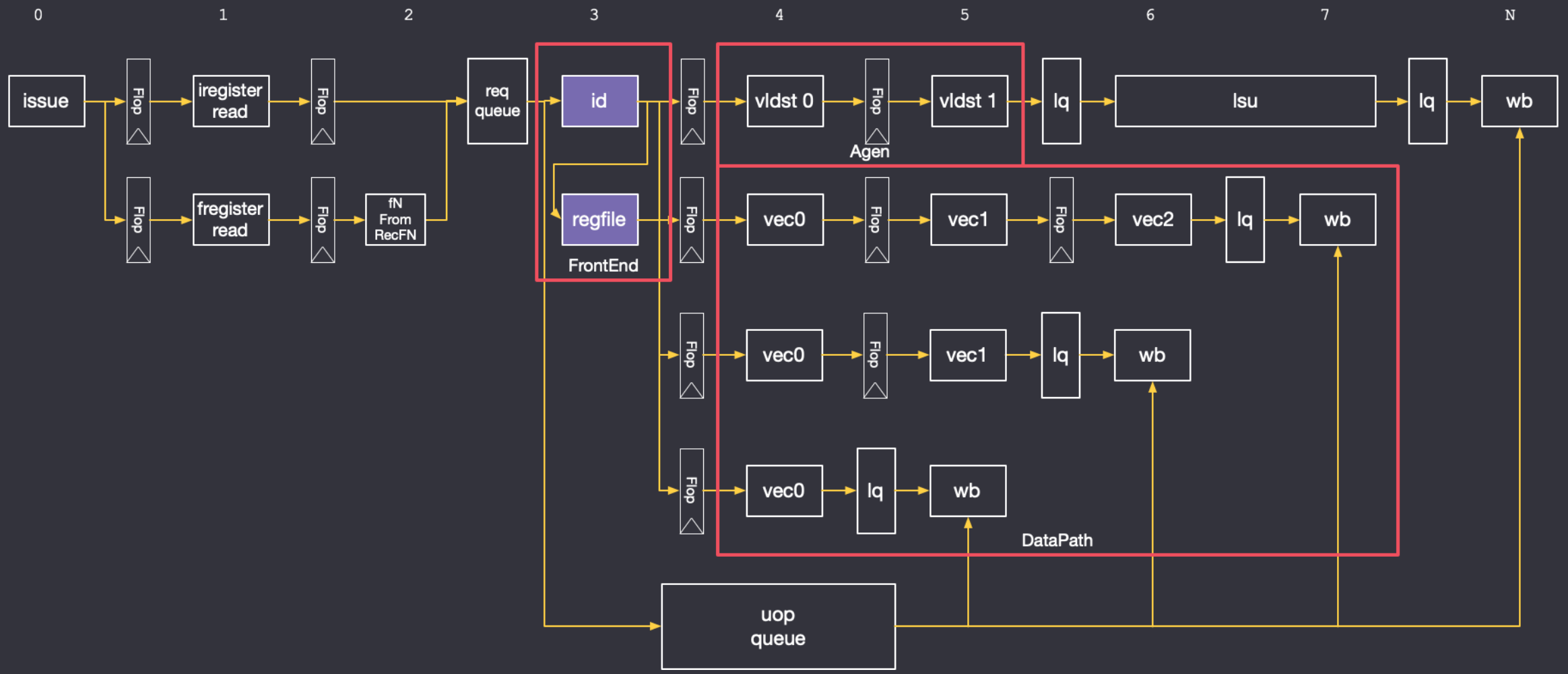
Berkeley Out-of-Order Machine (BOOM)



Ocelot2


ReOrder Buffer (128* entries)
µOP
µOP
µOP
µOP
Distributed Scheduler
Floating-point Physical Register File (128* Registers)
Integer Physical Register File (128* Registers)
FP Issue Queue 32* entries
INT Issue Queue 32* entries
MEM Issue Queue 32* entries
Port
Port
Port
Port
Port
Port
Port
µOP
µOP
µOP
µOP
µOP
µOP
µOP
ALU
Branch
IntToFP
ALU
CSRs
RoCC
ALU
IMul
FPU
FDiv
FPU
FPToInt
AGU
Store Data
AGU
Store Data
EU*s
VEC Issue Queue 32* entries
ReqQueue
VPU
OVI
VecConfig
OviWrapper
Detached Store Buffer
Detached Load Queue
Load Queue (32* entries)
8B/cycle
Store Buffer & Forwarding (32* entries)
8B/cycle
8B/cycle
8B/cycle
L1 Data Cache 32* KiB 8*-Way
8* MSHRs
Line Fill Buffers (10* entries)
128bit
DCache TLB*
Load/Store Unit

Baby RISC-V (Black Hole)



Improvements of Ocelot2

- Improved branch misprediction handling by incorperating Point-of-No-Return mechamism
- Pipelined vset(i)vl(i) to allow unrolling stripmining programs
- Adapted Open Vector Interface
- Redesigned LSU integration, and addressed missing Load/Store hazard protection
- Significantly improved memory performance with support of packing load/store instructions
- Widened Core-L1$ bus to 128b
- (WIP) Out-of-order VPU with register renaming support

Outline

- Background
- OVI
- VPU
- LSU
- Performance
- Takeaway

Open Vector Interface (OVI)

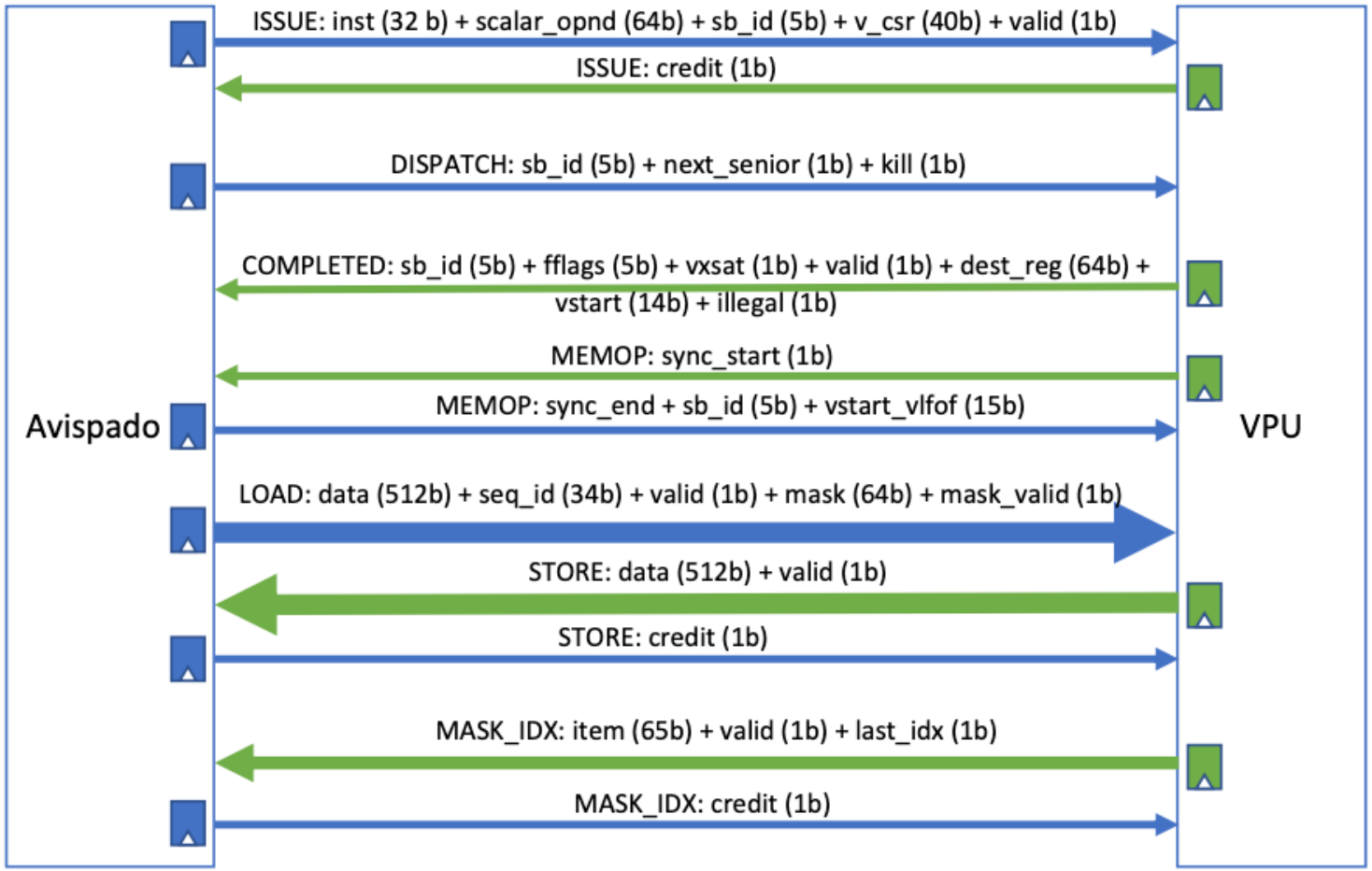


Figure 1. Signals and buses connecting Avispado to the VPU

Compliance Comparison

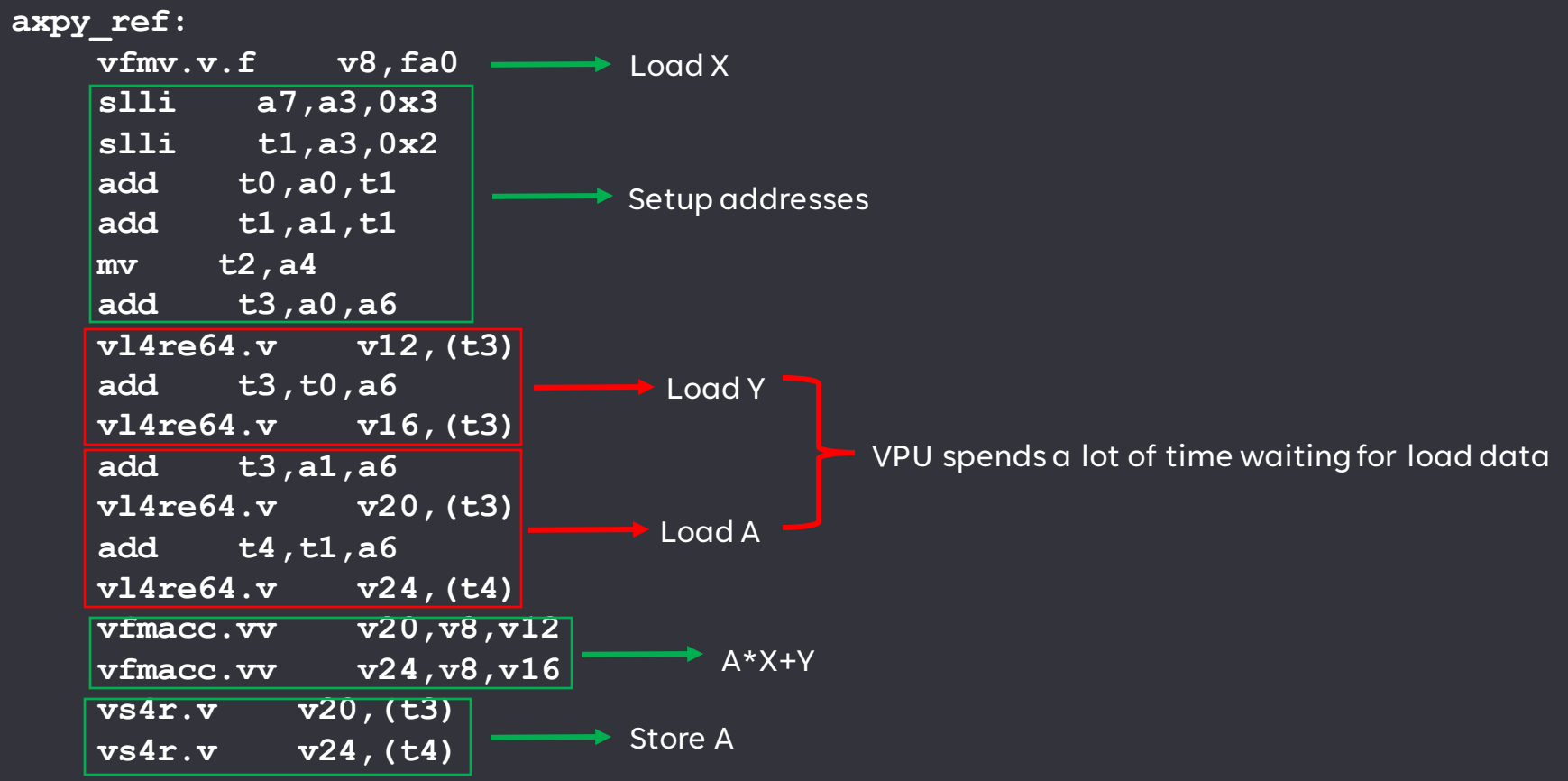
- Deviation from OVI
  - Not support dispatch.kill
    - Every issue is guaranteed to be non-speculative
  - Added support of fractional lmul
  - Zero-ext index offset instead of sign-ext
  - Tied vstart to 0
- Deviation from RVV 1.0
  - Not support segment load/store
  - Not support fdiv/fsqrt/div/sqrt
  - Not support precise trap
    - Assuming every vector load/store won't cause trap

Outline

- Background
- OVI
- VPU
- LSU
- Performance
- Takeaway

Example Vector Program: AXPY

- A*X+Y

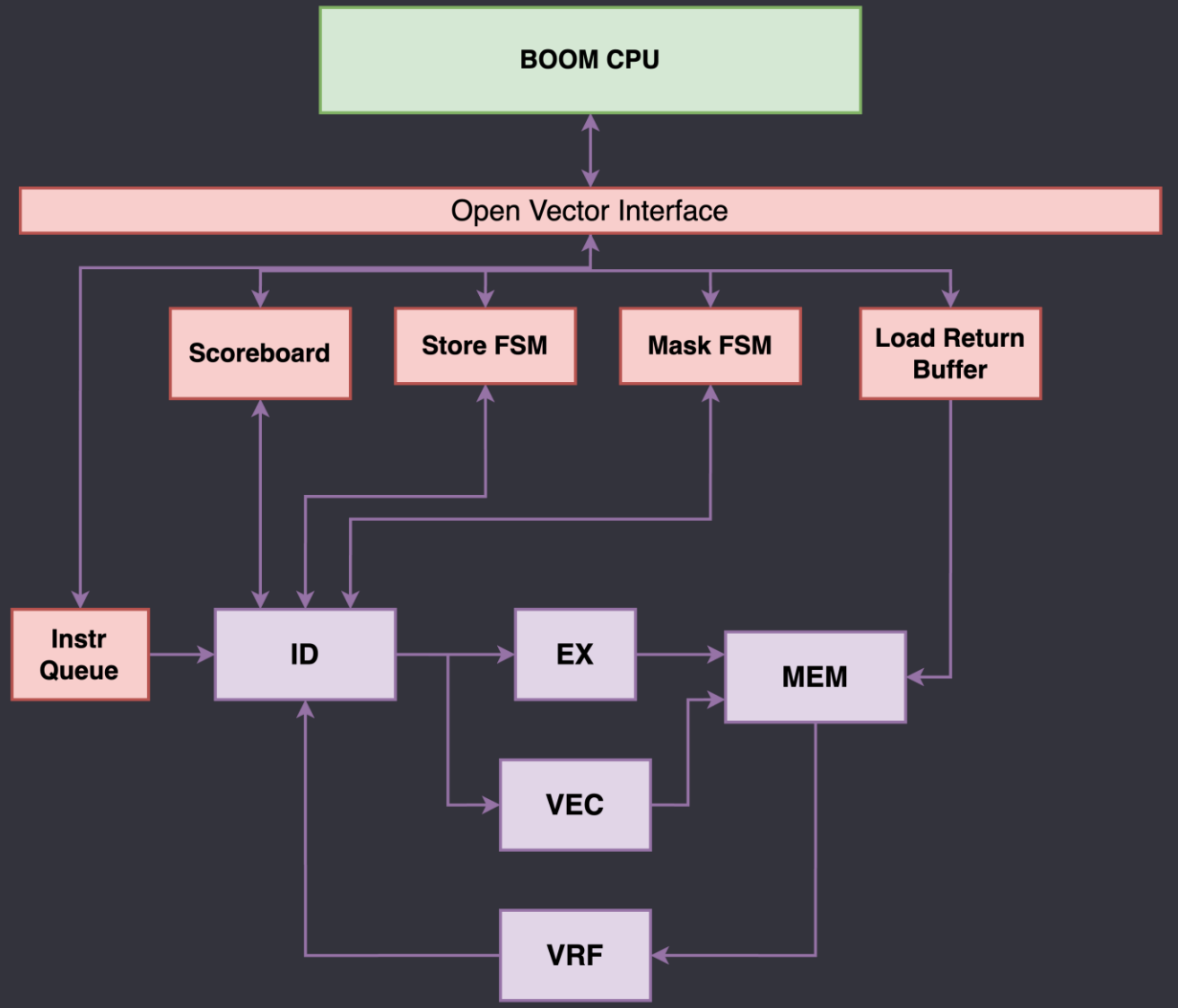


Example Vector Program: Stripmining

```
    # Example: Load 16-bit values, widen multiply to 32b, shift 32b result
    # right by 3, store 32b values.
    # On entry:
    # a0 holds the total number of elements to process
    # a1 holds the address of the source array
    # a2 holds the address of the destination array
loop:
      vsetvli a3, a0, e16, m4, ta, ma # vtype = 16-bit integer vectors. Also update a3 with vl (# of elements this iteration)
      vle16.v v4, (a1) # Get 16b vector
      slli t1, a3, 1 # Multiply # elements this iteration by 2 bytes/source element
      add a1, a1, t1 # Bump pointer
      vwmul.vx v8, v4, x10 # Widening multiply into 32b in <v8--v15>
      vsetvli x0, x0, e32, m8, ta, ma # Operate on 32b values
      vsrl.vi v8, v8, 3
      vse32.v v8, (a2) # Store vector of 32b elements
      slli t1, a3, 2 # Multiply # elements this iteration by 4 bytes/destination element
      add a2, a2, t1 # Bump pointer
      sub a0, a0, a3 # Decrement count by vl
      bnez a0, loop # Any more?
```

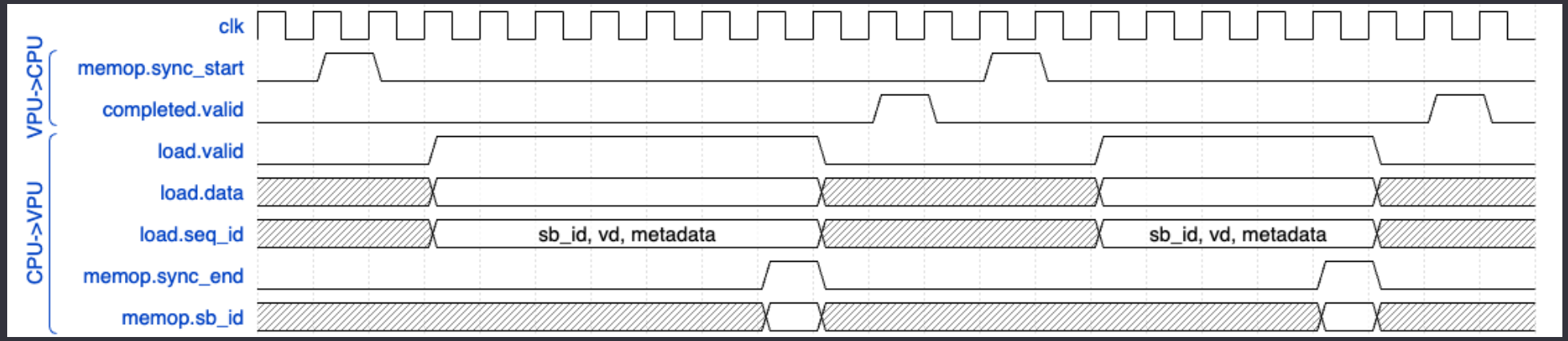
Pipelined Load/Store

- Scoreboard keeps track of all outstanding loads

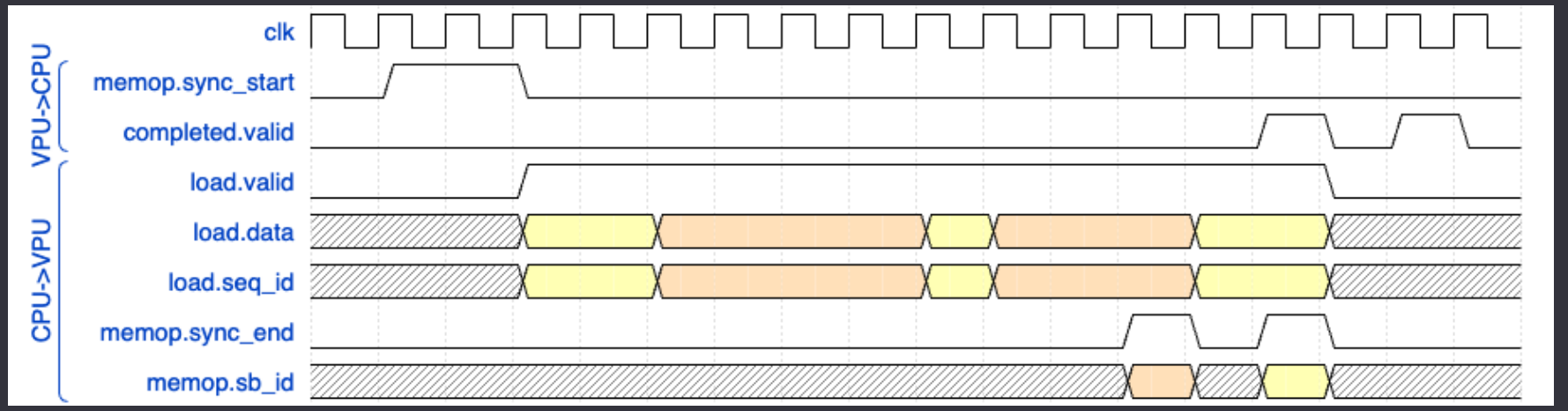


Pipelined Load/Store

- Before pipelined L/S

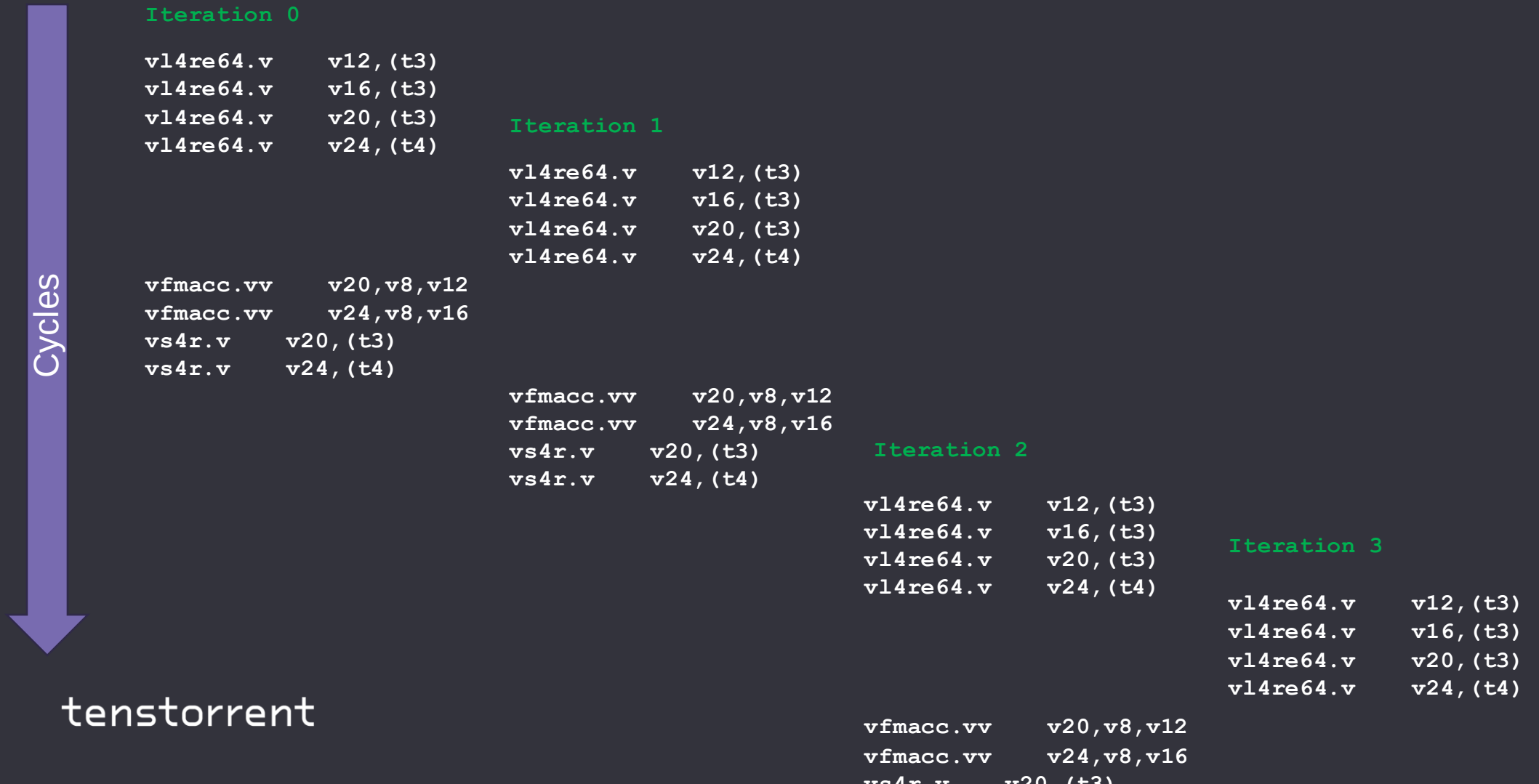


- After pipelined L/S



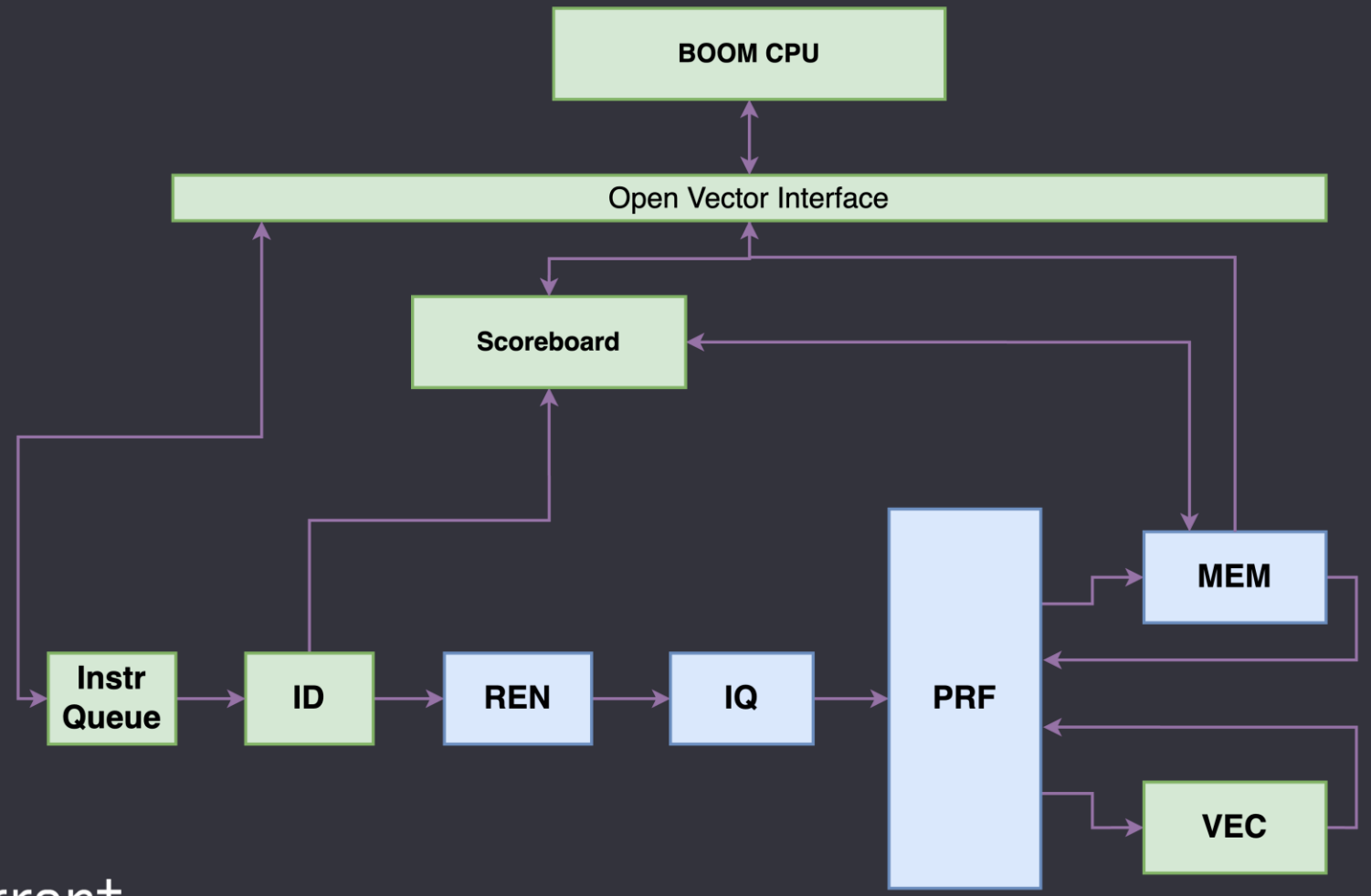
Register Renaming

- Overlap multiple iterations of a loop



Register Renaming & Out-of-Order Execution

- Proposed Design

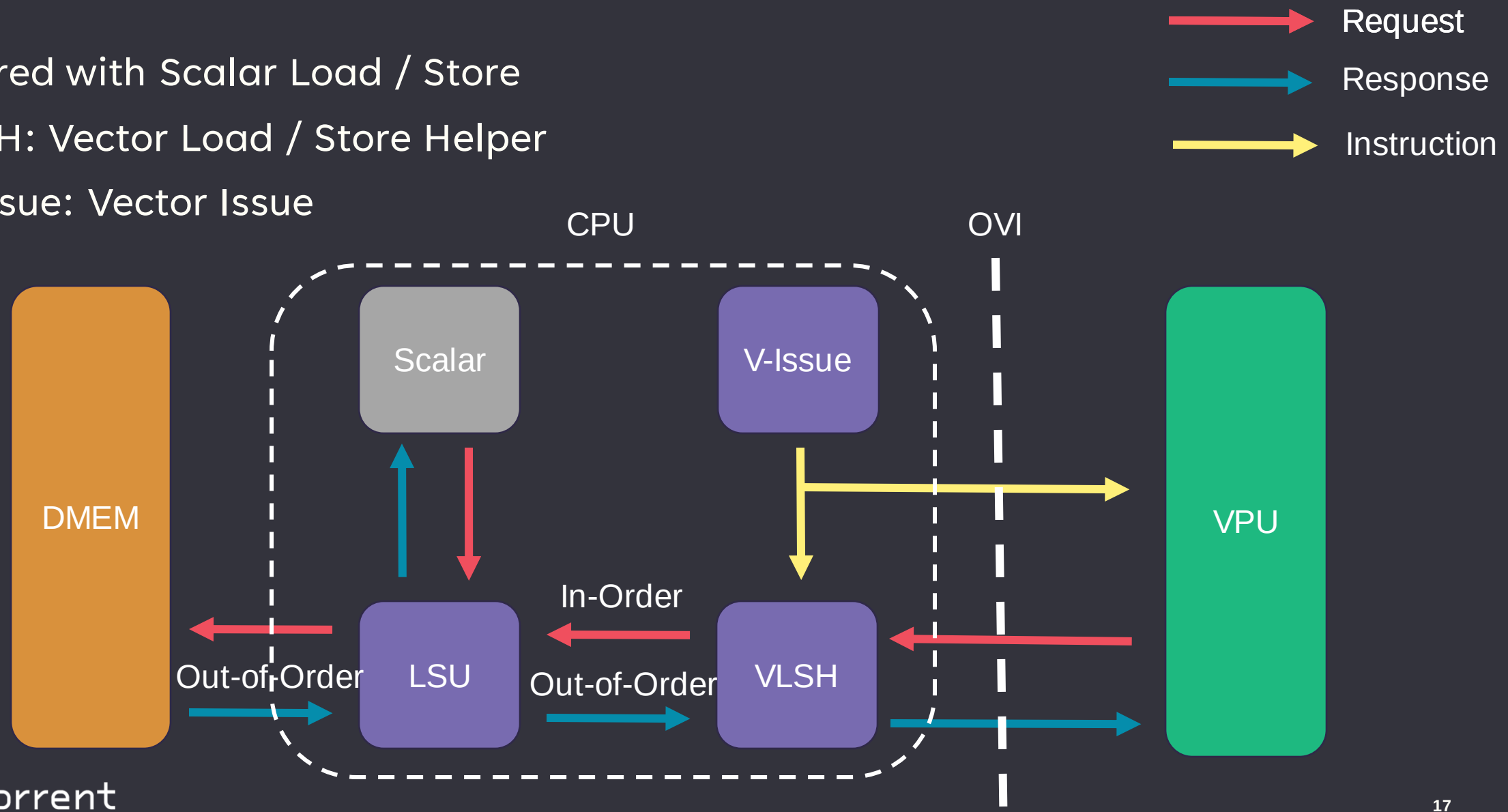


Outline

- Background
- OVI
- VPU
- LSU
- Performance
- Takeaway

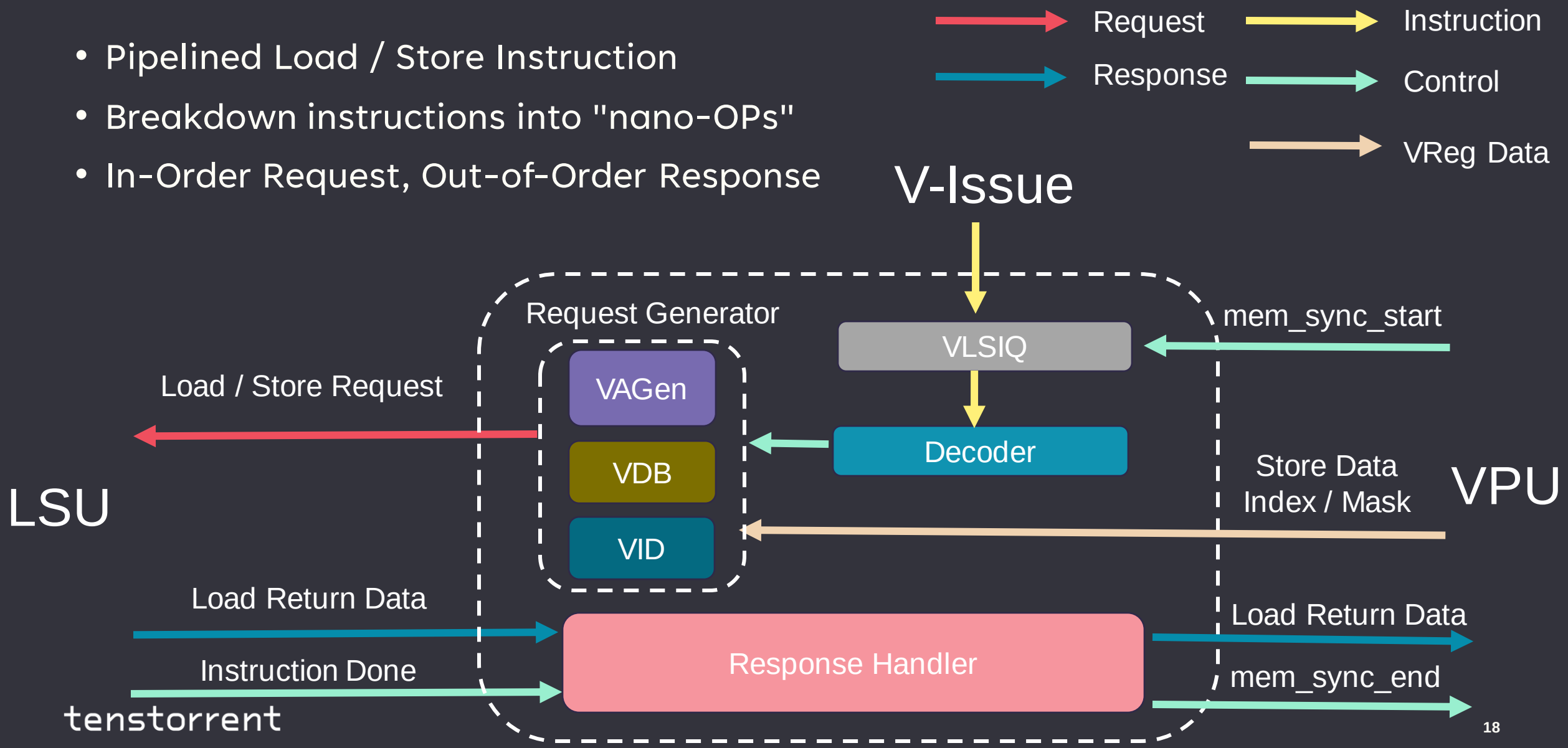
Vector Load / Store with OVI

- Shared with Scalar Load / Store
- VLSH: Vector Load / Store Helper
- V-Issue: Vector Issue



Vector Load / Store Helper (VLSH)

- Pipelined Load / Store Instruction
- Breakdown instructions into "nano-OPs"
- In-Order Request, Out-of-Order Response



Out-of-Order Load Response in OVI

- Sequence ID in Load response
  - enables out-of-order response
    - Sb_id (Scoreboard ID)
    - Element ID
    - V-Reg
  - Put entire memory data (regardless of data width) on data bus (512 bits)
    - Element Count
    - Element Offset
- Optimized for ±1, ±2, ±4 EEW strides: "Good Strides" (including unit stride)

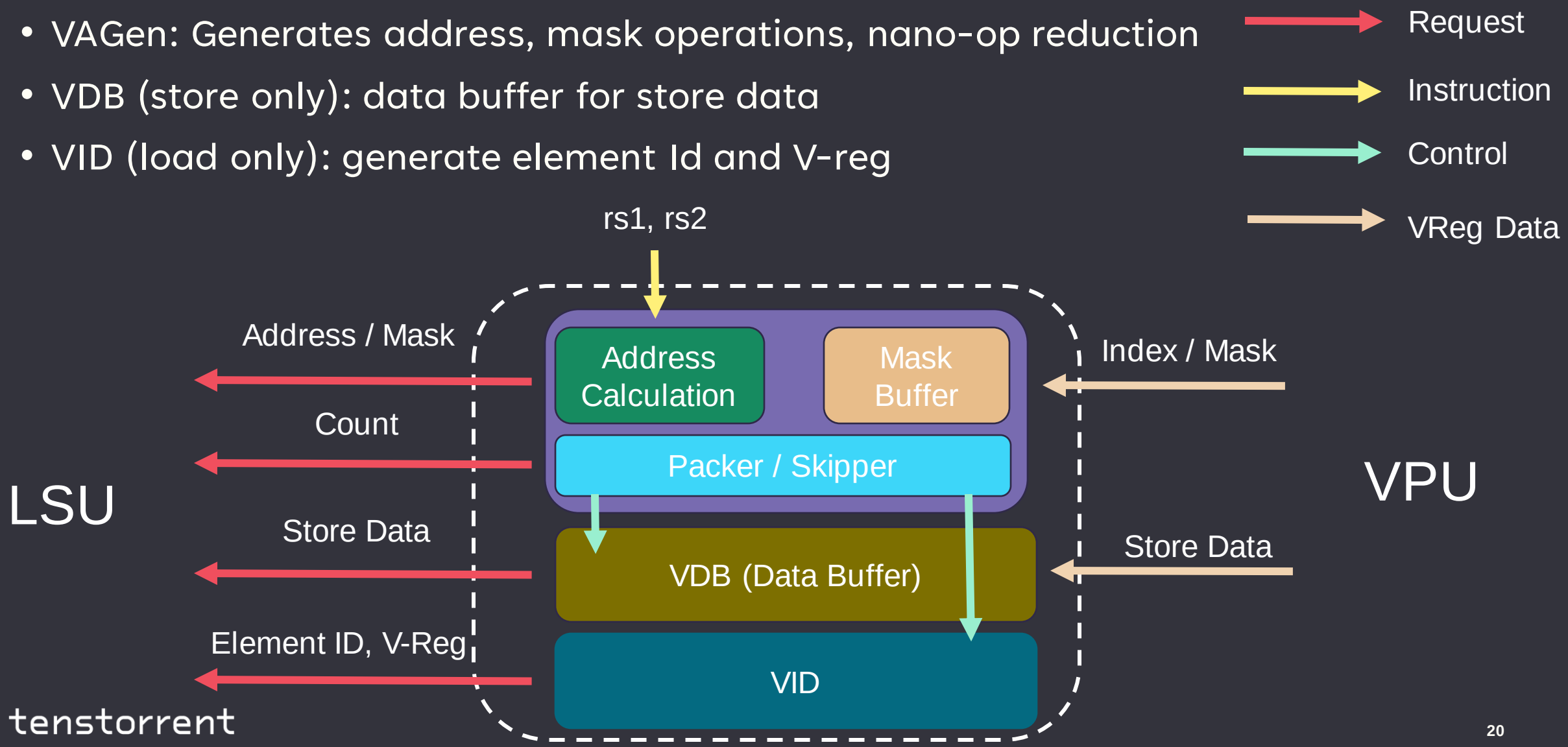
2SEW-Stride (stride 8), v_reg = A, el_id= 16, el_off = 0, el_count=8, sb_id = B

(e.g.: Base@=10240)

ID		23		22		21		20		19		18		17		16
@		184		176		168		160		152		144		136		128

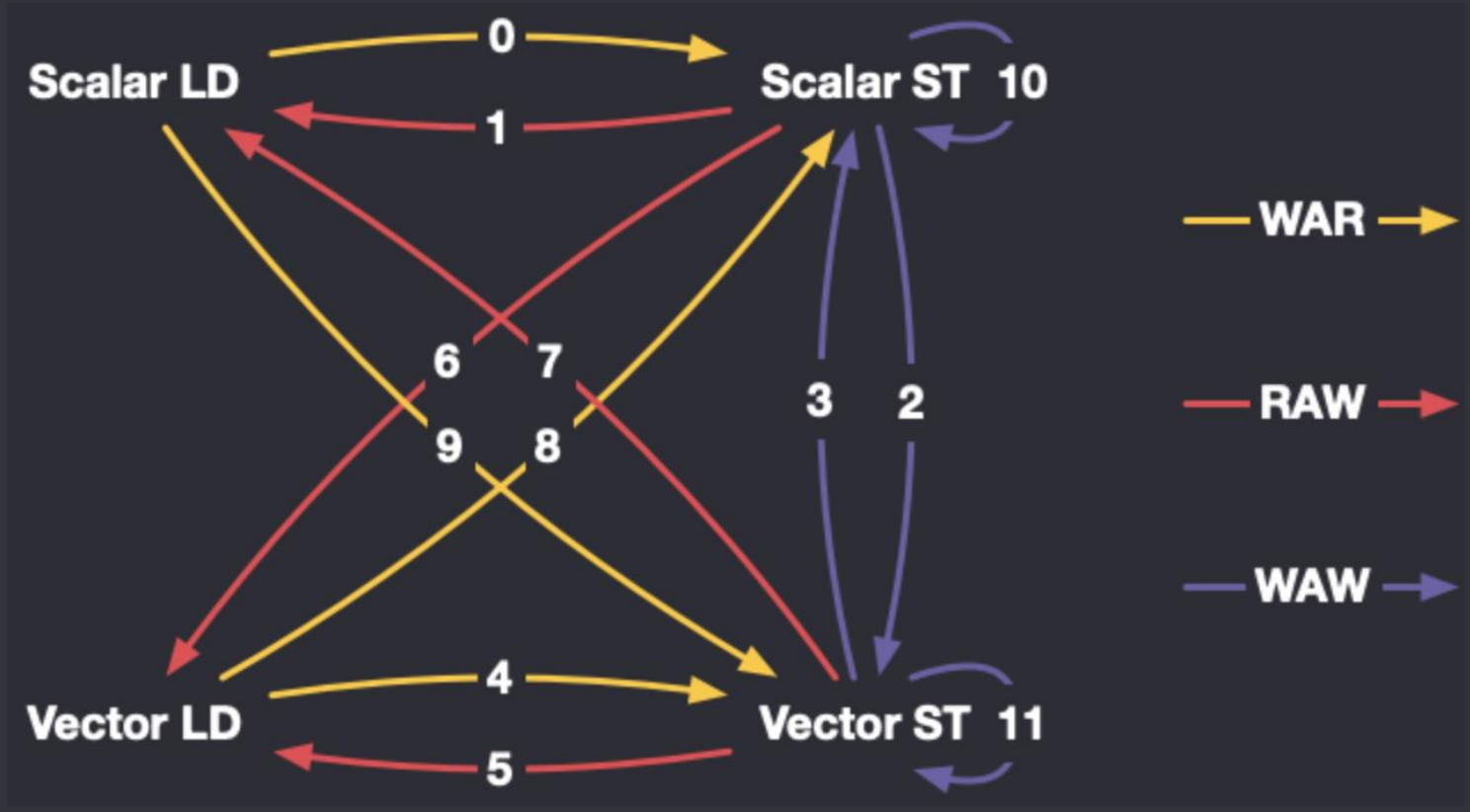
Request Generator

- VAGen: Generates address, mask operations, nano-op reduction
- VDB (store only): data buffer for store data
- VID (load only): generate element Id and V-reg



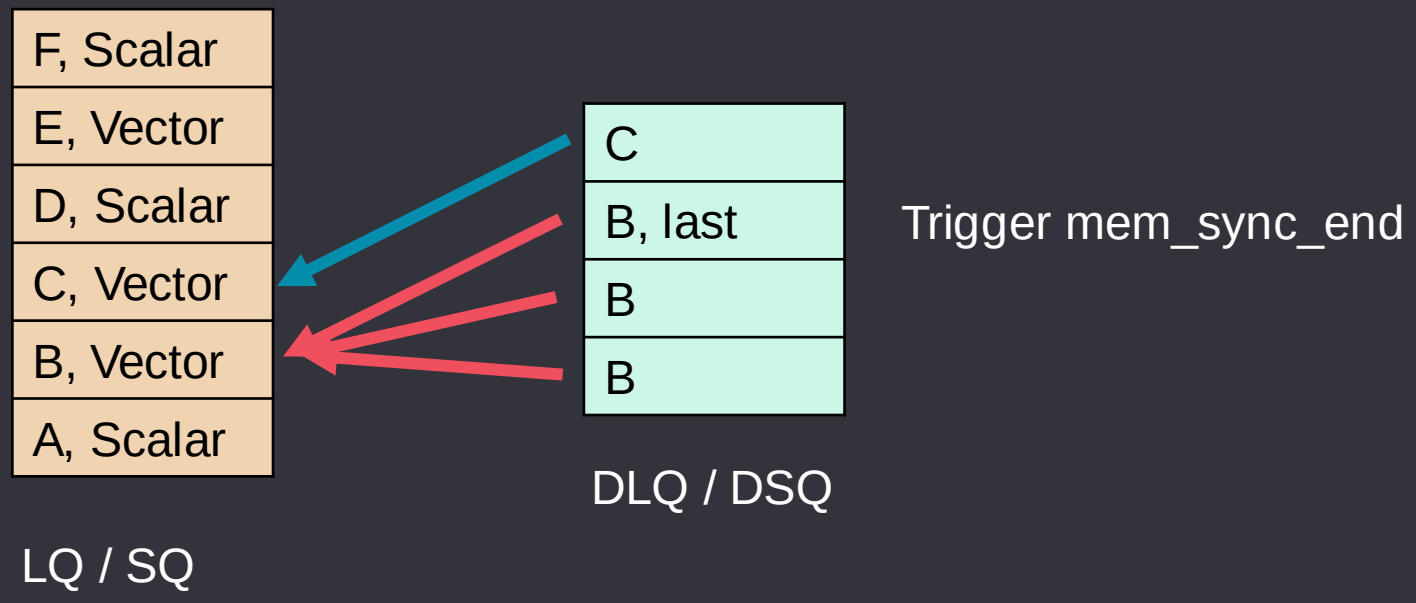
LSU Challenges

- Handles dependencies
- Ensures scalar load / store not affected
- DMEM is oblivious of scalar / vector difference



Detached Queue (DSQ, DLQ)

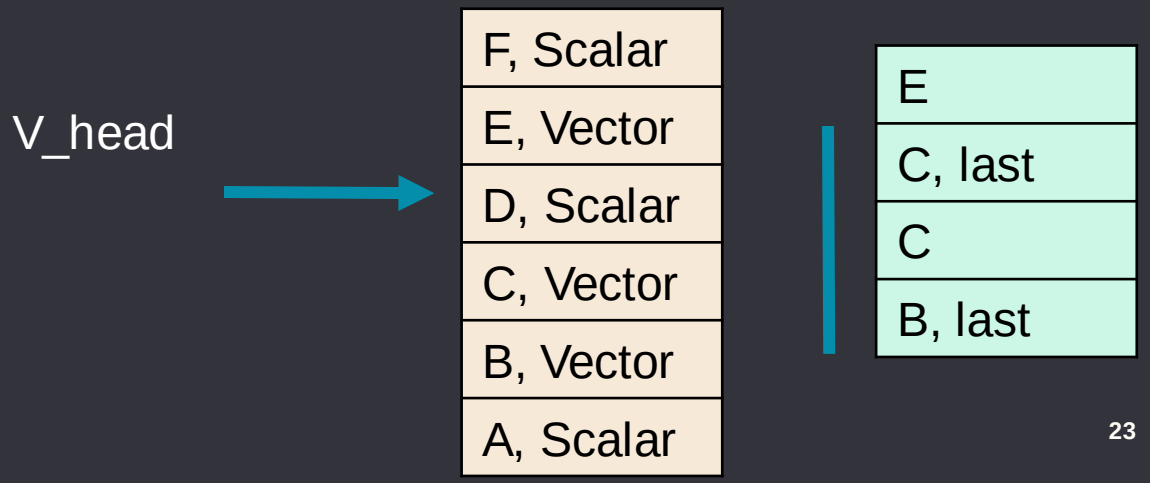
- Scalar Load / Store: 1 Memory Request per Instruction
- Vector Load / Store: Multiple Memory Request per Instruction
  - DMEM Interface <<< Vector Data
- 1 Entry allocated in LQ / SQ at instruction dispatch
  - Detached Queue needed!
  - Stores Queue entry ID of "parent instruction"



Vector Instruction Sprinting

- Conservative Dependency Assumption
  - Assume all scalar load match with older vector load / store
  - Vector can only execute when all older scalar are done

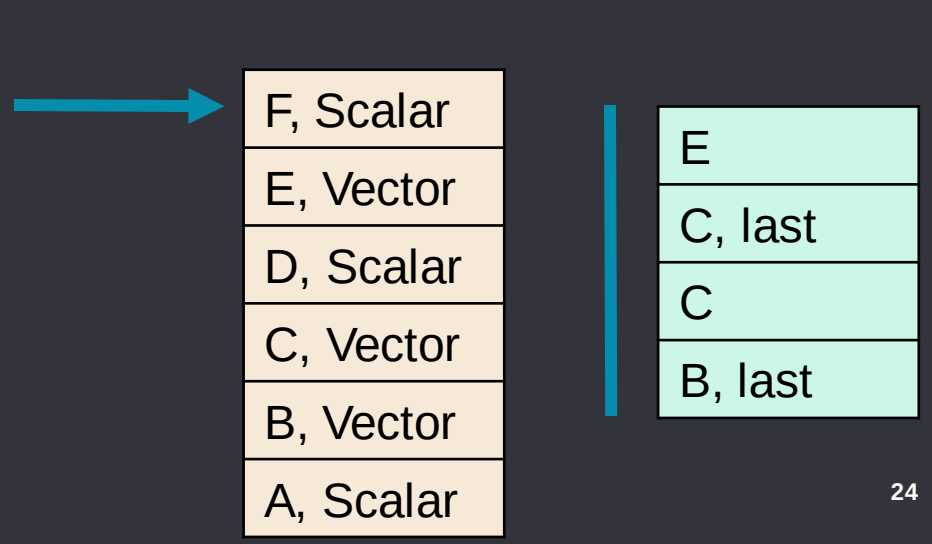
- V_head Pointer: Every vector entry that is older can execute
  - Eliminate latency from VPU complete to ROB commit
  - Enable Back-to-Back vector instruction execution
  - Increment Condition:
    - Scalar that has succeeded
    - Vector that has no older scalar



Vector Instruction Sprinting

- Conservative Dependency Assumption
  - Assume all scalar load match with older vector load / store
  - Vector can only execute when all older scalar are done

- V_head Pointer: Every vector entry that is older can execute
  - Eliminate latency from VPU complete to ROB commit
  - Enable Back-to-Back vector instruction execution
  - Increment Condition:
    - Scalar that has succeeded
    - Vector that has no older scalar



Optimization

- Goal:
  - Reduce number of nano-op
  - Reduce latency for generating requests of each instruction
- Packing: Multiple elements in one nano-op
  - Reduce number of nano-op
- Widening: Widen the interface between LSU and DMEM
  - Increase max number of elements in nano-op
- Skipping: Skip multiple masked off elements in one cycle
  - Decrease number of bubbles

Optimization Example

- DMEM Width = 64 bits

- Example 1: Strided load: stride = 2*EEW, vl = 32, EEW = 16 bits, vstart = 0
  - Original: Number of nano-op: 32
  - After Packing: Number of nano-op: 16, each nano-op contains 2 elements
  - After Widening to 128 bits: Number of nano-op: 8, each nano-op contains 4 elements
  - Nano-op reduction: 4X, if unit-stride: 8X

- Example 2: Masked unit stride store: vl = 6, vstart = 0, mask = 100001
  - Original: Number of nano-op: 2, 6 cycles to finish sending request
  - After Skipping: Number of nano-op: 2, 3 cycles to finish sending request
  - Cycle reduction: 2 X

Summary of Optimization

Load / Store	Address Scheme	Mask	Optimization
Load	Good Stride	Unmasked	Packing + Widening
	Bad Stride		N/A
	Index		N/A
	Good Stride	Maked	Packing + Widening
	Bad Stride		Skipping
	Index		N/A
Store	Unit Stride	Unmasked	Packing + Widening
	Other Stride		N/A
	Index		N/A
	Unit Stride	Masked	Skipping
	Other Stride		Skipping
	Index		N/A

Different Optimization Strategy for Load / Store

- For masked / strided loads: overloading is fine
  - Use Seq_id to determine the "useful" elements
  - Need to take care of alignment

- Stores: Cannot "over-store", no "store mask" in BOOM dmem
  - Stricter requirements on alignment
  - Break things down into power of 2 (1, 2, 4, 8...) elements

Outline

- Background
- OVI
- VPU
- LSU
- Performance
- Takeaway

Different Configurations of Bobcat

	Large-Ocelot	Small-Bobcat	Medium-Bobcat	Large-Bobcat	Mega-Bobcat
Issue Width	3	1	2	3	4
Memory Pipe	1	1	1	1	2
DMEM Bandwidth	64	64	64	128	128

Bobcat v.s. Ocelot

test name	Large-Ocelot (cycles)	Large-Bobcat (cycles)	Speedup
arith-mean-auto-vector	1333	489	273%
arith-mean-unroll-vector	1654	984	168%
arith-mean-vector	1830	995	184%
axpy-auto-vector	49217	4294	1146%
axpy-vector	12907	4207	307%
conv1d-vector	30751	14337	214%
conv2d-vector	123106	56875	216%
inner-prod-auto-vector	4534	1420	319%
inner-prod-unroll-vector	5893	2894	204%
inner-prod-vector	6570	2892	227%
relu-auto-vector	11862	1075	1103%
relu-unroll-vector	4370	1206	362%
relu-vector	4695	1187	396%
sgemm64	4623	1958	236%
geometry mean			310%

Observations

- Bobcat is more adaptable to different compiling schemes

Outline

- Background
- OVI
- VPU
- LSU
- Performance
- Takeaway

Takeaway

- AI-aid design
- CHISEL
- Benefit of having clean interface