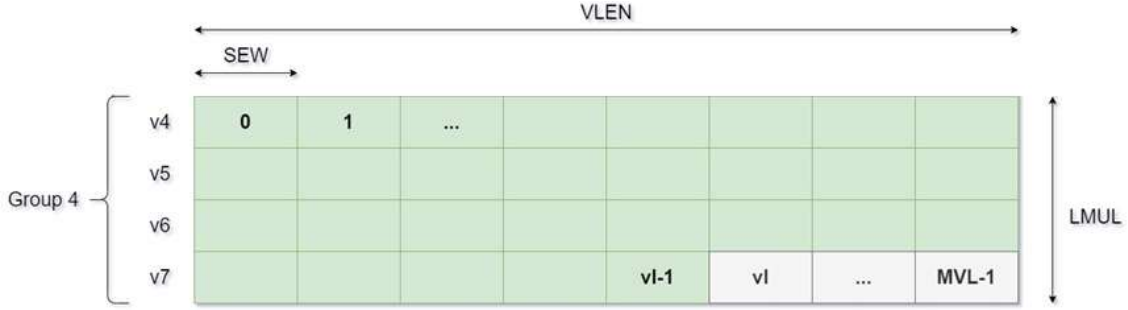
A Pluggable Vector Unit for an Open-source 64-bit RISC-V Implementation

Vincenzo Maisto
Università degli Studi di Napoli Federico II, Italy, and Hensoldt Cyber GmbH, Germany
Alessandro Cilardo
Università degli Studi di Napoli Federico II, Italy

The RISC-V Vector 'V' Extension

- Introduces 32 vector registers
- Adds 7 new Control and Status Registers (CSRs)
- Implementation-defined parameters
  - VLEN : bits per vector register
  - ELEN : maximum width per element
- Run-time hardware reconfiguration
  - SEW : selected element width
  - VL : number of elements in a vector
  - LMUL : number of grouped vector registers
- It derives a Maximum Vector Length $MVL = LMUL\frac{VLEN}{SEW}$
- Vector Length Agnostic programming model

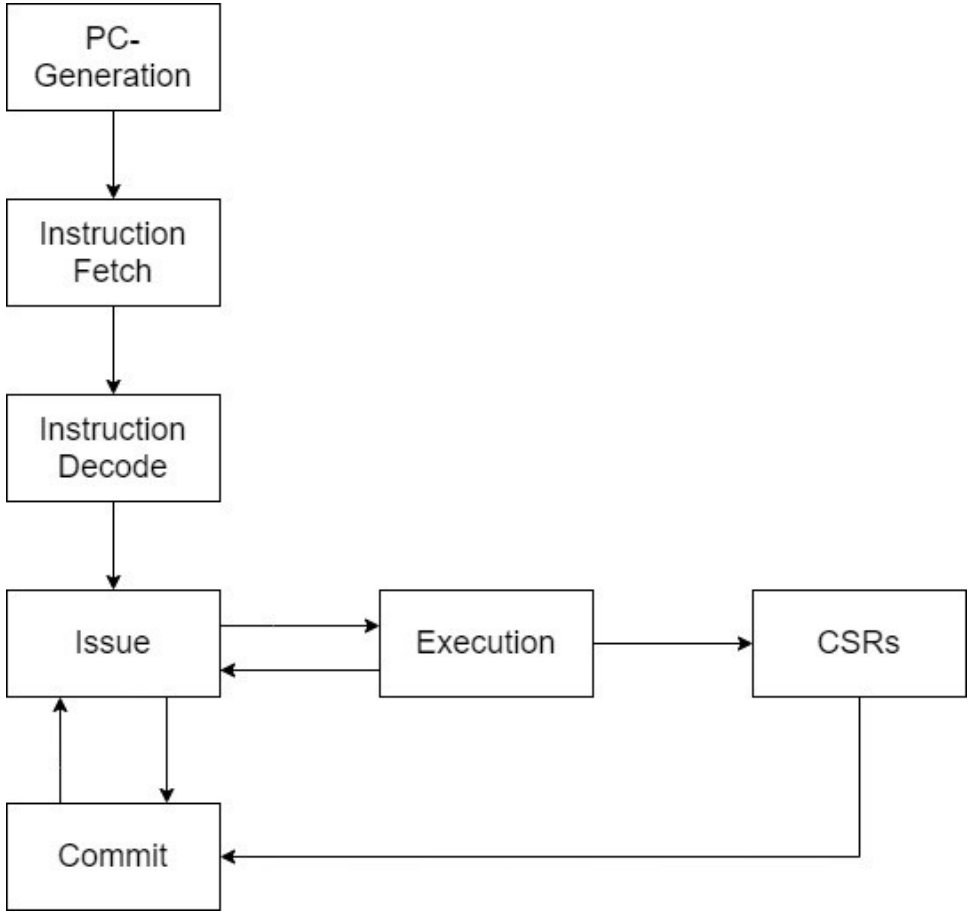


Extending a Base Micro-Architecture

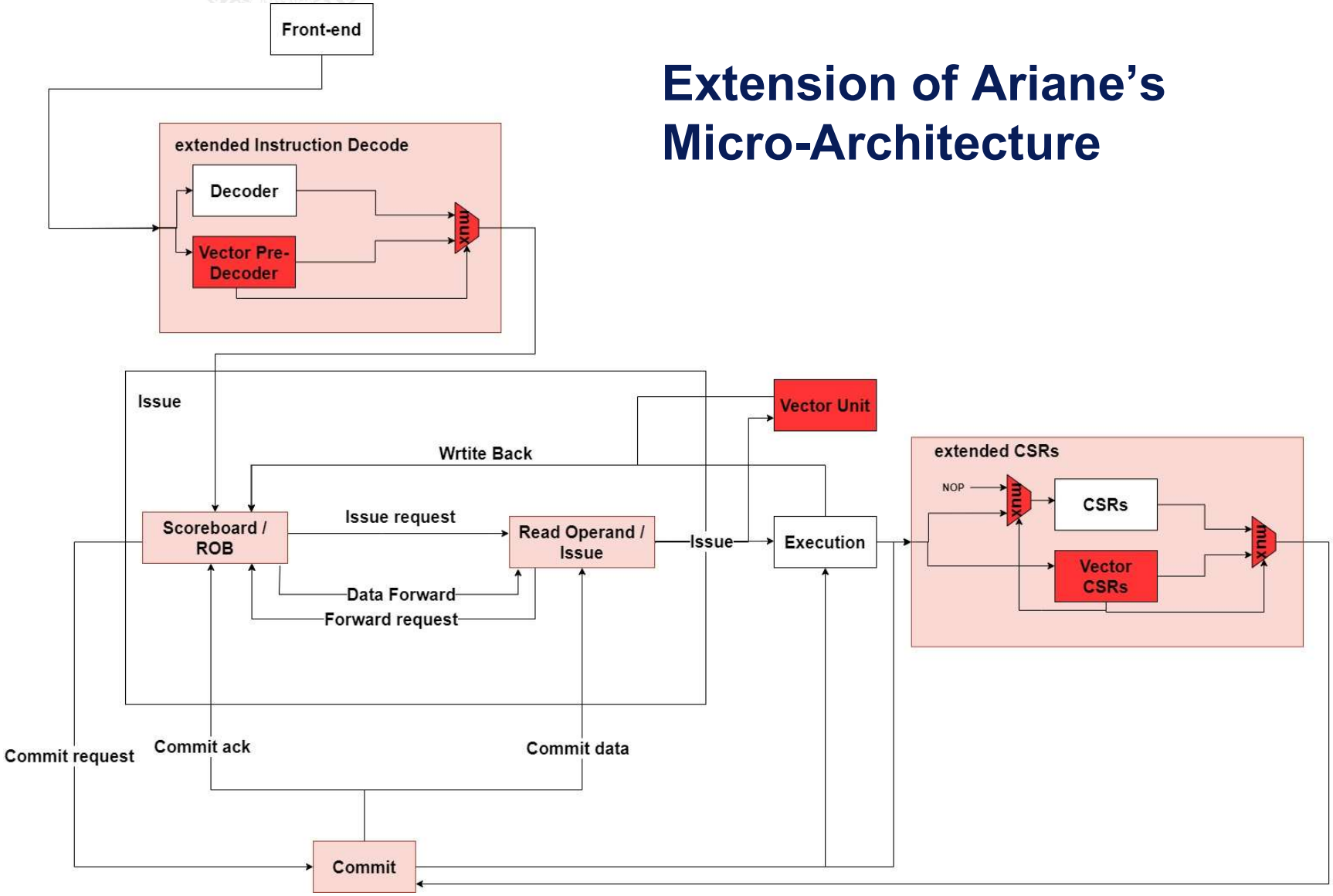
- Minimize
  - source code impact
  - micro-architectural impact
  - dependencies between the Vector Unit and the rest of the (micro)architecture
- Required features in the base micro-architecture:
  - Decode Vector instruction dependencies with the base architecture
  - Distinguish among Vector, fp and gp registers
  - No register renaming for Vector registers
  - Forward scalar operands from/to Vector instructions to/from fp and gp registers
  - Introduce new Vector CSRs
  - Induce RAW between Vector CSRs updates and Vector instructions
  - Update Vector CSR for each retiring Vector instruction
  - Simplification, disallow double commit of Vector instructions
  - Simplification, disallow CSR operation commit together with Vector instructions
  - Allow write back from the Vector Unit
  - Stall the Vector Unit in case of Vector Configuration instructions

The Ariane Core as an industrial use case

- Base RISC-V 64-bit open-source implementation
- Used by Hensoldt Cyber GmbH for its products
- Sv39 Virtual Memory
- In-order issue
  - Merged with Read-Operands
- Out-of-order write back
- In-order commit
- Originally from ETH Zurich
  - Currently maintained by the **Open Hardware Group** with the new name **CVA6**

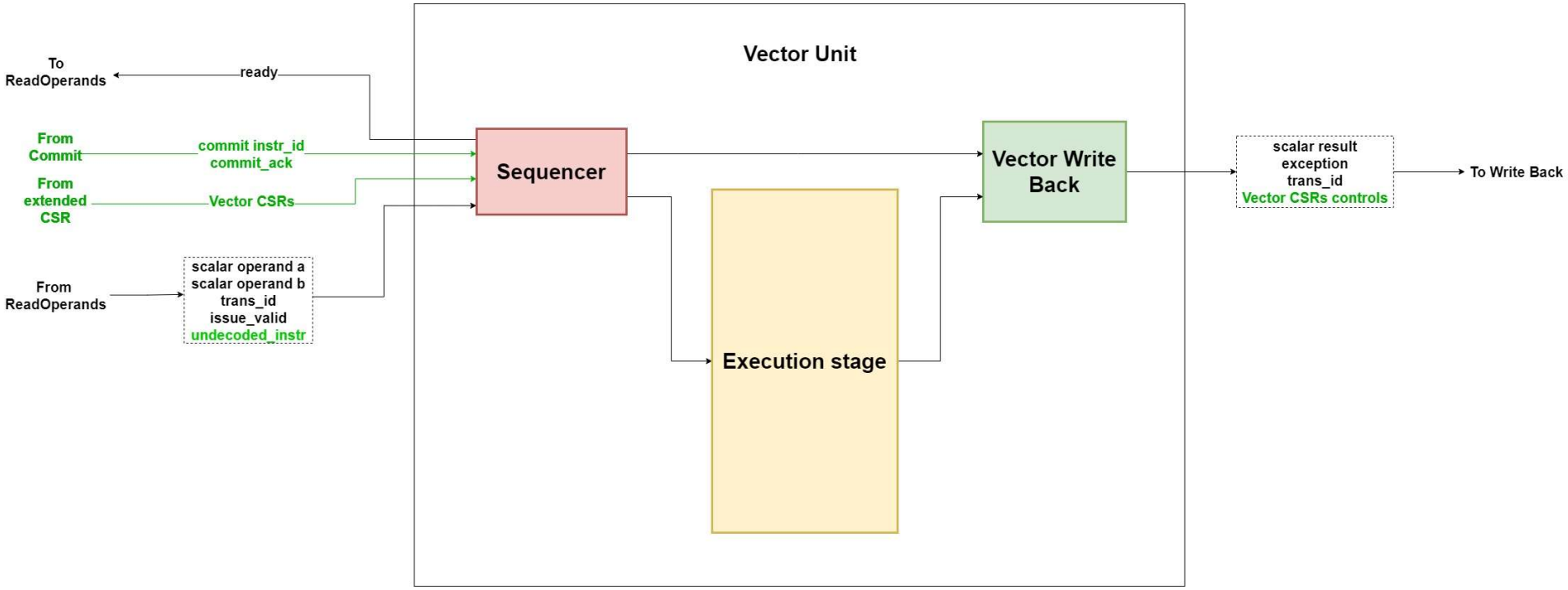


Extension of Ariane's Micro-Architecture



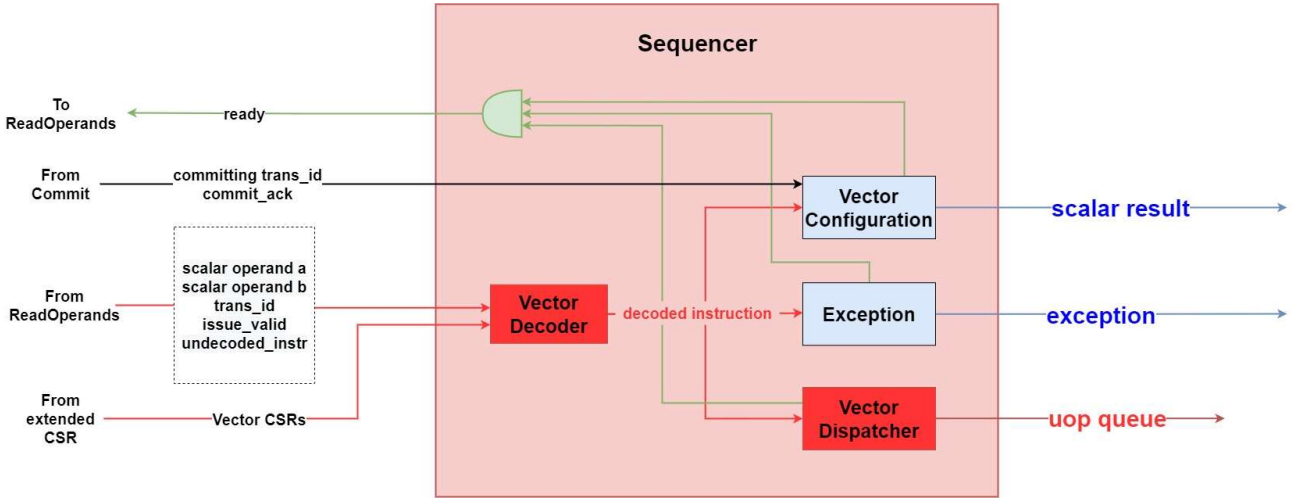
Vector Unit Micro-Architecture

- It integrates as a regular functional unit
  - With some other special ports
  - Do not rely on Ariane's (micro)architectural features
- 3 logical stages pipeline



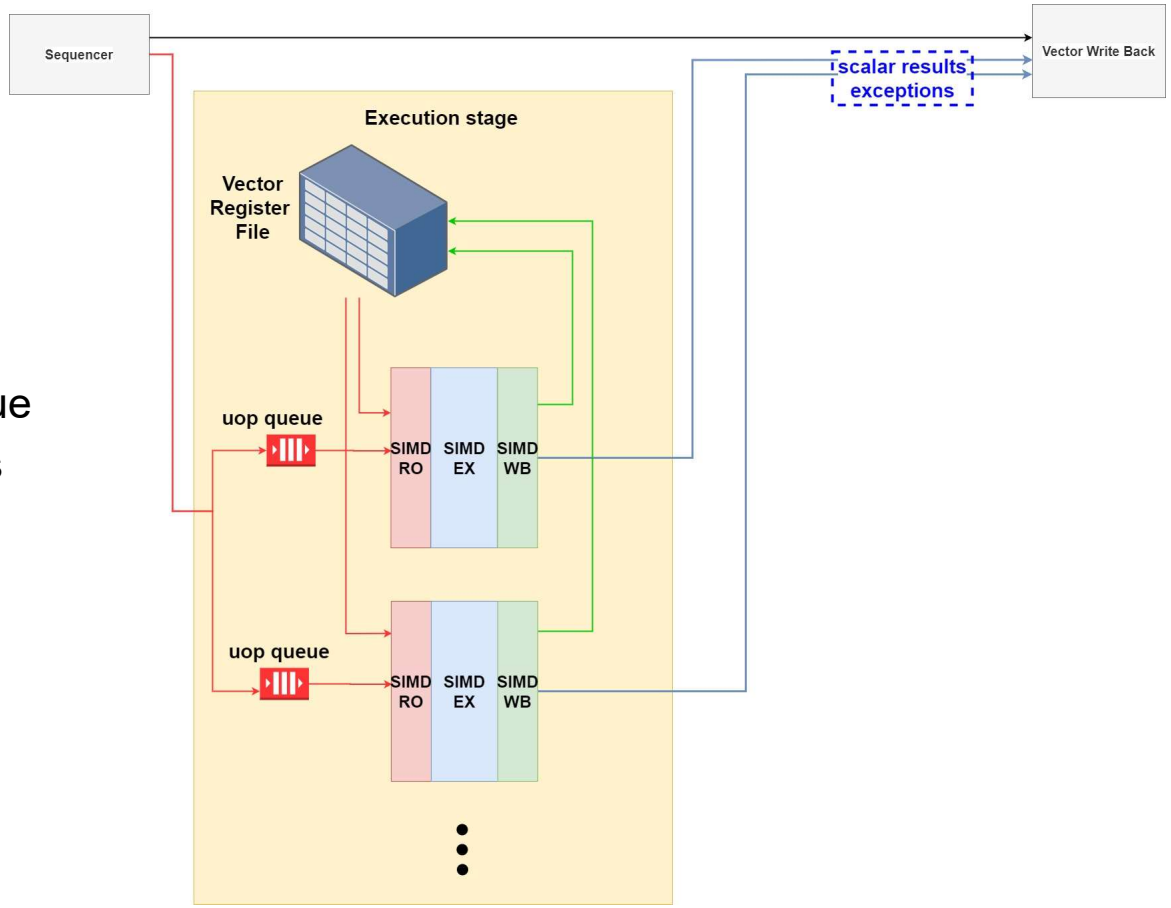
Sequencer

- **Vector Decoder** : full decoding of the raw instruction
- **Exception** : handles the exceptions generated by the Vector Decoder
- **Vector Configuration** :
  - performs the strip-mining
  - Stalls the Vector Unit until its instructions retire *(or just write-back)*
- **Vector Dispatcher** :
  - Sequences Vector instructions into LMUL Packed-SIMD μ-operations
  - Dispatches the instruction on the target SIMD functional unit



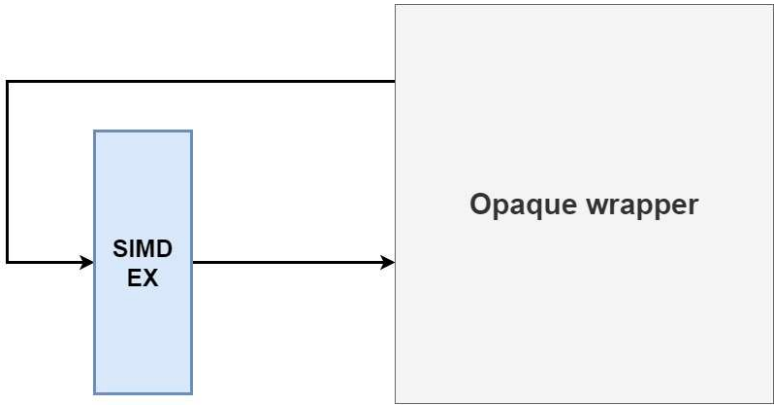
Execution stage

- Regular structure
- Easily extensible
- SIMD functional units, composed by:
  - Micro-operations queue
  - SIMD Read Operands
  - *SIMD Execute (Specialized for functionality)*
  - SIMD Write back
- Vector Register File



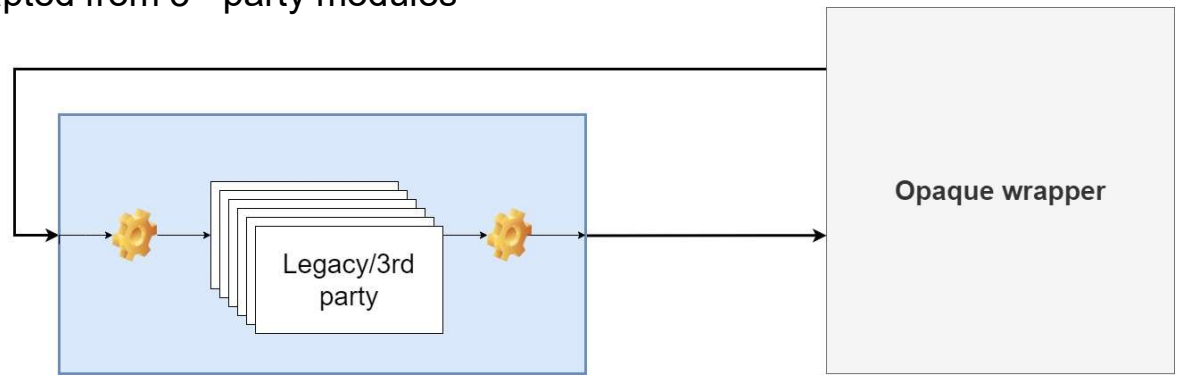
Integration of Functional Units

- Functional Units can be integrated in the design without any knowledge of the micro-architectural details
  - E.g., operand preparation, scalar/vector write back, instruction sequencing
- Requirements
  - Packed-SIMD capabilities
  - Support for variable SEW and VLEN
- Required interface
  - Input SIMD and scalar operands
  - Input SEW
  - Input and output ready/valid signals
  - Output SIMD and scalar result
  - Output exception signaling
  - Some of these ports might also be not implemented
- Low coupling
  - FU only interacts only with an opaque wrapper
  - All the complexity is hidden



Integration of Functional Units

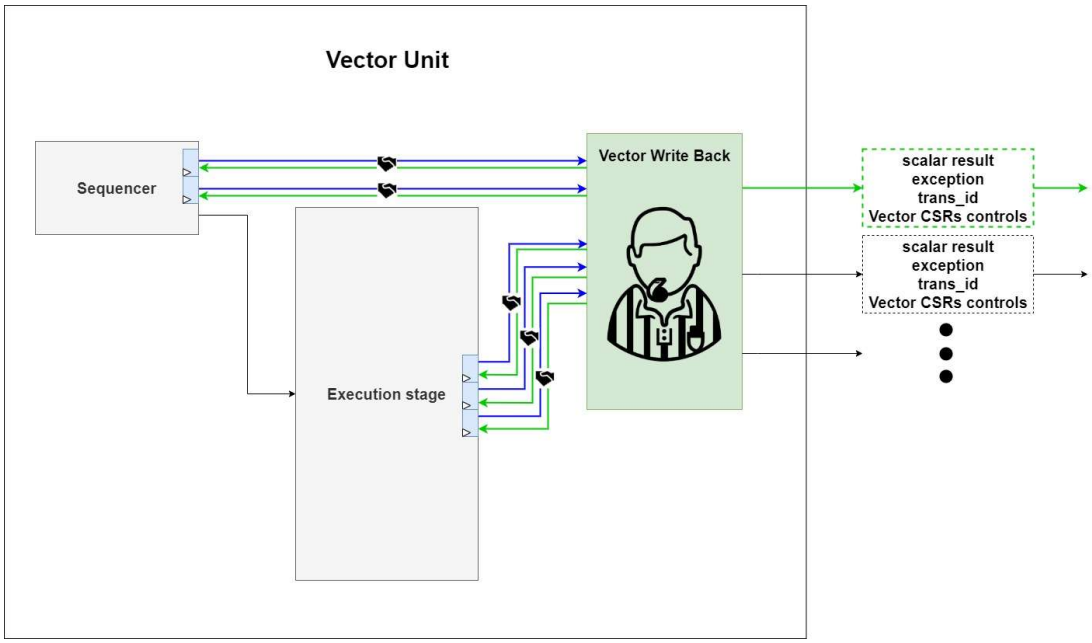
- Design freedom
  - FU can be fully combinatorial, pipelined, have some sequential logic, …
- SIMD functional units can be
  - Designed from scratch
  - Adapted from legacy components
  - Adapted from 3rd party modules



- Together with the base interface, a richer set of input signals is available for more complex instructions
  - μ-operations' metadata
  - All the Vector CSRs status value

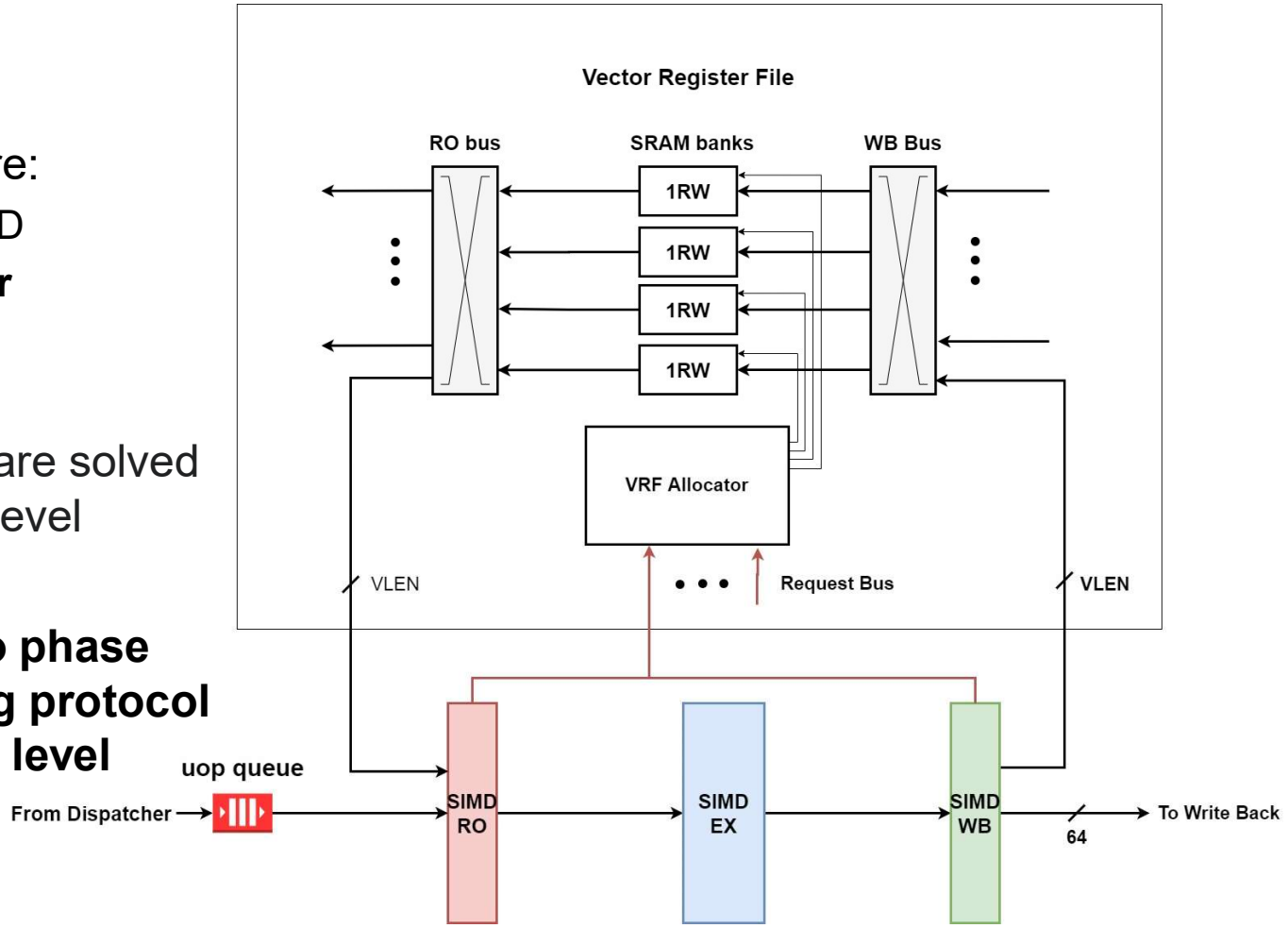
Vector Write Back

- Accepts write back requests carrying:
  - A scalar result
  - An exception
  - Controls for the vector CSRs
- Serves requests
  - From the Vector Configuration and Exception module
  - From the functional units
  - To the write back ports *(over an enable signal)*
- A request/grant handshake
- Simple static priority scheme



Vector Register File

- Multi-banked memory (4 1RW banks)
- µ-operations are:
  - Packed-SIMD
  - **Out-of-order**
  - **Chained**
- **Data hazards** are solved at µ-operation level
- Arbitration: **two phase shared locking protocol at µ-operation level**

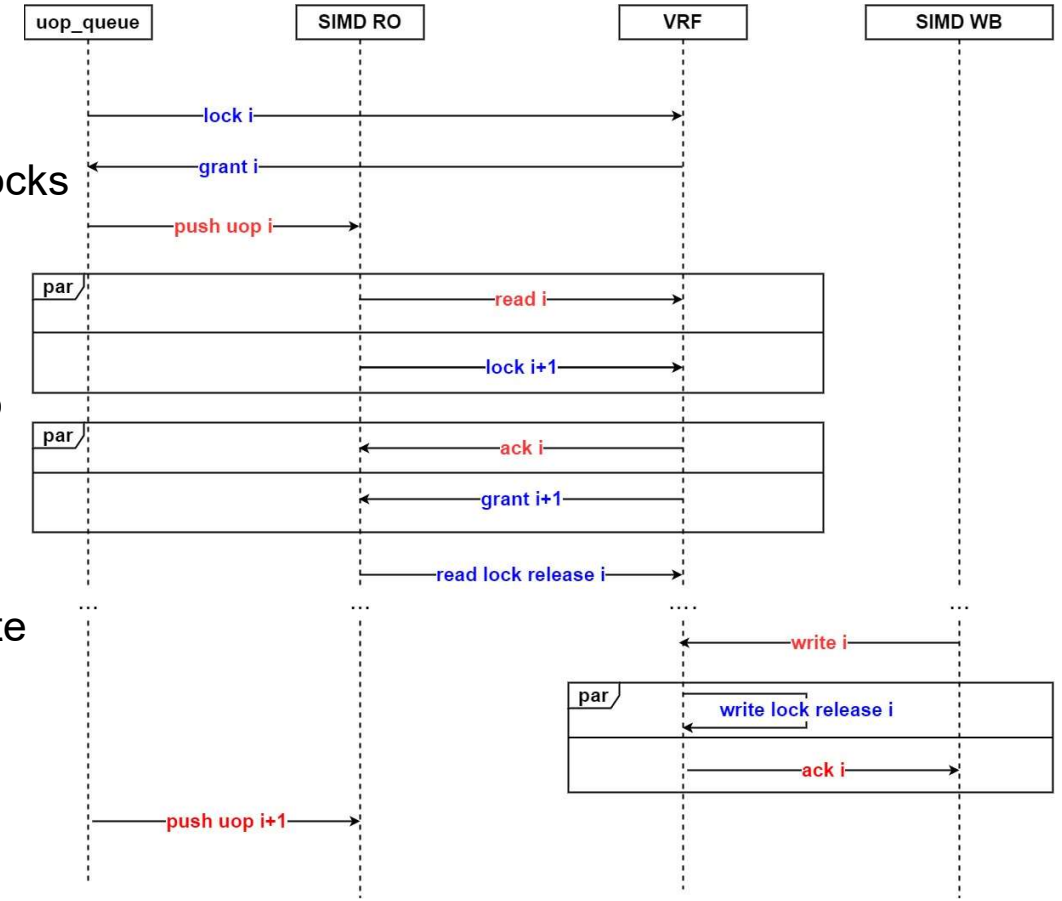


Locking Protocol

- Requirements:
  - "*Each μ-operation must hold a proper lock on all its vector registers operands before performing any operation*“
  - Allow μ-operation chaining
- Chosen base protocol:
  - Two-phase shared lock at μ-operation level
- Main rules:
  1. after being pushed, a μ-operation can perform reads and writes on its operands being sure to be holding the proper locks;
  2. during operand read all locks for the next μ-operation, read and write, must be acquired;
  3. read locks can be released after the read operation is successfully completed and the requested locks are granted;
  4. write locks must be released only after write back.
- There are some corner cases
  - e.g., exceptions, first and last μ-operation, mask register

Locking Protocol, scenario

- Micro-operations queue must lock the operand registers only for the first active μ-op
  - Because of rule 1
- As soon as the VRF grants all the locks the first μ-op can be pushed
- SIMD Read Operands, in parallel:
  - Reads its operands
  - Locks the operands of the next μ-op
- As soon as both the acks and the grants arrive, read locks can be released
- When the result is ready, SIMD Write Back writes it and the register is automatically released on ack
- Next μ-op can now be issued



Future work

- Alternative VRF design
  - Flip-flop design with checkpointing techniques for speculative execution
  - Simpler allocation logic
  - Wider area
- Additional SIMD functional units
  - Presented design offers an easily extensible framework
- Memory Architecture for Vector Load/Store Unit
  - Heavily impacted by the memory consistency model (still unspecified for RV V-extension)
- Performance evaluation
- Post-synthesis analysis
  - Only preliminary results so far