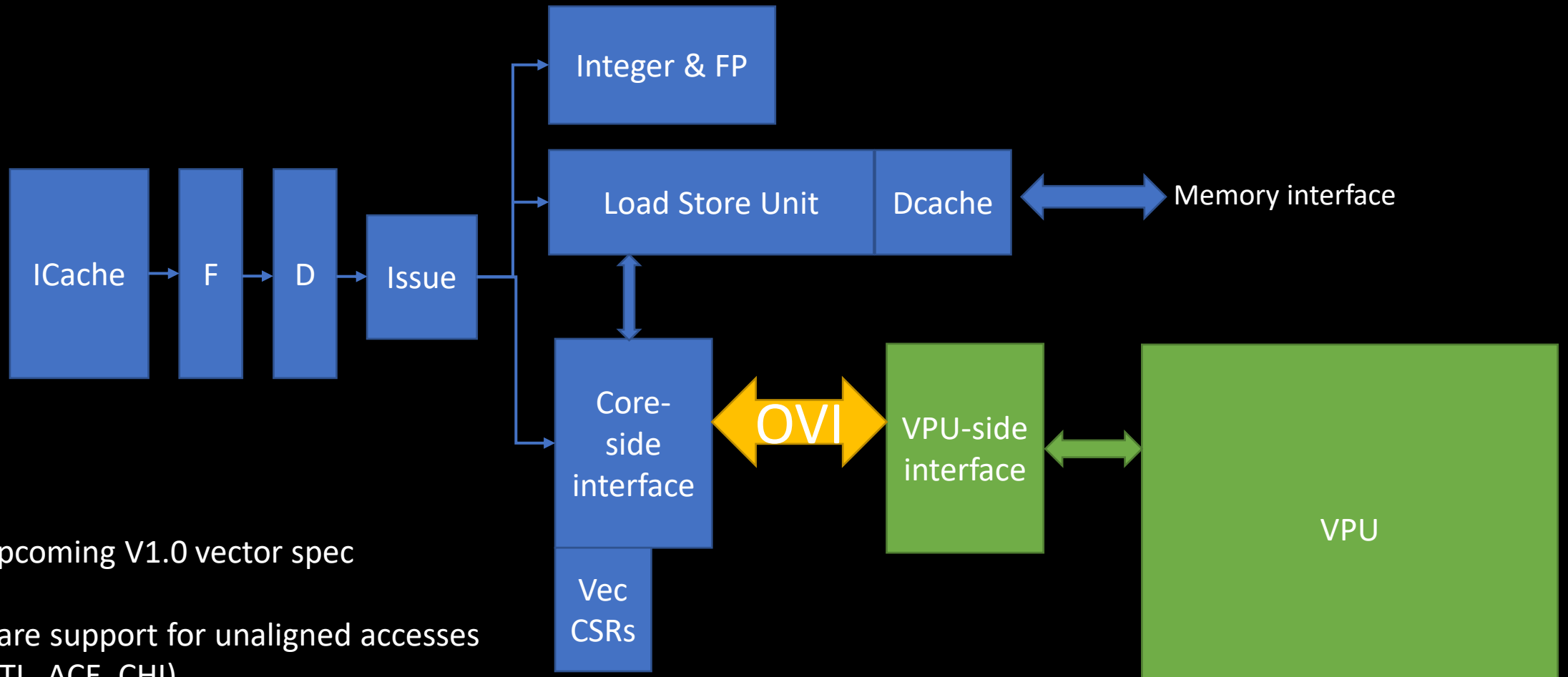
OVI:
The Open Vector Interface

Roger Espasa, PhD, CEO
Alberto Moreno, PhD, HW Architect

What IS OVI?

- An open-source specification to connect a RISC-V VPU to a RISC-V core
- Division of responsibilities
  - Core performs vector memory accesses, holds control CSRs
  - VPU performs vector computations
  - All vector instructions sent to VPU nonetheless
- Simple, credited, synchronization scheme between core & vpu
- Protocol does not constrain VPU implementation
  - Can be “in-order” or “out-of-order”
- Semidynamics has used this protocol to connect
  - Avispado 220 core from Semidynamics
  - Vitruvius VPU from BSC
  - For the European Processor Initiative (EPI) accelerator (EPAC)

Avispado 220 core with OVI



- SV48
- 16KB I$
- Decodes upcoming V1.0 vector spec
- 32KB D$
- Full hardware support for unaligned accesses
- Coherent (TL, ACE, CHI)
- Vector Memory (vle, vlse, vlxe, vse, …) processed by Load Store Unit

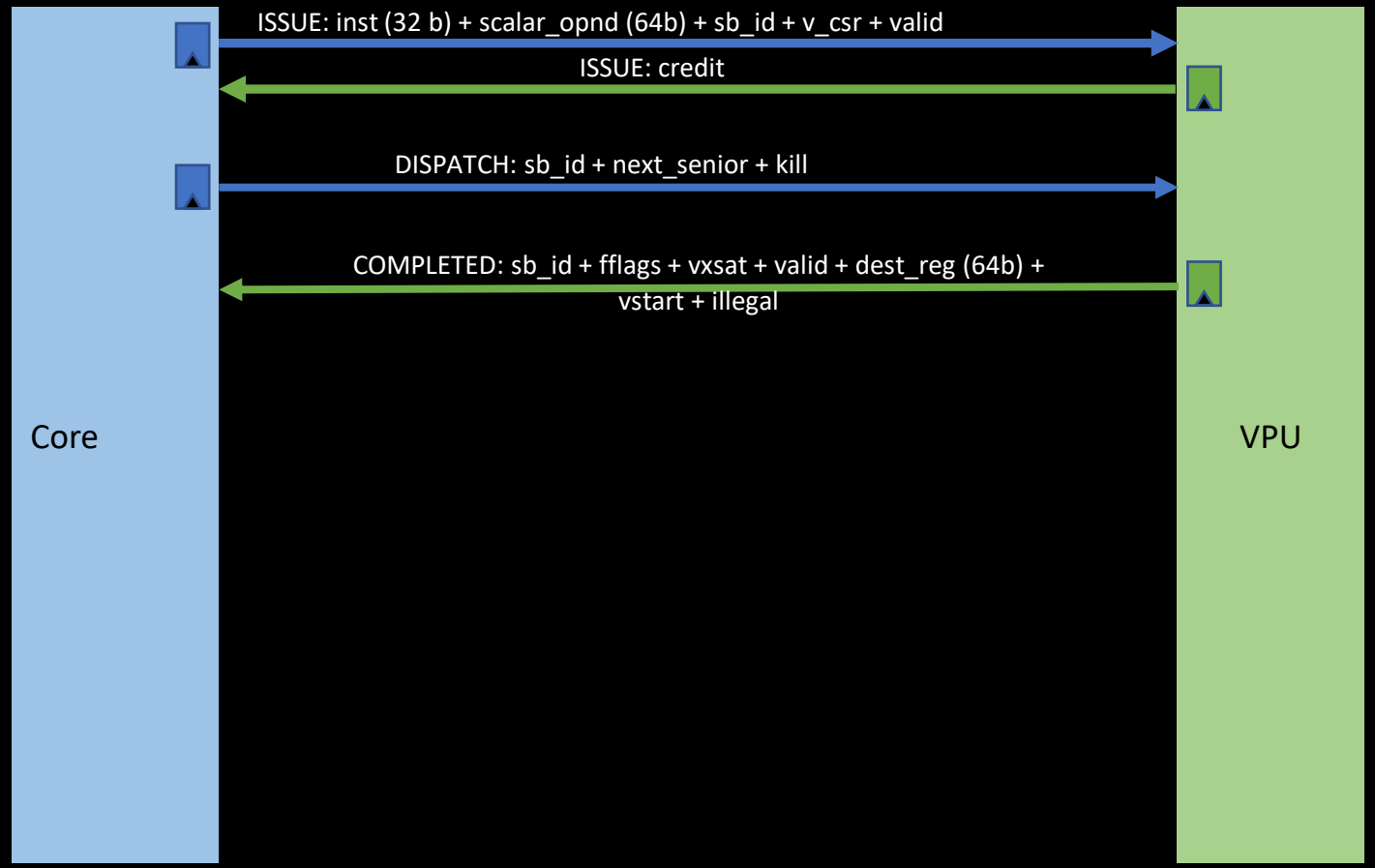
OVI main features

- Vector processing unit (VPU) decoupled from the core
- Core only decodes configuration and memory instructions
- Allows the VPU to have “secret” instructions
- VPU may be implemented Out-of-order or in-order

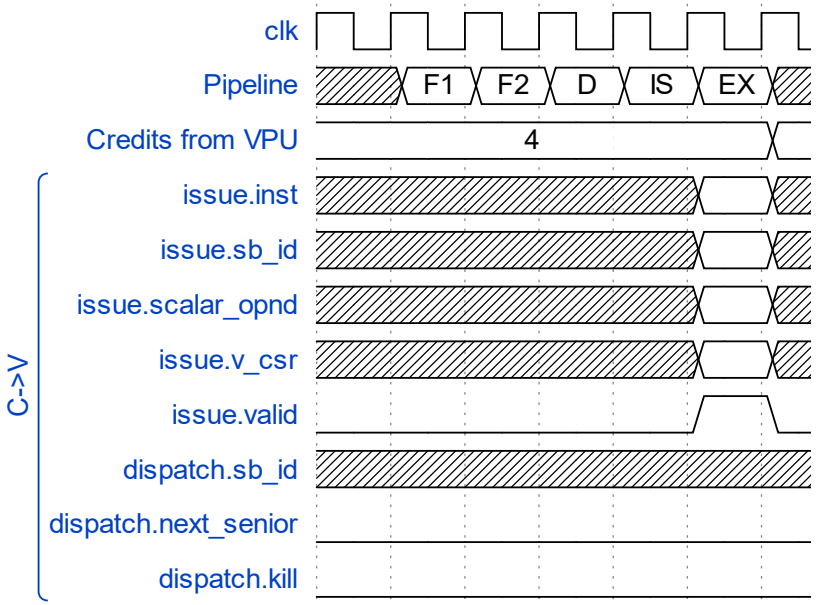
Vector instruction types

- Configuration instructions (csr read/write, vset{i}vl{i})
  - Executed on the core without involving the OVI
- Arithmetic operations
  - Executed on the VPU. OVI is used to issue the instruction
- Memory operations – Vload & Vstore
  - Executed on the VPU and the Core. OVI intermediates data transfers

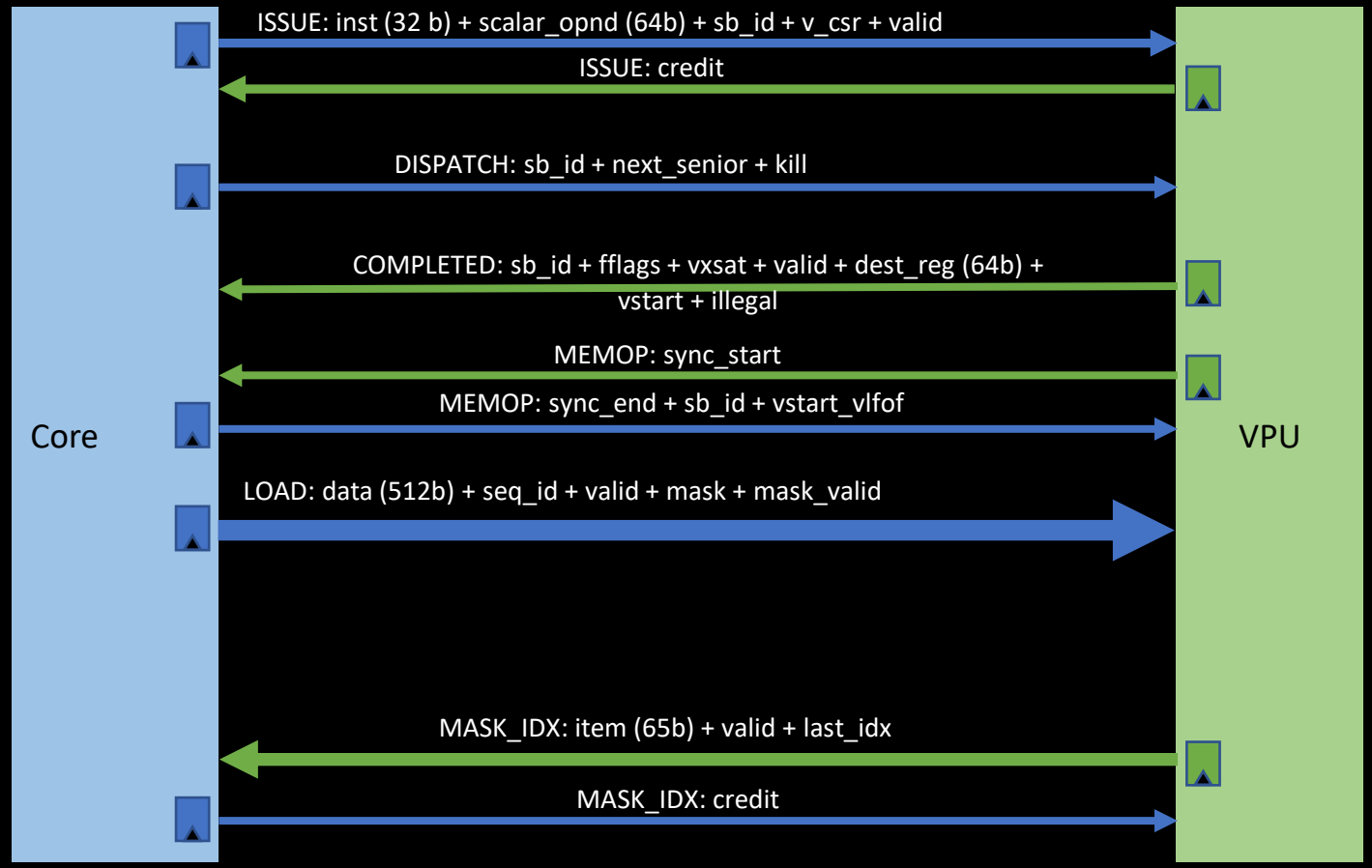
Interface overview – Issue & completion



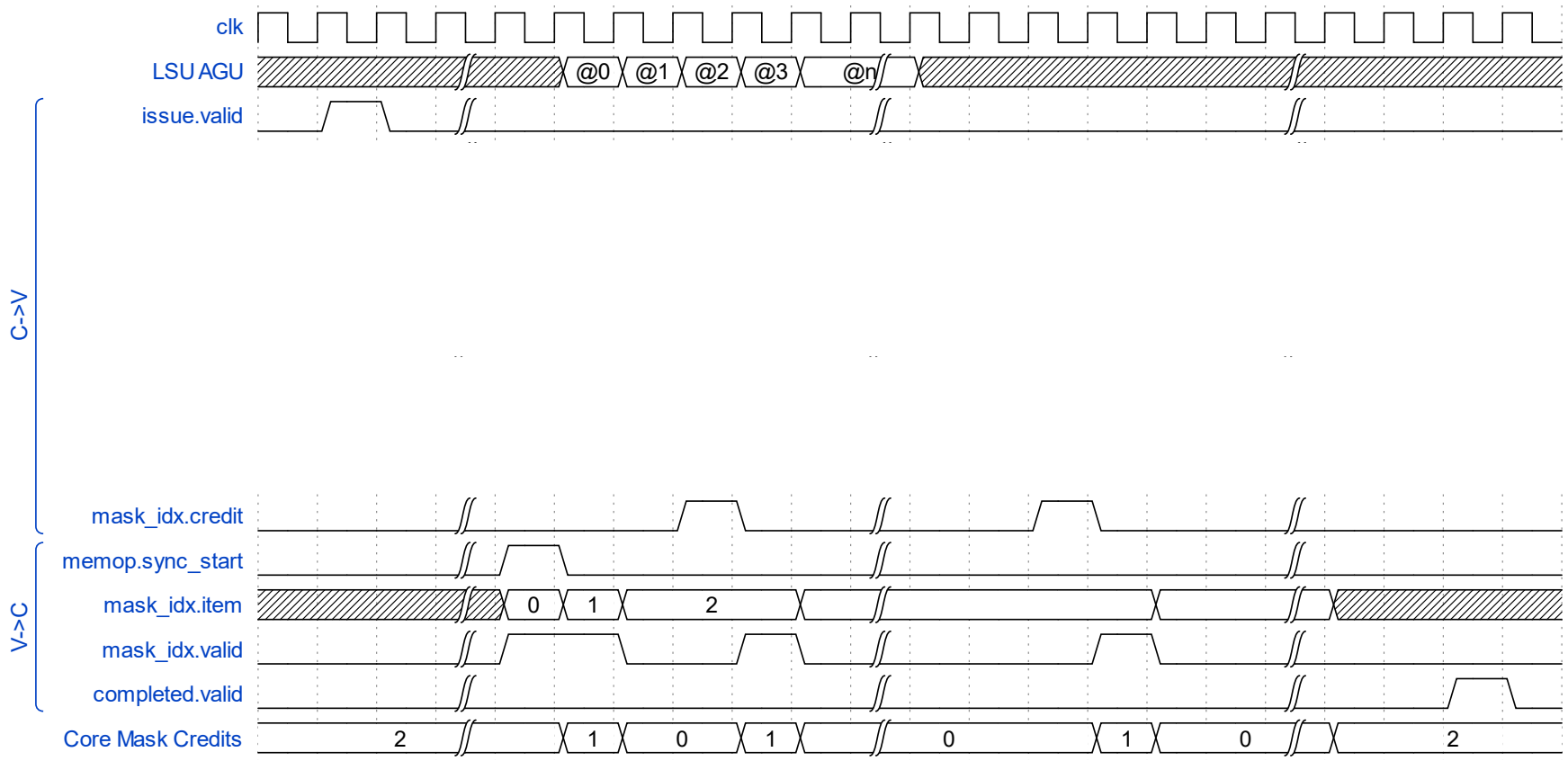
Arithmetic instructions



Interface overview – Vload instructions



Vload instructions



Vload instructions - Sequence ID

- The core may receive out-of-order data from memory
- Multiple vloads can cohexist at the same time
- The core sends “cache lines” to the VPU, without processing them
- For all these reasons, we need a way to identify where to write the data on the VPU

Vload instructions - Sequence ID

- The sequence ID encodes, for every line of cache served to the VPU:
  - Logic vector register
  - Id for the element or elements contained on the line
  - Offset for the valid data
  - Count for the amount of valid elements
  - Instruction ID to identify the specific vload
- Only specific strides are supported, called “fast strides”:
  - Unit-stride, 2-SEW-stride, 4-SEW-stride and their negative counterparts
- For any other stride, the sequence ID identifies exactly 1 element

Vload instructions - Sequence ID

- Examples for cache lines 512b, SEW=32b:

Unit-stride, element ID = 0, offset = 11, count = 5

ID	4	3	2	1	0										
offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

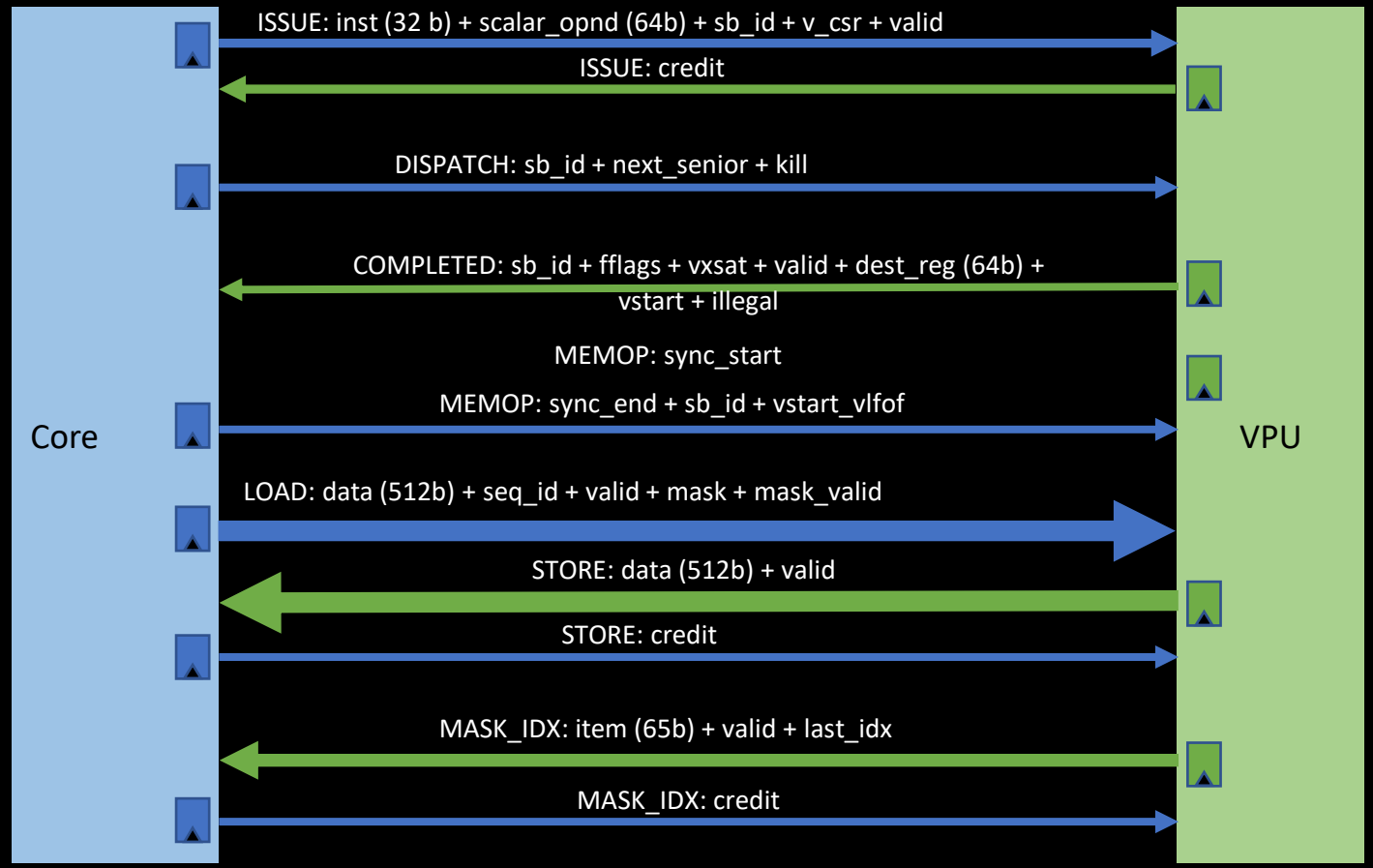
2-SEW-stride, element ID = 0, offset = 10, count = 3

ID		2		1		0									
offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

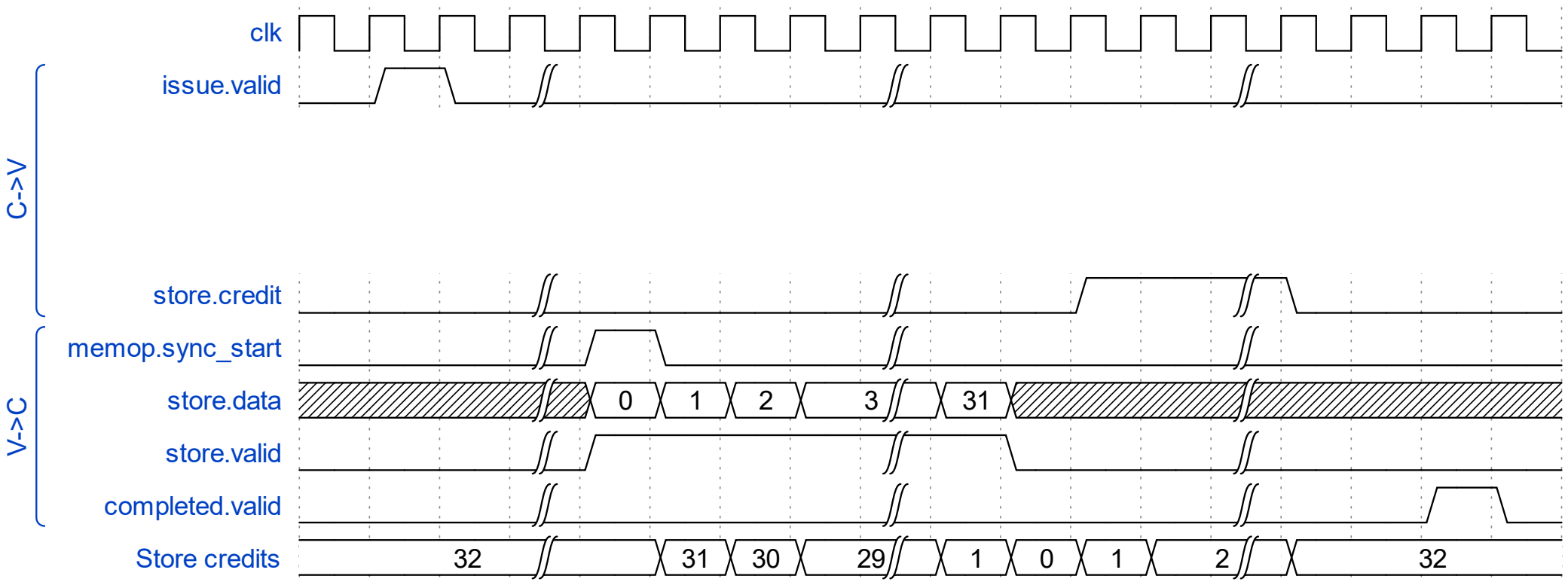
Indexed, element ID = 123, offset = 13, count = 1

ID			123												
offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Vstore instructions



Vstore instructions



Conclusions

- OVI defines a way to interconnect a VPU with a Core
- The interface is appropriate for very decoupled VPUs
- OVI makes few assumptions about the Core design
- VPU implements a vector decoder, so “secret instructions” are possible

GitHub repository

https://github.com/semidynamics/OpenVectorInterface