CSL

EVE: Ephemeral Vector Engines

Khalid Al-Hawaj, Tuan Ta, Nick Cebry, Shady Agwa, Olalekan Afuye, Eric Hall,
Courtney Golden, Alyssa B. Apsel, Christopher Batten

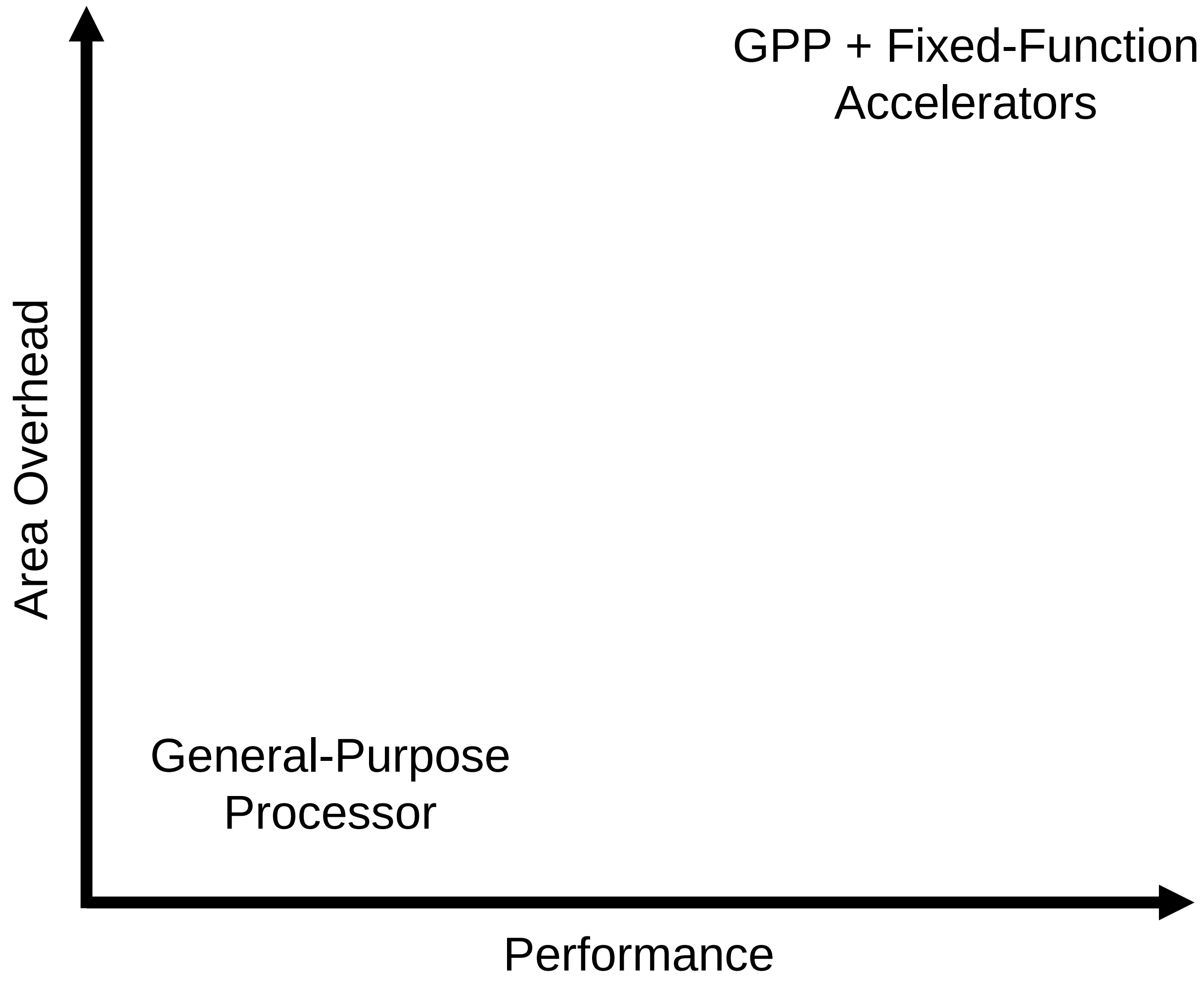
Electrical and Computer Engineering

Cornell University

HPCA-29
28th February 2023

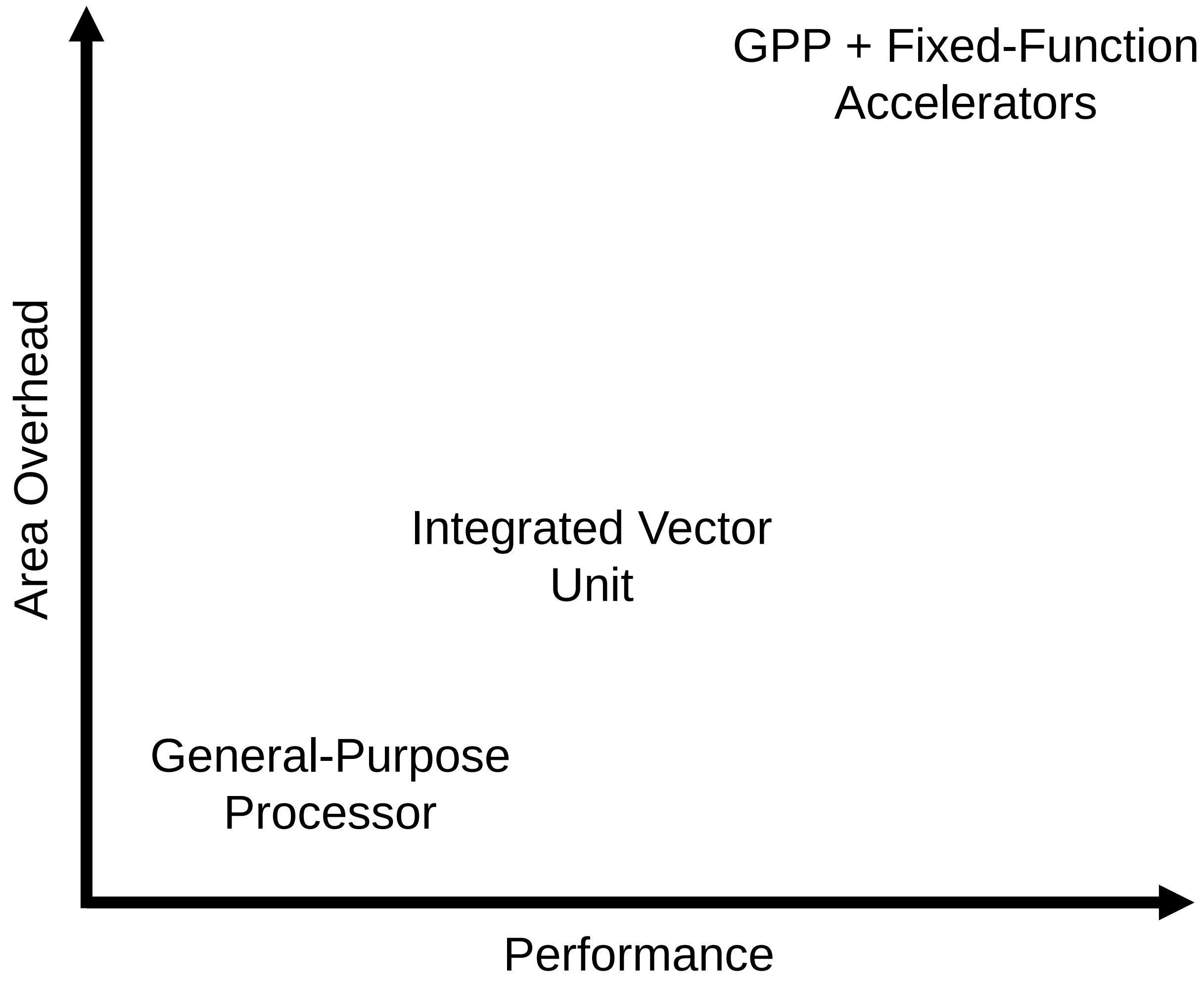
Trend 1: Resurgence of Interest In Vector Architectures

Trend 1: Resurgence of Interest In Vector Architectures

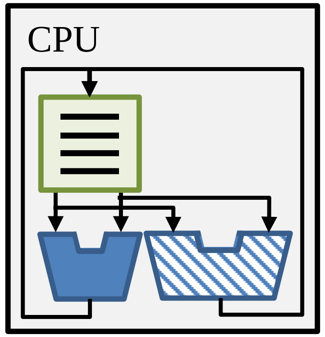


- Computer architects rely on specialization to increase performance and efficiency
- Vector micro-architectures to tackle regular data-parallel applications

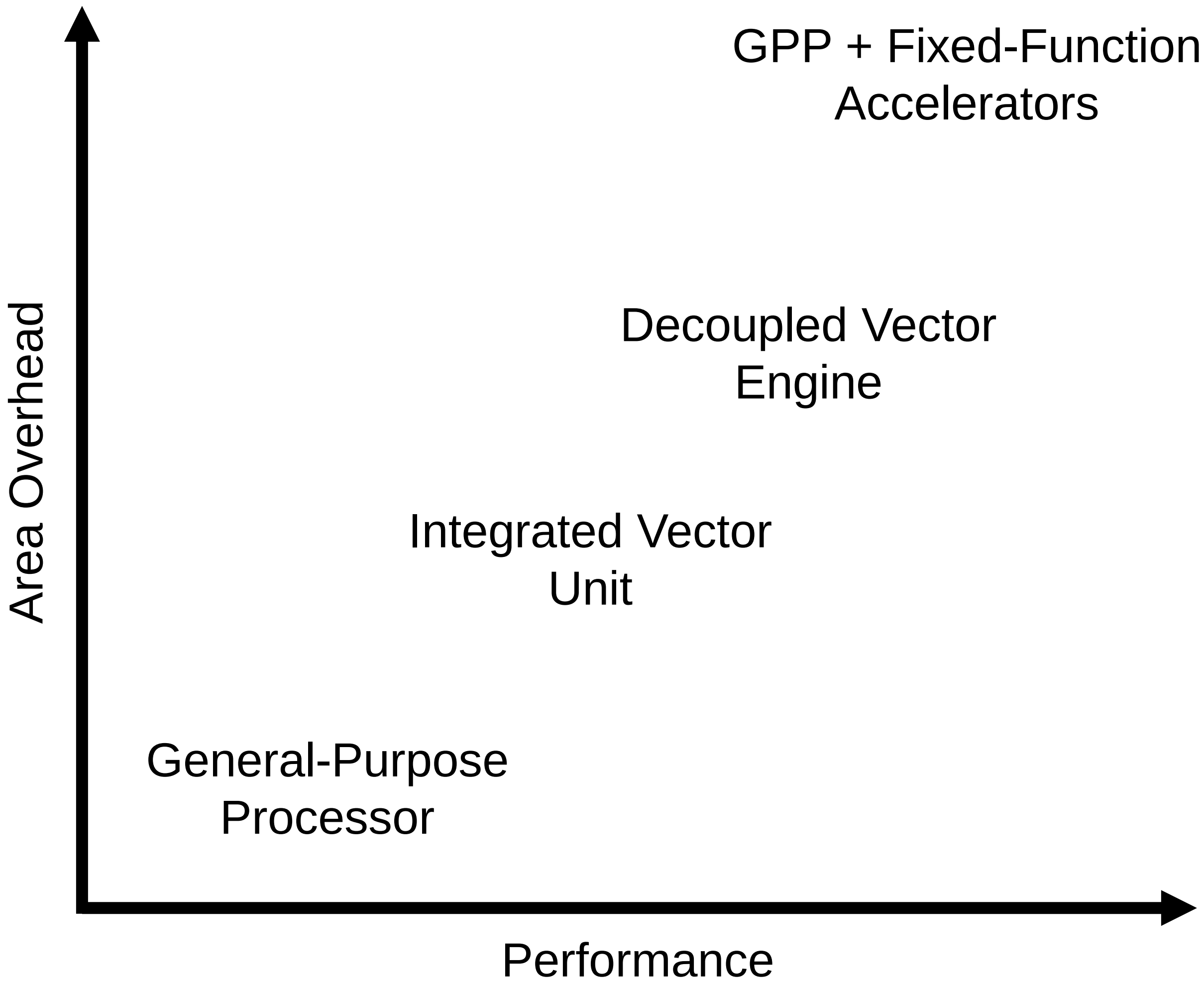
Trend 1: Resurgence of Interest In Vector Architectures



- Computer architects rely on specialization to increase performance and efficiency
- Vector micro-architectures to tackle regular data-parallel applications
  - Integrated vector unit (IV)



Trend 1: Resurgence of Interest In Vector Architectures



GPP + Fixed-Function Accelerators

Decoupled Vector Engine

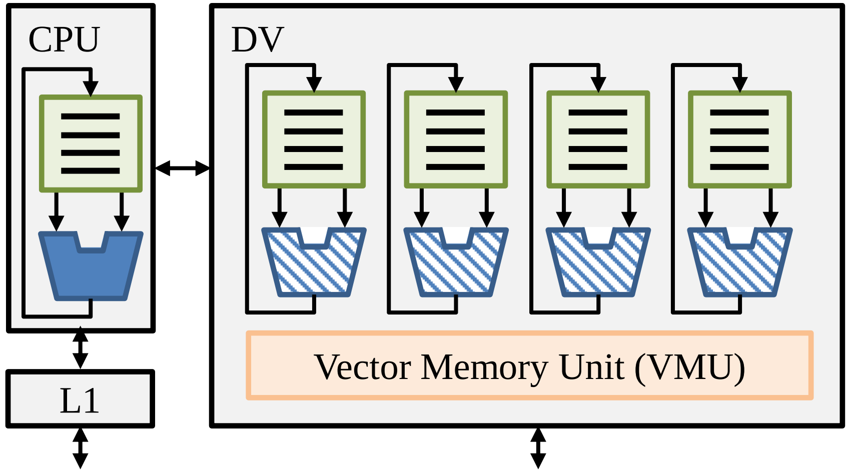
Integrated Vector Unit

General-Purpose Processor

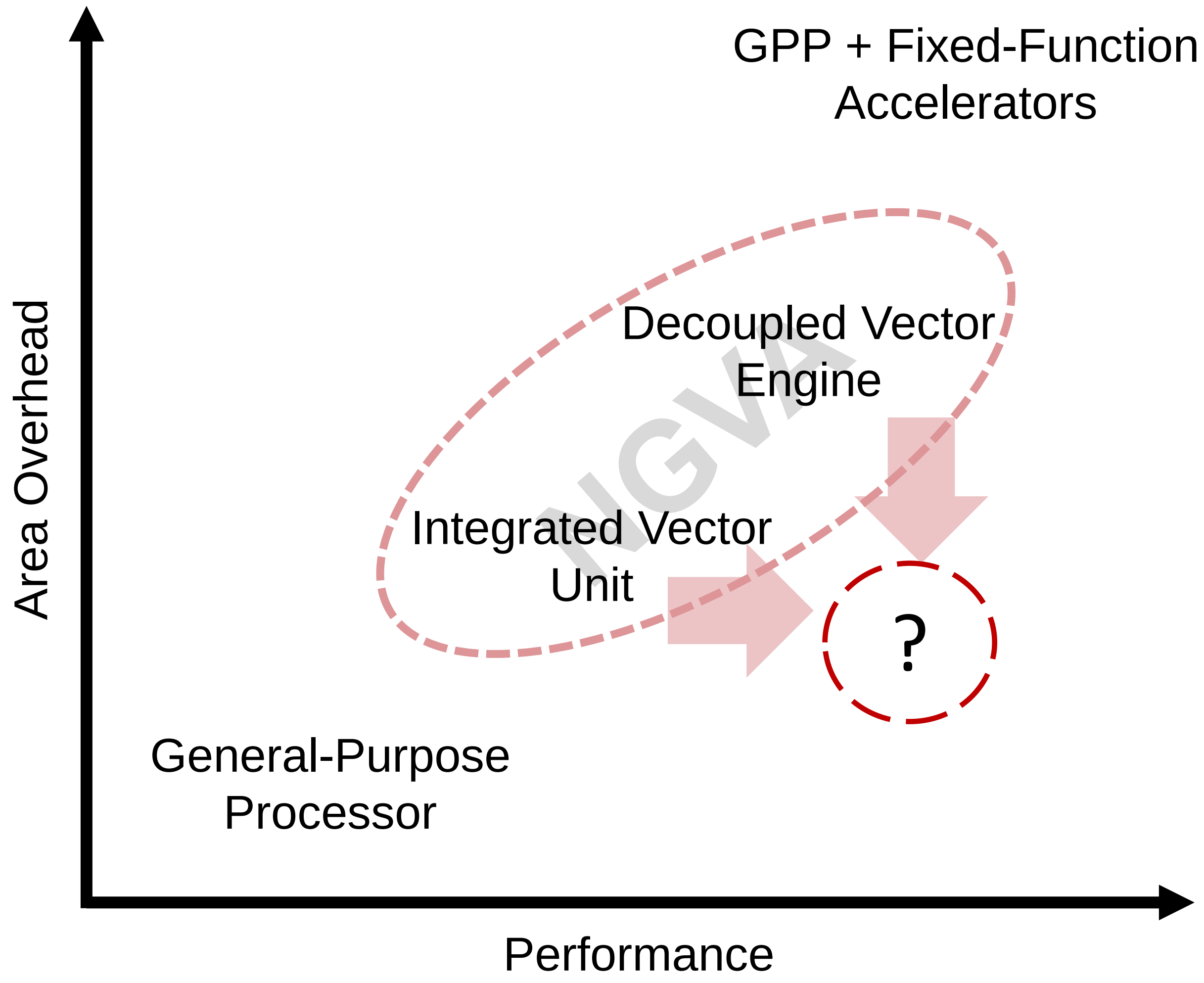
Area Overhead

Performance

- Computer architects rely on specialization to increase performance and efficiency
- Vector micro-architectures to tackle regular data-parallel applications
  - Integrated vector unit (IV)
  - Decoupled vector engine (DV)



Trend 1: Resurgence of Interest In Vector Architectures



- Computer architects rely on specialization to increase performance and efficiency
- Vector micro-architectures to tackle regular data-parallel applications
  - Integrated vector unit (IV)
  - Decoupled vector engine (DV)
- Emerging trend of next-generation vector architectures (NGVA)

Is it possible to achieve performance comparable to a *DV* while incurring an area overhead equivalent to an *IV*?

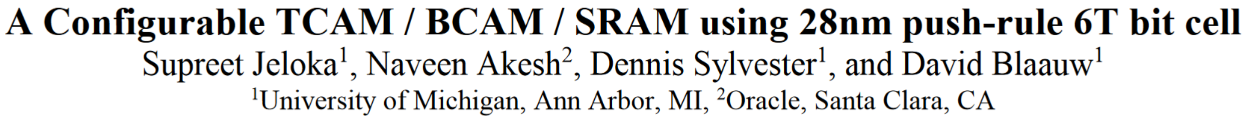
Trend 2: Recent Work on SRAM-Based Compute-In-Memory

Trend 2: Recent Work on SRAM-Based Compute-In-Memory

- Recent work on SRAM-based compute-in-memory have shown promise in alleviating the area-overhead often associated with vector execution

- Subsequent work has explored implementing more complex operations on-top of bit-line compute

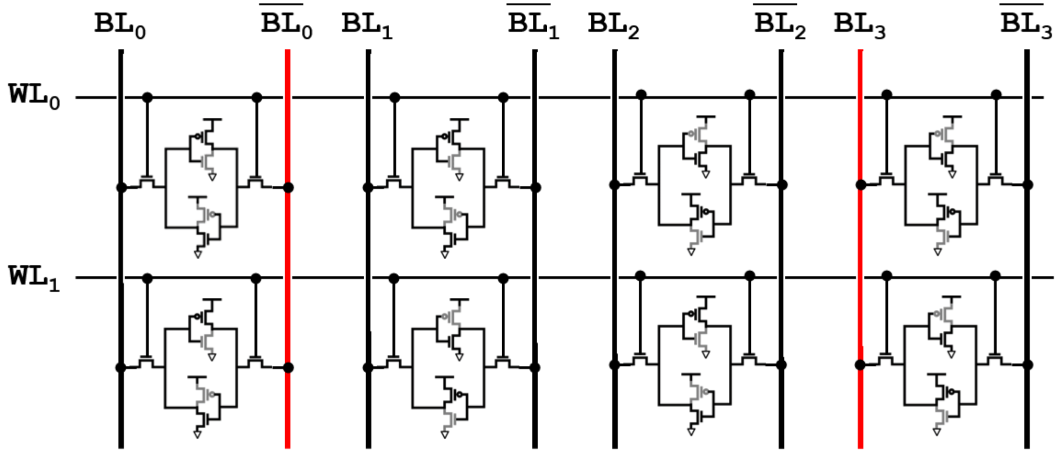
What abstraction would be more suitable to enable high-programmability of an SRAM-based compute-in-memory micro-architecture?



A Configurable TCAM / BCAM / SRAM using 28nm push-rule 6T bit cell

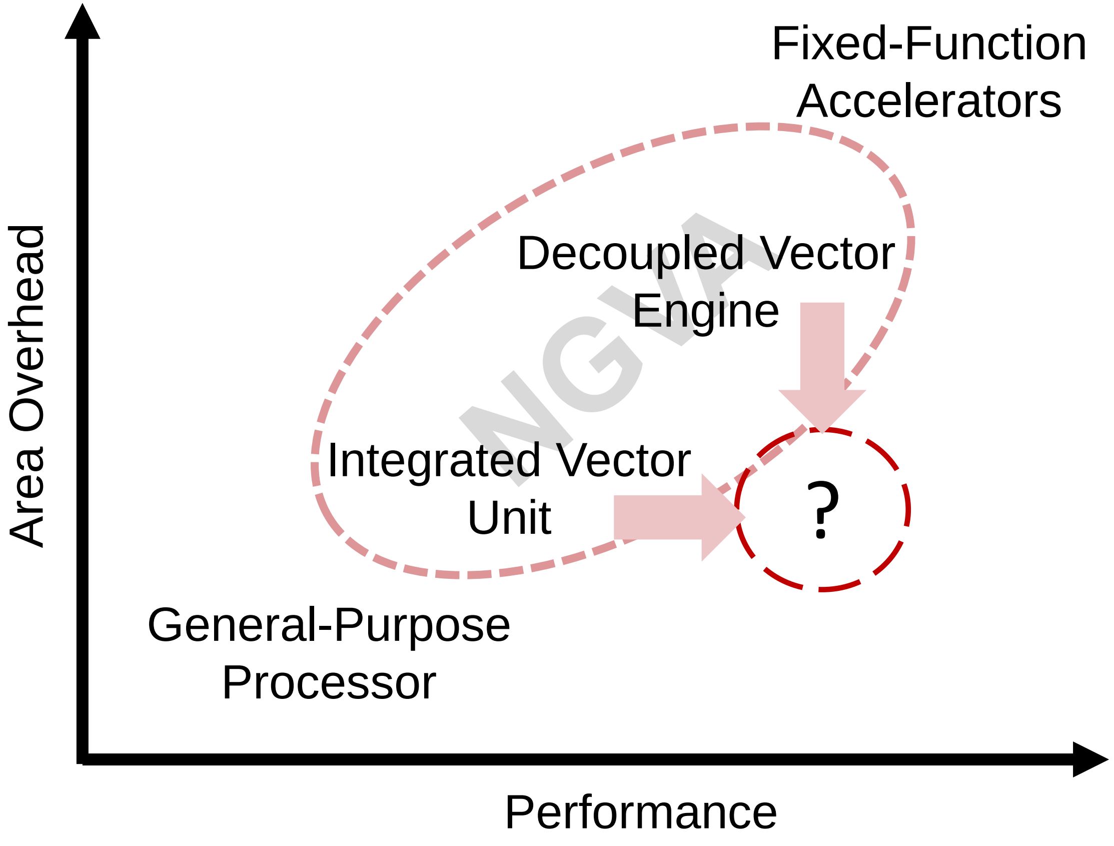
Supreet Jeloka[1], Naveen Akesh[2], Dennis Sylvester[1], and David Blaauw[1]

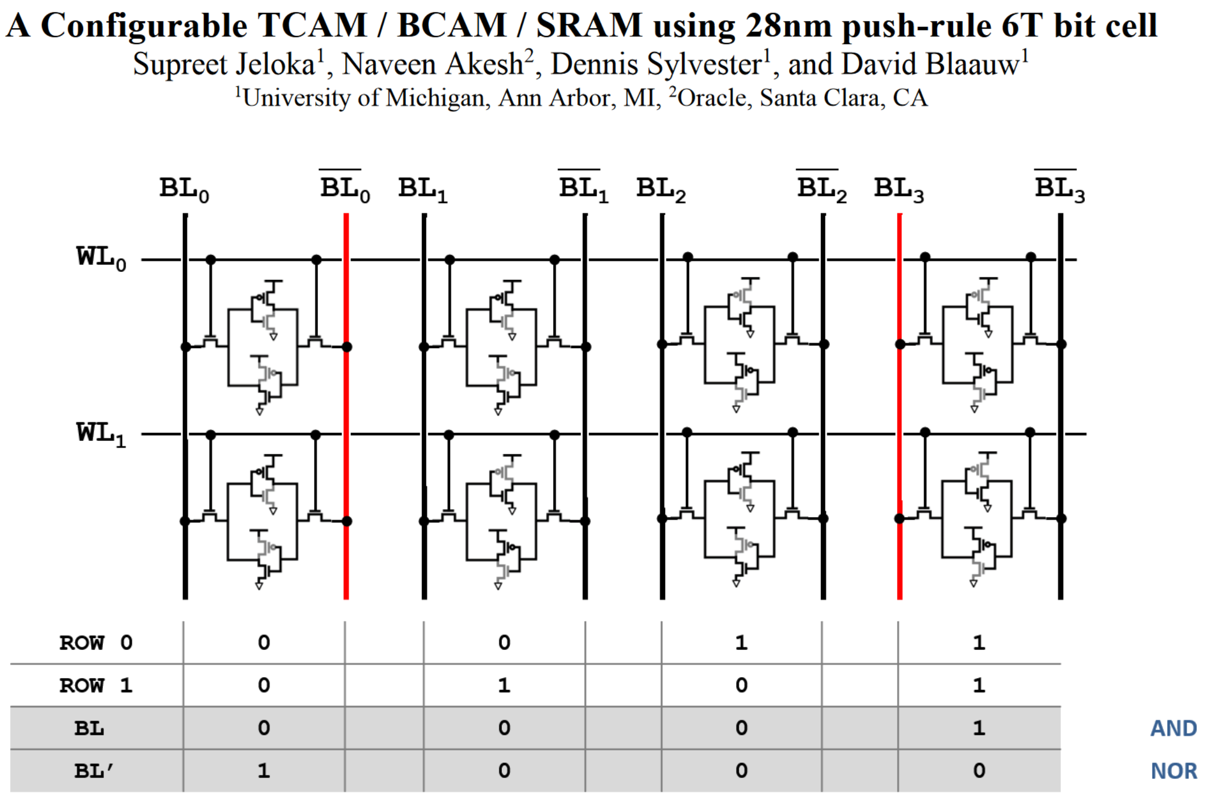
[1]University of Michigan, Ann Arbor, MI, [2]Oracle, Santa Clara, CA



ROW 0	0		0		1		1	
ROW 1	0		1		0		1	
BL	0		0		0		1	AND
BL′	1		0		0		0	NOR

EVE's Goals





A Configurable TCAM / BCAM / SRAM using 28nm push-rule 6T bit cell

Supreet Jeloka[1], Naveen Akesh[2], Dennis Sylvester[1], and David Blaauw[1]

[1]University of Michigan, Ann Arbor, MI, [2]Oracle, Santa Clara, CA

ROW 0	0	0	1	1	
ROW 1	0	1	0	1	
BL	0	0	0	1	AND
BL′	1	0	0	0	NOR

Is it possible to achieve performance comparable to a *DV* while incurring an area overhead equivalent to an *IV*?

What abstraction would be more suitable to enable high-programmability of an SRAM-based compute-in-memory micro-architecture?

EVE's Contributions

	Duality Cache [ISCA2019]	EVE
Abstraction	SIMT/Threading	Next-Generation Vector
Cache Hierarchy	Last Level Cache	Level-2 Cache
Private or Shared	Shared	Private
Execution Paradigm	Bit-Serial Execution	Bit-Hybrid Execution

- Architectural template for a novel SRAM-based compute-in-memory next-gen vector engine that supports the full RISC-V RVV specifications

- Bit-hybrid execution to balance throughout and latency by alleviating row and column under-utilization

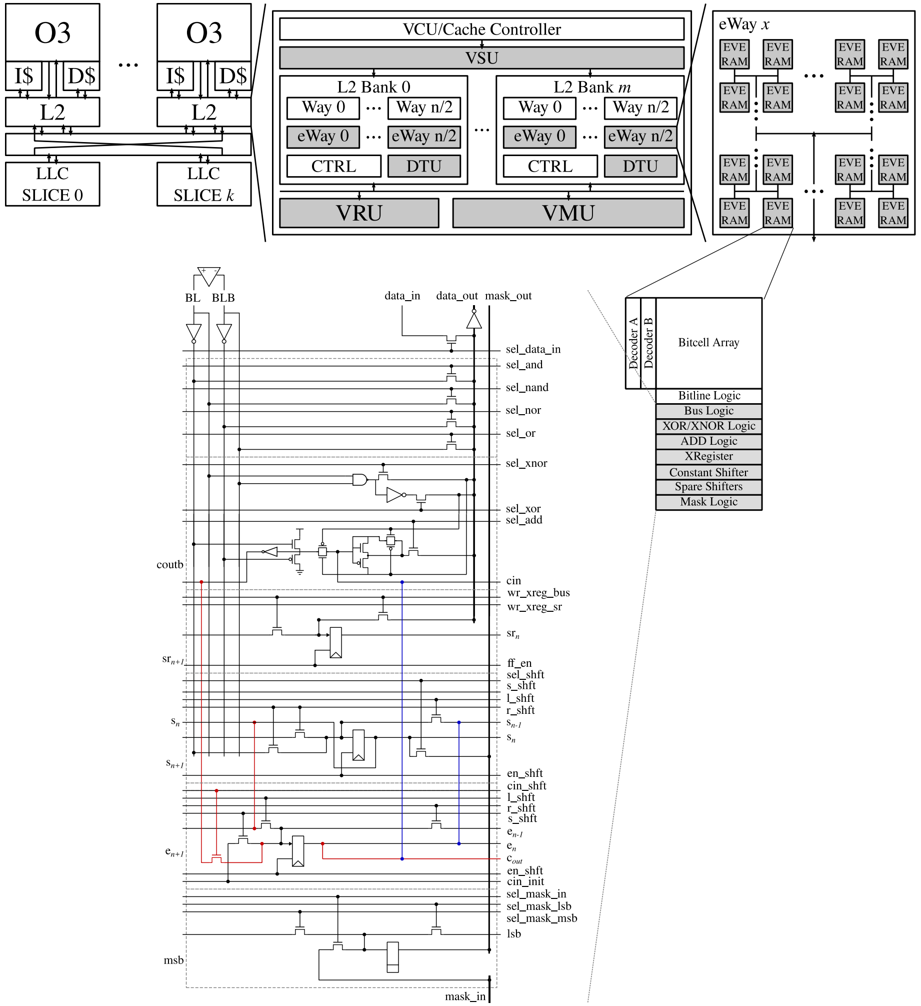
- Detailed evaluation of EVE show-casing the impacts and benefits of bit-hybrid execution on an SRAM-based compute-in-memory micro-architecture

OUTLINE

Motivation

- EVE Micro-Architecture
- EVE Bit-Hybrid Execution Paradigm
- EVE Micro-Programming & Circuits
- EVE Evaluation

Conclusion

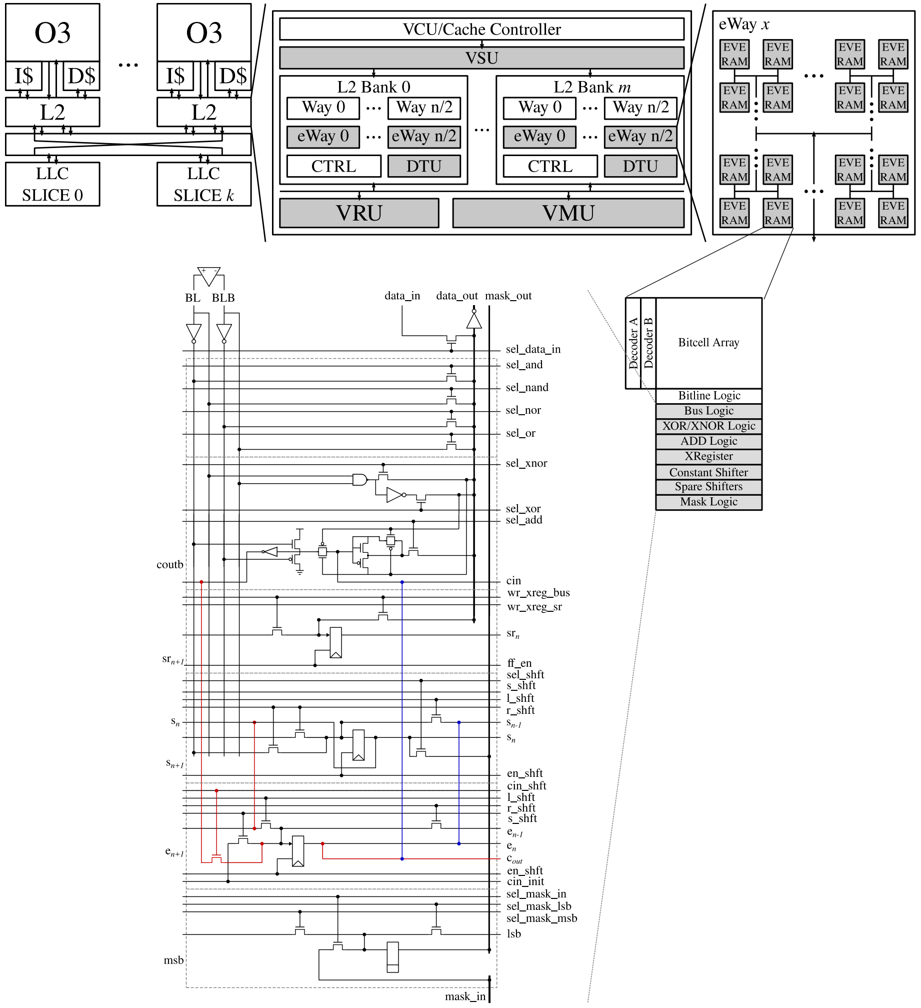


OUTLINE

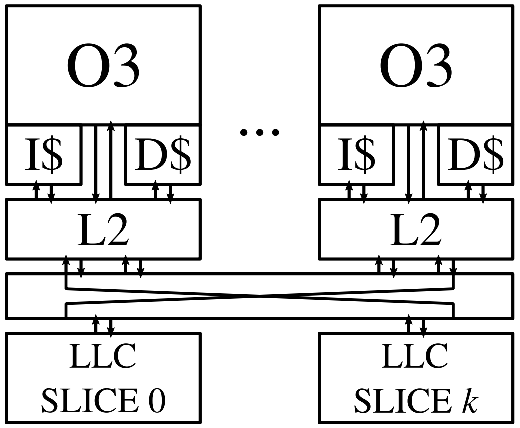
Motivation

- **EVE Micro-Architecture**
- EVE Bit-Hybrid Execution Paradigm
- EVE Micro-Programming & Circuits
- EVE Evaluation

Conclusion

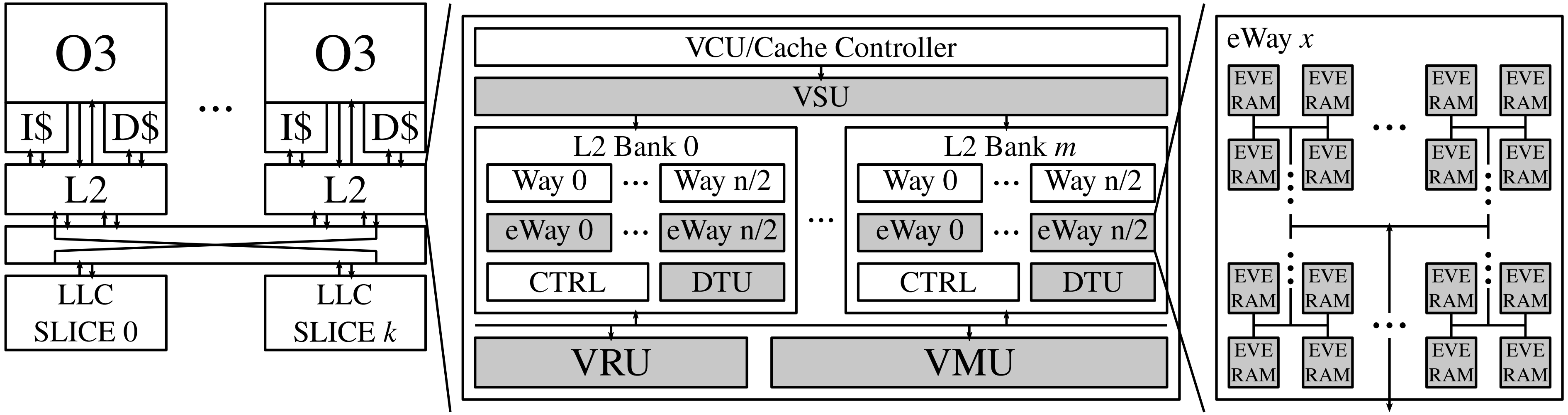


EVE MICRO-ARCHITECTURE

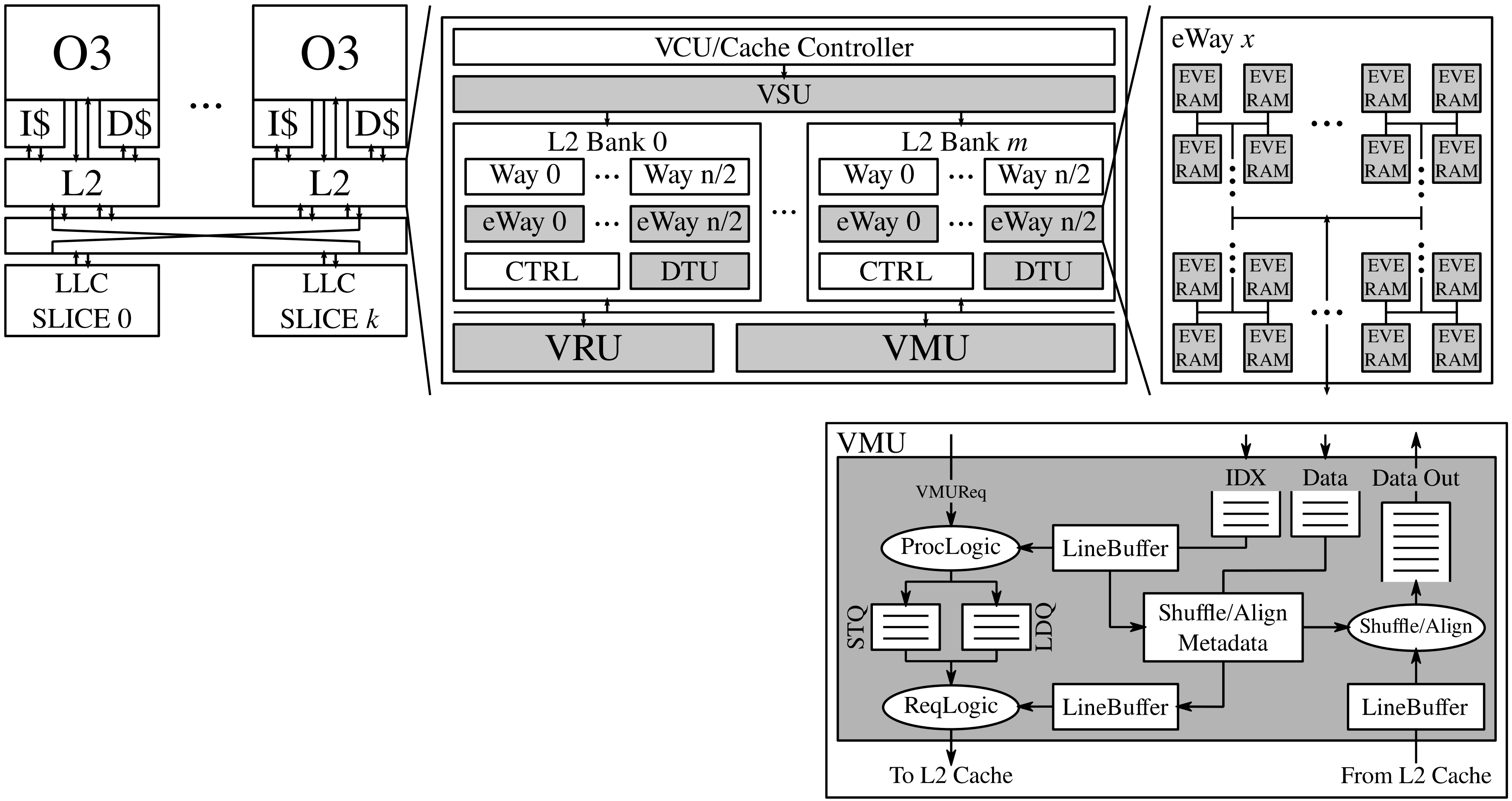


EVE reconfigure parts of the private **L2 cache** to act as the vector execution hardware **on-demand**

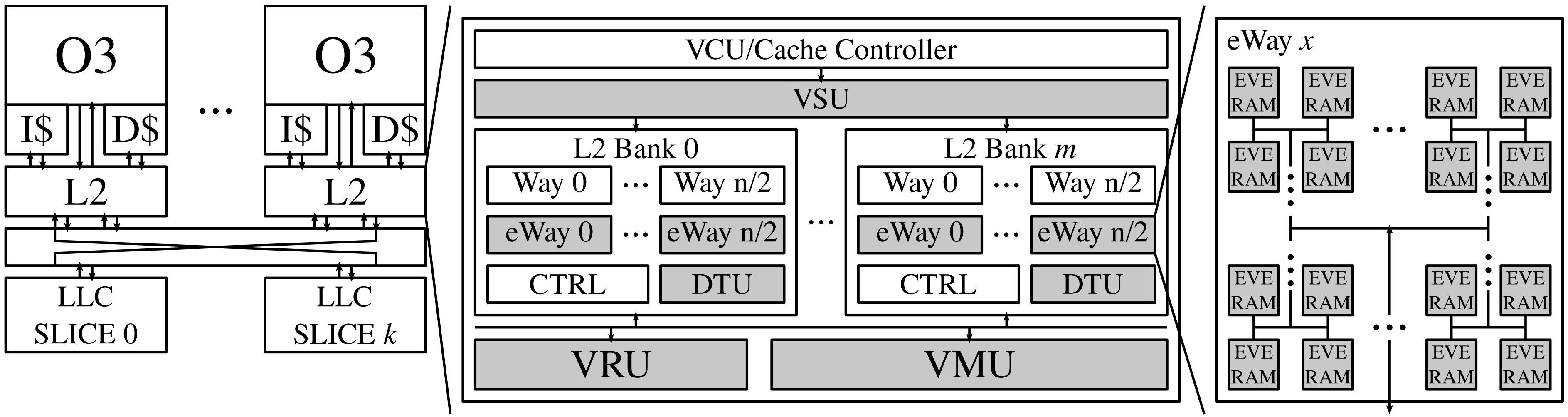
EVE MICRO-ARCHITECTURE



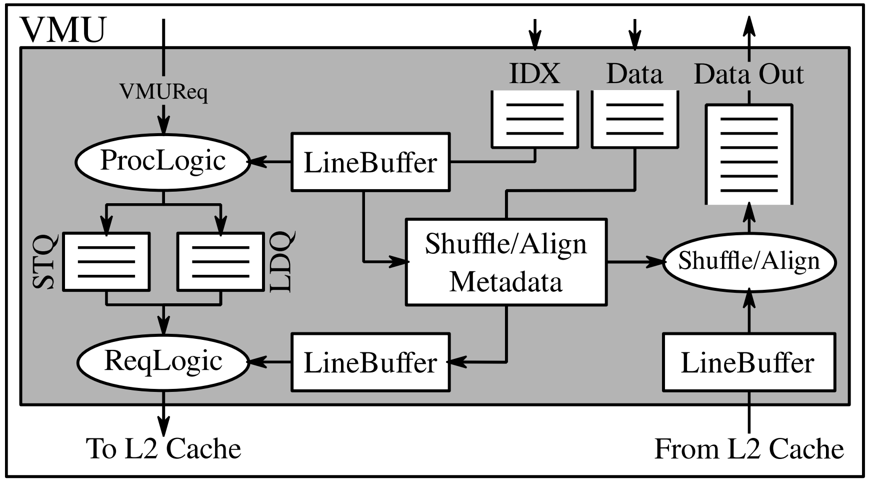
EVE Micro-Architecture



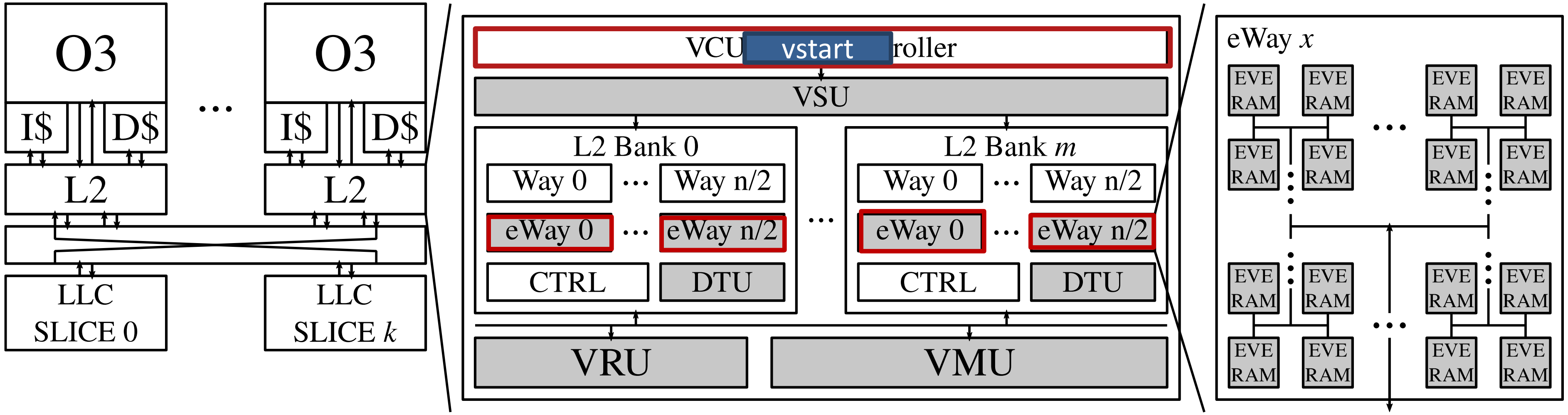
EVE Micro-Architecture



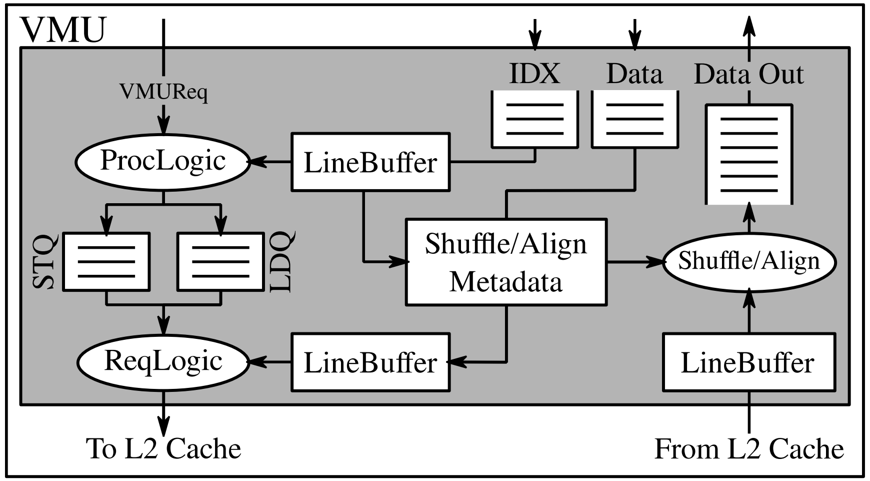
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



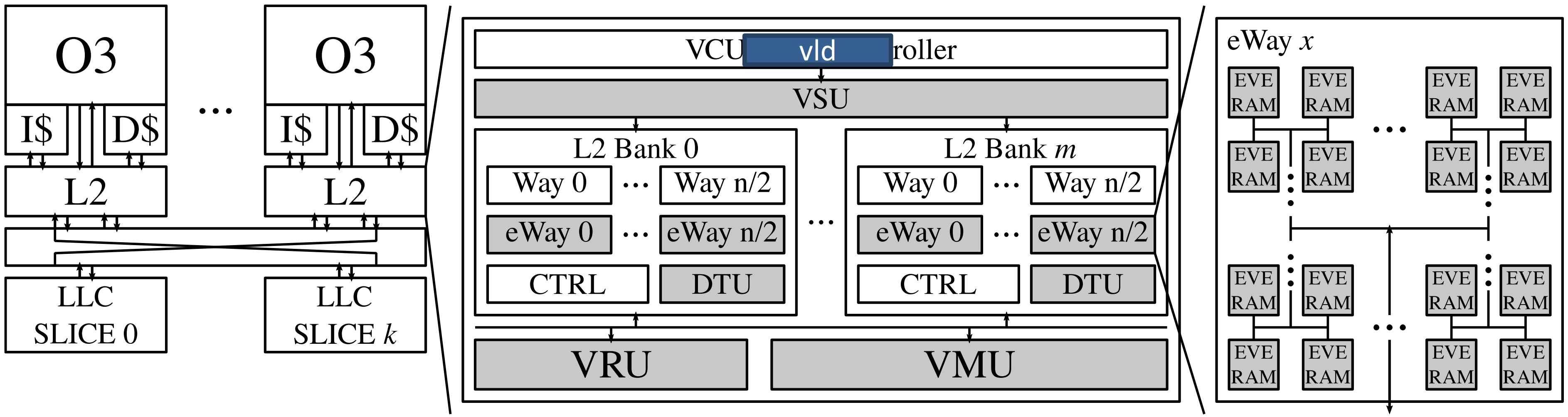
EVE Micro-Architecture



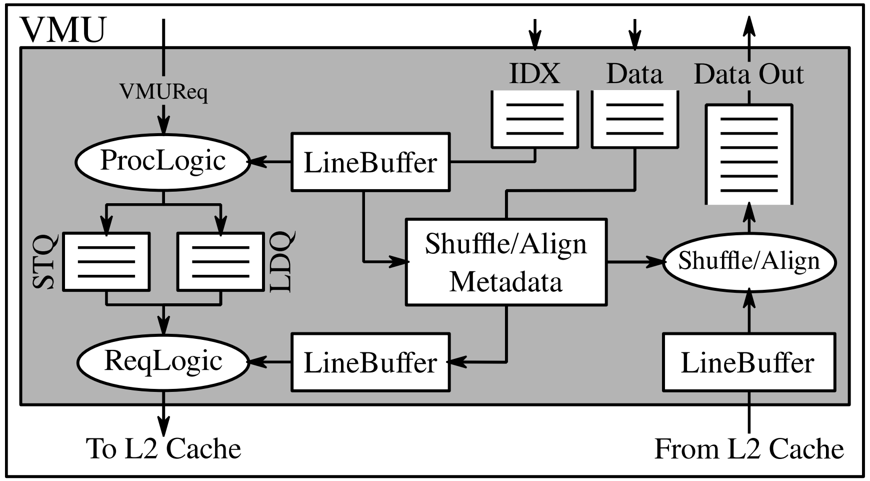
```
void vvadd( int* c, int* a, int * b) {
 vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



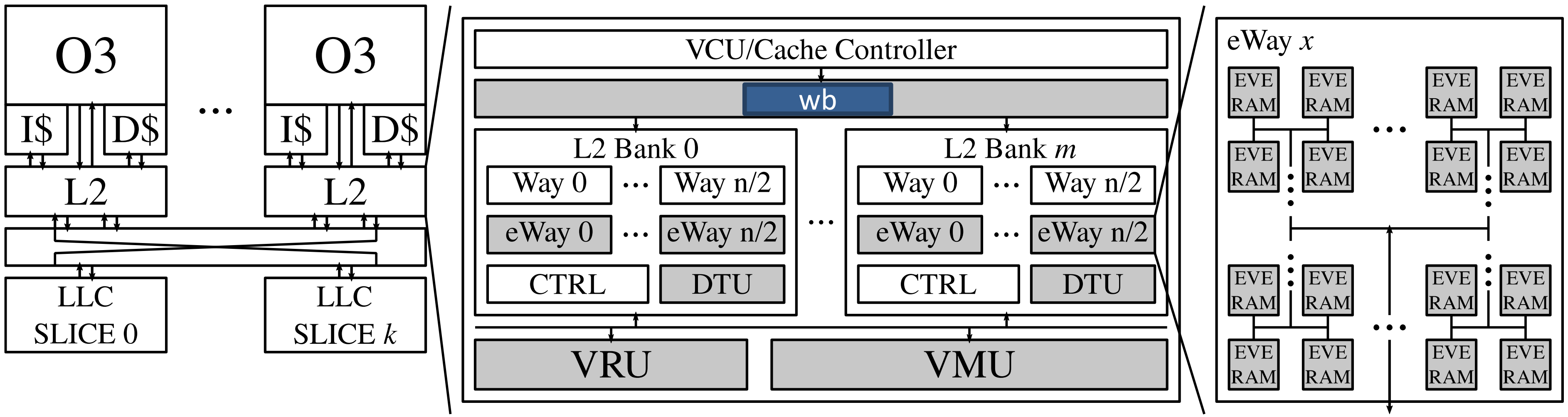
EVE Micro-Architecture



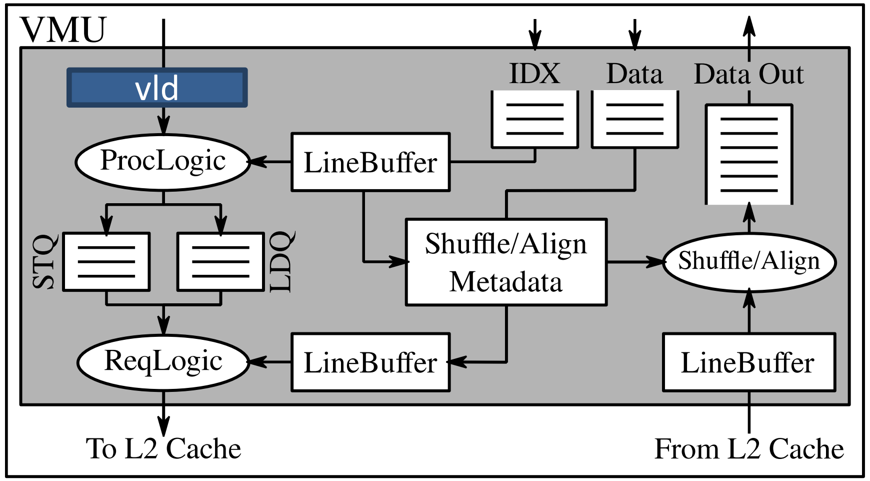
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



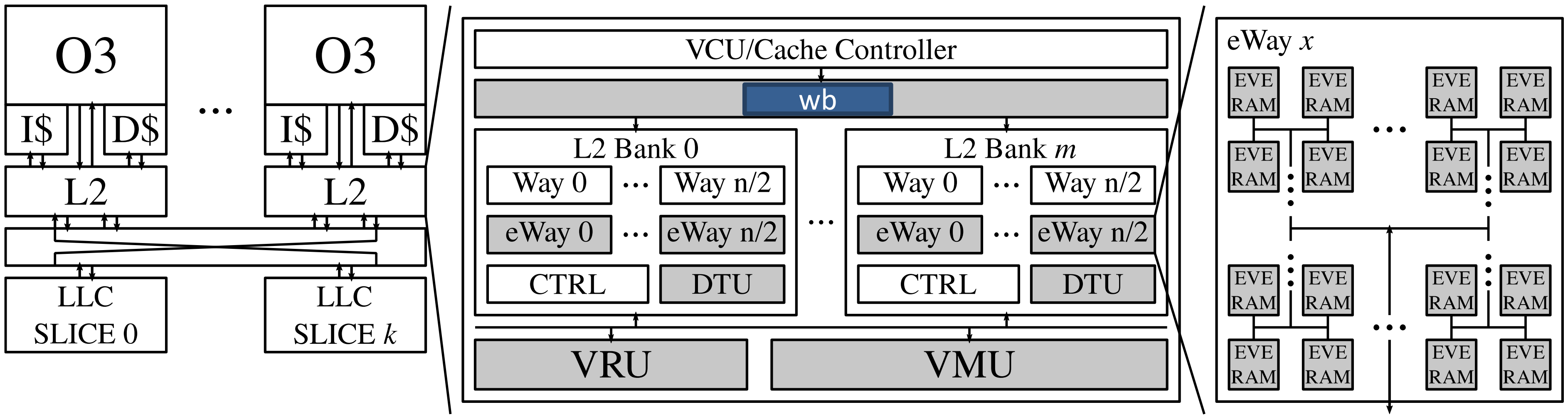
EVE MICRO-ARCHITECTURE



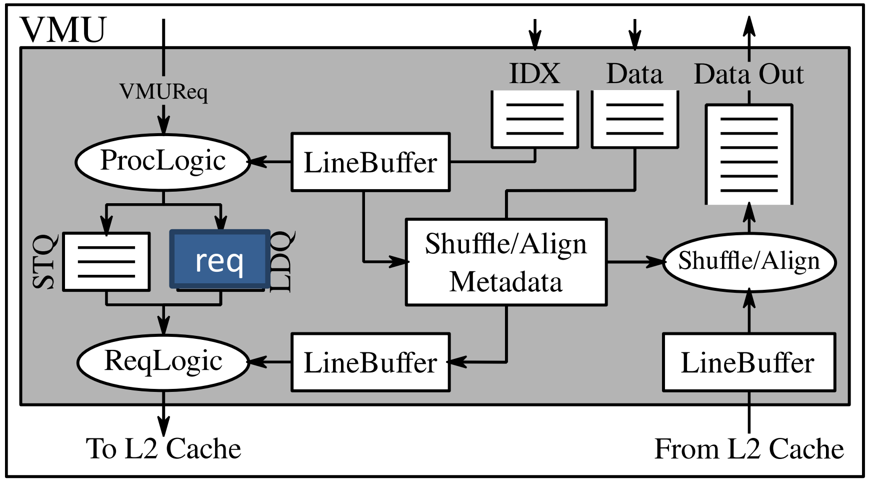
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



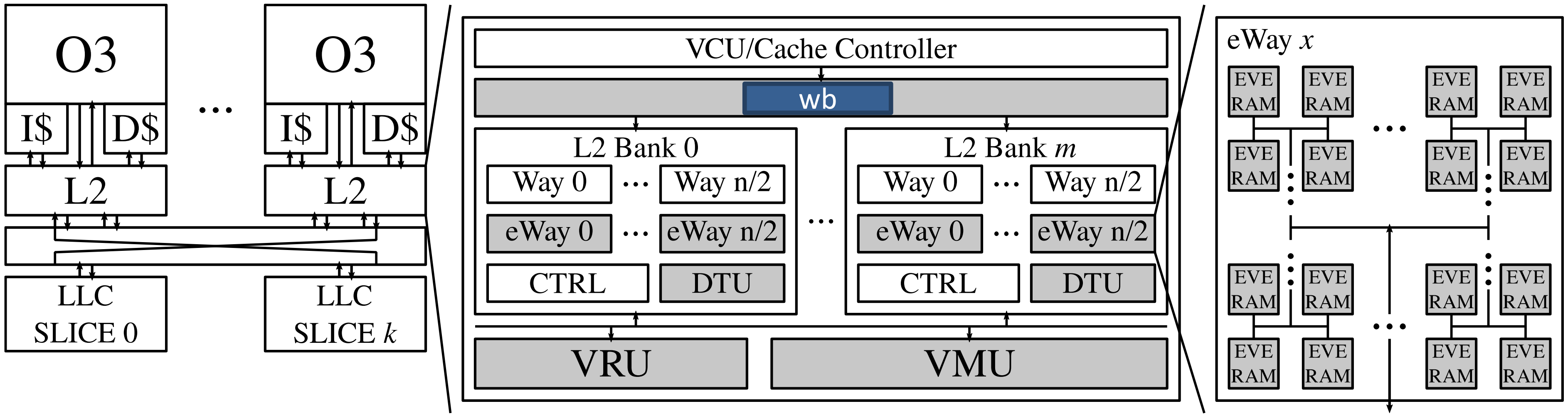
EVE MICRO-ARCHITECTURE



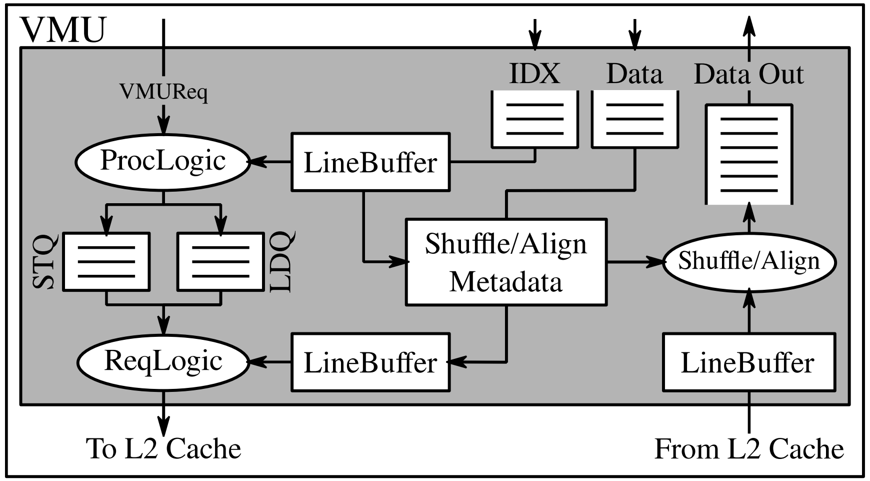
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



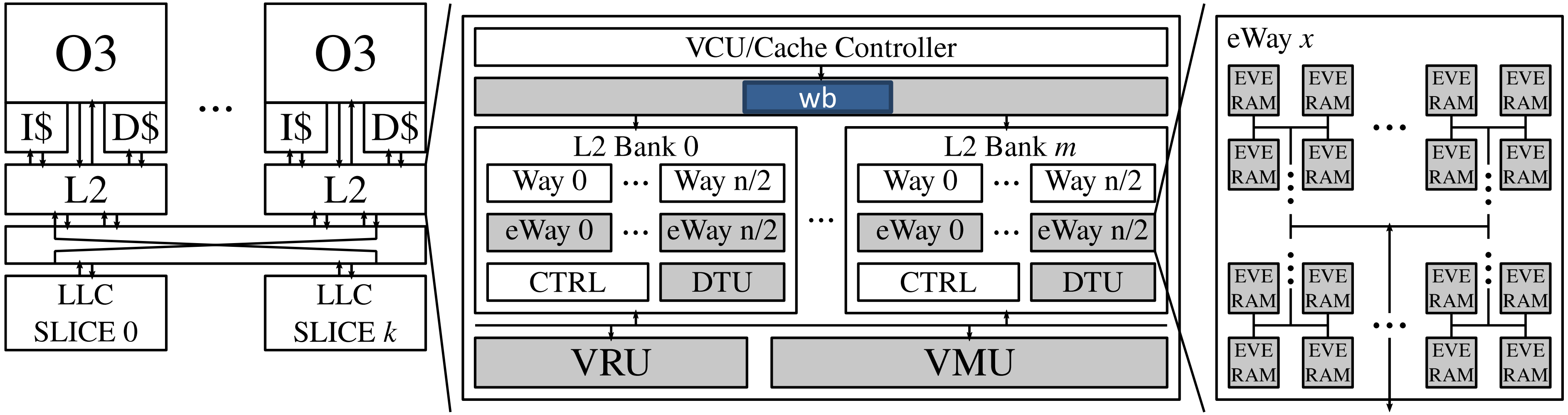
EVE MICRO-ARCHITECTURE



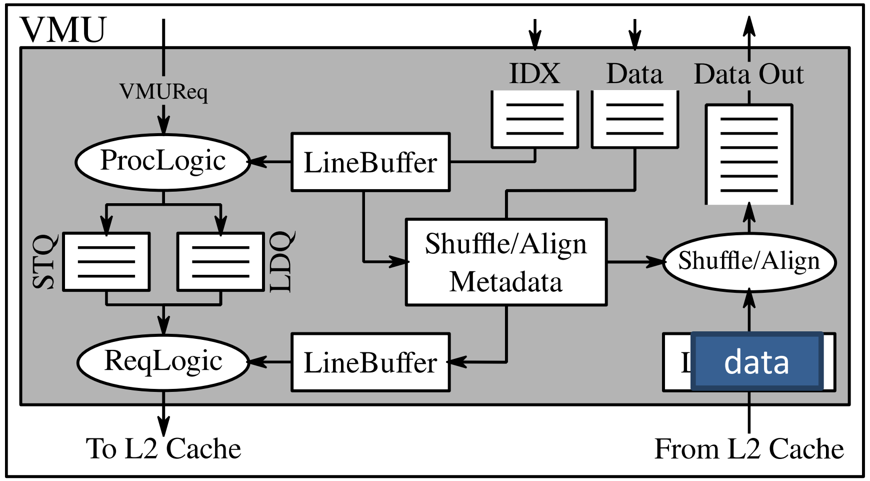
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



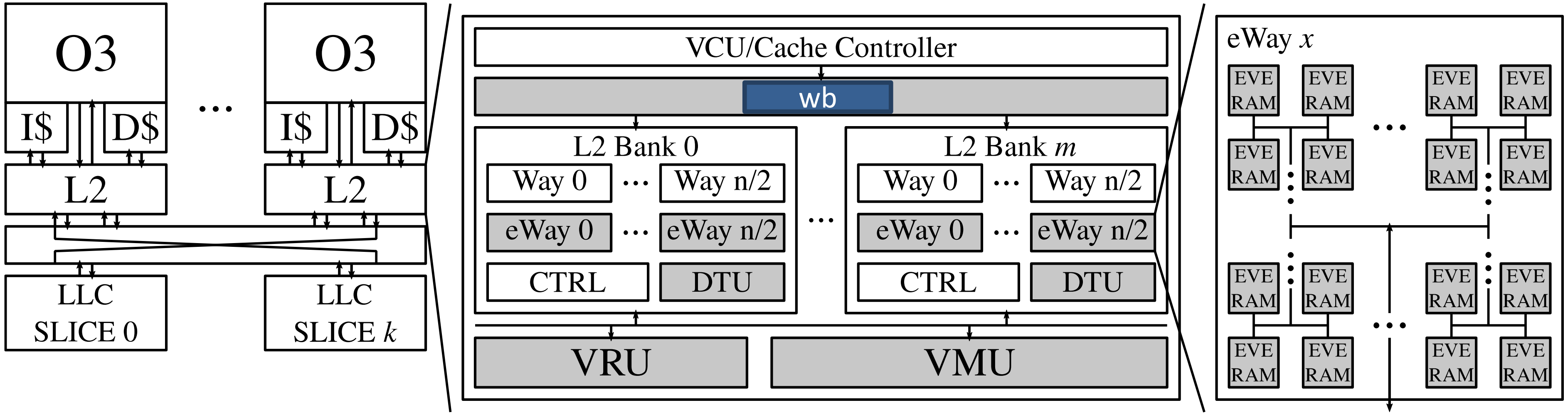
EVE Micro-Architecture



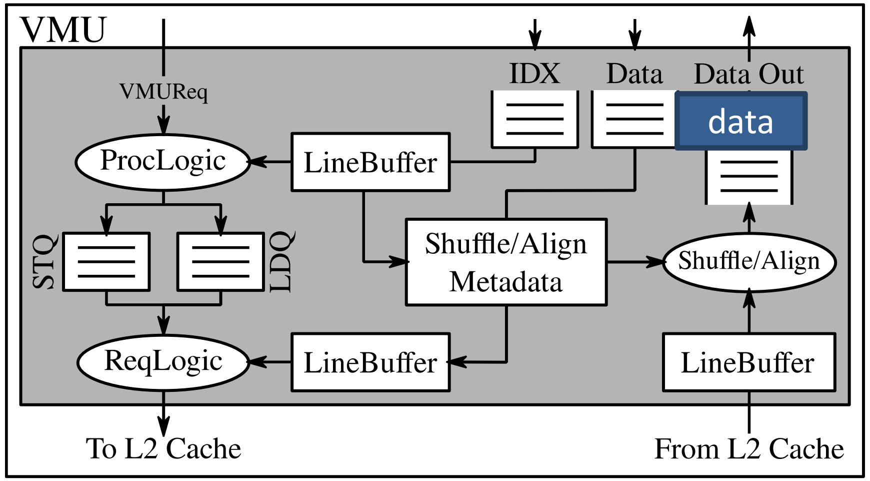
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



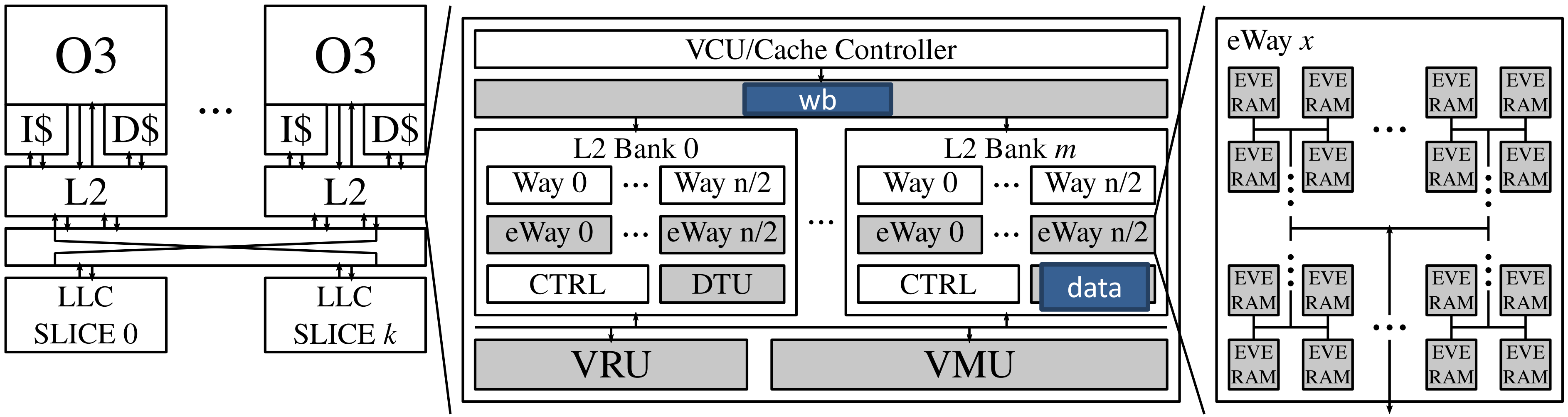
EVE Micro-Architecture



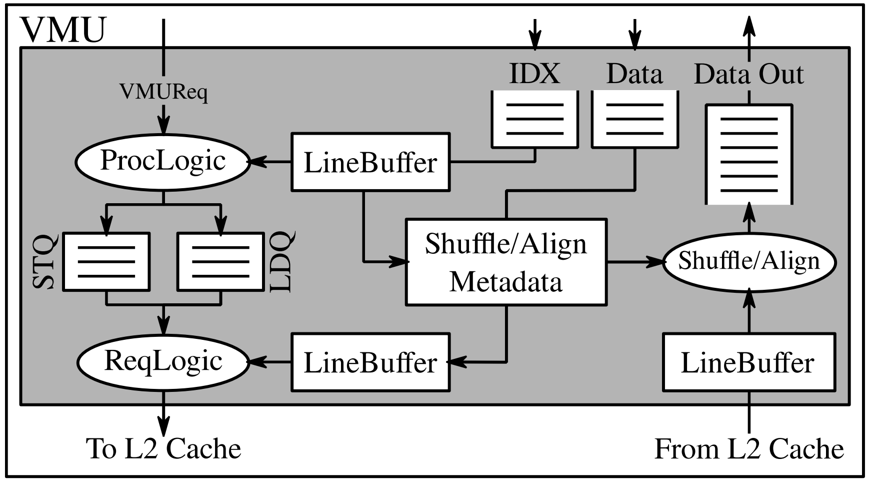
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



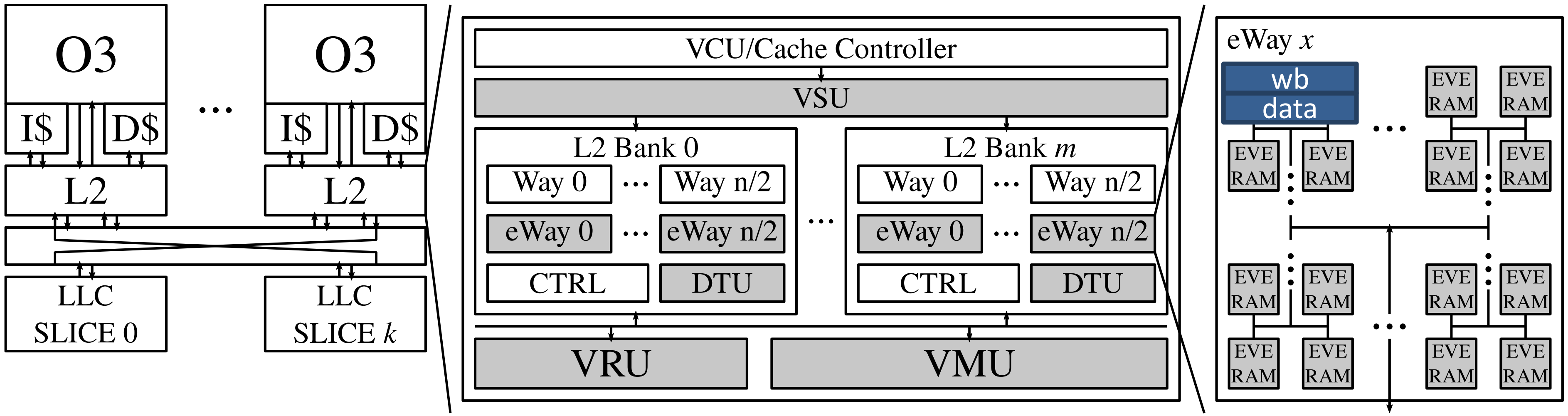
EVE Micro-Architecture



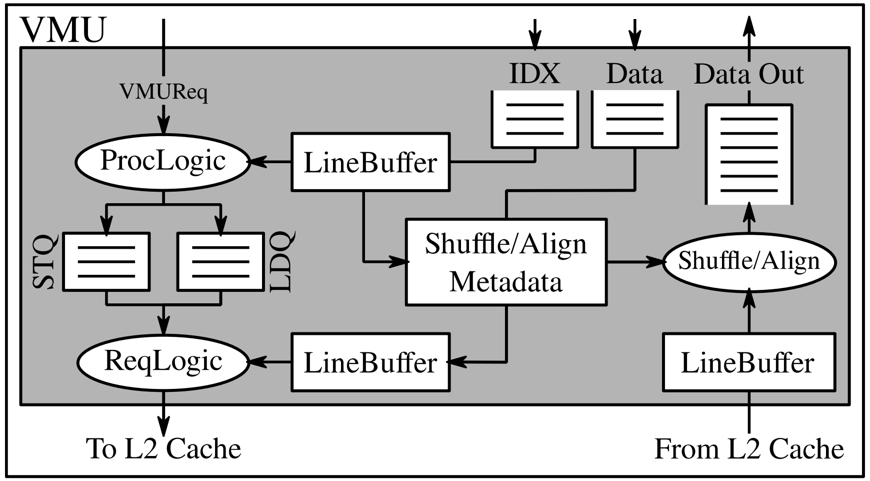
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



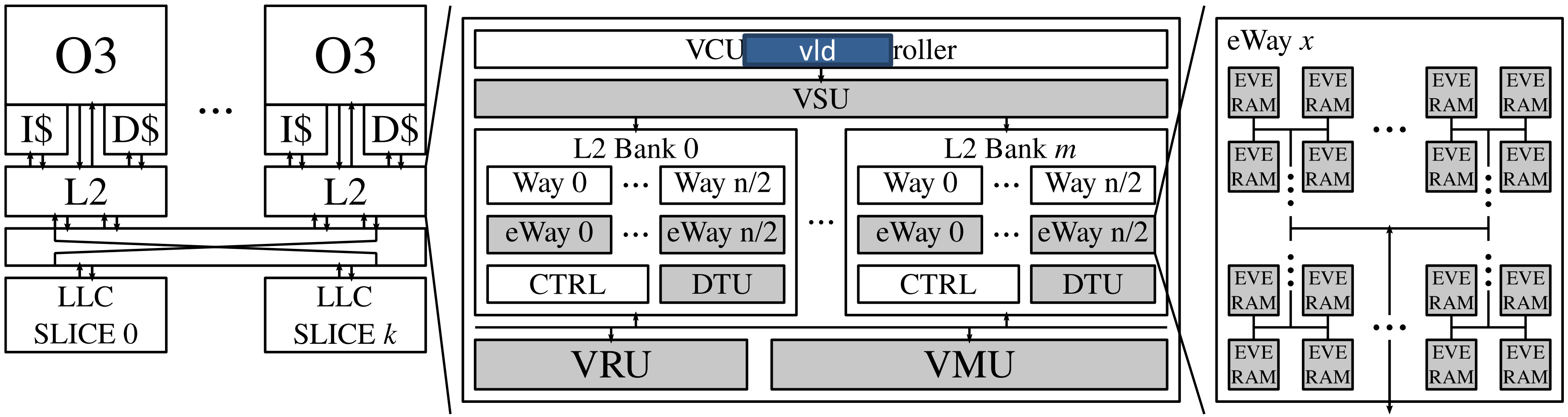
EVE Micro-Architecture



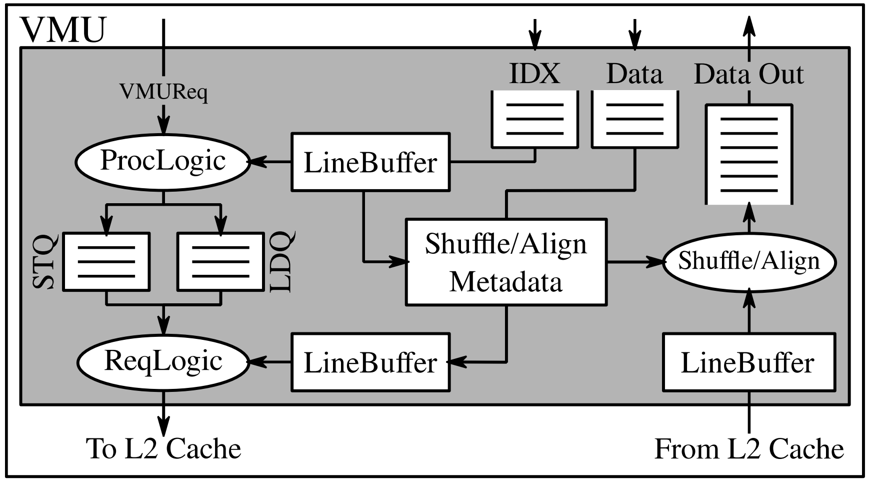
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



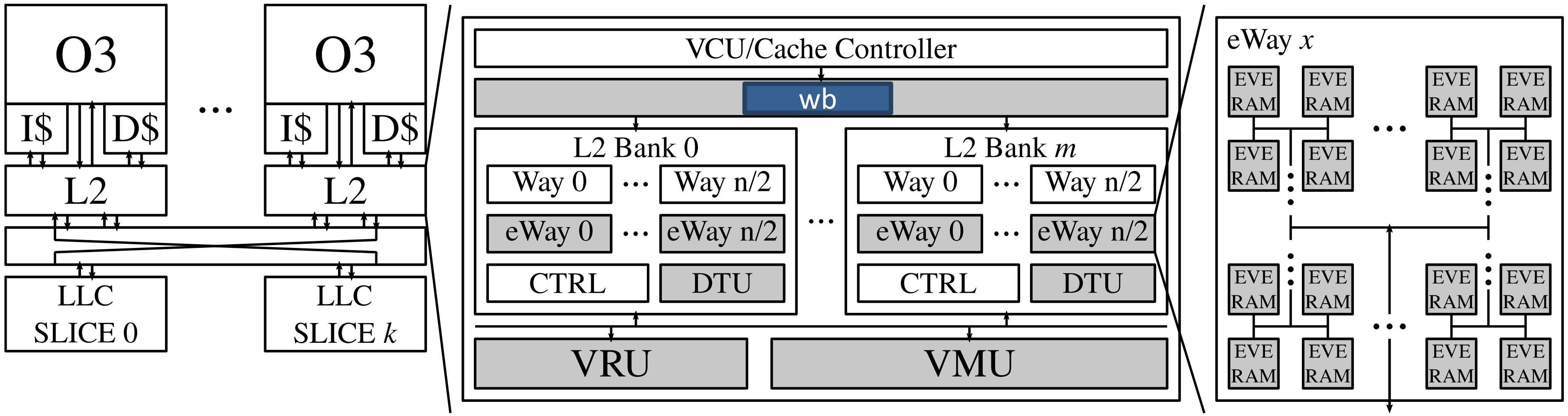
EVE MICRO-ARCHITECTURE



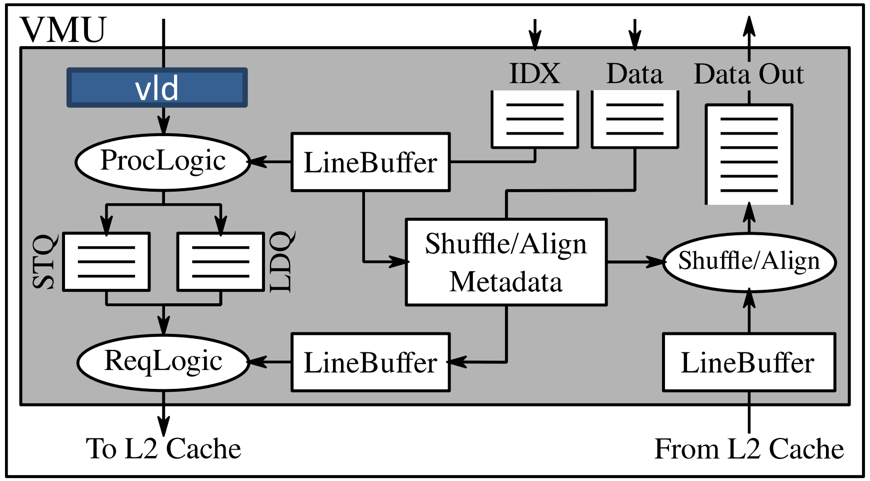
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



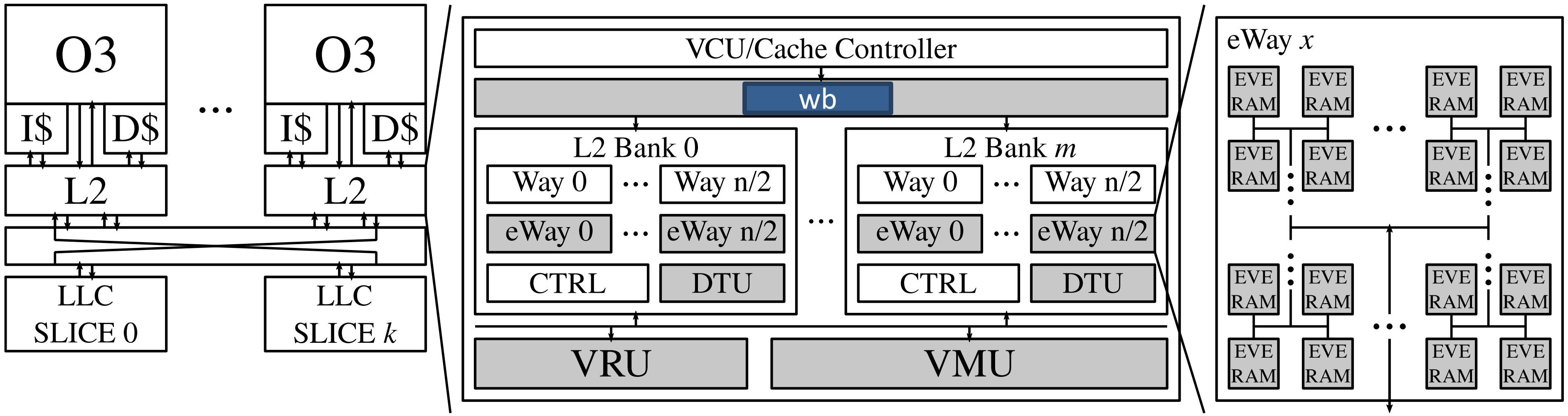
EVE Micro-Architecture



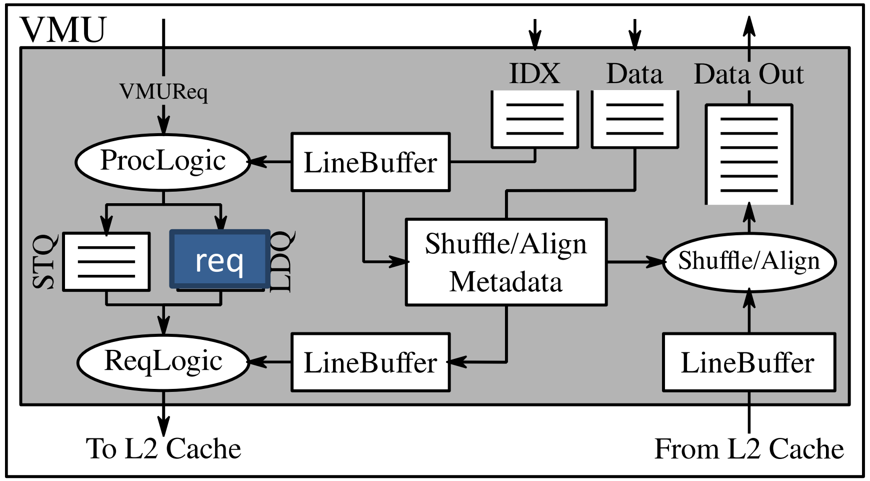
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



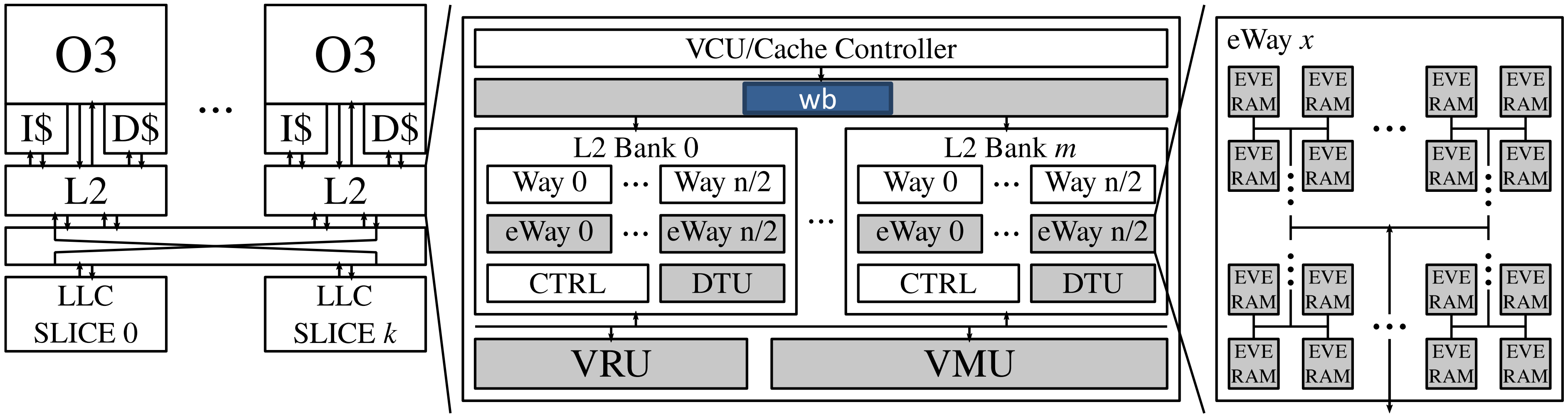
EVE Micro-Architecture



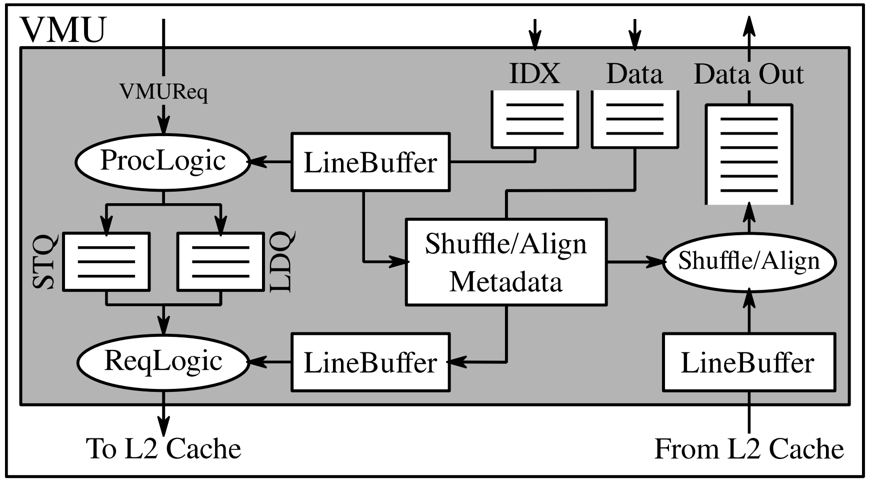
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



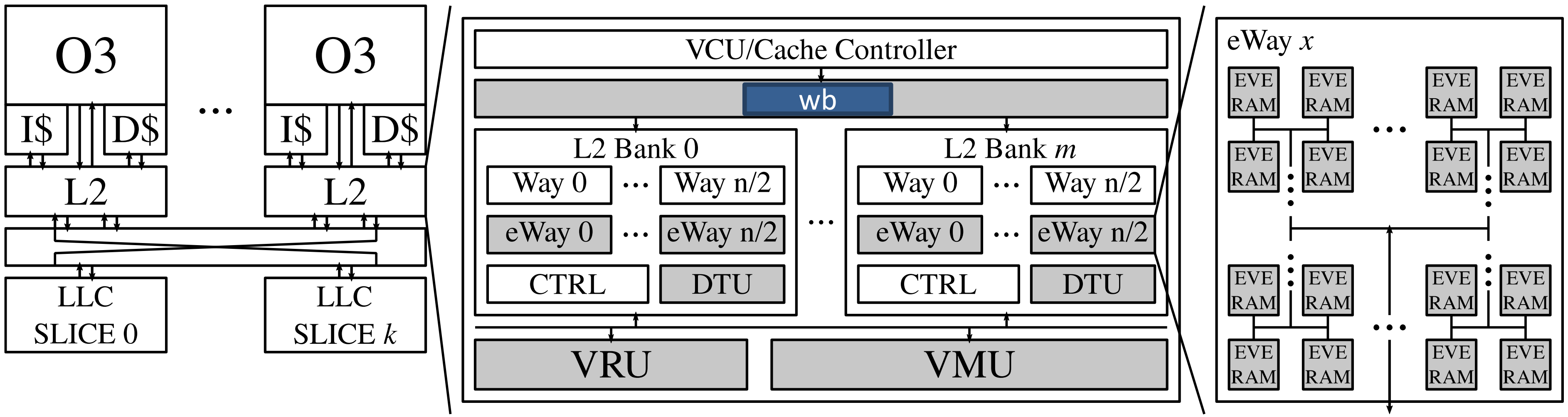
EVE Micro-Architecture



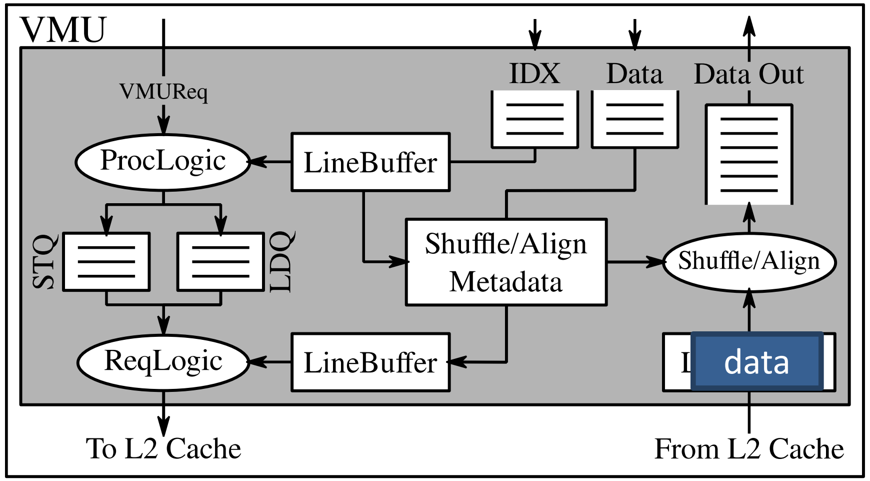
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



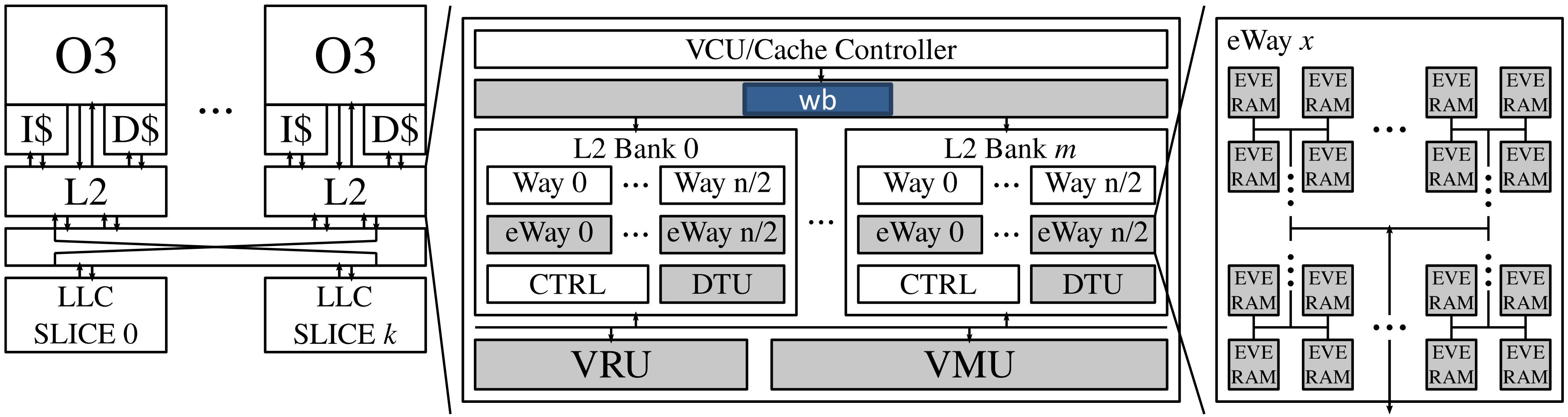
EVE Micro-Architecture



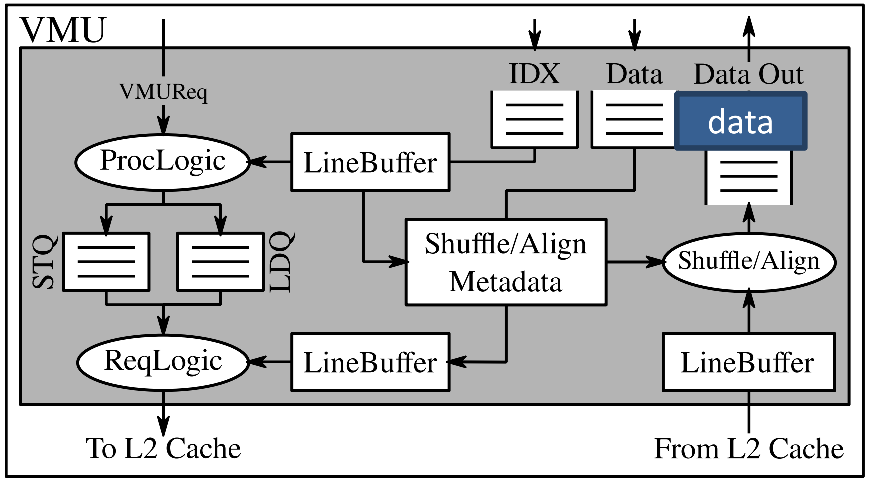
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



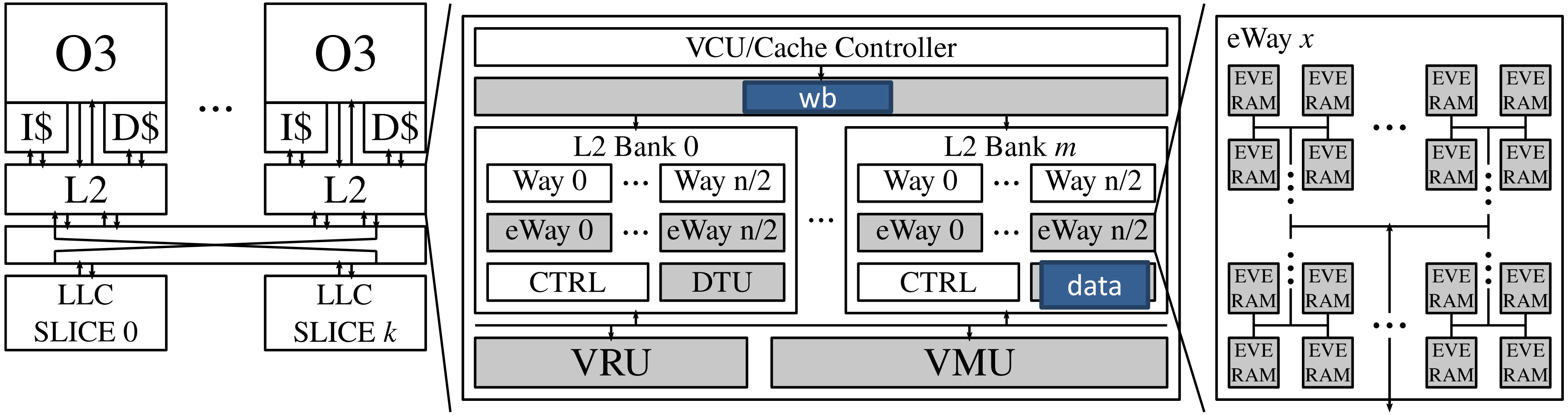
EVE Micro-Architecture



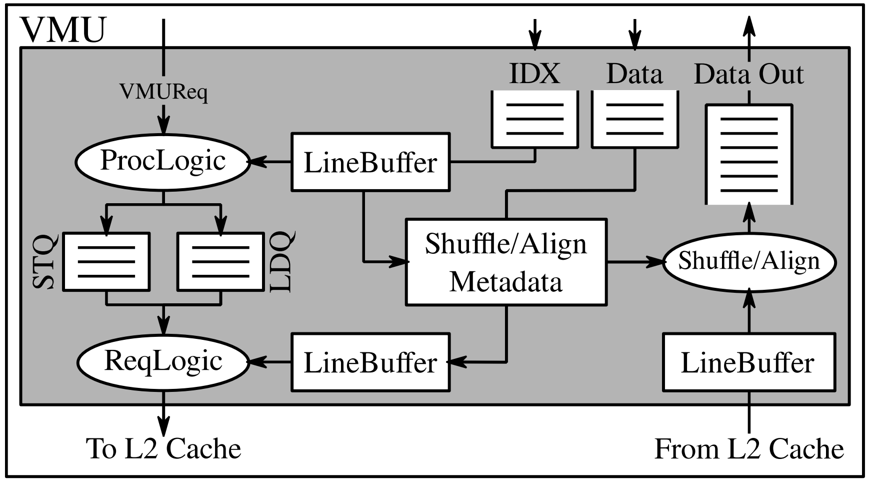
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



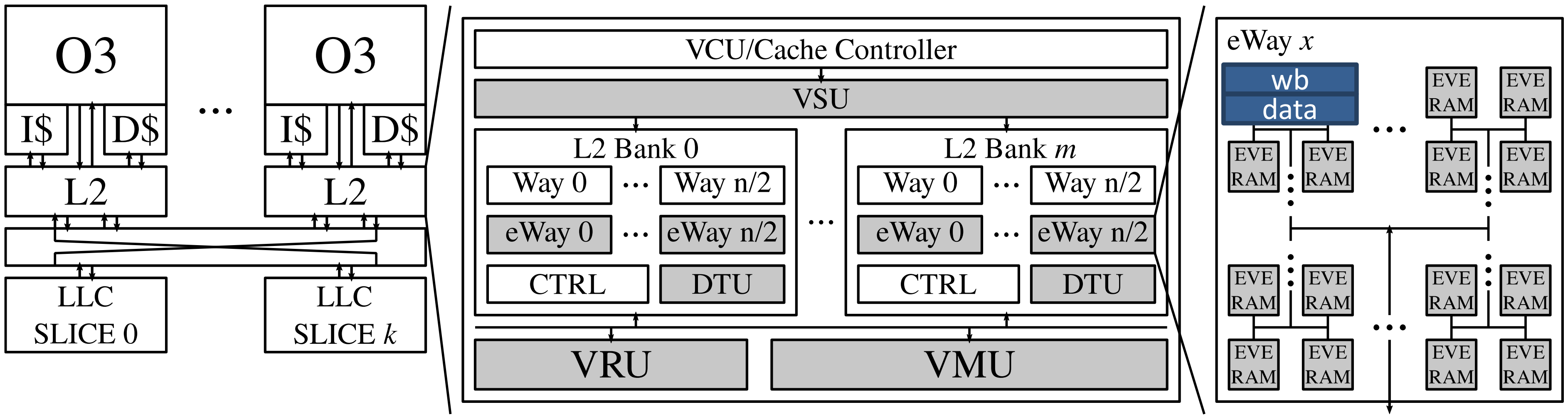
EVE Micro-Architecture



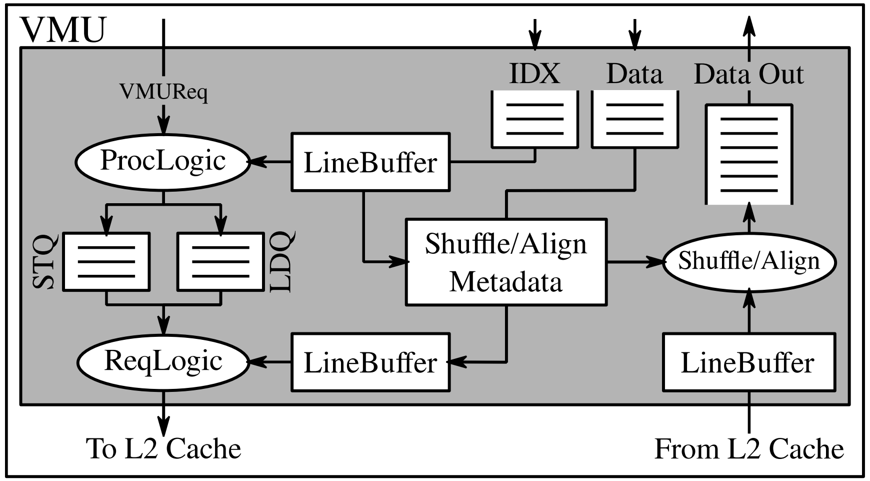
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



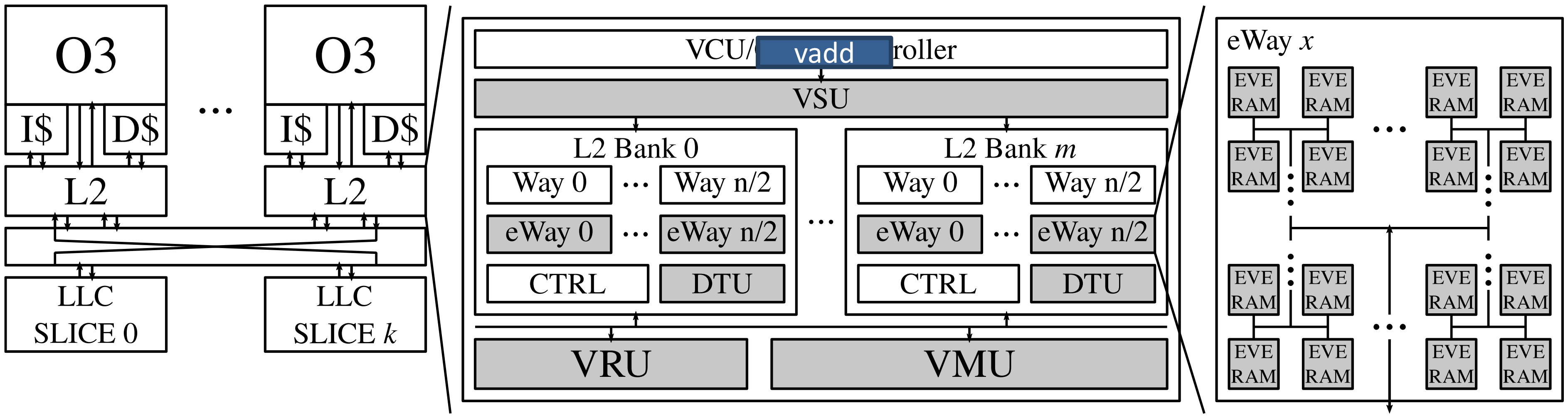
EVE Micro-Architecture



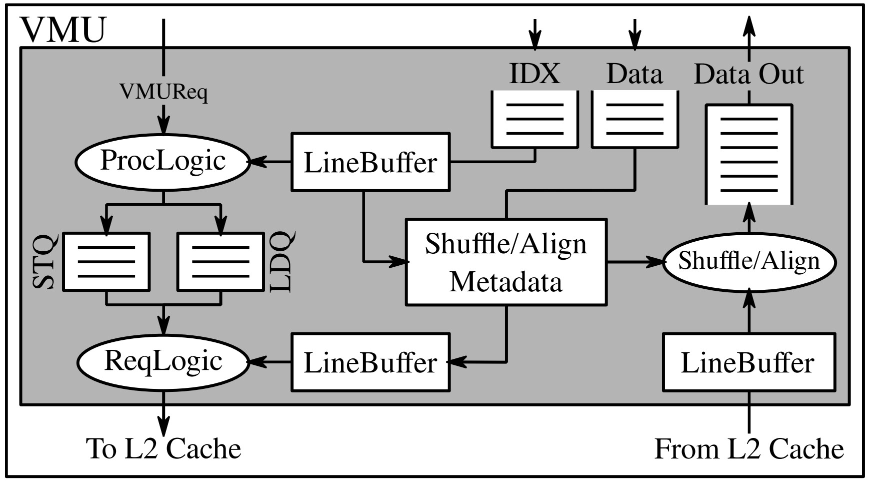
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



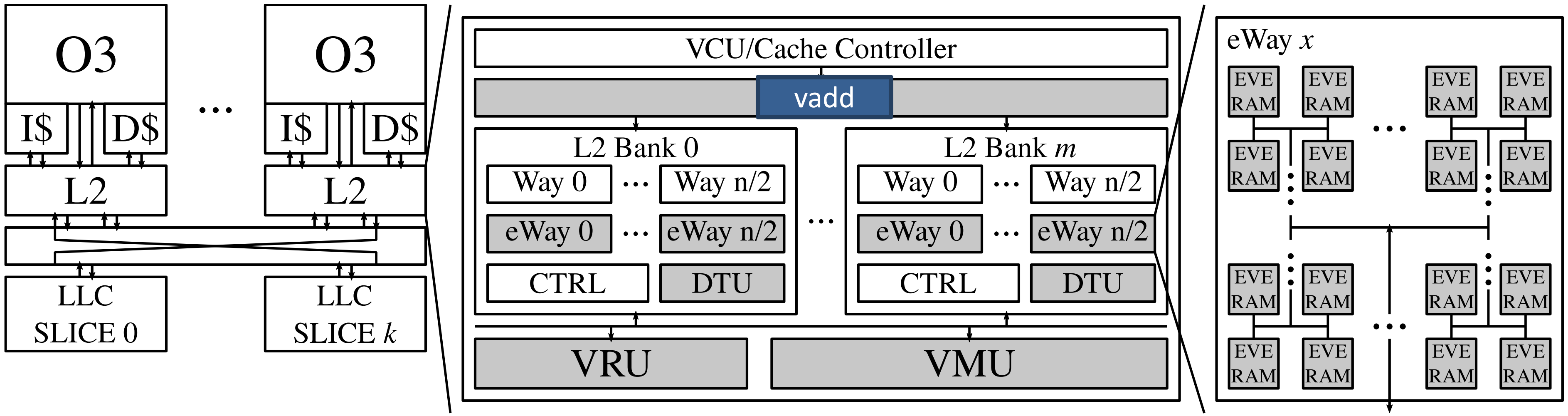
EVE MICRO-ARCHITECTURE



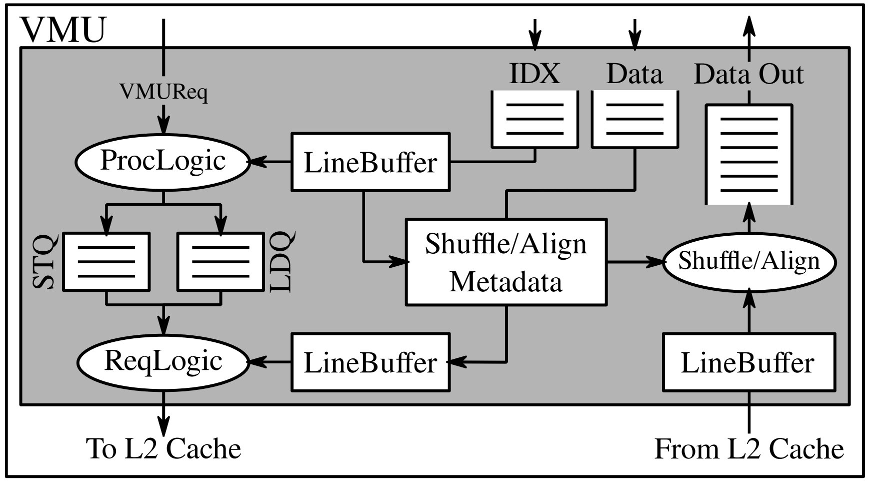
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



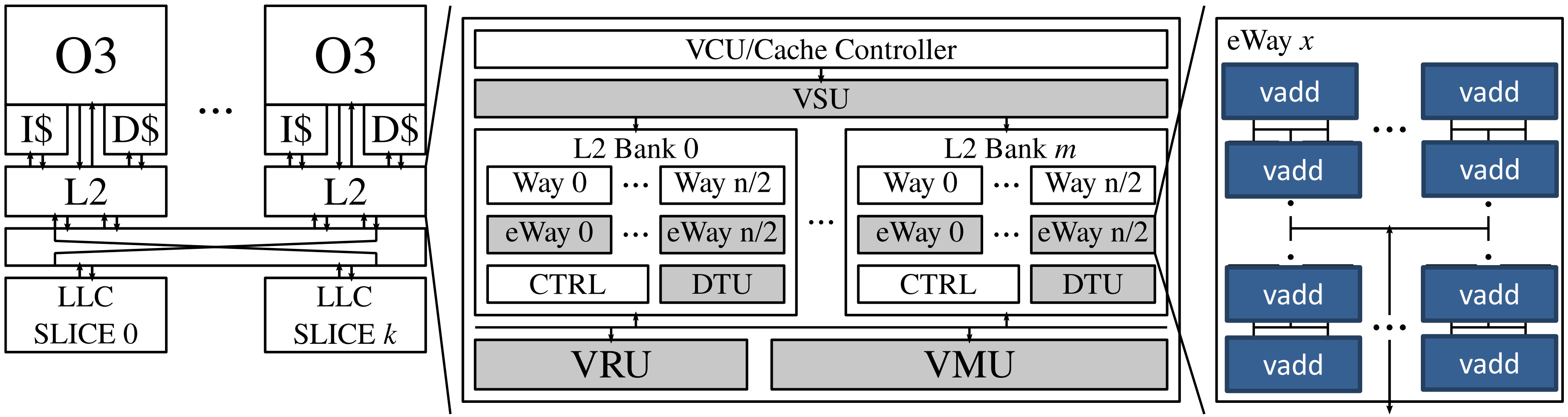
EVE Micro-Architecture



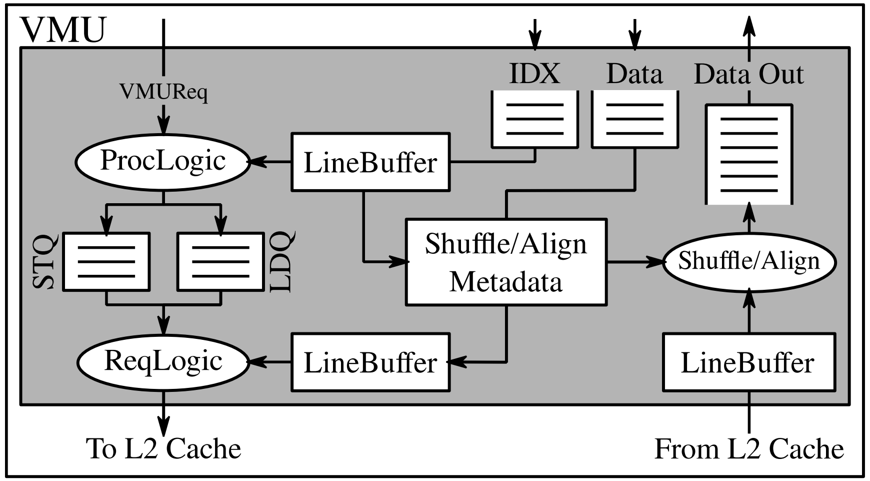
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



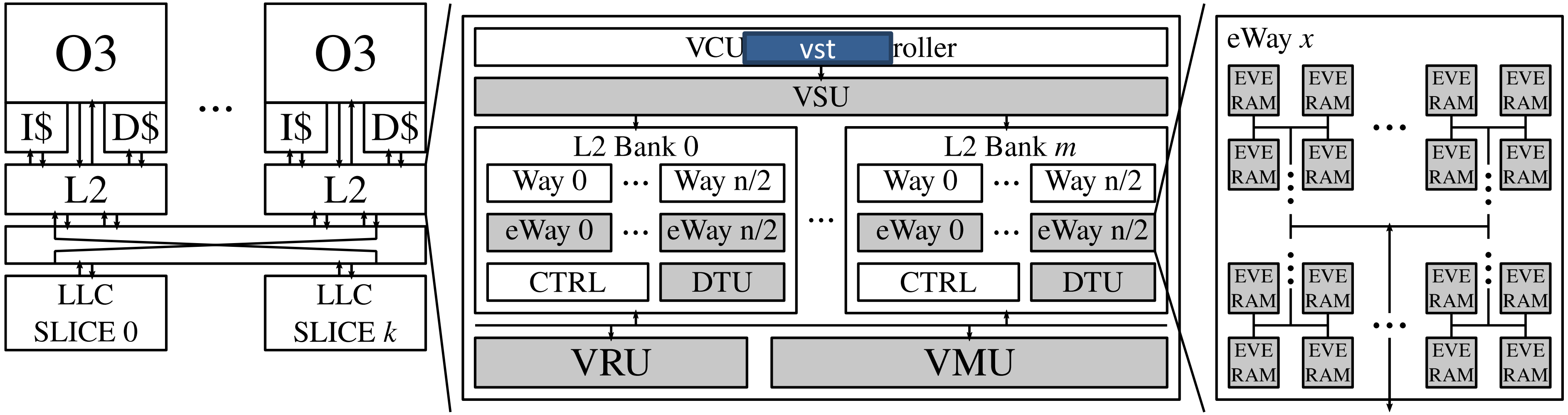
EVE Micro-Architecture



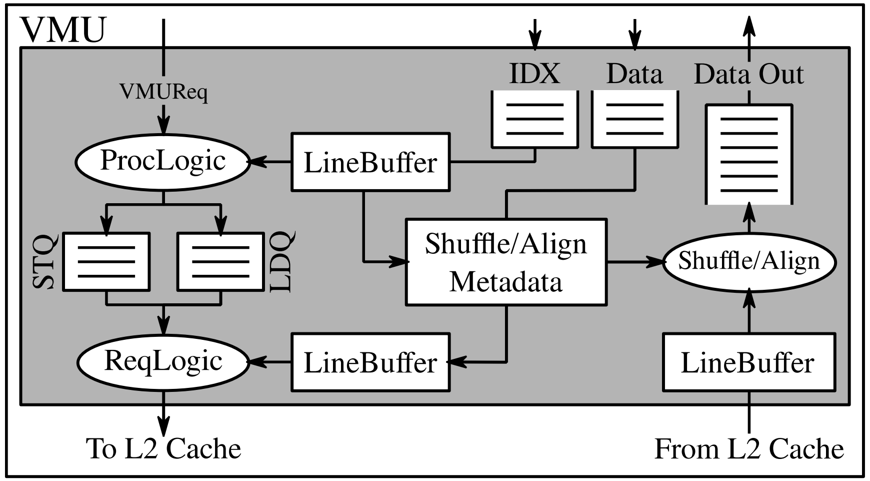
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



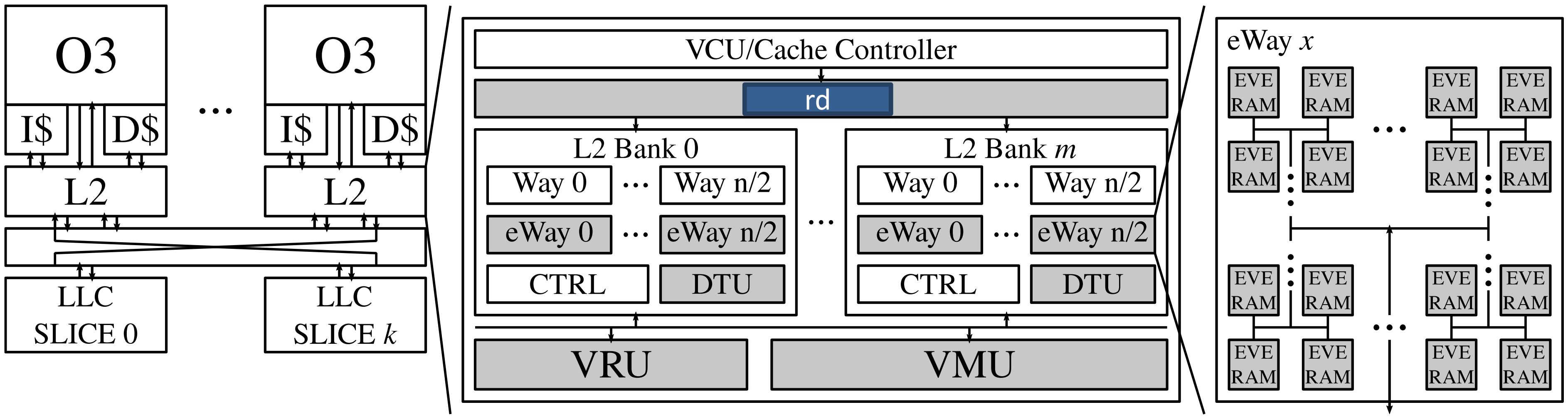
EVE Micro-Architecture



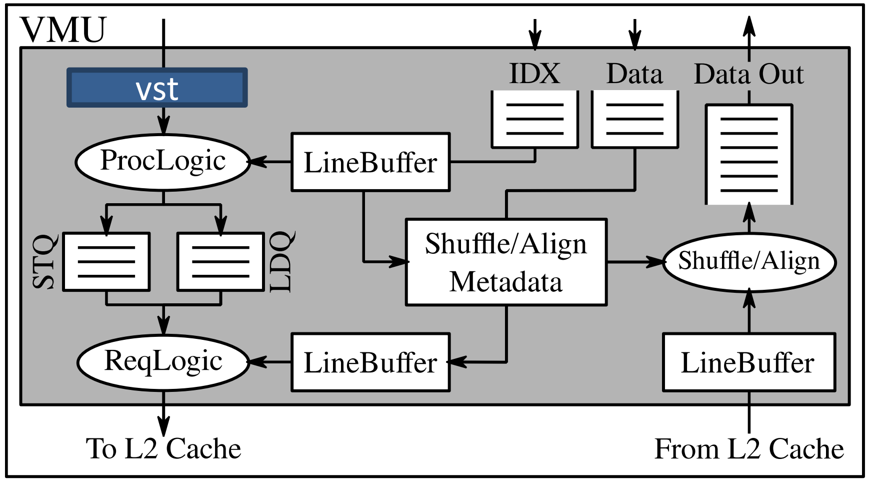
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



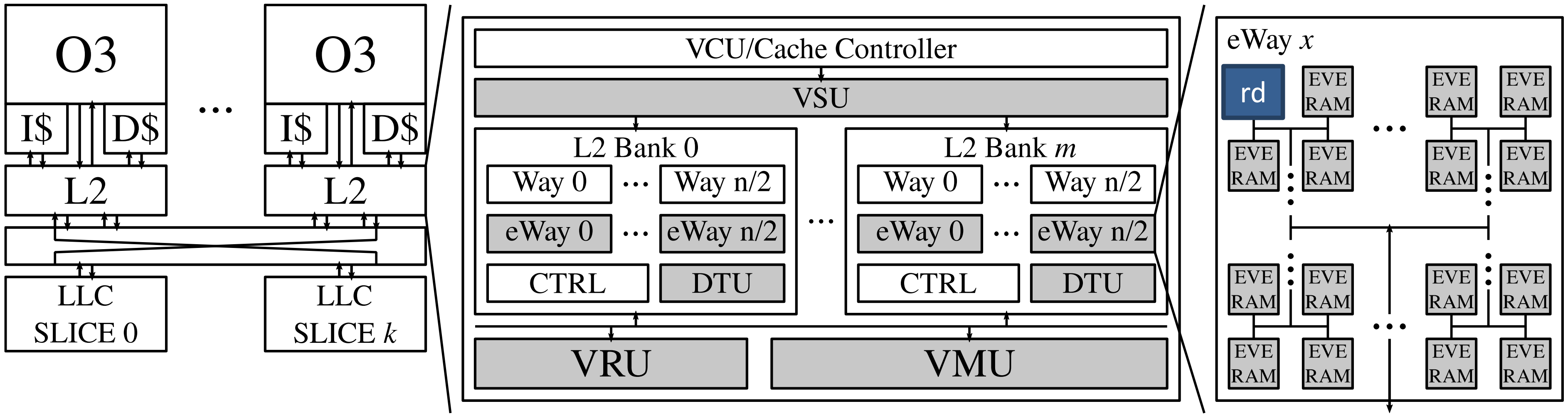
EVE Micro-Architecture



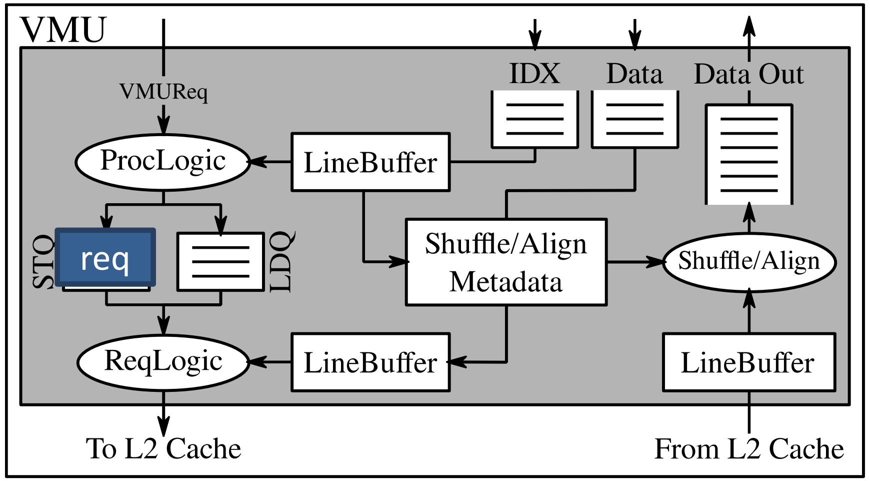
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



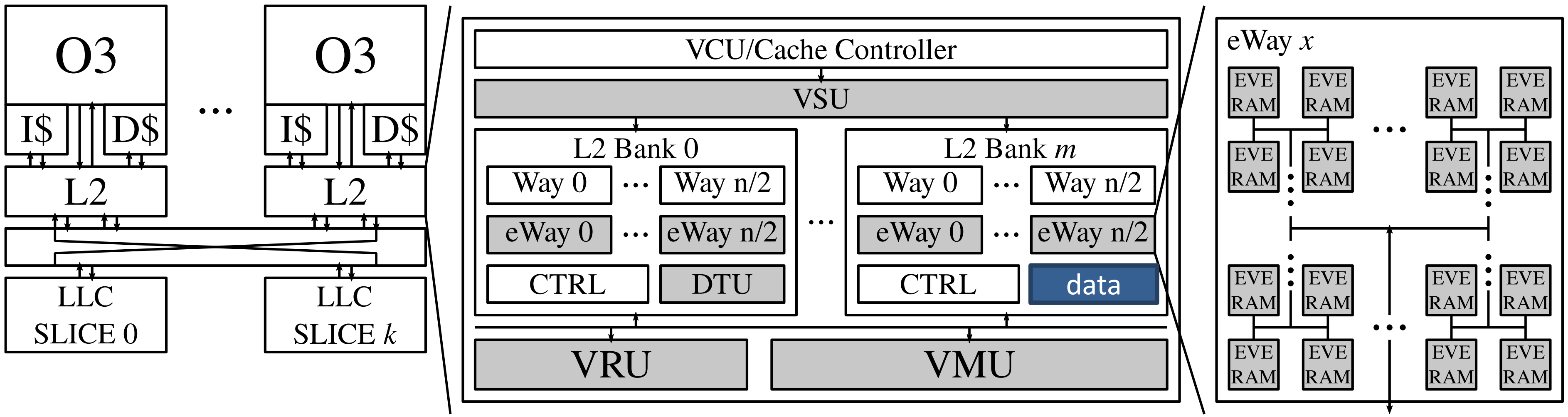
EVE Micro-Architecture



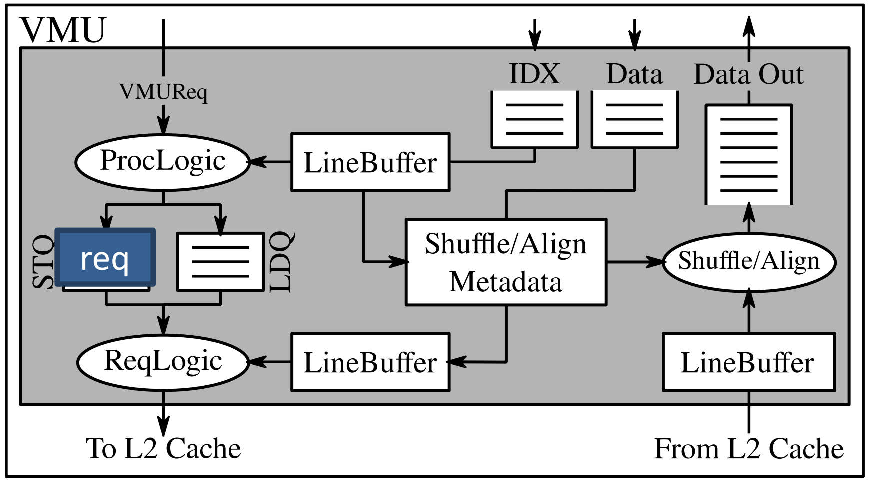
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



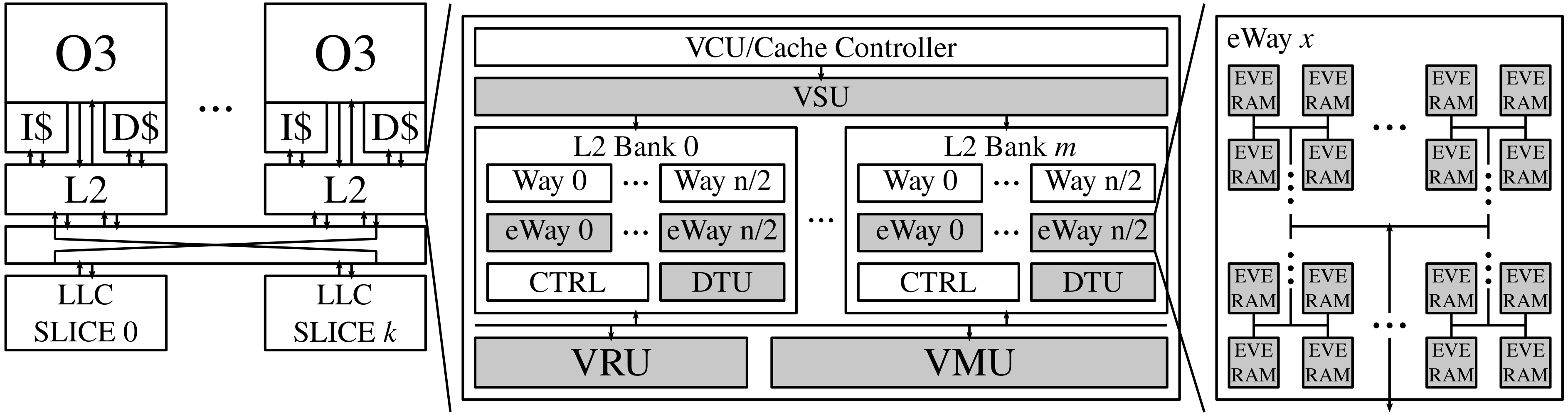
EVE Micro-Architecture



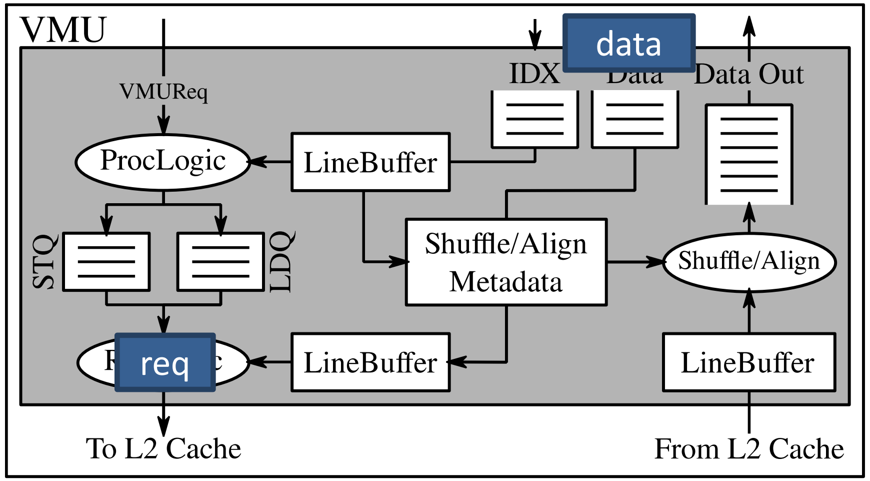
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



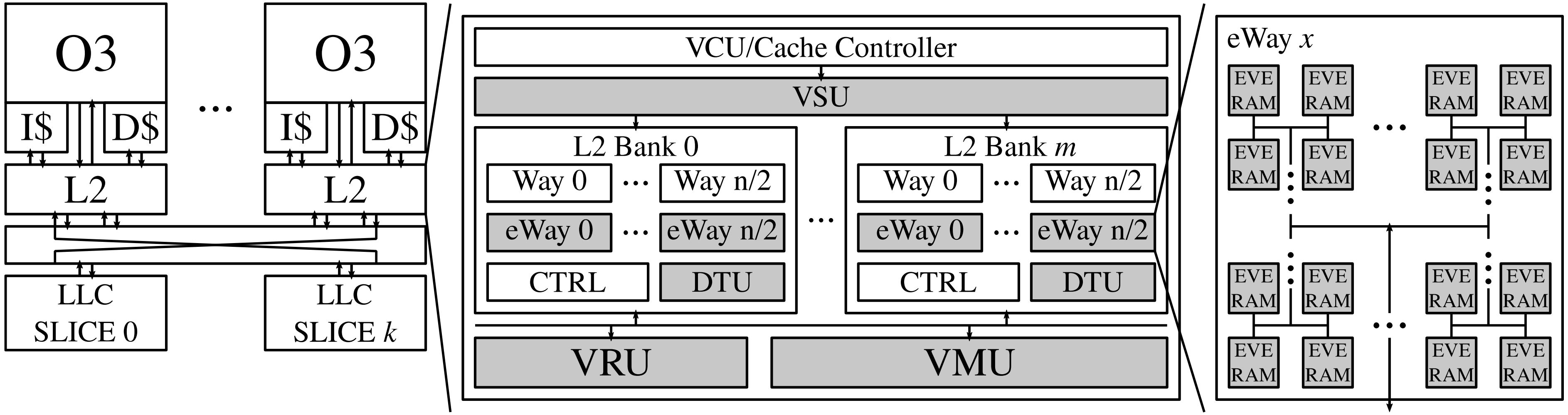
EVE Micro-Architecture



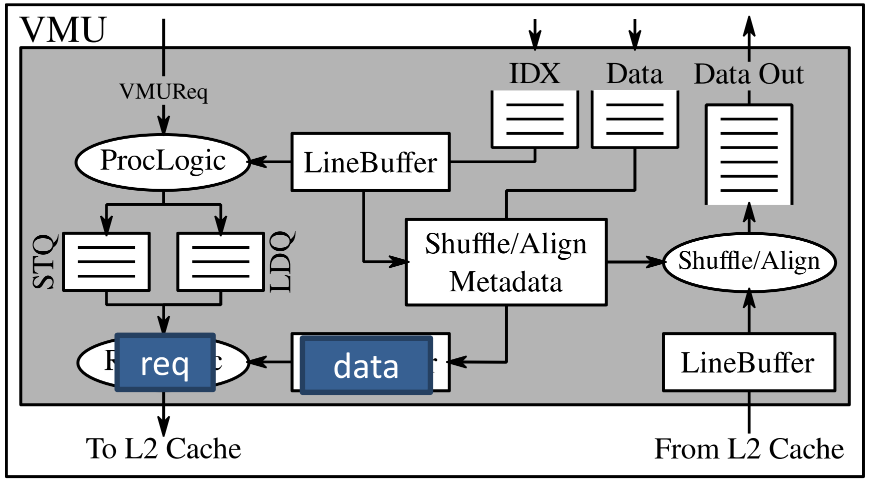
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



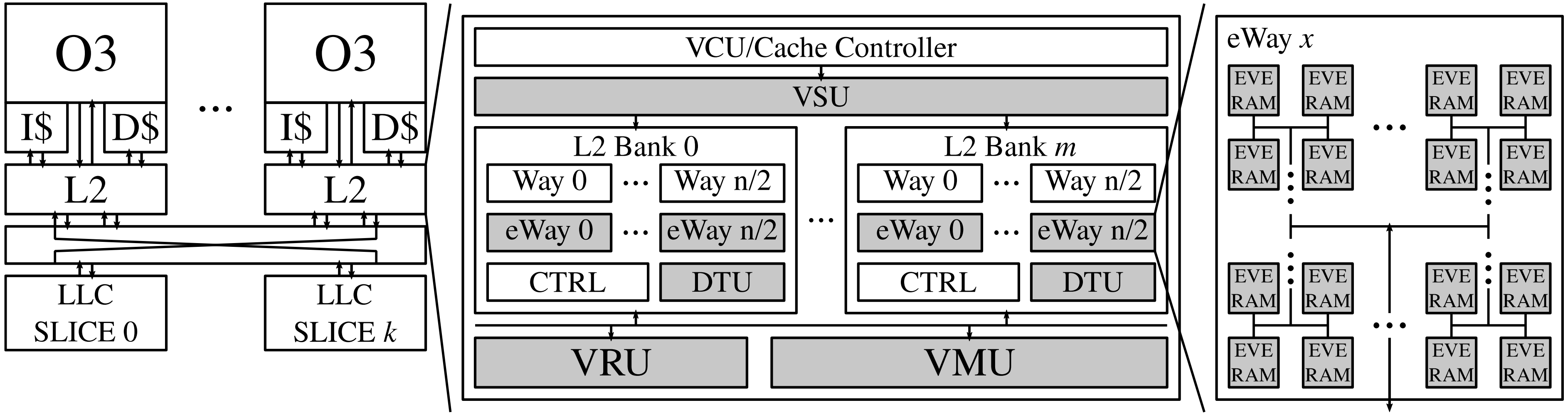
EVE Micro-Architecture



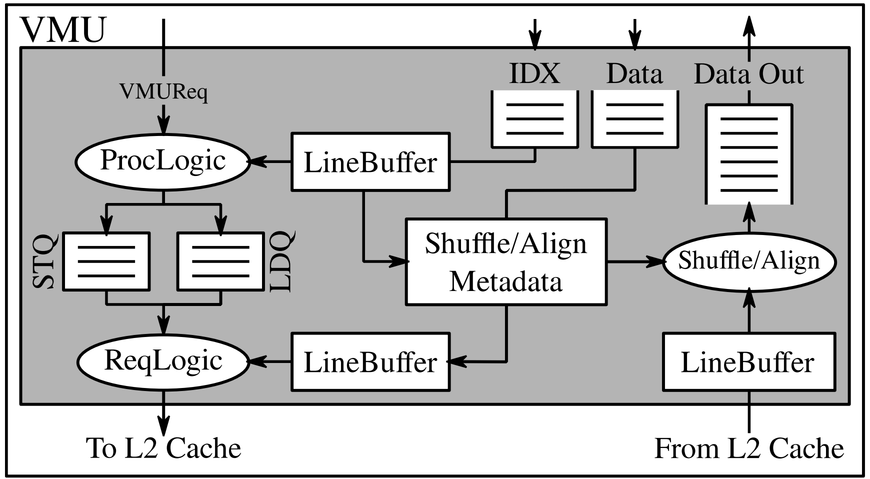
```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```



EVE Micro-Architecture

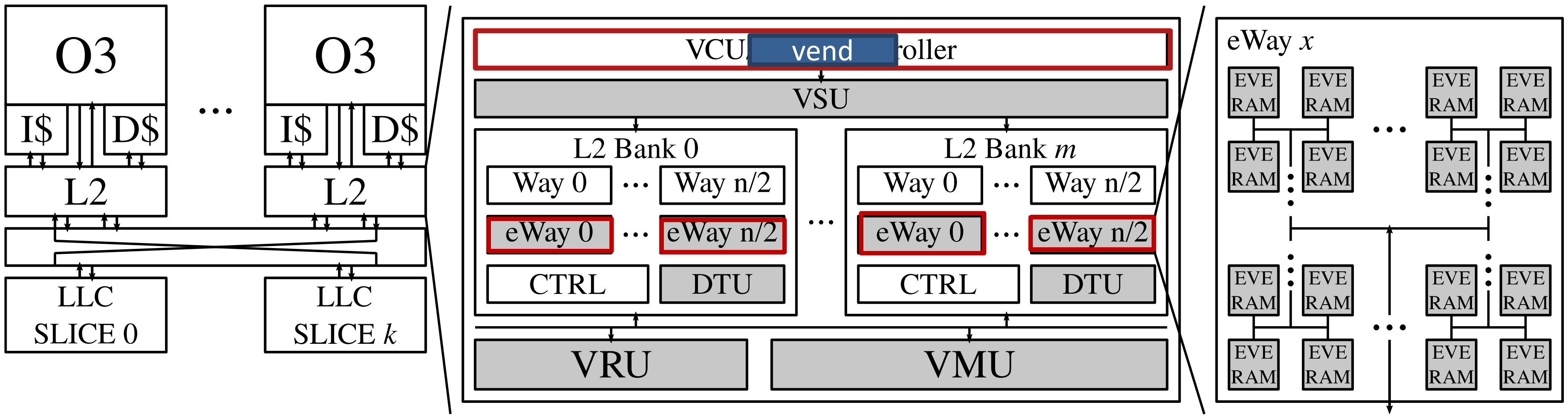


```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```

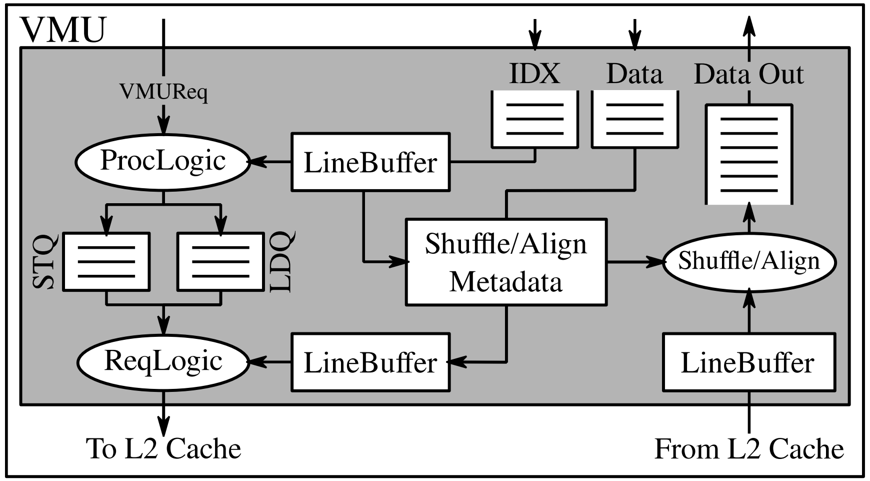


req data

EVE MICRO-ARCHITECTURE



```
void vvadd( int* c, int* a, int * b) {
  vstart();
  for (int i = 0; i < len; i += vsetvl(len)) {
    vld( v0, a + i );
    vld( v1, b + i );
    vadd.vv( v2, v0, v1 );
    vst( v2, c + i );
  }
  vend();
}
```

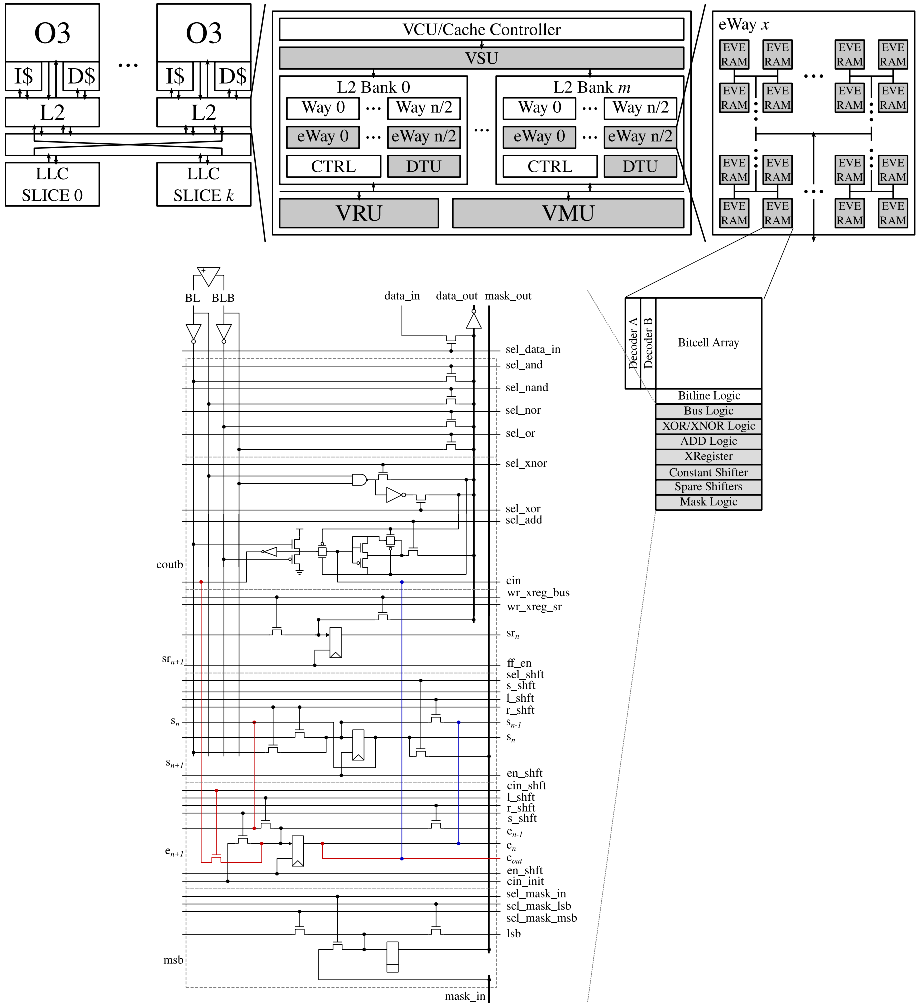


OUTLINE

Motivation

- EVE Micro-Architecture
- **EVE Bit-Hybrid Execution Paradigm**
- EVE Micro-Programming & Circuits
- EVE Evaluation

Conclusion



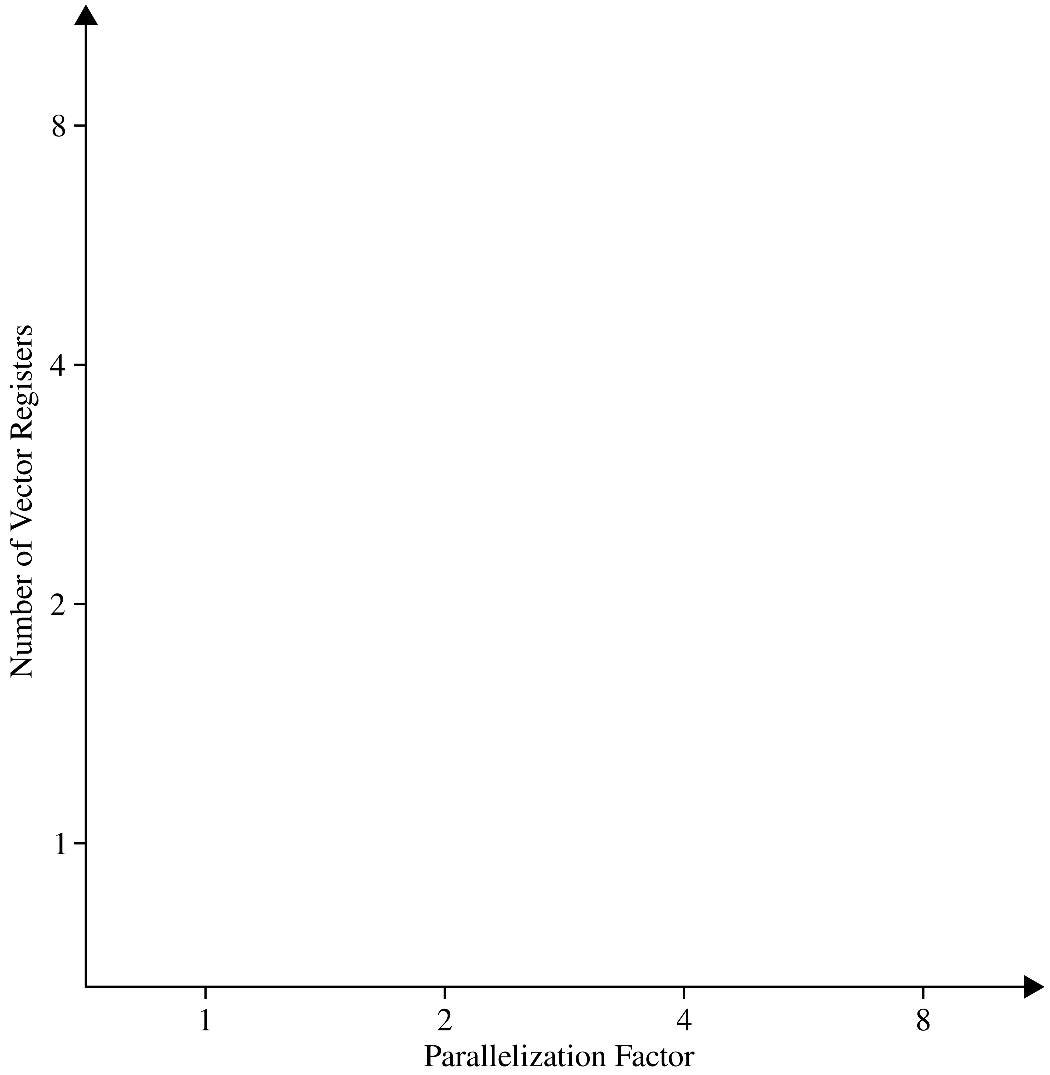
EVE Bit-Hybrid Execution Paradigm

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



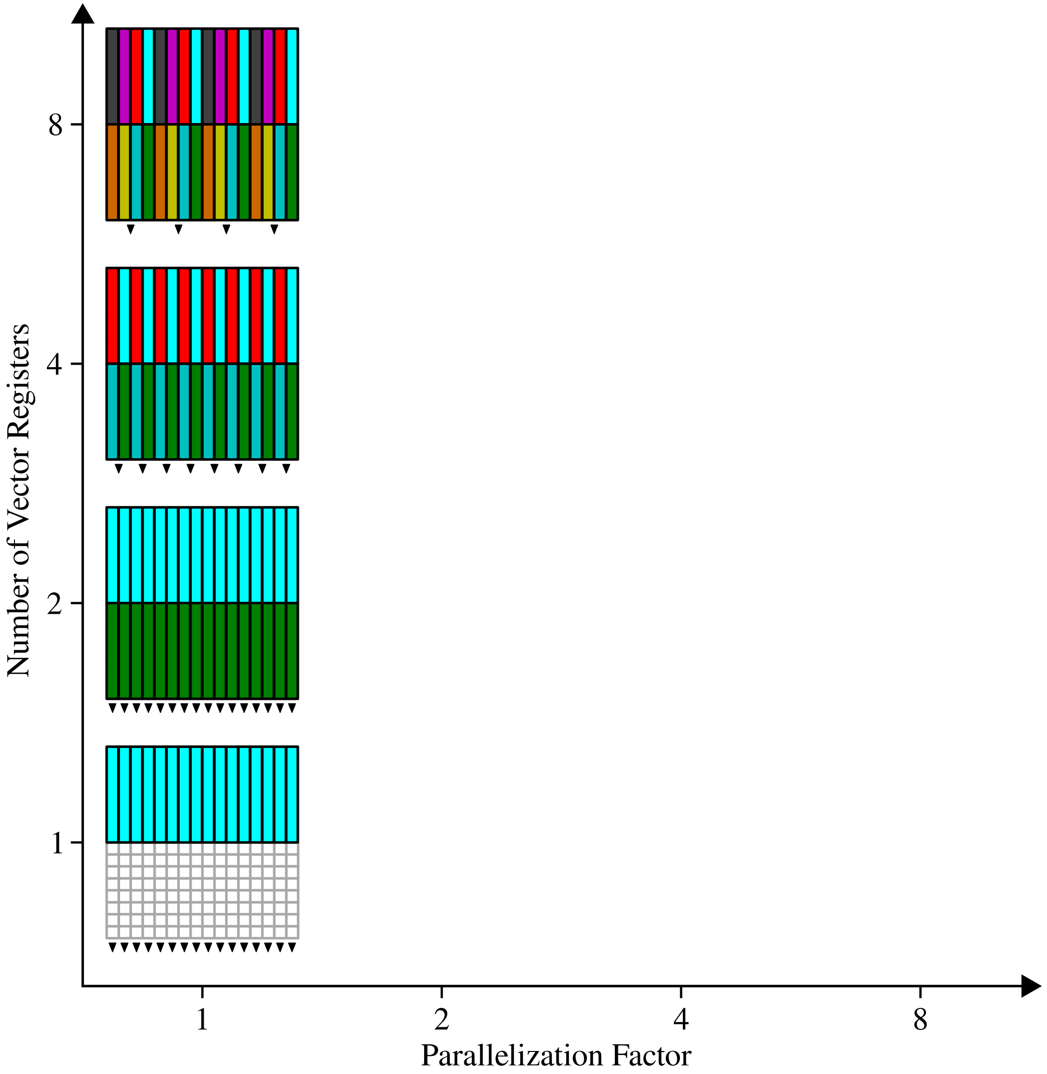
EVE Bit-Hybrid Execution Paradigm

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



Balanced Utilization:

Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

Column Under-Utilization:

Reducing the number of in-situ ALU in-favor of storage in the SRAM array

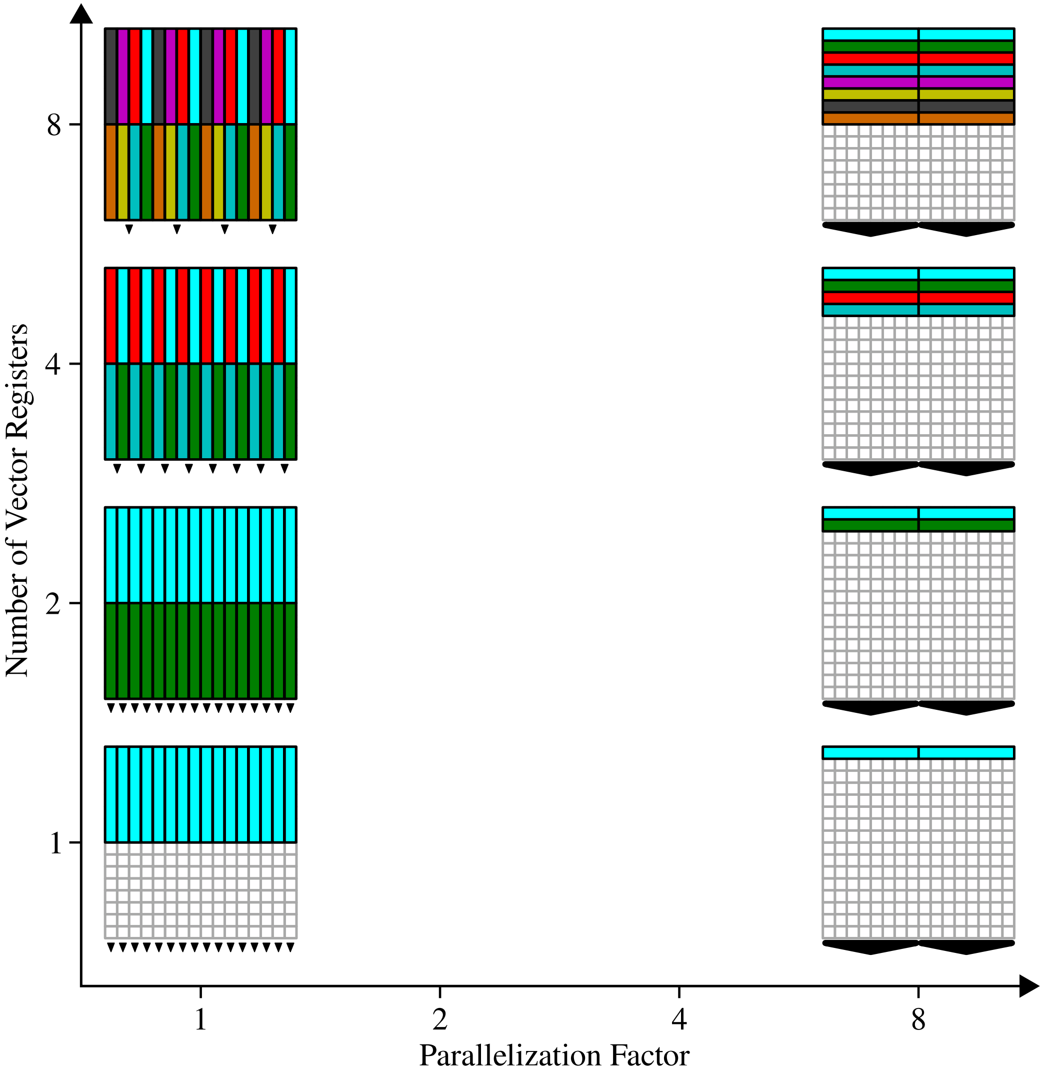
EVE Bit-Hybrid Execution Paradigm

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



Balanced Utilization:
Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

Column Under-Utilization:
Reducing the number of in-situ ALU in-favor of storage in the SRAM array

Row Under-Utilization:
Under-utilized bit-cells in the SRAM array

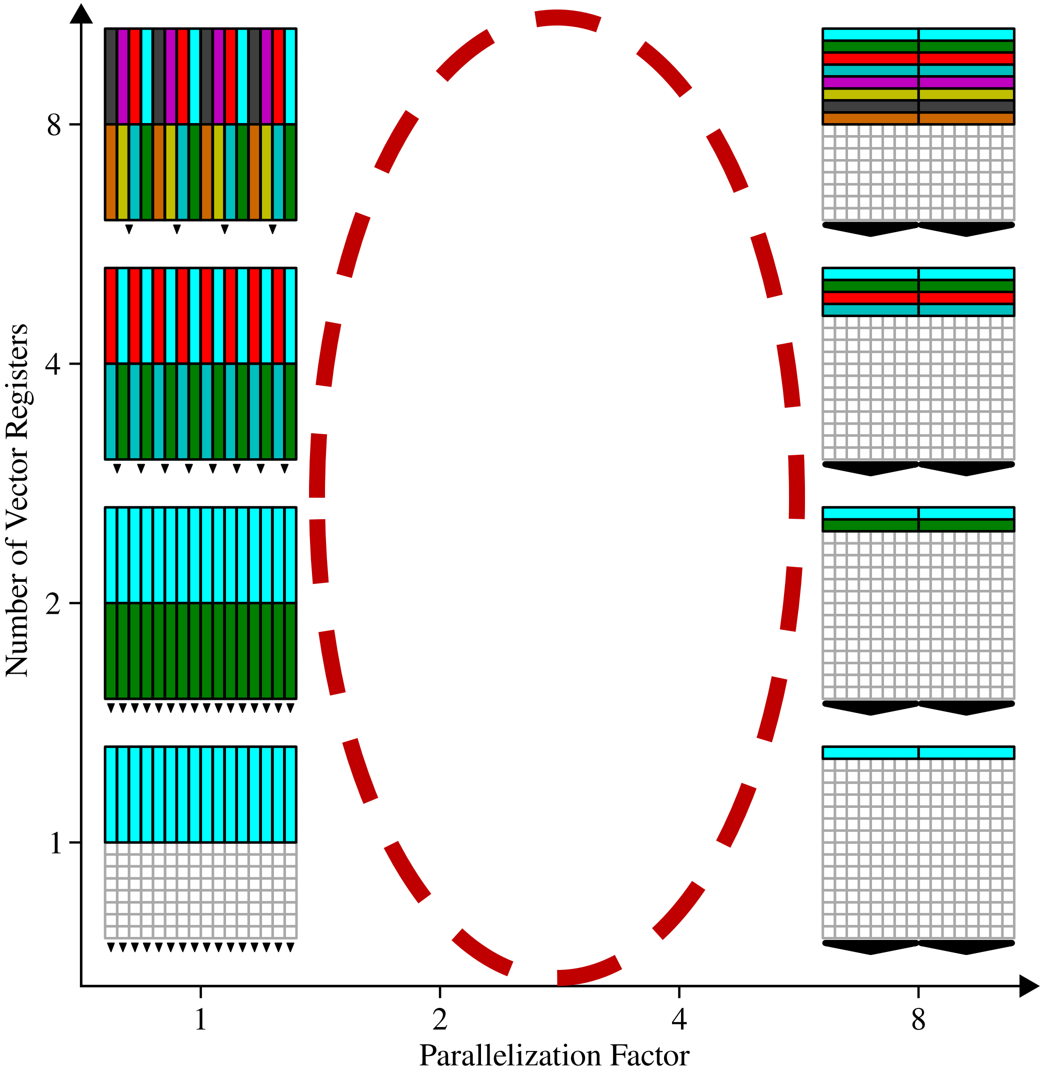
EVE Bit-Hybrid Execution Paradigm

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



Balanced Utilization:
Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

Column Under-Utilization:
Reducing the number of in-situ ALU in-favor of storage in the SRAM array

Row Under-Utilization:
Under-utilized bit-cells in the SRAM array

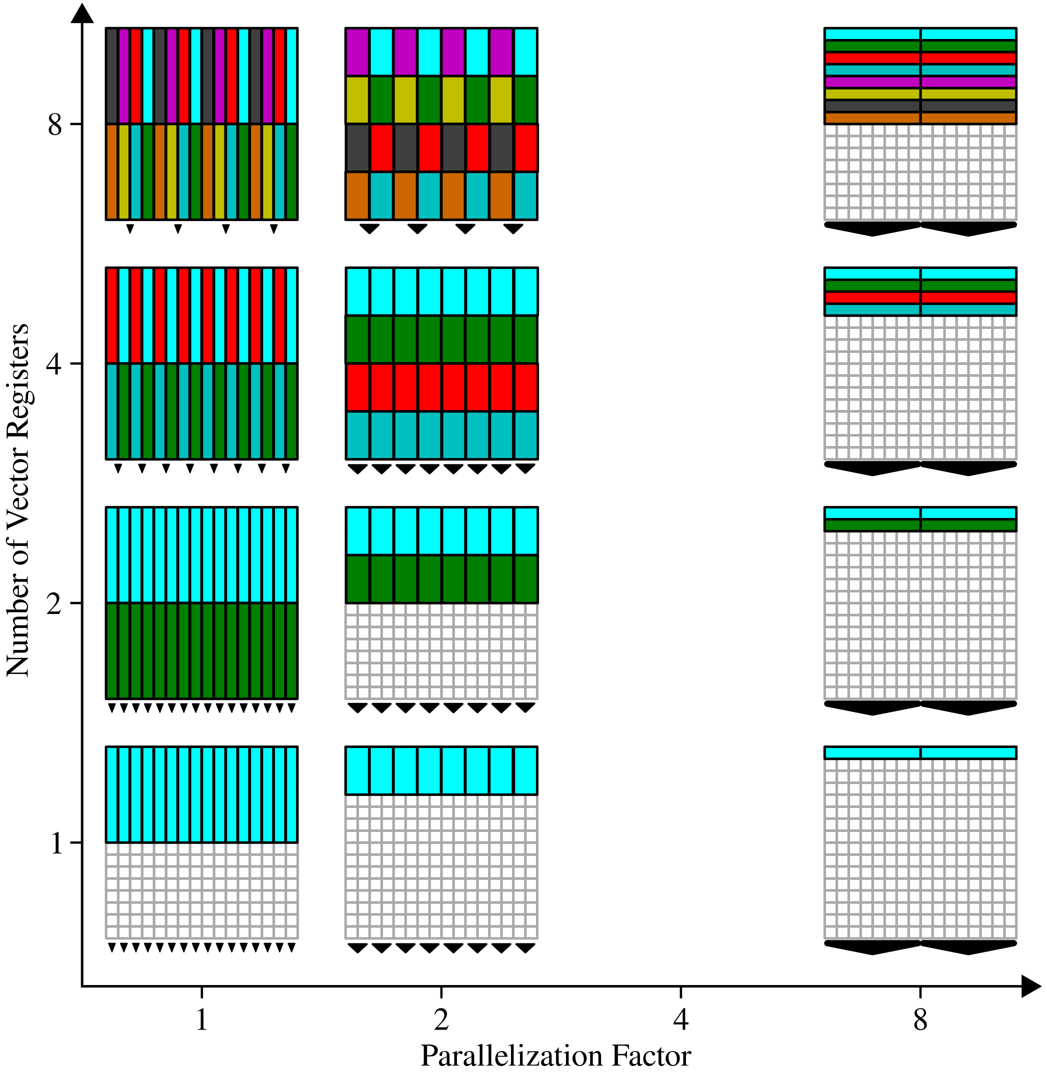
EVE Bit-Hybrid Execution Paradigm

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



Balanced Utilization:
Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

Column Under-Utilization:
Reducing the number of in-situ ALU in-favor of storage in the SRAM array

Row Under-Utilization:
Under-utilized bit-cells in the SRAM array

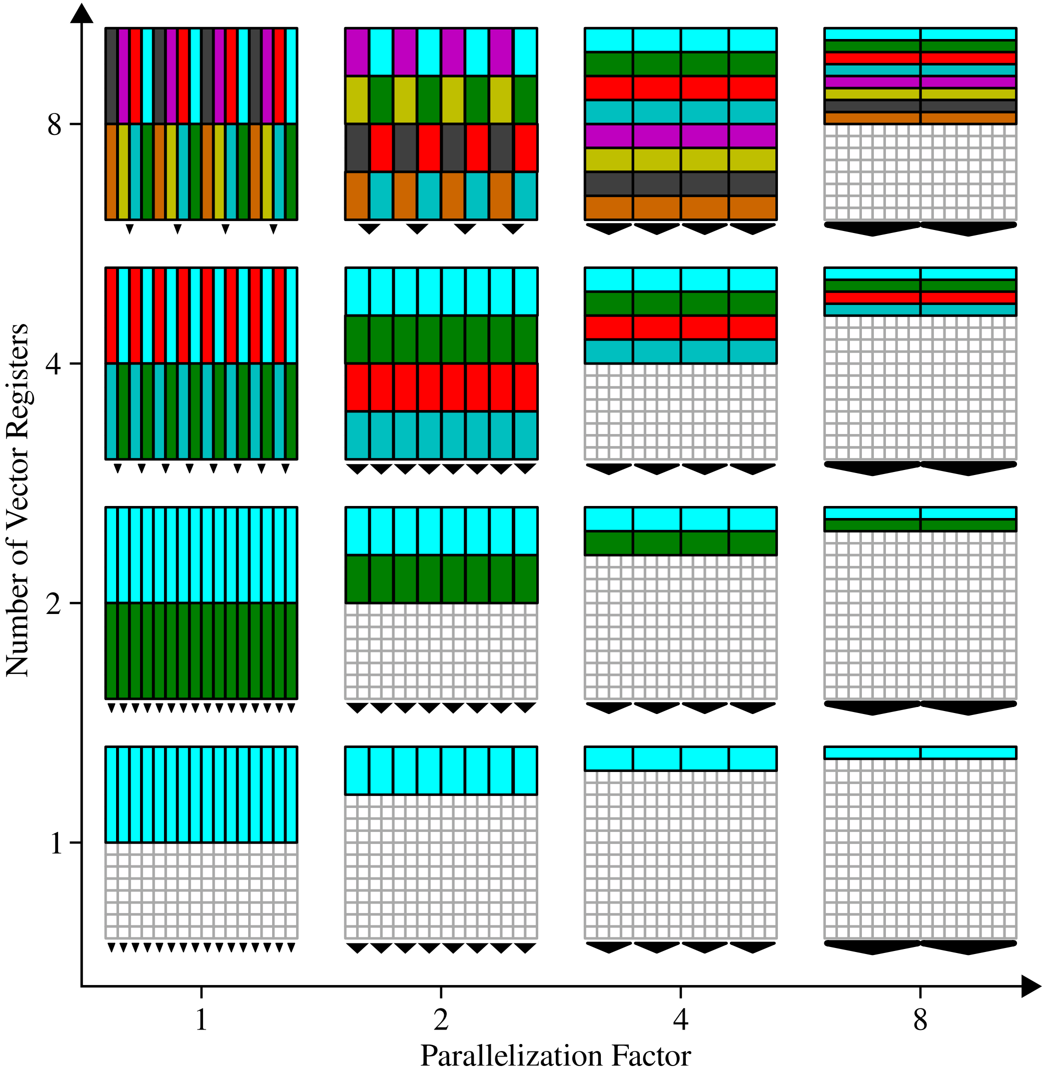
EVE BIT-HYBRID EXECUTION PARADIGM

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



Balanced Utilization:

Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

Column Under-Utilization:

Reducing the number of in-situ ALU in-favor of storage in the SRAM array

Row Under-Utilization:

Under-utilized bit-cells in the SRAM array

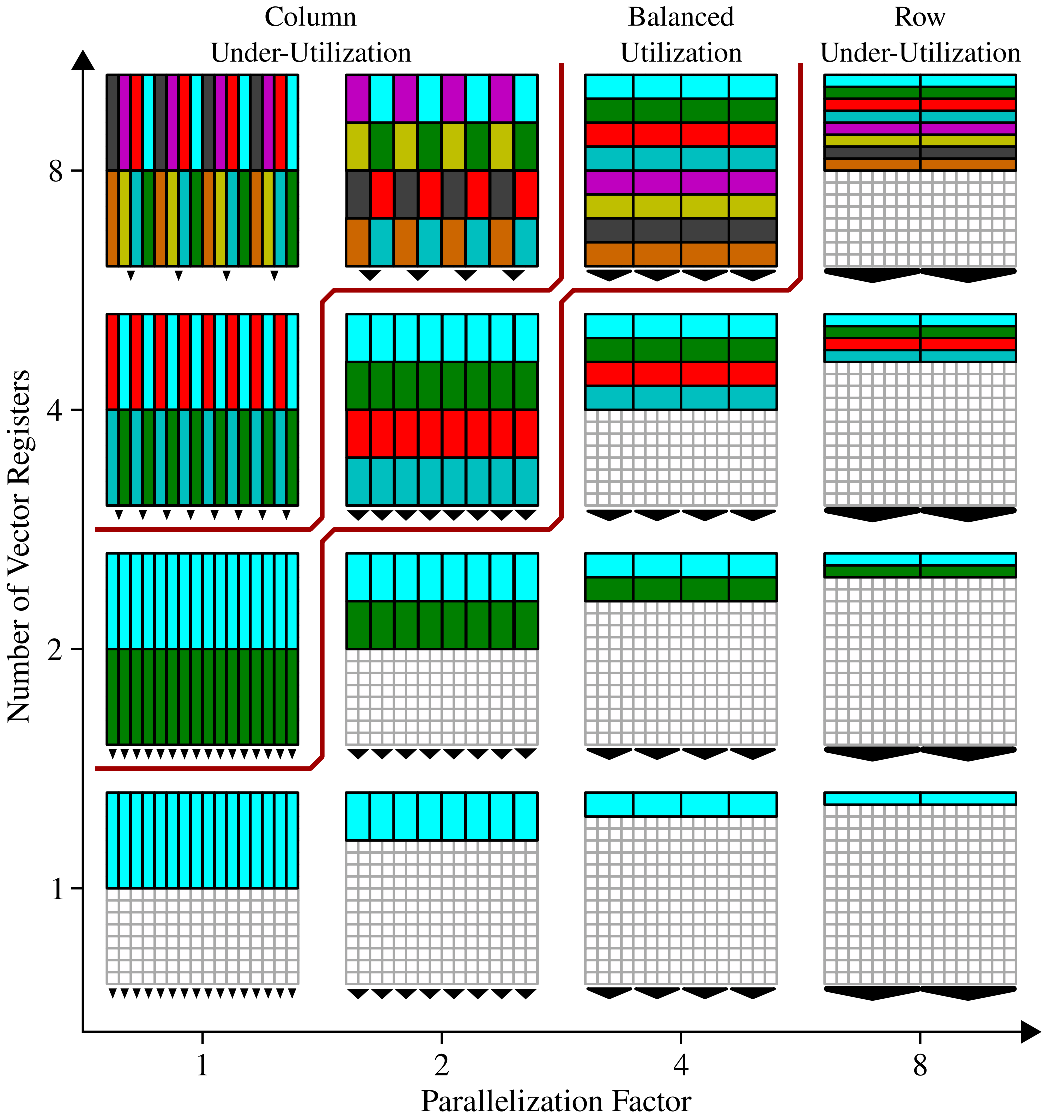
EVE Bit-Hybrid Execution Paradigm

Assumptions:

- 8-bit elements
- 16x16 SRAM array

Definitions:

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



Balanced Utilization:

Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

Column Under-Utilization:

Reducing the number of in-situ ALU in-favor of storage in the SRAM array

Row Under-Utilization:

Under-utilized bit-cells in the SRAM array

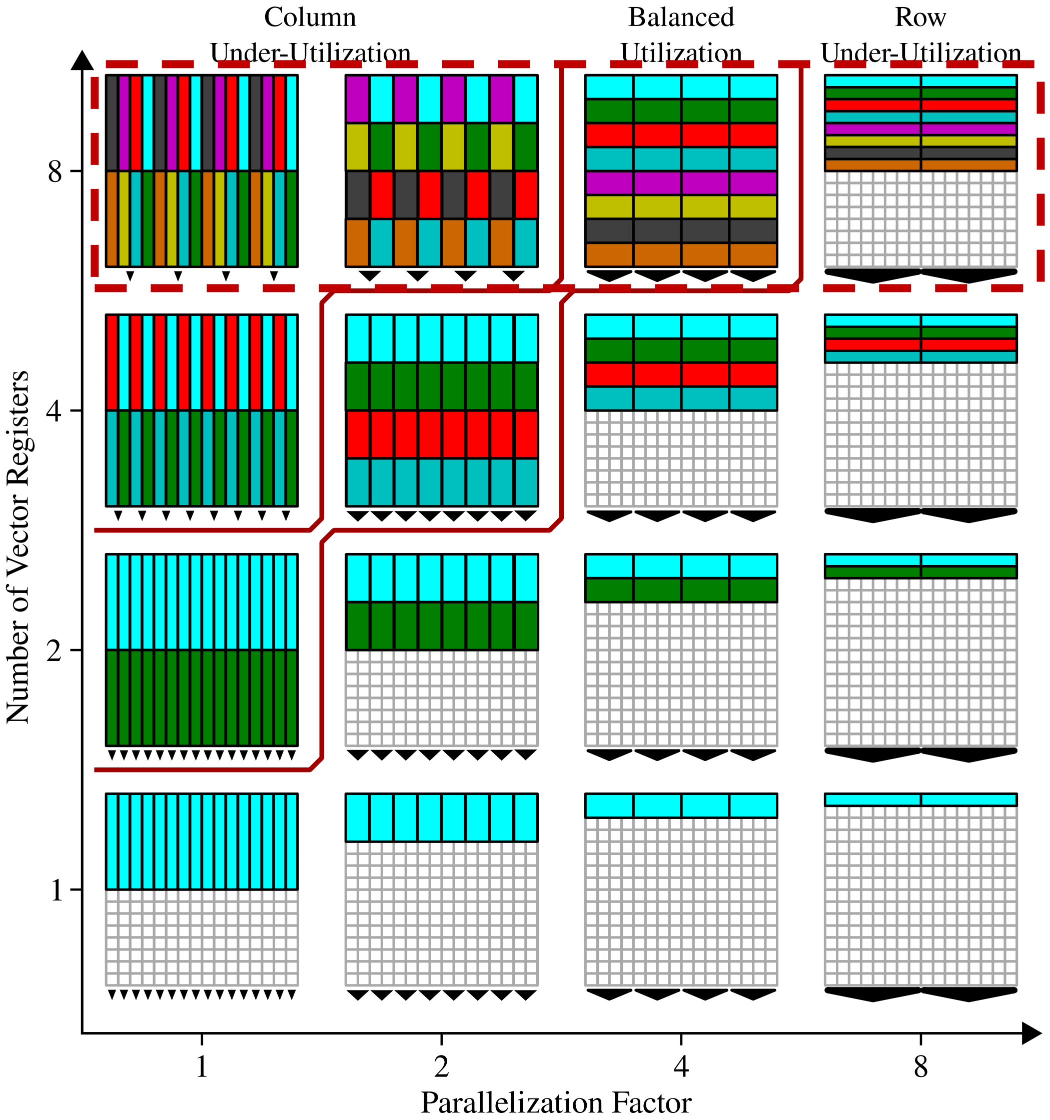
EVE Bit-Hybrid Execution Paradigm

<u>**Assumptions:**</u>

- 8-bit elements
- 16x16 SRAM array

<u>**Definitions:**</u>

- *Segment:* Elements are broken down into segments. A segment size can vary from 1 bit to 8 bit.
- *Parallelization factor:* Size of the segment (to be processed in parallel) in bits.
- Bit-Serial has parallelization factor of 1
- Bit-Parallel has parallelization factor of 8



<u>**Balanced Utilization:**</u>

Perfect utilization of all bit-cells and in-situ ALUs in the SRAM array

<u>**Column Under-Utilization:**</u>

Reducing the number of in-situ ALU in-favor of storage in the SRAM array

<u>**Row Under-Utilization:**</u>

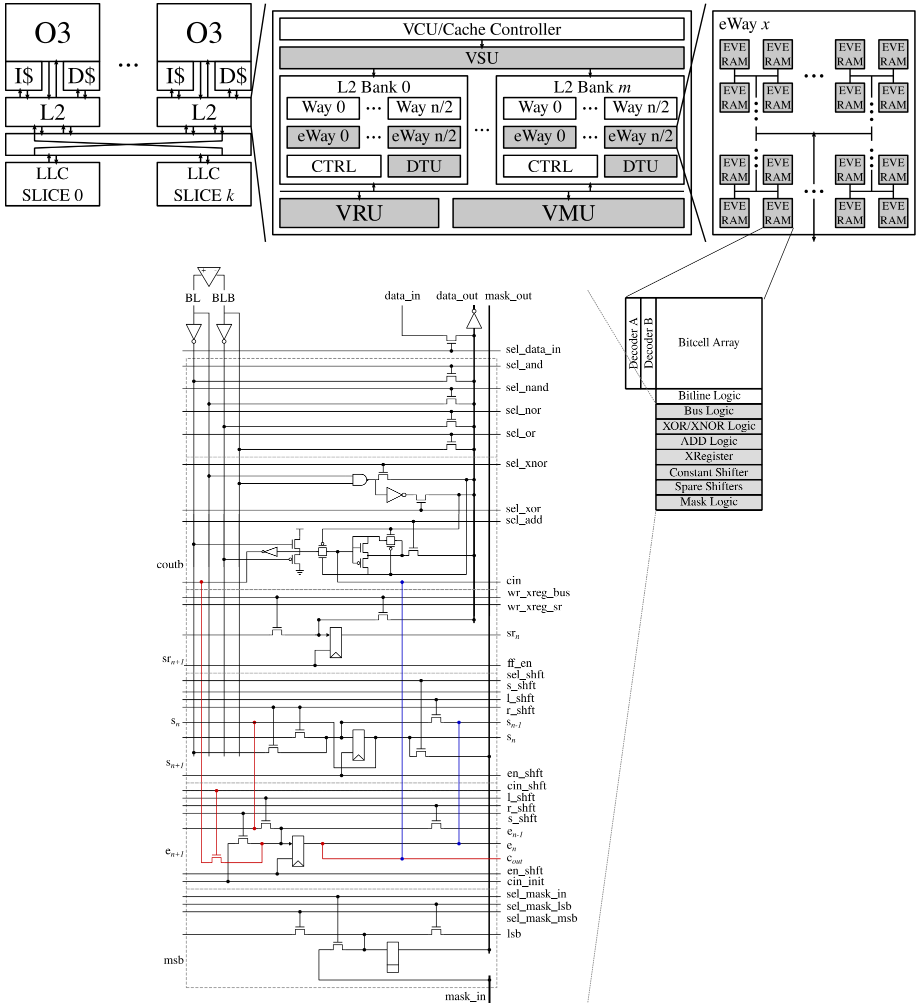
Under-utilized bit-cells in the SRAM array

OUTLINE

Motivation

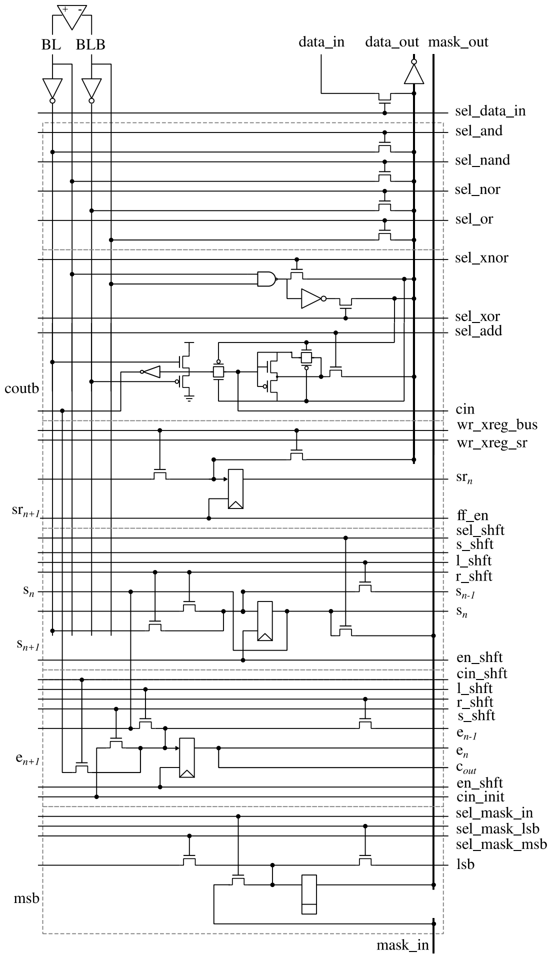
- EVE Micro-Architecture
- EVE Bit-Hybrid Execution Paradigm
- **EVE Micro-Programming & Circuits**
- EVE Evaluation

Conclusion

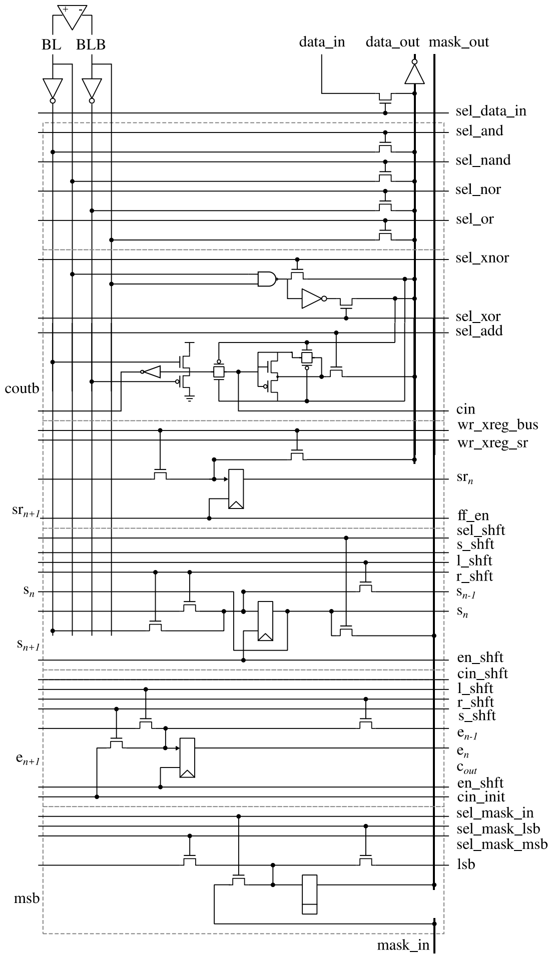


EVE Micro-Programming & Circuits

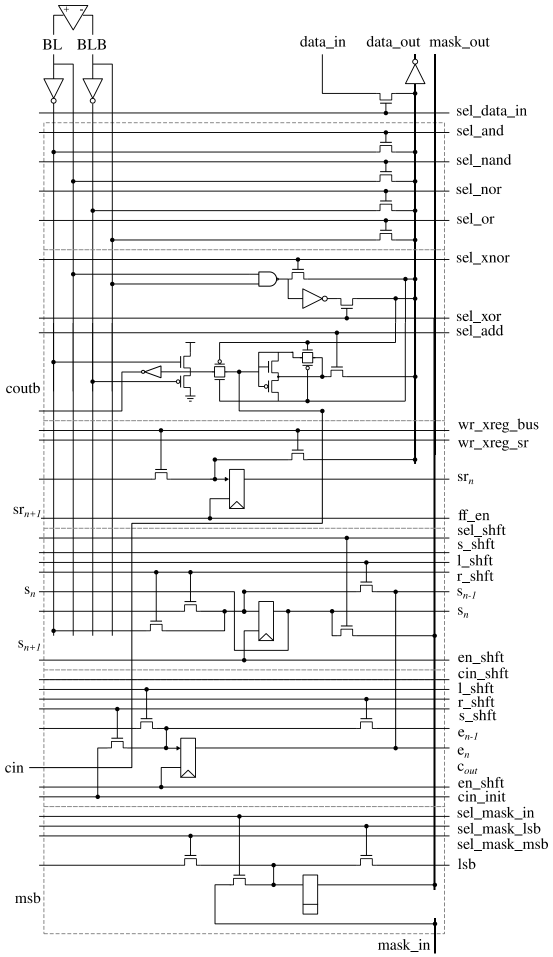
```
1   init    cnt_0, N
 loop:
1   blc     addr_a, addr_b
    ; decr  cnt_0
2   wb      addr_c, add
    ; bnz   cnt_0, loop
```



MSB



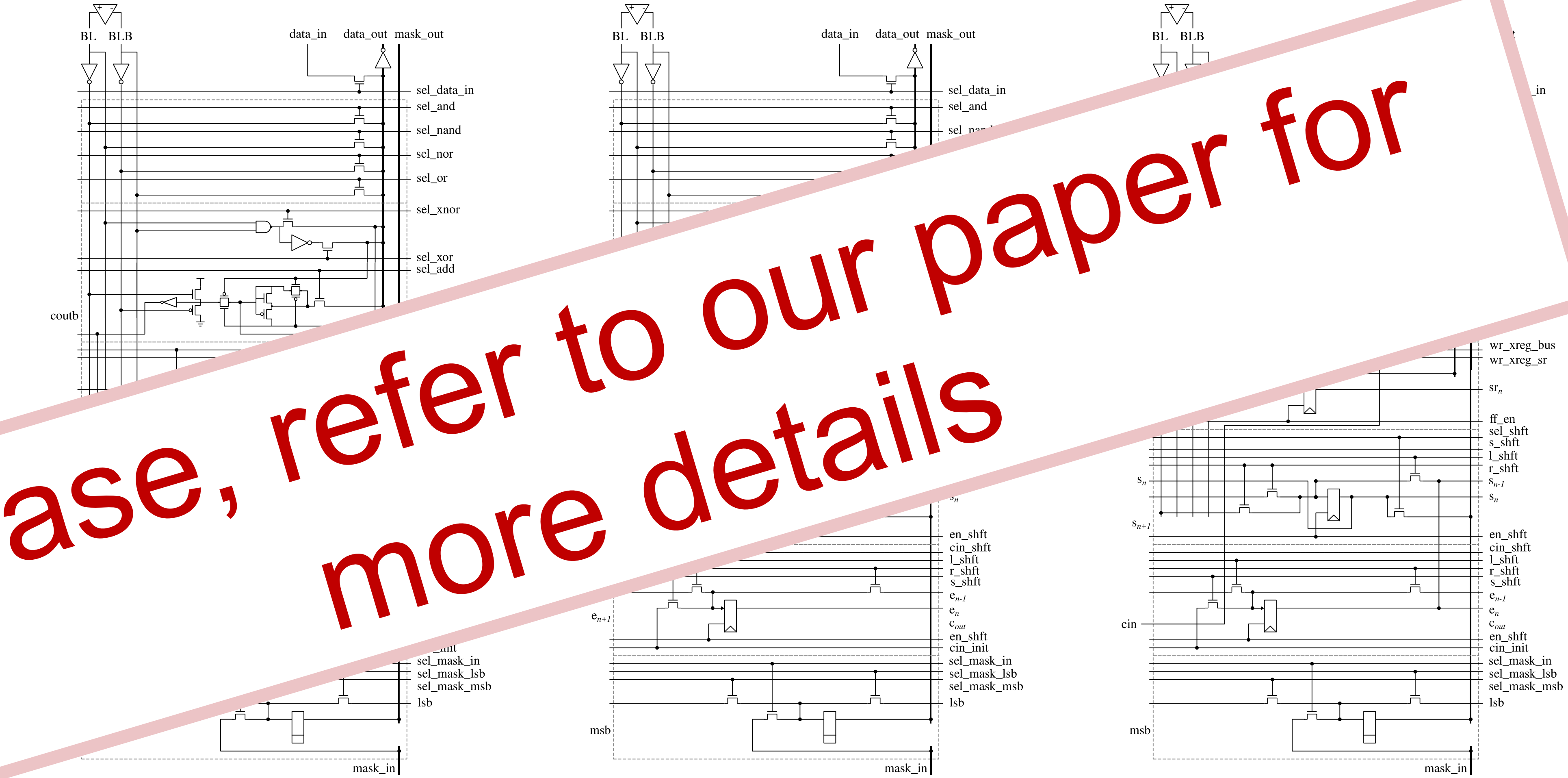
. . . .



LSB

EVE Micro-Programming & Circuits

```
1   init     cnt_0, N
  loop:
1   blc      addr_a, addr_b
    ; decr   cnt_0
2   wb       addr_c
    ; bnz
```



MSB . . . . LSB

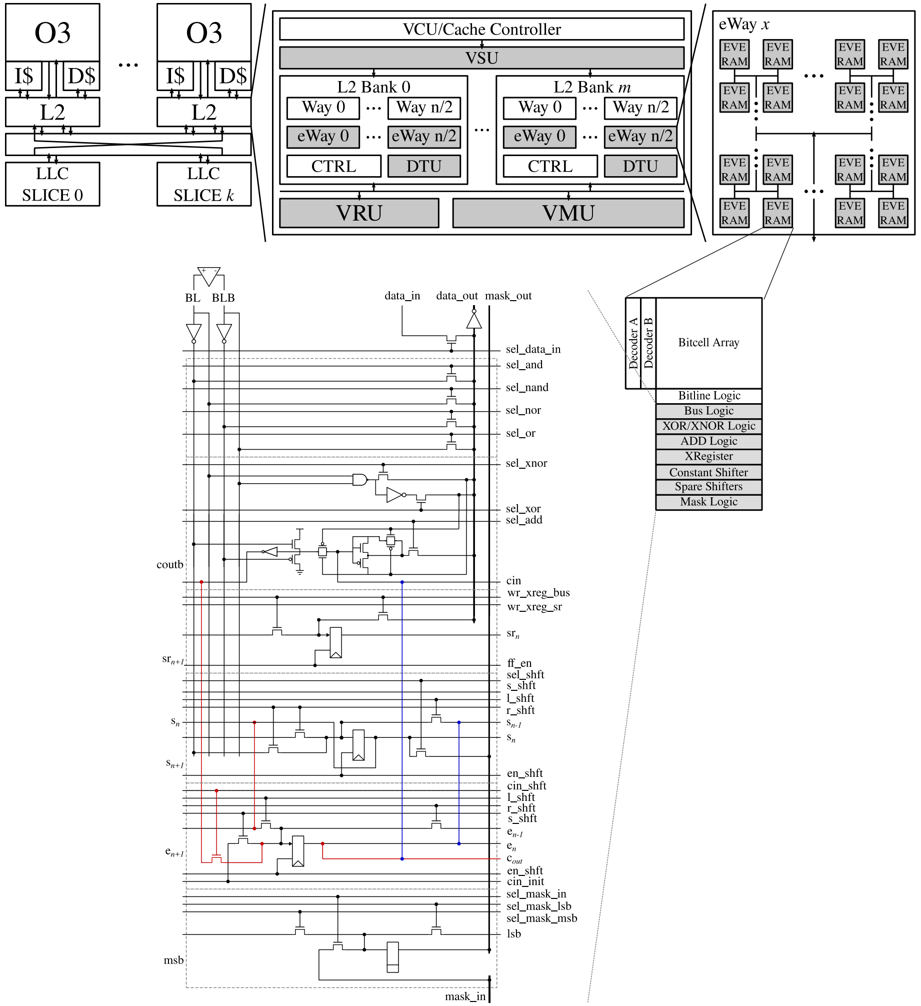
Please, refer to our paper for more details

Outline

Motivation

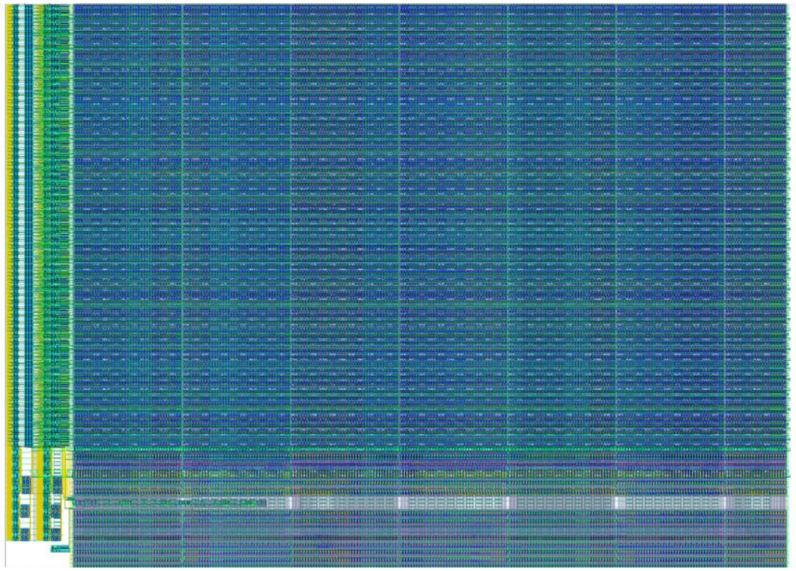
- EVE Micro-Architecture
- EVE Bit-Hybrid Execution Paradigm
- EVE Micro-Programming & Circuits
- **EVE Evaluation**

Conclusion



EVALUATION — CIRCUITS

- Utilized a simplified version of EVE circuits to estimate area and cycle-time of EVE SRAM:
  - Area: OpenRAM-generated layout
  - Cycle-time: Extracted SPICE-netlist
- Area for other components (VCU, VSU, DTU, VRU, and VMU) was estimated through SRAM-array equivalence
- EVE incurs around 12% area overhead over O3 core
- EVE incurs no cycle-time overhead for parallelization factors of eight or less



EVE-1	EVE-2	EVE-4	EVE-8	EVE-16	EVE-32
10.1%	11.7%	11.7%	11.7%	11.7%	11.0%

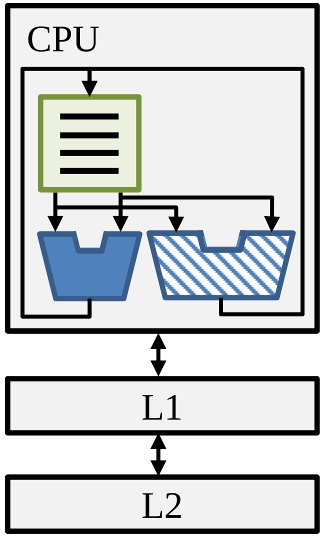
Area Overhead Over O3

EVE-1	EVE-2	EVE-4	EVE-8	EVE-16	EVE-32
1.0x	1.0x	1.0x	1.0x	1.15x	1.51x

Cycle-Time Overhead Over O3

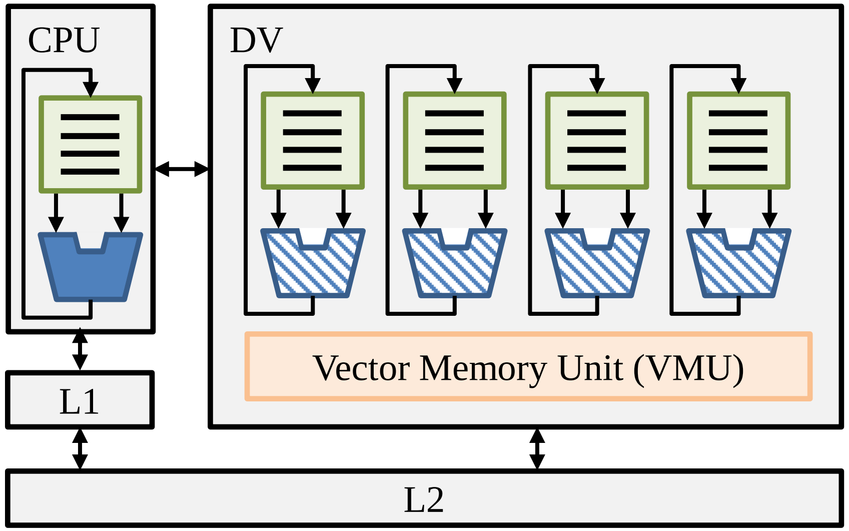
Evaluation — Architecture

O3+IV



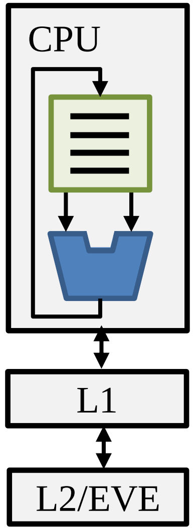
- 4 elements
- OoO execution
- 3 exec. pipes

O3+DV



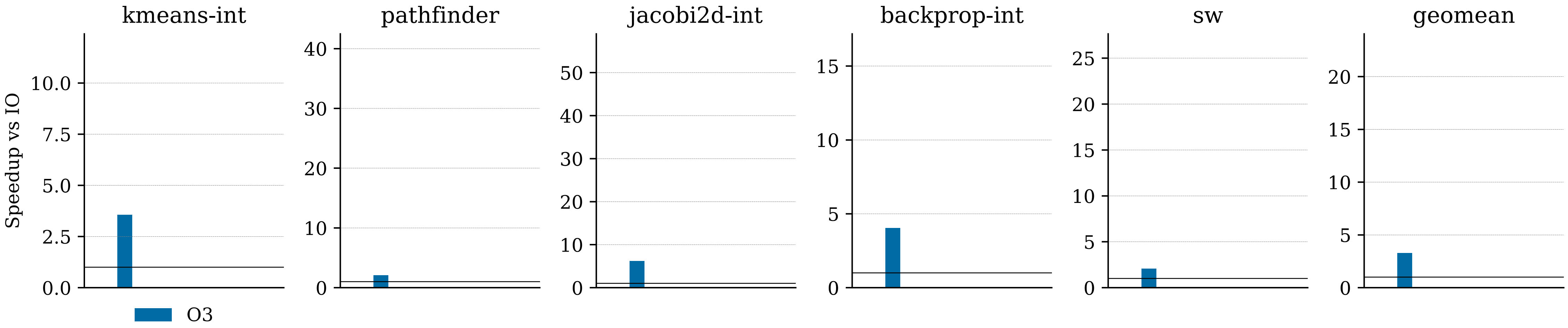
- 64 elements
- In-order execution
- 4 exec. pipes

O3+EVE

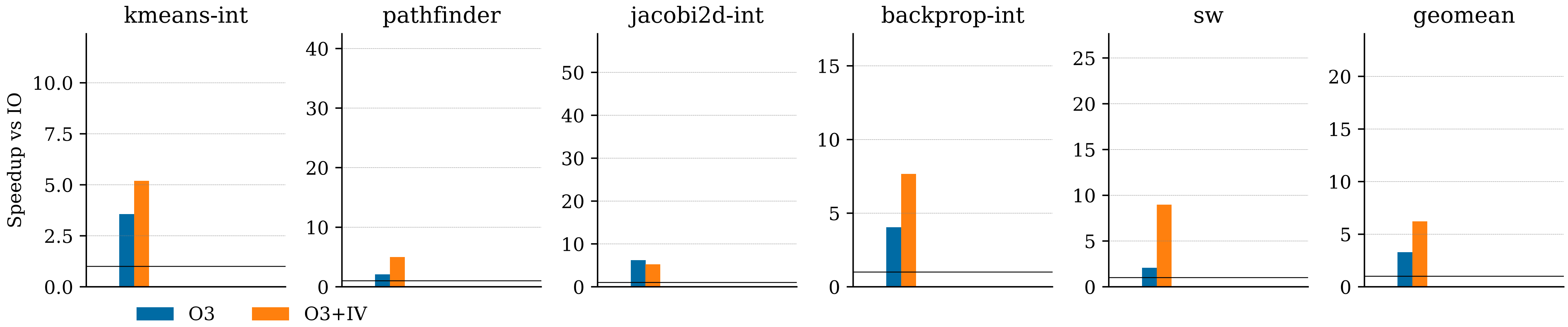


- 2048-256 elements
- In-order execution
- 1 exec. pipe

EVALUATION — ARCHITECTURE

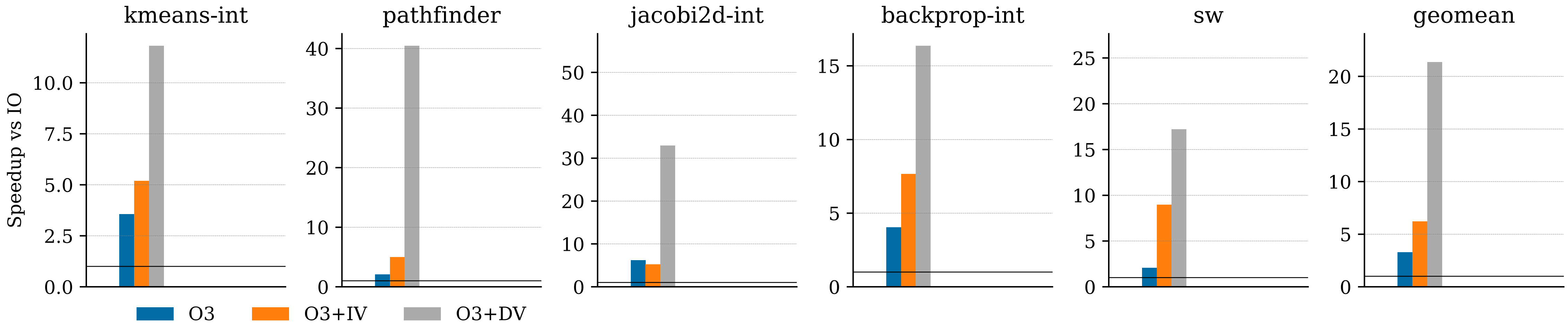


EVALUATION — ARCHITECTURE



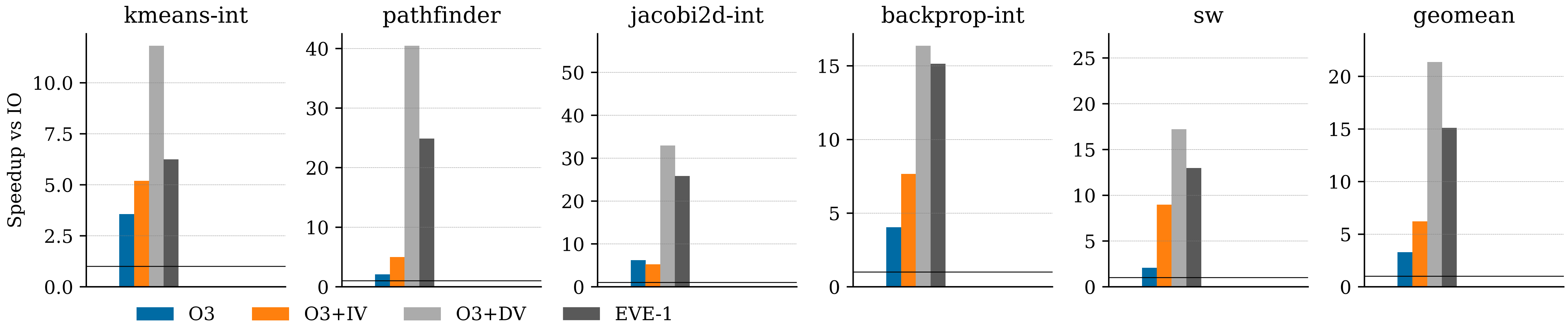
IV unit offers modest speedups at a very low area-overhead cost (~10%).

Evaluation — Architecture

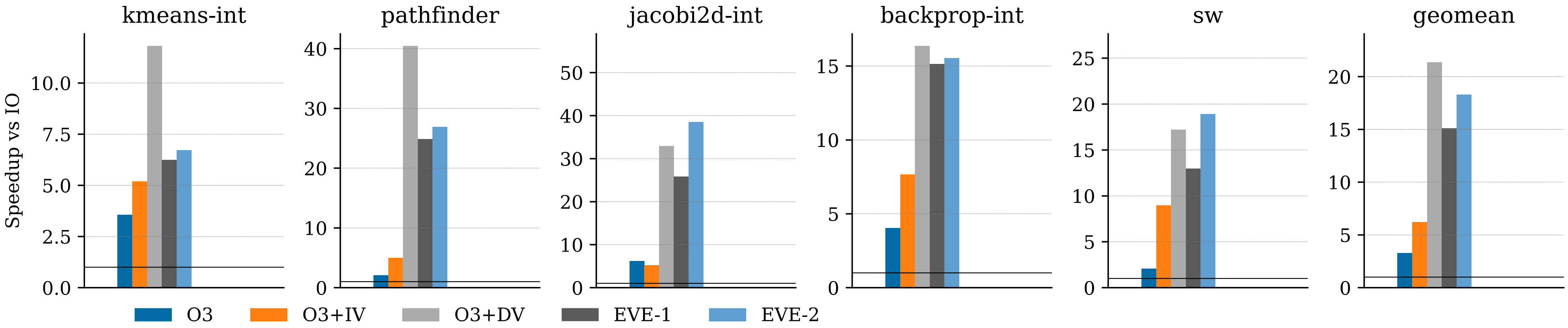


DV engine achieves much higher speedups at the cost of larger area-overhead (~100%).

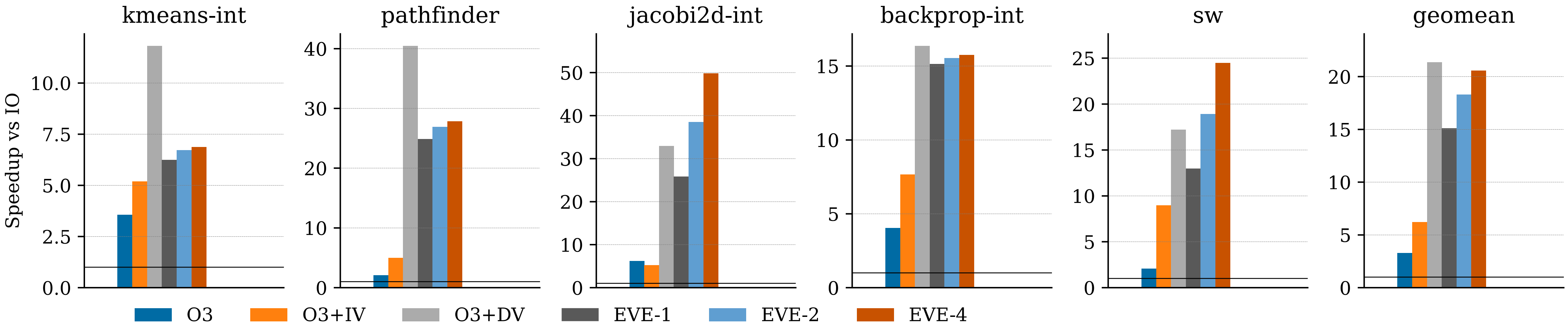
Evaluation — Architecture



EVALUATION — ARCHITECTURE

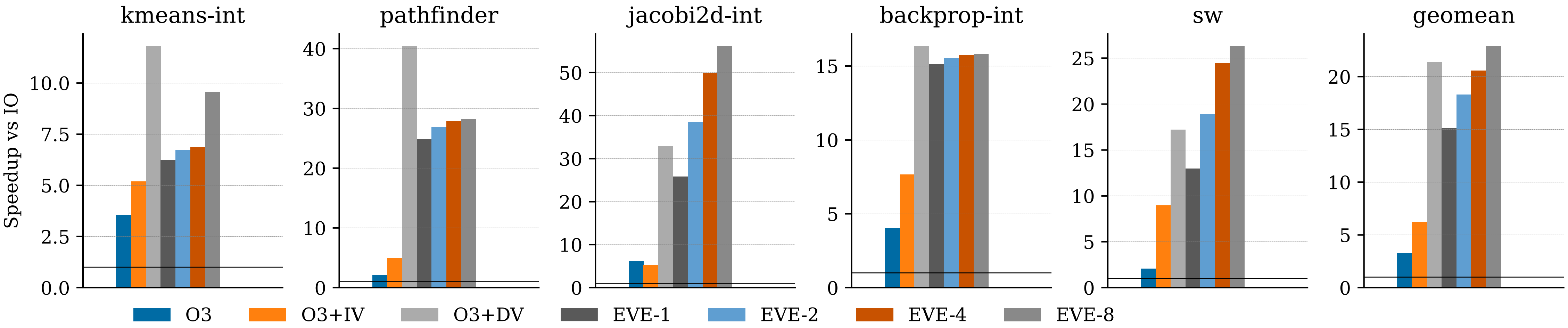


Evaluation — Architecture



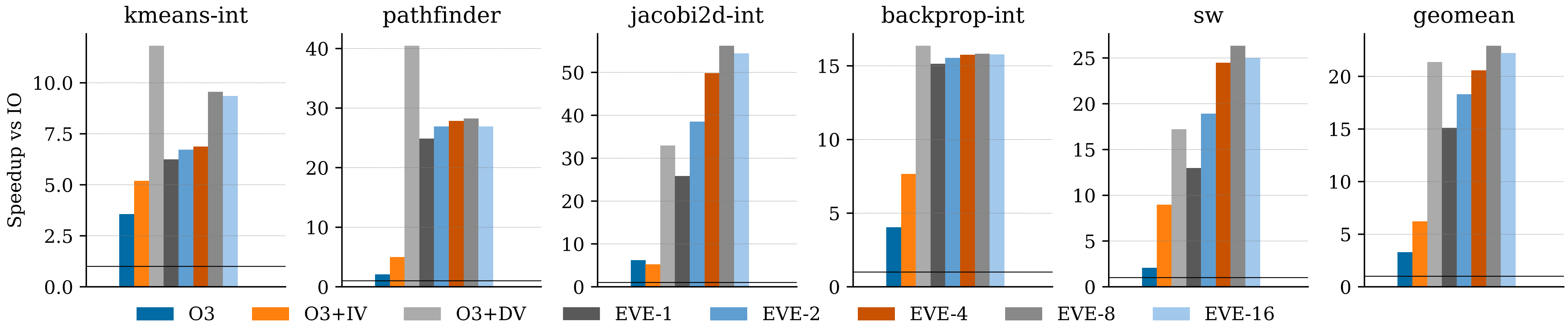
We reach balanced utilization with EVE-4 (i.e., parallelization factor of 4).

EVALUATION — ARCHITECTURE



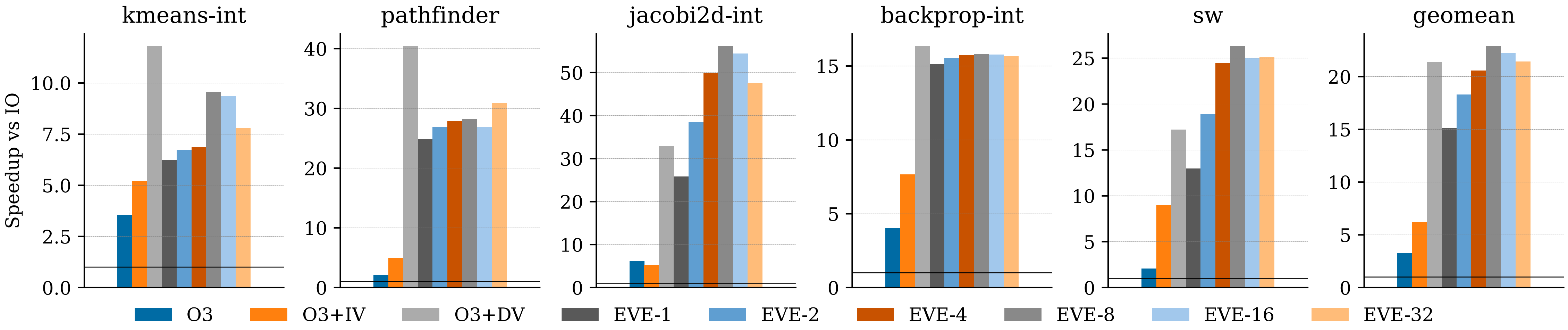
Due to sub-optimized cache-subsystem for vector memory operations, EVE-8 achieves better performance as the lower latency of EVE-8 is preferred to the longer vector lengths of EVE-4.

EVALUATION — ARCHITECTURE



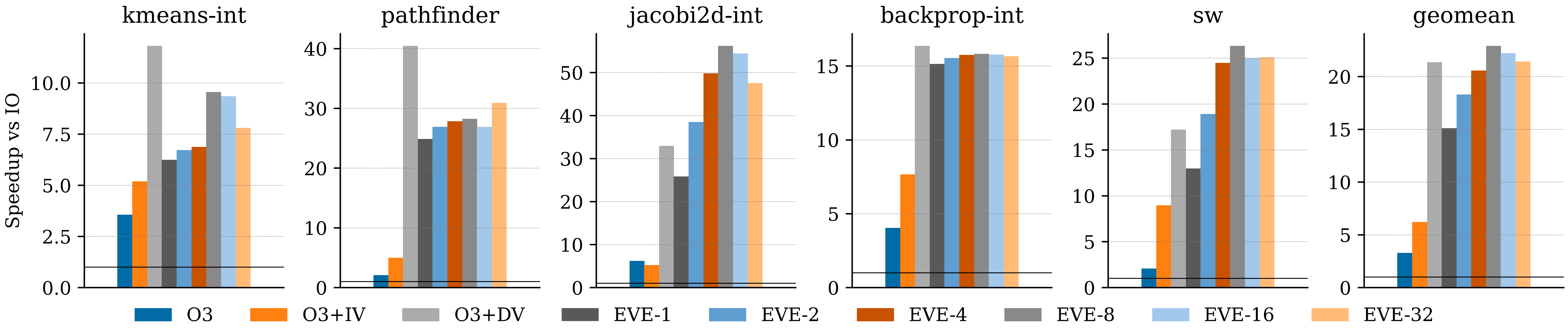
EVE-16 takes a cycle-time penalty of 15%; thus, it performs slower than EVE-8 despite the decreased latency

EVALUATION — ARCHITECTURE



EVE-32 takes a cycle-time penalty of over 50%; despite having no transpose overhead, it performs very poorly.

Evaluation — Architecture



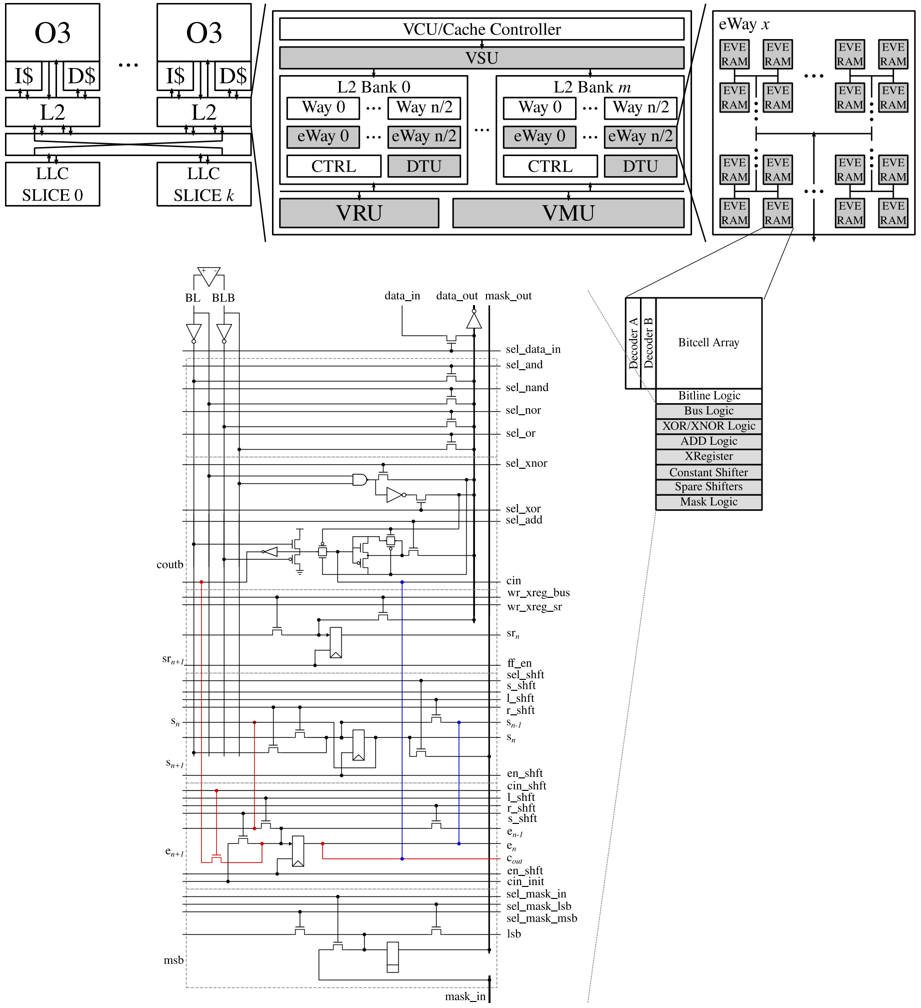
EVE-8 achieves comparable performance to an aggressive DV (~26x), while incurring an area-overhead equivalent to an IV (~12%).

Outline

Motivation

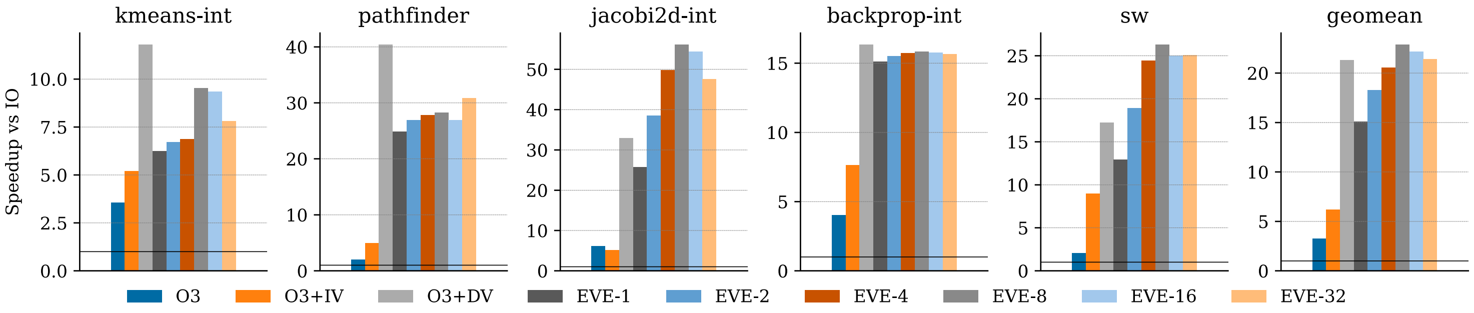
- EVE Micro-Architecture
- EVE Bit-Hybrid Execution Paradigm
- EVE Circuits and Micro-Programming
- EVE Evaluation

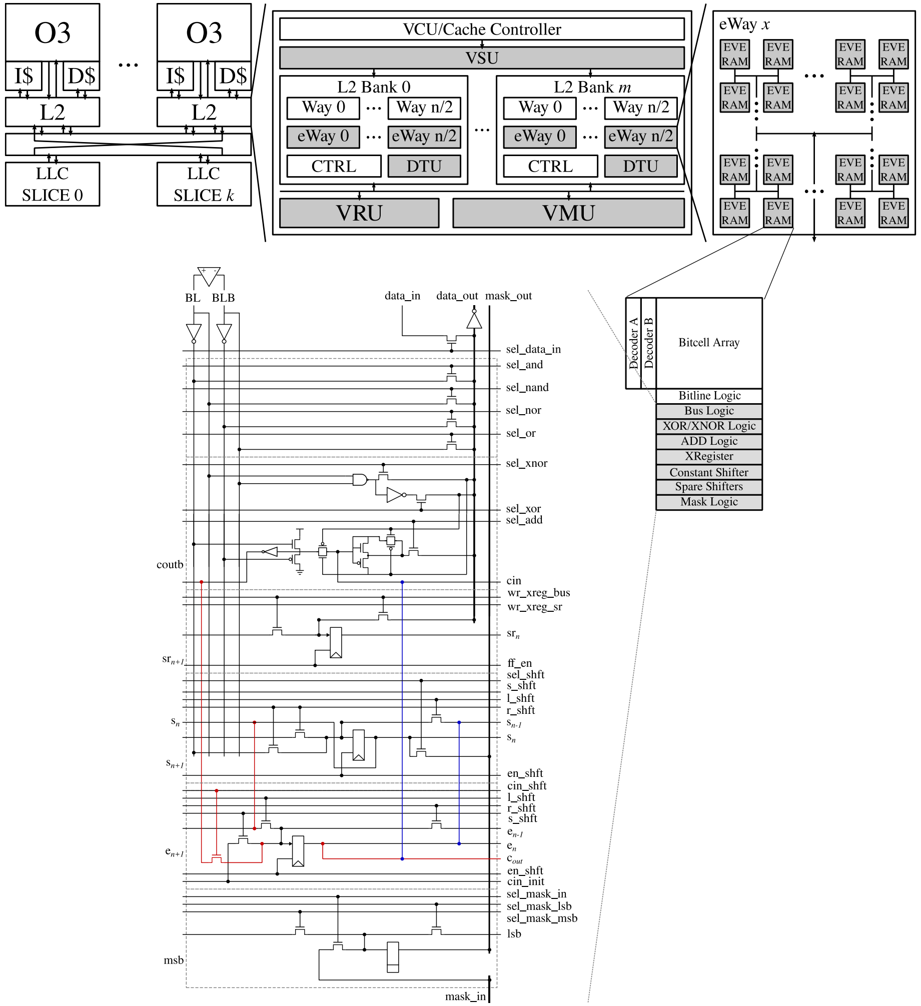
Conclusion



EVE Is Able To Balance Throughput And Latency

- Architectural template for a novel SRAM-based compute-in-memory next-gen vector engine that supports the full RISC-V RVV specifications
- Bit-hybrid execution to balance throughout and latency by alleviating row and column under-utilization
- Detailed evaluation of EVE show-casing the impacts and benefits of bit-hybrid execution on an SRAM-based compute-in-memory micro-architecture


kmeans-int
pathfinder
jacobi2d-int
backprop-int
sw
geomean
Speedup vs IO
O3
O3+IV
O3+DV
EVE-1
EVE-2
EVE-4
EVE-8
EVE-16
EVE-32


O3
I$
D$
L2
LLC SLICE 0
O3
I$
D$
L2
LLC SLICE k
VCU/Cache Controller
VSU
L2 Bank 0
Way 0
Way n/2
eWay 0
eWay n/2
CTRL
DTU
L2 Bank m
Way 0
Way n/2
eWay 0
eWay n/2
CTRL
DTU
VRU
VMU
eWay x
EVE RAM
Decoder A
Decoder B
Bitcell Array
Bitline Logic
Bus Logic
XOR/XNOR Logic
ADD Logic
XRegister
Constant Shifter
Spare Shifters
Mask Logic
BL
BLB
data_in
data_out
mask_out
sel_data_in
sel_and
sel_nand
sel_nor
sel_or
sel_xnor
sel_xor
sel_add
coutb
cin
wr_xreg_bus
wr_xreg_sr
sr_n
sr_{n+1}
ff_en
sel_shft
s_shft
l_shft
r_shft
s_n
s_{n-1}
s_n
s_{n+1}
en_shft
cin_shft
l_shft
r_shft
s_shft
e_{n-1}
e_n
e_{n+1}
c_{out}
en_shft
cin_init
sel_mask_in
sel_mask_lsb
sel_mask_msb
lsb
msb
mask_in