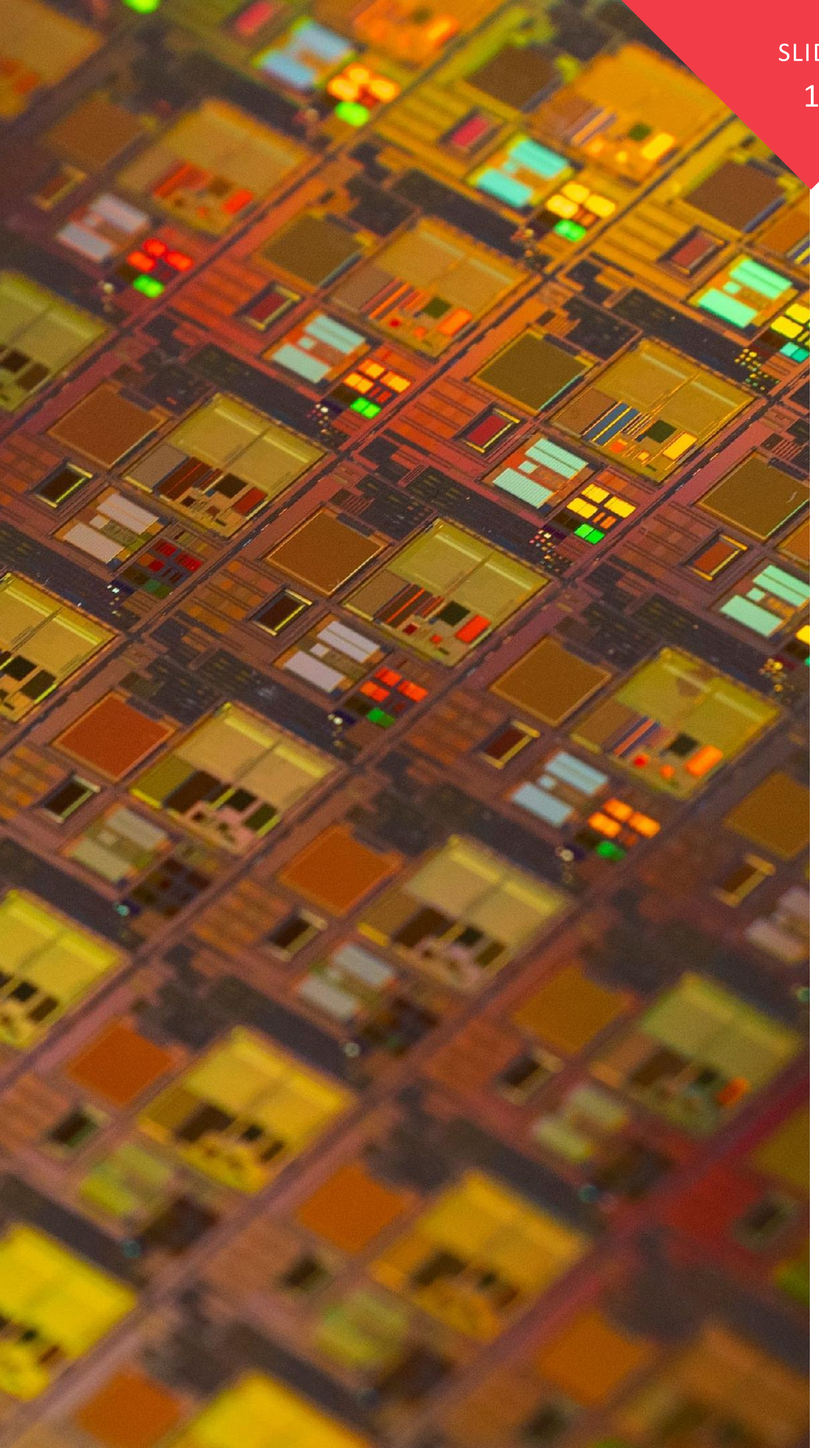


One ISA, Many Microarchitectures

Standardizing Matrix Extensions for RISC-V across the AI–HPC Continuum

Dr. Philipp Tomsich
Chief Technologist & Founder, VRULL GmbH

Actively engaged in RISC-V International

Building the standards of tomorrow

TECHNOLOGY, BUSINESS, AND COMMUNITY LEADERSHIP

With over 25+ years as a technology and business leader, Philipp's experience encompasses academia, mission-critical enterprise applications, HPC, high-assurance products, turnkey embedded design, and device manufacturing.

After building and exiting a successful embedded-systems business, he currently serves as the Founder and Chief Technologist at VRULL, where his engineering team provides outsourced semiconductor R&D for the toughest industry challenges.

ROLES AT RISC-V

- Vice-Chair, Technical Steering Committee
- Chair, Applications & Tools Committee
- Elected Board Member (Strategic Tier)

STANDARDS DEVELOPMENT

- Policies for vendor-defined extensions
- RISC-V bit-manipulation instructions
- Vector cryptographic extension
- Attached Matrix Extension for HPC and AI/ML

AWARDS

- RISC-V Technical Leadership Award 2021
- RISC-V Community Leadership Award 2021

More importantly for today’s presentation…

Instigator of standards development for matrix operations in RISC-V

RISC-V International (https://riscv.org/about/)

- RISC-V International is the global non-profit home of the open standard RISC-V Instruction Set Architecture (ISA), related specifications, and stakeholder community.
- RISC-V members across the globe contribute and collaborate to define RISC-V open specifications as well as convene and govern related technical, industry, domain, and special interest groups.
- As an open standard, anyone may leverage RISC-V as a building block in their open or proprietary solutions and services.
- Ratified specification: https://riscv.org/specifications/ratified/
- Specifications under development: https://riscv.org/specifications/development/

The development process for all RISC-V specifications is open, transparent, and primarily conducted on GitHub.

However, to contribute to the development of specifications, you must be a member of RISC-V International.

LEARN HOW TO CONTRIBUTE

RISC-V can change the economics of matrix and tensor acceleration

GOALS FOR RISC-V MATRIX STANDARDISATION

Industry-wide sharing of a common

matrix ISA extension

REDUCING DUPLICATION
COLLABORATING ON THE COMMON PARTS OF EVERY IMPLEMENTATION, RISC-V REDUCES DUPLICATION AND FACILITATES REUSE

Micro-architecture

allows differentiation and acceleration

ENABLING COMPETITION AND INNOVATION
TAILOR AND OPTIMISE PERFORMANCE WITH DOMAIN-SPECIFIC ACCELERATION AND CUSTOM INSTRUCTIONS FOR A COMPETITIVE EDGE

A shared ecosystem

will support rapid productisation

LOWERING THE BARRIER OF ENTRY
A UNIFIED FOUNDATION FOR DEVELOPERS TO BUILD ON A CONSISTENT, OPEN PLATFORM
ACCELERATING INNOVATION AND COMPATIBILITY

More than AI/ML

Standardising RISC-V matrix extensions touches on many more use-cases than just AI/ML.

Embedded applications

- **Image and video processing**
  Filtering, transformations (e.g., DCT/FFT), and color space conversions in devices like drones, smartphones, and surveillance systems.
- **Audio processing**
  Noise suppression, echo cancellation, and beamforming in devices like smart speakers,
  hearing aids, and voice assistants
- **Communication systems**
  MIMO (Multiple Input, Multiple Output) signal processing and channel estimation in wireless communication protocols like 5G
- **Robotics and Automation**
  matrix equations for real-time control in robotics, including motion planning and dynamic control

More than AI/ML

Standardising RISC-V matrix extensions touches on many more use-cases than just AI/ML.

High-performance computing applications

- Weather and Climate Modeling
- Molecular Dynamics
- Astrophysics
- Finite Element Analysis
- Protein Structure Prediction
- Risk Analysis and Algorithmic Trading
- Graph Analytics
- Computational Fluid Dynamics
- Particle Simulations
- Ray Tracing and 3D Modeling
- …

Requirements to standardise an extension

The same quality gates apply both for the standardisation of AME and IME.

Quality gates

- Development of a consensus specification
- Development of a test specification and architectural compatibility tests
- Proof-of-concept implementation to allow end-to-end validation/experimentation and collection of quantitative data
  - Simulators and formal model
  - Basic software enablement
  - Porting of key applications

A small, self-contained specification brings significant advantages for a fast ratification

Building RISC-V into a “common language” for AI/ML, HPC and IoT

A TALE OF TWO STANDARDS

RISC-V Integrated Matrix Extension

Scalable matrix extension built on the Vector Extension intended to reuse its micro-architectural resources

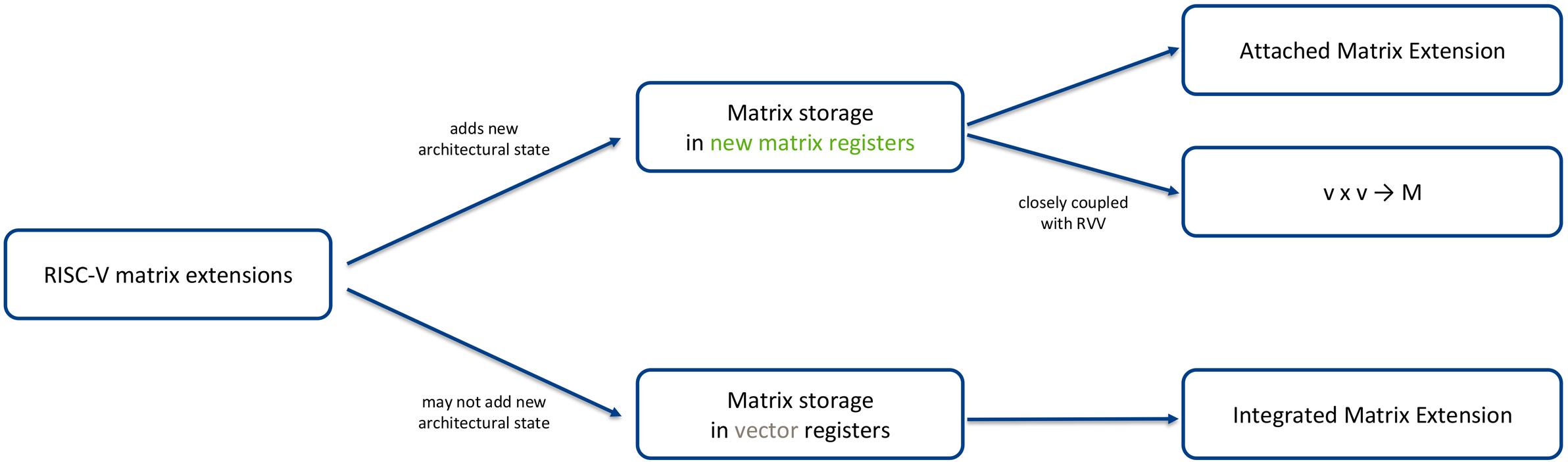
Designed to provide similar levels of geometry agnosticism and integrate with the memory model of the RISC-V Vector Extension

RISC-V Attached Matrix Extension

Matrix extension in a self-contained, orthogonal execution unit that scales for multiple domains: IoT/Embedded, AI/ML, and HPC.
Designed for maximum freedom for integrators and accepting higher software complexity in return.

Shall provide a path to encompass tensor operations in the future.

A taxonomy of approaches to designing RISC-V matrix extensions



A DESIGN SPACE, NOT A CONTEST

Four aspects anchor the extension in the design space

The four specification efforts are targeting different design spaces.
Four aspects define how we should think about them — and explain why no single design point fits the whole RISC-V ecosystem.

1

RVV reuse

How much of the V register state and programming model is reused?

2

New state

Does the OS need to store new state?

3

Tile size & shape

What is the largest tile that can be supported efficiently?

This drives computational intensity and determines the performance ceiling.

4

Implementation freedom

Can we share one matrix unit across RISC-V harts?

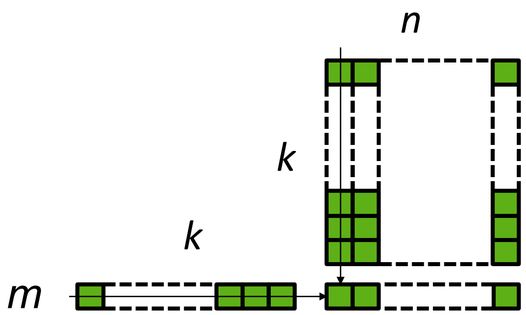
Can matrix acceleration be provided in cores that do not implement RVV?

From most RVV-integrated (BDP) to most independent (AME):
increasing capability and increasing specialization.

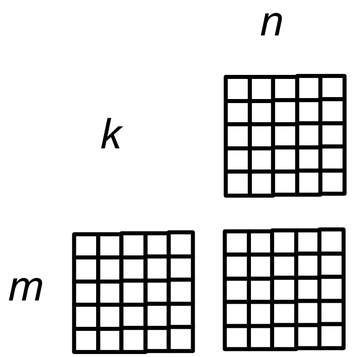
A DESIGN SPACE, NOT A CONTEST

Four structurally different approaches

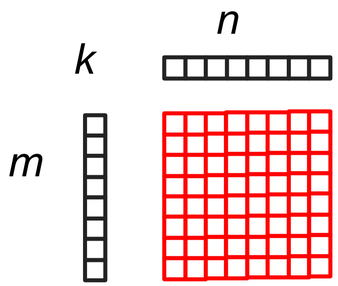
Dot-product
(fast-track)



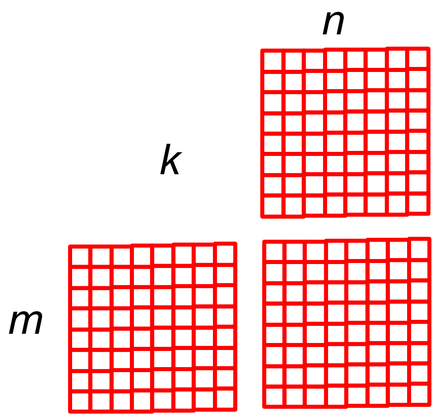
Zvvm
(Integrated Matrix)



Zvt
(Vector-Matrix)



Zvm
(Attached Matrix)



THE DEFINING QUESTION

Where do A, B, and C live?

The sharpest way to tell the four apart: are the operands and the accumulator held in RVV vector registers, or in dedicated matrix state?

	Operands A, B	Accumulator C	Abstraction
BDP	RVV vector registers	RVV vector registers	vector-matrix primitive
IME	RVV vector registers (as tiles)	RVV vector registers (as tile)	tile GEMM
VME	RVV vector registers	Dedicated accumulators	tile GEMM
AME	Dedicated tile registers	Dedicated accumulators	tile GEMM

THE PROBLEM

Every matrix ISA forces a binary fork

Every matrix ISA outlives its first microarchitecture: the same binary must run on not-yet-designed silicon, and at vector widths.
Hard-coding tile shape or adding dedicated state freezes those choices at spec time and forces a fork whenever the datapath changes. The design goal instead: encode intent (streaming depth, precision, granularity), and leave hardware free to choose how to deliver it.

Arm SME

Dedicated state, dedicated mode

- Dedicated ZA tile register file
  New architectural state the OS must save, restore and context switch
- A separate streaming mode, with explicit entry/exit the programmer model has to manage
- Tile geometry is tied to the implementation's SVL — wider hardware needs a different binary
- Microscaling bolted-on SME2 instructions, not part of original design

Intel AMX

Geometry frozen at design time

- Fixed 16×16 tiles
  Tile shape is hard-coded into the ISA, not chosen by software
- Eight dedicated TMM tile registers, plus a TILECFG state that must be configured and restored
- Cannot exploit wider datapaths without an ISA revision: no path to scale a binary forward
- FP8 microscaling added as a later extension (AMX-FP8)

The ecosystem fragments: per-platform tuning and multi-version liibraries at every architecture boundary.

INTRODUCING ZVMM (THE INTEGRATED MATRIX EXTENSION)

Geometry is derived, not declared

Zvvm encodes that intent directly: no new state to freeze, no shape to hard-code.
Tile state reuses the vector register file, and tile shape is derived from VLEN, SEW, and the aspect-ratio field λ.
The same binary expresses streaming depth, precision, and granularity as intent; each implementation chooses the geometry that fits its datapath.

No fixed state

Matrix tiles are the standard vector register file. No dedicated tile file, no separate accumulator state. The matrix extension adds capability, not architectural state.

No fixed shape

There are no tile-shape instructions and no shape constants. The programmer never declares M, N, or K: we will never outgrow a frozen into the binary at spec time.

The tile fits the datapath

The dimensions are derived with λ shaping the aspect ratio. Each implementation's geometry falls out of its own VLEN. Hardware chooses the shape that fits, not the other way around.

So scaling comes for free. Next up: The algebra that explains how.

Beyond Vectors

Scalars 0-dimensional compute
(single values processed sequentially)

Vectors 1-dimensional data parallelism
(RVV enables scalable SIMD operations)

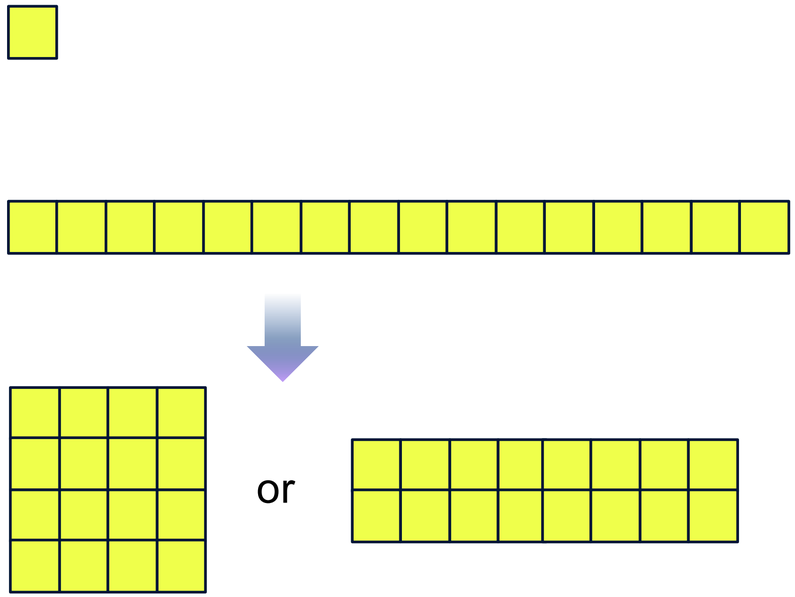
Matrices 2-dimensional data parallelism
(the natural abstraction for ML workloads)

Modern AI and some HPC workloads are fundamentally matrix computations.

Vectors already express 1-D parallelism

Next step: lift vector computation to 2-D matrix operations

Same state, different meaning



Matrix Multiplication

$A \cdot B^T + C \rightarrow C$

A, B VRs are matrix tiles of dimension ***σ x λ***

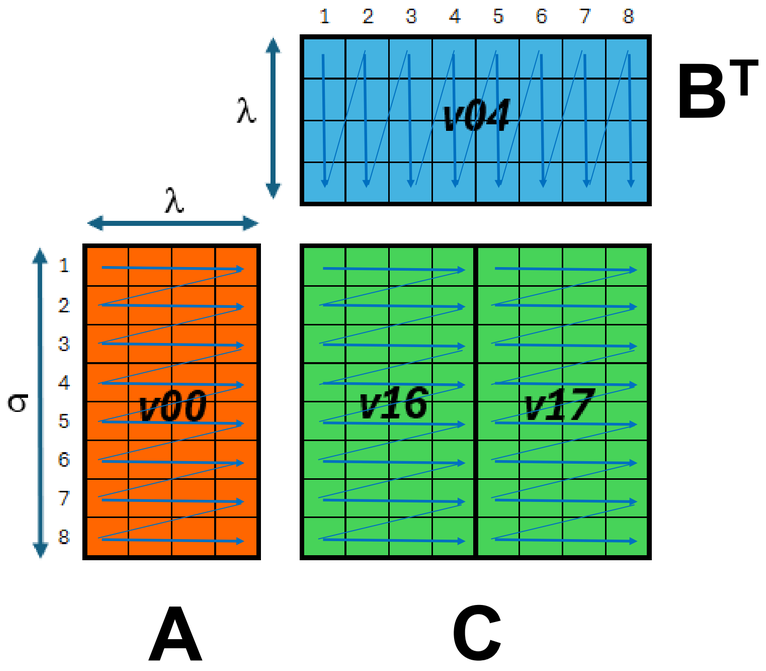
- **λ** must be one of 1, 2, 4, 8, 16, 32, 64
  - choice of implementor
  - is the **only geometry parameter**
- ***σ = VLEN/ (λ SEW)***
- Any VLEN is supported (up to 64kbits)
- Elements of width SEW in a vector register
  VL_{MAX} = VLEN/SEW

Output C matrix tile:

- One or multiple VRs
- ***$MUL_C = VL_{MAX} / λ^2$***
- At VL_{MAX} C is square! ***σxσ***
  - Maximizes compute intensity

VL = 32
λ = 4
σ = 8

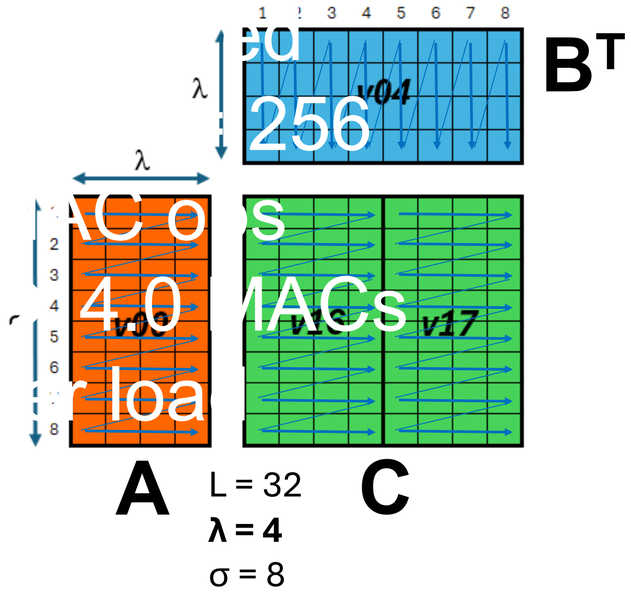
Example



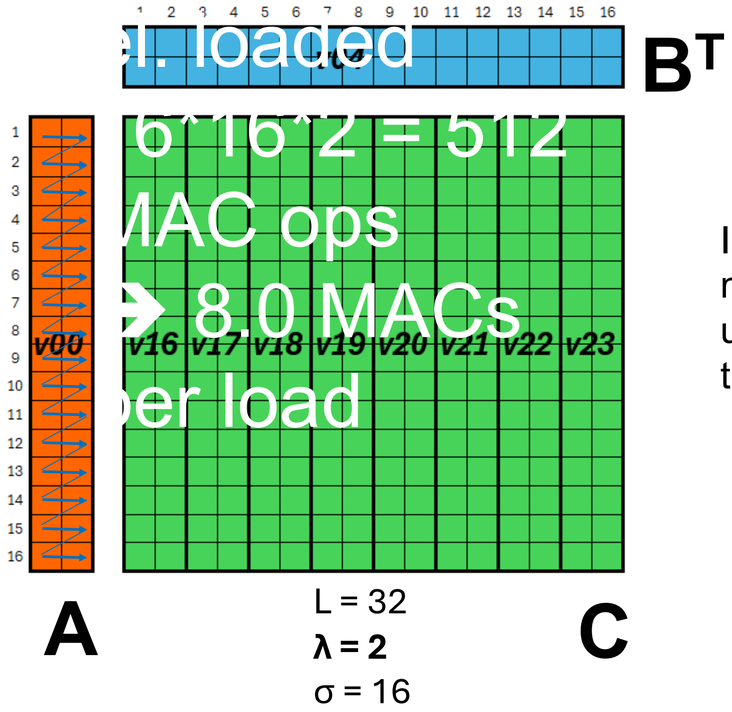
Multiply: ***vmmacc.vv vd,vs1,vs2***
Tile Load: ***vmtl vd,(base),ld***
Transposed Load: ***vmttl vd,(base),ld***

Compute Intensity

(C remains in place)
2*32 = 64 A, B



(C remains in place)
2*32 = 64 A, B

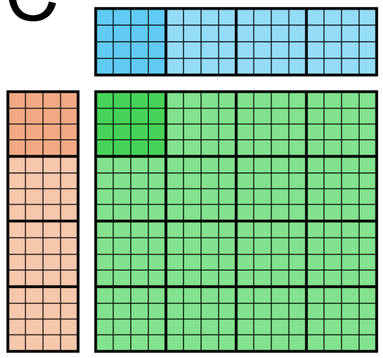


Implementation may alias C to VRs or use a hidden accumulator: the ISA is agnostic.

Compute Intensity vs. Multiply Instructions

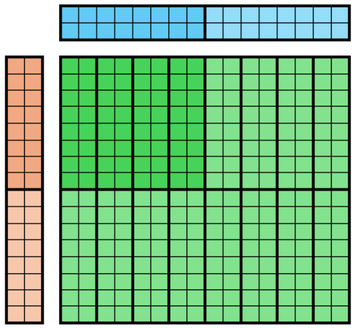
With 16 VRs for C

When VLMAX is even power of 2



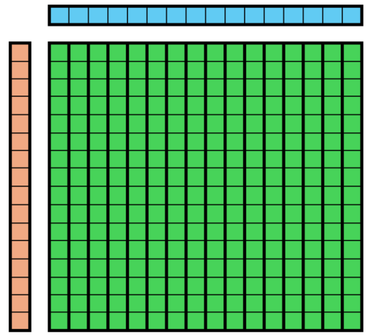
|C|=1

16 instructions



|C|=4

4 instructions

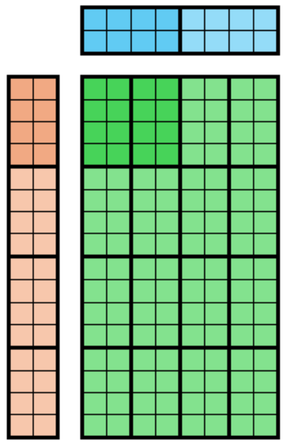


|C|=16

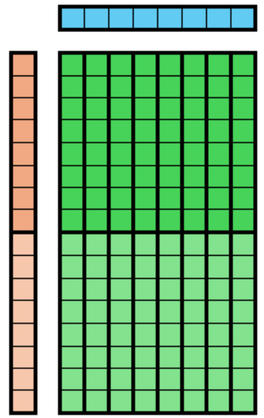
1 instruction

When VLMAX is odd power of 2

8 instructions



|C|=2

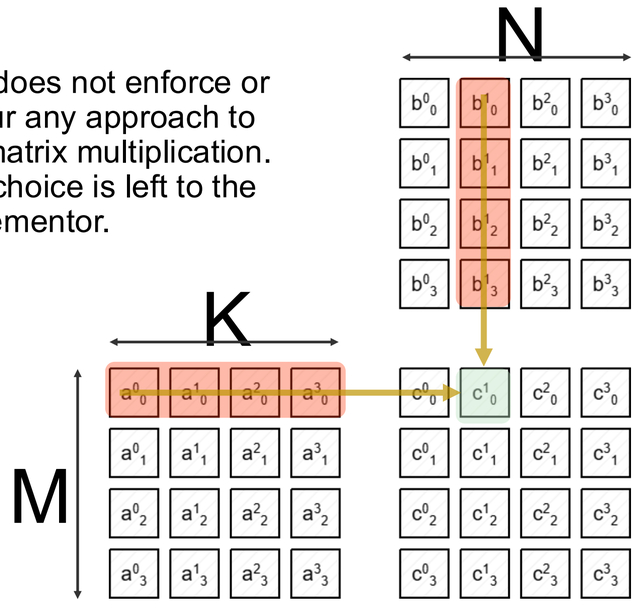


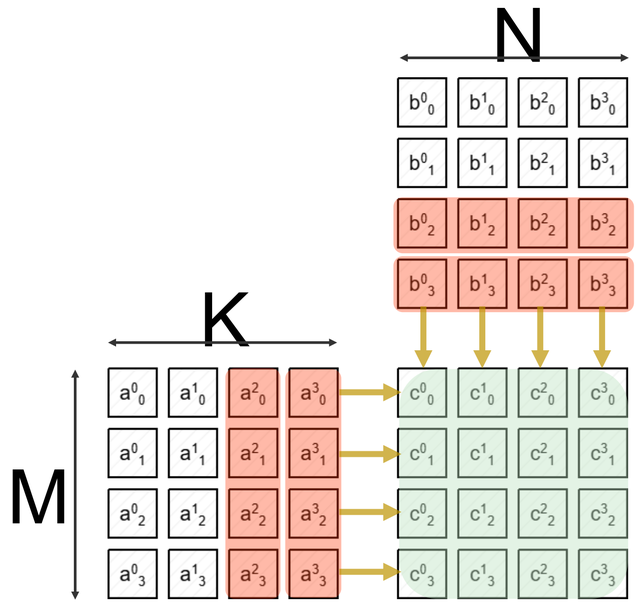
|C|=8

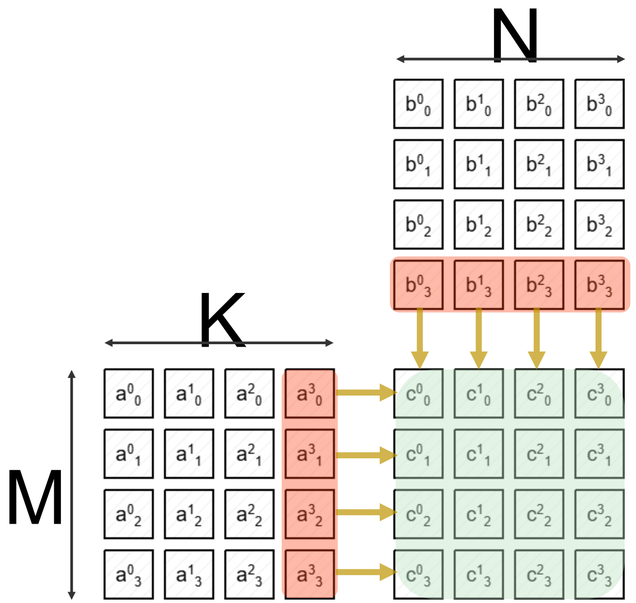
2 instructions

Inner versus Outer Product

IME does not enforce or favour any approach to the matrix multiplication. The choice is left to the implementor.







Inner Product

Core operation: **Dot Product**

Examples: systolic array;
one or several „fat“ dot products

```
for (i=0;i<N;i++)
  for (j=0;j<M;j++)
    sum = 0
    for (k=0;k<K;k++)
      sum += a^k_j*b^i_k
    c^i_j = sum
```

Outer Product

Core operation: **Rank-2 Update**

```
for (k=0;k<K;k+=2)
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      c^i_j += a^k_j*b^i_k+ a^k+1_j*b^i_k+1
```

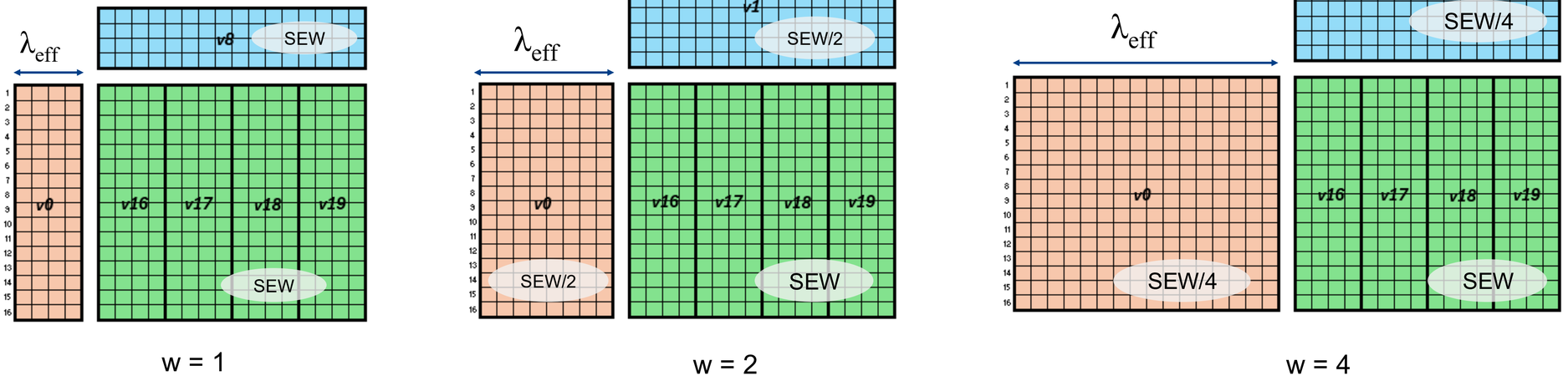
Outer Product

Core operation: **Rank-1 Update**

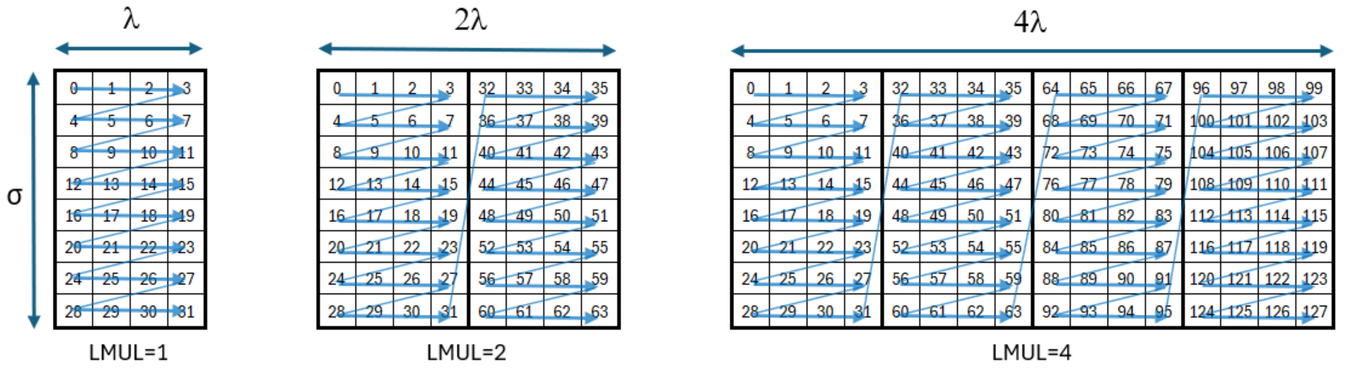
```
for (k=0;k<K;k++)
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      c^i_j += a^k_j*b^i_k
```

Widening and LMUL $\lambda_{eff} = \lambda \cdot w \cdot LMUL$

Widening multiply & accumulate up to 8x



LMUL only applies t
A and B tiles!



Performance Scales

$$CI = \frac{M \cdot M \cdot VL/VLMAX \cdot \mathrm{K_{eff}}}{M \cdot K_{eff}(1 + VL/VLMAX)} = \frac{VLMAX \cdot VL}{\lambda\,(VLMAX + VL)} \rightarrow \frac{VLMAX}{2\lambda}$$

Compute intensity
MACs/load

$$Performance = CI \cdot BW = \frac{VLMAX \cdot BW}{2\lambda}$$

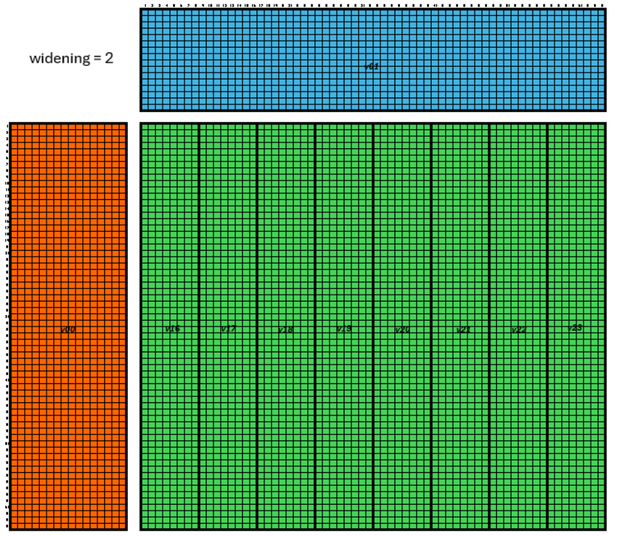
HW	VLEN	λ	BW
SW	VL	w	LMUL

With the same Cache-to-VR bandwidth: a wider implementation sustains full utilization with no recompilation. Same binary, VLEN=256 → 65536, 16× throughput.

HPC example:

- VLEN=16384
- FP32 += FP16*FP16 (w=2)
- VL = 1024 FP16 elements → **VRs fully utilized**
- Compute intensity: 64 ops/load

Performance is limited by the 32 architectural VRs, more would enable larger or more C panels.



E.g. FP32 += FP16*FP16

Multiple panels
compute intensity:
85.3 ops/load

DEGREES OF FREEDOM

4 degrees of freedom + zero-cost MX

Four orthogonal knobs let software and hardware each express intent — VLEN, λ, LMUL, and VL shape every point from embedded streaming to HPC bulk. Where SME and AMX bolted microscaling on later, Zvvm gets it from one already-present bit: no new opcodes, no new registers, no new modes.

Variable	Range	Purpose
VLEN	64 → 64K	Binary-compatible throughput scaling
λ	1 → 64	Tile geometry; dot-product at λ = VL
LMUL	1 → 8	Trade register groups for K_eff depth
VL	K_eff → max	Embedded streaming → full HPC tile
W	1, 2, 4, 8	Precision ladder: INT8→INT32, FP16→FP32

Microscaling, free encoding

vm = 0 aliases the FP MAC opcode onto the MX decode path.

- No new opcodes
- No new registers
- No new modes
- MXFP8 / MXFP4 / MXINT8 + mixed

THE RISK

Scaling the ecosystem with the hardware.

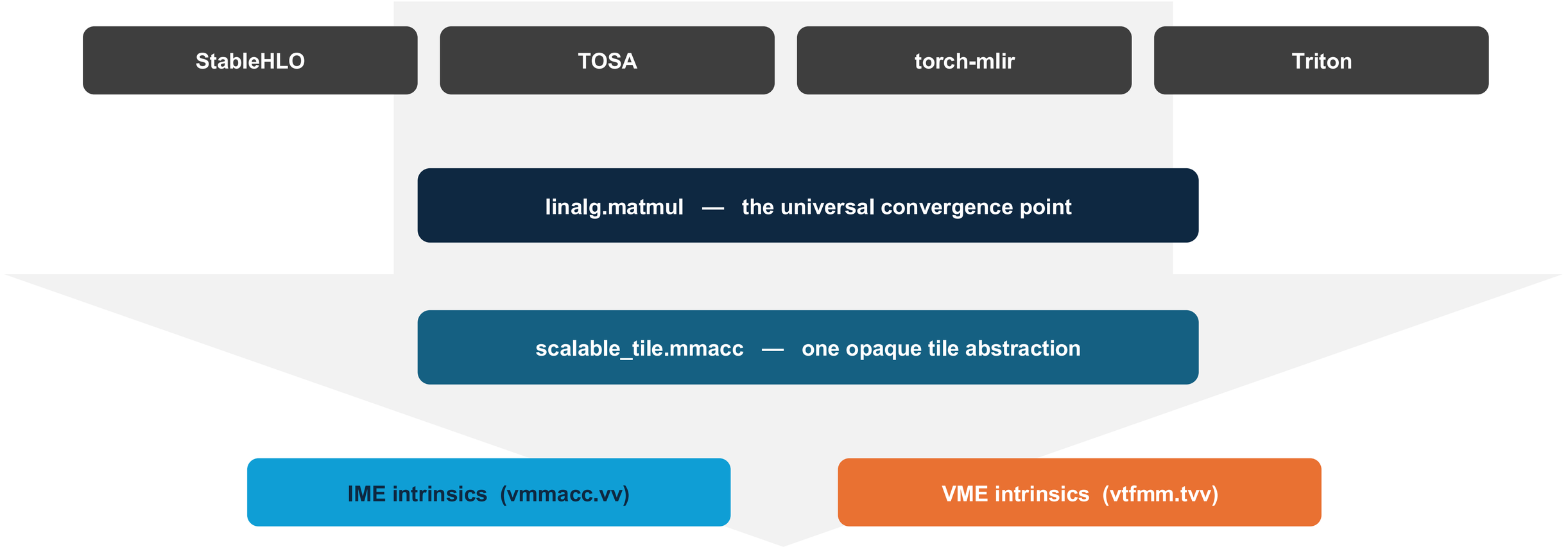
If IME and VME diverge in the toolchain, RISC-V repeats that mistake of SME and AMX.
To avoid it, the software ecosystem must scale with the hardware instead of forking.

The answer is not to pick a winner.
It is to make the hardware choice invisible to software.

THE UNIFICATION

One MLIR abstraction targets both ISAs

Every ML frontend lowers to a common abstraction: the linalg.matmul primitive.
If we lower it once to a tile op that abstracts both extensions, the cost of differentiation is abstracted away.



TWO BACKENDS VALIDATE THE ABSTRACTION

Every difference forces a design choice. The MLIR dialect must abstract these choices consistently.

Every abstraction exists because one ISA needs it and the other handles it differently.

What differs	IME (Zvvm)	VME (Zvt)	Dialect resolves by…
Tile storage	V register file	mt0..mt15 + mstatus.MS	opaque tile type
Configuration	vsetvl (one CSR)	msetmtype + msettm/k/n	config as attribute
Aspect ratio λ	1…64 — scales w/ VLEN	none (fixed TE×TE)	λ attr; gate λ≠1
Widening W	opcode, W=1,2,4,8	mtwiden, W=1,2,4	w attr; gate W=8
K depth	unbounded (λ·W·LMUL)	KMAX ≤ 4	capability query
Register alloc	standard regalloc	tile alloc + punning + spill	backend-local pass
Context save	none (V regs)	mstatus.MS, vtdiscard	backend prologue

SAME IR, TWO LOWERINGS

The compiler provides the resilient abstraction

```
scalable_tile.mmacc  %c, %a, %b      // one op, type-dispatched
```

IME (Zvvm) lowering

```
vsetvl   x0, vtype(sew=16, λ=2, lmul=1)
vmtl.v   v0,  A_ptr, lda
vmtl.v   v8,  B_ptr, ldb
vmtl.v   v16, C_ptr, ldc
vfmmacc.vv v16, v0, v8
vmts.v   v16, C_ptr, ldc
```

→ no tile allocator, no context CSR

VME (Zvt) lowering

```
msetmtype mtype(tew=16,mtwiden=1),vtype
vtle16.v  mt0, A_ptr, TSS(row, ki)
vtle16.v  mt2, B_ptr, TSS(row, ni)
vtle16.v  mt4, C_ptr, TSS(row, mi)
vtfmm.tvv mt4, mt0, mt2
vtse16.v  mt4, C_ptr, TSS(row, mi)
```

+ tile-register alloc, + mstatus.MS mgmt

TAKEAWAYS

All the scaling. **One software abstraction.**

01 RISC-V matrix support is no longer a roadmap item.
The **next step is the software abstraction** that accelerates the ecosystem enablement.

02 RISC-V deliberate specified a family of four specifications for different design points: covering **from microcontrollers to supercomputer**.
All can be wrapped under **a single software abstraction**.

03 The Zvmm specification is **in Internal Review**.
We expect ratification later this year.

SOFTWARE ENABLEMENT DELIVERED AS A STANDARDIZATION ARTIFACT


QEMU model


GCC intrinsics


BLAS kernels


Test suite

Co-developed with the specification.
Lowering the barrier for implementers, accelerating the upstream adoption in open-source, and making the specification testable from the start.