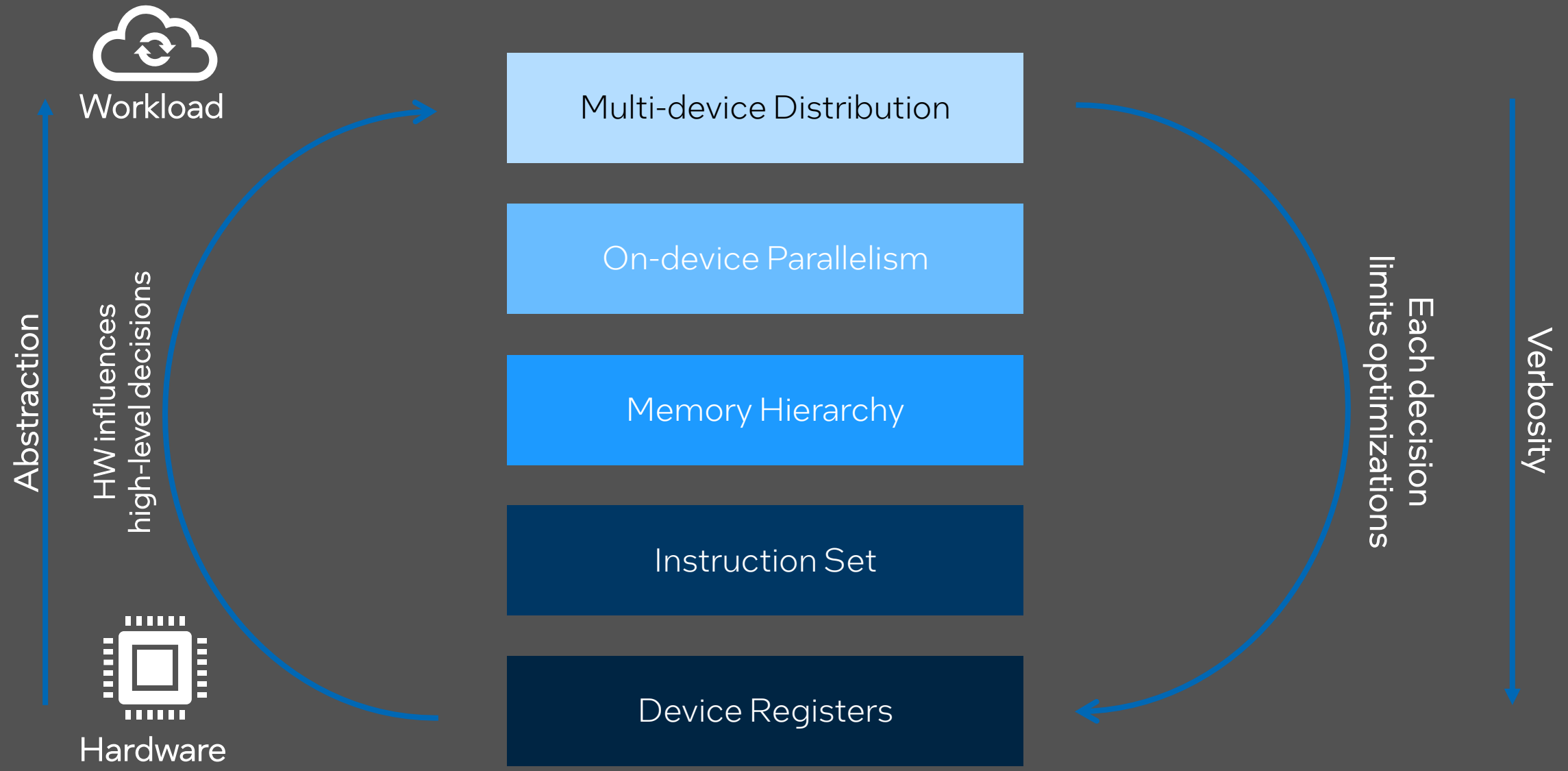
MLIR GEMM Code Generation

Adam Siemieniuk, Renato Golin, Rolf Morel

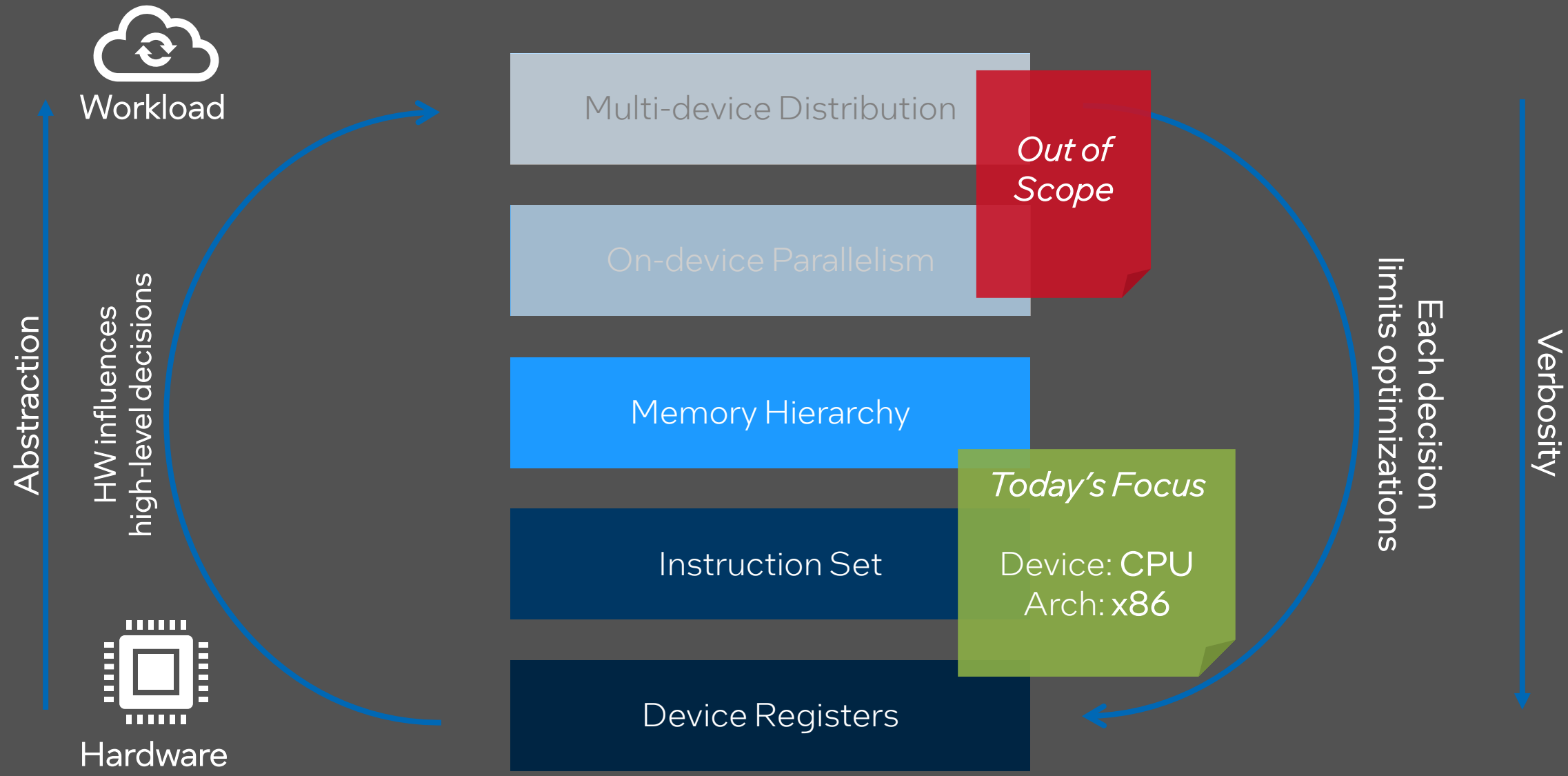
Disclaimer

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- No computer system can be absolutely secure.
- Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.
- Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling and provided to you for informational purposes.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- Intel, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

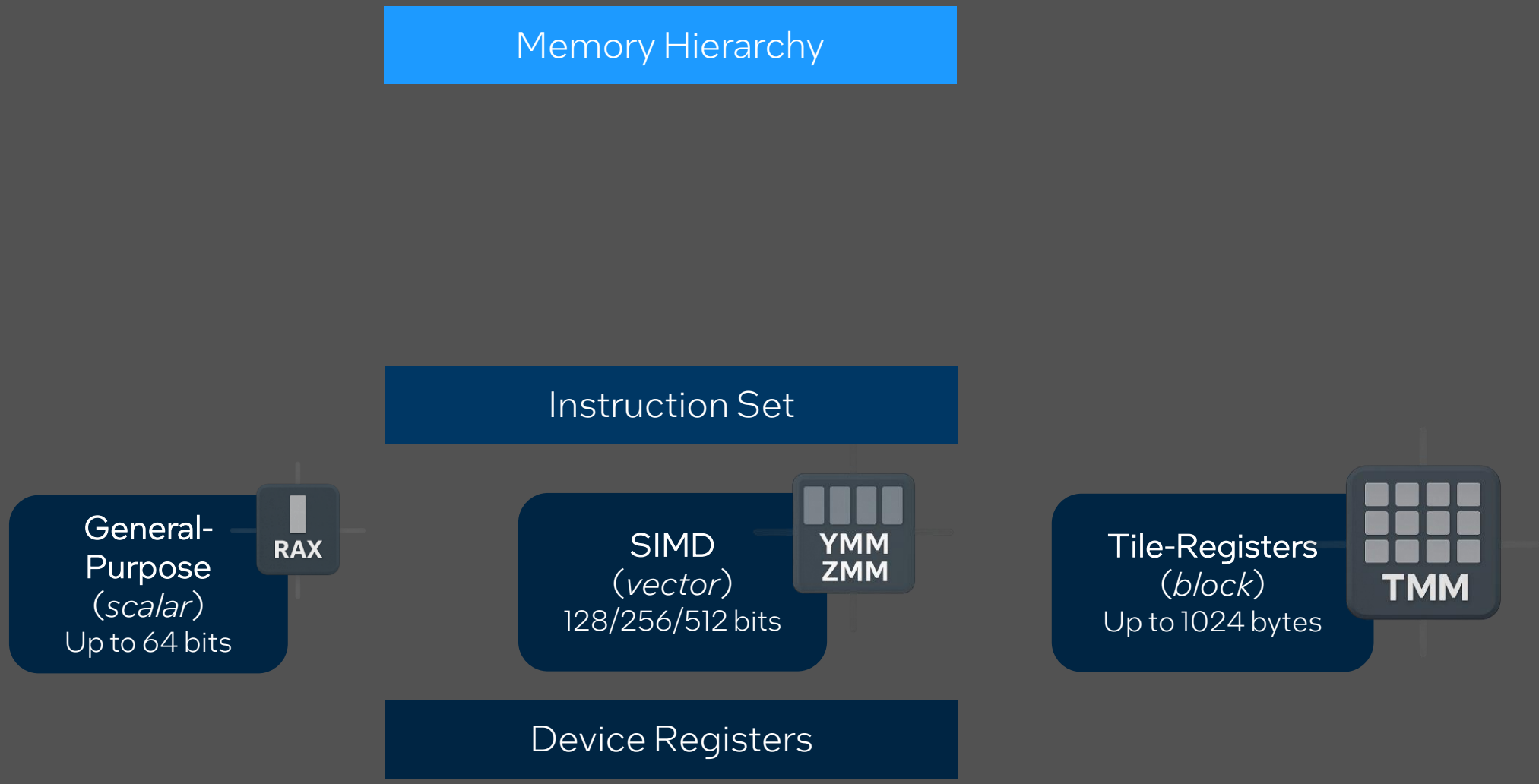
Hardware-Software Feedback Loop



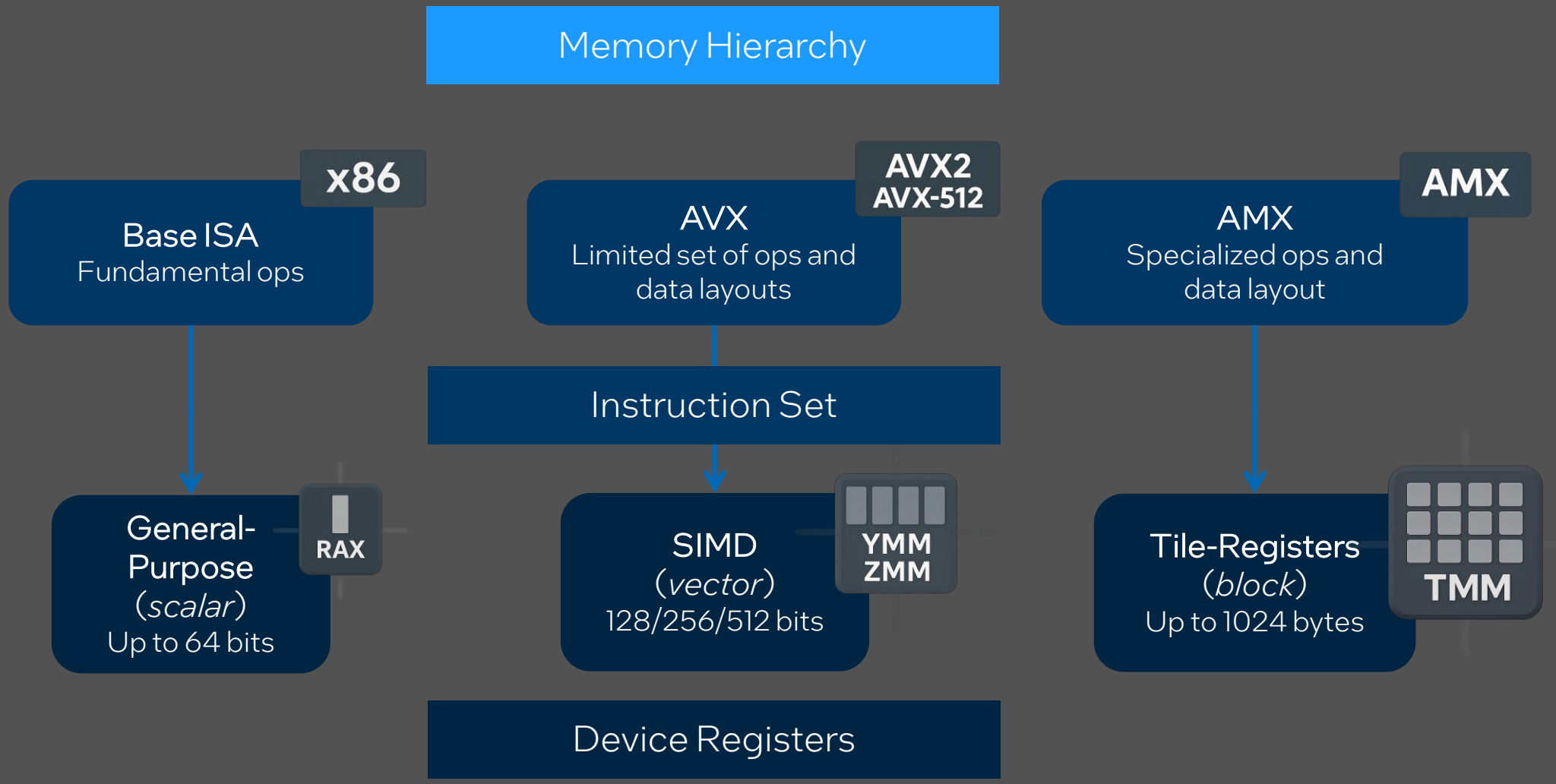
Hardware-Software Feedback Loop



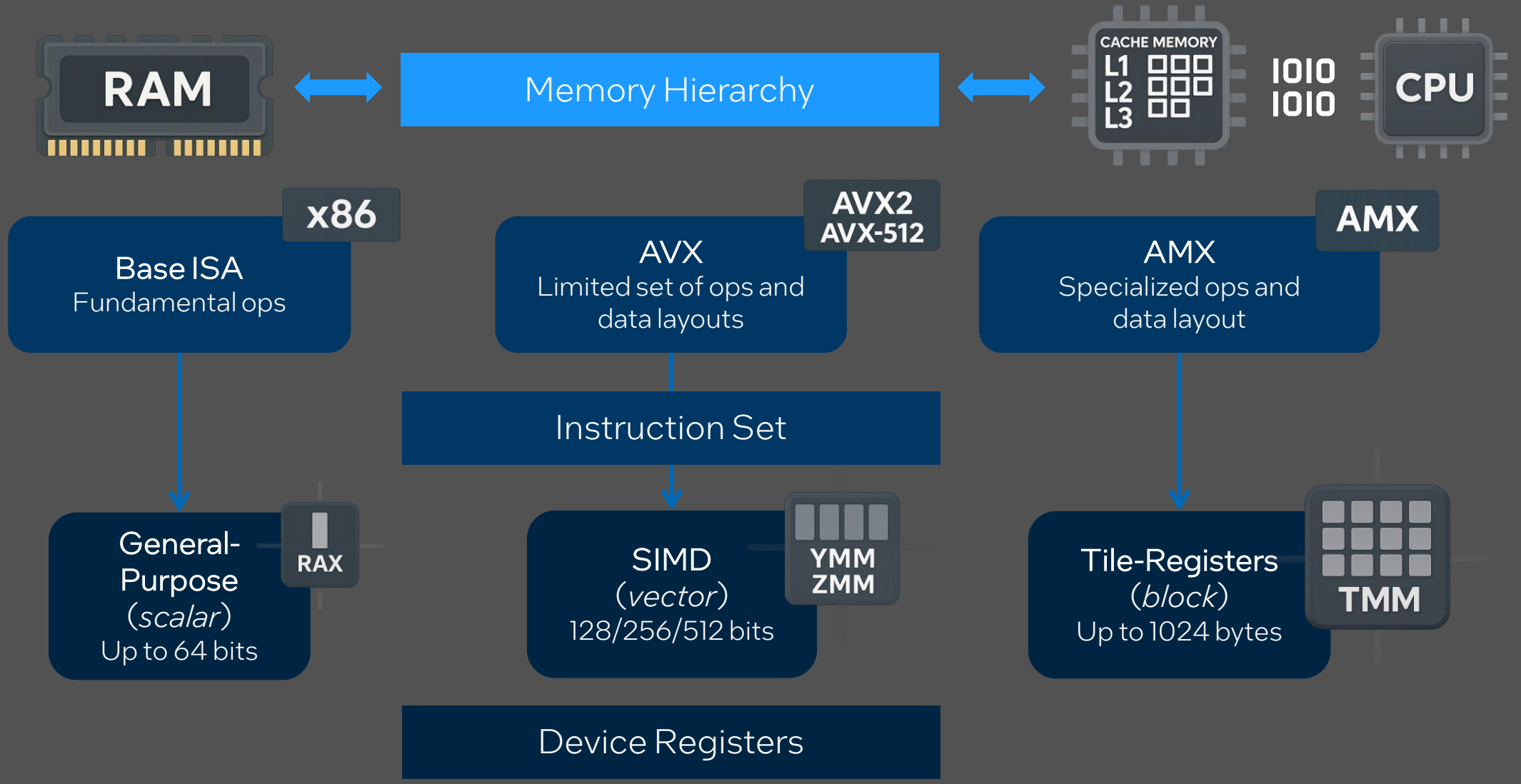
Hardware Building Blocks



Hardware Building Blocks



Hardware Building Blocks



MLIR ➡️ Memory

Memory Hierarchy

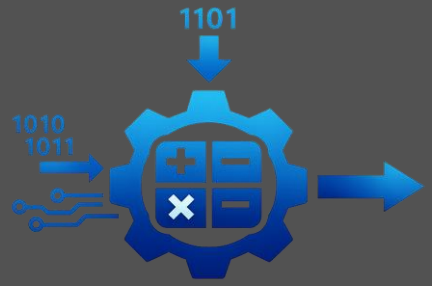
Matrix Multiply:

```
C += A * B

A <M x K>
B <K x N>
C <M x N>
```

B

A



C

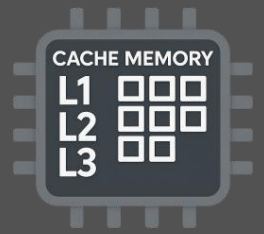
linalg.matmul

MLIR ➡️ Memory

Matrix Multiply:
`C += A * B`

`A <M x K>`
`B <K x N>`
`C <M x N>`

Tile
Upper
Bound



Memory Hierarchy

Tile

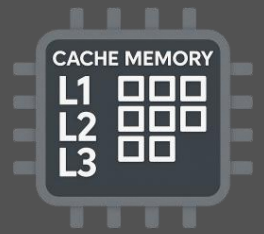
Block Pack

Fuse

Tile
Lower
Bound

MLIR ➡️ Memory

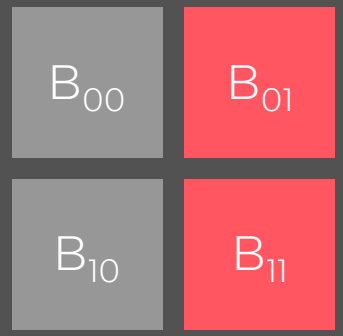
Tile
Upper
Bound

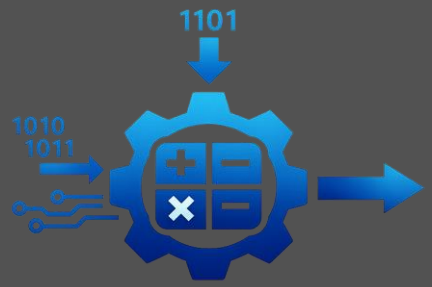


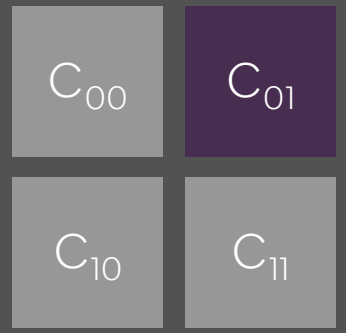
Memory Hierarchy

Matrix Multiply:
`C += A * B`

`A <Mb x K>`
`B <K x Nb>`
`C <Mb x Nb>`







linalg.matmul

Tile
Block Pack
Fuse

Tile
Lower
Bound

MLIR ➡️ Registers

Instruction Set

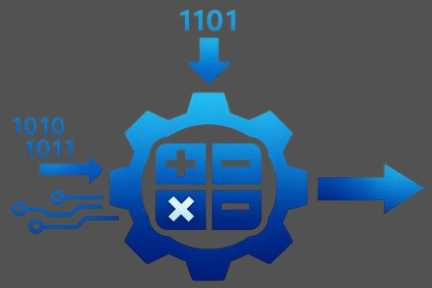
MICRO KERNEL

Matrix Multiply:

```
C += A * B

A <Mb x K>
B <K x Nb>
C <Mb x Nb>
```

B_{K1}



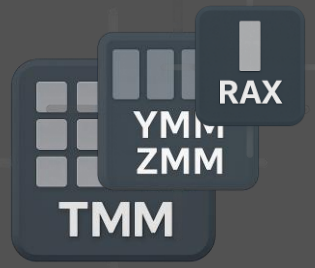
A_{0K}

C_{01}

linalg.matmul

MLIR ➡️ Registers

Tile
Upper
Bound

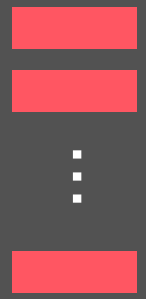


Instruction Set

Matrix Multiply:

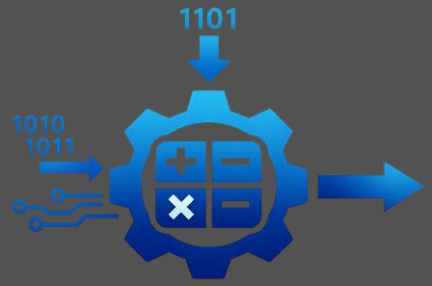
```
C += A * B

A <mr x k>
B <k x nr>
C <mb x nr>
```



$B_{[0:k:K]0}$

$A_{1[0:k:K]}$



linalg.matmul

C_{00}	C_{01}
C_{10}	C_{11}
C_{20}	C_{21}

Tile

Loop
Interchange

Loop Peel

Tile
Lower
Bound

MLIR ➡️ Instructions

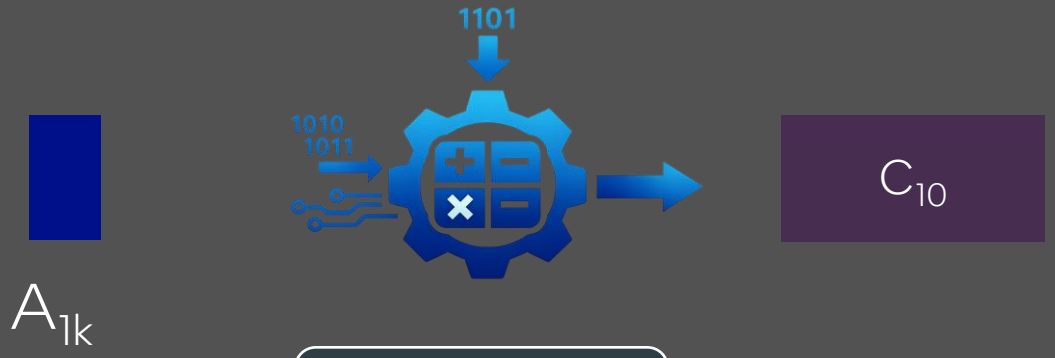
Instruction Set

Matrix Multiply:
`C += A * B`

AVX2
AVX-512

```
A <8 x 2>  BF16
B <2 x 32> BF16
C <8 x 32> F32
```

B_{k0}



linalg.matmul

MLIR ➡️ Instructions

Tile Target

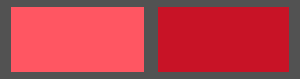
x86
AVX-512
AMX

Instruction Set

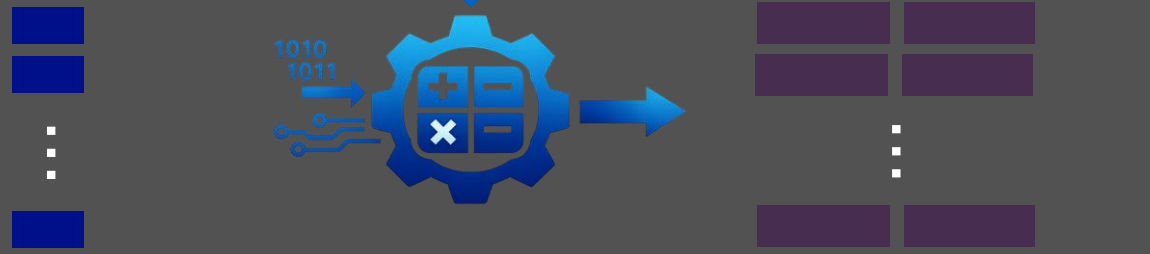
Matrix Multiply:
`C += A * B`

AVX2
AVX-512

```
A <1 x 2>   BF16
B <2 x 16>  BF16
C <1 x 16>  F32
```



$B_{k[0:16:32]}$



$A_{[0:1:8]k}$

linalg.matmul

$C_{[0:1:8][0:16:32]}$

Tile

Loop Unroll

x86
AVX-512

MLIR ➡️ Instructions

Tile Target

x86
AVX-512
AMX

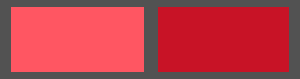
Instruction Set

NANO KERNEL

Matrix Multiply:
`C += A * B`

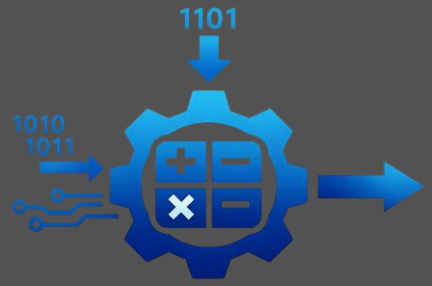
AVX2
AVX-512

```
A <1 x 2>   BF16
B <2 x 16>  BF16
C <1 x 16>  F32
```

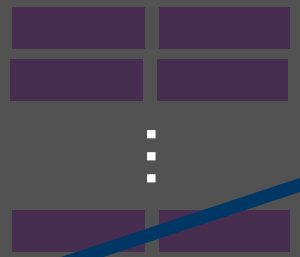


$B_{k[0:16:32]}$

$A_{[0:1:8]k}$



vector.contract



$C_{[0:1:8][0:16:32]}$

Tile

Loop Unroll

Vectorize

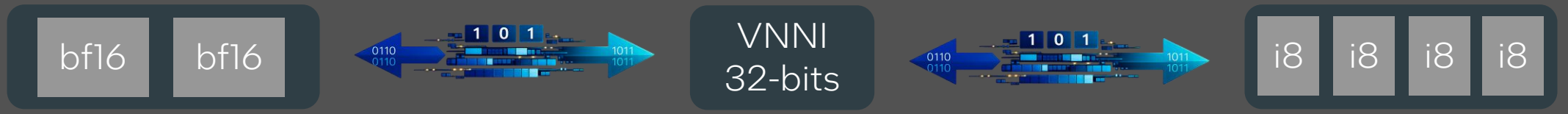
Loop Hoist

x86
AVX-512

VNNI Data Layout

AVX2
AVX-512

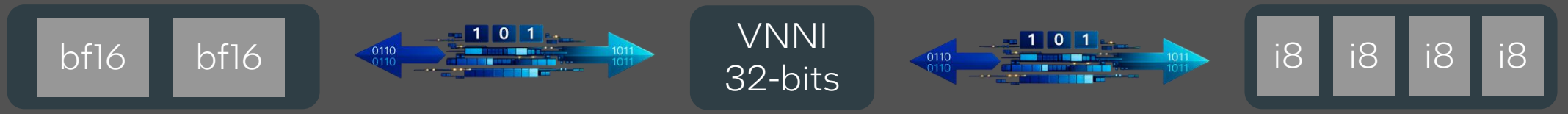
AMX



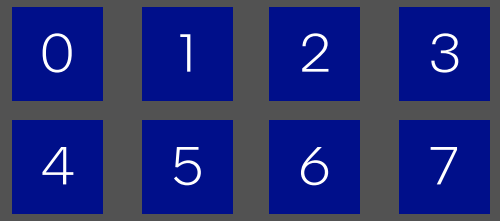
VNNI Data Layout

AVX2
AVX-512

AMX



A <2 x 4> BF16



A <2 x (2 x 2)> BF16

VNNI Data Layout

AVX2
AVX-512

AMX

bf16 | bf16

VNNI
32-bits

i8 | i8 | i8 | i8

A <2 x 4> BF16

0	1	2	3
4	5	6	7

A <2 x (2 x 2)> BF16

0	1	2	3
4	5	6	7

B <4 x 2> BF16

0	1
4	5
2	3
6	7

B <(2) x 2 x (2)> BF16

0	4	1	5
2	6	3	7

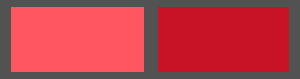
MLIR ➡️ AVX-512 BF16

AVX-512 Instruction Set

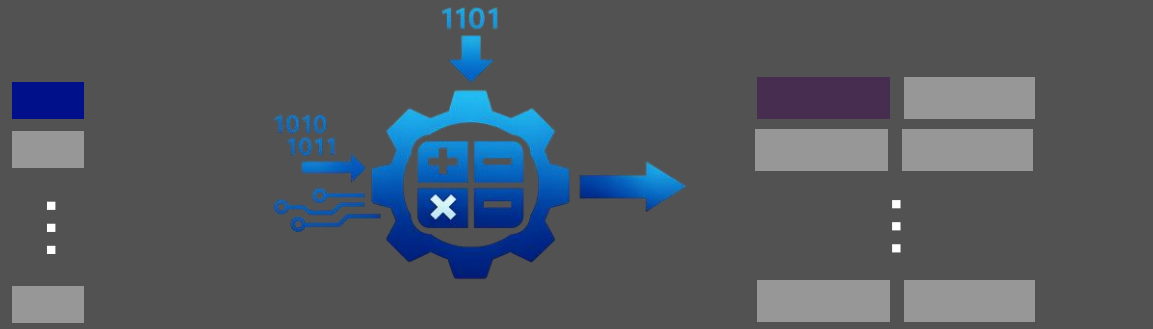
Matrix Multiply:

```
C += A * B
```

```
A <1 x 2>   BF16
B <2 x 16>  BF16
C <1 x 16>  F32
```

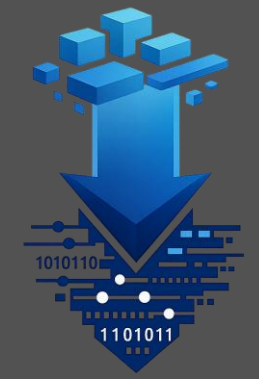


$B_{k[0:16:32]}$



$A_{[0:1:8]k}$

$C_{[0:1:8][0:16:32]}$



x86

AVX-512

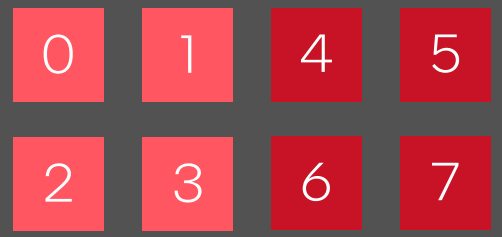
MLIR ➡️ AVX-512 BF16

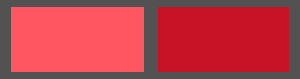
AVX-512 Instruction Set

Matrix Multiply:
`C += A * B`

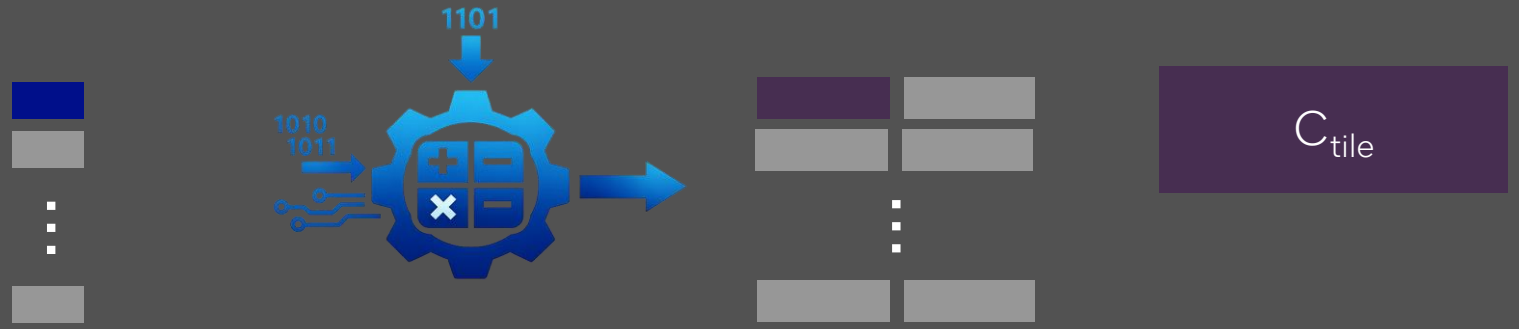
```
A <1 x 2>  BF16
B <2 x 16> BF16
C <1 x 16> F32
```

`2 x B <2 x 2> BF16`





$B_{k[0:16:32]}$



$A_{[0:1:8]k}$

vector.contract

$C_{[0:1:8][0:16:32]}$

C_{tile}

`A <1 x 2> BF16`

NANO KERNEL

x86

AVX-512

MLIR ➡️ AVX-512 BF16

AVX-512 Instruction Set

A <1 x 2> BF16

2 x B <2 x 2> BF16

x86 Patterns

x86

AVX-512

MLIR ➡️ AVX-512 BF16

AVX-512 Instruction Set

A <1 x 2> BF16

x86 vector.bitcast

x86 vector.broadcast

2 x B <2 x 2> BF16

AVX-512 vector.shuffle

NANO KERNEL

x86 Patterns

x86

AVX-512

MLIR ➡️ AVX-512 BF16

AVX-512 Instruction Set

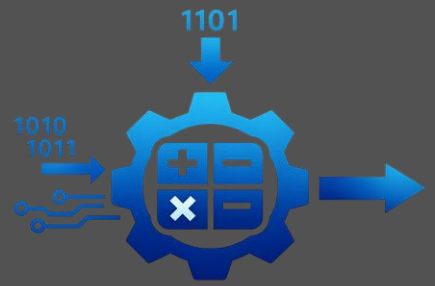
A <1 x 2> BF16

x86 vector.bitcast

x86 vector.broadcast

2 x B <2 x 2> BF16

AVX-512 vector.shuffle



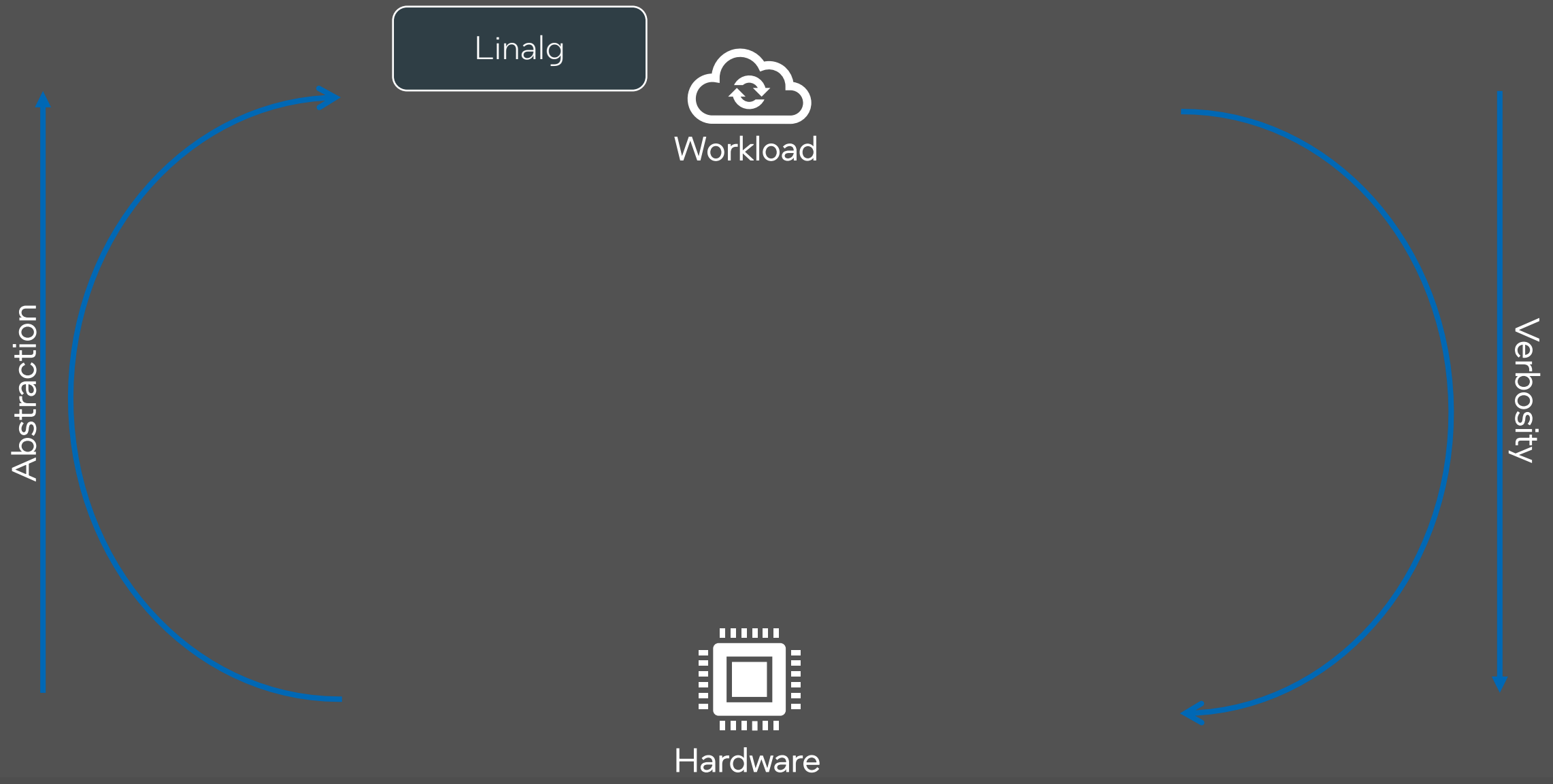
C_{tile}

AVX-512 x86.avx512.dot

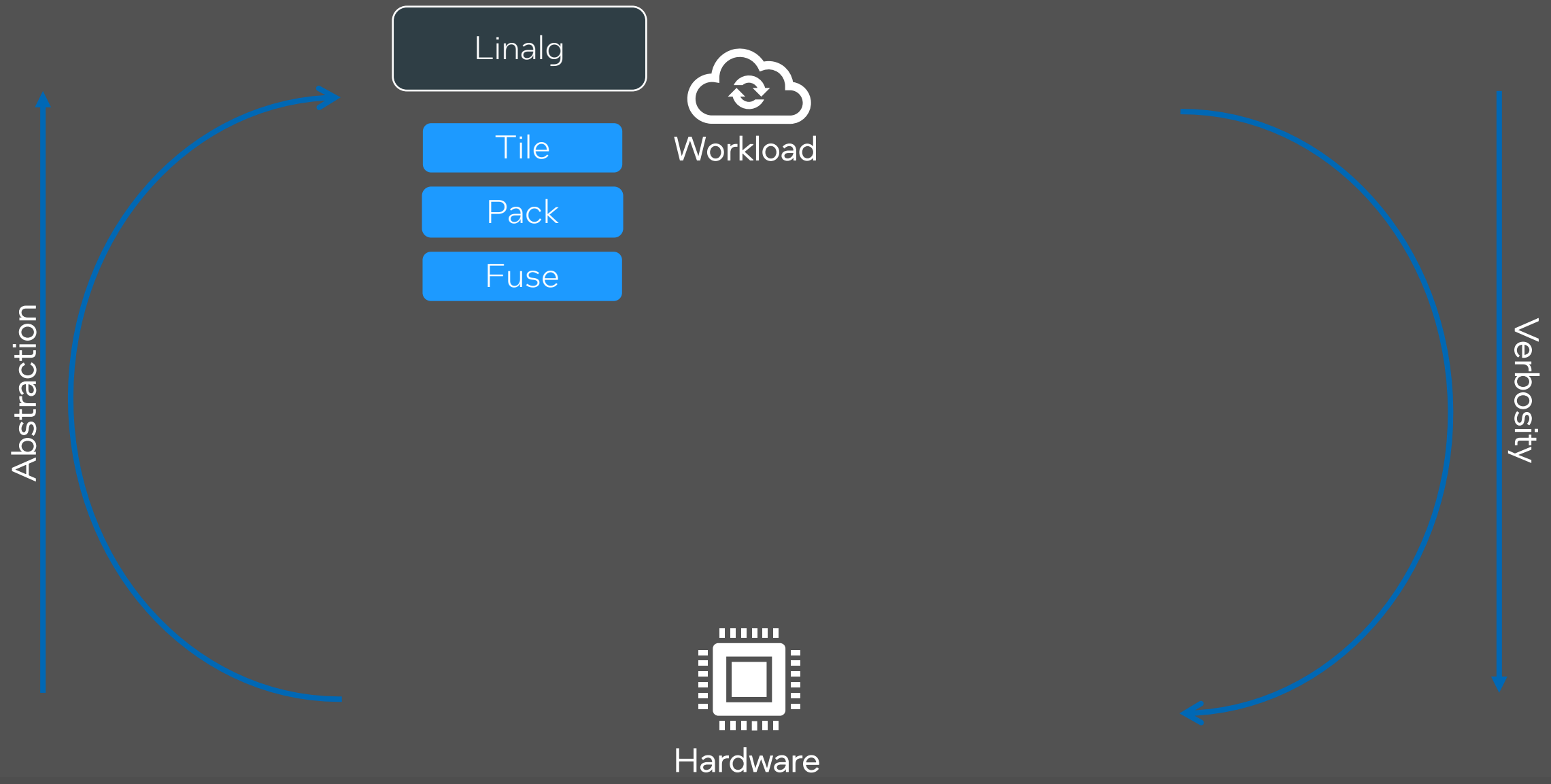
x86 Patterns

x86
AVX-512

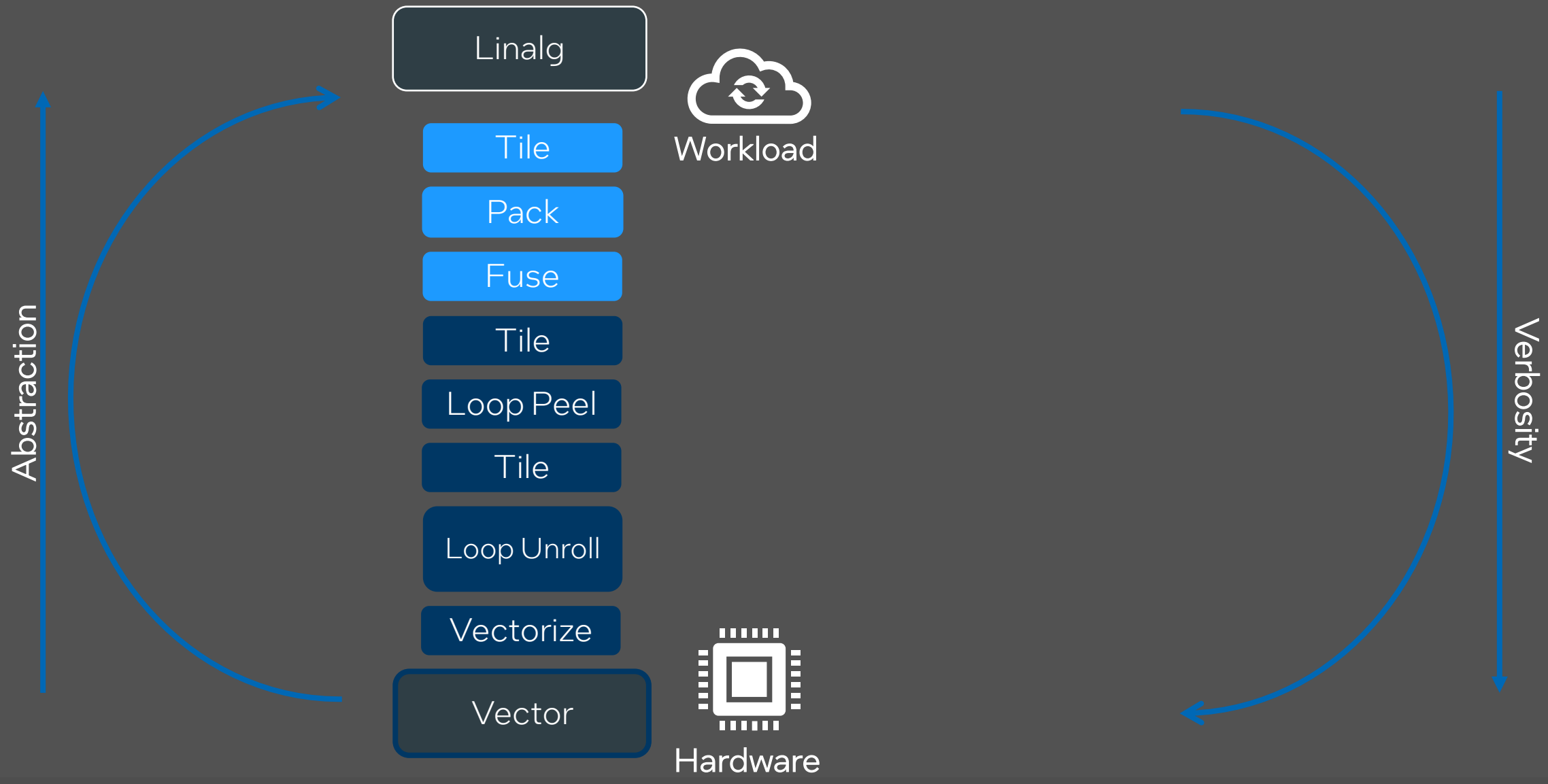
MLIR ➡️ x86



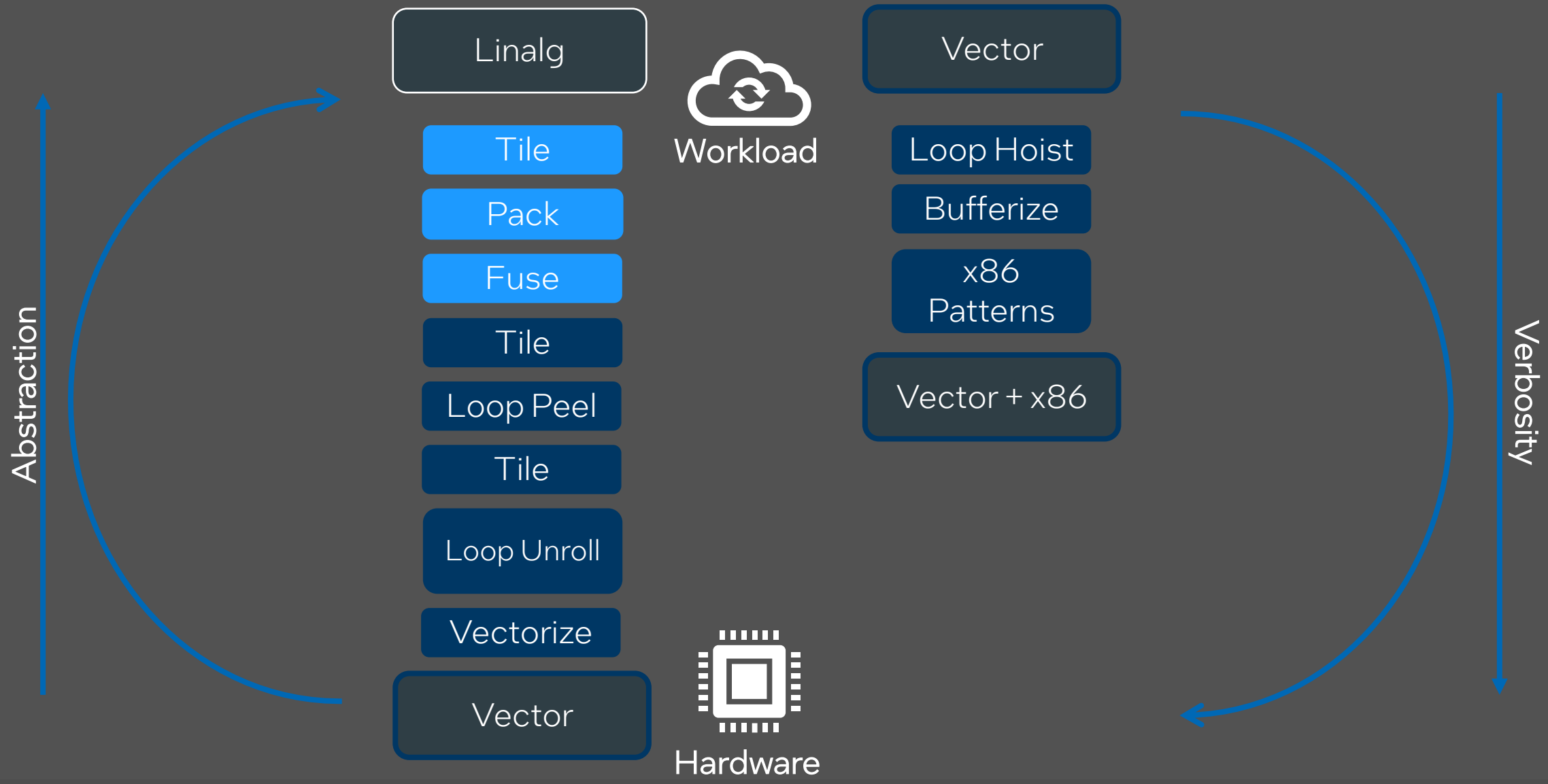
MLIR ➡️ x86


Linalg
Tile
Pack
Fuse
Workload
Abstraction
Verbosity
Hardware

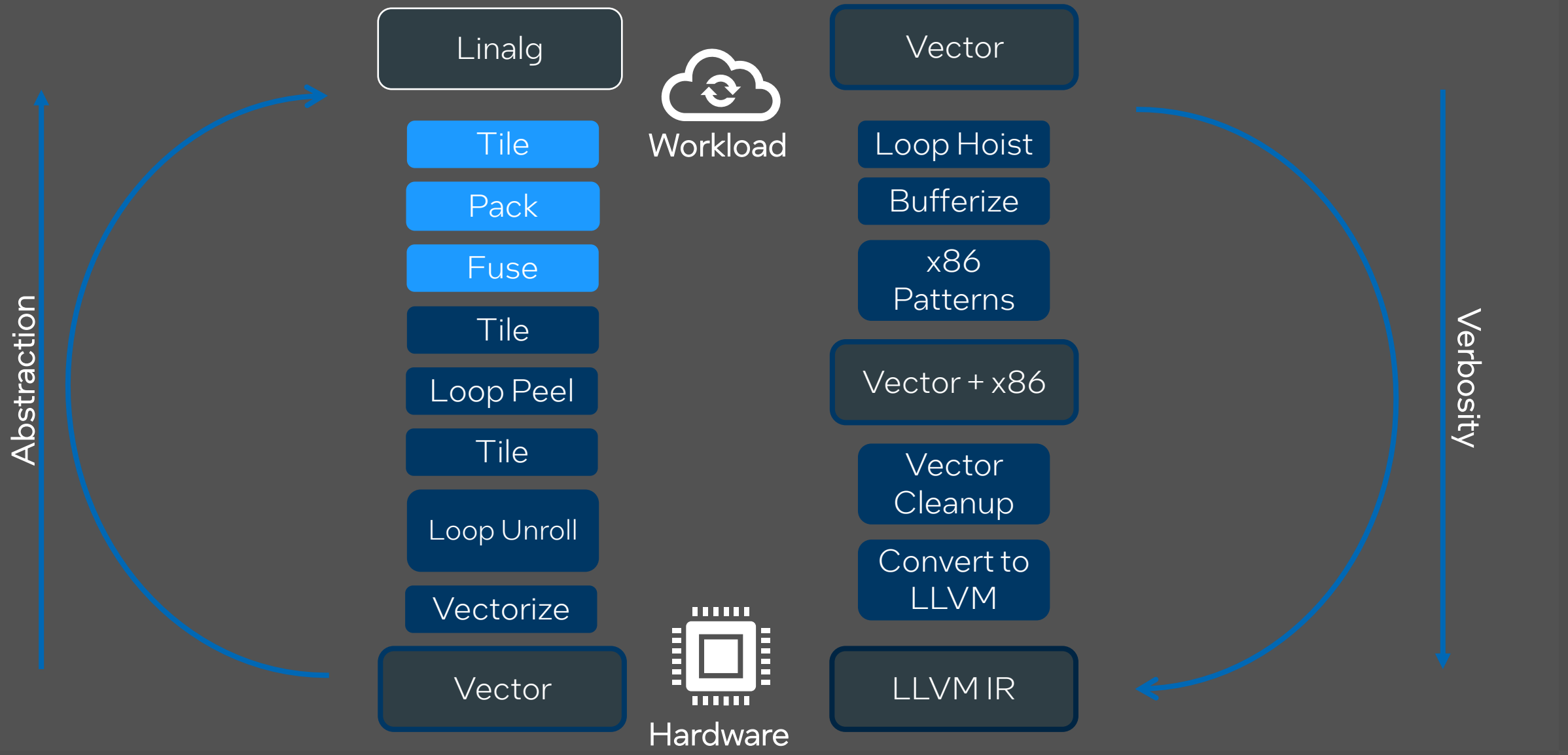
MLIR ➡️ x86


Linalg
Tile
Pack
Fuse
Tile
Loop Peel
Tile
Loop Unroll
Vectorize
Vector
Workload
Hardware
Abstraction
Verbosity

MLIR ➡️ x86


Linalg
Tile
Pack
Fuse
Tile
Loop Peel
Tile
Loop Unroll
Vectorize
Vector
Workload
Hardware
Vector
Loop Hoist
Bufferize
x86 Patterns
Vector + x86
Abstraction
Verbosity

MLIR ➡️ x86


Linalg
Tile
Pack
Fuse
Tile
Loop Peel
Tile
Loop Unroll
Vectorize
Vector
Workload
Hardware
Vector
Loop Hoist
Bufferize
x86 Patterns
Vector + x86
Vector Cleanup
Convert to LLVM
LLVM IR
Abstraction
Verbosity

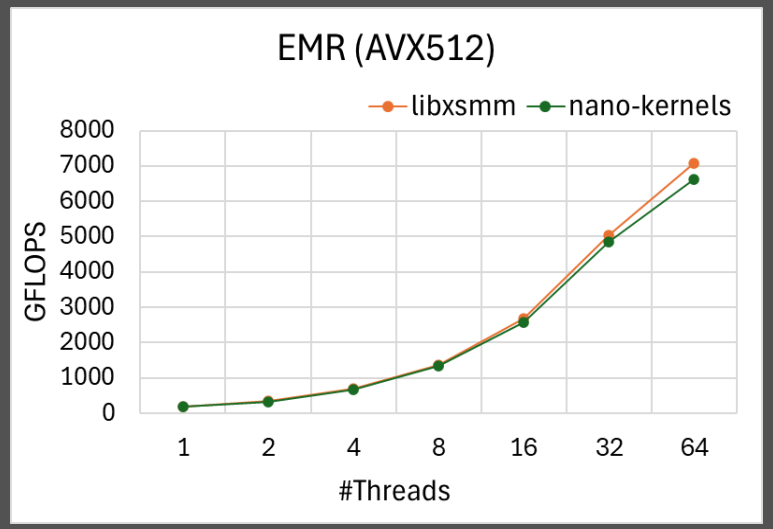
Benchmarks - Setup

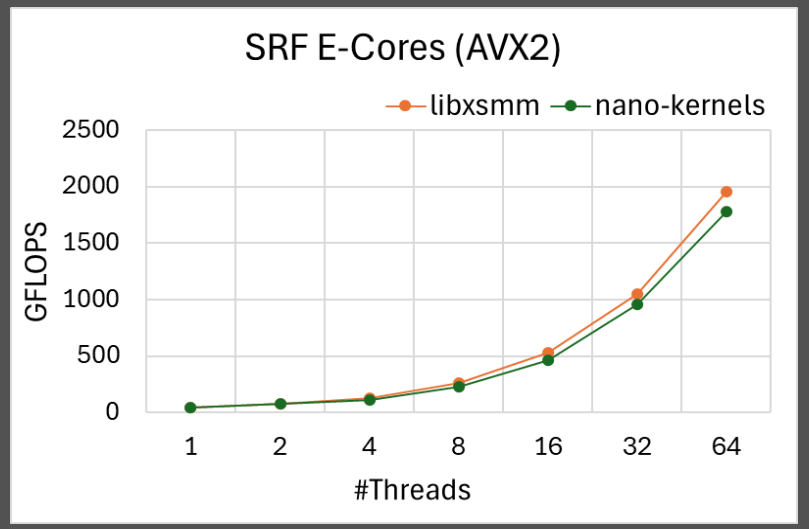
- TPP-MLIR (libxsmm)
  - libxsmm library for microkernels
  - Perf better than Intel oneMKL and oneDNN

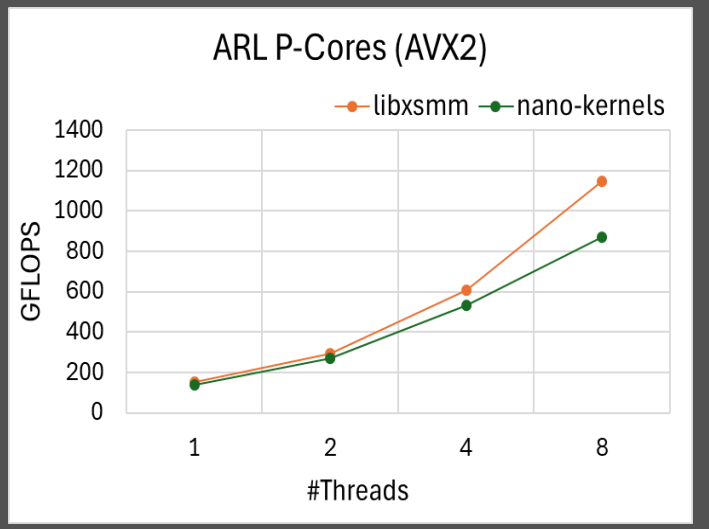
- TPP-MLIR (nanokernels)
  - MLIR target-specific code generation
  - Self-contained, no external libraries

- Methodology
  - Measure only benchmark loop (no init, others…)
  - https://github.com/libxsmm/tpp-mlir/tree/main/benchmarks

Benchmarks – F32



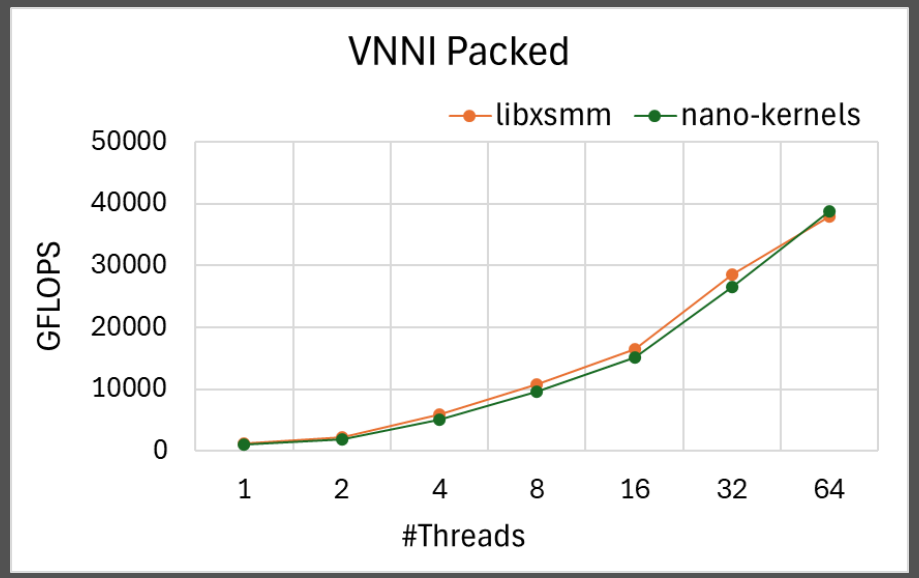


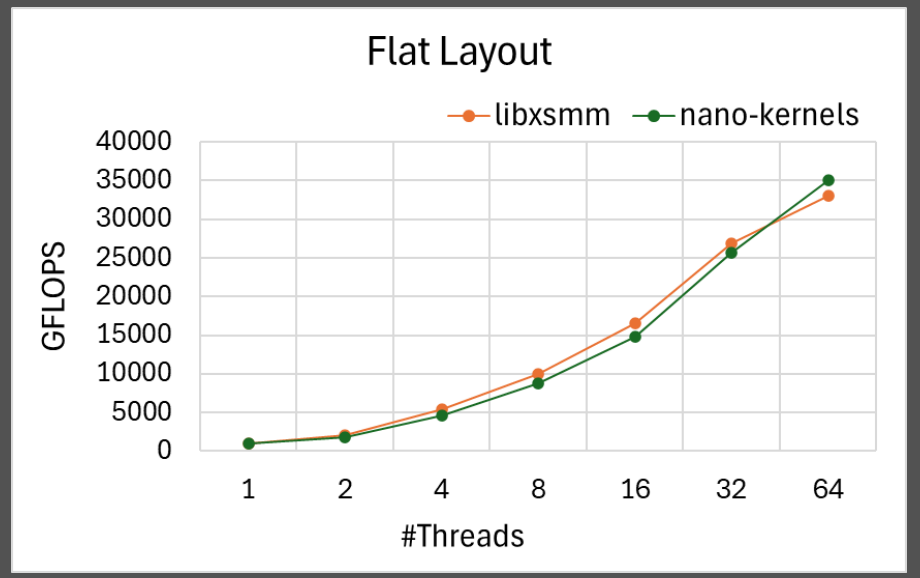


- Register Tile sizes:
  - AVX512 – M:8 N:32 K:1
  - AVX2 - M:4 N:16 K:1

- ~93% and ~91% of libxsmm performance on **EMR** and **SRF**
- **ARL** dip with 8 threads due to **parallelization scheme**

Benchmarks – BF16 AMX (EMR)

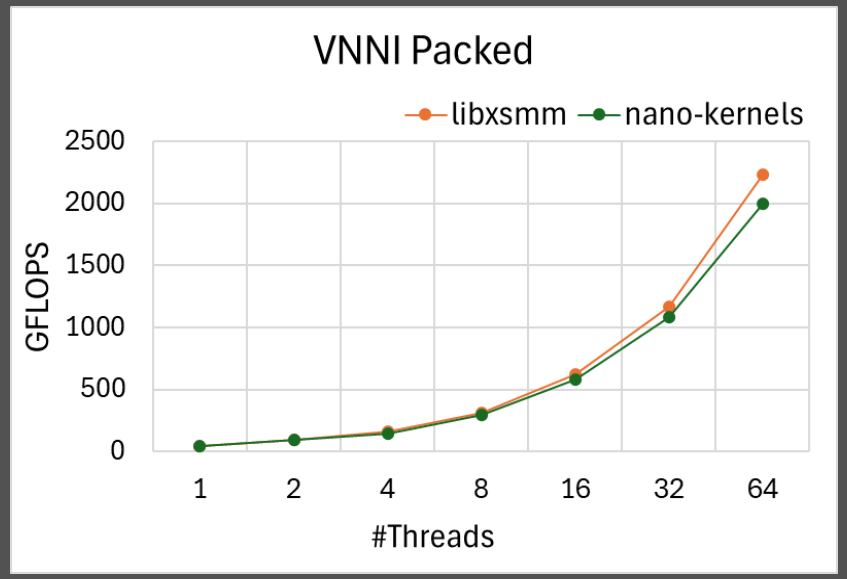


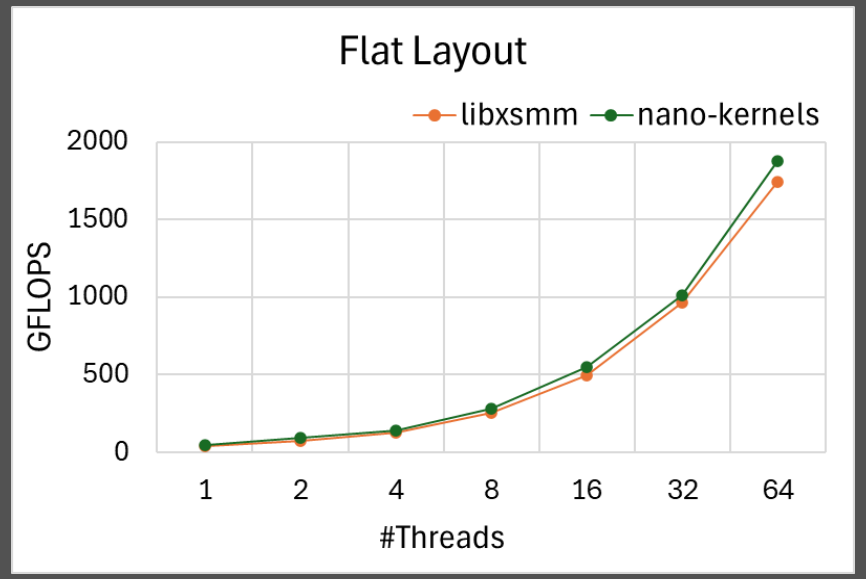


- Register Tile size:
  - M:32 N:32 K:32

- Flat layout input is shuffled to VNNI packed
- ~91% and ~93% libxsmm performance for **VNNI packed** and **Flat** input.

Benchmarks – BF16 AVX2 (SRF E-Cores)





- Register Tile size:
  - M:2 N:32 K:2 (VNNI) K:1(Flat),

- No packing from Flat to VNNI - use of BF16 packed instructions
- Nano-kernels outperform libxsmm by ~15% on Flat inputs – due to libxsmm packing input to VNNI

Try It Yourself!

Lighthouse Example

https://github.com/llvm/lighthouse/blob/main/examples/cpu/x86/matmul.py

Upstreaming

Deeper Dive

White Paper

Thangamani et al.
Library Liberation (2025)
https://arxiv.org/abs/2511.13764


intel