μ-ORCA: Optimizing Acceleration for Microsecond-Scale Deep Neural Network Inference on ACAP

Great Lakes Symposium on VLSI (GLSVLSI) 2026

Shixin Ji, Jinming Zhuang, Zhuoping Yang, Xingzhen Chen, Wei Zhang, Peipei Zhou

Brown University

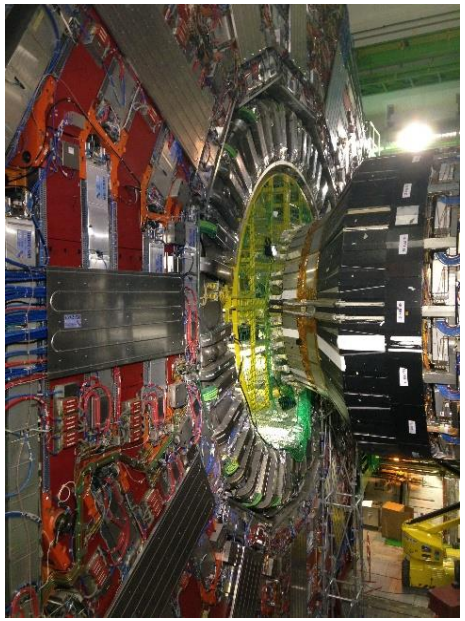
shixin_ji@brown.edu
peipei_zhou@brown.edu
https://peipeizhou-eecs.github.io/

BROWN

While Everyone is talking about LLMs…

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…

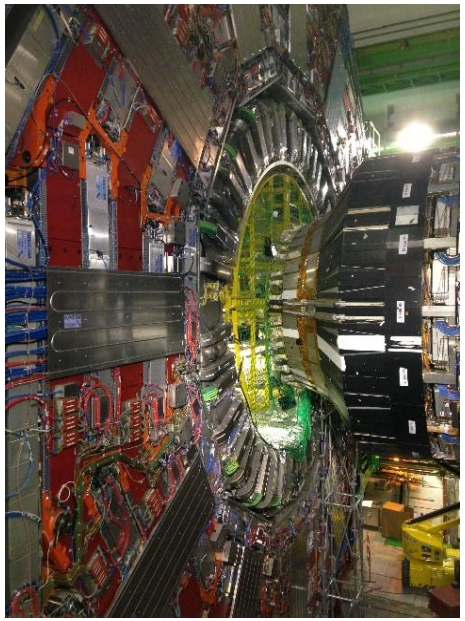


Large Hadron Collider(LHC)[1]

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…

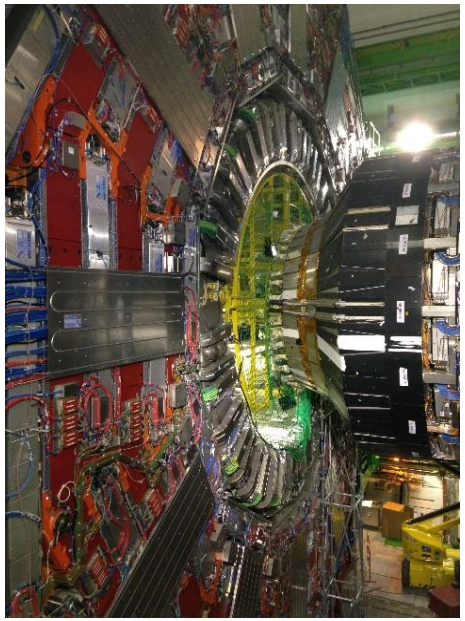
Useful physics data



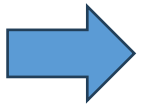
Large Hadron Collider(LHC)[1]

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…

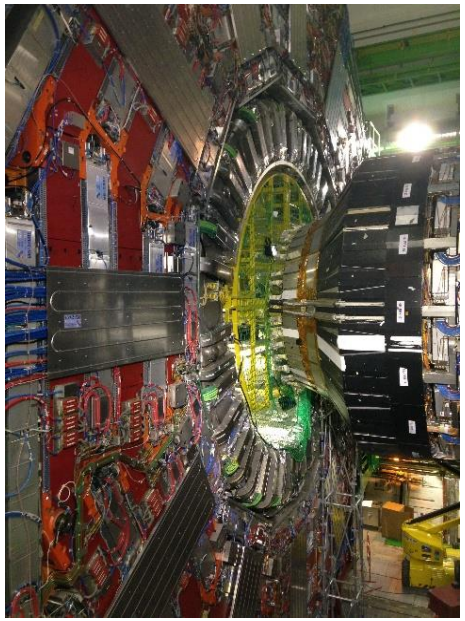


Large Hadron Collider(LHC)[1]

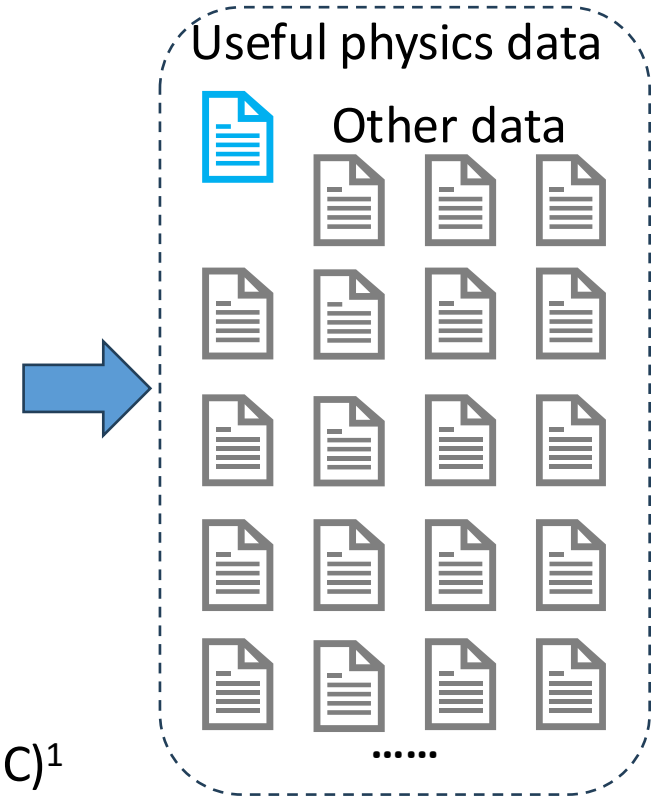


[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…



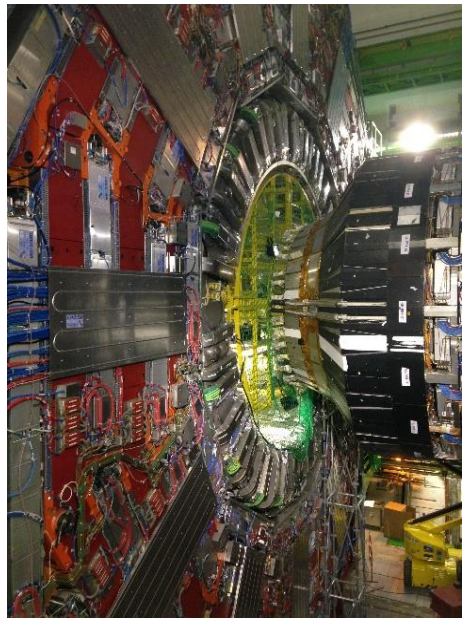
Large Hadron Collider(LHC)[1]



40 MHz event gen freq.

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…

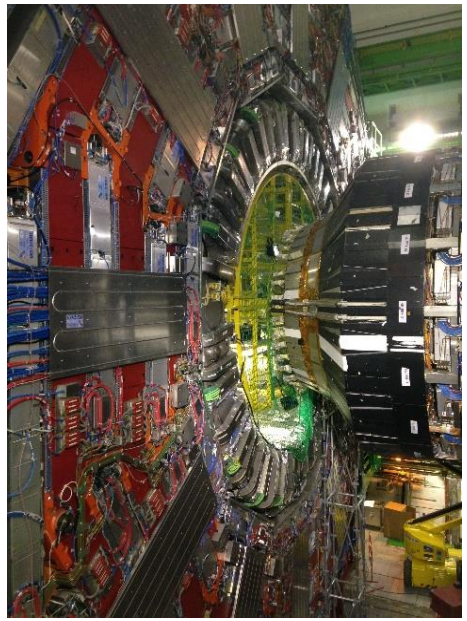


Large Hadron Collider(LHC)[1]

40 MHz event gen freq.
Several TB data per second
Can not be analyzed by conventional computers

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…

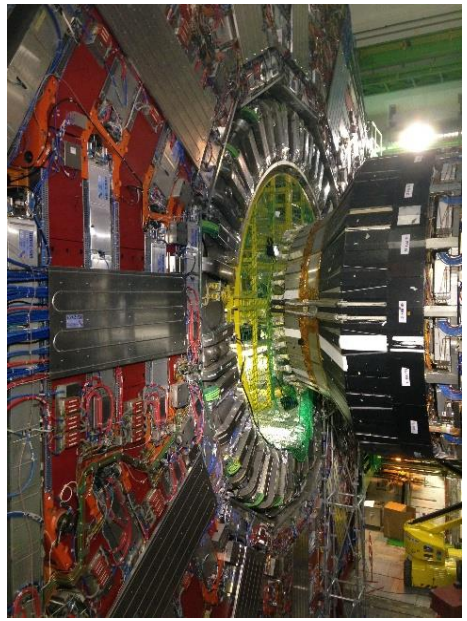


Large Hadron Collider(LHC)[1]

40 MHz event gen freq.
Several TB data per second
Can not be analyzed by conventional computers

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…



Large Hadron Collider(LHC)[1]

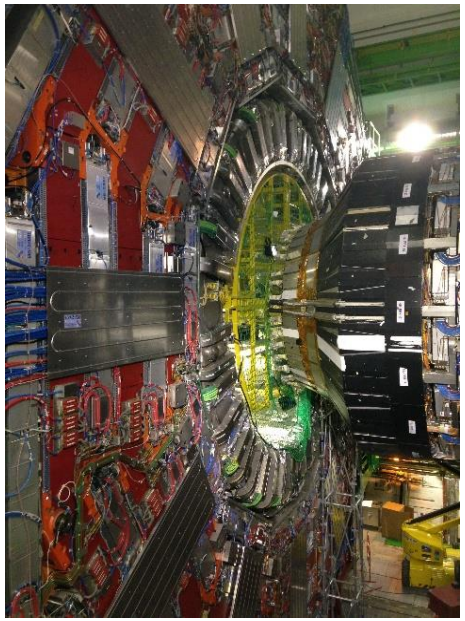
40 MHz event gen freq.
Several TB data per second
Can not be analyzed by conventional computers

40 MHz

L1 trigger system

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…



Large Hadron Collider(LHC)[1]

40 MHz event gen freq.
Several TB data per second
Can not be analyzed by conventional computers

40 MHz

L1 trigger system

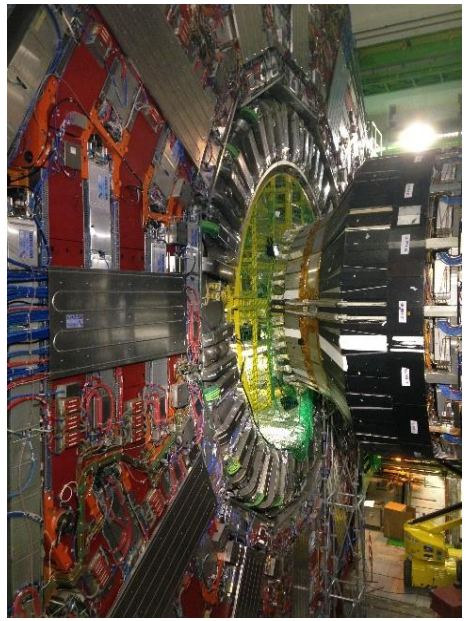
100 KHz

High-Level Trigger

Software system
30,000 CPU cores[2]

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…



Large Hadron Collider(LHC)[1]

40 MHz event gen freq.
Several TB data per second
Can not be analyzed by conventional computers

40 MHz

L1 trigger system

400x reduction
3.8 µs to decide after collision
Next-gen: more data, require ~1 µs

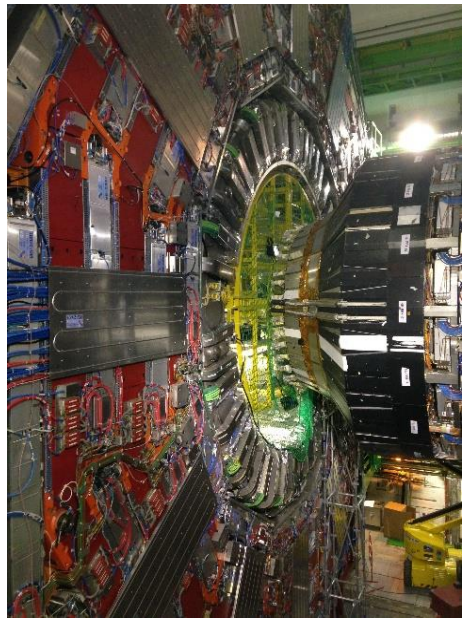
100 KHz

High-Level Trigger

Software system
30,000 CPU cores[2]

[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

While Everyone is talking about LLMs…



Large Hadron Collider(LHC)[1]

40 MHz event gen freq.
Several TB data per second
Can not be analyzed by conventional computers

40 MHz

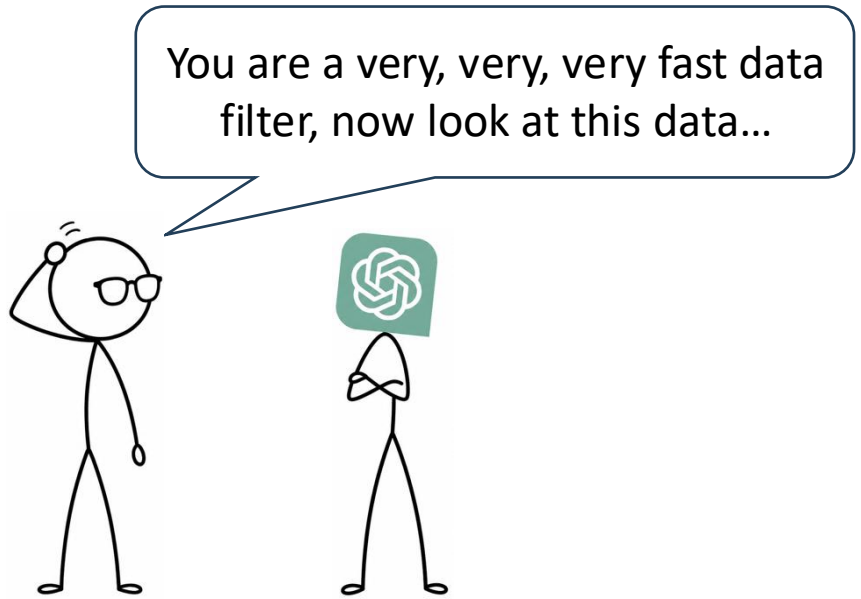
L1 trigger system

400x reduction
3.8 µs to decide after collision
Next-gen: more data, require ~1 µs

100 KHz

High-Level Trigger

Software system
30,000 CPU cores[2]

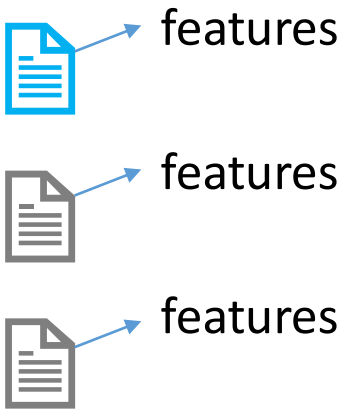


[1] https://en.wikipedia.org/wiki/Large_Hadron_Collider
[2] https://cms.cern/news/real-time-analysis-cms-level-1-trigger

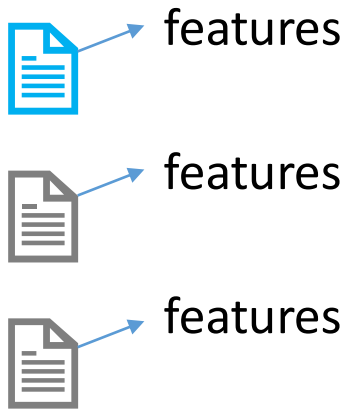
Accelerating the Jet-tagging task

Accelerating the Jet-tagging task

Accelerating the Jet-tagging task

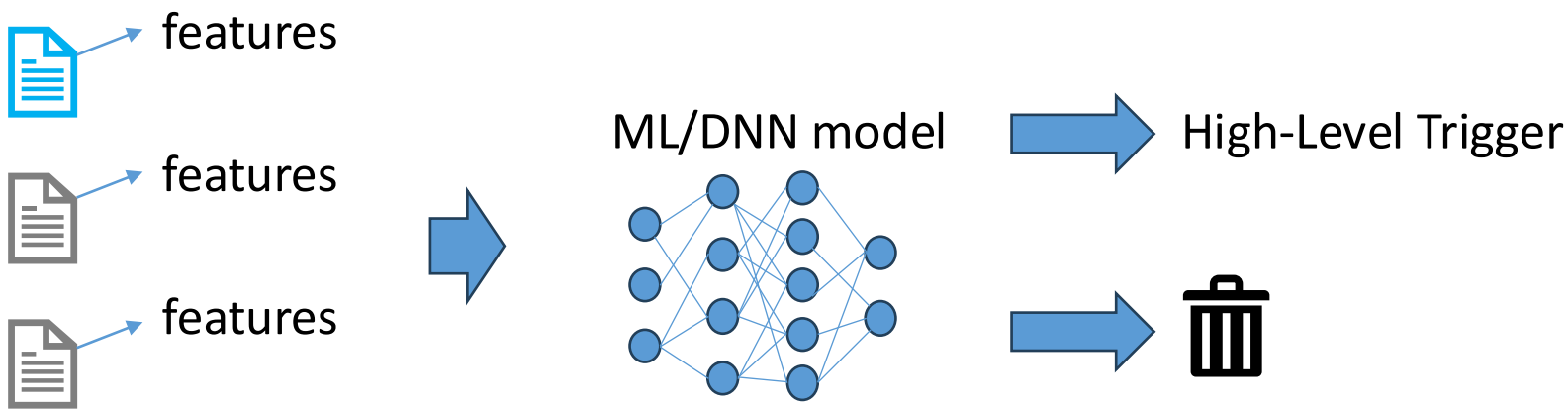


Accelerating the Jet-tagging task



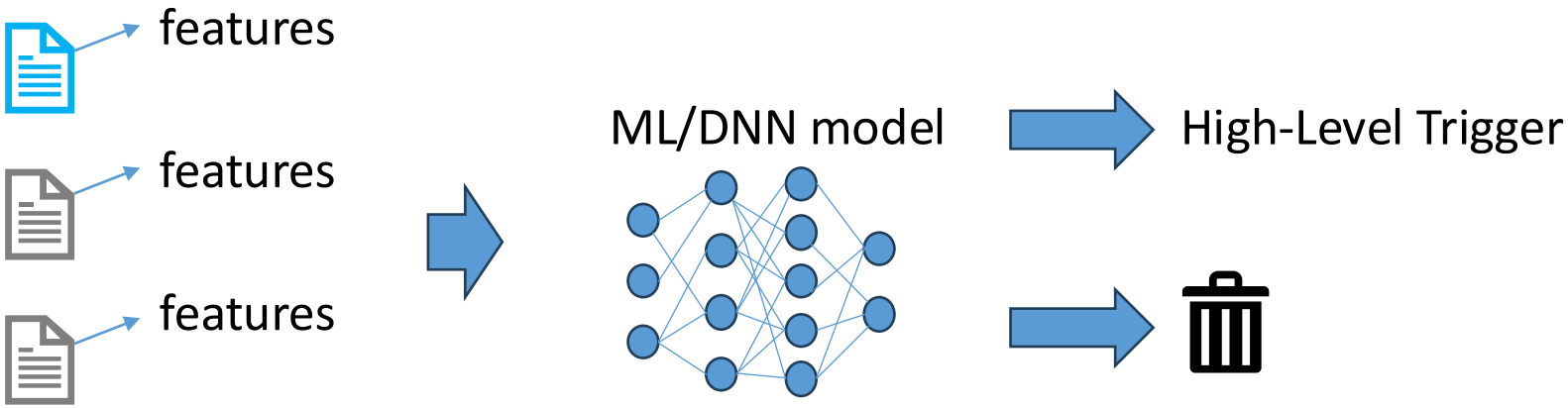
Manually abstract: (1, 16)
Low level feature: (128, 17)

Accelerating the Jet-tagging task



Manually abstract: (1, 16)
Low level feature: (128, 17)

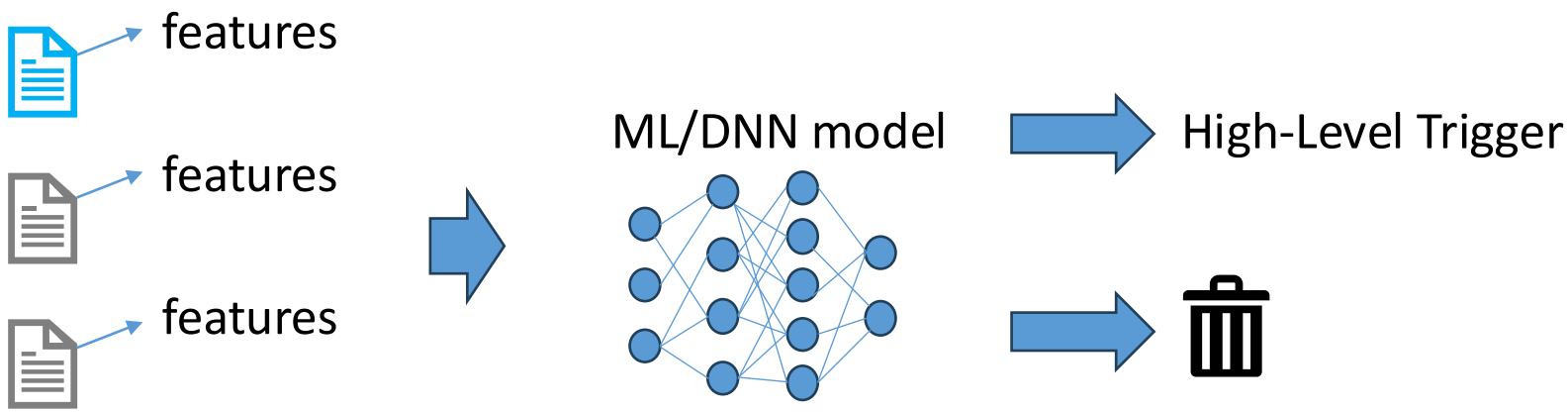
Accelerating the Jet-tagging task



Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

Accelerating the Jet-tagging task

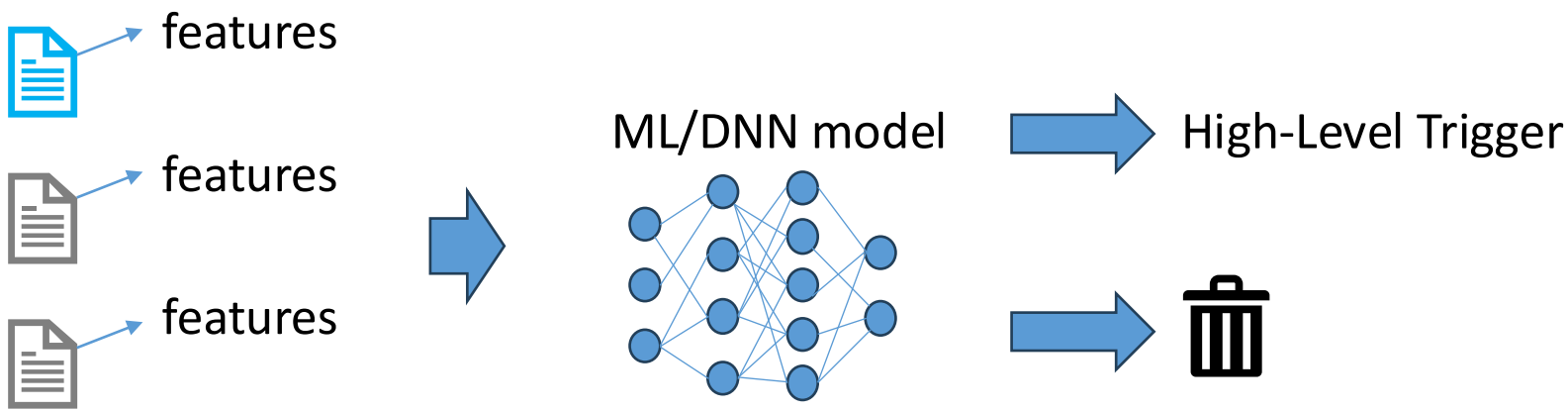


Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

Accelerating the Jet-tagging task



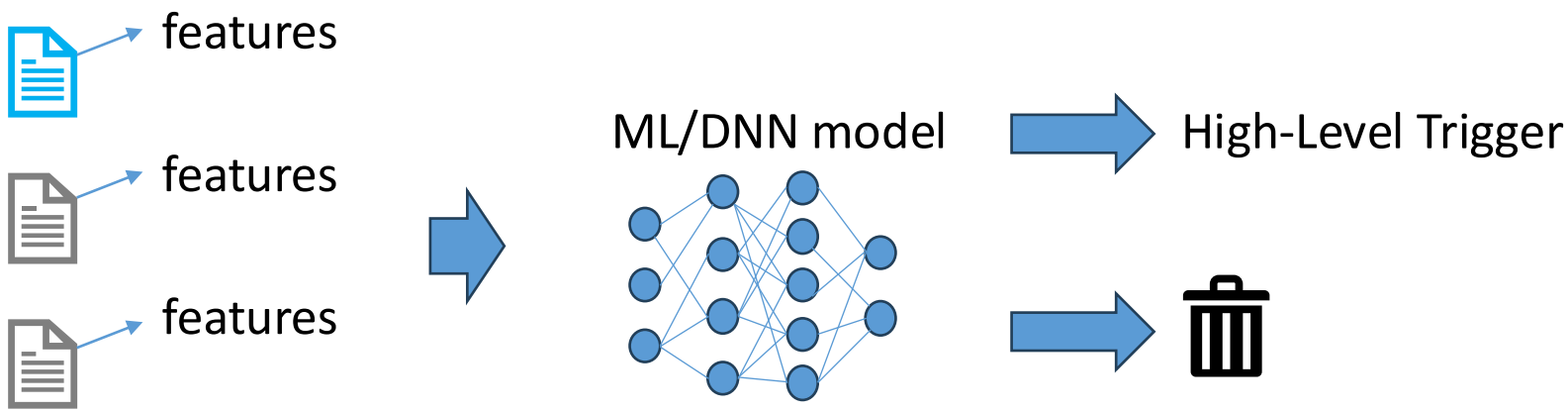
Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

✓ Fully unroll

Accelerating the Jet-tagging task



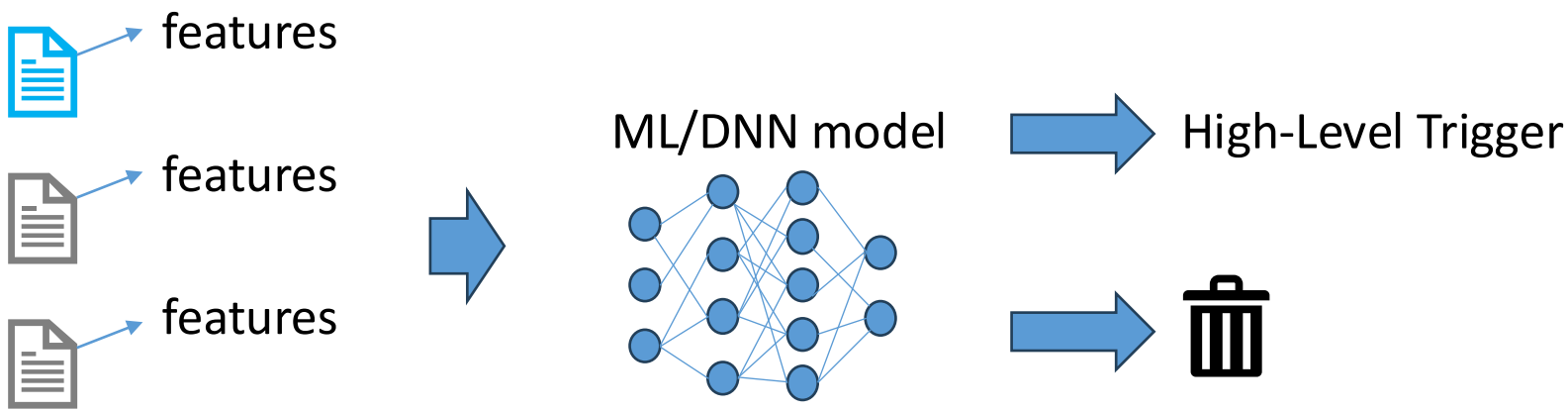
Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

- ✓ Fully unroll
- ✓ 1 cycle per layer

Accelerating the Jet-tagging task



Manually abstract: (1, 16)
Low level feature: (128, 17)

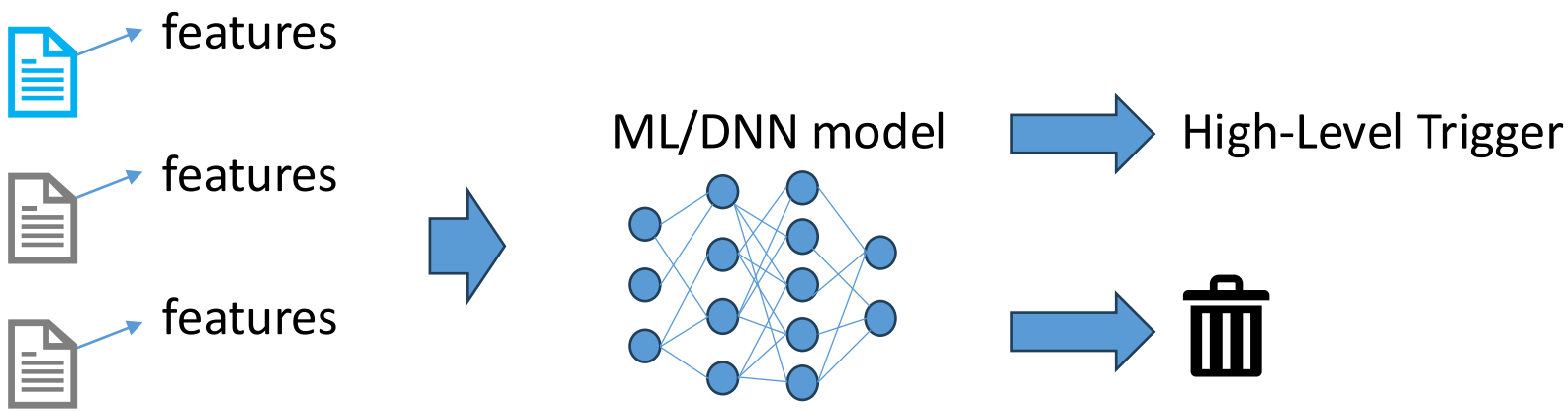
Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

✓ Fully unroll
✓ 1 cycle per layer

⚠ Limited LUT resource

Accelerating the Jet-tagging task



Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

✓ Fully unroll
✓ 1 cycle per layer

⚠ Limited LUT resource

⚠ Quantization, sparsity…
→ trade-off accuracy

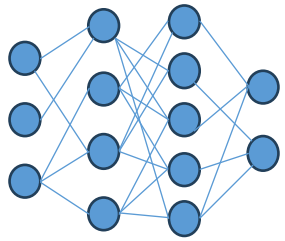
Accelerating the Jet-tagging task

features

features

features

ML/DNN model



High-Level Trigger

Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

✓ Fully unroll
✓ 1 cycle per layer

⚠ Limited LUT resource

⚠ Quantization, sparsity…
→ trade-off accuracy

FPGA+AIE architecture/ AMD Versal

Accelerating the Jet-tagging task

features

features

features

ML/DNN model

High-Level Trigger

Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

✓ Fully unroll
✓ 1 cycle per layer

⚠ Limited LUT resource

⚠ Quantization, sparsity…
→ trade-off accuracy

FPGA+AIE architecture/ AMD Versal

✓ AI Engines are flexible

Accelerating the Jet-tagging task

features

features

features

ML/DNN model

High-Level Trigger

Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

✓ Fully unroll
✓ 1 cycle per layer

⚠ Limited LUT resource

⚠ Quantization, sparsity…
→ trade-off accuracy

FPGA+AIE architecture/ AMD Versal

✓ AI Engines are flexible

✓ 1.25 GHz Freq

Accelerating the Jet-tagging task

features

features

features

ML/DNN model

High-Level Trigger

Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

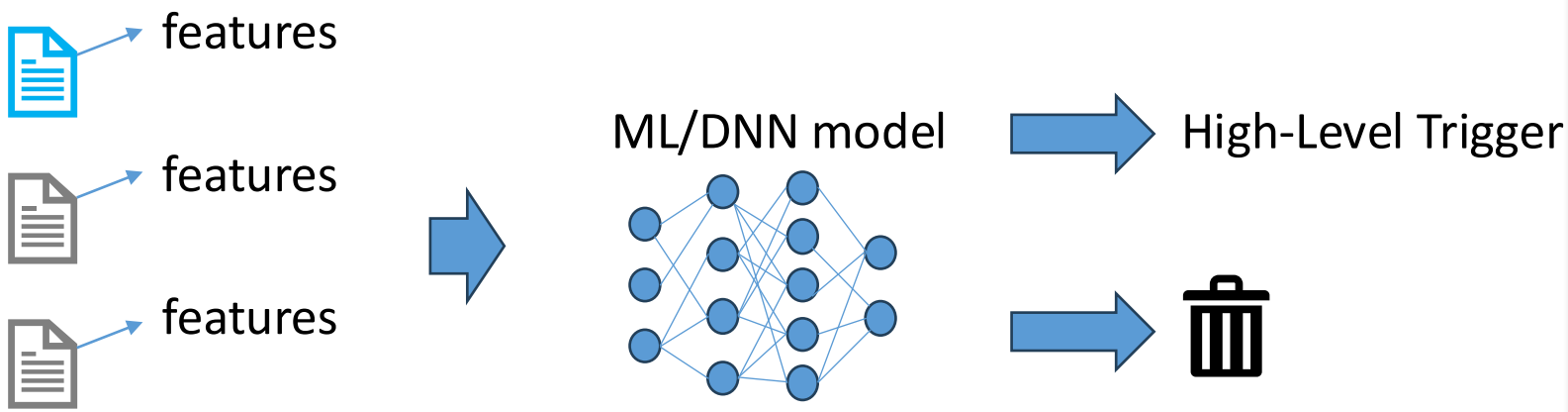
- ✓ Fully unroll
- ✓ 1 cycle per layer

- ⚠ Limited LUT resource
- ⚠ Quantization, sparsity…
  → trade-off accuracy

FPGA+AIE architecture/ AMD Versal

- ✓ AI Engines are flexible
- ✓ 1.25 GHz Freq
- ✓ 256 INT8 MAC per cycle
  × 304 AIE tiles

Accelerating the Jet-tagging task



Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

- ✓ Fully unroll
- ✓ 1 cycle per layer

- ⚠ Limited LUT resource
- ⚠ Quantization, sparsity…
  → trade-off accuracy

FPGA+AIE architecture/ AMD Versal

- ✓ AI Engines are flexible
- ✓ 1.25 GHz Freq
- ✓ 256 INT8 MAC per cycle
  × 304 AIE tiles
- ✓ Excel in INT8, BF16, INT16,…

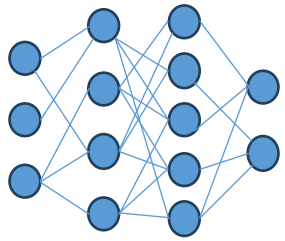
Accelerating the Jet-tagging task

features

features

features

ML/DNN model



High-Level Trigger

Manually abstract: (1, 16)
Low level feature: (128, 17)

Small size, simple architecture
e.g. JSC-XL: 320 neuron, 20K param(dense)

FPGA accelerators

- ✓ Fully unroll
- ✓ 1 cycle per layer

- ⚠ Limited LUT resource
- ⚠ Quantization, sparsity…
  → trade-off accuracy

FPGA+AIE architecture/ AMD Versal

- ✓ AI Engines are flexible
- ✓ 1.25 GHz Freq
- ✓ 256 INT8 MAC per cycle
  × 304 AIE tiles
- ✓ Excel in INT8, BF16, INT16,…

? Can we use Versal for 1-us scale NN inference?

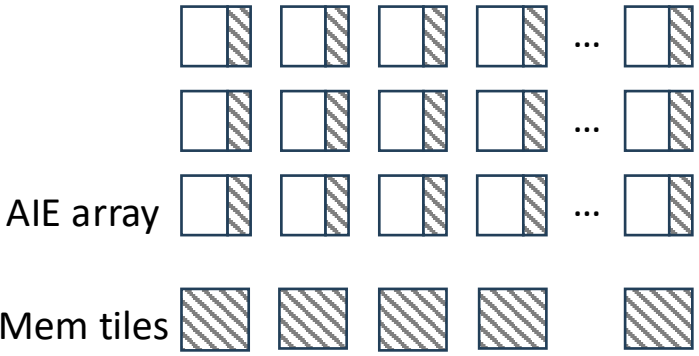
Background: Communication patterns in Versal

Background: Communication patterns in Versal

- On-chip communication patterns of AIE

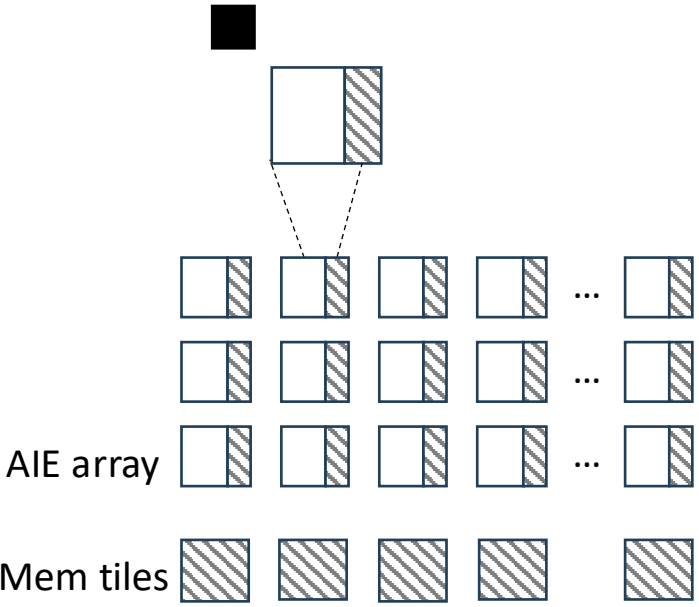
Background: Communication patterns in Versal

- On-chip communication patterns of AIE



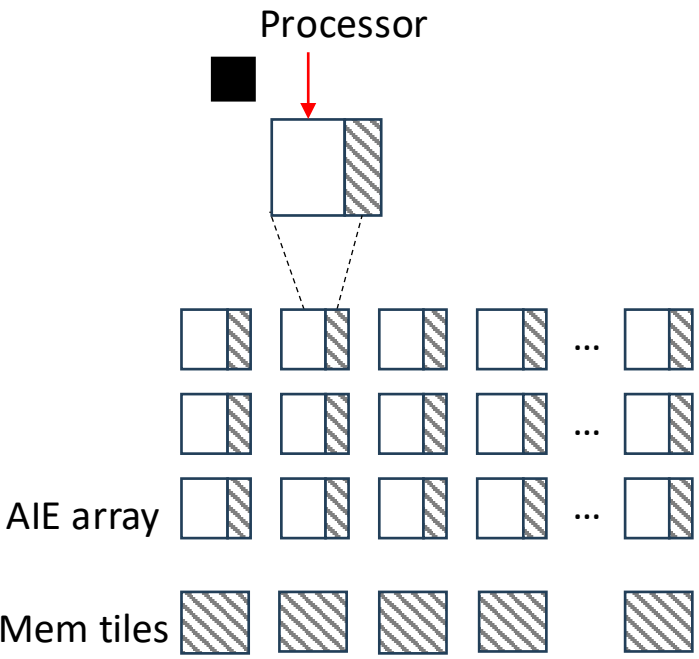
Background: Communication patterns in Versal

- On-chip communication patterns of AIE



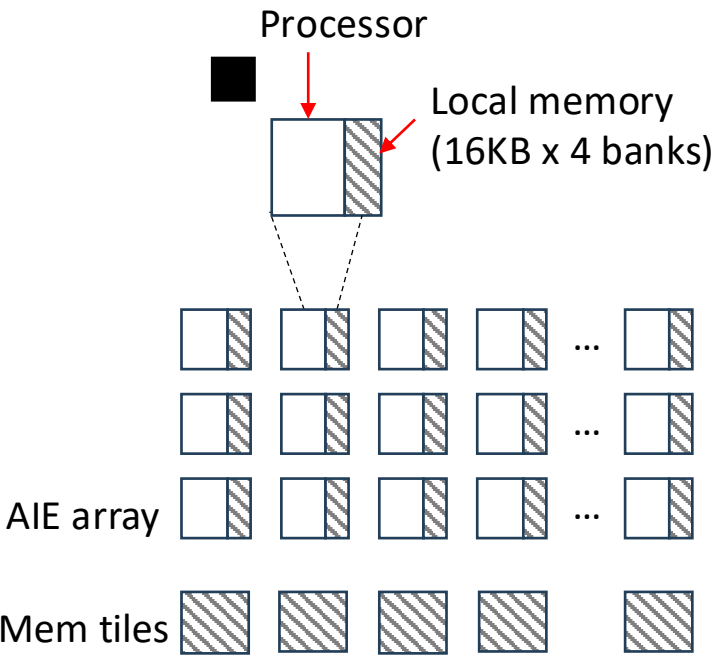
Background: Communication patterns in Versal

- On-chip communication patterns of AIE



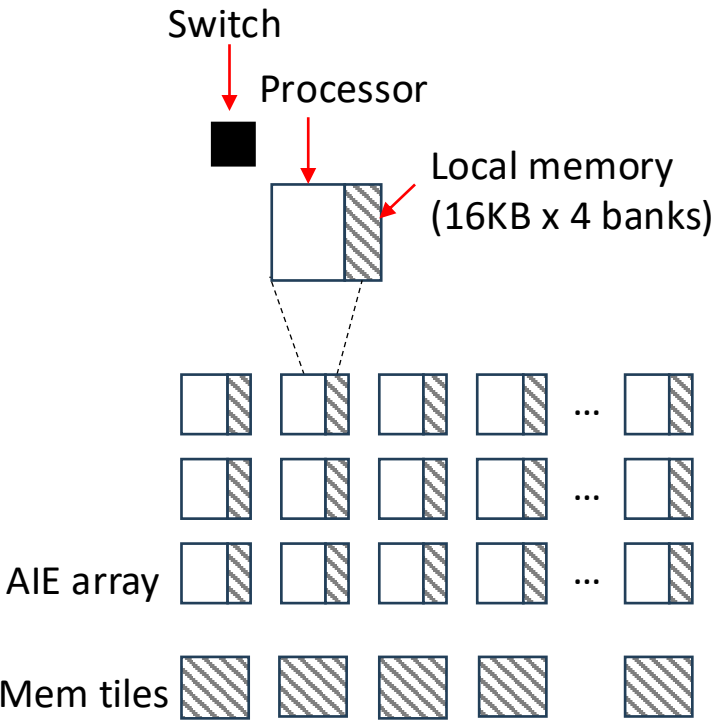
Background: Communication patterns in Versal

- On-chip communication patterns of AIE



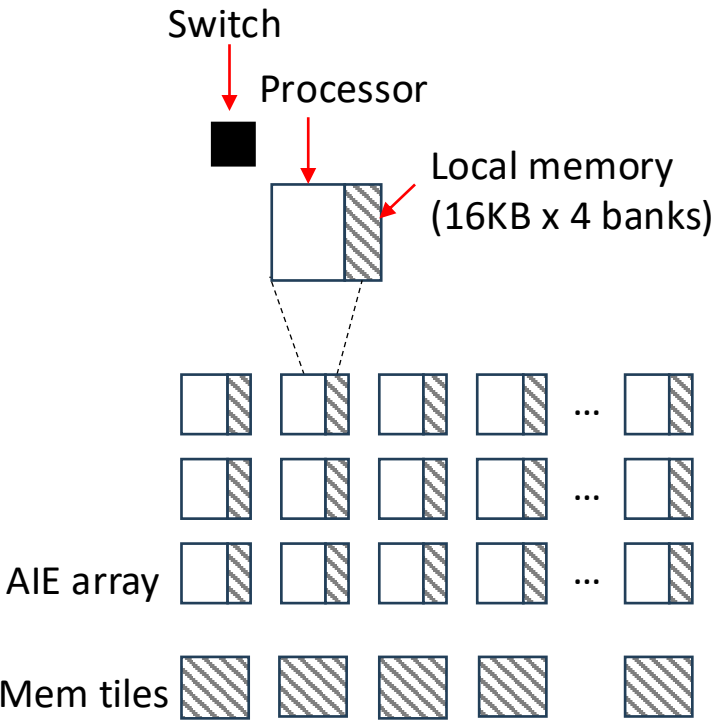
Background: Communication patterns in Versal

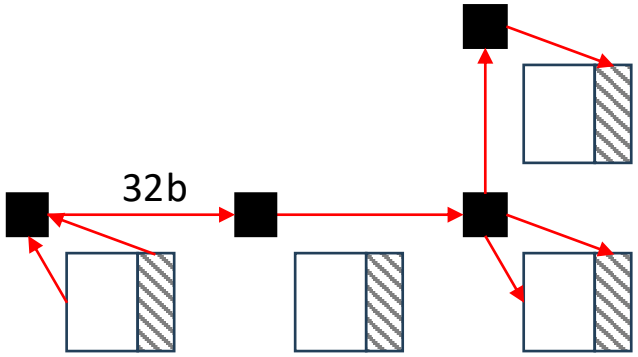
- On-chip communication patterns of AIE



Background: Communication patterns in Versal

- On-chip communication patterns of AIE



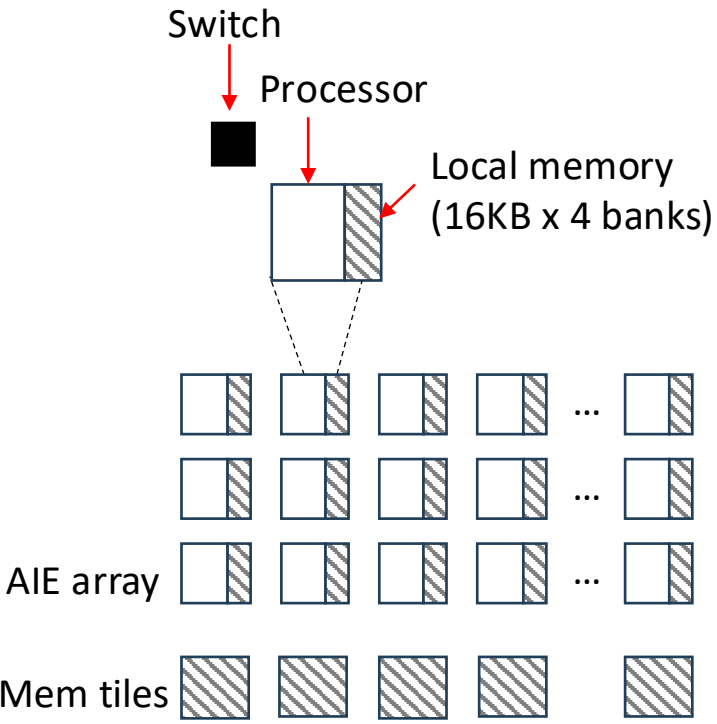


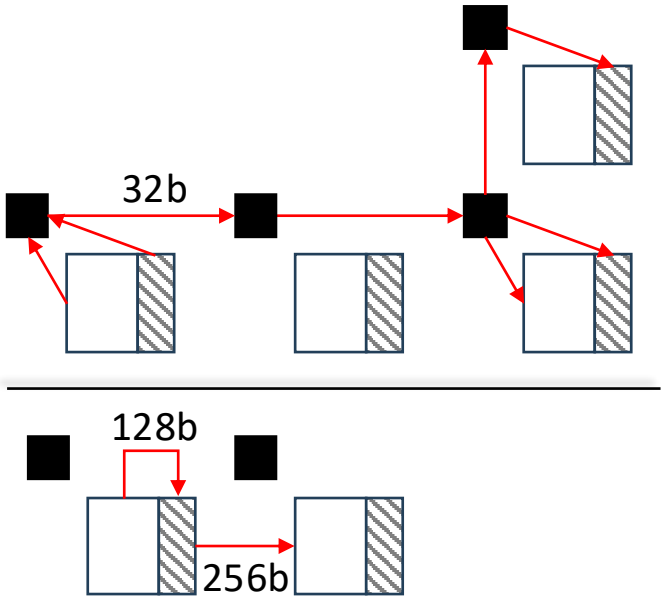
Switch-based connection: stream or DMA

- can send from any tile to any tile
- can broadcast
- Only 32 bits/cycle bandwidth, large overhead

Background: Communication patterns in Versal

- On-chip communication patterns of AIE





Switch-based connection: stream or DMA

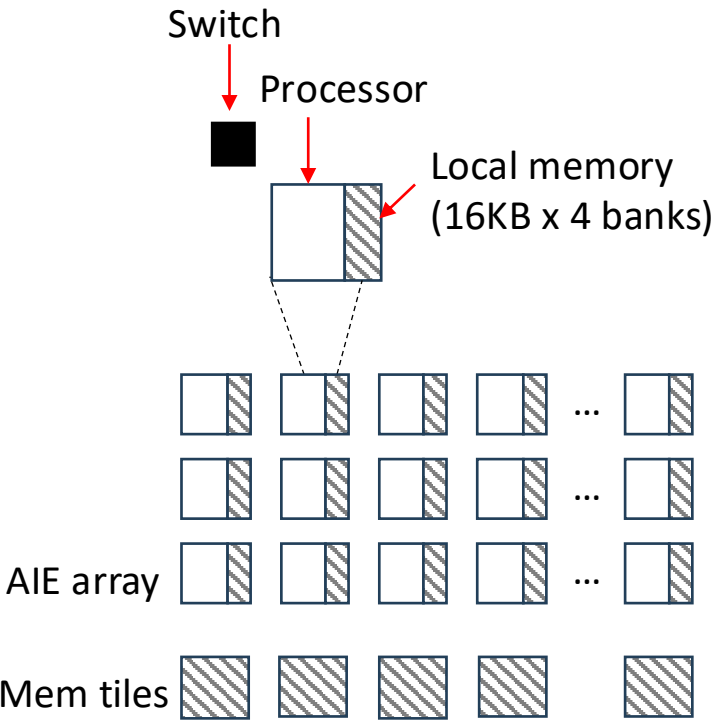
- can send from any tile to any tile
- can broadcast
- Only 32 bits/cycle bandwidth, large overhead

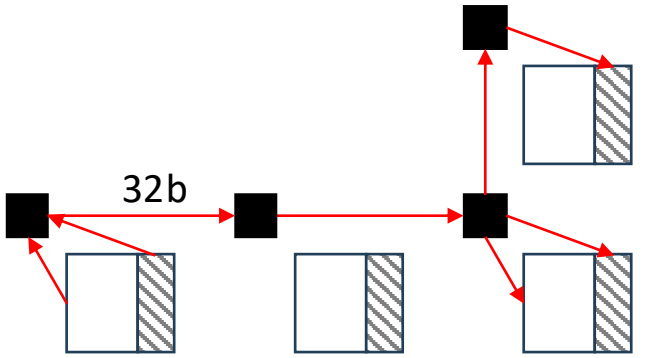
Shared memory connection

- can only send to adjacent tile in 4 directions
- at most 256 bits/cycle, smaller overhead

Background: Communication patterns in Versal

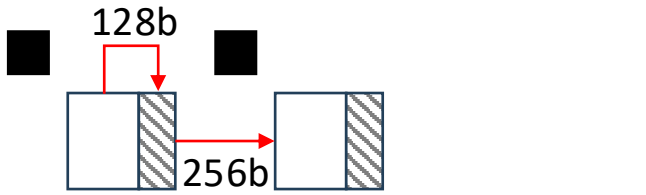
- On-chip communication patterns of AIE





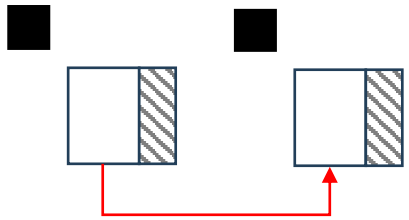
Switch-based connection: stream or DMA

- can send from any tile to any tile
- can broadcast
- Only 32 bits/cycle bandwidth, large overhead



Shared memory connection

- can only send to adjacent tile in 4 directions
- at most 256 bits/cycle, smaller overhead

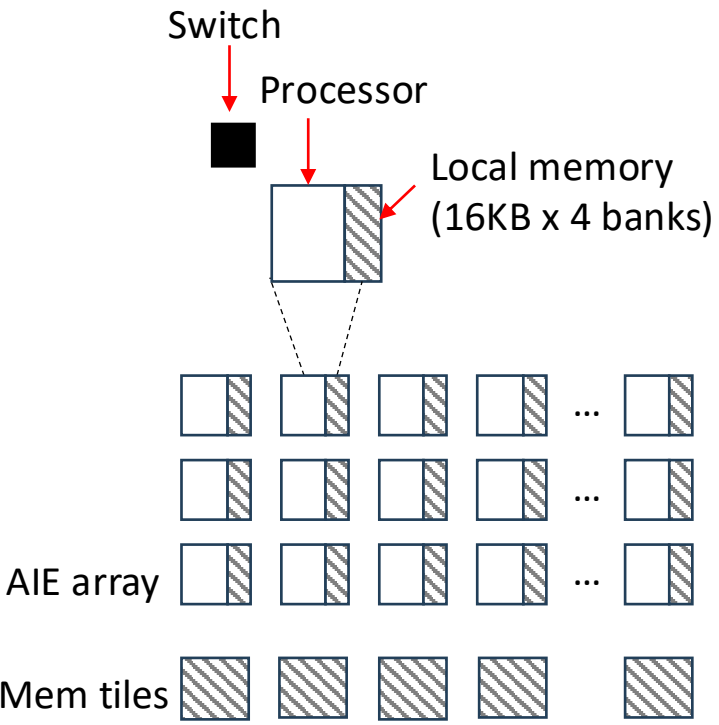


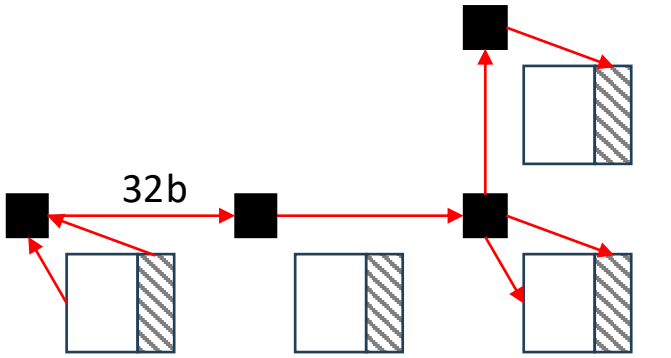
Cascade connection

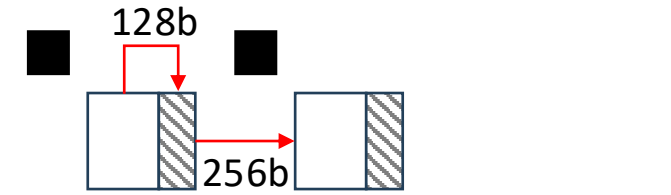
- Can only send to adjacent tile in W->E or N->S directions
- 512 bits/cycle, smallest overhead

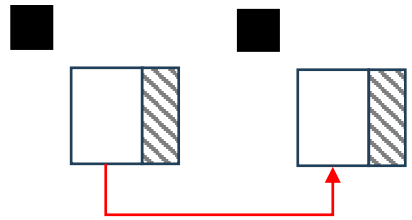
Background: Communication patterns in Versal

- On-chip communication patterns of AIE









Switch-based connection: stream or DMA

- can send from any tile to any tile
- can broadcast
- Only 32 bits/cycle bandwidth, large overhead

Shared memory connection

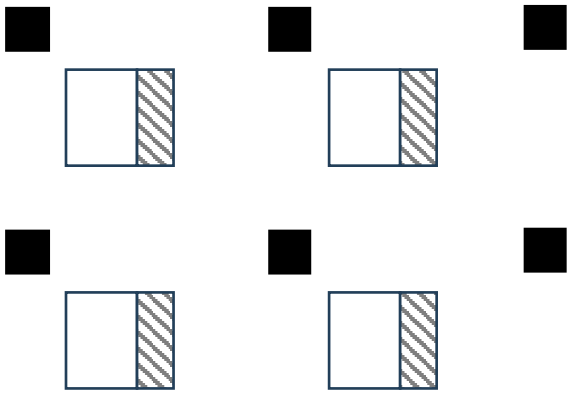
- can only send to adjacent tile in 4 directions
- at most 256 bits/cycle, smaller overhead

Cascade connection

- Can only send to adjacent tile in W->E or N->S directions
- 512 bits/cycle, smallest overhead

! We should leverage the high-bandwidth connection

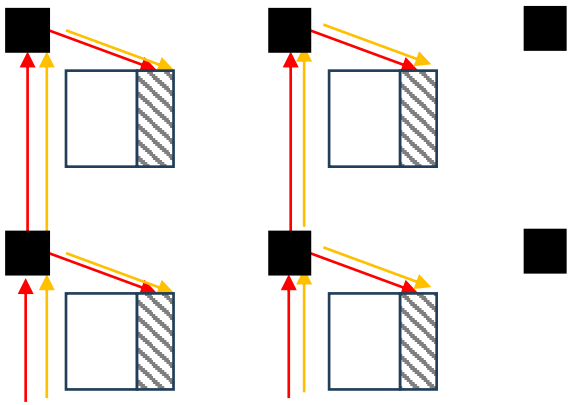
Baseline communication patterns



DMA
Cascade
Shared-mem

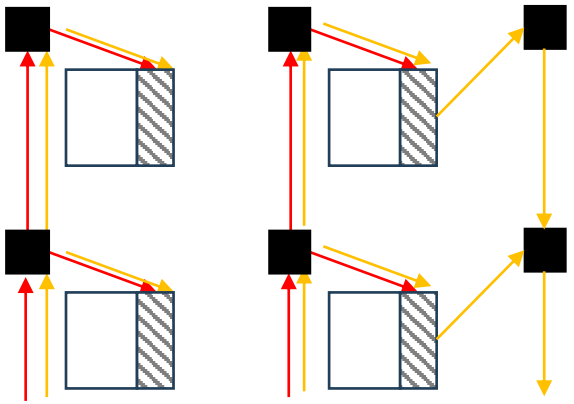
activation
weights
output
Partial results

Baseline communication patterns



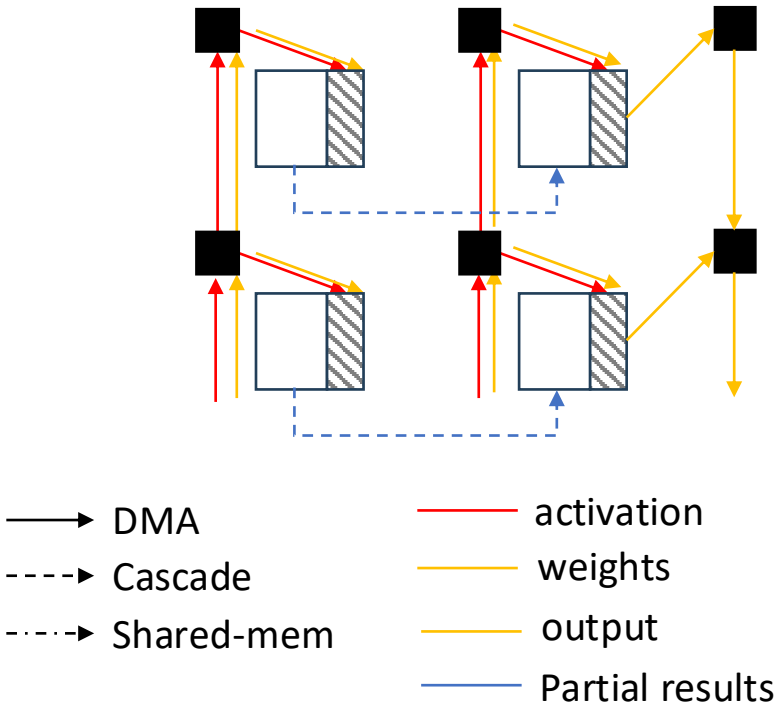
- Load Activations from DMA
- Load weights from DMA

Baseline communication patterns



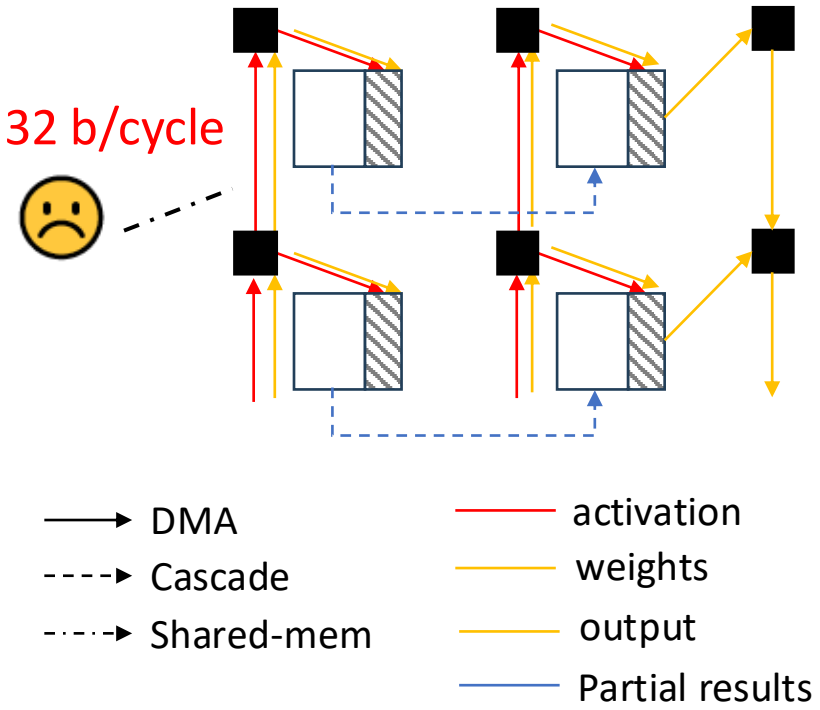
- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer

Baseline communication patterns



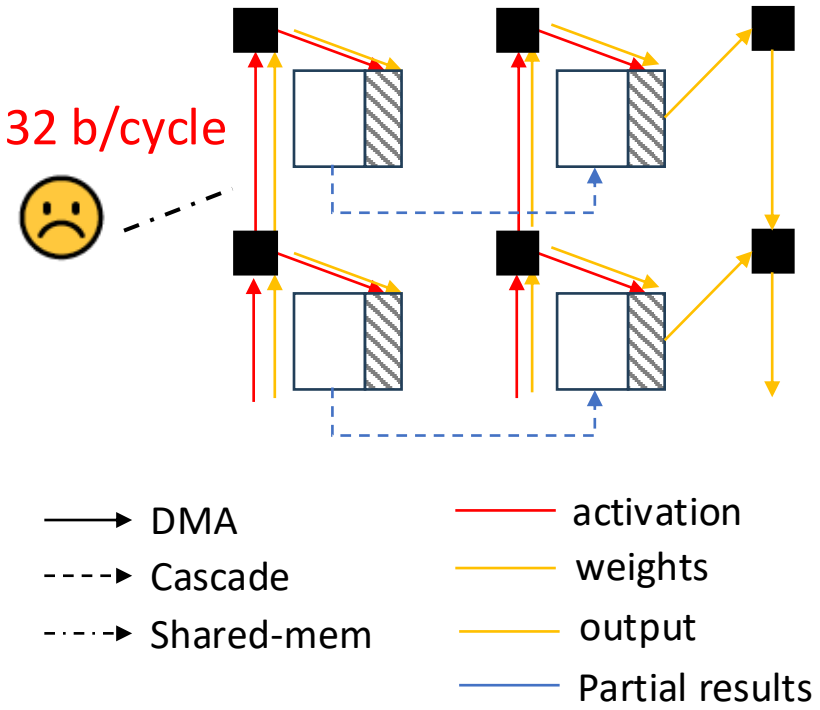
- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer
- To enable the spatial partitioning K-dim, cascade for the partial results

Baseline communication patterns



- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer
- To enable the spatial partitioning K-dim, cascade for the partial results

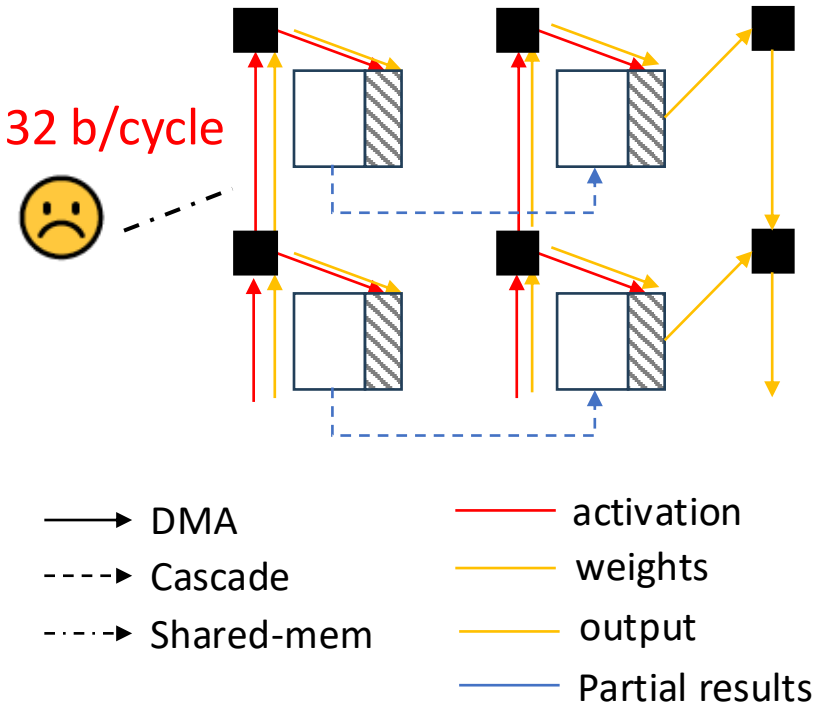
Baseline communication patterns



- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer
- To enable the spatial partitioning K-dim, cascade for the partial results

⚠ Quick Math: 8x32x64 INT8 kernel (256 MAC/cycle, 1GHz):

Baseline communication patterns



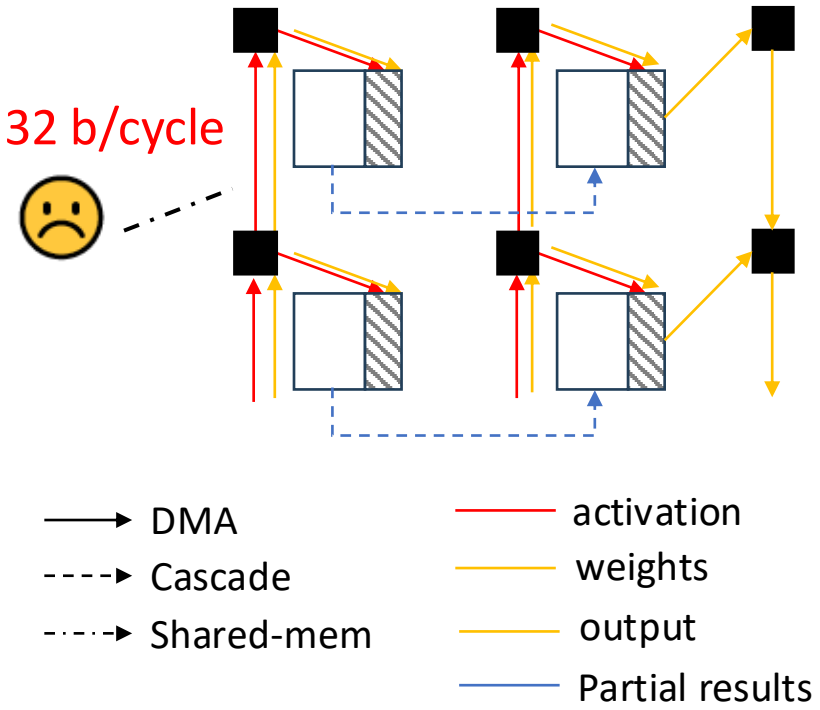
- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer
- To enable the spatial partitioning K-dim, cascade for the partial results

⚠ Quick Math: 8x32x64 INT8 kernel (256 MAC/cycle, 1GHz):

- 64 cycles to load the activation
- 512 cycles to load the weights
- 64 cycles to compute
- 128 cycles to store the results

→512+64+128=704 cycles(704 ns) for one single layer,

Baseline communication patterns



- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer
- To enable the spatial partitioning K-dim, cascade for the partial results

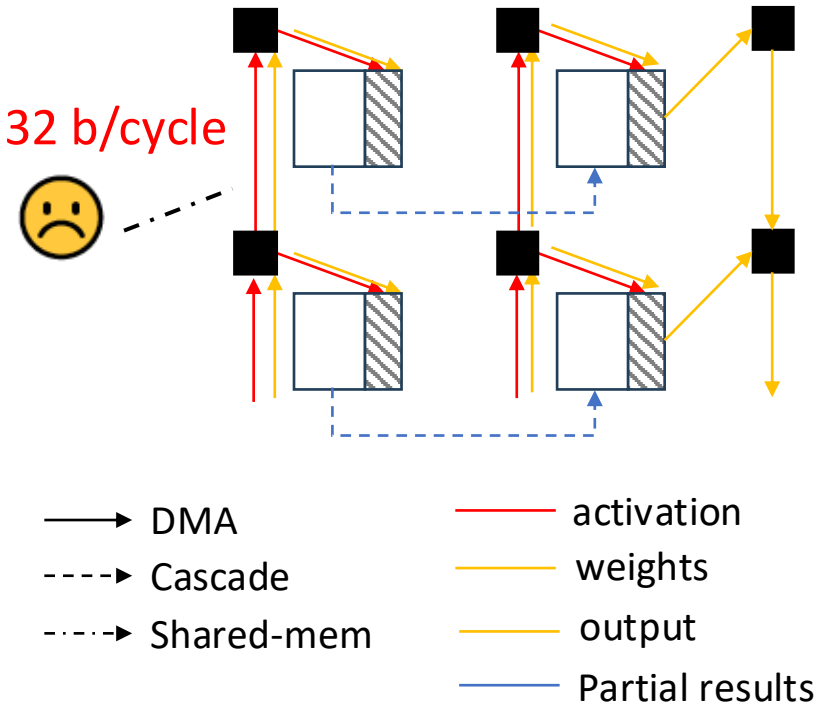
⚠ Quick Math: 8x32x64 INT8 kernel (256 MAC/cycle, 1GHz):

- 64 cycles to load the activation
- 512 cycles to load the weights
- 64 cycles to compute
- 128 cycles to store the results

→512+64+128=704 cycles(704 ns) for one single layer,

⚠ Impossible to finish whole model in 1 us!

Baseline communication patterns



- Load Activations from DMA
- Load weights from DMA
- Store outputs using DMA
- The output needs to be re-loaded for next layer
- To enable the spatial partitioning K-dim, cascade for the partial results

⚠ Quick Math: 8x32x64 INT8 kernel (256 MAC/cycle, 1GHz):

- 64 cycles to load the activation
- 512 cycles to load the weights
- 64 cycles to compute
- 128 cycles to store the results

→512+64+128=704 cycles(704 ns) for one single layer,

⚠ Impossible to finish whole model in 1 us!

Optimizing on-chip communication is the key!

Other Gaps: Non-linear layers such as Reduction

Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN

Other Gaps: Non-linear layers such as Reduction

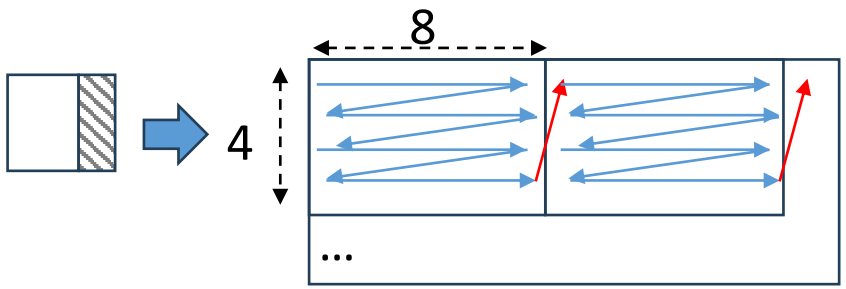
The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN

Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN

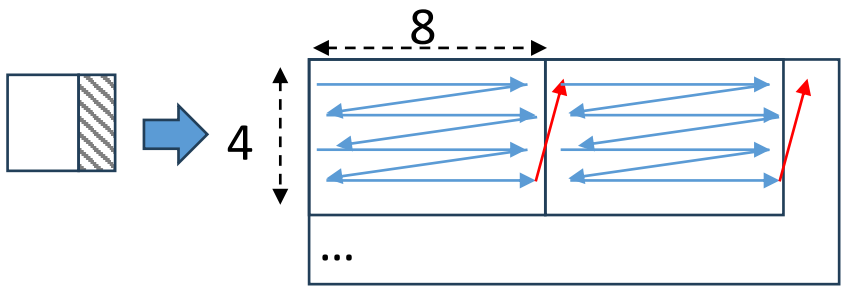


(a) Blocked layout of intermediate data

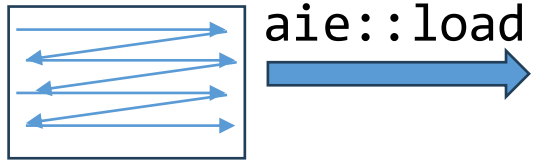
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data

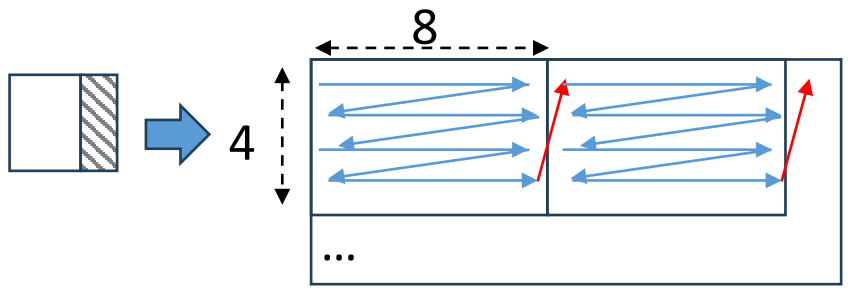


(b) The vector register layout and aggregation computation pattern

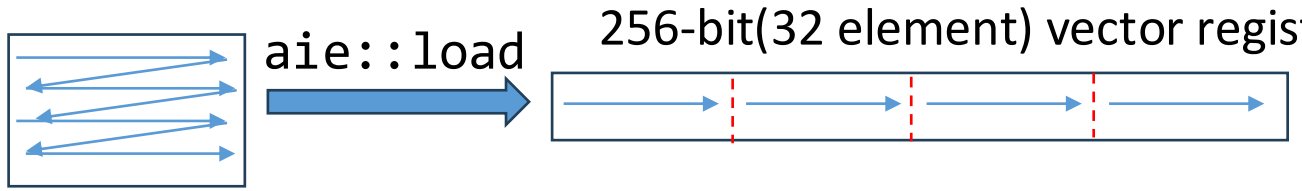
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data

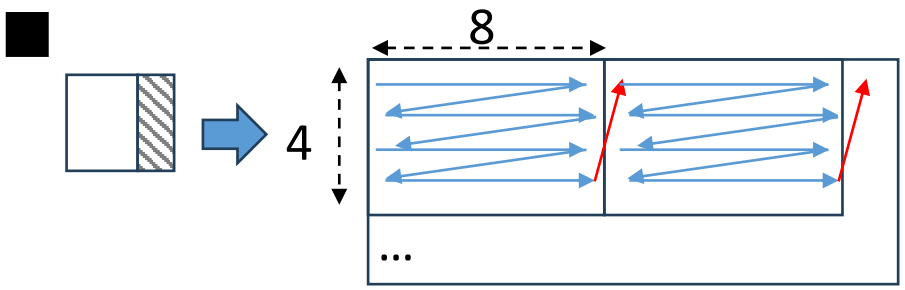


(b) The vector register layout and aggregation computation pattern

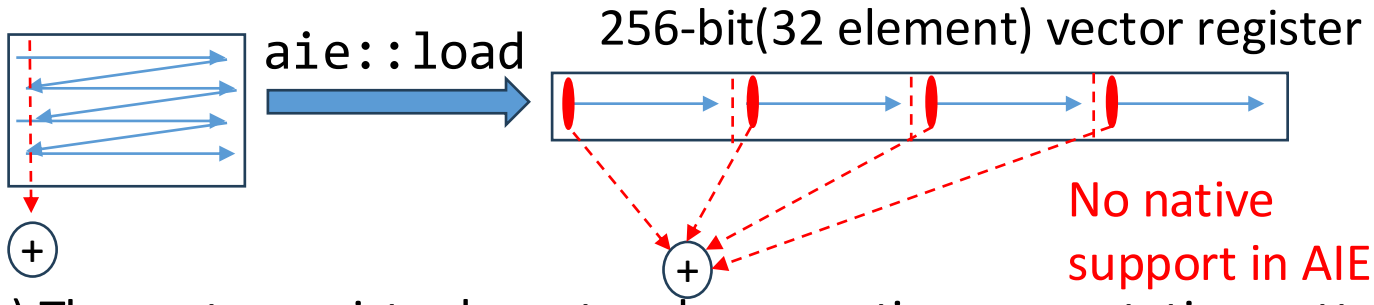
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data

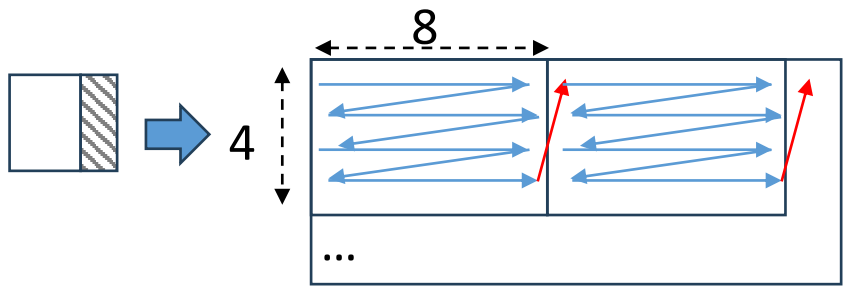


(b) The vector register layout and aggregation computation pattern

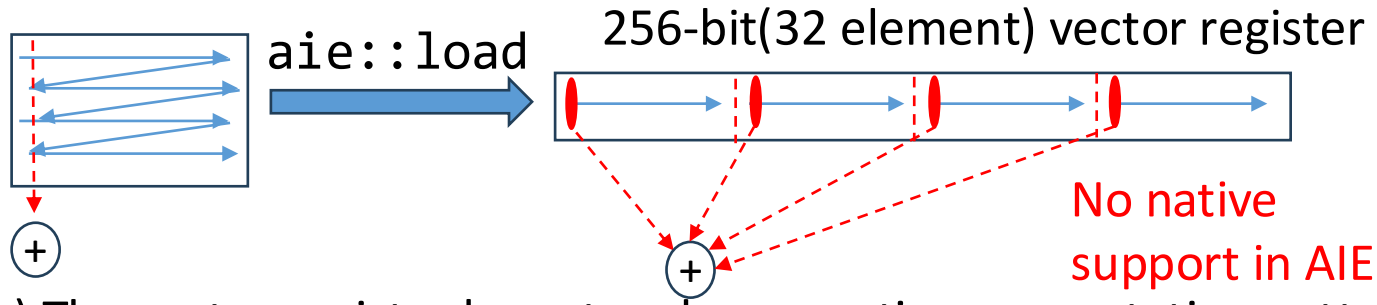
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data

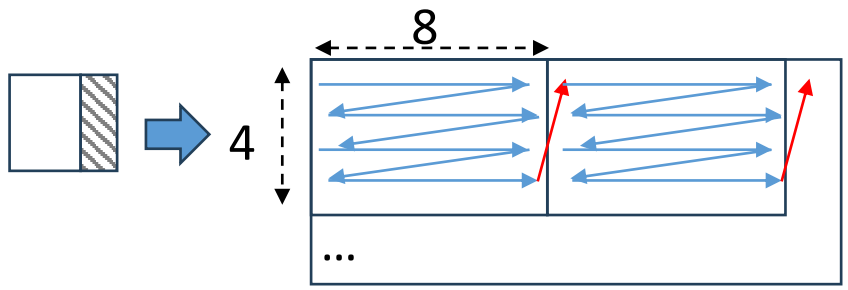


(b) The vector register layout and aggregation computation pattern

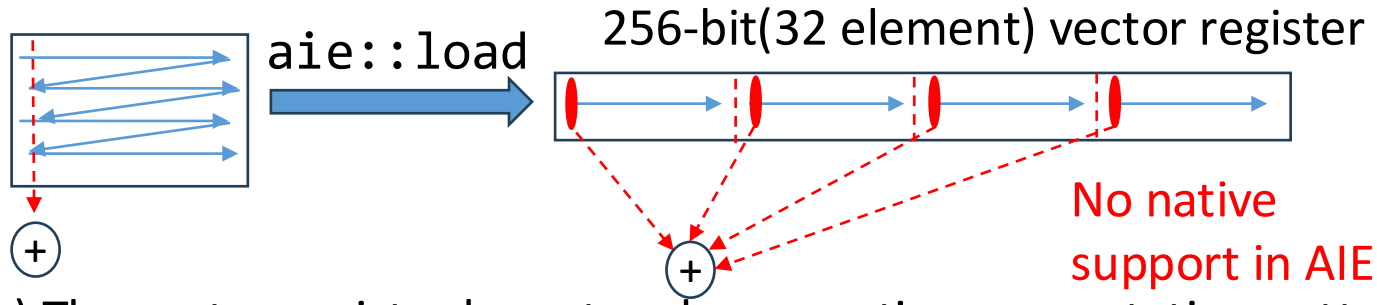
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

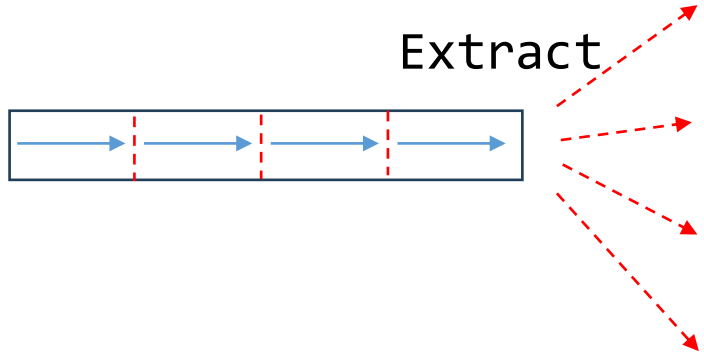
E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data



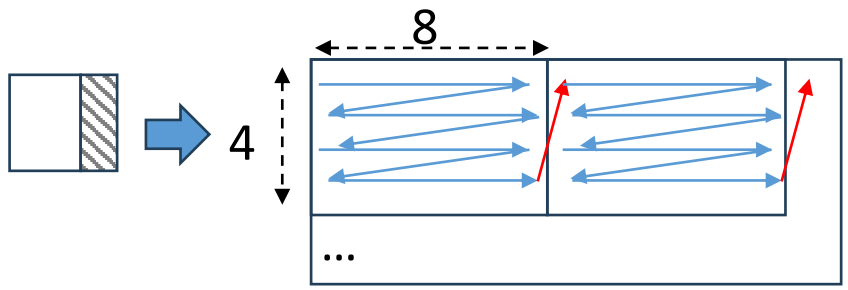
(b) The vector register layout and aggregation computation pattern



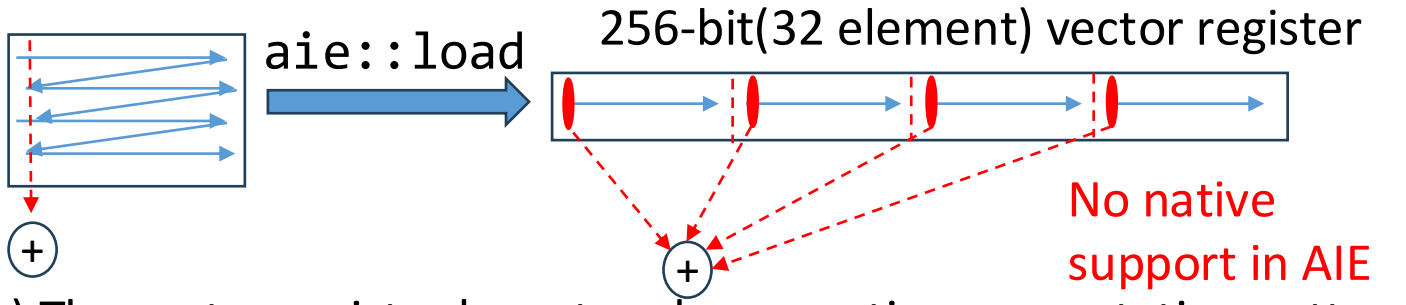
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

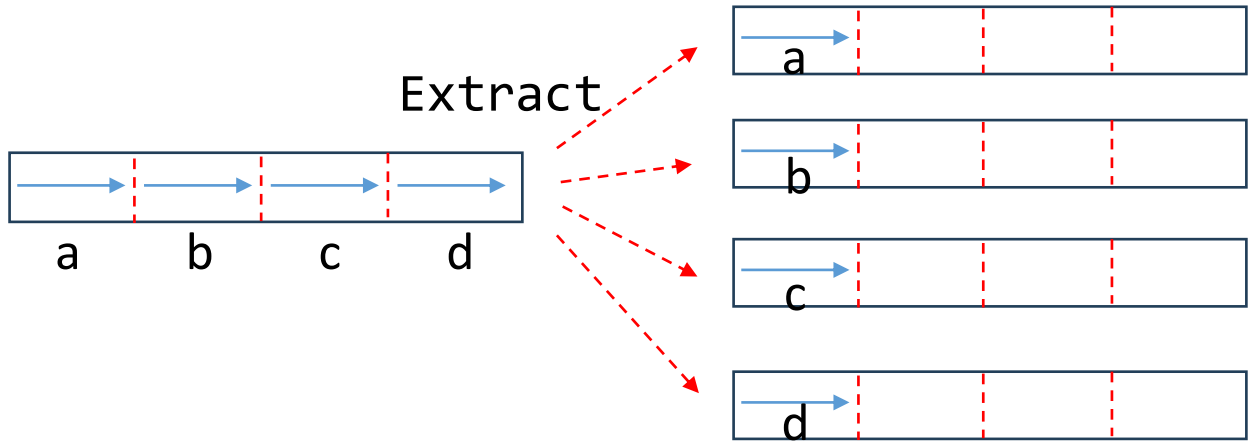
E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data



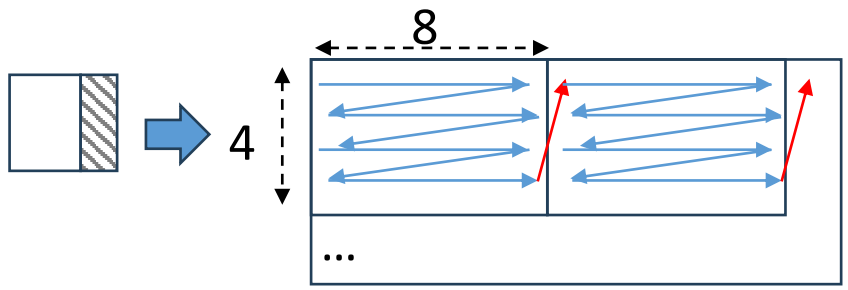
(b) The vector register layout and aggregation computation pattern



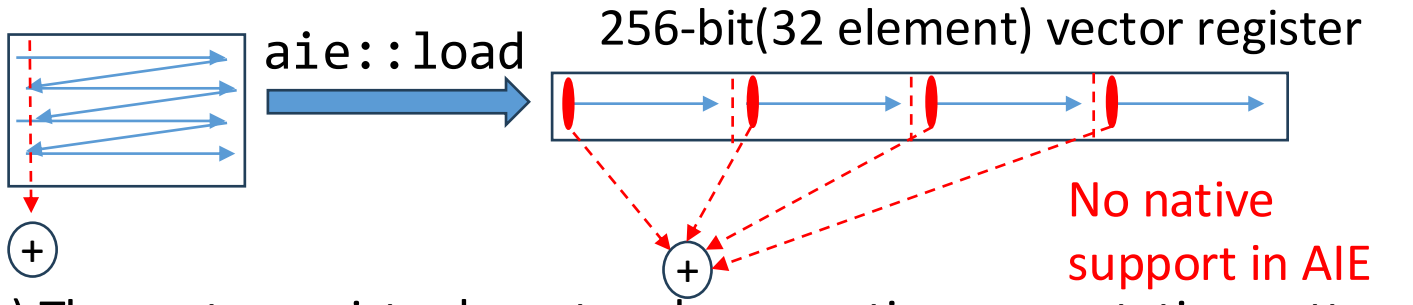
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

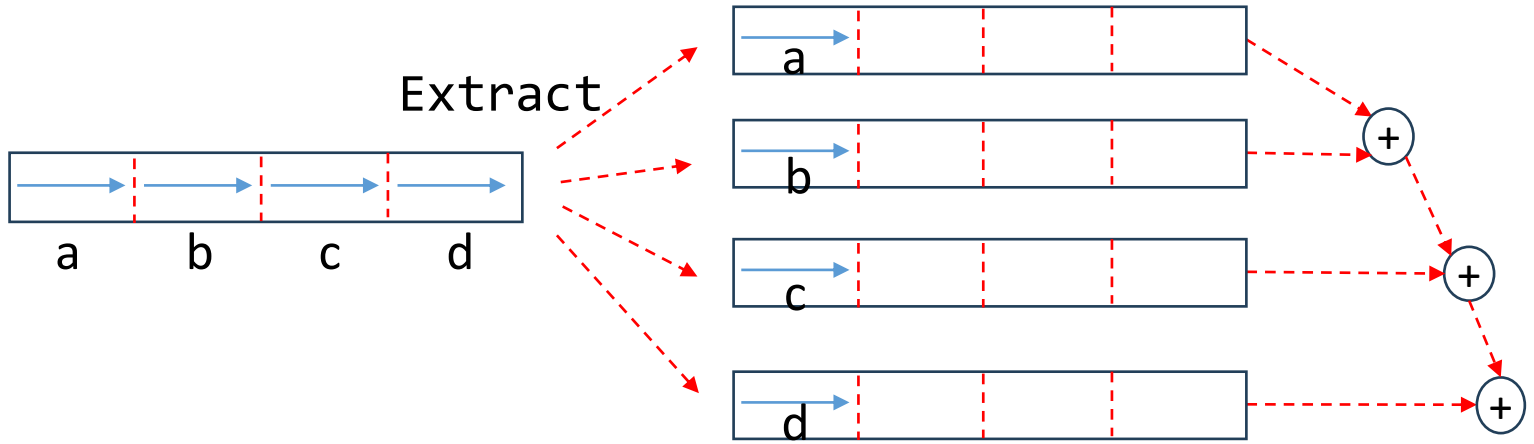
E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data



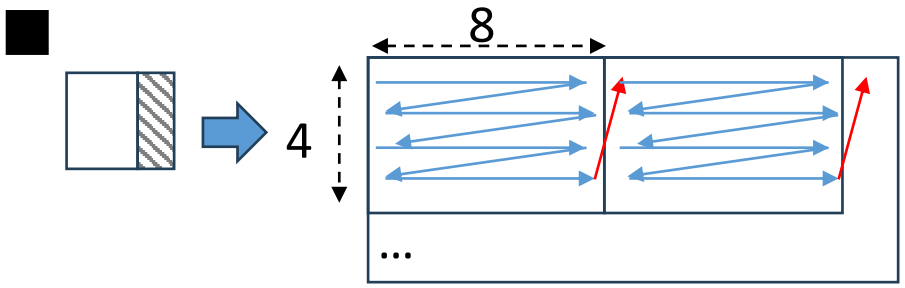
(b) The vector register layout and aggregation computation pattern



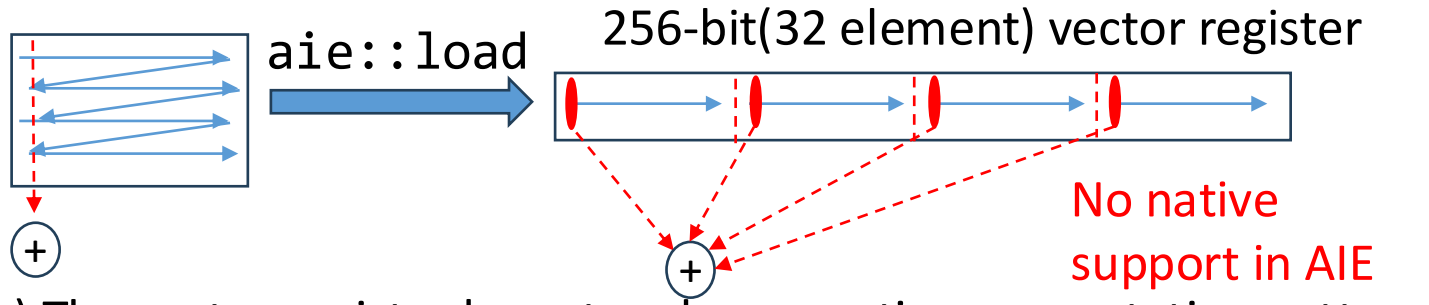
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

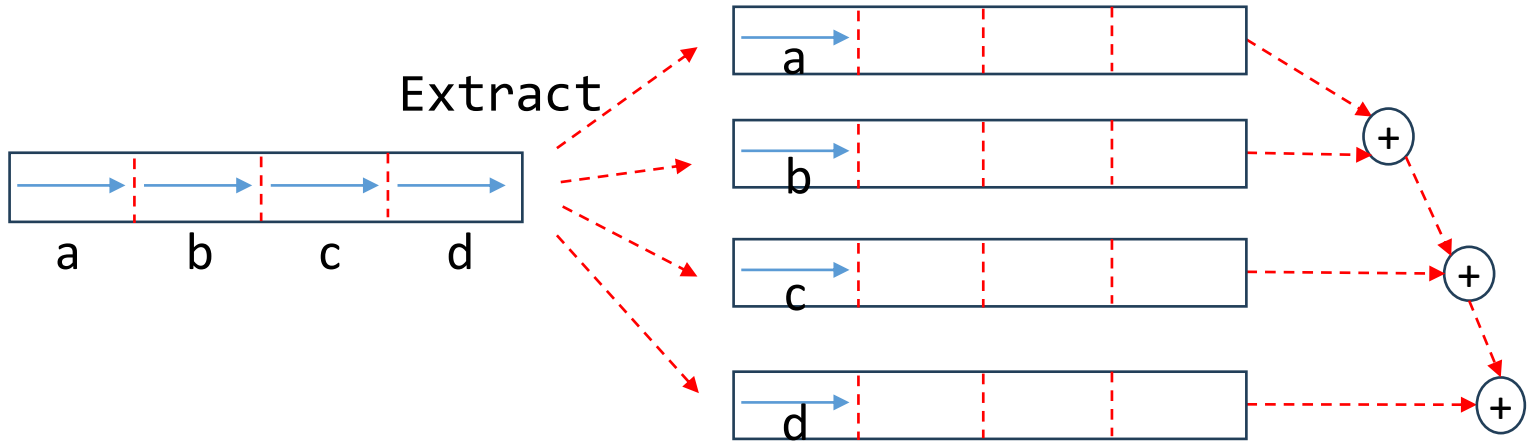
E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data



(b) The vector register layout and aggregation computation pattern

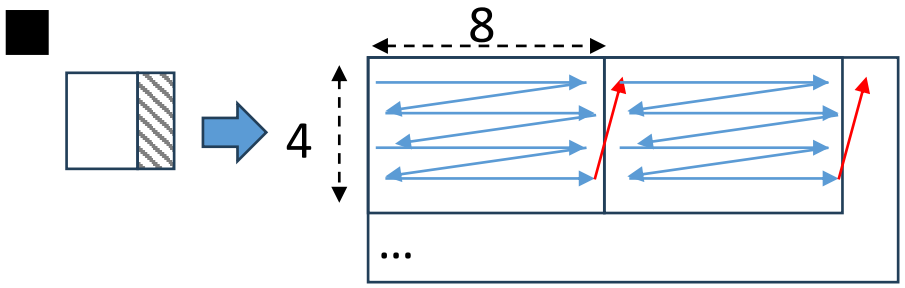


⚠ Baseline reduction implementation, **7 sequential Instrs**(4 extraction, 3 addition) operations are needed

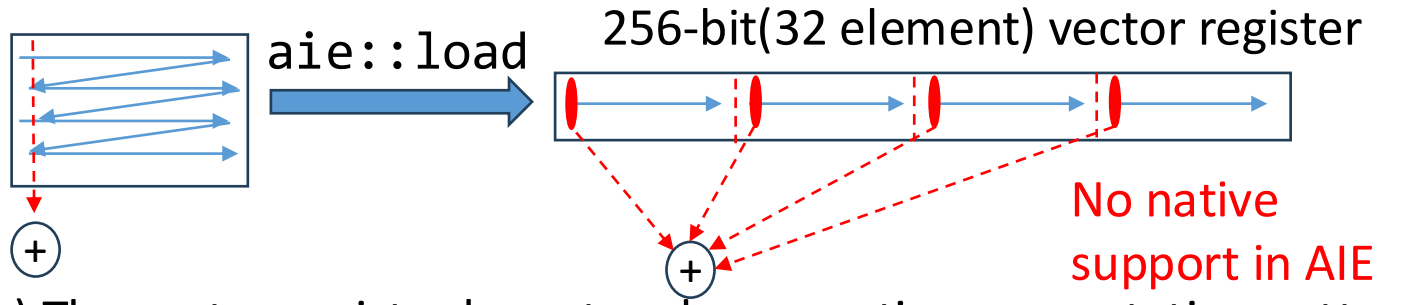
Other Gaps: Non-linear layers such as Reduction

The Cost for computation efficiency is **data layout requirement**

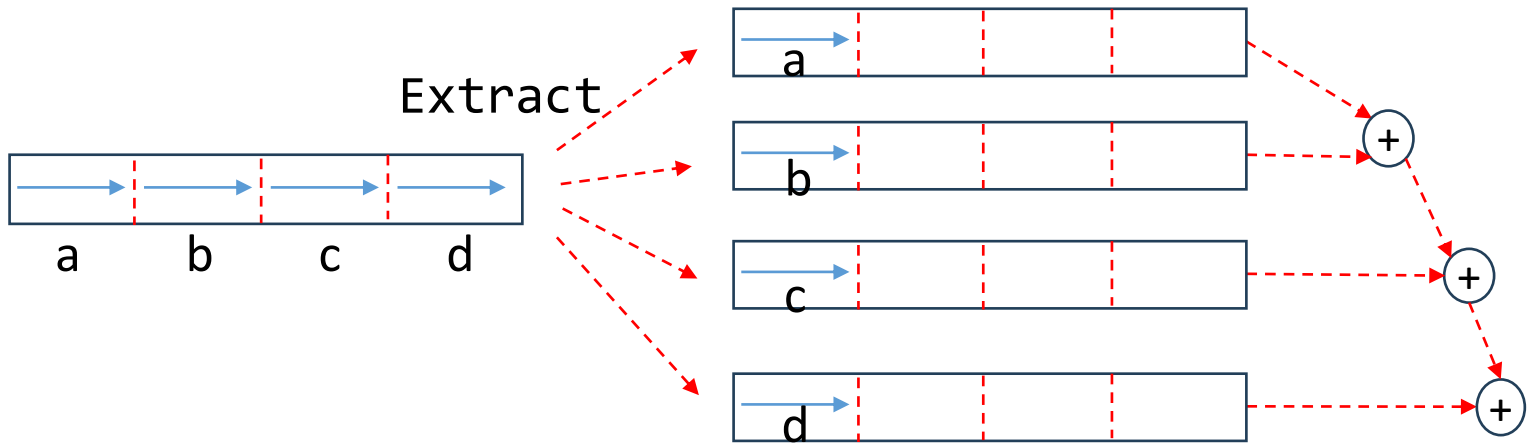
E.g. Suppose we do (1) Matrix Multiplication then (2) reduce the MxN mat to 1xN



(a) Blocked layout of intermediate data



(b) The vector register layout and aggregation computation pattern



! We achieve this operation in one instruction

⚠ Baseline reduction implementation, **7 sequential Instrs**(4 extraction, 3 addition) operations are needed

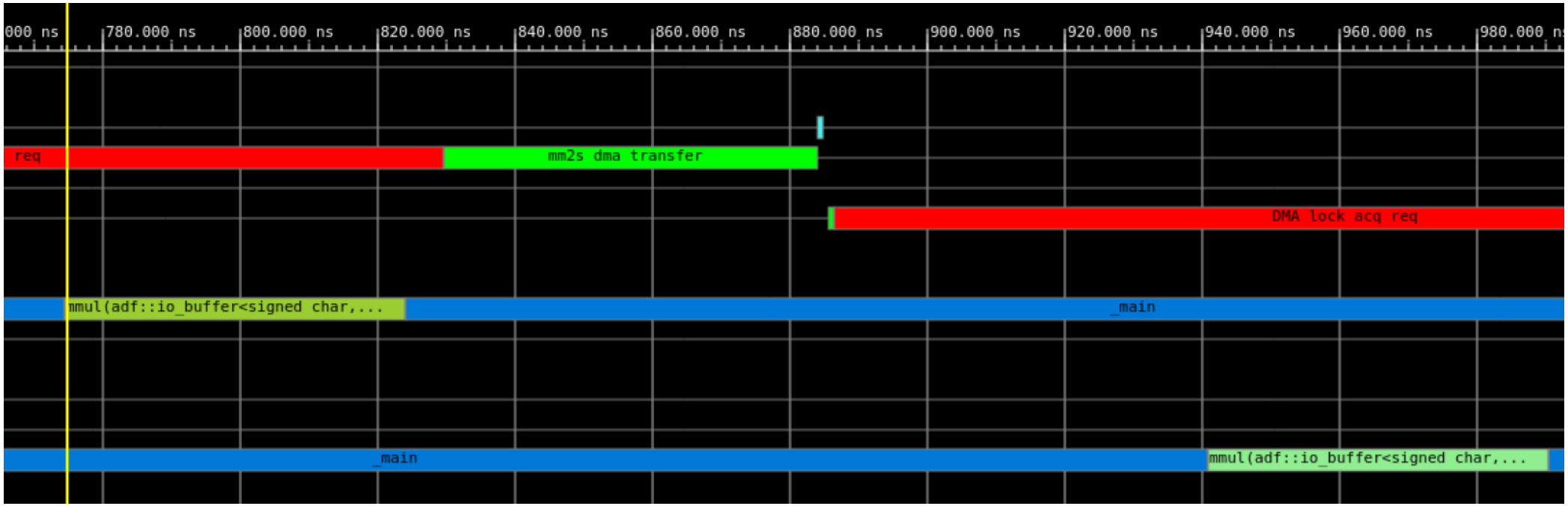
Other Gaps: to Estimate the Performance

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

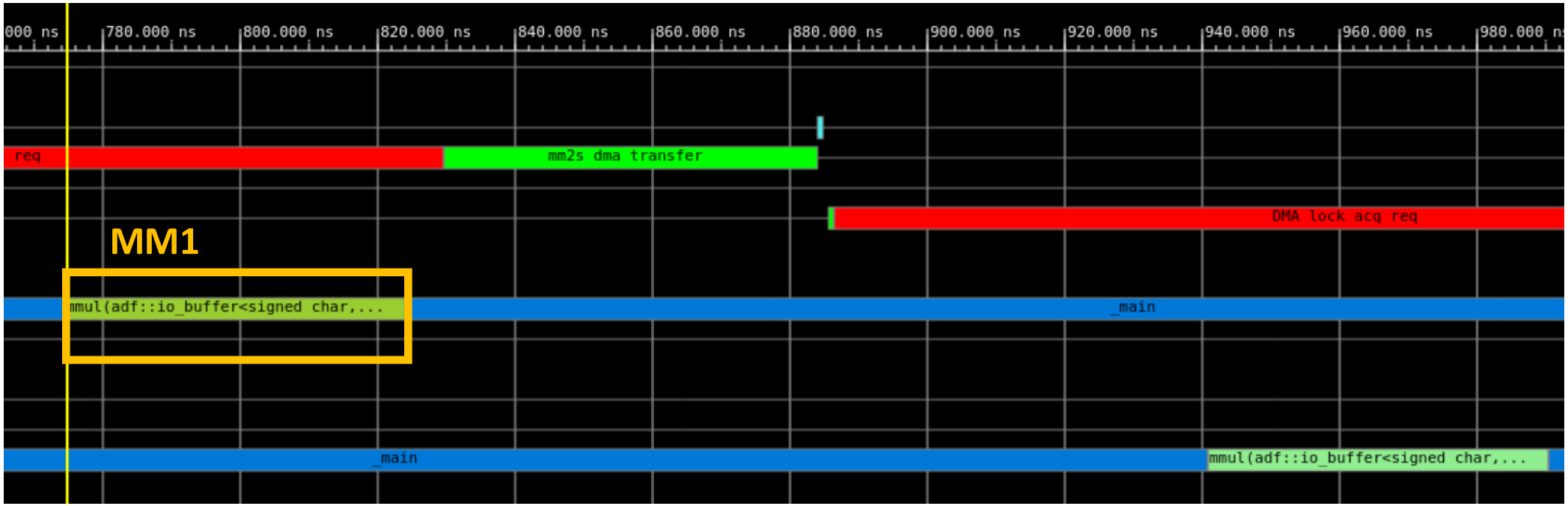
Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile



Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile



Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

	Ideal
MM1	25.6ns
MM2	25.6ns
comm	51.2ns
Total	102.4ns

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

	Ideal	Profiled
MM1	25.6ns	49.2 ns
MM2	25.6ns	50.4 ns
comm	51.2ns	54.2 ns
Total	102.4ns	215.9 ns

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

	Ideal	Profiled	Error
MM1	25.6ns	49.2 ns	92%
MM2	25.6ns	50.4 ns	96%
comm	51.2ns	54.2 ns	6%
Total	102.4ns	215.9 ns	110%

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

	Ideal	Profiled	Error
MM1	25.6ns	49.2 ns	92%
MM2	25.6ns	50.4 ns	96%
comm	51.2ns	54.2 ns	6%
Total	102.4ns	215.9 ns	110%

→ **Instr latency, prologue/epilogue in kernel**

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

	Ideal	Profiled	Error
MM1	25.6ns	49.2 ns	92%
MM2	25.6ns	50.4 ns	96%
comm	51.2ns	54.2 ns	6%
Total	102.4ns	215.9 ns	110%

→ **Instr latency, prologue/epilogue in kernel**

↗ **Sync overhead**

Other Gaps: to Estimate the Performance

Example: MM1: 8x32x32 →8x32 intermediate data → MM2: 8x32x32, weights are preloaded to AIE tile

	Ideal	Profiled	Error
MM1	25.6ns	49.2 ns	92%
MM2	25.6ns	50.4 ns	96%
comm	51.2ns	54.2 ns	6%
Total	102.4ns	215.9 ns	110%

Instr latency, prologue/epilogue in kernel

Sync overhead

⚠ Performance estimation and tuning can be hard

u-ORCA: Optimizing Acceleration for Microsecond-Scale Deep Neural Network Inference on ACAP

us-scale NN inference can not be reached by simply scaling down the existing accelerator architecture

On-chip communication bottleneck

(DMA transfer, 32-bit/cycle)

Inefficient kernel implementation for full model

Inaccurate performance model

u-ORCA: Optimizing Acceleration for Microsecond-Scale Deep Neural Network Inference on ACAP

us-scale NN inference can not be reached by simply scaling down the existing accelerator architecture

On-chip communication bottleneck

(DMA transfer, 32-bit/cycle)

→ support transferring intermediate data
(1) directly between 2 layers as AIE arrays (w/o L2 access)
(2) With local mem or cascade communication enabled
→ Up to 16x bandwidth improvement

Inefficient kernel implementation for full model →

Inaccurate performance model →

u-ORCA: Optimizing Acceleration for Microsecond-Scale Deep Neural Network Inference on ACAP

us-scale NN inference can not be reached by simply scaling down the existing accelerator architecture

On-chip communication bottleneck

(DMA transfer, 32-bit/cycle)

→

support transferring intermediate data
(1) directly between 2 layers as AIE arrays (w/o L2 access)
(2) With local mem or cascade communication enabled
→ Up to 16x bandwidth improvement

Inefficient kernel implementation for full model

→

(1) Global aggregation support for Deepset models
(2) Bias and ReLU support
(3) Quantization support
…

Inaccurate performance model

→

u-ORCA: Optimizing Acceleration for Microsecond-Scale Deep Neural Network Inference on ACAP

us-scale NN inference can not be reached by simply scaling down the existing accelerator architecture

On-chip communication bottleneck

(DMA transfer, 32-bit/cycle)

→ support transferring intermediate data
(1) directly between 2 layers as AIE arrays (w/o L2 access)
(2) With local mem or cascade communication enabled
→ Up to 16x bandwidth improvement

Inefficient kernel implementation for full model

→ (1) Global aggregation support for Deepset models
(2) Bias and ReLU support
(3) Quantization support
…

Inaccurate performance model

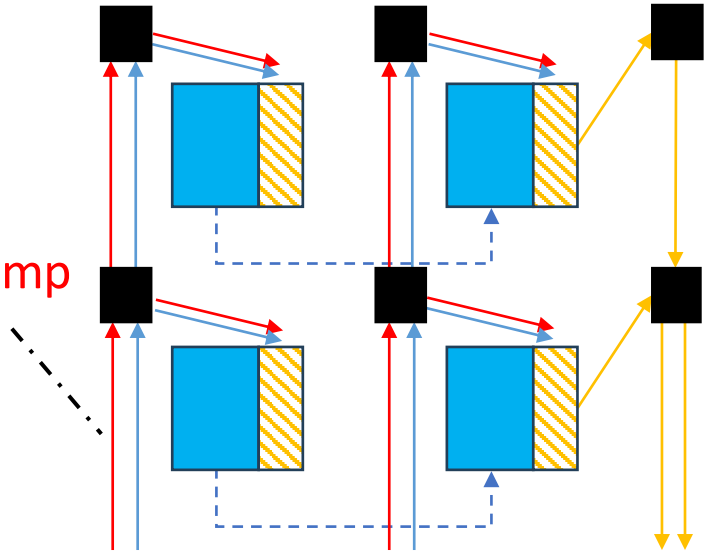
→ Fine-grained performance model

Design space exploration

u-ORCA: Single Kernel Architecture

Input Weight Output Partial Results DMA Cascade Parameter (pre-loaded)

32 b/cycle
Sync w/t comp

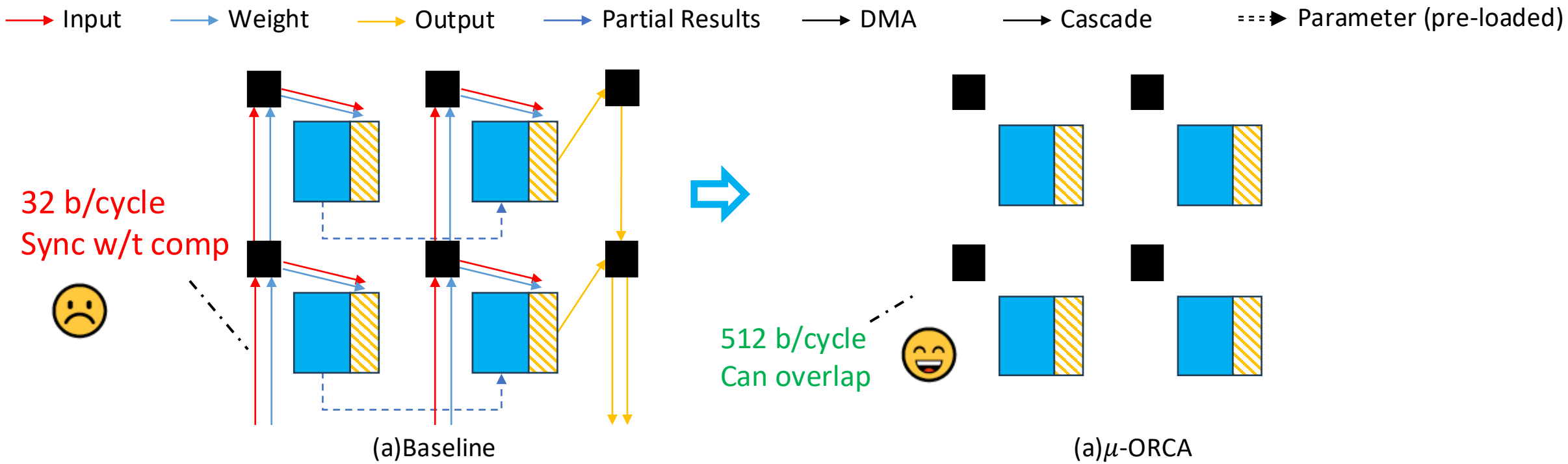


(a)Baseline

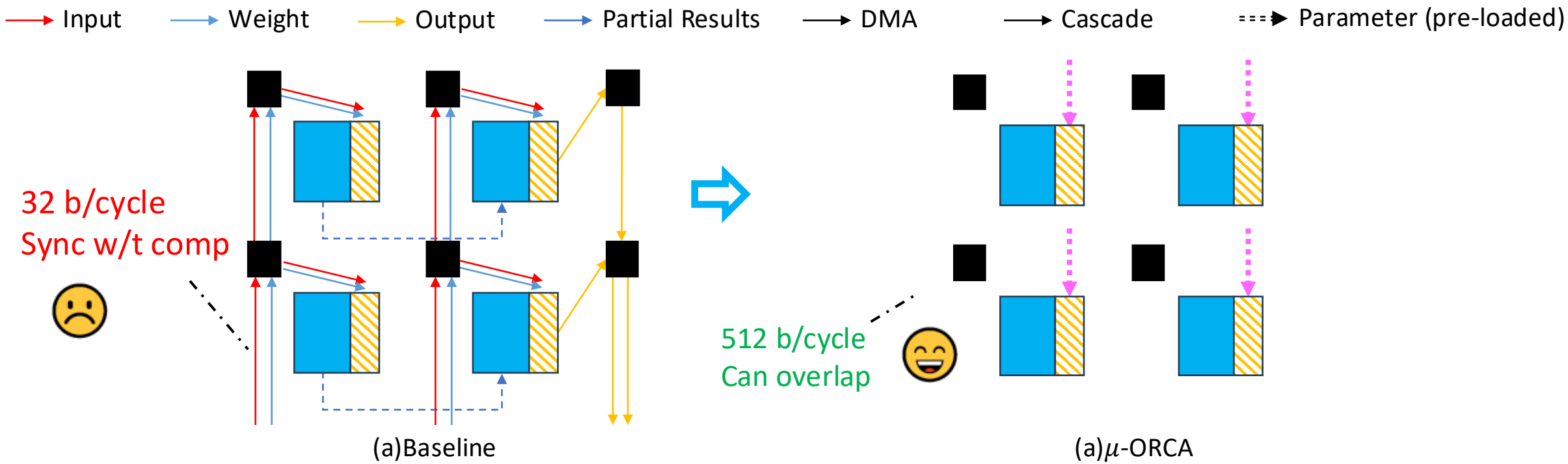
512 b/cycle
Can overlap

(a)μ-ORCA

u-ORCA: Single Kernel Architecture

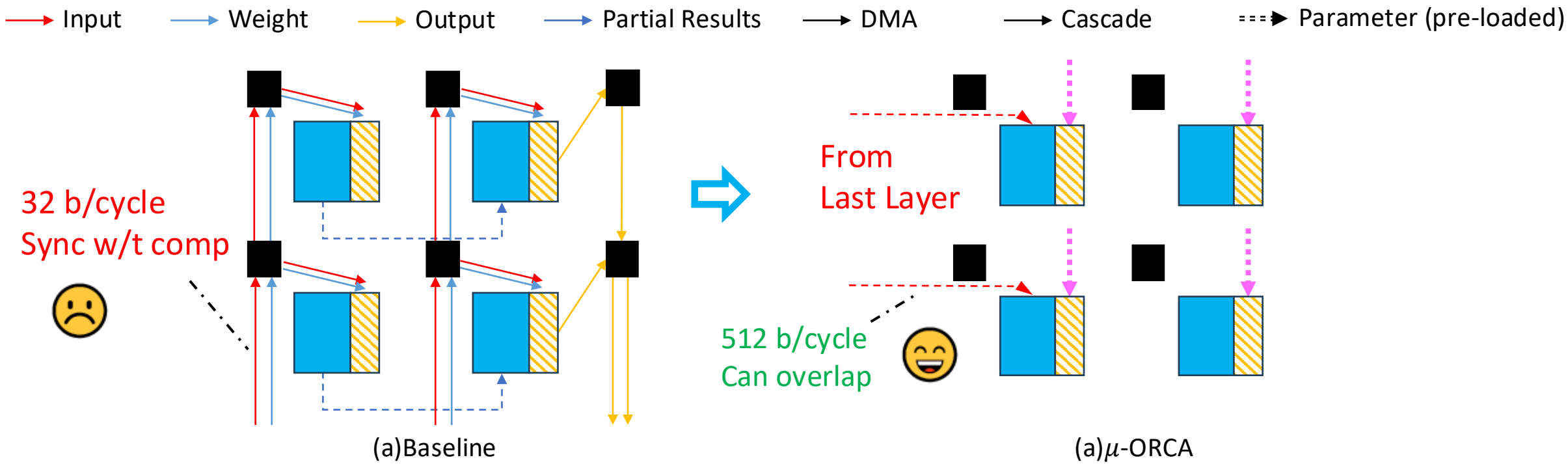


u-ORCA: Single Kernel Architecture



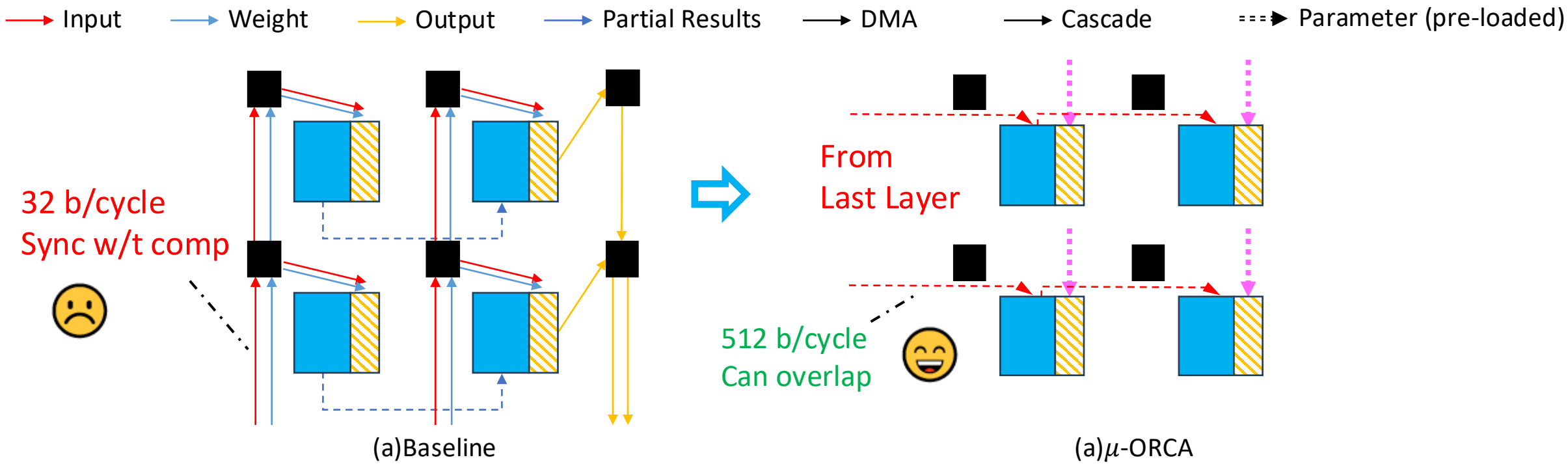
- Pre-load the weight and bias

u-ORCA: Single Kernel Architecture



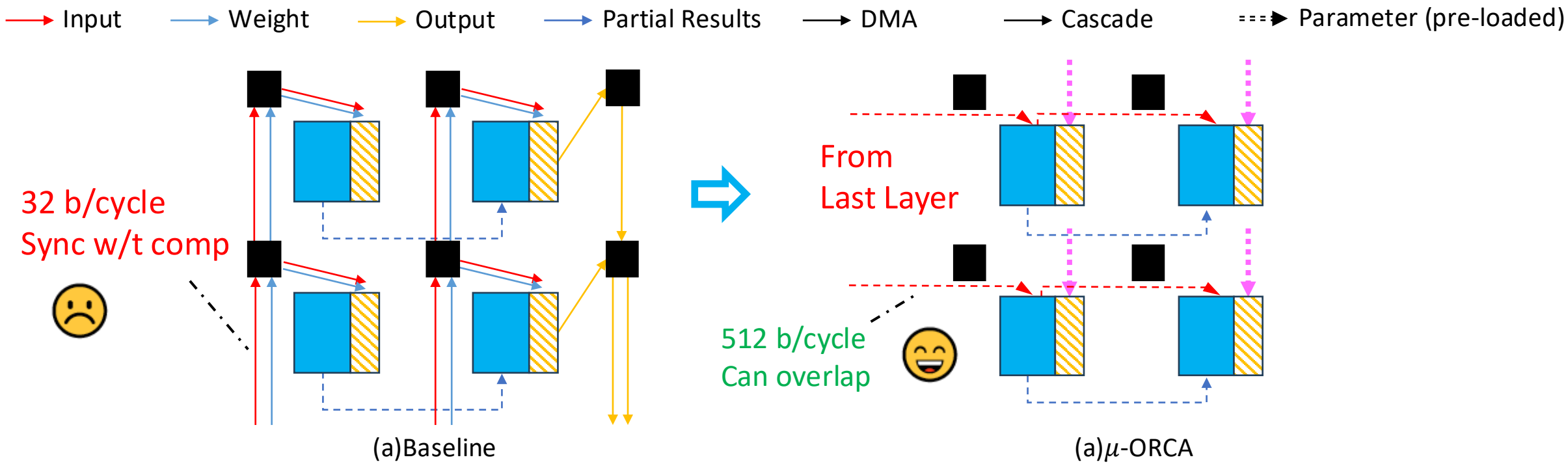
- Pre-load the weight and bias
- Load the input directly from last layer

u-ORCA: Single Kernel Architecture



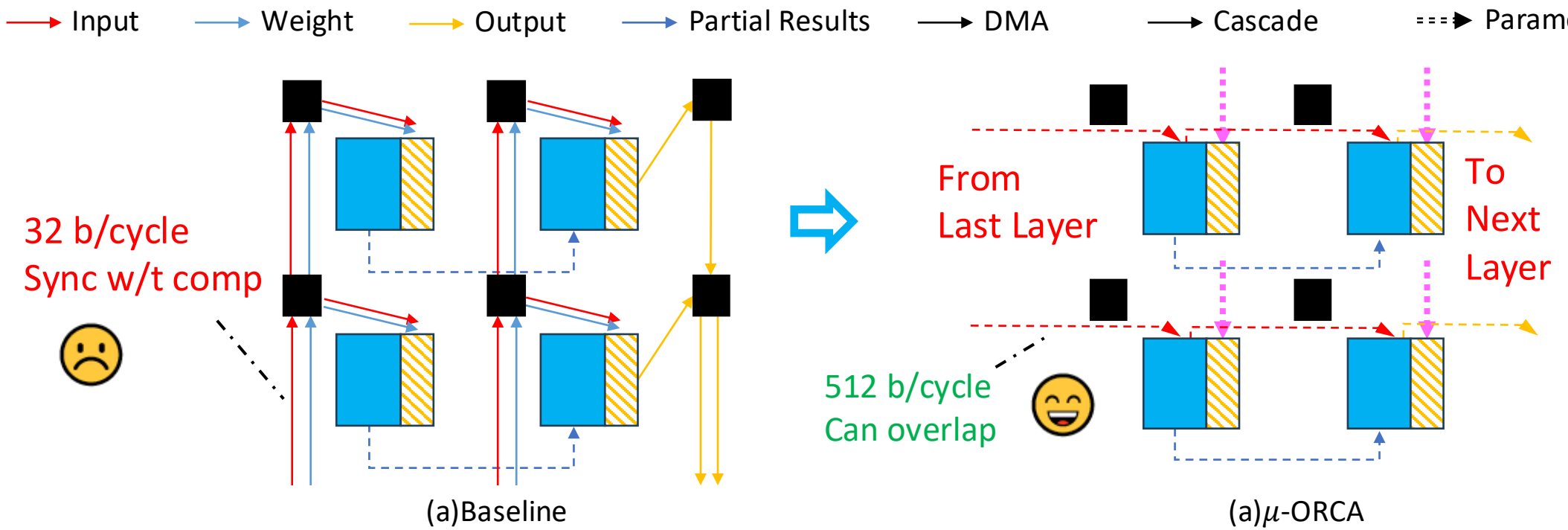
- Pre-load the weight and bias
- Load the input directly from last layer
- Spread the input via cascade

u-ORCA: Single Kernel Architecture



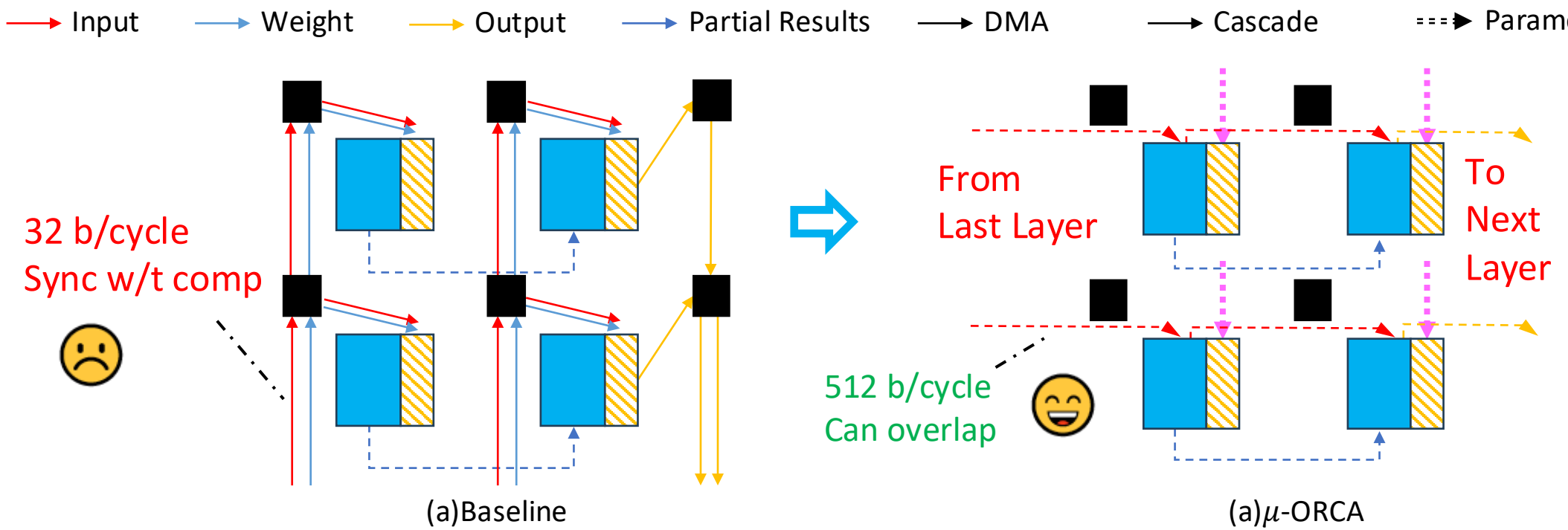
- Pre-load the weight and bias
- Load the input directly from last layer
- Spread the input via cascade
- Compute, accumulate partial results via cascade

u-ORCA: Single Kernel Architecture



- Pre-load the weight and bias
- Load the input directly from last layer
- Spread the input via cascade
- Compute, accumulate partial results via cascade
- Send results directly via cascade

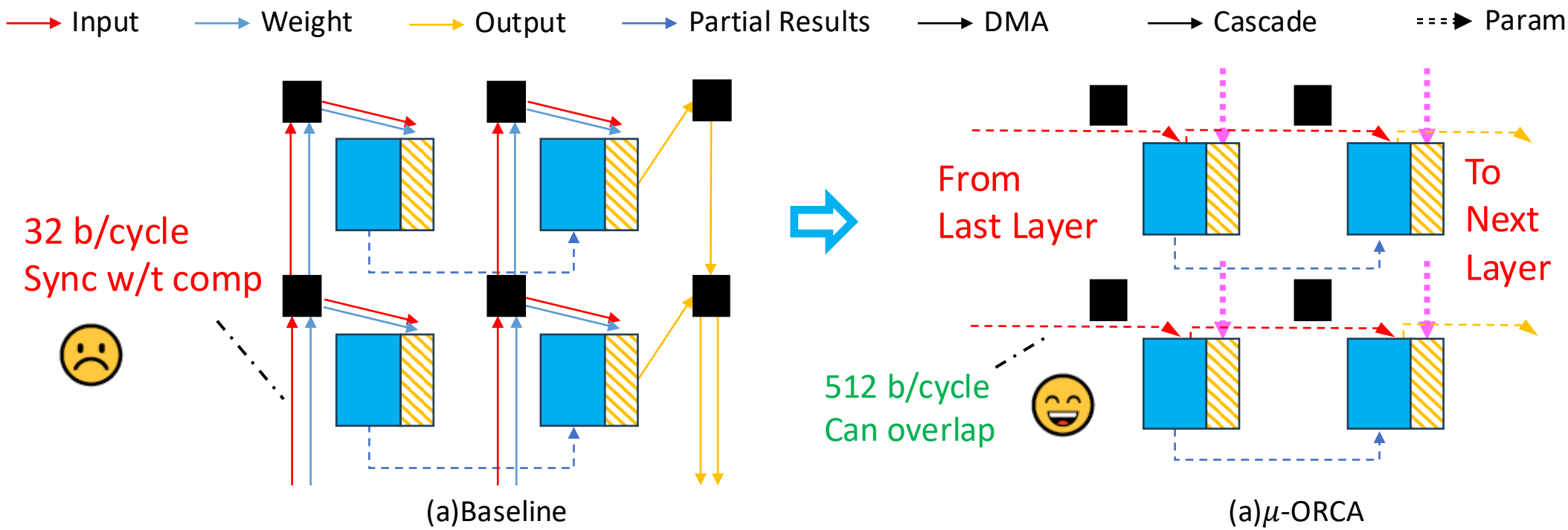
u-ORCA: Single Kernel Architecture



Quick Math: 8x32x64 INT8 kernel:

- Pre-load the weight and bias
- Load the input directly from last layer
- Spread the input via cascade
- Compute, accumulate partial results via cascade
- Send results directly via cascade

u-ORCA: Single Kernel Architecture

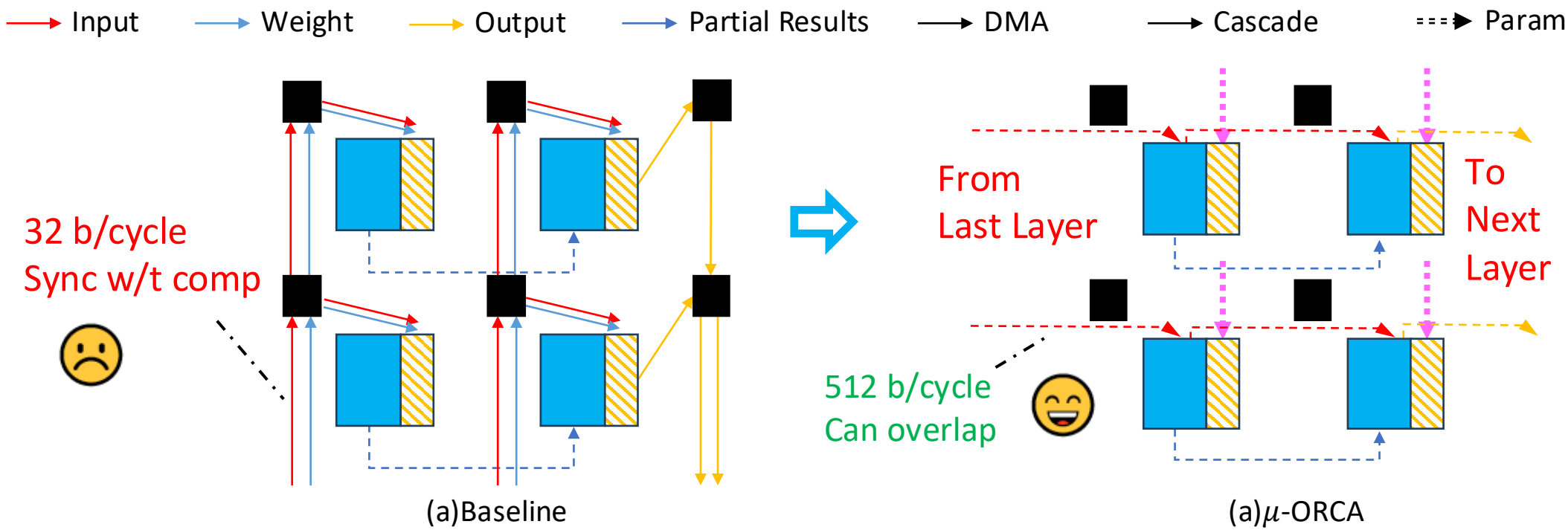


Quick Math: 8x32x64 INT8 kernel:

- 4 cycles to load the activation
- 64 cycles to compute
- 8 cycles to store the results

- Pre-load the weight and bias
- Load the input directly from last layer
- Spread the input via cascade
- Compute, accumulate partial results via cascade
- Send results directly via cascade

u-ORCA: Single Kernel Architecture



Quick Math: 8x32x64 INT8 kernel:

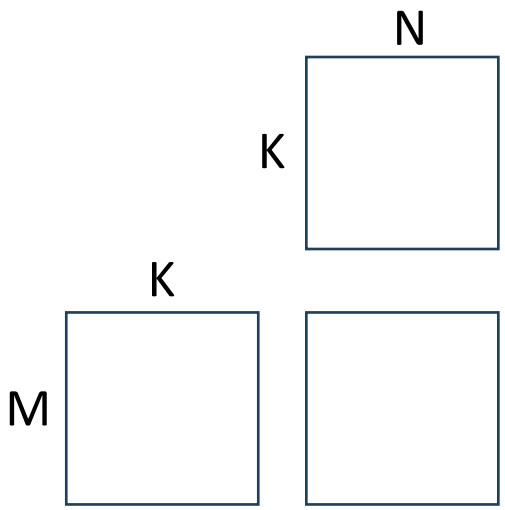
- 4 cycles to load the activation
- 64 cycles to compute
- 8 cycles to store the results

→4+64+8=76 cycles, 9.2x gain over the baseline

→preload weights in baseline: 76 vs. 256, 3.4x gain

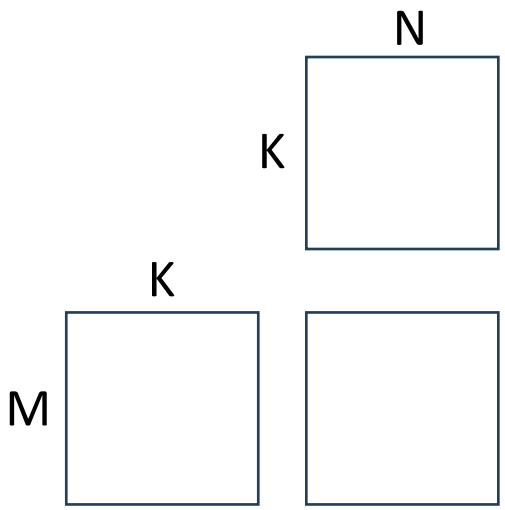
- Pre-load the weight and bias
- Load the input directly from last layer
- Spread the input via cascade
- Compute, accumulate partial results via cascade
- Send results directly via cascade

Scale to Multiple AIE Tiles



Current layer

Scale to Multiple AIE Tiles

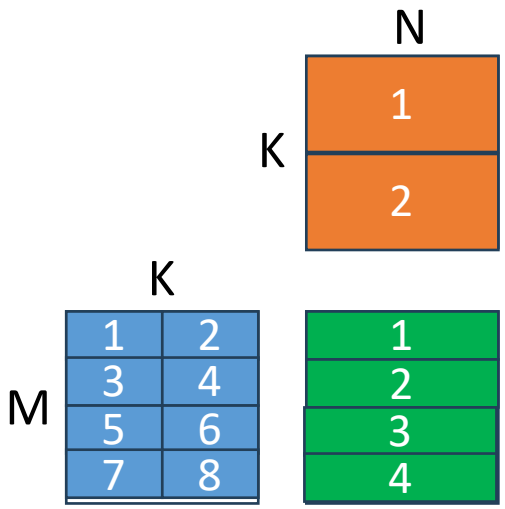


Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

Current layer

Scale to Multiple AIE Tiles

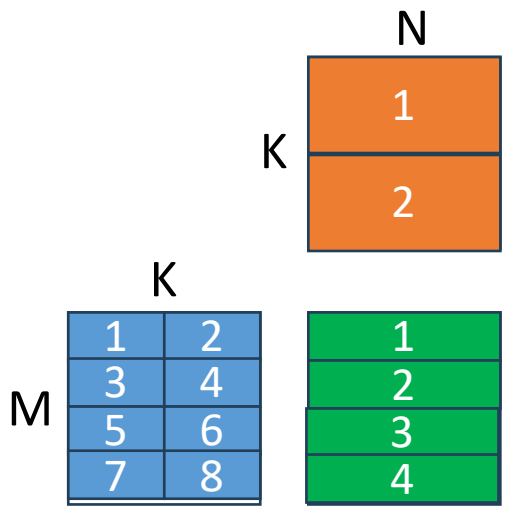


Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

Current layer

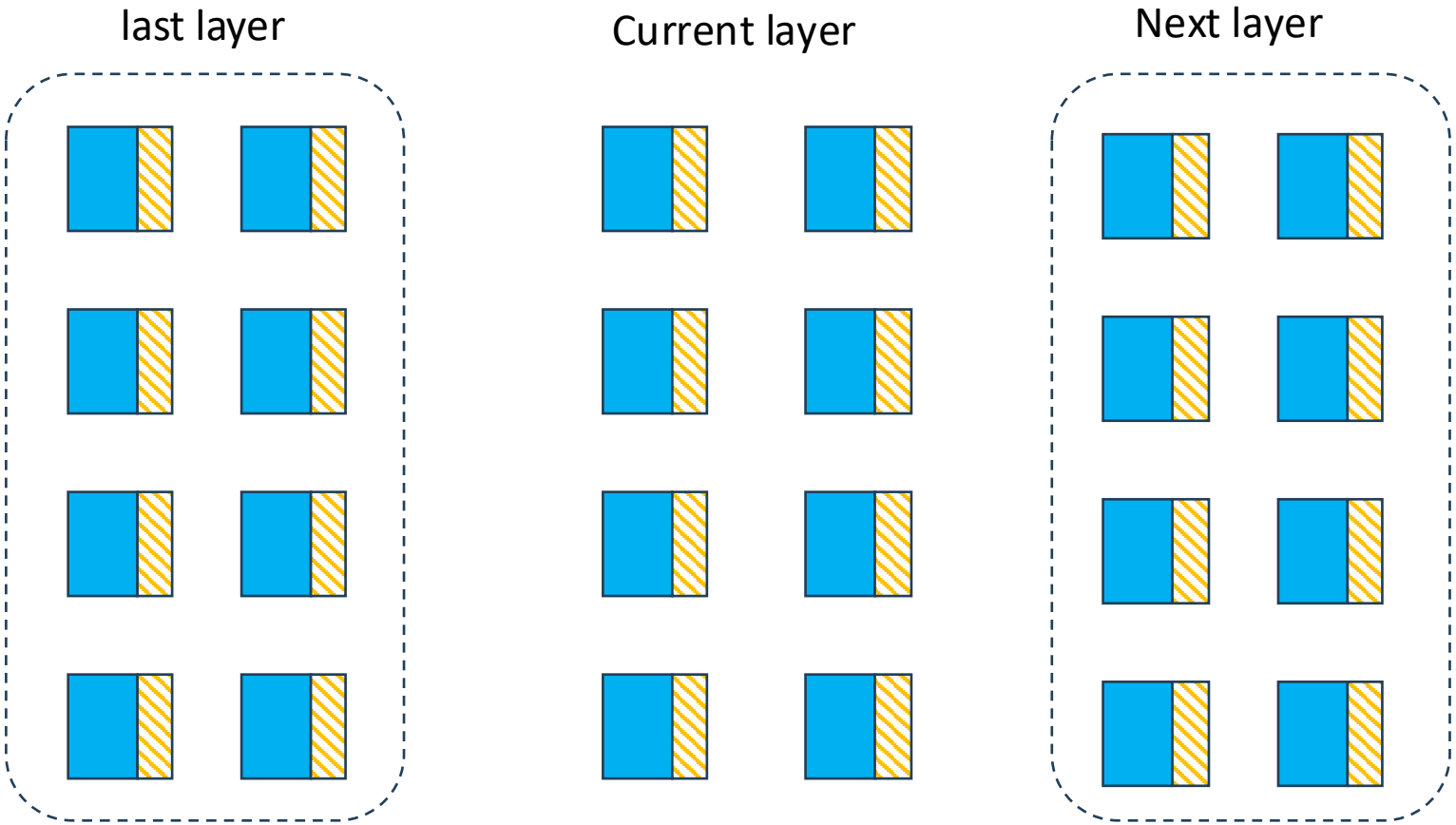
Scale to Multiple AIE Tiles



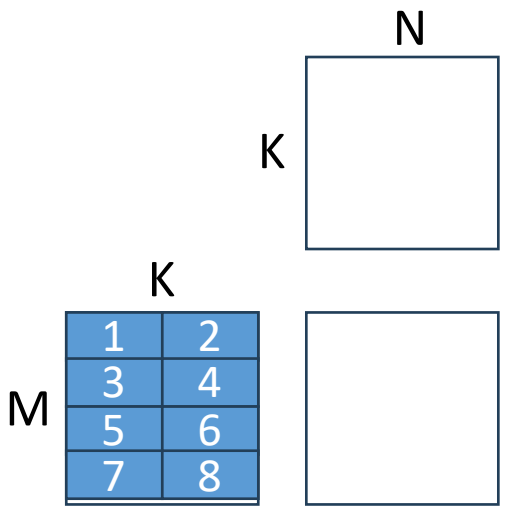
Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

For consecutive two layers, if A=A’ and C=C’=1:



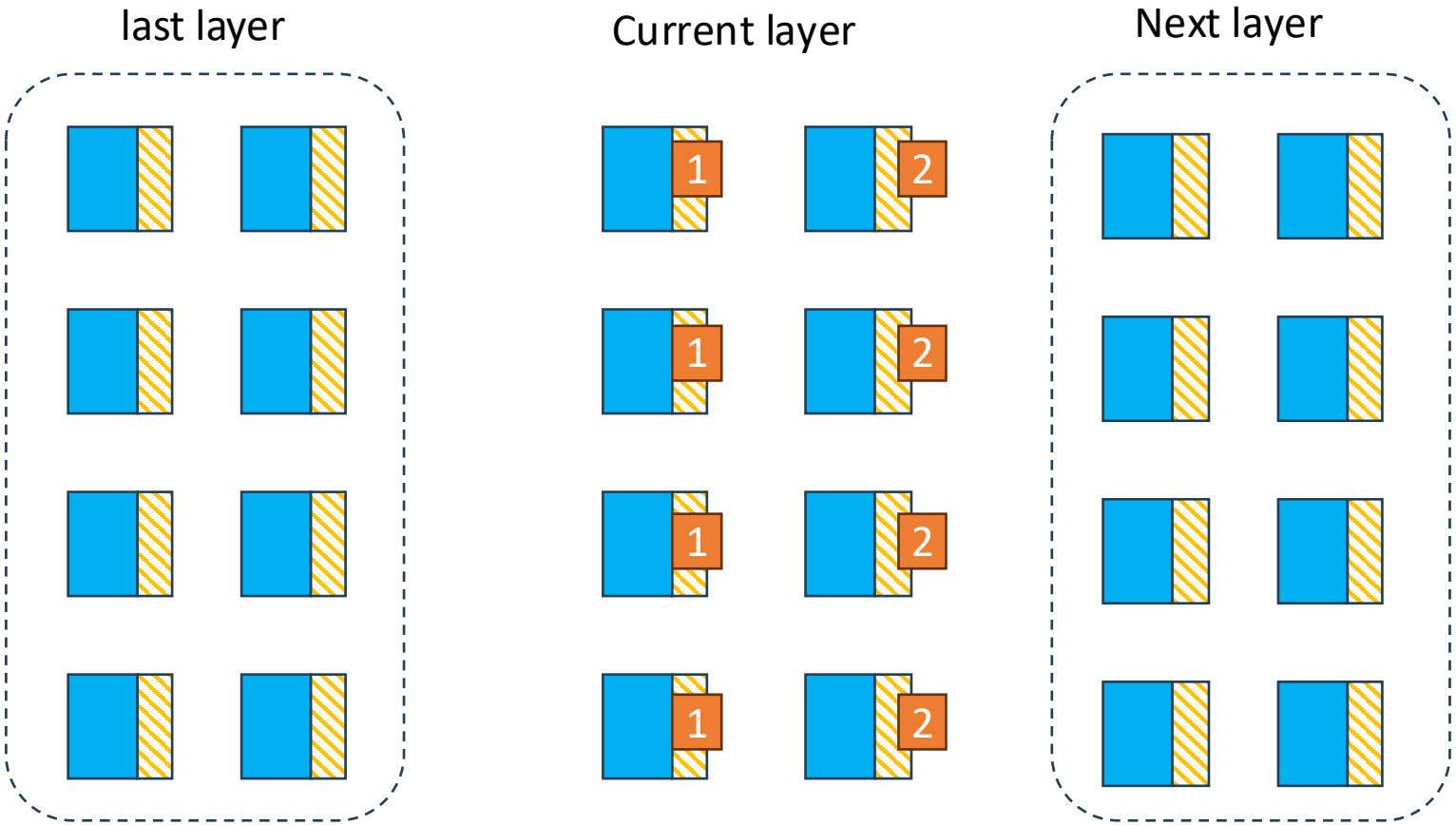
Scale to Multiple AIE Tiles



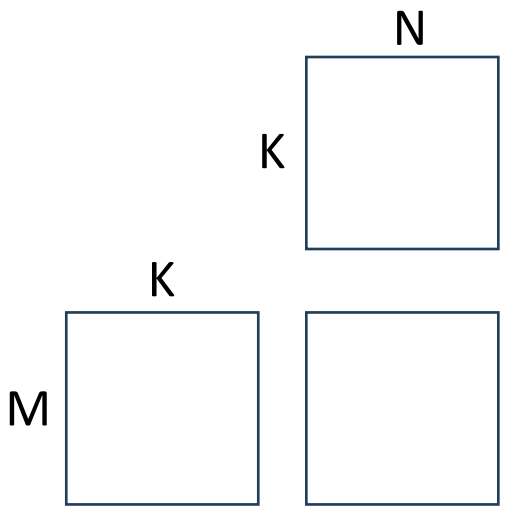
Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

For consecutive two layers, if A=A' and C=C'=1:



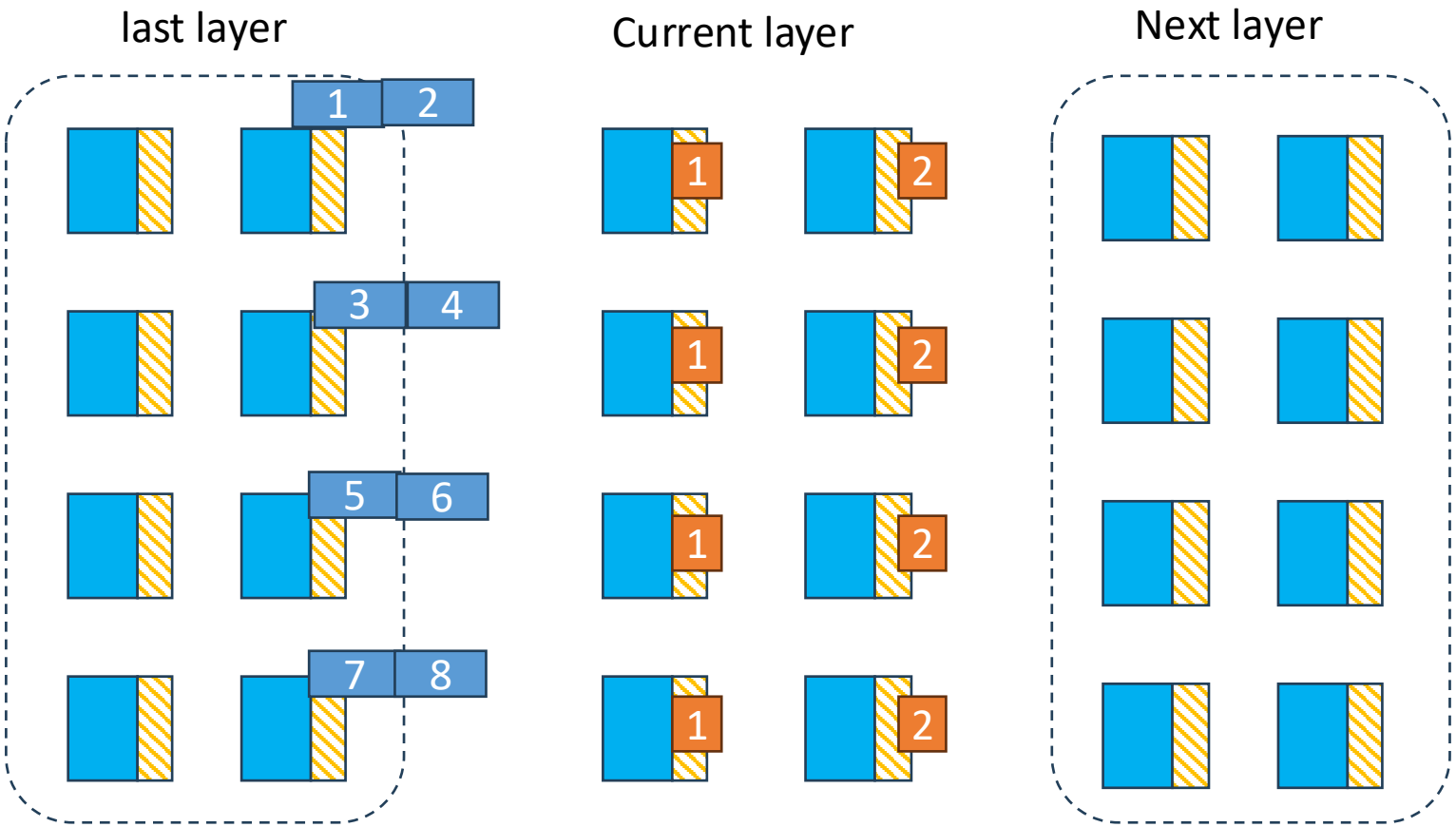
Scale to Multiple AIE Tiles



Partition M,K,N dim to A,B,C pieces

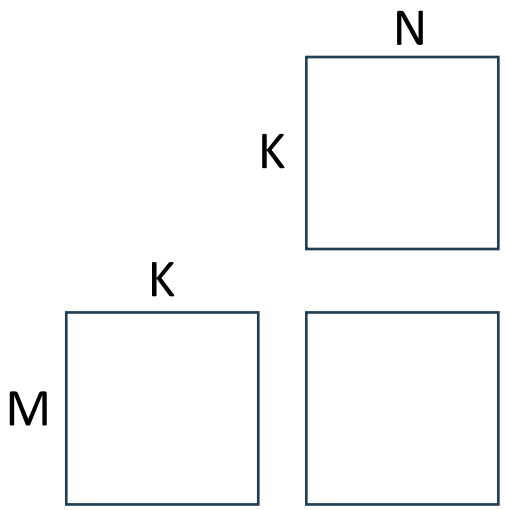
E.g. A=4, B=2, C=1

For consecutive two layers, if A=A’ and C=C’=1:



Input data distributed in the last layer

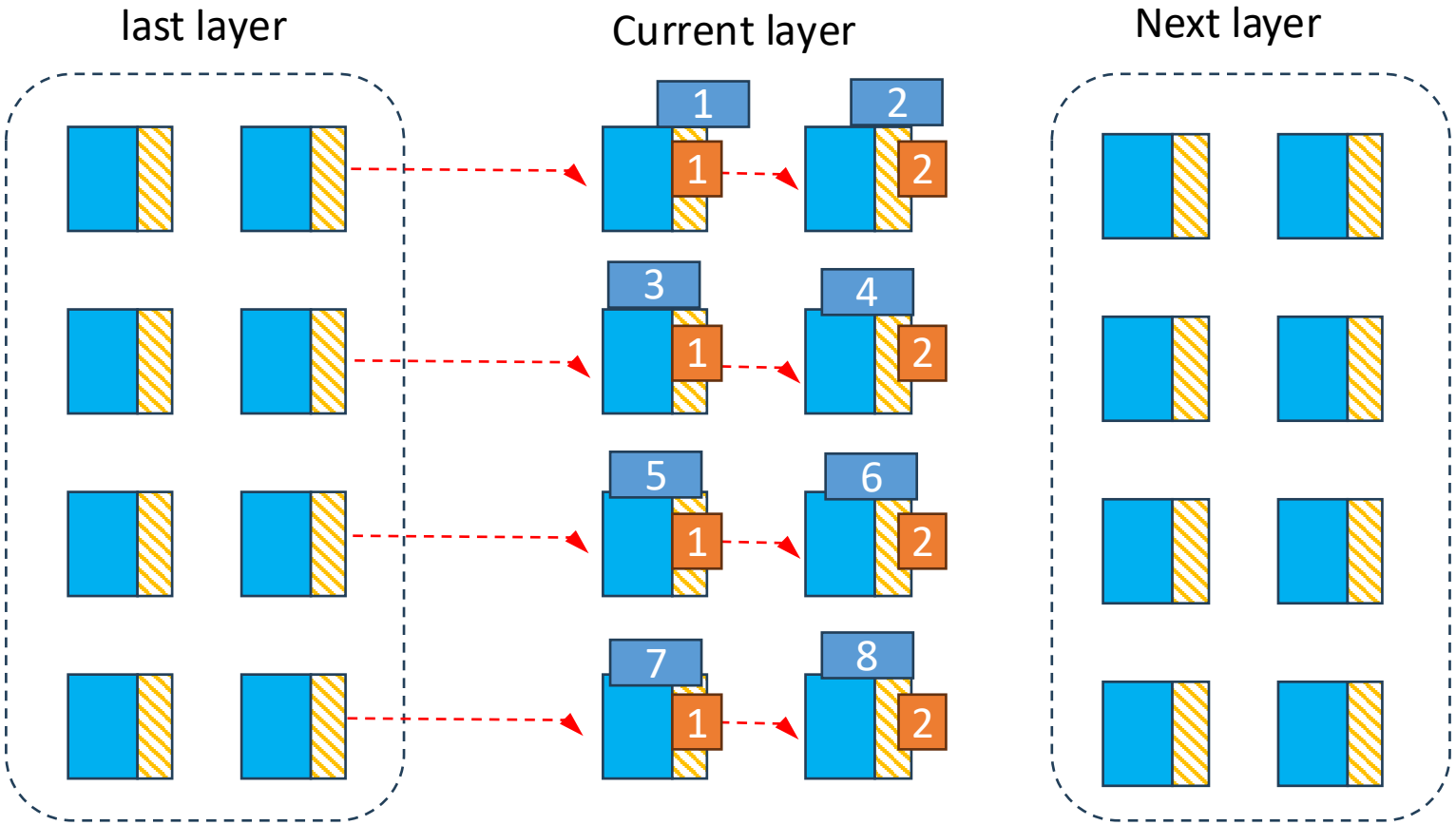
Scale to Multiple AIE Tiles



Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

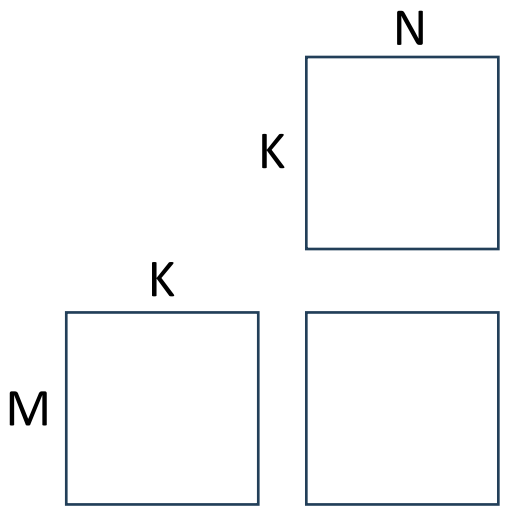
For consecutive two layers, if A=A' and C=C'=1:



Input data distributed in the last layer

Horizontal data transfer:
Can use cascade!

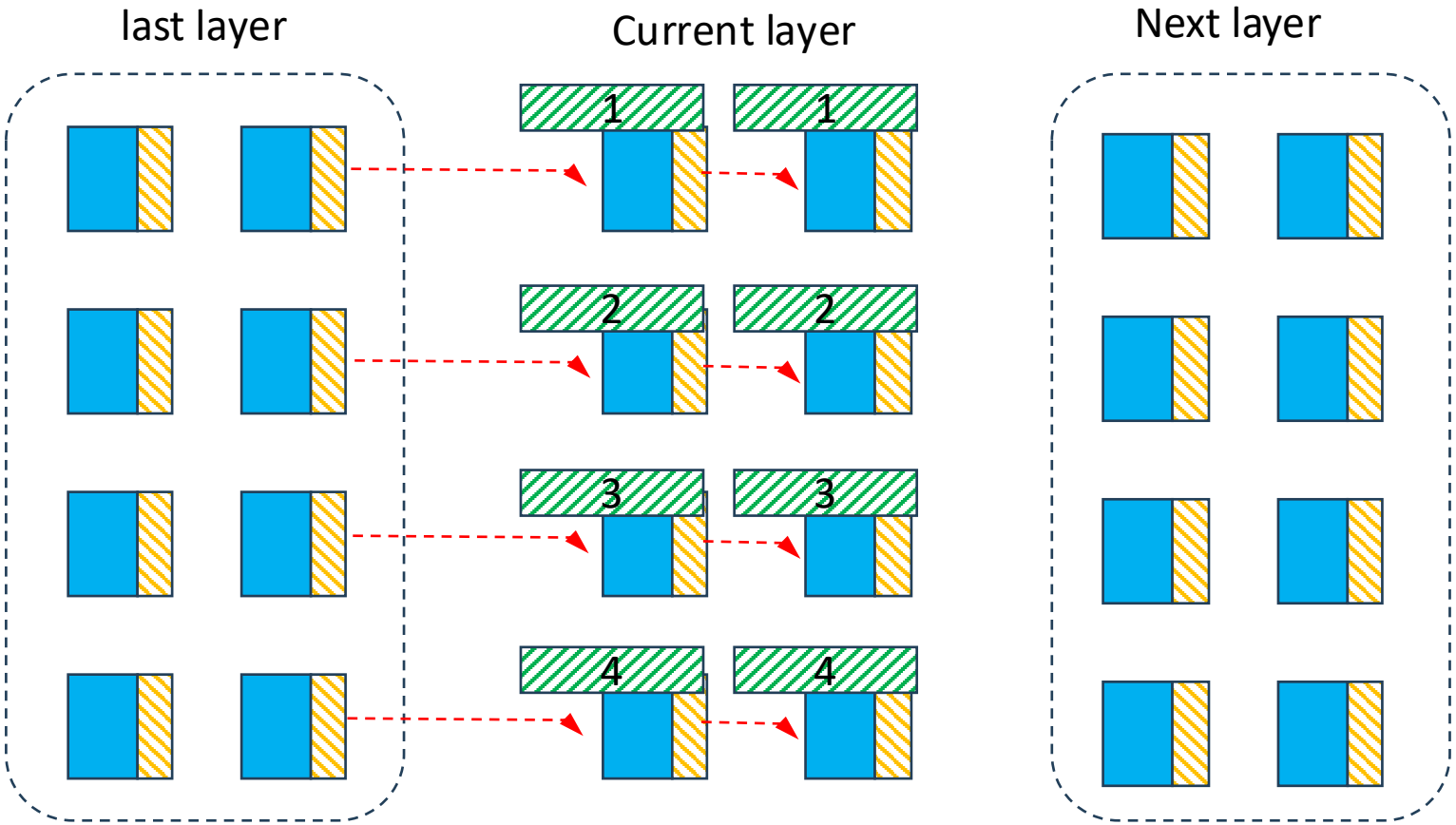
Scale to Multiple AIE Tiles



Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

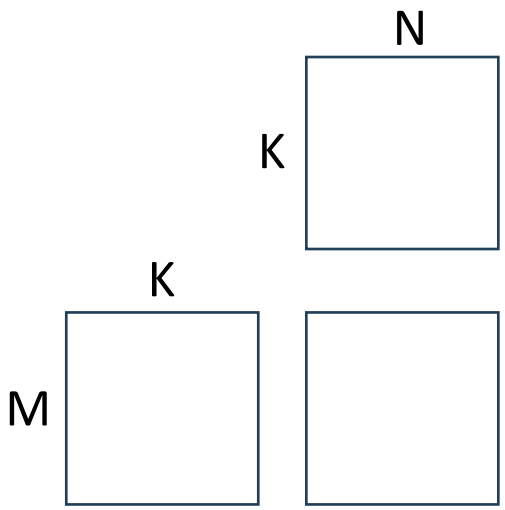
For consecutive two layers, if A=A' and C=C'=1:



Input data distributed in the last layer

Horizontal data transfer:
Can use cascade!

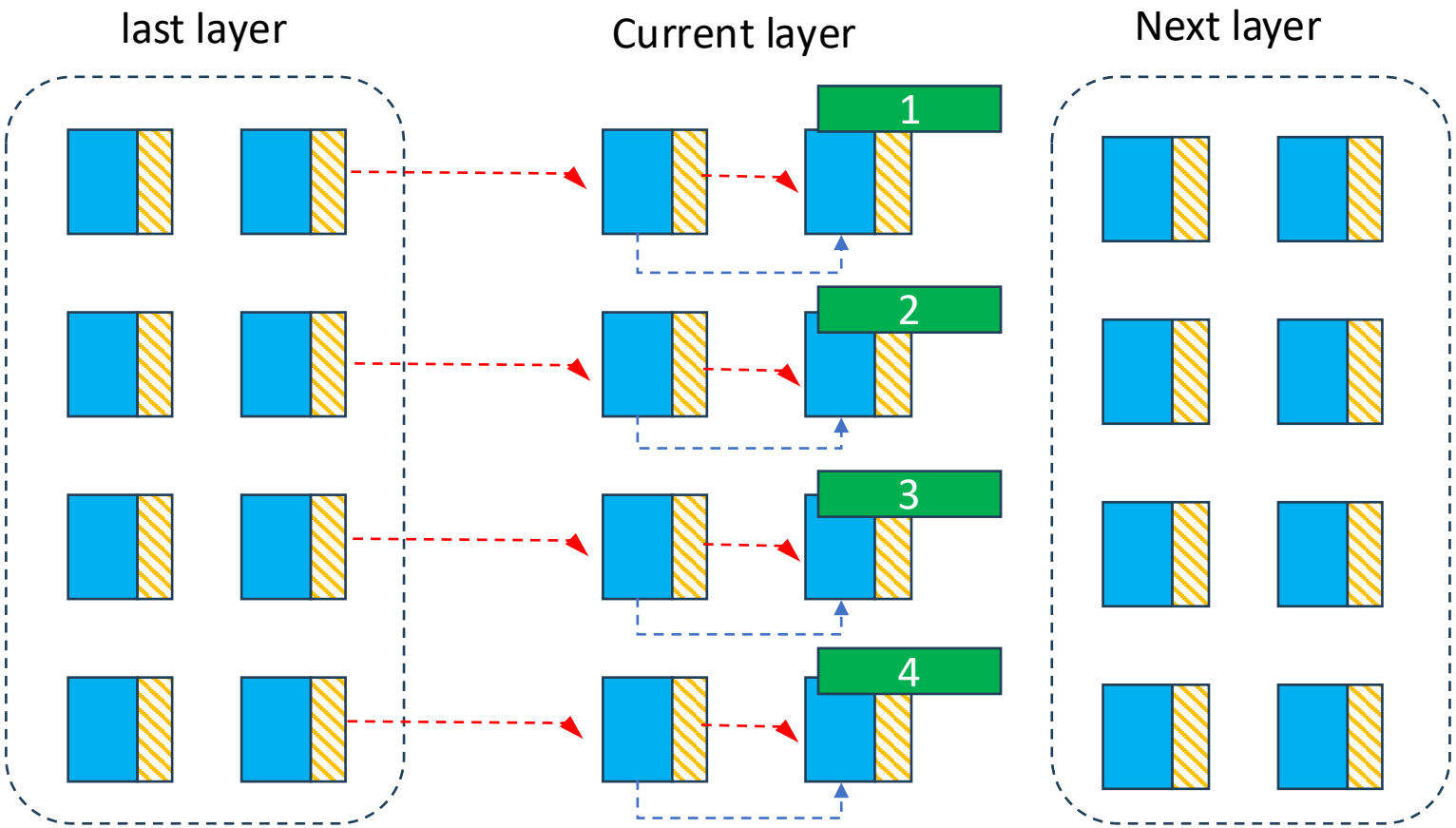
Scale to Multiple AIE Tiles



Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

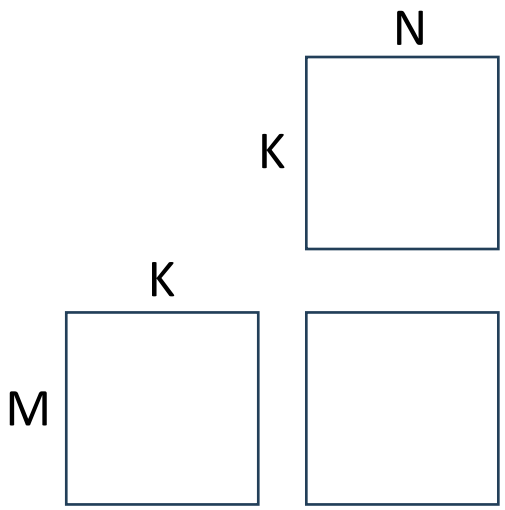
For consecutive two layers, if A=A' and C=C'=1:



Input data distributed in the last layer

Horizontal data transfer:
Can use cascade!
During computation:
Cascade reused for partial results

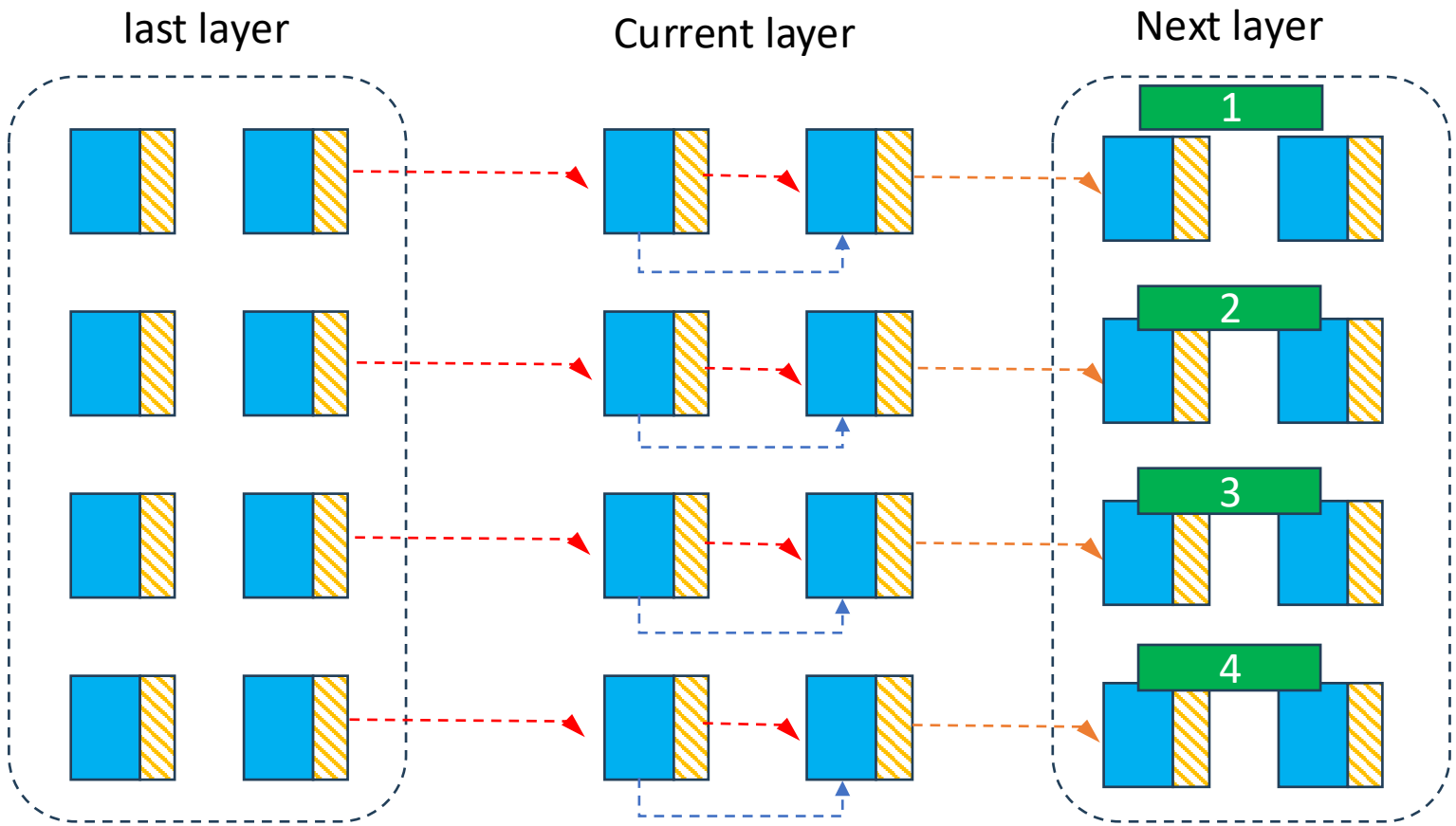
Scale to Multiple AIE Tiles



Partition M,K,N dim to A,B,C pieces

E.g. A=4, B=2, C=1

For consecutive two layers, if A=A' and C=C'=1:



Input data distributed in the last layer

Horizontal data transfer:
Can use cascade!
During computation:
Cascade reused for partial results

Data send to next layer

Computation-communication Tradeoff

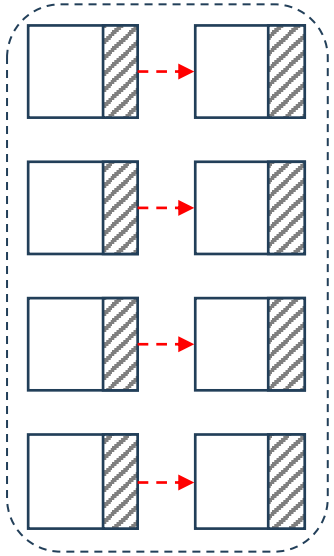
The shape constrain could limit the parallelism…

Computation-communication Tradeoff

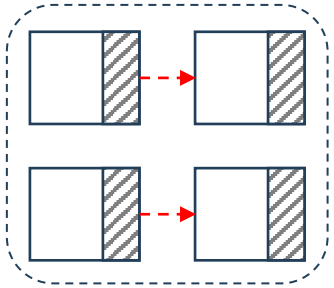
The shape constrain could limit the parallelism…

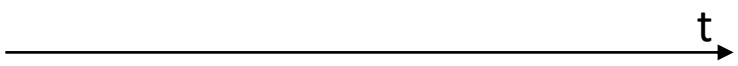
Layer 1:
A,B,C=2,2,2

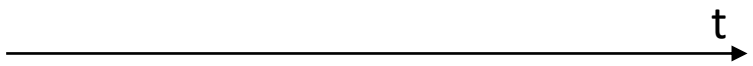
Layer 2:
A,B,C=2,2,1



Layer 2:







DMA transfer

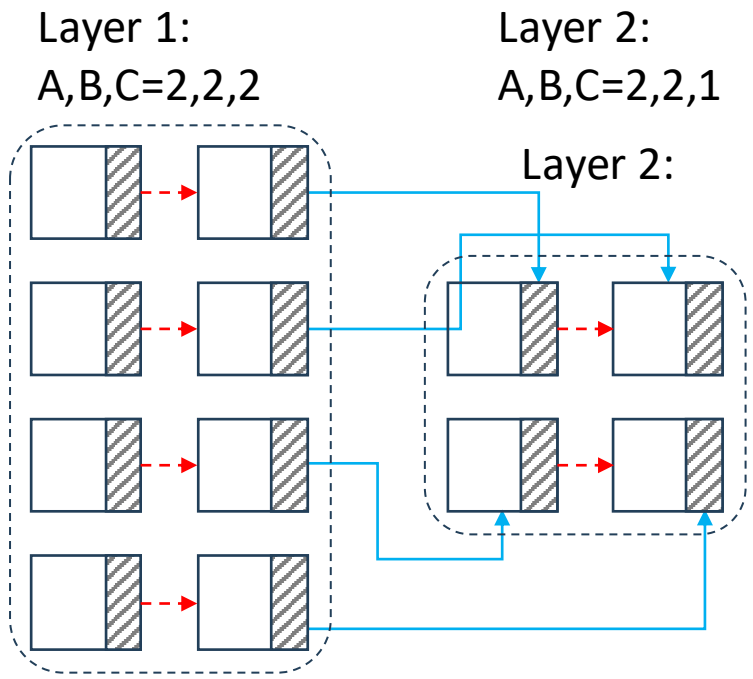
Cascade transfer

AIE tile

layer boundary

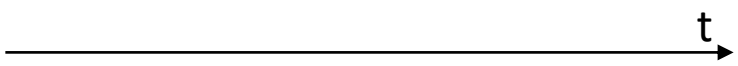
Computation-communication Tradeoff

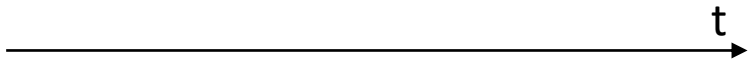
The shape constrain could limit the parallelism…

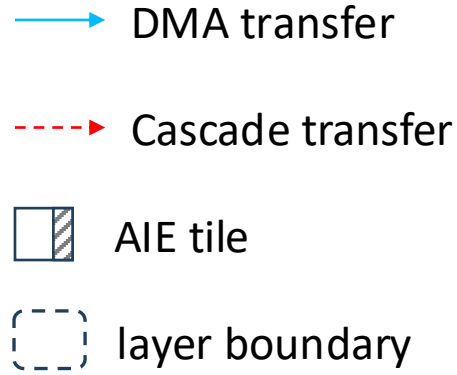


Inconsistent shape and placement:

Fallback to direct DMA

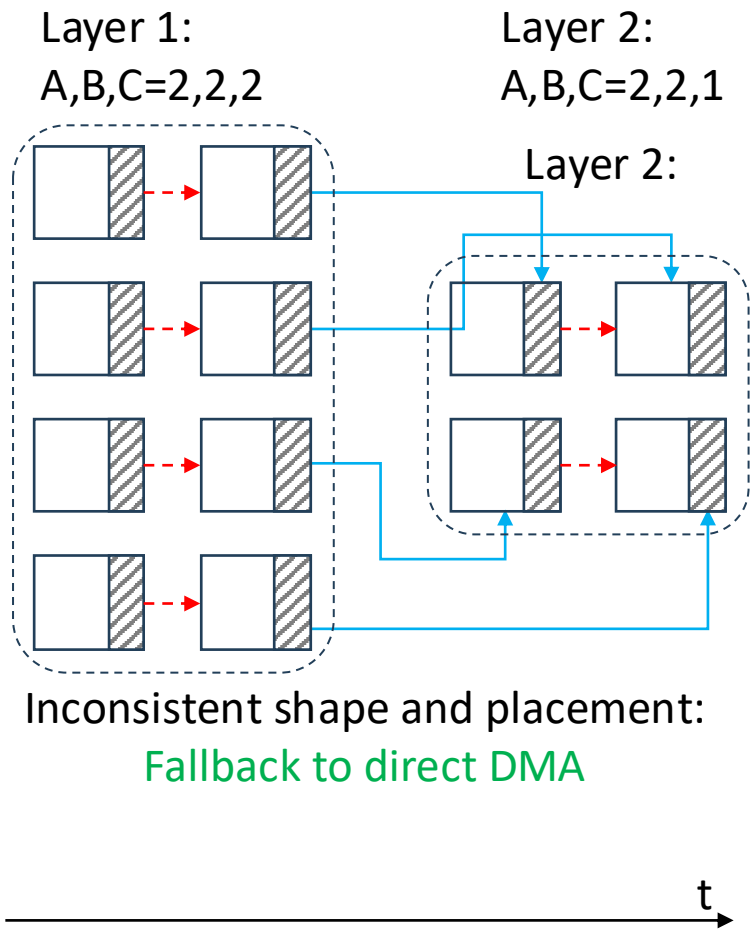






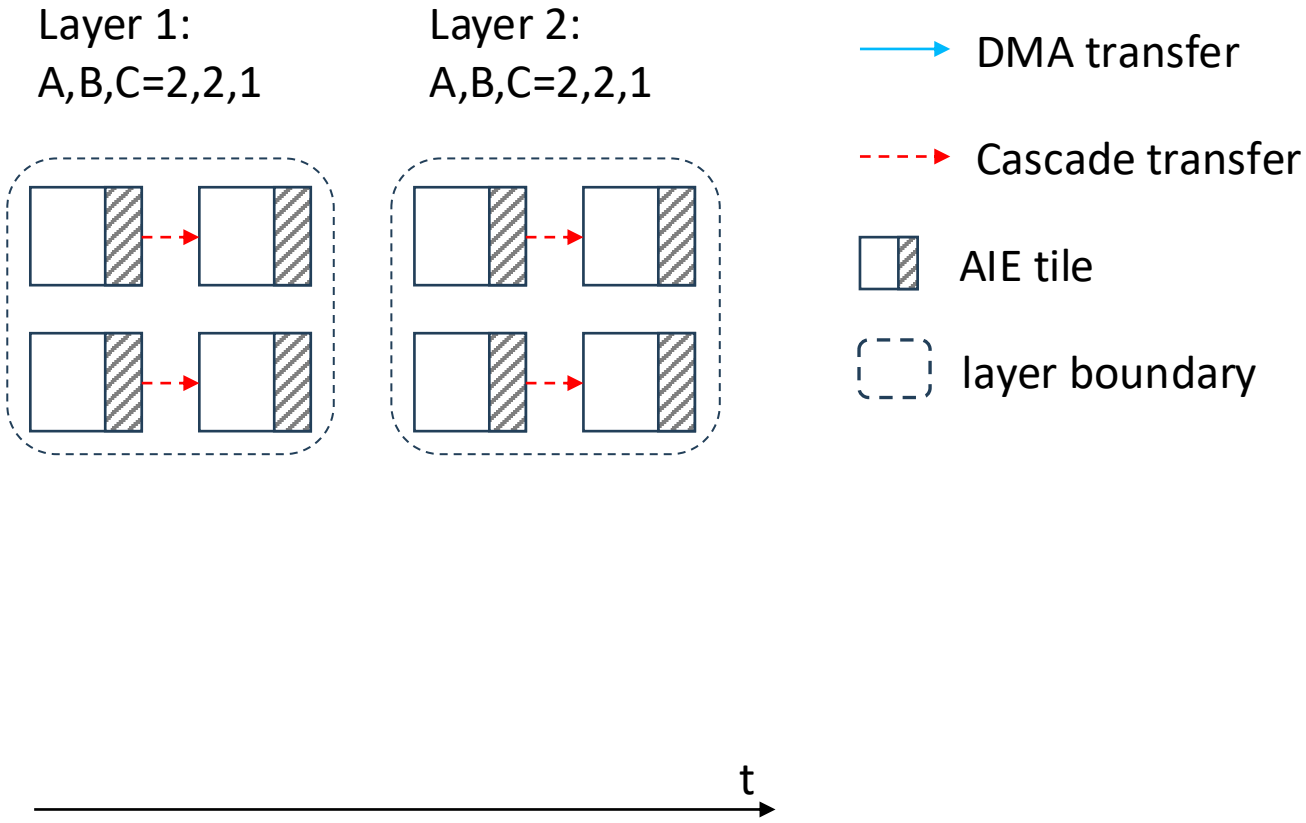
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



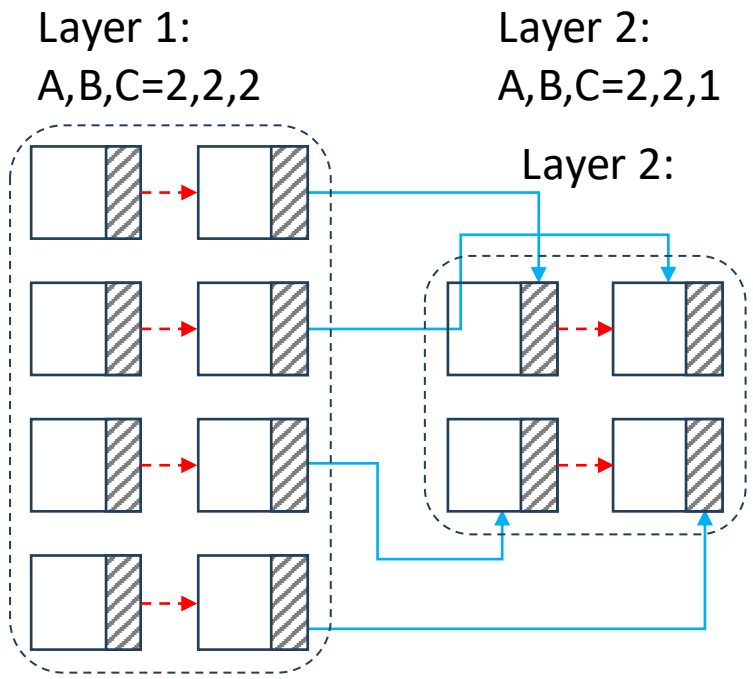
Inconsistent shape and placement:

Fallback to direct DMA



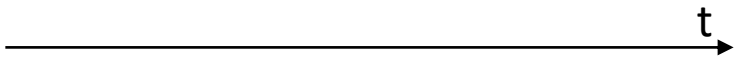
Computation-communication Tradeoff

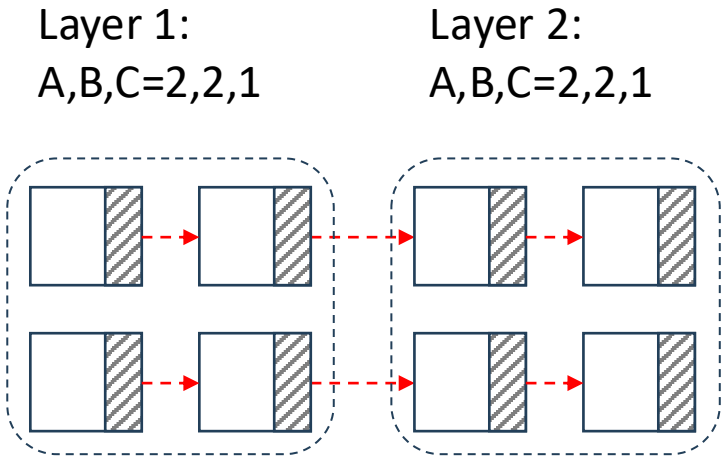
The shape constrain could limit the parallelism…



Inconsistent shape and placement:

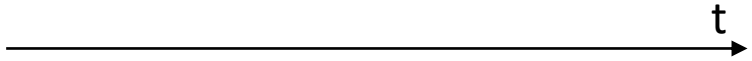
Fallback to direct DMA

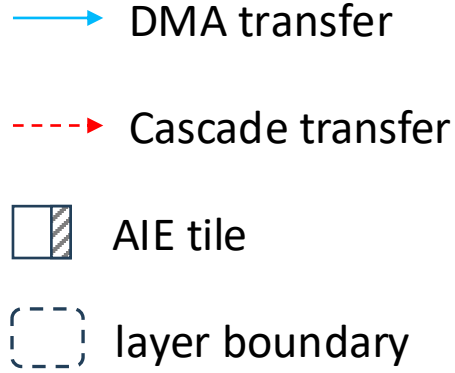




consistent shape and placement:

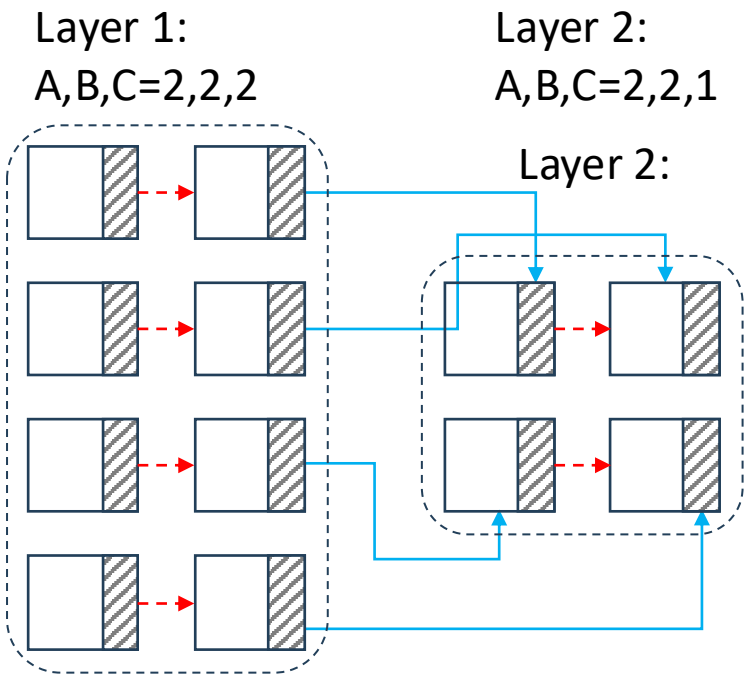
Use cascade





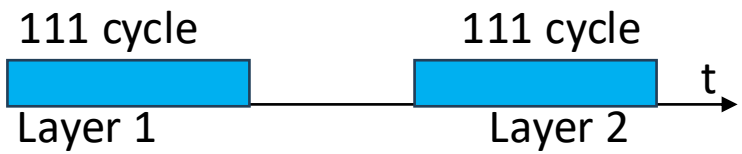
Computation-communication Tradeoff

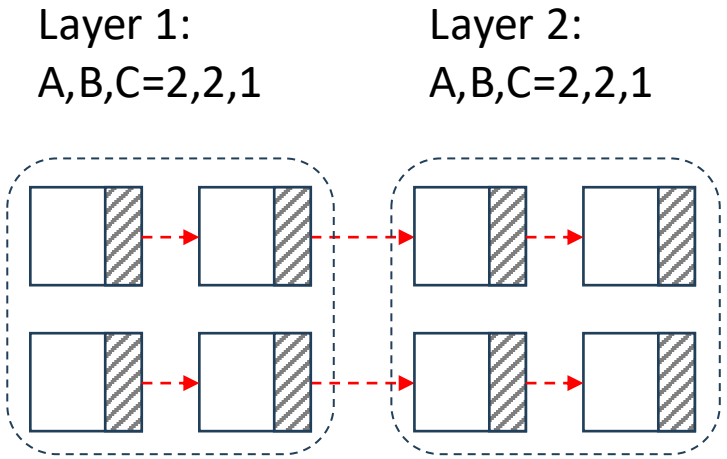
The shape constrain could limit the parallelism…



Inconsistent shape and placement:

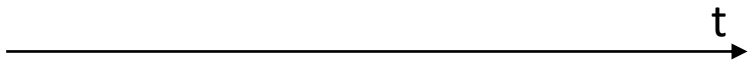
Fallback to direct DMA

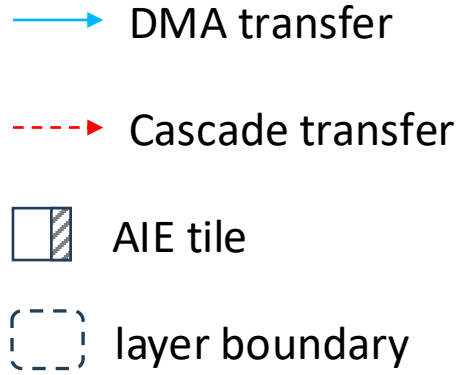




consistent shape and placement:

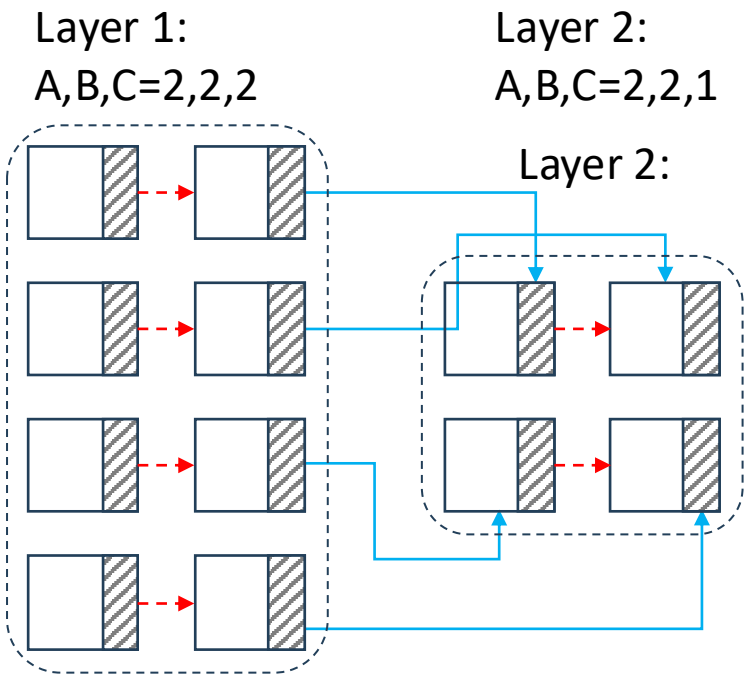
Use cascade





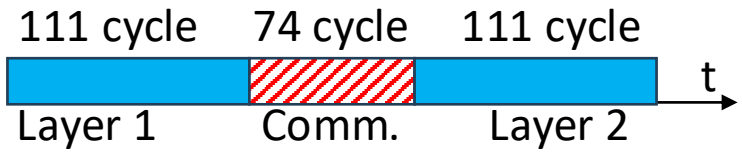
Computation-communication Tradeoff

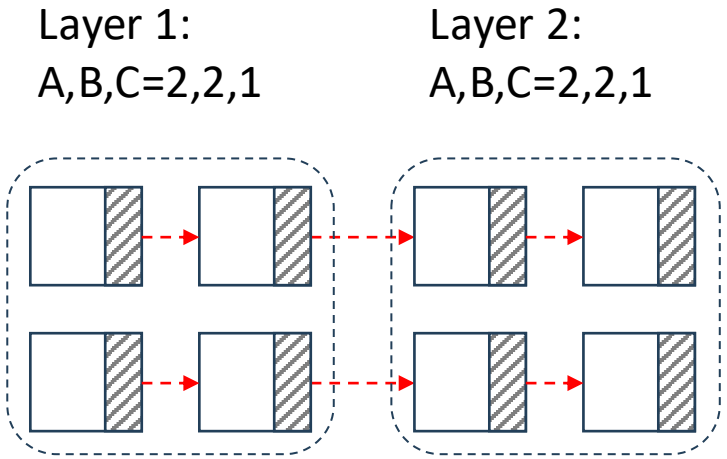
The shape constrain could limit the parallelism…



Inconsistent shape and placement:

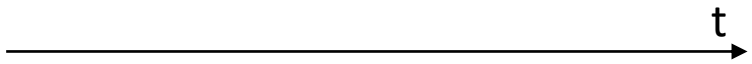
Fallback to direct DMA

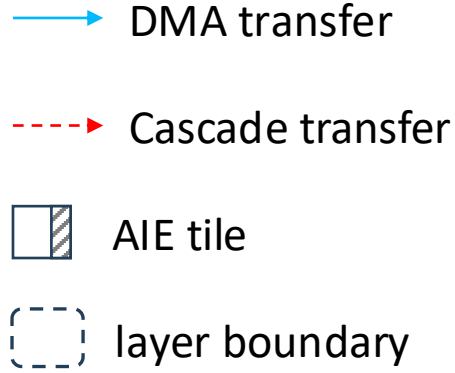




consistent shape and placement:

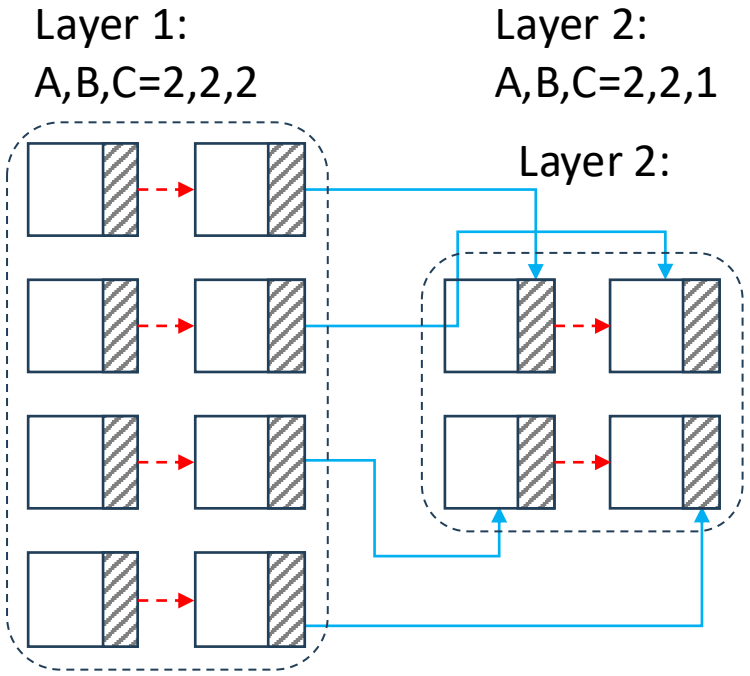
Use cascade





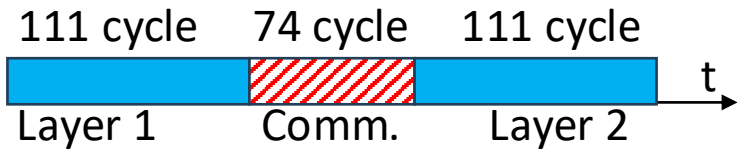
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



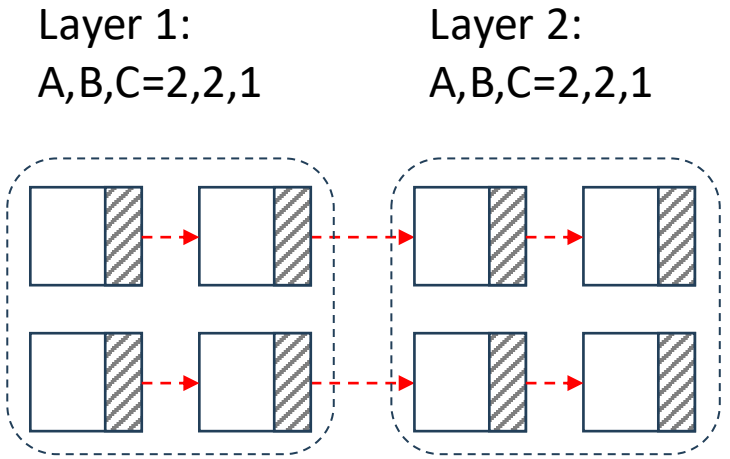
Inconsistent shape and placement:

Fallback to direct DMA



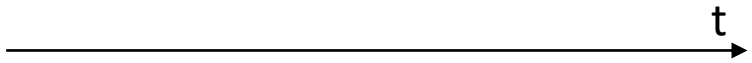
OK comp. latency

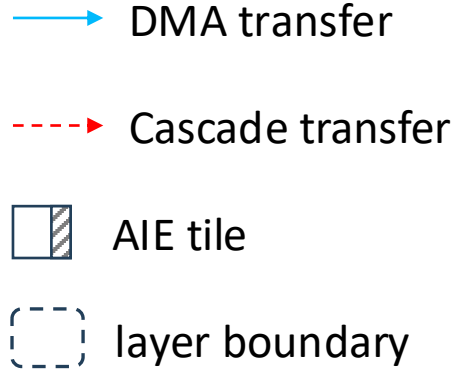
Large Comm. latency



consistent shape and placement:

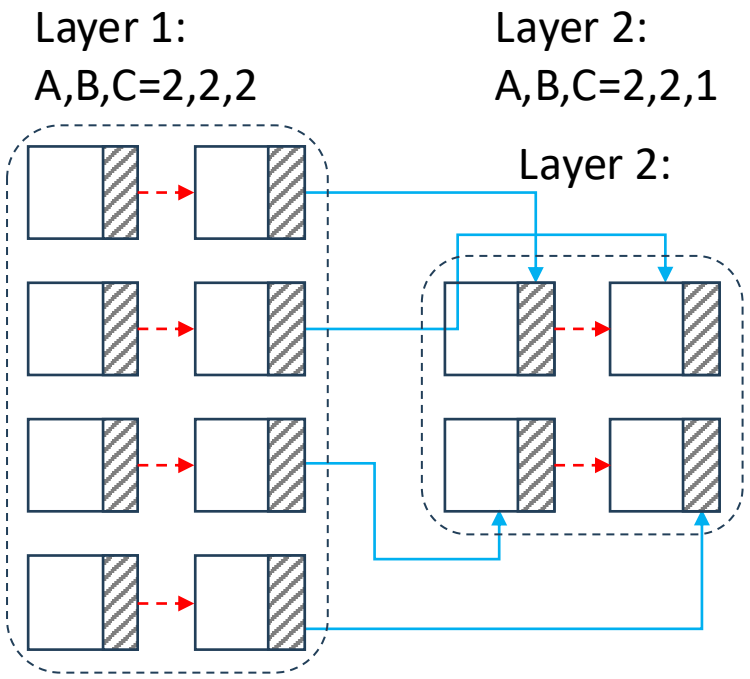
Use cascade





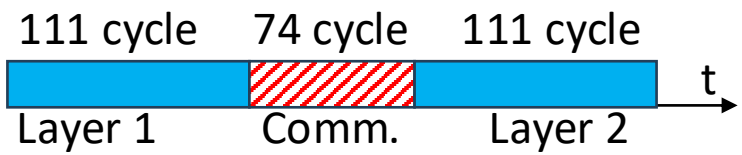
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



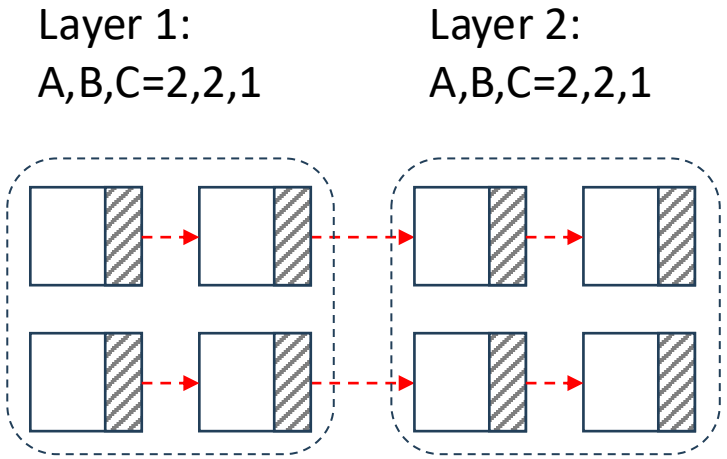
Inconsistent shape and placement:

Fallback to direct DMA



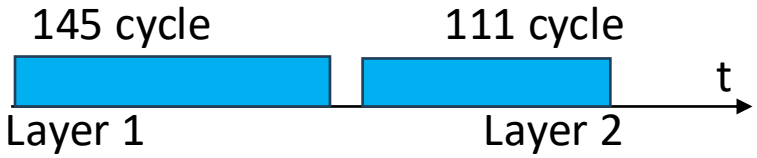
OK comp. latency

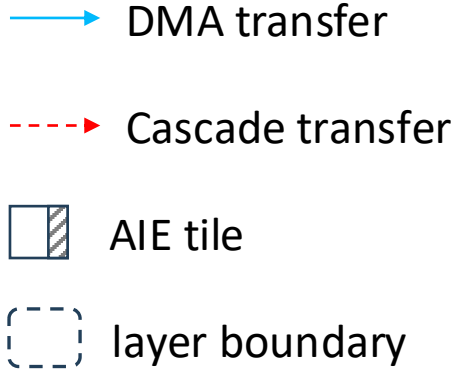
Large Comm. latency



consistent shape and placement:

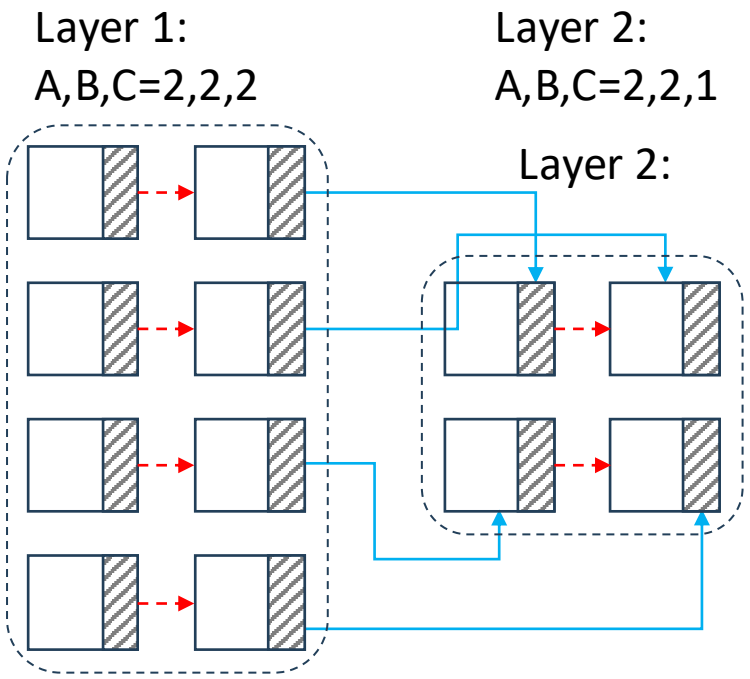
Use cascade





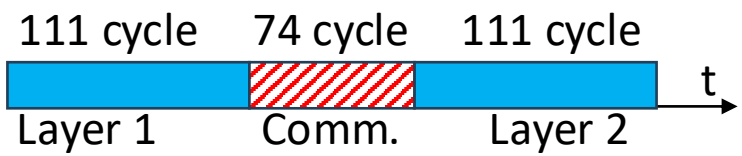
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



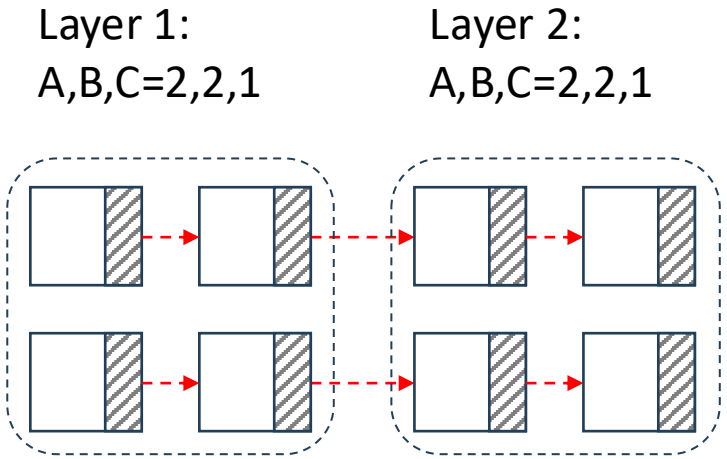
Inconsistent shape and placement:

Fallback to direct DMA



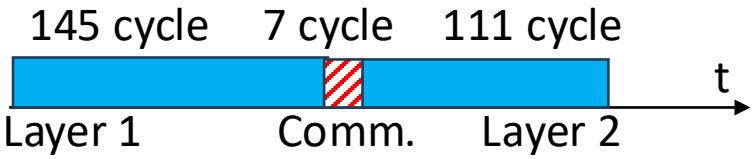
OK comp. latency

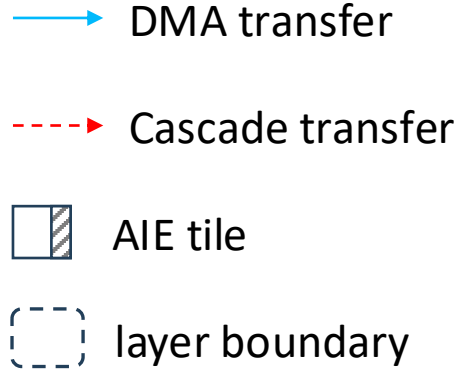
Large Comm. latency



consistent shape and placement:

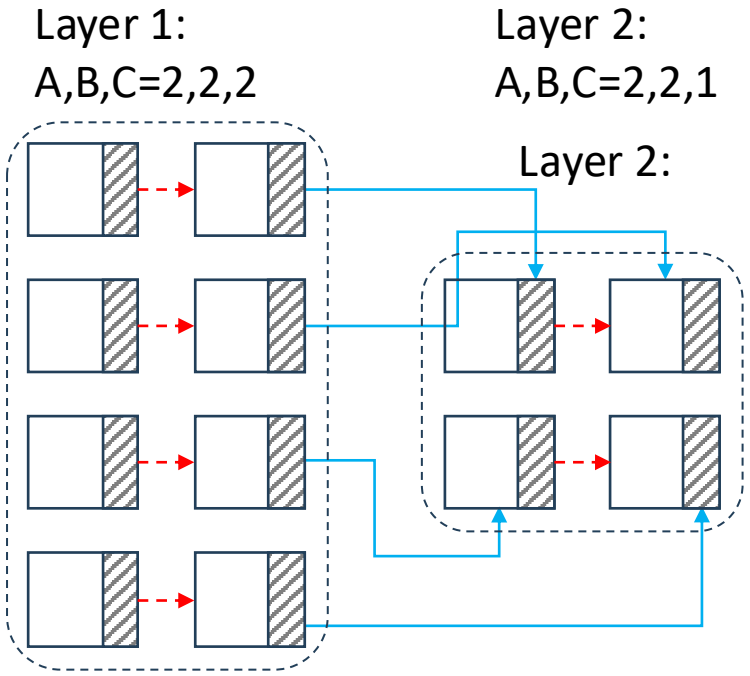
Use cascade





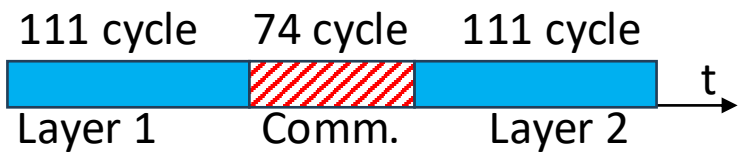
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



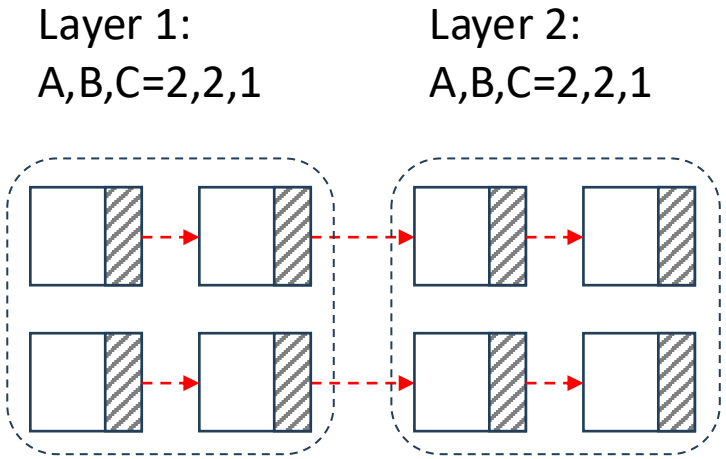
Inconsistent shape and placement:

Fallback to direct DMA



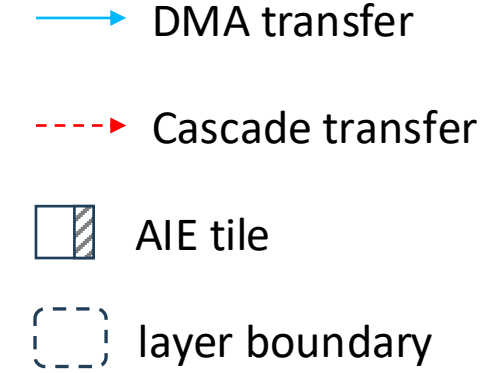
OK comp. latency

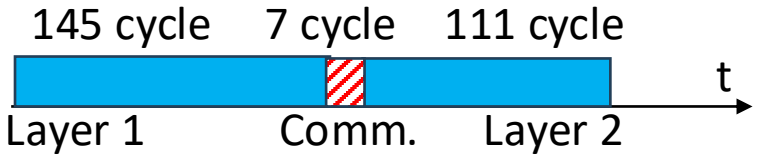
Large Comm. latency



consistent shape and placement:

Use cascade



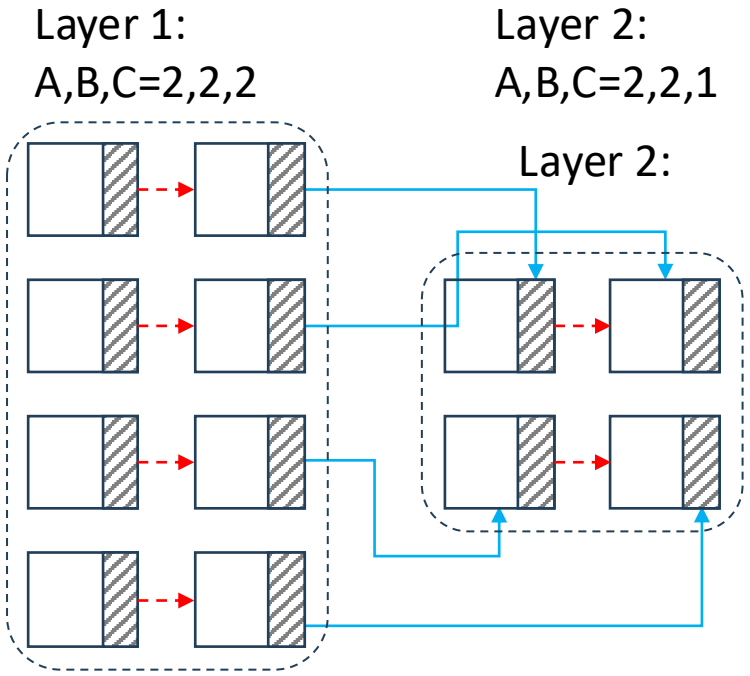


Higher comp. latency

Greatly reduced comm. latency!

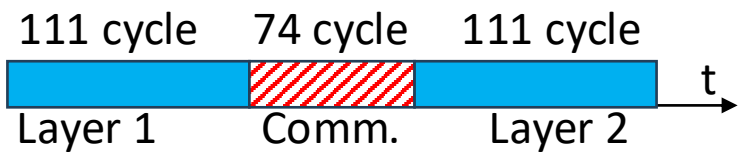
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



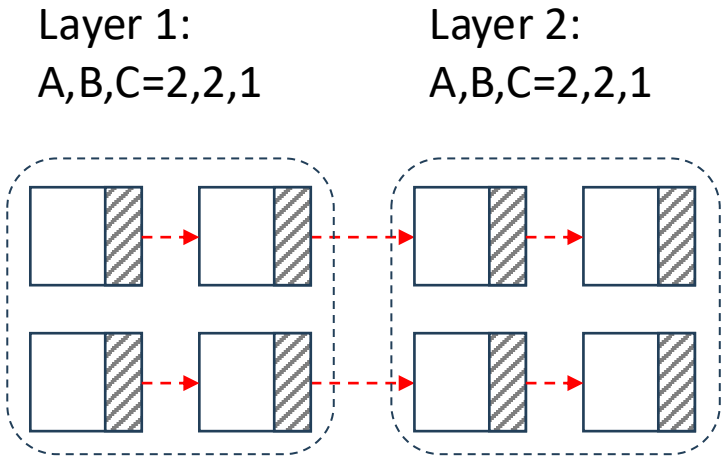
Inconsistent shape and placement:

Fallback to direct DMA



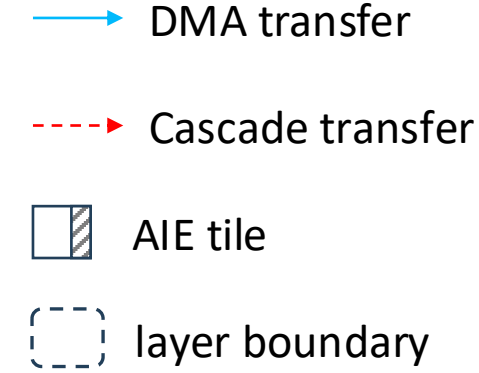
OK comp. latency

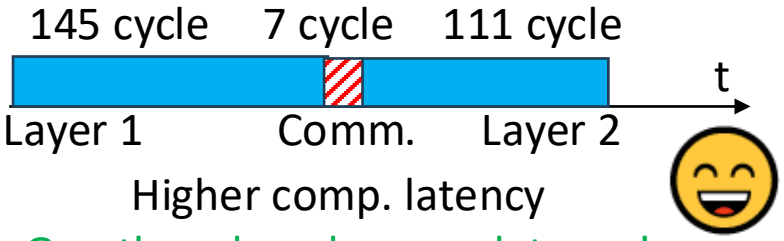
Large Comm. latency



consistent shape and placement:

Use cascade



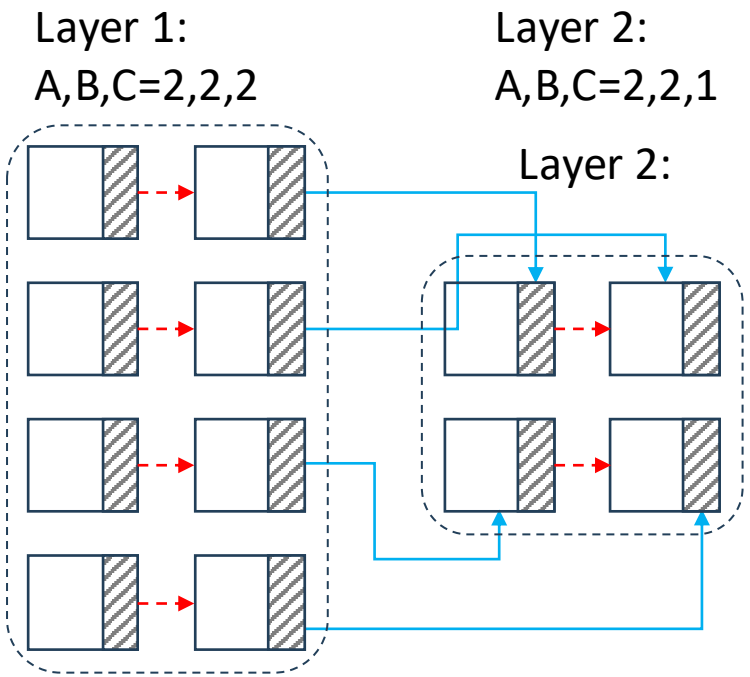


Higher comp. latency

Greatly reduced comm. latency!

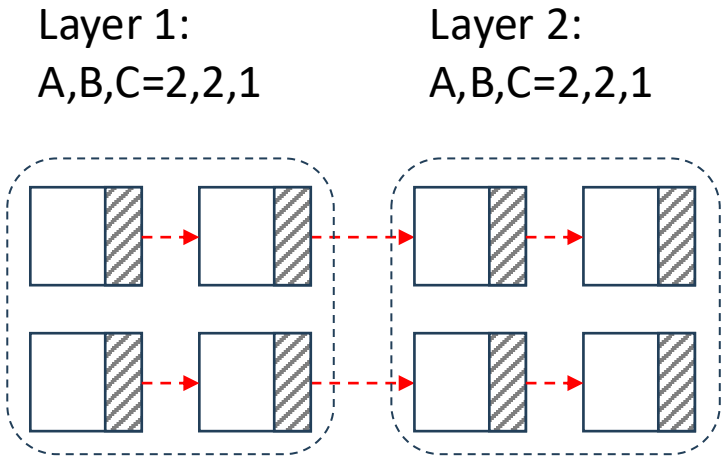
Computation-communication Tradeoff

The shape constrain could limit the parallelism…



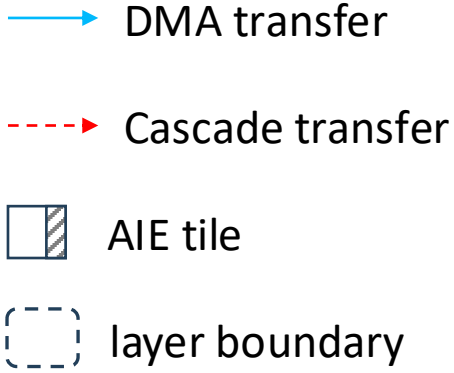
Inconsistent shape and placement:

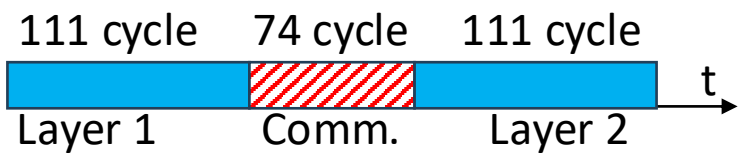
Fallback to direct DMA



consistent shape and placement:

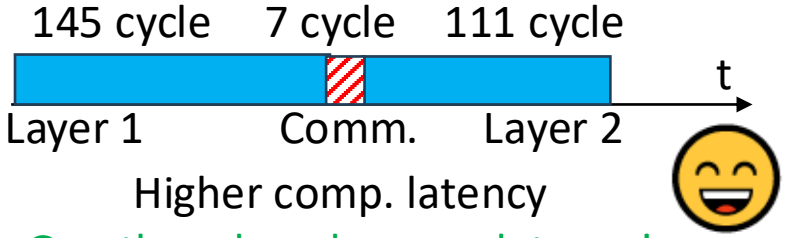
Use cascade





OK comp. latency

Large Comm. latency



Higher comp. latency

Greatly reduced comm. latency!

DSE:
Resource allocation
+
Placement

The Global Reduction Layer

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

One INT8 VMAC instr. → 4x8x8 MM kernel

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

1	1	1	1
0	0	0	0
…			

One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

b00	b01	..	b07
b10	…		
…			
			b77

1	1	1	1
0	0	0	0
…			

One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$

b00	b01	..	b07
b10	…		
…			
			b77

1	1	1	1
0	0	0	0
…			

Σb_{i0}	Σb_{i1}	…	Σb_{i7}
		0	

One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

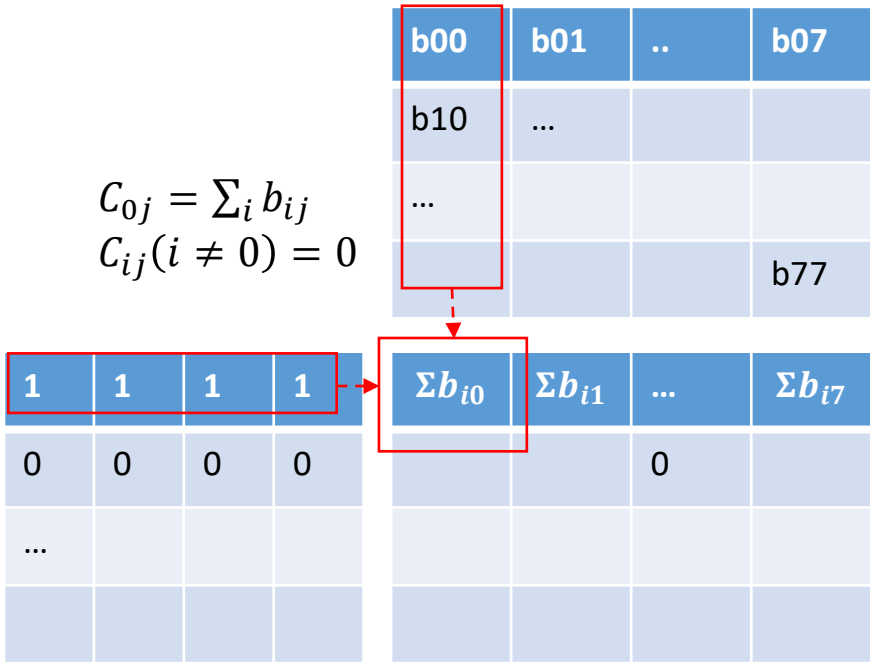
RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

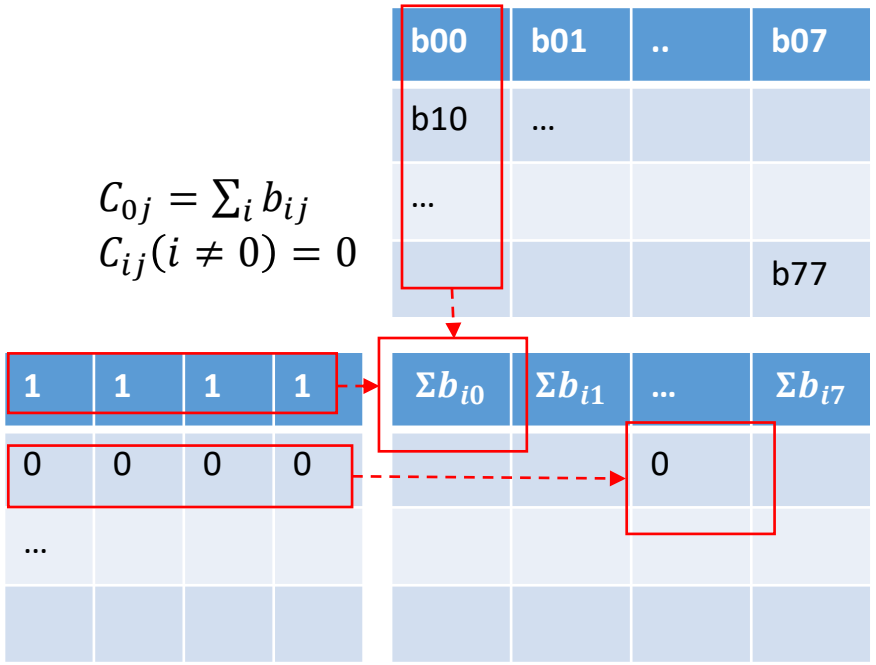
RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

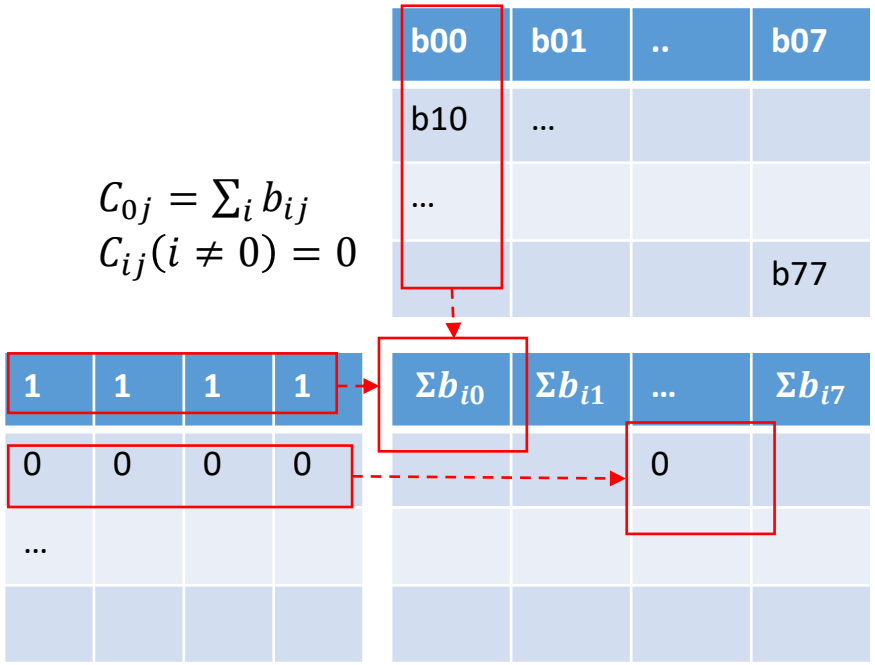
RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

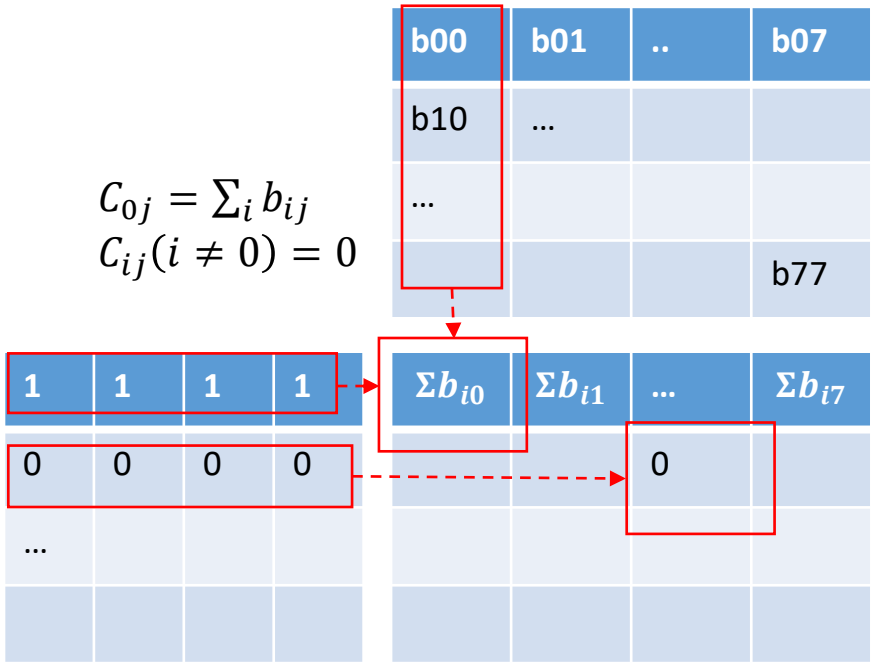
Output: the first row(first 8 elements) is the accumulated results

No layout change: **7x reduction** in #instr

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

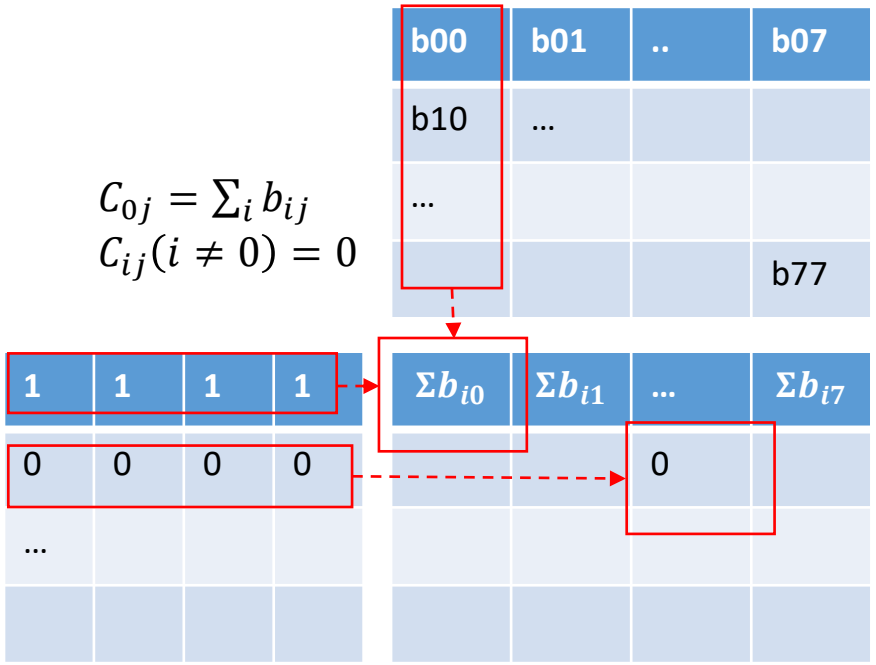
No layout change: **7x reduction** in #instr

- Reduction across multiple AIEs

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

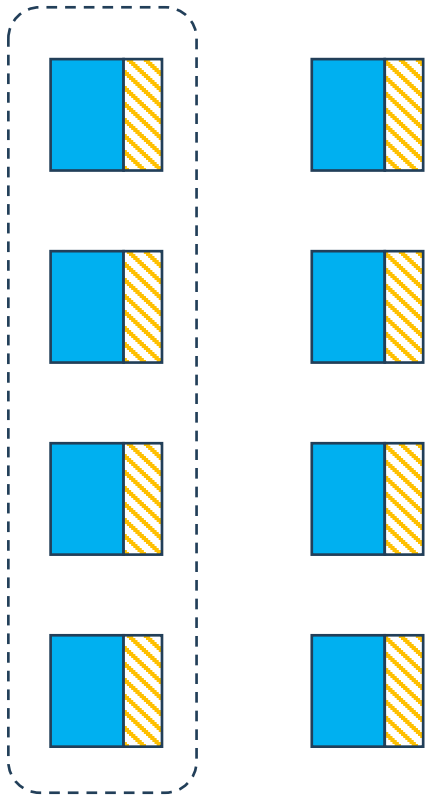
LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

No layout change: **7x reduction** in #instr

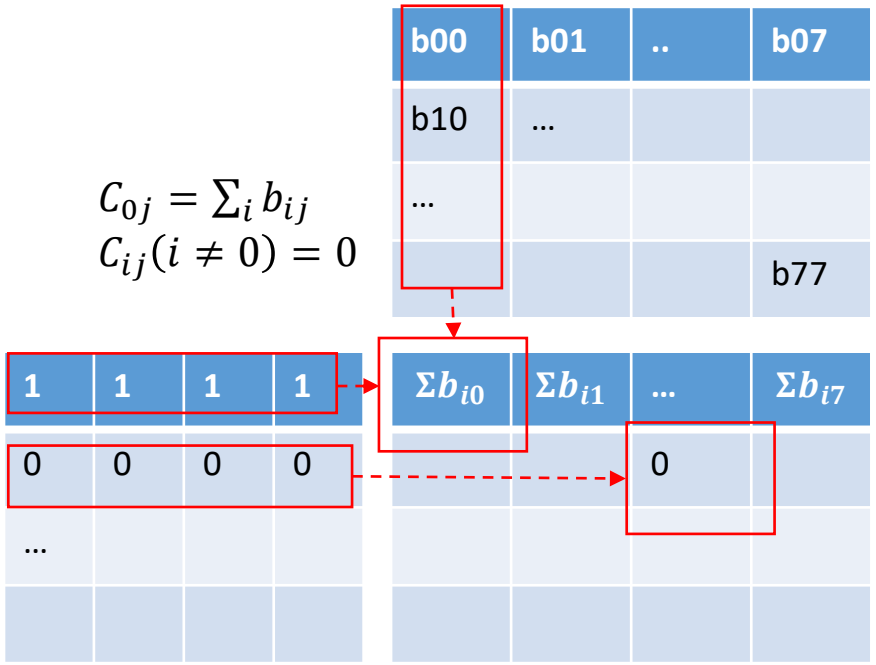
- Reduction across multiple AIEs



The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

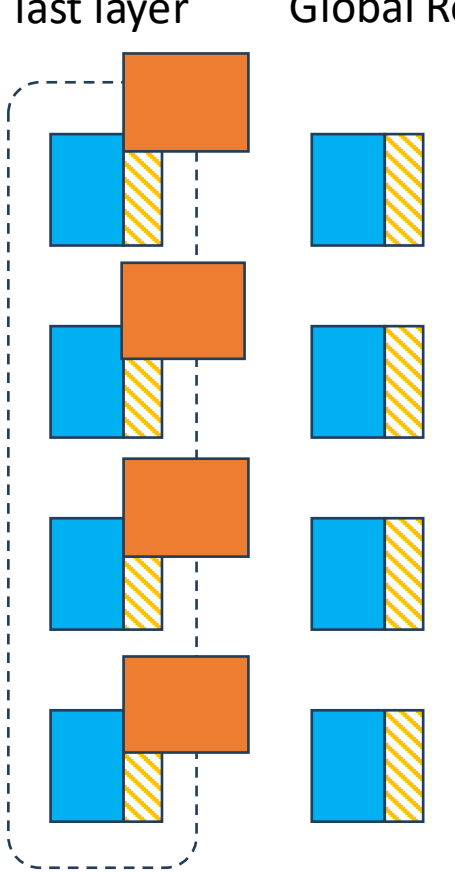
LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

No layout change: **7x reduction** in #instr

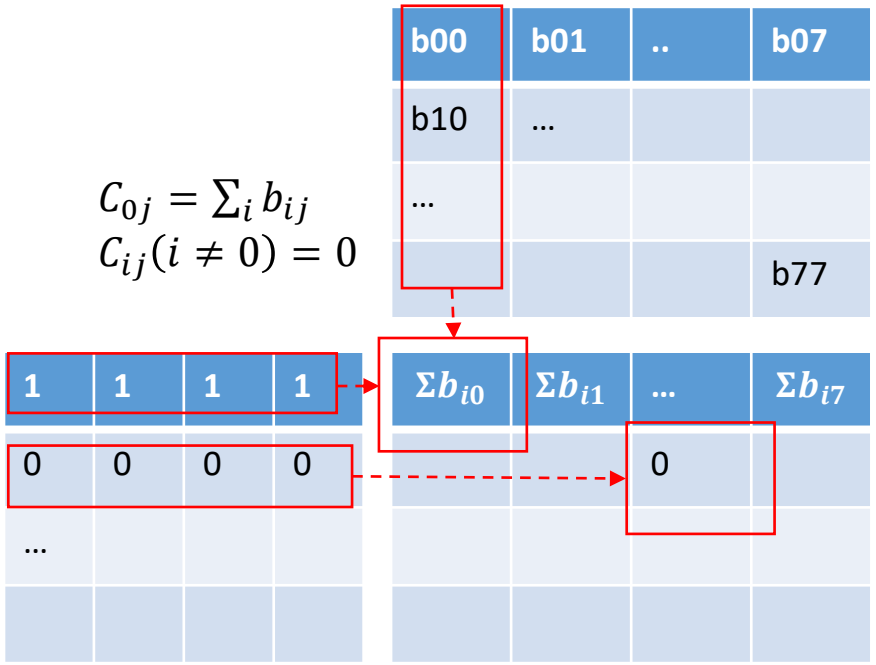
- Reduction across multiple AIEs



The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

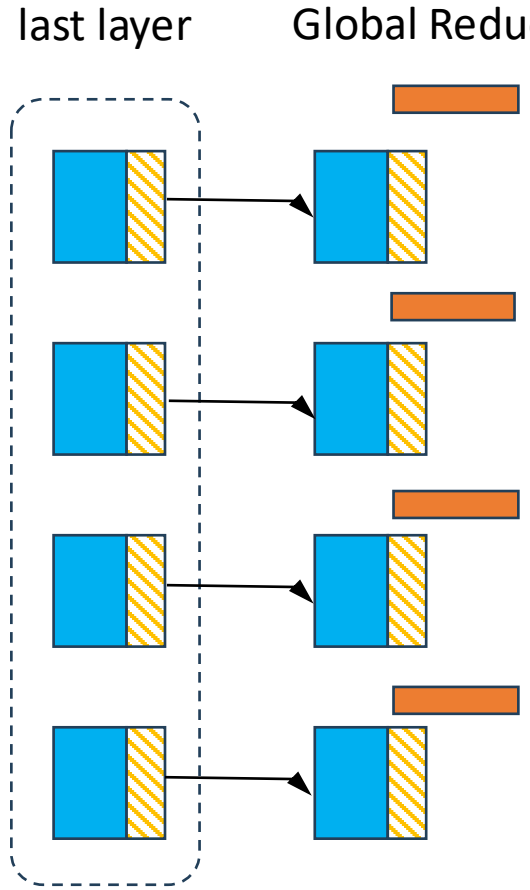
LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

No layout change: **7x reduction** in #instr

- Reduction across multiple AIEs

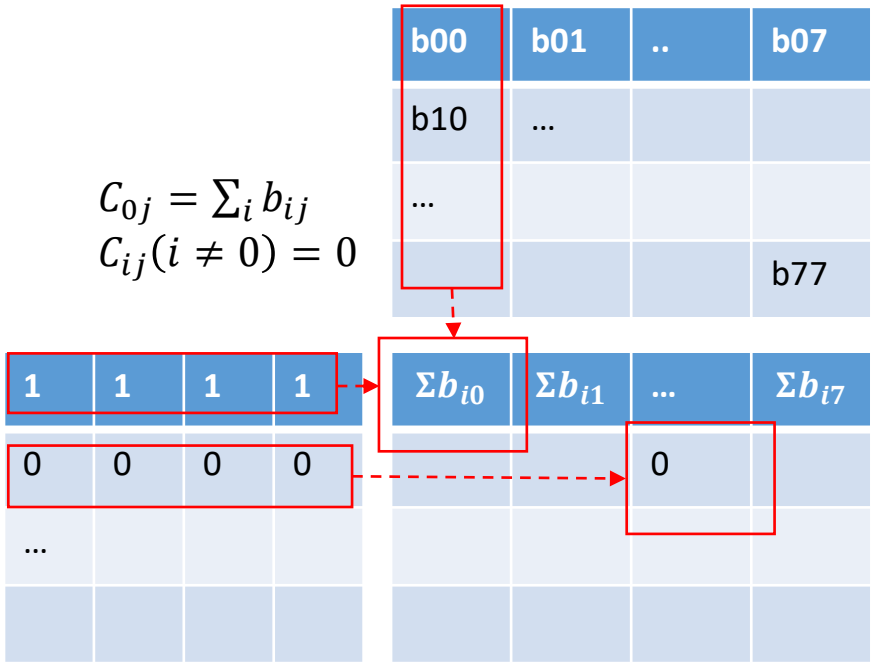


Input: shared mem

The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

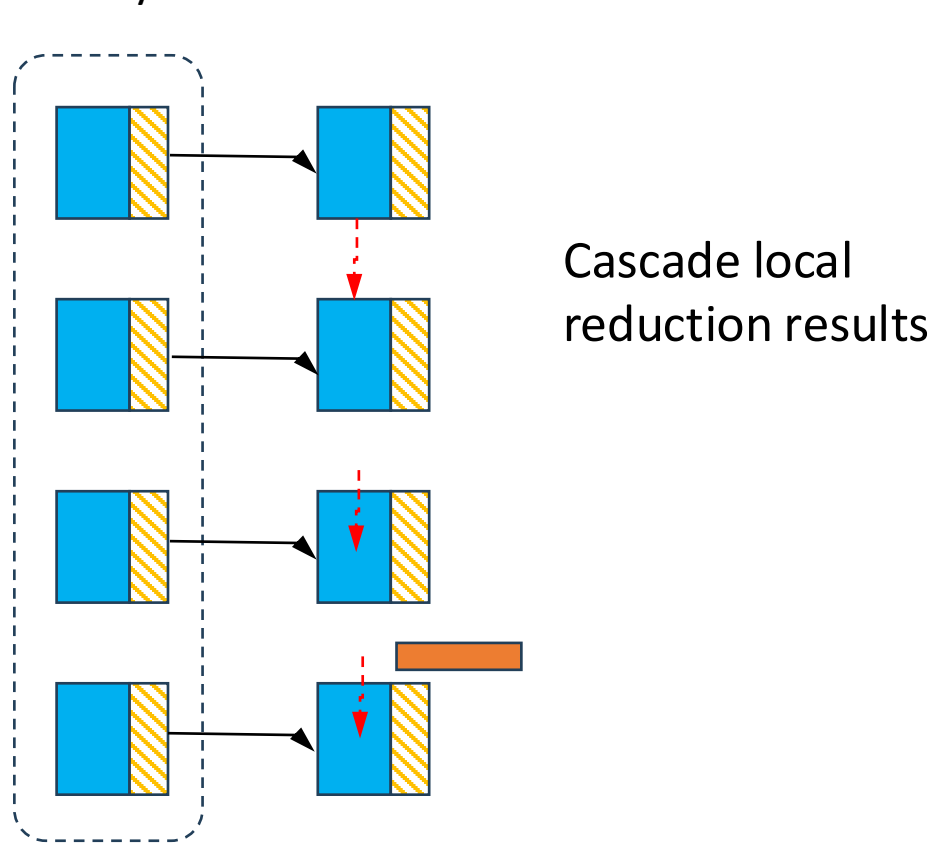
LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

Output: the first row(first 8 elements) is the accumulated results

No layout change: **7x reduction** in #instr

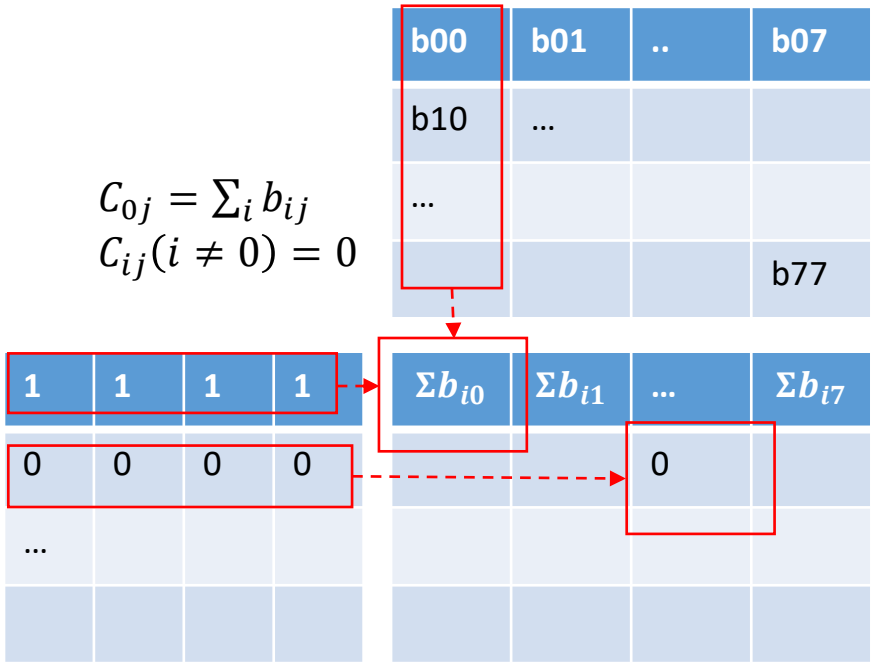
- Reduction across multiple AIEs



The Global Reduction Layer

- Single AIE: using MAC operations, such reductions can be done in 1 VMAC instruction

$C_{0j} = \sum_i b_{ij}$
$C_{ij}(i \neq 0) = 0$



One INT8 VMAC instr. → 4x8x8 MM kernel

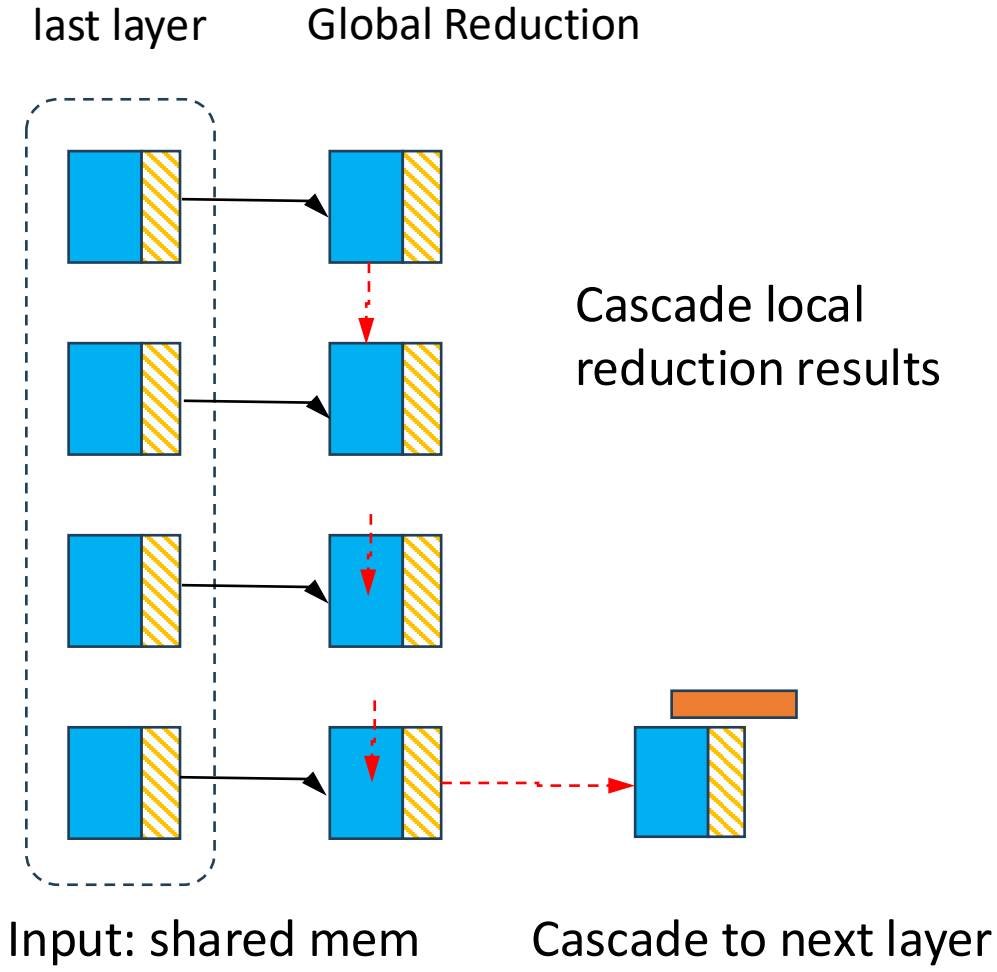
LHS: static matrix with first line = 1, other = 0

RHS: Two input blocks from last layer

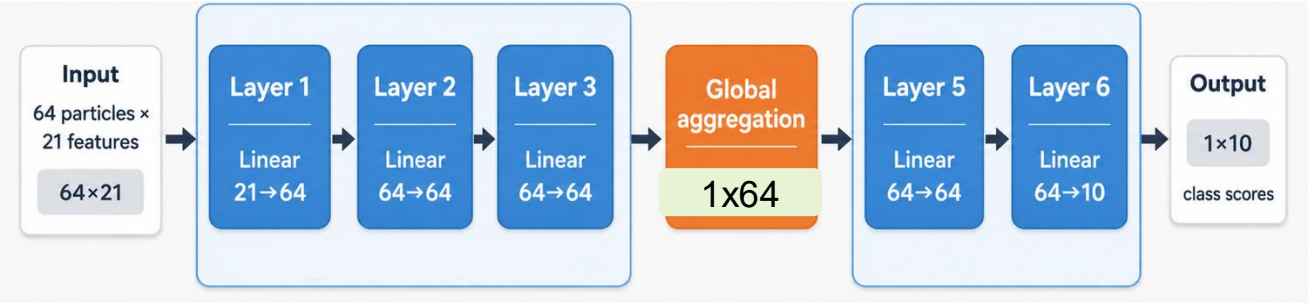
Output: the first row(first 8 elements) is the accumulated results

No layout change: **7x reduction** in #instr

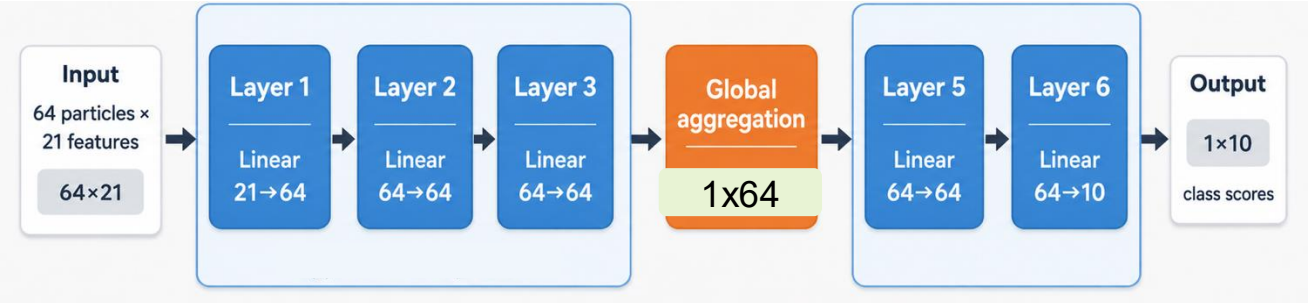
- Reduction across multiple AIEs

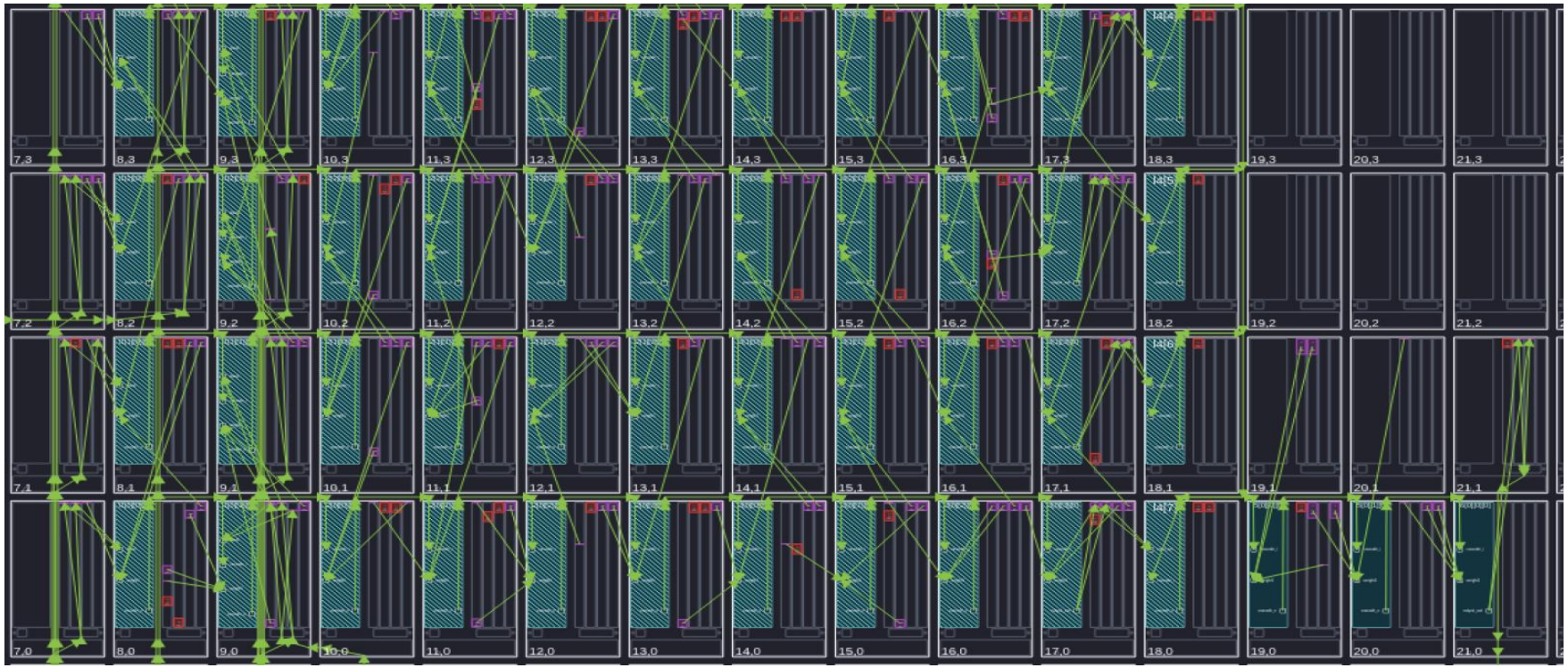


Case Study: The DeepSets Model

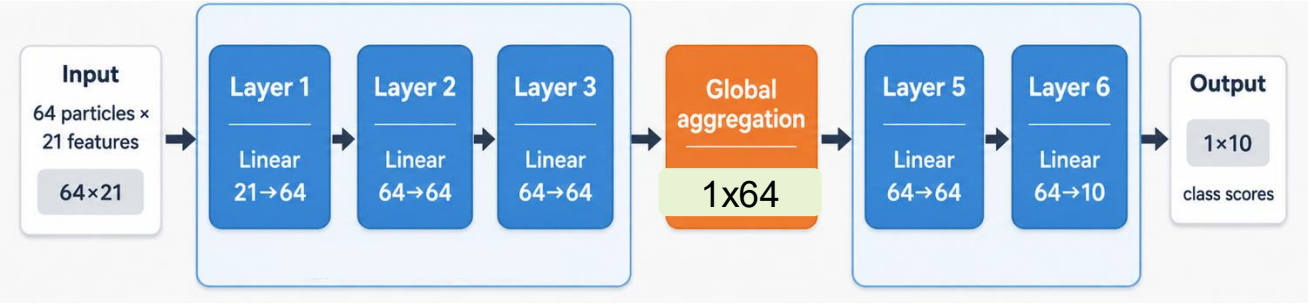


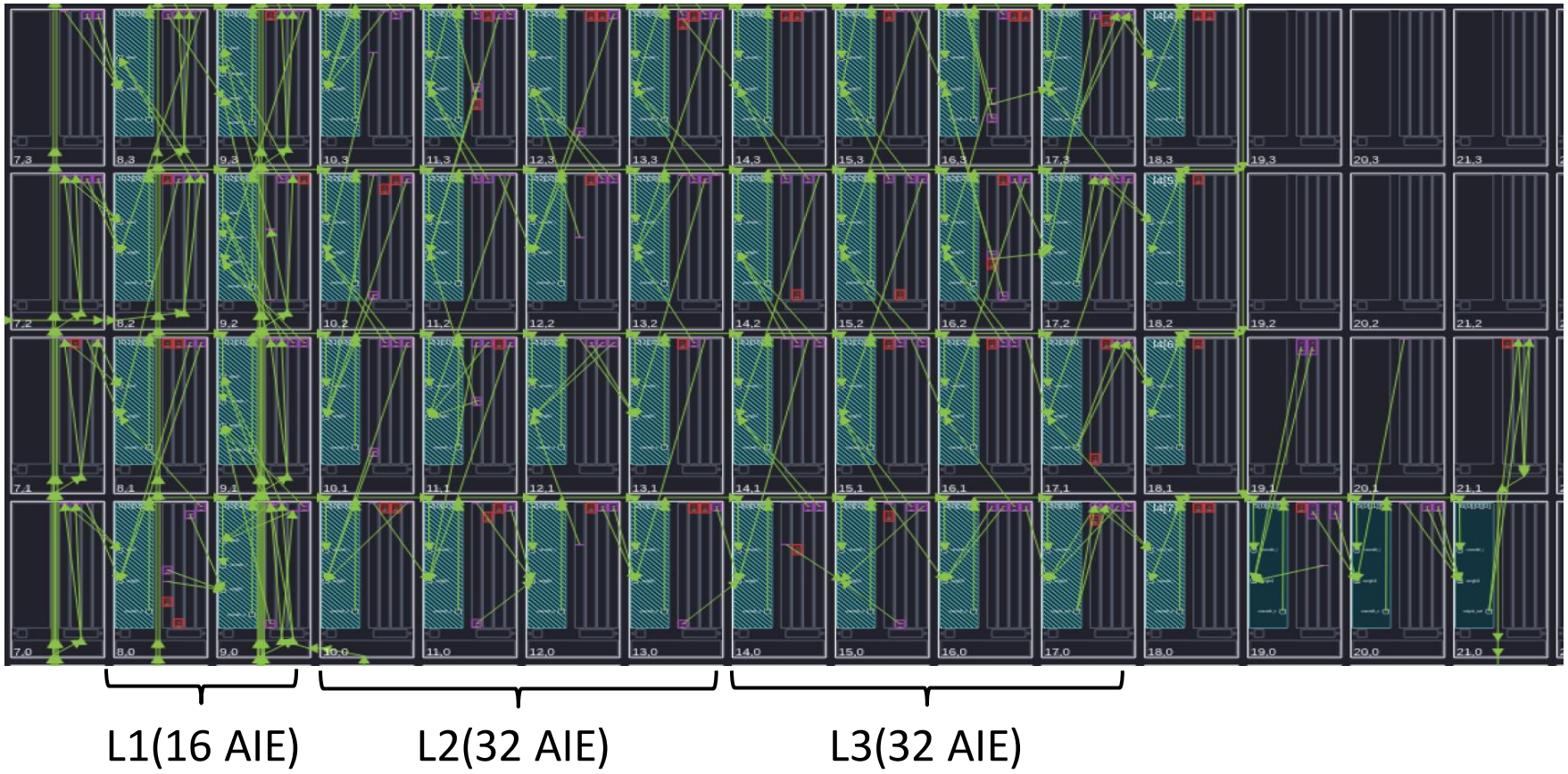
Case Study: The DeepSets Model



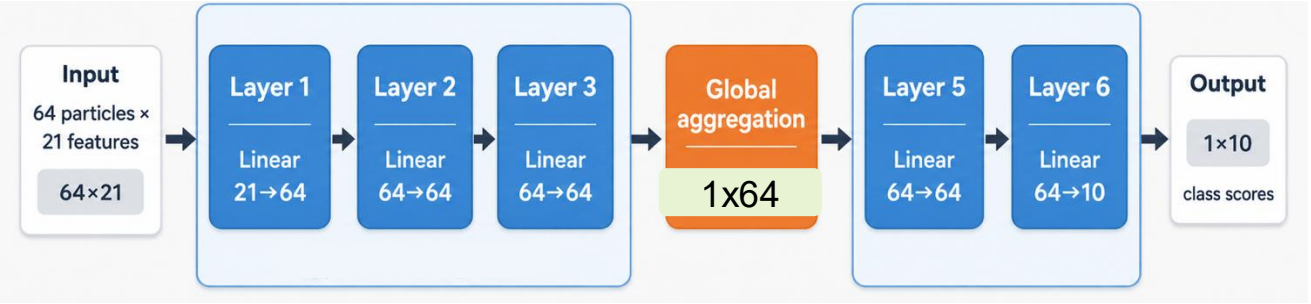


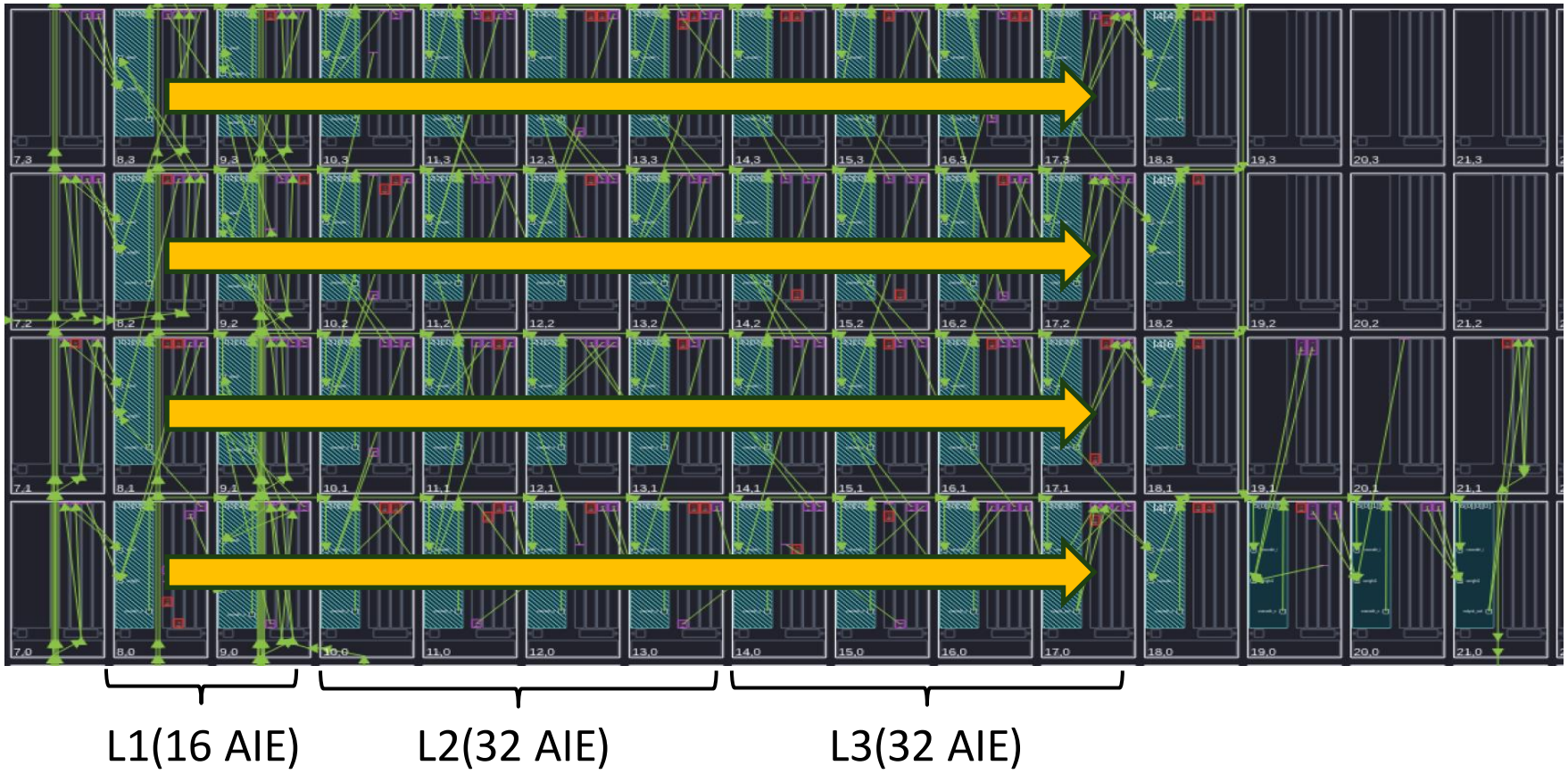
Case Study: The DeepSets Model



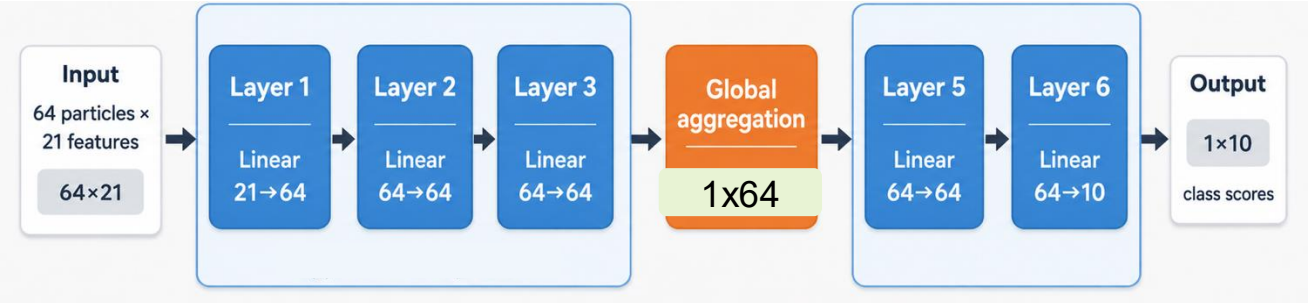


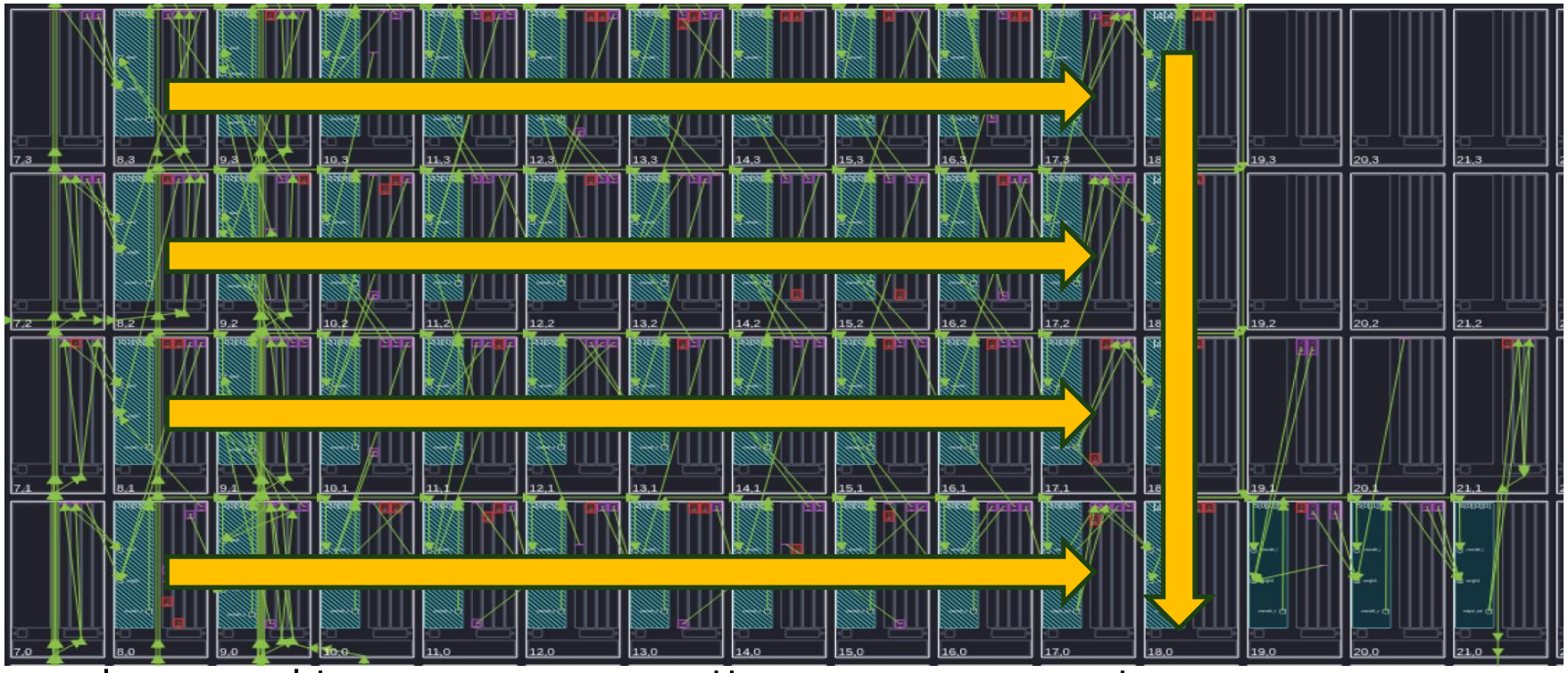
Case Study: The DeepSets Model





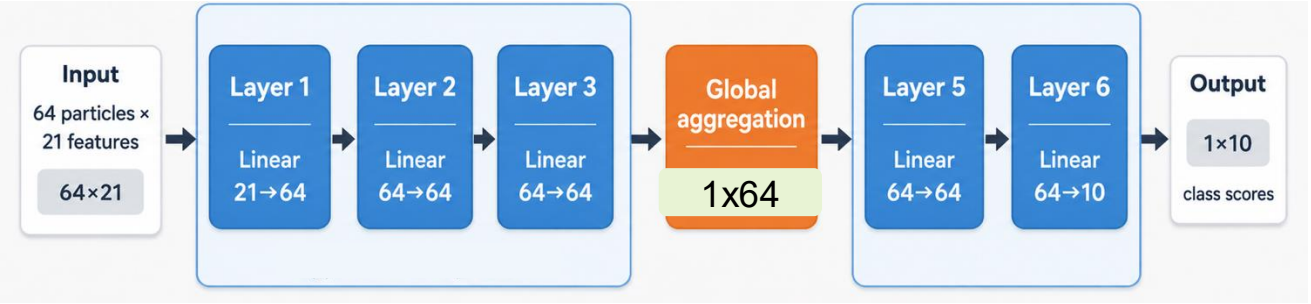
Case Study: The DeepSets Model

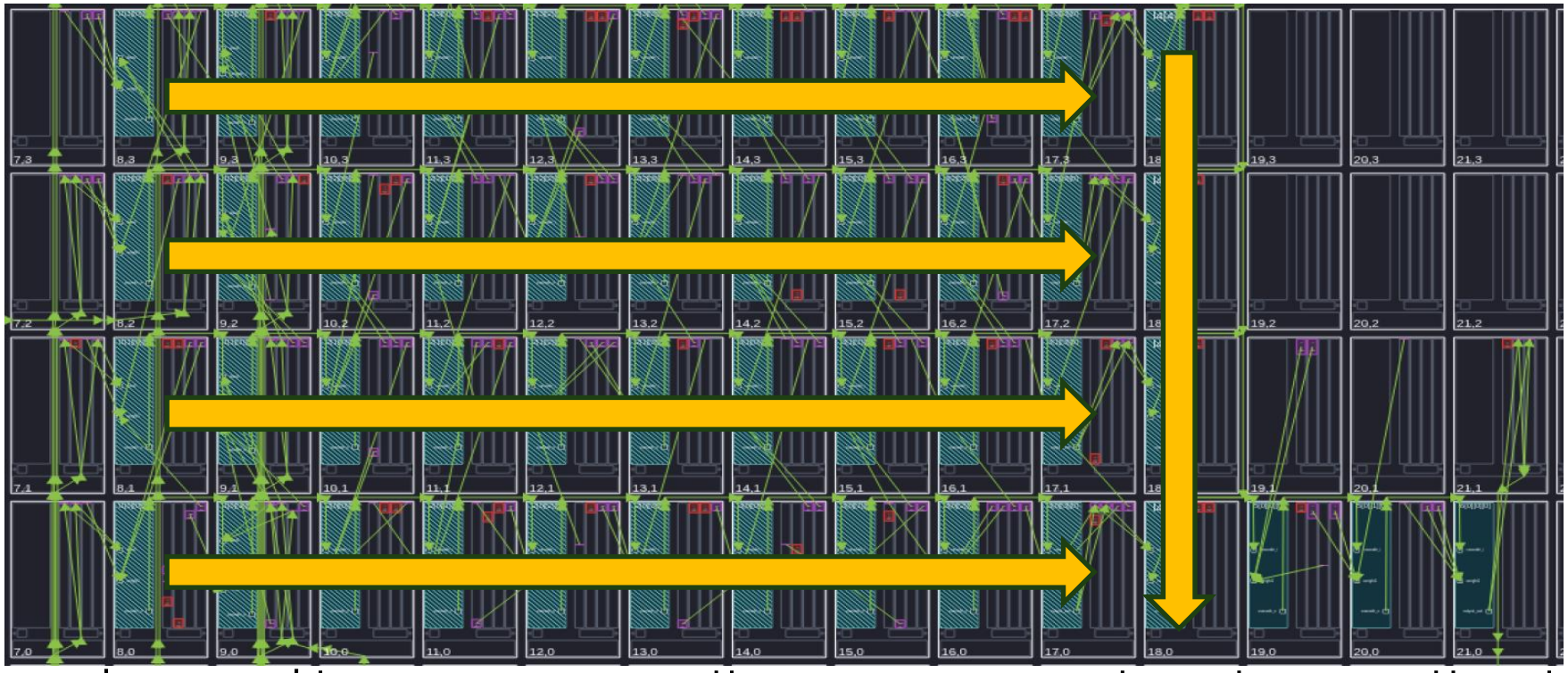




L1(16 AIE) L2(32 AIE) L3(32 AIE) GA (8AIE)

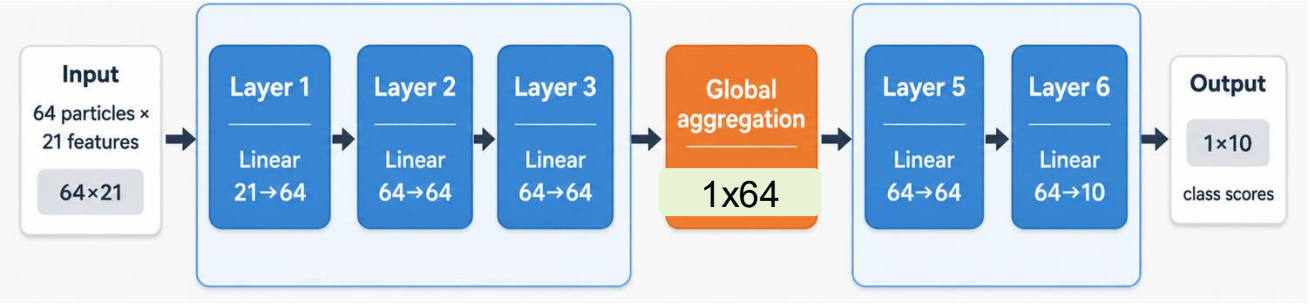
Case Study: The DeepSets Model

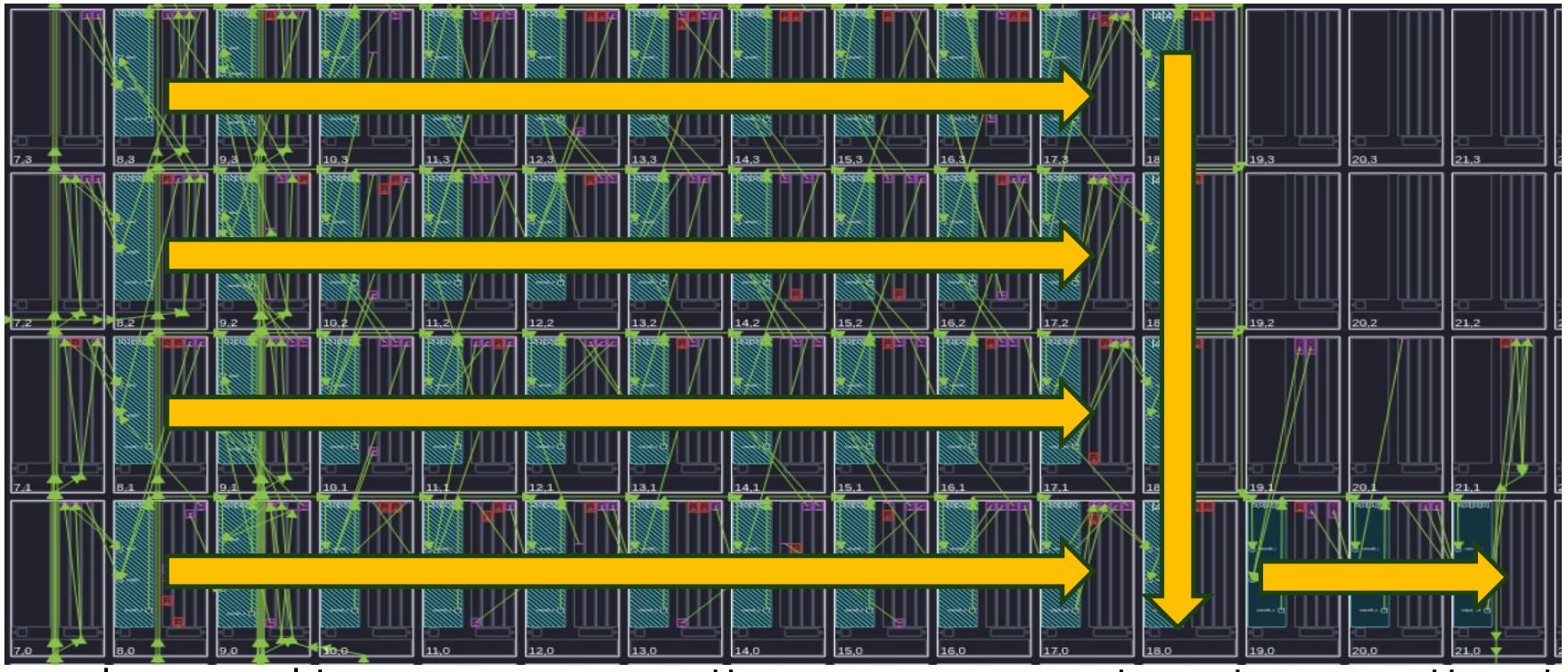




L1(16 AIE) L2(32 AIE) L3(32 AIE) GA (8AIE) L5(2 AIE) L6(2 AIE)

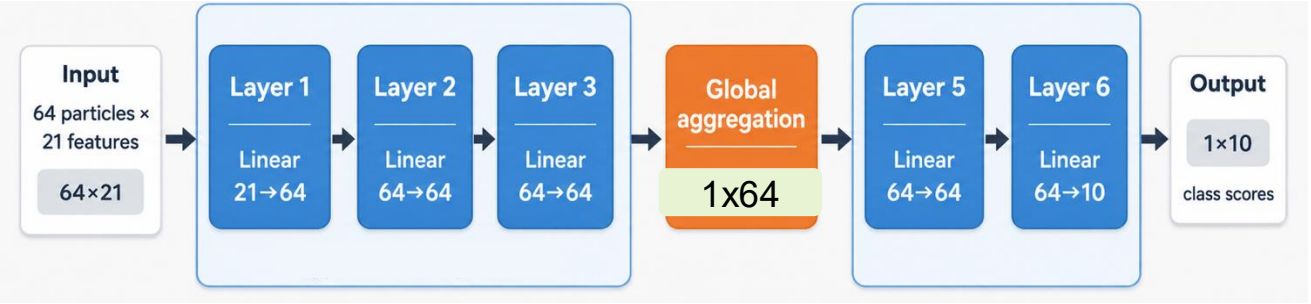
Case Study: The DeepSets Model

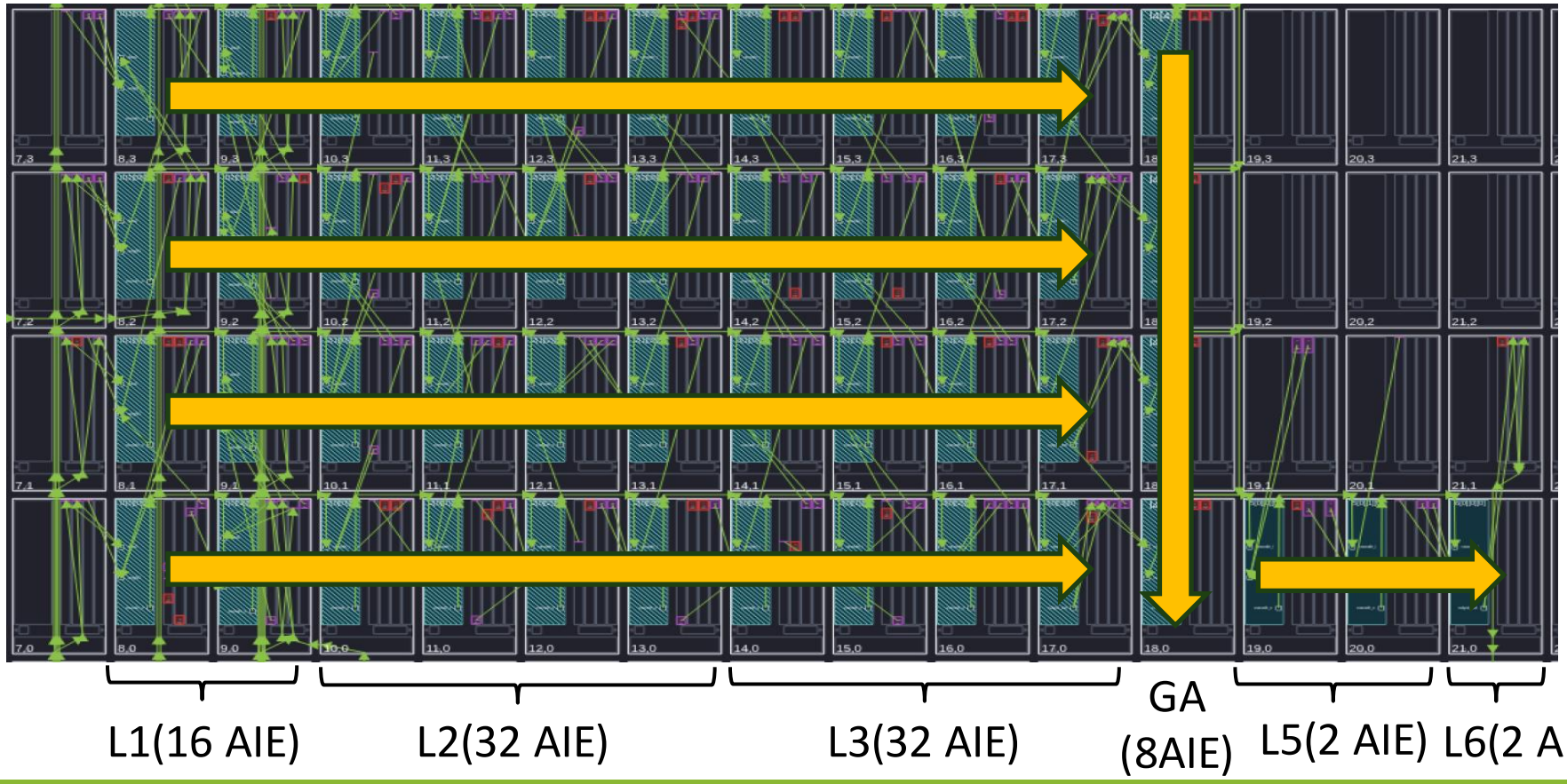




L1(16 AIE) L2(32 AIE) L3(32 AIE) GA (8AIE) L5(2 AIE) L6(2 AIE)

Case Study: The DeepSets Model





Before: 5.8 us

→

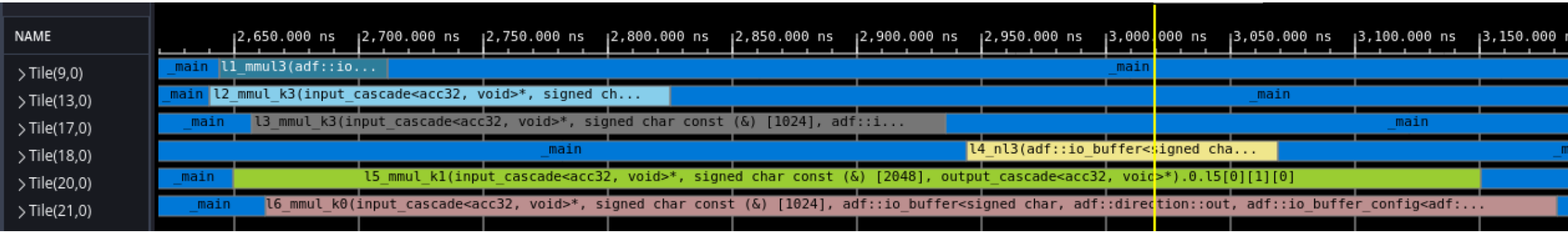
After: 0.538 us

10x latency reduction

Meet 1-us deadline

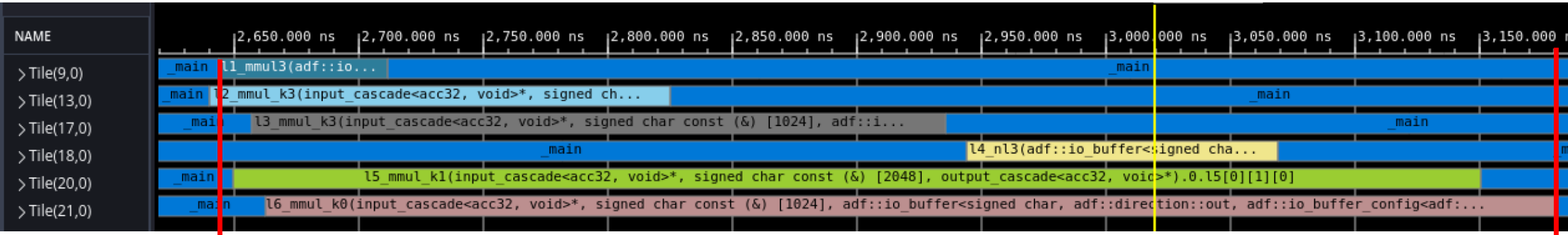
Case Study: The Deepsets Model

AIE simulator trace for tiles in each layer, single inference



Case Study: The Deepsets Model

AIE simulator trace for tiles in each layer, single inference



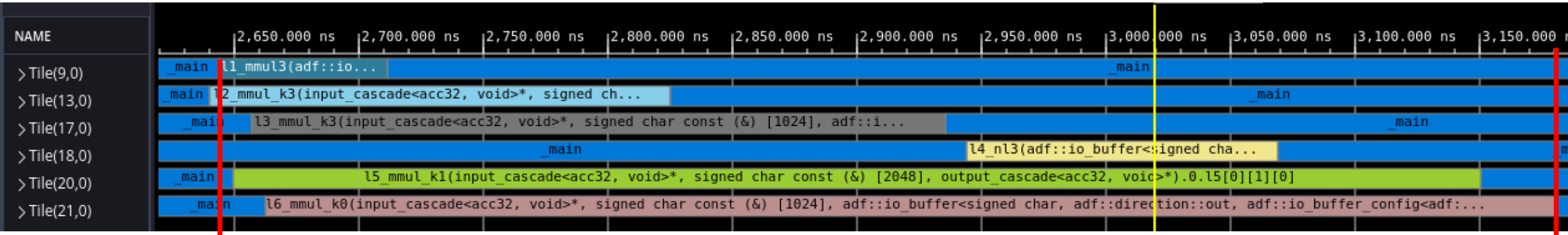
Layer1 start:
T=2643 ns

0.538 us computation latency

Layer6 end:
T=3181 ns

Case Study: The Deepsets Model

AIE simulator trace for tiles in each layer, single inference



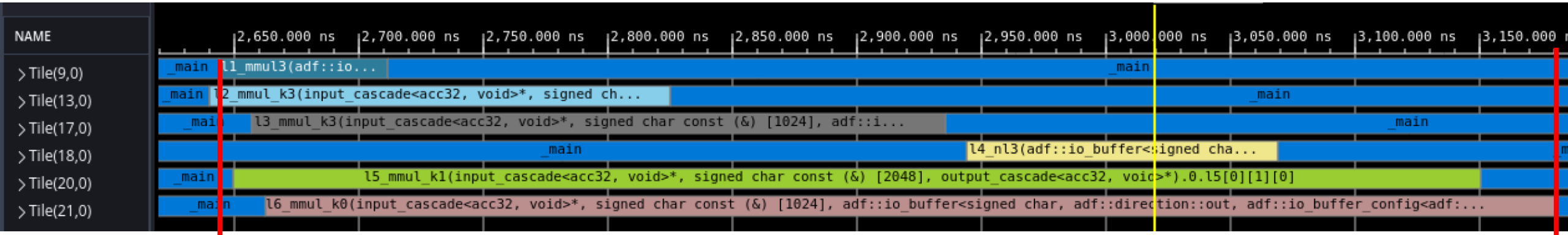
Layer1 start:
T=2643 ns

0.538 us computation latency
0.88 us computation with PL latency

Layer6 end:
T=3181 ns

Case Study: The Deepsets Model

AIE simulator trace for tiles in each layer, single inference

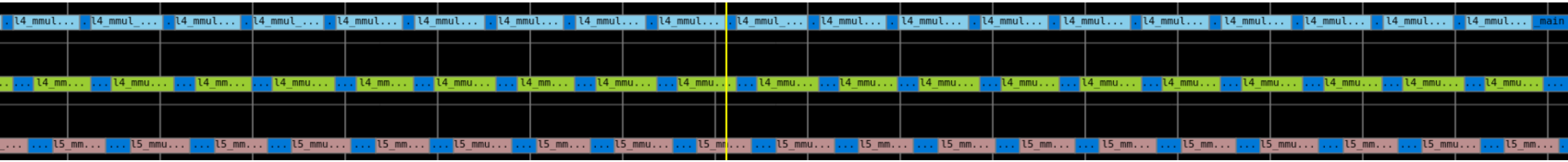


Layer1 start:
T=2643 ns

0.538 us computation latency
0.88 us computation with PL latency

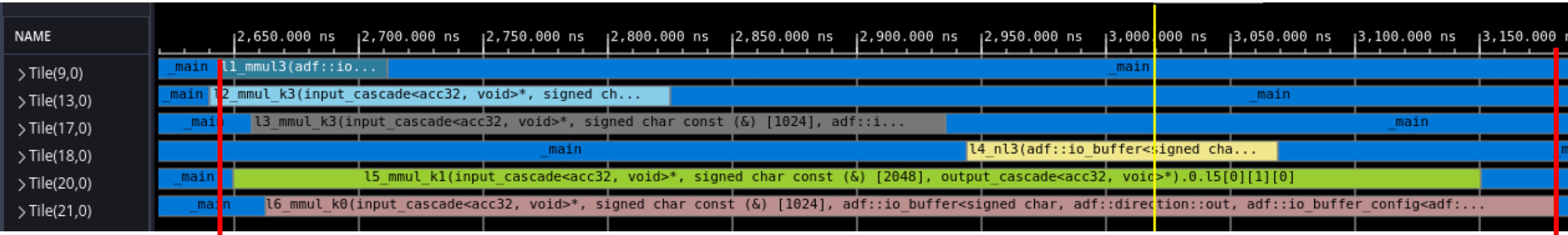
Layer6 end:
T=3181 ns

AIE simulator trace for last layer, pipelined inference



Case Study: The Deepsets Model

AIE simulator trace for tiles in each layer, single inference

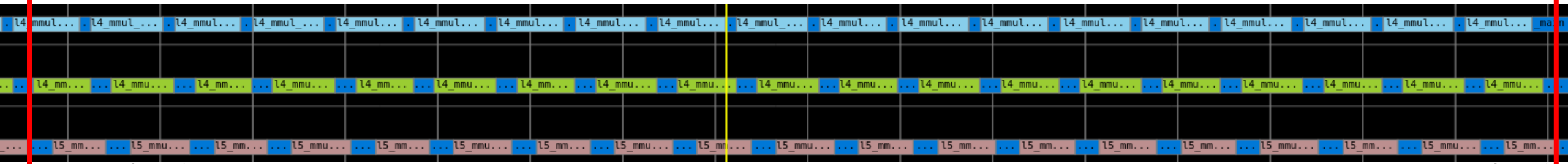


Layer1 start:
T=2643 ns

0.538 us computation latency
0.88 us computation with PL latency

Layer6 end:
T=3181 ns

AIE simulator trace for last layer, pipelined inference



Frame 0 finish:
4481 ns

19 frame/3138ns → 6.05M FPS, 165 ns/frame

Frame 19 finish:
7619 ns

Experiments

Synthetic MLPs: sequence of square MM, INT8, dense

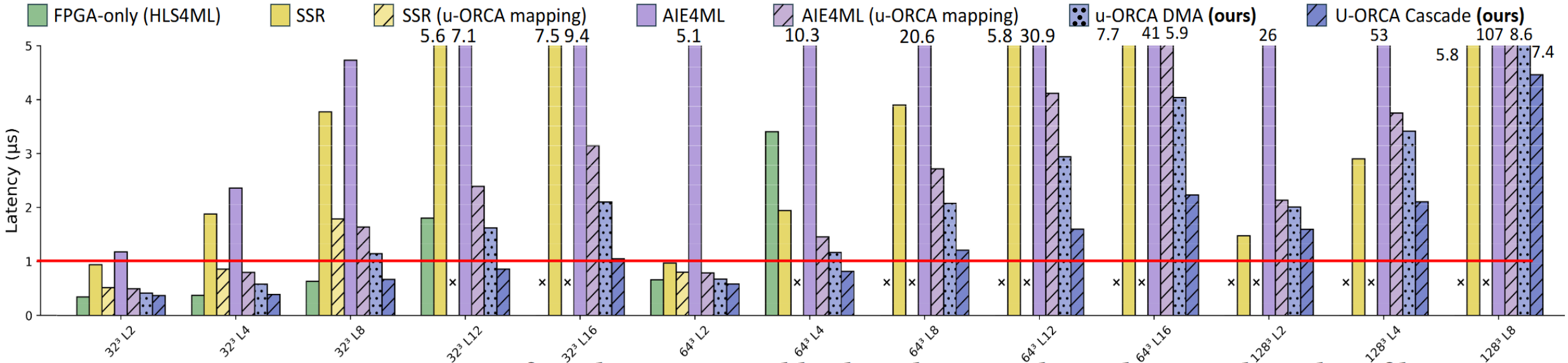


Figure 10: Latency comparison of synthetic MLP workloads with various layer shapes and number of layers.

Experiments

Synthetic MLPs: sequence of square MM, INT8, dense

AMD VEK280, Vitis 2024.1

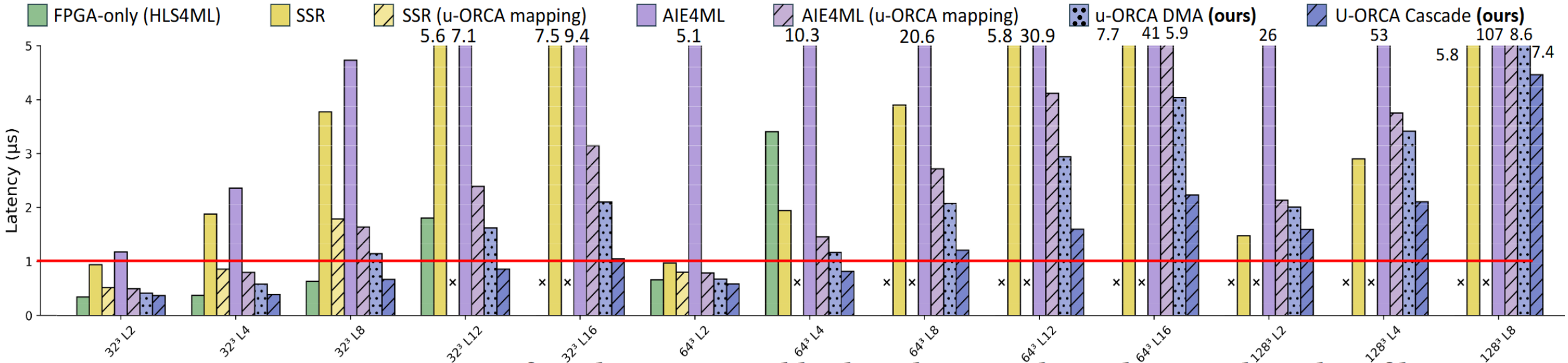


Figure 10: Latency comparison of synthetic MLP workloads with various layer shapes and number of layers.

Experiments

Synthetic MLPs: sequence of square MM, INT8, dense

AMD VEK280, Vitis 2024.1

- FPGA baseline: Excel at small workloads, resource bounded for large ones

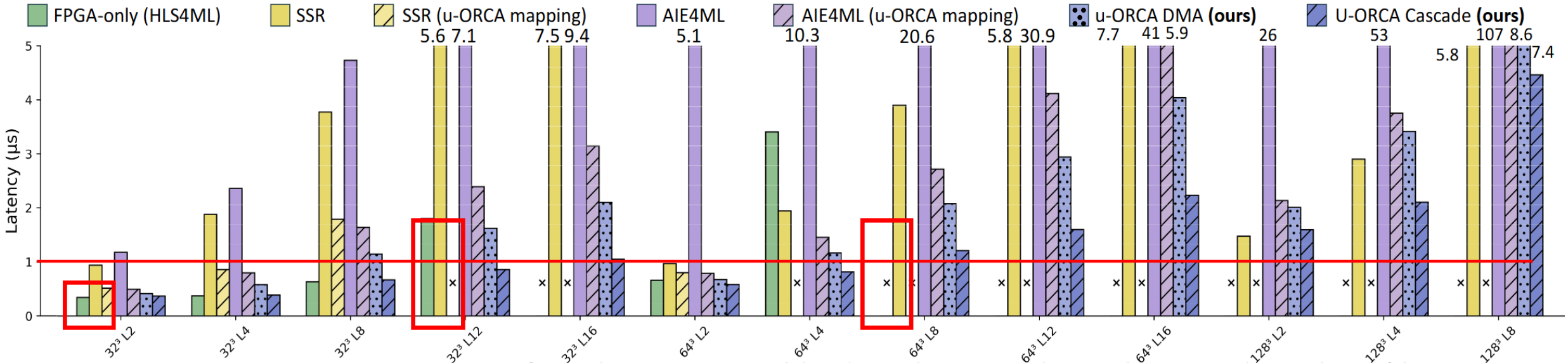


Figure 10: Latency comparison of synthetic MLP workloads with various layer shapes and number of layers.

Experiments

Synthetic MLPs: sequence of square MM, INT8, dense

AMD VEK280, Vitis 2024.1

- FPGA baseline: Excel at small workloads, resource bounded for large ones
- AIE baselines: Better flexibility at larger scales, cannot meet 1-us budget
- u-ORCA(ours): optimized on-chip communication, can handle 12 32x32x32 layers or 6 64x64x64 layers

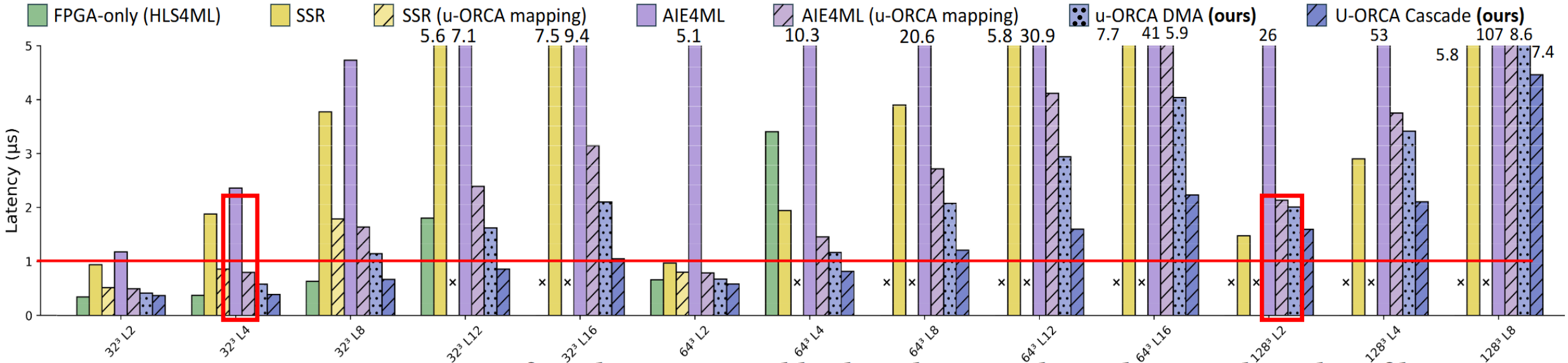


Figure 10: Latency comparison of synthetic MLP workloads with various layer shapes and number of layers.

Experiments

Realistic workloads: MLP and Deepset variants

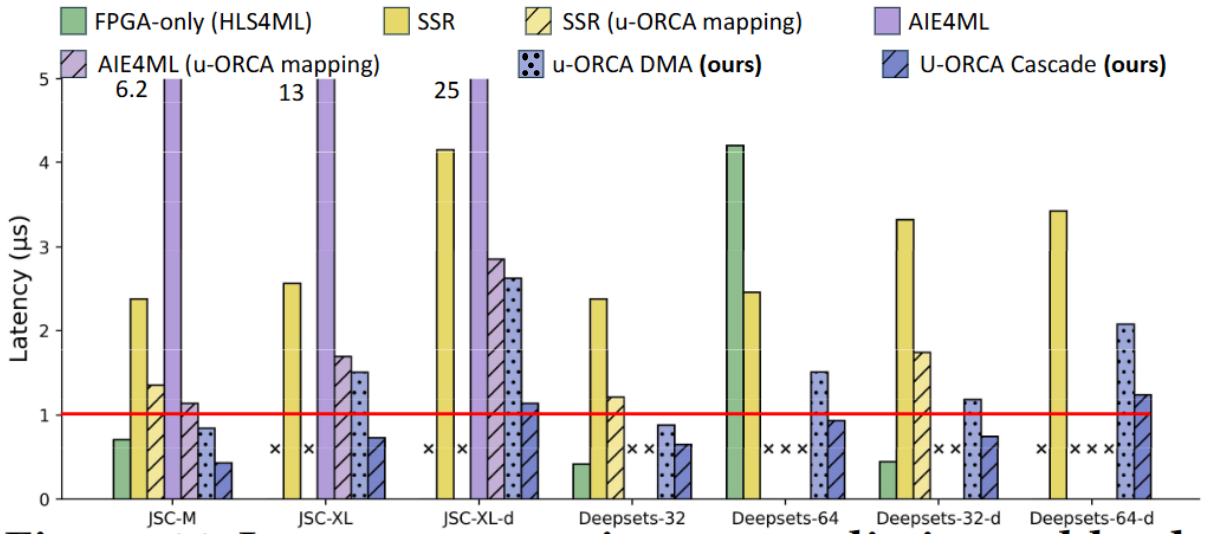


Figure 11: Latency comparison on realistic workloads

Experiments

Realistic workloads: MLP and Deepset variants

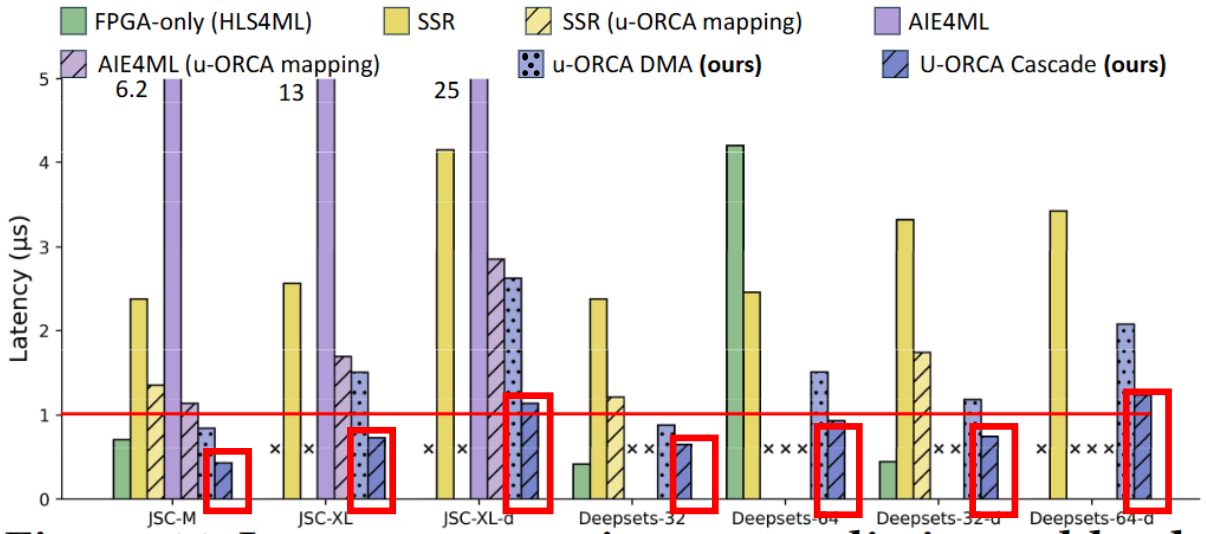


Figure 11: Latency comparison on realistic workloads

- u-ORCA excels at us-scale DNN inference!

Experiments

Realistic workloads: MLP and Deepset variants

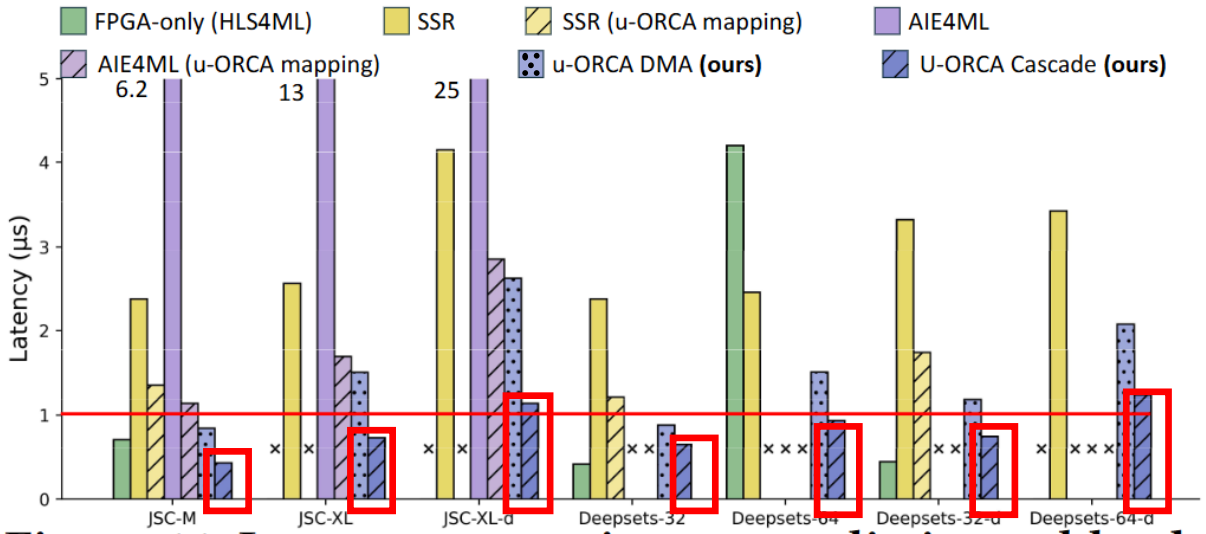


Figure 11: Latency comparison on realistic workloads

Latency of different global aggregation layers

Input	#AIE	Baseline	Ours	Speedup
32 × 32	4	373	66	5.65×
32 × 64	4	760	72	10.56×
64 × 32	8	397	139	2.86×
64 × 64	8	834	145	5.75×

- u-ORCA excels at us-scale DNN inference!

Experiments

Realistic workloads: MLP and Deepset variants

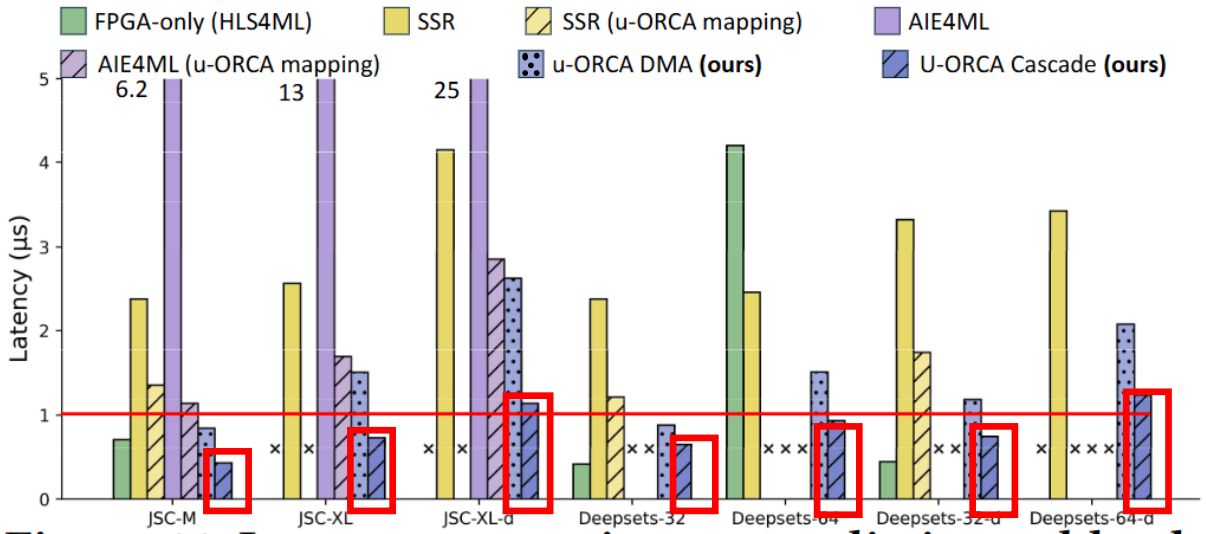


Figure 11: Latency comparison on realistic workloads

Latency of different global aggregation layers

Input	#AIE	Baseline	Ours	Speedup
32 × 32	4	373	66	5.65×
32 × 64	4	760	72	10.56×
64 × 32	8	397	139	2.86×
64 × 64	8	834	145	5.75×

- u-ORCA excels at us-scale DNN inference!
- Average **6.2x speedup** for the global aggregation

Experiments

Single AIE kernel performance: measured vs performance model

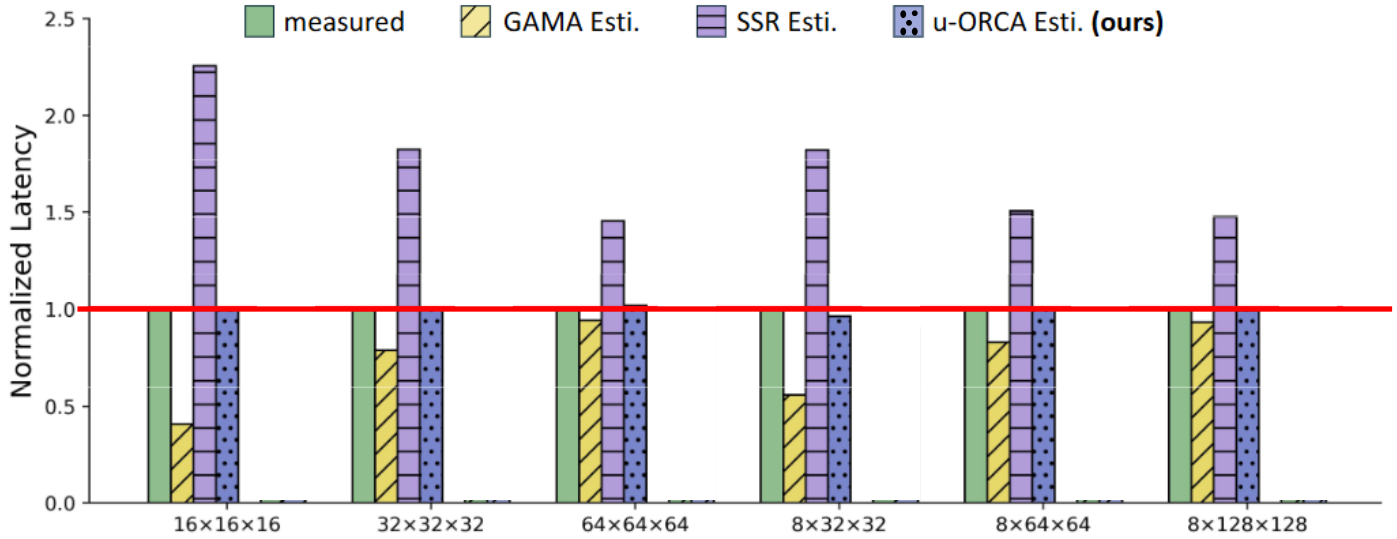


Figure 9: Normalized single AIE measured latency and estimated latency from different performance models.

Experiments

Single AIE kernel performance: measured vs performance model

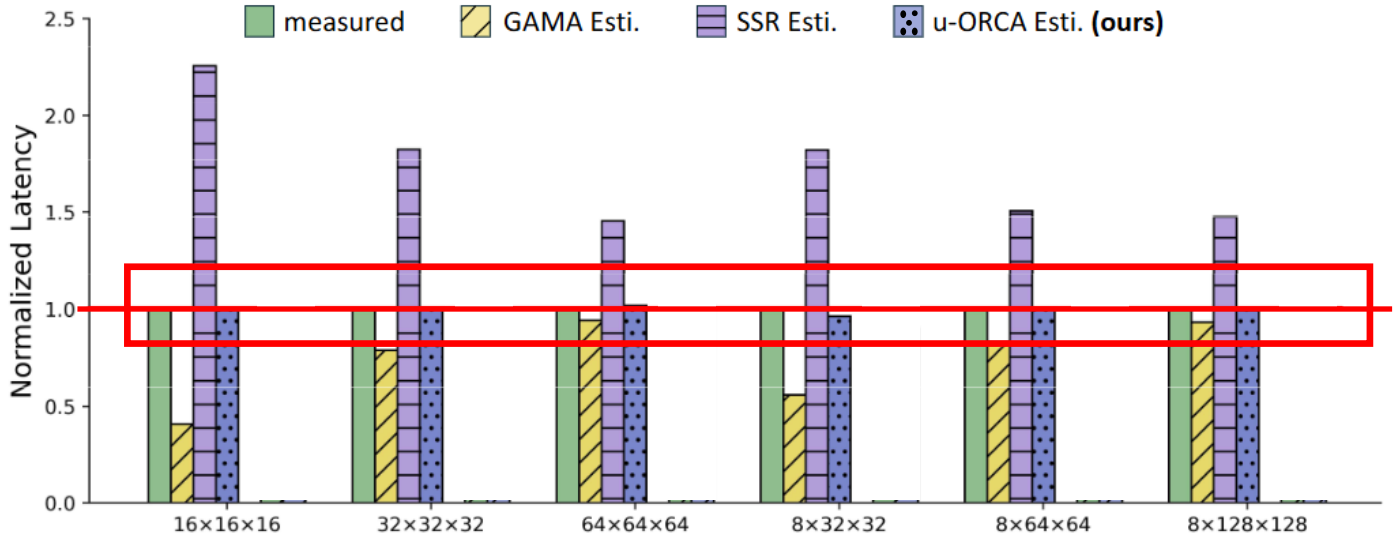


Figure 9: Normalized single AIE measured latency and estimated latency from different performance models.

- U-ORCA performance model can achieve high accuracy!

Thank You

μ-ORCA: Optimizing Acceleration for Microsecond-Scale Deep Neural Network Inference on ACAP

Shixin Ji*, Jinming Zhuang*, Zhuoping Yang*, Xingzhen Chen*, Wei Zhang*, Peipei Zhou*

Brown University*

shixin_ji@brown.edu

peipei_zhou@brown.edu

https://peipeizhou-eecs.github.io/

U.S. DEPARTMENT of ENERGY

National Science Foundation

AMD