Tensorlib: A Spatial Accelerator Generation Framework for Tensor Algebra

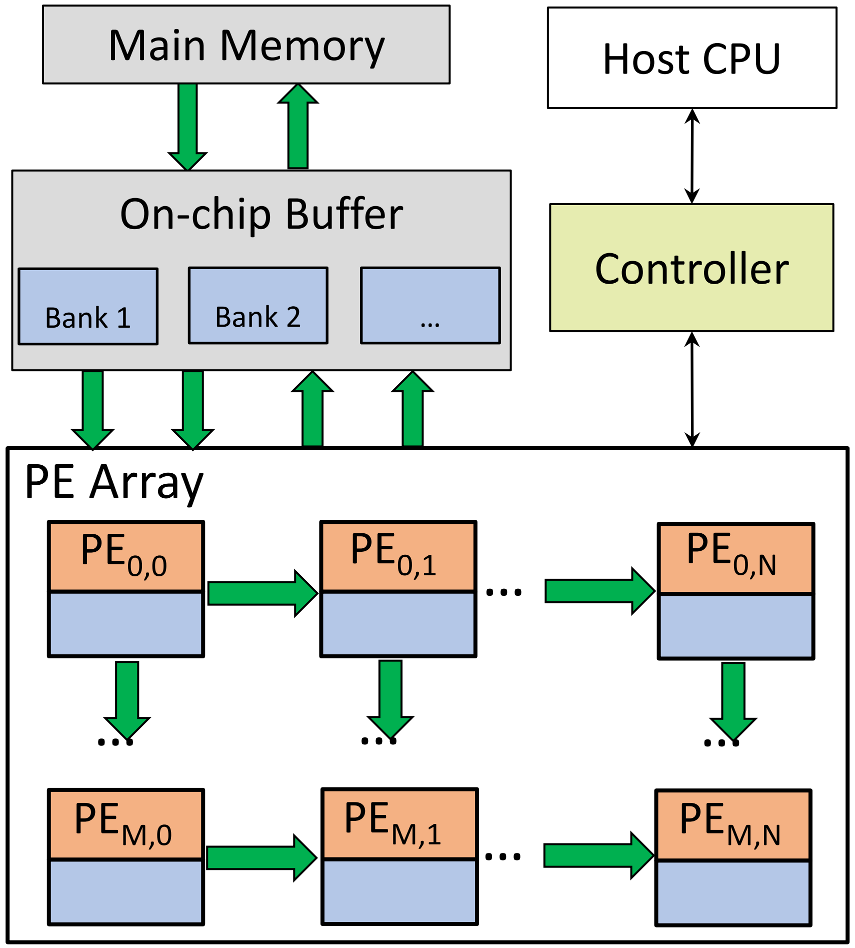
Liancheng Jia[1], Zizhang Luo[1], Liqiang Lu[1], and Yun Liang[1,2]
[1]Center for Energy-Efficient Computing and Applications, School of EECS, Peking University
[2]Pengcheng laboratory, China
{jlc, semiwaker, liqianglu, ericlyun}@pku.edu.cn

Overview

- **HASCO**
  - HW/SW Co-design for Tensor Computation
- **TENET**
  - Modeling Tensor Dataflow Based on Relation-centric Notation
- **Tensorlib**
  - Auto-Generation of Spatial Hardware Accelerators

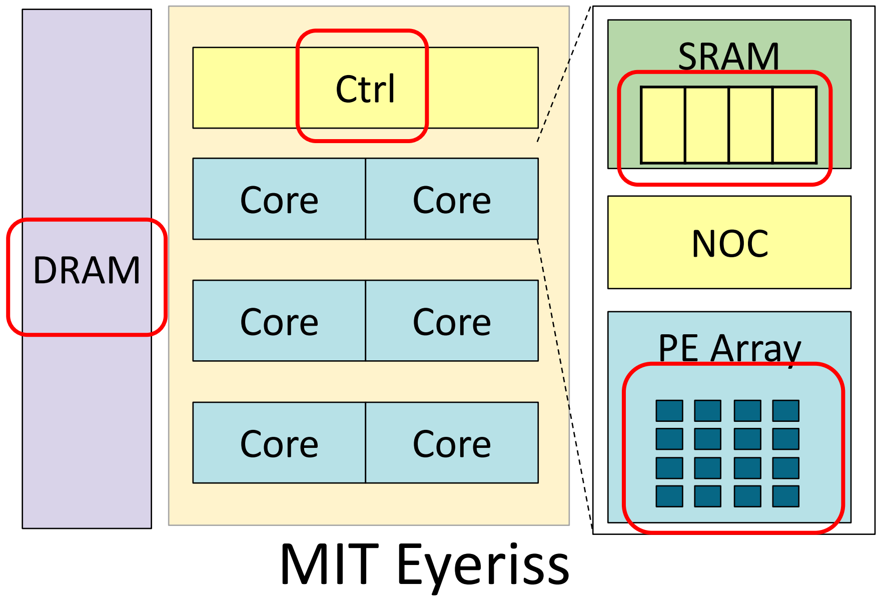
Spatial Accelerator Architecture

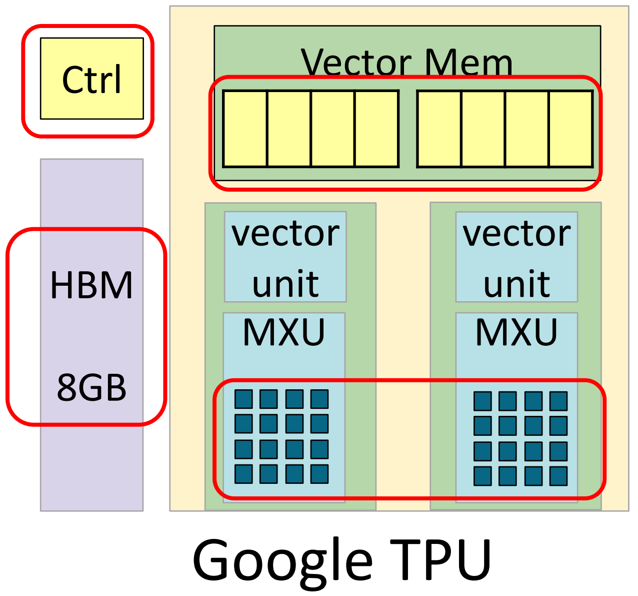
Typical Architecture for Spatial Accelerators

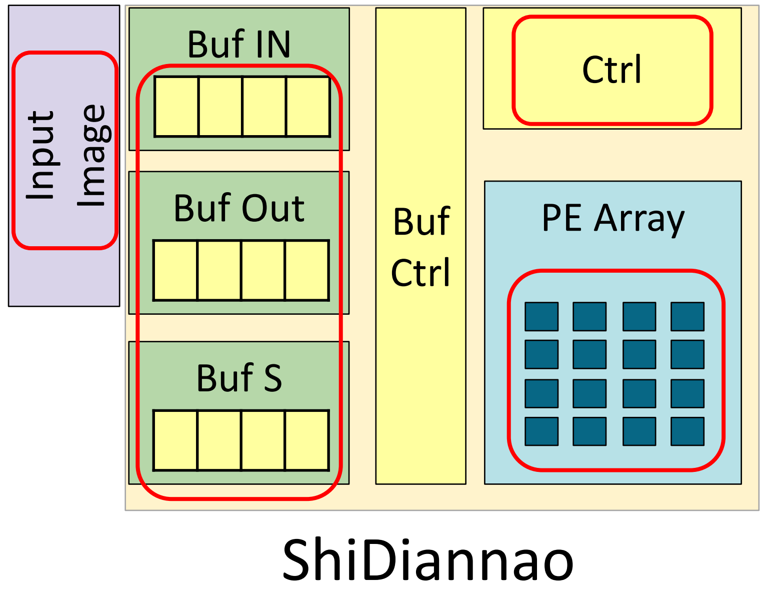


- ☐ PE Array
- ☐ On-Chip Buffer
- ☐ Controller

Spatial Accelerators for Tensor Algebra







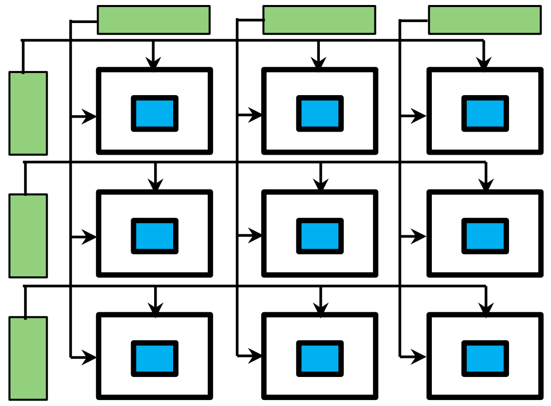
Multi-Level Storage

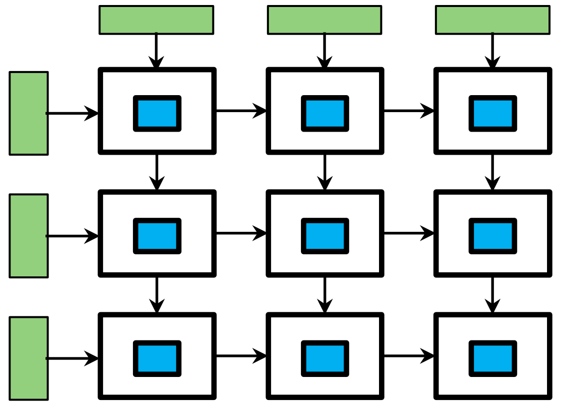
Unified Controller

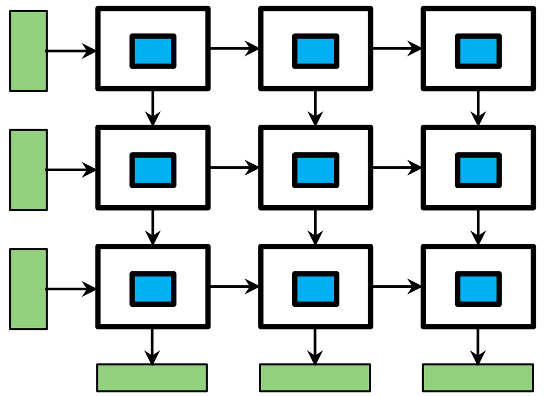
Multiple Memory Banks

2D PE Array

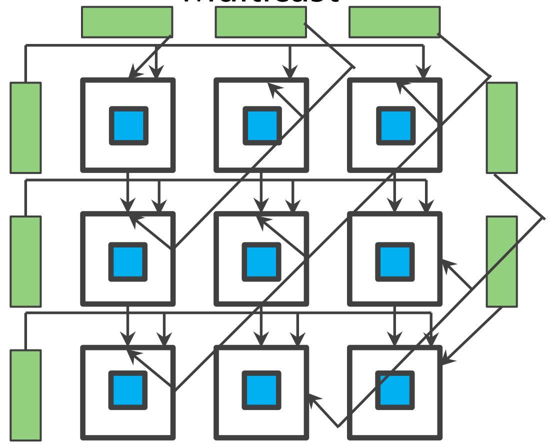
Various Hardware Dataflows

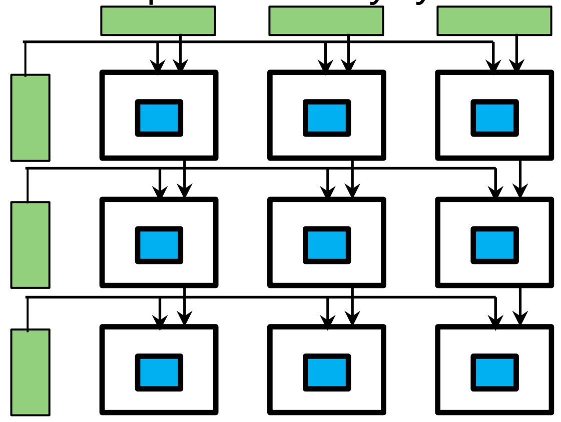

Multicast

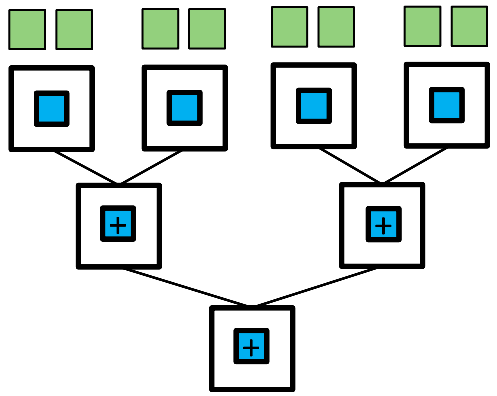

Output stationary systolic


Weight stationary systolic
(Applied by TPU)

......


Row Stationary
(Applied by Eyeriss)


Multicast+systolic
(Applied by ShiDiannao)


Reduction Tree
(Applied by MAERI)

......

Jouppi, Norman P., et al. "In-datacenter performance analysis of a tensor processing unit." ISCA 2017
Chen, Yu-Hsin, et al. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." JSSC 2016
Du, Zidong, et al. "ShiDianNao: Shifting vision processing closer to the sensor." ISCA 2015
H Kwon et al. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects ASPLOS 2018

Architecture Similarity and Difference

- ☐ Multi-Level Storage Hierarchy
- ☐ On-chip buffer is divided into banks
- ☐ PEs form a spatial array
- ☐ Controlled by a unified controller

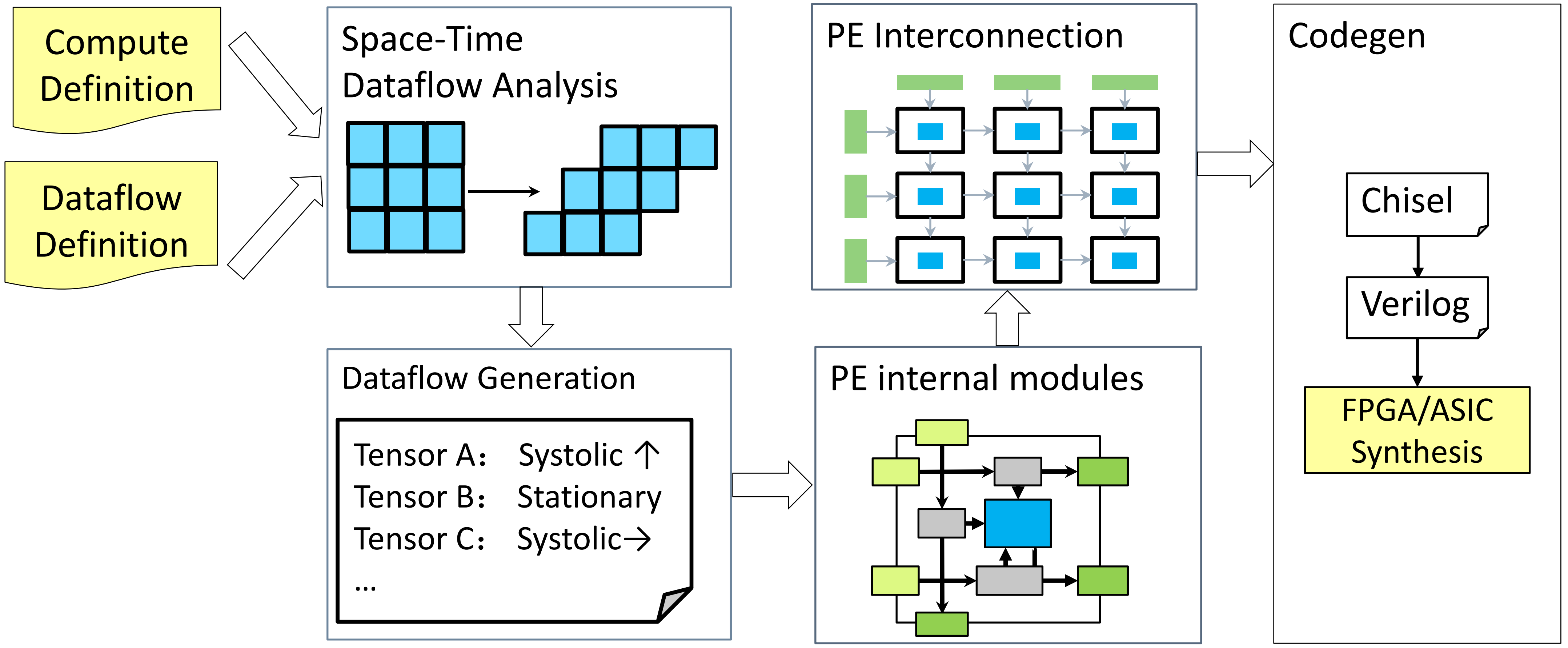
- ☐ Different PE array dataflow
- ☐ Different PE size and data type
- ☐ Different kernel implementation
- ☐ Different NoC architecture

Current Solution of Spatial Accelerator Development

	Expression	Performance	Productivity	Dataflow Support
Manual HDL	Verilog	★★★	★	★
High-Level Synthesis	C/C++	★★	★★	★
HLS + DSL	Halide / TVM DSL	★	★★★	★★
Tensorlib (Ours)	Space-Time Transformation	★★★	★★★	★★★

Dilemma between productivity and performance

Overview of Tensorlib



Simple Expression

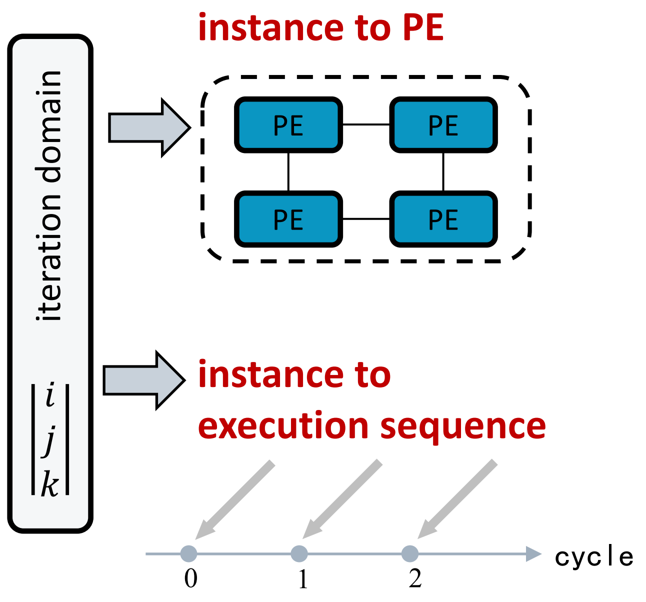
Various Dataflows

Good Productivity

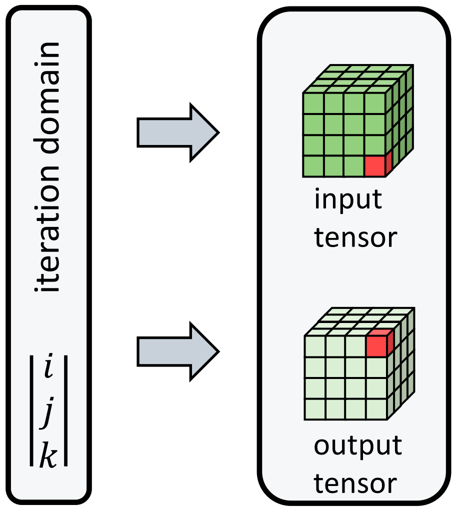
High Performance

Dataflow Analysis

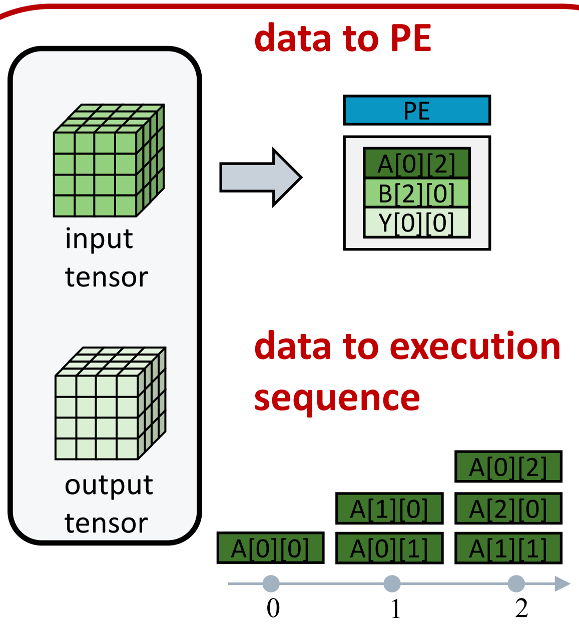
Relations in Spatial Accelerators



Dataflow Relation
Instance-Hardware



Instance-Data

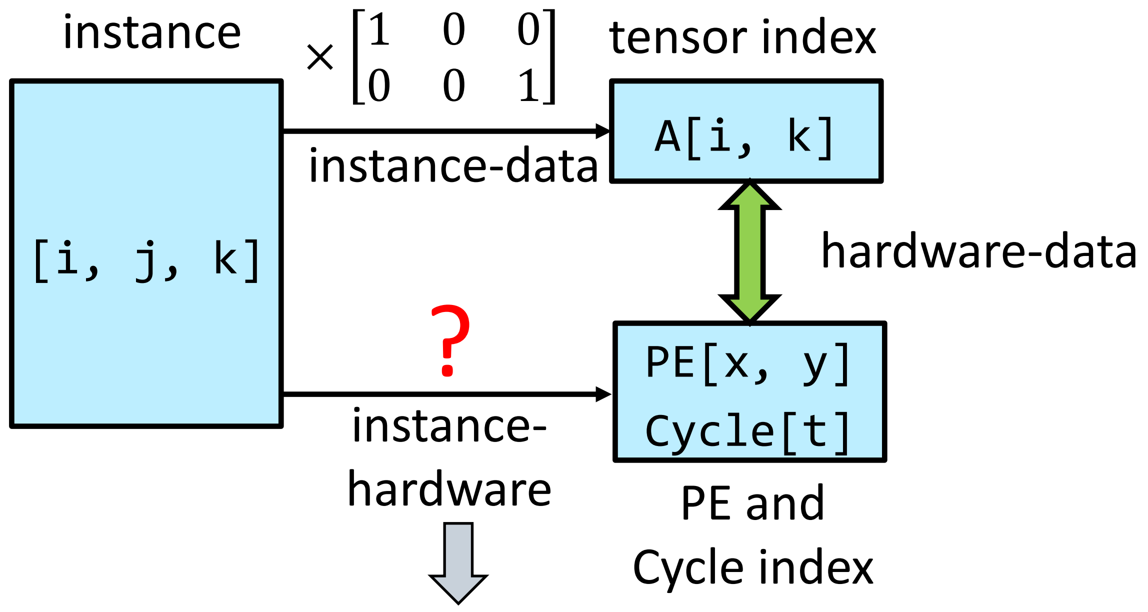


Data Assignment
Hardware-Data

L.Lu et al. TENET: A Framework for Modeling Tensor Dataflow Based on Relation-centric Notation ISCA, 2021

How to represent relations?

```
loop nest:
for  (i = 0; i < 3; i++)
  for  (j = 0; j < 3; j++)
    for  (k = 0; k < 3; k++)
      instance [i,j,k]:
      Y[i,j] += A[i,k] * B[k,j];
```



Space-Time Transformation

Space-Time Transformation

```
loop nest:
for  (i = 0; i < 3; i++)
  for  (j = 0; j < 3; j++)
    for  (k = 0; k < 3; k++)
      instance [i,j,k]:
      Y[i,j] += A[i,k] * B[k,j];
```

Space-Time Transformation (STT):

Transform loop iteration domain into hardware space-time domain

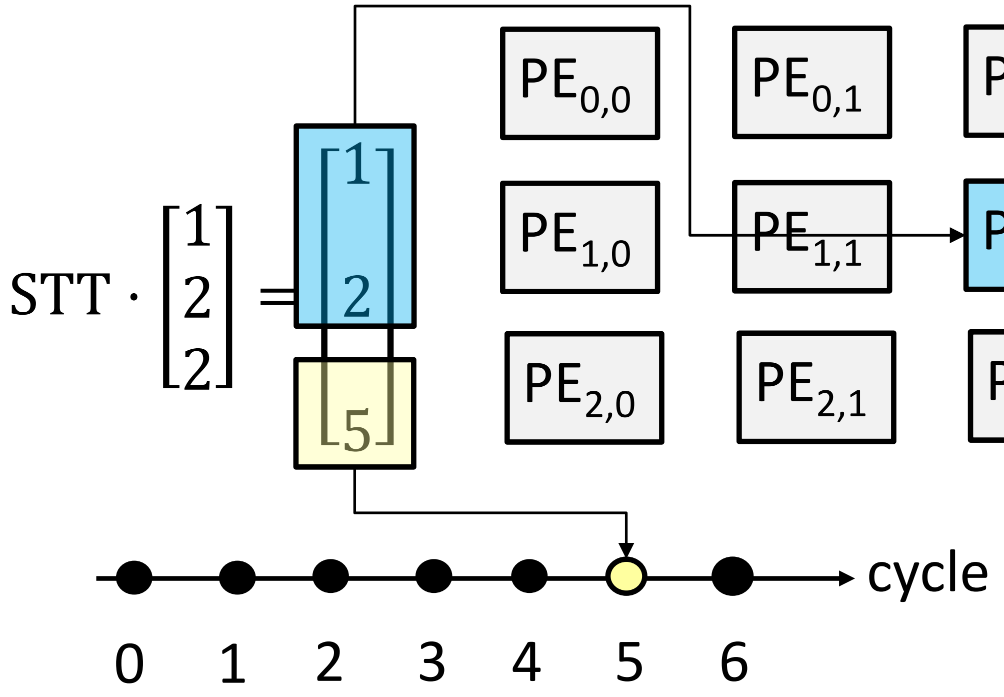
Can be expressed with Matrix Multiplication

$$STT \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} x \\ y \\ t \end{bmatrix}$$

Example:

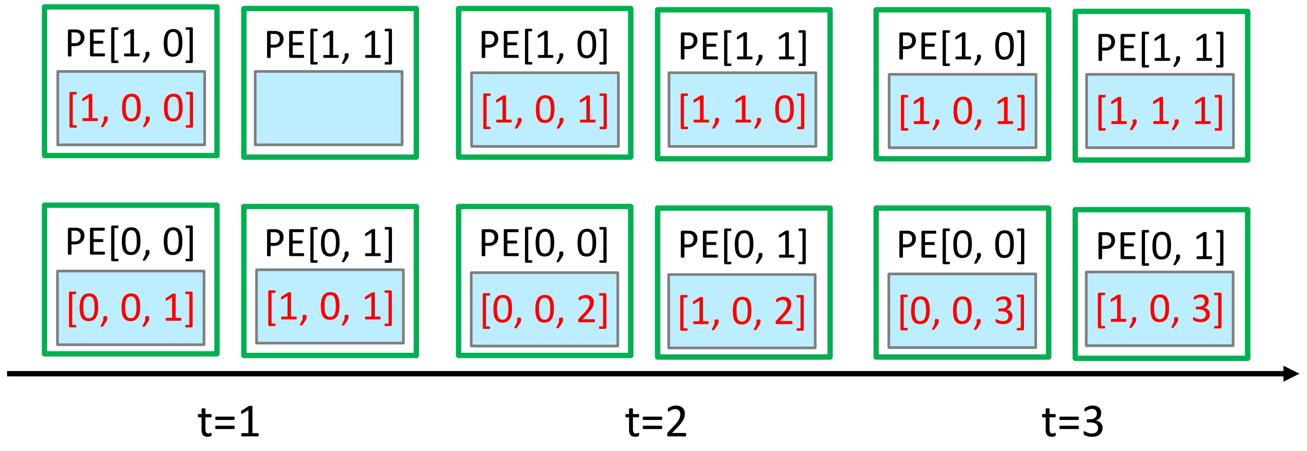
$$STT = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{Index:} \begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

$$STT \cdot \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$$

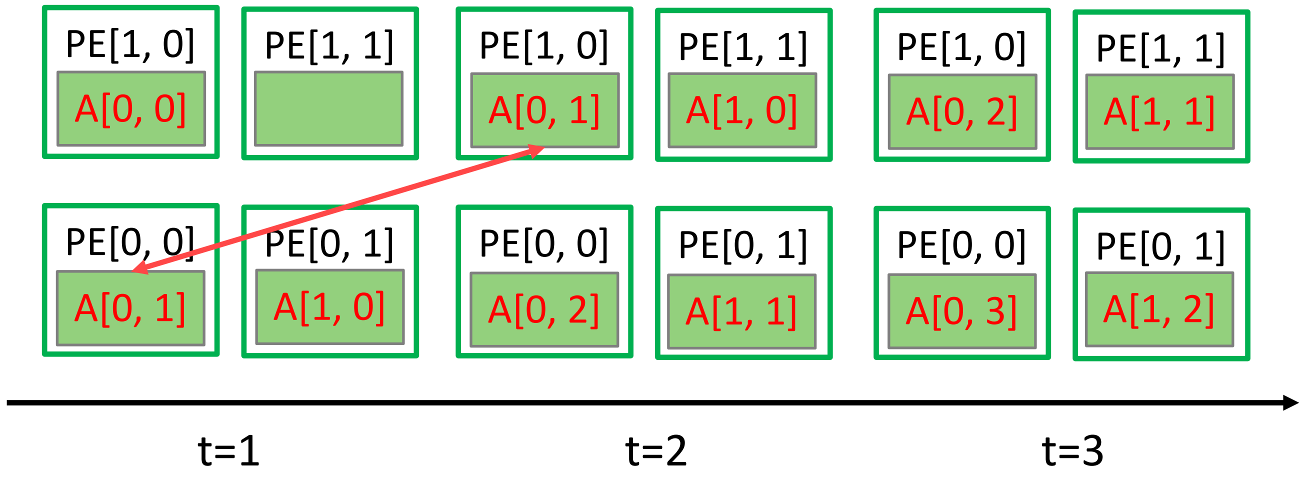


Formulating Hardware-Data Relation

Step 1: space-time domain -> instance domain



Step 2: instance domain -> tensor index domain



Step 1: Inverse Space-Time Transformation:

$$STT^{-1}\begin{bmatrix} x \\ y \\ t \end{bmatrix} = \begin{bmatrix} i \\ j \\ k \end{bmatrix}$$

Step 2: Tensor Access Formula:

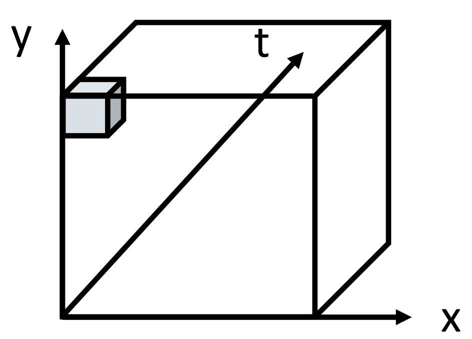
$$TA\begin{bmatrix} i \\ j \\ k \end{bmatrix} = \begin{bmatrix} m \\ n \end{bmatrix}$$

Combine two steps together:

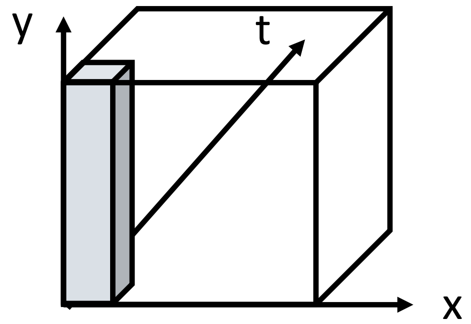
$$TA \cdot STT^{-1}\begin{bmatrix} x \\ y \\ t \end{bmatrix} = \begin{bmatrix} m \\ n \end{bmatrix}$$

Reuse Space and PE-Memory Interconnection

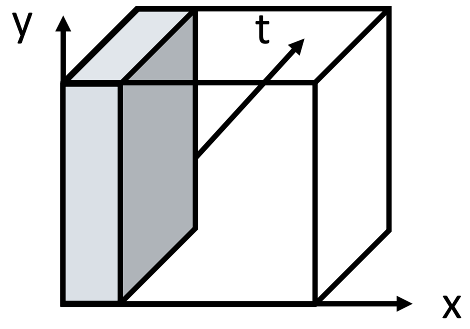
Reuse subspace
In (x, y, t) domain



0-D

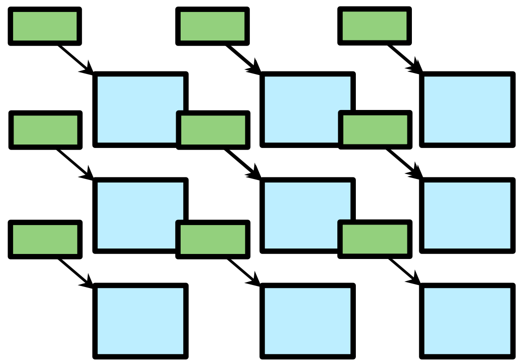


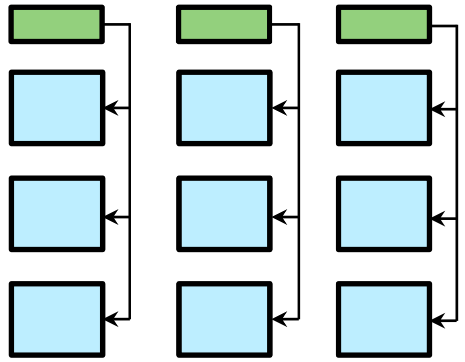
1-D

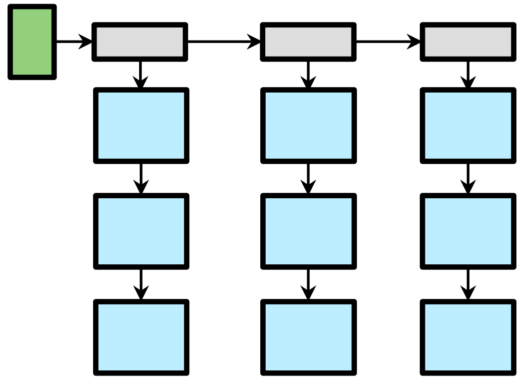


2-D

PE-Memory
Interconnection
Example







Each PE reads data once
from memory bank
No data reuse

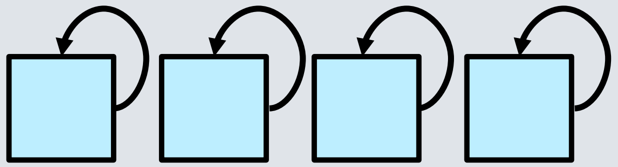
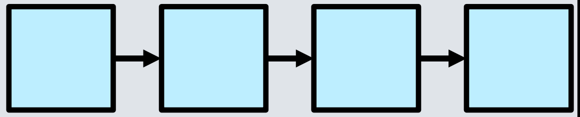
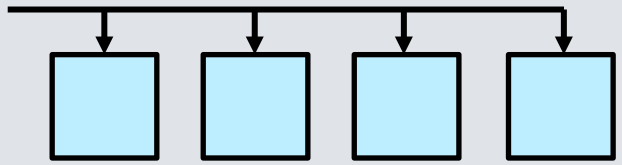
Reuse subspace forms
a line in space-time

Reuse subspace forms
a plane

1-D Reuse Type Identification

1-D reuse type forms a vector in space-time domain

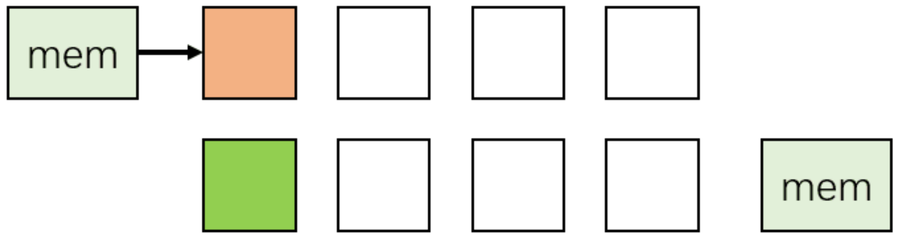
$$\text{Reuse direction } x = \begin{bmatrix} \vec{p} \\ t \end{bmatrix} \begin{array}{l} \longleftarrow \text{space part} \\ \longleftarrow \text{time part} \end{array}$$

Reuse Vector Direction	$\vec{p} = \vec{0}$ $t \neq 0$	$\vec{p} \neq \vec{0}$ $t \neq 0$	$\vec{p} \neq \vec{0}$ $t = 0$
PE Behavior	 Stay in the same PE during execution stage	 Delay data before send to the next PE	 Data arrive at PEs at the same time
Dataflow Type	Stationary	Systolic	Multicast

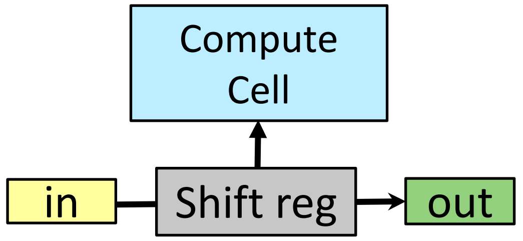
Hardware Generation

Systolic Dataflow

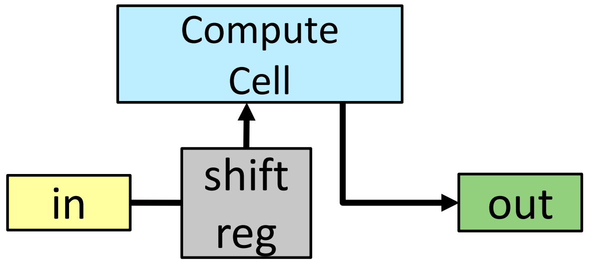
Reuse Vector $x = \begin{bmatrix} \vec{p} \\ t \end{bmatrix} \begin{matrix} \longleftarrow \vec{p} \neq \vec{0} \\ \longleftarrow t \neq 0 \end{matrix}$



- Transfer to PE $x + \vec{p}$ from PE x after delaying t cycles
- Use register to delay input data in each PE



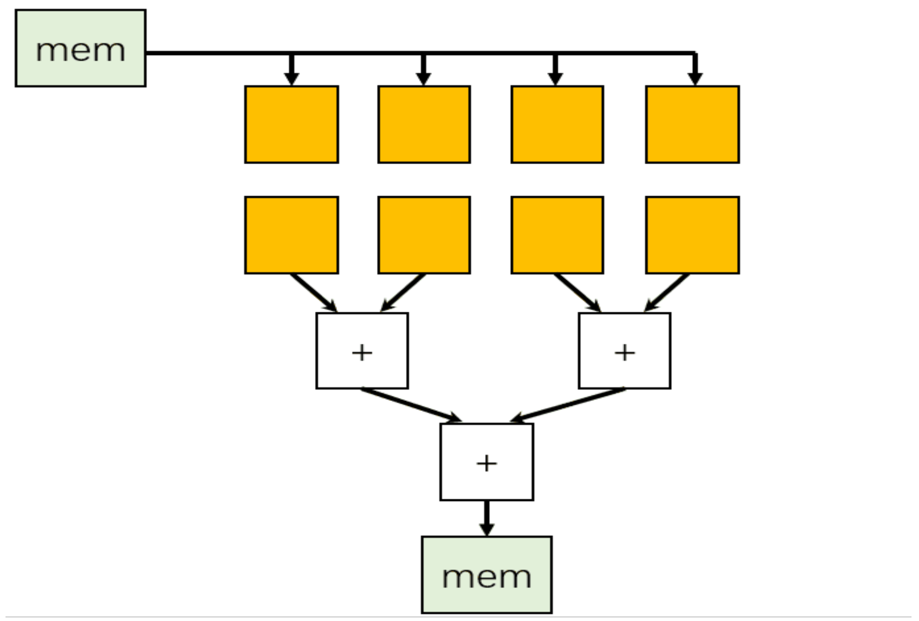
PE Dataflow for Input Tensor

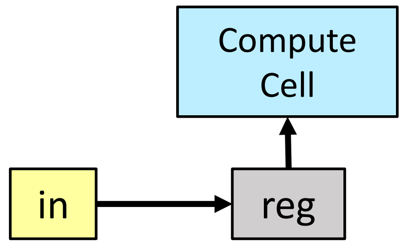


PE Dataflow for Output Tensor

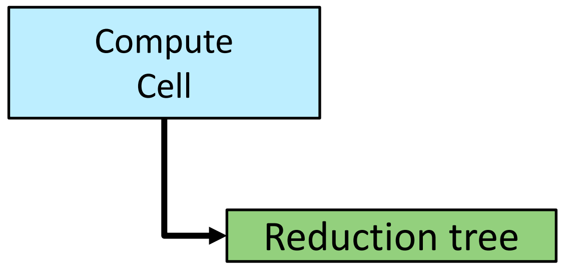
Multicast Dataflow

$$\text{Reuse Vector } x = \begin{bmatrix} \vec{p} \\ t \end{bmatrix} \begin{array}{l} \longleftarrow \vec{p} = \vec{0} \\ \longleftarrow t \neq 0 \end{array}$$





PE Dataflow for Input Tensor

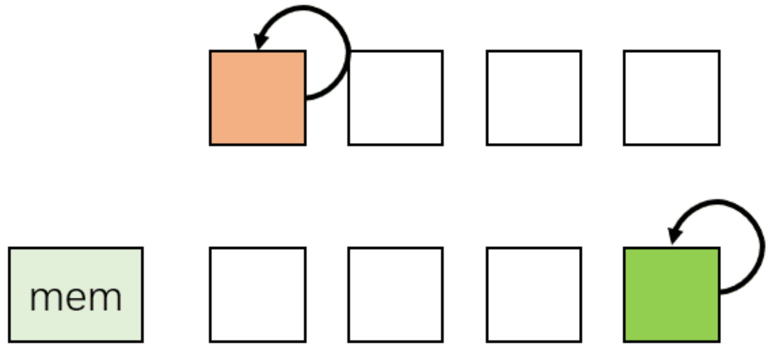


PE Dataflow for Output Tensor

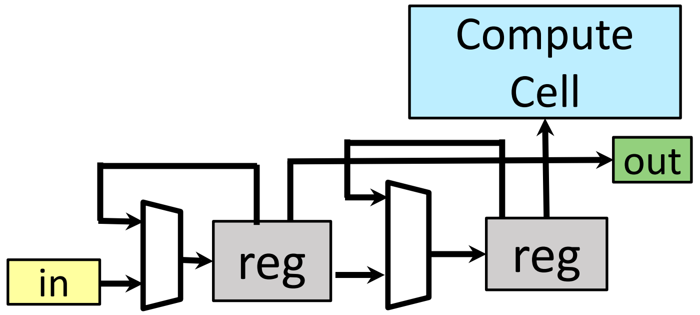
- Transfer to PEs in the same line simultaneously
- Directly send to/from compute cell
- Use reduction tree for output communication

Stationary Dataflow

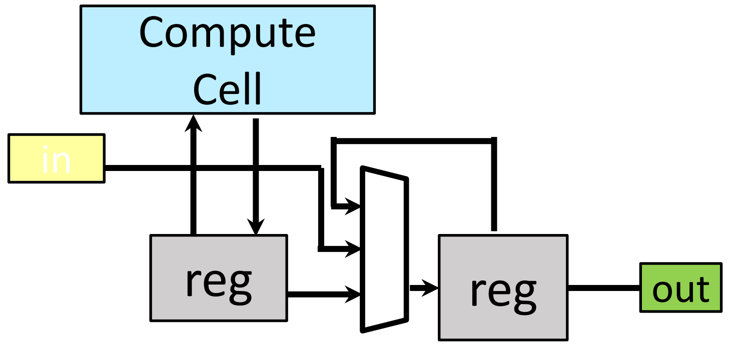
$$\text{Reuse Vector } x = \begin{bmatrix} \vec{p} \\ t \end{bmatrix} \begin{matrix} \longleftarrow \vec{p} = \vec{0} \\ \longleftarrow t \neq 0 \end{matrix}$$



- Keep stationary inside PE during execution
- Requires systolic dataflow to move in/out to/from PE array
- Use double buffer for interleaving two dataflows

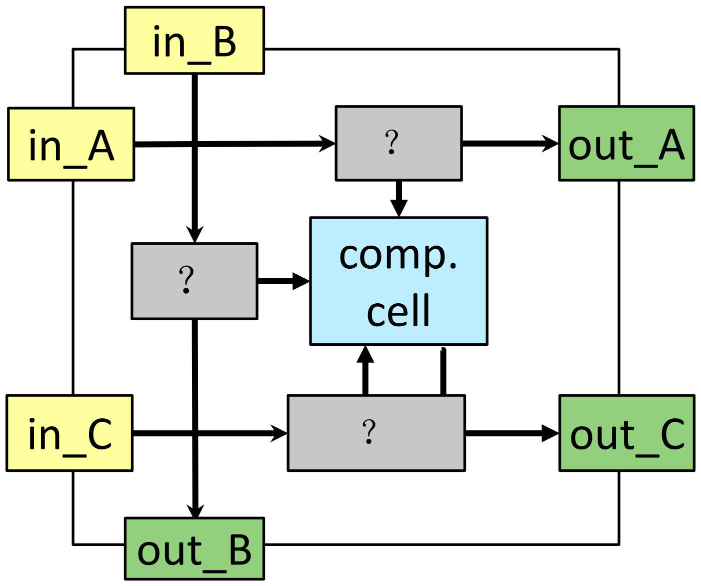


PE Dataflow for Input Tensor

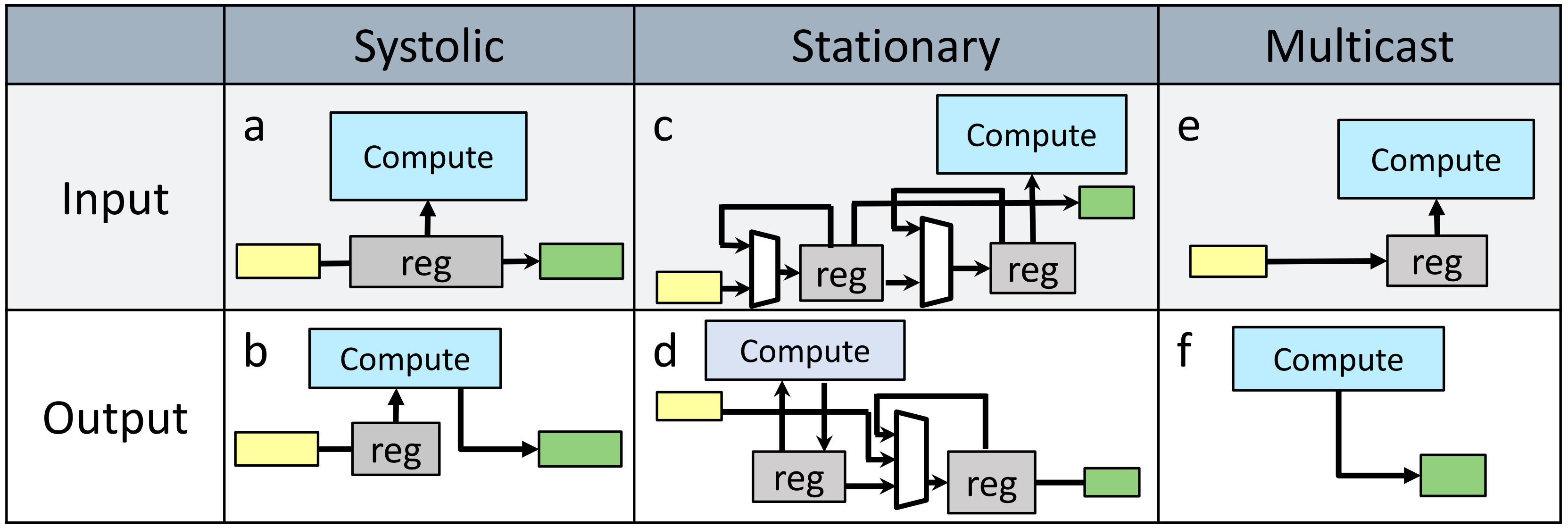


PE Dataflow for Output Tensor

Building PE Inner Architecture



PE Basic Structure



PE Dataflow Structure

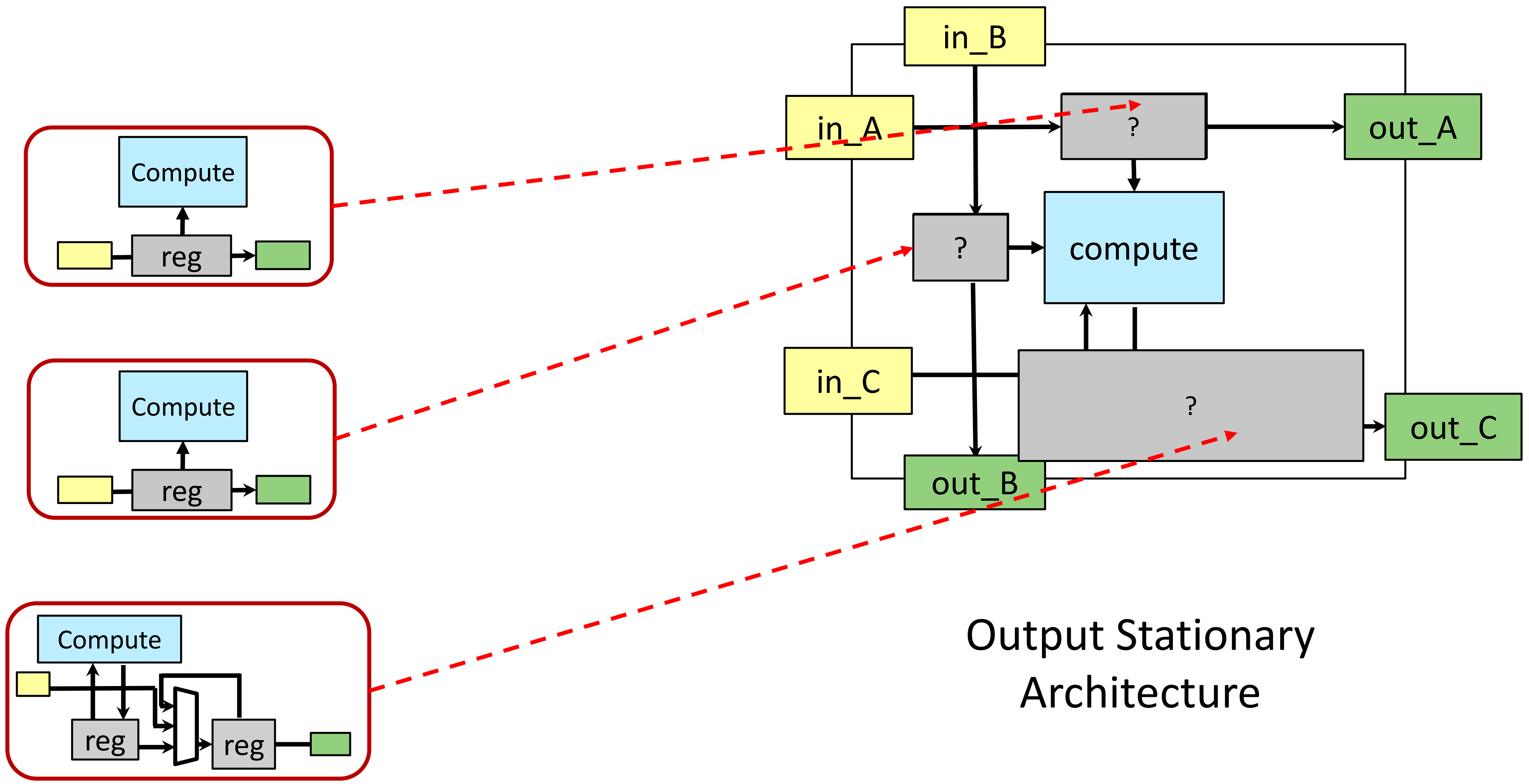
Substitute each of the ? module to one of the 6 dataflow modules based on the dataflow of each tensor

Building PE Architecture - Example

A: Systolic

B: Systolic

C: Stationary



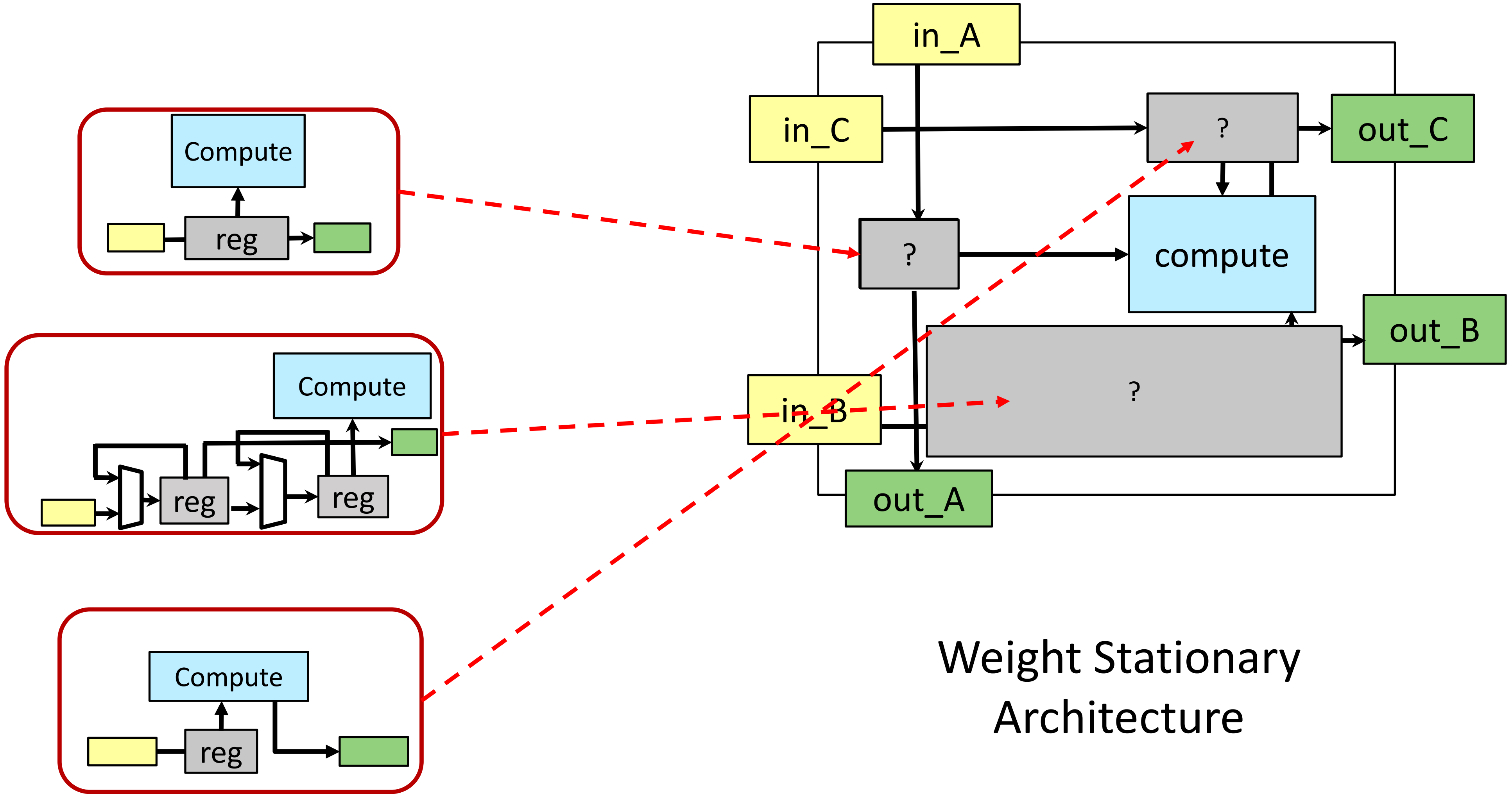
Output Stationary Architecture

Building PE Architecture - Example

A: Systolic

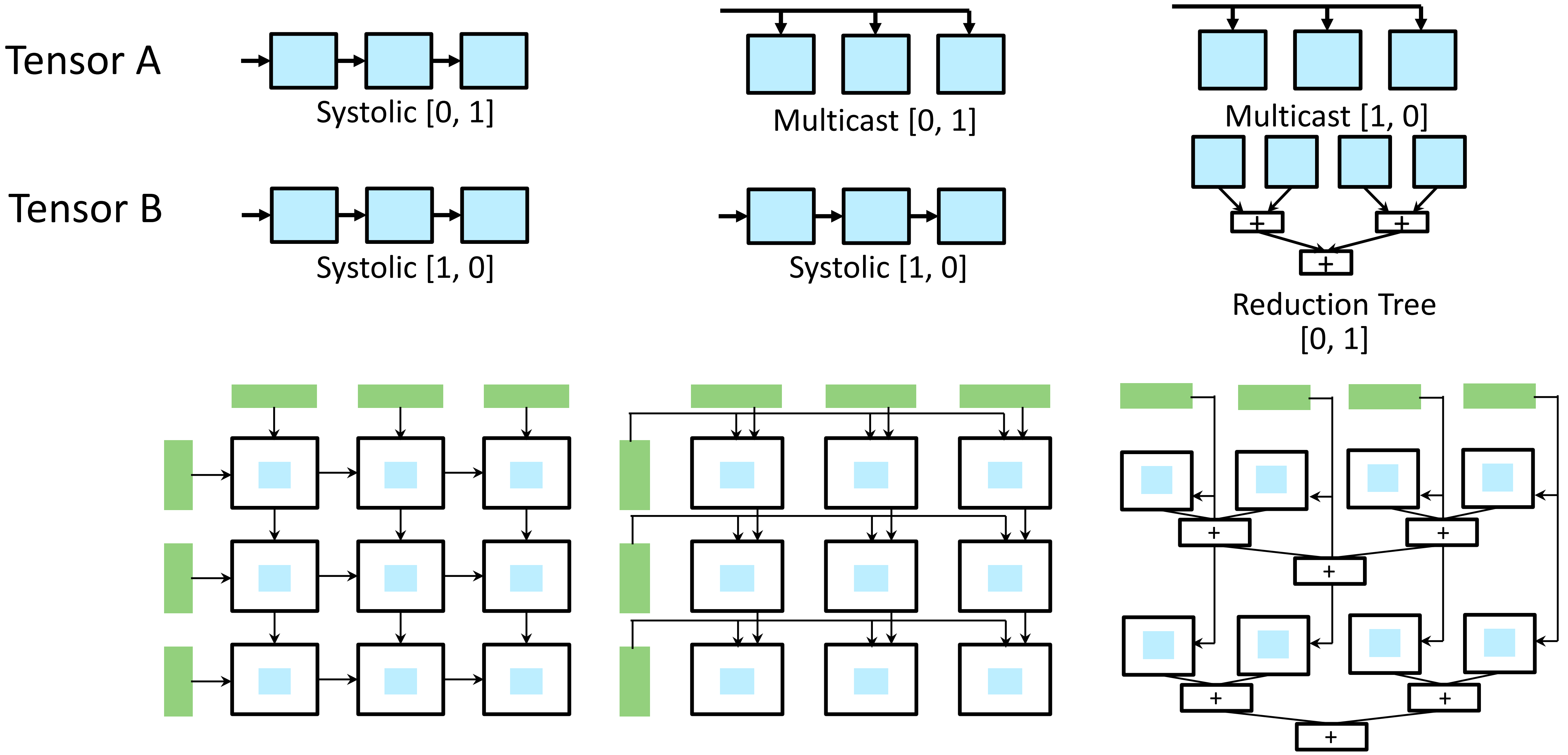
B: Stationary

C: Systolic

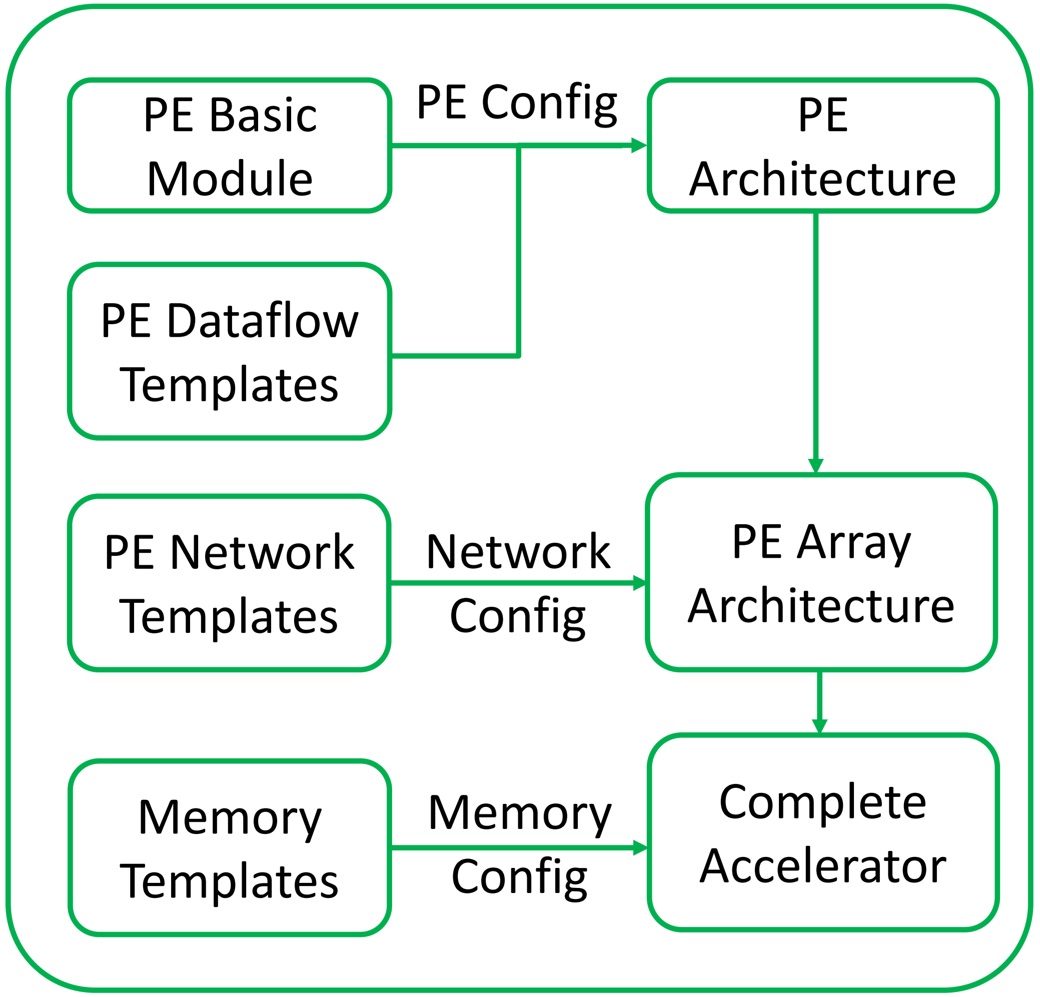


Weight Stationary Architecture

Building PE Array Interconnection - Example



Implementation



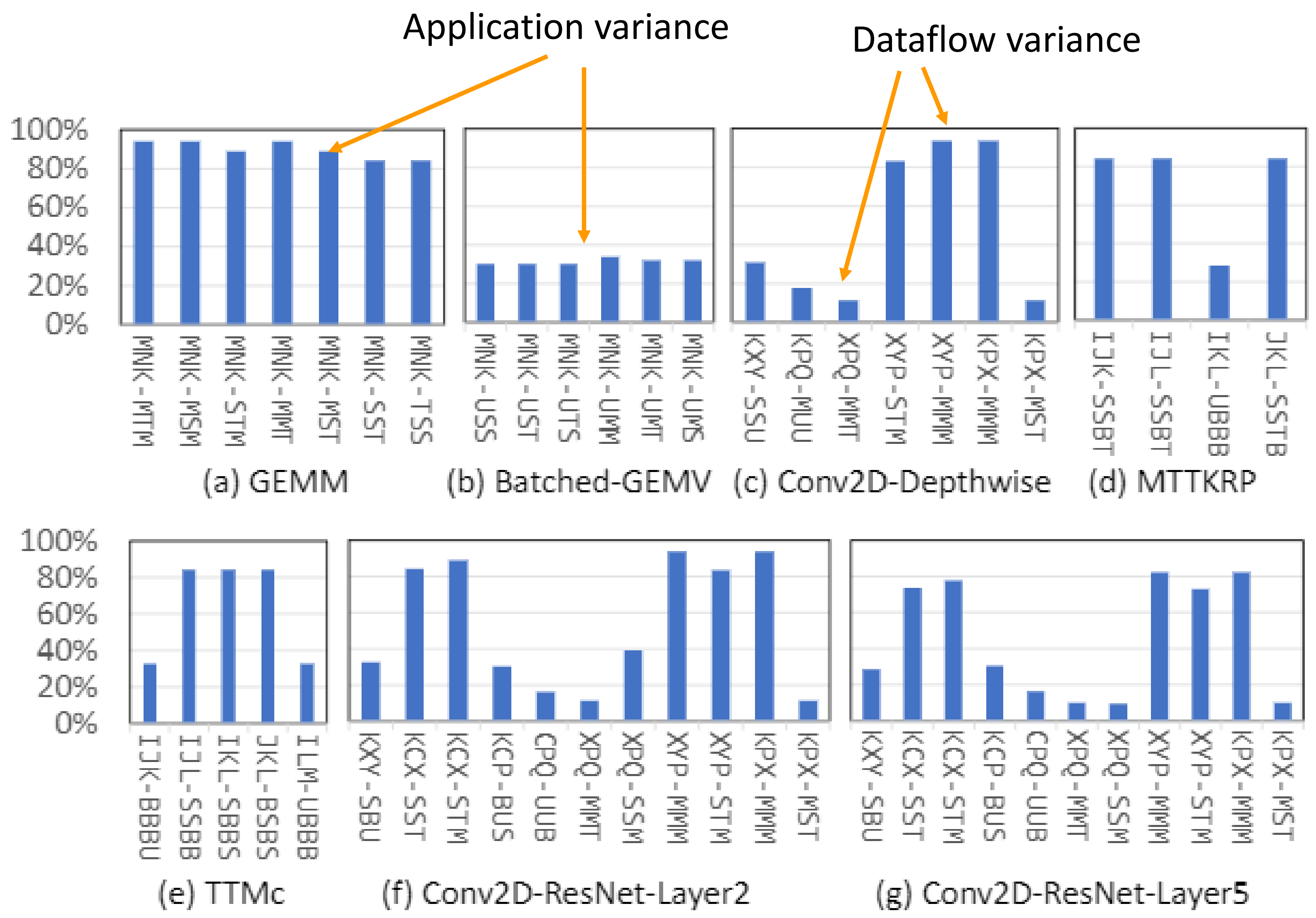
```
class PEArray(params):

  val PEs=Array[size]new PE(p
e_config)

  val pe_net = new PENetwork(
net_config)
  connect_PE(PEs, pe_net)
  val mem = Array[banks] new
MemBank(mem_config)
  connect_mem(pe_net, mem)
```

Experiments and Evaluation

Dataflow Performance Evaluation



- Evaluated 6 tensor applications
- Included Systolic, Multicast, Stationary, Unicast, Broadcast dataflows
- Performance variance because of dataflow mapping and bandwidth

FPGA Performance Results

	Susy [ICCAD'20]		PolySA [ICCAD'18]		Tensorlib	
	Arria-10		Xilinx VU9P		Xilinx VU9P	
Application	GEMM	Conv	GEMM	Conv	GEMM	Conv
LUT(%)	40%	35%	49%	49%	68%	75%
DSP(%)	93%	84%	89%	89%	75%	75%
BRAM(%)	32%	30%	89%	71%	51%	73%
Freq(MHz)	202	220	229	229	263	245
Throughput (Gop/s)	547	551	555	548	673	626

21% Frequency Improvement
15% Throughput Improvement

Summary

- Tensorlib: A Spatial Accelerator Generation Framework for Tensor Algebra
- Dataflow Representation
  - Analysis based on space-time transformation
- Hardware Generation
  - Hierarchical bottom-up generation
- Open source: https://github.com/pku-liang/tensorlib

Framework Tutorial : Tensorlib API

- **Tensorlib provides API to define computation**

Tensorlib API	Description
genIterators(x)	Define x iterators and return the iterator list
genTensors(x)	Define x tensors and return the tensor list
setExpr(expr)	Define the compute expression with tensors and iterators
iterator.setRange(x)	Set the range of iterators as x
Tensor.setWidth(x)	Set the tensor data bitwidth as x
setLatency(x)	Define the latency of computation pipeline
useCustomKernel	Whether use customized IP for computation cell

Framework Tutorial: GEMM

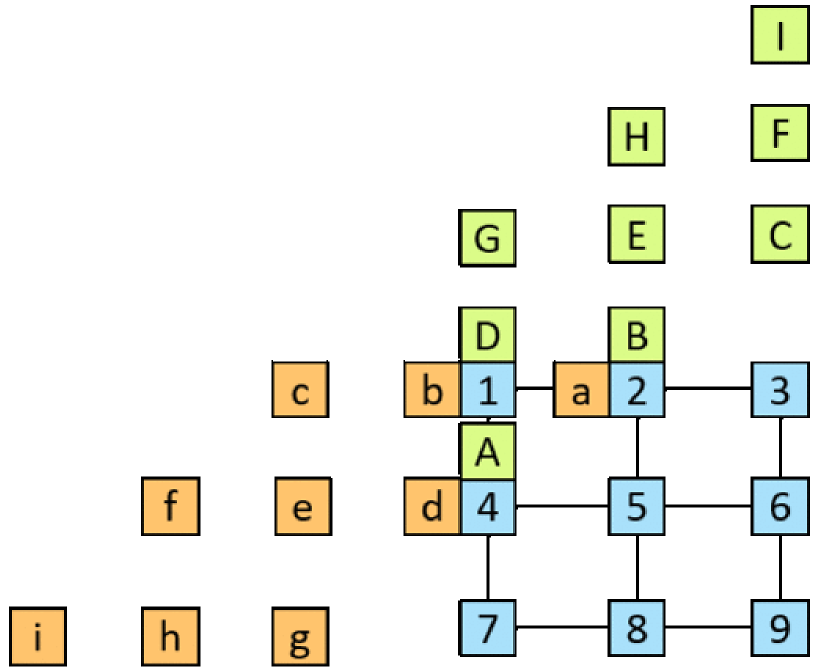
Example:

GEMM

Dataflow:

output stationary Systolic array

```
for  (i = 0; i < 3; i++)
  for  (j = 0; j < 3; j++)
    for  (k = 0; k < 3; k++)
        C(i)(j) += A(i)(k) * B(k)(j)
```



Framework Tutorial: Example

Step 1: Workload Definition

```
val opSpec_Gemm = new OperatorSpec {
      val i :: j :: k :: Nil = genIterators(3)
      val A :: B :: C :: Nil = genTensor(3)
      setExpr(C(i)(j) += A(i)(k) * B(k)(j))
      i.setRange(8)
      k.setRange(20)
      j.setRange(8)
      setLatency(1)
      A.setWidth(16)
      B.setWidth(16)
      C.setWidth(16)
    }
```

Step 2: Space-Time matrix definition

```
  val stt = DenseMatrix(
   (1,  0,  0),
   (0,  1,  0),
   (1,  1,  1),
  )
```

Step 3: Generate dataflow and Verilog code

```
val config = gen_dataflow(opSpec_Gemm,
stt)

chisel3.Driver.execute(args, () => new
PEArray(config))
```

Framework Tutorial: Using Pre-defined Configs

- **Running Tensorlib with Pre-defined dataflow configs**

```
sbt "runMain tensorlib.ParseJson
examples/gemm.json output/output.v"
```

- **Config File Example (Can be generated by HASCO):**

```
{
    "benchmark": "GEMM",
    "bitwidth": 16,
    "length":[8,8,8],
    "STT":[[1,0,0],[0,0,1],[1,1,1]]
}
```

Framework Tutorial: Customized Kernels

- Default Kernel: Integer Multiply-Add
- Can be substituted with user-provided IPs

Step 1: Choose to use customized kernel In computation definition

```
val opSpec = new OperatorSpec {
        ......
        useCustomKernel(true)
        ......
}
```

Step 2: Provide customized computation kernel & reduction kernel

```
module ComputeCell_Customized(
  input         clock,
  input         reset,
  input  [15:0] io_data_2_in,
  input  [15:0] io_data_1_in,
  input  [15:0] io_data_0_in,
  output [15:0] io_data_0_out
);
```

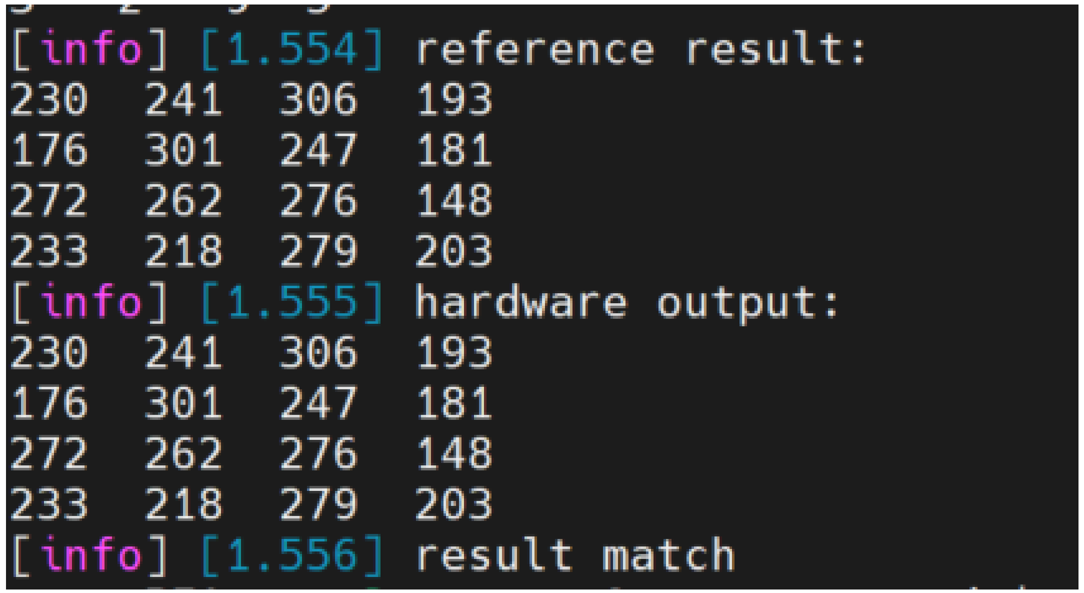
```
module Reduction_Customized(
  input  [15:0] io_in_a,
  input  [15:0] io_in_b,
  output [15:0] io_out
);
```

Result Validation

- **Chisel Tester Validation**

```
sbt "runMain tensorlib.Test_Gemm"
```

- Runs a validation program of tensorlib generated GEMM accelerator based on Chisel cycle-level simulator



```
[info] [1.554] reference result:
230  241  306  193
176  301  247  181
272  262  276  148
233  218  279  203
[info] [1.555] hardware output:
230  241  306  193
176  301  247  181
272  262  276  148
233  218  279  203
[info] [1.556] result match
```