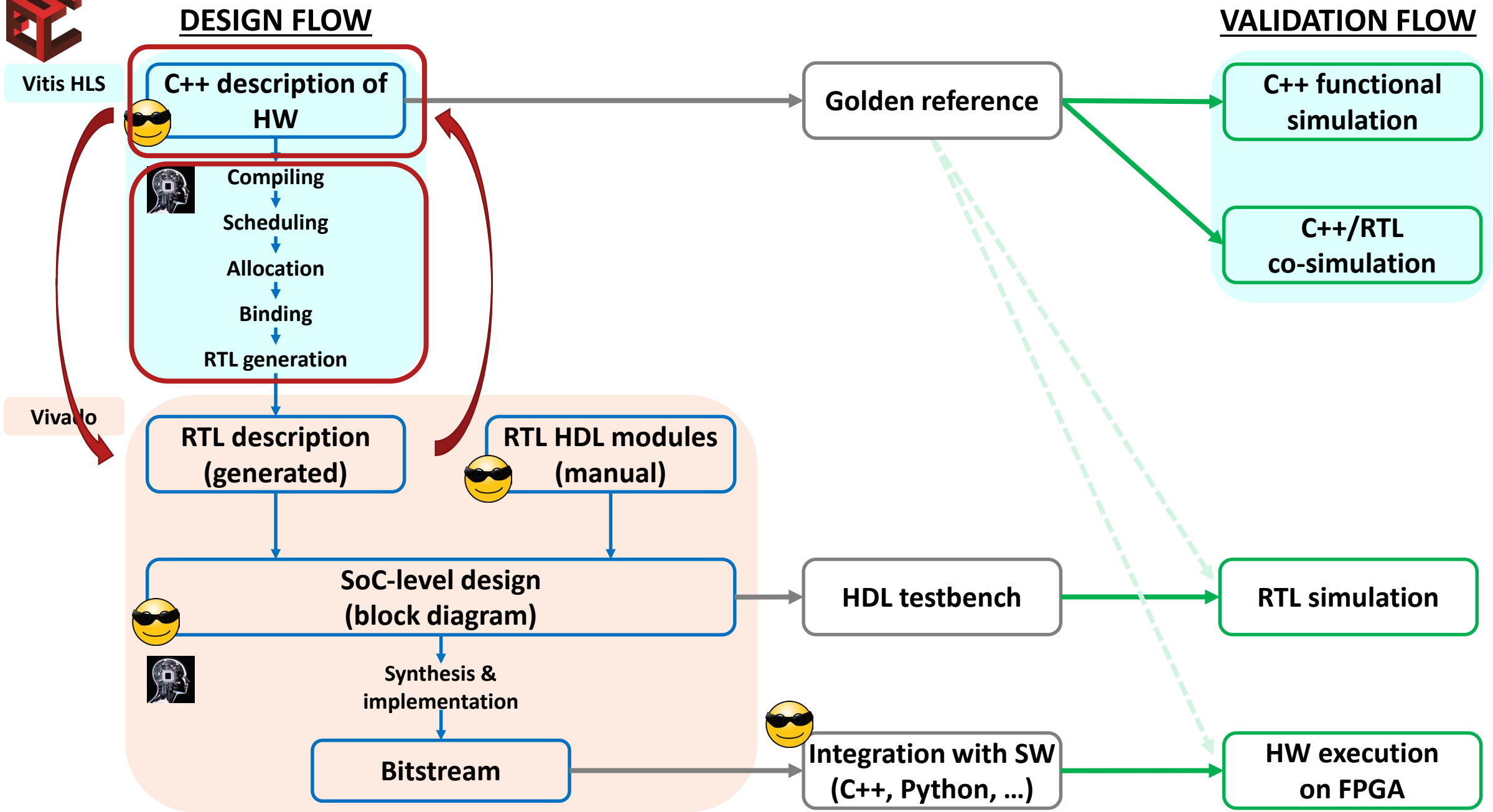
EPFL

Lab. On HW-SW Digital Systems Codesign
EE-390(a)

Session 4
Array optimization using HLS

Prof. David Atienza
Dr. Denisa Constantinescu, Dr. Miguel Peón-Quirós,
Mr. Rubén Rodríguez-Álvarez, Ms. Stasa Kostic, Mr. Karan Pathak

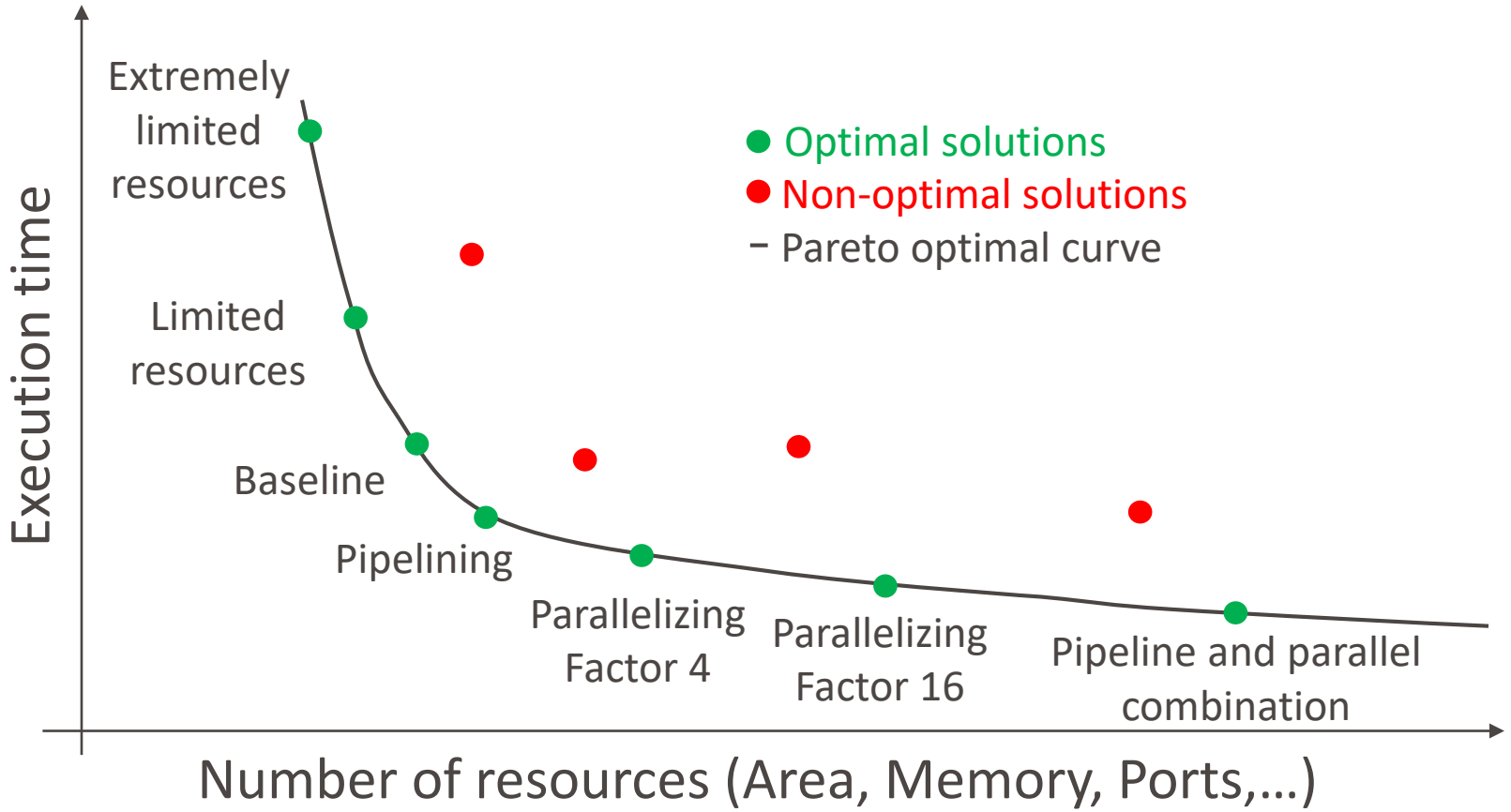
High-level synthesis (HLS) design flow for HW accelerators



Design Space Exploration (DSE)

Metrics

- Latency
- Throughput
- Energy
- Resources



Learning Objectives: HLS optimizations

- Loops optimizations (last week with Rubén)
  - Pipelining
  - Unrolling
  - Merging
  - Flattening
- Array optimizations
- What types of storage resources do we have on the FPGA
  - How to map software arrays to hardware memory
    - Partitioning
    - Reshaping
- How to combine loop & array optimizations?

.... there is a full list of pragmas and directives optimizations

What types of storage resources do we have on the FPGA?

Remember the HLS translation table

C++	Generated HW
Functions	Modules with I/Os
Arguments	Ports of modules
Private variables	Local signals
Static variables	???
Arrays (internal)	???
Arrays or pointers in arguments	???
Variables	Signals (Wires)
Function calls	Modules instantiation

Types of storage units on the FPGA

- RAM: BRAM, UltraRAM (URAM)
- LUTs (LUTRAM)
- Shift Register Logic (SRL)
- Registers (FFs)

What does the HLS compiler infer from these HW descriptions?

A. int my_array_A[ ]; ???

B. int my_array_B[**10**]; ???

C. int my_array_C[**1024**]; → shift register logic (SRL)

D. int my_array_D[**1025**]; → BRAM (default); LUTRAM or URAM with **BIND_STORAGE** directive/pragma

HLS is not C++ → not all C++ features are available!
dynamic memory allocation for arrays is not supported

Exercise: how many cycles it takes to the loop?

```
int A[N];
// The loop below is the shift operation
for (int i = 0; i < N-1; ++i){
#pragma unroll
    A[i] = A[i+1];
}
```

a) N=1024 → ?
b) N=1025 → ?

Array optimizations
Types of storage resources

Internal arrays:

```
#pragma HLS bind_storage variable=<variable> type=<type>
```

- FIFO
- RAM single port
- RAM single write port and multiple read ports
- RAM dual port (1 read only, and the other for read and write)
- RAM dual port (1 read only, and the other write only)
- True dual port RAM (both ports for read and write)
- Single port ROM, Dual port ROM, and multi-port ROM.

More about supported types and how to use the pragma_bind storage in AMD documentation:
https://docs.amd.com/r/en-US/ug1399-vitis-hls/pragma-HLS-bind_storage

Why do we need array optimizations?

```
// assuming single port memory for each
a[2048];
b[2048];
o[2048];

Loop_1: for(int i = 0; i  < 2048; i++) {
#pragma unroll factor=10
   o[i] = a[i] + b[i];
}
```

Latency

		baseline	loop_unroll_5	loop_unroll_10
Latency (cycles)	min	1002	202	102
	max	1002	202	102
Latency (absolute)	min	10.020 us	2.020 us	1.020 us
	max	10.020 us	2.020 us	1.020 us
Interval (cycles)	min	1003	203	103
	max	1003	203	103

Two main internal array optimizations

- Partitioning
- Reshaping

Array optimizations
Partitioning

Original data

Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7
Data 8
Data 9
Data 10
Data 11

Partitioned data

Complete

Data 0	Data 6
Data 1	Data 7
Data 2	Data 8
Data 3	Data 9
Data 4	Data 10
Data 5	Data 11

The complete partitioning separates all the elements of the array in separate **registers individually accessible**

Block factor 4

Mem_0	Data 0
	Data 1
	Data 2
Mem_1	Data 3
	Data 4
	Data 5
Mem_2	Data 6
	Data 7
	Data 8
Mem_3	Data 9
	Data 10
	Data 11

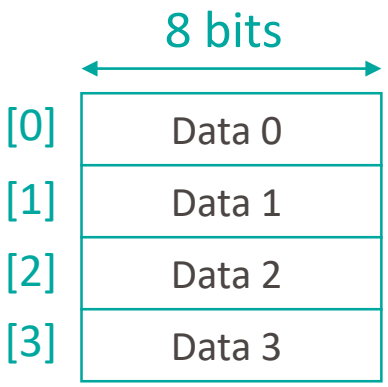
Cyclic factor 4

Mem_0	Data 0	Mem_1	Data 1
	Data 4		Data 5
	Data 8		Data 9
Mem_2	Data 2	Mem_3	Data 3
	Data 6		Data 7
	Data 10		Data 11

Each memory block has a **limited amount of access ports** (single or dual ports), which limits the data that can be obtained in parallel

Array optimizations
Reshaping

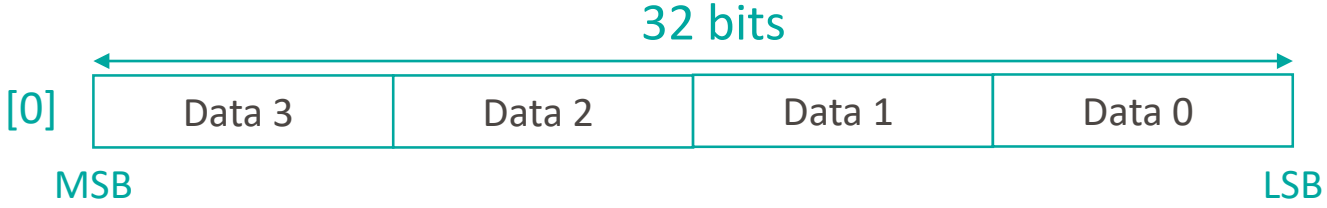
Original data



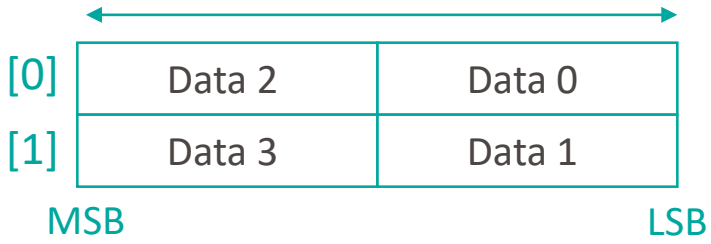
BRAM types
- 4096 x 8 bits
- 2048 x 16 bits
- 1024 x 32 bits

Reshaped data

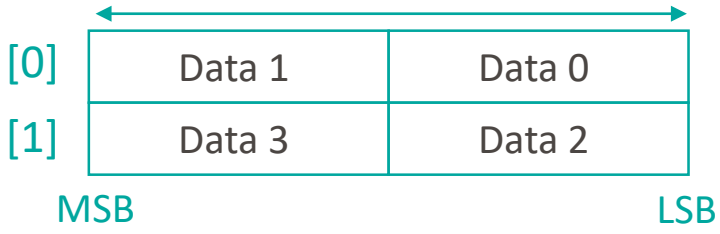
Complete



Block factor 2



Cyclic factor 2



Partitioning and reshaping can be done for each dimension in multidimensional arrays

The ***dim*** field indicates the dimension in which the partition or reshape is applied (dim 0 applies the optimization to all dimensions)

What if the original data is has 16 or 32 bits width?

Combining Loop and Array Optimizations

baseline code

```
ap_uint<8> a[512][8];

loop_1: for(int i = 0; i  < 512; i++) {
   loop_2: for(int j = 0; j  < 8; j++) {
   #pragma HLS unroll factor=4
      acc += a[i] [j];
   }
}
```

Partitioning:

#pragma HLS **array_partition** variable=a
type=? factor=? dim=?

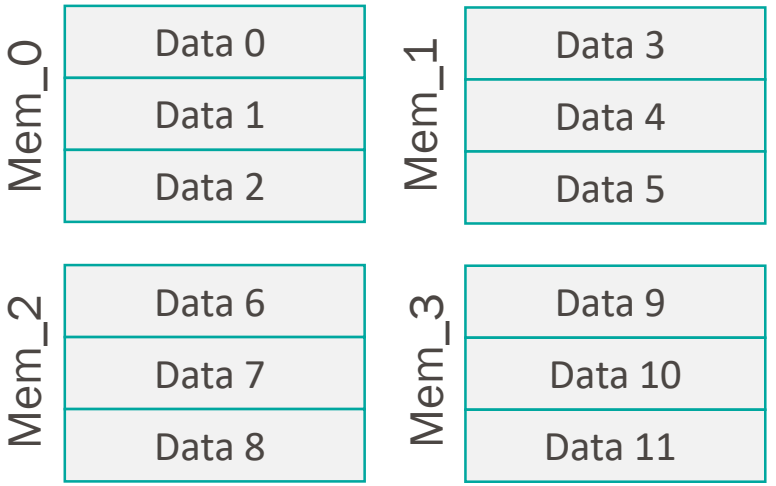
Reshaping:

#pragma HLS **array_reshape** variable=a
type=? factor=? dim=?

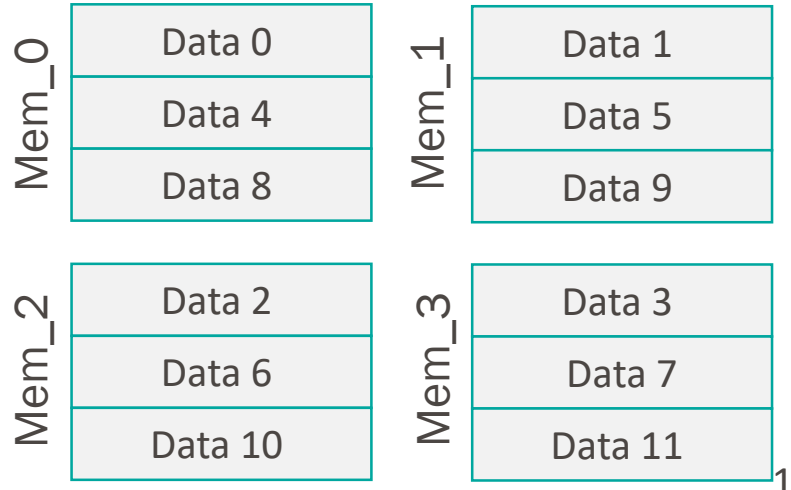
BRAM types

- 4096 x 8 bits
- 2048 x 16 bits
- 1024 x 32 bits

Partition **block** factor 4



Partition **cyclic** factor 4



Combining Loop and Array optimizations

When to use partition and when to use reshape?

Assuming **single port** memories and **1 BRAM** indivisible units

baseline code

```
ap_uint<8> a[512][8];
acc = 0;
loop_1: for(int i = 0; i  < 512; i++) {
   loop_2: for(int j = 0; j  < 8; j++) {
   #pragma HLS unroll factor=4
      acc += a[i] [j];
   }
}
```

Partitioning:

#pragma HLS **array_partition** variable=a type=? factor=? dim=?

Reshaping:

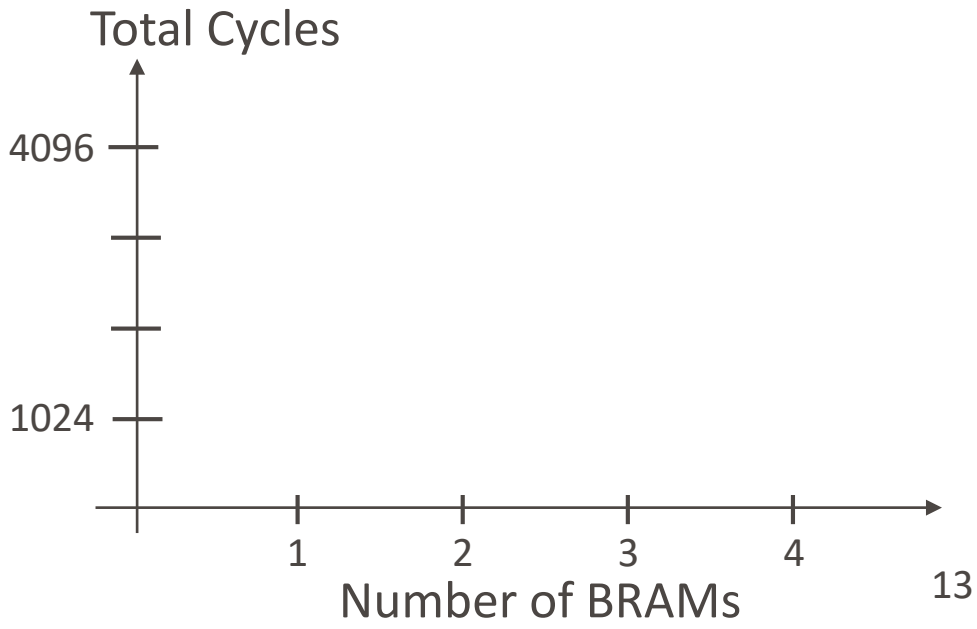
#pragma HLS **array_reshape** variable=a type=cyclic factor=? dim=?

BRAM types

- 4096 x 8 bits
- 2048 x 16 bits
- 1024 x 32 bits

Solution	# BRAMS	Utilization	Cycles
baseline	???		
partition_4	???		
reshape_4	???		

Exercise: can we do better? Tip: #pragma HLS unroll factor=8



Combining Loop and Array optimizations

When to use partition and when to use reshape?

Assuming **dual port** memories OR **0.5 BRAM** indivisible units (realistic)

baseline code

```
ap_uint<8> a[512][8];

loop_1: for(int i = 0; i  < 512; i++) {
   loop_2: for(int j = 0; j  < 8; j++) {
   #pragma HLS unroll factor=4
      acc += a[i] [j];
   }
}
```

Partitioning:

#pragma HLS **array_partition** variable=a type=? factor=? dim=?

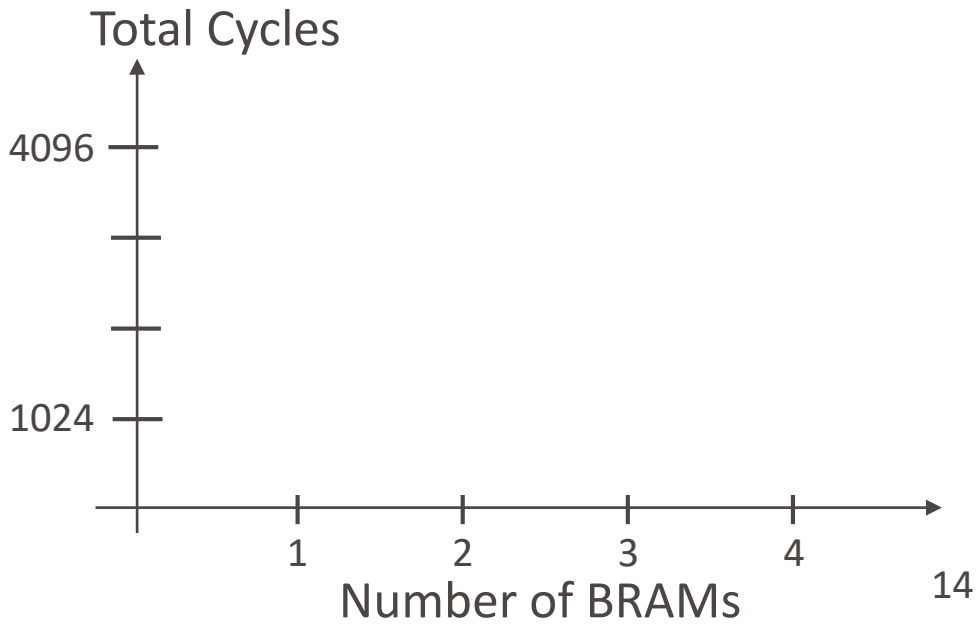
Reshaping:

#pragma HLS **array_reshape** variable=a type=? factor=? dim=?

BRAM types

- 4096 x 8 bits
- 2048 x 16 bits
- 1024 x 32 bits

Solution	# BRAMS	Utilization	Cycles
baseline	???		
partition_4	???		
reshape_4	???		
partition_2	???		
reshape_2	???		



Combining Loop and Array optimizations

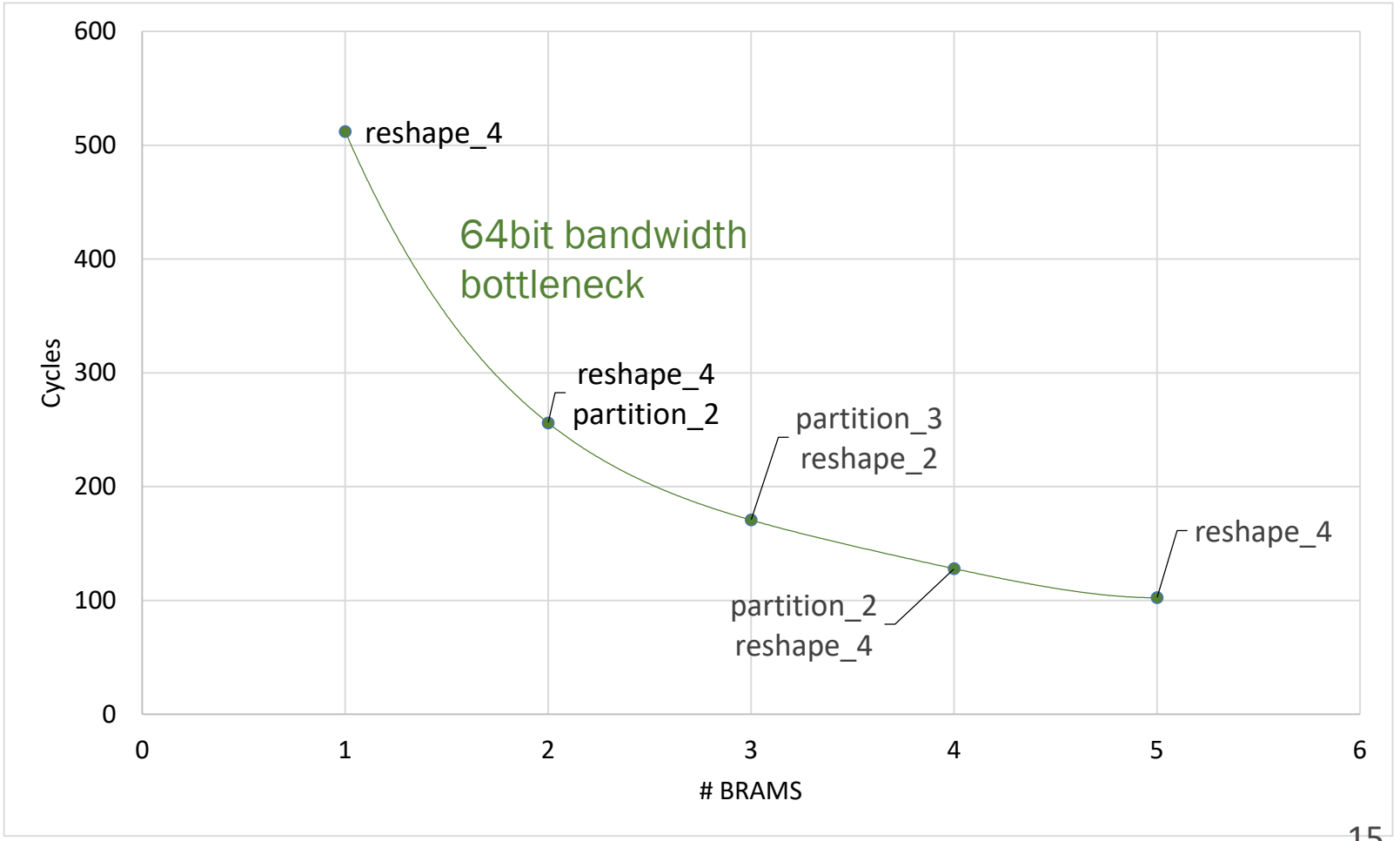
When to use partition and when to use reshape?

Assuming **dual port** memories OR **0.5 BRAM** indivisible units (realistic)

baseline code

```
ap_uint<8> a[512][8];
#pragma HLS array_partition variable=a
type=cyclic factor=? dim=2
#pragma HLS array_reshape variable=a
type=cyclic factor=? dim=2

loop_1: for(int i = 0; i  < 512; i++) {
   loop_2: for(int j = 0; j  < 8; j++) {
   #pragma HLS unroll factor=?
      acc += a[i] [j];
   }
}
```



Combining Loop and Array optimizations

Assuming **dual port** memories OR **0.5 BRAM** indivisible units (realistic)

Example code

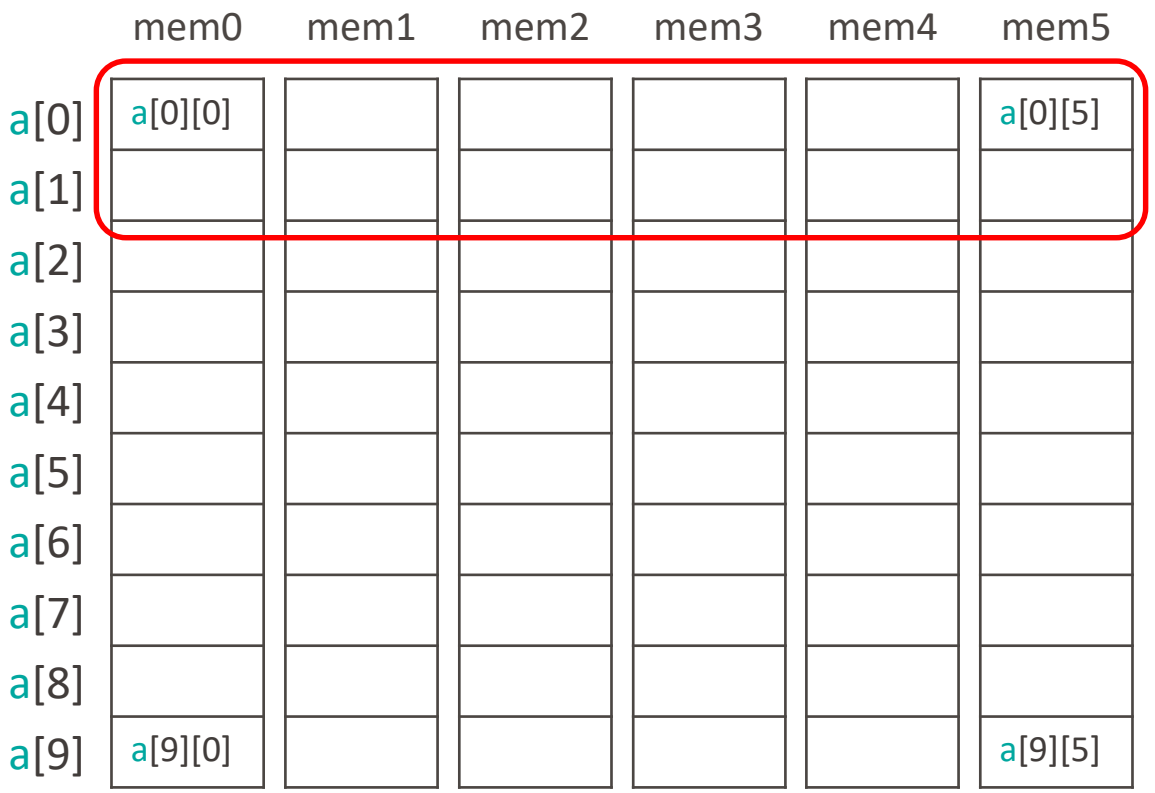
```
ap_uint<8> a[2050][6];

loop_1: for(int i = 0; i  < 2050; i++) {
#pragma HLS unroll factor=2
   loop_2: for(int j = 0; j  < 6; j++) {
   #pragma HLS unroll complete
      acc = a[i] [j];
   }
}
```

Partitioning:

#pragma HLS array_partition variable=a
type=? Factor=? dim=1

#pragma HLS array_partition variable=a
type=? Factor=? dim=2



Data accessed in 1 cycle

BRAM types

- **4096 x 8 bits**
- 2048 x 16 bits
- 1024 x 32 bits

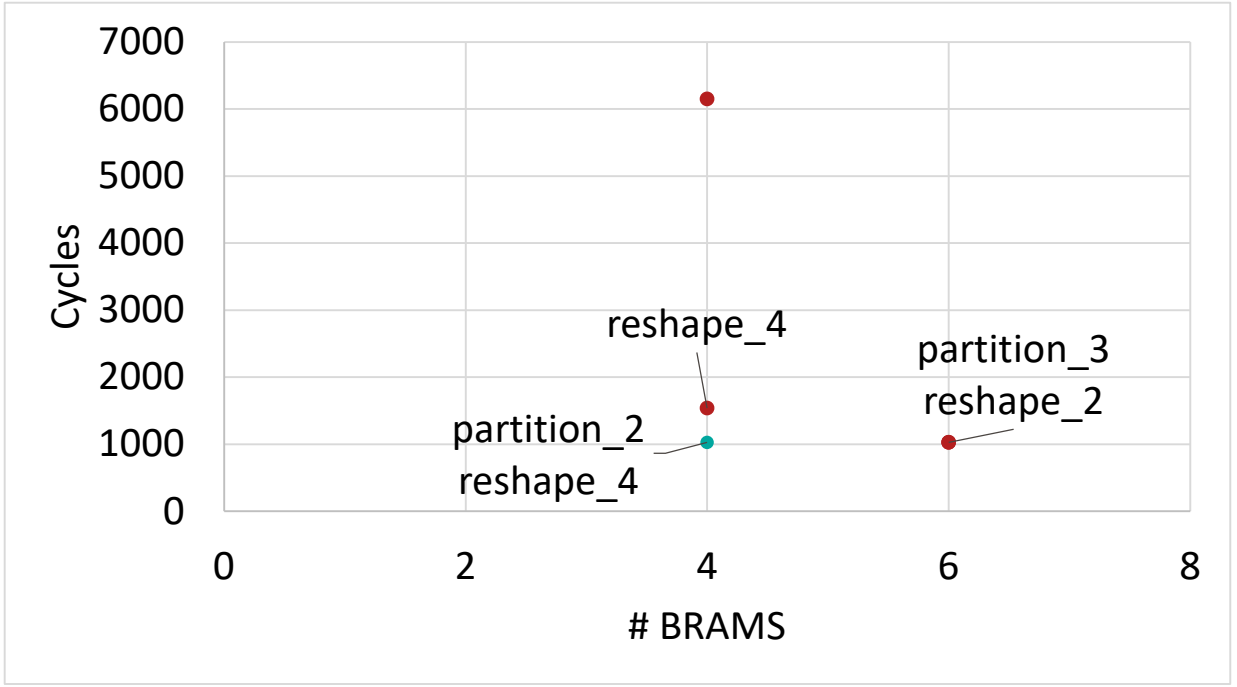
Combining Loop and Array optimizations

Assuming **dual port** memories OR **0.5 BRAM** indivisible units (realistic)

Example code

```
ap_uint<8> a[2050][6];

loop_1: for(int i = 0; i  < 2050; i++) {
#pragma HLS unroll factor=2
   loop_2: for(int j = 0; j  < 6; j++) {
   #pragma HLS unroll complete
      acc = a[i] [j];
   }
}
```



BRAM types
- 4096 x 8 bits
- 2048 x 16 bits
- 1024 x 32 bits

Solution	# BRAMS	Utilization	Cycles
baseline	???		
partition_6	???		
partition_3 reshape_2	???		
partition_2 reshape_4	???		
reshape_4	???		

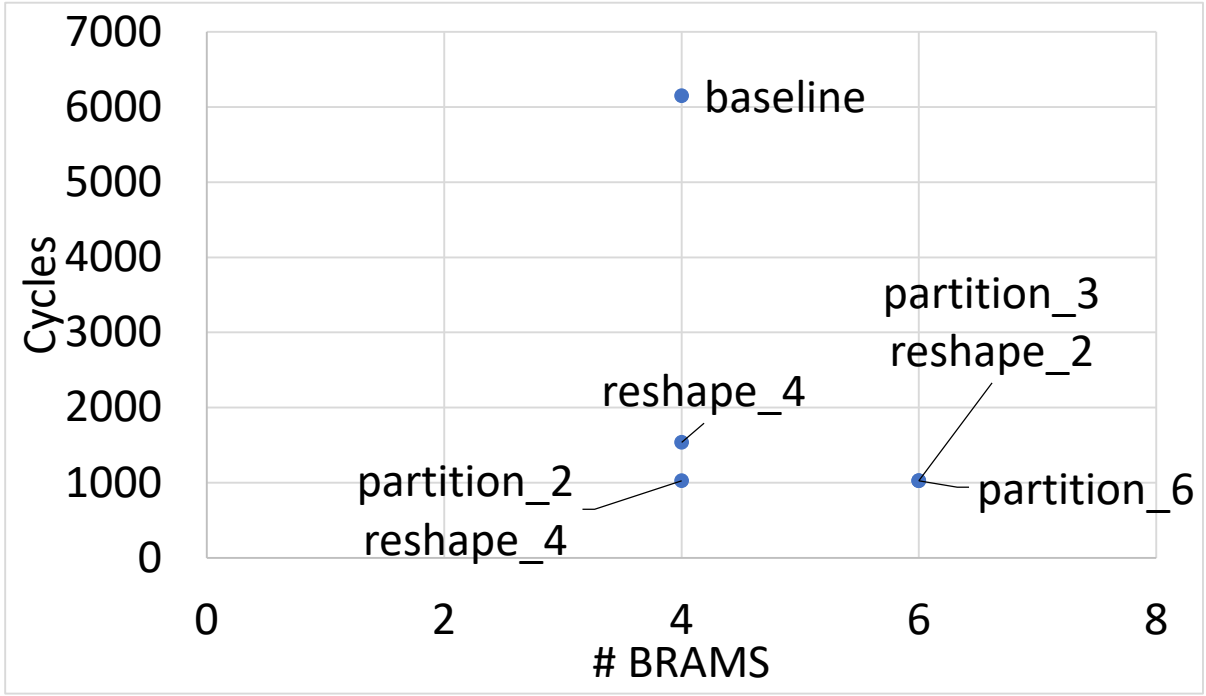
Combining Loop and Array optimizations

Assuming **dual port** memories OR **0.5 BRAM** indivisible units (realistic)

Example code

```
ap_uint<8> a[2050][6];

loop_1: for(int i = 0; i  < 2050; i++) {
#pragma HLS unroll factor=2
   loop_2: for(int j = 0; j  < 6; j++) {
   #pragma HLS unroll complete
      acc = a[i] [j];
   }
}
```



BRAM types
- 4096 x 8 bits
- 2048 x 16 bits
- 1024 x 32 bits

Solution	# BRAMS	Utilization	Cycles
baseline	4 (8 bits)	85,798%	6150
partition_6	6 (8 bits)	50,005%	1025
partition_3 reshape_2	6 (16 bits)	50,005%	1025
partition_2 reshape_4	4 (32 bits)	75.073%	1025
reshape_4	4 (32 bits)	85,798%	1538

Array optimizations: Type of interfaces

Interfaces:

- Slave interfaces (slave registers)
- Master interfaces
  - Master AXI
  - BRAM (single or dual port)
  - FIFOs

```
My_module(int *in_a,
          int *in_b,
          int *out) {
#pragma interface m_axi variable=in_a
#pragma interface m_axi variable=in_b
#pragma interface m_axi variable=out
```

How many ports are generated?

C-argument type	Paradigm	Interface protocol (I/O/Inout)
Scalar(pass by value)	Slave register	s_axilite (Write only)
Pointer used as scalar	Slave register	s_axilite (Read and/or Write)
Reference	Slave register	s_axilite (Read and/or Write)
Array	Master port	m_axi
Pointer used as array	Master port	m_axi
hls::stream	Stream	AXI4-Stream (axis)

Other Interfaces:

- None
- Valid
- Ack
- Valid & ack
- Output valid
- Control chain (Start, continue, idle, done, ready)

Array optimizations: Type of interfaces

Interfaces:

- Slave interfaces (slave registers)
- Master interfaces
  - Master AXI
  - BRAM (single or dual port)
  - FIFOs

```
My_module(int *in_a,
          int *in_b,
          int *out) {
#pragma interface bram variable=in_a
#pragma interface bram variable=in_b
#pragma interface bram variable=out
```

What if I access true dual-port BRAMs?

C-argument type	Paradigm	Interface protocol (I/O/Inout)
Scalar(pass by value)	Slave register	s_axilite (Write only)
Pointer used as scalar	Slave register	s_axilite (Read and/or Write)
Reference	Slave register	s_axilite (Read and/or Write)
Array	Master port	m_axi
Pointer used as array	Master port	m_axi
hls::stream	Stream	AXI4-Stream (axis)

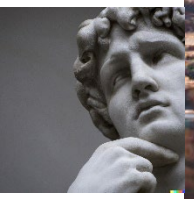
Other Interfaces:

- None
- Valid
- Ack
- Valid & ack
- Output valid
- Control chain (Start, continue, idle, done, ready)

Homework: check preconditions and limitations of burst transfer here

Summary

- Types of storage resources and how to optimize their usage
- How to perform a **Design Space Exploration** in HLS using:
  - **Array optimizations** pragmas: partitioning, reshaping.
  - Combining loop and array optimizations
- Types of interfaces
  - Slave
  - Master
  - Other protocols



Questions?

Denisa Constantinescu

EPFL – Embedded Systems Laboratory
denisa.constantinescu@epfl.ch