POLITECNICO
MILANO 1863

Iris: Automatic Generation of Efficient Data Layouts for High Bandwidth Utilization

Stephanie Soldavini, Donatella Sciuto, Christian Pilato
Dipartimento di Elettronica, Informazione e Bioingegneria
stephanie.soldavini@polimi.it

ASPDAC'23 – January 17, 2023

Motivation

- Optimizing **data movements** is one of the biggest challenges in heterogeneous computing to cope with modern **big data applications**
- High-level synthesis (HLS) tools are increasingly efficient at optimizing computation, but data transfers have not been adequately improved
- Novel architectures such as **high-bandwidth memory (HBM)** have been developed to be able to transfer more data in parallel
  - DDR5 has 2 channels of 32 bits/channel
  - HBM can have e.g. 32 channels of 256 bits/channel
- However, designers must follow strict coding-style rules to exploit this extra bandwidth

Iris

- We designed a method ("Iris"[1]) for efficiently transferring **arbitrarily sized data** from global memory to an accelerator
- Individual arrays of data are packed into global memory as one unified block, to make **one large transfer** instead of many small transfers
- Inspired by processor scheduling, where the data arrays are treated as preemptible tasks and the goal is to optimize such that the data arrives to their relevant processing units as soon as possible

[1] In Greek mythology, Iris is the messenger of the gods

Example

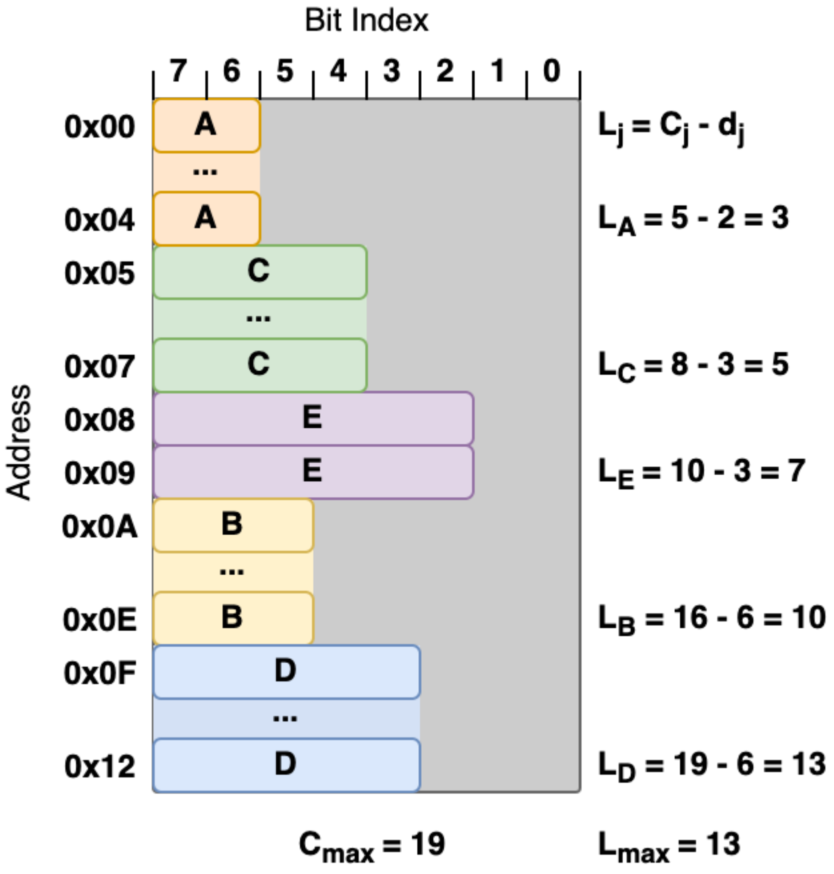
Array	Width	Depth	Due Date
A	2	5	2
B	3	5	6
C	4	3	3
D	5	4	6
E	6	2	3

Example

Array	Width	Depth	Due Date ▼
A	2	5	2
C	4	3	3
E	6	2	3
B	3	5	6
D	5	4	6

Example

Array	Width	Depth	Due Date ▼
A	2	5	2
C	4	3	3
E	6	2	3
B	3	5	6
D	5	4	6



$L_j = C_j - d_j$

$L_A = 5 - 2 = 3$

$L_C = 8 - 3 = 5$

$L_E = 10 - 3 = 7$

$L_B = 16 - 6 = 10$

$L_D = 19 - 6 = 13$

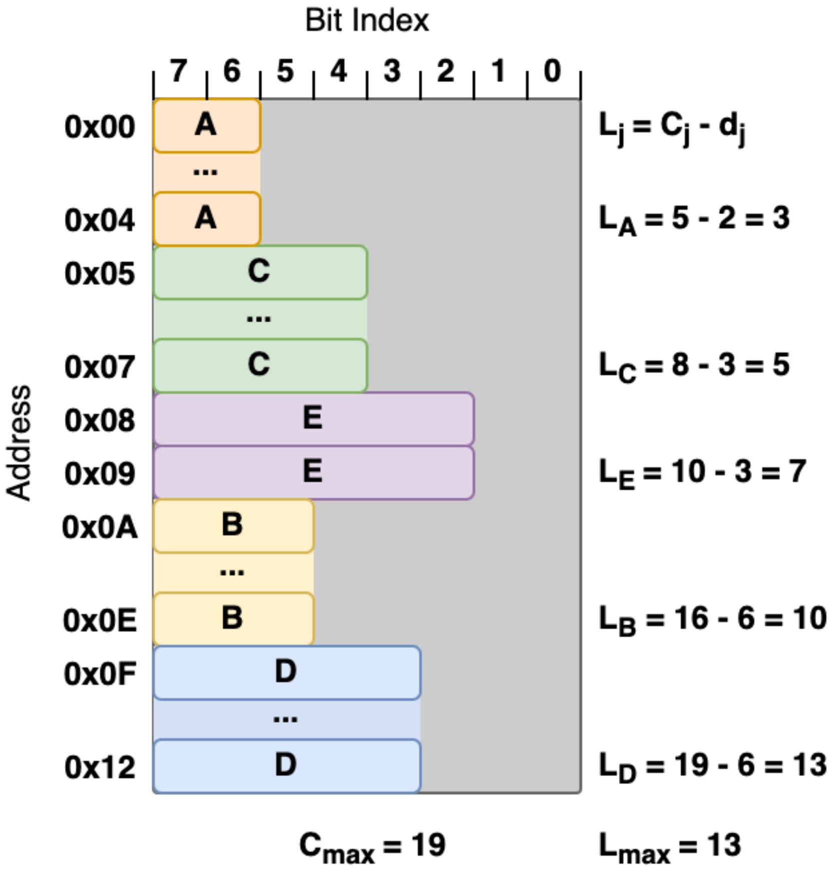
$C_{max} = 19$

$L_{max} = 13$

Fig. 1: Naive layout

Example

Array	Width	Depth	Due Date ▼
A	2	5	2
C	4	3	3
E	6	2	3
B	3	5	6
D	5	4	6



$L_j = C_j - d_j$

$L_A = 5 - 2 = 3$

$L_C = 8 - 3 = 5$

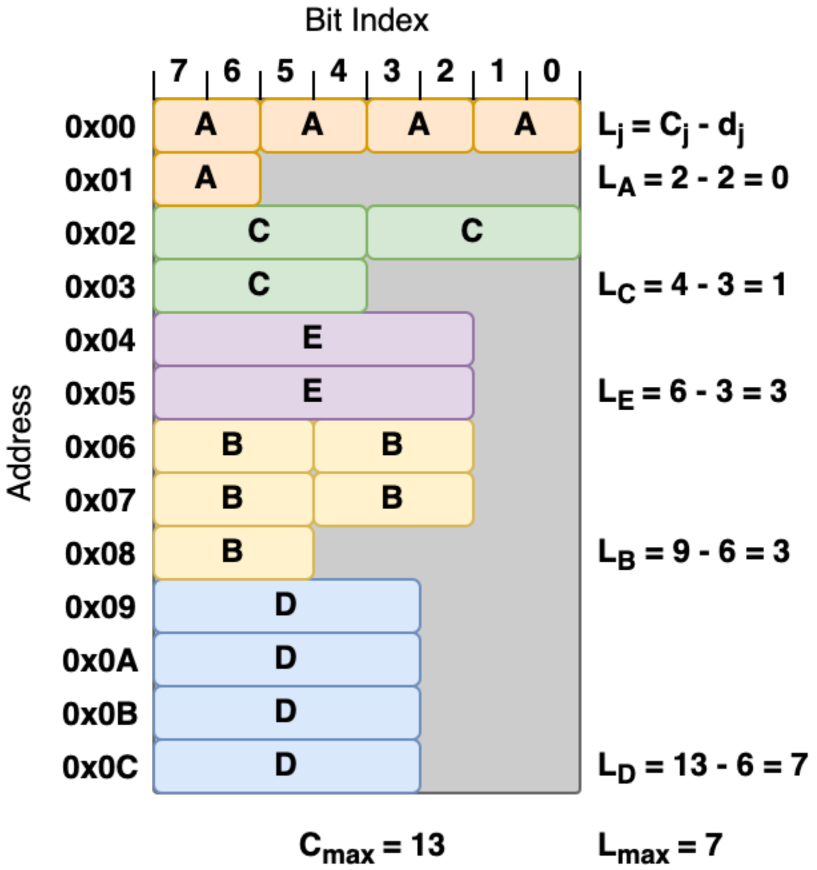
$L_E = 10 - 3 = 7$

$L_B = 16 - 6 = 10$

$L_D = 19 - 6 = 13$

$C_{max} = 19$ $L_{max} = 13$

Fig. 1: Naive layout



$L_j = C_j - d_j$

$L_A = 2 - 2 = 0$

$L_C = 4 - 3 = 1$

$L_E = 6 - 3 = 3$

$L_B = 9 - 6 = 3$

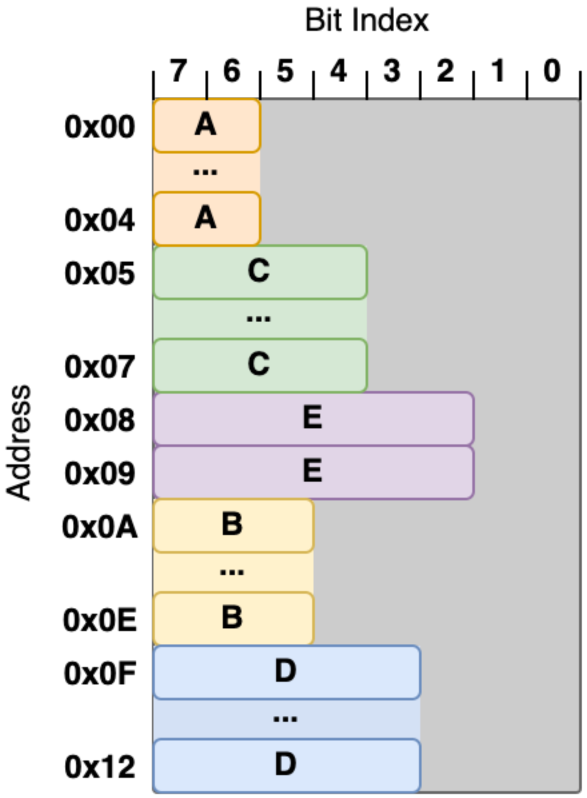
$L_D = 13 - 6 = 7$

$C_{max} = 13$ $L_{max} = 7$

Fig. 2: Naively packed layout

Example

Array	Width	Depth	Due Date ▼
A	2	5	2
C	4	3	3
E	6	2	3
B	3	5	6
D	5	4	6



$L_j = C_j - d_j$

$L_A = 5 - 2 = 3$

$L_C = 8 - 3 = 5$

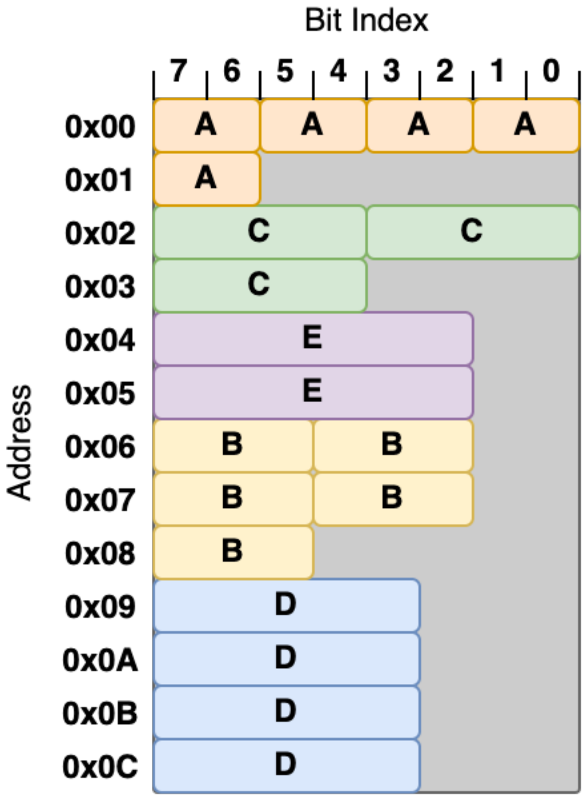
$L_E = 10 - 3 = 7$

$L_B = 16 - 6 = 10$

$L_D = 19 - 6 = 13$

$C_{max} = 19$ $L_{max} = 13$

Fig. 1: Naive layout



$L_j = C_j - d_j$

$L_A = 2 - 2 = 0$

$L_C = 4 - 3 = 1$

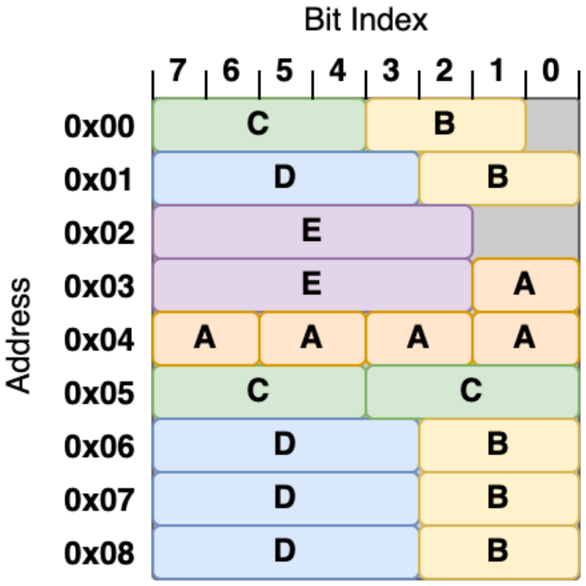
$L_E = 6 - 3 = 3$

$L_B = 9 - 6 = 3$

$L_D = 13 - 6 = 7$

$C_{max} = 13$ $L_{max} = 7$

Fig. 2: Naively packed layout



$L_j = C_j - d_j$

$L_E = 4 - 3 = 1$

$L_A = 5 - 2 = 3$

$L_C = 6 - 3 = 3$

$L_B = 9 - 6 = 3$

$L_D = 9 - 6 = 3$

$C_{max} = 9$ $L_{max} = 3$

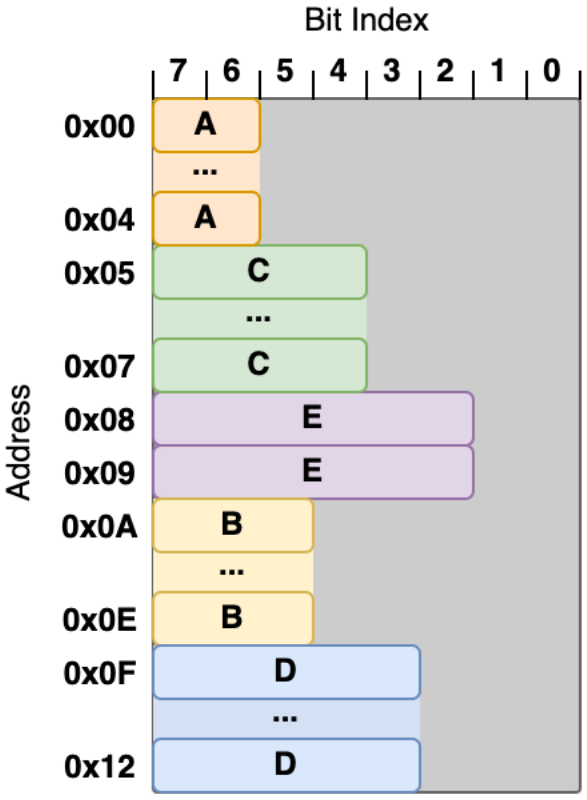
Fig. 3: Iris layout

Example

Array	Width	Depth	Due Date ▼
A	2	5	2
C	4	3	3
E	6	2	3
B	3	5	6
D	5	4	6

Table 1: Decode logic HLS estimates

Strategy	Latency	FF	LUT
Naively packed	43	54	452
Iris layout	11	29	194
% Reduction	74%	46%	57%



$L_j = C_j - d_j$

$L_A = 5 - 2 = 3$

$L_C = 8 - 3 = 5$

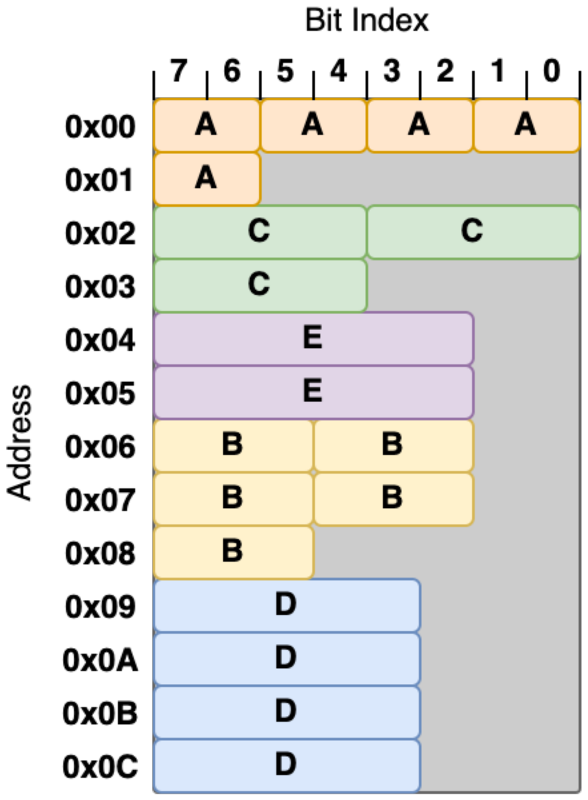
$L_E = 10 - 3 = 7$

$L_B = 16 - 6 = 10$

$L_D = 19 - 6 = 13$

$C_{max} = 19$ $L_{max} = 13$

Fig. 1: Naive layout



$L_j = C_j - d_j$

$L_A = 2 - 2 = 0$

$L_C = 4 - 3 = 1$

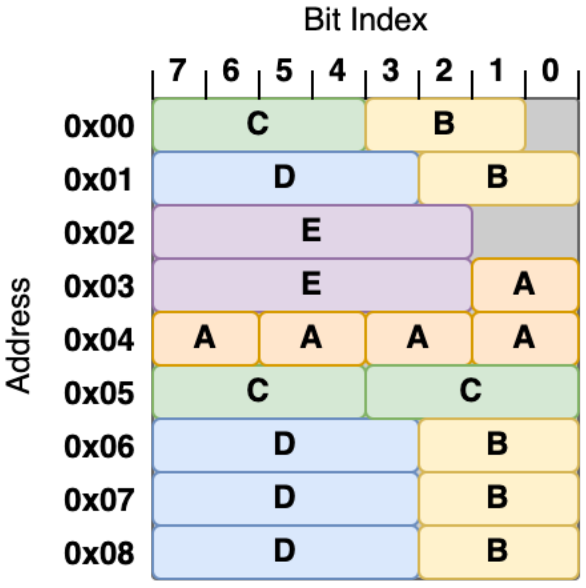
$L_E = 6 - 3 = 3$

$L_B = 9 - 6 = 3$

$L_D = 13 - 6 = 7$

$C_{max} = 13$ $L_{max} = 7$

Fig. 2: Naively packed layout



$L_j = C_j - d_j$

$L_E = 4 - 3 = 1$

$L_A = 5 - 2 = 3$

$L_C = 6 - 3 = 3$

$L_B = 9 - 6 = 3$

$L_D = 9 - 6 = 3$

$C_{max} = 9$ $L_{max} = 3$

Fig. 3: Iris layout

Problem Formulation as Scheduling

Problem Formulation as Scheduling

Memory Layout Problem

Problem Formulation as Scheduling

Memory Layout Problem

- Given:
  - Bus width (m)
  - A set of accelerator arrays, each with:
    - Bitwidth (W_j) and Depth (D_j)
    - Desired due date (d_j)

Problem Formulation as Scheduling

Memory Layout Problem

- Given:
  - Bus width (m)
  - A set of accelerator arrays, each with:
    - Bitwidth (W_j) and Depth (D_j)
    - Desired due date (d_j)
- Want a memory layout where:
  - Data are packed **most densely**
  - Arrays **arrive as close to their due dates** as possible when transferred from memory to accelerator

Problem Formulation as Scheduling

Memory Layout Problem → Scheduling Problem

- Given:
  - Bus width (m)
  - A set of accelerator arrays, each with:
    - Bitwidth (W_j) and Depth (D_j)
    - Desired due date (d_j)
- Want a memory layout where:
  - Data are packed **most densely**
  - Arrays **arrive as close to their due dates** as possible when transferred from memory to accelerator

Problem Formulation as Scheduling

Memory Layout Problem

- Given:
  - Bus width (m)
  - A set of accelerator arrays, each with:
    - Bitwidth (W_j) and Depth (D_j)
    - Desired due date (d_j)
- Want a memory layout where:
  - Data are packed **most densely**
  - Arrays **arrive as close to their due dates** as possible when transferred from memory to accelerator

Scheduling Problem

- Given:
  - m identical processors
  - A set of preemptible tasks, each with:
    - Processing time ($W_j \times D_j$)
    - Desired due date (d_j)

Problem Formulation as Scheduling

Memory Layout Problem

- Given:
  - Bus width (m)
  - A set of accelerator arrays, each with:
    - Bitwidth (W_j) and Depth (D_j)
    - Desired due date (d_j)
- Want a memory layout where:
  - Data are packed **most densely**
  - Arrays **arrive as close to their due dates** as possible when transferred from memory to accelerator

Scheduling Problem

- Given:
  - m identical processors
  - A set of preemptible tasks, each with:
    - Processing time ($W_j \times D_j$)
    - Desired due date (d_j)
- Want a schedule where:
  - Processors are **maximally used**
  - Tasks complete **as close to their due dates** as possible

Isomorphic Scheduling Problem

Given m identical processors, we want to schedule preemptible tasks with release time r_j across several processors to minimize the total schedule length (C_{max}) [1]

---

[1] Maciej Drozdowski. “Real-time scheduling of linear speedup parallel tasks”. In: Information Processing Letters 57.1 (1996), pp. 35–40.

Isomorphic Scheduling Problem

Given m identical processors, we want to schedule preemptible tasks with release time r_j across several processors to minimize the total schedule length (C_{max}) [1]

- The release time r_j is time step when a task j is ready to begin execution

[1] Maciej Drozdowski. “Real-time scheduling of linear speedup parallel tasks”. In: Information Processing Letters 57.1 (1996), pp. 35–40.

Isomorphic Scheduling Problem

Given m identical processors, we want to schedule preemptible tasks with release time r_j across several processors to minimize the total schedule length (C_{max}) [1]

- The release time r_j is time step when a task j is ready to begin execution
- To convert between the two problems, each due date d_j is converted to a release time r_j by subtracting it from the maximum (latest) due date: $r_j = d_{max} - d_j$

---

[1] Maciej Drozdowski. “Real-time scheduling of linear speedup parallel tasks”. In: Information Processing Letters 57.1 (1996), pp. 35–40.

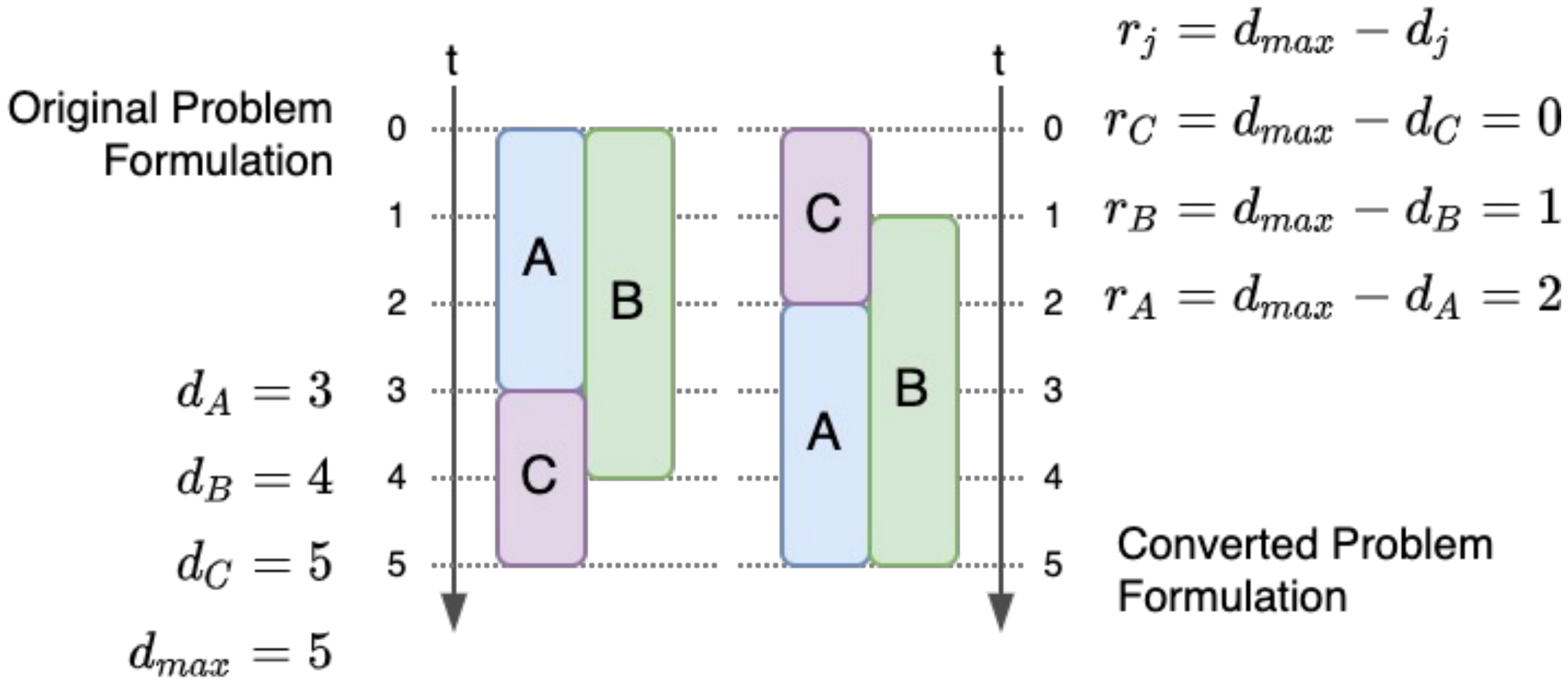
Isomorphic Scheduling Problem

Given m identical processors, we want to schedule preemptible tasks with release time r_j across several processors to minimize the total schedule length (C_{max}) [1]

- The release time r_j is time step when a task j is ready to begin execution
- To convert between the two problems, each due date d_j is converted to a release time r_j by subtracting it from the maximum (latest) due date: $r_j = d_{max} - d_j$
- The solution to the isomorphic problem is read backwards to obtain the solution to the original memory layout problem

[1] Maciej Drozdowski. “Real-time scheduling of linear speedup parallel tasks”. In: Information Processing Letters 57.1 (1996), pp. 35–40.

Isomorphic Problem: Transformation



$d_A = 3$

$d_B = 4$

$d_C = 5$

$d_{max} = 5$

$r_j = d_{max} - d_j$

$r_C = d_{max} - d_C = 0$

$r_B = d_{max} - d_B = 1$

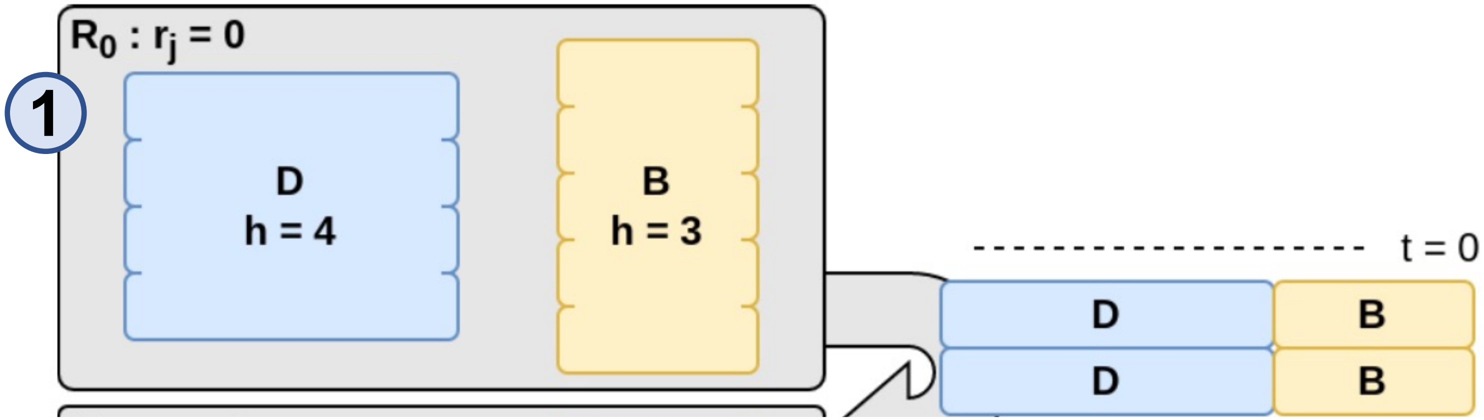
$r_A = d_{max} - d_A = 2$

Sample schedule showing conversion between due dates and release times

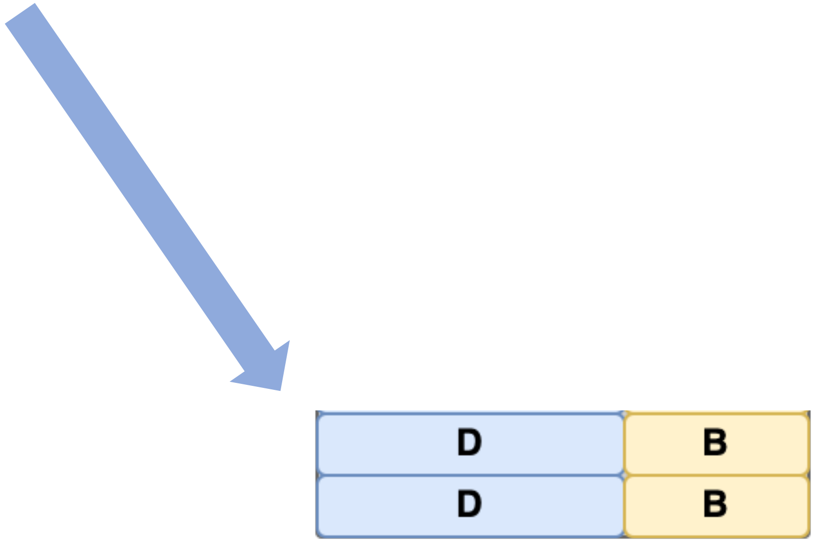
Algorithm Example

Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4

Algorithm Example



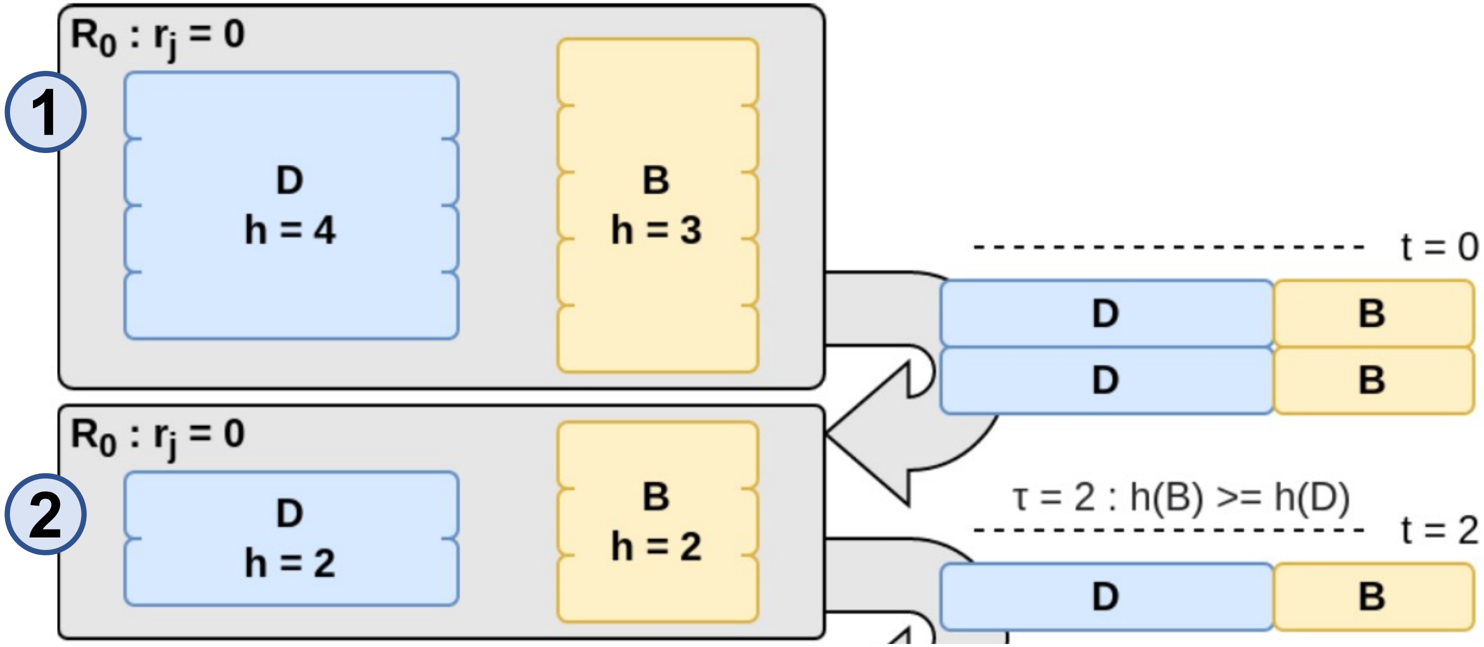
Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4

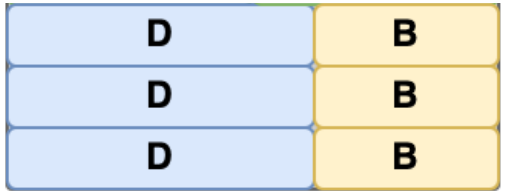


Schedule
(Building Backwards)

Algorithm Example

Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4

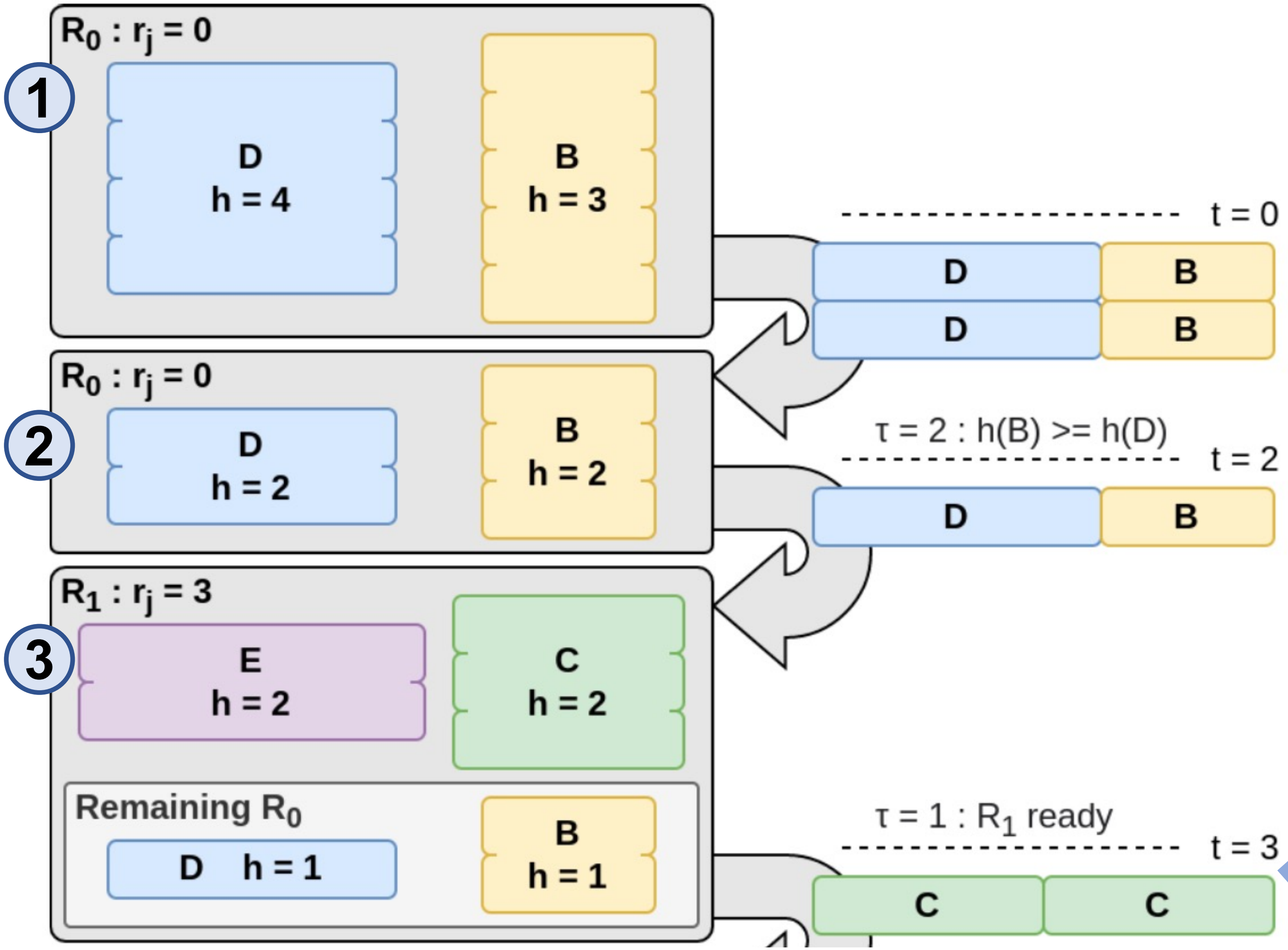


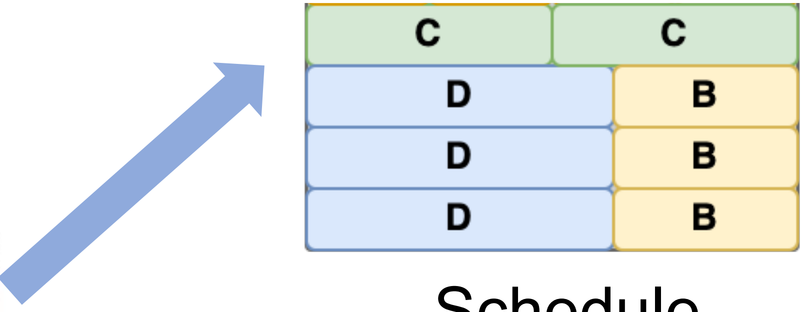


Schedule
(Building Backwards)

Algorithm Example

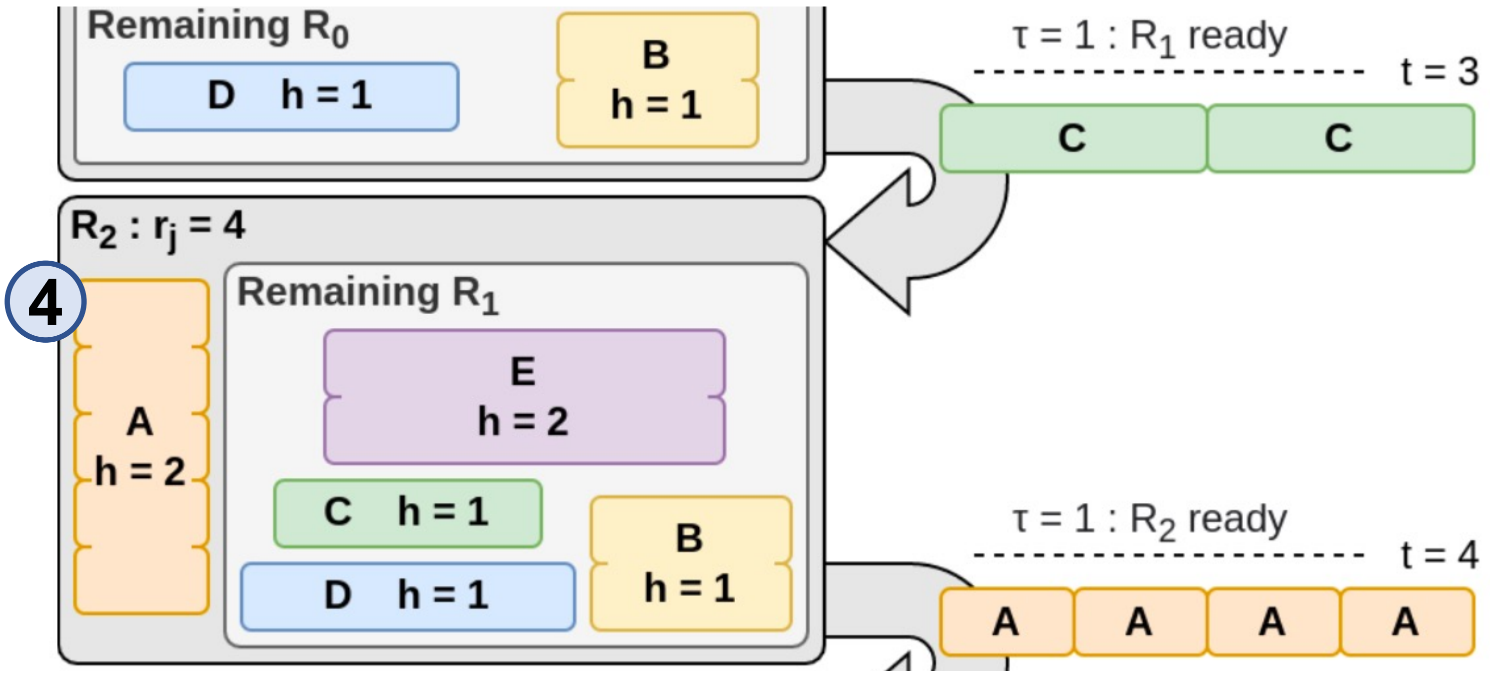
Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4



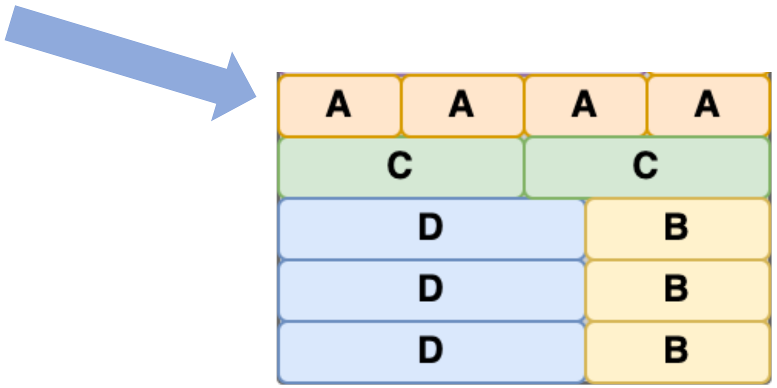


Schedule
(Building Backwards)

Algorithm Example

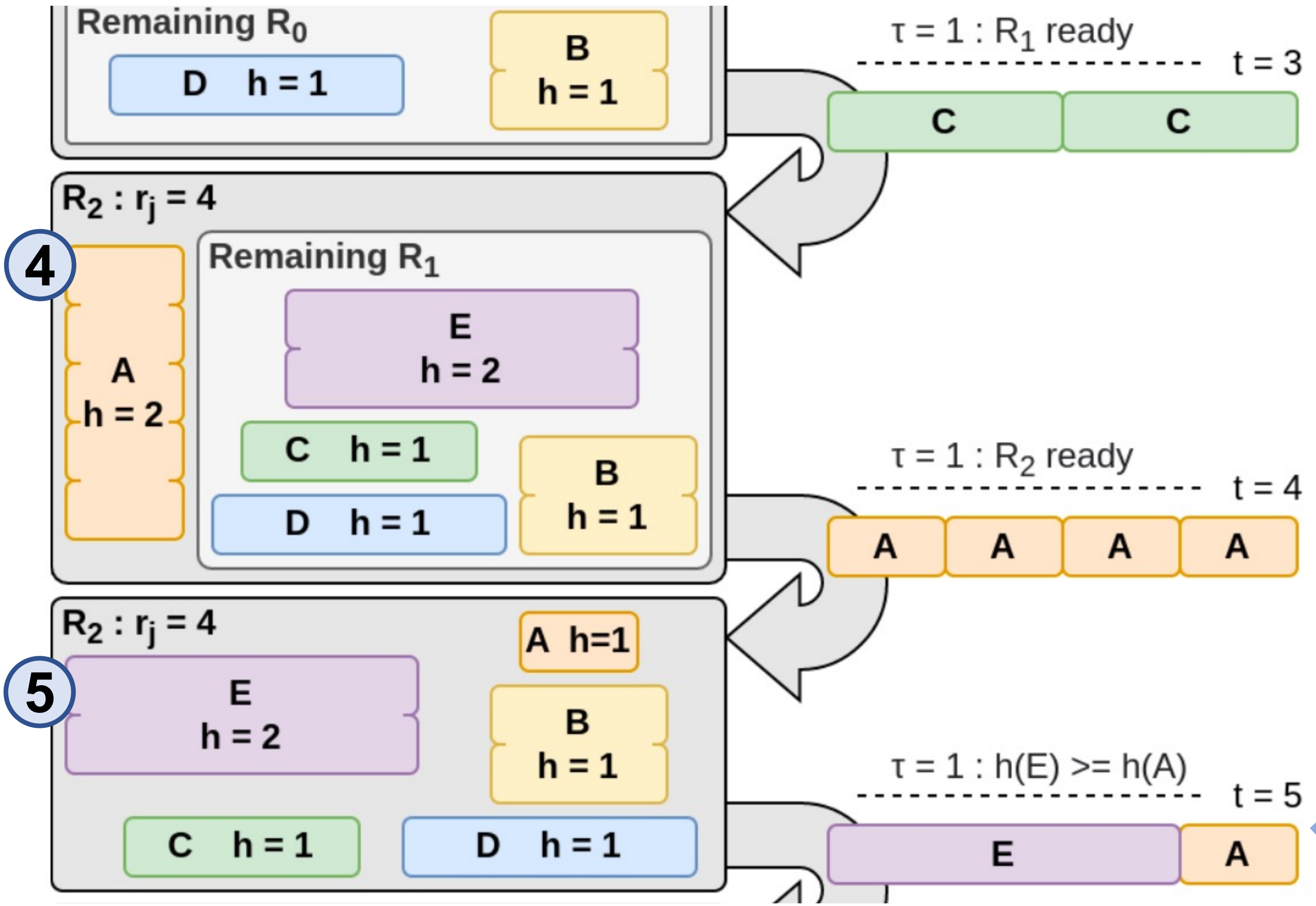


Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4

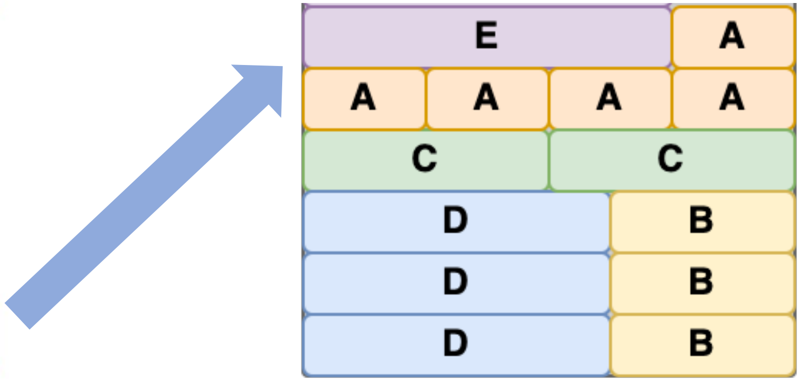


Schedule
(Building Backwards)

Algorithm Example



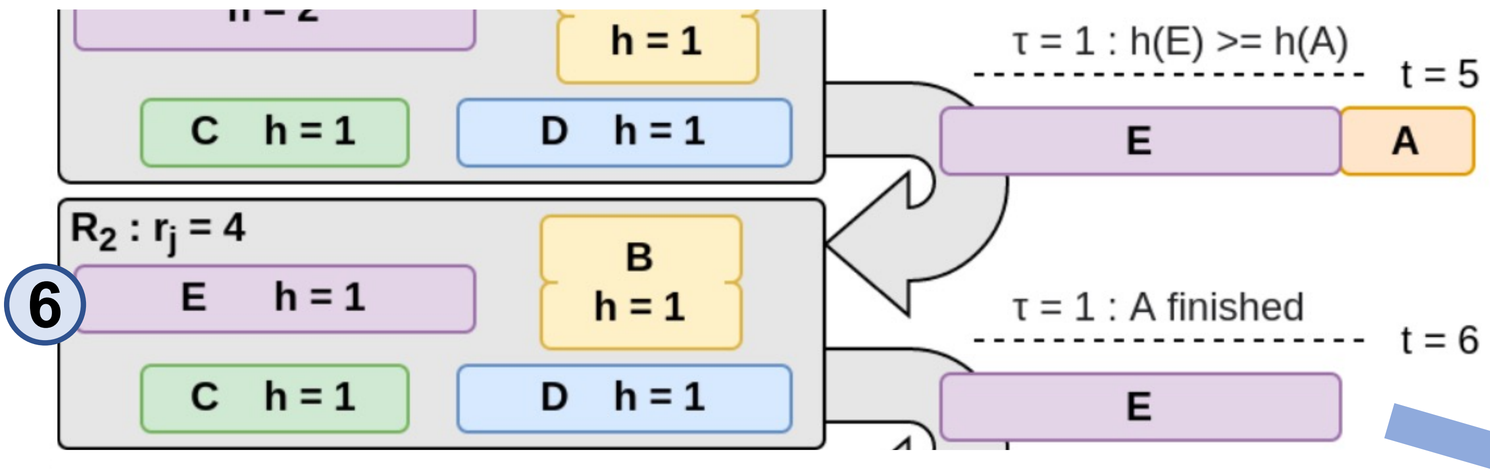
Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4

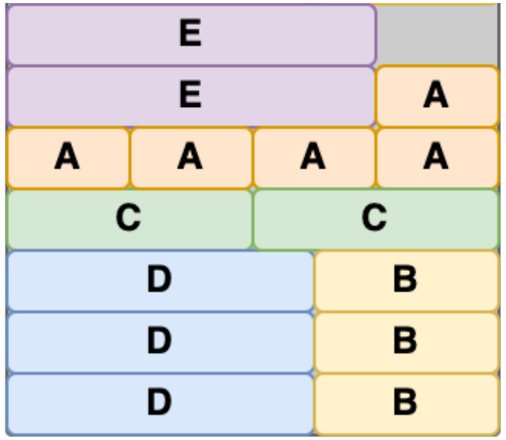


Schedule
(Building Backwards)

Algorithm Example

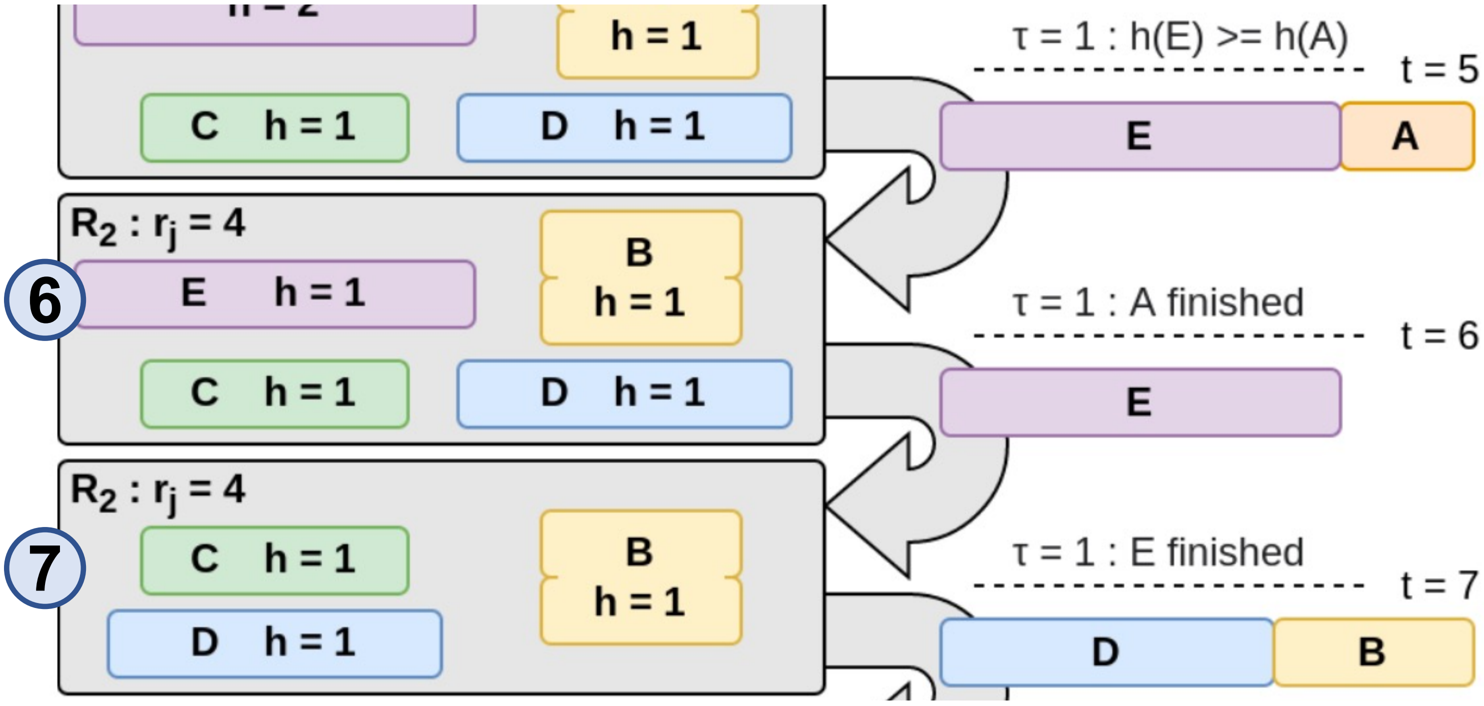
Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4



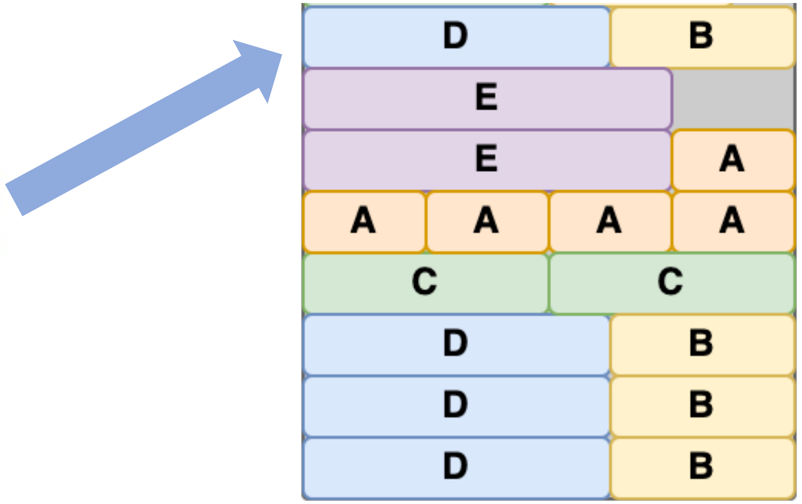


Schedule
(Building Backwards)

Algorithm Example



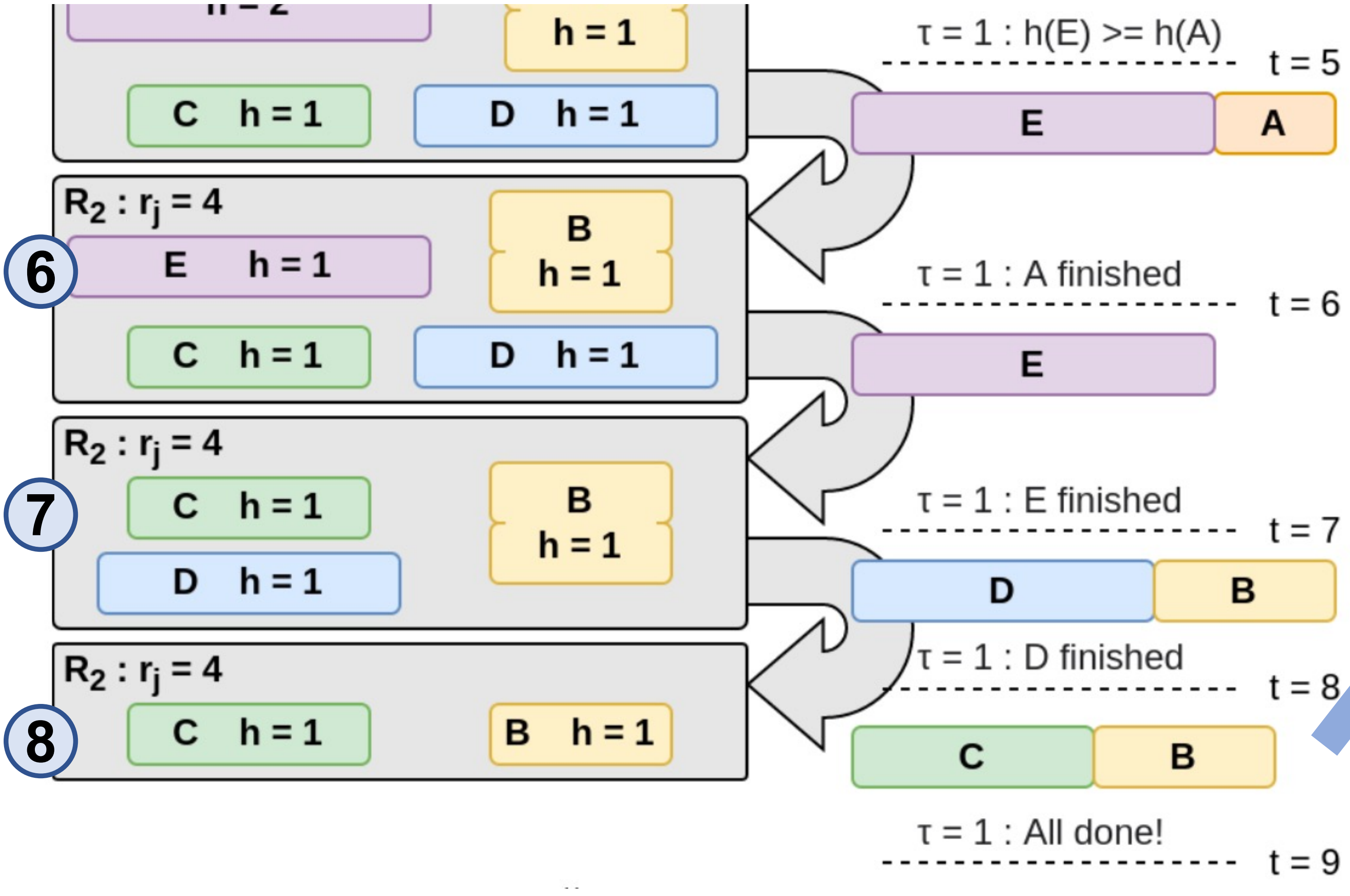
Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4

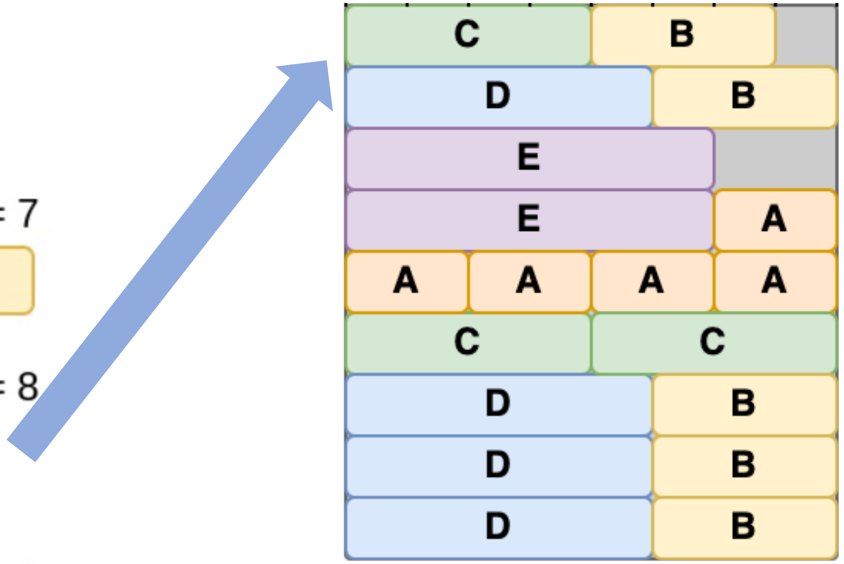


Schedule
(Building Backwards)

Algorithm Example

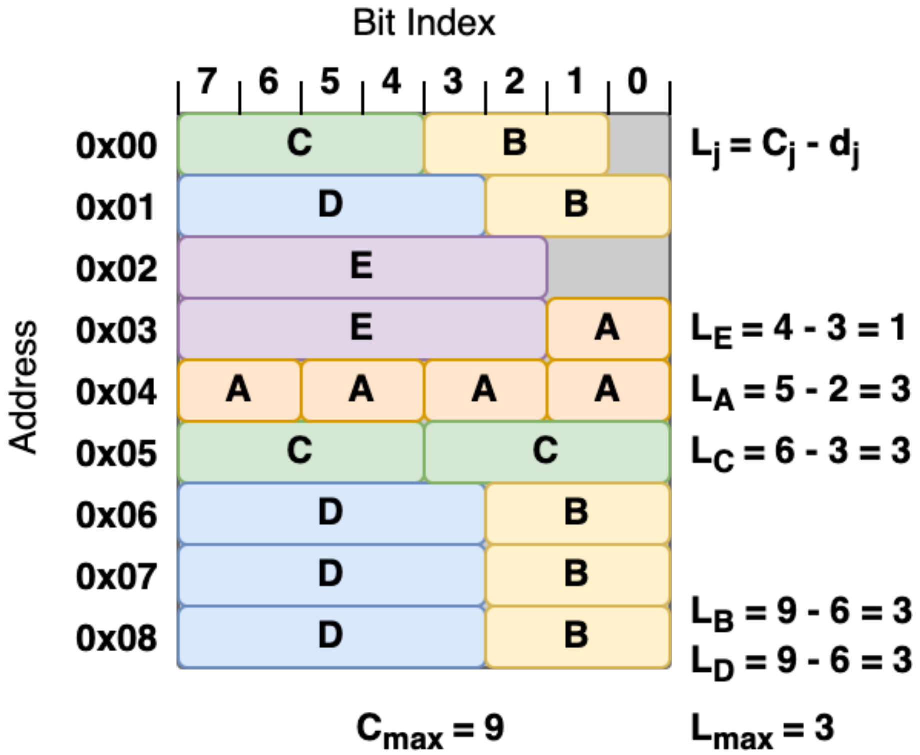
Array	Width	Depth	Due Date	Release Time
D	5	4	6	0
B	3	5	6	0
E	6	2	3	3
C	4	3	3	3
A	2	5	2	4





Schedule
(Building Backwards)

Algorithm Example



Bit Index

Address	7	6	5	4	3	2	1	0
0x00	C	C	C	C	B	B	B	
0x01	D	D	D	D	D	B	B	B
0x02	E	E	E	E	E	E		
0x03	E	E	E	E	E	E	A	A
0x04	A	A	A	A	A	A	A	A
0x05	C	C	C	C	C	C	C	C
0x06	D	D	D	D	D	B	B	B
0x07	D	D	D	D	D	B	B	B
0x08	D	D	D	D	D	B	B	B

$L_j = C_j - d_j$

$L_E = 4 - 3 = 1$

$L_A = 5 - 2 = 3$

$L_C = 6 - 3 = 3$

$L_B = 9 - 6 = 3$

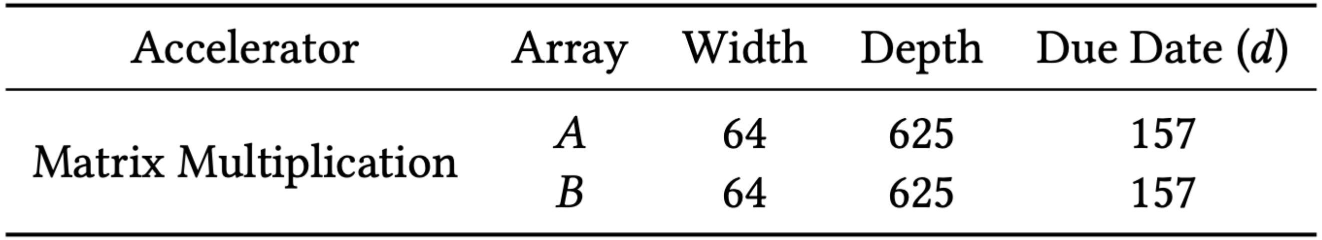
$L_D = 9 - 6 = 3$

$C_{max} = 9$

$L_{max} = 3$

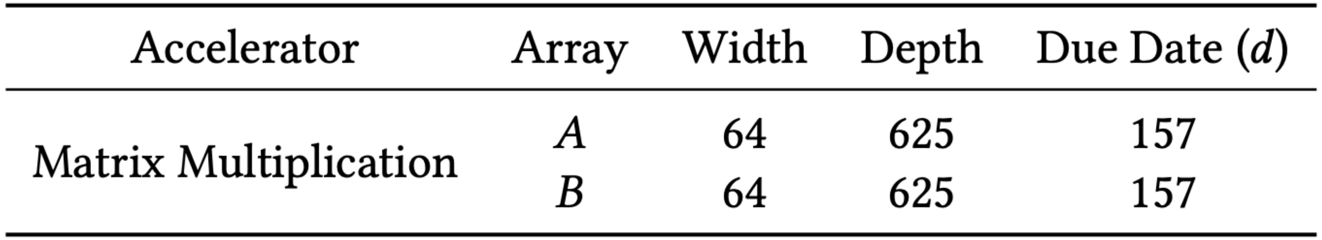
Finished Schedule /
Memory Layout

Matrix Multiply



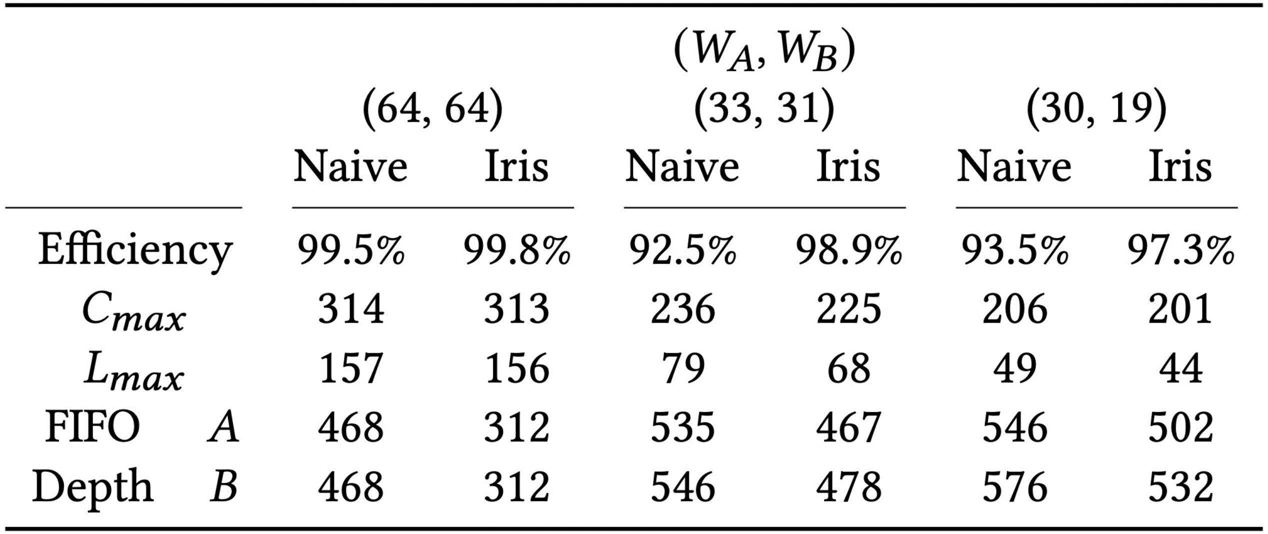
Accelerator	Array	Width	Depth	Due Date (*d*)
Matrix Multiplication	*A*	64	625	157
	B	64	625	157

Matrix Multiply



Accelerator	Array	Width	Depth	Due Date (d)
Matrix Multiplication	A	64	625	157
	B	64	625	157

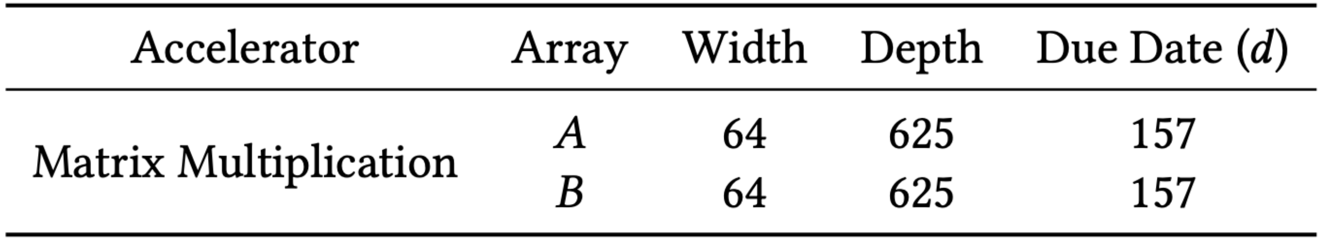
Layout metrics with buswidth of 256 and varied Array Width (W) (Matrix Multiply)



		(W_A, W_B)					
		(64, 64)		(33, 31)		(30, 19)	
		Naive	Iris	Naive	Iris	Naive	Iris
Efficiency		99.5%	99.8%	92.5%	98.9%	93.5%	97.3%
C_{max}		314	313	236	225	206	201
L_{max}		157	156	79	68	49	44
FIFO	A	468	312	535	467	546	502
Depth	B	468	312	546	478	576	532

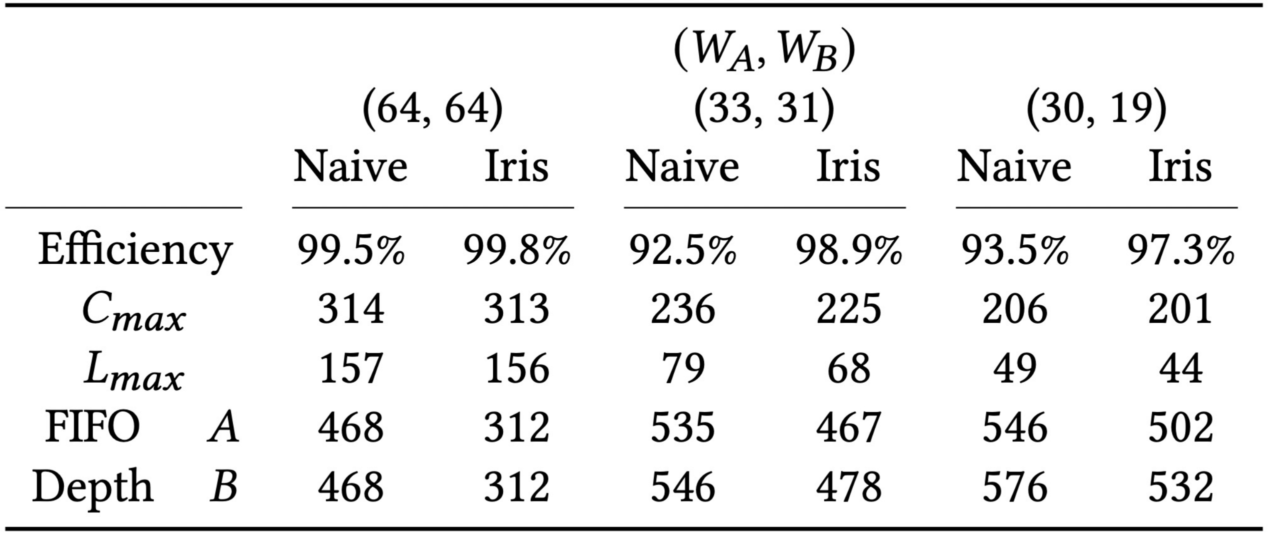
- Unconventional bitwidths are used to save time and area in applications such as neural networks

Matrix Multiply



Accelerator	Array	Width	Depth	Due Date (d)
Matrix Multiplication	A	64	625	157
	B	64	625	157

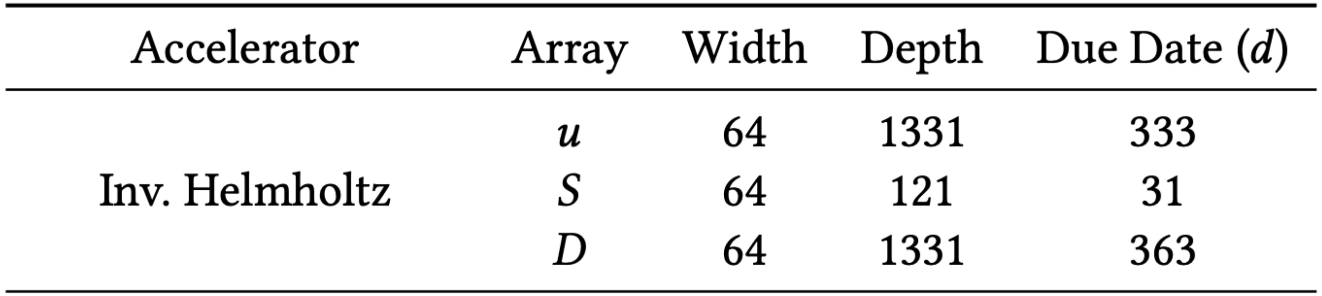
Layout metrics with buswidth of 256 and varied Array Width (W) (Matrix Multiply)



		(W_A, W_B)					
		(64, 64)		(33, 31)		(30, 19)	
		Naive	Iris	Naive	Iris	Naive	Iris
Efficiency		99.5%	99.8%	92.5%	98.9%	93.5%	97.3%
C_{max}		314	313	236	225	206	201
L_{max}		157	156	79	68	49	44
FIFO Depth	A	468	312	535	467	546	502
	B	468	312	546	478	576	532

- Unconventional bitwidths are used to save time and area in applications such as neural networks
- Iris uses the bandwidth 6.4% more efficiently, which is significant over millions of executions

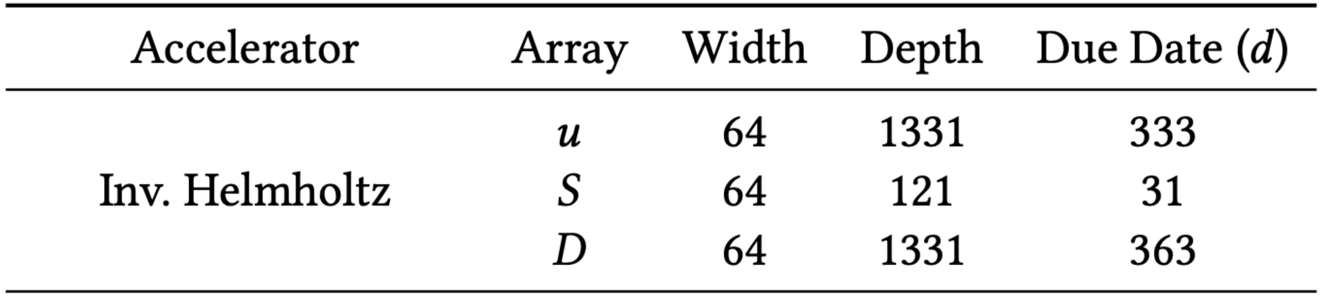
Inverse Helmholtz



Accelerator	Array	Width	Depth	Due Date (d)
Inv. Helmholtz	u	64	1331	333
	S	64	121	31
	D	64	1331	363

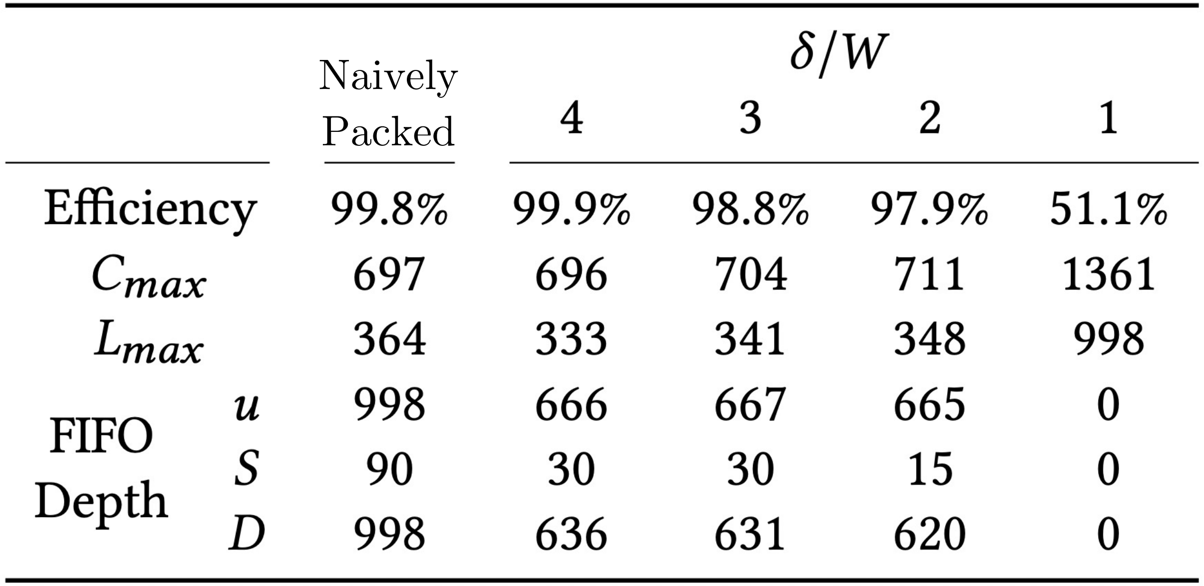
- The Inverse Helmholtz is a computational fluid dynamics operator, with three input arrays

Inverse Helmholtz



Accelerator	Array	Width	Depth	Due Date (d)
Inv. Helmholtz	u	64	1331	333
	S	64	121	31
	D	64	1331	363

Layout metrics with buswidth of 256 and varied Due date/Width (δ/W)* (Inv. Helmholtz)



		Naively Packed	δ/W			
			4	3	2	1
Efficiency		99.8%	99.9%	98.8%	97.9%	51.1%
C_{max}		697	696	704	711	1361
L_{max}		364	333	341	348	998
FIFO Depth	u	998	666	667	665	0
	S	90	30	30	15	0
	D	998	636	631	620	0

- The Inverse Helmholtz is a computational fluid dynamics operator, with three input arrays
- The naive packing strategy requires large FIFO depths to accommodate several data elements in a single cycle

Inverse Helmholtz

Accelerator	Array	Width	Depth	Due Date (d)
Inv. Helmholtz	u	64	1331	333
	S	64	121	31
	D	64	1331	363

Layout metrics with buswidth of 256 and varied Due date/Width (δ/W)* (Inv. Helmholtz)

		Naively Packed	δ/W			
			4	3	2	1
Efficiency		99.8%	99.9%	98.8%	97.9%	51.1%
C_{max}		697	696	704	711	1361
L_{max}		364	333	341	348	998
FIFO Depth	u	998	666	667	665	0
	S	90	30	30	15	0
	D	998	636	631	620	0

- The Inverse Helmholtz is a computational fluid dynamics operator, with three input arrays
- The naive packing strategy requires large FIFO depths to accommodate several data elements in a single cycle
- Iris reduces the FIFO depths by 1/3 while maintaining similar efficiency, useful for when an application is device-area-bound

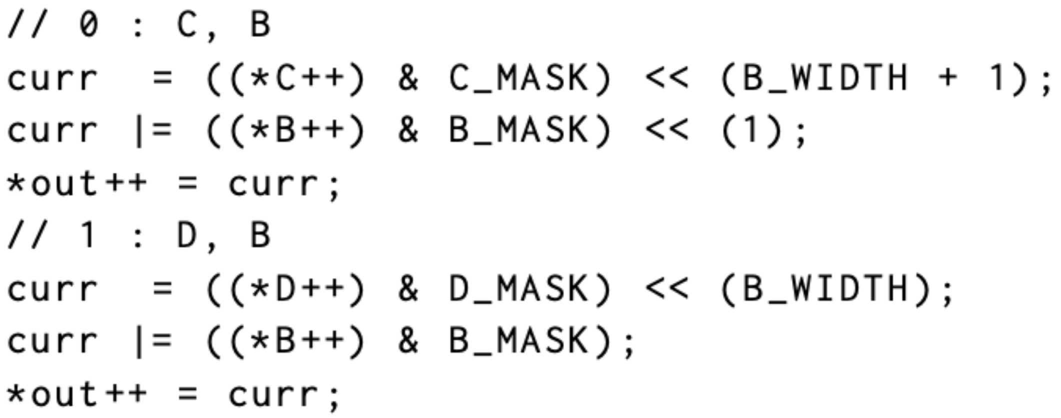
Data Packing and Unpacking

- The protoype tool is able to automatically generate the packing and unpacking functions

Data Packing and Unpacking

- The protoype tool is able to automatically generate the packing and unpacking functions
- **Host-side C code** for arranging the separate input arrays into the Iris layout as one unified array

…



```
// 0 : C, B
curr  = ((*C++) & C_MASK) << (B_WIDTH + 1);
curr |= ((*B++) & B_MASK) << (1);
*out++ = curr;
// 1 : D, B
curr  = ((*D++) & D_MASK) << (B_WIDTH);
curr |= ((*B++) & B_MASK);
*out++ = curr;
```

…

Data Packing and Unpacking

- The protoype tool is able to automatically generate the packing and unpacking functions
- **Host-side C code** for arranging the separate input arrays into the Iris layout as one unified array

```
                        ...
// 0 : C, B
curr  = ((*C++) & C_MASK) << (B_WIDTH + 1);
curr |= ((*B++) & B_MASK) << (1);
*out++ = curr;
// 1 : D, B
curr  = ((*D++) & D_MASK) << (B_WIDTH);
curr |= ((*B++) & B_MASK);
*out++ = curr;
                        ...
```

- **Accelerator-side HLS C code** for unpacking the array elements and moving them into streams to be consumed by the accelerator

```
                        ...
    for(unsigned int t = 0; t < 9; t++){
#pragma HLS pipeline II=1
        elem = in_buf[t];
        if (t == 0) {
            dataC << elem.range(7, 4);
            dataB << elem.range(3, 1);
        } else if (t == 1) {
                        ...
```

Conclusion

- Iris is designed to automatically create an efficient data layout that maximizes the use of the available bandwidth
- Iris was able to achieve:
  - Higher bandwidth efficiency and lower lateness L_{max}
  - Lower FPGA resource utilizations for the data read module, particularly in the case of the data FIFOs
- Iris is an automatic process which relieves the designer of a huge manual effort and supports rapid design space exploration of custom data types

Questions?

stephanie.soldavini@polimi.it

This work was partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST). http://www.everest-h2020.eu