FTF 2016
TECHNOLOGY FORUM

IMAGE SENSOR PROCESSOR (ISP)

FTF-AUT-N1807

MARIE-ANNE LE MENN
ADAS MICROPROCESSOR APPLICATION ENGINEER
FTF-AUT-N1807
MAY 19, 2016

AGENDA

- The purpose of the ISP
  - S32v234
  - Camera interface
  - Image sensor Processing
  - Image pipeline
- Its architecture
- How to program it

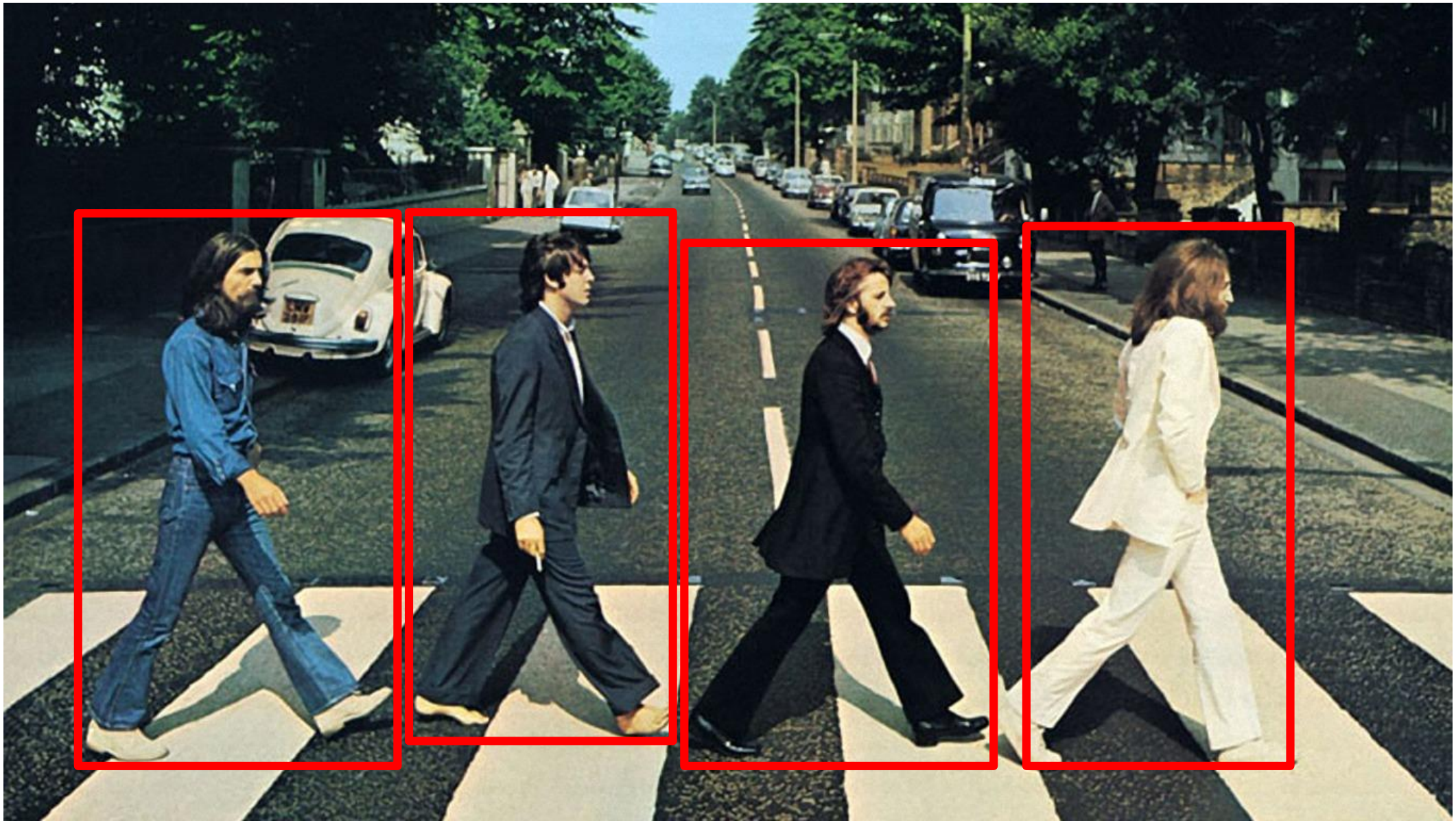
THE PURPOSE OF THE ISP

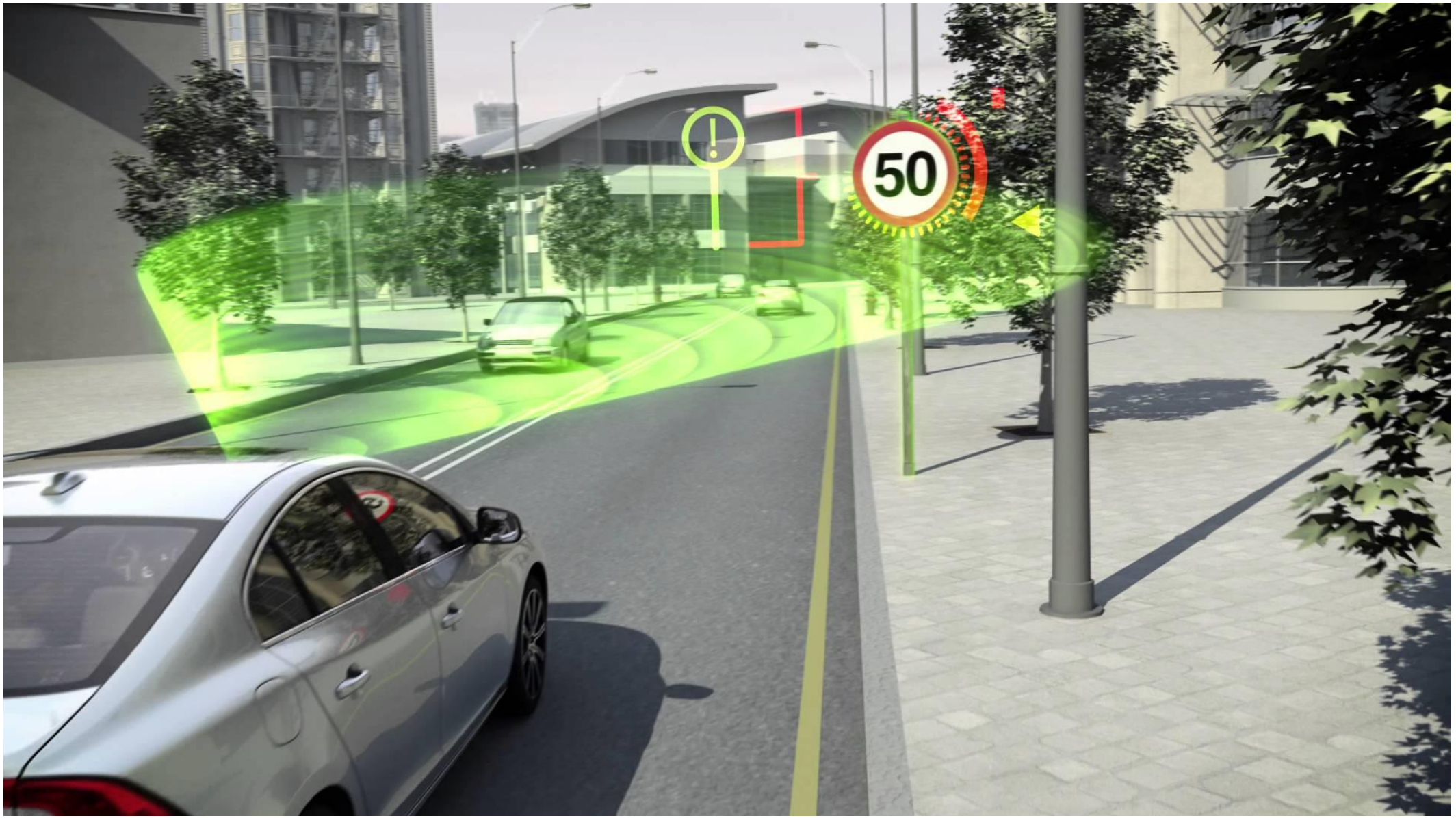
S32V234:
ADAS PROCESSOR

S32v234

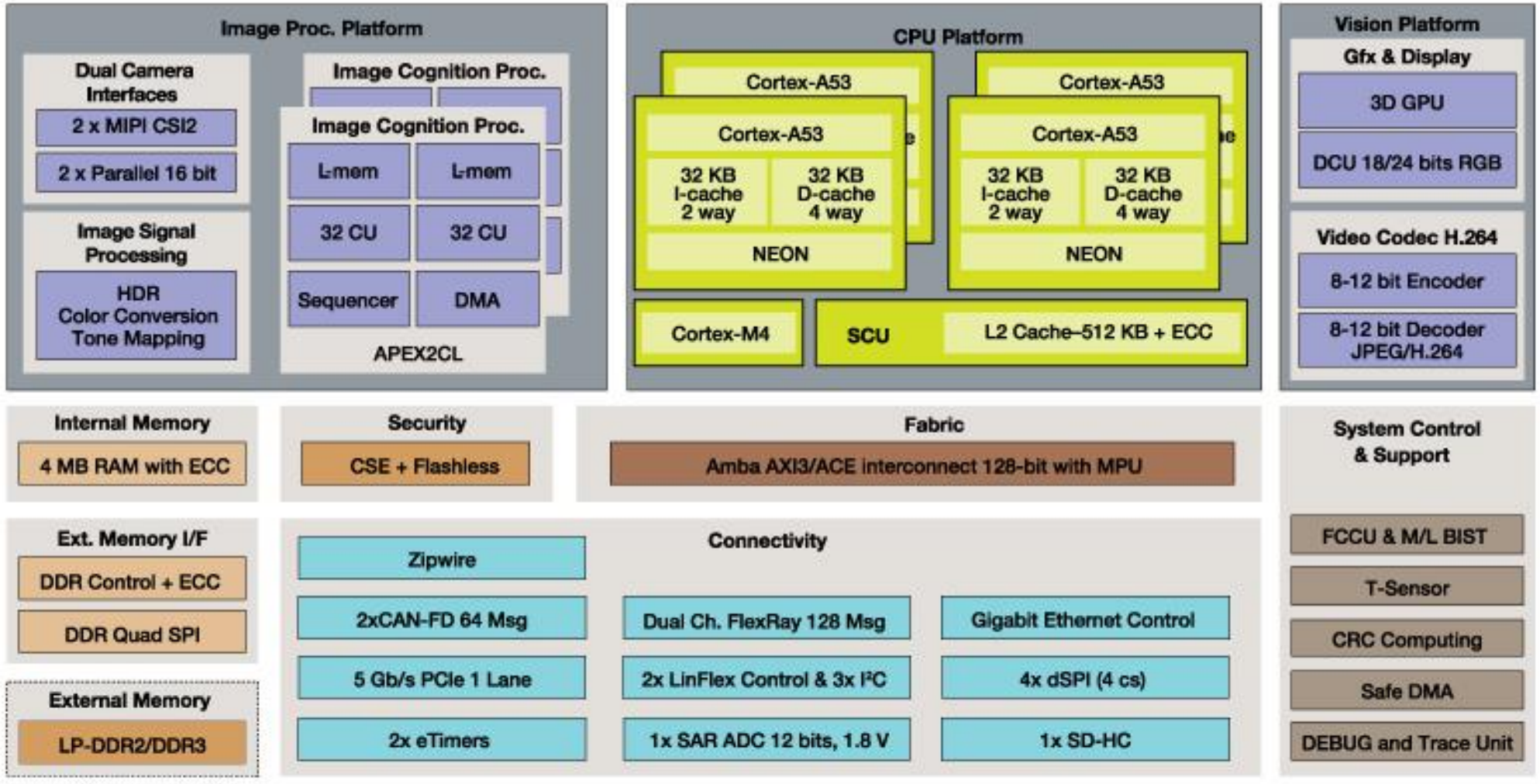
- ADAS processor:
  - Sensor fusion
  - Front view camera
  - Surround-view


SORENTO

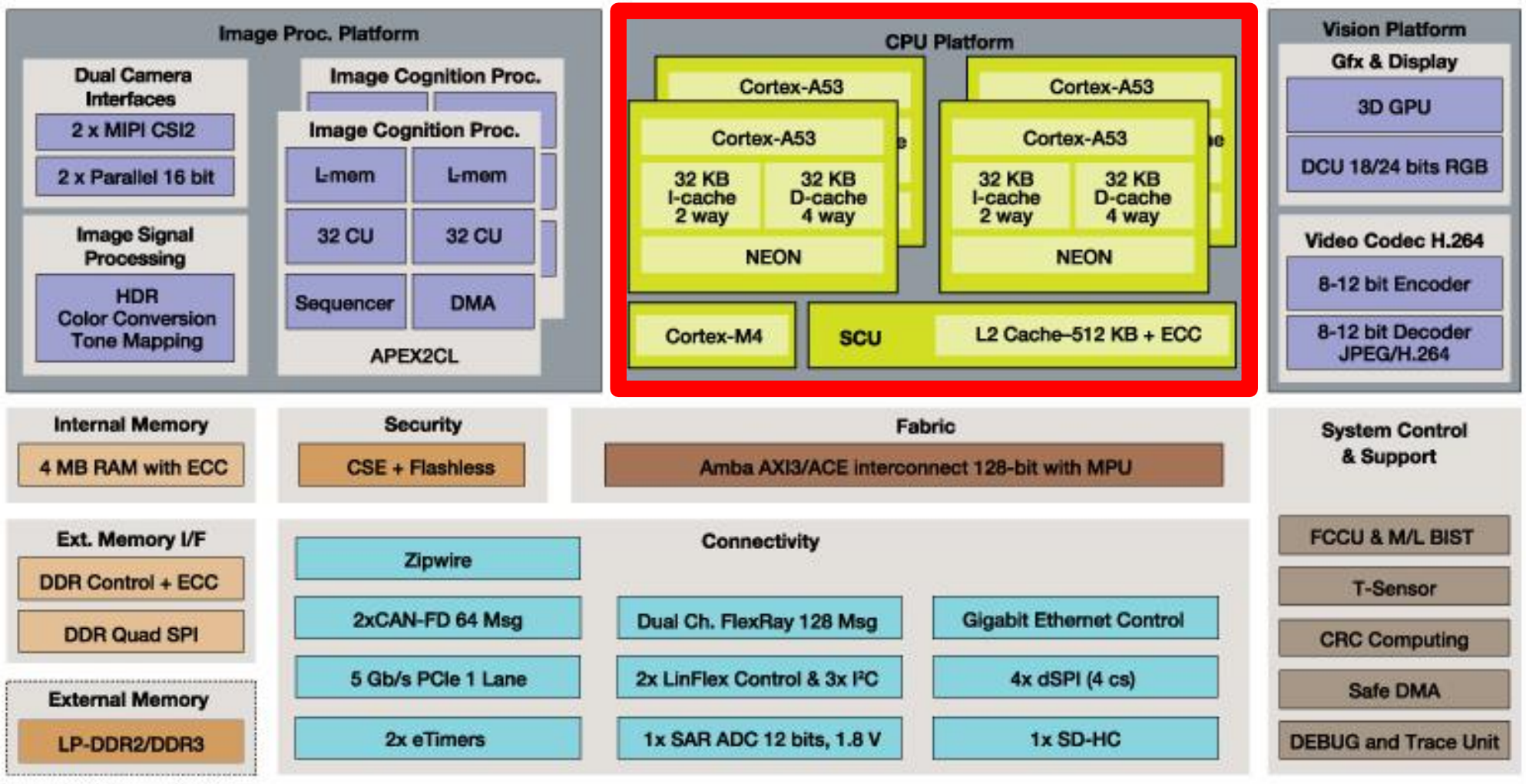



50

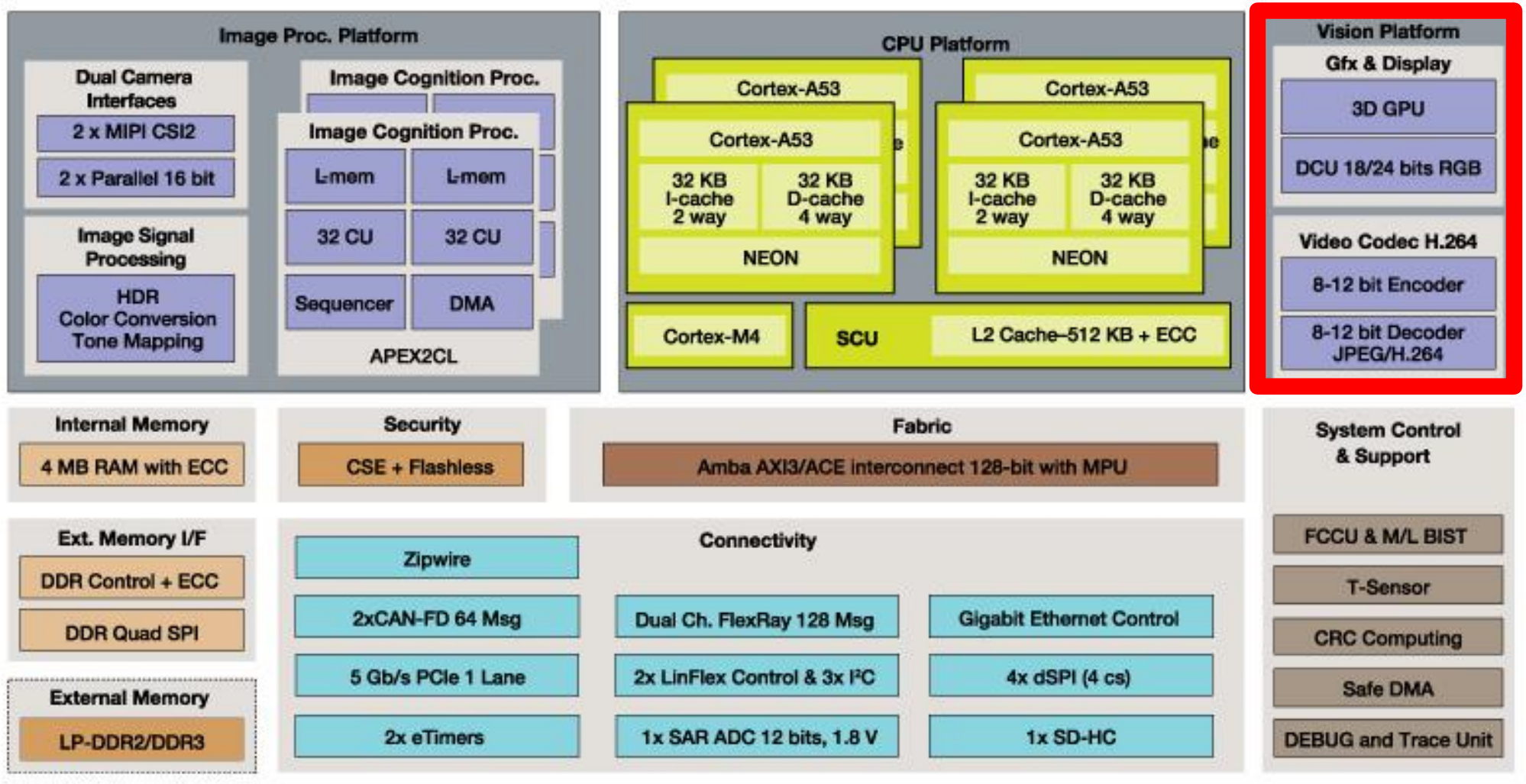
S32v234: Block Diagram



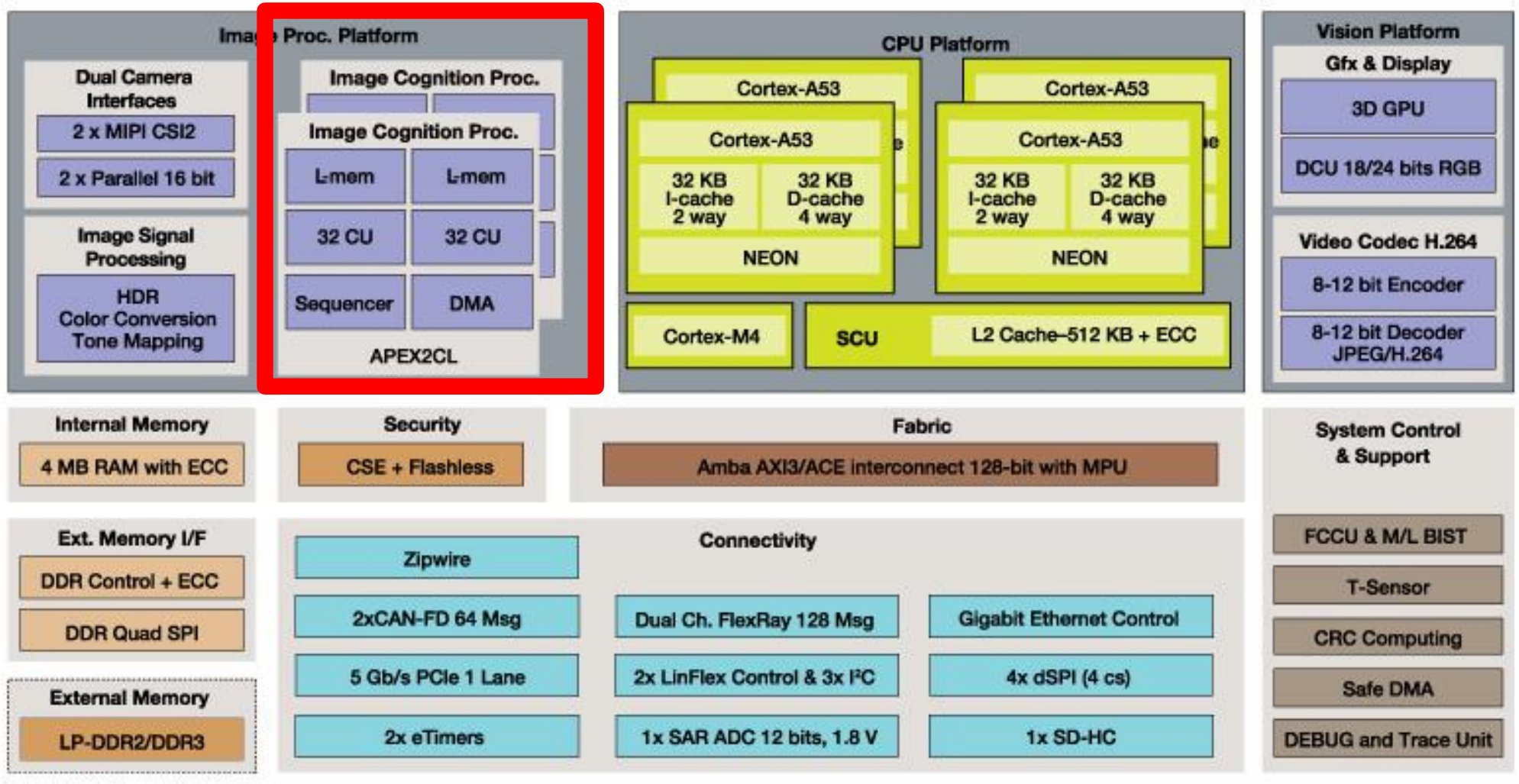
S32v234: Block Diagram



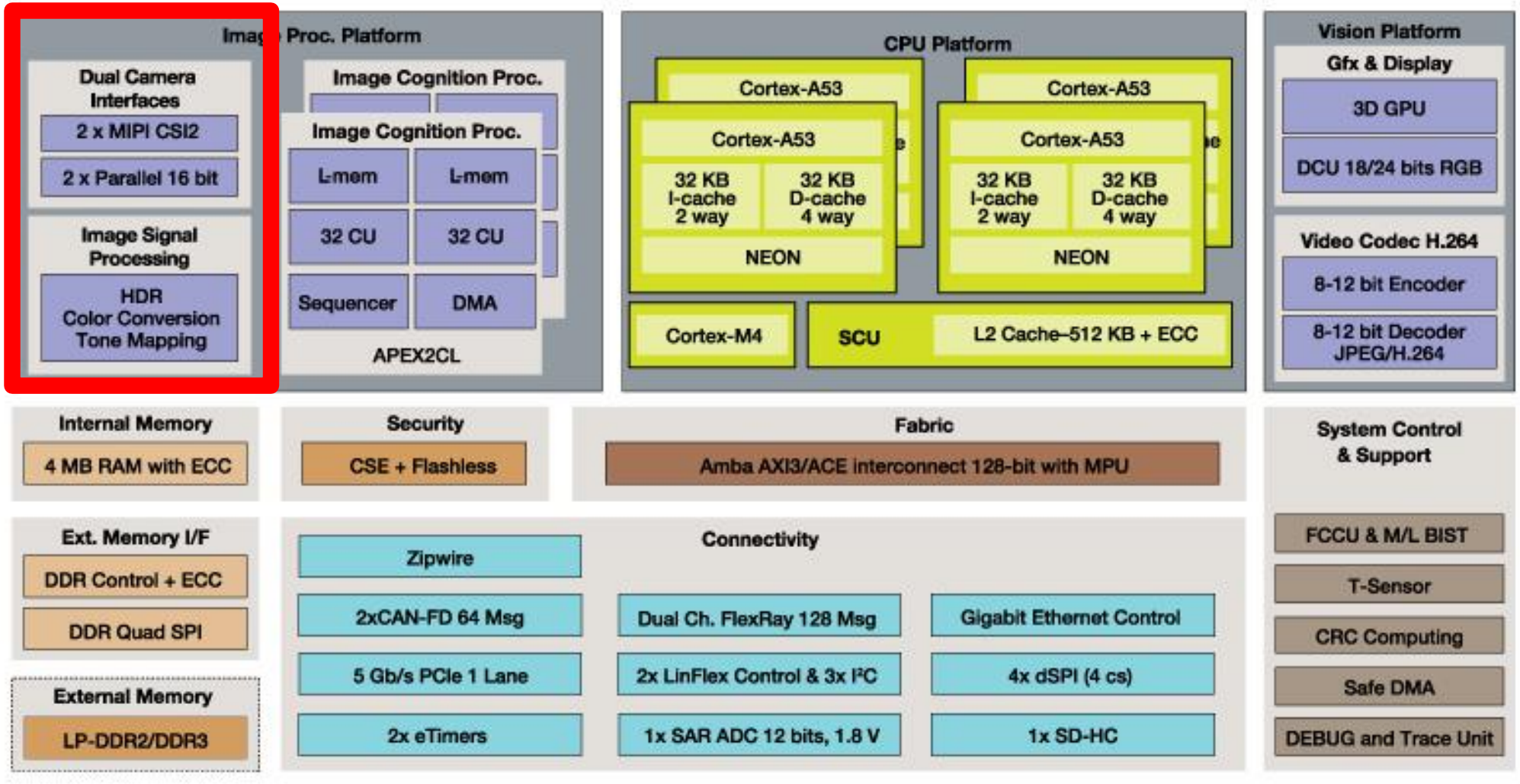
S32v234: Block Diagram



S32v234: Block Diagram



S32v234: Block Diagram



S32v234: Vision Application

Image Pre-Processing
- ISP
- Multi-streams connectivity

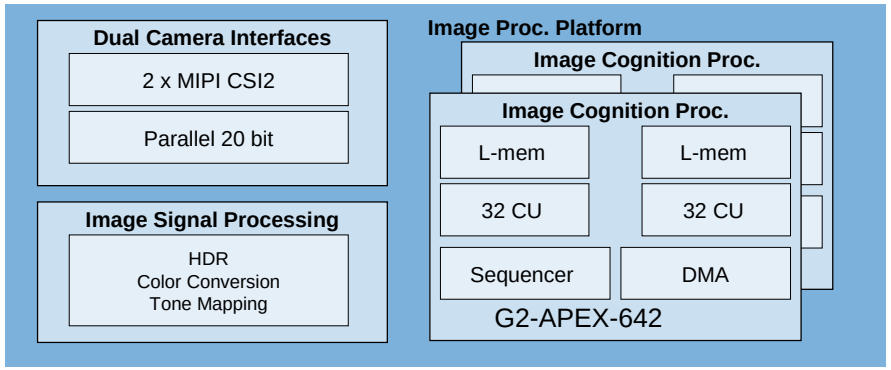
Features Extraction
- Corners
- Edges
- Intensity gradients
- Shapes

Features Classification & Prediction
- SVM
- Adaboost
- K-nearest neighbor

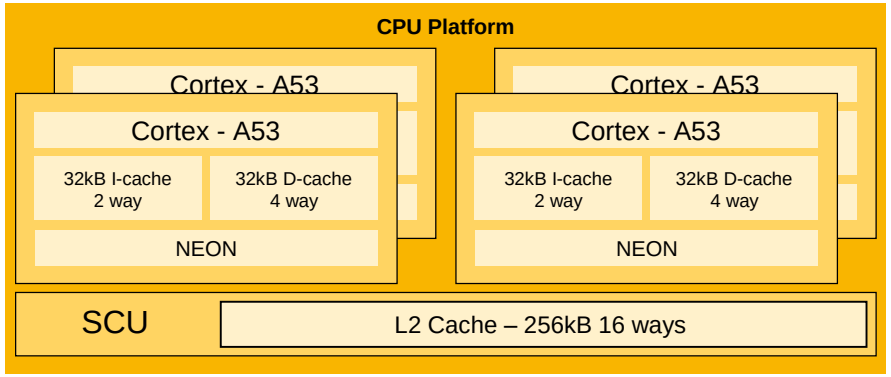
Multi Image Processing
- Tracking, motion estimation
- Optical flow & disparity
- Stitching

Object Recognition & Fusion
- Object recognition/pedestrian
- Augmented reality
- Face detection

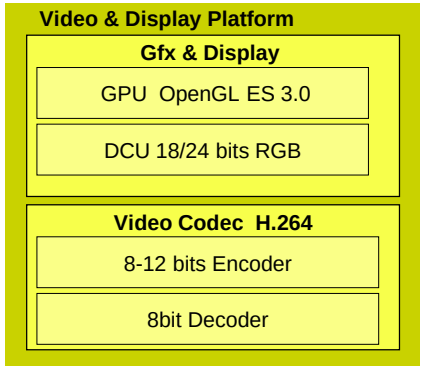
GFX Overlay & Video Distribution
- Safe fusion
- Graphic overlay & display
- 2D vs. 3D projection



High bandwidth operations
Scalable MIMD local memory
Soft ISP



Scalable RISC – data fusion
SIMD co-processor - neon
Memory hierarchy and coherency



Graphic
Video codec
Smart display

CAMERA INTERFACES

Camera Interface: MIPI-CSI

- MIPI-CSI2 standard
- 4 lanes up to 1.5Gbps each
- 4 virtual channels

- In theory: up to 6Gbps for 1 MIPI interface
- In reality:
  - No camera with such bandwidth
  - LVDS limitation

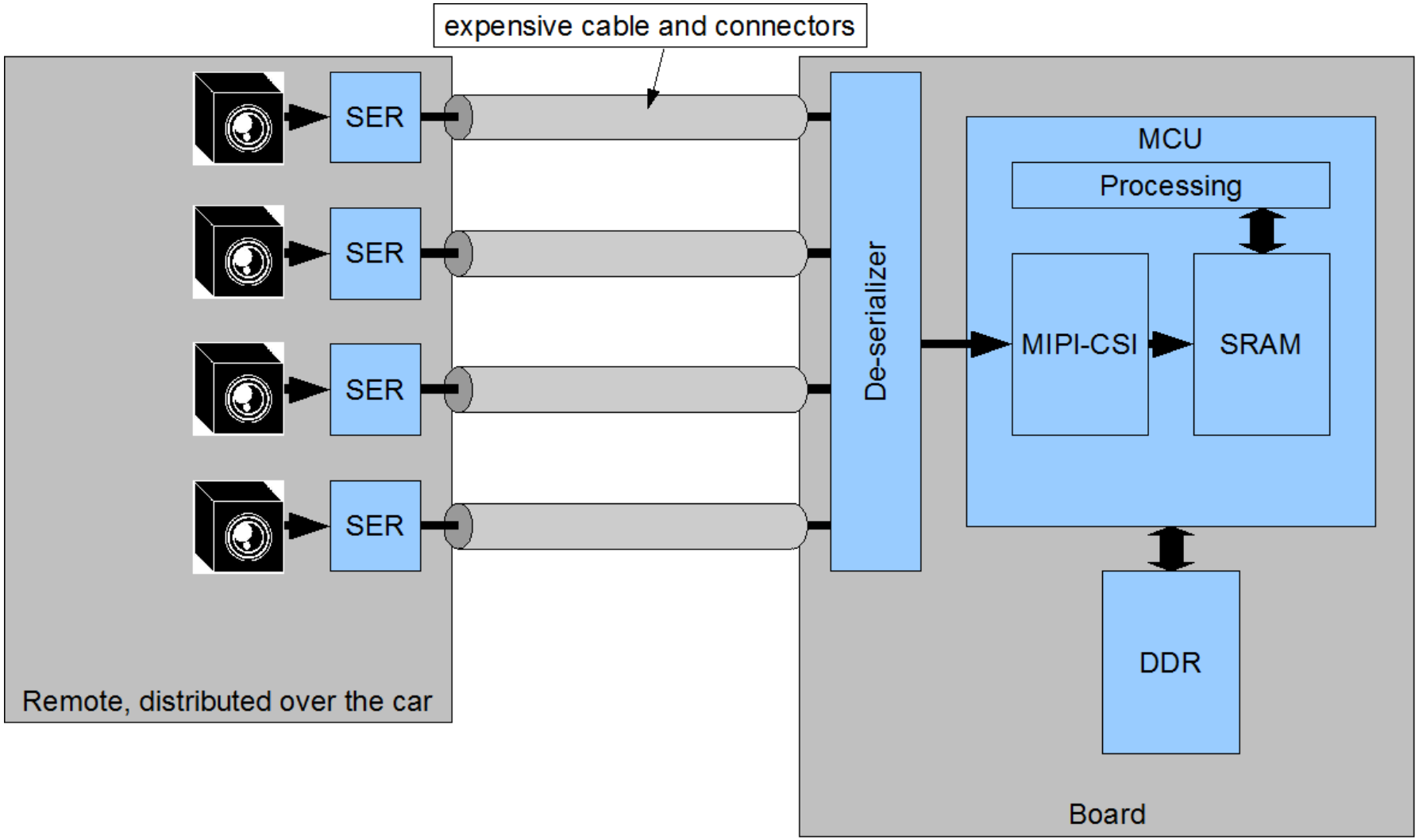
S32v234 has 2 MIPI-CSI interfaces

Camera Interface: MIPI-CSI

- Typical sensor resolution:
  - 1280x800 @ 30/60 fps
  - 1920x1080 @ 30/60 fps

- Examples of camera:
  - Sony: IMX224MQV
  - Omnivision: OV10640

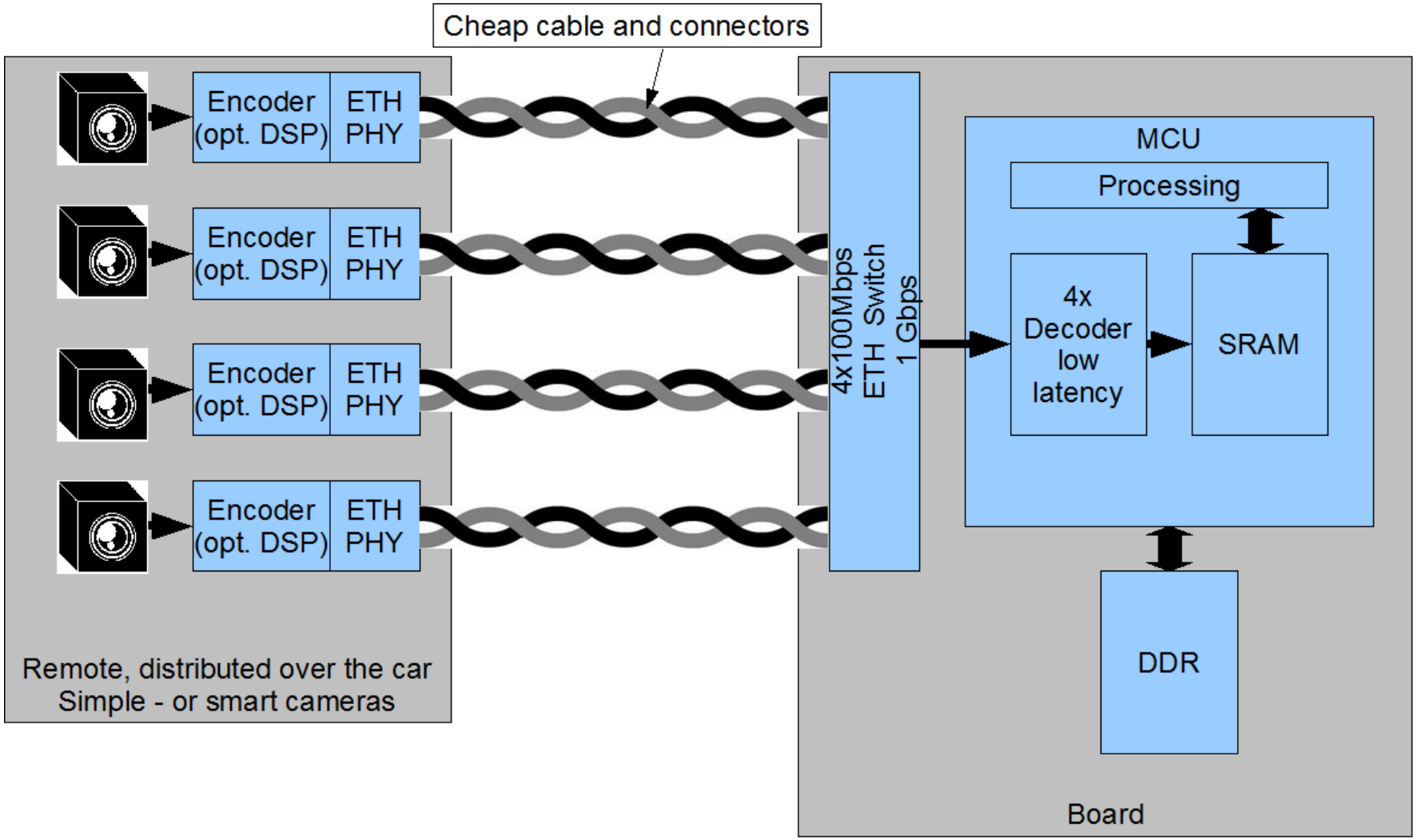
Camera Interface: Multiple Cameras



Camera Interface: Ethernet

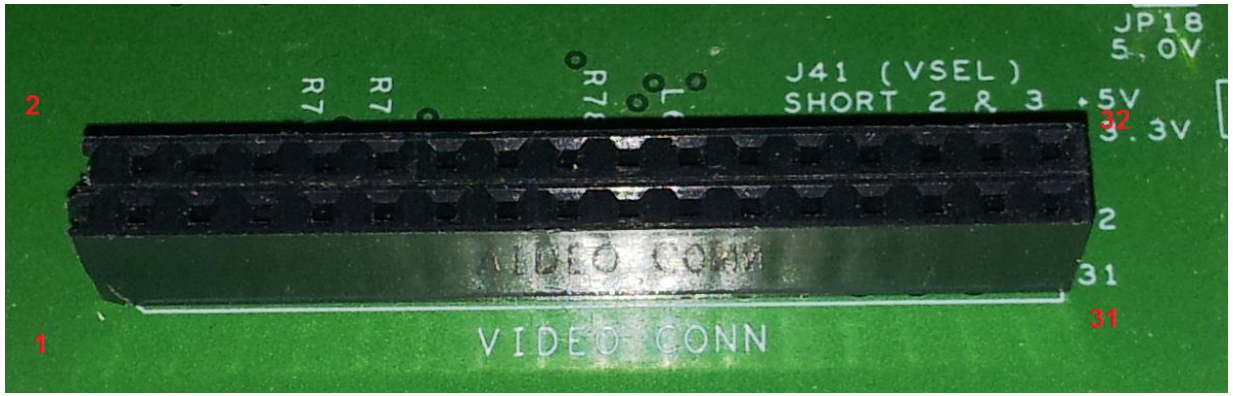
- 1Gbps interface
- Easy synchronisation via AVB
- Embedded decoder for up to **4 video streams**

Camera Interface: Ethernet



Camera Interface: Parallel

- Up to 100MHz pixel clock
- Pin out options:
  - 2 interfaces x16-bit
  - 1 interface x20-bit + 1 Interface x12-bit

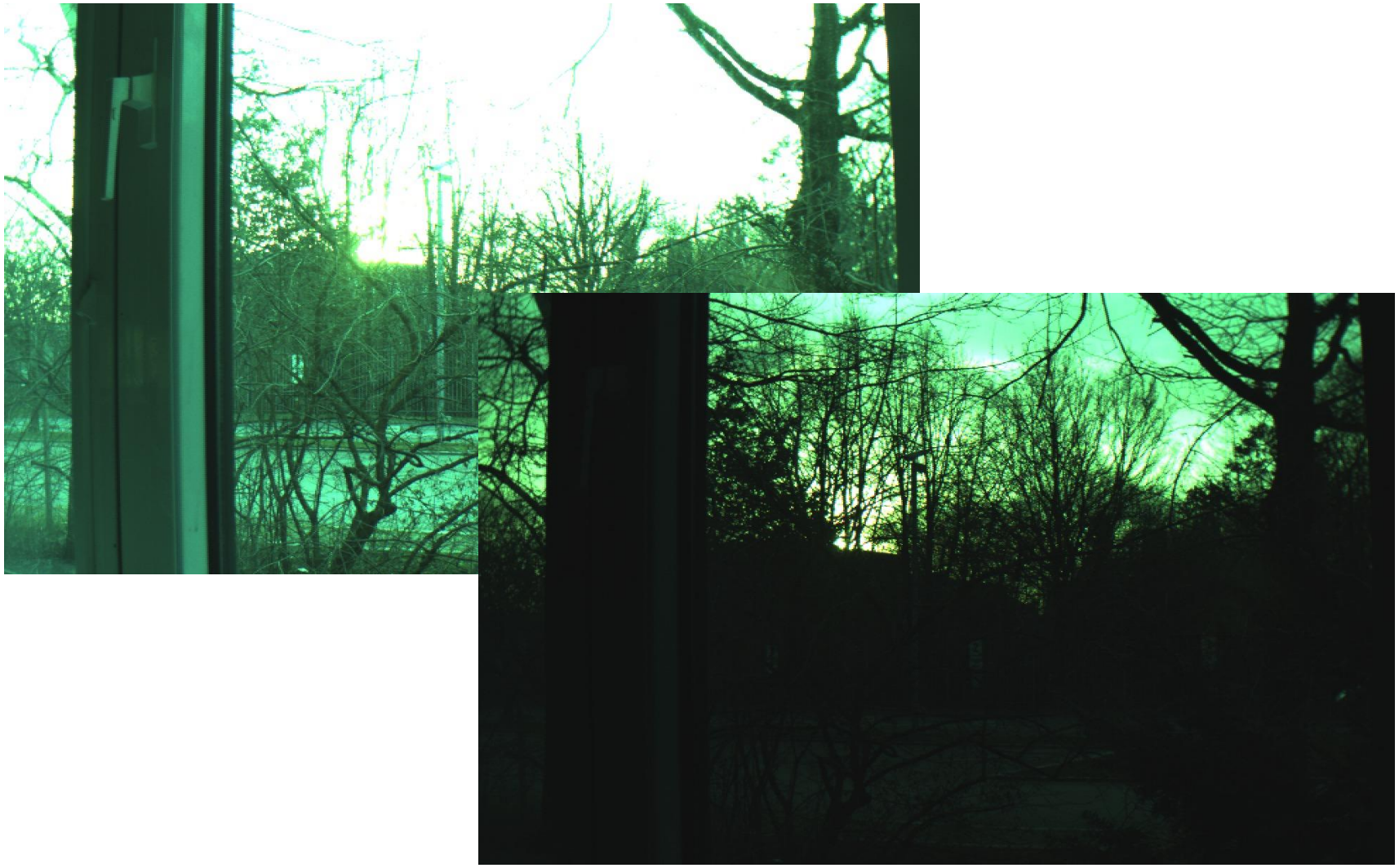


Signals:

- Data [0:19]
- Vsync, Hsync, PixClk
- I2C
- Power-down

IMAGE SENSOR PROCESSING

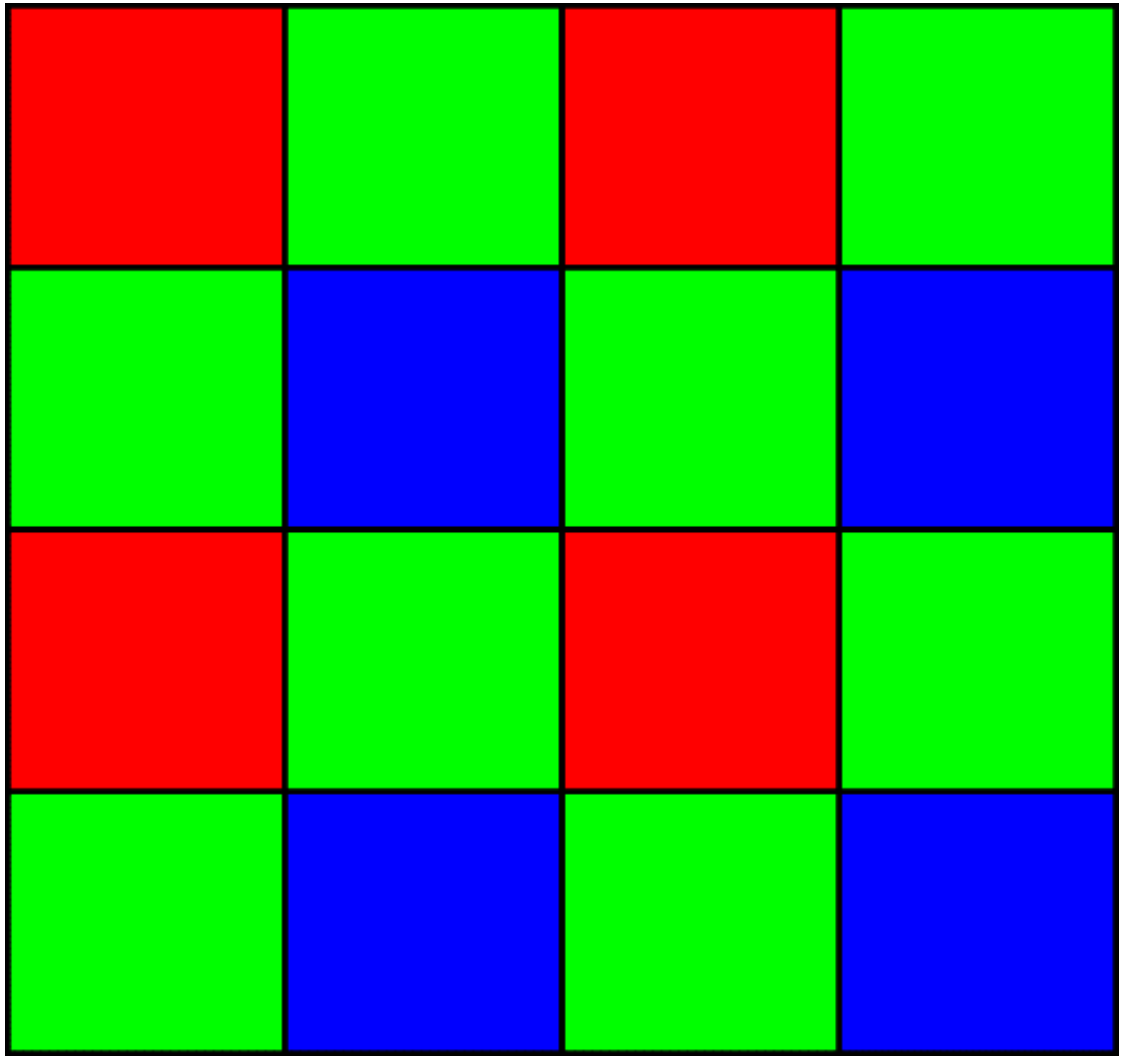
Raw Camera Image



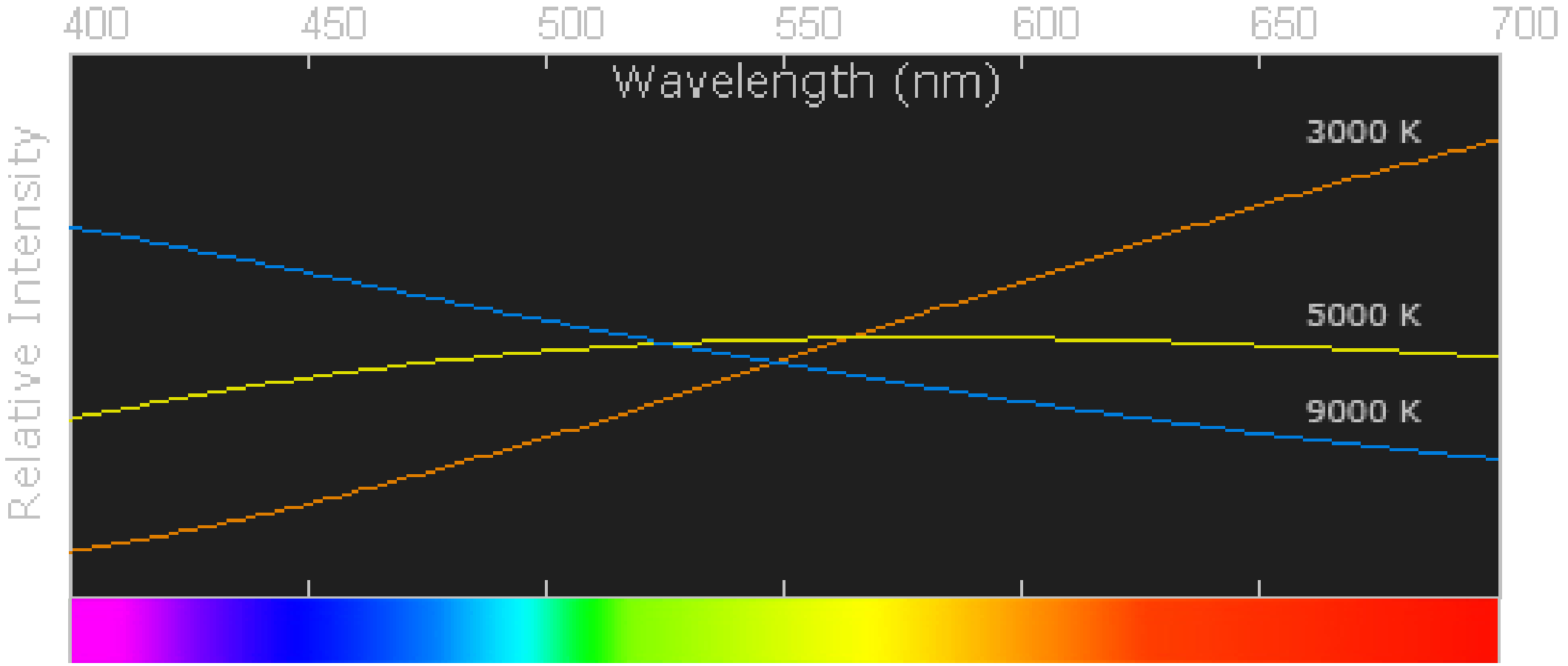
Raw Camera Image

Why does it look greenish?

Bayern Pattern



White Balancing



Relative intensity has been normalized for each temperature (in Kelvins).

White Balancing

Raw Camera Image

Why is there 2 frames?

HDR: High Dynamic Range

HDR: High Dynamic Range

High exposure

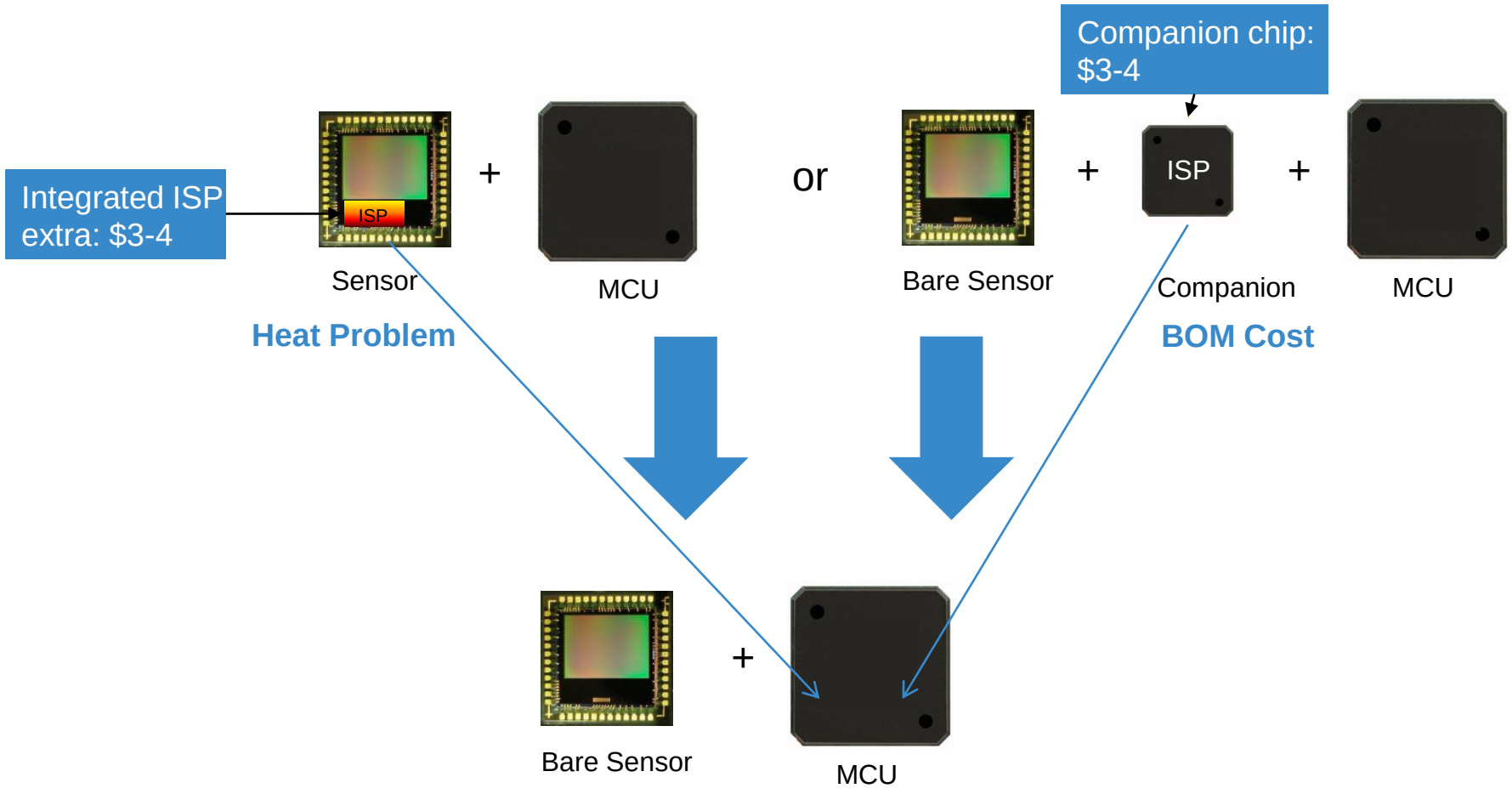
Low exposure

Combination of the 2 exposures

Image Sensor Processing

- White balancing
- De-bayering
- HDR
- Black level
- Noise filtering
- Vignetting
- Colour conversion: YUV, Gray scale

ISP Solutions



S32v234: integrated solution

Advantages

- Manage power on sensor
- Optimize BOM and board
- Smaller size through integration

Save Money

Image Pipeline

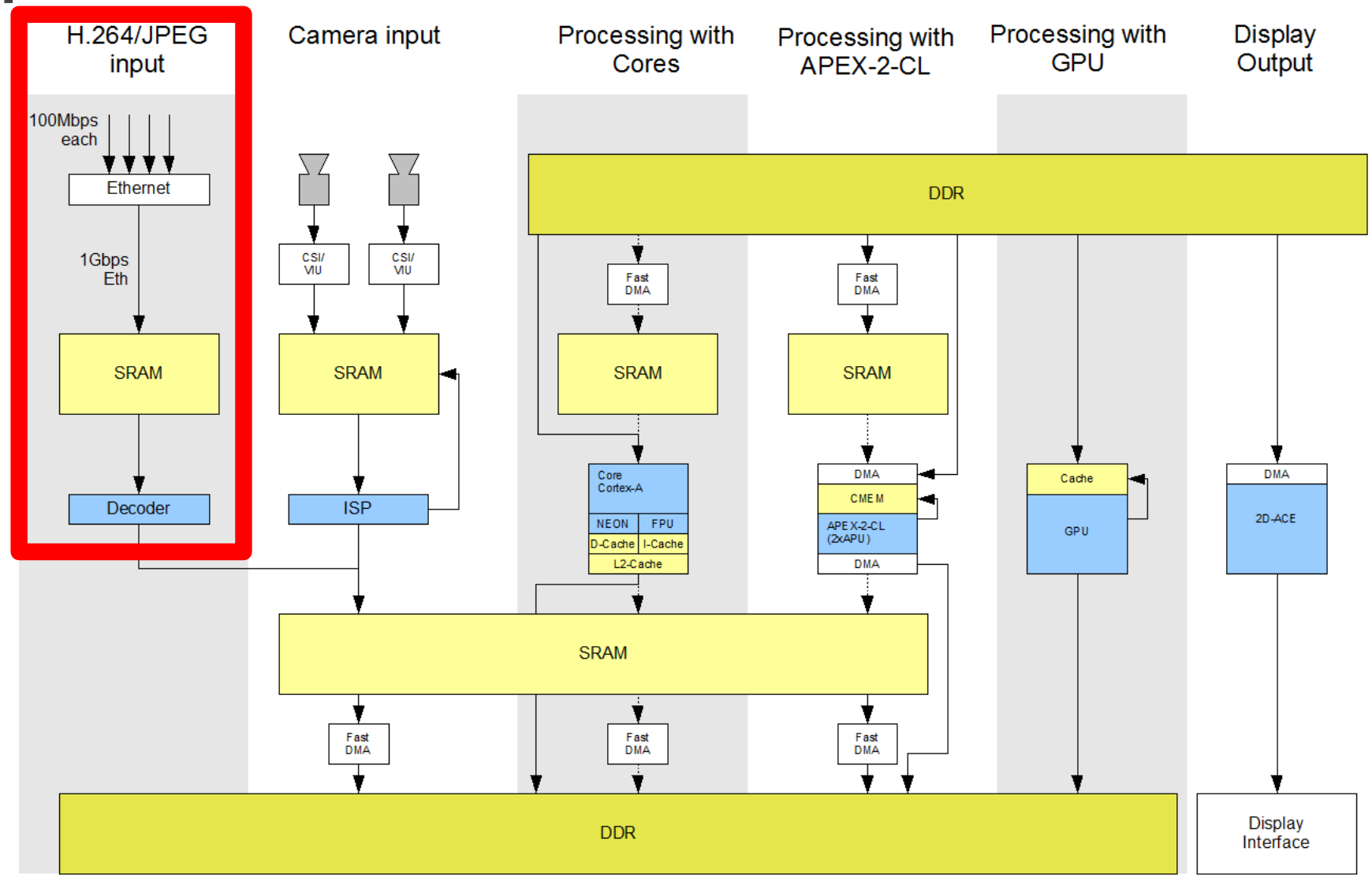


Image Pipeline

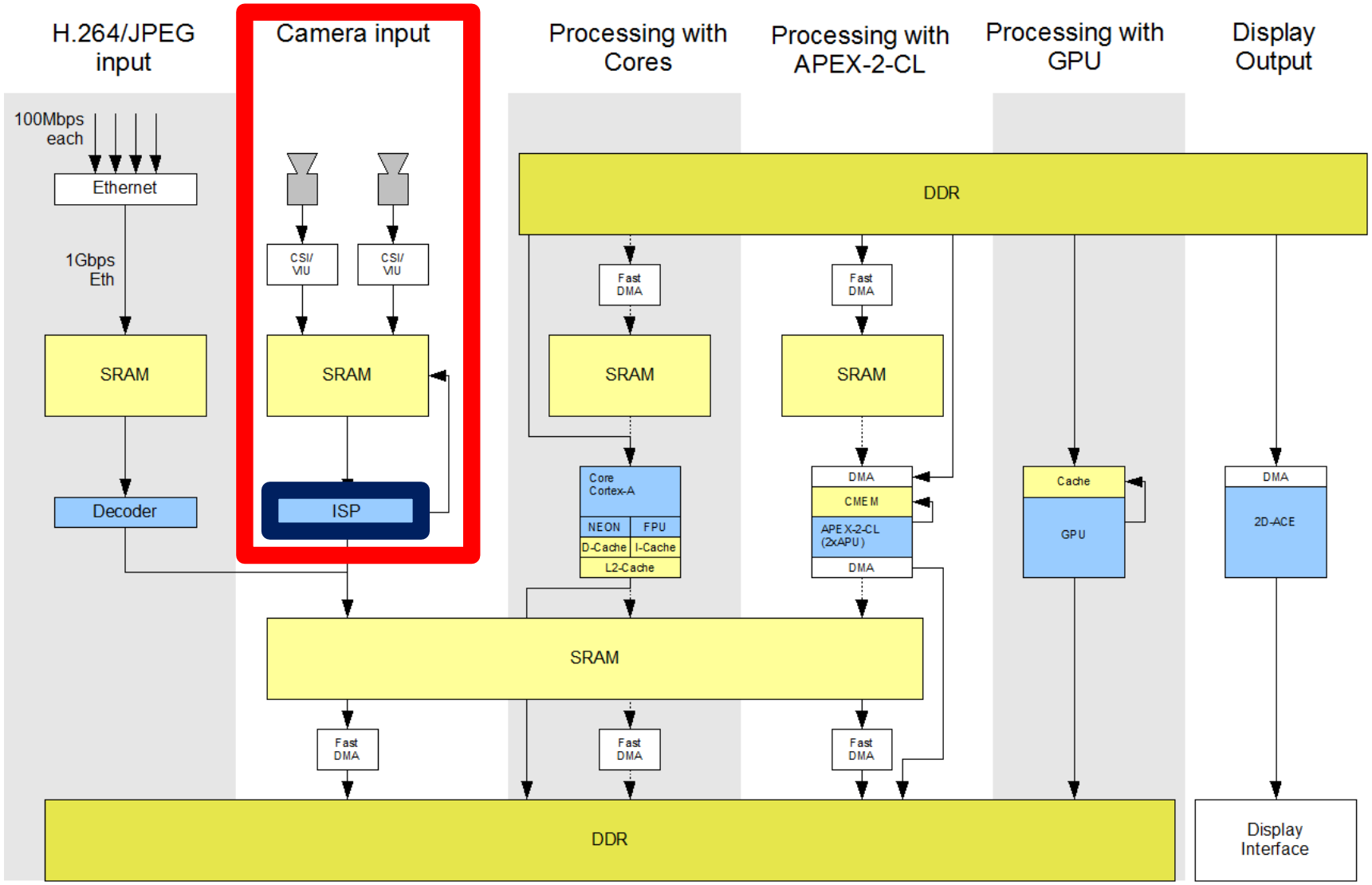


Image Pipeline

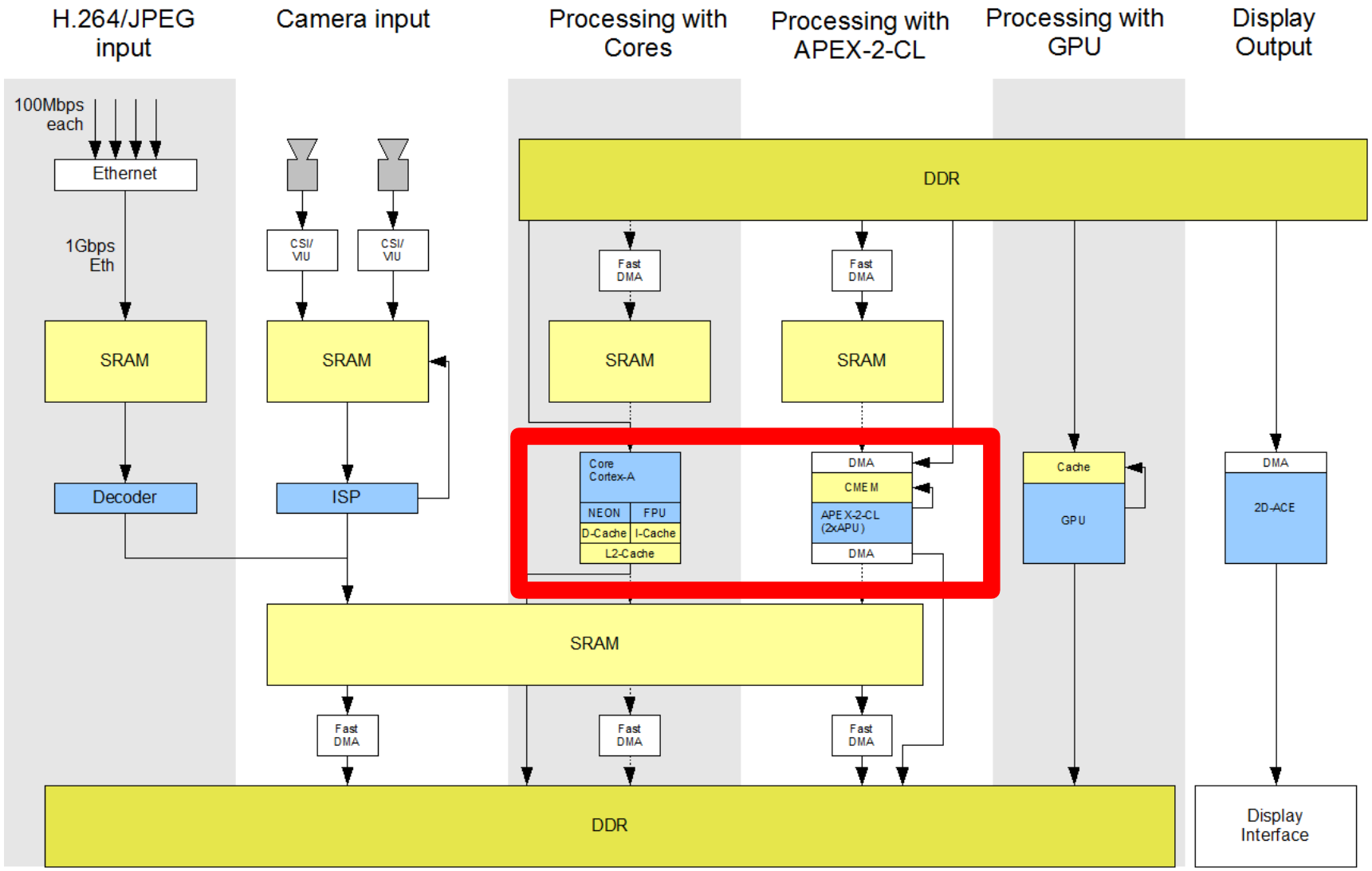
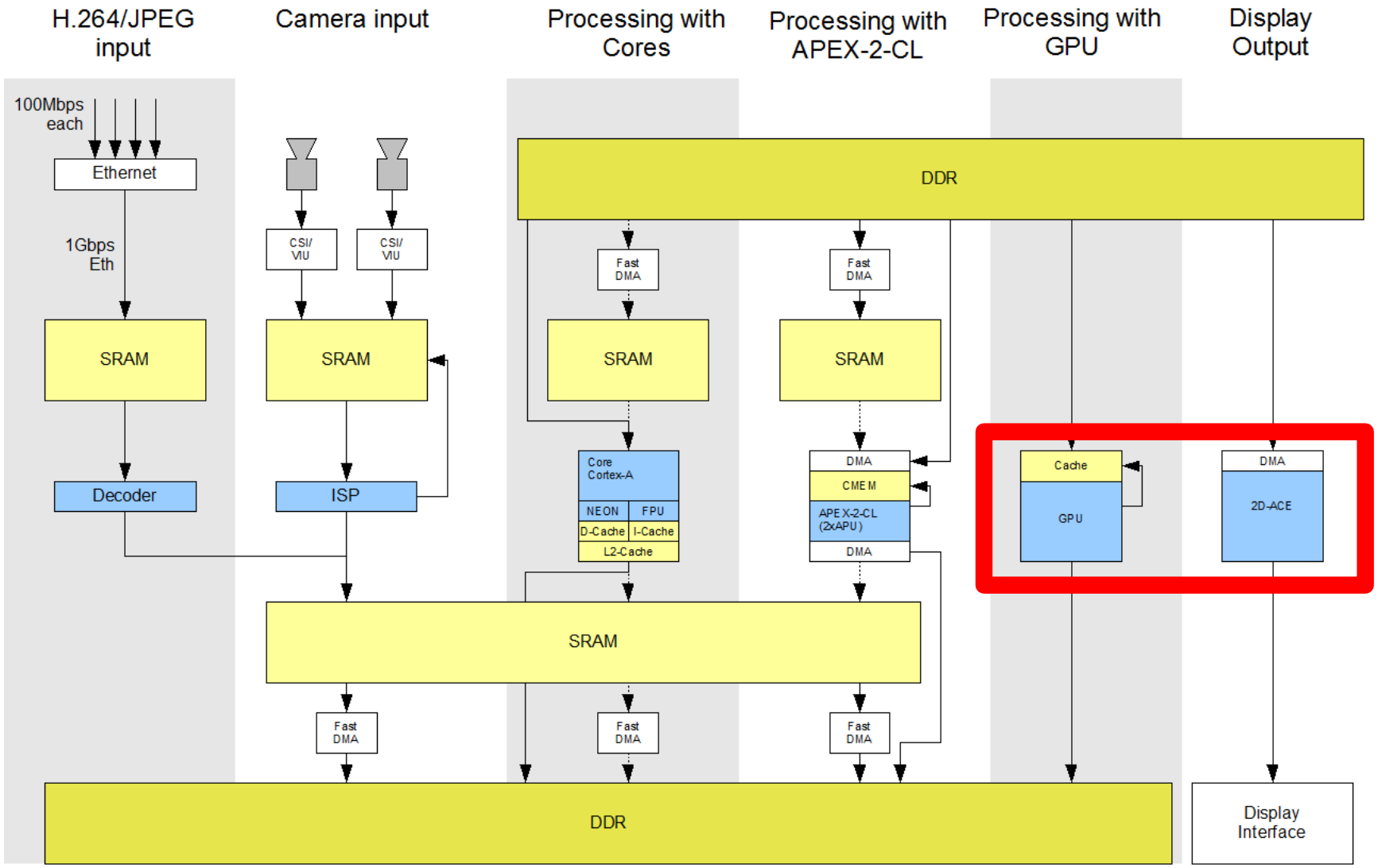


Image Pipeline



When to Use the ISP?

- Ethernet: ISP is already integrated with the camera
- MIPI-CSI & parallel interface:
  - External ISP required for more than 2Mpix
  - Internal ISP will save money for mono or stereo cameras

ITS ARCHITECTURE

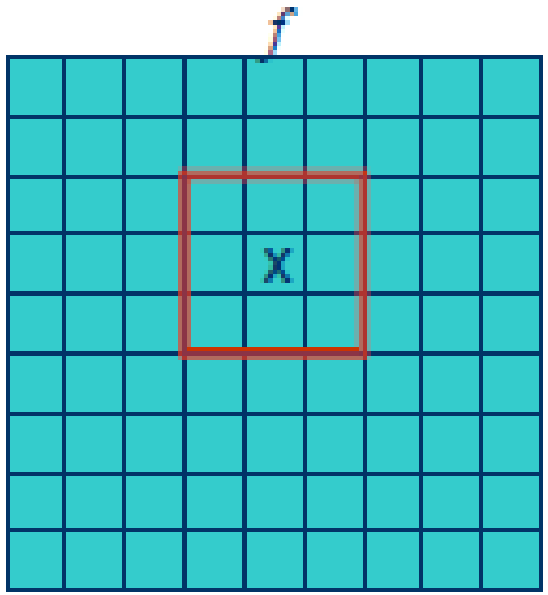
WHAT IS A KERNEL?

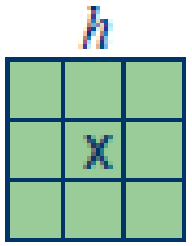
Kernel

- **Do not mistake:**
  - **Linux kernel**

Kernel

- **Do not mistake:**
  - Linux kernel
  - **Image processing kernel**

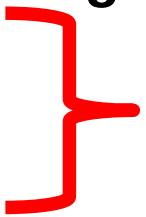




$$f * h = \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(x-i, y-i)h(i, j)$$

Kernel

- **Do not mistake:**
  - **Linux kernel**
  - **Image processing kernel**
  - **APEX kernel**
  - **ISP kernel**



- Both can execute image processing kernel but not only
- Different:
  - Programing method
  - Capabilities
  - Engines

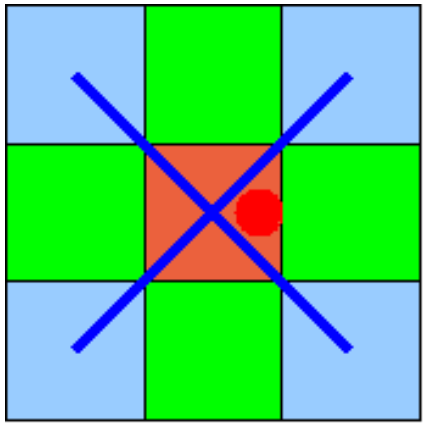
ISP Kernel

- Assembly code
- Running on the Image Processing Units (IPU) of the ISP
- Each IPU can run a different kernel

- Examples:
  - De-bayering
  - Noise filtering
  - Colour conversion
  - …

ISP Kernel

- A lot of image processing steps (kernels) require multiple lines:



De-bayering

-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

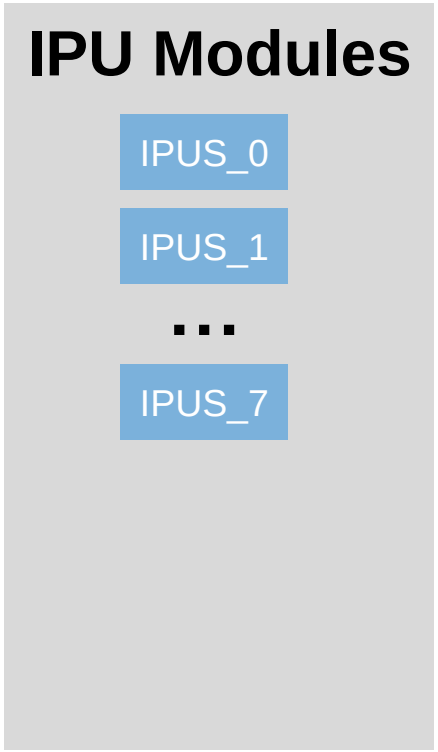
y filter

Sobel filtering

IPUs can fetch multiple lines at the same time

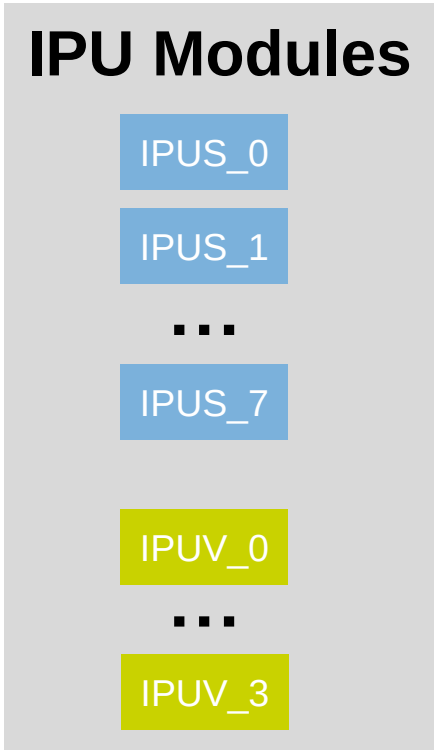
ISP SUBSYSTEM

ISP Sub-system



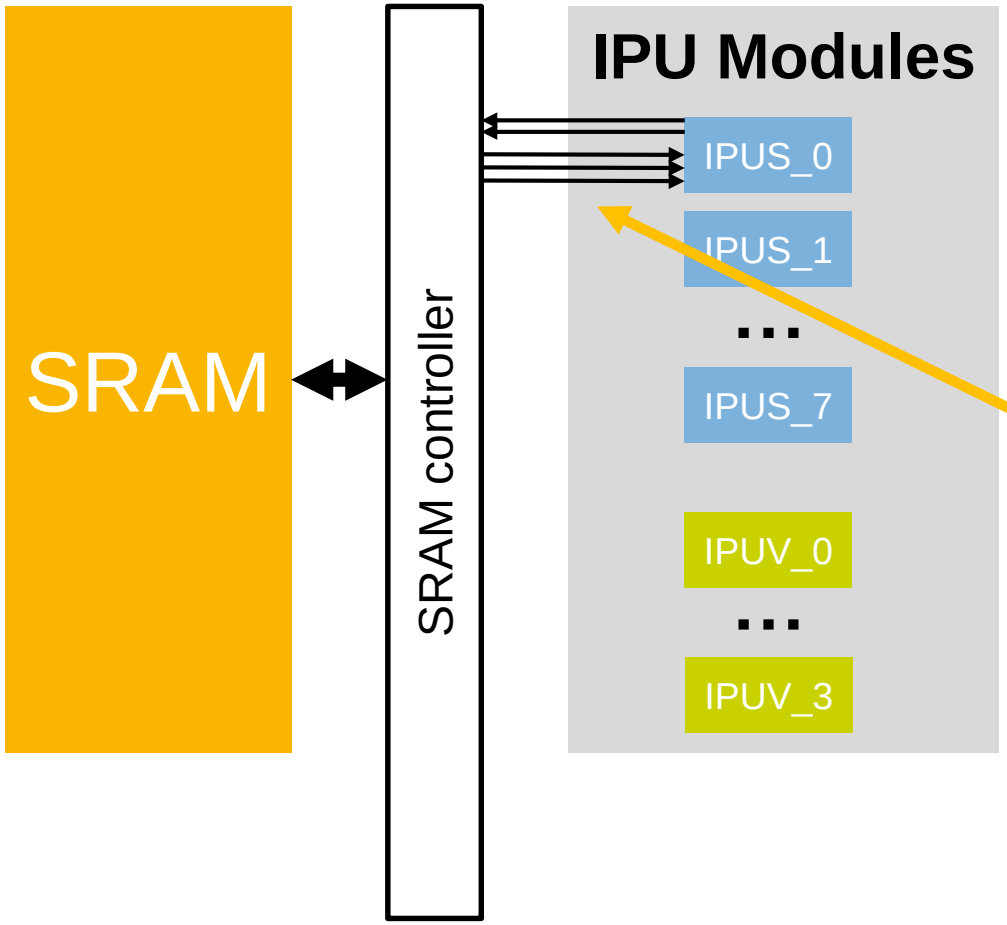
IPUS
=
Scalar Image Processing Unit

ISP Sub-system



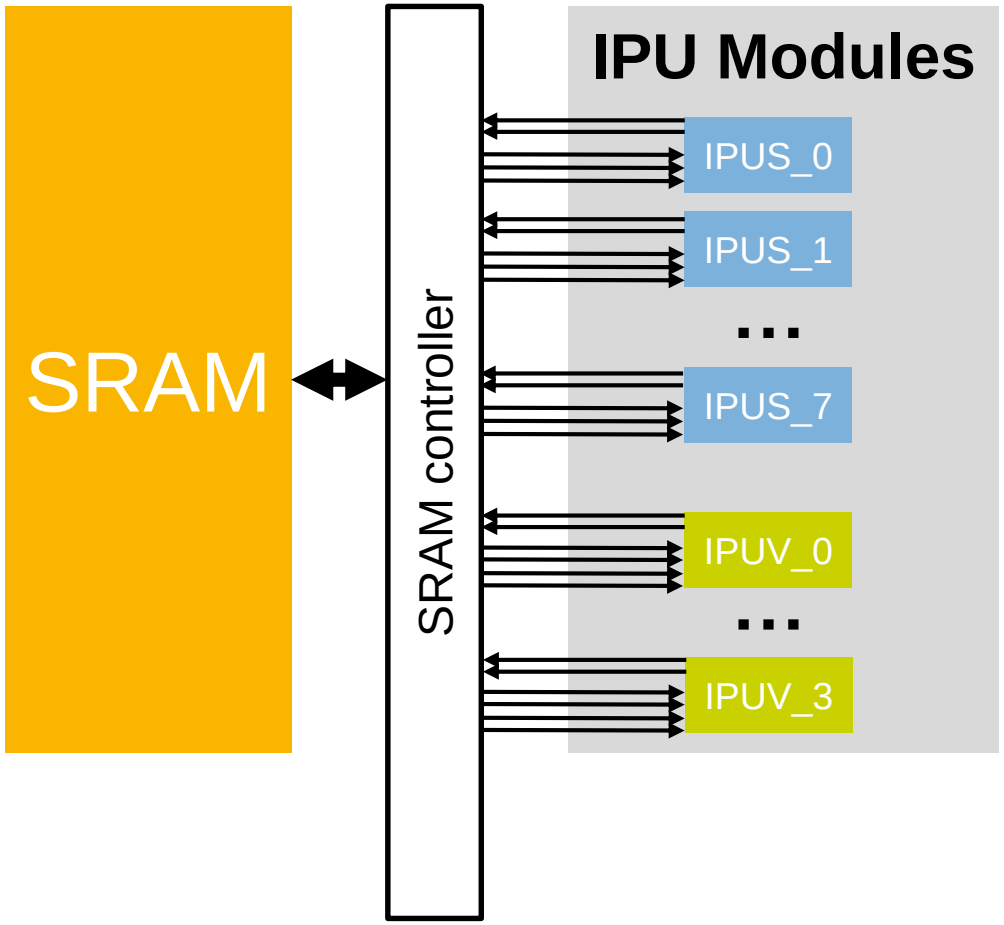
IPUV
=
Vector Image Processing Unit

ISP Sub-system

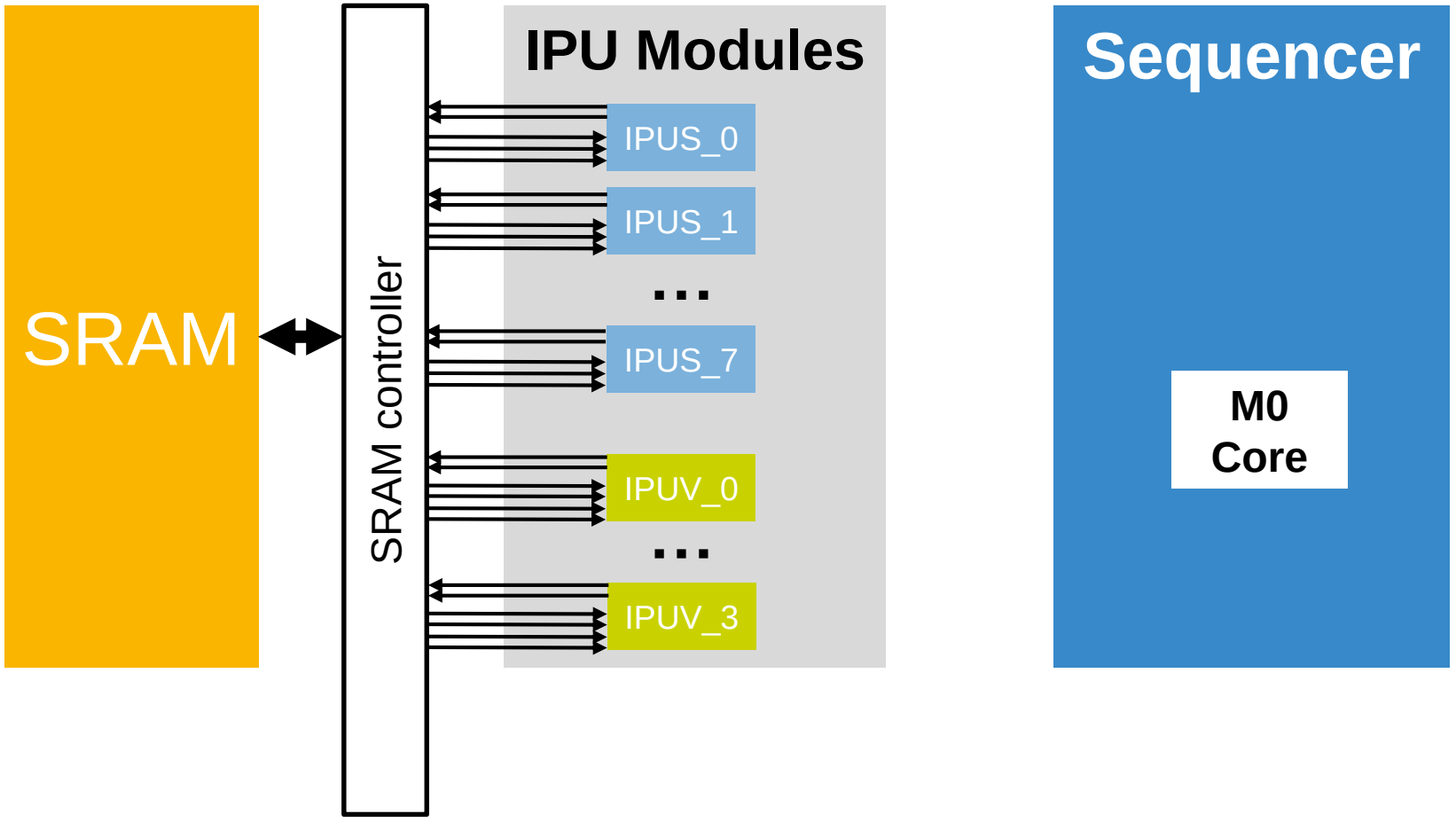


Each input/output can fetch/output multiple data at the same time

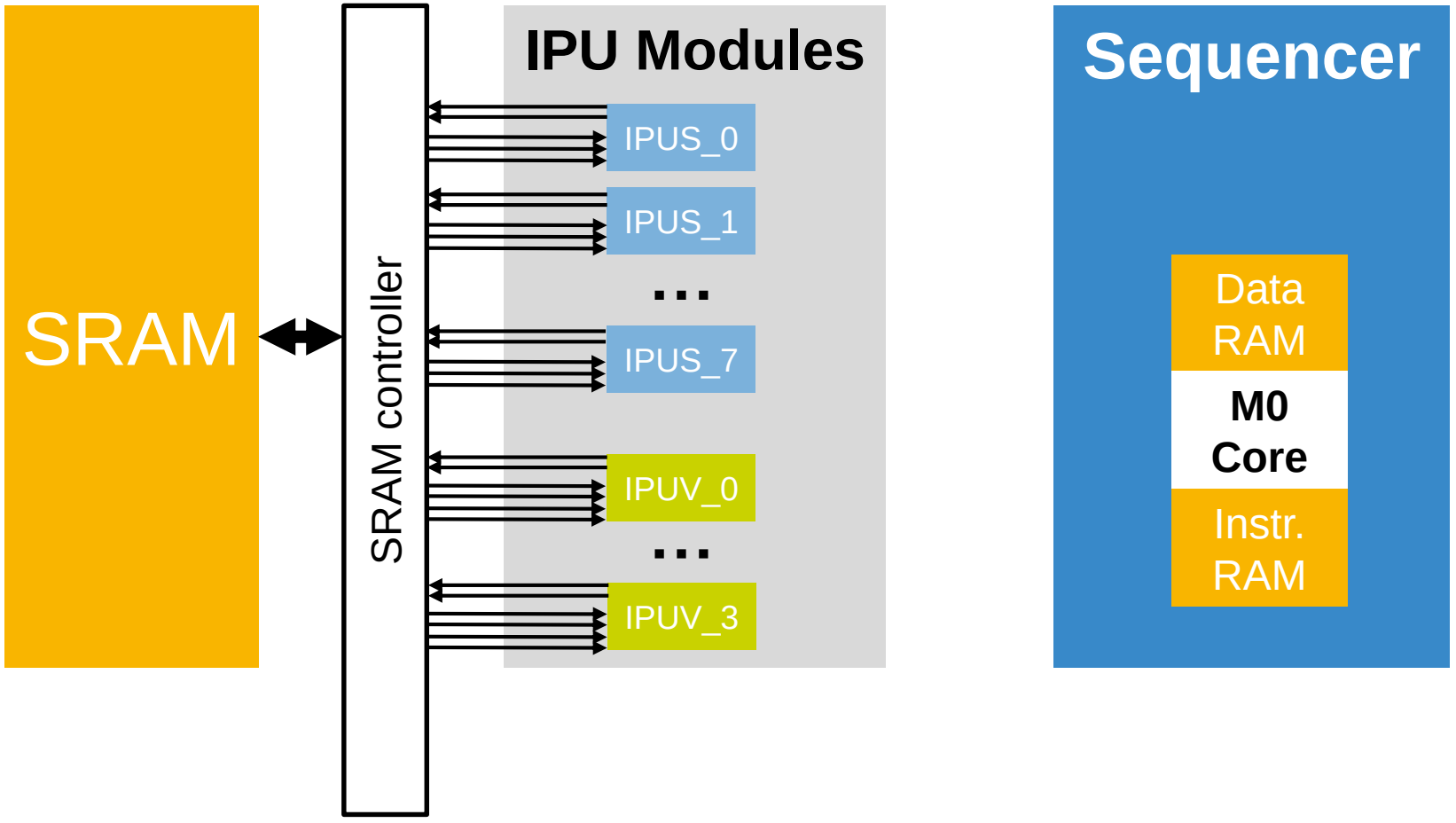
ISP Sub-system



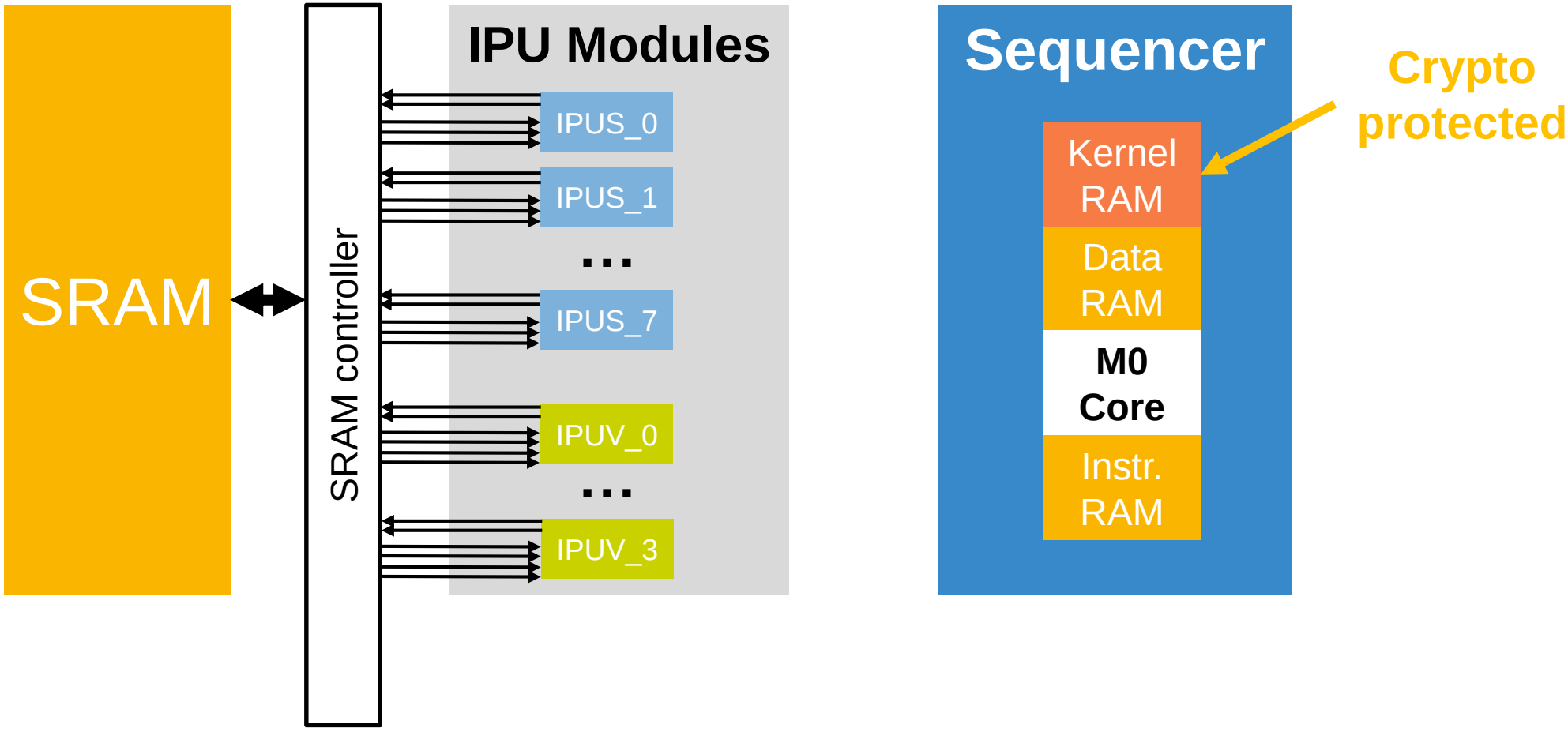
ISP Sub-system



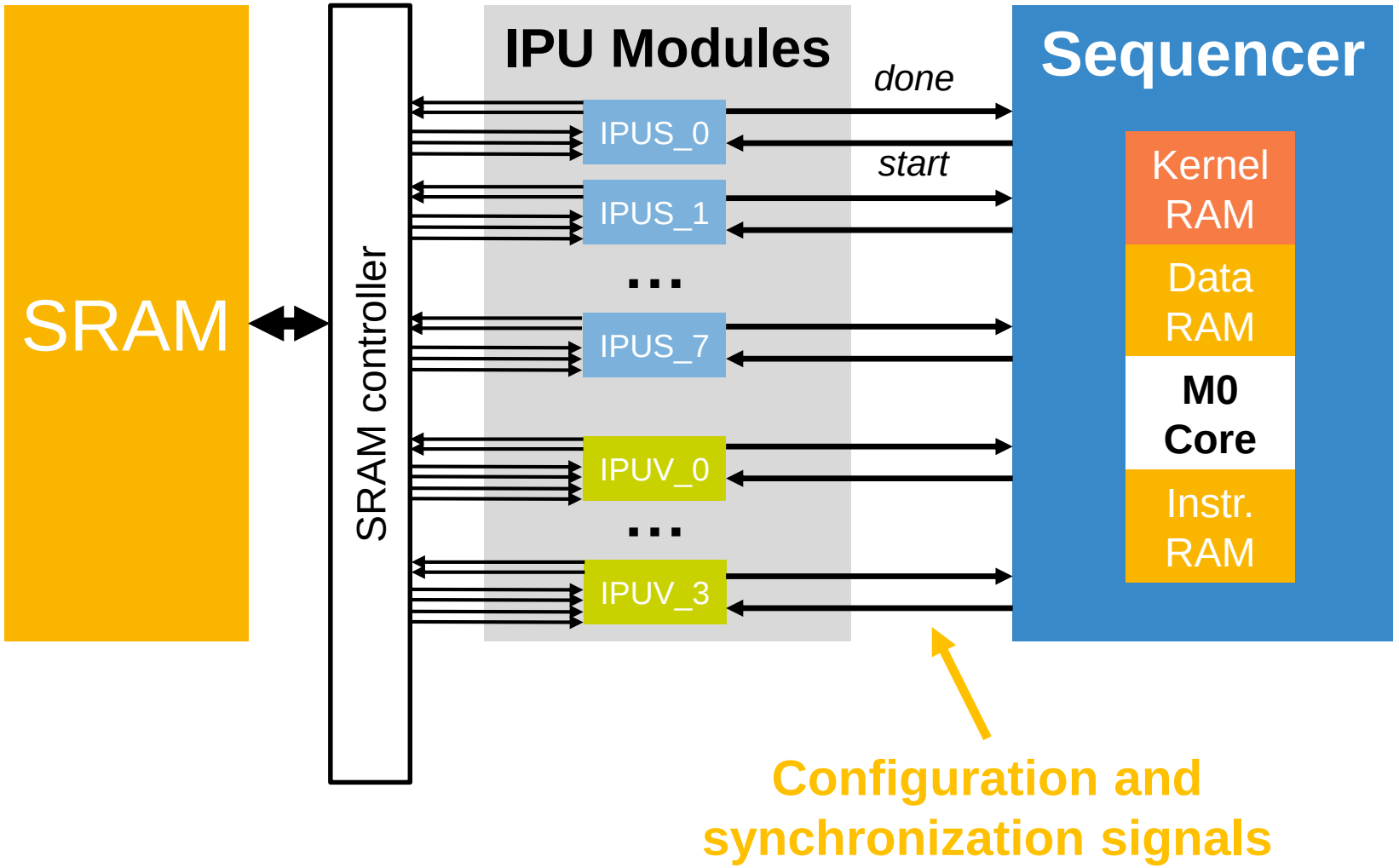
ISP Sub-system



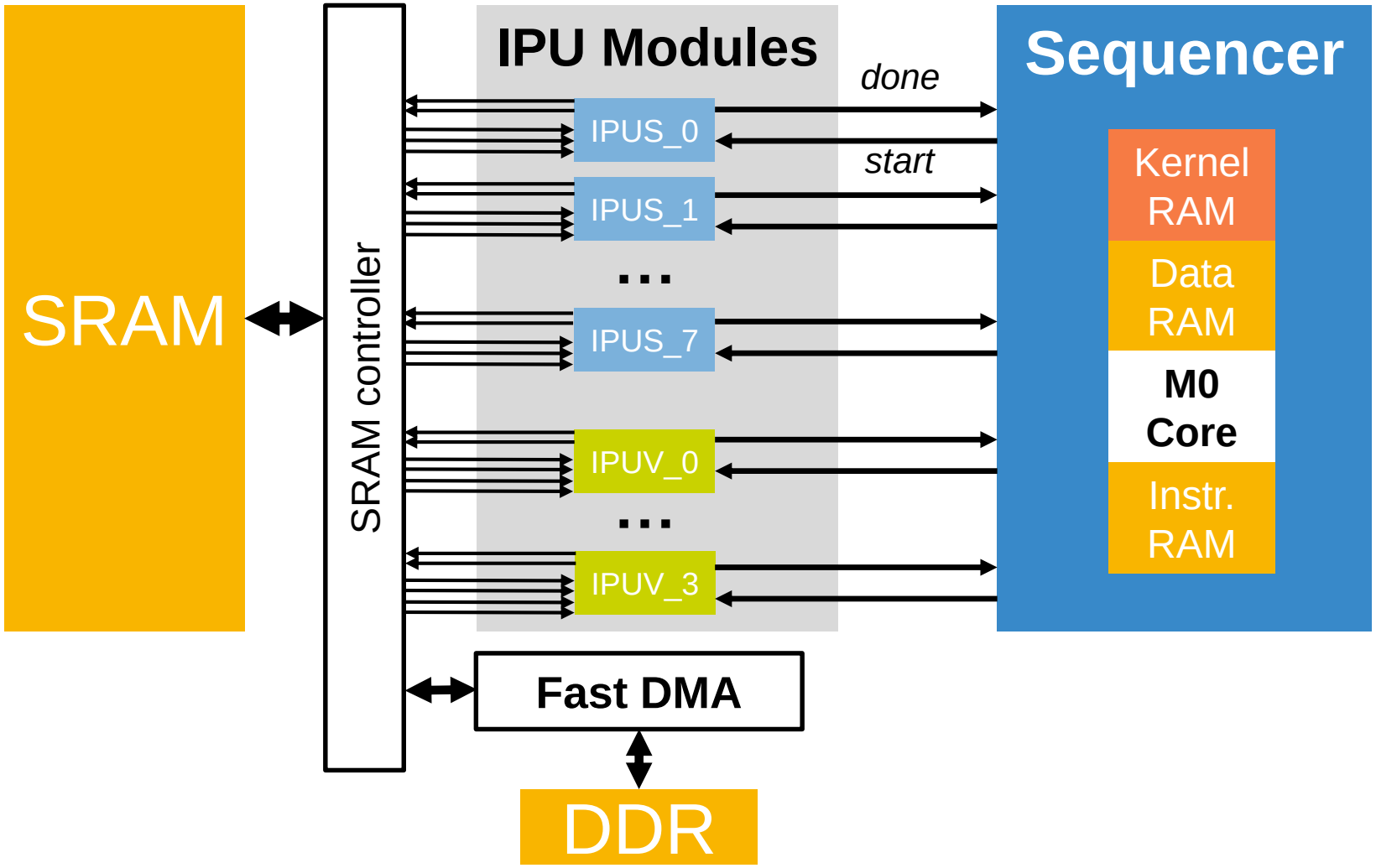
ISP Sub-system



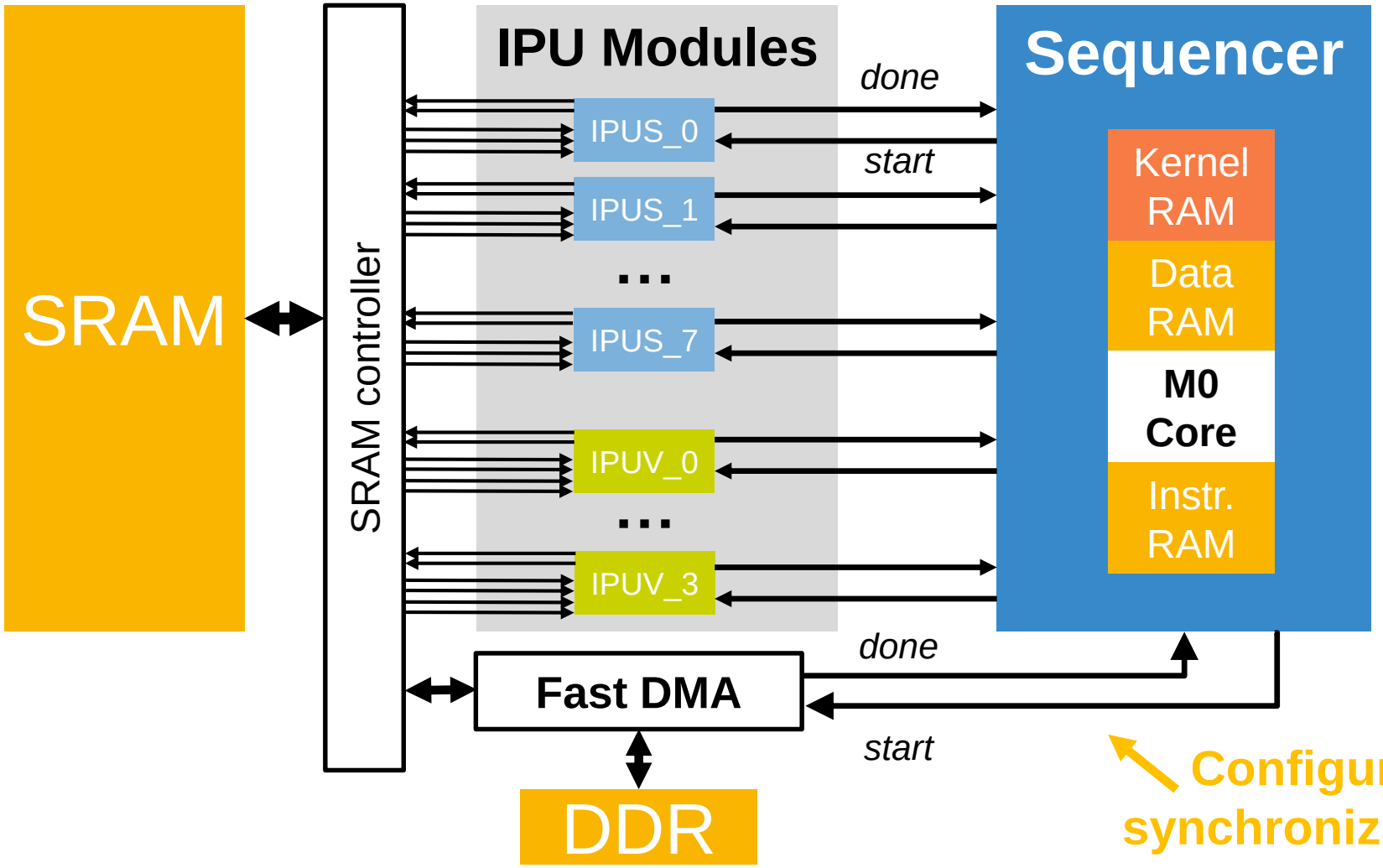
ISP Sub-system



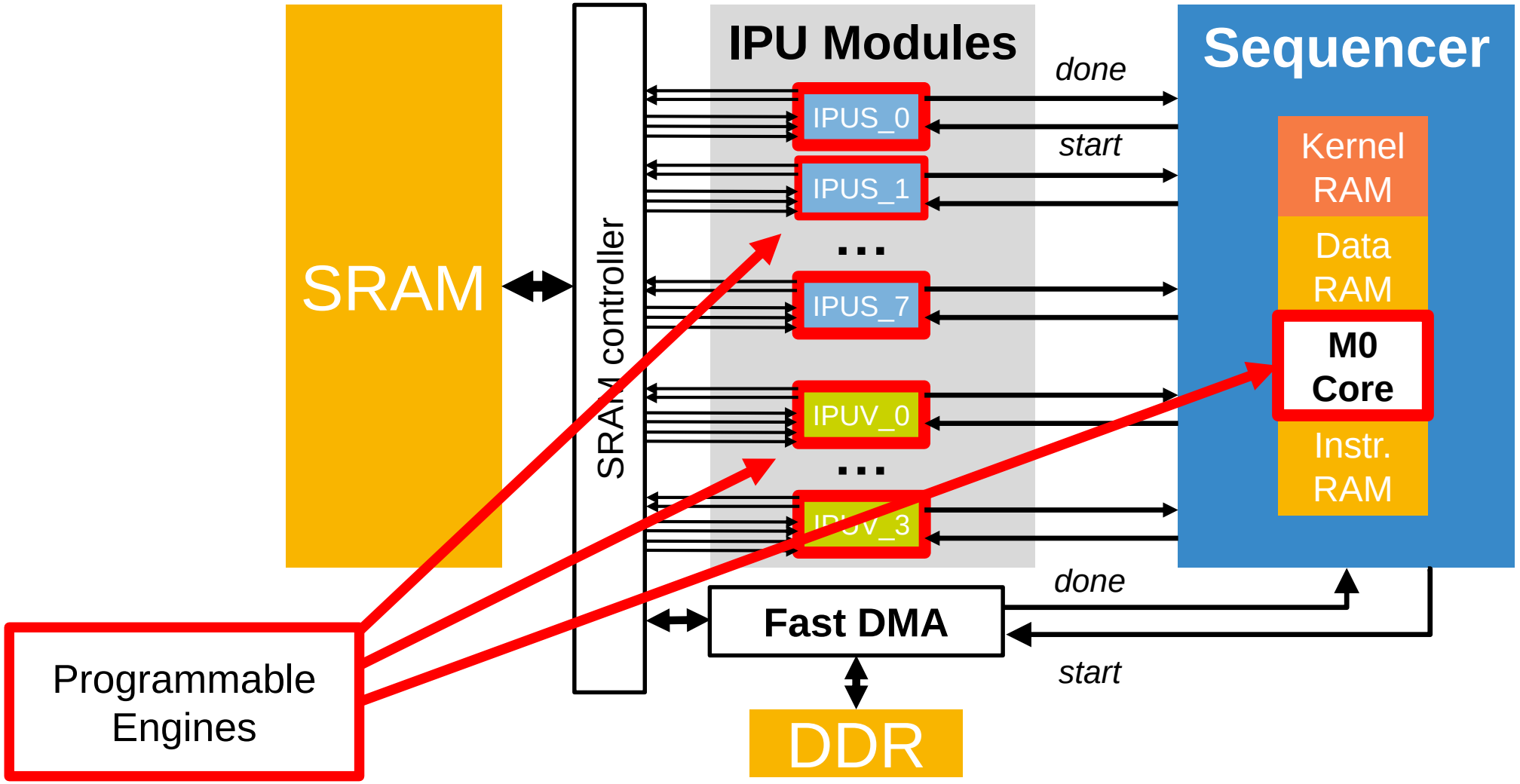
ISP Sub-system



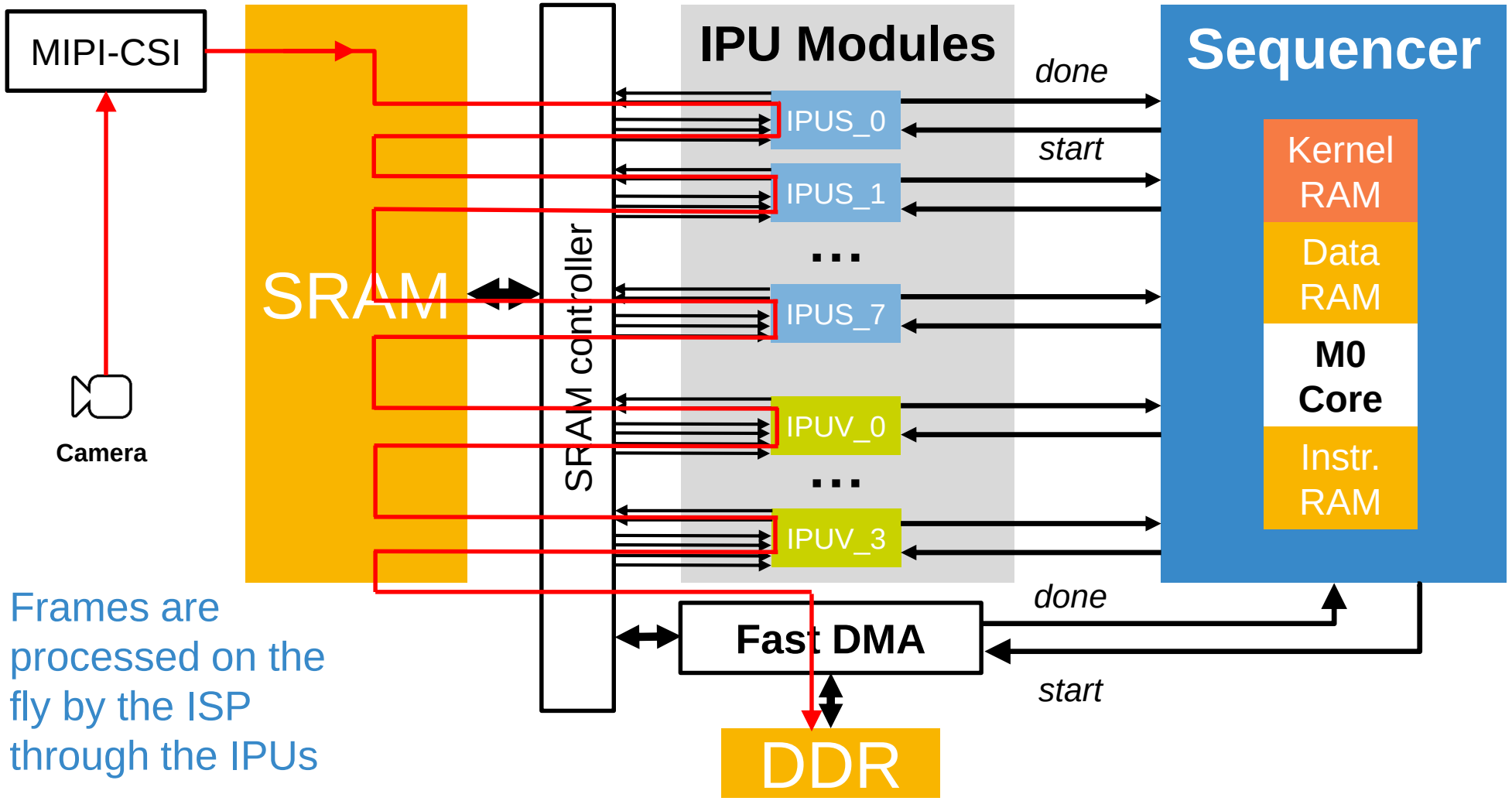
ISP Sub-system



ISP Sub-system



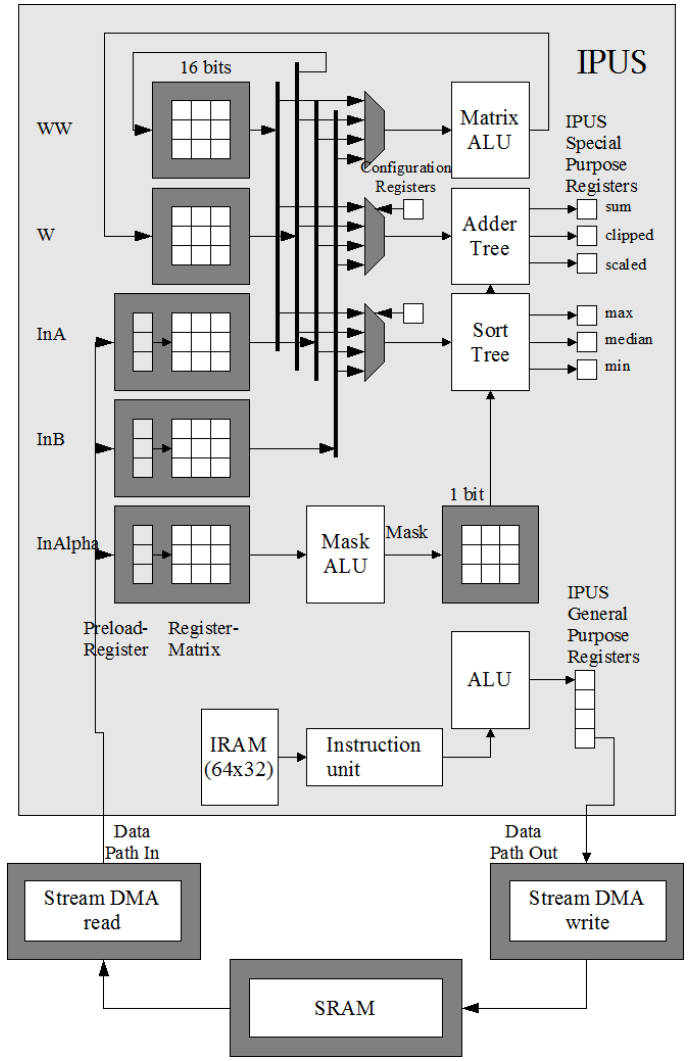
ISP Sub-system



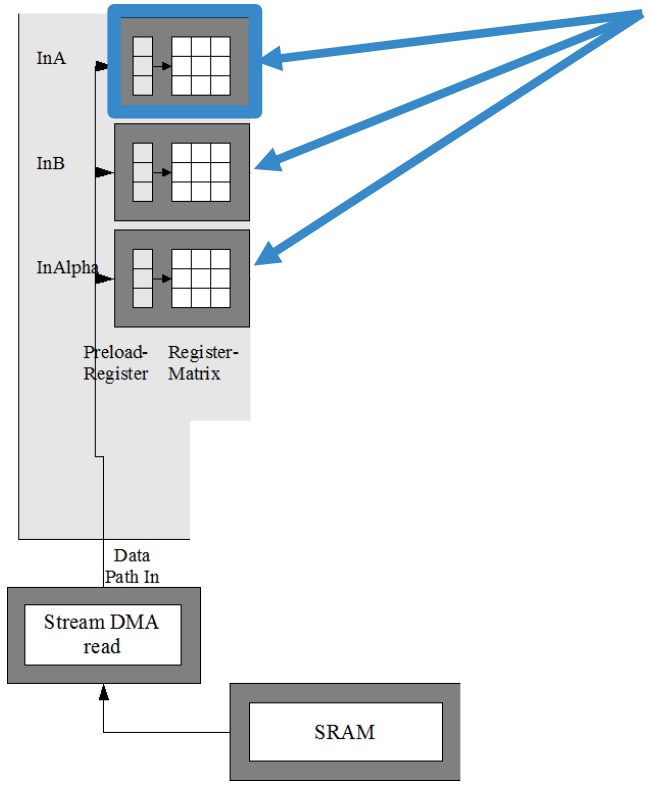
Frames are processed on the fly by the ISP through the IPUs

IPU

Scalar Engine (IPUS)



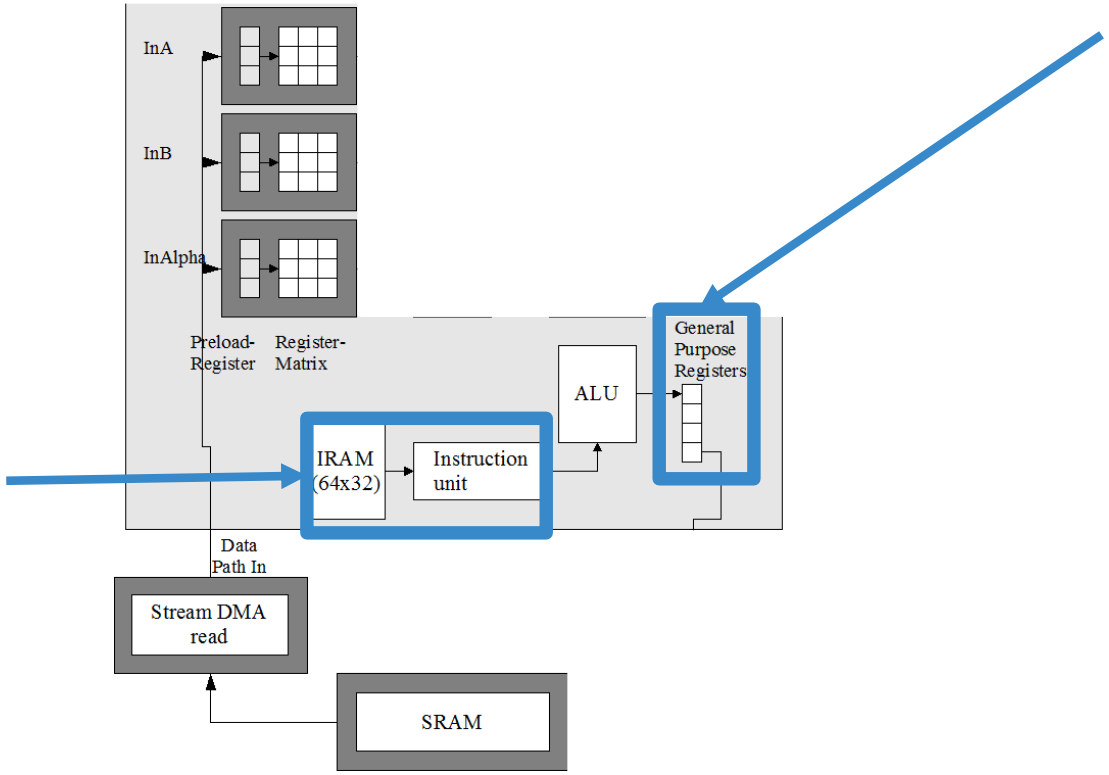
Scalar Engine (IPUS)



- Input window: 3x3 or 9x1
- Up to 3 Inputs:
  - Ex: R, G, B

Scalar Engine (IPUS)

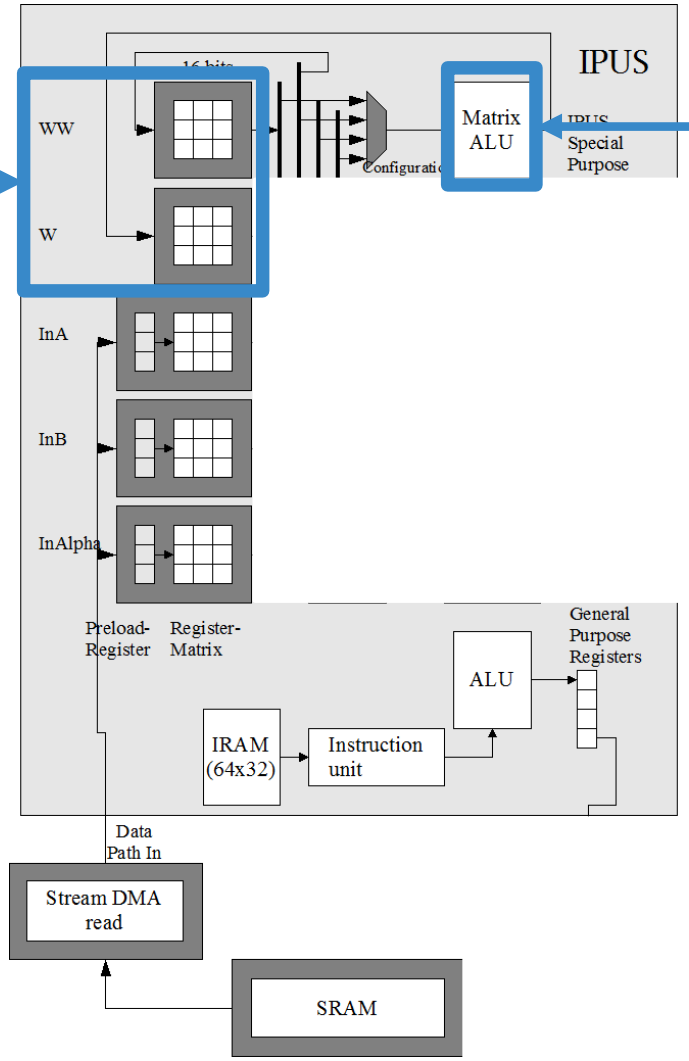
- No data memory (no load/store), only registers



- Instruction memory
- Decoding unit

Scalar Engine (IPUS)

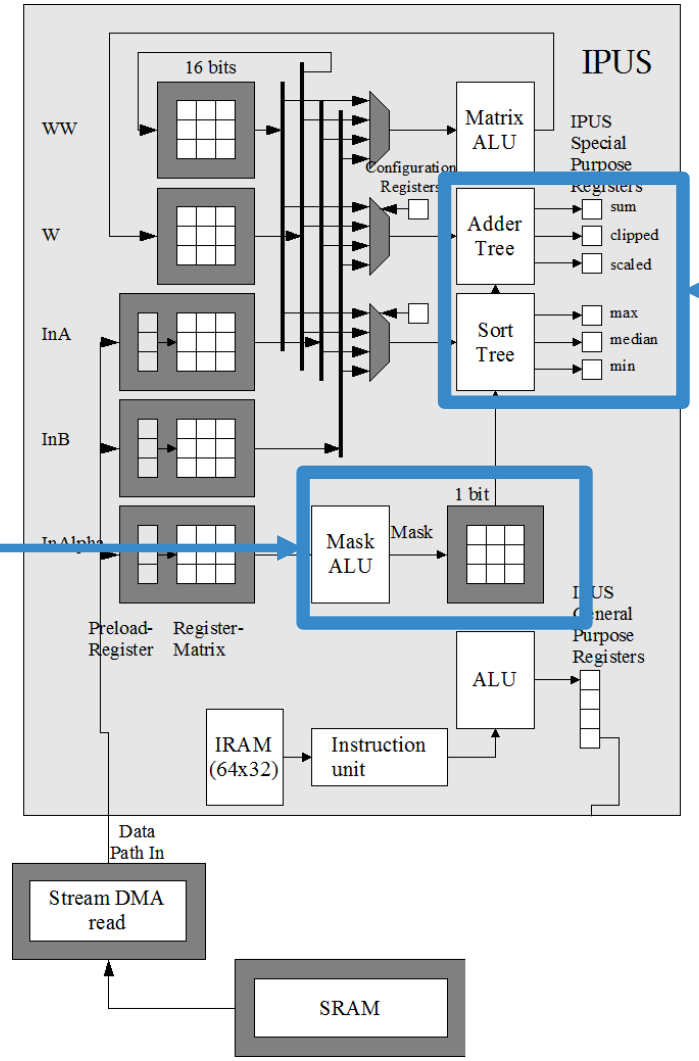
Working matrix



Matrix ALU (3x3 -> 3x3)

Scalar Engine (IPUS)

Mask matrix: To select on which elements of the matrix the operation or accelerator should be applied to



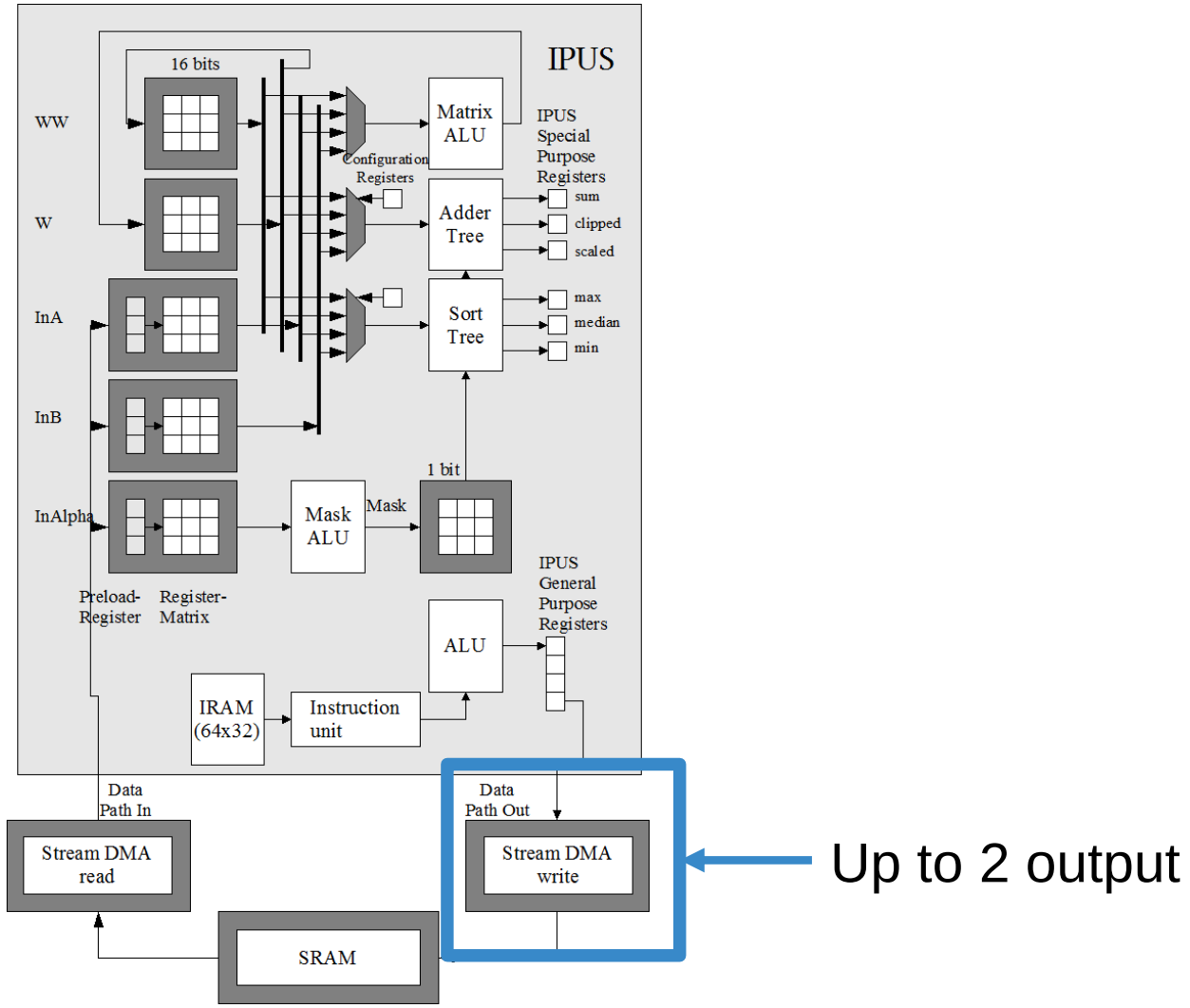
3X3 Accelerators:

- Sum, scaled, clipped
- Max, median, min

Other accelerators:

- Histogram
- PRNG

Scalar Engine (IPUS)

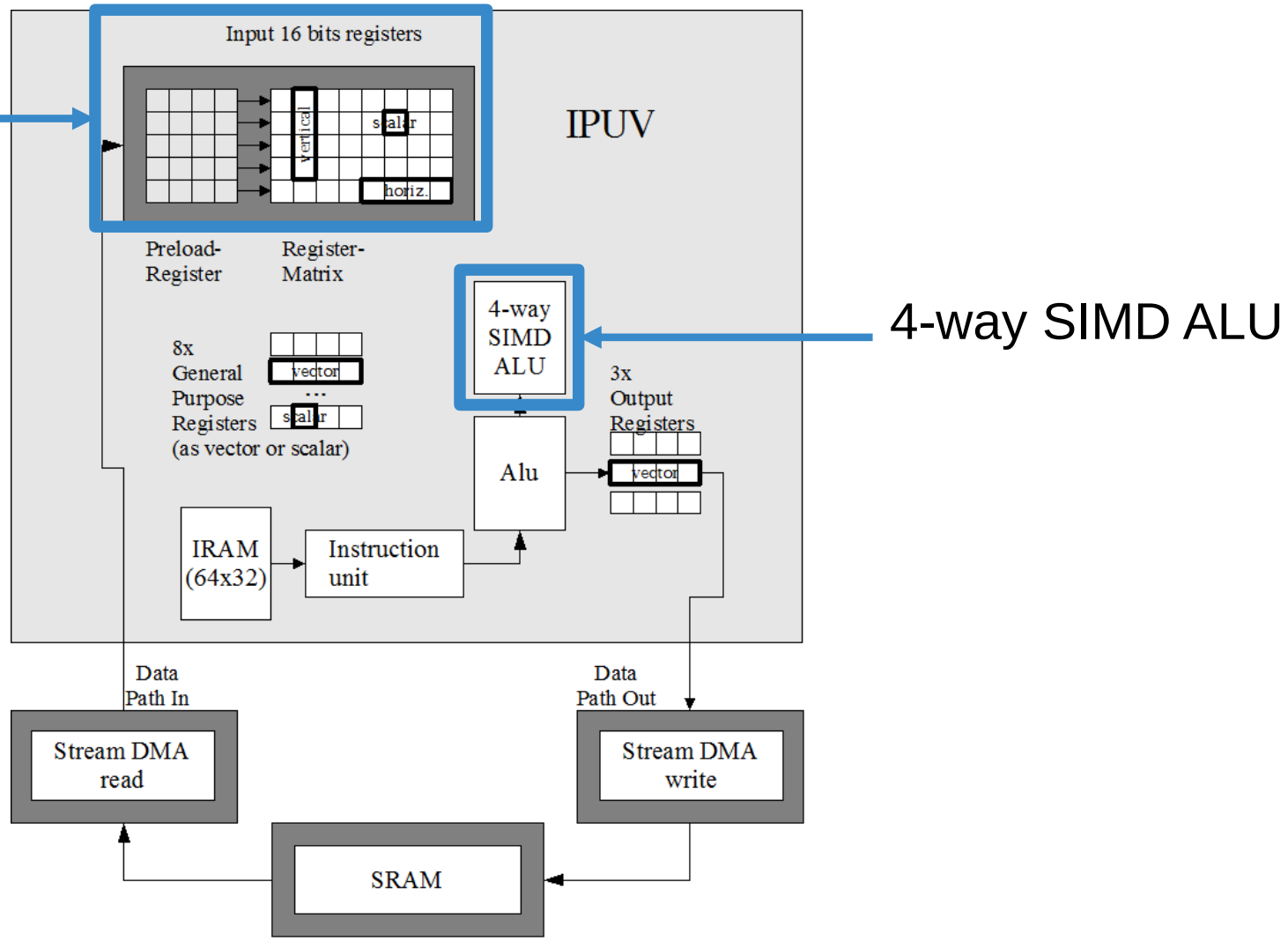


Vector Engine (IPUV)

5x5 input windows

Handles operation with:

- Horizontal vectors
- Vertical vectors
- Scalars



4-way SIMD ALU

IPU Engines

- 8 x IPUS & 4 x IPUV
- 500MHz per Engine: 6000MHz total
- Pixels are 16-bit fixed point data in signed or unsigned mode
- Input and output managed by the StreamDMA engines

Input Matrix: IPUS

In2	In1	In0
In4	In2	In3
In7	In6	In5

3x3

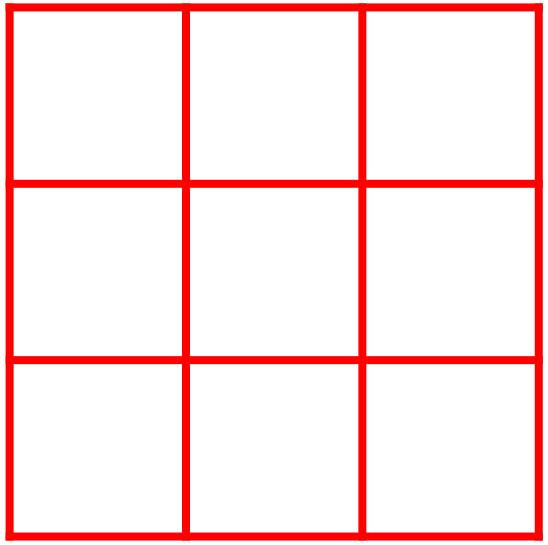
In8	In7	In6	In5	In4	In3	In2	In1	In0

9x1

Input Matrix: IPUS

Input image

Input matrix



Pix 0	Pix 1	Pix 2	Pix 3	Pix 4	Pix 5	Pix 6	Pix 7	Pix 8
Pix 9	Pix 10	Pix 11	Pix 12	Pix 13	Pix 14	Pix 15	Pix 16	Pix 17
Pix 18	Pix 19	Pix 20	Pix 21	Pix 22	Pix 23	Pix 24	Pix 25	Pix 26
Pix 27	Pix 28	Pix 29	Pix 30	Pix 31	Pix 32	Pix 33	Pix 34	Pix 35
Pix 36	Pix 37	Pix 38	Pix 39	Pix 40	Pix 41	Pix 42	Pix 43	Pix 44
Pix 45	Pix 46	Pix 47	Pix 48	Pix 49	Pix 50	Pix 51	Pix 52	Pix 53

Input Matrix: IPUS

		Pix 0	Pix 1	Pix 2	Pix 3	Pix 4	Pix 5	Pix 6	Pix 7	Pix 8
		Pix 9	Pix 10	Pix 11	Pix 12	Pix 13	Pix 14	Pix 15	Pix 16	Pix 17
		Pix 18	Pix 19	Pix 20	Pix 21	Pix 22	Pix 23	Pix 24	Pix 25	Pix 26
			Pix 27	Pix 28	Pix 29	Pix 30	Pix 31	Pix 32	Pix 33	Pix 34	Pix 35
			Pix 36	Pix 37	Pix 38	Pix 39	Pix 40	Pix 41	Pix 42	Pix 43	Pix 44
			Pix 45	Pix 46	Pix 47	Pix 48	Pix 49	Pix 50	Pix 51	Pix 52	Pix 53

1st Input

Input Matrix: IPUS

	Pix 0	**Pix 1**	Pix 2	Pix 3	Pix 4	Pix 5	Pix 6	Pix 7	Pix 8
	Pix 9	**Pix 10**	Pix 11	Pix 12	Pix 13	Pix 14	Pix 15	Pix 16	Pix 17
	Pix 18	**Pix 19**	Pix 20	Pix 21	Pix 22	Pix 23	Pix 24	Pix 25	Pix 26
Pix 27	Pix 28	Pix 29	Pix 30	Pix 31	Pix 32	Pix 33	Pix 34	Pix 35	
Pix 36	Pix 37	Pix 38	Pix 39	Pix 40	Pix 41	Pix 42	Pix 43	Pix 44	
Pix 45	Pix 46	Pix 47	Pix 48	Pix 49	Pix 50	Pix 51	Pix 52	Pix 53	

2nd Input

Input Matrix: IPUS

… 5th input

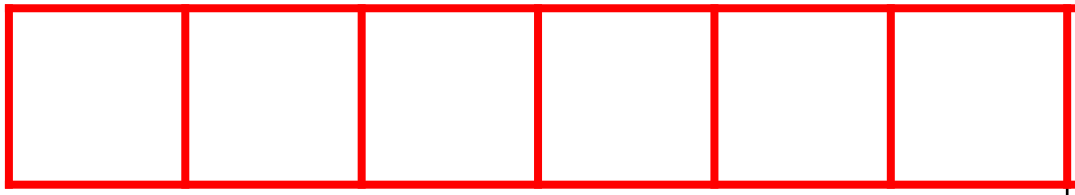
Pix 0	Pix 1	**Pix 2**	**Pix 3**	**Pix 4**	Pix 5	Pix 6	Pix 7	Pix 8
Pix 9	Pix 10	**Pix 11**	**Pix 12**	**Pix 13**	Pix 14	Pix 15	Pix 16	Pix 17
Pix 18	Pix 19	**Pix 20**	**Pix 21**	**Pix 22**	Pix 23	Pix 24	Pix 25	Pix 26
Pix 27	Pix 28	Pix 29	Pix 30	Pix 31	Pix 32	Pix 33	Pix 34	Pix 35
Pix 36	Pix 37	Pix 38	Pix 39	Pix 40	Pix 41	Pix 42	Pix 43	Pix 44
Pix 45	Pix 46	Pix 47	Pix 48	Pix 49	Pix 50	Pix 51	Pix 52	Pix 53

Input Matrix: IPUS

			Pix 0	Pix 1	Pix 2	Pix 3	Pix 4	Pix 5	Pix 6	Pix 7	Pix 8
			Pix 9	Pix 10	Pix 11	Pix 12	Pix 13	Pix 14	Pix 15	Pix 16	Pix 17
			Pix 18	Pix 19	Pix 20	Pix 21	Pix 22	Pix 23	Pix 24	Pix 25	Pix 26
			Pix 27	Pix 28	Pix 29	Pix 30	Pix 31	Pix 32	Pix 33	Pix 34	Pix 35
			Pix 36	Pix 37	Pix 38	Pix 39	Pix 40	Pix 41	Pix 42	Pix 43	Pix 44
			Pix 45	Pix 46	Pix 47	Pix 48	Pix 49	Pix 50	Pix 51	Pix 52	Pix 53

1st Input of the 2nd line

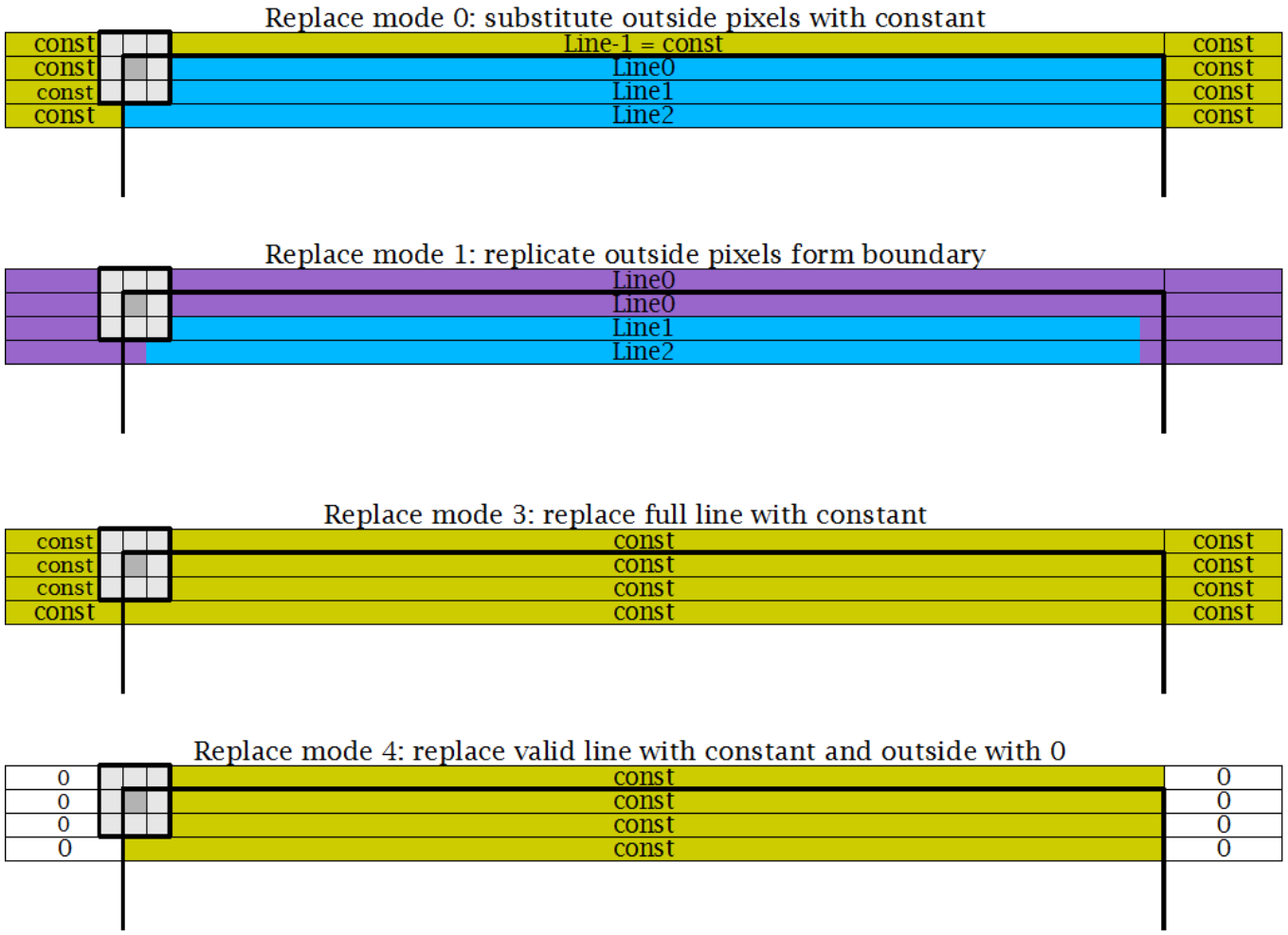
Input Matrix: IPUS



						Pix 0	**Pix 1**	**Pix 2**	Pix 3	Pix 4	Pix 5
						Pix 9	Pix 10	Pix 11	Pix 12	Pix 13	Pix 14
						Pix 18	Pix 19	Pix 20	Pix 21	Pix 22	Pix 23
						Pix 27	Pix 28	Pix 29	Pix 30	Pix 31	Pix 32
						Pix 36	Pix 37	Pix 38	Pix 39	Pix 40	Pix 41
						Pix 45	Pix 46	Pix 47	Pix 48	Pix 49	Pix 50

3x3 mode 1st Input

HW Replacement Modes for StreamDMA Engines

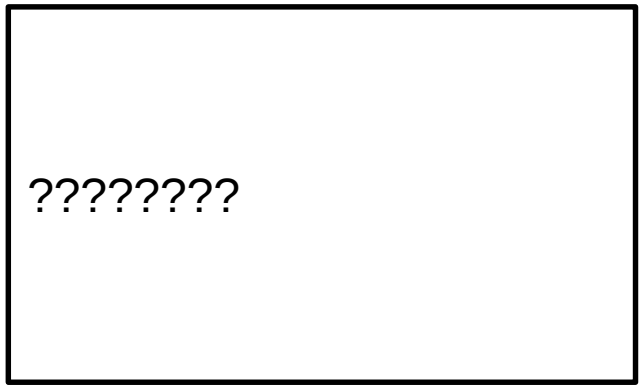


StreamDMA has a lot of additional features:

- Scaling
- Multiple data format support

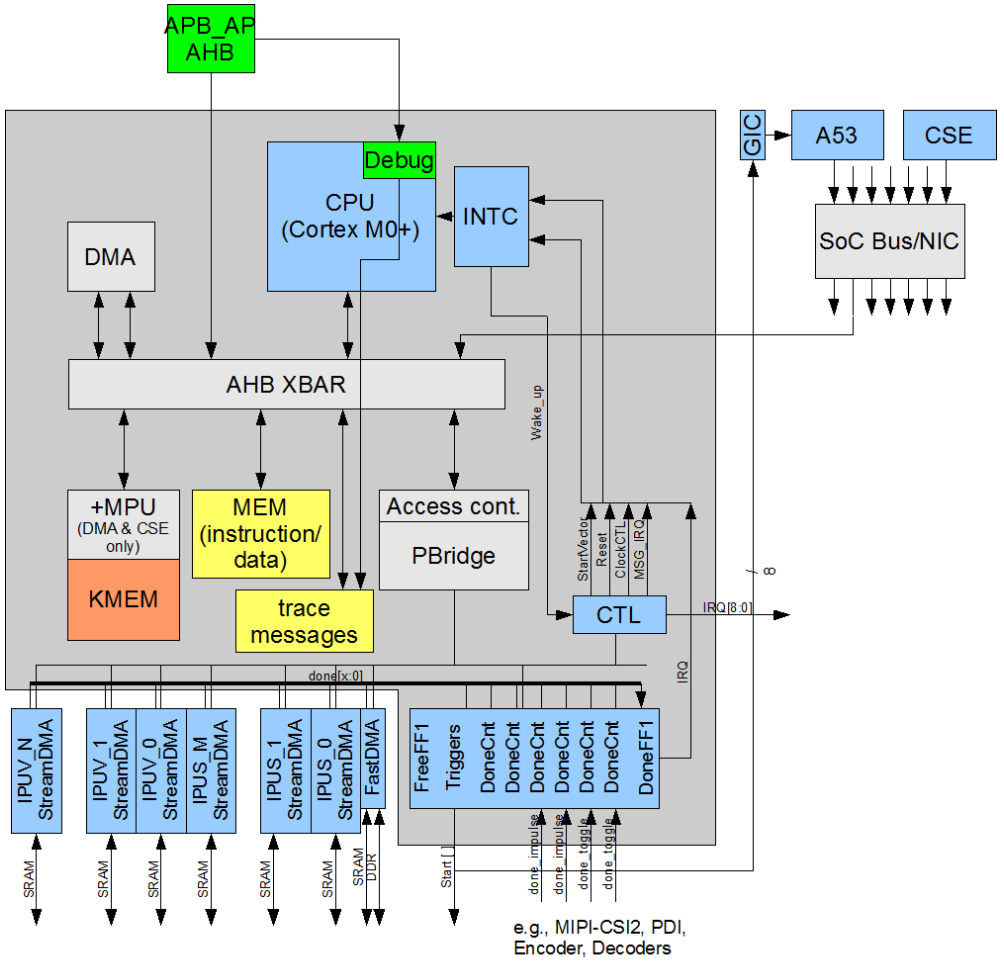
Input Matrix: IPUV

In2	In1	In0			
In4	In2	In3			
In7	In6	In5			



SEQUENCER

Sequencer: Block Diagram

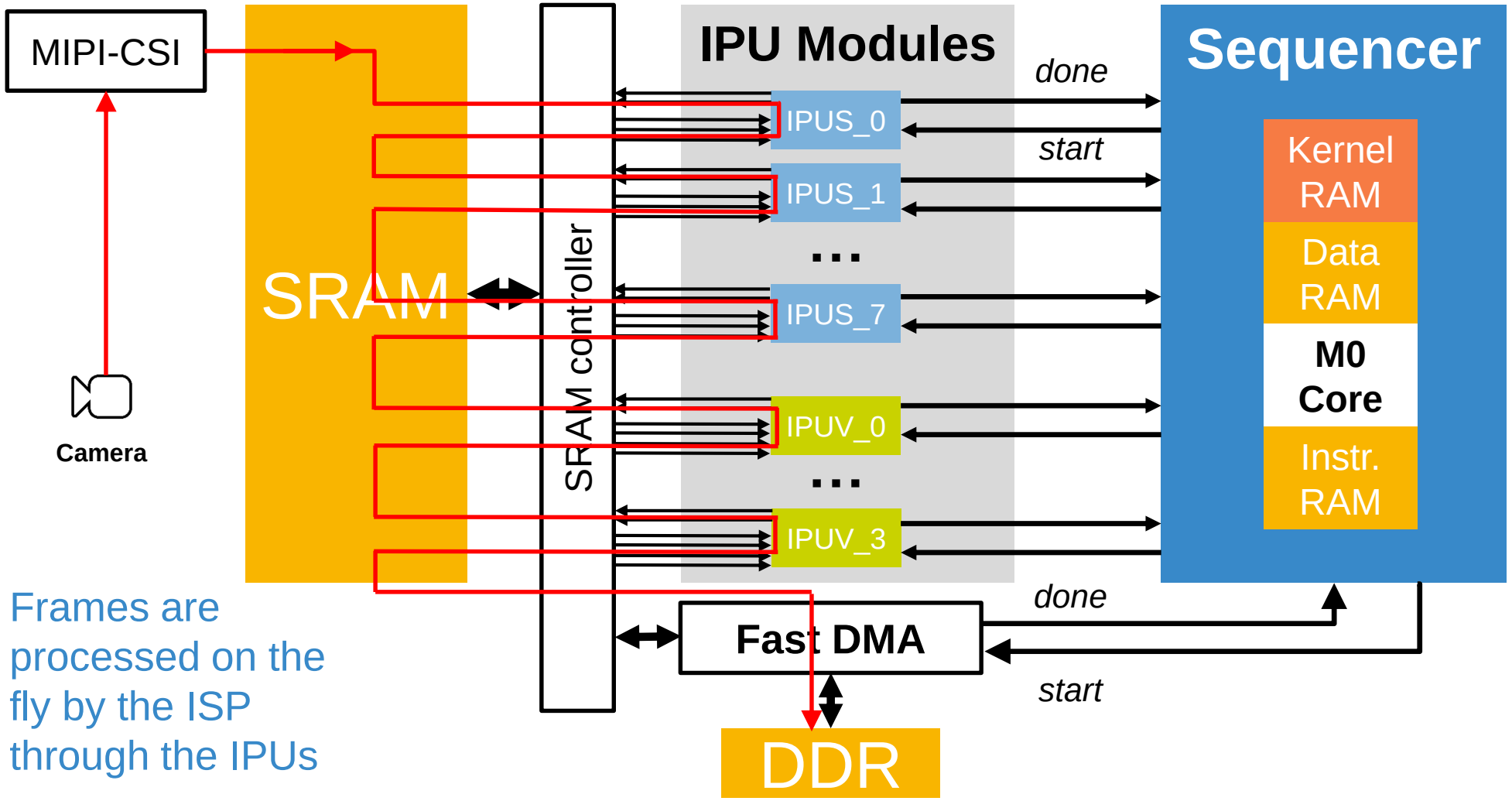


- Mapped into Host address space
- Peripherals can also be controlled by Host CPU: IPUs and FastDMA
- Servicing the Debug Unit for the IPUs

HOW TO PROGRAM IT

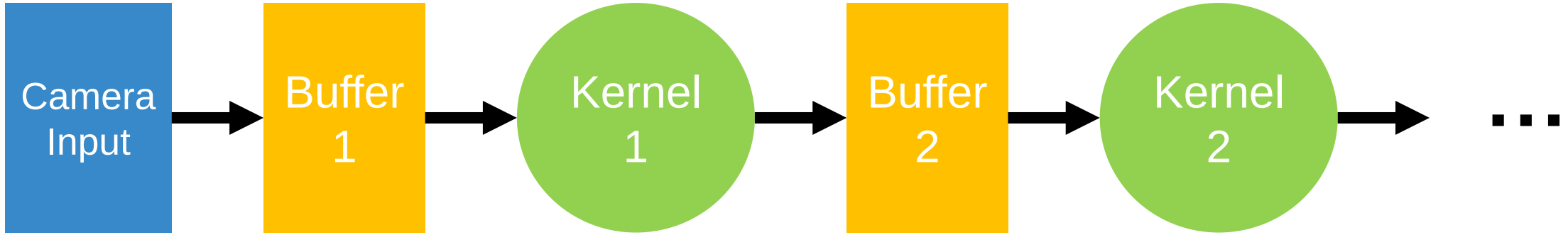
ISP GRAPH

ISP Sub-system

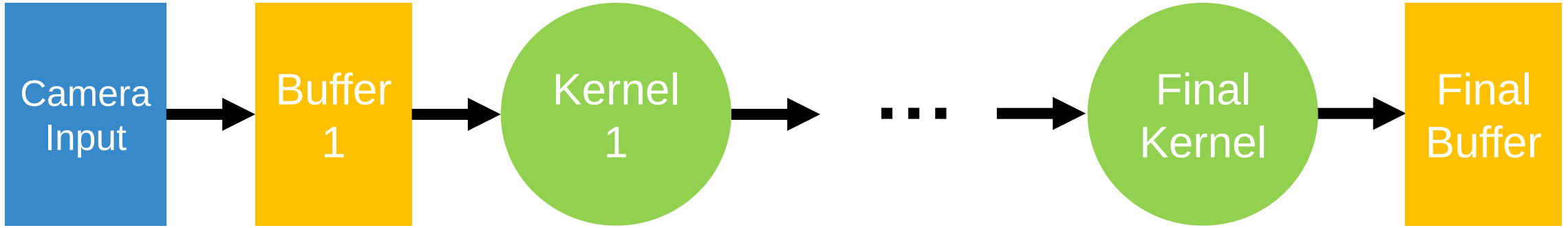


Frames are processed on the fly by the ISP through the IPUs

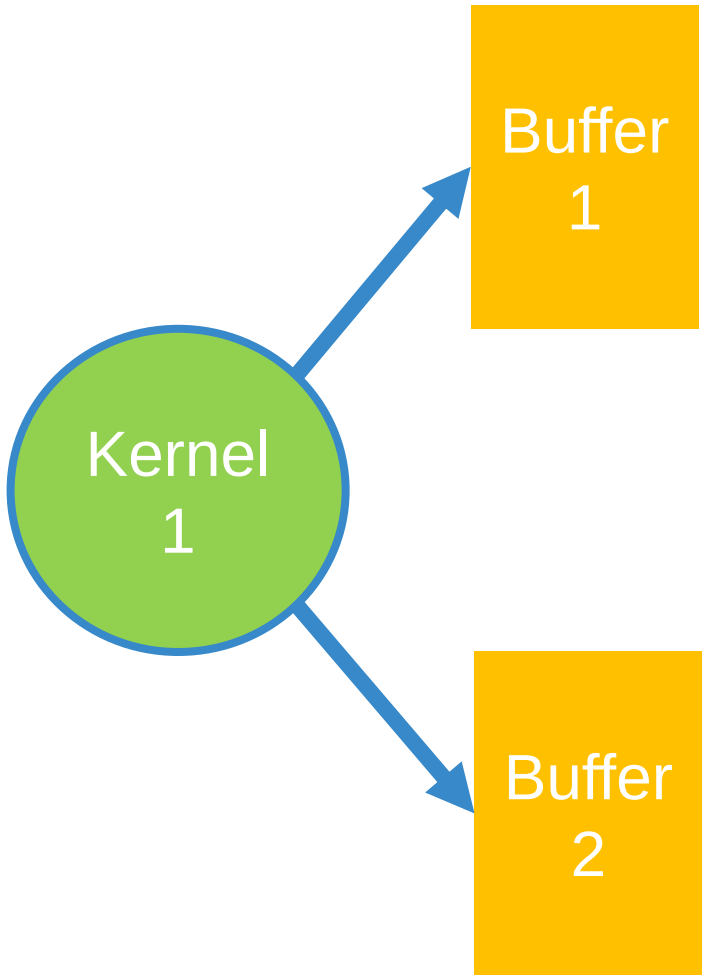
Graph



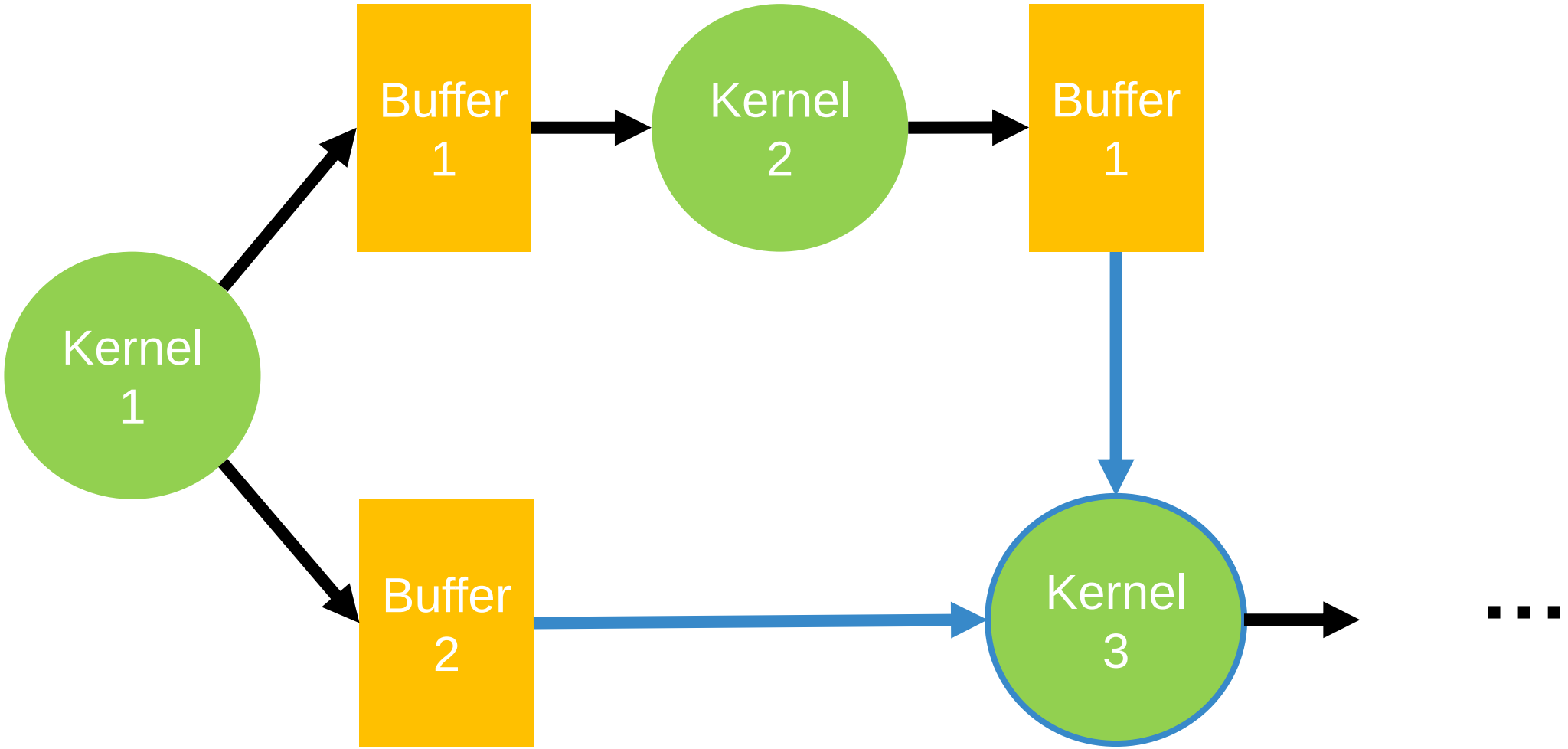
Graph



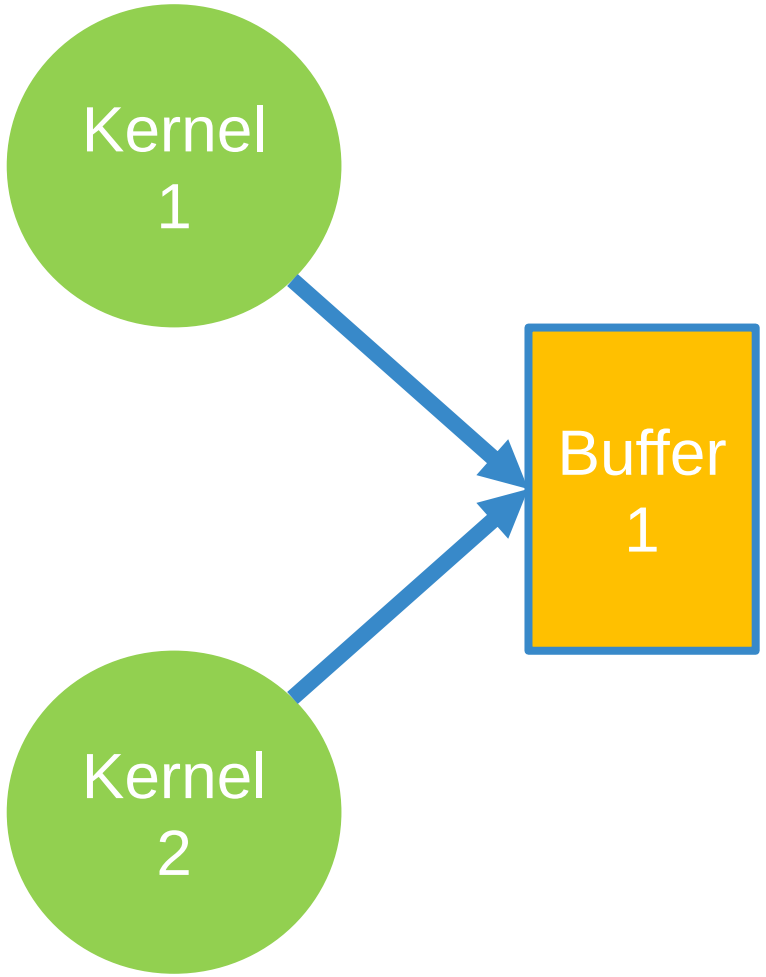
Graph



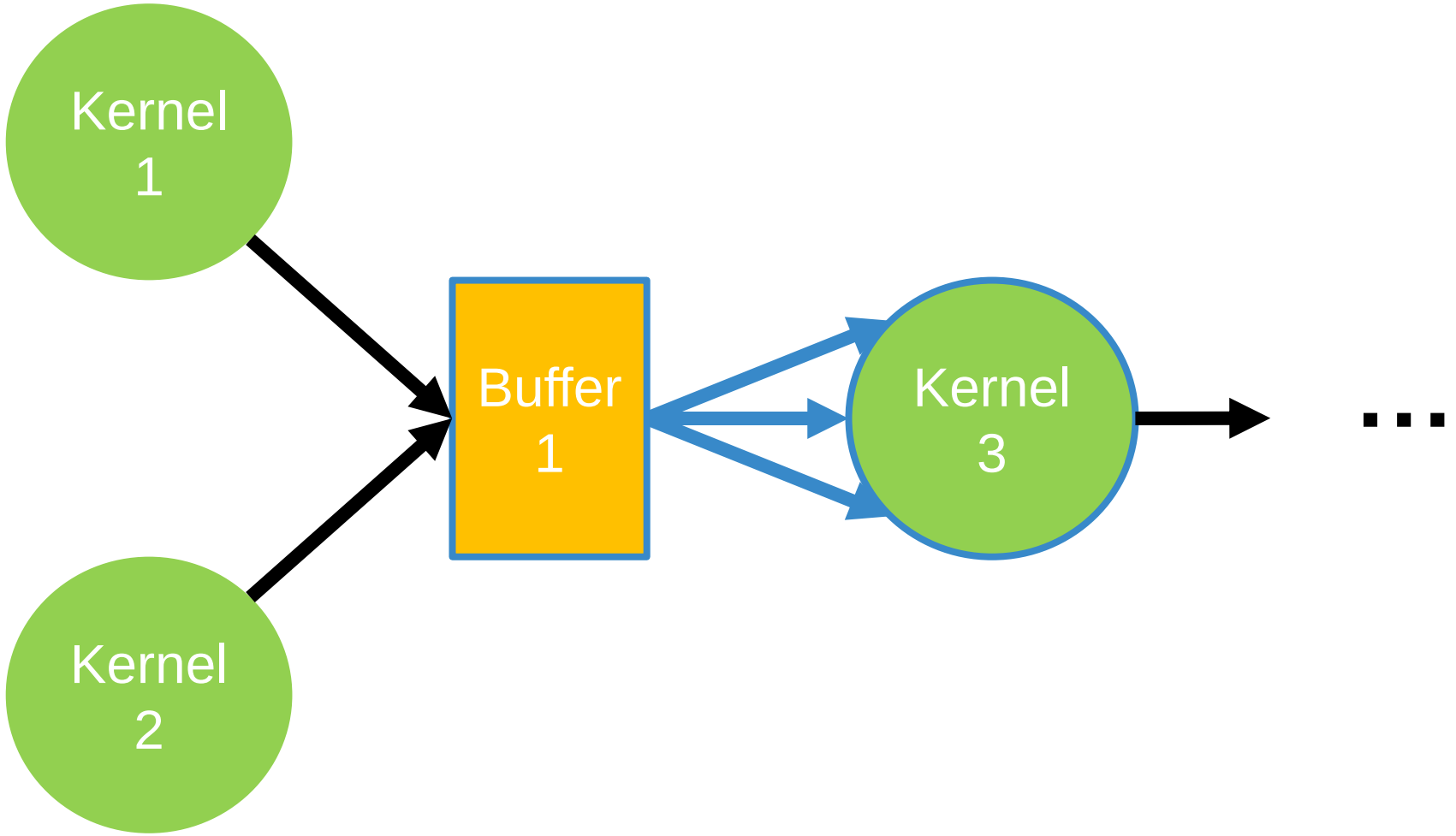
Graph



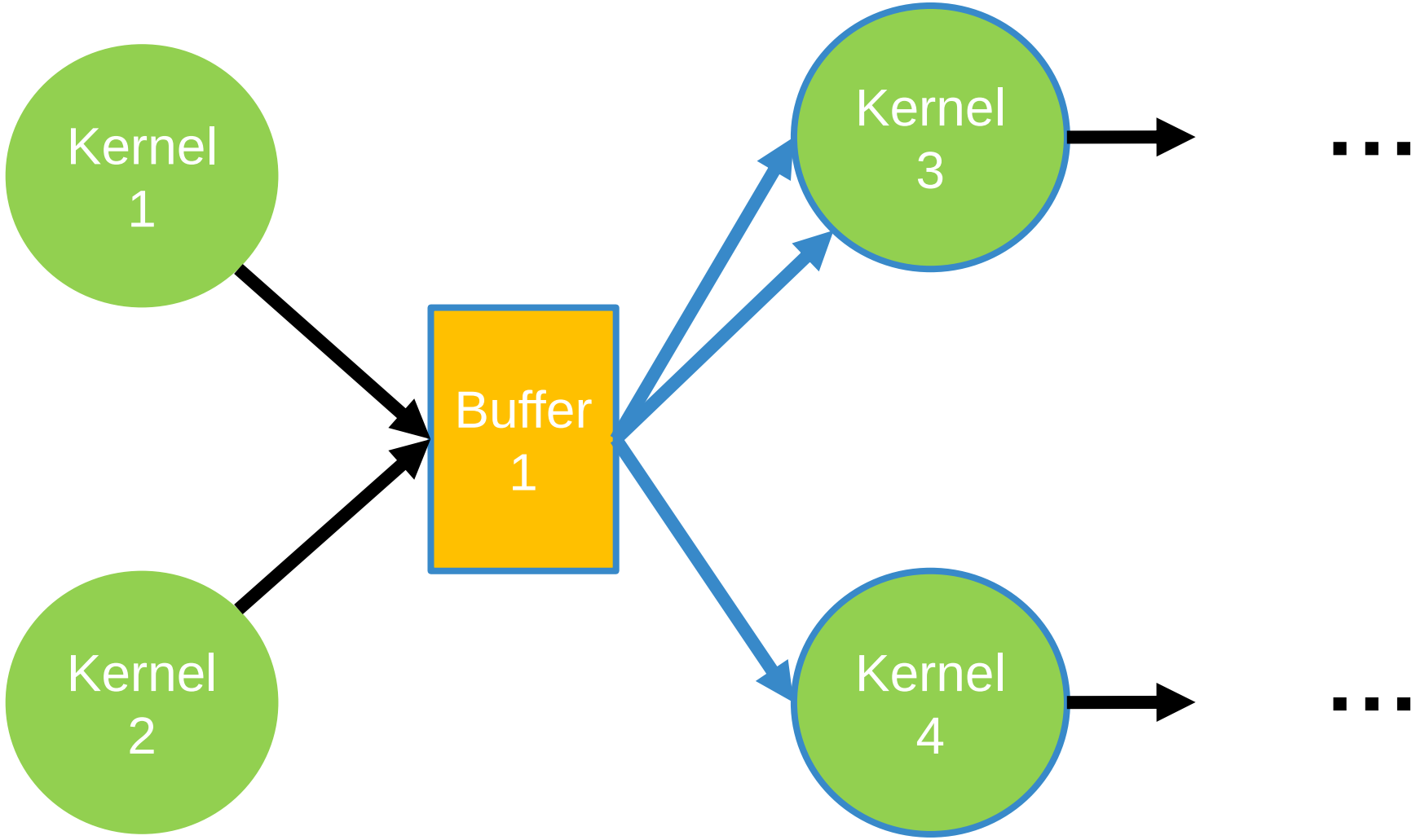
Graph



Graph

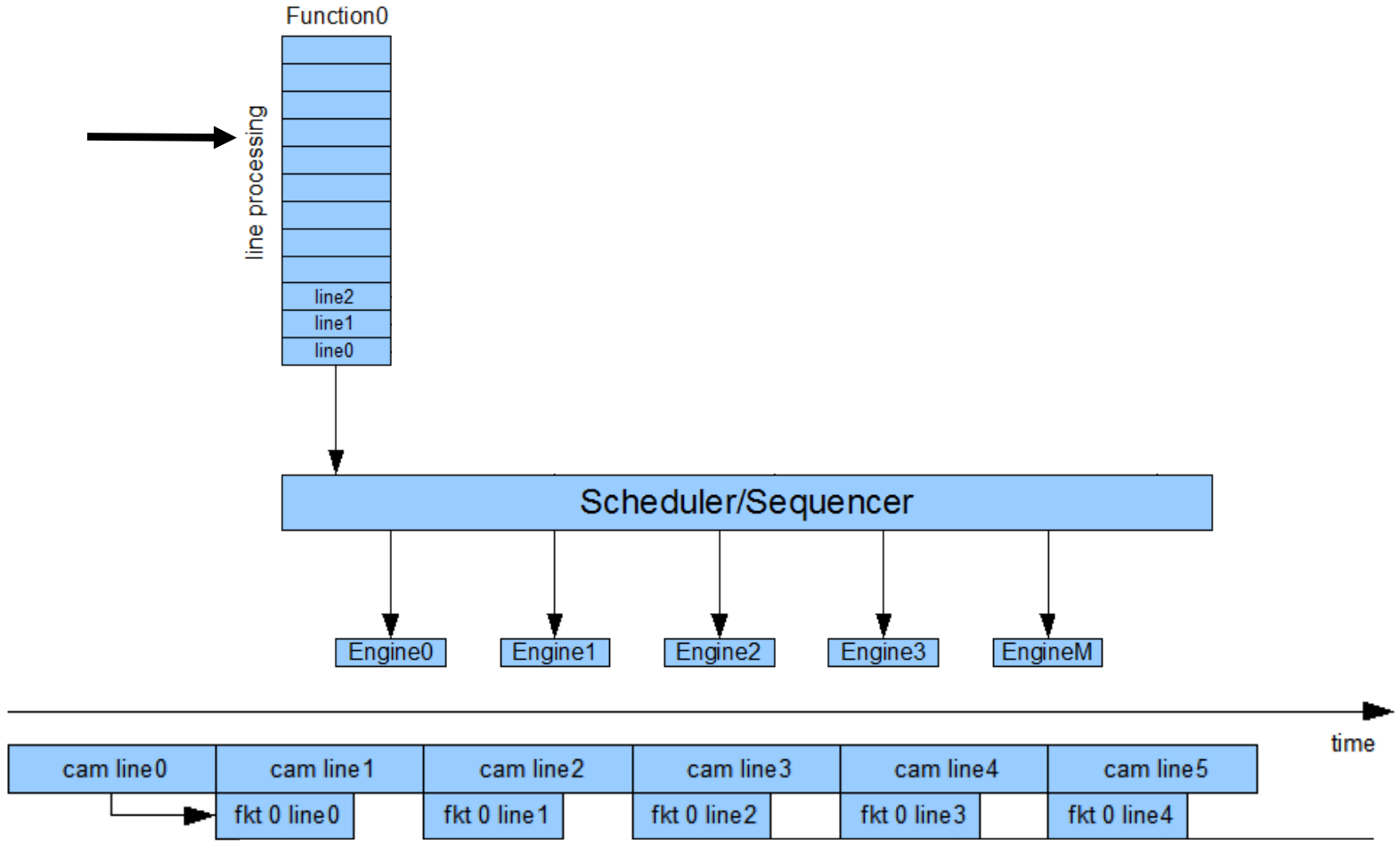


Graph



ISP Pipeline

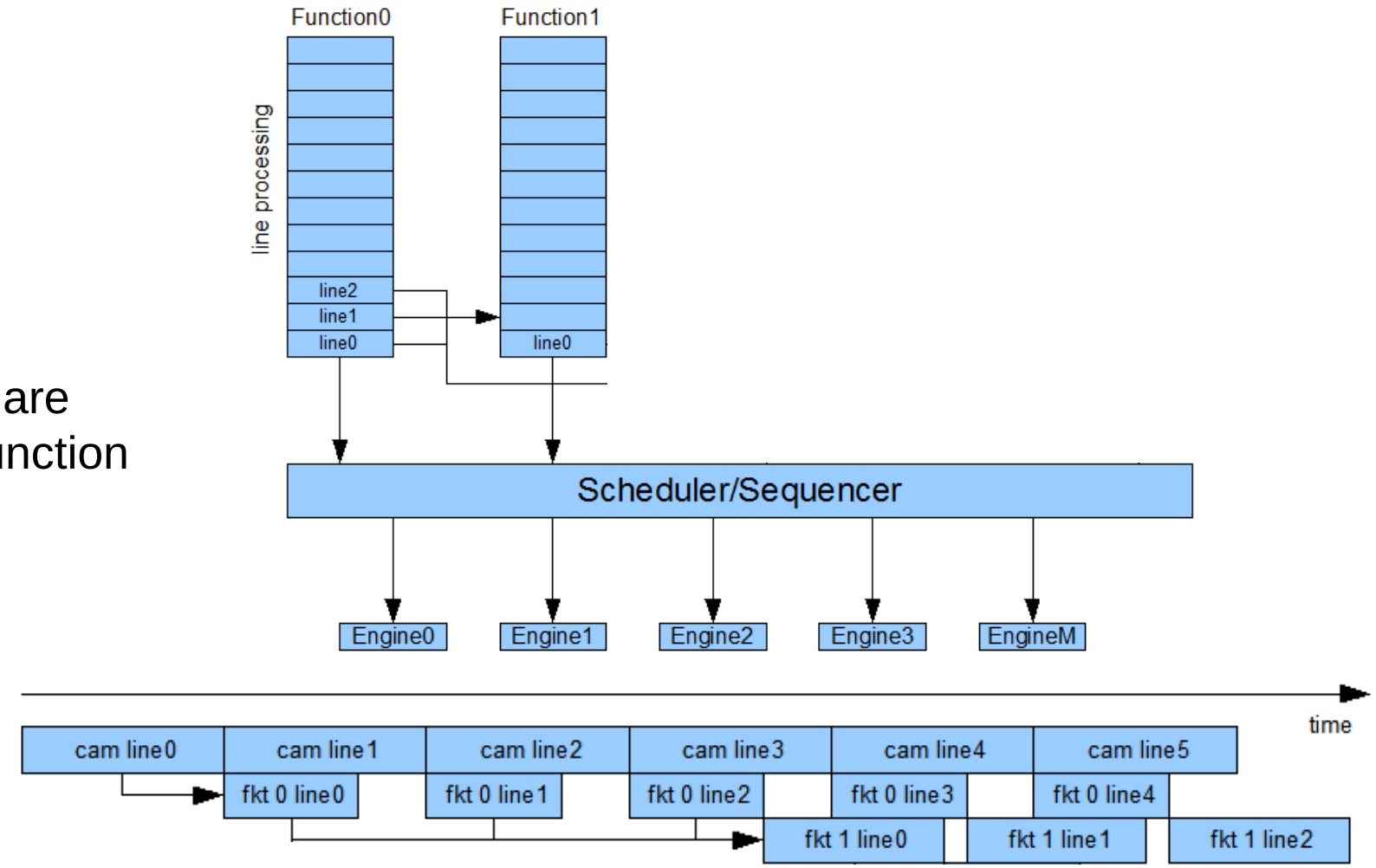
Output buffer of the kernel (function)



Function 0 requires one line in input

ISP Pipeline

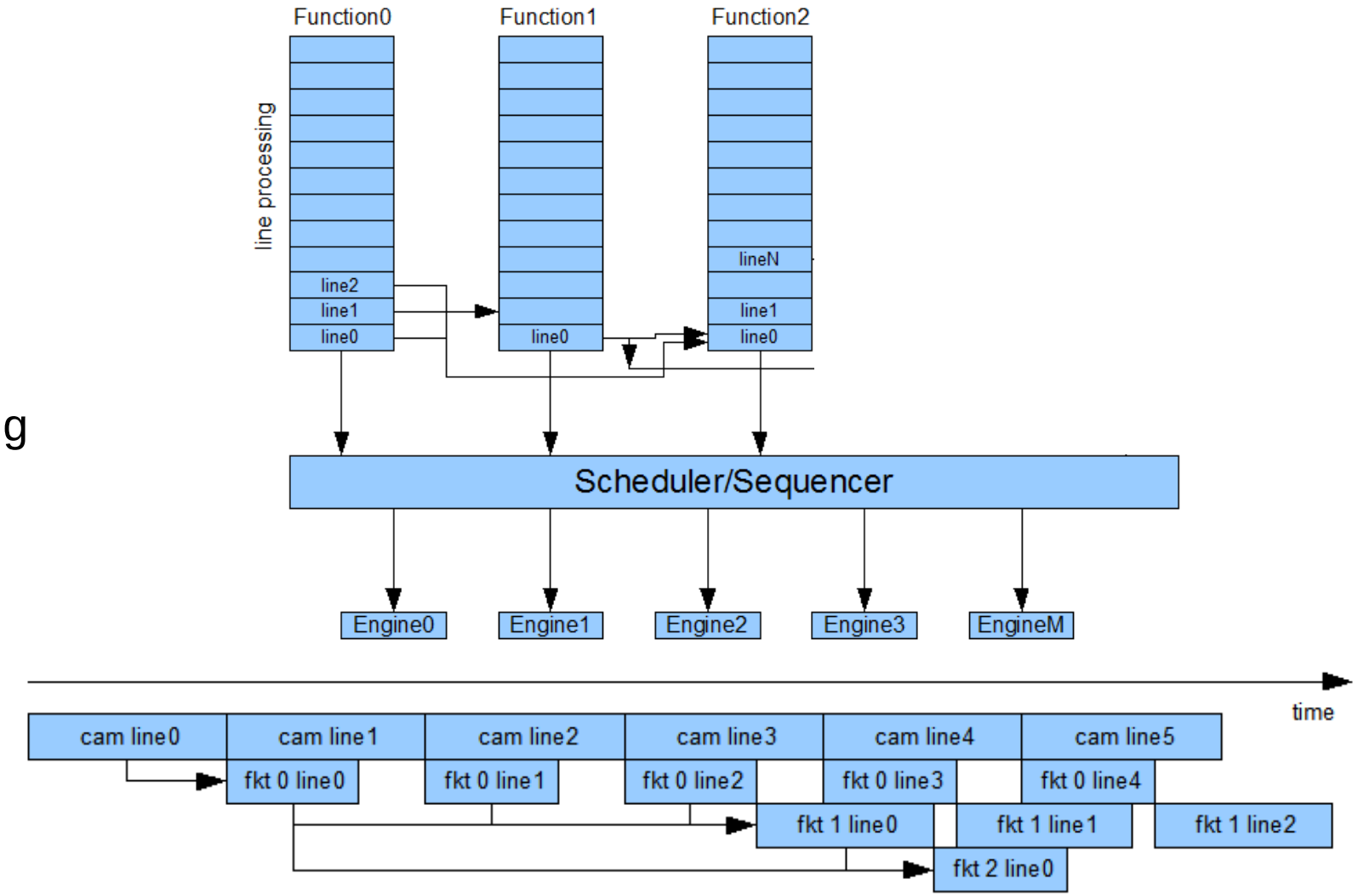
When enough lines are available the next function can be executed



Function 1 requires three lines in input

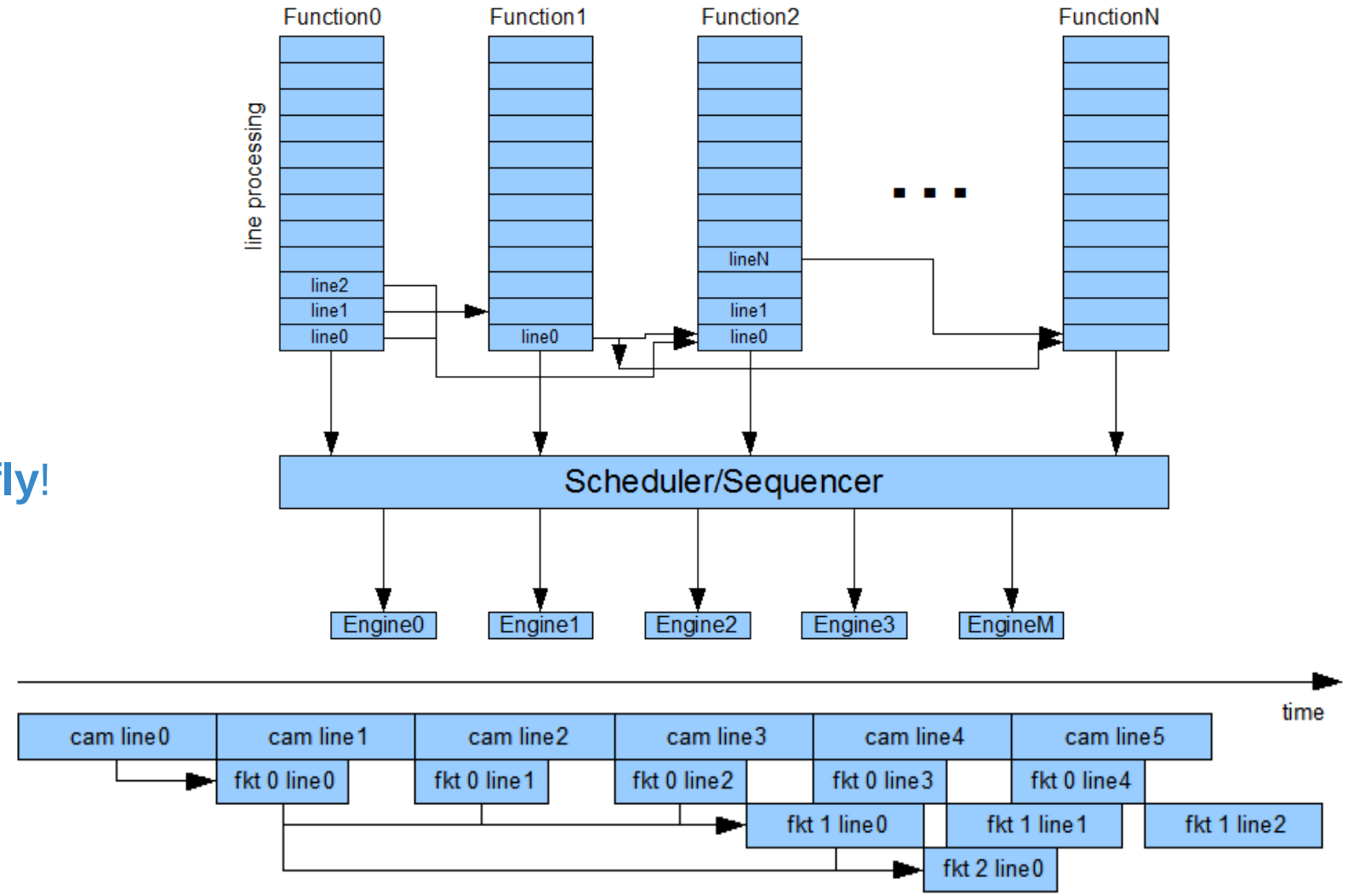
ISP Pipeline

Pipelined processing
on multiple engines



ISP Pipeline

The image can be **processed on the fly**!



GRAPH TOOL (DEMO)

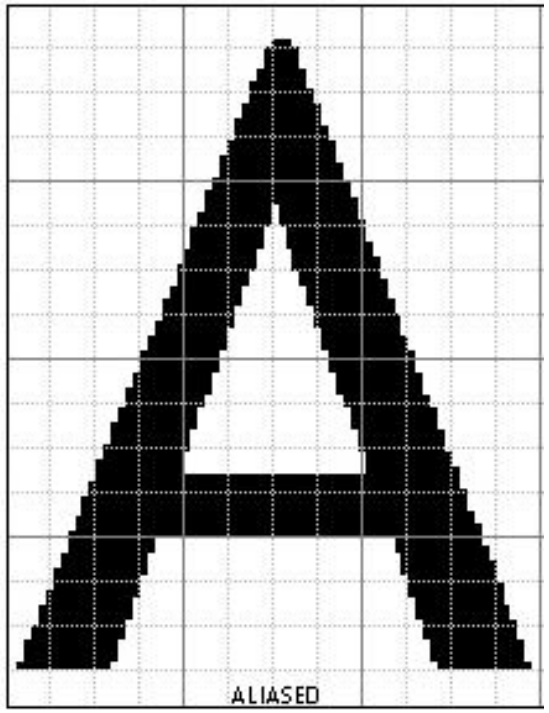
ISP KERNEL

Example of 2:1 Scaling

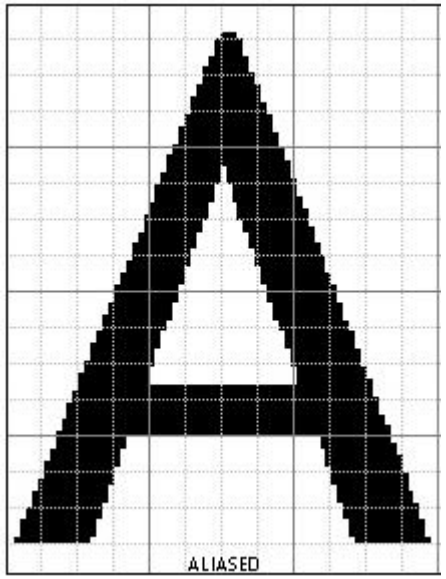
When simply copying
1 pixel over 2

[feature of the StreamDMA]

Aliasing problem



Example of 2:1 Scaling

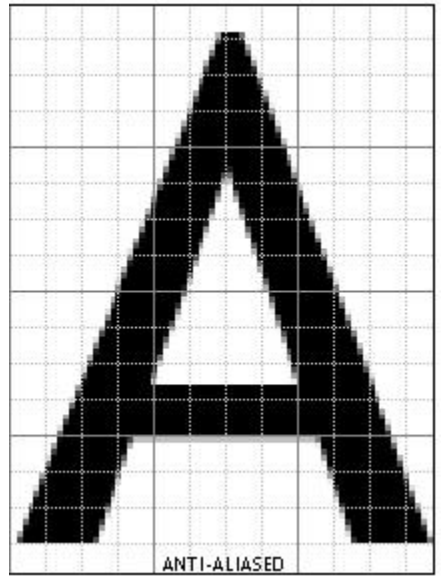


Aliased

1	2	1
2	4	2
1	2	1

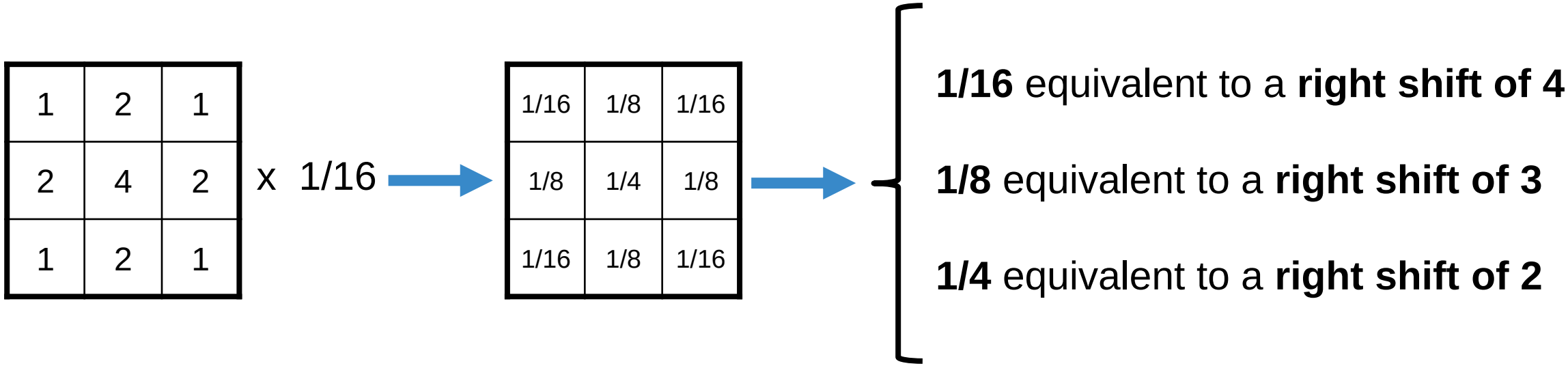
3x3 Gaussian filter
Low pass filter

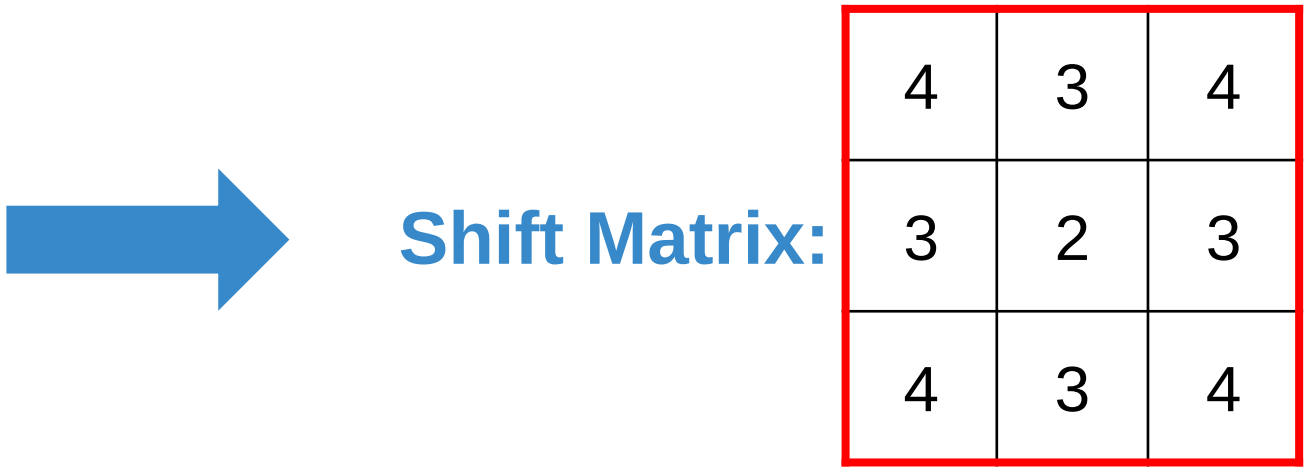
x 1/16



Anti-aliased

Example of 2:1 Scaling





Example of 2:1 Scaling

Code: IPU configuration

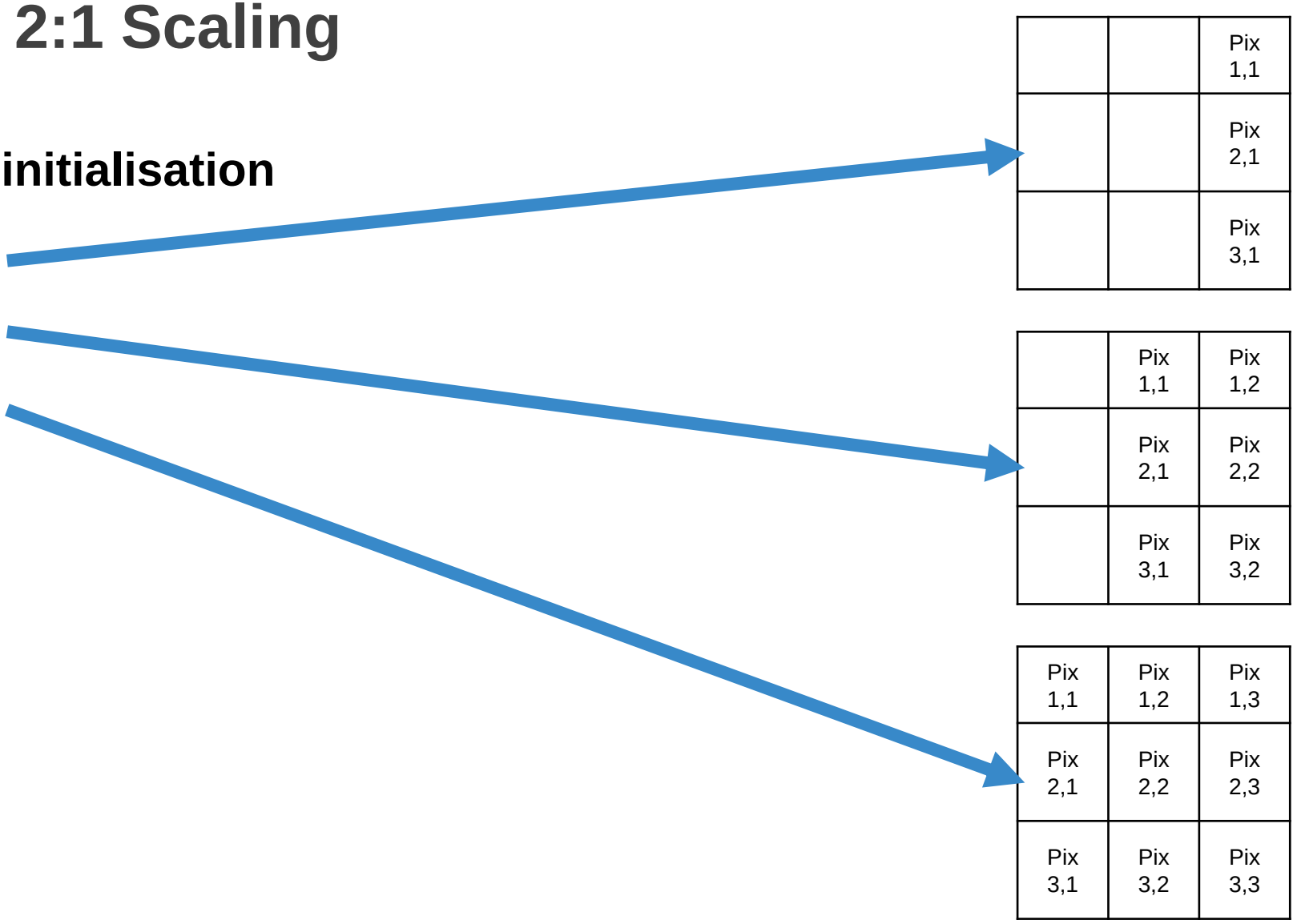
```
start_scale:
        mov confalu, (0 /*unsigned*/ |
                      (1<<1) /*saturate*/)

       mov confaddt,(0 /*w*/ |
                     (0<<3) /*unsigned*/ |
                     (5<<5) /*shift*/ |
                     (0x40 << 9) /*
scale*/)
```

Confalu and Confaddt are core registers of the IPU

Example of 2:1 Scaling

Code: Input initialisation

```
    done d0,i
d0:
    done d1,i
d1:
    done d2,i
```

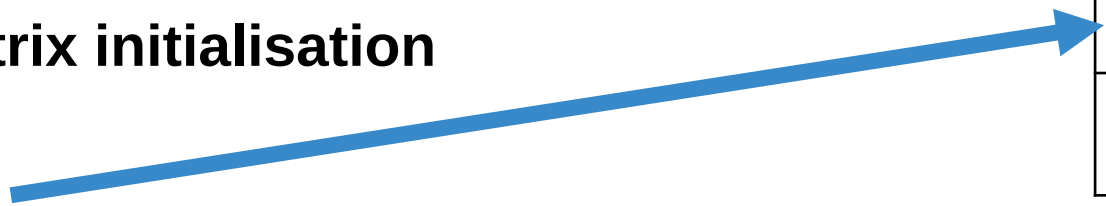


Input matrix InA

Example of 2:1 Scaling

Code: Shifting matrix initialisation

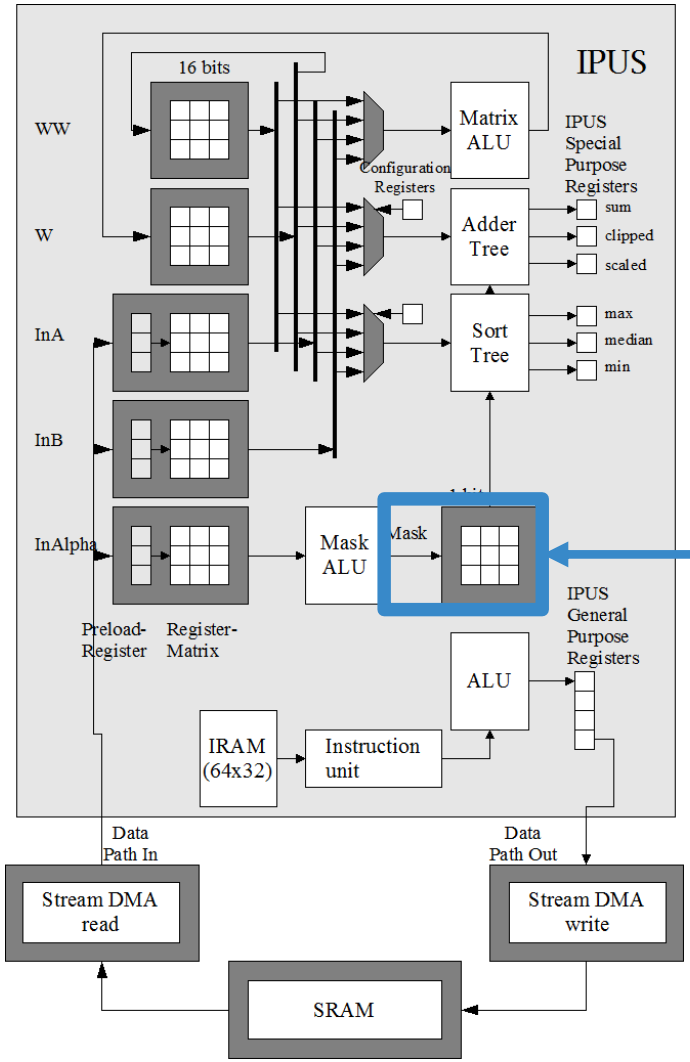
```
d2:
    mset 0b101000101
    mov 4               // to w
    mset 0b010101010
    mov 3               // to w
    mov w4,2
```



1	0	1
0	0	0
1	0	1

Mask Matrix

Example of 2:1 Scaling

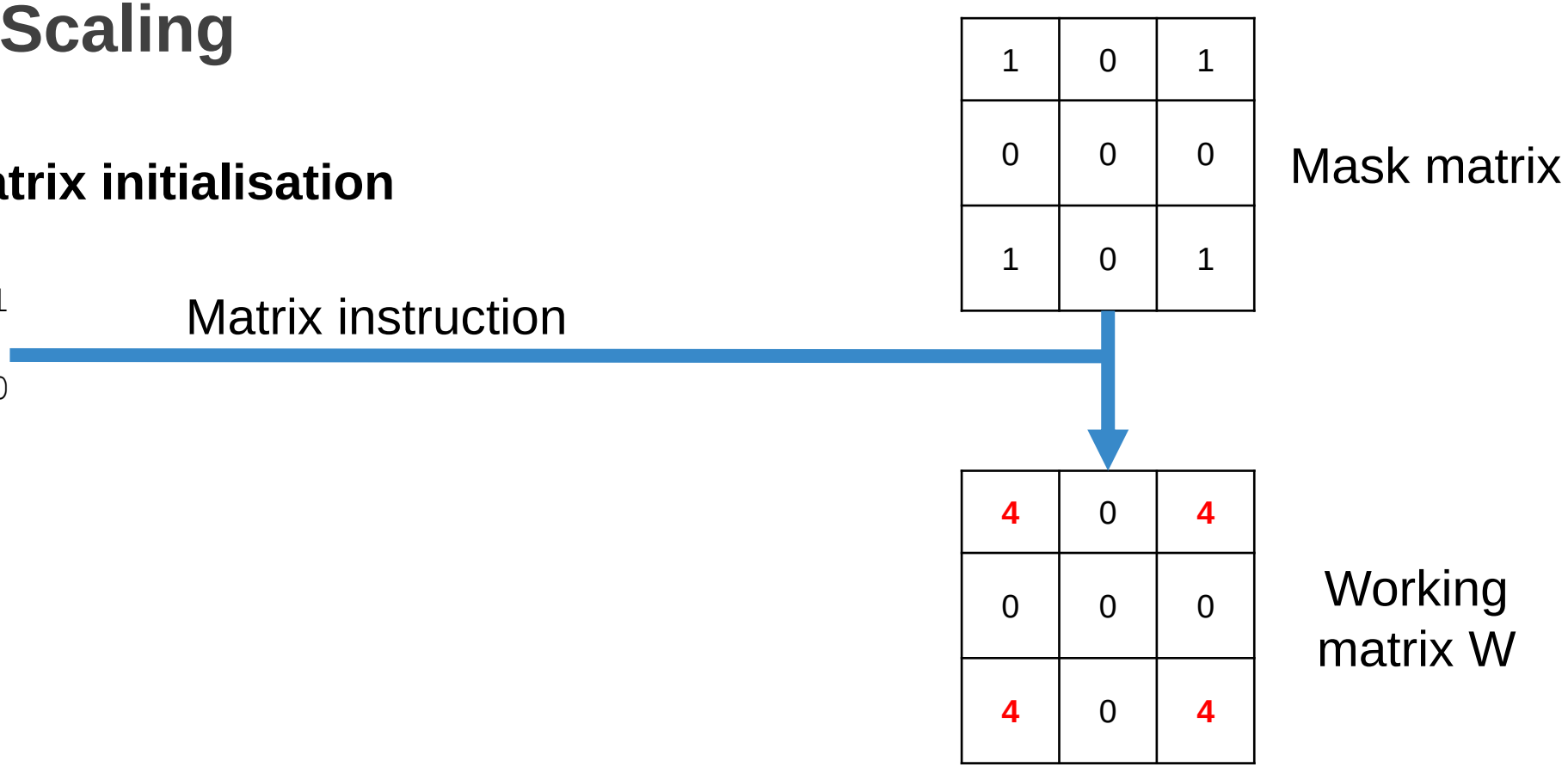


Mask matrix

Example of 2:1 Scaling

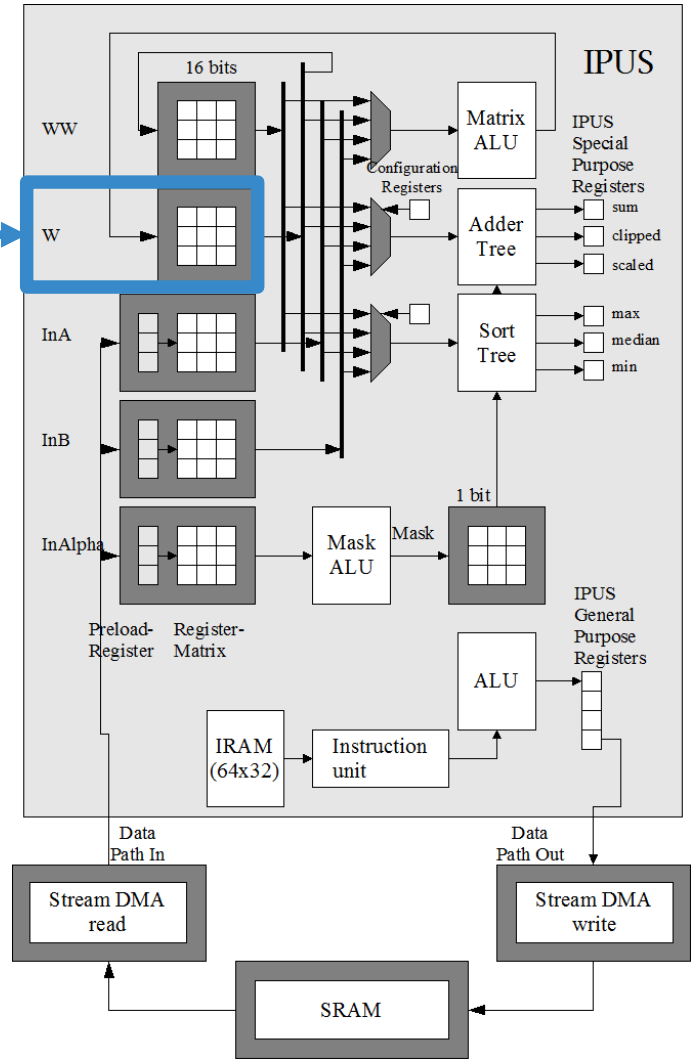
Code: Shifting matrix initialisation

```
d2:
    mset 0b101000101
    mov 4
    mset 0b010101010
    mov 3
    mov w4,2
```



Example of 2:1 Scaling

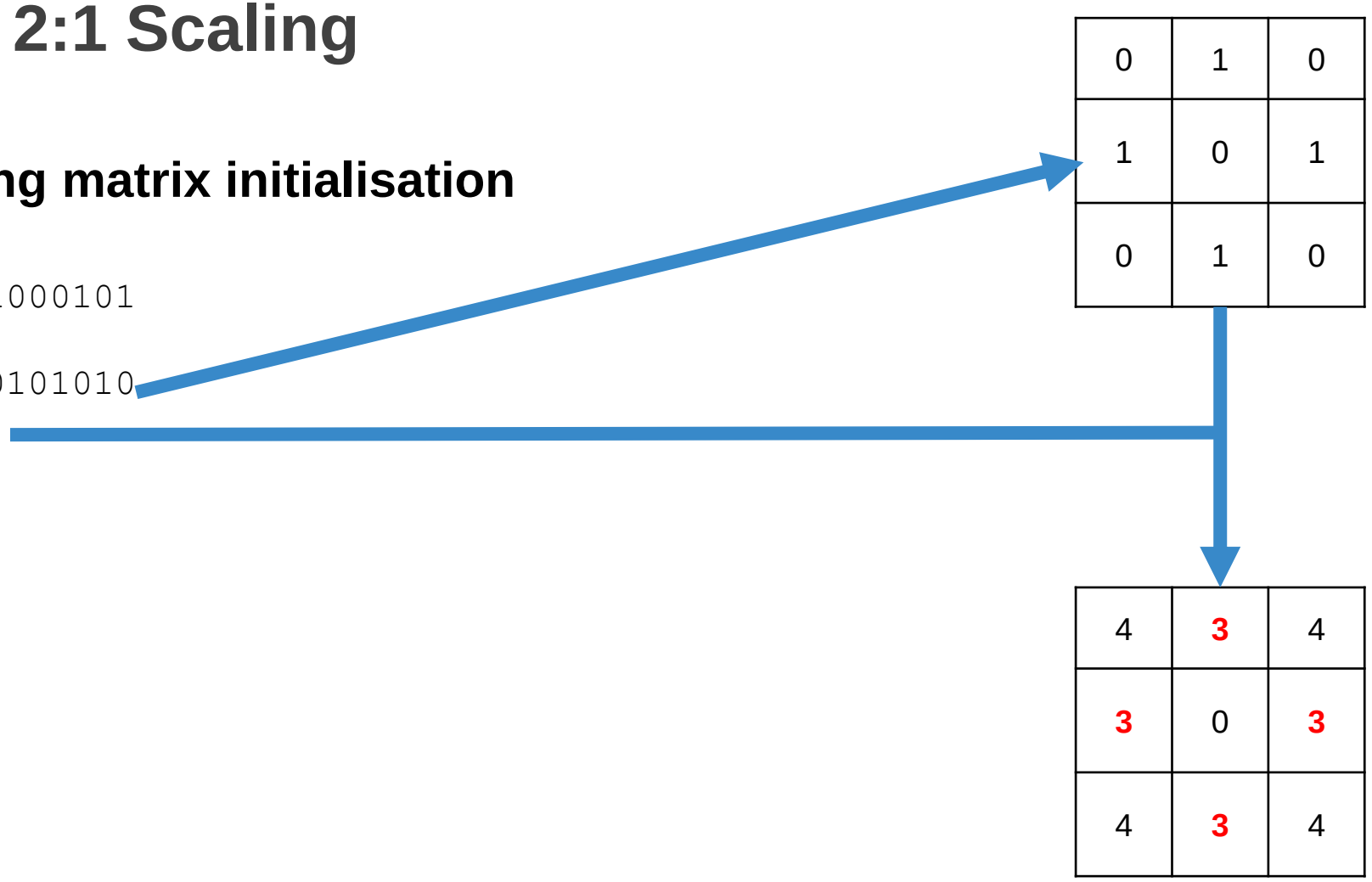
Working matrix W



Example of 2:1 Scaling

Code: Shifting matrix initialisation

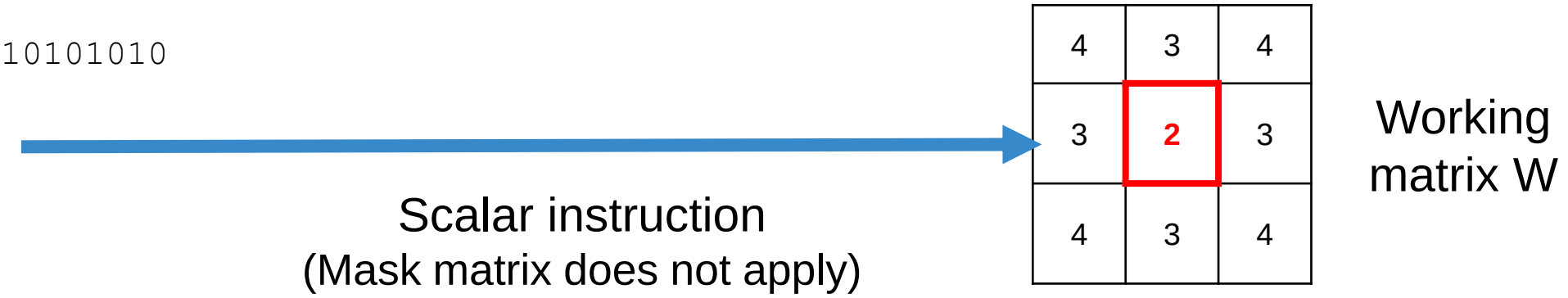
```
d2:
    mset 0b101000101
    mov 4
    mset 0b010101010
    mov 3
    mov w4,2
```



Example of 2:1 Scaling

Code: Shifting matrix initialisation

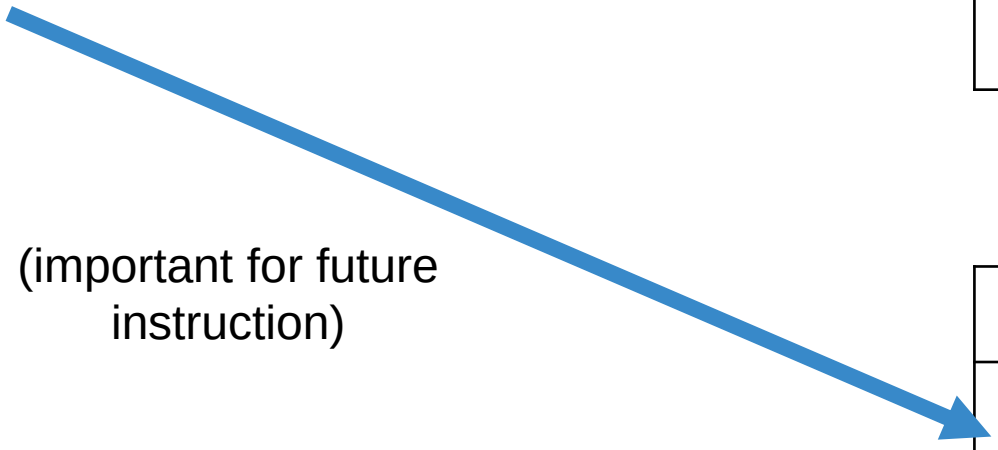
```
d2:
    mset 0b101000101
    mov 4
    mset 0b010101010
    mov 3
    mov w4,2
```



Example of 2:1 Scaling

Code: Shifting matrix initialisation

```
swp                 // w -> ww
mset 0b111111111
```



(important for future instruction)

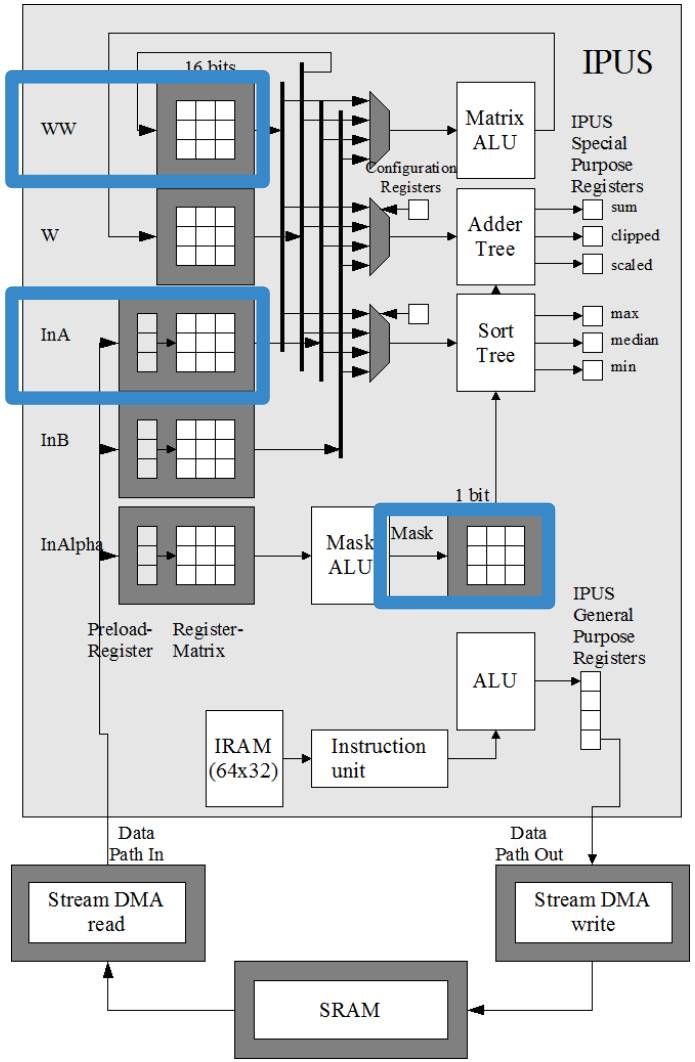
4	3	4
3	2	3
4	3	4

Shift matrix saved in Working matrix **WW**

1	1	1
1	1	1
1	1	1

Mask matrix

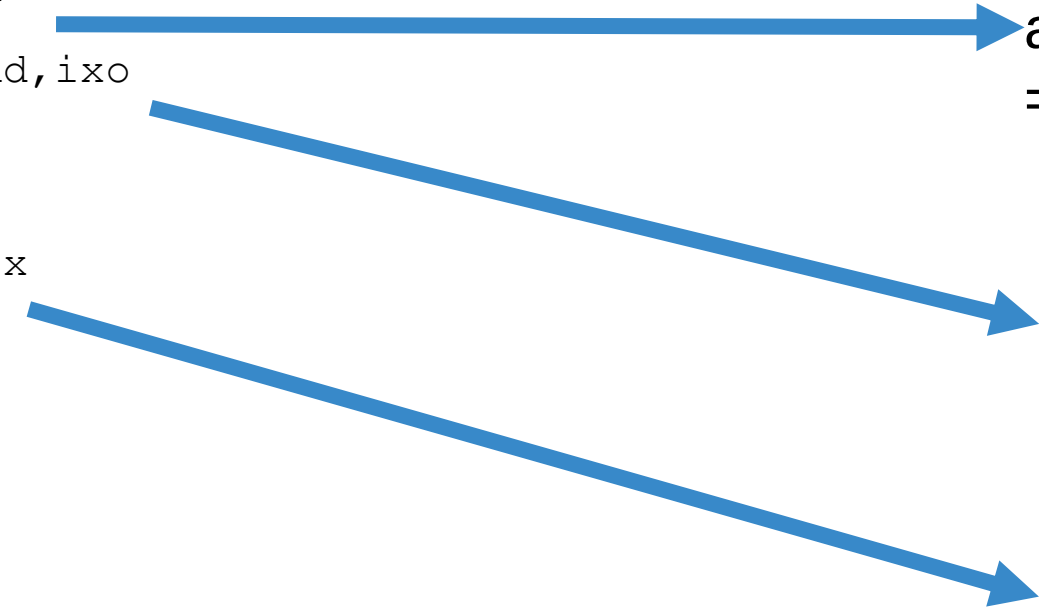
Example of 2:1 Scaling



Example of 2:1 Scaling

Code: Main Loop

```
loop:
  lsr  ina,ww
  dout sum,odd,ixo

odd:
  done loop,ix
end_scale::
```



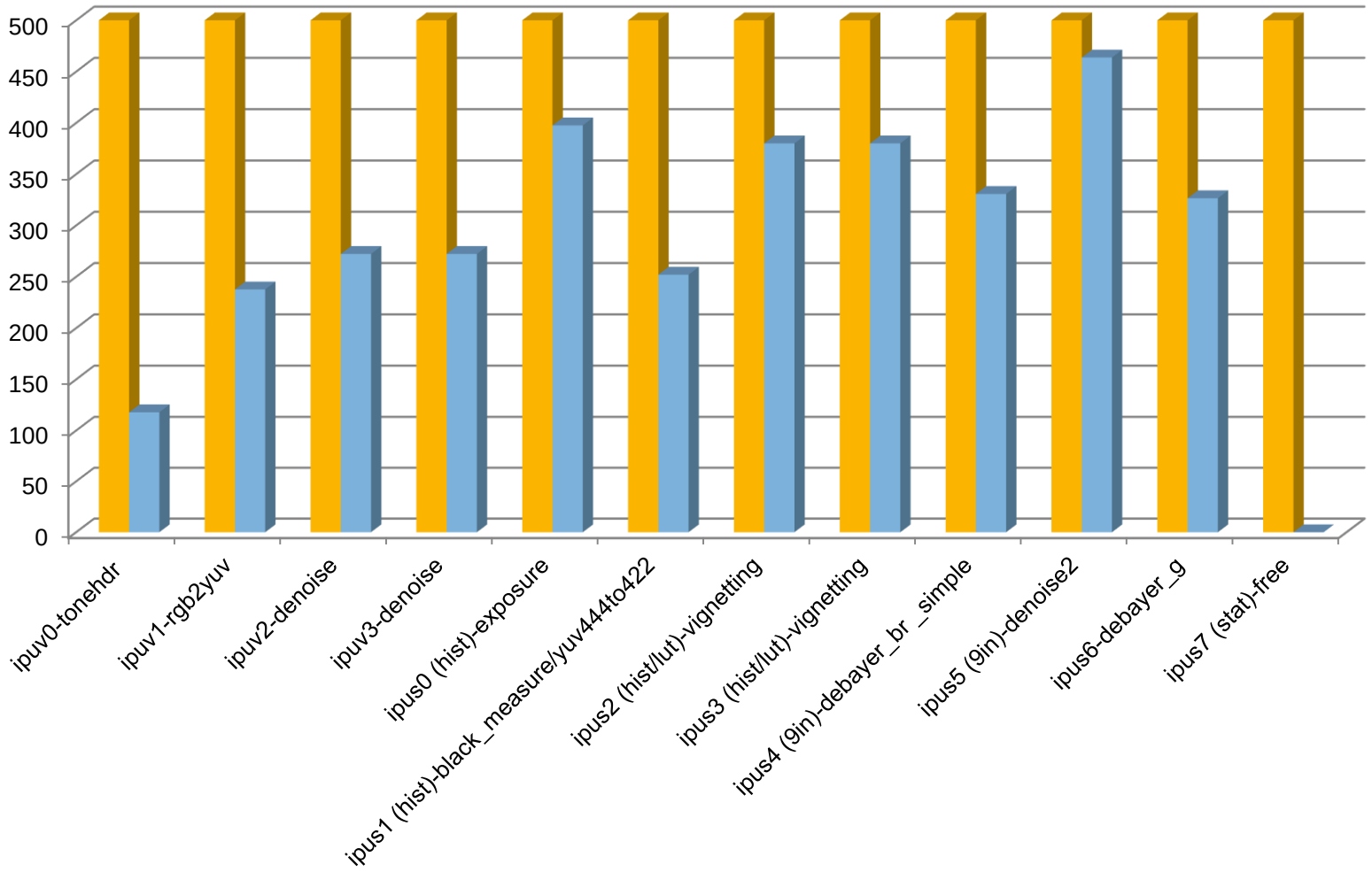
Load-right shift InA elements according to the coefficient in WW => result saved in W

Add all the W elements together and put it in output + Fetch new entry

Fetch new entry without creating any output: scaling of 2:1 Continue the loop

The kernel will restart from the beginning for each new line

Kernel performances



2 MPixels @ 30 fps, dual exposure

ISP Performances and Limitations

- ???

SW

ISP SW

- NXP provides:
  - Sequencer code
  - Graph tool + compiler
  - ISP library
  - Kernel examples
  - Graph examples

- Programmer will create its own:
  - Graph
  - Kernels

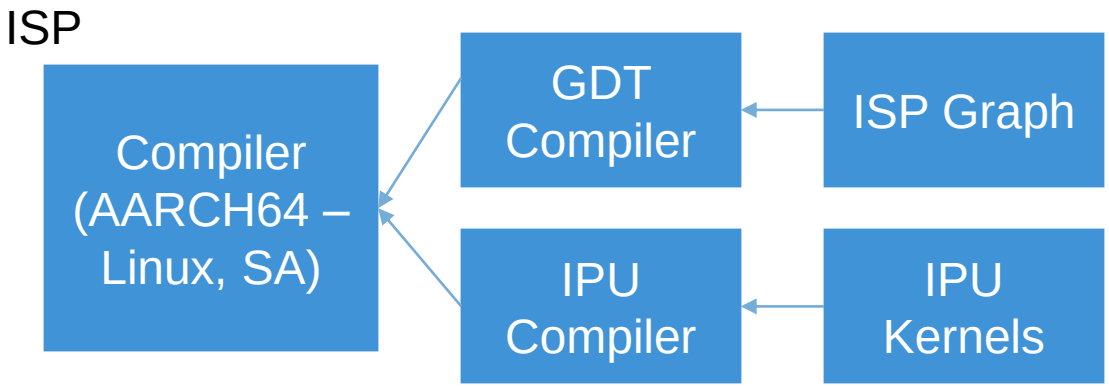
ISP Application

ISP

ISP Graph

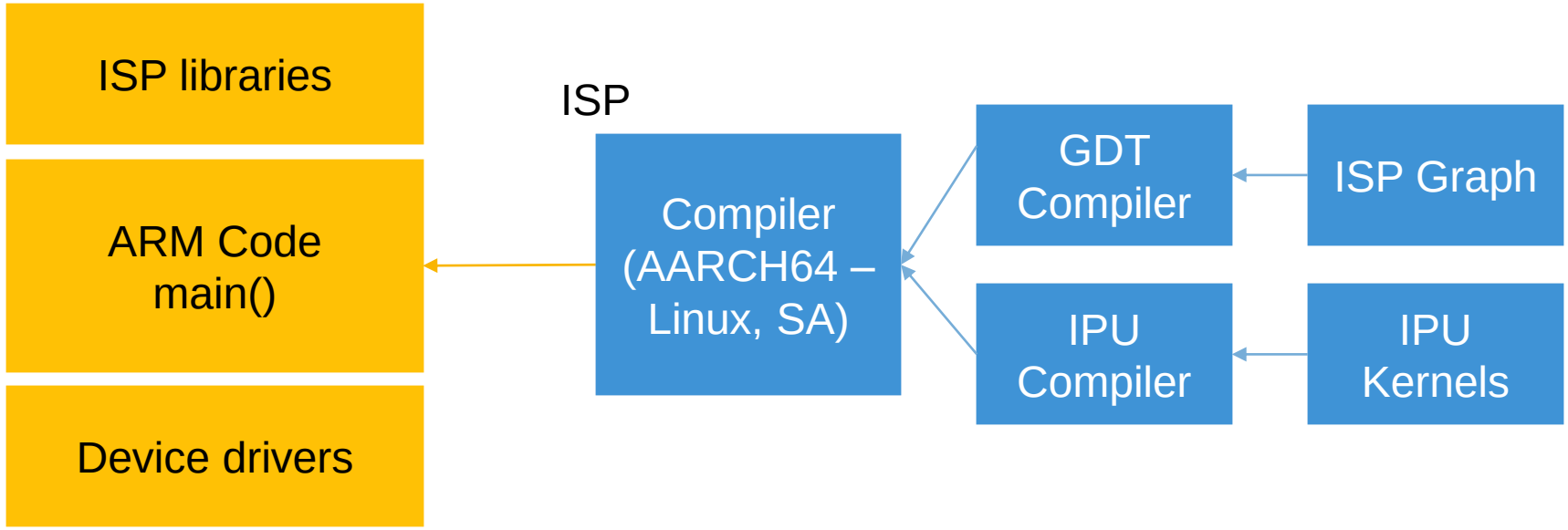
IPU Kernels

ISP Application

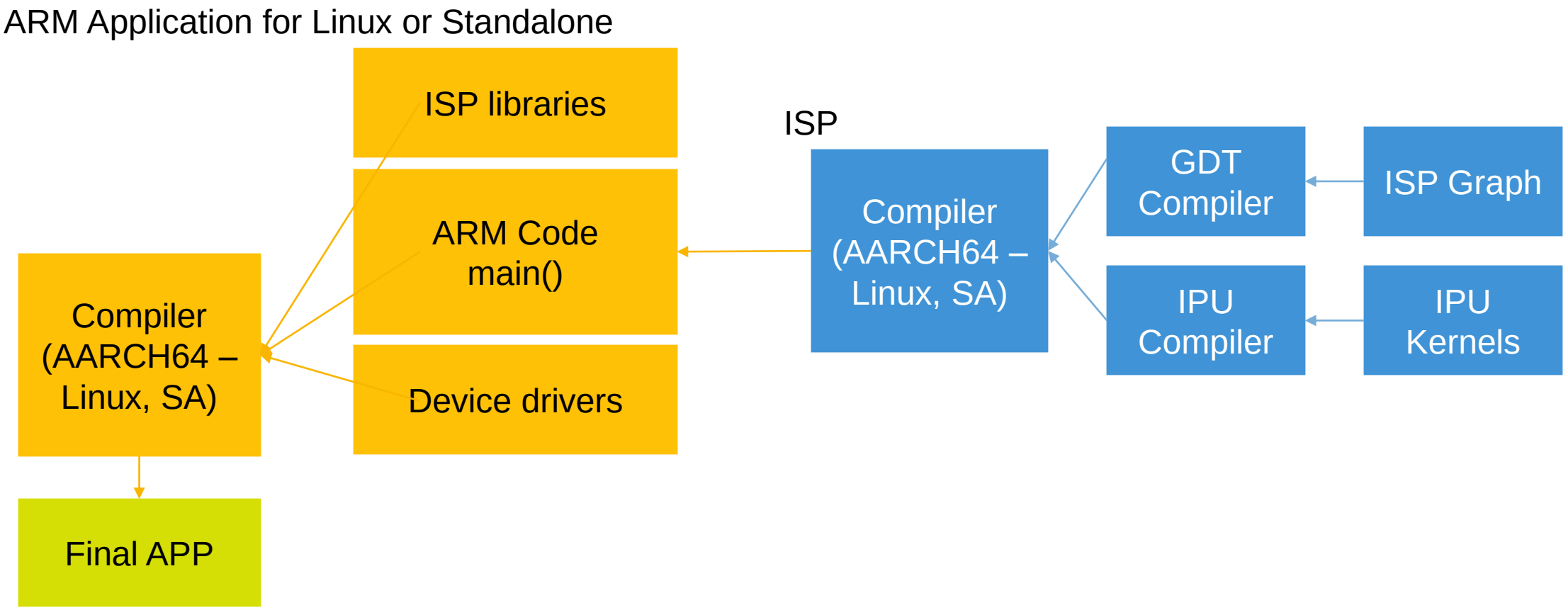


ISP Application

ARM Application for Linux or Standalone



ISP Application



CONCLUSION

QUESTIONS

DEMOS

SECURE CONNECTIONS
FOR A SMARTER WORLD

ATTRIBUTION STATEMENT