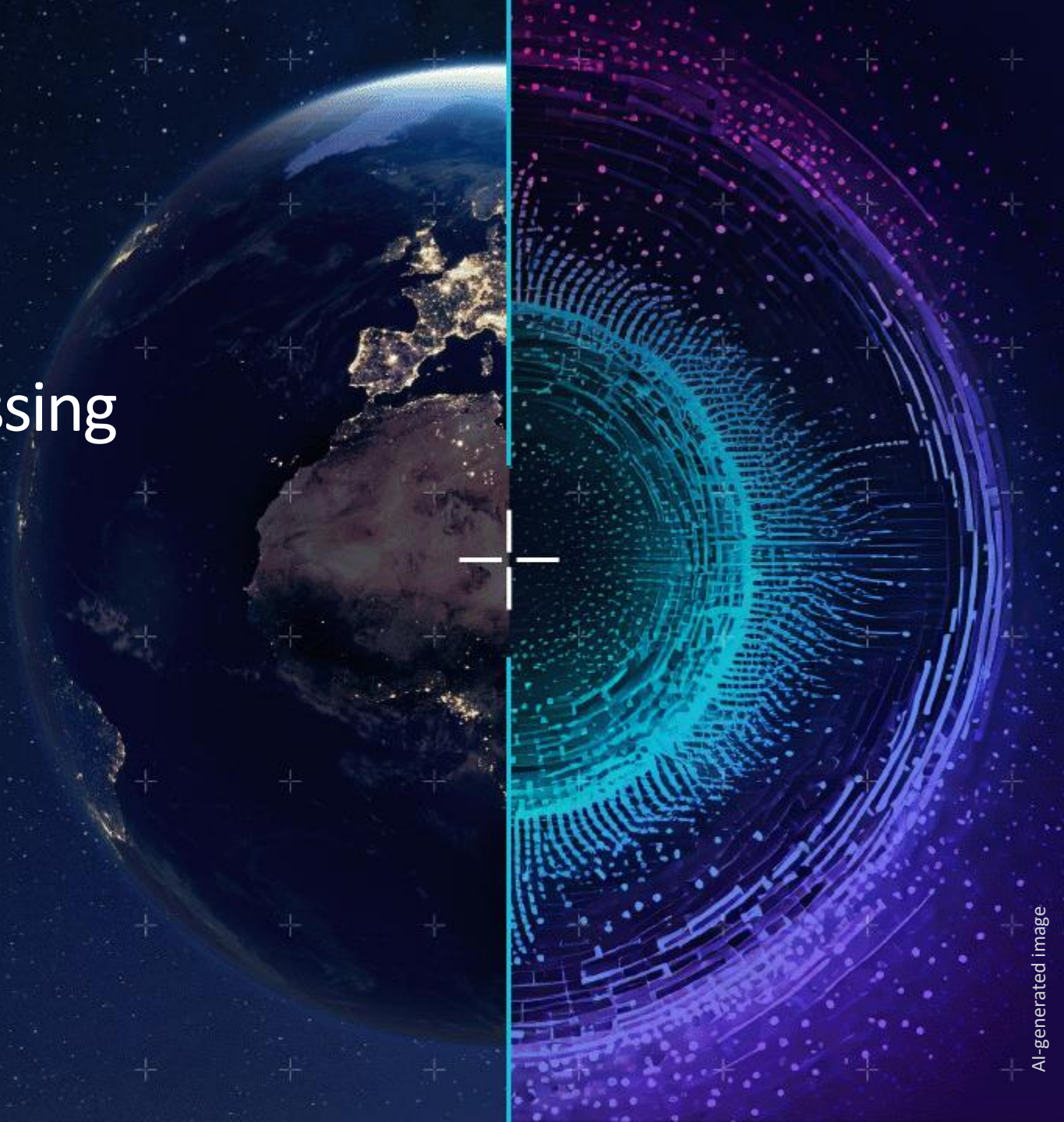
arm

A graph interpreter for DSP/ML stream-based processing



AI-generated image

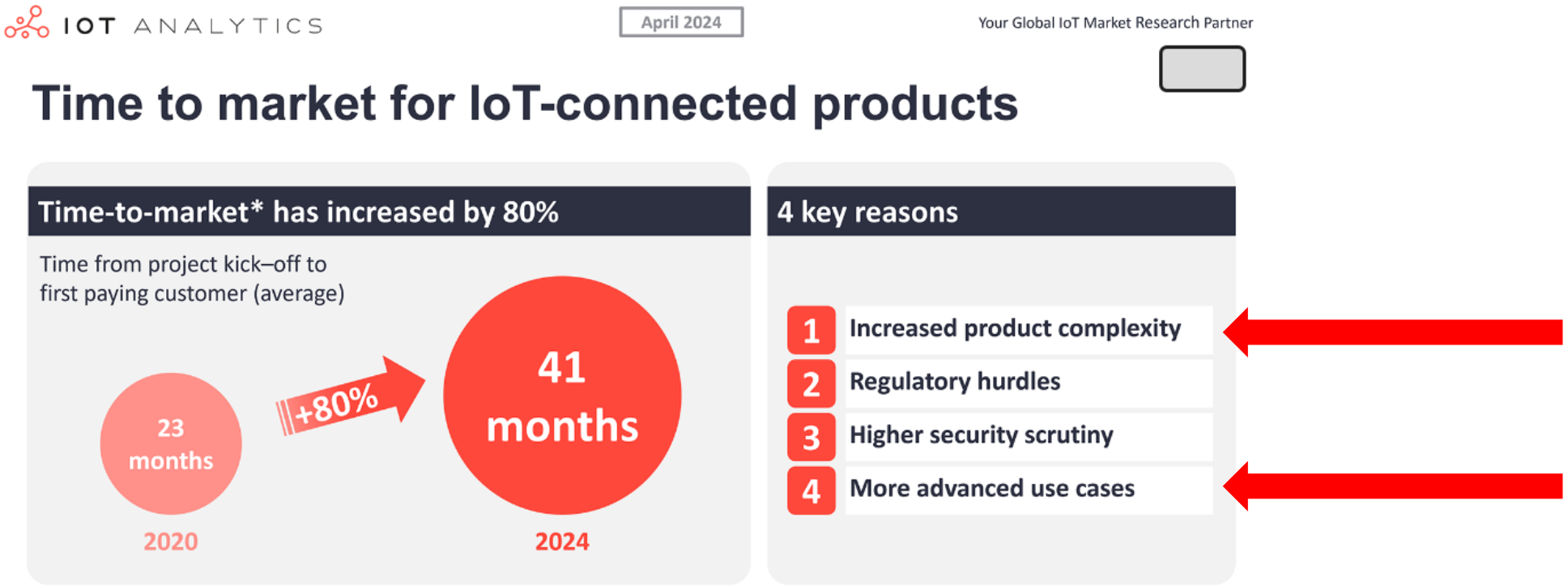
Summary : DSP/ML is complex and it slows time-to-market

Complexity from the “physical domains” and the “software computation domains”

⇒ Split the problem in smaller pieces (“computing nodes”)

⇒ The Nodes are in Flash, activated by an interpreted scenario of stream processing

⇒ A “low-code” scheme is used to add other Nodes



Note: *Time to market = time needed (in months) to get from project kick-off to first paying customer.
Source: IoT Analytics Research 2024 – IoT Commercialization & Business Model Adoption Report 2024. We welcome republishing of images but ask for source citation with a link to the original post and company website.

arm

Splitting the problem, looking at the different focus

Silicon vendors

First interest : brute force demonstration of the architecture on micro-kernel

Other interests : the software ecosystem can switch to a new HW architecture

Software developers

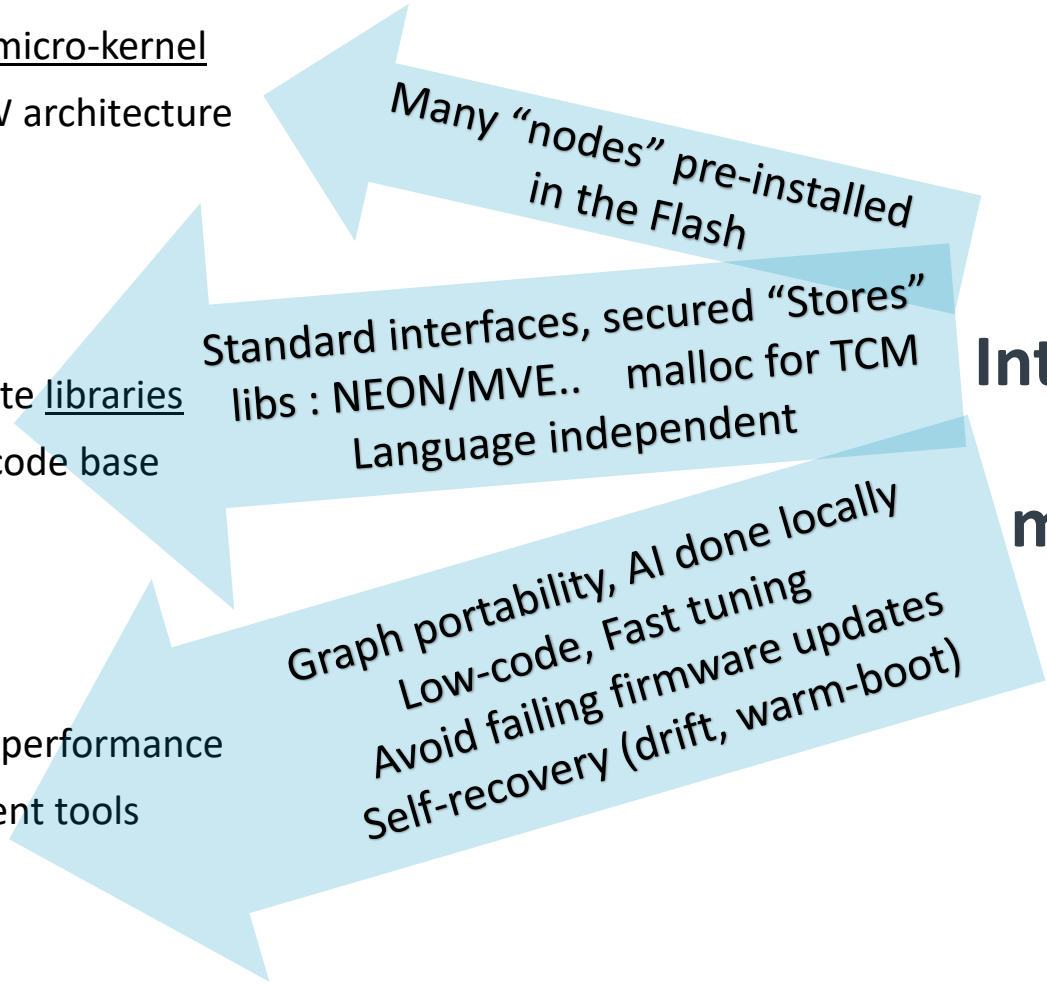
First interest : ease of developments with tools, tutorials, compute libraries

Other interests : portability and performance scaling of existing code base

System integrators

First interest : wide catalog of applications, multi-source, decent performance

Other interests : accelerate time to market with good development tools



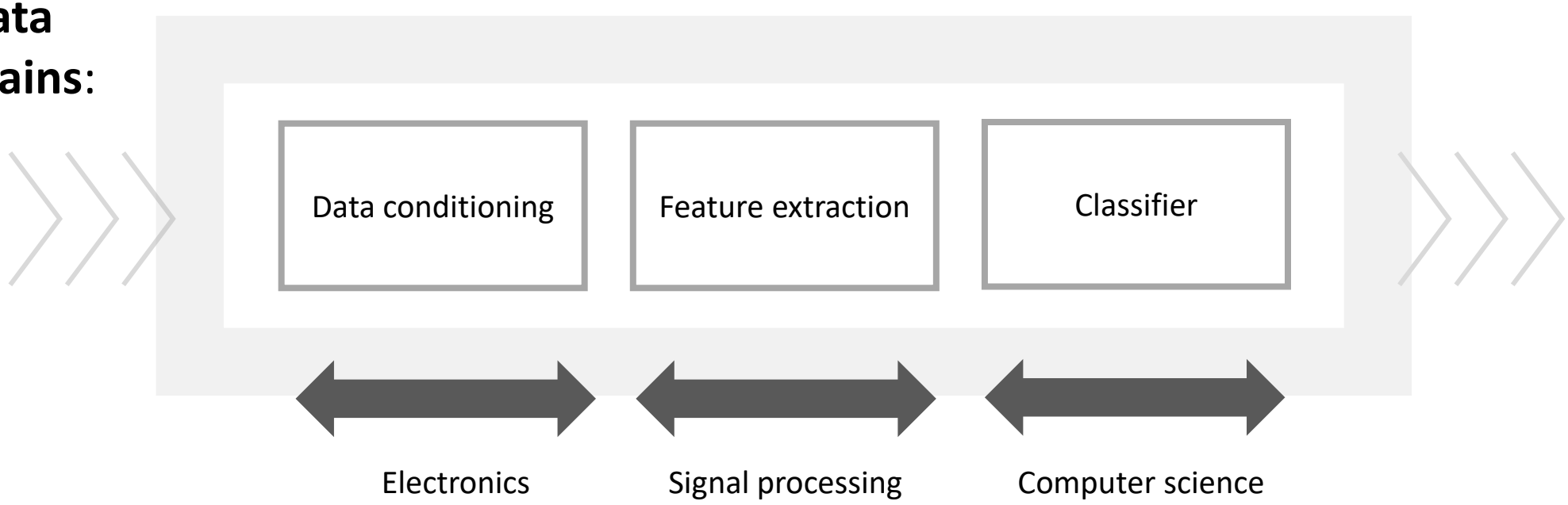
Graph Interpreter selling message

Stream-based processing - different domains of expertise

System and software complexity of a graph of DSP/ML processing is originated from the various domains of expertise

Different **data physical domains**:

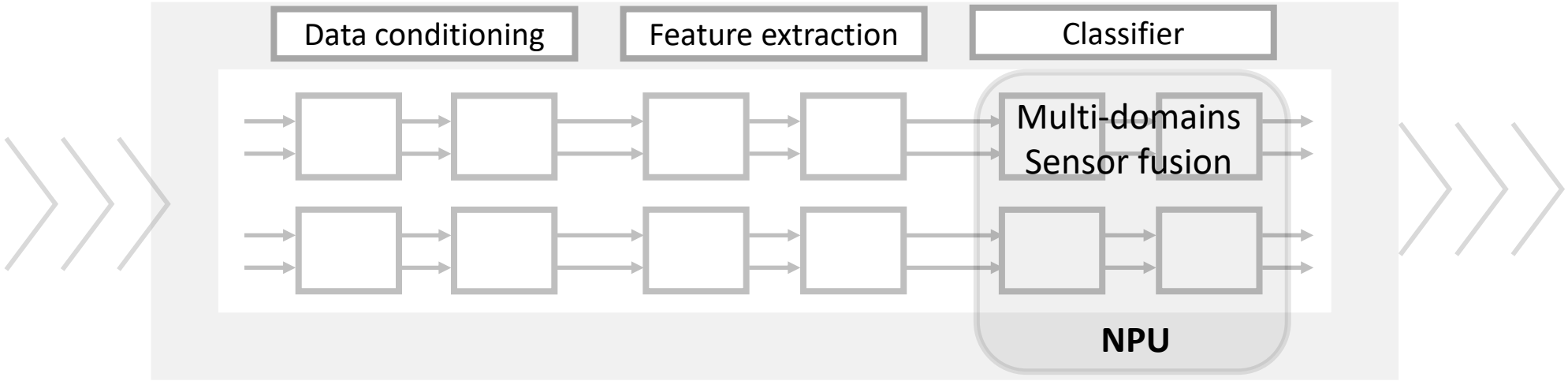
- Acoustics
- Electrical
- Chemical
- Mechanical
- . . .



Different **software engineering domains**

Stream-based processing with graph of computing nodes

- Acoustics
- Electrical
- Chemical
- Mechanical
- . . .

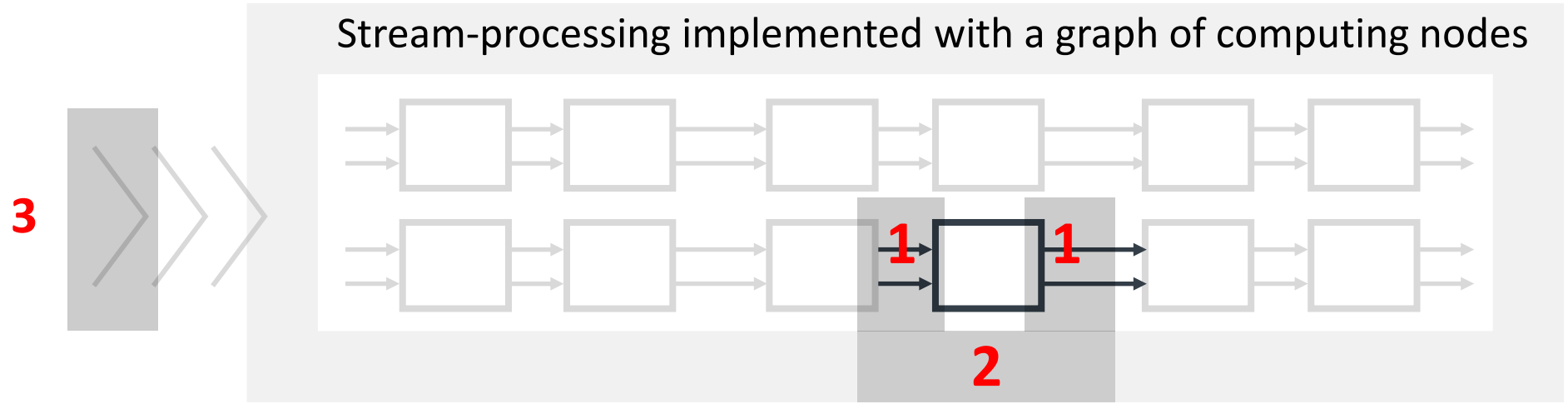


Our proposal : stream-processing is implemented with a graph of computing nodes **designed independently** (from different providers), some nodes can be pre-installed in the Flash of the platform manufacturer

The proof of concept is in production with the graph of EEMBC audiomark using four DSP nodes (beamformer, echo and noise suppressor and a classifier node for Key Word Spotting) running with or without NPU, but with the same node's interface

Manifests of interfaces for Nodes, Graph-I/O, Processor

Standardized, and formalized interfaces, between nodes, with the scheduler and with the graph interfaces



1 **Inter-node interface** : data format (sampling rate, interleaving, raw format, frame size)

2 **Processor interface** with nodes : memory allocation and TCM, compute libraries and NPU

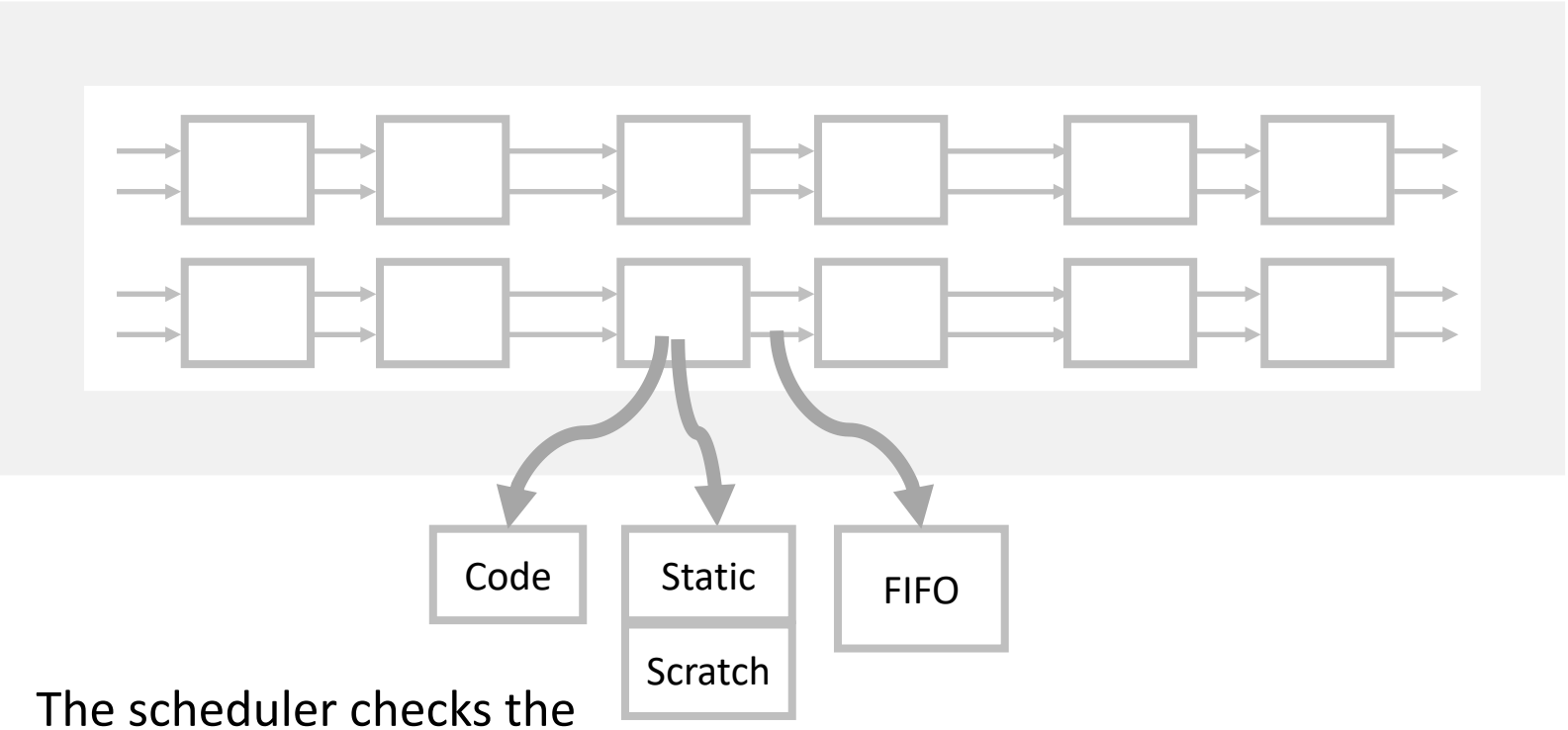
3 **Graph-I/O interfaces** : buffering and polling scheme, mixed-signal configuration of the domains

arm

Graph interpreter and scheduler

The graph intermediate format is a text file, “compiled” to build a scheduling table and a memory mapping

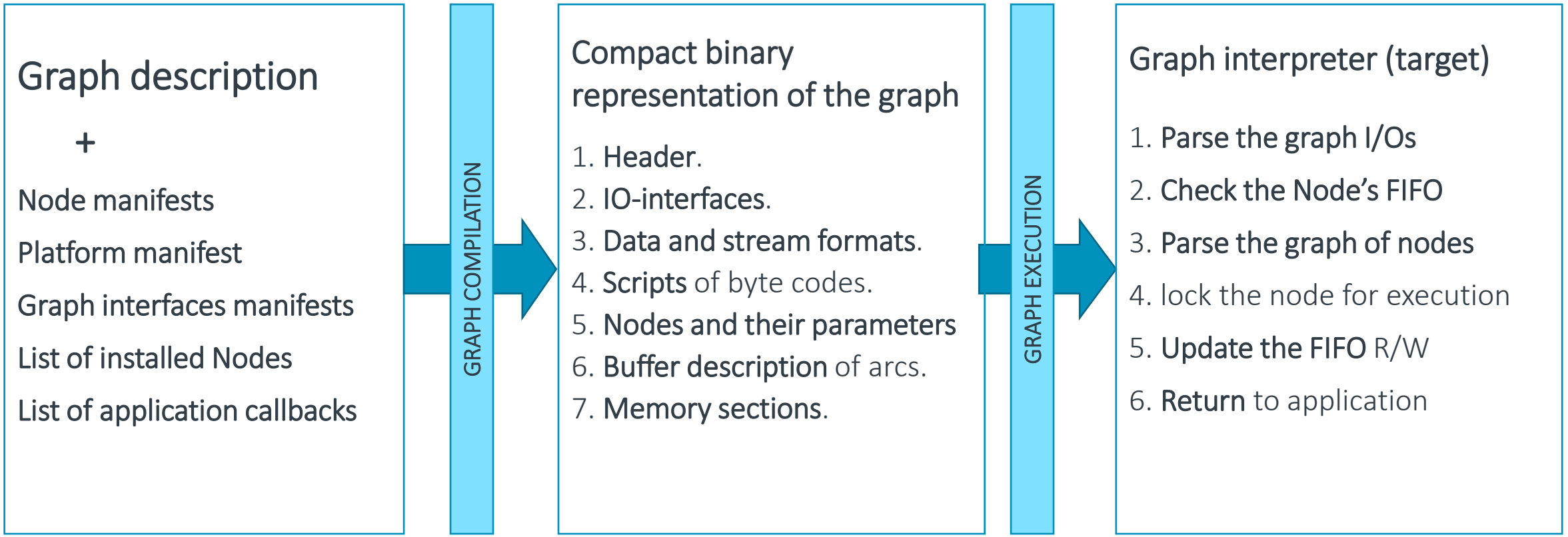
The compiled graph is a linked-list with references to memory buffers and node addresses



The scheduler checks the FIFO buffers before calling a node instance

Compilation process using “Manifests”

“manifests” of the I/O and nodes are helping the “graph compiler” to build the memory map and the data flow between Nodes

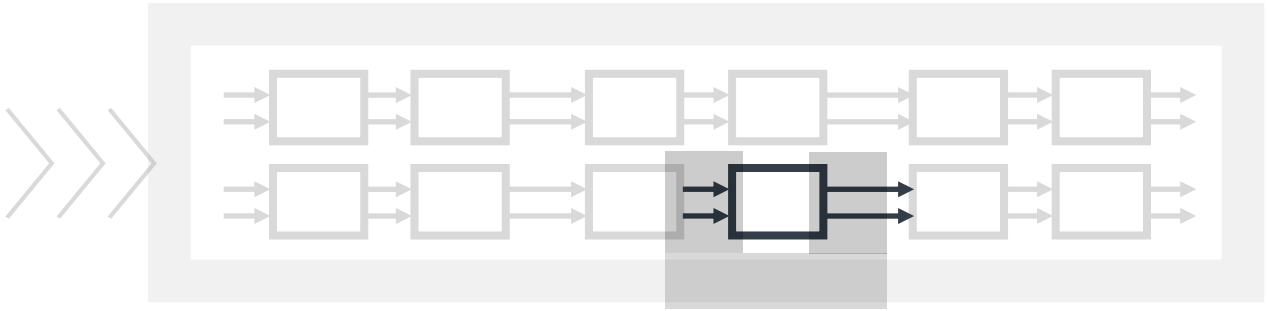


arm

arm

Graph design

Manifests of Nodes



1 **Inter-node interface** and interface with the platform :

a text file (readable syntax)

done once at node delivery

```
; ------------------------------------------------------------------------------
; SOFTWARE COMPONENT MANIFEST - "arm_stream_filter"
; ------------------------------------------------------------------------------
;
node_developer_name   ARM                    ; developer name
node_name             arm_stream_filter      ; node name

node_using_arc_format  1           ; to let filter manage q15 and fp32
node_mask_library     64           ; dependency with DSP services

;-------------------------------------------------------------------------------
;   MEMORY ALLOCATIONS

node_mem               0           ; first memory bank (node instance)
node_mem_alloc        76           ; amount of bytes

node_mem               1           ; second memory bank (node fast working area)
node_mem_alloc        52           ;
node_mem_type          1           ; working memory
node_mem_speed         2           ; critical fast
;-------------------------------------------------------------------------------
;    ARCS CONFIGURATION
node_arc            0
node_arc_nb_channels      {1 1 2}   ; arc intleaved,  options for the number of channels
node_arc_raw_format     {1 17 27}   ; options for the raw format STREAM_S16, STREAM_FP32

node_arc            1
node_arc_nb_channels    {1 1 2}     ; options for the number of channels
node_arc_raw_format     {1 17 27}   ; options for the raw format STREAM_S16, STREAM_FP32

end
```

Graph (a text file : manual input or generated by a GUI)

Nodes

```
arm_stream_filter   0

parameters
   1  u8;  0
   2  u8;  2 0
   5 h16; 1231 1D28 1231 63E8 D475
   5 h16; 1231 0B34 1231 2470 9821
_end_
```

Node name instance index Boot preset,

Options : Memory allocation, pre/post processing script,

Dedicated architecture, processor (or any), priority, trace verbose level

Memory mapping of each segment

Arcs

```
arm_stream_filter        0   1
arm_stream_detector      0   0
```

Node name instance index arc output index

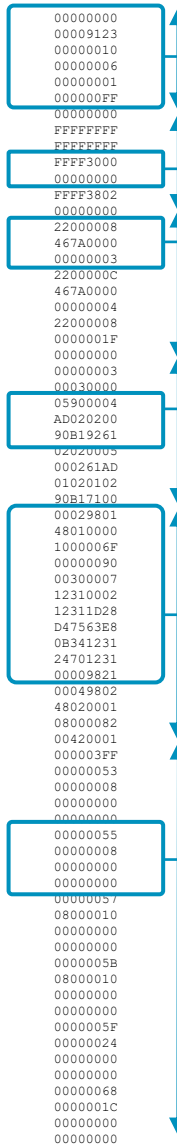
Node name instance index arc input index

```
_graph_interface         4   1
arm_stream_filter        0   0
```

BOUNDARY ARCS

Index of the IO

“Compiled” Graph (used by the scheduler)



Header

Memory and address destination of the graph in the device,

Indexes to the sub-sections below, amount of memory released from maximum allowed

IO interfaces

Index to table of IO functions

```
(io_function_ctrl) (command, uint8_t *data, uint32_t length);
```

“Reset”, “Set Parameter”, “Move data”, “Stop”

Arc ID interacting with this IO, Data format, Settings at reset

Data formats

Frame size, interleaving and time-stamp scheme,

raw data type, Nb of channels, Sampling rate,

Domain-dependent settings

Scripts

Byte codes : arithmetics, access to FIFO,

update the nodes parameters,

software timers and basic state-machines

Nodes

5 words per nodes + parameters : processor and arch selection, number of arcs, scripts to call before and after, index of the node

List of arcs IDs, their block of memory assigned for debug/trace.

List of memory blocks (address, size, address of swap/copy to slower memory) asked in the node manifest

Parameters : preset (among 16), selection of a specific parameter subset to update after preset, Byte-stream of parameters

Arcs

ARC descriptors :

Producer and consumer Data format

Base address of the buffer using a base offset ID and a signed byte index. Buffer size, Read and Write index.

Debug command and result register index.

Collision of node access protocol (Dekker’s algorithm), flag for the need of buffer data realignment.

Root file of the platform details (all the manifests used by the translation tool)

```
; ------------------------------------------------------------------------------
; list of paths for the included files

    3                                       three file paths
    ../../stream_platform/                   "" path index 0 is local
    ../../stream_platform/windows/manifest/  "" path index 1
    ../../stream_nodes/                      "" path index 2

; ------------------------------------------------------------------------------
; PLATFORM DIGITAL, MIXED-SIGNAL AND IO MANIFESTS - max 32 IOs => iomask

    1   platform_manifest_computer.txt      path index + file name

    ;   path + manifests file + index used in the graph + processor affinity bit-field +

    10  number of IO streams available      aligned with struct platform_io_control plat
    ;Path      Manifest          fw_io_idx ProcCtrl clock-domain   definition          (c
    1   io_platform_data_in_0.txt       0     1        0    application processor     #c
    1   io_platform_data_in_1.txt       1     1        0    application processor     #c
    1   io_platform_analog_sensor_0.txt 2     1        0    ADC                       #c
    1   io_platform_motion_in_0.txt     3     1        0    accelero=gyro             #c
    1   io_platform_audio_in_0.txt      4     1        0    microphone                #c
    1   io_platform_2d_in_0.txt         5     1        0    camera                    #c
    1   io_platform_line_out_0.txt      6     1        0    audio out stereo          #c
    1   io_platform_gpio_out_0.txt      7     1        0    GPIO/LED                  #c
    1   io_platform_gpio_out_1.txt      8     1        0    GPIO/PWM                  #c
    1   io_platform_data_out_0.txt      9     1        0    application processor     #c

; ------------------------------------------------------------------------------
; SOFTWARE COMPONENTS MANIFESTS

    19  nodes     path index + file name, in the same order of p_stream_node node_entry_

    ;  p_stream_node node_entry_point_table[NB_NODE_ENTRY_POINTS] =
    ;                                          /*  0 node disabled */
    2         Basic/arm/script/swc_manifest_script.txt          /*  1 arm_script
    2         Basic/arm/router/swc_manifest_router.txt          /*  2 arm_stream_router
    2      Basic/arm/converter/swc_manifest_converter.txt       /*  3 arm_stream_convert
    2      Basic/arm/amplifier/swc_manifest_amplifier.txt       /*  4 arm_stream_amplifi
    2          Basic/arm/mixer/swc_manifest_mixer.txt           /*  5 arm_stream_mixer
    2         Audio/arm/filter/swc_manifest_filter.txt          /*  6 arm_stream_filter
    2       Audio/arm/detector/swc_manifest_detector.txt        /*  7 arm_stream_detectc
    2       Basic/arm/rescaler/swc_manifest_rescaler.txt        /*  8 arm_stream_rescale
    2     Audio/arm/compressor/swc_manifest_compressor.txt      /*  9 arm_stream_compres
    2   Audio/arm/decompressor/swc_manifest_decompressor.txt    /* 10 arm_stream_decompr
    2      Basic/arm/modulator/swc_manifest_modulator.txt       /* 11 arm_stream_modulat
    2    Basic/arm/demodulator/swc_manifest_demodulator.txt     /* 12 arm_stream_demodul
    2   Basic/arm/interpolator/swc_manifest_interpolator.txt    /* 13 arm_stream_interpc
    2           Basic/arm/qos/swc_manifest_qos.txt              /* 14 arm_stream_qos
    2          Basic/arm/split/swc_manifest_split.txt           /* 15 arm_stream_split
    2     image/arm/detector2D/swc_manifest_detector2D.txt      /* 16 arm_stream_detectc
    2       image/arm/filter2D/swc_manifest_filter2D.txt        /* 17 arm_stream_filter2
    2 image/arm/interpolator2D/swc_manifest_interpolator2D.txt  /* 18 arm_stream_interpc
    2        Basic/arm/synchro/swc_manifest_synchro.txt         /* 19 arm_stream_synchrc
```

Processor manifest

List of available IO for stream processing

Abstraction layer = data move, set buffer, set parameter, stop

Stream physical domains: generic data_stream, audio, gpio, motion, 2D, analog_sensor, analog_transducer, rtc

List of available Nodes

arm

Processor manifest : memory mapping

```
    1 1



    1 1 4 1 9 15
```

Number of architectures, number of processors

Processor ID, Main processor identification, FPU characteristics

Number of memory banks below, bit-field of installed compute libraries

```
    0    h20000000
    1    h28000000
    2    h2C000000
    3    h08000000
```

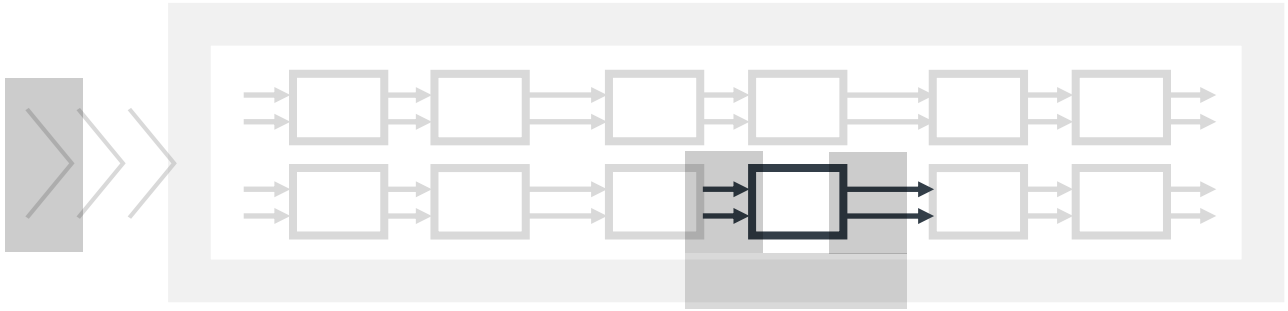
Indexes used in the FIFO descriptors to address the data

```
;    ID VID   S W P H D      Size offset
     0   0    1 0 0 0 0     95526 10
     0   1    0 0 0 0 0     65526 10
     0   2    0 0 0 0 0     30000 65536
     0   3    0 1 0 0 0     15000 95536
     0   4    0 1 0 0 0    256000 262144
     2   1    3 1 1 0 0      1024 262144
     1  10    0 2 0 0 0      1024 524288
     3  20    0 0 0 0 0    200000 10
     2   8    3 1 1 0 1     16384 0
```

Physical memory banks:
- base offset ID reference above
- virtual ID used in the graph for manual mapping
- Speed normal/fast/critical-Fast, static0/working1/retention2, shared0/private1, for DMA, Data0/Prog1
- minimum size guaranteed
- minimum offset from the base offset ID

Manifests of interfaces for Graph-I/Os

3



3 Graph-I/O interfaces :

a text file (readable syntax)

done once at platform manufacturing

```
io_platform_sensor_in_0                    ; name for the tools
analog_in                                  ; domain name,   unit: dB, Vrms, mV/Gauss, dps, kWh, ...

io_commander0_servant1 1                   ; commander=0 servant=1 (default is servant)
io_buffer_allocation   2.0 1               ; default is 0, which means the buffer is declared outside of the graph, VID 1
io_direction_rx0tx1    1                   ; direction of the stream  0:input 1:output from graph point of view
io_raw_format          {1 17}              ; options for the raw arithmetics computation format here  STREAM_S16
io_nb_channels         {1 1 2}             ; multichannel intleaved (0), deinterleaved by frame-size (1) + options for the number of channels
io_frame_length        {1 2 16}            ; [ms]0/[samp]1  +  options of possible frame_size
io_subtype_units       104                 ; depending on the domain. Here Units_Vrms of the "general" domain (0 = any or underfined)
io_analogscale         0.55                ; 0.55V is corresponding to full-scale (0x7FFF or 1.0f) with the default setting
io_sampling_rate       {1 16000 44100 48000} ; sampling rate options (enumeration in Hz)
io_rescale_factor      12.24  -44.3        ; [1/a off] analog_input = invinterpa x ((samples/Full_Scale_Digital) - interpoff)
end
```

Graph API (one entry-point to the scheduler)

1) Graph interpreter interface for the application :

```
void arm_graph_interpreter (uint32_t command,  arm_stream_instance_t *S, uint8_t *data, uint32_t size)
```

Commands : reset the graph, execute, check boundary FIFO filling state and move data in/out, update the use-case

Instance : structure of pointers to the graph, to the installed nodes and application callbacks, to the data stream interfaces functions (below), control fields and static memory of the scheduler instance.

2) Stream interfaces used by the scheduler to initiate data moves (abstraction layer of the BSP):

```
void (io_function_ctrl) (uint32_t command, uint8_t *data, uint32_t length);
```

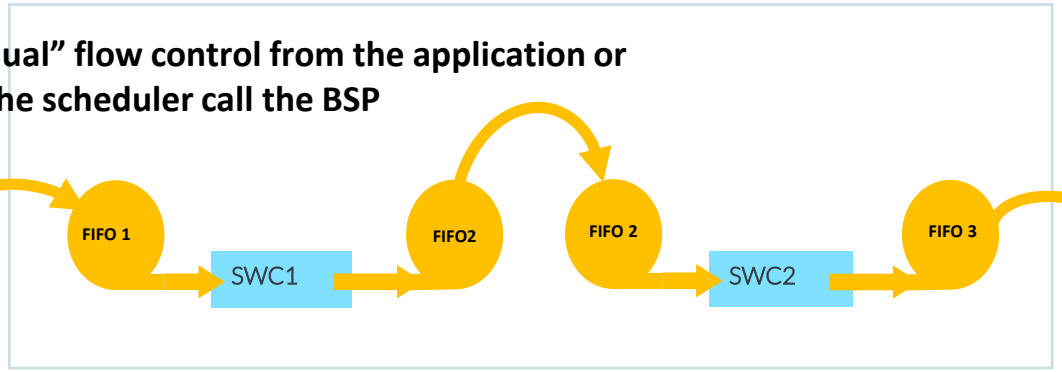
Commands : set buffer, set parameters, data move, stop

3) One callback, after data moves (to update the FIFO descriptors) :

```
void arm_graph_interpreter_io_ack (uint8_t fw_io_idx, uint8_t *data,  uint32_t data_size)
```

4) One prototype for all nodes :

```
void node_XXXX (uint32_t command, void *instance, void *data, uint32_t *status)
```



Abstraction layer of IOs : data-move and settings + callback to set the FIFO
or
Data move from the application with same functions for FIFO setting

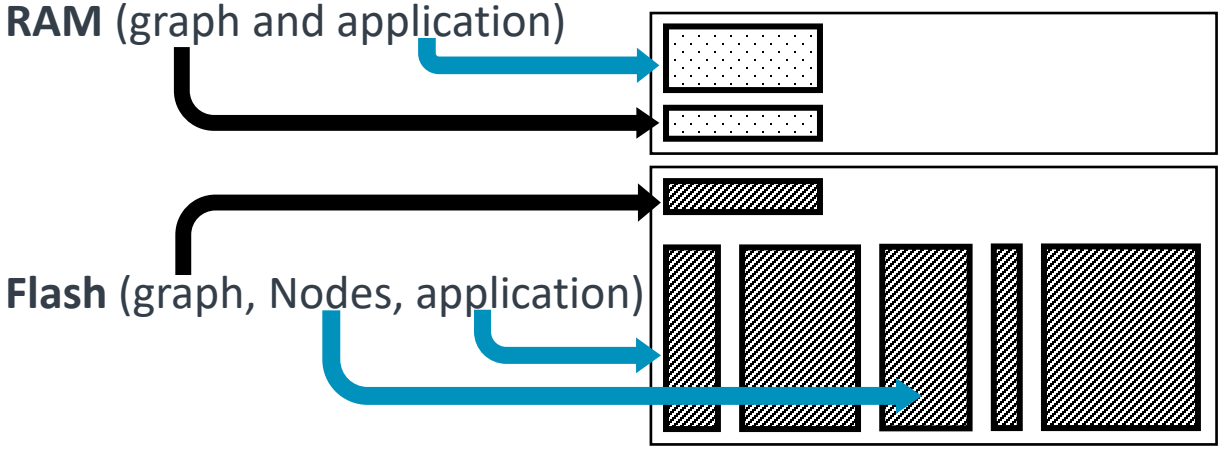
Small memory footprint for LoRA

Remote sensors connected through LoRA have a data rate as low as 50Bytes/s

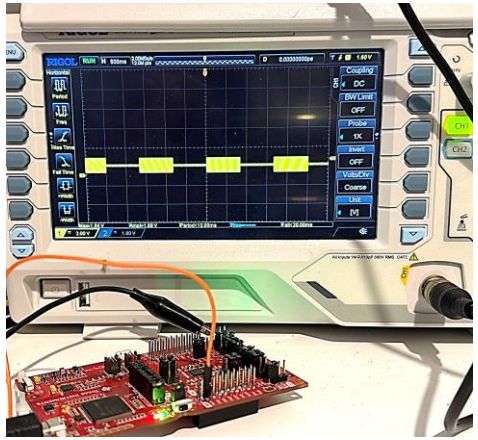
A graph size of two nodes (+ their respective parameters and a script) is in the 500Bytes range

An interpreter eliminates the risk of malware injection during firmware updates

A memory map of the LoRA device :



Filter and detector nodes with 1kB-RAM



Graph with embedded scripts

A graph can incorporate nodes with interpreted code using basic integer/float arithmetics.

The instruction “CALLSYS” gives access to nodes (set/read parameters), arcs (read/write, check access time-stamps), application callbacks, etc..

The script interpreter is consuming less than 100 Bytes of stack memory.

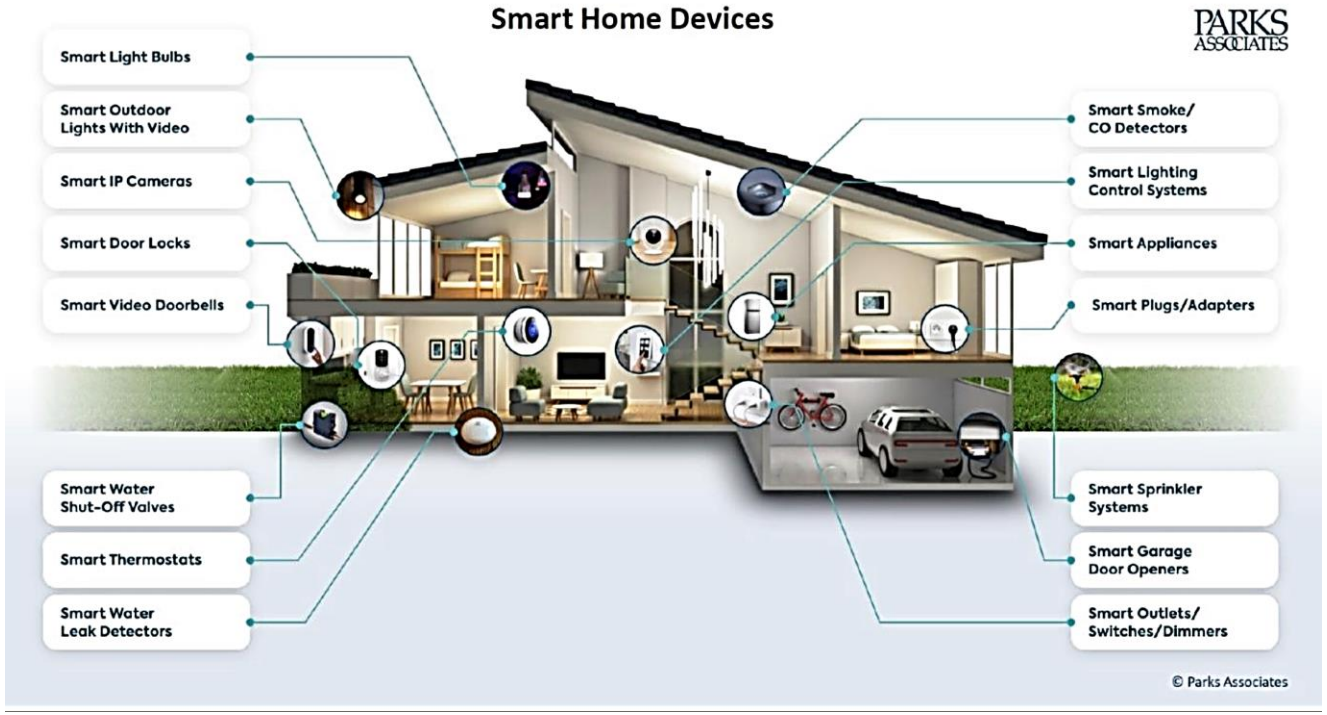
Why would you need Python for very simple operations ?

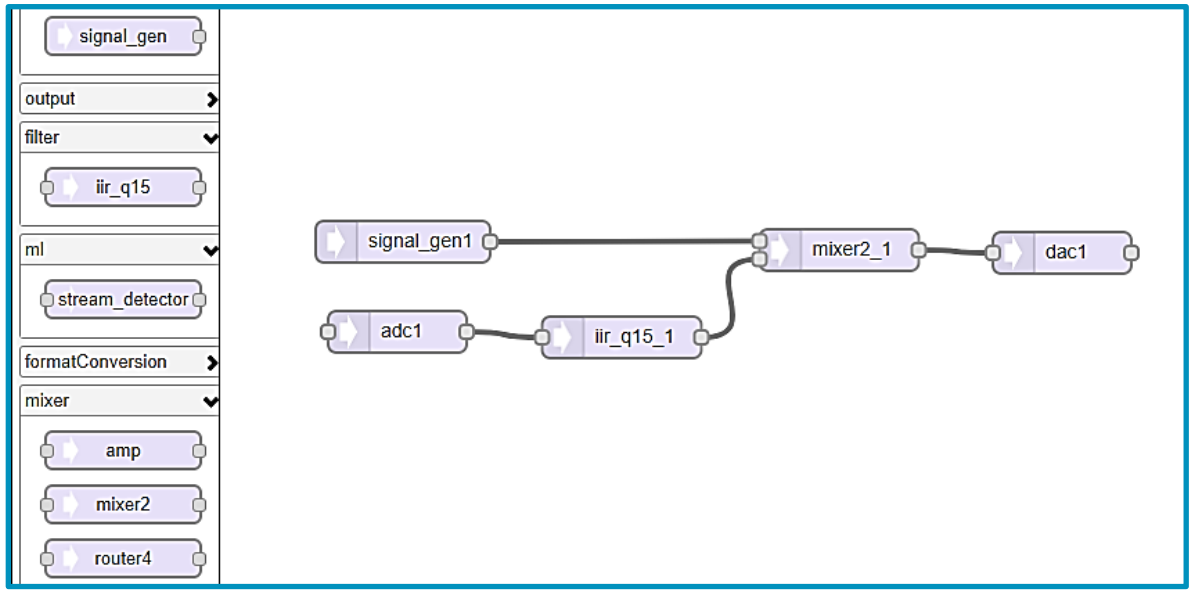
Examples of instructions

```
r6 = add r5 3              r6 = ( r5 + 3 )
sp1 = r6                   push the result on stack
test_eq r6 sub r5 r4       test if r6 == ( r5 - r4 )
if_yes call label_xyz      conditional call
r0 = mul r6 #float 3.14    r0 loaded with r6 multiplied by 3.14
r3 | 8 15 | = r2           bit-field load of r2 to the 2nd byte of r3
r3 = [ r4 ]  r0            gather load
banz L1 r2                 decrement r2 and branch if not zero
call L1 r2 r3              call a subroutine and push 2 registers
Other operations: div,or,nor,and,xor,shr,shl,set,clr,max,min,amax,amin,norm,addmodulo
callsys 4 r1 r5 r10        system call #4 with three parameters
```

Next steps : low-code for smart-home sensors

Do we need a complex programming environment to drag and drop software components from a Store ?





arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు