arm

NanoGraph

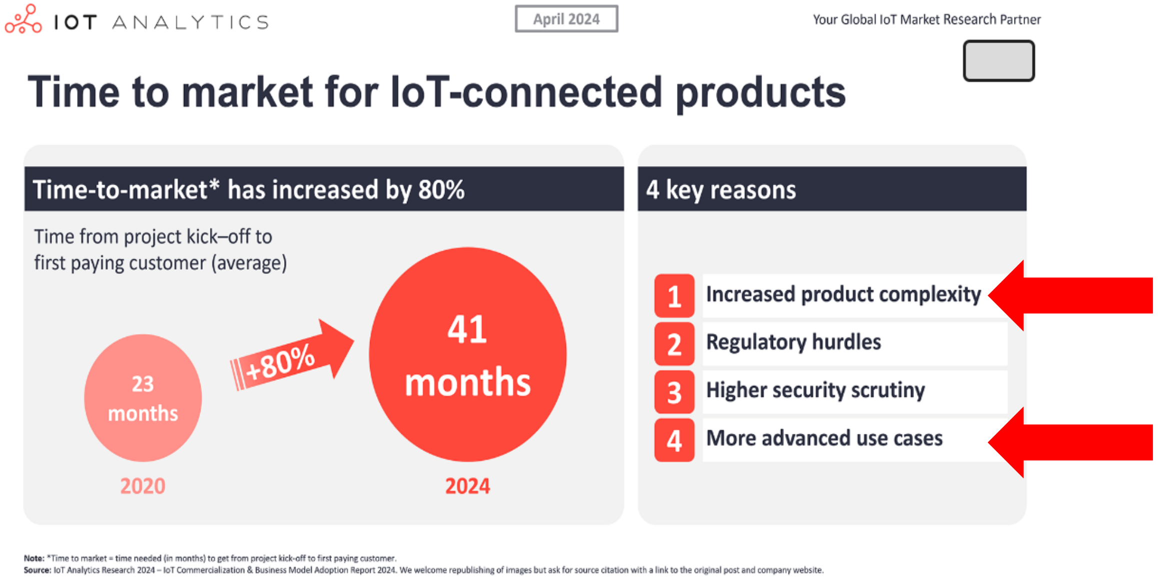
A graph interpreter for DSP/ML stream-based processing

AI-generated image

DSP/ML systems are complex, which slows down time-to-market

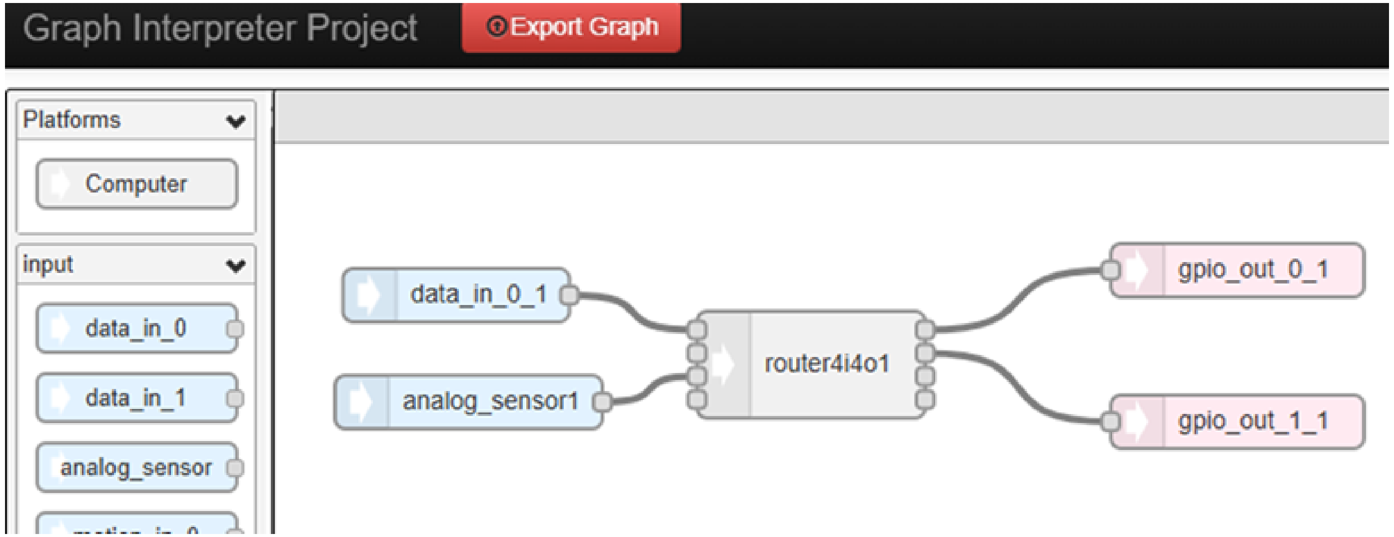
Real-world complexity - coming from multiple physical domains - and the complexity of DSP/ML software both contribute to **long development cycles**. To address this:

- The overall problem is broken into smaller, manageable **computing nodes**.
- These nodes are stored in Flash and activated through an **interpreted** stream-processing scenario.
- A **low-code** development model makes it easy to add new nodes as needed.



What are the ultimate purposes

1. Provide a single, consistent interface for all graph nodes.
2. Allow node designers to build DSP/ML nodes without knowing how they are integrated.
3. Enable system integrators to use nodes without revealing product details.
4. Let silicon vendors add accelerators (FFT, NN, TCM, etc.) without affecting node developers.
5. Support sensor replacement with minimal or no changes through formal interface descriptions.
6. Update the graph without recompiling device firmware.



arm

Splitting the problem, by looking at the users' focus

Silicon vendors

Demonstrate architecture capabilities with micro-kernels

Ensure the software ecosystem can easily migrate to new hardware

Software developers

Want simple development with solid tools and libraries

Need portability and scalable performance for existing code

System integrators

Need access to a wide catalog of applications with acceptable performance

Require strong tools to accelerate time-to-market

Key selling points of NanoGraph

Many pre-installed nodes stored in Flash

Standard interfaces, secured "Stores"
libs : NEON/MVE.. malloc for TCM
Nodes language independent

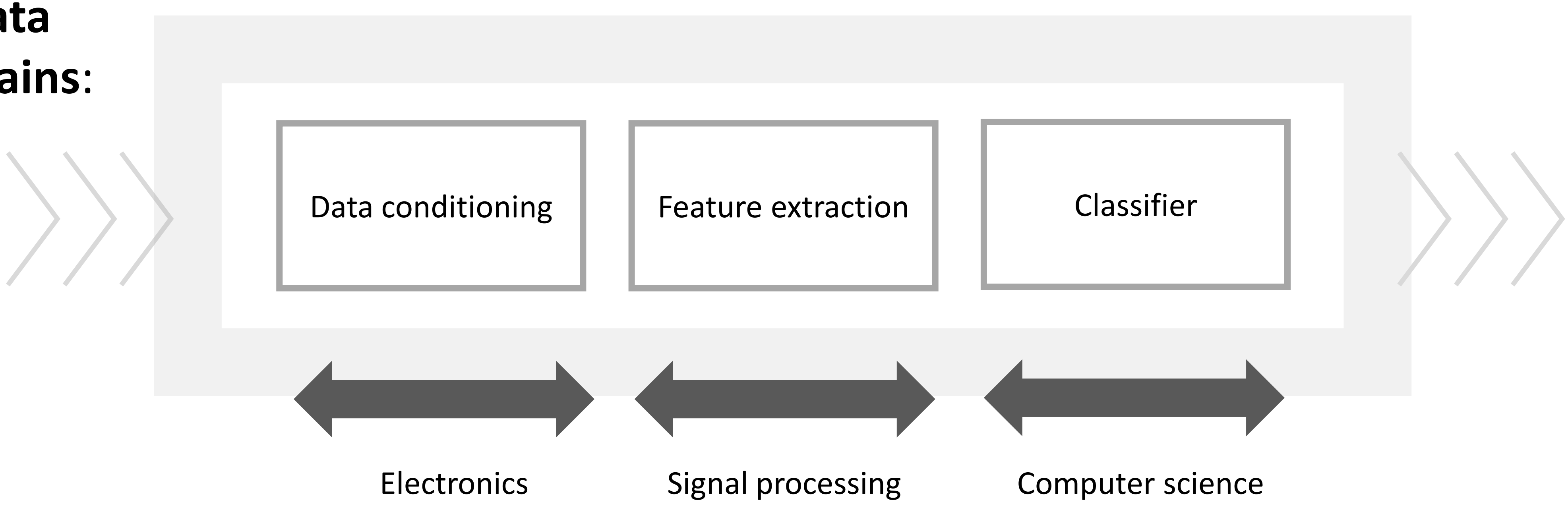
Graph portability, AI runs locally
Low-code, Fast tuning
Lower the risk of failed FW updates
Self-recovery (drift, warm-boot)

Stream-based processing - different domains of expertise

The complexity of DSP/ML processing graphs stems from the need to integrate expertise across multiple domains

Different **data physical domains**:

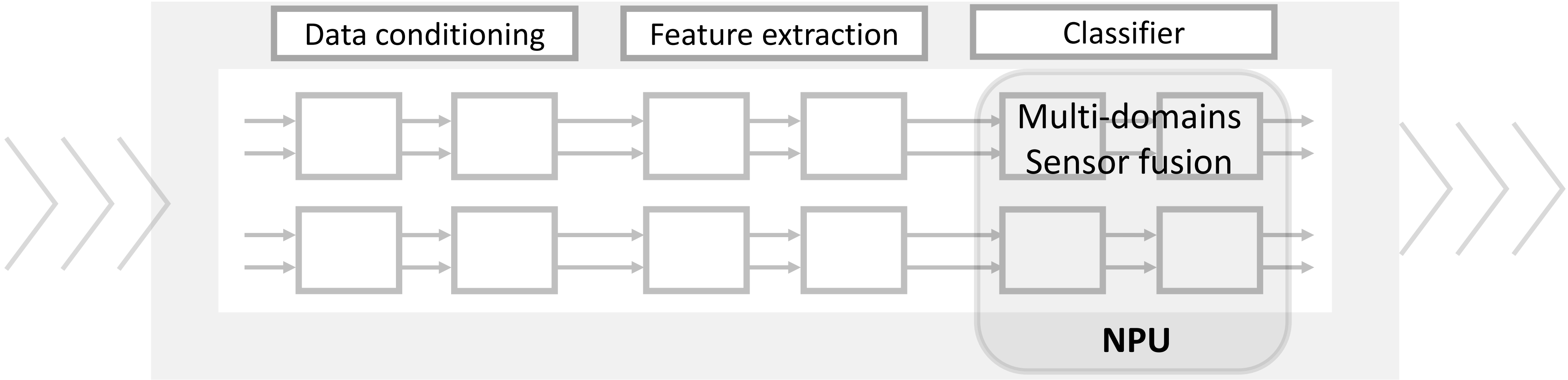
- Acoustics
- Electrical
- Chemical
- Mechanical
- . . .



Different **software engineering domains**

Stream-based processing with graph of computing nodes

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

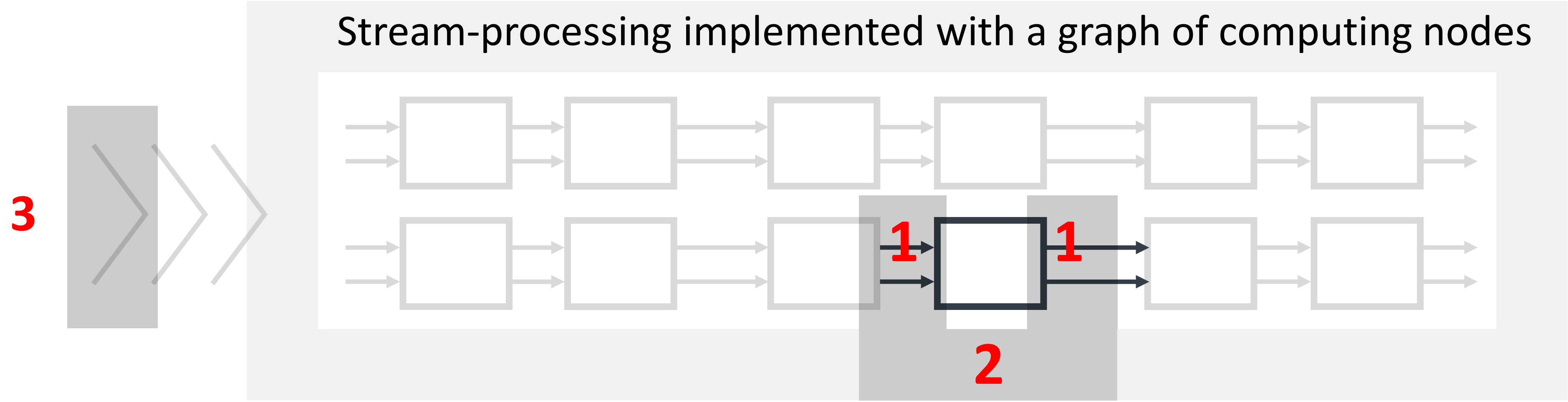


Our proposal : stream-processing is implemented with a graph of computing nodes **designed independently** (from different providers), some nodes can be pre-installed in the Flash of the platform manufacturer

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

Interfaces between nodes, the scheduler, and graph I/Os are standardized and formalized



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

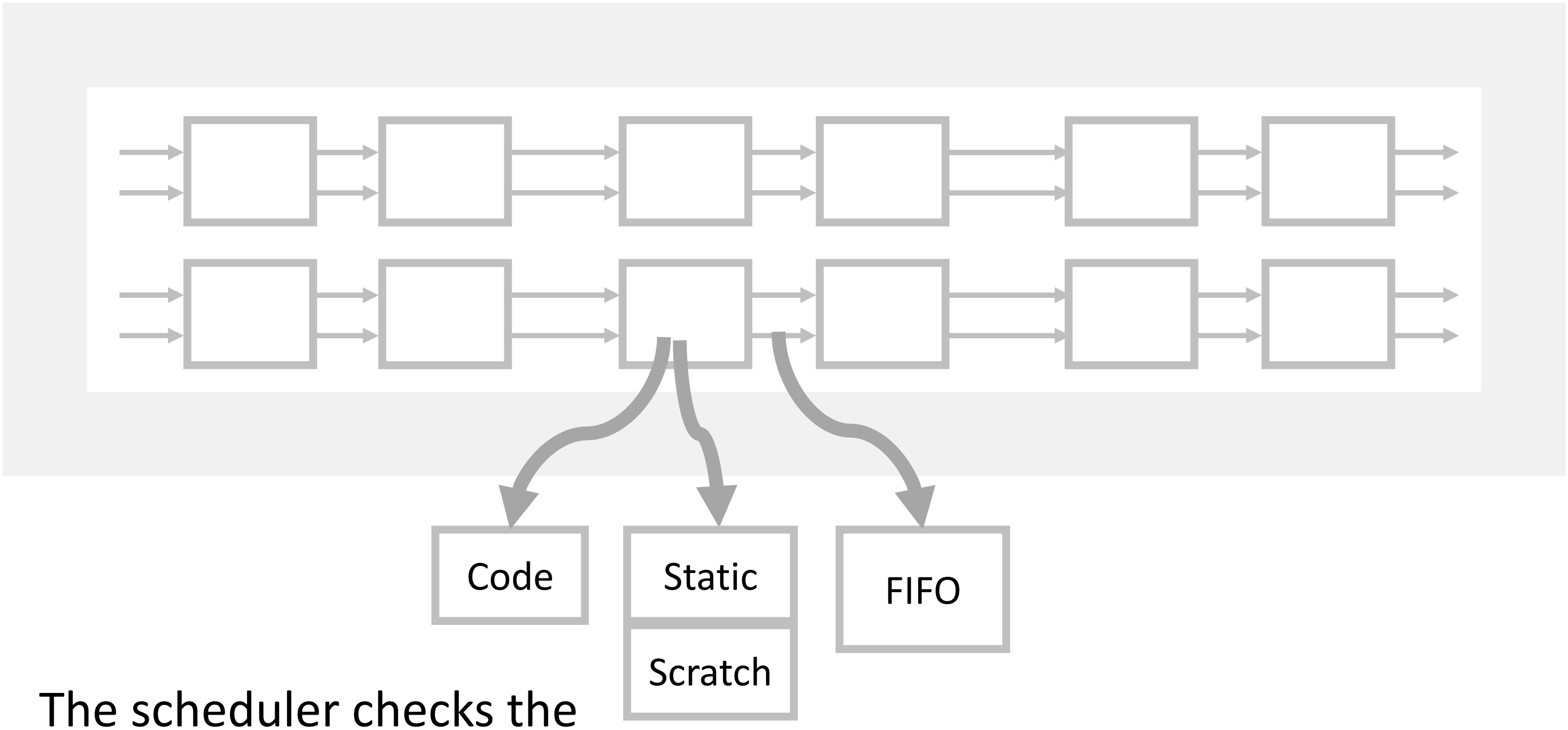
2 **Processor interface** with nodes : memory allocation and TCM, compute libraries and NPU

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

NanoGraph 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 referencing memory buffers and node addresses



The scheduler checks the FIFO buffers before calling a node instance

NanoGraph vs. Traditional Firmware/Monolithic DSP/ML

Aspect	NanoGraph (Graph Interpreter)	Traditional Firmware / Monolithic DSP/ML
Modularity, Extensibility	Highly modular, with independent and reusable nodes. Add new functions with minimal code. Integrates a tiny virtual machine (byte-code interpreter)	Tightly coupled code that is difficult to reuse or update. Adding features typically requires significant rewrite
Portability	Portable thanks to flexible interfaces	Hardware-specific and expensive to port
Time-to-Market, Reliability	Faster prototyping using pre-built nodes; no risk of bricking the device; self-recovery through the interpreter.	Larger source code size to manage all the possible interfaces and data formats. Manual Flow error management managed manually.
Resource Efficiency	Optimized for bare metal and tiny devices. Ease memory mapping to TCM.	Manual operations for memory mapping.
DSP/ML Integration	Transparent support for DSP/ML accelerators	Manual integration is more complex
Use-cases	Audio, Vision and environmental sensors, flexible data formats, sampling rate from months to MHz,	Audio only, fixed frame size
Other	Apache license, multiprocessing (AMP) and multi-threads, Arch32/64bits, stream flow error management	Proprietary or platform specific

Graph design

Compilation process using “Manifests”

The NanoGraph interpreter task is simplified with the help of “off-line” graph compilation.

Platform capabilities and I/O descriptions are defined in manifest files and used by the graph compiler.

Platform manifest of HW capabilities

List of memory banks and I/O stream characteristics.

nanoApp manifests (nodes of the graph)

Interfaces, activation keys, and memory requirements.

GUI → **Graph** of arcs and nodes

GRAPH COMPILATION

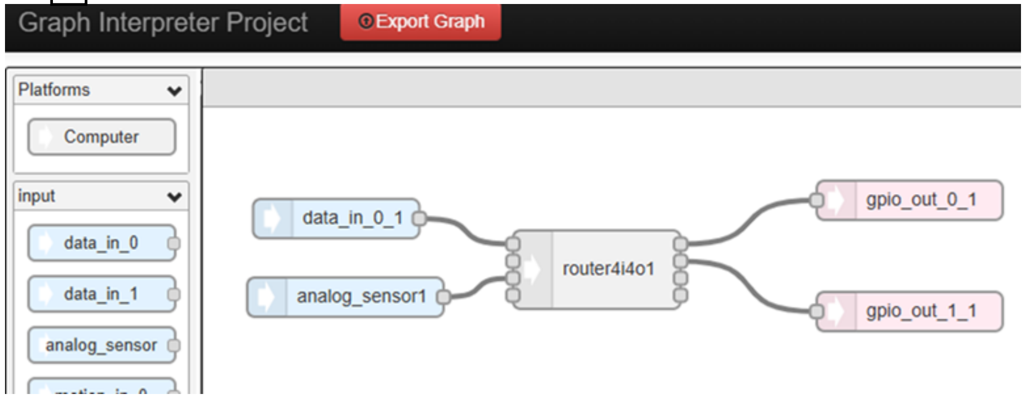
Compact Binary Representation of the Graph (in shared memory)

- Data & stream formats
- Byte-code scripts
- nanoApp list + parameters
- Circular buffer descriptors
- Allocated memory sections

GRAPH EXECUTION

Graph interpreter operations

- Parse I/Os and initiate data exchange
- Identify executable nanoApps (enough input/output space)
- Handle multiprocessing locks
- Run optional pre/post scripts
- Return control when processing is done



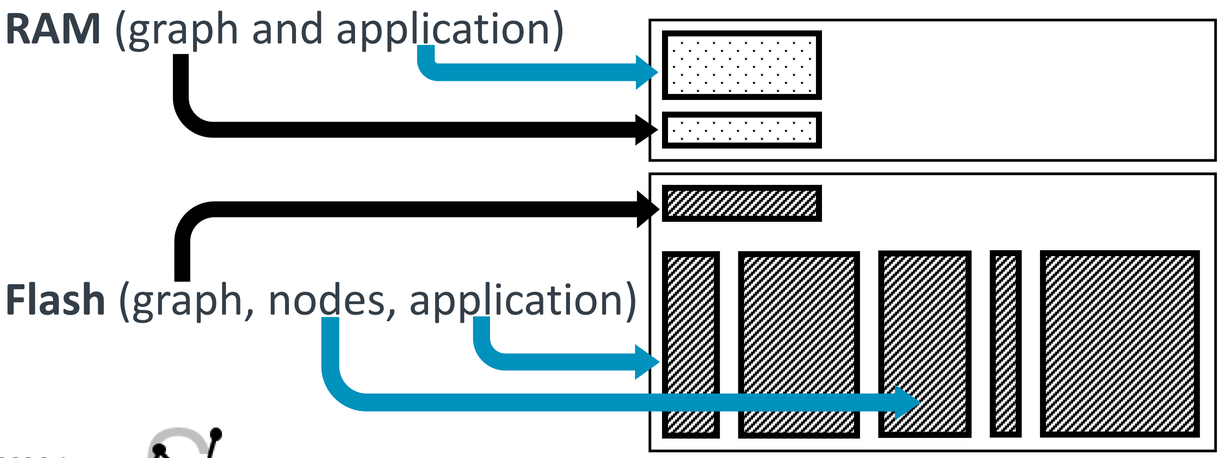
Small memory footprint

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

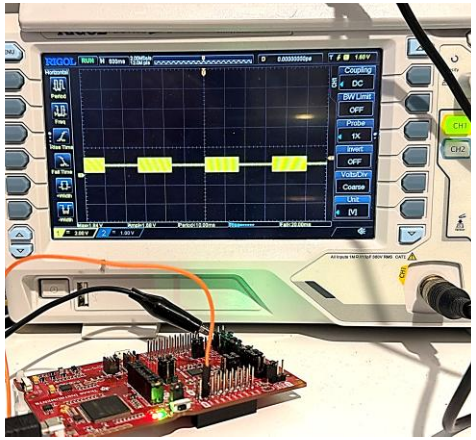
A graph size of two nodes (+ their respective parameters and a script) is in the 500Bytes range

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 “SYSCALL” gives access to nodes (set/read parameters), arcs (read/write, check access time-stamps), application callbacks (change of use-case, access to I/Os and trace), compute libraries..

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

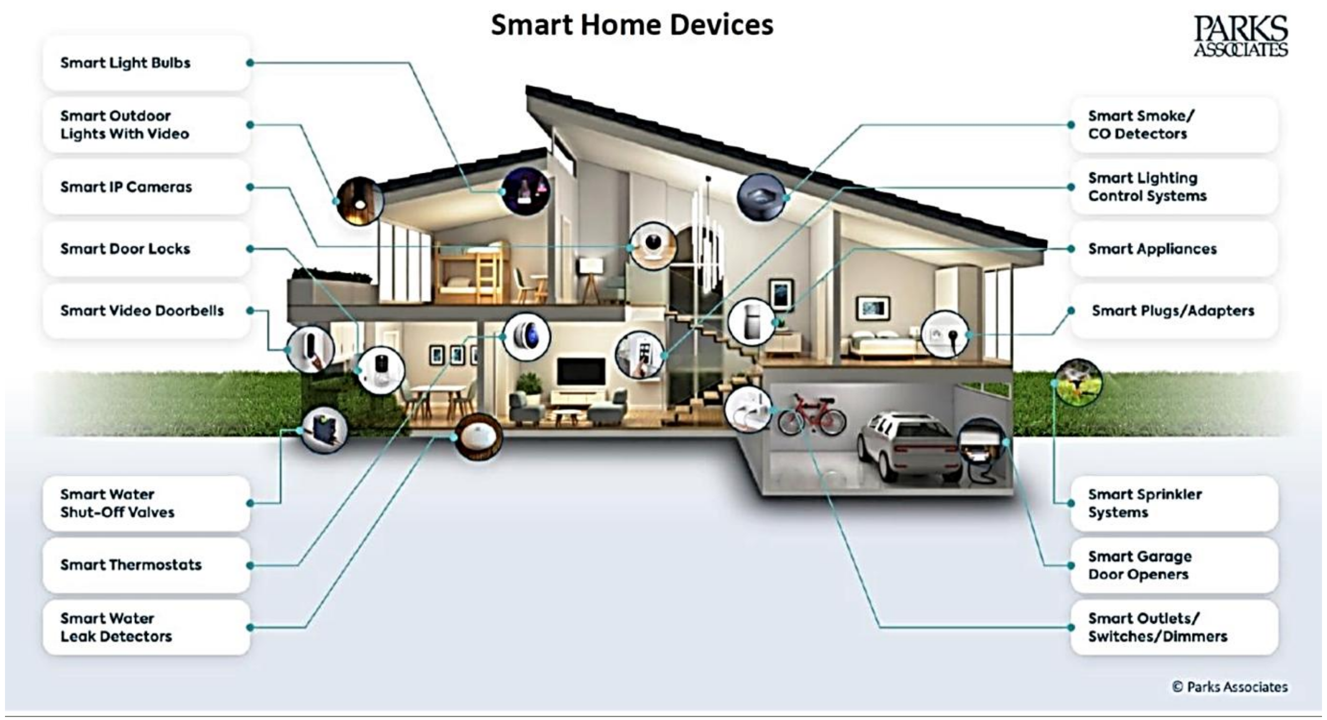
Why would you need Python for very simple operations similar to the ones of pocket calculator ?

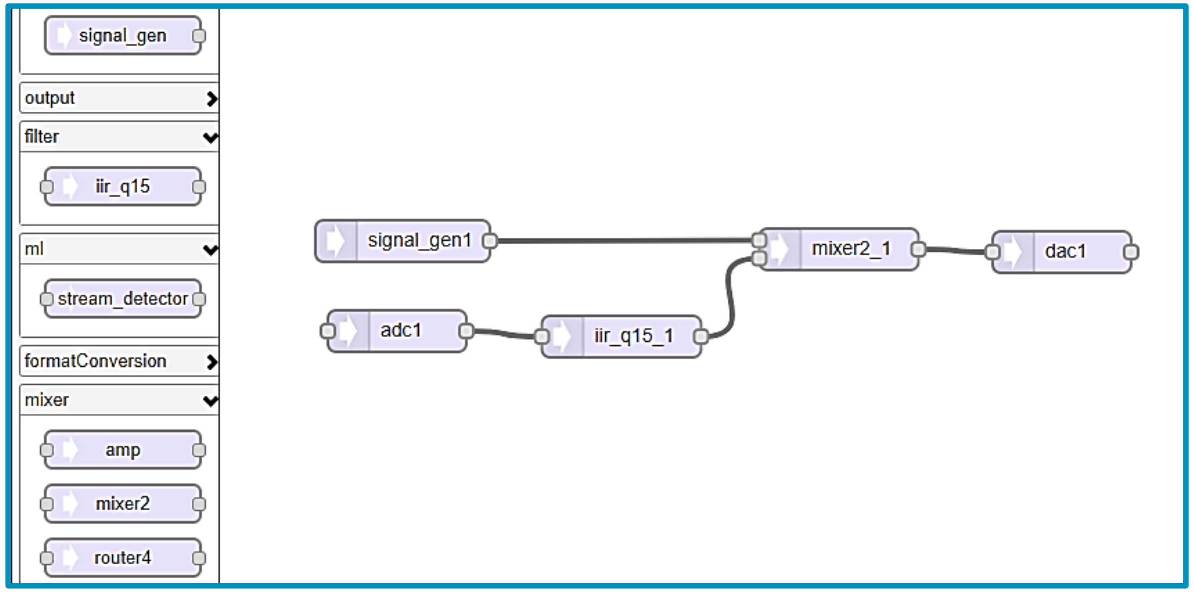
Examples of instructions

```
r6 = r5 + 3.14159            r6 = r5 + 3.14159
test r6 == r5 - r4           test if r6 == ( r5 - r4 )
if_yes call label_xyz        conditional call
r3|8 15| = r2                bit-field load of r2 to the 2nd byte of r3
r3 = r4[3]                   gather load r3 = r4[3]
r0[r1++] = SP                scatter store with post increment r0[r1++]= top of stack
syscall 4 r1 r5 r10          system call #4 with the application using 3 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 ?

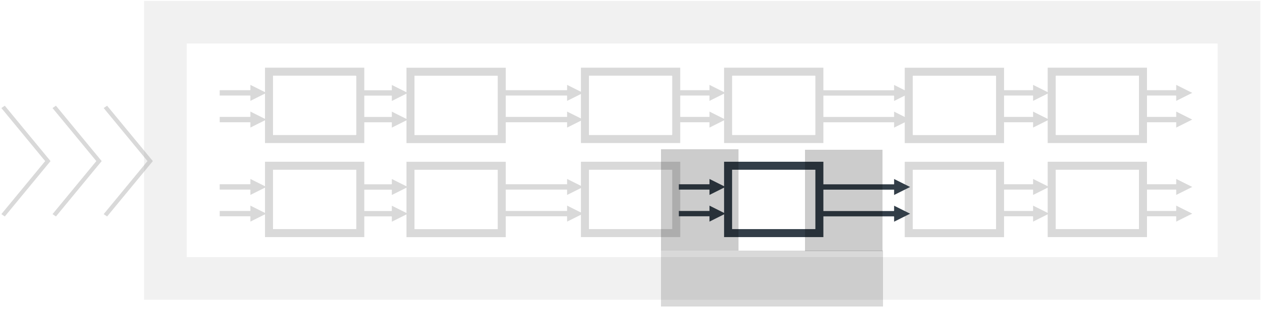




GUI proof of concept using node-RED

Backup

Manifests for nodes



```
; ----------------------------------------------------------------------------------------
; 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 2 1}      ; 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 2 1}      ; options for the raw format STREAM_S16, STREAM_FP32

end
```

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

a text file (readable syntax)

done once at node delivery

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

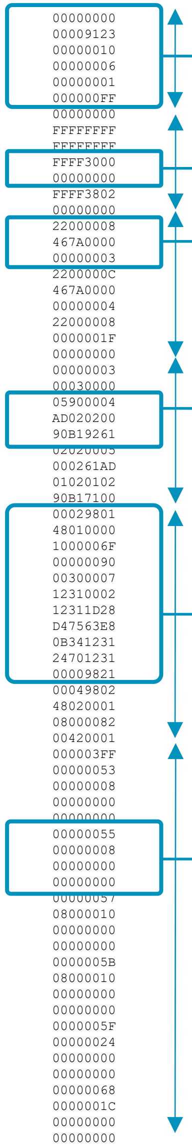
```
arm_stream_filter        0   1
arm_stream_detector      0   0
```

node name   instance index   arc output index
Node name   instance index   arc input index

```
arc_input    0  0
arm_stream_filter   0  0  0
```

Arcs at the boundary of the graph
node it is connected to

“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 :

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

Producer and consumer Data format. Alignment to cache-line width for multiprocessing.

Collision of node access protocol, flag for the need of buffer data realignment.

Processor manifest : memory mapping

```
;==================================================================
; TOP MANIFEST :
;   - paths to the files
;   - shared memories
;   - list of processors and their private memories and IOs
;   - list of the nodes installed in each processor and/or
architectures
;==================================================================
    6                      six file paths

    0 ../../../stream_platform/
    1 ../../../stream_platform/computer/manifest/
    2 ../../../stream_nodes/arm/
    3 ../../../stream_nodes/signal-processingFR/
    4 ../../../stream_nodes/bitbank/
    5 ../../../stream_nodes/elm-lang/

;==================================================================
    3           number of processors
    2           number of shared memory

;   processor and architecture ID are in the range [1..7]
;
;   Access     0 data/prog R/W, 1 data R, 2 R/W, 3 Prog, 4 R/W
;   Speed      0 slow/best effort, 1 fast/internal, 2 TCM
;   Type       0 Static, 1 Retention, 2 scratch mem

;---MEMID SIZE   A S T   Comments
    0     8000   1 0 0   shared memory
    3     1000   1 0 1   simulates shared retention memory

;==================================================================
;   Processor #1 - architecture #1 , two processors
;   proc ID, arch ID, main proc, nb mem, service mask, I/O
    1        1        1          2       15             7

;---MEMID SIZE   A S T   Comments
    1     1000   1 2 0   simulates DTCM
    2     1000   1 2 1   simulates ITCM

;-----IO AFFINITY WITH PROCESSOR 1------------------------------
    ;Path      Manifest          IO_AL_idx    Comments
    1 io_platform_data_in_0.txt       0    application proc
    1 io_platform_data_in_1.txt       1    application proc
    1 io_platform_analog_sensor_0.txt 2    ADC
    1 io_platform_audio_in_0.txt      4    microphone
    1 io_platform_line_out_0.txt      6    audio out stereo
    1 io_platform_gpio_out_0.txt      7    GPIO/LED
    1 io_platform_data_out_0.txt      9    application proc
```

Shared memory declaration

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

Private memory declaration

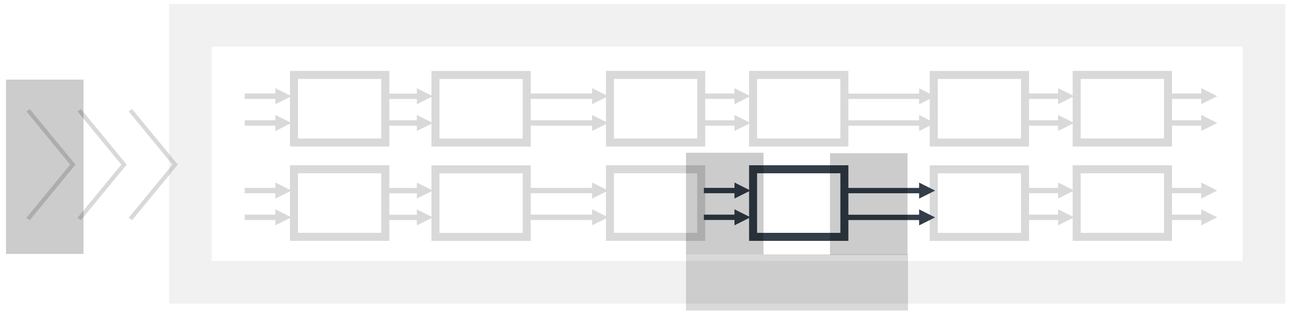
List of streams the processor can activate

List of nodes mapped to processors

```
                ... (continued) ...
;==================================================================
;   Processor #2
;   proc ID, arch ID, main proc, nb mem, service mask, I/O
    2        1        0          2       15             2

;---MEMID SIZE   A S T   Comments
    1     1000   1 2 0   index 1/2 point to different physical addresses
    2     1000   1 2 1   index 1/2 point to different physical addresses

;------IO AFFINITY WITH PROCESSOR 2------------------------------
    ;Path    IO Manifest         IO_AL_idx    Comments
    1 io_platform_data_in_0.txt        0     shared with Proc 1
    1 io_platform_motion_in_0.txt      3     accelero=gyro

;==================================================================
;   Processor #3 - new architecture, one processor
;   proc ID, arch ID, main proc, nb mem, service mask, I/O
    1        2        0          0       15             2

;---MEMID SIZE   A S T   Comments

;------IO AFFINITY WITH PROCESSOR 2------------------------------
    ;Path    IO Manifest         IO_AL_idx    Comments
    1 io_platform_2d_in_0.txt          5     camera
    1 io_platform_gpio_out_1.txt       8     GPIO/PWM

;======== ALL NODES ============================================
; scheduler algorithm :
; if the node archID > 0 then check compatibility with processor archID and exit
; if the node procID > 0 then check compatibility with processor procID and exit
;
;   Path                   Node Manifest                  PROC ARCH |  ID
    2        script/node_manifest_script.txt                 0    0  |   1 runs everywhere
    2        router/node_manifest_router.txt                 0    1  |   2 SMP on archID-1
    2    amplifier/node_manifest_amplifier.txt               0    1  |   3
    2        filter/node_manifest_filter.txt                 0    1  |   4
    2    modulator/node_manifest_modulator.txt               0    1  |   5
    2  demodulator/node_manifest_demodulator.txt             0    1  |   6
    3     detector/node_manifest_detector.txt                0    1  |   7
    3    resampler/node_manifest_resampler.txt               0    1  |   8
    3   compressor/node_manifest_compressor.txt              0    1  |   9
    3 decompressor/node_manifest_decompressor.txt            0    1  |  10
    4      JPEGENC/node_manifest_bitbank_JPEGENC.txt         0    1  |  11
    5      TJpgDec/node_manifest_TjpgDec.txt                 0    1  |  12
;-------------------------------------------------------------   |     ----
    2      filter2D/node_manifest_filter2D.txt               2    2  |  13 only archID-2
    3    detector2D/node_manifest_detector2D.txt             0    2  |  14 single processor
;
;
end ; the platform manifest ends here
;==================================================================
```

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 interleaved (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 undefined)
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) :

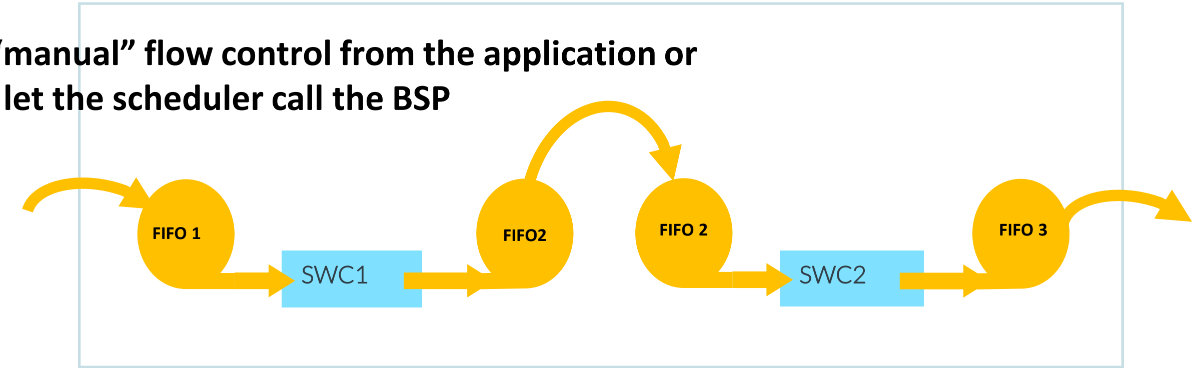
```
void arm_graph_interpreter_io_ack (uint8_t fw_io_idx, uint8_t *data,  uint32_t data_size)
```

4) One prototype for all nodes :

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

command = reset, set parameters, run, stop

"manual" flow control from the application or let the scheduler call the BSP



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

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