

Technology and Design Tools for Multicore Embedded Systems Software Development

Yuriy Sheynin, Alexey Syschikov, Boris Sedov

Saint Petersburg State University of Aerospace Instrumentation

Why do we need such technology?

1. "Two-in-one" developer is required:



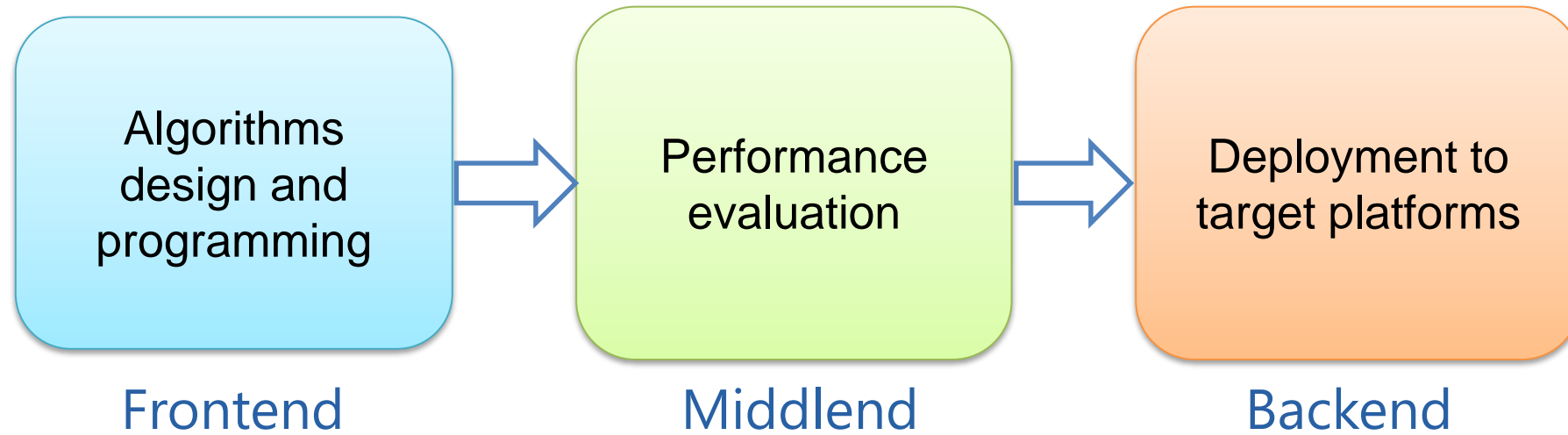
skilled domain experts +



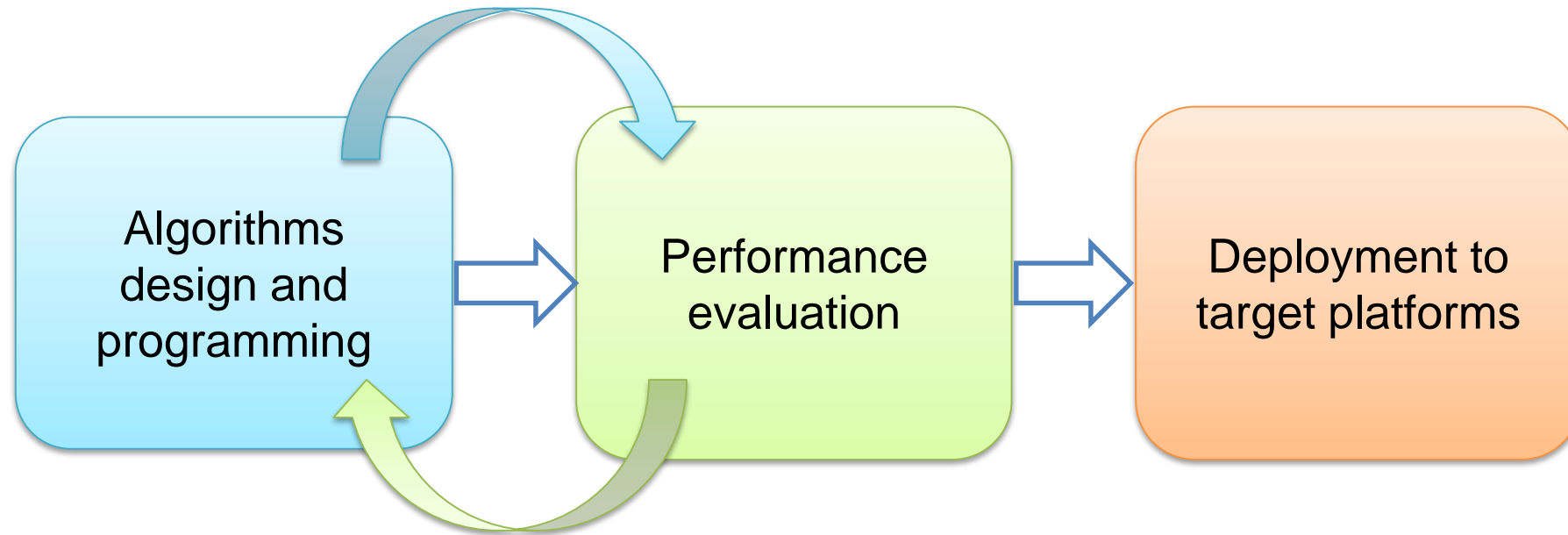
skilled programmer

2. Contradictive requirements to hardware platforms
3. Development of an algorithm and program should be started before the selection of a specific platform
4. Hardware platforms become more and more complex, includes many cores, are heterogeneous in all aspects (cores, memory, interconnect)
5. In order to achieve the necessary requirements an adaptation of algorithms for the platform and the platform for algorithms is needed

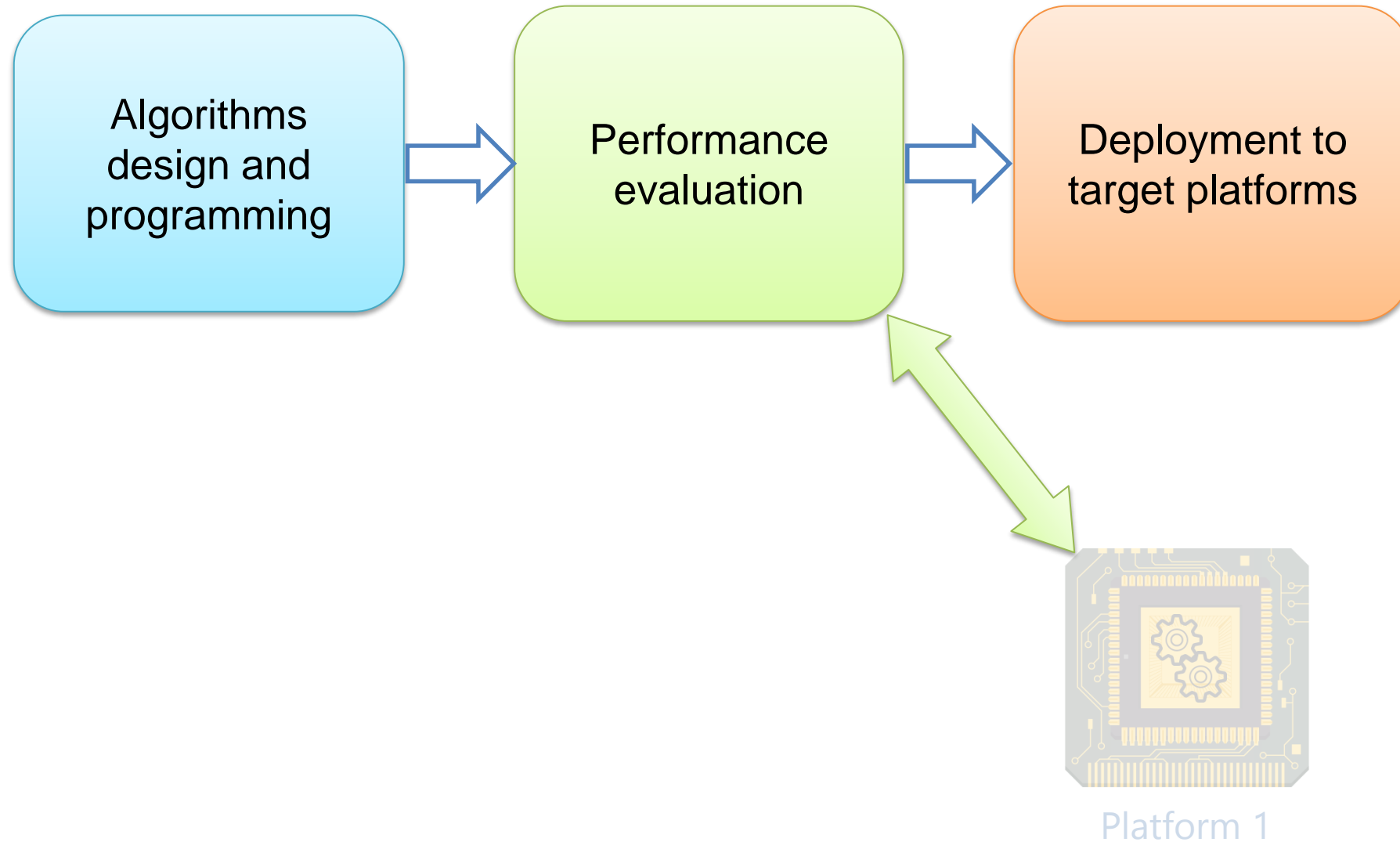
The life cycle of the program in the technology



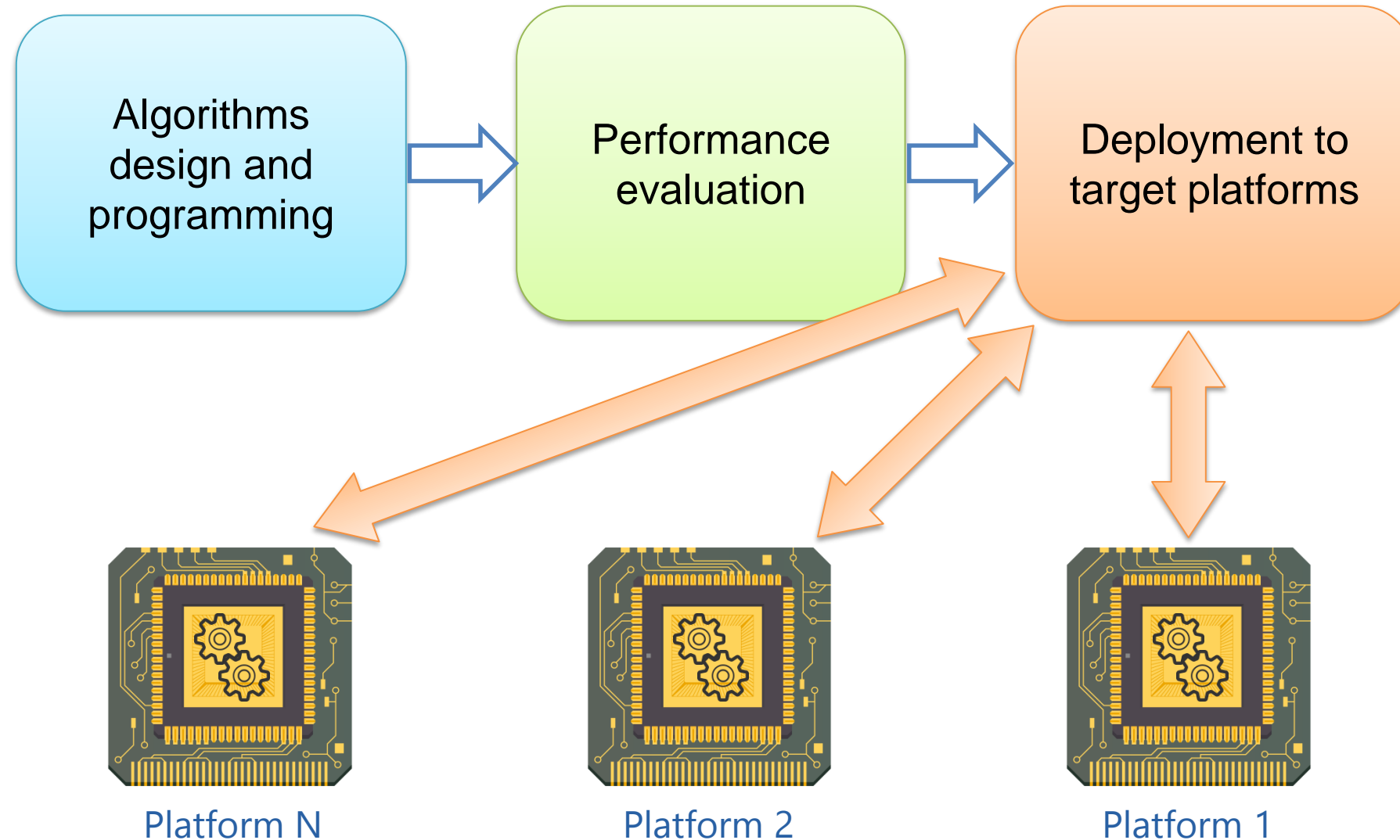
The life cycle of the program in the technology



The life cycle of the program in the technology

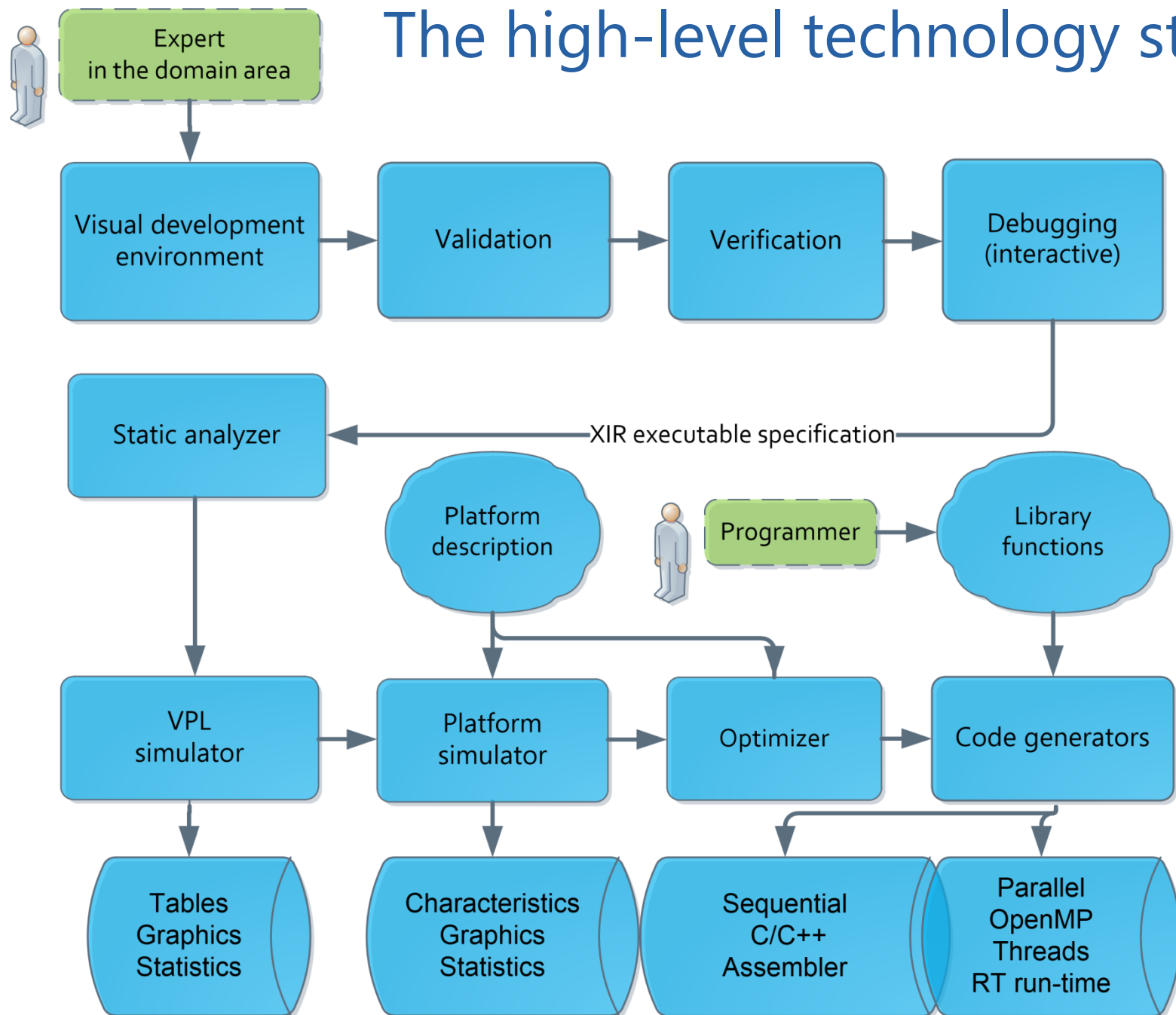


The life cycle of the program in the technology



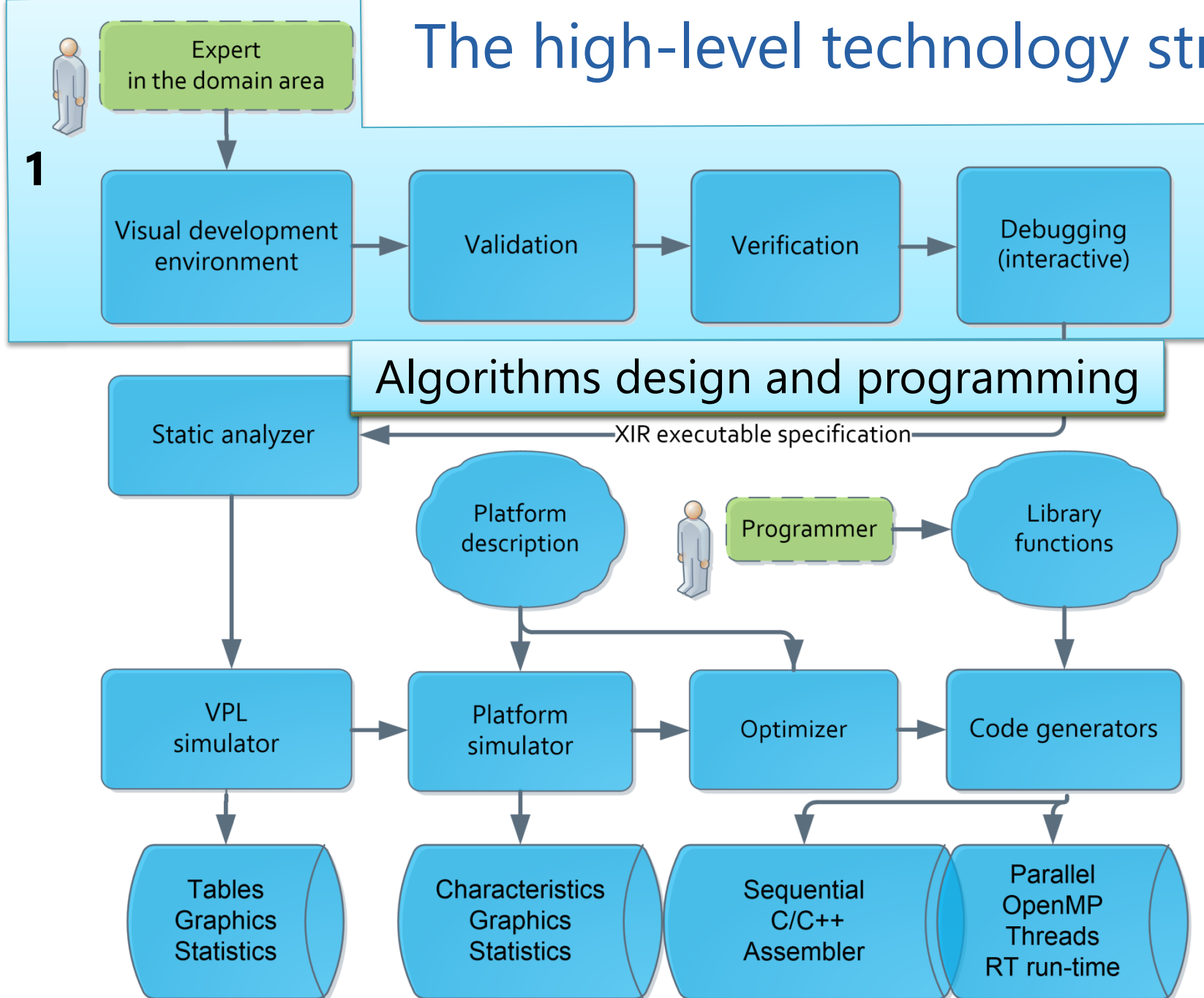


The high-level technology structure



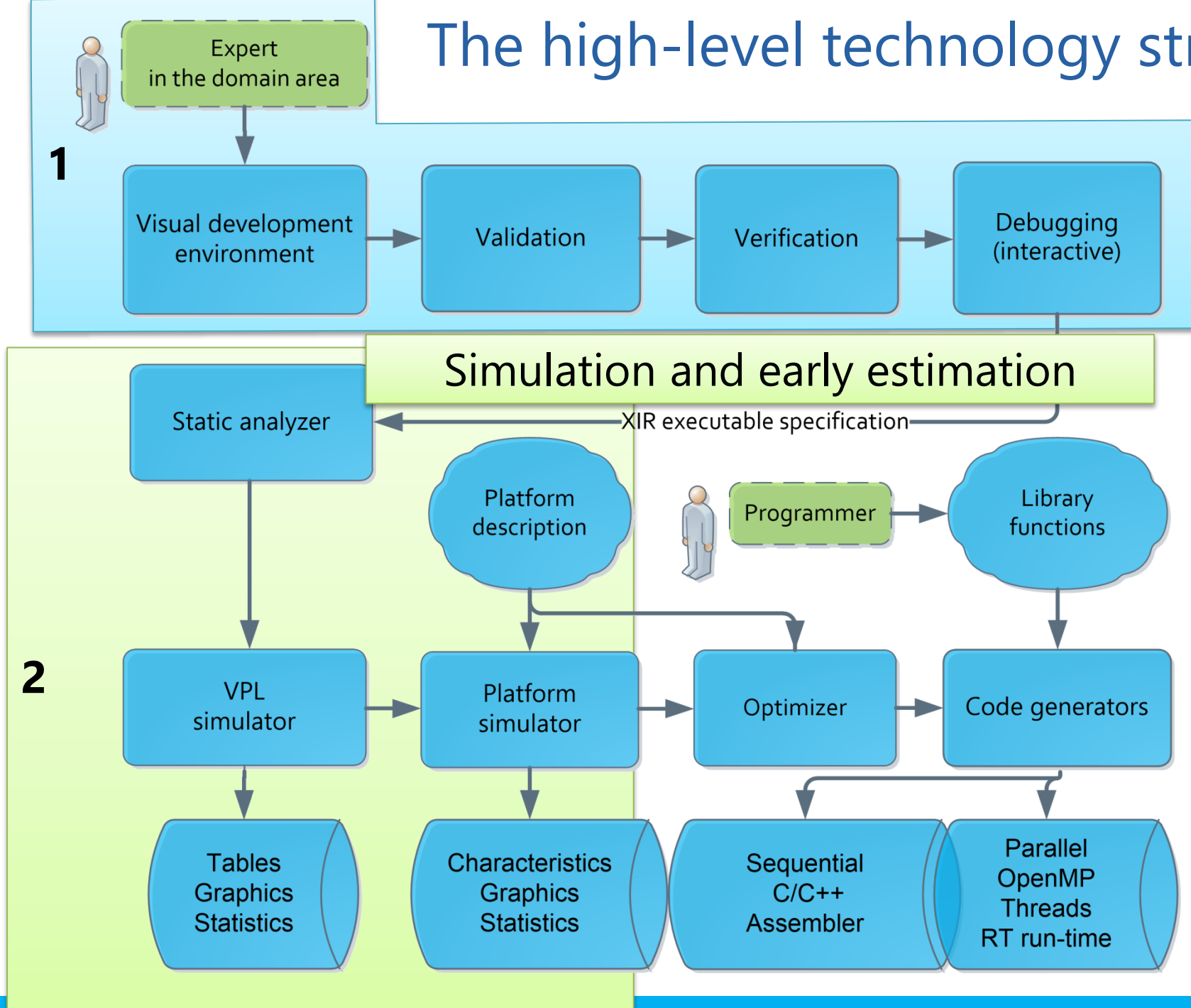


The high-level technology structure



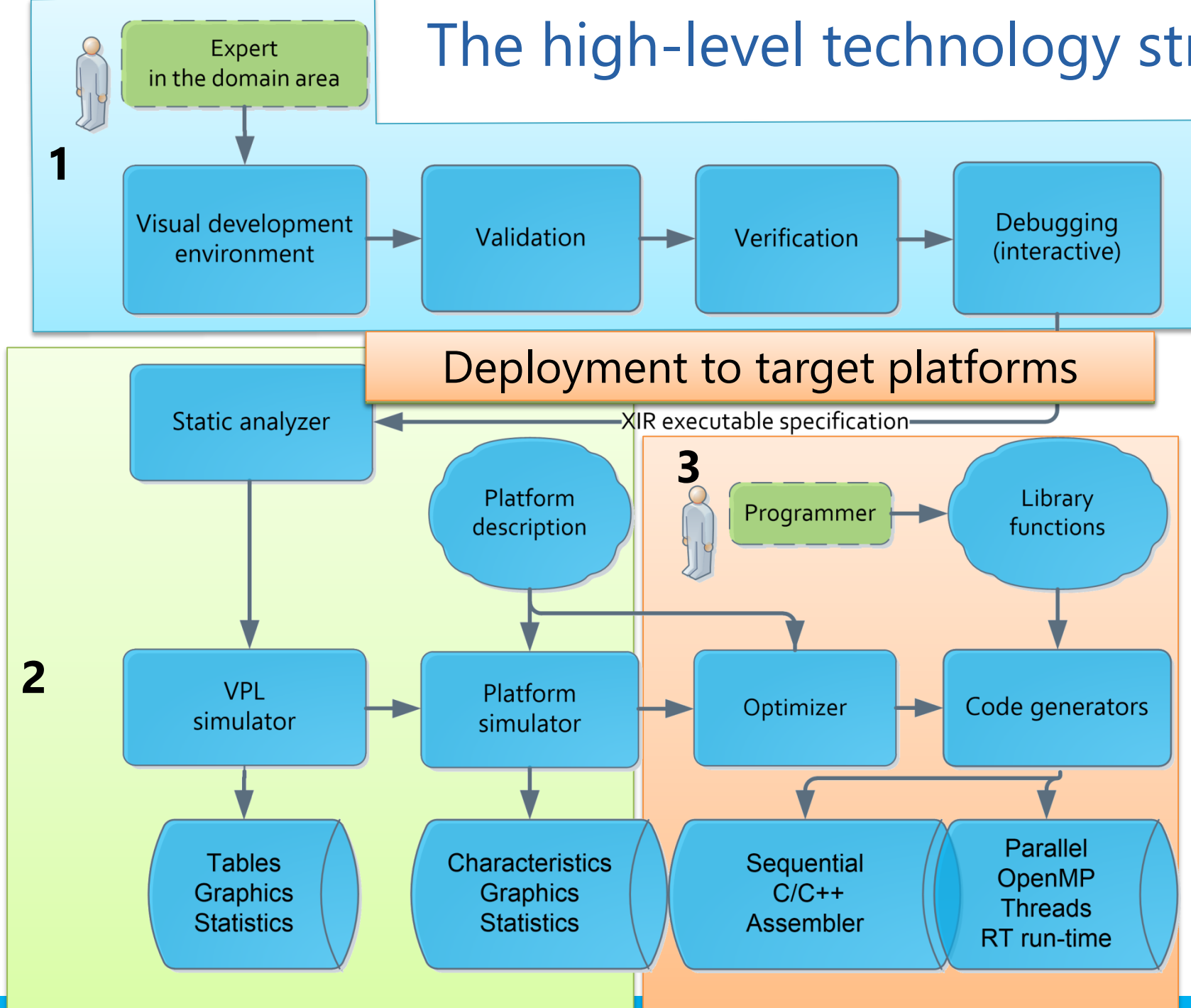


The high-level technology structure



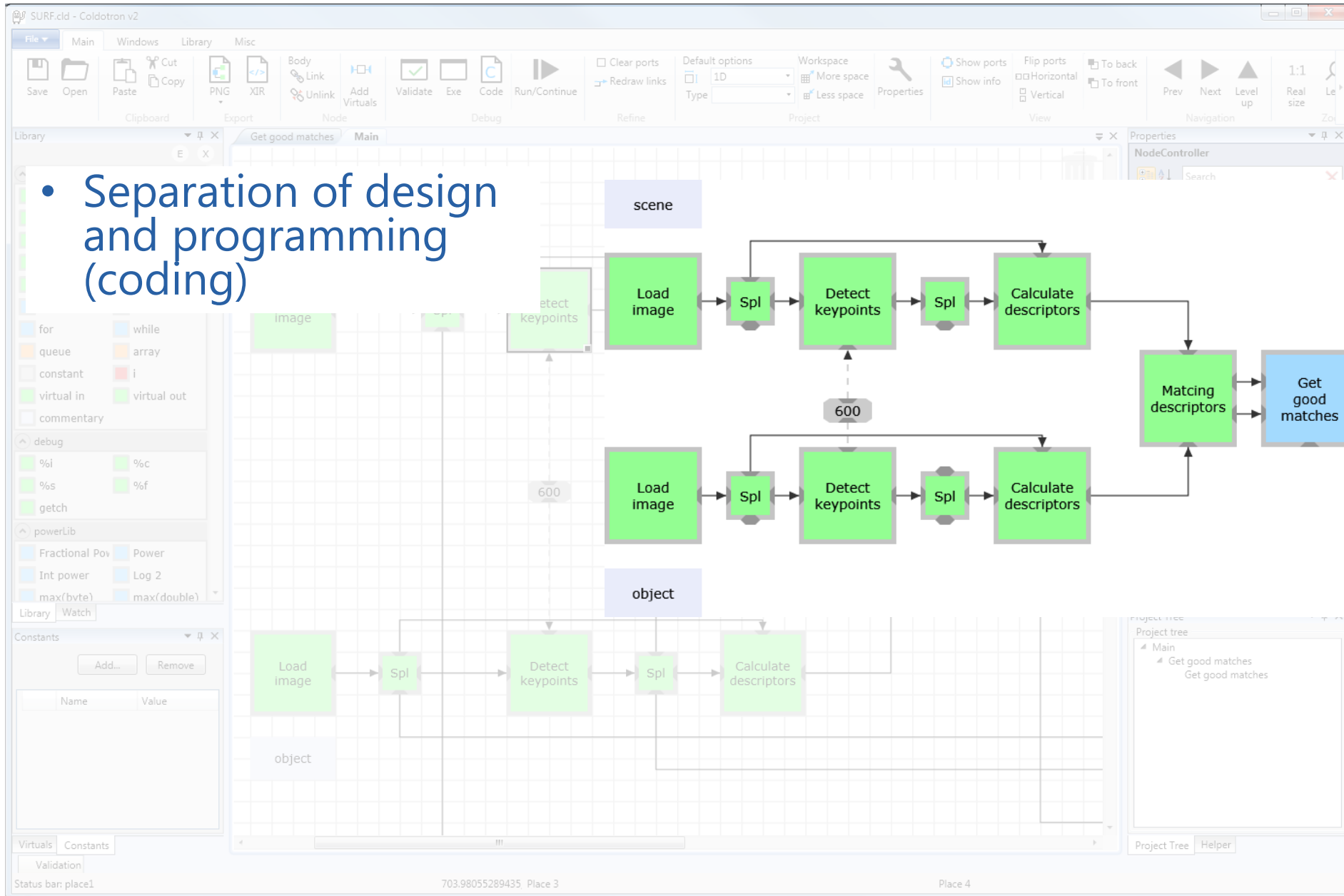


The high-level technology structure



The Visual Programming Language (VPL)

- Separation of design and programming (coding)



The Visual Programming Language (VPL)

• Separation of design and programming (coding)

Expert

Programmer

scene

image

600

600

Load image

Spl

Detect keypoints

Spl

Calculate descriptors

Matching descriptors

Get good matches

Template text

Open Save Save As...

//

Title Parameters

```
int dhCalcAirlight ( DataLink *in11, DataLink *in31, DataLink *out2: )  
{  
    memcpy(&p, in11->Data, sizeof(int*));  
    CImg<double>* brightestDarkPixels = (CImg<double>*)p;  
    memcpy(&p, in31->Data, sizeof(int*));  
    CImg<double>* data = (CImg<double>*)p;  
    float airLight[3] = { 0.0, 0.0, 0.0 };
```

C/C++

OpenCL

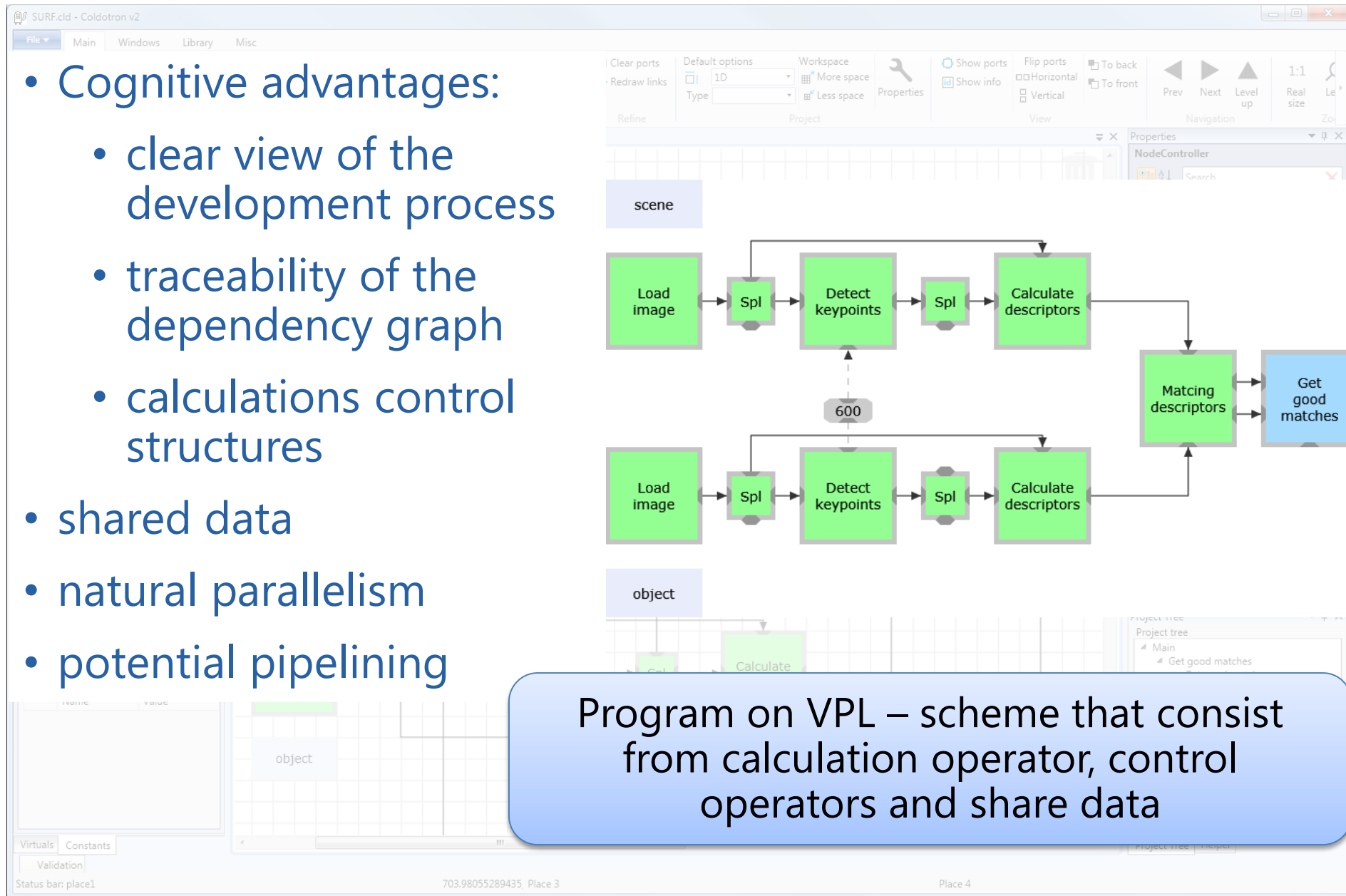
Asm

Project Tree Helper

Place 4

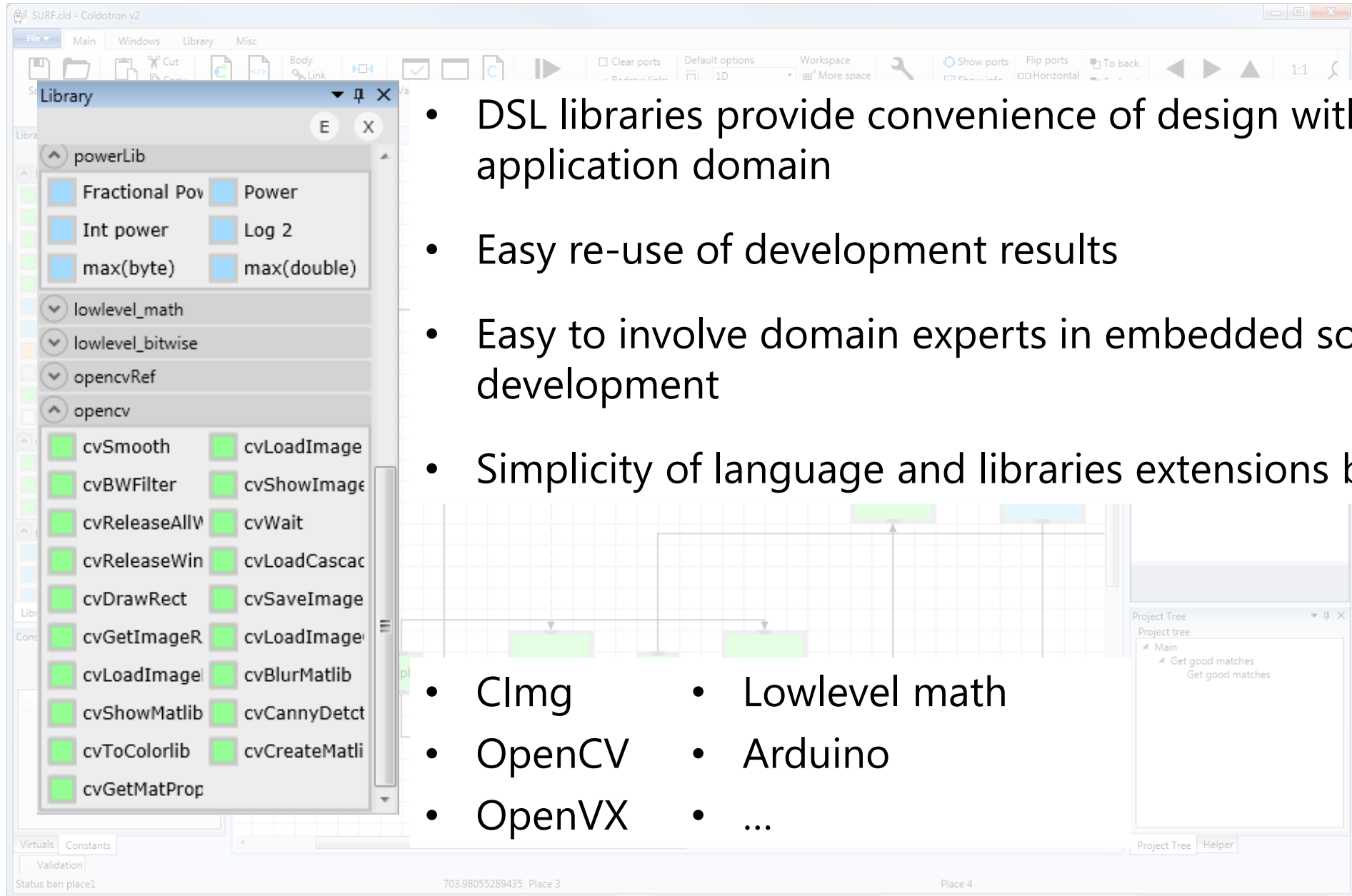
The Visual Programming Language (VPL)

- Cognitive advantages:
 - clear view of the development process
 - traceability of the dependency graph
 - calculations control structures
- shared data
- natural parallelism
- potential pipelining



Program on VPL – scheme that consist from calculation operator, control operators and share data

VPL and Domain Specific Programming



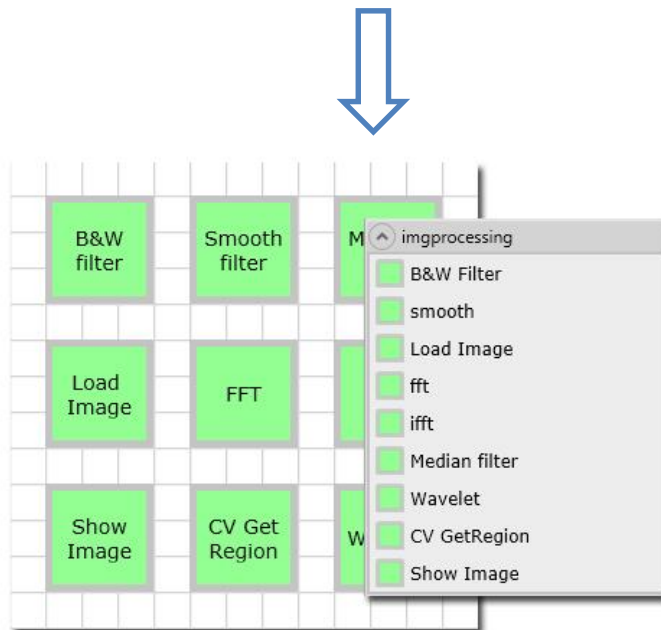
The screenshot shows the SURF.cld IDE interface. On the left, a 'Library' window displays a list of functions categorized into 'powerLib', 'lowlevel_math', 'lowlevel_bitwise', 'opencvRef', and 'opencv'. The 'opencv' category is expanded, showing various image processing functions like cvSmooth, cvLoadImage, cvBWFILTER, etc. On the right, a 'Project Tree' window shows a project structure with a 'Main' folder containing two sub-items, 'Get good matches'. The main workspace area is partially visible, showing a grid and some colored blocks.

- DSL libraries provide convenience of design within an application domain
- Easy re-use of development results
- Easy to involve domain experts in embedded software development
- Simplicity of language and libraries extensions by users

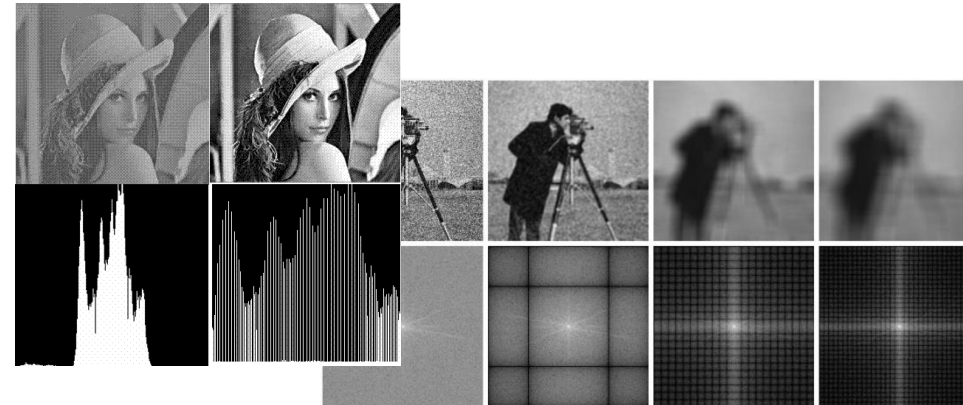
- CImg
- OpenCV
- OpenVX
- Lowlevel math
- Arduino
- ...

An example: DSL creation for image processing (OpenCV)

1. Analysis of the domain area



2. Creation of the functional elements (FE) library



3. Development of FE functionality on C++ & OpenCV

```
int CVBWFilter(DataLink *in11, DataLink *out21)
{
    IplImage* src=0;
    IplImage *im_bw=0;

    src = DecodeImage(in11,src);
    im_bw = cvCreateImage(cvGetSize(src),IPL_DEPTH_8U,1);
    cvCvtColor(src,im_bw,CV_RGB2GRAY);
    EncodeImage(im_bw,out21);

    cvReleaseImage(&src);
    cvReleaseImage(&im_bw);
    return 0;
}

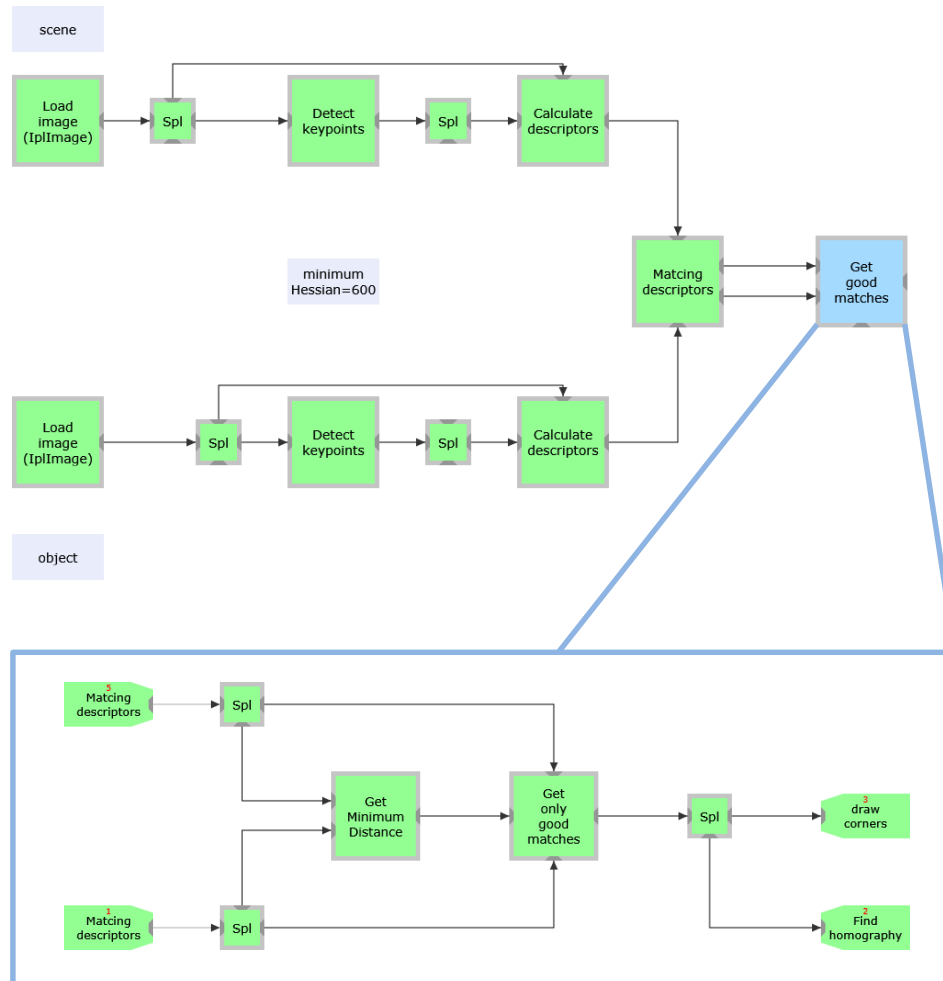
int CVSmooth(DataLink *in11, DataLink *out21,DataLink *in31,
             int radius,int filter_type)
{
    IplImage* src=0;
    IplImage* smooth=0;
    int r;

    memcpy(&r,in31->Data,sizeof(int));
    src = DecodeImage(in11,src);
    smooth = cvCloneImage(src);
    cvSmooth(src,smooth, filter_type, radius, radius,0.0,0.0);
    EncodeImage(smooth,out21);

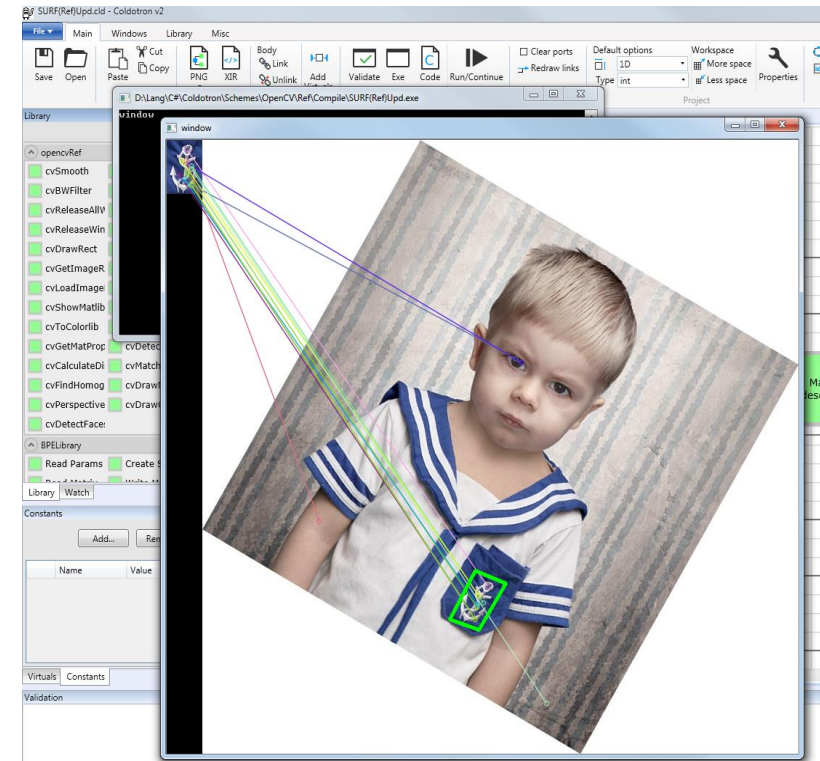
    cvReleaseImage(&src);
    cvReleaseImage(&smooth);
    return 0;
}
```


An example: DSL usage for image processing

Image recognition



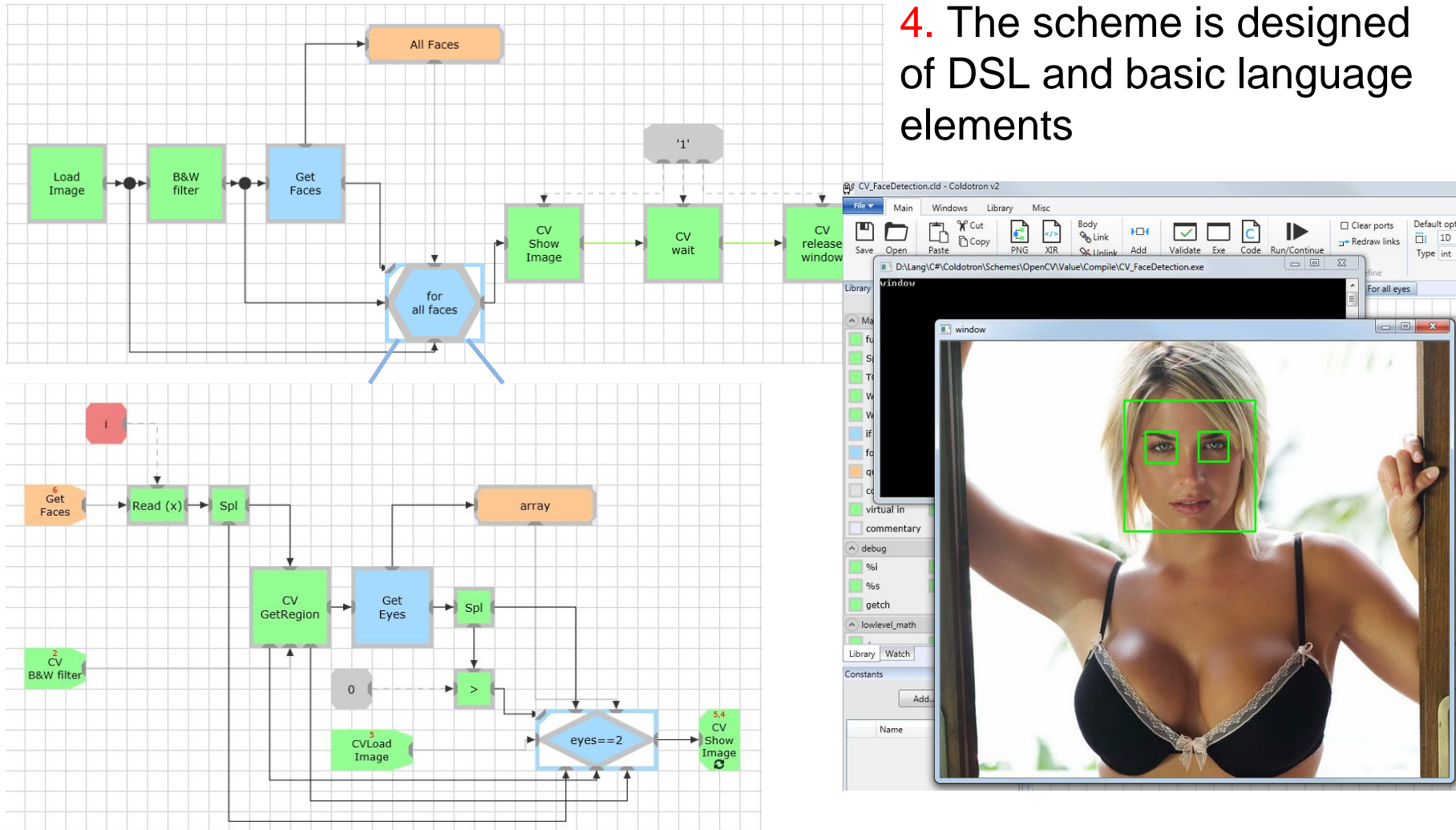
4. The scheme is designed of DSL and basic VPL language elements



An example: DSL usage for image processing

Face/eyes recognition

4. The scheme is designed of DSL and basic language elements



About OpenVX

OpenVX

- C-based programming approach with mixed C/non-C computing model
- Includes functions and data types of video processing domain area
- Functions library can be expanded, but it is inconvenient (non-portable)

OpenVX support in VIPE

- Full implementation for spec. v.1.0.1 (working on v.1.1)
- VPL:
 - DSL for OpenVX functions
 - OpenVX-specific data objects
- Code generation:
 - Plain C mode with OpenVX functions (vxu)
 - OpenVX graph mode
 - **Mixed mode with OpenVX graphs and other DSLs**

An OpenVX graph – a limited subset of VPL program schemes.
VPL scheme + OpenVX functions combines all benefits

Asynchronous Growing Processes (AGP) formal computational model

AGP defines:

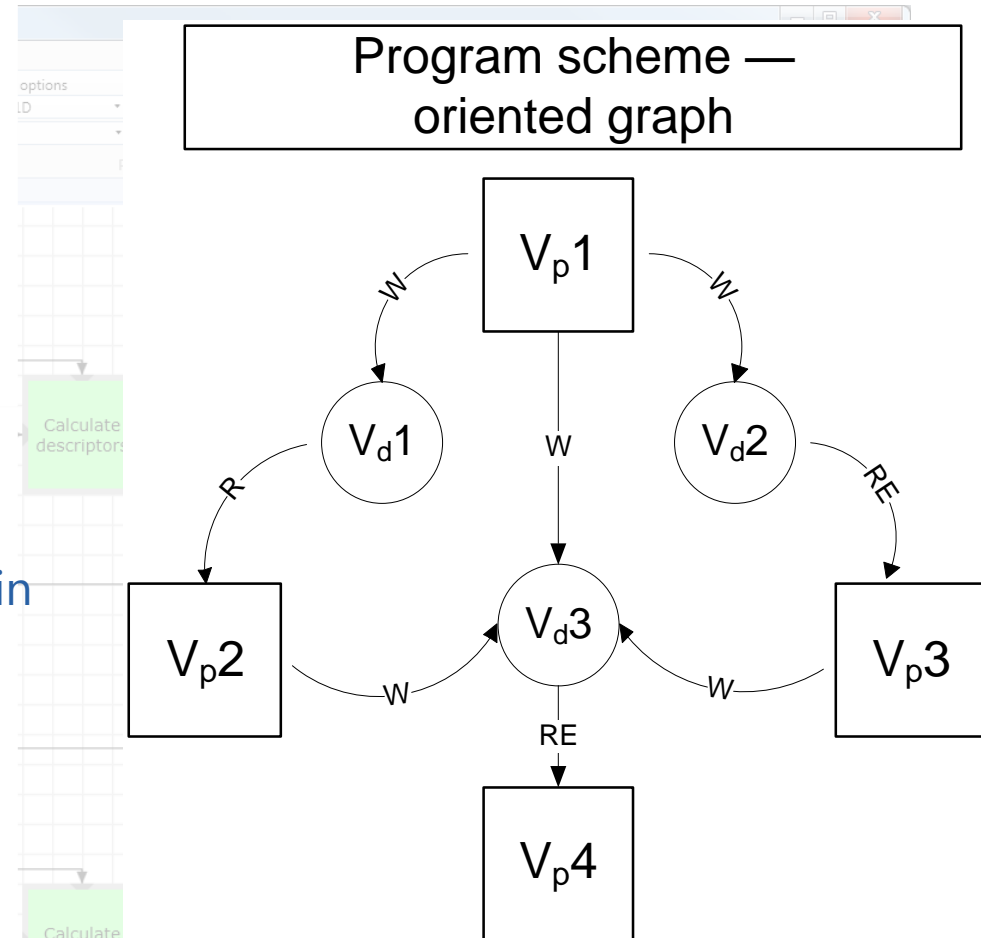
- VPL language syntax
- semantics of VPL language objects
- control units

AGP provides:

- formal verification
- identical results in different run-time environments
- dynamics of parallel computations
- combination of working in shared and distributed memory models

AGP – the single model for all types of parallel computations and kernel - data interaction (shared memory / message passing)

Domain



V_p – operator vertex,
 V_d – data-object vertex,
 $W/R/RE$ – arcs (links) marking

Visual programming environment: VIPE

Terminator demo.cld - VIPE

File Main Additional Windows Library Misc

Save Open Add Virtuals Validate Exe Code Run / Continue

Set print Set getchar Properties Clear ports Show ports Show info 1:1 Real size Less More Autoscale Undo Redo

Library

- i vxData
- virtual in virtual out
- commentary glNode
- debug
- lowlevel_math
- opencvRef
- opencvVal
 - cvSmooth cvLoadImage
 - cvBWFILTER cvShowImage
 - cvReleaseAll... cvWait
 - cvReleaseWi... cvLoadCasca...
 - cvDrawRect cvSaveImage
 - cvGetImage... cvLoadImag...
 - cvLoadImag... cvBlurMatlib

Constants

Name	Value
ENABLE_EYES_D	0
ENABLE_EYES_D	0
DRAW_FRAMES_I	1

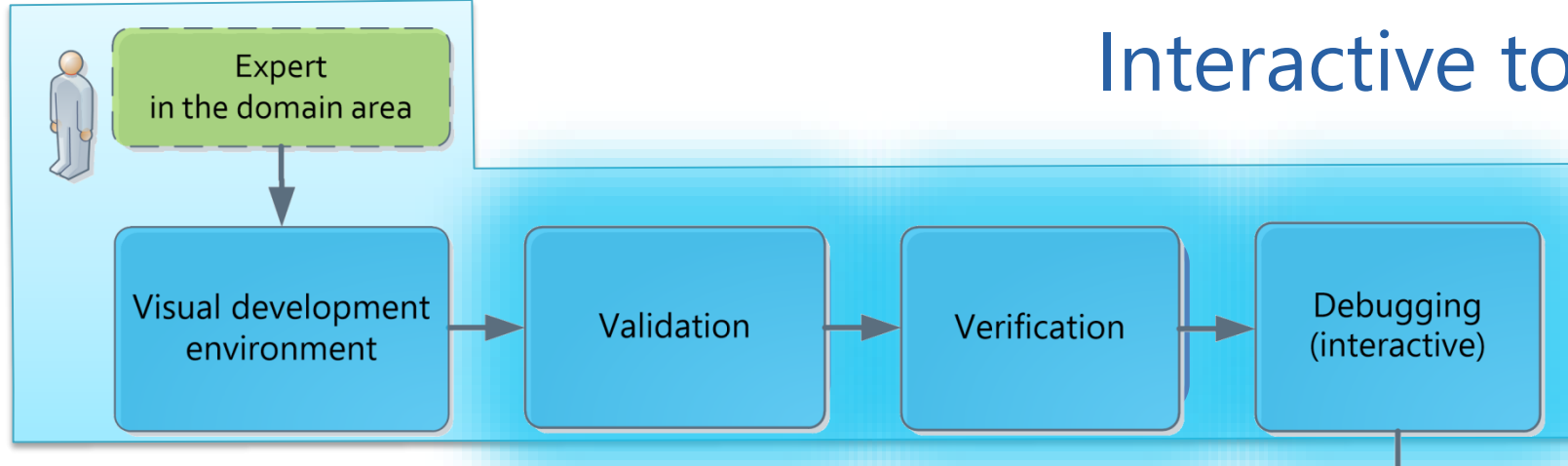
Constants Virtuals Globals

Validation Watch Output

X:586; Y:577

VIPE version: 2.2.5.0

Interactive tools



Development process support tools:

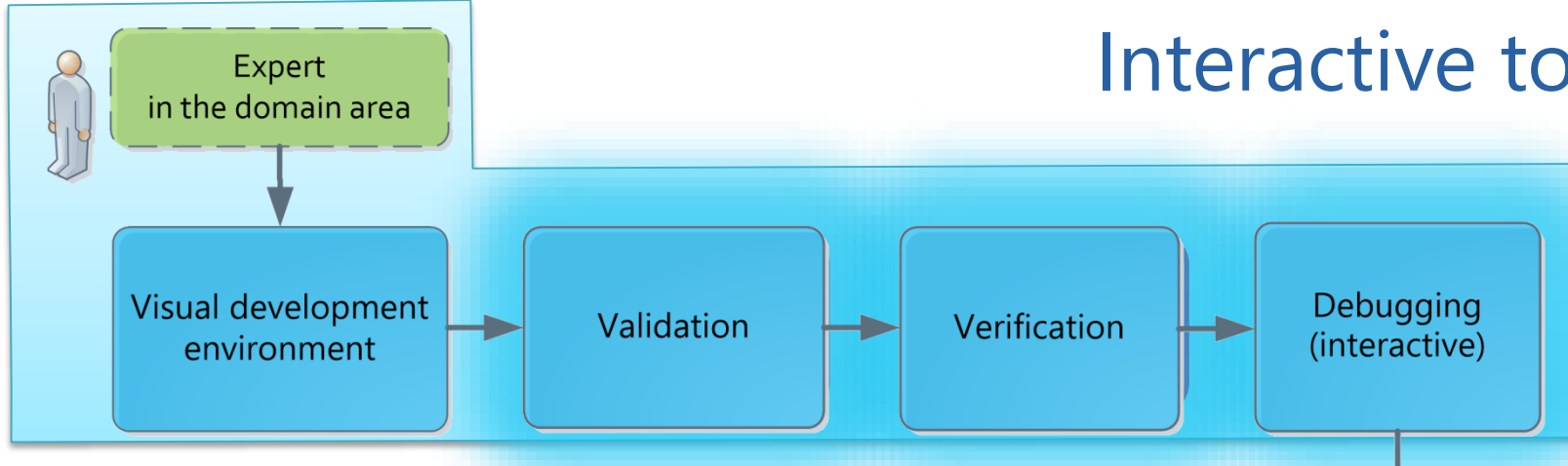
- parallel program scheme validation

- syntax checking
- types checking
- links correctness
- scheme completeness
- etc.

The screenshot shows the SURF development environment. The main workspace displays a block diagram with nodes labeled 'Mod', 'Spl', and 'I=' connected by arrows. A red box highlights a 'cond' node. The 'Validation' status bar at the bottom indicates 'Virtual 102 has 0 links'. A 'Watch' window on the right shows a table of variables and their values.

Name	Values
Link 425	[X]
Link 3651	[X]
Link 3652	[10]
array 446	[1][2][3][4][5][6][7][8][9][10]

Interactive tools



Development process support tools:

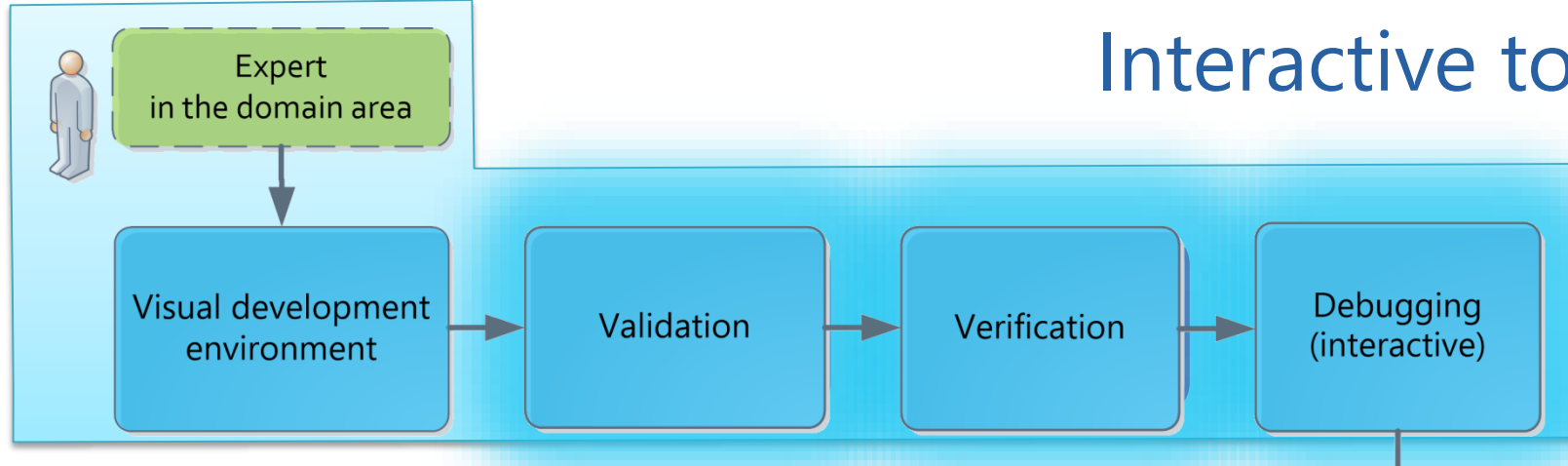
- parallel program scheme validation
- verification (in progress)
 - unambiguity
 - deadlocks
 - livelocks
 - finitness
 - etc.

The screenshot shows a development tool interface with several components:

- Library:** A list of components including Main, fur, Spl, TG, Wr, if, for, que, cor, virt, cor, debu, %i, %s, get, pow, Fra, Int, ma, Library, and Constant.
- Main window:** Displays a program scheme diagram with nodes labeled 'Mod', 'Spl', and 'I=' connected by arrows. A red box highlights a 'cond' node.
- Watch window:** Shows a table of variables and their values.
- Validation status:** A message at the bottom left states 'Virtual 102 has 0 links'.

Name	Values
Link 425	[X]
Link 3651	[X]
Link 3652	[10]
array 446	[1][2][3][4][5][6][7][8][9][10]

Interactive tools

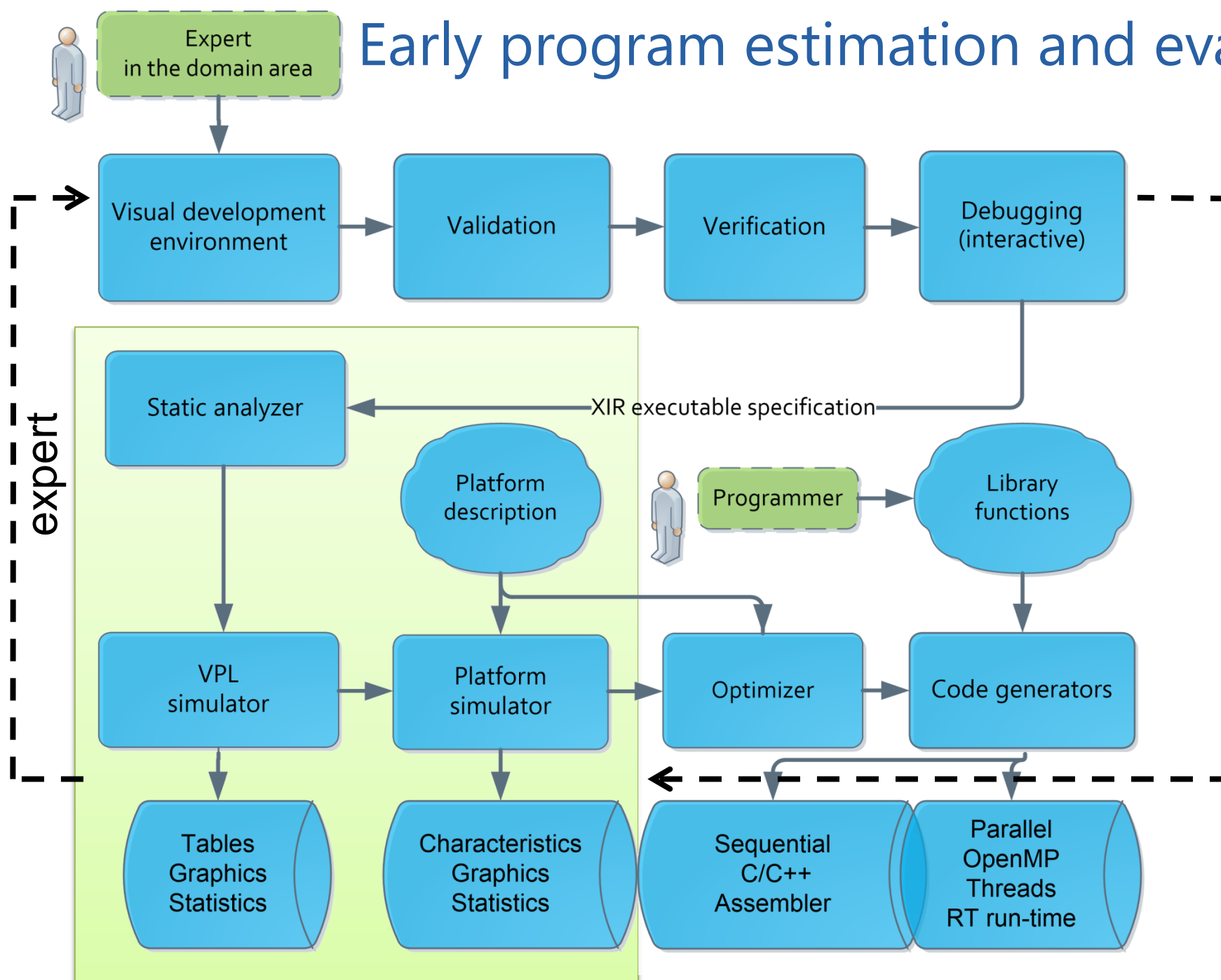


Development process support tools:

- parallel program scheme validation
- verification (in progress)
- interactive debugging, etc.
- step-by-step debug
- breakpoints
- watches
- data transfers
- computation traces
- operators executions
- functional debugging by serial execution
- etc.

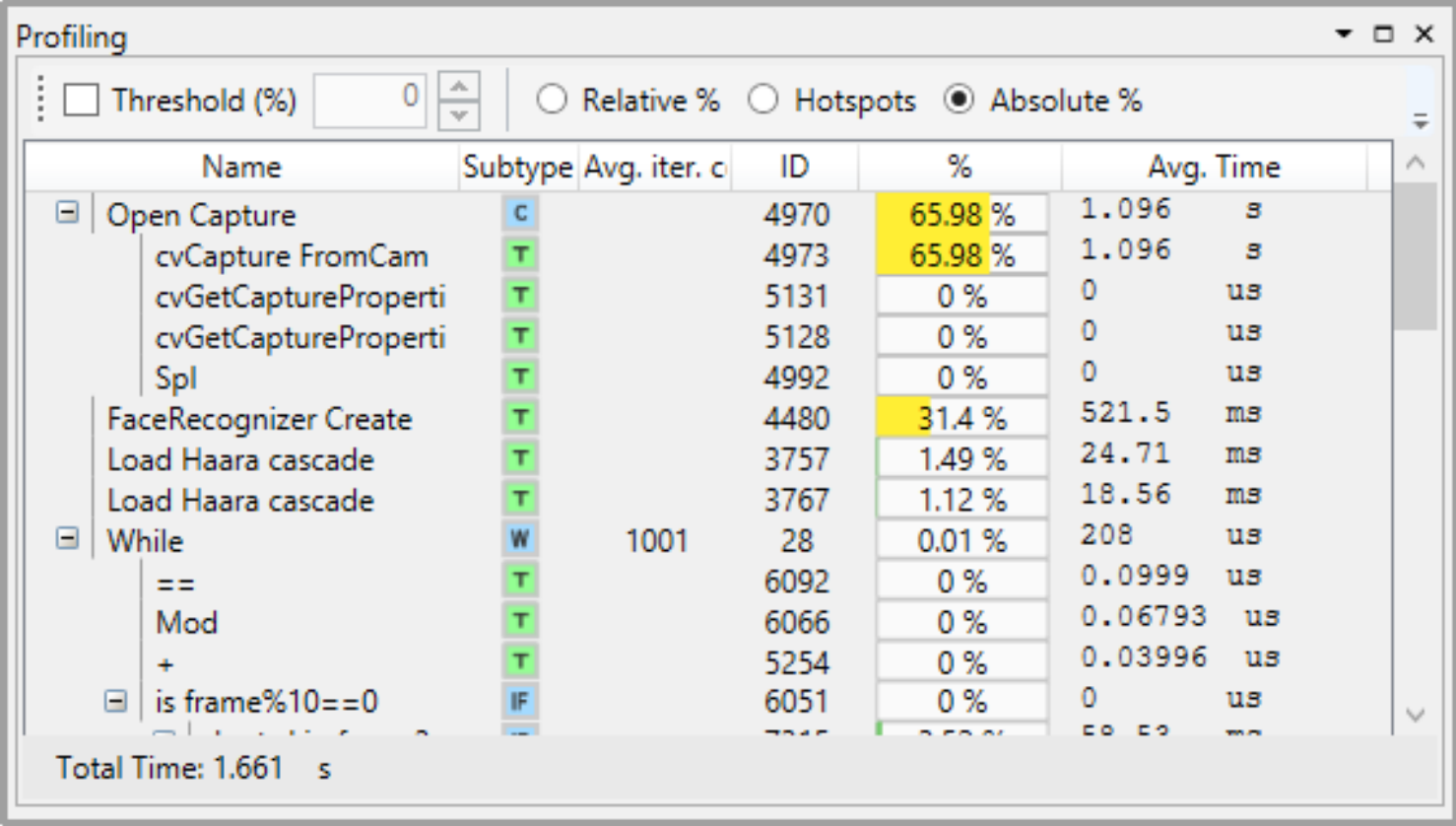
The screenshot shows a development tool interface. On the left is a 'Library' pane with various symbols. The main area displays a control flow graph with nodes labeled 'Mod', 'Spl', and 'I=' connected by arrows. A red box highlights a 'cond' node. To the right is a 'Watch' window showing the expression 'i=9; i >= 0;' with a value of '434'. Below the watch window is a 'Validation' status bar indicating 'Virtual 102 has 0 links'. In the top right corner, there is a navigation control panel with buttons for 'Prev', 'Next', 'Level up', 'Real size', and 'Zoom', along with a 'Navigation' label and a 'Zoi' indicator.

Early program estimation and evaluation



Performance evaluation. Visual Profiler

Hot-spots detection



Profiling

Threshold (%) Relative % Hotspots Absolute %

Name	Subtype	Avg. iter. c	ID	%	Avg. Time
Open Capture	C		4970	65.98 %	1.096 s
cvCapture FromCam	T		4973	65.98 %	1.096 s
cvGetCaptureProperti	T		5131	0 %	0 us
cvGetCaptureProperti	T		5128	0 %	0 us
Spl	T		4992	0 %	0 us
FaceRecognizer Create	T		4480	31.4 %	521.5 ms
Load Haara cascade	T		3757	1.49 %	24.71 ms
Load Haara cascade	T		3767	1.12 %	18.56 ms
While	W	1001	28	0.01 %	208 us
==	T		6092	0 %	0.0999 us
Mod	T		6066	0 %	0.06793 us
+	T		5254	0 %	0.03996 us
is frame%10==0	IF		6051	0 %	0 us

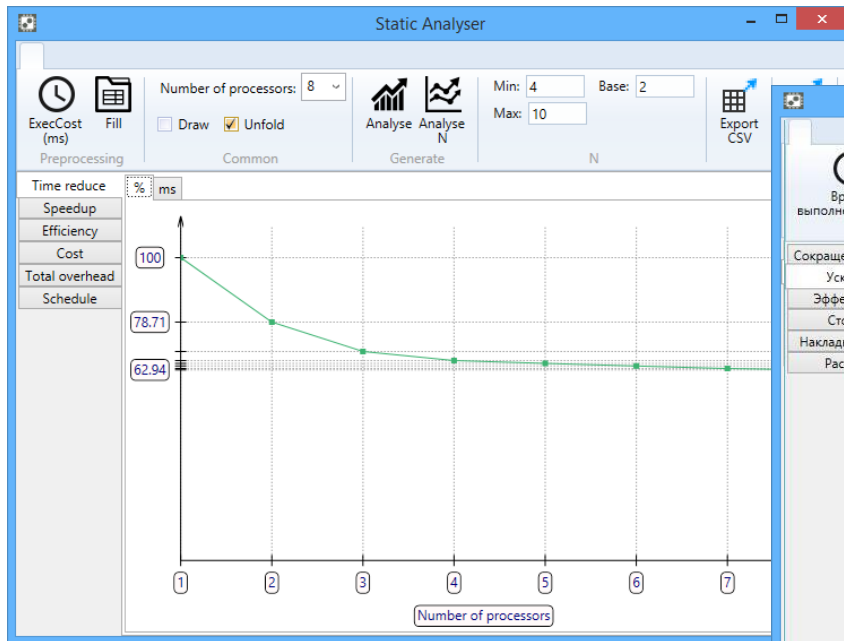
Total Time: 1.661 s

Modes :

- Absolute execution time of each node
- Relative execution time of each node
- Hot-spots

Performance evaluation. Static Analysis

Fast, early estimation of the program performance on the many core platform

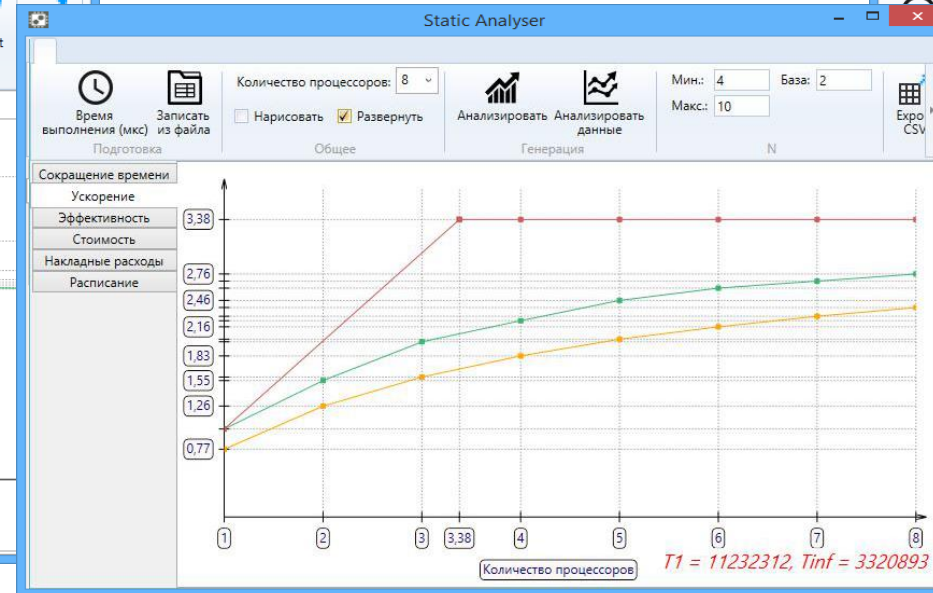


Time reduce

$$R = \frac{T_n}{T_1} * 100\%$$

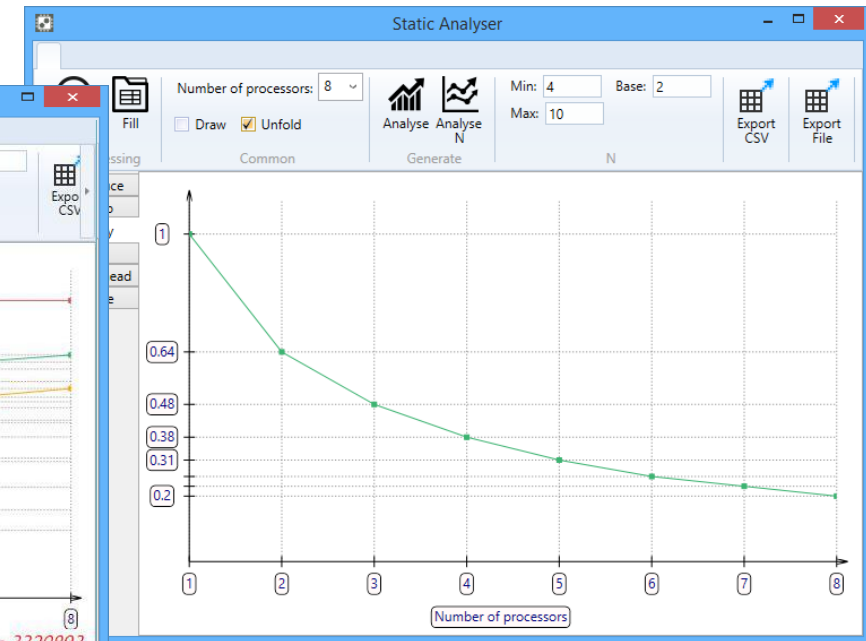
T_1 – program execution time on the 1 processor

T_n – program execution time on N processors



Speedup

$$S = \frac{T_1}{T_n}$$



Efficiency

$$E = \frac{T_1}{N * T_n}$$

N – number of processors

Performance evaluation. VPL Simulator

The screenshot displays the VPL Simulator interface. At the top, the 'Simulator console' window shows a progress bar at 1% and a simulation time of 150 us. Below the progress bar, there are tabs for 'output', 'messages', 'output_log', and 'packets'. The main console area shows a 'Message filter' set to 'Debug' and a 'Write log' checkbox. The console text shows simulation progress with timestamps and messages such as 'Simulation started...', 'Terminal i2291 (Load image (IpImage)) send, spent on transmission time: 10 ns', and 'Link i2376 received data 10 ns'. On the left side, there is a 'List of objects' panel showing a list of variables including 'i1007', 'i1017', 'i1018', 'i1057', 'i1062', 'i1066', 'i1067', 'i1077', 'i1079', 'i1113', 'i1126', 'i1159', 'i1162', and 'i1171'. A blue box highlights the 'i1017' variable. On the right side, there is a block diagram showing a 'Calc disp image' block (ID 1017) connected to a 'Disp Img' block (ID 992). The diagram also shows several input ports labeled 'int' with IDs like 2999, 1026, 3023, 3000, 3014, and 3015. Red and blue arrows point from text labels to the corresponding parts of the simulator interface.

Simulation progress status

Simulation time

List of objects

Progress in simulation

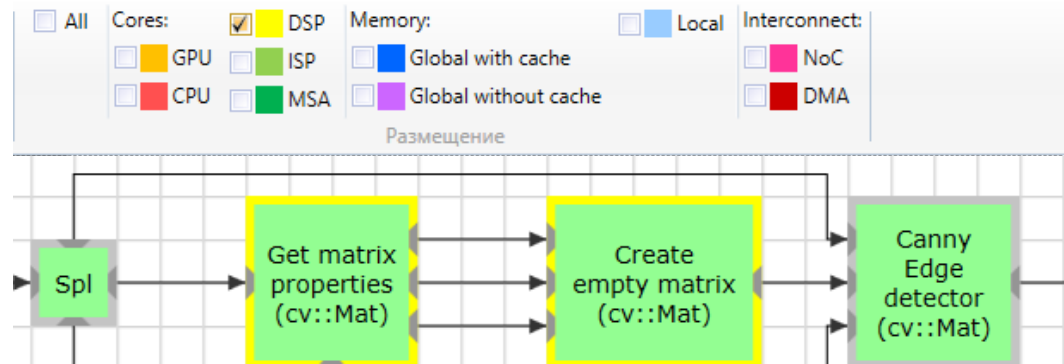
VPL program

Allows estimating :

1. Performance requirements for cores of the embedded system
2. Memory requirements
3. Load balance of various allocations
4. Volume and intensity of data exchange
5. Efficiency of hardware occupation
6. Bottlenecks of hardware platform, program and task distribution

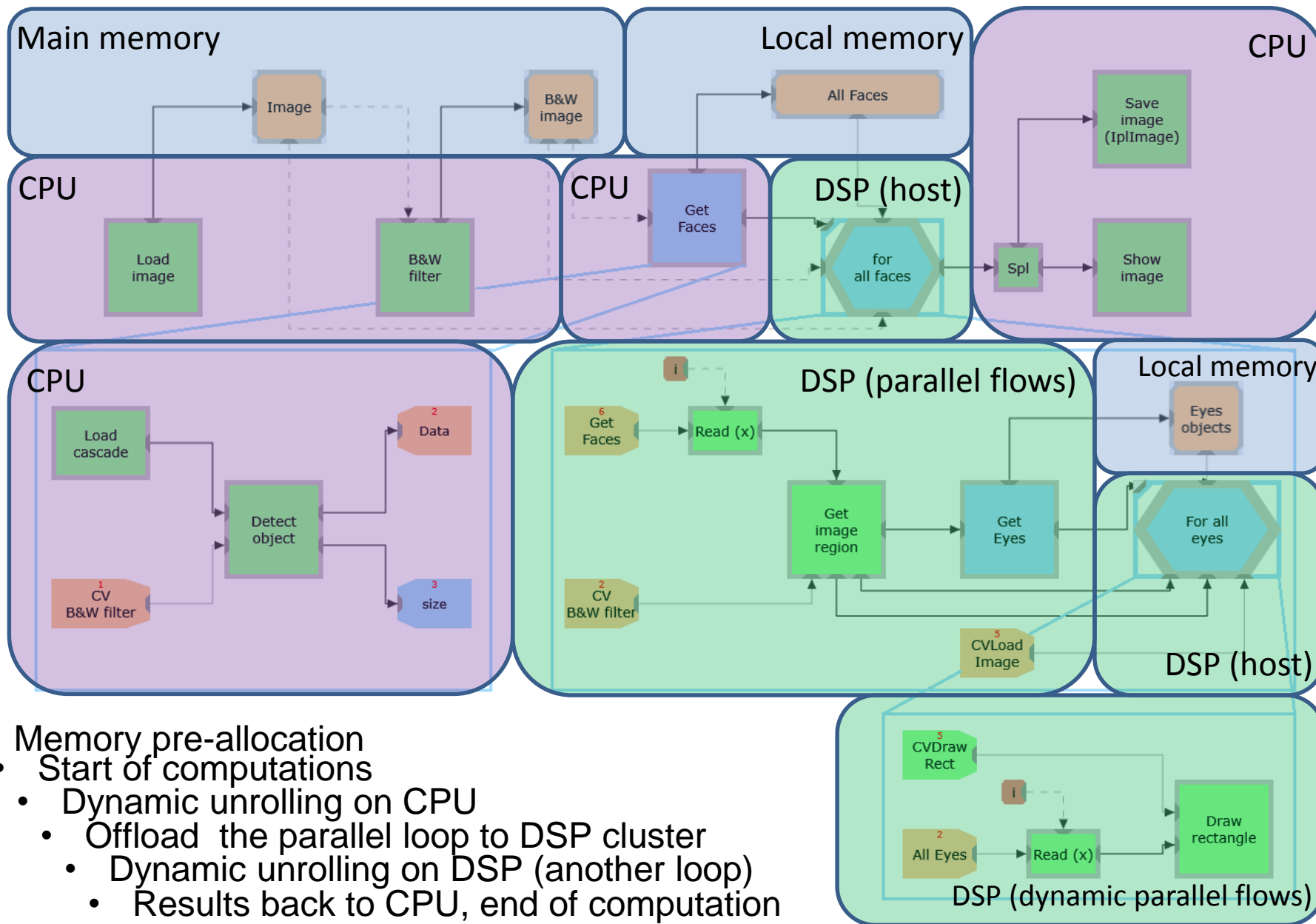
Support of heterogeneous platforms programming

Clock/Reset	MIPS-based XBurst 1.2GHz+	MIPS-based XBurst 1.2GHz+	PowerVR SGX540 GPU	HD Video CODEC
Interrupt Controller	L2 Cache (512 KB)	L2 Cache (32 KB)	OpenGL ES 1.1/2.0 and OpenGL 2.1 graphics	JPEG CODEC
Timers/PWM				
WDT				
PDMA				
SADC				
RTC				
1-wire	Audio Codec	DMA	Crypto Engine	MIPI I/f
GPS	SDRAM Controller		Boot ROM	HDMI I/f
5 x UART	DDR3/DDR2/LPDDR3/LPDDR2 PHY	NAND/eMMC/SD Card controller	Multi-display system	Camera I/f
5 x I ² C				RGB I/f
2 x SPI				LVDS
USB OTG				
3 x SDIO				



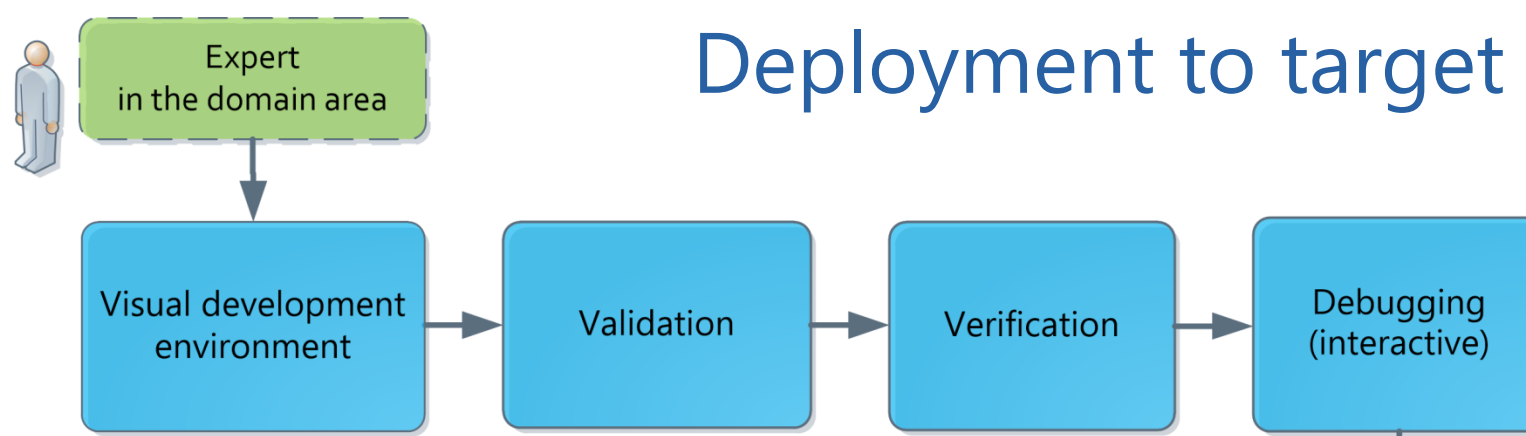
- Mapping operators to one or several core types (CPU, GPU, DSP, DMA)
 - Operators on various core types
 - Data on various data types
 - Data exchanges on various connection types
- Selecting the implementation for data processing operators
- Preparation of initial data and the results of operator of the program, taking into account the specifics of the different communication mechanisms

Heterogeneous allocation



- Memory pre-allocation
- Start of computations
 - Dynamic unrolling on CPU
 - Offload the parallel loop to DSP cluster
 - Dynamic unrolling on DSP (another loop)
 - Results back to CPU, end of computation

Deployment to target platforms

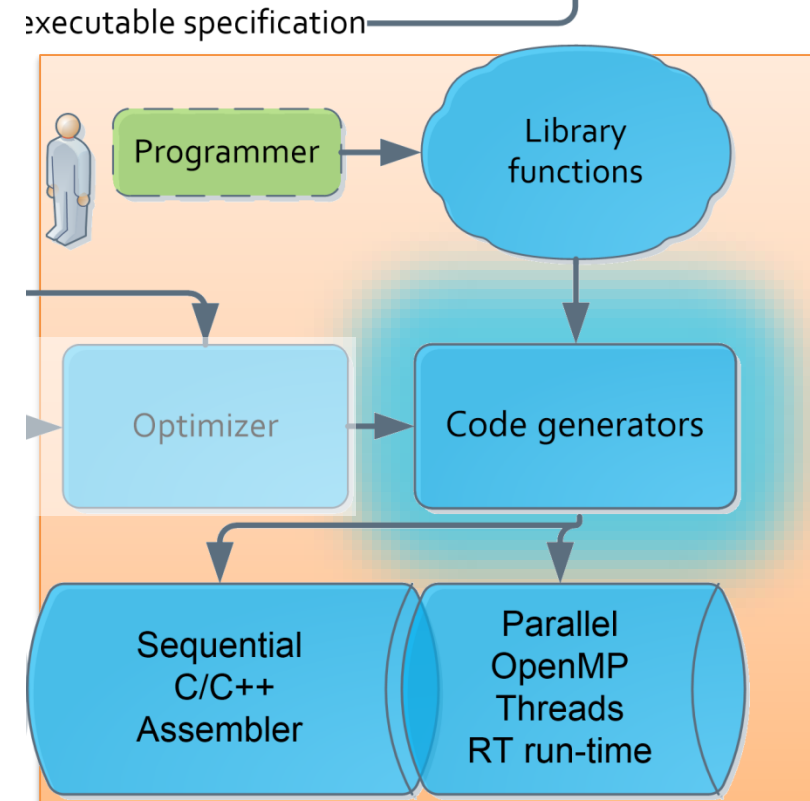


Working prototypes

- ANSI C
- C++
- RT-run-time on a multicore platform (in progress)
- Parallel OpenMP

Proof of concept

- Parallel threads
- MPI
- Assembler MIPS, DSP

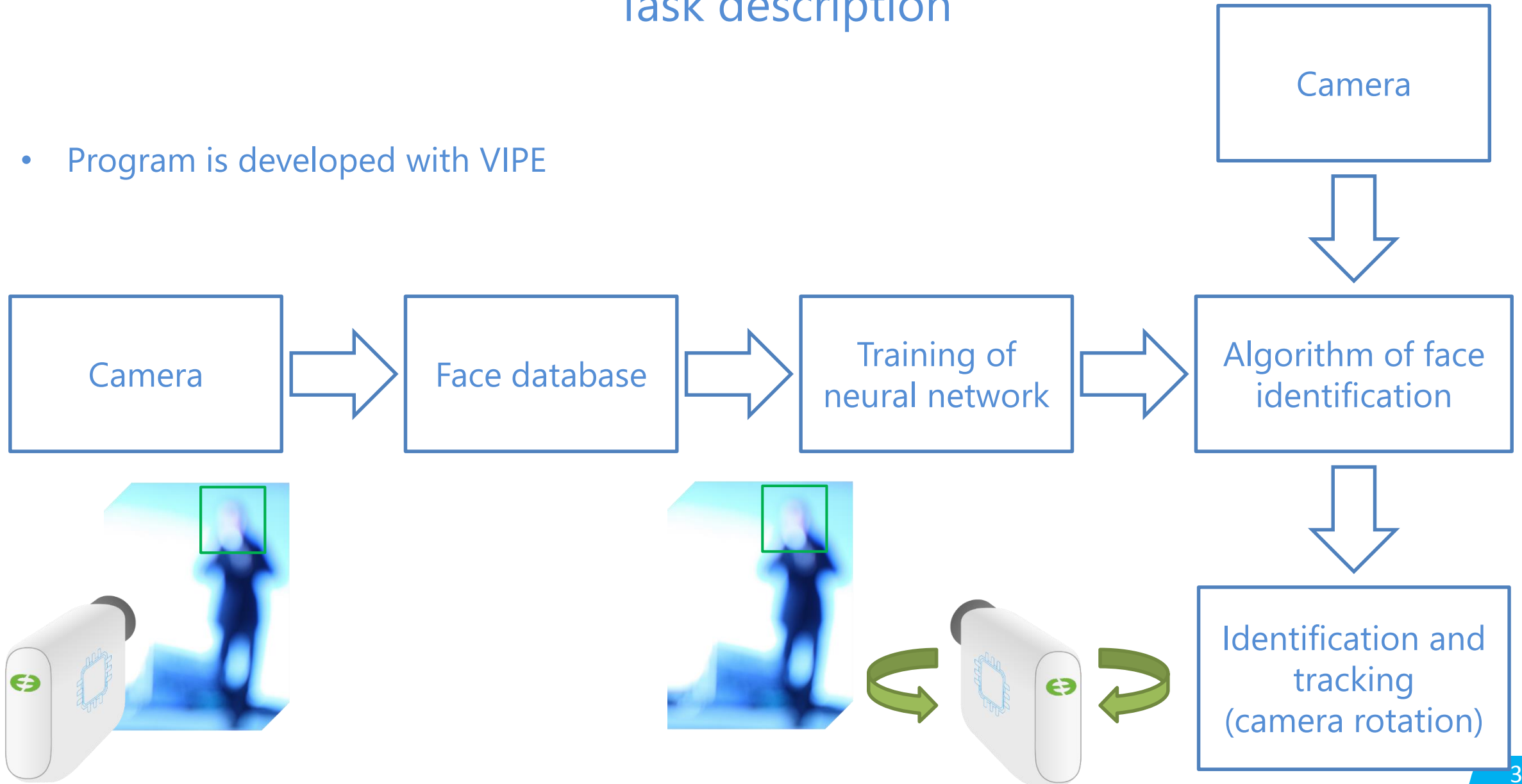


Use cases and demonstrators

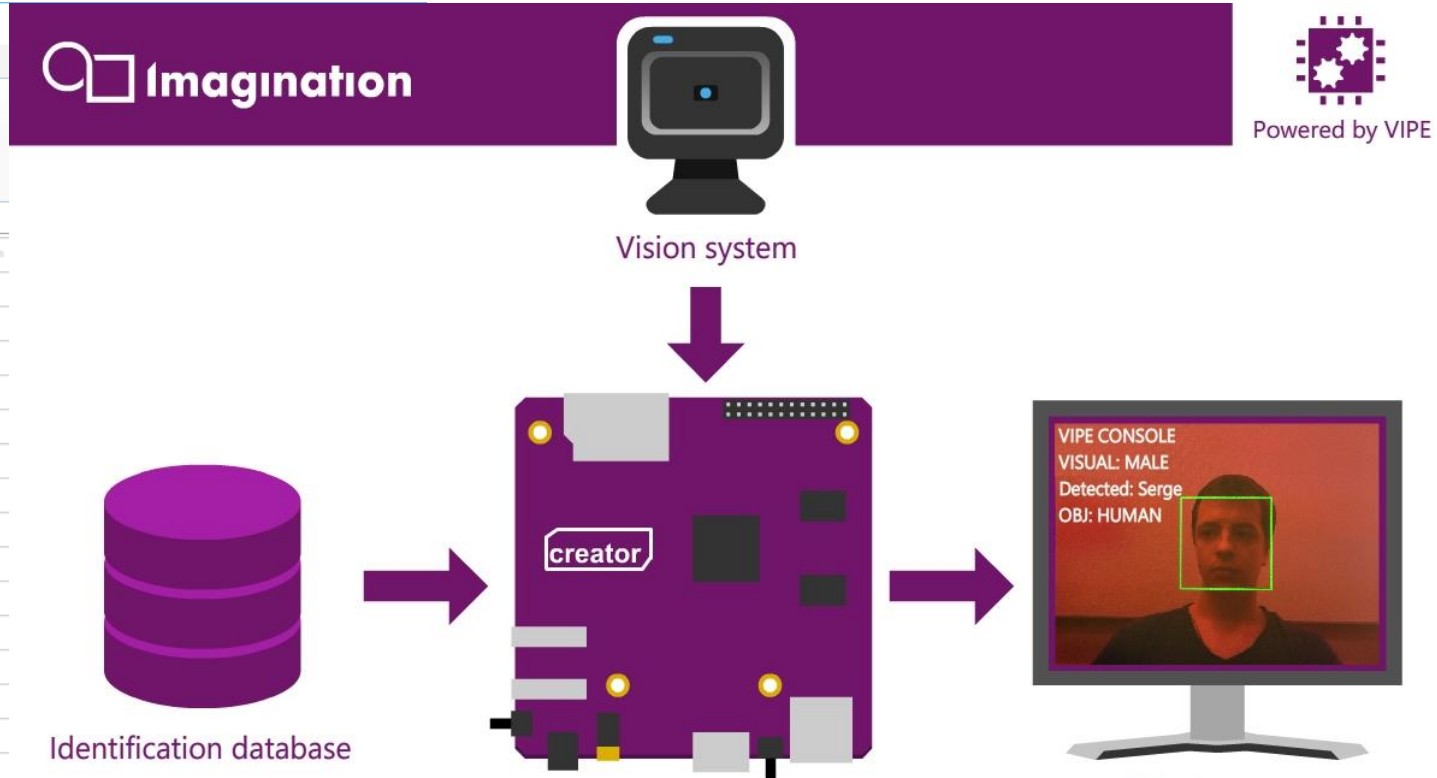
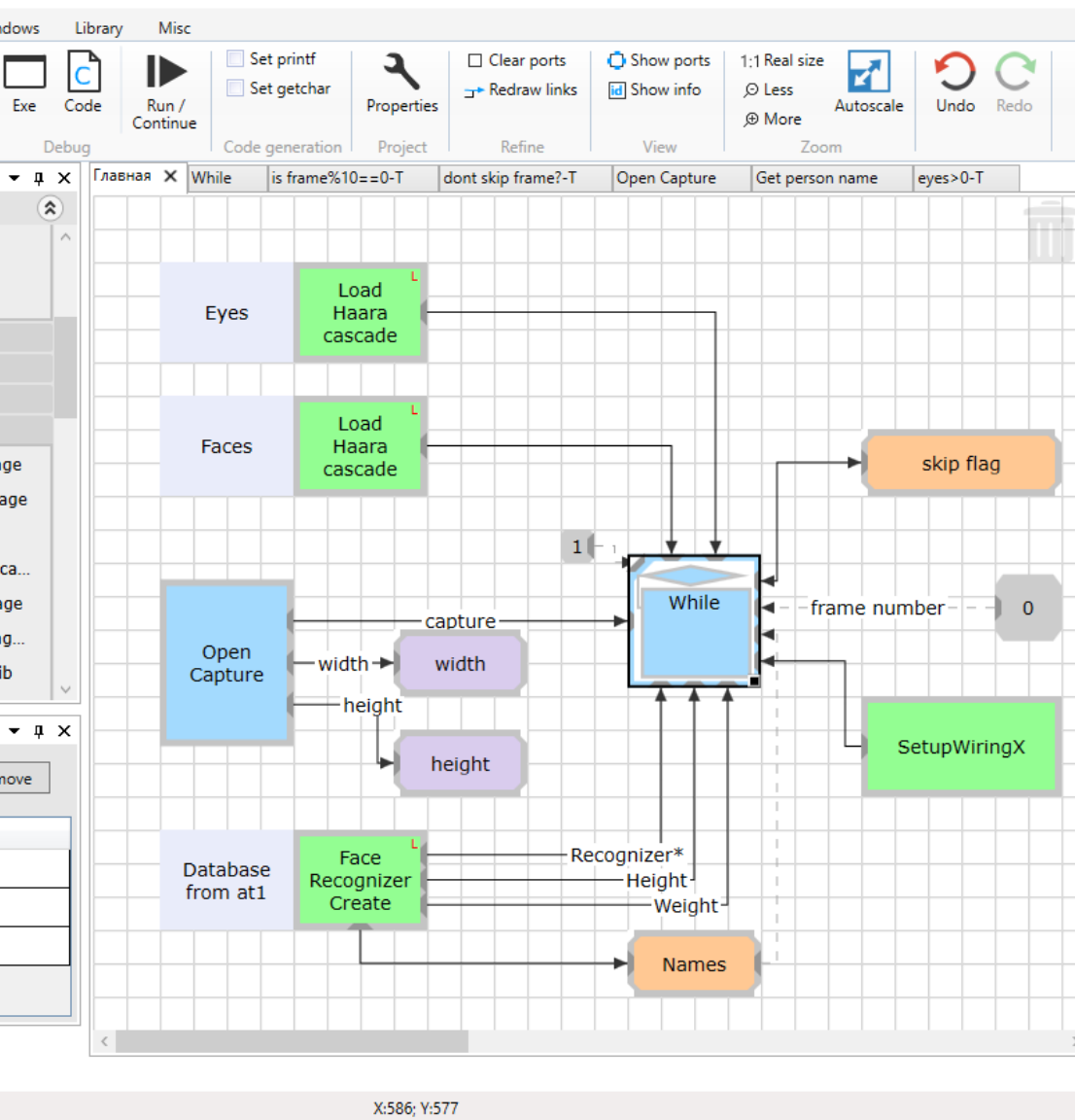
Use case: face identification

Task description

- Program is developed with VIPE



Use case: face identification



- Software part: low quality of face recognition
- Hardware part: Ci20 (perspective – ELISE by ELVEES)
- Computations only on the CPU
- Works slowly

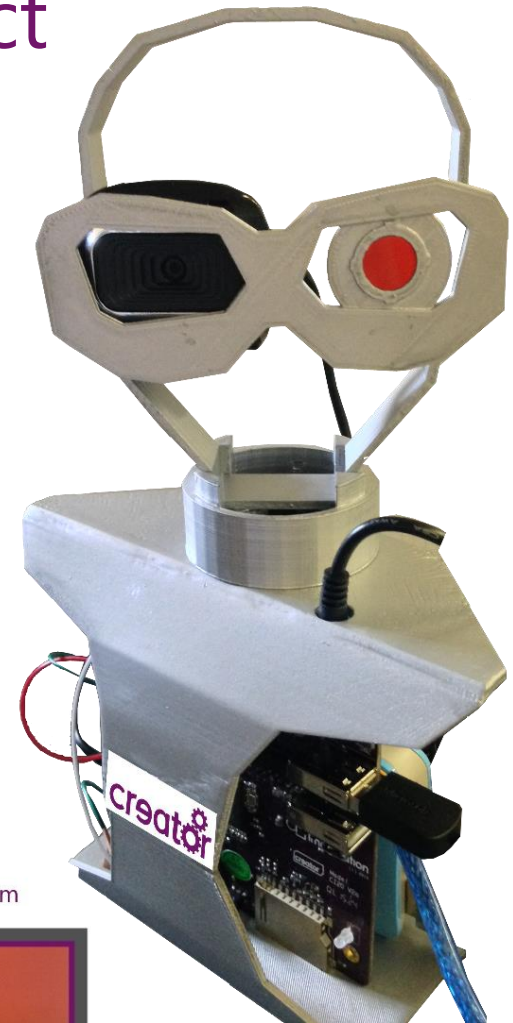
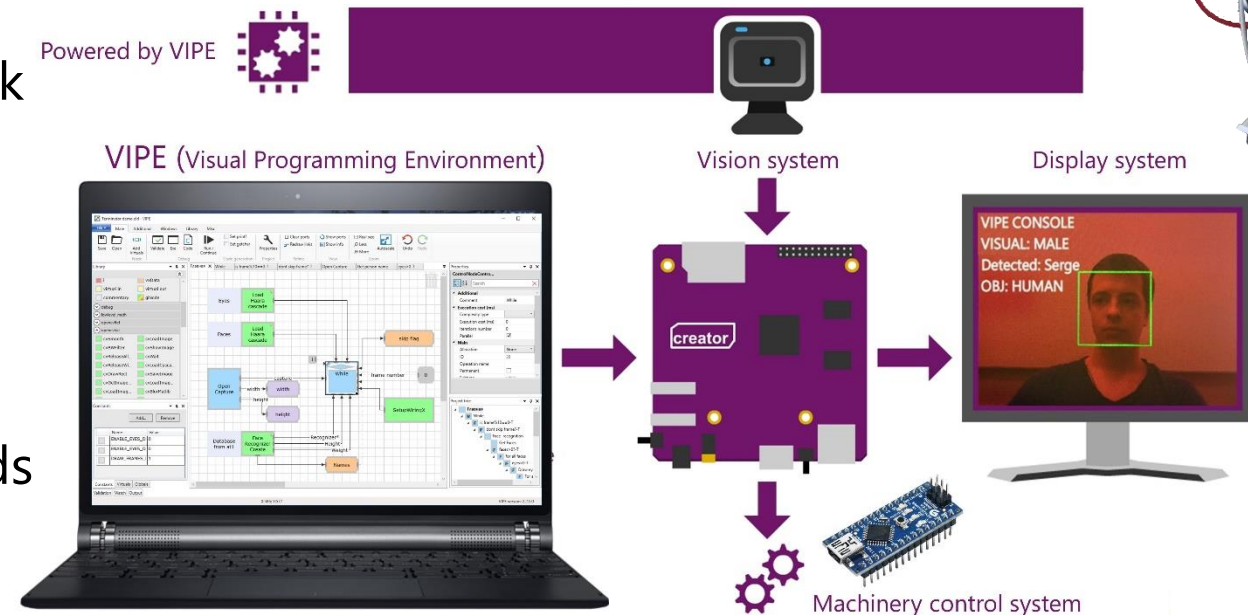
Terminator Vision System. Student project

Autonomous Cyber-Physical System combining multicore computations, control and mechanical parts. The Vision System identifies people from the database and tracks them with rotating camera.

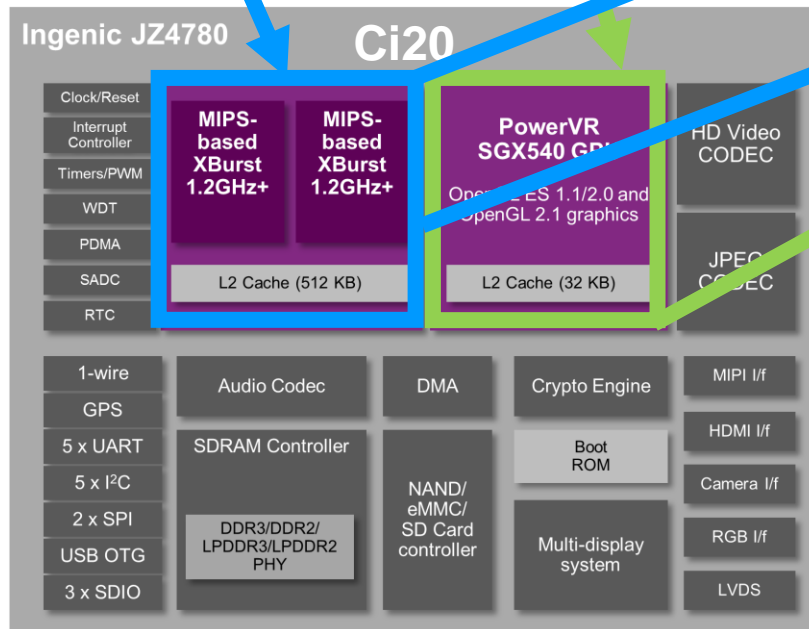
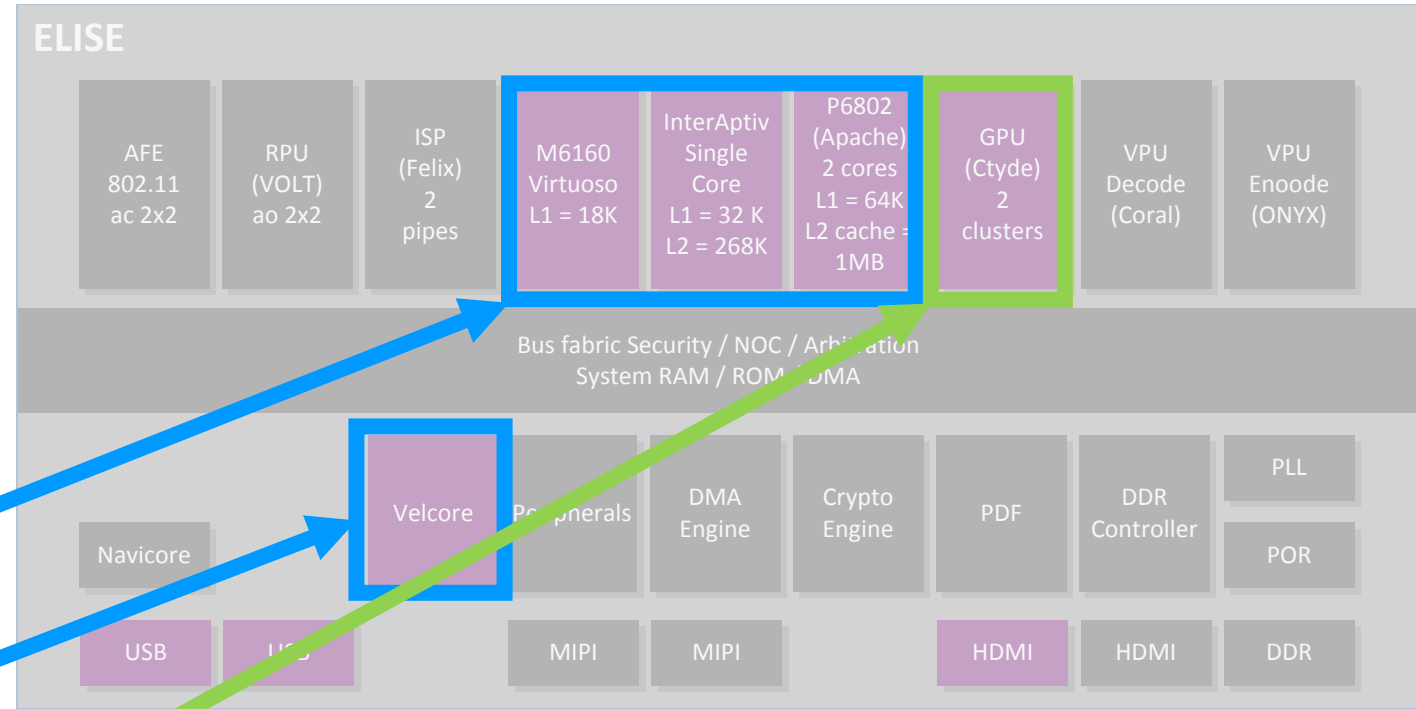
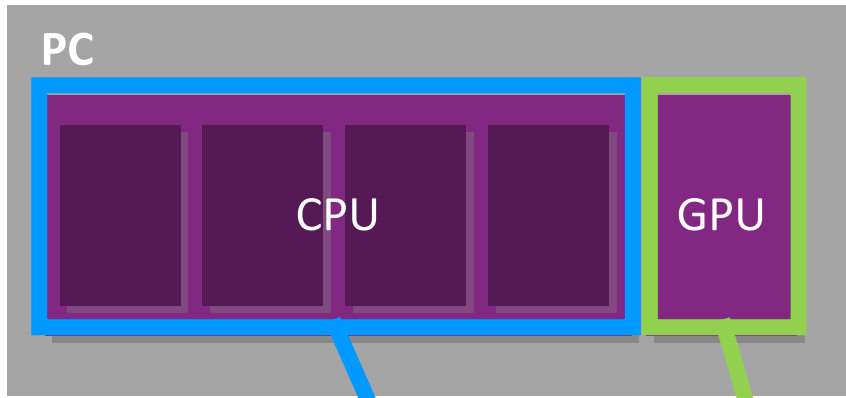
Project presented on [hackster.io](https://www.hackster.io) + **Imagination** challenge:

<https://www.hackster.io/contests/Ci20>

- Project is developed with VIPE
- Face recognition is performed by using training neural network
- Database of faces was created for face classification
- Tracking is performed by using servo, which is controlled by Arduino that receives commands from the Ci-20 board

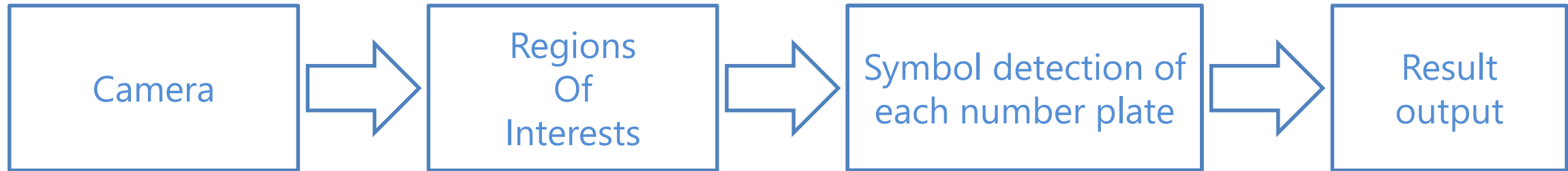


Use case: number plate recognition



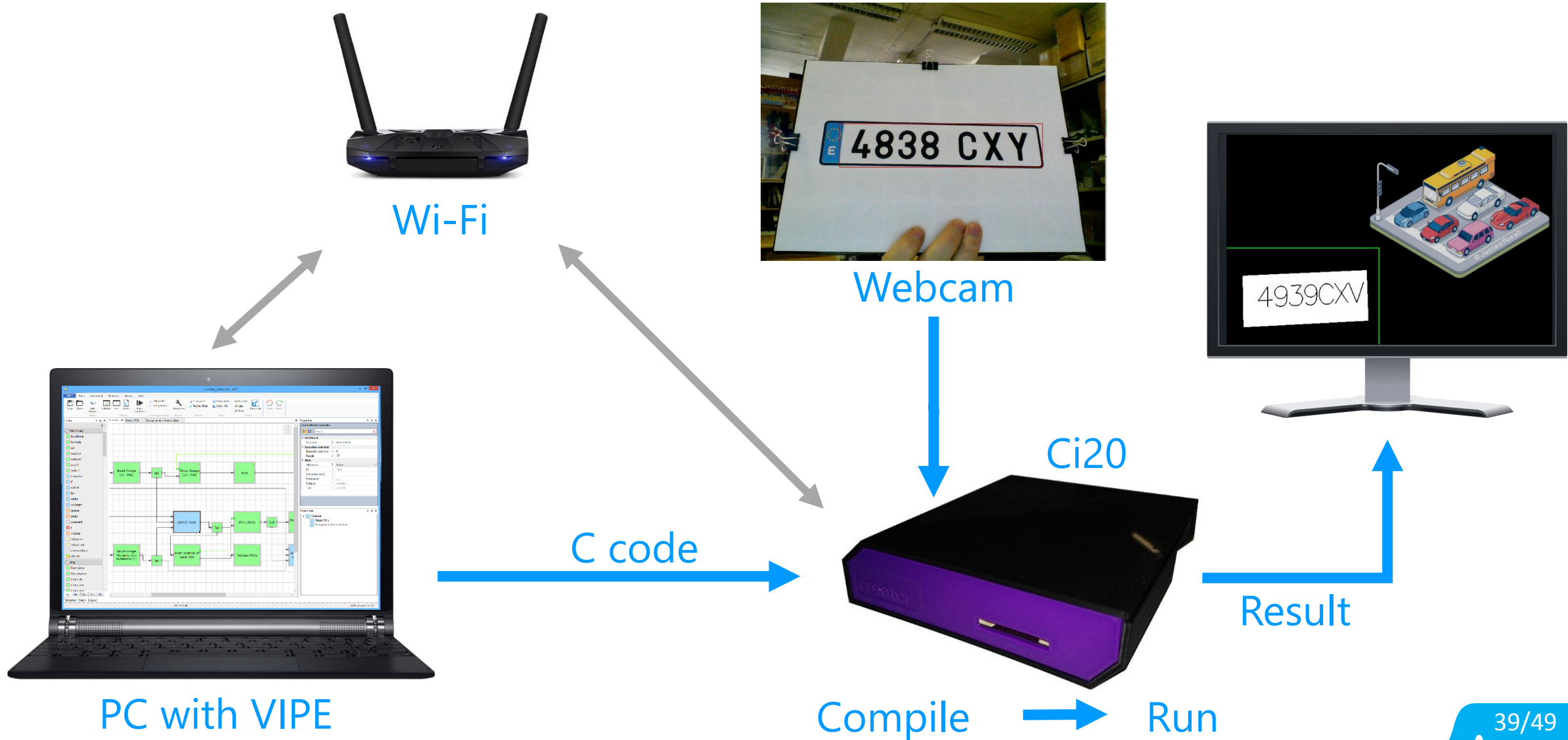
- Software part: low quality of number plate recognition
- Hardware part: Ci20 (perspective – ELISE by ELVEES)
- Computations only on the CPU
- Works slowly

Use case: number plate recognition



Use case: number plate recognition

Scheme of working with Imagination Creator Ci20 board



VIPE one button deployment

The screenshot shows the VIPE (Visual Interactive Programming Environment) interface. The main workspace contains a flowchart with the following components:

- Open Capture**: A blue node that feeds into a **While** loop.
- While**: A central blue loop node with a condition `is frame%10==0-T`. It is connected to **Load Haara cascade** nodes, a **capture** node, a **Face Recognizer Create** node, and a **Names** node.
- Face Recognizer Create**: A green node that outputs **Recognizer***, **Height**, and **Weight** to the **Names** node.
- Names**: An orange node that outputs to a **skip flag** node.
- SetupWiringX**: A green node connected to the **While** loop.
- frame number**: A grey node with value `0` connected to the **While** loop.

The **Properties** panel on the right shows the configuration for a **ControlNodeContro...** node:

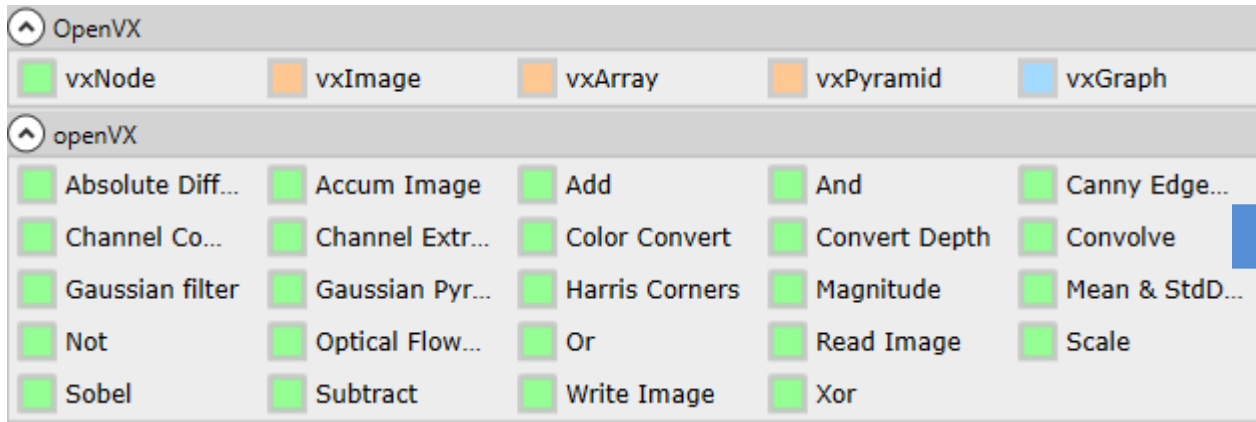
- Additional**: Comment: While
- Execution cost (ms)**: Complexity type: [dropdown], Execution cost (ms): 0, Iterations number: 0, Parallel:
- Main**: Allocation: None, ID: 28, Operation name: [empty], Permanent:

The **Project tree** on the bottom right shows the project structure:

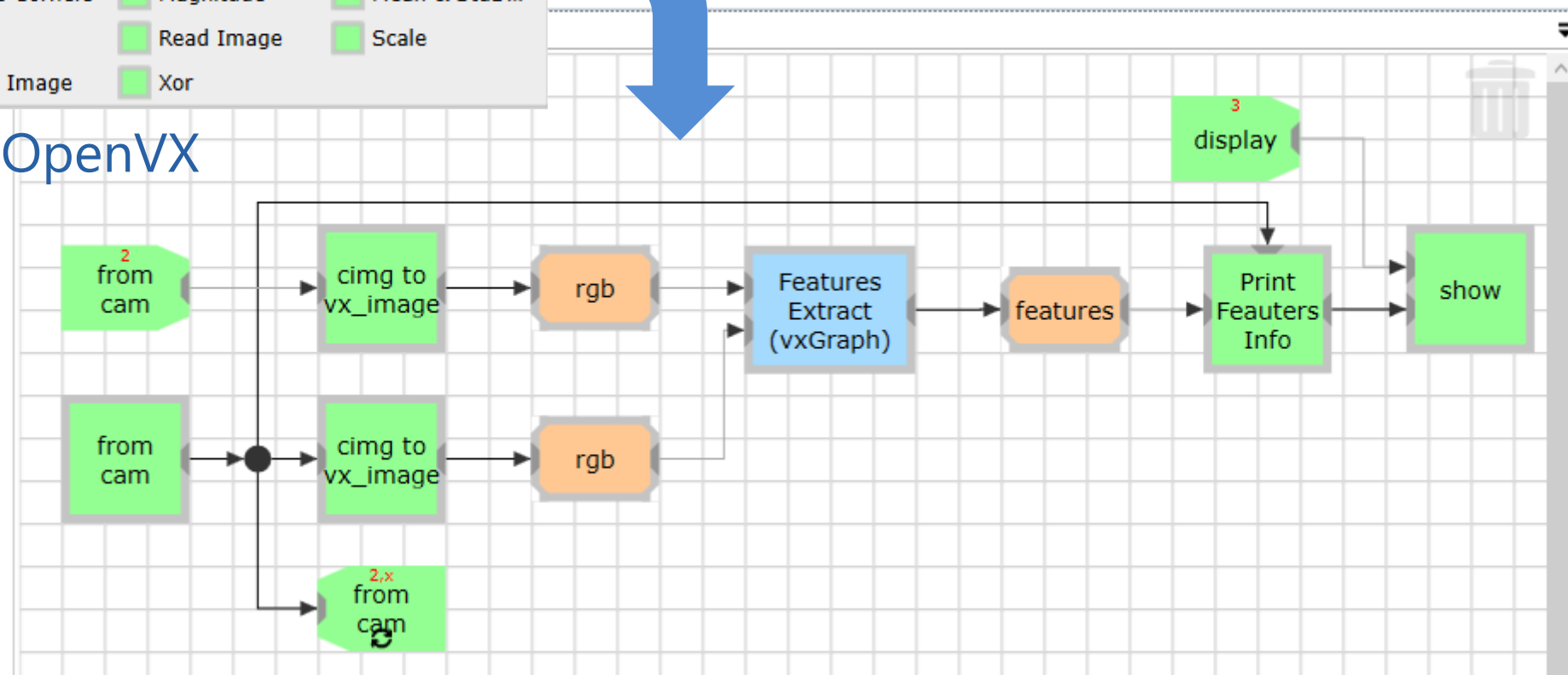
- Главная
 - While
 - IF is frame%10==0-T
 - IF dont skip frame?-T
 - Face recognition
 - IF faces>0?-T
 - F for all faces
 - IF eyes>0-T
 - IF Draw ey
 - F For a

Feature tracking (OpenVX)

DSL and design

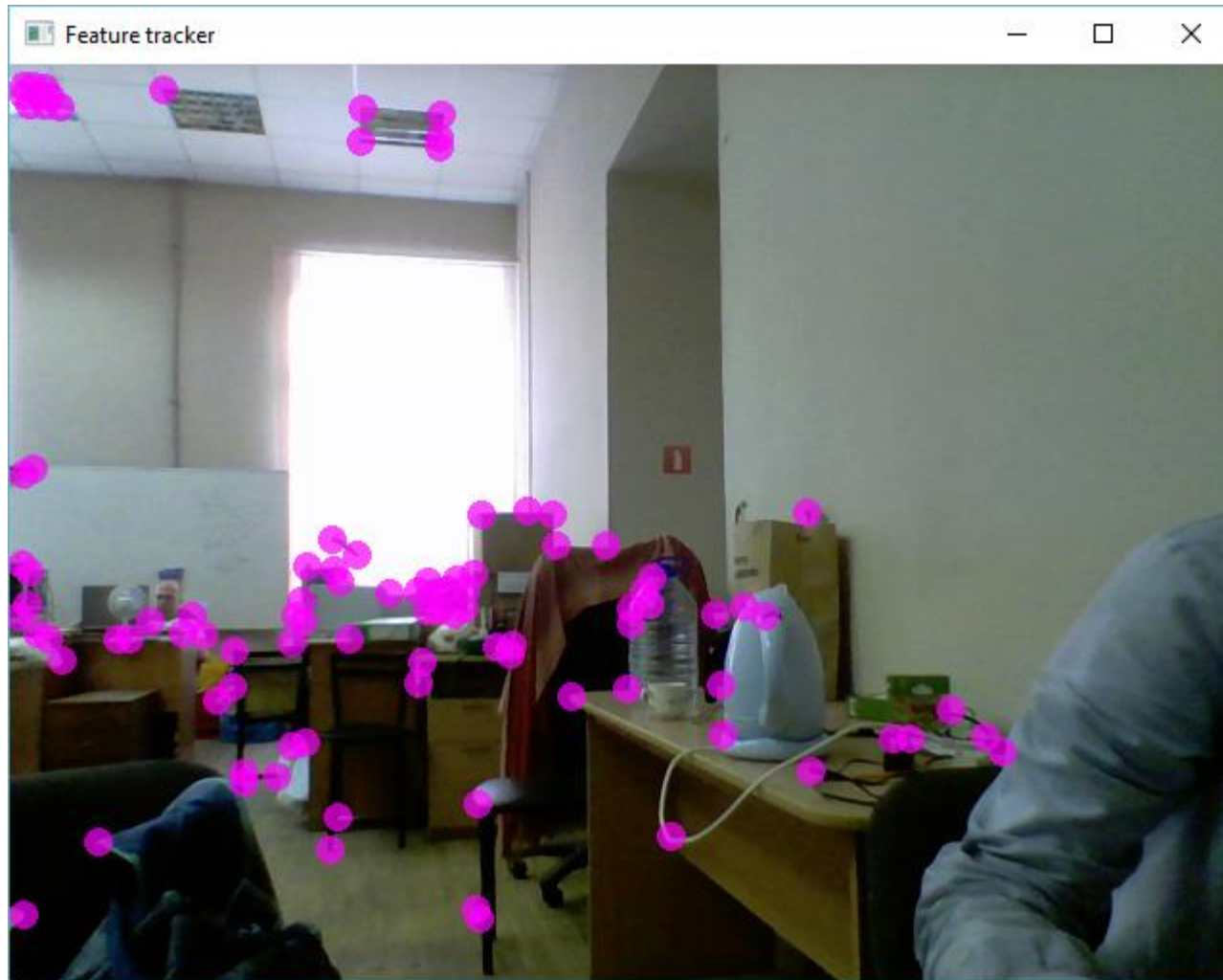


DSL, based on OpenVX

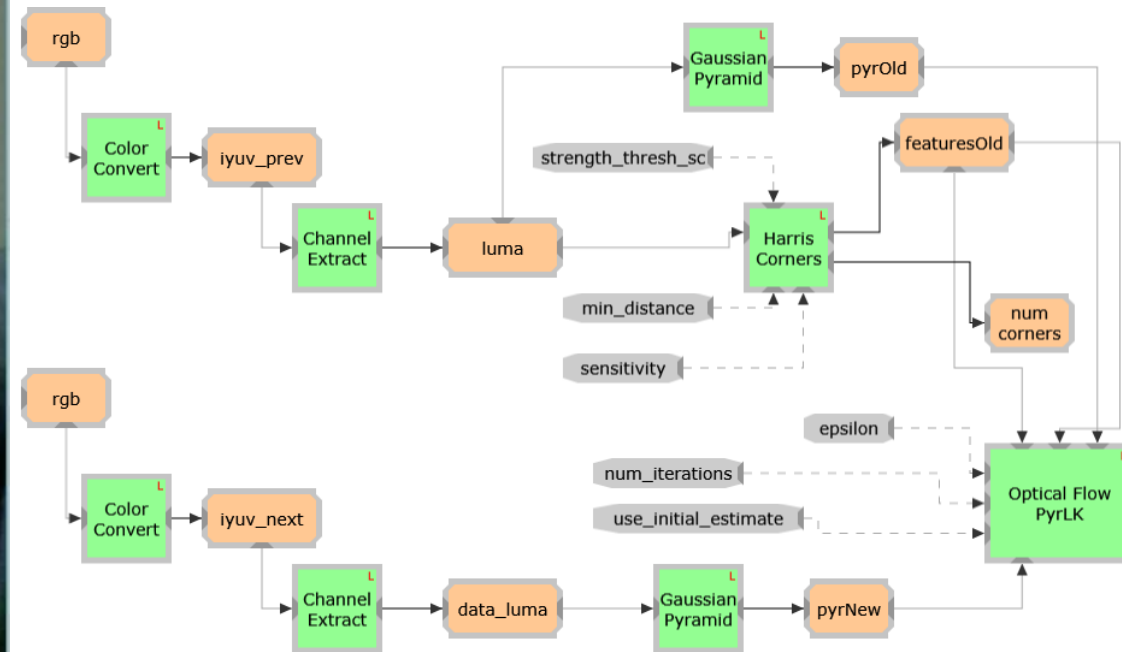


Feature tracking program in VIPE

Feature tracking (OpenVX) Results

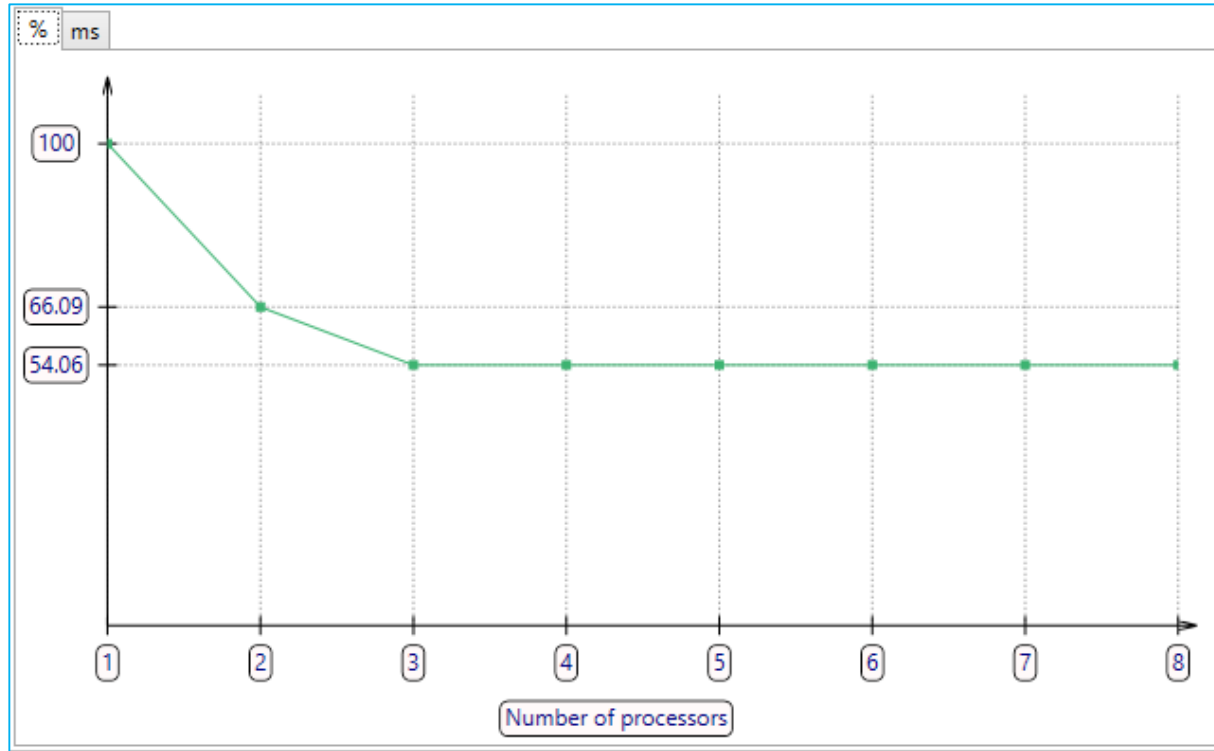


Feature tracking program run on the x86 platform with using the sample implementation by Khronos

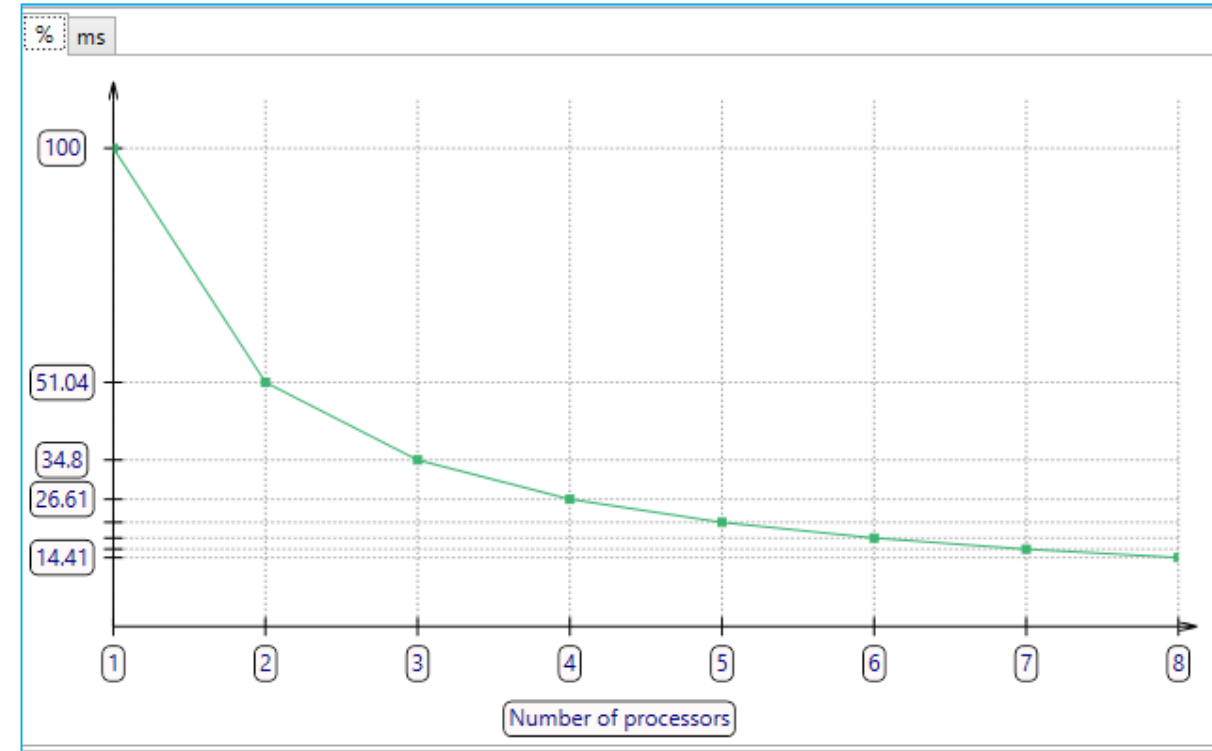


Feature tracking

Static analysis



Performance estimation of the feature tracking program with **sequential** frame processing



Performance estimation of the feature tracking program with **parallel** frame processing

Feature tracking Visual Profiler

Name	Subtype	Avg. iter. c	ID	%	Total Time
while	w	50	9277	98.68 %	35.99 s
Harris Corners	T		9326	22.39 %	8.167 s
cing to vx_image	T		9413	16.02 %	5.844 s
cing to vx_image	T		9423	15.96 %	5.821 s
Gaussian Pyramid	T		9370	12.4 %	4.522 s
Gaussian Pyramid	T		9321	12.24 %	4.464 s
Optical Flow PyrLK	T		9381	9.55 %	3.477 s
Color Convert	T		9311	2.31 %	843.6 ms
Color Convert	T		9360	2.26 %	825.6 ms
	T		9426	1.05 %	383.2 ms
Channel Extract	T		9316	1.01 %	368.2 ms
Channel Extract	T		9365	1 %	366.3 ms
from cam	T		9480	0.99 %	360.1 ms
show	T		9519	0.88 %	321.2 ms
PrintFeautersInfo	T		9402	0.63 %	231 ms
from cam	T		9484	1.27 %	464.9 ms
create display	T		9520	0.04 %	15.45 ms

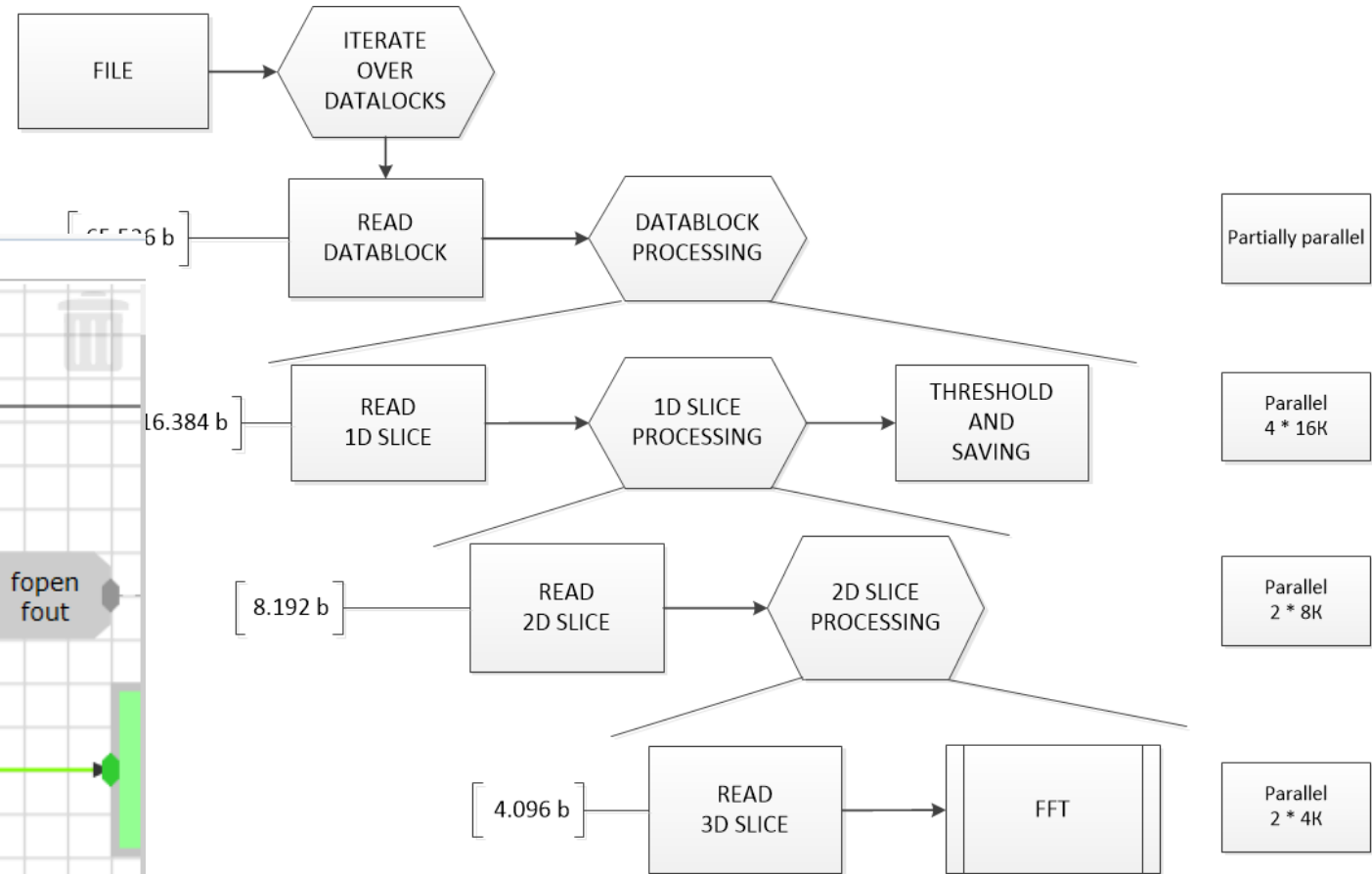
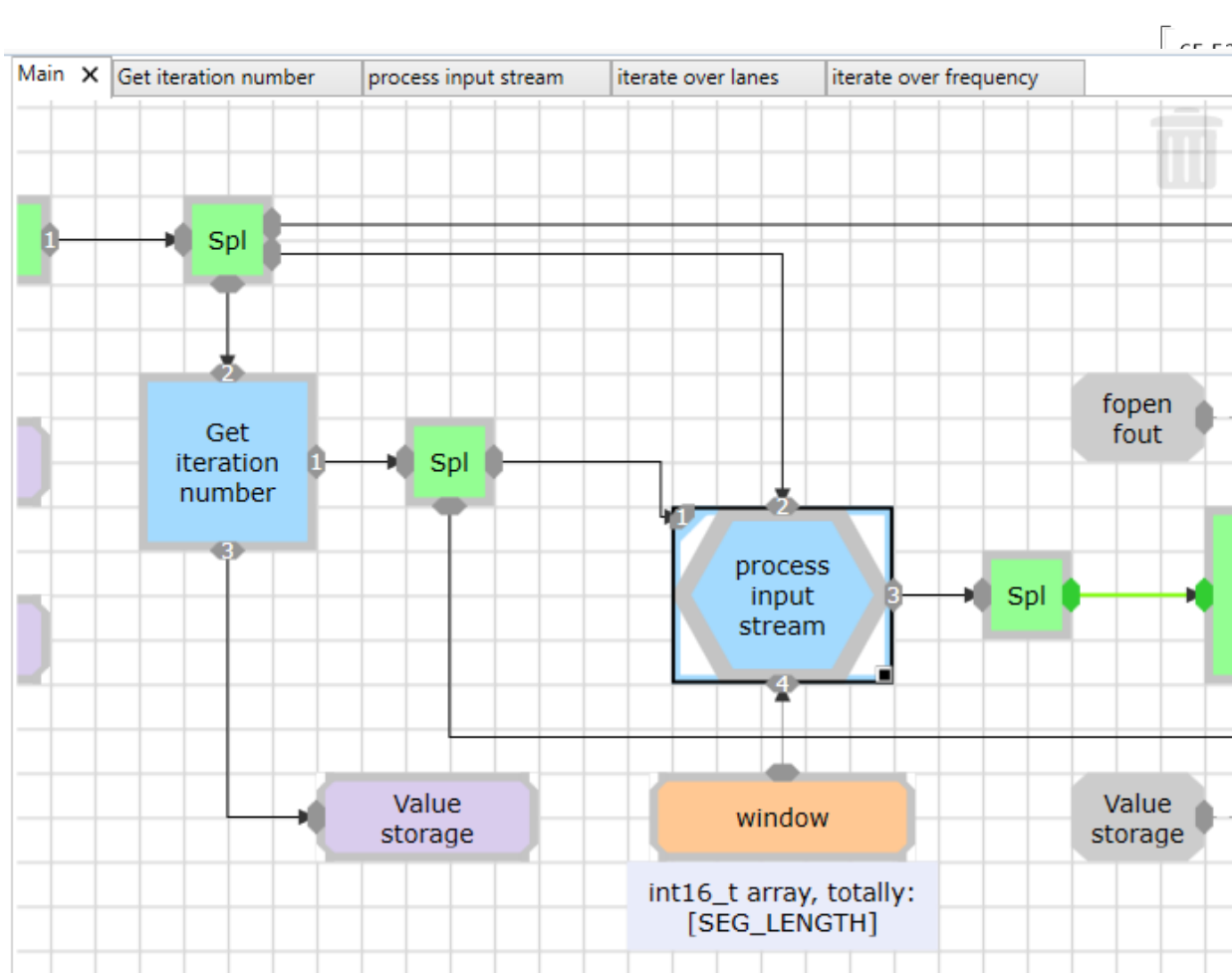
Total Time: 36.47 s

Large amount of time is taken by image format conversion function (from OpenVX format and back)

Profiling of the feature tracking program

Traffic radar object detection

Development

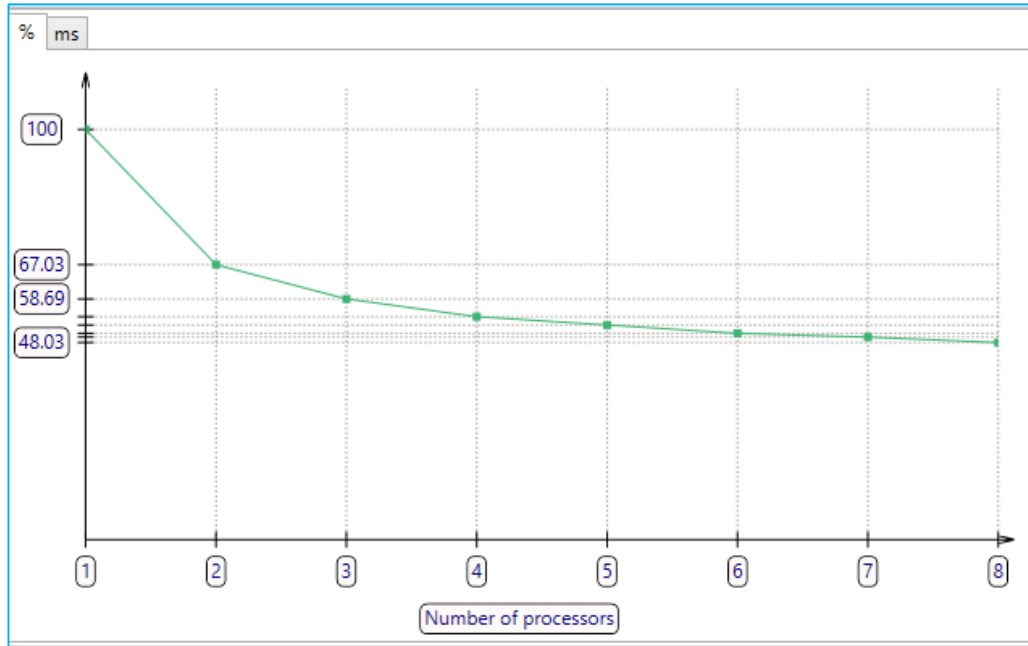


Traffic radar object detection is developed in the PaPP project, part of the European technical platform ARTEMIS/ECSEL.

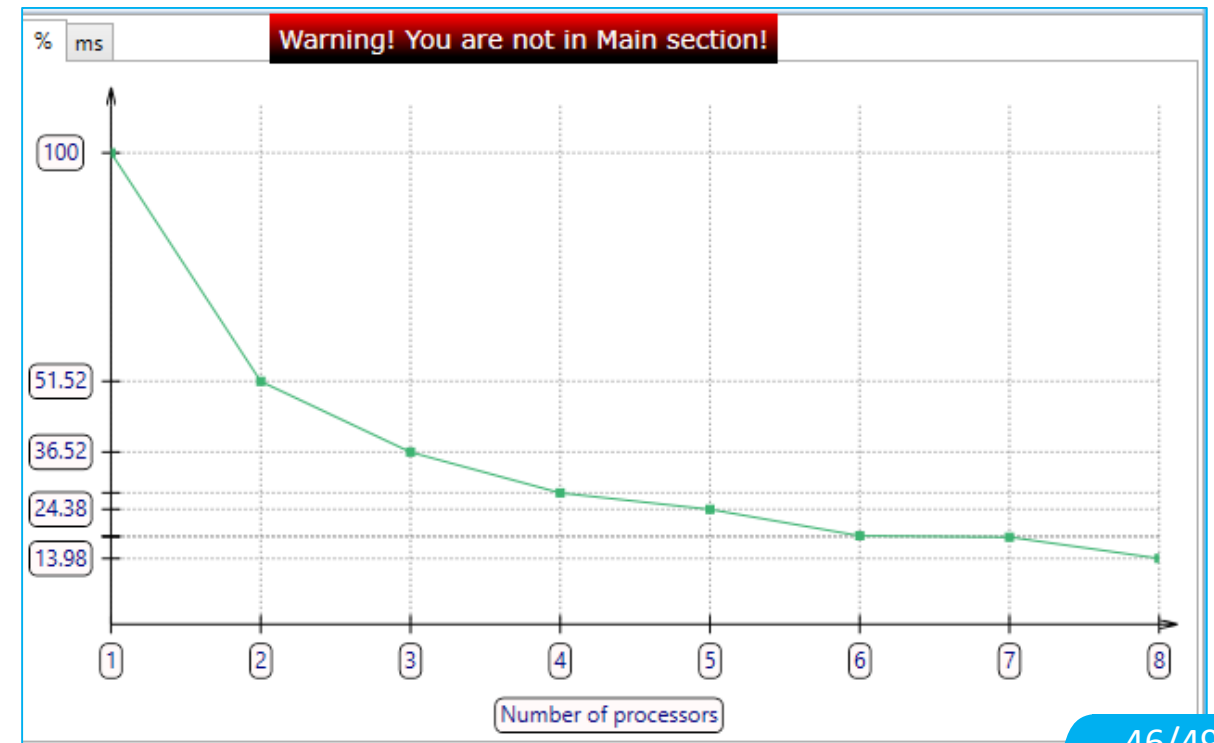
Traffic radar object detection

Static analysis

Static analysis shows acceptable reduction of time on 2-3 cores

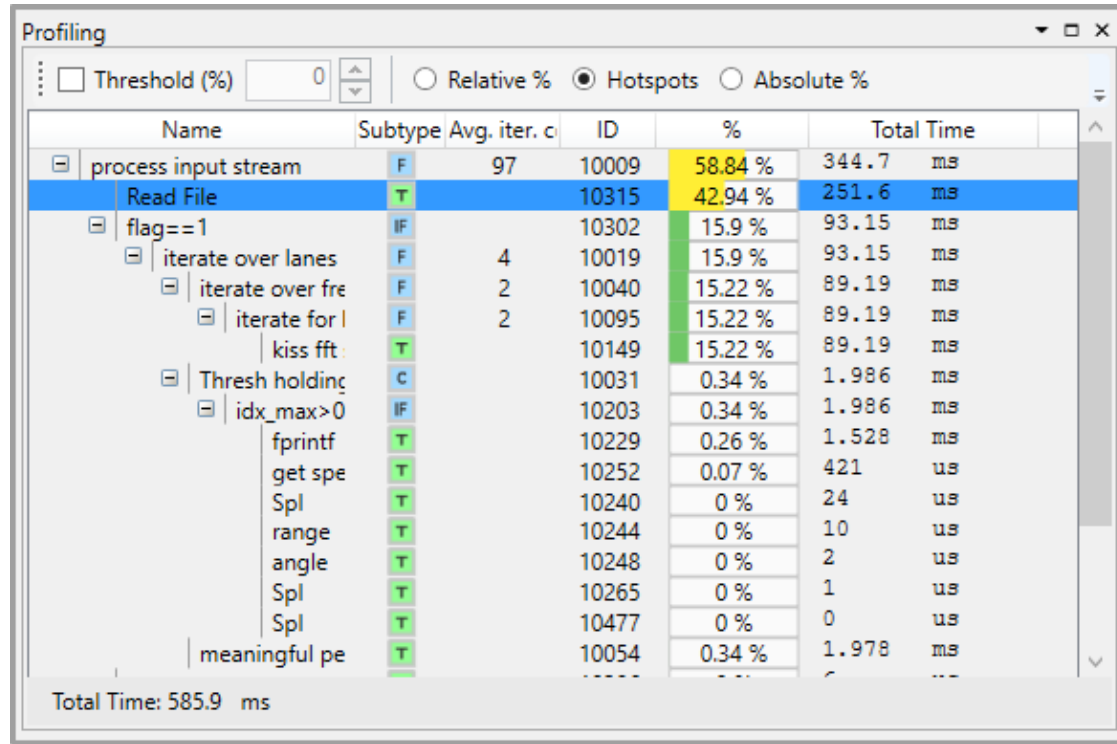


However, the results were worse than expected. Static analysis of subprogram "Data processing units" shows close to a linear reduction of time for 8 cores



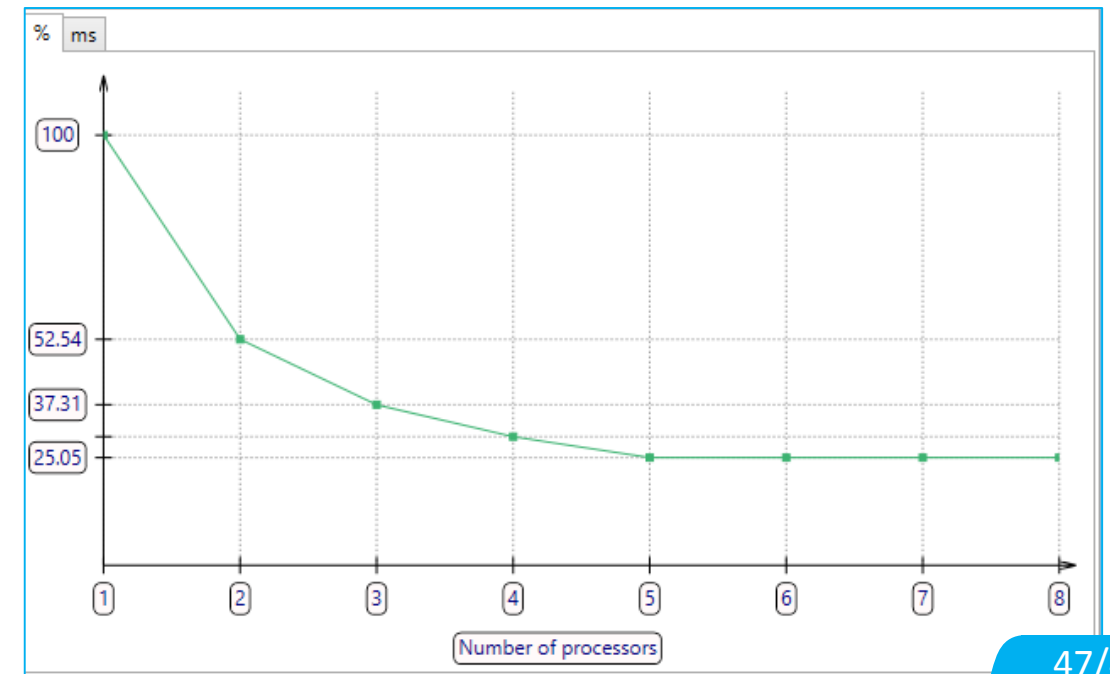
Traffic radar object detection

Visual Profiler



Visual Profiler shows, that a large amount of time is taken by function for reading the input file (prototype uses data from the input file).

File reading function is in sequential part, hence the parallelism is limited by Amdahl's law. Actual process of getting the input data should be optimized to take less time. Evaluation of program with reduced operating time of reading function shows satisfactory results.



Traffic radar object detection

Comparison of the results of analysis, simulation and execution

Cores (simulator) or threads(OpenMP)	Static analysis sec. / %	Modeling VPL sec. / %	Execution sec. / %
Without OpenMP			1.60
1	1.26 / 100	1.29 / 100	1.65 / 100
2	0.64 / 50.8	0.88 / 68.2	1.00 / 60.6
3	0.60 / 47.6	0.72 / 55.8	0.81 / 49.1
4	0.34 / 30.0	0.55 / 42.6	1.25 / 76.7

Hardware platform

- Core i7 8 cores-> VirtualBox VM 4 cores
- Ubuntu 14.04, GCC 4.9.2.

Input data

- 12 MB signal samples

Summary

- Technology covers various requirements of embedded SW development
 - Design, programming, evaluation, porting etc.
- DSLs for involving domain specialists into development process
- Rapid SW prototyping for early customer presentations
- Formal model basis for proofed and predictable results, including debugging
- Fast tools adaptation for new cores and platforms

- Supporting development tool for heterogeneous cores, platforms, system software infrastructure