

K H R O N O STM
G R O U P

OpenVXTM

An open, royalty-free standard for cross platform acceleration
of computer vision and neural network applications.

OpenVX

Wide range of vision hardware architectures
OpenVX provides a high-level Graph-based abstraction

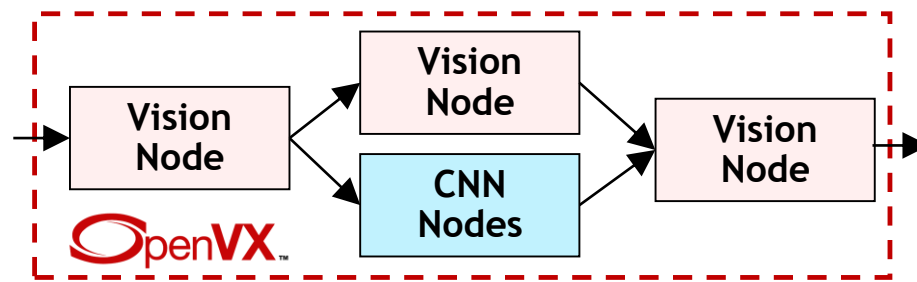
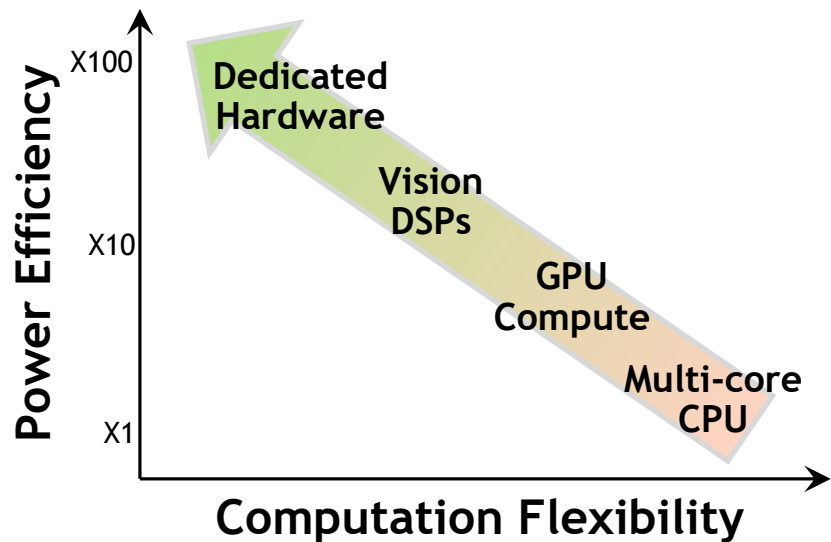
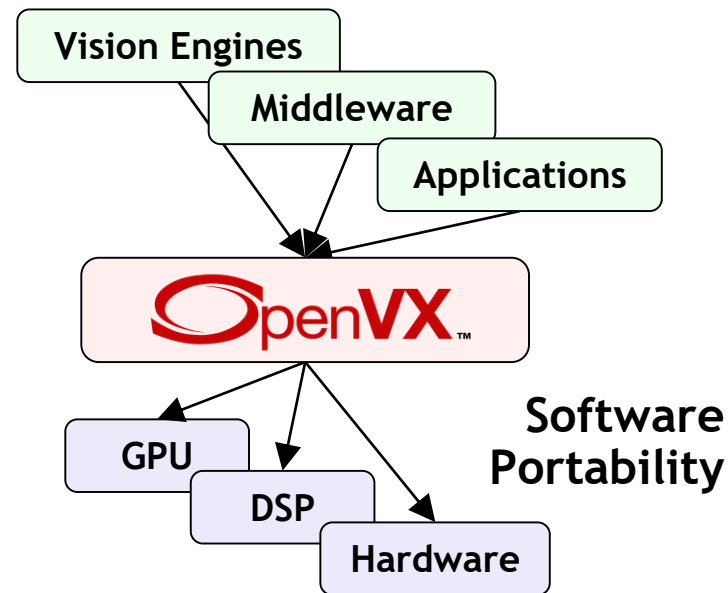
->

Enables Graph-level optimizations!

Can be implemented on almost any hardware or processor!

->

Portable, Efficient Vision Processing!



Vision Processing Graph

OpenVX Evolution



Conformant Implementations

AMD

cadence

CEVA



socionext

SYNOPSYS



AMD OpenVX Tools

- Open source, highly optimized for x86 CPU and OpenCL for GPU
 - "Graph Optimizer" looks at entire processing pipeline and removes, replaces, merges functions to improve performance and bandwidth
 - Scripting for rapid prototyping
 - live 360° camera stitching
 - neural networks (develop)
- bit.ly/openvx-amd

OpenVX 1.0 and OpenVX 1.1

New Functionality

- Conditional node execution
- Feature detection
- Classification operators
- Expanded imaging operations

Extensions

- Neural Network Acceleration
- Graph Save and Restore
- 16-bit image operation

Safety Critical

- OpenVX 1.1 SC for safety-certifiable systems

New Functionality Under Discussion

Neural Network Import for Inference (NNEF)

Programmable user kernels with accelerator offload

Streaming/pipelining

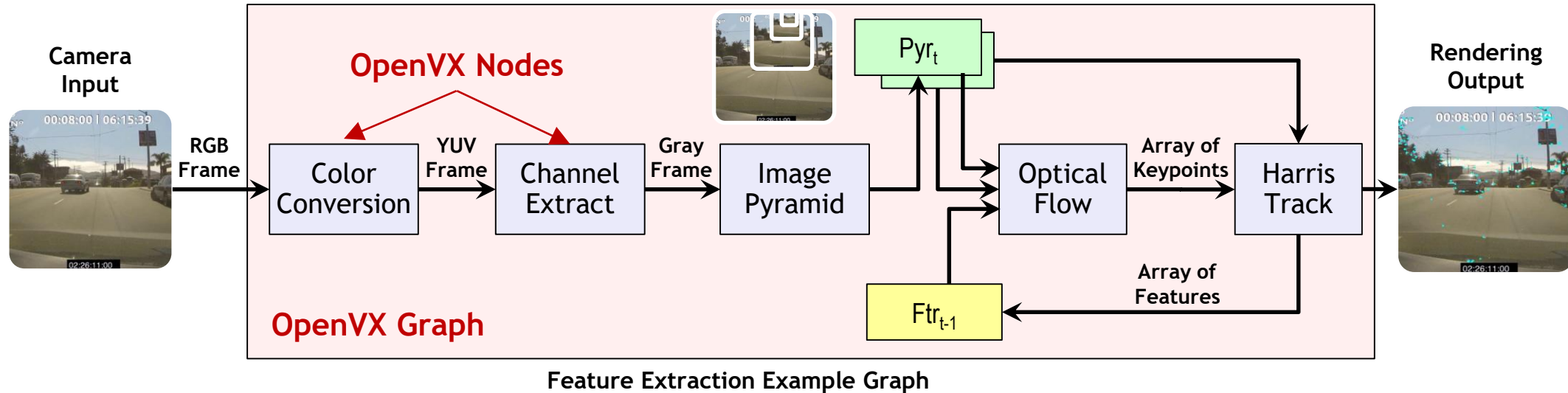
OpenVX Roadmap

OpenVX 1.2

Spec released May 2017

OpenVX - Graph-Level Abstraction

- OpenVX developers express a graph of image operations ('Nodes')
 - Using a C API
- Nodes can be executed on any hardware or processor coded in any language
 - Implementers can optimize under the high-level graph abstraction
- Graphs are the key to run-time power and performance optimizations
 - E.g. Node fusion, tiled graph processing for cache efficiency etc.



OpenVX Efficiency through Graphs..

Graph Scheduling

Split the graph execution across the whole system: CPU / GPU / dedicated HW

Faster execution or lower power consumption

Memory Management

Reuse pre-allocated memory for multiple intermediate data

Less allocation overhead, more memory for other applications

Kernel Fusion

Replace a sub-graph with a single faster node

Better memory locality, less kernel launch overhead

Data Tiling

Execute a sub-graph at tile granularity instead of image granularity

Better use of data cache and local memory

Simple Edge Detector in OpenVX

```
vx_graph g = vxCreateGraph();
```

```
vx_image input = vxCreateImage(1920, 1080);
```

Declare Input and Output Images

```
vx_image output = vxCreateImage(1920, 1080);
```

```
vx_image horiz = vxCreateVirtualImage(g);
```

Declare Intermediate Images

```
vx_image vert = vxCreateVirtualImage(g);
```

```
vx_image mag = vxCreateVirtualImage(g);
```

```
vxSobel3x3Node(g, input, horiz, vert);
```

Construct the Graph topology

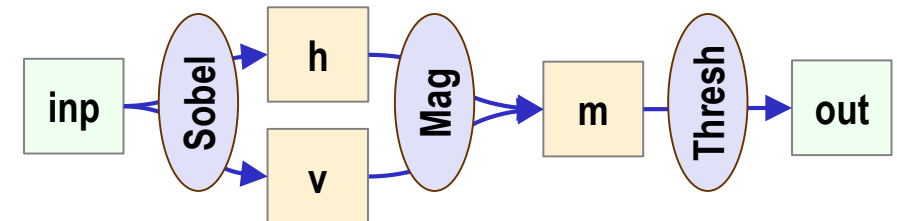
```
vxMagnitudeNode(g, horiz, vert, mag);
```

```
vxThresholdNode(g, mag, THRESH, output);
```

Compile the Graph
Execute the Graph

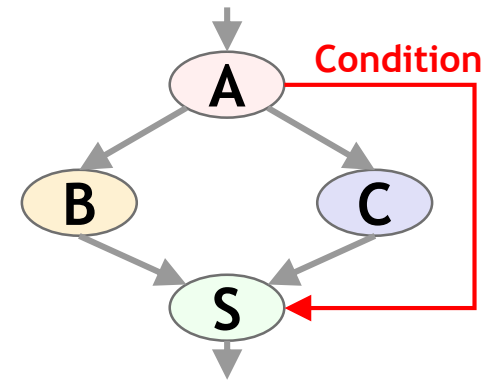
```
status = vxVerifyGraph(g);
```

```
status = vxProcessGraph(g);
```



New OpenVX 1.2 Functions

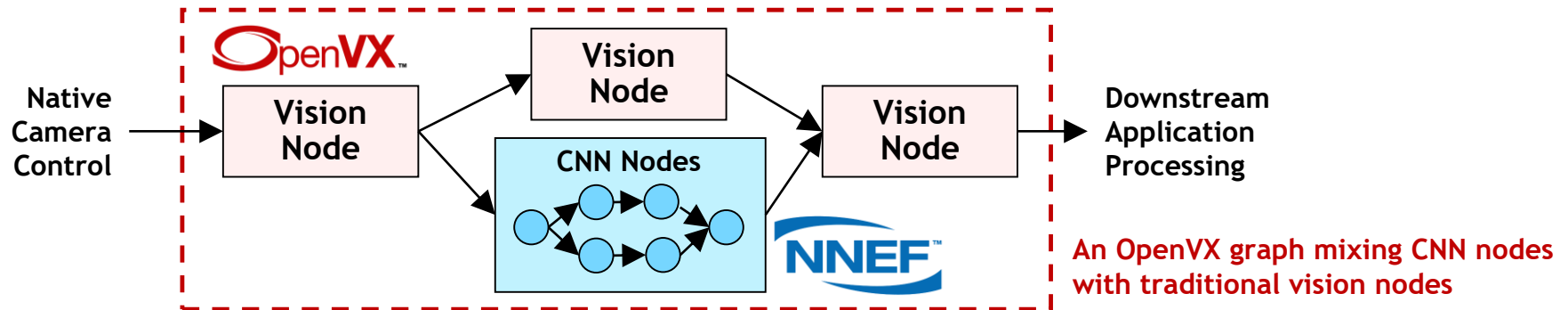
- **Feature detection:** find features useful for object detection and recognition
 - Histogram of gradients - HOG
 - Local binary patterns - LBP
 - Template matching
 - Line finding
- **Classification:** detect and recognize objects in an image based on a set of features
 - Import a classifier model trained offline
 - Classify objects based on a set of input features
- **Image Processing:** transform an image
 - Generalized nonlinear filter: Dilate, erode, median with arbitrary kernel shapes
 - Non maximum suppression: Find local maximum values in an image
 - Bilateral filter: edge-preserving noise reduction
- **Conditional execution & node predication**
 - Selectively execute portions of a graph based on a true/false predicate
- Many, many minor improvements
- New Extensions
 - **Import/export:** compile a graph; save and run later
 - **16-bit support:** signed 16-bit image data
 - **Neural networks:** Layers are represented as OpenVX nodes



If A then $S \leftarrow B$ else $S \leftarrow C$

OpenVX 1.2 and Neural Net Extension

- Convolution Neural Network topologies can be represented as OpenVX graphs
 - Layers are represented as OpenVX nodes
 - Layers connected by multi-dimensional tensors objects
 - Layer types include convolution, activation, pooling, fully-connected, soft-max
 - CNN nodes can be mixed with traditional vision nodes
- Import/Export Extension
 - Efficient handling of network Weights/Biases or complete networks
- OpenVX will be able to import NNEF files into OpenVX Neural Nets



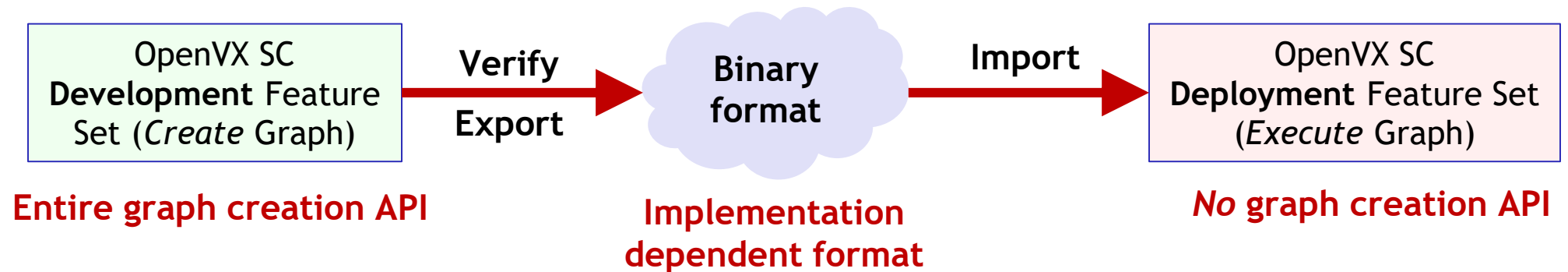
OpenVX Neural Network Extension

- Tensor types of INT16, INT8, and UINT8 are supported
 - Other types may be supported by a vendor
- Conformance tests will be up to some “tolerance” in precision
 - To allow for optimizations, e.g., weight compression
- Eight neural network “layer” nodes:



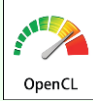
vxActivationLayer	vxConvolutionLayer	vxDeconvolutionLayer
vxFullyConnectedLayer	vxNormalizationLayer	vxPoolingLayer
vxSoftmaxLayer	vxROIPoolingLayer	...

OpenVX SC - Safety Critical Vision Processing

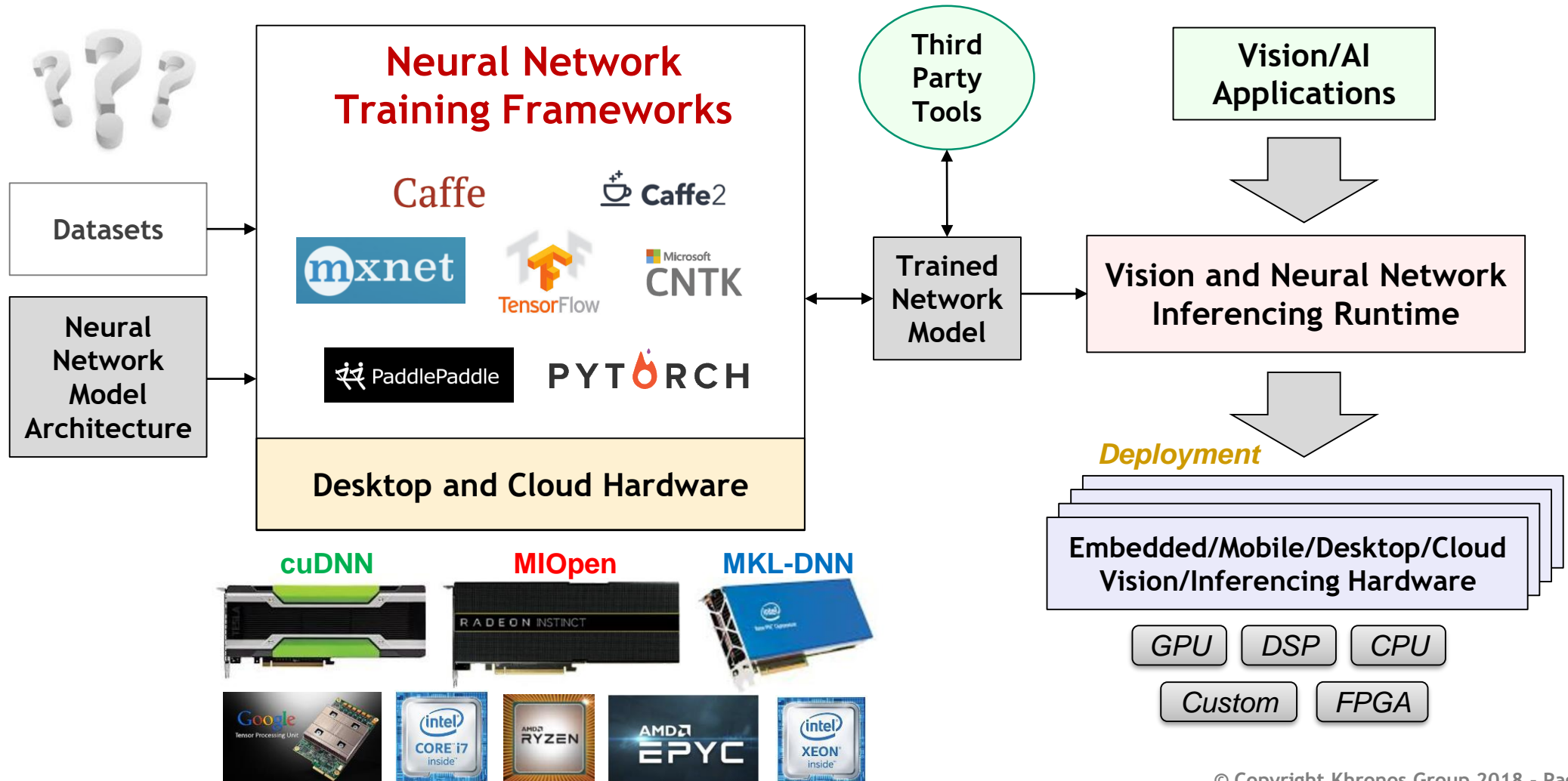
- OpenVX 1.1 - based on OpenVX 1.1 main specification
 - Enhanced determinism
 - Specification identifies and numbers requirements
- MISRA C clean per KlocWorks v10
- Divides functionality into “**development**” and “**deployment**” feature sets
 - Adds requirement to support import/export extension



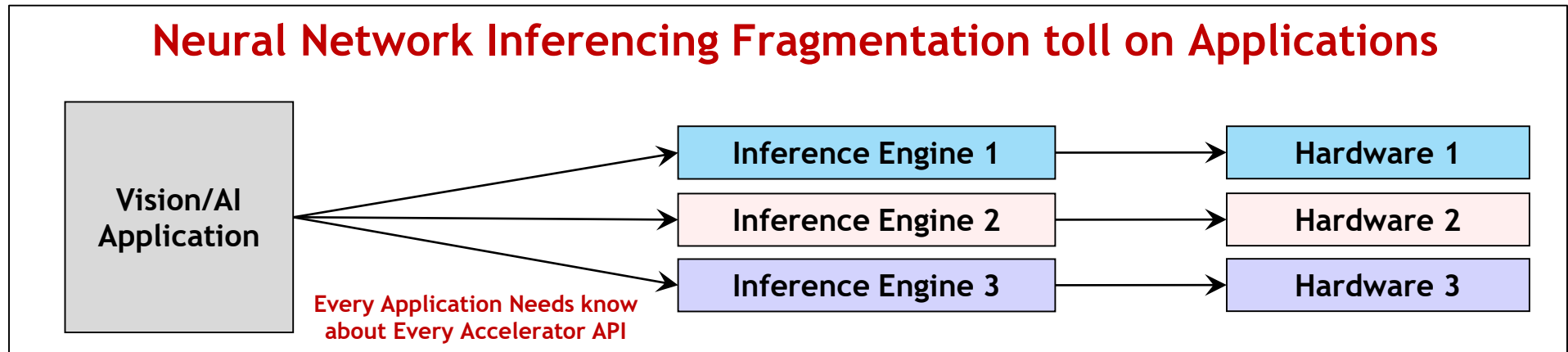
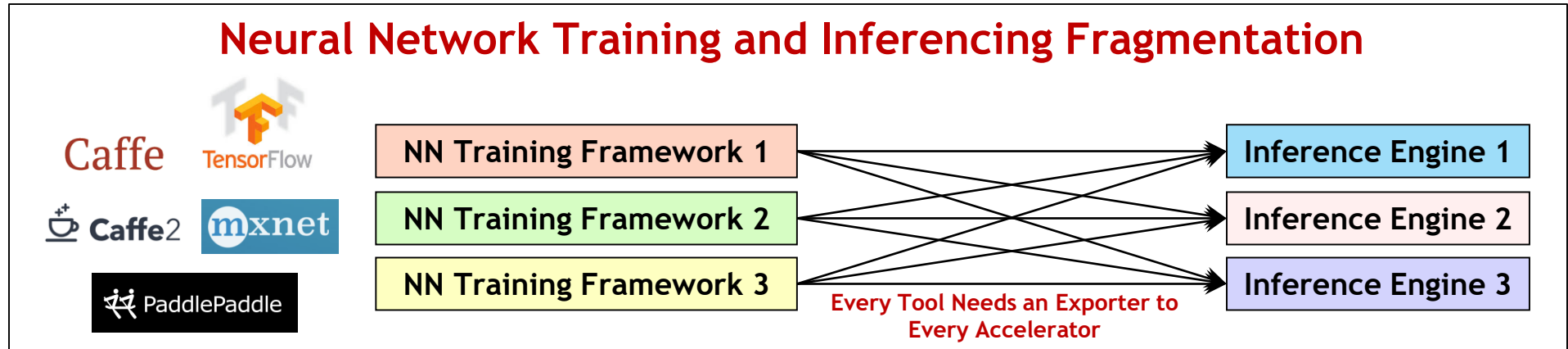
How OpenVX Compares to Alternatives

			
Governance	Open standard API designed to be implemented and shipped by IHVs	Community-driven, open source library	Open standard API designed to be implemented and shipped by IHVs
Programming Model	Graph defined with C API and then compiled for run-time execution	Immediate runtime function calls - reading to and from memory	Explicit kernels are compiled and executed via run-time API
Built-in Vision Functionality	Small but growing set of popular functions	Vast. Mainly on PC/CPU	None. User programs their own or call vision library over OpenCL
Target Hardware	Any combination of processors or non-programmable hardware	Mainly PCs and GPUs	Any heterogeneous combination of IEEE FP-capable processors
Optimization Opportunities	Pre-declared graph enables significant optimizations	Each function reads/writes memory. Power performance inefficient	Any execution topology can be explicitly programmed
Conformance	Implementations must pass conformance to use trademark	Extensive Test Suite but no formal Adopters program	Implementations must pass conformance to use trademark
Consistency	All core functions must be available in conformant implementations	Available functions vary depending on implementation / platform	All core functions must be available in all conformant implementations

Neural Network End-to-End Workflow

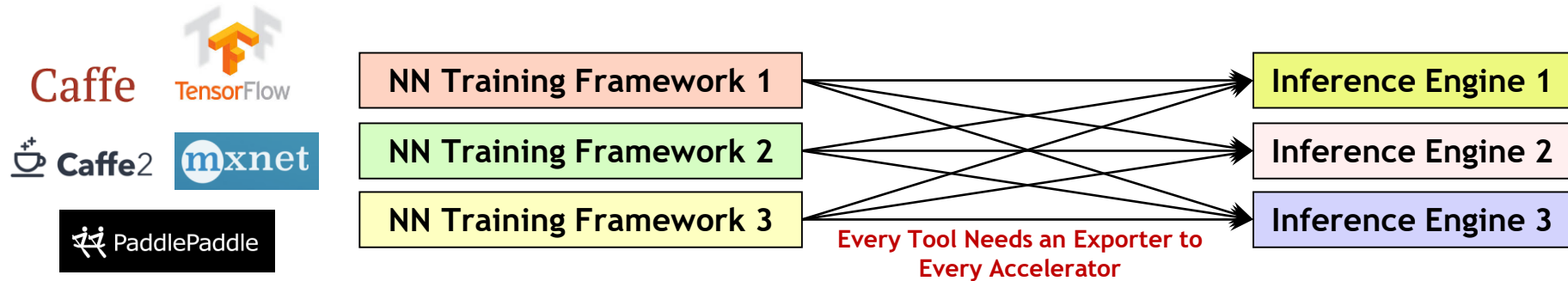


Problem: Neural Network Fragmentation

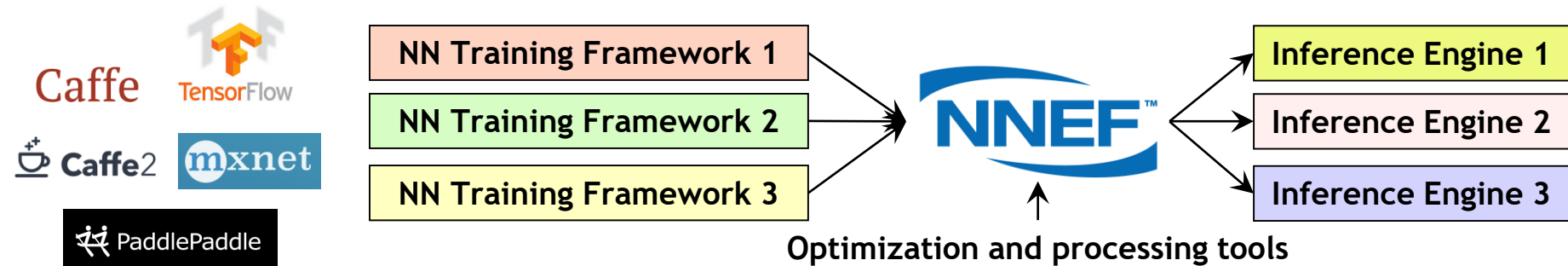


NNEF - Solving Neural Network Fragmentation

Before NNEF - NN Training and Inferencing Fragmentation



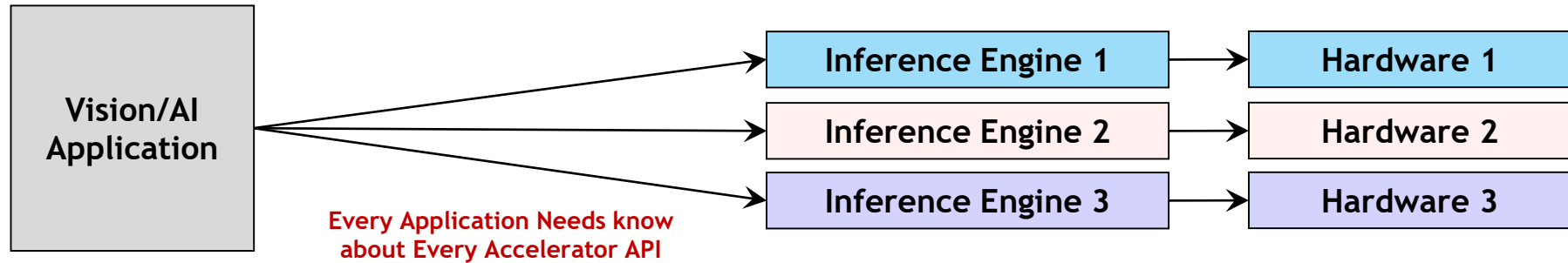
With NNEF- NN Training and Inferencing Interoperability



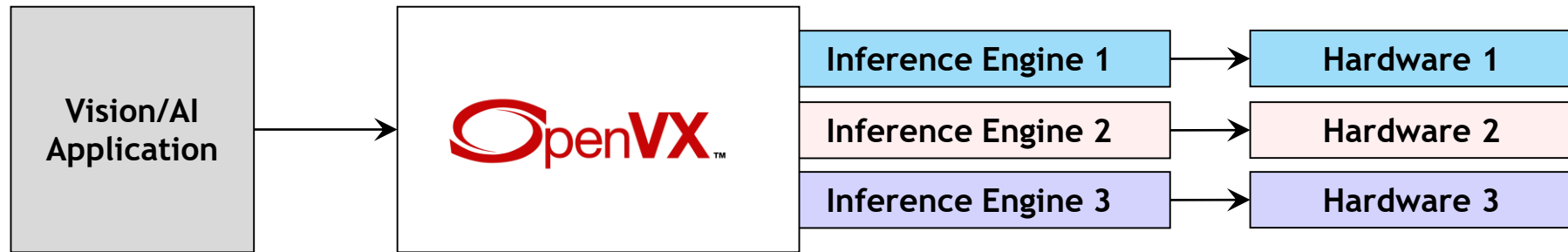
NNEF is a Cross-vendor Neural Net file format
Encapsulates network formal semantics, structure, data formats,
commonly-used operations (such as convolution, pooling, normalization, etc.)

OpenVX - Solving Inferencing Fragmentation

Before OpenVX - Vision and NN Inferencing Fragmentation

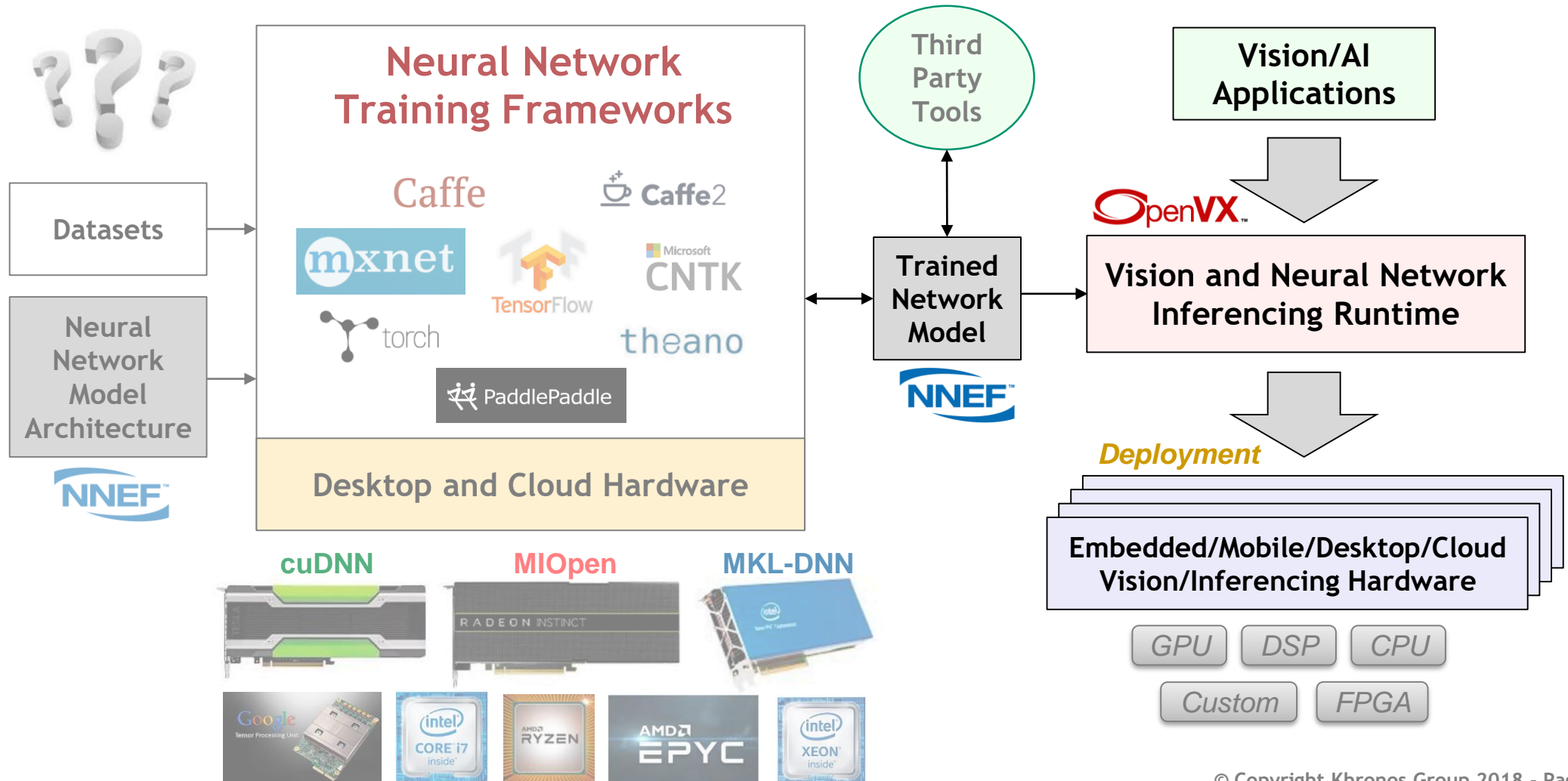


With OpenVX - Vision and NN Inferencing Interoperability



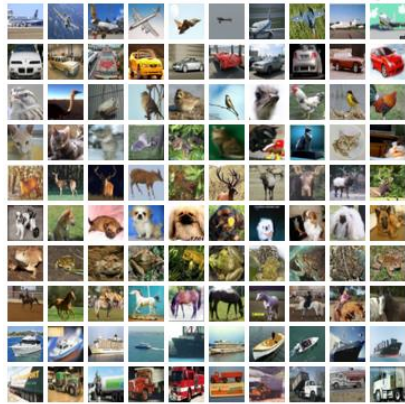
OpenVX is an open, royalty-free standard for cross platform acceleration of computer vision and neural network applications.

Neural Net Workflow with Khronos Standards

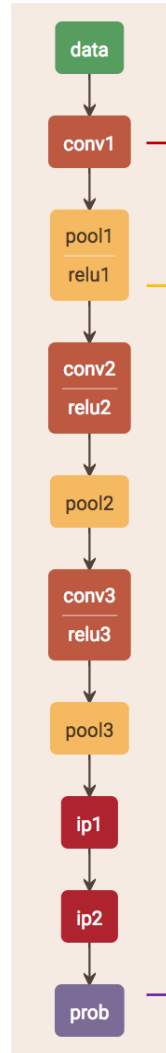


CIFAR-10 Example: NNEF Structure

Input: 32x32 RGB image



Output: 10 classes
airplane, automobile,
bird, cat, deer, dog, frog,
horse, ship, truck

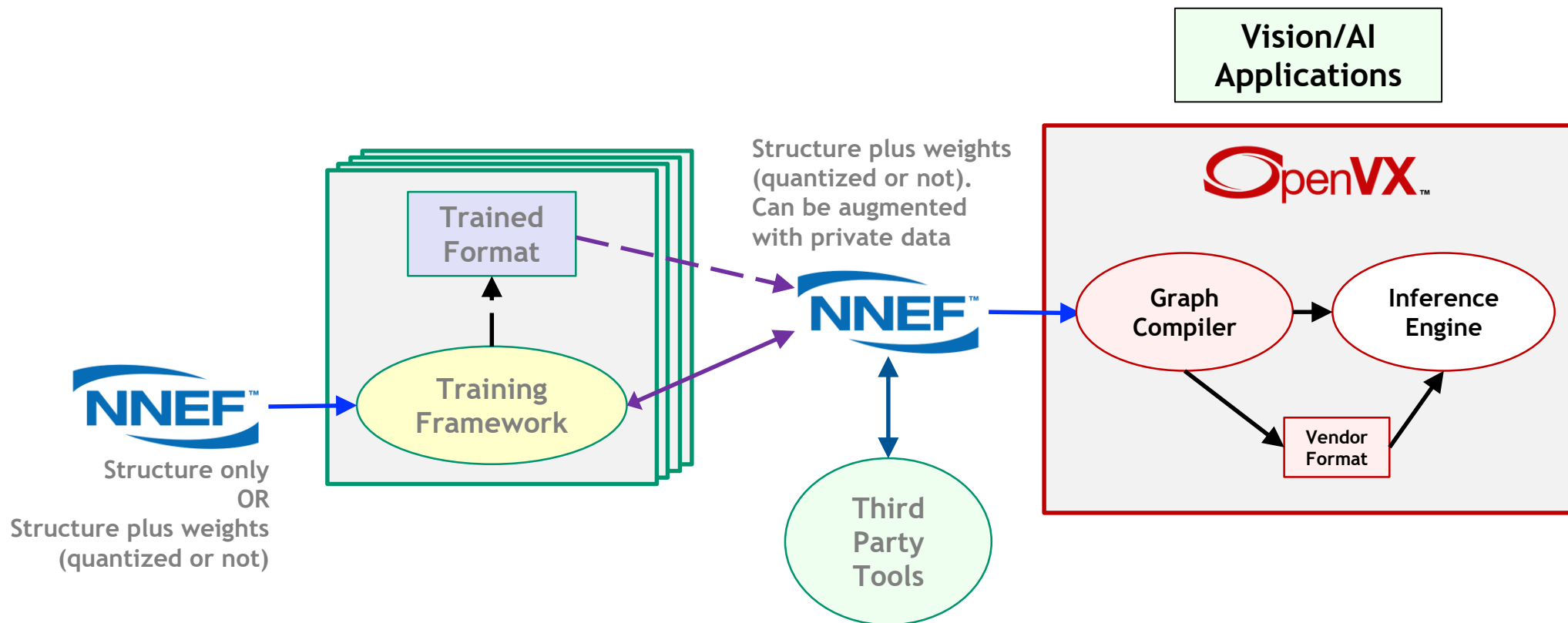


version 1.0

```
graph cifar10( data ) -> ( prob ) {  
    data = external(shape = [0,3,32,32]);  
    conv1w = variable(shape = [32,3,5,5], label = 'conv1/weights');  
    conv1b = variable(shape = [1,32], label = 'conv1/bias');  
    conv1 = conv(data, conv1w, conv1b, padding = [(2,2),(2,2)]);  
    pool1 = max_pool(conv1, size=[1,1,3,3], stride=[1,1,2,2], no_border);  
    relu1 = relu(pool1);  
    conv2w = variable(shape = [32,32,5,5], label = 'conv2/weights');  
    conv2b = variable(shape = [1,32], label = 'conv2/bias');  
    conv2 = conv(relu1, conv2w, conv2b, padding = [(2,2),(2,2)]);  
    relu2 = relu(conv2);  
    pool2 = avg_pool(relu2, size=[1,1,3,3], stride=[1,1,2,2], no_border);  
    conv3w = variable(shape = [64,32,5,5], label = 'conv3/weights');  
    conv3b = variable(shape = [1,64], label = 'conv3/bias');  
    conv3 = conv(pool2, conv3w, conv3b, padding = [(2,2),(2,2)]);  
    relu3 = relu(conv3);  
    pool3 = avg_pool(relu3, size=[1,1,3,3], stride=[1,1,2,2], no_border);  
    ip1w = variable(shape = [64,64,4,4], label = 'ip1/weights');  
    ip1b = variable(shape = [1,64], label = 'ip1/bias');  
    ip1 = conv(pool3, ip1w, ip1b, padding = [(0,0),(0,0)]);  
    ip2w = variable(shape = [10,64,1,1], label = 'ip2/weights');  
    ip2b = variable(shape = [1,10], label = 'ip2/bias');  
    ip2 = conv(ip1, ip2w, ip2b, padding = [(0,0),(0,0)]);  
    prob = softmax(ip2);  
}
```

border = 'ignore'

Application Development Workflow



Khronos open-source projects:

- **NNEF Parser:** base parser for use by “NNEF validator tool” and “NNEF Importers”
- **NNEF Exporters:** generate NNEF from deep learning frameworks

AMD OpenVX open-source project with NN

bit.ly/openvx-amd

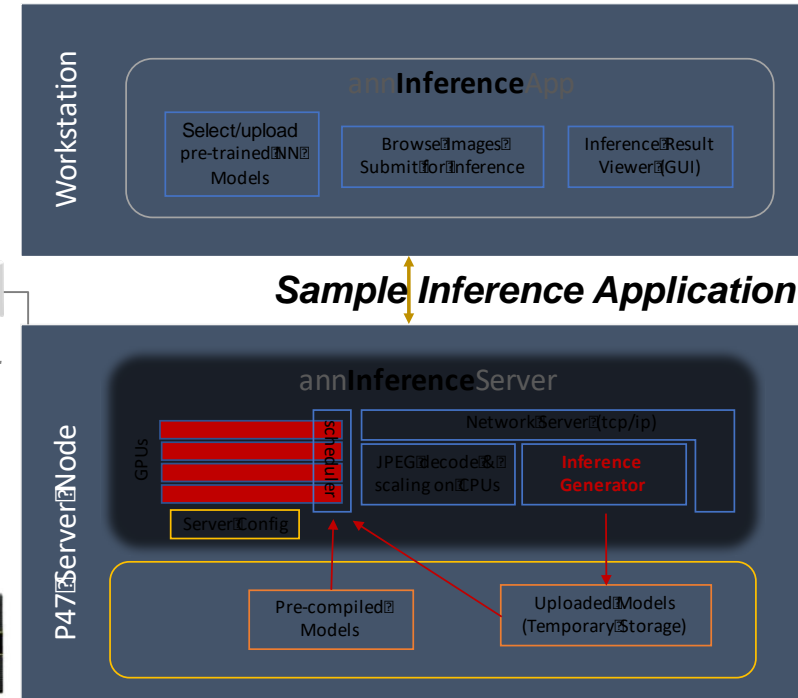
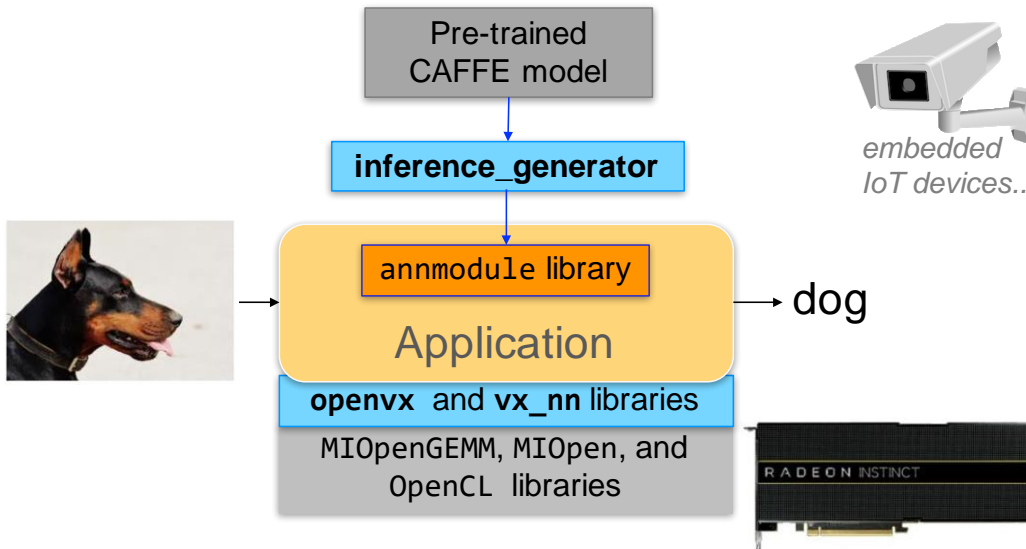
All OpenVX projects on GitHub

LOOM: 360° live video stitching

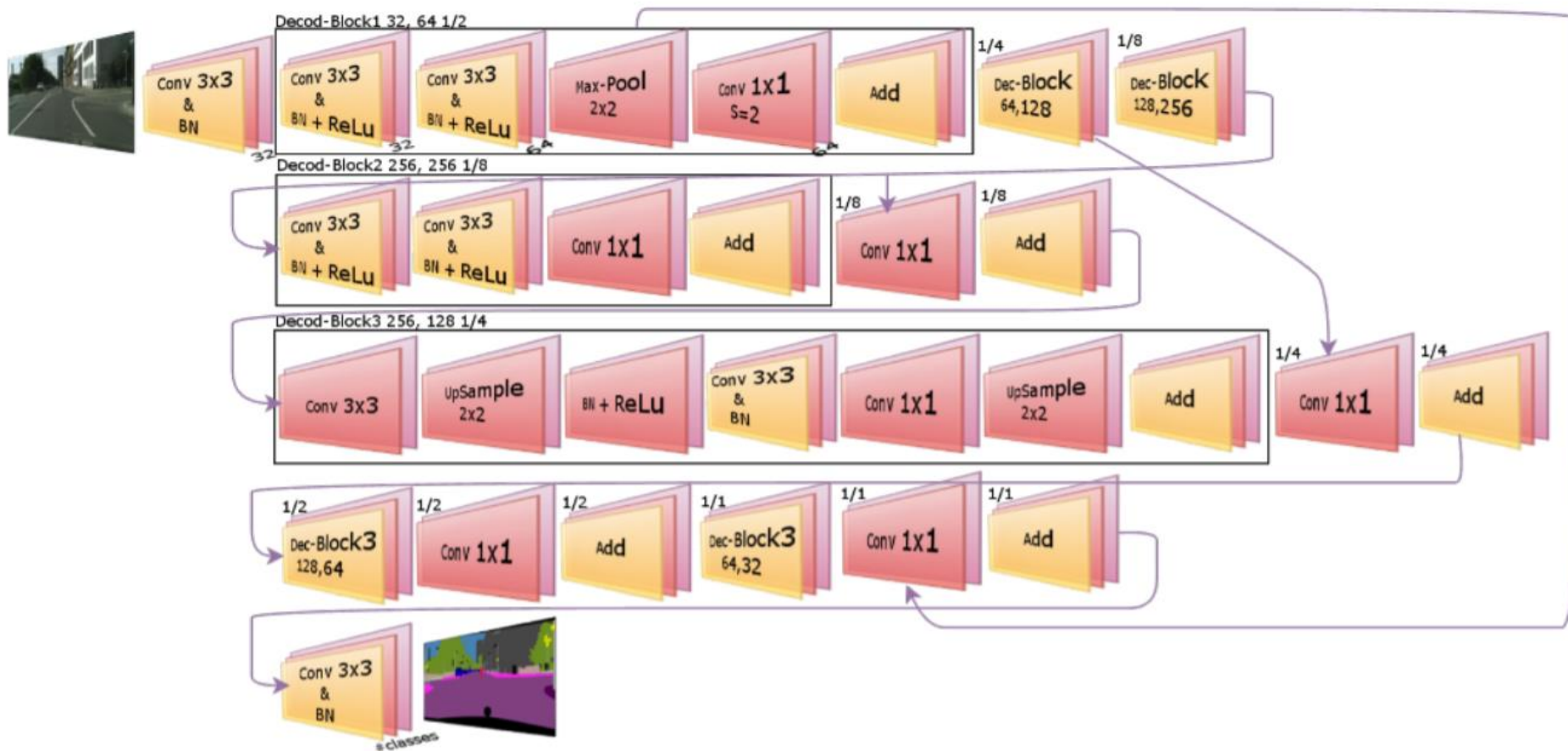


Neural network acceleration for inference

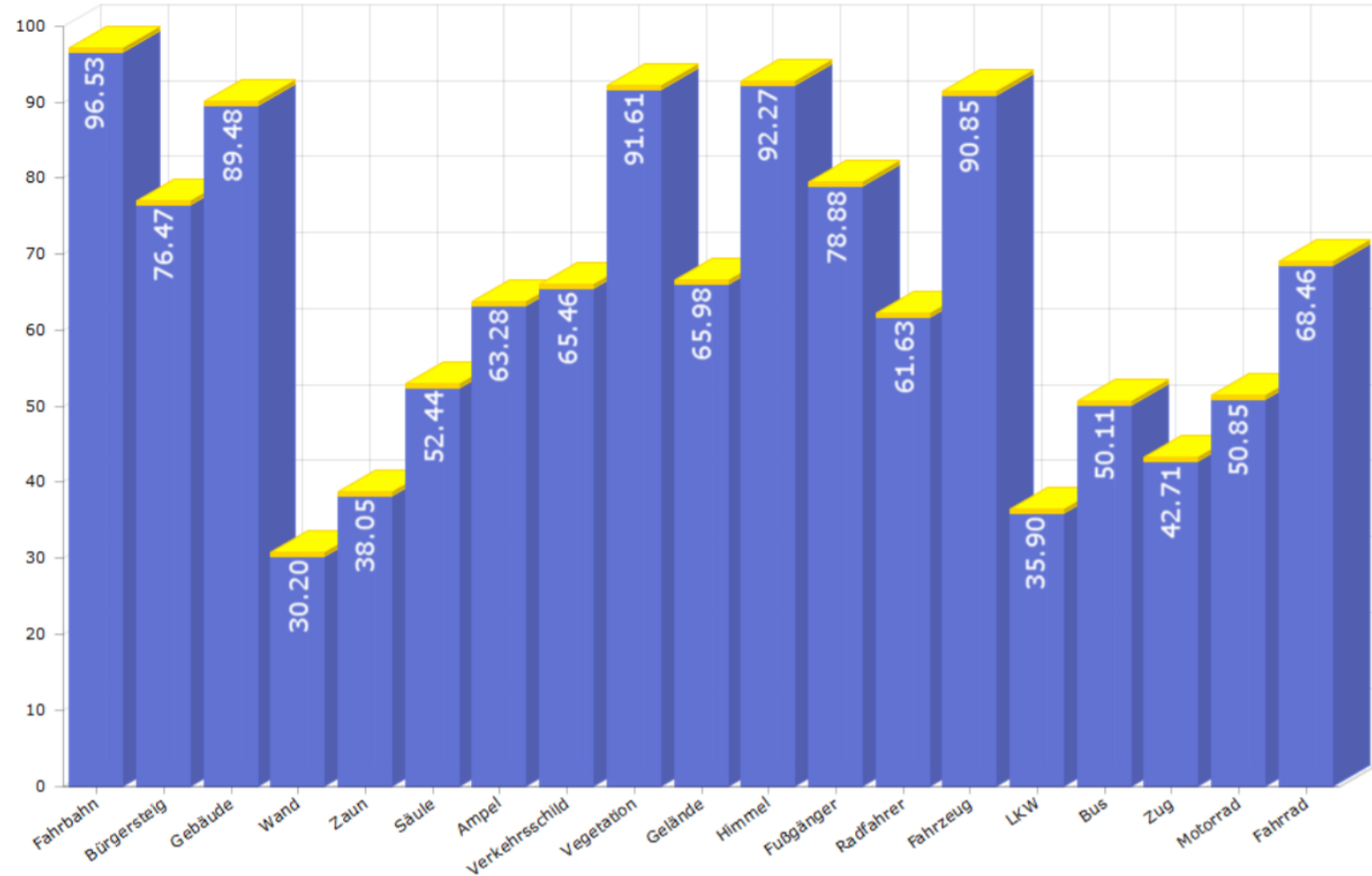
Inference Application Development Workflow



AMD OpenVX open-source project with NN



AMD OpenVX open-source project with NN



AMD OpenVX open-source project with NN



AMD OpenVX open-source project with NN



	Params	Layers	IoU_cls	iIoU_cls	IoU_cat	iIoU_cls	M5000	WX7100	Vega56
Netz-2	3500	26	65,32	39,28	79,60	65,39	6,3	15	18

