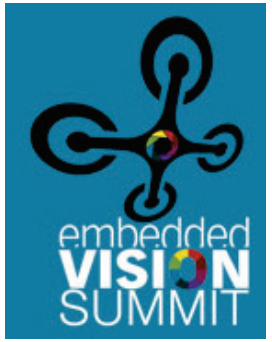


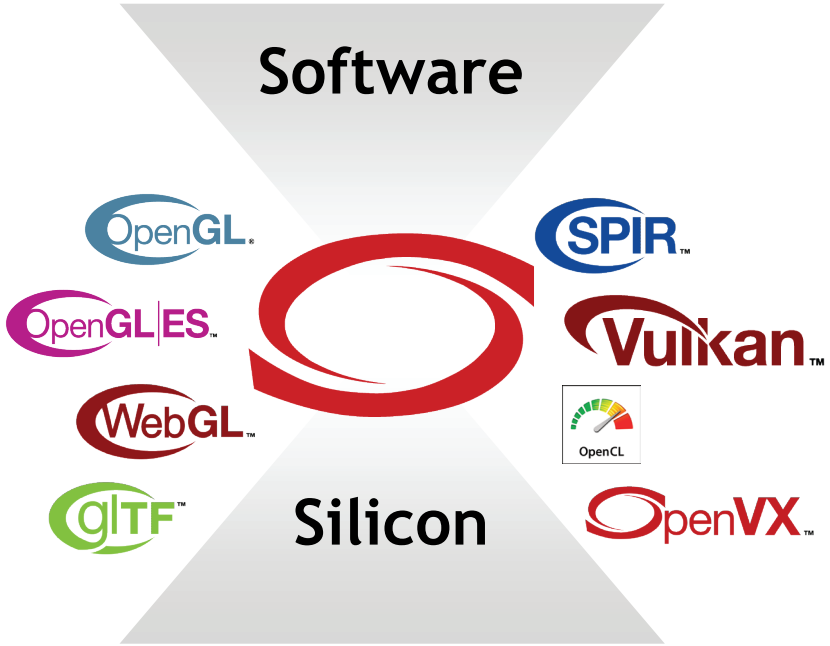
KHRONOS™ GROUP

OpenVX Ecosystem Overview

OpenVX Workshop, Embedded Vision Summit, May 2016

Neil Trevett | Khronos President
NVIDIA Vice President Developer Ecosystem

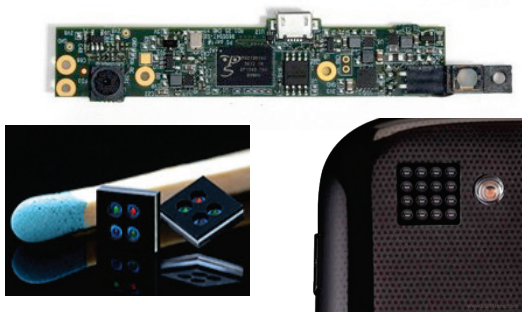
Khronos Open Standards



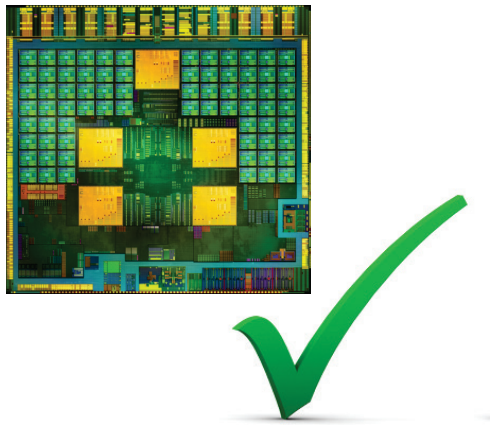
Khronos is an Industry Consortium of over 100 companies creating royalty-free, open standard APIs to enable software to access hardware acceleration for graphics, parallel compute and vision

Vision Pipeline Challenges and Opportunities

Growing Camera Diversity



Diverse Vision Processors



Sensor Proliferation

Flexible sensor and camera control to GENERATE an image stream

OpenKCam™

Use efficient acceleration to PROCESS the image stream

OpenVX™

Combine vision output with other sensor data on device

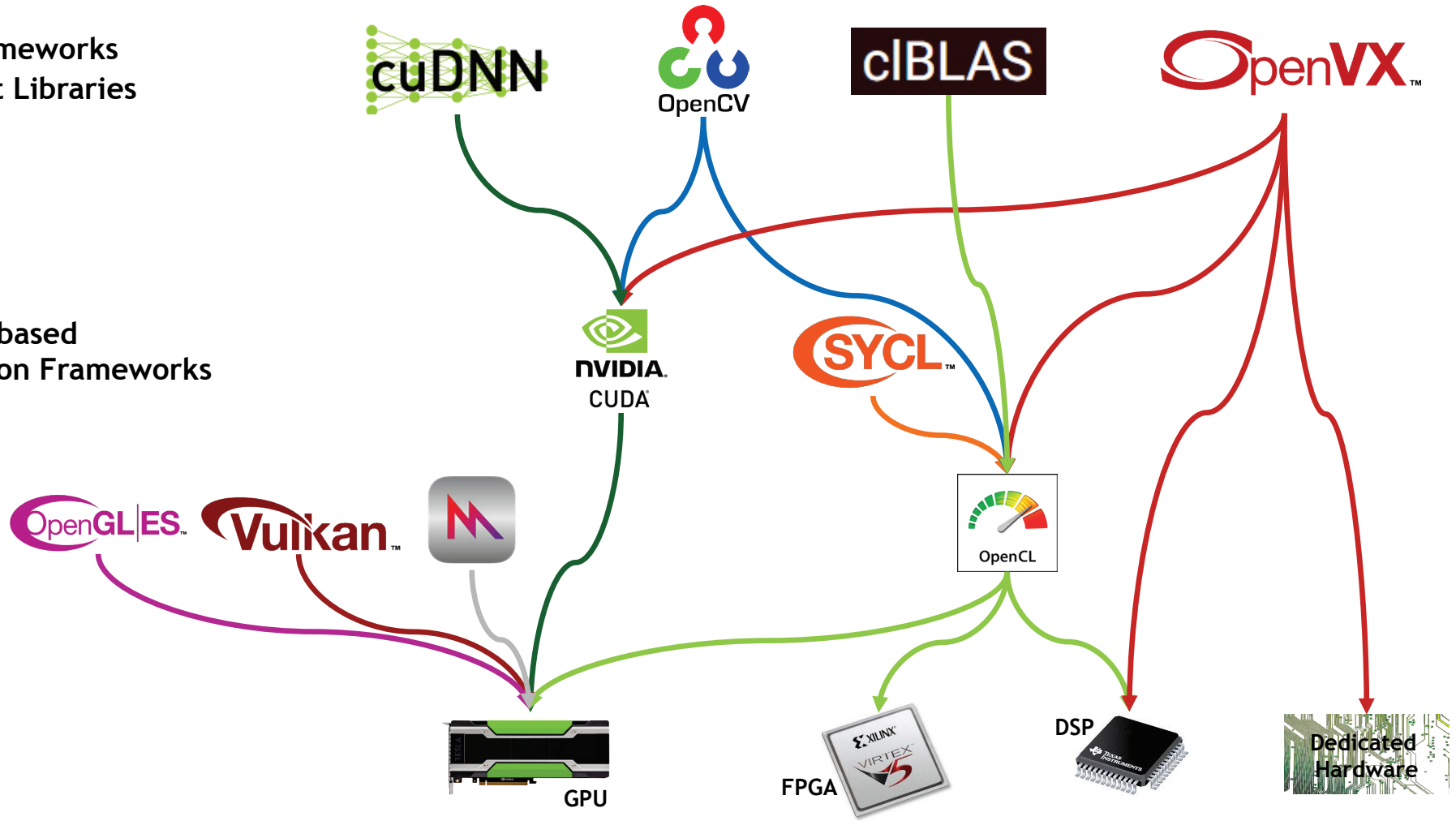
StreamInput™

Accelerated Vision API Jungle

Vision Frameworks
Neural Net Libraries

Language-based
Acceleration Frameworks

Explicit
Kernels

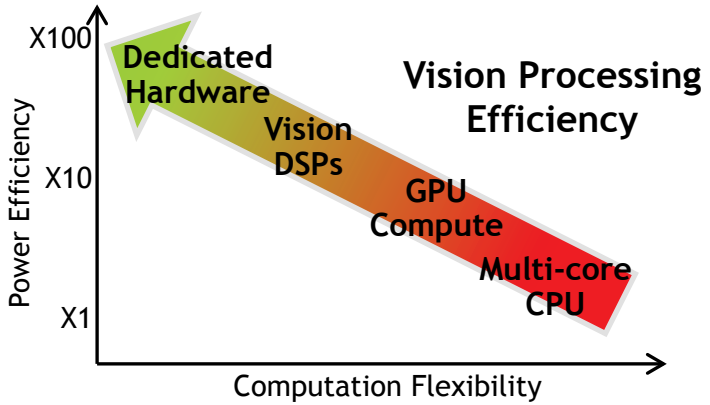


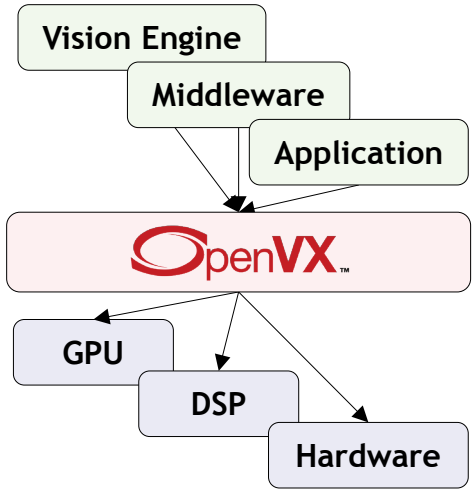
OpenVX and OpenCV are Complementary

	OpenCV	OpenVX
Implementation	Community driven open source library	Open standard API designed to be implemented by hardware vendors
Conformance	Extensive OpenCV Test Suite but no formal Adopters program	Implementations must pass defined conformance test suite to use trademark
Consistency	Available functions can vary depending on implementation / platform	All core functions must be available in all conformant implementations
Scope	Very wide 1000s of imaging and vision functions Multiple camera APIs/interfaces	Tight focus on core hardware accelerated functions for mobile vision – but extensible Uses external/native camera API
Efficiency	Memory-based architecture Each operation reads and writes to memory	Graph-based execution Optimizable computation and data transfer
Typical Use Case	Rapid experimentation and prototyping - especially on desktop	Production development & deployment on mobile and embedded devices
Embedded Deployment	Re-usable code	Callable API

OpenVX – Low Power Vision Acceleration

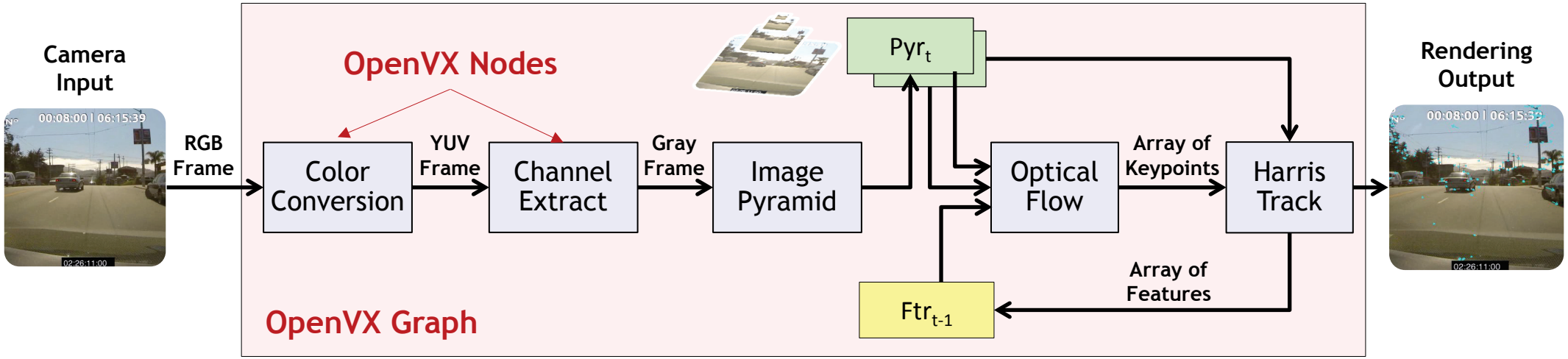
- **Precisely defined API for production deployment of vision acceleration**
  - Targeted at real-time mobile and embedded platforms
- **Higher abstraction than OpenCL for performance portability across diverse architectures**
  - Multi-core CPUs, GPUs, DSPs and DSP arrays, ISPs, Dedicated hardware…
- **Extends portable vision acceleration to very low power domains**
  - Doesn't require high-power CPU/GPU Complex or OpenCL precision
  - Low-power host can setup and manage frame-rate graph





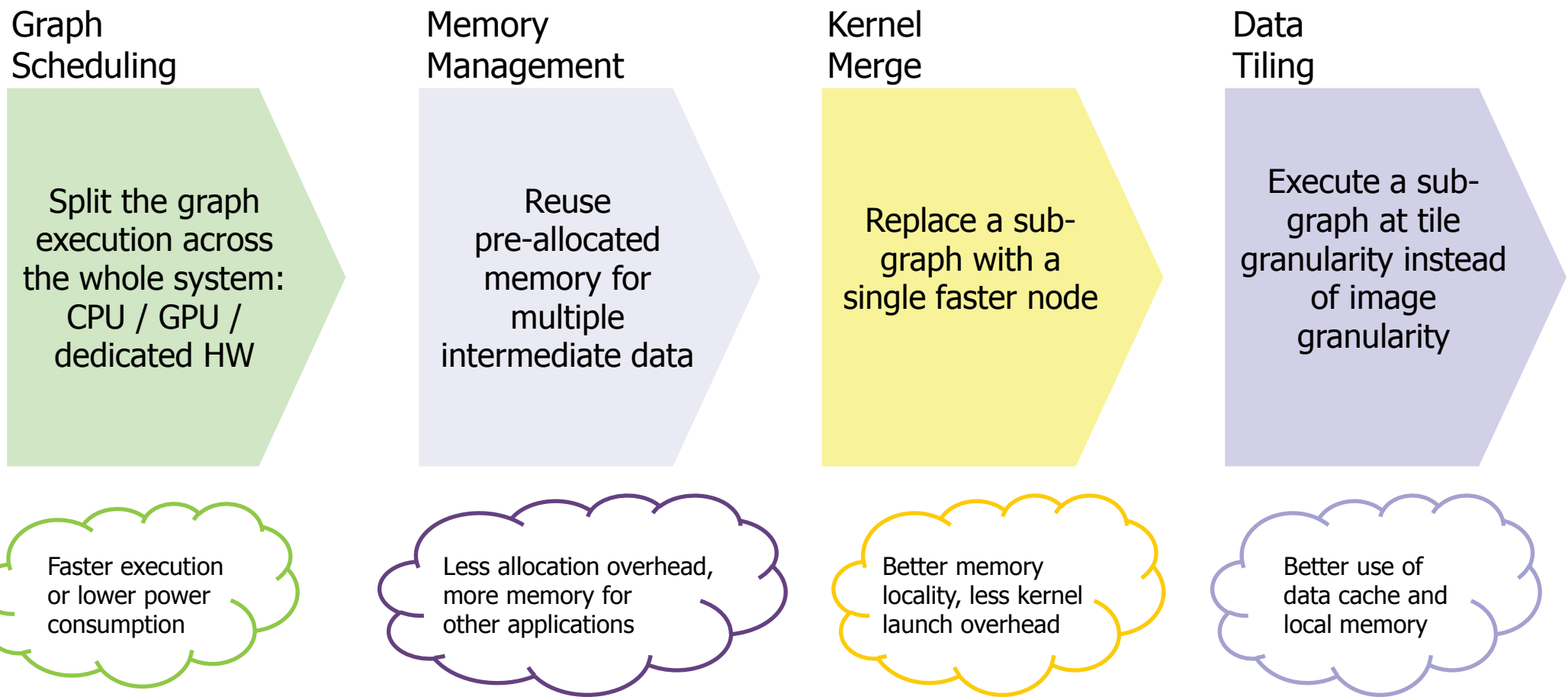
OpenVX Graphs

- **OpenVX developers express a graph of image operations ('Nodes')**
  - Nodes can be on any hardware or processor coded in any language
- **Graphs can execute almost autonomously**
  - Possible to Minimize host interaction during frame-rate graph execution
- **Graphs are the key to run-time optimization opportunities…**

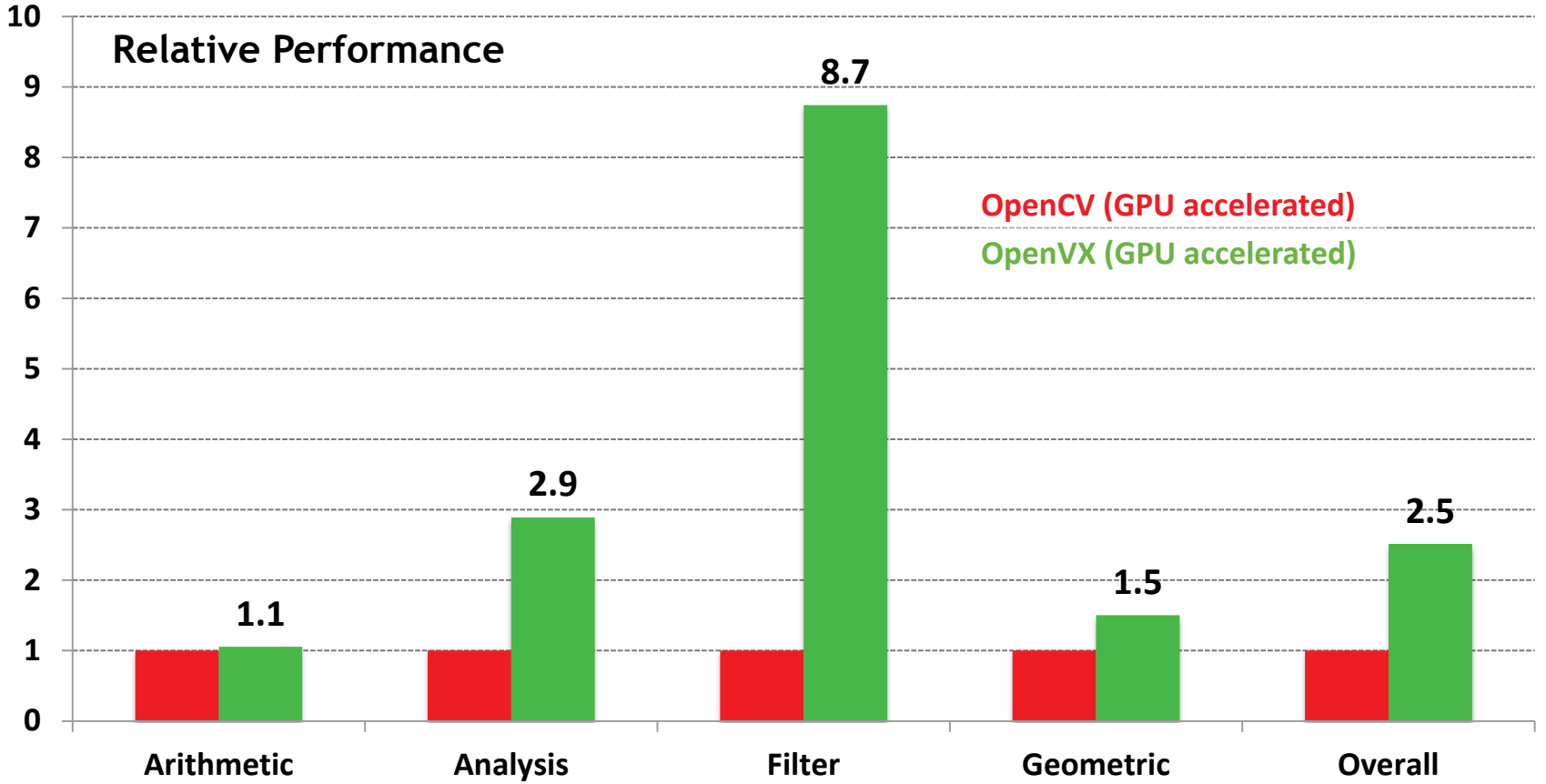


Feature Extraction Example Graph

OpenVX Efficiency through Graphs..



Example Relative Performance



NVIDIA implementation experience.
Geometric mean of >2200 primitives, grouped into each categories, running at different image sizes and parameter settings

Layered Vision Processing Ecosystem

Implementers may use OpenCL or Compute Shaders to *implement* OpenVX nodes on programmable processors

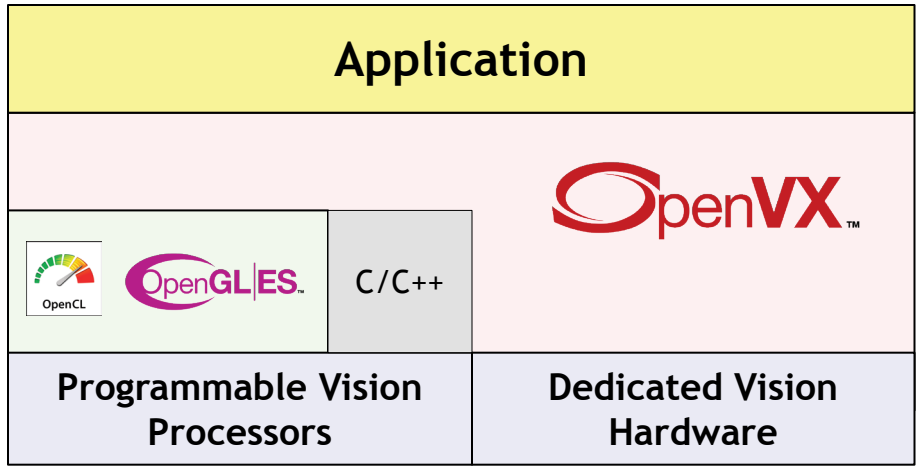
And then developers can use OpenVX to enable a developer to easily *connect* those nodes into a graph

Application Software

Engines/frameworks

Powerful, flexible low-level APIs / languages

Processor Hardware



OpenVX enables the graph to be *extended* to include hardware architectures that don't support programmable APIs

The OpenVX graph enables implementers to *optimize* execution across diverse hardware architectures an drive to lower power implementations

OpenVX 1.0 Shipping, OpenVX 1.1 Released!

- **Multiple OpenVX 1.0 Implementations shipping – spec in October 2014**
  - Open source sample implementation and conformance tests available
- **OpenVX 1.1 Specification released 2nd May 2016 at Embedded Vision Summit**
  - Expands node functionality AND enhances graph framework
  - Sample source and conformance tests will be updated to OpenVX 1.1 in 1H16
- **OpenVX is EXTENSIBLE**
  - Implementers can add their own nodes at any time to meet customer and market needs

What's New in OpenVX 1.1?

- **Expanded node functionality AND enhanced graph framework**
  - Plus many minor improvements and clarifications
- **Laplacian pyramids**
  - Computational photography use cases
- **Targets - for execution flexibility on heterogeneous devices**
  - Application can control on which accelerator to run nodes
- **Median, erode and dilate image filters**
  - Including custom patterns
- **Improved read and write data to and from OpenVX objects**
  - Easier to use and less error prone
- **Improved API for extending OpenVX with user kernels**
  - More convenience and flexibility

OpenVX Roadmap and Safety Critical APIs

OpenGL SC 1.0 - 2005
Fixed function graphics subset

OpenGL ES 1.0 - 2003
Fixed function graphics

OpenGL ES 2.0 - 2007
Shader programmable pipeline

OpenGL SC 2.0 - April 2016
Shader programmable pipeline subset

Experience and Guidelines

New Generation APIs for safety certifiable vision, graphics and compute
e.g. ISO 26262 and DO-178B/C

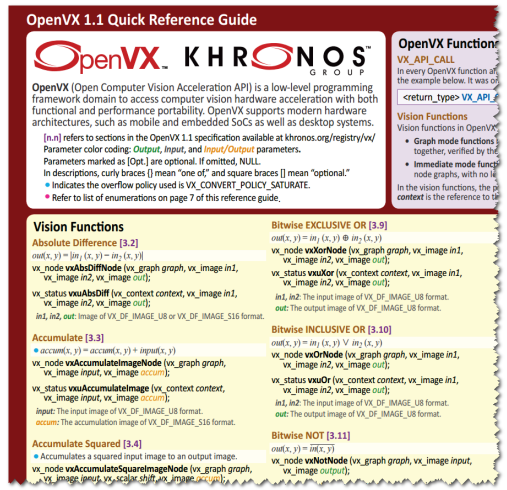
OpenVX SC?

OpenVX Roadmap Discussions
Significantly broaden node functionality
In-graph neural nets
Programmable nodes (OpenCL or SPIR-V?)
Market-specific feature sets

Small driver size
Advanced functionality
Graphics and compute

Thank You for Coming Today!

- **PDF Quick Reference Cards available for all versions of Khronos specificationa**
  - https://www.khronos.org/files/openvx-11-reference-card.pdf
- **OpenVX Forums!**
  - https://forums.khronos.org/showthread.php/13063-Khronos-Releases-OpenVX-1-1-Specification
- **Please give us your feedback on today's workshop**
  - We want to work to make these days as useful as possible
- **Please talk to us about what you would like to see in the OpenVX roadmap**
  - We are genuinely interested in your input!
- **Any company or organization is welcome to join Khronos for a voice and a vote in any of its standards**
  - www.khronos.org
- **Neil Trevett**
  - ntrevett@nvidia.com
  - @neilt3d


OpenVX 1.1 Quick Reference Guide

OpenVX (Open Computer Vision Acceleration API) is a low-level programming framework domain to access computer vision hardware acceleration with both functional and performance portability. OpenVX supports modern hardware architectures, such as mobile and embedded SoCs as well as desktop systems.

[n.n] refers to sections in the OpenVX 1.1 specification available at khronos.org/registry/vx/

Parameter color coding: *Output*, *Input*, and *Input/Output* parameters.

Parameters marked as [Opt.] are optional. If omitted, NULL.

In descriptions, curly braces {} mean "one of," and square braces [] mean "optional."

- Indicates the overflow policy used is VX_CONVERT_POLICY_SATURATE.
- Refer to list of enumerations on page 7 of this reference guide.

OpenVX Function

VX_API_CALL

<return_type> VX_API_

Vision Functions

Vision functions in OpenVX

- Graph mode functions
- Immediate mode funct

node graphs, with no le

In the vision functions, the p

context is the reference to t

Vision Functions

Absolute Difference [3.2]

$out(x, y) = |in_1(x, y) - in_2(x, y)|$

vx_node vxAbsDiffNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out);

vx_status vxuAbsDiff (vx_context context, vx_image in1, vx_image in2, vx_image out);

in1, in2, out: Image of VX_DF_IMAGE_U8 or VX_DF_IMAGE_S16 format.

Accumulate [3.3]

- $accum(x, y) = accum(x, y) + input(x, y)$

vx_node vxAccumulateImageNode (vx_graph graph, vx_image input, vx_image accum);

vx_status vxuAccumulateImage (vx_context context, vx_image input, vx_image accum);

input: The input image of VX_DF_IMAGE_U8 format.

accum: The accumulation image of VX_DF_IMAGE_S16 format.

Accumulate Squared [3.4]

- Accumulates a squared input image to an output image.

vx_node vxAccumulateSquareImageNode (vx_graph graph, vx_image input, vx_scalar shift, vx_image accum);

Bitwise EXCLUSIVE OR [3.9]

$out(x, y) = in_1(x, y) \oplus in_2(x, y)$

vx_node vxXorNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out);

vx_status vxuXor (vx_context context, vx_image in1, vx_image in2, vx_image out);

in1, in2: The input image of VX_DF_IMAGE_U8 format.

out: The output image of VX_DF_IMAGE_U8 format.

Bitwise INCLUSIVE OR [3.10]

$out(x, y) = in_1(x, y) \vee in_2(x, y)$

vx_node vxOrNode (vx_graph graph, vx_image in1, vx_image in2, vx_image out);

vx_status vxuOr (vx_context context, vx_image in1, vx_image in2, vx_image out);

in1, in2: The input image of VX_DF_IMAGE_U8 format.

out: The output image of VX_DF_IMAGE_U8 format.

Bitwise NOT [3.11]

$out(x, y) = \overline{in(x, y)}$

vx_node vxNotNode (vx_graph graph, vx_image input, vx_image output);