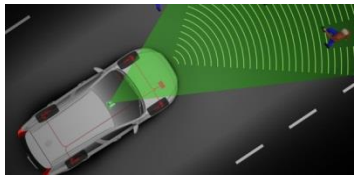
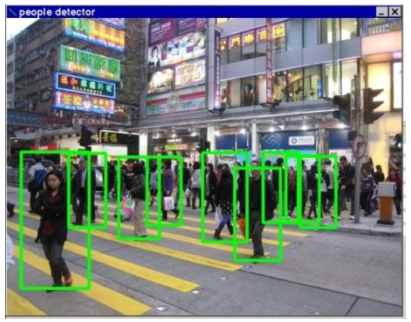
SYNOPSYS®
Silicon to Software™

Effective Programming Models for Embedded Vision Processing





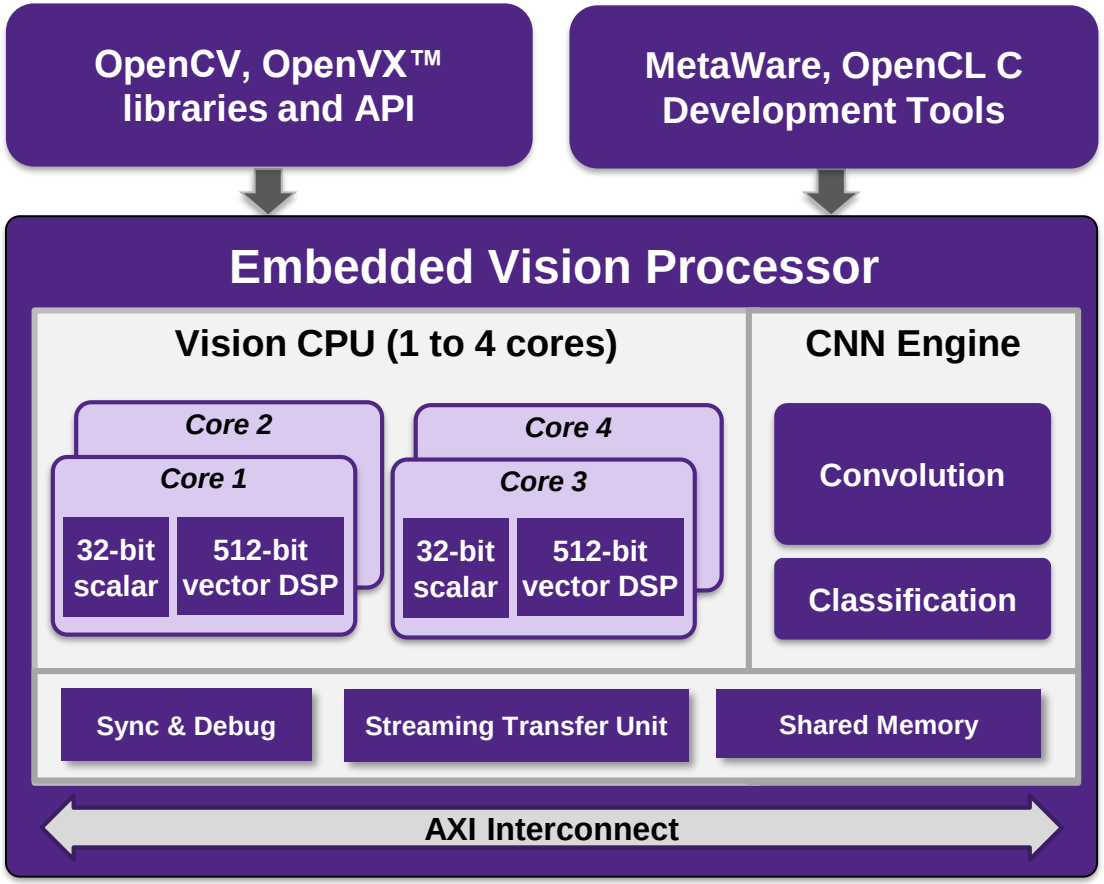
Pierre Paulin
Director of R&D

July 2016

Outline

- EV6 Embedded Vision Processor Overview
- EV Programming Tools
  - OpenVX runtime
  - OpenCL C with auto-vectorization
  - CNN programming tools

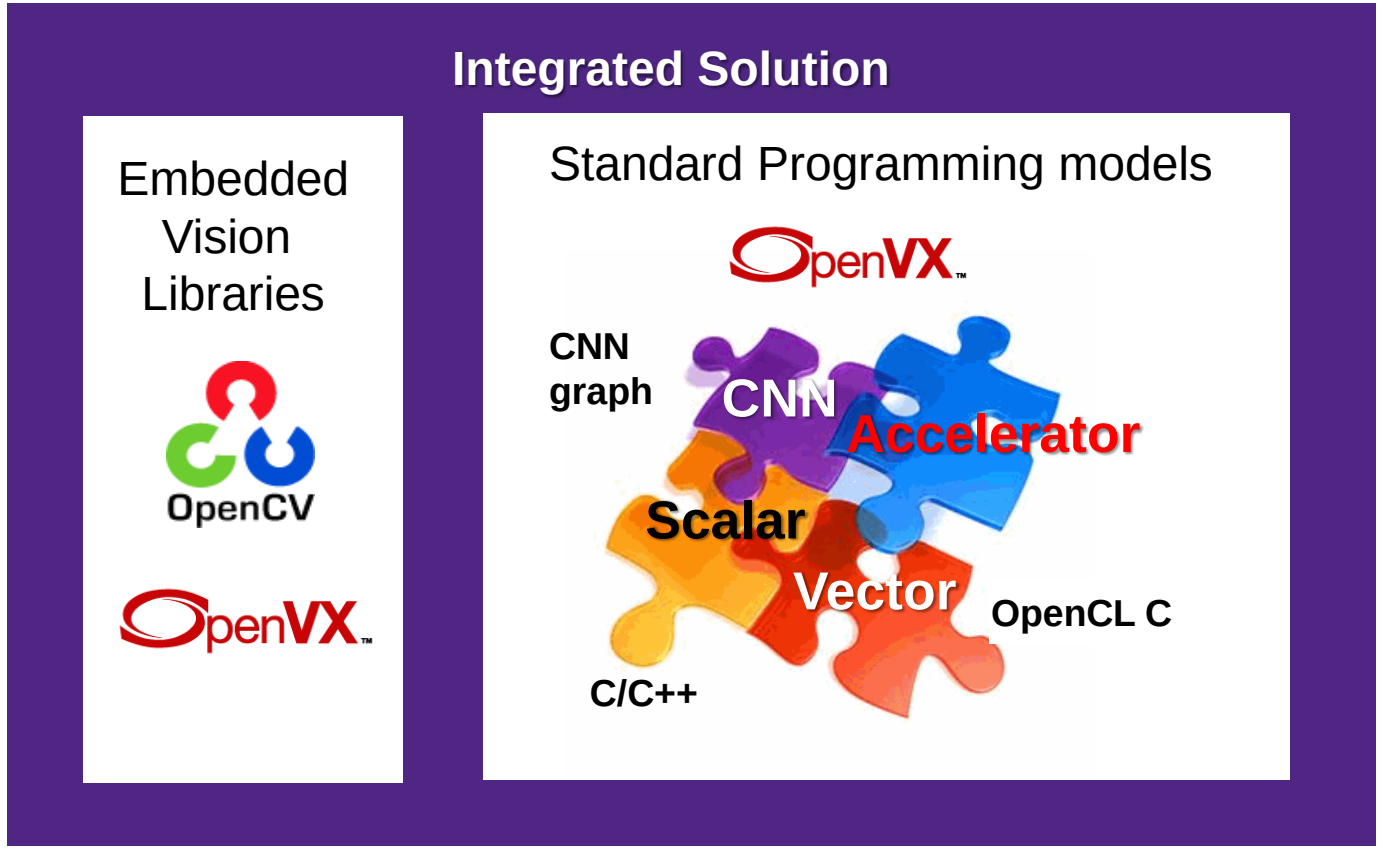
EV6x Scalable Embedded Vision Processors



- Highly integrated and configurable
  - Configurable scalar, vector DSP and convolutional neural network (CNN) architecture
  - Supports 1080p - 4K vision streams
- User scalable for optimum performance
  - 1 to 4 Vision CPU cores
  - Programmable CNN engine
- State-of-the-art performance-efficiency
- High productivity toolset
  - OpenCV, OpenVX, OpenCL C

SYNOPSYS®

EV Processor Programming Tools

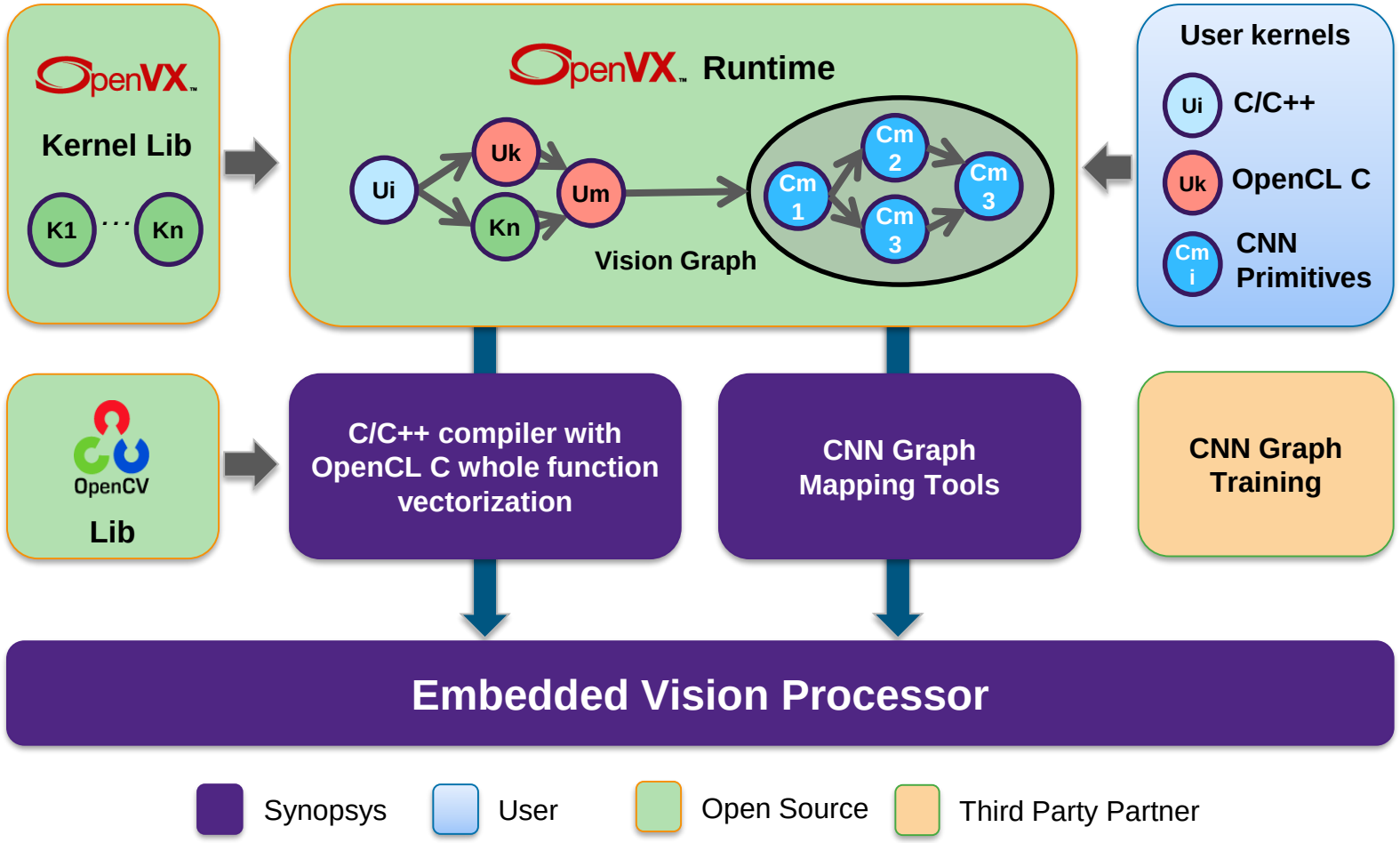


- Embedded vision algorithms
  - Very dynamic space
  - High levels of innovation
  - High differentiation
- High-productivity tools provide strong competitive advantage

- Leverage standards for better portability

SYNOPSYS®

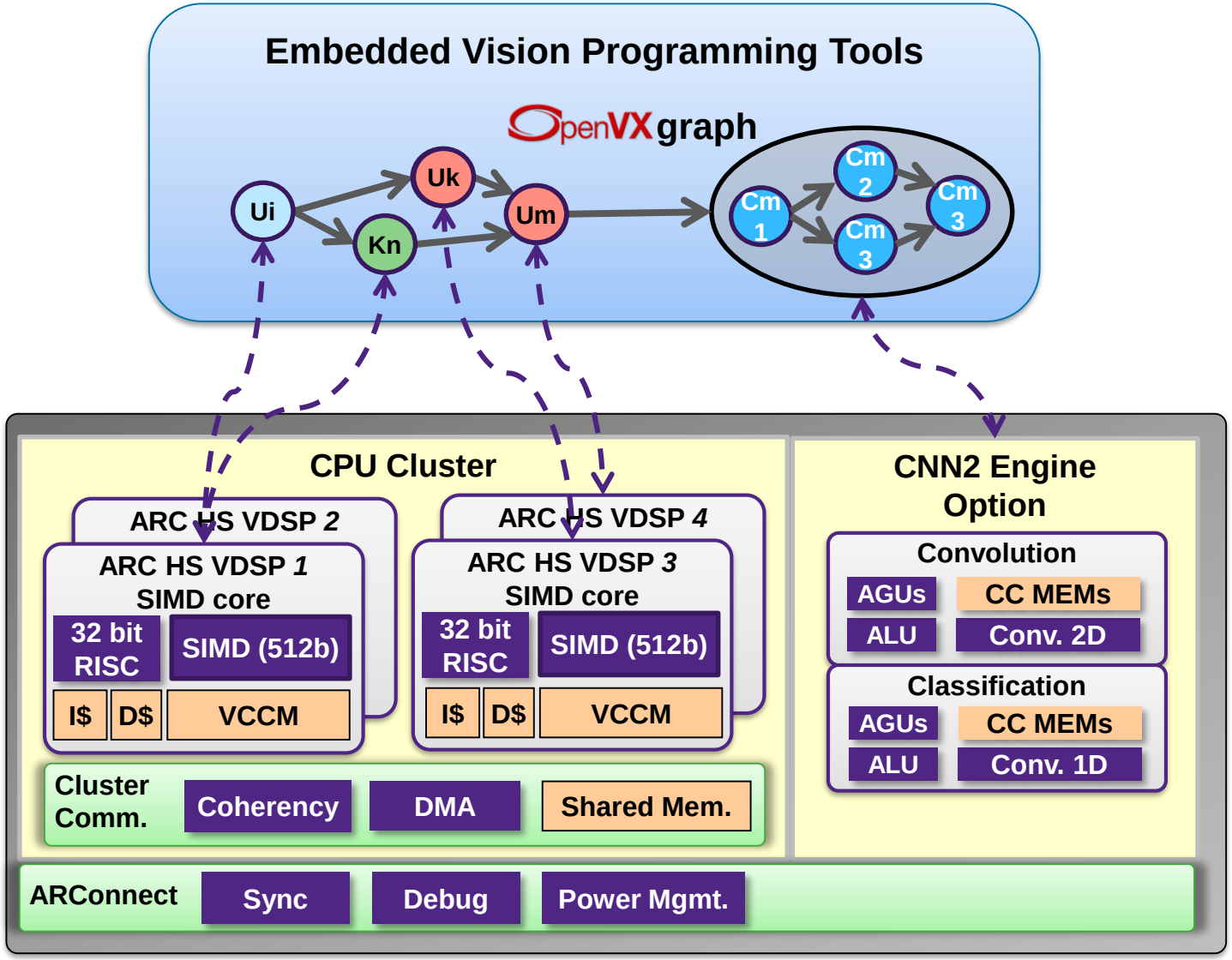
High Productivity Tools Increase Flexibility



- OpenVX eases vision graph development
- OpenCV open source library of 2500 vision algorithms helps build vision applications
- MetaWare C/C++ compiler delivers optimize program coding
- OpenCL C instructions with whole function vectorization simplifies DSP programming
- CNN graph mapping tools automate CNN programming

SYNOPSYS®

OpenVX Graph Mapping in EV Processor



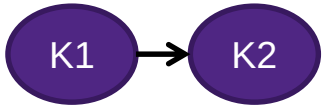
- Runtime performs OpenVX node to processor core assignment and load balancing
  - Option for user-guided assignment
  - Frame or tile-based
- Automatic insertion of communication buffers and memory allocation
  - Option for user-guided memory allocation
  - Extensible to customer H/W accelerators

SYNOPSYS®

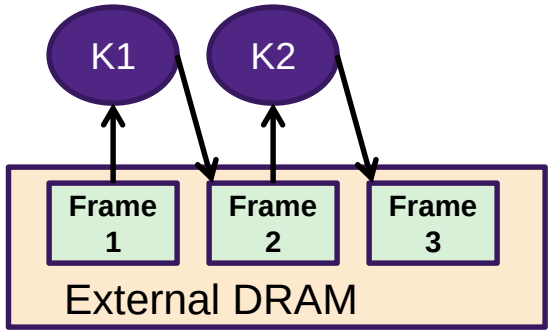
OpenVX Tiling in EV Processor

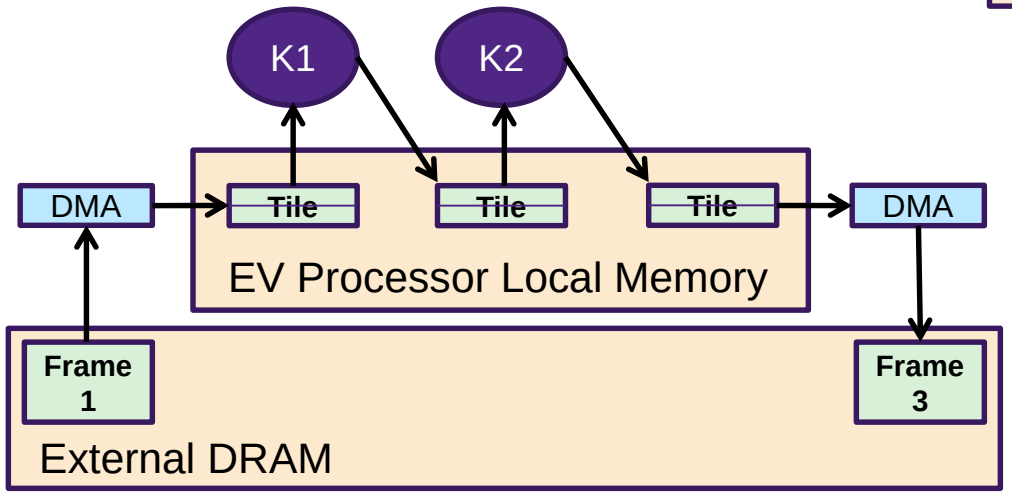
Reducing memory size and power

- Logical Model
  - Data flow between Kernels



- Classical OpenCL Kernel Implementation
  - Host-Device frame buffer movement
  - Efficiency/memory size/power issues!





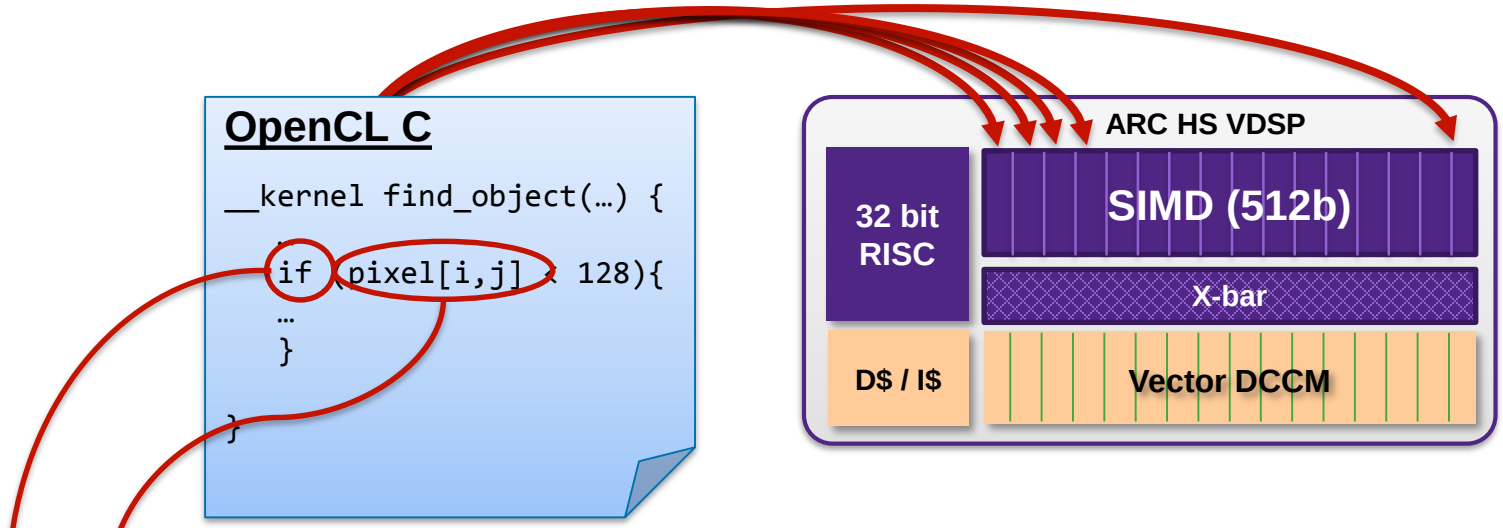
- EV Proc. tiled implementation
  - Data “tunneled” through small(er) local vector memory
  - Enhanced OpenVX/OpenCL runtime
  - Runtime calls kernels directly
  - No round-trip to host

SYNOPSYS®

OpenCL™ C Whole Function Vectorization

OpenCL 2.0, embedded profile

The compiler maps OpenCL C kernel on all the SIMD lanes



- Lanes execute the same program on different data
  - Every lane works on a different pixel, image patch, decision tree,….
- Every lane can do independent load/stores to the shared Vector DCCM with the X-bar (Scatter-Gather)
- Lane-dependent control-flow is mapped to predicated execution

Scalar, OpenCL C, and Manual Vector code: an Example

Value = high productivity for high performance code.

- Software developer to write algorithms in Scalar C code
- Standard OpenCL syntax can be used to guide the compiler in vectorizing any loop

Scalar C code

```
for (short i = 0; i < currentcount; i++)
{
  for (short j = 0; j < prevcount; j++) {
    uchar dist = 0;
    for (short k = 0; k < 8; k++) {
       dist += popcount( current[i][k] ^ prev[j][k]);
    }
    if (dist < min_dist) {
      best_match2 = best_match;
      min_dist2 = min_dist;
      best_match = j;
      min_dist = dist;
    } else if ( dist < min_dist2 ) {
      best_match2 = j;
      min_dist2 = dist;
    }
  }
  best[i] = best_match;
  secondbest[i] = best_match2;
}
```

1 line change to vectorize loop.

OpenCL C

```
for (short i = get_local_id(); i < currentcount; i+=16) {
  for (short j = 0; j < prevcount; j++) {
    uchar dist = 0;
    for (short k = 0; k < 8; k++) {
       dist += popcount( current[i][k] ^ prev[j][k]);
    }
    if (dist < min_dist) {
      best_match2 = best_match;
      min_dist2 = min_dist;
      best_match = j;
      min_dist = dist;
    } else if ( dist < min_dist2 ) {
      best_match2 = j;
      min_dist2 = dist;
    }
  }
  best[i] = best_match;
  secondbest[i] = best_match2;
}
```

Vectorize one loop by changing indexing

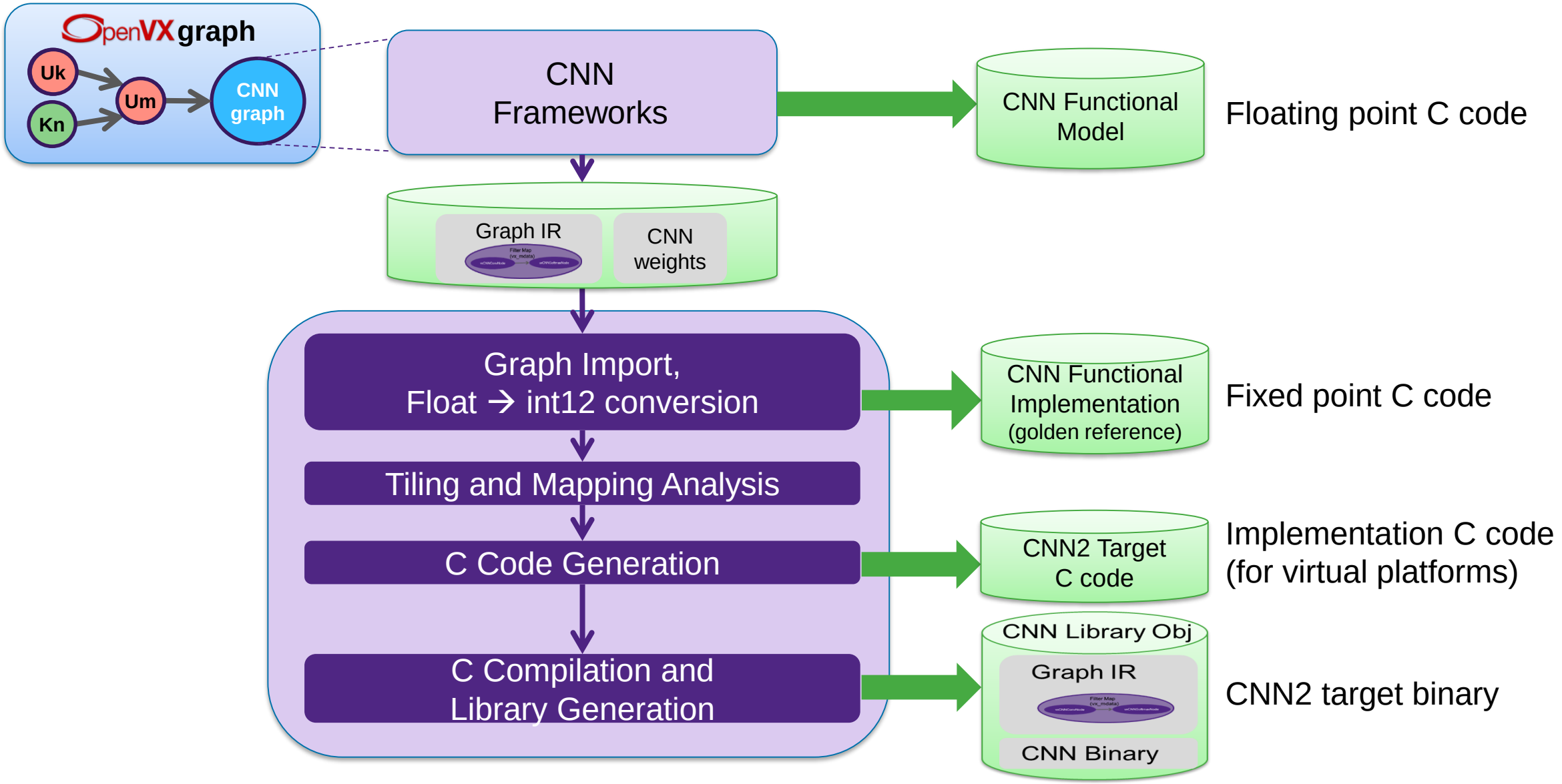
- Change datatypes to vector types for some variables
- Use intrinsics for load/stores
- Manually convert simple control flow into vector expressions
- Explicitly deal with vectorized loop boundaries

Manually Vectorized Code

```
short32 id_32 = (short32)(0, 1, 2, 3, 4, 5, 6, 7, 8,
  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
  22, 23, 24, 25, 26, 27, 28, 29, 30, 31);

  for (ushort i = 0; i < currentcount; i+=32) {
    short32 offsets = (i + id_32) * 16;
    ushort32 vcurrent[16];
    ushort32 mask = offsets < currentcount;
    for (ushort k = 0; k < 16; k++) {
      vcurrent[k] = vgather32(mask, current, offsets++);
    }
    for (ushort j = 0; j < prevcount; j++) {
      ushort32 dist = 0;
      for (ushort k = 0; k < 16; k++) {
        dist += popcount(vcurrent[k] ^ prev[j][k]);
      }
      short32 found_best_match  = dist < min_dist;
      short32 found_best_match2 = dist < min_dist2;
      found_best_match2 &= !found_best_match;
      best_match2 =
        found_best_match? best_match :
        found_best_match2? j    : best_match2;
      min_dist2   =
        found_best_match? min_dist   :
        found_best_match2? dist : min_dist2;
      best_match  = found_best_match? j    : best_match;
      min_dist    = found_best_match? dist : min_dist;
    }
    vstore32(mask, best_match, 0, &best[i]);
    vstore32(mask, best_match2, 0, &secondbest[i]);
  }
}
```

CNN Programming Tools



SYNOPSYS®

Conclusions

- High productivity is essential in dynamic vision market
- OpenVX becoming de facto standard for high-level vision applications
  - Leveraged in EV Programming Tools for automation of load balancing, data communication and synchronization
- OpenCL C
  - Offers right semantics for effective use of data level parallelism
  - Enables powerful whole function auto-vectorization
- CNN Programming
  - Optimized code generation on CNN engine from high-level CNN graph description

SYNOPSYS®
Silicon to Software™

Thank You