Multi/Many-core Processing Perspective for Embedded Computer Vision

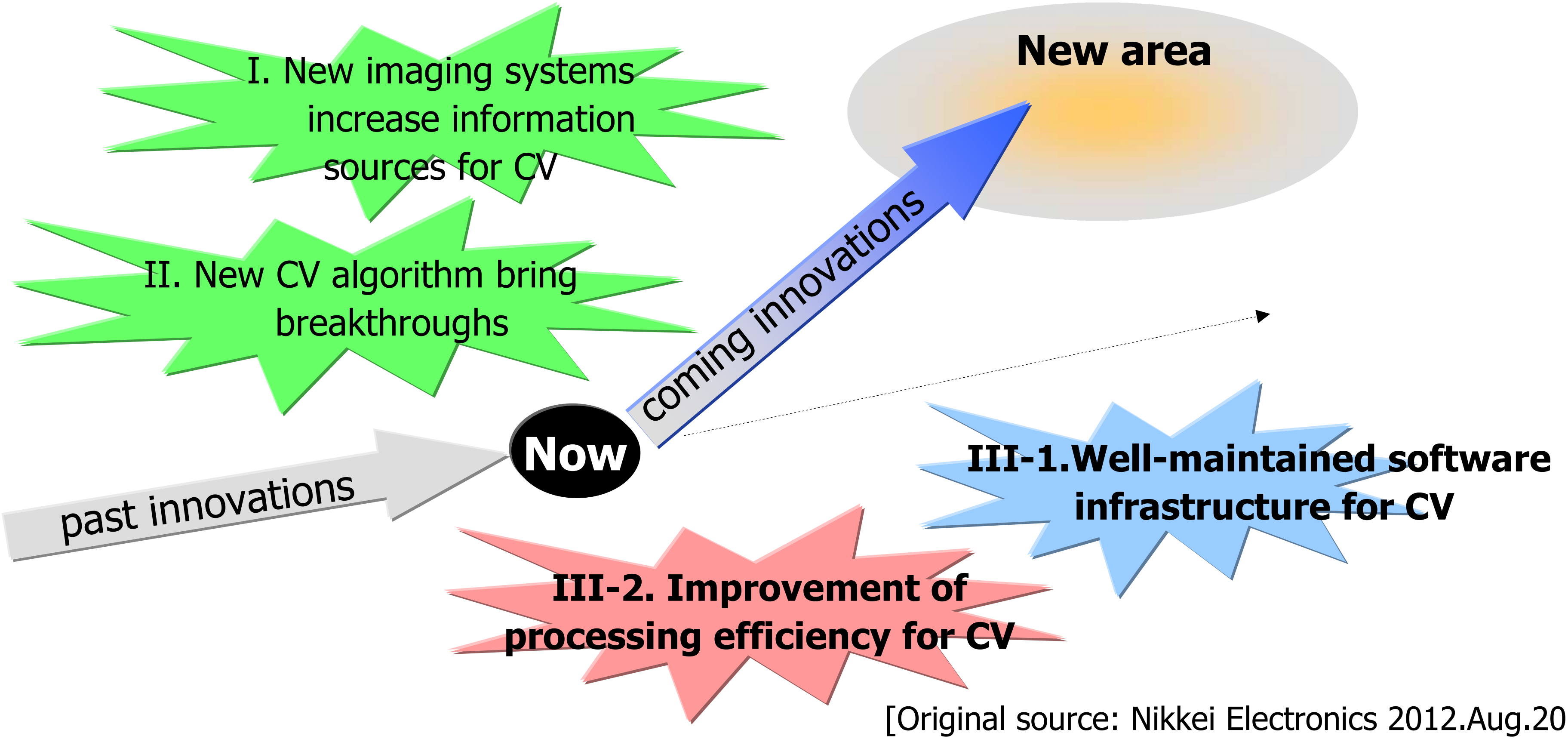
Advanced LSI Systems Research
New Core Technology Development Division
Renesas Electronics Corporation
Shorin Kyo

Outline

- Background
  - Anytime, everywhere CV apps
  - Issues to be addressed for CV HW and SW
- Hardware Architecture for CV
  - Approaches for performance and power
  - Case studies
- Software Framework for CV
  - Approaches for performance portability
  - Toward an open, efficient, and portable CV platform
- Conclusions

Coming Leap of CV Technologies

I. New imaging systems
II. Breakthrough CV algorithms
III. Computation hardware and software innovations



New area
I. New imaging systems increase information sources for CV
II. New CV algorithm bring breakthroughs
coming innovations
Now
past innovations
III-1.Well-maintained software infrastructure for CV
III-2. Improvement of processing efficiency for CV

[Original source: Nikkei Electronics 2012.Aug.20]

Coming "Anytime and Everywhere CV" Era

New imaging systems

Breakthrough CV algorithms

Performant & low-cost CV HW

Open & portable CV SW

Low cost distance image cature camera

Multi focus camera

Camera embedded lighting

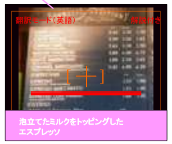
Mobile phone UI

Augmented reality

Lane Departure Warning

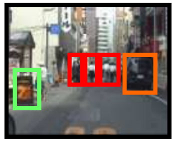
Home Appliance UI

Interpreting Camera



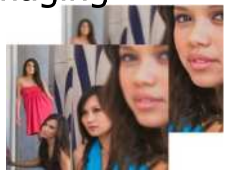
Natural UI

Collision Warning



Surveillance

Computational Imaging



Issues to be Addressed for CV HW

- Need performant and low-cost CV HW
  - **Low-cost (low-area and low-power) devices are in general not performant**
    - No general way to avoid this trade-off
    - Way to address the issue under the context of CV ?

Issues to be Addressed for CV SW

- Toward Open And Portable CV
  - "*Write once and* ***run*** *everywhere*" is not sufficient

  - Goal is to be "***write once and performant everywhere***"
    - Most CV applications are seriously deadline conscious
      - ADAS, AR, UI, ...

    - Most CV accelerators will be multi/many-cores
      - Different memory system and diverse granularities of parallelism
        - Difficult to achieve performance portability

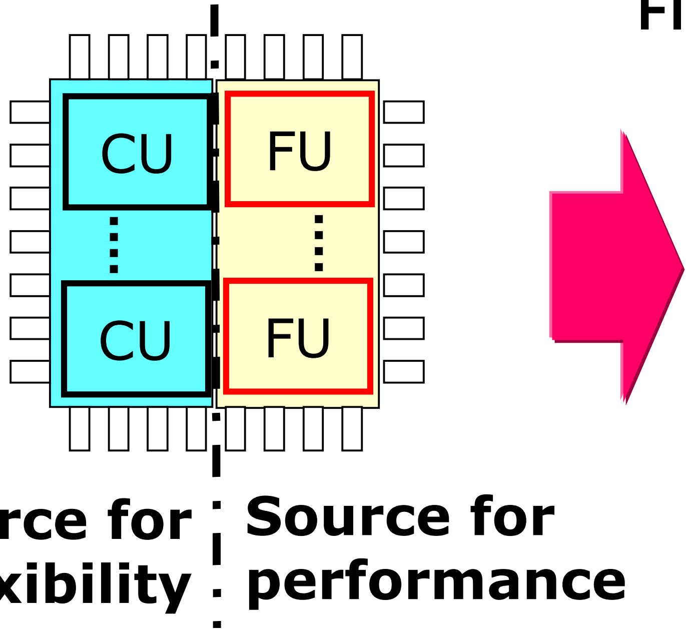
    - Way to address these issues under the context of CV ?

Outline

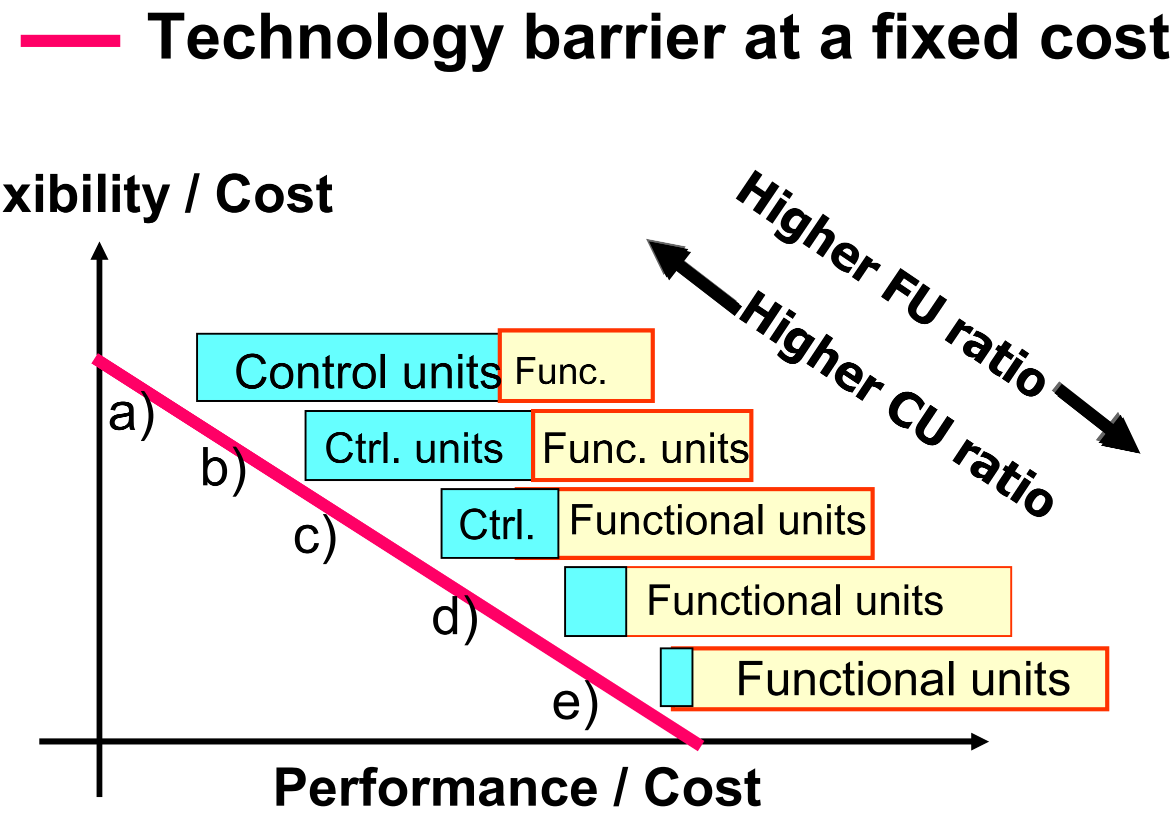
- Background
  - Anytime, everywhere CV apps
  - Issues to be addressed for CV HW and SW
- Hardware Architecture for CV
  - Approaches for performance and power
  - Case studies
- Software Framework for CV
  - Approaches for performance portability
  - Toward “open, efficient, and portable”
- Conclusions

Inevitable Trade-offs in Processor Design

The Limited die space



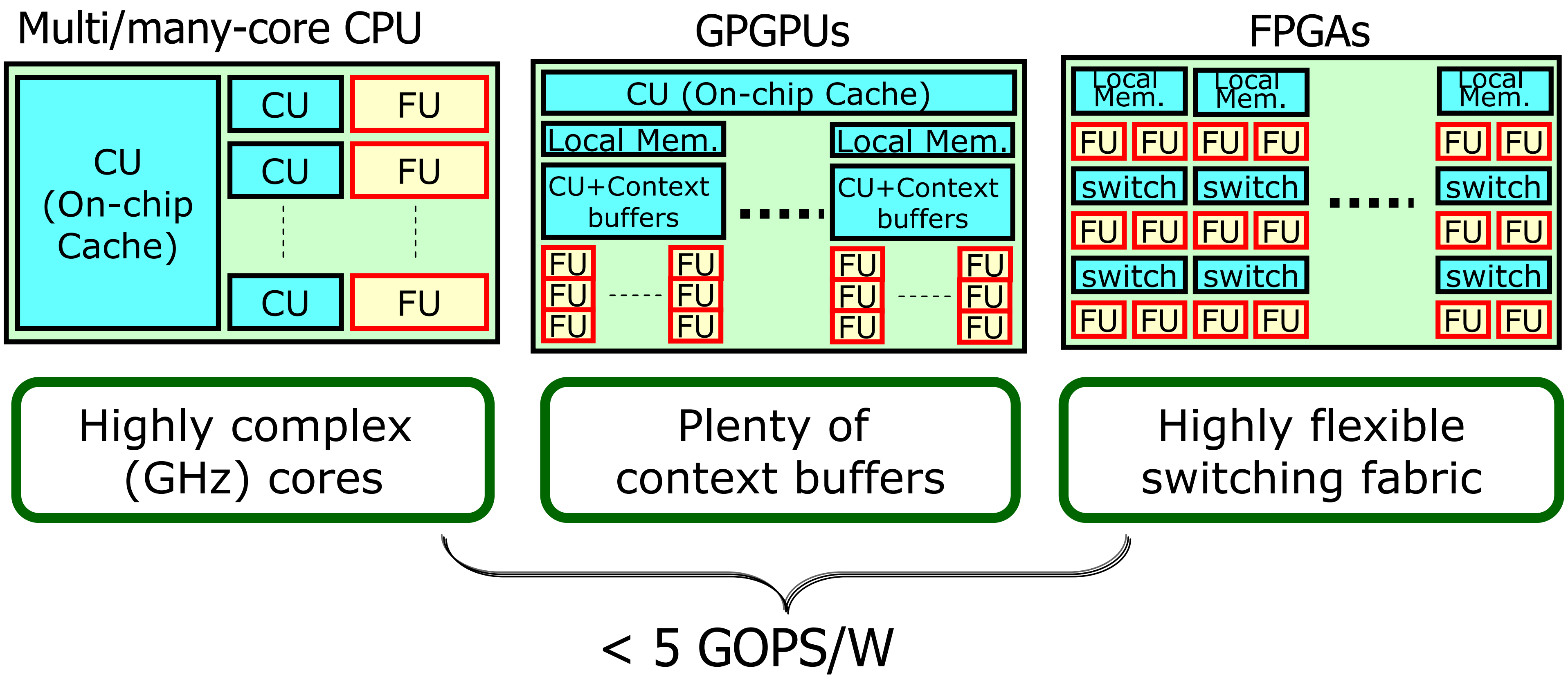
FU : Function Unit
CU : Control Unit



(a) General Purpose Processors
(b) DSPs, MIMDs. FPGAs, GPGPU
(c) Highly parallel SIMDs
(d) Wired logics+DSP core
(e) Wired logics only

State-of-the-Art Design Choices

- Often results in ***high CU ratio*** configurations
  - Not an adequate starting-point for designing low-cost CV accelerators



Multi/many-core CPU

GPGPUs

FPGAs

Highly complex (GHz) cores

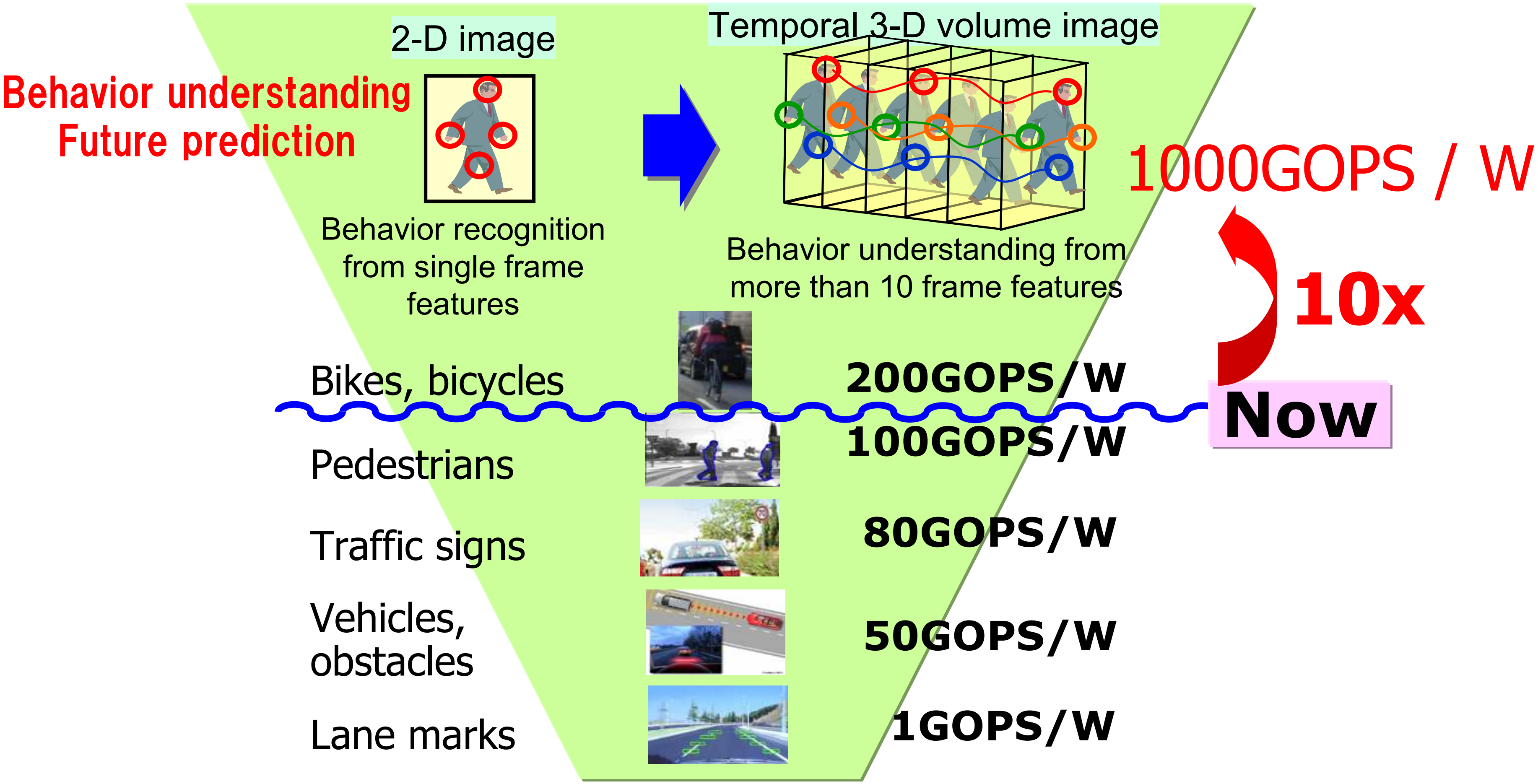
Plenty of context buffers

Highly flexible switching fabric

< 5 GOPS/W

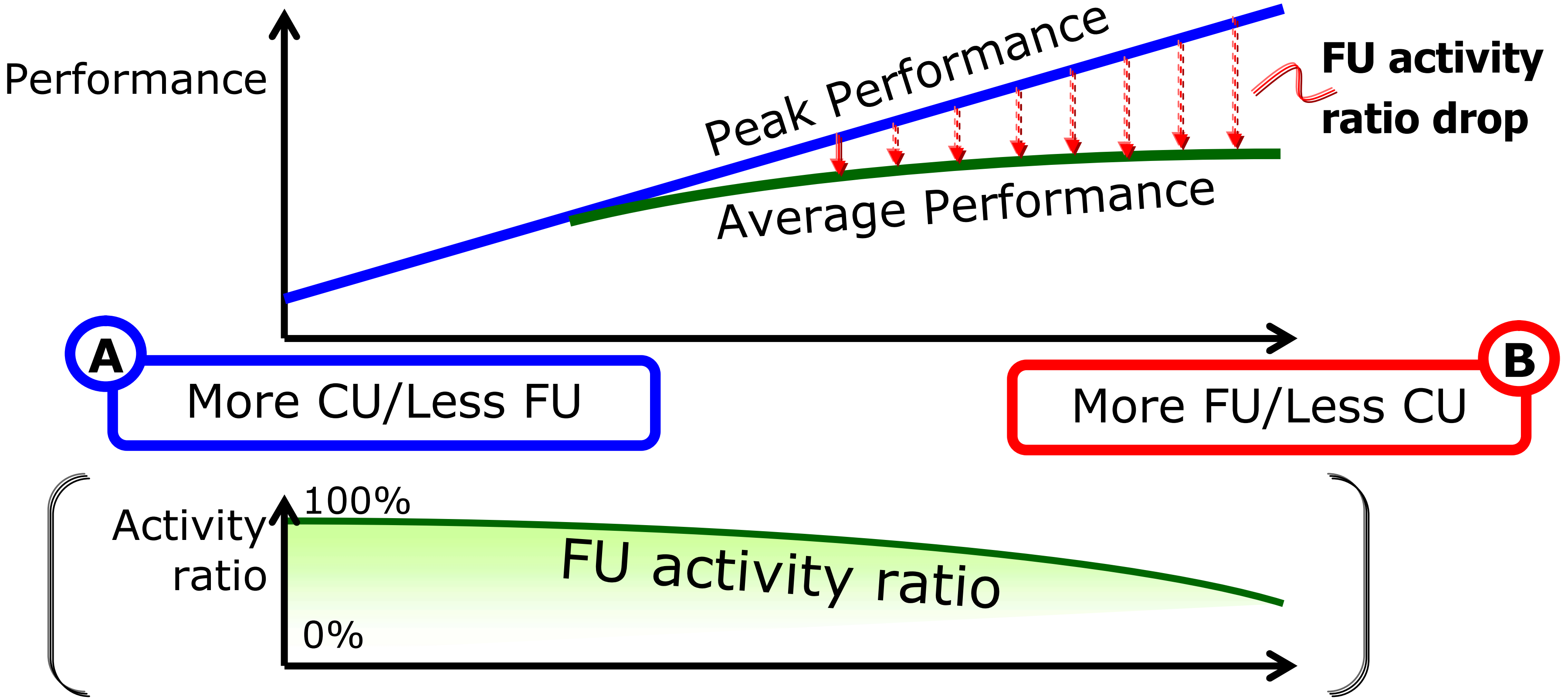
CV Performance Requirement Perspective

- Case study of ADAS: 1 TOPS/W is required in very near future



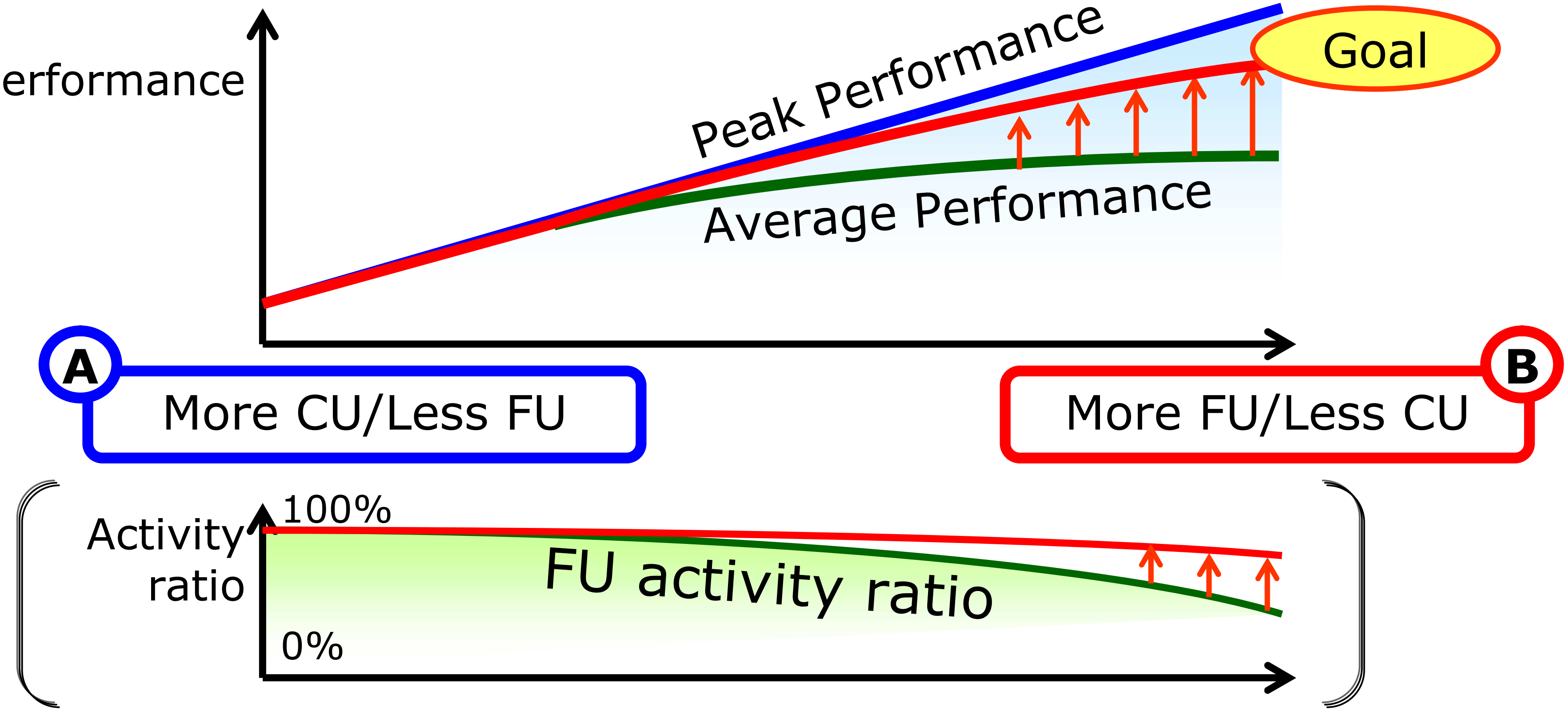
The Inevitable Design Choice

- How to achieve the ***performance per power*** goal ?
  - Need to choose **configuration B (=More FU/Less CU)**
    - Need to find way to mitigate **FU activity ratio drop**



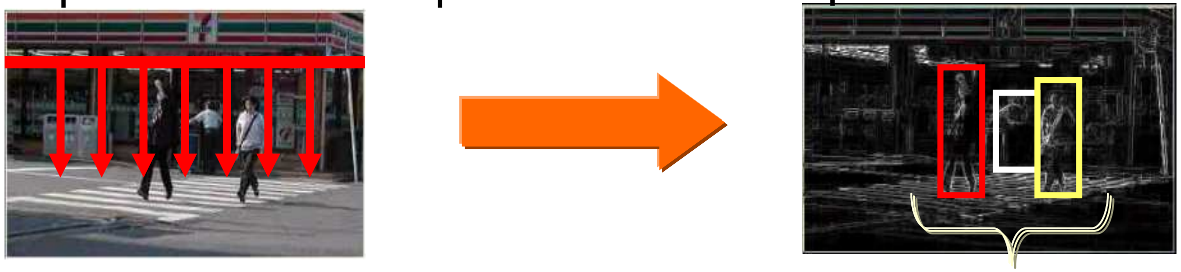
Design Goal for CV Accelerators

- Under configuration B, find a **low-cost** while **smart CU design** which can maximize FU activity ratio
  - Use of **CV domain specific knowledges** will be the key



CV Domain Specific Knowledges

- Two-step operation



Step 1. Detection step

Step 2. Verification step

ROI: Region of Interest

- Parallelism and control
  - Extensive parallelism at both (pixel)-data and ROI level
  - Identical operations against all targets (Step 1:**pixel**, Step 2:**ROI**)
    - ☞**SIMD is the typical low-cost deisgn choice**
- Asynchronous termination of verification processes
  - E.g. Boosting: frequently used since around 2008
    - ☞**May need SPMD control**

CV Domain Specific Knowledges - Cont.

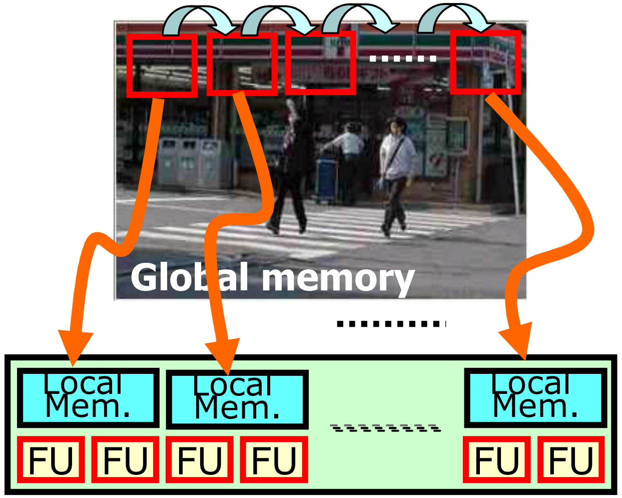
- Memory access aspects

Need to cope with irregular accesses

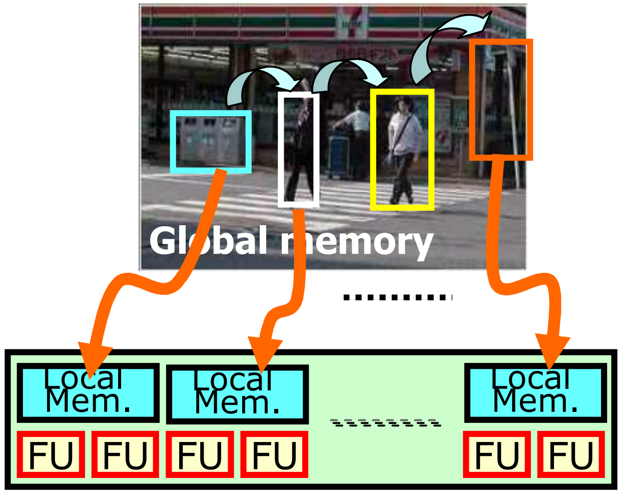
	Block wise	ROI wise	Table lookup
Access address regularity	Yes	Yes/No	No
Access address predictivity	Yes	Yes	Yes

Non-blocking DMA is a typical low-cost design choice

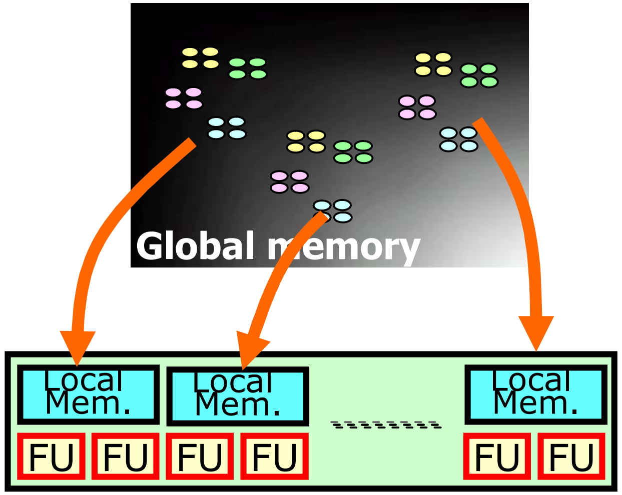
(Typical for Step 1)
Block wise



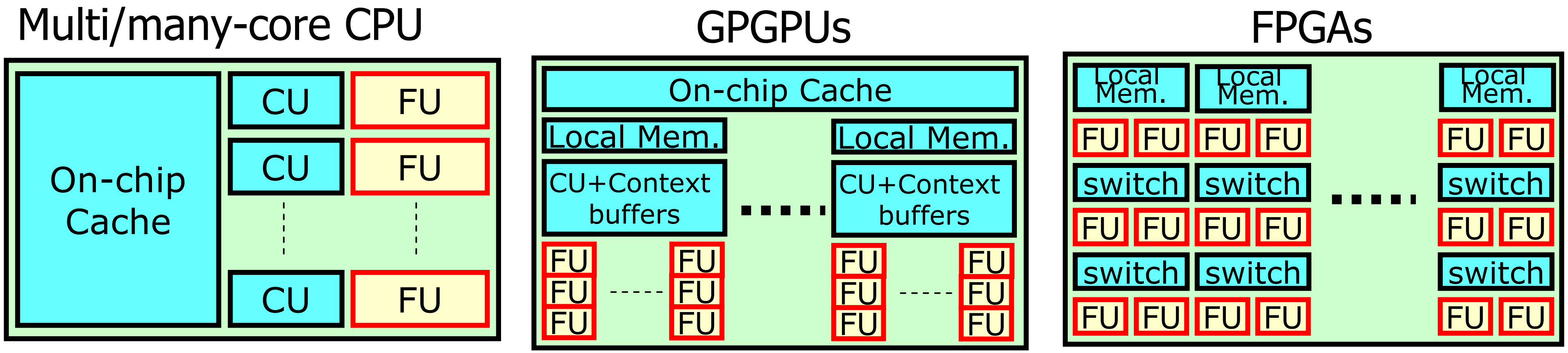
(Typical for Step 2)
ROI wise



(Common for both Step)
Dictionary or table lookup



1st Gen. Design Choice Case Study



~~Highly complex (GHz) cores~~

~~Plenty of context buffers~~

~~Highly flexible switching fabric~~

Decisions Based on CV Domain Specific Knowledges

Highly parallel SIMD

Non-blocking DMA

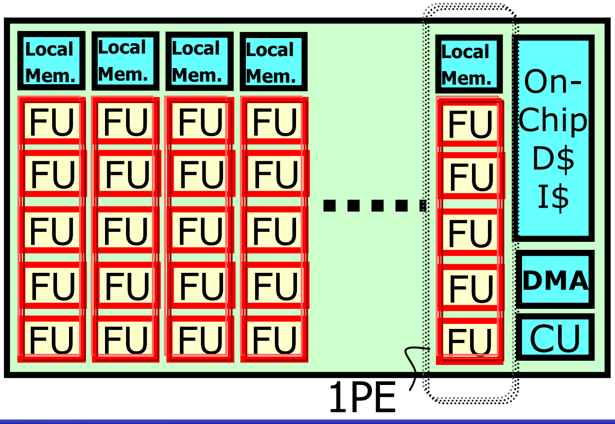
Cheapest PE network

Typical 1st Gen. CV Accelerators (- 2008)

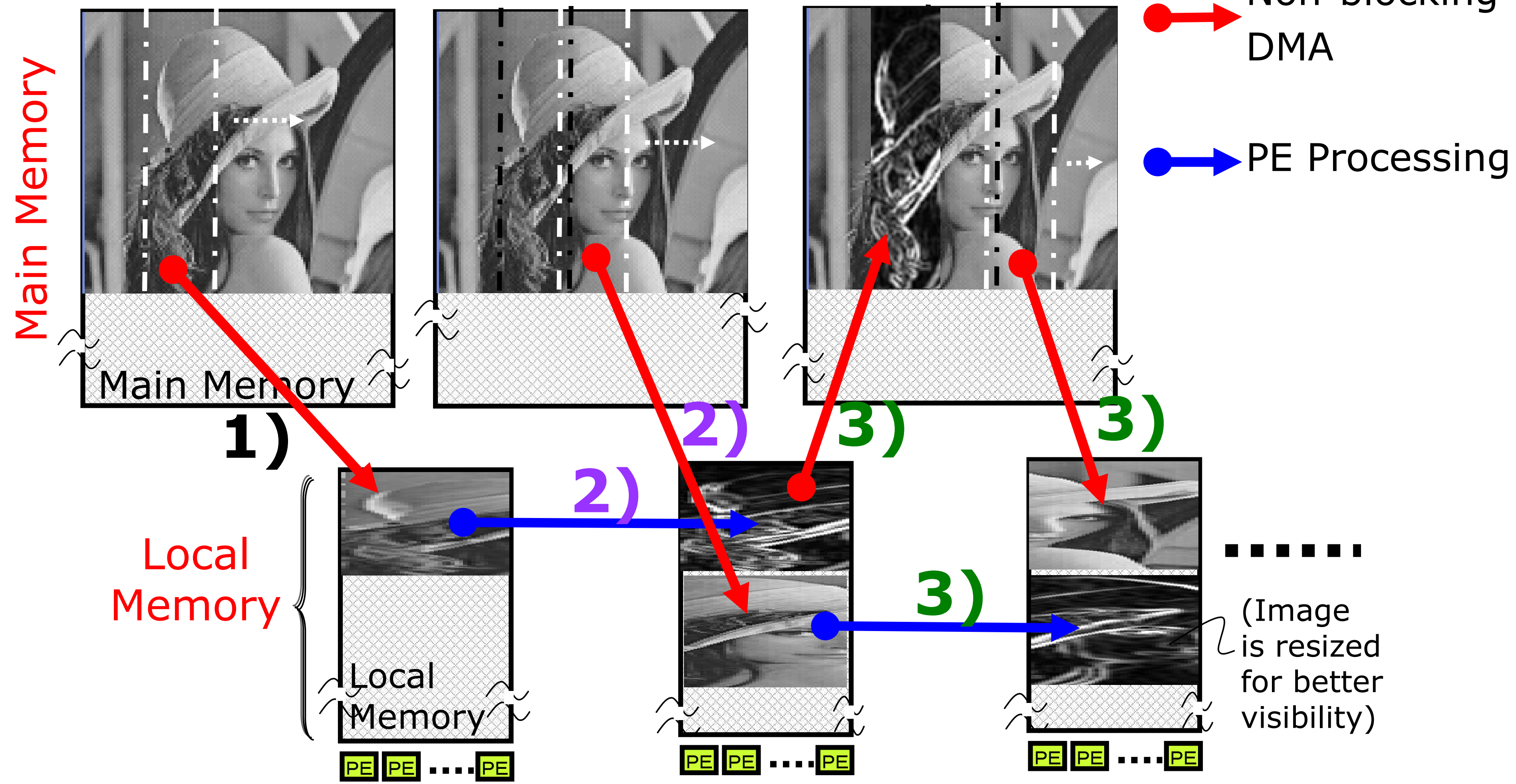
SIMD array name	Organization	#PE	bit /PE	RAM /PE	FPU	VLIW	non-blocking DMA	Multi Bank
IMAPCAR [‘08]	**NEC**	**128**	**8**	**2KB**	**-**	**X**	**X**	**X**
CSX [‘08]	ClearSpeed	96	64	6KB	X	-	X	X
Xetal II [‘07]	NXP	320	16	4KB	-	-	X	-
Ri2001	Ricoh	352	16	1KB	-	-	X	-
MX [‘06]	Renesas	2048	2	64B	-	-	X	-

- Low-cost controller techniques that used for raising the PE activity ratio
  - Non-blocking DMA
  - VLIW
  - Multi-bank local memory

IMAP-CE[’03,’05], IMAPCAR[’08]



Non-Blocking DMA Transfer



Non-Blocking DMA Transfer - Cont.

Basic idea:

w/o Non-blocking DMA

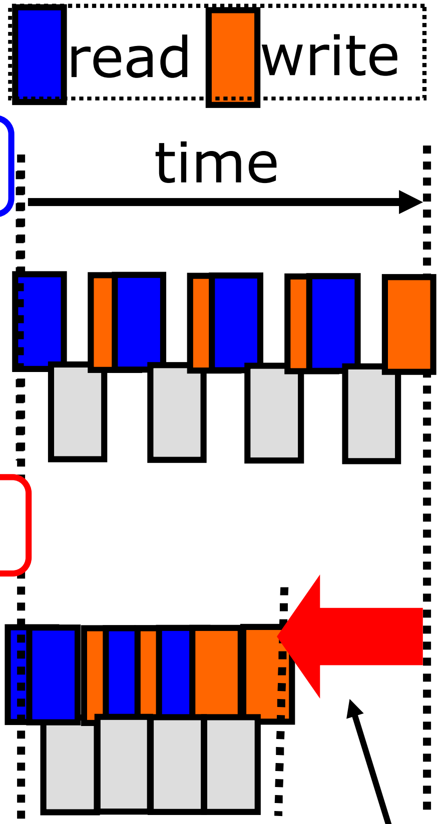
Memory transfer:

User kernel:

with Non-blocking DMA

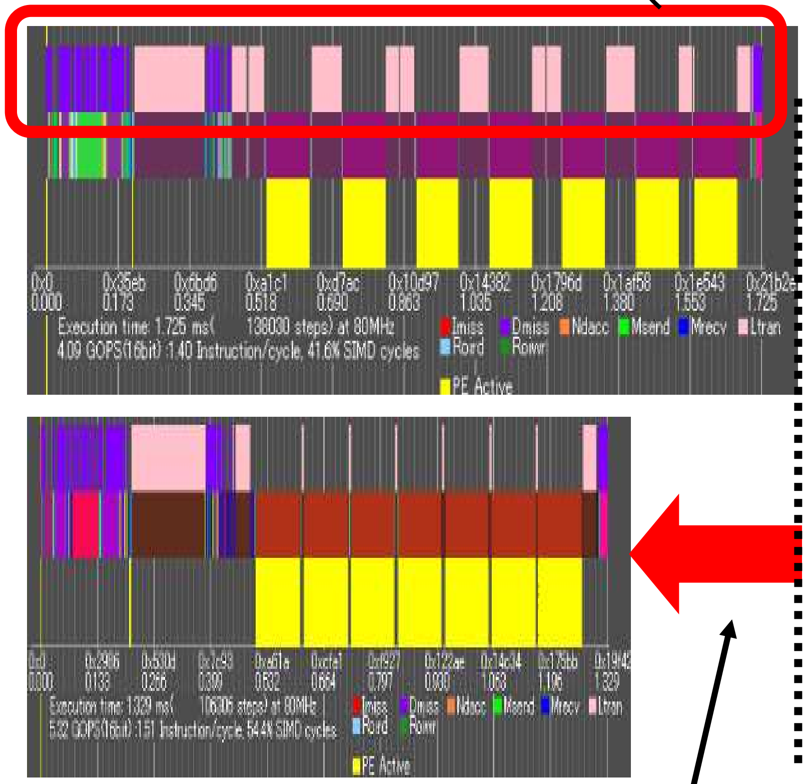
Memory transfer:

User kernel:



Profiler output snapshots:

Waiting cycles



```
Execution time: 1.725 ms(      138030 steps) at 80MHz
4.09 GOPS(16bit) 1.40 Instruction/cycle, 41.6K SIMD cycles
```

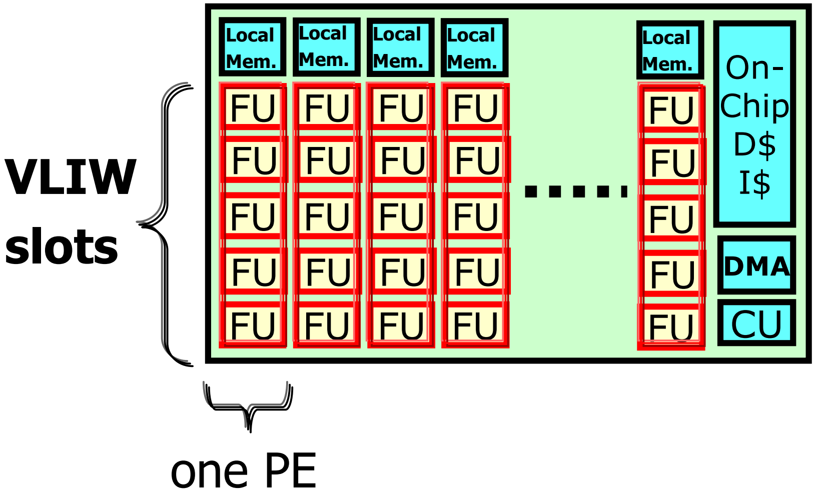
```
Execution time: 1.329 ms(      106306 steps) at 80MHz
5.32 GOPS(16bit) 1.51 Instruction/cycle, 54.4K SIMD cycles
```

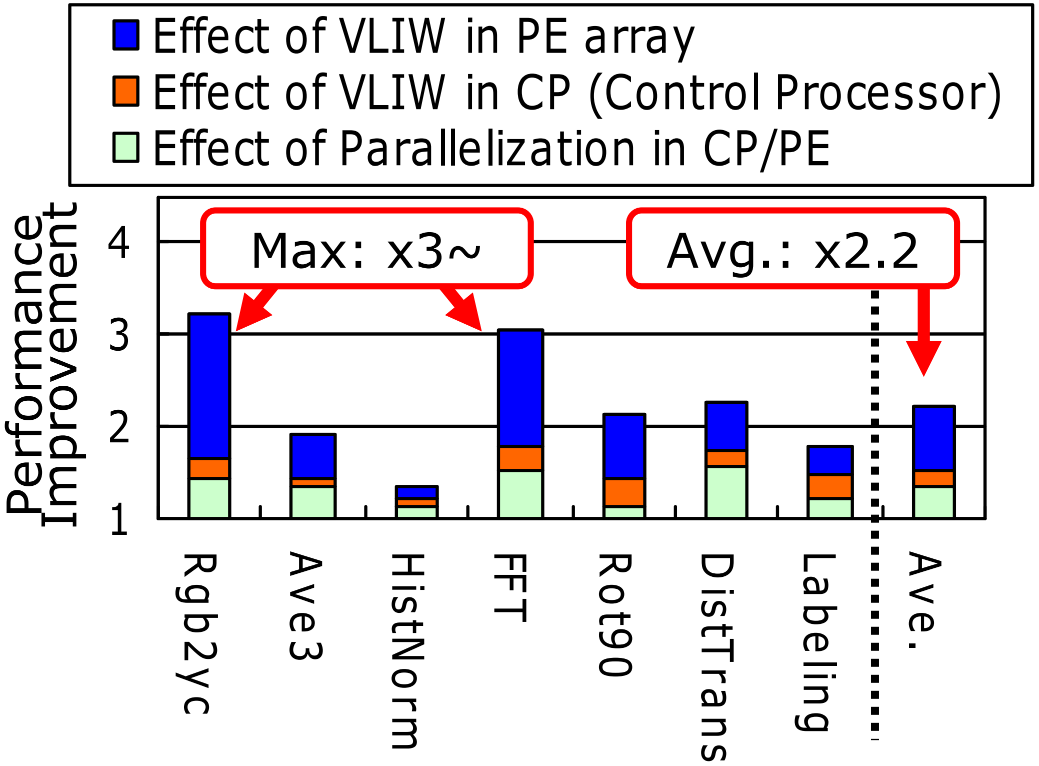
- Up to 2x performance (by reduction of waiting cycle) with area overhead <5%

(SIMD +) VLIW

- x2.2 performance gain (in average) with area overhead <13%
  - Asymmetric VLIW is adopted to reduce area overhead

IMAP-CE['03,'05], IMAPCAR['08]

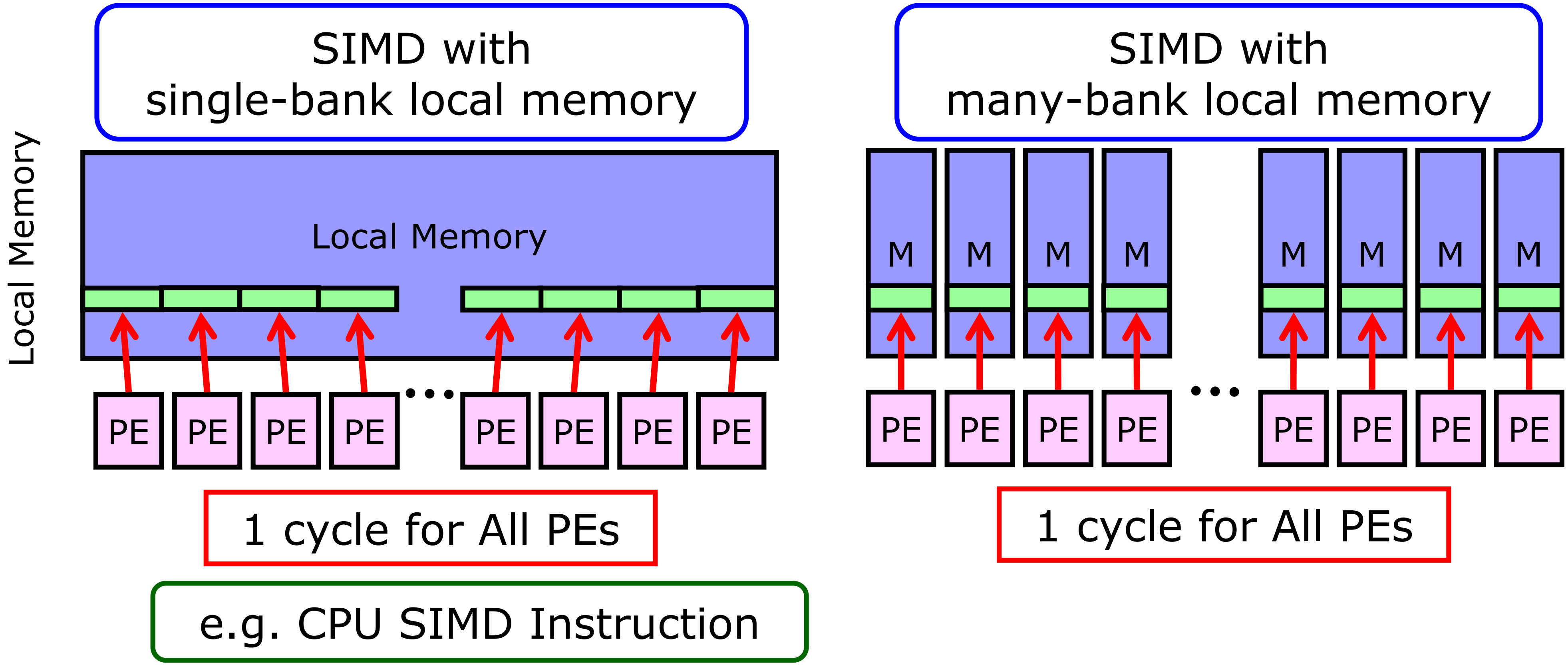




Example of performance gain by VLIW

Multi/Many-Bank Local Memory

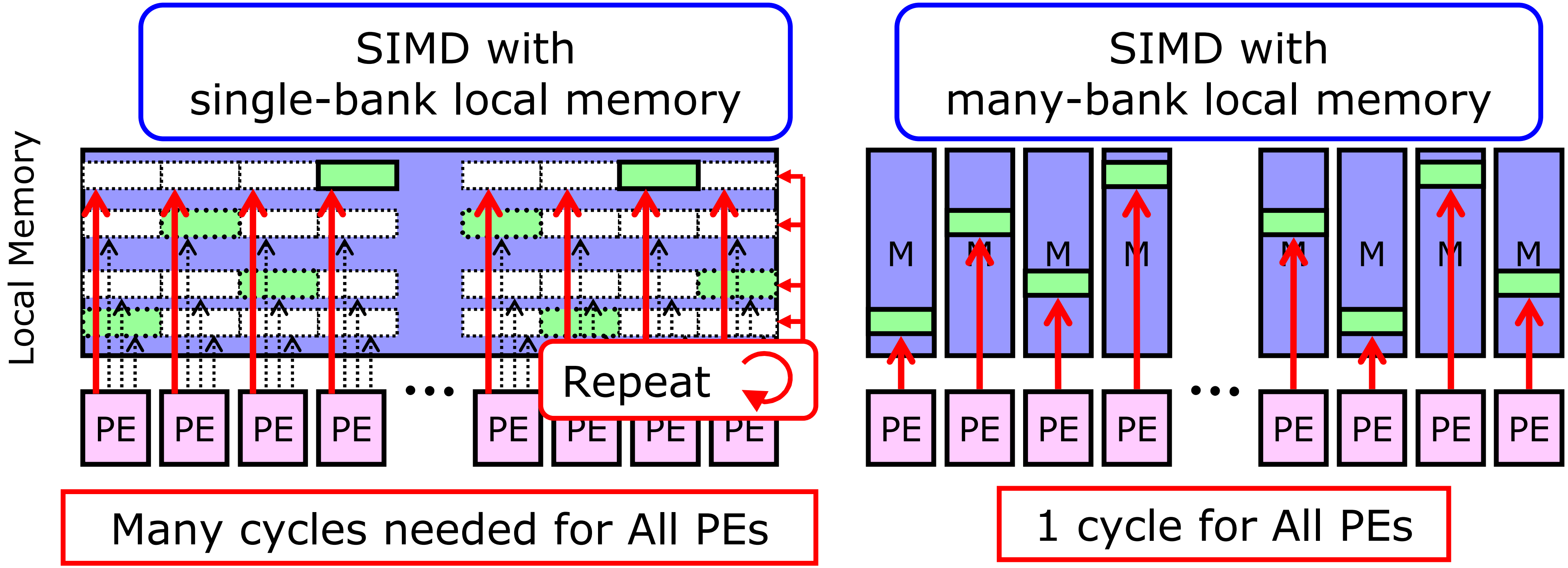
Case for uniform accesses



- Both types can access the same address for all PE

Multi/Many-Bank Local Memory - Cont.

Case for non-uniform accesses

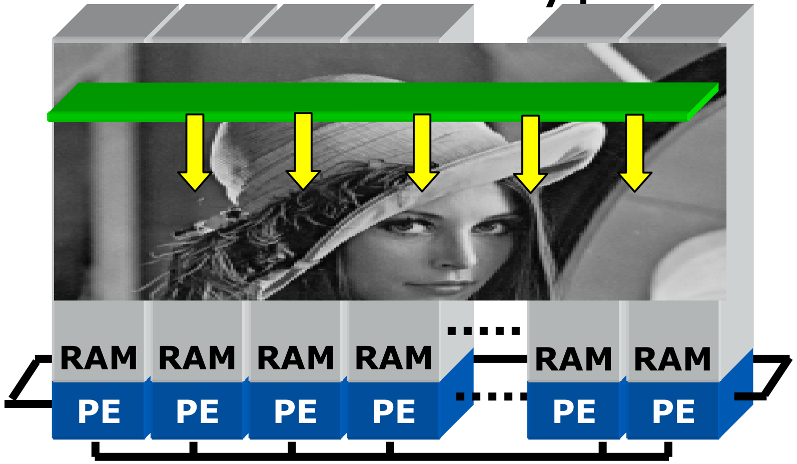


- Many-bank memory mitigates load/store bottleneck
  - Improve PE activity with lower cost
  - 10% overhead for the case of IMAP-CE & IMAPCAR (128PE) design

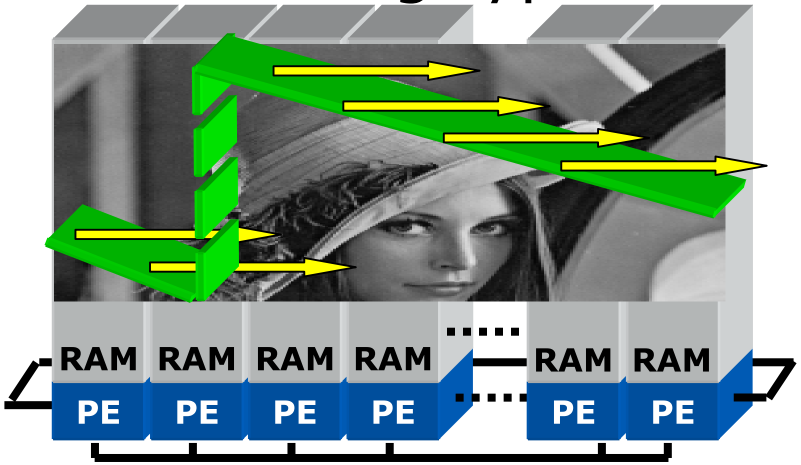
Multi-bank Local Memory Use Cases

🟩 : pixels on a line

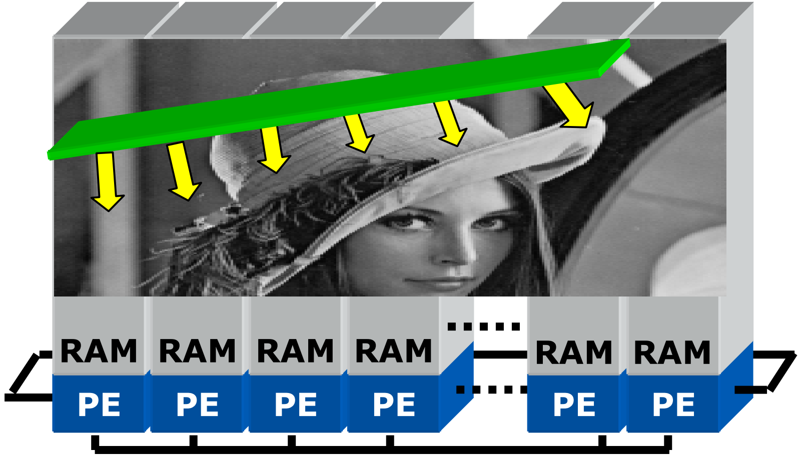
Horizontal type



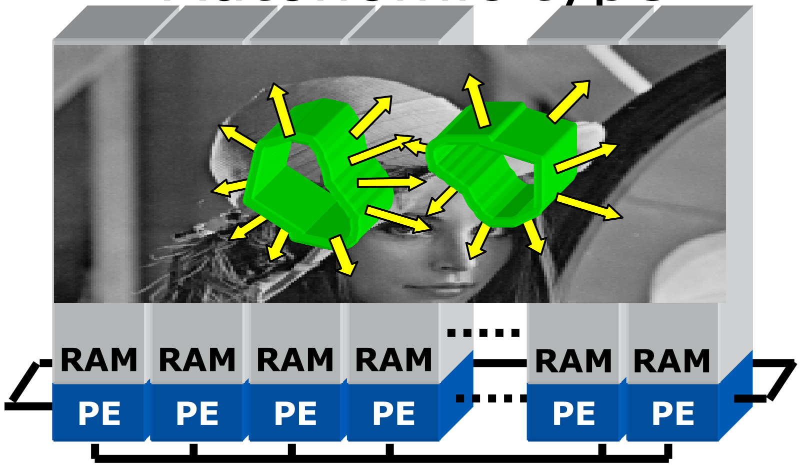
Folding type



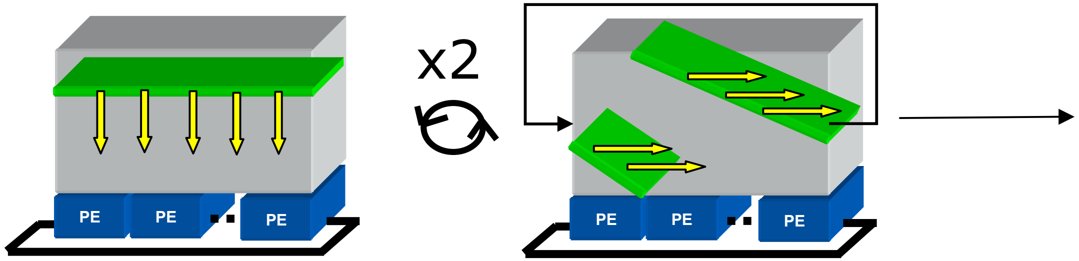
Slope type



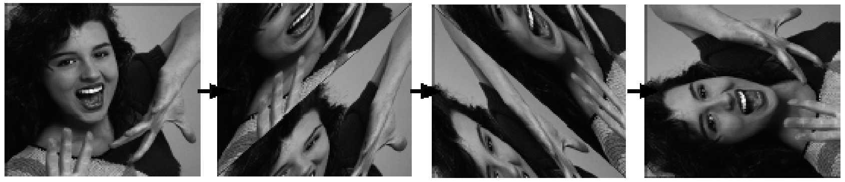
Autonomic type

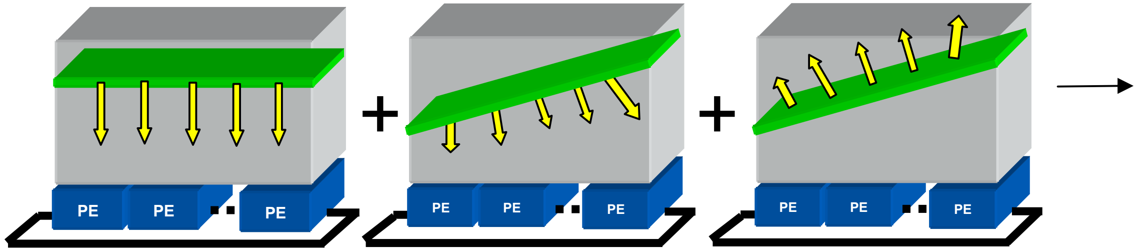


Multi-bank Local Memory Use Cases - Cont.

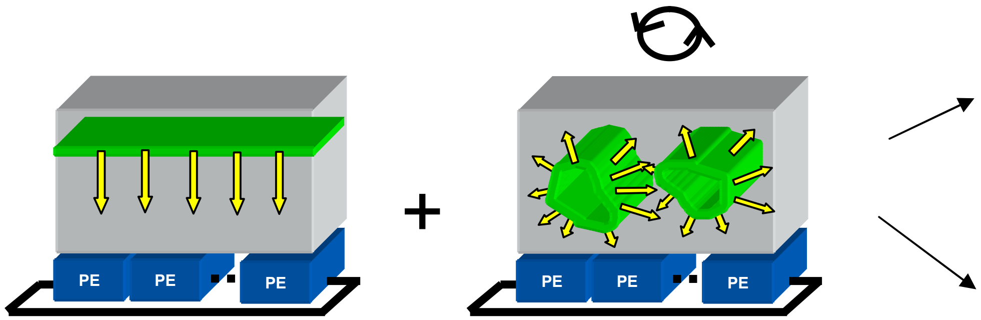


Rotation in 90 degree

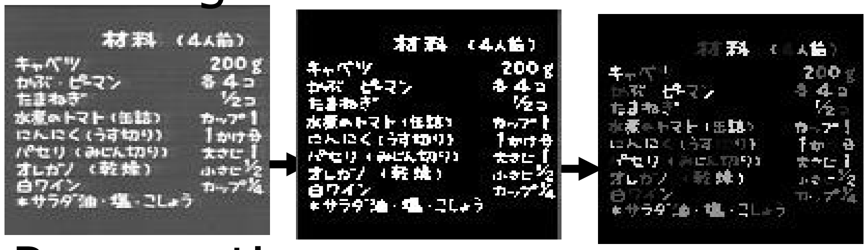




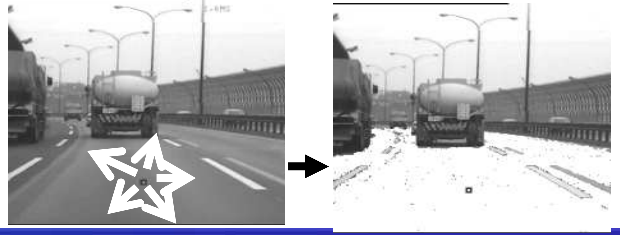
Thinning



Labeling

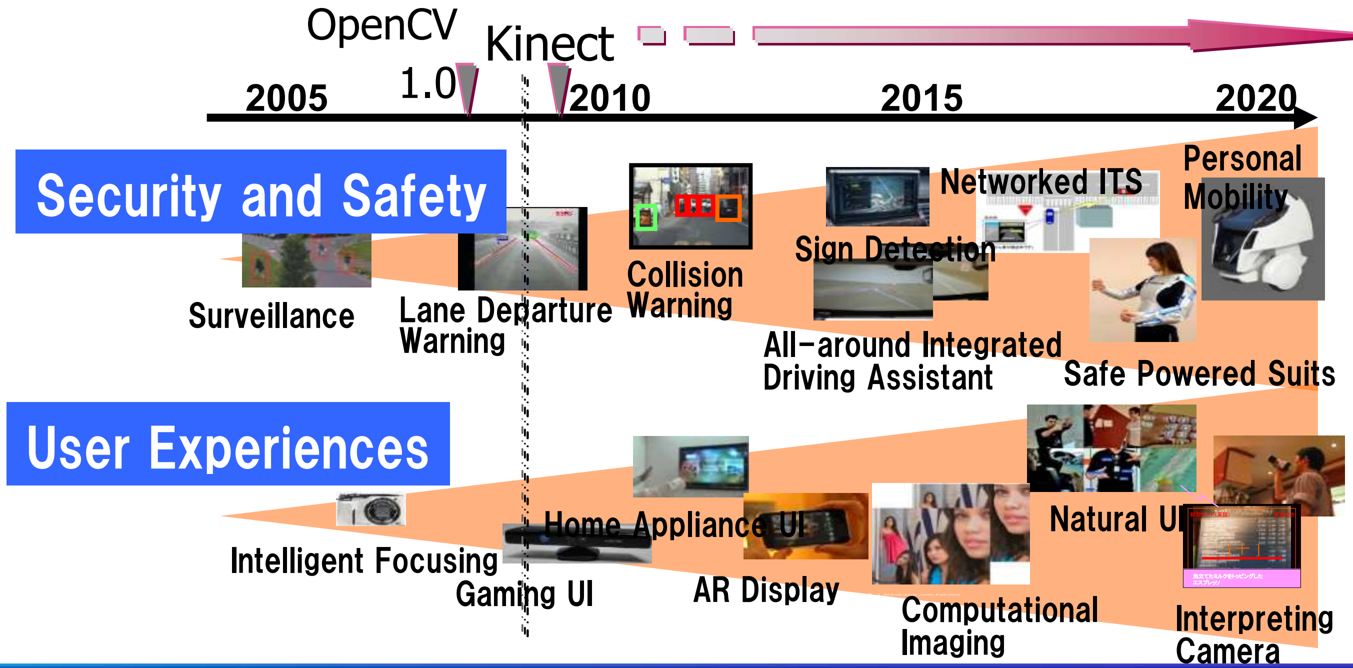


Propagation



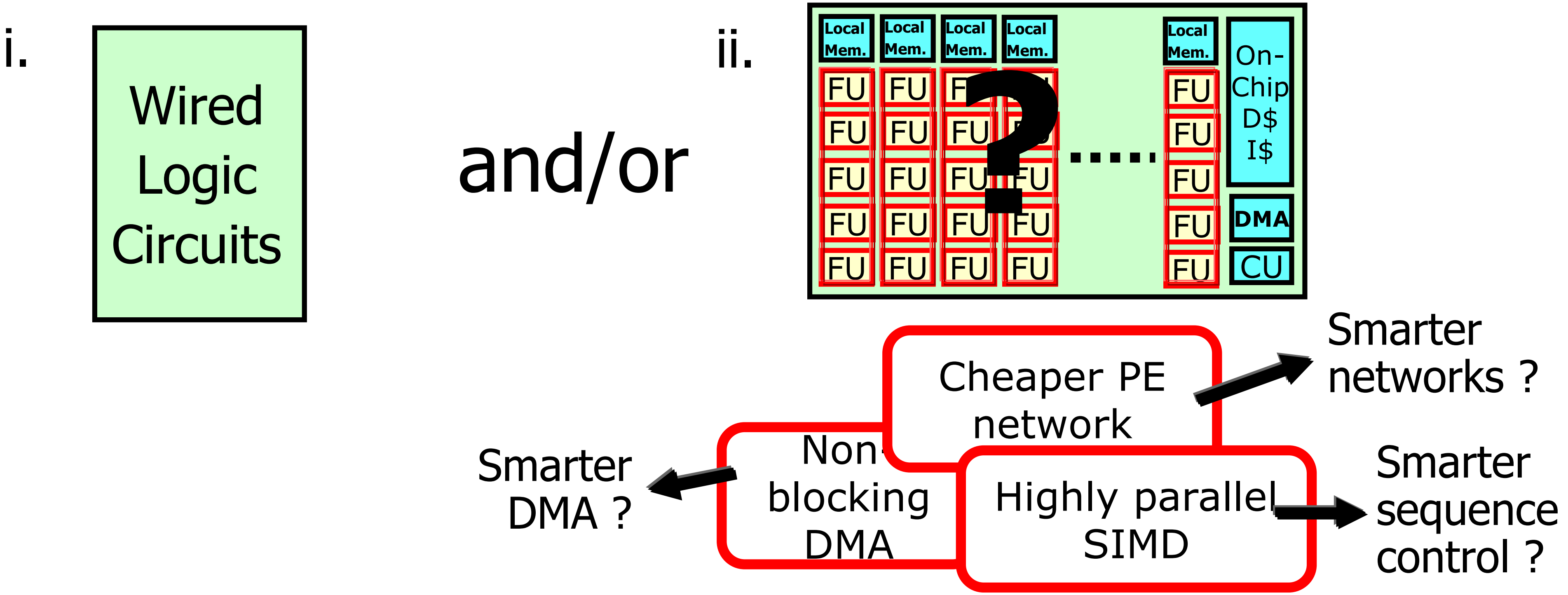
Toward 2nd Gen. CV Accelerators (approx. 2009～)

- Evolution of infrastructure, CV algorithm and applications
  - OpenCV 1.0 (2006), 2.0 (2009)
  - Application and sensor evolutions
    - ADAS, UI (Kinect), AR, computational imaging, ...
  - New CV algorithms (V&J, Boosting, HoG, SIFT, SURF, ...)
    - Need more **control and/or memory access irregularities**



2nd Gen. CV Accelerator Design Space

- Way to address the issues
  - i. Introduce purpose built circuits (i.e. wired logic circuits)
  - ii. Introduce more sophisticated controllers
  - iii. Combination of i. and ii.



i. Wired Logic Circuits

and/or

ii.

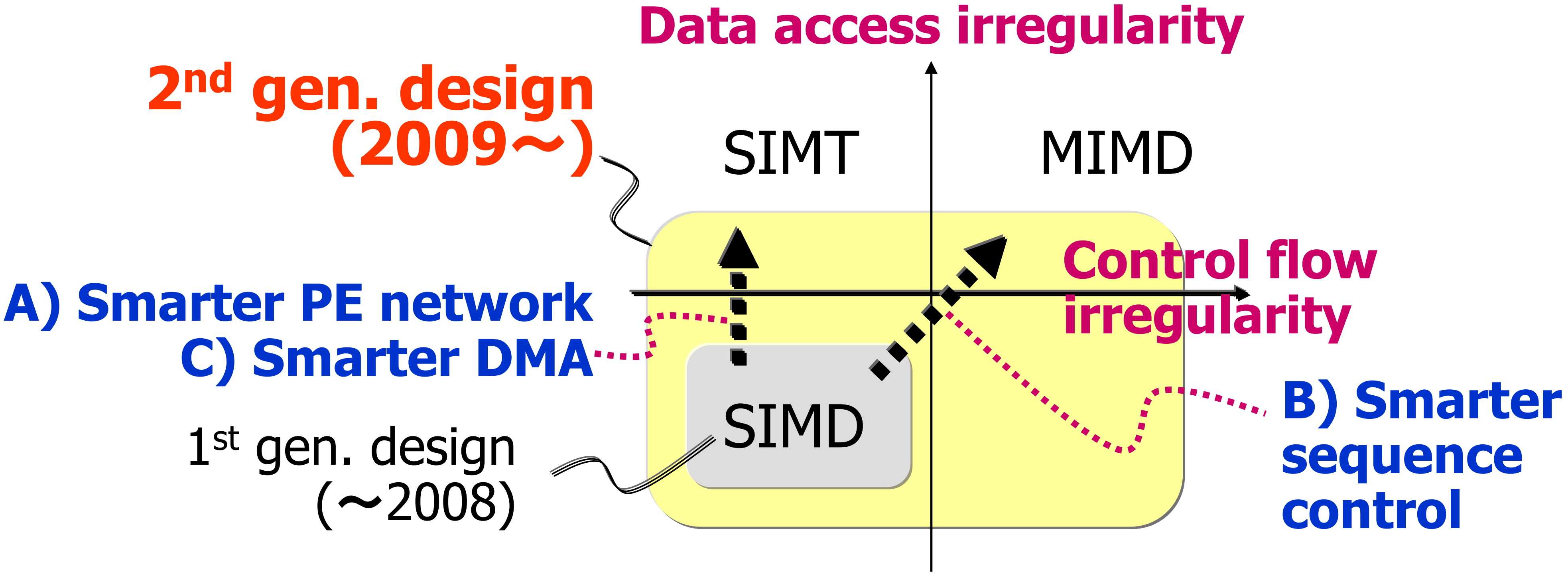
Cheaper PE network → Smarter networks ?

Non-blocking DMA → Smarter DMA ?

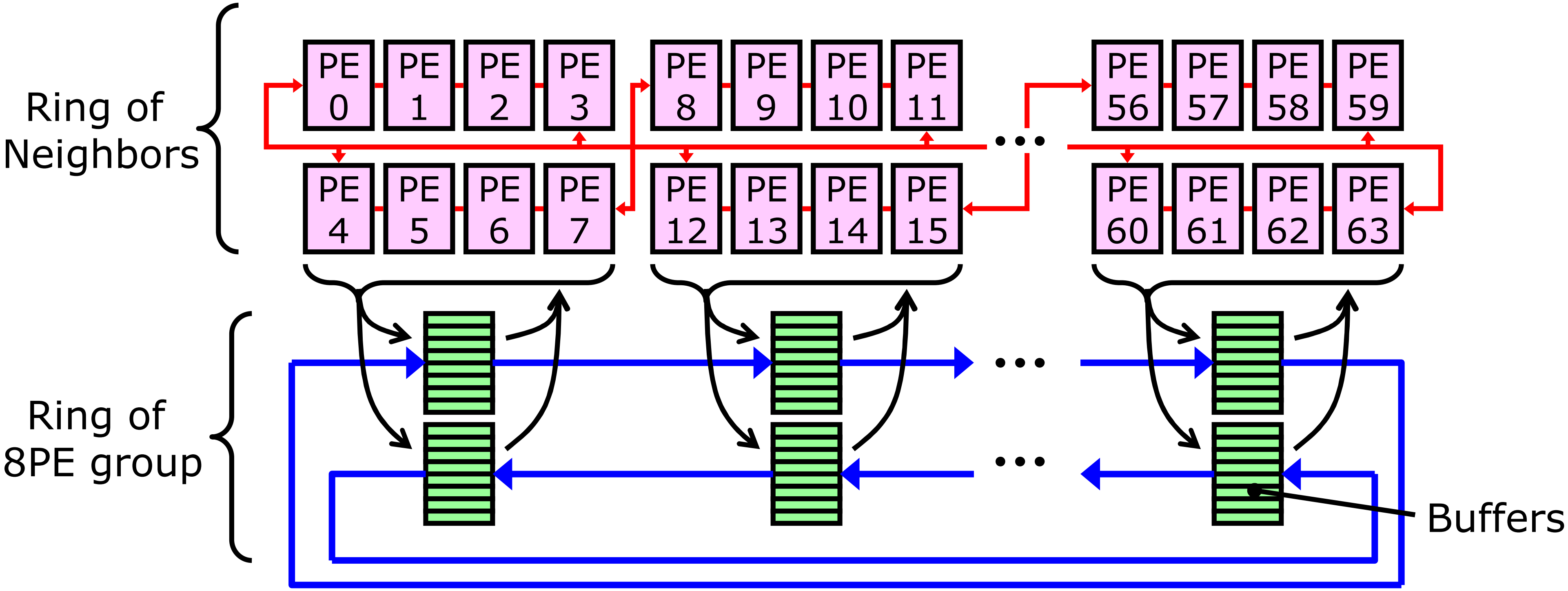
Highly parallel SIMD → Smarter sequence control ?

2nd Gen. CV Accelerator Case study

- XC series products and IP cores (IMAPCAR2 [‘09], ...)
  - Challenged the issues of highly parallel SIMD in low-cost
    - A) Smarter PE network
    - B) Smarter sequence control
    - C) Smarter DMA

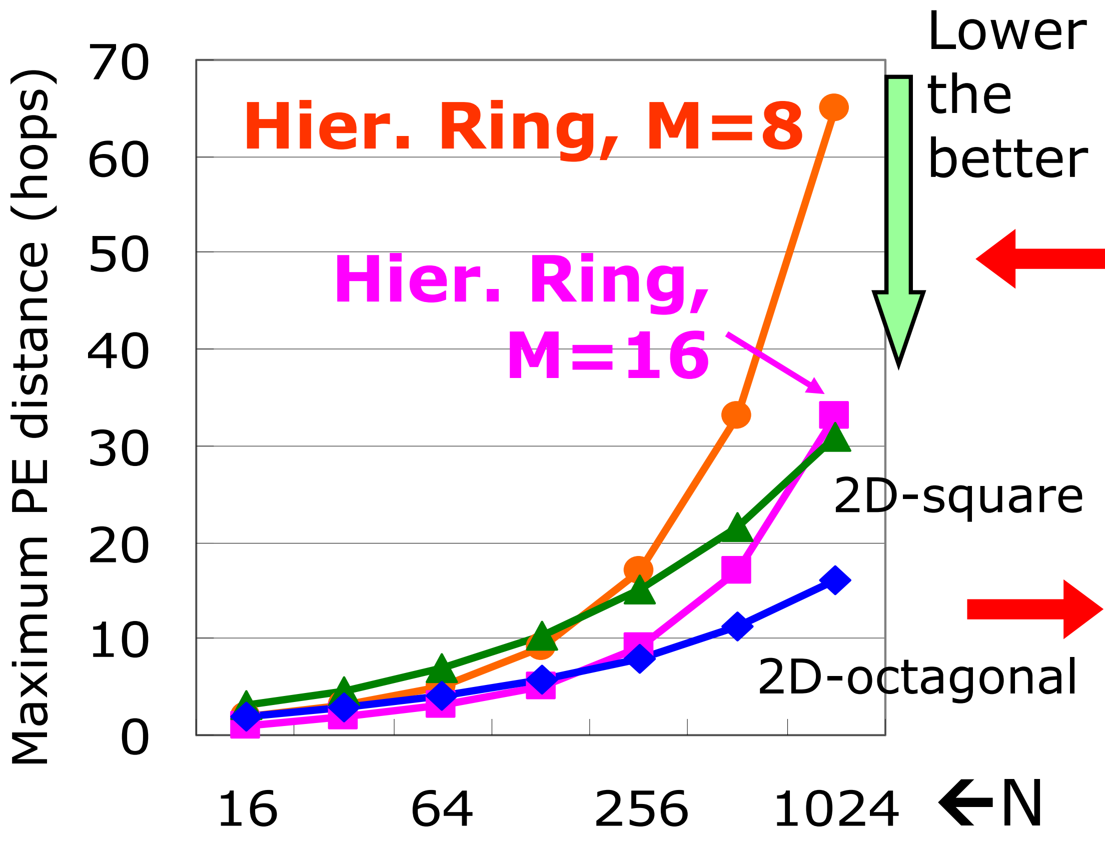


A) Smarter PE Network



- Inter-PE ring network for neighborhood
- Inter-PE hierarchical ring network
  - Data transit 8 PEs at a cycle to both direction
  - 64 parallel data transfer to arbitrary destination only in 7 cycle @64PE LPA

Comparison with Other 2-D Network



Maximum PE distance

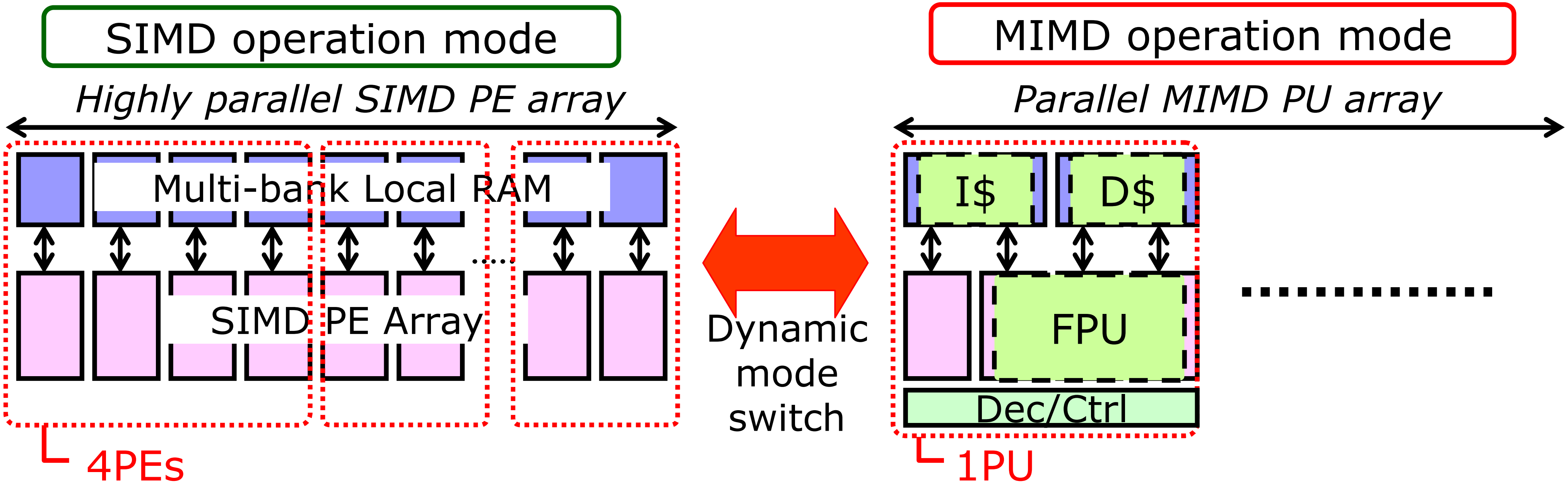
	Max. Distance
Hier. Ring	((N/2)/M)+1
2D-square	sqrt(N)-1
2D-octagonal	sqrt(N)/2

N: #PEs in processor
M: #PEs in a PE group

Hier. Ring outperforms
1) 2D-square (if #PE<1024)
2) 2D-octagonal (if #PE<256)

- Better performance without complex routing logic needed in 2D network (cf. NoC)

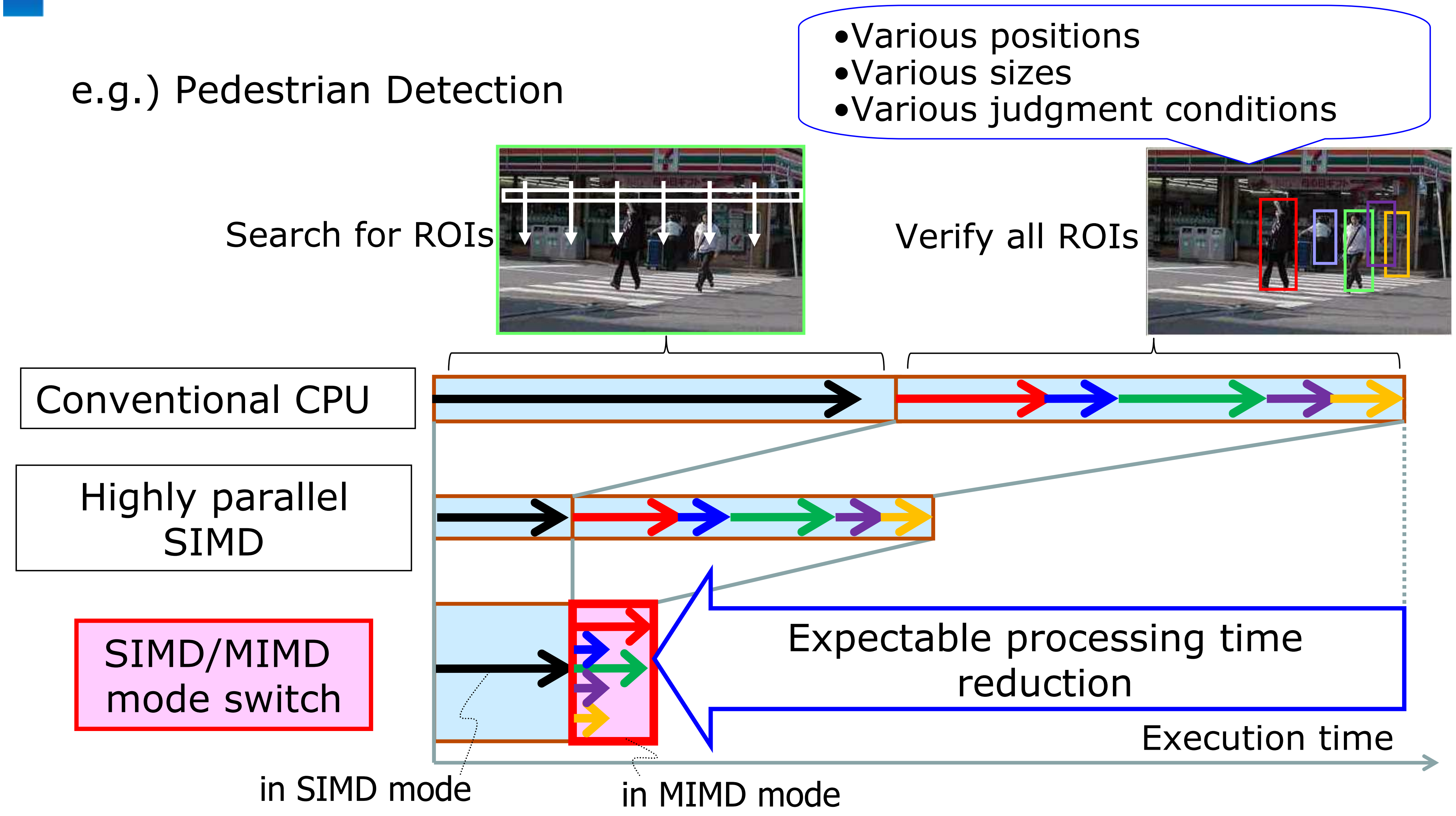
B) Smarter Sequence Control



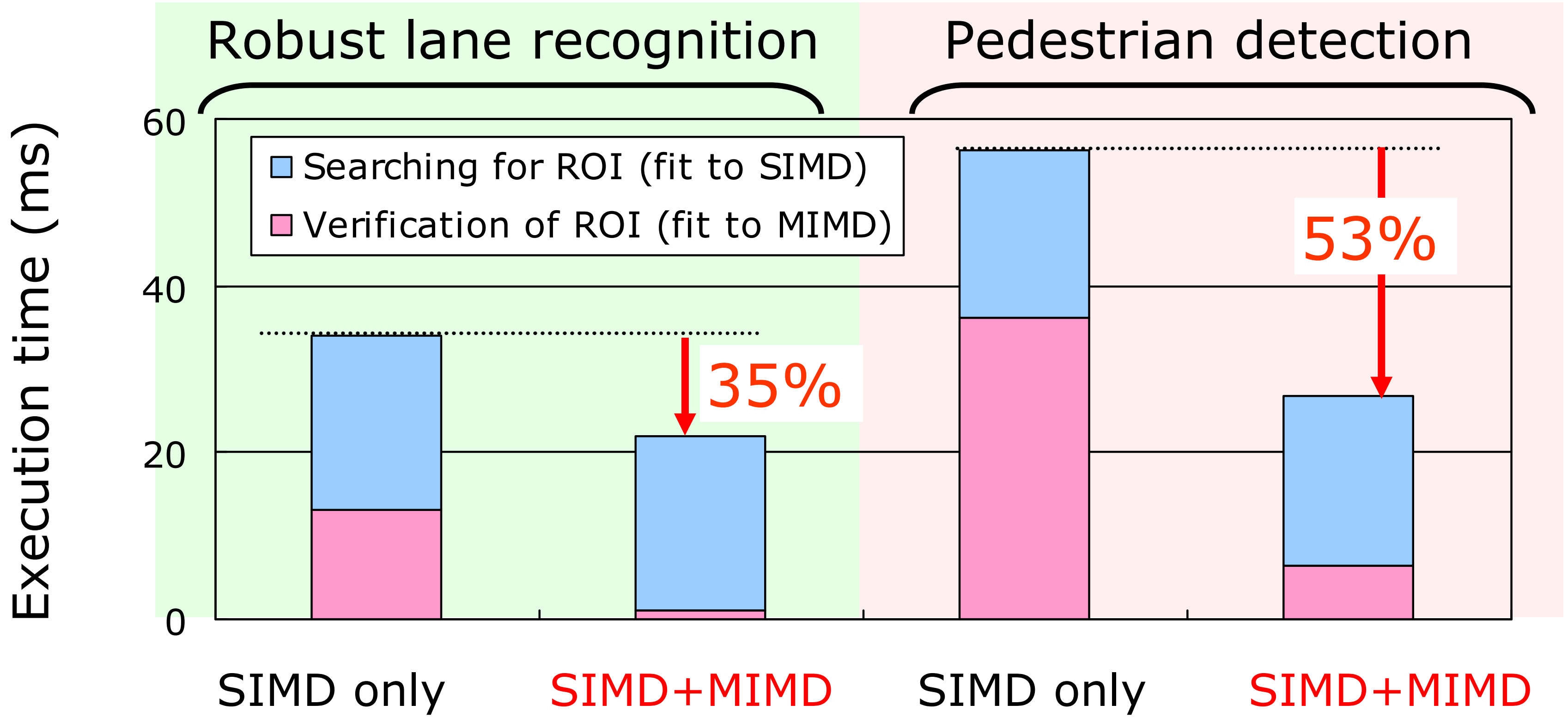
- 4 PEs dynamically reconfigures into a Processing Unit (PU)
  - E.g. 64 SIMD PEs into 16PUs (=16 MIMD processors)
- >10% of HW area overhead
  - Data paths and RAMs are reused/reconfigured as FPU and caches

Mode Switch Scenario

e.g.) Pedestrian Detection



Mode Switch Performance Example

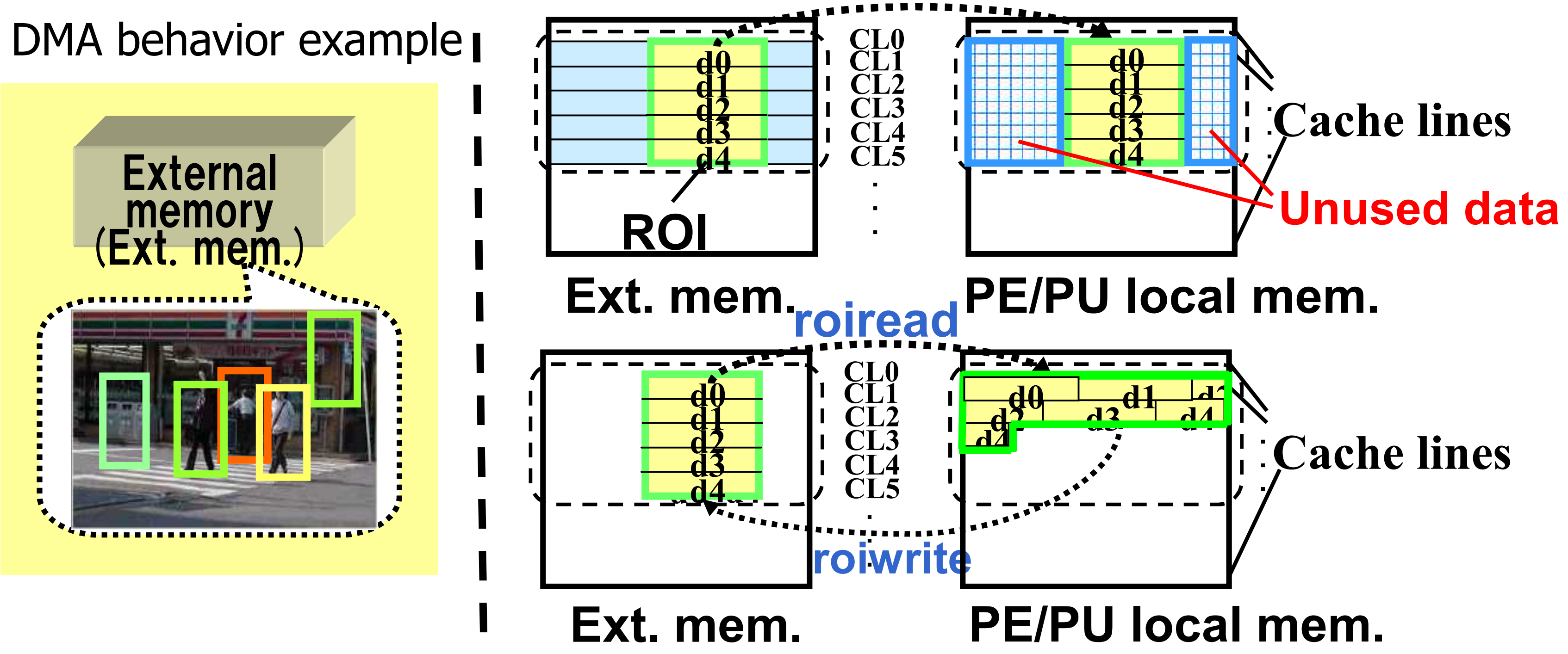


- Up to 2x speedup by switching to MIMD

C) Smarter DMA

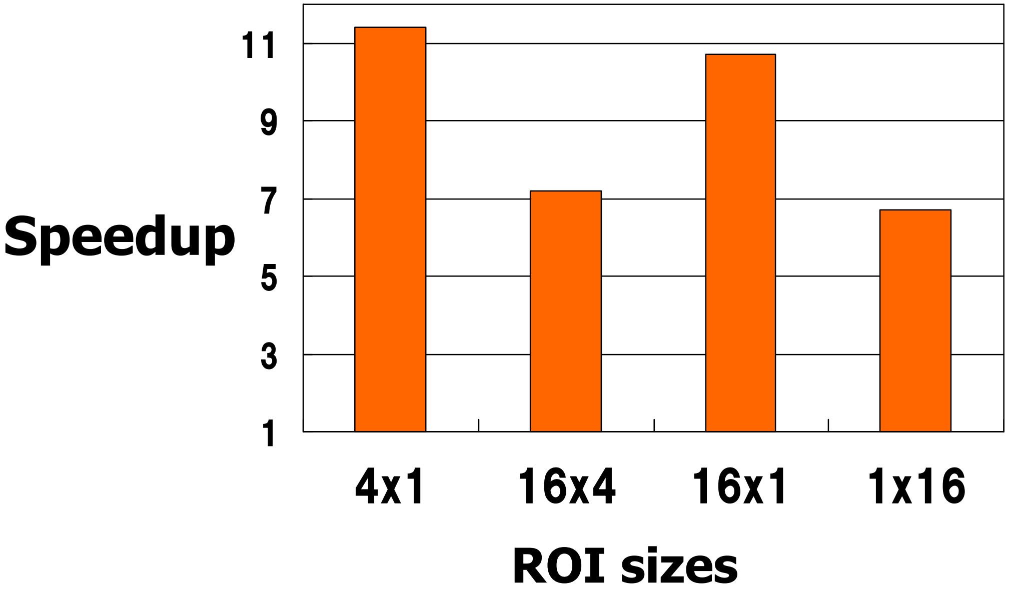
- Asynchronous DMA engine for
  - ROI to/from continuous data transfer (e.g. for PyrLK optical flow, SIFT, SURF, ...)
  - Random to/from continuous data transder (e.g. for Haar-like features)

ROI DMA behavior example

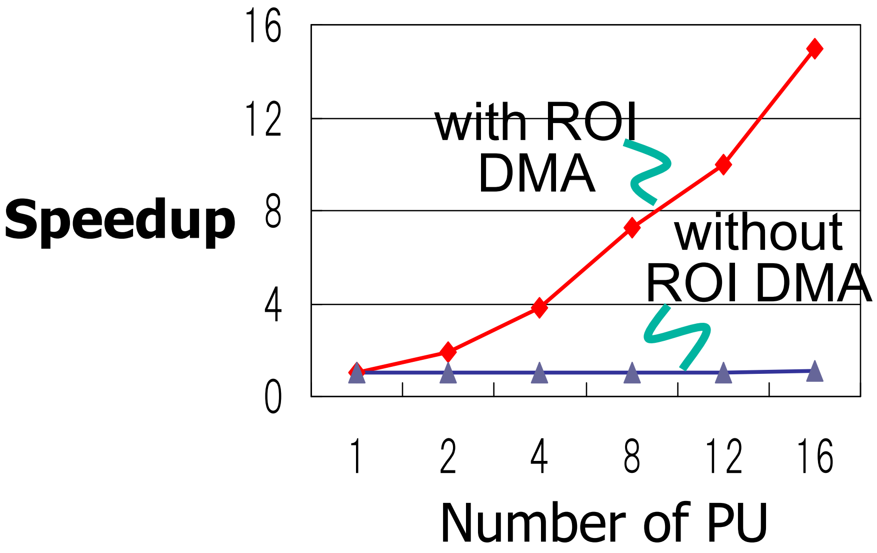


ROI DMA Use Case Performance Example

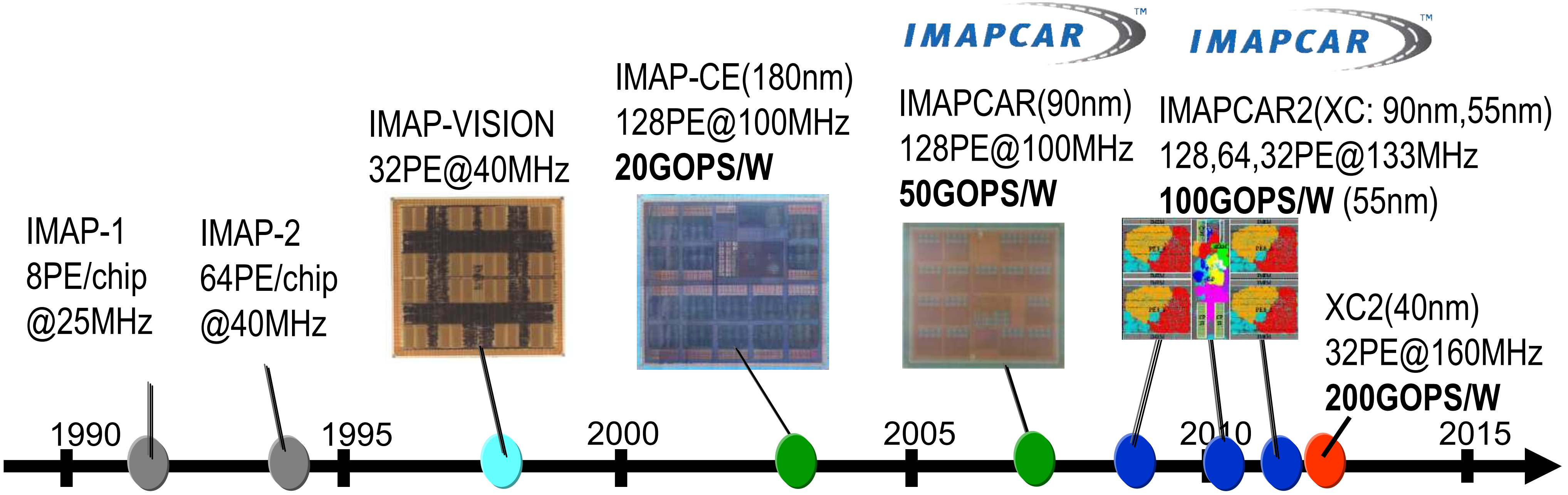
- Speedup based on the use of ROI DMA **in SIMD mode**



- 16PU performing Histogram Equalization against 256 ROIs **in MIMD mode**



CV Accelerator Products and IP Cores



IMAP-1
8PE/chip
@25MHz

IMAP-2
64PE/chip
@40MHz

IMAP-VISION
32PE@40MHz

IMAP-CE(180nm)
128PE@100MHz
20GOPS/W

IMAPCAR(90nm)
128PE@100MHz
50GOPS/W

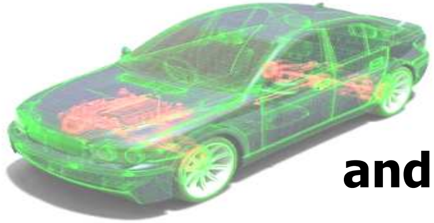
IMAPCAR2(XC: 90nm,55nm)
128,64,32PE@133MHz
100GOPS/W (55nm)

XC2(40nm)
32PE@160MHz
200GOPS/W

Traffic monitoring systems

Reticle inspection systems

Surveilance systems



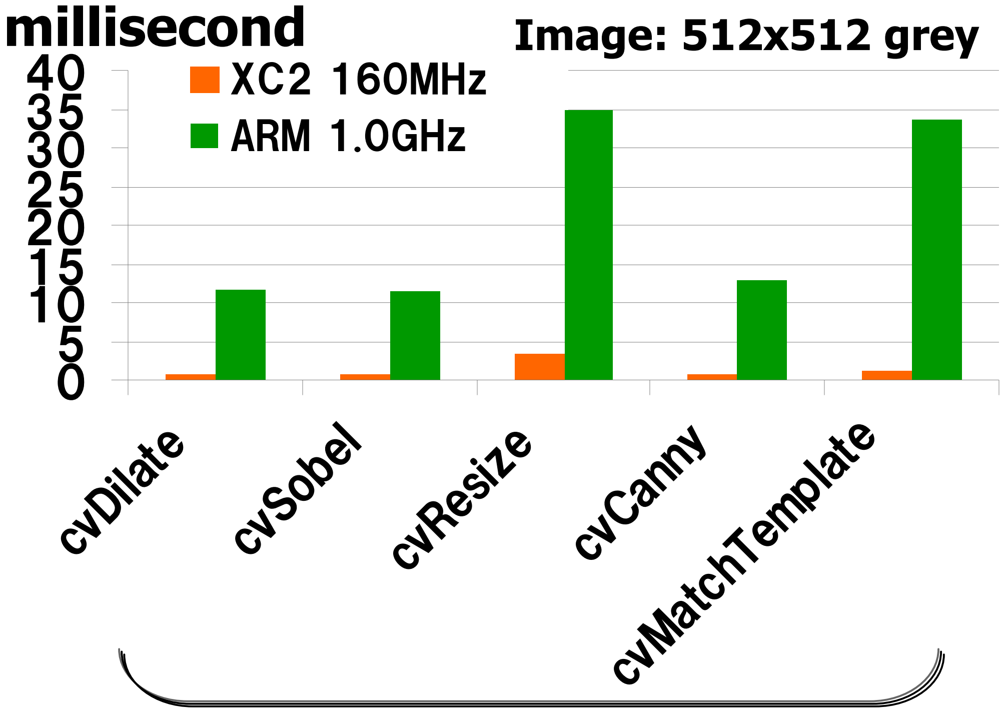
and more

Precrash safety systems
Navigation systems

1st generation design ← → 2nd generation design

Specification of A Latest CV Accelerator Core (XC2)

- **Performance and power vs. a 1.0GHz CPU**
  - **18x in average more performant**
  - **>3x more power efficient**



Typical OpenCV functions

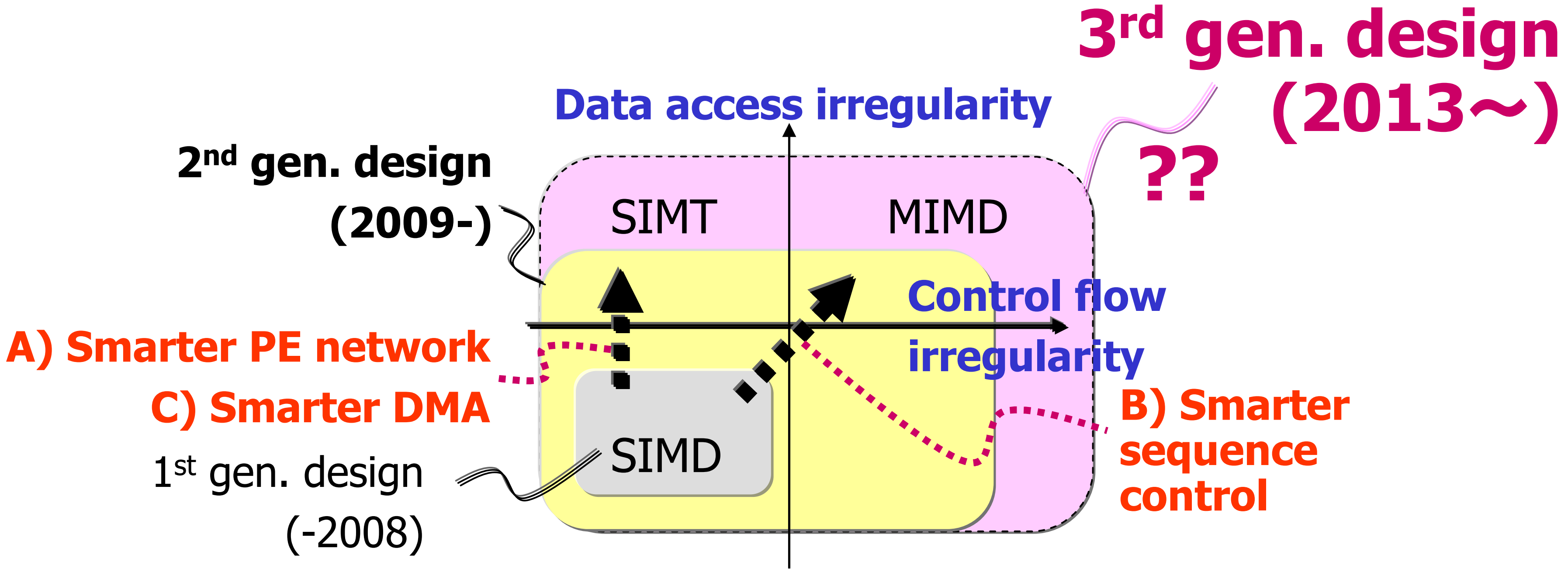
# of PE	32
Memory	4KB/PE
Freq.	160MHz
Process	40nm
Power	100mW
Die area	2.7mm^2
Peak Perf.	51.2GOPS

Available SW environment

- **Eclipse IDE and graph based GUI tool**
- **>30 OpenCV compatible functions ready**
- **>100 proprietary CV functions ready**

Perspective Toward Next Gen. CV Accelerators

- Incorporate MIMD multi/many-core ?
- Adding more smarter SIMD controllers ?
- More fixed function circuits ?



Outline

- Background
  - Anytime, everywhere CV apps
  - Issues to be addressed for CV HW and SW
- Hardware Architecture for CV
  - Approaches for performance and power
  - Case studies
- Software Framework for CV
  - Approaches for performance portability
  - Toward “open, efficient, and portable”
- Conclusions

Issue of Current CV Software Layer

- Lack of **open standard CV library**
  - OpenCV is currently the de-facto standard for research & prototyping
    - **Need modification for mobile and embedded use**
      - **An issue which Khronos OpenVX is trying to cover**

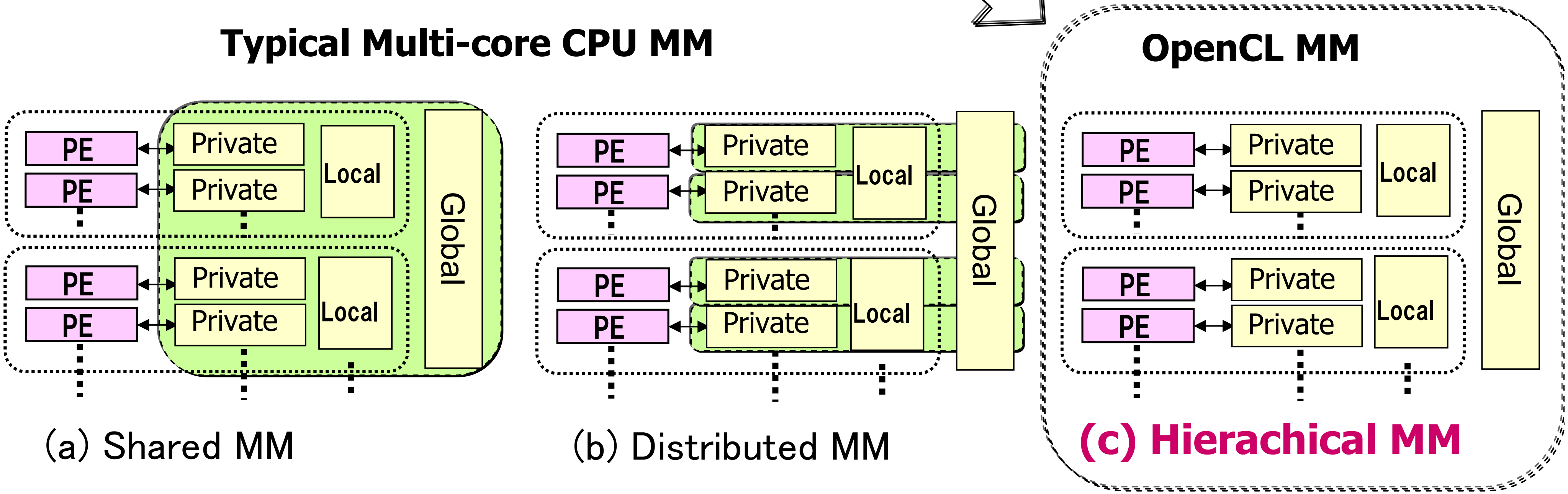
?

- Lack of adequate **open standard accelerator language**
  - OpenCL : an industrial standard accelerator language
    - **However, lacks performance portability**

?

Pros and Cons of OpenCL

- Supported by major vendors (Apple, Intel, AMD, NVIDIA, ...)
    - **Cross-platform**
- A hierarchical memory model (MM) is exposed to programmers
    - Good for tuning code for each target
        - ✓ **Not performance portable**



Typical Multi-core CPU MM

(a) Shared MM

(b) Distributed MM

OpenCL MM

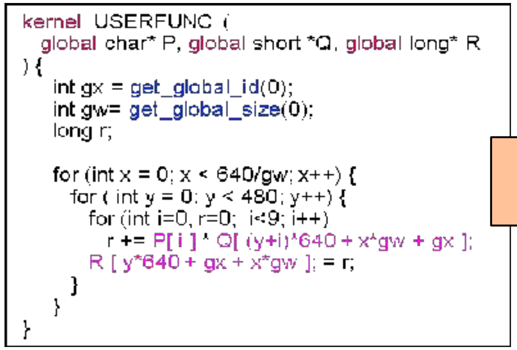
(c) Hierachical MM

🟩 **: MM abstraction range**

Target Dependency vs. Performance

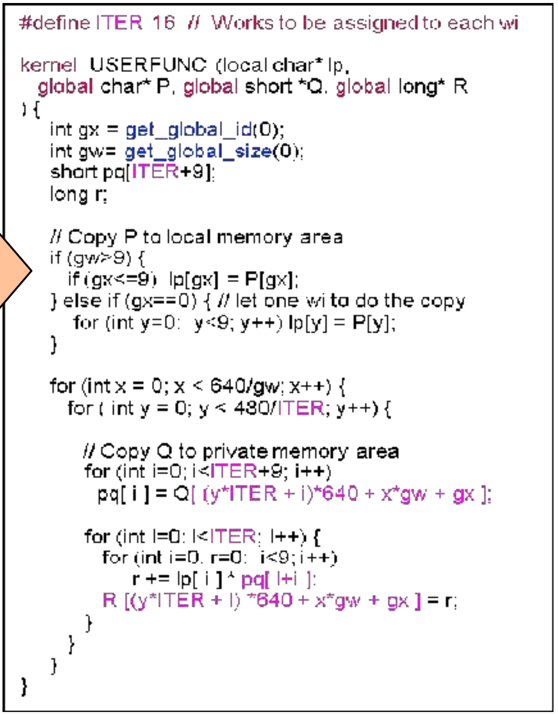
- Exposing the MM : losing performance portability
- Not exploiting the MM : losing performance

OpenCL-C code that tells the essential algorithm



```
kernel  USERFUNC (
   global char* P, global short *Q, global long* R
) {
     int gx = get_global_id(0);
     int gw= get_global_size(0);
     long r;

     for (int x = 0; x < 640/gw; x++) {
        for ( int y = 0; y < 480; y++) {
           for (int i=0, r=0;  i<9; i++)
              r += P[ i ] * Q[ (y+i)*640 + x*gw + gx ];
           R [ y*640 + gx + x*gw ]; = r;
        }
     }
}
```

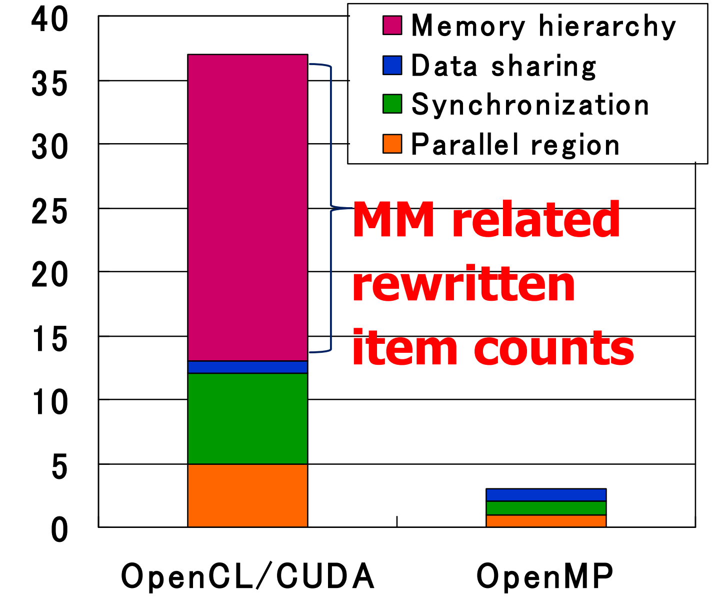
OpenCL-C code tuned for performance by using local and private memory



```
#define ITER  16  //  Works to be assigned to each wi

kernel  USERFUNC (local char* lp,
   global char* P, global short *Q, global long* R
) {
     int gx = get_global_id(0);
     int gw= get_global_size(0);
     short pq[ITER+9];
     long r;

     // Copy P to local memory area
     if (gw>9) {
        if (gx<=9)  lp[gx] = P[gx];
     } else if (gx==0) { // let one wi to do the copy
         for (int y=0;  y<9; y++) lp[y] = P[y];
     }

     for (int x = 0; x < 640/gw; x++) {
        for ( int y = 0; y < 480/ITER; y++) {

           // Copy Q to private memory area
           for (int i=0; i<ITER+9; i++)
              pq[ i ] = Q[ (y*ITER + i)*640 + x*gw + gx ];

           for (int l=0; l<ITER; l++) {
              for (int i=0, r=0;  i<9; i++)
                   r += lp[ i ] * pq[ l+i ];
              R [(y*ITER + l) *640 + x*gw + gx ] = r;
           }
        }
     }
}
```

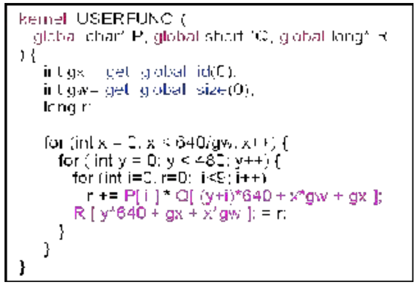
Clarity is lost, however, achieves better performance

Number of items need to be rewritten when porting a face detection C code for performance [Cho '10]



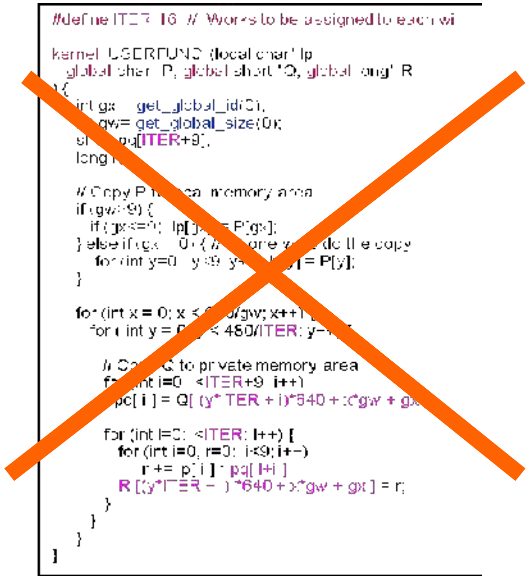
Can We Do it Like this (in the Context of CV) ?

Essential algorithmic kernel



```
kernel USERFUNC (
  global char* P, global short *Q, global long* R
){
    int gx = get_global_id(0);
    int gw= get_global_size(0);
    long r;

    for (int x = 0; x < 640/gw; x++) {
      for ( int y = 0; y < 480; y++) {
        for (int i=0, r=0; i<9; i++)
          r += P[ i ] * Q[ (y+i)*640 + x*gw + gx ];
        R [ y*640 + gx + x*gw ]; = r;
      }
    }
}
```

Mixed by target dependent code (Approach of OpenCL)

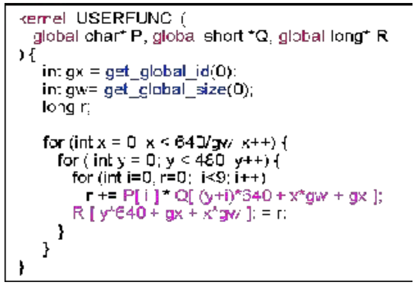


```
#define ITER 16 // Works to be assigned to each wi

kernel USERFUNC (local char* lp
  global char* P, global short *Q, global long* R
){
    int gx = get_global_id(0);
    int gw= get_global_size(0);
    short pq[ITER+9];
    long r;

    // Copy P to local memory area
    if (gw>9) {
      if (gx<=9) lp[gx] = P[gx];
    } else if (gx == 0) { // one work do the copy
      for (int y=0; y<9; y++) lp[y] = P[y];
    }

    for (int x = 0; x < 640/gw; x++) {
      for ( int y = 0; y < 480/ITER; y++) {

        // Copy Q to private memory area
        for (int i=0; i<ITER+9; i++)
          pq[ i ] = Q[ (y*ITER + i)*640 + x*gw + gx];

        for (int l=0; l<ITER; l++) {
          for (int i=0, r=0; i<9; i++)
            r += lp[ i ] * pq[ l+i ]
          R [(y*ITER + l)*640 + x*gw + gx ] = r;
        }
      }
    }
}
```

New SW Framework

Goal

Essential algorithmic kernel



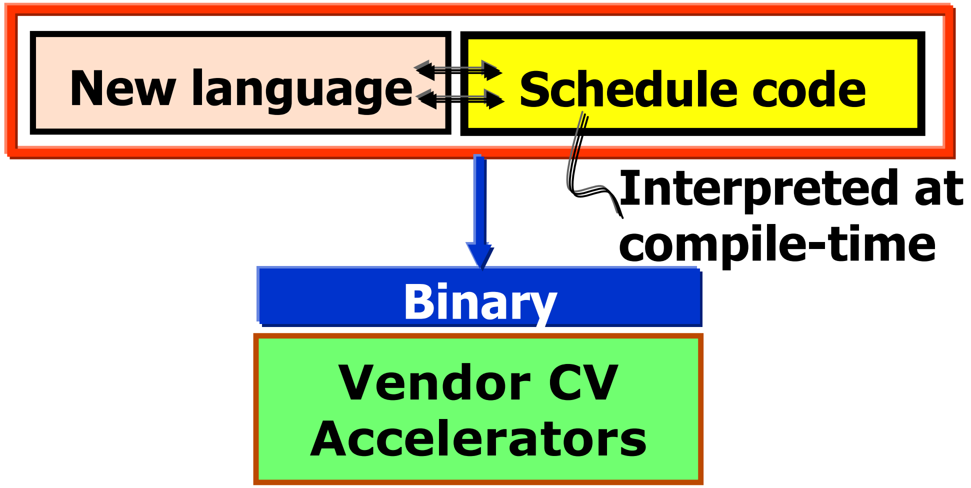
```
kernel USERFUNC (
  global char* P, global short *Q, global long* R
){
    int gx = get_global_id(0);
    int gw= get_global_size(0);
    long r;

    for (int x = 0; x < 640/gw; x++) {
      for ( int y = 0; y < 480; y++) {
        for (int i=0, r=0; i<9; i++)
          r += P[ i ] * Q[ (y+i)*640 + x*gw + gx ];
        R [ y*640 + gx + x*gw ]; = r;
      }
    }
}
```

+

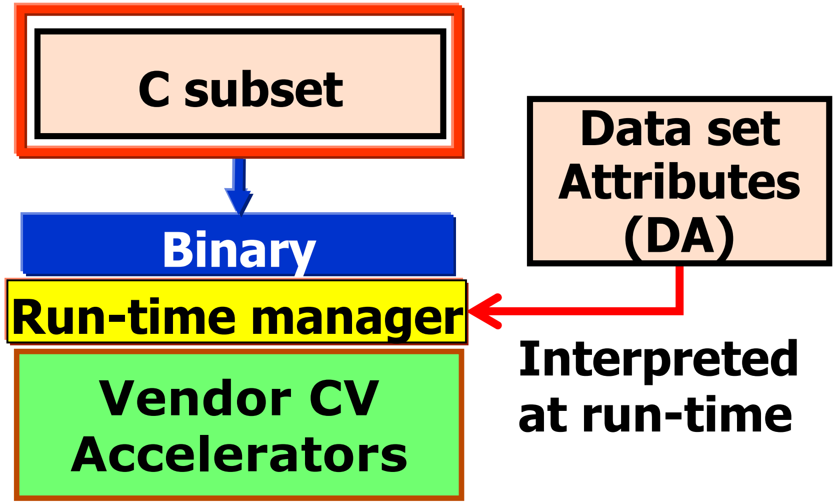
Some kind of target dependent description ?

Related Works

- Compile-time approach([Halide ’12])
  - MM opaque code (new language)
  - MM aware schedule code



- Run-time approach ([AC-FW ’11])
  - MM opaque code (C subset)
  - Data set attribute
  - Run-time manager

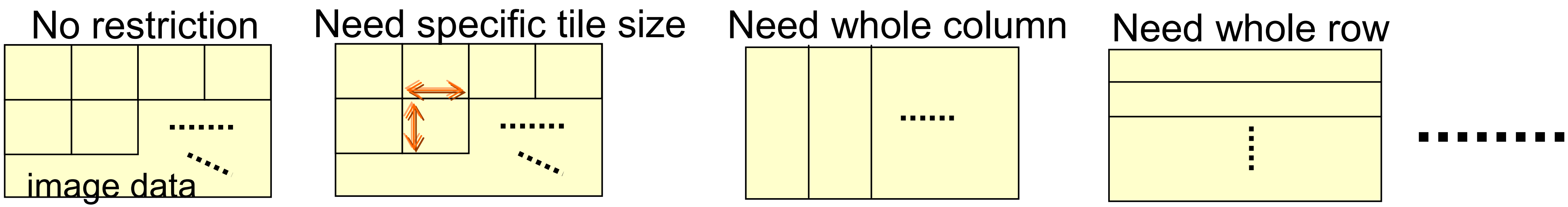


AC-FW: Automated Chaining Framework

□ Compiler input □ Algorithmic features 🟨 Target dependent part

AC-FW : DA (Data-set Attribute) as Algorithmic Features

◆ Dividing (or Tiling) attribute



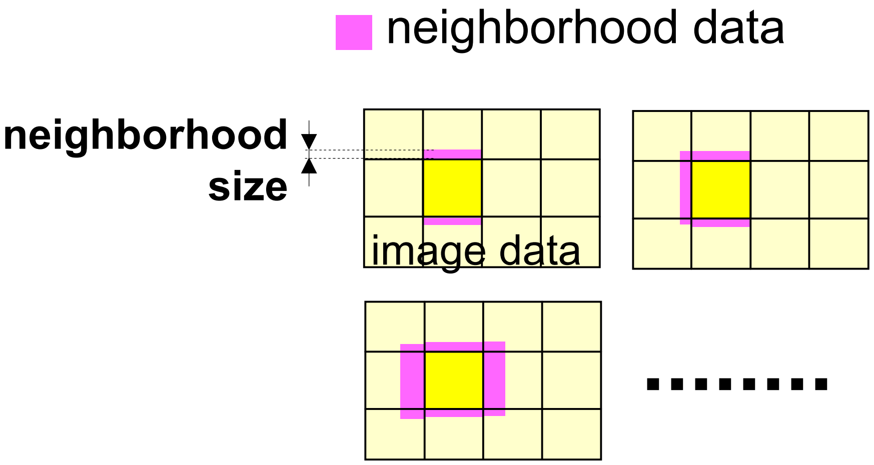
No restriction

Need specific tile size

Need whole column

Need whole row

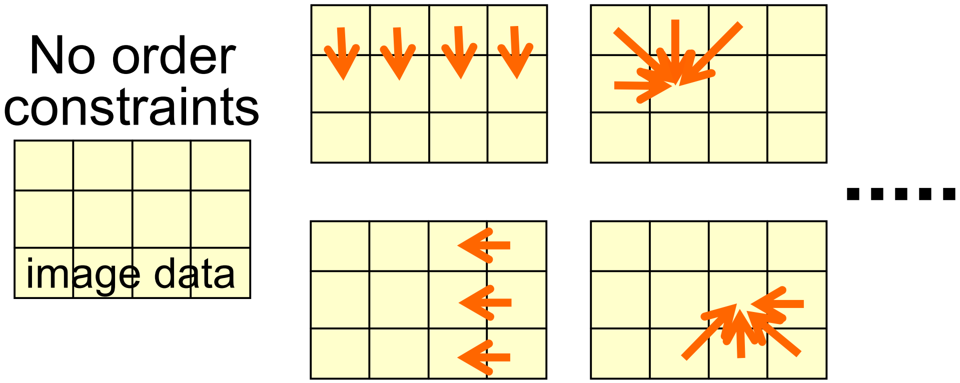
◆ Neighborhood size attribute



■ neighborhood data

neighborhood size

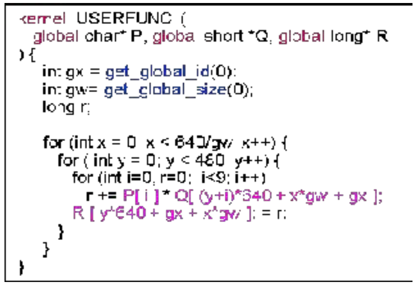
◆ Ordering attribute



No order constraints

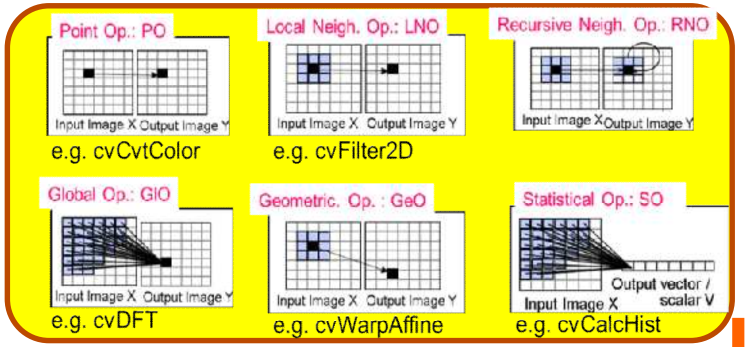
AC-FW : The Run-time Manager

Essential algorithmic kernel



```
kernel  USERFUNC (
  global char* P, globa  short *Q, global long* R
) {
    int gx = get_global_id(0);
    int gw= get_global_size(0);
    long r;

    for (int x = 0; x < 640/gw; x++) {
      for ( int y = 0; y < 480; y++) {
        for (int i=0, r=0;  i<9; i++)
          r += P[ i ] * Q[ (y+i)*640 + x*gw + gx ];
        R [ y*640 + gx + x*gw ]; = r;
      }
    }
}
```

DA derived from memory access pattern of the algorithm



Target independent descriptions

Apply frequently used optimizing techniques

- Divide data into tiles
- Asynchronous DMA
- Pipelining
- ……

Taking into account target HW features:

- Number of cores …
- Private memory size …
- Local memory size …
- Global mem. acc. latency

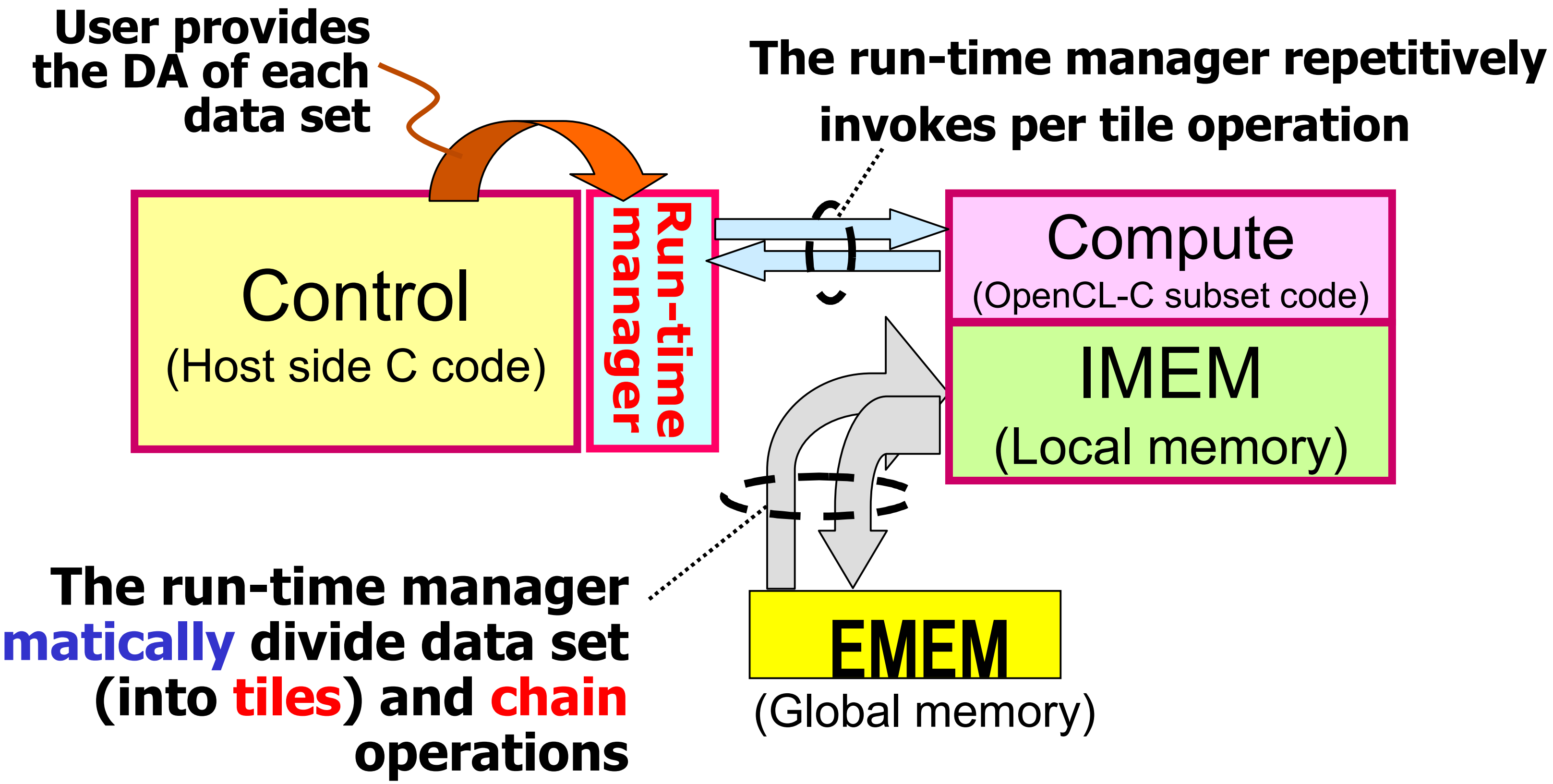
….

Run-time manager

A run-time having better knowledge about the target HW than the programmer

AC-FW : Control and Data Flow

- **The run-time manager *tiles*** data transfer, ***chains*** kernels and repetitively ***invokes*** per tile operation



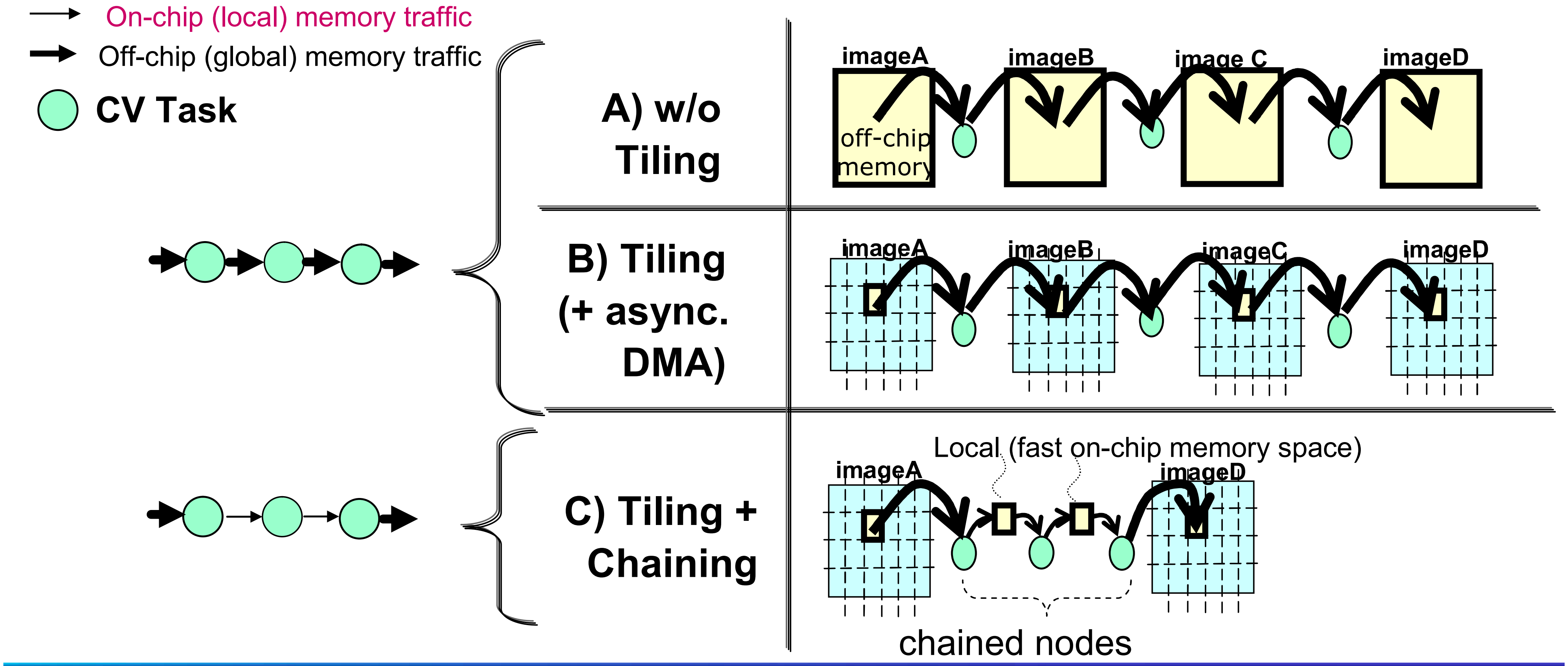
User provides the DA of each data set

The run-time manager repetitively invokes per tile operation

The run-time manager automatically divide data set (into tiles) and chain operations

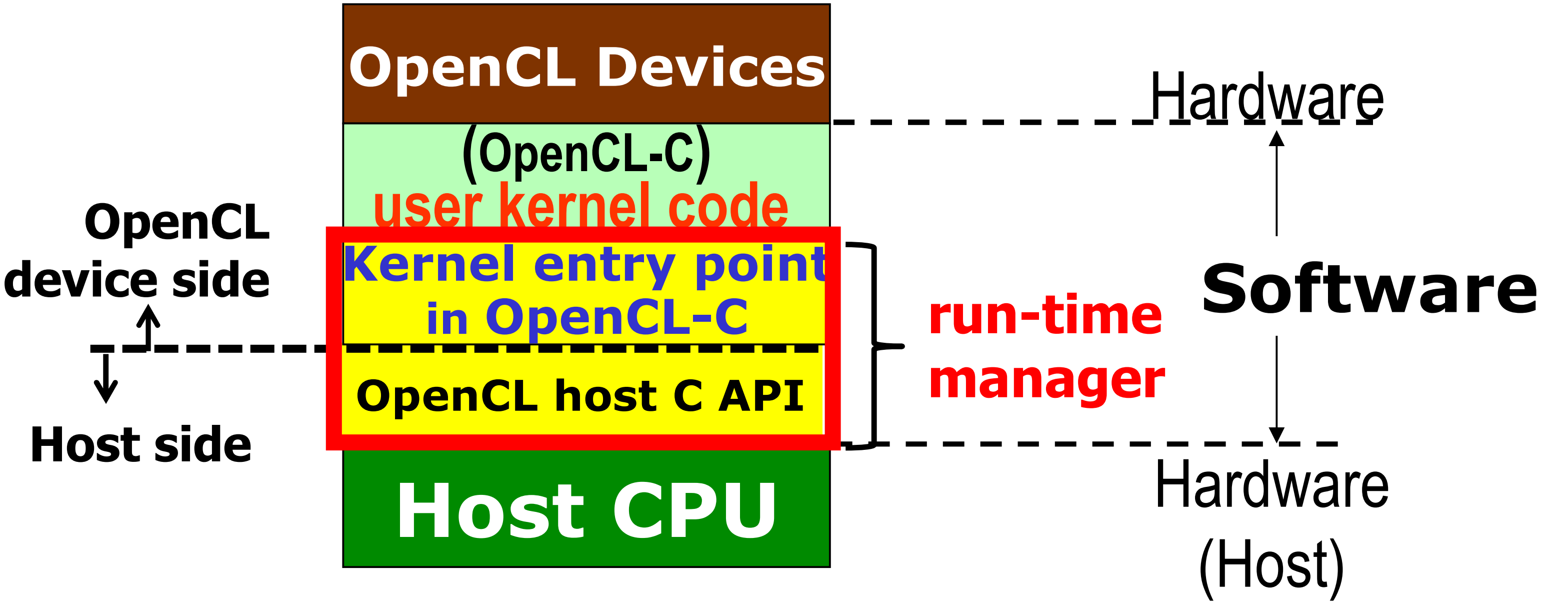
AC-FW : Automated Chaining

- Tiling : Reduce waiting time due to access latency
- Chaining: Reduce memory bandwidth



Run-time Manager Design Example for OCL Devices

- ***Run-time manager*** generates host C API calls
  - Decide the best **work-item size**
- ***Run-time manager*** generates kernel entry point OpenCL-C code
  - Chain consecutive user kernel code
  - Exploit the **use of local memory**



Non-Trivial OCL Device Dependent Optimizations

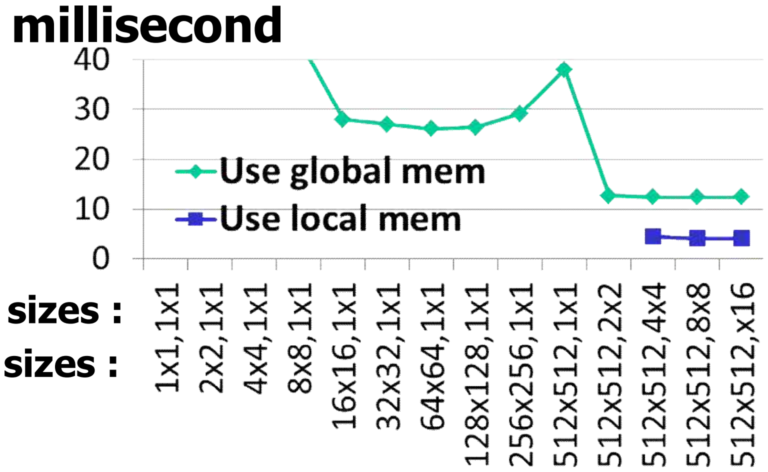
- Each target requires **different work-item size decisions**
- Each target requires **different local memory usage strategy**

Case study of a 3x3 image convolution (image: VGA size)

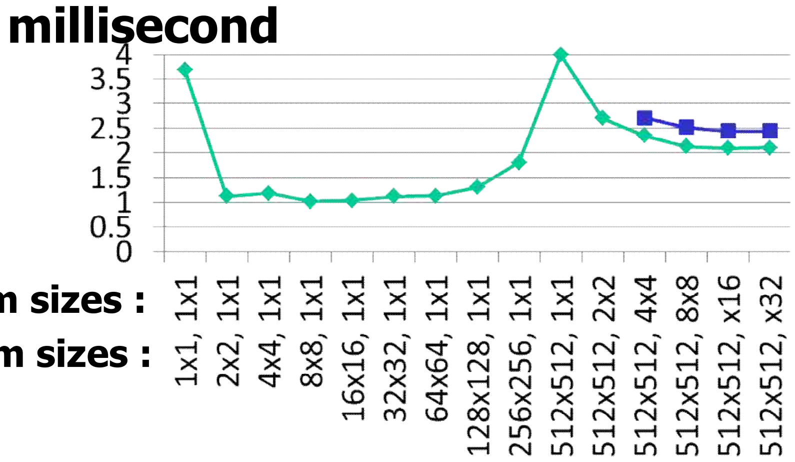
a. **NVIDIA Quadro FX3700 (14*8PE) @500MHz, OpenCL@CUDA SDK 3.2**

b. **Intel Core(TM) i5-2400 @3.10GHz, Intel OpenCL SDK 1.1**

Global item sizes :
Local item sizes :

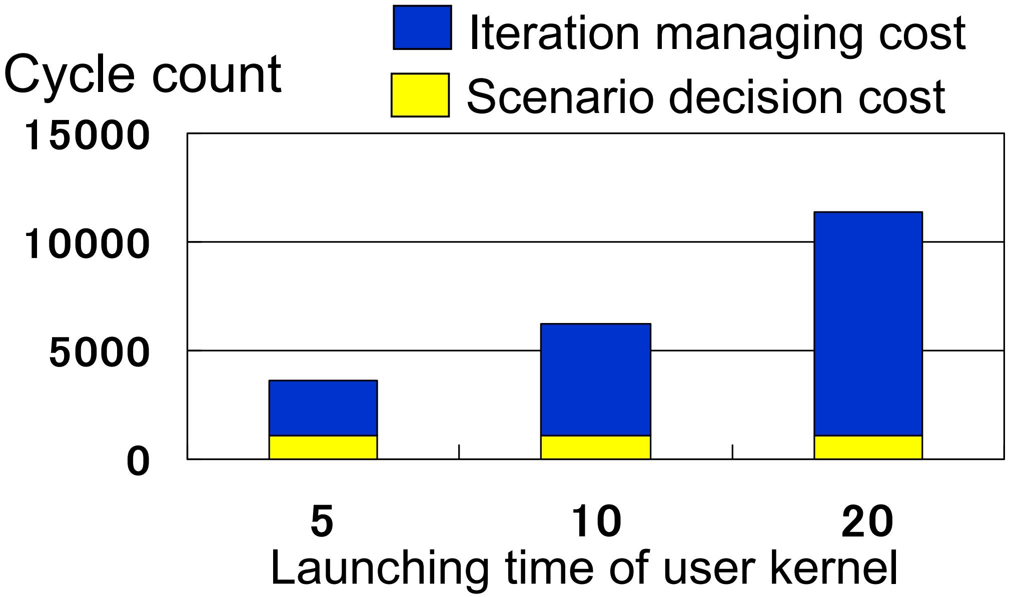


Global item sizes :
Local item sizes :



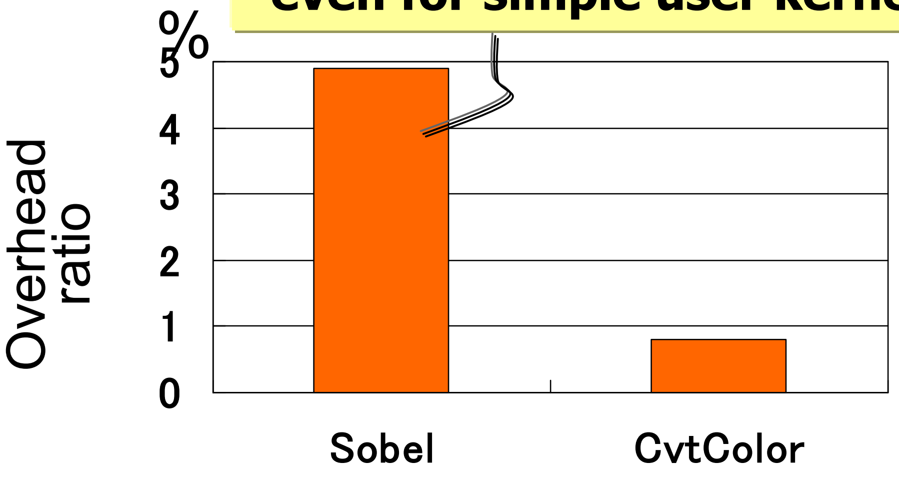
Run-time Manager Overhead Case Study

- Fixed overhead occured by scenario decision
- Overhead occured per user kernel call (data copy etc.)



(a) Overhead composition of the run-time manager

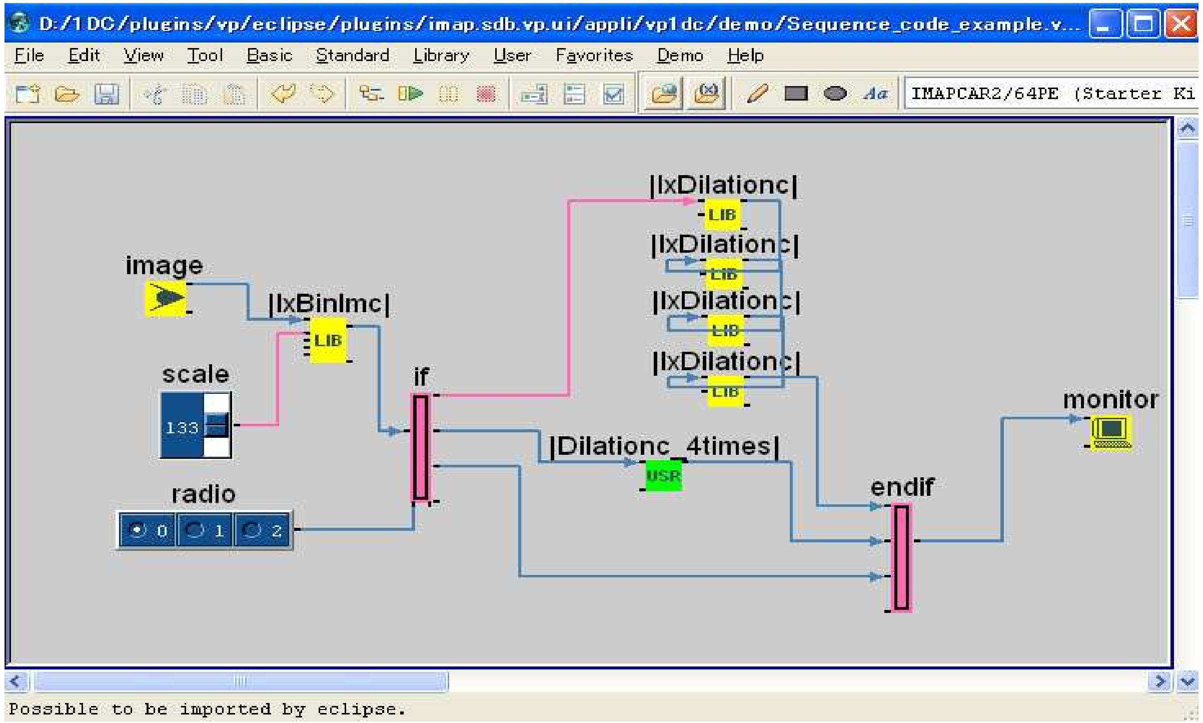
Overhead is small (< 5%) even for simple user kernel



(b) Ratio of the overhead against total processing time

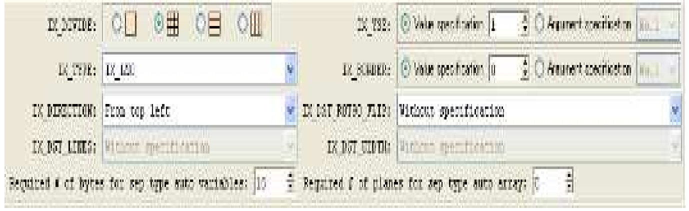
Graph Based GUI Environment for AC-FW

- DA is pre-registered for each pre-defined library node
- User defines DA of each user-node
- Node tiling is automated by the run-time manager


D:/1DC/plugins/vp/eclipse/plugins/imap.sdb.vp.ui/appli/vp1dc/demo/Sequence_code_example.v...
File Edit View Tool Basic Standard Library User Favorites Demo Help
IMAPCAR2/64PE (Starter Ki
image
|IxBinImc|
LIB
scale
133
radio
0 1 2
if
|IxDilationc|
LIB
|IxDilationc|
LIB
|IxDilationc|
LIB
|IxDilationc|
LIB
|Dilationc_4times|
USR
endif
monitor

Possible to be imported by eclipse.

Panel for DA setting



IX_DIVIDE: ○ ● ○ ○

IX_YSZ: ● Value specification 1 ○ Argument specification

IX_TYPE: IX_LRC

IX_BORDER: ● Value specification 0 ○ Argument specification

IX_DIRECTION: From top left

IX_DST_ROT90_FLIP: Without specification

IX_DST_LINES: Without specification

IX_DST_WIDTH: Without specification

Required # of bytes for sep type auto variables: 16

Required # of planes for sep type auto array: 0

LIB Library node

USR User node

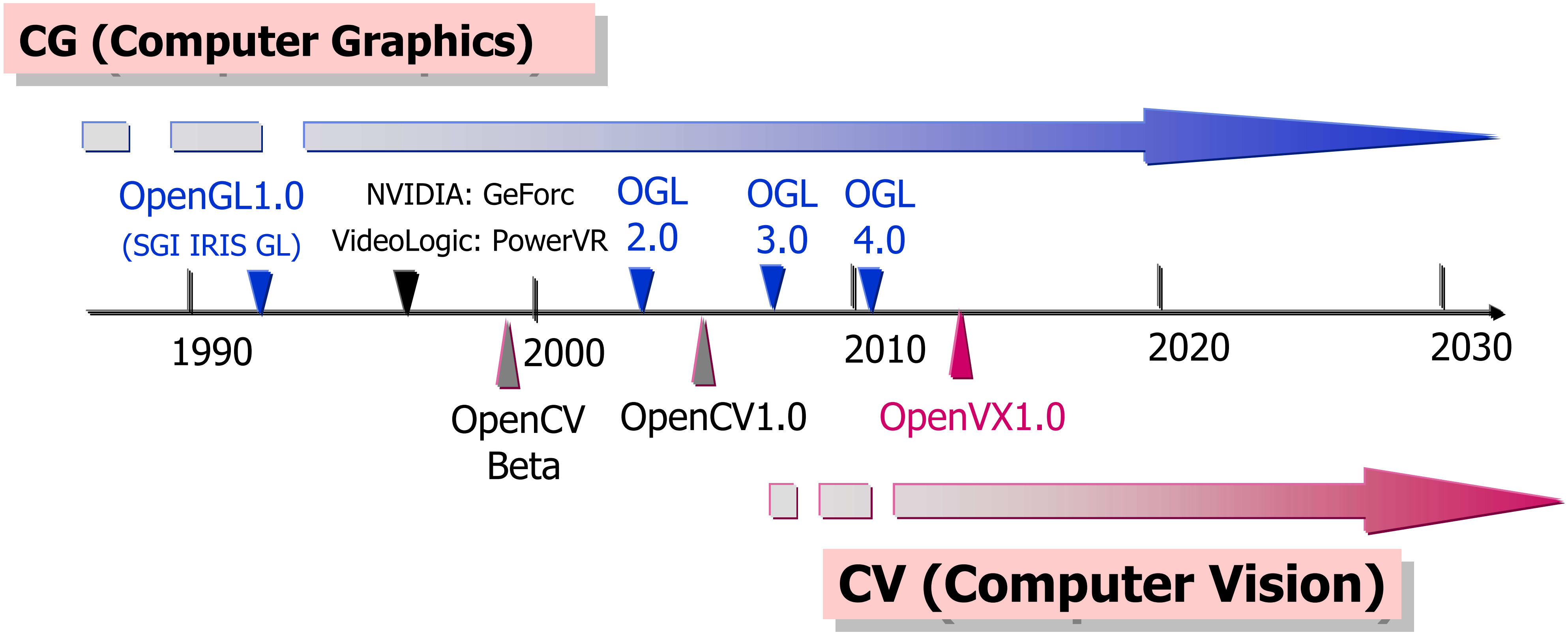
RENESAS

Outline

- Background
  - Anytime, everywhere CV apps
  - Issues to be addressed for CV HW and SW
- Hardware Architecture for CV
  - Approaches for performance and power
  - Case studies
- Software Framework for CV
  - Approaches for performance portability
  - Toward “open, efficient, and portable”
- Conclusions

Toward Open, Efficient, and Portable CV SW

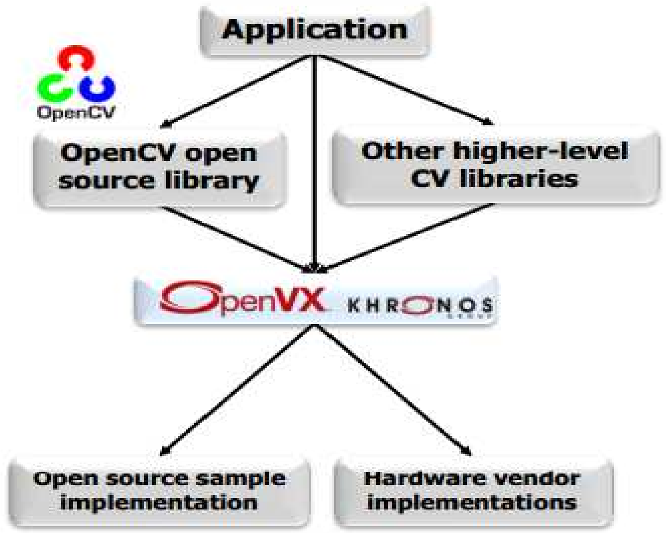
- Analogous with the CG evolution in early 1990s
  - OpenCV bridged the gap between research and applications
  - OpenVX V1.0 is coming up



The Khronos OpenVX

- OpenVX is under design to serve as an open standard HW acceleration layer
  - Specification before end of 2013
  - Support graph execution
  - Support a subset of OpenCV functions

Position of OpenVX



Application

OpenCV

OpenCV open source library

Other higher-level CV libraries

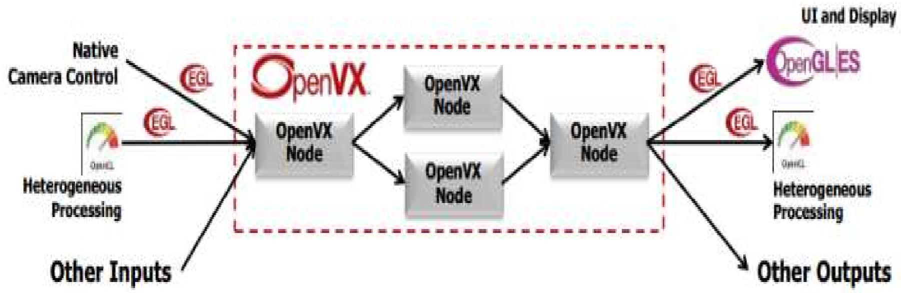
OpenVX KHRONOS

Open source sample implementation

Hardware vendor implementations

AMD
Aptina
ARM
BARCO
BROADCOM
BDTi
CEVA
cognivue
DMP
ERICSSON
freescale
FUJITSU
HI CORP.
hp
intel
itseez
KISHONTI
MARVELL
MEDIATEK
Movidius
NVIDIA
QUALCOMM
QNX
RENESAS
SAMSUNG
SONY
ST
symbio
TAKUMI
TEXAS INSTRUMENTS
Think Silicon
XILINX
VIVANTE

OpenVX graph



Native Camera Control

EGL

OpenCL

Heterogeneous Processing

Other Inputs

OpenVX

OpenVX Node

OpenVX Node

OpenVX Node

OpenVX Node

UI and Display

OpenGL ES

OpenCL

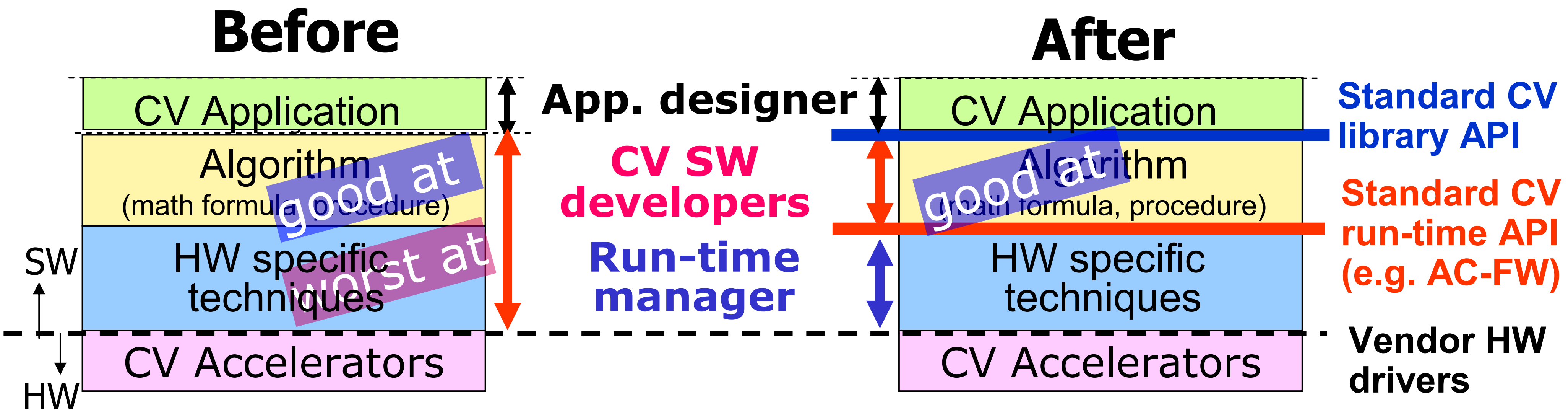
Heterogeneous Processing

Other Outputs

http://www.khronos.org/openvx

Toward “Write Once, Performant Everywhere”

- **Open standard CV library API**
  - Delivers fully tuned **off-the-shelf CV techniques**
- **Open standard common C-subset + run-time manager**
  - Facilitates on-time delivery of **latest CV technologies**
  - Run-time manager **ensures an usable degree of performance**



Outline

- Background
  - Anytime, everywhere CV apps
  - Issues to be addressed for CV HW and SW
- Hardware Architecture for CV
  - Approaches for performance and power
  - Case studies
- Software Framework for CV
  - Approaches for performance portability
  - Toward "open, efficient, and portable"
- Conclusions

Conclusions

- Perspective of CV accelerator designs
  - “FU rich, CU poor” configuration is a necessity
    - Design smarter CUs under the context of CV
    - Case study of a series CV accelerator design
    - Perspective of next generation design
- Perspective of CV software framework
  - Approaches to address the performance portability issue
    - AC-FW as a case study
    - The coming Khronos OpenVX V1.0
    - Toward “write once and performant everywhere !”

References

- [IMAPCAR ‘08] S.Kyo et al.: “A 100 GOPS in-vehicle vision processor for pre-crash safety systems based on a ring connected 128 4-Way VLIW processing elements” , 2008 IEEE Symp.on VLIS Circuits, 28-29.
- [IMAPCAR2 ‘09] S.Kyo et al.: “IMAPCAR2: A Dynamic SIMD/MIMD Mode Switching Processor for Embedded Systems,” Hot Chips 21, 2009
- [IMAPCAR2 ‘10] S.Kyo et al.: “Performance Evaluation of the SIMD/MIMD Dynamic Mode Switching Processor IMAPCAR2,” IEICE Tech. Rep., vol. 110, no. 204, RECONF2010-36, pp.109-114, Sep., 2010.
- [Xetal ‘07] A.Abbo et al: “XETAL-II: A 107 GOPS, 600mW Massively-Parallel Processor for Video Scene Analysis, Proc. ISSCC, 2007.
- [CSX ‘08] A.Lokhmotov et al.: “Revisiting SIMD programming,” In Languages and Compilers for Parallel Computing, pp.32-46, Springer-Verlag, 2008.
- [MX ‘06] M.Nakajima et al: “A 40GOPS 250mW Massively Parallel Processor Based on Matrix Architecture,” ISSCC 2006.
- [Cho ‘10] S.M.Cho et.al.,”OpenCL and parallel primitives for digital TV applications”, IBM J. Res. & Dev., Vol.54, No.5, Paper 7, 2010.
- [Halide ‘12] Jonathan Ragan-Kelley et al.:”Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines”, ACM Transactions on Graphics 31(4), 2012
- [AC-FW ‘11] S. Kyo, et.al. Basic API Proposals for Improving OpenCL Performance Portability, 2011-ARC-196(12), 1-8.

Renesas Electronics Corporation