Rechnerarchitektur (RA)

Sommersemester 2020

Architecture-Aware Optimizations
- Hardware-Software co-Optimizations-

Jian-Jia Chen
Informatik 12
Jian-jia.chen@tu-..
http://ls12-www.cs.tu-dortmund.de/daes/
Tel.: 0231 755 6078

Outline

High-Level Optimization
- *Loop transformation*
- *Loop tiling/blocking*
- *Loop (nest) splitting*

Heterogeneous System Architecture (HSA)
- *Integrated CPU/GPU platforms*
- *Recent movement in chip designs*

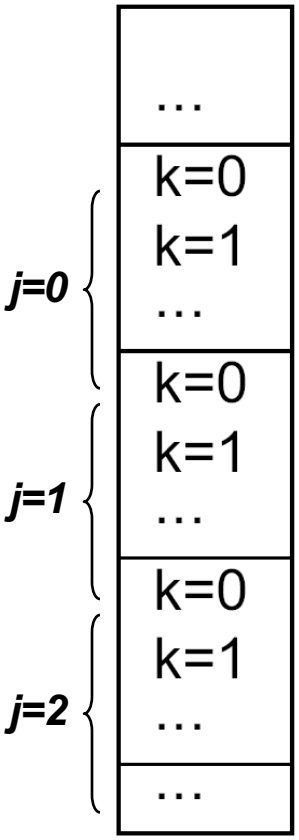
Architecture-aware software designs
- *Energy-efficiency issues*
- *Darkroom*
- *Halide*

Multicore revolutions
- *Impact on the "safety-critical" industry sector*

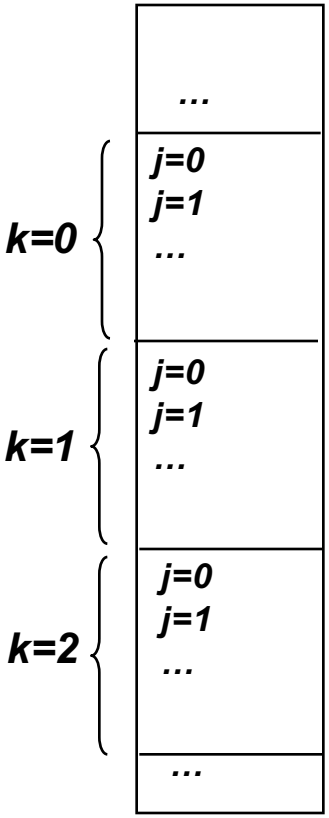
Impact of memory allocation on efficiency

Array p[j][k]

Row major order (C)



Column major order (FORTRAN)



Best performance of innermost loop corresponds to rightmost array index

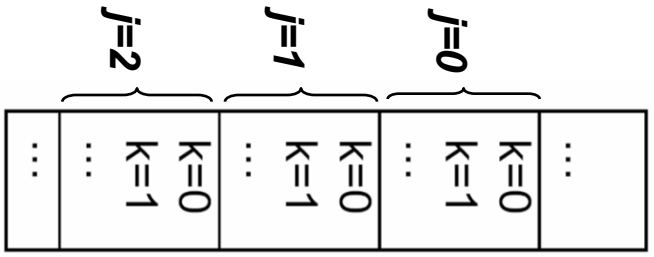
Two loops, assuming row major order (C):

```
for (k=0; k<=m; k++)
  for (j=0; j<=n; j++) )
    p[j][k] = …
```

```
for (j=0; j<=n; j++)
    for (k=0; k<=m; k++)
      p[j][k] = …
```

Same behavior for homogeneous memory access, but:

For row major order



↑ Poor cache behavior *Good cache behavior ↑*

☞ *memory architecture dependent optimization*

☞ Program transformation “Loop interchange”

Example:

```
…#define iter 400000
int a[20][20][20];
void computeijk() {int i,j,k;
   for (i = 0; i < 20; i++) {
       for (j = 0; j < 20; j++) {
             for (k = 0; k < 20; k++) {
                   a[i][j][k] += a[i][j][k];}}}}
void computeikj() {int i,j,k;
   for (i = 0; i < 20; i++) {
       for (j = 0; j < 20; j++) {
             for (k = 0; k < 20; k++) {
                   a[i][k][j] += a[i][k][j] ;}}}}…
```

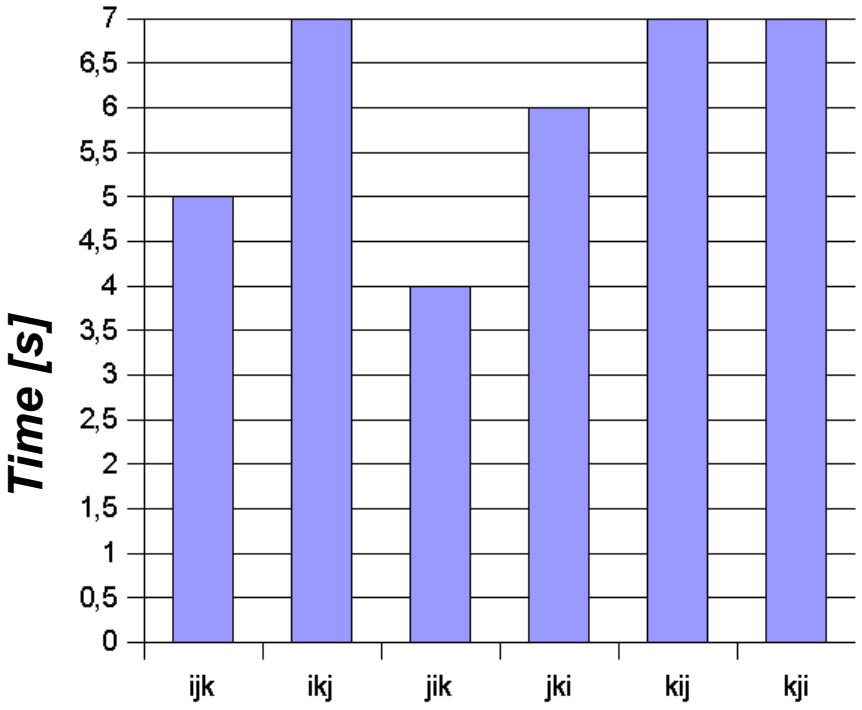
☞ *Improved locality*

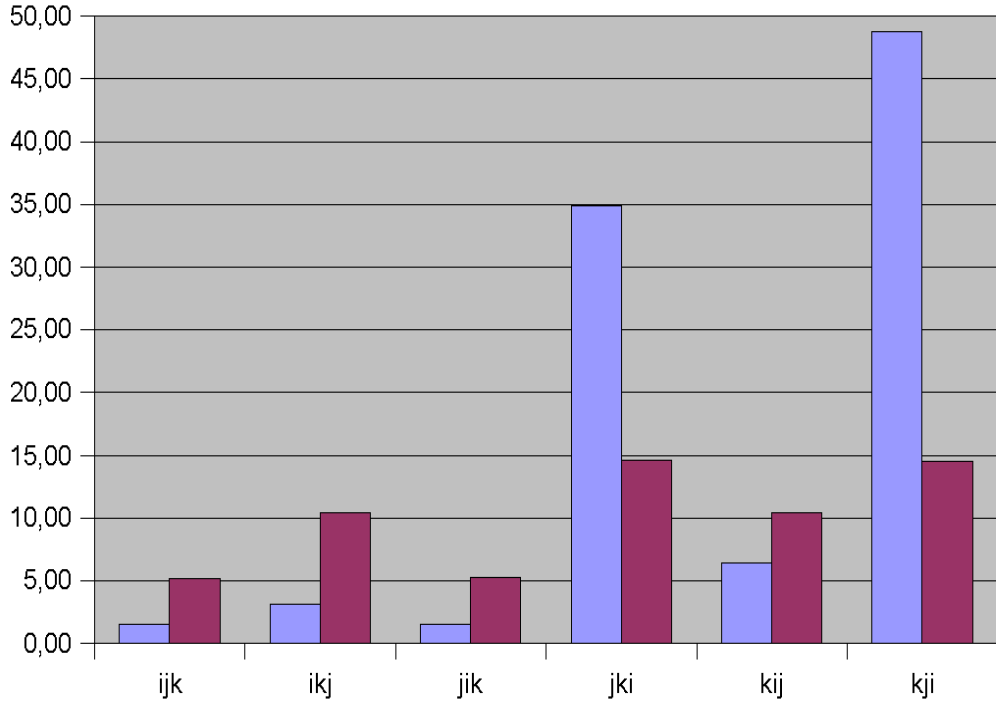
Results:
strong influence of the memory architecture

Loop structure: i j k

Dramatic impact of locality

Processor	*Ti C6xx*	*Sun SPARC*	*Intel Pentium*
reduction to [%]	*~ 57%*	*35%*	*3.2 %*



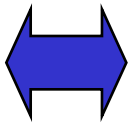


Not always the same impact ..

[Till Buchwald, Diploma thesis, Univ. Dortmund, Informatik 12, 12/2004]

Transformations
“Loop fusion” (merging), “loop fission”

```
for(j=0; j<=n; j++)
  p[j]= ... ;
for (j=0; j<=n; j++) ,
  p[j]= p[j] + ...
```



```
for (j=0; j<=n; j++)
  {p[j]= ... ;
  p[j]= p[j] + ...}
```

Loops small enough to allow zero overhead Loops

Better locality for access to p.
Better chances for parallel execution.

Which of the two versions is best?
Architecture-aware compiler should select best version.

Example: simple loops

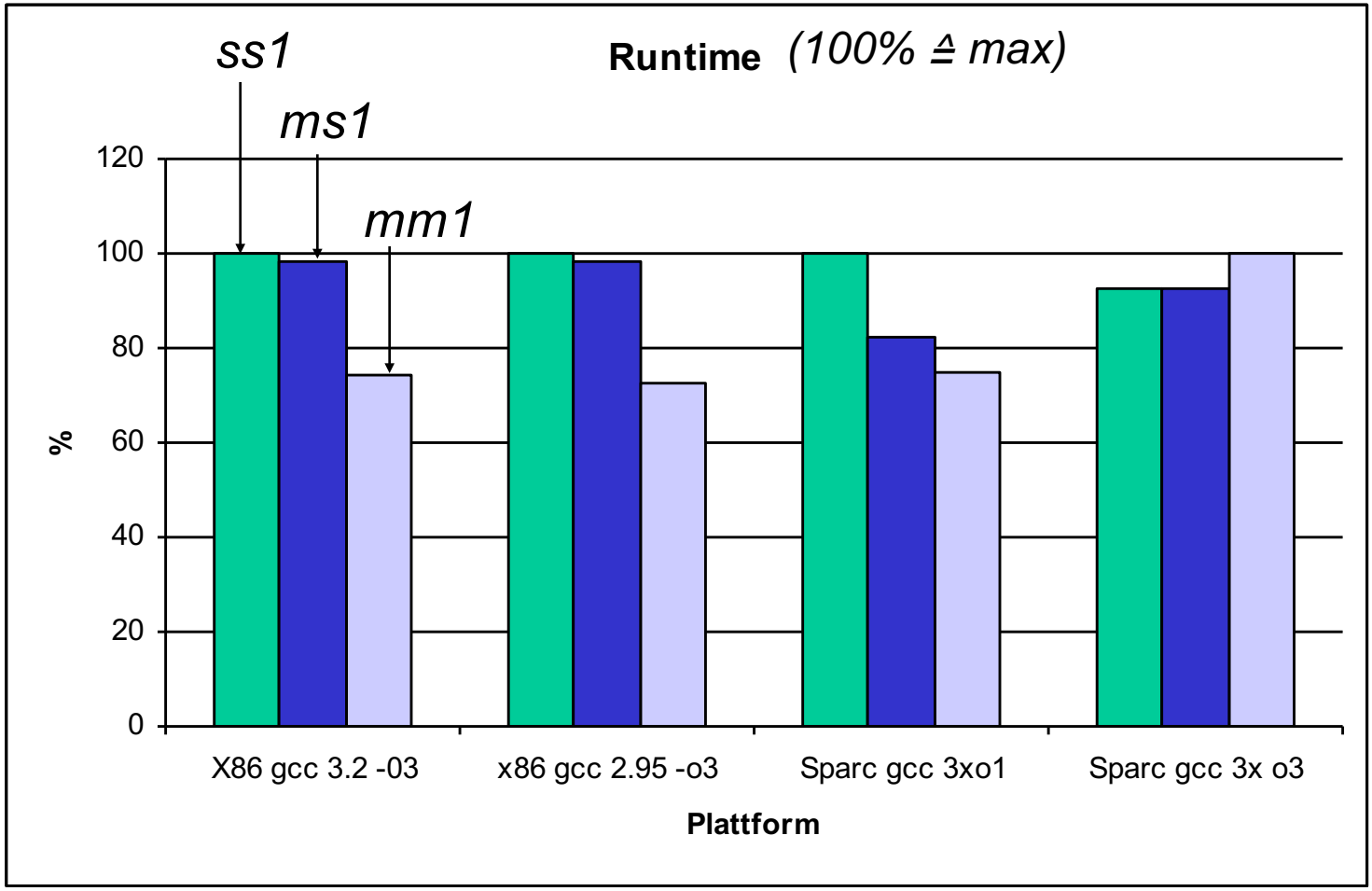
```
#define size 30
#define iter 40000
int a[size][size];
float b[size][size];
```

```
void ss1() {int i,j;
 for (i=0;i<size;i++){
  for (j=0;j<size;j++){
   a[i][j]+= 17;}}
 for(i=0;i<size;i++){
  for (j=0;j<size;j++){
   b[i][j]-=13;}}}
```

```
void ms1() {int i,j;
 for (i=0;i< size;i++){
  for (j=0;j<size;j++){
   a[i][j]+=17;     }
  for (j=0;j<size;j++){
   b[i][j]-=13; }}}
```

```
void mm1() {int i,j;
 for(i=0;i<size;i++){
  for(j=0;j<size;j++){
   a[i][j] += 17;
   b[i][j] -= 13;}}}
```

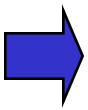
Results: simple loops



☞ *Merged loops superior; except Sparc with –o3*

Loop unrolling

```
for (j=0; j<=n; j++)
p[j]= ... ;
```



```
for (j=0; j<=n; j+=2)
    {p[j]= ... ; p[j+1]= ...}
```

factor = 2

Better locality for access to p.

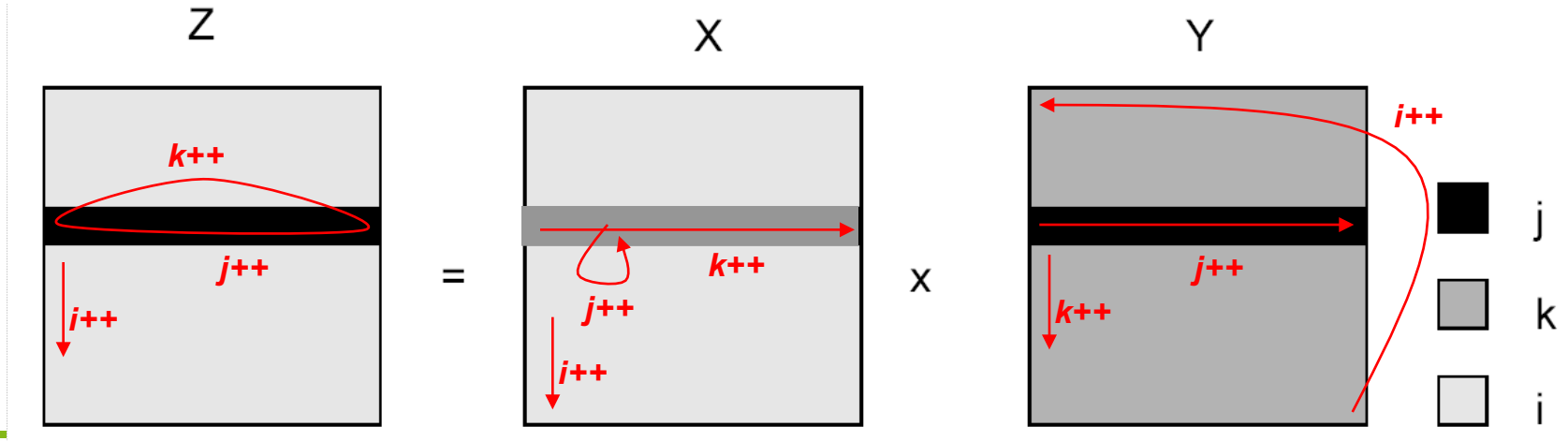
Less branches per execution of the loop. More opportunities for optimizations.

Tradeoff between code size and improvement.

Extreme case: completely unrolled loop (no branch).

Program transformation
Loop tiling/loop blocking: - Original version -

```
for (i=1; i<=N; i++)
        for(k=1; k<=N; k++){
                r=X[i,k]; /* to be allocated to a register*/
                for (j=1; j<=N; j++)
                        Z[i,j] += r* Y[k,j]
} % Never reusing information in the cache for Y and Z if N
is large or cache is small (O(N³) references for Z).
```



Loop tiling/loop blocking
- tiled version -

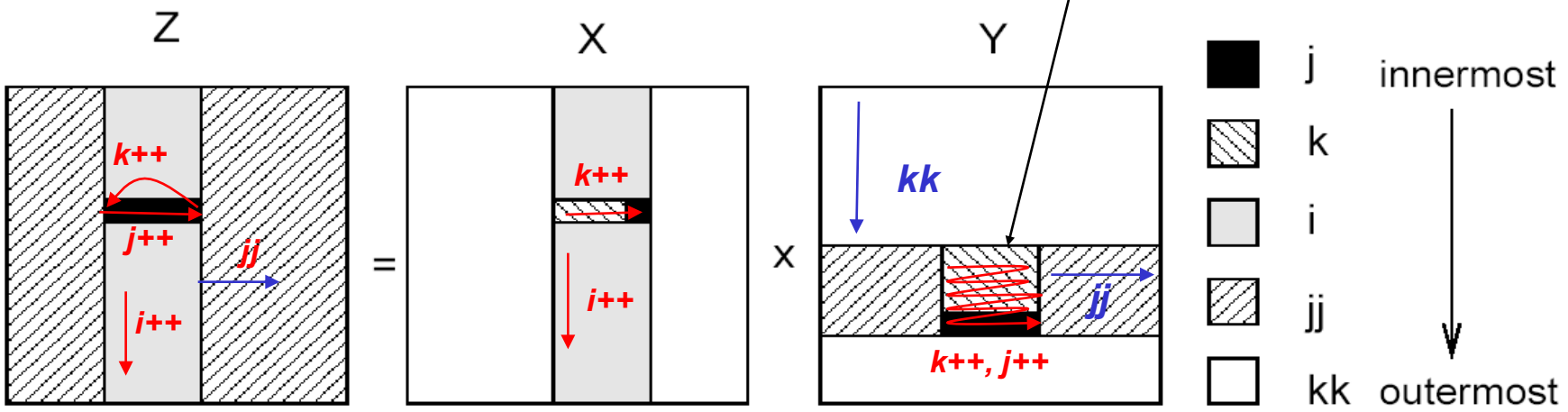
```
for (kk=1; kk<= N; kk+=B)
  for (jj=1; jj<= N; jj+=B)
    for (i=1; i<= N; i++)
      for (k=kk; k<= min(kk+B-1,N); k++){
        r=X[i][k]; /* to be allocated to a register*/
        for (j=jj; j<= min(jj+B-1, N); j++)
          Z[i][j] += r* Y[k][j]
        }
```

Reuse factor of B for Z, N for Y

O(N³/B) accesses to main memory

Compiler should select best option

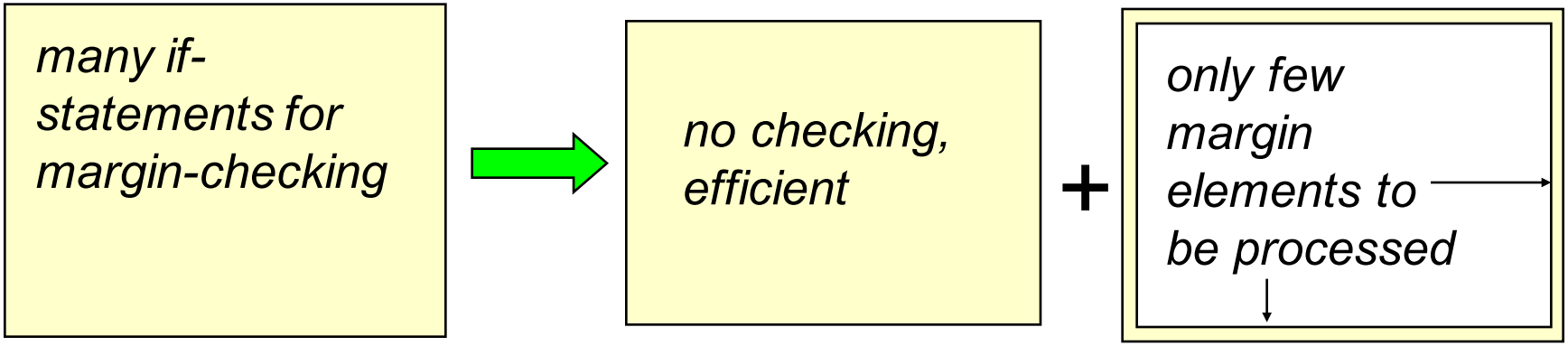
Same elements for next iteration of i



Monica Lam: The Cache Performance and Optimization of Blocked Algorithms, ASPLOS, 1991

Transformation “Loop nest splitting”

Example: Separation of margin handling



Outline

High-Level Optimization
- *Loop transformation*
- *Loop tiling/blocking*
- *Loop (nest) splitting*

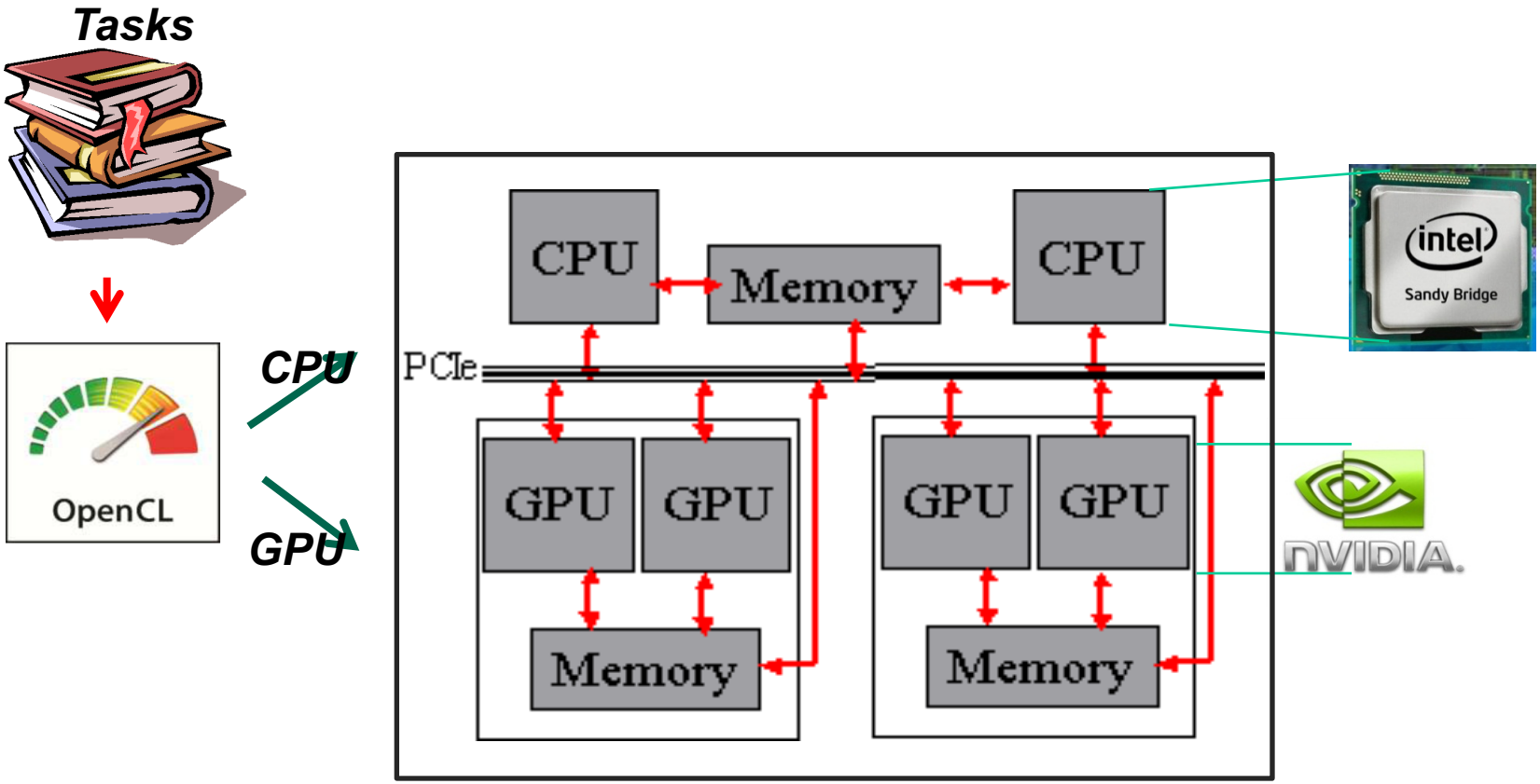
Heterogeneous System Architecture (HSA)
- *Integrated CPU/GPU platforms*
- *Recent movement in chip designs*

Architecture-aware software designs
- *Energy-efficiency issues*
- *Darkroom*
- *Halide*

Multicore revolutions
- *Impact on the "safety-critical" industry sector*

What is Heterogeneous Computing?

Use processor cores with various type/computing power to achieve better performance/power efficiency



http://www.browndeertechnology.com/docs/BDT_OpenCL_Tutorial_NBody-rev3.html

Advantage of Heterogeneous Computing

CPU is ideal for scalar processing

- Out of order x86 cores with low latency memory access
- Optimized for sequential and branching algorithms
- Runs existing applications very well

Serial/Task-parallel workloads → CPU

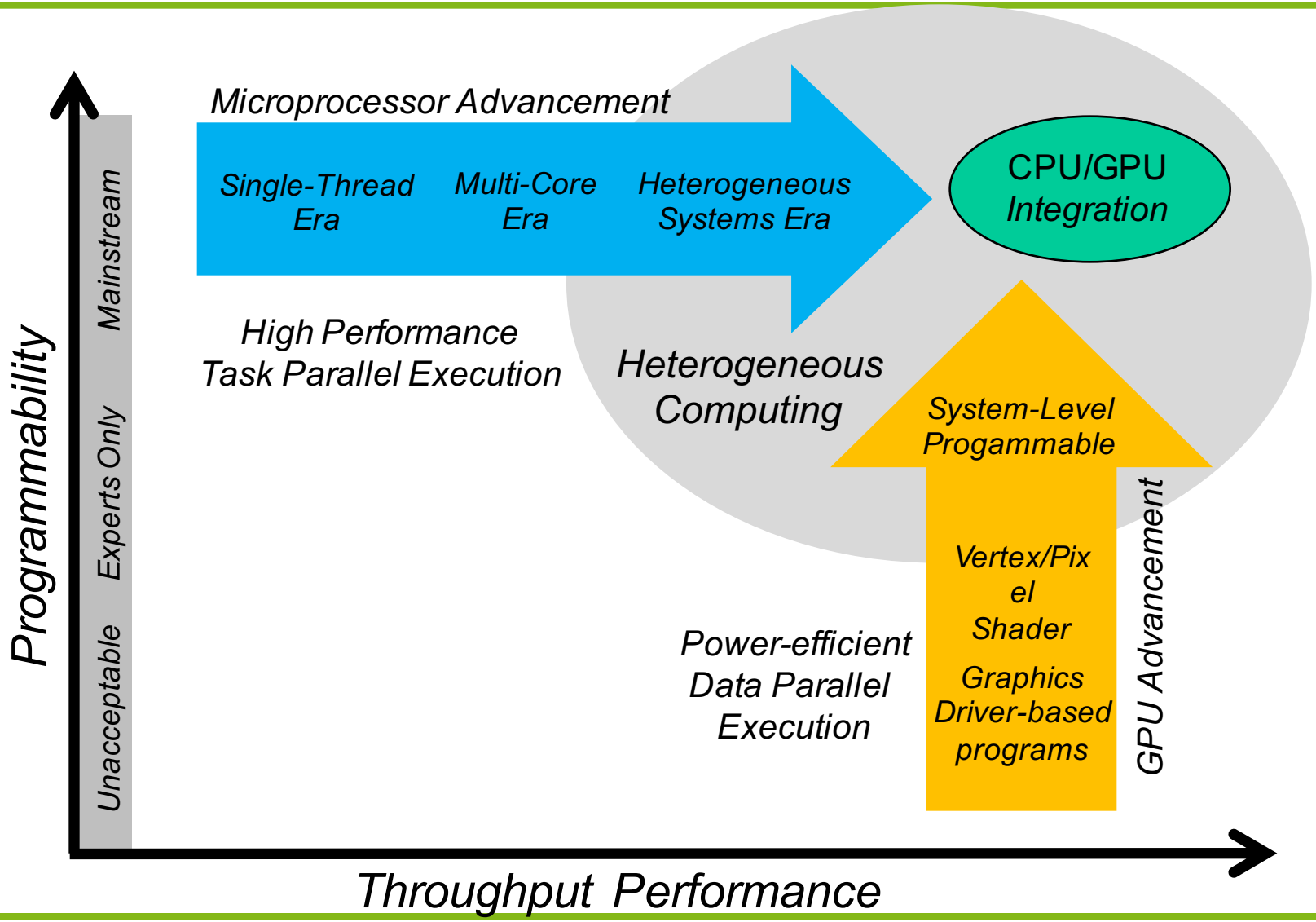
GPU is ideal for parallel processing

- GPU shaders optimized for throughput computing
- Ready for emerging workloads
- Media processing, simulation, natural UI, etc.

Graphics/Data-parallel workloads → GPU

Heterogeneous Computing -> Fusion, Norm Rubin, SAAHPC 2010

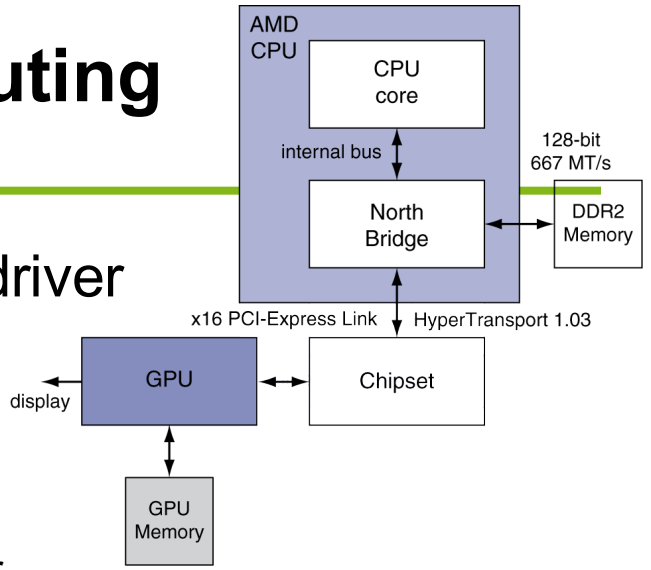
CPU/GPU Integration: CPU's Advancement Meets GPU's

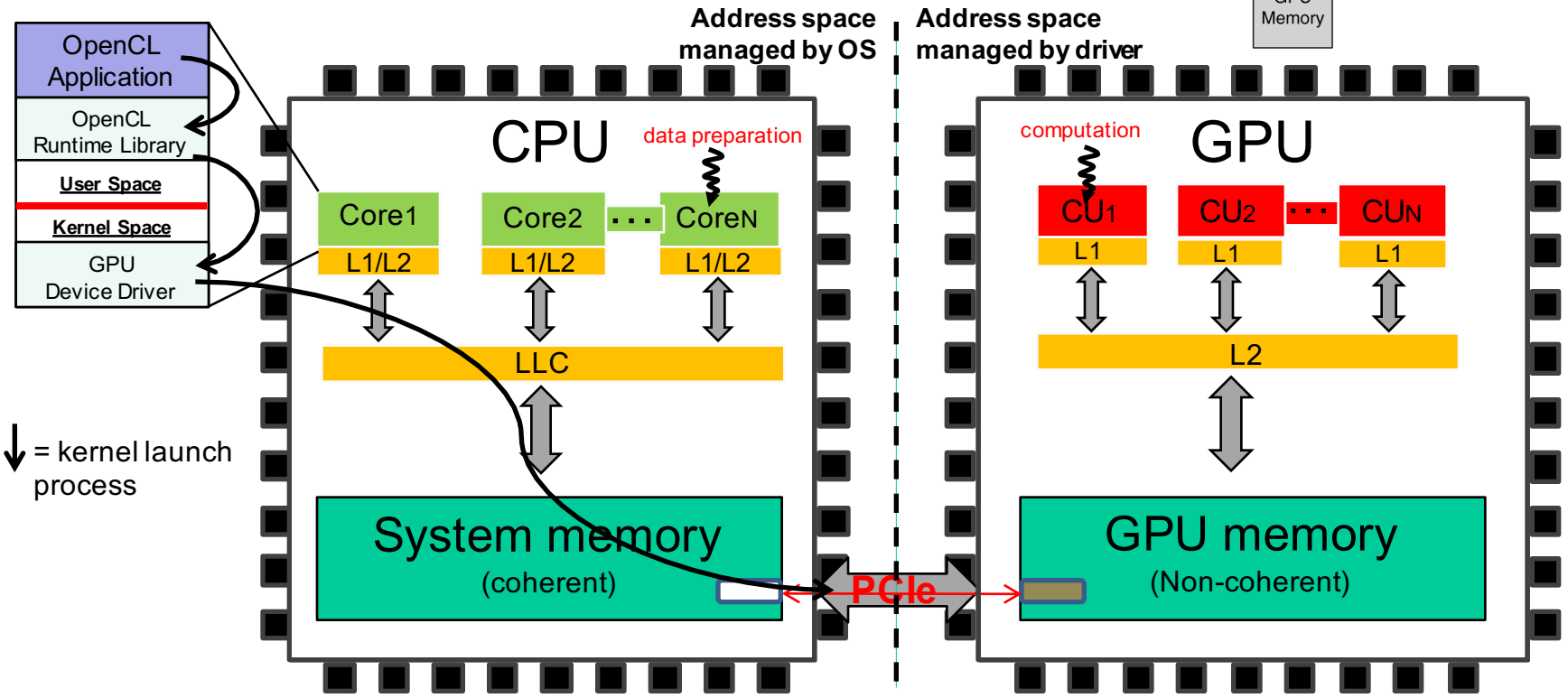


Evolution of Heterogeneous Computing

Dedicated GPU

- GPU kernel is launched through the device driver
- Separate CPU/GPU address space
- Separate system/GPU memory
- Data copy between CPU/GPU via PCIe

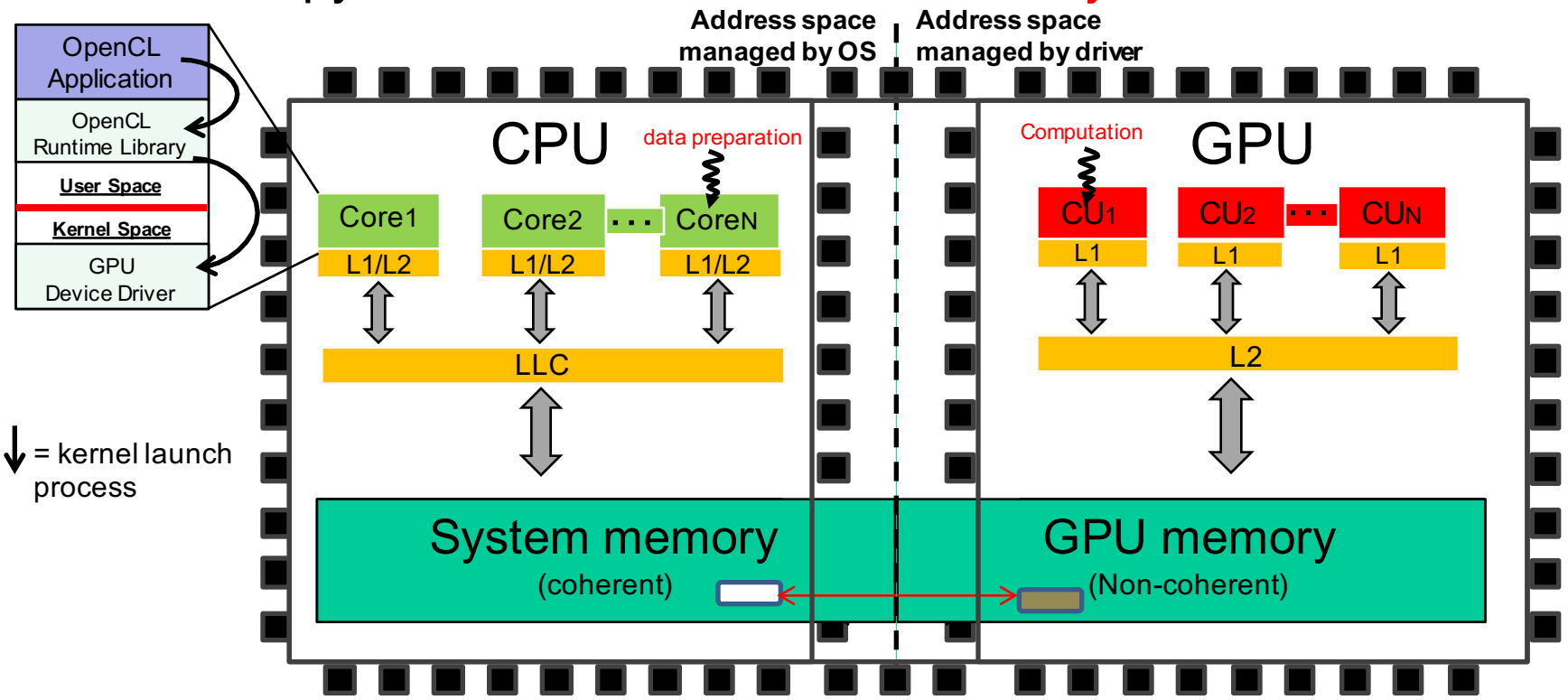




Evolution of Heterogeneous Computing

Integrated GPU architecture

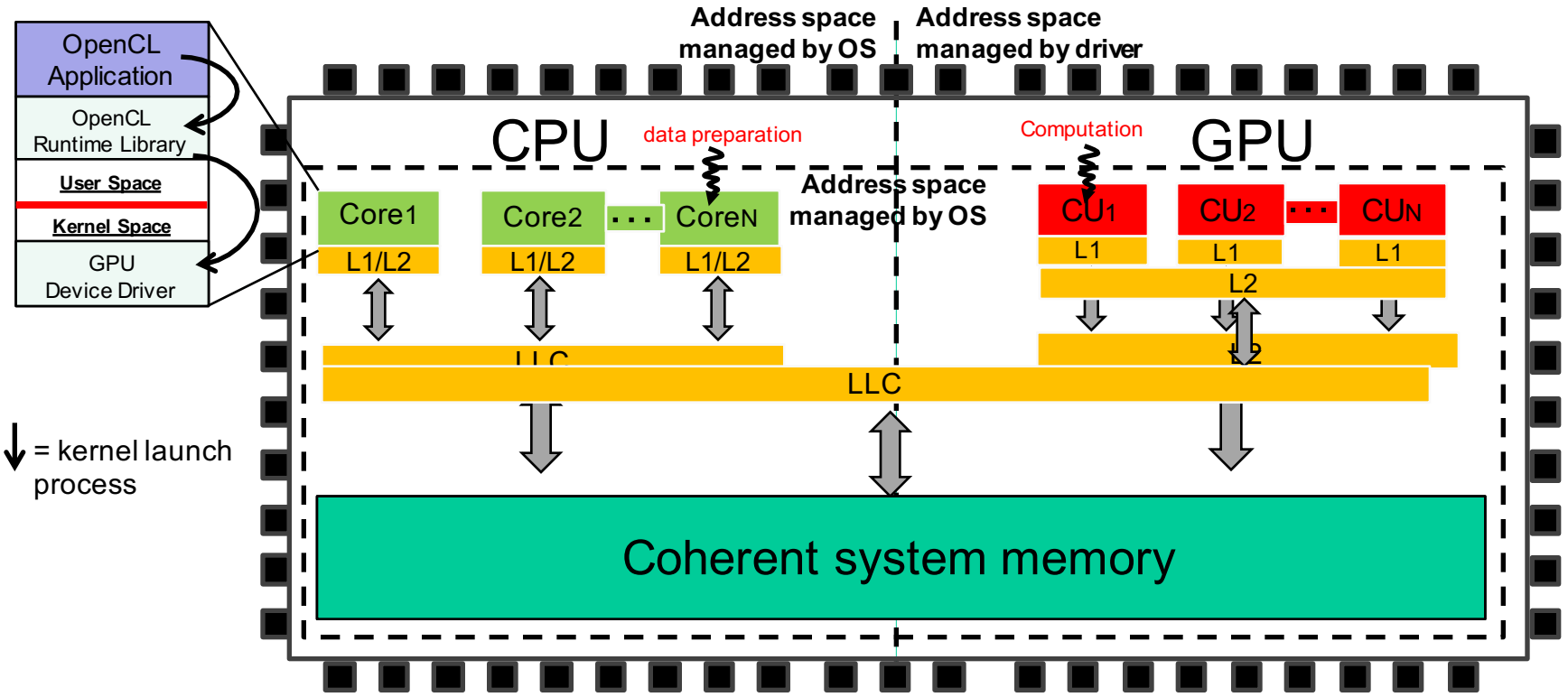
- GPU kernel is launched through the device driver
- Separate CPU/GPU address space
- Separate system/GPU memory
- Data copy between CPU/GPU *via memory bus*



Evolution of Heterogeneous Computing

Integrated CPU/GPU architecture

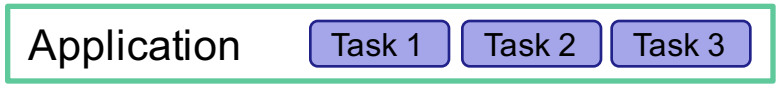
- GPU kernel is launched through the device driver
- Unified CPU/GPU address space (managed by OS)
- Unified system/GPU memory
- No data copy - data can be retrieved by pointer passing

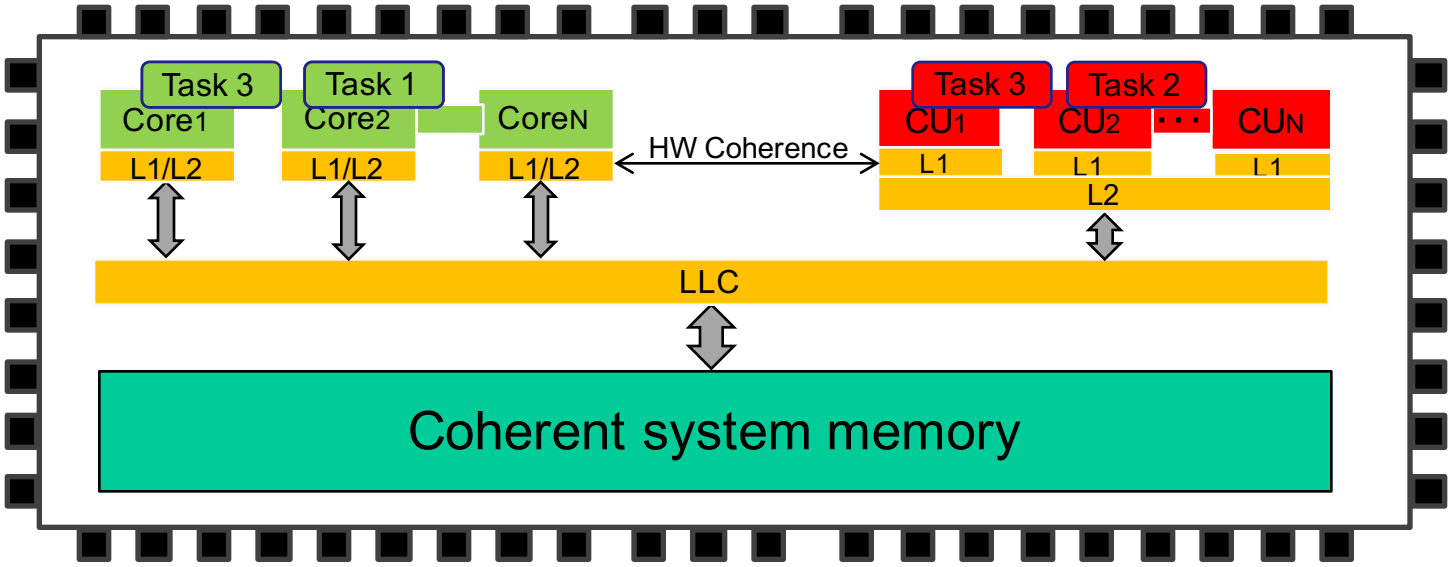


Utopia World of Heterogeneous Computing

Processors are architected to operate cooperatively
- Tasks in an application are executed on different types of core
- Unified coherent memory enables data sharing across all processors

Designed to enable the applications to run on different processors at different time
- Capability to translate from high-level language to target binary at run-time
- User-level task dispatch
- Decision making module





HSA Foundation

Founded in June 2012

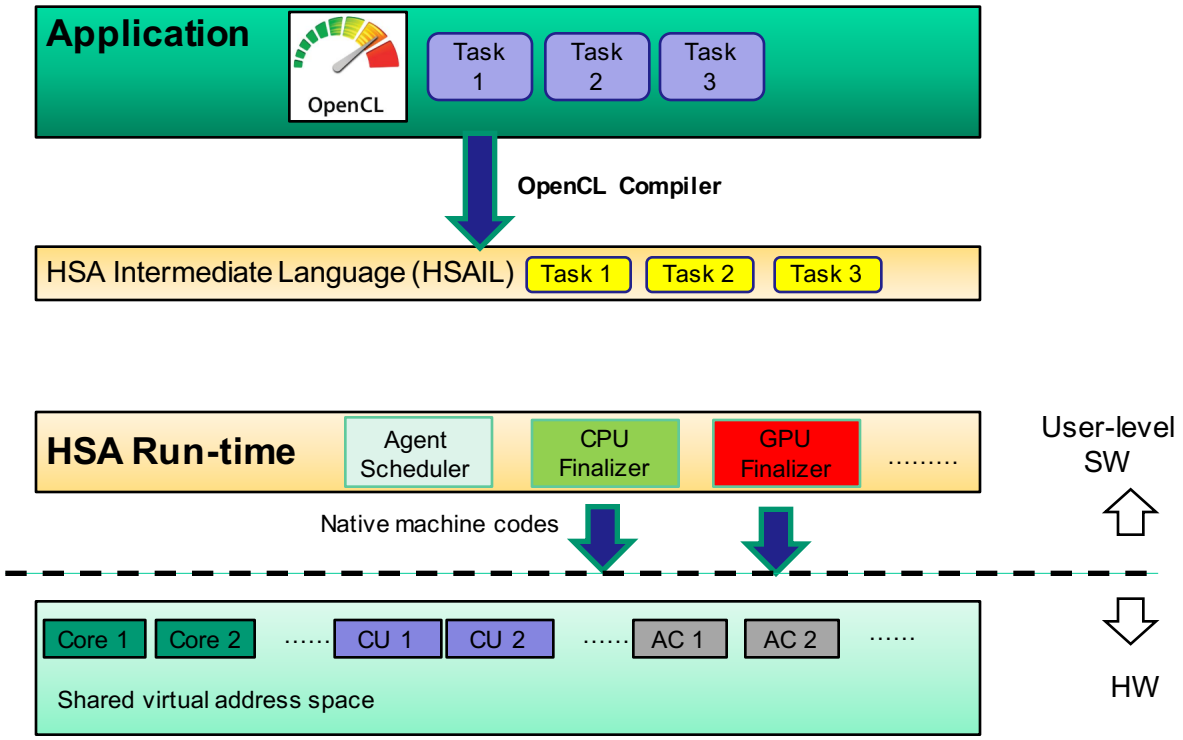
Developing a new platform for heterogeneous systems

www.hsafoundation.com

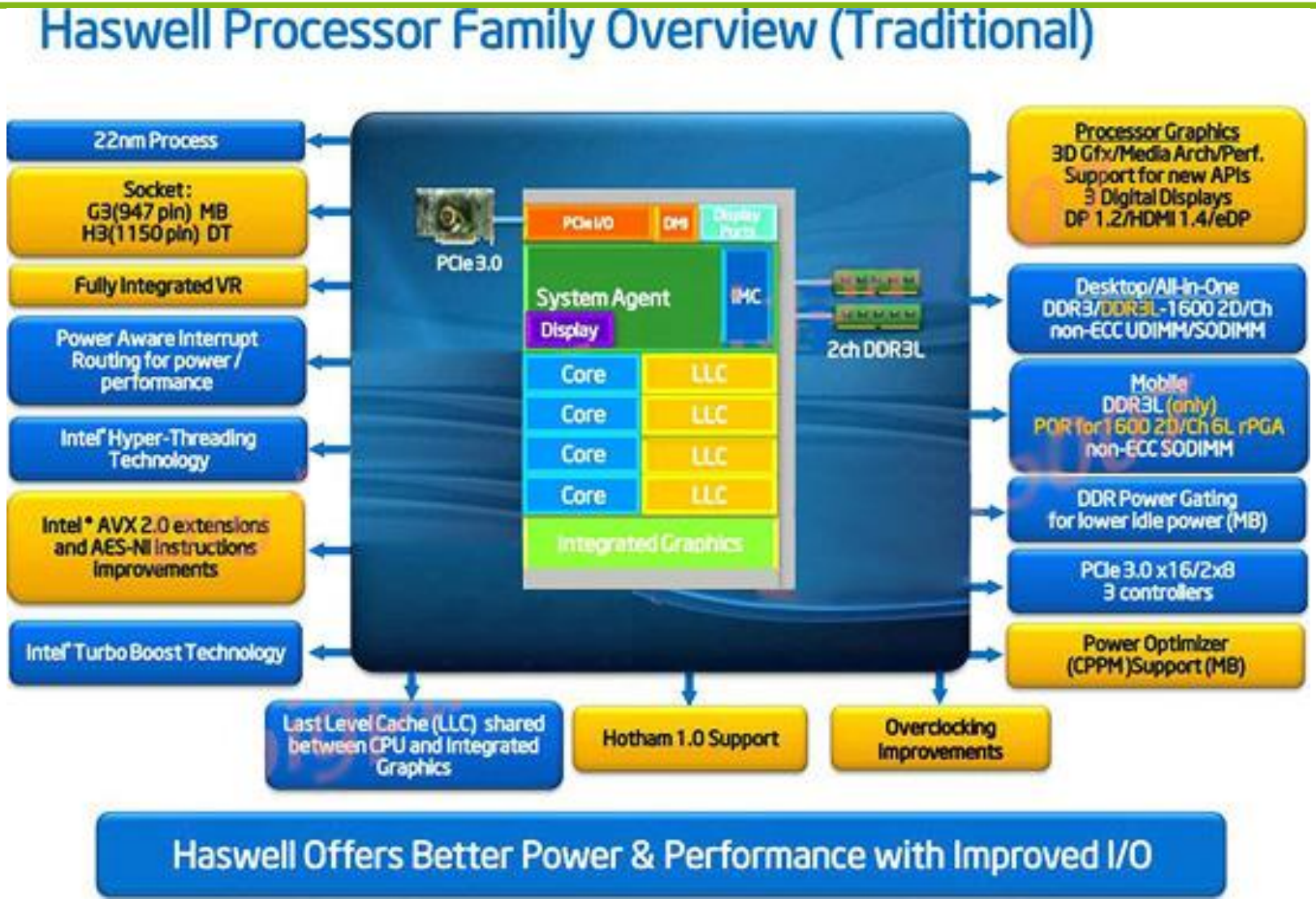
Specifications under development in working groups to define the platform

Diverse Partners Driving Future of Heterogeneous Computing

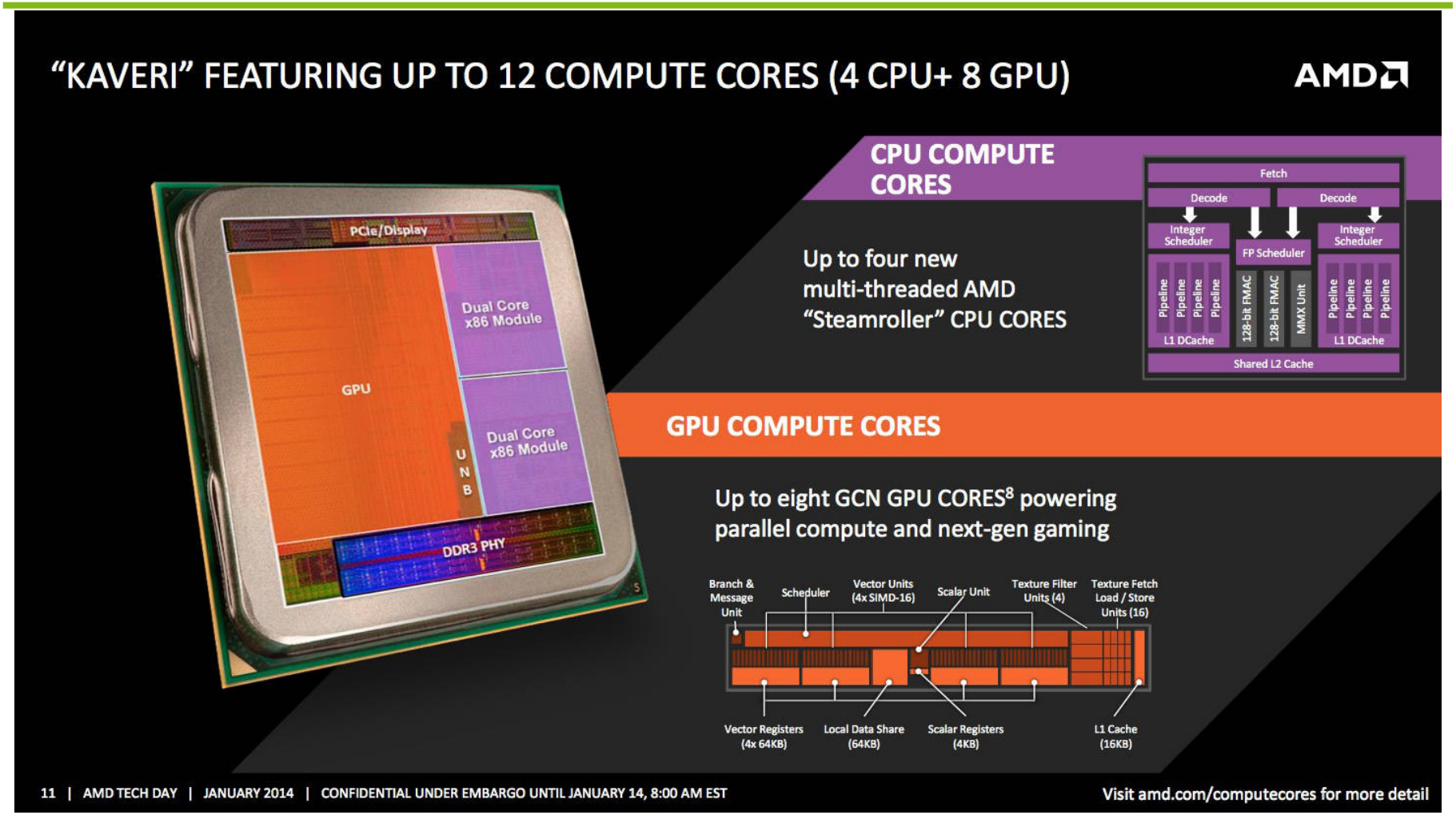
HSA (Heterogeneous System Architecture) Hardware-Software Stack



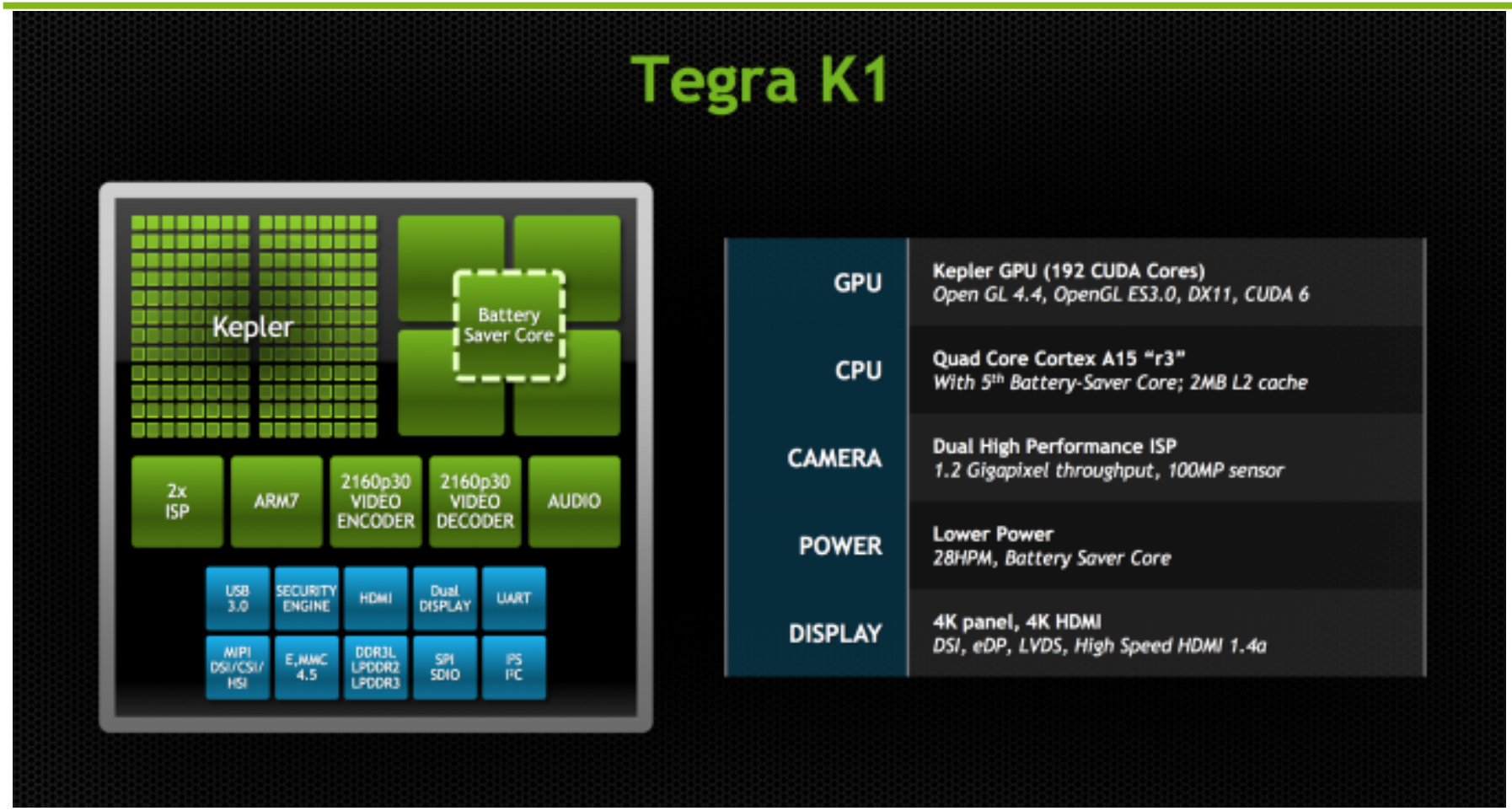
Intel Haswell



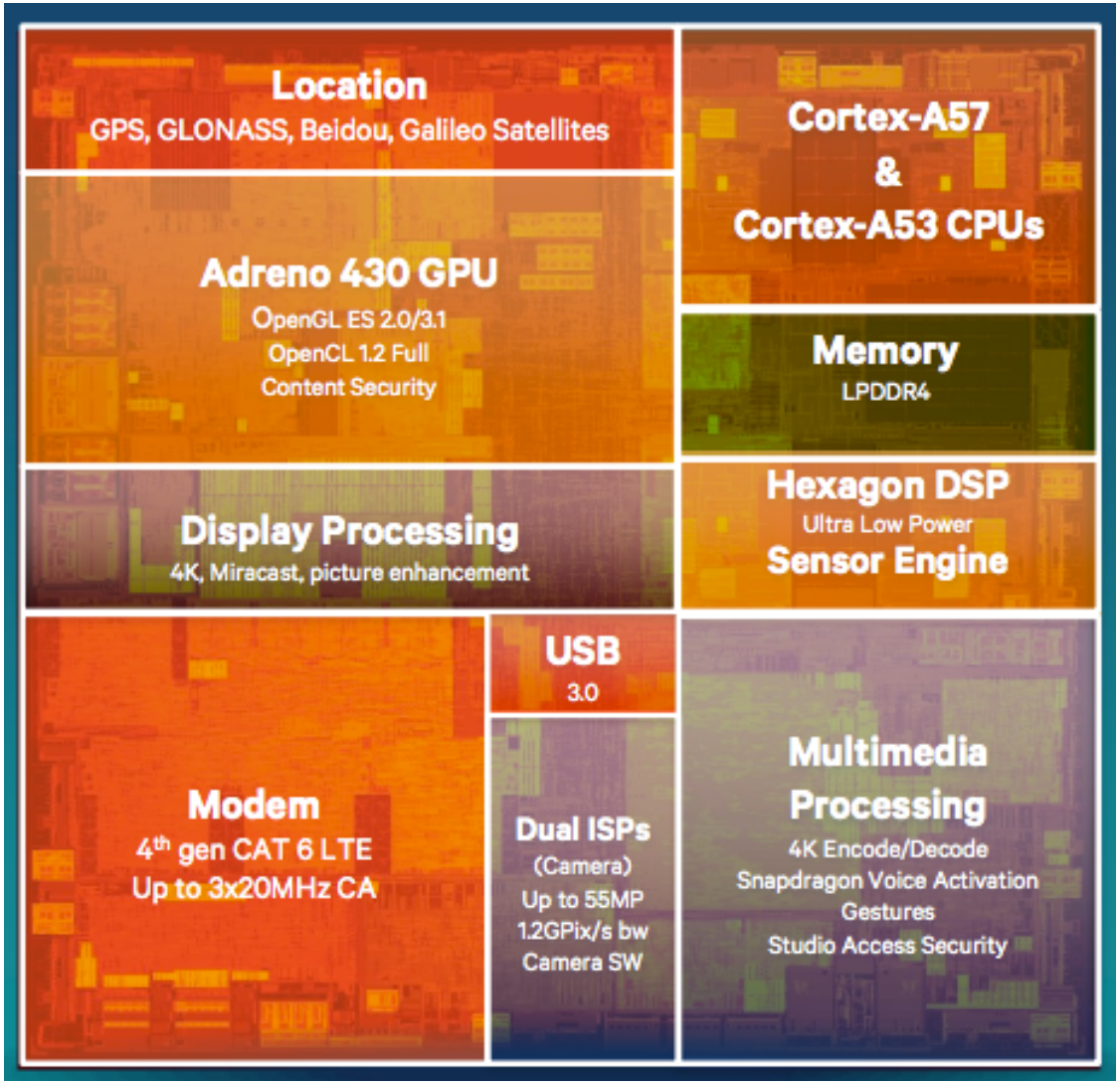
AMD Kaveri APU



NVIDIA Tegra K1



Qualcomm Snapdragon



Outline

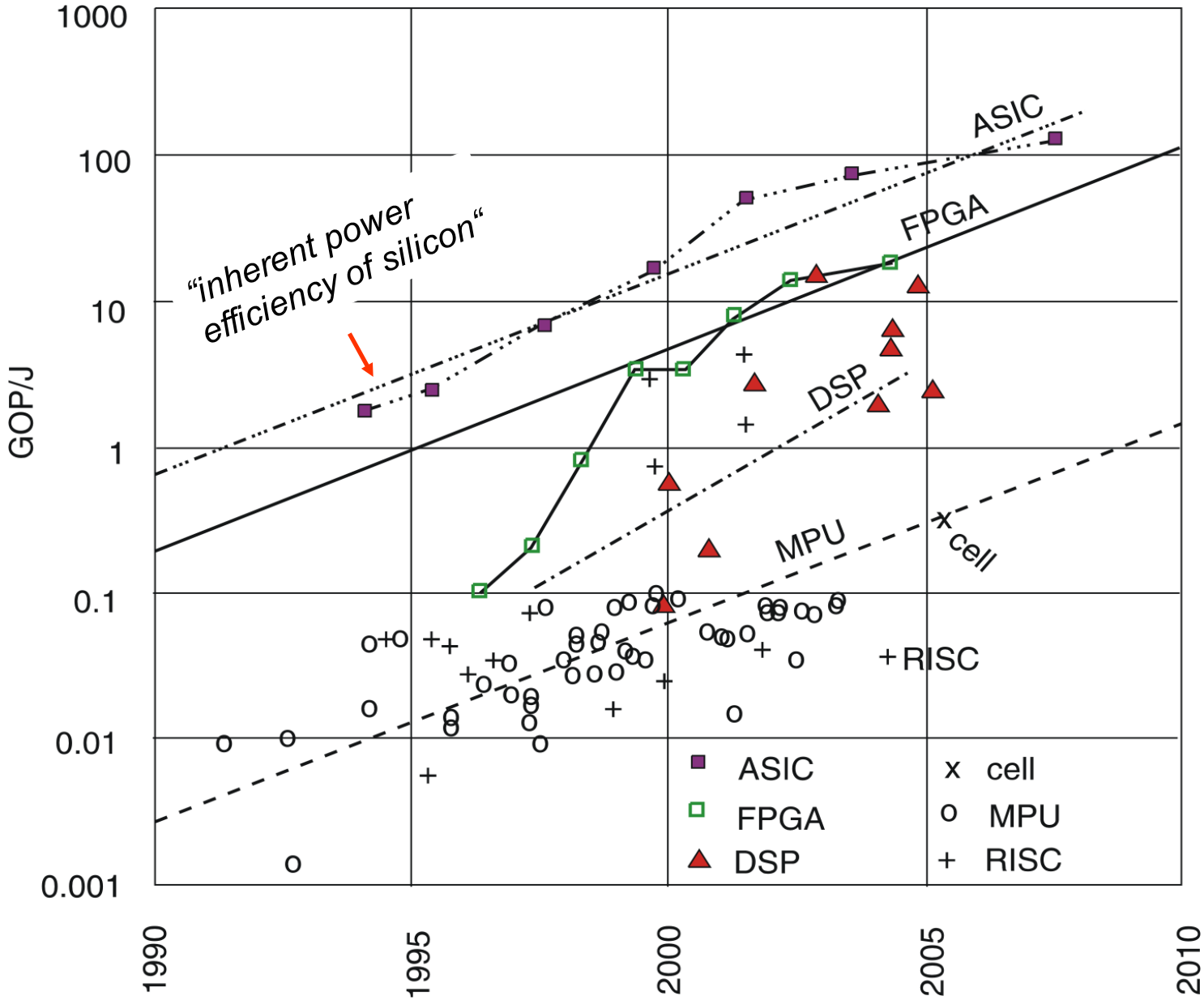
High-Level Optimization
- *Loop transformation*
- *Loop tiling/blocking*
- *Loop (nest) splitting*

Heterogeneous System Architecture (HSA)
- *Integrated CPU/GPU platforms*
- *Recent movement in chip designs*

Architecture-aware software designs
- *Energy-efficiency issues*
- *Darkroom*
- *Halide*

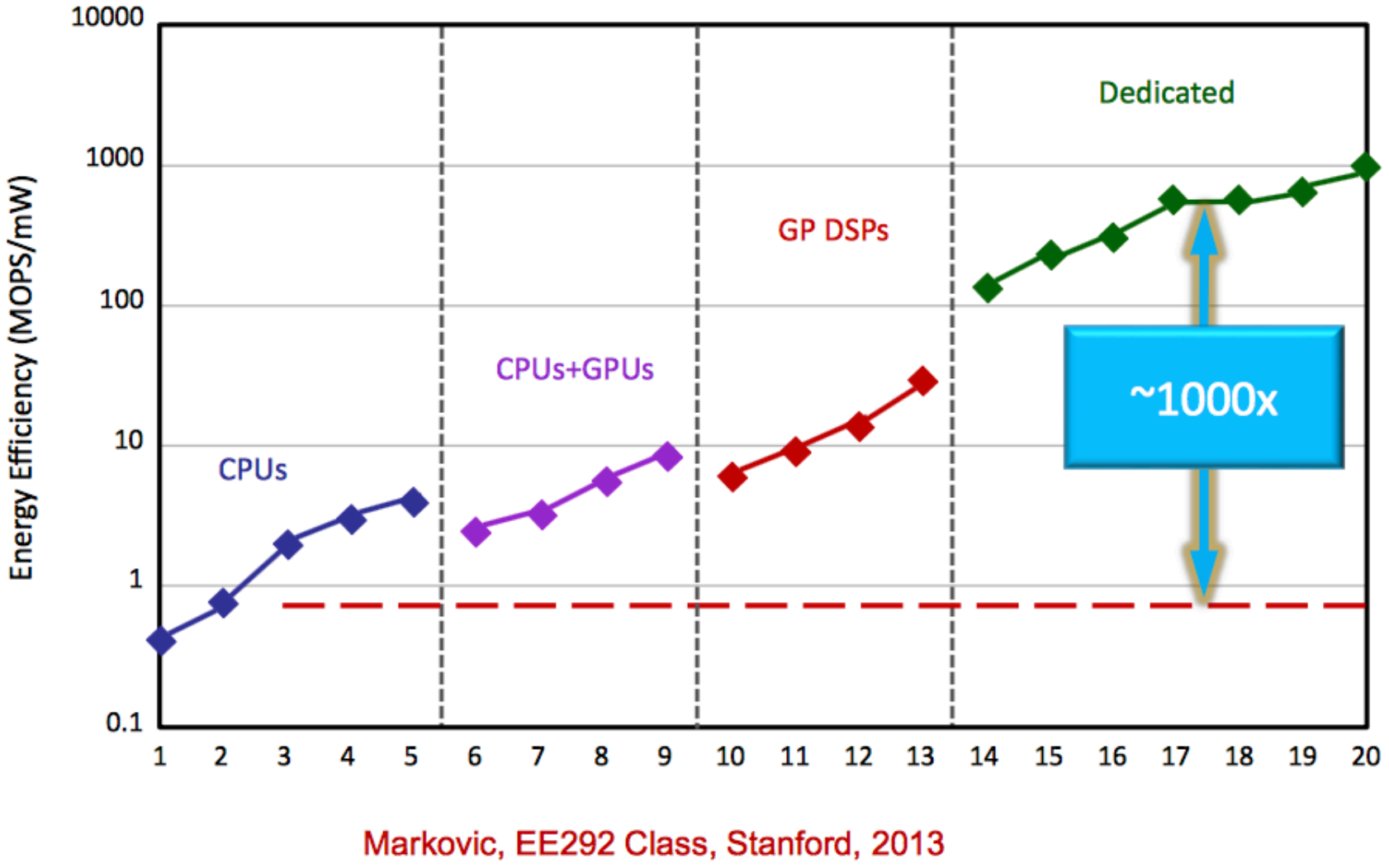
Multicore revolutions
- *Impact on the "safety-critical" industry sector*

Energy Efficiency of different target platforms



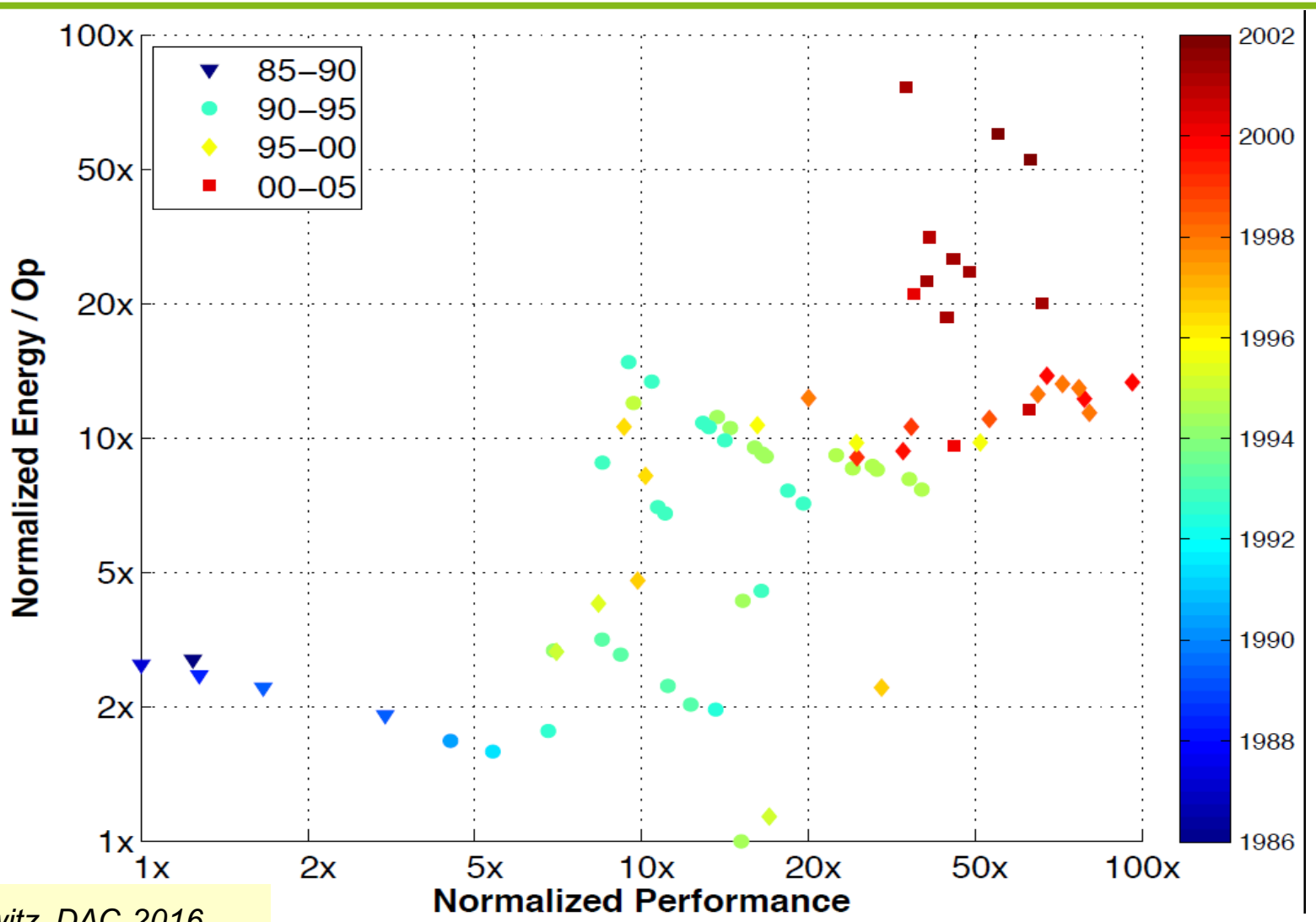
© Hugo De Man, IMEC, Philips, 2007

Signal Processing ASICs



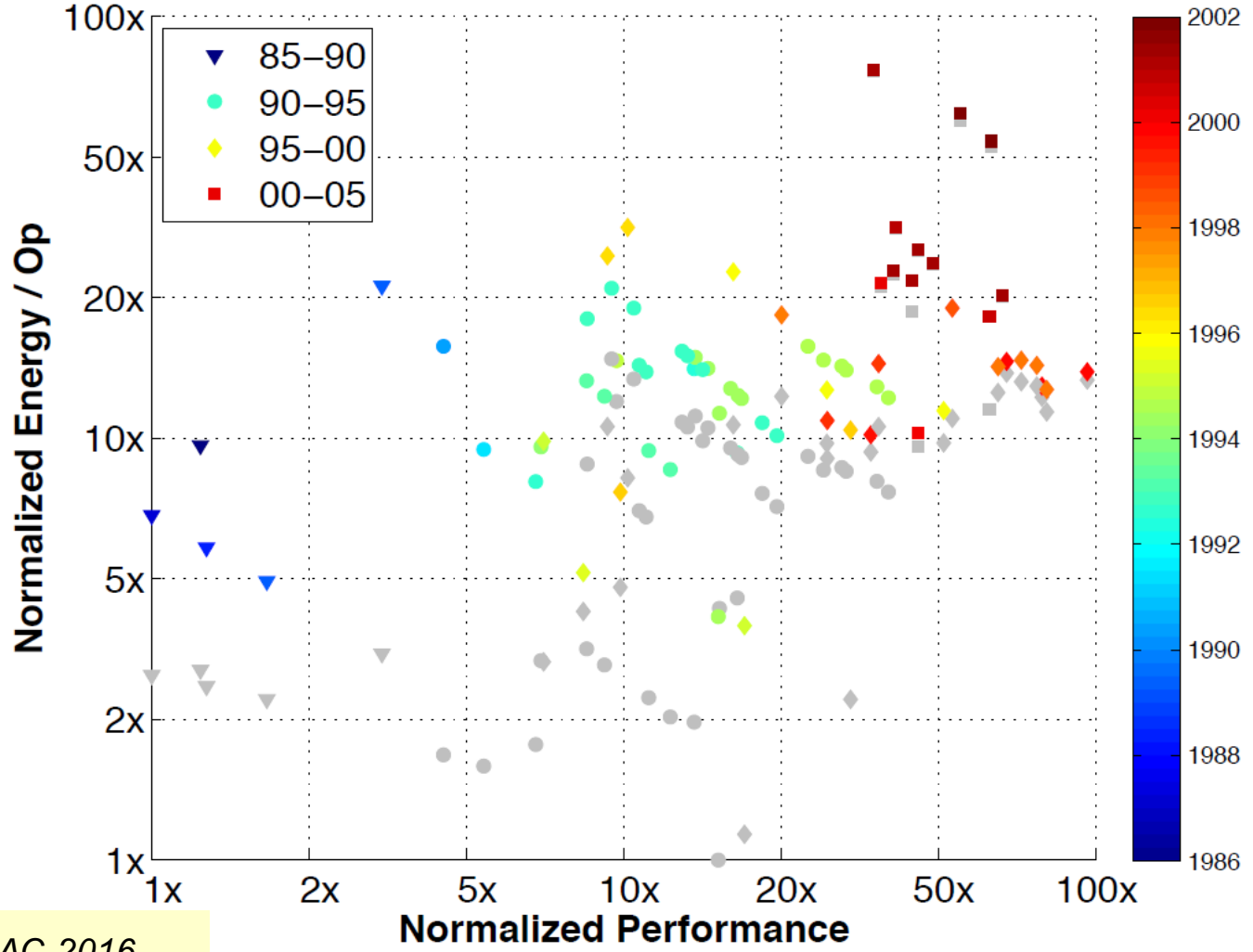
Markovic, EE292 Class, Stanford, 2013

How about Memory?



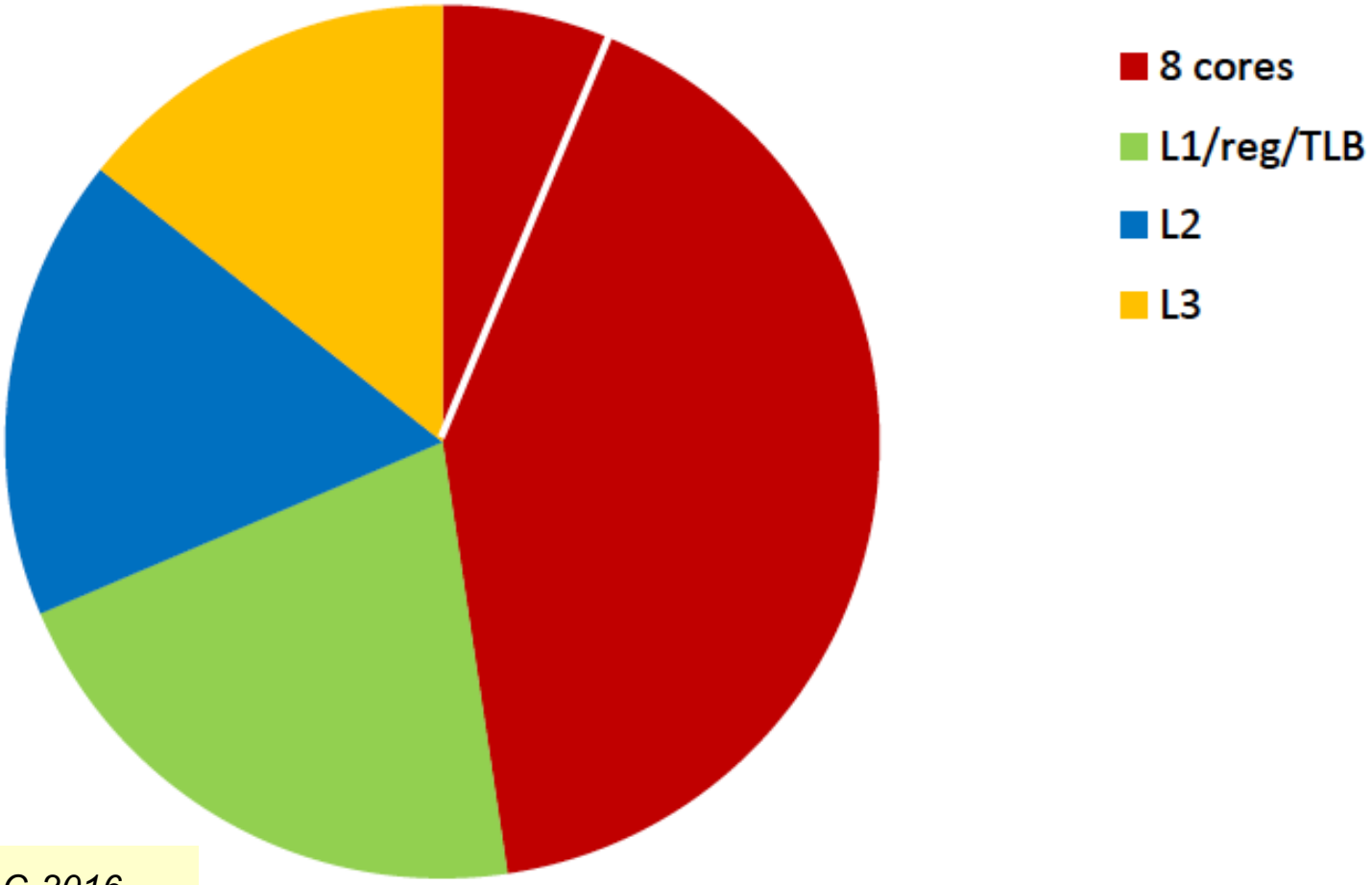
© *Horowitz, DAC 2016*

Processor Energy with Corrected Cache Sizes



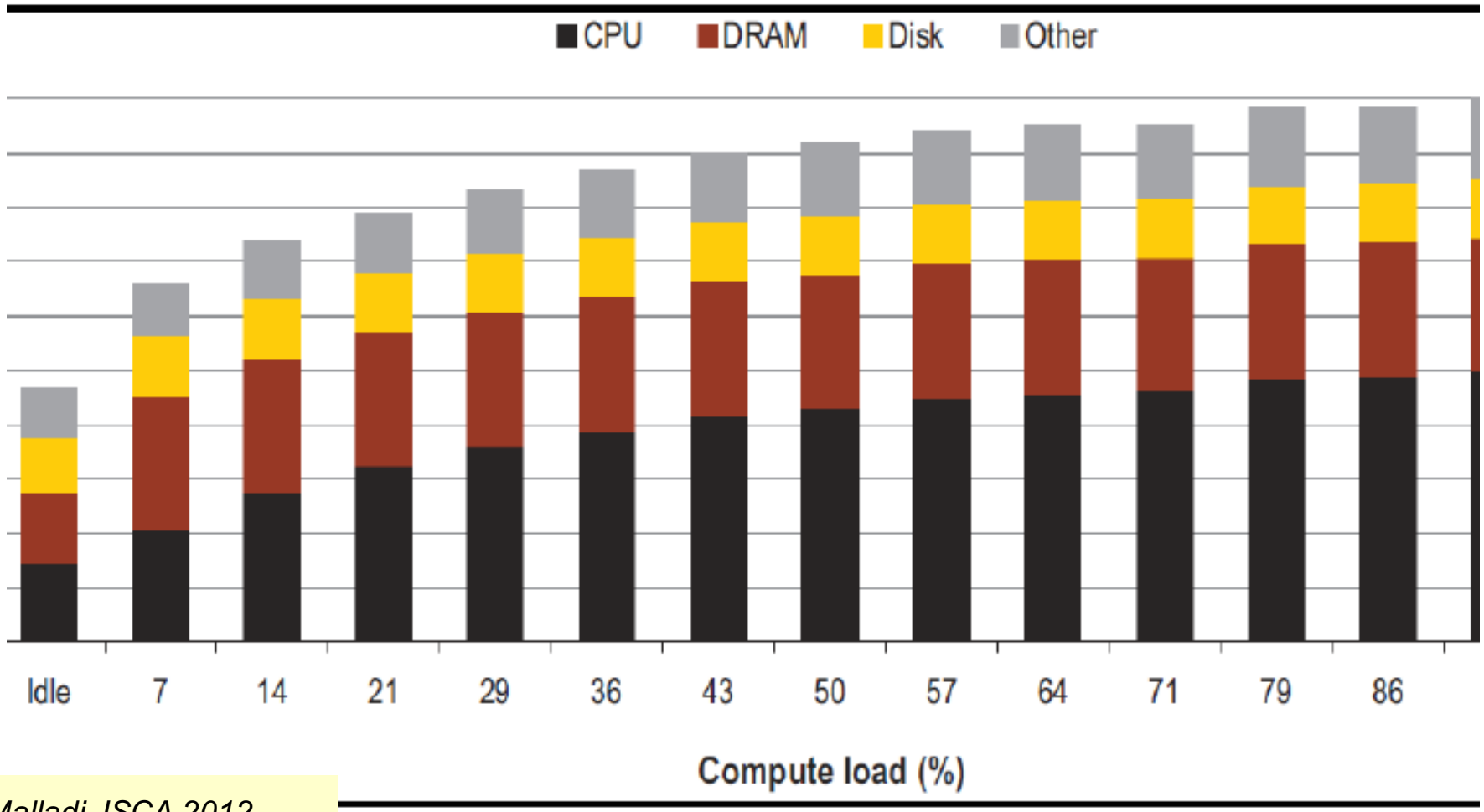
© Horowitz, DAC 2016

Processor Energy Breakdown



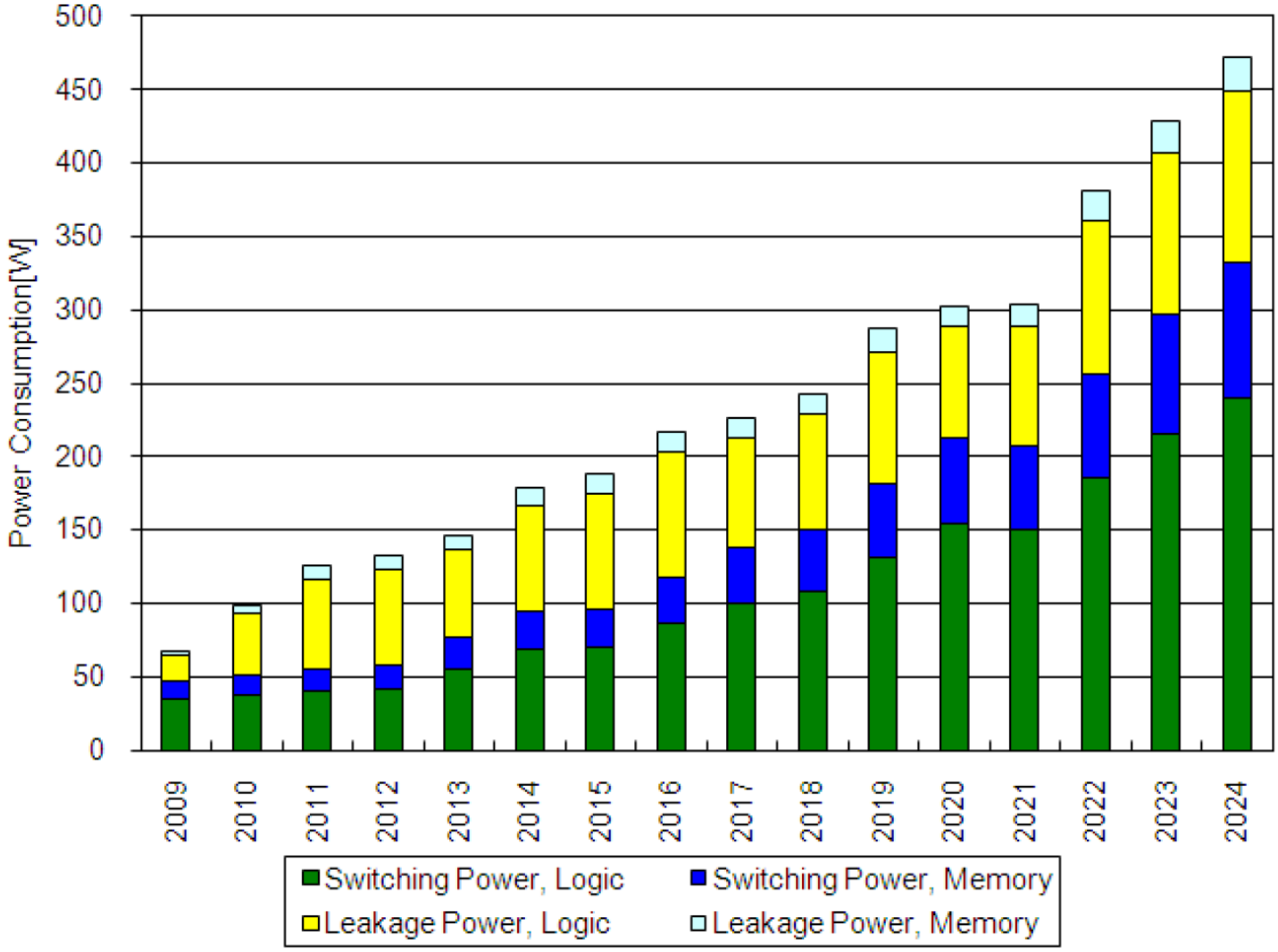
© Horowitz, DAC 2016

Data Center Energy Specs

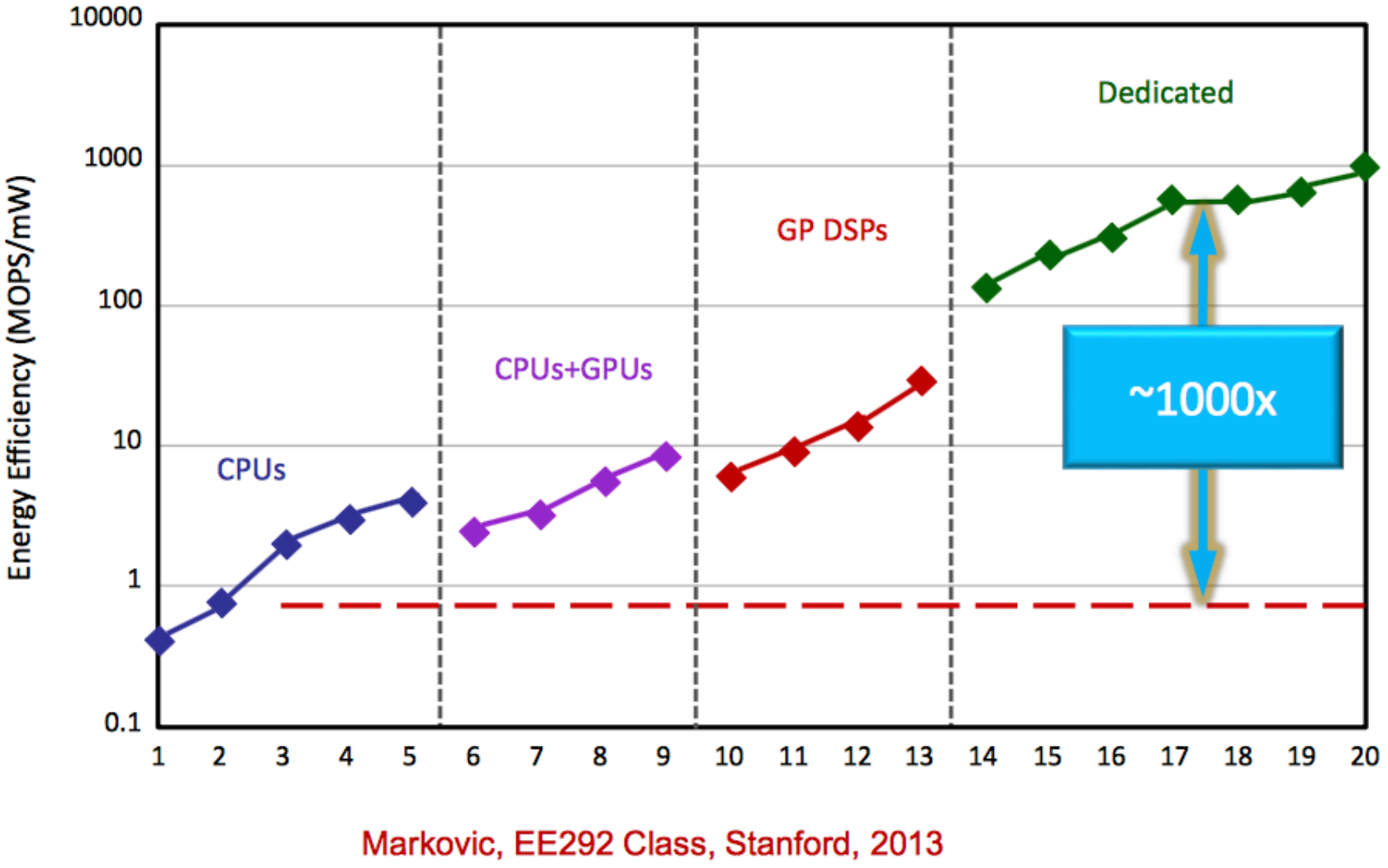


© *Malladi, ISCA 2012*

ITRS Power Consumption Projection -Station Systems -



What Is Going On Here?



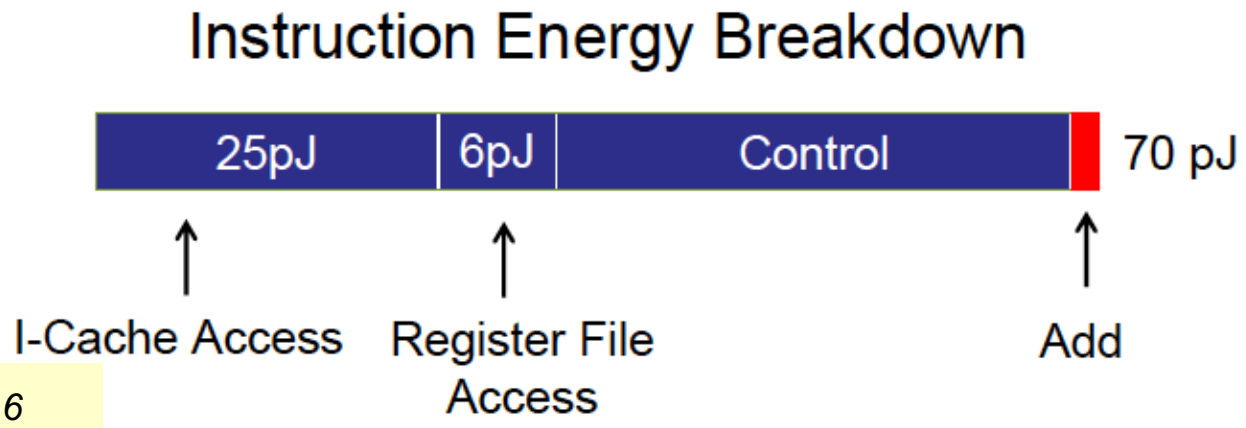
Markovic, EE292 Class, Stanford, 2013

Energy Consumption (Approximate, 45nm)

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3 pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1pJ
32 bit	4pJ

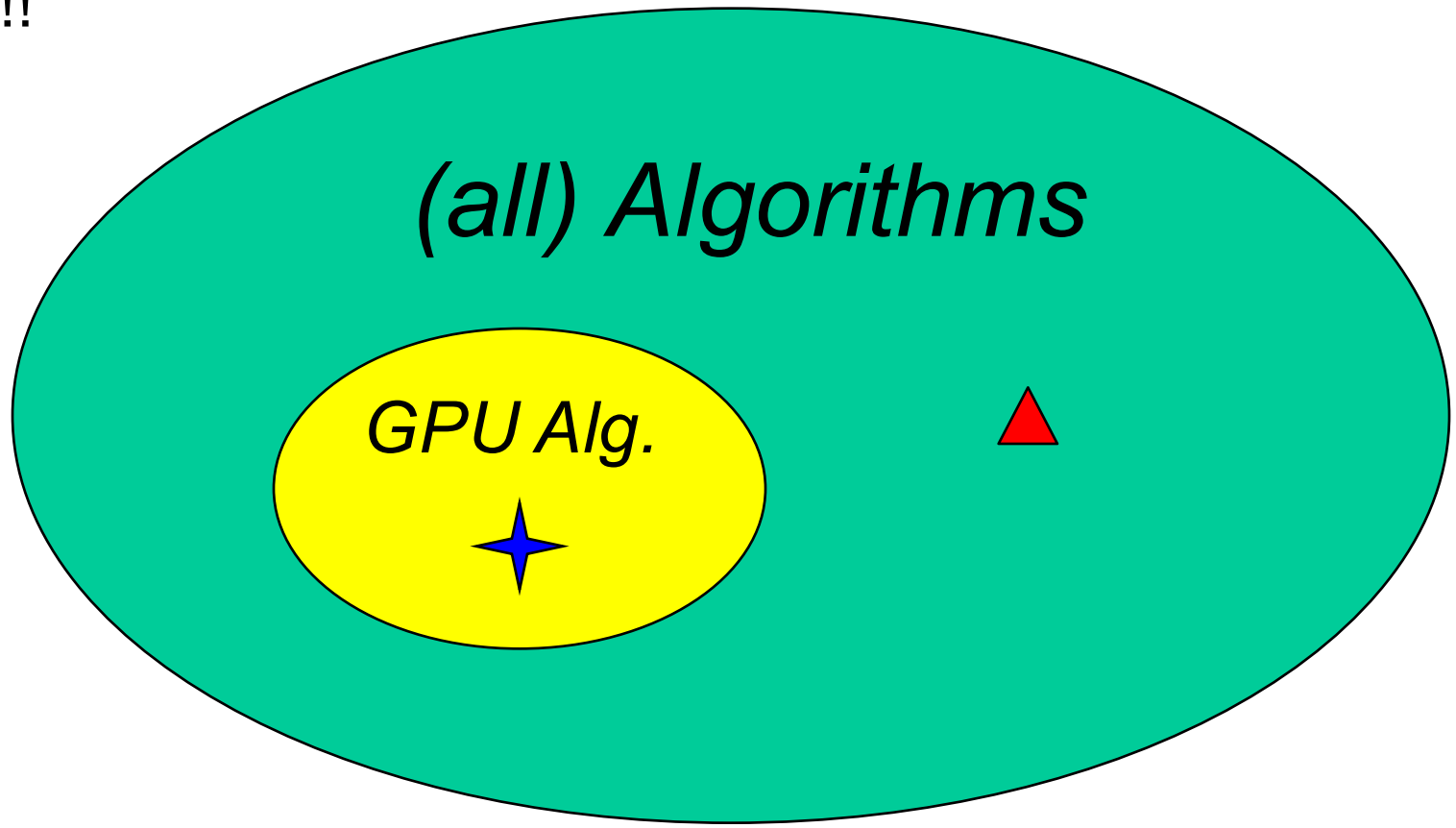
Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ



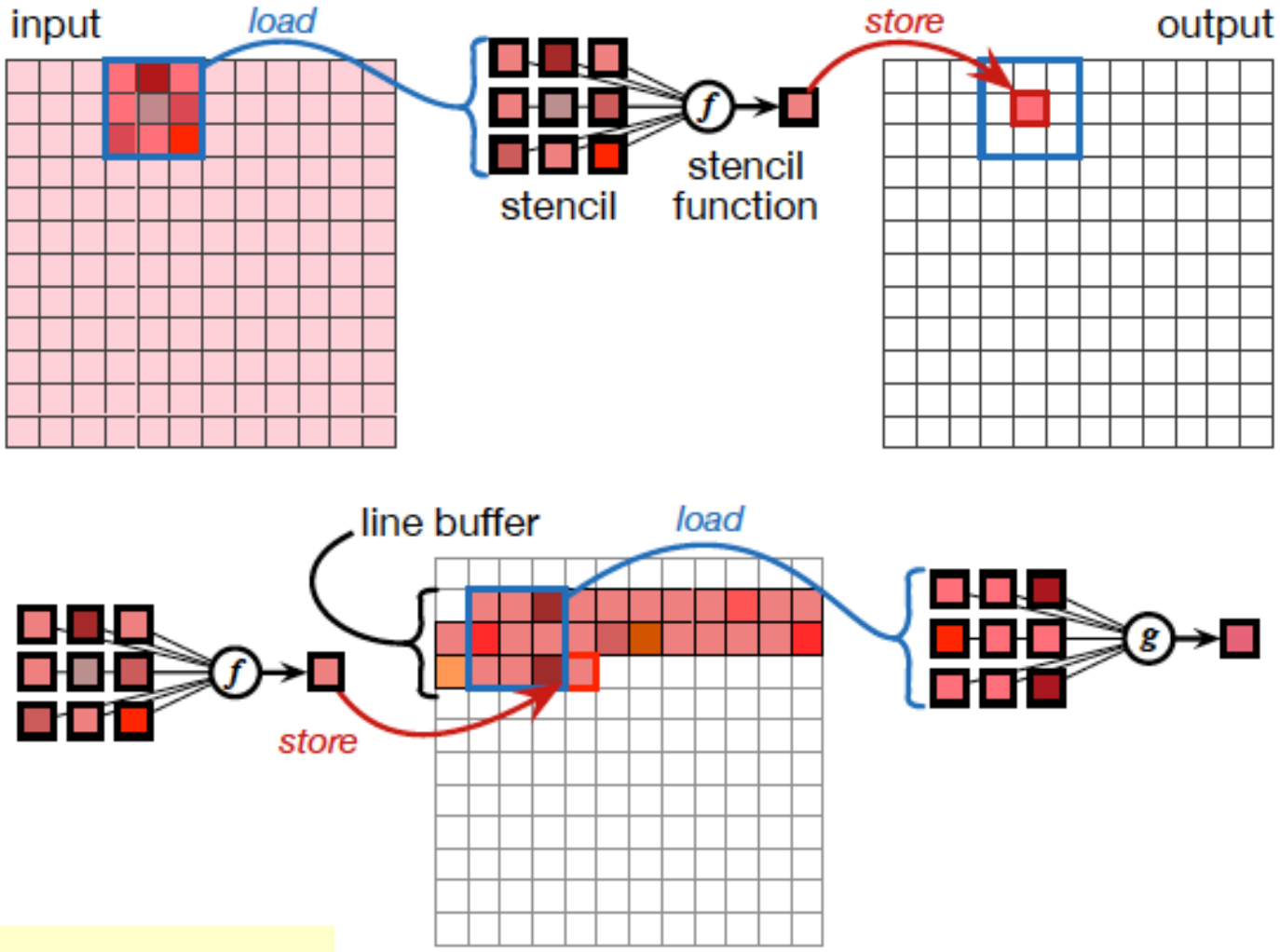
© Horowitz, DAC 2016

True Story

It's more about the algorithm than the hardware
The efficiency cannot be achieved unless the algorithm is right!!

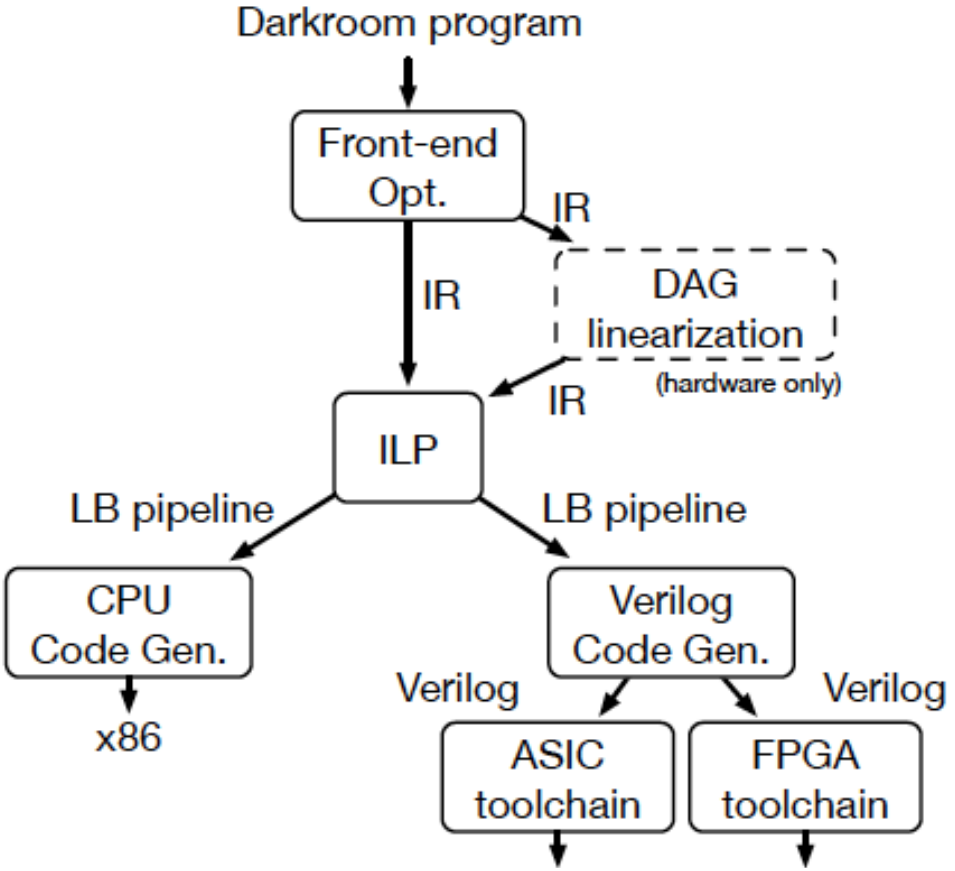


Locality, Locality, and Locality!!!



© *Hegarty et al. , SIGGraph 2014*

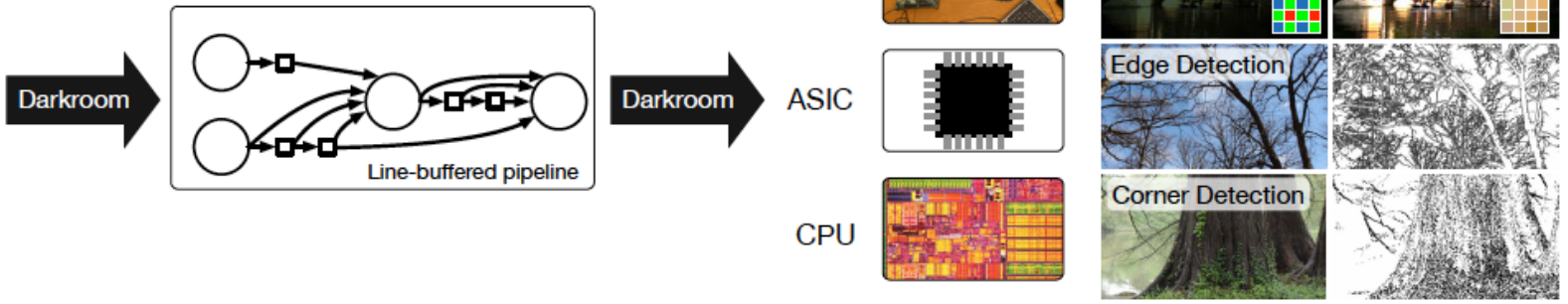
Darkroom (Stanford/MIT)



© Hegarty et al. , SIGGraph 2014

```
bx = im(x,y)
  (I(x-1,y) +
   I(x,y) +
   I(x+1,y))/3
end
by = im(x,y)
  (bx(x,y-1) +
   bx(x,y) +
   bx(x,y+1))/3
end
sharpened = im(x,y)
  I(x,y) + 0.1*
  (I(x,y) - by(x,y))
end
```

Stencil Language



Locality versus Parallelism

Halide Programming Language:

- http://halide-lang.org/

Performance needs a lot of tradeoffs

- Locality
- Parallelism
- Redundant recomputation

Outline

High-Level Optimization
- *Loop transformation*
- *Loop tiling/blocking*
- *Loop (nest) splitting*

Heterogeneous System Architecture (HSA)
- *Integrated CPU/GPU platforms*
- *Recent movement in chip designs*

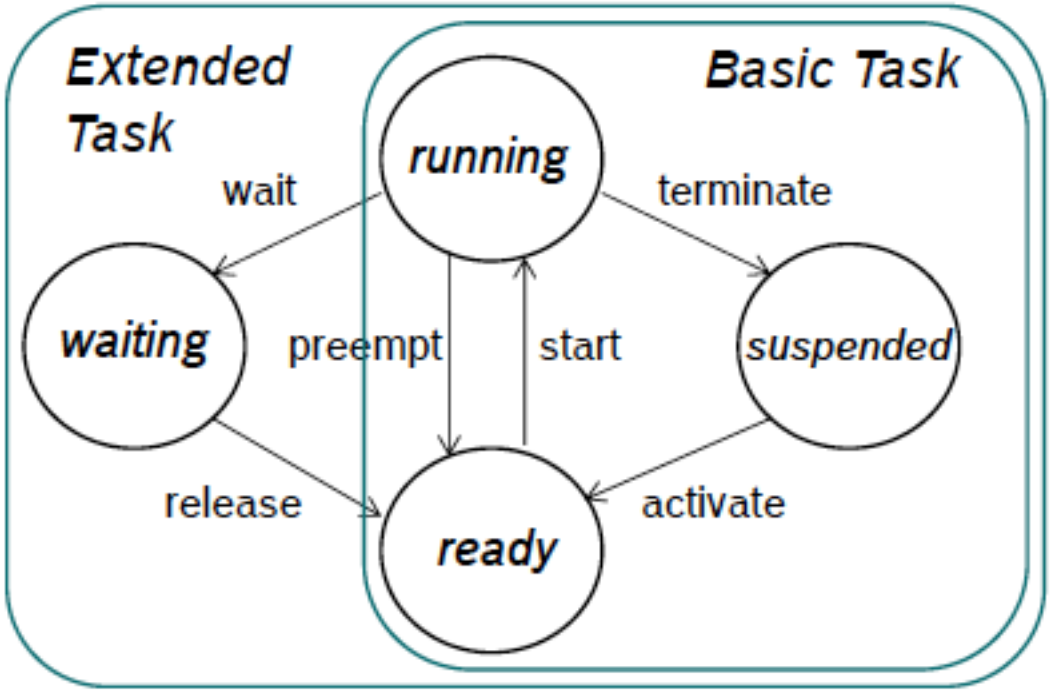
Architecture-aware software designs
- *Energy-efficiency issues*
- *Darkroom*
- *Halide*

Multicore revolutions
- *Impact on the "safety-critical" industry sector*

Automotive Software

Task

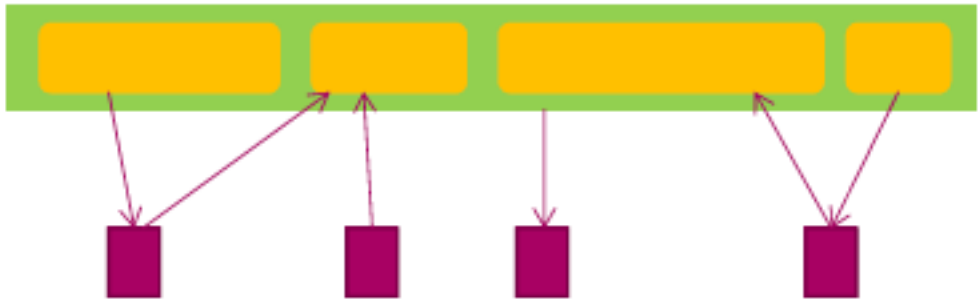
- Activation Pattern:
  - Periodic: 1 to 1000 ms
  - Angle synchronous
  - Sporadic
- Scheduled by the OS
  - Fixed Priorities
  - Preemptively or cooperatively



OSEK task states

Assessment of Multi-Core Worst-Case Execution Behavior

Labels



Communication between runnables is realized with reading and writing of labels

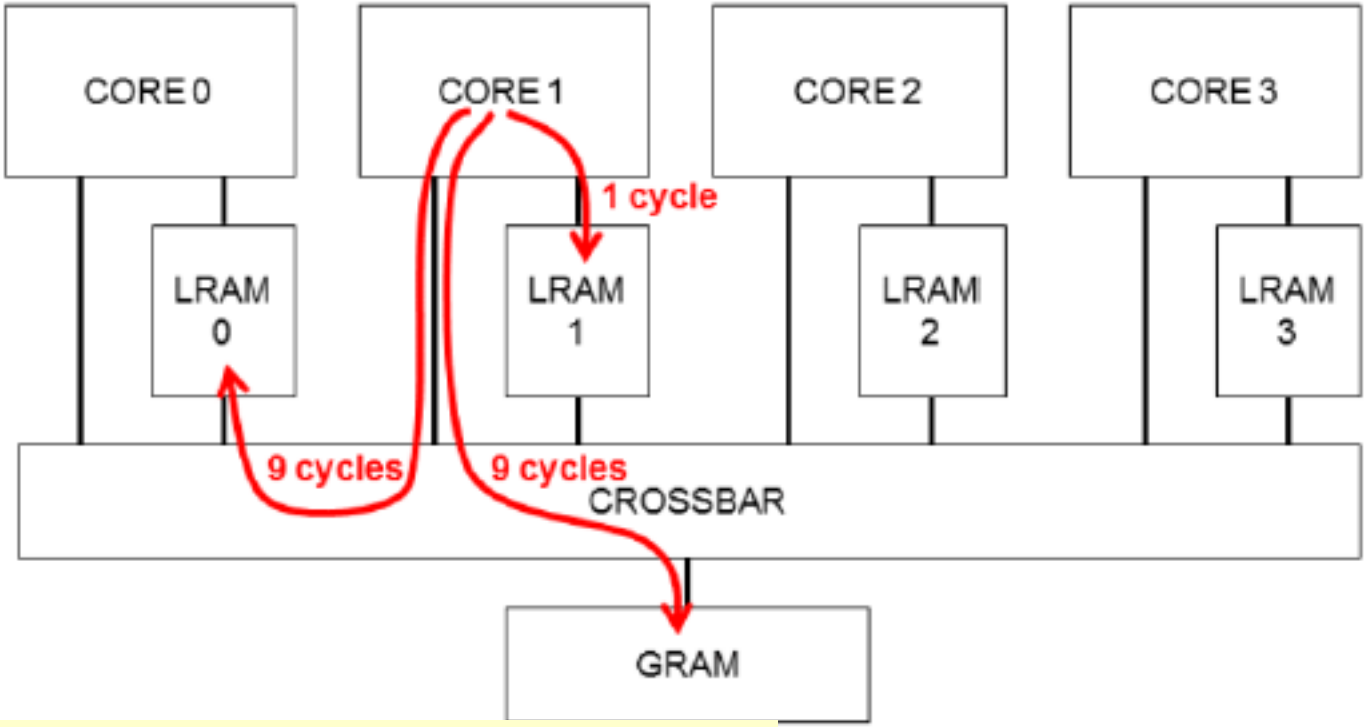
Communication	Share
Forward	25 %
Backward	35 %
InterTask	40 %

Size	Share
Atomic (1-4 bytes)	97 %
Structs / Arrays	3 %

Access type	Share
Read-only	40 %
Write-only	10 %
Read-Write	50 %

Multi-Core Memory Access Models

- Access time to data in different memories (local & global)



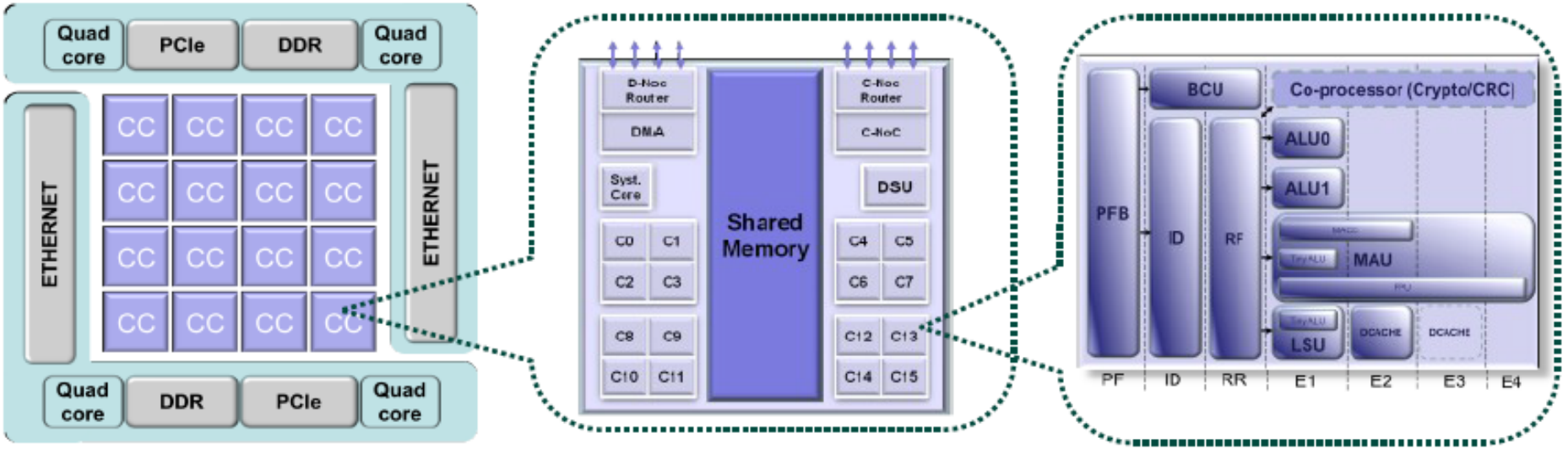
An Industrial Challenge (FMTV 2017)

Precise analysis of worst-case end-to-end latencies

- mainly due to different involved periods and time domains
- What is the effect on memory layout and interconnect on the execution times?
- Automatic optimized application and data mapping
- Evaluation of digital (multi-core) execution platforms
- Evaluation of software growth scenarios

© Hamann, Kramer, Ziegenbein, Lukasiewycz (Bosch), 2016

MPPA-256 Processor Architecture (Kalray)



Manycore Processor

- 16 compute clusters
- 2 I/O clusters each with quad-core CPUs, DDR3, 4 Ethernet 10G and 8 PCIe Gen3
- Data and control networks-on-chip
- Distributed memory architecture
- 634/317 GFLOPS SP/DP @ 600Mhz

Compute Cluster

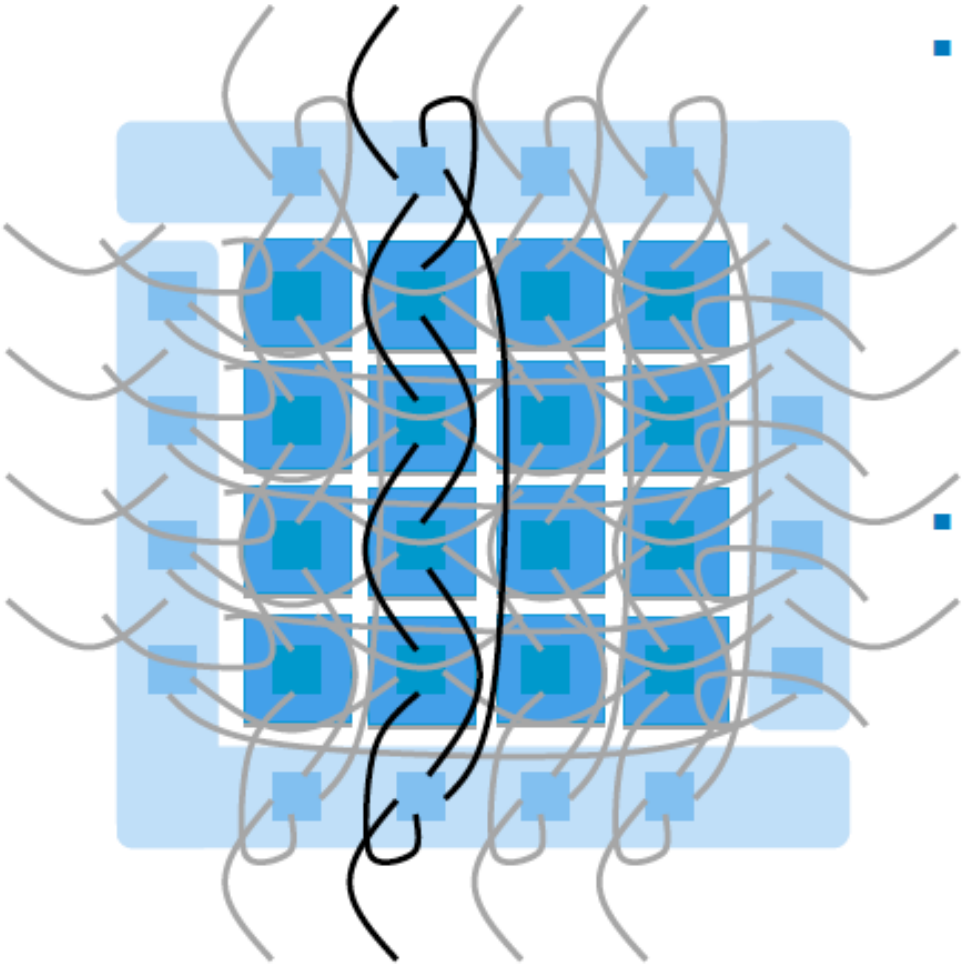
- 16 user cores + 1 system core
- NoC Tx and Rx interfaces
- Debug & Support Unit (DSU)
- 2 MB multi-banked shared memory
- 77GB/s Shared Memory BW
- 16 cores SMP System

VLIW Core

- 32-bit or 64-bit addresses
- 5-issue VLIW architecture
- MMU + I&D cache (8KB+8KB)
- 32-bit/64-bit IEEE 754-2008 FMA FPU
- Tightly coupled crypto co-processor
- 2.4 GFLOPS SP per core @600Mhz

Kalray, 2016

MPPA-256 NoC



- Dual 2D-torus NoC
  - D-NoC: high-bandwidth RDMA
  - C-NoC: low-latency mailboxes
  - 4B/cycle per link direction per NoC
  - Nx10Gb/s NoC extensions for connection to FPGA or other MPPA®
- Predictability
  - Data NoC is configured by selecting routes and injection parameters
  - Injection parameters are the (σ,ρ) or (burst, rate) of Cruz network calculus
  - Guaranteed services rely on same methods as in AFDX Ethernet

Safety-Critical Systems with Multicore Platforms

Goal: deploy multi-core processors for safety-critical real-time applications (avionics, automotive,…)

Problem: concurrent use of shared resources (e.g. interconnect, main memory)

- unknown access latency for a concrete resource access
- complicated timing analysis
- hardware platforms may not be predictable
- Many features are designed by computer architects for *average cases* only

Solution?

- Maybe it is up to you.
- Did you see the above challenges?