General Purpose GPUs
GP-GPUs

Paolo Burgio
paolo.burgio@unimore.it

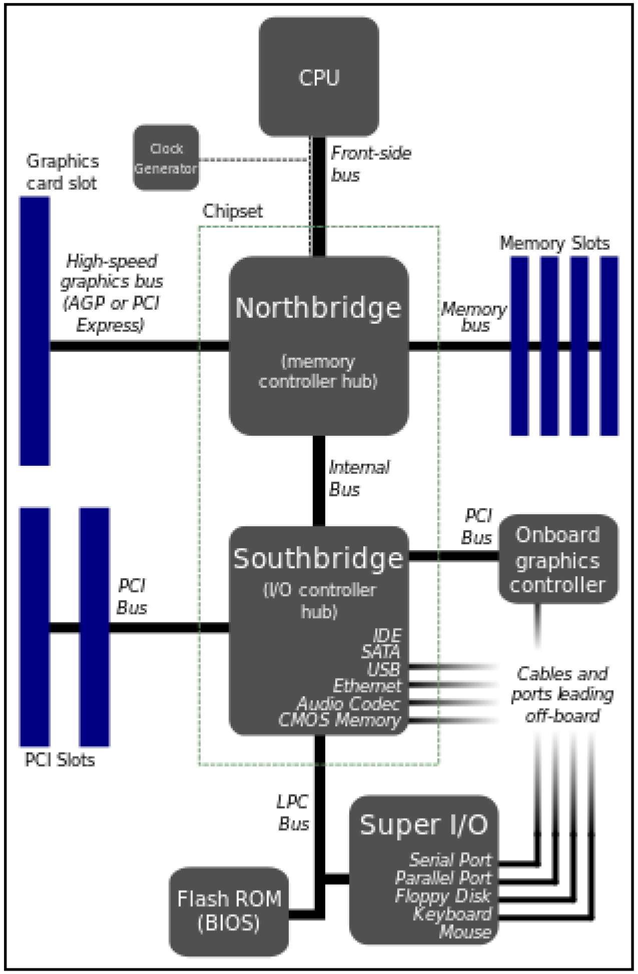

Multithreaded programming
Theory
Actual

Graphics Processing Units

- (Co-)processor devoted to graphics
  - Built as "monolithical" chip
  - Integrated as co-processor
  - Recently, SoCs

- Main providers
  - NVIDIA
  - ATI
  - AMD
  - Intel…

- We will focus on NVIDIA
  - Widely adopted
  - Adopted by us



A bit of history…

- 70s: first "known" graphic card on a board package
- Early 90s: 3D graphics popular in games
- 1992: OpenGL
- 1999: NVIDIA GeForce 256 "World's first GPU"
- 2001: NVIDIA GeForce 3, w/programmable shaders (First GP-GPU)
- 2008: NVIDIA GeForce 8800 GTX w/CUDA capabilities - Tesla arch.
- 2009: OpenCL 1.0 inside MAC OS X Snow Leopard
- 2010: NVIDIA GeForce 400 Series - Fermi arch.
- 2010-1: OpenCL 1.1, 1.2
- 2012: NVIDIA GeForce 600 Series - Kepler arch.
- 2013: OpenCL 2.0
- 2014: NVIDIA GeForce 745 OEM - Maxwell arch.
- 2015 Q4: NVIDIA and HiPeRT Lab start cooperation ;)
- 2017 Q1: NVIDIA Drive Px2 for Self-Driving Cars
- 2019 Q1: NVIDIA Pegasus for Self-Driving Cars

…a bit of confusion!

- Many architectures
  - Tesla, Fermi, Maxwell, Pascal, Volta..
- Many programming ~~librar~~… ~~languag~~… frameworks
  - OpenGL
  - CUDA
  - OpenCL
  - …
- Many application domains!
  - Graphics
  - GP-GPUs?
  - Automotive!??!?!??!
- Let's start from scratch…

GPU for graphics - OpenGL

- Use GPUs for rendering of graphics
  - A library of functions and datatypes
  - Use directly in the code
  - High-level operations on lights, shapes, shaders…

- Tailored for the specific domain and programmer skills
  - Hides away the complexity of the machine
  - Takes care of "low" level optimizations/operations

GPU for graphics - OpenGL

- Use GPUs for rendering of graphics
  - A library of functions and datatypes
  - Use directly in the code
  - High-level operations on lights, shapes, shaders…
- Tailored for the specific domain and programmer skills
  - Hides away the complexity of the machine
  - Takes care of "low" level optimizations/operations

```
int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("blender");
  glutDisplayFunc(display);
  glutVisibilityFunc(visible);

  glNewList(1, GL_COMPILE); /* create ico display list */
  glutSolidIcosahedron();
  glEndList();

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
  glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
  glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
  glLightfv(GL_LIGHT2, GL_DIFFUSE, light2_diffuse);
  glLightfv(GL_LIGHT2, GL_POSITION, light2_position);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  glEnable(GL_LINE_SMOOTH);

  glLineWidth(2.0);
  glMatrixMode(GL_PROJECTION);
  gluPerspective( /* field of view in degree */ 40.0,
                  /* aspect ratio */ 1.0,
                  /* Z near */ 1.0,
                  /* Z far */ 10.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0.0, 0.0, 5.0, /* eye is at (0,0,5) */
            0.0, 0.0, 0.0, /* center is at (0,0,0) */
            0.0, 1.0, 0.); /* up is in positive Y direction */
  glTranslatef(0.0, 0.6, -1.0);

  glutMainLoop();
  return 0; /* ANSI C requires main to return int. */
}
```

OpenGL

…

rogrammer skills

ons

GPU for graphics - OpenGL

- Use GPUs for rendering of graphics
  - A library of functions and datatypes
  - Use directly in the code
  - High-level operations on lights, shapes, shaders…
- Tailored for the specific domain and programmer skills
  - Hides away the complexity of the machine
  - Takes care of "low" level optimizations/operations

```
GLfloat light0_ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat light0_diffuse[] = {0.0, 0.0, 0.0, 1.0};
GLfloat light1_diffuse[] = {1.0, 0.0, 0.0, 1.0};
GLfloat light1_position[] = {1.0, 1.0, 1.0, 0.0};
GLfloat light2_diffuse[] = {0.0, 1.0, 0.0, 1.0};
GLfloat light2_position[] = {-1.0, -1.0, 1.0, 0.0};
```

GPU for graphics - OpenGL

› Use GPUs for rendering of graphics
  - A library of functions and datatypes
  - Use directly in the code
  - High-level operations on lights, shapes, shaders…

› Tailored for the specific domain and programmer skills
  - Hides away the complexity of the machine
  - Takes care of "low" level optimizations/operations

```
GLfloat light0_ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat light0_diffuse[] = {0.0, 0.0, 0.0, 1.0};
GLfloat light1_diffuse[] = {1.0, 0.0, 0.0, 1.0};
GLfloat light1_position[] = {1.0, 1.0, 1.0, 0.0};
GLfloat light2_diffuse[] = {0.0, 1.0, 0.0, 1.0};
GLfloat light2_position[] = {-1.0, -1.0, 1.0, 0.0};
```

General Purpose - GPUs

- We have a machine with thousand of cores
  - why should we use it only for graphics?

- Use it for General Purpose Computing!
  - **GP-GPU**
  - ~yr 2000

NdA: Computing modes
- **G**eneral **P**urpose Computing
- **H**igh-**P**erformance **C**omputing
- **E**mbedded **C**omputing
- **R**eal-**T**ime Computing
- …

General Purpose - GPUs

- We have a machine with thousand of cores
  - why should we use it only for graphics?

- Use it for General Purpose Computing!
  - **GP-GPU**
  - ~yr 2000

NdA: Computing modes
- **G**eneral **P**urpose Computing
- **H**igh-**P**erformance **C**omputing
- **E**mbedded **C**omputing
- **R**eal-**T**ime Computing
- …

General Purpose - GPUs

- We have es
  - why sho
- Use it for
  - **GP-GPU**
  - ~yr 2000

NdA: Computing modes
- **G**eneral **P**urpose Computing
- **H**igh-**P**erformance **C**omputing
- **E**mbedded **C**omputing
- **R**eal-**T**ime Computing
- …

General Purpose - GPUs

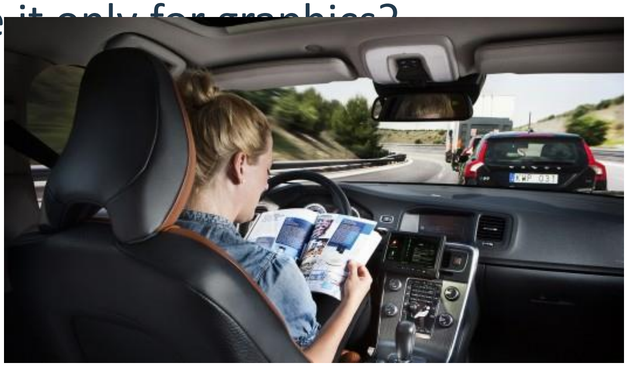
- We have a machine with thousand of cores
  - why should we use it only for graphics?

- Use it for General Purpose Computing!
  - **GP-GPU**
  - ~yr 2000

NdA: Computing modes
- **G**eneral **P**urpose Computing
- **H**igh-**P**erformance **C**omputing
- **E**mbedded **C**omputing
- **R**eal-**T**ime Computing
- …

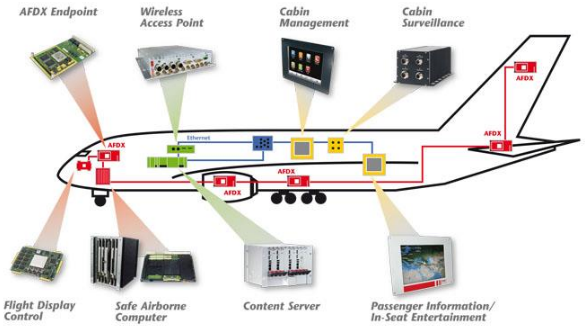
General Purpose - GPUs

- We have a machine with thousand of cores
  - why should we use it only for graphics?

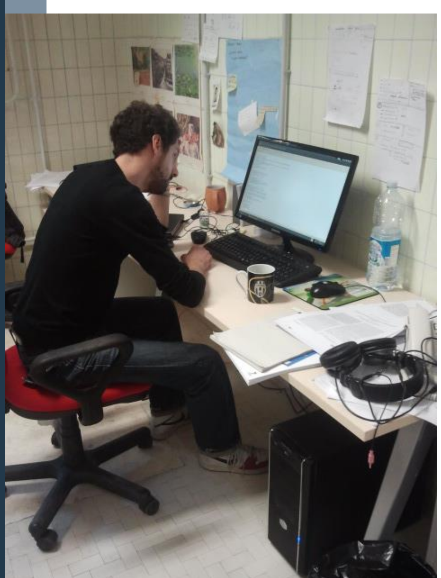


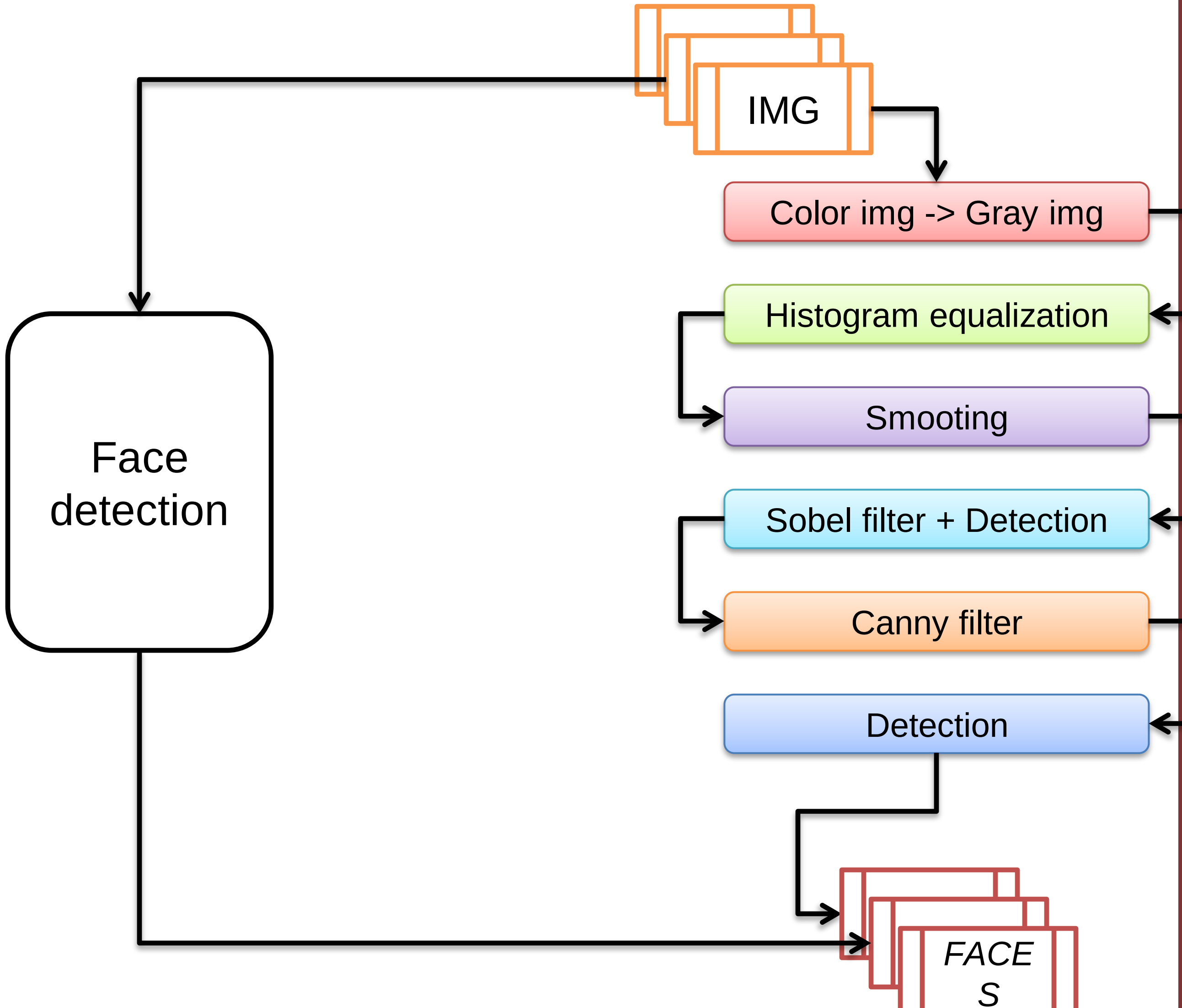
- Use it for Genera
  - **GP-GPU**
  - ~yr 2000

NdA: Computing modes
- General Purpose Computing
- High-Performance Computing
- Embedded Computing
- Real-Time Computing
- …

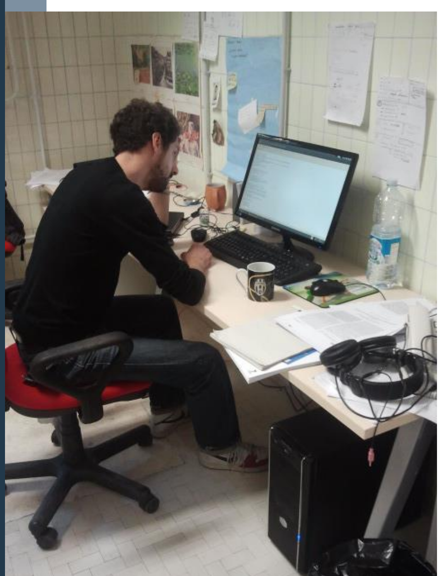


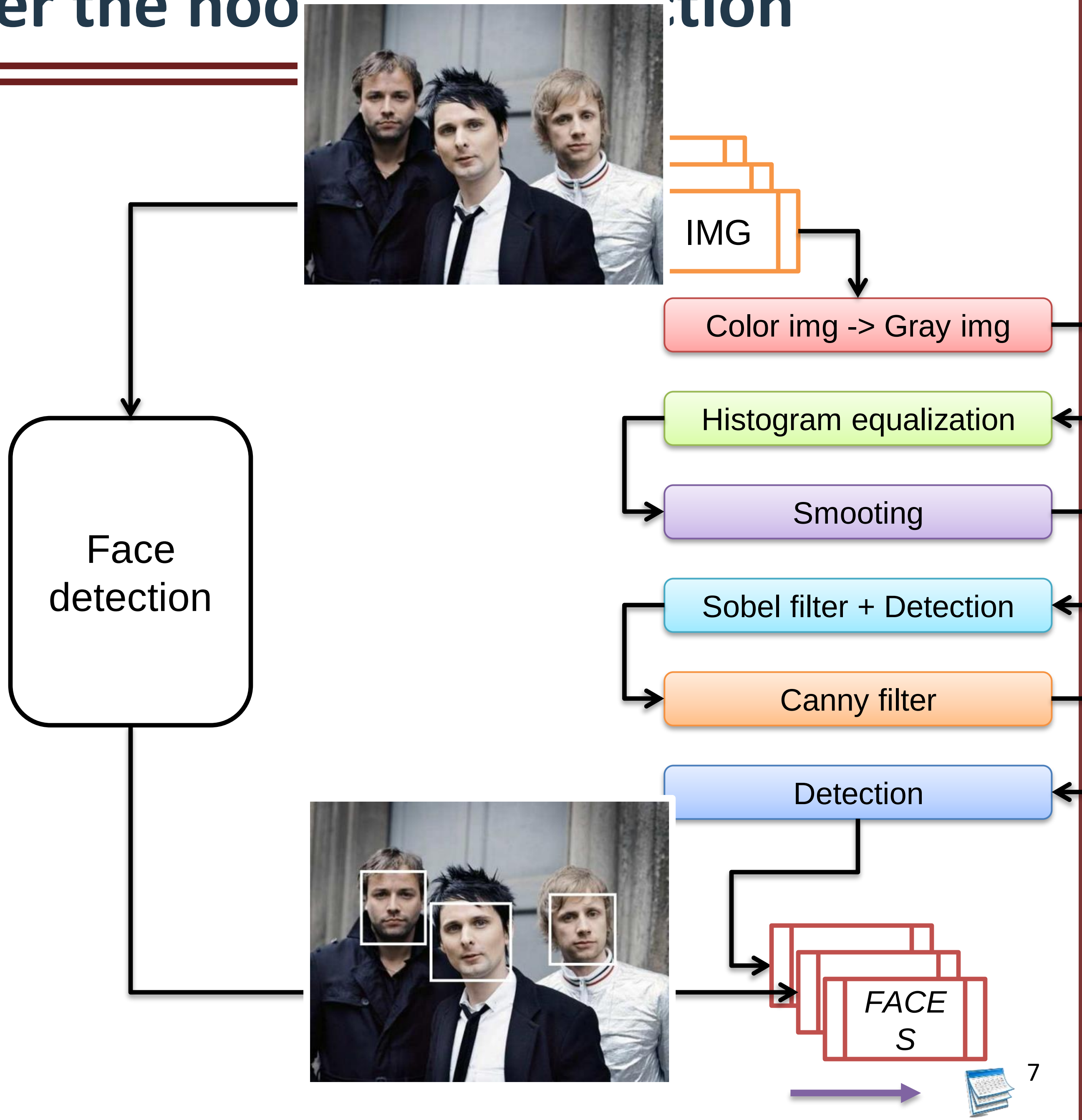
Under the hood: face detection



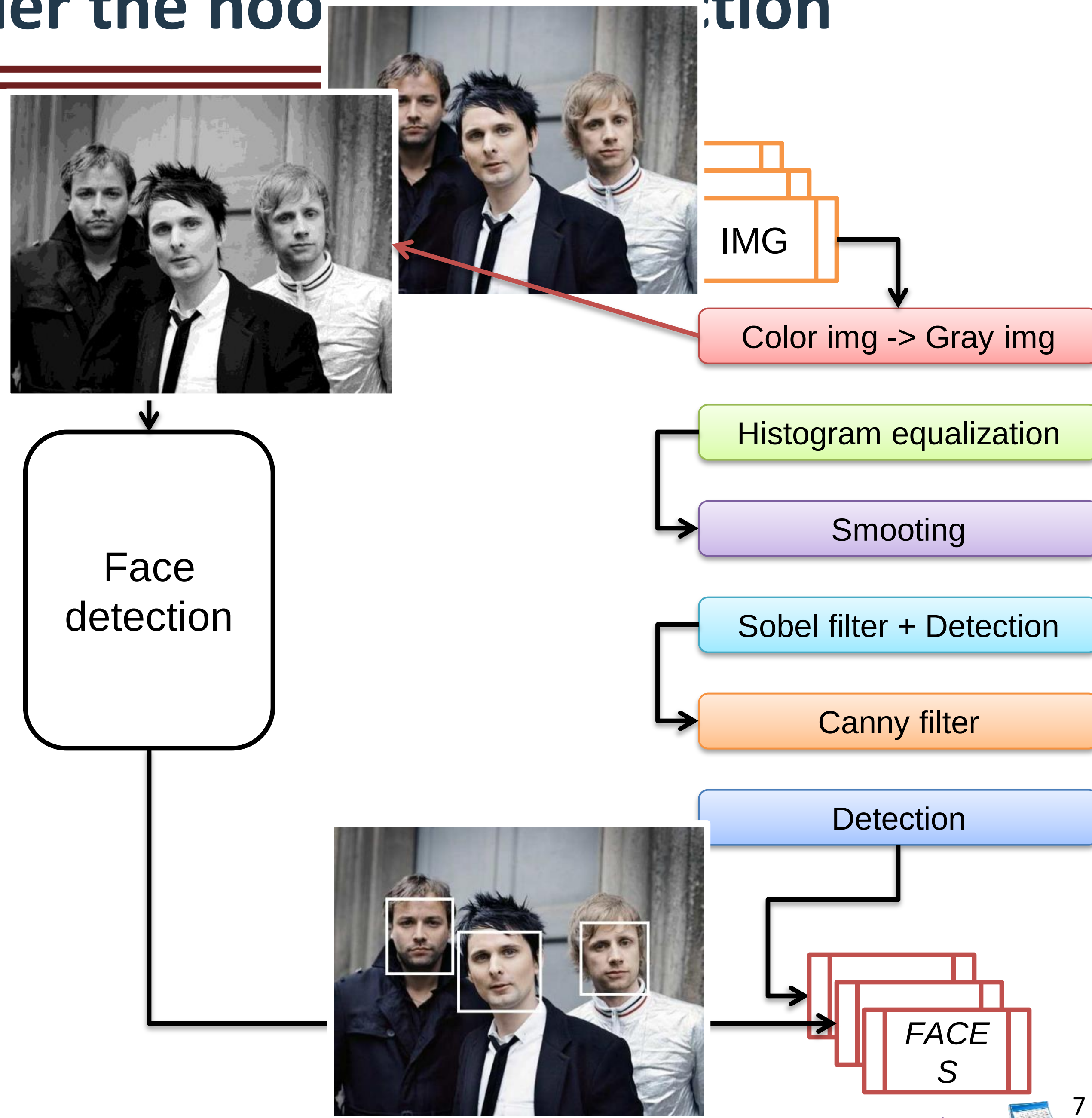


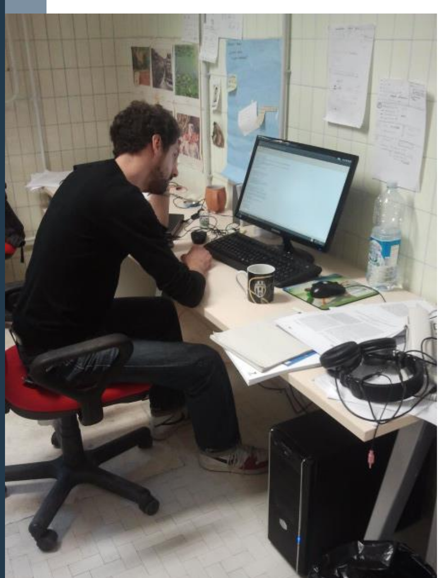
Under the hood: face detection



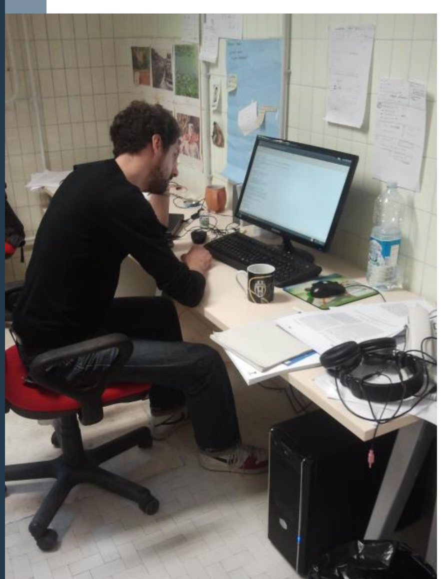


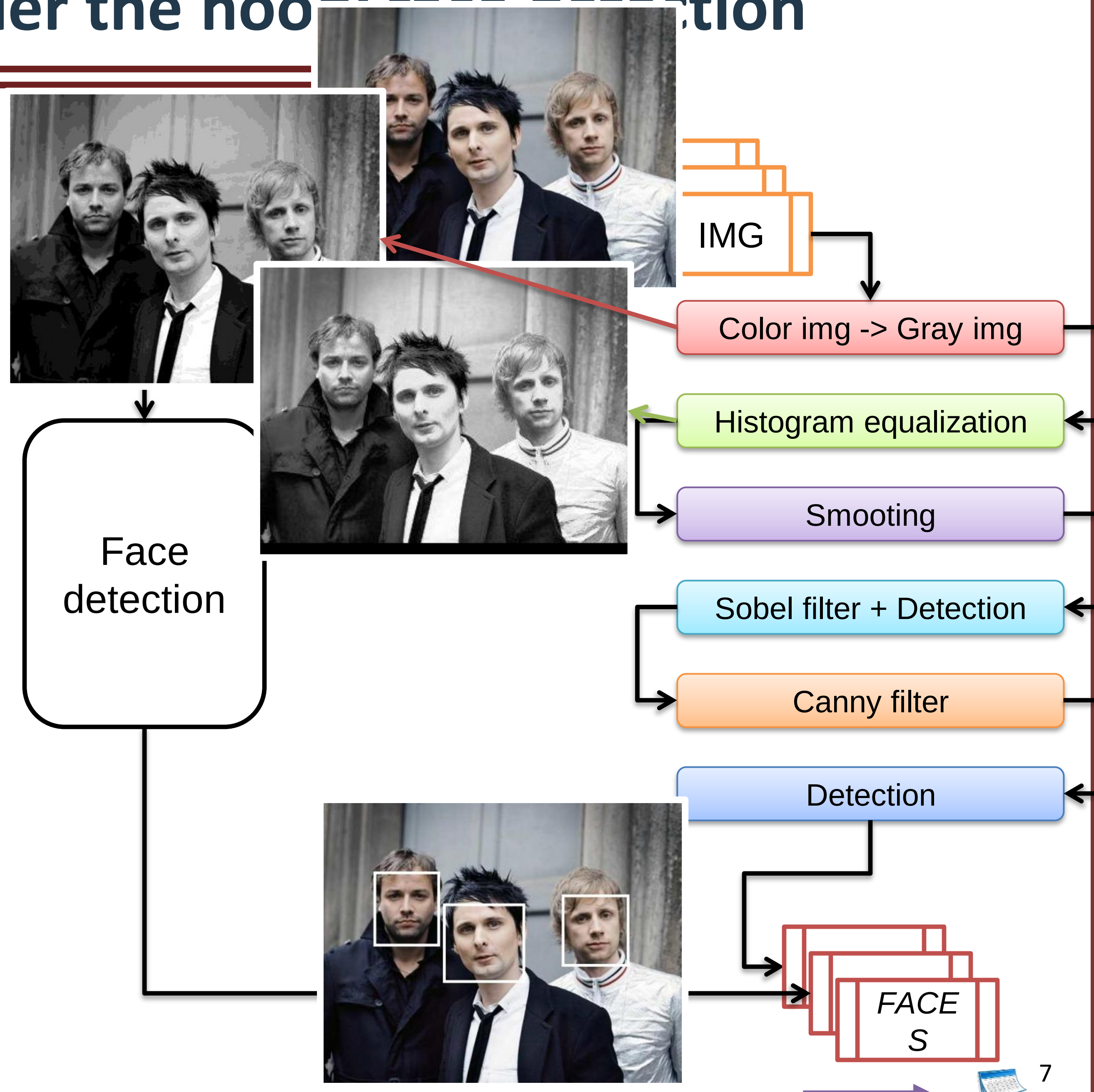
Under the hood: face detection



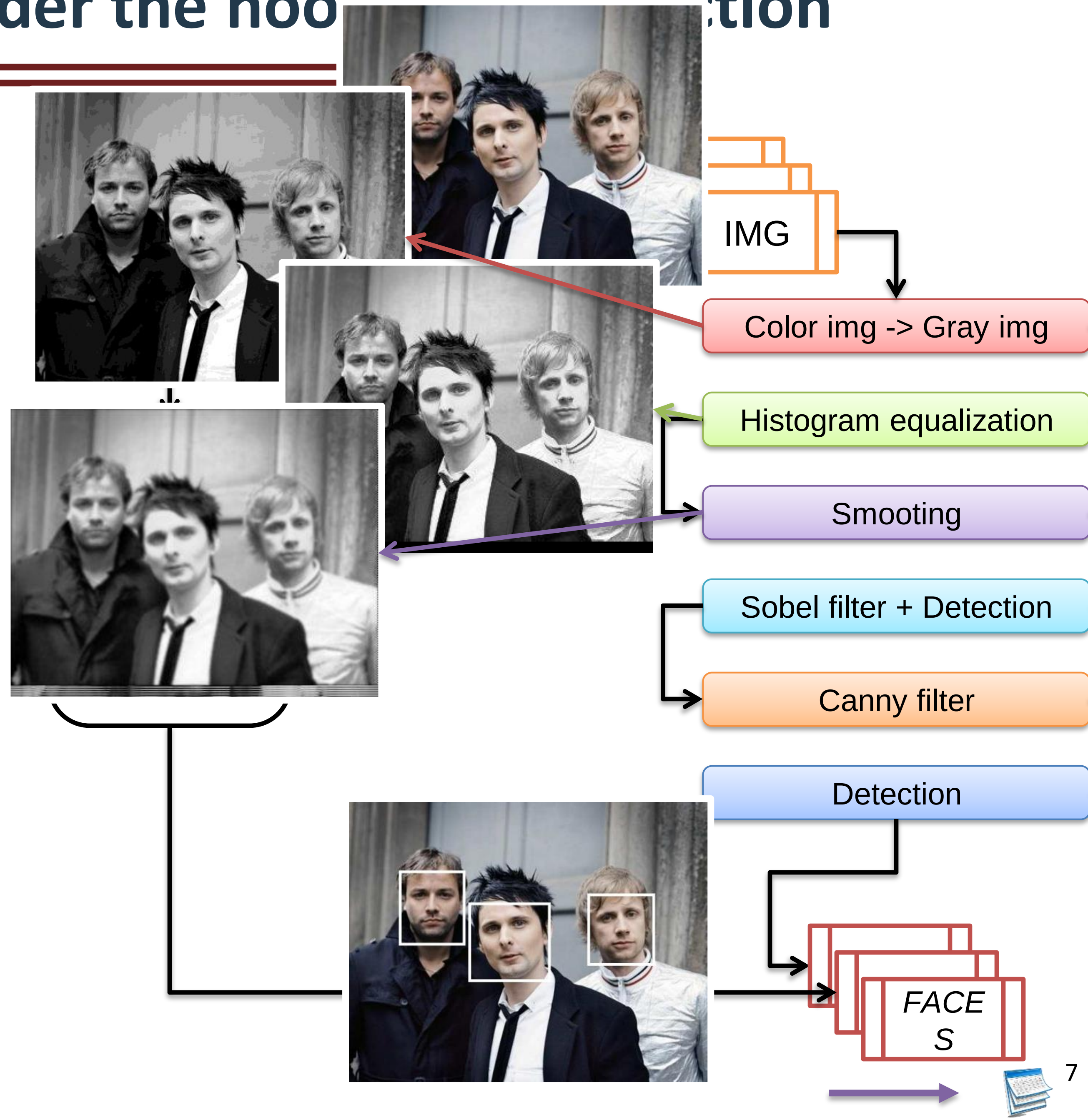


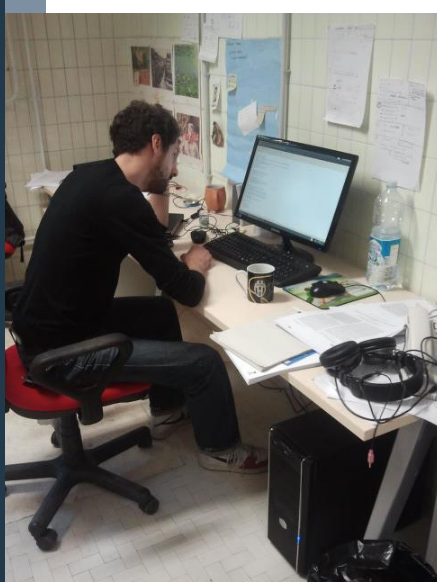
Under the hood: face detection



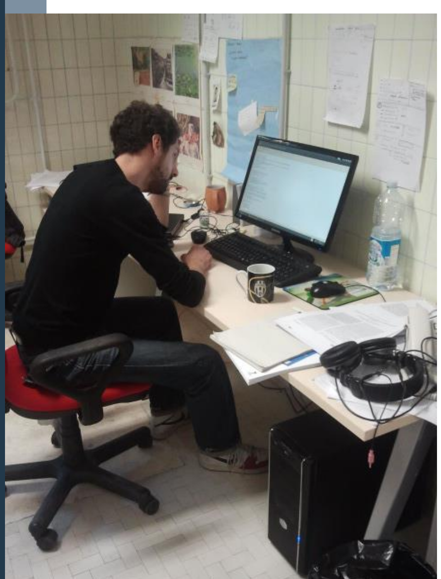


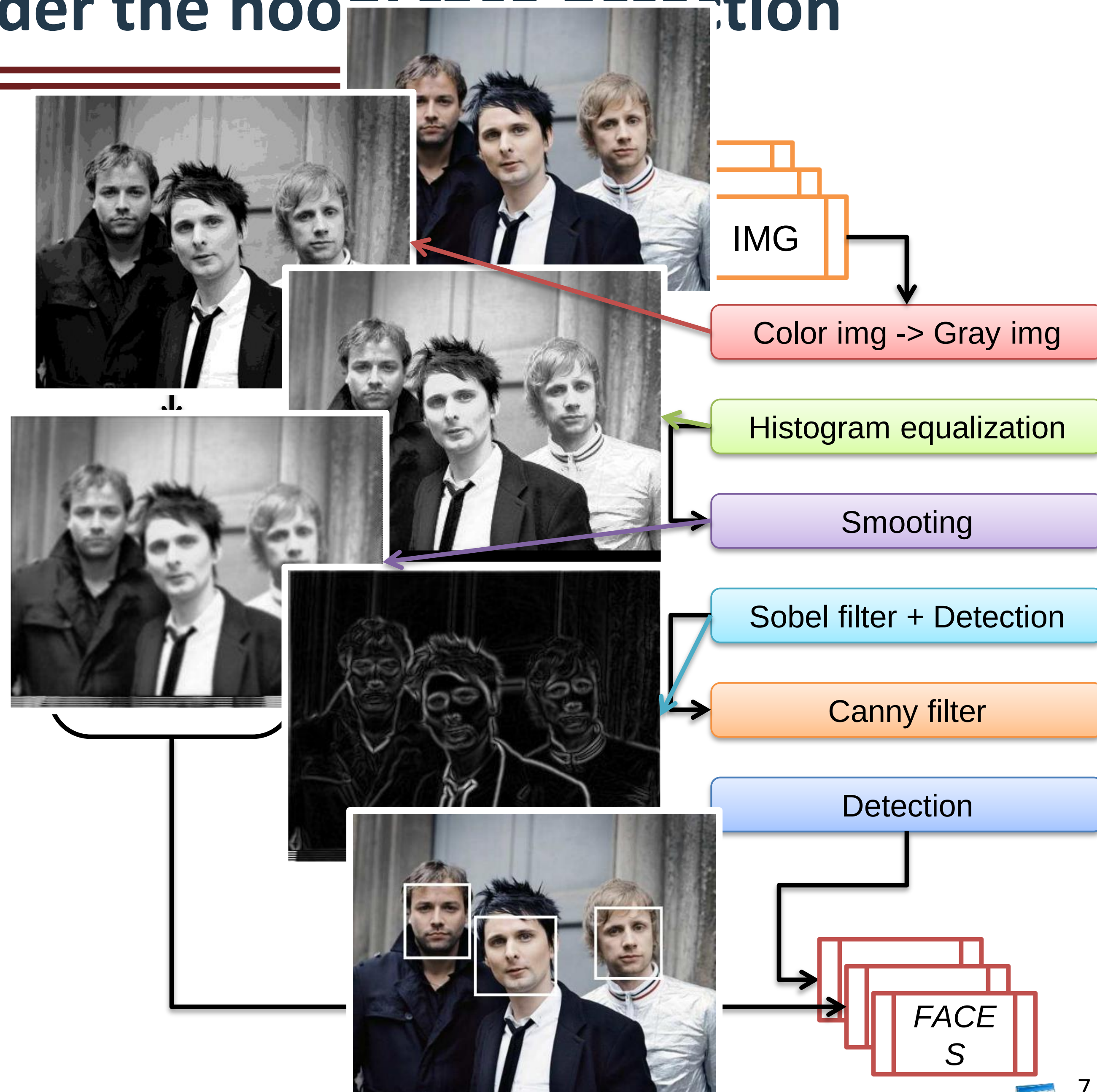
Under the hood: face detection



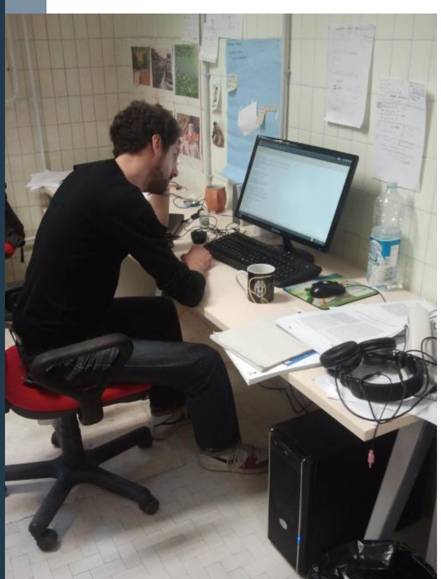


Under the hood: face detection





Under the hood: face detection



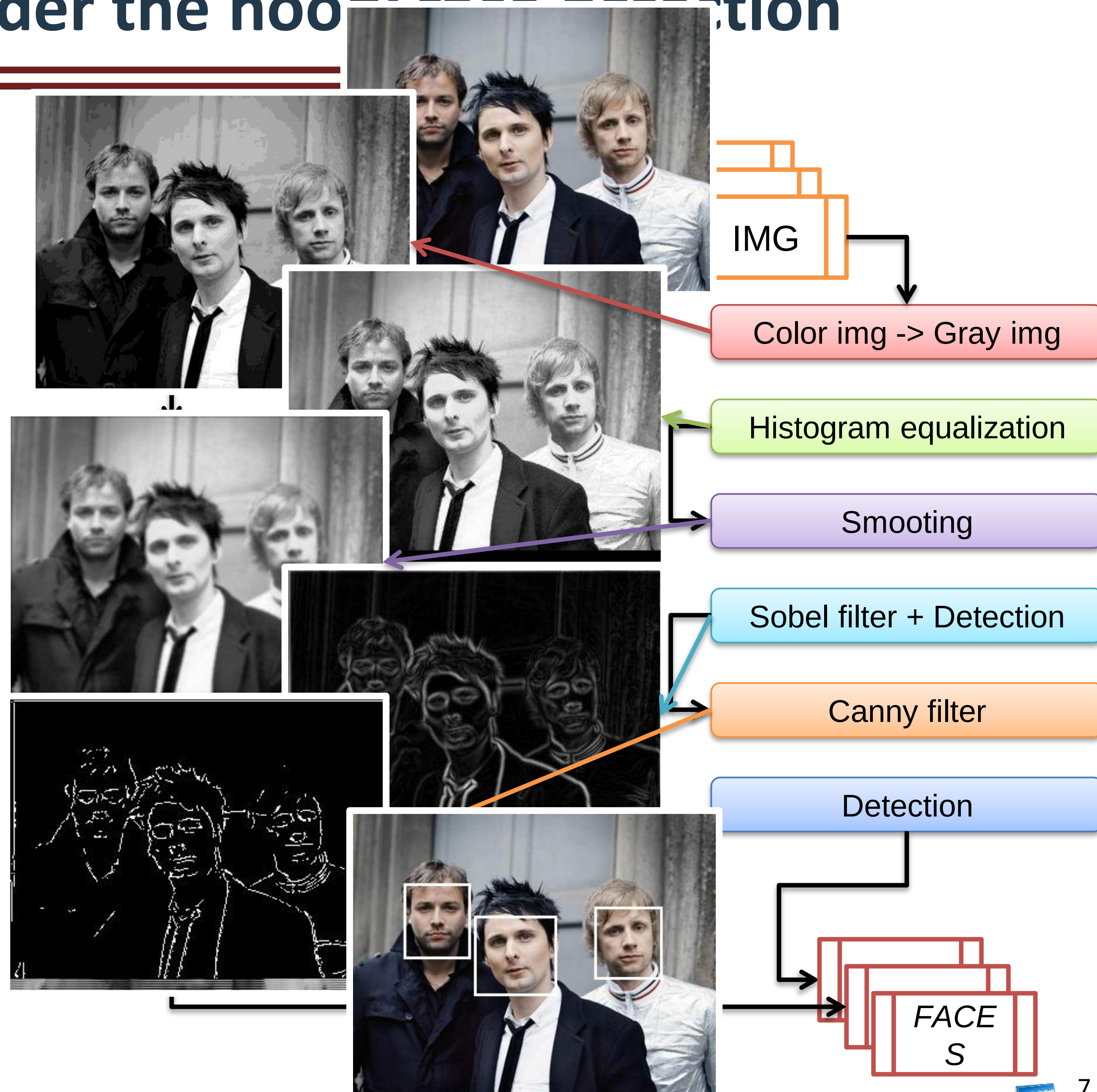
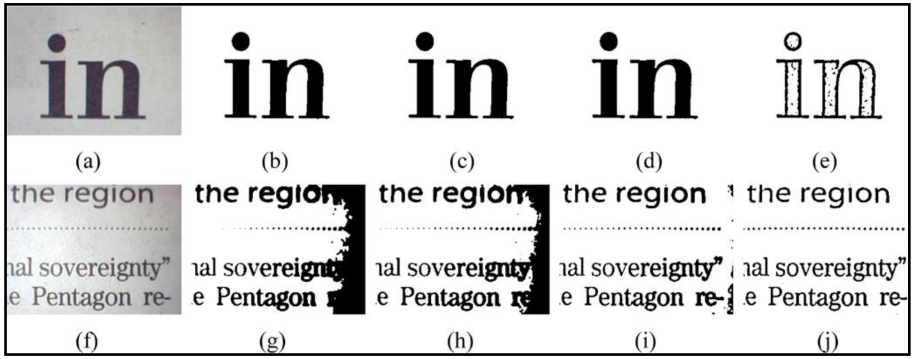


Image binarization

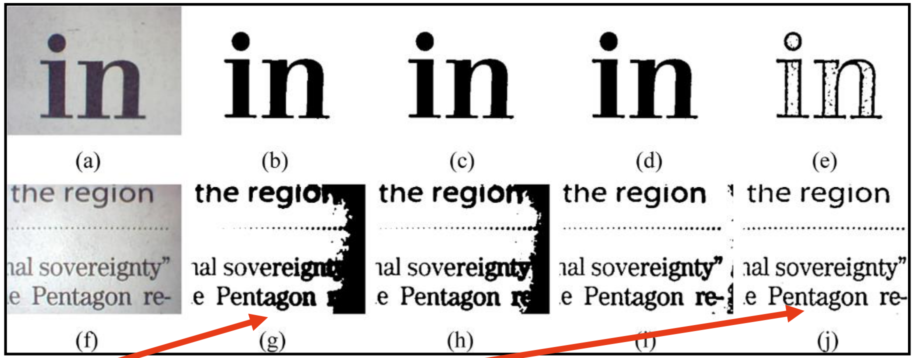
- Graylevel image => B/W image
- Pixel: 256 shades of gray
  - `unsigned chars`
  - 255 => white
  - 0 => black




```c
#define GRAY_THRESHOLD 100
#define WHITE 255
#define BLACK 0
void binarizeImage(const unsigned char inputImg[],
                   unsigned char outputImg[],
                   unsigned int imgDim)
{
  for(int i=0; i<imgDim; i++)
    if(inputImg[i] >= GRAY_THRESHOLD)
      outputImg[i] = WHITE;
    else
      outputImg[i] = BLACK;
}
```

Image binarization

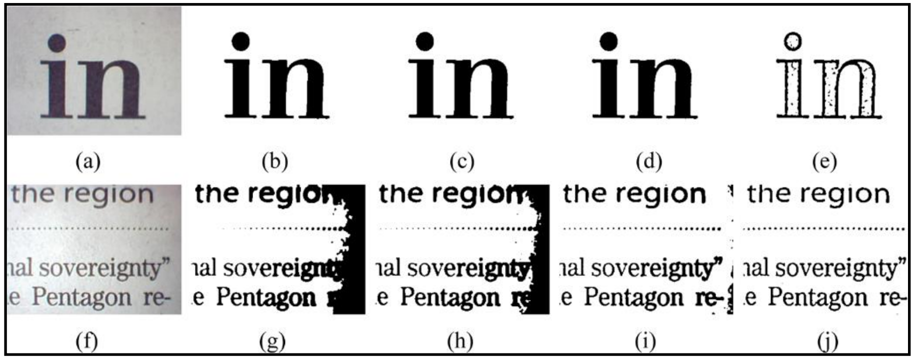
- Graylevel image => B/W image
- Pixel: 256 shades of gray
  - `unsigned chars`
  - 255 => white
  - 0 => black




```c
#define GRAY_THRESHOLD 100
#define WHITE 255
#define BLACK 0
void binarizeImage(const unsigned char inputImg[],
                   unsigned char outputImg[],
                   unsigned int imgDim)
{
  for(int i=0; i<imgDim; i++)
    if(inputImg[i] >= GRAY_THRESHOLD)
      outputImg[i] = WHITE;
    else
      outputImg[i] = BLACK;
}
```

Image binarization

- Graylevel image => B/W image
- Pixel: 256 shades of gray
  - `unsigned chars`
  - 255 => white
  - 0 => black

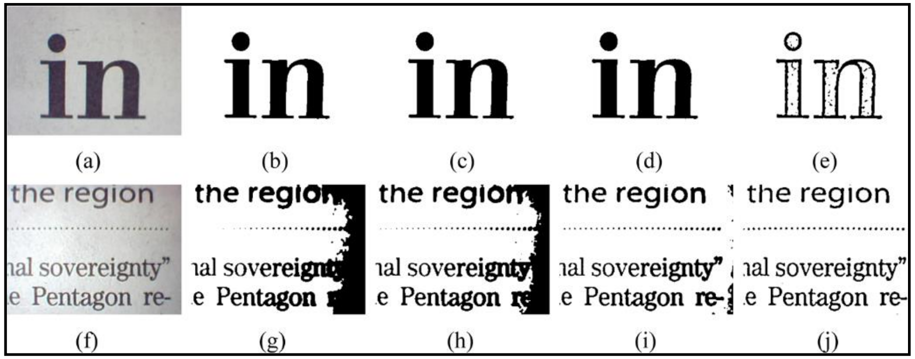



```
#define GRAY_THRESHOLD 100
#define WHITE 255
#define BLACK 0
void binarizeImage(const unsigned char inputImg[],
                   unsigned char outputImg[],
                   unsigned int imgDim)
{
  for(int i=0; i<imgDim; i++)
    if(inputImg[i] >= GRAY_THRESHOLD)
      outputImg[i] = WHITE;
    else
      outputImg[i] = BLACK;
}
```

Multiple Data

Image binarization

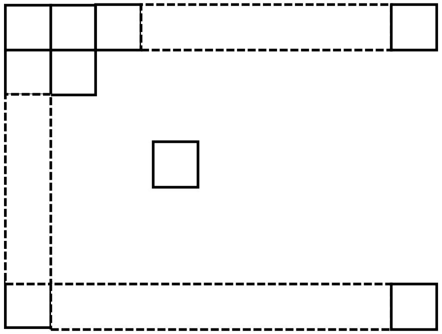
- Graylevel image => B/W image
- Pixel: 256 shades of gray
  - `unsigned chars`
  - 255 => white
  - 0 => black




```c
#define GRAY_THRESHOLD 100
#define WHITE 255
#define BLACK 0
void binarizeImage(const unsigned char inputImg[],
                   unsigned char outputImg[],
                   unsigned int imgDim)
{
  for(int i=0; i<imgDim; i++)
    if(inputImg[i] >= GRAY_THRESHOLD)
      outputImg[i] = WHITE;
    else
      outputImg[i] = BLACK;
}
```

Single Program

GPUs



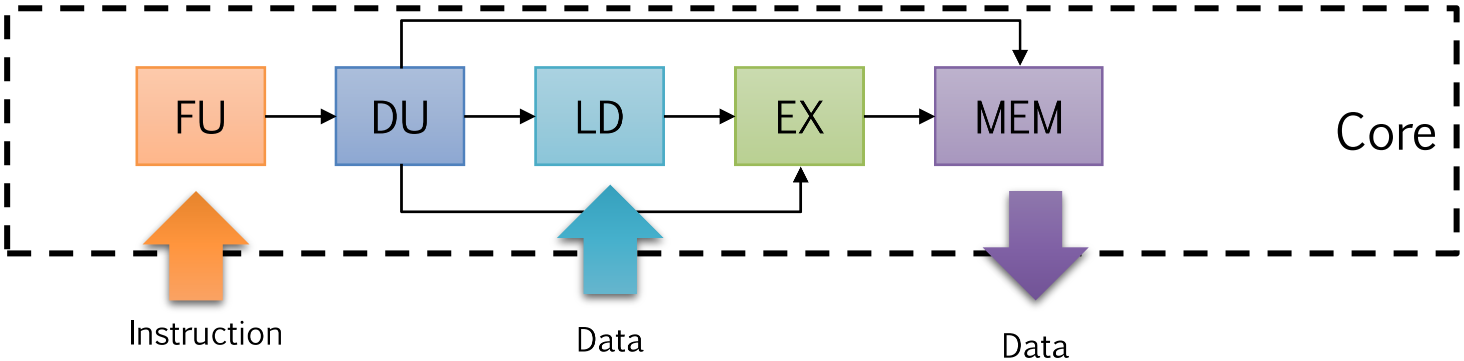
- Let's (re)design them!
- We want to perform graphics
  - E.g., filters, shaders…
- Ultimately, operations on pixels!
  - Same algorithm repeated for each (subset of) pixels
- Algorithm => program
- (subset of) pixels => data
- Same (single) Program, Multiple Data – SPMD
  - Not SIMD!

A (programmable) machine

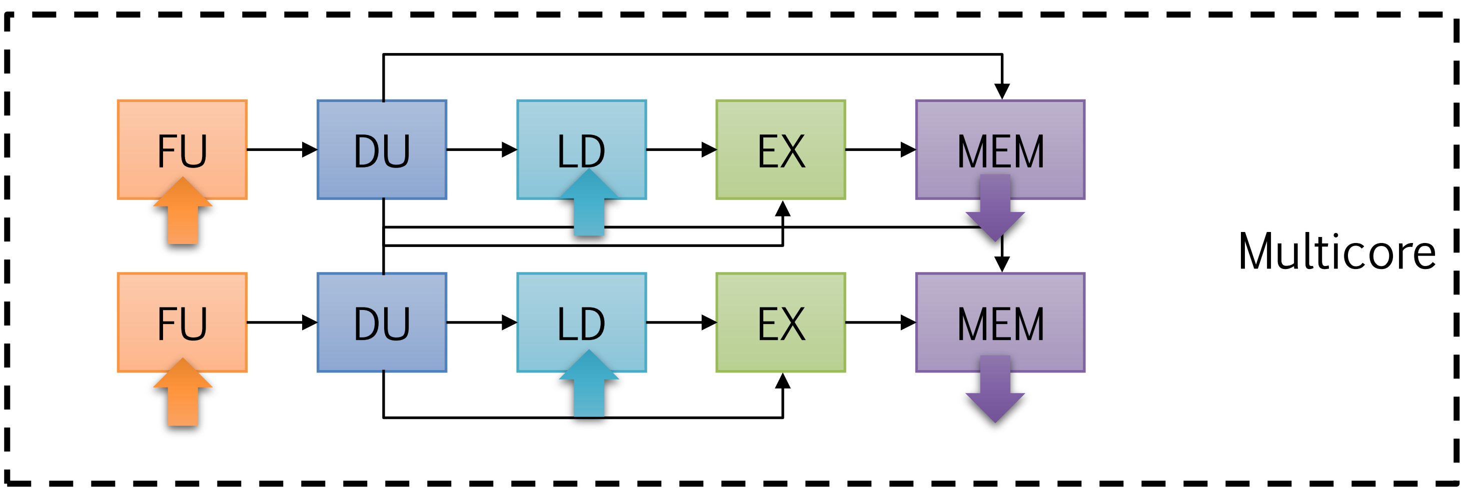
- Algorithms for image processing are
  - Highly regular (loop-based, with well known boundaries at image rows/columns)
  - Massively parallel (thousands of threads)
- Regular, "big" loops
  - Single Program (Loop Iteration) Multiple Data - SPMD
  - Parallel threads perform the very same operation on adjacent data
- We need a massively parallel machine
  - Thousands of cores
- With simple cores
  - FP Support
- To perform the very same instruction!
  - Same Fetch Unit and Decode Unit

Fetch and decode units

› Traditional pipeline

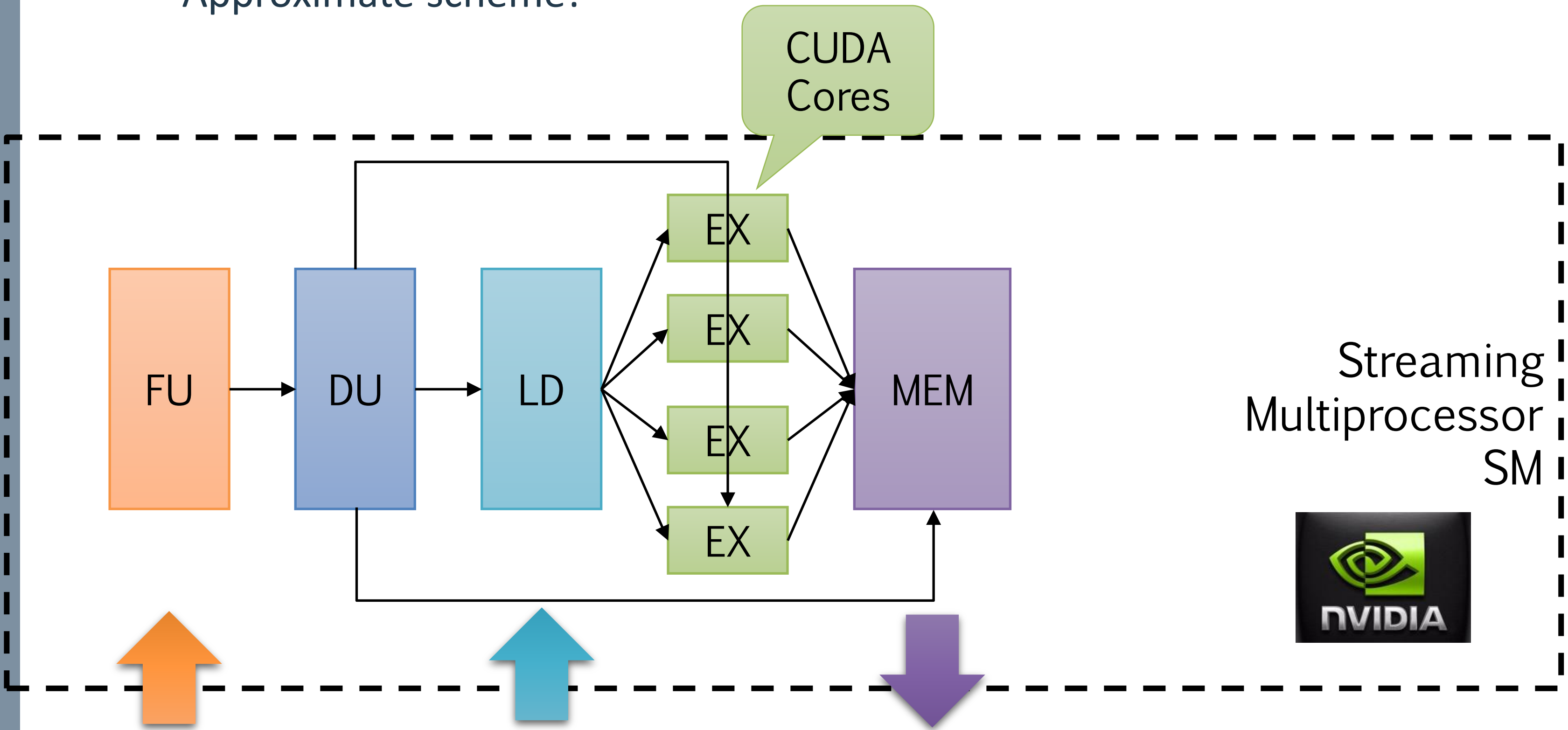


› Traditional parallel pipeline



GPU multi-core

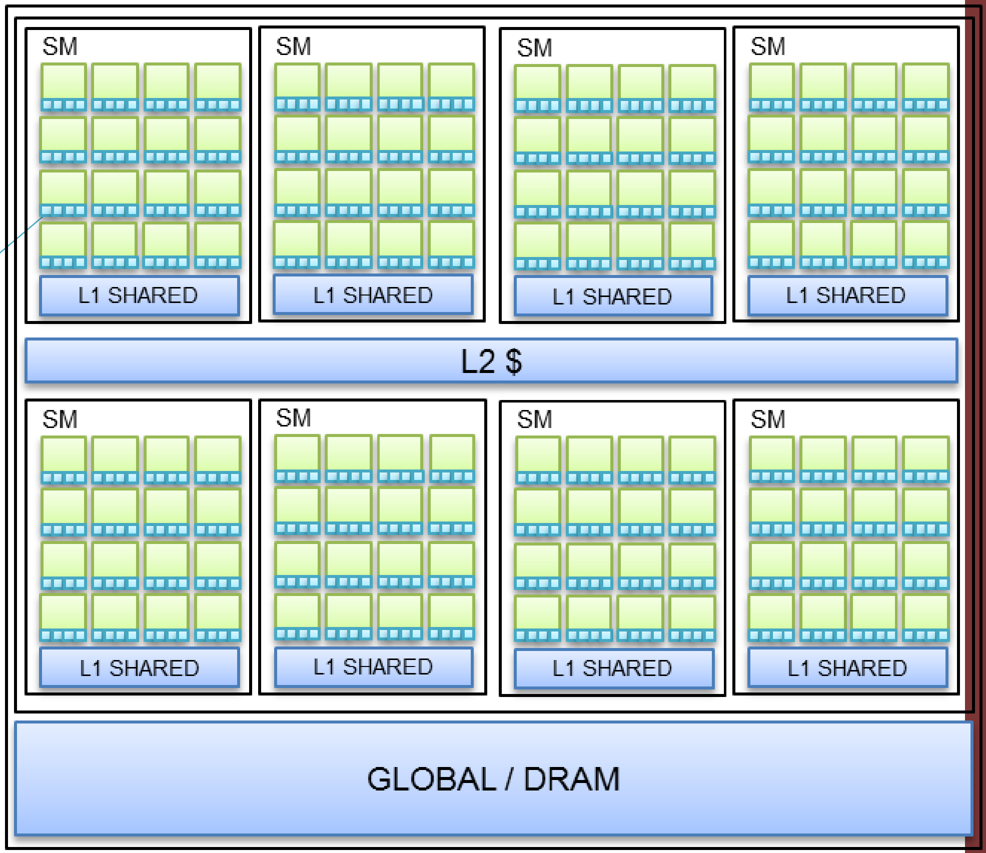
- Share FU, DU, MEM units
  - Approximate scheme!



SMs as building block

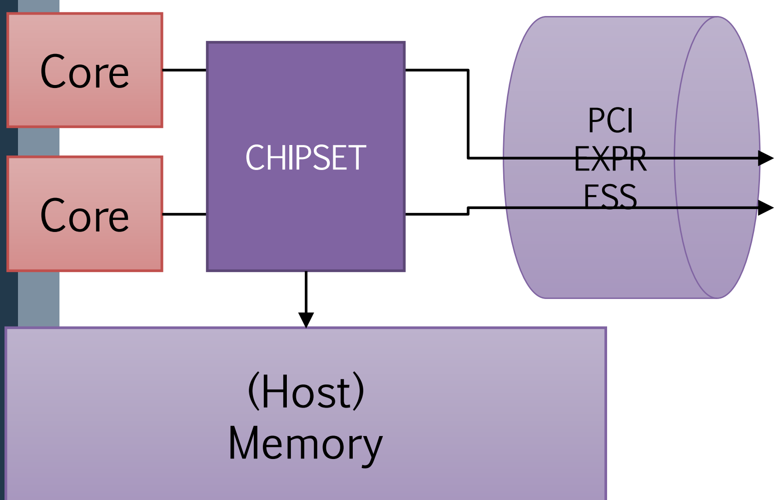
- Architecture of the SM
  - GPU "class"
  - Kepler has 192 cores
  - Maxwell/Pascal has 128 cores
- Number of SMs
  - GPU model
  - Maxwell's GTX980 has 10
  - Pascal's GTX1080 has 20
  - Pascal's Drive PX1 has 2
- NUMA memory system

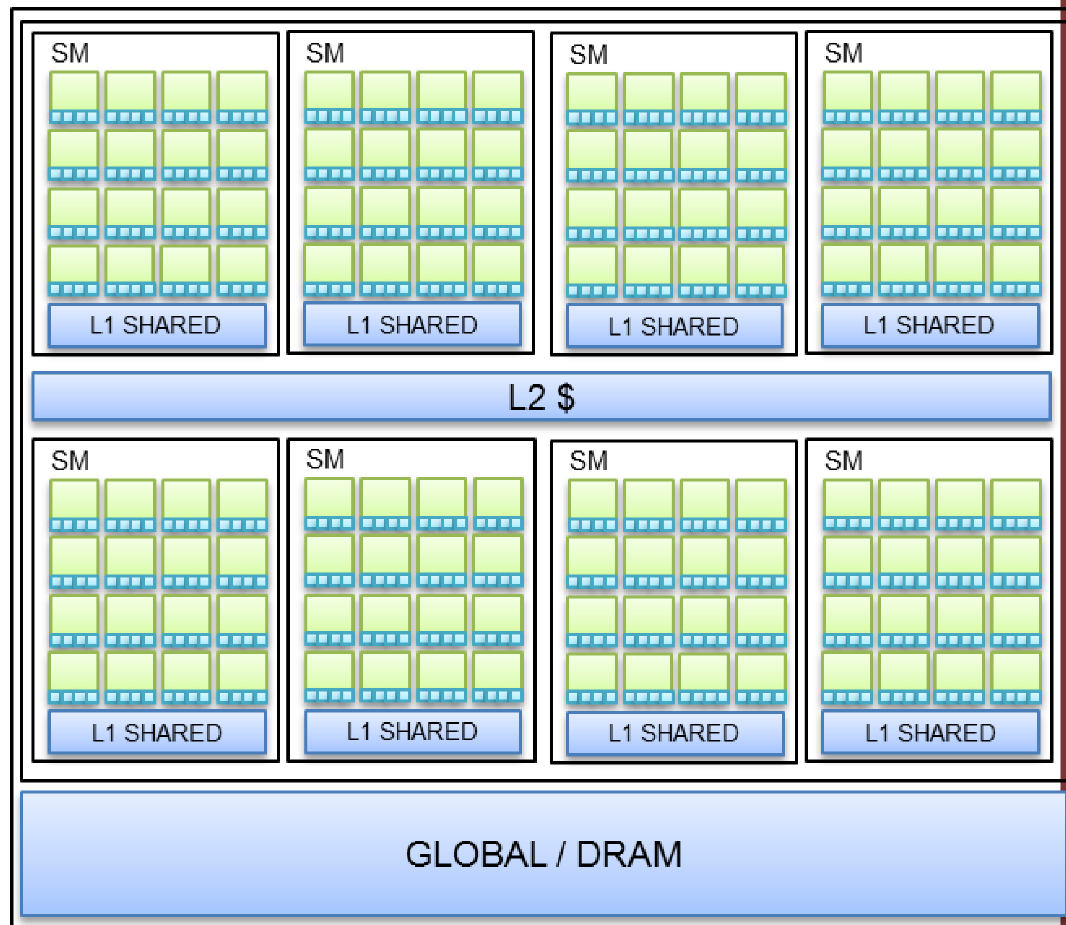
Local Memory



GPU as a device (Discrete GPGPUs)

- Host-device scheme
- Hierarchical NUMA space
  - Non-Uniform Mem Access

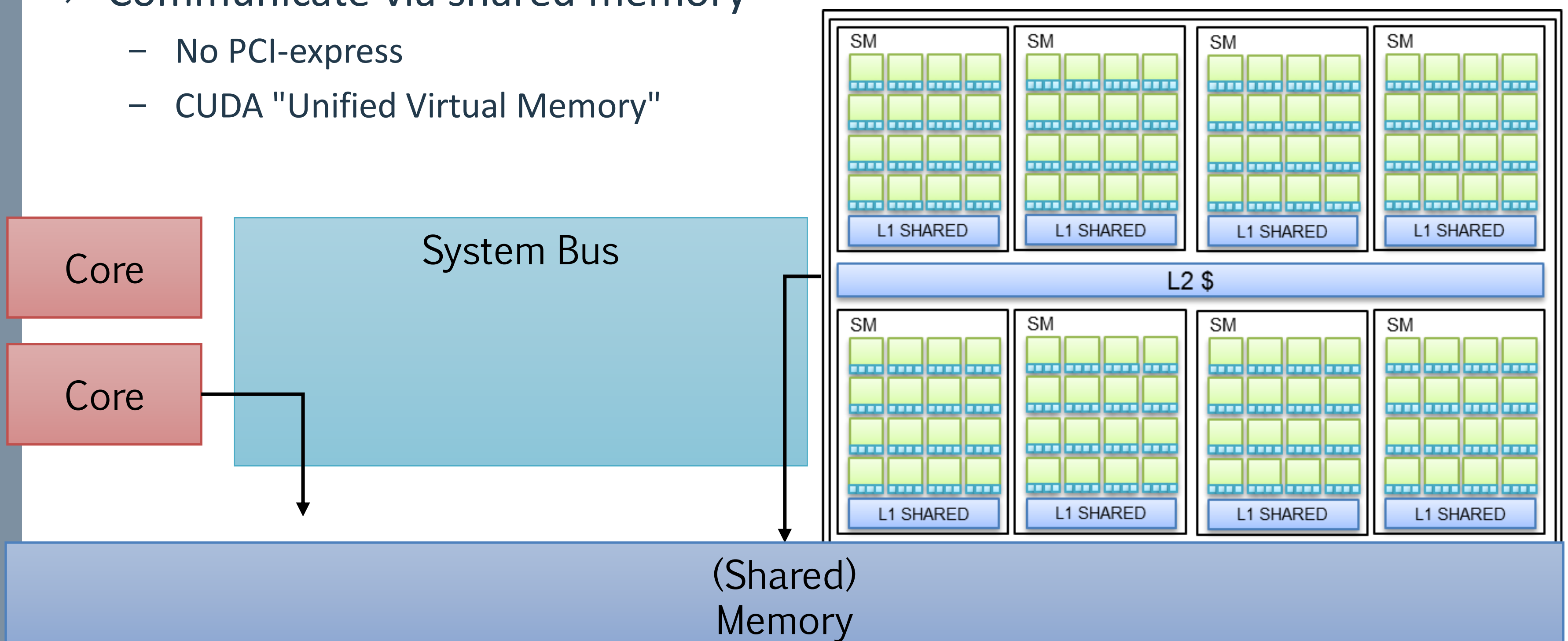




Integrated GPGPUs

GP-GPU based embedded platforms

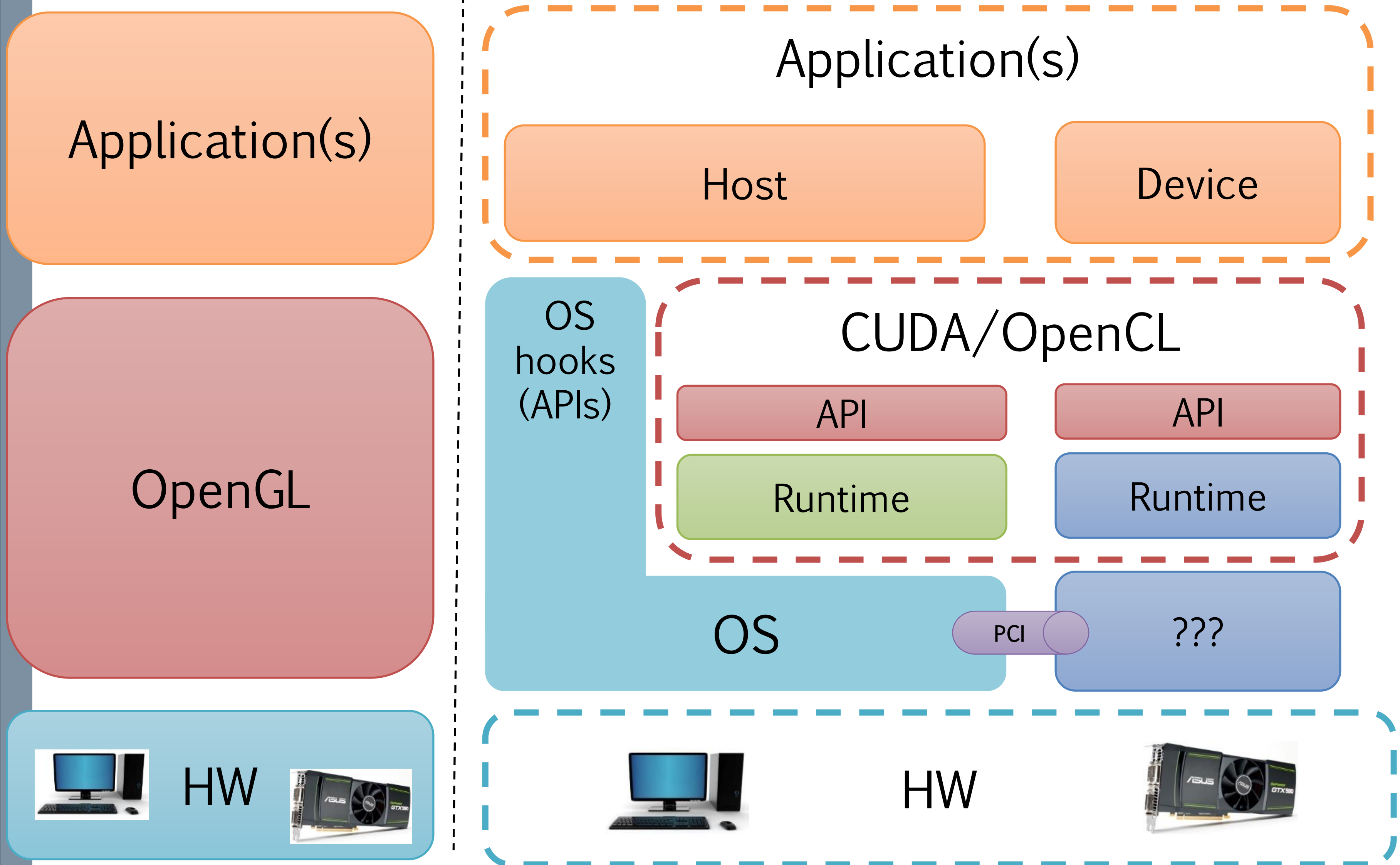
- As opposite to, traditional "discrete" GP-GPUs
- Still, host + accelerator model
- Communicate via shared memory
  - No PCI-express
  - CUDA "Unified Virtual Memory"



To summarize…

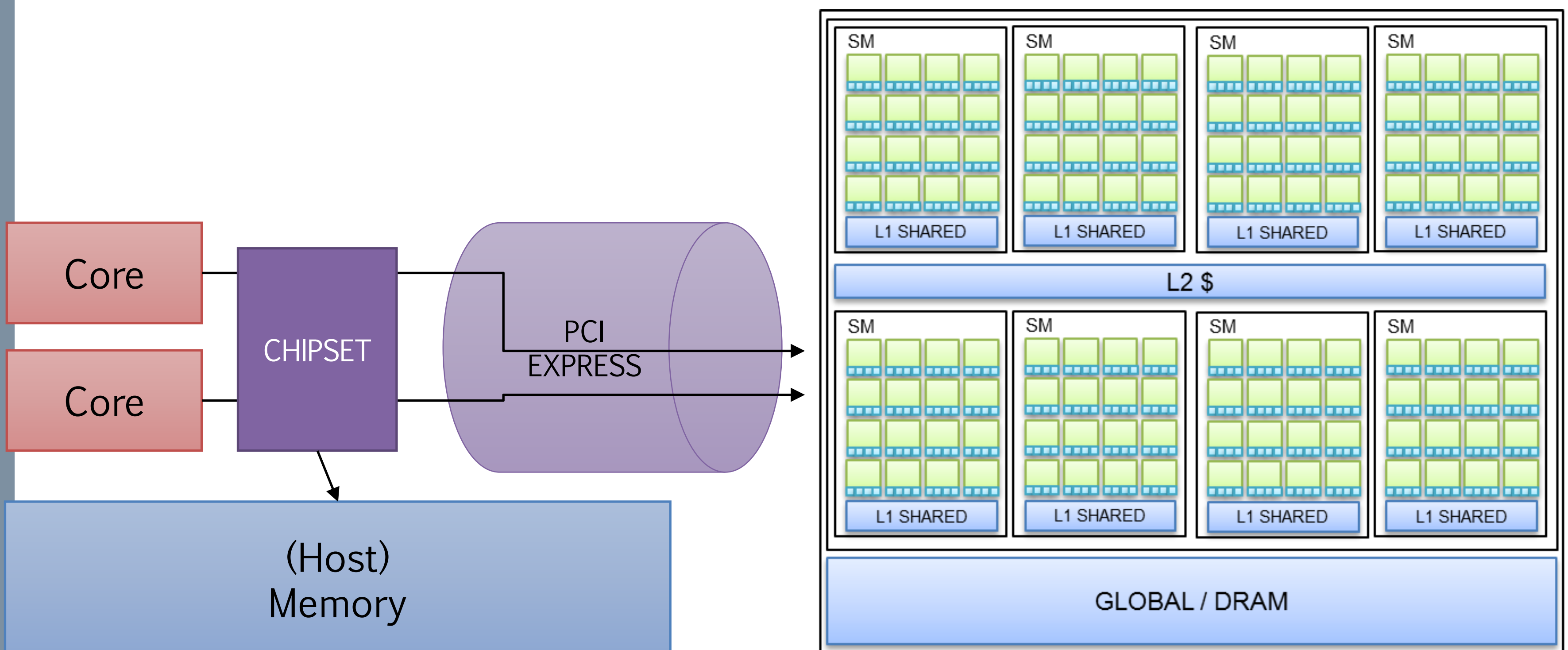
- Tightly-coupled SMs
  - Multiple cores sharing HW resources: L1 cache, Fetch+Decode Unit, (maybe even) Memory controllers, DSPs…
  - GPU "Class" (NVIDIA Kepler, Maxwell, Pascal, Volta..)
  - ~100s cores
- Multiple SMs integrated onto one chip
  - GPU "name" (NVIDIA GTX980, GT640…)
  - 1000s cores
  - NUMA hiearchy
- Typically (but not only) used as co-processor/accelerator
  - PCIEXPRESS connectivity
  - Shared memory

(GP)GPU programming stack



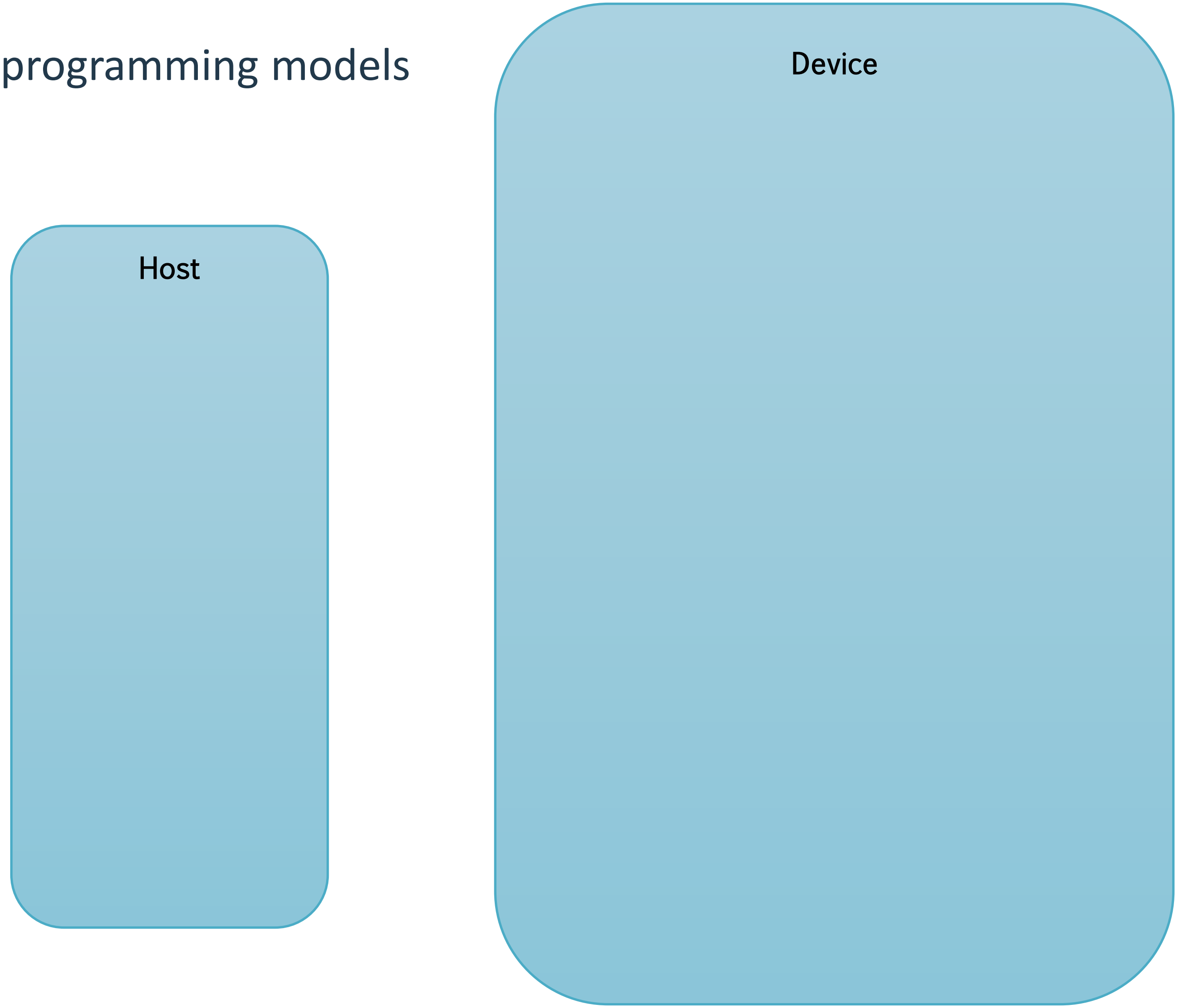
GPU programming

› We need a programming model that provides
  1. Simple offloading subroutines
  2. An easy way to write code which runs on thousand threads
  3. A way to exploit the NUMA hierarchy



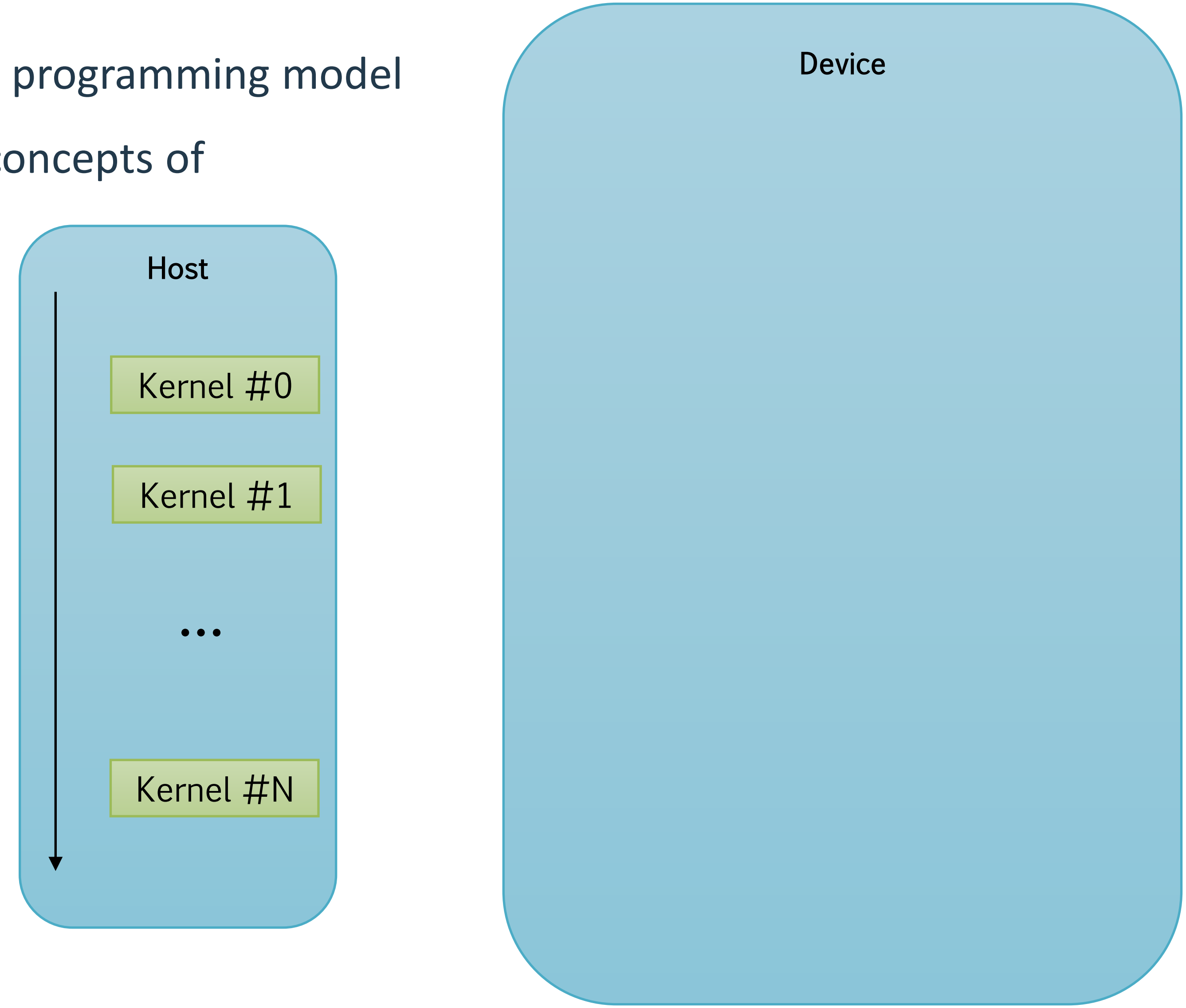
1) Offload-based programming

- Offload-based programming models
  - CUDA
  - OpenCL
  - OpenMP 4.5



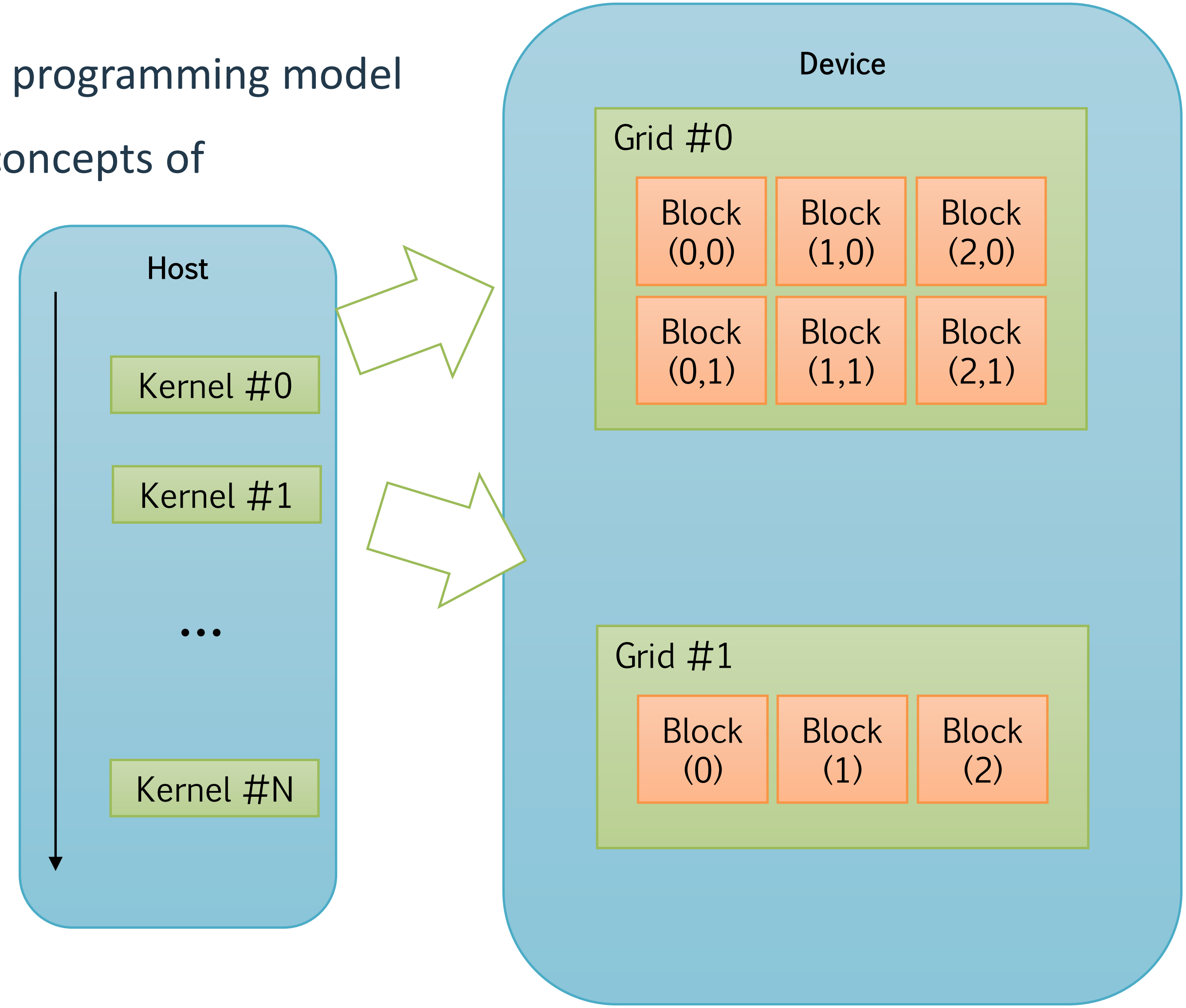
2) Parallelism in CUDA

- Esposed in the programming model
- Based on the concepts of
  - Grid(s)
  - Block(s)
  - Thread(s)



2) Parallelism in CUDA

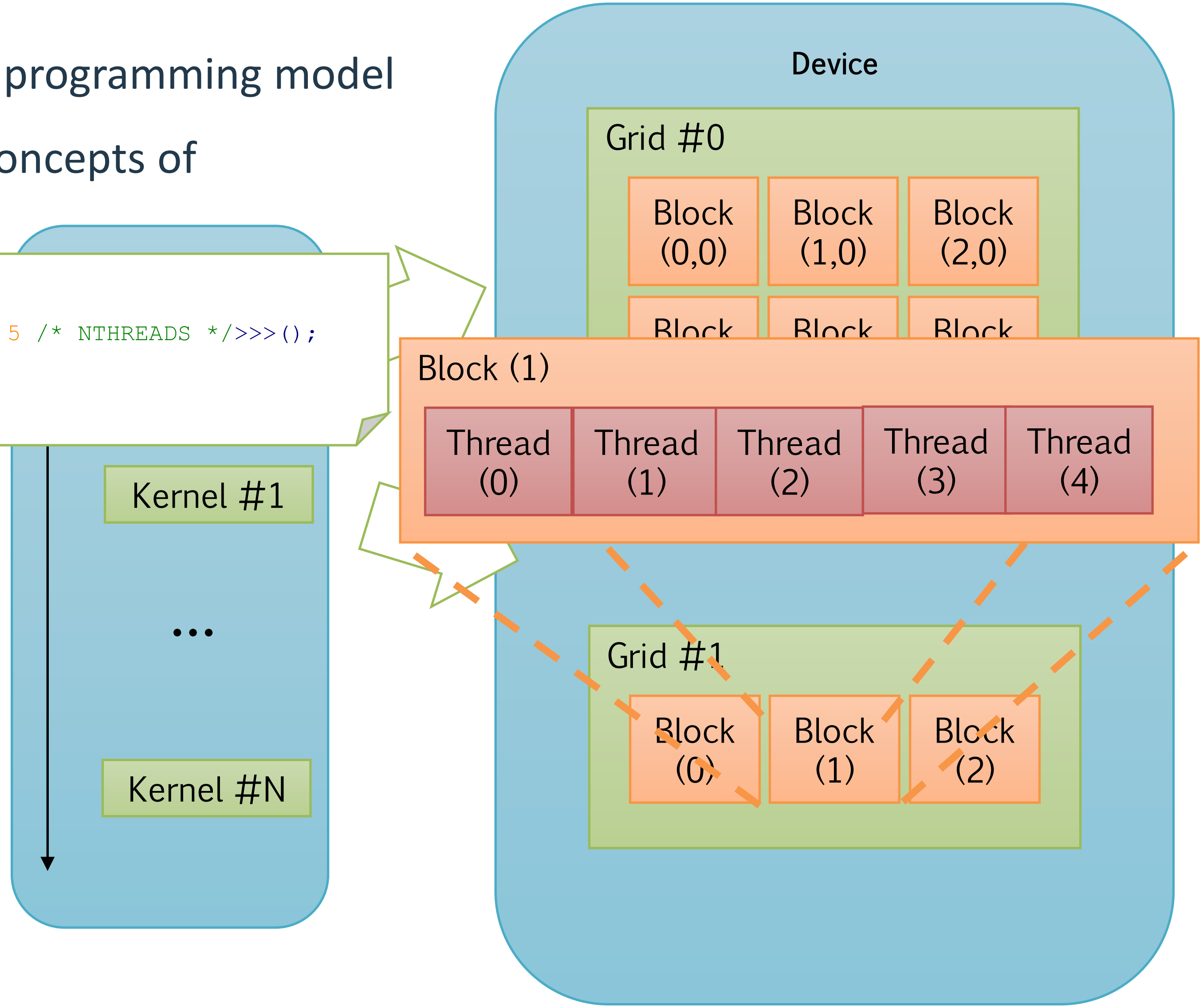
- Esposed in the programming model
- Based on the concepts of
  - Grid(s)
  - Block(s)
  - Thread(s)



2) Parallelism in CUDA

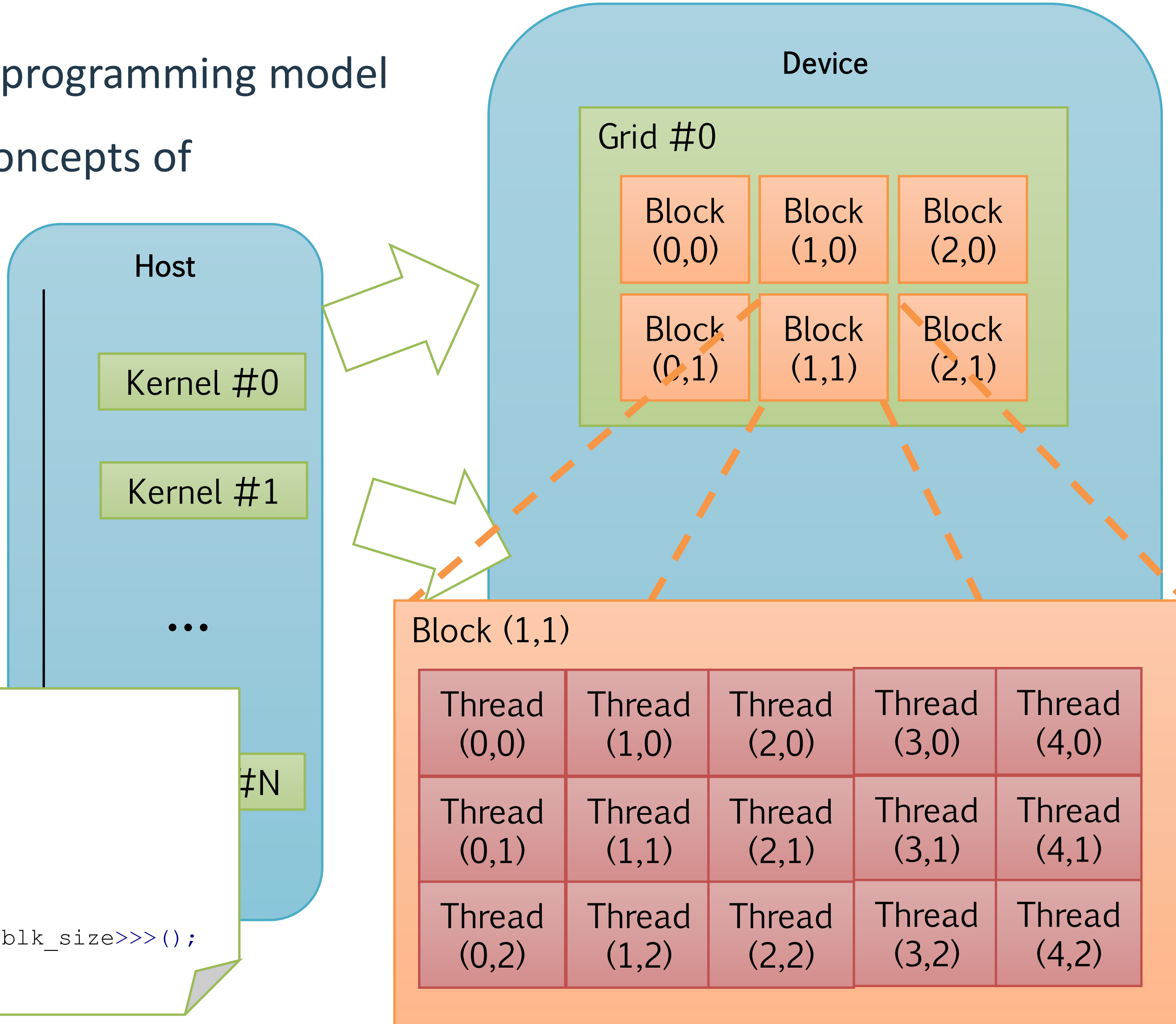
- Esposed in the programming model
- Based on the concepts of
  - Grid(s)

```
myKernel<<<3 /* NBLOCKS */, 5 /* NTHREADS */>>>();
```



2) Parallelism in CUDA

- Esposed in the programming model
- Based on the concepts of
  - Grid(s)
  - Block(s)
  - Thread(s)




```
dim3 grid_size;
grid_size.x = 3;
grid_size.y = 2;

dim3 blk_size;
blk_size.x = 5;
blk_size.y = 3;

myKernel<<<grid_size,blk_size>>>();
```

Complexity of GPUs

- Grids ➔ kernels
- Blocks X Threads represent a "work-space"
  - Synchronization is possible only within the same CUDA Block
    - `__syncthreads()`
  - Each thread retrieves its "point" inside this space, and maps it on a specific
    - Data item, such as array element, matrix element, matrix row…
    - "Job item", such as a function
    - Can be 2x1D, 2x2D, 2x3D: extremely (too much) flexible and scalable

Complexity of GPUs

› Grids ➔ kernels

› Blocks X Threads represent a "work-space"

  – Synchronization is possible only within the same CUDA Block

    › `__synchthreads()`

  – Each thread retrieves its "point" inside this space, and maps it on a specific

    › Data item, such as array element, matrix element, matrix row…

    › "Job item", such as a function

y (too much) flexible and scalable

```
#define GRAY_THRESHOLD 100
#define WHITE 255
#define BLACK 0
void binarizeImage(const unsigned char inputImg[],
                   unsigned char outputImg[],
                   unsigned int imgDim)
{
  for(int i=0; i<imgDim; i++)
    if(inputImg[i] >= GRAY_THRESHOLD)
      outputImg[i] = WHITE;
    else
      outputImg[i] = BLACK;
}
```

Complexity of GPUs

- Grids ➔ kernels
- Blocks X Threads represent a "work-space"
  - Synchronization is possible only within the same CUDA Block
    - `__syncthreads()`
  - Each thread retrieves its "point" inside this space, and maps it on a specific
    - Data item, such as array element, matrix element, matrix row…
    - "Job item", such as a function

```
#define GRAY_THRESHOLD 100
#define WHITE 255
#define BLACK 0
void binarizeImage(const unsigned char inputImg[],
                   unsigned char outputImg[],
                   unsigned int imgDim)
{
  for(int i=0; i<imgDim; i++)
    if(inputImg[i] >= GRAY_THRESHOLD)
      outputImg[i] = WHITE;
    else
      outputImg[i] = BLACK;
}
```

y (too much) flexible and scalable

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP

```
GRAY_THRESHOLD = 150
inputImg[0] = 200
inputImg[1] = 100
```

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

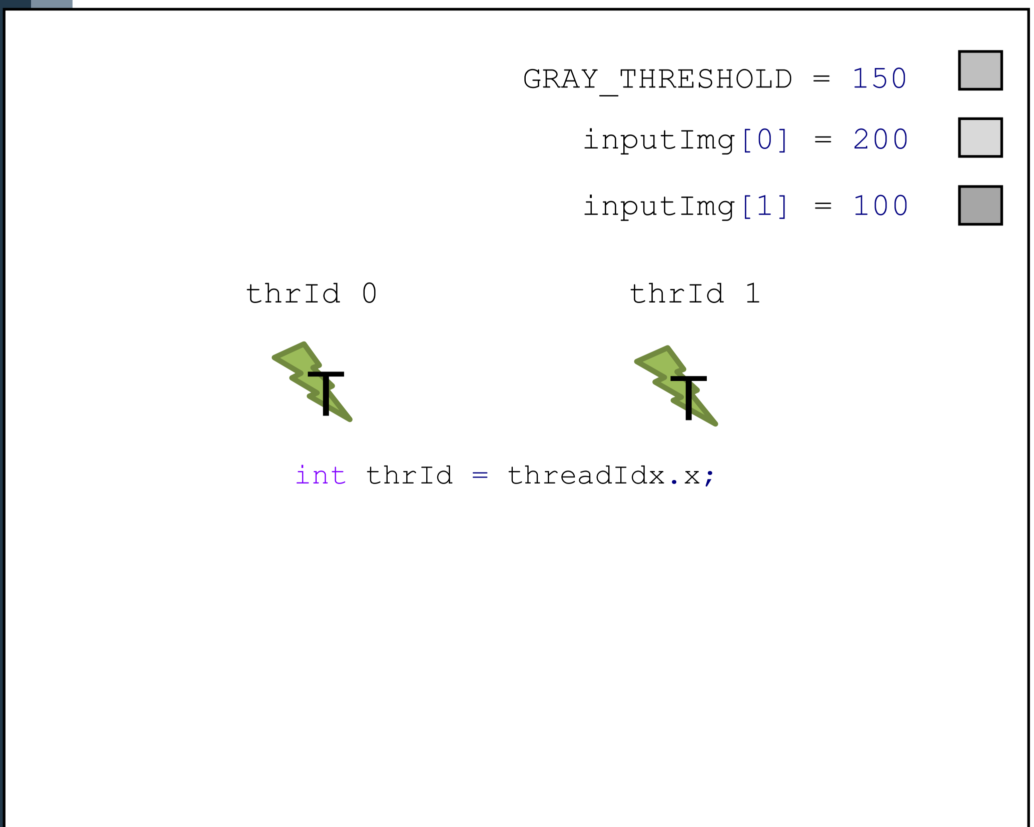
Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP


```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

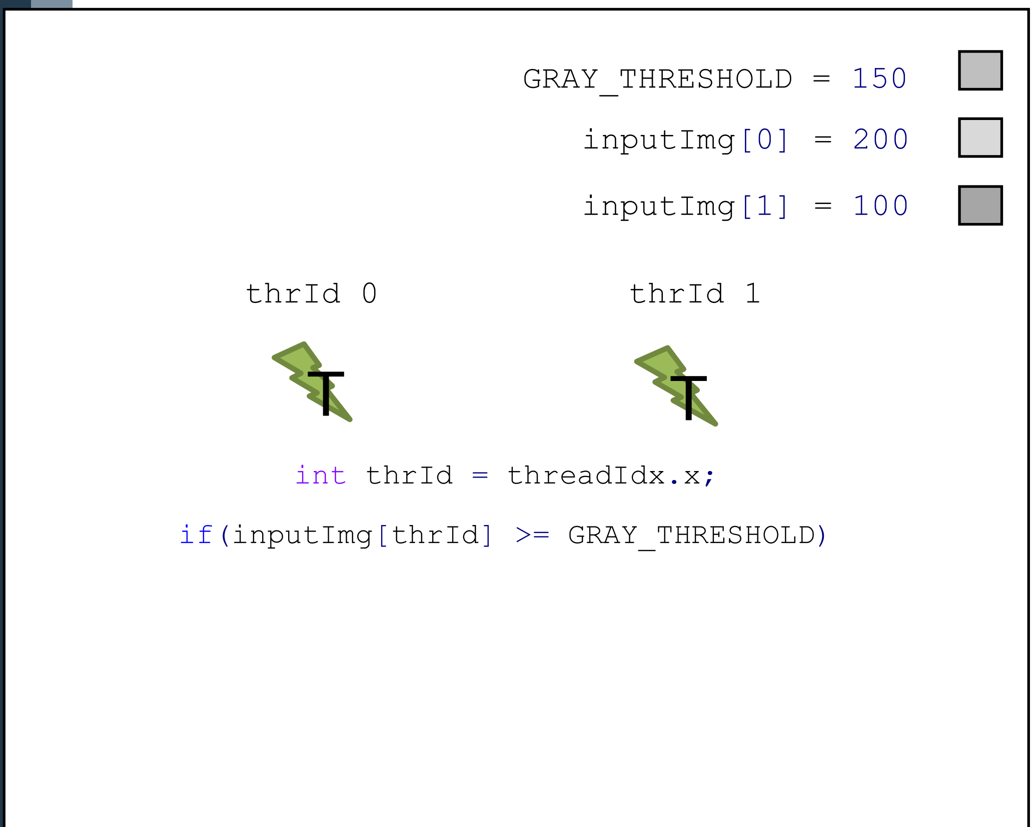
- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP




```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP




```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP


```
GRAY_THRESHOLD = 150
inputImg[0] = 200
inputImg[1] = 100
```

thrId 0

thrId 1

```
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
```

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP


```
GRAY_THRESHOLD = 150
inputImg[0] = 200
inputImg[1] = 100
```

thrId 0 thrId 1

```
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
outputImg[thrId] = WHITE;
```

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP


```
GRAY_THRESHOLD = 150
inputImg[0] = 200
inputImg[1] = 100
```

thrId 0

thrId 1

```
int thrId = threadIdx.x;

if(inputImg[thrId] >= GRAY_THRESHOLD)

outputImg[thrId] = WHITE;            NOP
```

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP

```
GRAY_THRESHOLD = 150
inputImg[0] = 200
inputImg[1] = 100
```

thrId 0   thrId 1


```
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
outputImg[thrId] = WHITE;          NOP
else
```

```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Lockstep

- (Groups of) cores share the same instruction Fetch/Decode Units
  - Ultimately, the same Program Counter!!!
  - Threads cannot do branches - LOCKSTEP

GRAY_THRESHOLD = 150

inputImg[0] = 200

inputImg[1] = 100

thrId 0 — thrId 1

int thrId = threadIdx.x;

if(inputImg[thrId] >= GRAY_THRESHOLD)

outputImg[thrId] = WHITE; — NOP

else

NOP — outputImg[thrId] = BLACK;

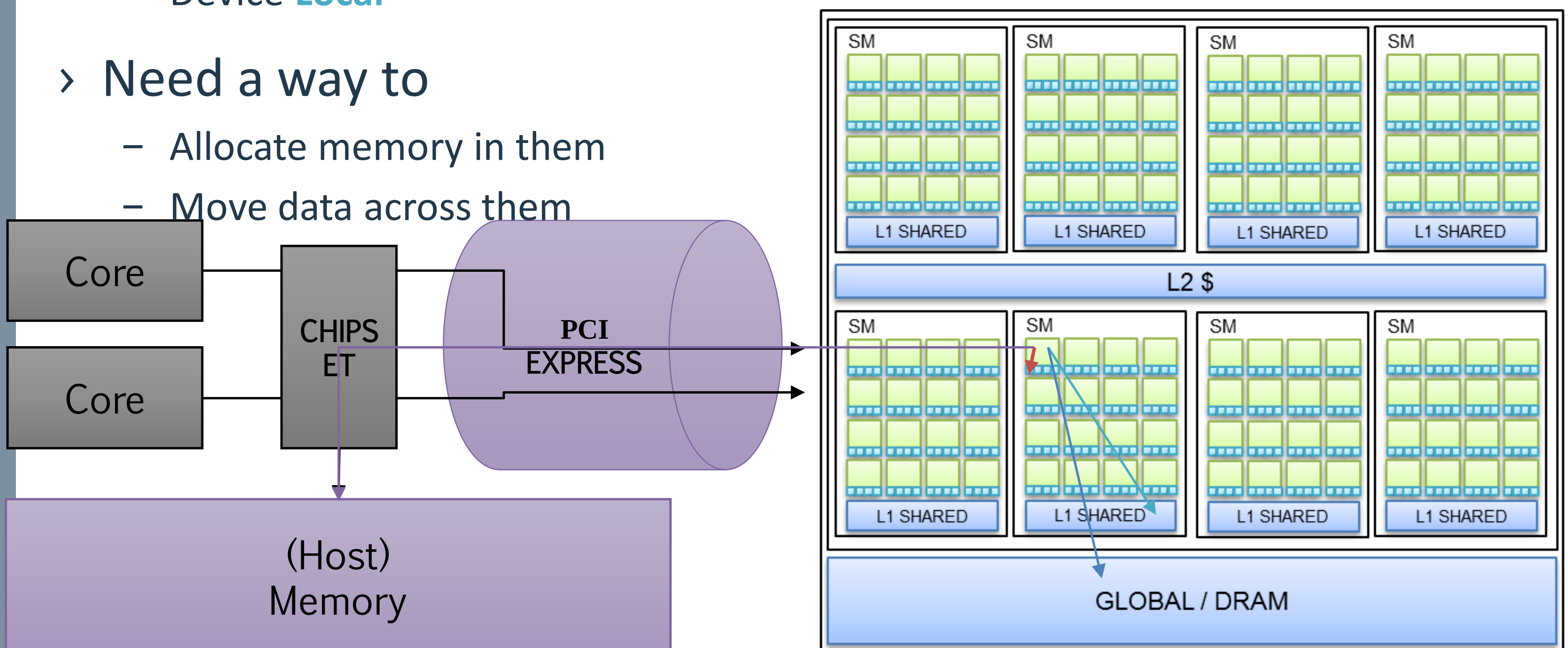
```
/* ... */

// 1 => # Blocks
// imgDim => #Threads
// 1 thread works on each pixel
int thrId = threadIdx.x;
if(inputImg[thrId] >= GRAY_THRESHOLD)
  outputImg[thrId] = WHITE;
else
  outputImg[thrId] = BLACK;

/* ... */
```

Warps, and lockstep

- Threads are grouped in warps
  - 1 warp <-> 32 CUDA threads
  - Units of scheduling
  - Threads of a single blocks are scheduled and de-scheduled 32 by 32
- Threads within the same warp run in LOCKSTEP
- Memory accesses within the single warp are coalesced

3) Exploit NUMA in CUDA

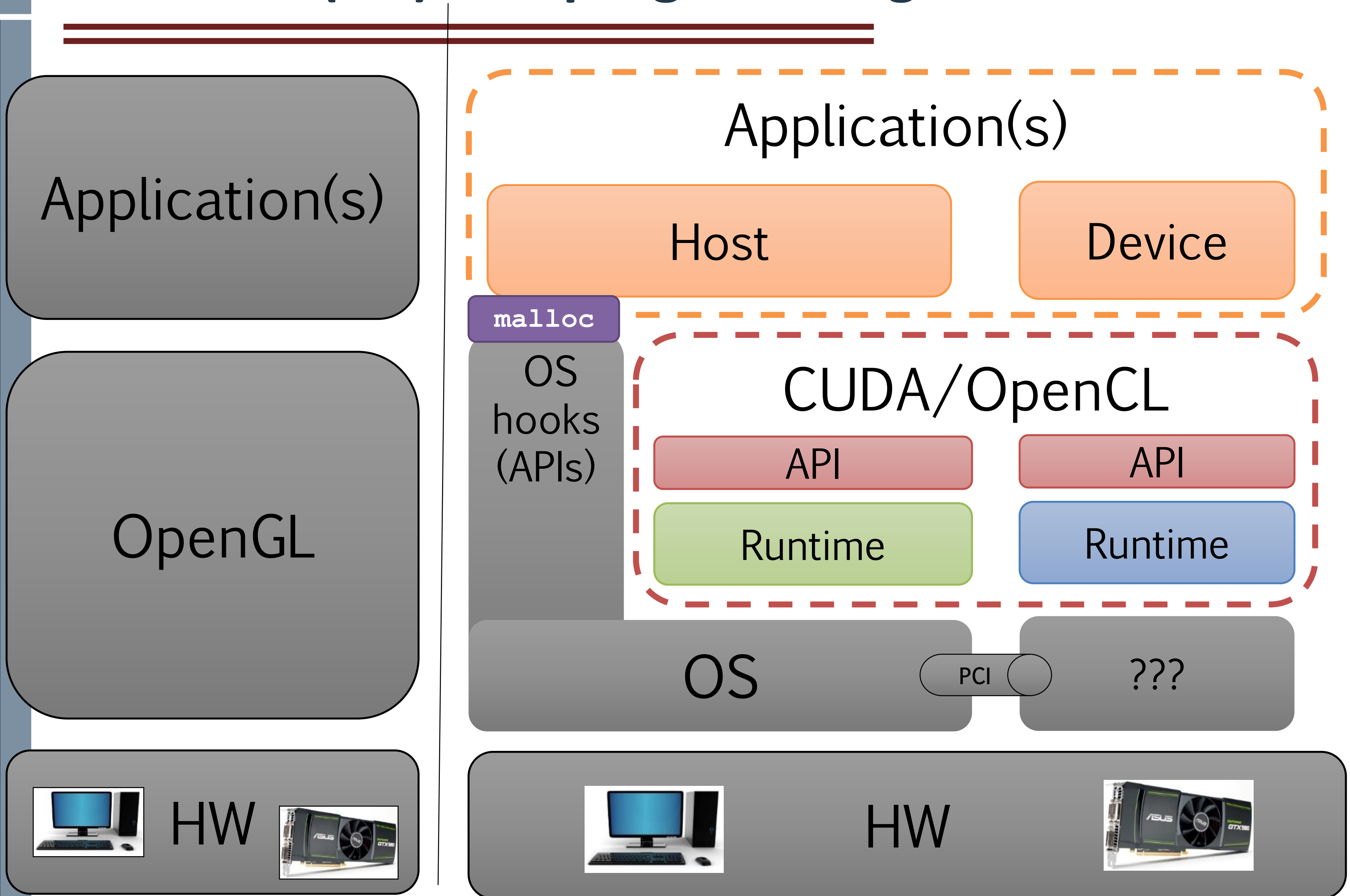
- Four memory spaces
  - **Host**
  - Device **Global**
  - Device **Shared**
  - Device **Local**
- Need a way to
  - Allocate memory in them
  - Move data across them



GPU memory size

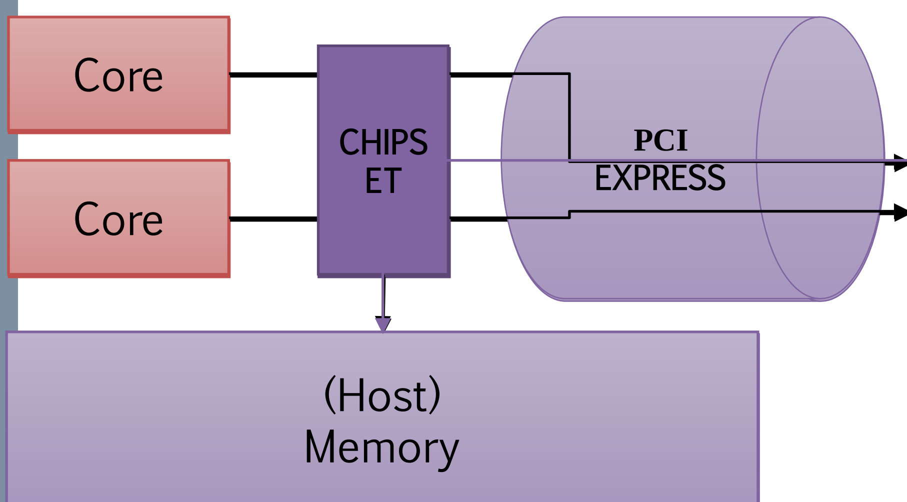
	GeForce GT 640 : Liu	GeForce GTX 980 : Turing
Microarchitettura	Kepler	Maxwell
Versione capacità di calcolo	3.0	5.2
Core CUDA	384	2048
Clock del processore	891 MHz	1126 MHz
Clock grafico	900 MHz	1216 MHz
Global memory	2047 MB	4095 MB
Constant memory	64 KB	64 KB
Shared memory per multiprocessor	48 KB	96 KB
Local memory per thread	512 KB	512 KB
Registri a 32-bit per multiprocessor	32 KB	64 KB
Velocità della memoria	1.8 Gbps	7.0 Gbps
Interfaccia della memoria	128-bit DD3	256-bit GDDR5
Supporto del bus	PCI-E 3.0	PCI-E 3.0

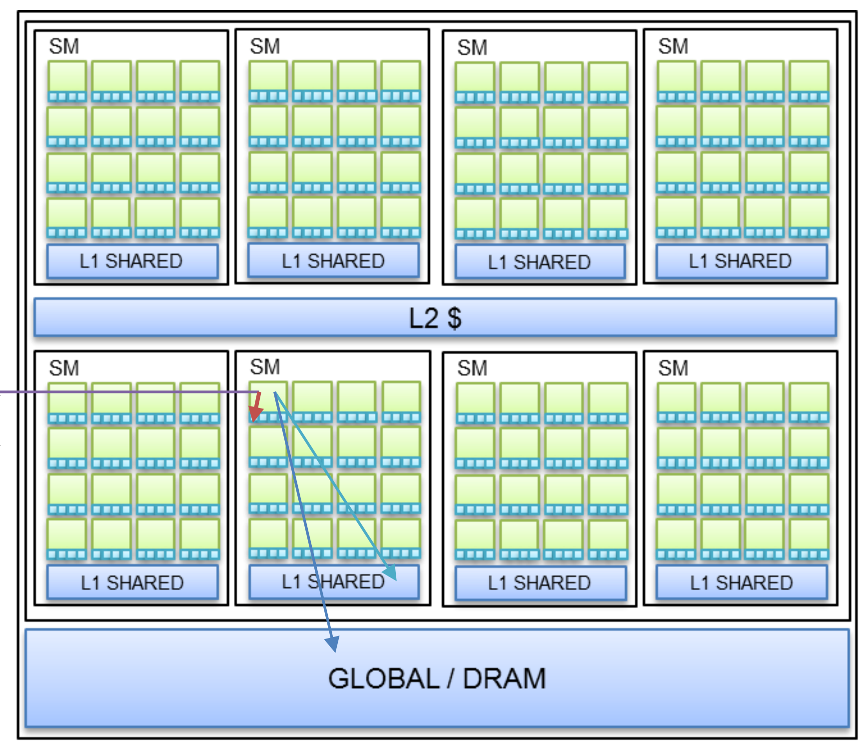
(GP)GPU programming stack



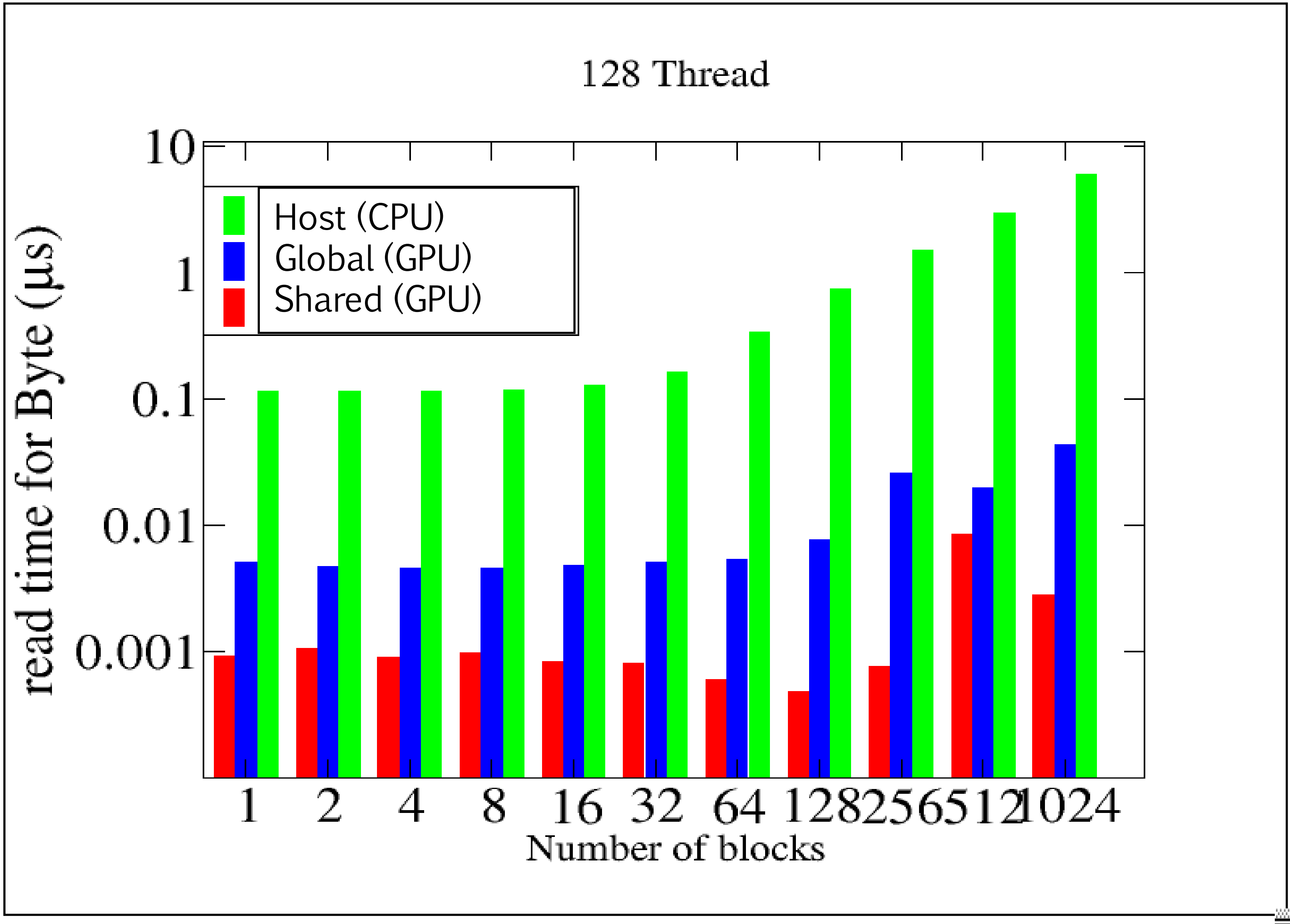
3) Exploit NUMA in CUDA

- Runtime must be aware of all
- Memory allocations
  - `cudaHostAlloc` ➔ Host mem
  - `cudaMalloc` ➔ Global mem
  - `__shared__` keyword ➔ Shared mem
- Data movements
  - `cudaMemcpy`
  - `cudaMemcpyAsync`



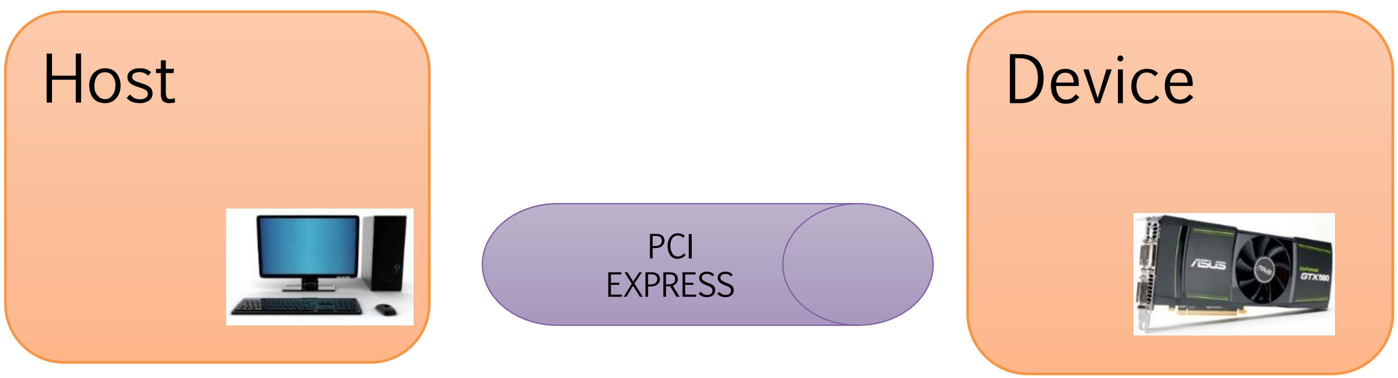


Non-Uniform Access Time



OpenCL

- Open Computing Language
  - More verbose than CUDA
- More "library-based" approach
- Different artifacts for managing parallelism
  - CUDA blocks, Threads
  - OpenCL Work Groups, work items



CUDA vs. OpenCL - Offload code

```
/* Create Command Queue */
command_queue = clCreateCommandQueue(context, device_id, 0, &ret);

/* Create Kernel Program from the source */
program = clCreateProgramWithSource(context, 1, (const char **)&source_str,
                                    (const size_t *) &source_size, &ret);

/* Build Kernel Program */
ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);

/* Create OpenCL Kernel */
kernel = clCreateKernel(program, "hello", &ret);

/* Execute OpenCL Kernel */
ret = clEnqueueTask(command_queue, kernel, 0, NULL,NULL);
```

```
helloworld<<<3,5>>>();

cudaDeviceSynchronize()
```

CUDA vs. OpenCL - Kernel code

```
__kernel void helloworld()
{
  int wiId = get_local_id(0);
  int wgId = get_group_id(0);
  int wiMum = get_local_size(0);
  int wgNum = get_num_groups(0);

  printf("\t\t\t\t\t[DEVICE] Hello World! \
         I am Work Item #%d out of %d, \
         and I belong to Work Group #%d out of %d\n",
         wiId, wiNum, wgId, wgNum);
  return;
}
```

OpenCL

NVIDIA CUDA

```
__global__ void helloworld()
{
  int thrId = threadIdx.x;
  int blkId = blockIdx.x;
  int thrNum = blockDim.x;
  int blkNum = gridDim.x;

  printf("\t\t\t\t\t[DEVICE] Hello World! \
         I am thread #%d out of %d, \
         and I belong to block #%d out of %d\n",
         thrId, thrNum, blkId, blkNum);

  return;
}
```

How to run the examples

Let's
code!

- Download the `Code/` folder from the course website

- Compile

```
$ nvcc code.c
```

- Run
  - ./file.exec

References

Course website

- http://hipert.unimore.it/people/paolob/pub/Industrial_Informatics/index.html

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/

Resources

- http://www.nvidia.it/object/cuda-parallel-computing-it.html
- https://www.khronos.org/opencl
- A "small blog"
  - http://www.google.com