# TAD: Python y CUDA en Computación de Altas Prestaciones

Alejandro Samarín
Lionel Aster Mena García
Sergio Armas Pérez

# Introducción

- Una GPU es un procesador especializado diseñado para el tratamiento gráfico.
- Se puede utilizar para manipular datos de aplicaciones ajenas al procesamiento gráfico (GPGPU).
- Podemos encontrar GPU en tarjetas gráficas, placas base e, incluso, integradas en algunas CPU.

# Introducción

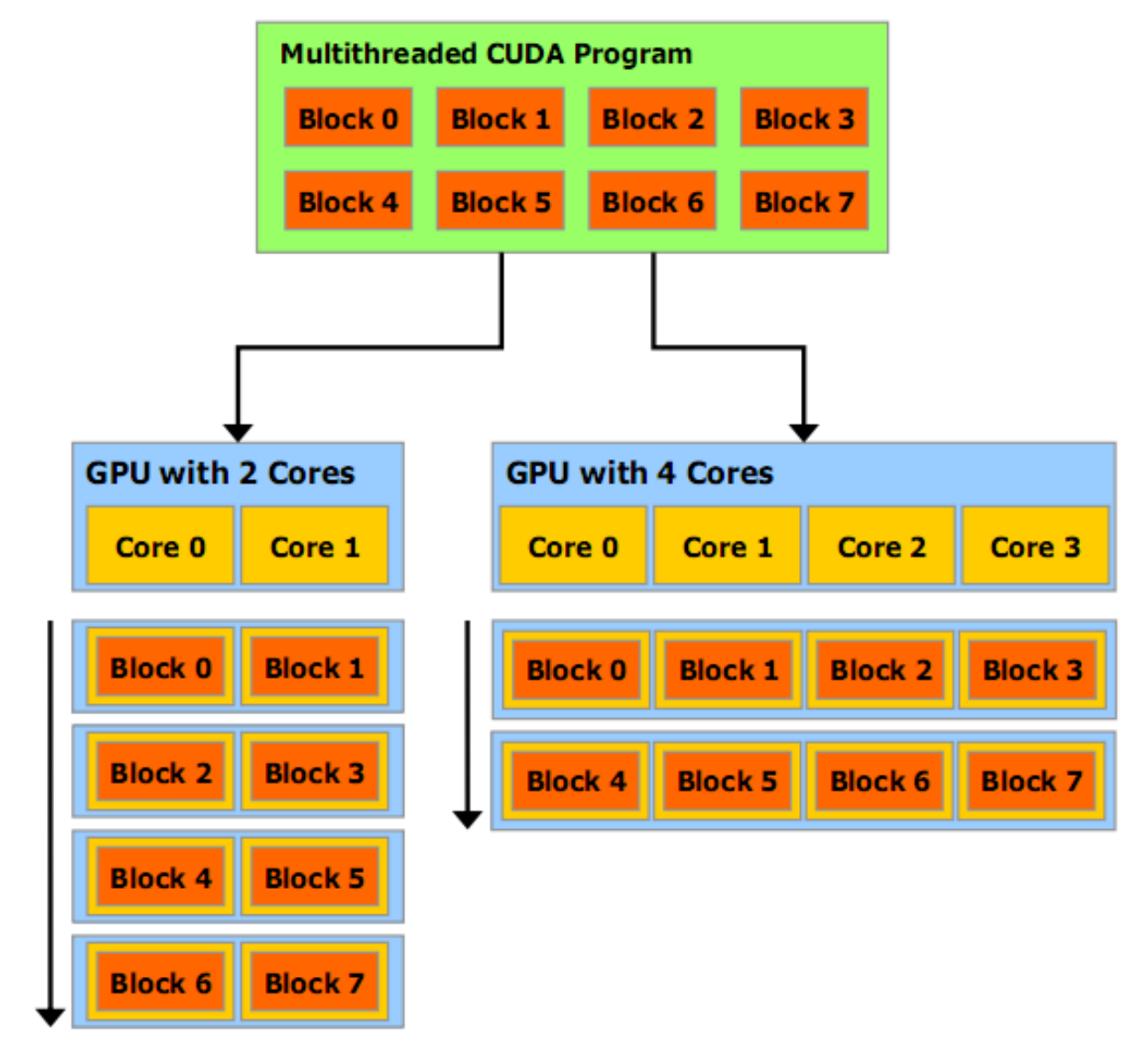
- CUDA (Compute Unified Device Architecture) es una estructura de computación paralela.
- El lenguaje que se emplea una variación de C.
- Diversos programadores, ajenos a NVIDIA, han creado wrappers para CUDA en Java, Perl…

# Introducción

- PyCUDA es un *wrapper* de Python para CUDA desarrollado por Andreas Klöckner.
- Python es un lenguaje interpretado de muy alto nivel cuyo uso está en auge.
- La librería SciPy provee interesantes funciones.

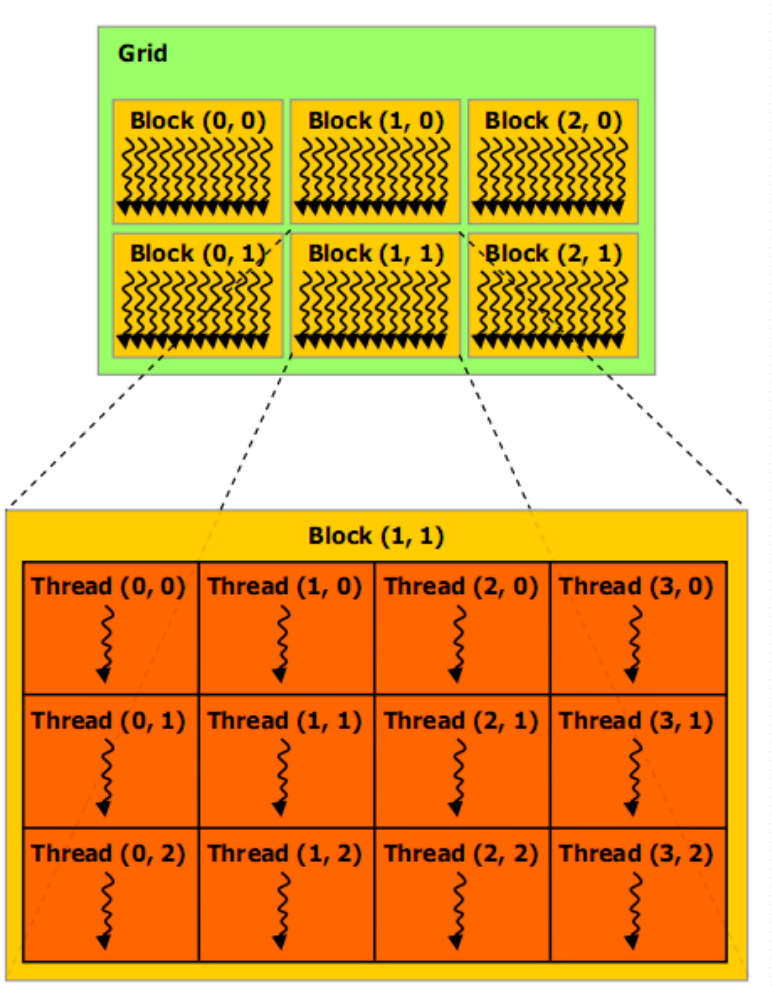
# Modelo CUDA

- Escalabilidad del paralelismo, basado en tres puntos claves:
  - Jerarquía de hilos.
  - Memoria compartida.
  - Sincronización por barrera.


Multithreaded CUDA Program
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 2 Cores
Core 0
Core 1
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
GPU with 4 Cores
Core 0
Core 1
Core 2
Core 3
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7

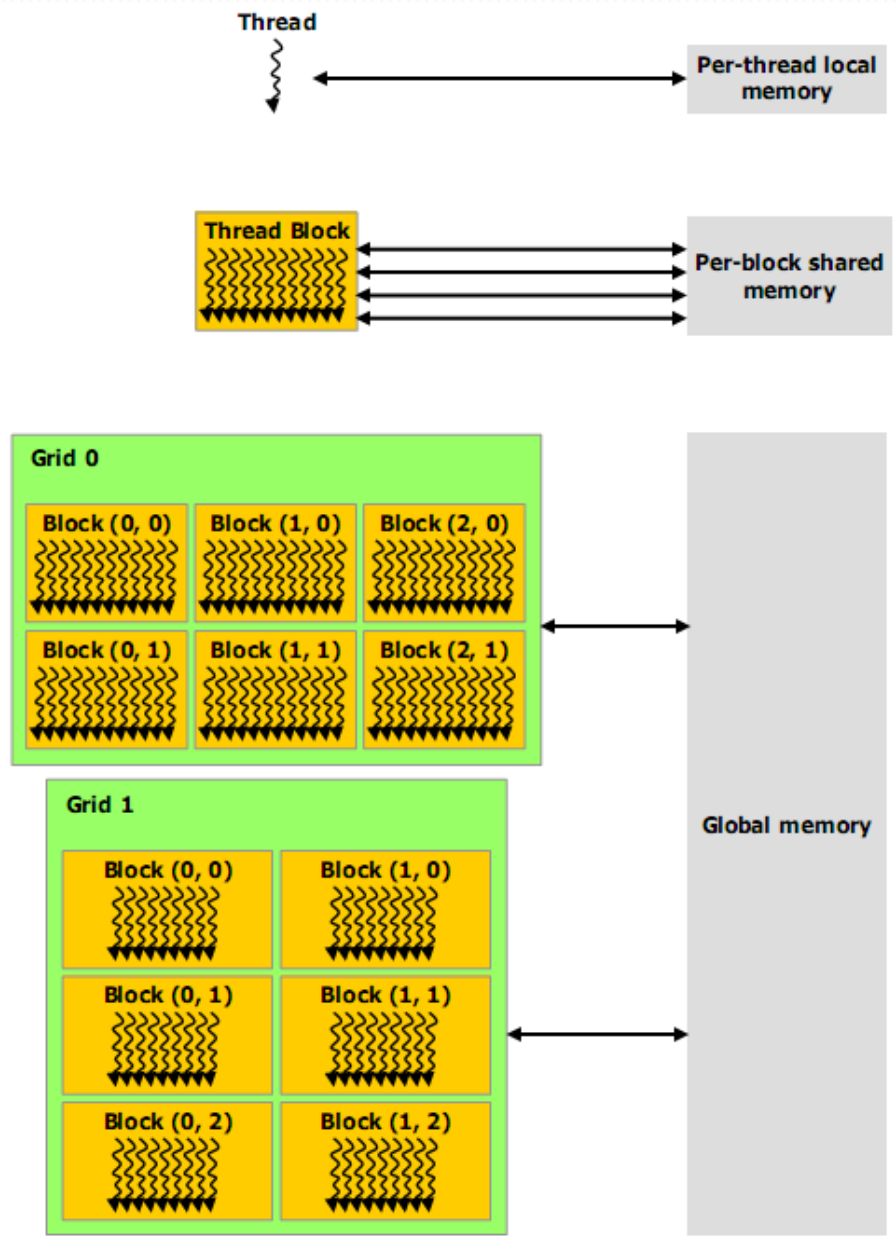
# Modelo CUDA: Hilos

- Los hilos están contenidos en *Bloques*, y los bloques dentro de *Grids*.
- Identificador de hilo *threadIdx*.
- Identificador de bloque *blockIdx*.



# Modelo CUDA: Memoria

- Cada hilo posee su **memoria local privada**.
- Cada bloque de hilos posee su **memoria compartida**.
- Todos los hilos pueden acceder a la **memoria global**.
- Adicionalmente existen 2 tipos de memorias de solo lectura y acceso global: *memoria de texturas* y *memoria constante*.


Thread
Per-thread local memory
Thread Block
Per-block shared memory
Grid 0
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 1
Block (0, 0)
Block (1, 0)
Block (0, 1)
Block (1, 1)
Block (0, 2)
Block (1, 2)
Global memory

# Modelo CUDA

- Kernels: ejecutado por hilos en paralelo.
- Sincronización por Barrera.
- Estructura Host-Device:
  - Hilos CUDA se ejecutan en device.
  - Resto del programa en el host.
  - Espacios de memoria propios.
  - Transferencia de datos host-device.

# Modelo CUDA: Ventajas

- Incrementar el número de núcleos computacionales.
- Provee de granularidad fina en el paralelismo de los datos y los hilos.
- Extiende el lenguaje con un conjunto reducido de instrucciones.

# Recursos Hardware

| | NVIDIA Tesla C2050 | NVIDIA Tesla C1060 |
|---|---|---|
| Capacidad de cómputo | 2.0 | 1.3 |
| Numero de Multiprocesadores/núcleos | 14 (32 núcleos) | 30 (8 núcleos) |
| Total de Núcleos | 448 | 240 |
| Memoria Global | 2.62Gb | 4Gb |
| Memoria Compartida/bloque | 48Kb | 16Kb |
| Máximo hilos/bloque | 1024 | 512 |
| Dimensión Max. bloque | 1024 x 1024 x 64 | 512 x 512 x 64 |
| Dimensión Max. grid | 65535 x 65535 x 1 | 65535 x 65535 x 1 |

# «Hola mundo» en PyCUDA

- Inclusión de las librerías necesarias.

```
import pycuda.autoinit
import pycuda.driver as cuda
from pycuda.compiler import SourceModule
```

- Carga de los datos en la memoria.

```
import numpy
a = numpy.random.randn(4,4).astype(numpy.float32)
```

- Reserva de espacio en el dispositivo.

```
a_gpu = cuda.mem_alloc(a.nbytes)
```

# «Hola mundo» en PyCUDA

- Transferencia de datos al *device*.

```
cuda.memcpy_htod(a_gpu, a)
```

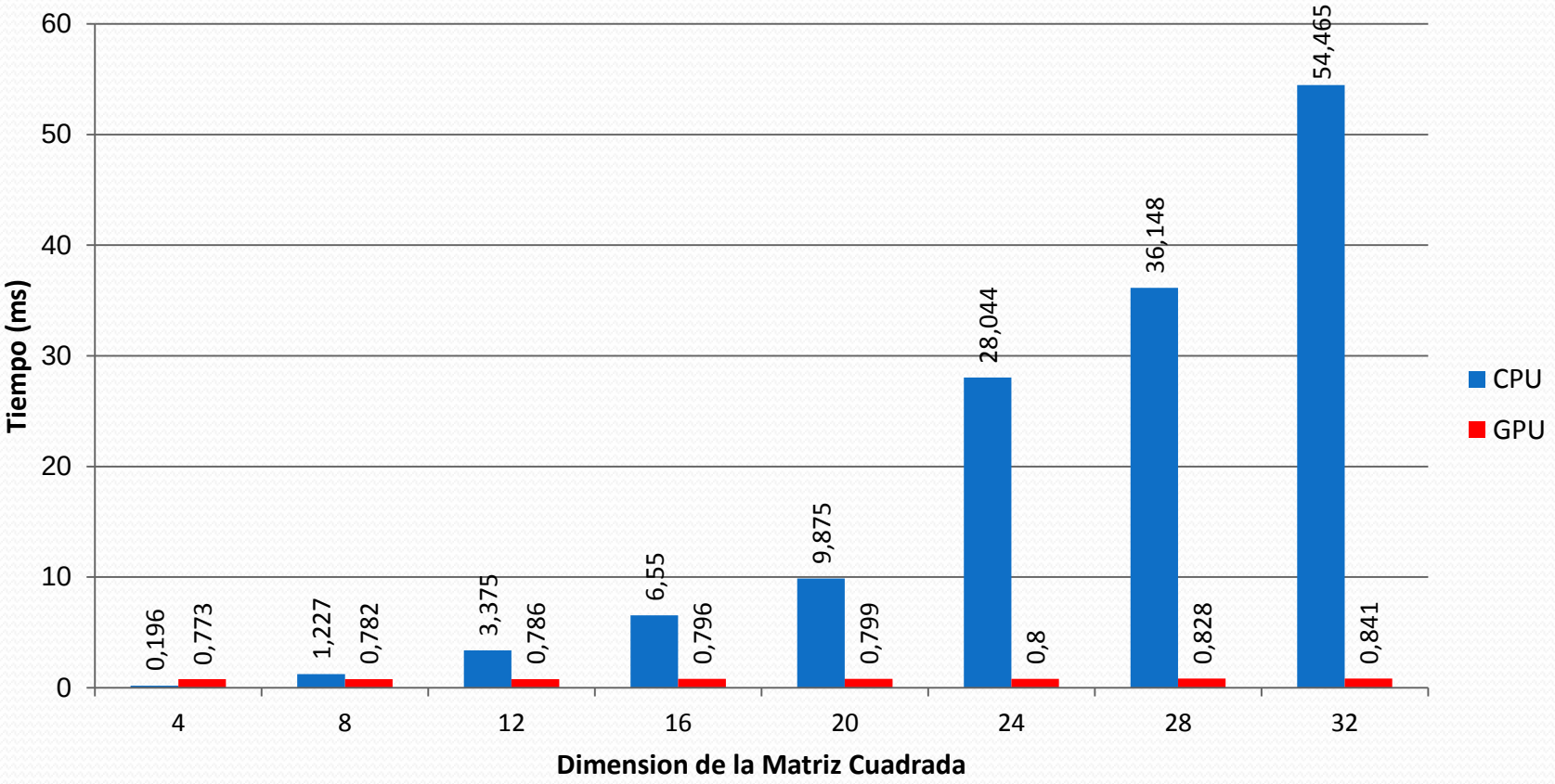
- Ejecución del kernel.

```
func = mod.get_function("kernel_name")
func(a_gpu, block=(4,4,1))
```

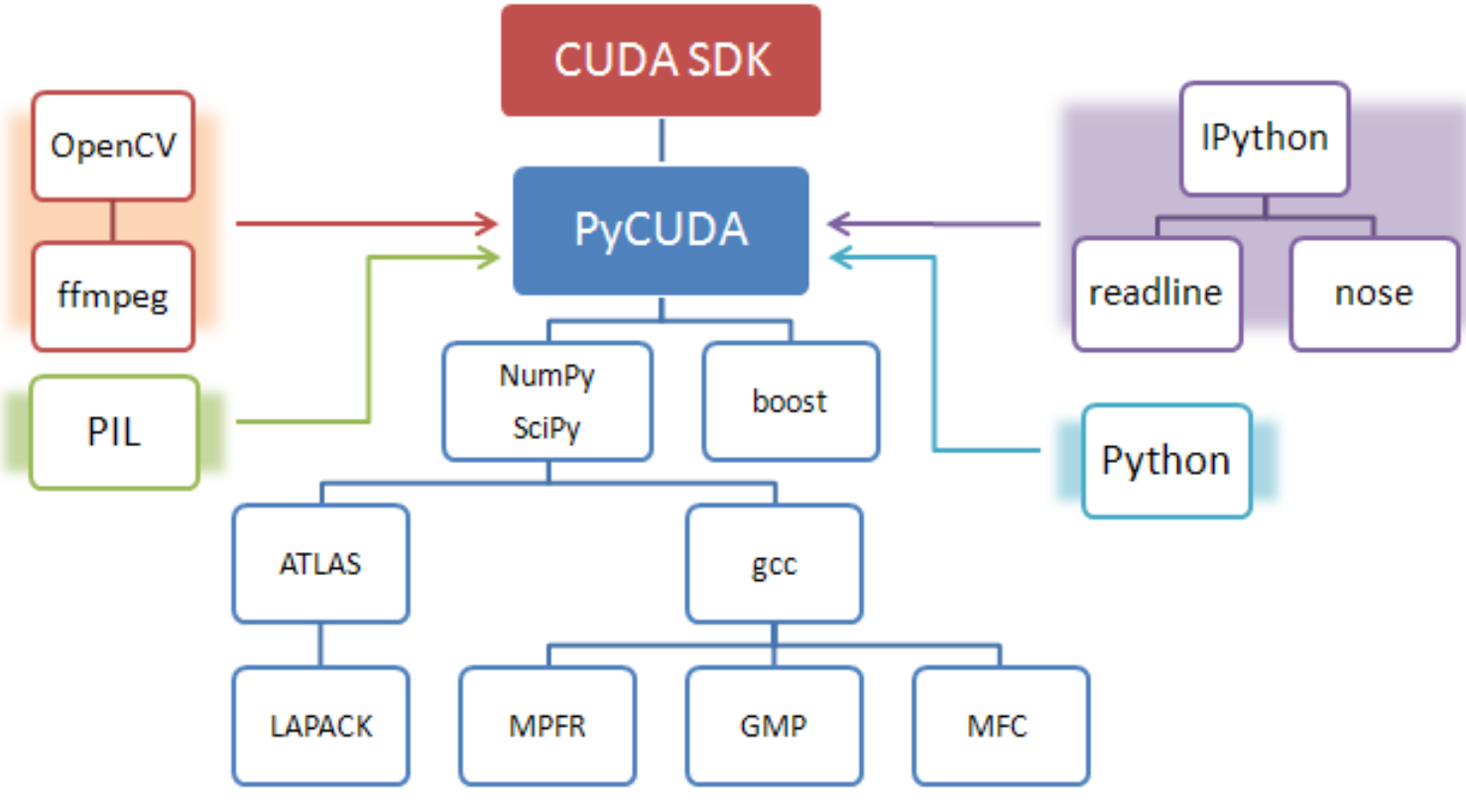
- Transferencia al *host*.

```
a_doubled = numpy.empty_like(a)
cuda.memcpy_dtoh(a_doubled, a_gpu)
```

# CPU vs GPU


60
50
40
30
20
10
0
Tiempo (ms)
0,196
0,773
1,227
0,782
3,375
0,786
6,55
0,796
9,875
0,799
28,044
0,8
36,148
0,828
54,465
0,841
4
8
12
16
20
24
28
32
Dimension de la Matriz Cuadrada
CPU
GPU

# Software Framework

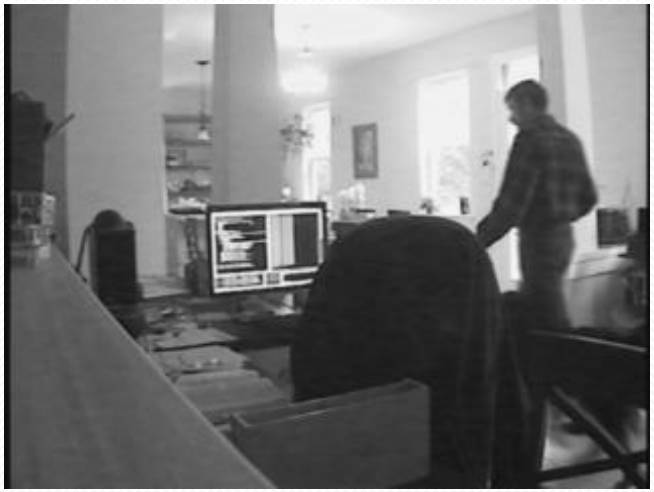


# Algoritmo de Detección de Movimiento

- Dados 2 frames del video, aplicar filtros consecutivamente:
  - Conversión a escala de grises
  - Filtro de Diferencia
  - Filtro Threshold
  - Filtro Erosion
  - Fusión en el canal R

# Algoritmo de Detección de Movimiento

- Conversión a escala de grises



- Filtro de Diferencia

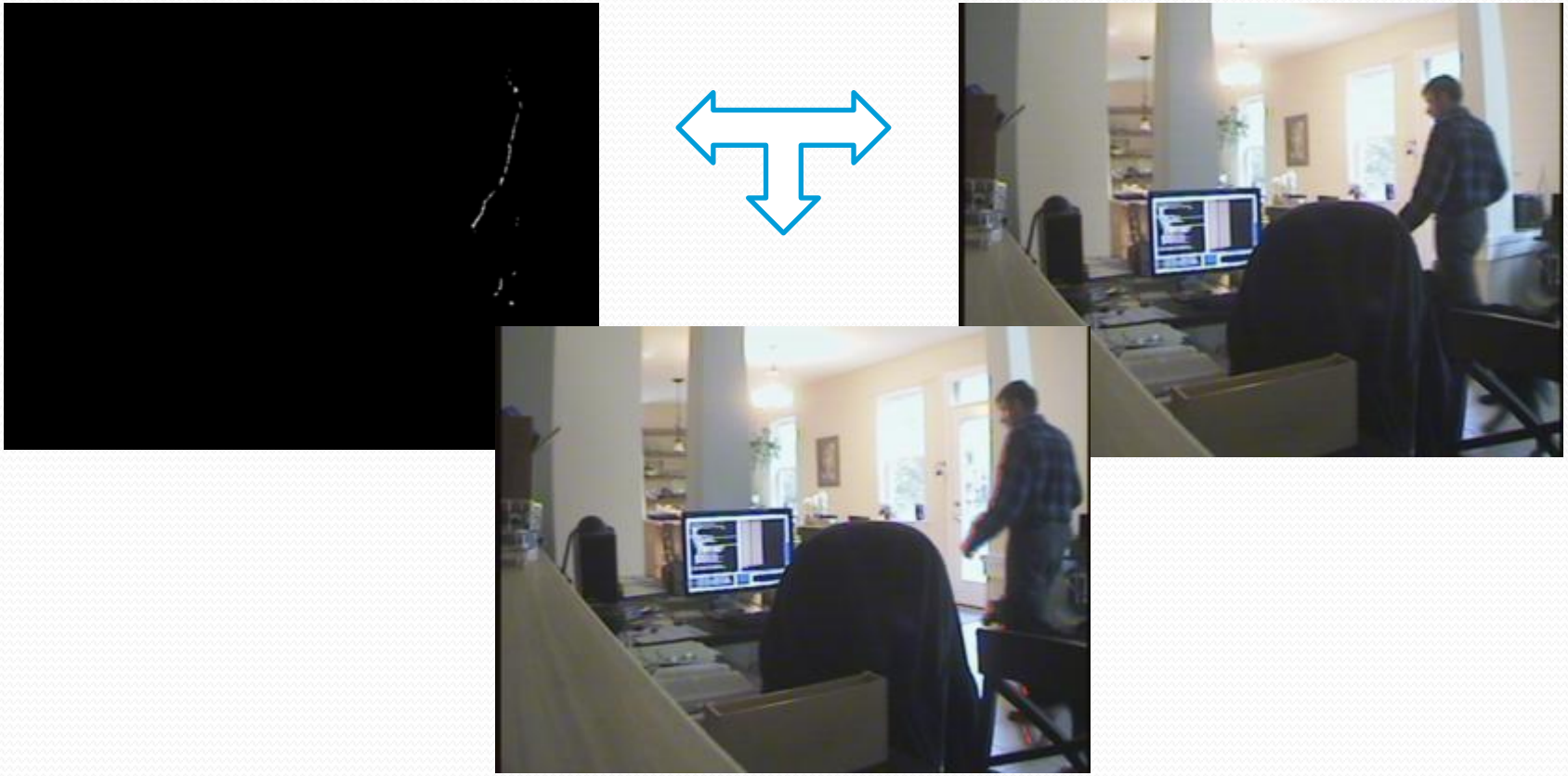
# Algoritmo de Detección de Movimiento

- Filtro Threshold

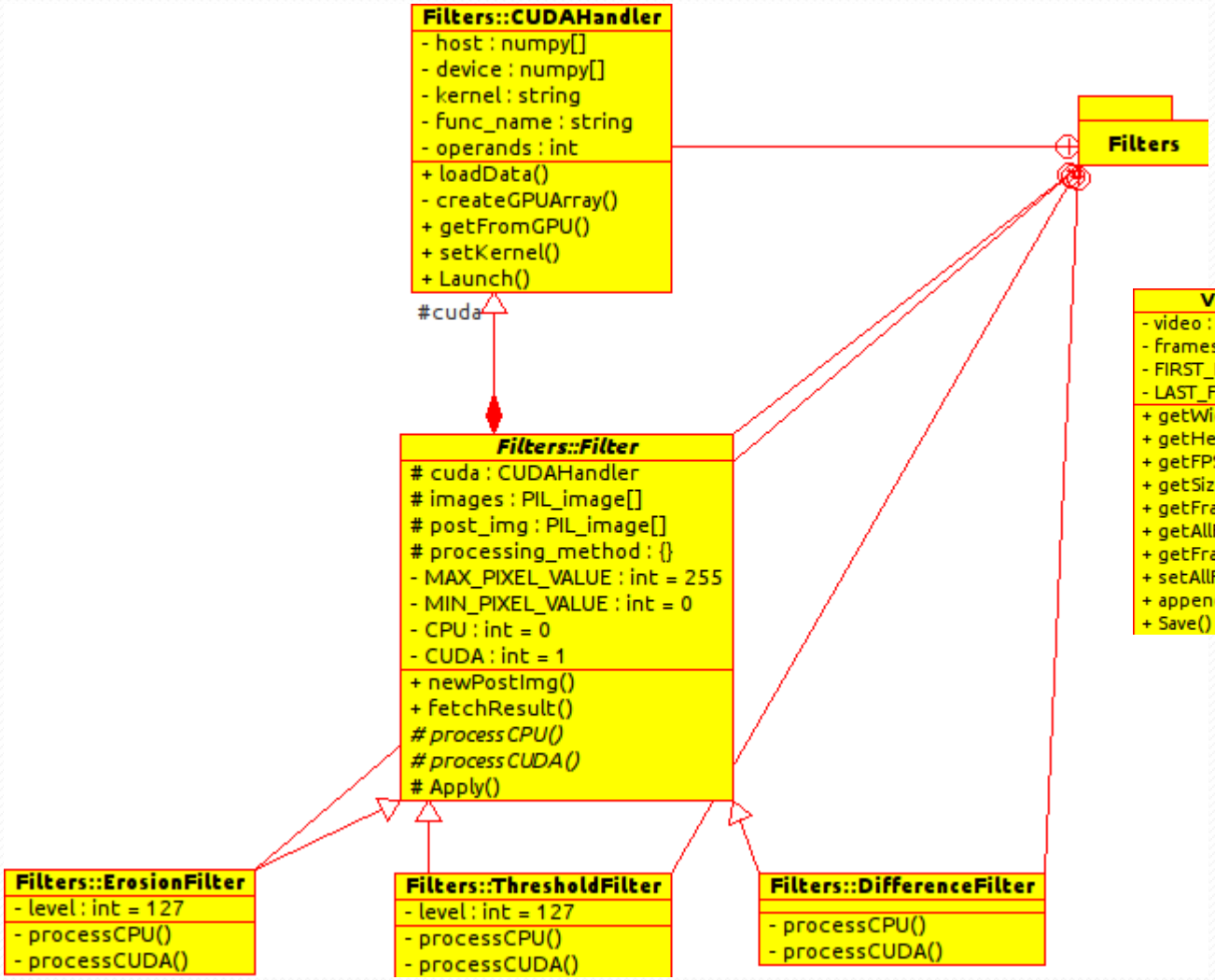
- Filtro de Erosión

# Algoritmo de Detección de Movimiento

- Fusión en el canal R de la imagen original



# Jerarquía de Clases



# Resultado

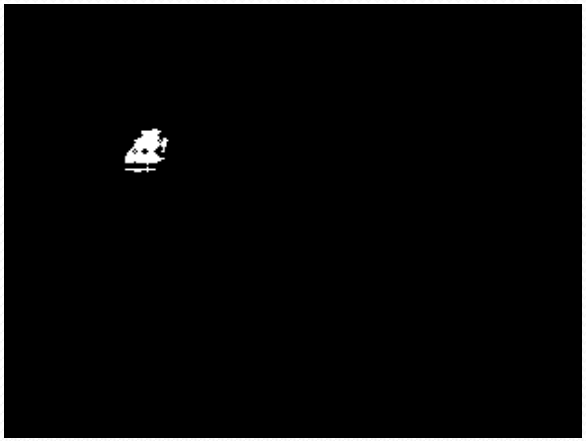
**Frames consecutivos de un vídeo**

# Resultado

**Difference**

**Threshold**

**Erosion**



# Resultado

**Detección de movimiento**

# Preguntas

# Gracias por su atención