

Procesadores Gráficos Avanzados

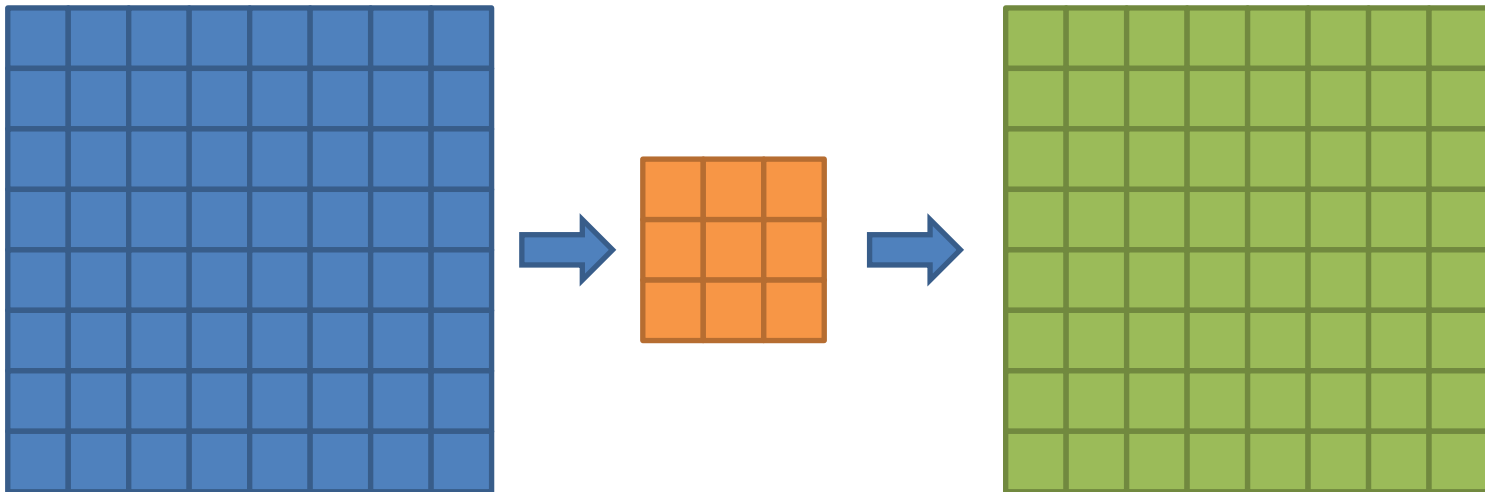
Práctica – CUDA

Dr. Antonio Sanz Montemayor

David Concha Gómez

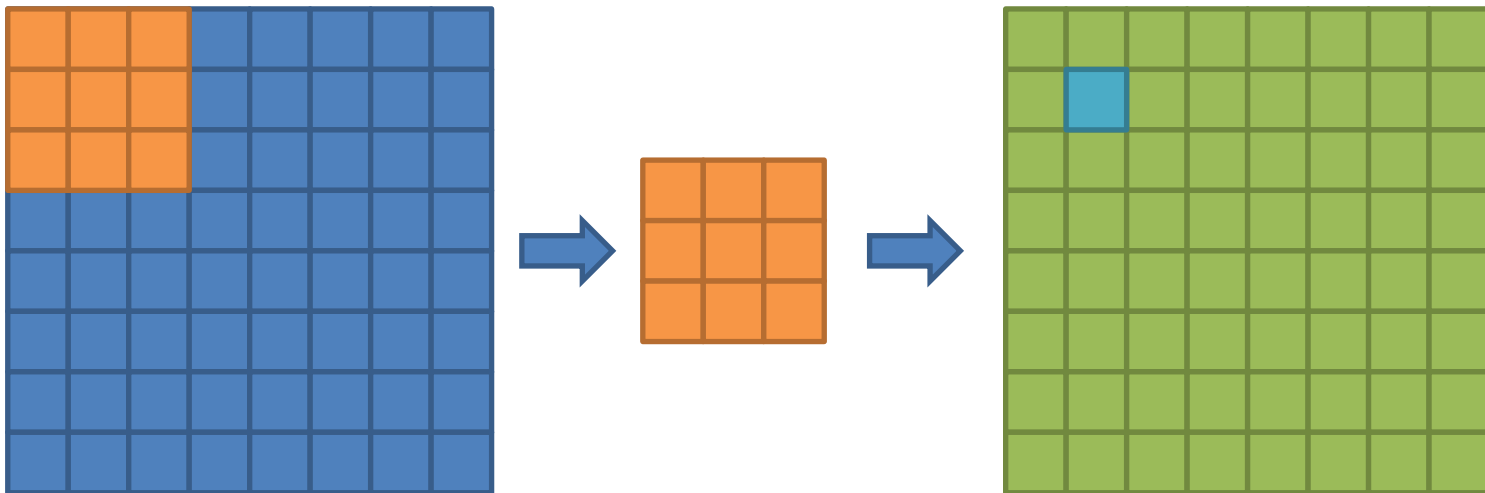
CUDA

- Convolución



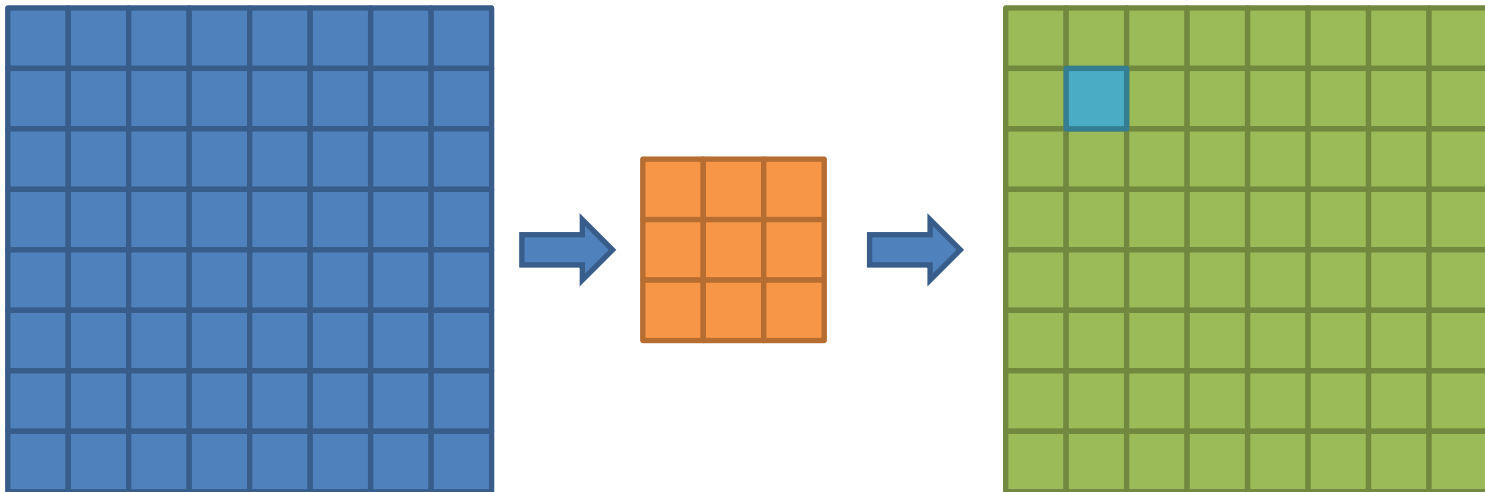
CUDA

- Convolución



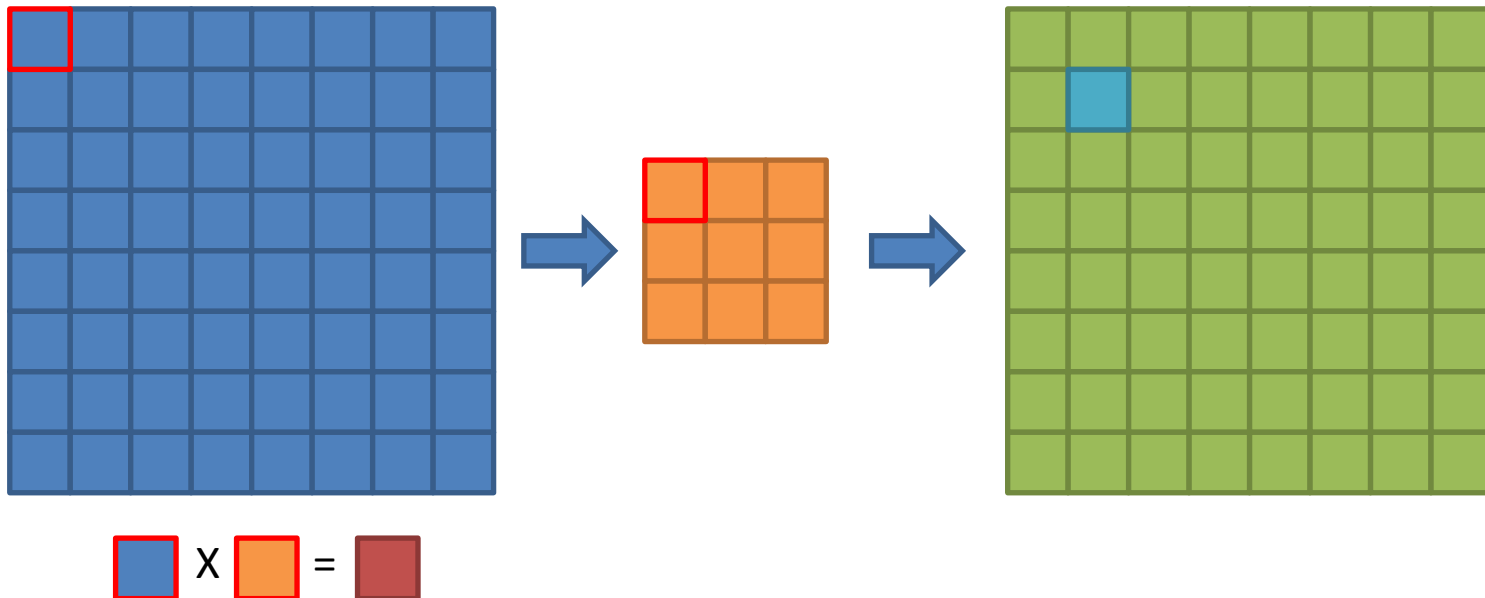
CUDA

- Convolución



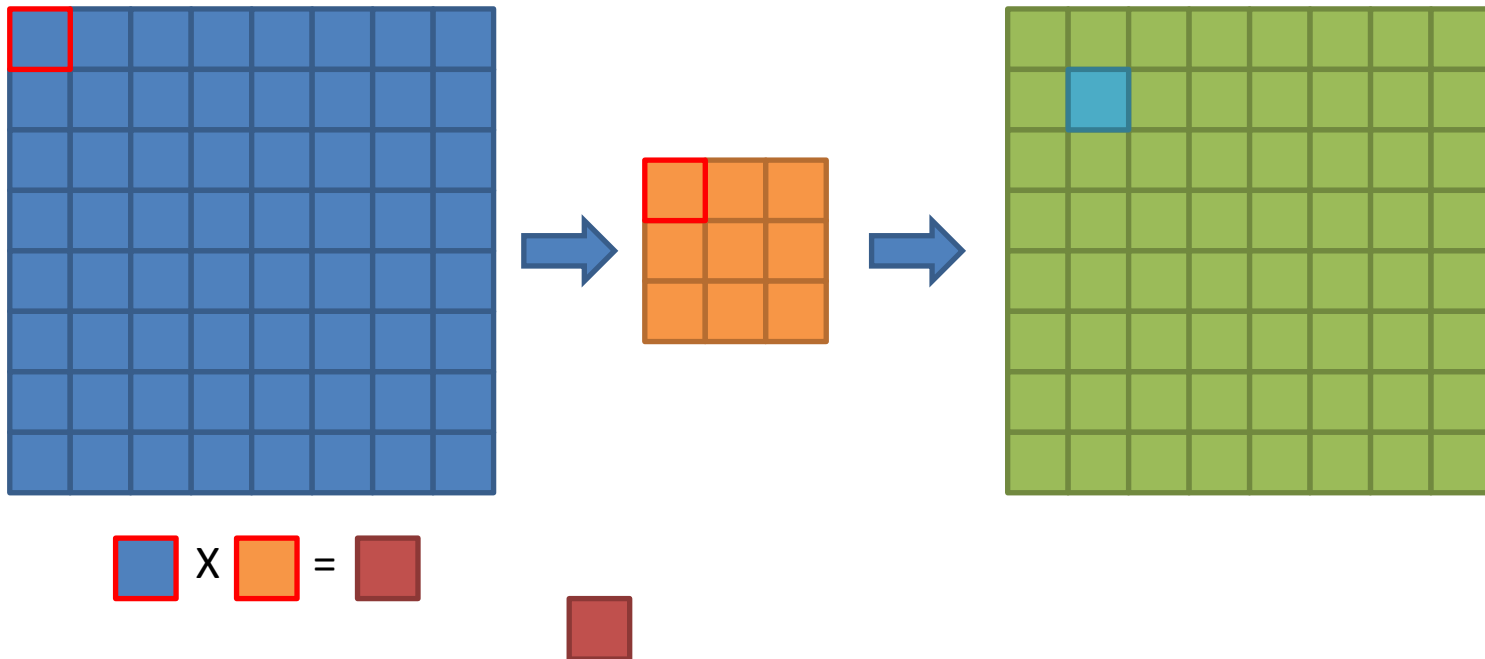
CUDA

- Convolución



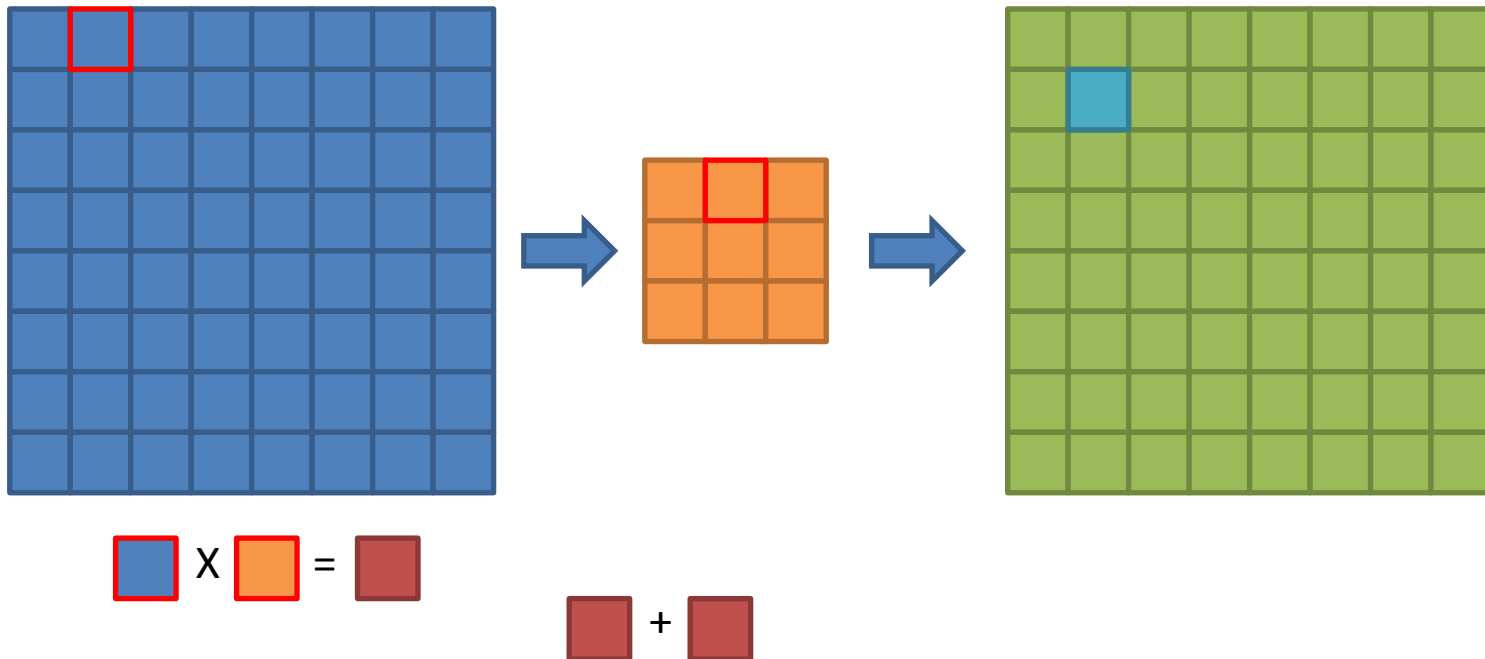
CUDA

- Convolución



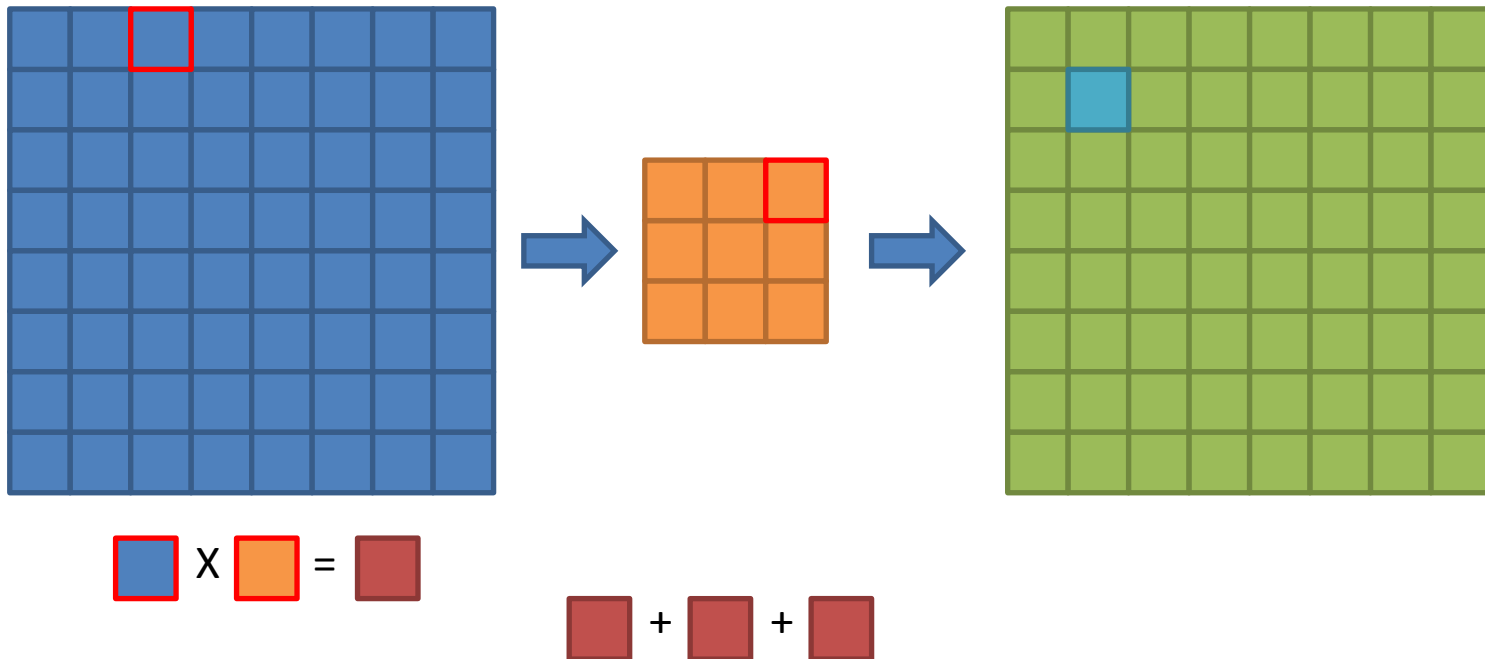
CUDA

- Convolución



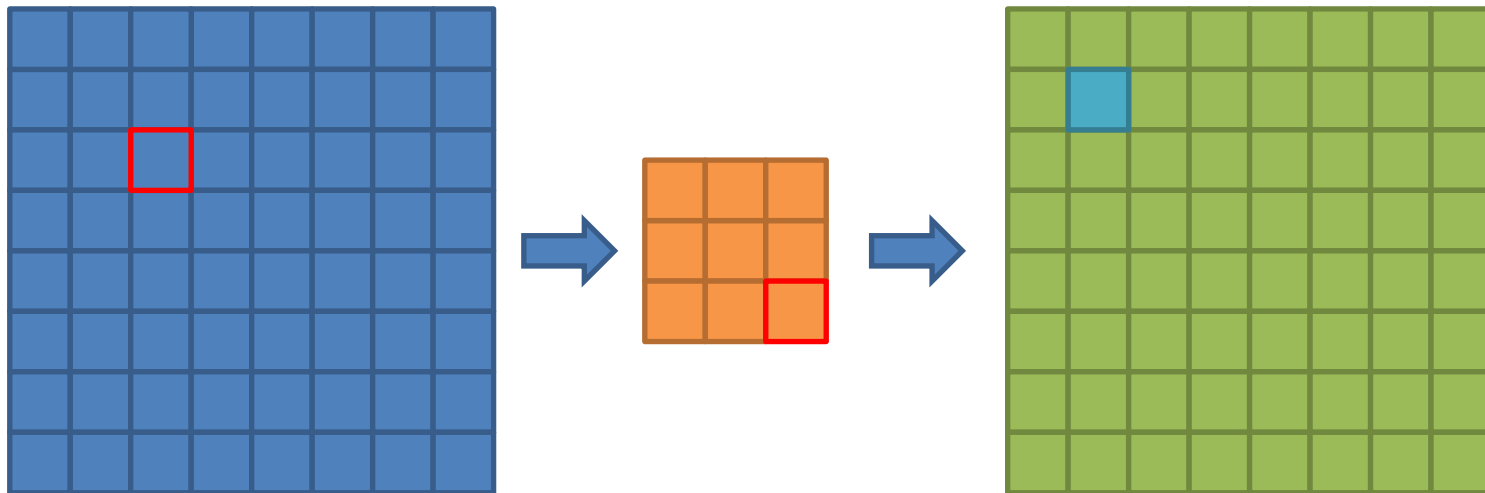
CUDA

- Convolución



CUDA

- Convolución

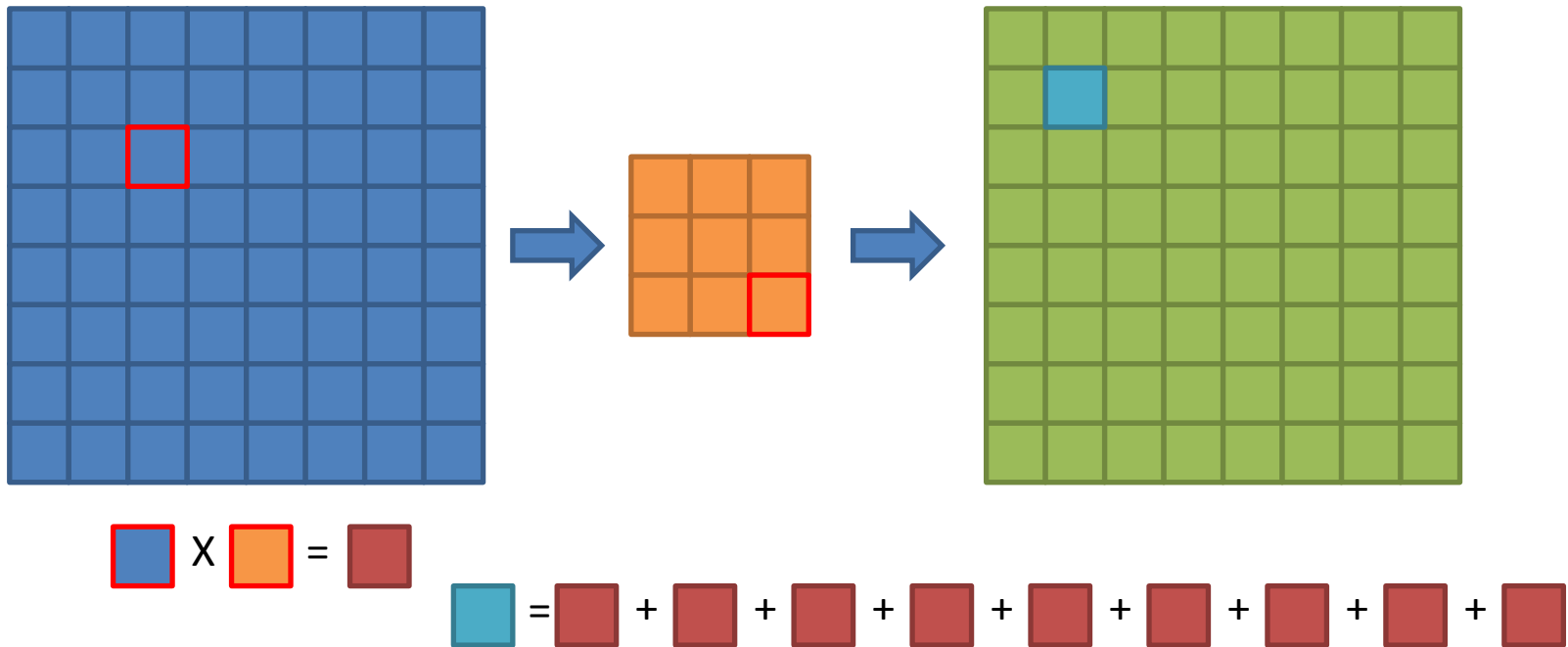


$$\text{blue square} \times \text{orange square} = \text{red square}$$

$$\text{red square} + \text{red square} + \text{red square} + \text{red square} + \text{red square} + \text{red square} + \text{red square} + \text{red square} + \text{red square}$$

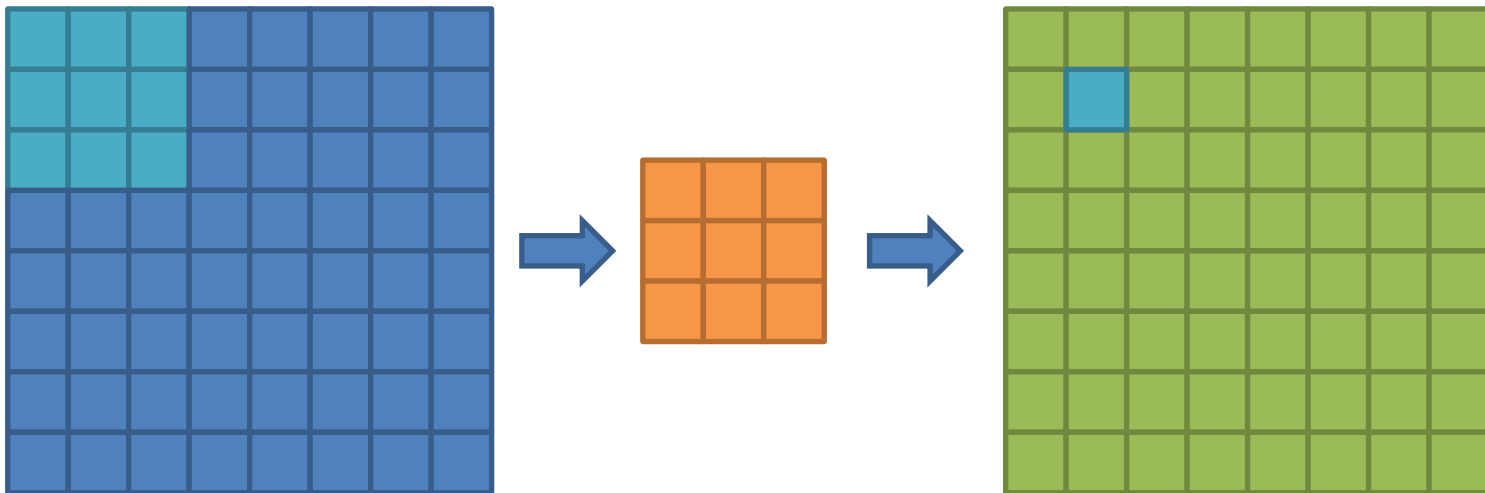
CUDA

- Convolución



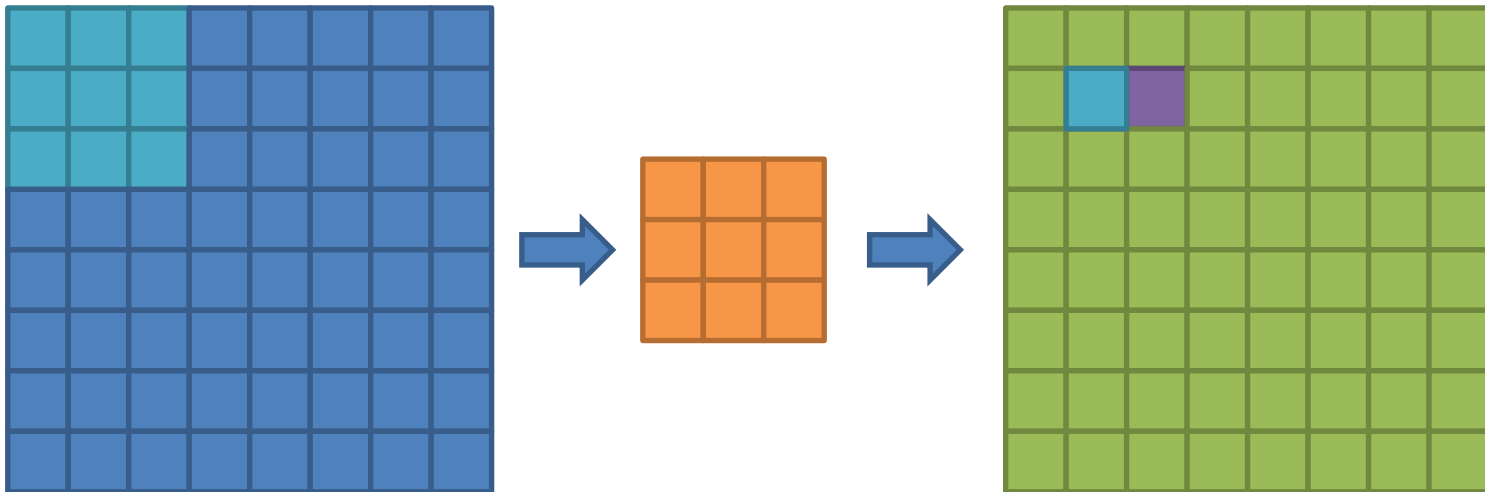
CUDA

- Convolución



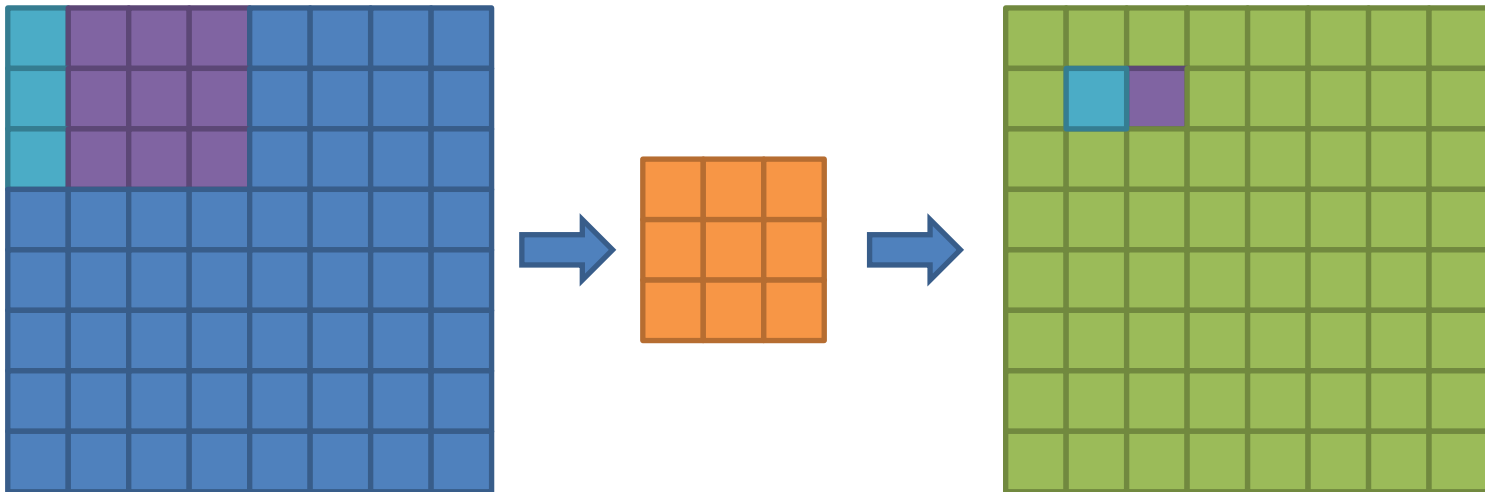
CUDA

- Convolución



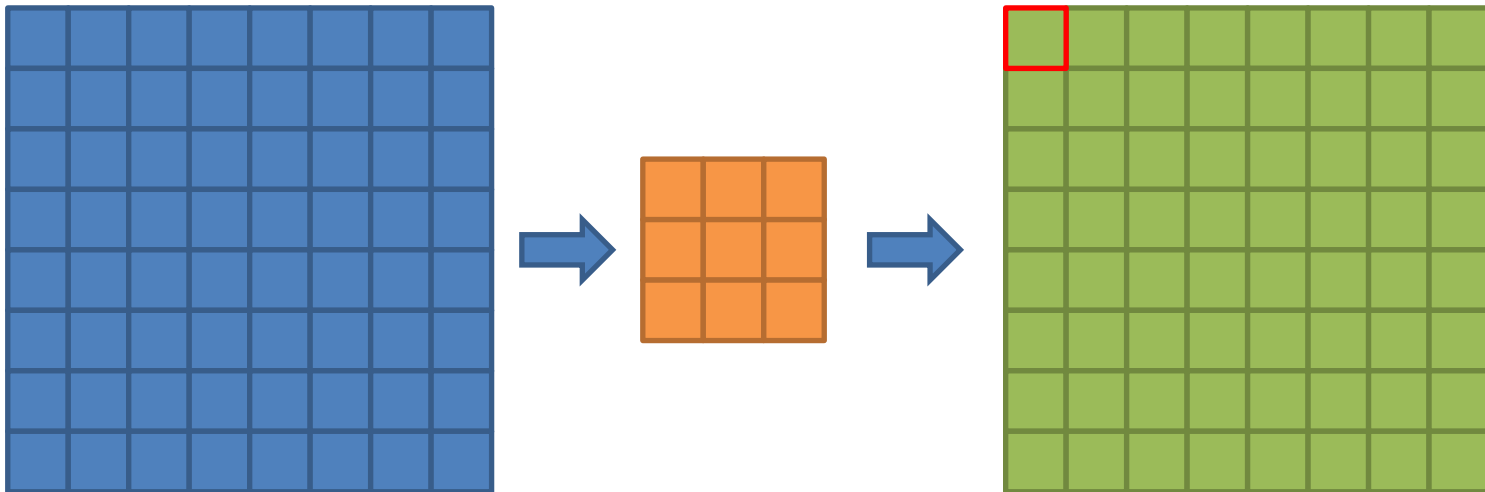
CUDA

- Convolución



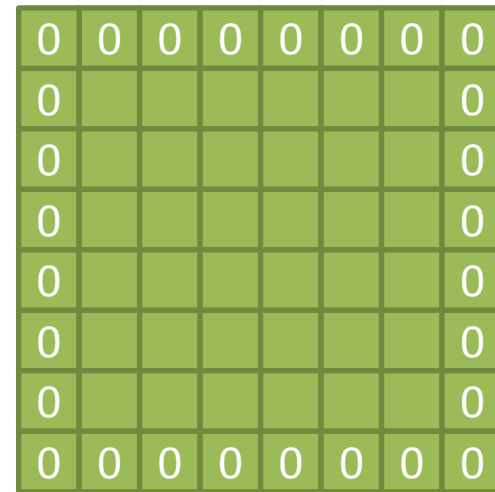
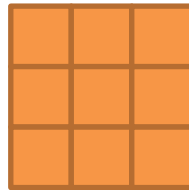
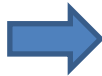
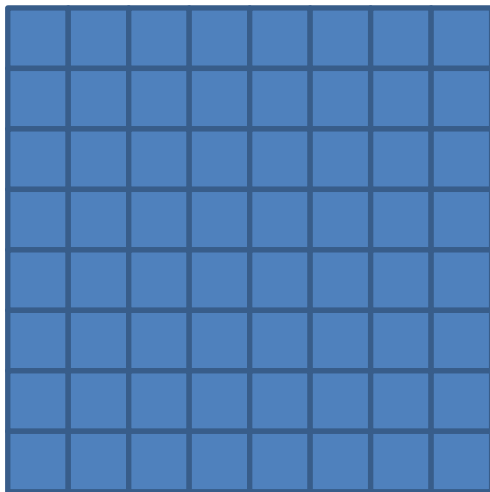
CUDA

- Convolución



CUDA

- Convolución

A 10x10 grid of green squares representing the output feature map. The top and bottom rows are filled with the number 0, while the middle 8 rows are empty, illustrating the result of a 3x3 convolution with zero padding.| | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | | | | | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

CUDA

- Convolución
 - Filtros:

0	0	0
0	1	0
0	0	0

CUDA

- Convolución
 - Filtros:
 - Copia

0	0	0
0	1	0
0	0	0

CUDA

- Convolución

- Filtros:

- Copia

0	0	0
0	1	0
0	0	0

1	0	0
0	0	0
0	0	0

CUDA

- Convolución

- Filtros:

- Copia

0	0	0
0	1	0
0	0	0

- Desplazamiento

1	0	0
0	0	0
0	0	0

CUDA

- Convolución

- Filtros:

- Copia

0	0	0
0	1	0
0	0	0

- Desplazamiento

1	0	0
0	0	0
0	0	0

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

CUDA

- Convolución

- Filtros:

- Copia

0	0	0
0	1	0
0	0	0

- Desplazamiento

1	0	0
0	0	0
0	0	0

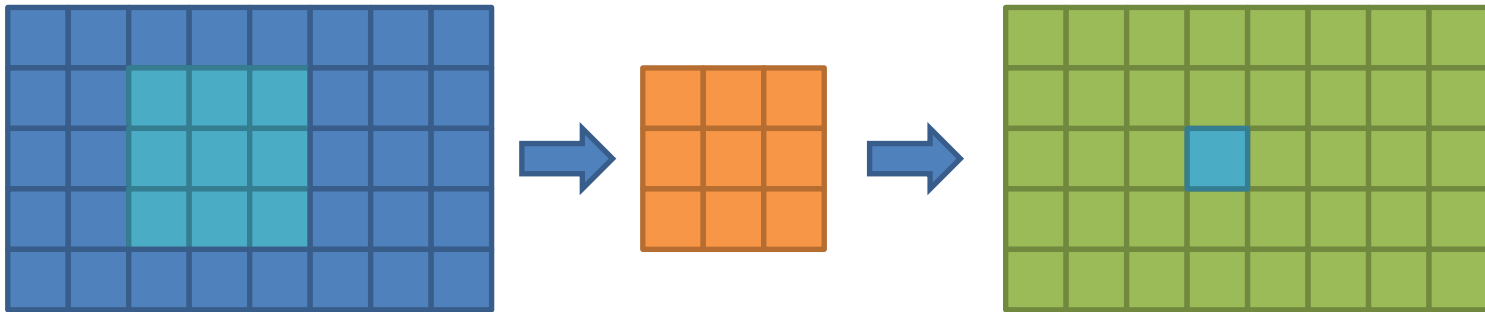
- Bordes

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

CUDA

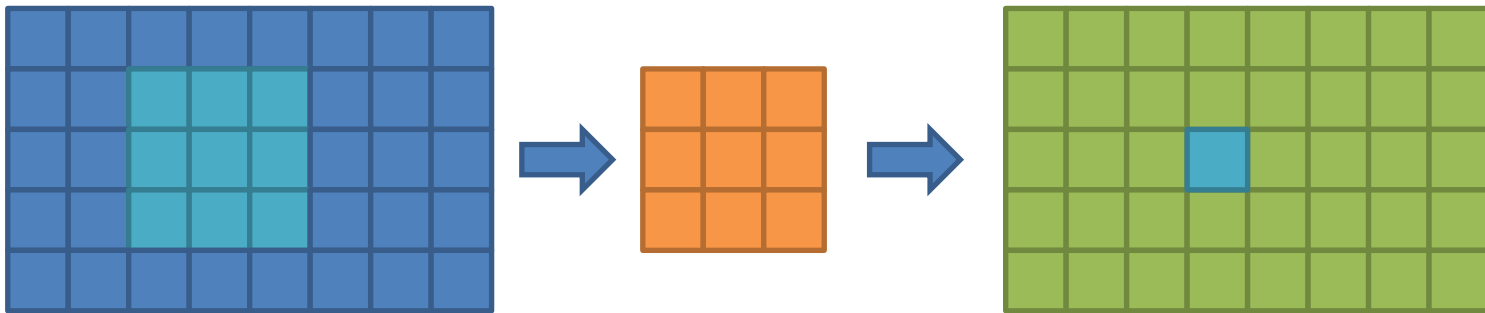
- Convolución



`dst[?] = ?`

CUDA

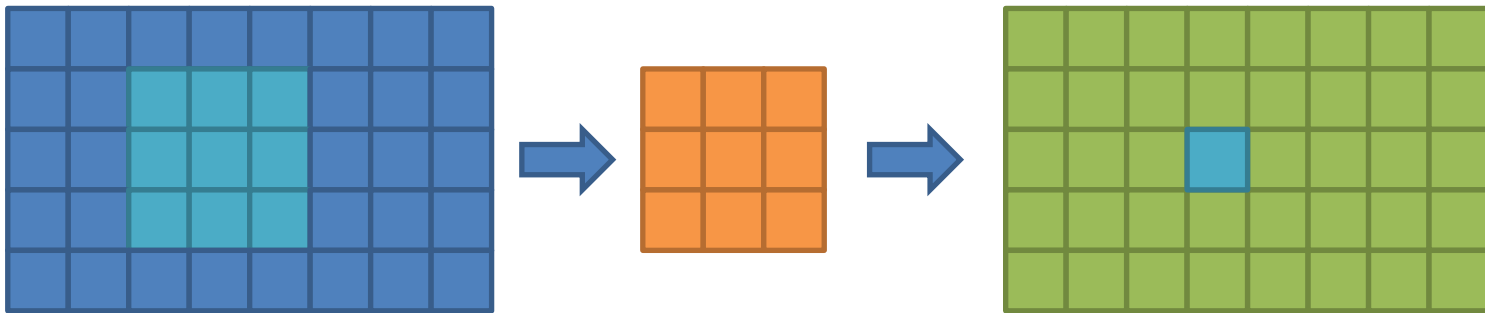
- Convolución



$\text{dst}[y * w + x] = ?$

CUDA

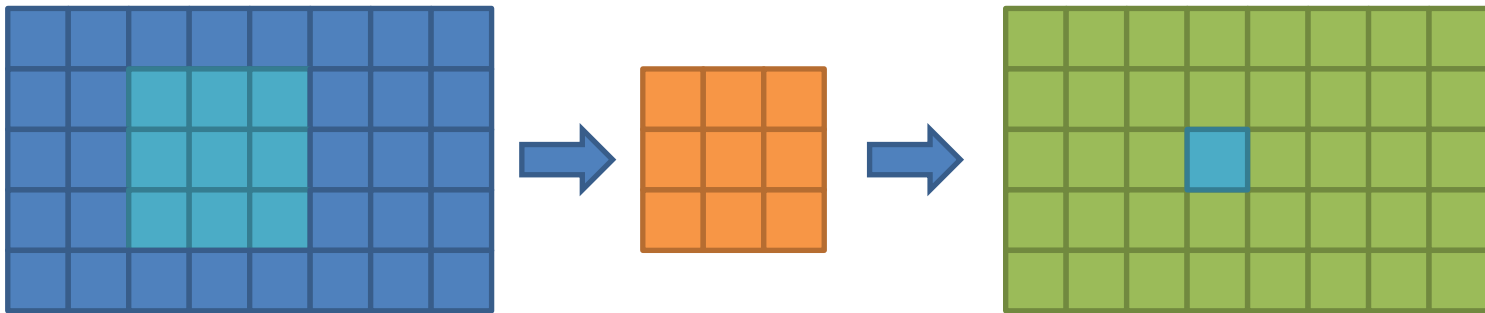
- Convolución



```
dst[y * w + x] = src[?] * f[?];
```


CUDA

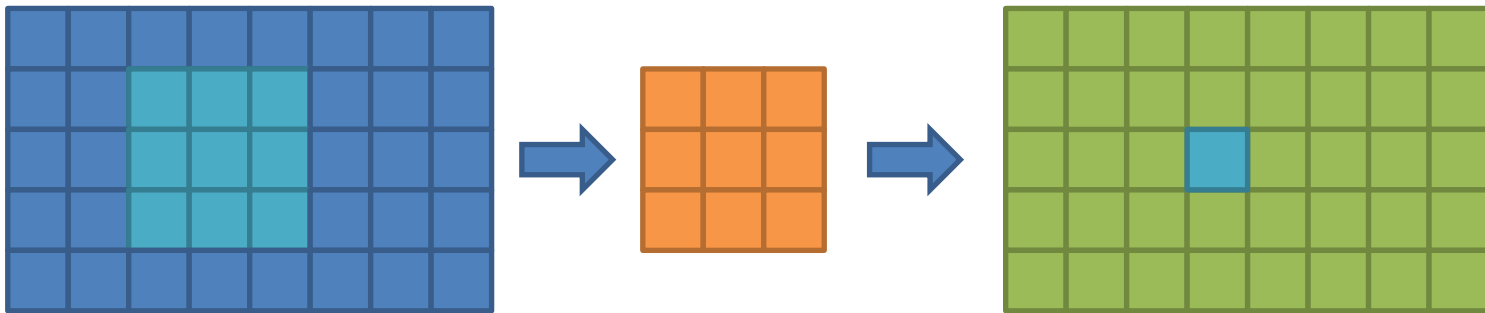
- Convolución



```
dst[y * w + x] += src[?] * f[?];
```

CUDA

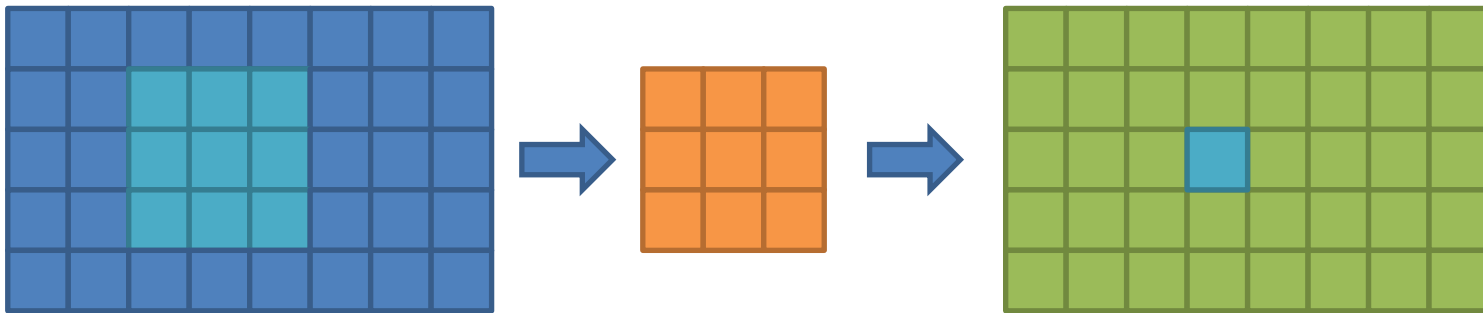
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    dst[y * w + x] += src[?] * f[?];  
}
```

CUDA

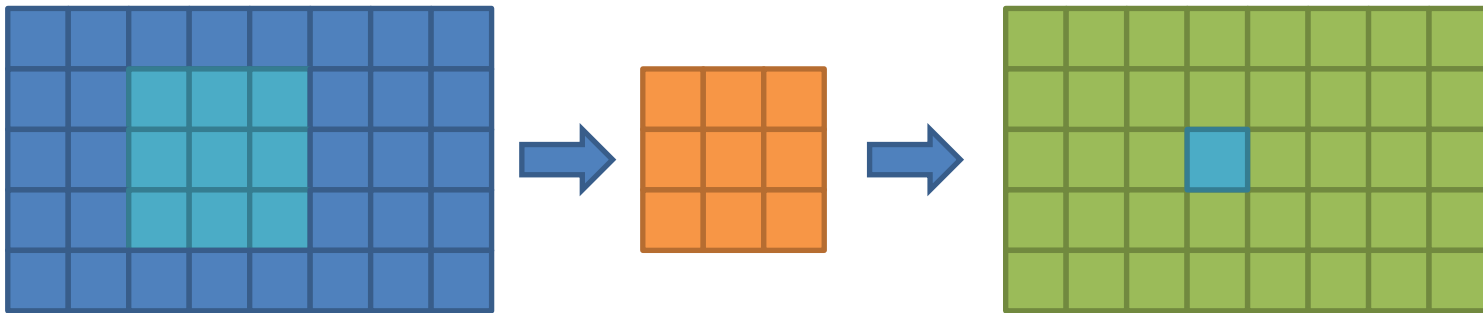
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[?] * f[?];  
    }  
}
```

CUDA

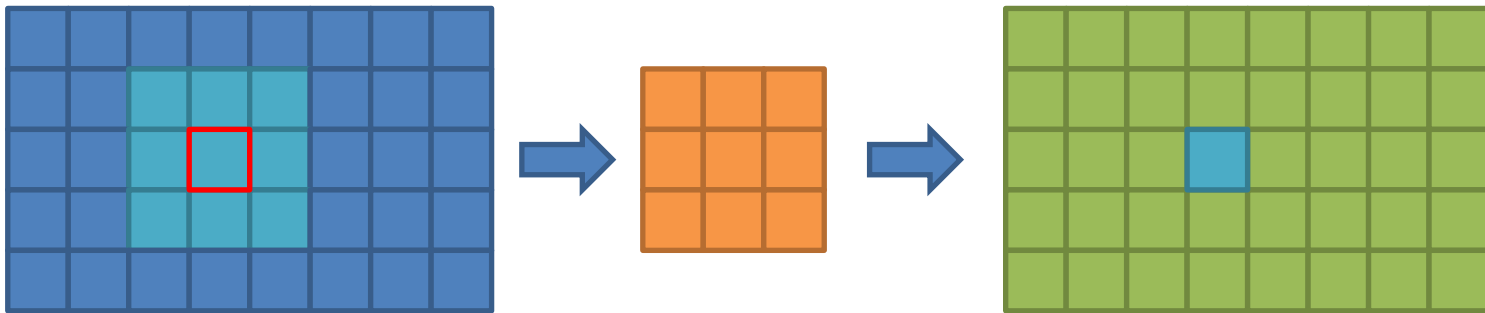
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[] * f[k * fsize + l];  
    }  
}
```

CUDA

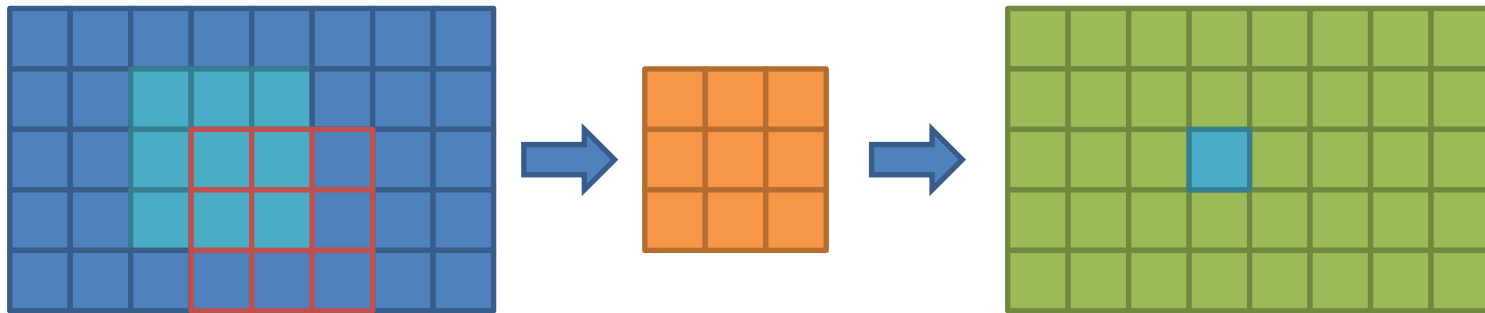
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[?] * f[k * fsize + l];  
    }  
}
```

CUDA

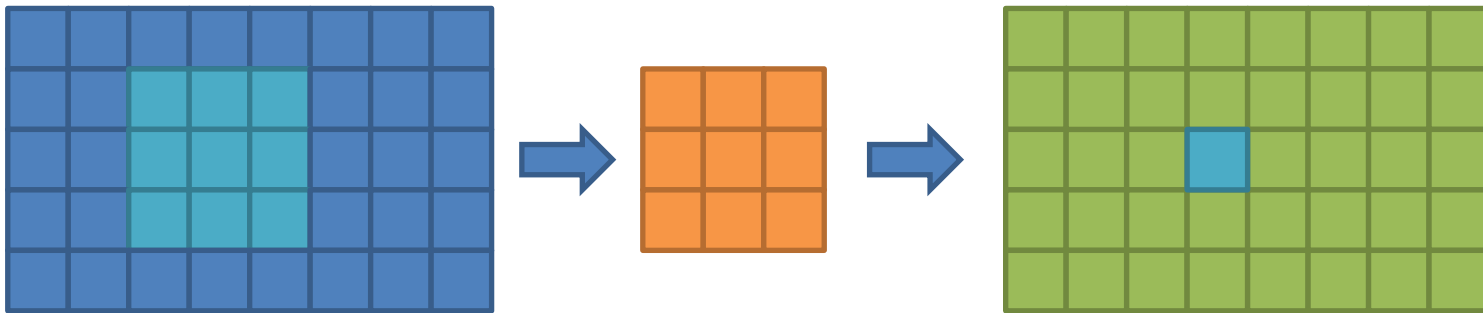
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k) * w + (x+l)] * f[k * fsize + l];  
    }  
}
```

CUDA

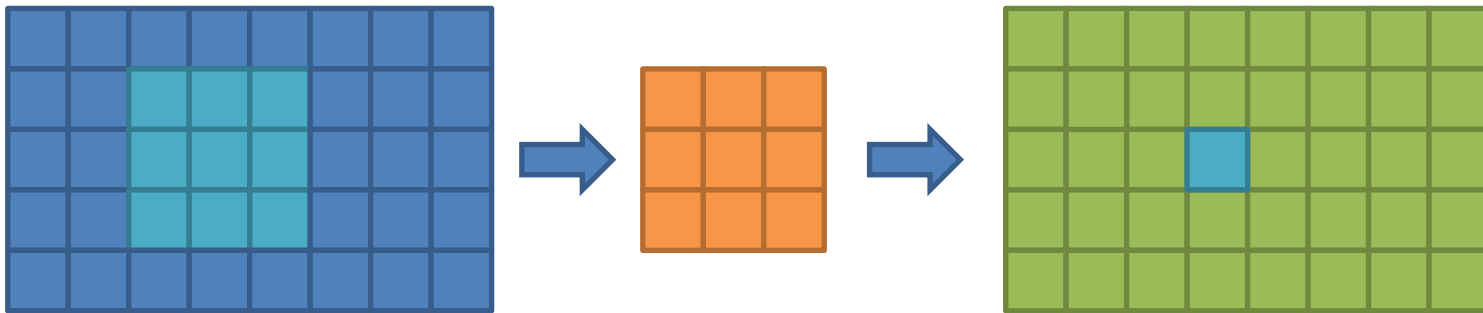
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k-1) * w + (x+l-1)] * f[k * fsize + l];  
    }  
}
```

CUDA

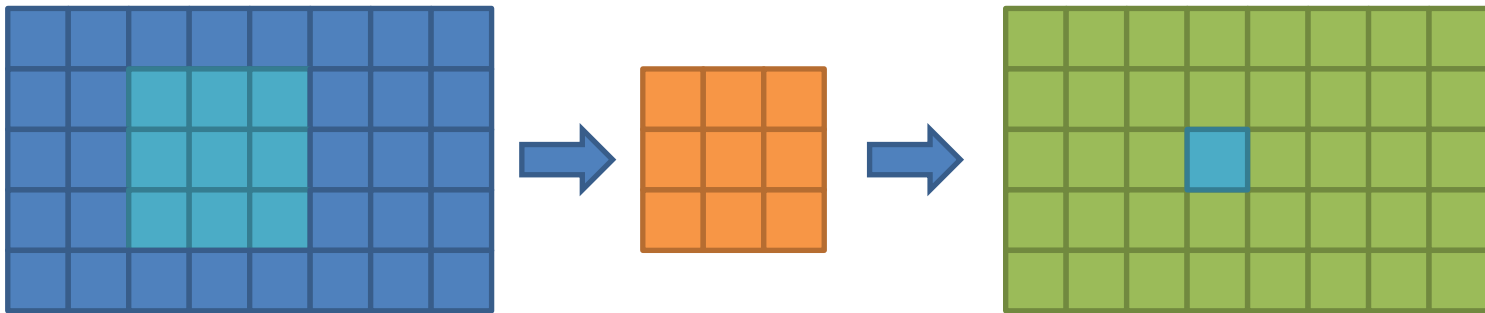
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k-1) * w + (x+l-1)] * f[k * fsize + l];  
    }  
}
```


CUDA

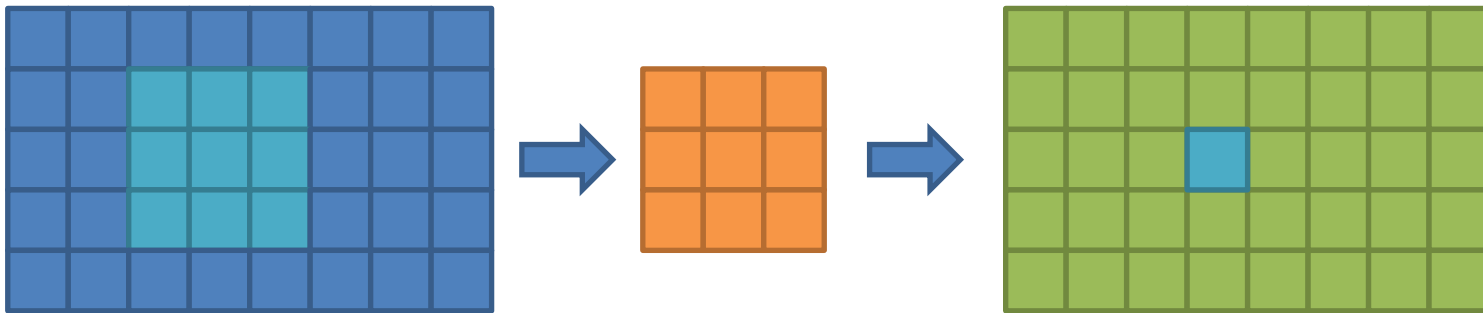
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k-1) * w + (x+l-1)] * f[k * fsize + l];  
    }  
}
```

CUDA

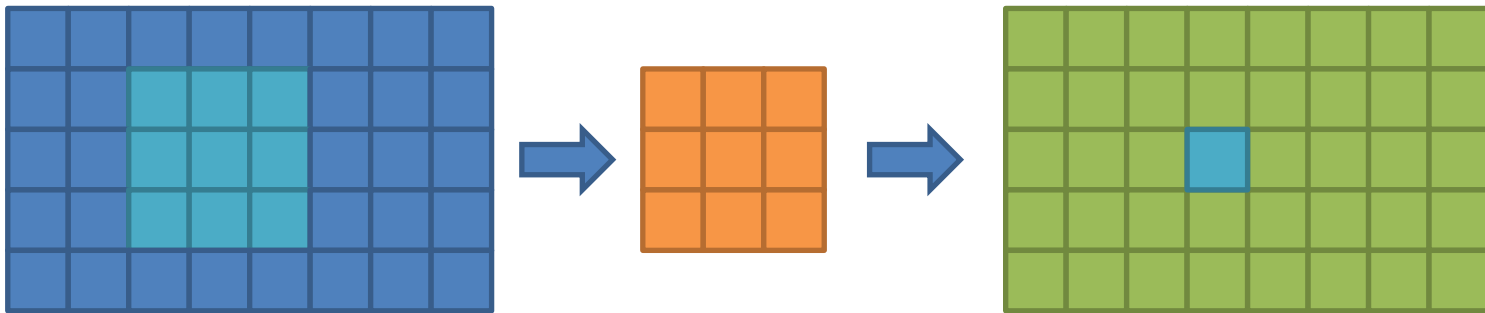
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k-1) * w + (x+l-1)] * f[k * fsize + l];  
    }  
}
```

CUDA

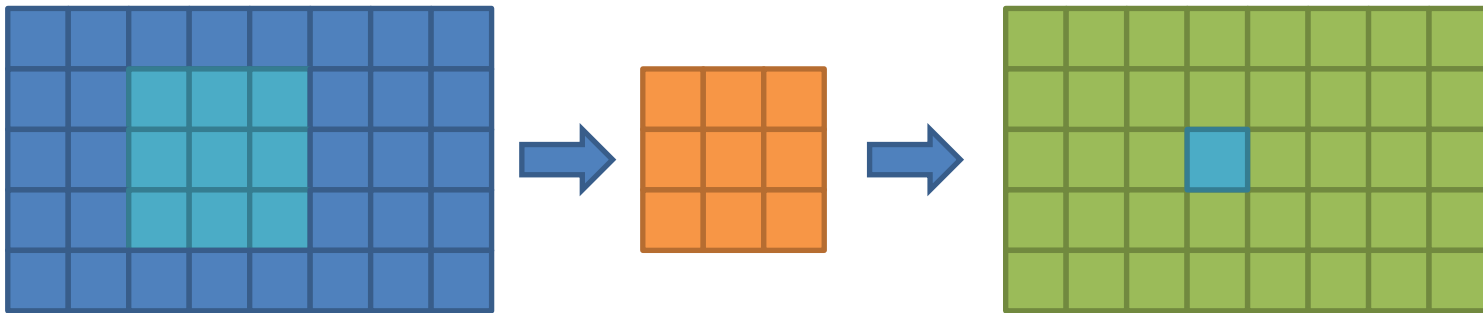
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k-1) * w + (x+l-1)] * f[k * fsize + l];  
    }  
}
```

CUDA

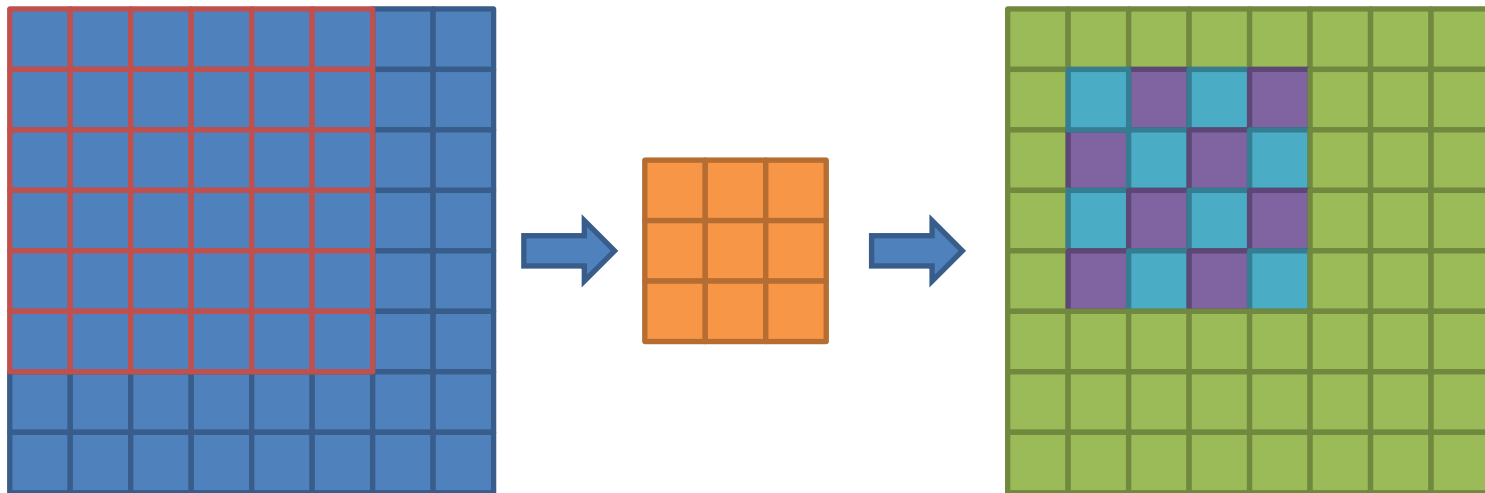
- Convolución



```
for(int k = 0; k < fsize; ++k) {  
    for(int l = 0; l < fsize; ++l) {  
        dst[y * w + x] += src[(y+k-1) * w + (x+l-1)] * f[k * fsize + l];  
    }  
}
```

CUDA

- Convolución



16 hilos X 9 píxeles = 144 lecturas de memoria global

Se leen $6 \times 6 = 36$ píxeles distintos

CUDA

- Convolución
 - Un kernel que haga uso de memoria compartida tiene, la menos, las siguientes partes:
 - Calculo de índices
 - Copia a memoria compartida
 - Sincronización (`__syncthreads();`)
 - Uso de la memoria compartida

CUDA

- Convolución

- Memoria compartida

- Una memoria más que la memoria principal y sin sus limitaciones.
 - Compartida por los threads de un bloque.
 - Tamaño limitado

- Uso

- Tamaño estático:

```
__shared__ int mem[NTHREAD][NTHREAD];
```

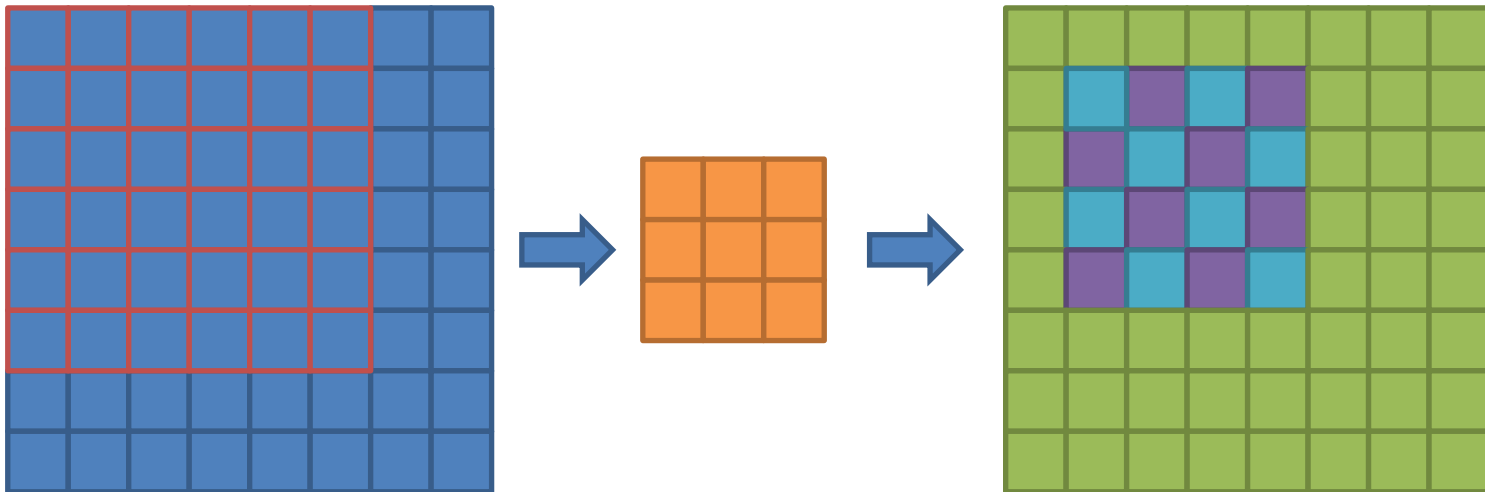
- Tamaño dinámico:

```
extern __shared__ int mem[];
```

```
kernelCall<<< blocks, thread, sharedsize>>> (...);
```

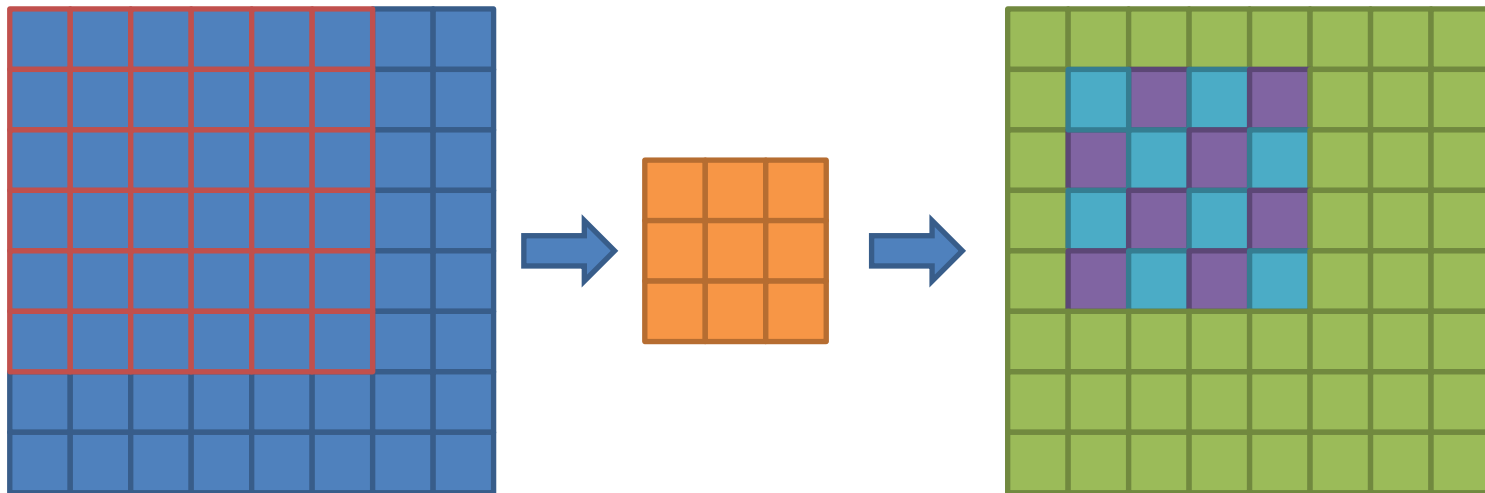
CUDA

- Convolución



CUDA

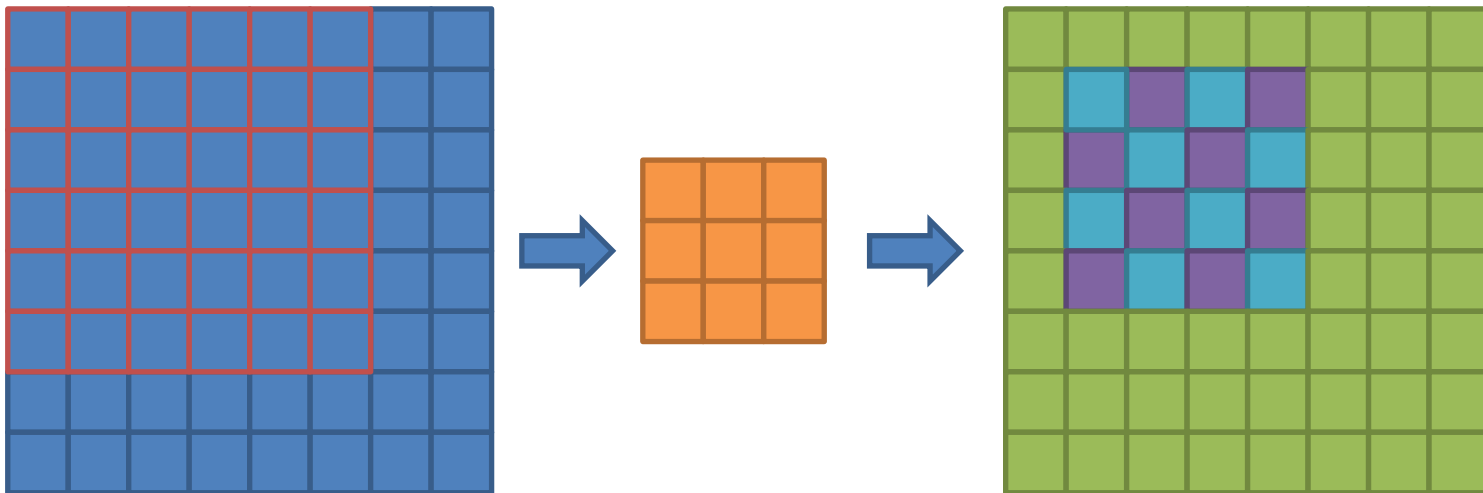
- Convolución



¿Cómo 16 hilos copian 36 píxeles?

CUDA

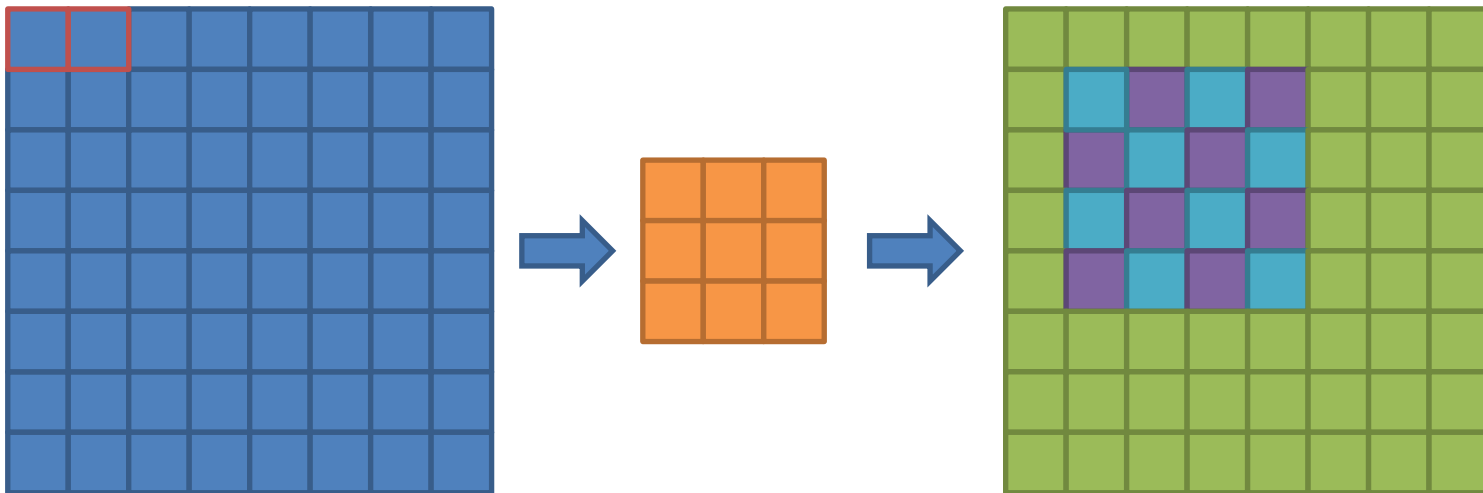
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

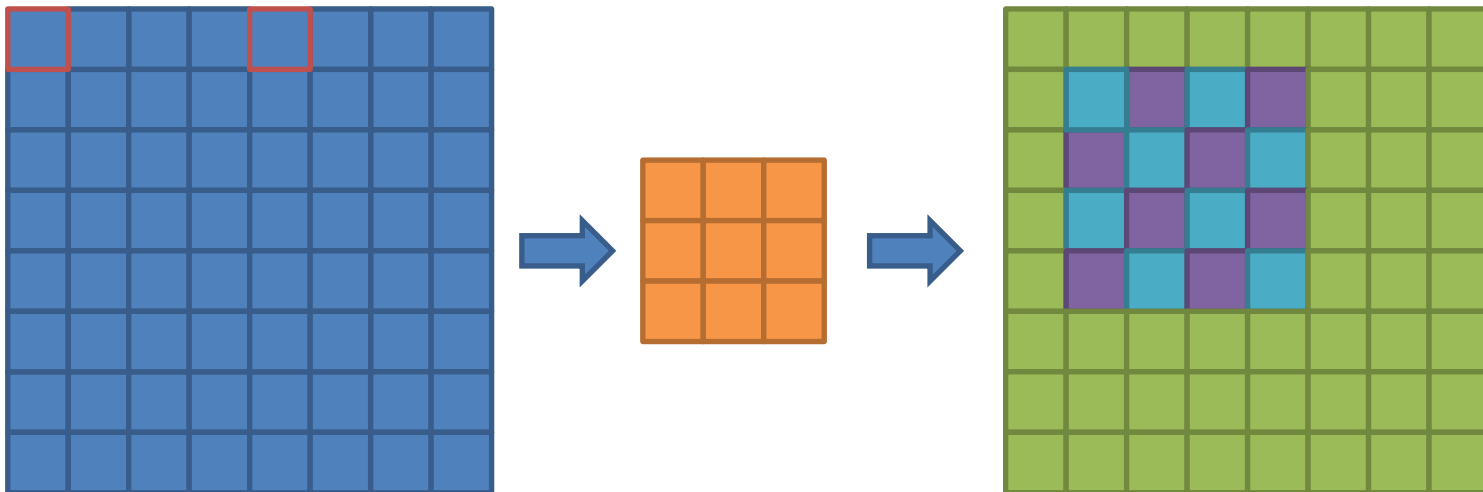
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

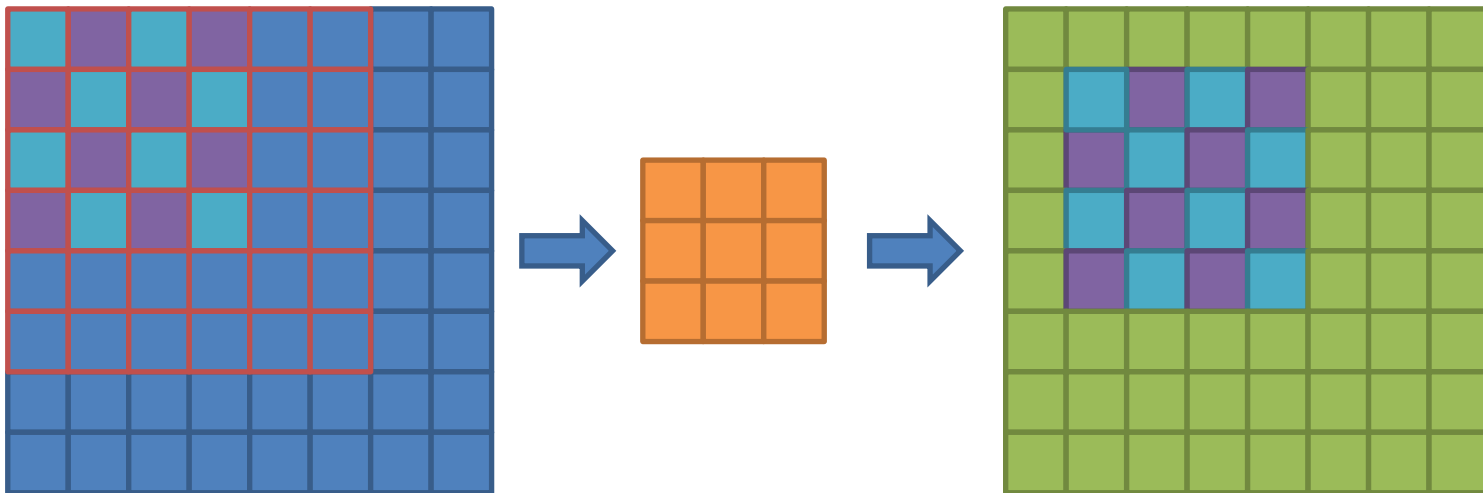
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

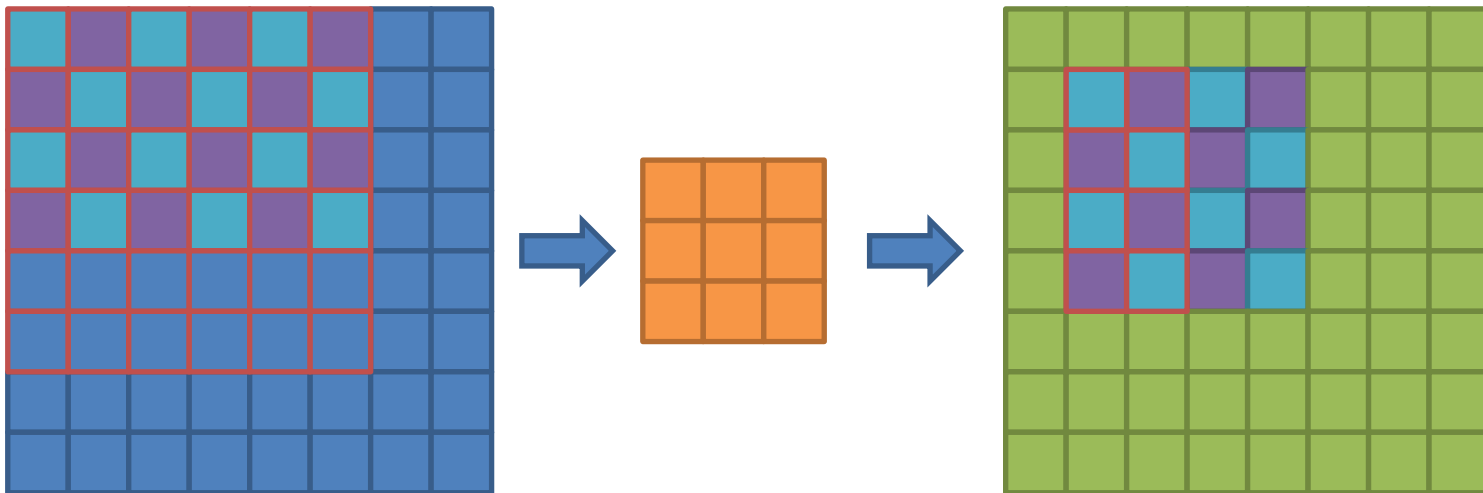
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

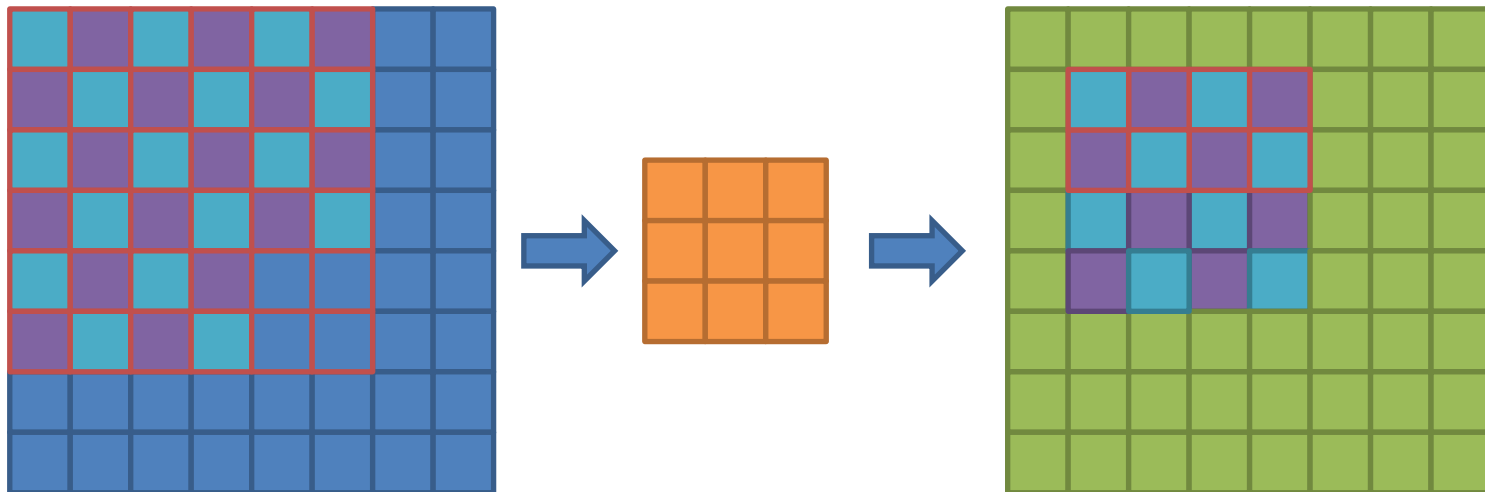
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

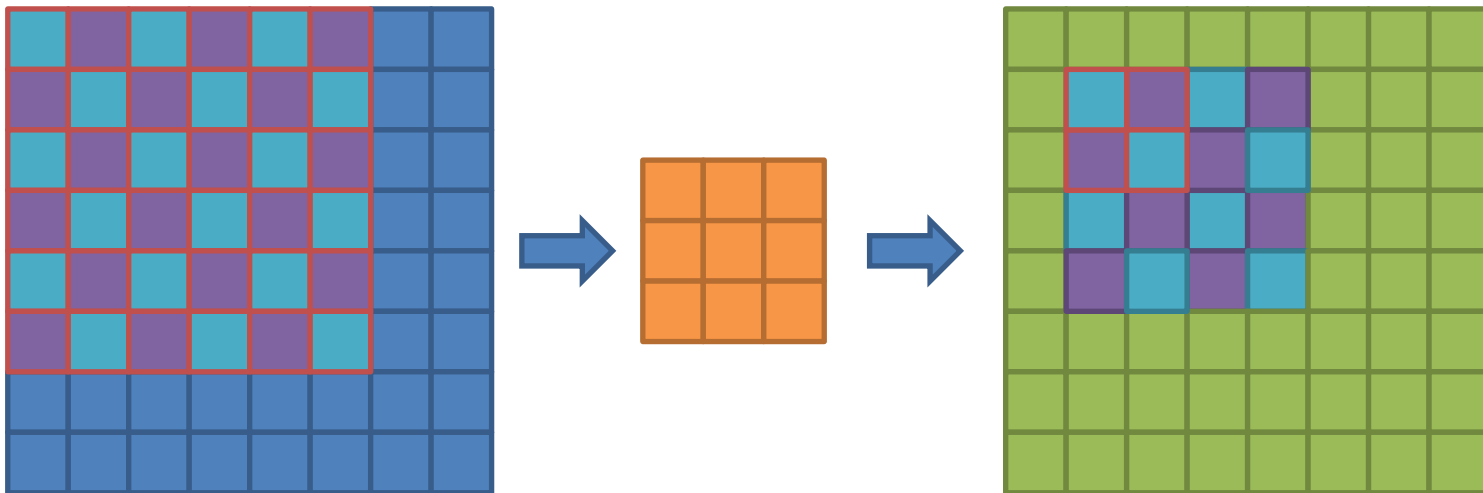
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

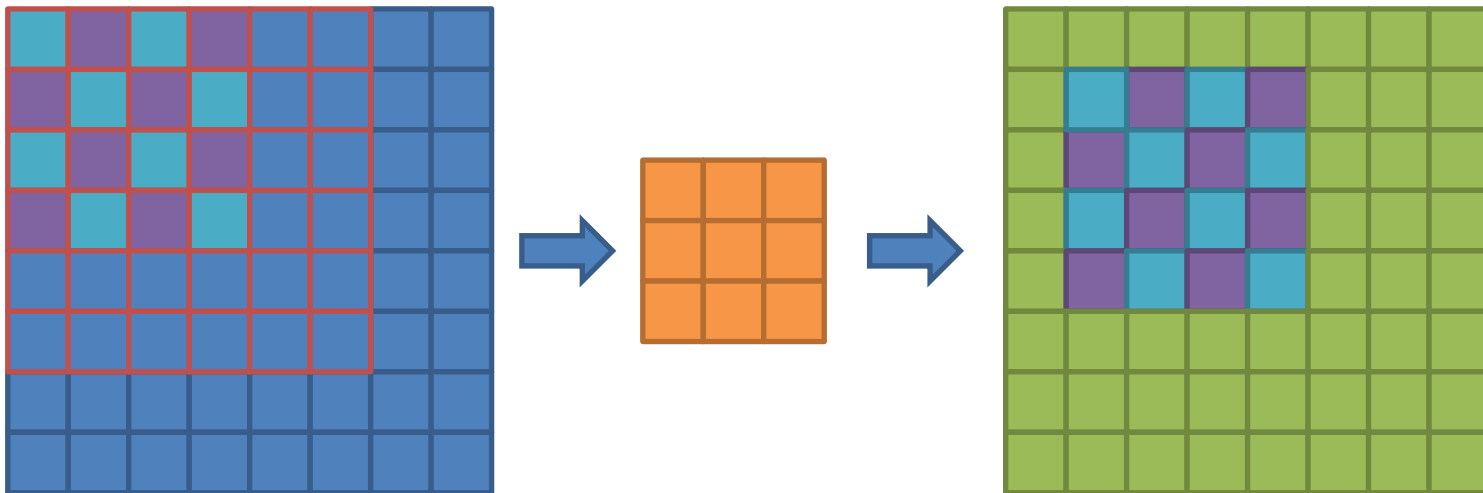
- Convolución



¿Cómo 16 hilos copian 36 píxeles?
Algunos hilos copian más de un pixel.

CUDA

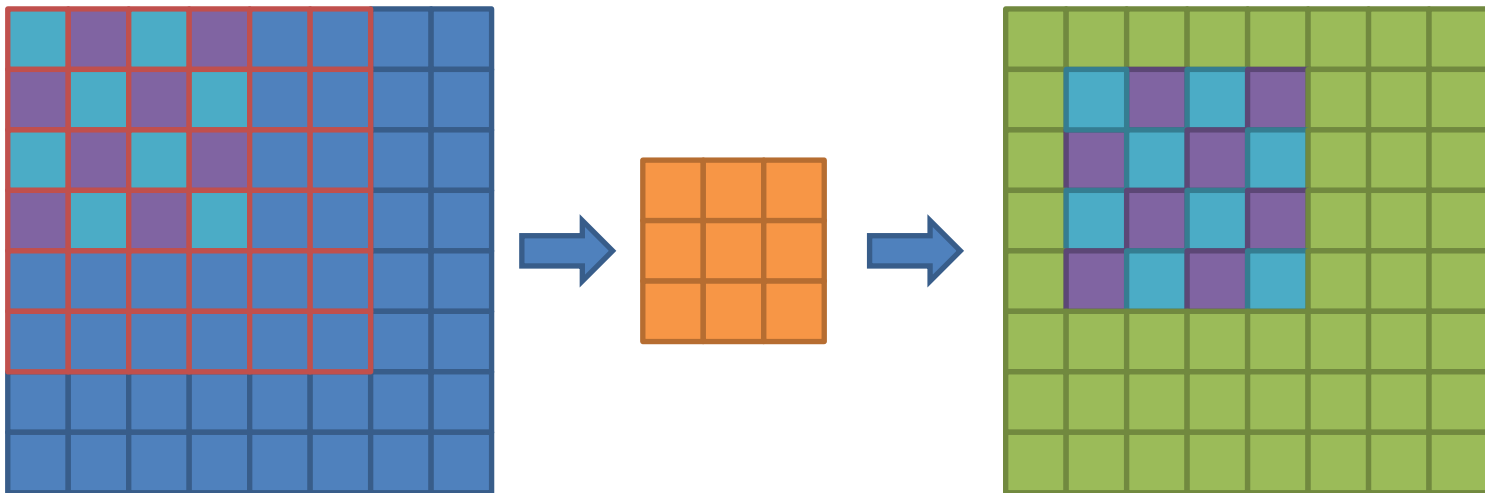
- Convolución



`smem[?] = ?`

CUDA

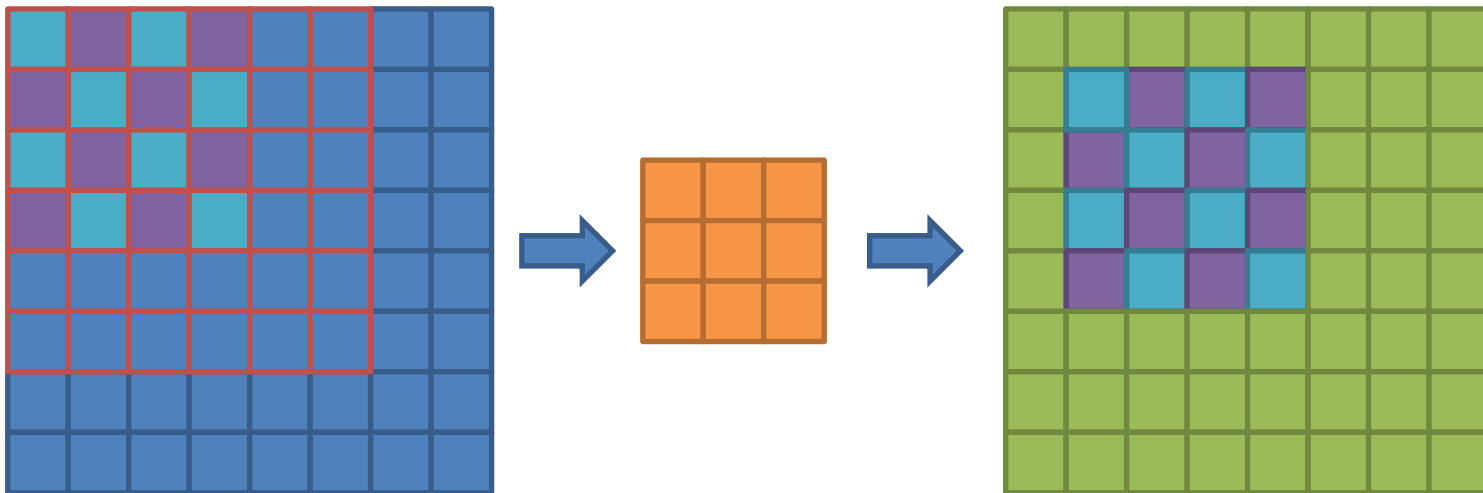
- Convolución



`smem[threadIdx.y * "w" + threadIdx.x] = ?`

CUDA

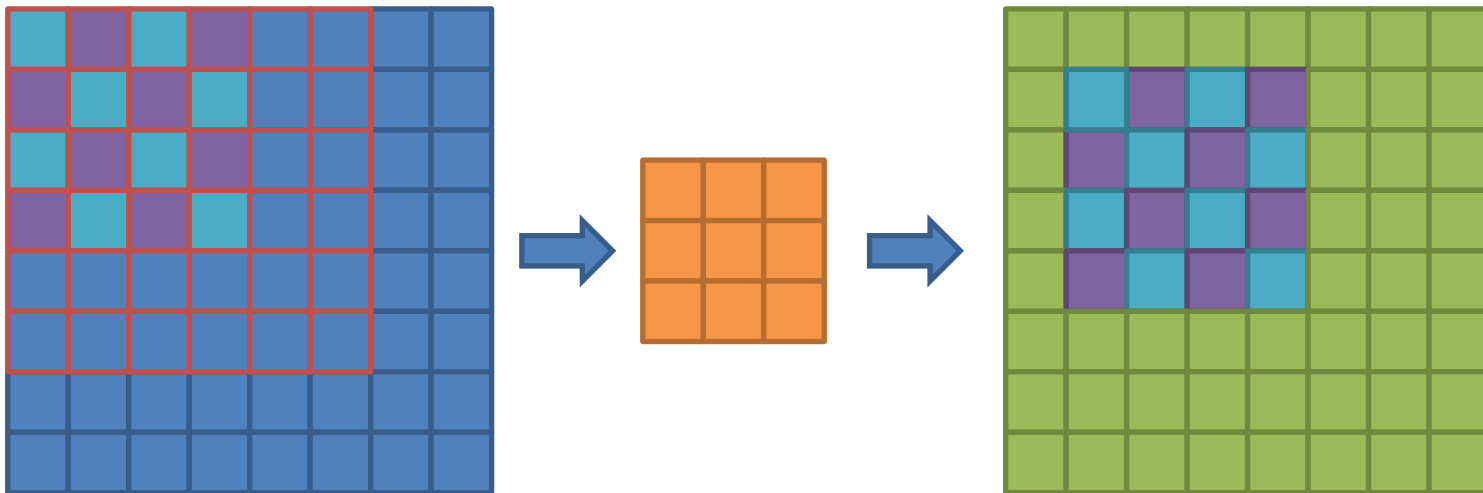
- Convolución



`smem[threadIdx.y * 18 + threadIdx.x] = ?`

CUDA

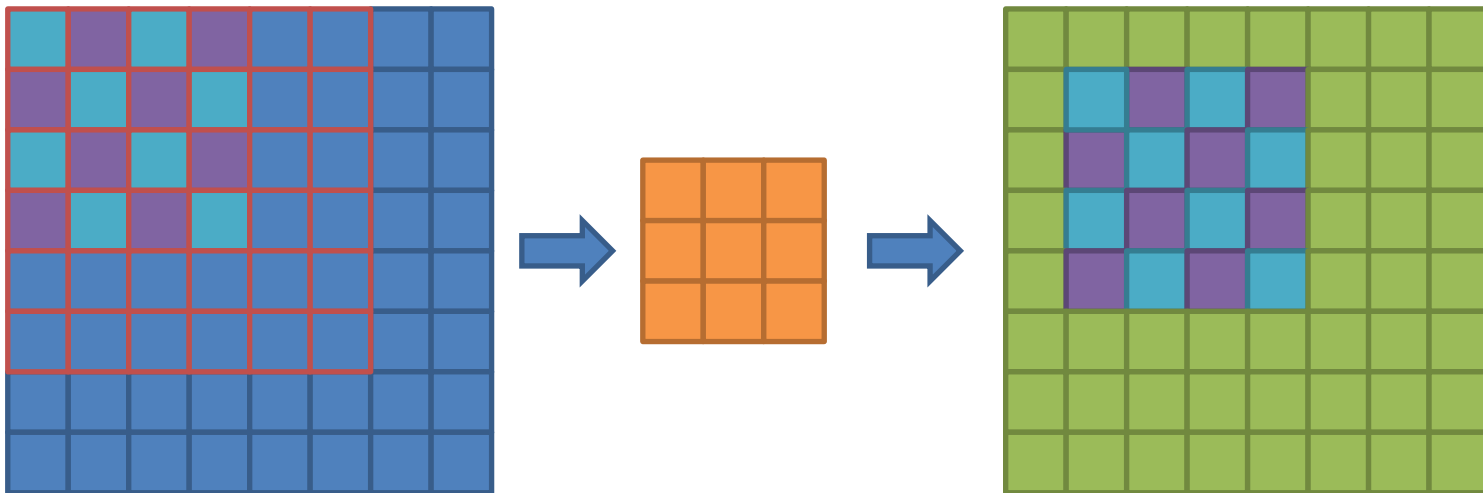
- Convolución



`smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x] = ?`

CUDA

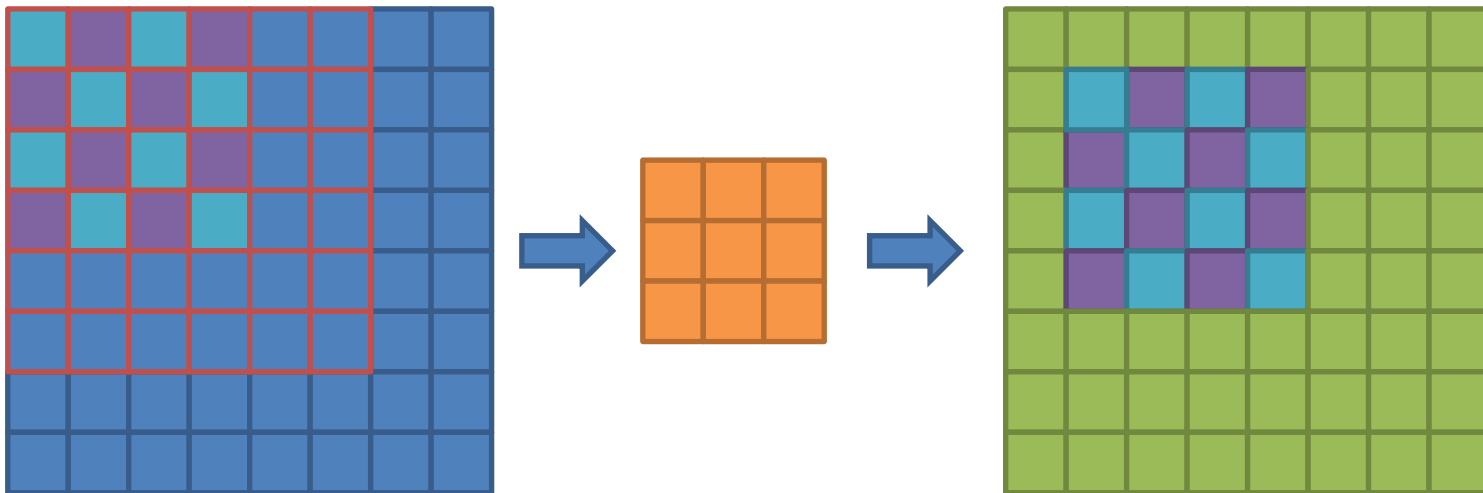
- Convolución



```
smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x] =  
    src[?];
```

CUDA

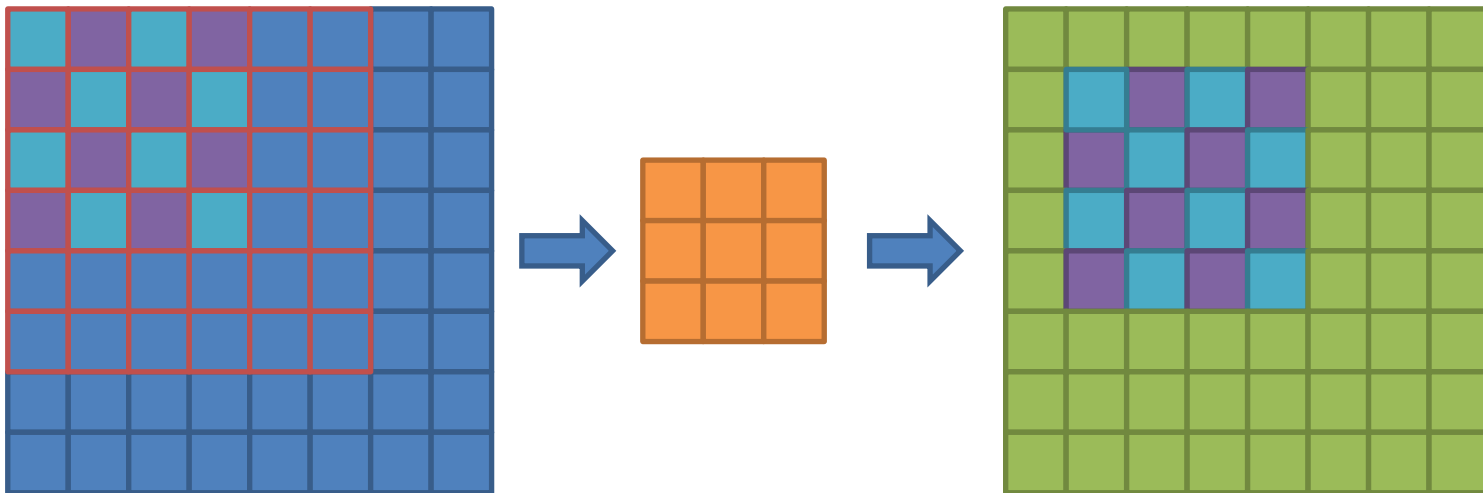
- Convolución



```
smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x] =  
    src[idy * cols + idx];
```

CUDA

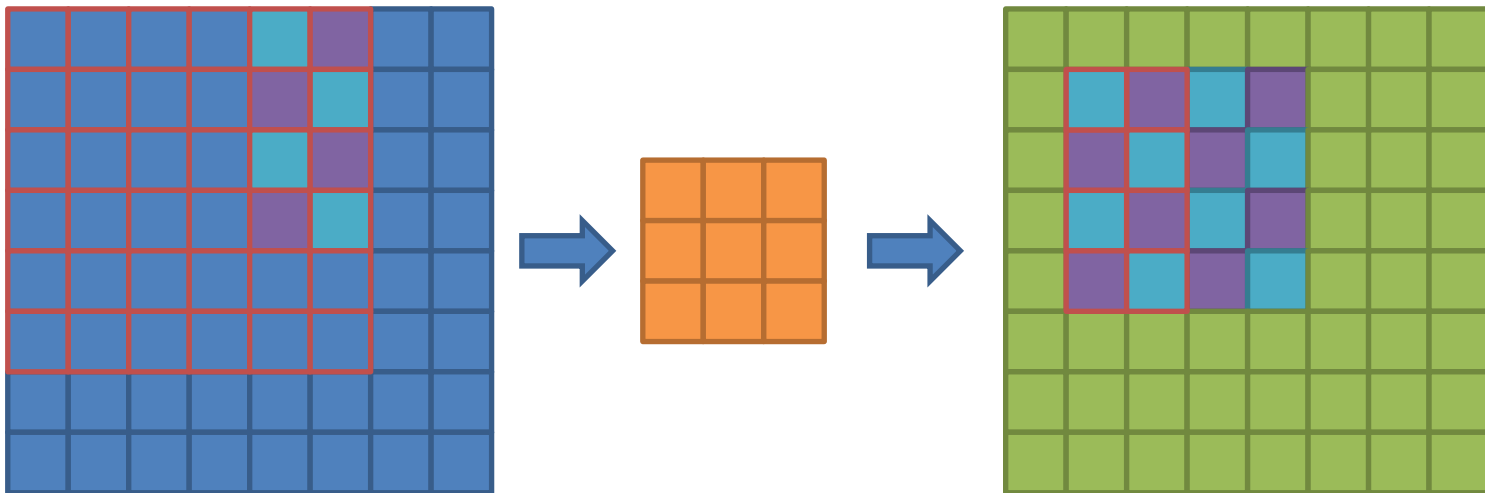
- Convolución



```
smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x] =  
    src[(idy - 1) * cols + (idx - 1)];
```

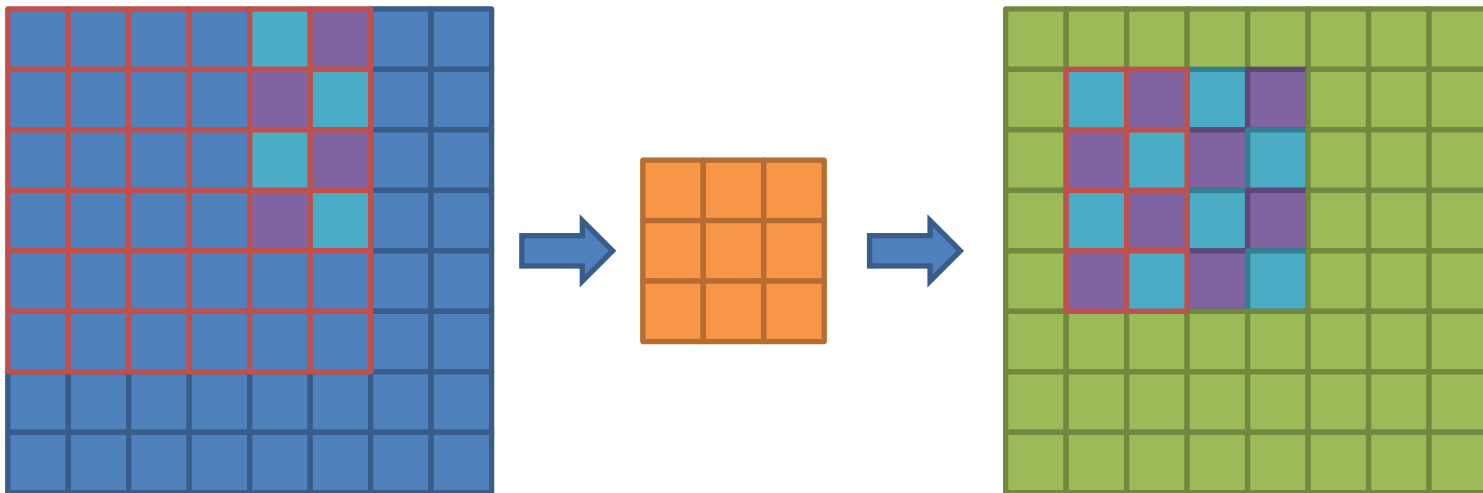
CUDA

- Convolución



CUDA

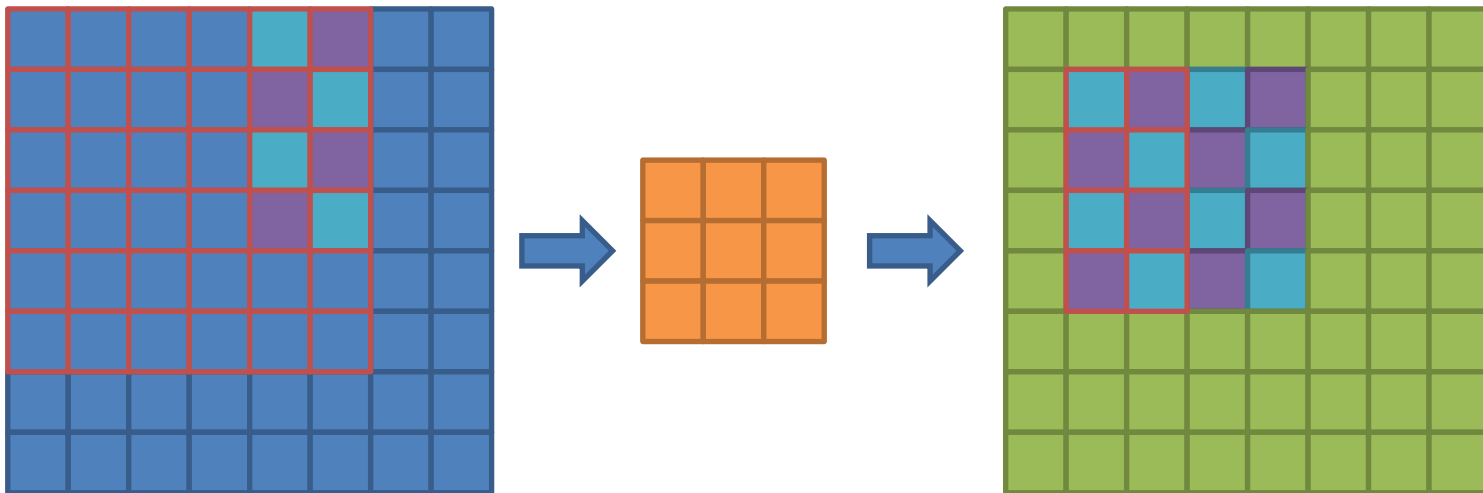
- Convolución



```
if(threadIdx.x < 2)
    smem[?] =
        src[?];
```

CUDA

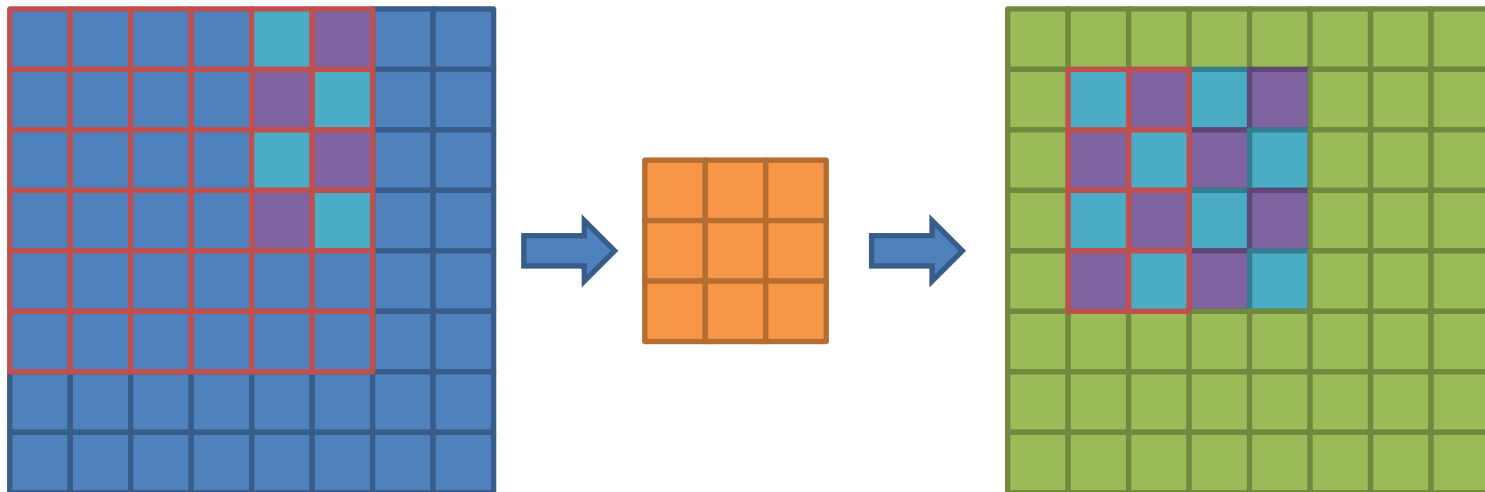
- Convolución



```
if(threadIdx.x < 2)
    smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x] =
        src[?];
```

CUDA

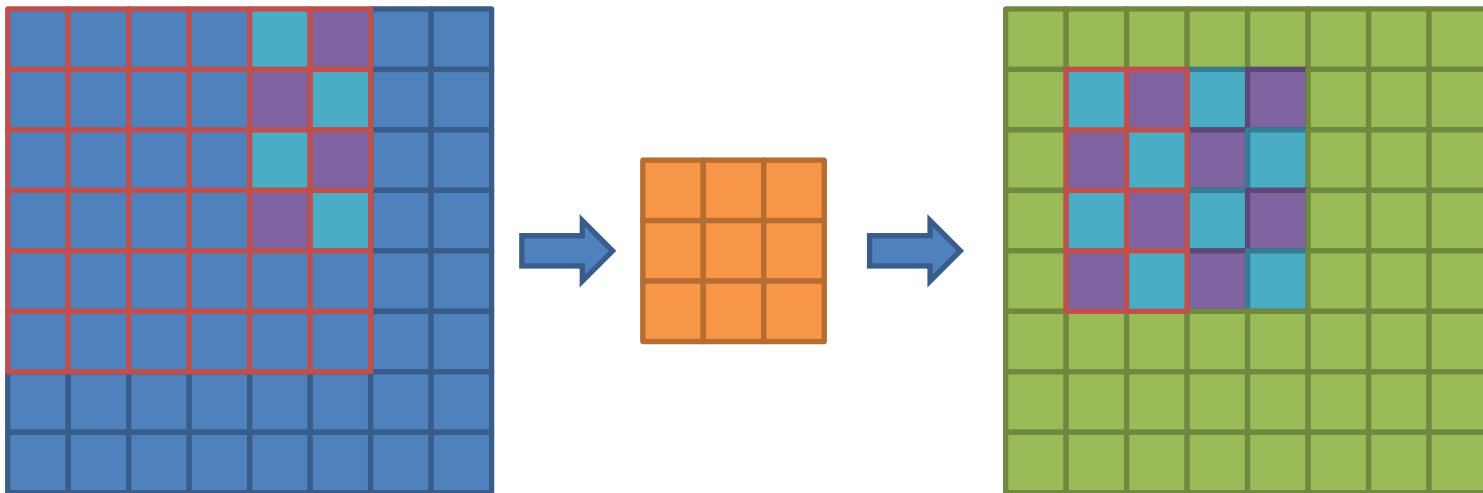
- Convolución



```
if(threadIdx.x < 2)
    smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x + blockDim.x] =
        src[?];
```

CUDA

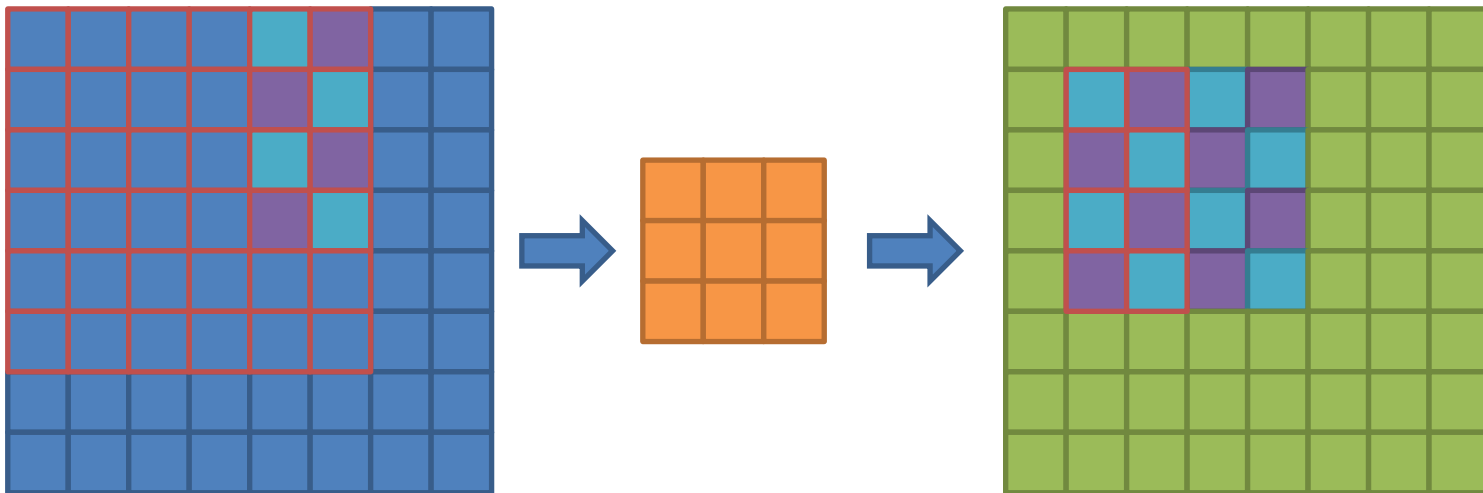
- Convolución



```
if(threadIdx.x < 2)
    smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x + blockDim.x] =
        src[(idy - 1) * cols + (idx - 1)];
```

CUDA

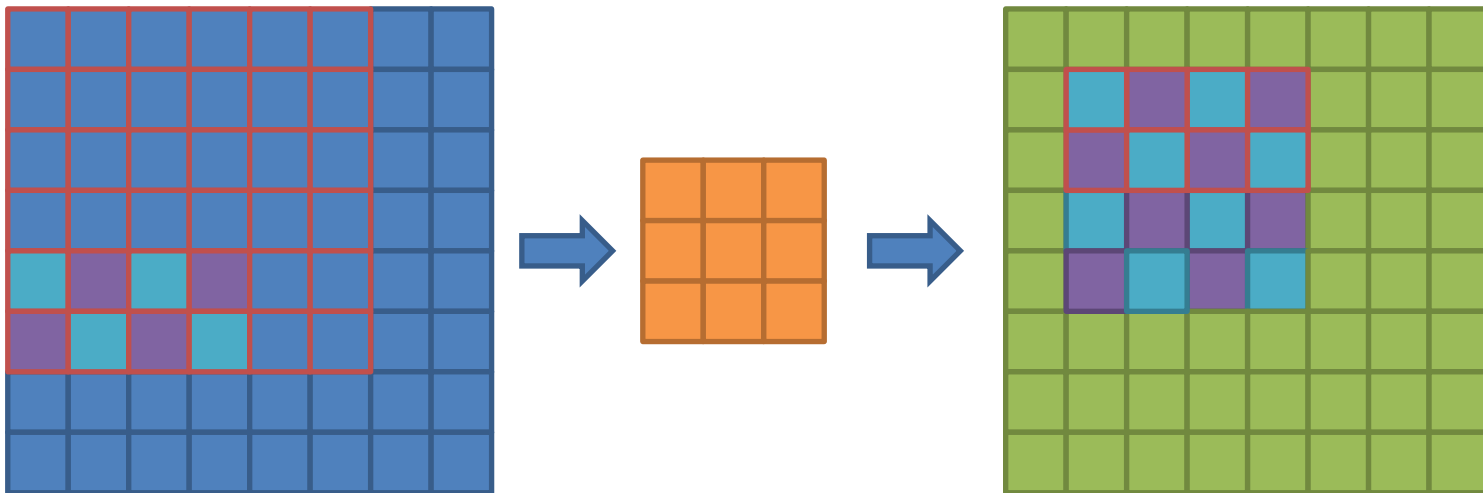
- Convolución



```
if(threadIdx.x < 2)
    smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x + blockDim.x] =
        src[(idy - 1) * cols + (idx - 1 + blockDim.x)];
```

CUDA

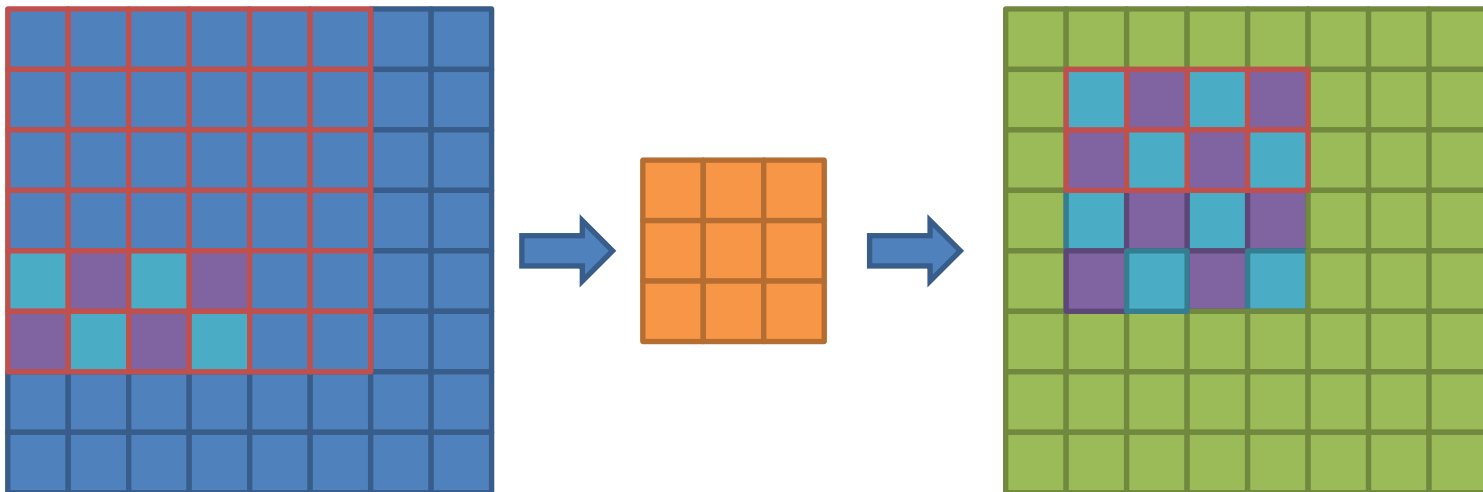
- Convolución



```
if(threadIdx.y < 2)
    smem[threadIdx.y * (blockdim.x + 2) + threadIdx.x] =
        src[(idy - 1) * cols + (idx - 1)];
```

CUDA

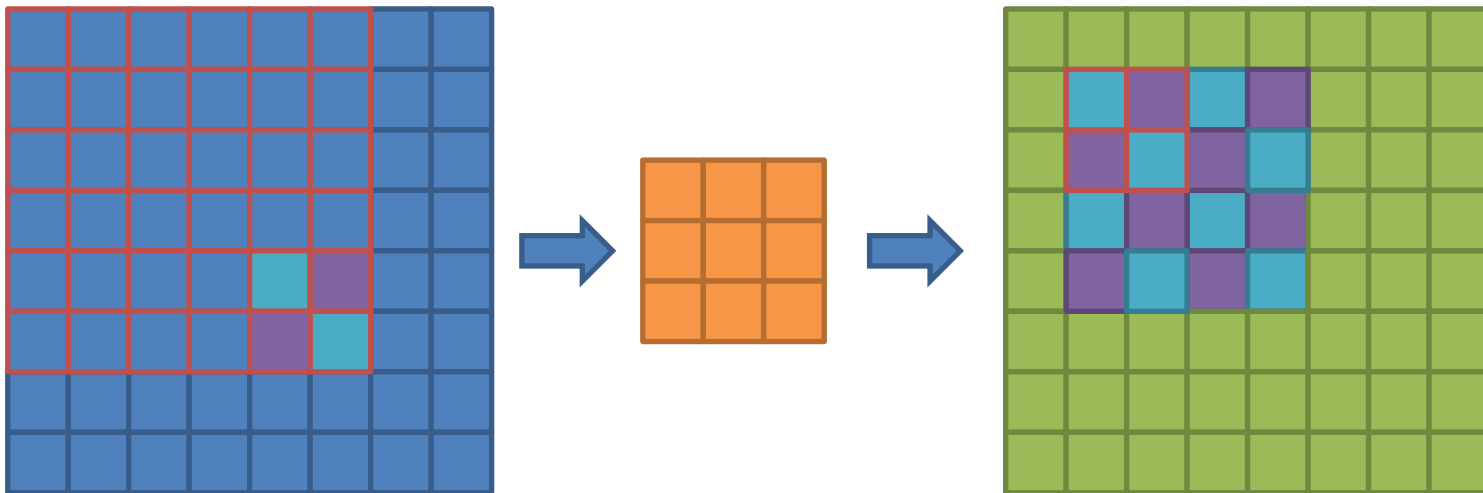
- Convolución



```
if(threadIdx.y < 2)
    smem[(threadIdx.y + blockDim.y) * (blockdim.x + 2) + threadIdx.x] =
        src[(idy - 1 + blockDim.y) * cols + (idx - 1)];
```

CUDA

- Convolución



```
if(threadIdx.x < 2 && threadIdx.y < 2)
    smem[(threadIdx.y + blockDim.y) * (blockdim.x + 2) + threadIdx.x + blockDim.x] =
        src[(idy - 1 + blockDim.y) * cols + (idx - 1 + blockDim.x)];
```


Procesadores Gráficos Avanzados

Práctica – CUDA

Dr. Antonio Sanz Montemayor

David Concha Gómez