GPGPU Programming with Triton

Abhinav Bhatele, Daniel Nichols

UNIVERSITY OF MARYLAND

Announcements

- Assignment 1 released tonight
  - Due Feb. 25th at midnight

What is Triton?

- Python library for GPU programming
- Targets ML applications
- Released by OpenAI, open-source
- Designed to automatically optimize
  - Memory movement and locality
  - Work partitioning within and between SMs

https://openai.com/index/triton/

Sneak Peek

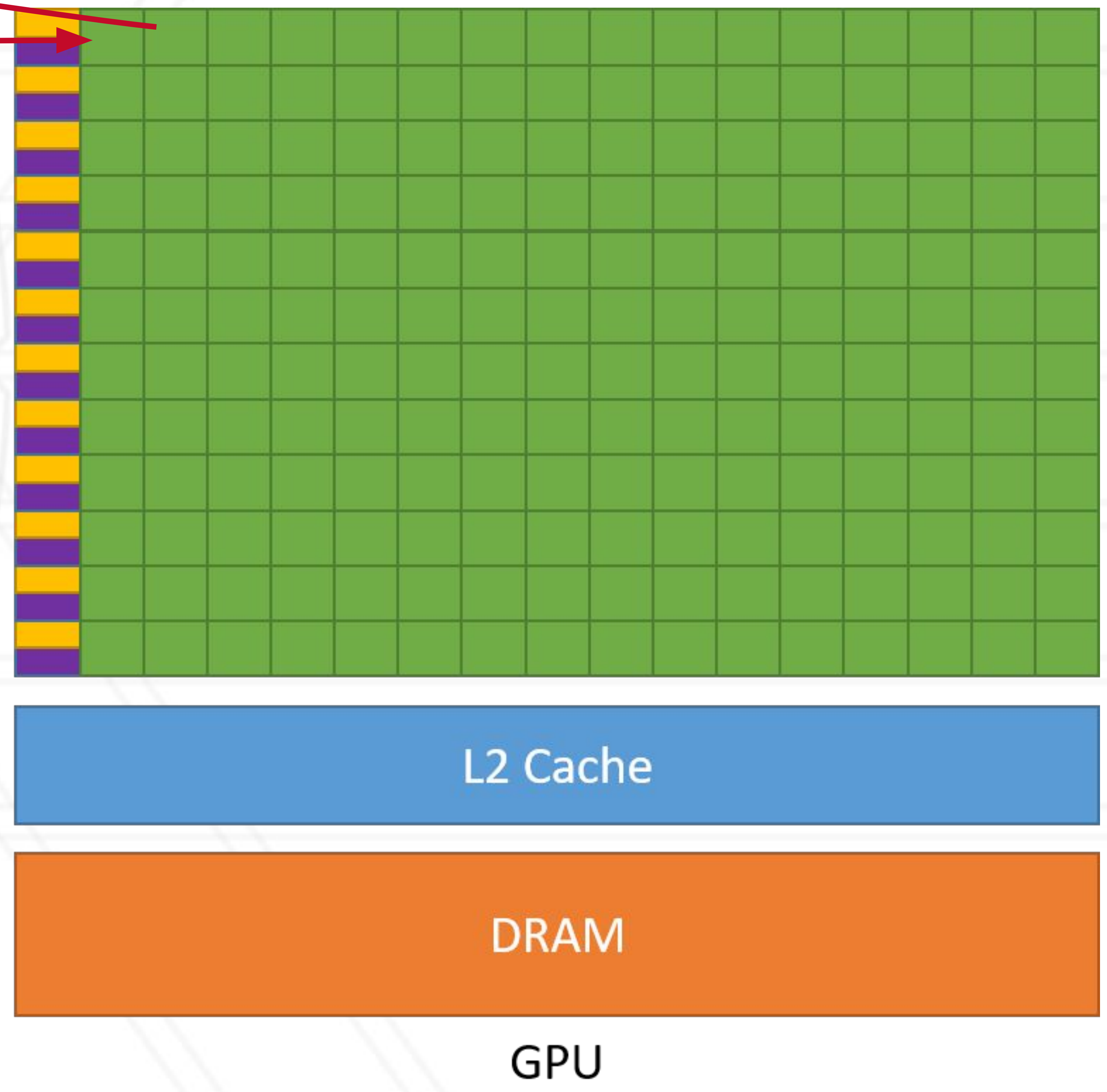
```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N, BLOCK_SIZE: tl.constexpr):
    pid = tl.program_id(0)
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    mask = offsets < N

    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + offsets, z, mask=mask)
```

CUDA vs Triton

```
__global__ void saxpy(float *x, float *y, float *z, float alpha, size_t N) {
    int t0 = blockDim.x * blockIdx.x + threadIdx.x;
    int stride = gridDim.x * blockDim.x;

    for (int i = t0; i < N; i += stride)
        z[i] = alpha*x[i] + y[i];
}
```

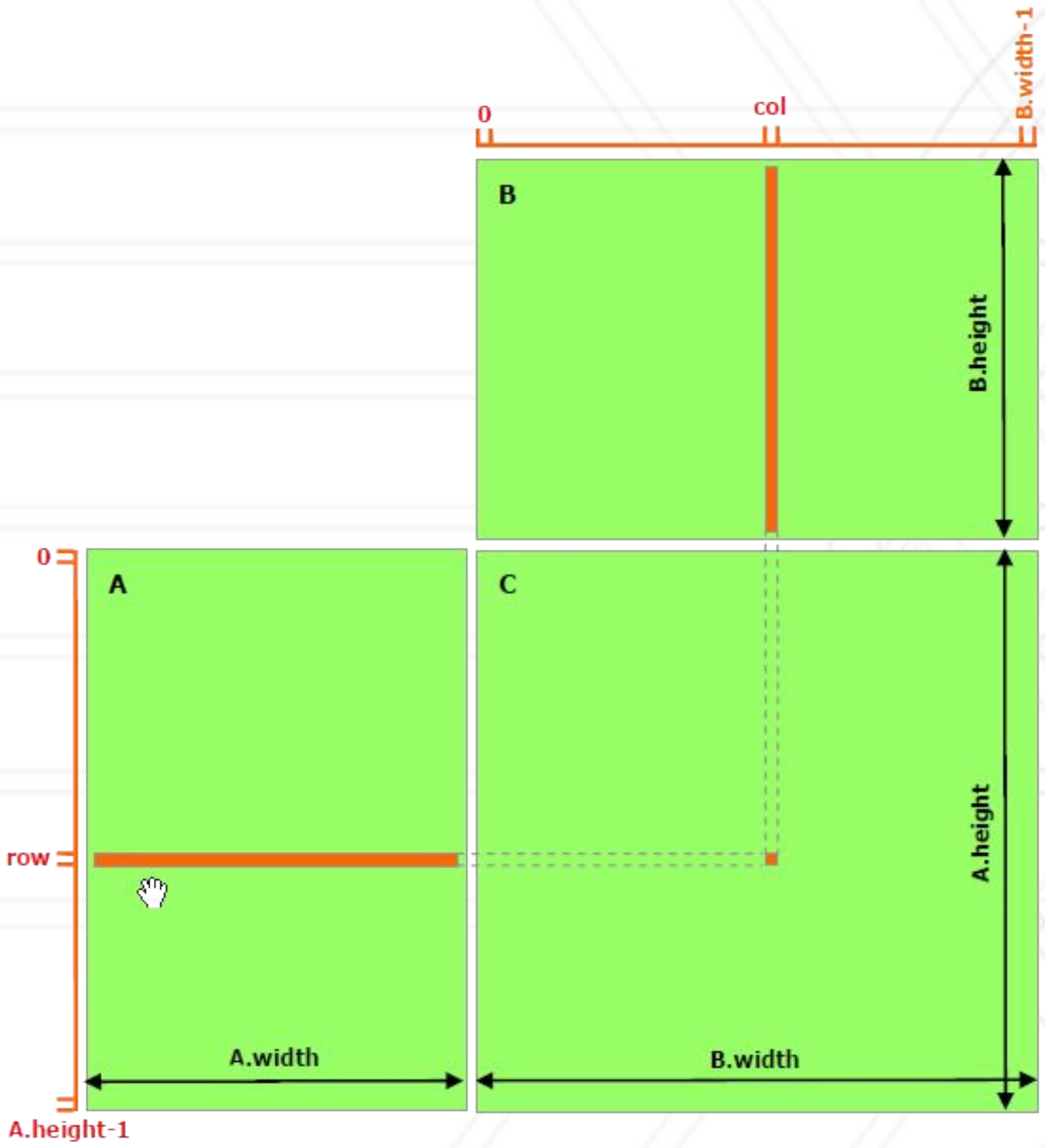
CUDA vs Triton

```
__global__ void saxpy(float *x, float *y, float *z, float alpha, size_t N) {
   int t0 = blockDim.x * blockIdx.x + threadIdx.x;
   int stride = gridDim.x * blockDim.x;

   for (int i = t0; i < N; i += stride)
       z[i] = alpha*x[i] + y[i];
}
```

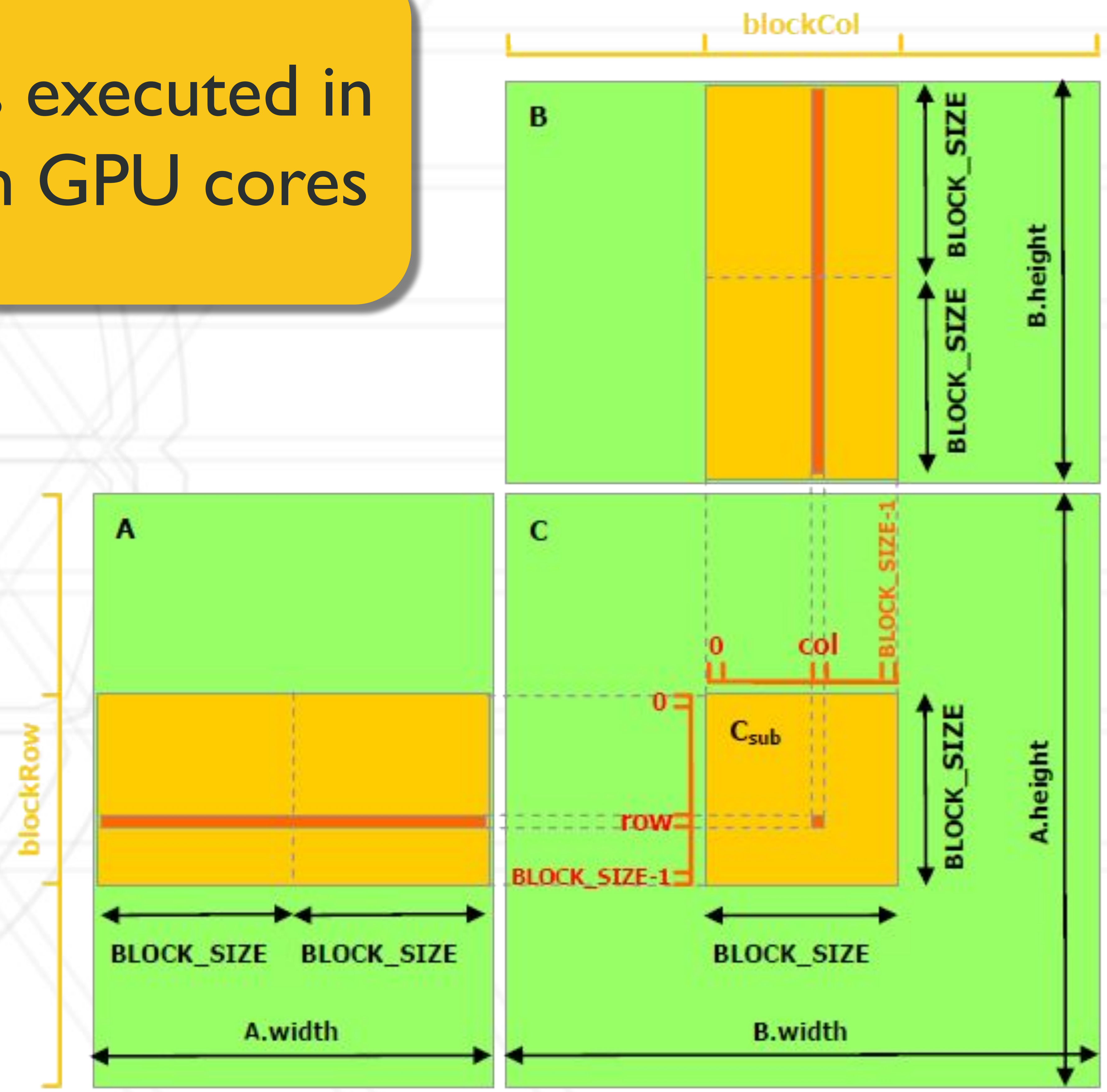


CUDA kernel is executed in parallel 1-1 with GPU cores

CUDA vs Triton



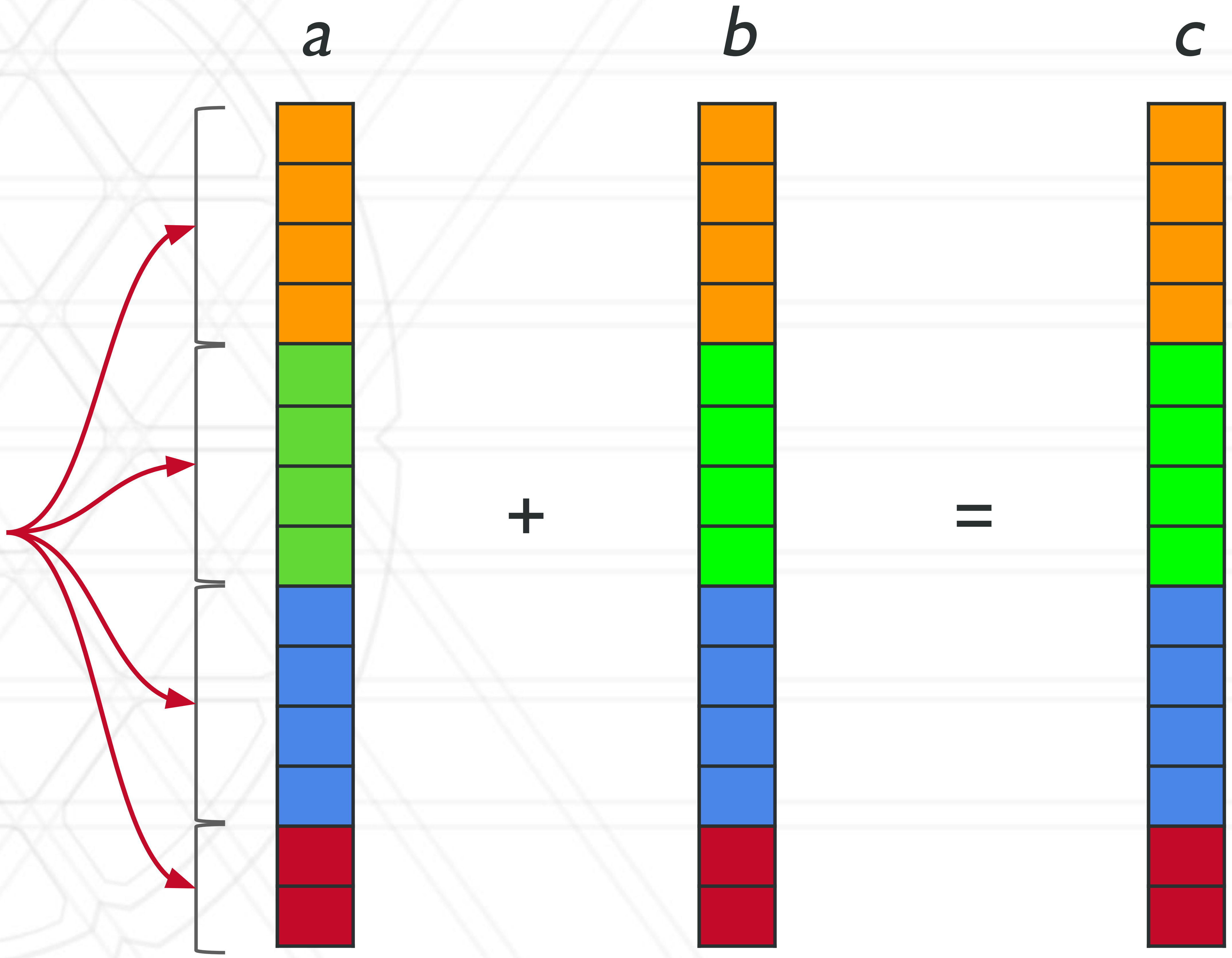
CUDA kernel is executed in parallel 1-1 with GPU cores



CUDA vs Triton

Triton kernels execute in parallel, but map 1-1 with blocks of data

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N, BLOCK_SIZE: tl.constexpr):
    pid = tl.program_id(0)
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    mask = offsets < N

    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + offsets, z, mask=mask)
```



Hardware mapping is opaque to developer

Triton basics: saxpy

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N):
    pid = tl.program_id(0)
    mask = pid < N

    x = tl.load(x_ptr + pid, mask=mask)
    y = tl.load(y_ptr + pid, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + pid, z, mask=mask)
```

triton.jit annotation indicates a triton kernel to be executed on the GPU

Triton basics: saxpy

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N):
    pid = tl.program_id(0)
    mask = pid < N

    x = tl.load(x_ptr + pid, mask=mask)
    y = tl.load(y_ptr + pid, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + pid, z, mask=mask)
```

data is passed to the kernel as pointers like in CUDA

Triton basics: saxpy

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N):
    pid = tl.program_id(0)
    mask = pid < N

    x = tl.load(x_ptr + pid, mask=mask)
    y = tl.load(y_ptr + pid, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + pid, z, mask=mask)
```

program_id gives the index of the current parallel kernel

kernel ids can be partitioned across multiple dimensions

Triton basics: saxpy

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N):
    pid = tl.program_id(0)
    mask = pid < N

    x = tl.load(x_ptr + pid, mask=mask)
    y = tl.load(y_ptr + pid, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + pid, z, mask=mask)
```

values are explicitly loaded and stored into memory with *tl.load* and *tl.store*

Triton basics: saxpy

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N):
    pid = tl.program_id(0)
    mask = pid < N

    x = tl.load(x_ptr + pid, mask=mask)
    y = tl.load(y_ptr + pid, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + pid, z, mask=mask)
```

one kernel per data point limits triton's optimizations

Saxpy with block size > 1

```
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N, BLOCK_SIZE: tl.constexpr):
    pid = tl.program_id(0)
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    mask = offsets < N

    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    z = alpha*x + y
    tl.store(z_ptr + offsets, z, mask=mask)
```

execute with a block: multiple values per kernel

tl.constexpr denotes a compile time constant

block sizes must be a power of 2 and constexpr

Launching the Kernel on the GPU

```
def mysaxpy(x: torch.Tensor, y: torch.Tensor, alpha.
    z = torch.empty_like(x)
    N = z.numel()

    BLOCK_SIZE = 1024
    grid = (triton.cdiv(N, BLOCK_SIZE), )
    saxpy[grid](x, y, z, alpha, N, BLOCK_SIZE=BLOCK_SIZE)
    return z
```

Triton implicitly converts torch tensors to pointers

The grid is passed as a N-D tuple to launch a grid of kernel instances

z is returned, but kernel is still running asynchronously

Launching the Kernel on the GPU

```python
def mysaxpy(x: torch.Tensor, y: torch.Tensor, alpha: float):
    z = torch.empty_like(x)
    N = z.numel()

    grid = lambda meta: (triton.cdiv(N, meta['BLOCK_SIZE']), )
    saxpy[grid](x, y, z, alpha, N, BLOCK_SIZE=1024)
    return z
```

The grid can also be a function that returns a tuple based on kernel args

Under the hood

- @triton.jit kernels are compiled to MLIR upon first execution
- MLIR is compiled to PTX, which is assembled into a CUBIN and run on GPU
  - MLIR enables a number of custom optimizations making Triton fast
- thread and block counts are determined at compile time
- Supports AMD GPU backend

Softmax

$$\boldsymbol{z} = \boldsymbol{x} - \max \boldsymbol{x}$$

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{\boldsymbol{z}_i}}{\sum_j e^{\boldsymbol{z}_j}}$$

Softmax

compute the softmax of each row in the matrix

$$\boldsymbol{z} = \boldsymbol{x} - \max \boldsymbol{x}$$

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{e^{\boldsymbol{z}_i}}{\sum_j e^{\boldsymbol{z}_j}}$$

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

compute the softmax of each row in the matrix

$$\boldsymbol{z} = \boldsymbol{x} - \max \boldsymbol{x}$$

$$\operatorname{softmax}(\boldsymbol{z})_i = \frac{e^{\boldsymbol{z}_i}}{\sum_j e^{\boldsymbol{z}_j}}$$

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Each kernel instance computes a row of the matrix

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

row = 1

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= in_ptr + 6

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= [0, 1, …, 7]

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= in_ptr + [6, 7, …, 13]

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```python
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= [T, T, T, T, T, T, F, F]

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= [1.1, -0.3, …, 1.6, -inf, -inf]

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```python
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= softmax(row)

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```python
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

= out_ptr + 6

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
   row = tl.program_id(0)

   row_start_ptr = in_ptr + row * n_cols
   col_offsets = tl.arange(0, BLOCK_SIZE)
   input_ptrs = row_start_ptr + col_offsets
   mask = col_offsets < n_cols

   row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

   row_minus_max = row - tl.max(row, axis=0)
   numerator = tl.exp(row_minus_max)
   denominator = tl.sum(numerator, axis=0)
   softmax_output = numerator / denominator

   output_row_start_ptr = out_ptr + row_idx * n_cols
   output_ptrs = output_row_start_ptr + col_offsets
   tl.store(output_ptrs, softmax_output, mask=mask)
```

= out_ptr + [6, 7, …, 13]

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Softmax

Tensor shapes are known at compile time

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    row_start_ptr = in_ptr + row * n_cols
    col_offsets = tl.arange(0, BLOCK_SIZE)
    input_ptrs = row_start_ptr + col_offsets
    mask = col_offsets < n_cols

    row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

    row_minus_max = row - tl.max(row, axis=0)
    numerator = tl.exp(row_minus_max)
    denominator = tl.sum(numerator, axis=0)
    softmax_output = numerator / denominator

    output_row_start_ptr = out_ptr + row_idx * n_cols
    output_ptrs = output_row_start_ptr + col_offsets
    tl.store(output_ptrs, softmax_output, mask=mask)
```

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
[illegible]	2.4	6.4	-7.9	-8.1	0.2
[illegible]	[illegible]	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Operations are automatically parallelized by Triton

Softmax

More work can be distributed to kernels using striding

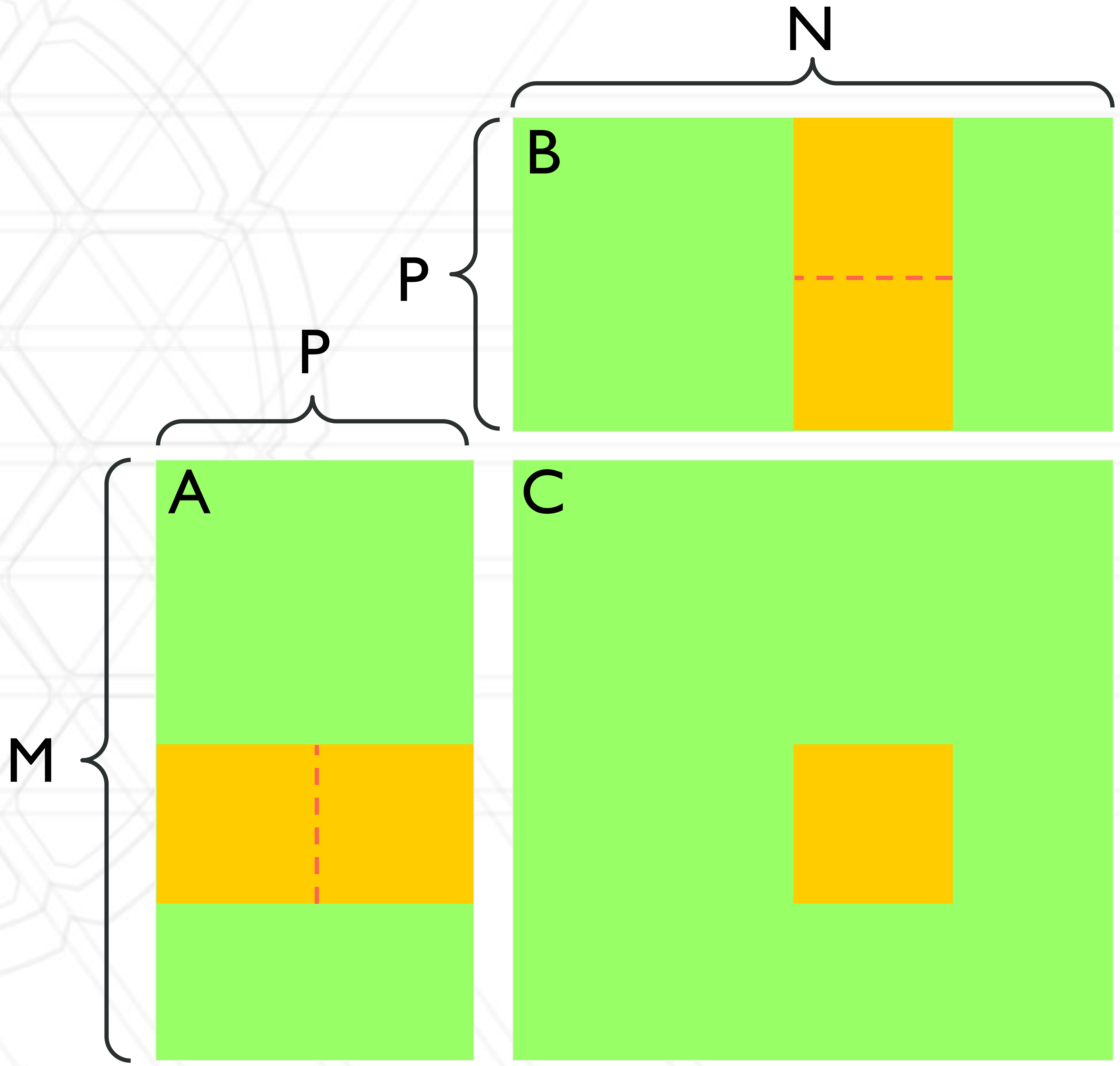
```python
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row_start = tl.program_id(0)
    stride = tl.num_programs(0)

    for row_idx in tl.range(row_start, n_rows, stride):
        row_start_ptr = in_ptr + row_idx * n_cols
        col_offsets = tl.arange(0, BLOCK_SIZE)
        input_ptrs = row_start_ptr + col_offsets
        mask = col_offsets < n_cols

        row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

        row_minus_max = row - tl.max(row, axis=0)
        numerator = tl.exp(row_minus_max)
        denominator = tl.sum(numerator, axis=0)
        softmax_output = numerator / denominator

        output_row_start_ptr = out_ptr + row_idx * n_cols
        output_ptrs = output_row_start_ptr + col_offsets
        tl.store(output_ptrs, softmax_output, mask=mask)
```

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Questions?

Matrix Multiplication

- Each kernel computes a sub-block of C
- *tl.dot* can be used to accumulate matrix products



tl.where

```
@triton.jit
def dropout(x_ptr,  mask_ptr, output_ptr, n_elements, p, BLOCK_SIZE: tl.constexpr):
   pid = tl.program_id(axis=0)
   block_start = pid * BLOCK_SIZE
   offsets = block_start + tl.arange(0, BLOCK_SIZE)
   mask = offsets < n_elements

   x = tl.load(x_ptr + offsets, mask=mask)
   x_mask = tl.load(mask_ptr + offsets, mask=mask)

   output = tl.where(x_mask, x / (1 - p), 0.0)

   tl.store(output_ptr + offsets, output, mask=mask)
```

returns x/(1-p) where x_mask is true, 0.0 otherwise

Performance Parameters

- num_stages
  - pipelines asynchronous loads (A100 onwards)

Softmax

pipeline loop into 4 stages

```
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row_start = tl.program_id(0)
    stride = tl.num_programs(0)

    for row_idx in tl.range(row_start, n_rows, stride, num_stages=4):
        row_start_ptr = in_ptr + row_idx * n_cols
        col_offsets = tl.arange(0, BLOCK_SIZE)
        input_ptrs = row_start_ptr + col_offsets
        mask = col_offsets < n_cols

        row = tl.load(input_ptrs, mask=mask, other=-float('inf'))

        row_minus_max = row - tl.max(row, axis=0)
        numerator = tl.exp(row_minus_max)
        denominator = tl.sum(numerator, axis=0)
        softmax_output = numerator / denominator

        output_row_start_ptr = out_ptr + row_idx * n_cols
        output_ptrs = output_row_start_ptr + col_offsets
        tl.store(output_ptrs, softmax_output, mask=mask)
```

2.3	-0.1	4.9	0.0	6.7	-2.0
1.1	-0.3	0.0	-5.8	9.2	1.6
-3.0	2.4	6.4	-7.9	-8.1	0.2
-0.6	1.0	3.2	0.9	7.6	-6.0
5.4	4.2	2.0	8.3	7.5	3.0
9.9	7.4	-0.7	-6.3	3.1	8.4

Performance Parameters

- num_stages
  - pipelines asynchronous loads (A100 onwards)
- num_warps
  - How many warps to assign to each kernel instance
- maxnreg
  - max registers per thread
- num_ctas
  - number of blocks in cluster
  - H100 and later
- block sizes

Autotuning

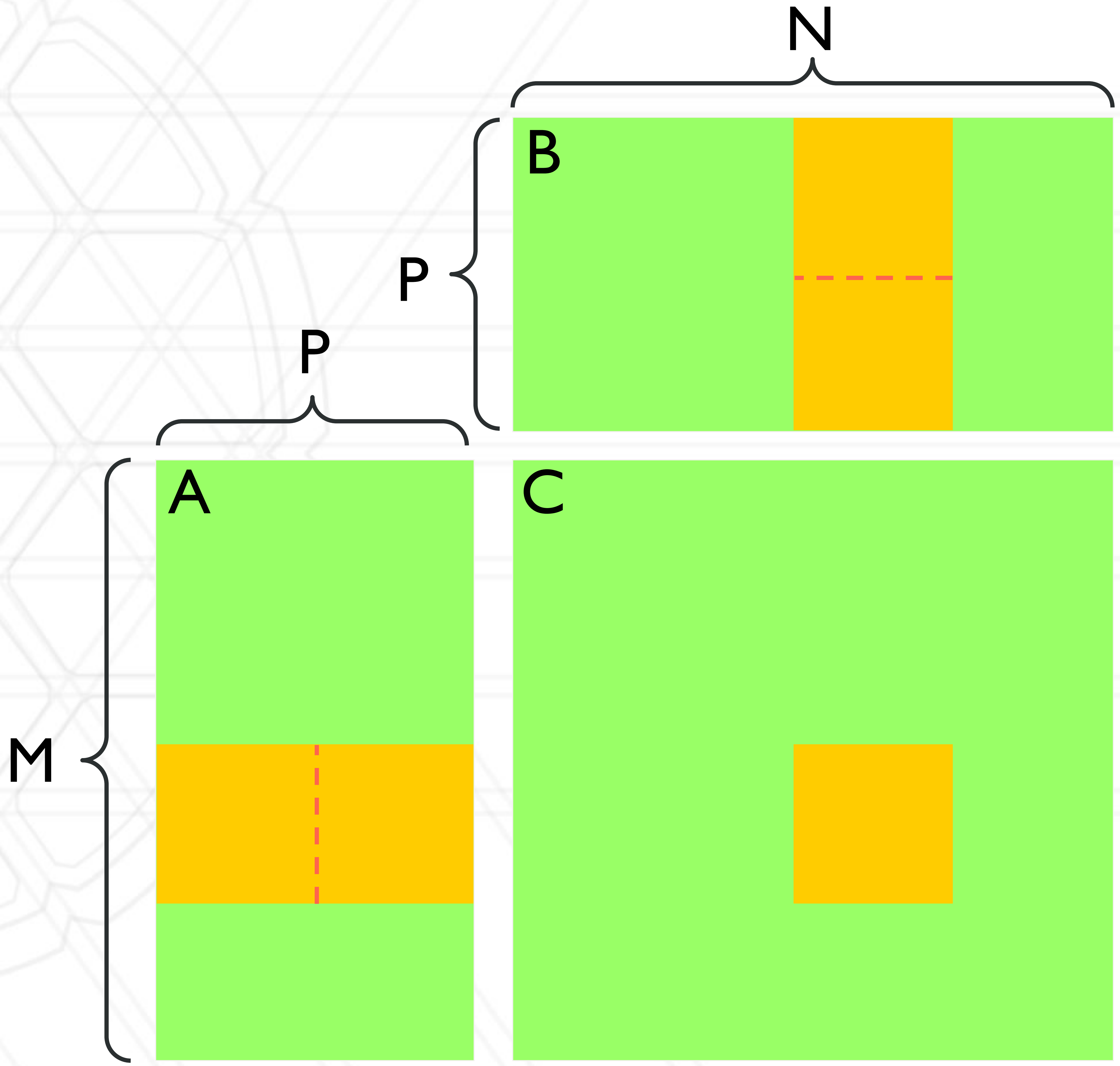
Automatically searches parameter space

```
@triton.autotune(
    configs=[
        triton.Config({BLOCK_SIZE: 32}, num_warps=4),
        triton.Config({BLOCK_SIZE: 64}, num_warps=4),
        triton.Config({BLOCK_SIZE: 64}, num_warps=2),
    ]
)
@triton.jit
def softmax_kernel(out_ptr, in_ptr, n_rows, n_cols, BLOCK_SIZE: tl.constexpr):
    row = tl.program_id(0)

    ..................
```

Returns kernel with fastest parameters

Kernel Fusion

- Combining two operator kernels into one
- Useful for consecutive memory-bound kernels
- Consider matrix multiply + activation



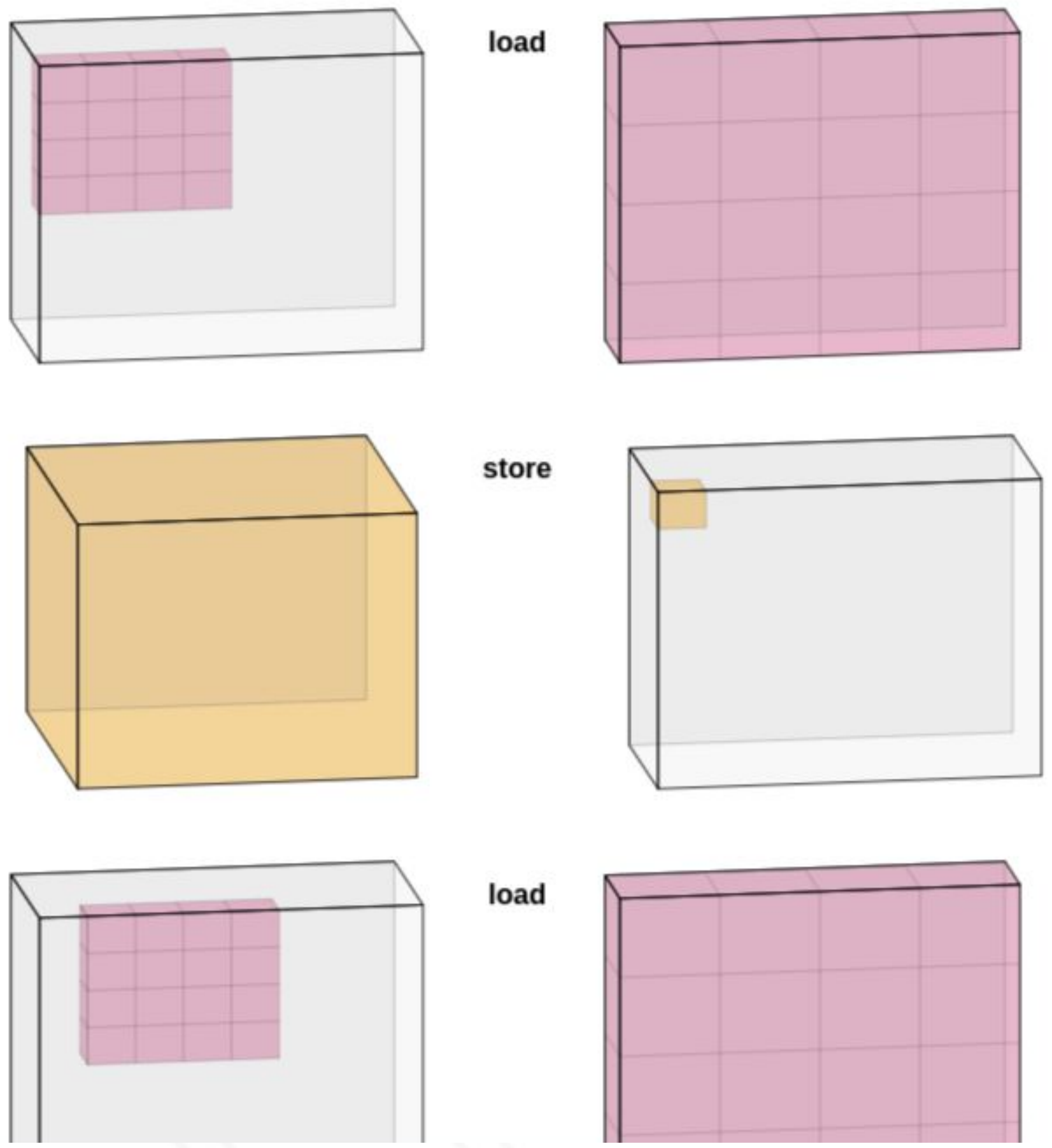
Debugging

- *tl.static_assert* and *tl.static_print*
  - check properties at compile time
- *tl.device_assert* and *tl.device_print*
  - TRITON_DEBUG must be set for assertions at runtime

```python
@triton.jit
def saxpy(x_ptr, y_ptr, z_ptr, alpha, N):
  pid = tl.program_id(0)
  mask = pid < N

  x = tl.load(x_ptr + pid, mask=mask)
  y = tl.load(y_ptr + pid, mask=mask)
  z = alpha*x + y
  tl.device_print("checking z", pid, z)
  tl.store(z_ptr + pid, z, mask=mask)
```

Debugging

- *tl.static_assert* and *tl.static_print*
  - check properties at compile time
- *tl.device_assert* and *tl.device_print*
  - TRITON_DEBUG must be set for assertions at runtime
- Interpreter
  - Set TRITON_INTERPRET environment variable
  - Runs kernels sequentially using numpy to compute results
  - Use print or pdb to step-by-step debug
- triton-viz
  - https://github.com/Deep-Learning-Profiling-Tools/triton-viz



Integrating with PyTorch

```
class MySaxpyOperator(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y, alpha):
        z = mysaxpy(x)
        ctx.save_for_backward(z, x, y, alpha)
        return z

    @staticmethod
    def backward(ctx, grad_output):
        z, x, y, alpha = ctx.saved_tensors
        return mysaxpy_backwards(z, grad_output)

MySaxpy = MySaxpyOperator.apply
```

autograd Function classes allow us to create new operators

Define forward and backward functions

Finally, we create the operator

Correctness Checking

- torch.allclose(tensor1, tensor2, atol=..., rtol=...)
  - checks if floating point tensors are "equal"
  - tolerances depend on the algorithm and amount of data
  - consider forward and backward error for linear algebra problems
- torch.autograd.gradcheck(op, input_tensor)
  - checks if gradient is implemented correctly

Assignment 1

- 2 operations, forward and backward = 4 kernels
- Refer to https://triton-lang.org/main/python-api/triton.language.html
- Kernel performance
  - Should be roughly similar to PyTorch
- Report
  - Try fusion and hyperparameter sweep
  - Write about and plot findings
- Start early, Zaratan GPUs are a shared resource
- Late policy

Happy Snow Day

UNIVERSITY OF
MARYLAND