Triton-Shared

Scaling Triton to Multiple Platforms

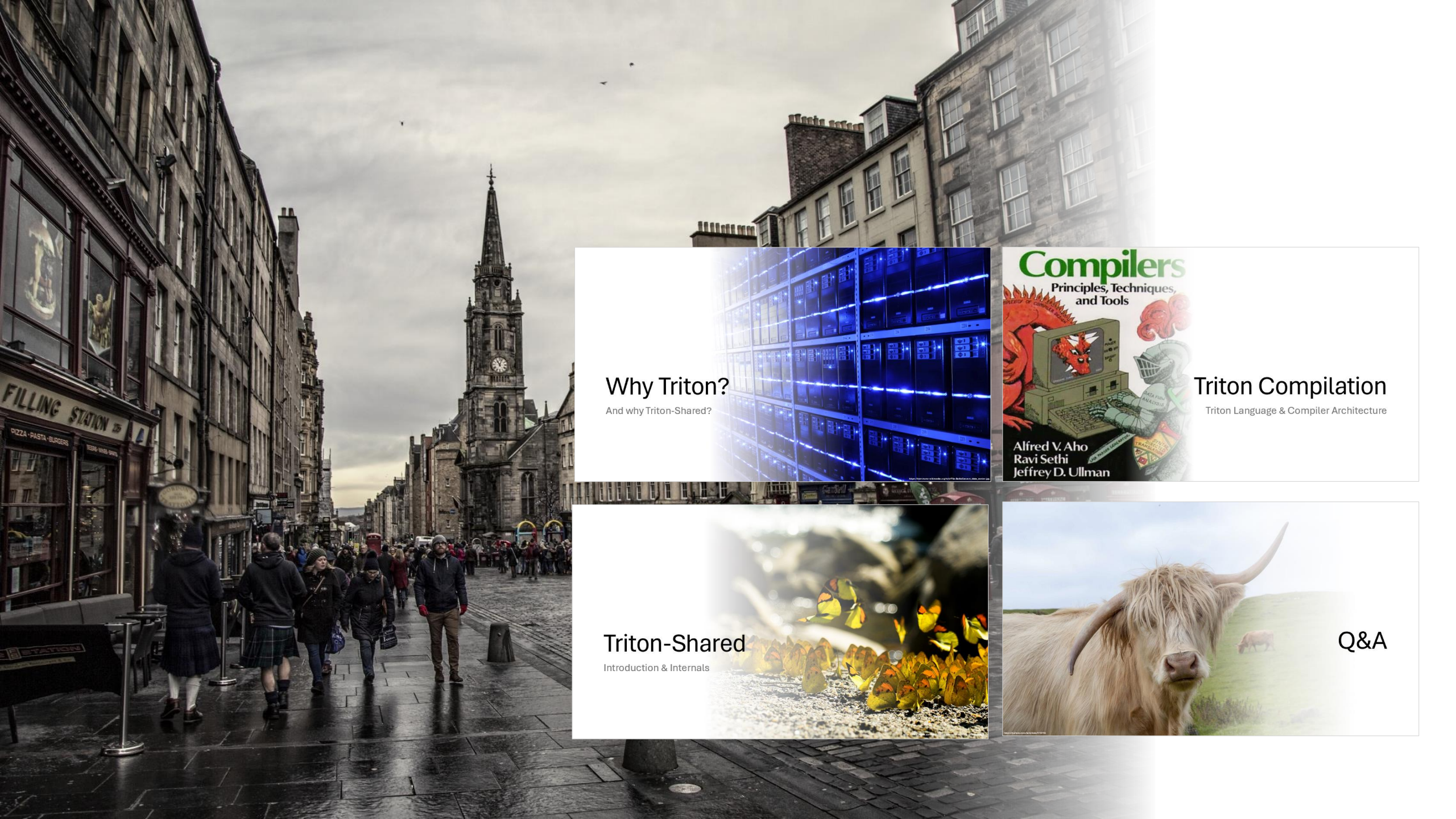
ian Bearman
Microsoft

C4ML
March 3, 2024

https://www.publicdomainpictures.net/en/view-image.php?image=236238

Today

- Why Triton?
  (5 minutes)
- Triton Compilation
  (10 minutes)
- Triton-Shared
  (15 minutes)
- Q&A
  (10 minutes)



Why Triton?

And why Triton-Shared?

Triton Compilation

Triton Language & Compiler Architecture

Triton-Shared

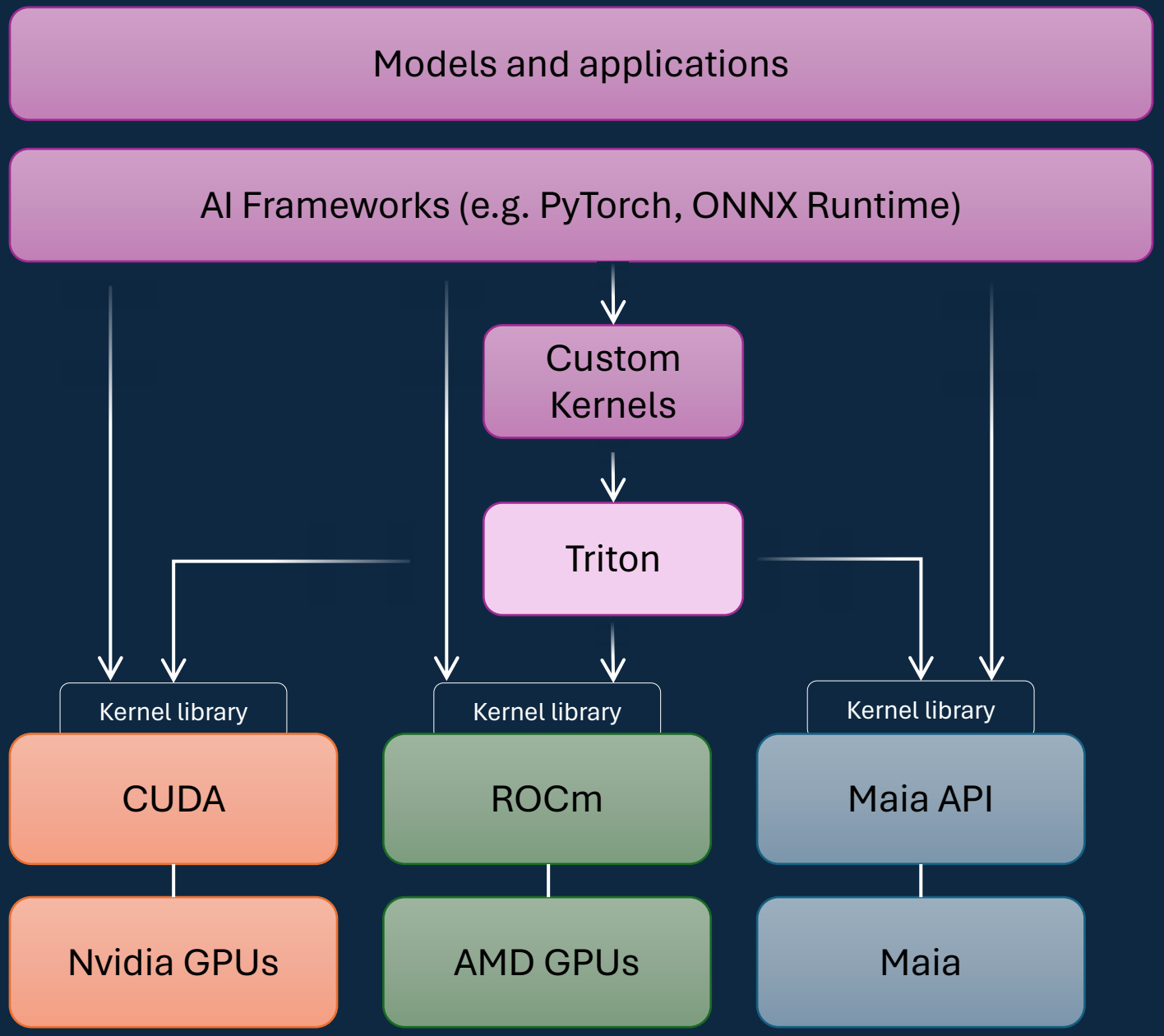
Introduction & Internals

Q&A

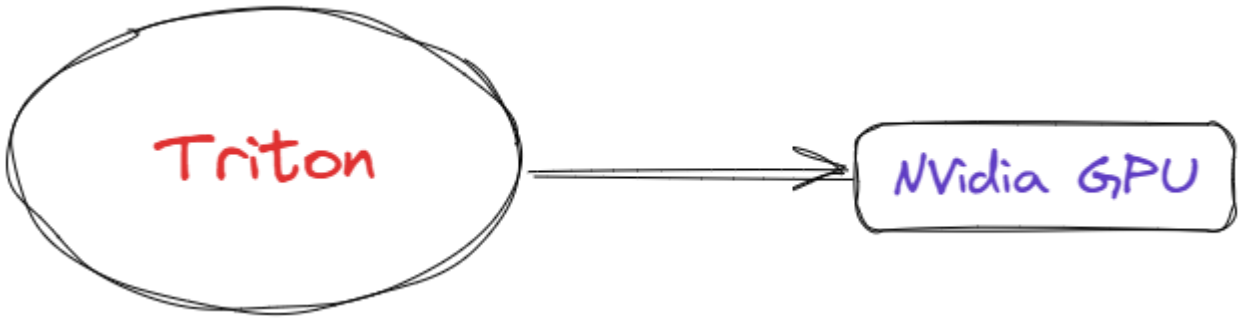
Why Triton?

And why Triton-Shared?

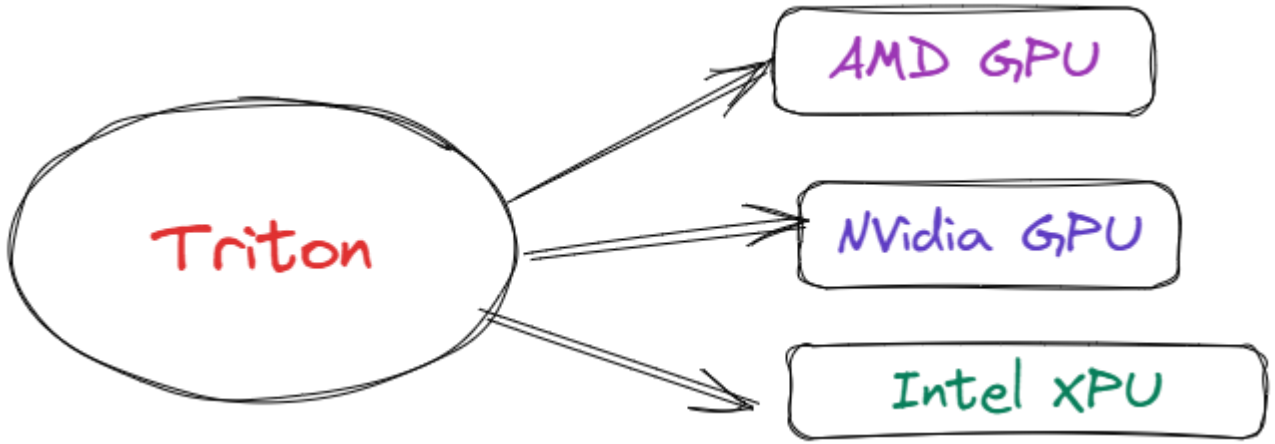
https://commons.wikimedia.org/wiki/File:BalticServers_data_center.jpg


Models and applications
AI Frameworks (e.g. PyTorch, ONNX Runtime)
Custom Kernels
Triton
Kernel library
Kernel library
Kernel library
CUDA
ROCm
Maia API
Nvidia GPUs
AMD GPUs
Maia

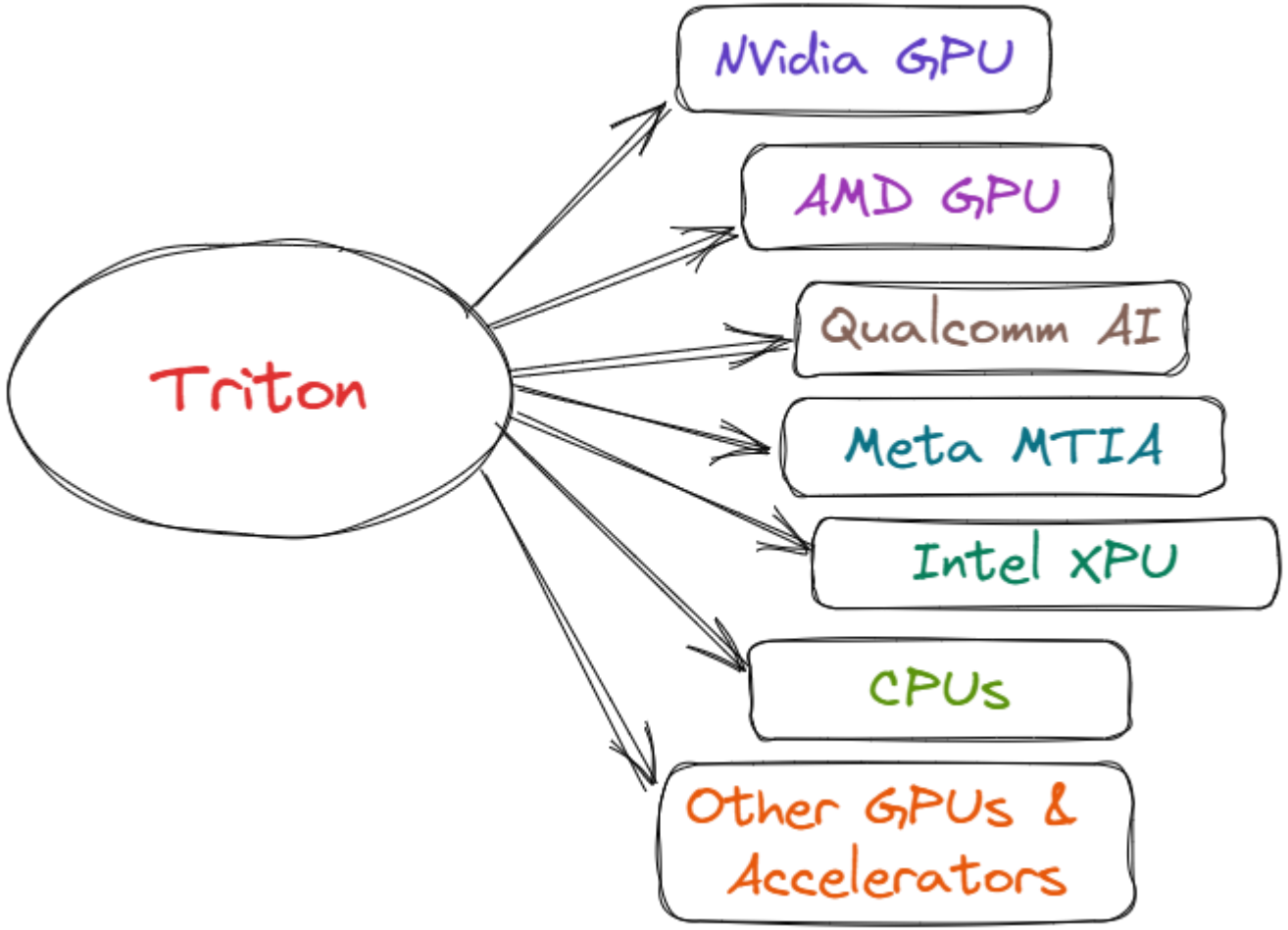
Triton for All



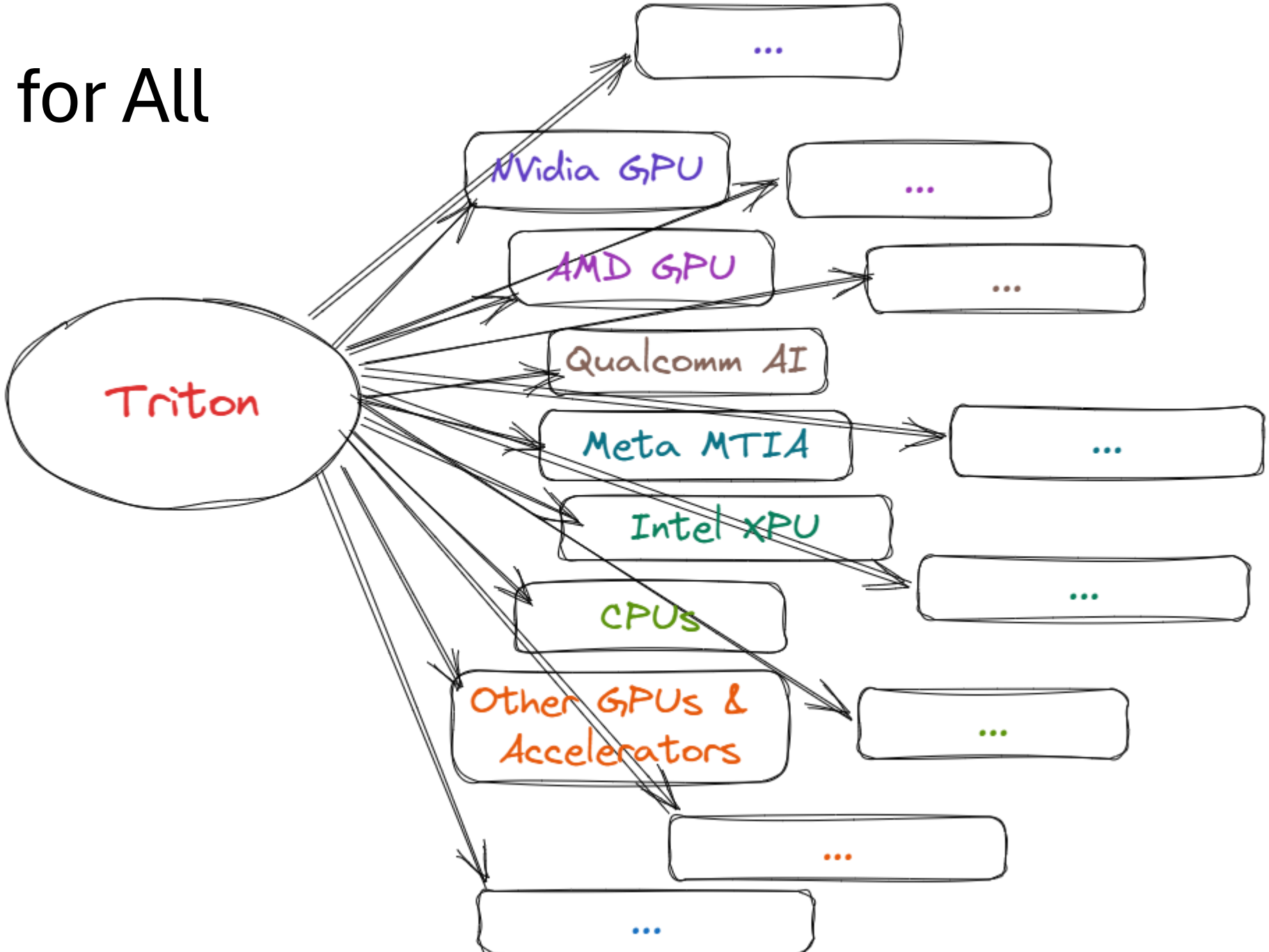
Triton for All

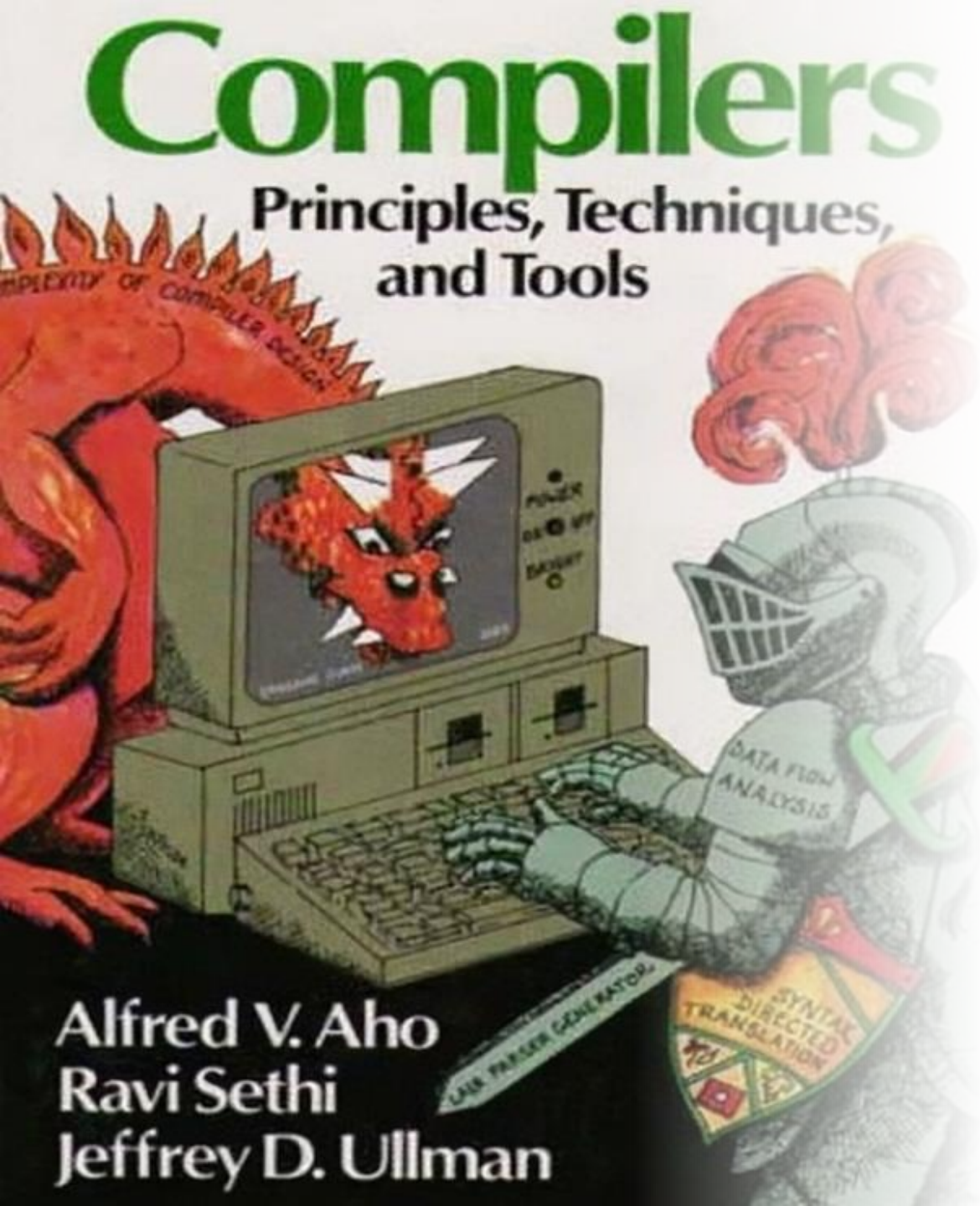


Triton for All



Triton for All





Triton Compilation

Triton Language & Compiler Architecture

What is Triton

- A domain specific **language** and **compiler** for *fast* DNN kernels

- Developed and open sourced by **Open AI**
  - Used by Open AI for latest models
  - Integrated into PyTorch 2.0 via TorchInductor
  - Growing interest throughout the ML community

- Language Features
  - A **Python-based** programming environment
  - Designed for **productively** writing custom DNN compute kernels.
  - Supports **automatic tuning** to find the fastest version of the code
  - Allows **non-CUDA-experts** to write high-performance kernels

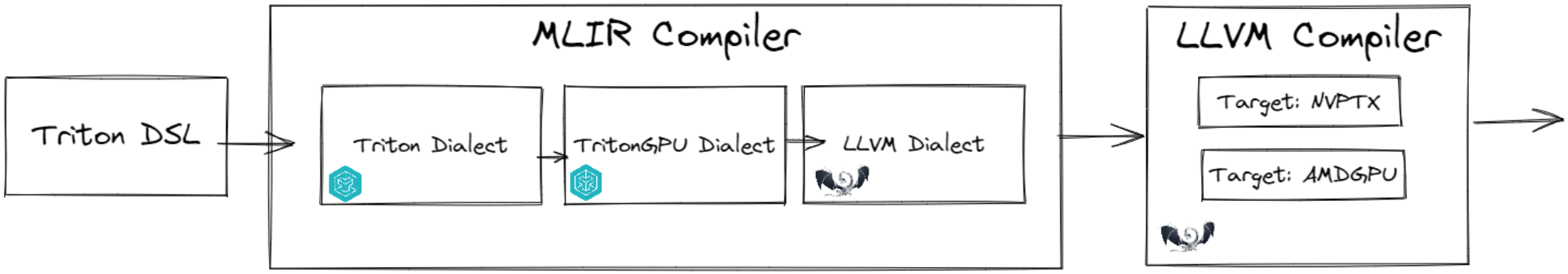
Triton Example Code

```
@triton.jit
def add_kernel(x_ptr,  y_ptr, output_ptr, n_elements, 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)
  y = tl.load(y_ptr + offsets, mask=mask)
  output = x + y
  tl.store(output_ptr + offsets, output, mask=mask)
```

```
def add(x, y):
  output = torch.empty_like(x)
  n_elements = output.numel()
  grid = lambda meta: (triton.cdiv(n_elements, meta['BLOCK_SIZE']),)
  add_kernel[grid](x, y, output, n_elements, BLOCK_SIZE=1024)
  return output

def my_model():
  x = torch.rand(size, device='cuda')
  y = torch.rand(size, device='cuda')
  output_triton = add(x, y)
```

Triton Compiler Architecture



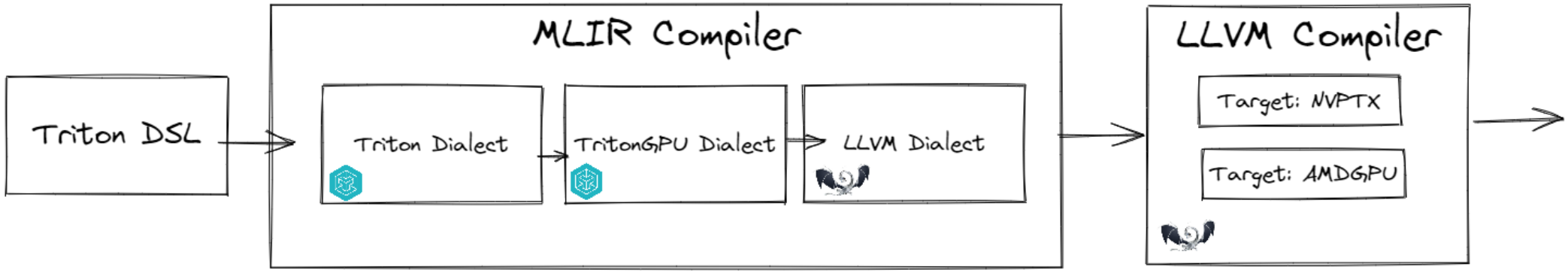
Middle Layer

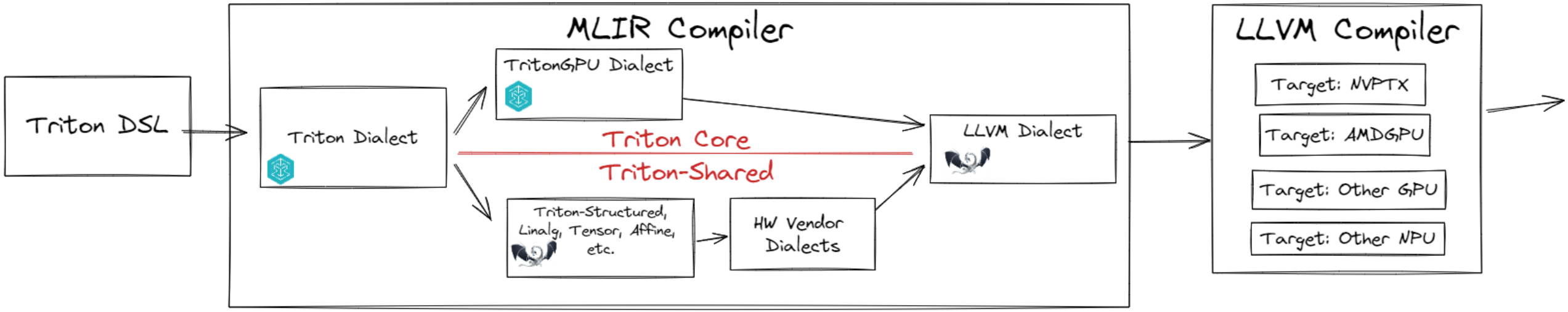
- Language-agnostic & Hardware-agnostic

- Why Hardware-Agnostic?
  - Share representation across hardware targets
  - Common analysis, optimizations and transformations

- Why Language-Agnostic?
  - Bring existing MLIR back-ends
  - Share with other front-ends

Triton Architecture, alternative





Triton-Shared

Introduction & Internals

What is Triton-Shared

https://github.com/microsoft/triton-shared

- Libraries for Your Compiler
  - Pointer Analysis (SIMT -> SIMD)
  - Dialect Lowering (Triton to Linalg)

- driver: triton-shared-opt

- Example compiler for non-GPU

- How to Use it & How to Contribute

Pointer Analysis

Pointer Analysis: Triton -> Triton IR

```python
import triton
import triton.language as tl

@triton.jit
def add_kernel(
    x_ptr,  # *Pointer* to first input vector.
    y_ptr,  # *Pointer* to second input vector.
    output_ptr,  # *Pointer* to output vector.
    n_elements,  # Size of the vector.
    BLOCK_SIZE: tl.constexpr,  # Number of elements each program should process.
                 # NOTE: `constexpr` so it can be used as a shape value.
):
    # There are multiple 'programs' processing different data. We identify which program
    # we are here:
    pid = tl.program_id(axis=0)  # We use a 1D launch grid so axis is 0.
    # This program will process inputs that are offset from the initial data.
    # For instance, if you had a vector of length 256 and block_size of 64, the programs
    # would each access the elements [0:64, 64:128, 128:192, 192:256].
    # Note that offsets is a list of pointers:
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    # Create a mask to guard memory operations against out-of-bounds accesses.
    mask = offsets < n_elements
    # Load x and y from DRAM, masking out any extra elements in case the input
    # multiple of the block size.
    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    output = x + y
    # Write x + y back to DRAM.
    tl.store(output_ptr + offsets, output, mask=mask)
```

```
module {
  tt.func public @add_kernel_01234(%arg0: !tt.ptr<f32>, %arg1: !tt.ptr<f32>, %arg2: !tt.ptr<f32>, %arg3: i32) {
    %c1024_i32 = arith.constant 1024 : i32
    %0 = tt.get_program_id {axis = 0 : i32} : i32
    %1 = arith.muli %0, %c1024_i32 : i32
    %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<1024xi32>
    %3 = tt.splat %1 : (i32) -> tensor<1024xi32>
    %4 = arith.addi %3, %2 : tensor<1024xi32>
    %5 = tt.splat %arg3 : (i32) -> tensor<1024xi32>
    %6 = arith.cmpi slt, %4, %5 : tensor<1024xi32>
    %7 = tt.splat %arg0 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
    %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
    %13 = arith.addf %9, %12 : tensor<1024xf32>
    %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
    %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
    tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
    tt.return
  }
}
```

Pointer Analysis: Triton Pointers

```
tt.func public @add_kernel_01234(%arg0: !tt.ptr<f32>, %arg1: !tt.ptr<f32>, %arg2: !tt.ptr<f32>, %arg
  %c1024_i32 = arith.constant 1024 : i32
  %0 = tt.get_program_id {axis = 0 : i32} : i32
  %1 = arith.muli %0, %c1024_i32 : i32
  %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<1024xi32>
  %3 = tt.splat %1 : (i32) -> tensor<1024xi32>
  %4 = arith.addi %3, %2 : tensor<1024xi32>
  %5 = tt.splat %arg3 : (i32) -> tensor<1024xi32>
  %6 = arith.cmpi slt, %4, %5 : tensor<1024xi32>
  %7 = tt.splat %arg0 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
  %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
  %13 = arith.addf %9, %12 : tensor<1024xf32>
  %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
  tt.return
```

Pointer Analysis: Triton Pointers

```
tt.func public @add_kernel_01234(%arg0: !tt.ptr<f32>, %arg1: !tt.ptr<f32>, %arg2: !tt.ptr<f32>, %arg
  %c1024_i32 = arith.constant 1024 : i32
  %0 = tt.get_program_id {axis = 0 : i32} : i32
  %1 = arith.muli %0, %c1024_i32 : i32
  %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<1024xi32>
  %3 = tt.splat %1 : (i32) -> tensor<1024xi32>
  %4 = arith.addi %3, %2 : tensor<1024xi32>
  %5 = tt.splat %arg3 : (i32) -> tensor<1024xi32>
  %6 = arith.cmpi slt, %4, %5 : tensor<1024xi32>
  %7 = tt.splat %arg0 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
  %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
  %13 = arith.addf %9, %12 : tensor<1024xf32>
  %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
  tt.return
```

Pointer Analysis: Triton Pointers

```
tt.func public @add_kernel_01234(%arg0: !tt.ptr<f32>, %arg1: !tt.ptr<f32>, %arg2: !tt.ptr<f32>, %arg
  %c1024_i32 = arith.constant 1024 : i32
  %0 = tt.get_program_id {axis = 0 : i32} : i32
  %1 = arith.muli %0, %c1024_i32 : i32
  %2 = tt.make_range {end = 1024 : i32, start = 0 : i32} : tensor<1024xi32>
  %3 = tt.splat %1 : (i32) -> tensor<1024xi32>
  %4 = arith.addi %3, %2 : tensor<1024xi32>
  %5 = tt.splat %arg3 : (i32) -> tensor<1024xi32>
  %6 = arith.cmpi slt, %4, %5 : tensor<1024xi32>
  %7 = tt.splat %arg0 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %8 = tt.addptr %7, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  %9 = tt.load %8, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
  %10 = tt.splat %arg1 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %11 = tt.addptr %10, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  %12 = tt.load %11, %6 {cache = 1 : i32, evict = 1 : i32, isVolatile = false} : tensor<1024xf32>
  %13 = arith.addf %9, %12 : tensor<1024xf32>
  %14 = tt.splat %arg2 : (!tt.ptr<f32>) -> tensor<1024x!tt.ptr<f32>>
  %15 = tt.addptr %14, %4 : tensor<1024x!tt.ptr<f32>>, tensor<1024xi32>
  tt.store %15, %13, %6 {cache = 1 : i32, evict = 1 : i32} : tensor<1024xf32>
  tt.return
```

Pointer Analysis: Triton Structured Pointers

```
tt.func public @add_kernel_01234(...) {
  %c1024 = arith.constant 1024 : index
  %c1024_i32 = arith.constant 1024 : i32
  %0 = tt.get_program_id x : i32
  %1 = arith.muli %0, %c1024_i32 : i32
  %2 = arith.index_cast %1 : i32 to index
  %3 = tts.make_tptr %arg0 to sizes: [1024], strides: [1], offsets: [%2] : tensor<1024x!tt.ptr<f32>>
  %4 = arith.addi %2, %c1024 : index
  %5 = arith.index_cast %arg3 : i32 to index
  %6 = arith.minsi %4, %5 : index
  %7 = arith.subi %6, %2 : index
  %8 = tts.load %3, %7 {...} : (tensor<1024x!tt.ptr<f32>>, index) -> tensor<1024xf32>
  %9 = tts.make_tptr %arg1 to sizes: [1024], strides: [1], offsets: [%2] : tensor<1024x!tt.ptr<f32>>
  %10 = tts.load %9, %7 {...} : (tensor<1024x!tt.ptr<f32>>, index) -> tensor<1024xf32>
  %11 = arith.addf %8, %10 : tensor<1024xf32>
  %12 = tts.make_tptr %arg2 to sizes: [1024], strides: [1], offsets: [%2] : tensor<1024x!tt.ptr<f32>>
  tts.store %12, %11, %7 {...} : (tensor<1024x!tt.ptr<f32>>, tensor<1024xf32>, index) -> ()
  tt.return
```

Pointer Analysis: 2-D Example

```
tts.make_tptr %arg1 sizes: [64, 256], strides: [%24, %25], offsets: [%arg15, %26] : tensor<64x256x!tt.ptr<bf16>>
tts.make_tptr %arg0 sizes: [128, 64], strides: [%21, %23], offsets: [%arg14, %c0] : tensor<128x64x!tt.ptr<bf16>>
```

Pointer Ana

```
tts.make_tptr %arg1 sizes: [64,                                                    5] : tensor<64x256x!tt.ptr<bf16>>
tts.make_tptr %arg0 sizes: [128                                                    0] : tensor<128x64x!tt.ptr<bf16>>
```

```
%0 = tt.get_program_id x : i32
%1 = arith.addi %arg3, %c127_i32 : i32
%2 = arith.divsi %1, %c128_i32 : i32
%3 = arith.addi %arg4, %c255_i32 : i32
%4 = arith.divsi %3, %c256_i32 : i32
%5 = arith.addi %arg5, %c63_i32 : i32
%6 = arith.divsi %5, %c64_i32 : i32
%7 = arith.muli %4, %c8_i32 : i32
%8 = arith.divsi %0, %7 : i32
%9 = arith.muli %8, %c8_i32 : i32
%10 = arith.subi %2, %9 : i32
%11 = arith.cmpi slt, %10, %c8_i32 : i32
%12 = arith.select %11, %10, %c8_i32 : i32
%13 = arith.remsi %0, %12 : i32
%14 = arith.addi %9, %13 : i32
%15 = arith.remsi %0, %7 : i32
%16 = arith.divsi %15, %12 : i32
%17 = arith.muli %14, %c128_i32 : i32
%18 = tt.make_range {end = 128 : i32, start = 0 : i32} : tensor<128xi32>
%19 = tt.splat %17 : (i32) -> tensor<128xi32>
%20 = arith.addi %19, %18 : tensor<128xi32>
%21 = arith.muli %16, %c256_i32 : i32
%22 = tt.make_range {end = 256 : i32, start = 0 : i32} : tensor<256xi32>
%23 = tt.splat %21 : (i32) -> tensor<256xi32>
%24 = arith.addi %23, %22 : tensor<256xi32>
%25 = tt.make_range {end = 64 : i32, start = 0 : i32} : tensor<64xi32>
%26 = tt.expand_dims %20 {axis = 1 : i32} : (tensor<128xi32>) -> tensor<128x1xi32>
%27 = tt.splat %arg6 : (i32) -> tensor<128x1xi32>
%28 = arith.muli %26, %27 : tensor<128x1xi32>
%29 = tt.expand_dims %25 {axis = 0 : i32} : (tensor<64xi32>) -> tensor<1x64xi32>
%30 = tt.splat %arg7 : (i32) -> tensor<1x64xi32>
%31 = arith.muli %29, %30 : tensor<1x64xi32>
%32 = tt.broadcast %28 : (tensor<128x1xi32>) -> tensor<128x64xi32>
%33 = tt.broadcast %31 : (tensor<1x64xi32>) -> tensor<128x64xi32>
%34 = arith.addi %32, %33 : tensor<128x64xi32>
%35 = tt.splat %arg0 : (!tt.ptr<bf16>) -> tensor<128x64x!tt.ptr<bf16>>
%36 = tt.addptr %35, %34 : tensor<128x64x!tt.ptr<bf16>>, tensor<128x64xi32>
%37 = tt.expand_dims %25 {axis = 1 : i32} : (tensor<64xi32>) -> tensor<64x1xi32>
%38 = tt.splat %arg8 : (i32) -> tensor<64x1xi32>
%39 = arith.muli %37, %38 : tensor<64x1xi32>
%40 = tt.expand_dims %24 {axis = 0 : i32} : (tensor<256xi32>) -> tensor<1x256xi32>
%41 = tt.splat %arg9 : (i32) -> tensor<1x256xi32>
%42 = arith.muli %40, %41 : tensor<1x256xi32>
%43 = tt.broadcast %39 : (tensor<64x1xi32>) -> tensor<64x256xi32>
%44 = tt.broadcast %42 : (tensor<1x256xi32>) -> tensor<64x256xi32>
%45 = arith.addi %43, %44 : tensor<64x256xi32>
%46 = tt.splat %arg1 : (!tt.ptr<bf16>) -> tensor<64x256x!tt.ptr<bf16>>
%47 = tt.addptr %46, %45 : tensor<64x256x!tt.ptr<bf16>>, tensor<64x256xi32>
```

Pointer Analysis: Convert-to-Memref

```
tts.make_tptr %arg1 to sizes: [1024], strides: [1], offsets: [%2] : tensor<1024x!tt.ptr<f32>>
```

```
memref.reinterpret_cast %arg0 to offset: [%1], sizes: [1024], strides: [1]
    : memref<*xf32> to memref<1024xf32, strided<[1], offset: ?>>
```

```
tts.make_tptr %arg1 sizes: [64, 256], strides: [%24, %25], offsets: [%arg15, %26] : tensor<64x256x!tt.ptr<bf16>>
tts.make_tptr %arg0 sizes: [128, 64], strides: [%21, %23], offsets: [%arg14, %c0] : tensor<128x64x!tt.ptr<bf16>>
```

```
memref.reinterpret_cast %arg0 to offset: [%54], sizes: [128, 64], strides: [%20, %22]
  : memref<*xbf16> to memref<128x64xbf16, strided<[?, ?], offset: ?>>
memref.reinterpret_cast %arg1 to offset: [%57], sizes: [64, 256], strides: [%23, %25]
 : memref<*xbf16> to memref<64x256xbf16, strided<[?, ?], offset: ?>>
```

Linalg Dialect

- MLIR built-in dialect
- Built-in lowering to other dialects
- The **value semantics** of linalg match Triton “blocks”
- Linalg provides a powerful way to do further transformations and optimizations (e.g., loop fusion and tiling)
- Built-in transformations -- use out-of-the-box or customize

Linalg Generic

```
%result = linalg.generic {
  indexing_maps = [affine_map<(i, j, k) -> (i, k)>,
                   affine_map<(i, j, k) -> (k, j)>,
                   affine_map<(i, j, k) -> (i, j)>],
  iterator_types = ["parallel", "parallel", "reduction"]
} ins(%lhs, %rhs : tensor<8x10xf32>,tensor<10x16xf32>)
  outs(%init :tensor<8x16xf32>) {
^bb0(%lhs_one: f32, %rhs_one: f32, %init_one: f32):
  %0 = arith.mulf %lhs_one, %rhs_one : f32
  %1 = arith.addf %init_one, %0 : f32
  linalg.yield %1 : f32
} -> tensor<8x16xf32>
```

Reference CPU Backend

- Example usage of Triton-Shared
- Used to validated Triton-Shared

- Currently supports x86-64
- Work ongoing to enable ARMv8 support

- Built using only built-in MLIR passes and dialects

- Proof-of-concept level functionality
  - Triton Tutorials + unit tests

How to Use Triton-Shared

Full directions at https://github.com/microsoft/triton-shared/blob/main/README.md#usage

- Stand-alone: includes Triton as a sub-module
- Includes triton-shared libs and CPU-reference back-end
- Reference triton-shared libs to use in *your* back-end

Contributing to Triton-Shared

https://github.com/microsoft/triton-shared

- We are accepting contributions small and large
- Looking for
  - Breadth coverage
  - Bug fixes
  - New functionality
  - QoL improvements