Modular

Mojo 🔥

A novel programming language for AI

Jack Clayton - ACAT 2024

Mojo🔥

AI Programming Status Quo

The three-world-problem leads to:

Fragmentation

Complexity

Rigid boundaries

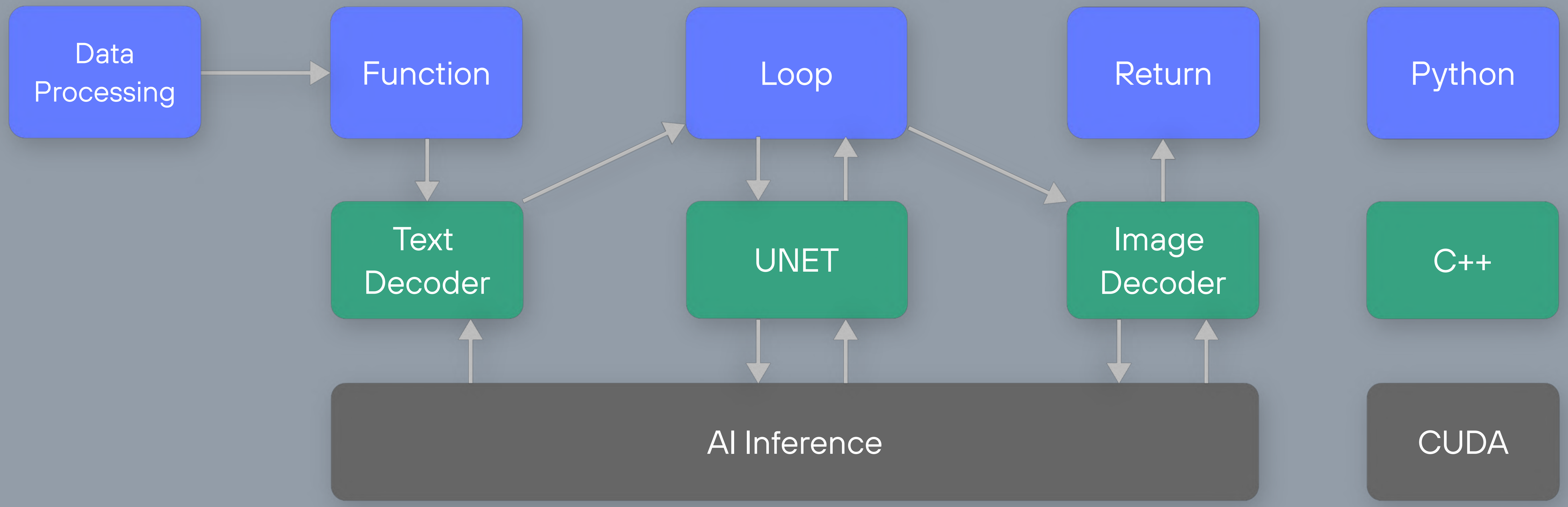
Model

System

Hardware

CUDA, OpenCL, ROCm

Stable Diffusion



Generality

Many common limitations…

- Dynamic shapes
- Sparsity
- Quantization
- Custom ops
- Embedded support
- Model coverage

AI is an end to end parallel compute problem, not just an accelerated matmul problem!

Full-stack AI R&D

Difficult to hire researchers & engineers ...

- ... who have AI modeling experience, and
- ... who know exotic numerics, and
- ... who know specialized Hardware details

AI (model) research cannot rely on: "compiler engineer in-the-loop"!

Thick layers of magic **between** what you write and what gets run

e.g. graph capture, torch script, torch inductor, Triton-Lang, backends, ONNX export, graph runtimes...

Build a new language!

Only way to deliver the *best quality result*

- AI developers are really important to the world
- We're tired of point solutions, research-quality tools, flashy demos that don't generalize.

However, this requires:

- ✔ Consistent vision
- ✔ Long term commitment
- ✔ Funding for the development
- ✔ Ability to attract specialized talent
- ✔ Big target market of developers

We have done this before:

OpenCL™ Clang Swift

Modular

Chris Lattner
Co-Founder & CEO

Tim Davis
Co-Founder & President

… and a world-class team of engineers with previous experience from Google, Meta, Apple, Microsoft, NVIDIA, Tesla, Intel, AMD, Graphcore, SambaNova, Tenstorrent, Tesla, working on MLIR, Clang, LLVM, PyTorch, ONNX, TensorFlow, etc.

Mojo🔥

Superpowers

01

Programmability

Superset of Python

Modern language features

Meta-programming

High quality development tools

02

Performance

System-level control of implementation

Zero-cost abstractions

Auto-tuning

Exponential speedup over Python

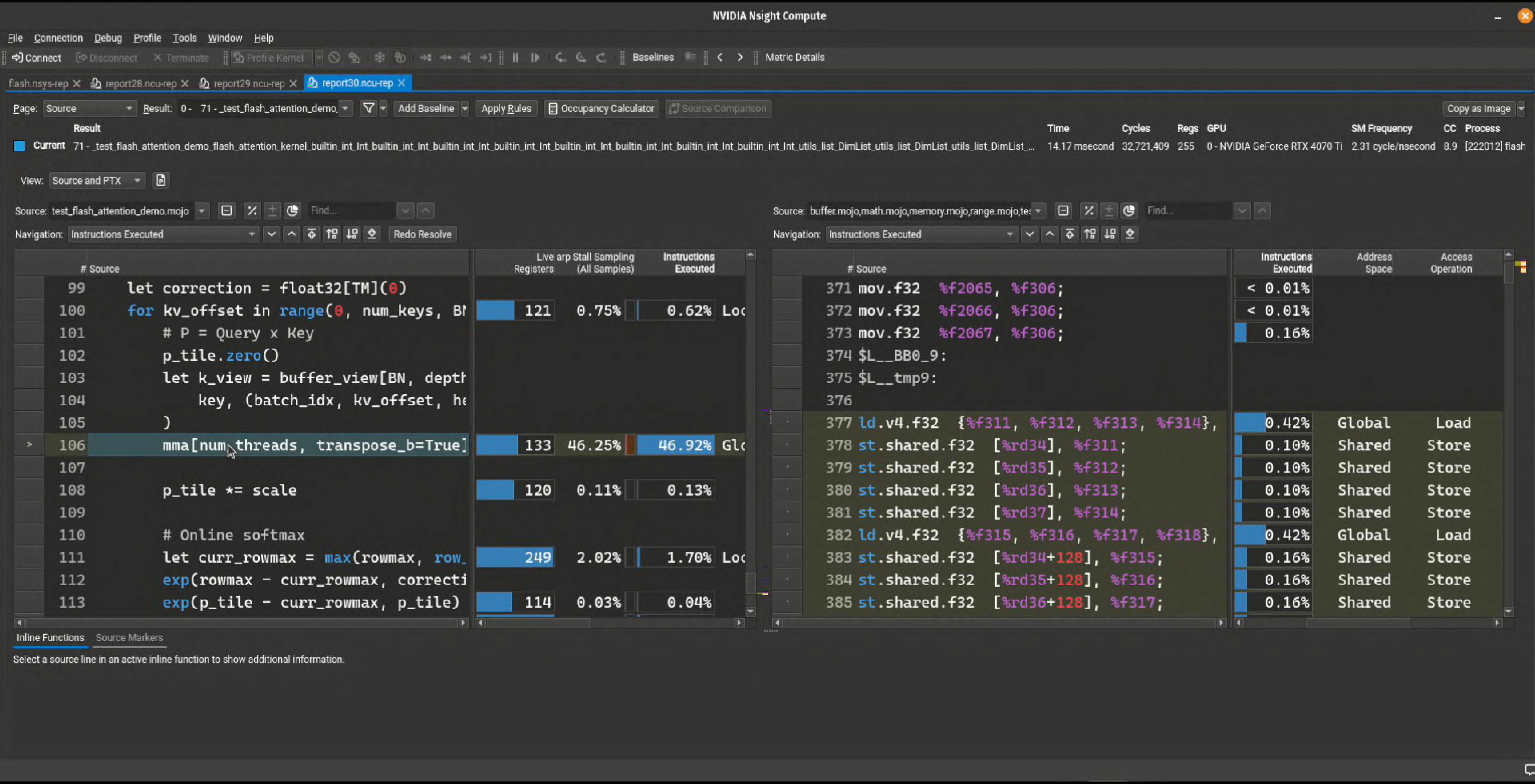
03

Portability

Host / device code in the same file

Same code works on CPU and GPU

Integrates with **any accelerator** that supports LLVM/MLIR


NVIDIA Nsight Compute
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel Baselines Metric Details
flash.nsys-rep report28.ncu-rep report29.ncu-rep report30.ncu-rep
Page: Source Result: 0 - 71 - _test_flash_attention_demo_ Add Baseline Apply Rules Occupancy Calculator Source Comparison Copy as Image
Result Time Cycles Regs GPU SM Frequency CC Process
Current 71 - _test_flash_attention_demo_flash_attention_kernel_builtin_int_Int_builtin_int_Int_builtin_int_Int_builtin_int_Int_builtin_int_Int_builtin_int_Int_builtin_int_Int_utils_list_DimList_utils_list_DimList_utils_list_DimList_... 14.17 msecond 32,721,409 255 0 - NVIDIA GeForce RTX 4070 Ti 2.31 cycle/nsecond 8.9 [222012] flash
View: Source and PTX
Source: test_flash_attention_demo.mojo
Navigation: Instructions Executed Redo Resolve
Source Live Registers Warp Stall Sampling (All Samples) Instructions Executed
99 let correction = float32[TM](0)
100 for kv_offset in range(0, num_keys, BN 121 0.75% 0.62% Loc
101 # P = Query x Key
102 p_tile.zero()
103 let k_view = buffer_view[BN, deptl
104 key, (batch_idx, kv_offset, he
105)
106 mma[num_threads, transpose_b=True] 133 46.25% 46.92% Glo
107
108 p_tile *= scale 120 0.11% 0.13%
109
110 # Online softmax
111 let curr_rowmax = max(rowmax, row_ 249 2.02% 1.70% Loc
112 exp(rowmax - curr_rowmax, correcti
113 exp(p_tile - curr_rowmax, p_tile) 114 0.03% 0.04%
Source: buffer.mojo,math.mojo,memory.mojo,range.mojo,te
Navigation: Instructions Executed
Source Instructions Executed Address Space Access Operation
371 mov.f32 %f2065, %f306; < 0.01%
372 mov.f32 %f2066, %f306; < 0.01%
373 mov.f32 %f2067, %f306; 0.16%
374 $L__BB0_9:
375 $L__tmp9:
376
377 ld.v4.f32 {%f311, %f312, %f313, %f314}, 0.42% Global Load
378 st.shared.f32 [%rd34], %f311; 0.10% Shared Store
379 st.shared.f32 [%rd35], %f312; 0.10% Shared Store
380 st.shared.f32 [%rd36], %f313; 0.10% Shared Store
381 st.shared.f32 [%rd37], %f314; 0.10% Shared Store
382 ld.v4.f32 {%f315, %f316, %f317, %f318}, 0.42% Global Load
383 st.shared.f32 [%rd34+128], %f315; 0.16% Shared Store
384 st.shared.f32 [%rd35+128], %f316; 0.16% Shared Store
385 st.shared.f32 [%rd36+128], %f317; 0.16% Shared Store
Inline Functions Source Markers
Select a source line in an active inline function to show additional information.

MAX

powered by Mojo🔥

Engine APIs

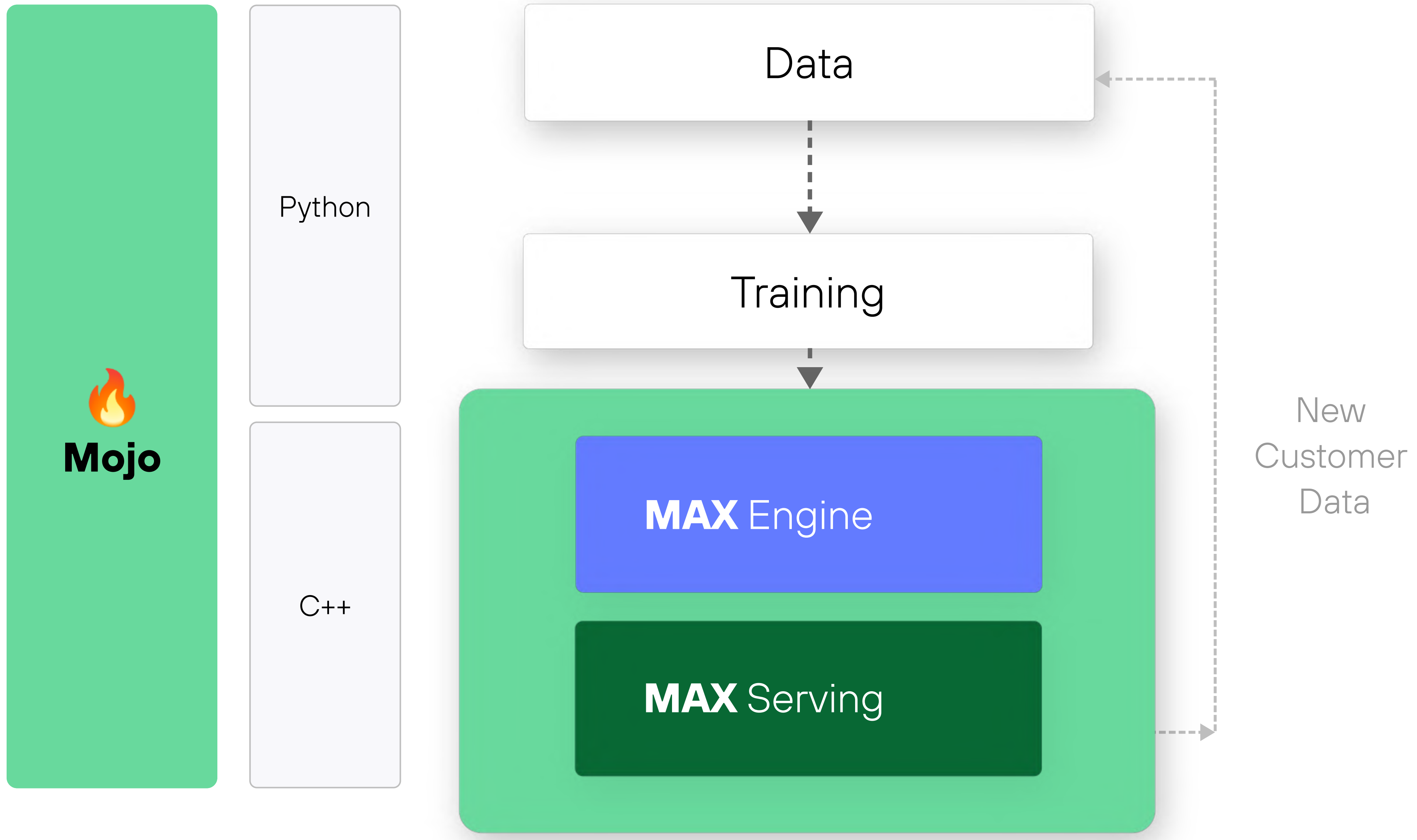
Graph APIs

CPU kernels

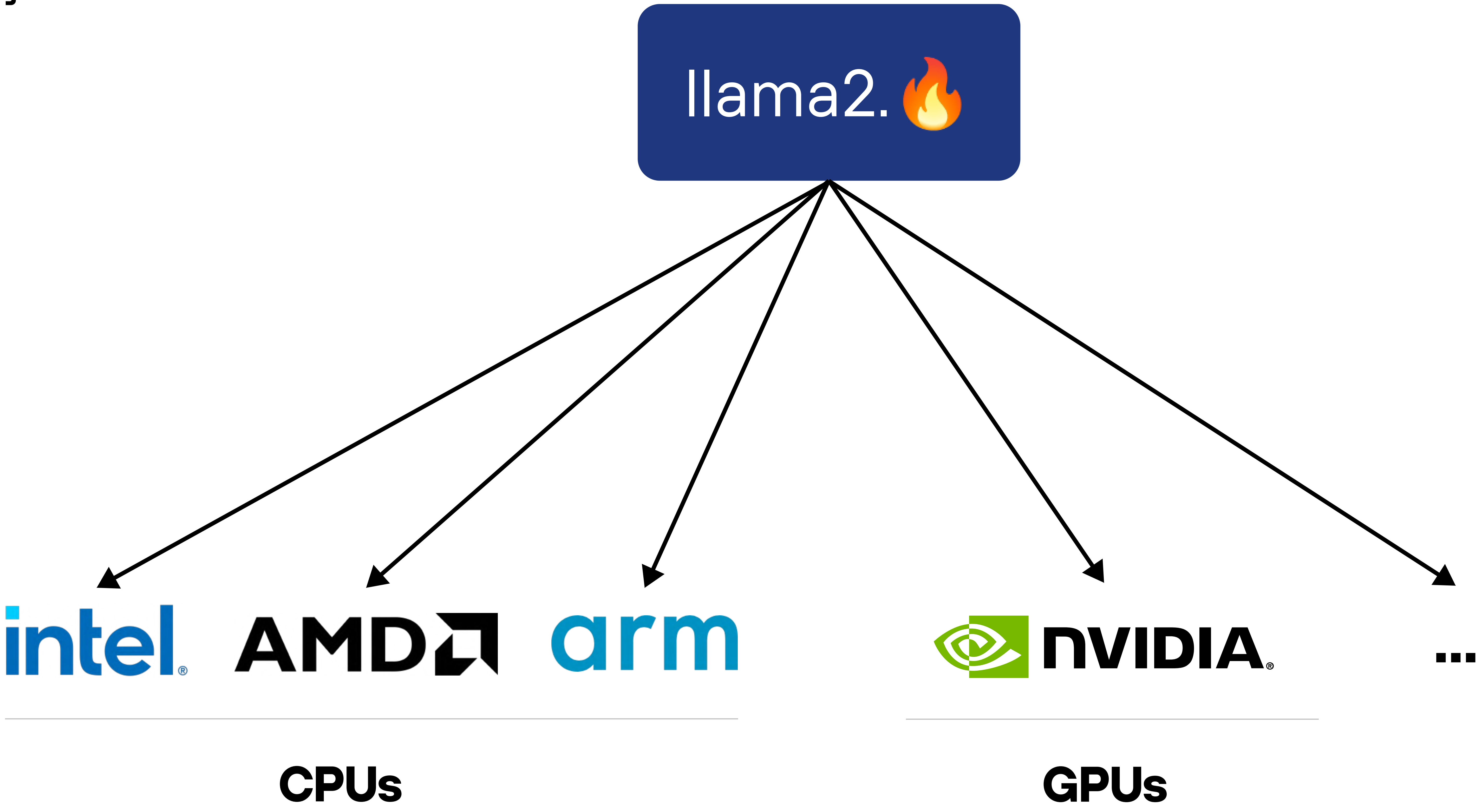
GPU kernels

Custom ops

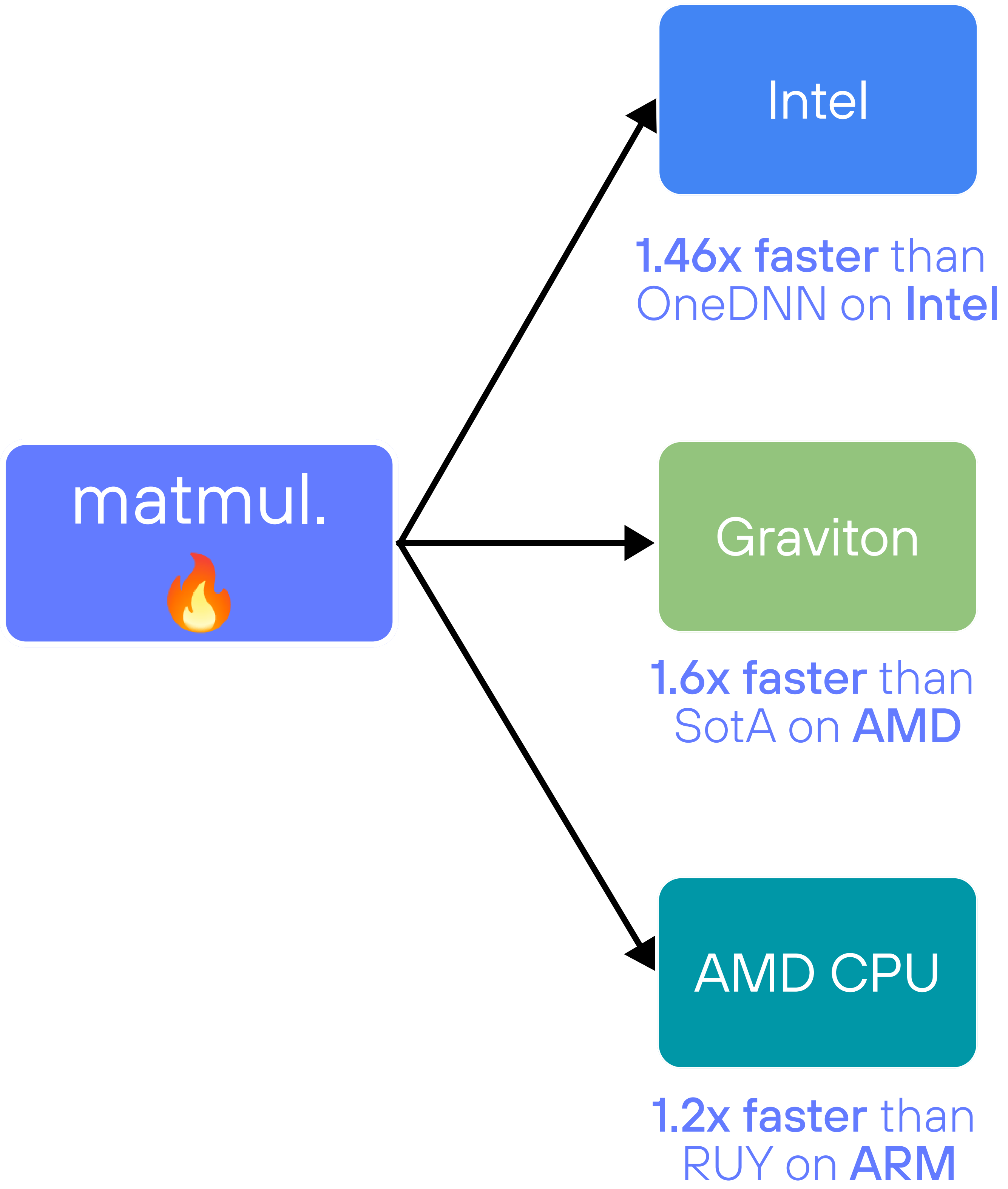
Op registration

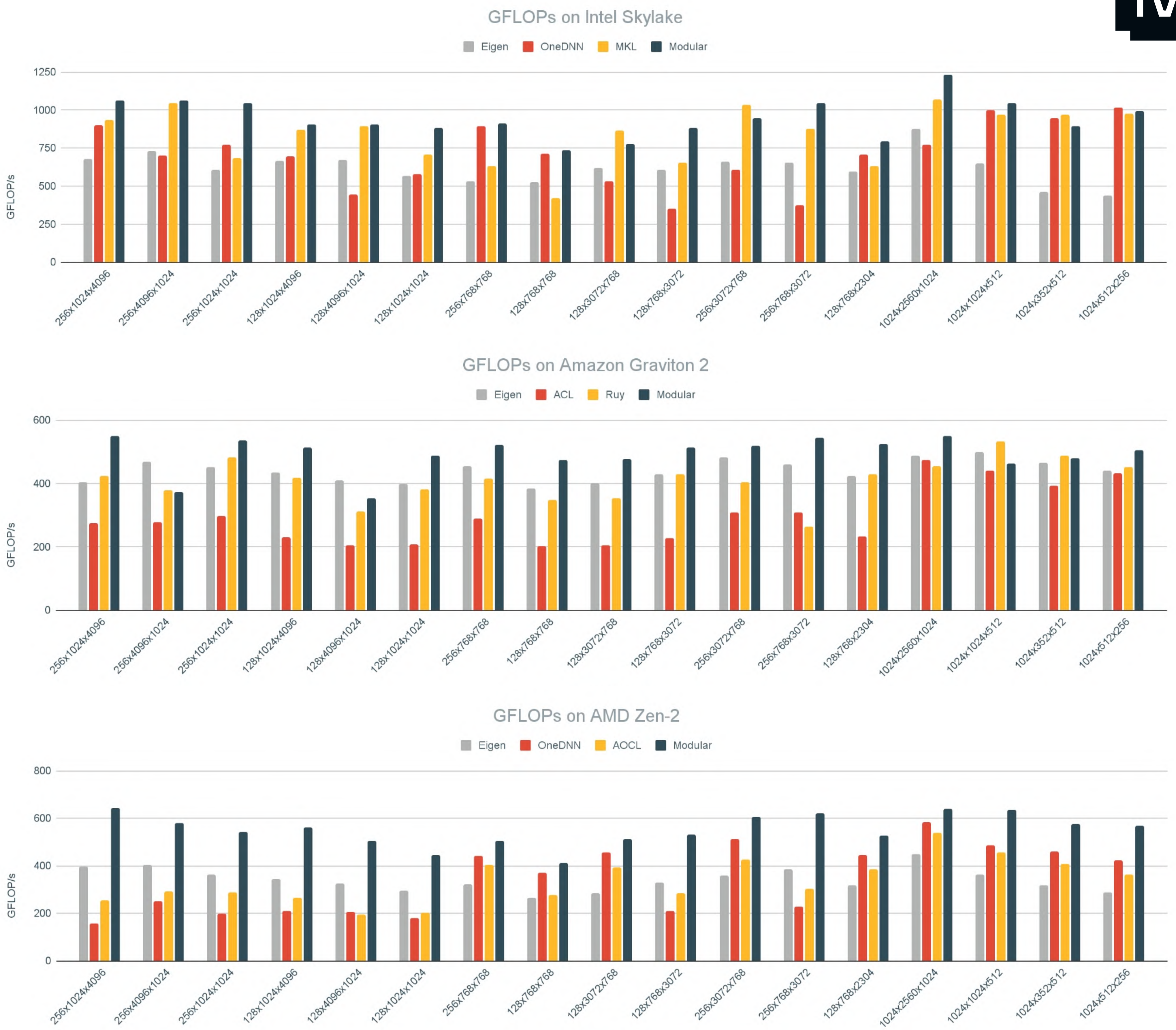


Mojo🔥



Mojo🔥


matmul.🔥
Intel
1.46x faster than
OneDNN on Intel
Graviton
1.6x faster than
SotA on AMD
AMD CPU
1.2x faster than
RUY on ARM


GFLOPs on Intel Skylake
Eigen
OneDNN
MKL
Modular
GFLOP/s
GFLOPs on Amazon Graviton 2
Eigen
ACL
Ruy
Modular
GFLOP/s
GFLOPs on AMD Zen-2
Eigen
OneDNN
AOCL
Modular
GFLOP/s
256x1024x4096
256x4096x1024
256x1024x1024
128x1024x4096
128x4096x1024
128x1024x1024
256x768x768
128x768x768
128x3072x768
128x768x3072
256x3072x768
256x768x3072
128x768x2304
1024x2560x1024
1024x1024x512
1024x352x512
1024x512x256

Mojo 🔥

Deep Dive

Design approach, new features, and what's coming next

Mojo🔥

In a nutshell

Designed for AI developers

- but not limited to AI

Combines best of Python and C

- Python syntax and ecosystem
- System-level control and performance

State of the art

- Adopts best known techniques
- Built from scratch using MLIR

```
def bpe_encode(inout tokens: List[Int],
               text: String, inout tok: Tokenizer):
    for c in text:
        tokens.append(tok.find(c))
    while True:
        ...
        for i in range(len(tokens) - 1):
            let str = (tok.vocab[tokens[i]] +
                       tok.vocab[tokens[i + 1]])
            let id = tok.find(str)
            if id != -1 and tok.scores[id] > best_score:
                best_score = tok.scores[id]
                best_id, best_idx = (id, i)

        if best_idx == -1:
            break

        tokens[best_idx] = best_id
        del tokens[best_idx + 1]
```

from https://github.com/tairov/llama2.mojo/blob/master/llama2.mojo

Mojo🔥

Performance Generality on CPU 🏎️

```
...
elif is_x86():
    ...
elif type.is_floating_point():
    return llvm_intrinsic[
        "llvm.vector.reduce.fmax"](self)
elif type.is_unsigned():
    return llvm_intrinsic[
        "llvm.vector.reduce.umax"](self)
else:
    return llvm_intrinsic[
        "llvm.vector.reduce.smax"](self)
```

```
fn vectorize[
    simd_width: Int,
    func: fn[width: Int](Int) -> None
](size: Int):
    let simd_end = (size // simd_width) * simd_width
    # Process a simd_width at a time.
    for i in range(0, simd_end, simd_width):
        func[simd_width](i)

    # Handle left-over elements with scalars.
    for i in range(simd_end, size):
        func[1](i)
```

MLIR + Metaprogramming = 🔥

Mojo🔥

Metaprogramming

Mojo🔥

Parameters

Underpin performance generality

Single-source-of-truth parametric kernels

Specialized and optimized when parameters are known

Encapsulated in types

Guaranteed codegen behaviour

```
struct Tensor[
    type: DType,
    static_shape: DimList = DimList(),
    static_strides: DimList = DimList(),
    input_lambda: Optional[
        fn[
            w: Int, t: DType, v: DimList
        ] (IntList[v]) capturing -> SIMD[t, w]
    ] = None,
    output_lambda: Optional[
        fn[
            w: Int, t: DType, v: DimList
        ] (IntList[v], SIMD[t, w]) capturing
          -> None
    ] = None,
]:
    ...
```

Function argument type parameters must be explicit on function signature!

```
fn kernel[
    type: DType,
    shape: DimList,
    strides: DimList,
    input_lambda: Optional[
        fn[
            w: Int, t: DType, v: DimList
        ] (IntList[v]) capturing -> SIMD[t, v]
    ],
    output_lambda: Optional[
        fn[
            w: Int, t: DType, v: DimList
        ] (IntList[v], SIMD[t, w]) capturing -> None
    ],
](x: Tensor[type, shape, strides,
            input_lambda, output_lambda]):
    pass
```

Now, add more input tensors!

```
fn kernel(x: Tensor):
    pass
```

See MAX Engine Extensibility talk!

Mojo🔥

Autoparameterization

Make parameters feel first-class

Allow partially bound types
Infer type parameters contextually

Access static type info from values

```
fn kernel(
    lhs: Tensor, rhs: Tensor[lhs.type]
) -> Tensor[lhs.type]:
    alias dim = lhs.static_shape[0]
    ...

let lhs = Tensor[DType.float32](…)
let rhs = Tensor[lhs.type](…)
print(kernel(lhs, rhs))
```

Mojo🔥

Partially Bound Types

Types with none or some of its parameters unspecified

- Explicitly unbind with discard pattern

Always prefer inferring parameters from context over defaults

- Local inference only: function calls, initializer expressions, etc.

```
fn foo[dt: DType](
    x: SIMD[dt], y: SIMD
):
    let result: SIMD[dt] = x + y

fn foo[dt: DType](
    x: SIMD[dt, _], y: SIMD[_, _]
):
    let result: SIMD[dt, _] = x + y

fn foo[
    dt: DType, x0: Int,
    y0: DType, y1: Int
](x: SIMD[dt, x0], y: SIMD[y0, y1]):
    let result: SIMD[
        dt, type(x+y).size
    ] = x + y
```

Mojo🔥

Generic Programming

Mojo🔥

Traits!! 🎉

Traits declare a type prototype

- The language knows what functions a type definitely has
- We can codegen generics!

Tablestakes feature

🚧 Still more features to come 🚧

```
trait TensorLike:
    def transpose(self) -> Self: ...
    def __mul__(self, rhs: Self) -> Self: ...

struct Matrix(TensorLike):
    def transpose(self) -> Self:
    def __mul__(self, rhs: Self) -> Self:

struct SparseTensor(TensorLike):
    def transpose(self) -> Self:
    def __mul__(self, rhs: Self) -> Self:

def tensor_square[T: TensorLike](a: T) -> T:
    return a.transpose() * a

tensor_square(Matrix())
tensor_square(SparseTensor())
```

Mojo🔥

Generic Functions

Trait functions

- The declared functions of a trait can be invoked on any instance of it

```
trait Sized:
    fn __len__(self) -> Int: ...

fn len[T: Sized](value: T) -> Int:
    return value.__len__()

trait Stringable:
    fn __str__(self) -> String: ...

fn str[T: Stringable](
    value: T
) -> String:
    return value.__str__()
```

Mojo🔥

Zero-Cost Generics

Types can be

- Register-passable (don't need an address)
- Memory-only (always have an address)

Mojo guarantees argument conventions of register-passable types

- Passing in register for performance
- Avoid C++ performance issues with generics

```
template <typename T>
T add(const T &lhs, const T &rhs) {
  return lhs + rhs;
}
```

```
fn add[T: Addable](
    lhs: T, rhs: T
) -> T:
    return lhs + rhs
```

Mojo🔥

Trait Inheritance

Type hierarchies in Mojo!

Traits can inherit from other traits

- Structs that implement child traits must conform to parent traits
- Instances of child traits can be passed as instances of parent trait

```
trait Movable:
    fn __moveinit__(
        inout self,
        owned existing: Self): ...

trait Copyable:
    fn __copyinit__(
        inout self,
        existing: Self): ...

trait Value(Movable, Copyable): pass

fn make_copy[T: Value](x: T) -> T:
    var copy = x # invokes __copyinit__
    return copy^ # invokes __moveinit__
```

Mojo🔥

Generic Collections

All generic types are destructible

- Allows defining proper generic collections

```
struct DynamicVector[T: Value]:
    fn __del__(owned self):
        for i in range(self.size):
            self[i].__del__()
        self.data.free()

    fn append(inout self, owned v: T):
        self._resize()
        (self.data + self.size).emplace(v^)
```

Check out Jack's new blogpost for more!

Mojo🔥

GPU Support

Mojo🔥

Extending to GPUs

The same approach for CPU scales to GPU

- Direct GPU hardware access via LLVM intrinsics

Other ways to access hardware

- NVVM Dialect operations in MLIR
- Inline PTX assembly

```
@always_inline
fn async_copy[
    size: Int, type: AnyRegType
](
    src: Pointer[type, AddressSpace.GLOBAL],
    dst: Pointer[type, AddressSpace.SHARED]
):
    constrained[size in [4, 8, 16]]()
    @parameter
    if size == 4:
        llvm_intrinsic[
            "llvm.nvvm.cp.async.ca.shared.global.4"
        ](dst, src)
    elif size == 8:
        llvm_intrinsic[
            "llvm.nvvm.cp.async.ca.shared.global.8"
        ](dst, src)
    else:
        llvm_intrinsic[
            "llvm.nvvm.cp.async.ca.shared.global.16"
        ](dst, src)
```

Mojo🔥

Codegen isn't enough!

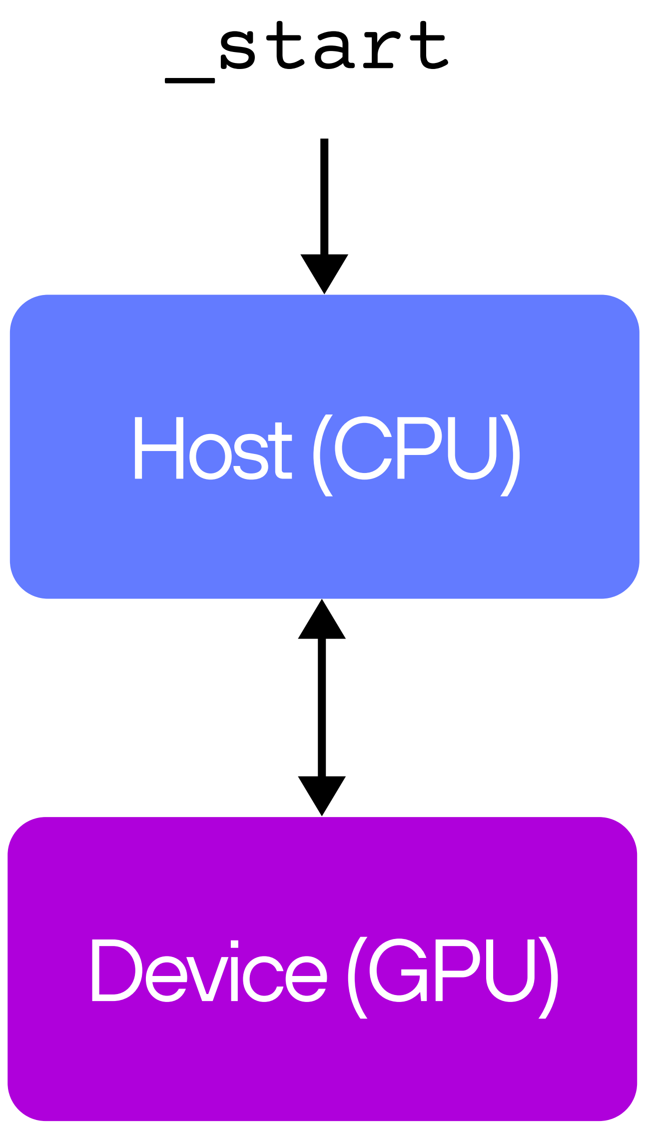
Mojo 🔥 can generate GPU code
But it must be driven by a host CPU program

Existing approaches
OpenGL / Vulkan/OpenCL
CUDA

What unified programming looks like
Share code between CPU and GPU
Offload vendor-specific logic to the library

Mojo🔥

OpenGL and GLSL



vertex.glsl

```
#version 330 core
layout (location = 0)
in vec3 aPos;

out vec4 vertexColor;

void main() {
  gl_Position = vec4(
    aPos, 1.0);
  vertexColor = vec4(
    0.5, 0.0, 0.0, 1.0);
}
```

frag.glsl

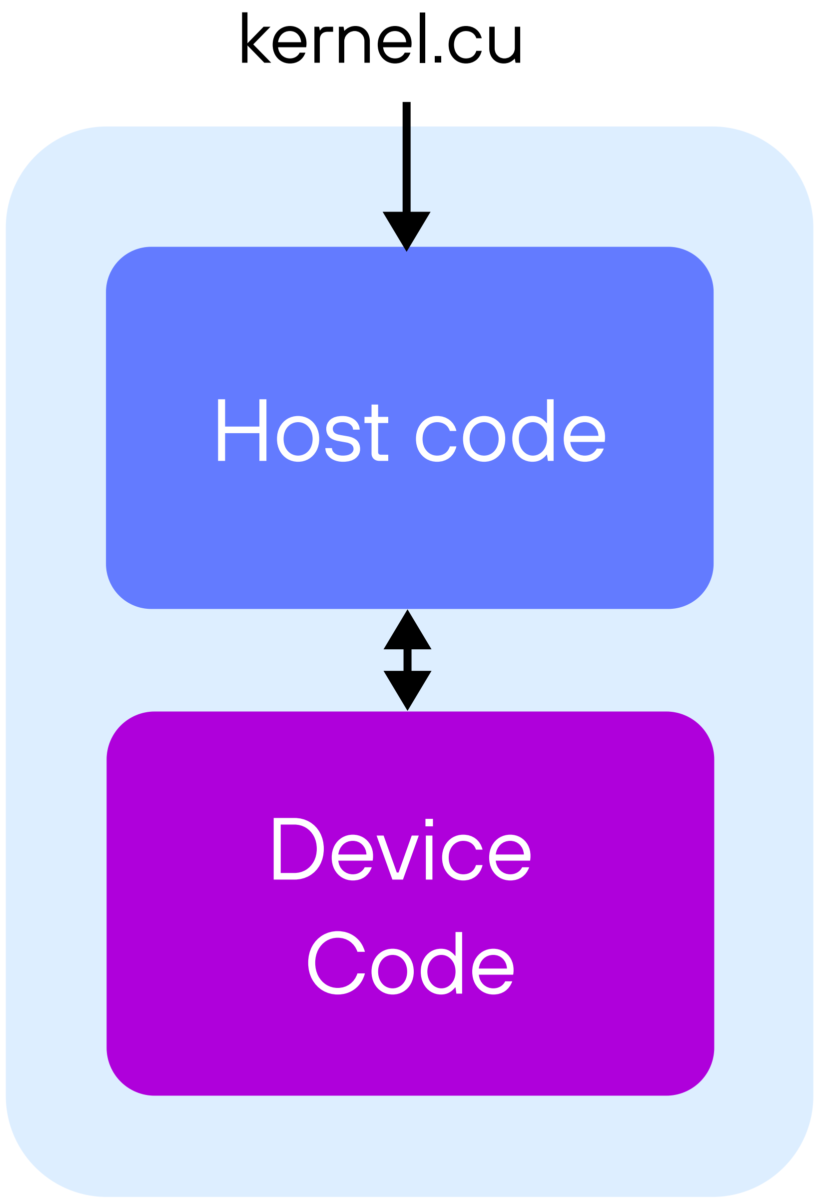
```
#version 330 core
out vec4 FragColor;

in vec4 vertexColor;

void main() {
  FragColor = vertexColor;
}
```

host.cpp

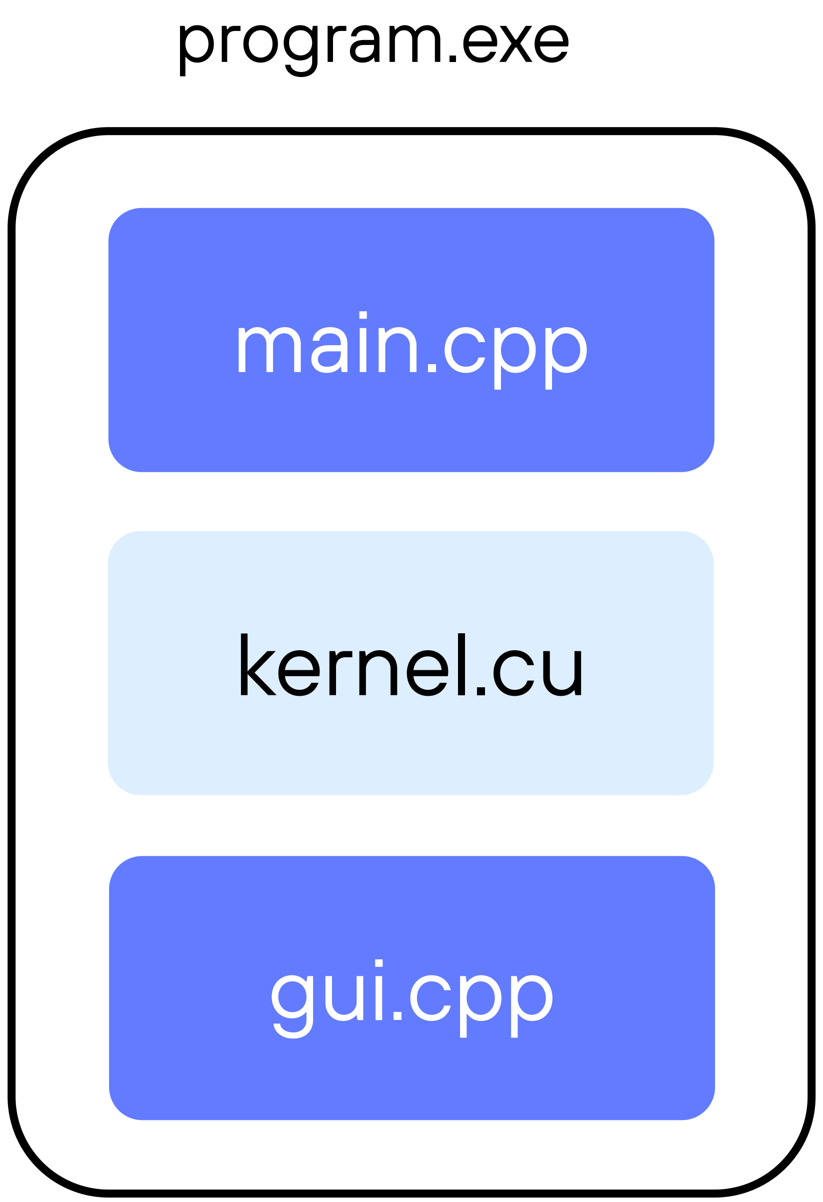
```
GLChar *vertex;
GLChar *frag;
...
glUseProgram(0);
```

Mojo🔥

CUDA — Unified Host and Device Program

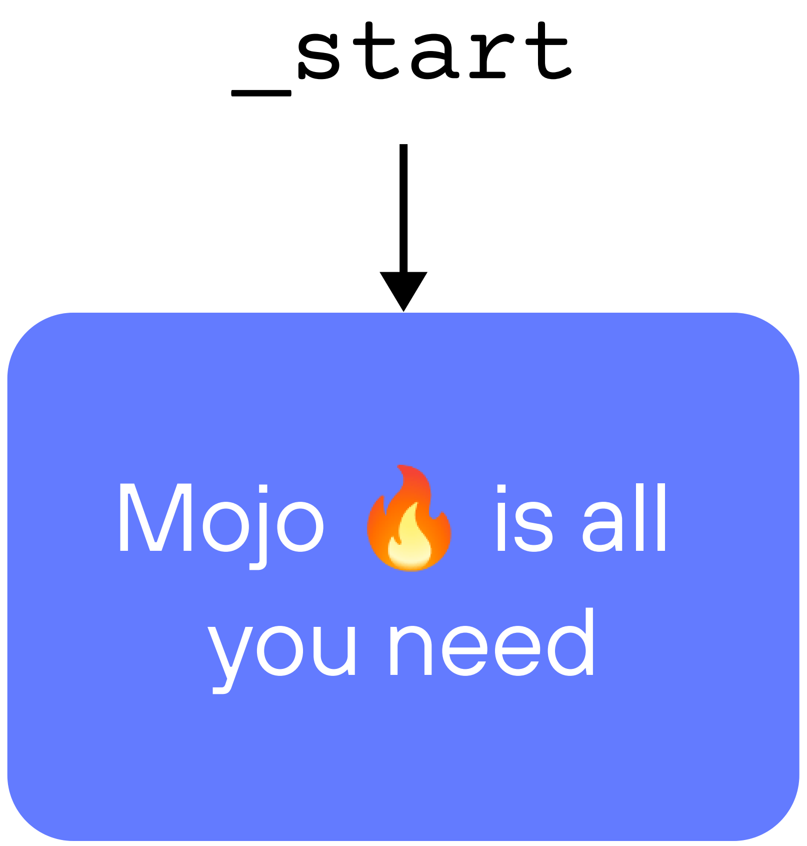



```
__global__ void cuda_hello() {
  printf("Hello from GPU!\n");
}

int main() {
  cuda_hello<<<1, 1>>>();
}
```



Mojo 🔥

Unified Programming



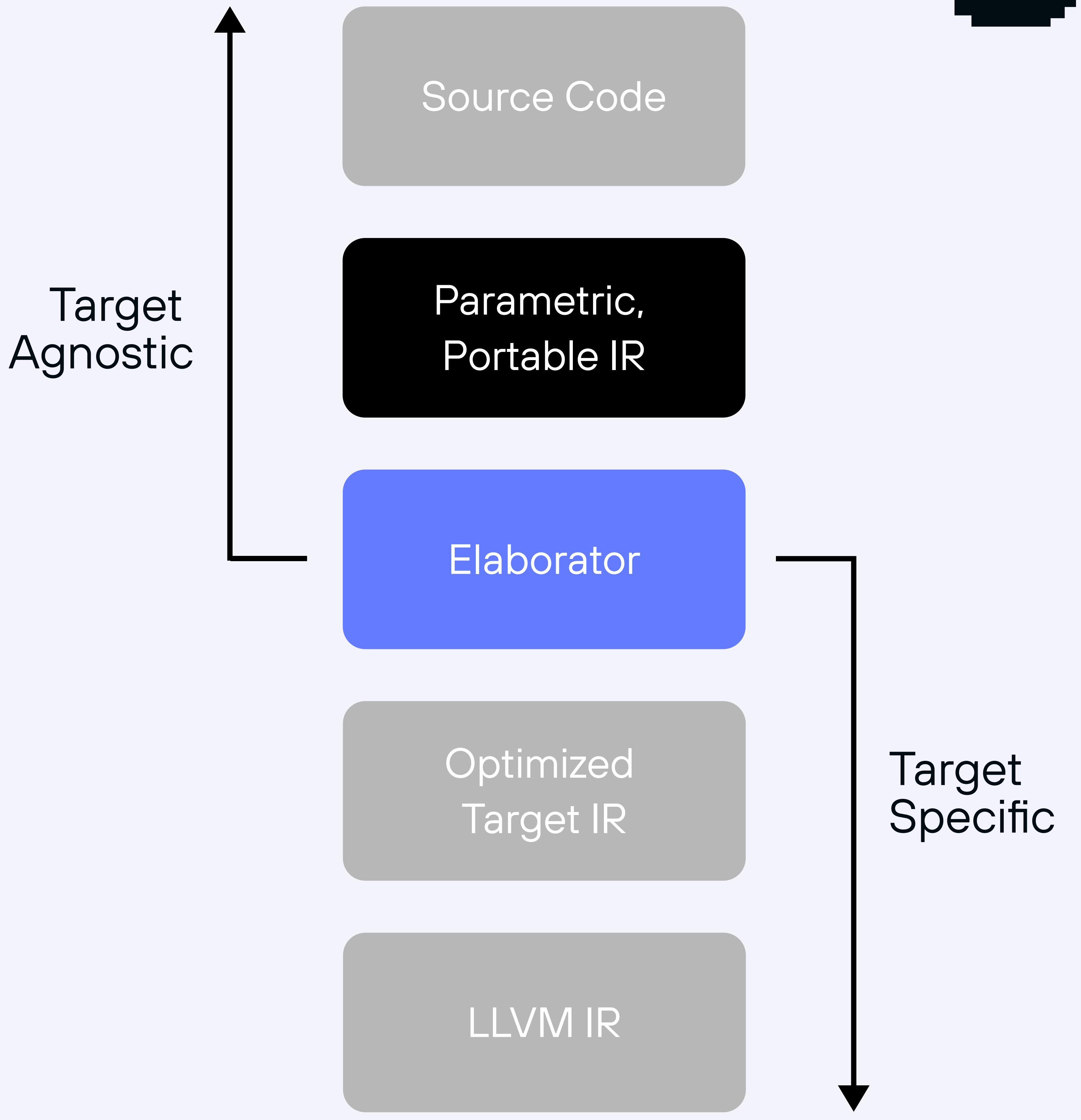

```
import gpu

fn say_hi(value: Int):
    print("hello", value)

def main():
    say_hi(42)
    let f = gpu.Function[say_hi]()
    f(11, grid_dim=1, block_dim=1)
```

Same function used on CPU and GPU!

Orchestration logic implemented in the `gpu` package

Mojo🔥
Compiler Technology

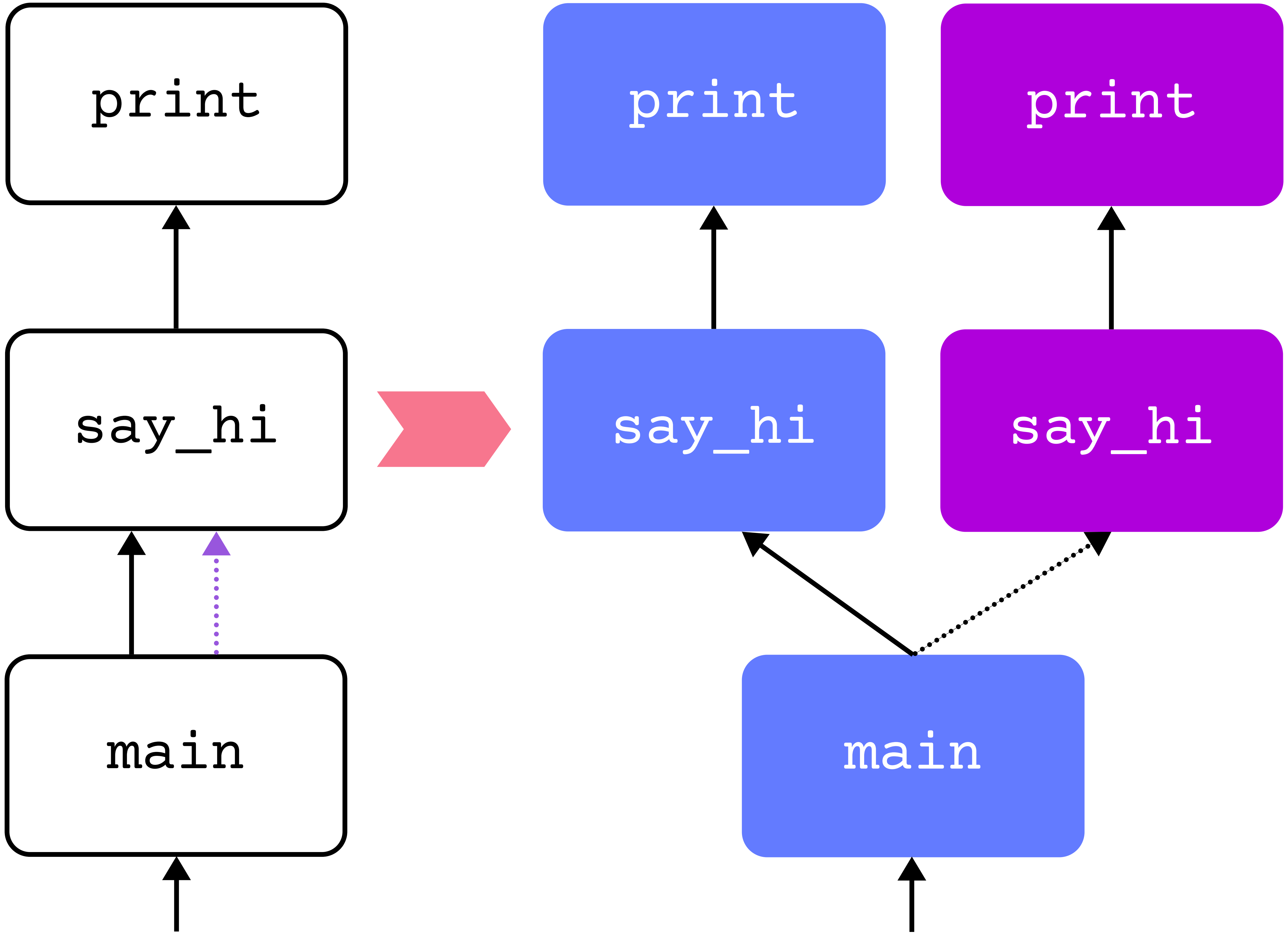
- **Target-agnostic IR**
  - Portable serialized code
- **JIT compiler**
  - Autotuning
- **Elaborator**
  - Parametric function instantiation



Mojo🔥

Split compilation

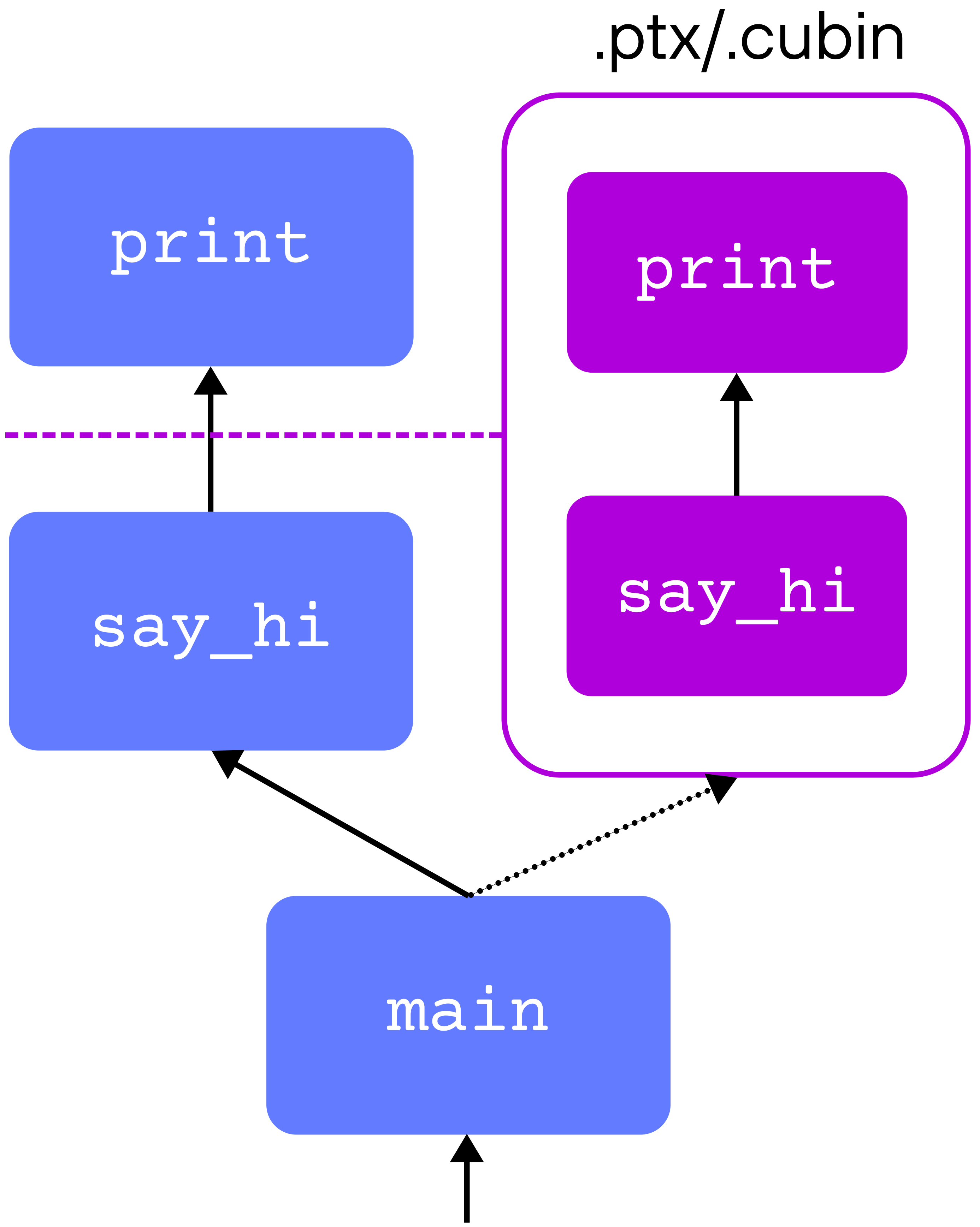
```
fn say_hi(value: Int):
    print("hello", value)

def main():
    say_hi(42)

    let f = gpu.Function[say_hi]()
    f(11, grid_dim=1, block_dim=1)
```



Mojo🔥

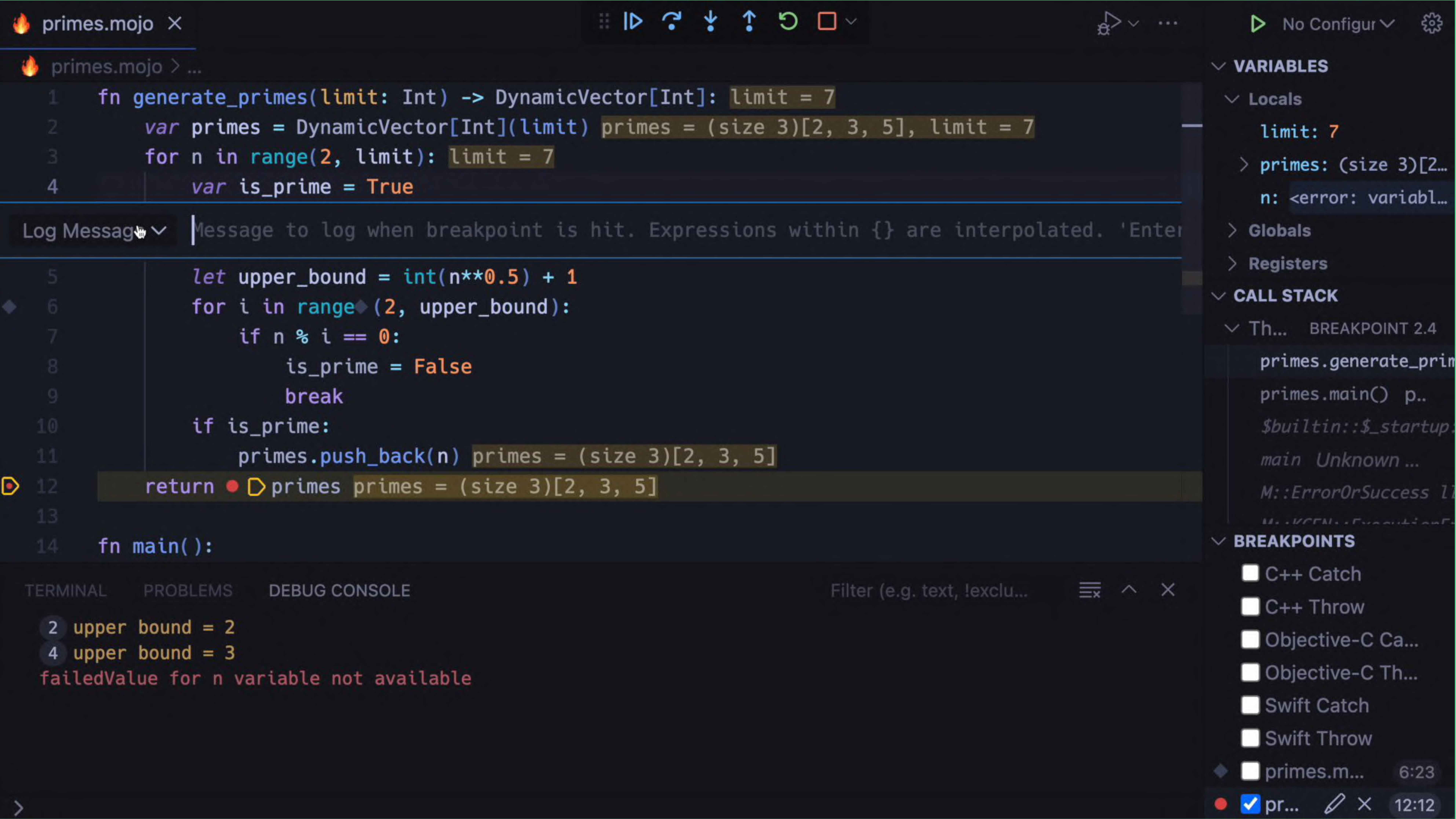
Split compilation

```
fn say_hi(value: Int):
    print("hello", value)

def main():
    say_hi(42)

    alias say_hi_ptx = (
        ".visible .entry say_hi() { ..."
    )
    let f = gpu.Function[say_hi_ptx]()
    f(11, grid_dim=1, block_dim=1)
```



Mojo🔥

Leading edge developer tooling


primes.mojo
No Configur
VARIABLES
Locals
limit: 7
primes: (size 3)[2…
n: <error: variabl…
Globals
Registers
CALL STACK
Th… BREAKPOINT 2.4
primes.generate_prim
primes.main() p..
$builtin::$_startup
main Unknown …
M::ErrorOrSuccess
BREAKPOINTS
C++ Catch
C++ Throw
Objective-C Ca…
Objective-C Th…
Swift Catch
Swift Throw
primes.m… 6:23
pr… 12:12
fn generate_primes(limit: Int) -> DynamicVector[Int]: limit = 7
var primes = DynamicVector[Int](limit) primes = (size 3)[2, 3, 5], limit = 7
for n in range(2, limit): limit = 7
var is_prime = True
Log Message
Message to log when breakpoint is hit. Expressions within {} are interpolated. 'Enter
let upper_bound = int(n**0.5) + 1
for i in range(2, upper_bound):
if n % i == 0:
is_prime = False
break
if is_prime:
primes.push_back(n) primes = (size 3)[2, 3, 5]
return primes primes = (size 3)[2, 3, 5]
fn main():
TERMINAL
PROBLEMS
DEBUG CONSOLE
Filter (e.g. text, !exclu…
2 upper bound = 2
4 upper bound = 3
failedValue for n variable not available

Mojo🔥

What's next?

Mojo🔥

Coming soon ...

- Debugger
- Lifetimes
- Closures
- Testing framework
- Even better traits
- More features from Python
- Project format
- Heterogenous variadics

https://docs.modular.com/mojo/roadmap.html

Mojo🔥
Open Source Journey

Today
- Open Sourcing Mojo documentation
- LLVM Apache 2.0 License
- Examples/Notebooks already there (~35PRs)

Q1 2024
- Open sourcing Mojo Standard Library
- CI and other tooling to facilitate contributions
- Nightly builds to support external development

Accepting Contributions

https://github.com/modularml/mojo

Mojo🔥

Additional resources:

- Get started with downloading **Mojo**: https://developer.modular.com/
- Join our **Discord** community: https://discord.gg/modular
- Read and subscribe to **Modverse Newsletter**: https://www.modular.com/newsletters
- Read **Mojo blog** posts: https://www.modular.com/blog
- Watch **developer videos** and past live streams: https://www.youtube.com/@modularinc/videos
- Report feedback, including issues on our **GitHub** tracker: https://github.com/modularml/mojo/issues

Thank you!