



CONFERENCE

—NORTH AMERICA 2026—

Speeding up torch.compile... A new FakeTensor

Angel Li, Meta

What is a FakeTensor?

Same metadata: shape (1024, 1024) dtype float32 device cuda:0 stride (1024, 1)

Real tensor

4 MB



runs real kernels

writes data into inputs

`x.add_(y)` overwrites x's data

Memory

Compute

Side effects

In-place ops

FakeTensor

0 bytes

computes metadata only

real tensors are never touched

`x.add_(y)` only updates the fake copy

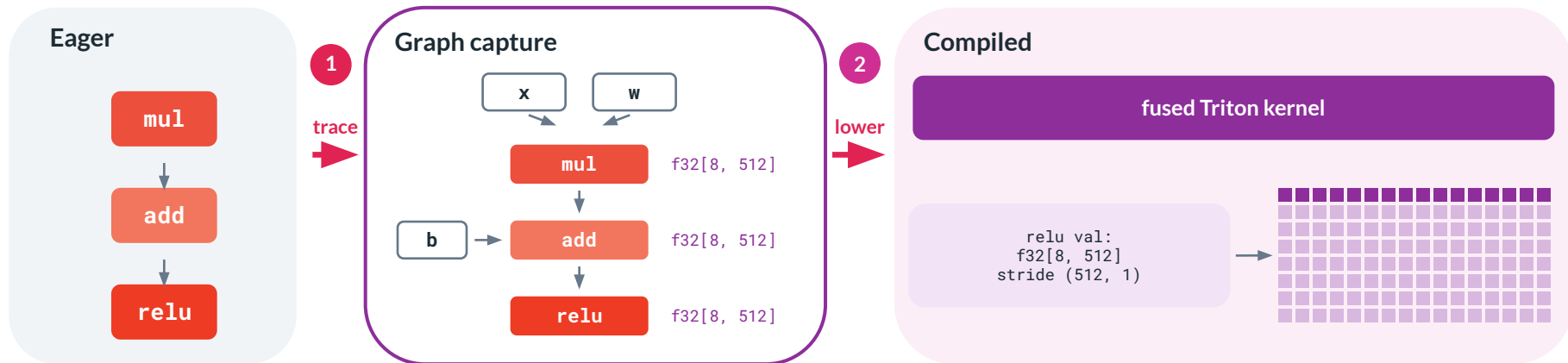
FakeTensor in torch.compile

A compiler has to know what the output metadata of ops are without actually running the op!

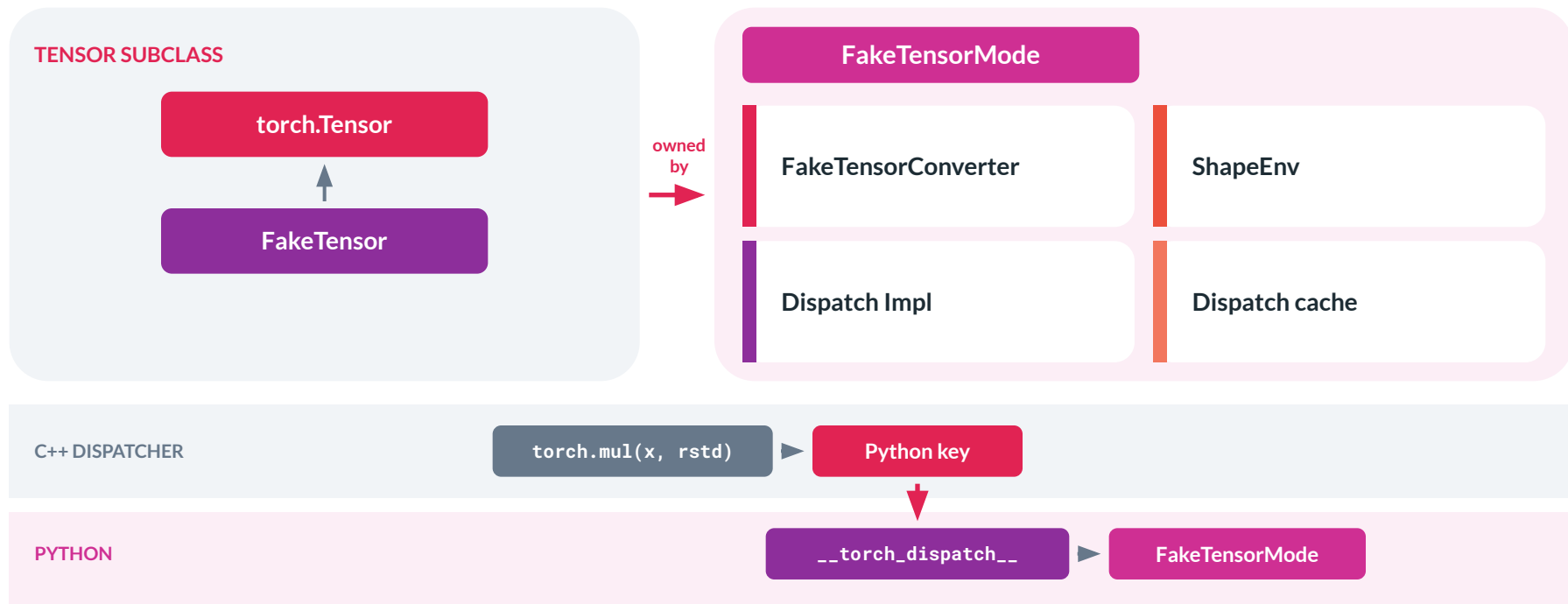
- 1 **TorchDynamo**: Traces user code into an FX graph. How does it resolve data-dependent shapes, branch on code? **FakeTensors!**
- 2 **TorchInductor**: Generates optimized fused kernels and plans memory. How does it know how much memory is needed? **FakeTensors!**
- 3 What if we want more generic graphs? We can use **symbolic shapes** with **FakeTensors!**

x: (8, 2048, 4096)

x: (s0, s1, 4096)



FakeTensor Implementation (today...)



FakeTensorConverter

FakeTensorConverter

- Graph inputs are passed in as real tensors
- Need to convert real tensor $\rightarrow$ FakeTensor



FakeTensorMode

FakeTensorConverter

ShapeEnv

Dispatch Impl

Dispatch cache

ShapeEnv

ShapeEnv

- Owns the symbolic shape info: symbols, hints, ranges and guards
- One ShapeEnv per compile, shared by every SymNode in it
- One SymNode per SymInt: holds its sympy expression (e.g. $2*s0$) and hint (16), and points to the ShapeEnv

FakeTensorMode

FakeTensorConverter

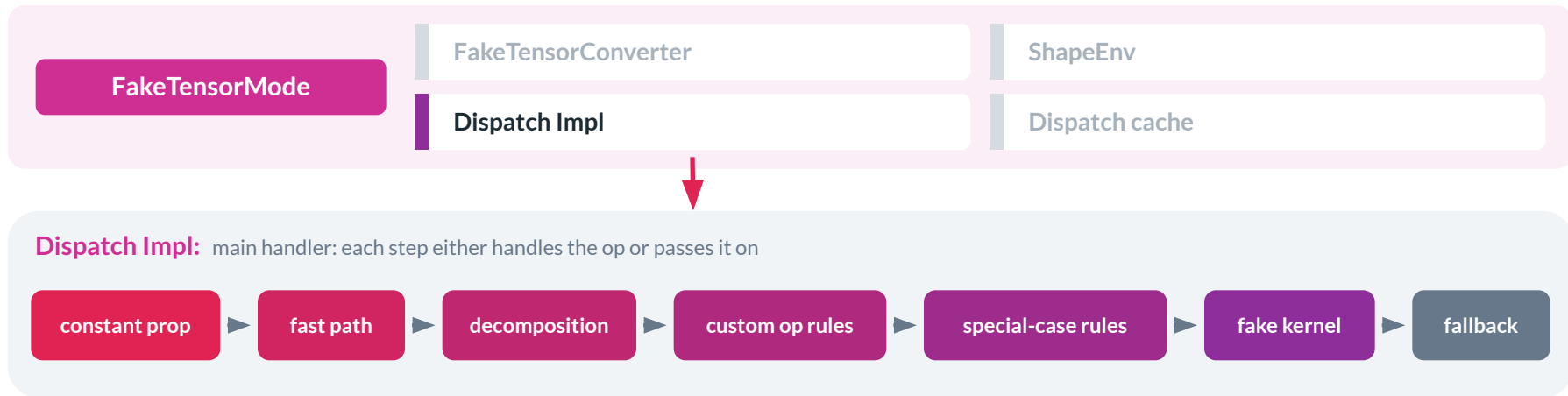
Dispatch Impl

ShapeEnv

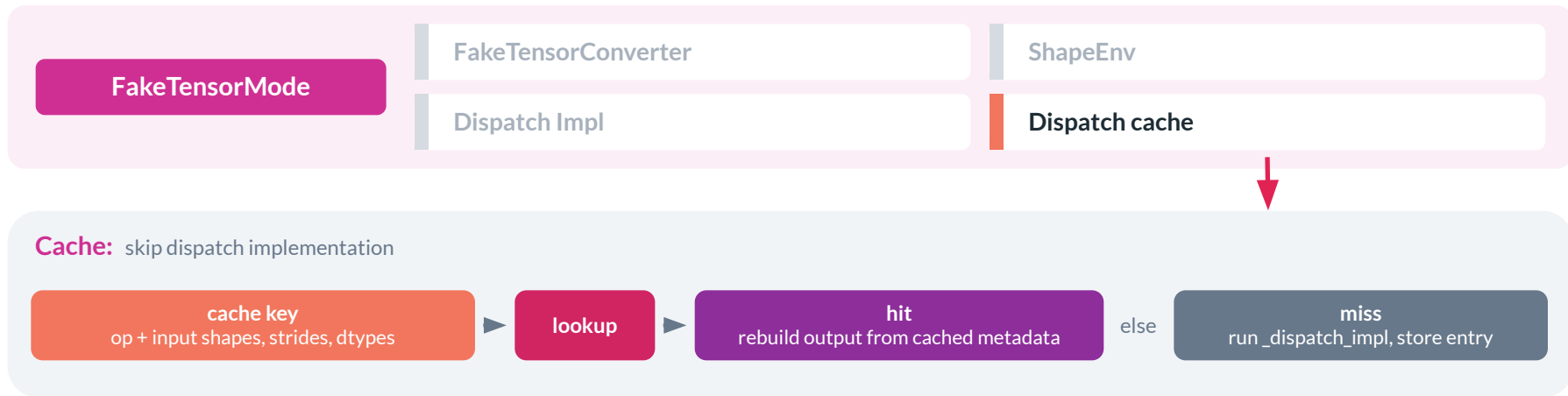
Dispatch cache



Dispatch Implementation



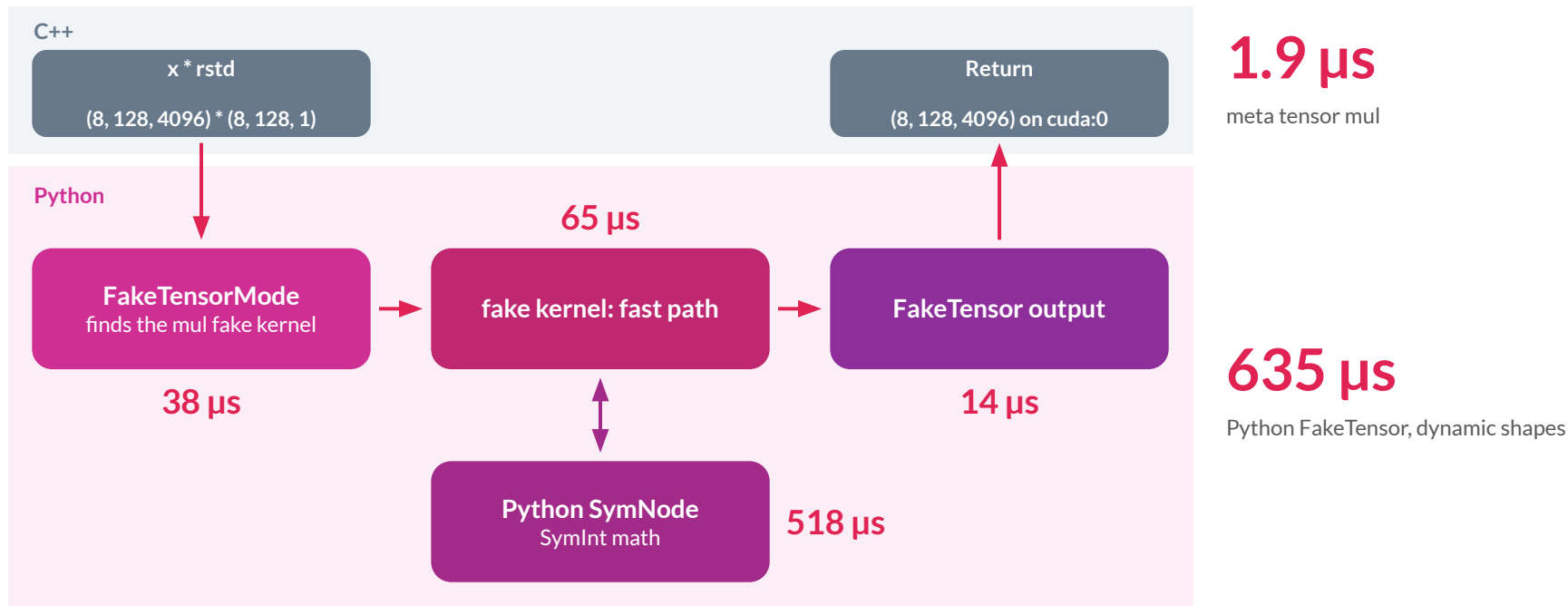
Dispatch Cache





So... what's the problem?

Python FakeTensor is Slow!



C++ FakeTensor: Eliminating Overhead

NEW

We are building FakeTensor in C++ from scratch, speeding up tracing time by 20x!

PER-OP OpInfo ops, in isolation

22.3x

118 ops

48.2 μ s $\rightarrow$ 2.2 μ s per op (meta: 1.2 μ s)

Per-op time, Python vs C++ FakeTensor on OpInfo samples; left card is the geomean over 118 ops

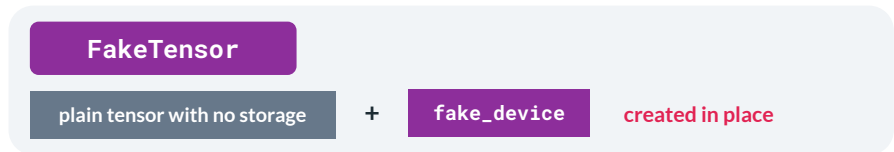
END-TO-END torchbench compile time + modern models

GPU config	C++ FakeTensor
inductor_inference	+10.7%
inductor_training	+10.3%
dynamic_inductor_inference	+6.8%
dynamic_inductor_training	+11.6%
aot_inductor_inference	+3.7%

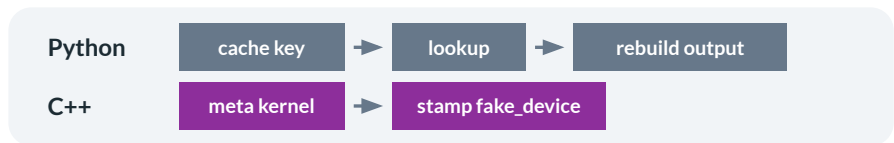
Many individual models gain **20–30%** (inductor_inference)

C++ FakeTensor: Design Decisions

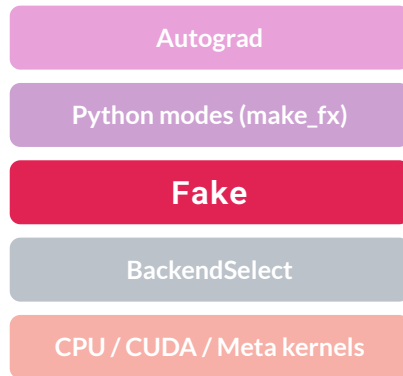
1 WHAT IS A C++ FAKETENSOR



3 NO DISPATCH CACHING



2 NEW FAKE DISPATCH KEY



4 INCREMENTAL PORTING

C++ FakeTensor: Revisiting our Example



One generic fallback handler for **ALL** ATen ops!

62 μ s

Python FakeTensor, static shapes



3.5 μ s!

C++ FakeTensor, static shapes

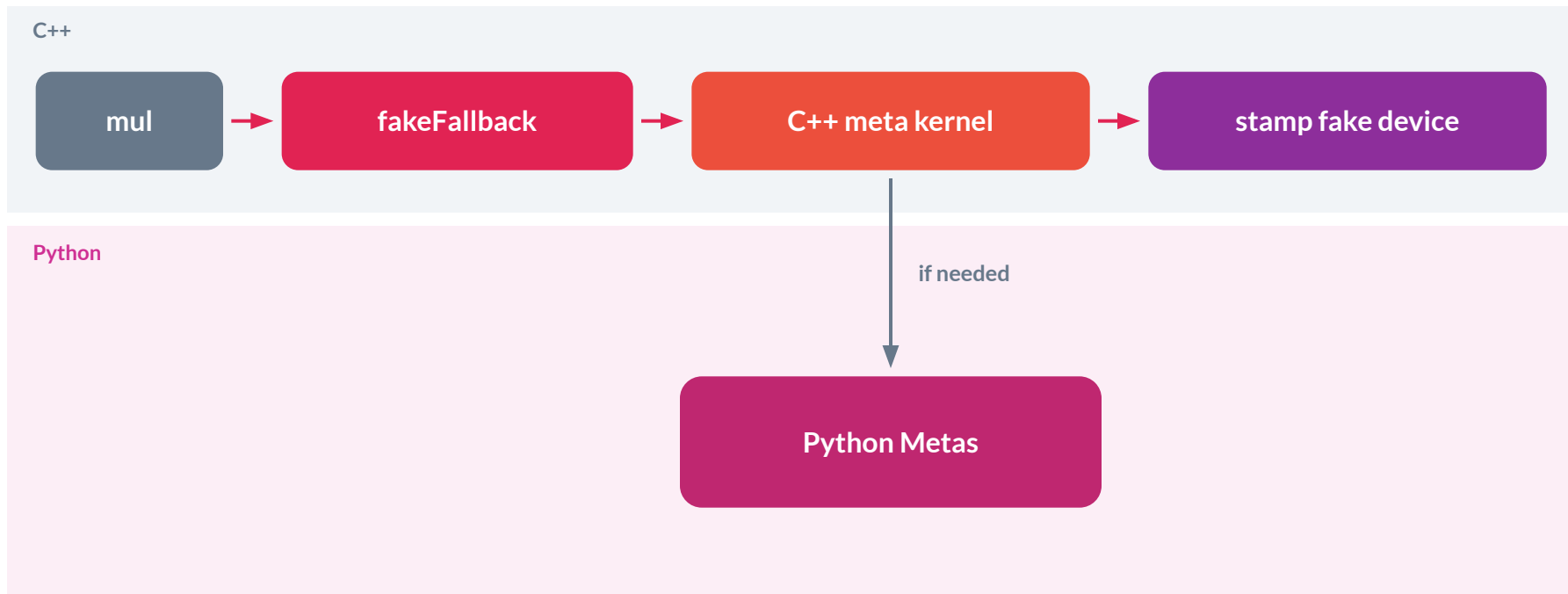
C++ FakeTensor: What's Still Left in Python?

C++

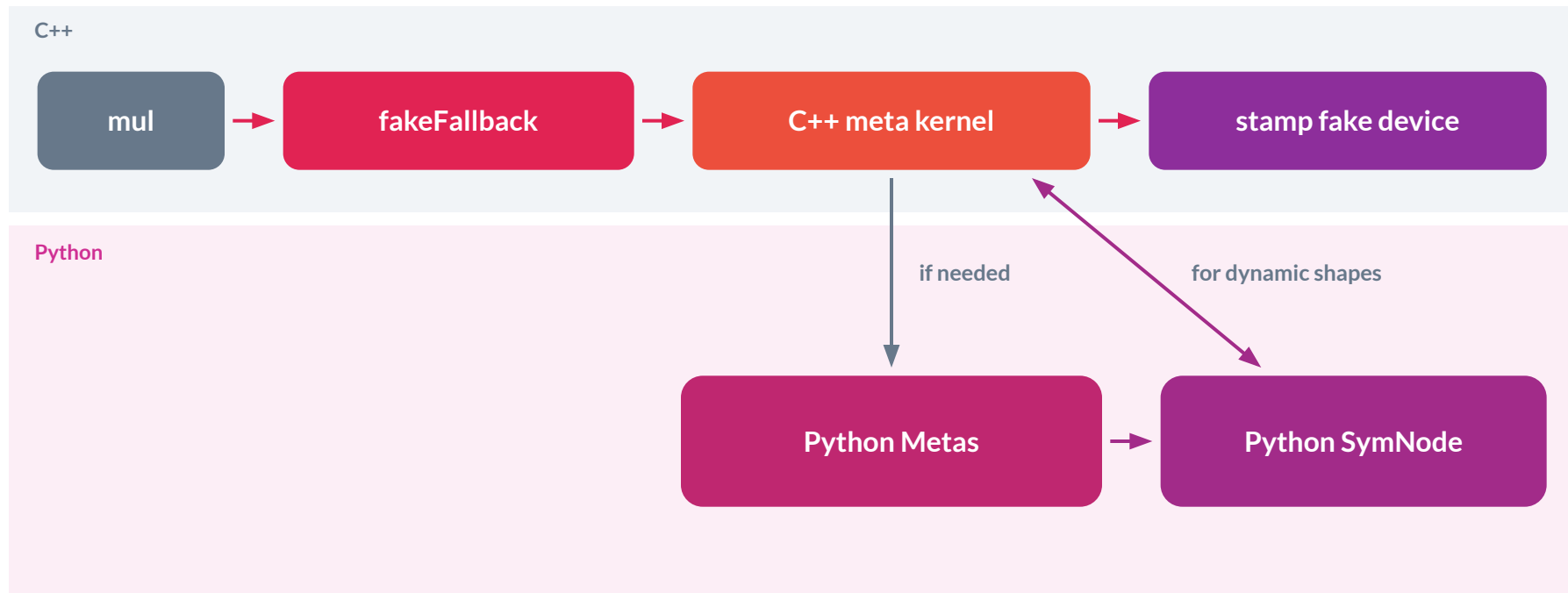


Python

1. Python Metas



2. Python SymNode



Performance with Python Dependencies...

PER-OP PERFORMANCE

22.3x

with C++ meta kernels

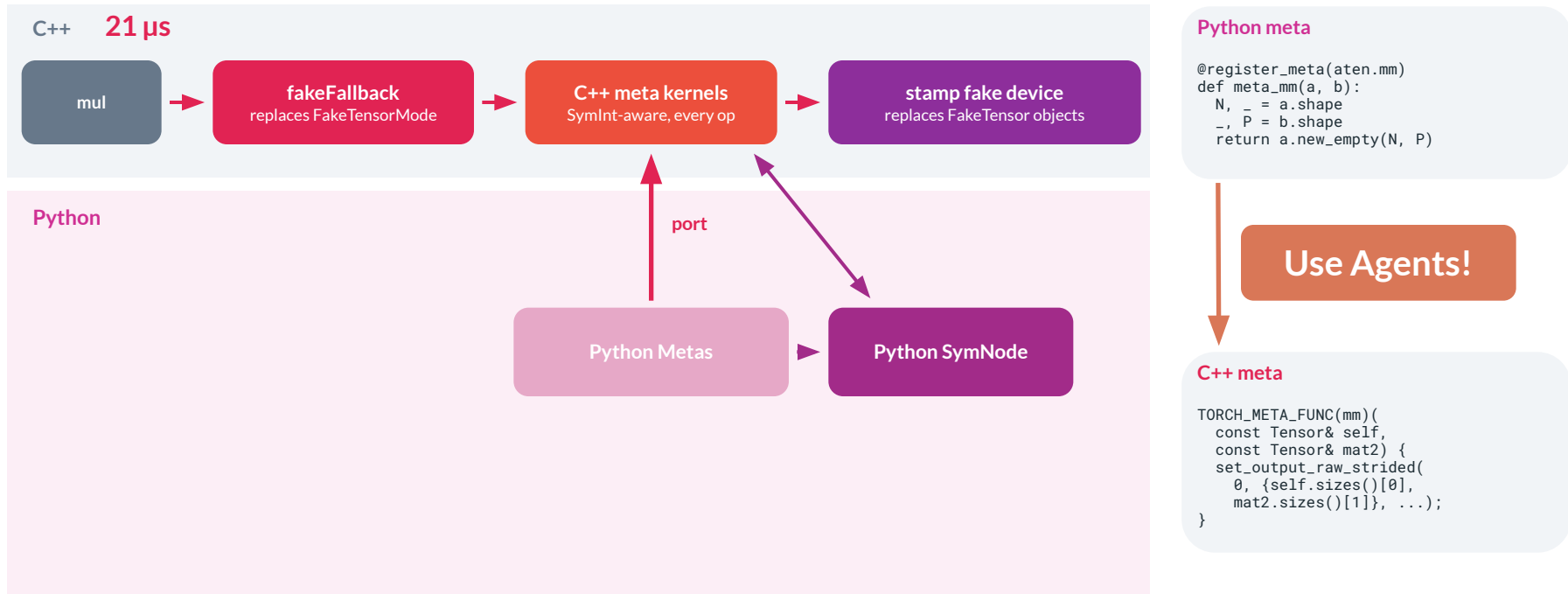
vs.

No C++ meta kernel yet still runs a Python meta

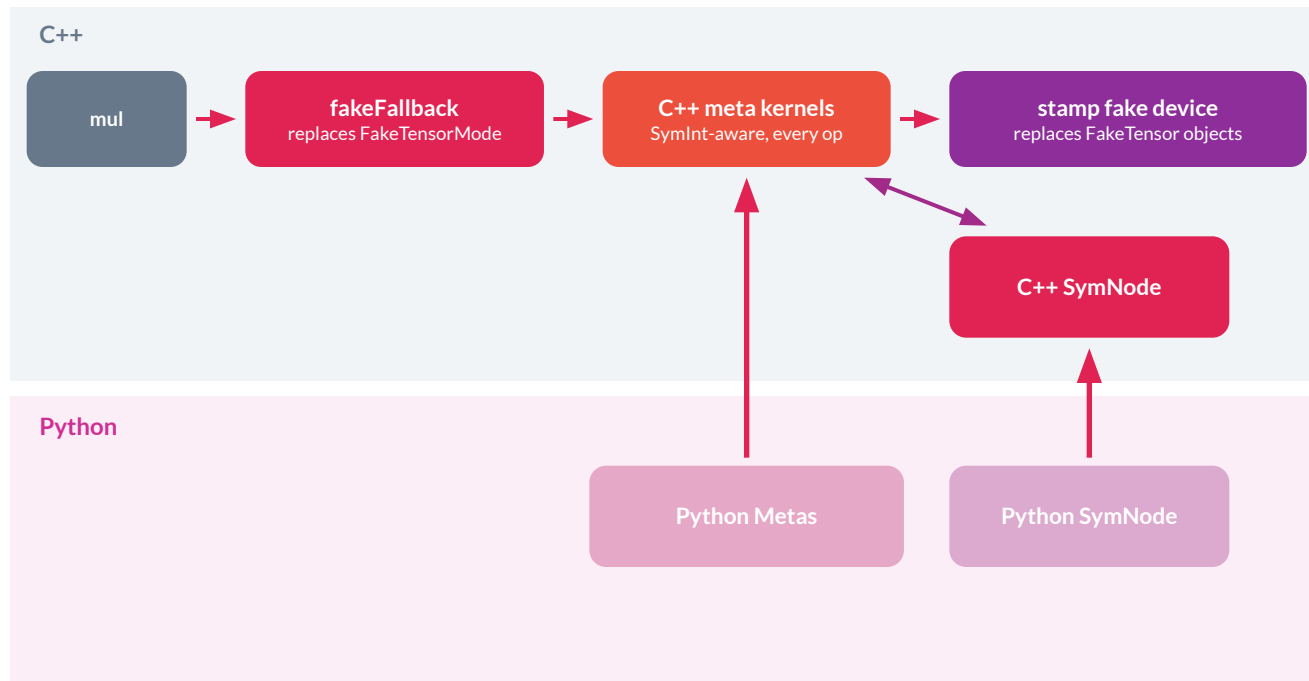
silu	39.8 μ s $\rightarrow$ 9.0 μ s	4.4x
softmax	48.0 μ s $\rightarrow$ 11.5 μ s	4.2x
max_pool2d	69.4 μ s $\rightarrow$ 21.8 μ s	3.2x

Next Step: Let's remove all Python dependencies!

Next Steps: 1. C++ Meta Kernels



Next Steps: 2. C++ SymNode



Additional improvement

+20.2%

Questions?