TensorFlow w/XLA: TensorFlow, Compiled!

Expressiveness with performance

Pre-release Documentation (or search GitHub repository for 'XLA'):
https://www.tensorflow.org/versions/master/resources/xla_prerelease.html

Jeff Dean
Google Brain team
g.co/brain
presenting work done by the XLA team and Google Brain team

"It takes a village to raise a compiler."

- Ancient proverb

Why Did We Build TensorFlow?

Wanted system that was **flexible**, **scalable**, and **production-ready**

DistBelief, our first system, was good on two of these, but lacked **flexibility**

Most existing open-source packages were also good on 2 of 3 but not all 3

TensorFlow Goals

Establish **common platform** for expressing machine learning ideas and systems

Make this platform the **best in the world** for both research and production use

Open source it so that it becomes a **platform for everyone**, not just Google

Facts and Figures

Launched on Nov. 9, 2015

Reasonably fully-featured:
auto differentiation, queues, control flow, fairly comprehensive set of ops, ...

Tutorials made system accessible

Out-of-the-box support for CPUs, GPUs, multiple devices, multiple platforms

Some Stats

500+ contributors, most of them outside Google

11,000+ commits since Nov, 2015

1M+ binary downloads

#16 most popular repository on GitHub by stars

Used in ML classes at quite a few universities now:
Toronto, Berkeley, Stanford, …

Many companies/organizations using TensorFlow:
Google, DeepMind, OpenAI, Twitter, Snapchat, Airbus, Uber, ...

TensorFlow Strengths

Flexible

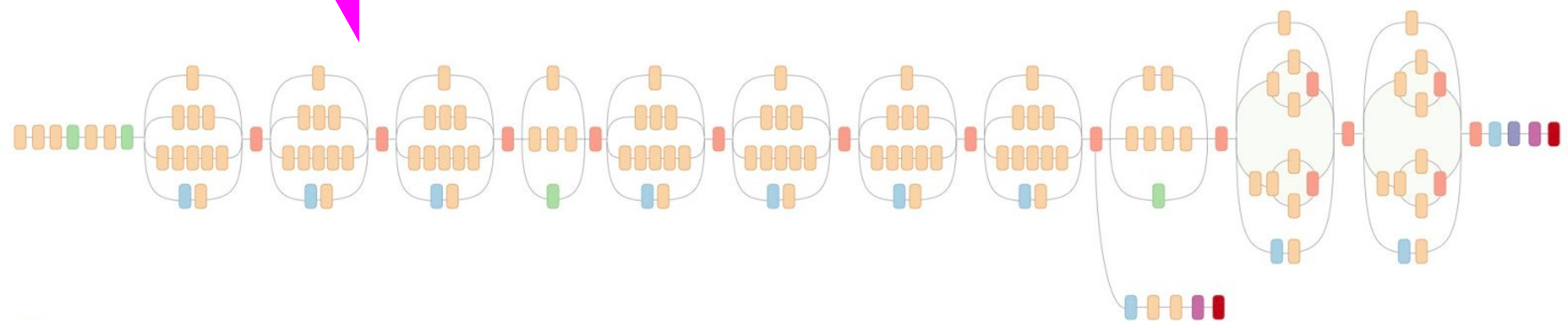
Expressive

Extensible

Just-In-Time Compilation

via XLA, "Accelerated Linear Algebra" compiler

TF graphs go in,

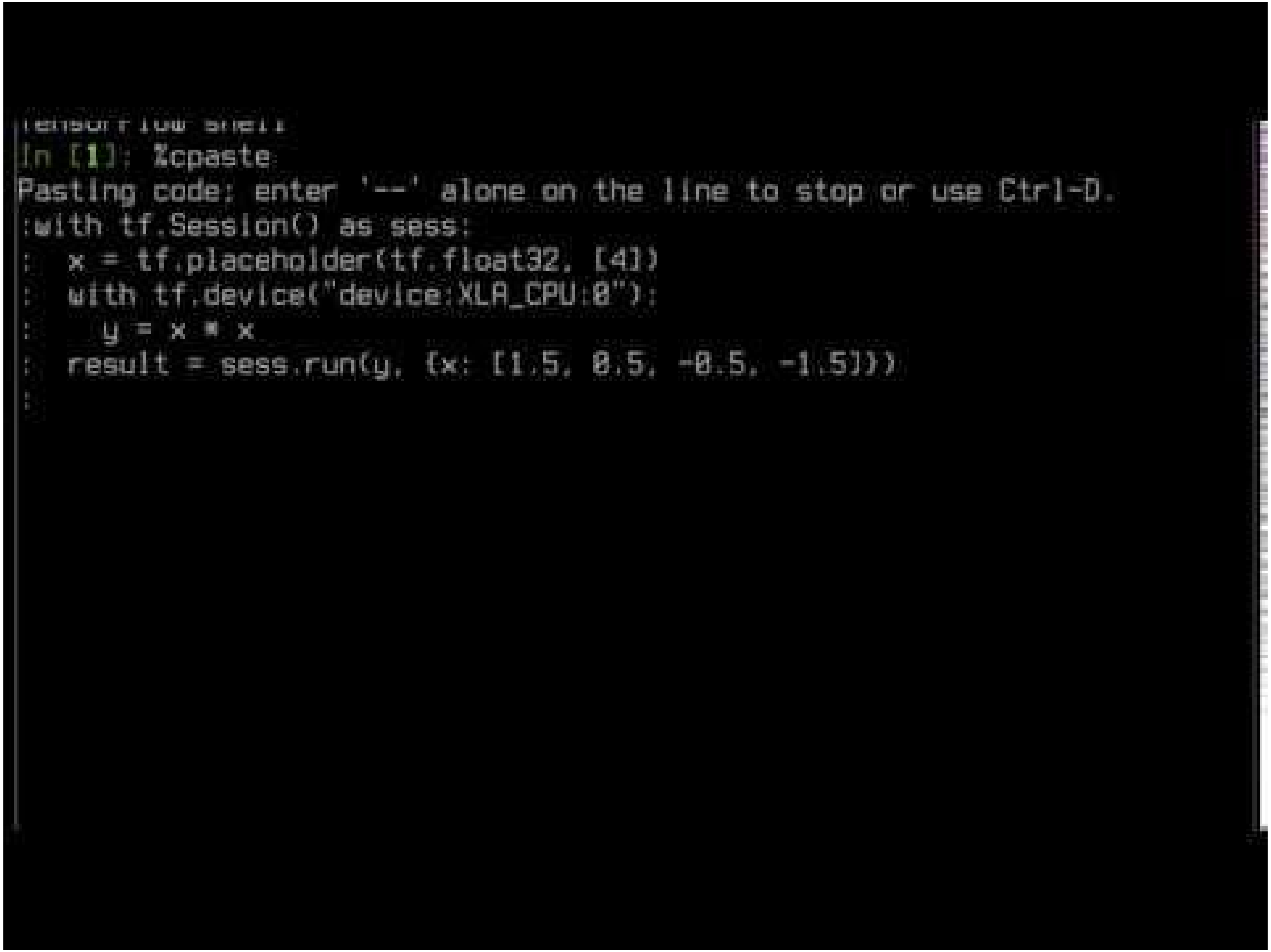


Optimized & specialized assembly comes out.

```
0x00000000      movq    (%rdx), %rax
0x00000003      vmovaps (%rax), %xmm0
0x00000007      vmulps  %xmm0, %xmm0, %xmm0
0x0000000b      vmovaps %xmm0, (%rdi)
                         ...
```

Let's explain that!

Demo:
Inspect **JIT code** in
TensorFlow
iPython shell



XLA : CPU

XLA : GPU

What's JIT all about?

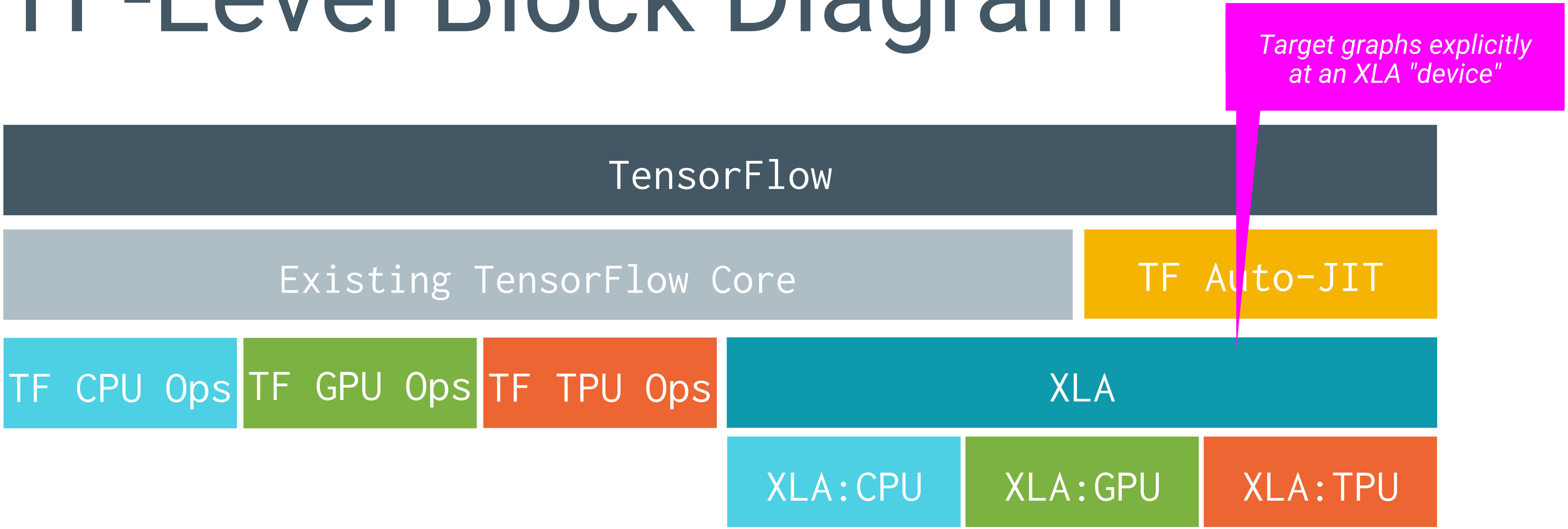
Program built at runtime

Low-overhead compilation

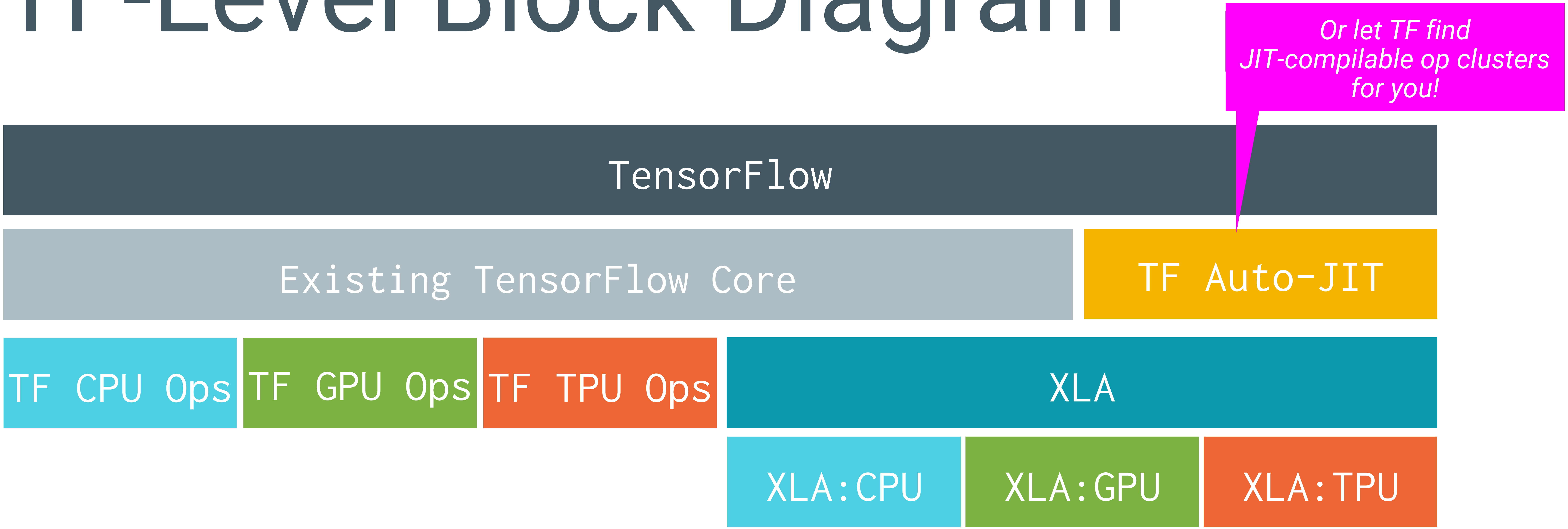
Dim variables (e.g. batch size) can bind very late

Prototype w/freedom of TF development

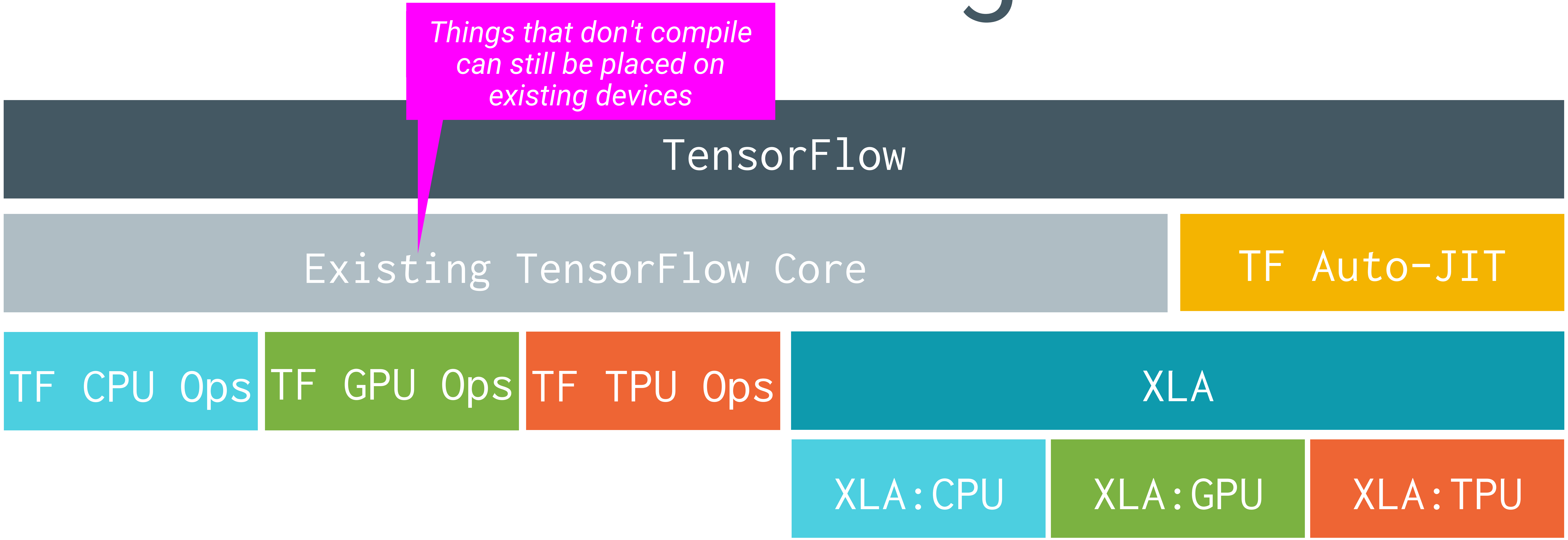
TF-Level Block Diagram



TF-Level Block Diagram



TF-Level Block Diagram



Complementary Attributes!

Flexible	Interpreted	Compiled
Expressive	Dynamic	Static
	Stateful	Pure
Extensible	"Black-Box" Modular	Primitives

Think & write this way...

But get optimization benefits of these!

What has us excited?

Server-side speedups

XLA's **JIT compilation** and **specialization**

Significant performance wins

SyntaxNet latency reductions: 200µs ⇒ 5µs (extreme case)

What has us excited?

Mobile footprint reductions

XLA's **Ahead-of-Time compilation**

Turn **models to executables**

- Eliminates much of TensorFlow runtime
- Cross-compile for ARM, PPC, x86

LSTM model for mobile: ~1MB ⇒ 10s of KBs

What has us excited?

Whole-Program Analysis made easy

XLA's **High-Level Optimizer**

Reusable toolkit of global optimizations

Layout (e.g. dim order, cache-line padding) is parameterized

Mix & match platform-agnostic & target specific passes

Caveats?

It's still early days!

Best time to start the dialogue :-)

Not all TensorFlow ops compile

Note: some won't compile by design (e.g. DynamicStitch)

Wins accumulating day by day, not everything is faster yet

Haven't devoted equal time to all platforms

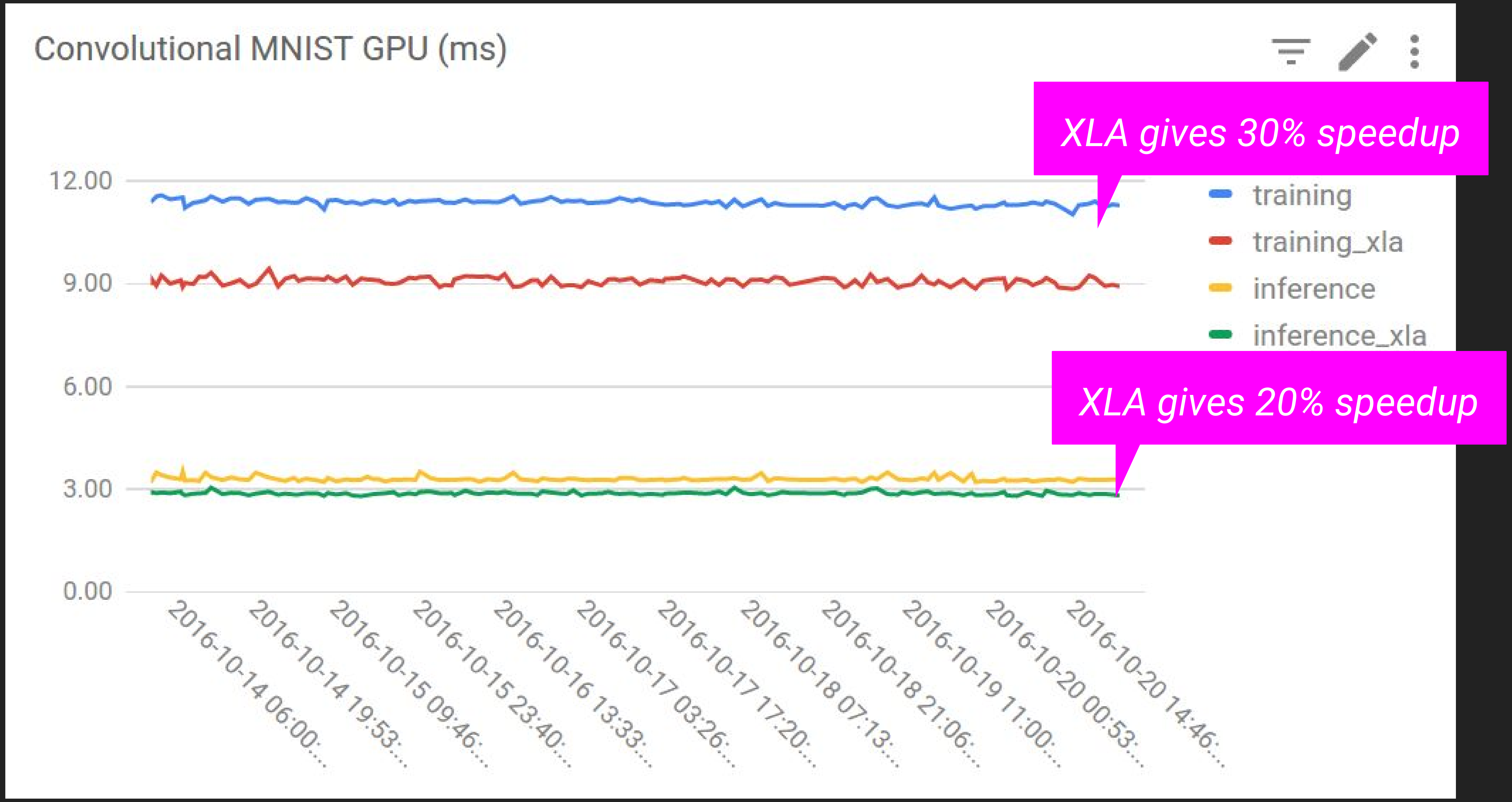
With the community we believe we could do much more!

Open source release in O(1 month)

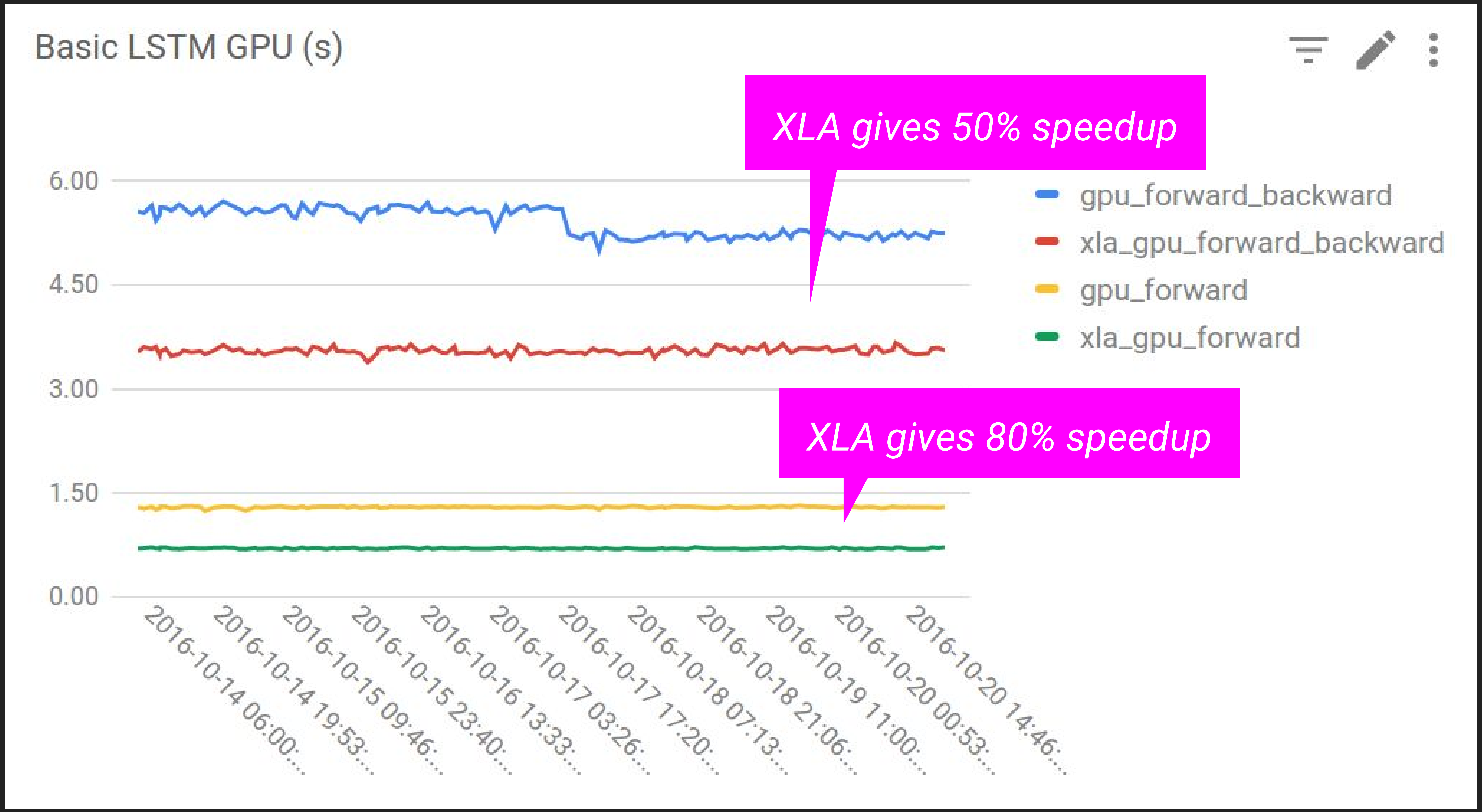
(That being said...)

Benchmark Results

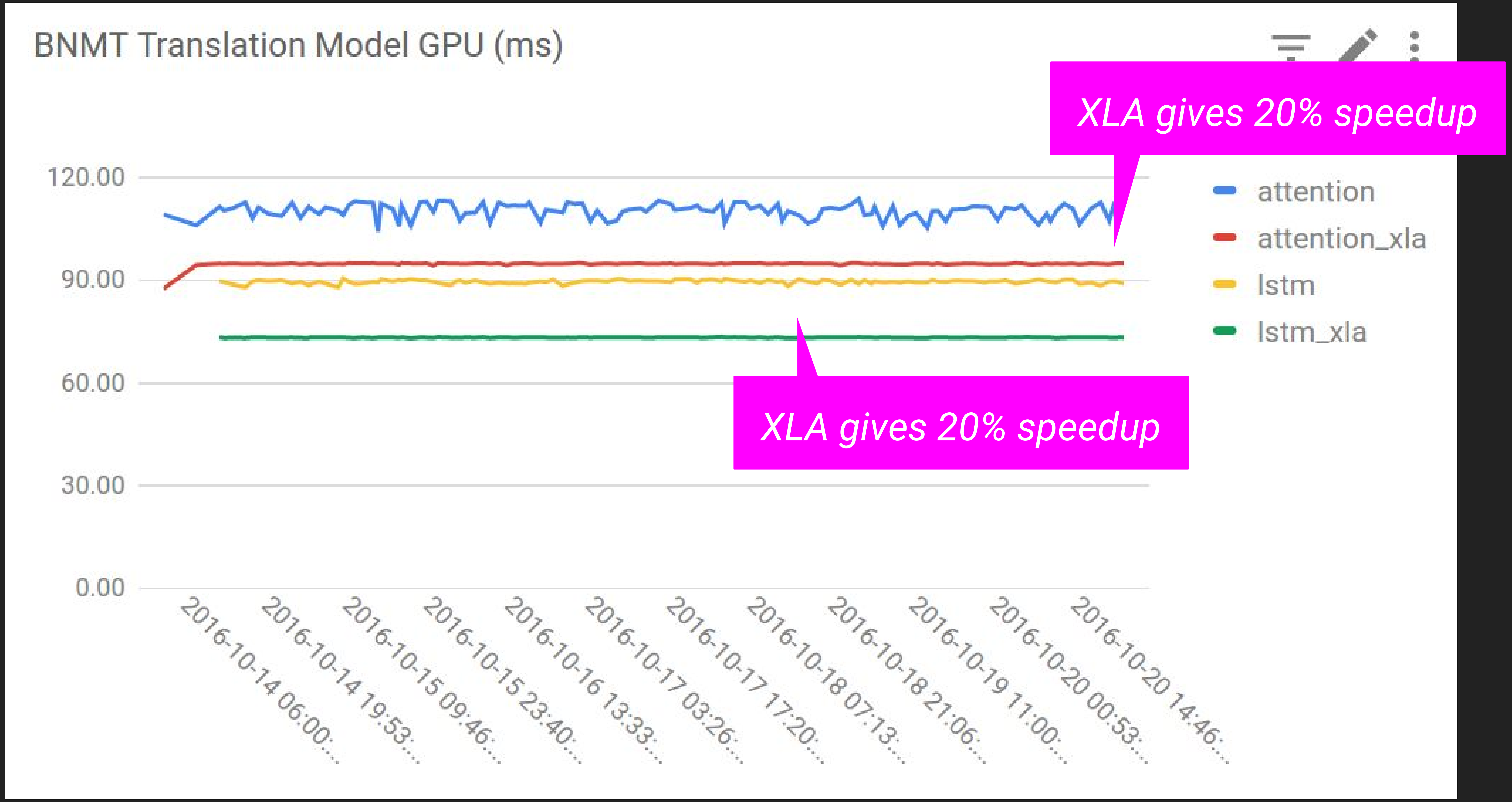
TF:XLA:GPU vs TF:GPU



Increasing complexity from "toy demo" to "large, complex neural nets"...



Ah, more real!
LSTMs have element-wise ops the compiler "fuses"
More on that later...



Very real: Neural Machine Translation! https://goo.gl/SzbQCS
Full-model runs also indicate ~20% speedup

New compiler optimizations tend to benefit across many models

Compilation benefits

Specializes the code for your computation

Eliminates op dispatch overhead

Fuses ops: avoids round trips to memory

Analyzes buffers: reuses memory, updates in-place

Unrolls, vectorizes via known dimensions

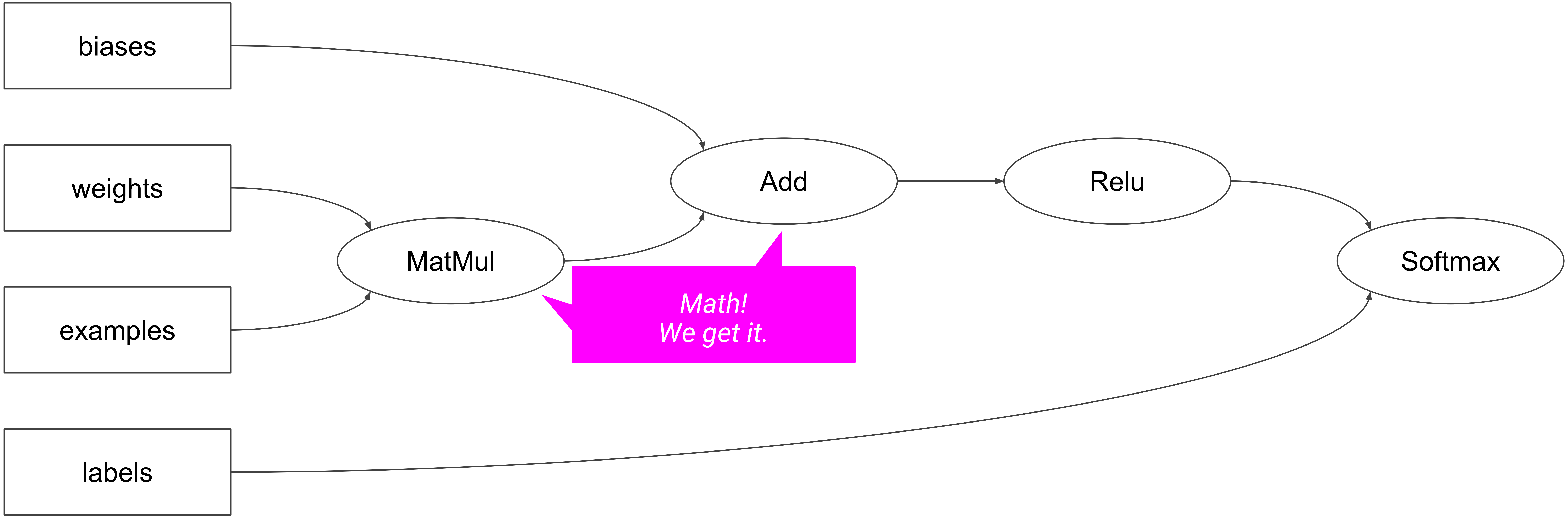
↓ executable size: generate what you need!

Under the Hood

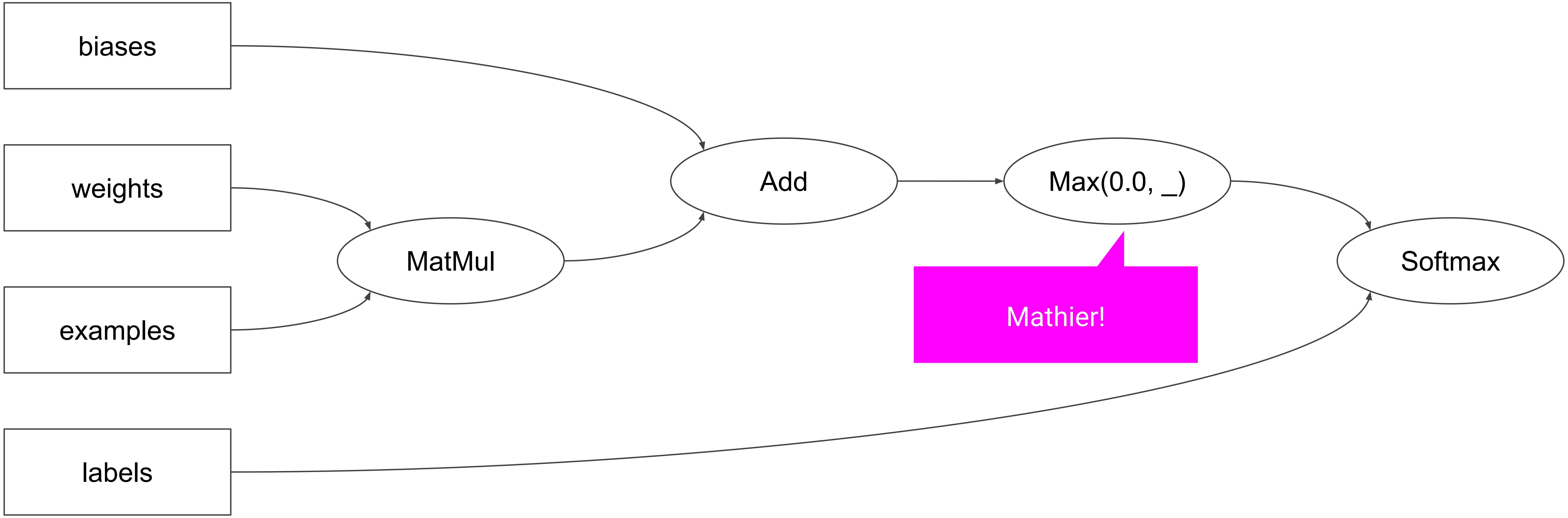
XLA program = static, decomposed TF ops

Math-looking **primitive ops**
Make **macro-ops by composition**
Supports many neural net definitions

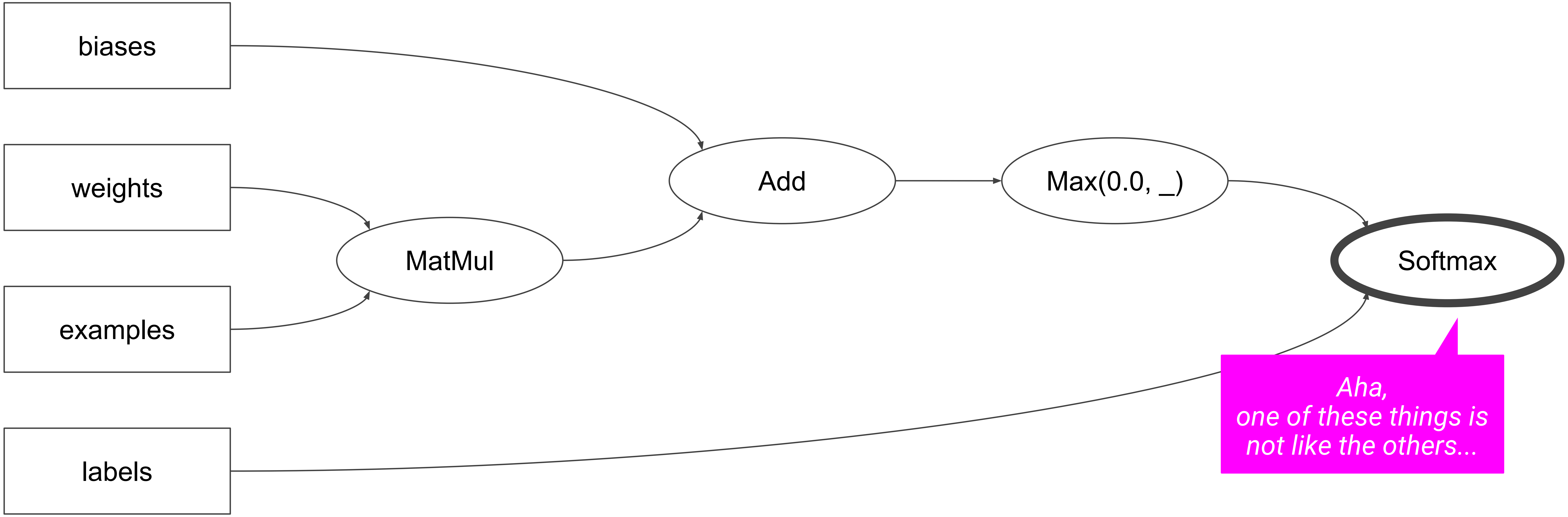
Classic TensorFlow example



Classic TensorFlow example



Classic TensorFlow example



A key question:

Why write every new macro-op in C++?

Why can't we just compose them out of existing TF ops?

An answer: you don't want to pay a performance penalty.

But, what if op composition had the performance of C++?

TensorFlow:XLA bridge does built-in op decomposition for you

The kind of stuff C++ SoftMax code has inside...

```
auto weighted = Dot(input, weights);
auto weighted_sum = Add(weighted, biases, /*broadcast=*/{1});
auto max_activation = Reduce(
    weighted_sum, Constant(MinValue(F32)), Max, /*reduce_dims=*/{1});
auto activations_normalized =
   Exp(Sub(weighted_sum, max_activation, /*broadcast=*/{0}));
auto activations_sum =
   Reduce(activations_normalized, Constant(0.0f), Add, /*reduce_dims=*/{1});
auto predicted = Div(activations_normalized,
                     activations_sum, /*broadcast=*/{0});
```

primitive operation composition ⇒ fused & optimized composite kernel

Automatic Operation Fusion

XLA composes & specializes primitive operations

Note: this is all expressible in TensorFlow

Not done due to performance concerns

XLA removes the performance concern

Avoids combinatorial explosion of op fusions

(e.g. for custom LSTM cell)

*macro-ops * primitives * dim sizes * backends * devices!*

XLA APIs

(never seen by normal TensorFlow users)

XLA Block Diagram

XLA is Designed for Reuse

Retargetability & pragmatism

Pluggable backends

HLO pass "toolkit"

Can emit calls to libraries like BLAS or CuDNN

Either use LLVM

Or Bring-Your-Own Low Level Optimizer

Minimal XLA backend:

An LLVM pipeline

A StreamExecutor plugin

XLA


Let's instantiate for different platforms!
TensorFlow
ComputationBuilder API
Executor API
High-Level Optimizer (HLO)
In-Memory Executable Object
Code Cache
TransferManager
StreamExecutor
Low-Level Optimizer (LLO)

XLA:CPU


TensorFlow
In-memory {ARM, PPC, x86} JIT blob
ComputationBuilder API
Executor API
In-Memory Executable Object
Code Cache
TransferManager
High-Level Optimizer (HLO)
StreamExecutor:Host
LLVM:$TARGET

XLA:GPU:CUDA


TensorFlow
In-memory kernels & library calls
ComputationBuilder API
Executor API
In-Memory Executable Object
Code Cache
TransferManager
High-Level Optimizer (HLO)
StreamExecutor:CUDA
LLVM:NVPTX

XLA:GPU:OpenCL

{CPU, GPU} HLO pipeline; one slide each

Mixes
target-independent passes
& dependent passes
in a pipeline

cpu_compiler.cc

```
HloPassPipeline pipeline("CPU");
pipeline.AddPass<Inliner>()
   .AddPass<ConvCanonicalization>()
   .AddPass<HloPassFix<ReshapeMover>>()
   .AddPass<HloSubcomputationUnification>()
   .AddPass<HloCSE>(/*is_layout_sensitive=*/false)
   .AddPass<CpuInstructionFusion>()
   .AddPass<CpuLayoutAssignment>();
   .AddPass<HloPassFix<AlgebraicSimplifier>>(
      /*is_layout_sensitive=*/true, /*add_bitcasts=*/true)
   .AddPass<HloCSE>(/*is_layout_sensitive=*/true)
   .AddPass<CopyInsertion>()
   .AddPass<ParallelizationPreparation>();
pipeline.Run(hlo_module);
```

gpu_compiler.cc

```
HloPassPipeline pipeline("GPU");
pipeline.AddPass<ConvolutionFolding>()
   .AddPass<ReshapeMover>().AddPass<TransposeFolding>()
   .AddPass<HloSubcomputationUnification>()
   .AddPass<HloCSE>(/*is_layout_sensitive=*/false)
   .AddPass<HloPassFix<ReduceFactorizer>>(
     device_desc.threads_per_core_limit() * device_desc.core_count())
   .AddPass<HloPassFix<AlgebraicSimplifier>>(false)
   .AddPass<ReduceSplitter>()
   .AddPass<GpuInstructionFusion>(/*may_duplicate=*/false)
   .AddPass<PadInsertion>().AddPass<GpuLayoutAssignment>()
   .AddPass<HloPassFix<AlgebraicSimplifier>>(
     /*is_layout_sensitive=*/true, /*add_bitcasts=*/true)
   .AddPass<HloCSE>(/*is_layout_sensitive=*/true).AddPass<GpuCopyInsertion>();
pipeline.Run(hlo_module);
```

Passes are reused across targets

Specialize/optimize for runtime-observed device

Not shown: buffer assignment & stream assignment too!

XLA: Prototype to Deployment

Potential at various phases of the lifecycle

JIT compilation when prototyping

Compilation caching as you scale

AoT compilation for mobile/embedded & latency

Control & observe **static properties** of the program

E.g. peak memory usage

Future Work

ALWAYS MORE PERFORMANCE!

Multi-device-targeting compilation

Cross-layer optimizations

Sparse operation support

Feedback-directed opt & auto-tuning

Conclusions:

XLA release for TensorFlow is coming **soon**!

Performance will improve across the board

Write the code naturally, let compiler deal with performance

Modular infrastructure

Whole-program optimization

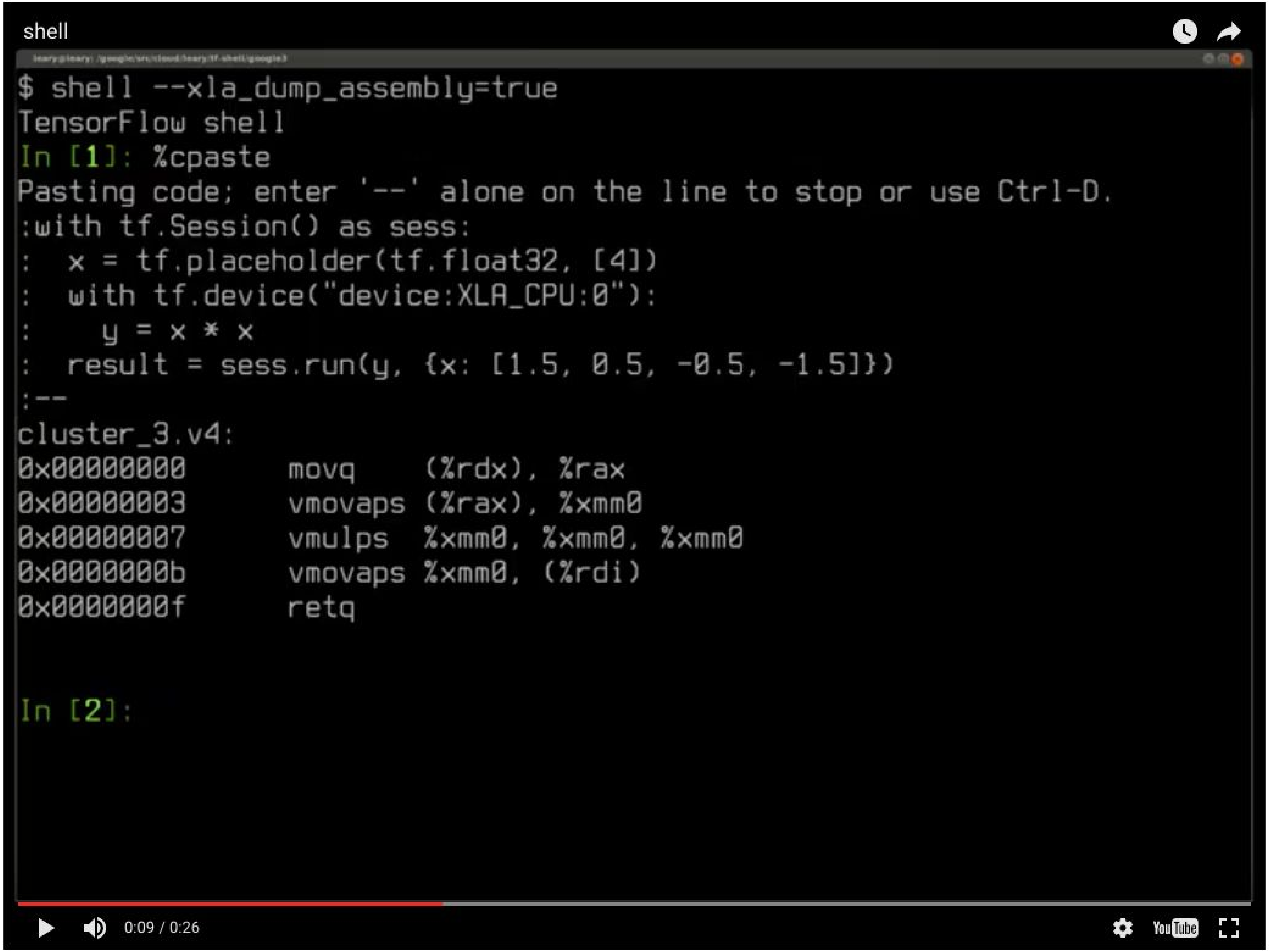
Mix compilation & library techniques

Easy to target wide variety of different kinds of HW

Pre-release Documentation (or search TensorFlow GitHub repository for ‘XLA’):
https://www.tensorflow.org/versions/master/resources/xla_prerelease.html

Backup slides in case internet doesn’t work for video

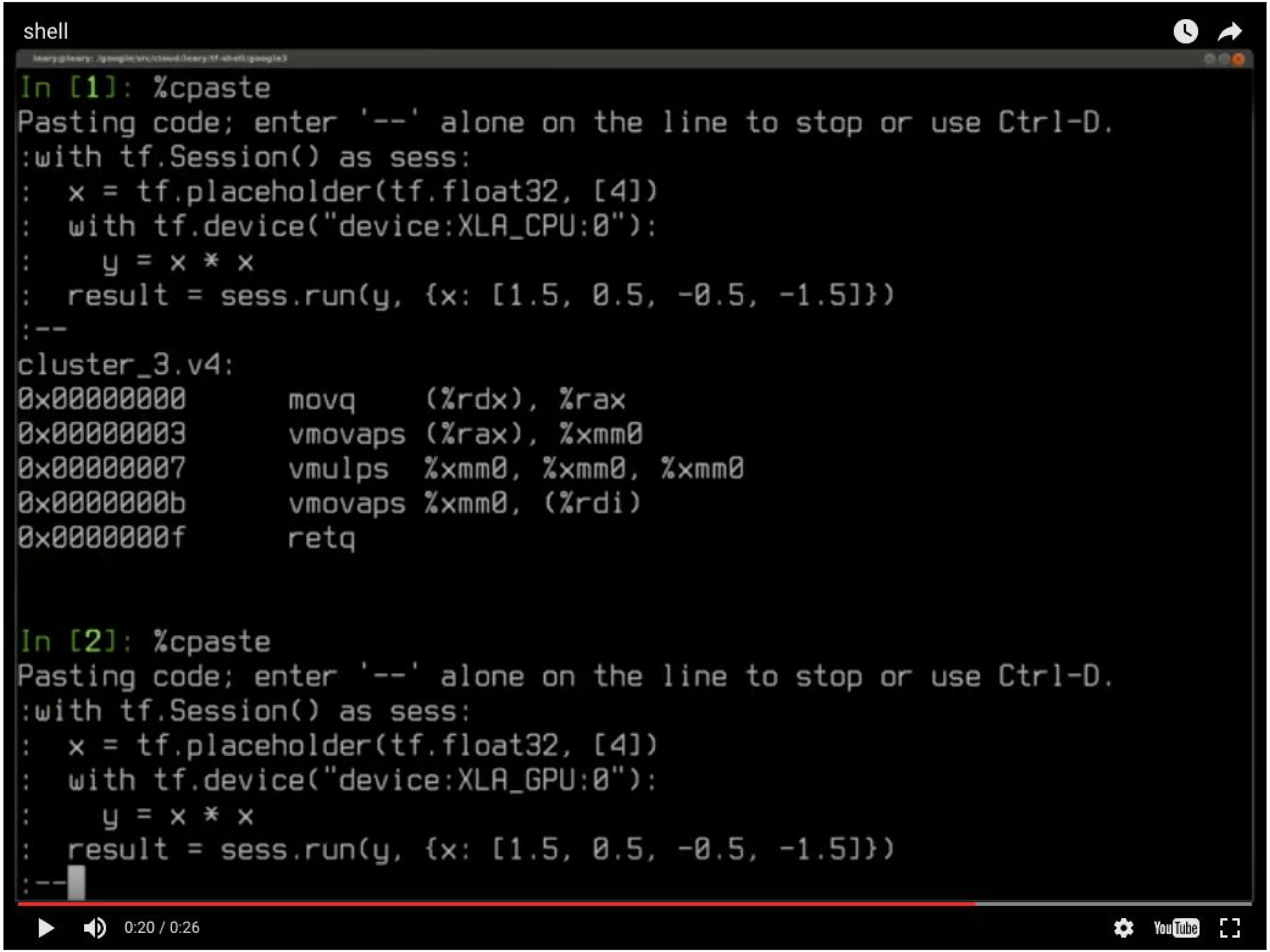
Demo:
Inspect **JIT code** in **TensorFlow iPython shell**



XLA : CPU

XLA : GPU

Demo:
Inspect **JIT code** in **TensorFlow iPython shell**



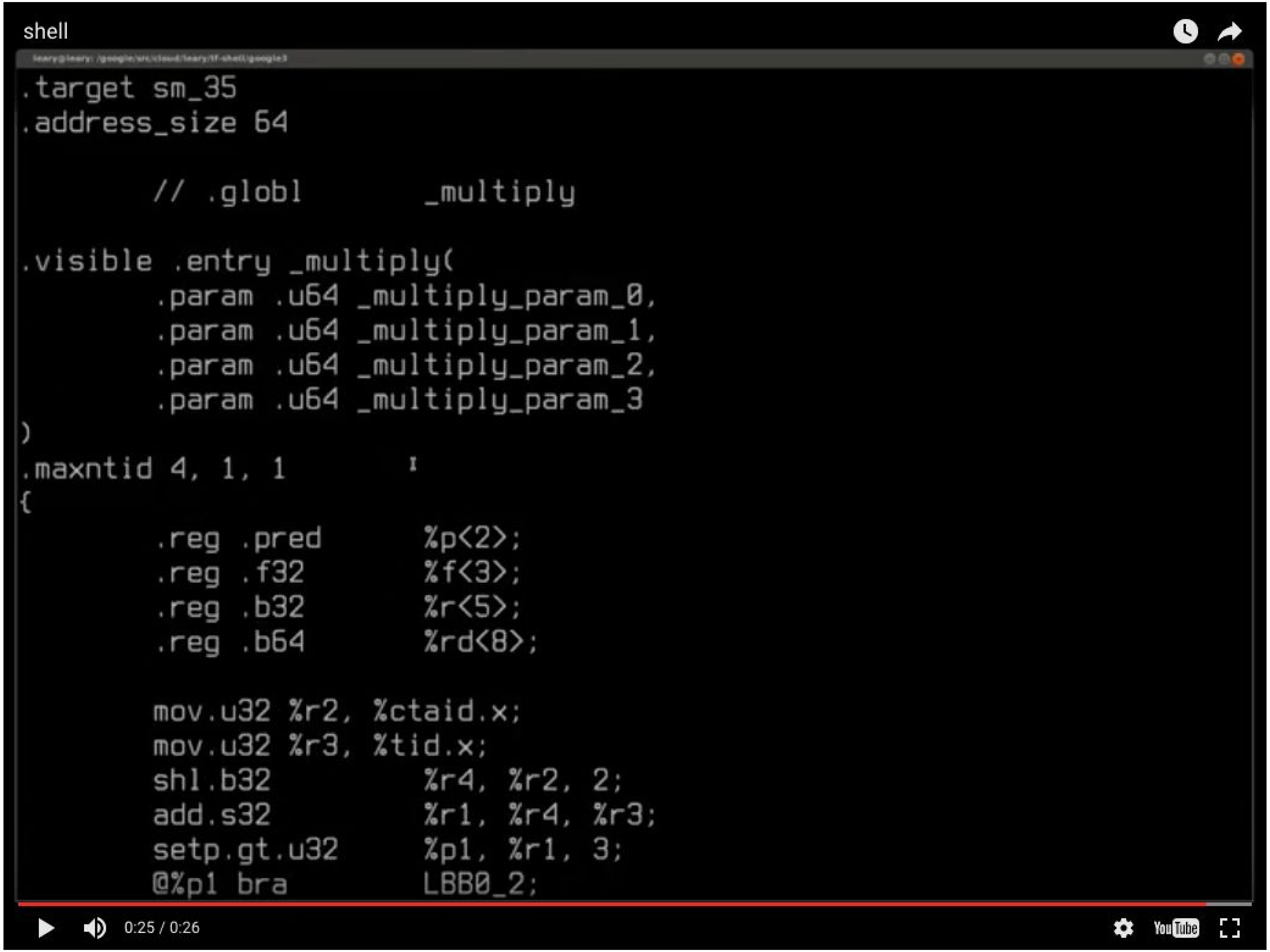
```
shell
In [1]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:with tf.Session() as sess:
:  x = tf.placeholder(tf.float32, [4])
:  with tf.device("device:XLA_CPU:0"):
:    y = x * x
:  result = sess.run(y, {x: [1.5, 0.5, -0.5, -1.5]})
:--
cluster_3.v4:
0x00000000      movq    (%rdx), %rax
0x00000003      vmovaps (%rax), %xmm0
0x00000007      vmulps  %xmm0, %xmm0, %xmm0
0x0000000b      vmovaps %xmm0, (%rdi)
0x0000000f      retq


In [2]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:with tf.Session() as sess:
:  x = tf.placeholder(tf.float32, [4])
:  with tf.device("device:XLA_GPU:0"):
:    y = x * x
:  result = sess.run(y, {x: [1.5, 0.5, -0.5, -1.5]})
:--
0:20 / 0:26
```

XLA : CPU

XLA : GPU

Demo:
Inspect **JIT code** in **TensorFlow iPython shell**



XLA : CPU

XLA : GPU