TorchDynamo Deep Dive

Agenda

- Motivation
- TorchDynamo bytecode analysis
- TorchDynamo components
- Practical session

TorchDynamo: Motivation

The Great ML Framework Debate

Eager Mode

- Preferred by users
- Easier to use programming model
- Easy to debug
- PyTorch is a primarily an eager mode framework

Graph Mode

- Preferred by backends and framework builders
- Easier to optimize with a compiler
- Easier to do automated transformations

PyTorch’s Many Attempts at Graph Modes

torch.jit.trace

- Record + replay
- Unsound
- Can give incorrect results because it ignores Python part of program

torch.jit.script

- AOT parses Python into graph format
- Only works on ~45% of real world models
- High effort to “TorchScript” models

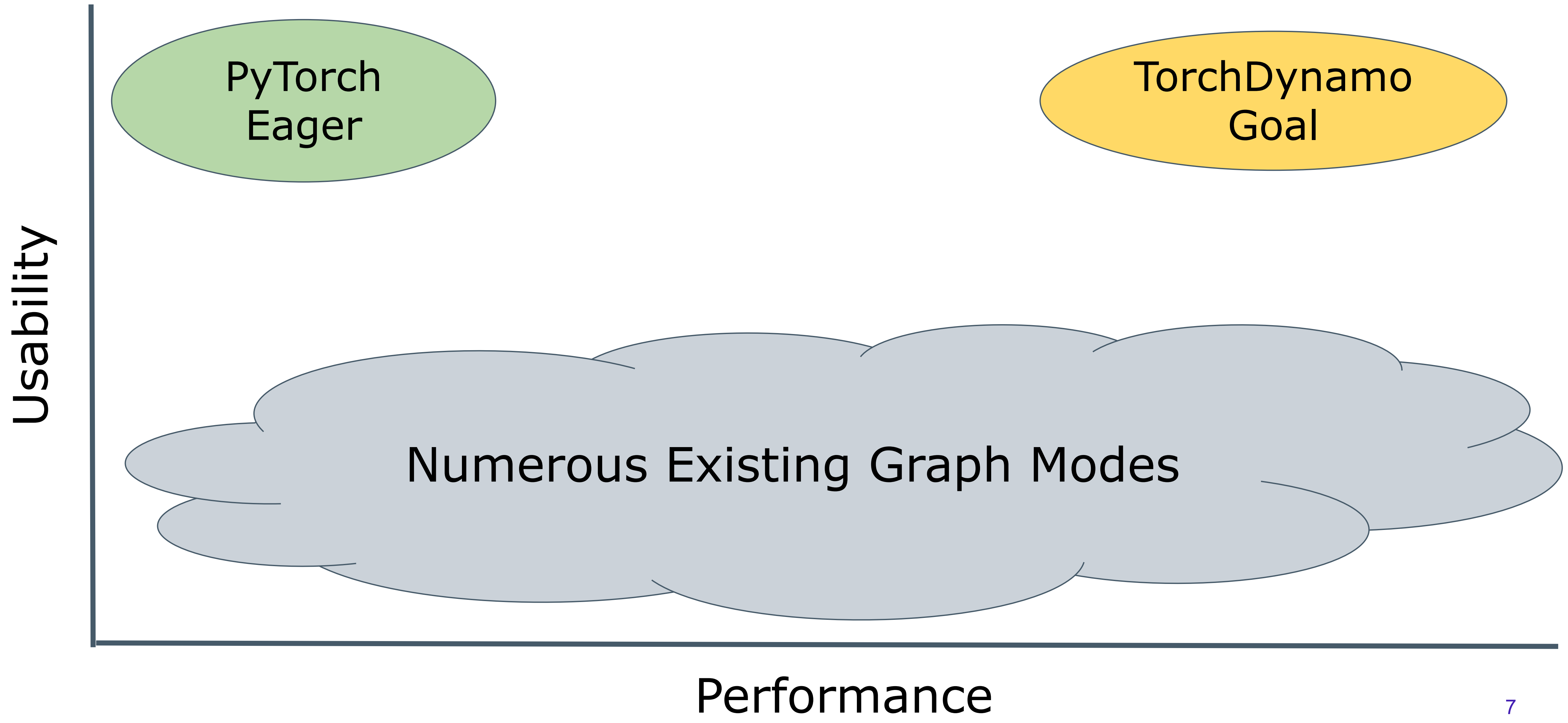
PyTorch Models Are Not Static Graphs

PyTorch users write models where program graphs are impossible

- Convert tensors native Python types (x.item(), x.tolist(), int(x), etc)
- Use other frameworks (numpy/xarray/etc) for part of their model
- Data dependent Python control flow or other dynamism
- Exceptions, closures, generators, classes, etc

These violate the assumptions of most graph mode backends

PyTorch Usability/Performance Tradeoff



7k+

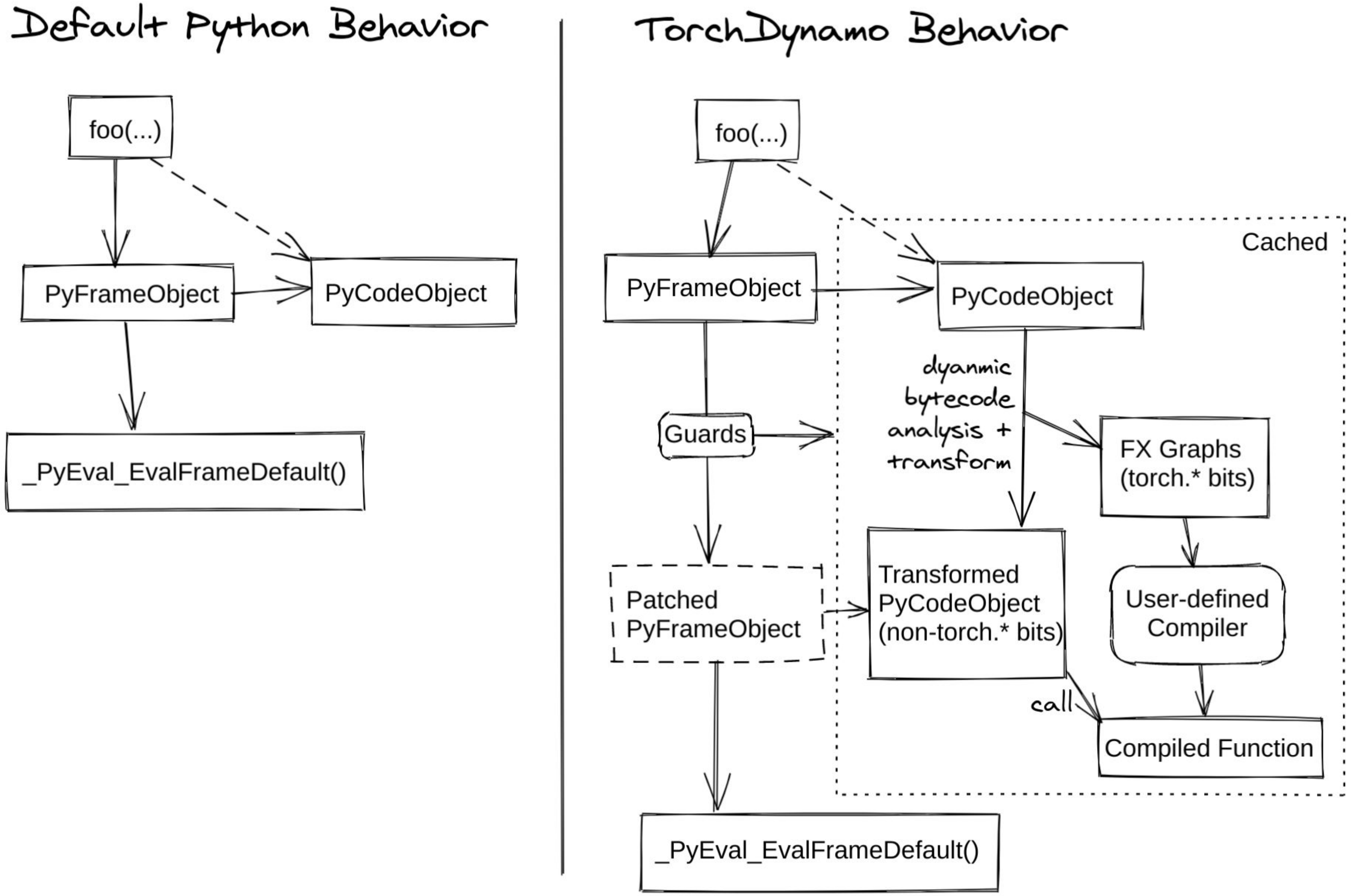
Crawled GitHub Models

TorchDynamo

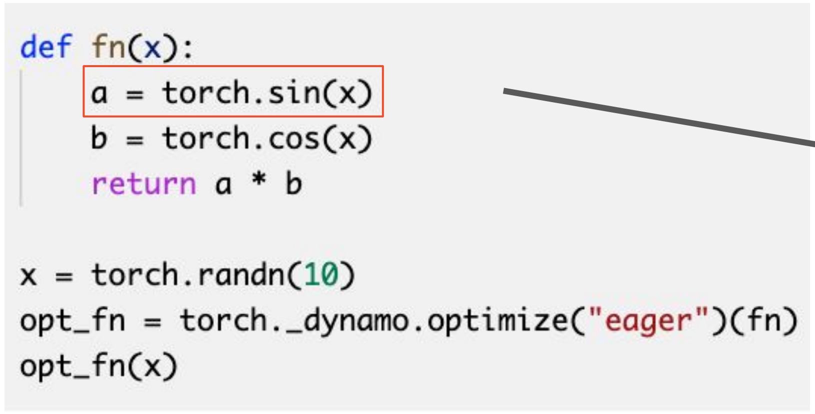
Python-level JIT compiler to make unmodified PyTorch programs run faster

- Sprinkles graphs in eager mode
- Fallback to Python for hard-to-accelerate (non-graph) code

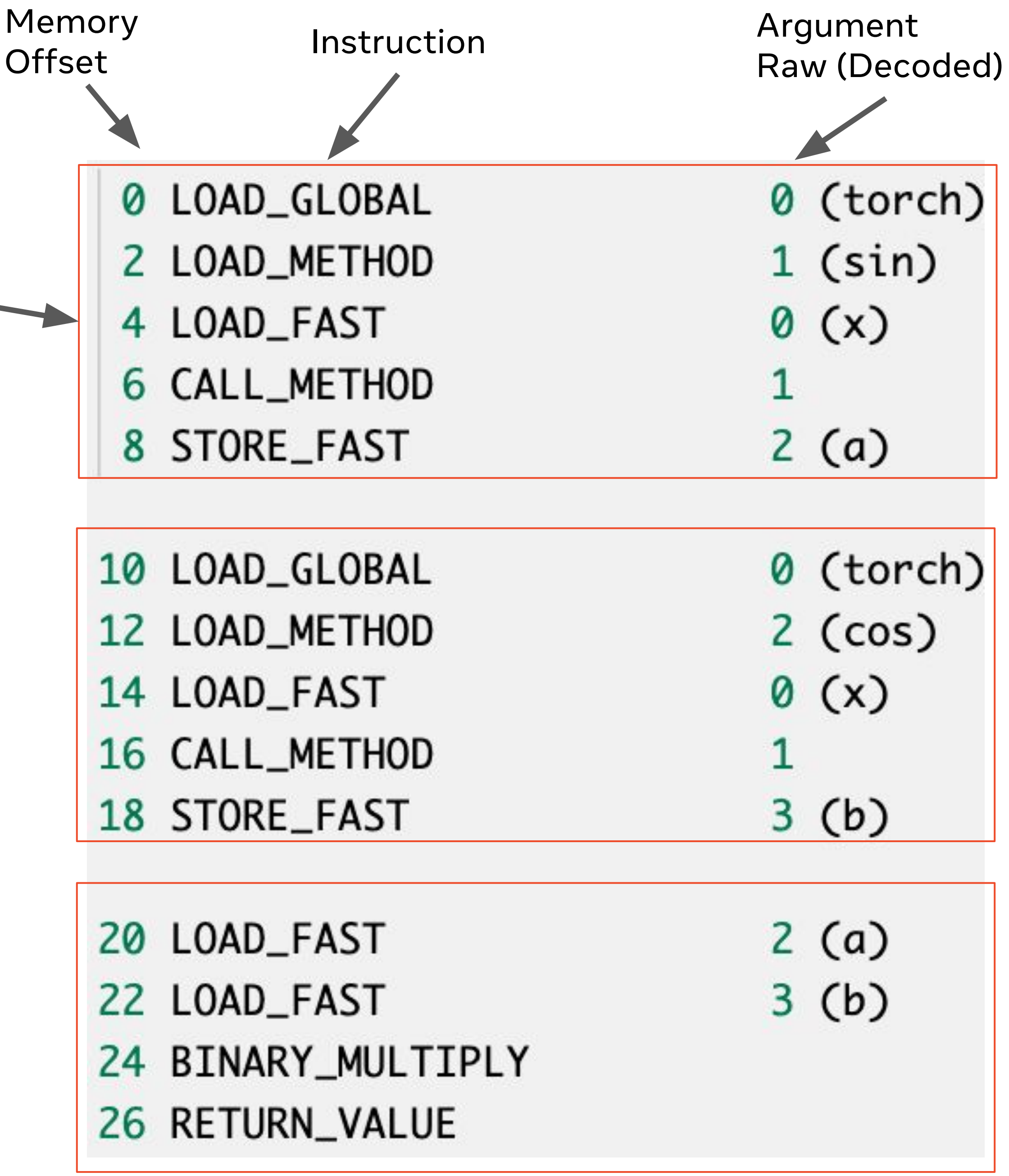
TorchDynamo: How does it work?


Default Python Behavior
foo(...)
PyFrameObject
PyCodeObject
_PyEval_EvalFrameDefault()
TorchDynamo Behavior
foo(...)
Cached
PyFrameObject
PyCodeObject
dyanmic bytecode analysis + transform
Guards
FX Graphs (torch.* bits)
Patched PyFrameObject
Transformed PyCodeObject (non-torch.* bits)
User-defined Compiler
call
Compiled Function
_PyEval_EvalFrameDefault()

Toy Example



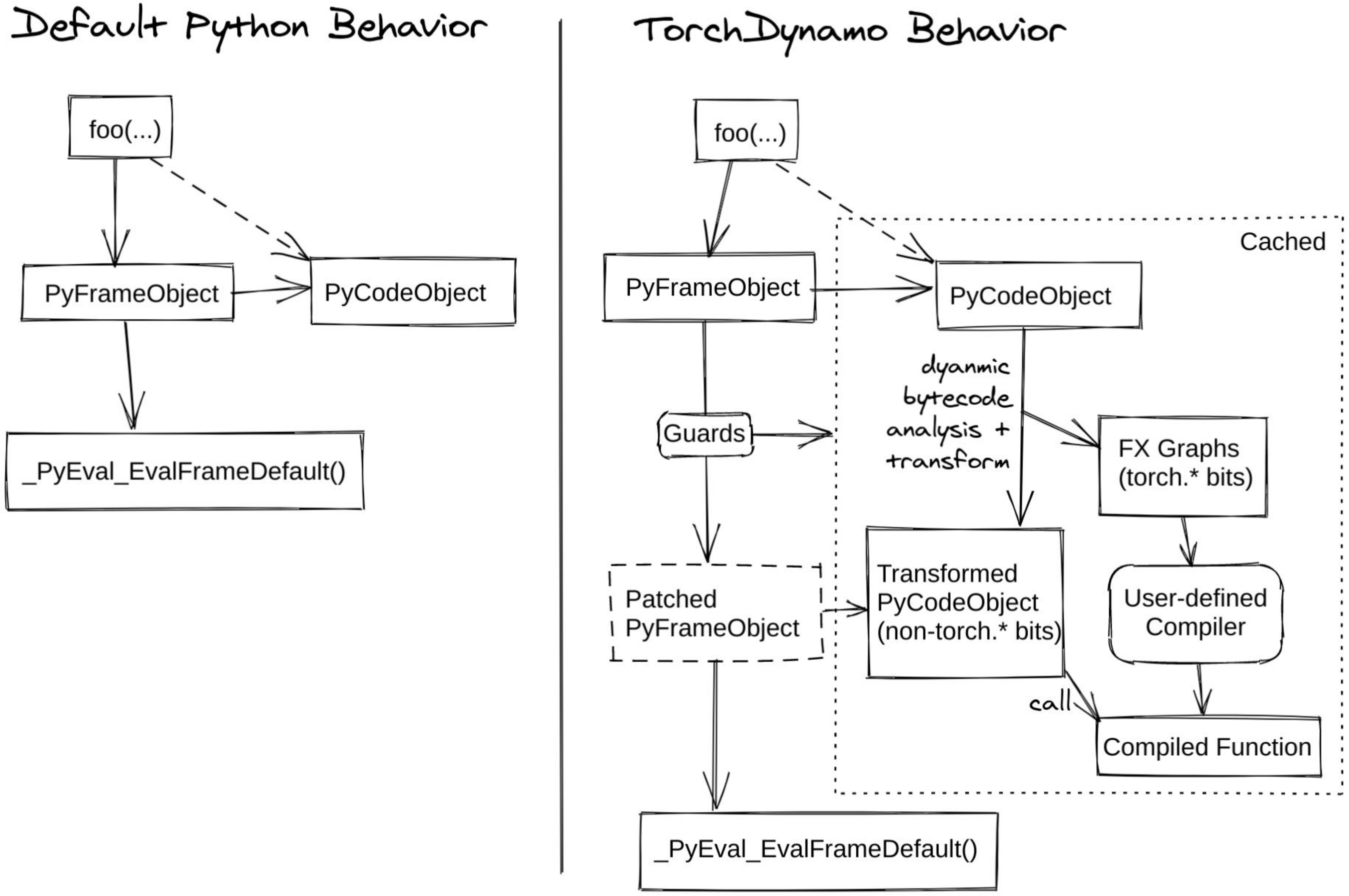
```python
def fn(x):
    a = torch.sin(x)
    b = torch.cos(x)
    return a * b

x = torch.randn(10)
opt_fn = torch._dynamo.optimize("eager")(fn)
opt_fn(x)
```



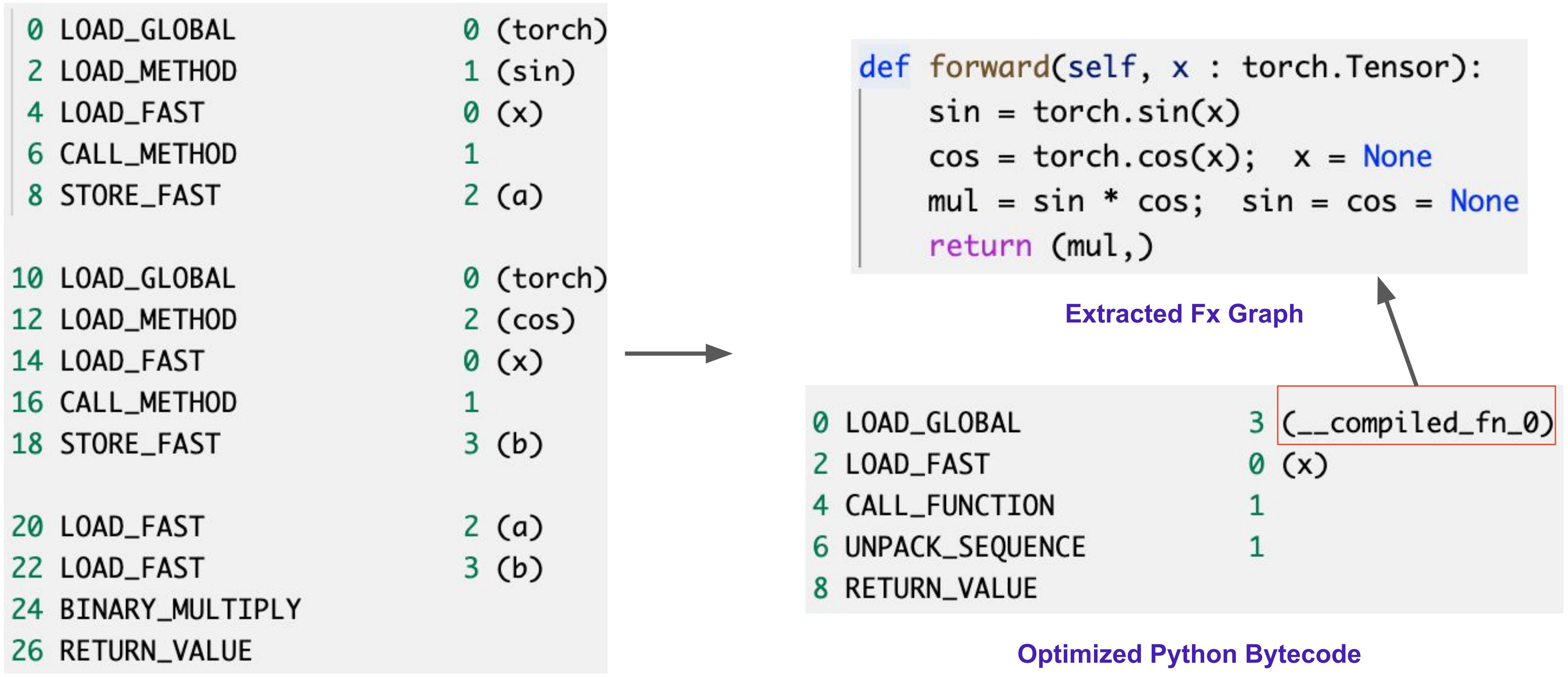
```
 0 LOAD_GLOBAL        0 (torch)
 2 LOAD_METHOD        1 (sin)
 4 LOAD_FAST          0 (x)
 6 CALL_METHOD        1
 8 STORE_FAST         2 (a)

10 LOAD_GLOBAL        0 (torch)
12 LOAD_METHOD        2 (cos)
14 LOAD_FAST          0 (x)
16 CALL_METHOD        1
18 STORE_FAST         3 (b)

20 LOAD_FAST          2 (a)
22 LOAD_FAST          3 (b)
24 BINARY_MULTIPLY
26 RETURN_VALUE
```

When opt_fn is called, TorchDynamo takes control:

- custom_eval_frame(PyFrameObject* frame)
  - frame->f_locals
    - {“x”: tensor([...]), “y”: tensor([...])}
  - frame->f_globals
    - {“torch”: …, …}
  - frame->f_code
    - Bytecode
    - …
  - …


Default Python Behavior
foo(...)
PyFrameObject
PyCodeObject
_PyEval_EvalFrameDefault()
TorchDynamo Behavior
foo(...)
Cached
PyFrameObject
PyCodeObject
dyanmic bytecode analysis + transform
Guards
FX Graphs (torch.* bits)
Patched PyFrameObject
Transformed PyCodeObject (non-torch.* bits)
User-defined Compiler
call
Compiled Function
_PyEval_EvalFrameDefault()

TorchDynamo: Bytecode Analysis and Graph Extraction



```
 0 LOAD_GLOBAL                0 (torch)
 2 LOAD_METHOD                1 (sin)
 4 LOAD_FAST                  0 (x)
 6 CALL_METHOD                1
 8 STORE_FAST                 2 (a)

10 LOAD_GLOBAL                0 (torch)
12 LOAD_METHOD                2 (cos)
14 LOAD_FAST                  0 (x)
16 CALL_METHOD                1
18 STORE_FAST                 3 (b)

20 LOAD_FAST                  2 (a)
22 LOAD_FAST                  3 (b)
24 BINARY_MULTIPLY
26 RETURN_VALUE
```

Original Python Bytecode

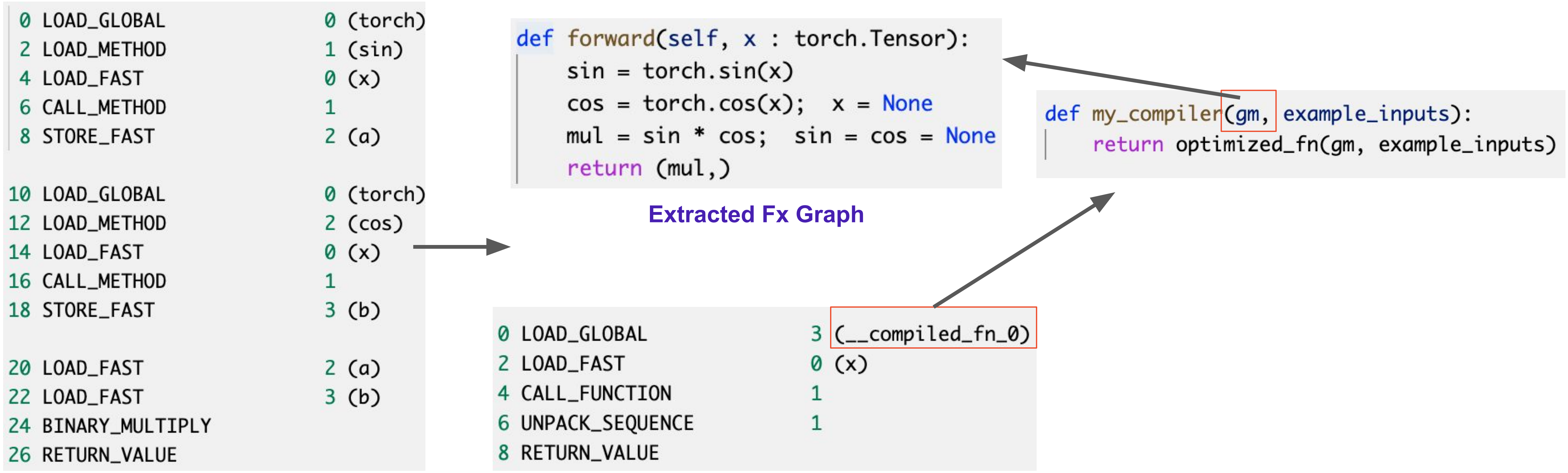
```python
def forward(self, x : torch.Tensor):
    sin = torch.sin(x)
    cos = torch.cos(x);   x = None
    mul = sin * cos;   sin = cos = None
    return (mul,)
```

Extracted Fx Graph

```
0 LOAD_GLOBAL                3 (__compiled_fn_0)
2 LOAD_FAST                  0 (x)
4 CALL_FUNCTION              1
6 UNPACK_SEQUENCE            1
8 RETURN_VALUE
```

Optimized Python Bytecode

TorchDynamo: Integration with Backend Compiler



```
 0 LOAD_GLOBAL              0 (torch)
 2 LOAD_METHOD              1 (sin)
 4 LOAD_FAST                0 (x)
 6 CALL_METHOD              1
 8 STORE_FAST               2 (a)

10 LOAD_GLOBAL              0 (torch)
12 LOAD_METHOD              2 (cos)
14 LOAD_FAST                0 (x)
16 CALL_METHOD              1
18 STORE_FAST               3 (b)

20 LOAD_FAST                2 (a)
22 LOAD_FAST                3 (b)
24 BINARY_MULTIPLY
26 RETURN_VALUE
```

```python
def forward(self, x : torch.Tensor):
    sin = torch.sin(x)
    cos = torch.cos(x);  x = None
    mul = sin * cos;  sin = cos = None
    return (mul,)
```

Extracted Fx Graph

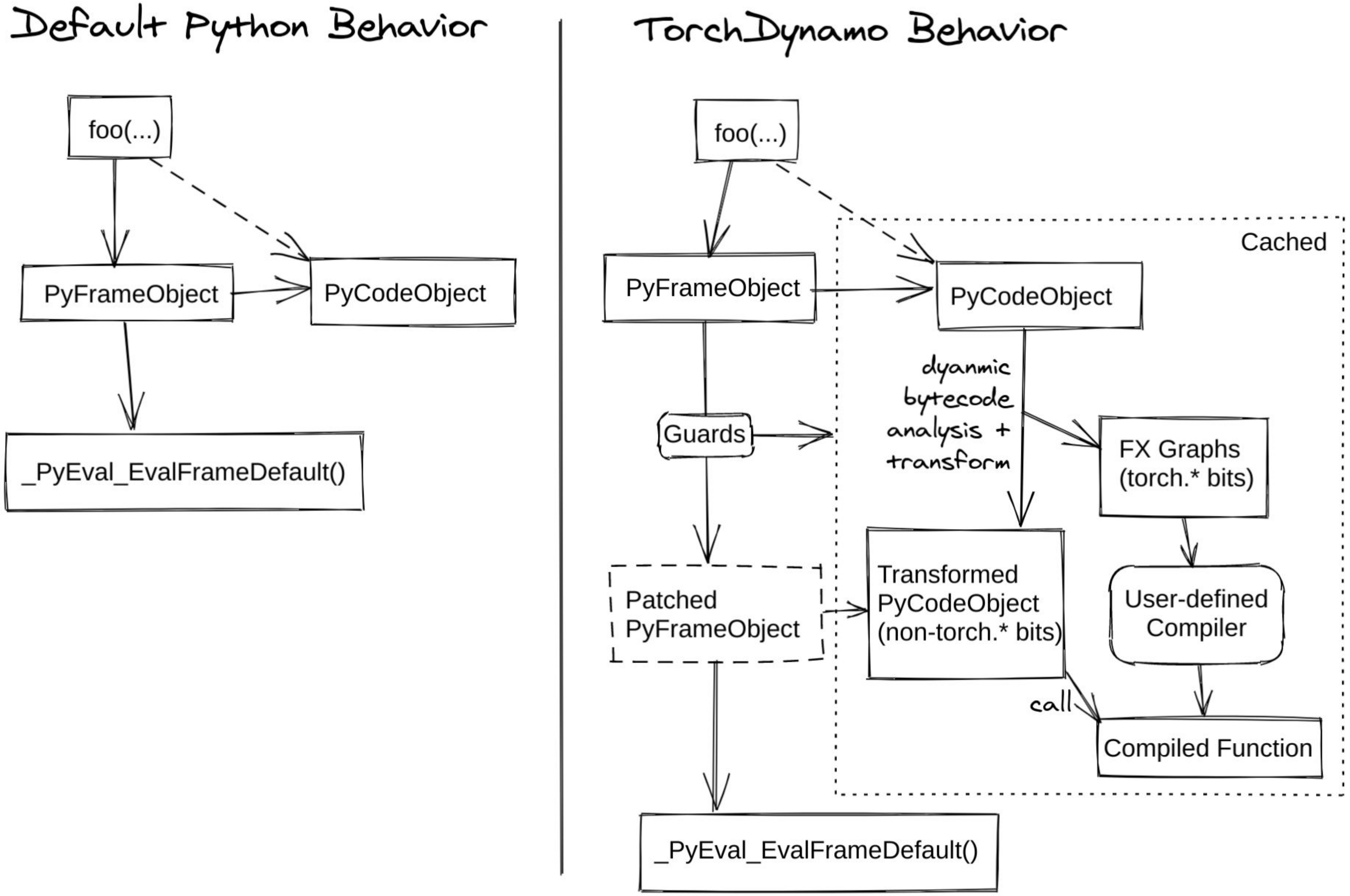
```python
def my_compiler(gm, example_inputs):
    return optimized_fn(gm, example_inputs)
```

```
0 LOAD_GLOBAL              3 (__compiled_fn_0)
2 LOAD_FAST                0 (x)
4 CALL_FUNCTION            1
6 UNPACK_SEQUENCE          1
8 RETURN_VALUE
```

Original Python Bytecode

Optimized Python Bytecode

Backend compilers accept a Fx graph module and return optimized callable


Default Python Behavior
foo(...)
PyFrameObject
PyCodeObject
_PyEval_EvalFrameDefault()
TorchDynamo Behavior
foo(...)
Cached
PyFrameObject
PyCodeObject
dyanmic bytecode analysis + transform
Guards
FX Graphs (torch.* bits)
Patched PyFrameObject
Transformed PyCodeObject (non-torch.* bits)
User-defined Compiler
call
Compiled Function
_PyEval_EvalFrameDefault()

TorchDynamo Concepts

Google Collab Demo

https://colab.research.google.com/drive/19JURKGhy_L82Y-2MUc2jurJwARCPy-YL?usp=sharing