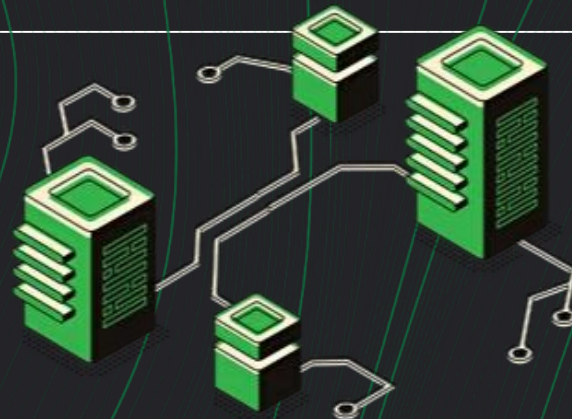


Обзор PyTorch и его экосистемы



Шпилевский Яромир

Ведущий разработчик First Line Software

Agenda

- PyTorch.
- «Вверх по программному стеку»: Lightning, Ignite.
- «Вниз по программному стеку»: TorchScript, C++.

Экосистема вокруг PyTorch

- Вокруг PyTorch развилась целая экосистема.
 - Как более высокоуровневые библиотеки и фреймворки,
 - так и есть опции, касающиеся низкоуровневой платформы для исполнения.



TorchScript



SKILLFACTORY

PyTorch



PyTorch. Истоки

- Авторы: Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan.
- Первый релиз – сентябрь 2016 года.
- Разрабатывается Facebook's AI Research lab (FAIR).
 - Detectron 2 [\[1\]](#) переписан с нуля на PyTorch.
- Автопилот Tesla использует PyTorch [\[2\]](#).
- Лицензия: модифицированная BSD.
- PyTorch [\[3\]](#) испытал влияние Torch [\[4\]](#), который тоже распространяется под лицензией BSD.
- PyTorch впитал в себя фреймворк Caffe 2 [\[5\]](#), который разрабатывался в University of California, Berkeley и тоже лицензируется под лицензией BSD (Berkeley Software Distribution).

PyTorch. Истоки. Caffe 2

- Convolutional Architecture for Fast Feature Embedding.
- Первый релиз – апрель 2017 года.
- Разработан в University of California, Berkeley.
- Создан Yangqing Jia в рамках диссертации на PhD.
- Изначально – для CNN архитектур.
- Позже были добавлена поддержка RNN, RCNN, LSTM и полносвязных архитектур.
- Написан на C++.
- Чуть позже была добавлена аппаратная акселерация, как через CUDA, так и через OpenCL.
- В марте 2018 наработки были влиты в кодовую базу PyTorch.

PyTorch. Истоки. Torch

- PyTorch испытал сильное влияние фреймворка Torch.
- Первый релиз – октябрь 2002 года.
- Авторы: Ronan Collobert, Samy Bengio, Johnny Mariéthoz.
- Активно использовался в Facebook's AI Research (FAIR).
 - FAIR разработал ещё несколько open source модулей.

PyTorch. Истоки. Torch. Native + Lua

- Ядро на C и C++, встроенная виртуальная машина Lua.
 - Различные реализации виртуальной машины языка Lua (Lua, LuaJIT, LuaVela) легко встраиваются в бинарные исполняемые файлы. [\[6\]](#) [\[7\]](#)
 - Исполняемые файлы могут быть скомпилированы из кода на C, C++, Rust.
- Lua – динамически типизированный язык с компактным синтаксисом.
 - Позволил создать среду, в которой легко и быстро пишется код.
 - Это позволяет легко и быстро прототипировать модели и проверять гипотезы.
- Были разработаны несколько Lua Rocks (модулей Lua) для научных вычислений.
 - `torch` – реализация тензоров и операций над ними (по большому счету).
 - `nn` – модели нейронных сетей (в PyTorch пакет так же называется `nn`).
- Чуть позже была добавлена поддержка аппаратного ускорения через CUDA.
 - Ядро на C легко могло осуществлять вызовы CUDA SDK.
 - Интерфейс CUDA SDK – процедуры на C.
- Был создан хороший программный стек от высокопроизводительного ядра до удобной среды для быстрого прототипирования моделей, ...

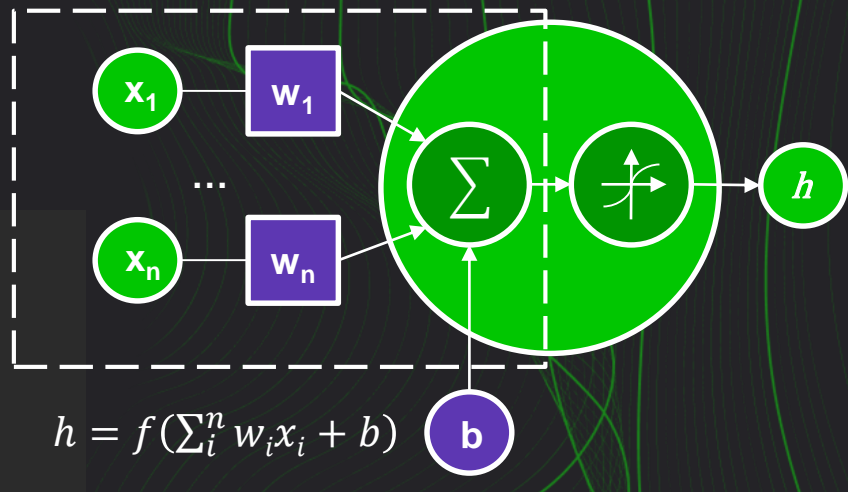
PyTorch. Концепция

- ... , но Lua – язык из 90-х.
- Что, если реализовать что-то похожее, но с более современным языком? Например, Python.
- Ядро PyTorch – на C++.
- С поддержкой аппаратной акселерации через CUDA.
- В качестве языка разработки моделей Python.
 - Тоже динамически типизированный язык с компактным синтаксисом.
 - Тоже позволяет быстро прототипировать архитектуры нейронных сетей и проверять гипотезы.
- Создан фреймворк на Python, предоставляющий «кубики» для создания моделей нейронных сетей: слои, функции активации и т.п.

PyTorch. Пример

ВХОДЫ => 1 ВЫХОД
Linear слоя

Linear слой –
несколько нейронов



```
...
# Пакет называется nn, как в Torch.
import torch.nn as nn
# Содержит различные функции,
# в т.ч. те, которые могут использоваться в качестве функций активации.
import torch.nn.functional as F
...

# Архитектура нейронной сети описывается классом,
# который наследуется от torch.nn.Module.
class Net(nn.Module):

    # Конструктор.
    def __init__(self):
        # Вызов конструктора суперкласса.
        super(Net, self).__init__()
        # Какие есть слои в нейронной сети.
        # Двумерные свёртки.
        self.conv1 = nn.Conv2d(16, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        # Dropout слои.
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        # Линейная трансформация. Кол-во входов -> кол-во выходов.
        # Входы и один выход - часть вычислительной модели нейрона до функции активации
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)
```

```
# Метод forward описывает прямое распространение сигнала.
# Как выходы предыдущего слоя соединены с входами последующих слоёв.
# Функции активации обычно тоже задаются тут.
def forward(self, x):
    x = self.conv1(x)
    x = F.relu(x)
    x = self.conv2(x)
    x = F.relu(x)
    x = F.max_pool2d(x, 2)
    x = self.dropout1(x)
    x = torch.flatten(x, 1)
    x = self.fc1(x)
    x = F.relu(x)
    x = self.dropout2(x)
    x = self.fc2(x)
    output = F.log_softmax(x, dim=1)
    return output
```

PyTorch. Необходимость в Ignite или Lightning

- Код на PyTorch сам по себе описывает только модель.
- Нет никакого управления жизненным циклом модели: процессом обучения, валидации и т.п.
- Долгое время жизненный цикл приходилось описывать самостоятельно.
- Ignite и Lightning предлагают два разных способа решения этой задачи.
- Ignite предлагает внешние сущности для управления жизненным циклом модели.
 - Событийный механизм: `ignite.engine.Engine` и `ignite.engine.Events`.
- Lightning предлагает структуру модели: кроме метода `forward` должны быть методы `configure_optimizers`, `training_step`, `validation_step` и т.п.

PyTorch Ignite.

- Набор внешних сущностей для управления жизненным циклом модели. [\[8\]](#)
- Позволяет упорядочить жизненный цикл: обучение модели, сохранение параметров, метрик, логов.
- `from ignite.engine import Engine`
- Предоставляет систему событий.
 - Например, сохранение параметров в конце эпохи обучения.
- `from ignite.engine import Events`
- Также реализует оценку метрик качества.
 - Precision
 - Recall
 - Confusion Matrix
 - ...
- `from ignite.metrics import Precision`

PyTorch Lightning.

- Структура модели на PyTorch. [\[9\]](#)
- `import pytorch_lightning as pl`
- `class SomeNnArch(pl.LightningModule):`
- `pl.LightningModule` определяет различные методы и жизненный цикл их вызова.
 - `forward` – прямое распространение сигналов.
 - `configure_optimizers` – какие оптимизаторы используются в модели
 - `training_step` – шаг обучения
 - `validation_step` – шаг валидации
 - ...



SKILLFACTORY

TorchScript и среда исполнения



Промежуточное представление TorchScript.

- Реализован Just-In-Time (JIT) компилятор кода на Python в промежуточное представление (Intermediate Representation (IR)) TorchScript. [\[10\]](#)
- TorchScript – статически типизированное подмножество Python.
- Это именно IR. Ориентирован на удобство для машины, не для человека. Не стоит пытаться писать код на TorchScript вручную.
- Можно как явно вызвать компиляцию всего участка кода, так и использовать трассирующую компиляцию.
- Полученный IR код на TorchScript можно использовать в бинарных модулях (могут компилироваться из кода на C, C++, Rust). [\[11\]](#)

TorchScript. Явный вызов компиляции

```
import torch

class SomeNnArch(torch.nn.Module):

    def __init__(self):
        super(SomeNnArch, self).__init__()
        self.linear = torch.nn.Linear(4, 4)

    def forward(self, x, h):
        new_h = torch.tanh(self.linear(x) + h)
        return new_h, new_h

# some_nn_arch - модель SomeNnArch в IR на TorchScript.
some_nn_arch = torch.jit.script(SomeNnArch())
# Можно получить код модели в IR на TorchScript.
print(some_nn_arch.code)
```

TorchScript. Трассирующая компиляция

```
import torch

class SomeNnArch(torch.nn.Module):

    def __init__(self):
        super(SomeNnArch, self).__init__()
        self.linear = torch.nn.Linear(4, 4)

    def forward(self, x, h):
        new_h = torch.tanh(self.linear(x) + h)
        return new_h, new_h

some_nn_arch = SomeNnArch()
x, h = torch.rand(3, 4), torch.rand(3, 4)
# Вызываем модель с конкретными входными значениями.
# JIT компилятор трассирует выполнение и генерирует соответствующий IR.
traced_cell = torch.jit.trace(some_nn_arch, (x, h))
print(traced_cell)
# Вызов тех же вычислений с помощью скомпилированной модели.
traced_cell(x, h)
```

TorchScript. Граф вычислений

```
print(traced_cell.graph)
```

```
graph(
  %self.1 : __torch__.SomeNnArch,
  %input : Float(3, 4, strides=[4, 1],
    requires_grad=0, device=cpu
  ),
  %h : Float(3, 4, strides=[4, 1], requires_grad=0, device=cpu):
  %21 : __torch__.torch.nn.modules.linear.Linear = prim::GetAttr[name="linear"](%self.1) %23 : Tensor =
  prim::CallMethod[name="forward"](%21, %input) %14 : int = prim::Constant[value=1]() # some_nn_arch.py:10:0
  %15 : Float(3, 4, strides=[4, 1], requires_grad=1, device=cpu) = aten::add(%23, %h, %14) # some_nn_arch.py:10:0
  %16 : Float(3, 4, strides=[4, 1], requires_grad=1, device=cpu) = aten::tanh(%15) # some_nn_arch.py:10:0
  %17 : (Float(3, 4, strides=[4, 1], requires_grad=1, device=cpu), Float(3, 4, strides=[4, 1], requires_grad=1, device=cpu)) =
  prim::TupleConstruct(%16, %16) return (%17)
```


- Иногда можно найти полезную информацию. ☺

TorchScript. Модель в IR

```
print(traced_cell.code)
```

```
def forward(  
    self,  
    input: Tensor,  
    h: Tensor  
)-> Tuple[Tensor, Tensor]:  
    _0 = torch.add((self.linear).forward(input, ), h, alpha=1)  
    _1 = torch.tanh(_0)  
    return (_1, _1)
```

Типы известны статически.



TorchScript. Модель в IR можно использовать из C++

Python

```
traced_cell.save("model.pt")
```

C++

```
// Всё, что нужно для работы с TorchScript – в одном
// заголовке.
#include <torch/script.h>

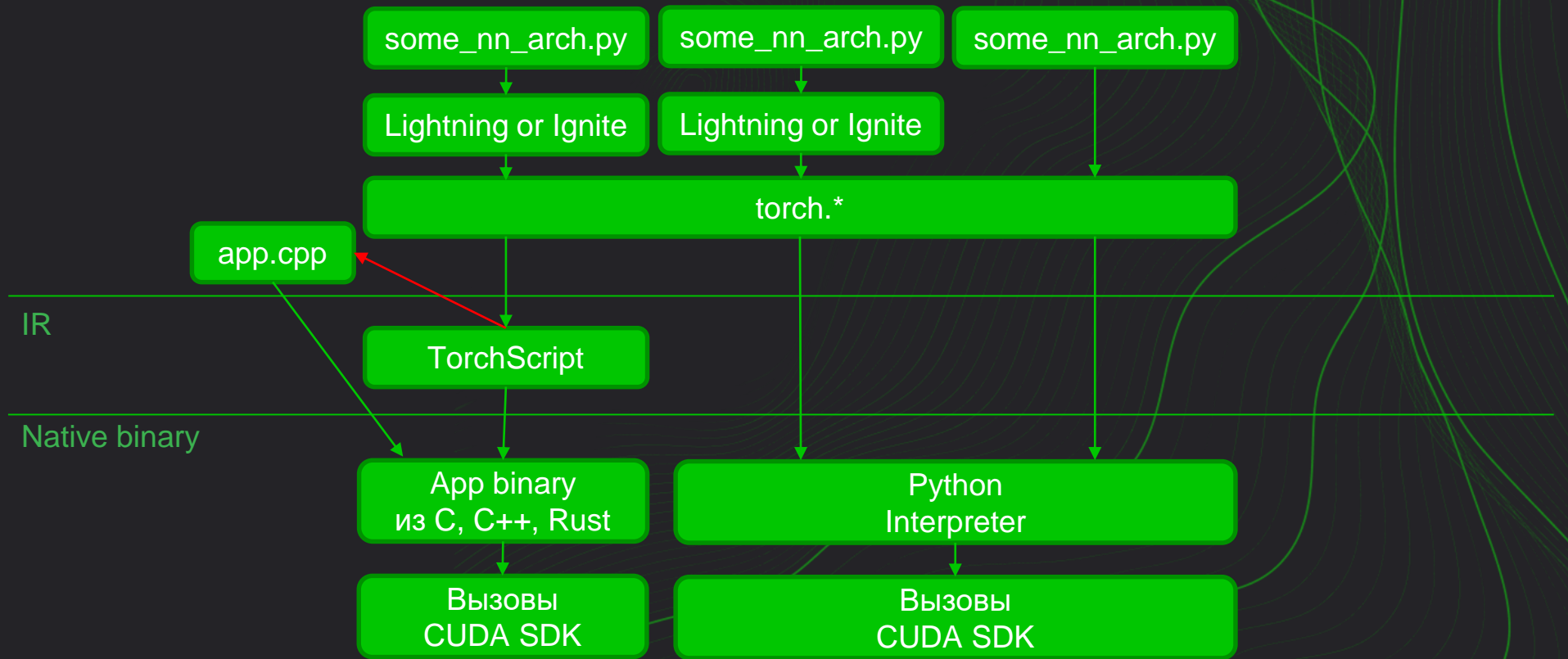
#include <iostream>
#include <memory>

int main(int argc, const char* argv[]) {

    torch::jit::script::Module module;
    try {
        // Deserialize the ScriptModule from a file using
        torch::jit::load().
        module = torch::jit::load("model.pt");
    }
    catch (const c10::Error& e) {
        ...
    }
}
```

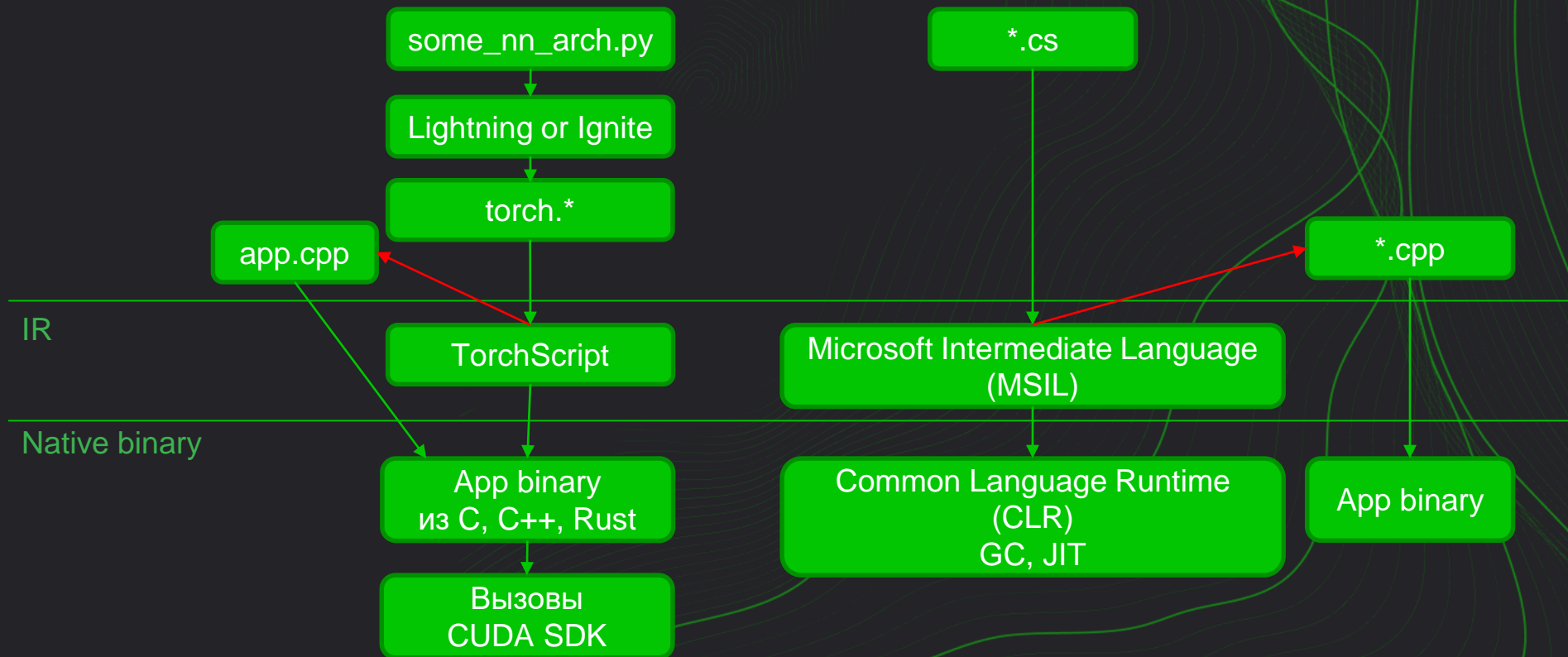

TorchScript. Стек среды исполнения

Source



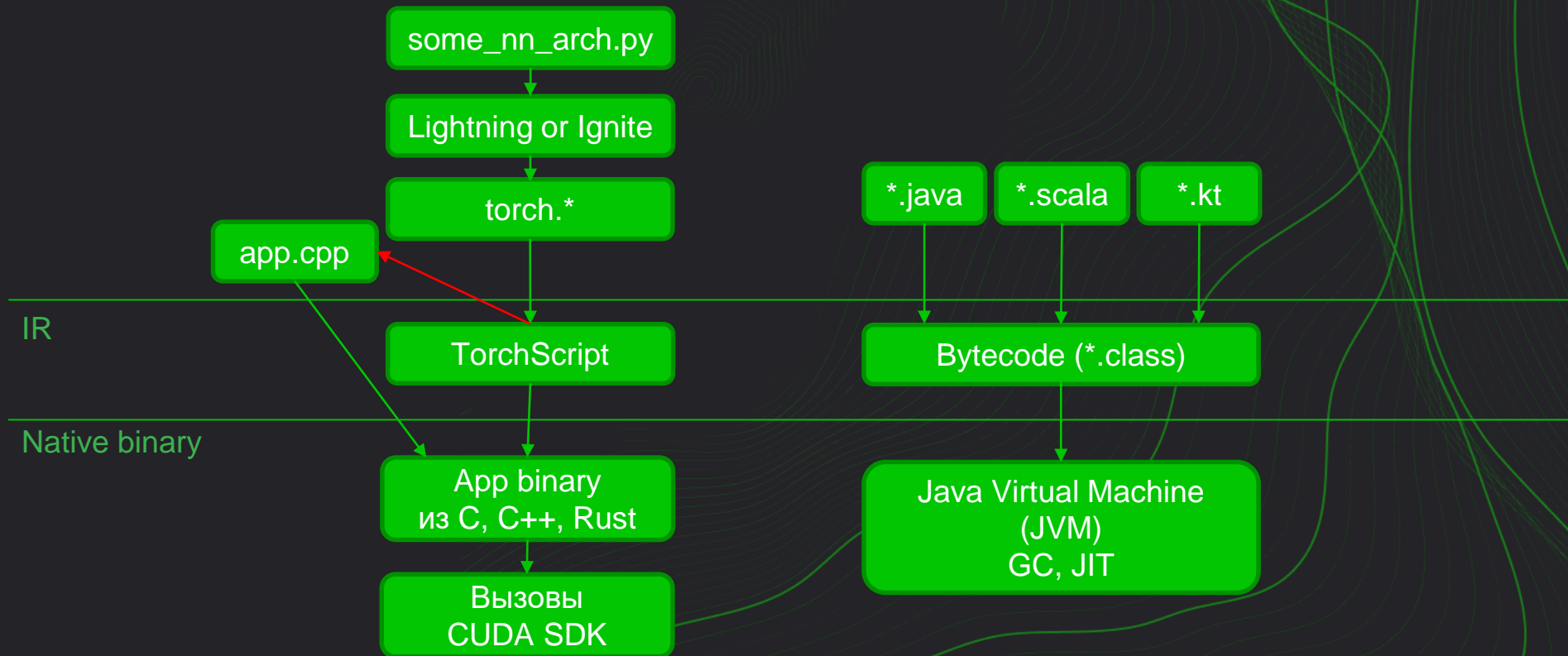
TorchScript. Стек среды исполнения. .NET для аналогии

Source



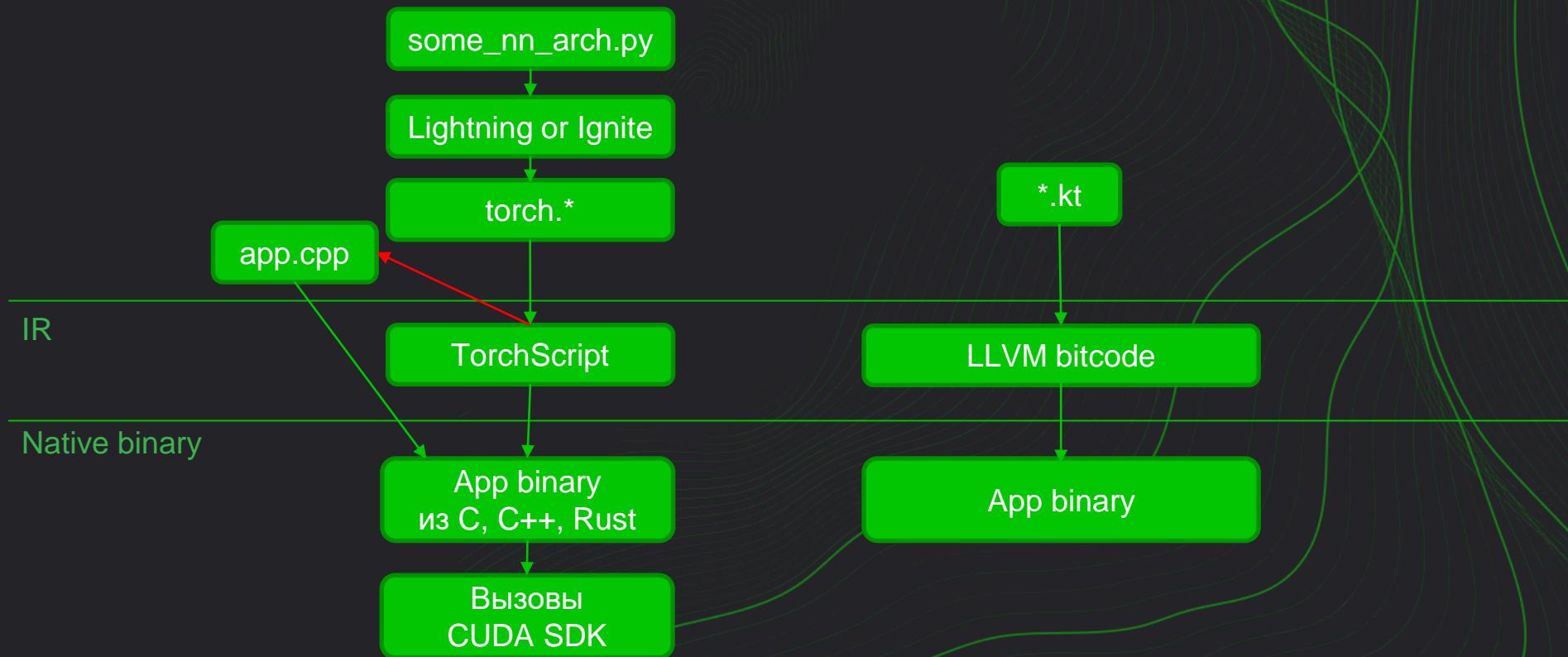
TorchScript. Стек среды исполнения. JVM для аналогии

Source



TorchScript. Стек среды исполнения. Kotlin Native для аналогии

Source



Ссылки

1. Facebook's AI Research Detectron2 (<https://github.com/facebookresearch/detectron2>)
2. PyTorch at Tesla (<https://www.youtube.com/watch?v=oBklItKXtDE>)
3. PyTorch (<https://github.com/pytorch/pytorch>)
4. Torch (<https://github.com/torch/torch7>)
5. Caffe 2 (<https://github.com/facebookarchive/caffe2>)
6. Использовать Lua с C++ легче, чем вы думаете. Tutorial по LuaBridge (<https://habr.com/ru/post/237503/>)
7. Embed Lua for scriptable apps (<https://developer.ibm.com/technologies/systems/tutorials/l-embed-lua/>)
8. PyTorch Ignite (<https://github.com/pytorch/ignite>)
9. PyTorch Lightning (<https://github.com/PyTorchLightning/pytorch-lightning>)
10. Introduction to TorchScript (https://pytorch.org/tutorials/beginner/Intro_to_TorchScript_tutorial.html)
11. Loading a TorchScript Model in C++ (https://pytorch.org/tutorials/advanced/cpp_export.html)

Резюме

- Рассмотрен PyTorch.
- Рассмотрены различные надстройки «вверх по программному стеку»:
 - Lightning.
 - Ignite.
- Рассмотрено устройство PyTorch «вниз по программному стеку»:
 - Рассмотрено промежуточное представление TorchScript.
 - Рассмотрено устройство native binary, компилируемых из C++, со вставками моделей на TorchScript.

Вопросы для самоконтроля

- Какие надстройки PyTorch вы можете назвать?
- Что такое TorchScript?
- Из каких компонентов компилируется native binary (если использовать эту возможность PyTorch)?

Спасибо!

