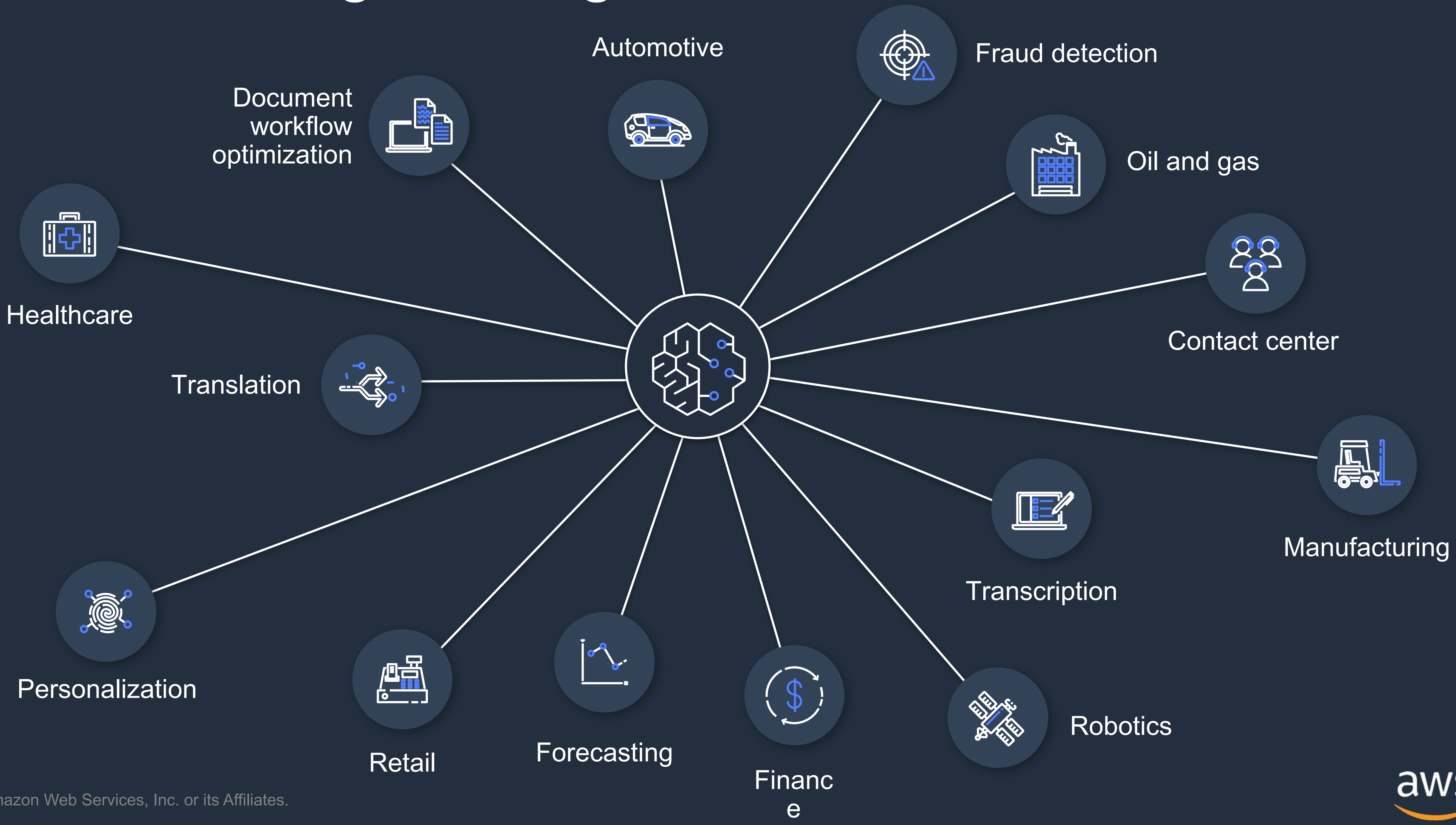
Preserving Developer Efficiency in the Adoption of New AI Hardware

Julien Simon
Global Technical Evangelist, AI & Machine Learning, AWS
@julsimon

Machine learning is eating the world



The advent of machine learning and deep learning

Large digital
datasets

User-generated content
Email, posts,
product reviews,
images, videos

Web and mobile customer data
Search queries,
views, clicks, purchases

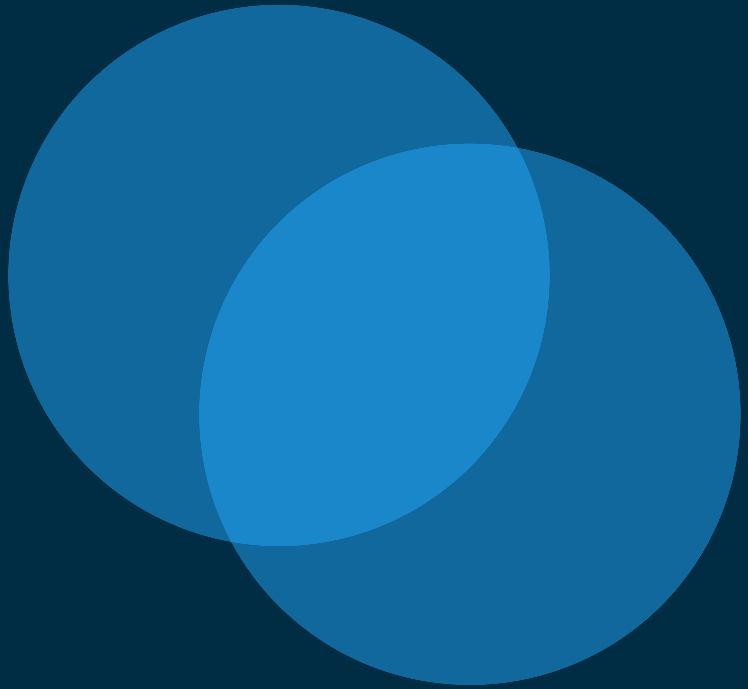
Open datasets
ImageNet,
WMT,
Medical
datasets

The advent of machine learning and deep learning

Large digital
datasets

Commodity hardware
Servers, storage, networking

Cloud computing
Built-in security and resilience
On-demand resources
Pay as you go
Managed services

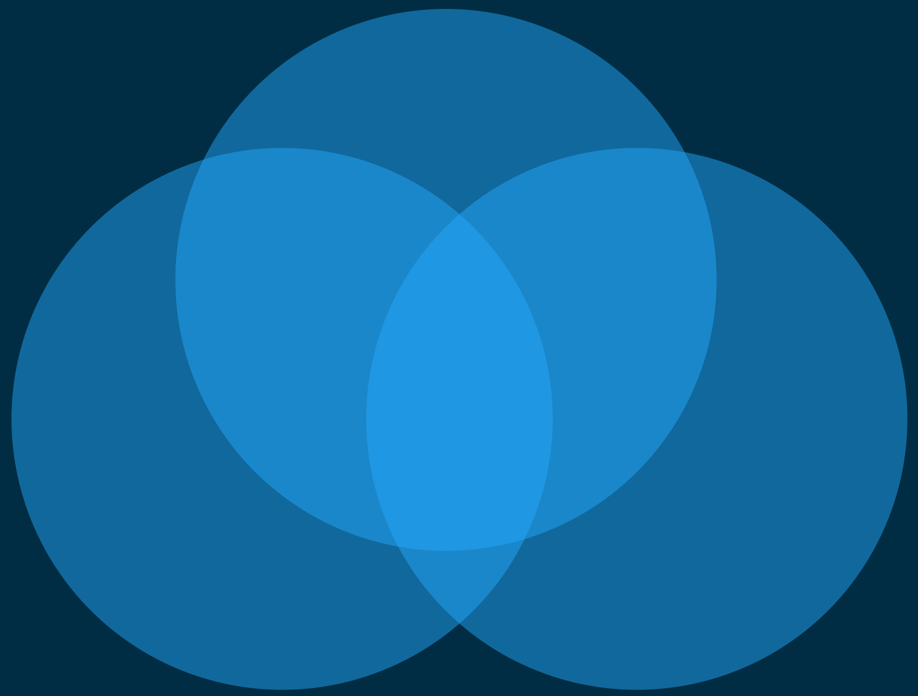


Scalable and cost-effective
infrastructure

96% of deep learning is running in a cloud environment,
89% of this runs on AWS
Nucleus Report, October 2019

The advent of machine learning and deep learning

Large digital
datasets



Open source

development

tools

Scalable and cost-effective
infrastructure

Torch, Theano
Hadoop, Spark
TensorFlow,Keras, MXNet, PyTorch

The advent of machine learning and deep learning

Large digital datasets

Open source development tools

Scalable and cost-effective infrastructure

Hardware acceleration

Nvidia GPUs + CUDA
Intel AVX
FGPAs
ASICs

Putting it all together

August 1999 - Nvidia launches the first GPU (GeForce 256)

August 2005 - “*Using GPUs for Machine Learning Algorithms*“, Steinkrau et al.

6 years

March 2006 - Amazon Web Services is launched. EC2 and S3 go live

June 2007 - CUDA 1.0 is released

June 2009 - “*Large-scale Deep Unsupervised Learning using Graphics Processors*”, Raina et al.

November 2010 - Theano 0.3 supports GPU training with CUDA

November 2010 - AWS launches GPU instances

1 year

2011-2012 - deep learning achieves superhuman performance on computer vision tasks (such as ImageNet)

2015-2016 - TensorFlow, Keras, PyTorch, and Apache MXNet are released

December 2016 - AWS launches FPGA instances

December 2018 - AWS launches Inferentia, a custom chip designed for high-throughput and low-cost inference

Today

State of the art models implemented with open source libraries are shared on Github or on the AWS Marketplace.

Within minutes, developers can deploy them on a wide selection of AWS instances (CPU, GPU, FPGA, Inferentia), and start testing them at minimal cost.

So, what does it take for new hardware platforms to win the hearts and minds of developers?

Observations

Hardware innovation is critical for ML performance, but…

- Most ML practitioners neither know much about hardware nor care much for it
- Ideally, it should be completely transparent

Developer-friendly tools are paramount for adoption

- Providing an SDK abstracting the hardware is important, but it's not enough
- That SDK must work with the tools and languages that ML practioners use
  - Through direct integration
  - Through a simple toolchain that doesn't require any hardware knowledge
- Make it as easy as possible to use your tools in the cloud

Don't assume that you know best

- Engage with potential users and collect as much feedback as possible on languages, tools, and workflows
- Communities, not community: ✅ scientists ✅ researchers ✅ data scientists ✅ ML engineers ✅ ML ops
- Attend meetups, write blog posts, push sample code to Github, hire Developer Advocates!

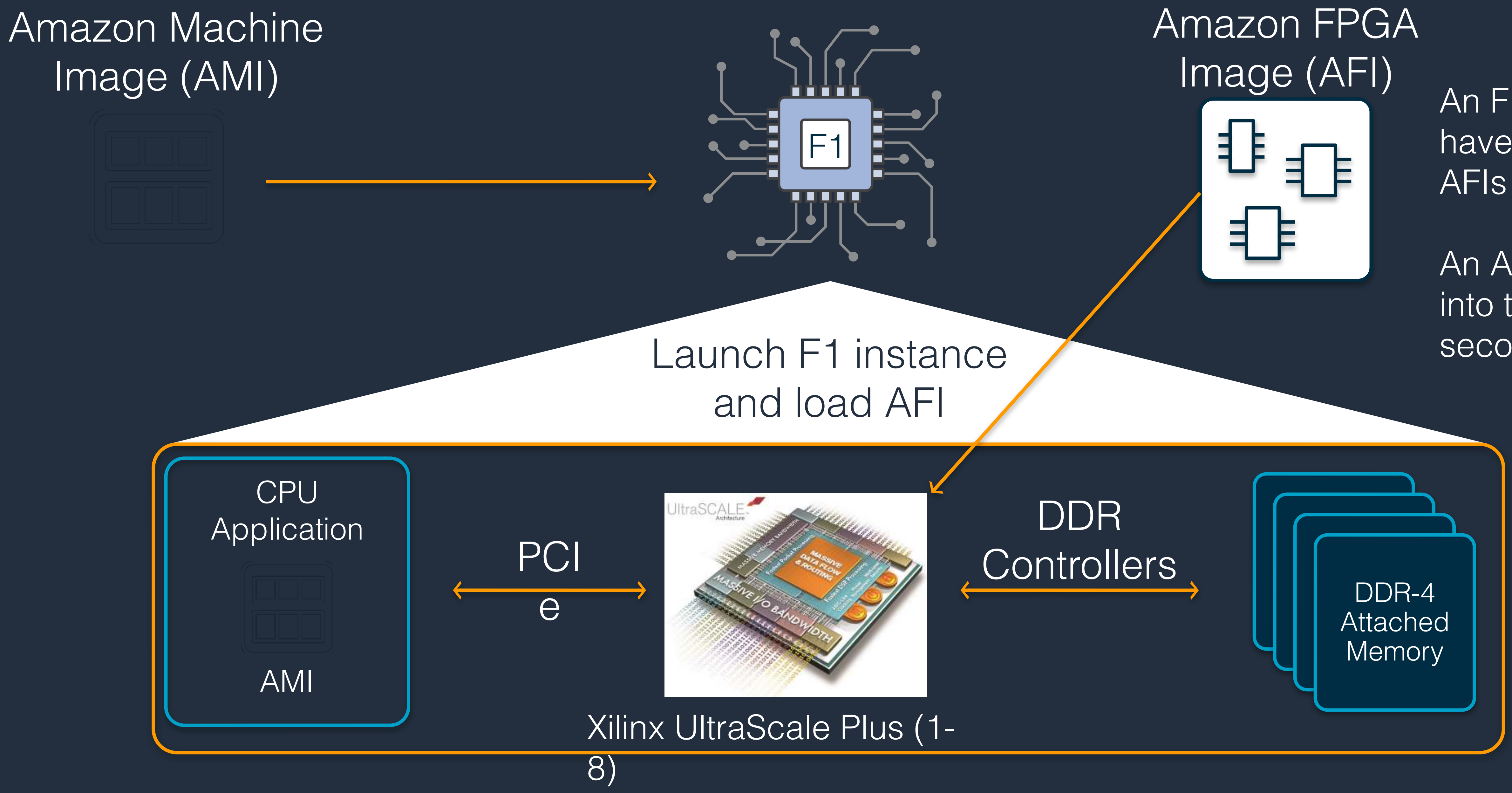
Let’s look at three examples in recent AWS history

Amazon EC2 F1 instances

Amazon SageMaker Neo

AWS Inferentia

Example #1: FPGA acceleration using EC2 F1 instances



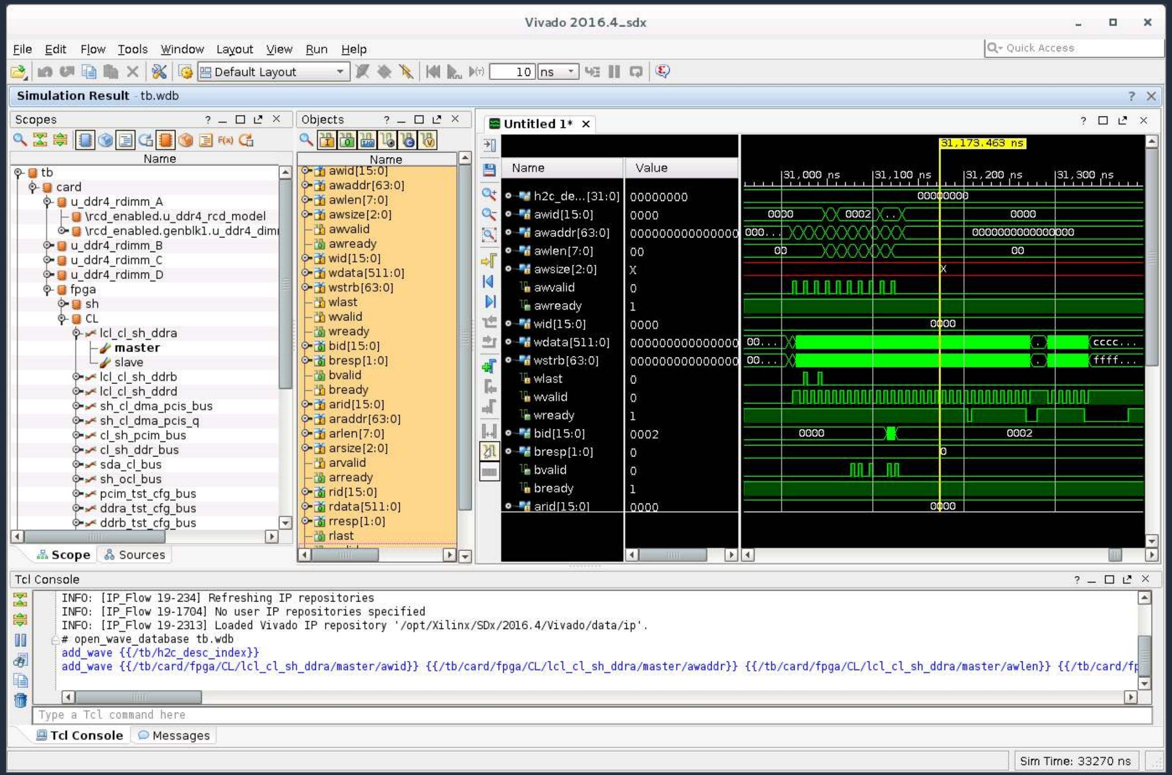
An F1 instance can have any number of AFIs

An AFI can be loaded into the FPGA in seconds

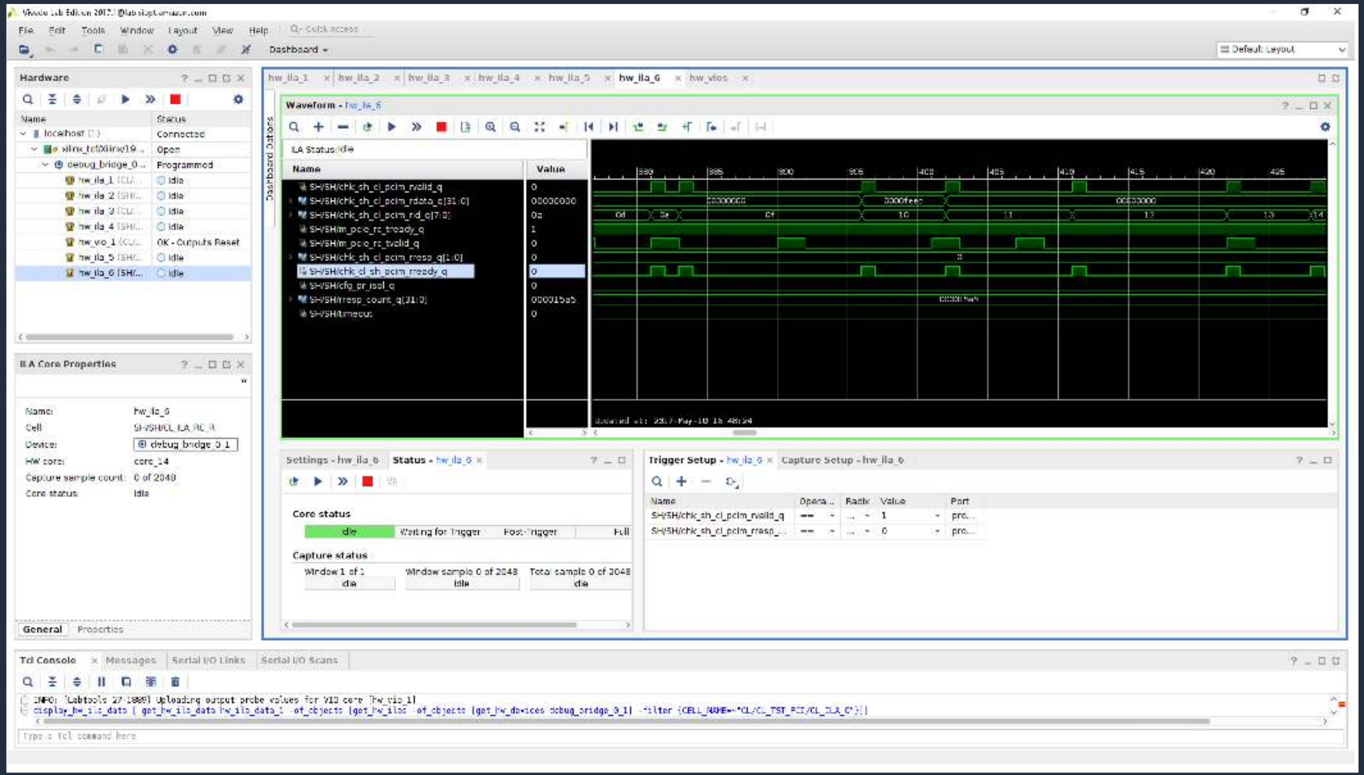
AWS FPGA developer AMI

Pre-integrated environment, spun up in minutes with latest compute servers
Use Xilinx Vivado (RTL or SDAccel) to describe and simulate your FPGA logic

Xilinx Vivado for custom logic development



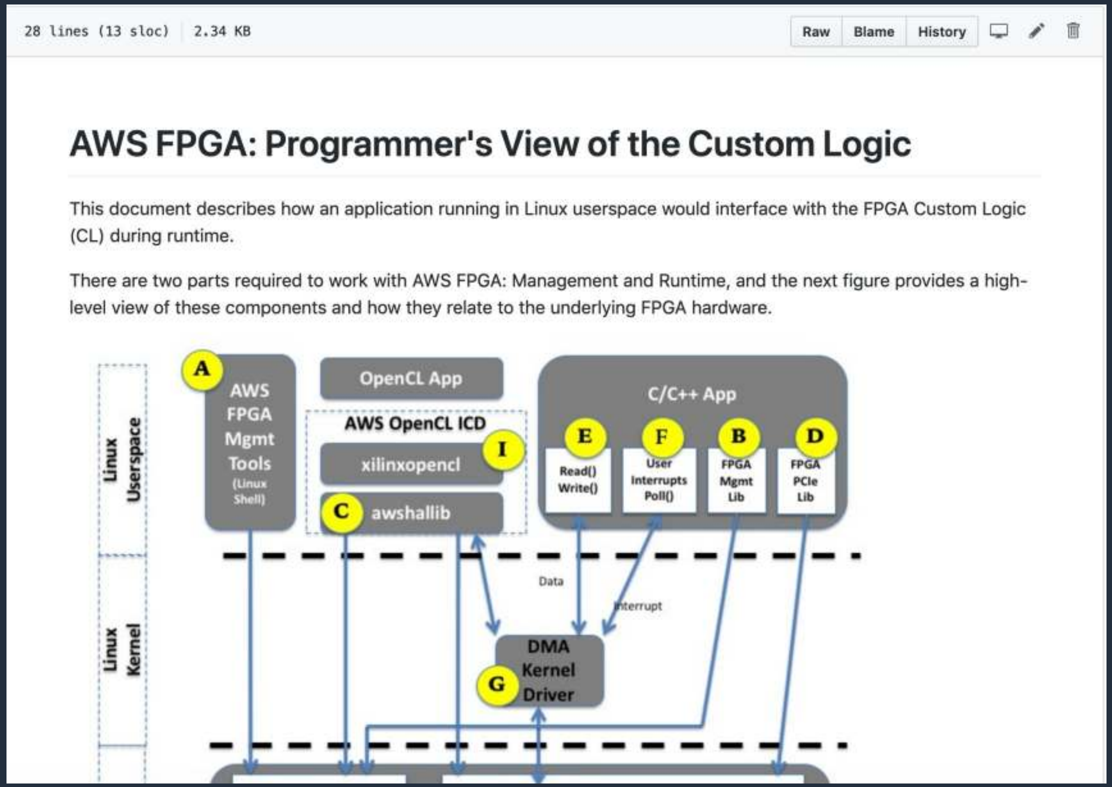
Virtual JTAG for interactive debugging



AWS FPGA developer GitHub

https://github.com/aws/aws-fpga/

AWS FPGA GitHub contains all the drivers, code, examples, and tutorials needed to develop hardware acceleration for the AWS FPGAs



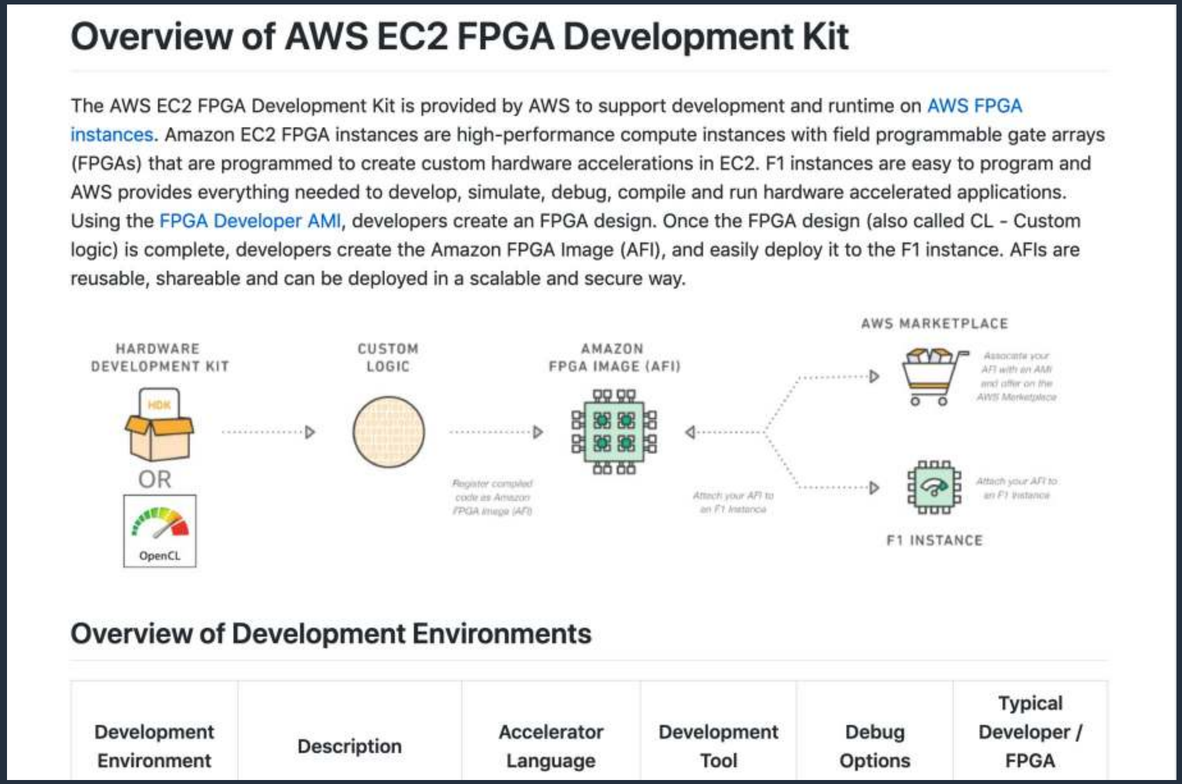
28 lines (13 sloc) | 2.34 KB

Raw | Blame | History

AWS FPGA: Programmer's View of the Custom Logic

This document describes how an application running in Linux userspace would interface with the FPGA Custom Logic (CL) during runtime.

There are two parts required to work with AWS FPGA: Management and Runtime, and the next figure provides a high-level view of these components and how they relate to the underlying FPGA hardware.



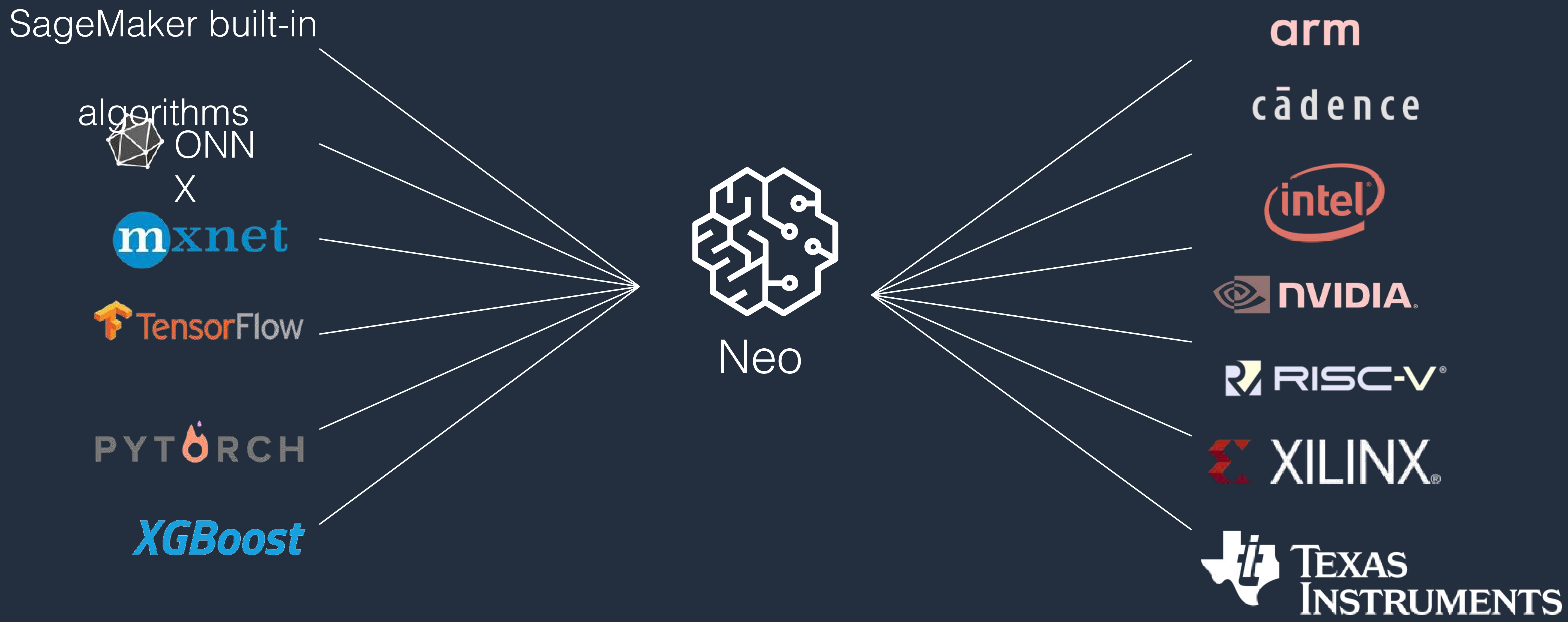
Overview of AWS EC2 FPGA Development Kit

The AWS EC2 FPGA Development Kit is provided by AWS to support development and runtime on AWS FPGA instances. Amazon EC2 FPGA instances are high-performance compute instances with field programmable gate arrays (FPGAs) that are programmed to create custom hardware accelerations in EC2. F1 instances are easy to program and AWS provides everything needed to develop, simulate, debug, compile and run hardware accelerated applications. Using the FPGA Developer AMI, developers create an FPGA design. Once the FPGA design (also called CL - Custom logic) is complete, developers create the Amazon FPGA Image (AFI), and easily deploy it to the F1 instance. AFIs are reusable, shareable and can be deployed in a scalable and secure way.

Overview of Development Environments

Development Environment	Description	Accelerator Language	Development Tool	Debug Options	Typical Developer / FPGA

Example #2: train once, run anywhere with SageMaker Neo



Compiling with Neo: one line of code

Parses Model

Convert an input model into a common format

Optimizes Graph

Detect patterns in the ML model structure to reduce the execution time

Optimizes Tensors

Detect patterns in the shape of input data to allocate memory efficiently

Generates Code

Use a low-level compiler to generate machine code for each target

```
ic_neo_model = ic.compile_model(
    target_instance_family='rasp3b',
    input_shape={'data':[1, 3, 224, 224]},
    role=role,
    framework='mxnet',
    framework_version='1.5.1',
    output_path=output_path)
```

Deploying with Neo: one line of code

Dispatches Model

Matches model with execution backend

TVM | Treelite | TFLite

Partitions Graph

Sends sub-graph to appropriate accelerator

CPU | GPU | ASIC

Deploy on Amazon SageMaker

```
ic_neo_predictor = ic_neo_model.deploy(
    endpoint_name=ic_neo_endpoint_name,
    initial_instance_count=1,
    instance_type='ml.c5.4xlarge')
```

Deploy on an embbeded platform with the Deep Learning Runtime https://github.com/neo-ai/neo-ai-dlr

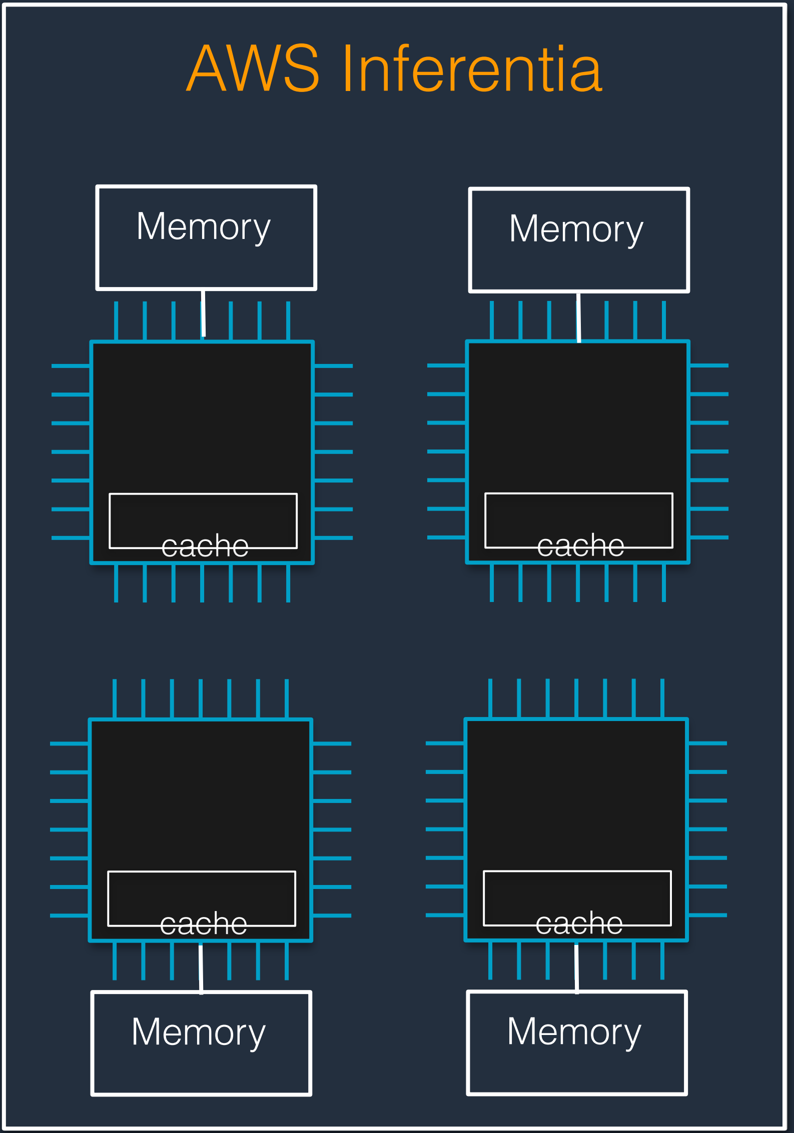
```
from dlr import DLRModel

model = DLRModel(model_path)
```

aws

Example #3: AWS Inferentia

- 4 NeuronCores
- Up to 128 TOPS
- 2-stage memory hierarchy:
  large on-chip cache and commodity DRAM
- Supports FP16, BF16, INT8 data types with mixed precision
- Fast chip-to-chip interconnect
- Hosted in EC2 Inf1 instances (1 to 4 chips)



AWS Neuron

High-performance Software Development Kit (SDK)

Neuron Compiler

Neuron Runtime

Profiling tools

Easy to get started

Integrated with major frameworks

TensorFlow mxnet PyTorch

AWS Neuron

Flexibility

Compile and deploy models on AWS services with just a couple of lines of code

Documentation, examples & support

https://github.com/aws/aws-neuron-sdk

Getting started

https://aws.amazon.com/ec2/instance-types/f1/
https://github.com/aws/aws-fpga/

https://aws.amazon.com/sagemaker
https://aws.amazon.com/sagemaker/neo/

https://aws.amazon.com/ec2/instance-types/inf1/
https://aws.amazon.com/machine-learning/inferentia/

https://medium.com/@julsimon
https://youtube.com/juliensimonfr

Published August 2020

Discount link for the paper edition on Amazon (US only):
https://www.amazon.com/gp/mpc/AOHJSZC7A0AV5

Discount code for the e-book edition on Packt:
20SAGEMAKER
https://www.packtpub.com/product/learn-amazon-sagemaker/9781800208919

Valid until November 11th

Thank you!

Julien Simon
Global Technical Evangelist, AI & Machine Learning, AWS
@julsimon