# Bring Your Own Datatypes

Gus Smith
University of Washington

sampl

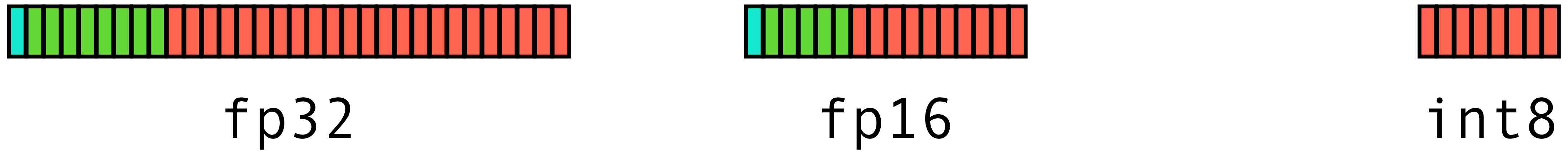
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

tvm.ai

# Why?

# Why?

**Data type**



# Why?

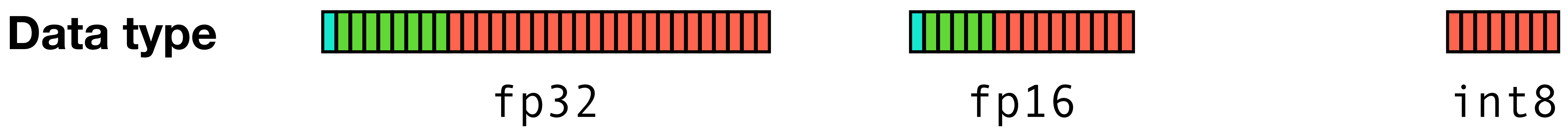

## Beating Floating Point at its Own Game: Posit Arithmetic

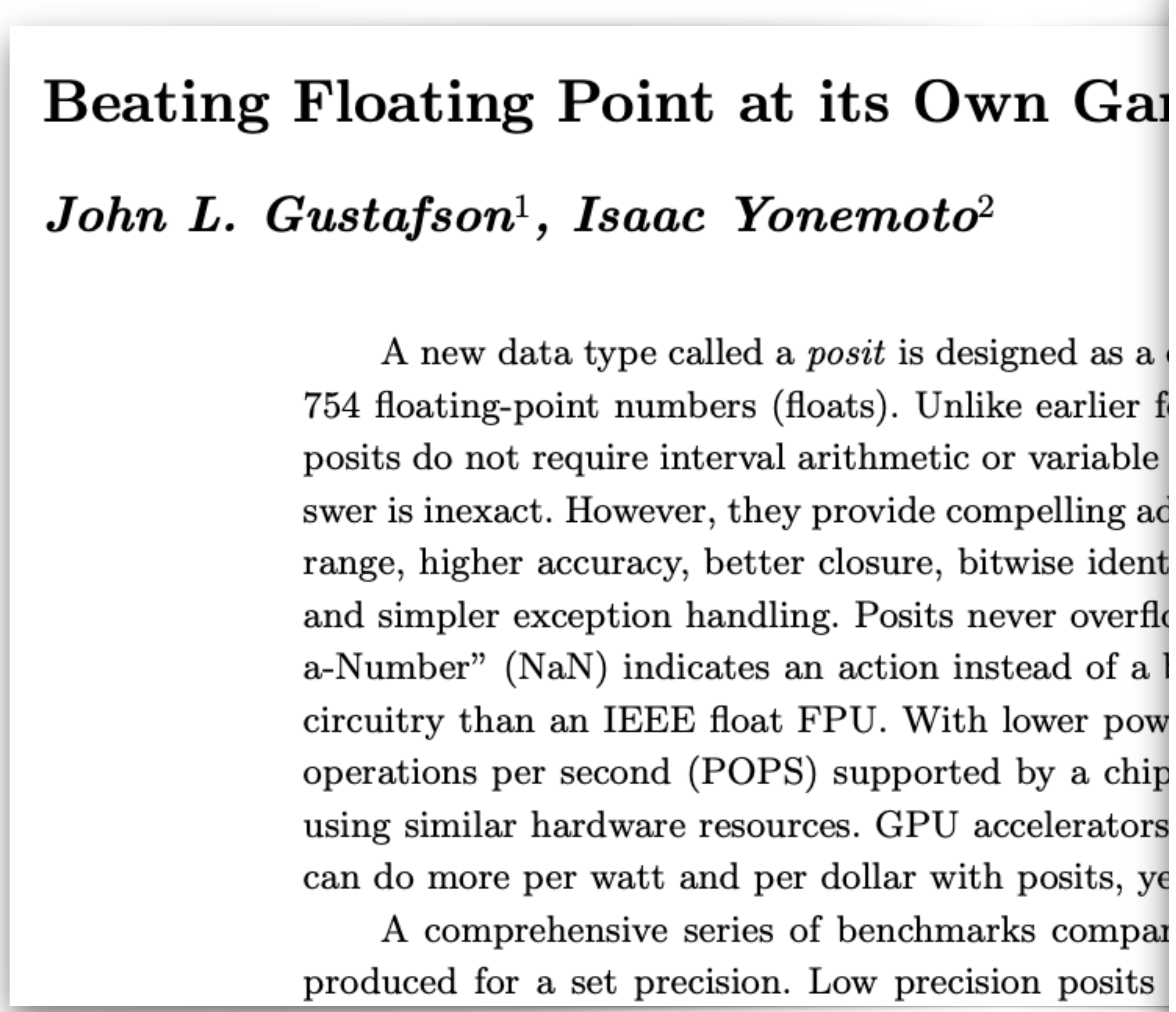
*John L. Gustafson*[1], *Isaac Yonemoto*[2]

A new data type called a *posit* is designed as a direct drop-in replacement for IEEE Standard 754 floating-point numbers (floats). Unlike earlier forms of universal number (unum) arithmetic, posits do not require interval arithmetic or variable size operands; like floats, they round if an answer is inexact. However, they provide compelling advantages over floats, including larger dynamic range, higher accuracy, better closure, bitwise identical results across systems, simpler hardware, and simpler exception handling. Posits never overflow to infinity or underflow to zero, and "Not-a-Number" (NaN) indicates an action instead of a bit pattern. A posit processing unit takes less circuitry than an IEEE float FPU. With lower power use and smaller silicon footprint, the posit operations per second (POPS) supported by a chip can be significantly higher than the FLOPS using similar hardware resources. GPU accelerators and Deep Learning processors, in particular, can do more per watt and per dollar with posits, yet deliver superior answer quality.

A comprehensive series of benchmarks compares floats and posits for decimals of accuracy produced for a set precision. Low precision posits provide a better solution than "approximate

# Why?

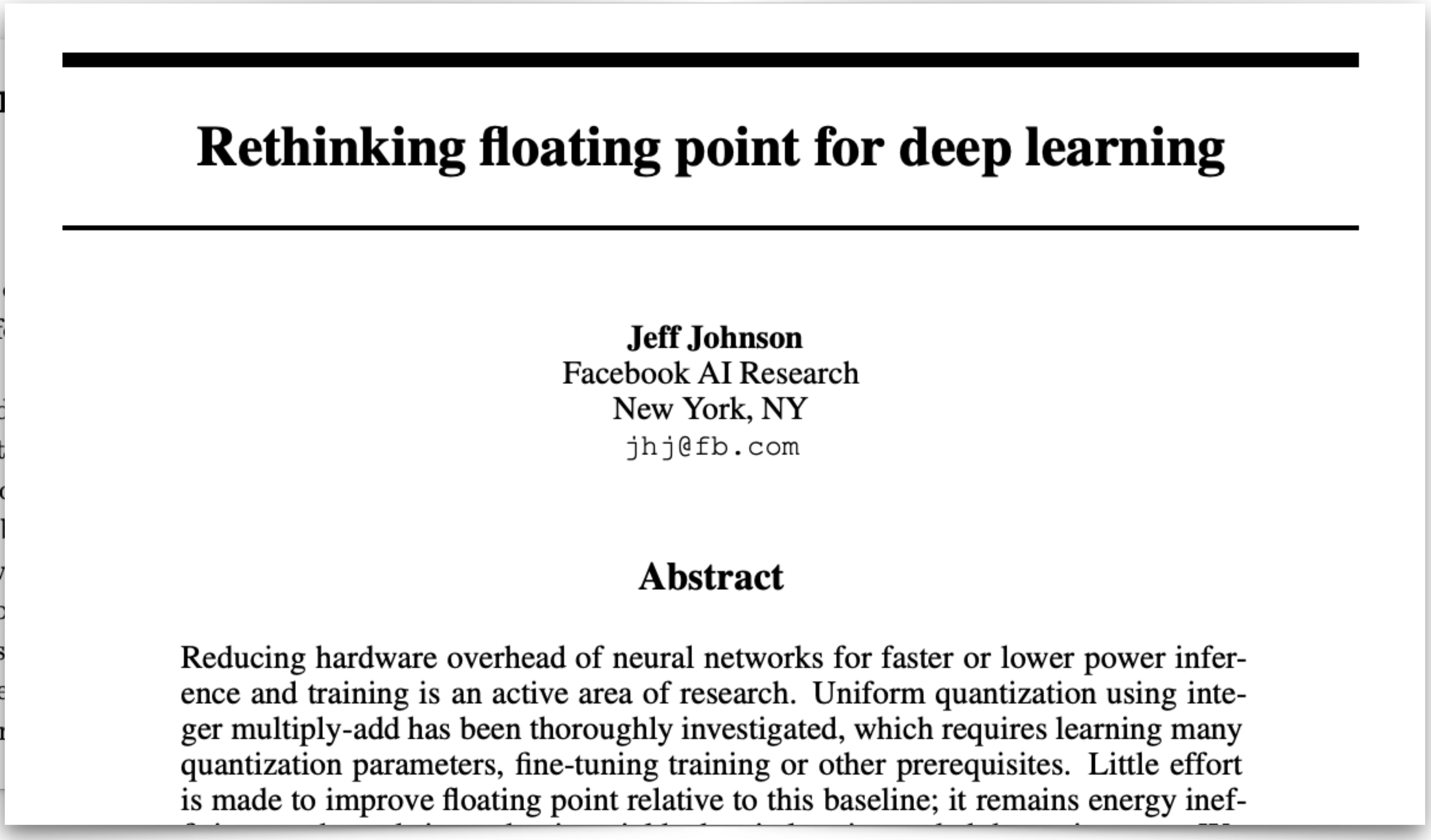


Beating Floating Point at its Own Ga

*John L. Gustafson*[1], *Isaac Yonemoto*[2]

A new data type called a *posit* is designed as a
754 floating-point numbers (floats). Unlike earlier f
posits do not require interval arithmetic or variable
swer is inexact. However, they provide compelling ad
range, higher accuracy, better closure, bitwise ident
and simpler exception handling. Posits never overfl
a-Number" (NaN) indicates an action instead of a
circuitry than an IEEE float FPU. With lower pow
operations per second (POPS) supported by a chip
using similar hardware resources. GPU accelerators
can do more per watt and per dollar with posits, ye
A comprehensive series of benchmarks compa
produced for a set precision. Low precision posits



**Rethinking floating point for deep learning**

**Jeff Johnson**
Facebook AI Research
New York, NY
jhj@fb.com

**Abstract**

Reducing hardware overhead of neural networks for faster or lower power inference and training is an active area of research. Uniform quantization using integer multiply-add has been thoroughly investigated, which requires learning many quantization parameters, fine-tuning training or other prerequisites. Little effort is made to improve floating point relative to this baseline; it remains energy inef-

# The Goal: BYO-Datatype

# The Goal: BYO-Datatype

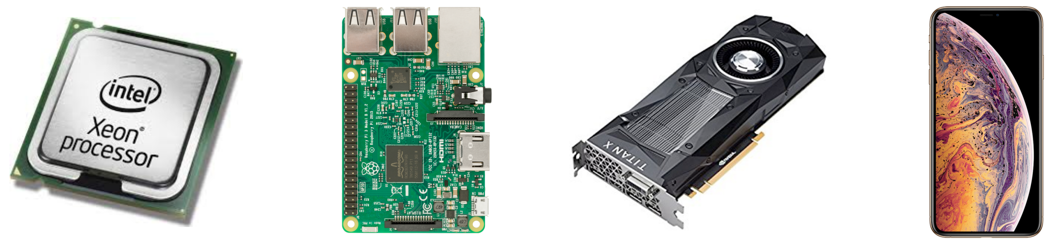
**+ your custom datatypes!**

High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal

VTA



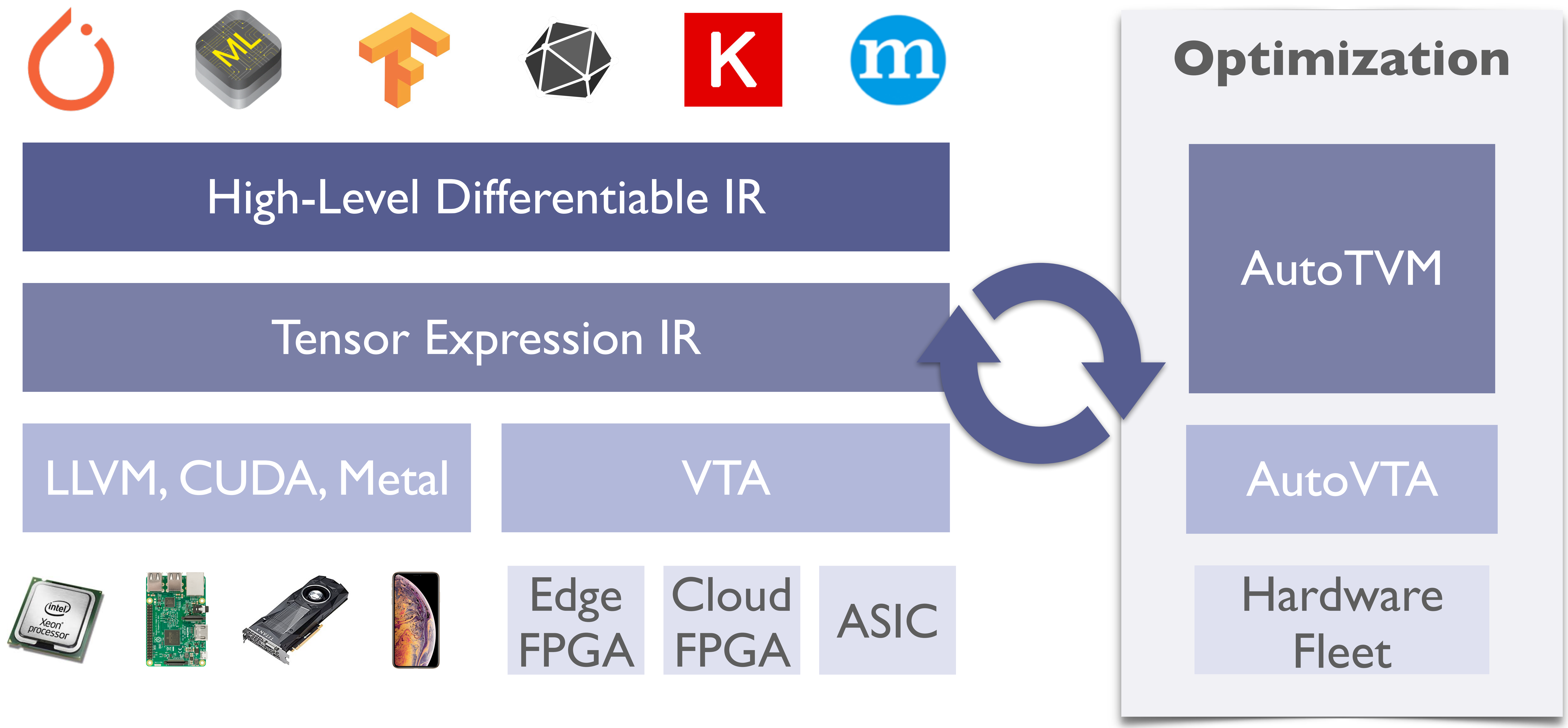
Edge FPGA

Cloud FPGA

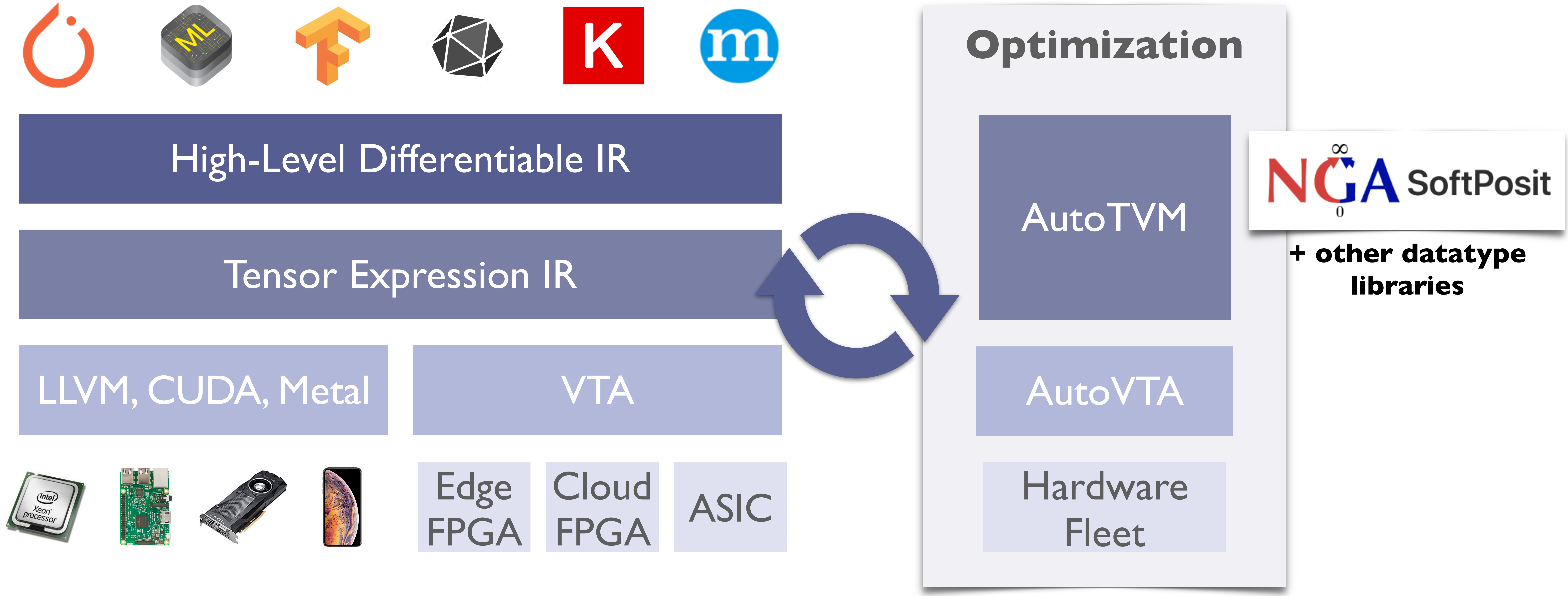
ASIC

# The Goal: BYO-Datatype

# The Goal: BYO-Datatype



# The Goal: BYO-Datatype

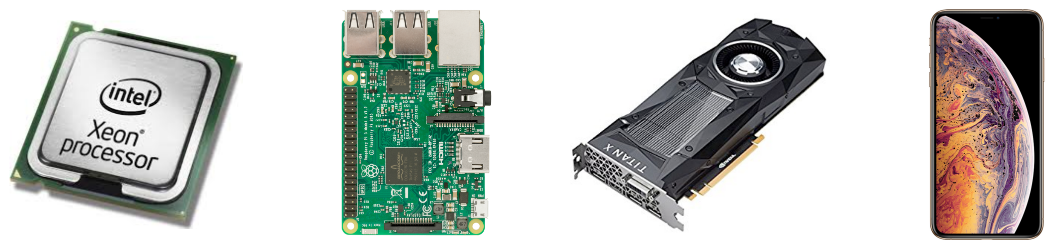


# Future Directions: Hardware

High-Level Differentiable IR

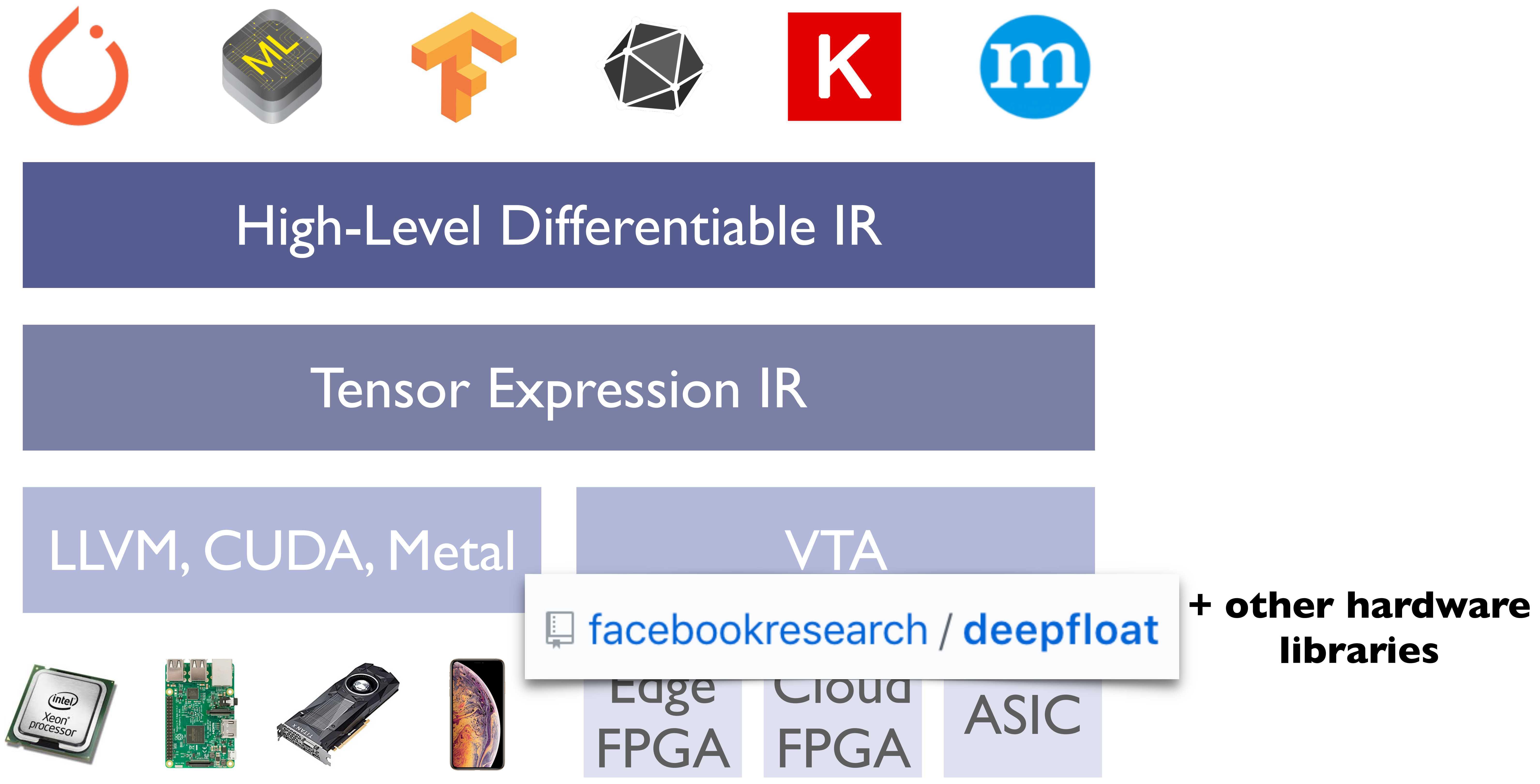
Tensor Expression IR

LLVM, CUDA, Metal | VTA

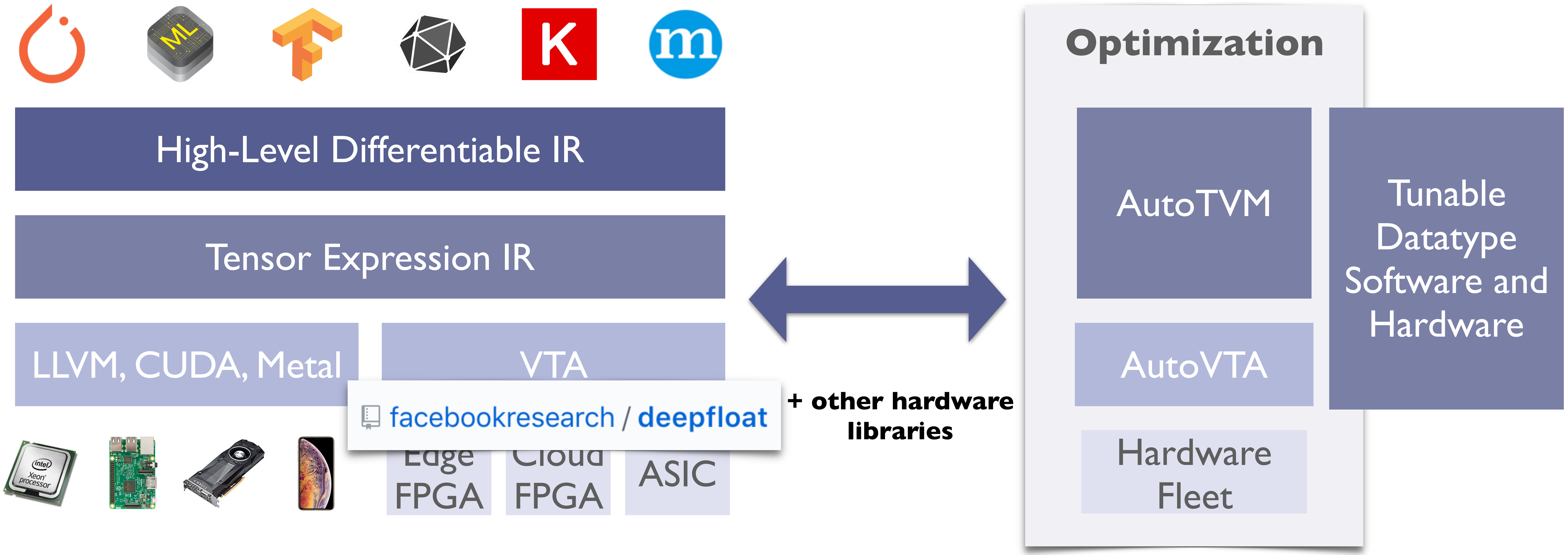



Edge FPGA | Cloud FPGA | ASIC

# Future Directions: Hardware



# Future Directions: Hardware



# What it Looks Like Today

# What it Looks Like Today

**Adding a custom datatype in Python:**

```
tvm.register_datatype("myfloat", 24)

X = tvm.placeholder((3,), name="X")
Y = topi.cast(X, dtype="custom[myfloat]32")
```

# What it Looks Like Today

**Adding a custom datatype in Python:**

```
tvm.register_datatype("myfloat", 24)

X = tvm.placeholder((3,), name="X")
Y = topi.cast(X, dtype="custom[myfloat]32")

cast = tvm.build(flist[0], target=tgt)
```

# What it Looks Like Today

**Adding a custom datatype in Python:**

```
tvm.register_datatype("myfloat", 24)

X = tvm.placeholder((3,), name="X")
Y = topi.cast(X, dtype="custom[myfloat]32")

cast = tvm.build(flist[0], target=tgt)

x = tvm.nd.array(np.random.uniform(size=3).astype(X.dtype), ctx)
y = tvm.nd.empty(Y.shape, dtype=Y.dtype, ctx=ctx)
```

# What it Looks Like Today

**Adding a custom datatype in Python:**

```
tvm.register_datatype("myfloat", 24)

X = tvm.placeholder((3,), name="X")
Y = topi.cast(X, dtype="custom[myfloat]32")

cast = tvm.build(flist[0], target=tgt)

x = tvm.nd.array(np.random.uniform(size=3).astype(X.dtype), ctx)
y = tvm.nd.empty(Y.shape, dtype=Y.dtype, ctx=ctx)

cast(x,y)
```

# What it Looks Like Today

**Adding a custom datatype in Python:**

```
tvm.register_datatype("myfloat", 24)

X = tvm.placeholder((3,), name="X")
Y = topi.cast(X, dtype="custom[myfloat]32")

cast = tvm.build(flist[0], target=tgt)

x = tvm.nd.array(np.random.uniform(size=3).astype(X.dtype), ctx)
y = tvm.nd.empty(Y.shape, dtype=Y.dtype, ctx=ctx)

cast(x,y)
```

**Defining lowering from our custom datatype in C++:**

```
TVM_REGISTER_GLOBAL("tvm.datatypes.lower.llvm.cast.myfloat.float")
.set_body([](runtime::TVMArgs args, runtime::TVMRetValue *rv) {
    Expr e = args[0];
    const ir::Cast* cast = e.as<ir::Cast>();
    internal_assert(cast);
    auto type = cast->type;
    *rv = reinterpret(tvm::UInt(type.bits(), type.lanes()), cast->value);
});
```

# What it Looks Like Today

**Adding a custom datatype in Python:**

```
tvm.register_datatype("myfloat", 24)

X = tvm.placeholder((3,), name="X")
Y = topi.cast(X, dtype="custom[myfloat]32")

cast = tvm.build(flist[0], target=tgt)

x = tvm.nd.array(np.random.uniform(size=3).astype(X.dtype), ctx)
y = tvm.nd.empty(Y.shape, dtype=Y.dtype, ctx=ctx)

cast(x,y)
```

**Defining lowering from our custom datatype in C++:**

```
TVM_REGISTER_GLOBAL("tvm.datatypes.lower.llvm.cast.myfloat.float")
.set_body([](runtime::TVMArgs args, runtime::TVMRetValue *rv) {
    Expr e = args[0];
    const ir::Cast* cast = e.as<ir::Cast>();
    internal_assert(cast);
    auto type = cast->type;
    *rv = reinterpret(tvm::UInt(type.bits(), type.lanes()), cast->value);
});
```

```
x: [0.25599805 0.5752605  0.0941305 ]
y: [1048777261 1058227270 1036044158]
```

# Try it out and get involved!

**My TVM fork:**
**https://github.com/gussmith23/tvm**
**Or reach out to me at**
**gussmith@cs.washington.edu**