Bring Your Own Datatypes into TVM

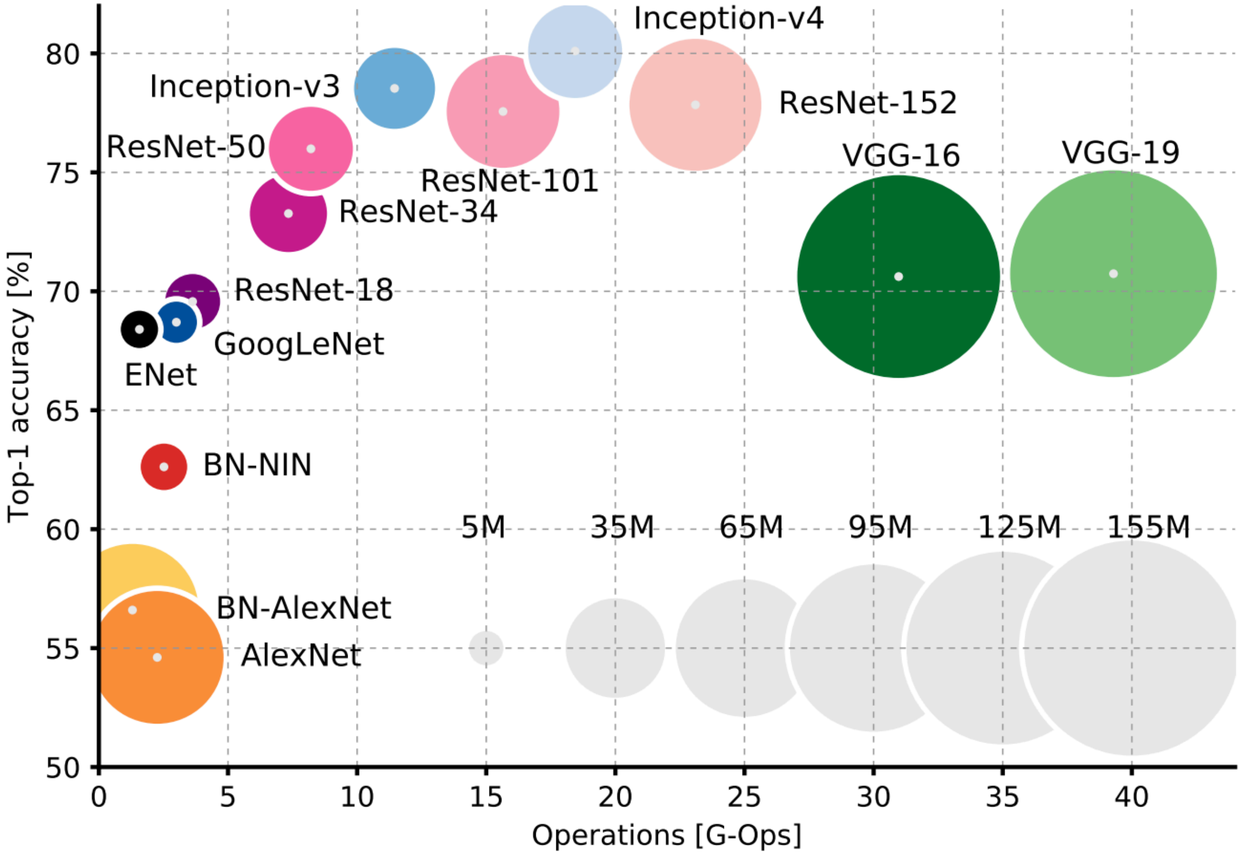
Gus Smith
Advisor: Luis Ceze
University of Washington

CRISP
Center for Research in Intelligent Storage and Processing in Memory

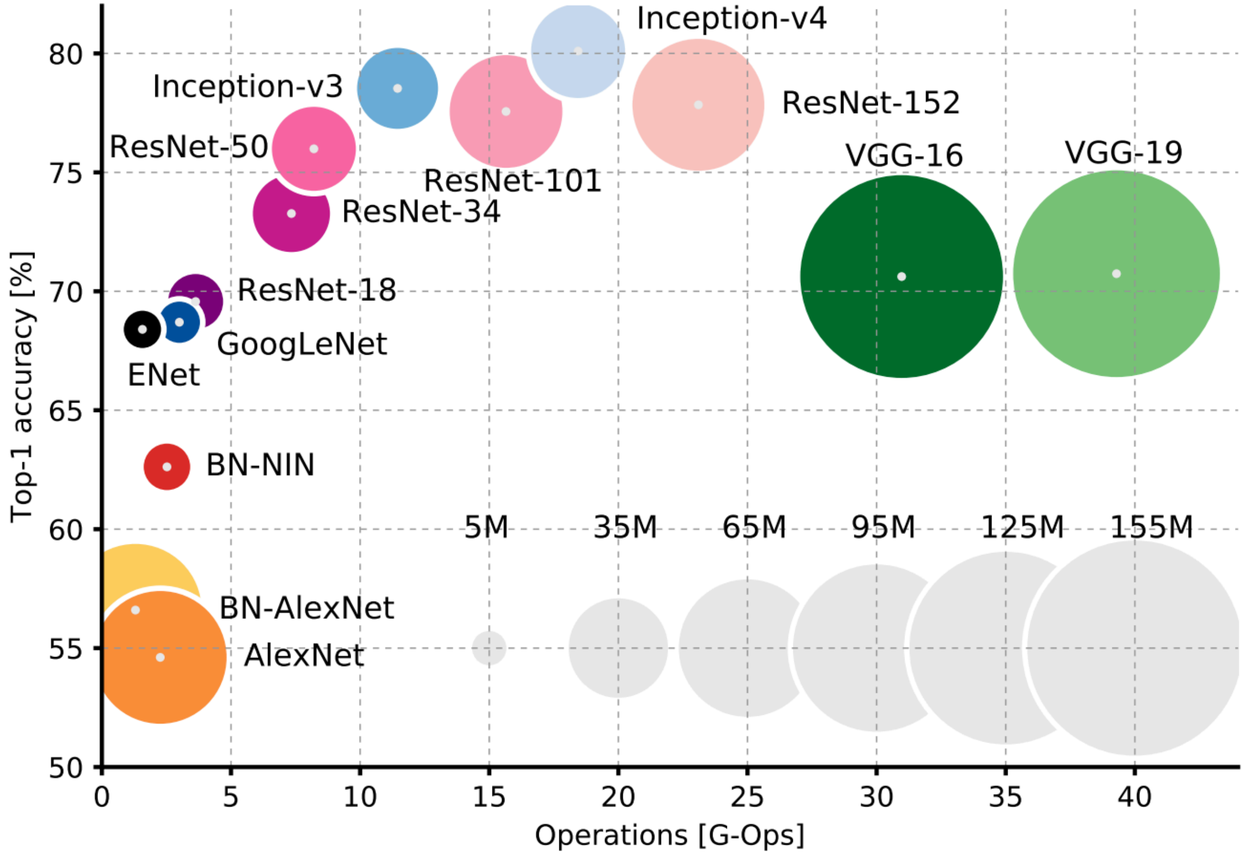
sampl

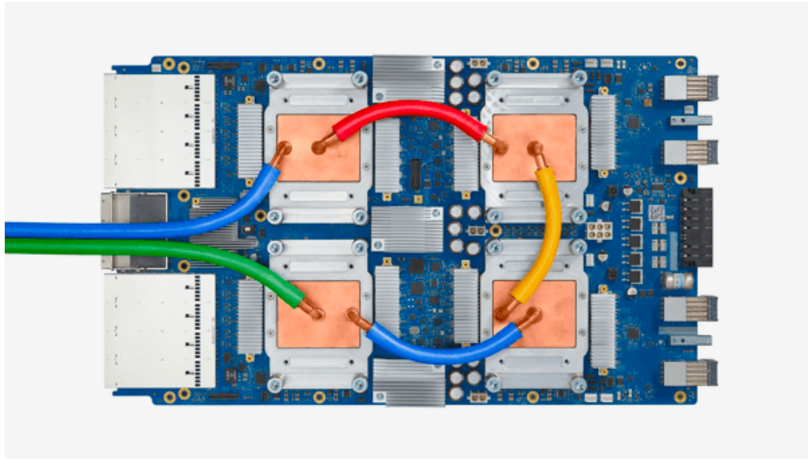
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

tvm.ai

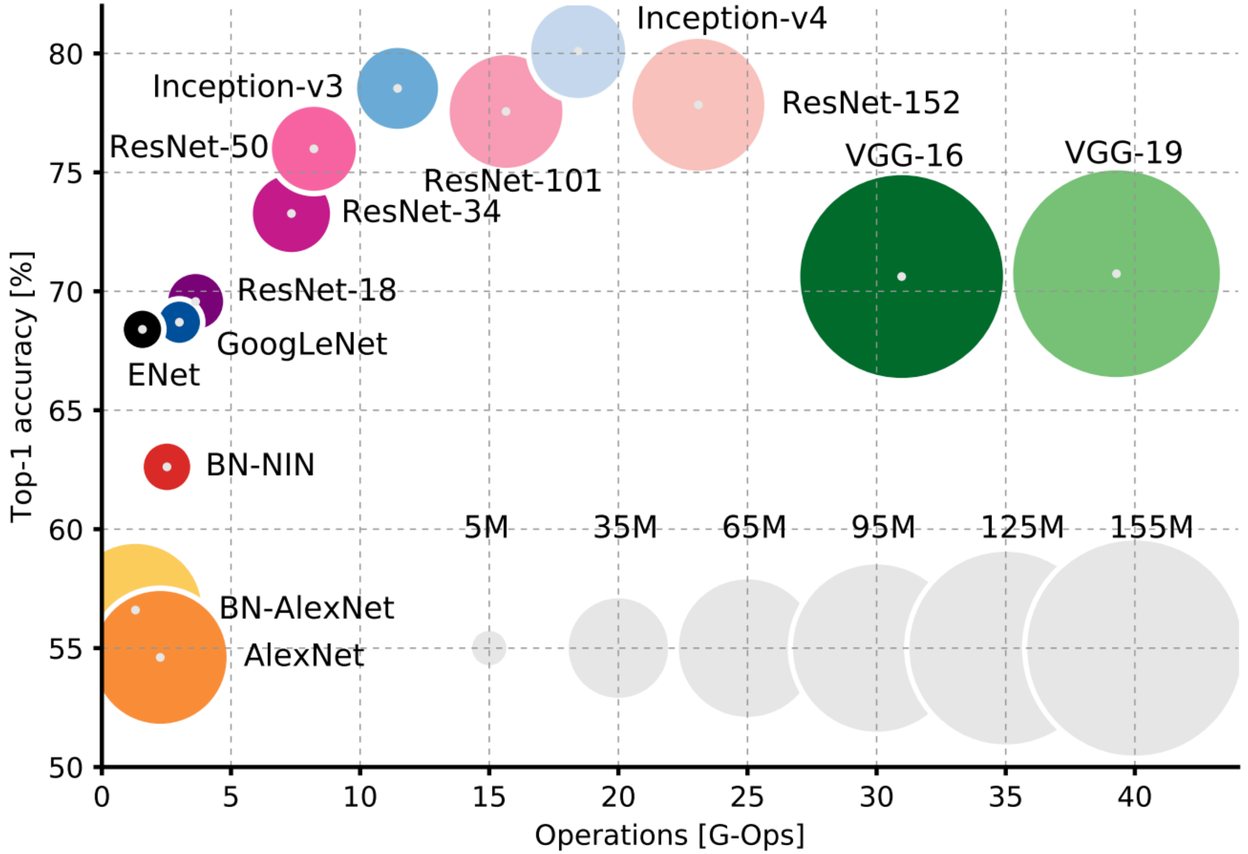


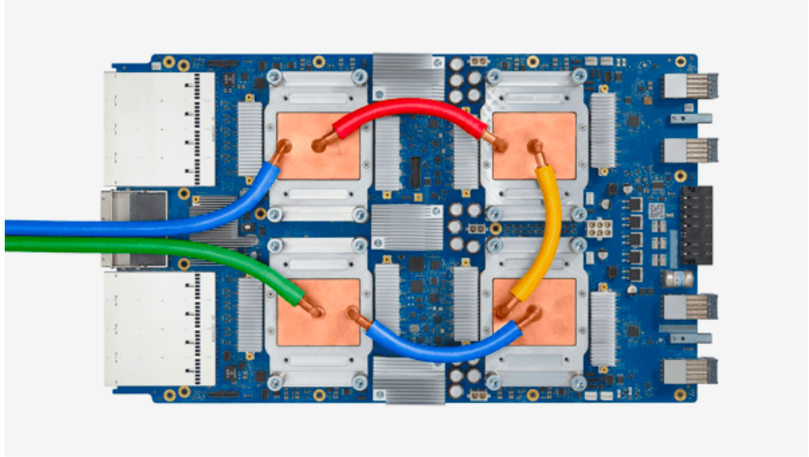
[An Analysis of Deep Neural Network Models for Practical Applications]

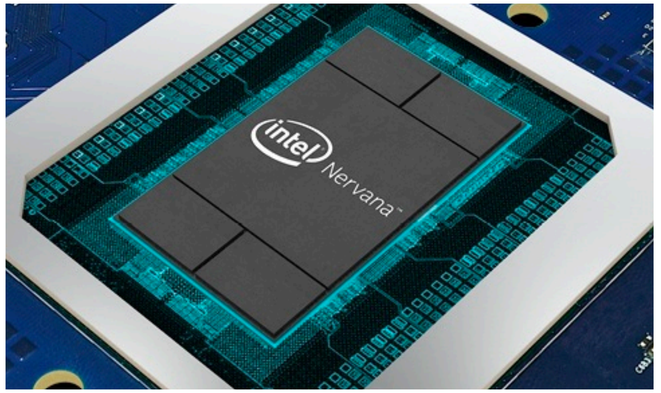




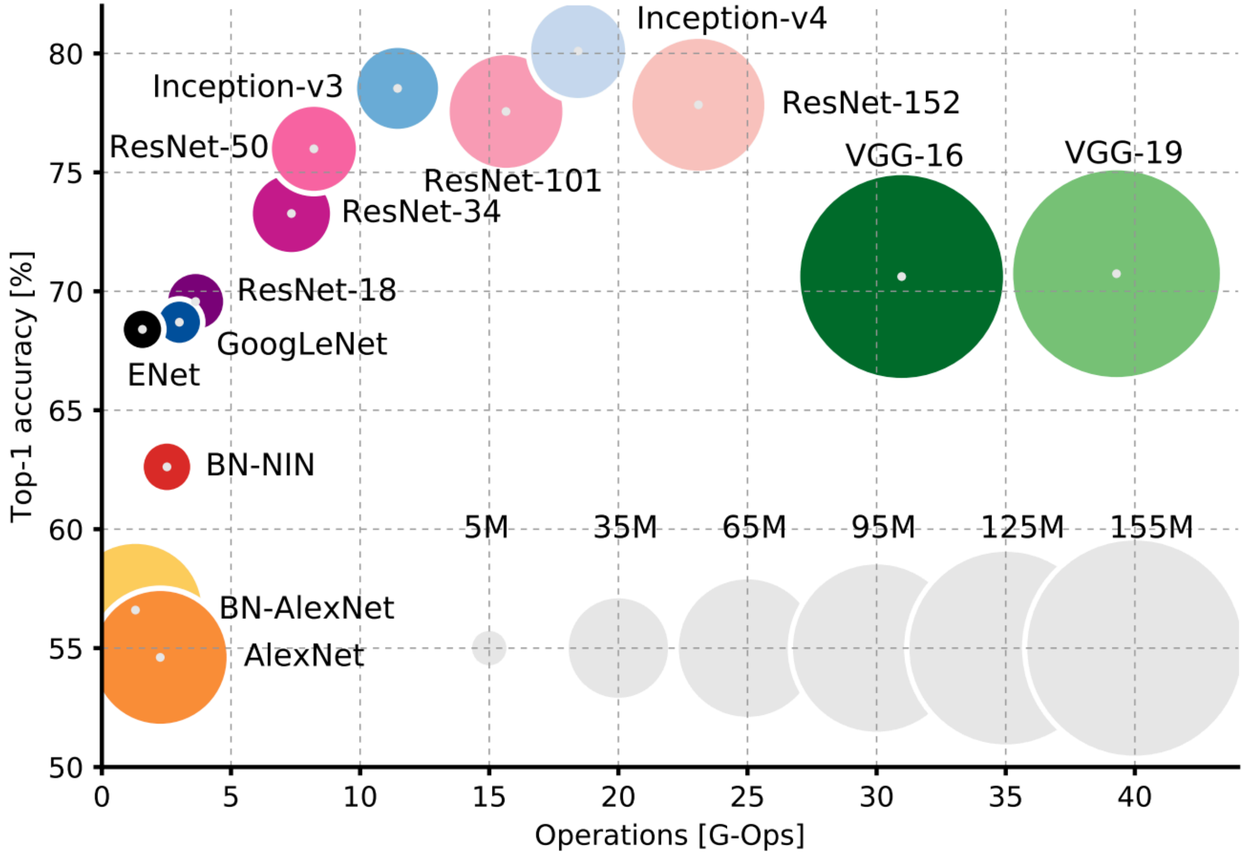
[An Analysis of Deep Neural Network Models for Practical Applications]

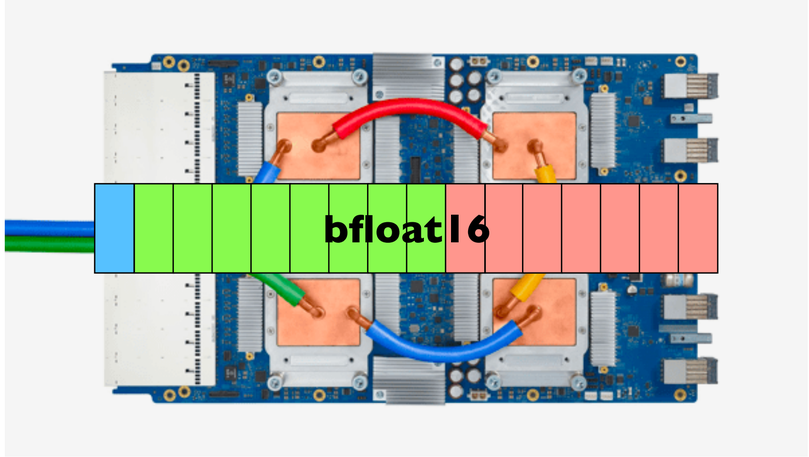


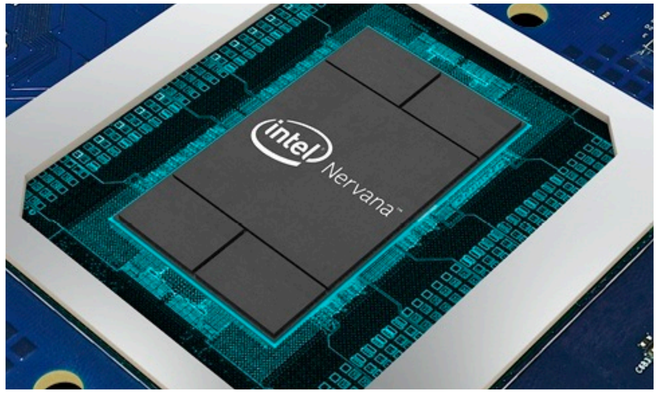




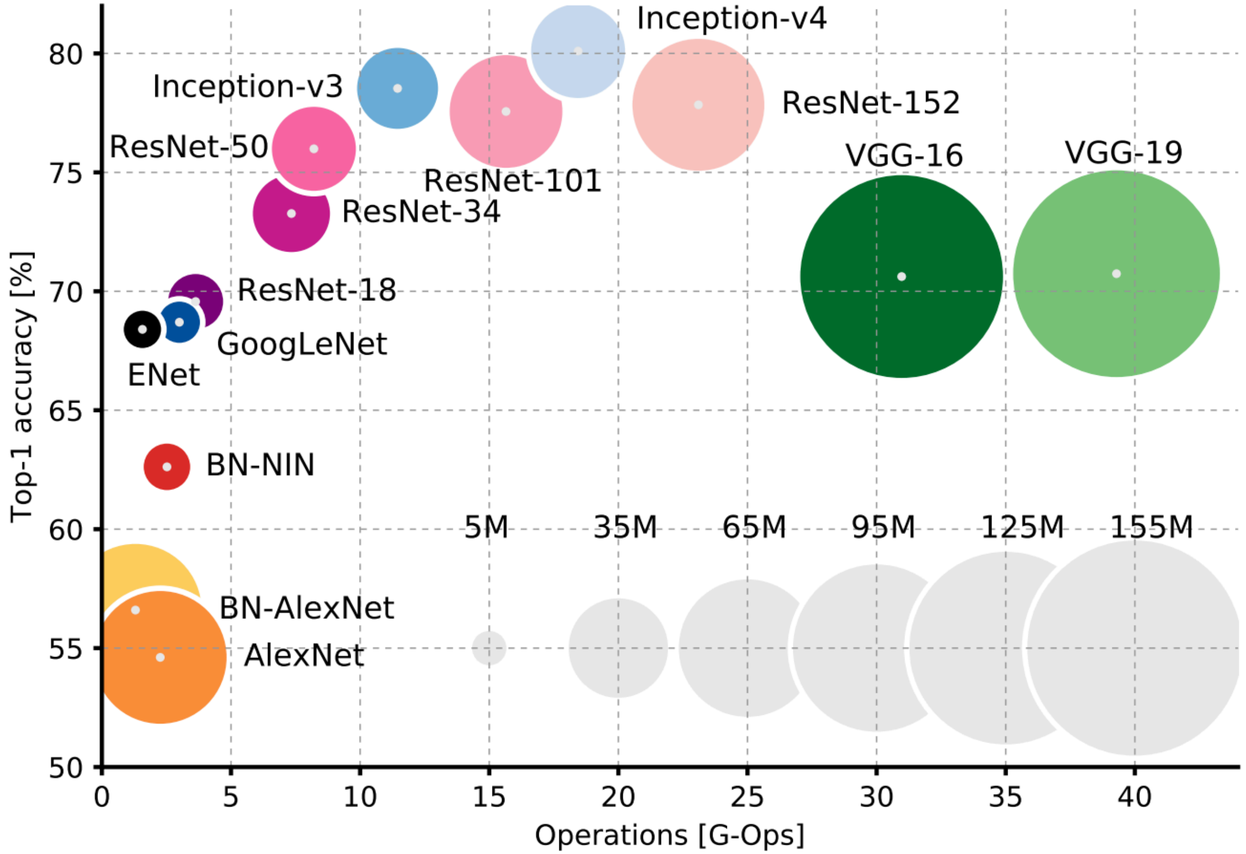
[An Analysis of Deep Neural Network Models for Practical Applications]

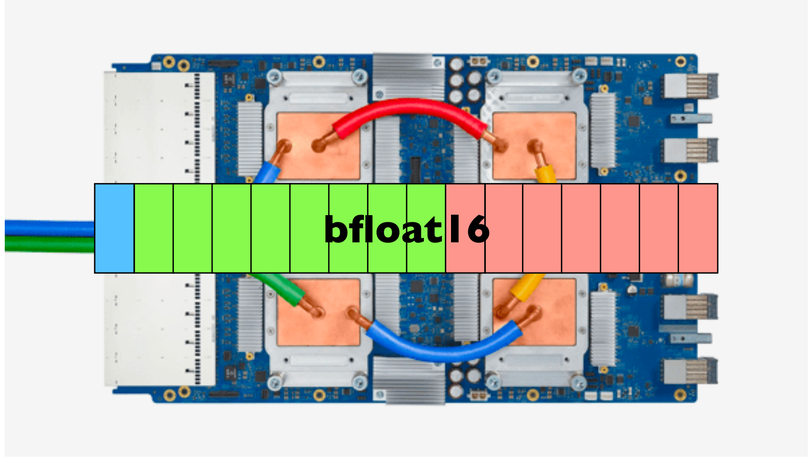


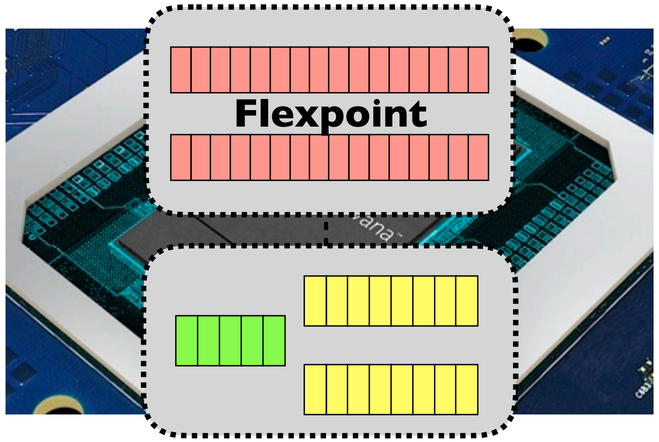




[An Analysis of Deep Neural Network Models for Practical Applications]







[An Analysis of Deep Neural Network Models for Practical Applications]

Deep learning needs extreme hardware specialization, and hardware specialization needs new datatypes.

Background

Background

What is TVM?

Background

What is TVM?

What do we mean by datatypes?

Background

What is TVM?

What do we mean by datatypes?

Why do we need new datatypes?

Background

What is TVM?

What do we mean by datatypes?

Why do we need new datatypes?

What do new datatypes look like?

What is TVM?

What is TVM?

An **extensible, optimizing compiler** for deep learning.

What is TVM?

An **extensible, optimizing compiler** for deep learning.

See tvm.ai for more info!

The TVM Stack

The TVM Stack

The TVM Stack

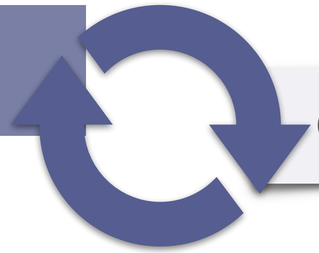
High-Level Differentiable IR

Tensor Expression IR

The TVM Stack

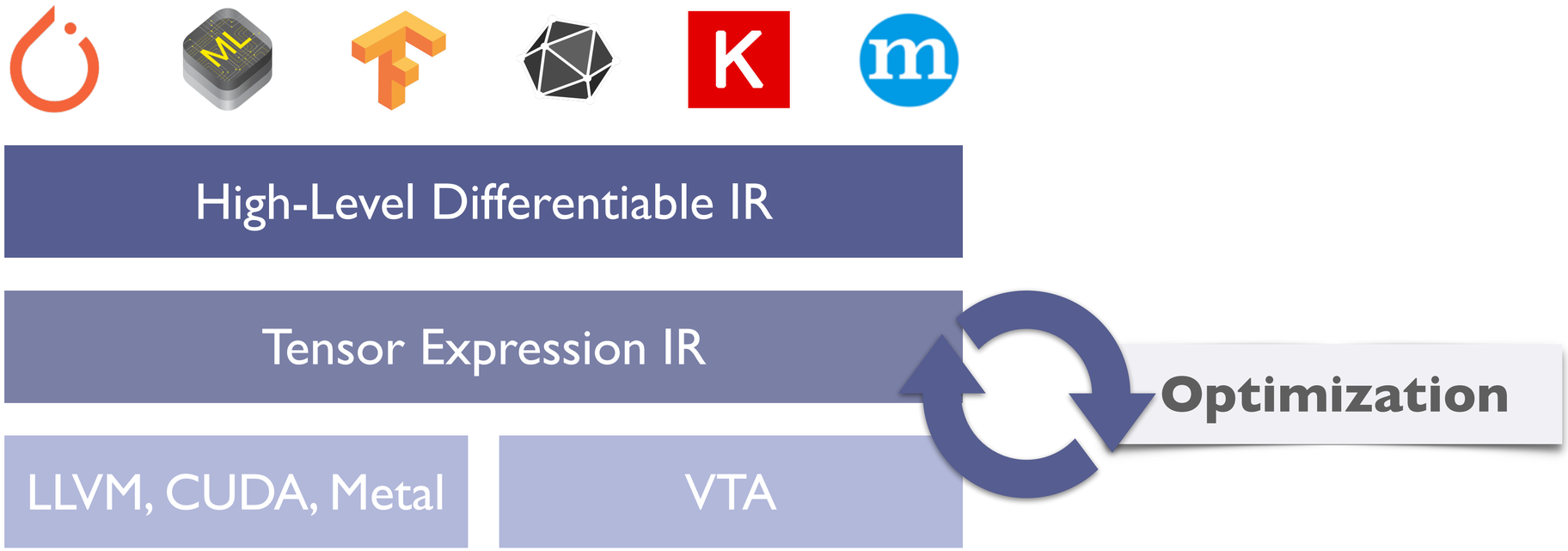
High-Level Differentiable IR

Tensor Expression IR

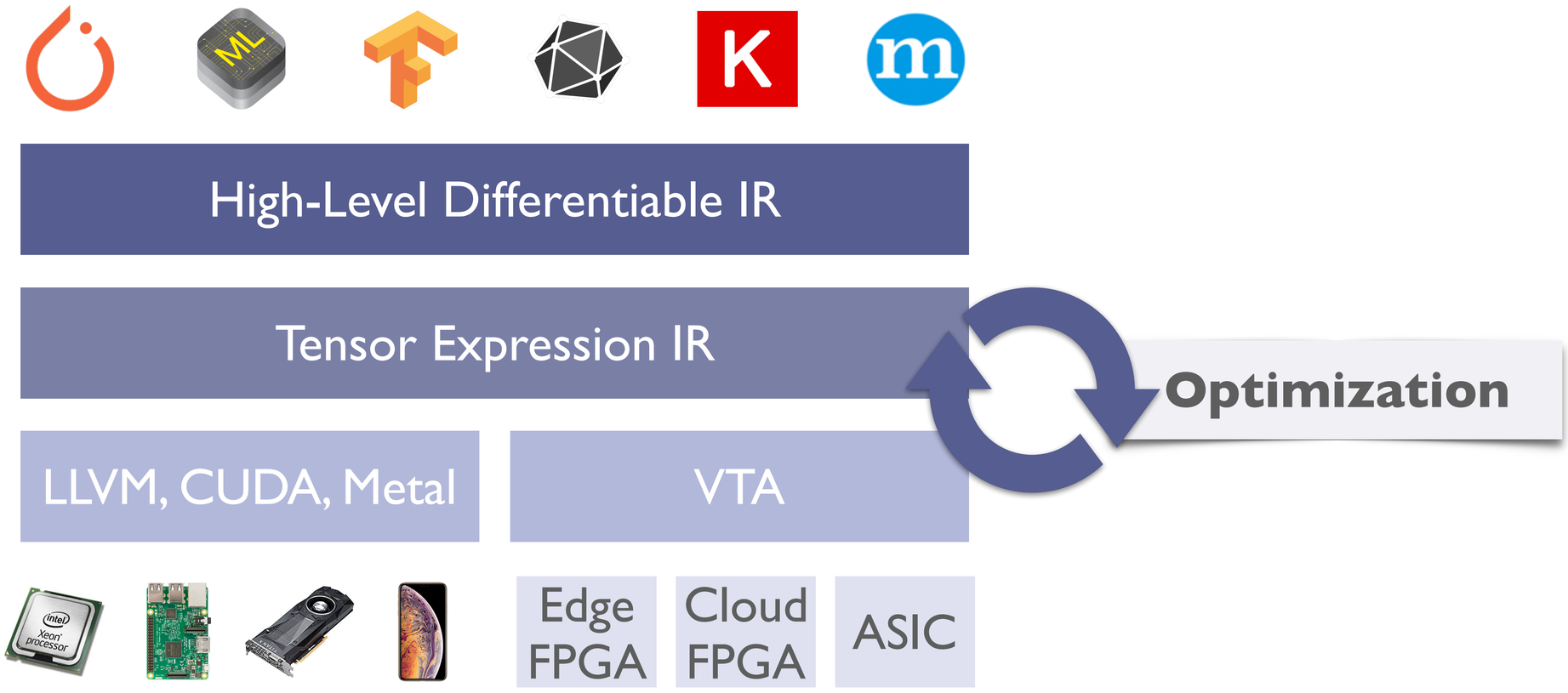


Optimization

The TVM Stack



The TVM Stack



Vector Add in TVM

Vector Add in TVM

```
import tvm
import topi
import numpy as np
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)

x = tvm.nd.array(np.random.rand(3).astype("float32"))
y = tvm.nd.array(np.random.rand(3).astype("float32"))
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)

x = tvm.nd.array(np.random.rand(3).astype("float32"))
y = tvm.nd.array(np.random.rand(3).astype("float32"))
z = tvm.nd.empty((3, ))
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)

x = tvm.nd.array(np.random.rand(3).astype("float32"))
y = tvm.nd.array(np.random.rand(3).astype("float32"))
z = tvm.nd.empty((3, ))
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)

x = tvm.nd.array(np.random.rand(3).astype("float32"))
y = tvm.nd.array(np.random.rand(3).astype("float32"))
z = tvm.nd.empty((3, ))

built_program(x, y, z)
```

Vector Add in TVM

```
import tvm
import topi
import numpy as np

X = tvm.placeholder(shape=(3, ))
Y = tvm.placeholder(shape=(3, ))
Z = topi.add(X, Y)

schedule = tvm.create_schedule([Z.op])
lowered_function = tvm.lower(schedule, [X, Y, Z])
built_program = tvm.build(lowered_function)

x = tvm.nd.array(np.random.rand(3).astype("float32"))
y = tvm.nd.array(np.random.rand(3).astype("float32"))
z = tvm.nd.empty((3, ))

built_program(x, y, z)
```

```
x: [0.5727742  0.16838571 0.9647888 ]
y: [0.75005066 0.12305858 0.9467064 ]
z: [1.3228248 0.2914443 1.9114952]
```

What do we mean by “datatypes”?

What do we mean by “datatypes”?

Numerical datatypes: how the hardware represents real numbers

What do we mean by “datatypes”?

Numerical datatypes: how the hardware represents real numbers

In the past, this generally meant IEEE 754 floating point:

What do we mean by “datatypes”?

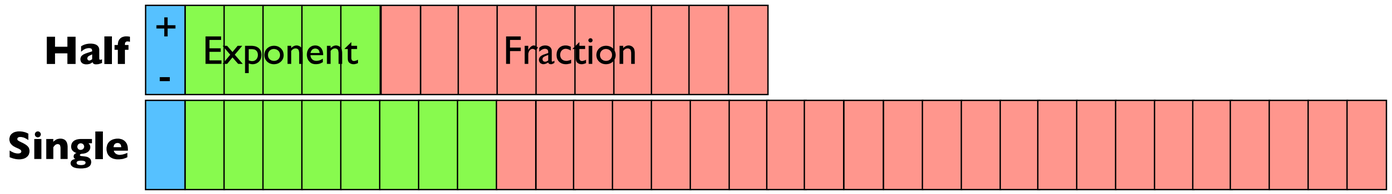
Numerical datatypes: how the hardware represents real numbers

In the past, this generally meant IEEE 754 floating point:

What do we mean by “datatypes”?

Numerical datatypes: how the hardware represents real numbers

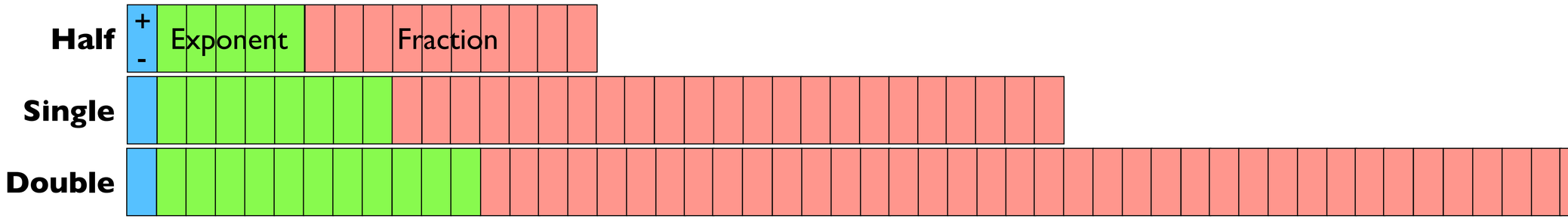
In the past, this generally meant IEEE 754 floating point:



What do we mean by “datatypes”?

Numerical datatypes: how the hardware represents real numbers

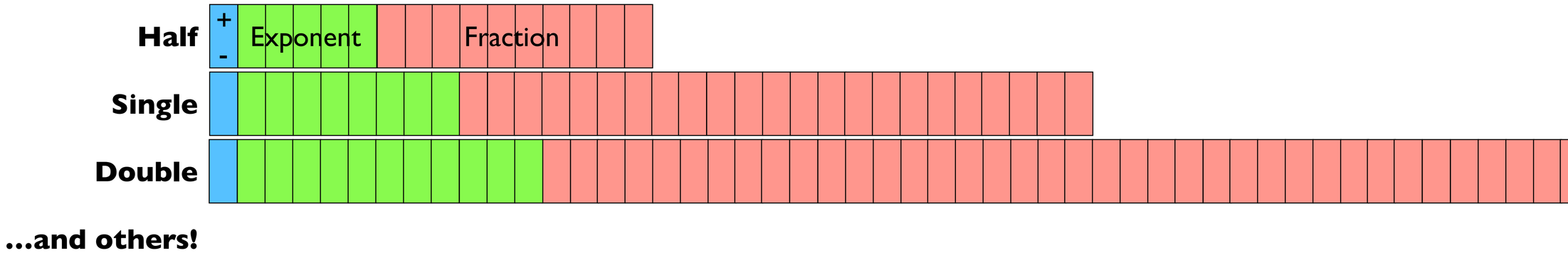
In the past, this generally meant IEEE 754 floating point:



What do we mean by “datatypes”?

Numerical datatypes: how the hardware represents real numbers

In the past, this generally meant IEEE 754 floating point:



Why do we need new datatypes?

Why do we need new datatypes?

Because IEEE floats…

Why do we need new datatypes?

Because IEEE floats…

- take up too much space

Why do we need new datatypes?

Because IEEE floats…

- take up too much space
- have poor dynamic range

Why do we need new datatypes?

Because IEEE floats…

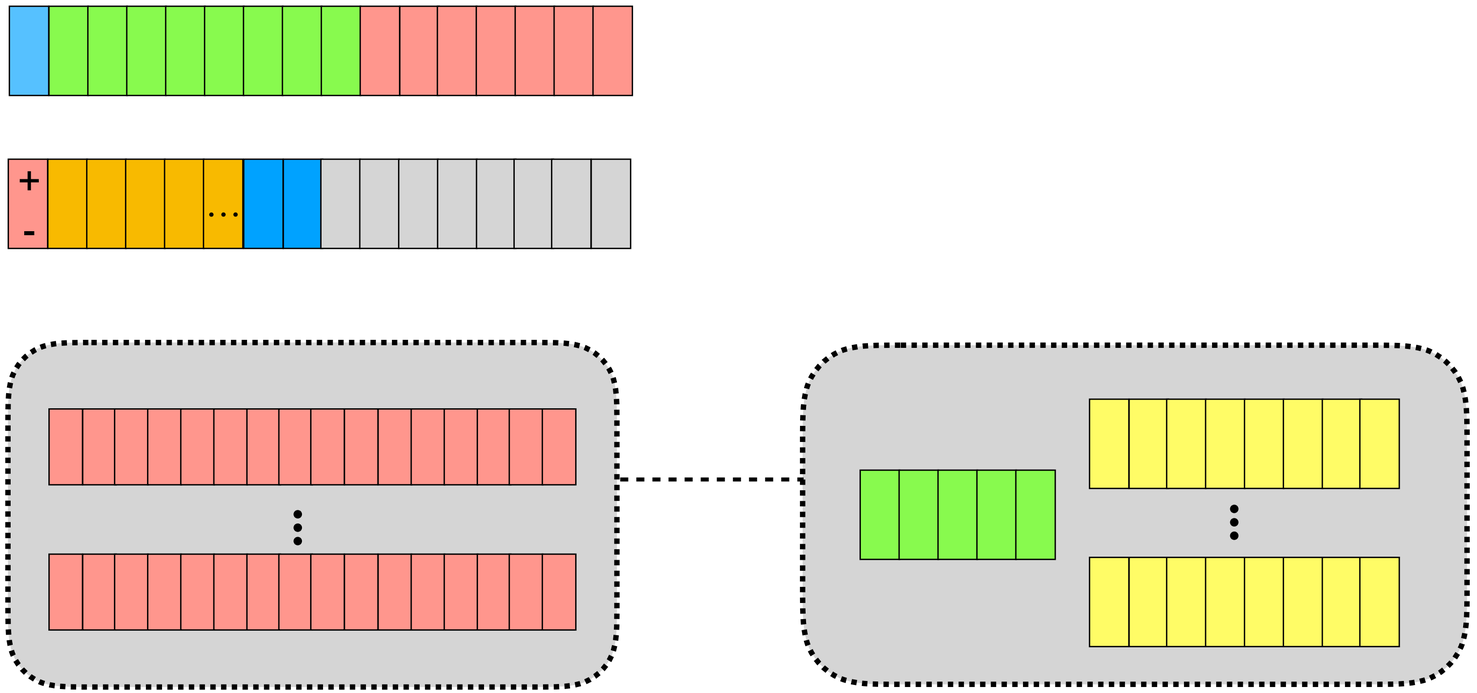
- take up too much space
- have poor dynamic range
- require overly-complex hardware

Why do we need new datatypes?

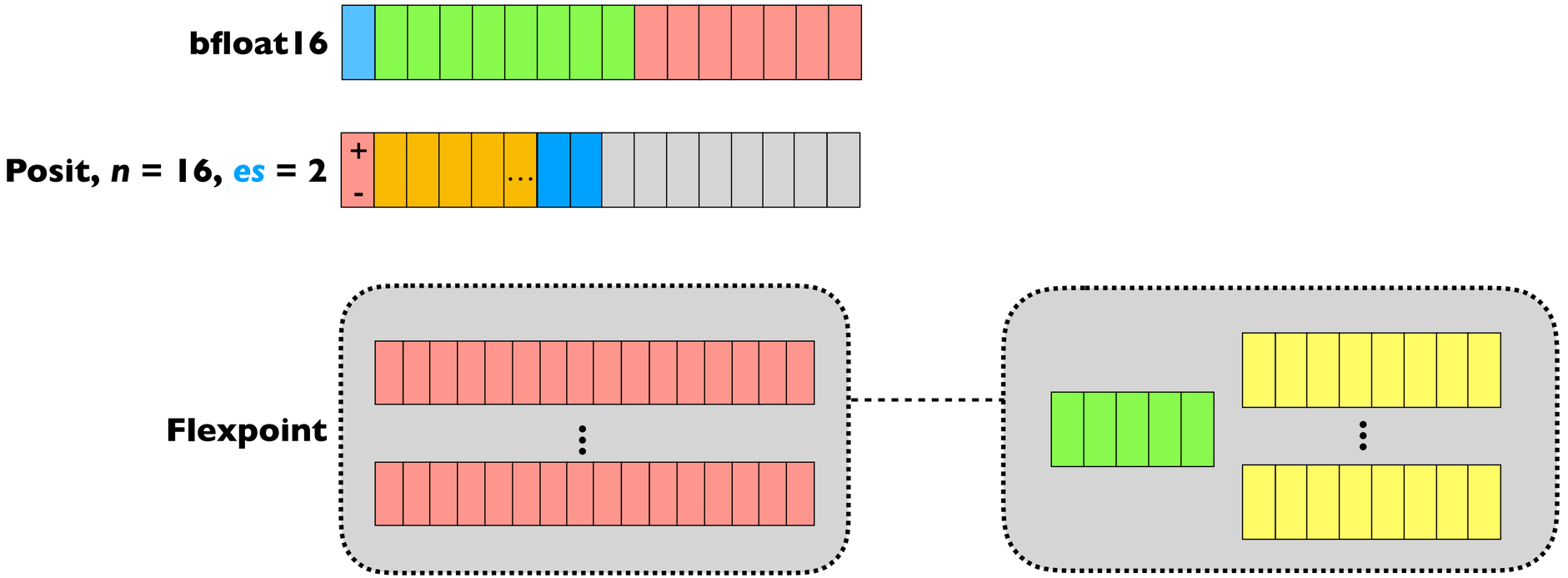
Because IEEE floats…

- take up too much space
- have poor dynamic range
- require overly-complex hardware
- create reproducibility issues

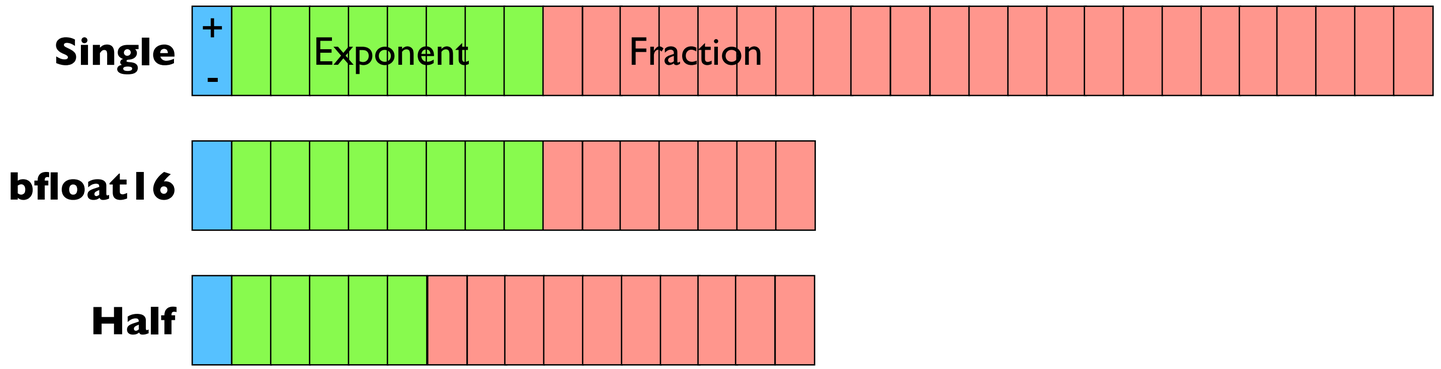
What do new datatypes look like?



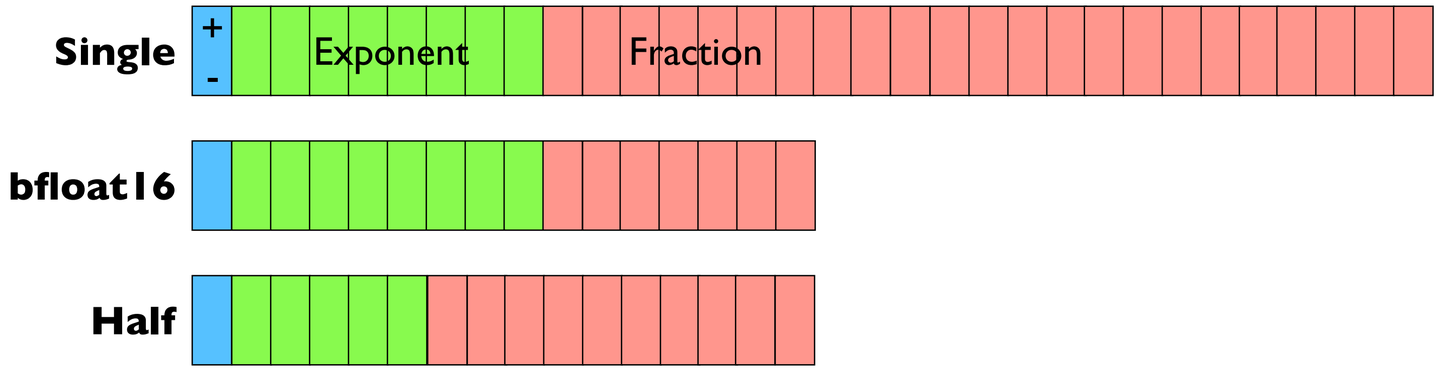
What do new datatypes look like?



bfloat16

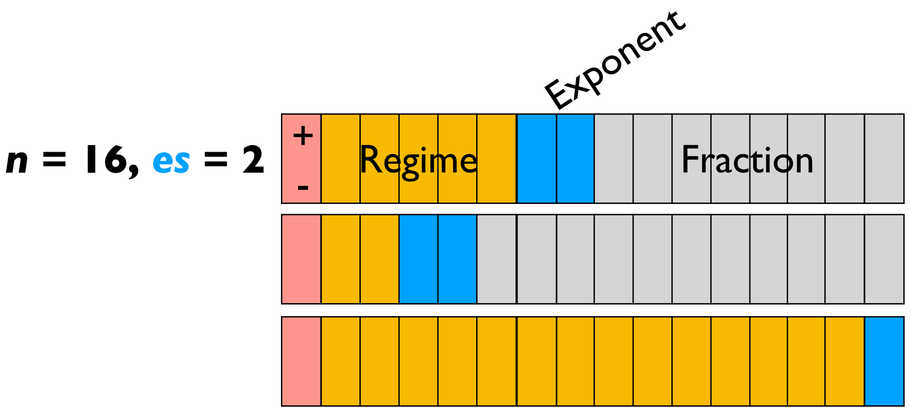


bfloat16

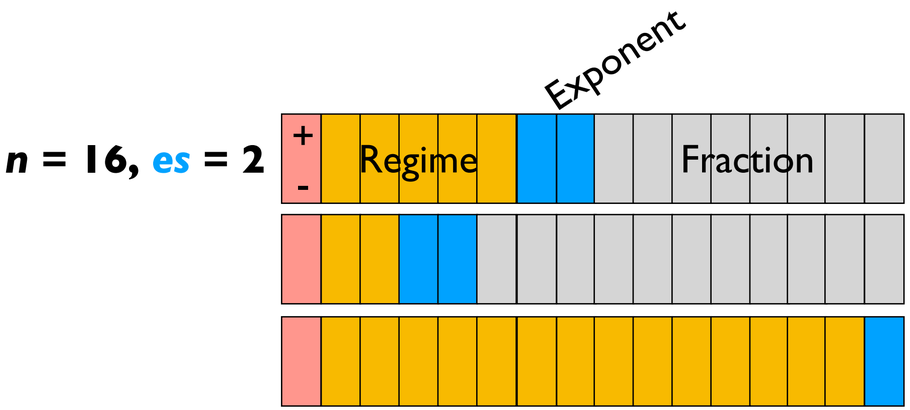


Greater dynamic range is more useful for deep learning workloads

Posits

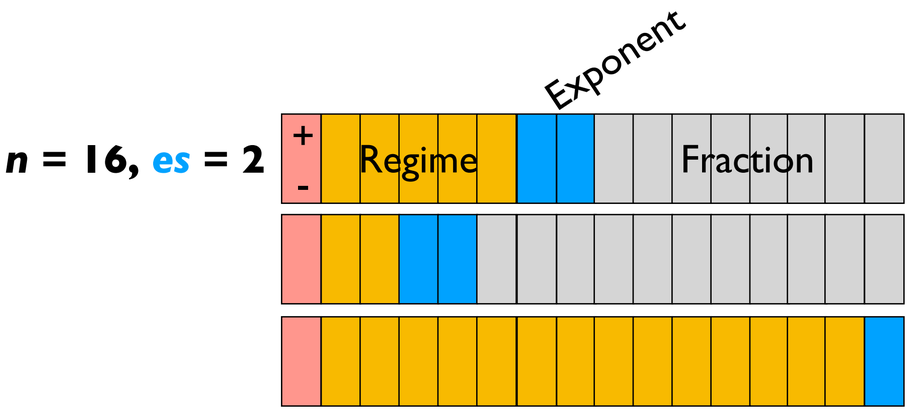


Posits



Posits use a variable-size **regime** field to add dynamic scaling: **regime** determines another multiplicative factor on the fraction

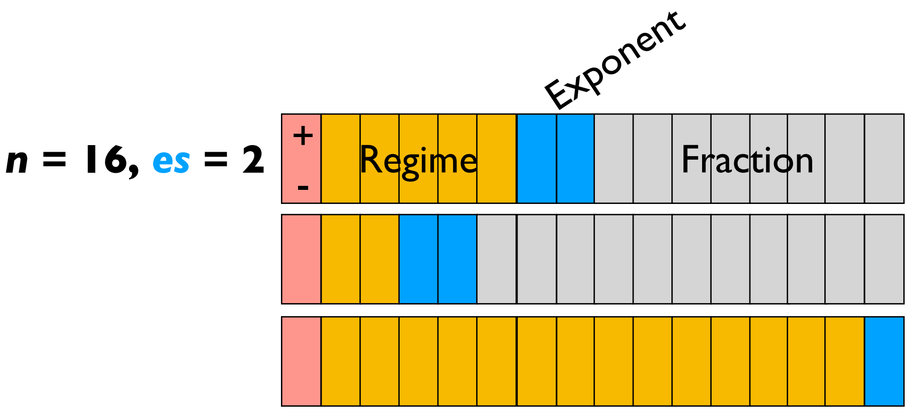
Posits



Posits use a variable-size **regime** field to add dynamic scaling: **regime** determines another multiplicative factor on the fraction

Posits can have large dynamic range even down to 8 bits

Posits

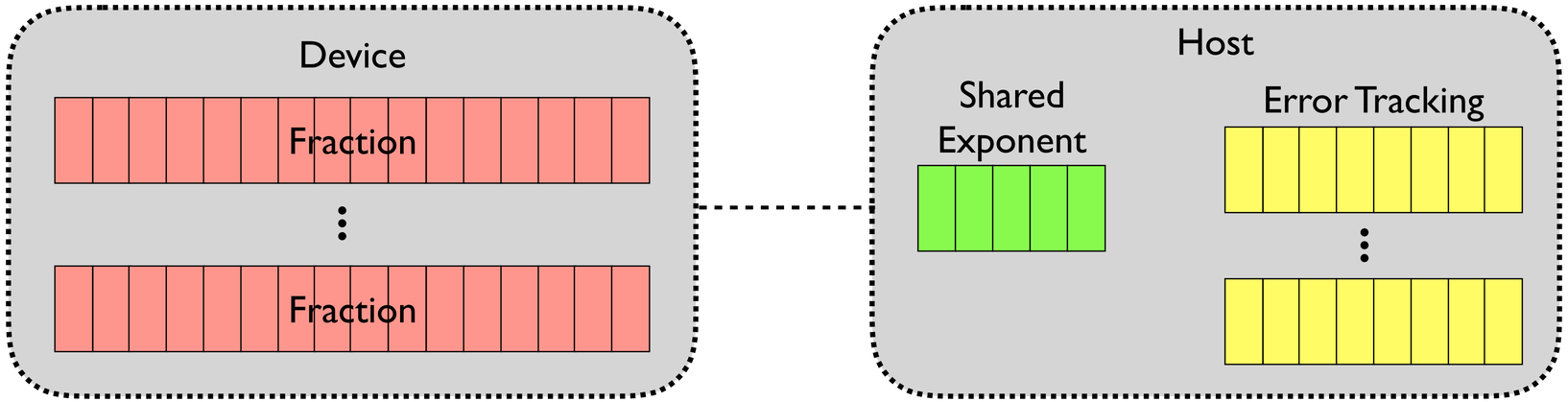


Posits use a variable-size **regime** field to add dynamic scaling: **regime** determines another multiplicative factor on the fraction

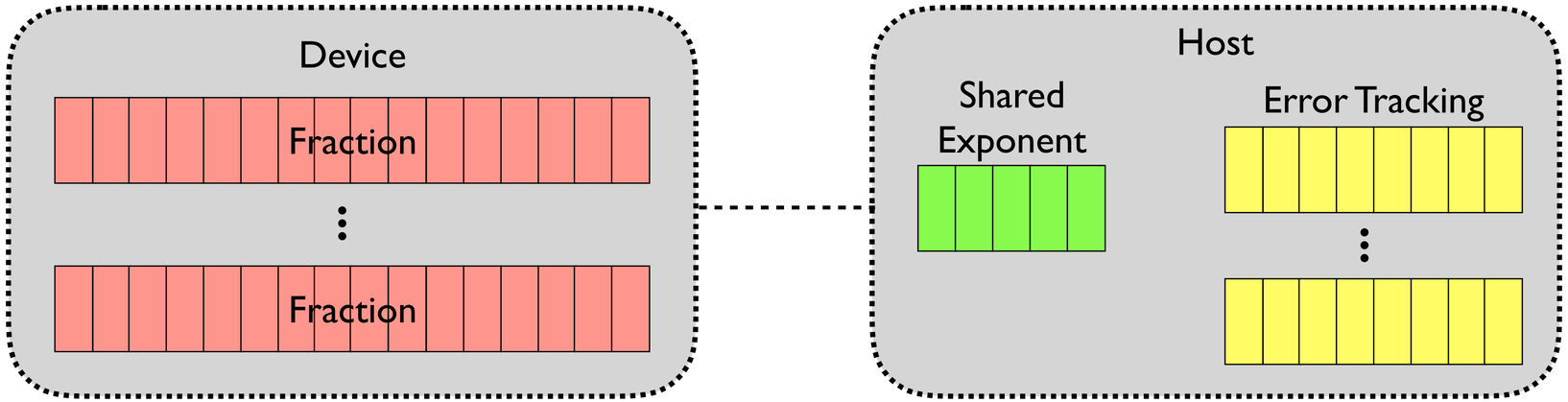
Posits can have large dynamic range even down to 8 bits

See John Gustafson's "Beating Floating Point at its Own Game"

Flexpoint

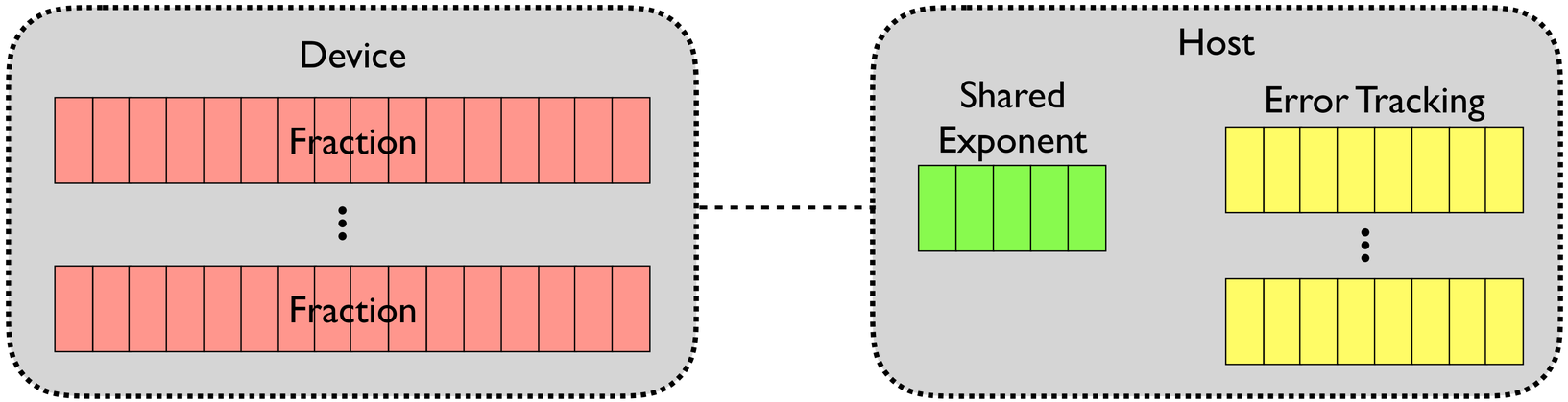

Device
Fraction
Fraction
Host
Shared
Exponent
Error Tracking

Flexpoint



Intel's format for their Nervana processor

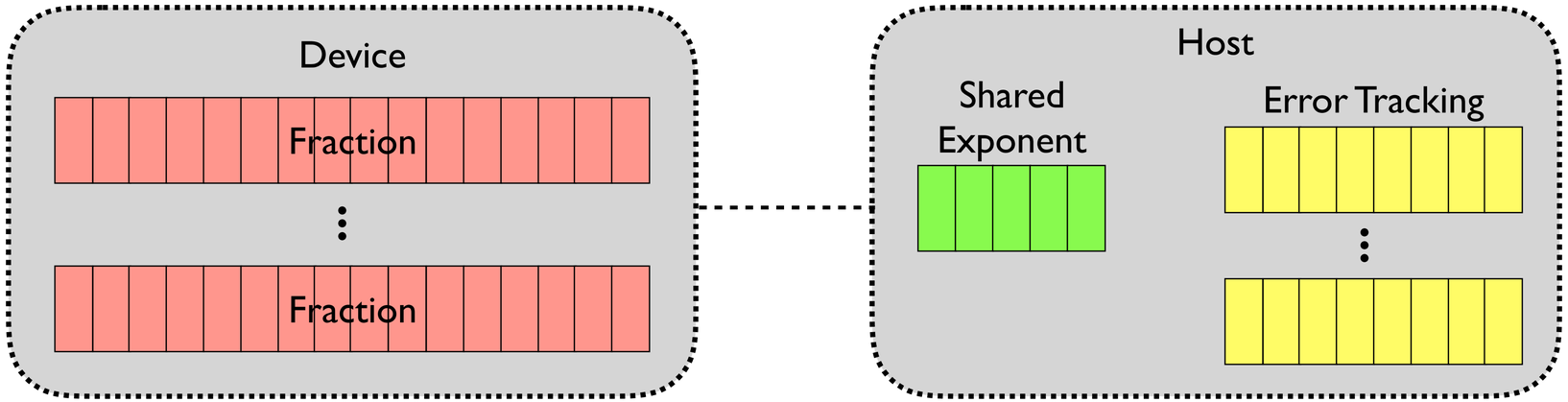
Flexpoint



Intel's format for their Nervana processor

Block floating point: a group of numbers share an exponent

Flexpoint

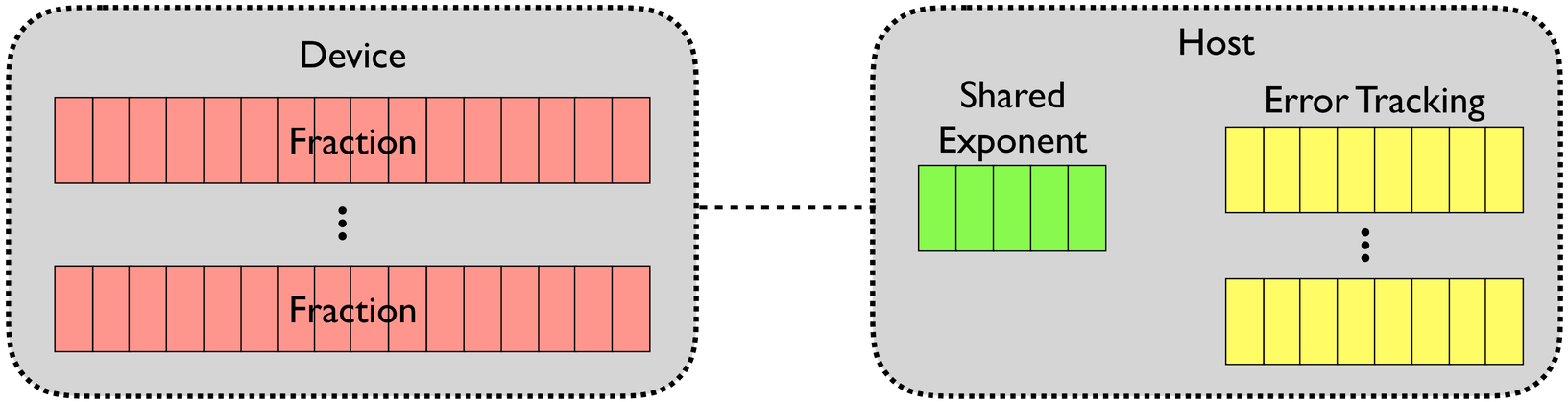


Intel's format for their Nervana processor

Block floating point: a group of numbers share an exponent

Plus, they split the fields between host and device!

Flexpoint



Intel's format for their Nervana processor

Block floating point: a group of numbers share an exponent

Plus, they split the fields between host and device!

See Köster et. al., "An Adaptive Numerical Format for Efficient Training of Deep Neural Networks"

Deepfloat

Deepfloat

Taking inspiration from posits, but with many improvements

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Hardware implementation from Jeff Johnson at Facebook is competitive with IEEE floats:

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Hardware implementation from Jeff Johnson at Facebook is competitive with IEEE floats:

Table 3: Chip area and power for 28 nm, 1-cycle multiply-add at 500 MHz

Component	**Area** μm^2	**Power** μW
int8/32 MAC PE	**336.672**	**283**

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Hardware implementation from Jeff Johnson at Facebook is competitive with IEEE floats:

Table 3: Chip area and power for 28 nm, 1-cycle multiply-add at 500 MHz

Component	**Area** μm^2	**Power** μW
int8/32 MAC PE	**336.672**	**283**
(8, 1, 5, 5, 7) log ELMA PE	**376.110**	**272**

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Hardware implementation from Jeff Johnson at Facebook is competitive with IEEE floats:

Table 3: Chip area and power for 28 nm, 1-cycle multiply-add at 500 MHz

Component	**Area** μm^2	**Power** μW
int8/32 MAC PE	**336.672**	**283**
(8, 1, 5, 5, 7) log ELMA PE	**376.110**	**272**
float16 (w/o denormals) FMA PE	**1545.012**	**1358**

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Hardware implementation from Jeff Johnson at Facebook is competitive with IEEE floats:

Table 3: Chip area and power for 28 nm, 1-cycle multiply-add at 500 MHz

Component	**Area μm^2**	**Power μW**
int8/32 MAC PE	**336.672**	**283**
(8, 1, 5, 5, 7) log ELMA PE	**376.110**	**272**
float16 (w/o denormals) FMA PE	**1545.012**	**1358**
(5, 10) (11, 11, 10) log ELMA PE	**1043.154**	**805**

Deepfloat

Taking inspiration from posits, but with many improvements

E.g. some computation is done in the log space to reduce hardware complexity

Hardware implementation from Jeff Johnson at Facebook is competitive with IEEE floats:

Table 3: Chip area and power for 28 nm, 1-cycle multiply-add at 500 MHz

Component	Area μm^2	Power μW
int8/32 MAC PE	**336.672**	**283**
(8, 1, 5, 5, 7) log ELMA PE	**376.110**	**272**
float16 (w/o denormals) FMA PE	**1545.012**	**1358**
(5, 10) (11, 11, 10) log ELMA PE	**1043.154**	**805**

See "Rethinking Floating Point for Deep Learning"

Bring Your Own Datatypes

How Datatype Research Works

How Datatype Research Works

How Datatype Research Works

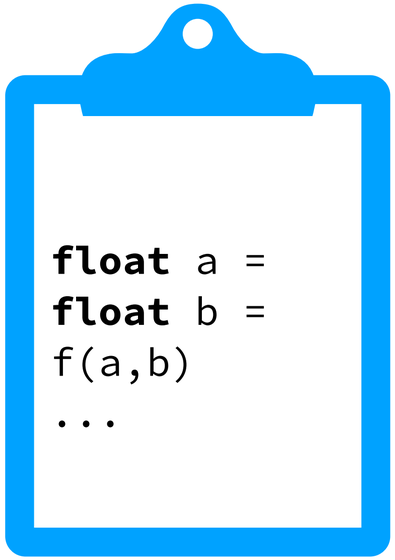
To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

How Datatype Research Works

To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

How Datatype Research Works

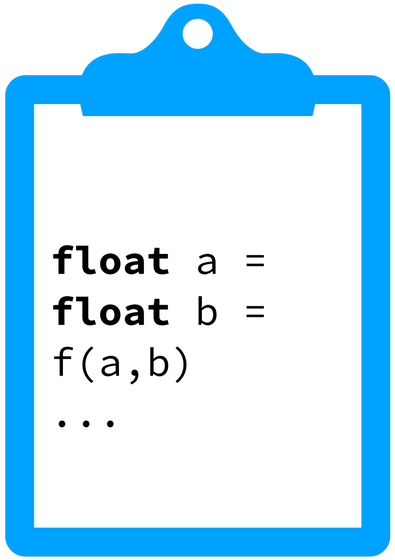
To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library



How Datatype Research Works

To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

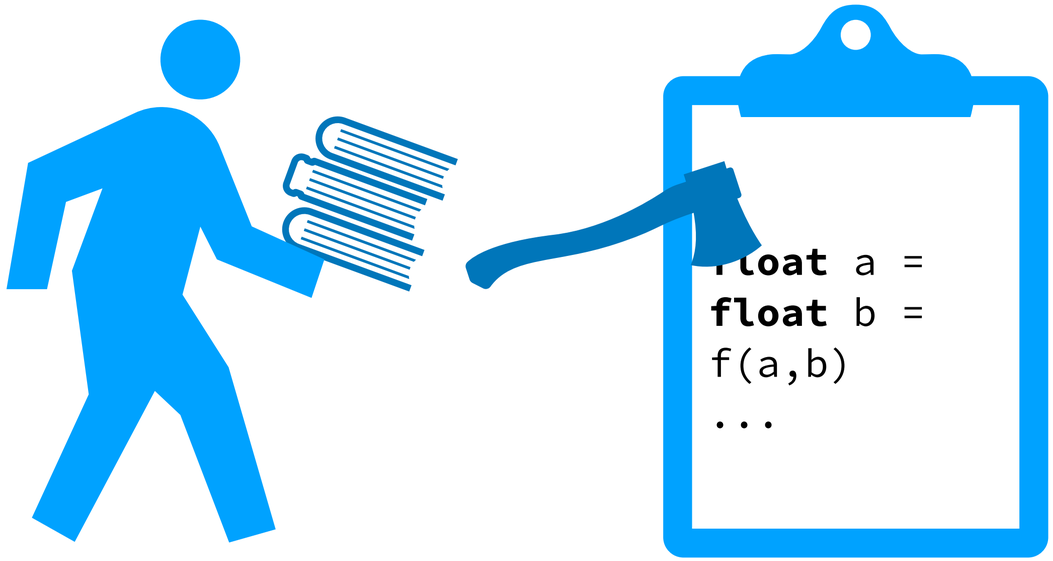
To test their datatype, they will hack apart a benchmark or application and shove their software library in



How Datatype Research Works

To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

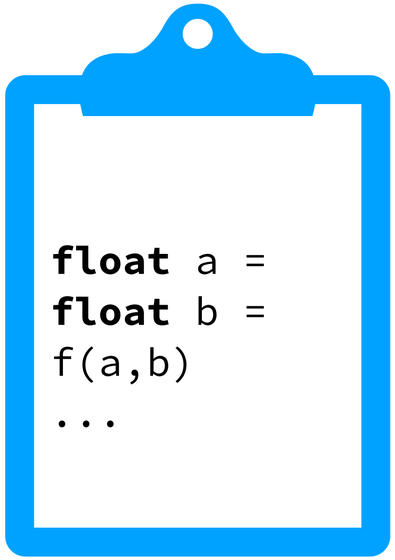
To test their datatype, they will hack apart a benchmark or application and shove their software library in



How Datatype Research Works

To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

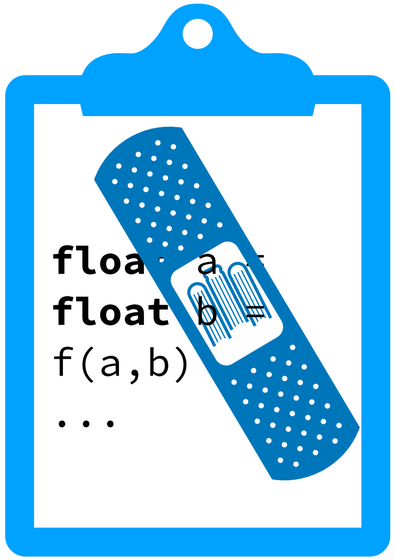
To test their datatype, they will hack apart a benchmark or application and shove their software library in



How Datatype Research Works

To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

To test their datatype, they will hack apart a benchmark or application and shove their software library in

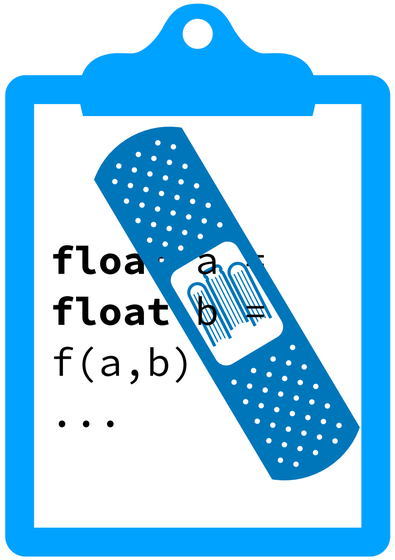


How Datatype Research Works

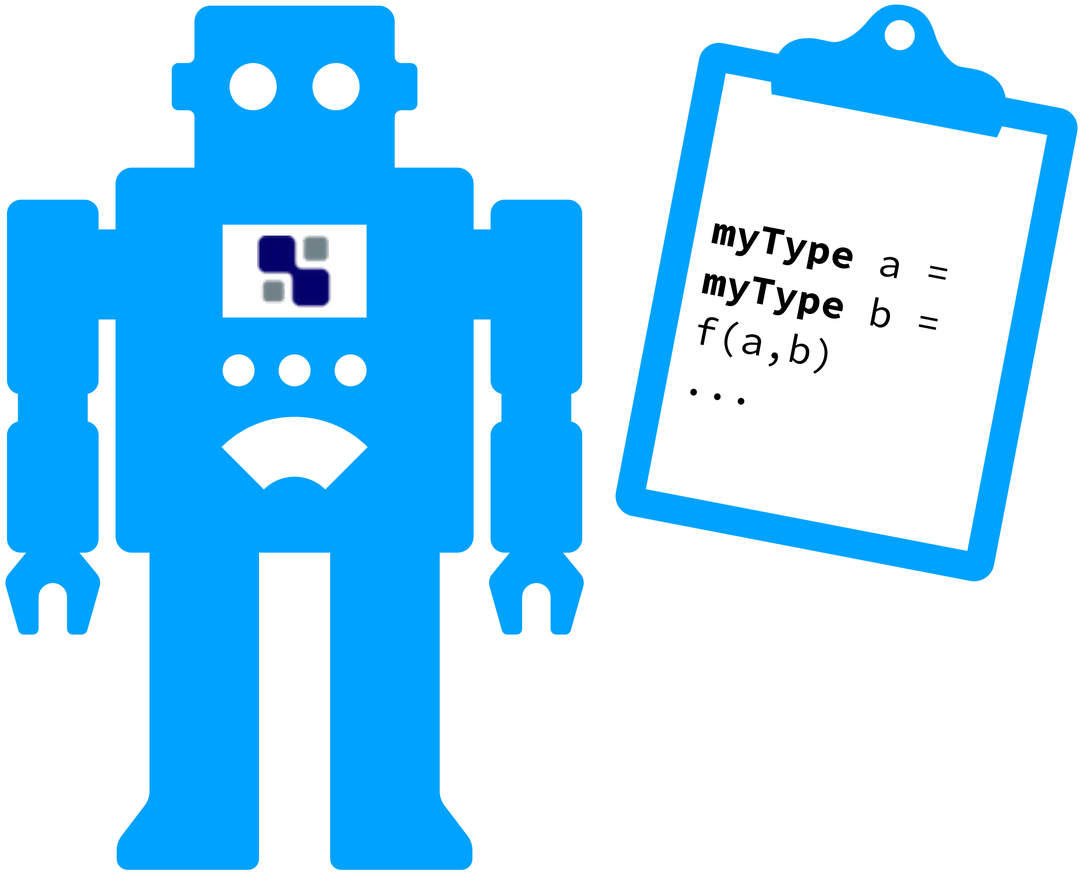
To prototype a hardware datatype, researchers will emulate the datatype in software by building a software library

To test their datatype, they will hack apart a benchmark or application and shove their software library in

Can we do better? Can we use TVM to compile workloads with new datatypes?

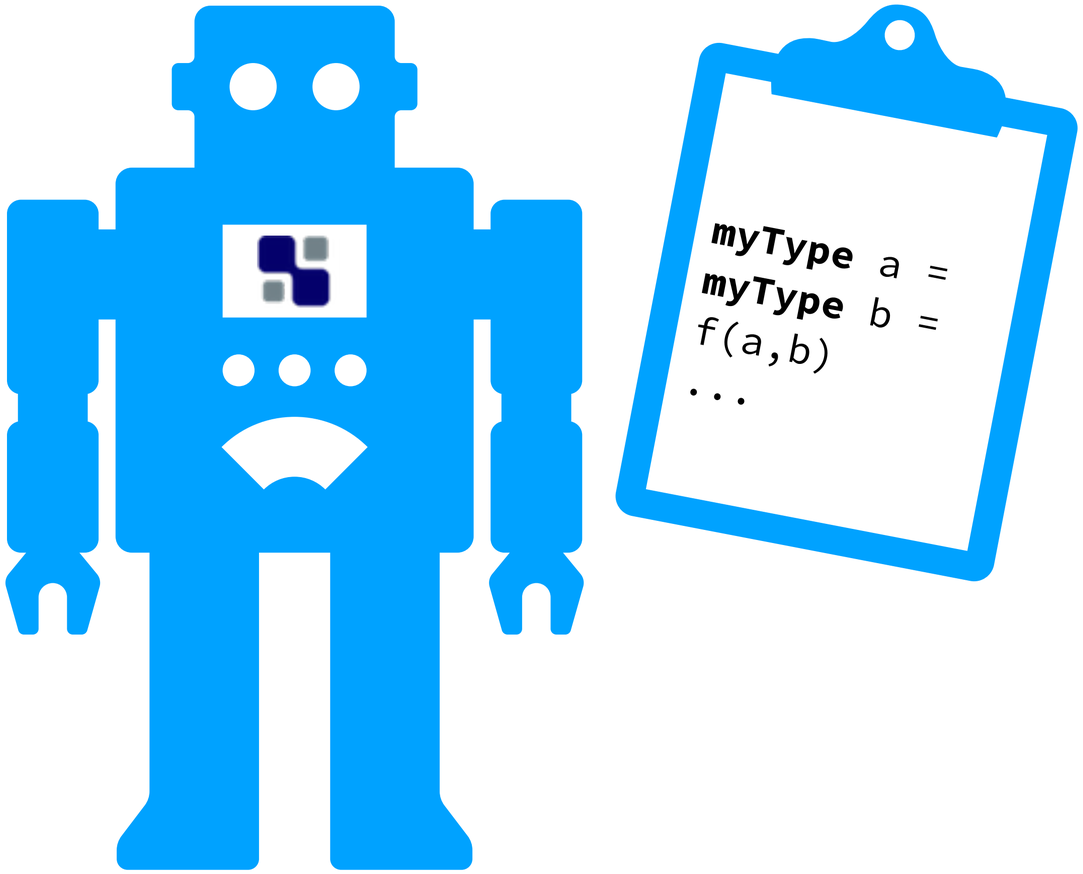


Supporting Datatype Research With TVM



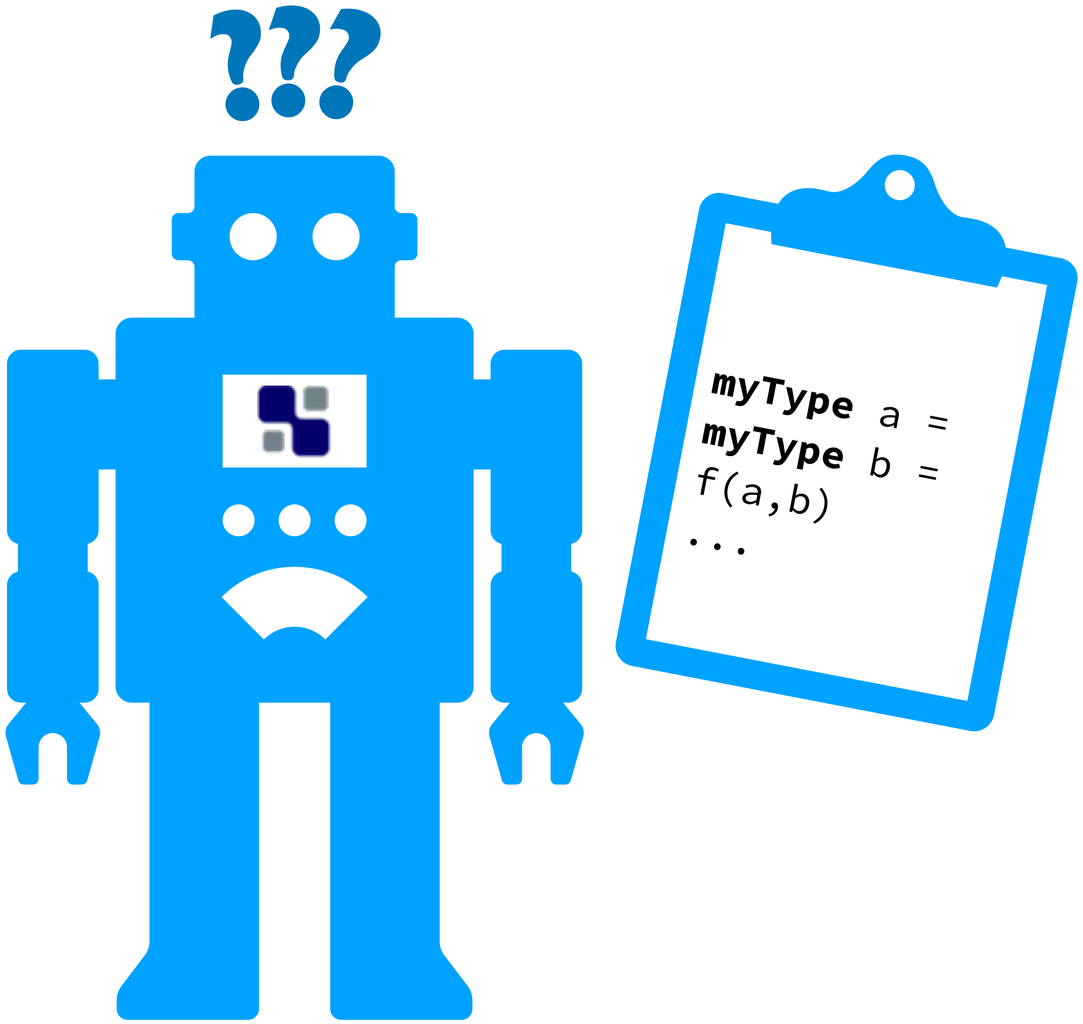
Supporting Datatype Research With TVM

Currently, TVM does not know how to interpret programs with arbitrary datatypes.



Supporting Datatype Research With TVM

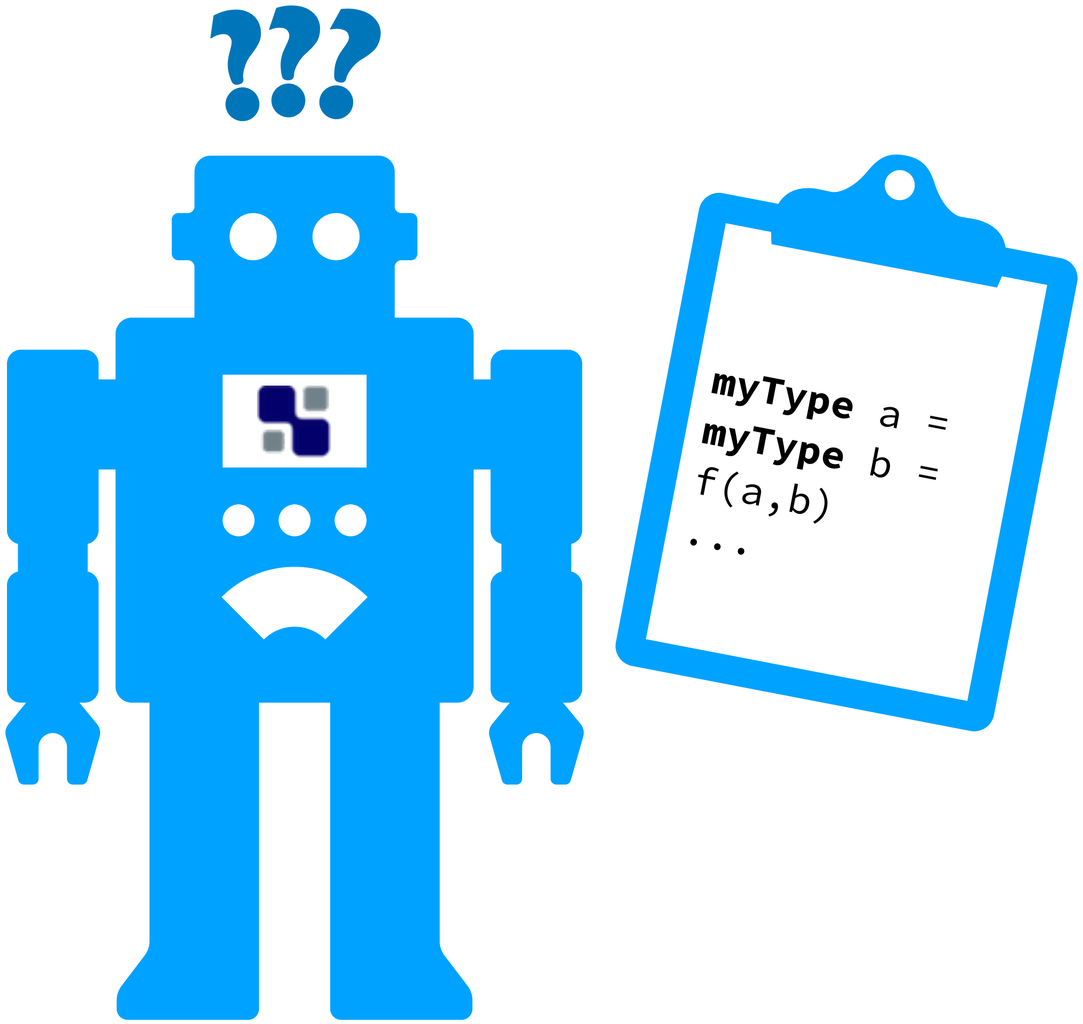
Currently, TVM does not know how to interpret programs with arbitrary datatypes.



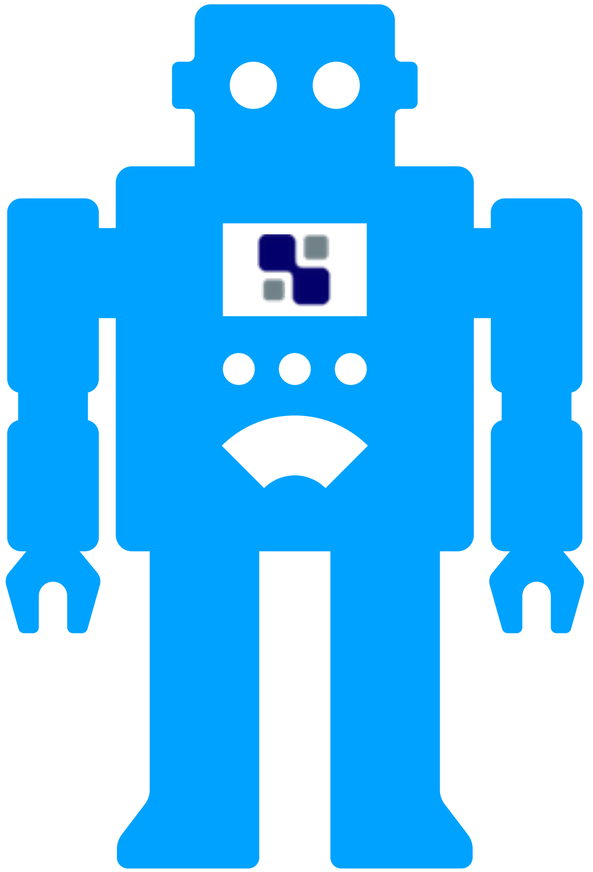
Supporting Datatype Research With TVM

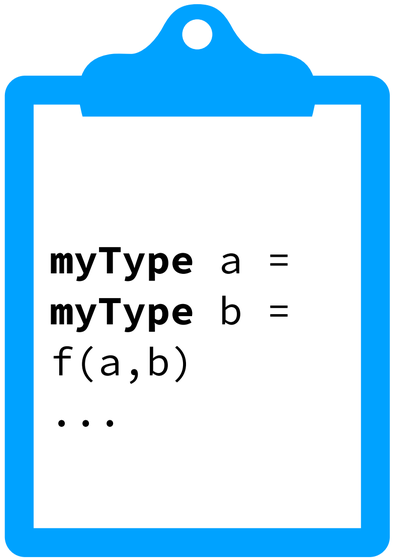
Currently, TVM does not know how to interpret programs with arbitrary datatypes.

Luckily, TVM is extensible!

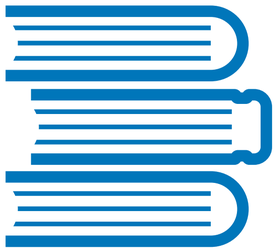


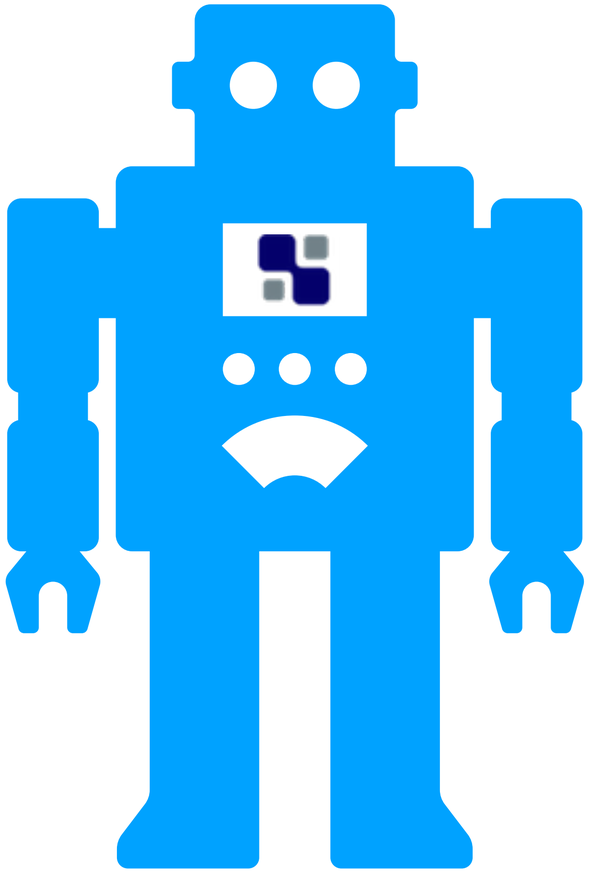
Supporting Datatype Research With TVM

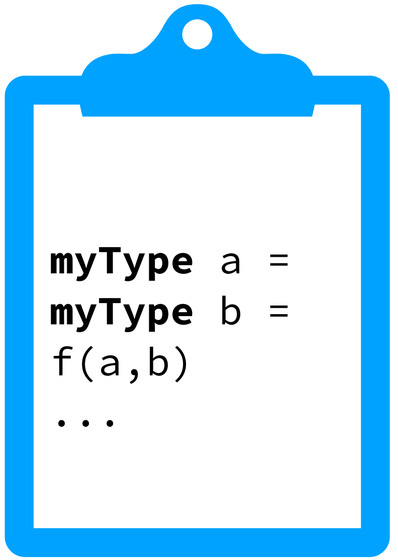




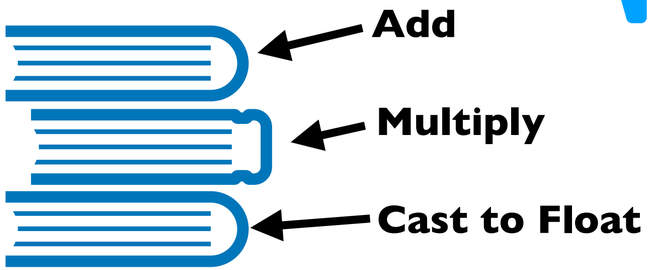
Supporting Datatype Research With TVM

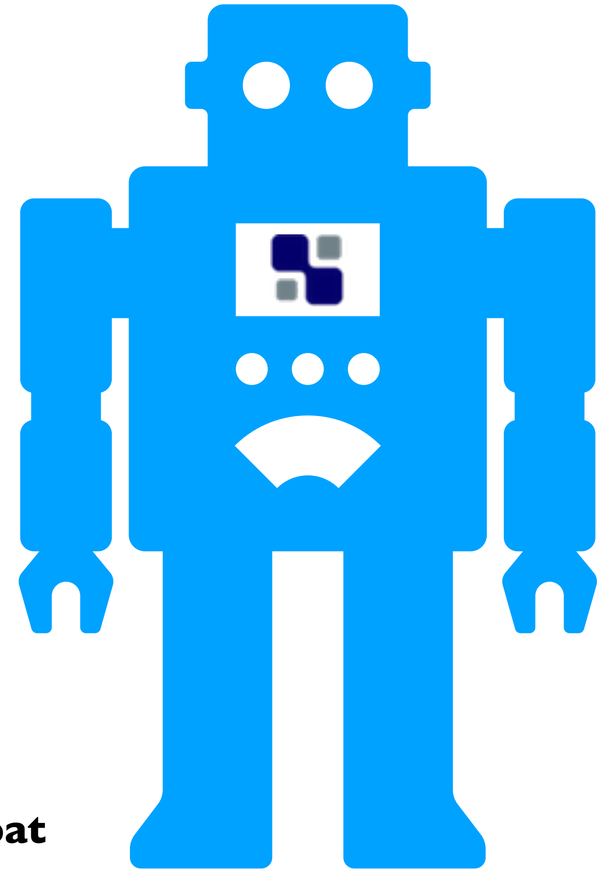


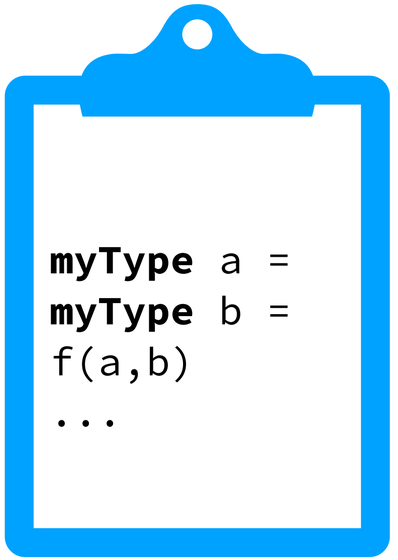




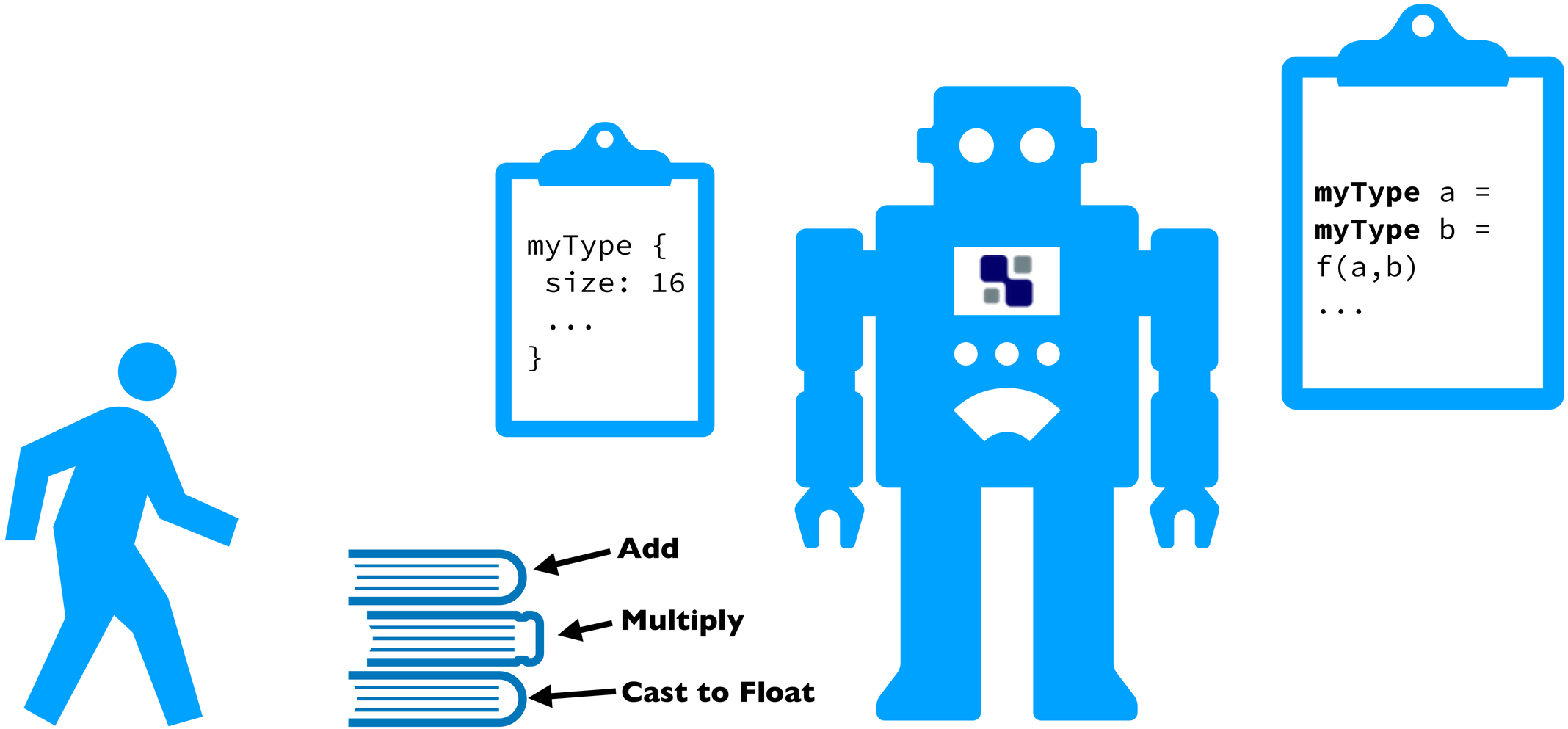
Supporting Datatype Research With TVM



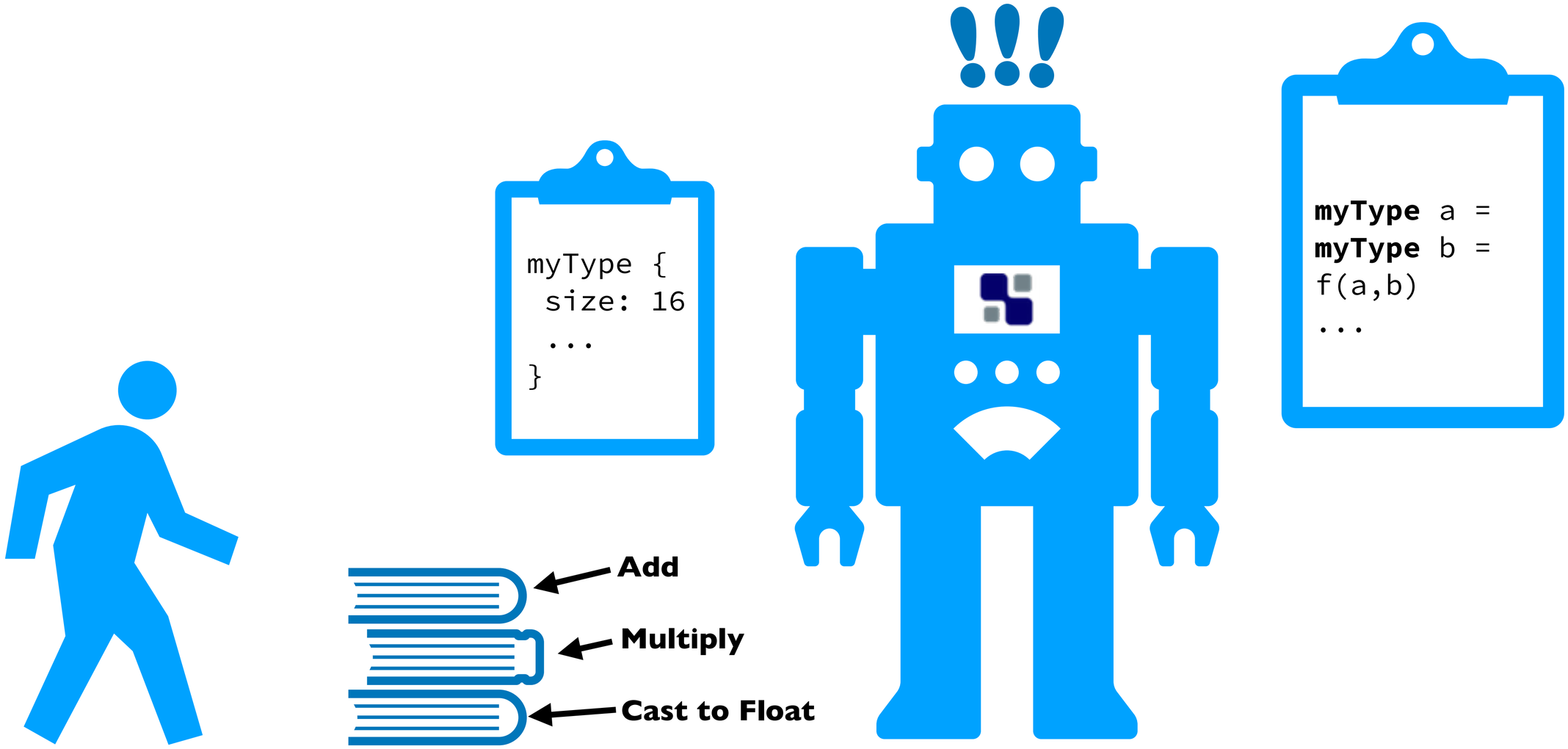




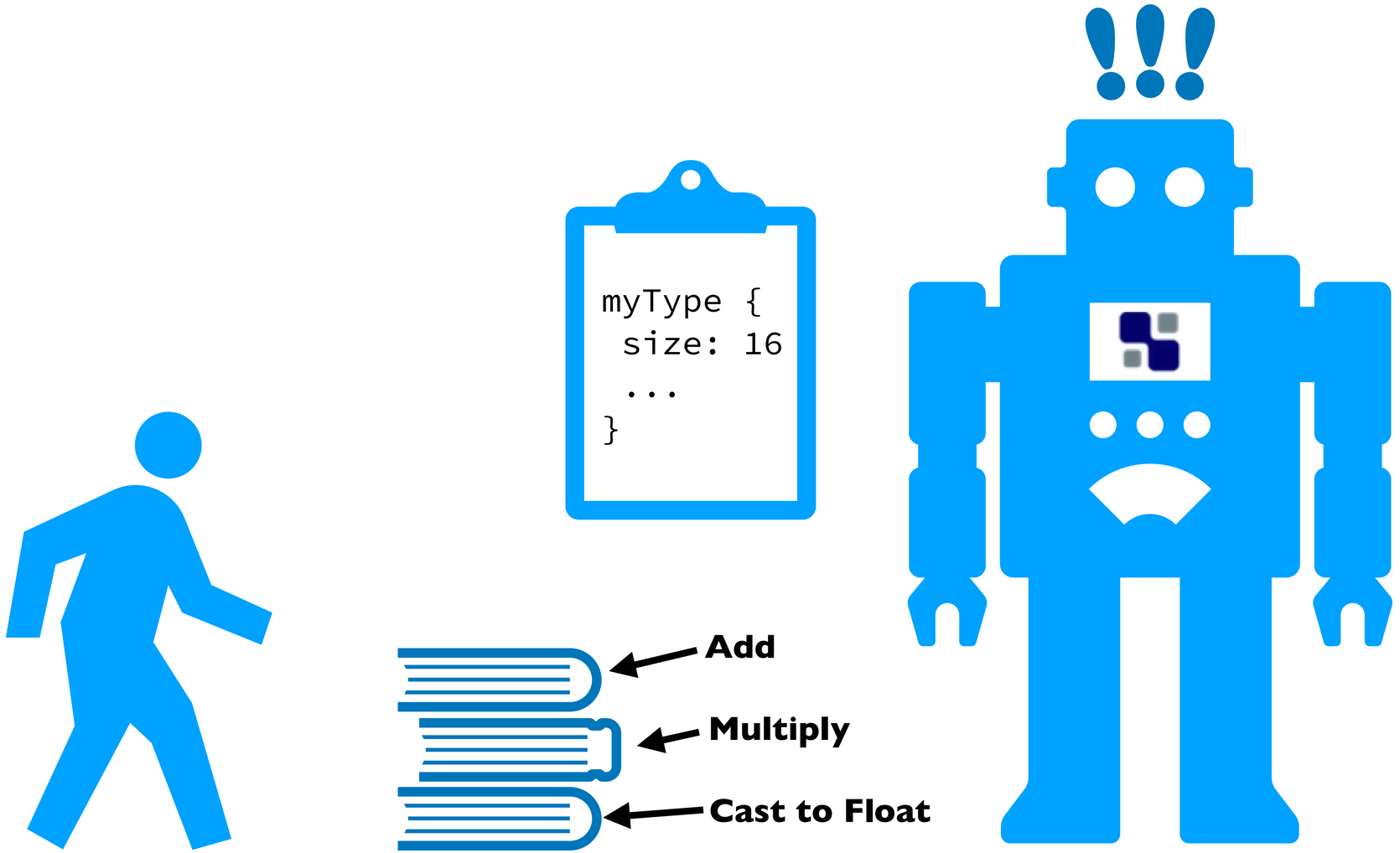
Supporting Datatype Research With TVM



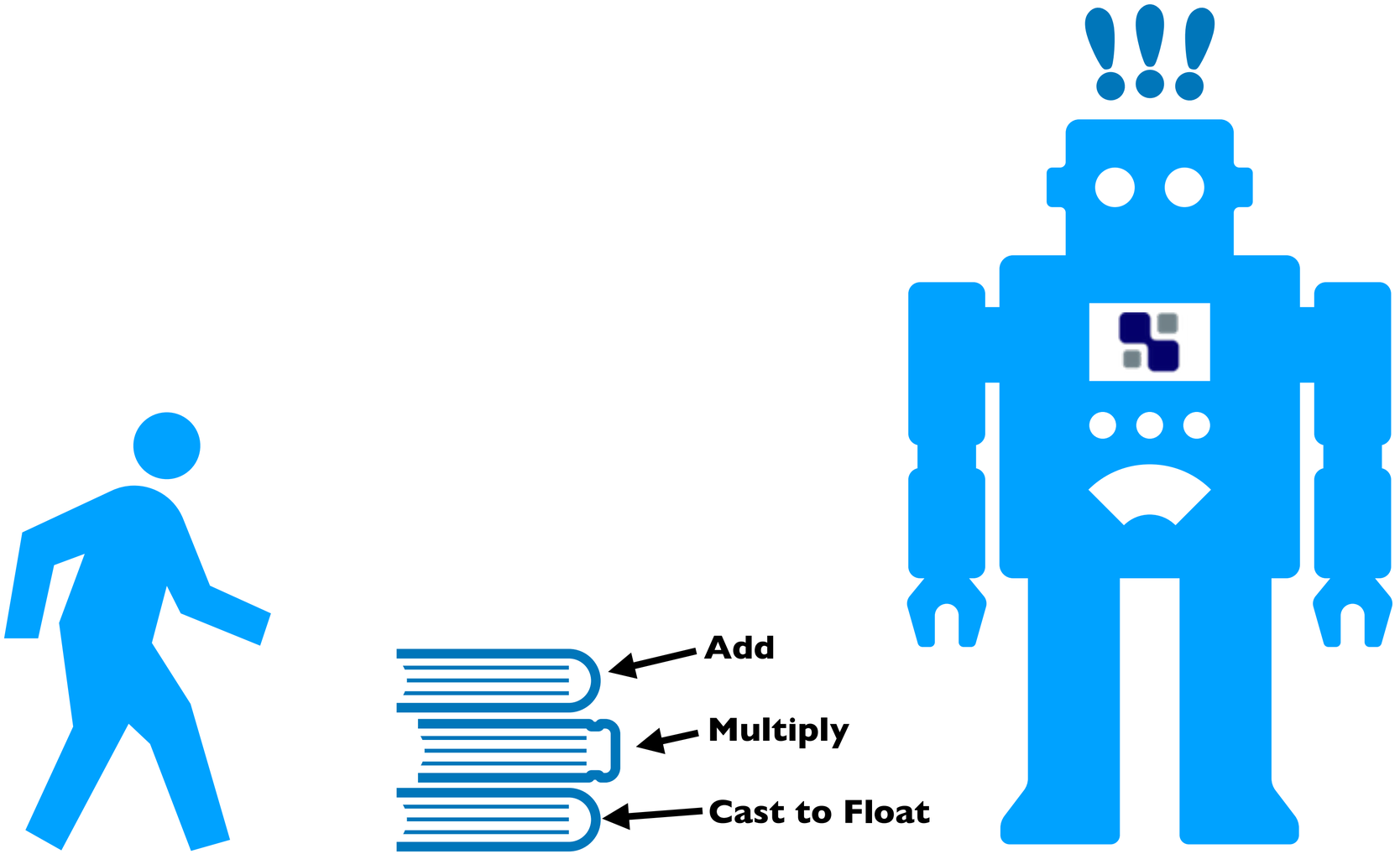
Supporting Datatype Research With TVM



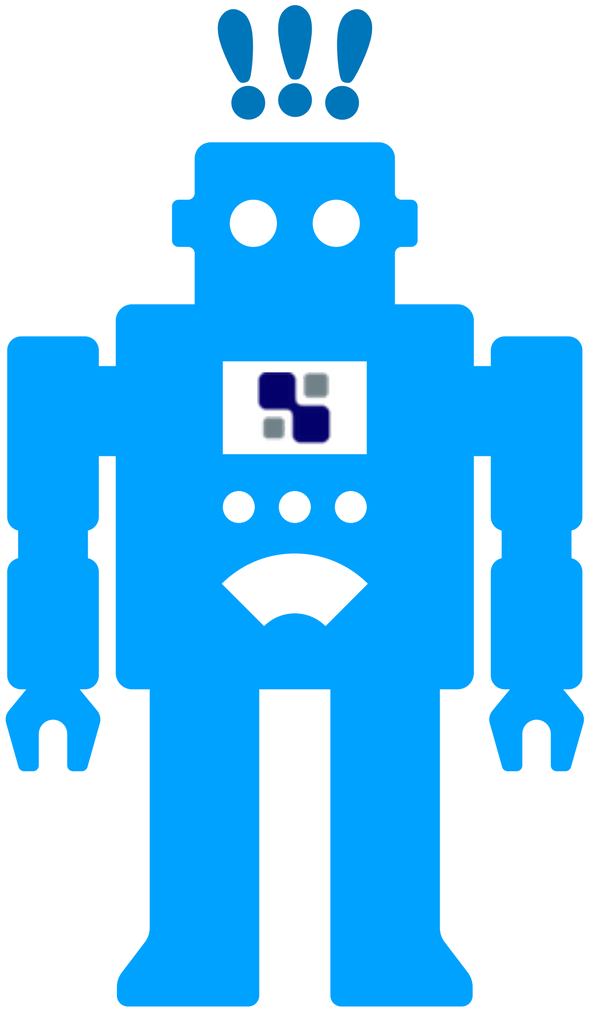
Supporting Datatype Research With TVM



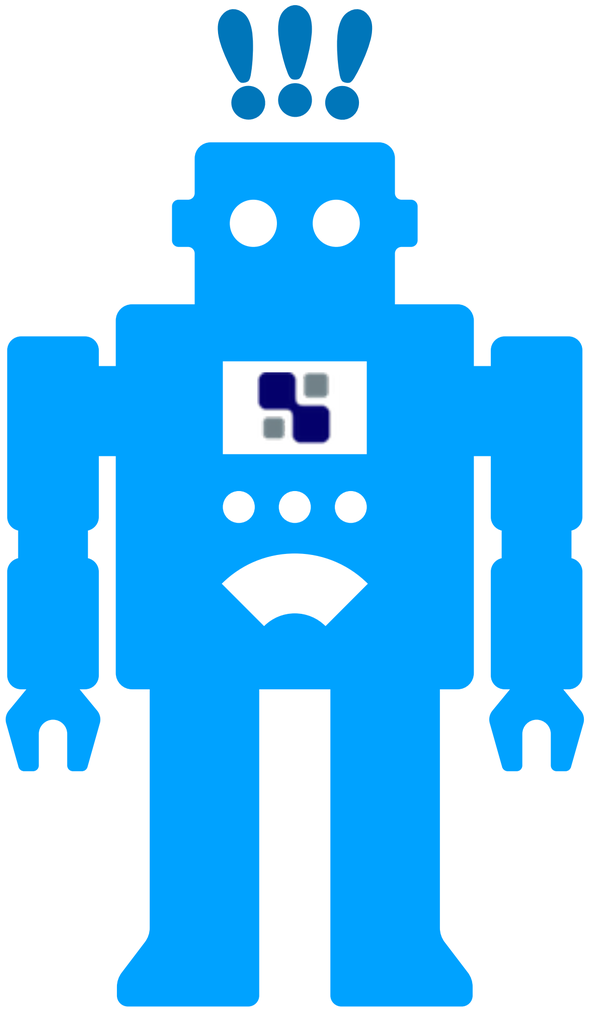
Supporting Datatype Research With TVM



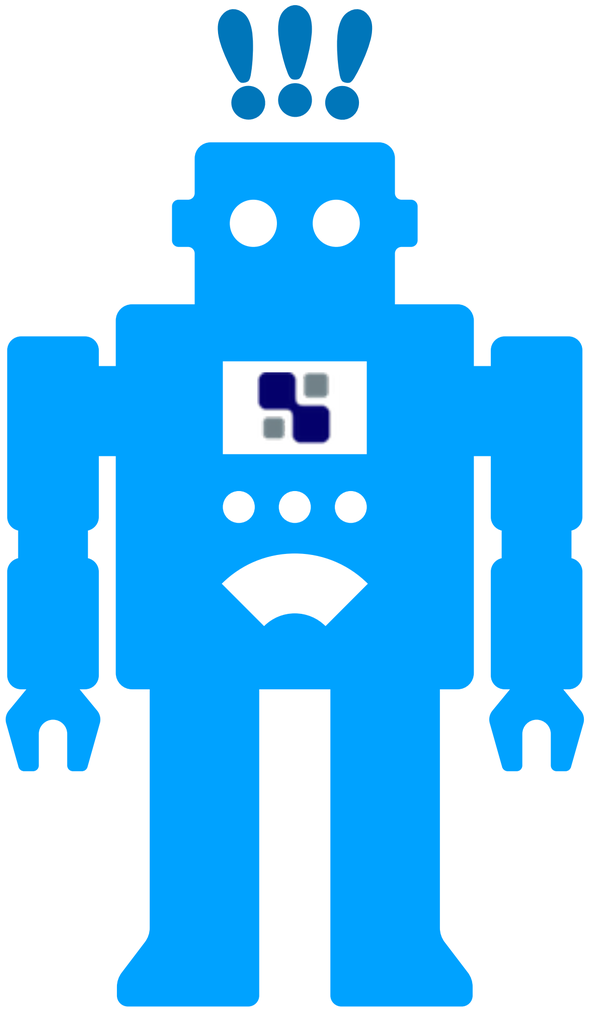
Supporting Datatype Research With TVM



Supporting Datatype Research With TVM



Supporting Datatype Research With TVM



Supporting Datatype Research With TVM

Supporting Datatype Research With TVM

1. User makes or finds a datatype library

Supporting Datatype Research With TVM

1. User makes or finds a datatype library
2. User writes a program using their datatypes directly

Supporting Datatype Research With TVM

1. User makes or finds a datatype library
2. User writes a program using their datatypes directly
3. User points TVM to the important functions (+, $*$) in the library

Supporting Datatype Research With TVM

1. User makes or finds a datatype library
2. User writes a program using their datatypes directly
3. User points TVM to the important functions (+, $*$) in the library
4. User gives TVM other information e.g. datatype size

Supporting Datatype Research With TVM

1. User makes or finds a datatype library
2. User writes a program using their datatypes directly
3. User points TVM to the important functions (+, $*$) in the library
4. User gives TVM other information e.g. datatype size
5. TVM compiles programs, handling the custom datatype by compiling calls into the provided library

Limitations of this Approach

Limitations of this Approach

Currently, we're only supporting **software implementations** of datatypes.

Limitations of this Approach

Currently, we're only supporting **software implementations** of datatypes.

Compiling for **hardware implementations** of the datatype is out of scope for the moment, but is planned.

Implementation

Datatype Registry

Datatype Registry

User registers their datatype:

Datatype Registry

User registers their datatype:

```
tvm.datatype.register("bfloat", 129)
```

Datatype Registry

User registers their datatype:

```
tvm.datatype.register("bfloat", 129)
```

Where 129 is a manually chosen code for the type

Datatype Usage

Datatype Usage

Then, in the code, we can give tensors the custom datatype:

Datatype Usage

Then, in the code, we can give tensors the custom datatype:

```
dtype=“custom[bfloat]16"
```

Datatype Usage

Then, in the code, we can give tensors the custom datatype:

```
dtype=“custom[bfloat]16"
```

Here, “16” is how we tell TVM the size of the datatype.

Operation Registry

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
```

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
```

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
```

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
        tvm.datatype.create_lower_func("BFloat16Add"),
```

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
        tvm.datatype.create_lower_func("BFloat16Add"),
        "Add", “llvm", “bfloat”)
```

Operation Registry

Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
        tvm.datatype.create_lower_func("BFloat16Add"),
        "Add", “llvm", “bfloat”)
```

This creates a lowering function which converts Adds to Calls to the bfloat add library function:

Operation Registry

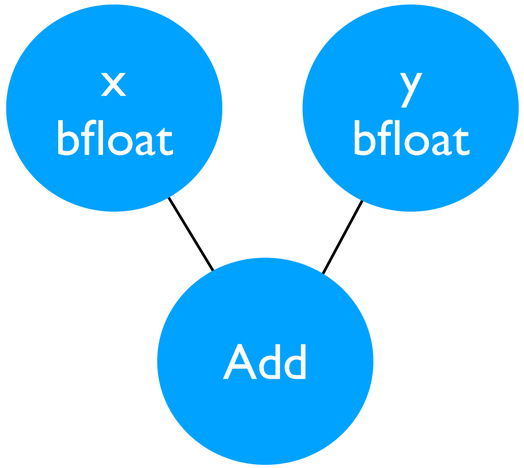
Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
        tvm.datatype.create_lower_func("BFloat16Add"),
        "Add", “llvm", “bfloat”)
```

This creates a lowering function which converts Adds to Calls to the bfloat add library function:



Operation Registry

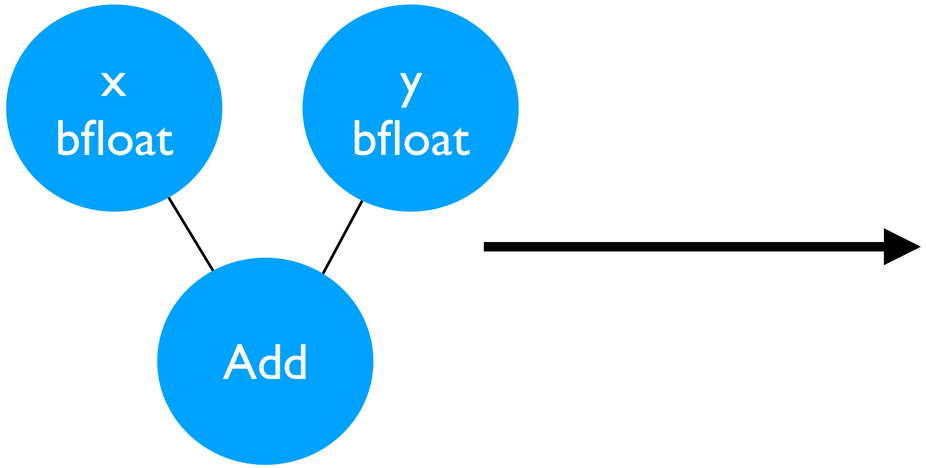
Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
        tvm.datatype.create_lower_func("BFloat16Add"),
        "Add", “llvm", “bfloat”)
```

This creates a lowering function which converts Adds to Calls to the bfloat add library function:



Operation Registry

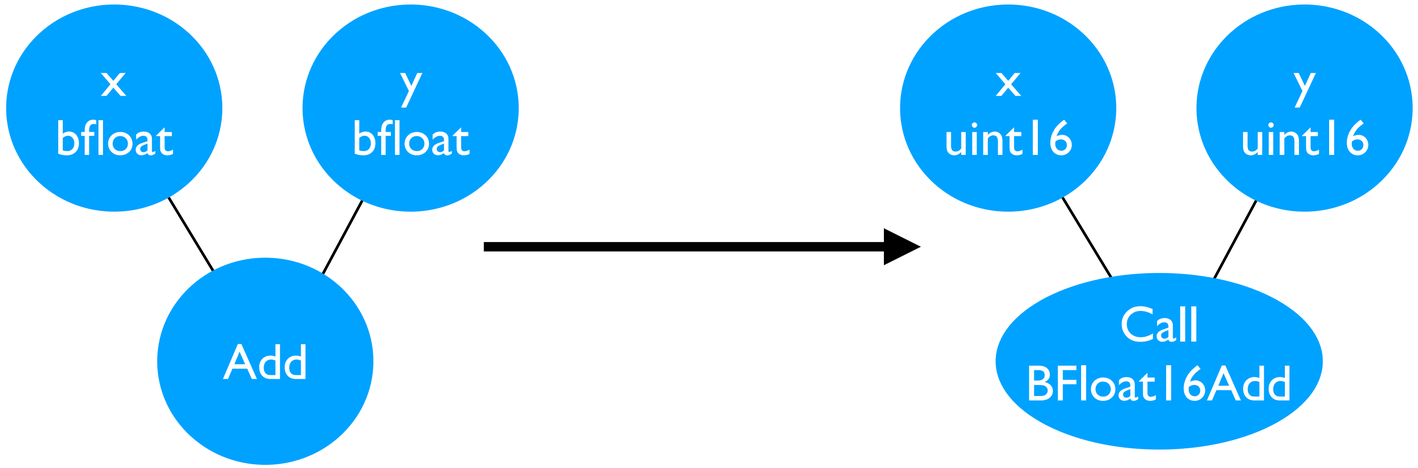
Then, the user registers a lowering function which lowers operations over the datatype:

```
tvm.datatype.register_op(
        lower_func,
        "Add", “llvm", “bfloat”)
```

In the common case, we will use:

```
tvm.datatype.register_op(
        tvm.datatype.create_lower_func("BFloat16Add"),
        "Add", “llvm", “bfloat”)
```

This creates a lowering function which converts Adds to Calls to the bfloat add library function:



Live Coding

Future Work

Future Work

- Short term: evaluating real deep learning models with modern datatypes

Future Work

- Short term: evaluating real deep learning models with modern datatypes
- Long term: supporting custom datatype hardware

Thank You!

CRISP
Center for Research in Intelligent
Storage and Processing in Memory

sampl

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

tvm.ai