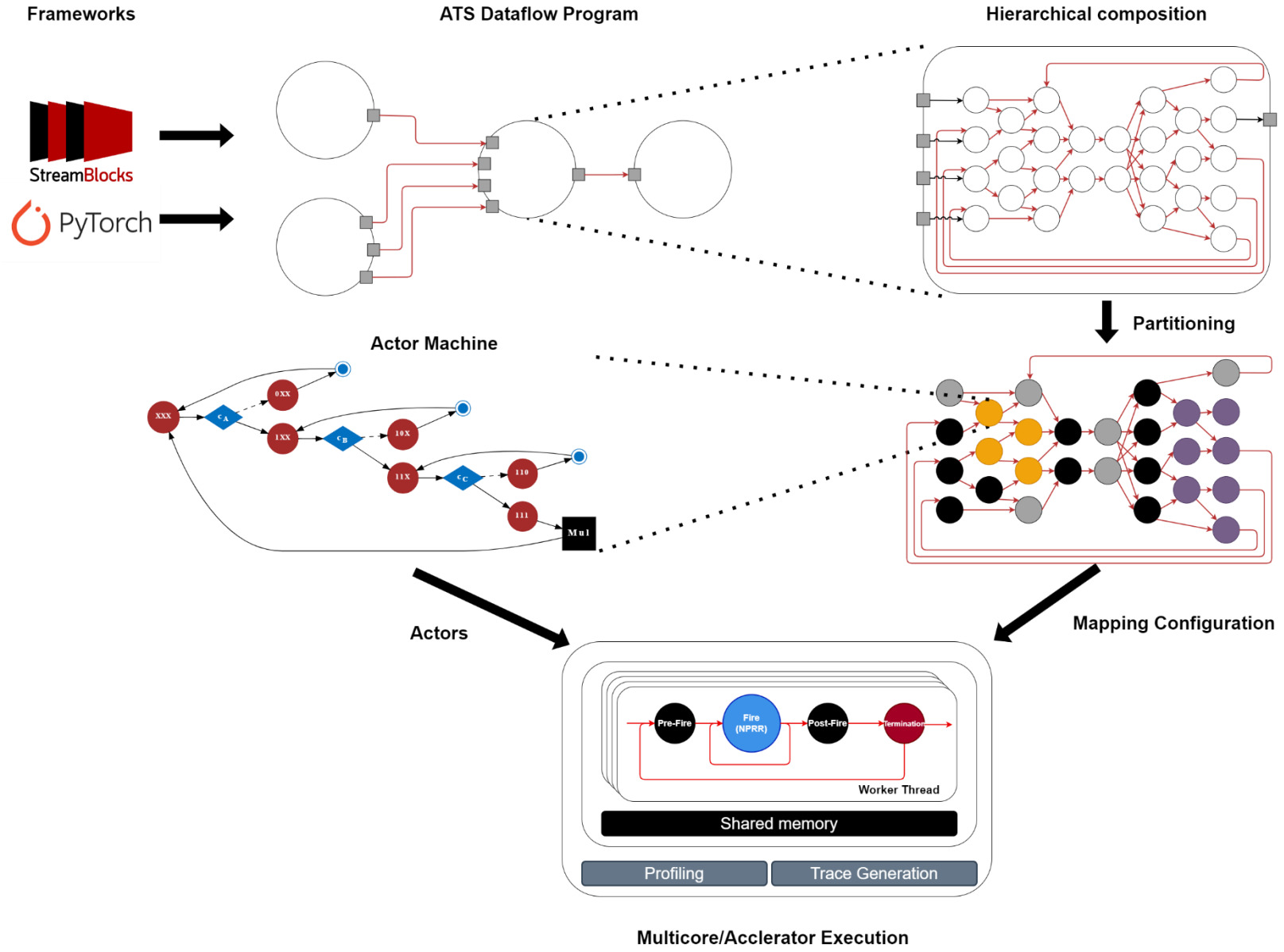
ART: An Actor transition systems RunTime for enabling efficient partitioning of neural network graphs

Endri BEZATI

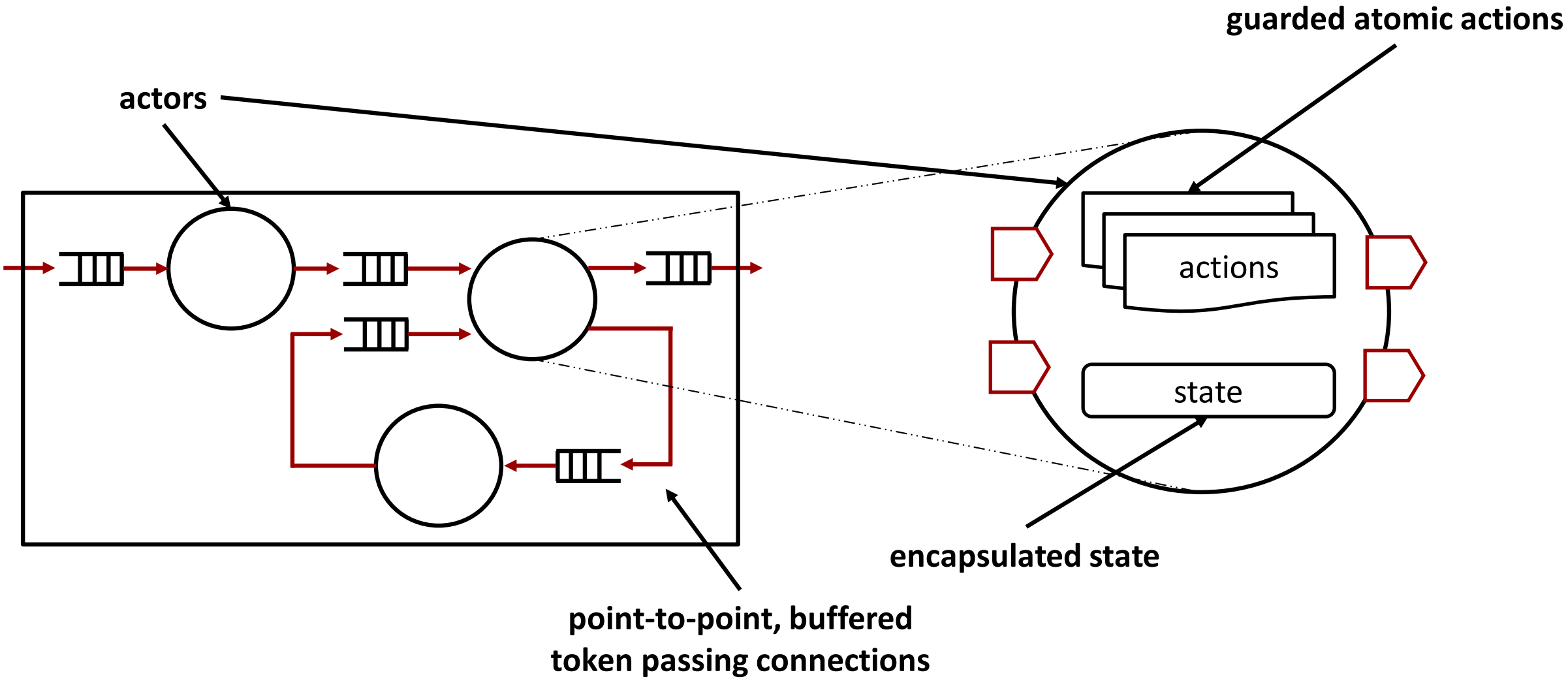
HUAWEI

Overview



HUAWEI

Actor transition systems (ATS) an extension to dataflow with firing



CAL as a notation for actors / Dataflow with Firing

```
actor Add () A, B ==> Out:

  action [a], [b] ==> [a + b]
  end
end
```

Actor Transition Systems - extensions to dataflow with firing

```
actor Add () A, B ==> Out:

  action [a], [b] ==> [a + b]
  end
end
```

```
actor Sum () A ==> X:

 s := 0;

 action [a] ==> [s]
 do
  s := s + a;
 end

end
```

Actor Transition Systems - extensions to dataflow with firing

```
actor Add () A, B ==> Out:

  action [a], [b] ==> [a + b]
  end
end
```

```
actor Sum () A ==> X:

 s := 0;

 action [a] ==> [s]
 do
  s := s + a;
 end

end
```

```
actor Route () A ==> X, Y:

 action [a] ==> X: [a]
 guard P(a)
 end

 action [a] ==> Y: [a]
 guard not P(a)
 end

end
```

Actor Transition Systems - extensions to dataflow with firing

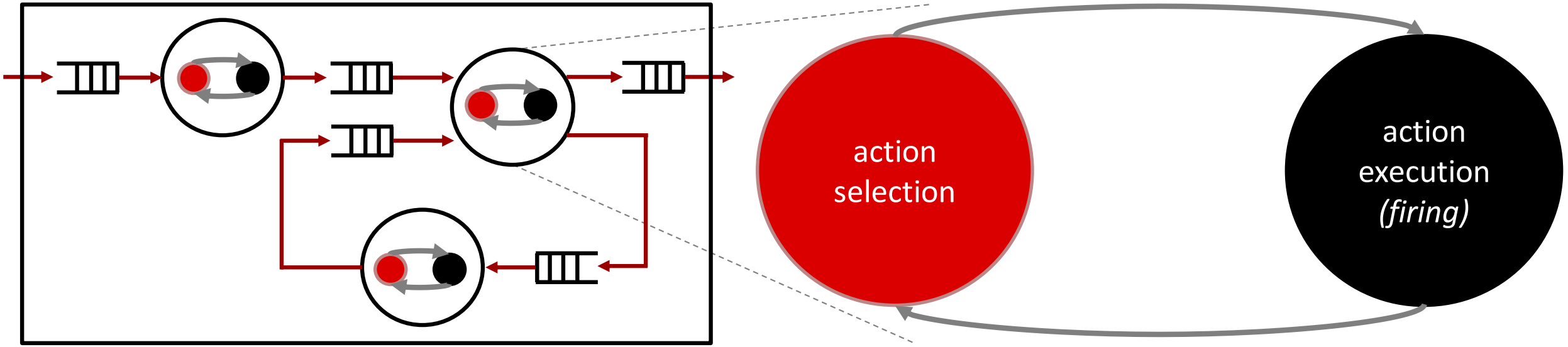
```
actor Add () A, B ==> Out:

  action [a], [b] ==> [a + b]
  end
end
```

```
actor Sum () A ==> X:

 s := 0;

 action [a] ==> [s]
 do
  s := s + a;
 end

end
```

```
actor Route () A ==> X, Y:

 action [a] ==> X: [a]
 guard P(a)
 end

 action [a] ==> Y: [a]
 guard not P(a)
 end

end
```

```
actor Route () A ==> X, Y:

 A: action [a] ==> X: [a]
 guard P(a)
 end

 B: action [a] ==> Y: [a]
 end

 priority
  A > B;
 end

end
```

Actor execution model



Action Selection based on the Actor Machine

```
actor Multiplication () A, B ==> Out:

  Mul: action [a], [b] ==> [a + b]
  end
end
```

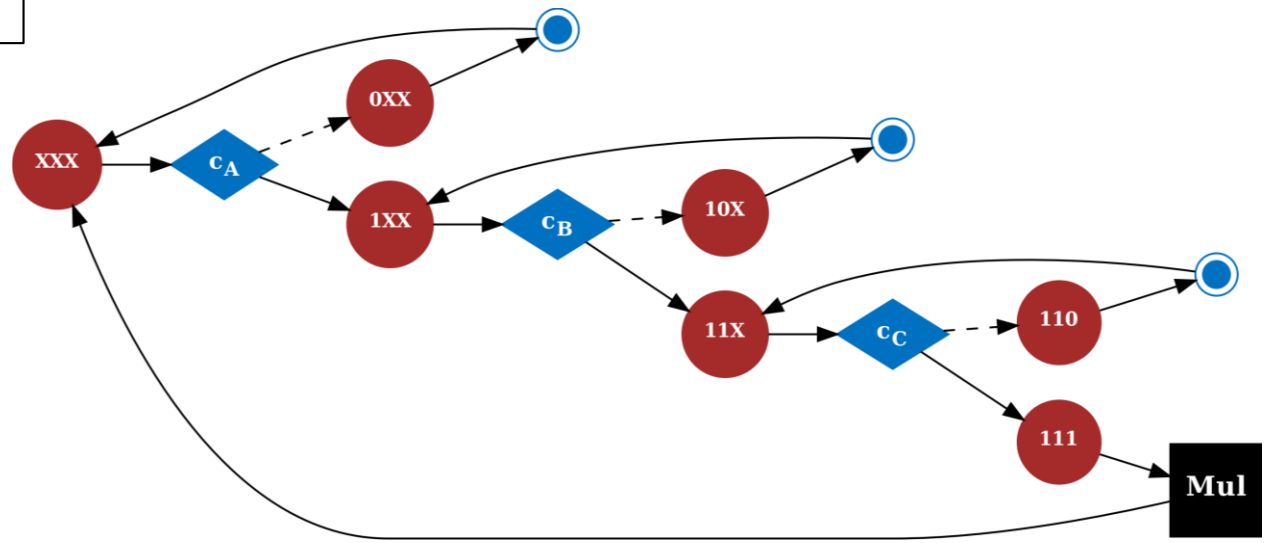
c_A : Available token on port ***A***
c_B : Available token on port ***B***
c_C : Available space on port ***Out***

Action Selection based on the Actor Machine

```
actor Multiplication () A, B ==> Out:

  Mul: action [a], [b] ==> [a + b]
  end
end
```

c_A : Available token on port ***A***
c_B : Available token on port ***B***
c_C : Available space on port ***Out***

Actor state

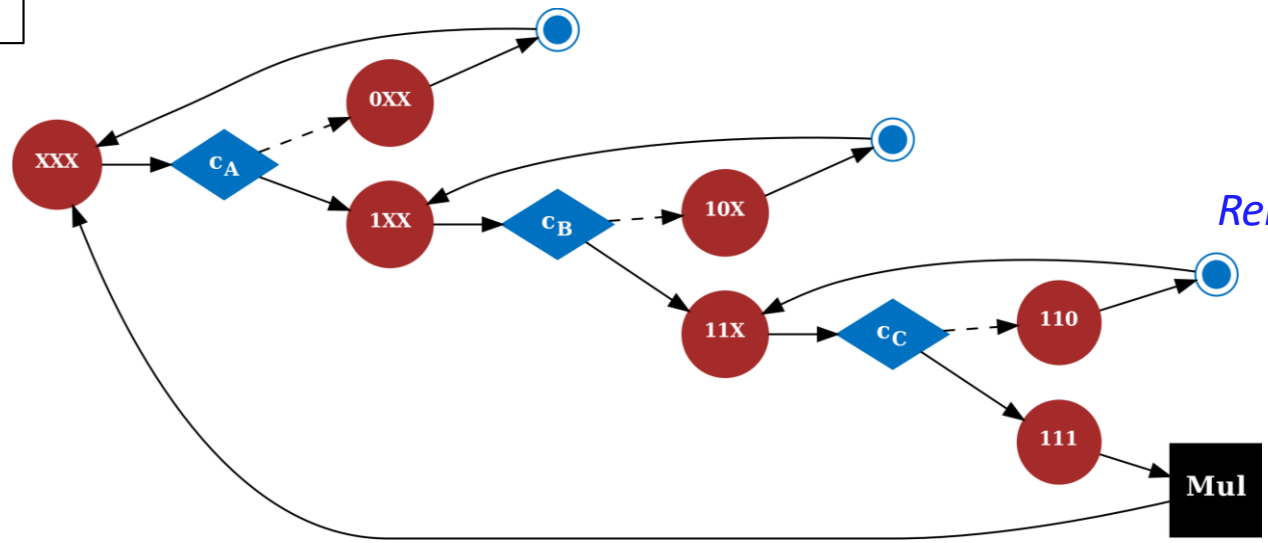


Action Selection based on the Actor Machine

```
actor Multiplication () A, B ==> Out:

  Mul: action [a], [b] ==> [a + b]
  end
end
```

c_A : Available token on port ***A***
c_B : Available token on port ***B***
c_C : Available space on port ***Out***

Actor state



Remember what conditions Have been checked

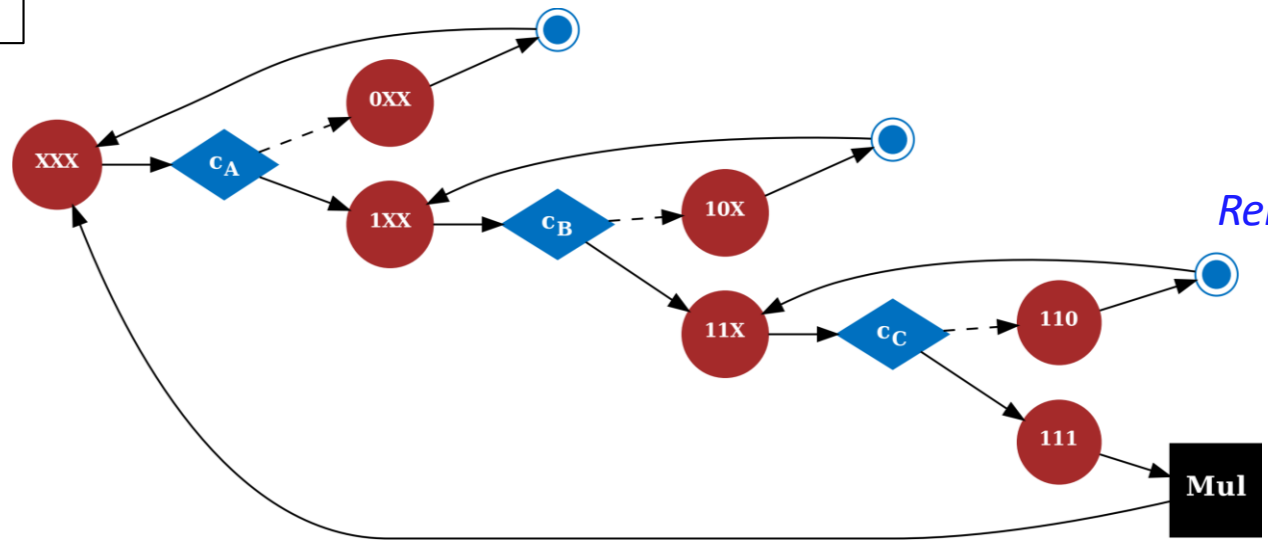
HUAWEI

Action Selection based on the Actor Machine

```
actor Multiplication () A, B ==> Out:

  Mul: action [a], [b] ==> [a + b]
  end
end
```

c_A : Available token on port ***A***
c_B : Available token on port ***B***
c_C : Available space on port ***Out***

Actor state



Remember what conditions Have been checked

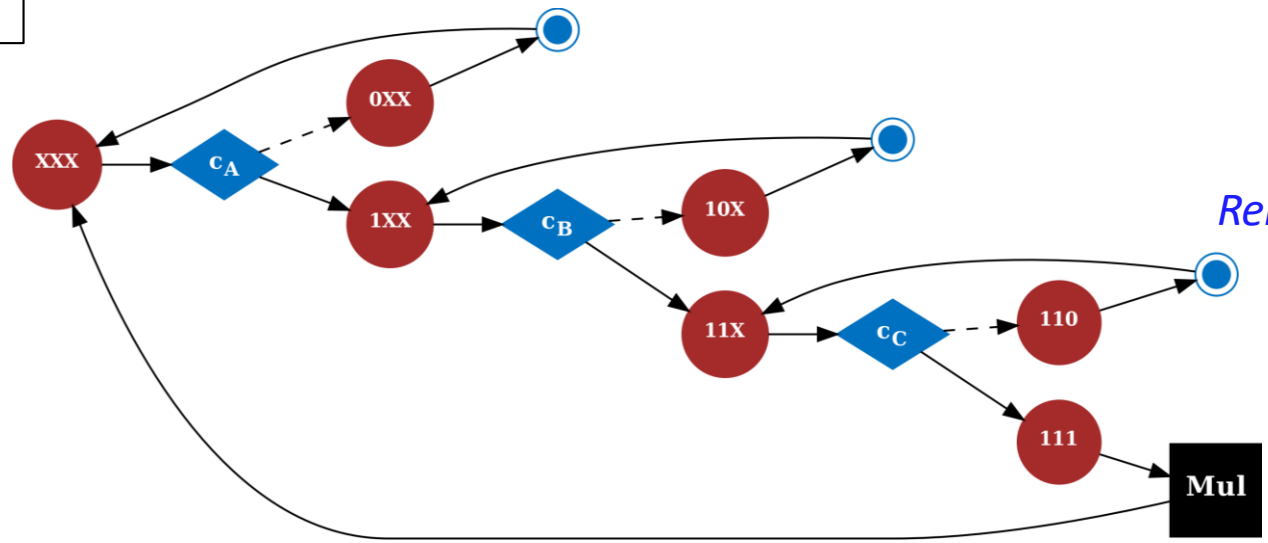
Execute the body of the action

HUAWEI

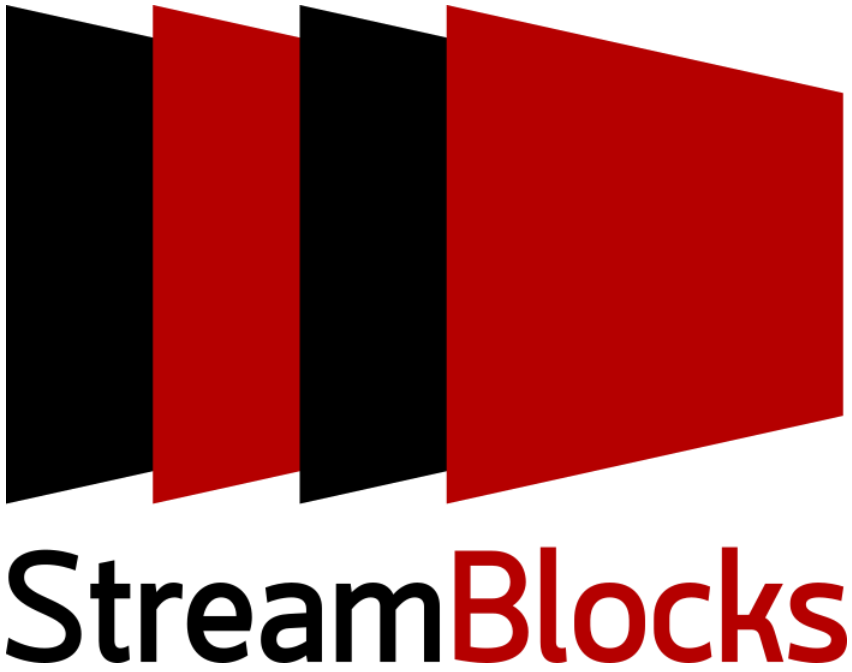
Action Selection based on the Actor Machine

```
actor Multiplication () A, B ==> Out:

  Mul: action [a], [b] ==> [a + b]
  end
end
```

c_A : Available token on port ***A***
c_B : Available token on port ***B***
c_C : Available space on port ***Out***



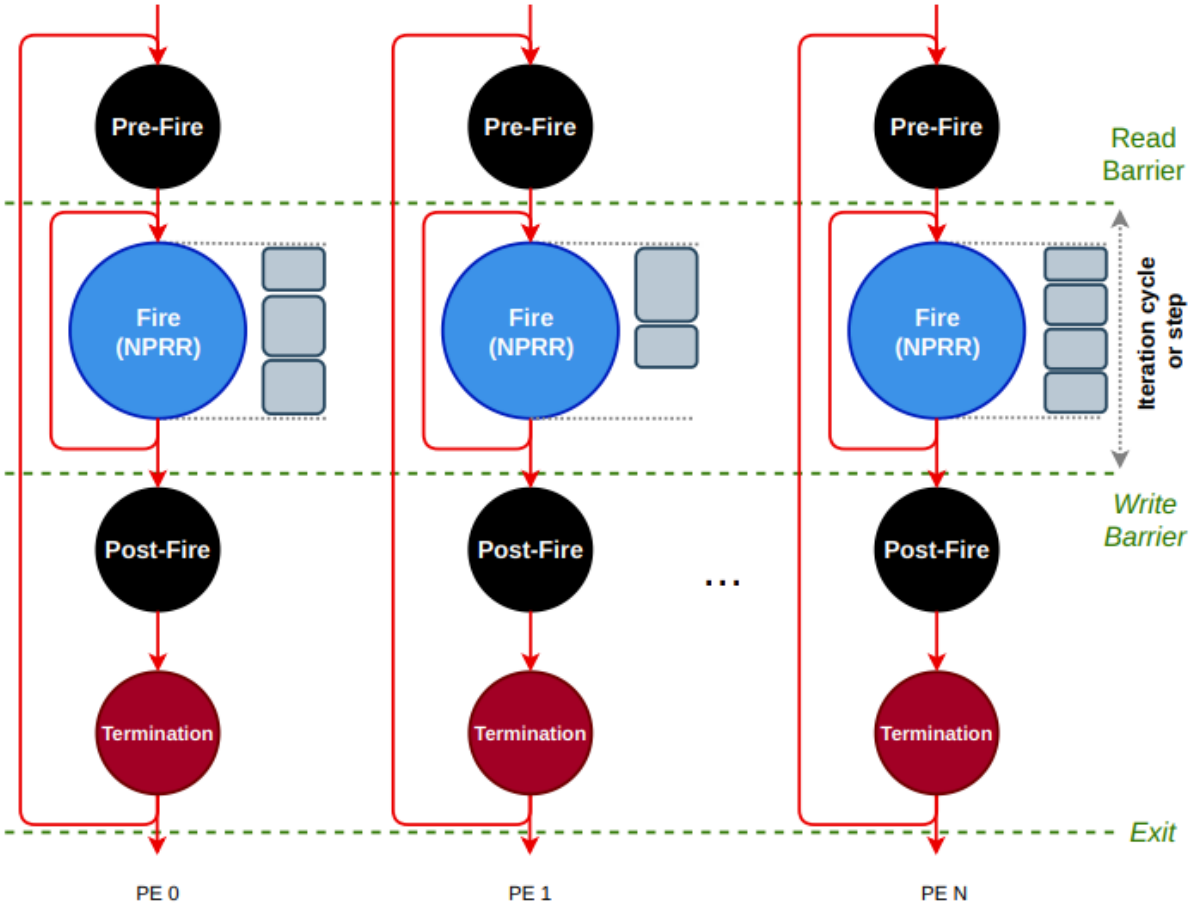
HUAWEI



- ATS Model of Computation
- CAL as programming language
- Actor Machine for Action Selection
- ART Runtime
- Code generation

https://github.com/streamblocks

Actor transition systems RunTime



Bulk Synchronous Parallel execution between PEs

For an actor to fire (execute):

1. The actor is mapped into a PE
2. The actor is selected for execution from a set of actors that are mapped on the PE
3. The actor firing conditions are checked
4. Iff the firing conditions are fulfilled the actor fires, otherwise chose another actor

End of execution:

1. All PEs sleep
2. Try once again to execute actors on PEs, if none has fire then terminate

Deadlock detection:

1. Some PEs sleep because of no input data
2. An actor can fire but can not write to its output port

Graph partitioning

- **Partitioning tools**
  - Metis : **multilevel recursive-bisection**, multilevel k-way, and multi-constraint partitioning
  - PatoH : multilevel hypergraph partitioning
- **Based on profiling information**
  - Actor weight(s)
    - Abstract weights : CAL statements or MAC operations
    - Platform profiling : using hardware counters
    - (Optional) Memory used by actor
  - Connection weight
    - Data type
    - Number of tokens traversed given an input stimulus

Placing actors to processing elements

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
    <network id="nn.TopInference"/>
    <partitioning>
        <partition id="0" >
            <instance id="i_load_arg2_1"/>
            <instance id="i_print_addmm_2"/>
            <instance id="i_convolution"/>
            <instance id="i_relu_"/>
            . . .
        </partition>
        <partition id="1“ >
            <instance id="i_convolution_4"/>
            <instance id="i_relu__4"/>
            <instance id="i_max_pool2d_with_indices_2"/>
            <instance id="i__adaptive_avg_pool2d"/>
             . . .
         </partition>
    </partitioning>
    <connections>
        <connection source="i_load_arg2_1" source-port="OUT" target="i_convolution" target-port="arg2_1" size="16"/>
        <connection source="i_addmm_2" source-port="addmm_2" target="i_print_addmm_2" target-port="IN" size="16"/>
        <connection source="i_convolution" source-port="convolution" target="i_relu_" target-port="convolution" size="16"/>
        <connection source="i_relu_" source-port="relu_" target="i_max_pool2d_with_indices" target-port="relu_" size="16"/>
        <connection source="i_max_pool2d_with_indices" source-port="max_pool2d_with_indices_0" target="i_convolution_1" target-port="getitem" size="16"/>
        . . .
     </connections>
</configuration>
```

Instances in first partition, pinned to core 0

Placing actors to processing elements

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
    <network id="nn.TopInference"/>
    <partitioning>
        <partition id="0" >
            <instance id="i_load_arg2_1"/>
            <instance id="i_print_addmm_2"/>
            <instance id="i_convolution"/>
            <instance id="i_relu_"/>
            . . .
        </partition>
        <partition id="1“ >
            <instance id="i_convolution_4"/>
            <instance id="i_relu__4"/>
            <instance id="i_max_pool2d_with_indices_2"/>
            <instance id="i__adaptive_avg_pool2d"/>
             . . .
         </partition>
    </partitioning>
    <connections>
        <connection source="i_load_arg2_1" source-port="OUT" target="i_convolution" target-port="arg2_1" size="16"/>
        <connection source="i_addmm_2" source-port="addmm_2" target="i_print_addmm_2" target-port="IN" size="16"/>
        <connection source="i_convolution" source-port="convolution" target="i_relu_" target-port="convolution" size="16"/>
        <connection source="i_relu_" source-port="relu_" target="i_max_pool2d_with_indices" target-port="relu_" size="16"/>
        <connection source="i_max_pool2d_with_indices" source-port="max_pool2d_with_indices_0" target="i_convolution_1" target-port="getitem" size="16"/>
        . . .
     </connections>
</configuration>
```

Instances in first partition, pinned to core 0

Instances in second partition, pinned to core 1

Placing actors to processing elements

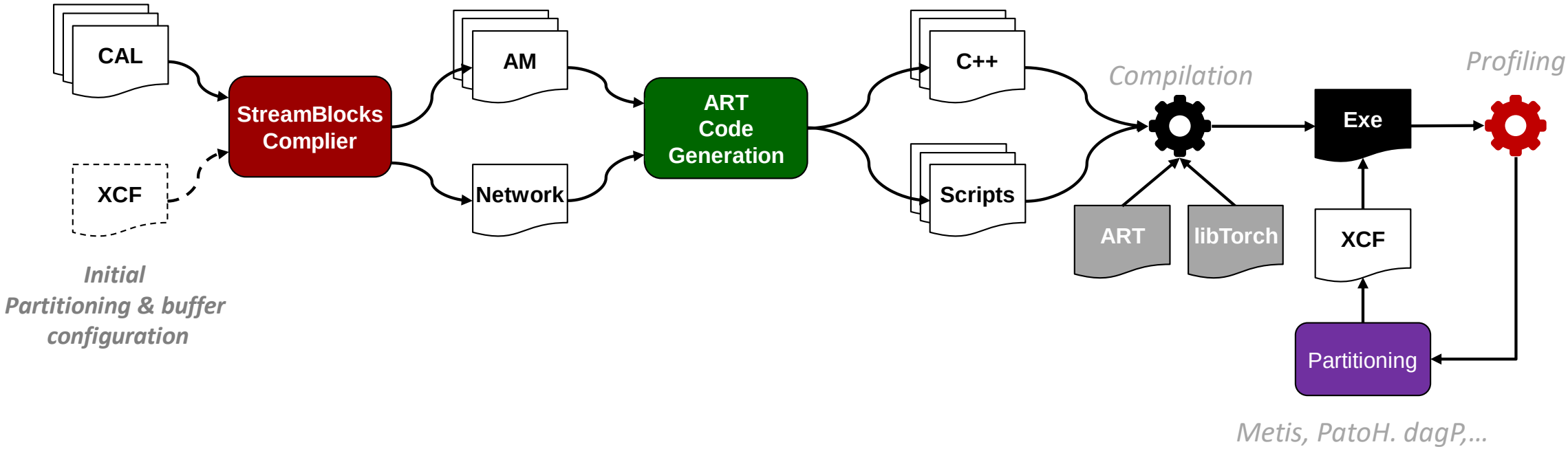
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
    <network id="nn.TopInference"/>
    <partitioning>
        <partition id="0" >
            <instance id="i_load_arg2_1"/>
            <instance id="i_print_addmm_2"/>
            <instance id="i_convolution"/>
            <instance id="i_relu_"/>
            . . .
        </partition>
        <partition id="1“ >
            <instance id="i_convolution_4"/>
            <instance id="i_relu__4"/>
            <instance id="i_max_pool2d_with_indices_2"/>
            <instance id="i__adaptive_avg_pool2d"/>
             . . .
         </partition>
    </partitioning>
    <connections>
        <connection source="i_load_arg2_1" source-port="OUT" target="i_convolution" target-port="arg2_1" size="16"/>
        <connection source="i_addmm_2" source-port="addmm_2" target="i_print_addmm_2" target-port="IN" size="16"/>
        <connection source="i_convolution" source-port="convolution" target="i_relu_" target-port="convolution" size="16"/>
        <connection source="i_relu_" source-port="relu_" target="i_max_pool2d_with_indices" target-port="relu_" size="16"/>
        <connection source="i_max_pool2d_with_indices" source-port="max_pool2d_with_indices_0" target="i_convolution_1" target-port="getitem" size="16"/>
        . . .
     </connections>
</configuration>
```

Instances in first partition, pinned to core 0

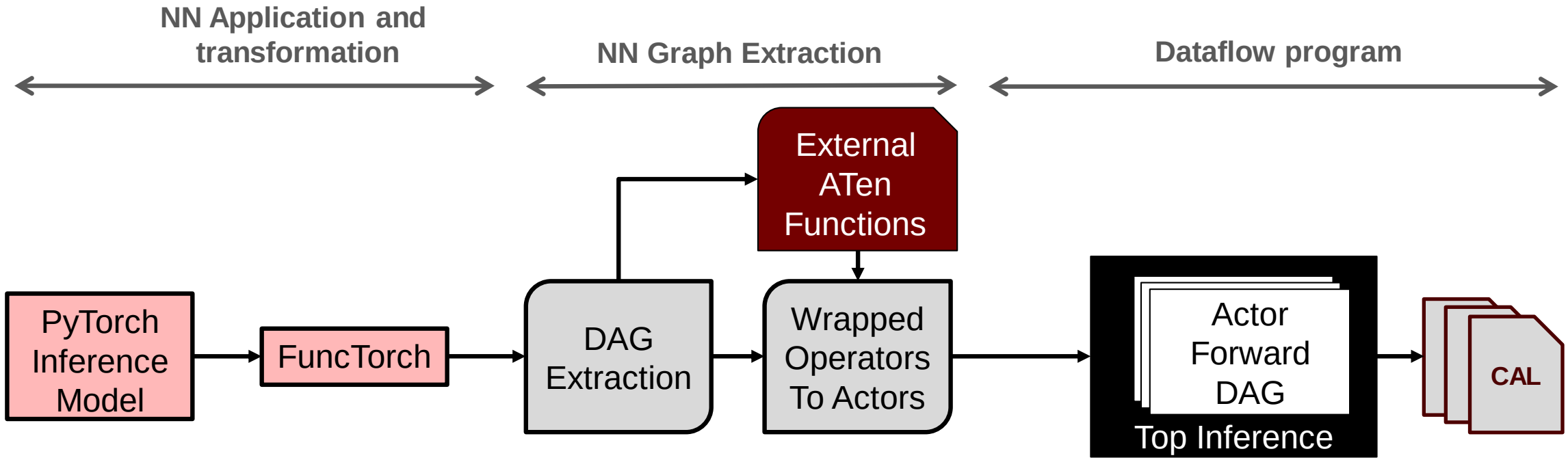
Instances in second partition, pinned to core 1

Configurable buffer size

Compilation infrastructure



From PyTorch to CAL



Torch FX Graph Operators to CAL actors (1)

- **Torch FX nodes kind considered**
  - placeholder : input arguments
  - call_function : ATen or python functions
  - output : return value(s), in general an array or tuple

- **call_function Node**
  - Only nodes with ‘schema’ attributed considered
    - Nodes without a ‘schema’ are python built-in operators like (getitem of a tuple)
    - Arguments: Tensors, arrays, literals (boolean, integer and float) and None
  - Operation name given by `node.target._schema.name`

- **placeholder**
  - Input arguments: parameters, buffers, input data and expected output for training
    - Parameters, and buffers are treated as constant state variables

- **output**
  - Return values: tensor or a tuple of tensors

Torch FX Graph Operators to CAL actors (2)

From ATen operator to CAL external function

```
call_function | convolution | aten.convolution.default |(arg2_1, arg0_1, arg0_2, [4,4], [2,2], [1,1], False, [0,0], 1 )
```

- Find the unique operators used in the Torch FX graph
- Create a unique external CAL function per ATen operator
  - For every argument find its type and convert it to a CAL type
  - The argument names are not stored on the Torch FX, naming them with ‘arg_<number>’

```
external function convolution(Tensor arg_0,
                              Tensor arg_1,
                              Tensor arg_2,
                              List(type:int(size=64)) arg_3,
                              List(type:int(size=64)) arg_4,
                              List(type:int(size=64)) arg_5,
                              bool arg_6,
                              List(type:int(size=64)) arg_7,
                              int(size=64) arg_8)
                              --> Tensor
end
```

Torch FX Graph Operators to CAL actors (3)

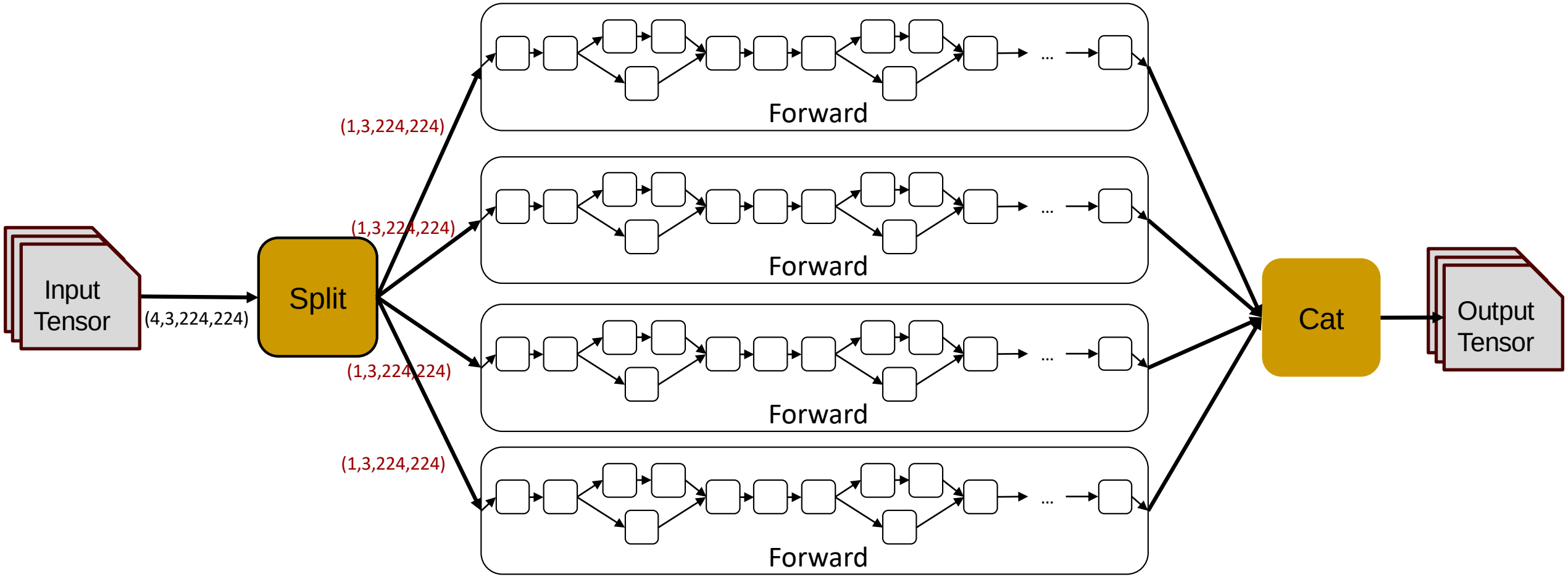
From ATen operator to CAL Actors

```
call_function | convolution | aten.convolution.default |(arg2_1, arg0_1, arg0_2, [4,4], [2,2], [1,1], False, [0,0], 1 )
```

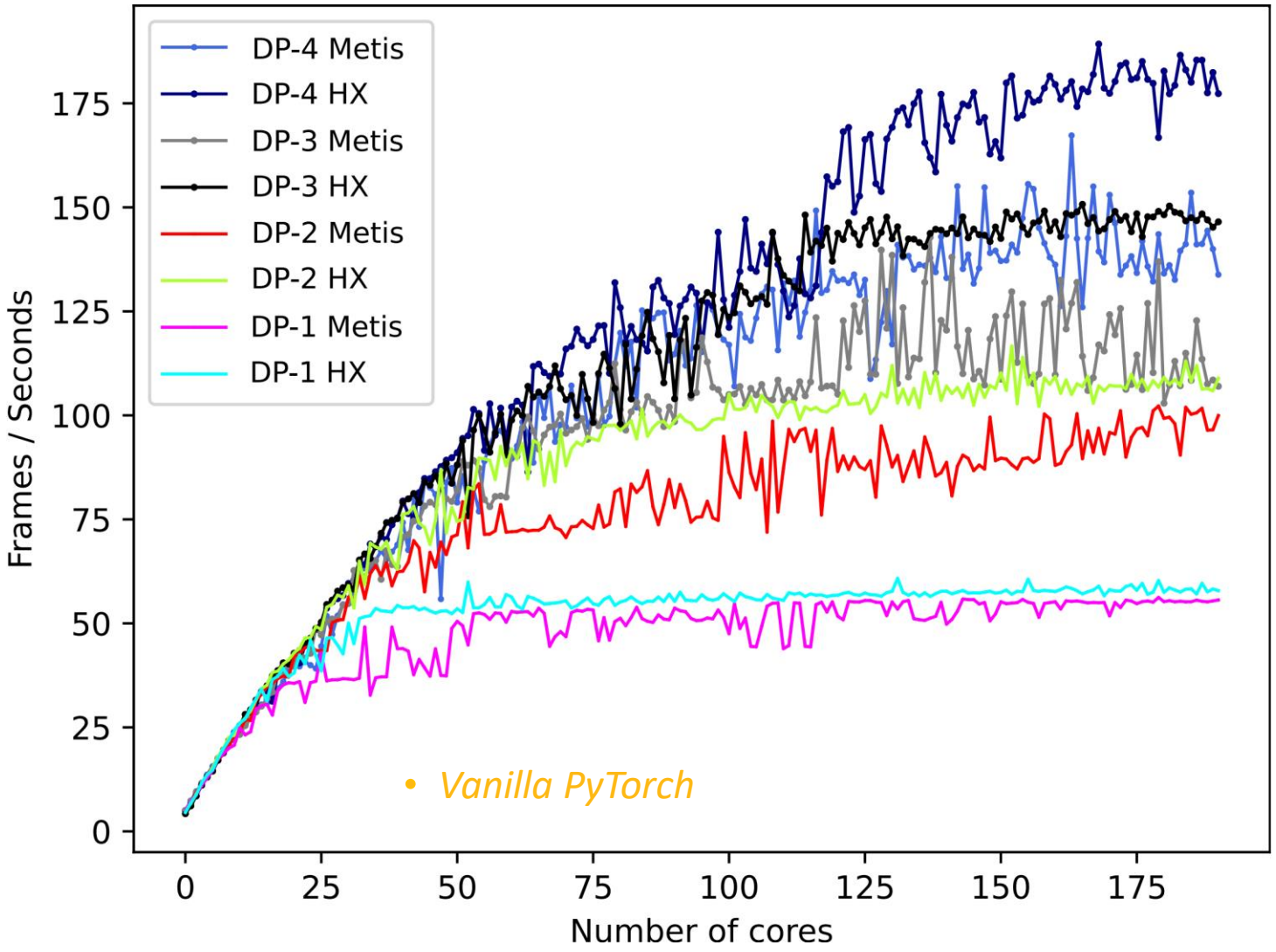
- Not all arguments are considered as actor inputs
  - All parameters and buffers are constant state variables, loaded on actor construction
  - Single action actor, the action calls the external ATen function with the arguments of the call_function
  - Output of the actor has the same name as the ‘call_function’ node’s name

```
actor convolution() Tensor arg2_1 ==> Tensor convolution :

    Tensor _arg0_1 = load_tensor_from_file("output/params_buffers_data/arg0_1.pt");
    Tensor _arg0_2 = load_tensor_from_file("output/params_buffers_data/arg0_2.pt");

    action [_arg2_1] ==> [_convolution]
    var
        Tensor _convolution = convolution(_arg2_1, _arg0_1, _arg0_2, [4, 4], [2, 2], [1, 1], false, [0, 0], 1)
    end
end
```

Expressing data-parallelism by splitting the batch size



Experimental results



- Test Platform
  - HiSilicon Kunpeng 920, ARM v8.2
  - 4 Sockets, 48-cores per socket
- PyTorch 2.0
  - Compiled natively on the platform
  - OpenBLAS as BLAS
- 8 Configurations
  - ResNet-50 from TorchVision
  - (B, 3, 224, 224) shape
  - Data parallelism from 1 to 4
  - Test implicit pipeline parallelism and data parallelism
  - Force OpenBLAS to use only one core
  - DP-1 has similar performance as vanilla PyTorch for a single request

Conclusion

- **Initial exploration of an actor runtime for executing ML Graphs**
- **ATS + Actor Machine + ART + PyTorch**
- **Firing conditions checking latency <<< operation execution latency**
- **Performance depends on a good partitioning tool/algorithm**
- **Use libTorch Ops with stream-based actor semantics**
- **Inter-op parallelism and implicit pipeline parallelism**

- **Future work**
  - Distributed execution
  - ML Training
  - Python bindings for representing stream-based actors
  - ART as a PyTorch backend

THANK YOU