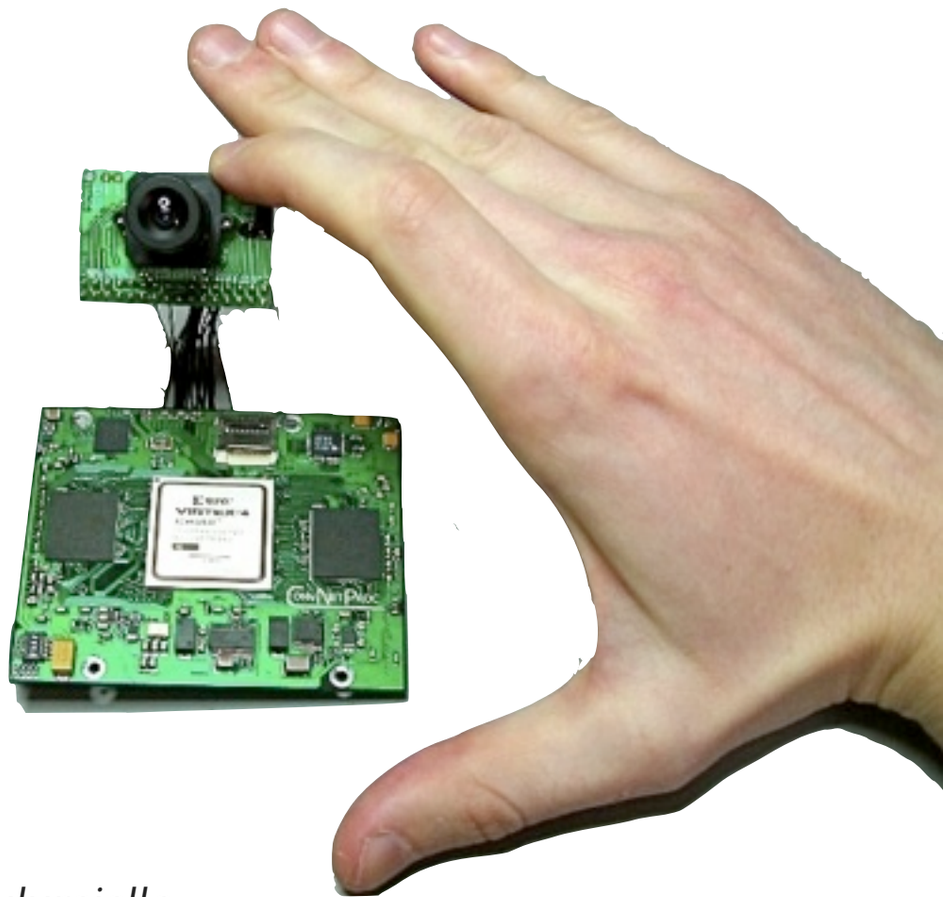
Bio-Inspired Vision Processor for Ultra-Fast Object Categorization

Clément Farabet

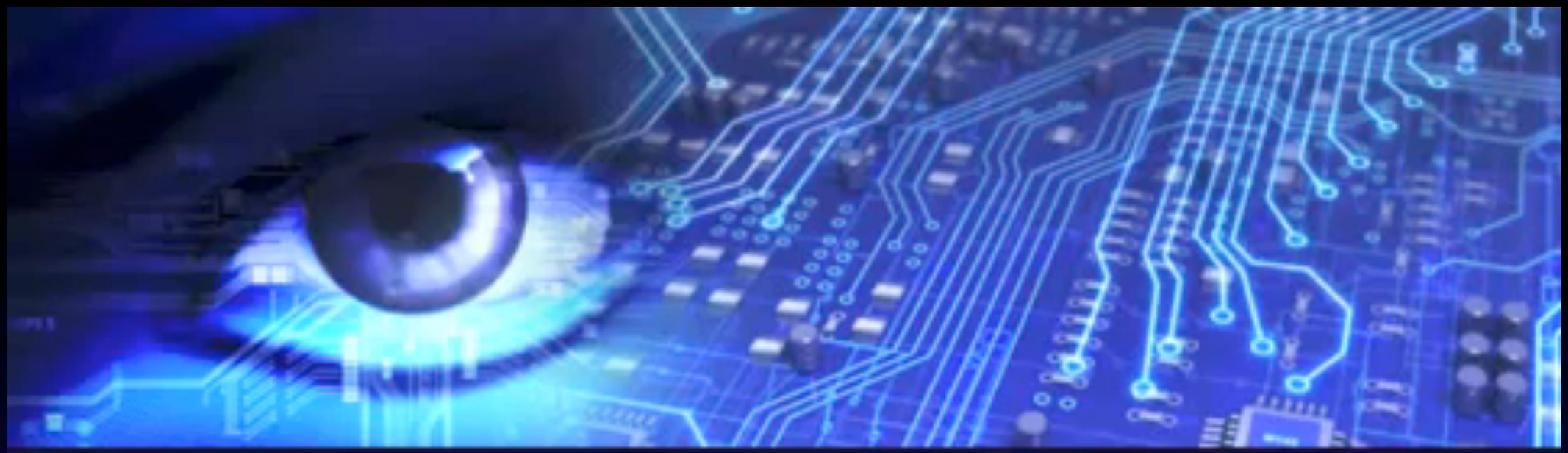


joint work with: Yann LeCun, Eugenio Culurciello,
Berin Martini, Polina Akselrod, Selcuk Talay, Benoit Corda

e-Lab

NEW YORK UNIVERSITY

Yale University



NeuFlow

Synthetic Vision System

e-Lab

Clement Farabet
Berin Martini
Polina Akselrod
Selcuk Talay

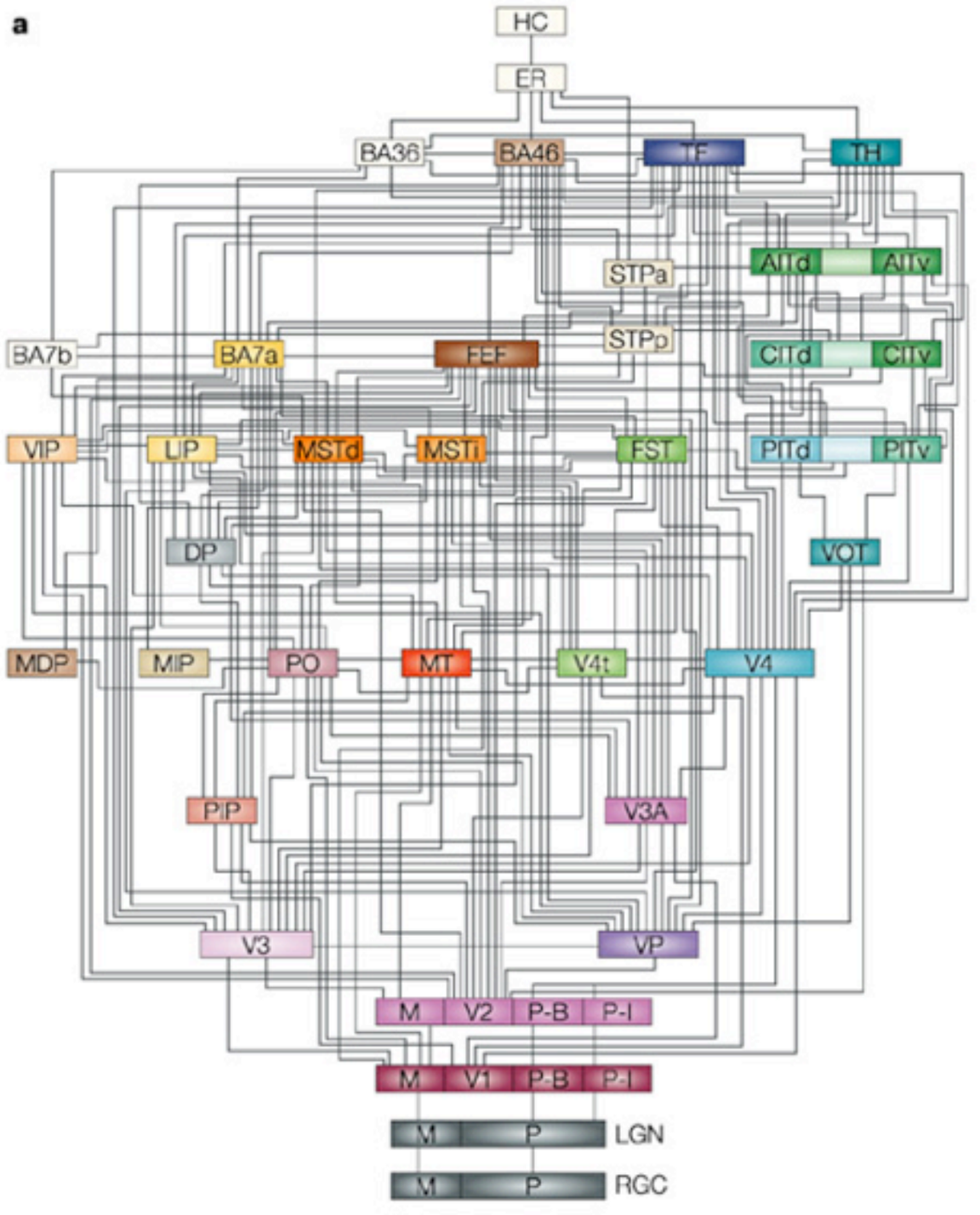
Computational & Biological Learning Laboratory

NEW YORK UNIVERSITY

Yann LeCun [NYU] + Eugenio Culurciello [Yale]

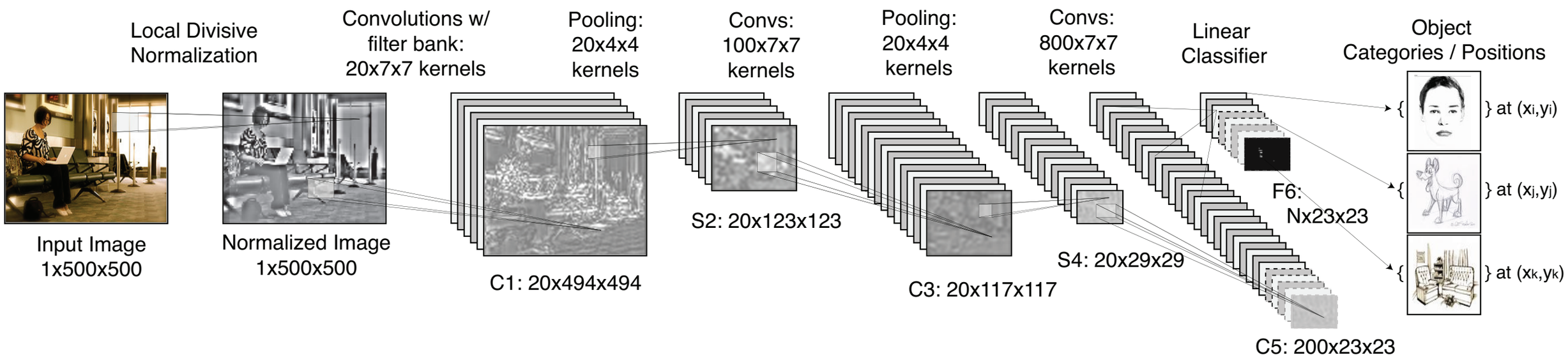
The Visual Cortex

neuroscientists have identified ~30 functional ‘modules’ in the cortex



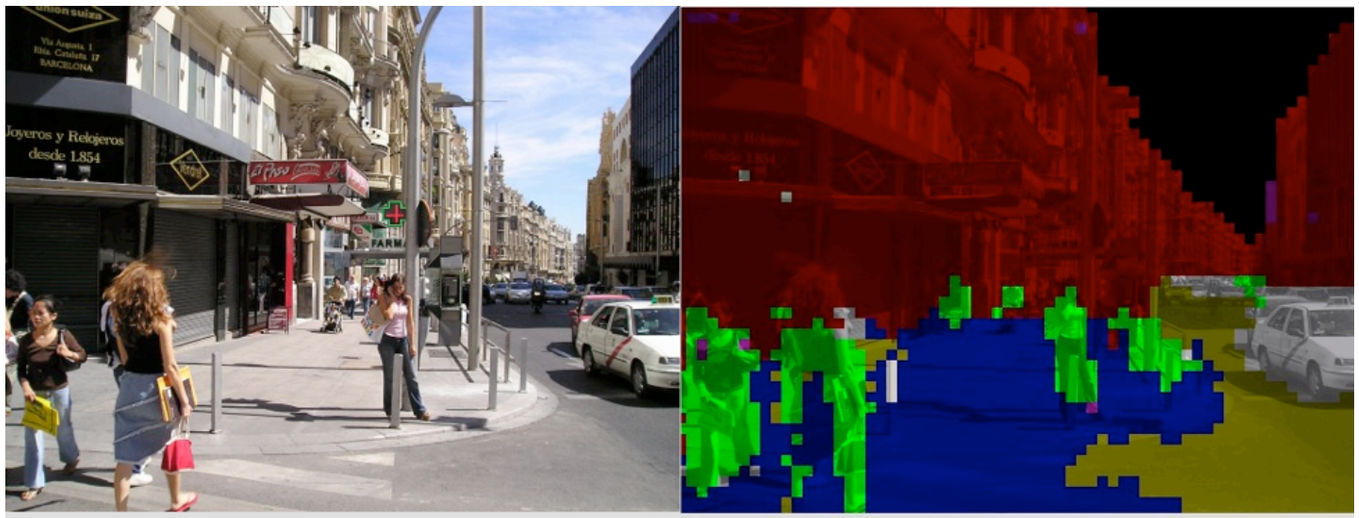
[Felleman & Van Essen, 1991]

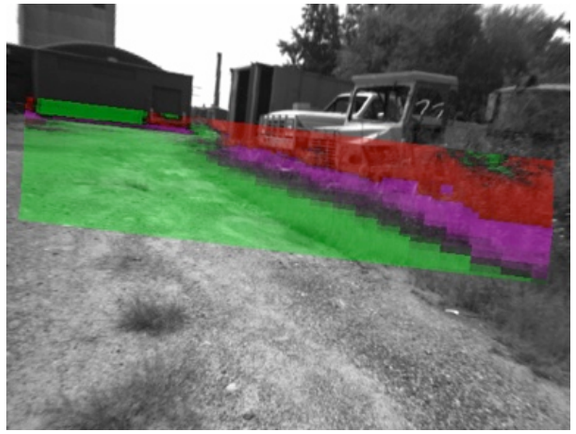
Convolutional Neural Nets

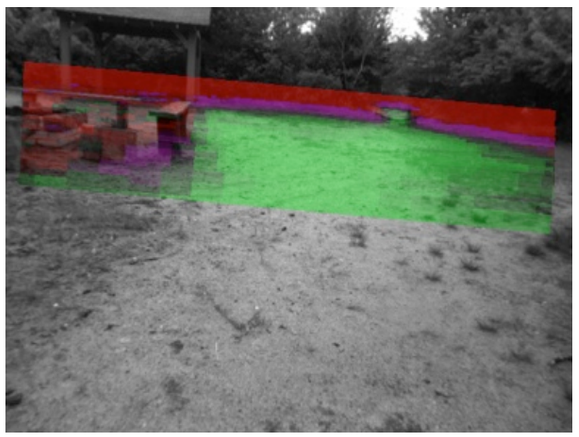


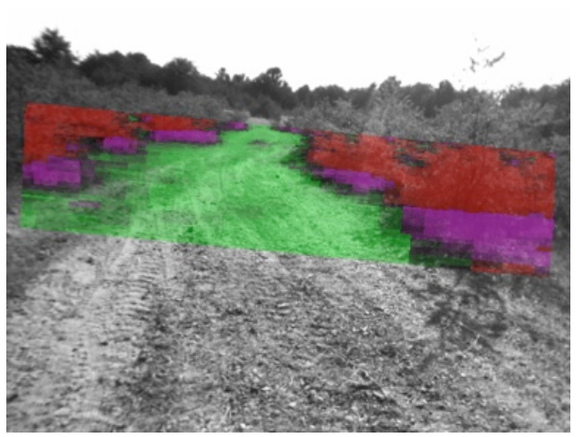
- they are hierarchical
- essentially feedforward
- each level is a transform that extracts or combines some features

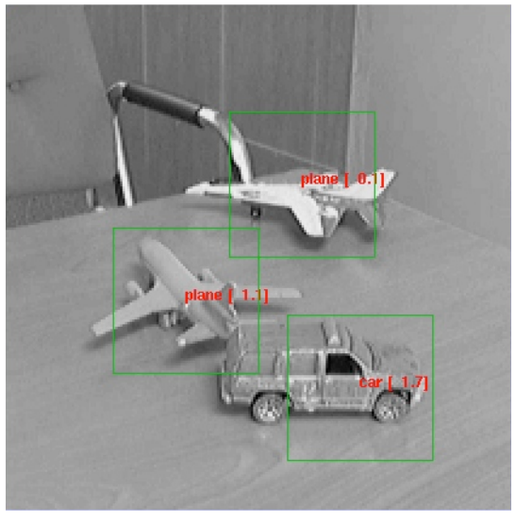
Convolutional Neural Nets

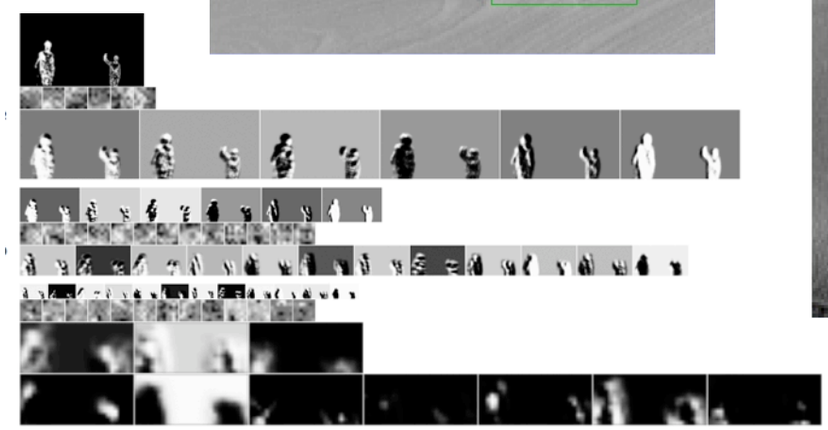


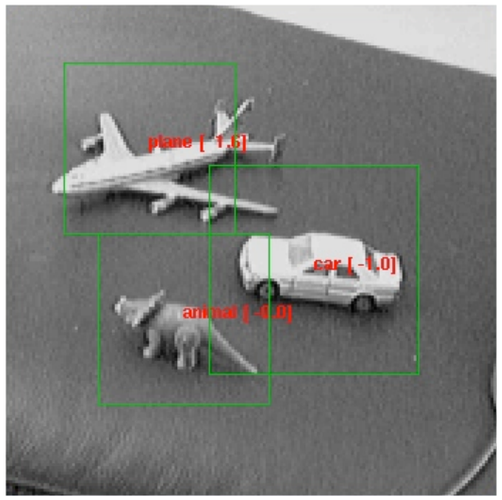




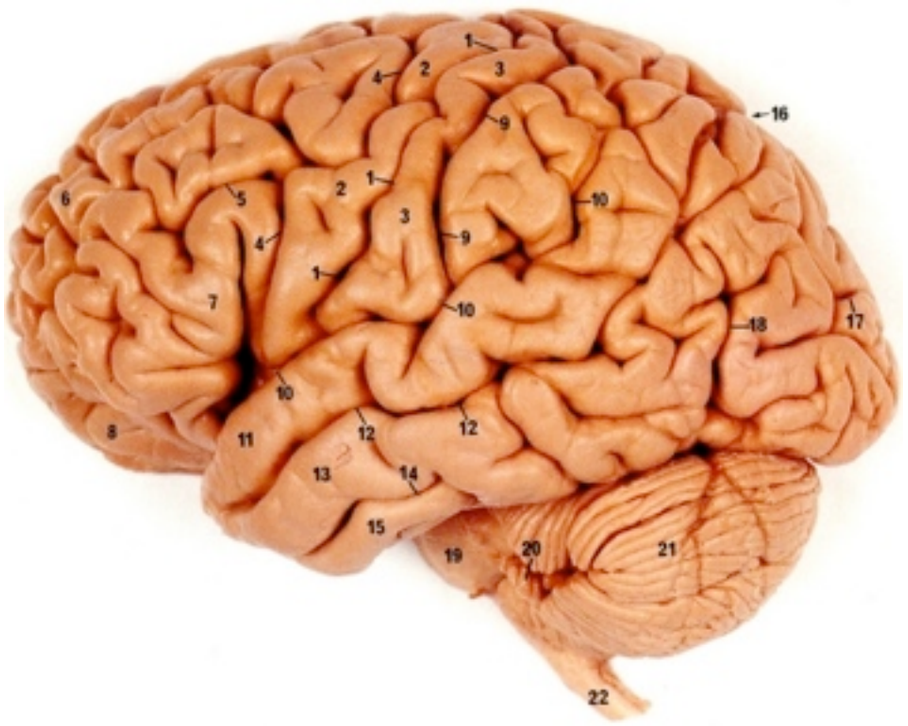








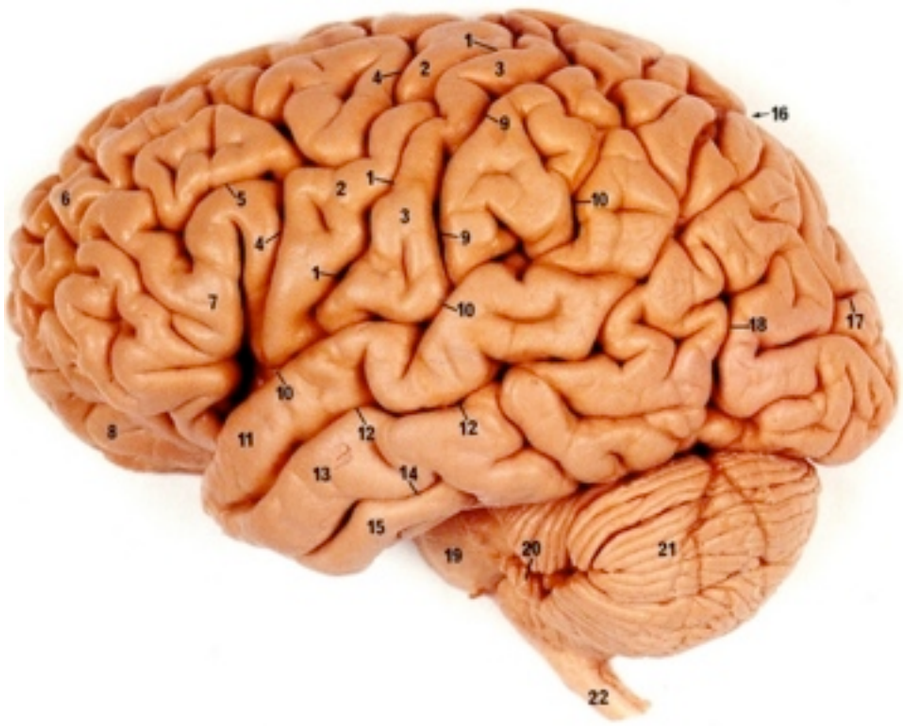
Dataflow Computing ?



the software written for the brain executes on several billion BPUs*, each connected to several thousands other BPUs

* Brain Processing Unit, more commonly known as neuron

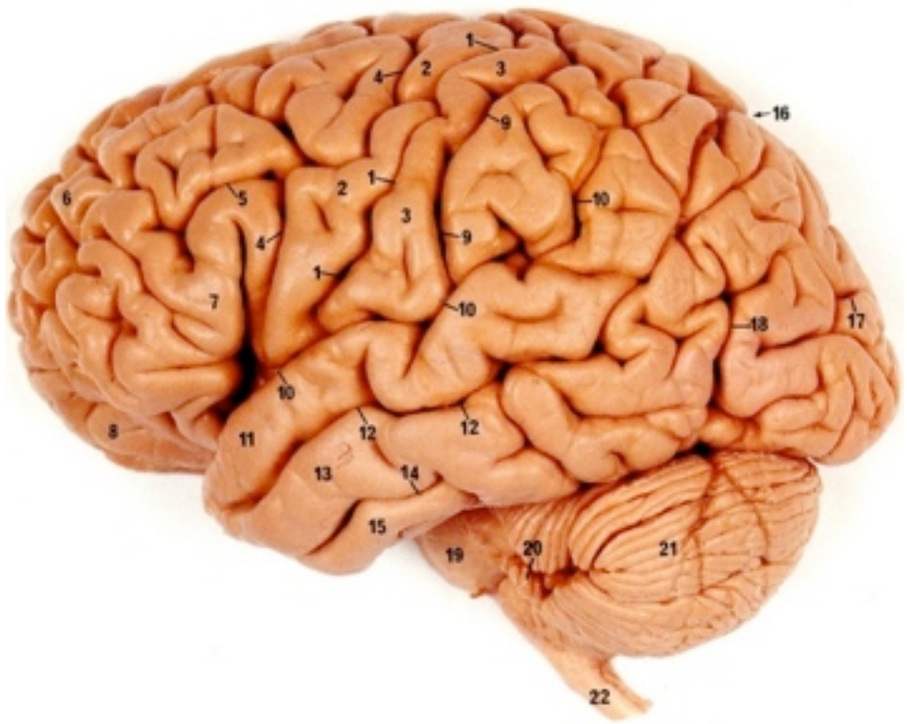
Dataflow Computing ?



the entire brain fabric is doing useful computation: if a synapse is assumed to perform a MAC* operation, then the brain computes ~20 petaOP/sec**

* Multiply and Accumulate = 2 OPs ** 20 billion neurons, ~1000 connections per neuron

Dataflow Computing ?



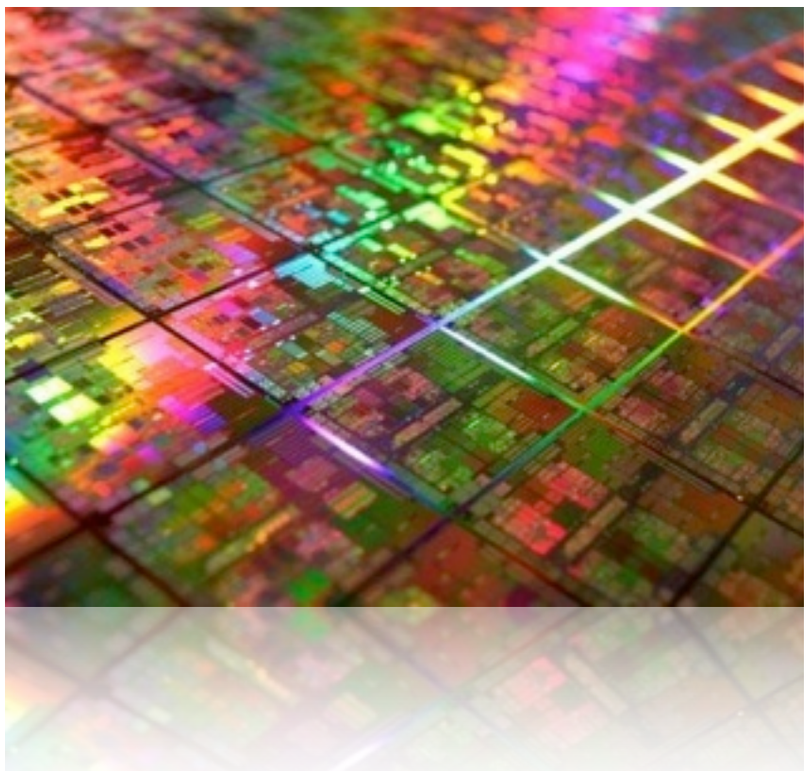
Brain (God)

-

>20POP/s 20W

-

1e6GOP/sec/W



Tesla GPU (nVidia)

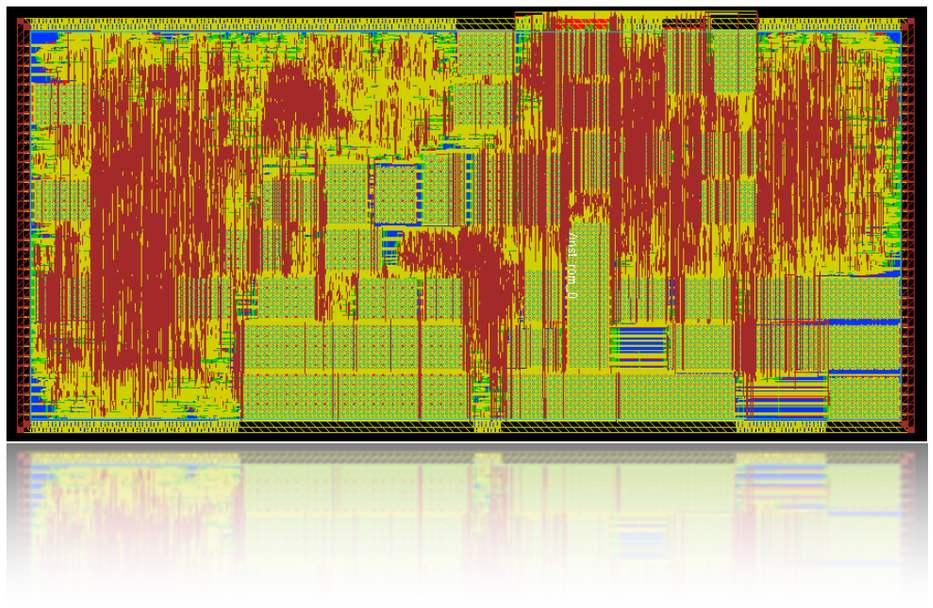
-

1TOP/s 200W

-

5GOP/sec/W

Dataflow Computing !



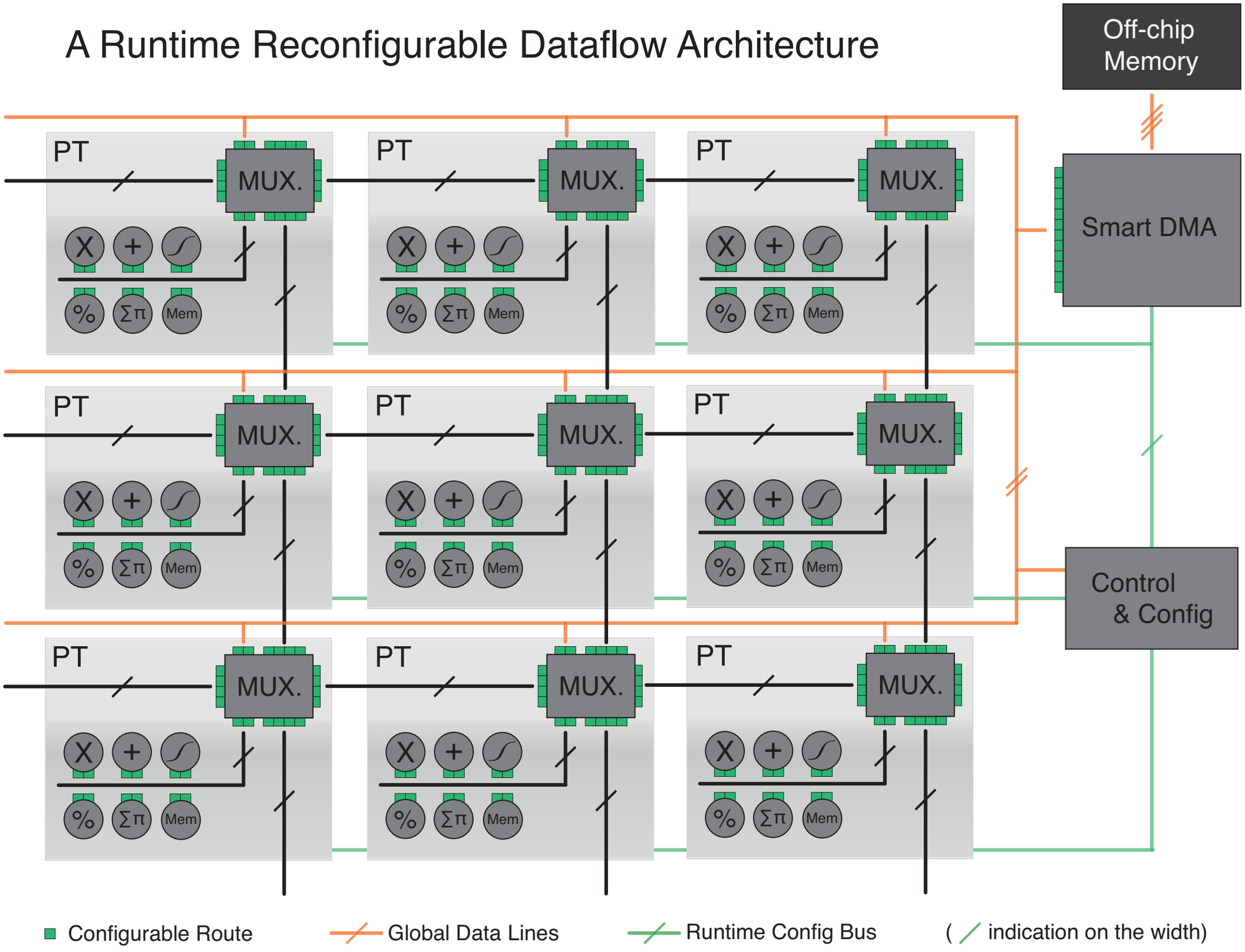
~1e3GOP/sec/W

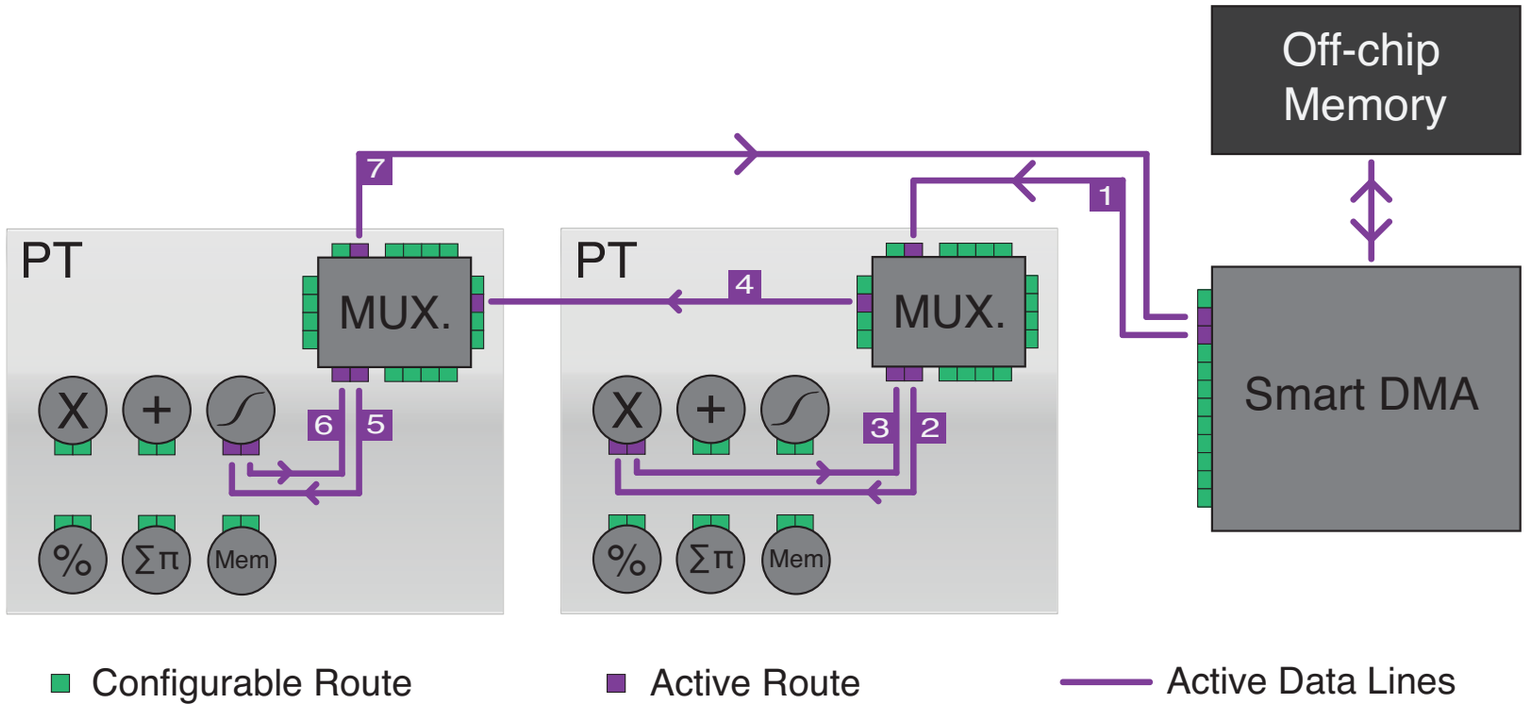
NeuFlow:
A Dataflow Computer

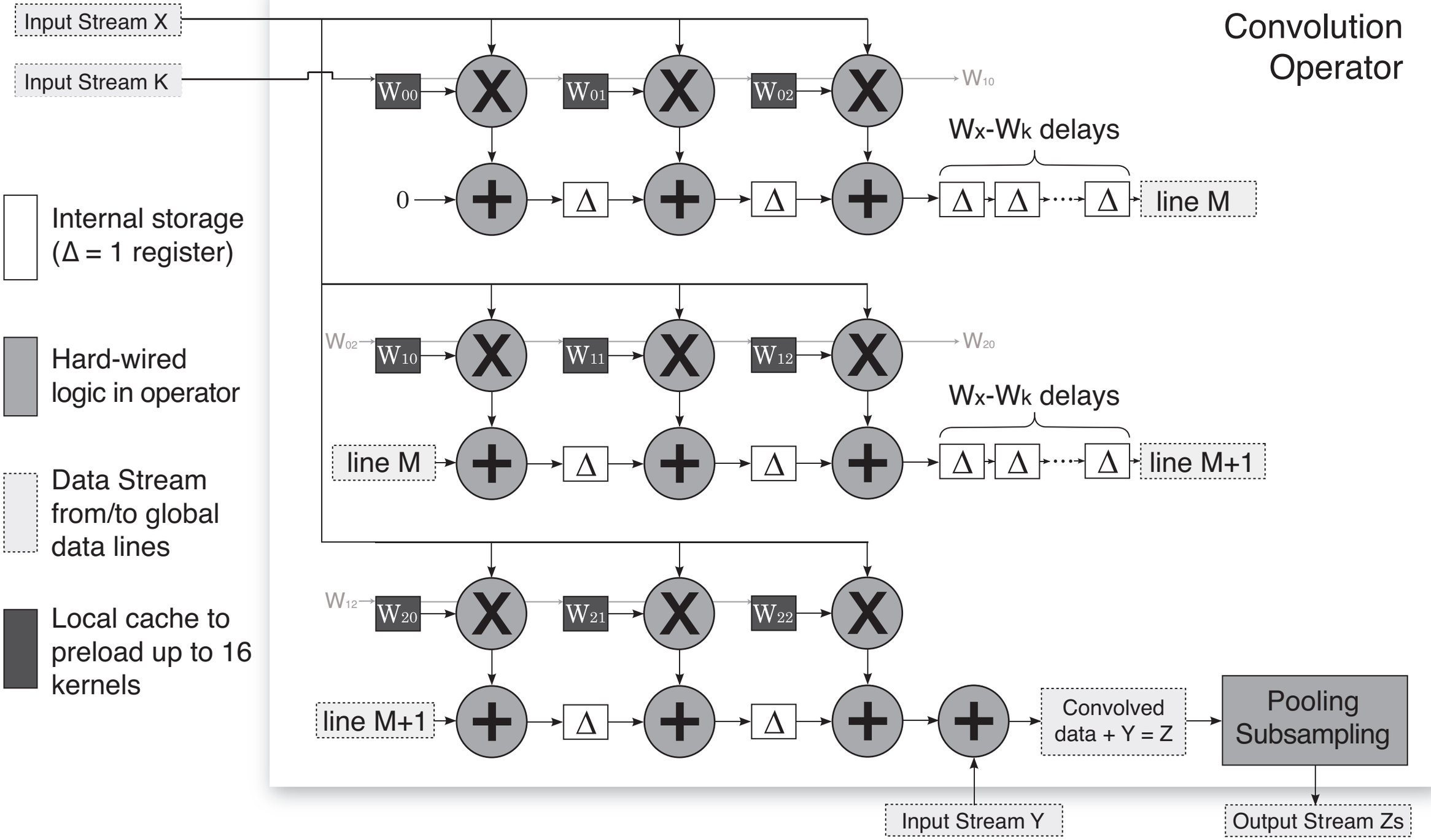
Play Movie

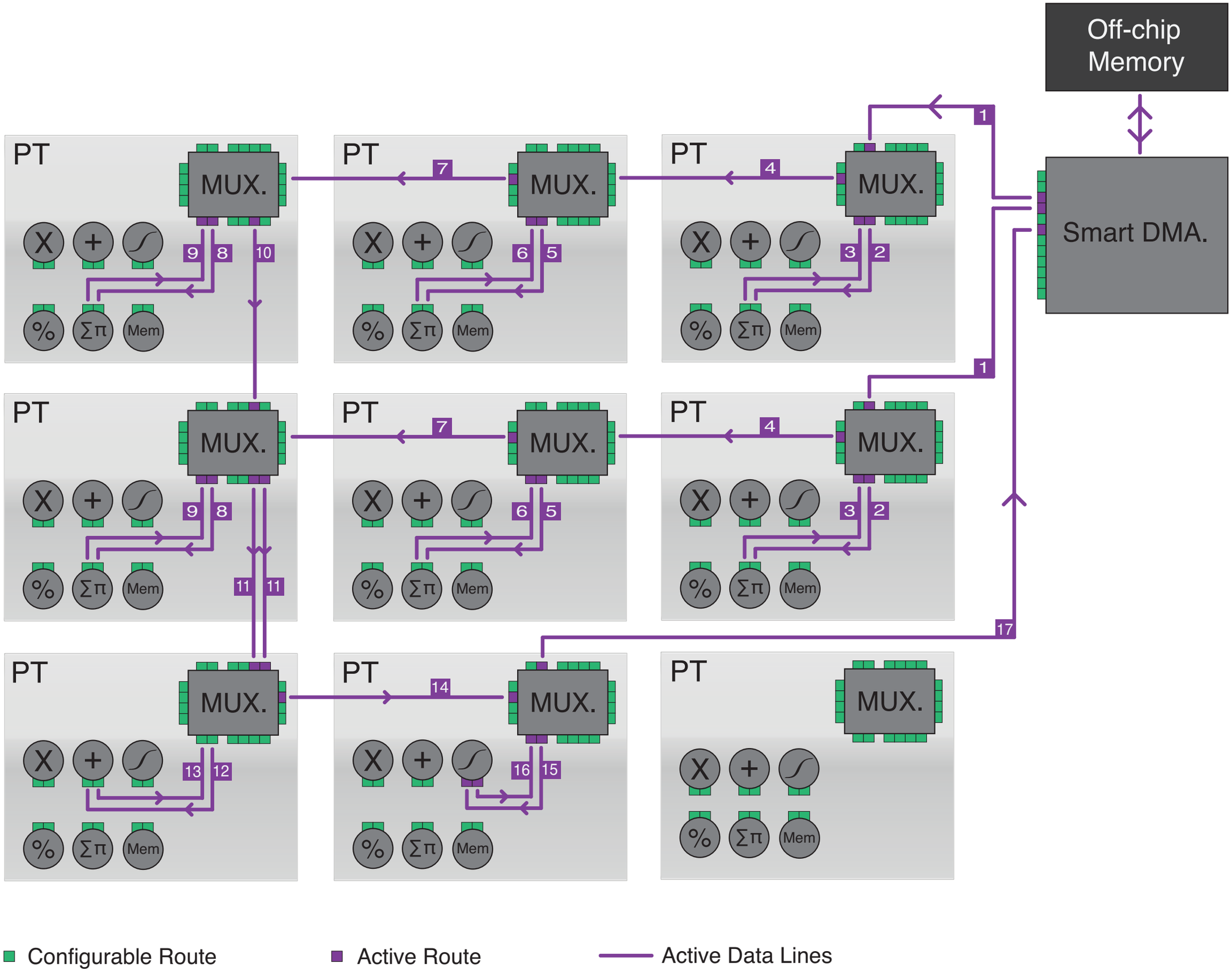
- processing units are kept simple/small with maximum throughput
- these units are replicated over a large grid, locally connected
- flow control exists only at a very coarse level, and is not distributed over the grid
- data drives computations, ‘for’ loops don’t exist

A Runtime Reconfigurable Dataflow Architecture




Off-chip
Memory
Smart DMA
PT
MUX.
X
+
6
5
%
∑π
Mem
PT
MUX.
4
X
+
3
2
%
∑π
Mem
7
1
Configurable Route
Active Route
Active Data Lines


Input Stream X
Input Stream K
Internal storage
(Δ = 1 register)
Hard-wired
logic in operator
Data Stream
from/to global
data lines
Local cache to
preload up to 16
kernels
Convolution
Operator
W00
W01
W02
W10
0
Δ
Wx-Wk delays
line M
W02
W10
W11
W12
W20
line M
line M+1
W12
W20
W21
W22
line M+1
Convolved
data + Y = Z
Pooling
Subsampling
Input Stream Y
Output Stream Zs


Off-chip Memory
Smart DMA.
PT
MUX.
X
+
%
Σπ
Mem
1
4
7
9
8
10
6
5
3
2
11
11
14
13
12
16
15
17
Configurable Route
Active Route
Active Data Lines

Instruction Set

- compiling this Lua code:

```
for i=1,100 do
    a[i] = b[i]
end
```

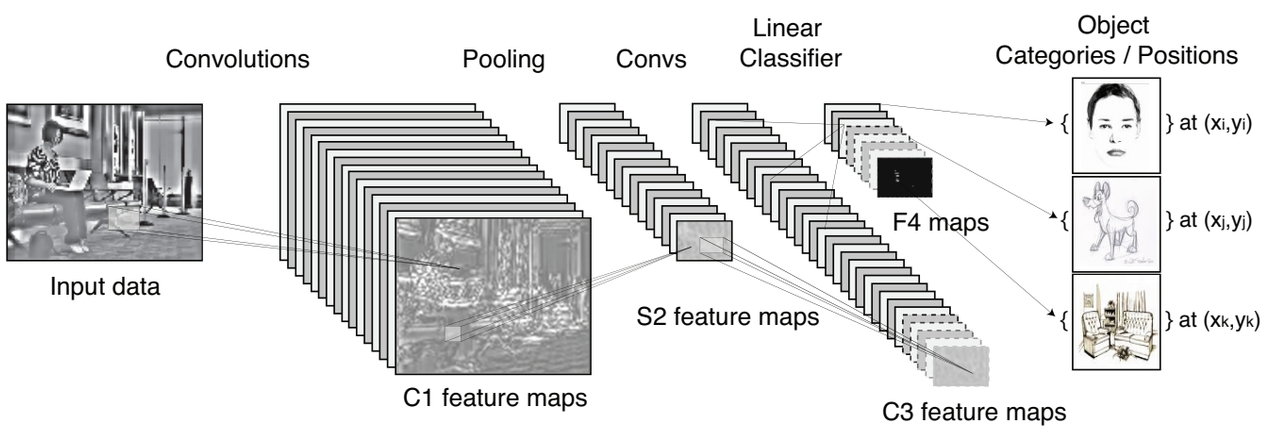
- for a classical flow-control machine:

```
SETI     REGD  12340
SETI     REGE  3455
READ     REGD  REGB
WRITE    REGE  REGB
INCR     REGD
INCR     REGE
COMPI    REGD  12440  REGC
JUMPREL -5     REGC
```

- for our flow-based CPU:

```
SETI     REGD  12340
SETI     REGE  3455
STREAM   REGD  REGE  100
```

LuaFlow:
A Dataflow Compiler



a specialized compiler that converts flow descriptions of algorithms to sequences of grid reconfiguration

LuaFlow:
A Dataflow Compiler/API

- the compiler is an API written in Lua, which allows full development in this easy-to-learn, fast prototyping language

- it relies on Torch (an efficient N-dim array library*) to allow algorithms to be described as functional transforms, or flow-graphs of computations

- the API is similar to CUDA in some ways, the developer first has to elaborate the code, e.g. build the code that will run on the platform, and then write the code for the host computer

- the compiler iterates over the input flow and tries to merge as many nodes as possible, to minimize grid reconfigurations

* similar to NumPy

A luaFlow Program (1/2)

- initializing neuFLow:

```
neuFlow = NeuFlow{mode='runtime'}
```

- describing a neural net:

```
input_host = torch.Tensor(100,100)
net = nn.Sequential()
net:add(nn.SpatialConvolution(1,16,9,9))
net:add(nn.Tanh())
net:add(nn.SpatialLinear(16,4))
```

- elaborating the code for neuFlow:

```
neuFlow:beginLoop('main')
    input_nf = neuFlow:copyFromHost(input_host)
    output_nf = neuFlow:compile(net, input_nf)
    output_host = neuFlow:copyToHost(output_nf)
neuFlow:endLoop('main')
```

A luaFlow Program (2/2)

- loading the bytecode on neuFlow:

```
neuFlow:loadBytecode()
-- at this point, neuFlow executes its new code
```

- now simply describe the host code:

```
while true do
    input_host = camera:getFrame()
    neuFlow:copyToDev(input_host)
    neuFlow:copyFromDev(output_host)
    result = soft_classifier:forward(output_host)
end
```

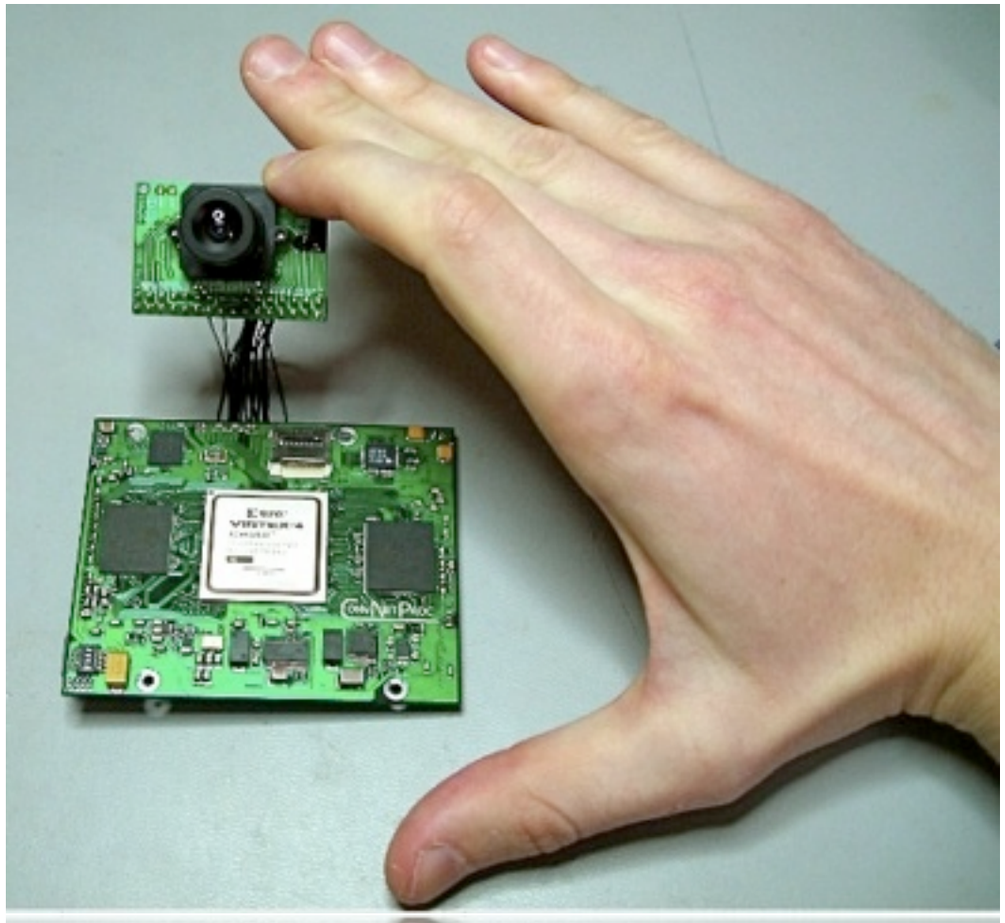
- at this point the code is running in a loop, neuFlow is computing the neural net, while the host computes a simple linear classifier on the results

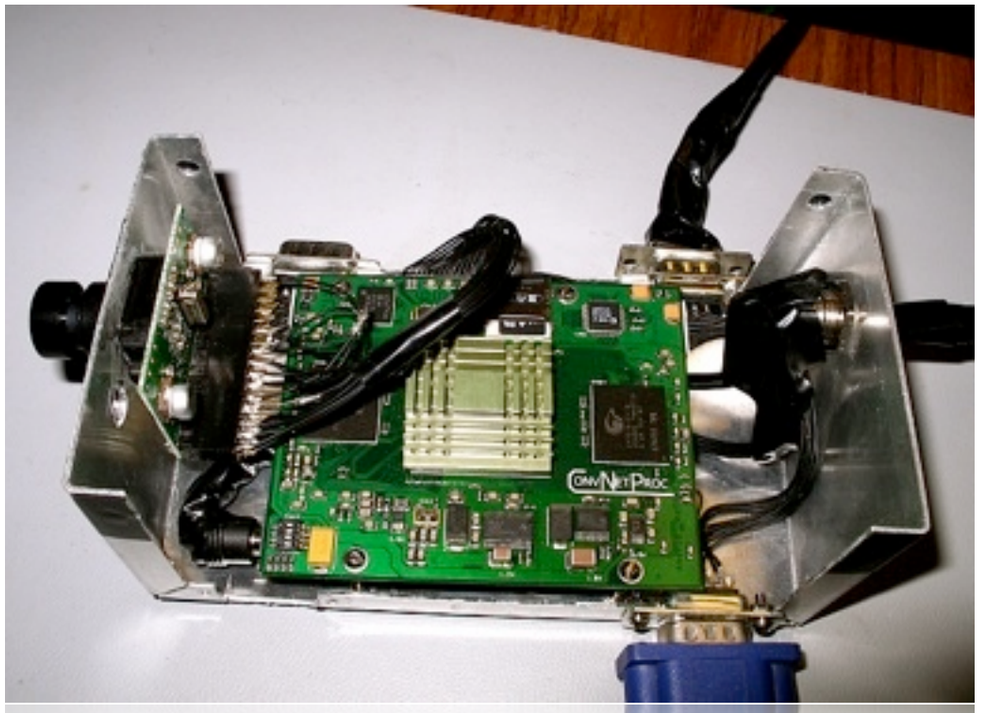
PROFILING*

	Intel 2Core	neuFlow Virtex4	neuFlow Virtex 6	nVidia GT335m	neuFlow65 nm	nVidia S1070
Peak GOP/sec	10?	40	160	182	1000	1000
Actual GOP/sec	1.1	37	147	54	928	290
FPS	1.4	46	182	67	1152	360
Power (W)	30	10	15	30	2	220
Embed? (GOP/s/W)	0.03667	3.7	9.8	1.8	464	1.31818

* computing a 16x10x10 filter bank over a 4x500x500 input image

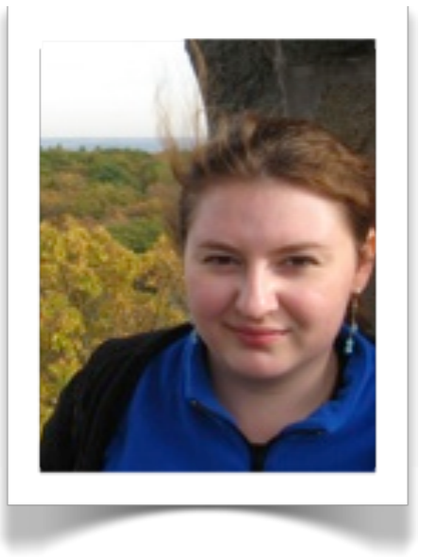
neuFlow fits Anywhere





a home-made PCB that includes a Virtex4 and some quite large bandwidth to/from QDR memories

Thank You.



Polina Akselrod

Neu Flow

Yann LeCun

Selçuk Talay

Berin Martini

Benoit Corda

Eugenio Culurciello