embedded VISION SUMMIT

Efficient Implementation of Convolutional Neural Networks using OpenCL on FPGAs

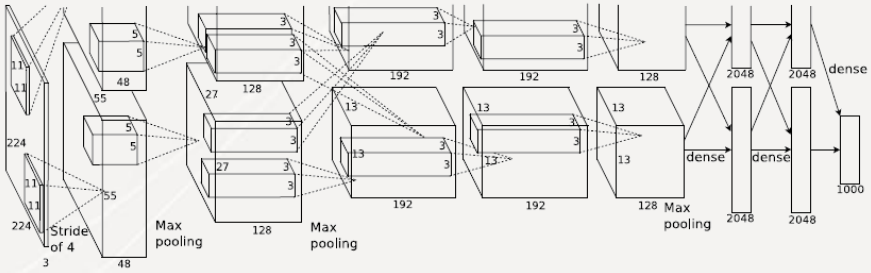
Dr. Deshanand Singh, Director of Software Engineering

ALTERA®

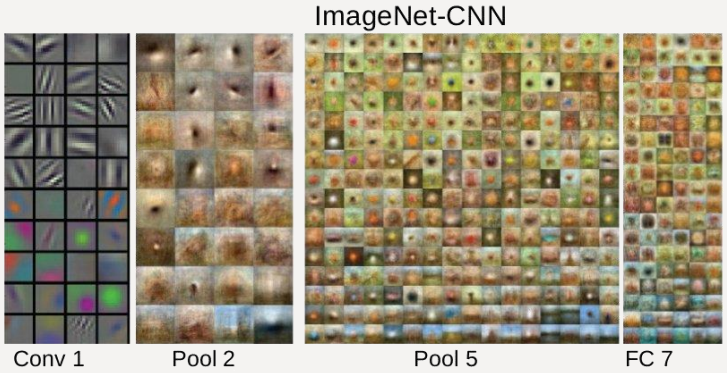
12 May 2015

Convolutional Neural Network (CNN)

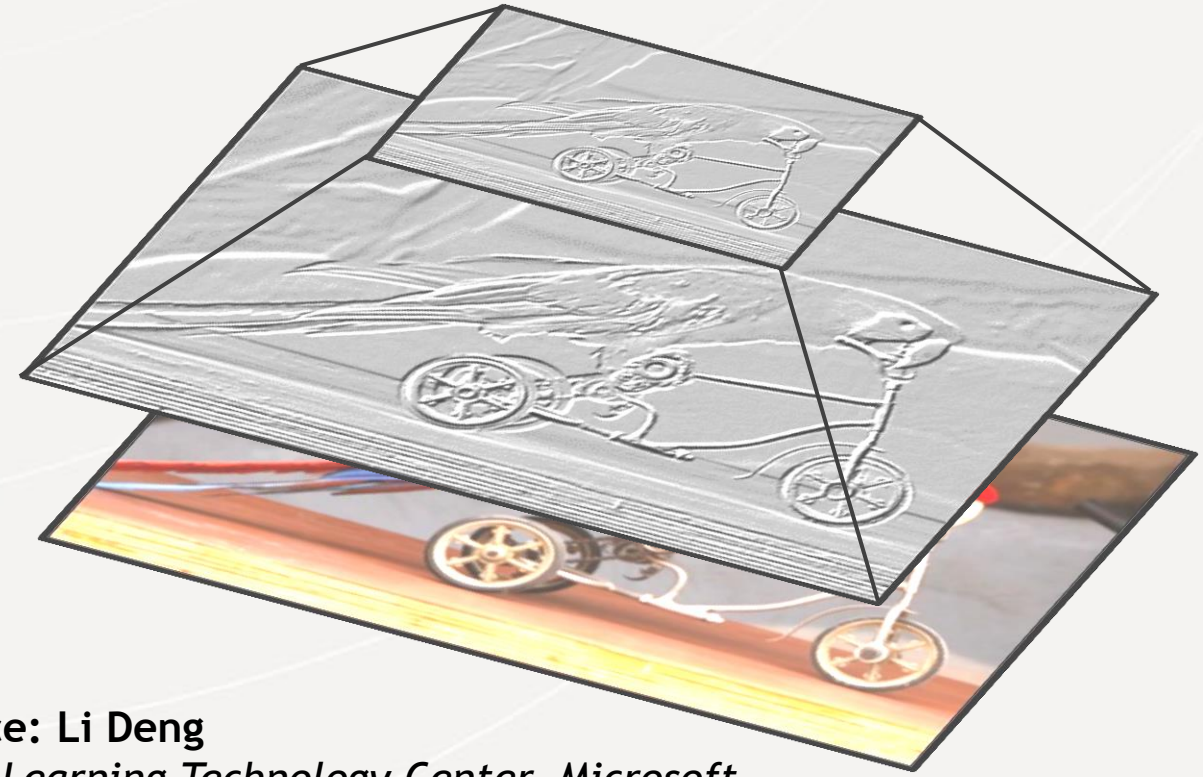
- **Convolutional Neural Network**
  - Feed forward network Inspired by biological processes
  - Composed of different layers
  - More layers increases accuracy
- **Convolutional Layer**
  - Extract different features from input
    - Low level features
      - e.g. edges, lines corners
- **Pooling Layer**
  - Reduce variance
    - Invariant to small translations
  - Max or average value
    - Feature over region in the image
- **Applications**
  - Classification & Detection, Image recognition/tagging



Prof. Hinton's CNN Algorithm



Basic Building Block of the CNN



Pooling

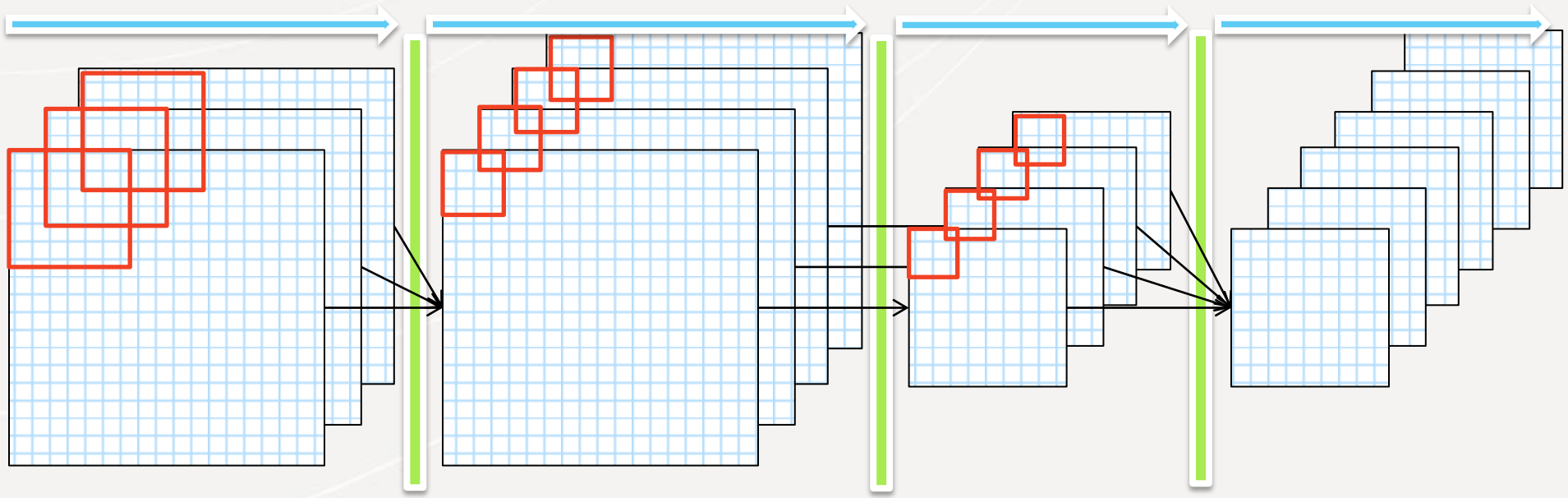
Convolution

Image

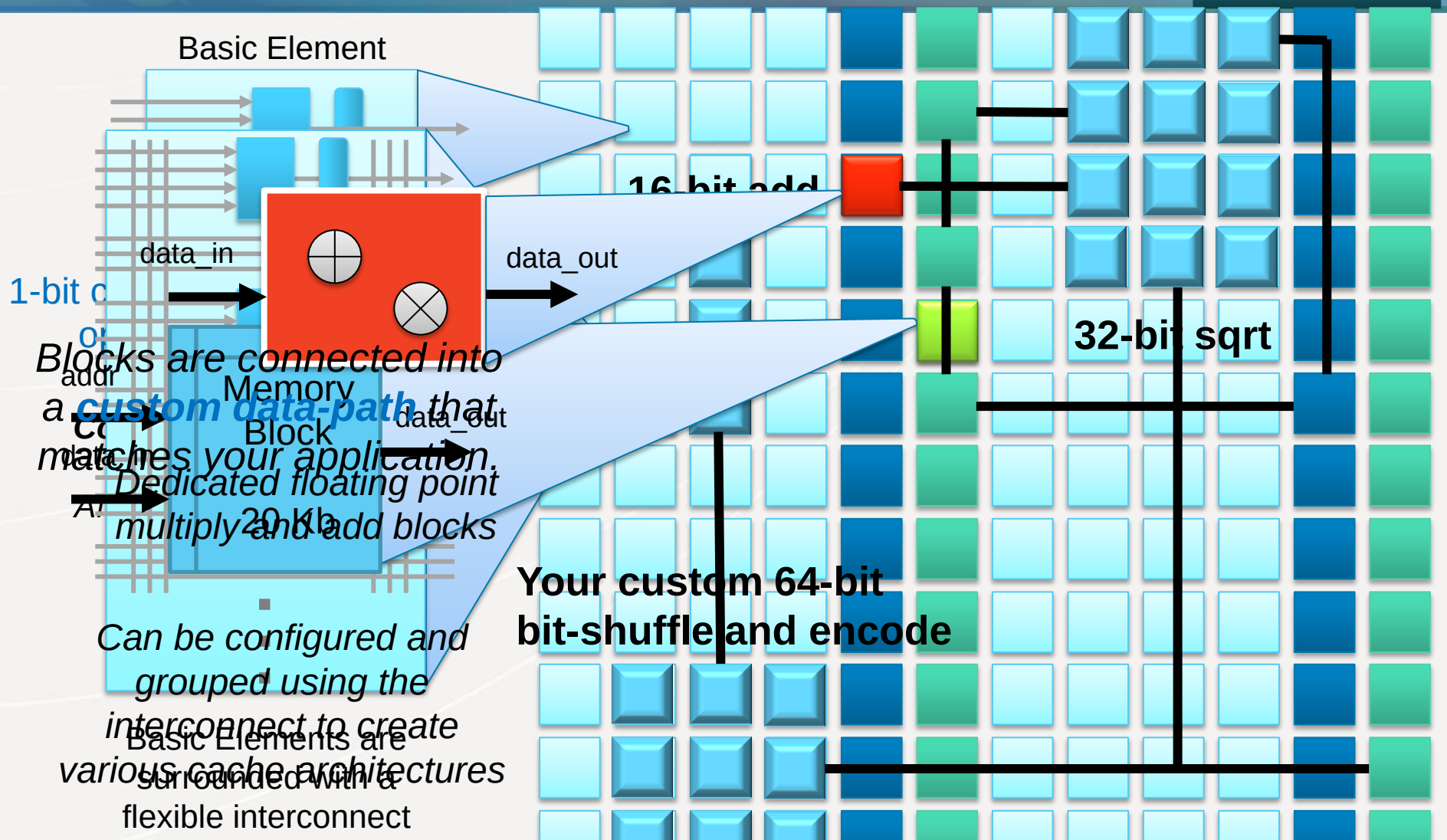
Source: Li Deng
Deep Learning Technology Center, Microsoft Research, Redmond, WA. USA
A Tutorial at International Workshop on Mathematical Issues in Information Sciences

Key Observation: Pipelining

- Dataflow through the CNN can proceed in pipelined fashion
  - No need to wait until the entire execution is complete
  - Can start a new set of data going to stage 1 as soon as the stage complete its execution



FPGAs : Compute fabrics that support pipelining



*Blocks are connected into a **custom data-path** that matches your application.*

Dedicated floating point multiply and add blocks

Can be configured and grouped using the interconnect to create various cache architectures

Programming FPGAs: SDK for OpenCL

OpenCL Host Program

OpenCL Kernels

Users write software

Altera OpenCL host library

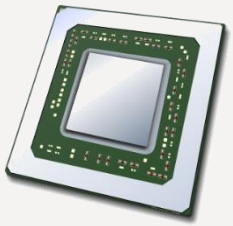
C compiler

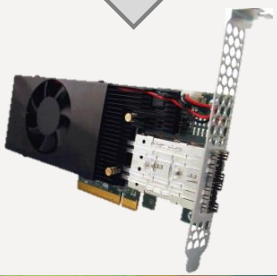
Altera Kernel Compiler

FPGA specific compilation target

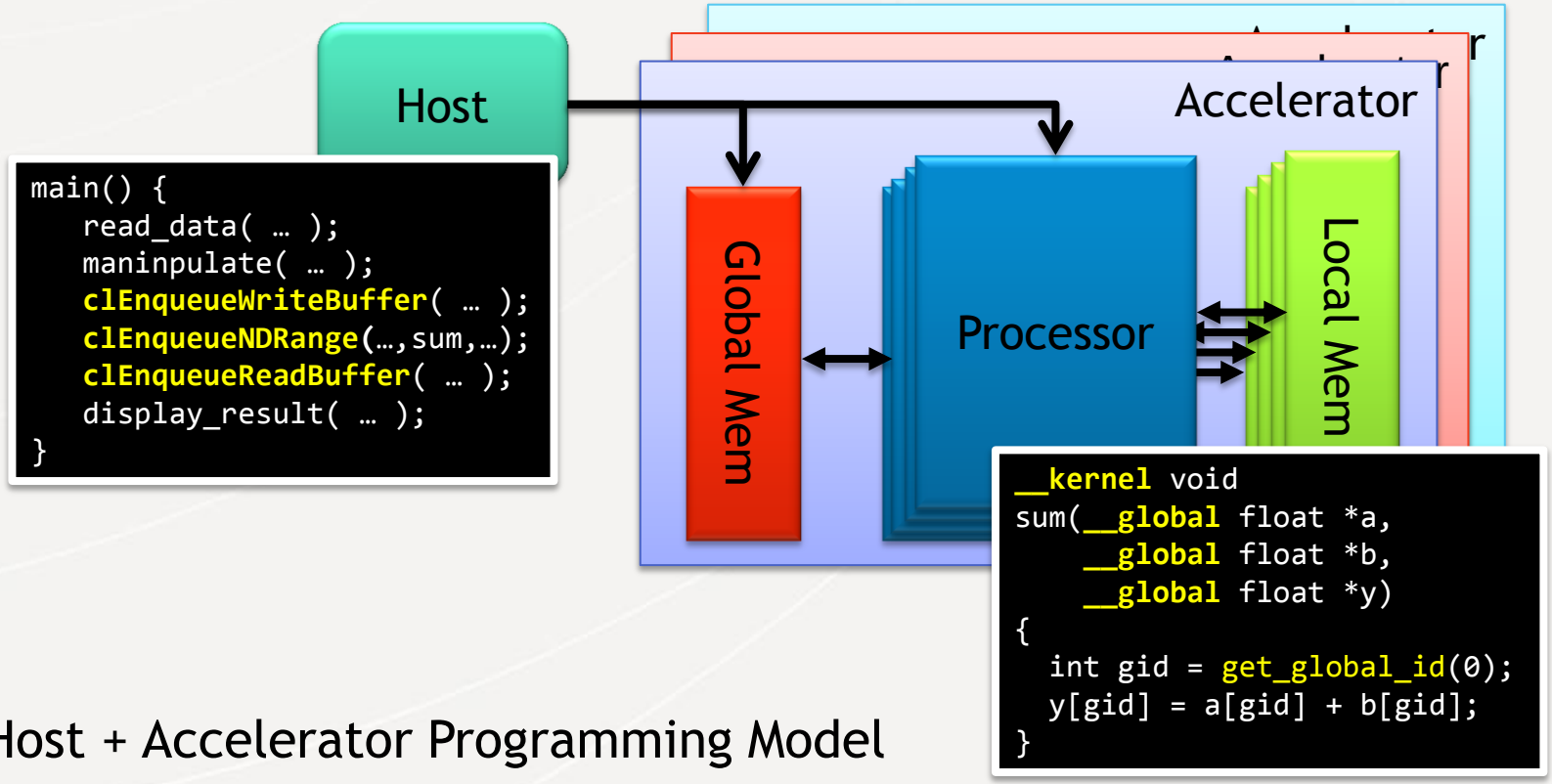
Host Binary

Device Binary





OpenCL Programming Model

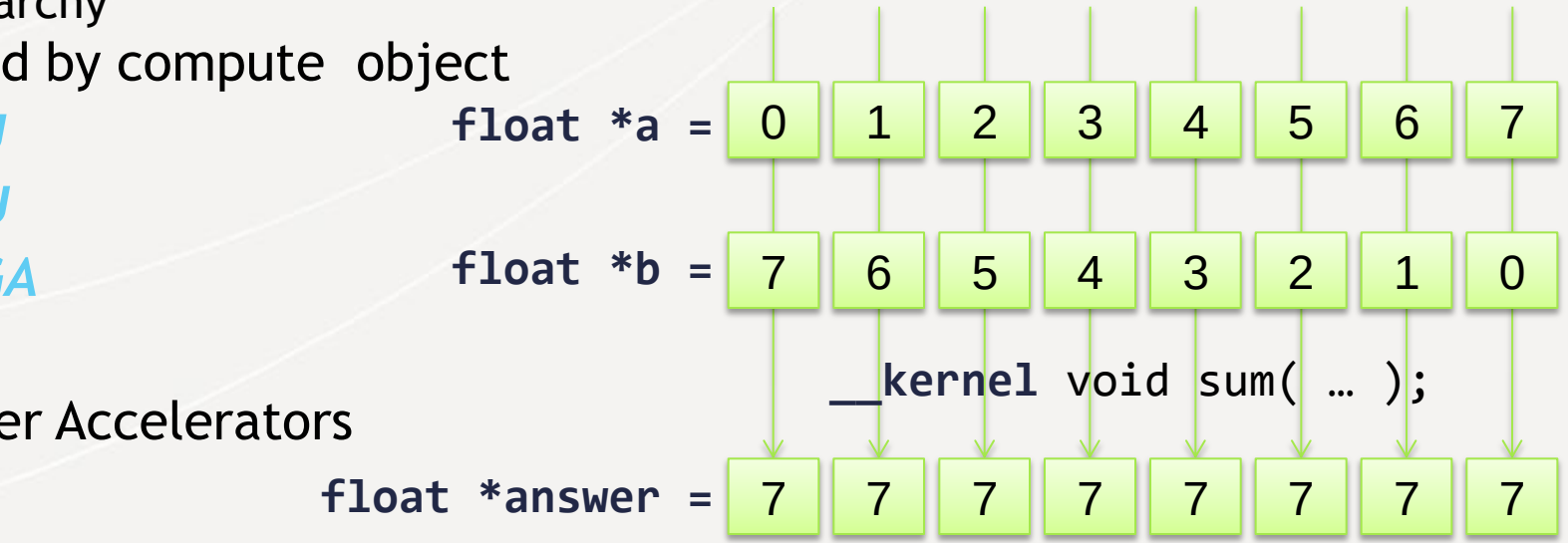


- Host + Accelerator Programming Model
- Sequential Host program on microprocessor
- *Function offload onto a highly parallel accelerator device*

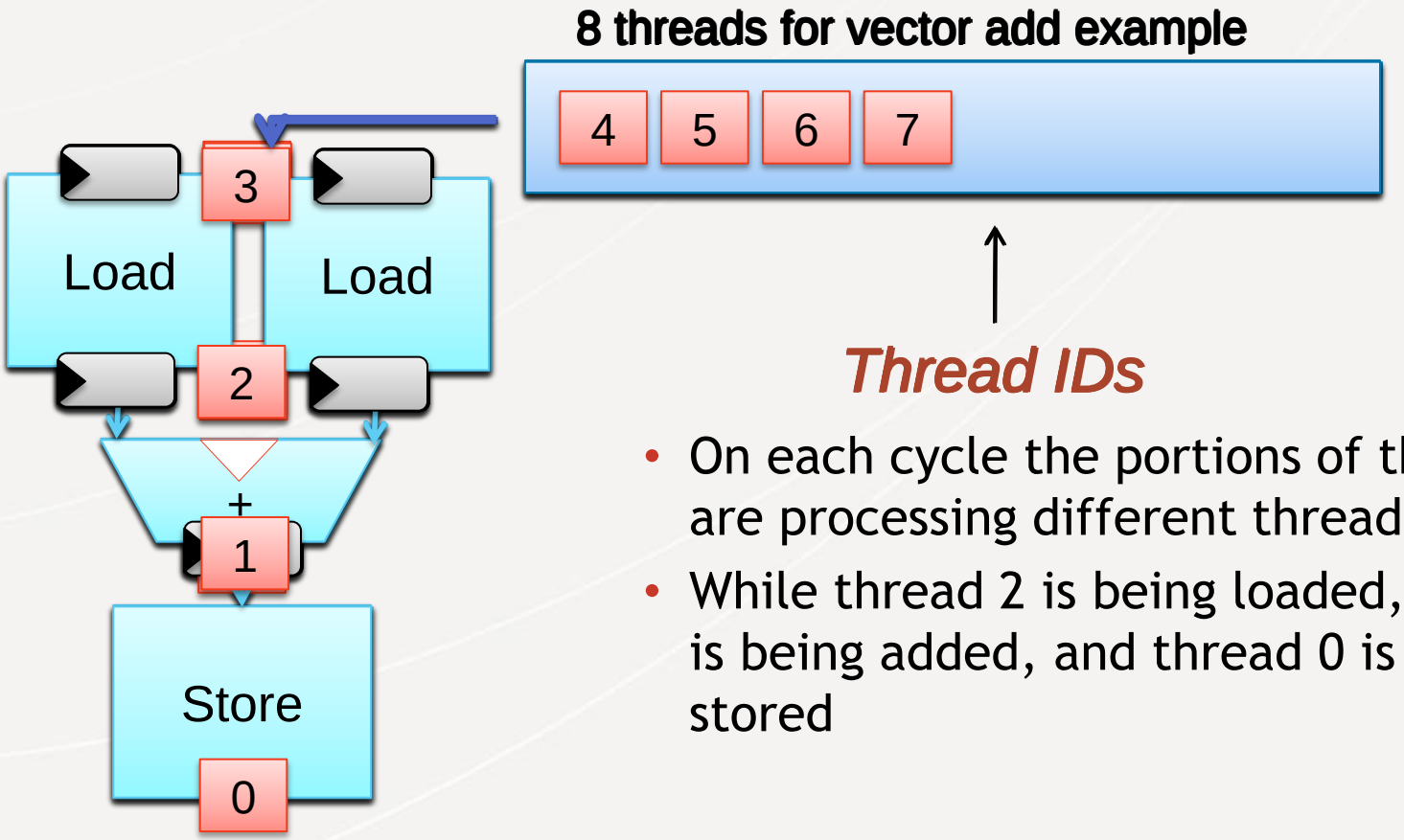
OpenCL Kernels

- Data-parallel function
  - Defines many parallel threads of execution
  - Each thread has an identifier specified by “**get_global_id**”
  - Contains keyword extensions to specify parallelism and memory hierarchy
- Executed by compute object
  - *CPU*
  - *GPU*
  - *FPGA*
  - *DSP*
  - Other Accelerators

```
__kernel void
sum(__global const float *a,
__global const float *b,
__global float *answer)
{
int xid = get_global_id(0);
answer[xid] = a[xid] + b[xid];
}
```

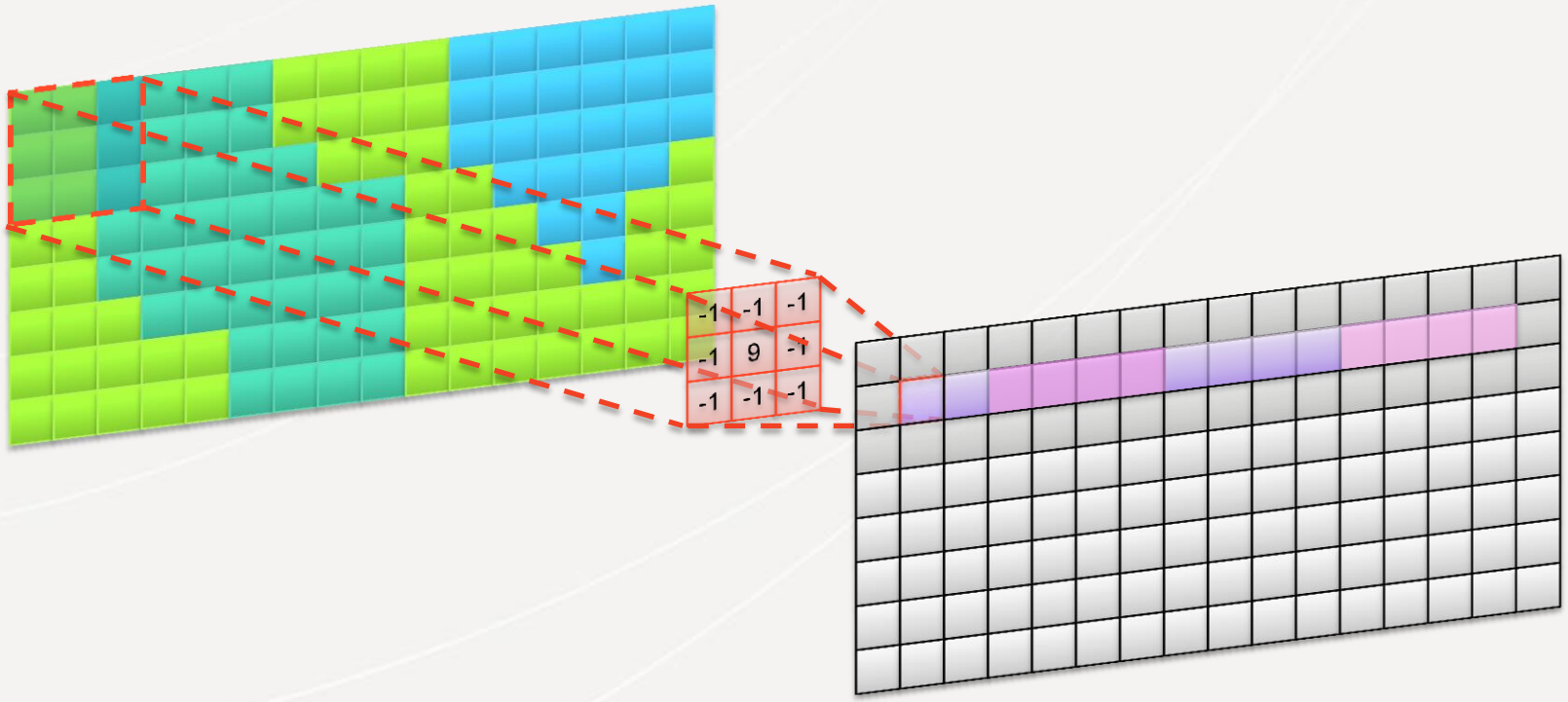


Dataflow / Pipeline Architecture from OpenCL



- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

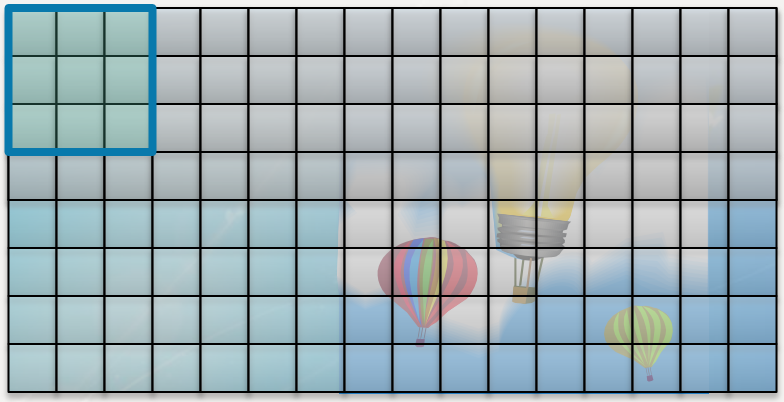
Convolutions: Our Basic Building Block

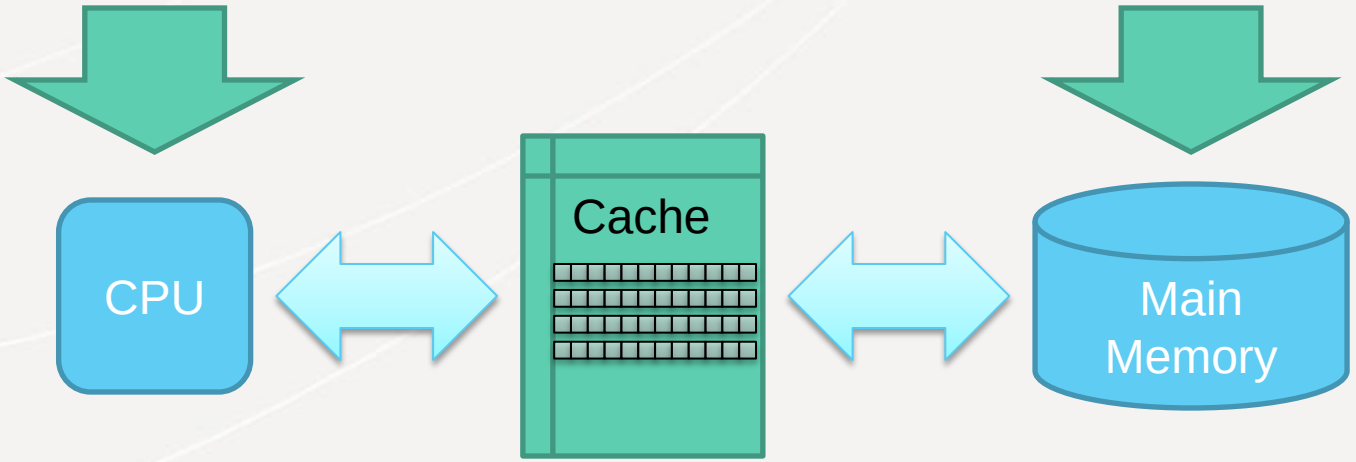


$$I_{new}[x][y] = \sum_{x'=-1}^{1} \sum_{y'=-1}^{1} I_{old}[x+x'][y+y'] \times F[x'][y']$$

Processor (CPU/GPU) Implementation

```
for(int y=1; y<height-1; ++y) {
  for(int x=1; x<width-1; ++x) {
    for(int y2=-1; y2<1; ++y2) {
      for(int x2=-1; x2<1; ++x2) {
        i2[y][x] += i[y+y2][x+x2]
                  * filter[y2][x2];
```

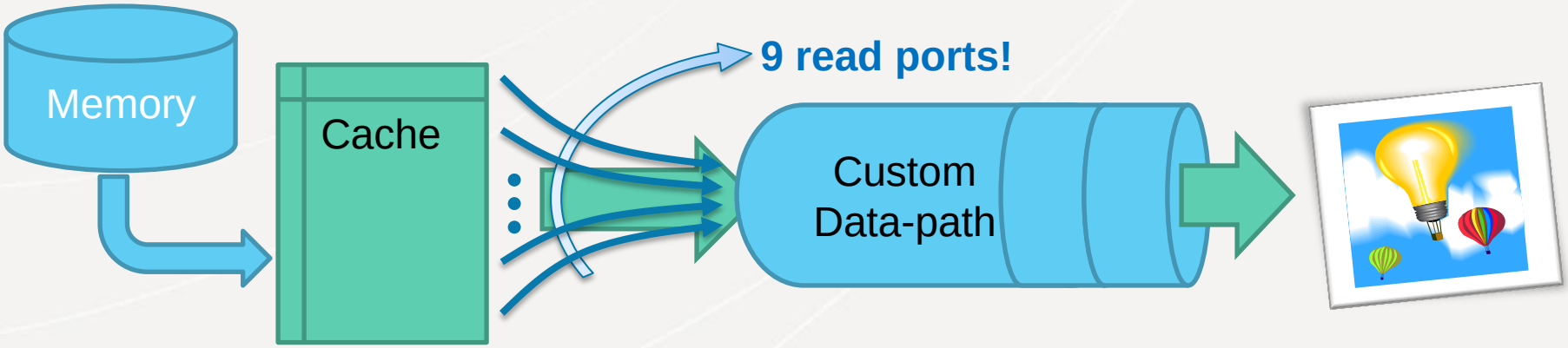




- A cache can hide poor memory access patterns

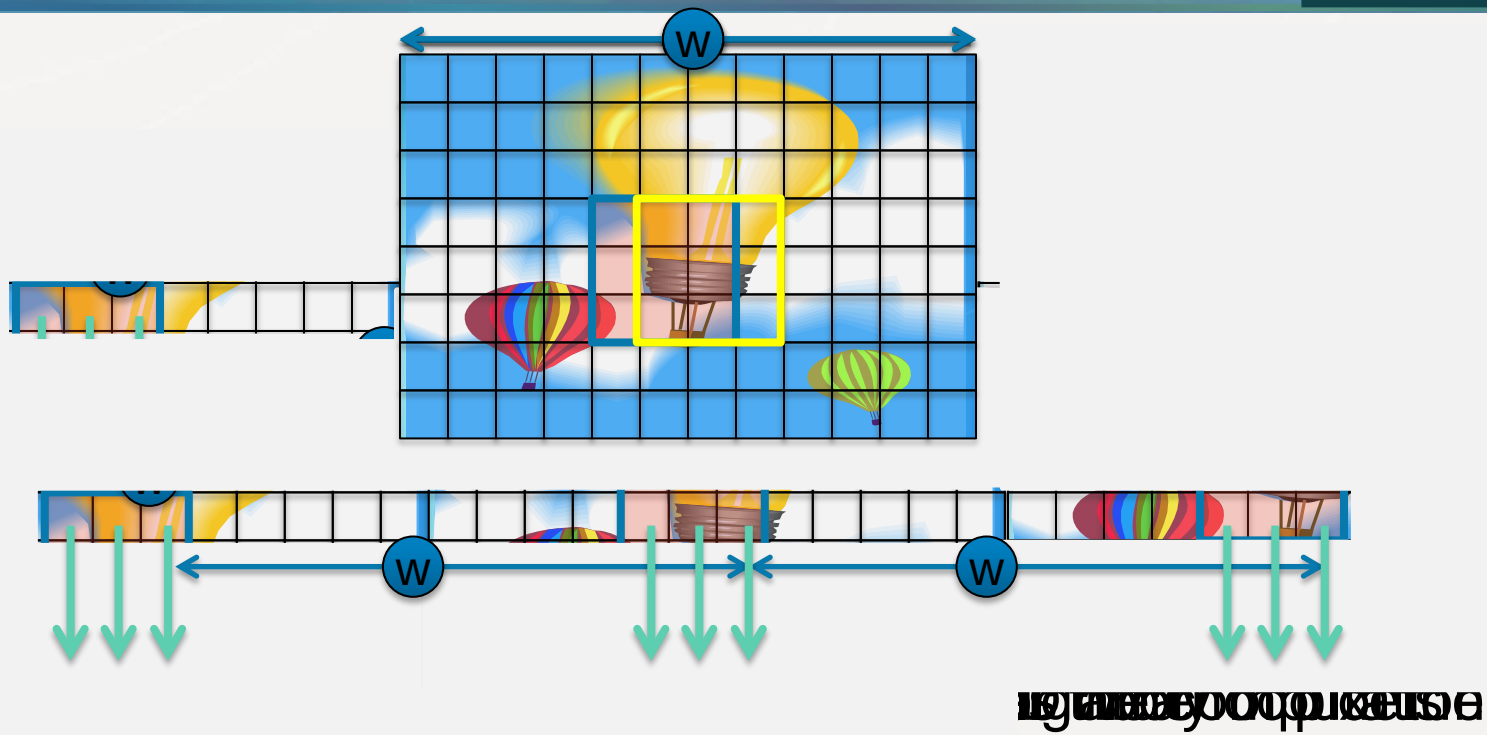
FPGA Implementation ?

- Example Performance Point: 1 pixel per cycle
- Cache requirements: 9 reads + 1 write per cycle



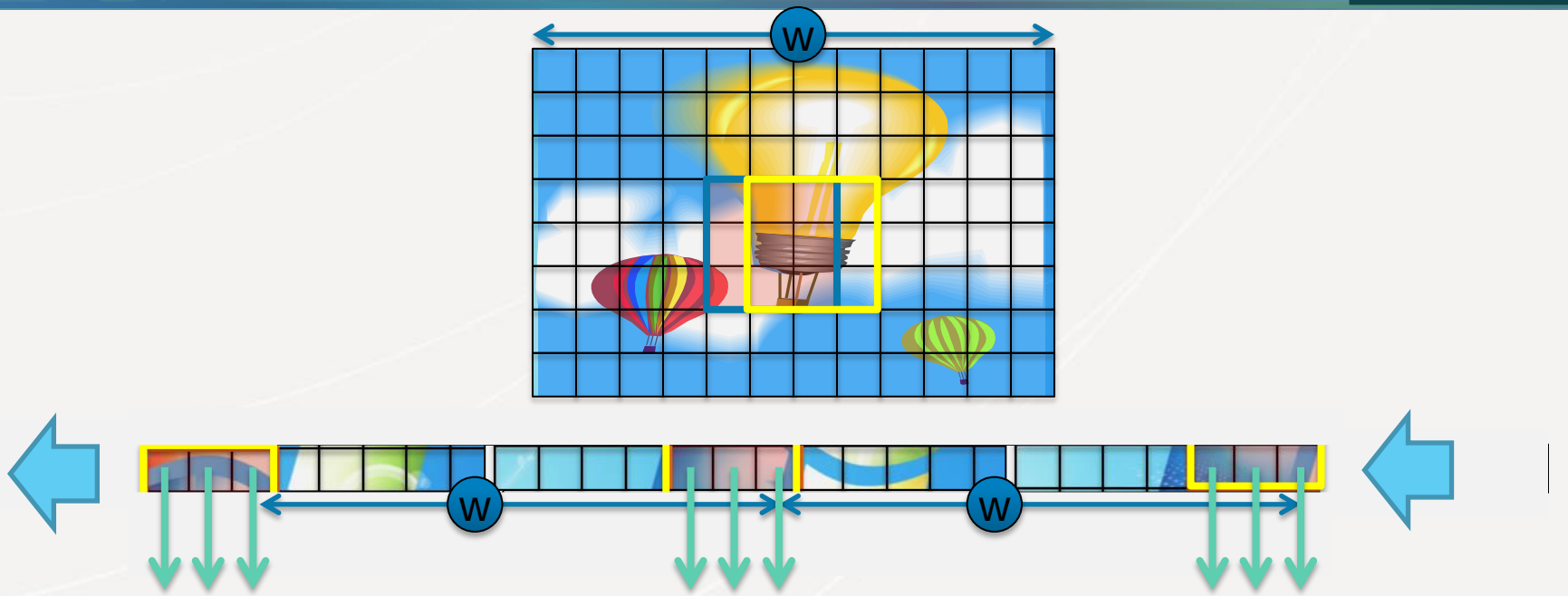
- Expensive hardware!
  - Power overhead
  - Cost overhead: More built in addressing flexibility than we need
- Why not customize the cache for the application?

Optimizing the "Cache"

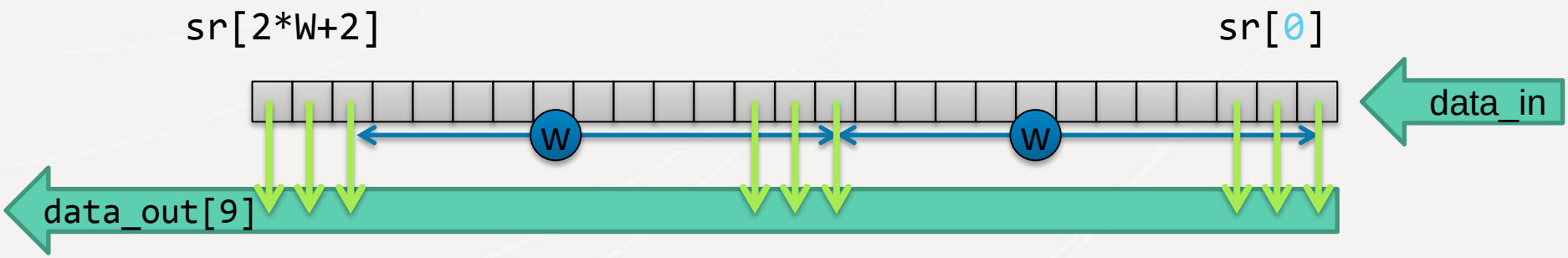


- What happens when we move the window one pixel to the right?

Optimizing the "Cache"



Shift Registers in Software

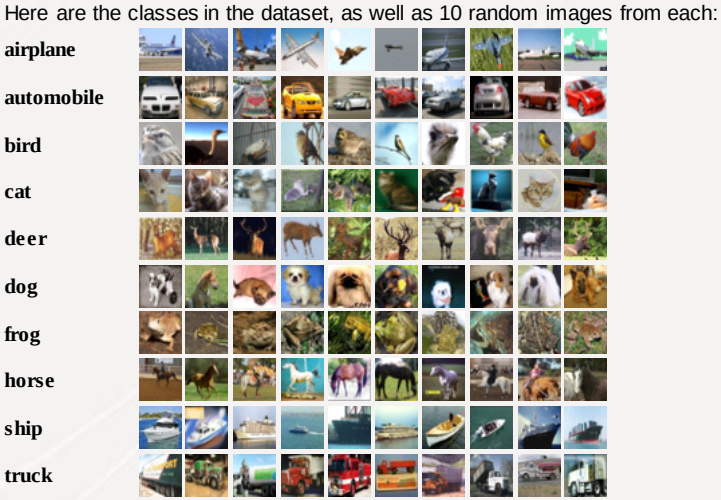



```
pixel_t sr[2*W+3];
while(keep_going) {
  // Shift data in
  #pragma unroll
  for(int i=1; i<2*W+3; ++i)
    sr[i] = sr[i-1]
  sr[0] = data_in;

  // Tap output data
  data_out = {sr[  0], sr[    1], sr[    2],
              sr[  w], sr[  w+1], sr[  w+2]
              sr[2*w], sr[2*w+1], sr[2*w+2]}
  // ...
}
```

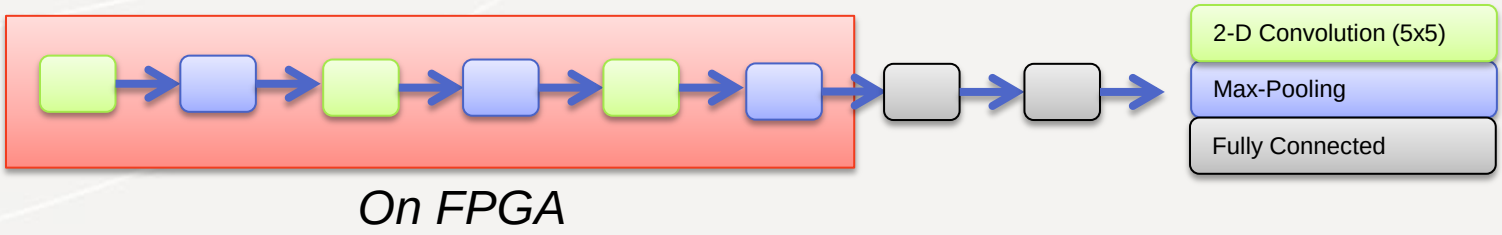
- *Managing data movement to match the FPGA's architectural strengths is key to obtaining high performance.*

Building a CNN for CIFAR-10 on an FPGA

- CIFAR-10 Dataset
  - 60000 32x32 color images
  - 10 classes
  - 6000 images per class
  - 50000 training images
  - 10000 test images



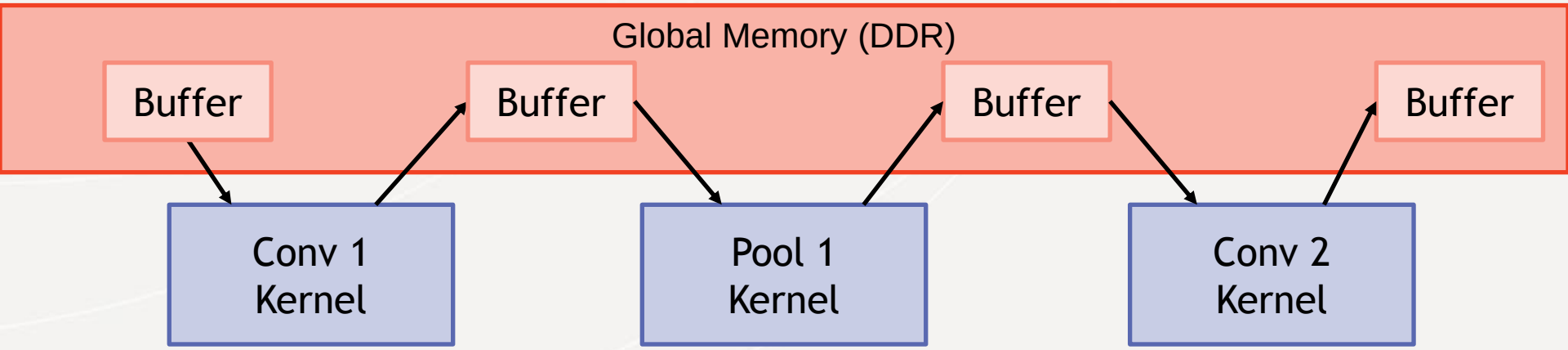
- Many CNN implementations are available for CIFAR-10
  - Cuda-convnet provides a baseline implementation that many works build upon



CIFAR-10 CNN: Traditional OpenCL Implementation

Stratix V Implementation :

- Processes 183 images per second for CIFAR-10



- High-latency: Requires access to global memory
- High memory-bandwidth
- Requires host coordination to pass buffers from one kernel to another

CIFAR-10 CNN: Kernel-to-Kernel Channels

Stratix V Implementation :
- Processes 400 images per second for CIFAR-10

- Channel declaration:
  - Create a queue:
    value_type channel();

```
channel int my_channel;
```

- Channel write:
  - Push data into the queue:
    void write_channel_altera(**channel** &ch, **value_type** data);

```
write_channel_altera(my_channel, x);
```

- Channel read:
  - Pop the first element from the queue
    value_type read_channel_altera(**channel** &ch);

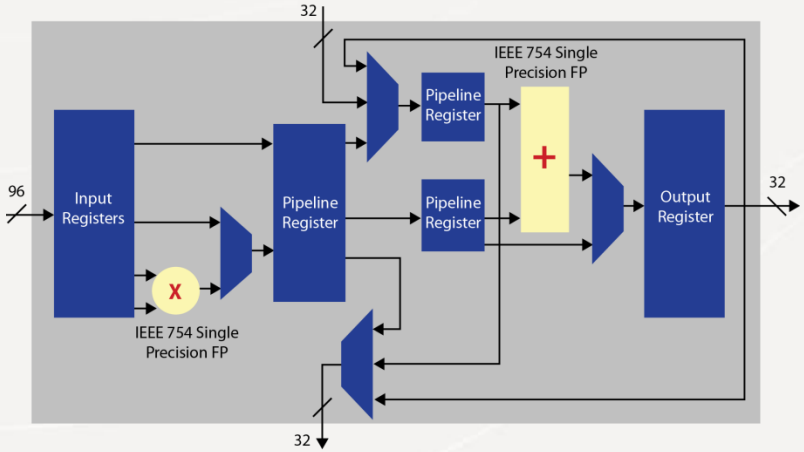
```
int y = read_channel_altera(my_channel);
```

CIFAR-10: FPGA Code

```
#pragma OPENCL_EXTENSION cl_altera_channel
: enable

// Declaration of Channel API data types
channel float prod_conv1_channel;
channel float conv1_pool1_channel;
channel float pool1_conv2_channel;
channel float conv2_pool2_channel;
channel float pool2_conv3_channel;
channel float conv3_pool3_channel;
channel float pool3_cons_channel;

__kernel void
convolutional_neural_network_prod(
  int batch_id_begin,
  int batch_id_end,
  __global const volatile float * restrict
input_global)
{
  for(...) {
   write_channel_altera(
    prod_conv1_channel,
    input_global[...]);

   write_channel_altera(
    prod_pool3_channel,
    input_global[...]);
   }
}
```

- Entire algorithm can be expressed in ~500 lines of OpenCL for the FPGA
- Kernels are written as standard building blocks that are connected together through channels
- The **shift register convolution building block** is the portion of this code that is used most heavily
- The concept of having multiple concurrent kernels **executing simultaneously** and **communicating directly** on a device is currently unique to FPGAs
  - Will be portable in OpenCL 2.0 through the concept of "**OpenCL Pipes**"

Effect of Floating Point FPGAs



Arria-10 Implementation :

- Processes 6800 images per second for CIFAR-10

- Altera's Arria-10 family of FPGA introduces DSP blocks with a dedicated floating point mode.
- Each DSP includes a IEEE 754 single precision floating-point multiplier and adder
  - FPGA logic blocks (lookup tables and registers) are no longer needed to implement floating point functions
  - Massive resource savings allows many more processing pipelines to run simultaneously on the FPGA

Lessons Learned

- CNN implementations are well suited to pipelined implementations
- Exploiting pipelining on the FPGA requires some attention to coding style to overcome the inherent assumptions of the writing “software”
  - FPGAs do not have caches
  - Need to exploit data reuse in a more explicit way
- The concept of dataflow pipelining will not realize its full potential if we write intermediate results to memory
  - Bandwidth limitations begin to dominate compute
  - Use direct kernel to kernel communication called channels
- Native support for floating point on the FPGA allows order of magnitude performance increase

Resources

- Altera's SDK for OpenCL
  - http://www.altera.com/products/software/opencl/opencl-index.html
- FPGA Optimized OpenCL examples (filters, convolutions, …)
  - http://www.altera.com/support/examples/opencl/opencl.html
- CIFAR-10 dataset
  - http://www.cs.toronto.edu/~kriz/cifar.html
- CUDA-Convnet
  - https://code.google.com/p/cuda-convnet/