

# CO405H

## Computing in Space with OpenSPL

### Topic 2: Basic Concepts of Computing in Space

Oskar Mencer

Georgi Gaydadjiev

Department of Computing  
Imperial College London

<http://www.doc.ic.ac.uk/~oskar/>

<http://www.doc.ic.ac.uk/~georgig/>

**CO405H course page:**

**WebIDE:**

**OpenSPL consortium page:**

<http://cc.doc.ic.ac.uk/openspl16/>

<http://openspl.doc.ic.ac.uk>

<http://www.openspl.org>

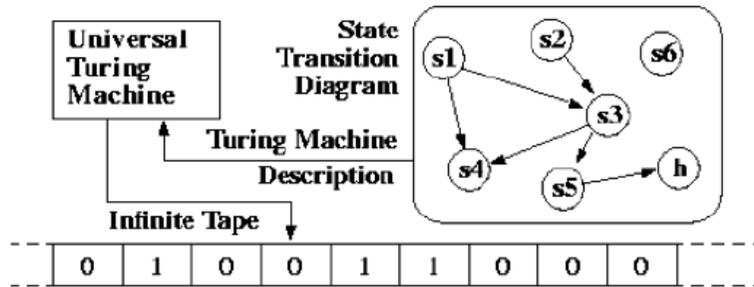
[o.mencer@imperial.ac.uk](mailto:o.mencer@imperial.ac.uk)

[g.gaydadjiev@imperial.ac.uk](mailto:g.gaydadjiev@imperial.ac.uk)

# Overview

- It is all about tradeoffs
  - Space vs Time
  - Control vs Data Flow
  - ...
- Space is great
  - Expressions become Executable Graphs
  - Numbers just flow through the graphs
- How to express all of the above

# Sequential recipe is simple but slow



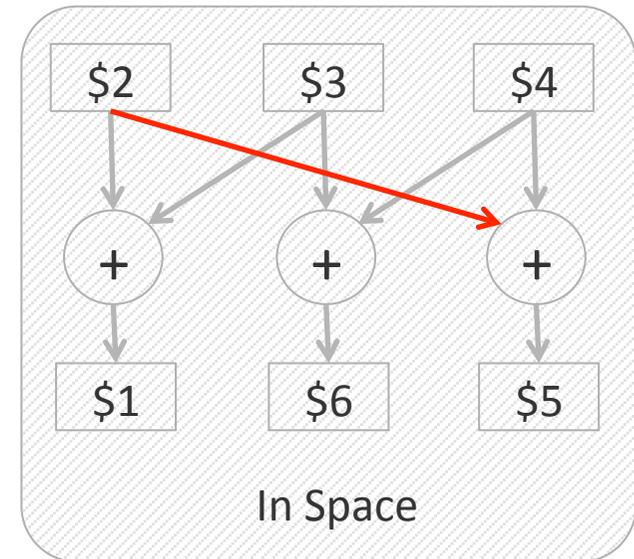
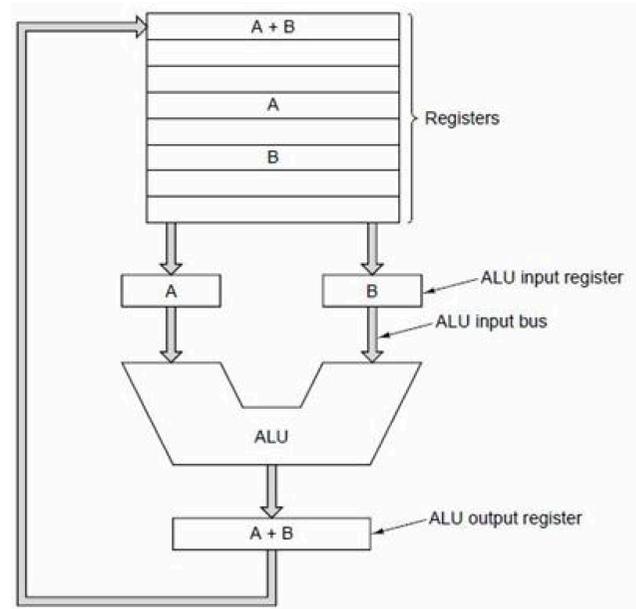
- Universal model but slow due to enforced sequencing
- Operations that can be executed in parallel are often serialized

For example let us consider three independent additions (MIPS):

```

...
add $1, $2, $3
add $5, $4, $2
add $6, $4, $3
...
    
```

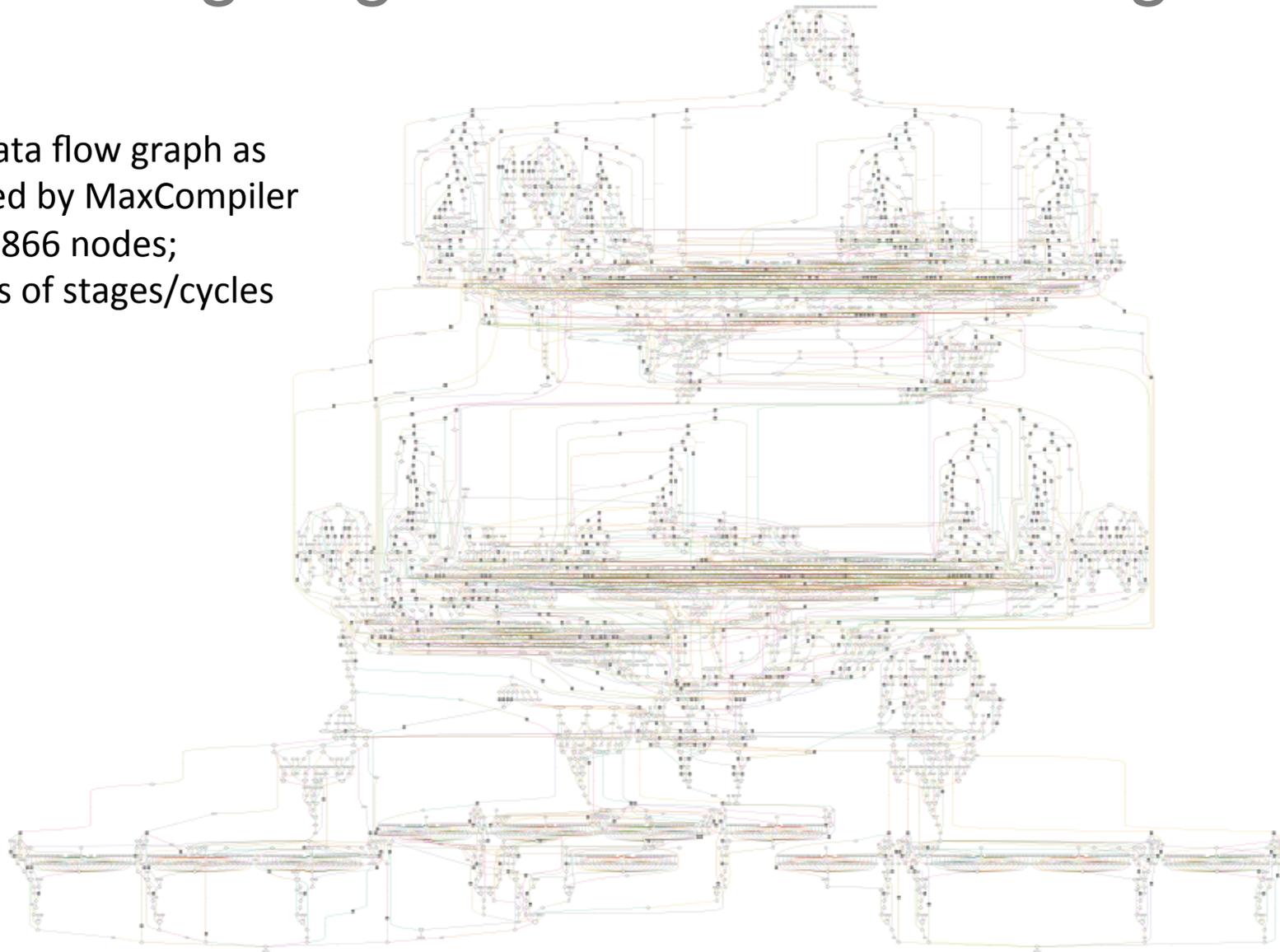
In Time



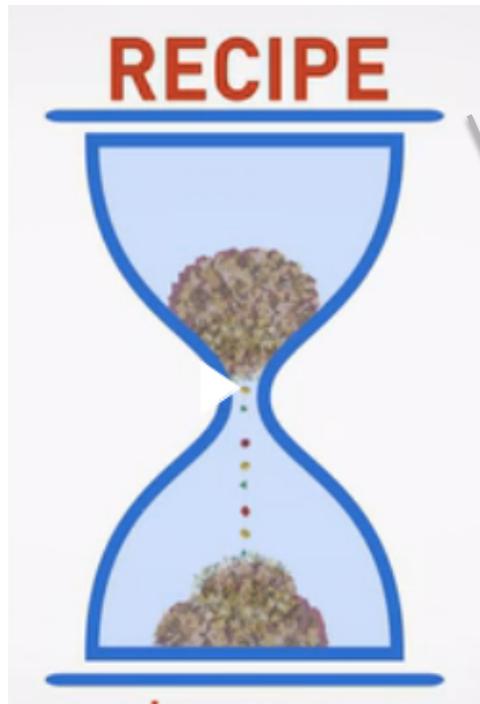
In Space

# Enabling large scale dataflow designs

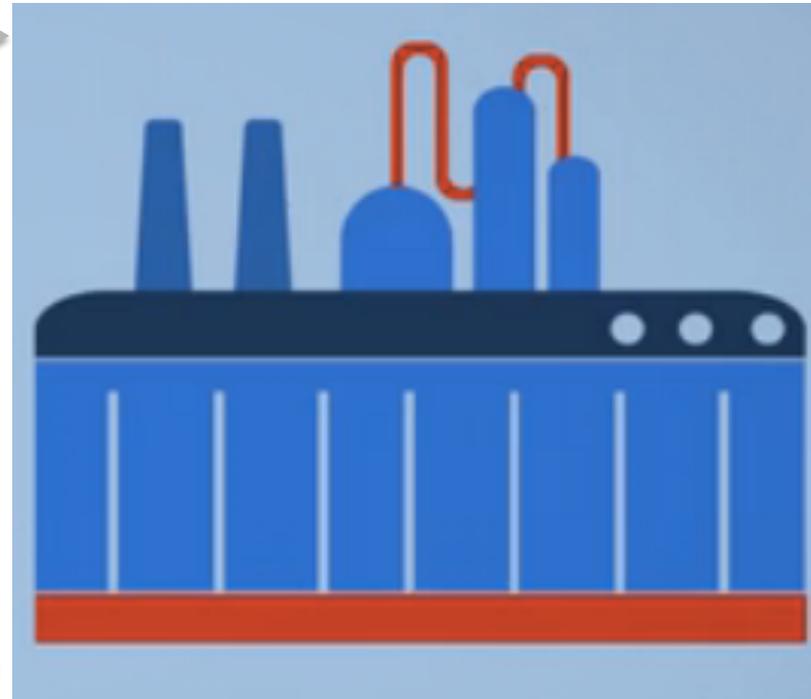
Real data flow graph as  
generated by MaxCompiler  
4866 nodes;  
10,000s of stages/cycles



# Computing in Space vs Time

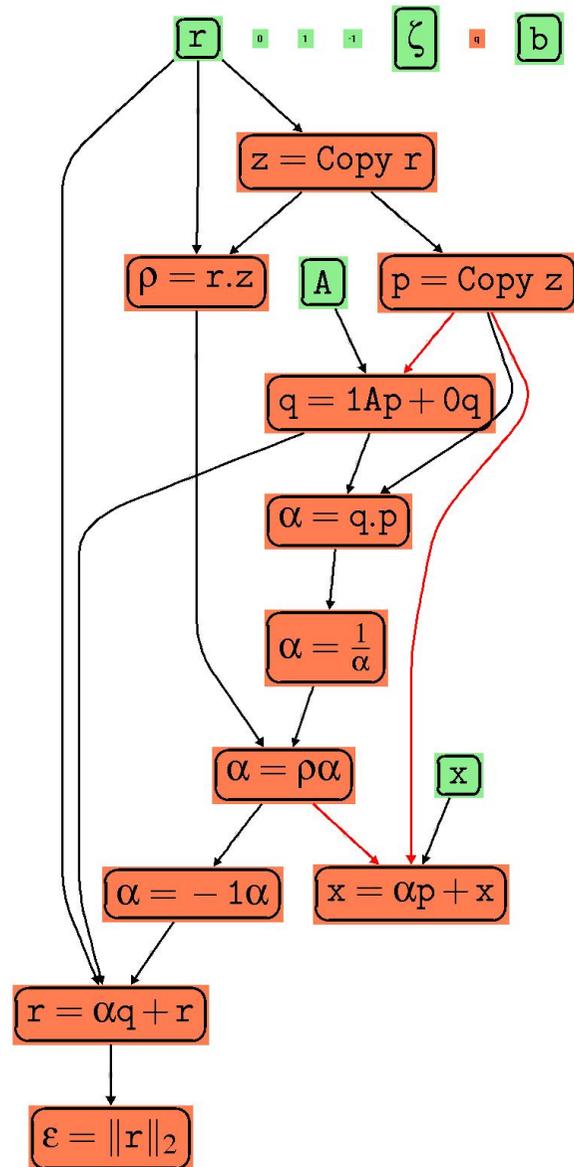


Computing in Time:  
*Follow a recipe step by  
step one at the time*



Computing in Space:  
*Build a "recipe specific" factory with  
multiple paths performed simultaneously*

# Spatial “recipe” is fast but larger in area



Conjugate Gradient method for solving  $Ax = b$

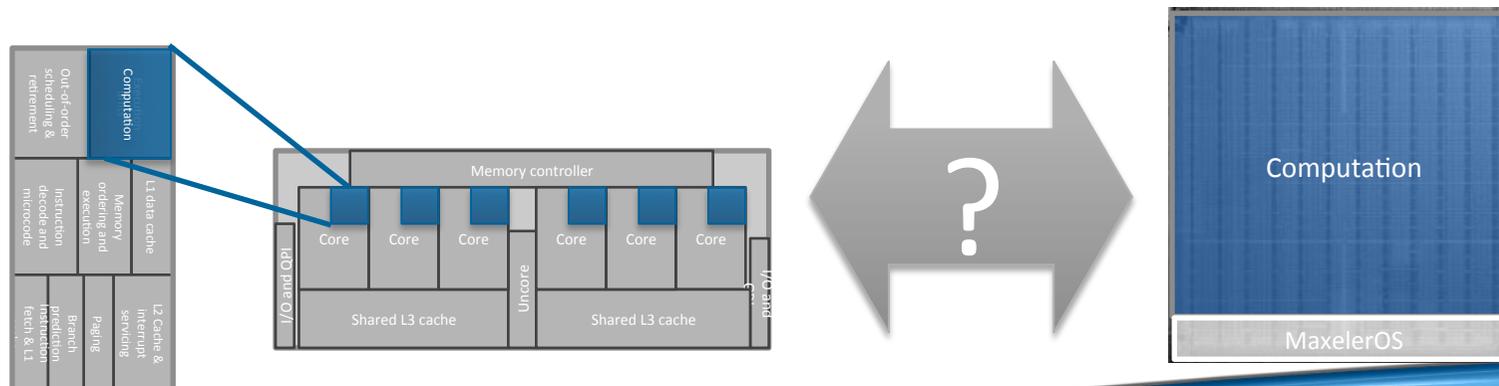
Multiple, interconnected hardware computational units (some of them ALUs) can implement this a bit more complicated function. The spatial version has 8 stages while the sequential (with 1 ALU) will need way too many iterations (even running at 10x faster speed)

In terms of area one ALU and a simple register file is way much smaller than the entire implementation of the DFG

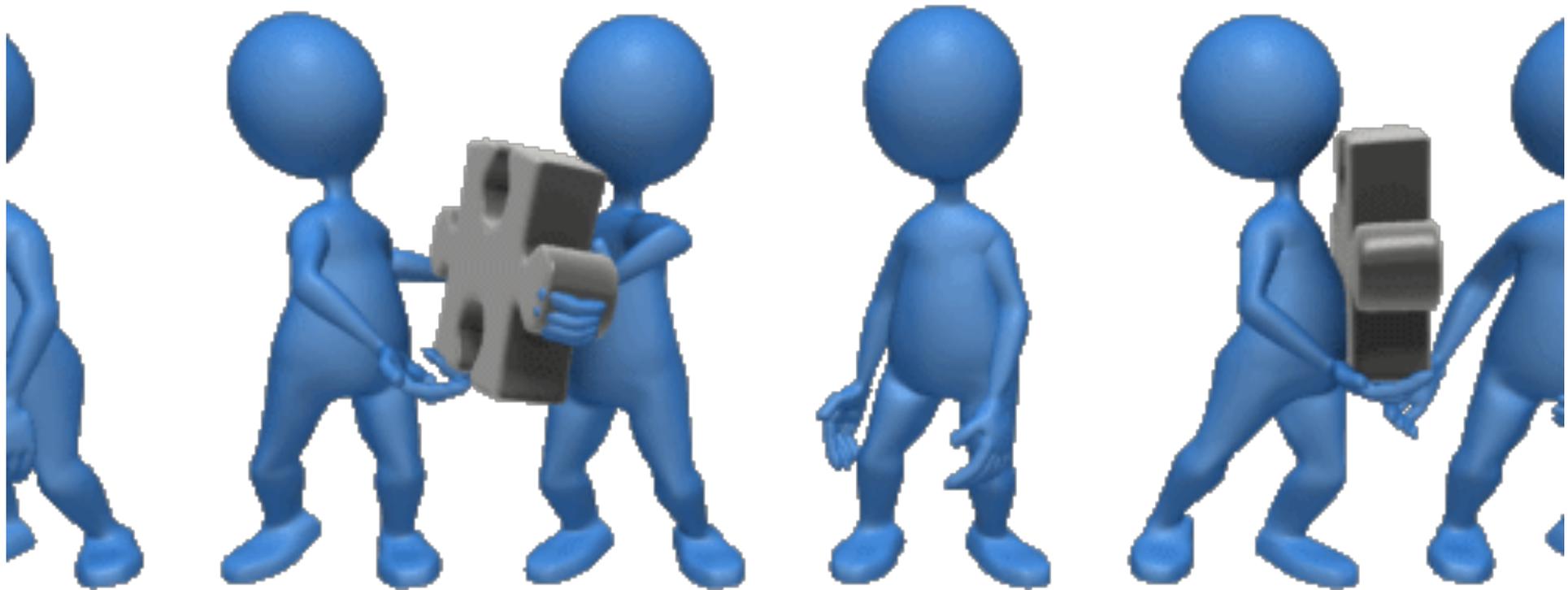
\*DFG- Data Flow Graph

# Is the space intensity a real problem?

- Technology provides steadily growing number of transistors per unit area
- In sequential systems the additional transistors are used to “cheat time” (bring often used data close and predict the control flow) leading to:
  - Large caches
  - Very complex control
- In spatial systems the data flows through a structure composed by arithmetic operations laid out on the chip surface avoiding:
  - Caches
  - Control structures
- CPU-based temporal architectures are “compute sparse”, hence spatial can offer benefits by being “compute dense”



The most efficient way to move **lot's** of stuff?



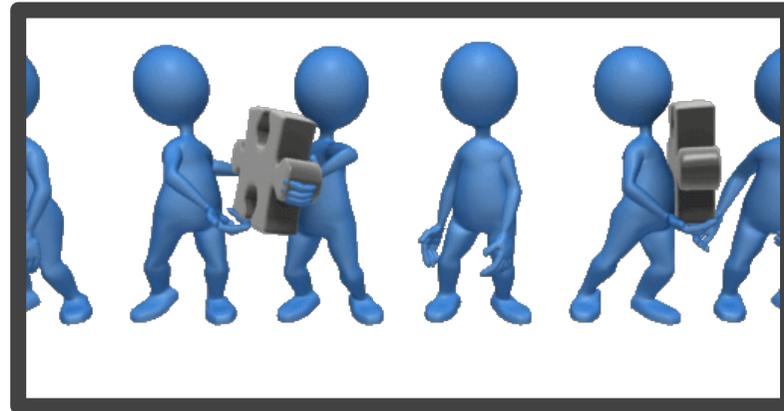
Courtesy of Bob Clapp, Stanford Geophysics

# Control Flow versus Data Flow

**CPUs**



**DFEs**



**Which one would you rather do now?**



# Control Flow versus Data Flow

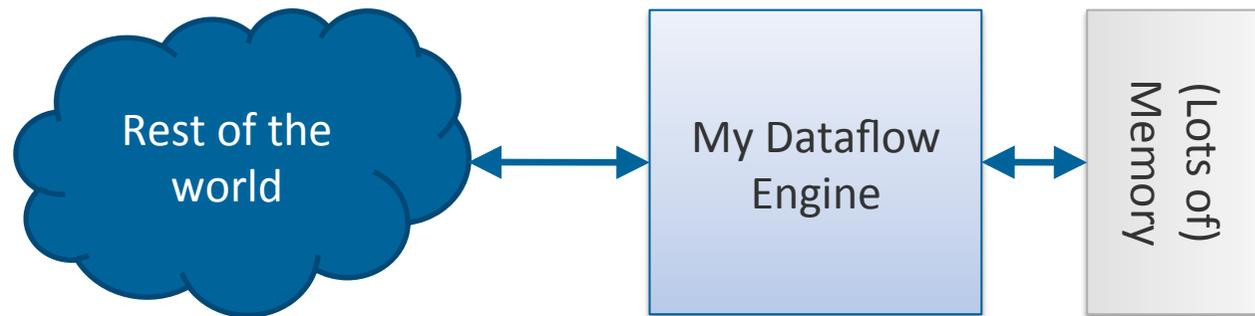
- Control Flow:
  - It is all about how instructions “move”
  - Data may move along with instructions (secondary issue)
  - Order of computation is the key
- Data Flow:
  - It is about how data moves through a set of “instructions” in 2D space
  - Data moves will trigger control
  - Data availability, transformations and operation latencies are the key

# Data Flow specific properties

- No needed for:
  - shared memory
  - program counter
  - control sequencer
  - branch prediction
- Special mechanisms are required to:
  - data availability detection
  - orchestration of data tokens and “instructions”
  - chaining of asynchronous “instruction” execution

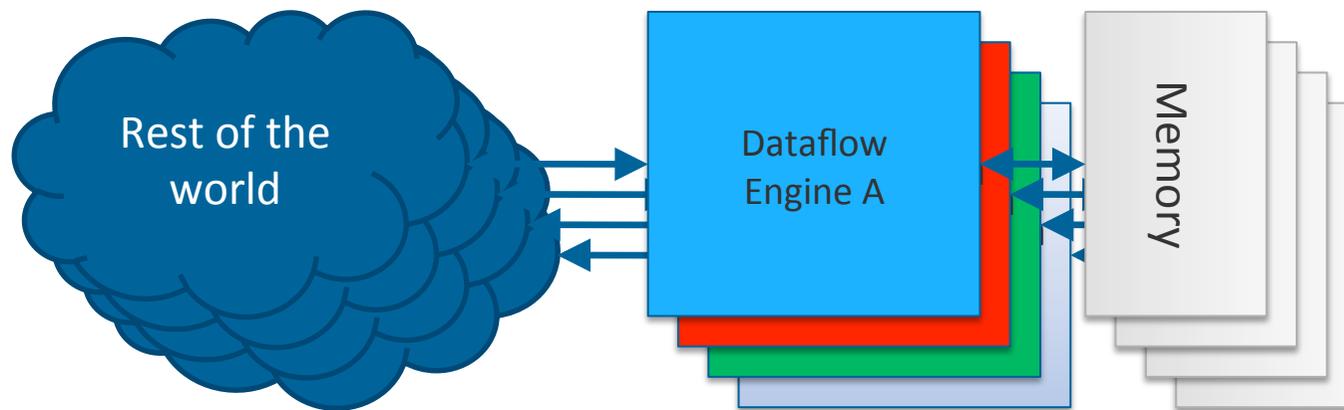
# Dataflow Computing

- A custom chip for a specific application
- No instructions → no instruction decode logic
- No branches → no branch prediction
- Explicit parallelism → No out-of-order scheduling
- Data streamed onto-chip → No multi-level caches



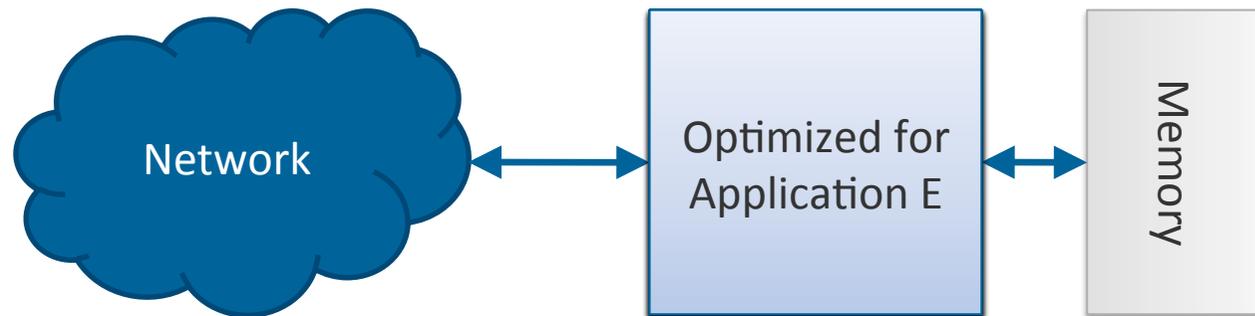
# Special Purpose Computers

- But we have more than one application
- Generally impractical to have machines that are completely optimized for only one code
  - Need to run many applications on a typical cluster

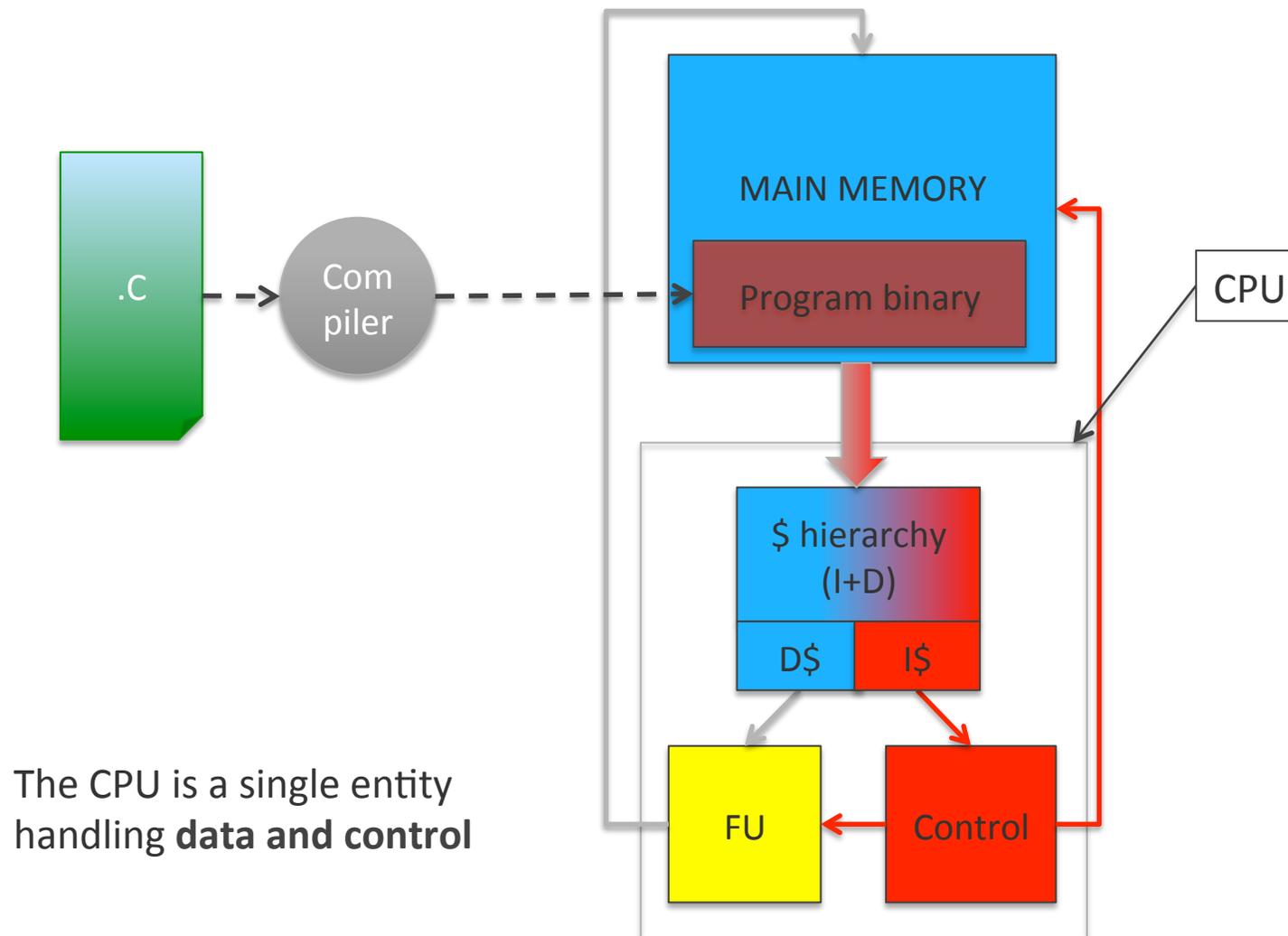


# A Special Purpose Computer

- Use a *reconfigurable* chip that can be reprogrammed at runtime to implement:
  - Different applications
  - Or different versions of the same application

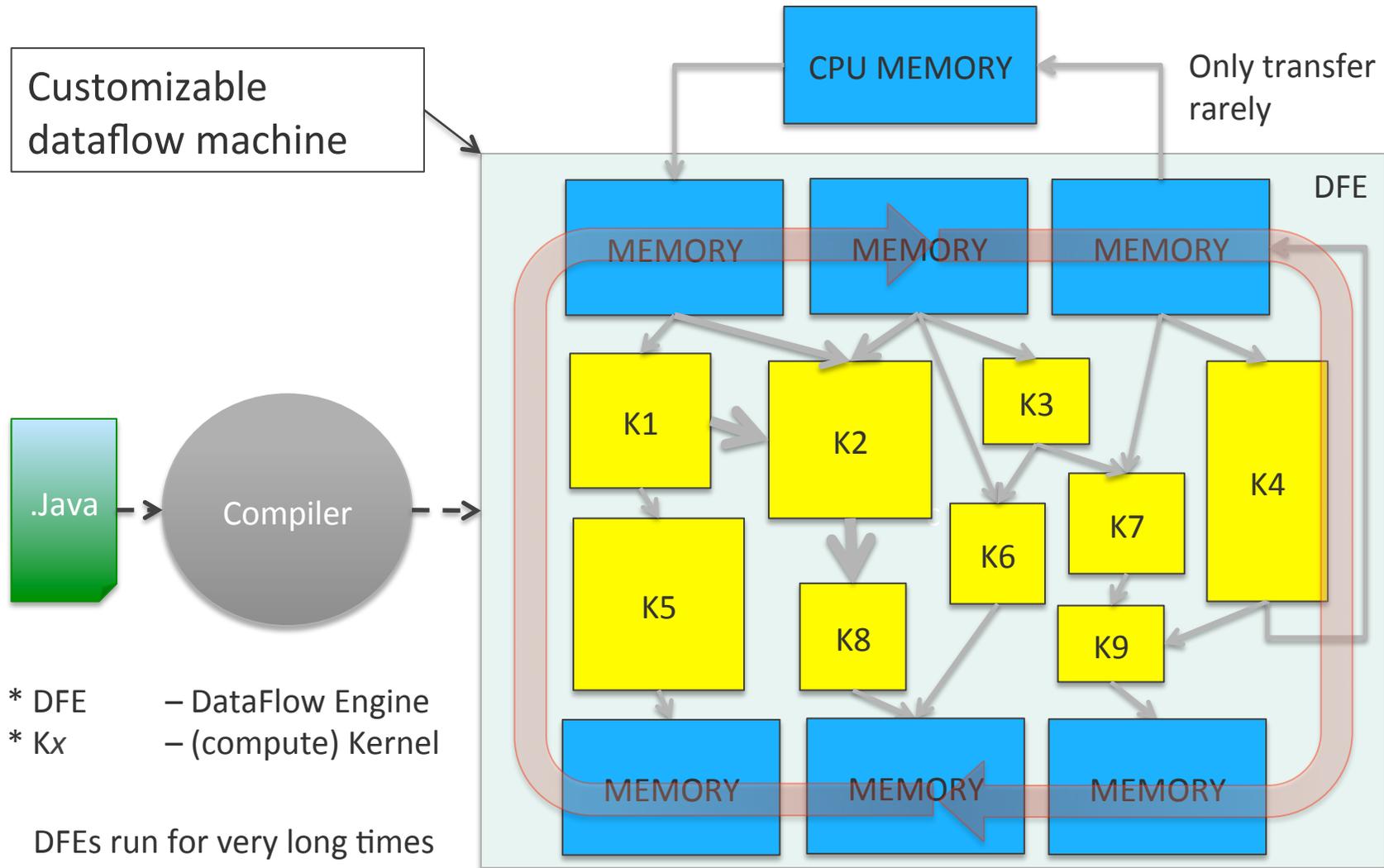


# Control-flow Machine



The CPU is a single entity handling **data and control**

# Spatial Computing Machine



# Accelerating Real Applications

- The majority of lines of code in most applications are scalar
- CPUs are good for: latency-sensitive, control-intensive, non-repetitive code
- Dataflow engines are good for: high throughput repetitive processing on large data volumes

**→ A system should contain both**



	Lines of code
Total Application	1,000,000
Kernel to accelerate	2,000
Software to restructure	20,000

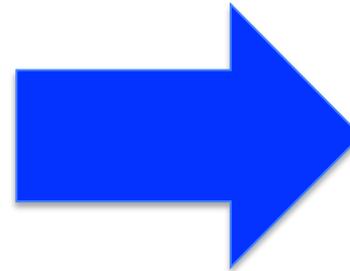
# Major Bottlenecks: Examples

	Throughput	Latency
Memory	Convolution	Graph algorithms
Compute	Monte Carlo	Optimization

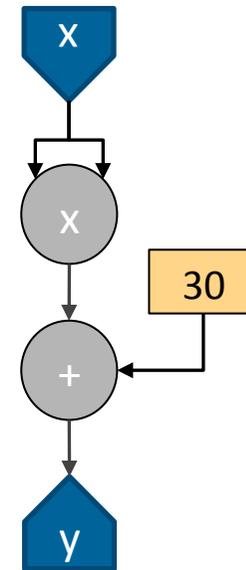
# Converting Expression to Space (simple)

$$y_i = x_i \times x_i + 30$$

```
int*x, *y;  
for (int i =0; i < DATA_SIZE; i++)  
  y[i]= x[i] * x[i] + 30;
```

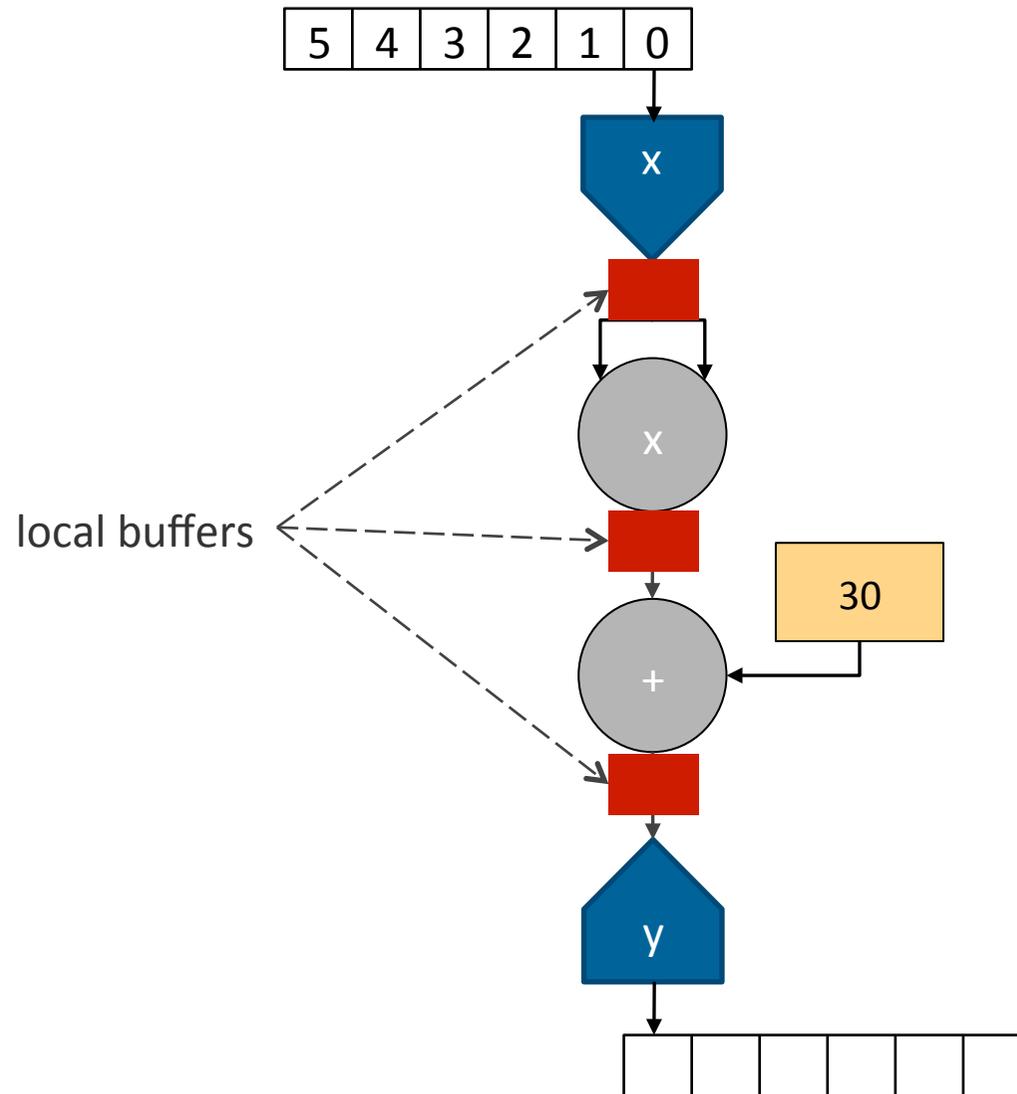


Input stream of integer elements 'x'

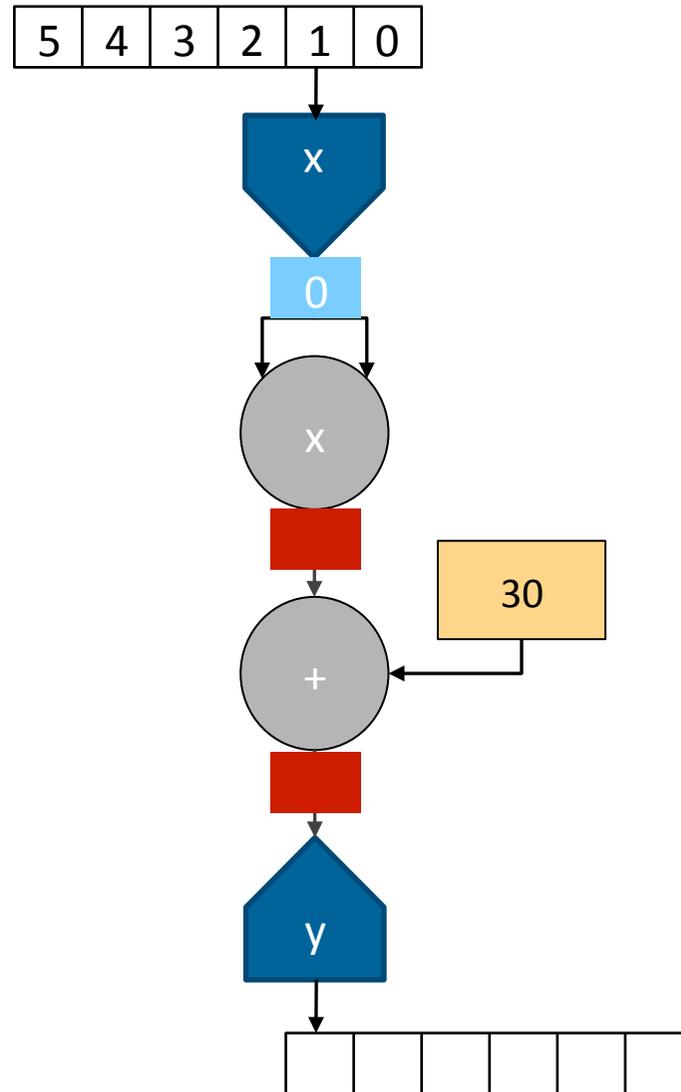


Output stream of integer elements 'y'

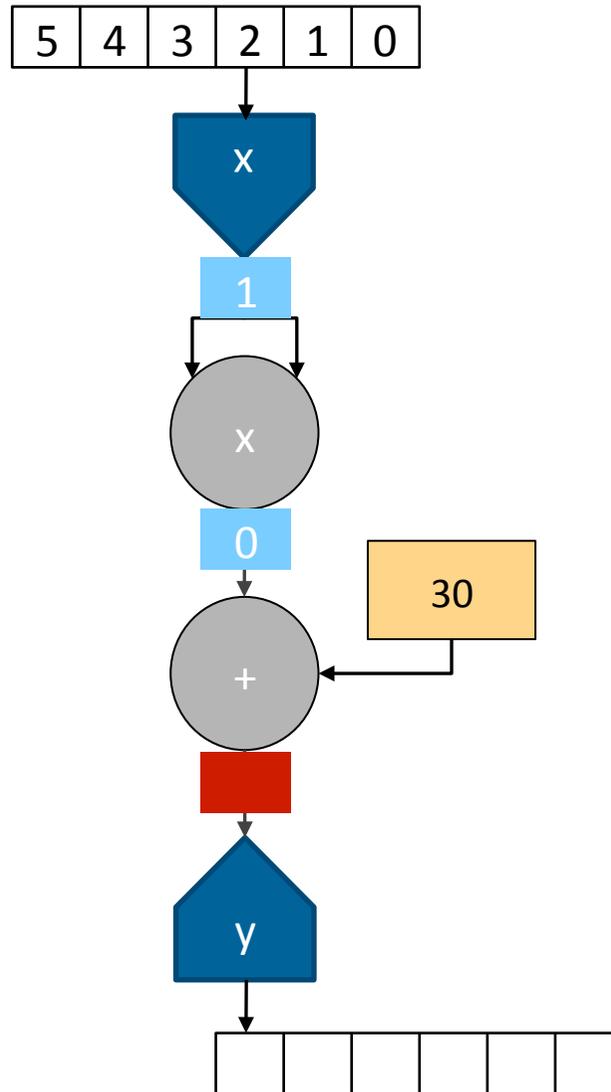
# Flowing elements



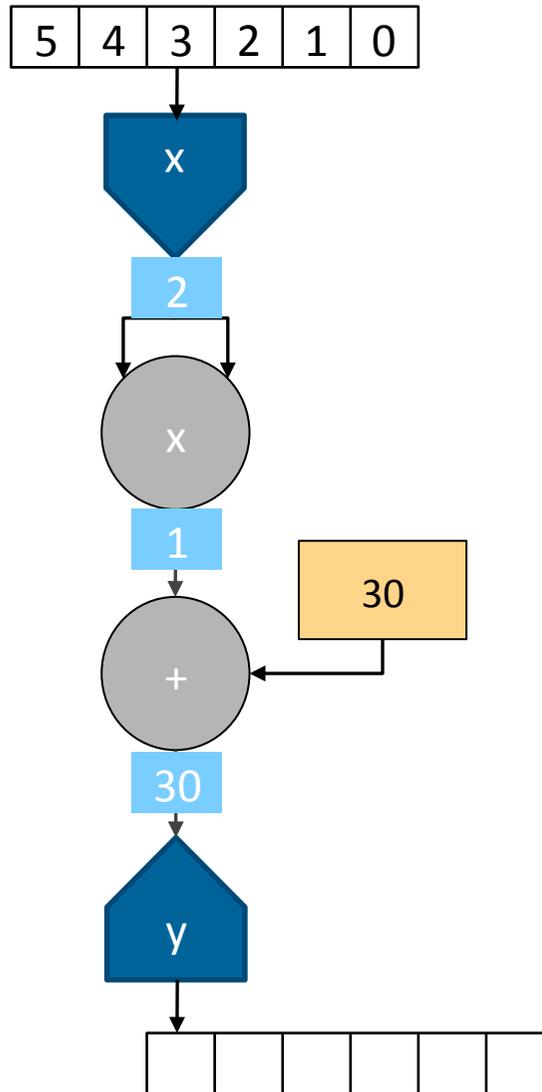
# Flowing elements



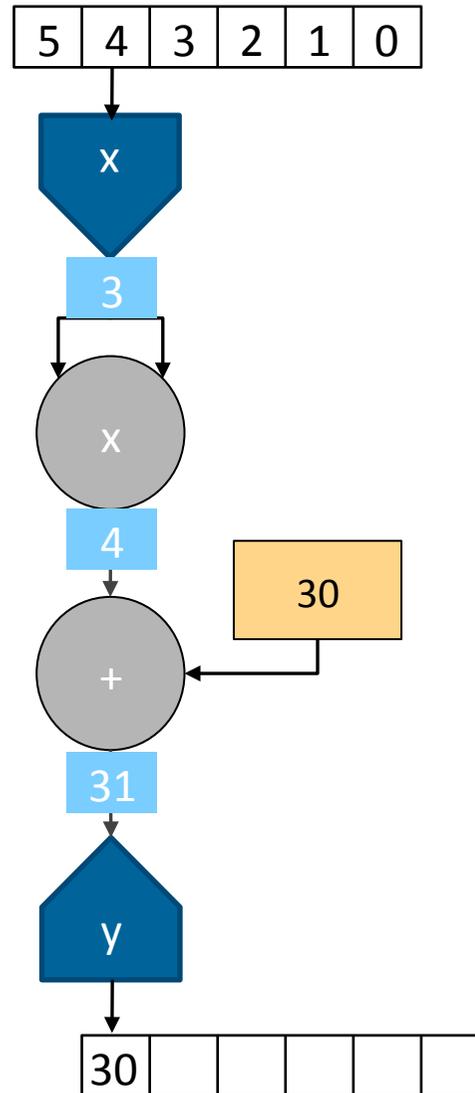
# Flowing elements



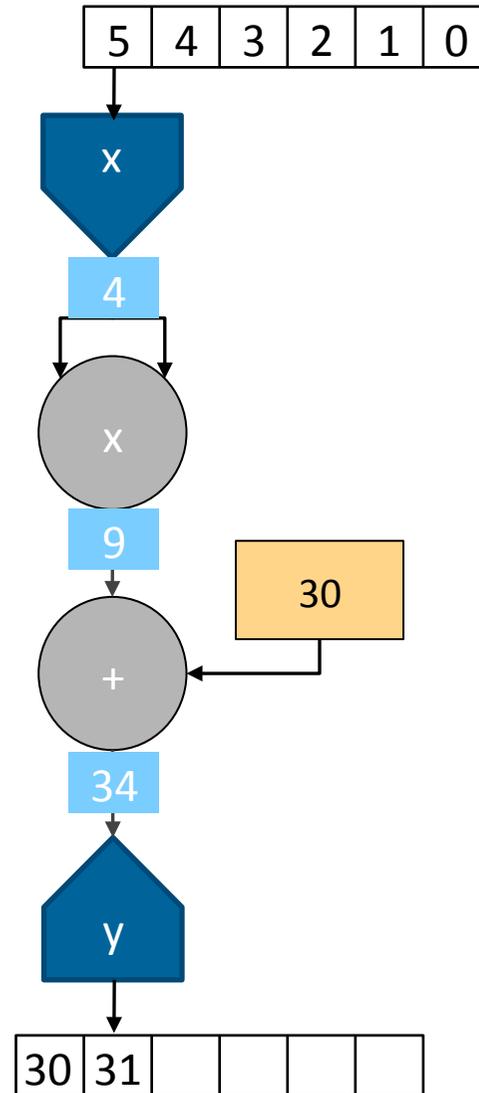
# Flowing elements



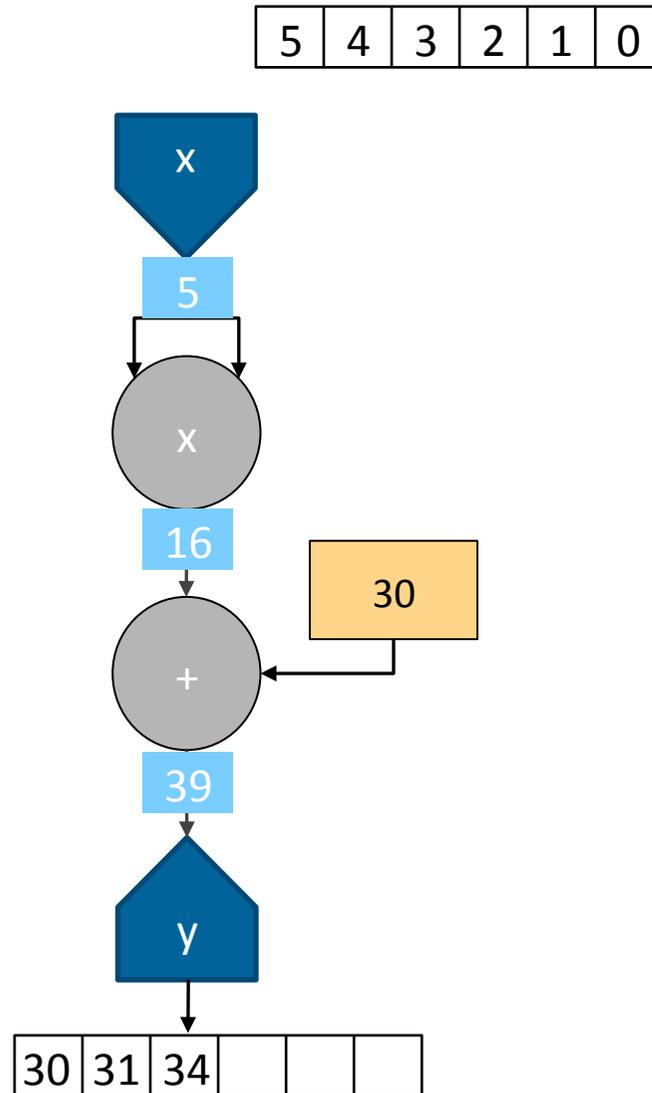
# Flowing elements



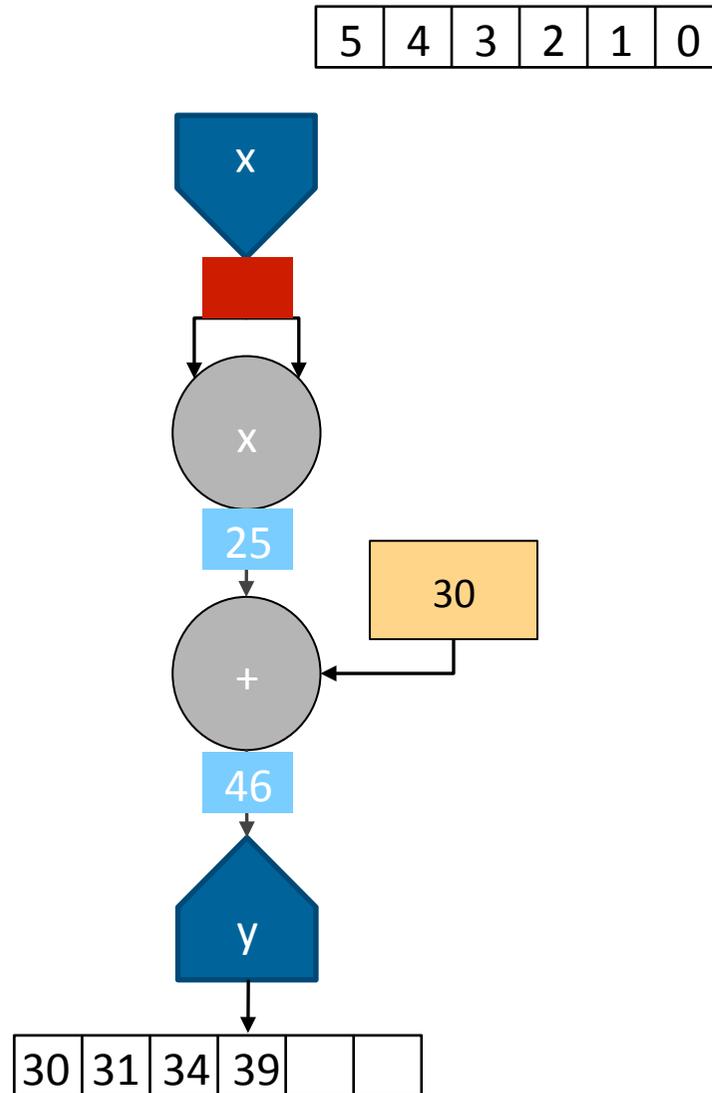
# Flowing elements



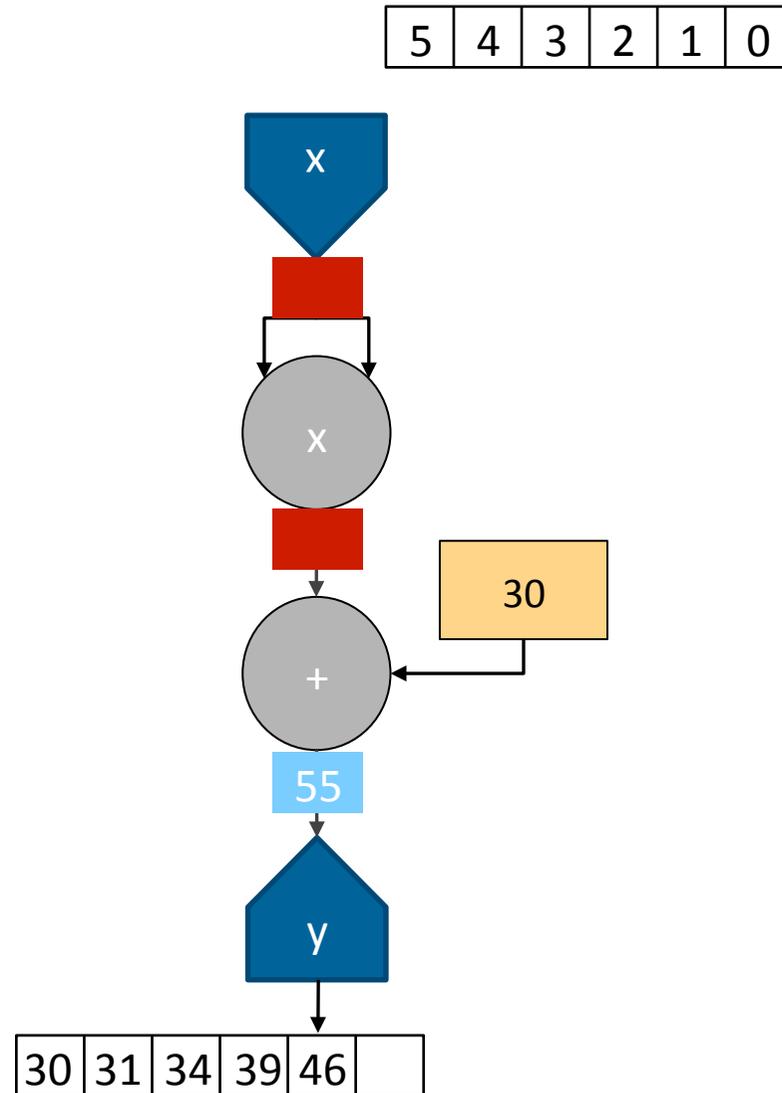
# Flowing elements



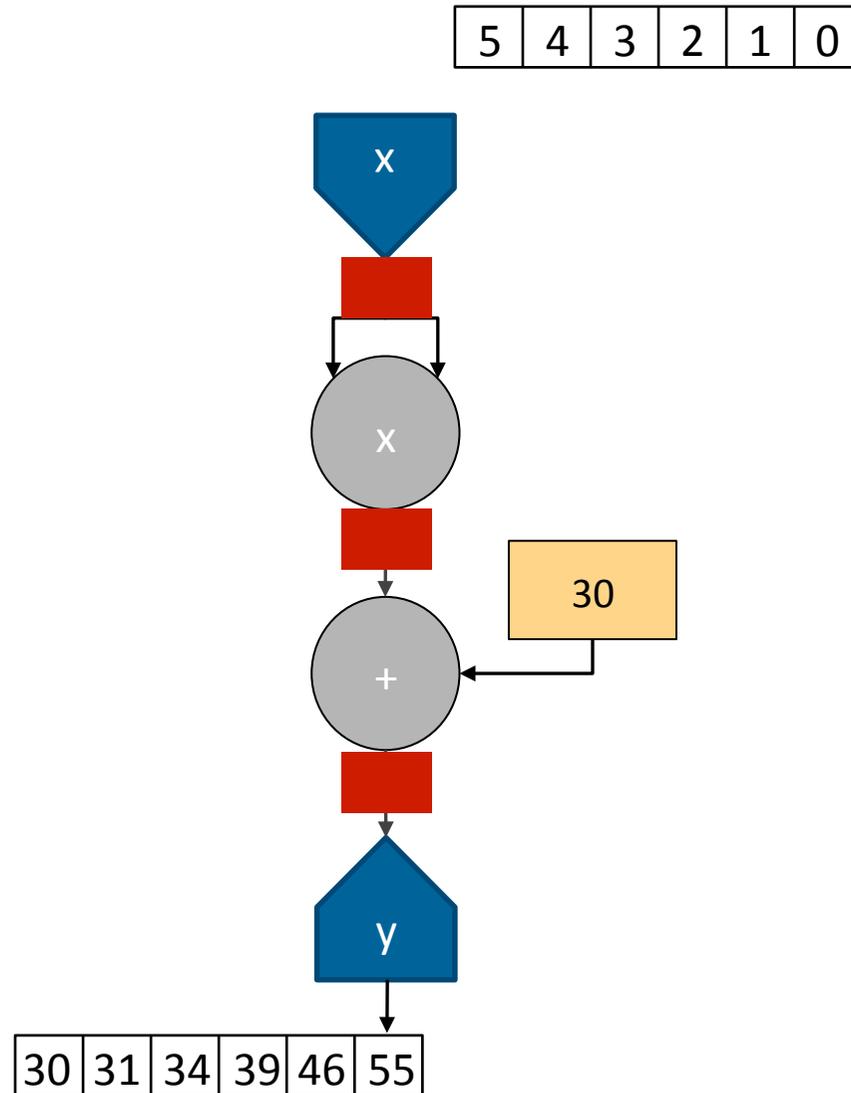
# Flowing elements



# Flowing elements

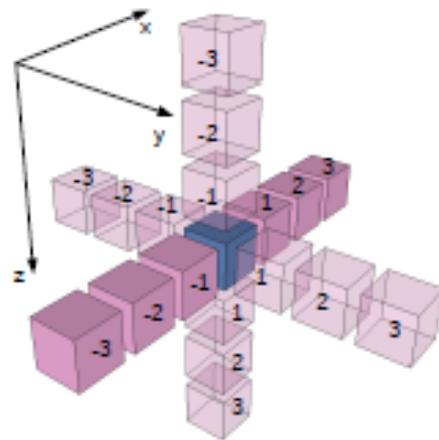


# Flowing elements



# But data can have multiple dimensions

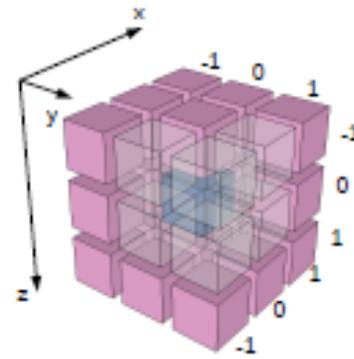
## 3D Finite Difference Coefficients



(a) 7-point star stencil

19 MulADDs

Local Buffer = 6 slices



(b) 3x3x3 cube stencil

27 MulADDs

Local Buffer = 3 slices

# How to express all of this?

- We need streaming variables type
- We should be able to access multi-dimensional data
- On the boundaries to main memory:
  - Address generation: walking through the multi-dim data
  - Address coalescing: maximise locality
  - Address contention avoidance
  - Memory channel parallelism management
- Managing local buffers
- Graph and loop “balancing”
- Minimisation of data movement
- ...

# Is there an optimal solution

- Optimality depends on the application / dataset
- More specifically it depends on the system's "Bottleneck" for the application
- Possible Bottlenecks:
  - Main memory access latency
  - Main memory access bandwidth
  - Main memory capacity
  - Local memory size
  - Arithmetic resources
  - Arithmetic operation latency
  - Interconnect bandwidths

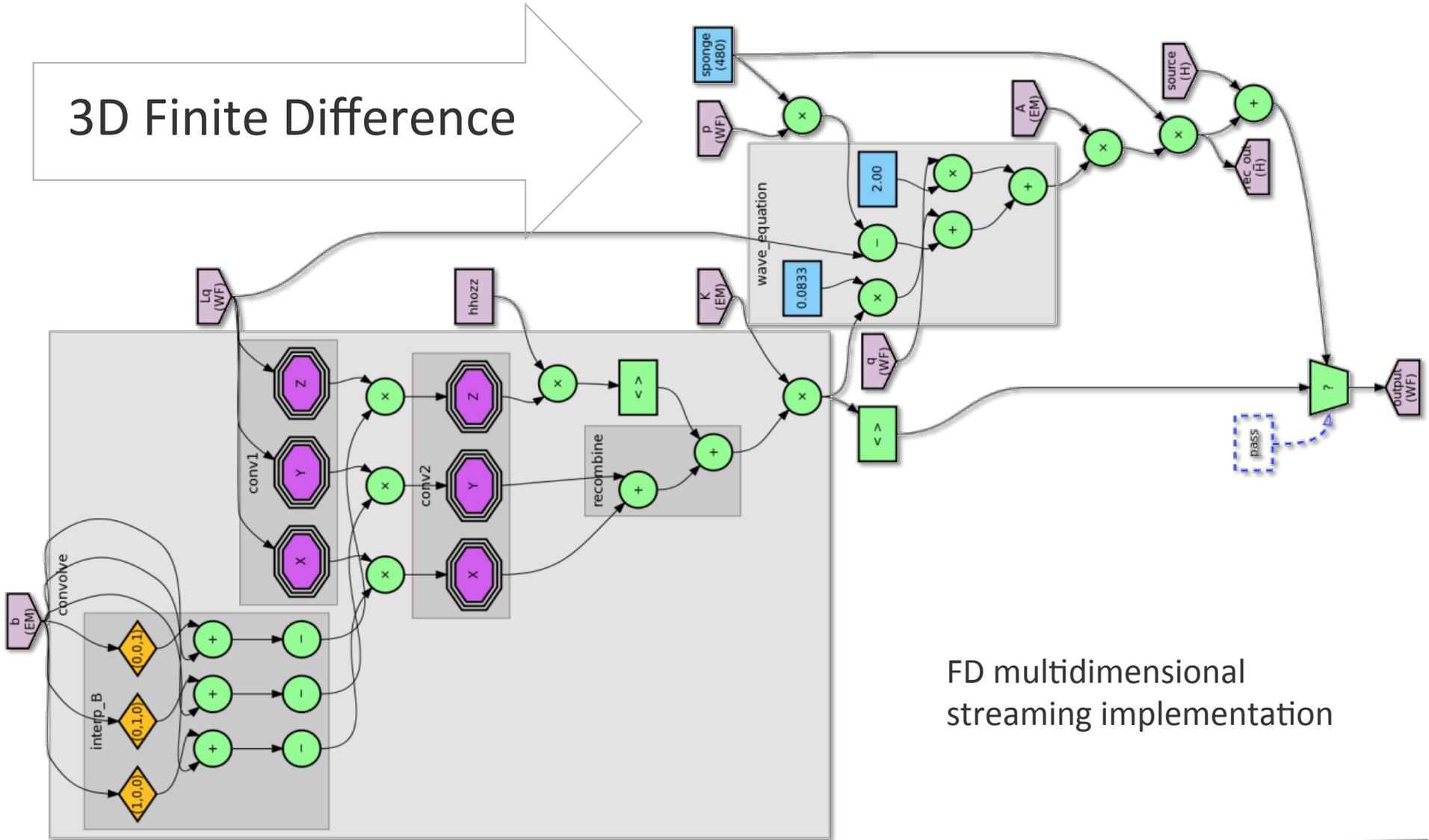


# Alternatives: How can we program in space

- Schematic entry of circuits
- Traditional Hardware Description Languages
  - VHDL, Verilog, SystemC.org
- Object-oriented languages
  - C/C++, Python, Java, and related languages
- Functional languages: e.g., Haskell
- High level interface: e.g., Mathematica, MatLab
- Schematic block diagram e.g., Simulink
- Domain specific languages (DSLs)

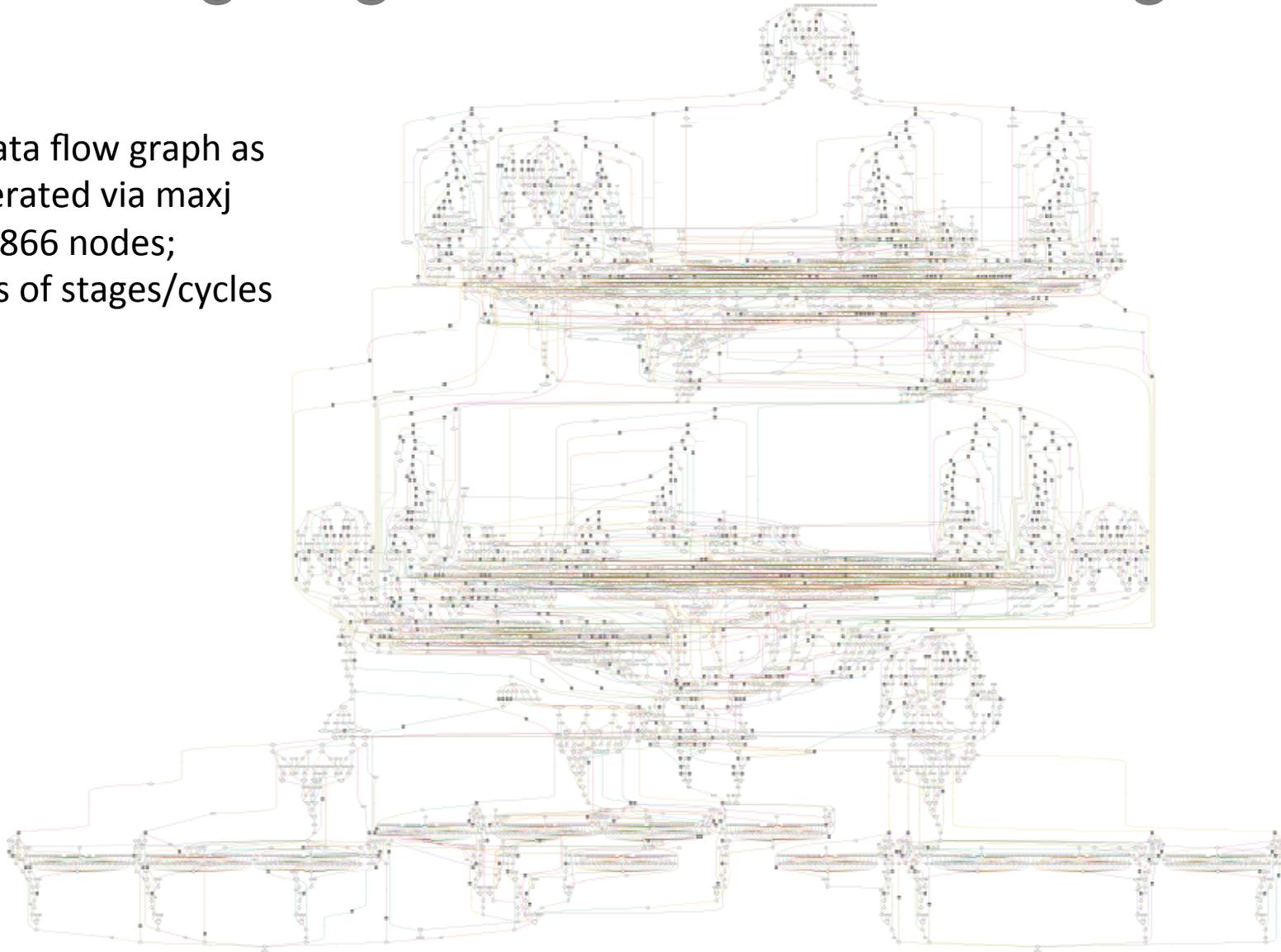
# What is our goal?

3D Finite Difference



# Enabling large scale dataflow designs

Real data flow graph as  
generated via maxj  
4866 nodes;  
10,000s of stages/cycles



# Summary

- The art of making the right tradeoffs is a key
- Application Specific systems are efficient
  - Hybrid Control + Data Flow systems are the best
- Space and time co-exist and should be balanced
- “Fixing” the computational structure and flowing the data through it brings simplicity
- Data access should closely match data structures organization
- There is a practical need to conveniently express very complex and large dataflow graphs
- OpenSPL provides a forum to develop the above
- MaxJ is a commercial implementation of OpenSPL principles