CO405H

Computing in Space with OpenSPL

Topic 1: 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/

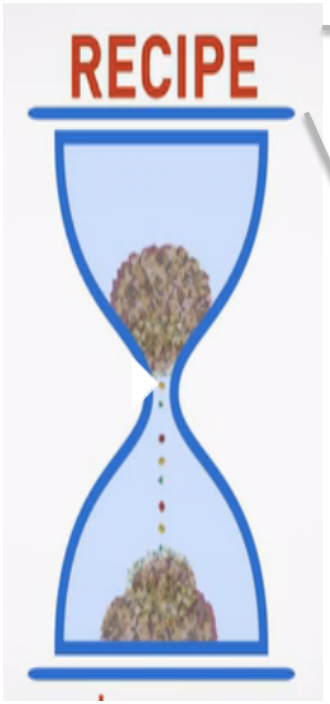
CO405H course page: http://cc.doc.ic.ac.uk/openspl14/
WebIDE: http://openspl.doc.ic.ac.uk
OpenSPL consortium page: http://www.openspl.org

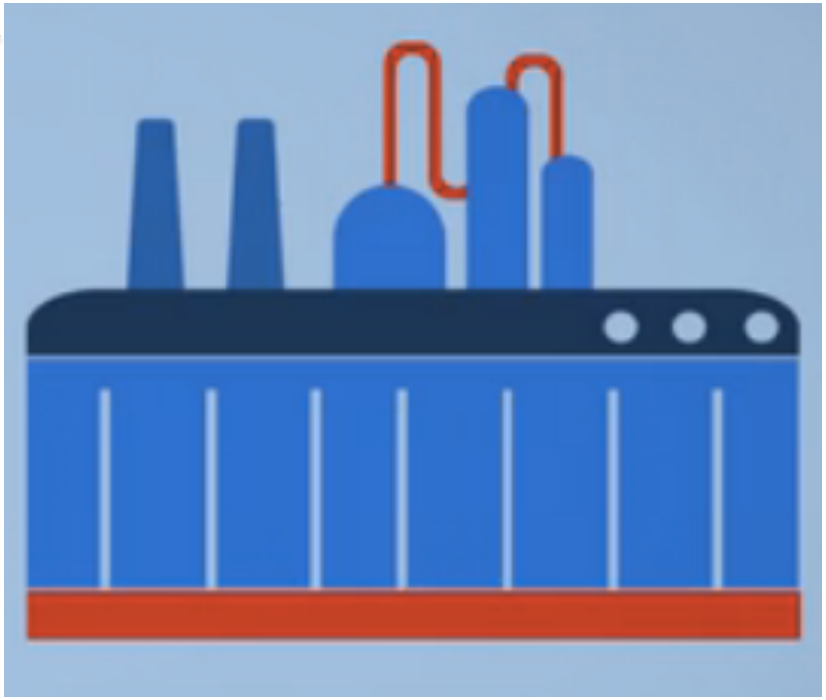
o.mencer@imperial.ac.uk g.gaydadjiev@imperial.ac.uk

Overview

- Tradeoffs Space vs Time
- Tradeoff Control vs Data Flow
- Tradeoffs Compute vs Memory

Computing in Time vs Space

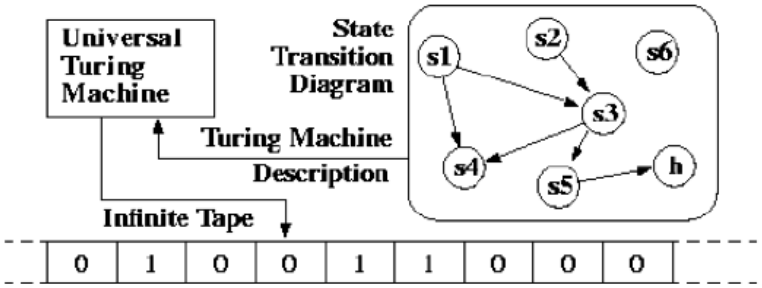




Computing in Time:
Follow a recipe step by step one at the time

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

Sequential recipe is simple but slow

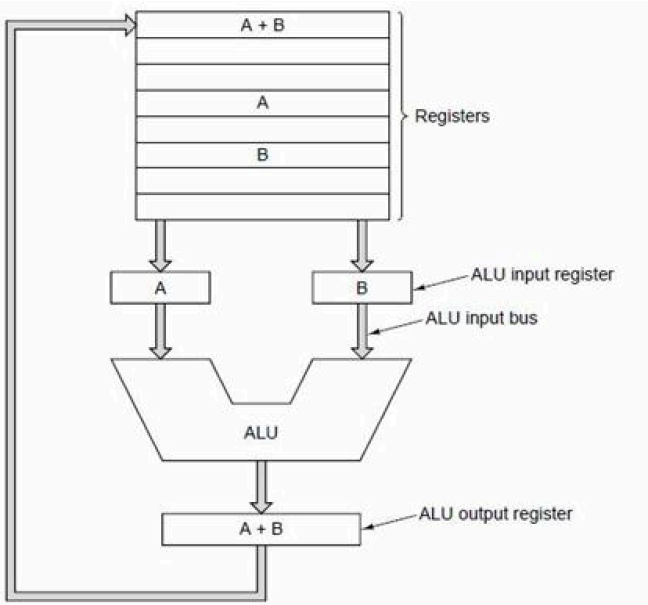


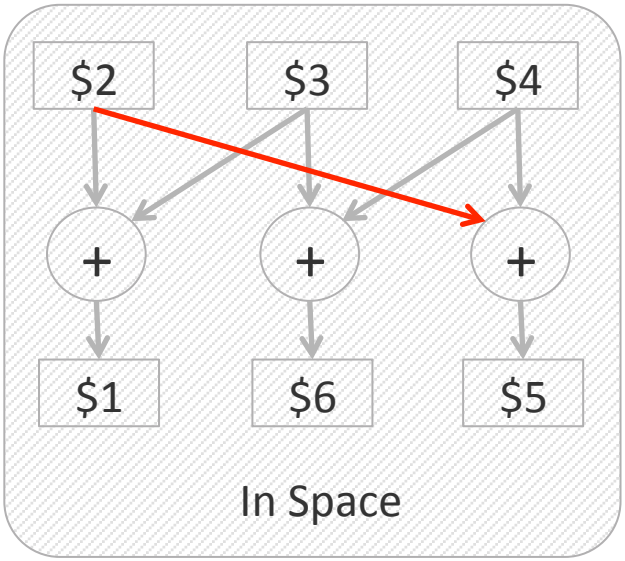
- Universal model but slow due to enforced sequencing
- Operations that can be executed in parallel are often serialized

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

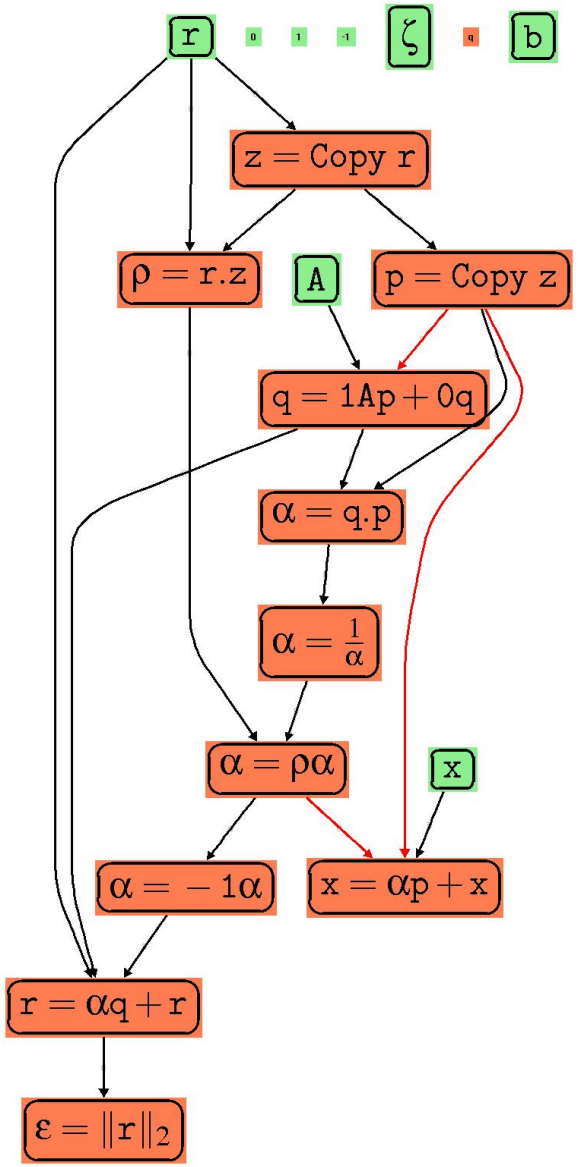
```
…
add $1,$2,$3
add $5,$4,$2
add $6,$4,$3
…
```

In Time





Spatial “recipe” is fast but larger in area



Conjugate Gradient method for solving Ax = b

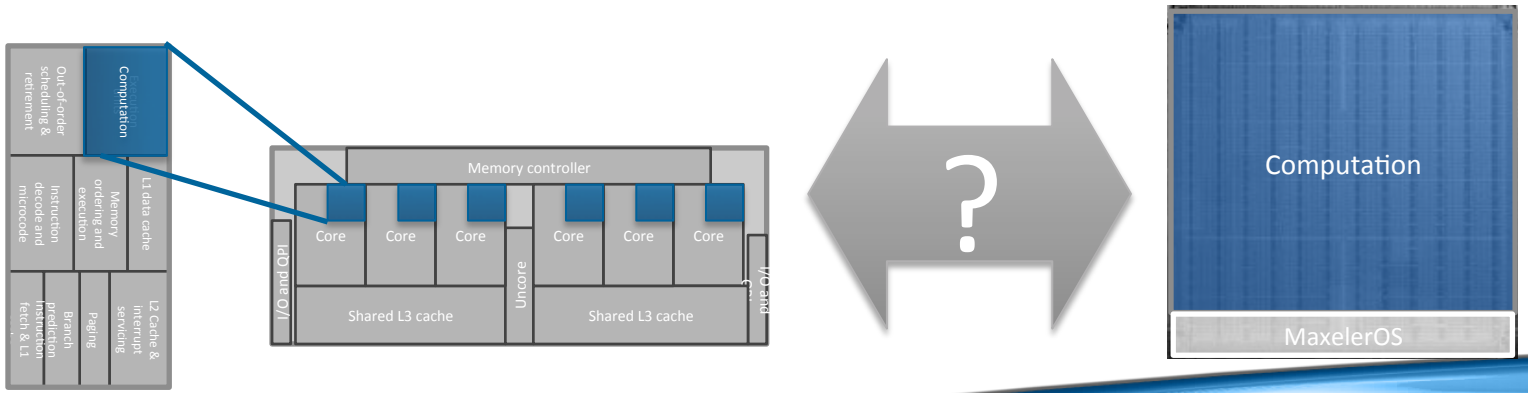
Multiple interconnected hardware computational units (some of them ALUs) can implement this bit more complicated function. The spatial version has 8 stages while the sequential (with 1 ALU) will need way too many iterations (even with a 10x faster speed)

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

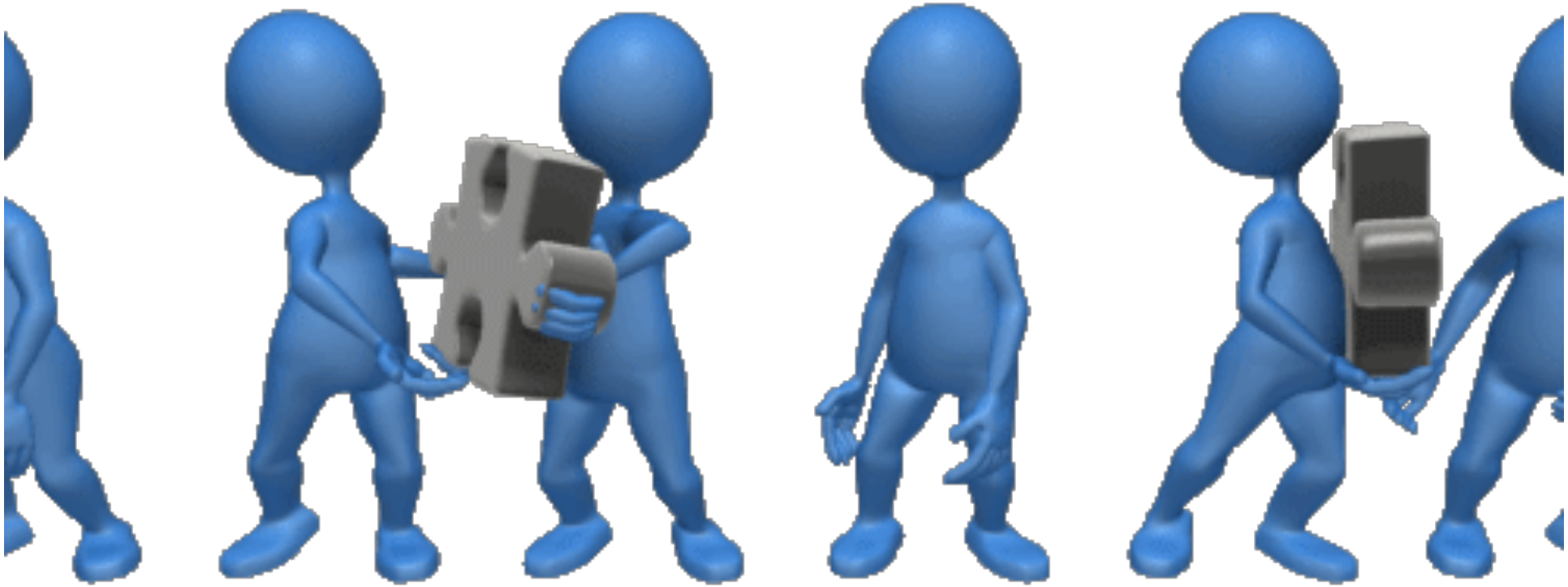
*DFG- Data Flow Graph

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
- Current temporal architectures are “compute sparse”, hence their spatial counterparts are at a win by being very “compute dense”



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

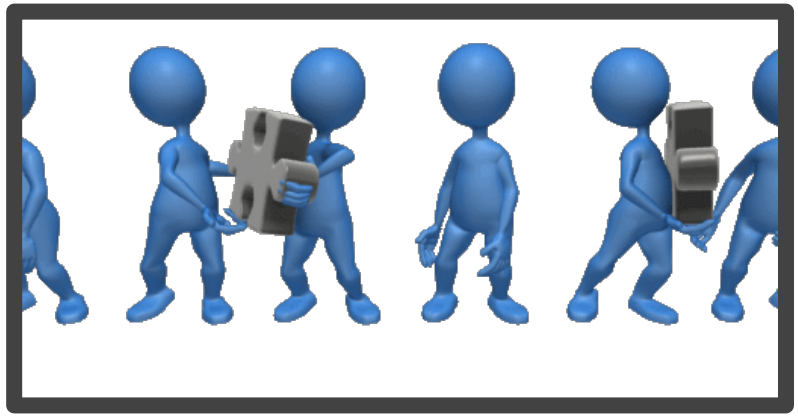


Courtesy of Bob Clapp, Stanford Geophysics

Control Flow versus Data Flow

CPUs

DFEs



Which one would you rather do now?

Control Flow versus Data Flow

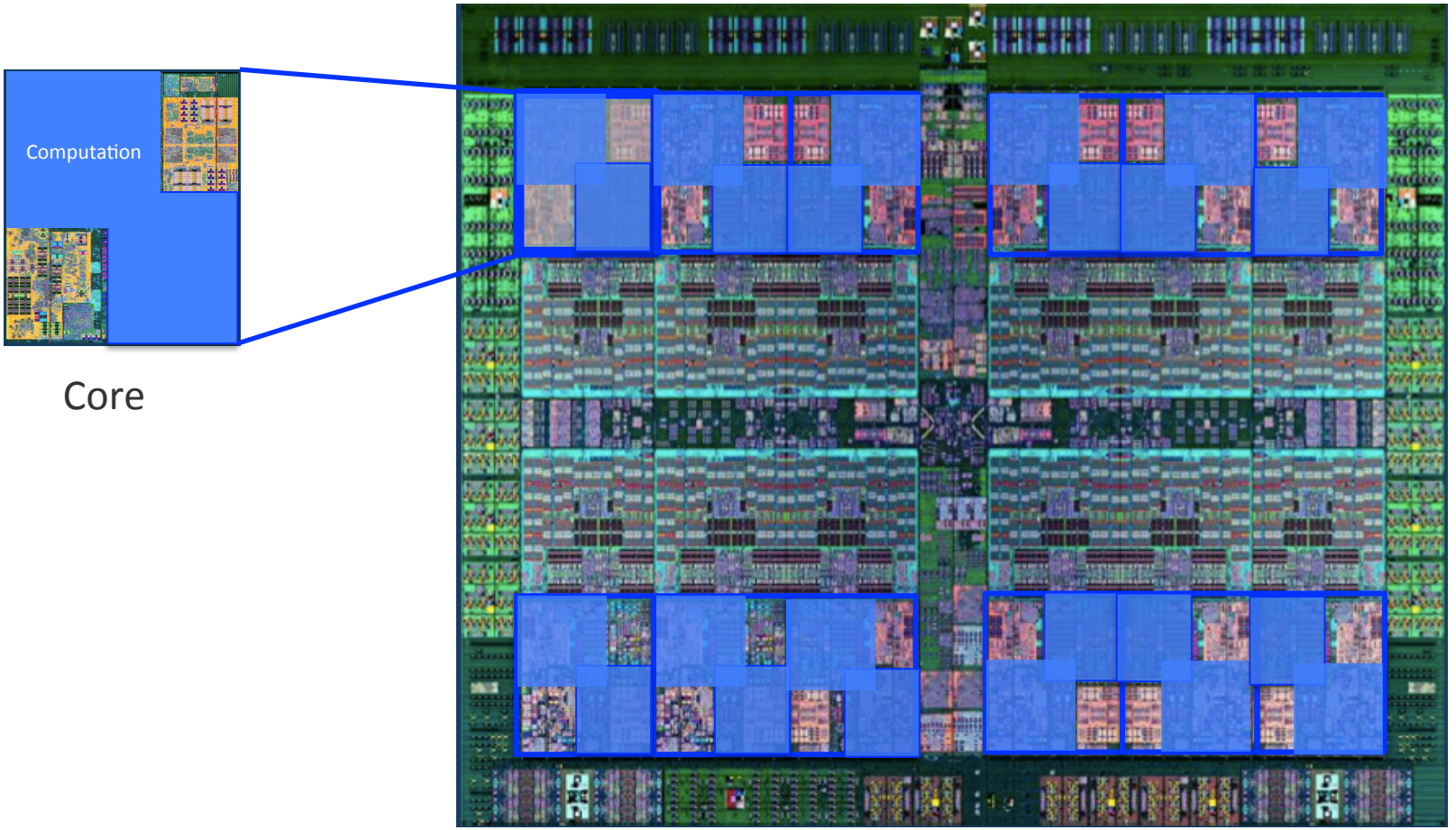
- Control Flow:
  - It is all about how instructions “move”
  - Data may move along with instructions (secondary issue)
  - Order of computation is the key

- 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

Control-flow Computing example: IBM POWER 8, 12 cores @ 4 GHz

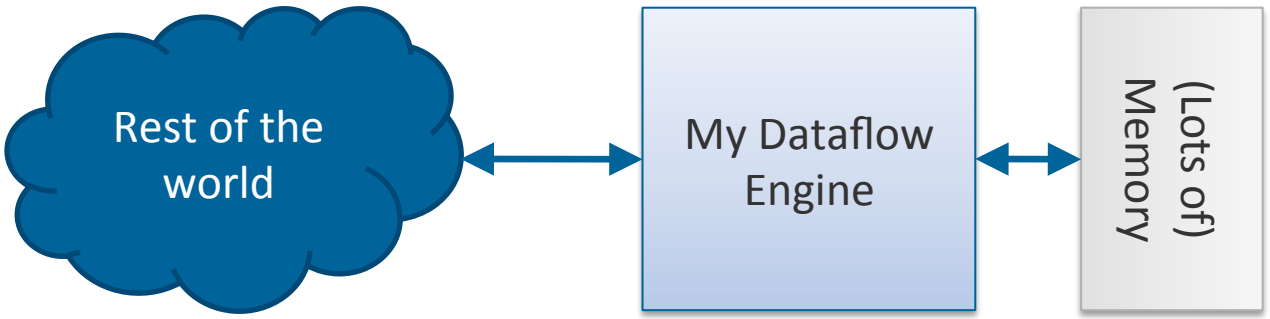


Core

22nm SOI, eDRAM, 15 ML 650mm2,12 cores (SMT8)

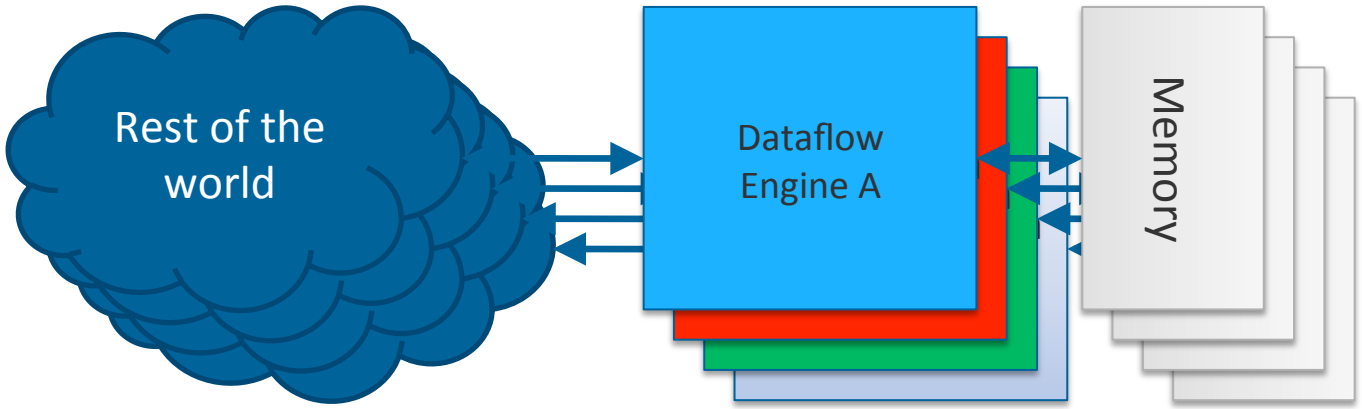
Dataflow Computing

- 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



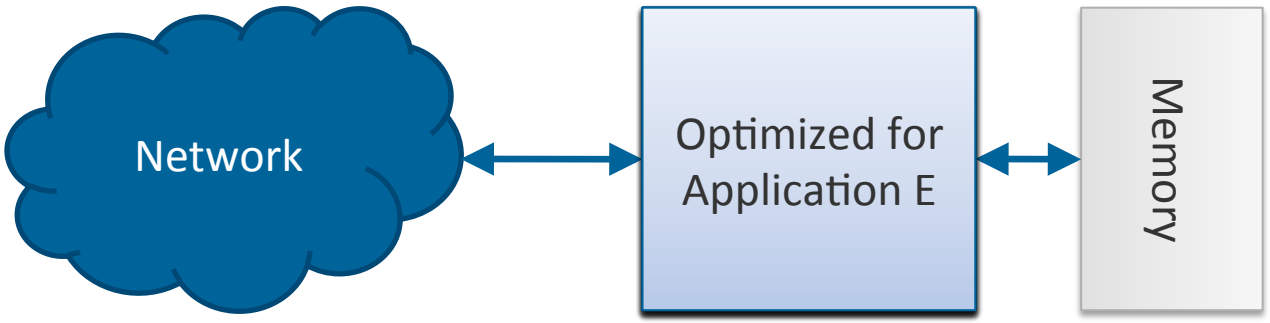
Special Purpose Computers

- 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

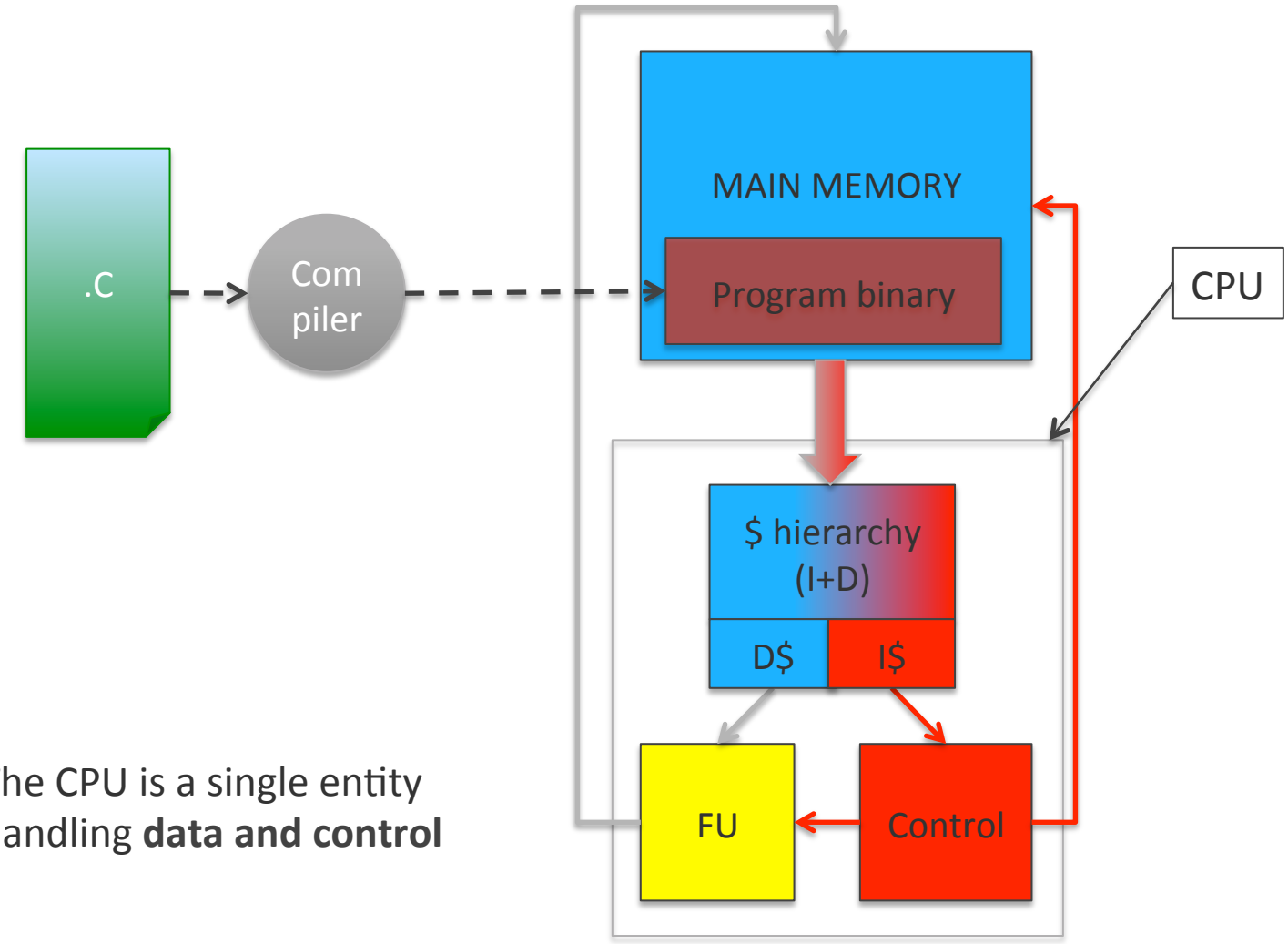


A Special Purpose Computer

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

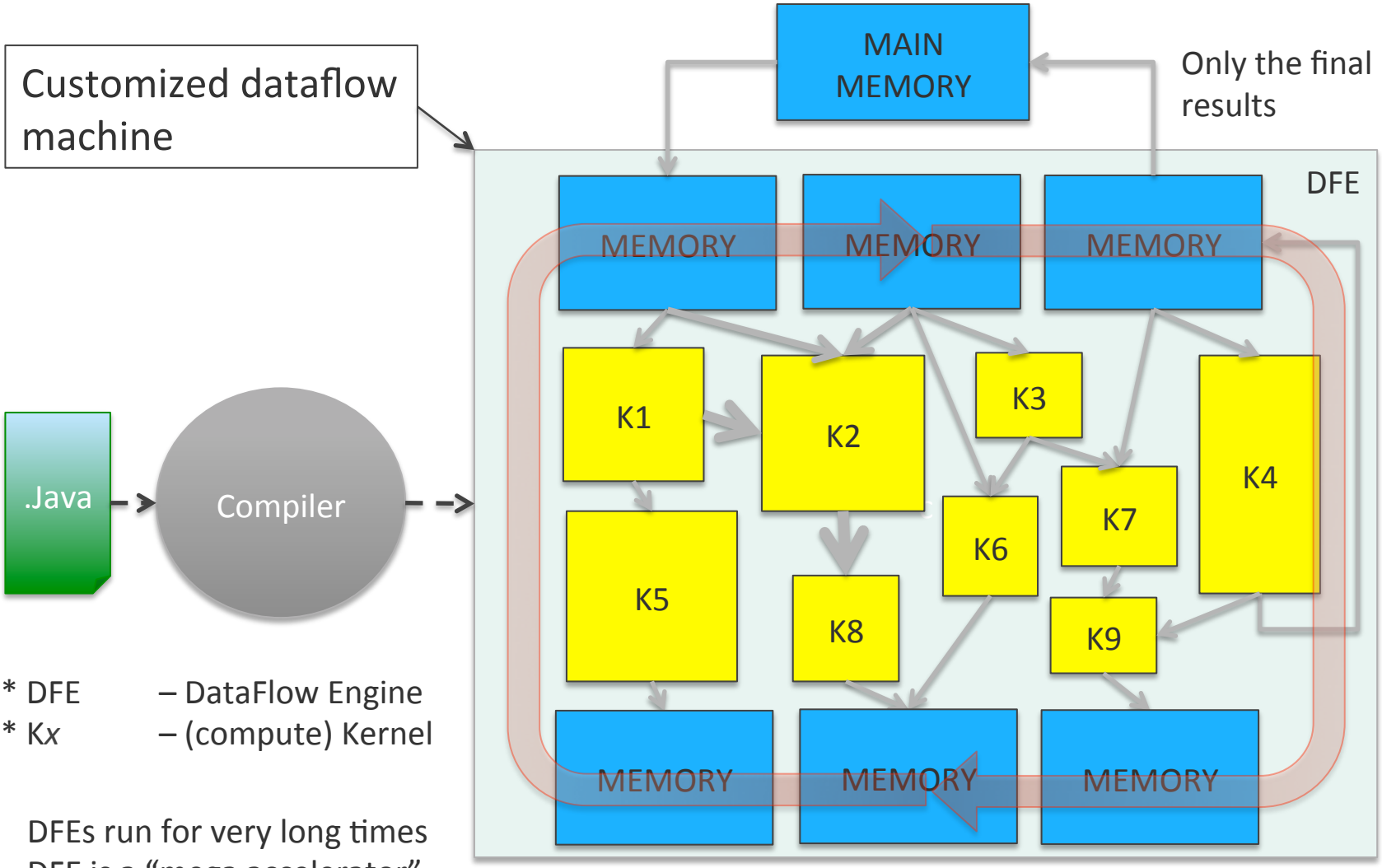


Control-flow Machine



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

Spatial Computing Machine



* DFE – DataFlow Engine
* K*x* – (compute) Kernel

DFEs run for very long times
DFE is a “mega accelerator”

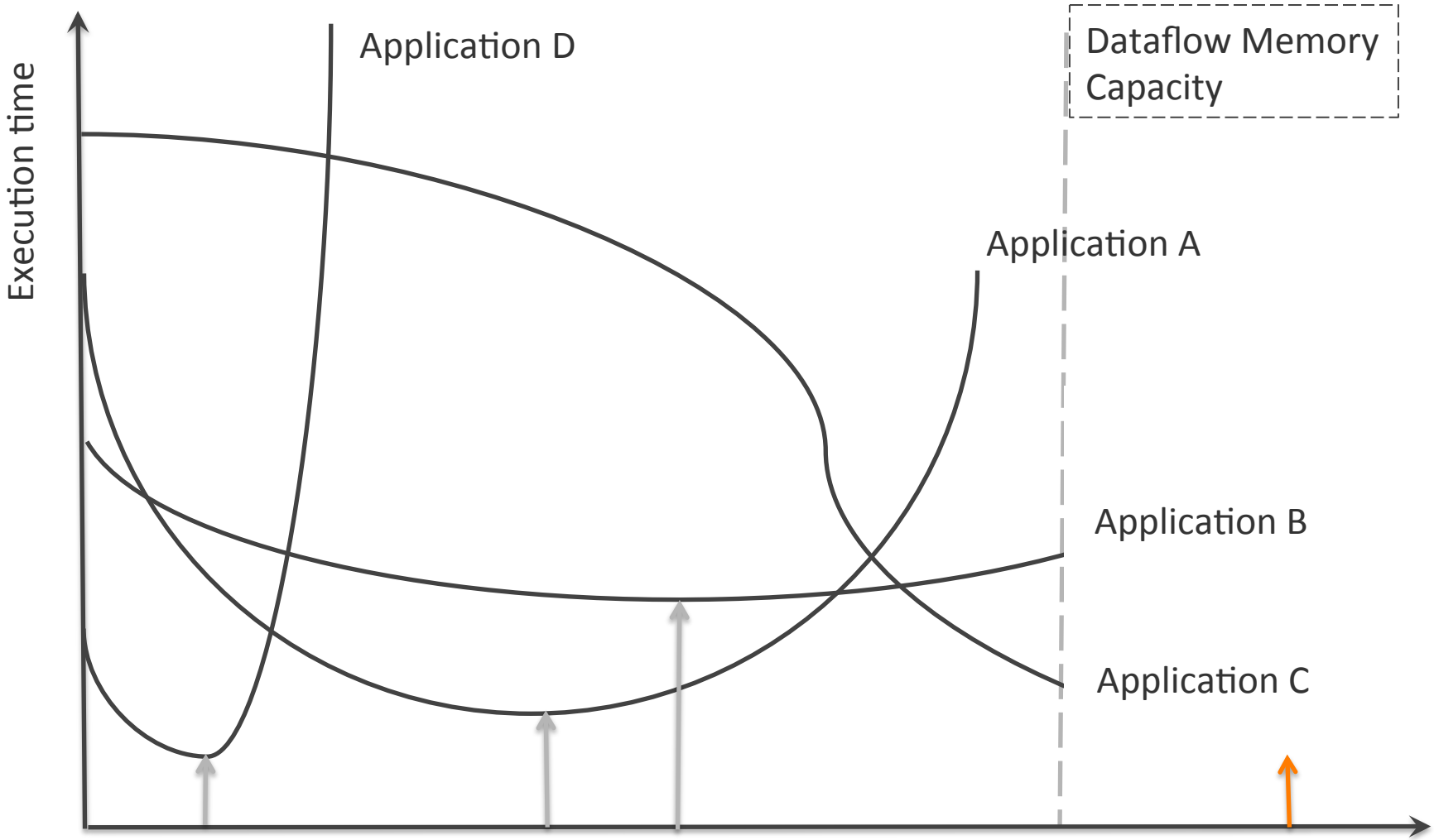
Accelerating Real Applications

- 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

Maximum Performance Computing



Each application requires different mix of control and dataflow

Amount of data placed* at the dataflow side

* placement is "intelligent" starting with the most intensively used data

Major Bottlenecks: Examples

	Throughput	Latency
Memory	Convolution	Graph algorithms
CPU	Monte Carlo	Optimization

Motivating Examples

```
for(int i=0;i<N;i++){
  a[i]=b[i];
}
```

is limited by: ……

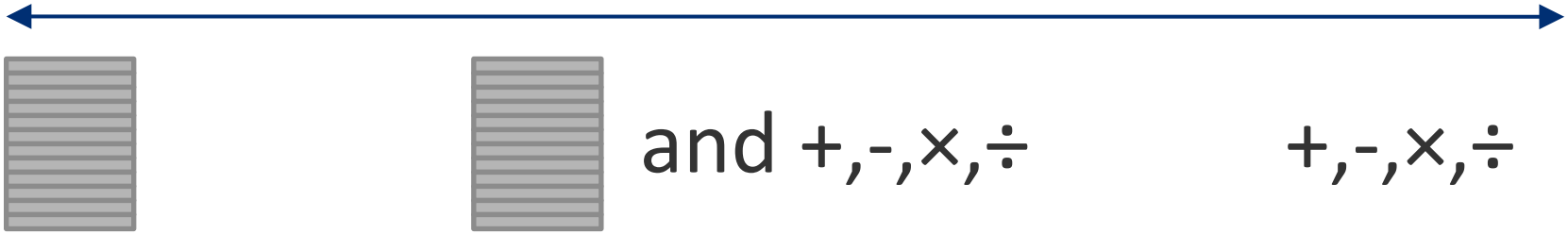
```
for(int i=0;i<N;i++){
  for(int j=0;j<1000;j++){
     a[i]=a[i]+j;
  }
}
```

is limited by: ……

Trading memory for compute resources

Computing f(x) in the range [a,b] with $|E| \leq 2^{-n}$

Table *Table+Arithmetic* *Arithmetic*



- uniform vs non-uniform
- number of table entries
- how many coefficients

- polynomial or rational approx
- continued fractions
- multi-partite tables

Underlying hardware/technology changes the optimum

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 should be balanced