Hardware Compilation in Hardware: Enabling Integration of Reconfigurable Fabrics and Processors

Benjamin A. Levine

IBM/SRC Fellow
Carnegie Mellon University

Outline

- Motivation for reconfigurable fabrics:
  - Changing semiconductor technologies.
  - Limitations of conventional architectures.
- Problems with hybrid reconfigurable HW.
- Single-executable fabrics and processors.
- Hardware compilation in hardware.
- Queue- and register-based ISAs.
- Results and future work.

New Process Technologies Give Us…

- **More transistors ⇨ *more complex designs***
  - Exploding design and verification costs.
  - Custom design increasingly expensive.
- **Faster transistors ⇨ *slower interconnect***
  - Meeting timing difficult, especially with long wires and large global structures.
- **Smaller transistors ⇨ *difficult fabrication***
  - Need more careful circuit design and layout.
  - Designs must consider manufacturability.

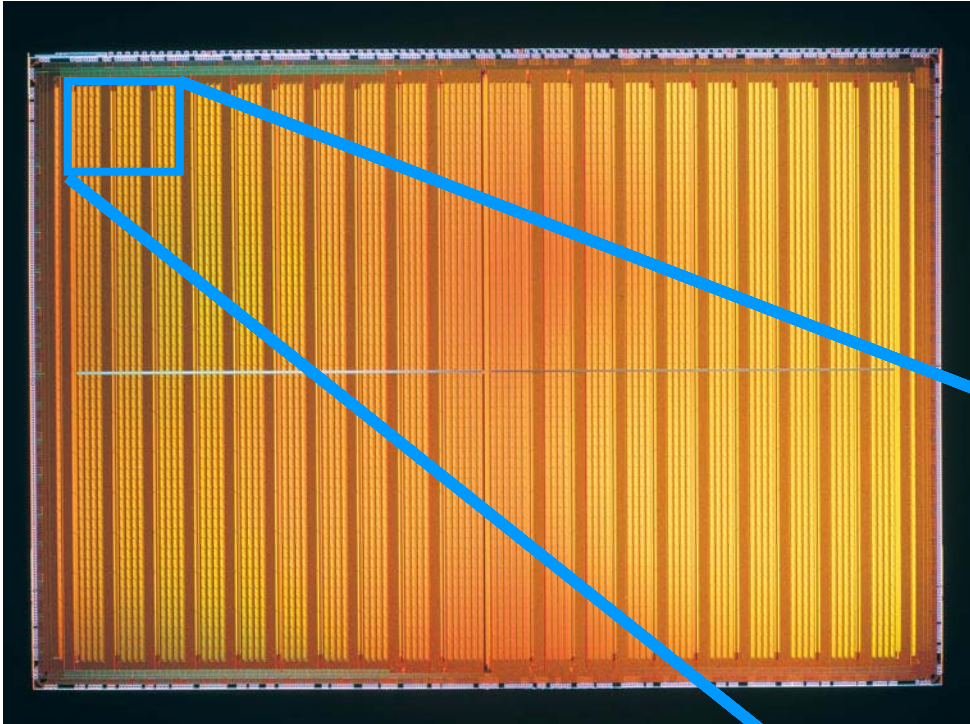
⇨ ***We need more and can afford less custom design.***

Part of the Solution: REUSE

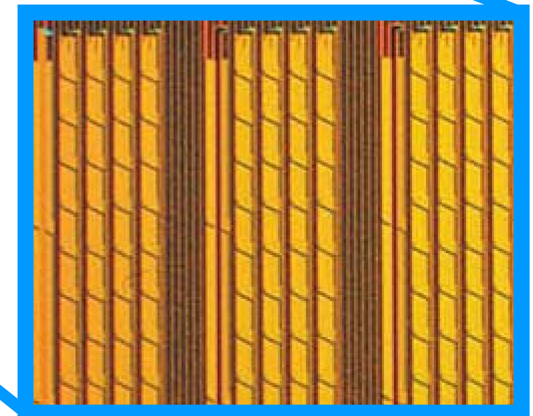
Reuse design effort on all levels!

- **Component reuse:**
  - Design using repeating, tiled blocks.
  - Use design regularity to multiply design effort.
- **Chip reuse:**
  - Create chips that can be used in many products over many product generations.
- **Software Reuse:**
  - Implement systems using programmable devices and reuse software for new systems. Software should be scalable and portable.

Component Reuse Example - FPGA



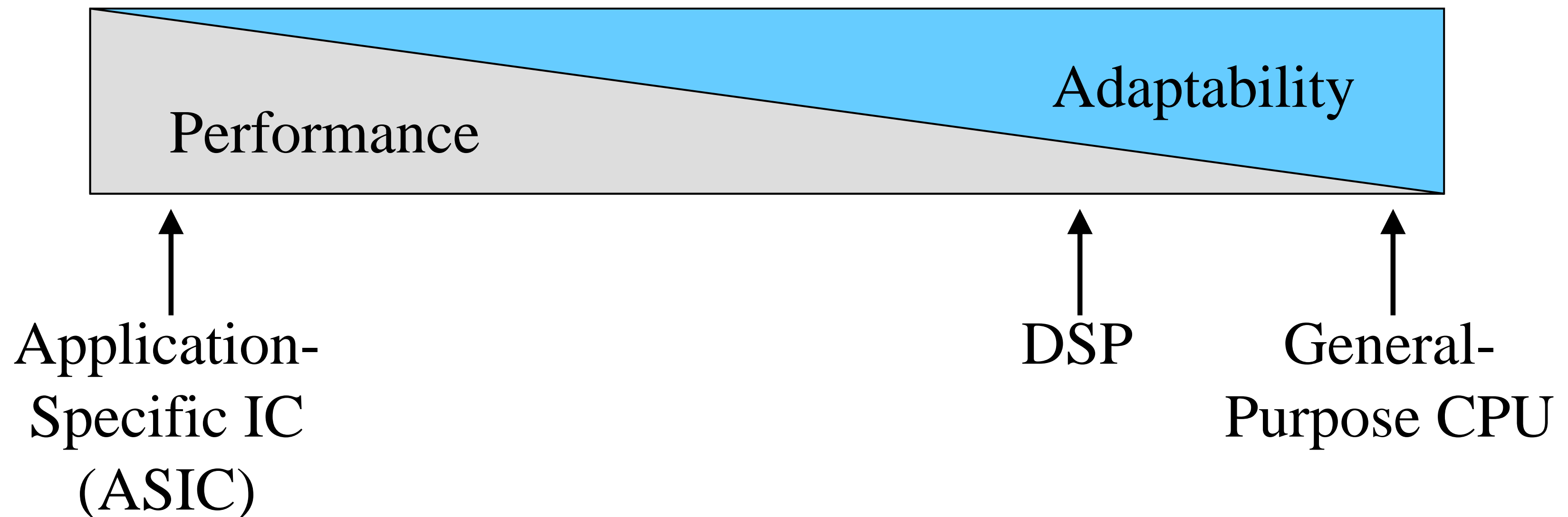
Photos: Xilinx Corporation



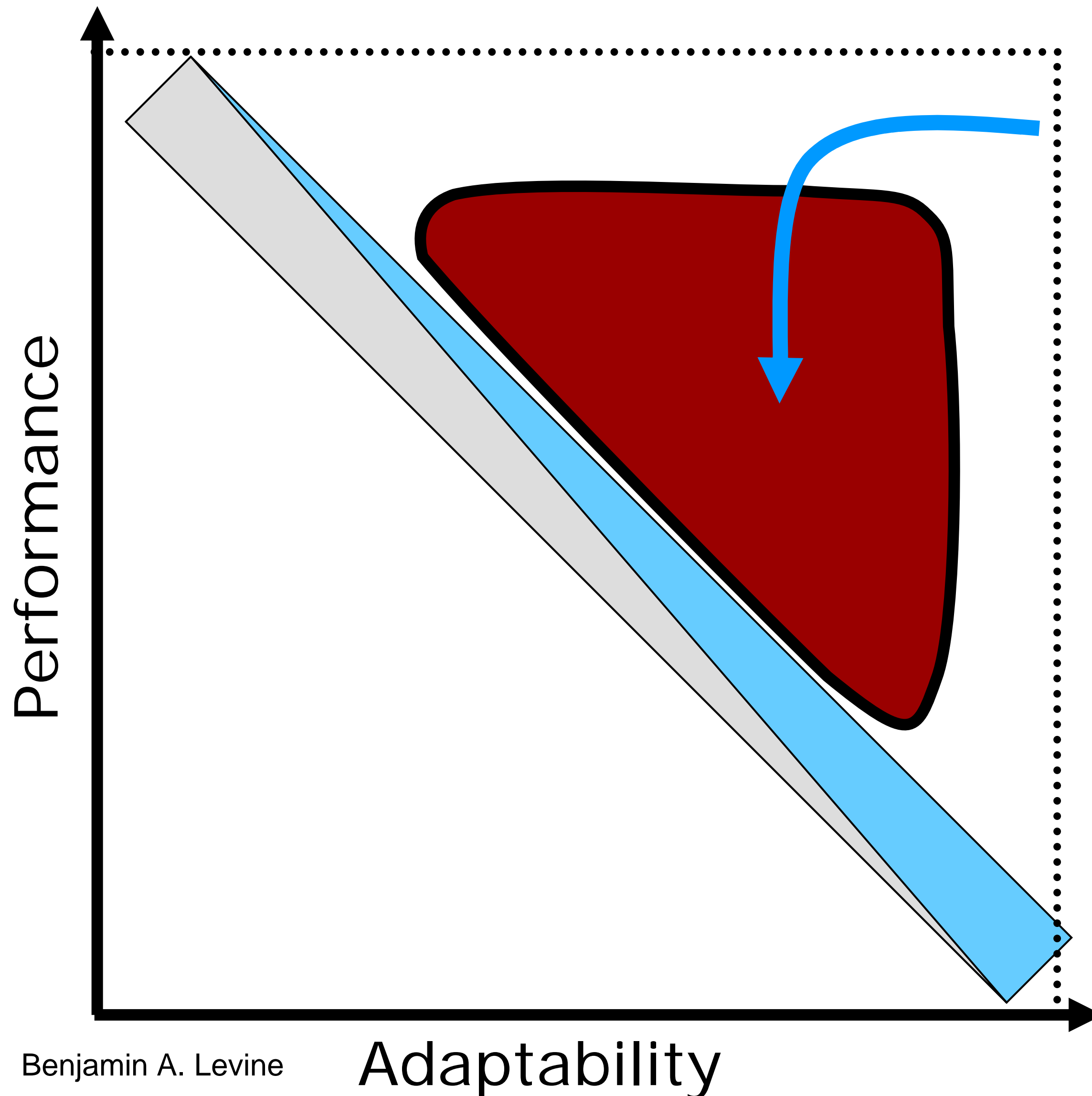
Xilinx Virtex Die
- **Fabric of identical CLBs**
- **Repeated interconnect**

Limitations on Chip Reuse

- Conventional chips exhibit tradeoff between adaptability and performance.
- If you need high-performance, you must limit adaptability (right?).



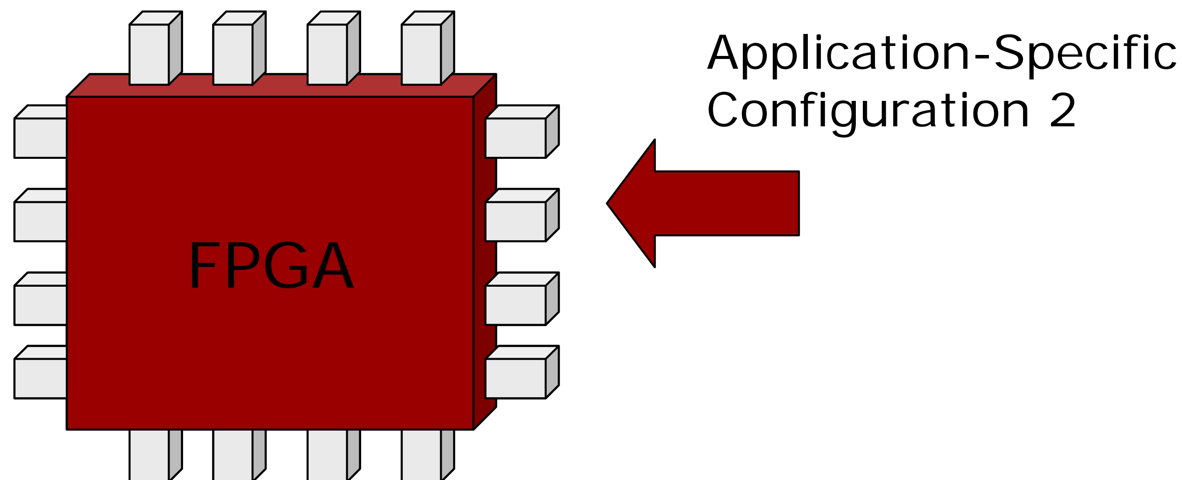
Chip Reuse Requires Both!



To enable more chip reuse, we need both high performance and high adaptability.

Reconfigurable Computing

Use programmable hardware that lets you *reconfigure* a fabric of logic and interconnect to give high **performance** for a specific application and **adaptability** for different applications.



Reconfigurable Computing Fabrics

Enable reuse:

☑ **Component Level**

- Most RC architectures can be built from small repeating blocks or tiles.

☑ **Chip Level**

- Reuse hardware for many products.

☒ **Software Level**

- Software tied to specific hardware instance. No scalability or portability.

Pipelined Reconfigurable Fabric

CMU PipeRench

A programmable, pipelined data path containing:

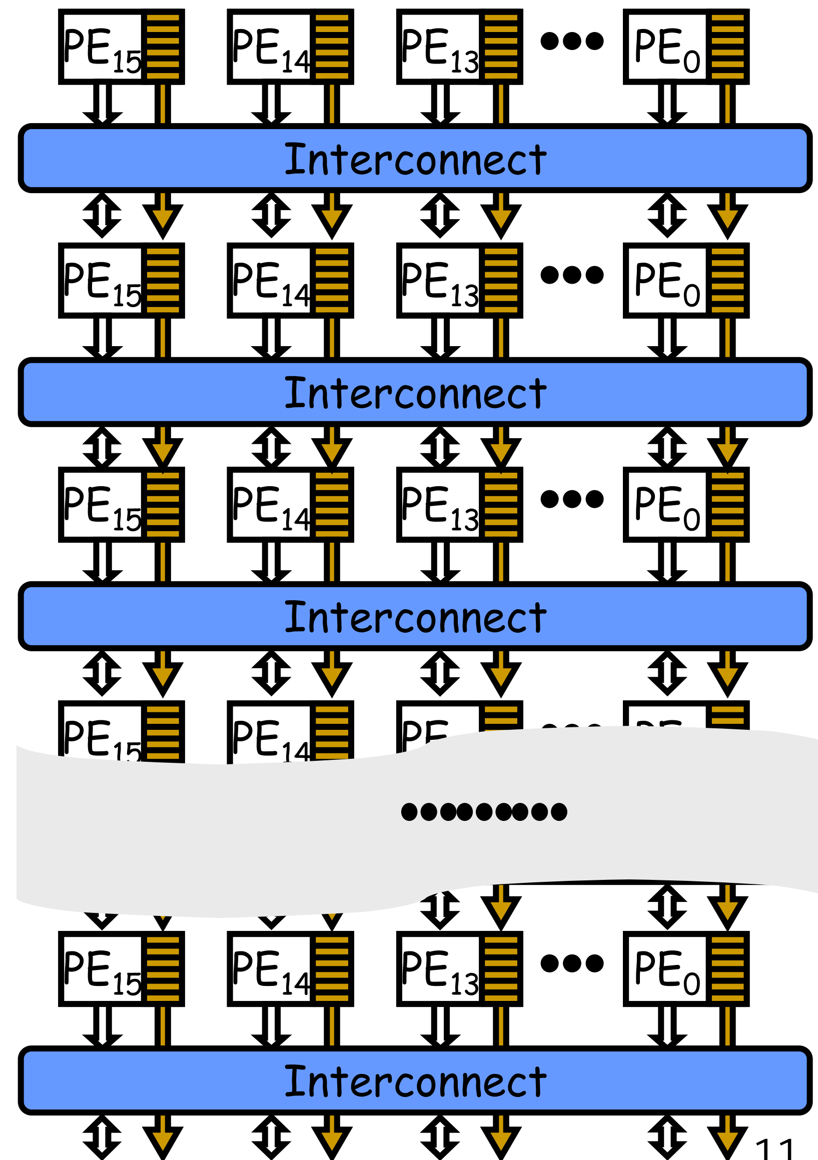
- Processing elements
- Local interconnect
- Pass Registers
- Unbounded Depth:

Physical hardware time-multiplexed at run-time to emulate pipeline of any depth.

⇨ Helps enable SW reuse.

Reconfigurable Fabric Benefits for DSM

- Uses local interconnect and does not require large global structures.
- Enables Reuse:
  - Reusable design elements - PEs, interconnect, memory.
  - Reuse fabrics on many chips.
  - Reuse chips for many applications by reconfiguring.
- Tiled structure advantageous for manufacturing, testing, verification, redundancy.

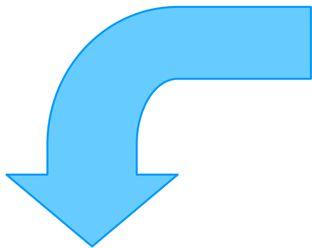


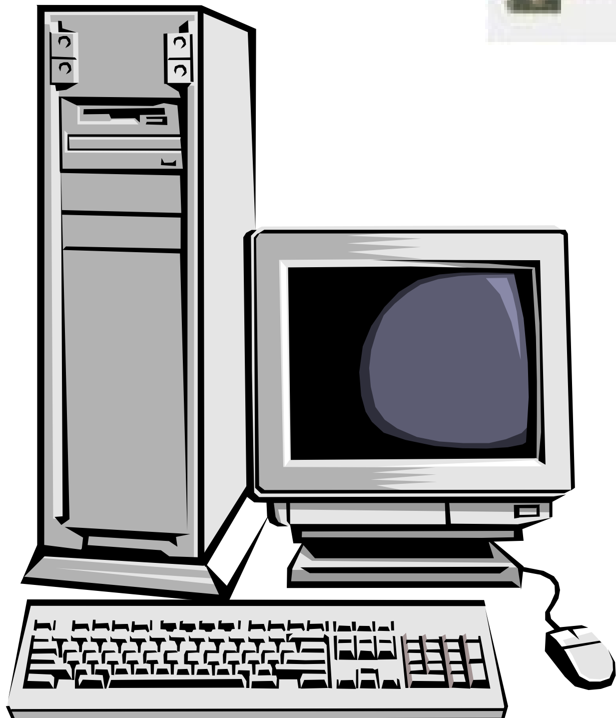
Problems With Reconfigurable Fabrics

- No computationally complete programming model.
- Inefficient for applications with lots of control flow.

Usual solution is a “Hybrid Architecture” - reconfigurable fabric combined with general purpose CPU.

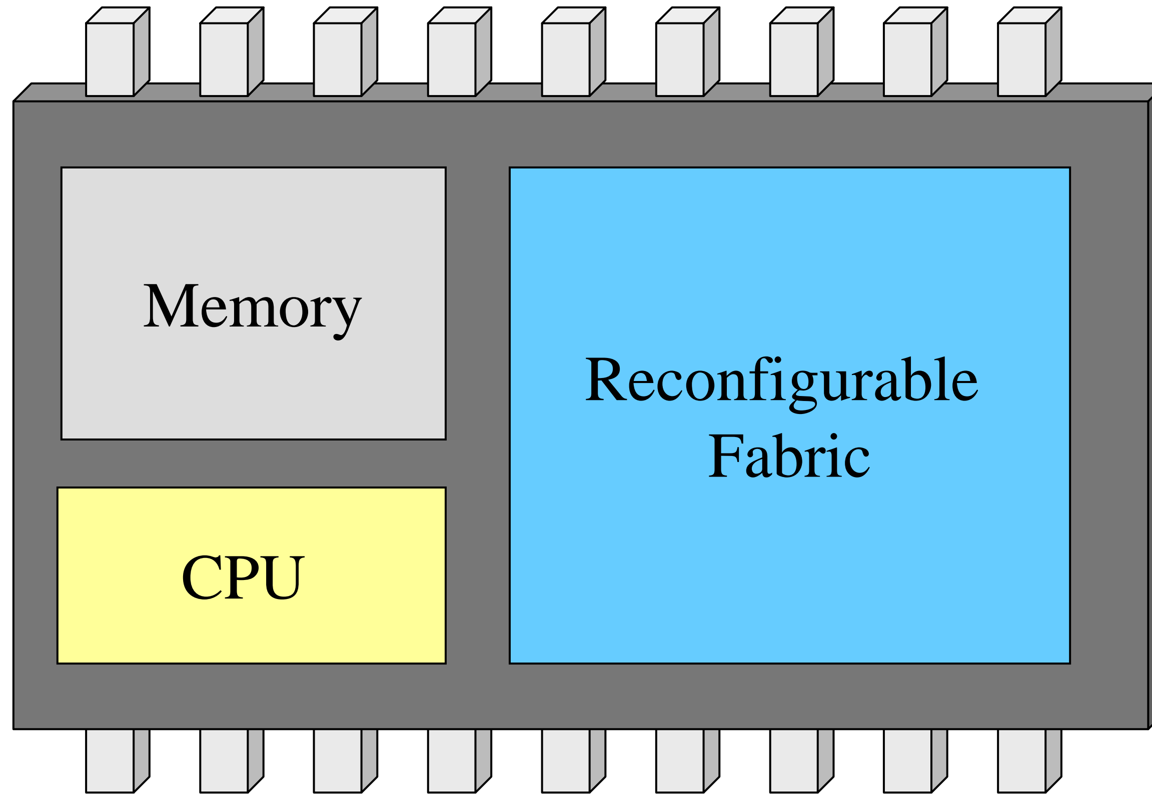
Hybrid Architecture I





Reconfigurable fabric as past of coprocessor board in standard workstation.

Hybrid Architecture II



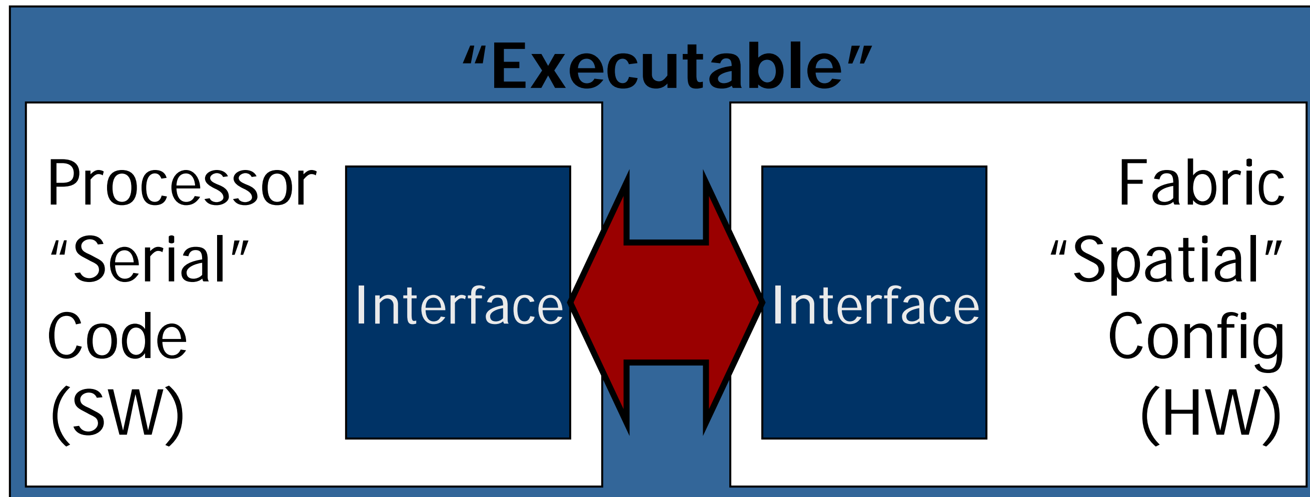
Reconfigurable fabric as part of system on a chip.

I.e., Xilinx Virtex II Pro.

Problem With Hybrid Architectures

Executables are heterogeneous

- Serial instruction sequence for the processor.
- Spatial configuration information for the fabric (placement and routing)



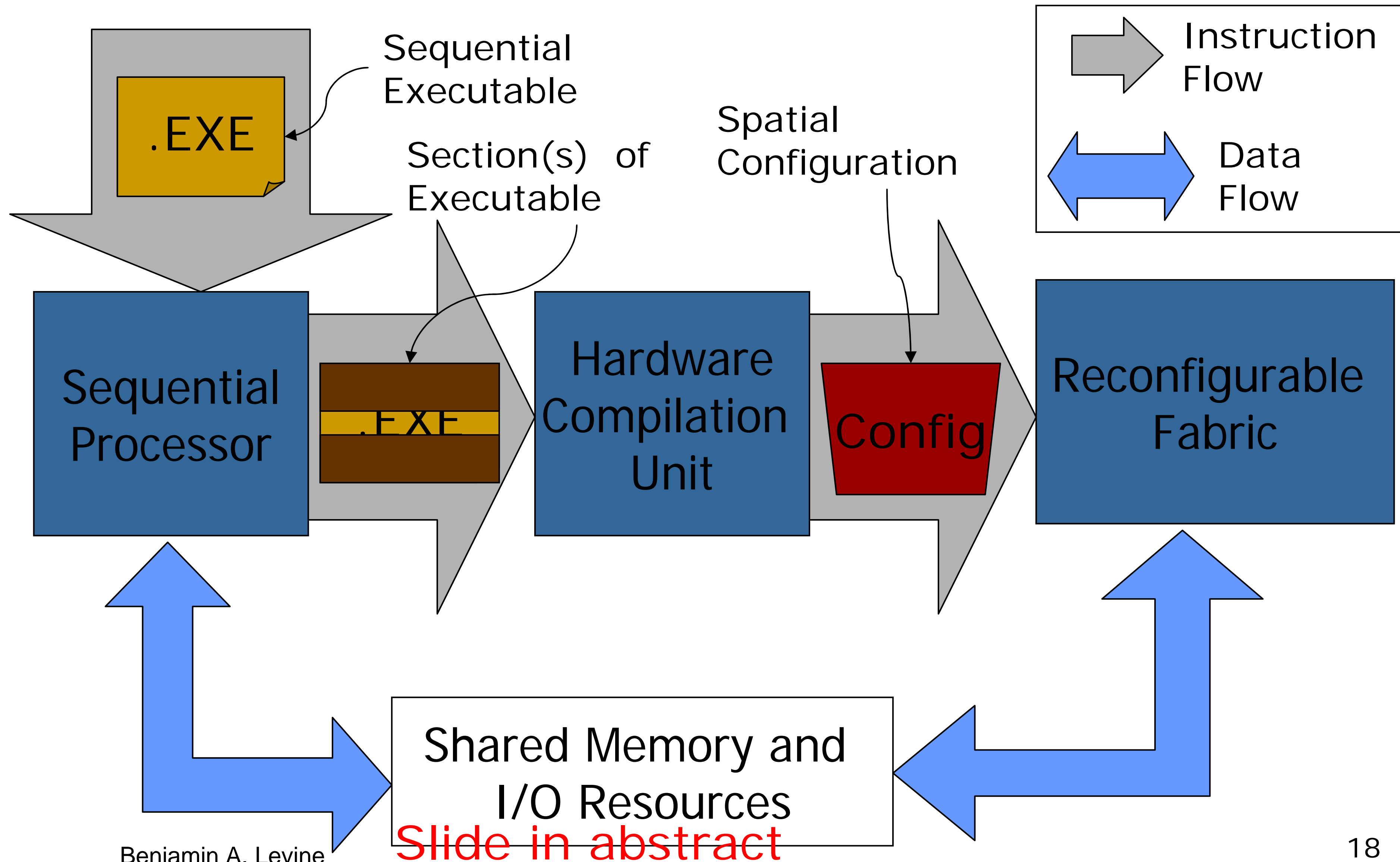
Problems with Heterogeneity

- Now two programming tasks.
- Designing interface software and hardware is hard.
- To set a standard platform requires:
  - Fixed Spatial Architecture (Fabric)
  - Fixed Serial Architecture (CPU)
  - Fixed Interfaces
- Executables tied to one platform
  - Limits portability
  - Limits performance scaling

A Better Way?

- Have a single executable.
- Execute it on the processor.
- Generate spatial configuration for fabric in HW at runtime: **"Hardware compilation in hardware"**
- Pass control to reconfigurable fabric.
- Return to serial execution when complete.

Single-Executable Hybrid Architecture



Example Code Fragment

```
if (x > 10) x = 10;
x = x * 4;
```

Sequential Code

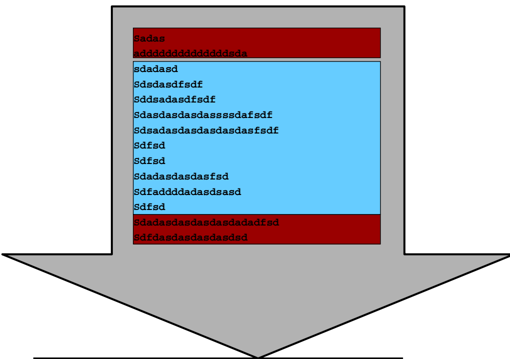
```
for (i = 0; i < x; i +=4) {
   a = A[i]; b=A[i+1];
   c = A[i+2]; d = A[i+3];
   e = a + b; f = a - b;
   g = c + d; h = c - d;
   w = e + g; x = f + h;
   y = e - g; z = f - h;
   B[i]   = w; B[i+1] = x;
   B[i+2] = y; B[i+3] = z;
}
```

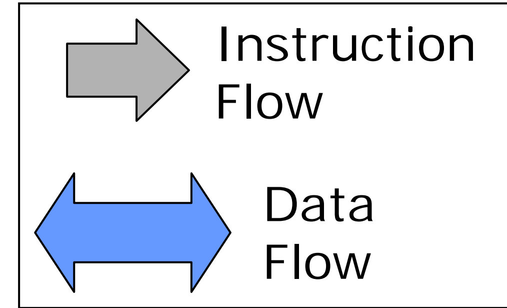
Parallelizable Loop Body

```
Z = x ** 4;
If (Z == 0) Z = 100;
```

Sequential Code

Pass Single Executable to Processor





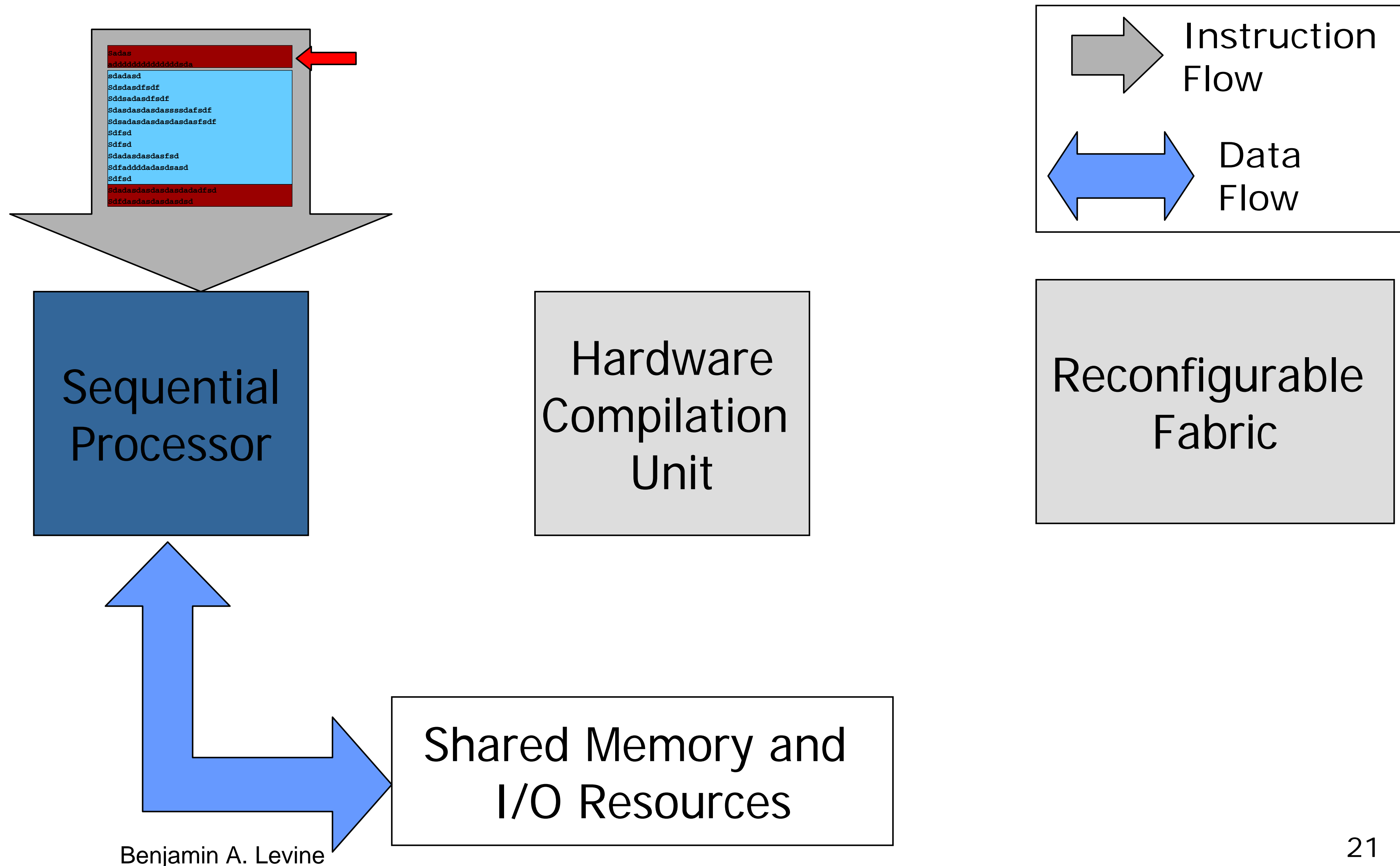
Sequential Processor

Hardware Compilation Unit

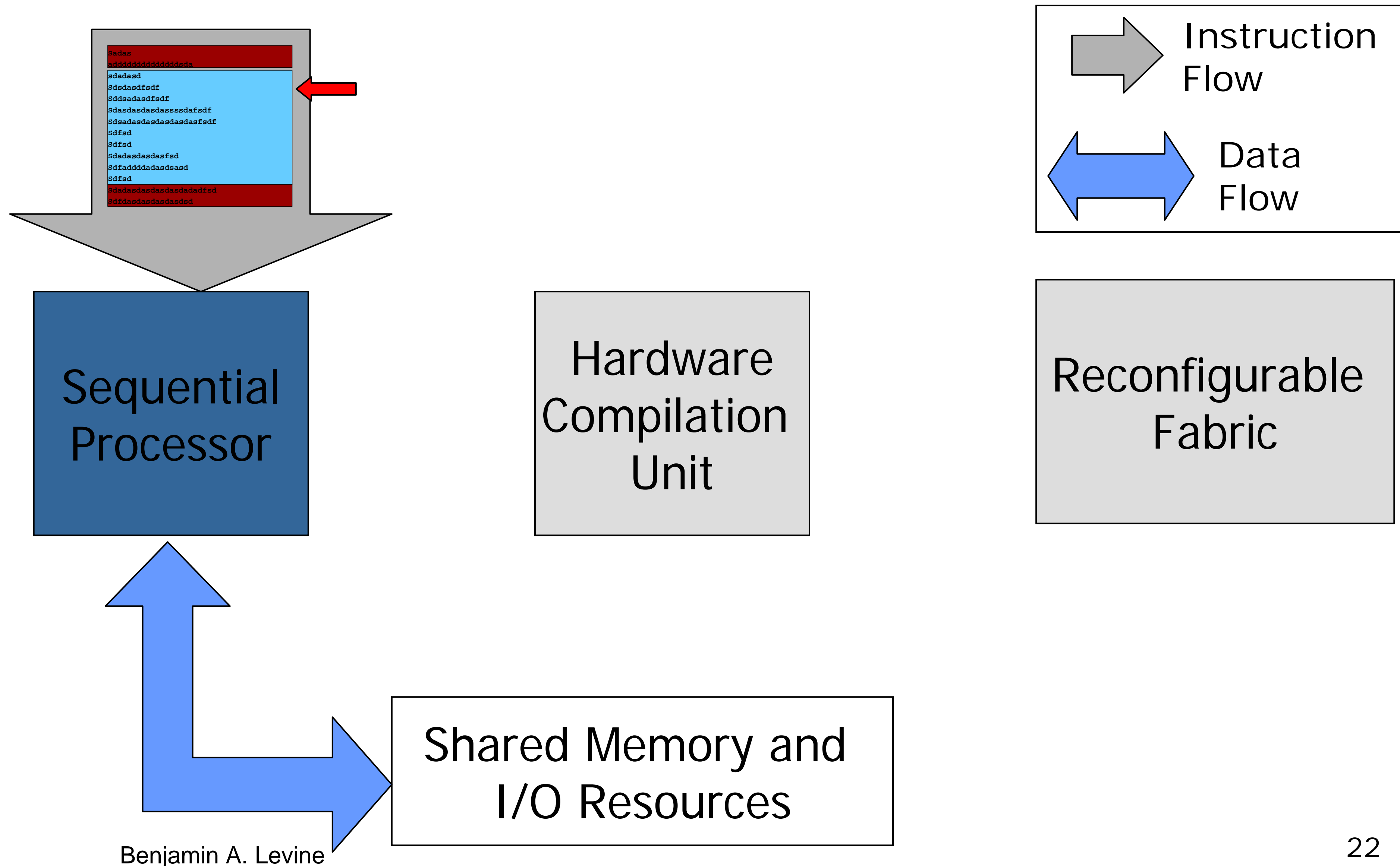
Reconfigurable Fabric

Shared Memory and I/O Resources

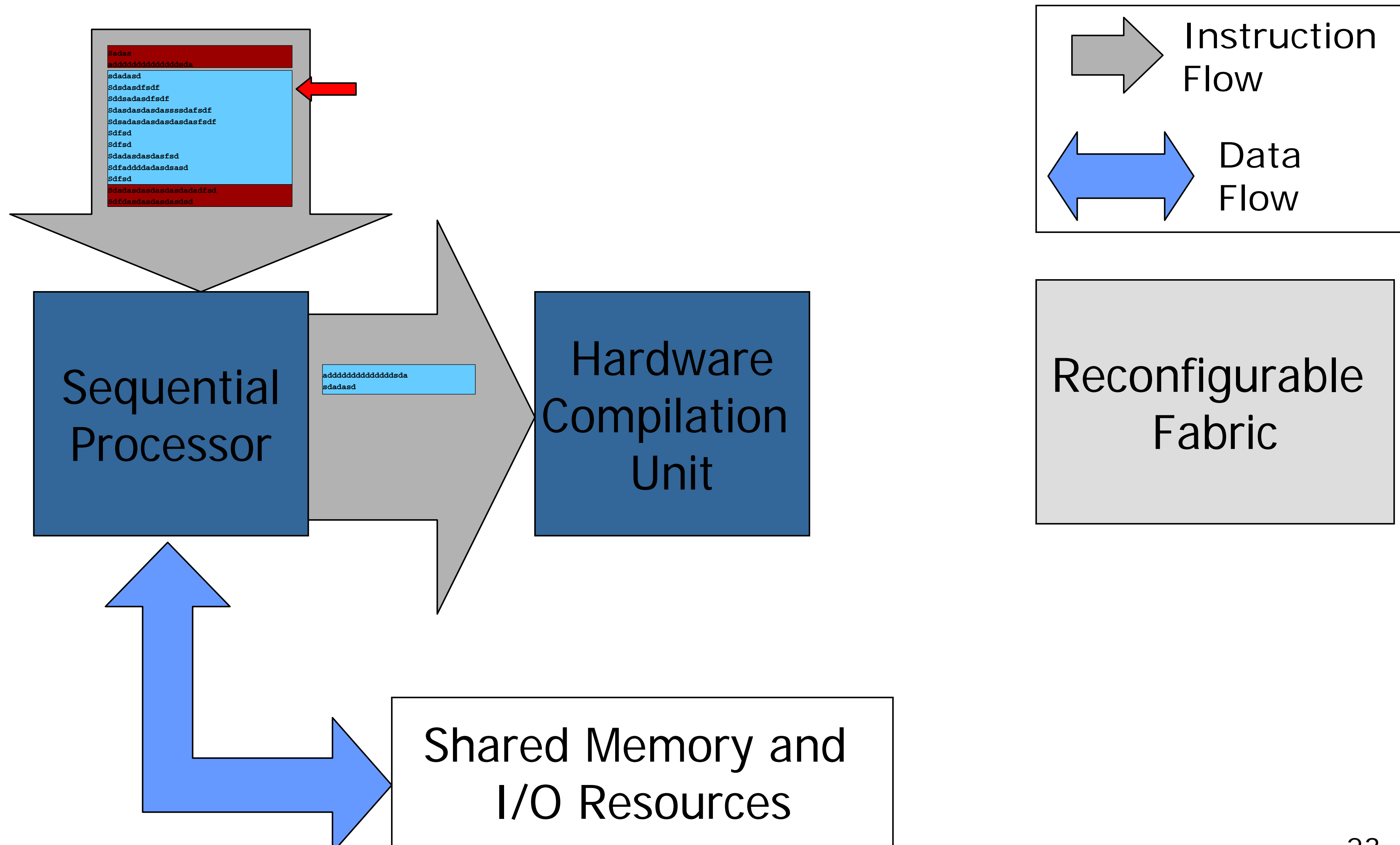
Run Sequential Code



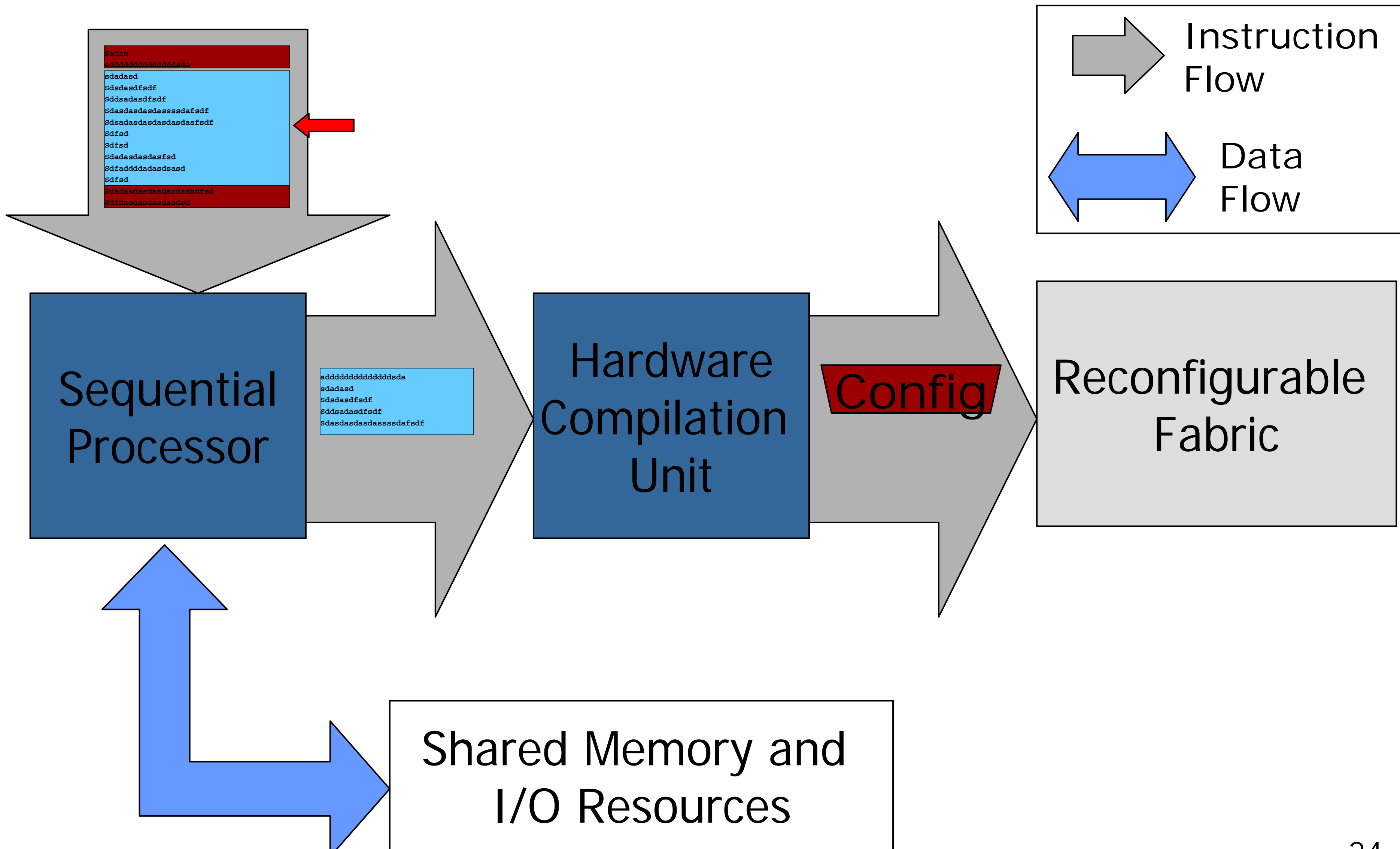
Start First Loop Iteration In Processor



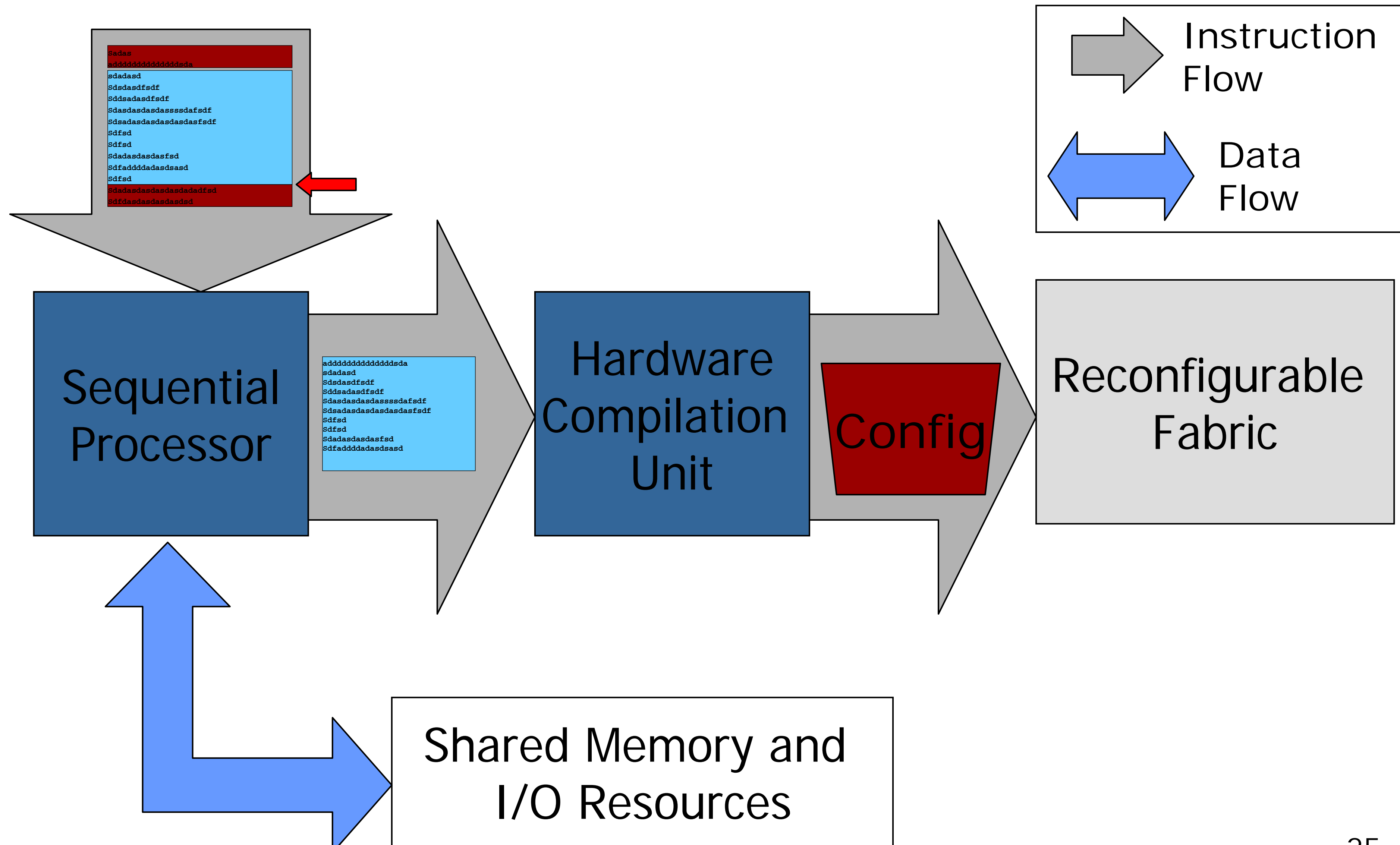
Code Passed to HCU While Running



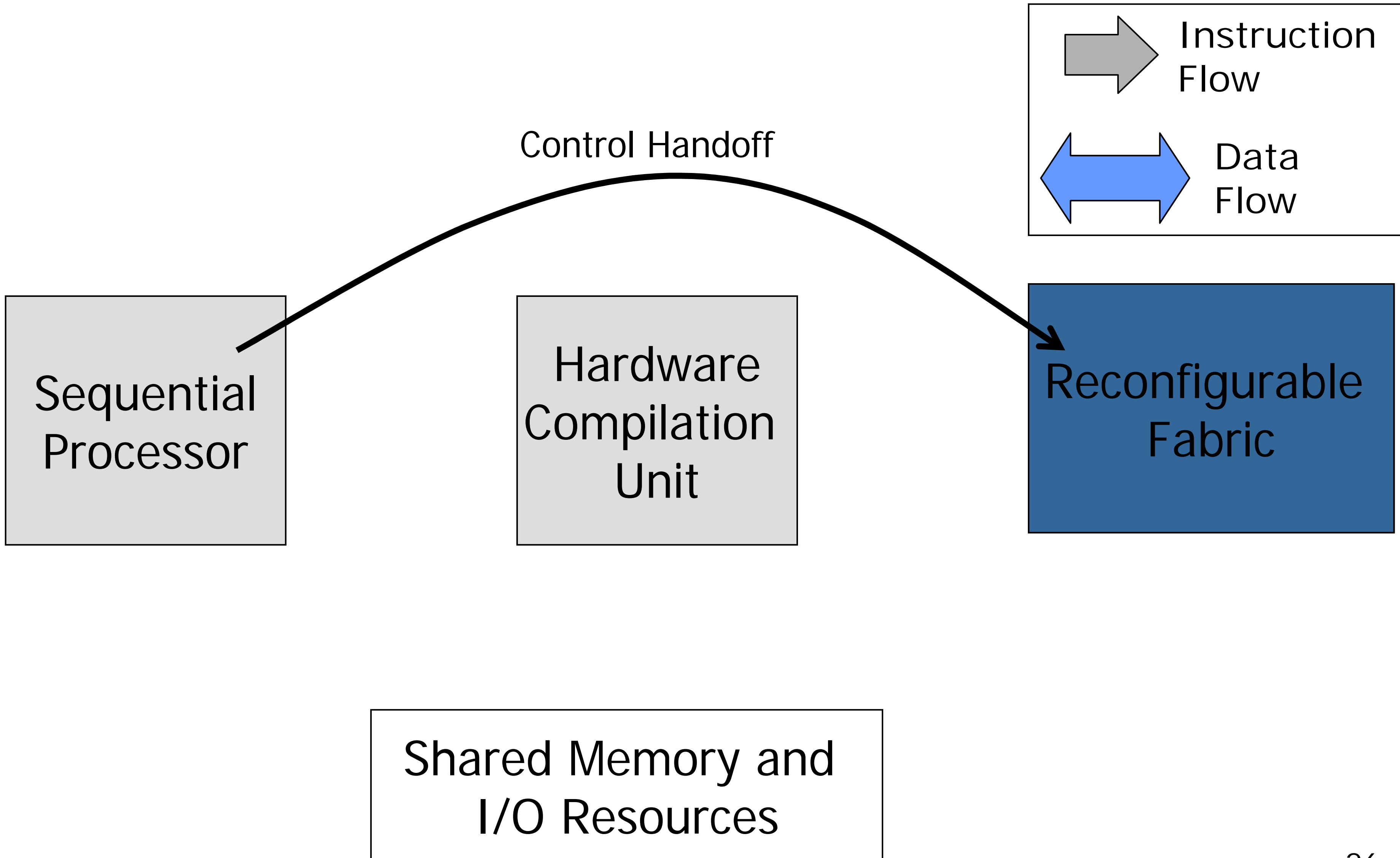
HCU Creates Fabric Configuration



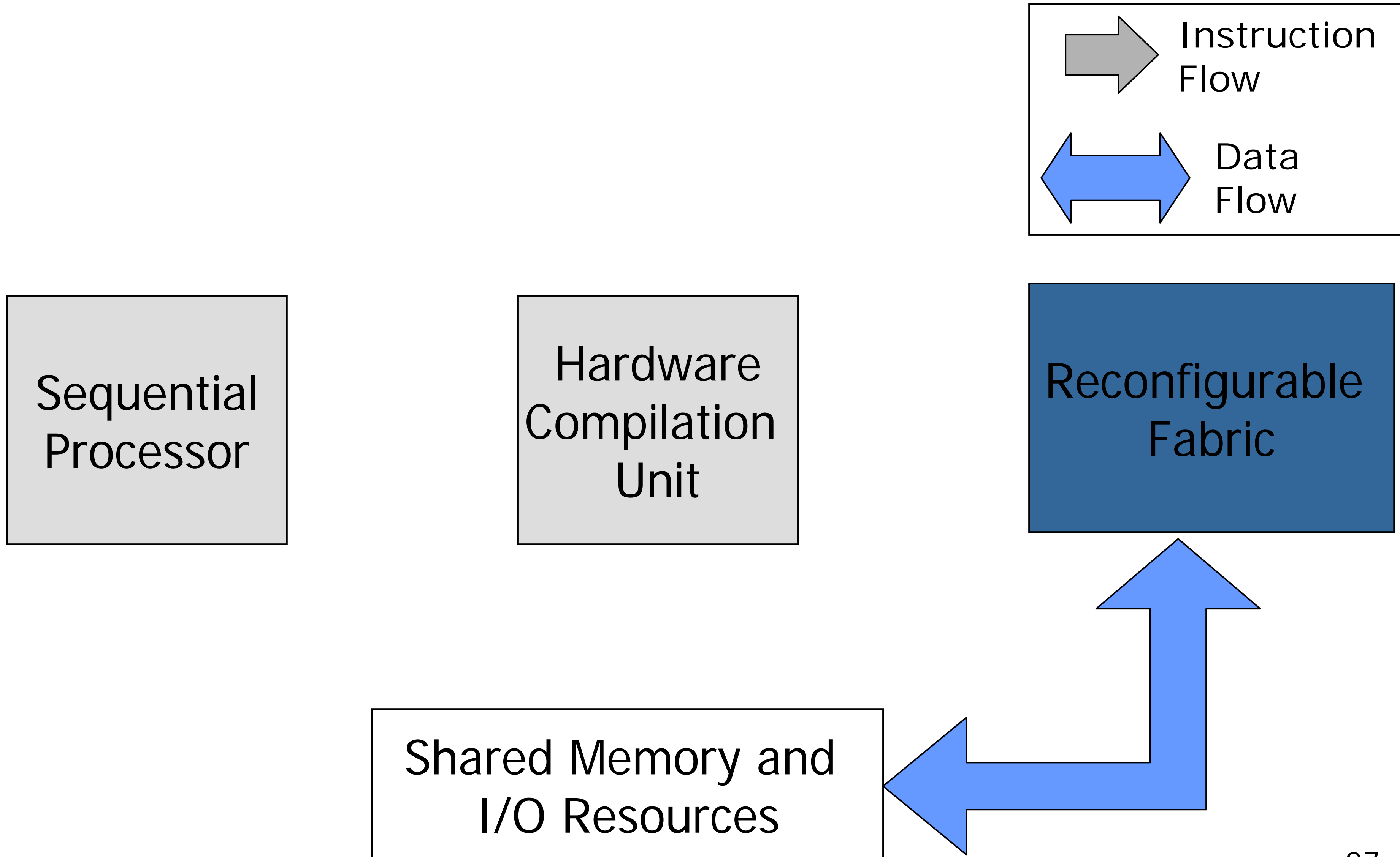
HCU Done After One Loop Iteration



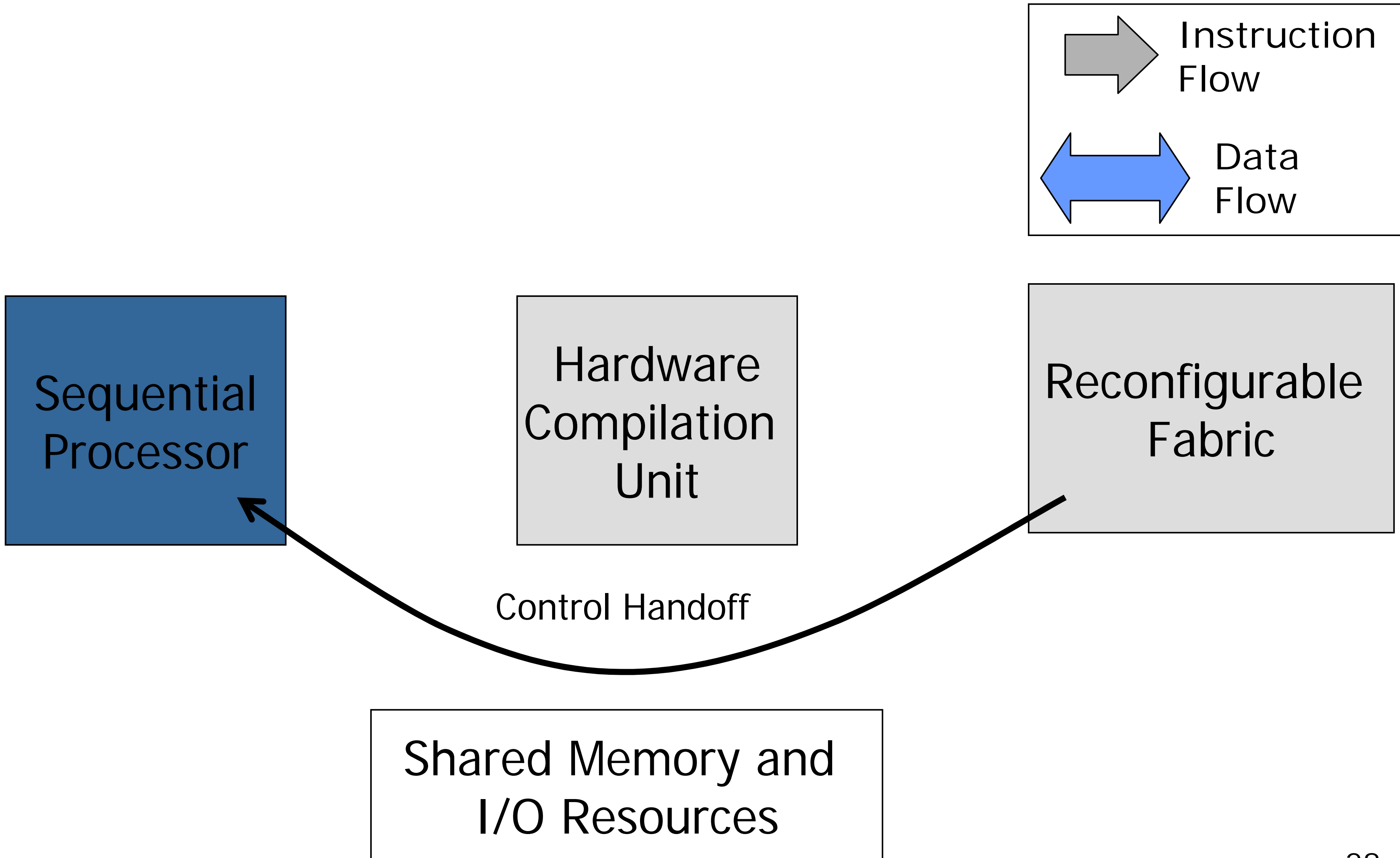
Pass Control to Fabric



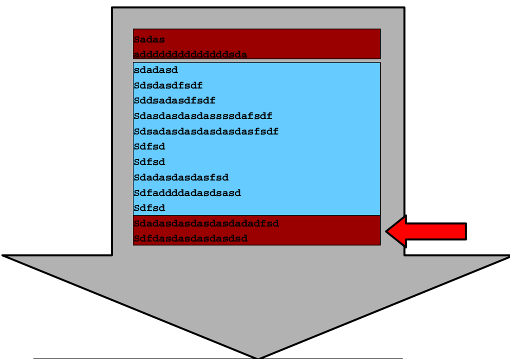
Run Many Loop Iterations in Fabric

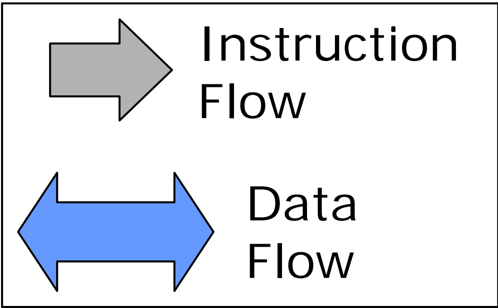


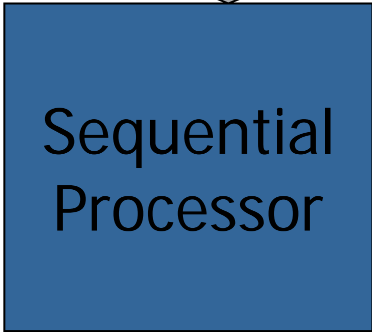
Pass Control Back to Processor

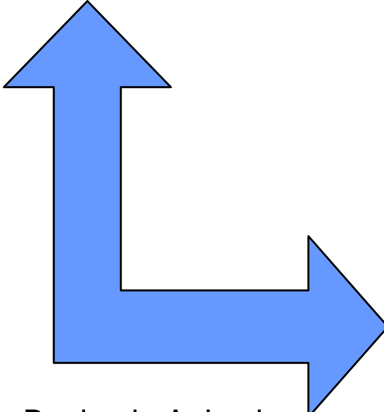


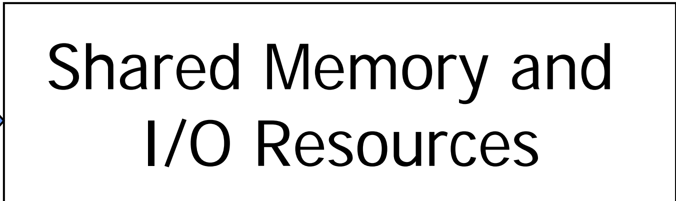
Run Remaining Sequential Code



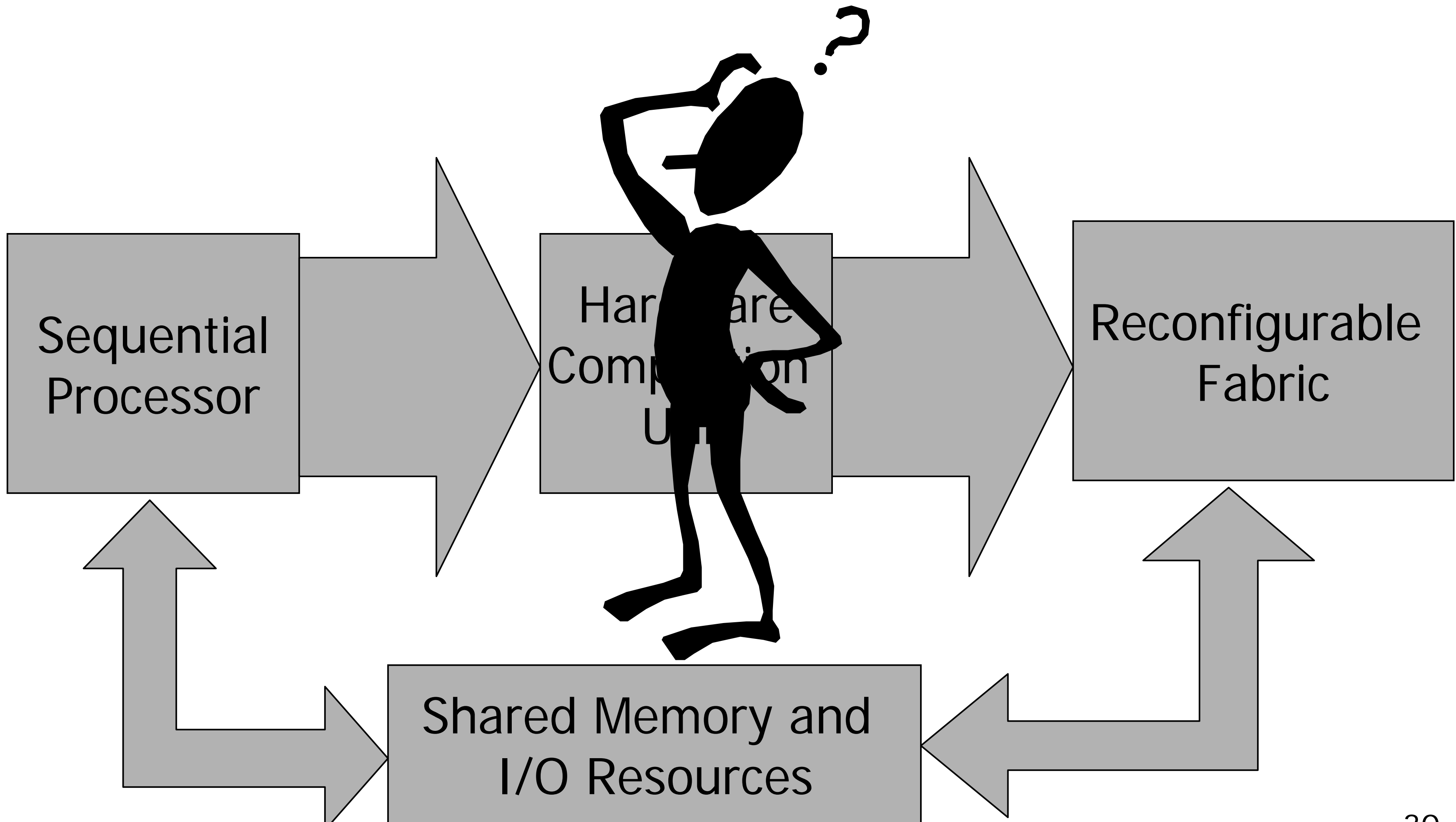






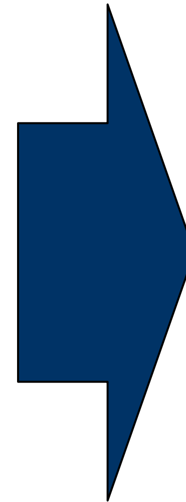


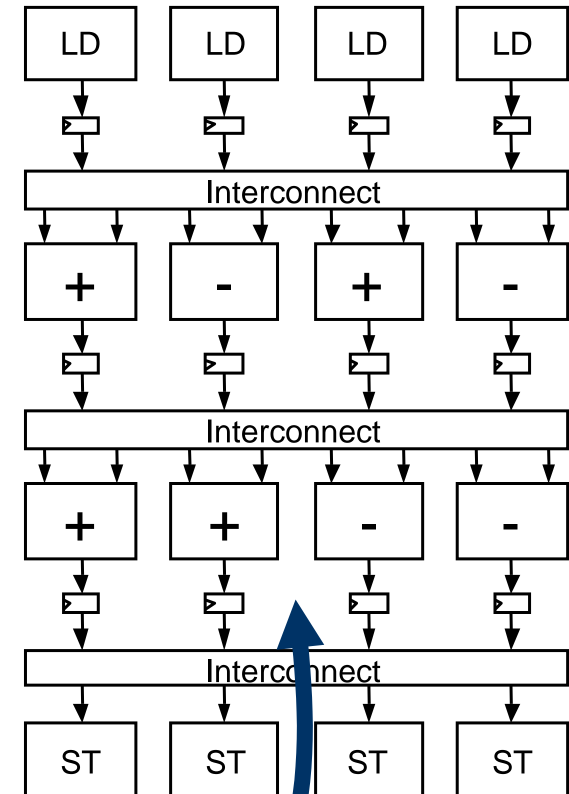
So How Does the HCU Work?



Goal: Code to Fabric Configuration…

```
for (i = 0; i < x; i +=4) {
        a = A[i]; b=A[i+1];
        c = A[i+2]; d = A[i+3];
        e = a + b; f = a - b;
        g = c + d; h = c - d;
        w = e + g; x = f + h;
        y = e - g; z = f - h;
        B[i]   = w; B[i+1] = x;
        B[i+2] = y; B[i+3] = z;
}
```

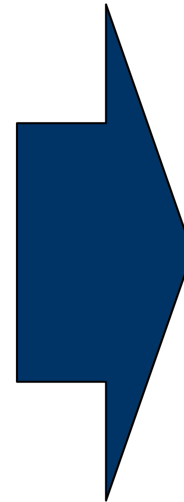




Spatial Information:
- Placement of Operations
- Routing of Operands

…and Processor Instructions…

```
for (i = 0; i < x; i +=4) {
        a = A[i]; b=A[i+1];
        c = A[i+2]; d = A[i+3];
        e = a + b; f = a - b;
        g = c + d; h = c - d;
        w = e + g; x = f + h;
        y = e - g; z = f - h;
        B[i]   = w; B[i+1] = x;
        B[i+2] = y; B[i+3] = z;
}
```



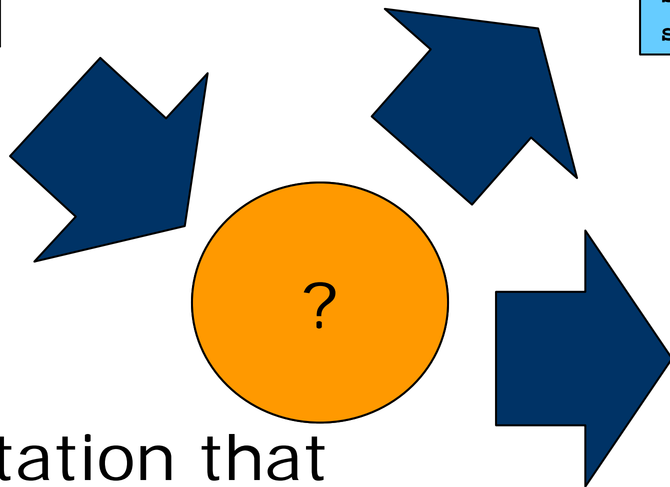
```
load [i]
load [i+1]
load [i+2]
…
add
sub
…
sub
sub
store
store
```

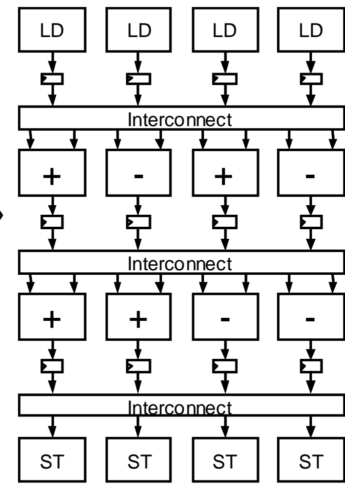
Temporal Information:
- Sequence of Operations
- Specification of Operands

…in a Single Executable!

```
for (i = 0; i < x; i +=4) {
            a = A[i]; b=A[i+1];
            c = A[i+2]; d = A[i+3];
            e = a + b; f = a - b;
            g = c + d; h = c - d;
            w = e + g; x = f + h;
            y = e - g; z = f - h;
            B[i]   = w; B[i+1] = x;
            B[i+2] = y; B[i+3] = z;
}
```

```
load [i]
load [i+1]
load [i+2]
…
add
sub
…
sub
sub
store
store
```



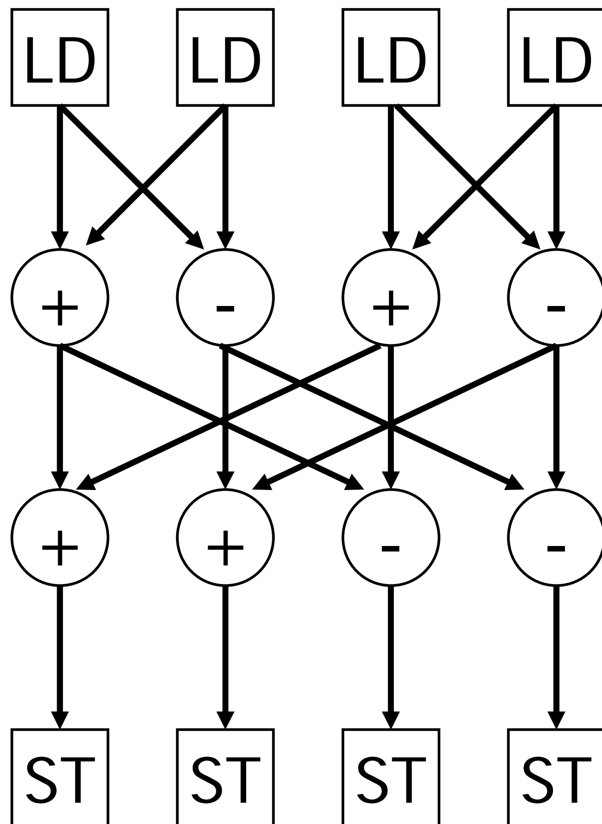


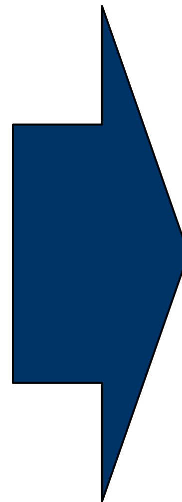
Need one representation that encapsulates operator location and data flow information, in addition to valid sequence information.

Benjamin A. Levine

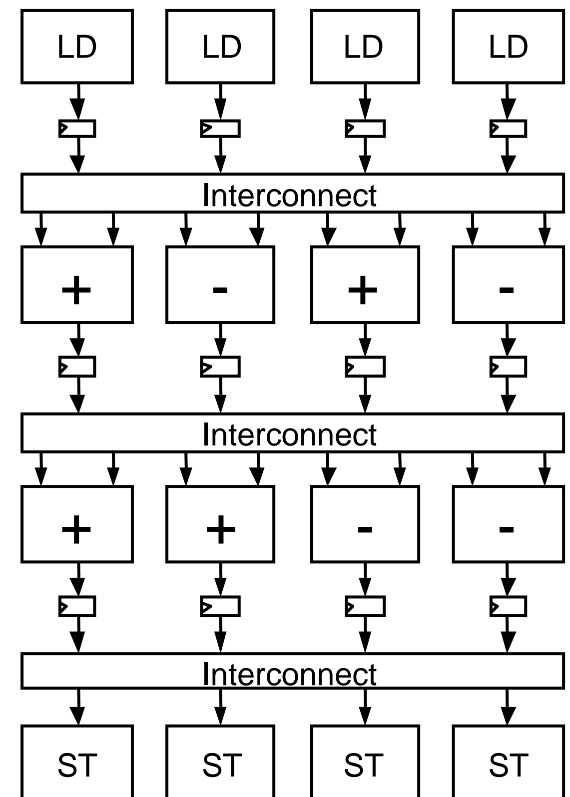
DFG to Configuration is Easy…

Data Flow Graph (DFG) used by compiler….





…could be used to create fabric configuration.



… Except for Traditional Register ISA

RISC code for DFG:

```
ld $r1, 0($r0)
ld $r2, 1($r0)
add $r3,$r1,$r2
sub $r4,$r1,$r2
ld $r1, 2($r0)
ld $r2, 3($r0)
add $r5,$r1,$r2
sub $r6,$r1,$r2
add $r1,$r3,$r5
add $r2,$r4,$r6
st $r1, 0($r7)
st $r2, 1($r7)
sub $r1,$r3,$r5
sub $r2,$r4,$r6
st $r1, 0($r7)
st $r2, 1($r7)
```

- Register file obscured dependencies
  - Re-use of registers
  - Total live variables < # registers
- Independent instructions are distant
- No info about length of dependencies
- The compiler obfuscates information about the DFG

ISA Requirements

- No fixed resources.
- No limit to size of fabric design we can represent.
- Some way to express locality.
- Maintain sequential semantics for processor.
- Easy to generate spatial representation for fabric.

QUEUE MACHINES!

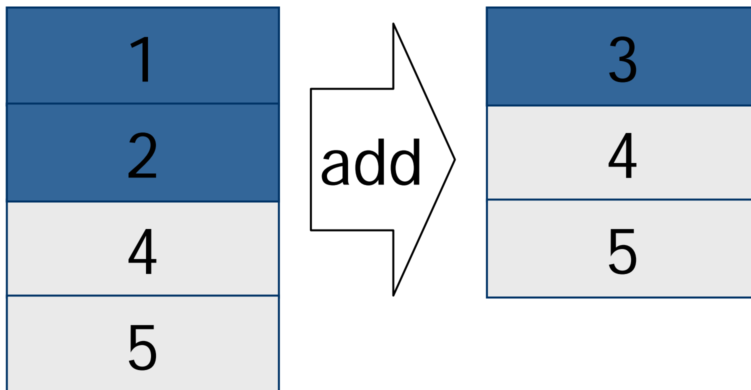
Queue Machine

Like a stack machine, but..

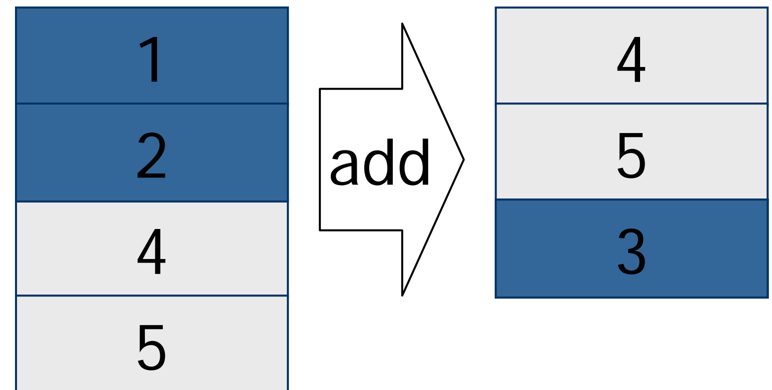
replaces stack with queue.

- Stack : Read from top, write to top.
- Queue: Read from head, write to tail.

Stack Machine



Queue Machine



Advantages of Queue and Stack ISAs

- No register file size limitations:
  - no limit to # of live variables
  - no limit to size of fabric configurations.
  - No register spill code.
- Implicit operand designation:
  - Data dependencies determined only by pattern of operations.
  - Easier to follow dependency chains.

Queue Machine ISA

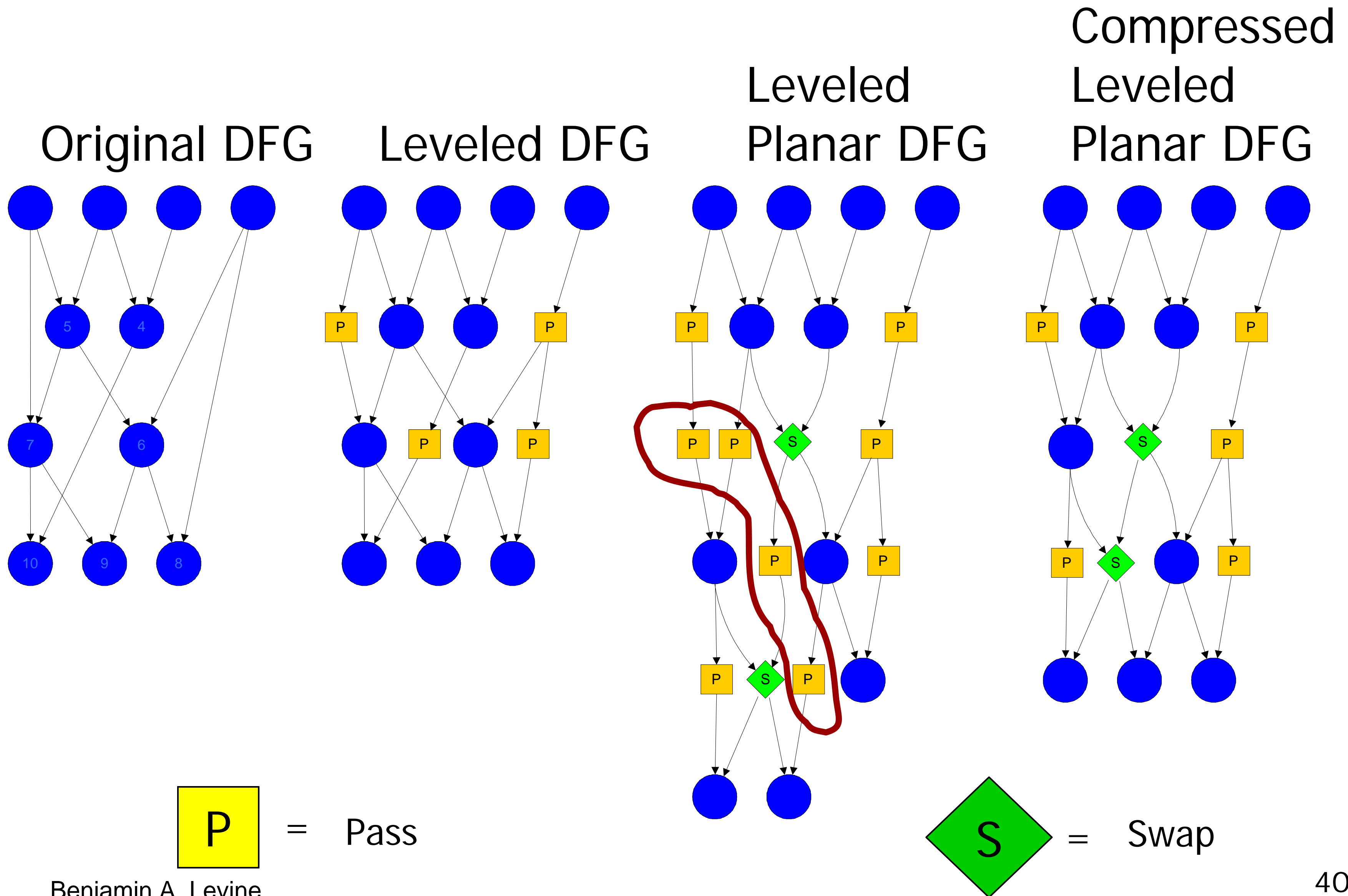
Queue ISA is better than Stack ISA because:

- Can convert any DFG to leveled planar DFG; leveled planar = runs on a queue machine.
- Leveled planar DFG matches pipelined HW.
- Allows simple fabric implementation – no crossing limits required interconnect.
- Sequential semantic of Queue ISA matches row-based fabric configuration.

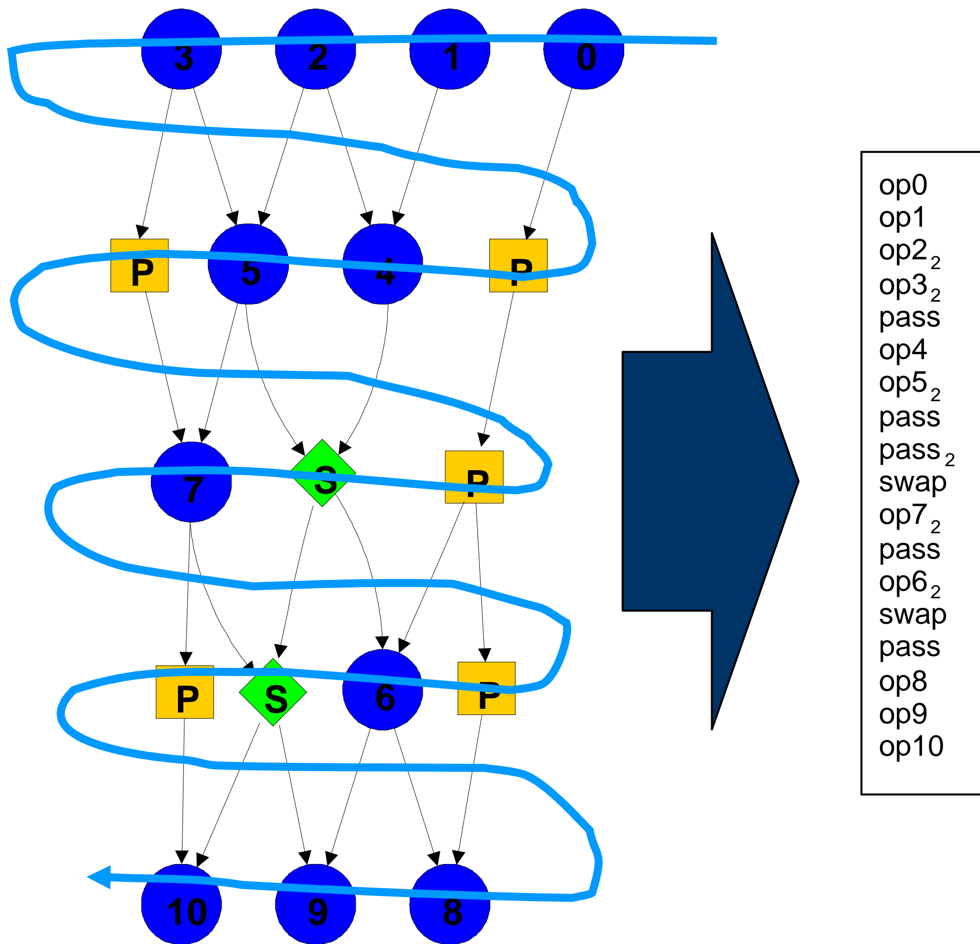
QUEUE ISA Enables

Hardware Compilation in Hardware

Example of Queue DFG generation



Embed DFG in Instruction Sequence



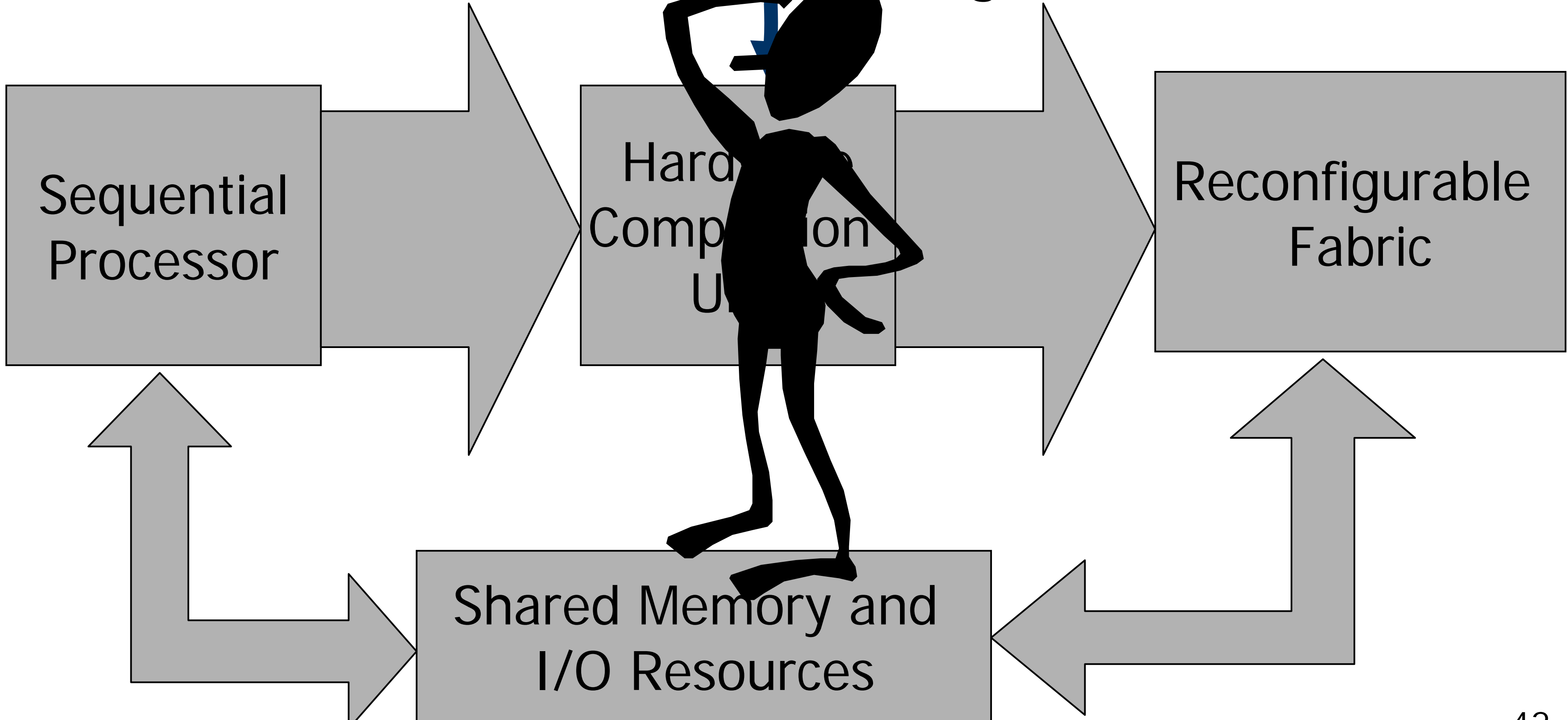
- Sequence has sequential semantic and runs on processor.
- Known properties of leveled planar graph and instruction order convey spatial information.

So How Does the HCU Work?

Instruction Sequence

- Queue ISA
- Embedded DFGs

Queue semantics and leveled planar DFG structure enable fabric generation



Completed Work - Algorithms

- Efficient compile-time heuristics using results from graph theory to allow conversion of any arbitrary DFG to a leveled planar DFG.
- Simple run-time algorithms that can be implemented in hardware for creation of fabric configuration after one sequential pass through the code.

Completed Work – Infrastructure

- Developed software tools for representing and manipulating DFGs and fabrics.
- Developed models for physical characteristics of different fabrics.
- Implemented automatic generation of single-queue executable DFGs from arbitrary DFG.
- Implemented automatic generation of interconnect limited DFGs.

Problem - Queue Code Overhead

	Orig Data Flow Graph		Leveled Planar Graph	
Benchmark	**Ops**	**Depth**	**Ops**	**Depth**
dct	122	18	537	49
fft	88	7	909	41
haar16	124	6	918	17
rc6	74	23	330	42
idea	303	160	1462	235
popcount	31	5	229	24

Ongoing and Future Work

- Evaluate improved ISAs:
  - Combine queue and register ISAs.
  - Modify basic queue ISA.
  - Improve compilation heuristics.
- Need to reduce code size overhead.
- Implement HCU algorithms in HW.
- Quantify how ISA choices impact physical characteristics of fabric.
- Evaluate overall system performance.

Summary

- We need design reuse at all levels to cope with DSM complications.
- Reconfigurable fabrics can help allow design reuse, but heterogeneous executables present obstacle.
- Queue ISA and DFG structure allow “hardware compilation in hardware” and thus single executable hybrid architectures are possible.

Conclusions

- Single-executable hybrid architectures:
  - are a feasible solution to technology challenges.
  - retain advantages of reconfigurable computing.
  - avoid disadvantages of conventional hybrids.
- Careful ISA design is required to:
  - enable hardware compilation in hardware.
  - balance code efficiency and HW complexity.