Accelerators and reconfigurable architectures

Mark Wijtvliet
m.wijtvliet@tue.nl

General purpose processors

- Run mixed application types
  - Operating system
  - Webserver
  - Games
  - Office applications

General purpose processors

- Cheap
  - E.g. Raspberry Pi's Broadcom processor
  - Many others

The Cortex A9 RK3188 quad-core tablet processor chip

Price: **US $10.00** / piece
Bulk Price

Shipping: **US $5.41** to Netherlands via China Post Registered Air Mail
Estimated Delivery Time: 15-39 days (ships out within 2 business days)

Quantity: 1 piece (997 pieces available)

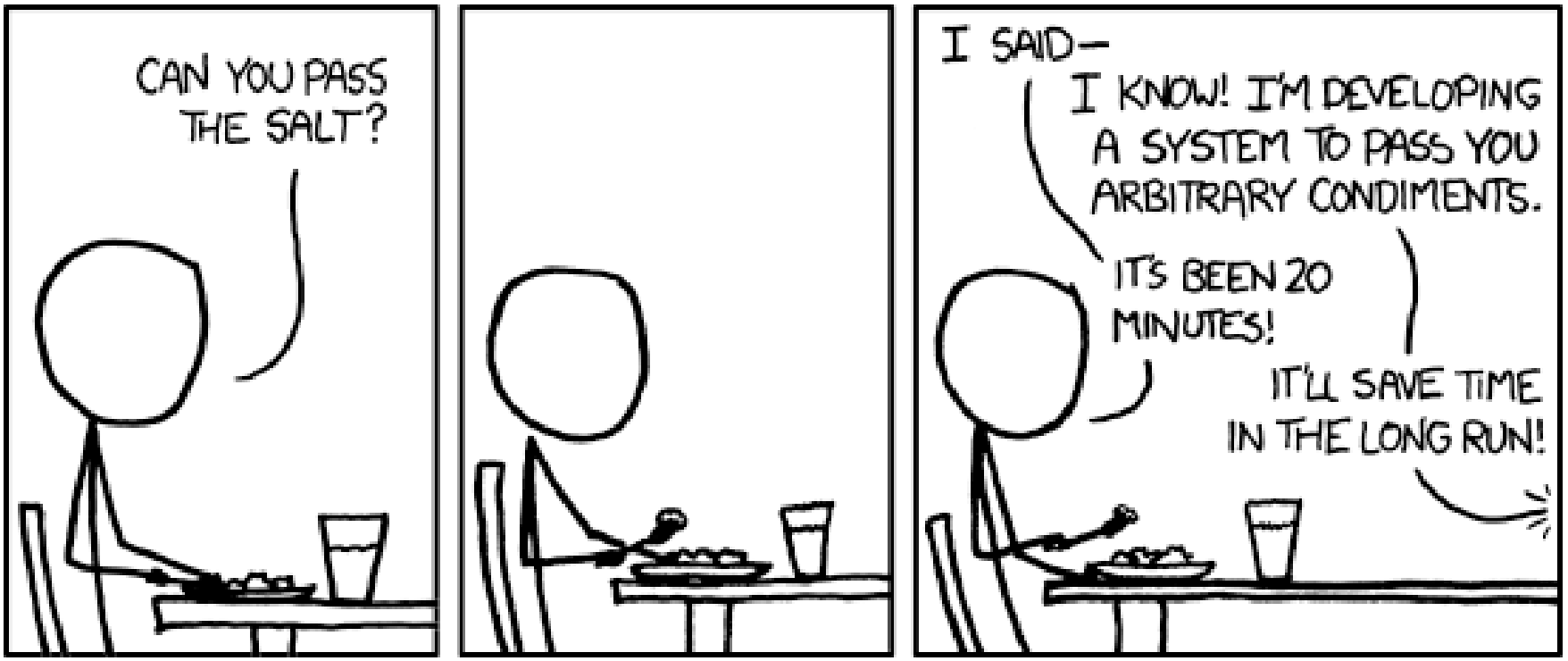
Total Price: **US $15.41**

Buy Now | Add to Cart

Add to Wish List (1 Adds)

General purpose processors

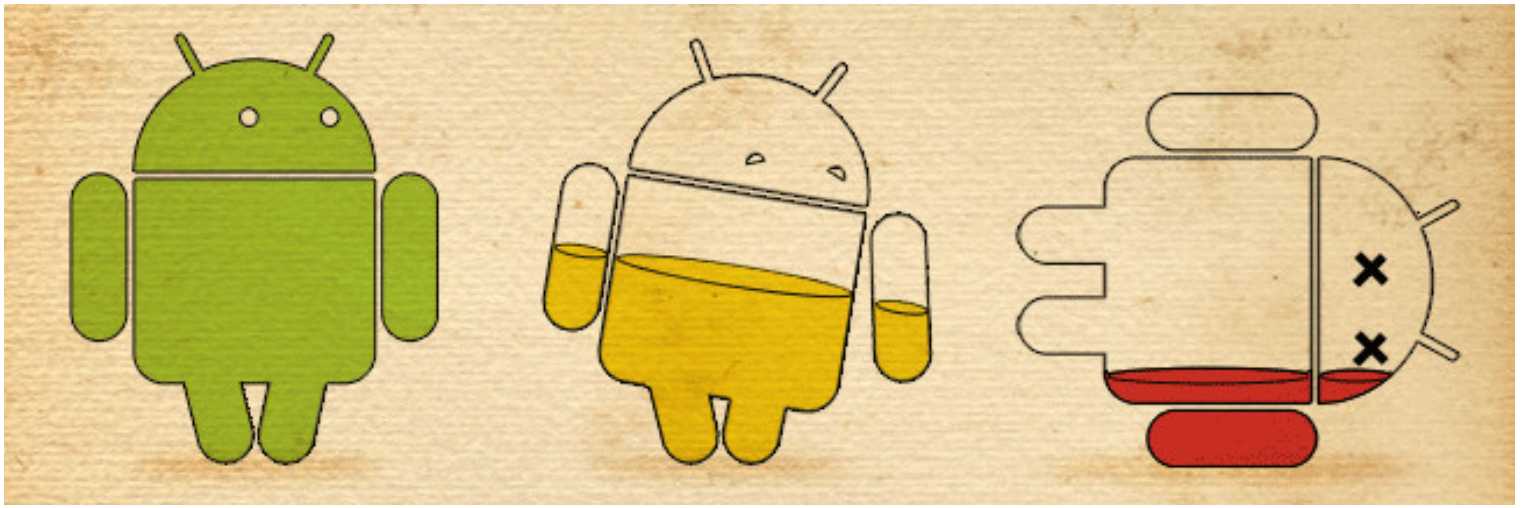
- They work great
- So why this accelerator talk?



[XKCD.com]

Example: mobile phones

- Typically a power budget of 1 Watt (1 J/Sec).
- Modern communication systems (4G) require 1000 GOPS
- That requires a compute efficiency of 1 pJ/OP



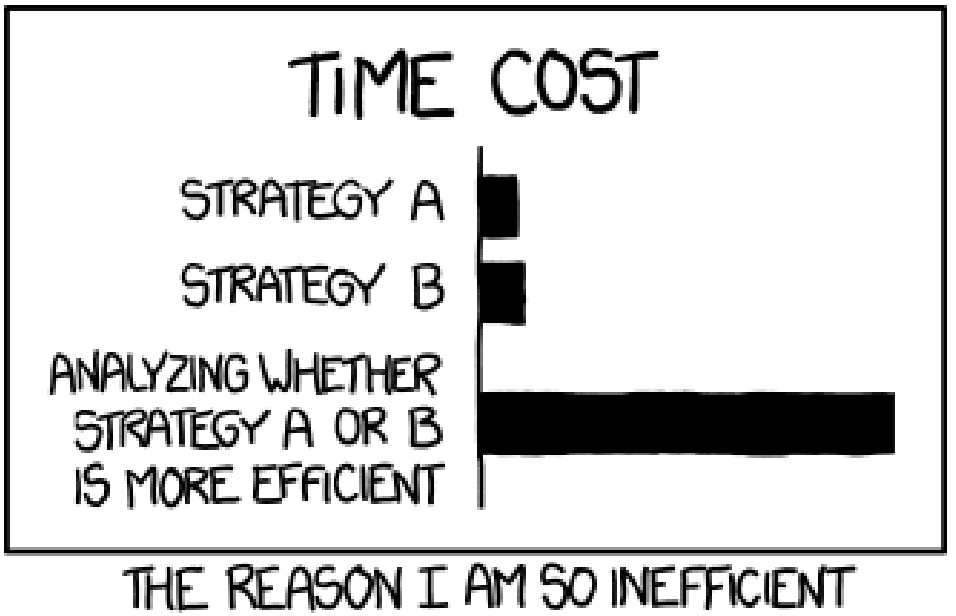
[Multicore for Mobile Phones, Kees van Berkel]

Example: mobile phones

- (Older) ARM11: 200 pJ/OP (65nm)
- A modern ARM, in 28nm
  - Scaling is 1/(S^2)
  - Should be around 37 pJ/OP
  - Numbers not public

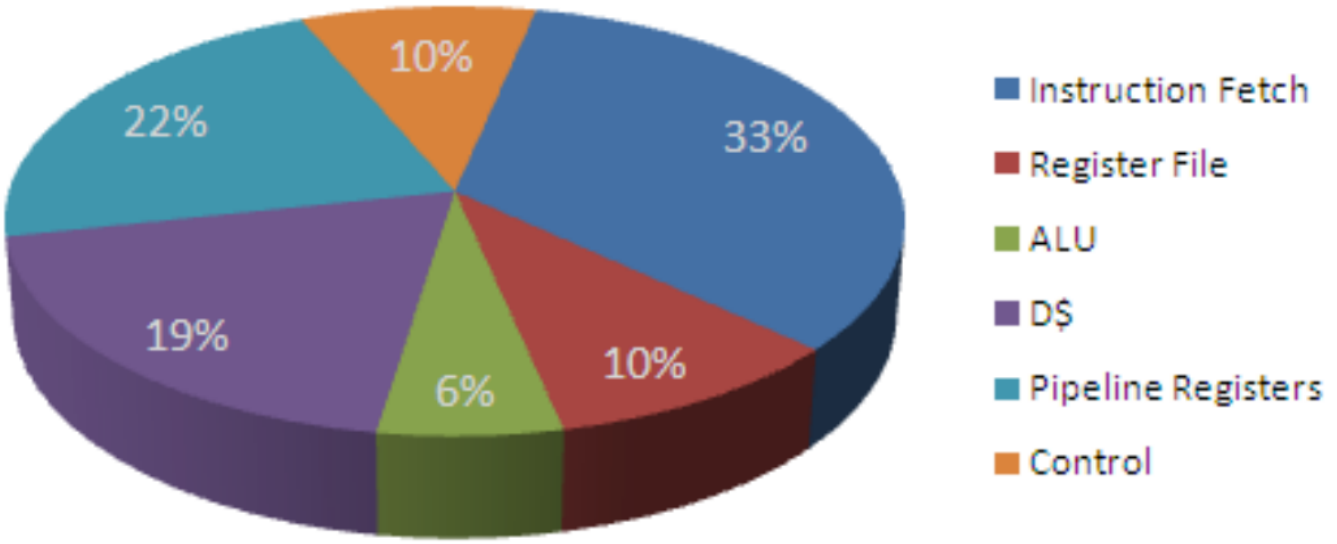
General purpose processors

- Why are general purpose processors not powerful and efficient enough?



[XKCD.com]

Inefficiencies in processors



[Understanding sources of inefficiency in General-Purpose Chips, Hameed et al.]

Inefficiencies in processors

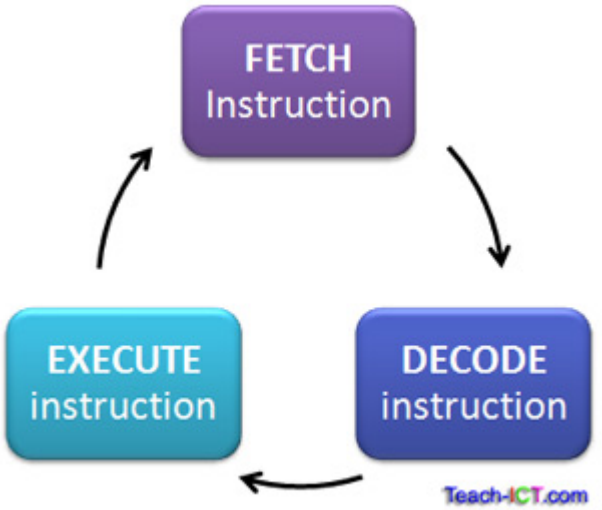
- Factor 500 difference to ASIC in energy.
  - For H.264 encoding
  - For a 2.8 GHz Pentium 4 and a Tensilica Micro-processor.

Inefficiencies in processors

- Instruction fetching and decoding
- Communication (register file, caches, etc.)
- Hardware reconfiguration (in processor pipeline)

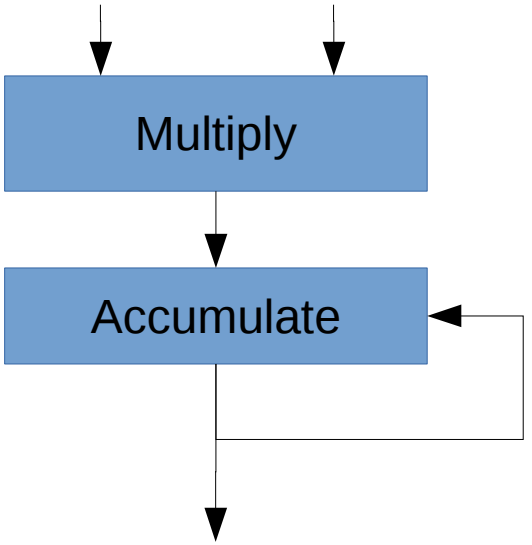
Instruction fetching and decoding

- Where does the overhead come from?
  - Addressing and loading the instruction word from memory.
  - Instruction caches.
  - Decoding the instruction to decoded instruction bits that control the processing pipeline.



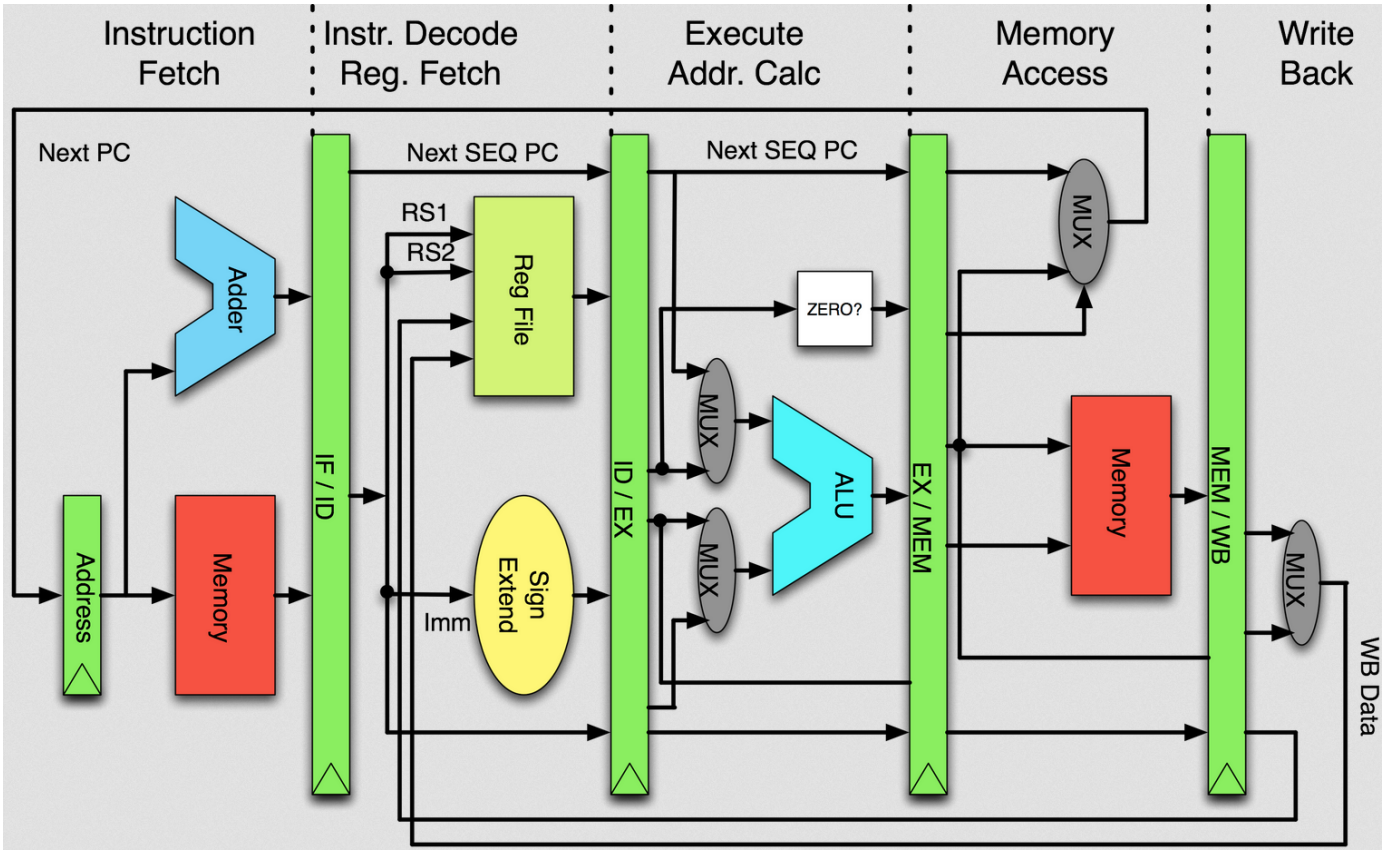
Data transport

- Where does the overhead come from?
  - Register file access
    - How do processors reduce this?
  - Data caches



Hardware reconfiguration

- Mostly multiplexers in the datapath



General purpose processors

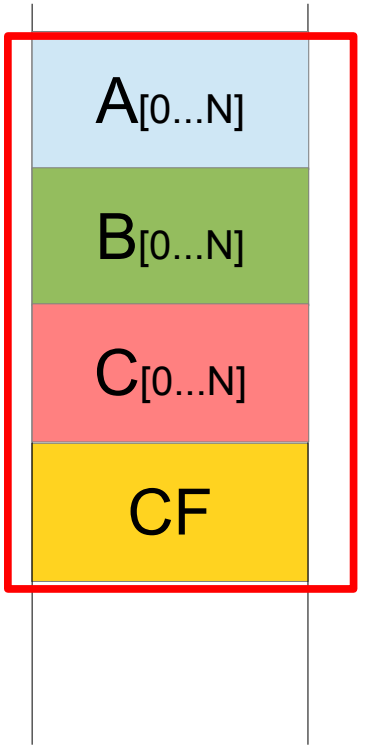
- Lessons learned:
  - Reduce instruction fetching and decoding
  - Reduce cycle-based hardware reconfiguration.
  - Reduce data transport to and from memories and RF.
  - Still needs to be programmable

Hardware acceleration

Static control

- Loops are the best candidate for static control
  - Do the same thing many times

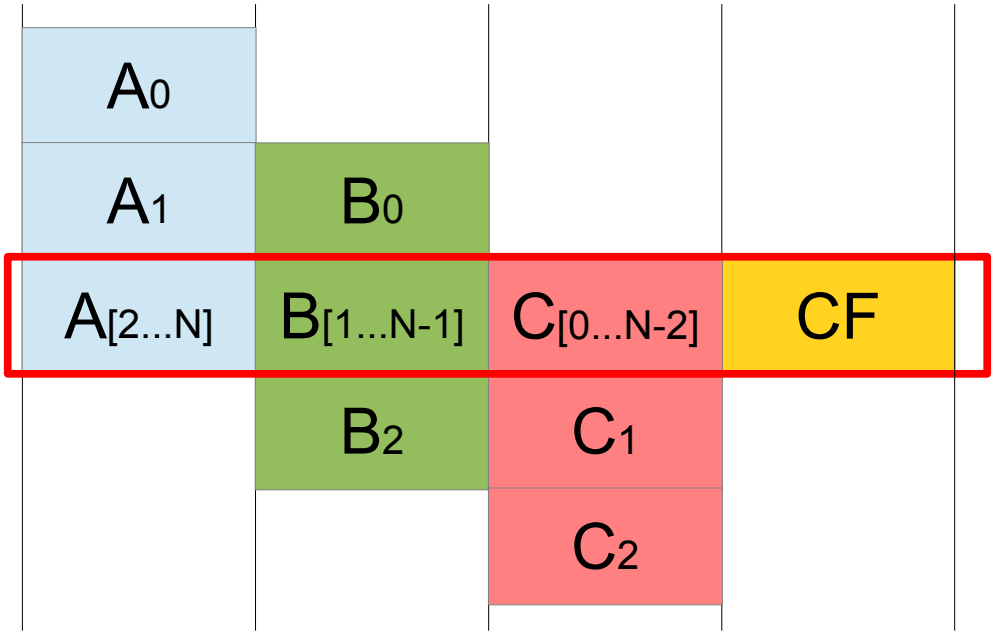
```
for i = 0 to N
  A
  B
  C
```



Static control

- Loops are the best candidate for static control
  - Do the same thing many times
- Software pipelining

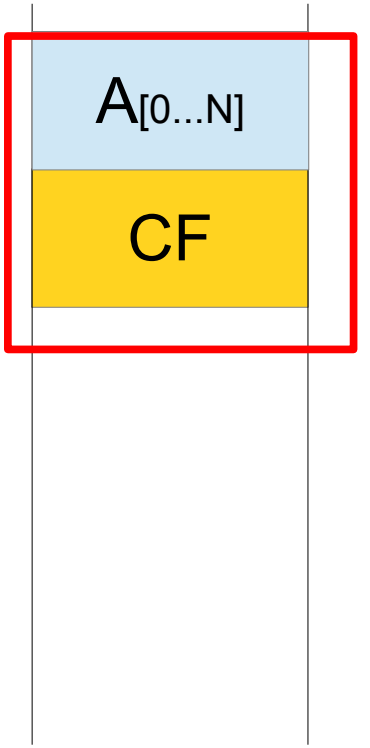
```
for i = 0 to N
   A
   B
   C
```



Static control

- However, this does not work on a general purpose processor...
  - Control is never static

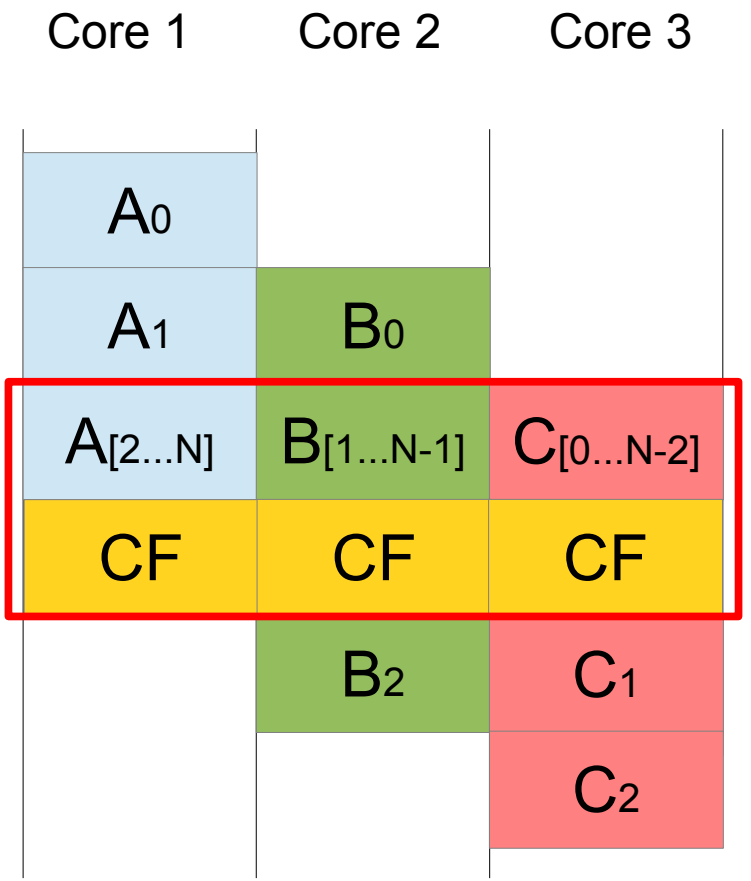
```
for i = 0 to N
  A
```



Static control

- Not even for multi-cores

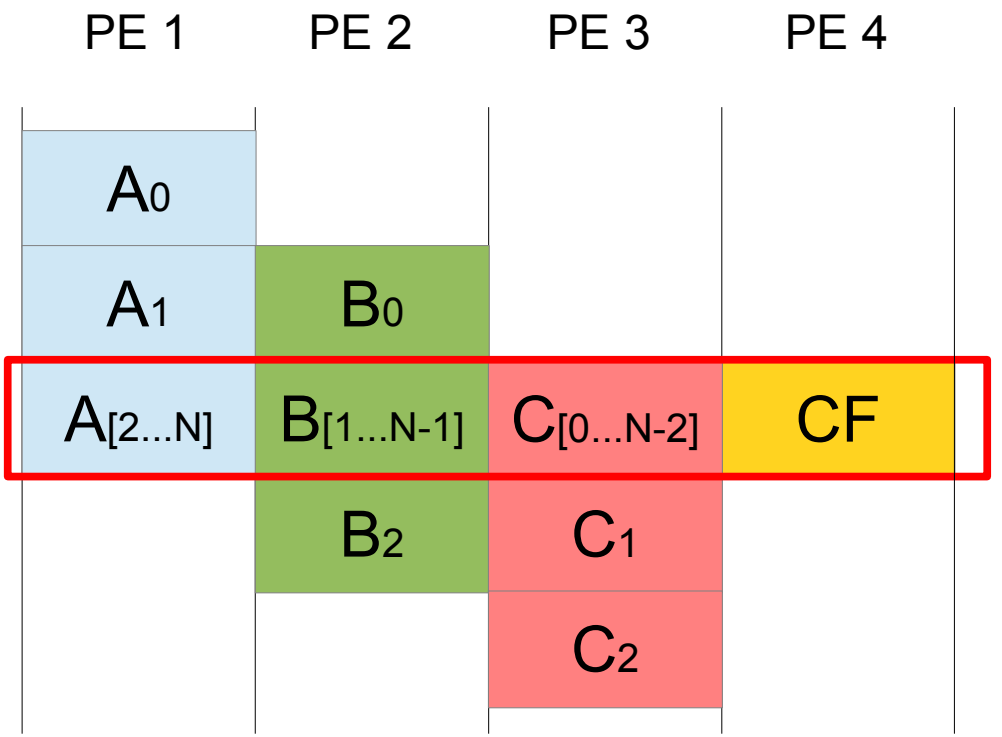
```
for i = 0 to N
   A
   B
   C
```



Static control

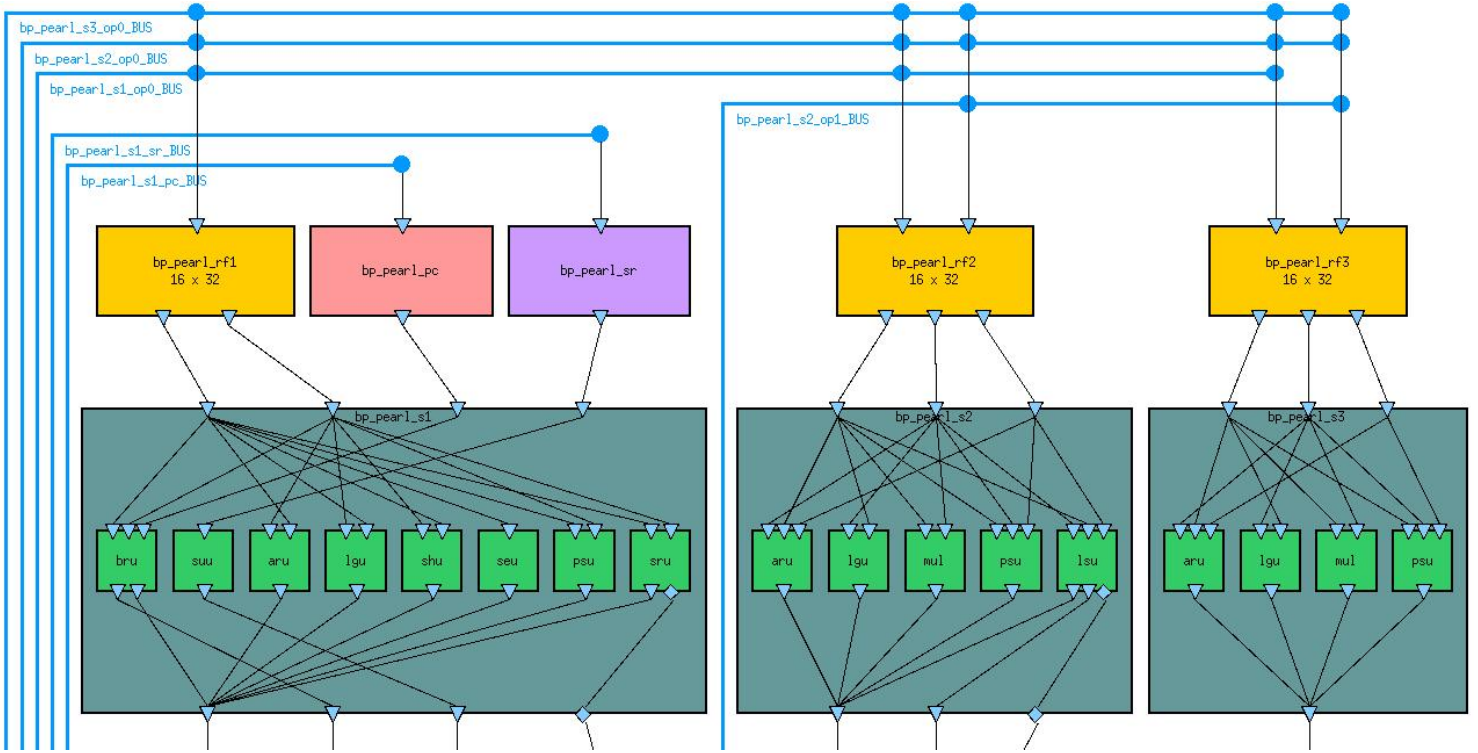
- What type of processor would you use?

```
for i = 0 to N
   A
   B
   C
```



Very Long Instruction Word processors (VLIW)

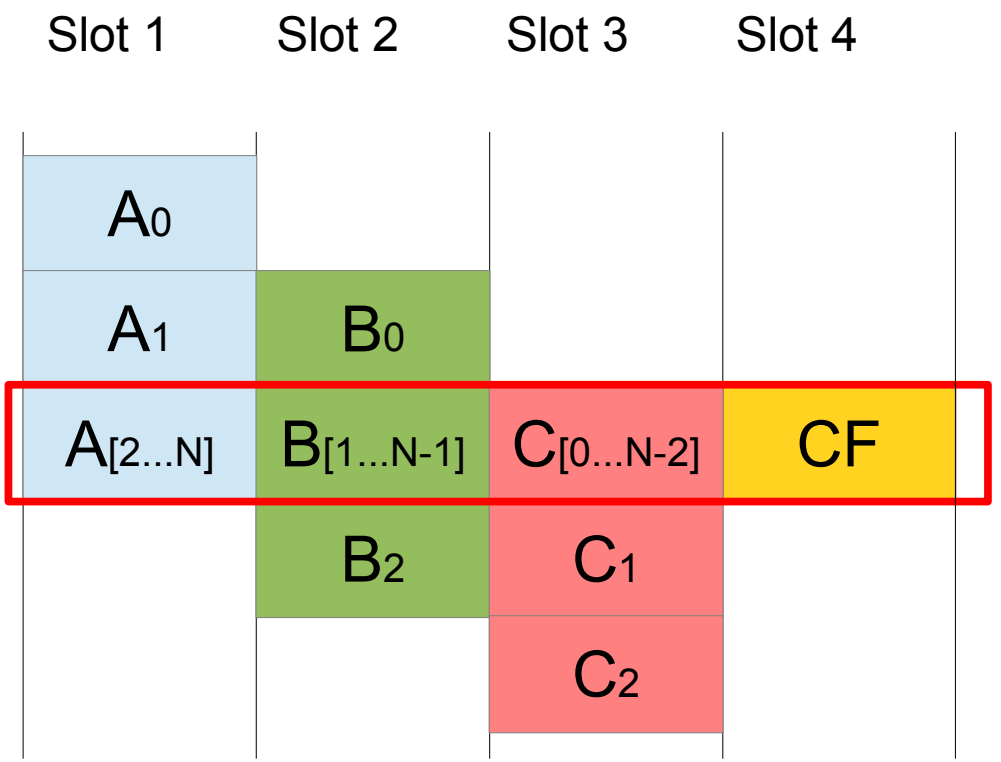
- Processor with multiple issue-slots
- One long instruction controlling them all



Very Long Instruction Word processors (VLIW)

- If we have a 4-issue VLIW we can do our loop in a single cycle.

```
for i = 0 to N
   A
   B
   C
```



Very Long Instruction Word processors (VLIW)

- But once the VLIW is manufactured the number (and functionality) of the issue slots is fixed.
- Problem ...

```
for i = 0 to N
   A
   B
   C
   D
```

Very Long Instruction Word processors (VLIW)

- The good:
  - Can achieve single-cycle loops
  - Programmability is OK.

- The bad:
  - Needs to have enough issue slots.
  - Too many issue slots result in wasted energy
  - Does little against data transport reduction
    - multiple multi-ported register files

ASICs

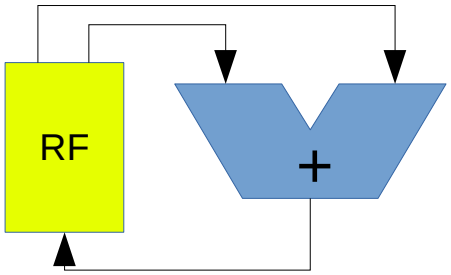
- The application is completely software pipelined and implemented ('hard coded') in hardware.
  - No more instruction fetching and decoding
  - But cannot be changed anymore after production
- But ASICs can do something about RF and memory accesses...

Spatial layout

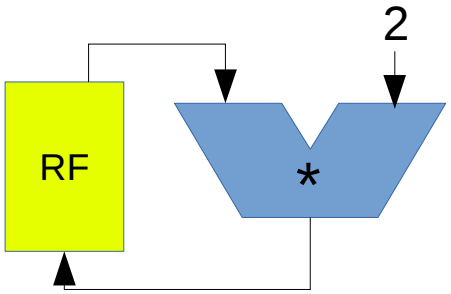
Compute: (A+B) * 2

General purpose | *VLIW* | *Spatial layout*

Cycle N

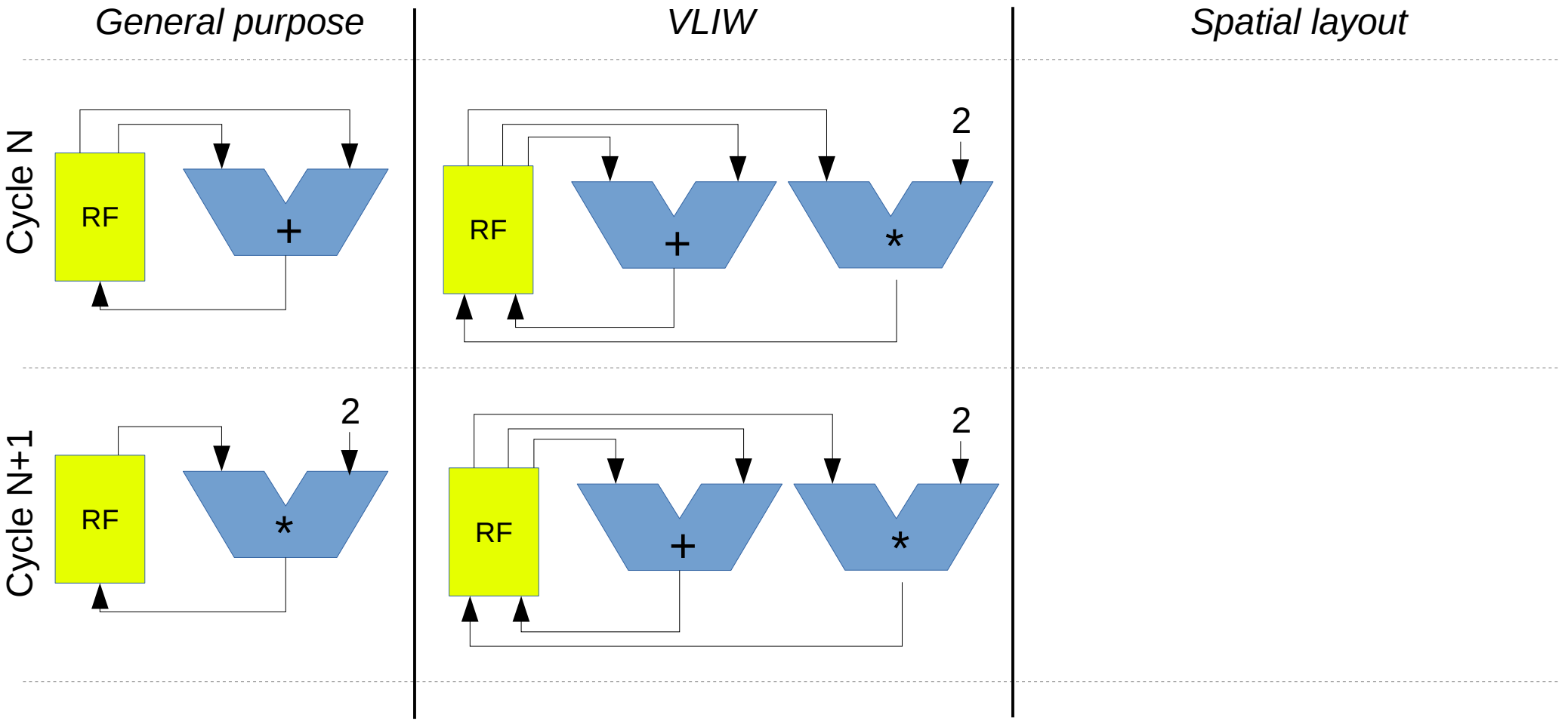


Cycle N+1



Spatial layout

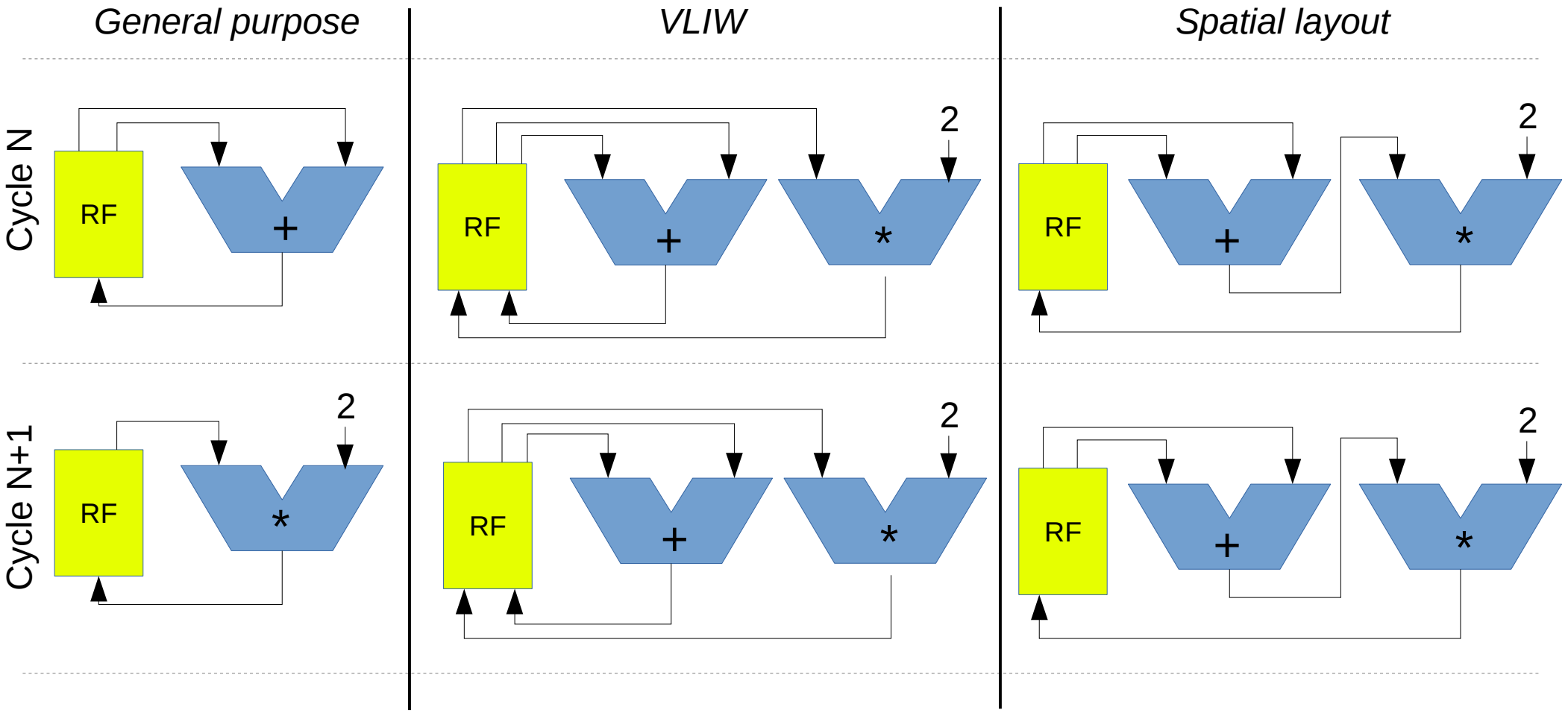
Compute: (A+B) * 2



2 times the throughput!

Spatial layout

Compute: (A+B) * 2



2 times the throughput!

2 times the throughput!

ASICs

- Very efficient
  - (Almost) no control
    - Some configuration registers
    - Fully software-pipelined hardware implementation
  - Reduce memory accesses
    - With spatial layout many register file (and memory) accesses can be avoided
- Very inflexible
  - Highly optimized for a very small application set

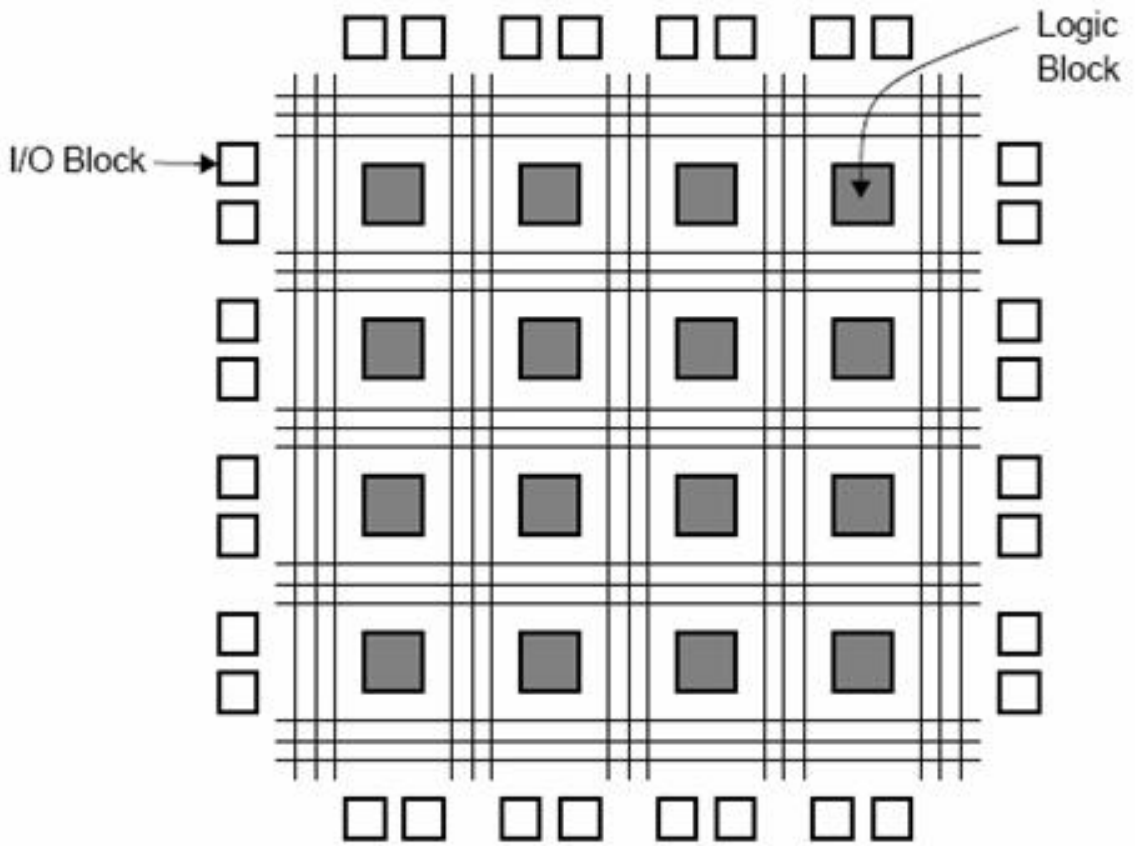
Reconfigurable hardware

Field Programmable Gate Arrays

- What are FPGAs?

Field Programmable Gate Arrays

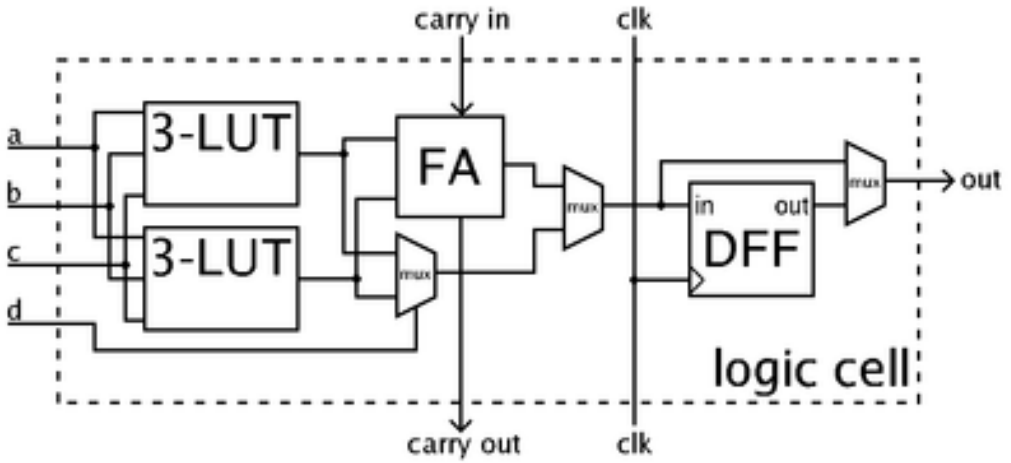
- Chip full of configurable logic blocks/cells



[fpgacentral.com]

Field Programmable Gate Arrays

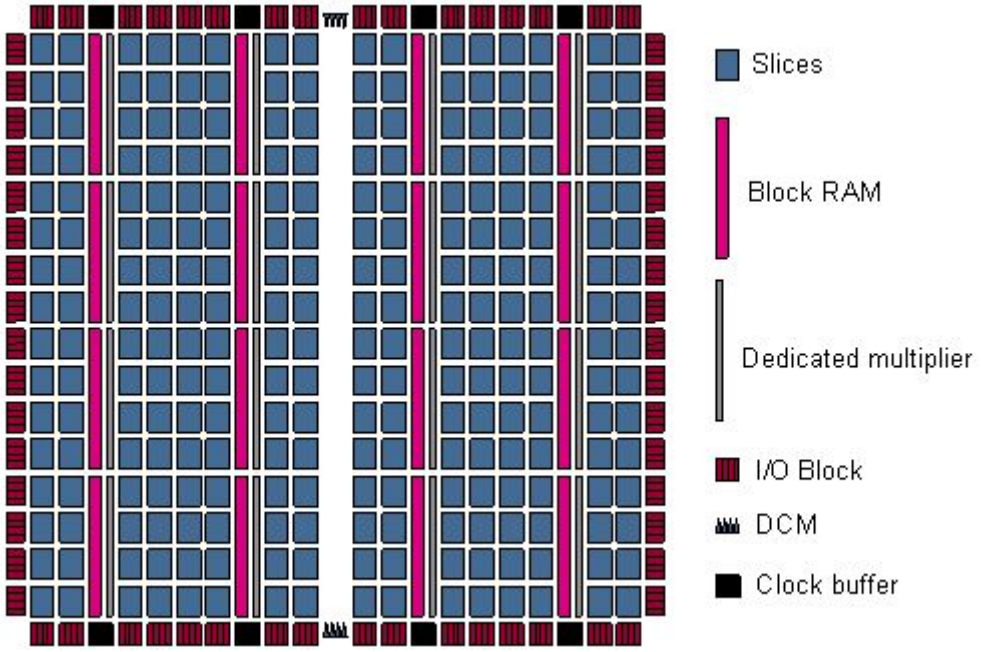
- What is in a logic block
  - Look-up tables
  - Full adder
  - Flip-Flop
  - Some multiplexers



[fpgacentral.com]

Field Programmable Gate Arrays

- Other blocks you find in FPGAs:
  - DSP blocks
  - Memory blocks (BRAMs)
  - Complete processor cores



[Intechopen.com]

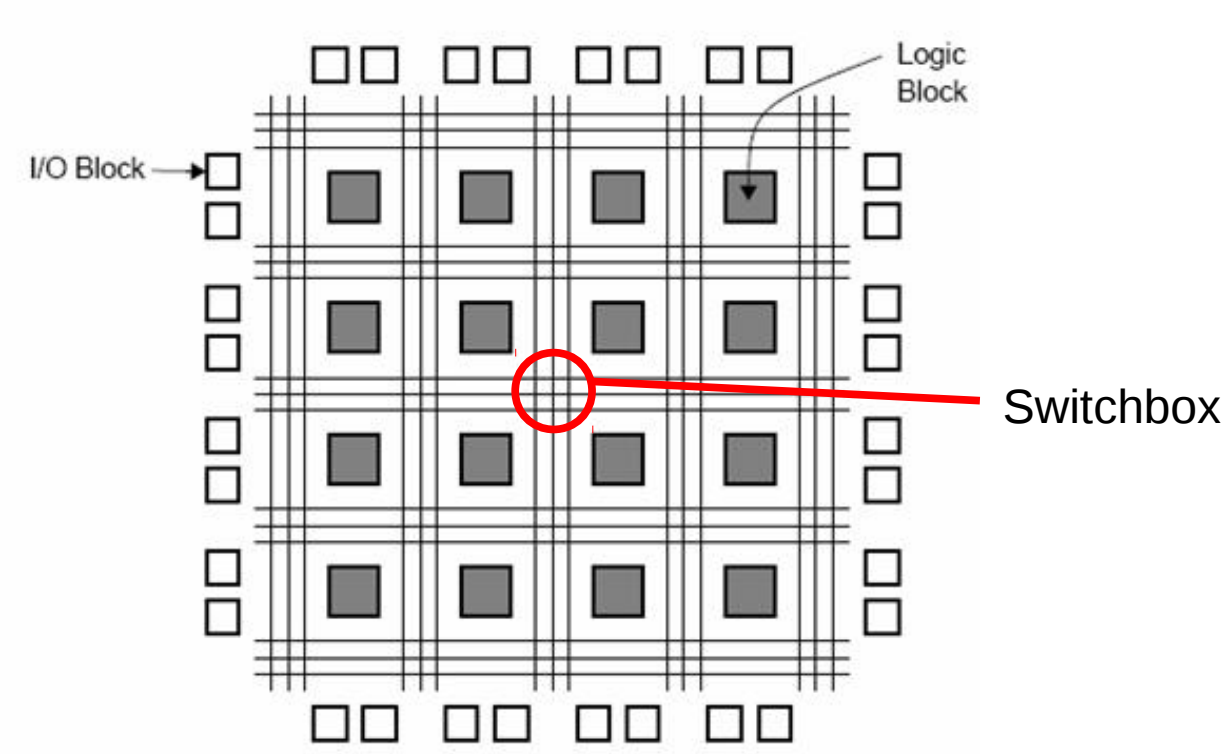
Field Programmable Gate Arrays

- With the exception of specialized blocks most FPGAs contain one-bit blocks.
- Allows you to build arbitrary hardware
  - Like a box full of logic gates to build circuits.
- These blocks can be connected together via the interconnect.

[chipsetc.com]

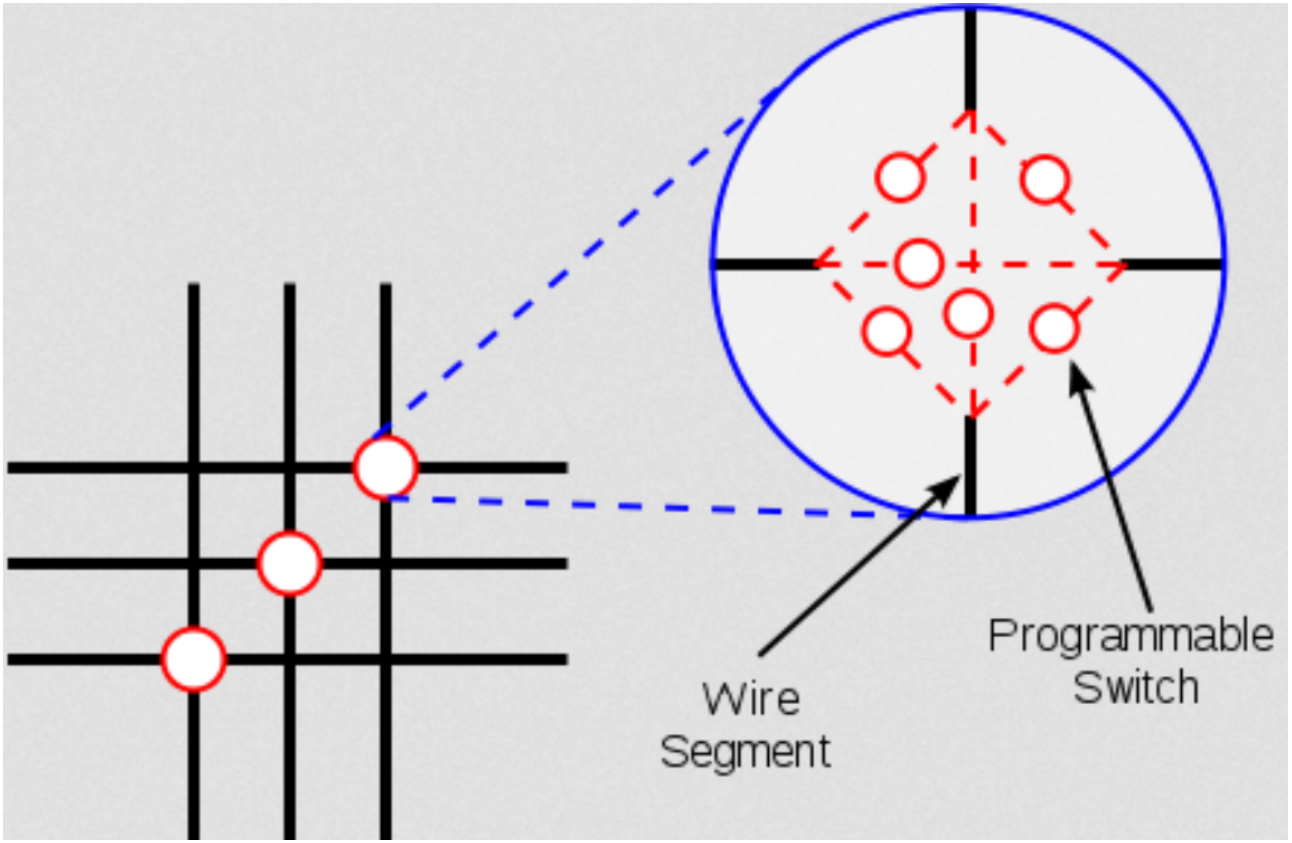
Field Programmable Gate Arrays

- The interconnect on a FPGA is static:
  - Configured at application level (typically when you power-up the FPGA).
  - Connections are fixed after that



Field Programmable Gate Arrays

- Switch-boxes
  - As the name implies ...



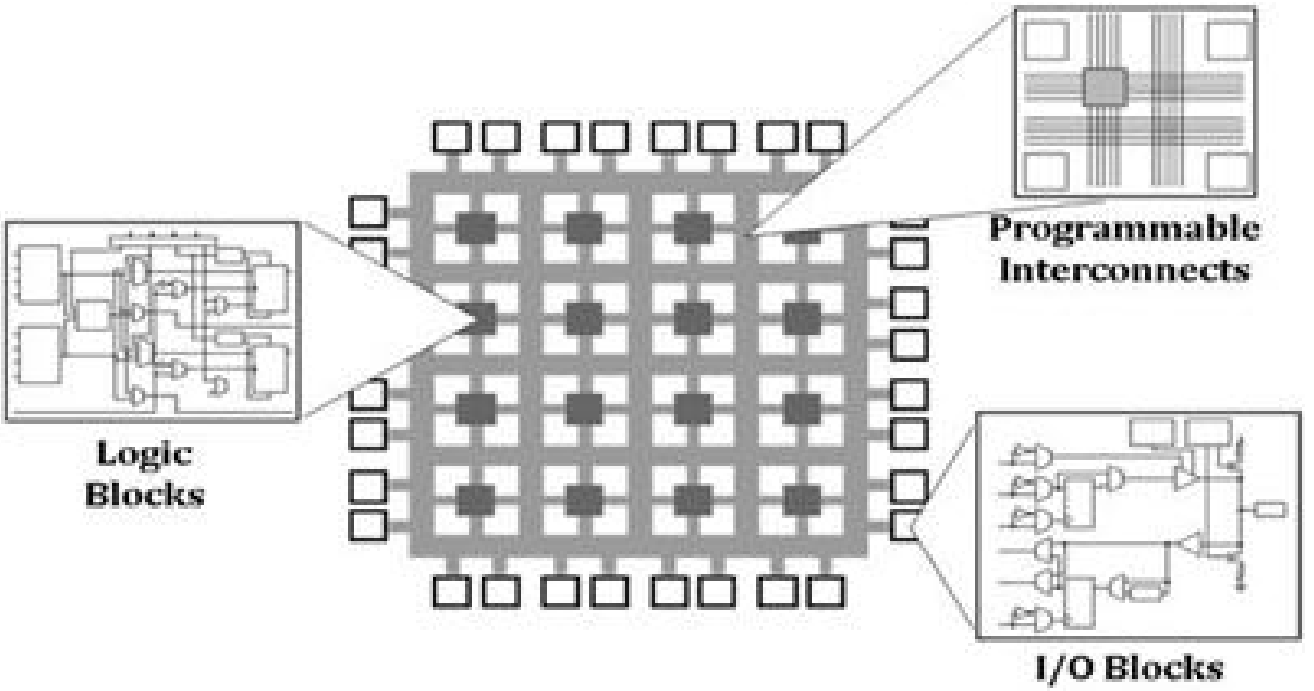
Field Programmable Gate Arrays

- By configuring the interconnect and the logic blocks arbitrary (digital) circuits are possible.
- This allows for building specialized circuits that implement your algorithm with:
  - Spatial layout
  - Static control
  - Or a processor that runs software if you like to...

Field Programmable Gate Arrays

- Spatial mapping and static control
- Reconfigurable
- … no such thing as a free lunch ...

Spatial Layout in FPGA



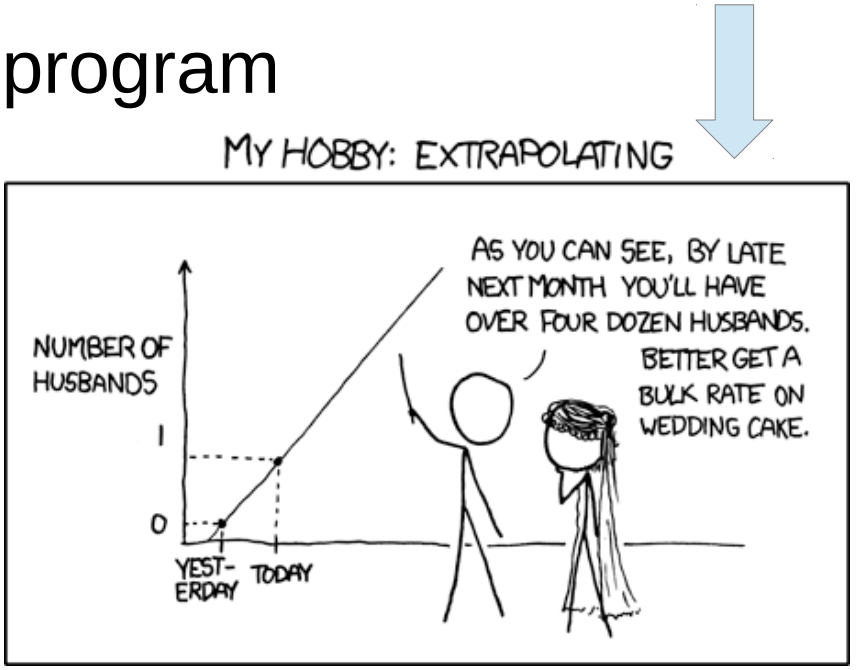
FPGA Configures at gate level, which incurs large overheads:

- Large configuration memory (SRAM leakage: high static power)
- Complex routing network (many long wires: high dynamic power)

Image Source: www.ni.com

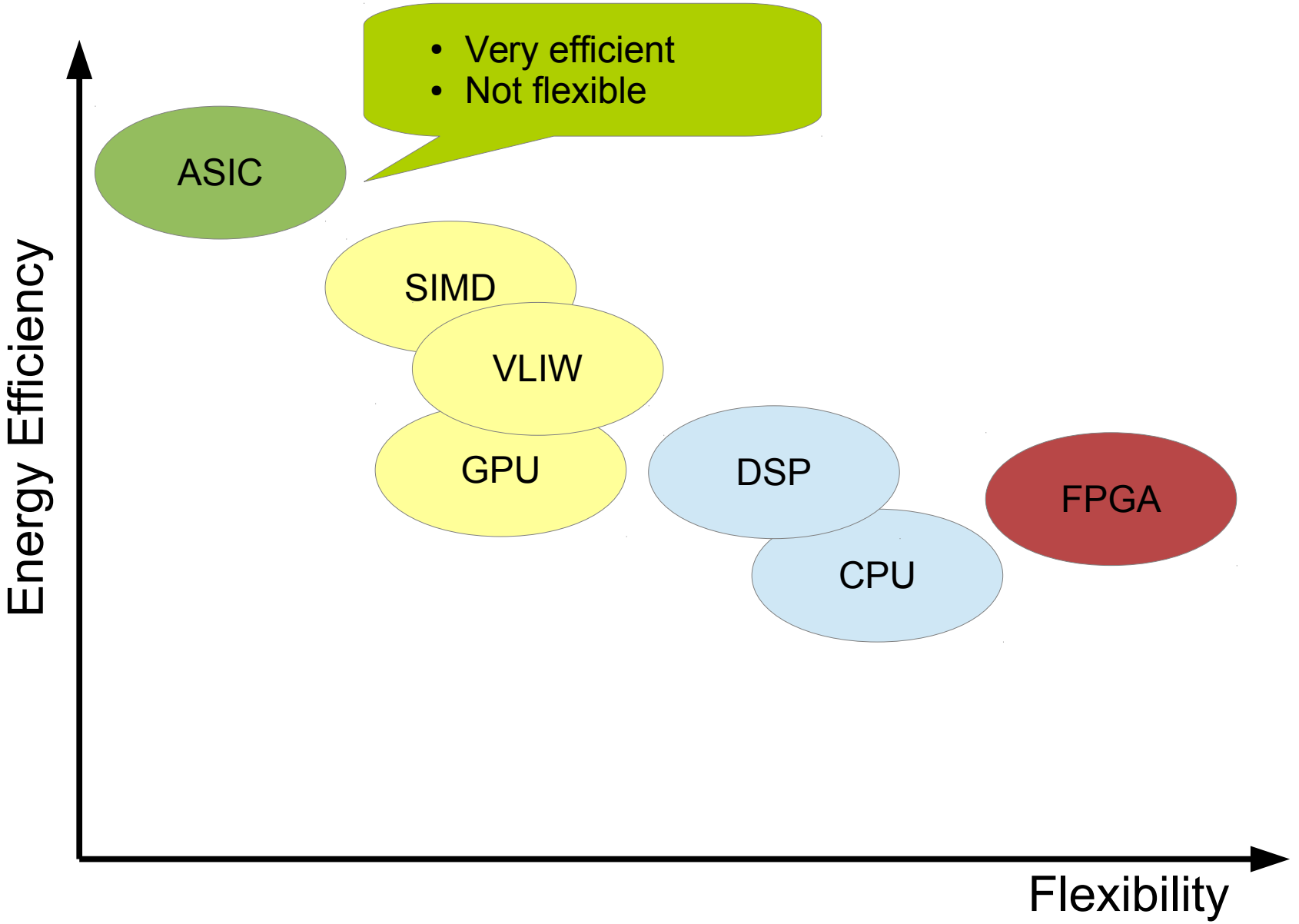
Field Programmable Gate Arrays

- Each configurable item has some configuration memory cells attached that configure it.
- Often several megabits
- Memory cells have leakage
- … Many cells have more leakage …
- Not trivial to program

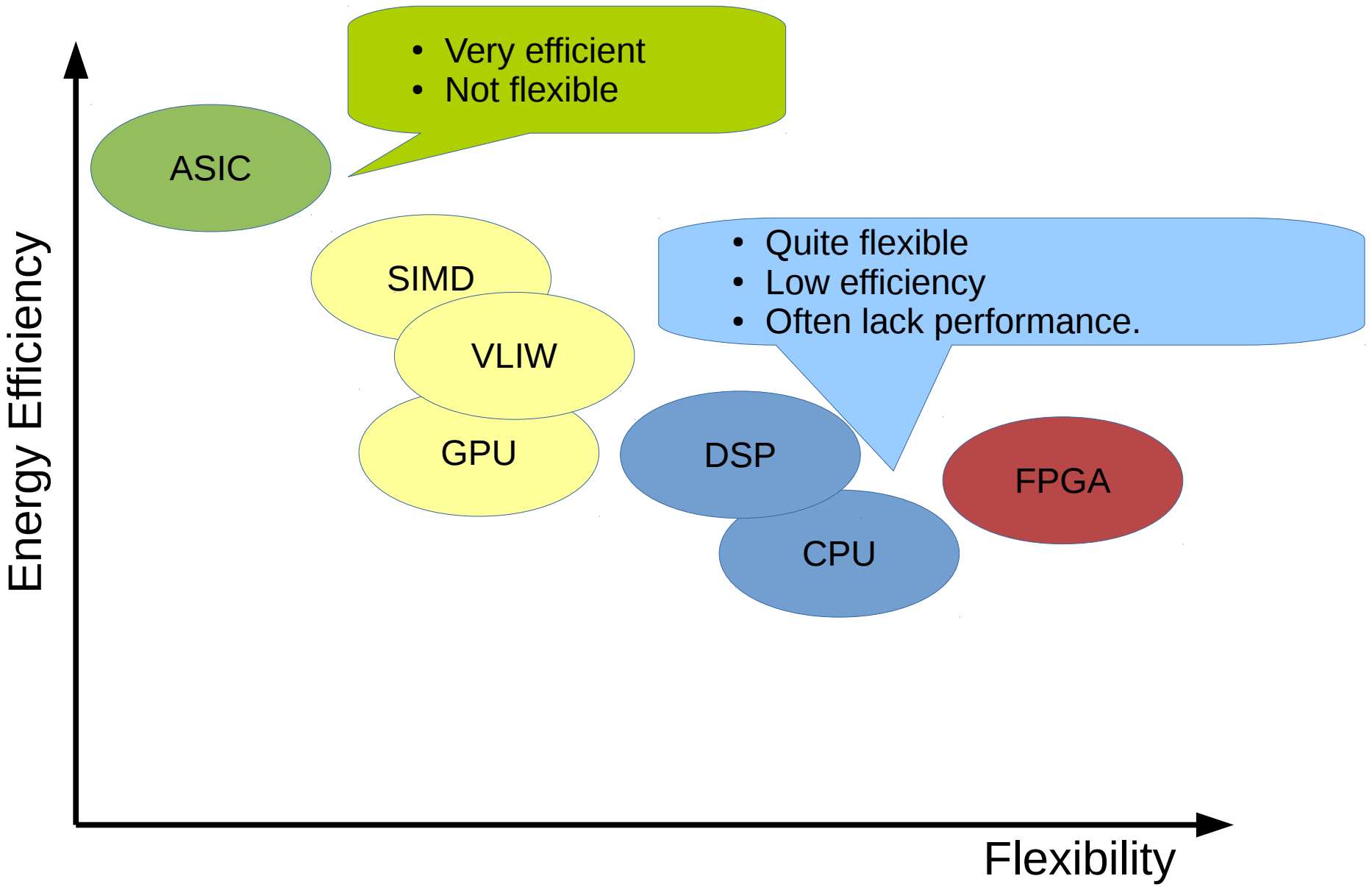


[xkcd.com]

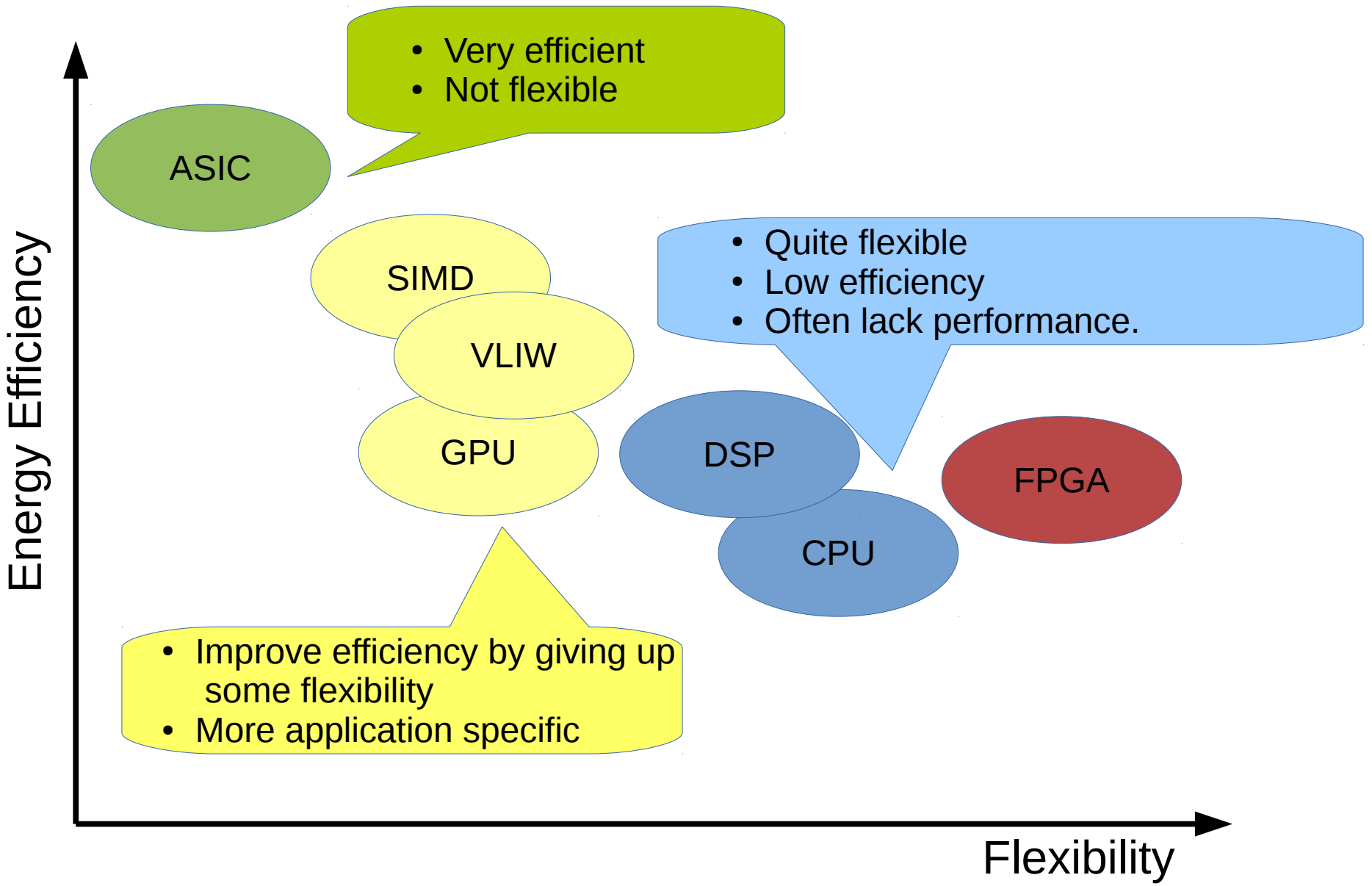
What have we learned so far?



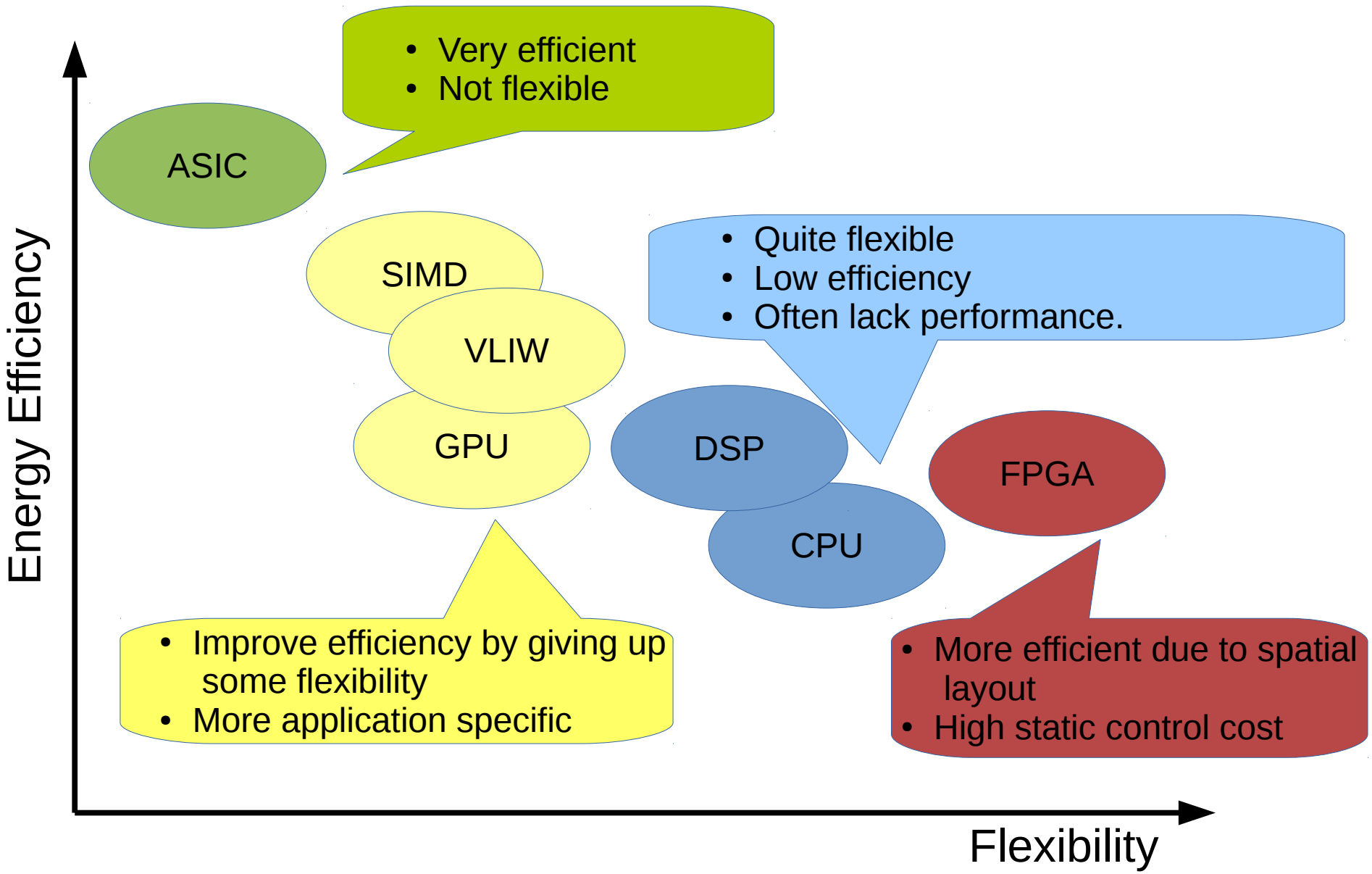
What have we learned so far?



What have we learned so far?

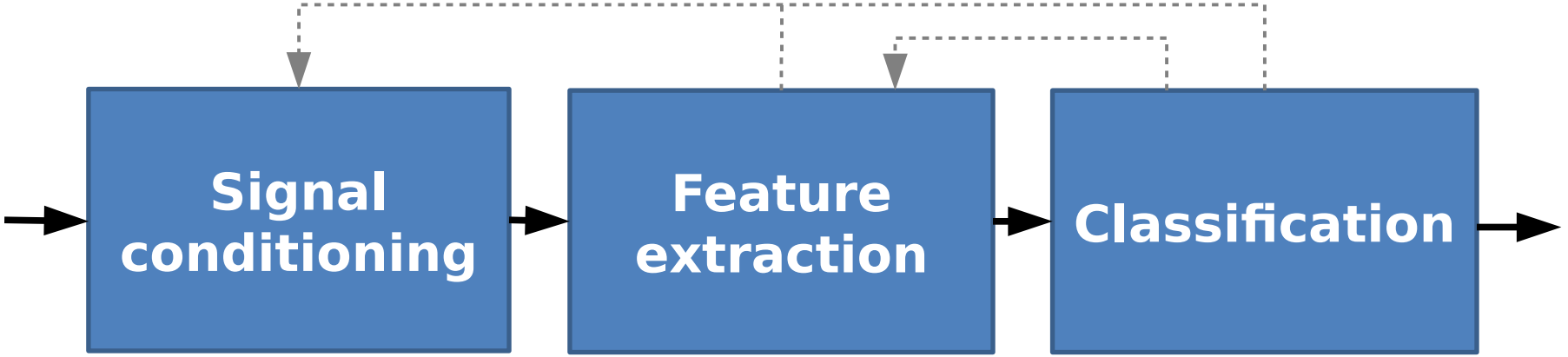


What have we learned so far?



Coarse Grained Reconfigurable Architecture

Signal processing domain

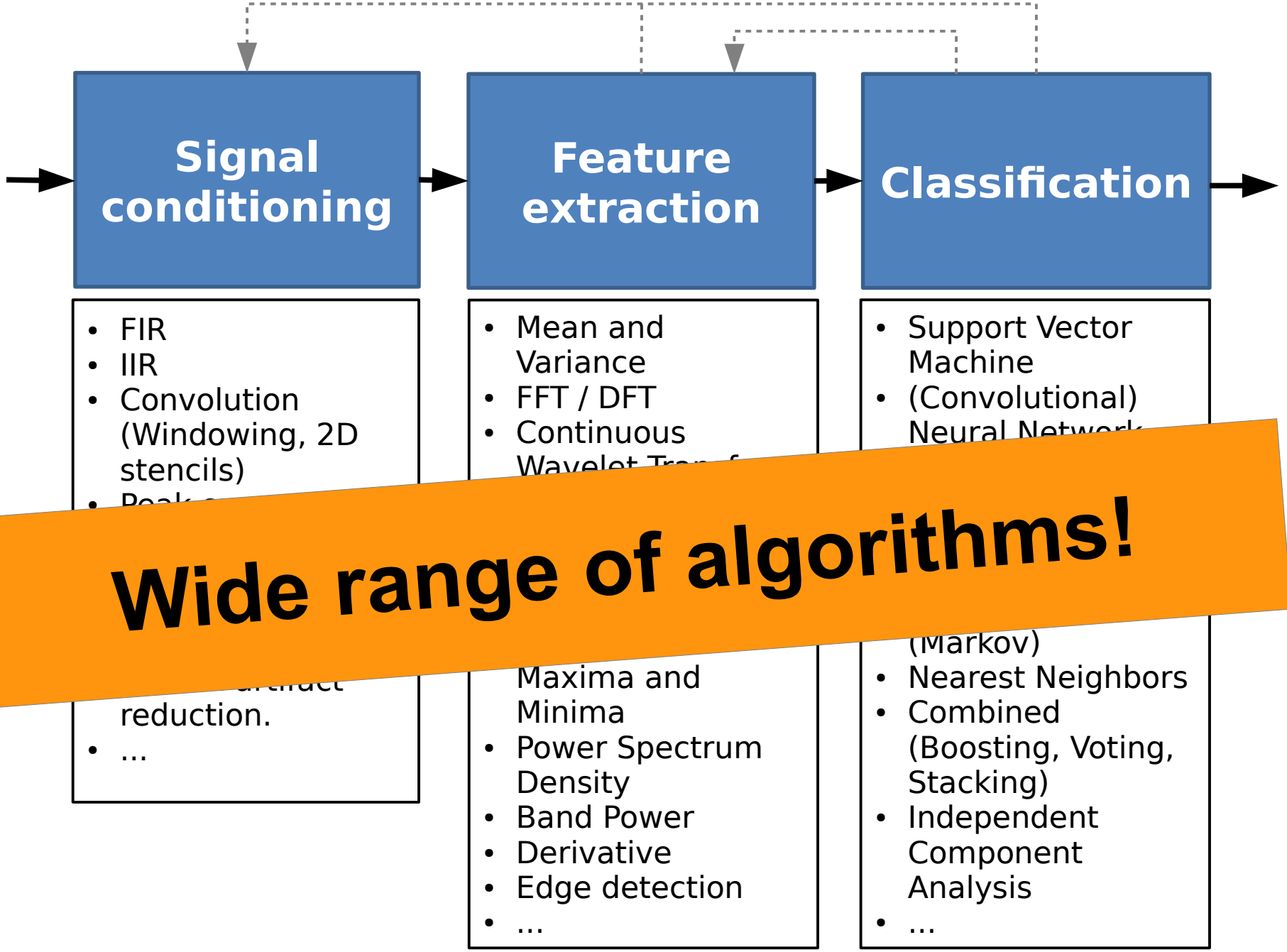


- FIR
- IIR
- Convolution (Windowing, 2D stencils)
- Peak-suppression (Clipping)
- Amplitude squaring
- Sensor fusion, motion artifact reduction.
- ...

- Mean and Variance
- FFT / DFT
- Continuous Wavelet Transform
- Discrete Wavelet Transform
- Line length
- Root Mean Square
- Number of Maxima and Minima
- Power Spectrum Density
- Band Power
- Derivative
- Edge detection
- ...

- Support Vector Machine
- (Convolutional) Neural Network
- Linear Discriminant Analysis
- Bayesian classifiers (Markov)
- Nearest Neighbors
- Combined (Boosting, Voting, Stacking)
- Independent Component Analysis
- ...

Signal processing domain



Spatial Layout on the Cheap

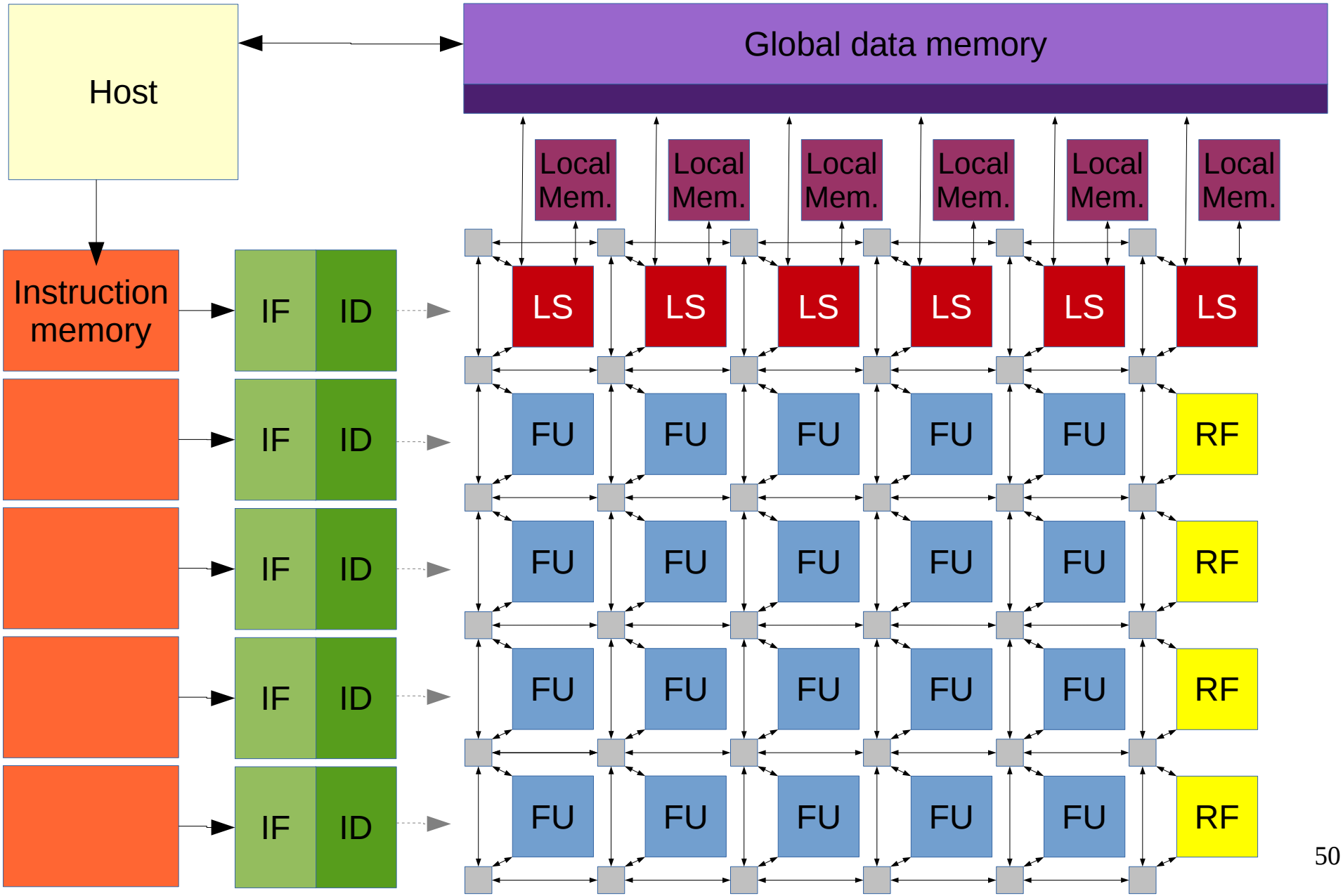
Gate-level granularity is not required for the application domain.

What our architecture does:

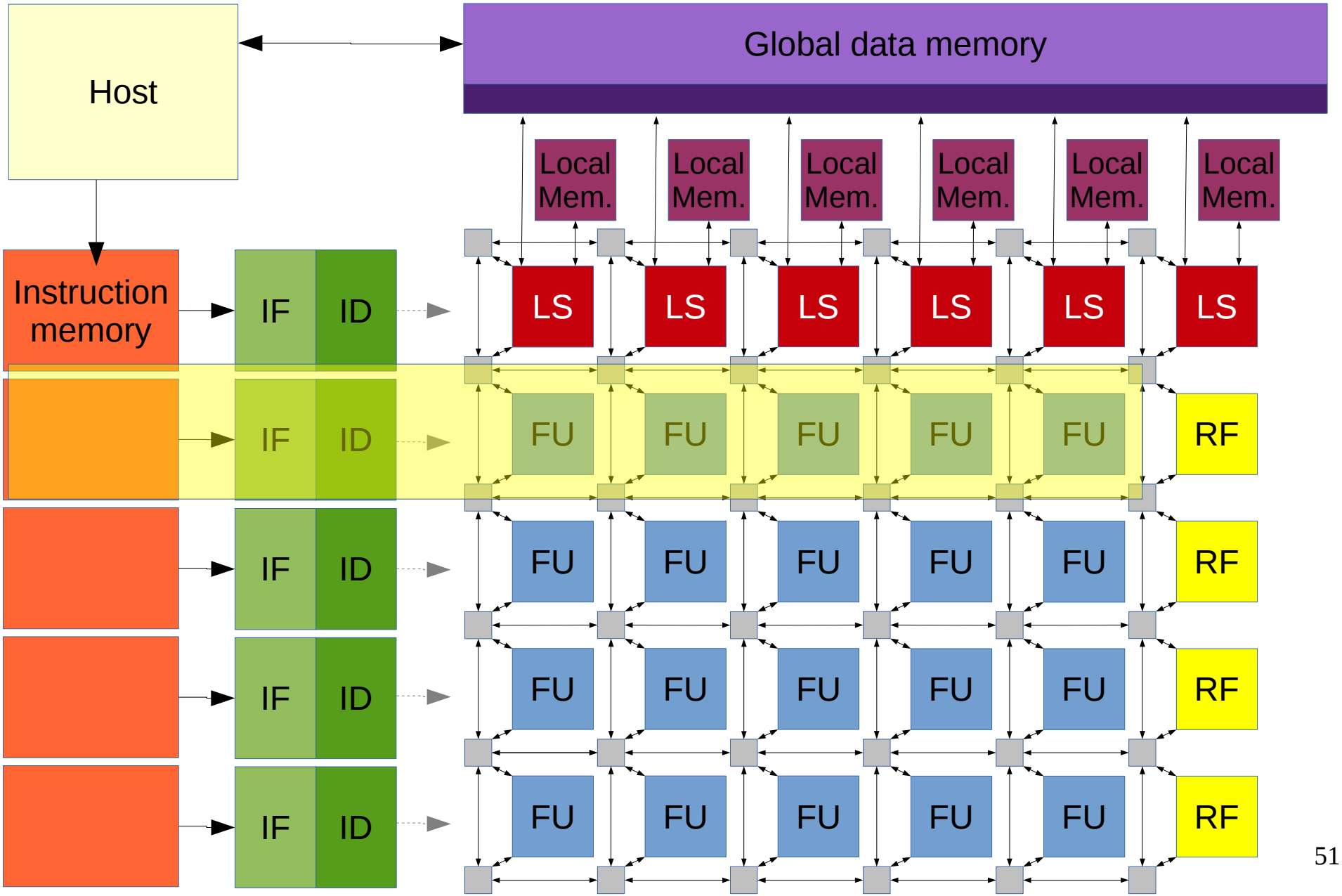
- Configure at functional unit level
- Statically route data-path and control-path (spatial layout) but allow instructions.

A specialized FPGA to build processors

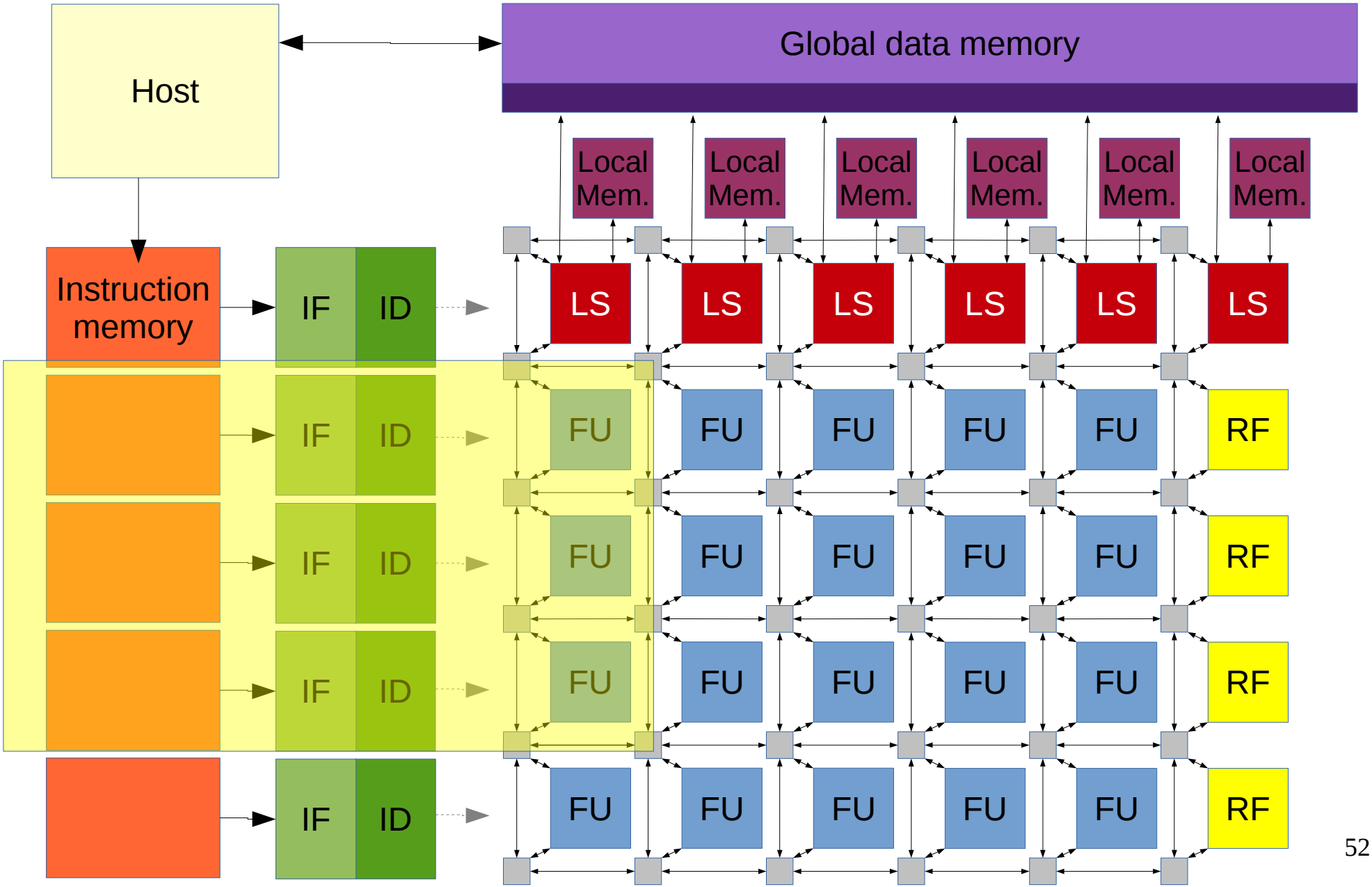
CGRA


Host
Global data memory
Local Mem.
Local Mem.
Local Mem.
Local Mem.
Local Mem.
Local Mem.
Instruction memory
IF
ID
LS
FU
RF

SIMD construction



VLIW construction

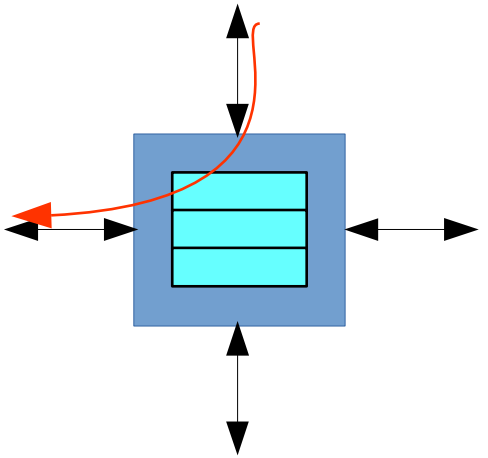


Reconfigurable architectures

- Lower static control power than FPGA
  - Higher granularity means less control bits
  - Reconfiguration will be faster
- Can adapt better to the application than VLIW
  - Only use the number of issue slots really required
  - Support spatial mapping and single-cycle loops
  - Unused units can be switched off.
- High number of operations per cycle
  - But as much as possible: the same instruction

Switch-boxes

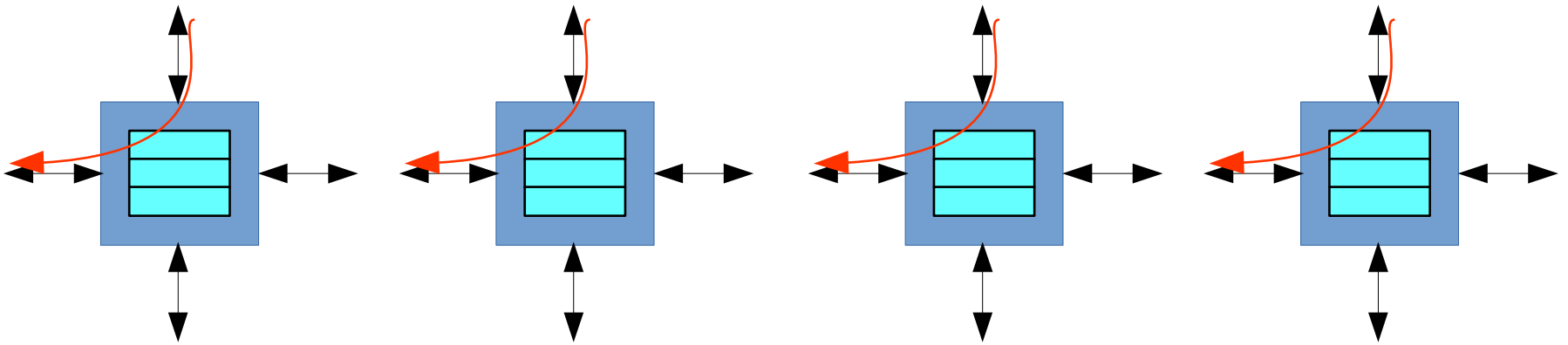
- Why do we save power compared to a FPGA?
- Routing a single wire through a FPGA switch-box:



= configuration memory

Switch-boxes

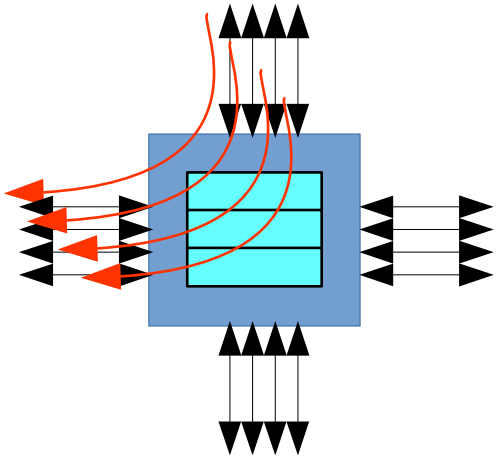
- Why do we save power compared to a FPGA?
- Routing a 4-wire bus through FPGA switch-boxes:



= configuration memory

Switch-boxes

- Why do we save power compared to a FPGA?
- Routing a 4-wire bus through a CGRA switch-box:



= configuration memory

Switch-boxes

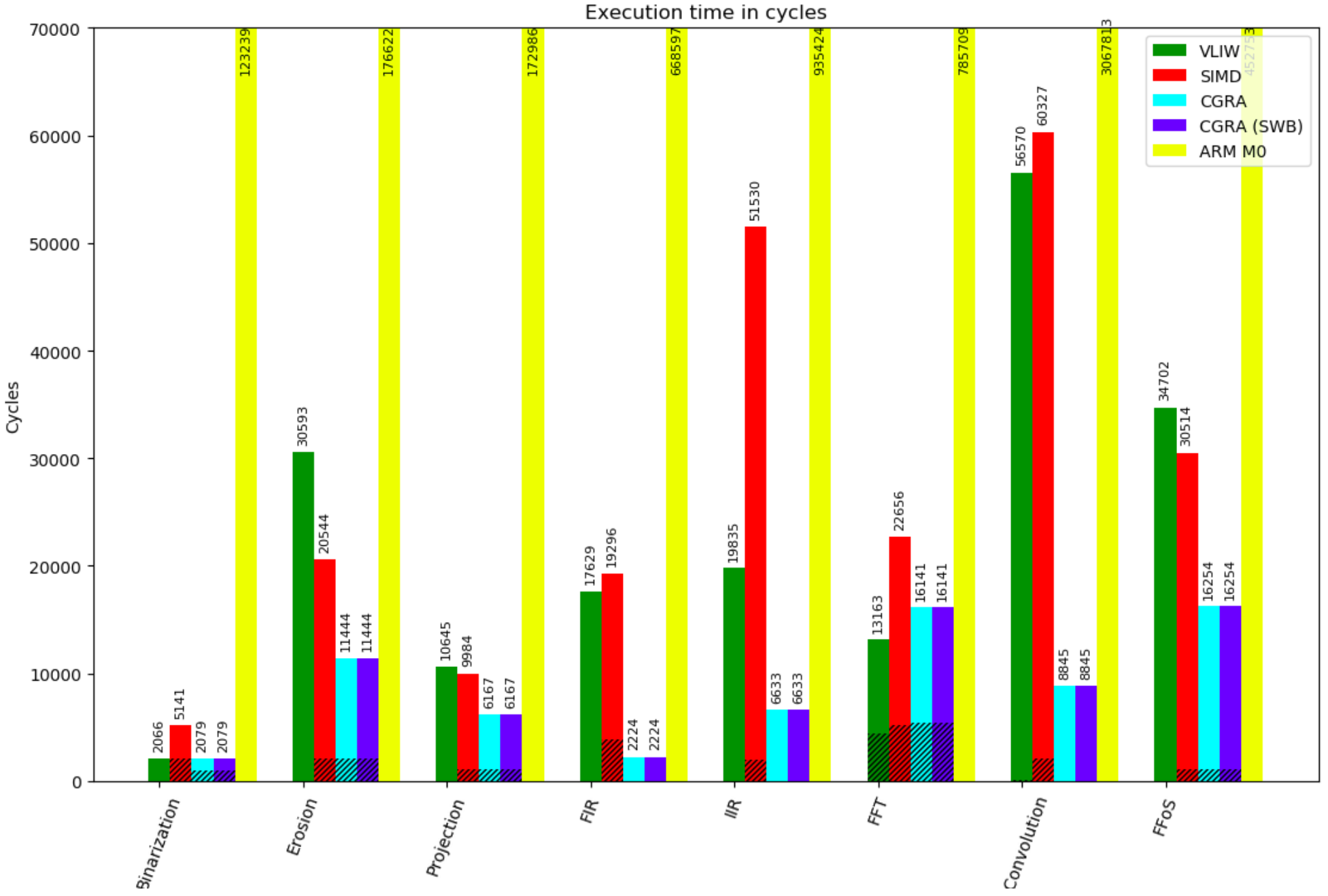
- Why do we save power compared to a FPGA?
- CGRAs have fewer switch-boxes than FPGAs:
  - Coarse granularity means bigger building blocks, less interconnect required and buses only.
  - Coarser blocks also means that fewer blocks are required.


= configuration memory

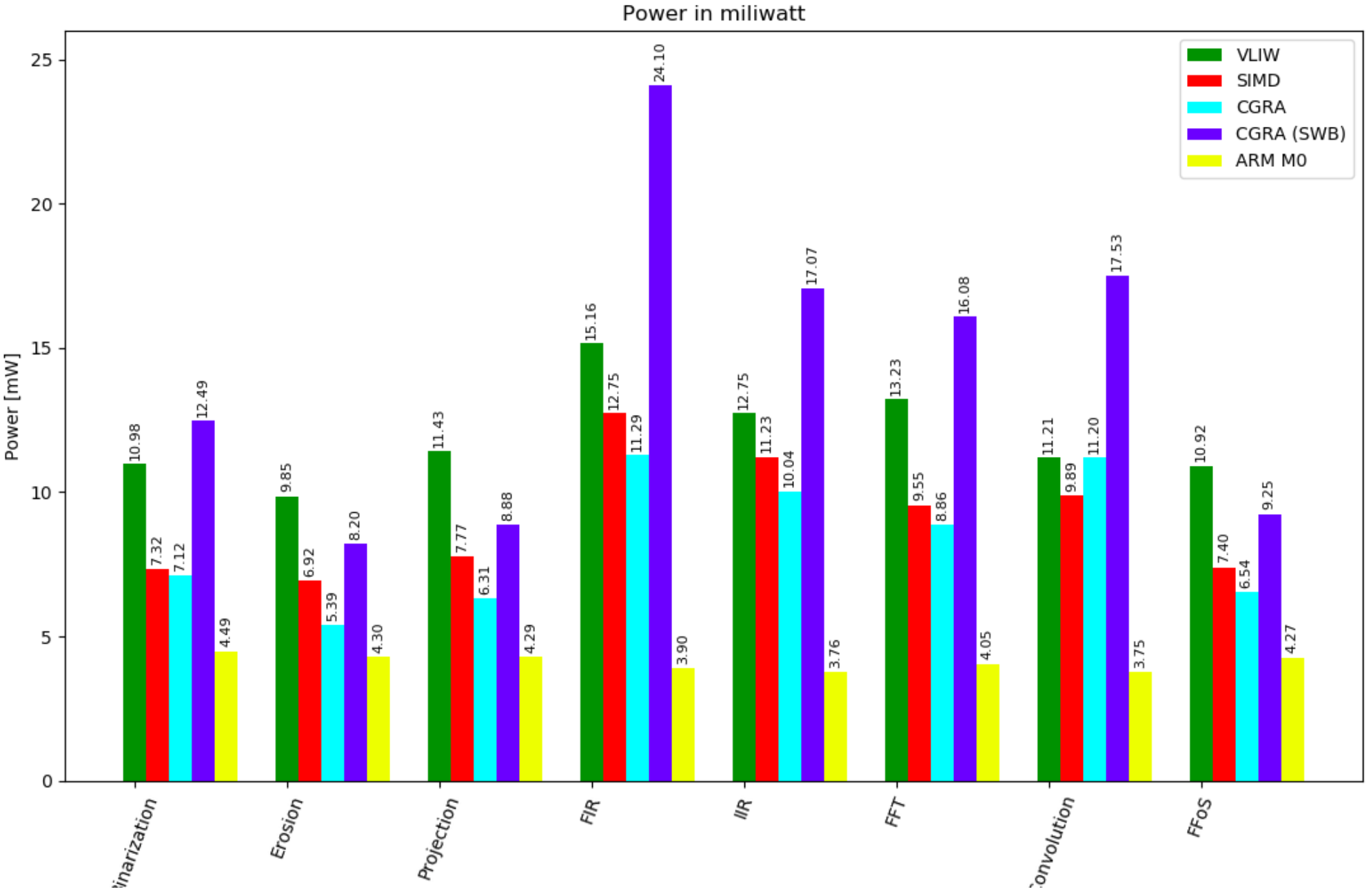
Some results

- “So, how good is this CGRA?” I hear you say…
- Well, let’s compare it with
  - A VLIW: 8-issue slots
  - A SIMD: 8-lanes + scalar processor
  - ARM Cortex M0+, low-power microcontroller
  - CGRA without switch-boxes. Direct connections, essentially an application specific processor.
  - CGRA with switch-boxes, same design for all benchmarks. Contains worst-case resources of all benchmarks.

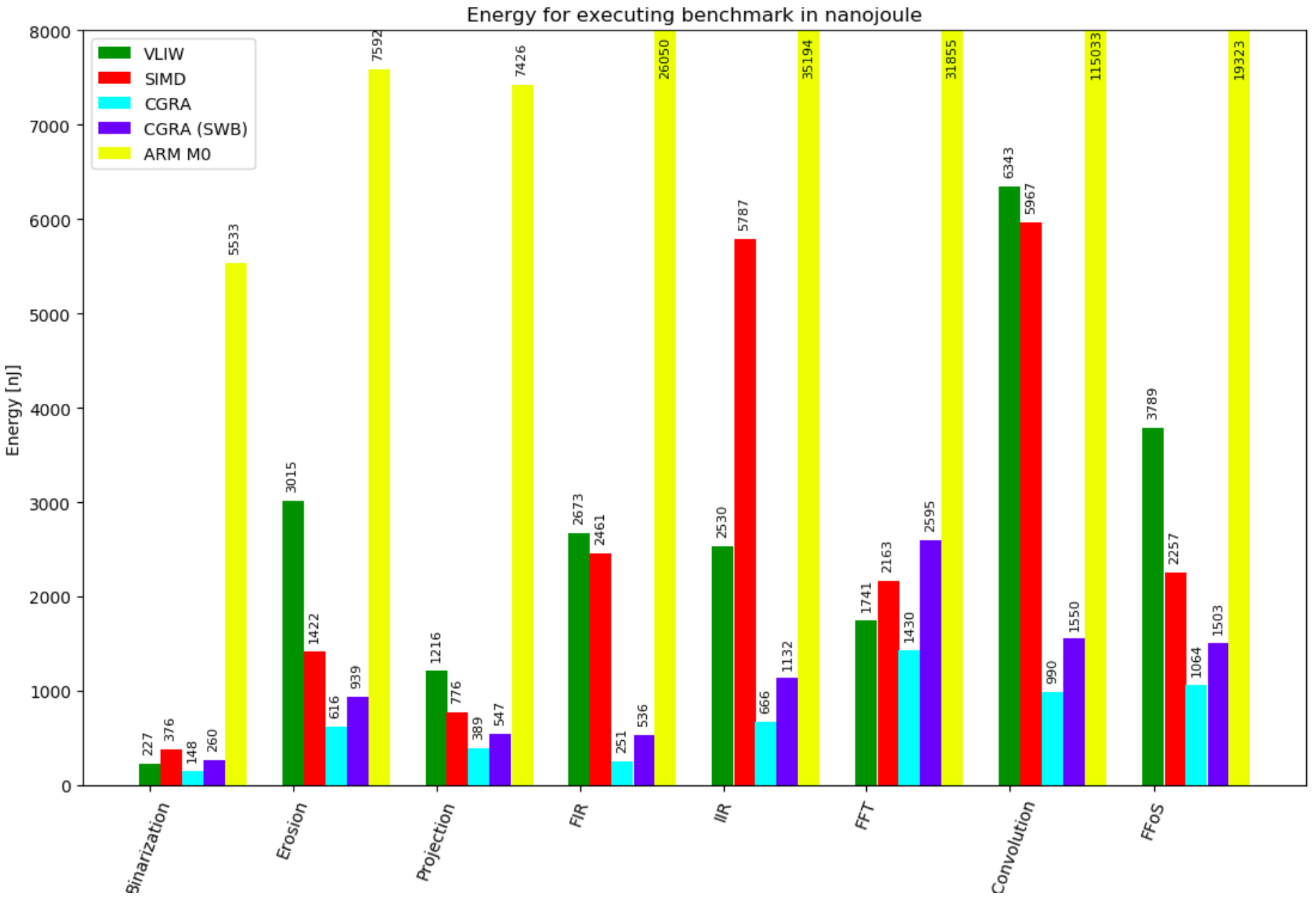
Execution time



Power


Power in miliwatt
Power [mW]
VLIW
SIMD
CGRA
CGRA (SWB)
ARM M0
0
5
10
15
20
25
10.98
7.32
7.12
12.49
4.49
9.85
6.92
5.39
8.20
4.30
11.43
7.77
6.31
8.88
4.29
15.16
12.75
11.29
24.10
3.90
12.75
11.23
10.04
17.07
3.76
13.23
9.55
8.86
16.08
4.05
11.21
9.89
11.20
17.53
3.75
10.92
7.40
6.54
9.25
4.27
Erosion
Projection
FIR
IIR
FFT
FFoS

Energy


Energy for executing benchmark in nanojoule
Energy [nJ]
VLIW
SIMD
CGRA
CGRA (SWB)
ARM M0
0
1000
2000
3000
4000
5000
6000
7000
8000
227
376
148
260
5533
3015
1422
616
939
7592
1216
776
389
547
7426
2673
2461
251
536
26050
2530
5787
666
1132
35194
1741
2163
1430
2595
31855
6343
5967
990
1550
115033
3789
2257
1064
1503
19323
Binarization
Erosion
Projection
FIR
IIR
FFT
Convolution
FFoS

Some results

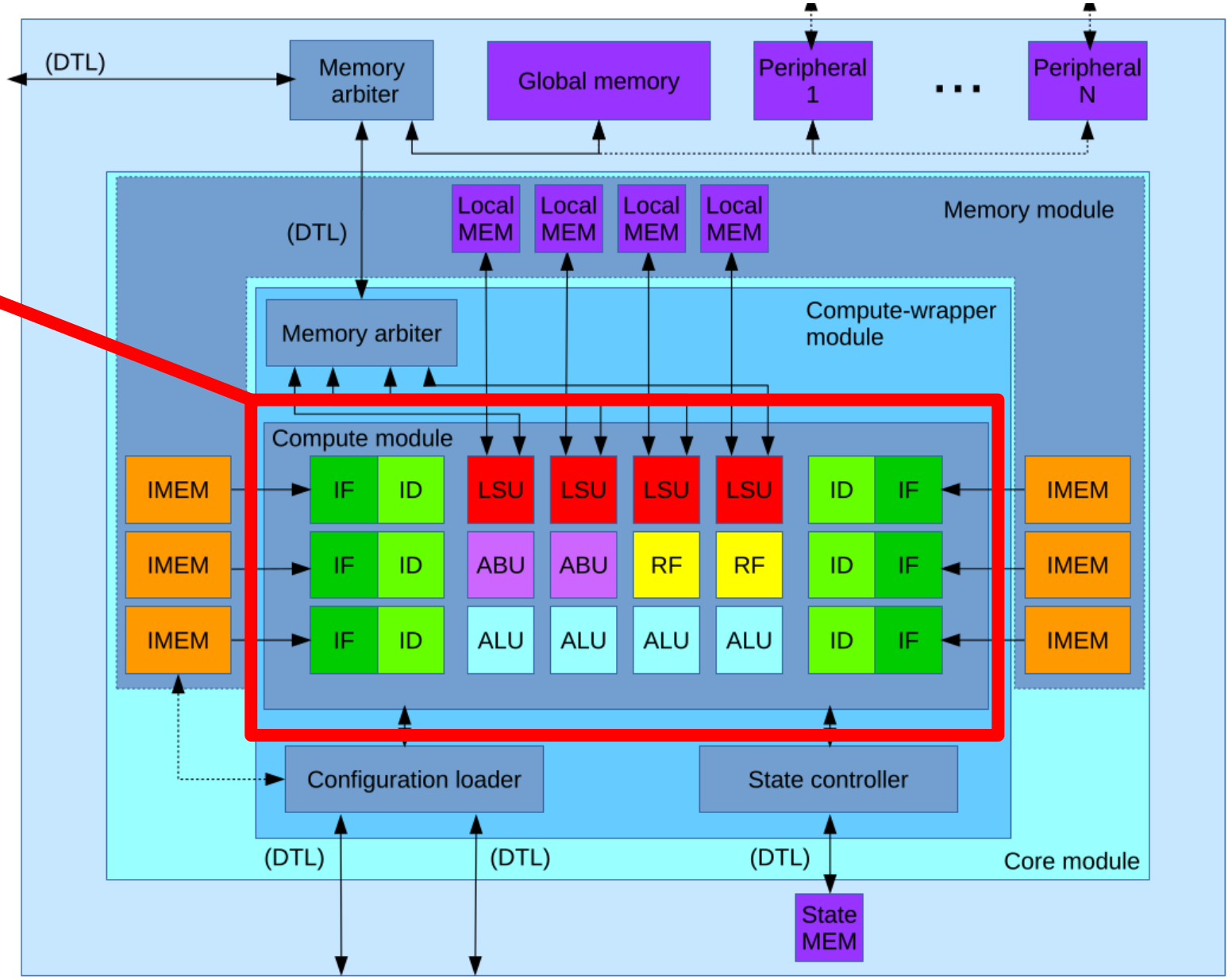
- What does this mean?
  - It is not bad if your design uses more power…
  - As long as the cycles get reduced more
    - (e.g. 50% more power, 33% less cycles → same energy)
  - Area is not shown here, experiments still ongoing
- You will get a energy/area prediction tool to use for your experiments
  - Less accurate but MUCH faster (minutes versus hours)
  - Performance is measured, and is accurate

How to build a processor

- The CGRA can be used to construct application specific processors at run-time.
- Several building blocks are available within the CGRA architecture.
- The building blocks can be interconnected using two networks:
  - Data network (between functional units)
  - Control network (between instruction decoders and functional units).

How to build a processor

Reconfigurable section



How to build a processor

Instruction Fetch/Decode (IF/ID)

Load Store Units (LSU)

Register Files (RF)

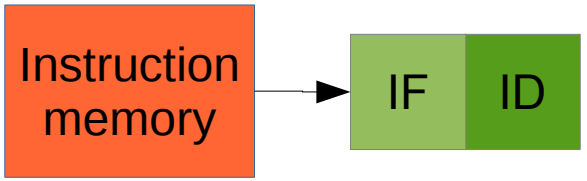
Memories (MEM)

Functional Units (FU)

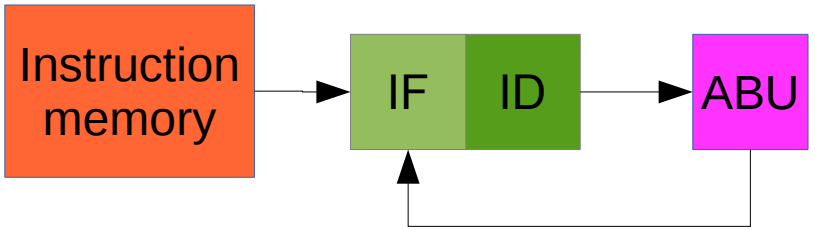
- Add, subtract, bitlevel operations
- Multiplication
- Accumulator
- …

- Switchboxes

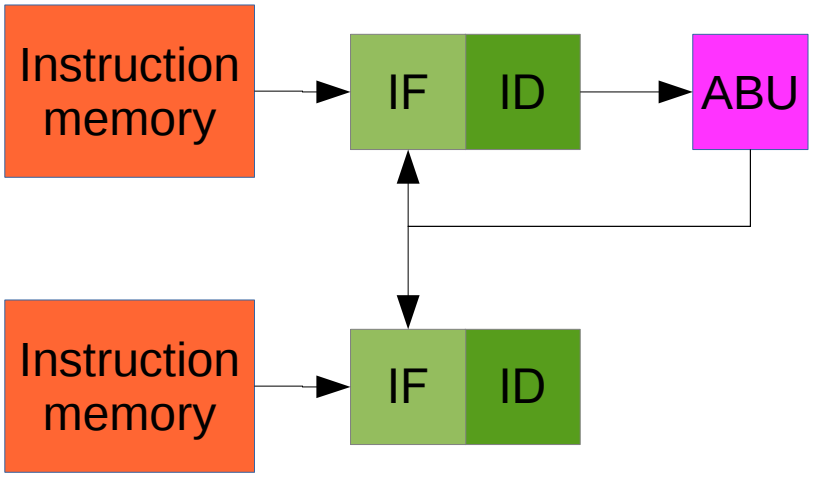
How to build a processor



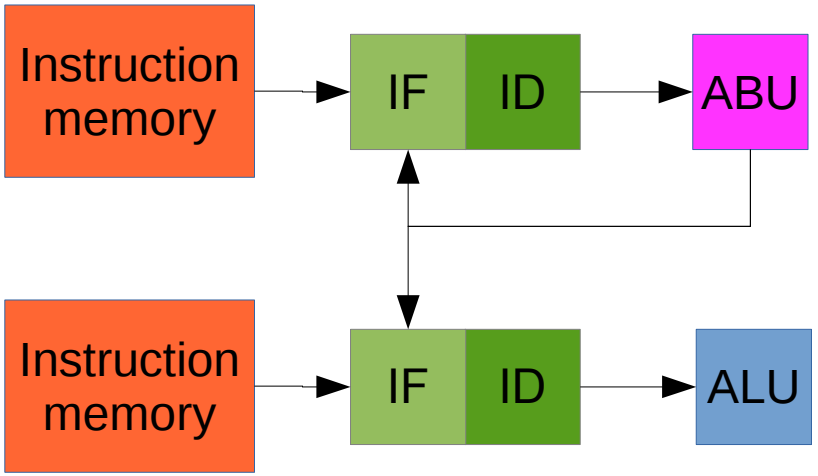
How to build a processor



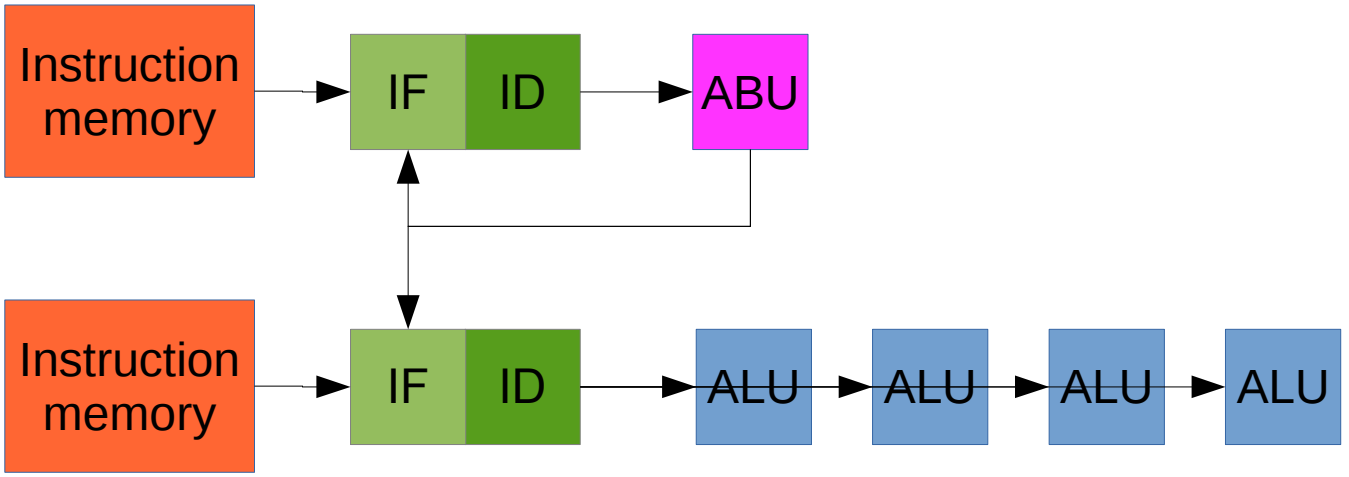
How to build a processor



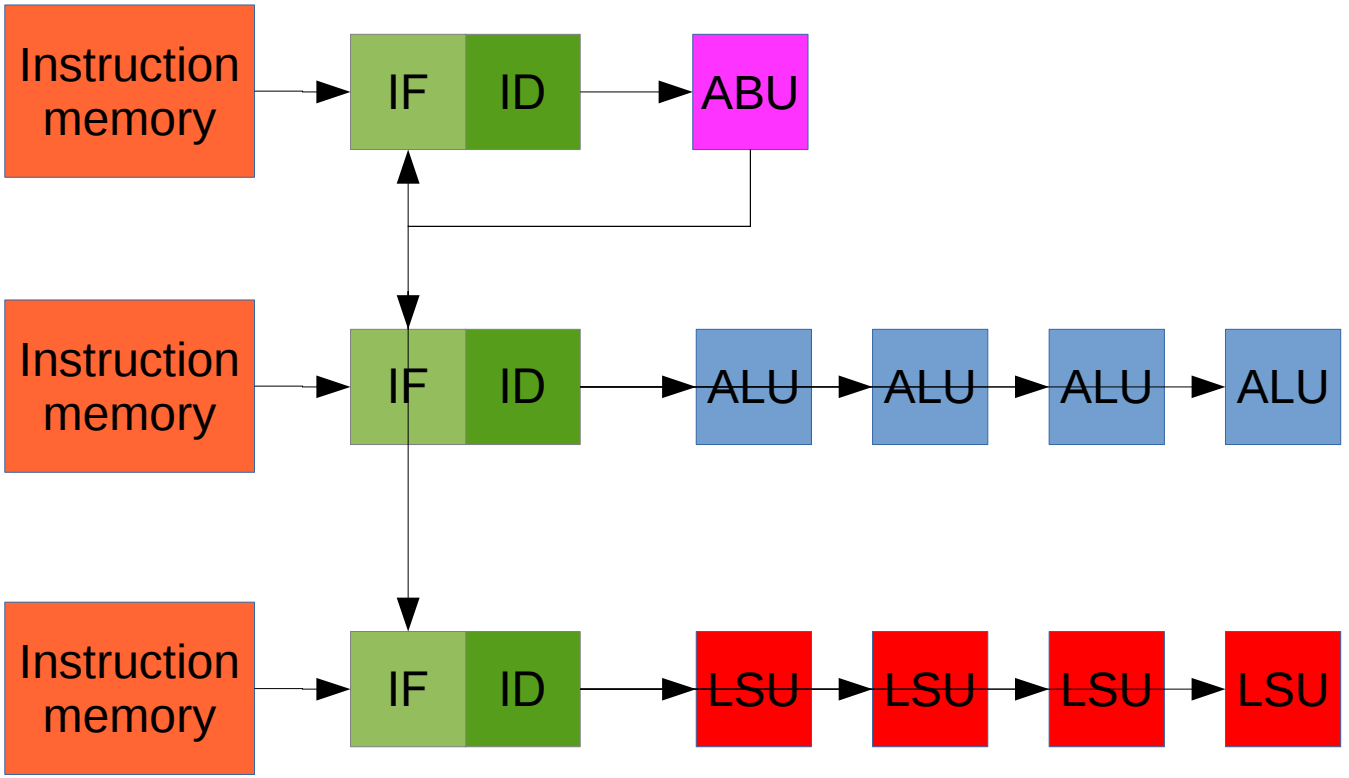
How to build a processor



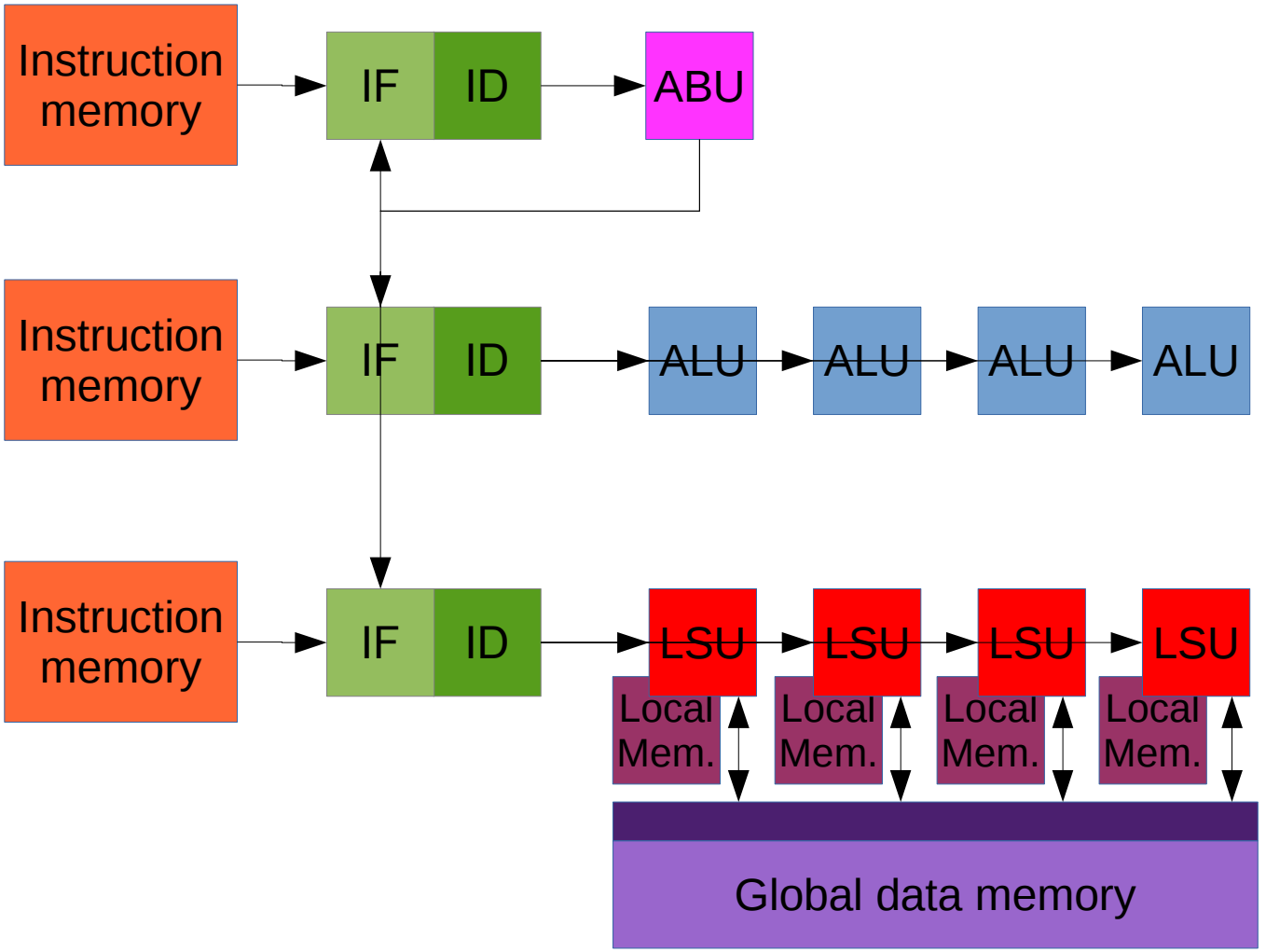
How to build a processor



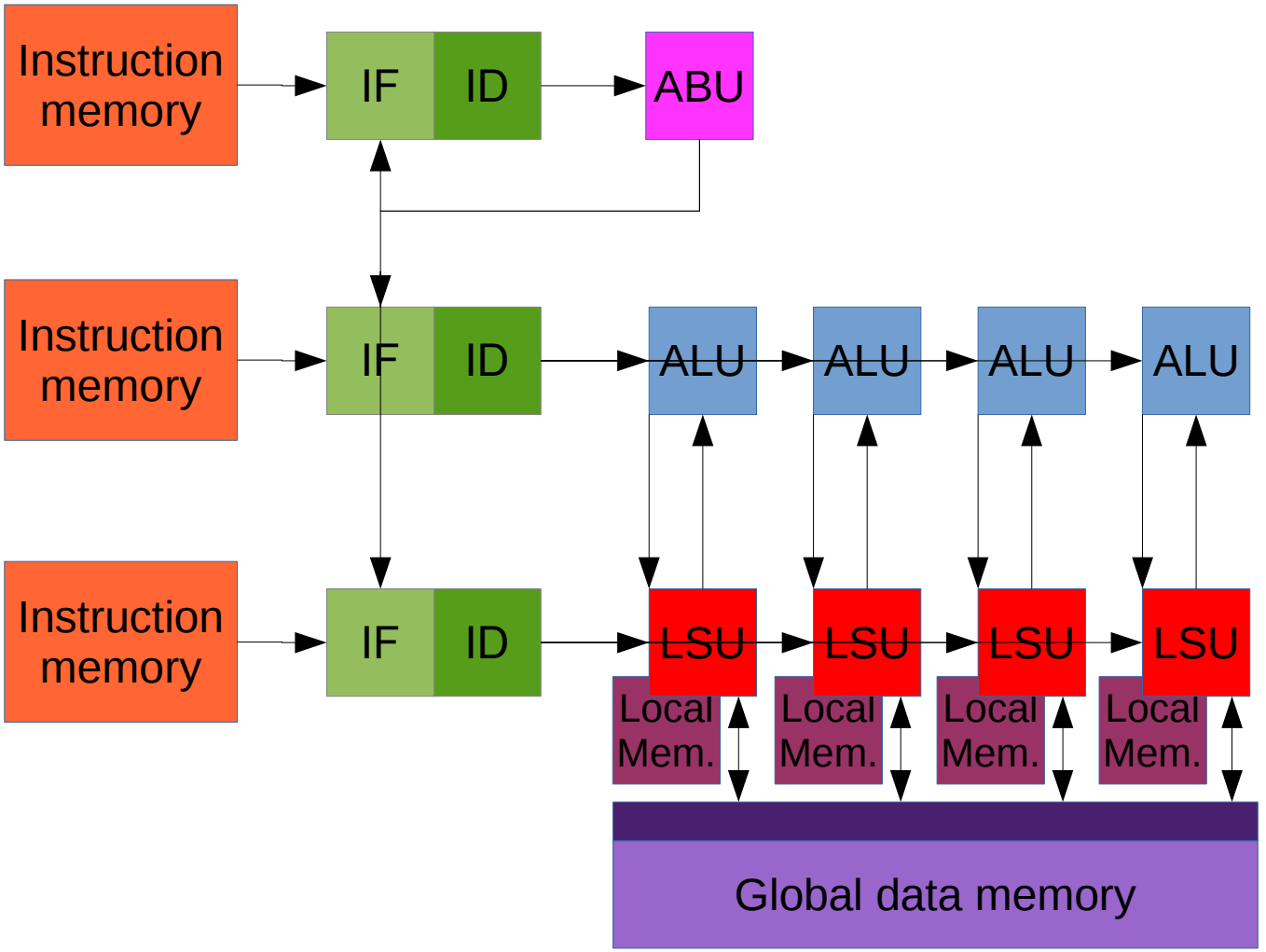
How to build a processor



How to build a processor



How to build a processor

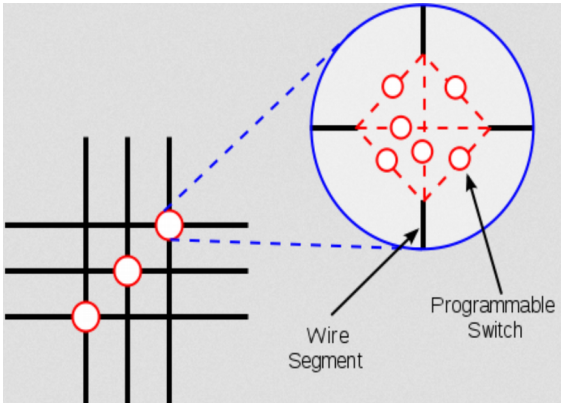


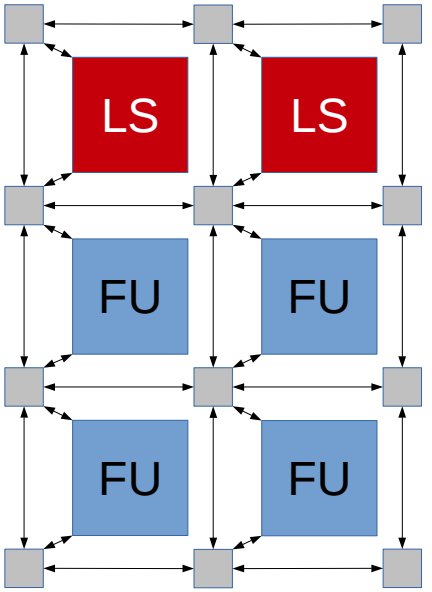
How to build a processor

- Architecture is specified with XML file
- Two versions:
  - Static: all connections are fixed at design time
  - Dynamic: connections can be configured at runtime
- The assignment will use the static version

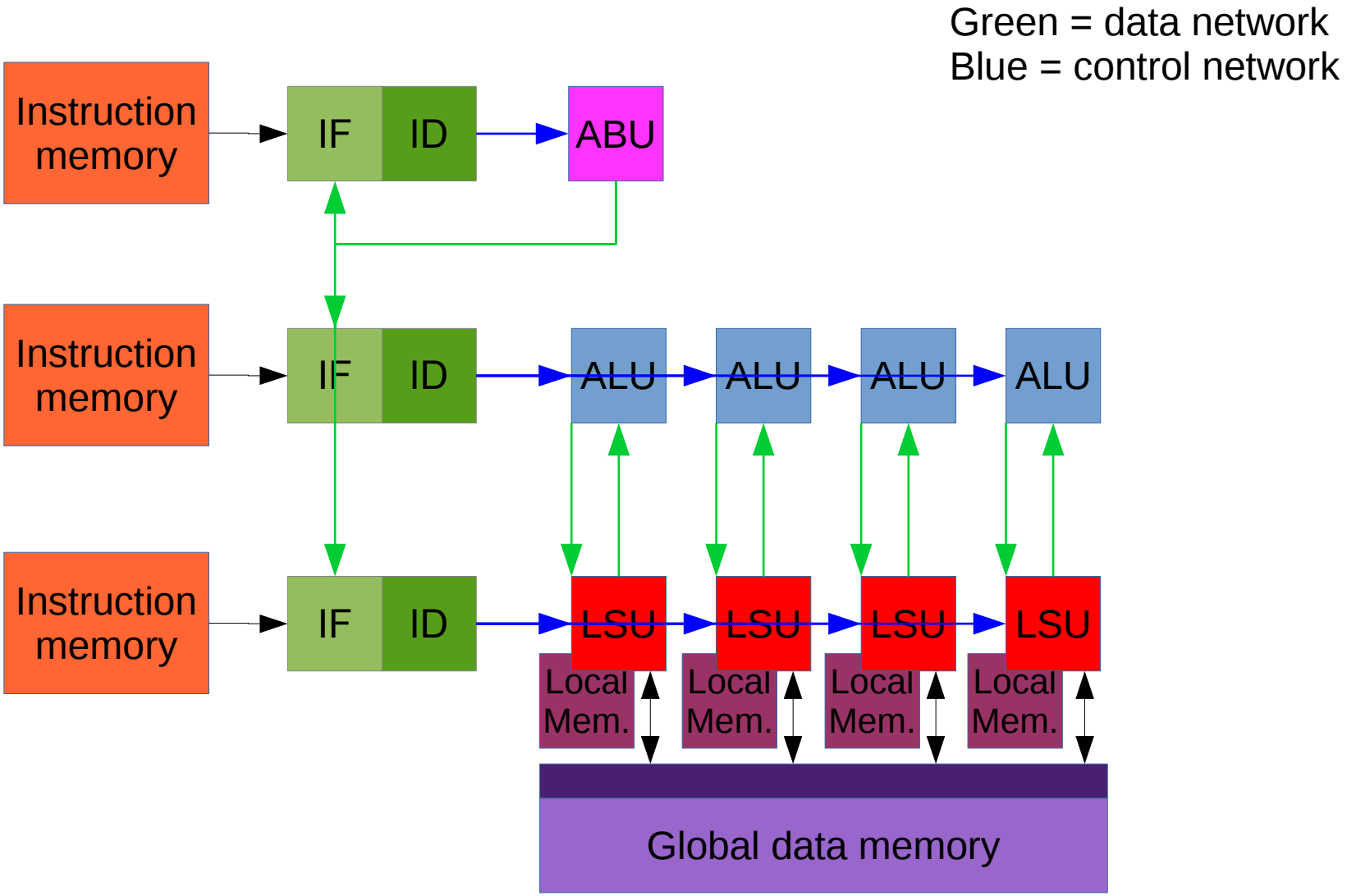
Dynamic CGRA

- Connections can be changed at run-time
  - Very similar to FPGA





Dynamic CGRA

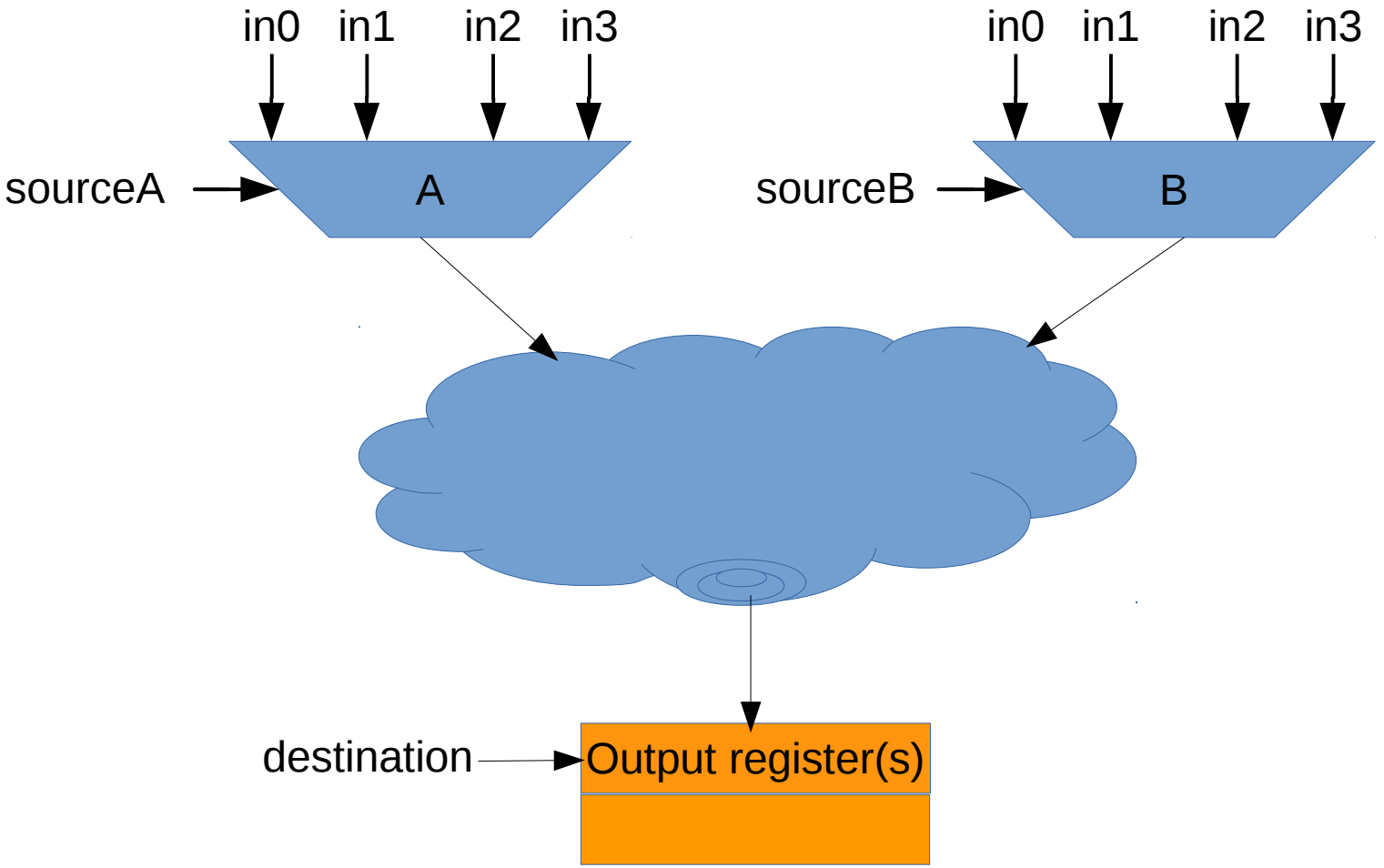


Functional units

- As mentioned before, we have several:
  - ALU Arithmetic Logic Unit
  - RF Register File
  - LSU Load-Store Unit
  - ABU Accumulate Branch Unit
  - MUL Multiplier
  - IU Immediate Unit

Functional units

- Most units have 4 inputs and 2 outputs



Functional units

- Source inputs and destination outputs specified in instruction.
- Instruction usually in the form:

```
opcode dest, inA, inB
```

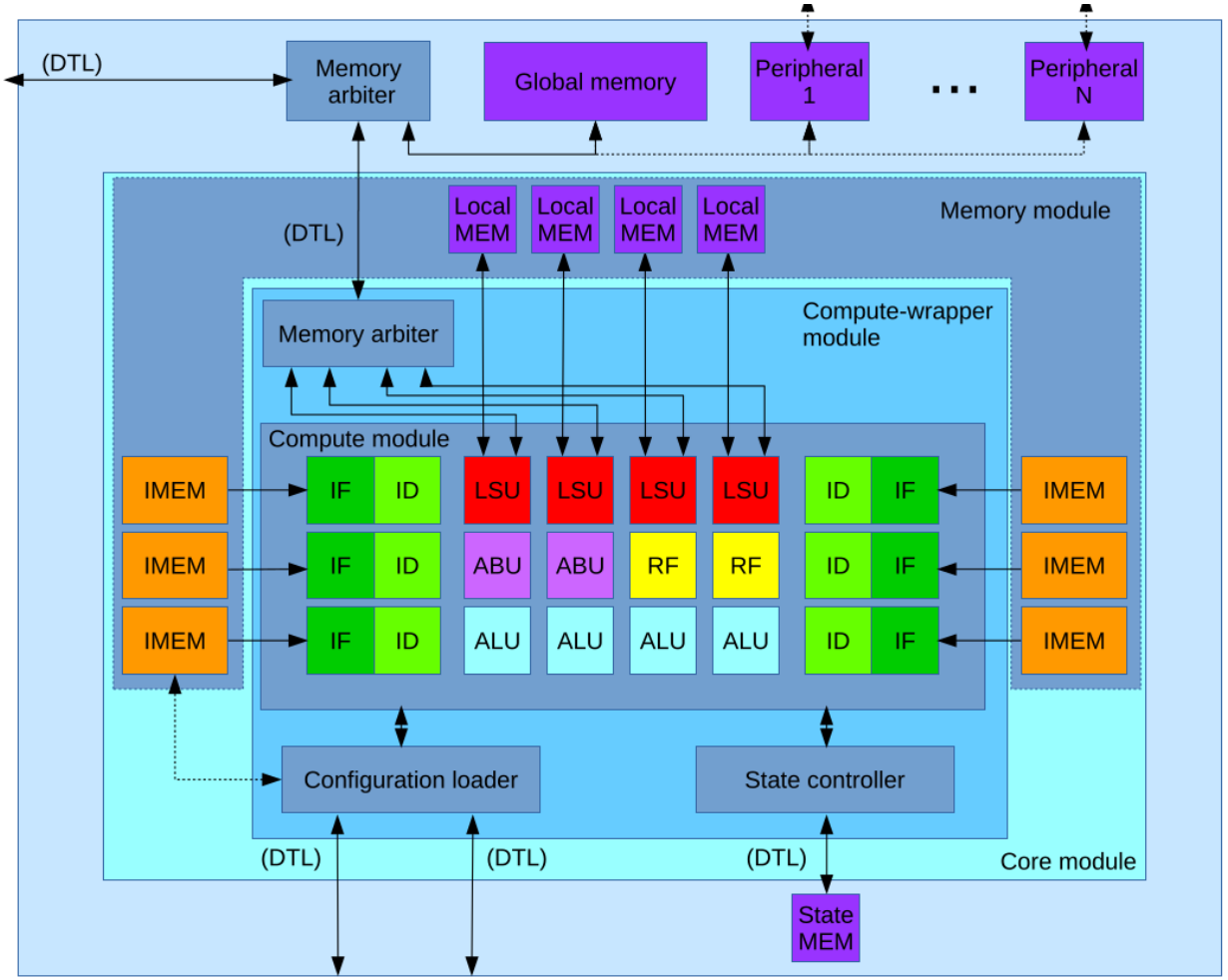
- Each output register is a source on the network.

Functional units

- Two highlighted Fus:
  - ABU, this unit can be used for both program counter generation and as an accumulator.
  - IU, the immediate unit can be used to generate constants on the data network.

Memory

- There are two memory ‘levels’
  - Global memory
  - Local memory



Memory

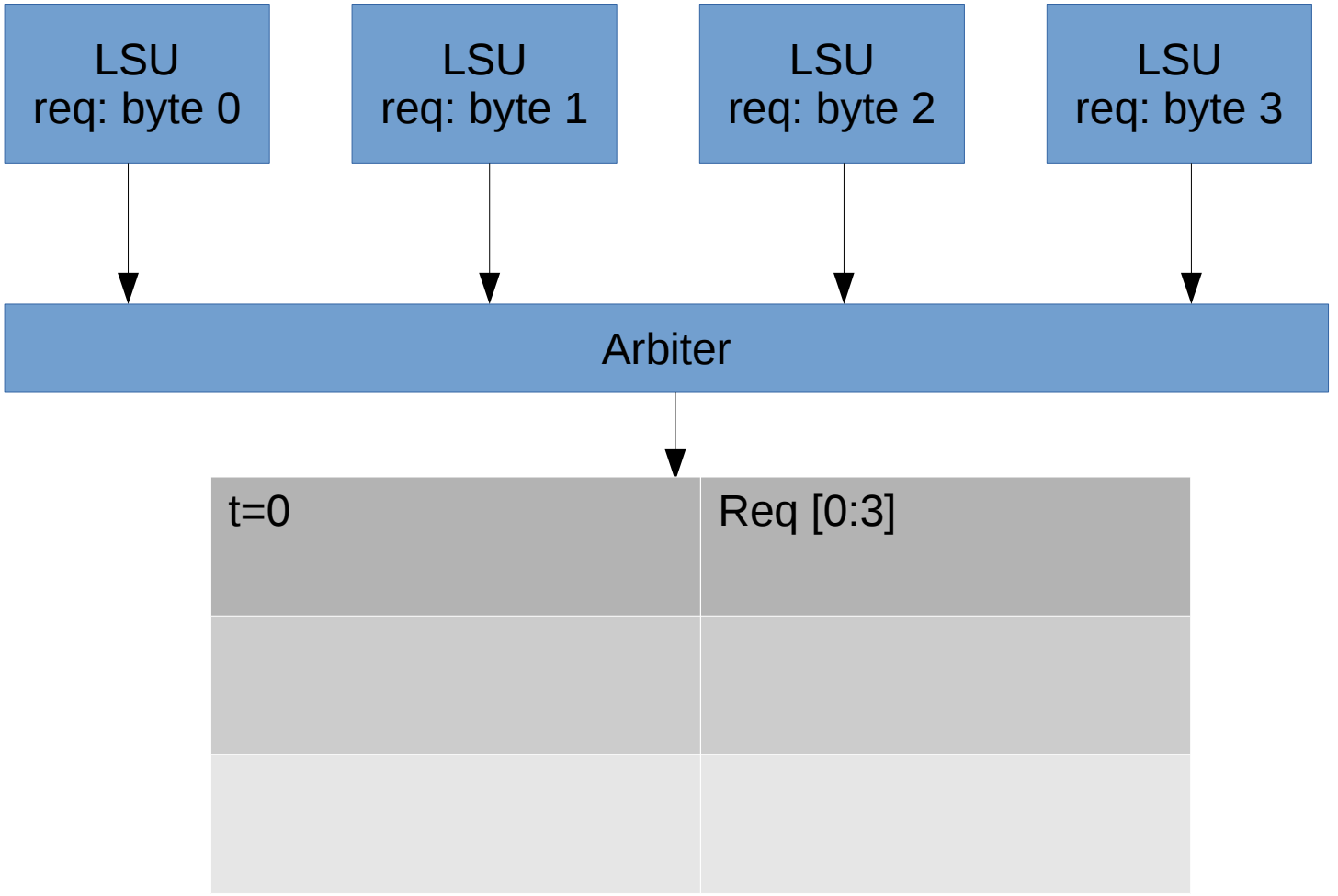
- The local memories are private to each LSU (meaning that one LSU can not read the local memory of another LSU)
- The data-bus is 32-bit wide and can perform a read AND write operation in a single cycle.
- Three data sizes are supported:
  - Byte
  - Half-word
  - Word

Memory

- The global memory is shared among all LSUs
- The data-bus is 32-bit wide and can perform a read OR a write. There is a few cycles latency on this bus (processor will stall).
- Any simultaneous reads and writes will be sequentialized, writes first.
- If multiple read or write requests can be coalesced they will be performed as one memory operation.

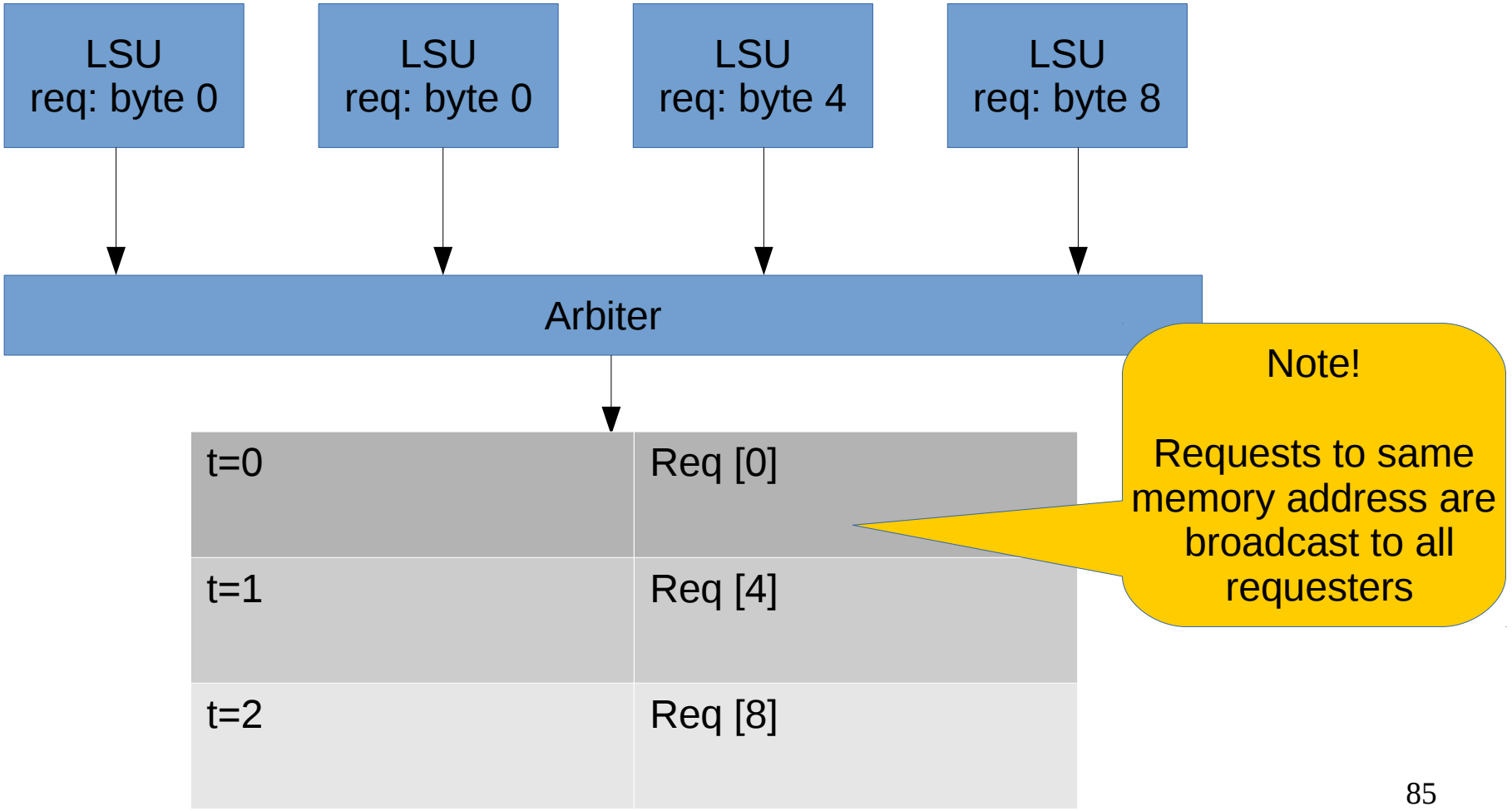
Memory

- Coalesced read:



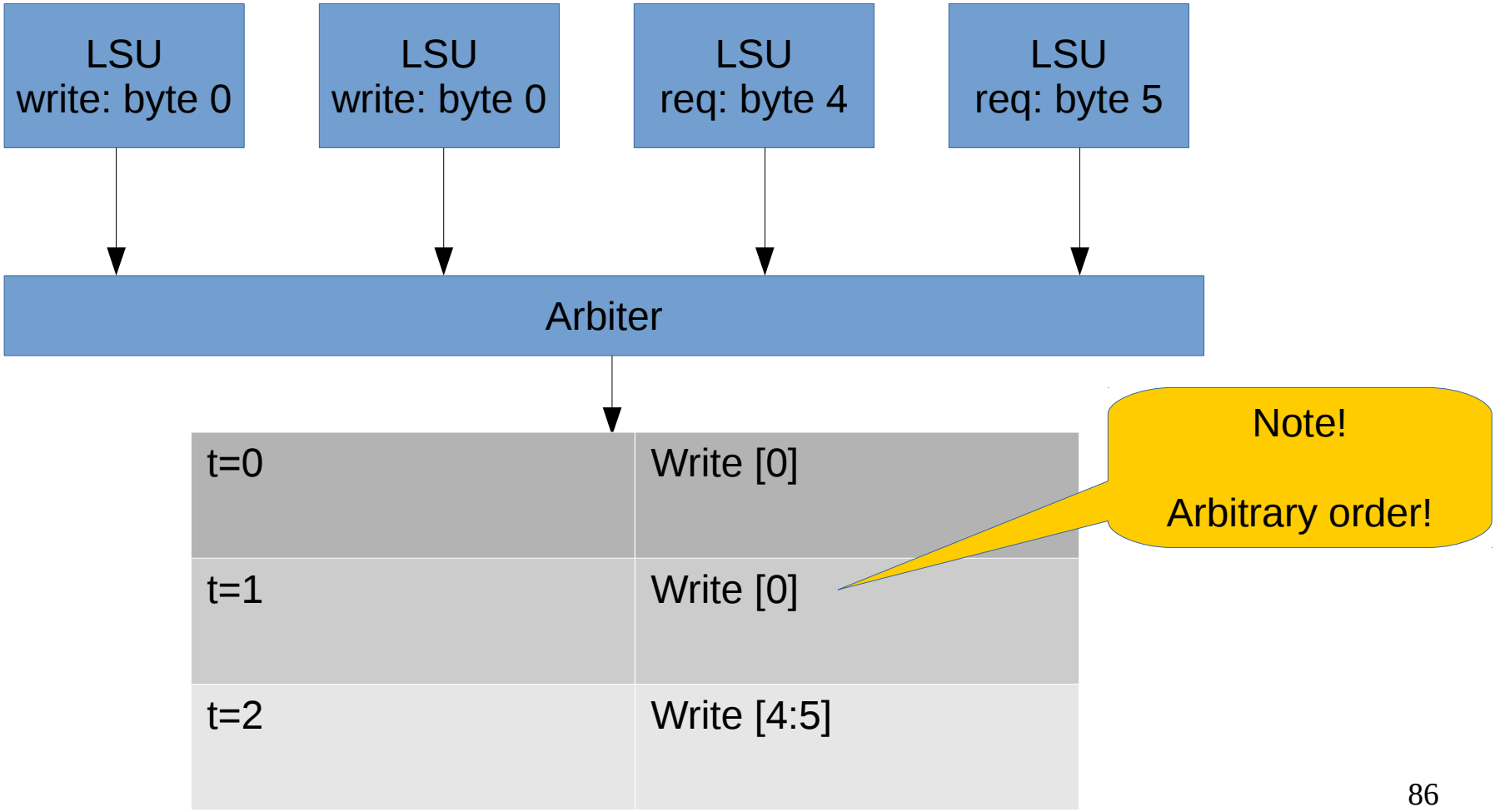
Memory

- (Partially) un-coalesced read:



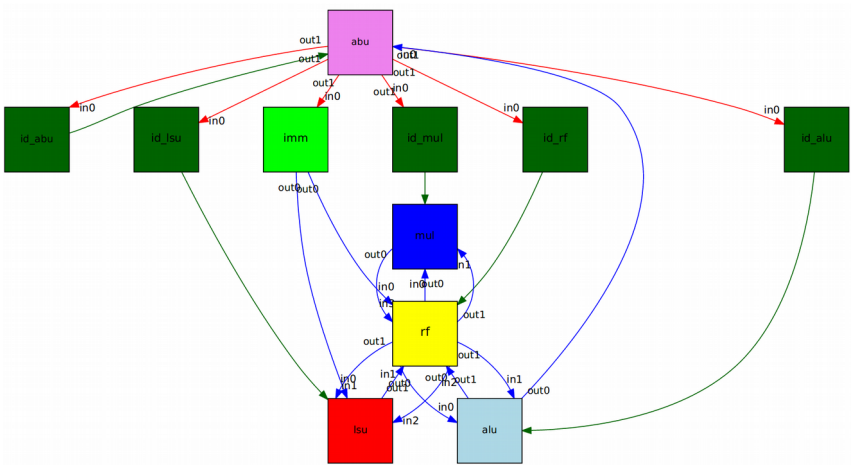
Memory

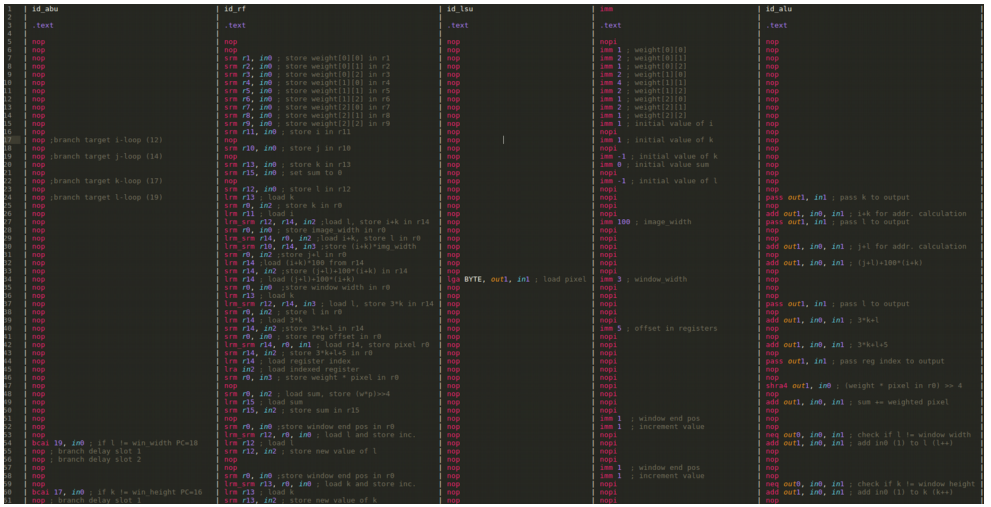
- Similar for writes



Using the CGRA

- Two things are required:
  - Architecture description
  - Program



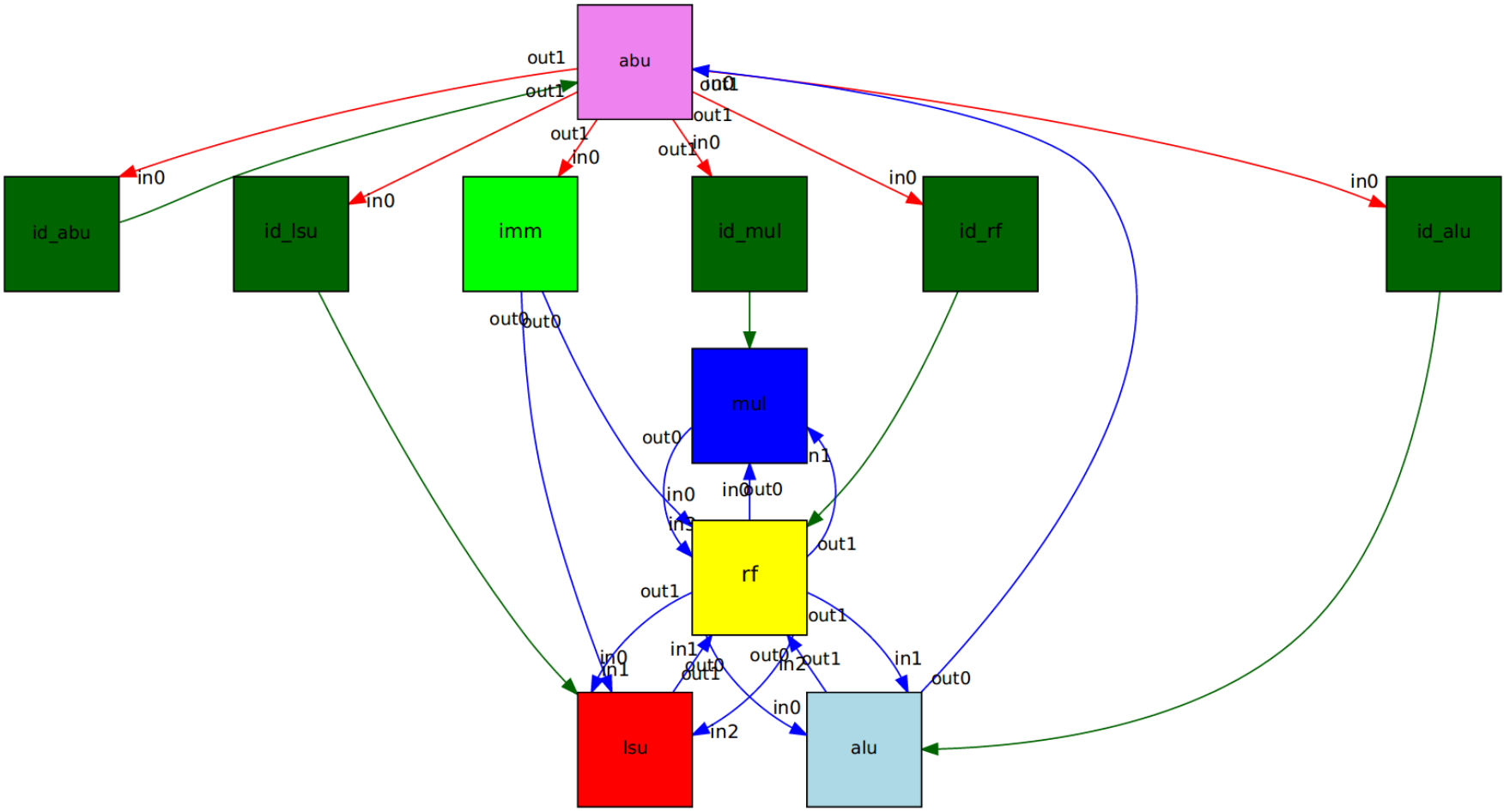


Architecture description

```
        <fu type="ID" name='id_lsu'>
            <input index="0" source="abu.1"/>
        </fu>
        <fu type="ID" name='id_alu'>
            <input index="0" source="abu.1"/>
        </fu>
        <fu type="ID" name='id_mul'>
            <input index="0" source="abu.1"/>
        </fu>
        <fu type="ID" name='id_rf'>
            <input index="0" source="abu.1"/>
        </fu>

        <fu type="LSU" name='lsu' ID="id_lsu"> <!-- FUs -->
            <input index="0" source="imm.0"/>
            <input index="1" source="rf.1"/>
            <input index="2" source="rf.0"/>
        </fu>
        <fu type="ALU" name='alu' ID="id_alu" config="1">
            <input index="0" source="rf.0"/>
            <input index="1" source="rf.1"/>
        </fu>
        <fu type="MUL" name='mul' ID="id_mul" config="1">
            <input index="0" source="rf.0"/>
            <input index="1" source="rf.1"/>
        </fu>
        <fu type="RF" name='rf' ID="id_rf">
            <input index="0" source="imm.0"/>
            <input index="1" source="lsu.1"/>
            <input index="2" source="alu.1"/>
            <input index="3" source="mul.0"/>
        </fu>
        <fu type="ABU" name='abu' ID="id_abu" config="1">
            <input index="0" source="alu.0"/>
        </fu>
        <fu type="IU" name='imm'>
            <input index="0" source="abu.1"/>
        </fu>
    </functionalunits>
```

Architecture description



Application

- We would liked to give you a compiler...
  - Still in development.
  - Works, but results not (yet) so efficient.
  - If you are interested in helping out with the compiler follow the 5LIM0 (Parallelism, Compilers, and Platforms) course and contact us!
- Programming is done in an assembler dialect
  - We call it PASM: Parallel Assembler

Application

```
id_alu

.text

nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
pass out1, in1 ; pass k to output
nop
add out1, in0, in1 ; i+k for addr. calculation
pass out1, in1 ; pass l to output
nop
nop
add out1, in0, in1 ; j+l for addr. calculation
nop
add out1, in0, in1 ; (j+l)+100*(i+k)
nop
nop
nop
nop
pass out1, in1 ; pass l to output
nop
add out1, in0, in1 ; 3*k+l
nop
nop
add out1, in0, in1 ; 3*k+l+5
nop
pass out1, in1 ; pass reg index to output
nop
nop
shra4 out1, in0 ; (weight * pixel in r0) >> 4
nop
add out1, in0, in1 ; sum += weighted pixel
nop
nop
nop
neq out0, in0, in1 ; check if l != window width
add out1, in0, in1 ; add in0 (1) to l (l++)
```

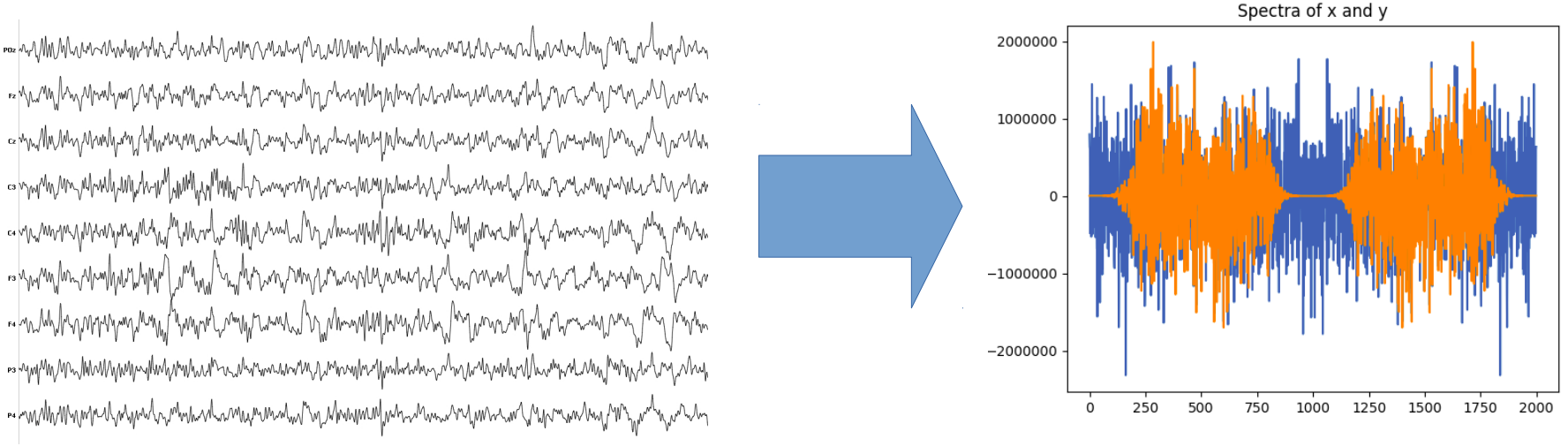
```
id_mul

.text

nop    ; always leave this line with only NOPs
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
mullu out0, in0, in1 ; (i+k)*img_width
nop
nop
nop
nop
nop
nop
mullu out0, in0, in1 ; k*window_width
nop
nop
nop
nop
nop
nop
mullu out0, in0, in1 ; weight * pixel
nop
nop
nop
nop
nop
nop
nop
```

Your assignment

- You will get a naive implementation for a Butterworth filter.



- Your job is to make a trade-off between energy, area and performance

Your assignment

- You can:
  - Modify the architecture:
    - Implement data-level parallelism
    - Implement instruction-level parallelism
    - Use bypassing
    - Use other nice hardware features
  - Modify the application:
    - There are algorithm level optimizations possible
    - To make use of the architecture changes

Your assignment

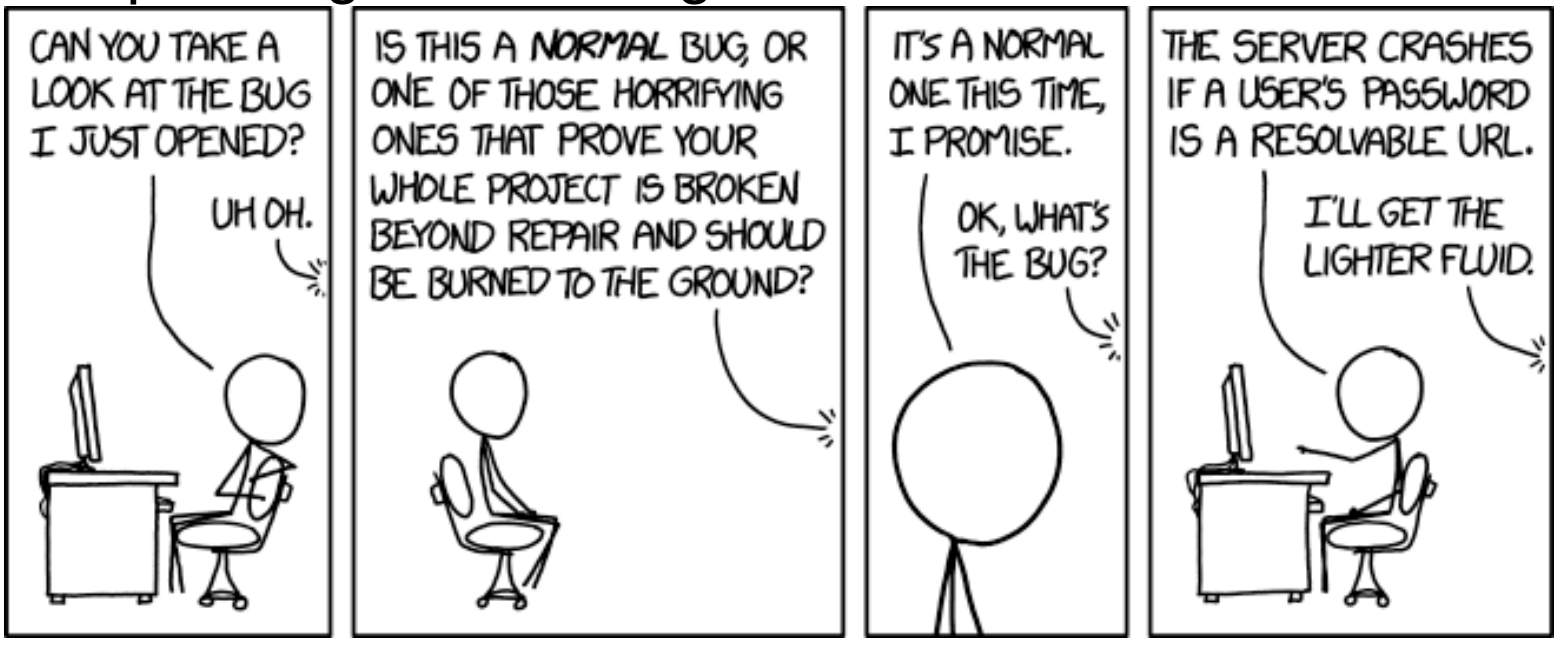
- The assignment document will describe everything in more detail.
  - Additional documentation and files can be found on the Oncourse page for this course.
- Tools are available to make energy, area and performance estimates.

Your assignment

- You can either work on our servers
- Or work at your own laptop
  - Virtual machine with Linux and required tools
    - Be aware that the VM is (much) slower than the servers, not only because of ‘hardware’, but also because of the simulator license restricting speed.
  - Directly on your laptop, but Linux only
    - If you want to port it to Windows be our guest!

One more thing...

- This is a research architecture…
  - Bugs will be present.
  - You will be among the very first users.
  - We will reward the best bug with a Walhalla beer card.
    - Bugs should be reproducible.
    - Report bugs on the bugs section on the forum



Interested?

- We are always looking for good master students for:
  - Internships
  - Master projects
- In the fields of:
  - Silicon level optimizations (gates, transistors, ...)
  - Hardware development (Verilog, functional units, …)
  - Toolflow development (Compilers, place & route, …)
- Look at:
  - www.cgra.nl for some student projects
  - Welcome to discuss your own ideas!