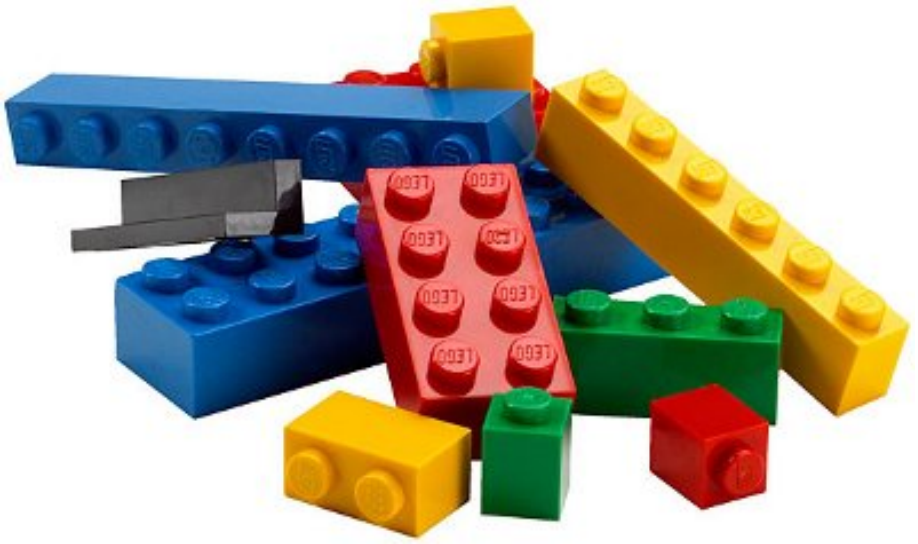
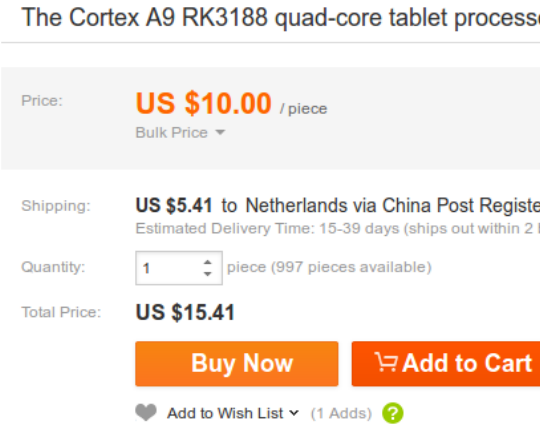
Accelerators and Coarse Grained Reconfigurable Architectures



Mark Wijtvliet
m.wijtvliet@tue.nl

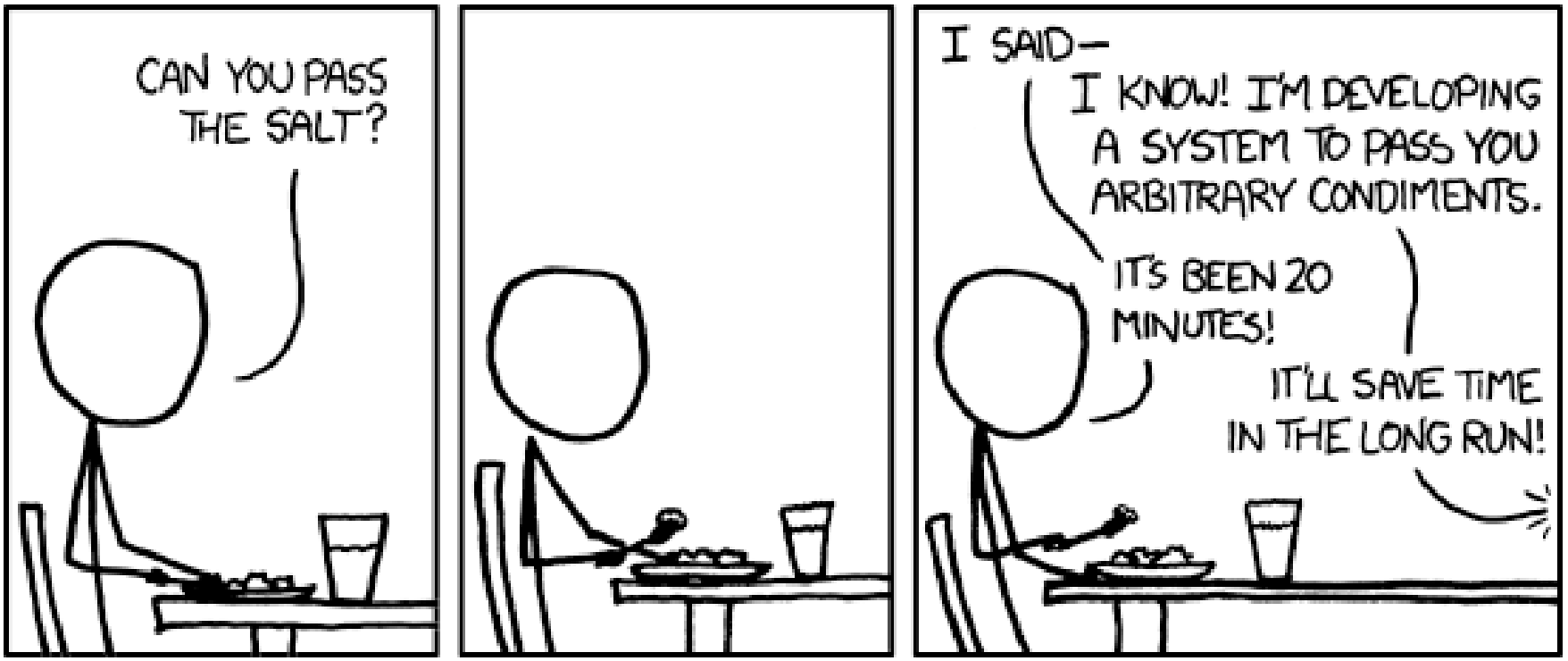
General purpose processors

- **Run mixed application types**
  - Operating system
  - Webserver
  - Games
  - Office applications
- **Cheap**
  - E.g. Raspberry Pi's Broadcom processor
  - Many others
- Generally **easy to program**
  - Well developed compilers and tool-flow.
  - Programming model hardly changed over the years.



General purpose processors

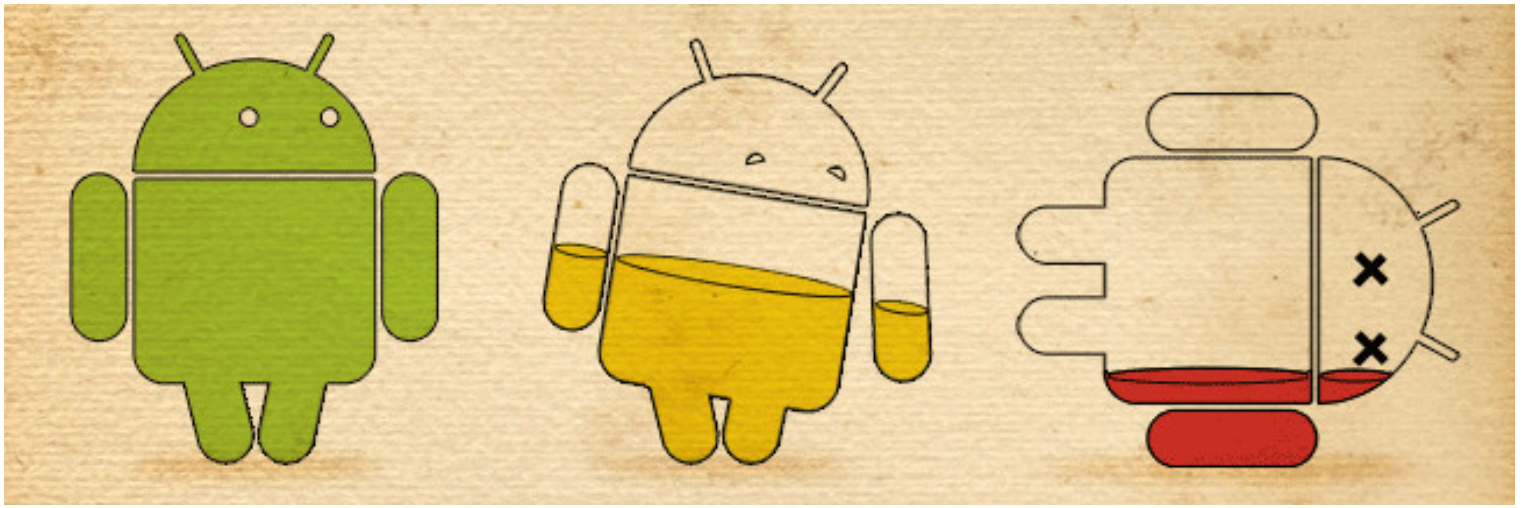
- They work great
- So why this accelerator talk?



[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

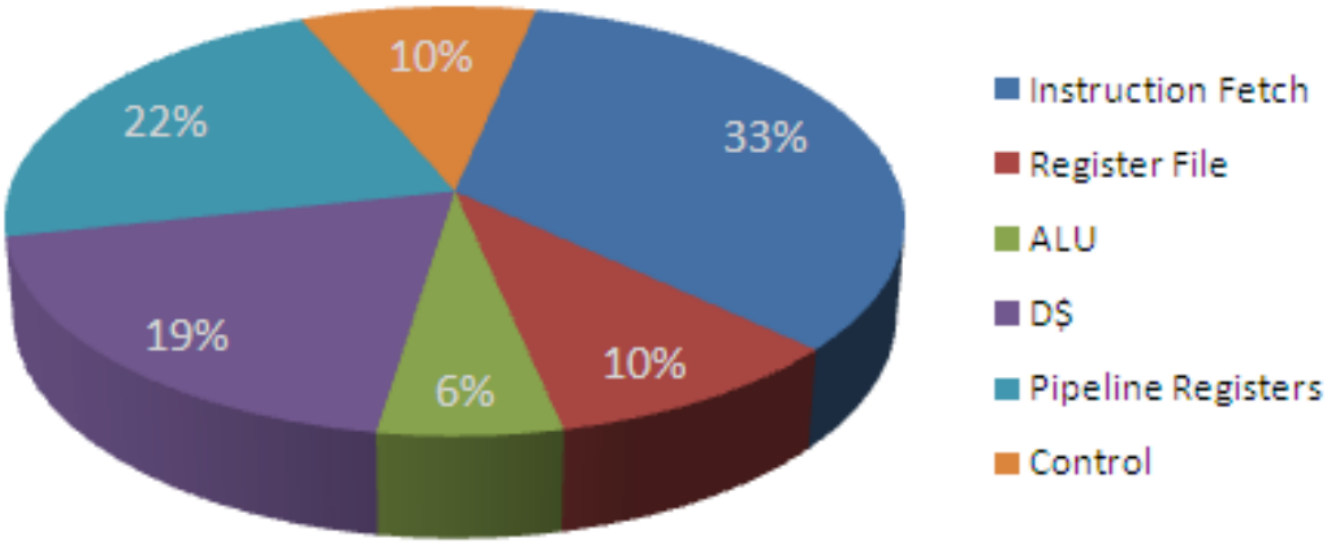
Example: mobile phones

- Another problem:
  - Quad-core ARM at 2 GHz: approx. 8 GOPS
  - Orders of magnitude too low performance
  - … And it would be nice if your phone can do something else than just 4G.

Example: mobile phones

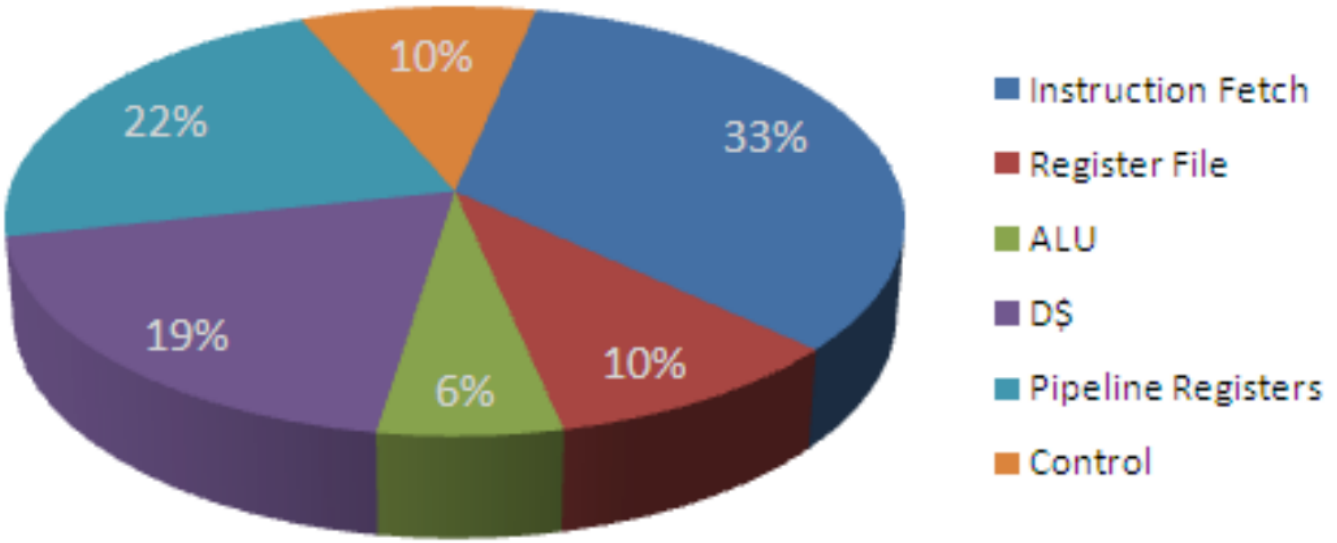
- Quick summary:
  - Compute efficiency is (at least) one order of magnitude off from 1 pJ/OP.
  - Performance is off by several orders of magnitude.

Inefficiencies in processors



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

Inefficiencies in processors



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

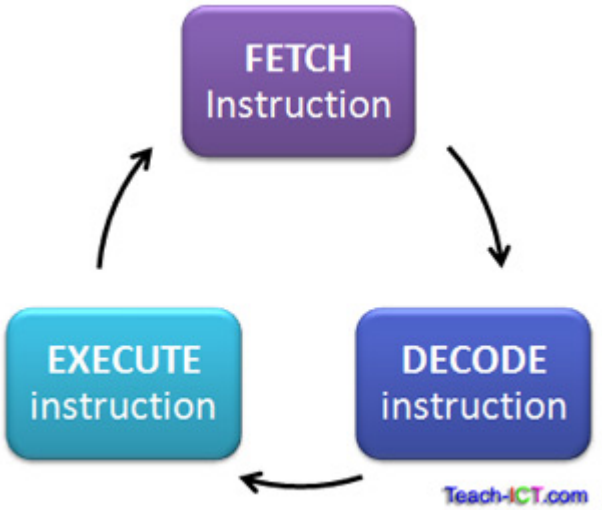
[Understanding sources of inefficiency in General-Purpose Chips, Hameed et al.]

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.

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.

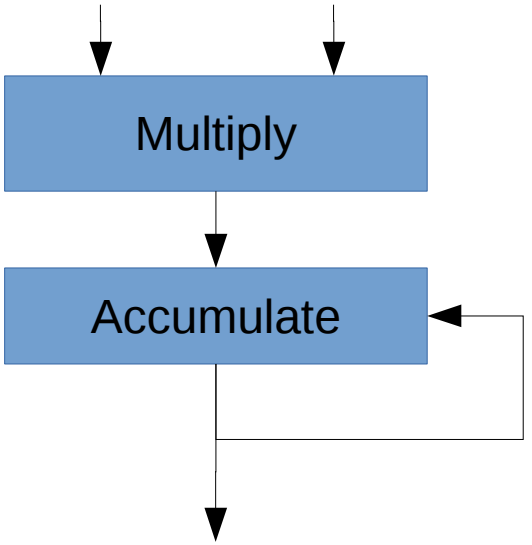


Data transport

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

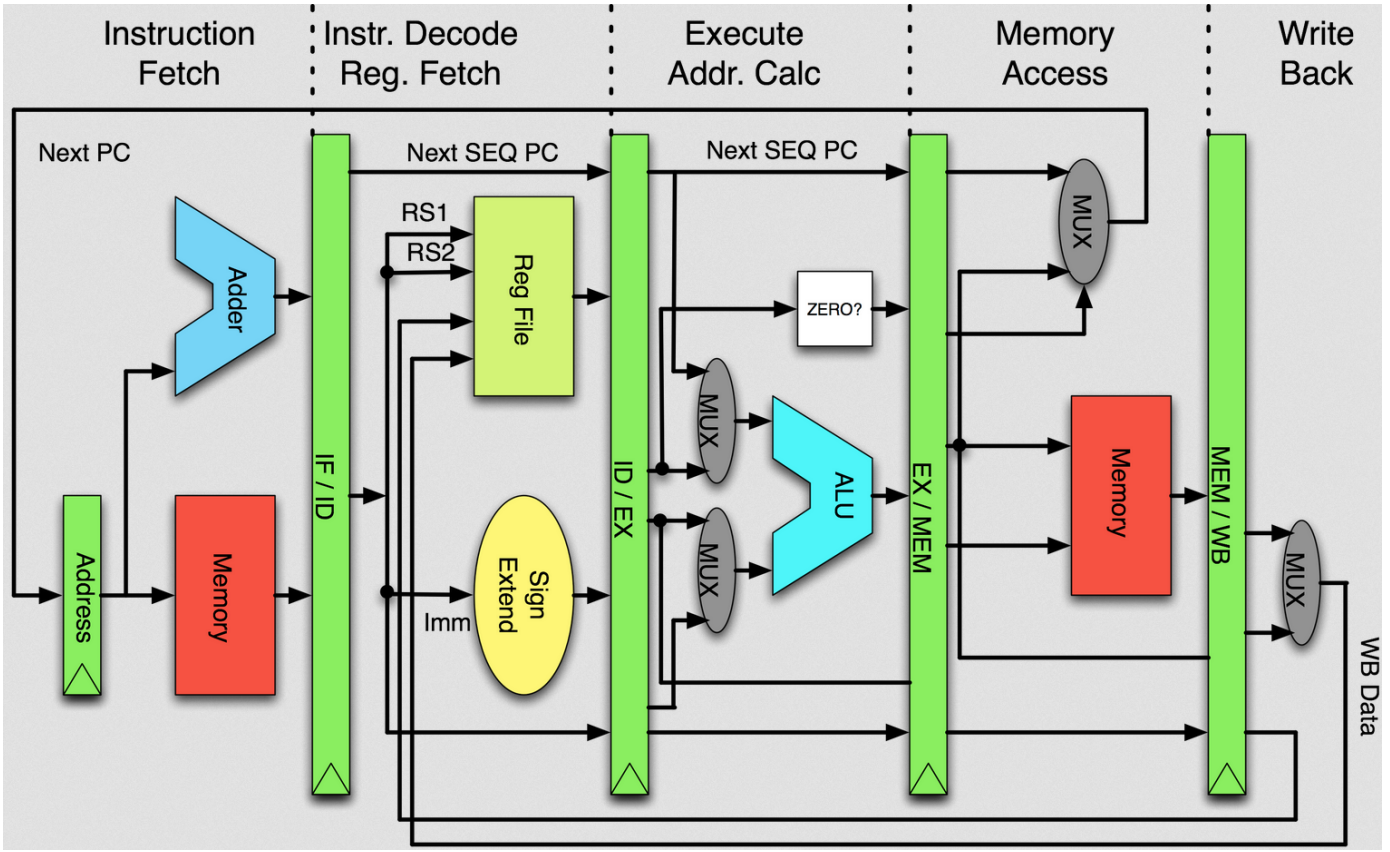
Data transport

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



Hardware reconfiguration

- Mostly multiplexers



General purpose processors

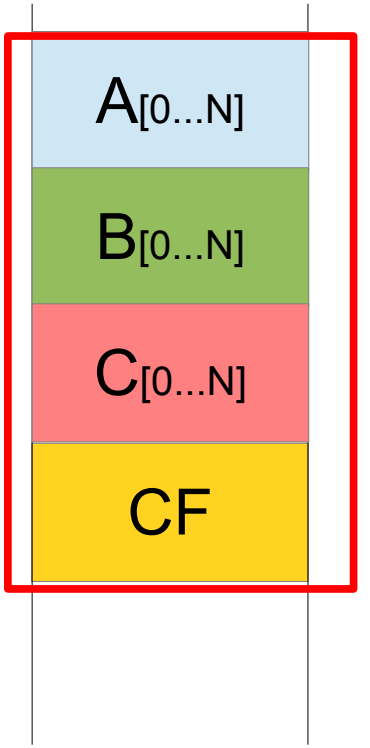
- Lessons learned:
  - Reduce instruction fetching and decoding
  - Reduce cycle-based hardware reconfiguration.
  - Reduce data transport to and from memories and RF.
  - Still needs to be programmable

Hardware acceleration

Static control

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

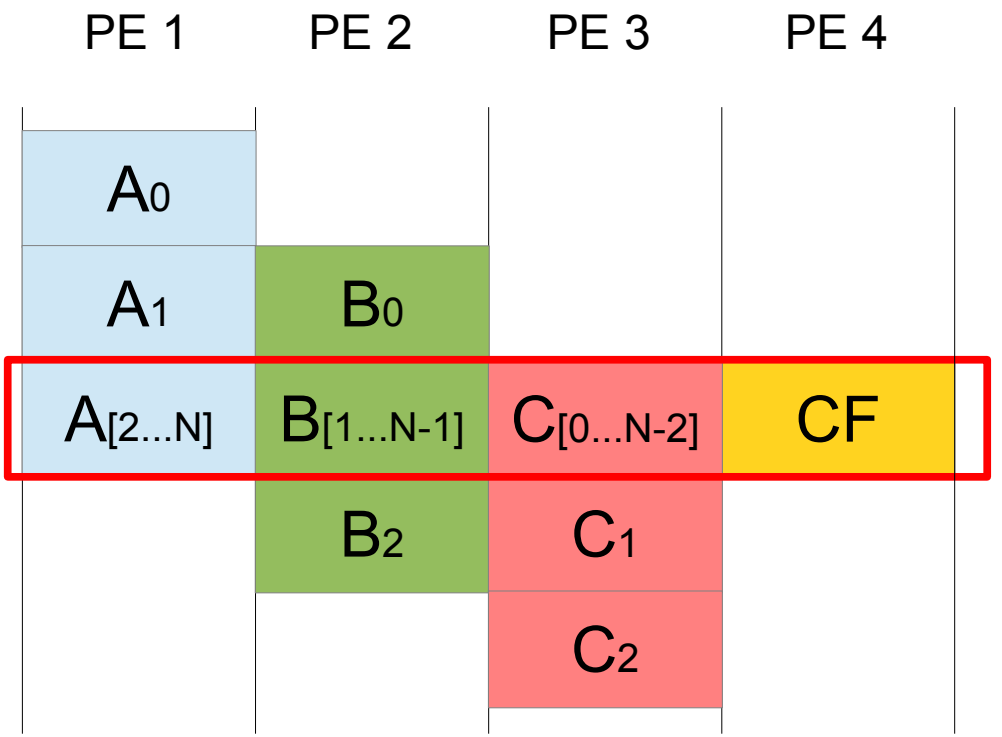
```
for i = 0 to N
  A
  B
  C
```



Static control

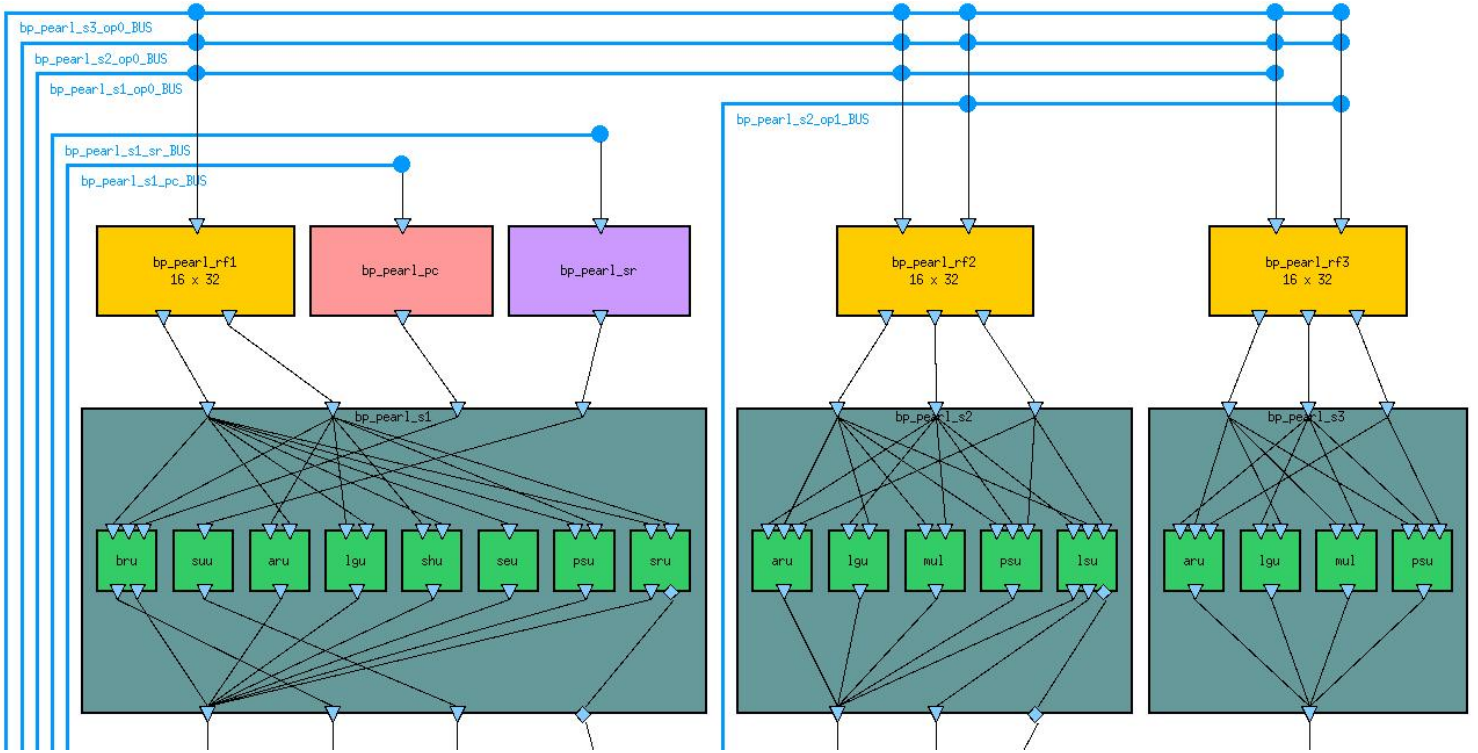
- Loops are the best candidate for static control
  - Do the same thing many times
- Software pipelining

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



Very Long Instruction Word processors (VLIW)

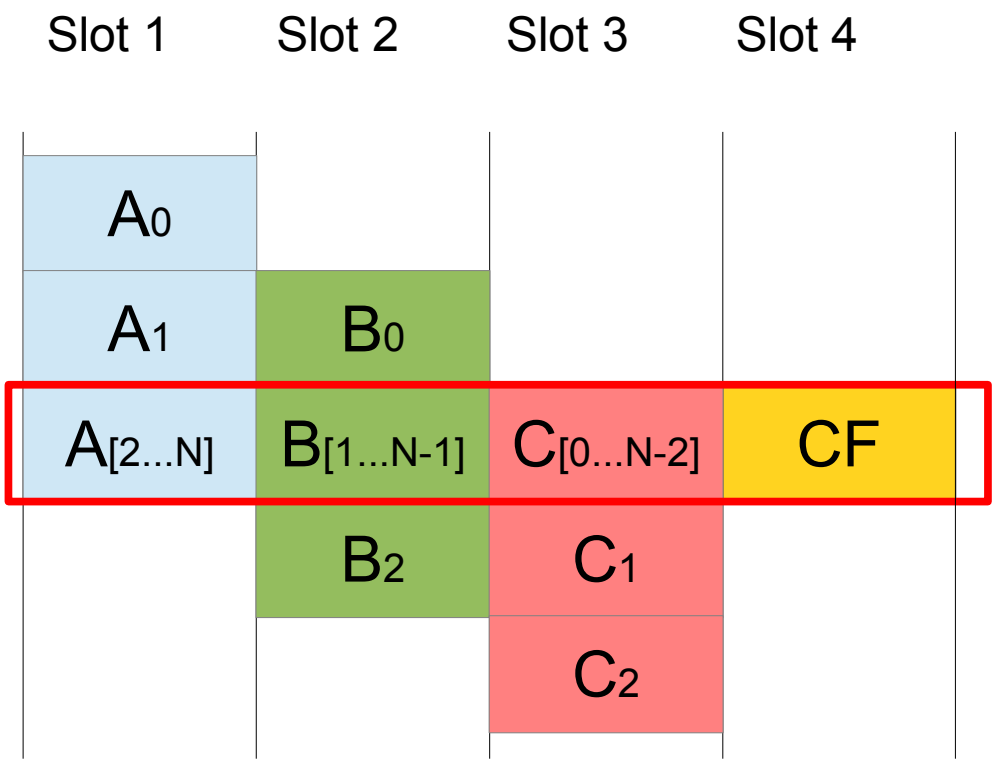
- Processor with multiple issue-slots
- One long instruction controlling them all



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
```

ASICs

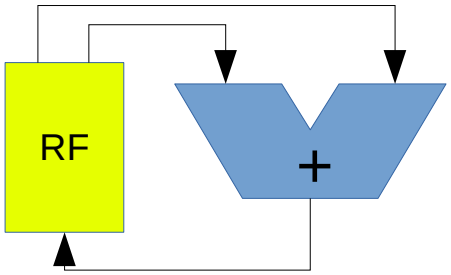
- The application is completely software pipelined and implemented ('hard coded') in hardware.
  - No more instruction fetching and decoding
  - But cannot be changed anymore after production
- But ASICs can do something about RF and memory accesses...

Spatial layout

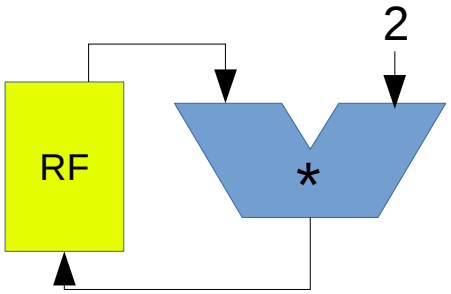
Compute: (A+B) * 2

General purpose | *VLIW* | *Spatial layout*

Cycle N

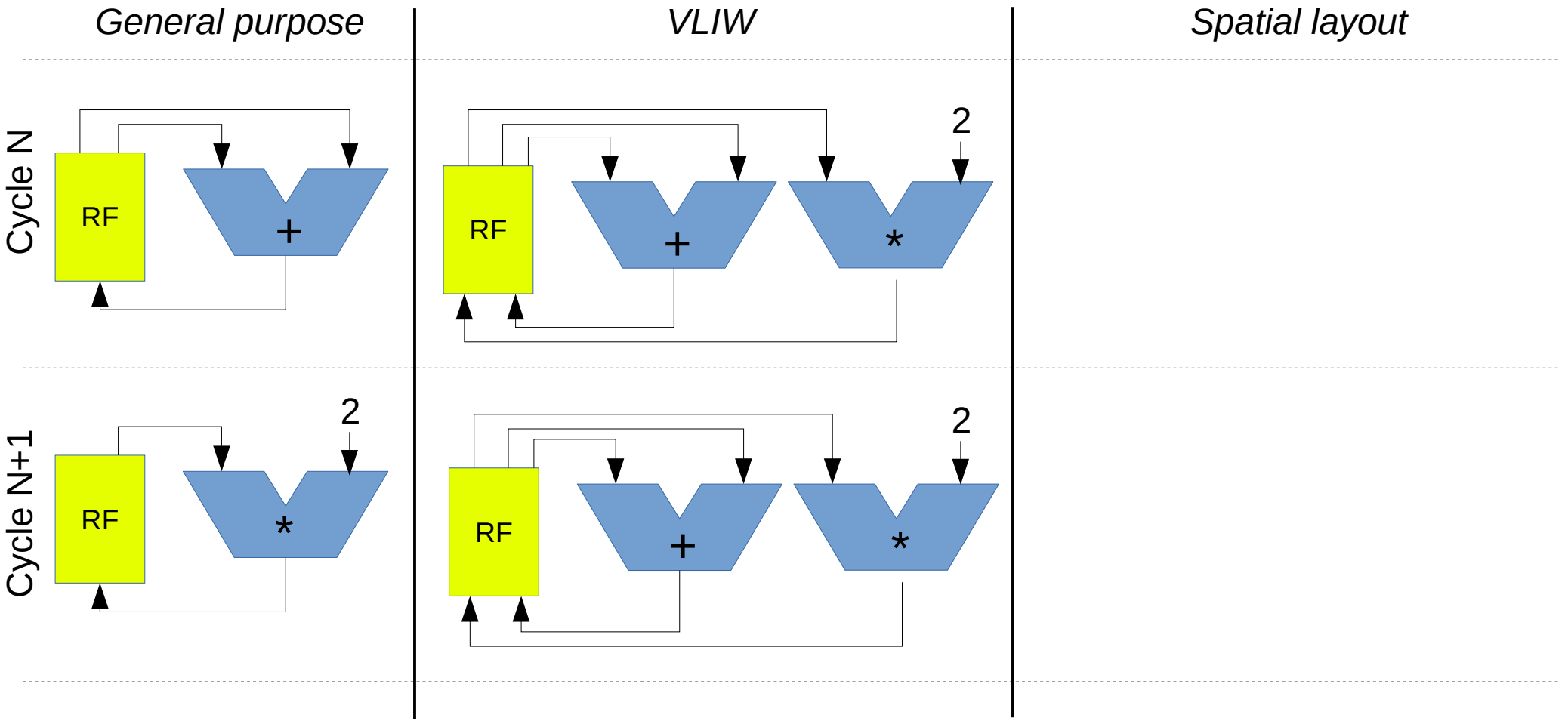


Cycle N+1



Spatial layout

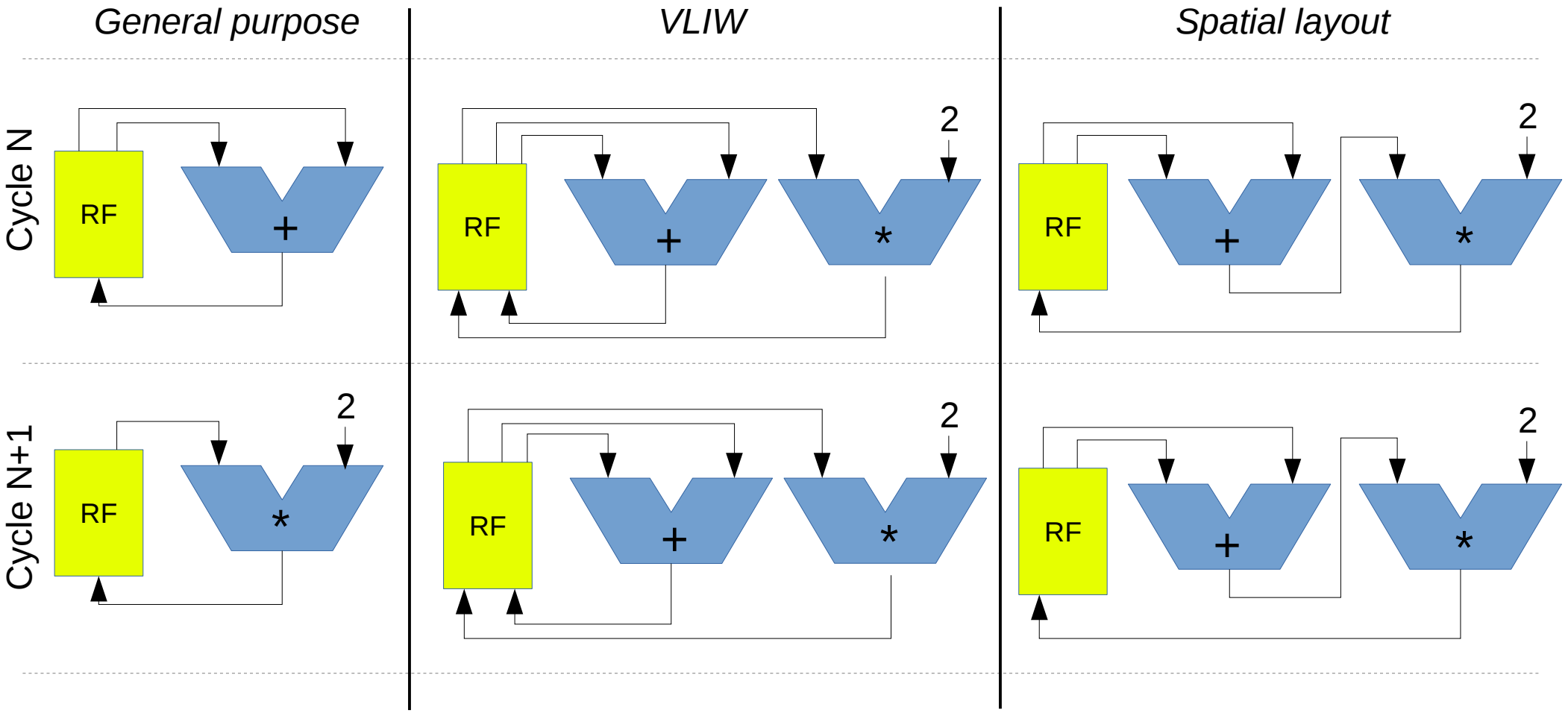
Compute: (A+B) * 2



2 times the throughput!

Spatial layout

Compute: (A+B) * 2



2 times the throughput!

2 times the throughput!

Spatial layout

- Essentially a software pipelined loop with direct connections between functional units.

- Specialized ASICs, e.g. dedicated H.264 decoders.
  - You probably have one in your smart-phone
- Example: H.264 decoding 1080p @ 30fps
  - ASIC: 186 mW (180nm) → 2.78mW (22nm)
  - i5 4300M: 2.84 W (22nm)

Spatial layout

- Essentially a software pipelined loop with direct connections between functional units.

- Specialized ASICs, e.g. dedicated H.264 decoders.
  - You probably have one in your smart-phone
- Example: H.264 decoding 1080p @ 30fps
  - ASIC: 186 mW (180nm) → 2.78mW (22nm)
  - i5 4300M: 2.84 W (22nm)

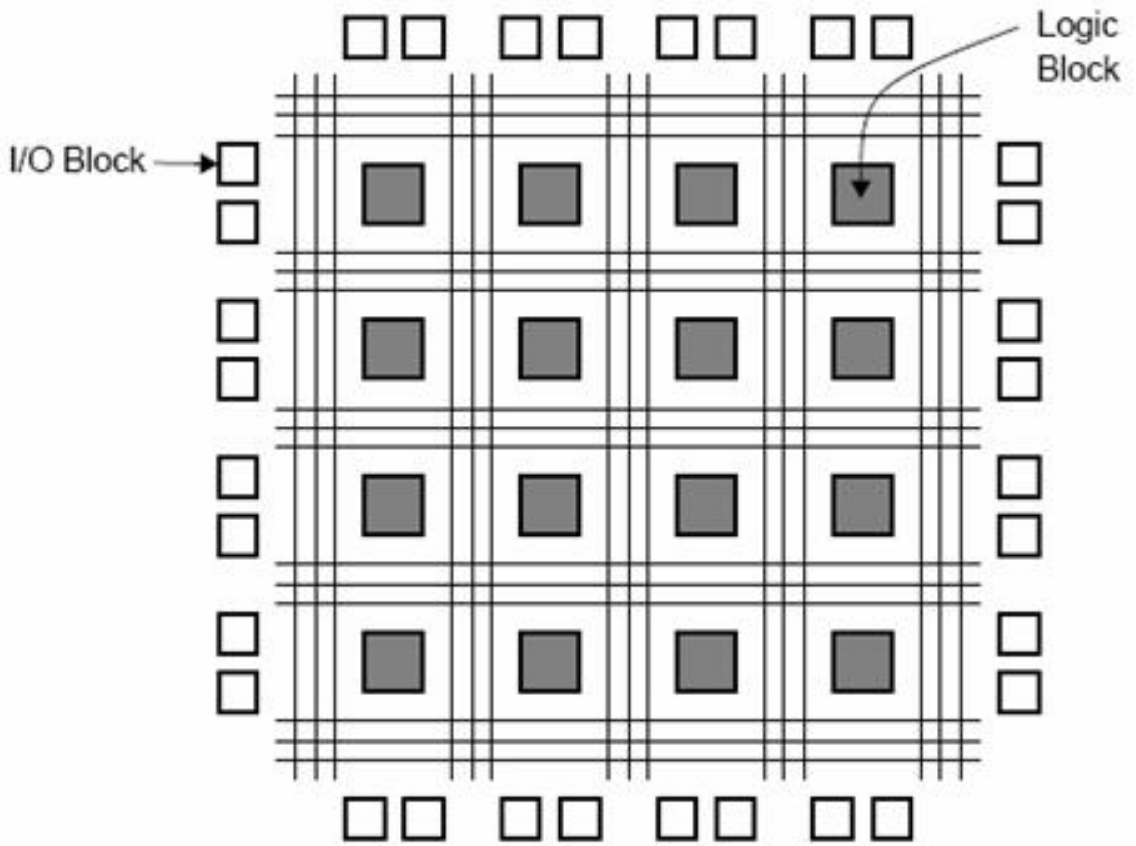
ASICs

- 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

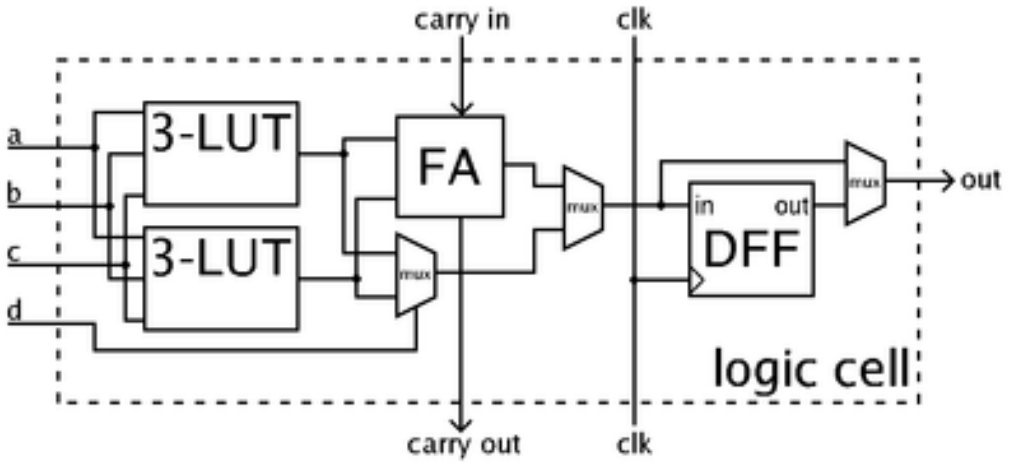
- Chip full of configurable logic blocks/cells



[fpgacentral.com]

Field Programmable Gate Arrays

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



[fpgacentral.com]

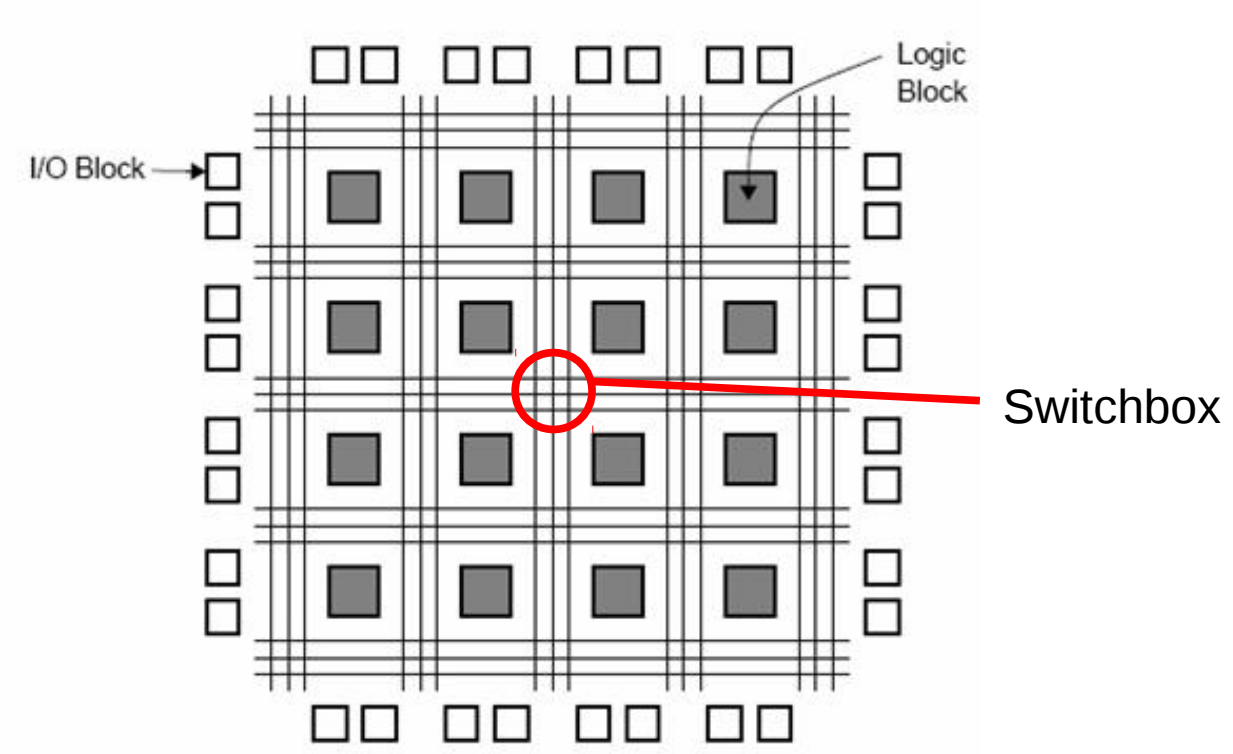
Field Programmable Gate Arrays

- With the exception of specialized blocks most FPGAs contain gate-level 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 (usually) fixed after that



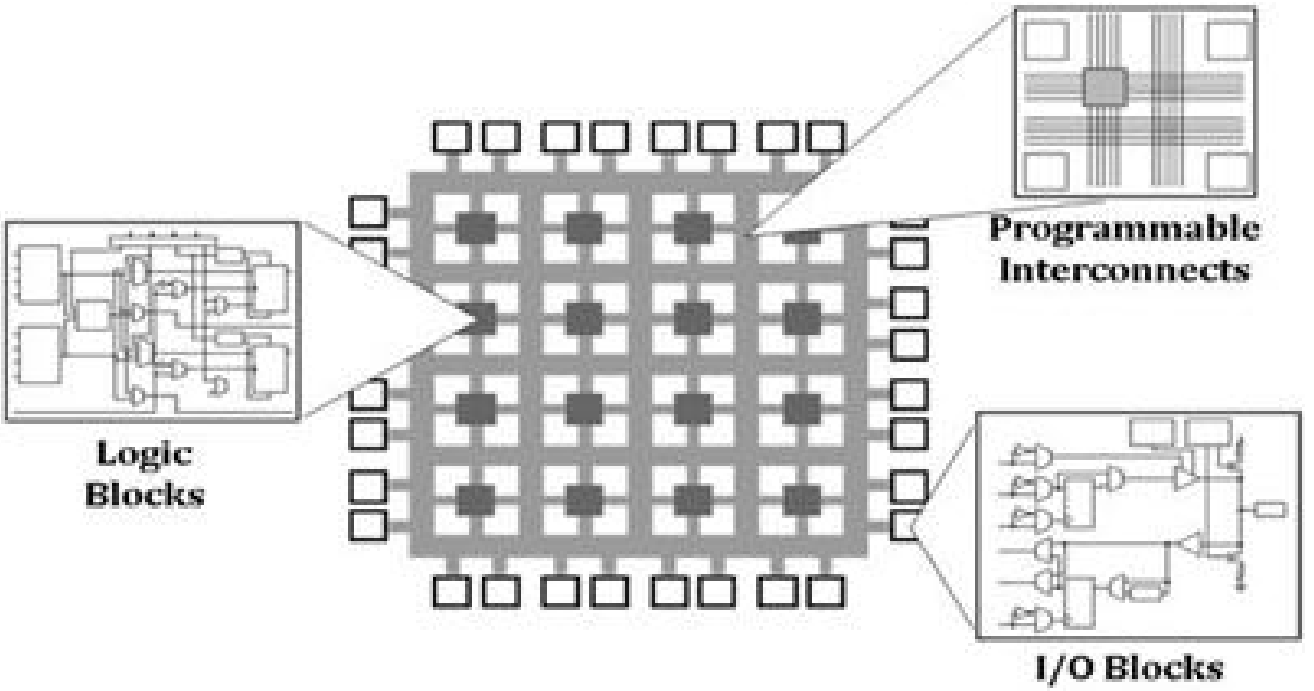
Field Programmable Gate Arrays

- 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



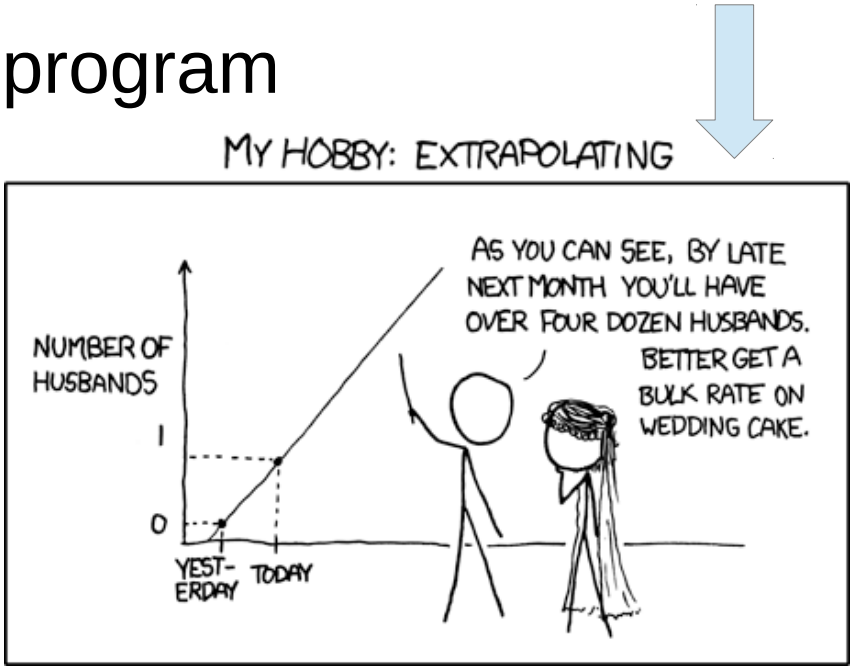
FPGA Configures at gate level, which incurs large overheads:

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

Image Source: www.ni.com

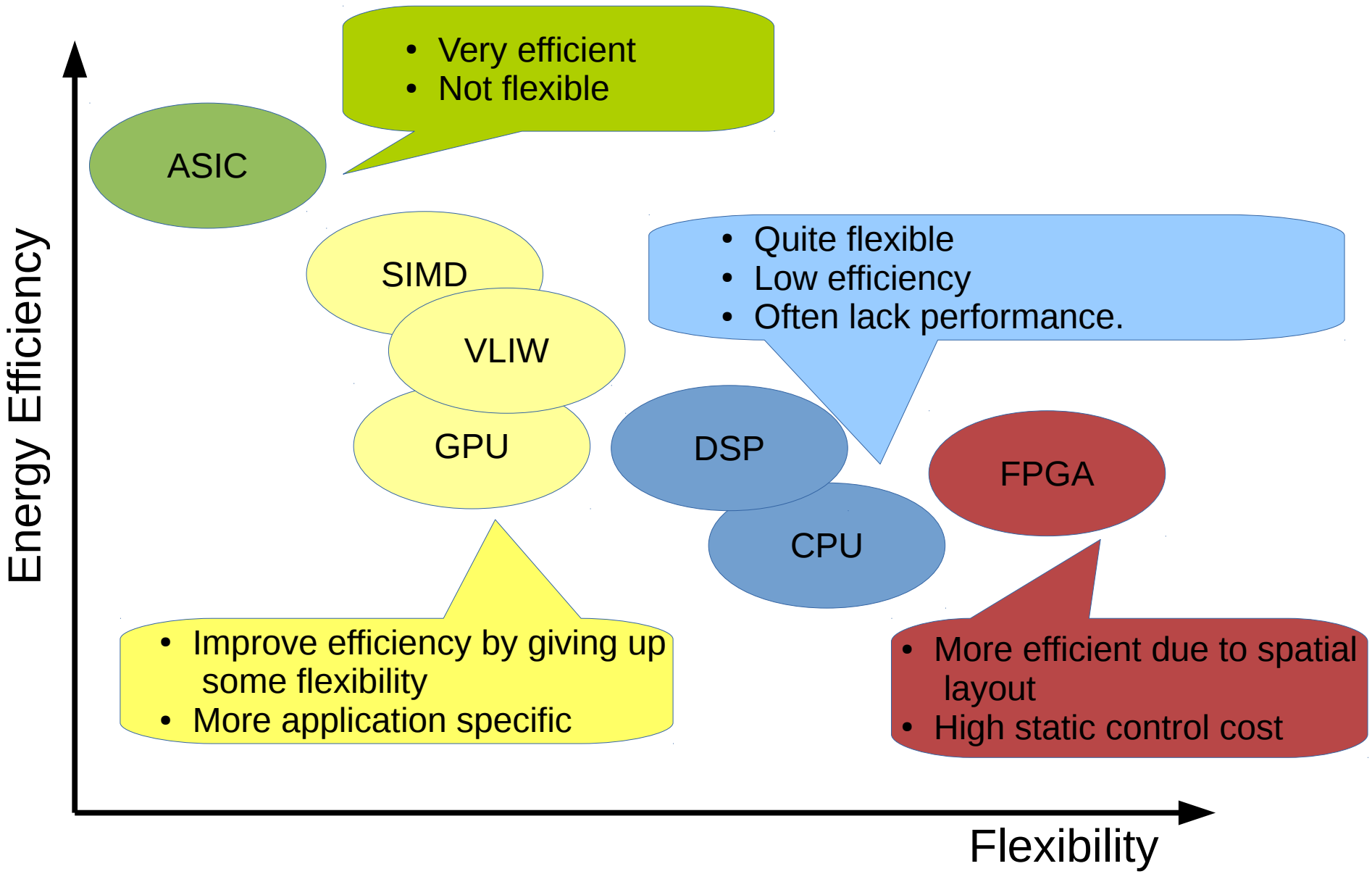
Field Programmable Gate Arrays

- 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



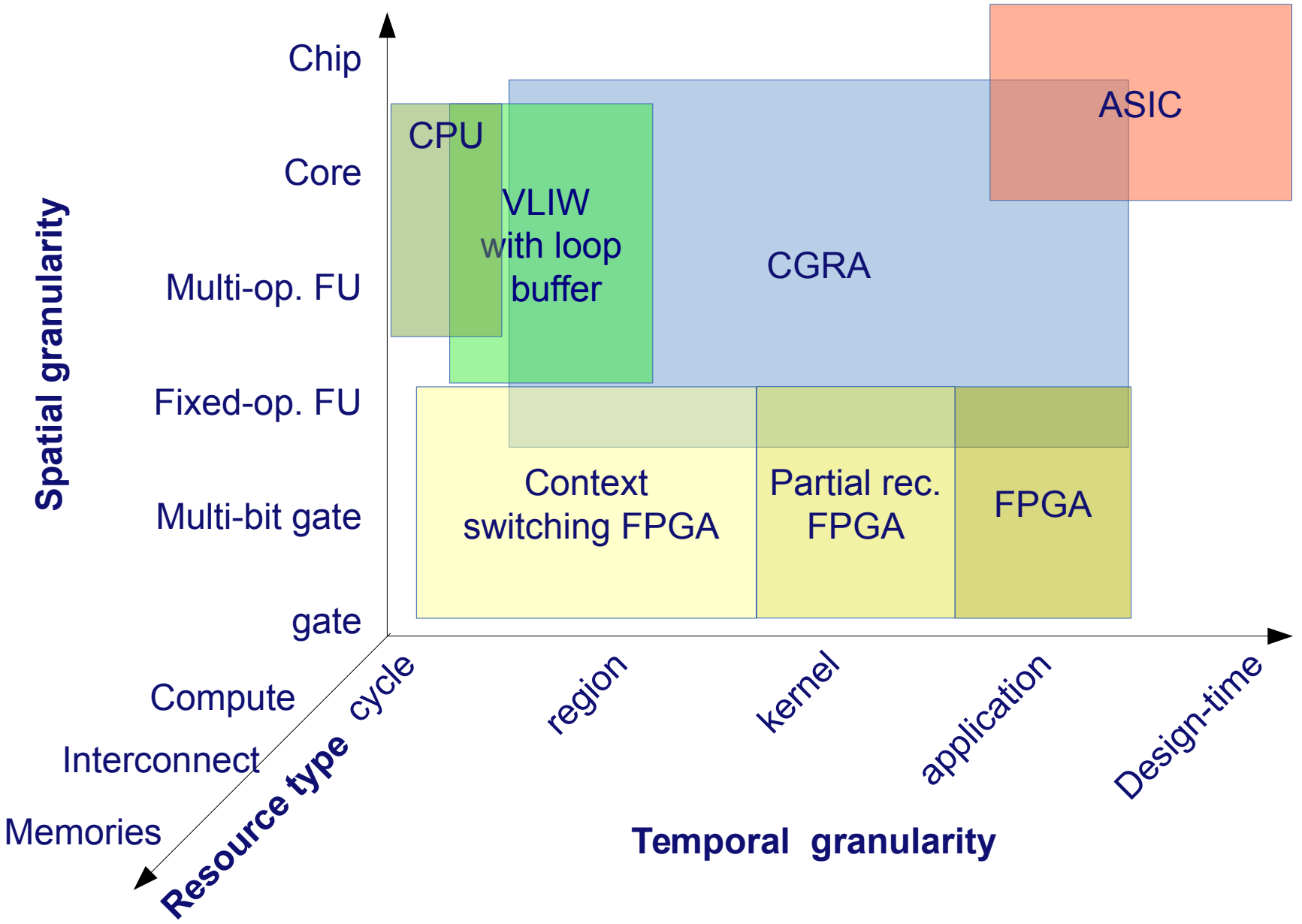
[xkcd.com]

What have we learned so far?



Coarse Grained Reconfigurable Architecture

Coarse Grained Reconfigurable Architecture



Spatial Layout on the Cheap

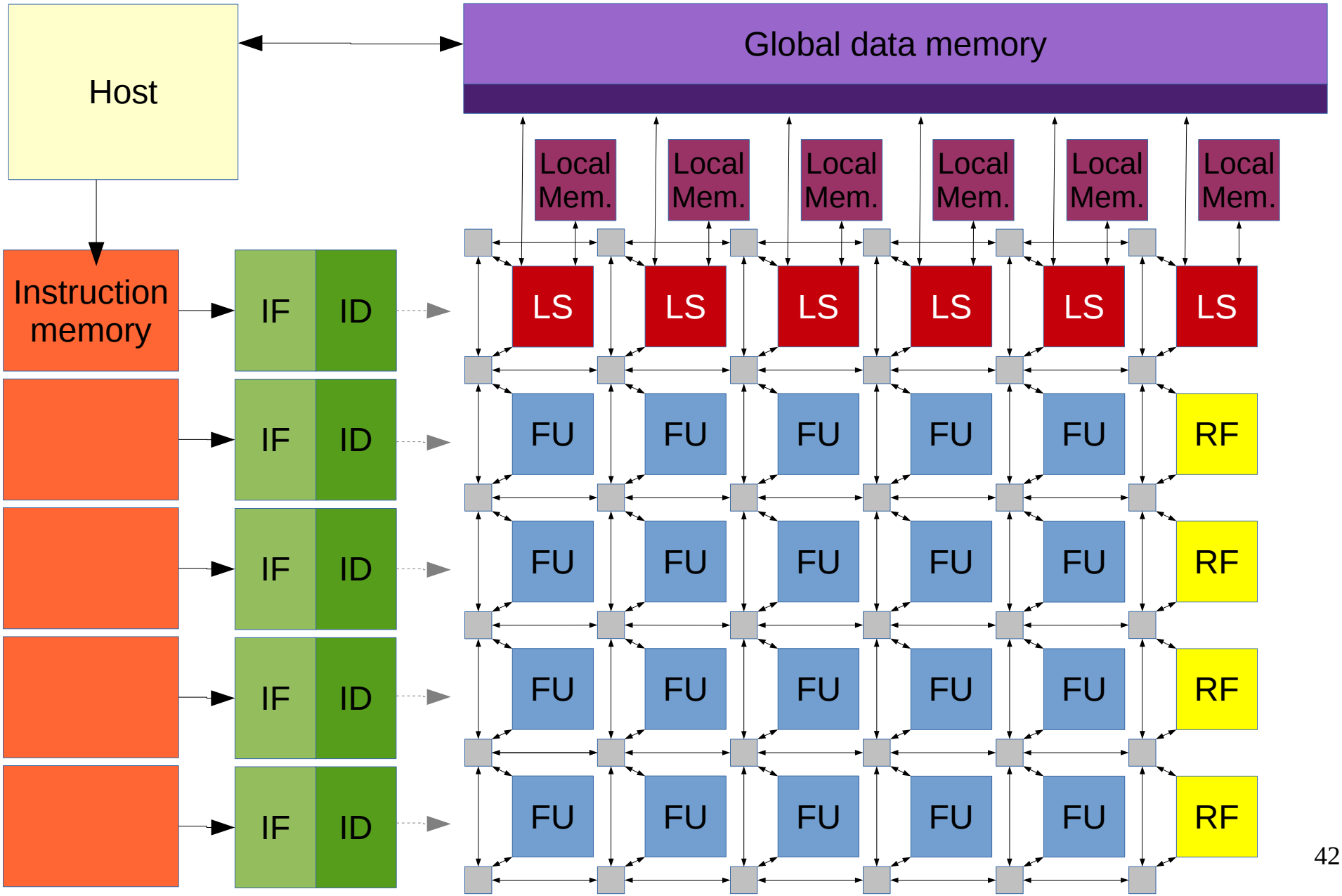
Gate-level granularity is often not required digital signal processing

What our architecture does:

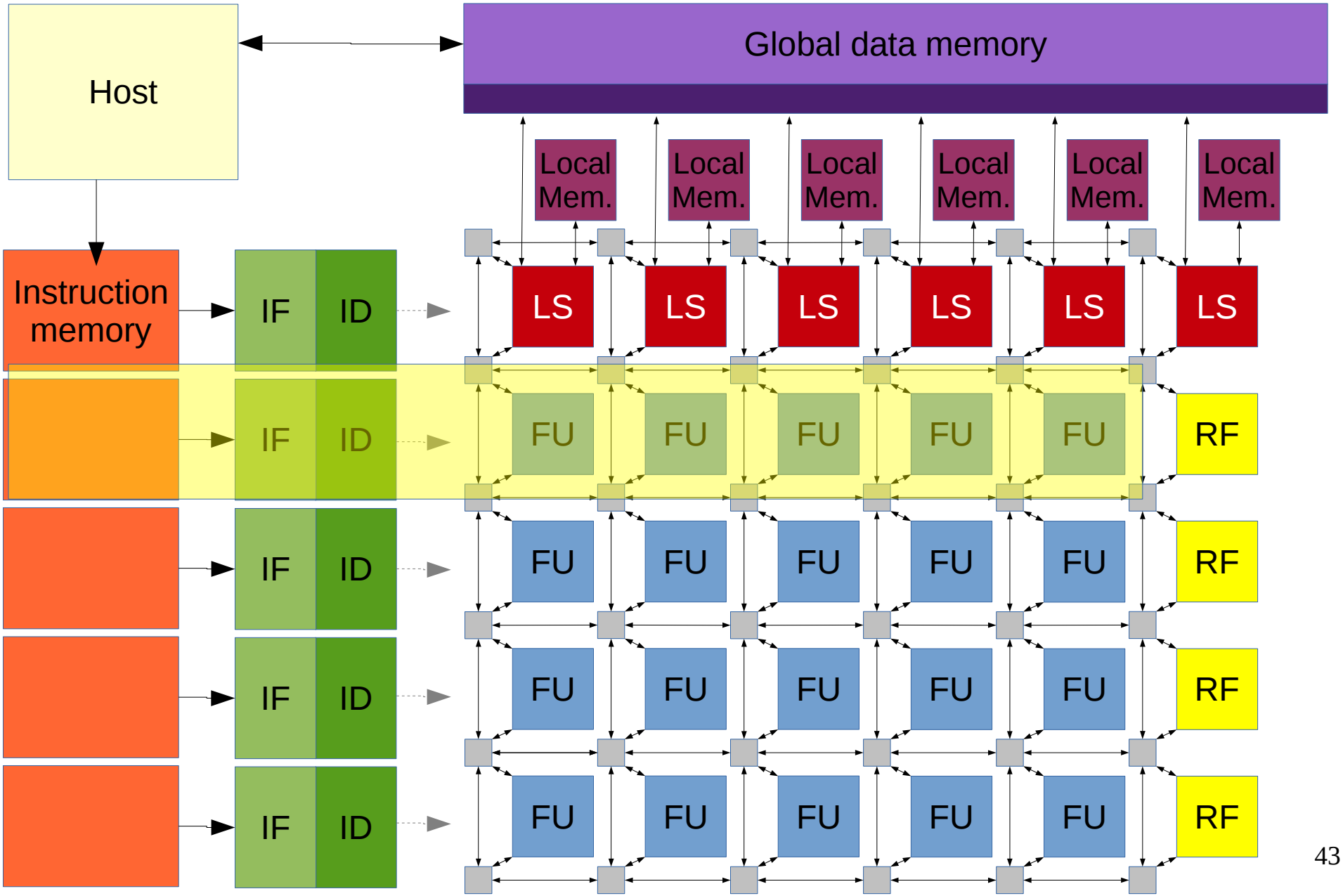
- Configure at functional unit level
- Statically route data-path (spatial layout) but allow instructions.

A specialized FPGA to build processors

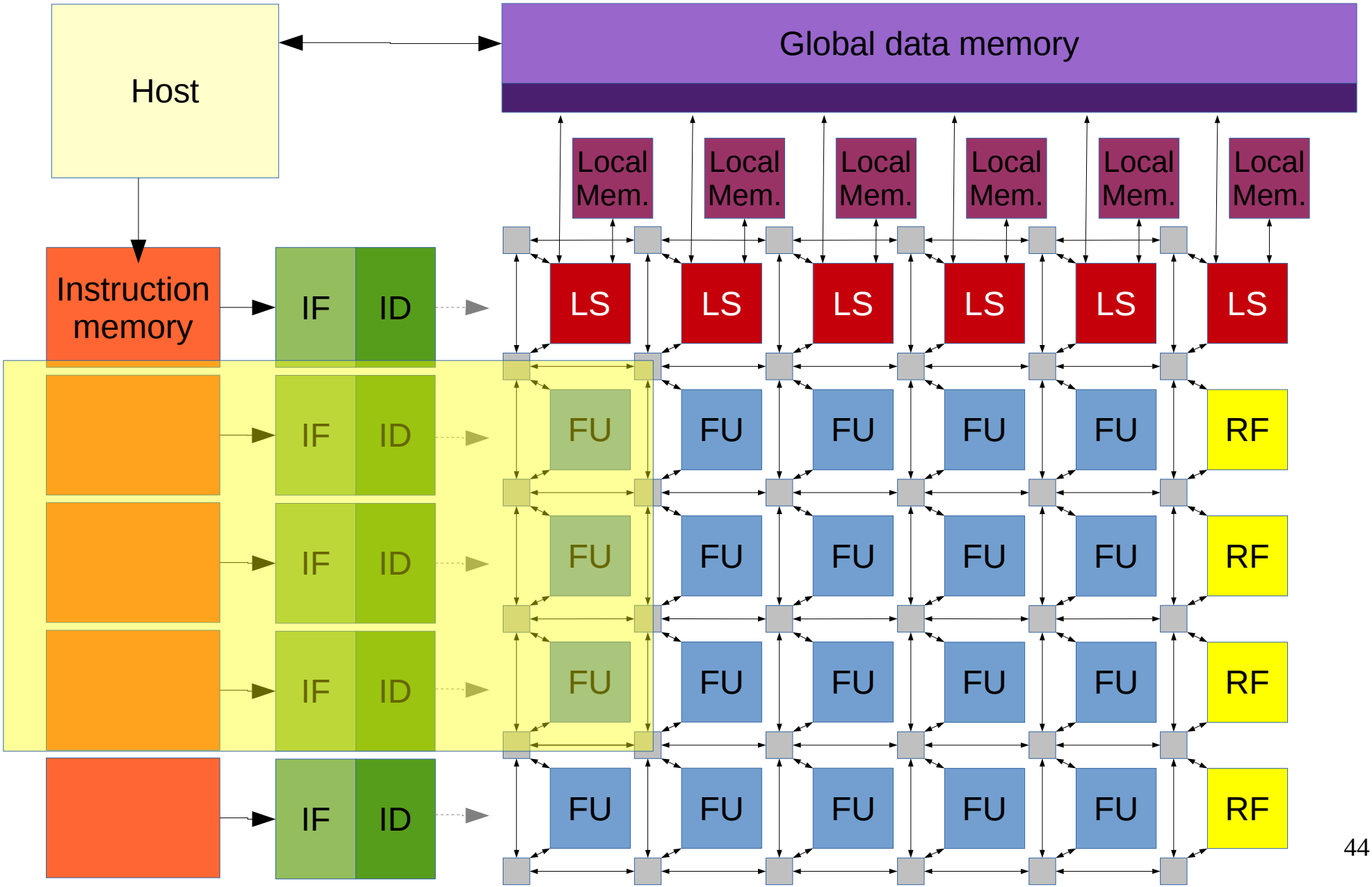
CGRA


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



Building Blocks

Instruction Fetch/Decode (IF/ID)

Load Store Units (LSU)

Register Files (RF)

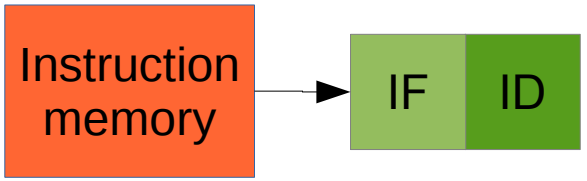
Memories (MEM)

Functional Units (FU)

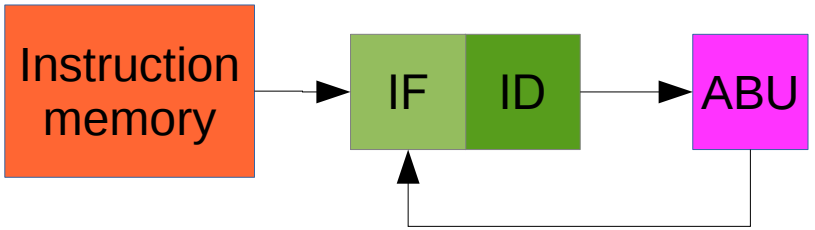
- Add, subtract, bitlevel operations
- Multiplication
- Accumulator
- …

- Switchboxes

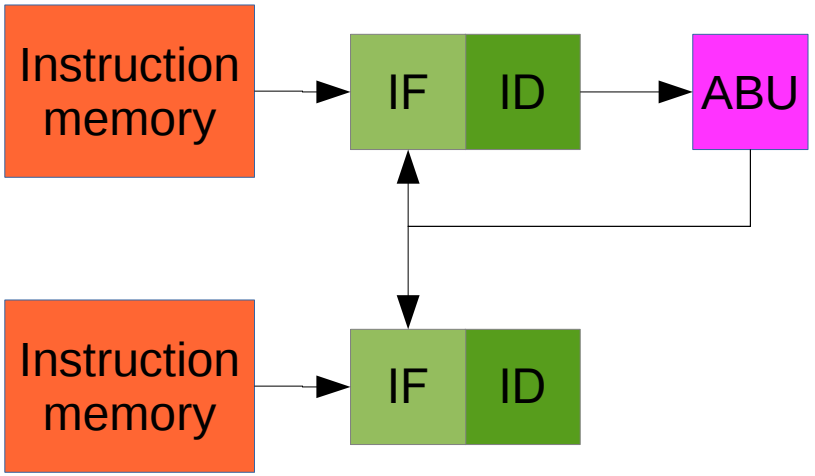
Building a processor



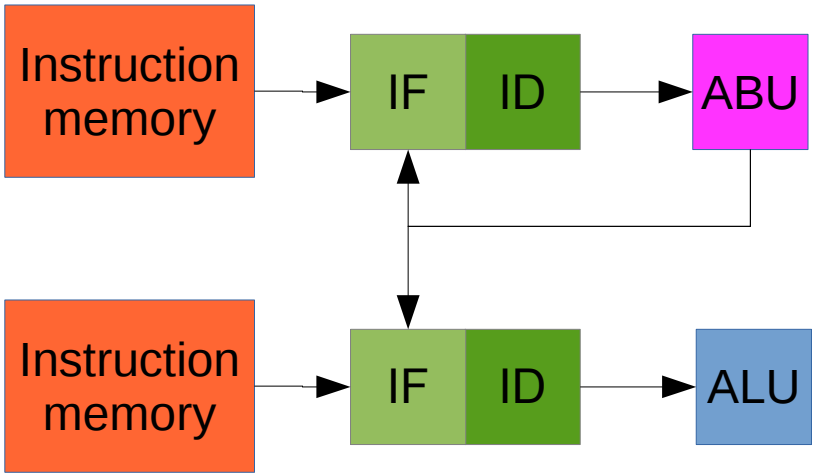
Building a processor



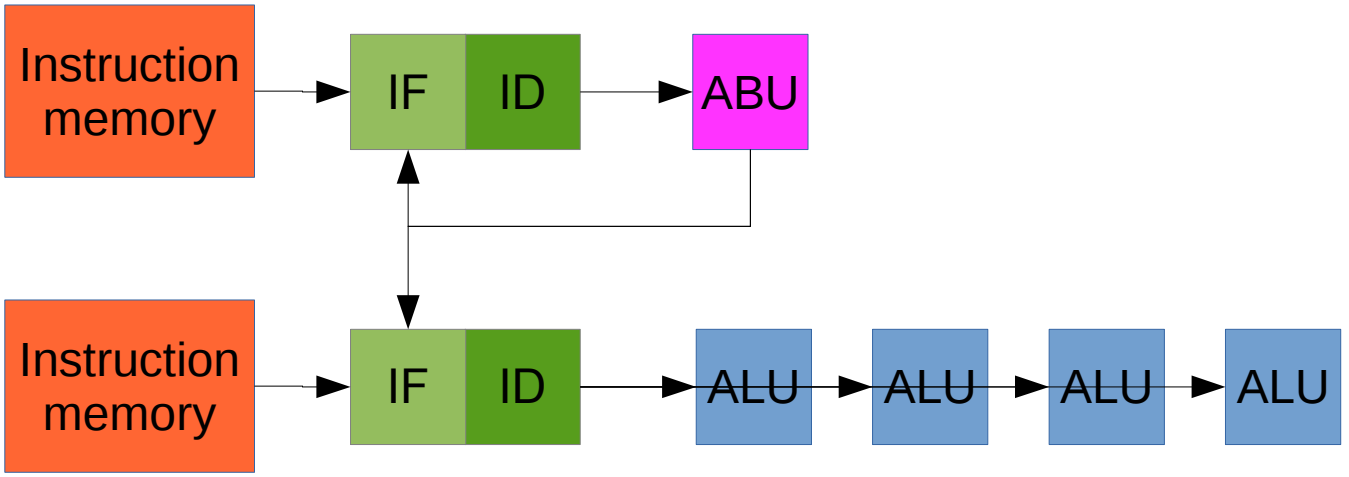
Building a processor



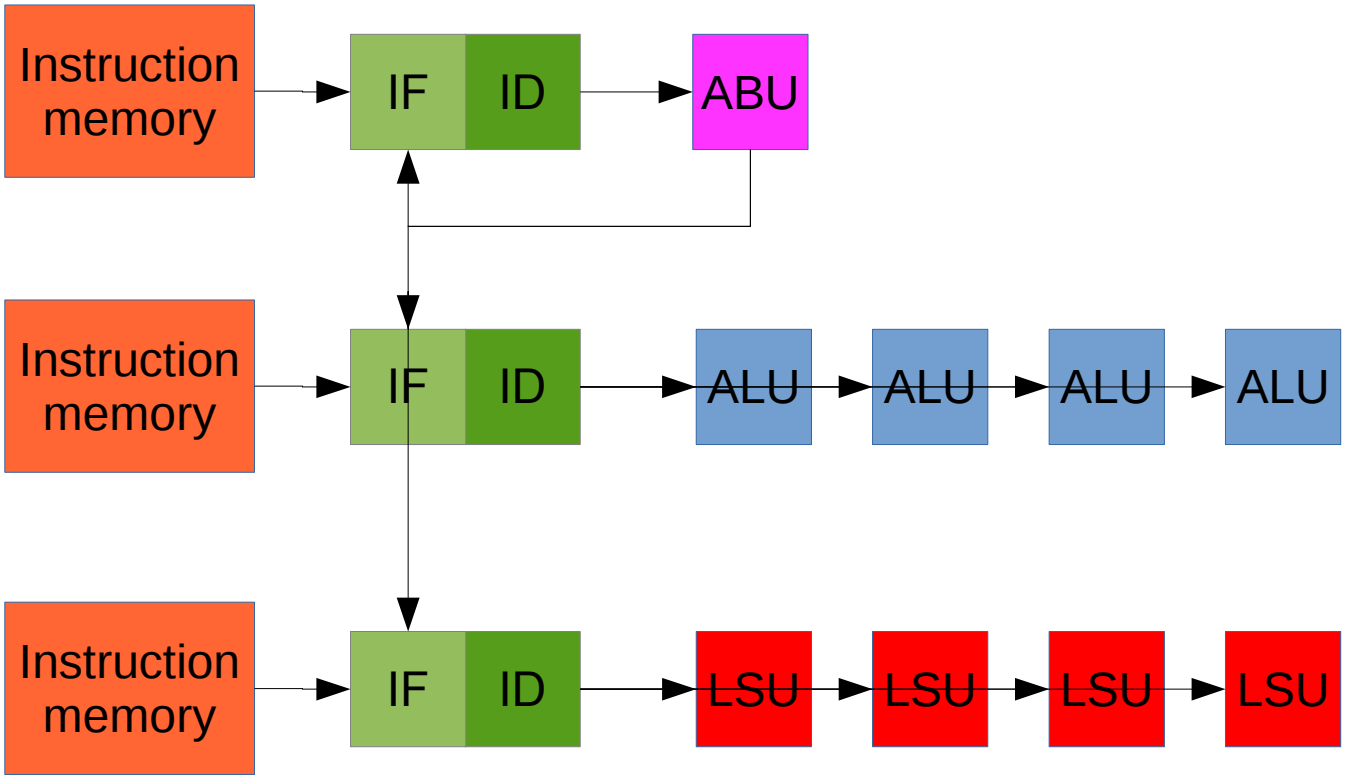
Building a processor



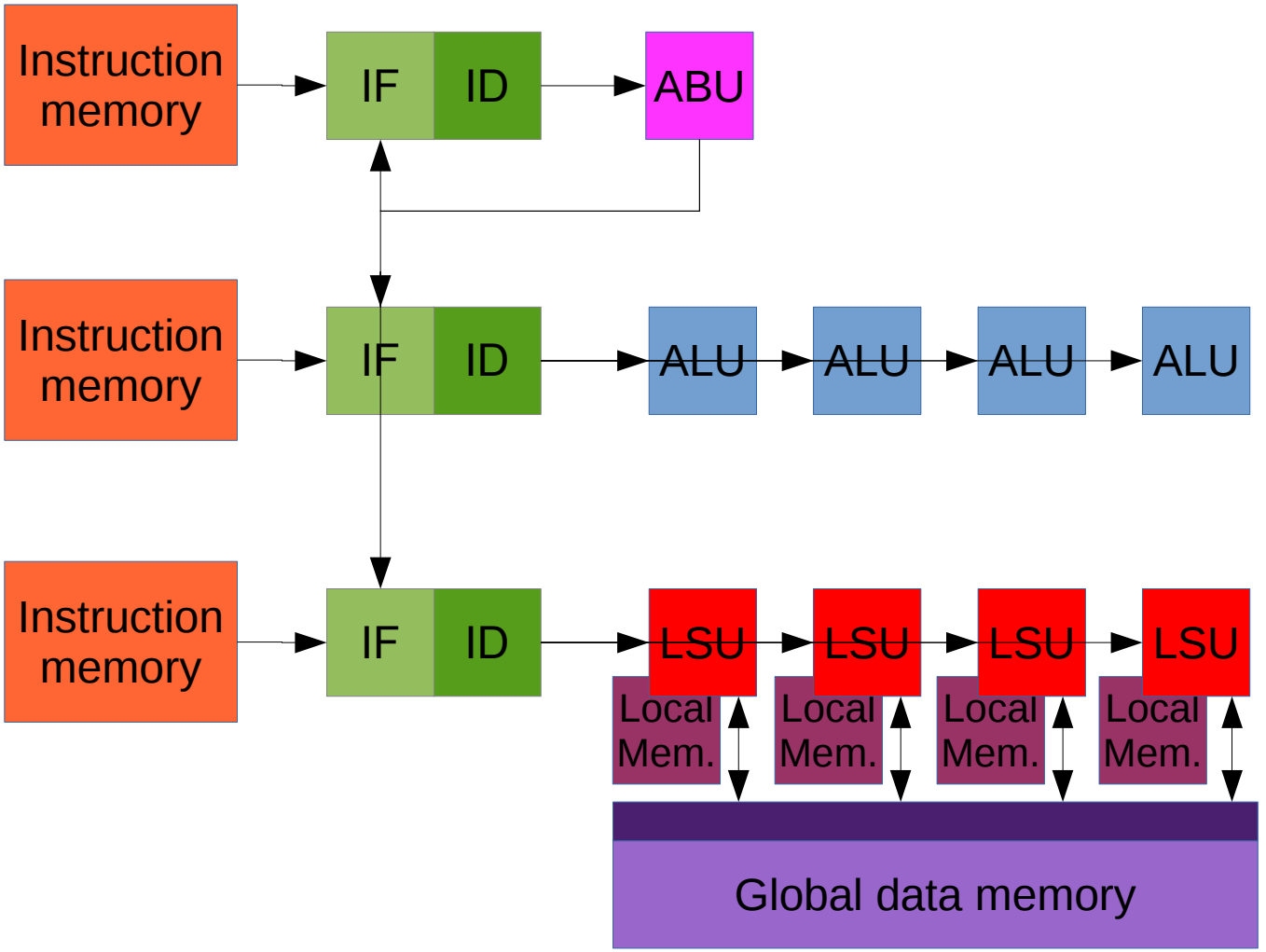
Building a processor



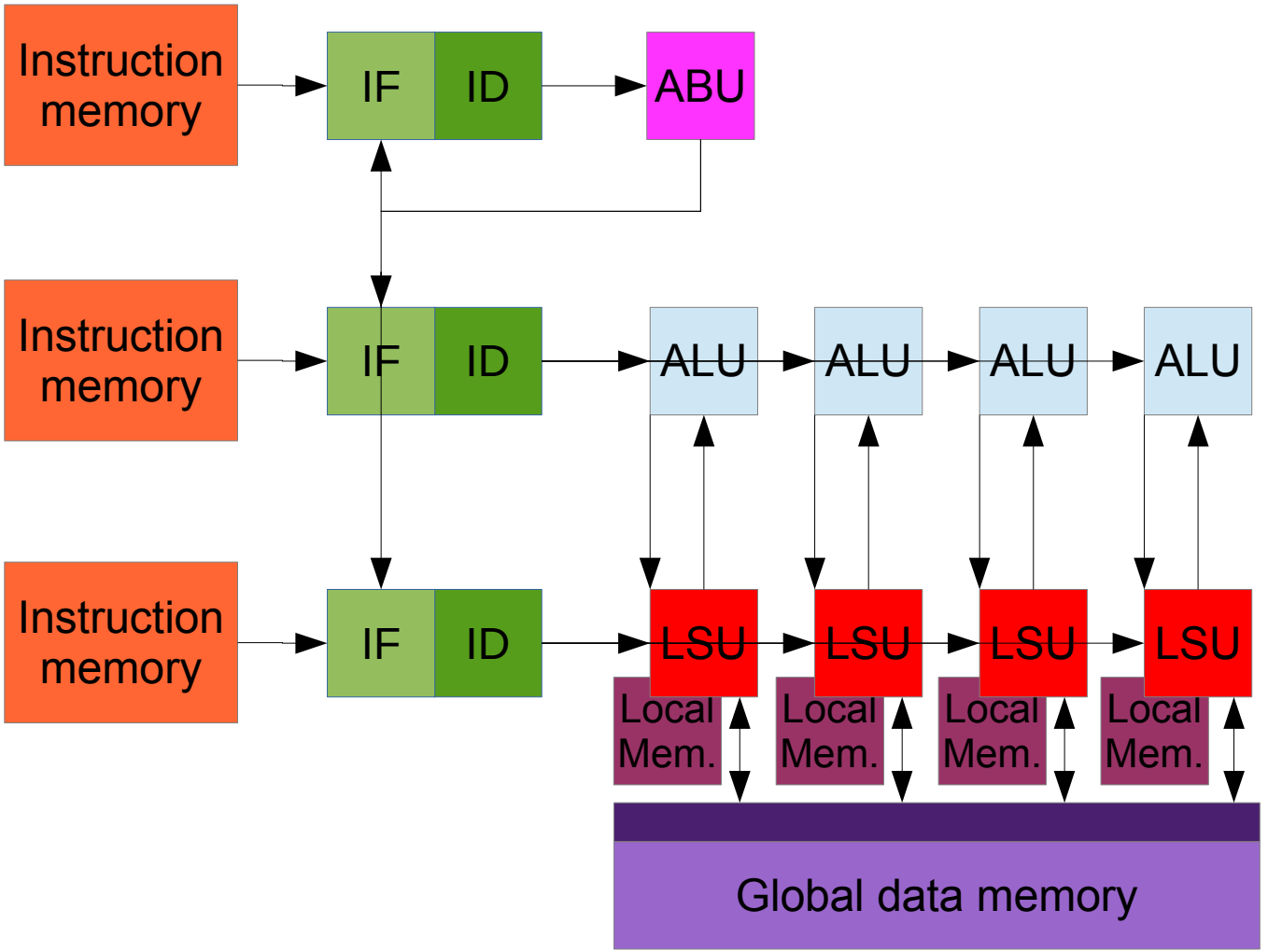
Building a processor



Building a processor



Building a processor



Building a processor

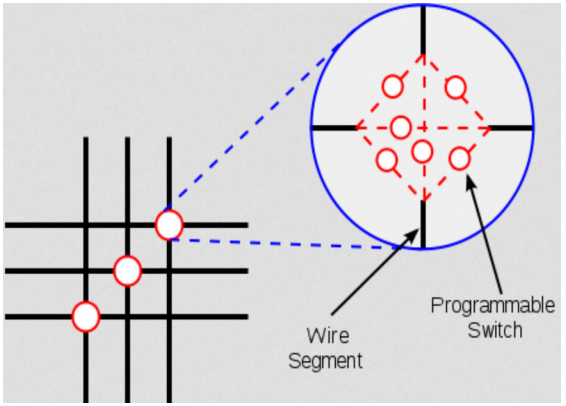
- We can have multiple ABUs
  - What does this mean?

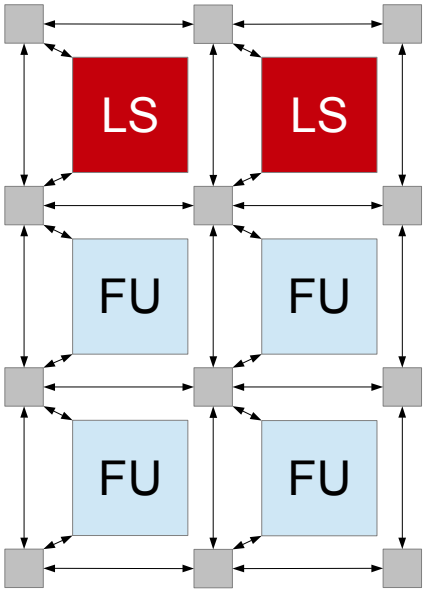
Building a processor

- 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

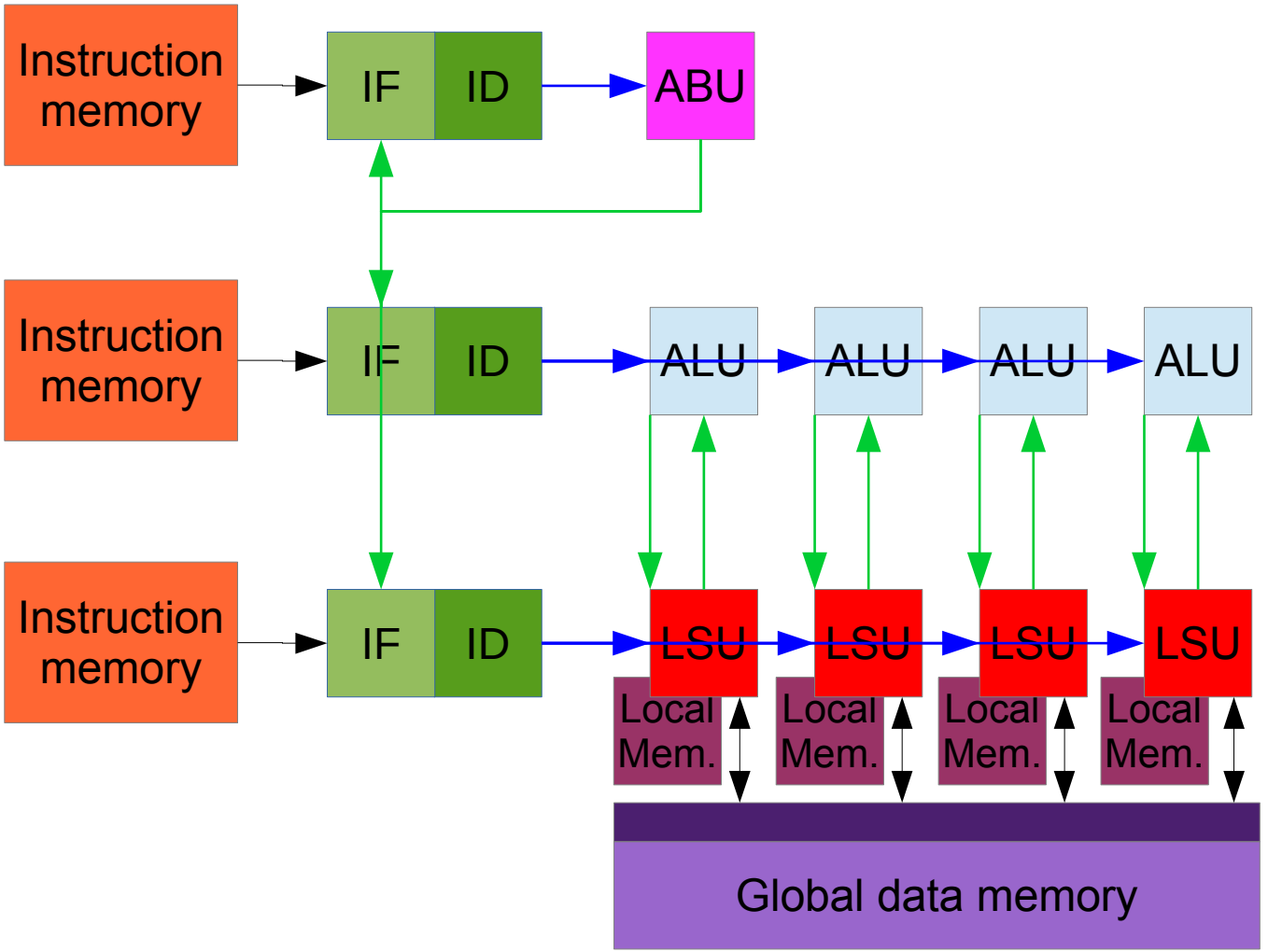
Dynamic CGRA

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





Dynamic CGRA

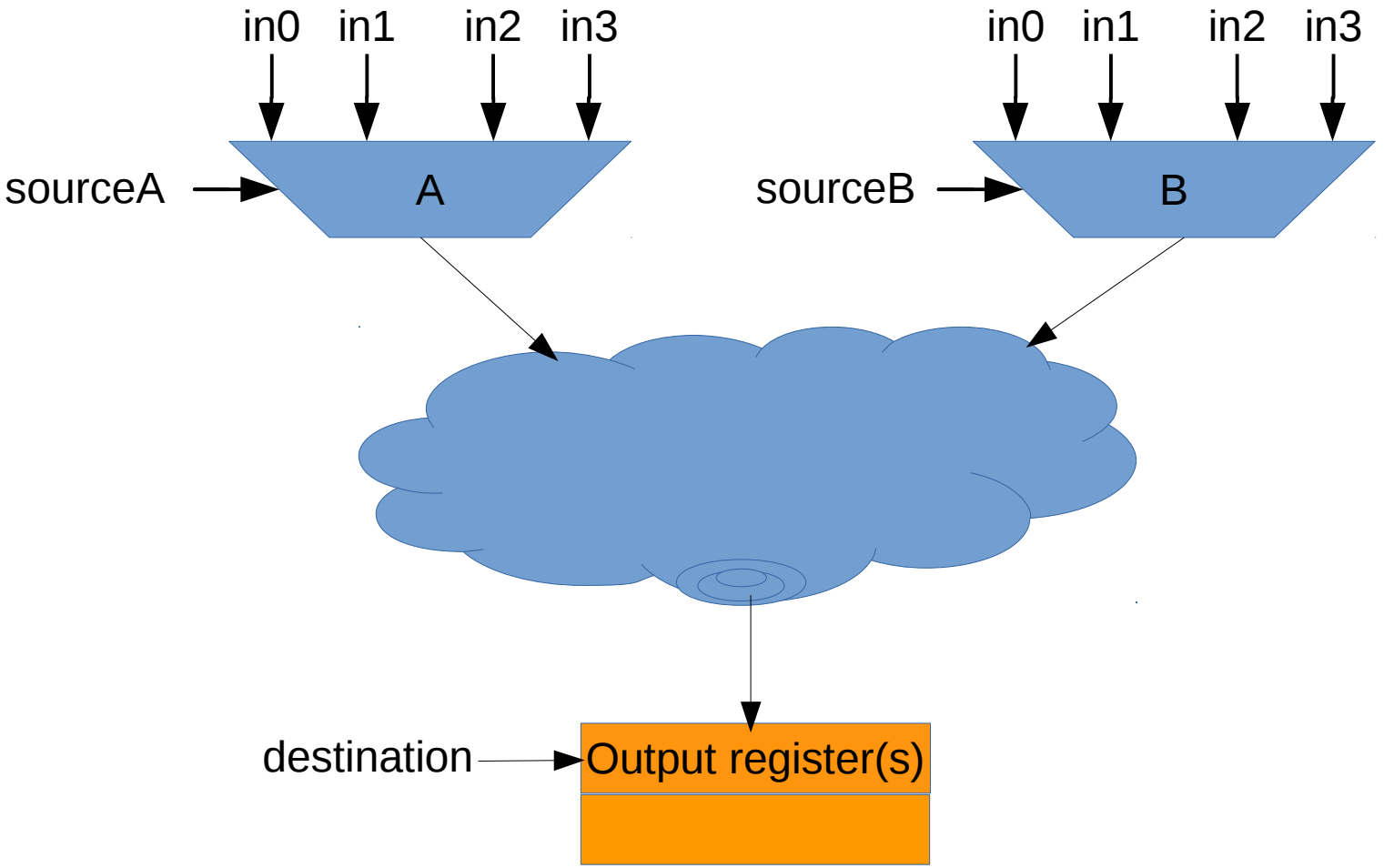


Functional units

- 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

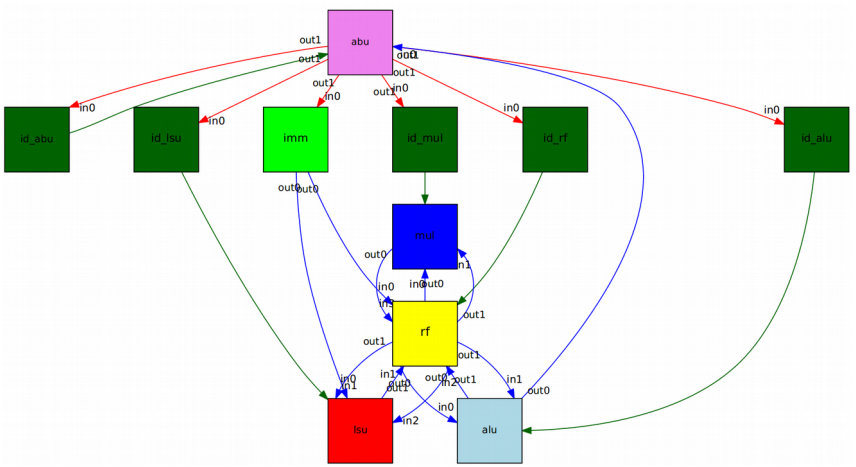
- Source inputs and destination outputs specified in instruction.
- Instruction usually in the form:

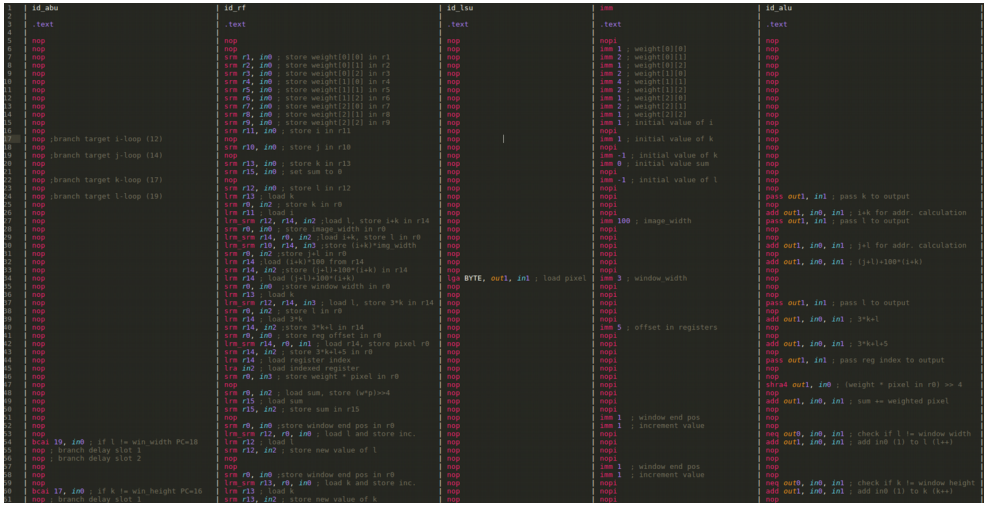
```
opcode dest, inA, inB
```

- Each output register is a source on the network.

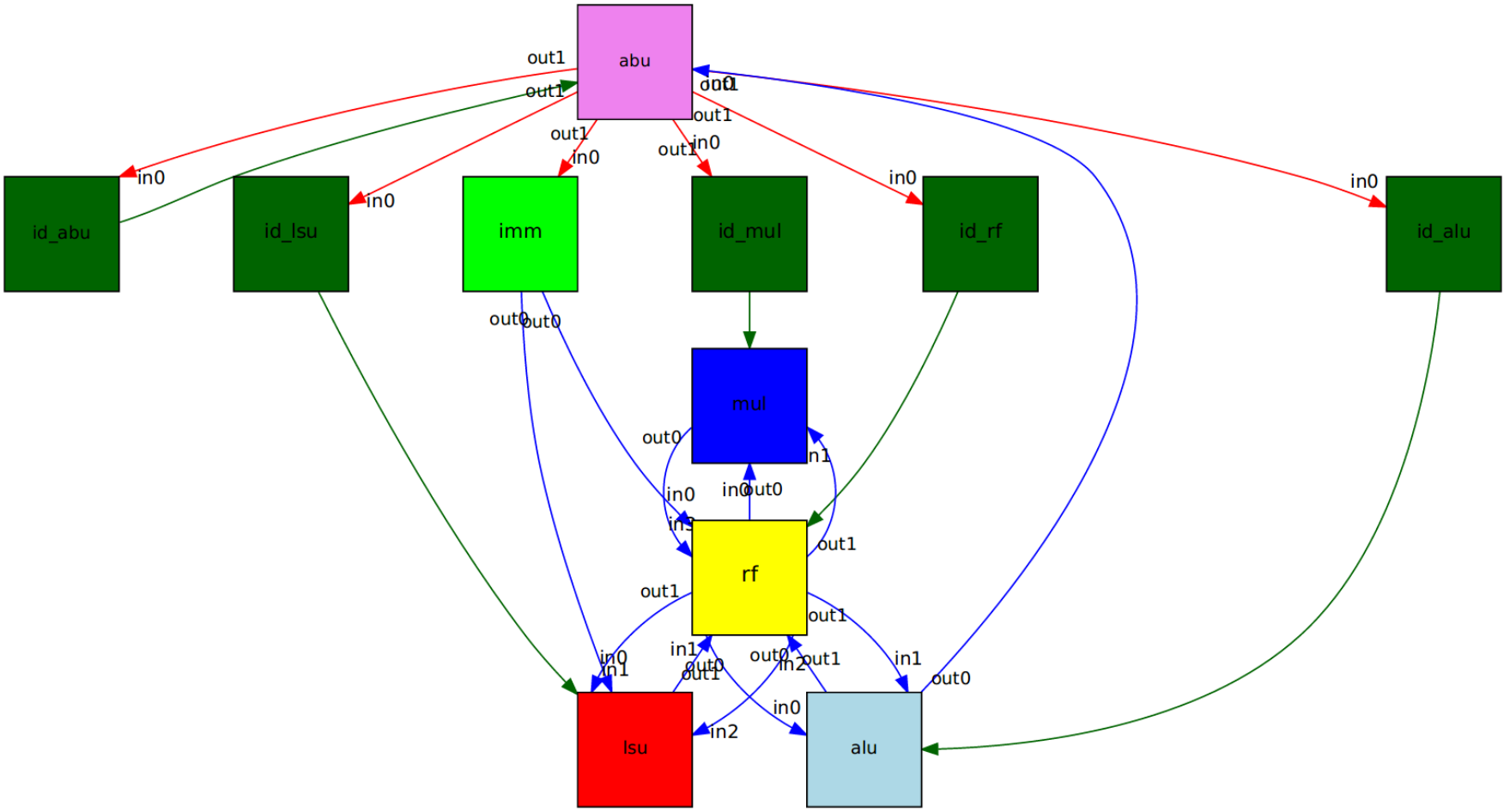
Using the CGRA

- Two things are required:
  - Architecture description
  - Program





Architecture description



Programming

- We would liked to give you a compiler...
  - Still in development.
  - Works but not yet using the architecture very efficiently.
- Programming is done in an assembler dialect
  - We call it PASM: Parallel Assembler

Some results

- Post place & route results for 2 benchmark applications
  - 40nm commercial library
  - Targeted at 100 MHz

Benchmark	Architecture	Cycles	Power [mW]	Energy [nJ]
Binarization	Cortex-M0	115007	1.57	1806
	CGRA scalar	8209	1.07	88
	CGRA vector 4	2069	2.34	48
FIR	Cortex-M0	665618	0.98	6523
	CGRA	2224	9.48	211

37X

31X

Reconfigurable architectures

- Lower static control power than FPGA
  - Higher granularity means less control bits
  - Reconfiguration is 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

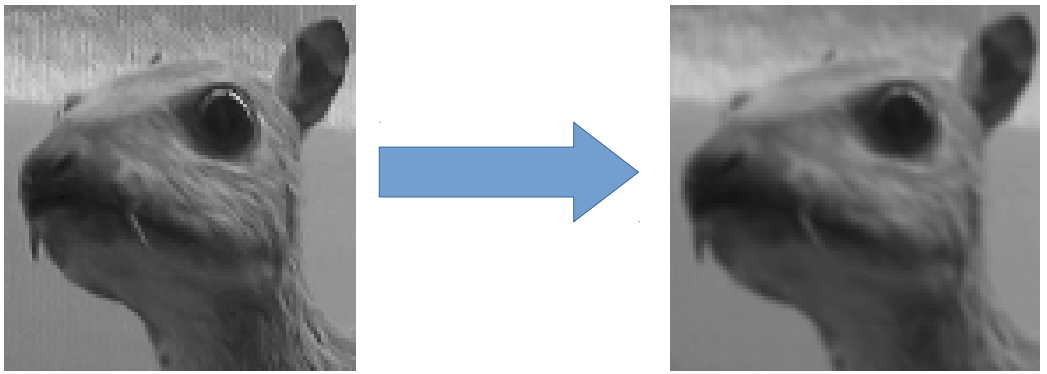
Current developments

- Single cycle loop support
- Debug support
- Approximate compute platform
- Compiler (LLVM, Roel Jordans, talk tomorrow)

- Tape-out plans:
  - Small design in October
  - More complex/optimized design in May 2018.

The assignment

- You will get a naive implementation for a Gaussian blur convolution kernel.



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

Your assignment

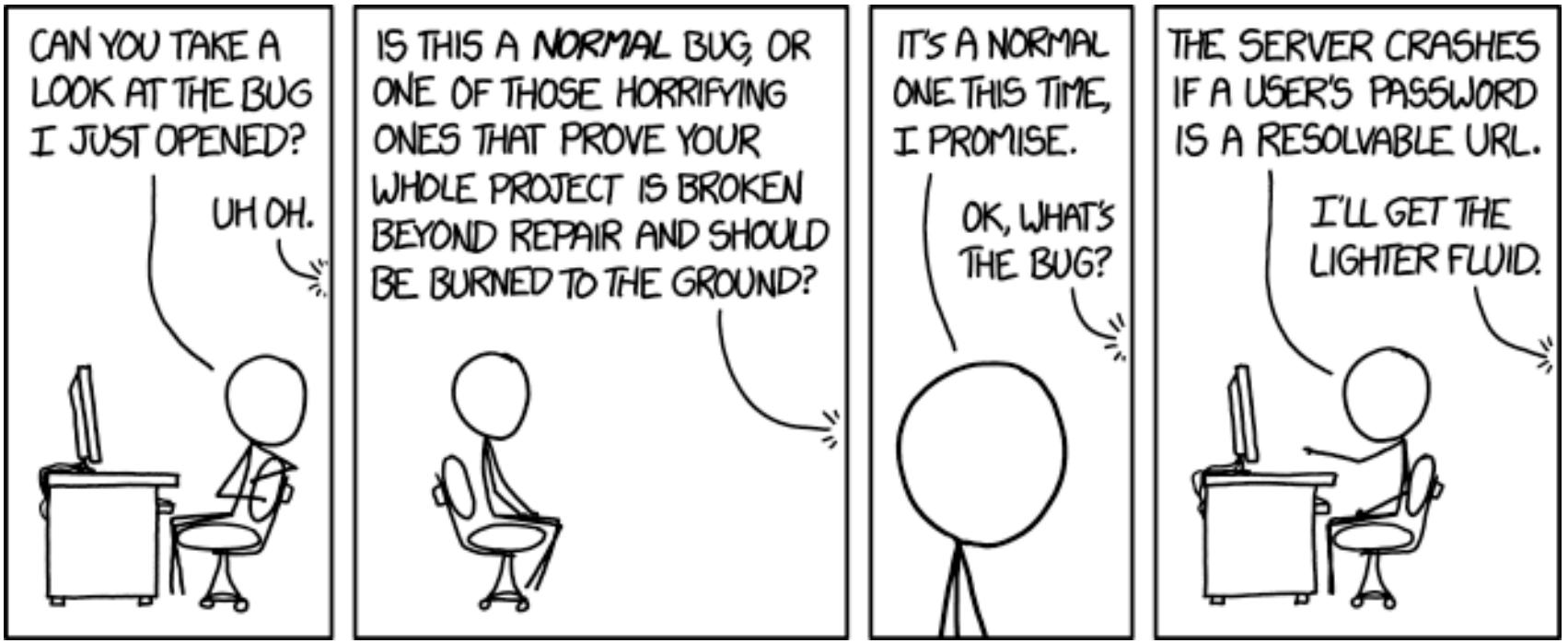
- You can:
  - Modify the architecture:
    - Implement data-level parallelism
    - Implement instruction-level parallelism
    - Use bypassing
    - Use other nice hardware features
  - Modify the application:
    - There are algorithm level optimizations possible
    - To make use of the architecture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

Your assignment

- The assignment document will describe everything in more detail.
  - Additional documentation and files can be found on asci.cgra.nl
- Tools are available to make energy, area and performance estimates.

One more thing...

- This is a research architecture…
  - Bugs will be present.
  - You will be among the first users.



Want to do the assignment?

Mark Wijtvliet (m.wijtvliet@tue.nl)
(or register at the forum at asci.cgra.nl and start directly)