Accelerators Development

26 Feb 2020

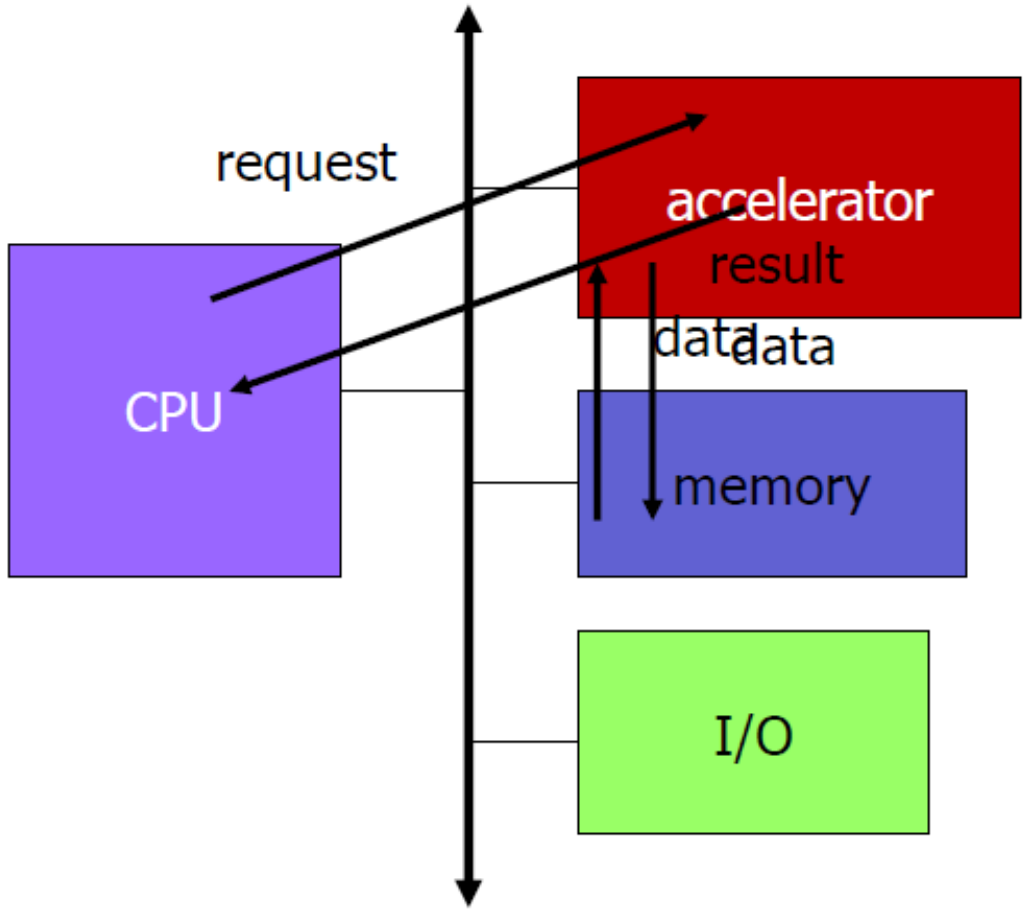
XILINX
ALL PROGRAMMABLE™

Outline

- Motivation
- Accelerators coupling to CPUs

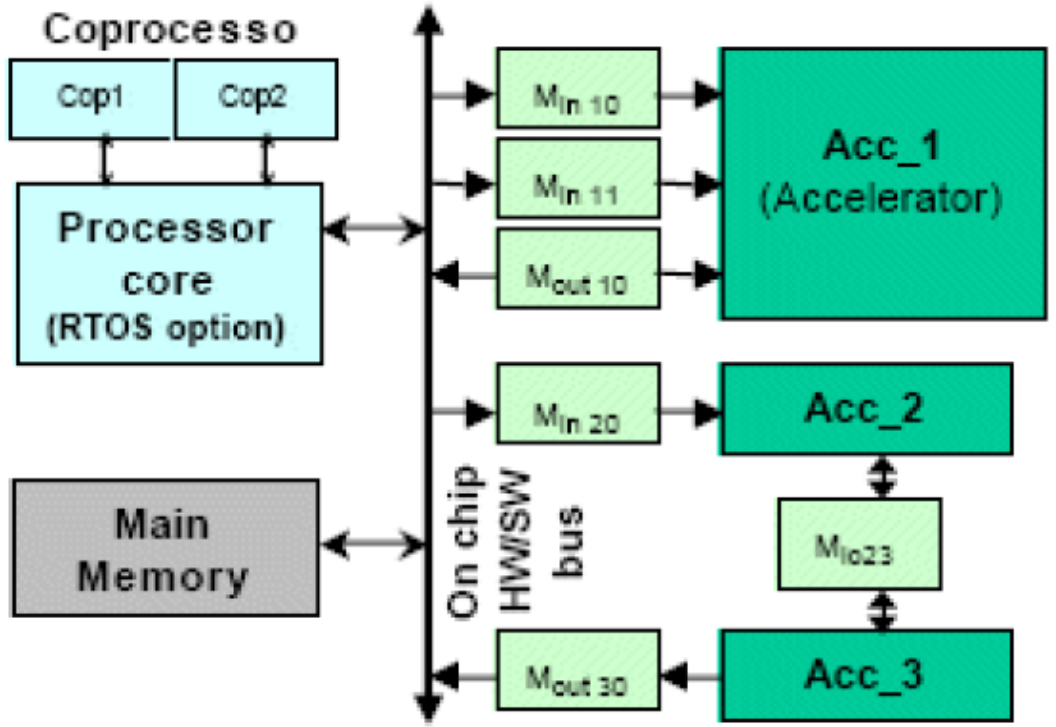
Accelerated Systems

- Use additional computational units dedicated to some functionality
- Hardware/software co-design: joint design of hardware and software architectures



Accelerator vs Coprocessor

- A coprocessor executes instructions
  - Instructions are dispatched by the CPU
- An accelerator appears as a device on the bus
  - Typically controlled by registers (memory-mapped IO)



Accelerated System Design

- First, determine that the system really needs to be accelerated
  - How much faster will the accelerator on the core function?
  - How much data transfer overhead? Compute bound vs memory bound vs I/O bound?
- Design accelerator and system interface
- If tighter CPU integration required:
  - Create a functional unit for augmented instructions
  - Compiler techniques to identify/use new functional unit

Amdahl's Law

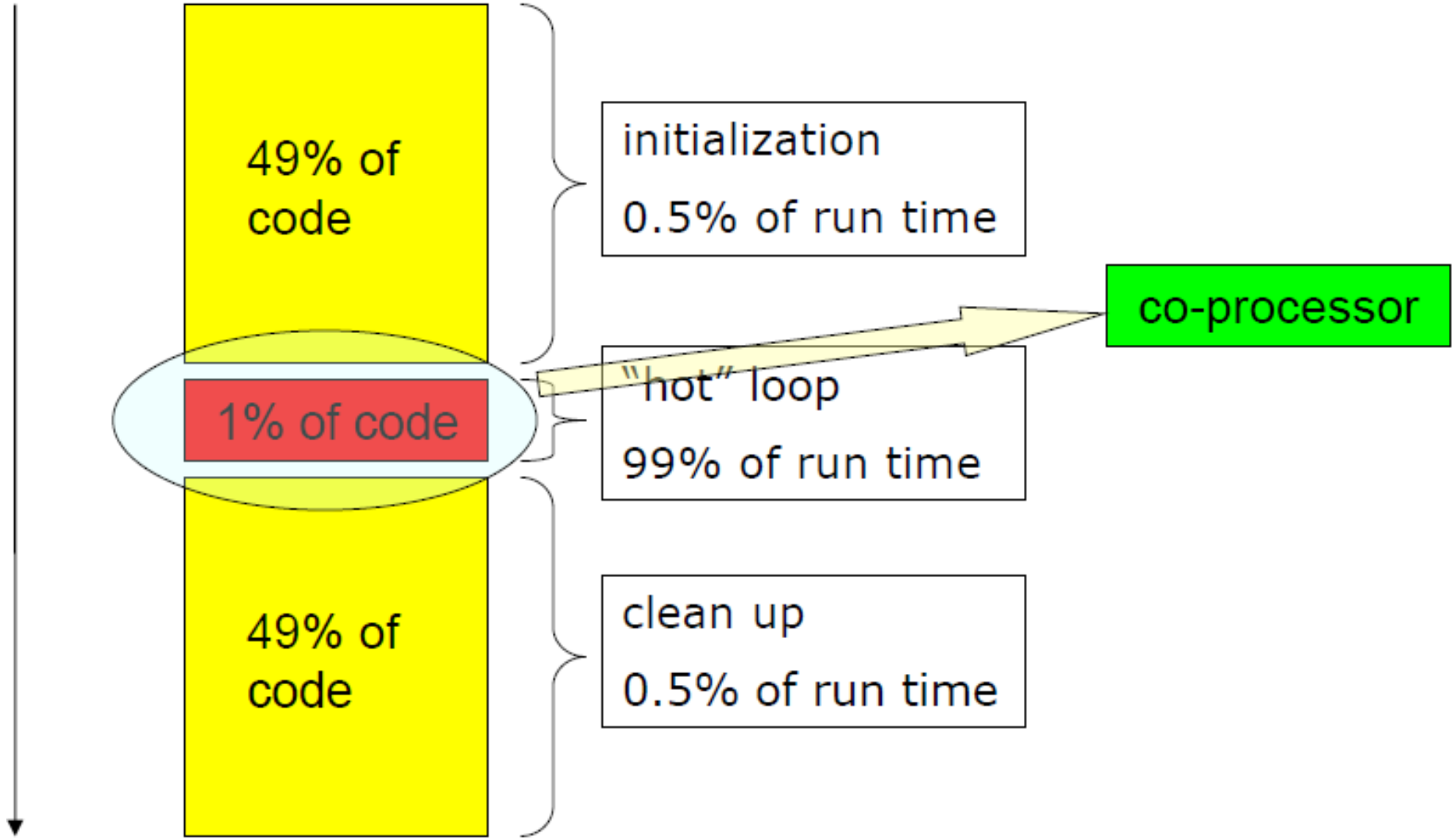
- If an optimization improves a fraction *f* of execution time by a factor of *a*

$$Speedup = \frac{Told}{[(1-f)+f/a]\times Told} = \frac{1}{(1-f)+f/a}$$

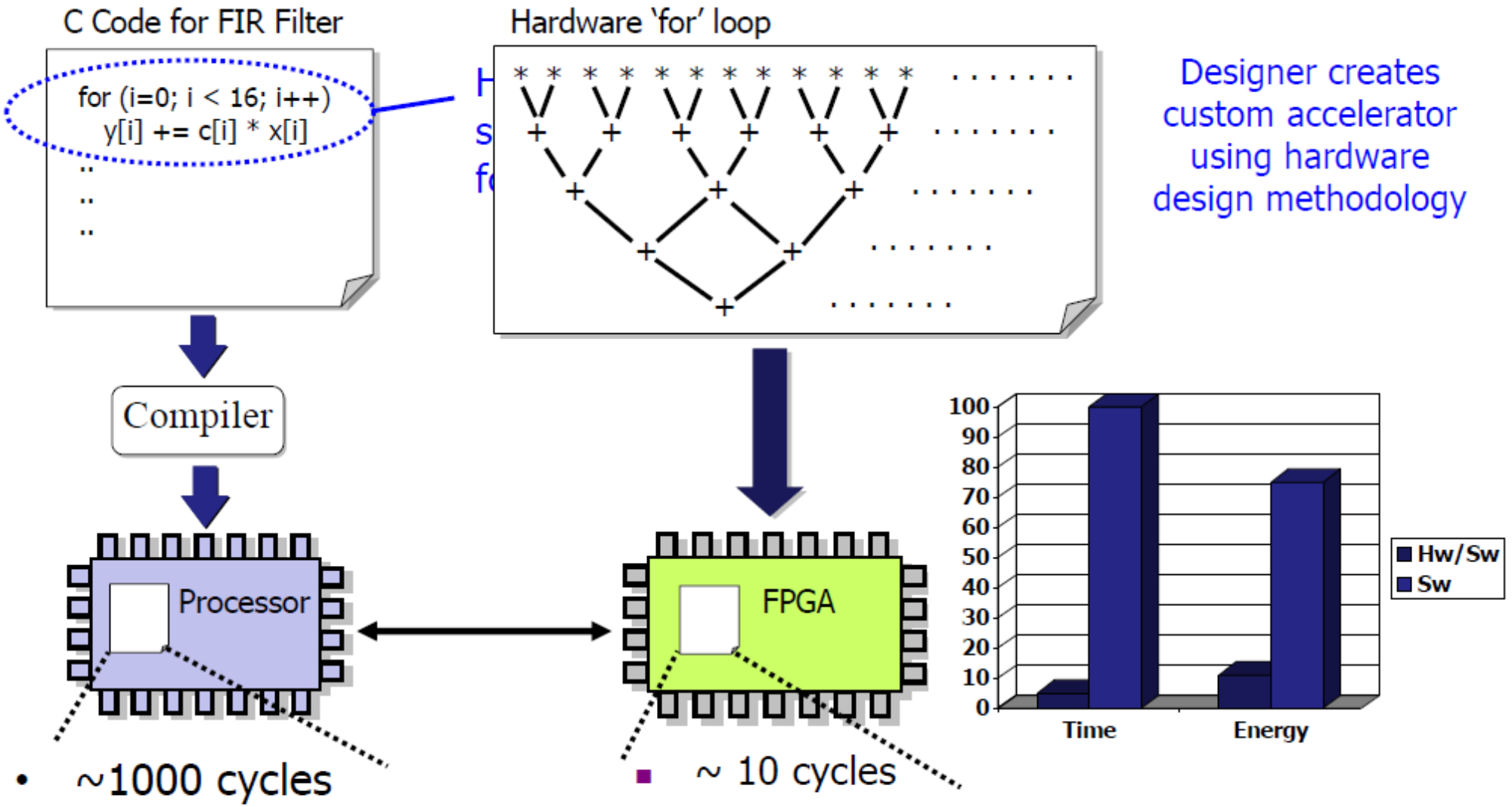
- This formula is known as Amdahl's Law
- Lessons from
  - If f $\rightarrow$100%, then speedup = *a*
  - If *a* $\rightarrow\infty$, the speedup = 1 / (1 – *f*)
- Summary
  - Make the common case fast
  - Watch out for the non-optimized component
-

Heterogeneous Execution Model

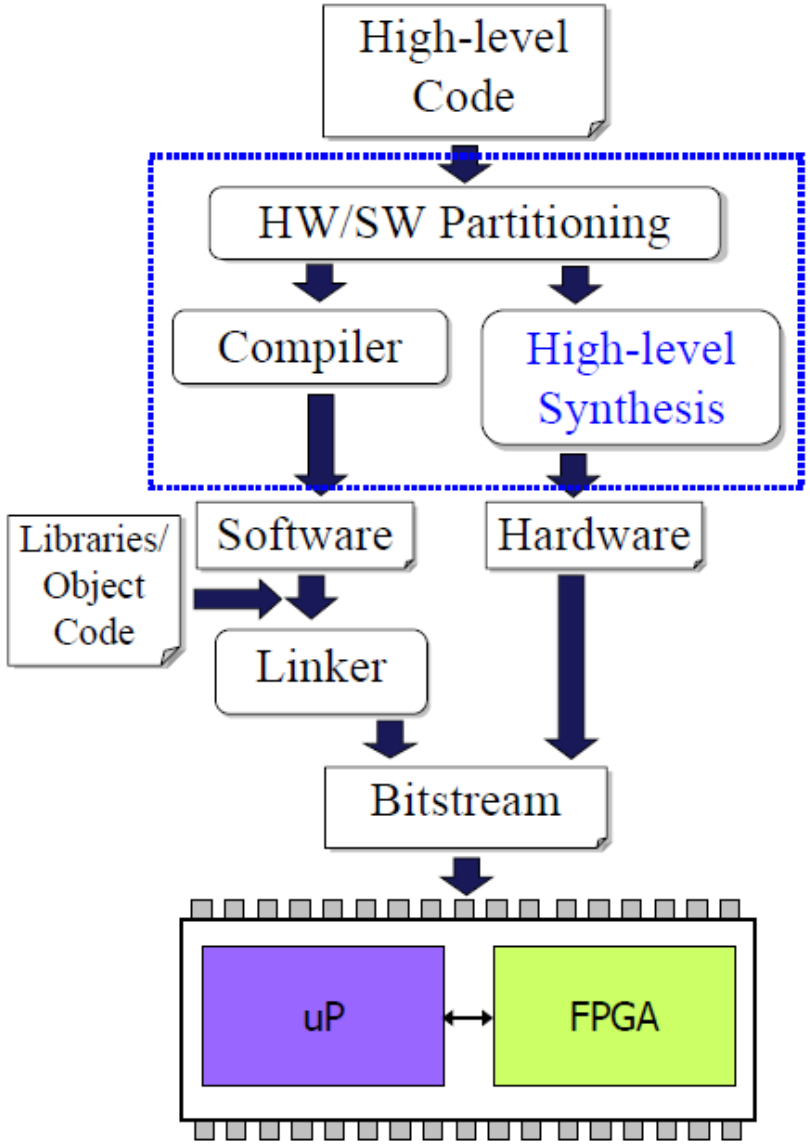
instructions executed over time



Hardware / Software Partitioning



High-Level Synthesis

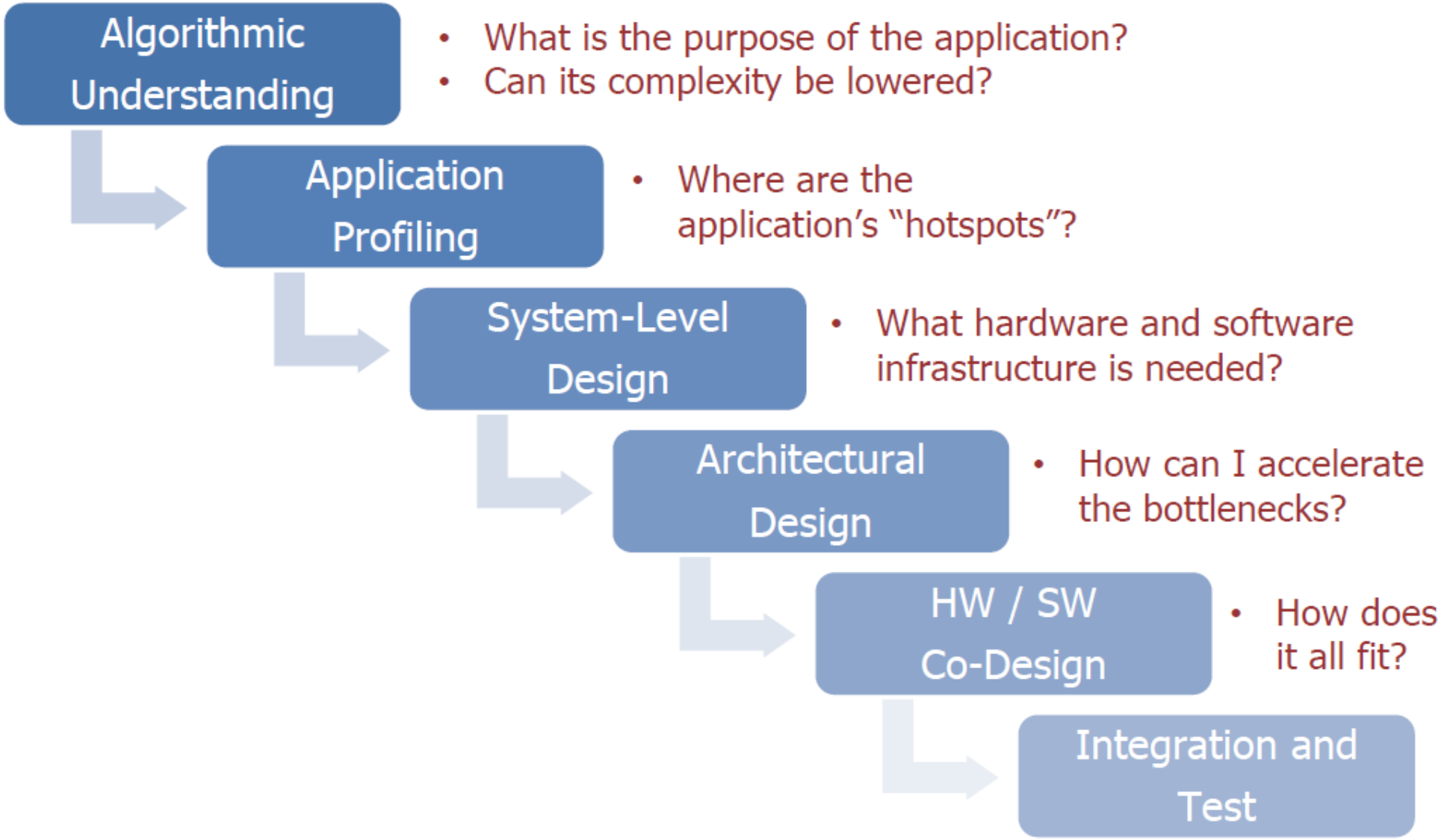


- *Problem:* Describing circuit using HDL is time consuming/difficult
- *Solution:* High-level synthesis
  - Create circuit from high-level code
  - Allows developers to use higher-level specification
  - Potentially, enables synthesis for software developers
- More on this in a bit

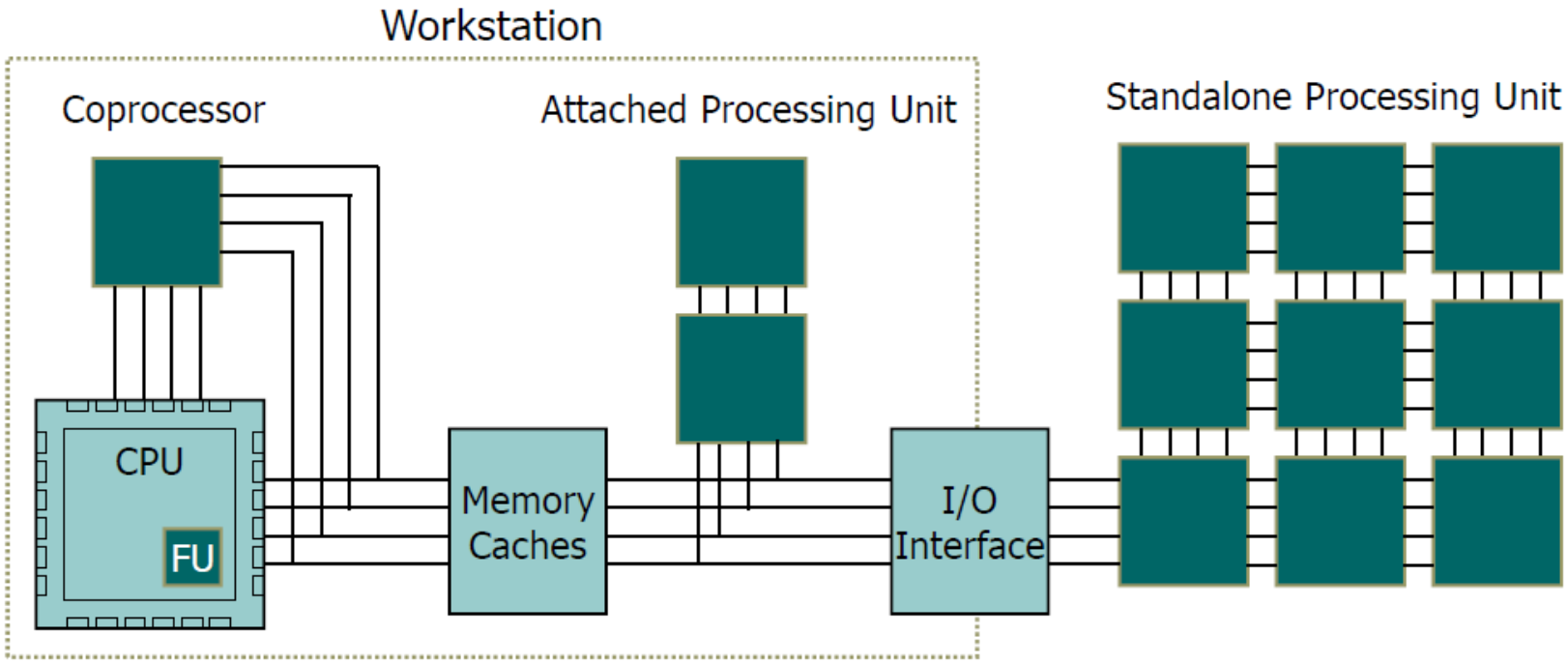
Accelerator Design Challenges

- Debugging – how to properly test the accelerator separately, and then in conjunction with the rest of the system (hw/sw co-simulation)
- Coherency – how to safely share results between CPU and accelerator
  - Impact on cache design, shared memory
  - Solutions looks similar to those for resource sharing in conventional operating systems, but are typically ad-hoc
- Analysis – determining the effects of any hardware parallelism on performance
  - Must take into account accelerator execution time, data transfer time, synchronization overhead
  - Heterogeneous multi-threading helps, but complicates design significantly
  - Overlapping I/O and computation (streaming)

Typical Application Acceleration Methodology

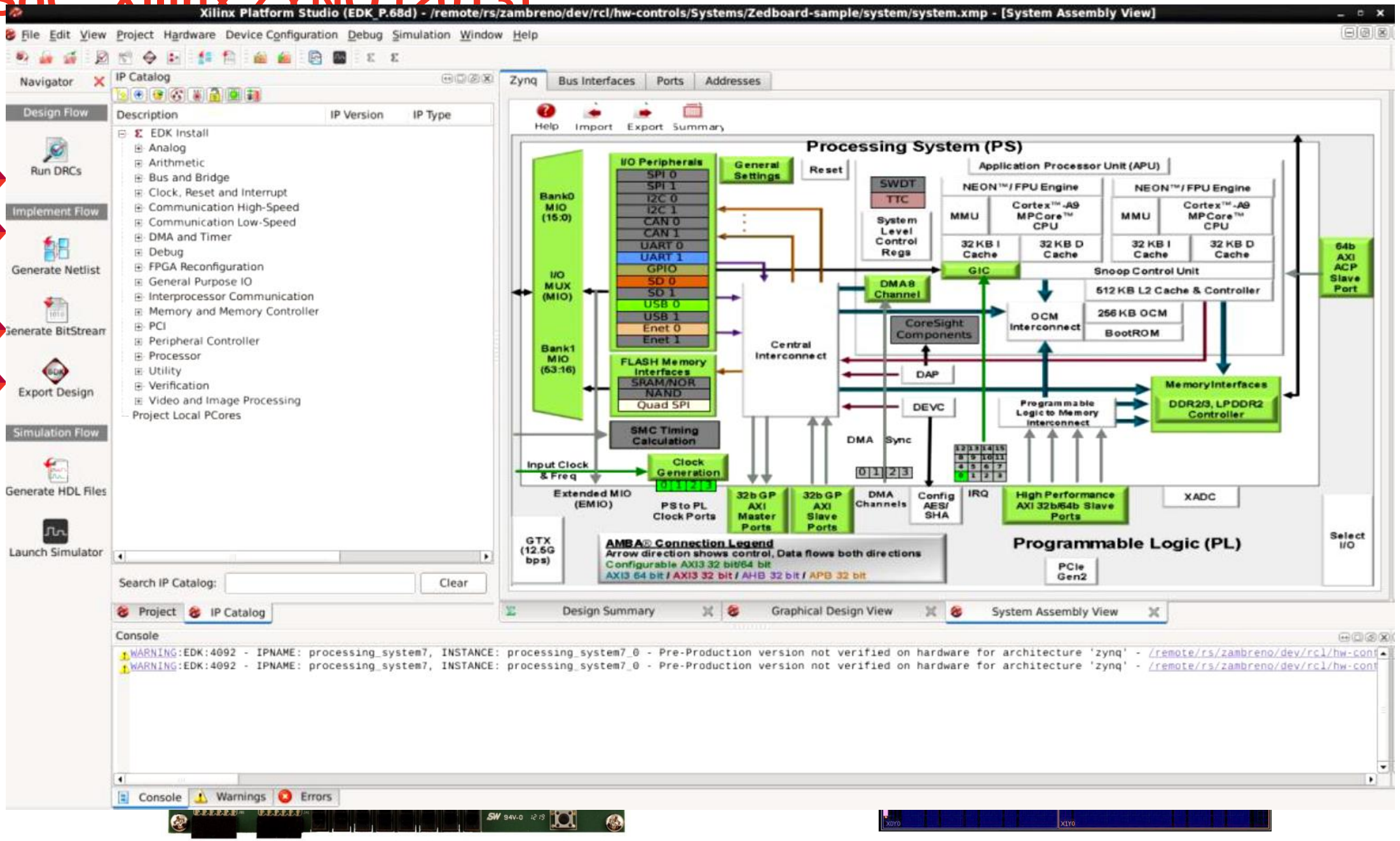


Accelerator Proximity



- Although self-reconfiguration is possible, some software integration with a reconfigurable accelerator is almost always present
- CPU – FPGA proximity has implications for programming model, device capacity, I/O bandwidth

SoC: Xilinx ZYNQ (2013)

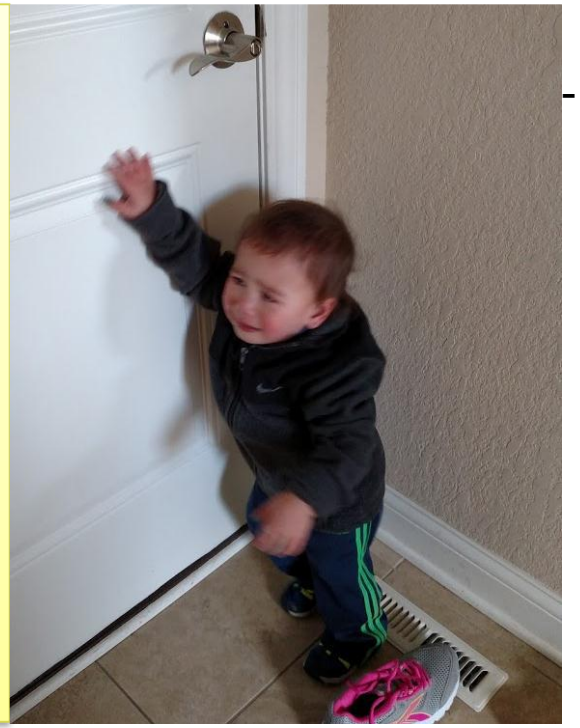


Solution: Automation (?)

- Developer e
  - Numerou
    level des
  - C-to-HDL
  - Handel-C
  - ROCCC (
  - Catapult (
  - SystemC
  - Cynthesiz
  - ImpulseC
- Many other comparable approaches:
  - HDL Coder (Mathworks)
  - Vivado High-Level Synthesis (Xilinx)
  - Bluespec, SystemVerilog

- Opinion: these tools can automate certain classes of logic, BUT:
  - Cannot generate efficient output for “hard problems”
  - Unfamiliar / uncomfortable syntax for both SW and HW engineers
  - Similar algorithmic challenges to auto-parallelizing compilers
  - Sorry students, you’re still learning VHDL….

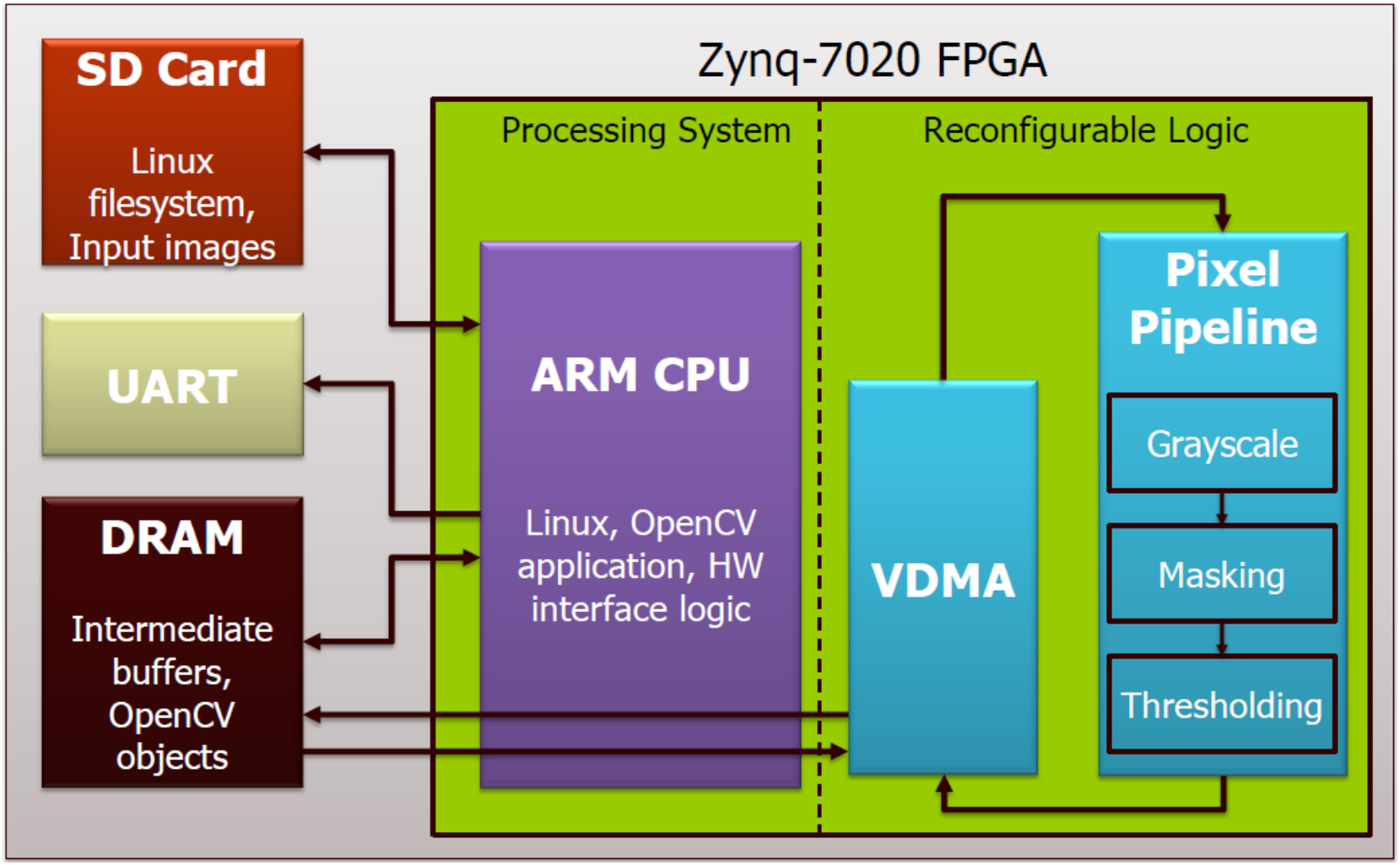
Escaping the Lab …???



Important Remarks

- Applications that map well to FPGA-based acceleration tend to have common characteristics:
  - Significant kernels of computation, significant data
- Amdahl's law is typically more relevant than Gustafson's law
- Profile. Don't Speculate. – Daniel Bernstein
  - Fixed-point or integer data representation
- If application is Gflop-constrained, use a GPU
- Changing (slowly), see Altera Stratix 10-based systems
  - Fine-grained parallelism
- But if working set fits in cache, will still be hard to beat x86 (MHz for MHz)
- Systolic model of computation, where FPGA pipeline depth > equivalent CPU depth and number of FPGA PEs >> number of x86 FUs
  - Real-time constraints, system integration
- HPC workloads should go on HPCs (i.e. accelerators are an orthogonal concern)
- Current GPUs cannot make useful latency guarantees

System-level Design




Thank you!