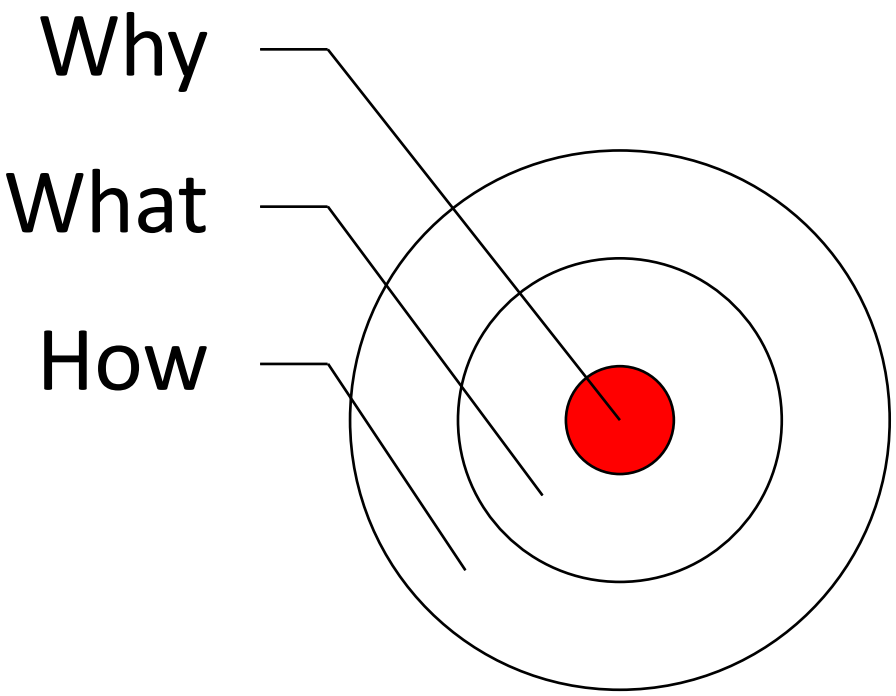
# CS249r

What is DSA and how do you go about it?

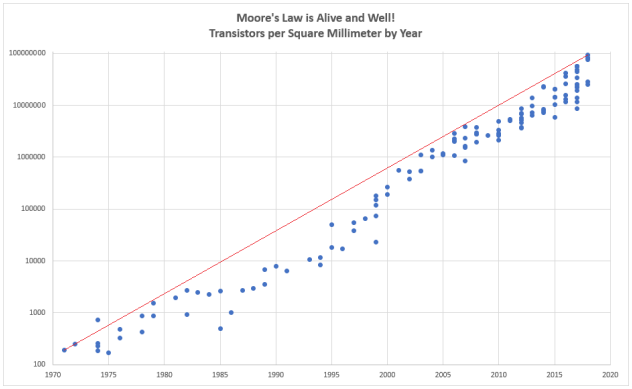
# Logistics

- Paper/HotCRP review depth and details
- Takeaway concepts from the lectures

# Goal Toward Domain Specific Architectures for Autonomous Machines

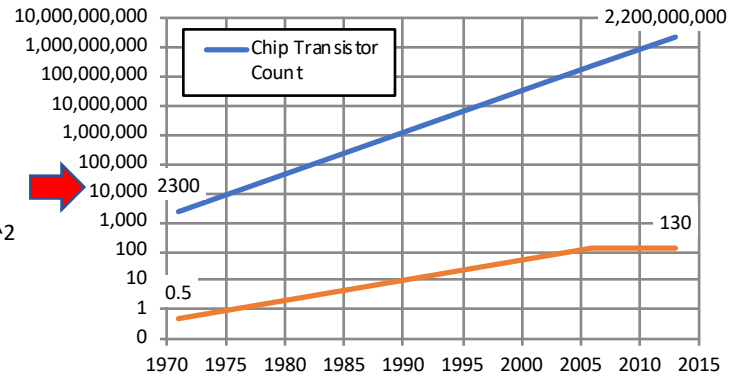


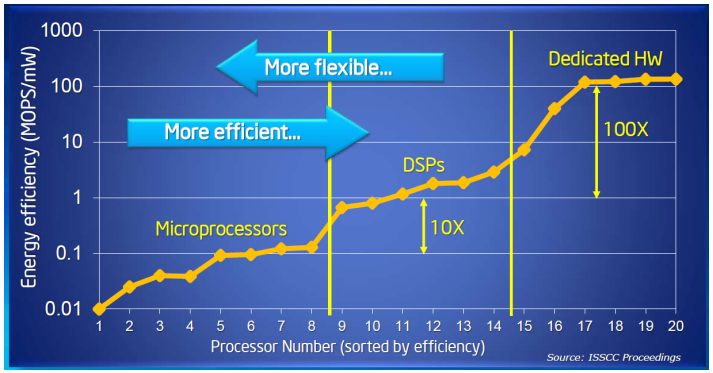
# **Why** are we talking about DSA?

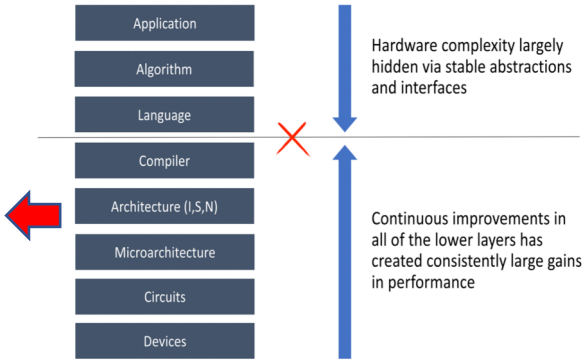


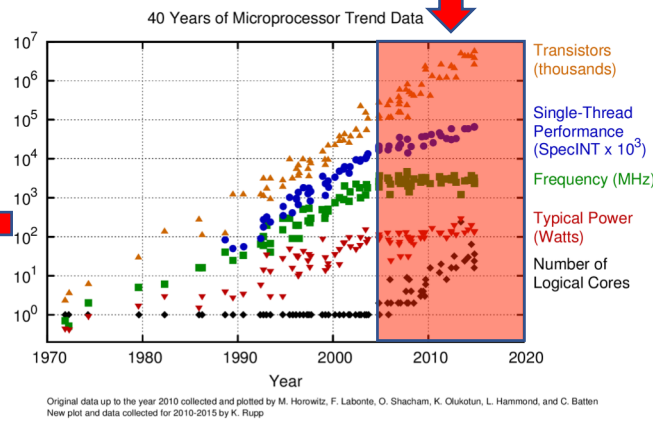

| Device or Circuit Parameter | Scaling Factor |
|---|---|
| Dimension, Tox, L, W | 1/k |
| Doping Concentration Na | k |
| Voltage (V) | 1/k |
| Current (I) | 1/k |
| Capacitance (eA/t) | 1/k |
| Delay time/circuit (VC/I) | 1/k |
| Power dissipation/circuit (VI) | 1/k^2 |
| Power density (VI/A) | 1 |

Historically, k ~= 1.4

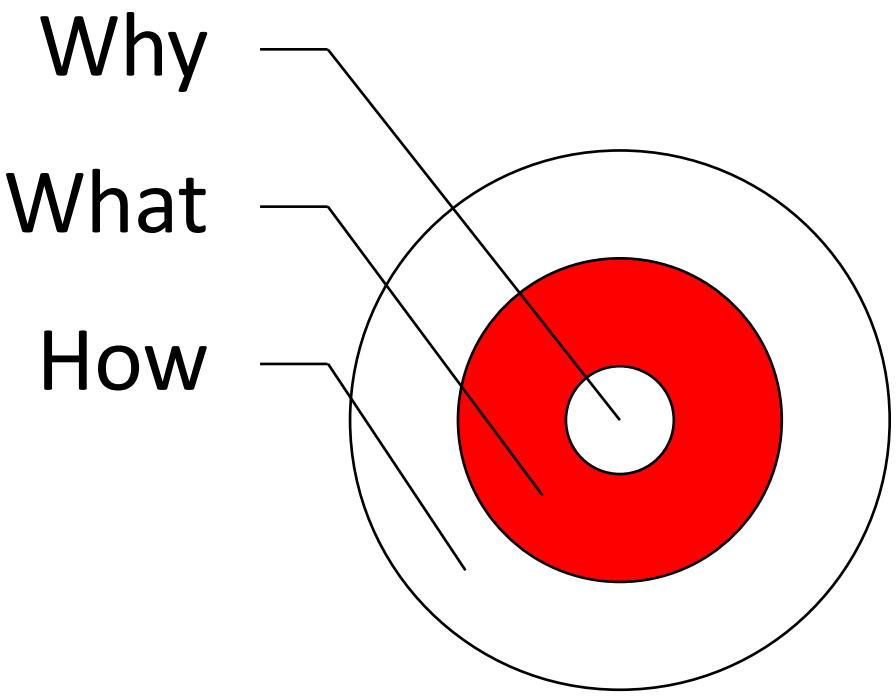


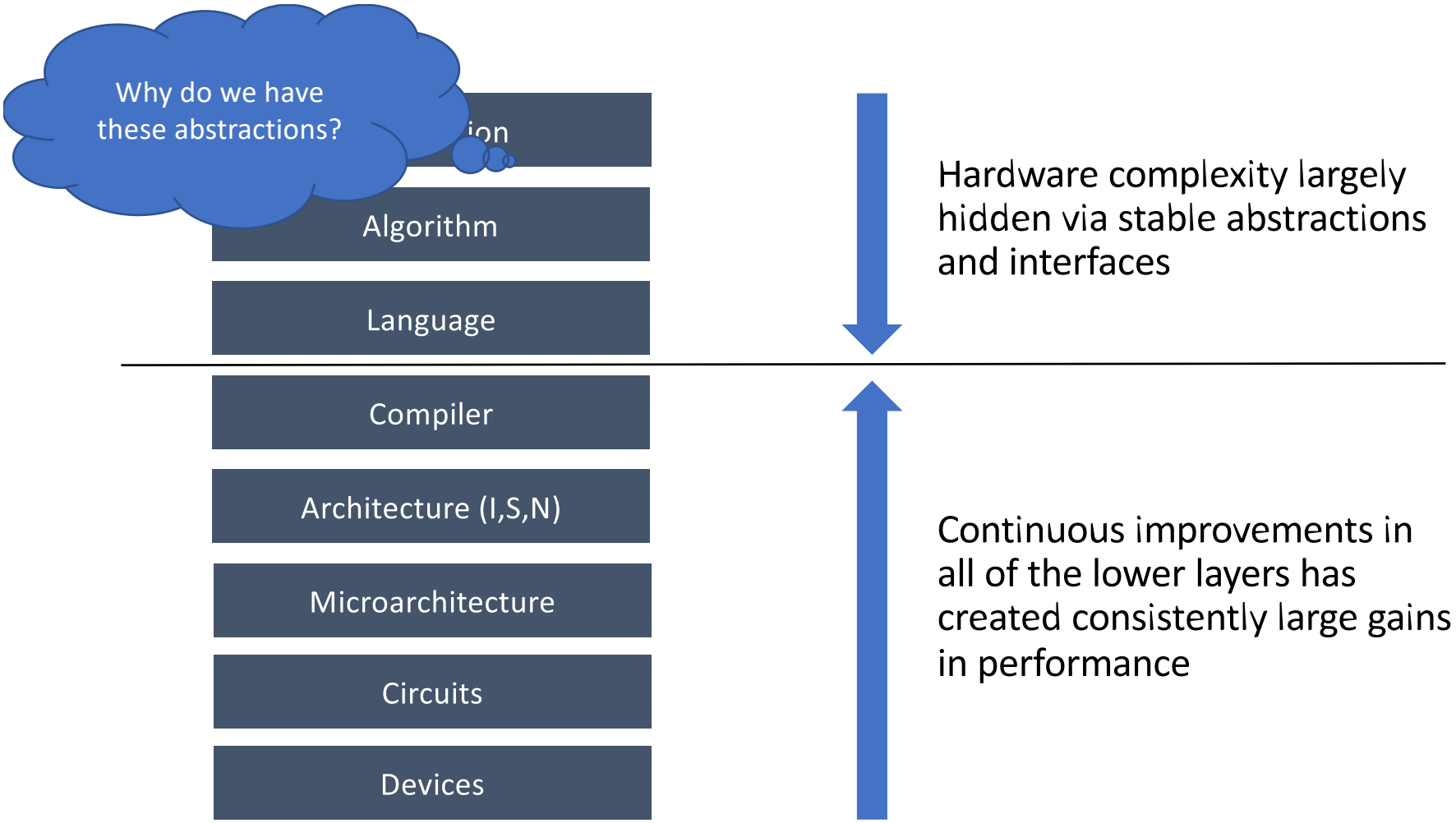



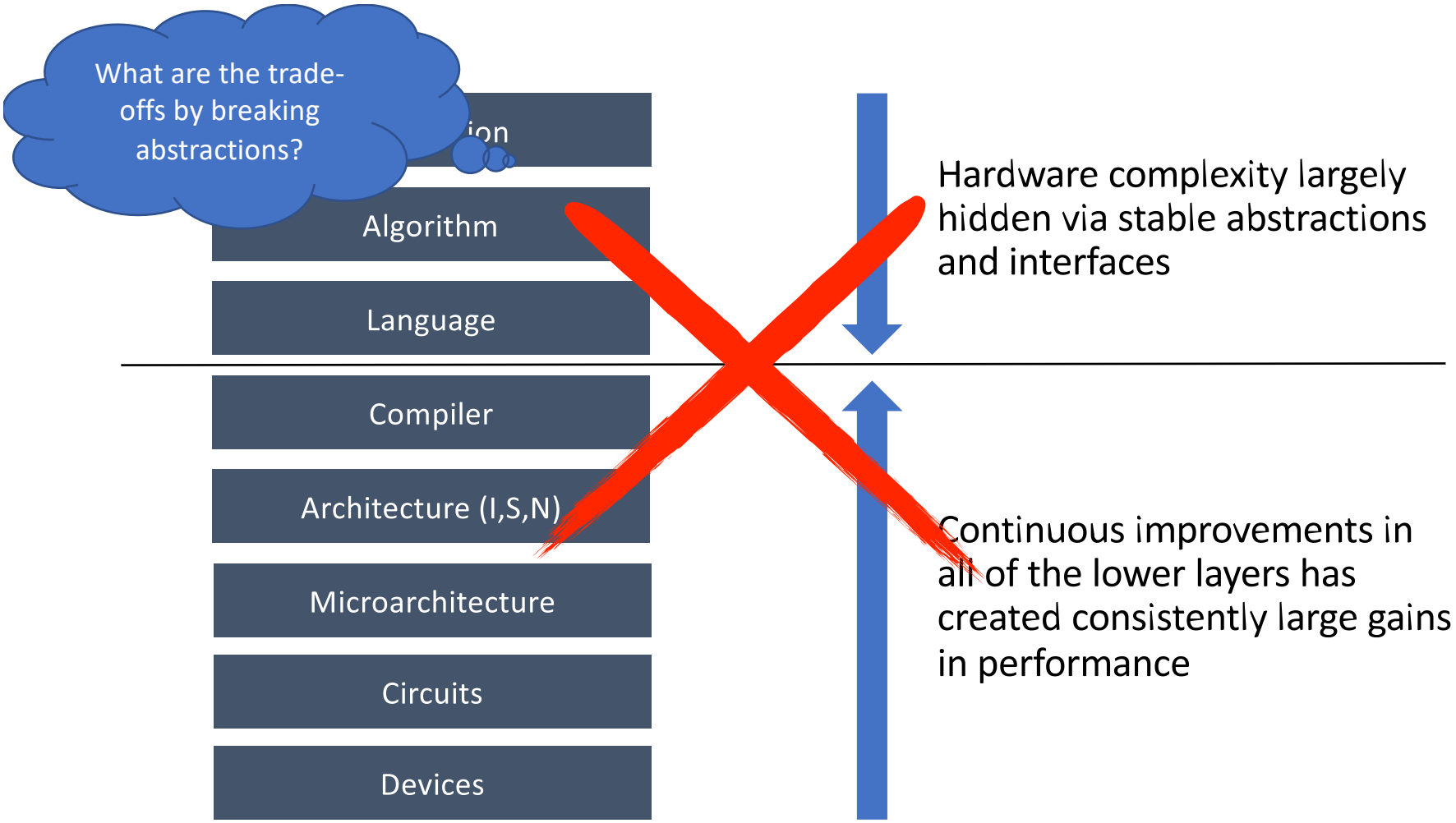



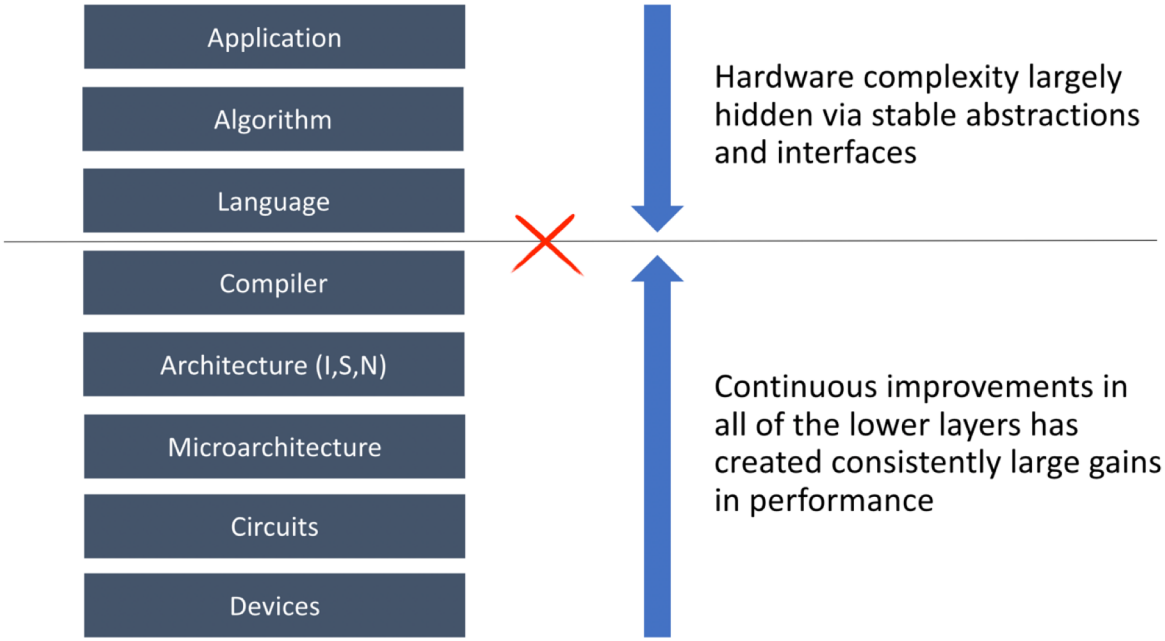
# Goal Toward Domain Specific Architectures for Autonomous Machines




Why do we have these abstractions?
Algorithm
Language
Compiler
Architecture (I,S,N)
Microarchitecture
Circuits
Devices
Hardware complexity largely hidden via stable abstractions and interfaces
Continuous improvements in all of the lower layers has created consistently large gains in performance


What are the trade-offs by breaking abstractions?
ion
Algorithm
Language
Compiler
Architecture (I,S,N)
Microarchitecture
Circuits
Devices
Hardware complexity largely hidden via stable abstractions and interfaces
Continuous improvements in all of the lower layers has created consistently large gains in performance

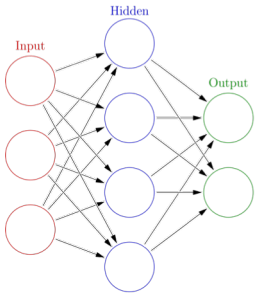
# Why break the abstractions?

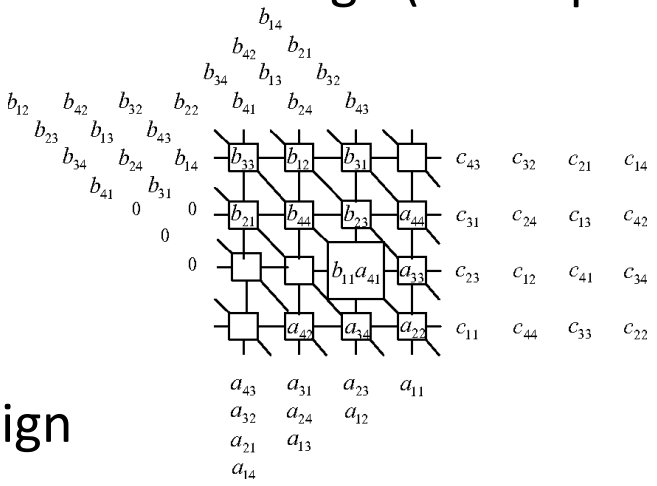



| | |
|---|---|
| Code specialization | 10x |
| Logic specialization | 100x |
| Circuit specialization | 1000x |
| Device specialization | 10000x |

# Co-design is at the heart of the solution

- **Hardware-software** co-design
  - What are some examples of hardware-software co-design?
- **Algorithm-hardware** software co-design (a.k.a specialization)



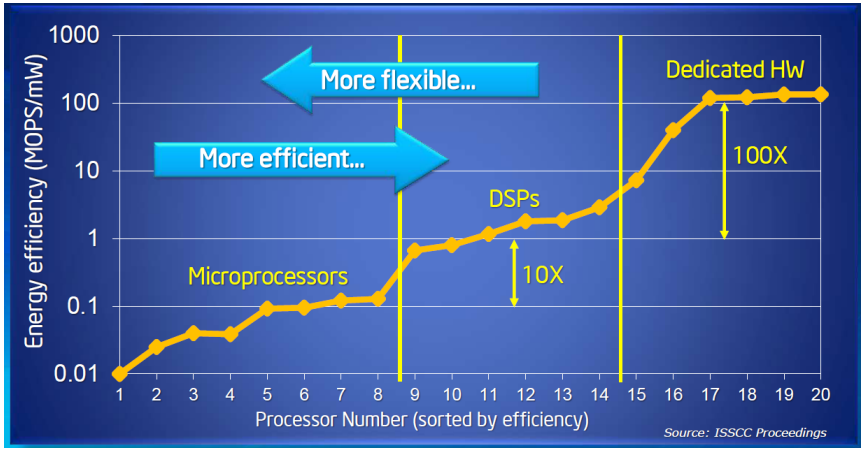



- **“Cyber-physical”** co-design

# Specialization makes a (big) difference

- Accelerators are much more efficient in their computation
- Need to figure out the right balance between flexibility versus efficiency
- There is no one right answer -- Why!?

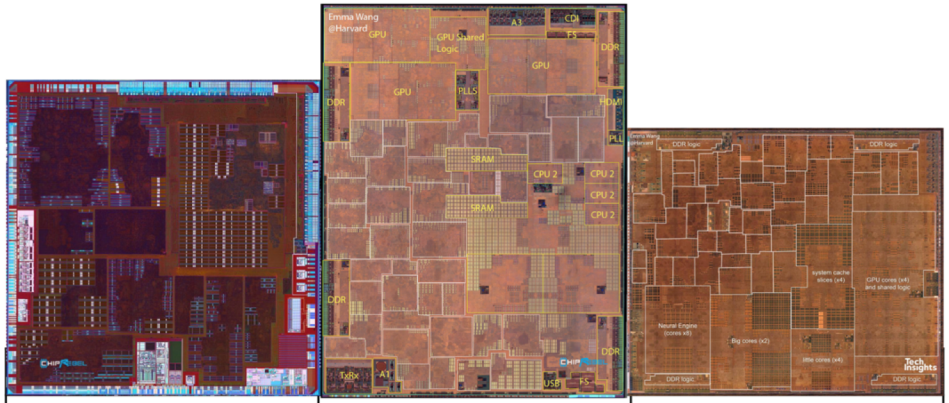
**Accelerators vs. Energy Efficiency**



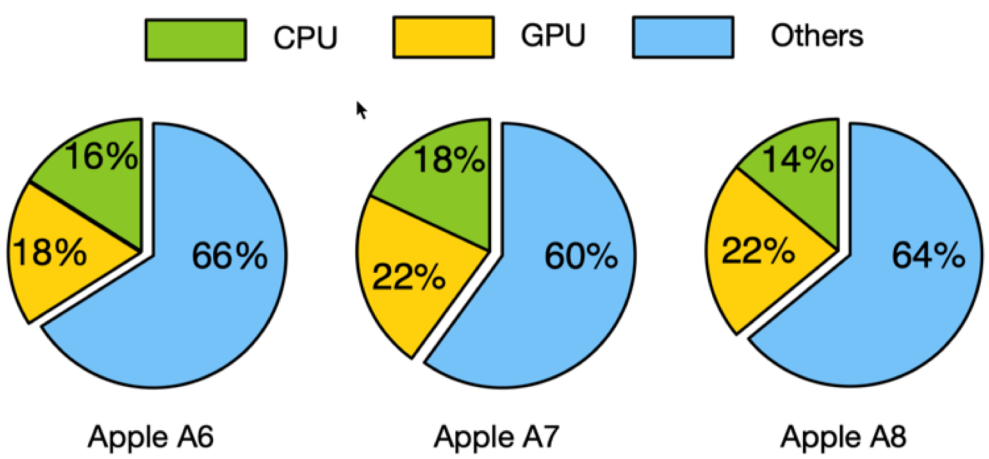
# Accelerators are widely used today

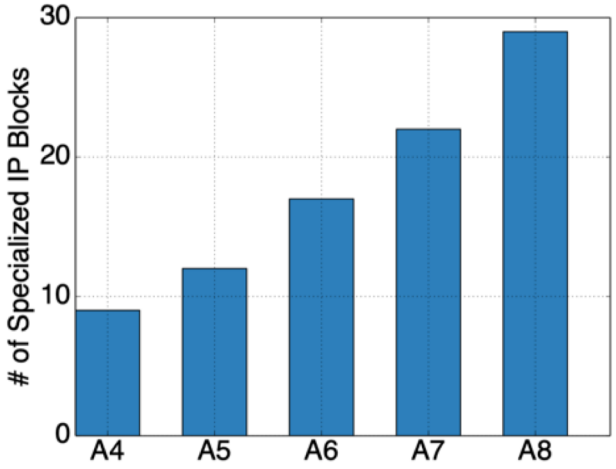
- Imagine the applications we use on a daily basis

- Accelerators are already widely deployed in the real world – ALP



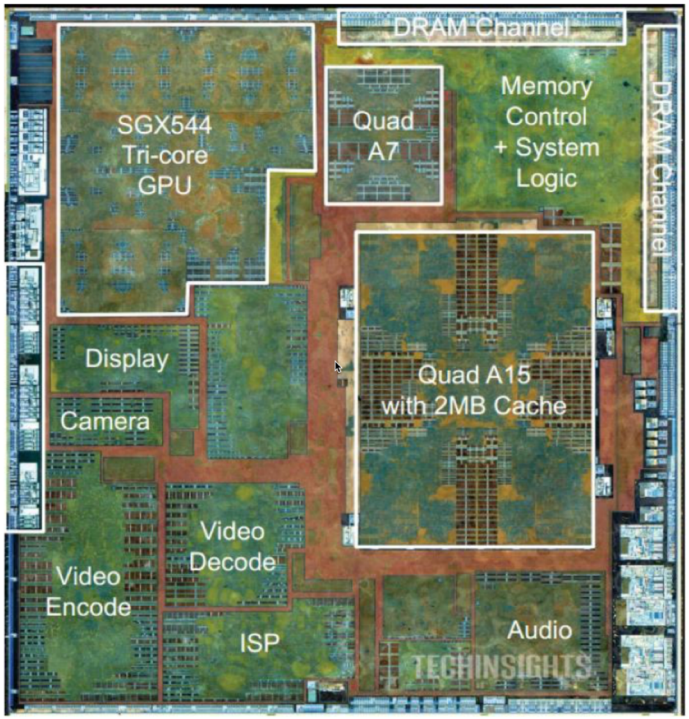
# Accelerators dominate state-of-the-art chips



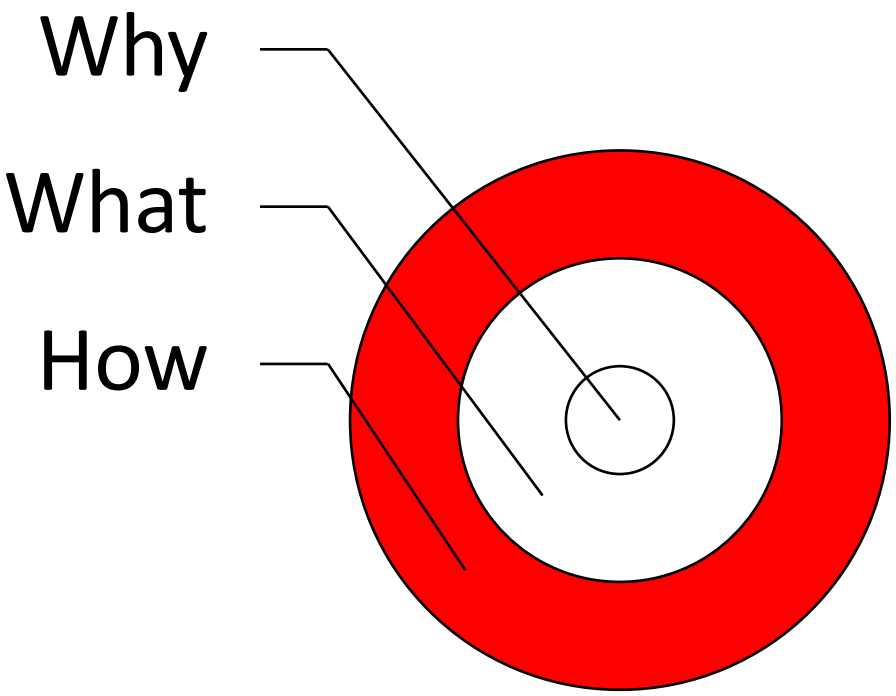



# Examples of what these accelerators are…

- Samsung's Exynos Octa processor
- The GPU, ISP, camera and video logic take up nearly as much area as the ARM Cortex-A7 and Cortex-A15 CPUs and associated cache



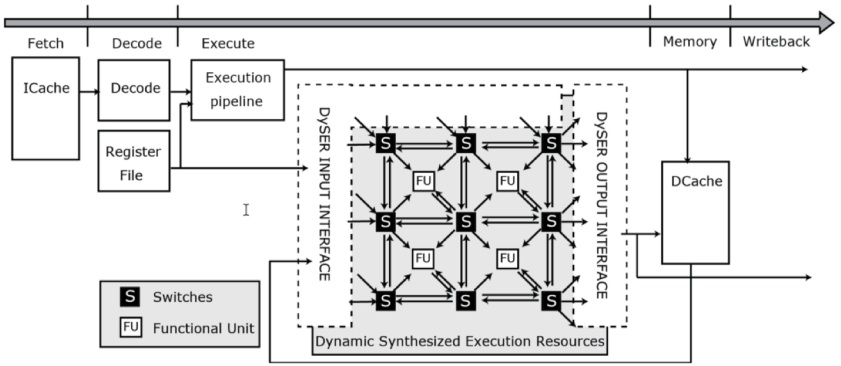
# Goal Toward Domain Specific Architectures for Autonomous Machines



# Need a framework by which to think about accelerators integration and execution

| | Part of the Pipeline | Attached to Cache | Attached to the Memory Bus | Attached to the I/O Bus |
|---|---|---|---|---|
| Instruction-Level | **FPU, SIMD,** DySER [58], | Hwacha [76, 95, 121], CHARM [43, 44], | | |
| Kernel-Level | NPU [52], 10x10 [40], Convolution Engine [98], H.264 [61], | SNNAP [91], C-Cores [119], | Database [35], Q100 [126], LINQits [41], AccStore [86], | |
| Application-Level | **x86 AES [18], Oracle/Cavium Crypto Acc [11, 69],** | Key-Value Stores [87], Memcached [80], | Sonic3D [103], DianNao [37, 38], HARP [125], **TI OMAP5 [16], IBM PowerEN [71], IBM POWER7+ [30],** | **GPU, Catapult [97], IBM Power8 CAPI Acc [4],** |

# Execution: Instruction-level Accelerators

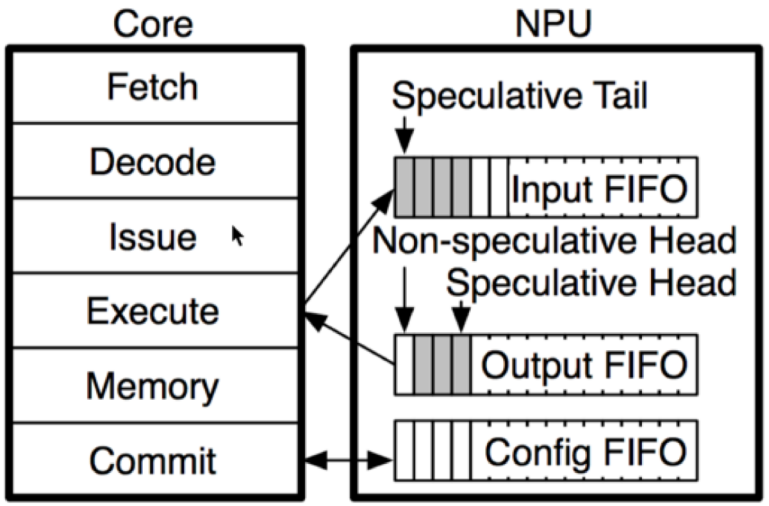



- Integrated into the datapath of the processor/CPU
  - e.g. sqrt


- Or integrated into the system as a co-processor

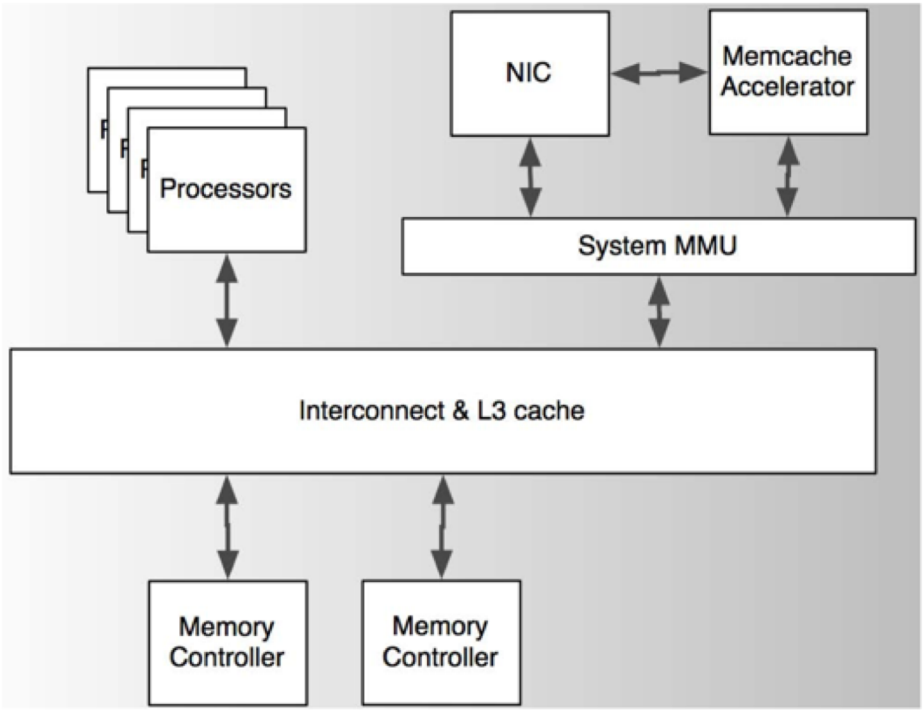
# Execution: Kernel-level Accelerators

- Compared to instruction-level accelerators, kernel-level accelerators look at bigger chunks
- E.g., multiple basic blocks, of computation (Matrix x Matrix, Matrix x Vector)



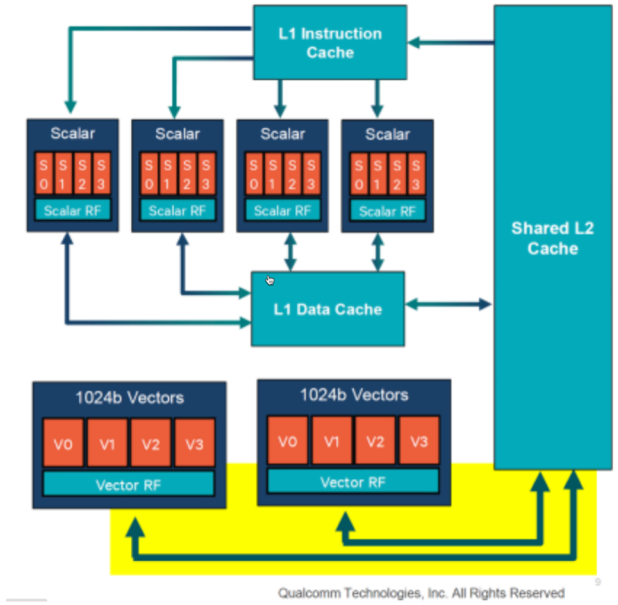
# Execution: Application-level Accelerators

- In-memory, key-value stores are an important component of modern data center services
- Memcached is implemented using a hash table, with a unique key used to index the stored data
- Thin Servers with Smart Pipes (TSSP) is designed for cost-effective, high-performance memcached deployment
- It couples an embedded-class low-power core to a memcached accelerator that can process GET requests entirely in hardware.

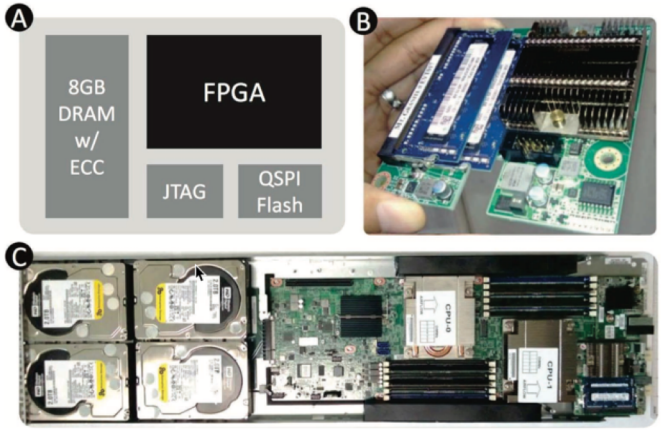


# Integration: Cache, Memory, Bus

**Cache Integration Example**




**I/O Bus Integration Example**

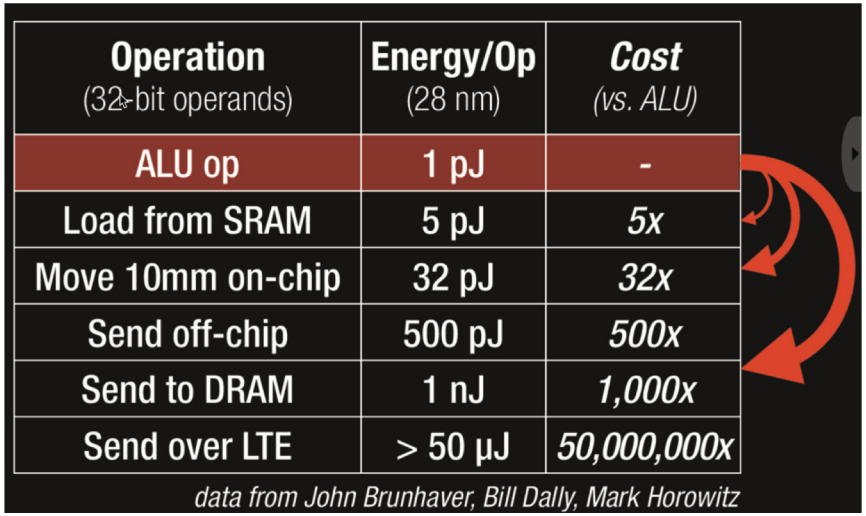


# DSA Guidelines

- **Use dedicated memories to minimize distances of data movement**
  - Hardware-controlled multi-level cache, domain-specific software controlled scratch-pad
- **Invest resources into more arithmetic units or bigger memories**
  - Core optimization (OoO, speculation, threading, etc), more domain-specific FU/memory
- **Use the easiest form of parallelism that matches the domain**
  - MIMD, SIMD or VLIW that matches domain
- **Reduce data size and type to the simplest needed for the domain**
  - General-purpose 32/64 integer/floatèdomain-specific 8/16 int/float
- **Use a domain-specific programming language**
  - General-purpose C/C++/Fortran Domain-specific language

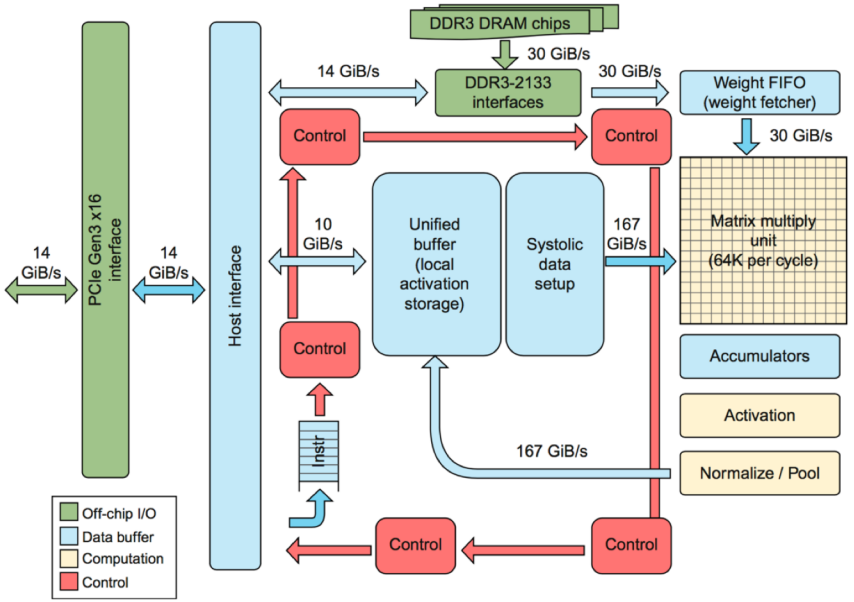
# DSA Guideline #1

- **Use dedicated memories to minimize distances of data movement**
  - Hardware-controlled multi-level cache, domain-specific software controlled scratch-pad



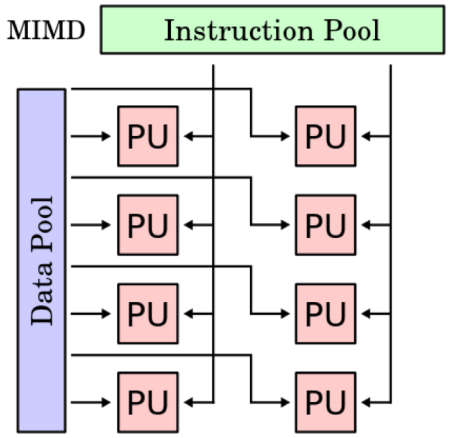
# DSA Guideline #2

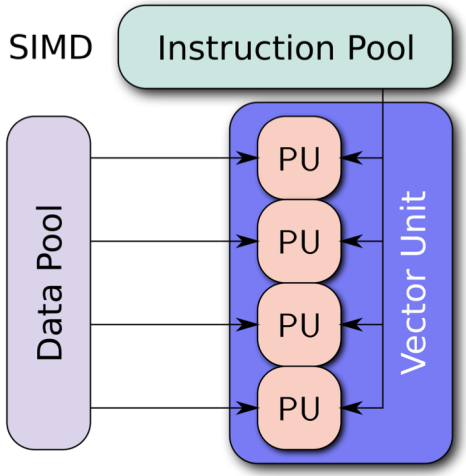
- **Invest resources into more arithmetic units or bigger memories**
  - Core optimization (OoO, speculation, threading, etc), more domain-specific FU/memory
  - Need to understand your workload to be able to make the right decision
  - Example of the TPU balancing the requirements for machine learning kernels, both compute and memory

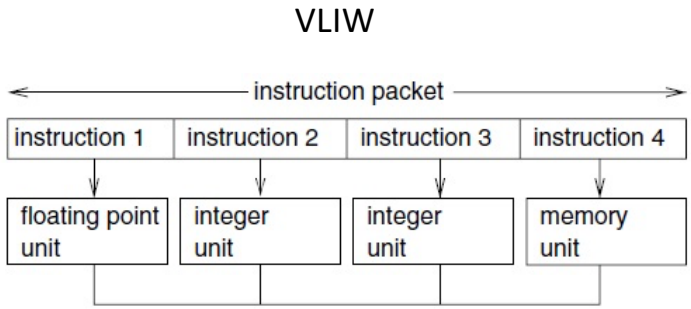


# DSA Guideline #3

- **Use the easiest form of parallelism that matches the domain**
  - MIMD, SIMD or VLIW that matches domain

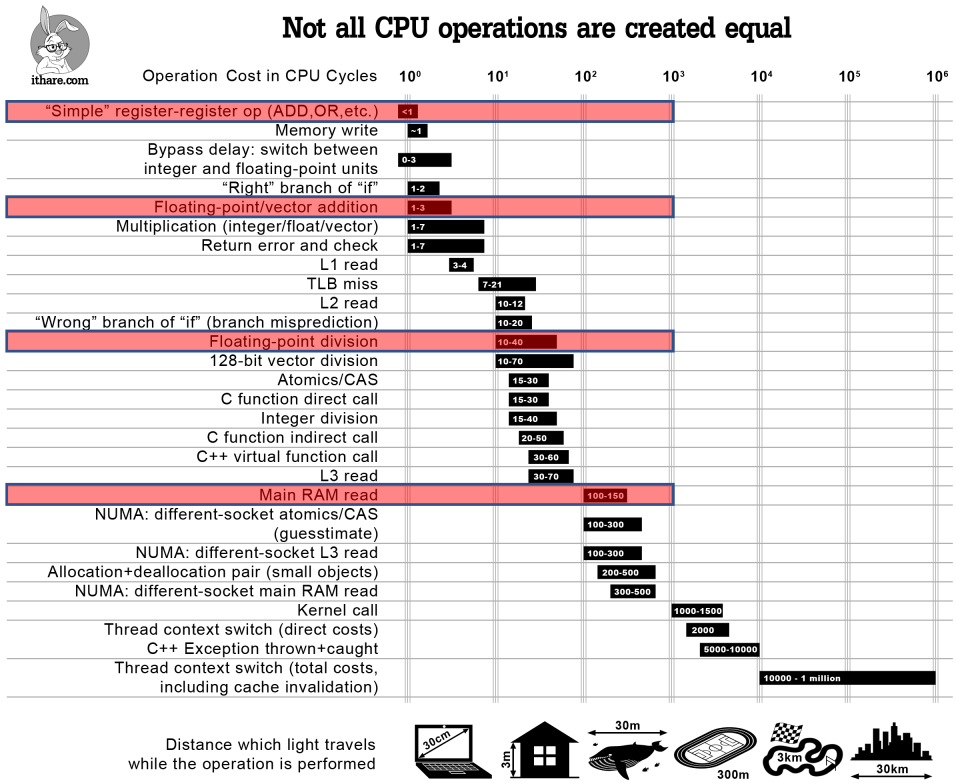


# DSA Guideline #4

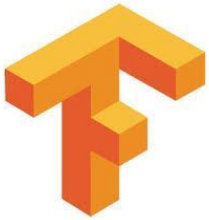
- **Reduce data size and type to the simplest needed for the domain**
  - General-purpose 32/64 integer/float domain-specific 8/16 int/float



# DSA Guideline #5

- **Use a domain-specific programming language**
  - General-purpose C/C++/Fortran Domain-specific language

Machine Learning



Machine Description

VHDL:

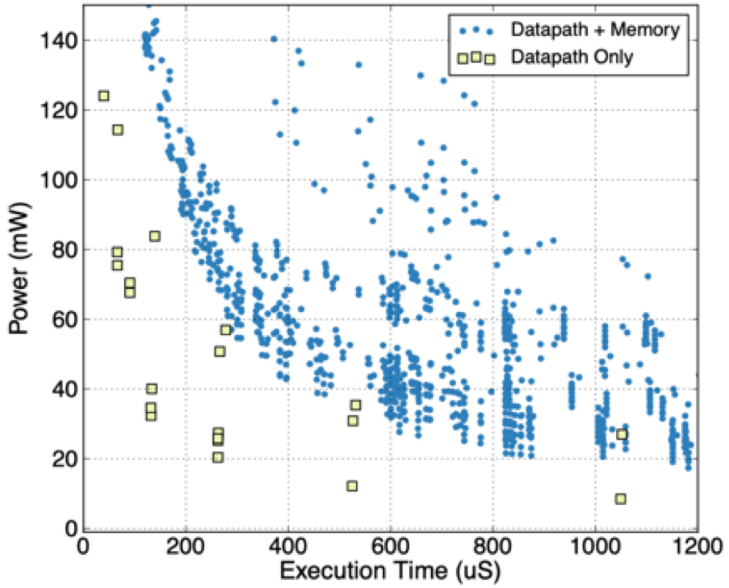
```
process (S0,S1,A,B,C,D)
begin
   case (S0,S1), is
      when "00" => Y <= A;
      when "01" => Y <= B;
      when "10" => Y <= C;
      when "11" => Y <= D;
      when others => Y <= A;
   end case;
end process;
```

Verilog:

```
always @((S0,S1), A, B, C, D)
      case ((S0,S1))
         2'b00: Y = A;
         2'b01: Y = B;
         2'b10: Y = C;
         2'b11: Y = D;
      endcase
```

# Unlocking the design space if you do DSE

- Accelerator design space is large given a range of architecture- and circuit-level alternatives.
- Exploring the design space exhaustively gives you a Pareto frontier of design choices.
- Allows hardware and software trade-offs (e.g. performance vs. power on the right-hand graph).



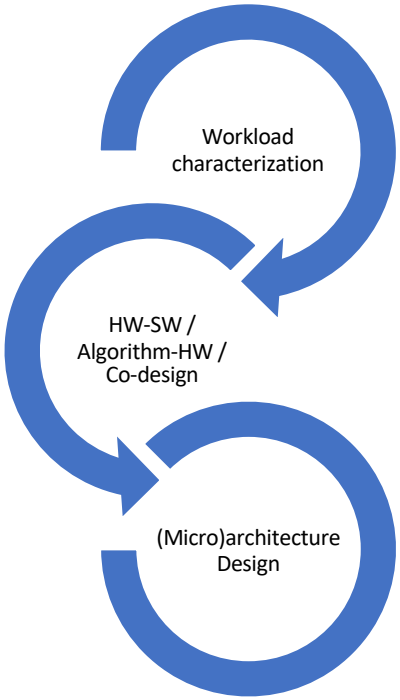

GEMM design space with and without memory hierarchy

# Examples of DSA

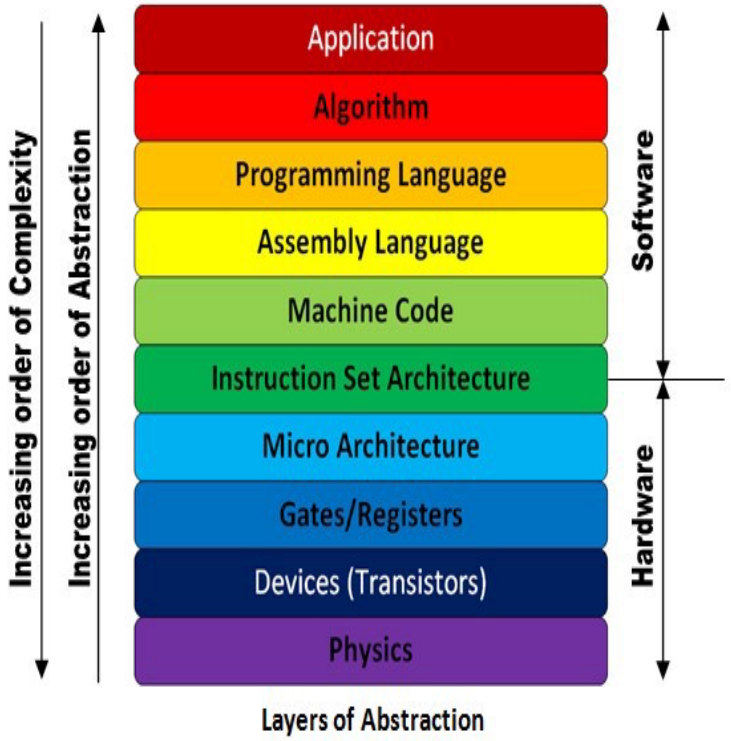
| Guideline | TPU | Catapult | Crest | Pixel Visual Core |
|---|---|---|---|---|
| Design target | Data center ASIC | Data center FPGA | Data center ASIC | PMD ASIC/SOC IP |
| 1. Dedicated memories | 24 MiB Unified Buffer, 4 MiB Accumulators | Varies | N.A. | Per core: 128 KiB line buffer, 64 KiB P.E. memory |
| 2. Larger arithmetic unit | 65,536 Multiply-accumulators | Varies | N.A. | Per core: 256 Multiply-accumulators (512 ALUs) |
| 3. Easy parallelism | Single-threaded, SIMD, in-order | SIMD, MISD | N.A. | MPMD, SIMD, VLIW |
| 4. Smaller data size | 8-Bit, 16-bit integer | 8-Bit, 16-bit integer 32-bit Fl. Pt. | 21-bit Fl. Pt. | 8-bit, 16-bit, 32-bit integer |
| 5. Domain-specific lang. | TensorFlow | Verilog | TensorFlow | Halide/TensorFlow |

**Figure 7.3 The four DSAs in this chapter and how closely they followed the five guidelines.** Pixel Visual Core typically has 2–16 cores. The first implementation of Pixel Visual Core does not support 8-bit arithmetic.

# How to do optimizations for DSA?




- Workload characterization
  - Hotspots
  - Frequently executed code
  - Understanding the algorithm
- HW-SW / Algo-HW Co-design
  - Scheduling and mapping / Control and memory dependence
- Microarchitecture design
  - Processing elements
  - Memory hierarchy
  - Data movement



# Takeaways

- Understand **why we need domain specific architectures (DSAs)**
- Know that there are **guiding principles in DSA design**
- Understand that we **need to have a systematic methodology**

# Readings: Accelerator-Level Parallelism

- I think it would have been helpful for the authors to clarify how the ideas for ALP that they were proposing are currently being implemented, if at all.
- Does discussing accelerators via their use on smartphones actually represent enough of a range of the problems and challenges?
- The paper seems to suggest that all accelerators present in mobile SoCs use a CPU controlled DMA to transfer data. Is this generally true, and if so, is(are) there a particular reason(s) the authors can point to? It seems somewhat crazy to me that this would be the general model of usage for accelerators in this space because of the resulting inefficiencies!
- The paper introduced smartphone chips as the "harbinger of ALP".
- It didn't feel like it contributed a lot of new insights. A lot of the challenges proposed seem to be problems in any parallel system, such as programmability, concurrency, etc.
- It is unclear what the audience for the paper is.
-

# Readings: A New Golden Age: Empowering the Machine Learning Revolution

- “The article itself is not that technical”
- “I think that its greatest strength is how it emphasizes the fallacies and pitfalls that architects should be aware of while designing specialized hardware for machine learning problems.”
- “I found much of the section on the six issues confusing because it used a large amount of jargon in order to shove all six issues into a very small amount of space.”
- “What are ways we can think about creating specific accelerators but bridge them with software in a usable fashion? How can the hardware community deal with the rapid changing of ML research over the span of time to make a chip?”
- “batch size corresponding to operand reuse is not explained in a meaningful way.”
- “What is hard about designing ML benchmarks and what is wrong with the current approaches?”
- “Can we go over the specific technical differences between CPUs and GPUs, and their use in applications like these?”
- “I'm also not sure how comparing the Moore's Law graph with the number of ML arXiv articles is relevant, it feels like comparing apples and oranges, “
- “How are the major chipmakers like Intel, AMD, Nvidia challenging this space?”
- “better compilation has the potential to bridge the gap between flexibility and specialization but that was not really explained in detail.”