FPGA BASED ACCELERATION OF SCIENTIFIC WORKLOADS – WHY? HOW?

Dr. Suleyman Demirsoy

HPC Systems FAE

Intel Programmable Solutions Group

AGENDA

- Trends
- FPGA architecture
- Parallelism
- High level design flows

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

*Other names and brands may be claimed as the property of others.

Industry Trends

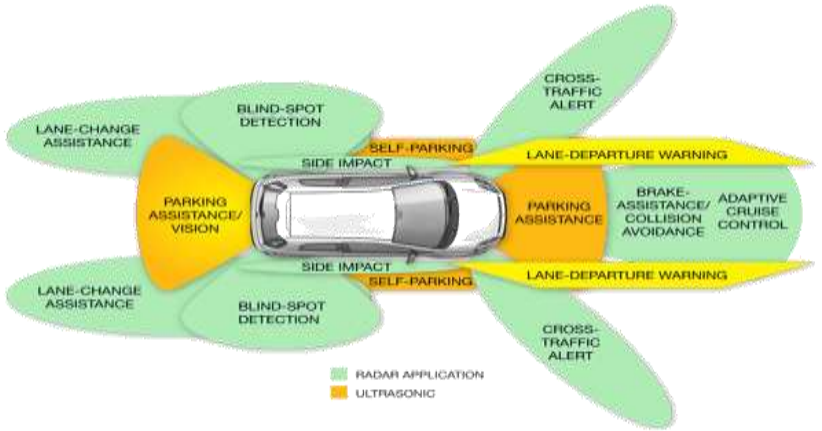
Ever increasing functionality and performance required

Data set sizes and pipelines to move data continue to increase

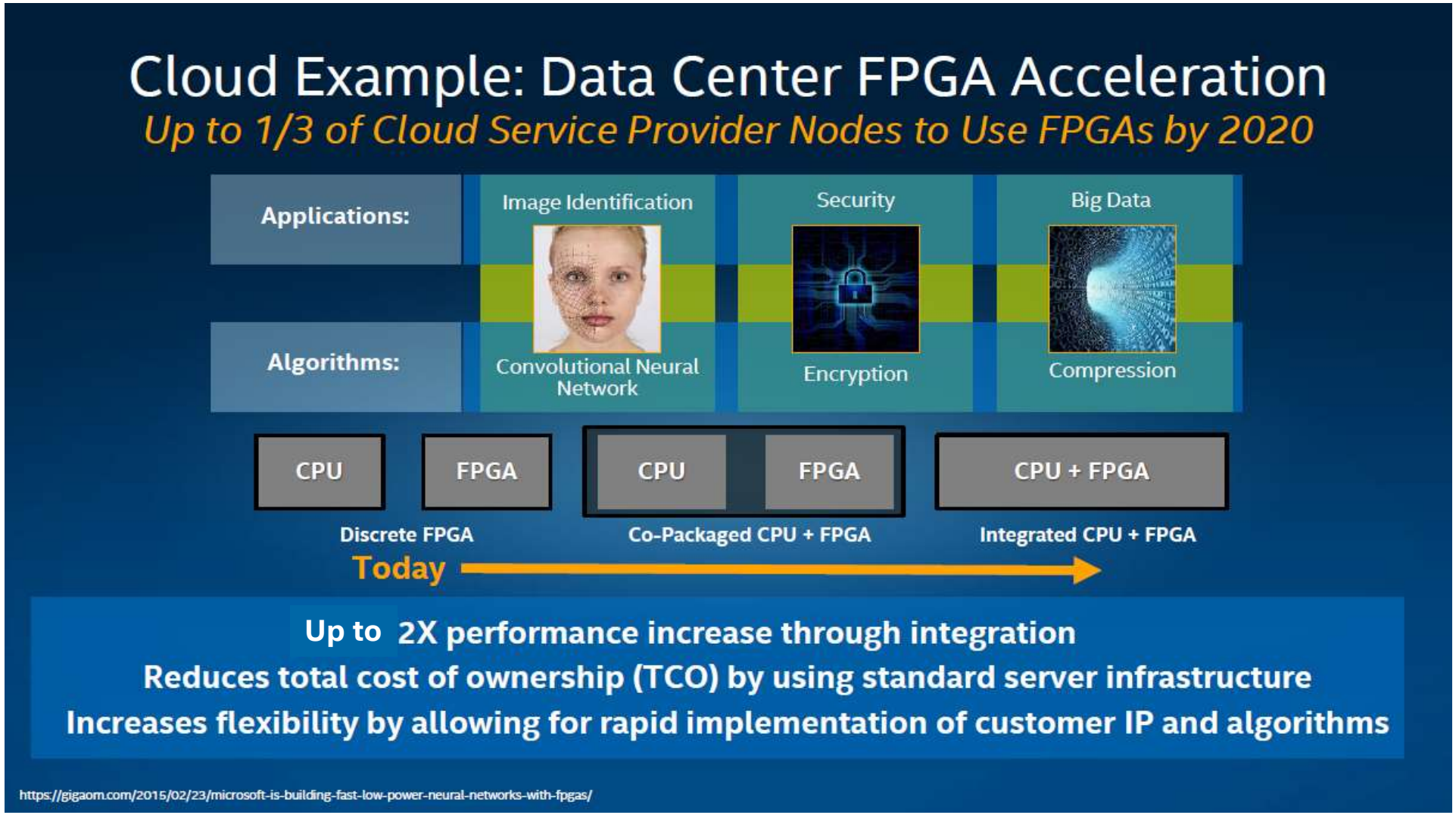
Struggle to sustain performance trajectory without massive increases in cost, power and system size

Time to market pressure always increasing

Size and capabilities of FPGAs are growing exponentially

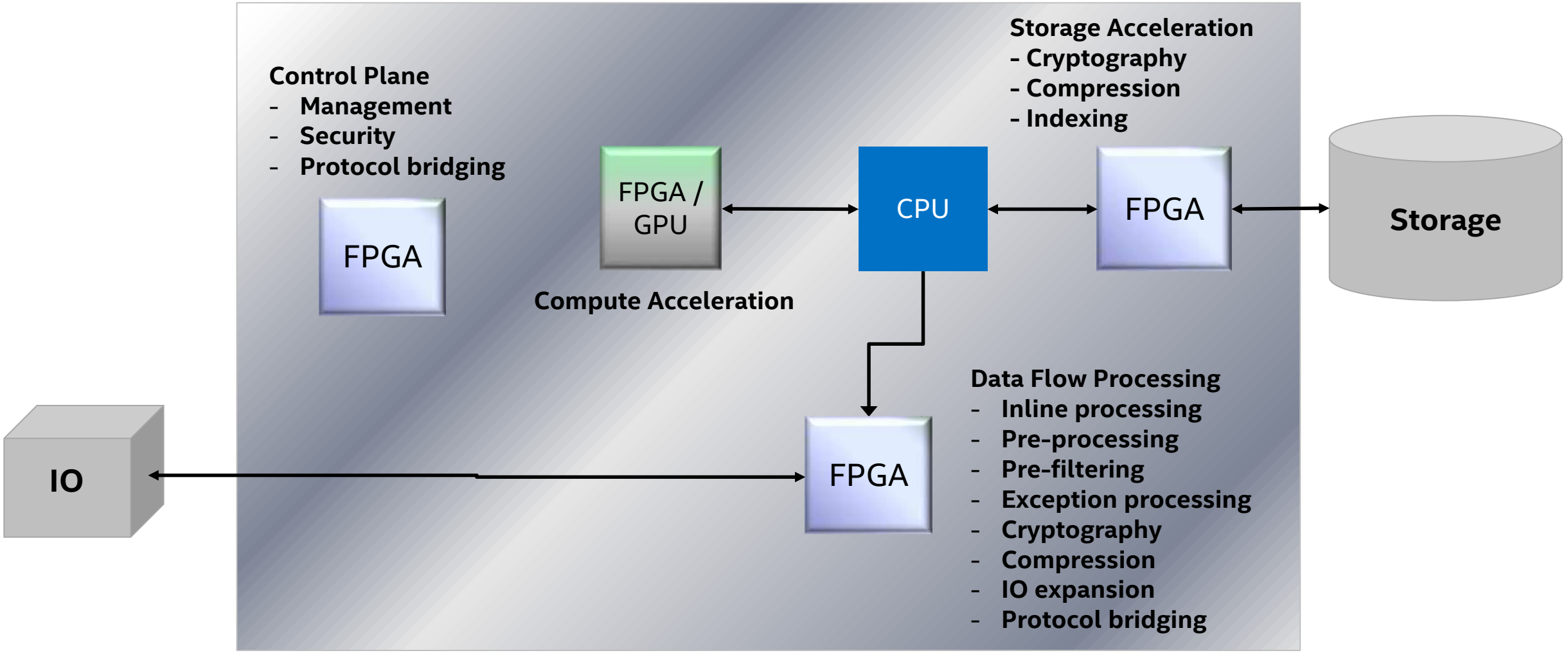


Intel Projecting 1/3 Cloud Nodes are FPGA by 2020



http://tinyurl.com/yd3c5e97

Where Do FPGAs Fit In?



FPGAs enable solving system level data movement issues

FPGA ARCHITECTURE

FPGA Architecture: Fine-grained Massively Parallel

Millions of reconfigurable logic elements

Thousands of 20Kb memory blocks

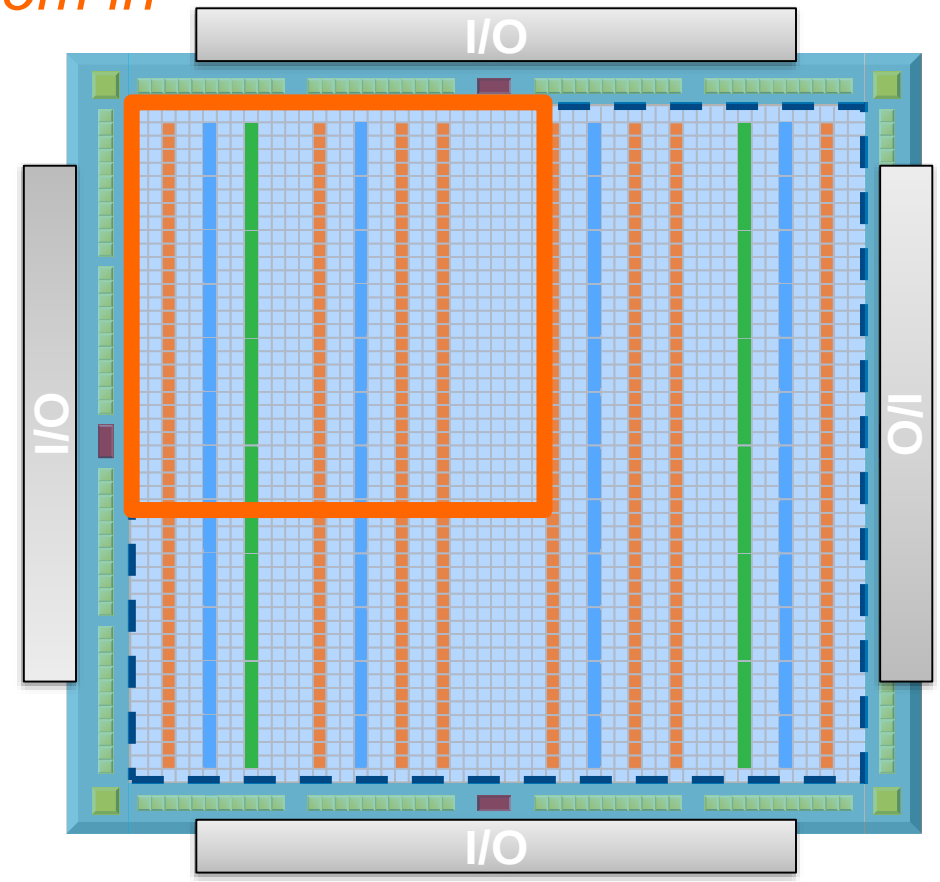
Thousands of Variable Precision DSP blocks

Dozens of High-speed transceivers

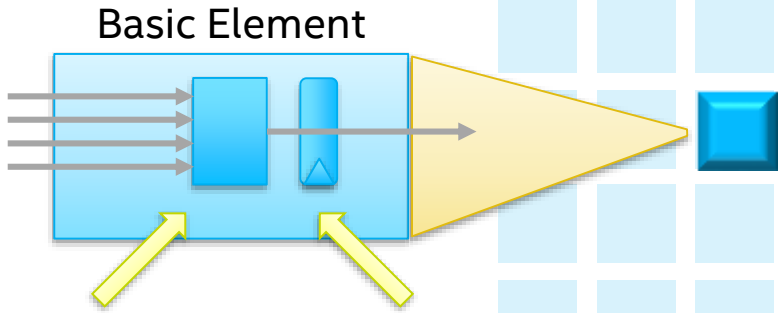
Multiple High Speed configurable Memory Controllers

Multiple ARM© Cores

Let's zoom in



FPGA Architecture: Basic Elements

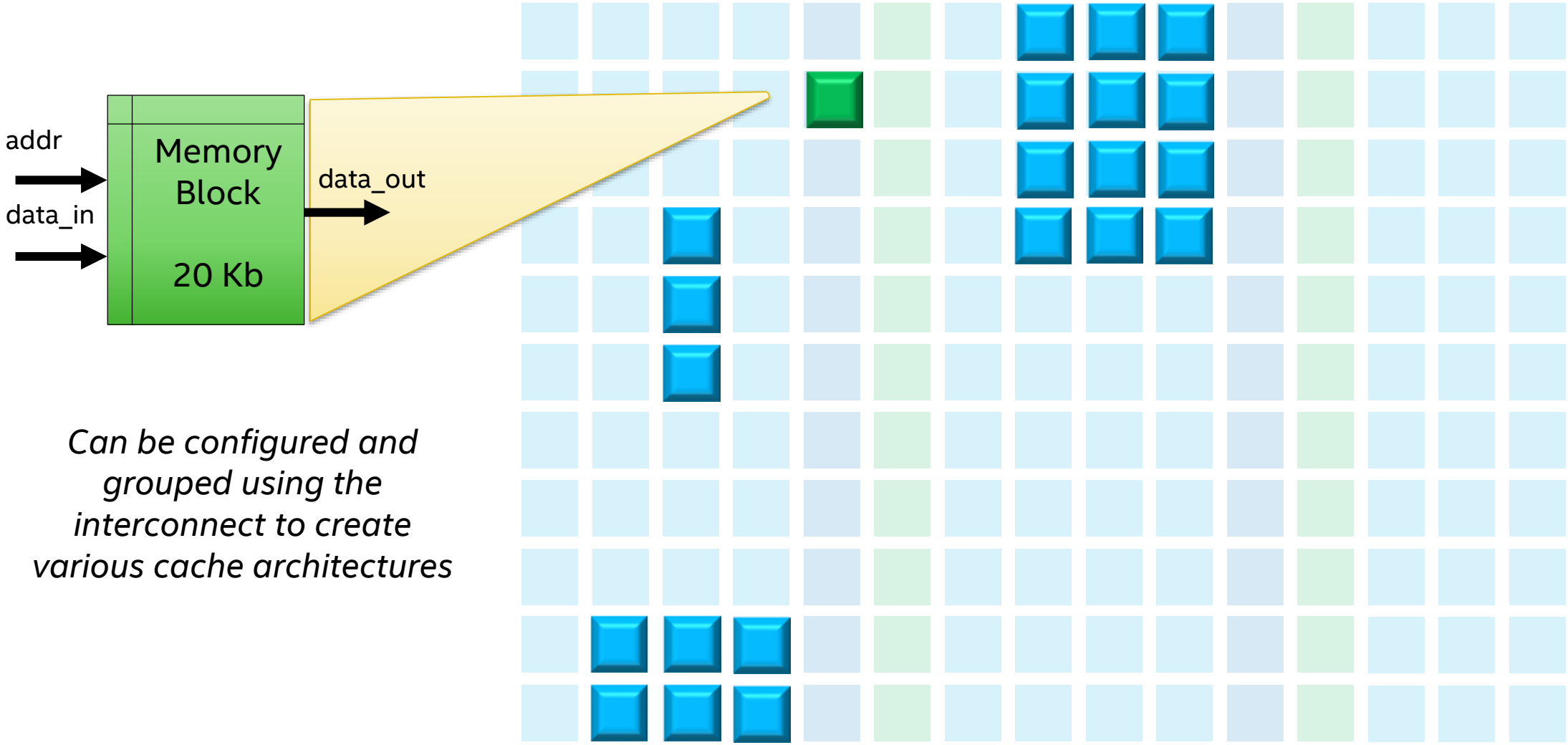


1-bit configurable operation

1-bit register (store result)

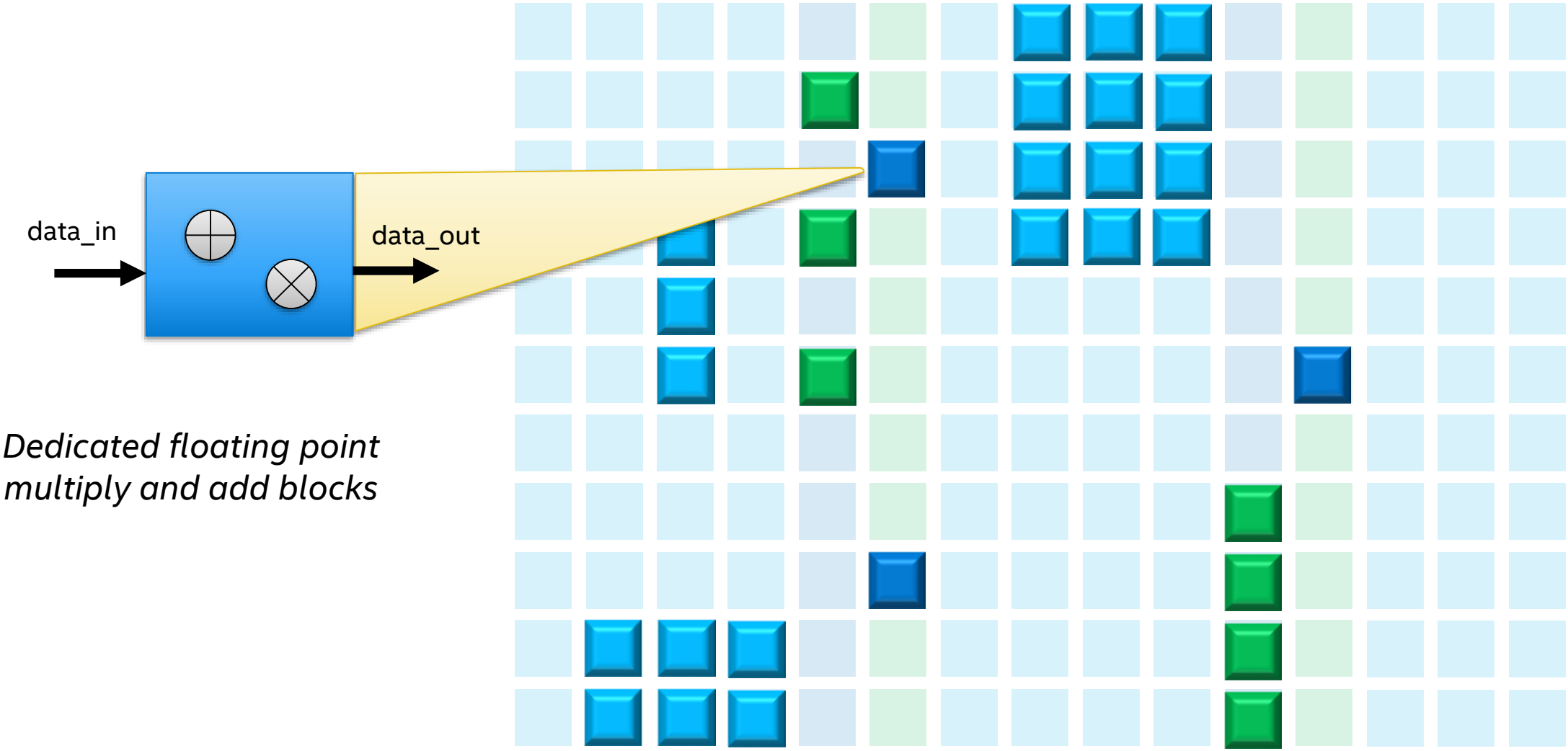
Configured *to perform any 1-bit operation: AND, OR, NOT, ADD, SUB*

FPGA Architecture: Memory Blocks



Can be configured and grouped using the interconnect to create various cache architectures

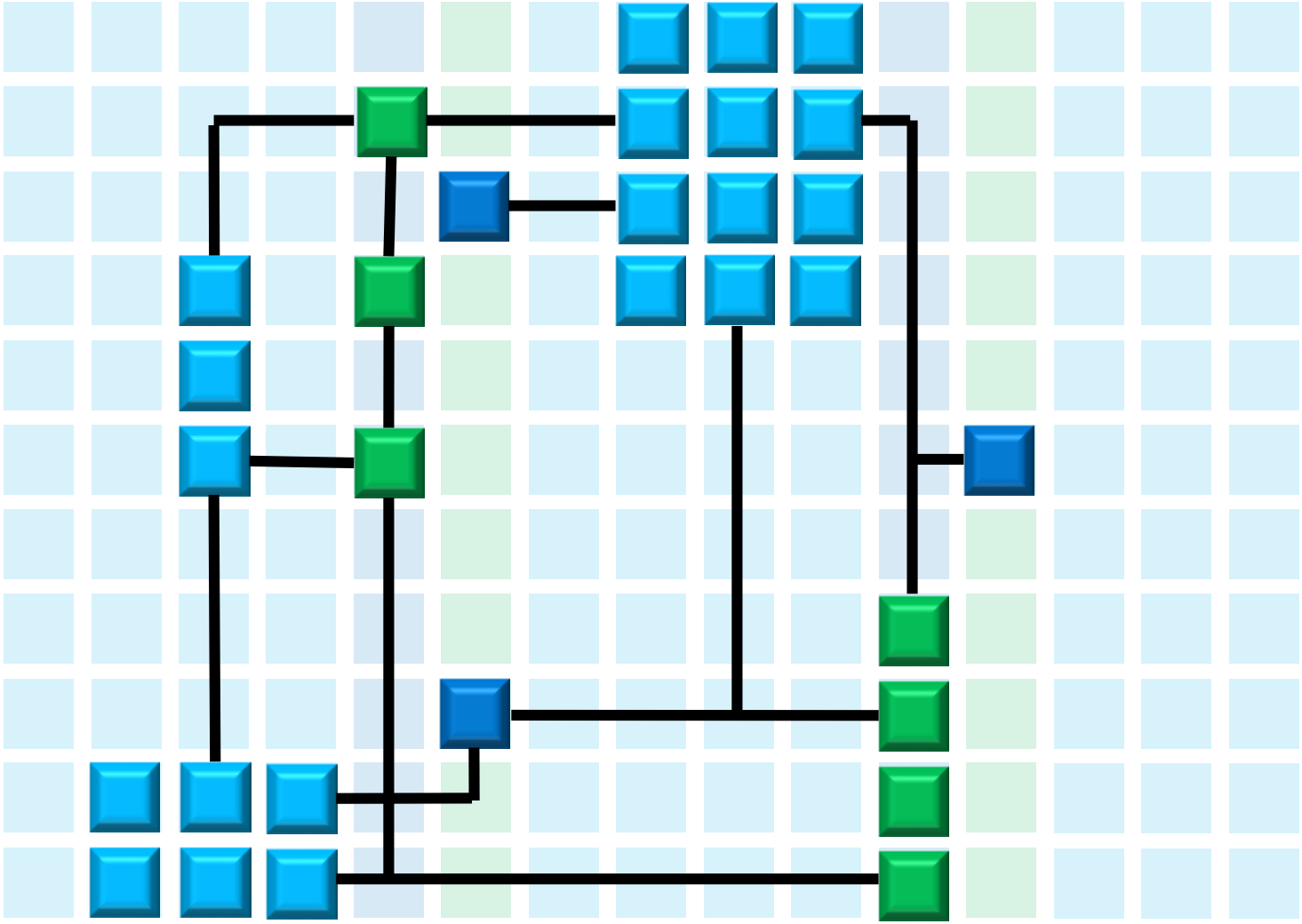
FPGA Architecture: Floating Point Mult/Add



Dedicated floating point multiply and add blocks

FPGA Architecture: Configurable Routing

*Blocks are connected into a **custom data-path** that matches your application.*



Incredible numbers

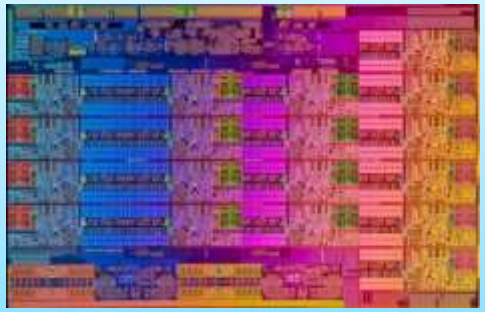
	Product Line	Arria 10 GX FPGAs[1]						
		GX 270	GX 320	GX 480	GX 570	GX 660	GX 900	GX 1150
Resources	Part number reference	0AX027	10AX032	10AX048	10AX057	10AX066	10AX090	10AX115
	LEs (K)	270	320	480	570	660	900	1,150
	Adaptive logic modules (ALMs)	01,620	118,730	181,790	217,080	250,540	339,620	427,700
	Registers	406,480	474,920	727,160	868,320	1,002,160	1,358,480	1,708,800
	M20K memory blocks	750	891	1,438	1,800	2,133	2,423	2,713
	M20K memory (Mb)	15	17	28	35	42	47	53
	MLAB memory (Mb)	2.4	2.8	4.3	5.0	5.7	9.2	12.7
	Hardened single-precision floating-point multipliers/adders	830/830	985/985	1,368/1,368	1,523/1,523	1,688/1,688	1,518/1,518	1,518/1,518
	18 x 19 multipliers	1,660	1,970	2,736	3,046	3,376	3,036	3,036
	Peak GMACS	1,826	2,167	3,010	3,351	3,714	3,340	3,340
	GFLOPS	747	887	1,231	1,371	1,519	1,366	1,366
Clocks, Maximum I/O Pins, and Architectural Features	Global clock networks	32	32	32	32	32	32	32
	Regional clocks	8	8	8	8	16	16	16
	I/O voltage levels supported (V)	1.2, 1.25, 1.35, 1.8, 2.5, 3.0[2]						
	I/O standards supported	**3 V I/O pins only:** 3 V LVTTL, 2.5 V CMOS **DDR and LVDS I/O pins:** POD12, POD10, Differential POD12, Differential POD10, LVDS, RSDS, mini-LVDS, LVPECL L-125, SSTL-18 (I and II), SSTL-15 (I and II), Differential SSTL-12, SSTL-12, HSTL-18 (I and II), HSTL-15 (I and II), HSTL-12 (I and II), HSUL-12, Differential SSTL-135, Differential SSTL-125, Differential SST Differential HSTL-18 (I and II), Differential HSTL-15 (I and II), Differential HSTL-12 (I and II), Differential HSUL-12						
	Maximum LVDS channels (1.6 G)	168	168	222	270	270	384	384
	Maximum user I/O pins	384	384	492	624	624	768	768
	Transceiver count (17.4 Gbps)	24	24	36	48	48	96	96
	Transceiver count (28.3 Gbps)	–	–	–	–	–	–	–
	PCIe hard IP blocks (Gen3)	2	2	2	2	2	4	4
	Maximum 3 V I/O pins	48	48	48	48	48	–	–
	Memory devices supported	DDR4, DDR3, DDR2, QDR IV, QDR II+, QDR II+ Xtreme, LPDDR3, LPDDR2, RLDRAM 3, RLDRAM II, LLDRAM II, HMC						

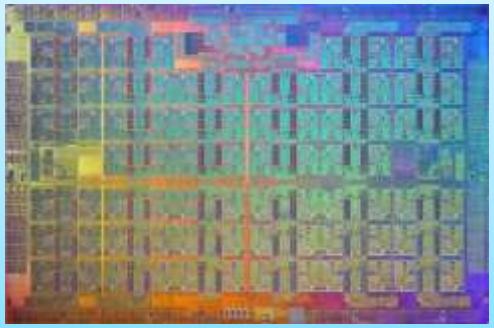
VARIOUS LEVEL OF PARALLELISM

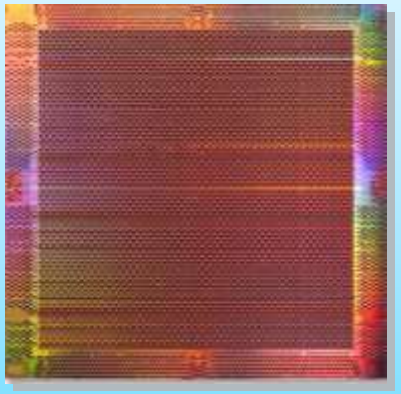
Performance in the Data Center

- Towards more a parallelism through spatial computing

Course Grain Parallelism → ***Fine Grain Parallelism***







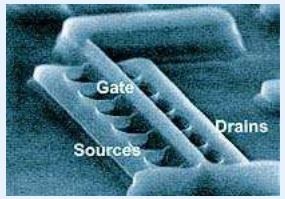
Intel® Xeon® processor E7 v4 product family: up to 24 Cores

Intel® Xeon® Phi Processor Family: up to 72 Cores

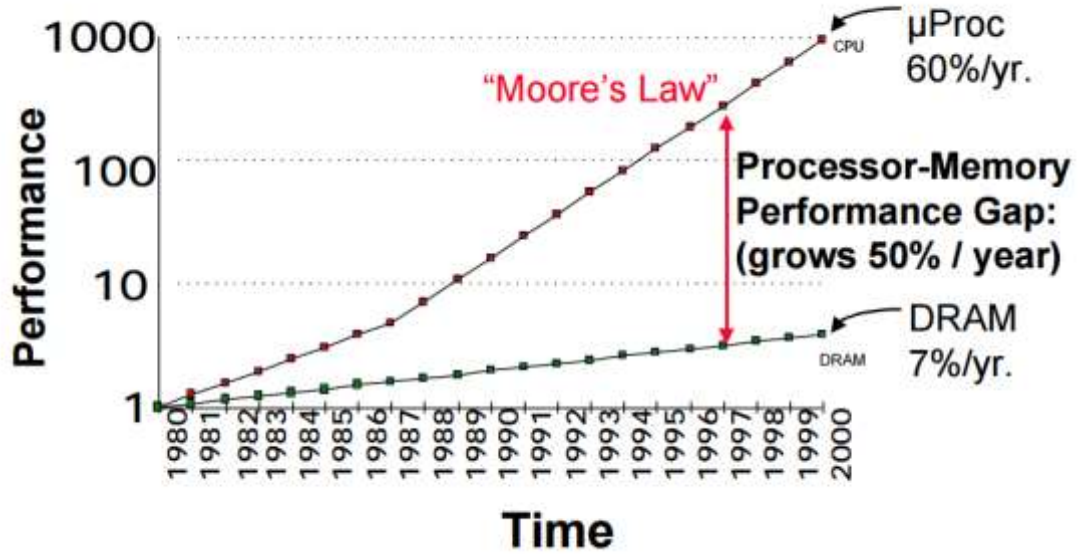
Intel® Arria 10: up to 1150K equivalent logic elements

https://mediastream.microsoft.com/events/2016/1609/Ignite/player/keynote-pm.html

What problem are we solving

Information	Memory Wall
	Memory architectures have limited bandwidth, and can't keep up with the processor
Computation	ILP Wall
	Compilers don't find enough parallelism in a single instruction stream to keep Von Neuman-based architectures busy
Realization	Power Wall
	Process scaling trends towards exponentially increasing power consumption

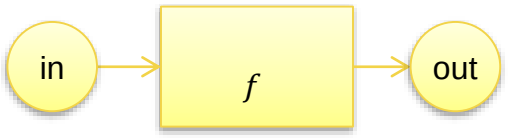
Memory wall



https://people.eecs.berkeley.edu/~pattrsn/talks/Cadence.pdf

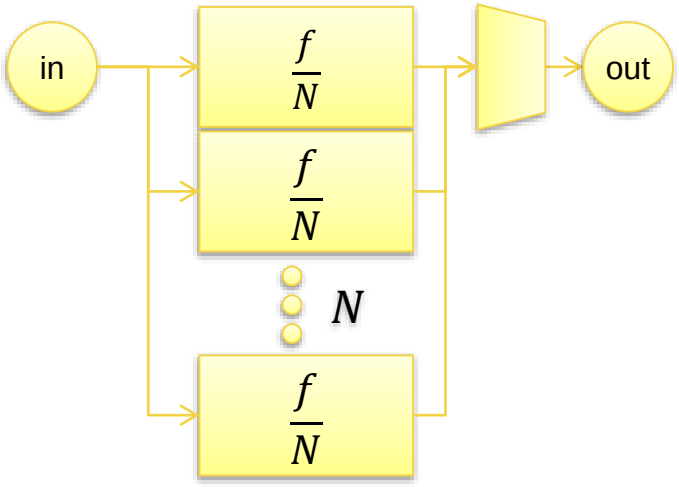
L1, L2, L3	Growing cache sizes to manage latency
GDDR	Higher bus size for bandwidth and power
QDR	Double clock efficient interleaving of read/write
HMC, HMB	Multi-layer cross-switch and memory control for bandwidth

Power wall



$$P = CV^2f$$

Lower V	Parallelism
Lower Vt	Tri-Gate



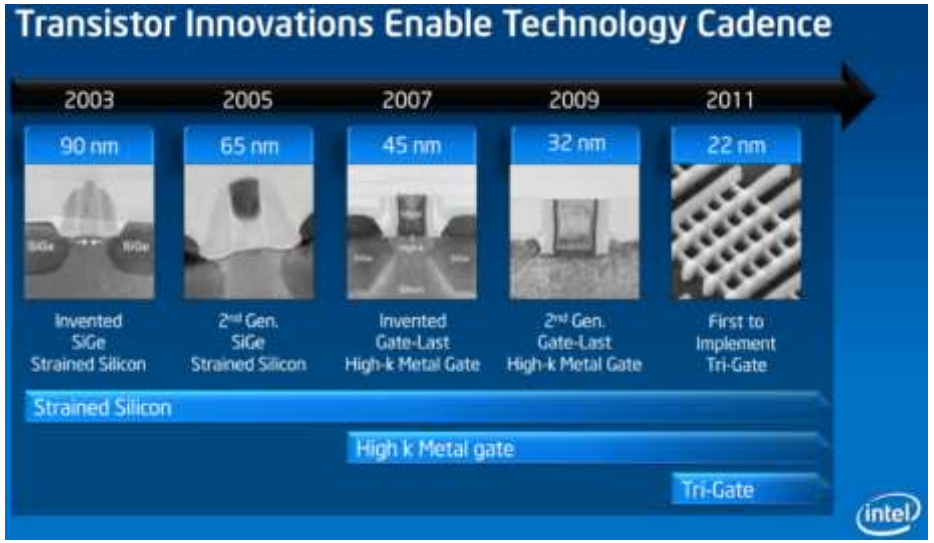
$$P = N * (\gamma C)(\upsilon V)^2 \left(\frac{f}{N}\right)$$

$$P = (\gamma \upsilon^2) C V^2 f$$

$$\gamma \approx 1.2$$

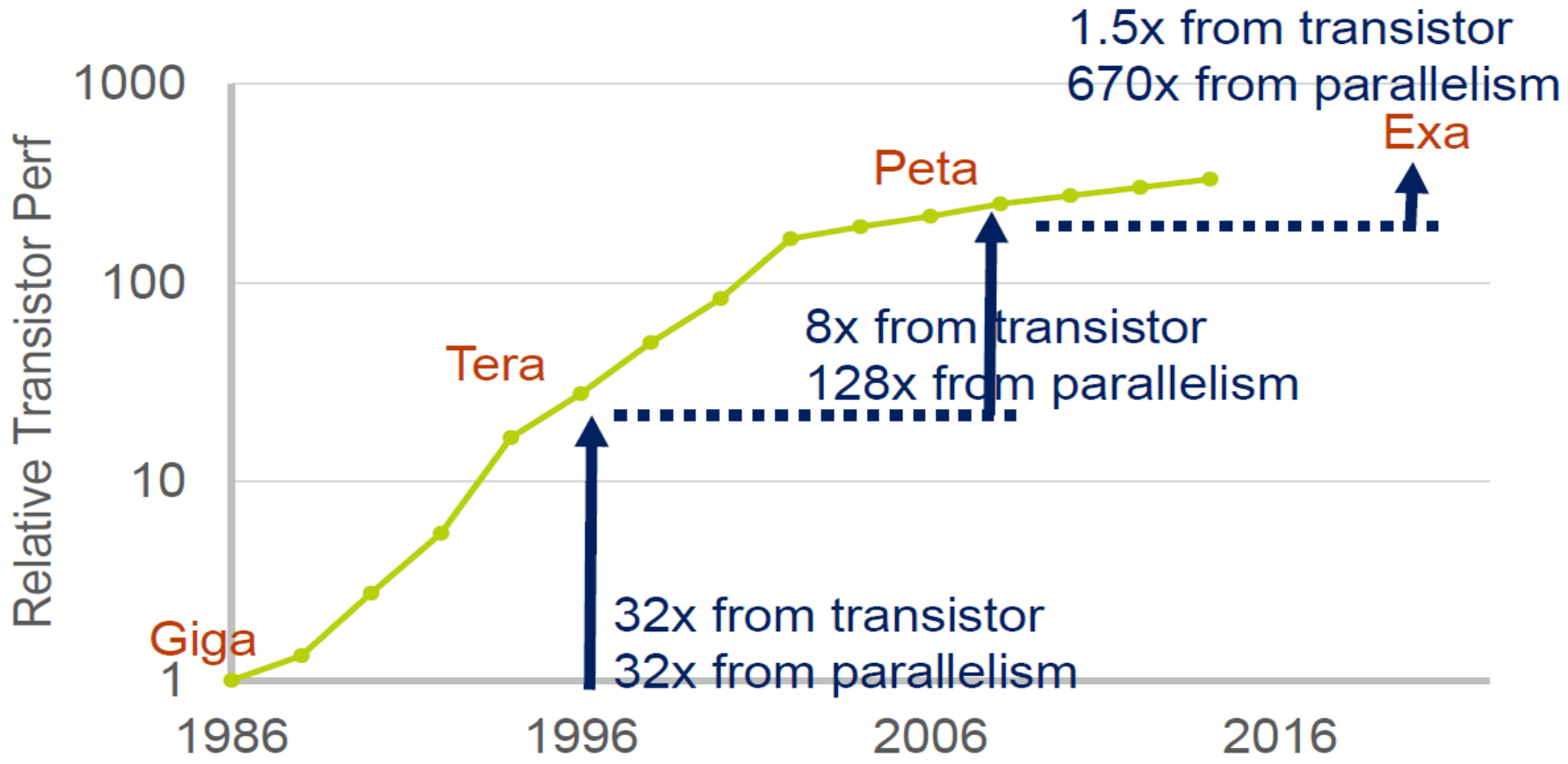
$$\upsilon \approx 0.7$$

$$\boldsymbol{P \approx 0.5CV^2f}$$



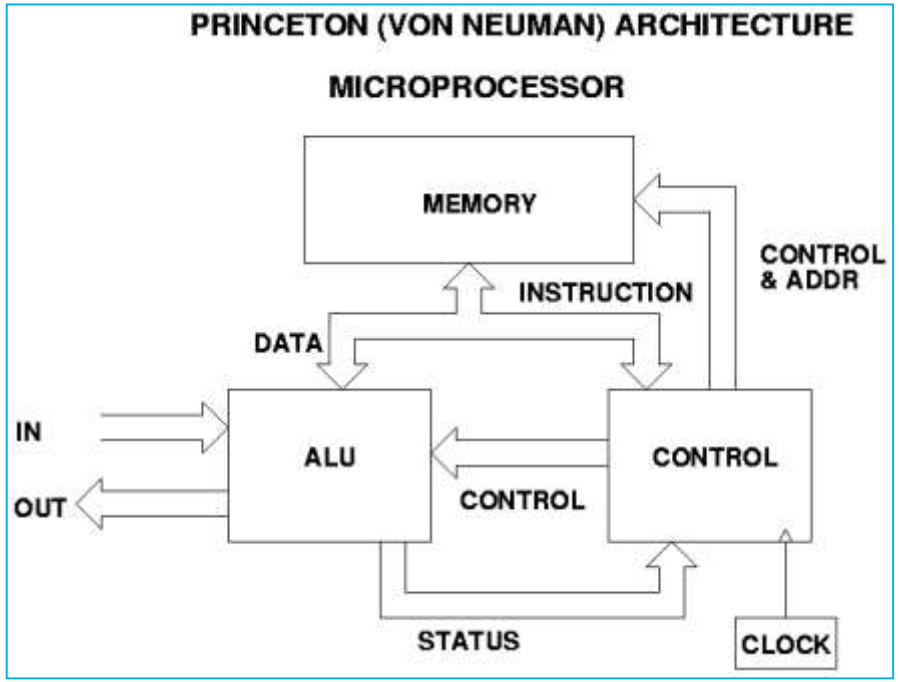
http://www.intel.com/content/dam/www/public/us/en/documents/backgrounders/standards-22nm-3d-tri-gate-transistors-presentation.pdf

Implications to HPC Roadmap

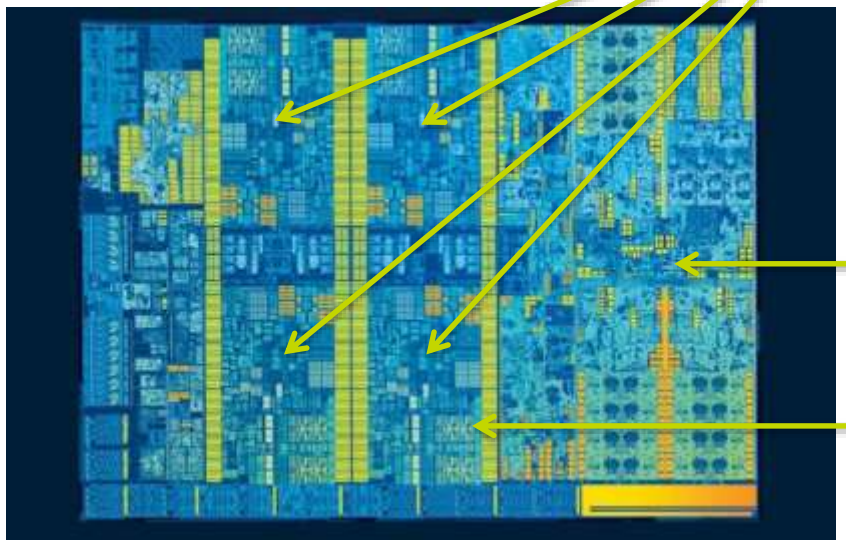


https://www.hpcwire.com/2016/06/14/us-carves-path-capable-exascale-computing/

Compute wall



SIMD	Vectorization
	Data Parallelism
MIMD	Task Parallelism



http://www.ee.nmt.edu/~rison/ee308_spr00/supp/000119/princeton.gif

http://images.anandtech.com/doci/9582/SkylakeFalseColor_678x452.jpg

Parallel Computing

"A form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (in parallel)"

~ Highly Parallel Computing, Amasi/Gottlieb (1989)

Task Parallelism

Multi-Threading (MT)

```
int main() {
 task1(x);
 task2(x);
 task3(x);
}
```

Data Parallelism

Single Instruction Multiple Data (SIMD)

```
int main() {
 for (int i=0;i<N;i++) {
  task(x[i]);
 }
}
```

Challenges in Parallel Programming

Finding Parallelism

- What activities can be executed concurrently?
  - Is parallelism explicit (programmer specified) or implicit?

Data sharing and synchronization

- What happens if two activities access the same data at the same time?
  - Hardware design implications
    - eg. Uniform address spaces, cache coherency
- Is MPI the right solution?

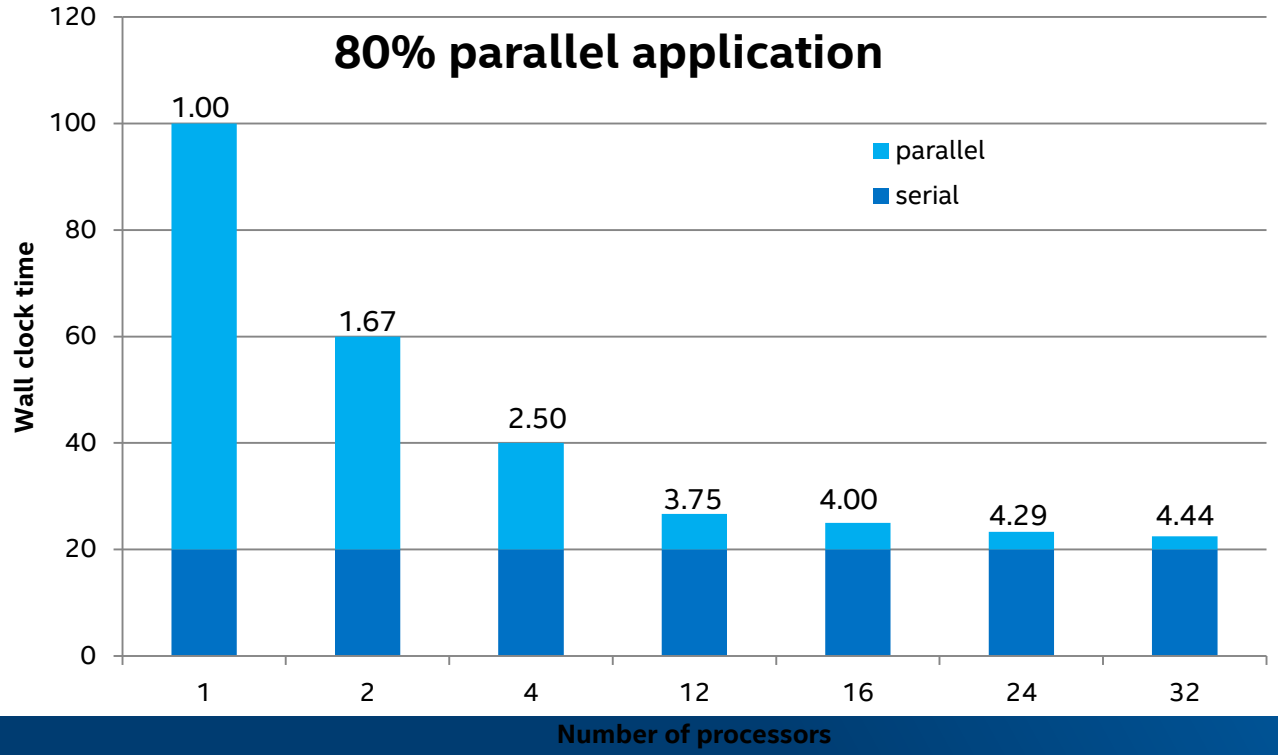
Applications exhibit different behaviors

- Control
  - Searching, parsing, etc…
- Data intensive
  - Image processing, data mining, etc…
- Compute intensive
  - Iterative methods, financial modeling, etc

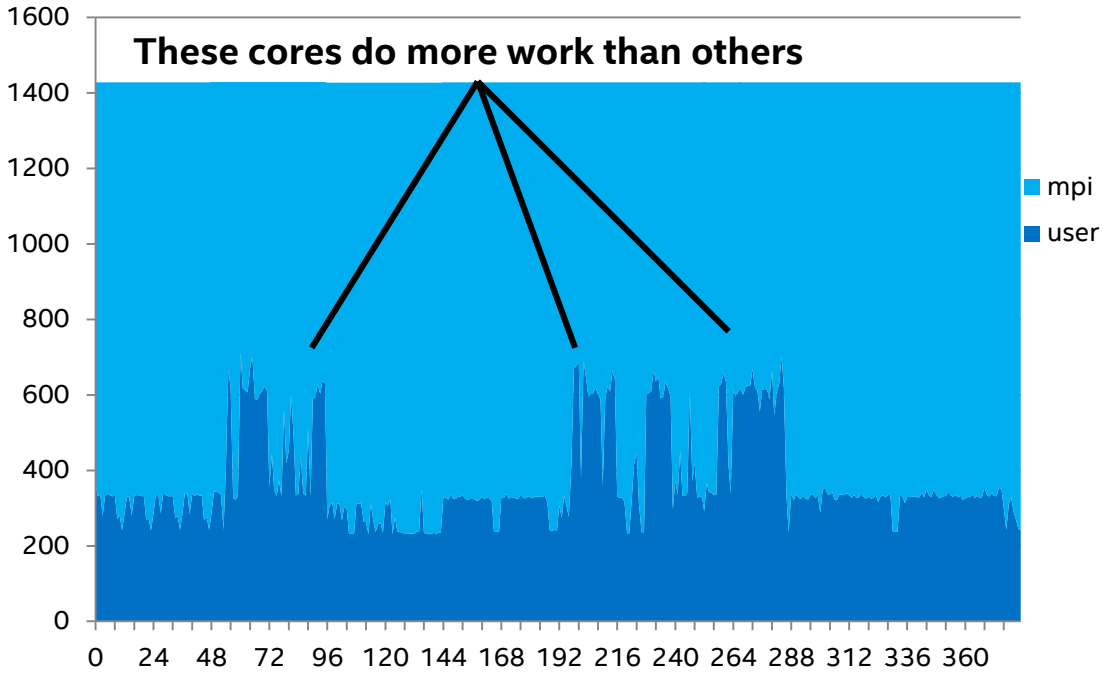
Amdahl's Law Limitations

Not all applications scale linearly

- Speed-up by multiple processors is limited by the time needed for the portion that can not be parallelized



Not all applications load balance across all CPU cores equally



Mapping a Simple Arithmetic expression

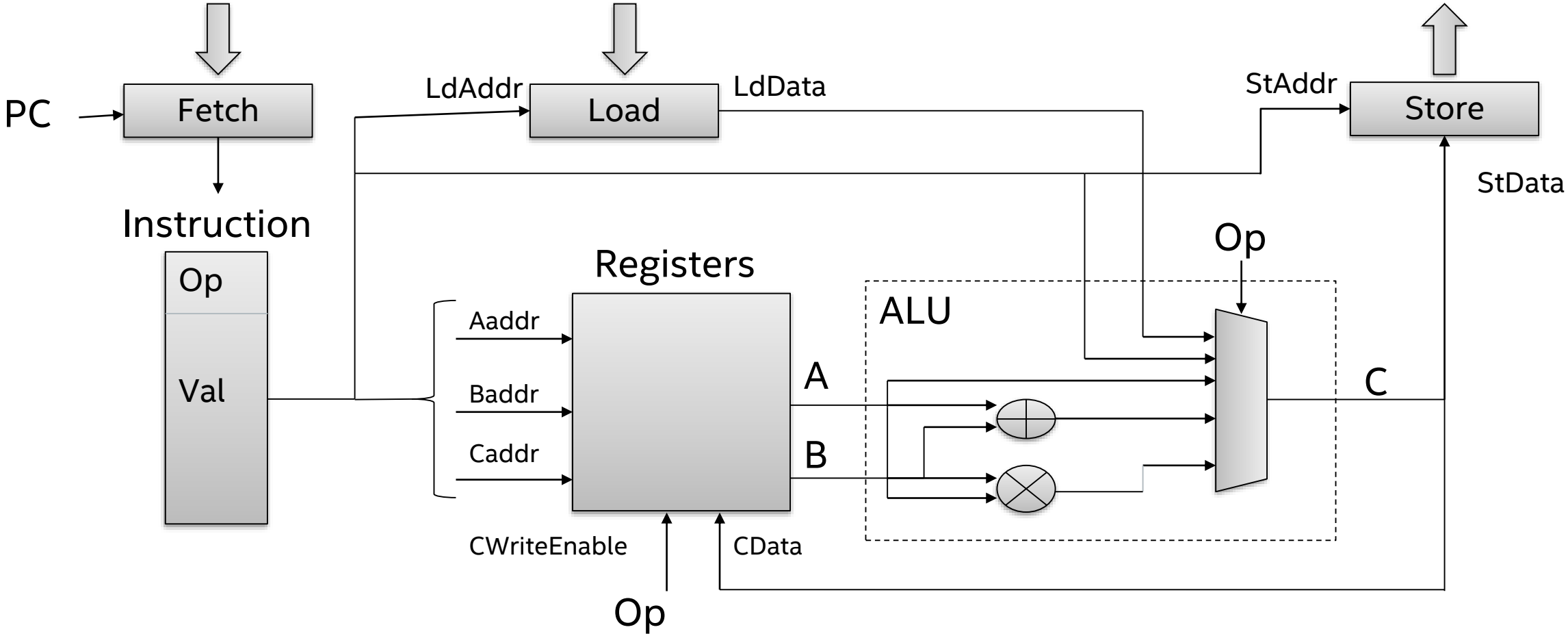
C/C++ instruction

```
Mem[100] += 42 * Mem[101]
```

CPU instructions

```
R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
```

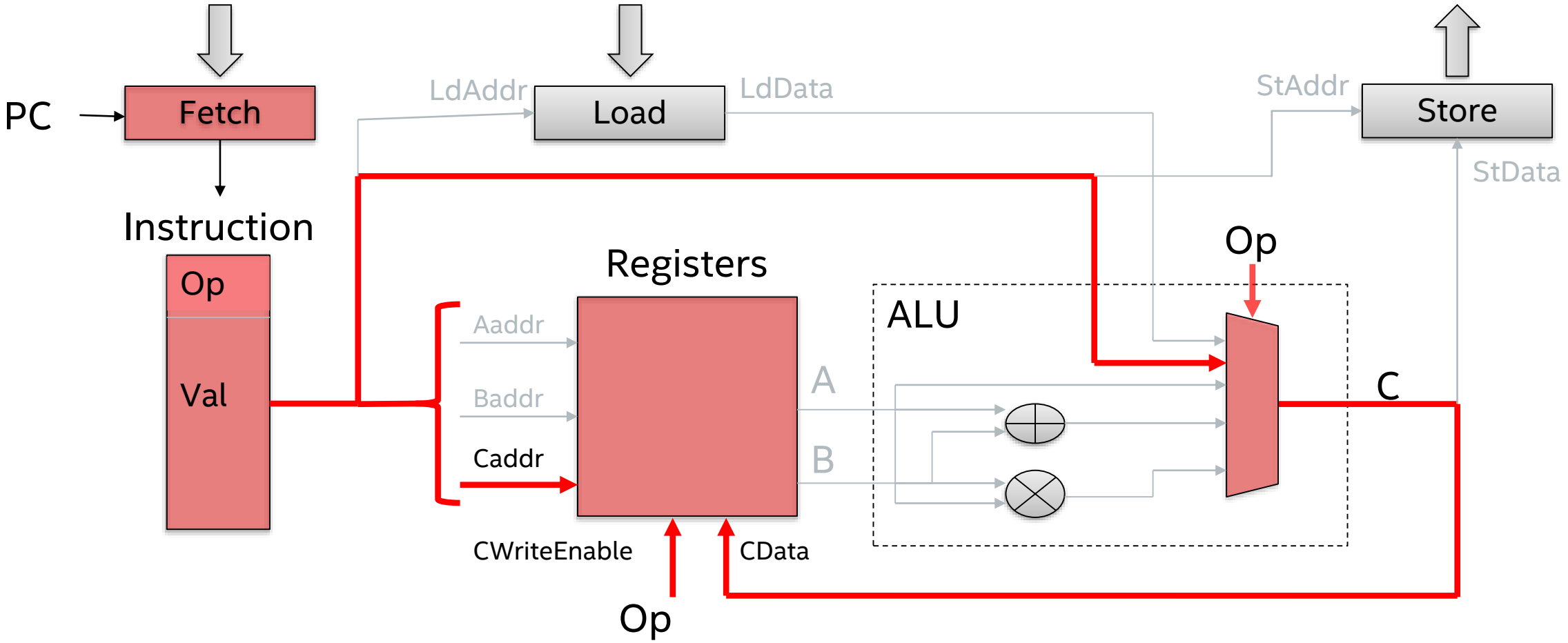
First let's take a look at execution on a *simple* CPU



Fixed and general architecture:

- **General "cover-all-cases" data-paths**
- **Fixed data-widths**
- **Fixed operations**

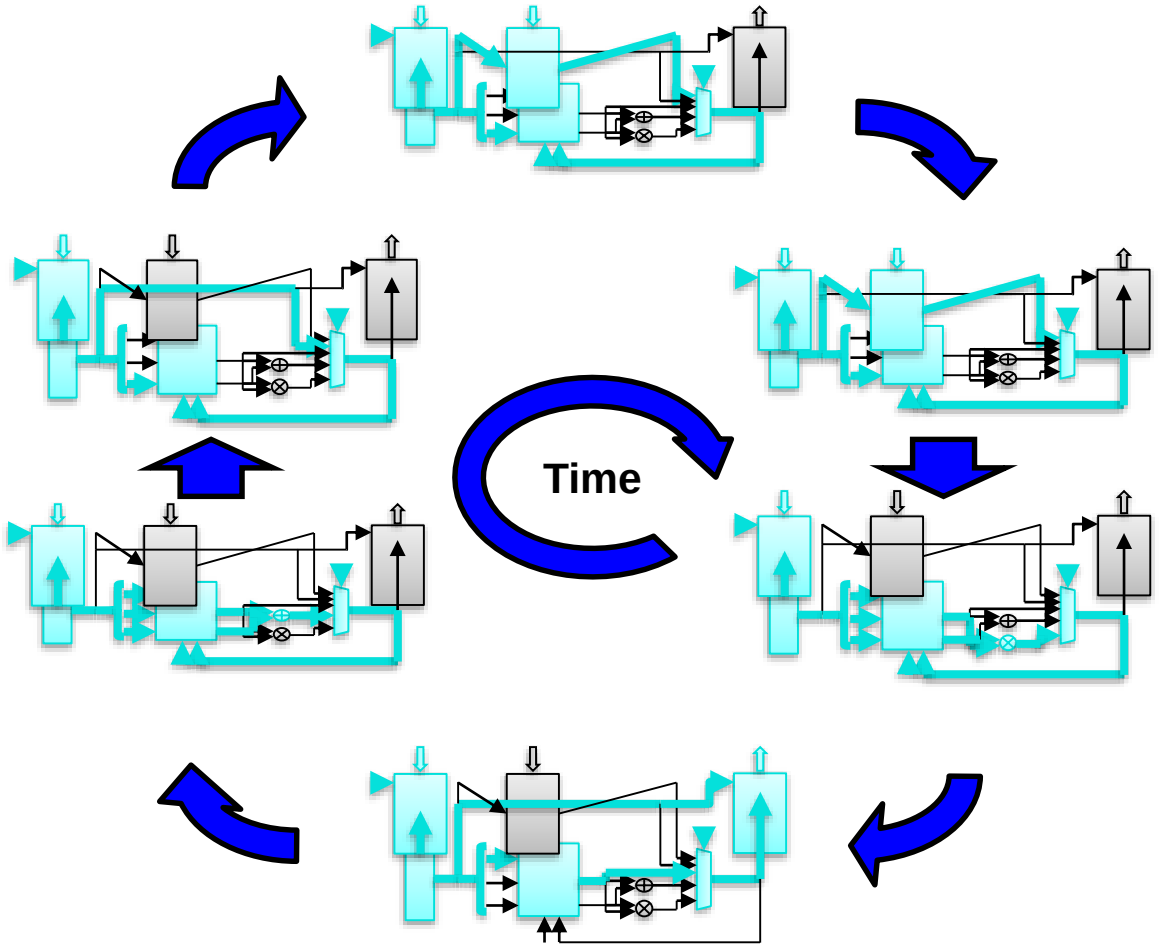
Looking at a Single Instruction



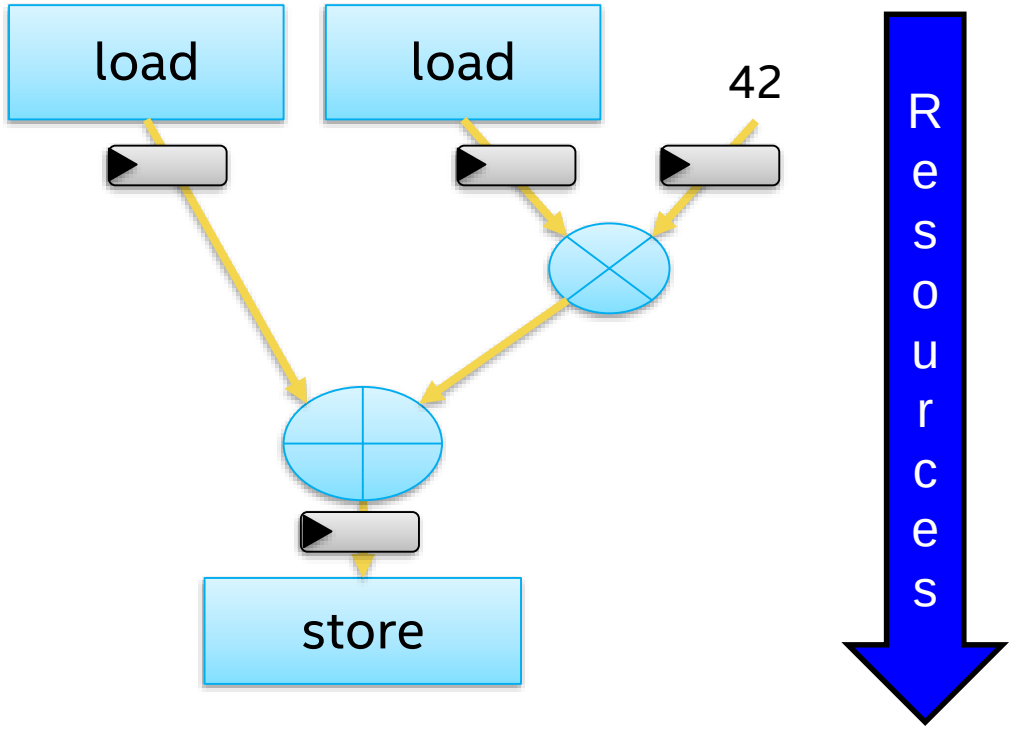
Very inefficient use of hardware!

Sequential Architecture vs. Dataflow Architecture

Sequential CPU Architecture



FPGA Dataflow Architecture

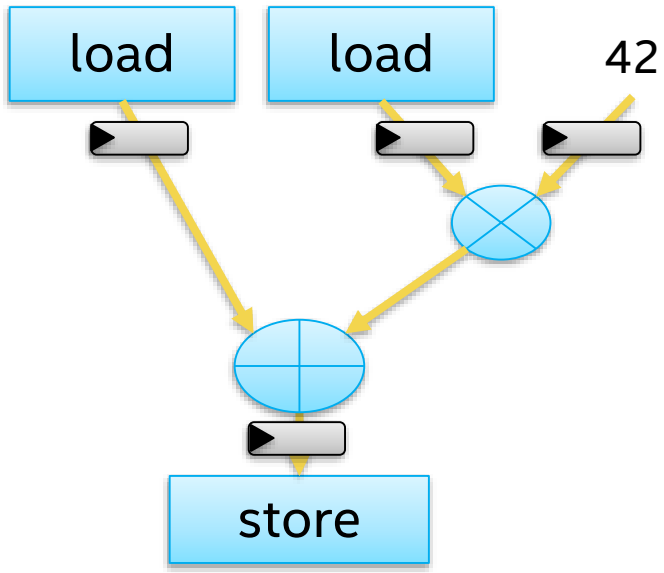


Custom Data-Path on the FPGA Matches Your Algorithm!

High-level code

```
Mem[100] += 42 * Mem[101]
```

Custom data-path



Build exactly what you need:

Operations

Data widths

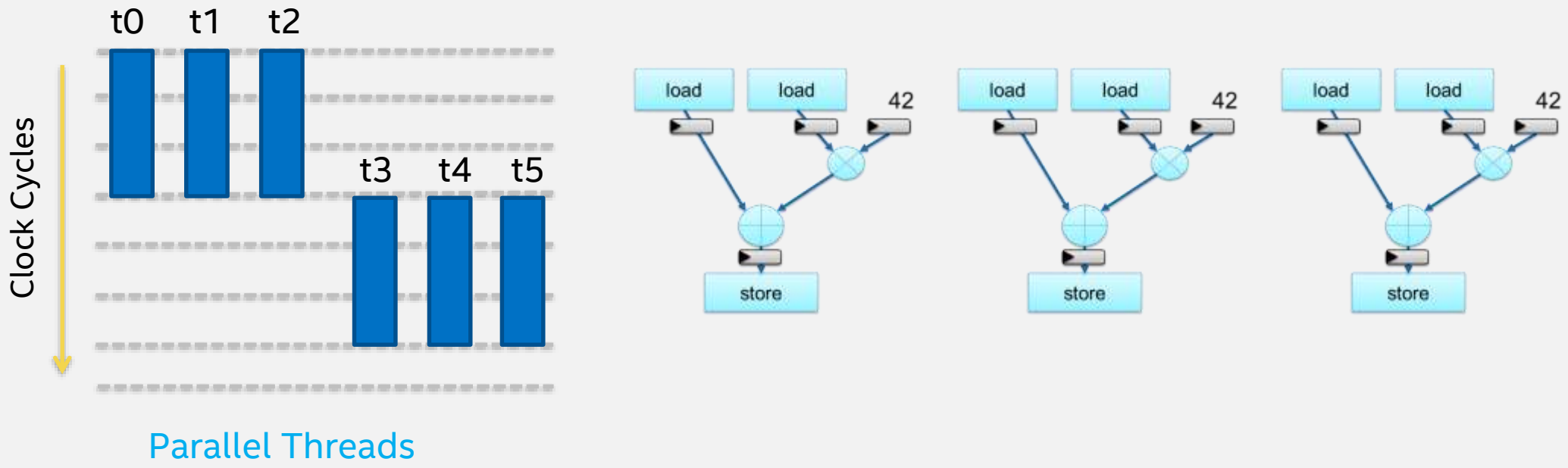
Memory size & configuration

Efficiency:

Throughput / Latency / Power

Execution of Threads on FPGA – Naïve Approach

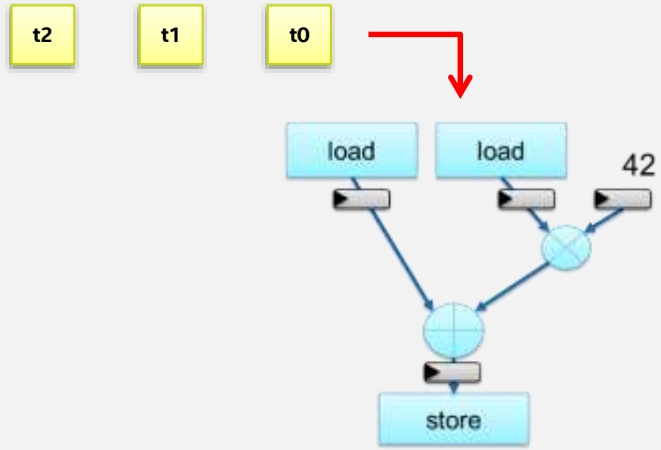
- Thread execution can be executed on *replicated* pipelines in the FPGA
  - ***Throughput = 1 thread per cycle***
  - Area inefficient



Execution of Threads on FPGA

- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined implementation of kernel
  - On each clock cycle, we attempt to send in new thread

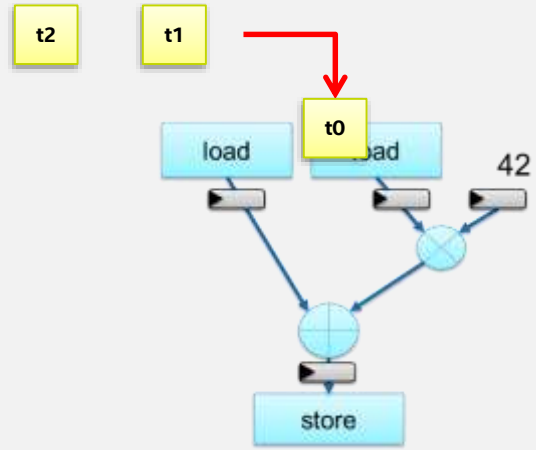
```
kernel void
add( global int* Mem ) {
   ...
   Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined implementation of kernel
  - On each clock cycle, we attempt to send in new thread

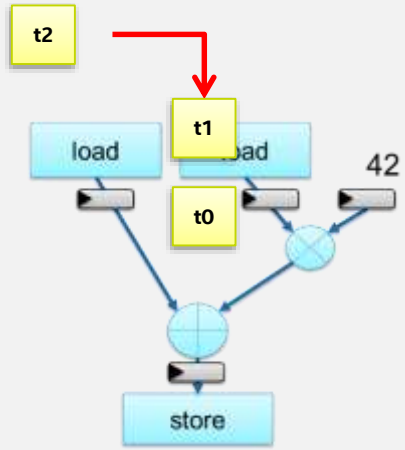
```
kernel void
add( global int* Mem ) {
   ...
   Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined implementation of kernel
  - On each clock cycle, we attempt to send in new thread

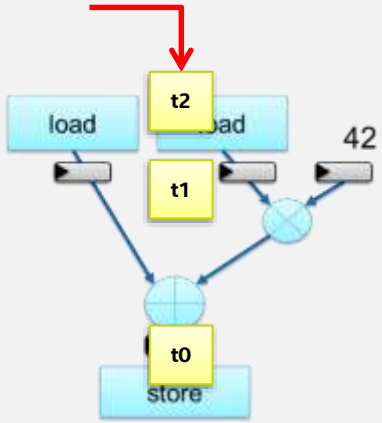
```
kernel void
add( global int* Mem ) {
   ...
   Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined implementation of kernel
  - On each clock cycle, we attempt to send in new thread

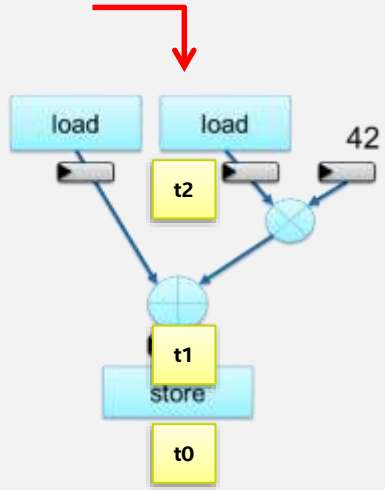
```
kernel void
add( global int* Mem ) {
   ...
   Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined implementation of kernel
  - On each clock cycle, we attempt to send in new thread

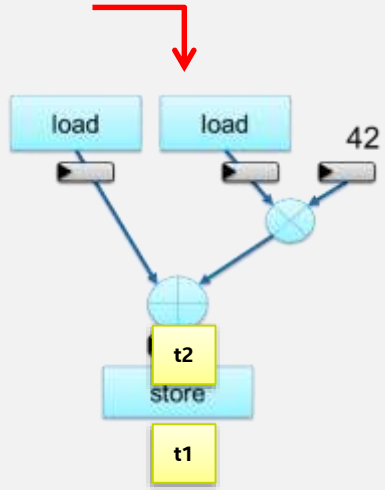
```
kernel void
add( global int* Mem ) {
   ...
   Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

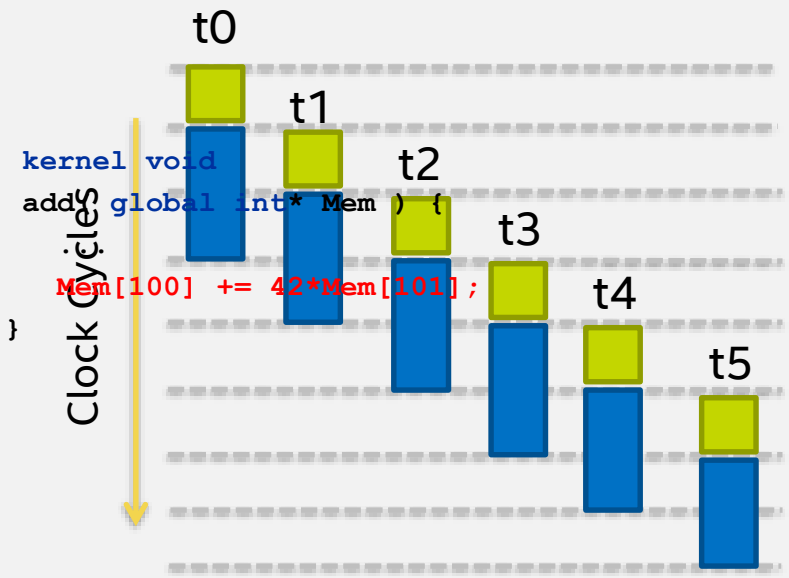
- Better method involves taking advantage of *pipeline parallelism*
  - Attempt to create a deeply pipelined implementation of kernel
  - On each clock cycle, we attempt to send in new thread

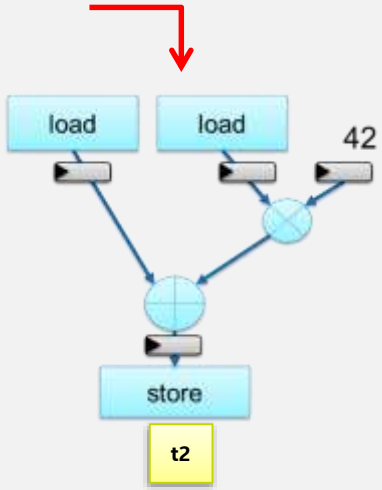
```
kernel void
add( global int* Mem ) {
   ...
   Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

- Better method involves taking advantage of *pipeline parallelism*
  - ***Throughput = 1 thread per cycle***





HIGH LEVEL TOOL FLOWS

High Level Design is the Bridge Between HW & SW

100x More Software Engineers than Hardware Engineers

- Companies don't want to acquire hardware engineers to use FPGAs
  - FPGA developers are a niche skillset and limited supply
- A more accessible abstraction of hardware
- Key to wide-spread adoption of FPGA in Datacenter
- Debugging software is much faster than hardware
- Many functions are easier to specify in software than RTL

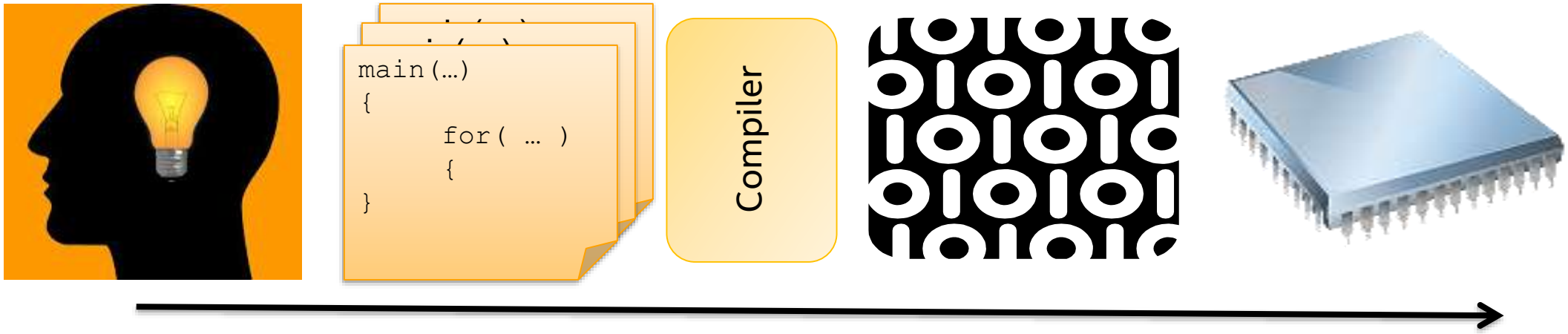
Simulation of RTL takes thousands times longer than software

Design Exploration is much easier and faster in software

We Need to Raise the Level of Abstraction

- Similar to what assembly programmers did with C over 30 years ago
  - (Today) Abstract away FPGA Design with Higher Level Languages
  - (Today) Abstract away FPGA Hardware behind Platforms
  - (Tomorrow) Leverage Pre-Compiled Libraries as Software Services

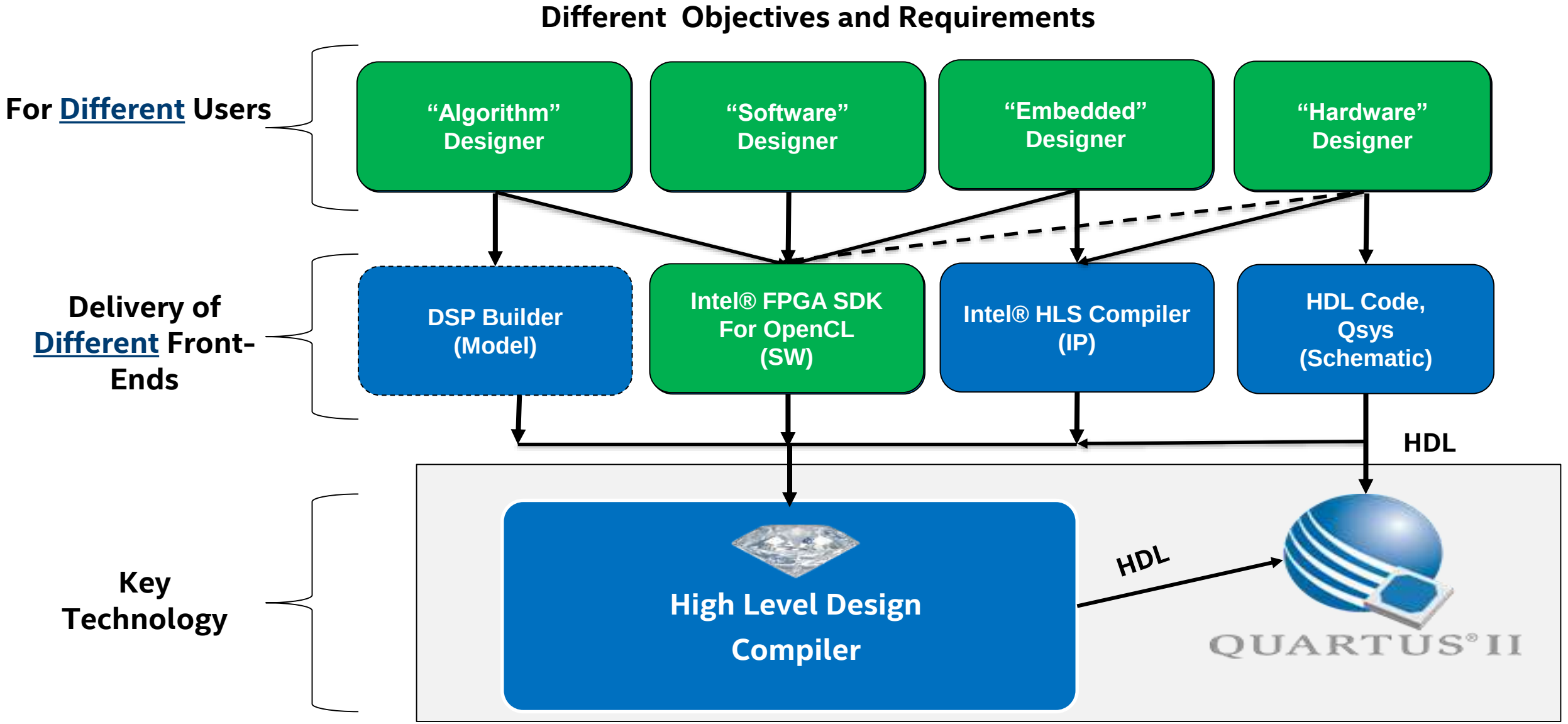
The Software Programmer's View



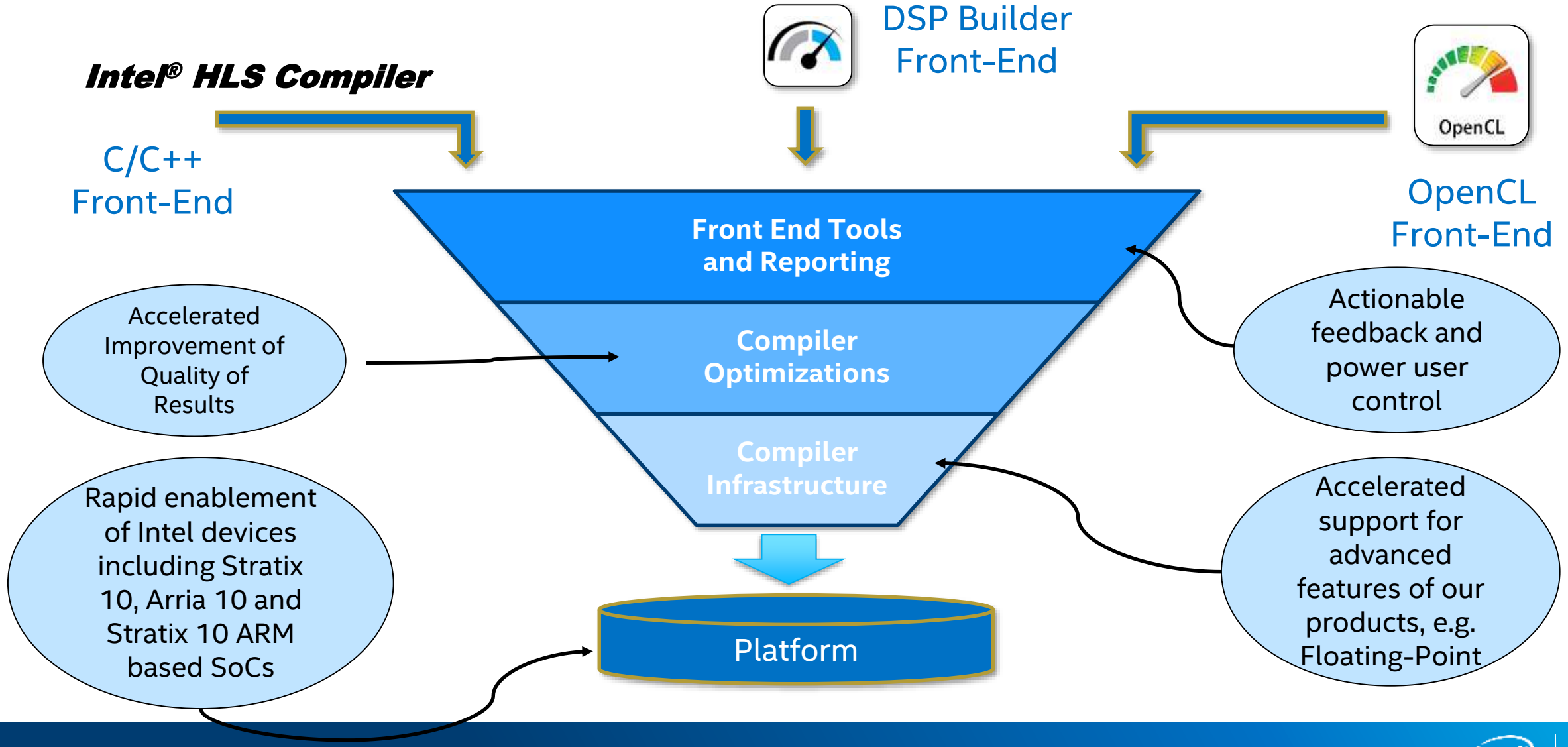
Programmers develop in mature software environments

- Ideas can easily be expressed in languages such as 'C'
  - Typically start with simple sequential program
  - Use parallel APIs / language extensions to exploit multi core for additional performance
- Compilation times are almost instantaneous
- Immediate feedback
- Rich debugging tools

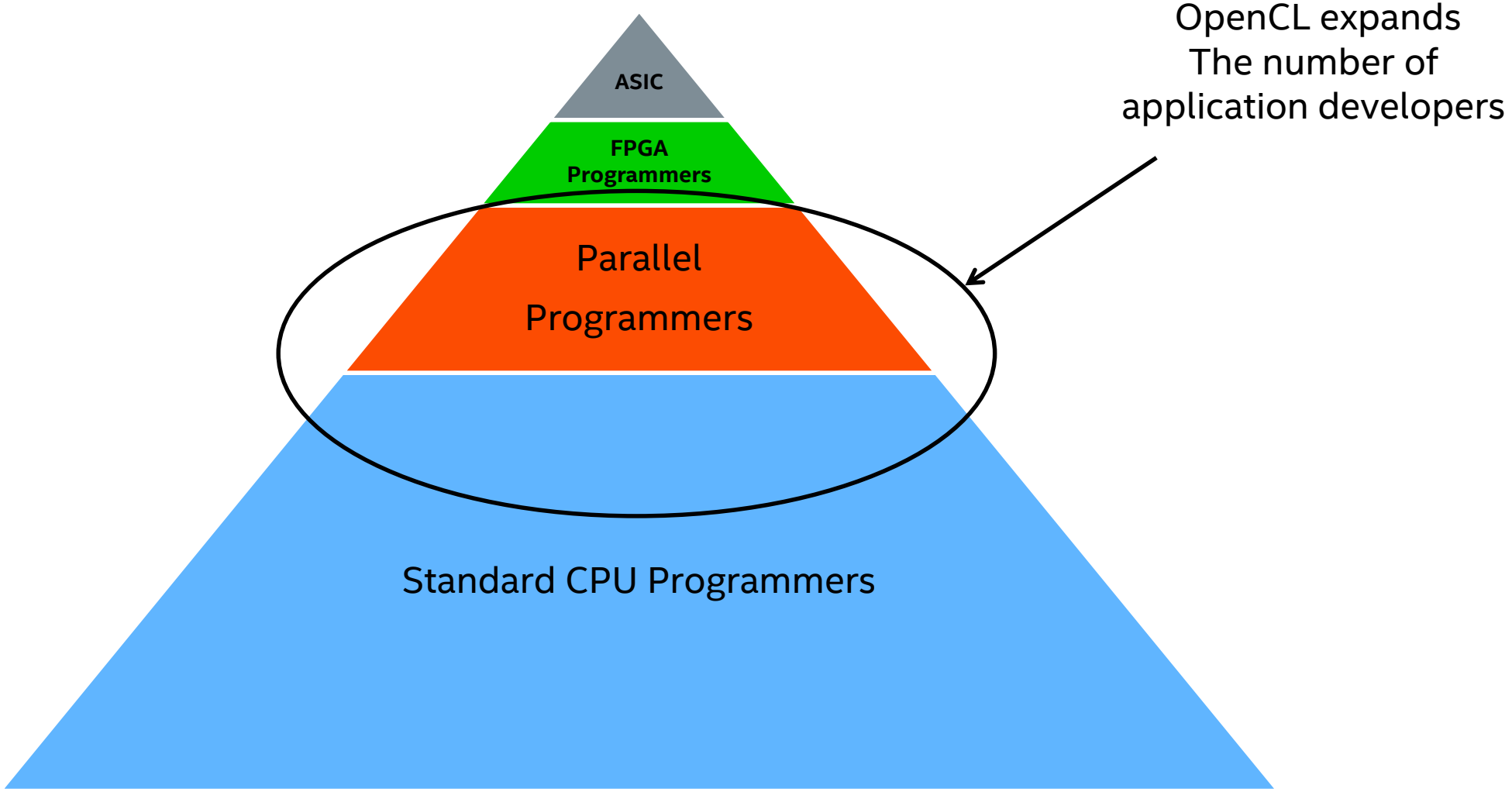
Different Solutions for Different Users



Accelerating HLD Tool Improvements



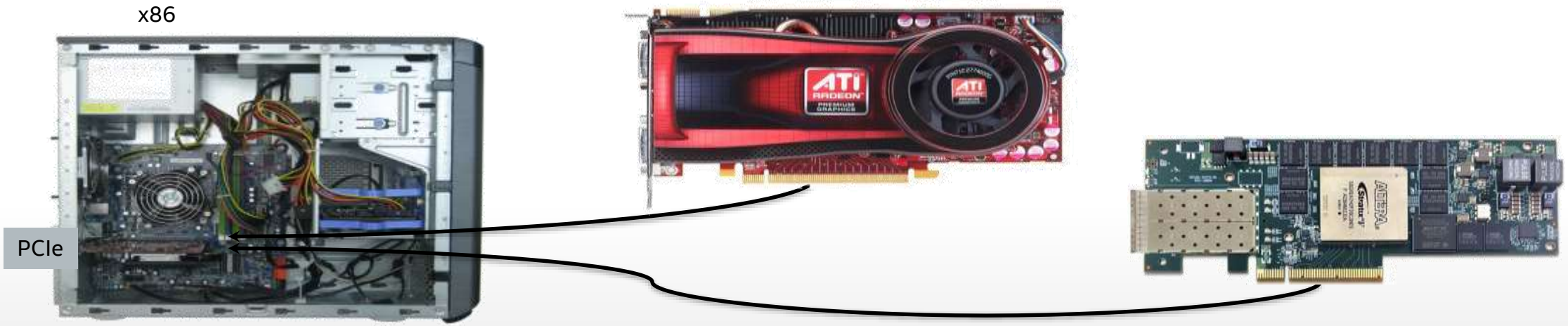
OpenCL Expanding the User Base



Heterogeneous Platform Model

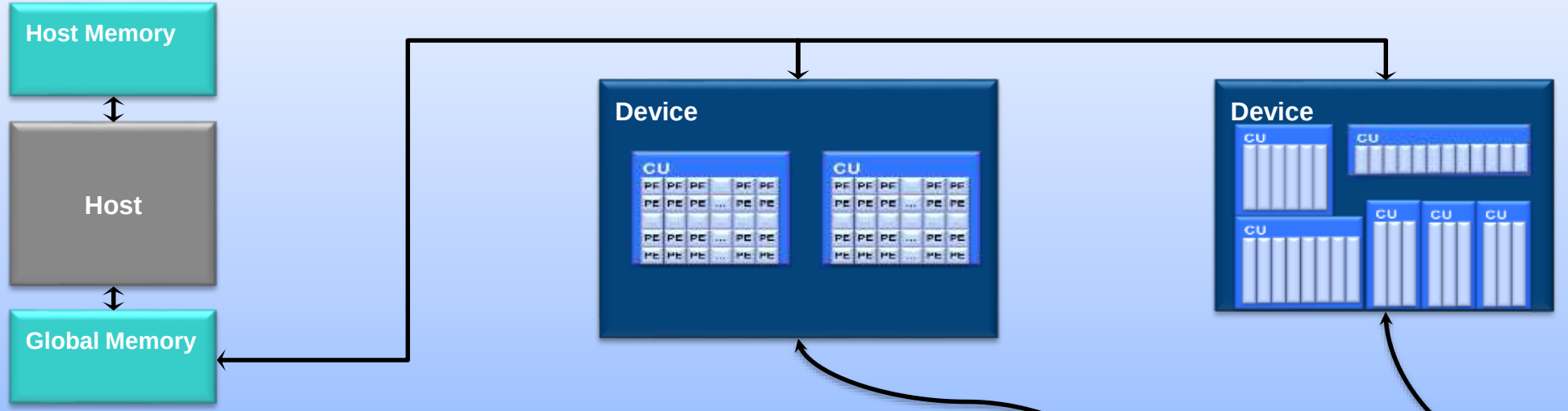
OpenCL Platform Model

Example Platform

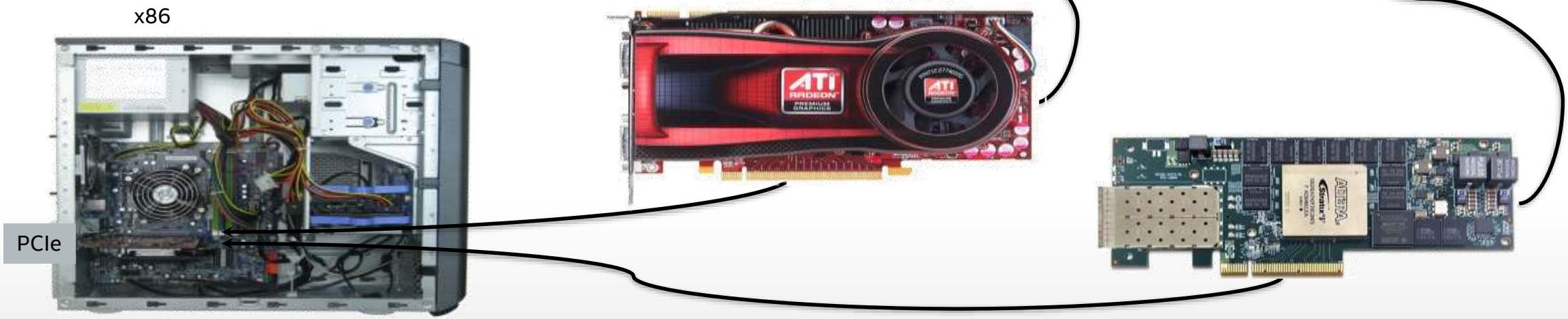


Heterogeneous Platform Model

OpenCL Platform Model



Example Platform



OpenCL Use Model

Host Code

```
main() {
  read_data( ... );
  manipulate( ... );
  clEnqueueWriteBuffer( ... );
  clEnqueueNDRange(...,sum,...);
  clEnqueueReadBuffer( ... );
  display_result( ... );
}
```

OpenCL Accelerator Code

```
__kernel void sum
  (__global float *a,
   __global float *b,
   __global float *y)
{
  int gid = get_global_id(0);
  y[gid] = a[gid] + b[gid];
}
```

Host

Standard gcc Compiler

Intel FPGA Offline Compiler

EXE

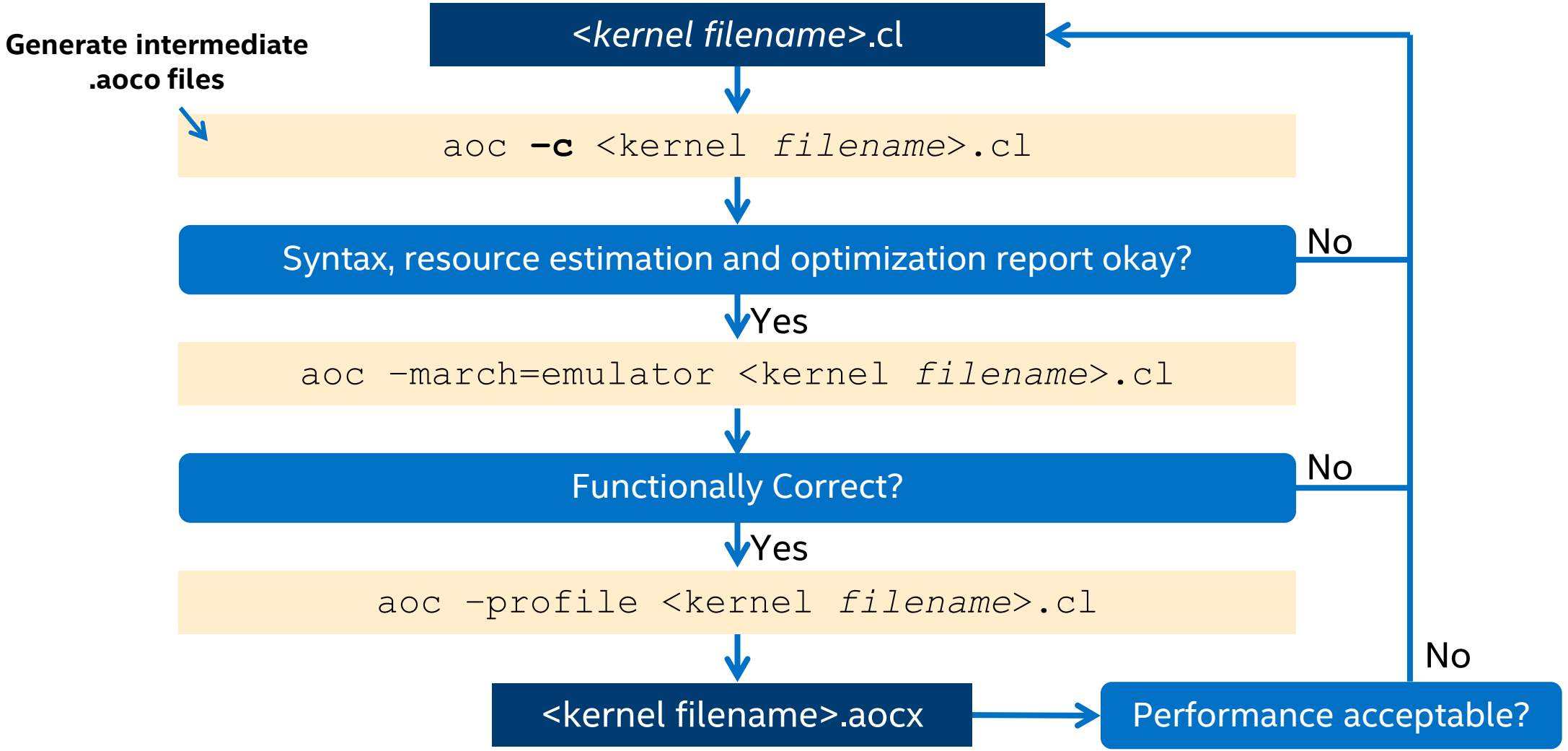
AOCX

Verilog

Quartus Prime Design Software

Accelerator

OpenCL Tool Flow

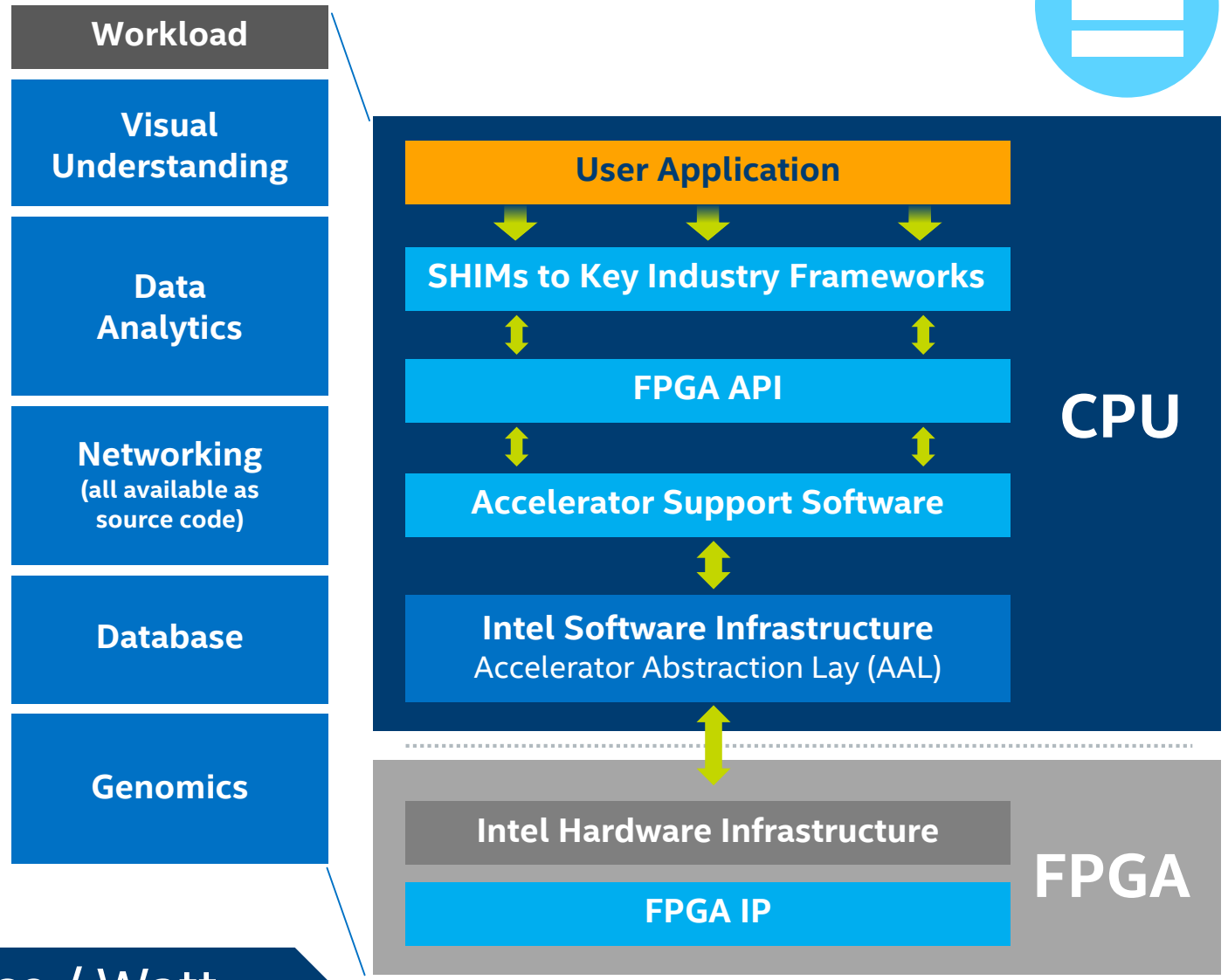


LIBRARIES

Library Approach

Intel Provided Building Blocks

- Intel® Xeon® processor library elements
- FPGA intellectual property (IP)
- Intel® Xeon® processor calls FPGA IP



Easier to Use and High Performance / Watt

Black Scholes Options Pricing

Price 100K to 1M options portfolio

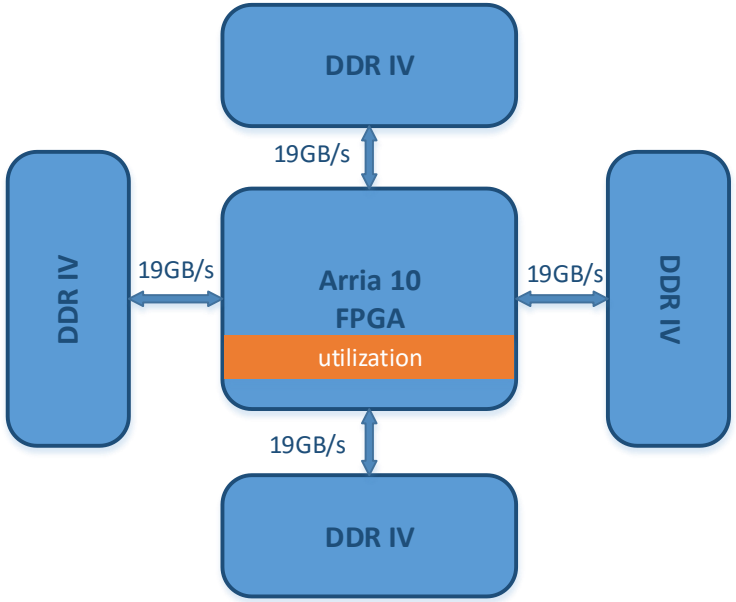
- 8 Black Scholes Engines, 4 DDR IV interfaces
- 5 Inputs, 1 output
- **3.2 Billion option/sec**

Adding Greeks

- One additional engine per DDR IV
- 32% increase in resources

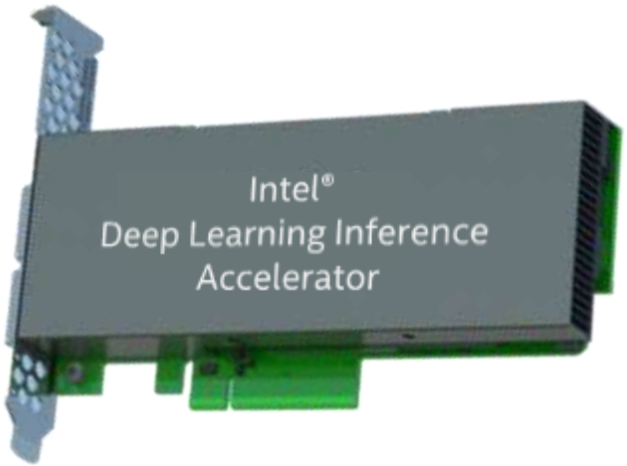
Fin-Lib phase 1

- Demo available in Q4 2016



Models	ALMs	RAMs	DSPs
Black-Scholes wo Greeks	1%	2%	5%
Black-Scholes with Greeks	1%	2%	8%
Bachelier	3%	12%	31%

Machine Learning Inference



Topology

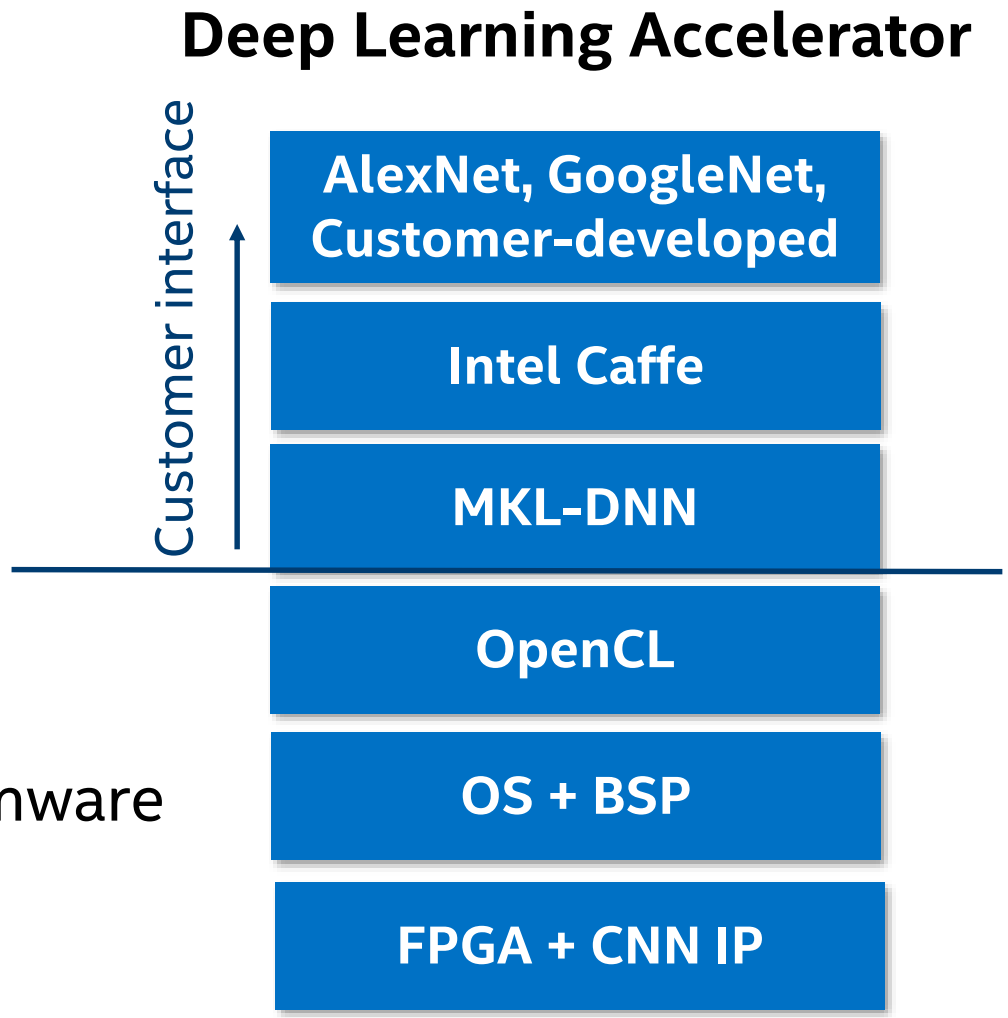
Framework

SW Library

Run time libraries

Operating System / Firmware

PCIe Hardware




intel