ECE 6775
High-Level Digital Design Automation
Fall 2024

Hardware Specialization

Cornell University

CSL

Announcements

- First reading assignment
  - A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression", IEEE CAS-M 2021
  - **Complete reading before Thursday 9/5**

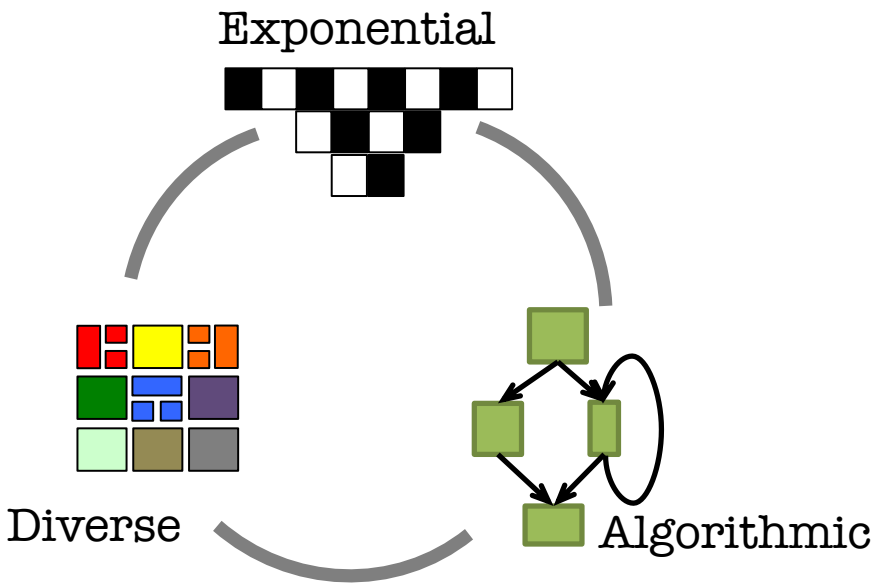
- Lab 1 and an HLS tool setup guide will be released soon (by Monday)

Recap: Our Interpretation of E-D-A

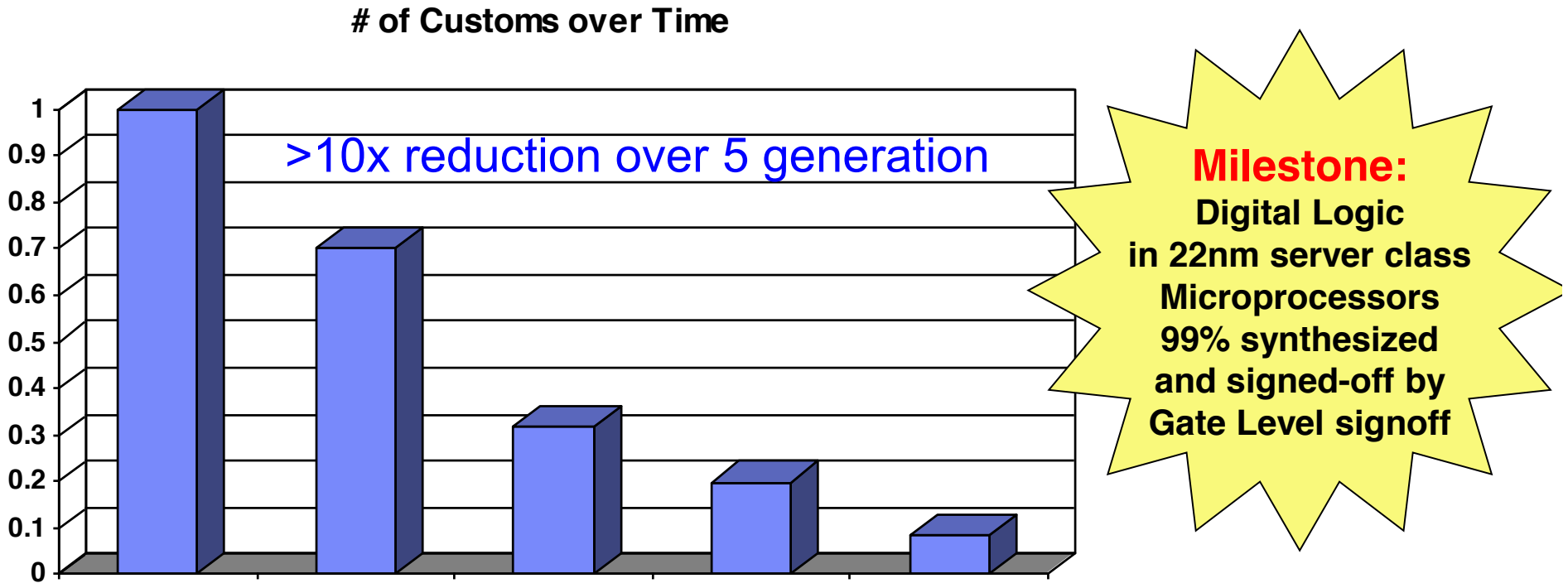
Exponential
in complexity (or **E**xtreme scale)

Diverse
increasing system heterogeneity

Algorithmic
intrinsically computational



Significance of EDA: Another Proof



Ruchir Puri, High Performance Microprocessor Design, and Automation: Challenges and Opportunities, TAU'2013 keynote.

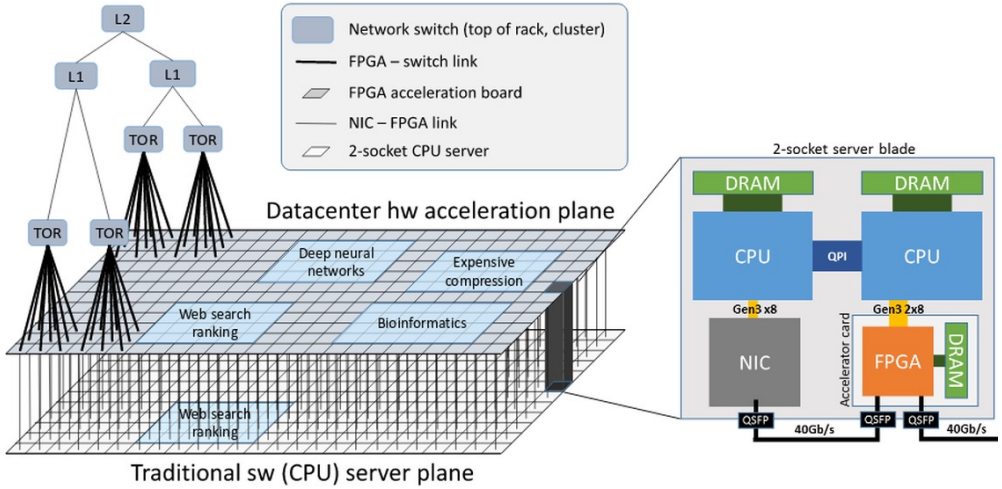
Agenda

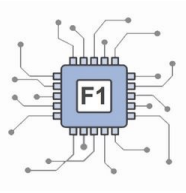
- Motivation for hardware specialization
  - Key driving forces from applications and technology
  - Main sources of inefficiency in general-purpose computing
- A taxonomy of common specialization techniques

A Golden Age of Hardware Specialization

- **Higher demand** on efficient compute acceleration, esp. for machine learning (ML) workloads

- **Lower barrier** with open-source hardware & accelerators in cloud coming of age





Rising Computational Demands of Emerging Applications

- Deep neural networks (DNNs) require enormous amount of compute
  - Consider ResNet50, a 70-layer model that performs 7.7 billion operations to classify an image (a relatively small model by today's standards)

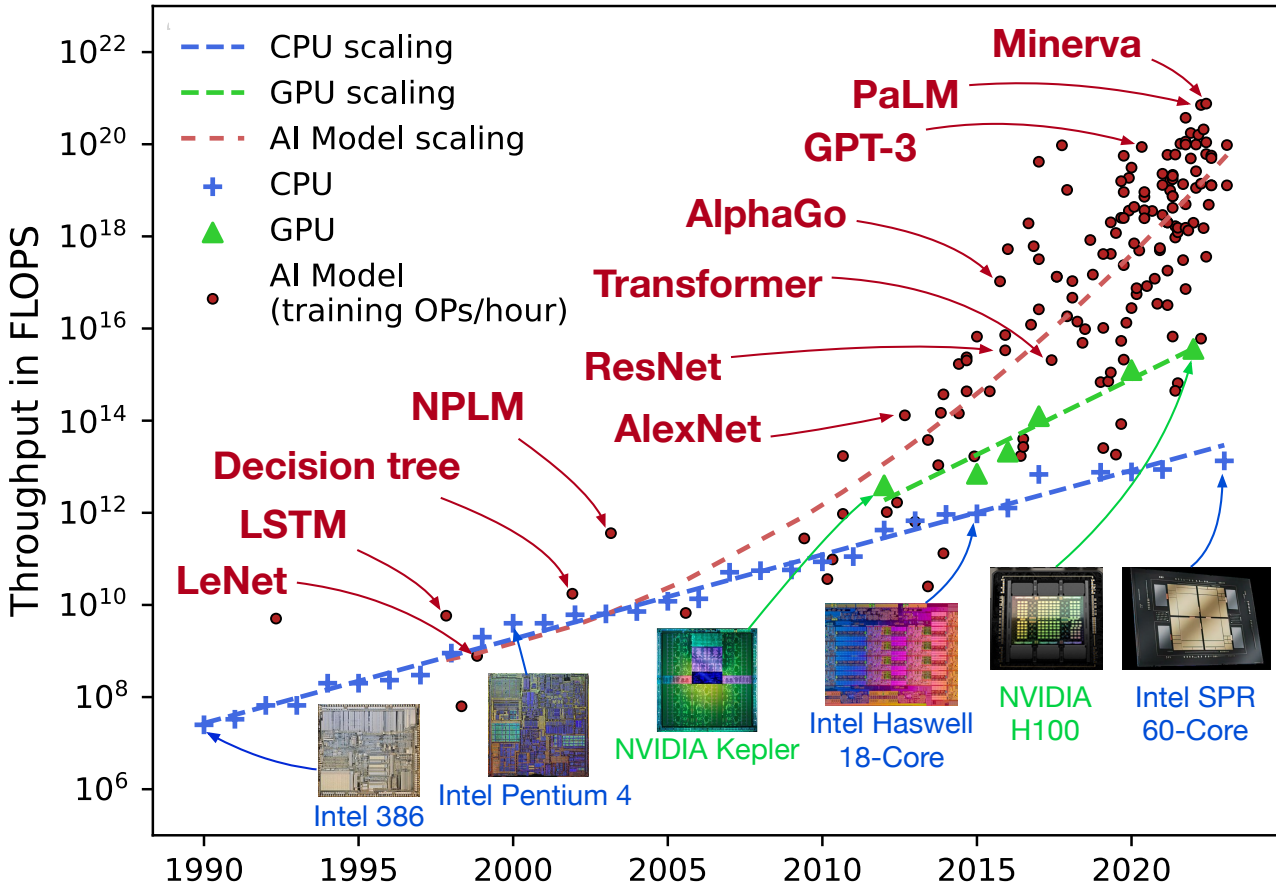
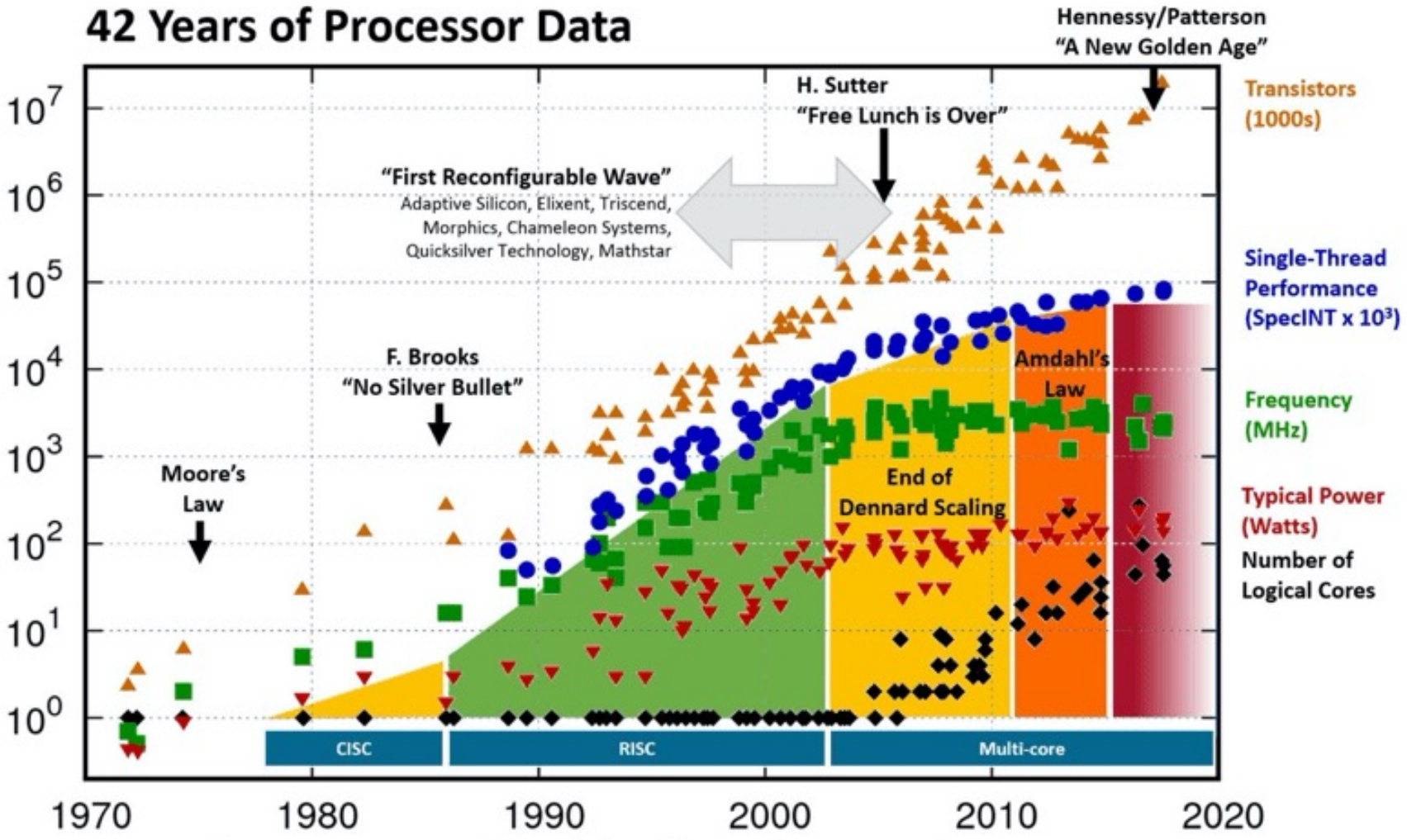


Figure source: Cornell Zhang Research Group

On Crash Course with the End of "Cheap" Technology Scaling



Hennessy and Patterson, Turing Lecture 2018, overlaid over "42 Years of Processors Data"
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/;"First Wave" added by Les Wilson, Frank Schirrmeister
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Dennard Scaling in a Nutshell

- Classical Dennard scaling
  - Frequency increases at constant power profiles
  - Performance improves "**for free**"**!**

Dennard scaling

Transistor (trans.) #	S^2
Capacitance / trans.	1/S
Voltage (V_{dd})	1/S
Frequency	S
Total power	**1**

Note: Dynamic power $\propto CV^2F$

End of Dennard Scaling and its Implications

- Power limited scaling
  - V_{th} scaling halted due to exponentially increasing leakage power
  - V_{DD} scaling nearly stopped as well to maintain performance

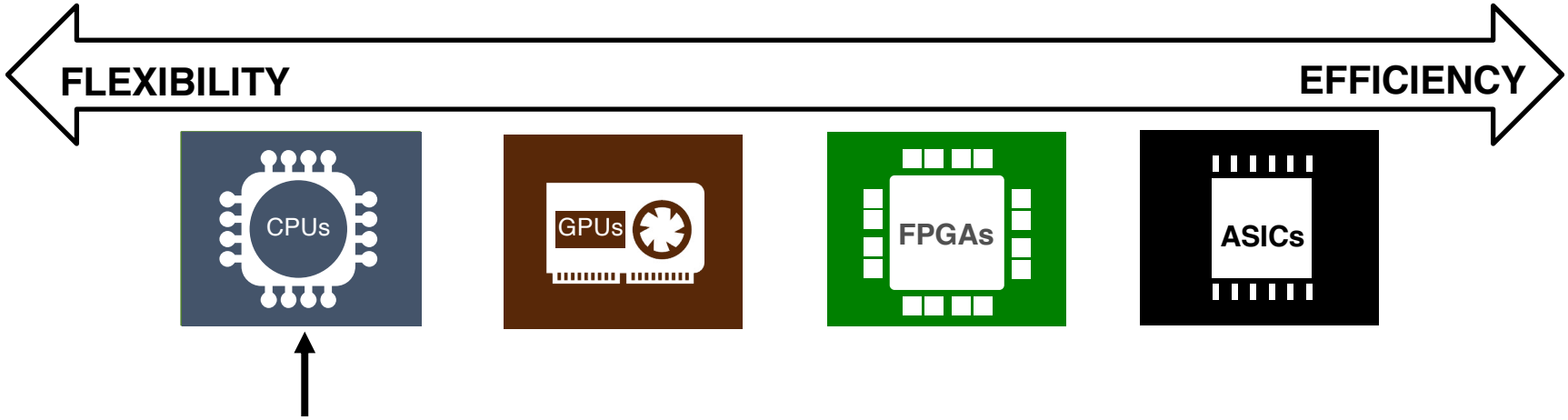
Leakage limited scaling

Transistor (trans.) #	S^2
Capacitance / trans.	1/S
Voltage (V_{dd})	**~1**
Frequency	**~1**
Total power	**S**

Note: Dynamic power $\propto CV^2F$

- Implication: “Dark silicon”?
  - Power limits restrict how much of the chip can be activated simultaneously
  - No longer 100% without more power

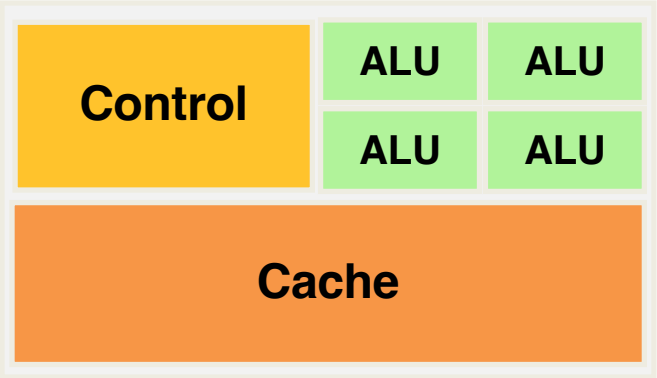
Trade-off Between Flexibility and Efficiency



Why are general-purpose CPUs less energy efficient?

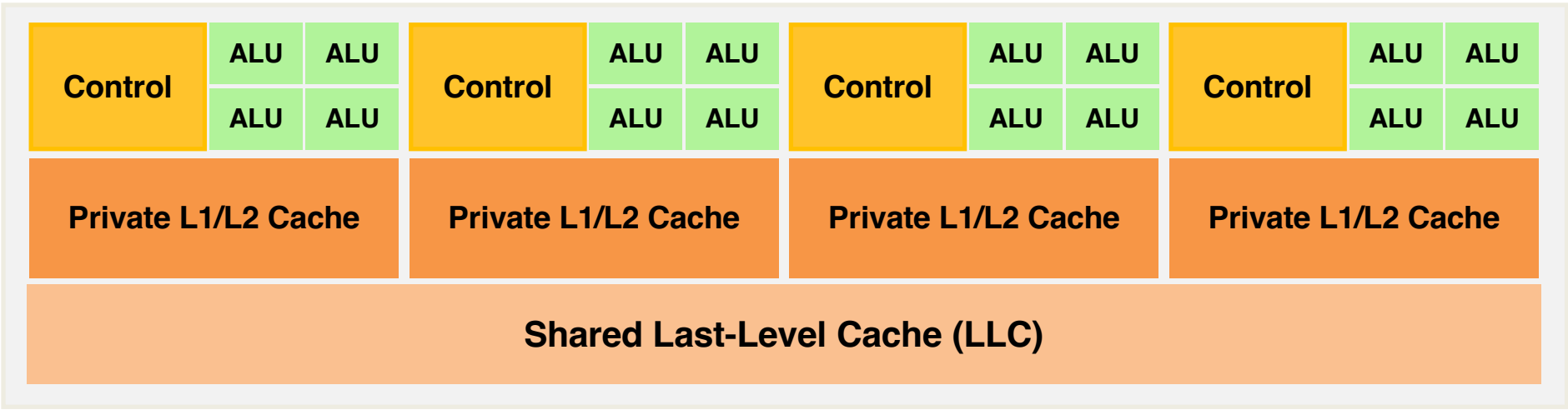
CPU Core Architecture

- Core = complex control + limited # of compute units + large caches
  - Scalar & vector instructions
    - Backward compatible ISA
  - Complex control logic: decoding, hazard detection, exception handling, etc.



- Mainly optimized to reduce latency of running serial code
  - Shallow pipelines (< 30 stages)
  - Superscalar, OOO, speculative execution, branch prediction, prefetching, etc.
  - Low throughput, even with multithreading

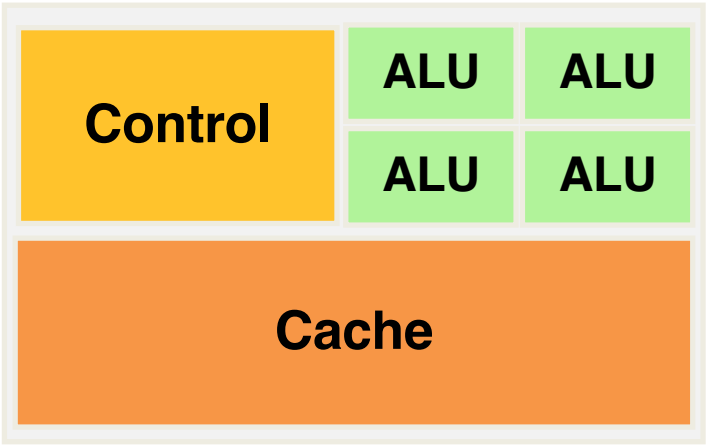
Multi-Core Architecture



With four cores, should we expect a 4x speedup on an arbitrary application?

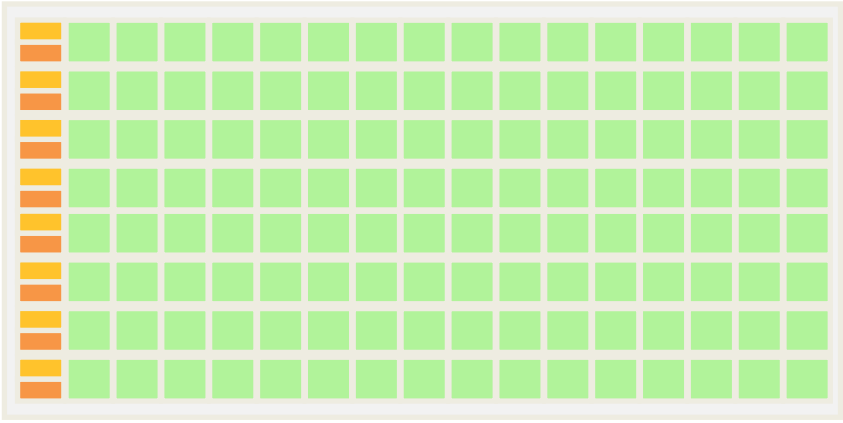
Graphics Processing Unit (GPU)

- GPU has thousands of cores to run many threads in parallel
  - Cores are simpler (compared to CPU)
    - No support of superscalar, OOO, speculative execution, etc.
    - ISA not backward compatible
  - Amortize overhead with SIMD + single instruction multiple threads (SIMT)



CPU

- Optimized to increase throughput of running data-parallel applications
  - Initially targeting graphics code
  - Latency tolerant with many concurrent threads



GPU

It's Not Just About Performance: Computing's Energy Problem

Reading two 32b words from	**Energy**
DRAM	1.3 nJ
Large SRAM (256MB)	58 pJ
Small SRAM (8KB)	5 pJ
Moving two 32b words by	**Energy**
40mm (across a 400mm2 chip)	77 pJ
1mm (local communication)	1.9 pJ
Arithmetic on two 32b words	**Energy**
FMA (float fused multiply-add)	1.2 pJ
IADD (integer add)	0.02 pJ

65,000x

250x

Data from [1], based on a 14nm process

Data supply far outweighs arithmetic operations in energy cost

[1] William Dally and Uzi Vishkin, On the Model of Computation, CACM'2022.

Rough Energy Breakdown for an Instruction

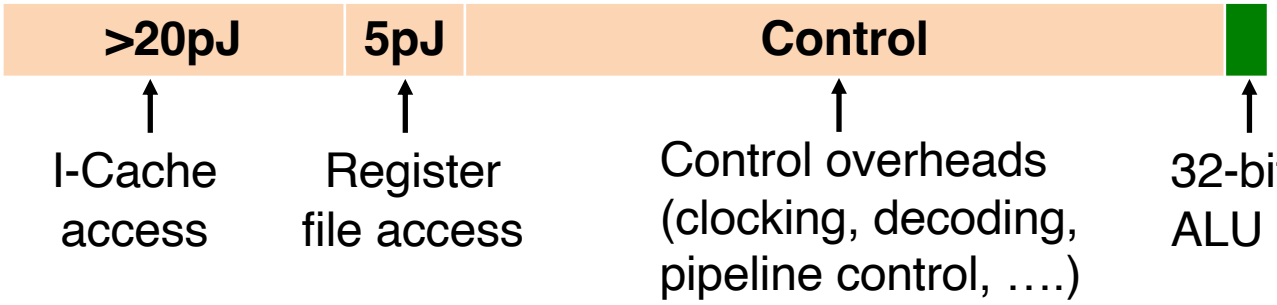


Diagram adapted from W. Qadder, et al., Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing, ISCA'2013.

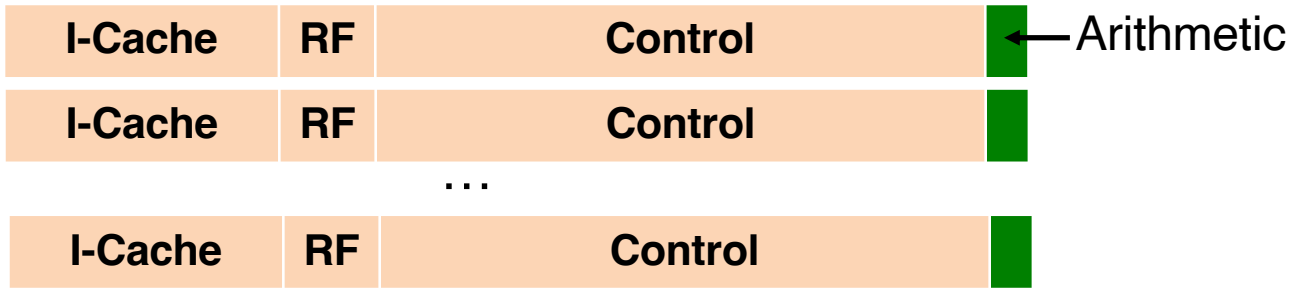
Principles for Improving Energy Efficiency

Do less work!

- **Amortize overhead** in control and data supply across multiple instructions

Amortizing the Overhead

A sequence of energy-inefficient instructions



Single instruction multiple Data (SIMD): tens of operations per instruction

Further specialization (what we achieve using accelerators)

Diagram adapted from W. Qadder, et al., Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing, ISCA'2013.

Principles for Improving Energy Efficiency

Do less work!

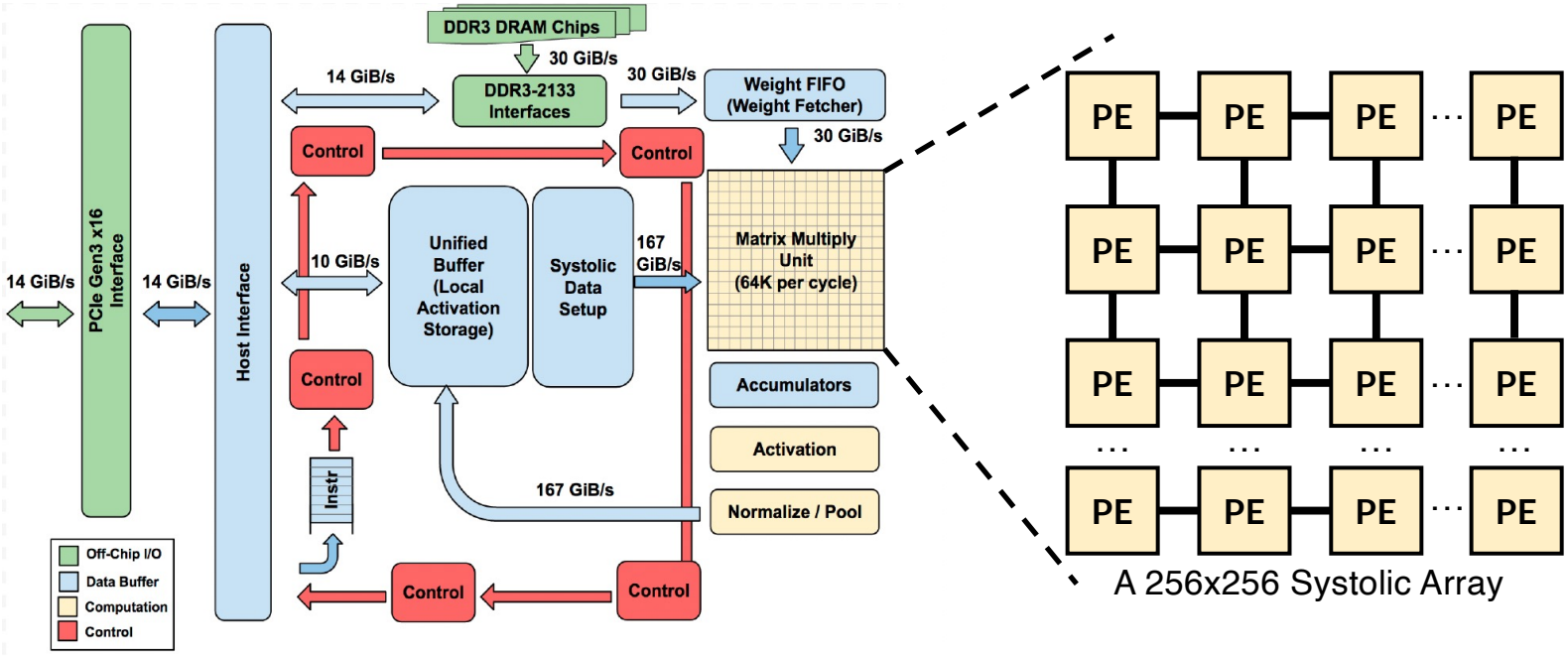
- **Amortize overhead** in control and data supply across multiple instructions

Do even less work!

- **Use smaller (or simpler) data** => cheaper operations, lower storage & communication costs
- **Move data locally and directly**
  - Store data nearby in simpler memory (e.g., scratchpads are cheaper than cache)
  - Wire compute units for direct (or even combinational) communication when possible

Tensor Processing Unit (TPU)

- A domain-specific accelerator specialized for deep learning
  - Main focus: accelerating matrix multiplication (MatMul) with a systolic array
    - Use CISC instructions: MatMul Unit may take thousands of cycles
  - TPUv1 does 8-bit integer (INT8) inference; TPUv2 supports a customized floating-point type (bfloat16) for training



Google TPU v1

Source: Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA 2017

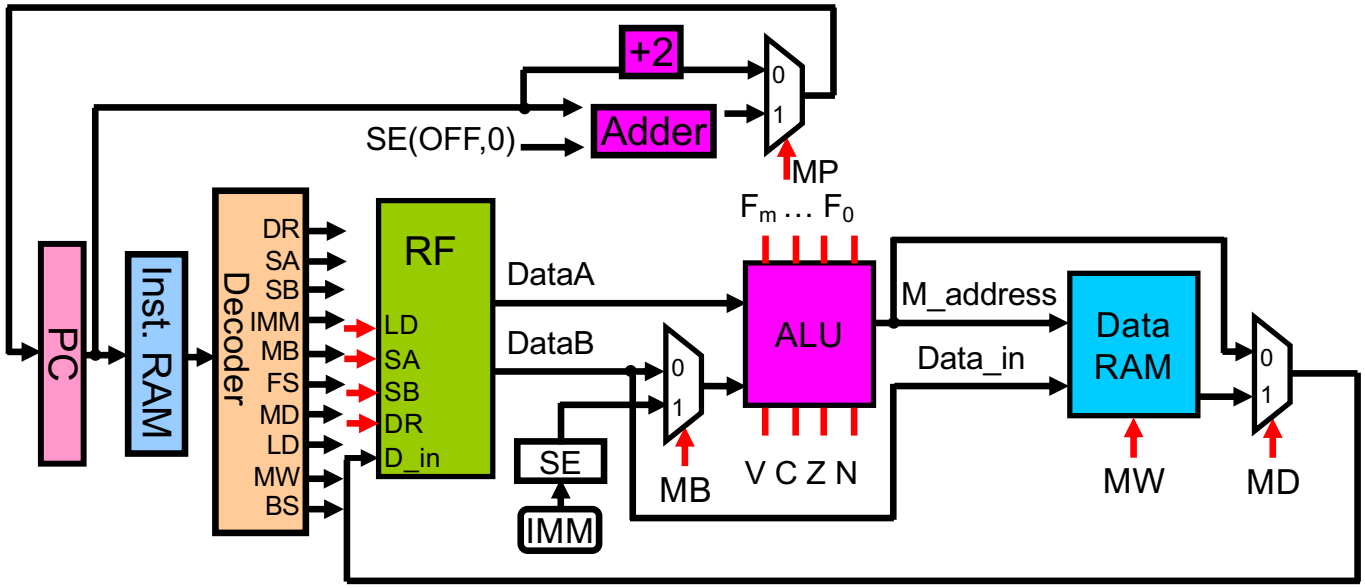
Common Hardware Specialization Techniques: A Taxonomy

- **Custom Compute Units:** Use complex instructions to amortize overhead (e.g., SIMD, “ASIC”-in-an-instruction)
- **Custom Numeric Types:** Trade off accuracy and efficiency with data types that use smaller bit widths or simpler arithmetic
- **Custom Memory Hierarchy:** Exploit data access patterns to reduce energy per memory operation
- **Custom Communication Architecture:** Tailor on-chip networks to data movement patterns

Common Hardware Specialization Techniques: A Taxonomy

- **Custom Compute Units:** Use complex instructions to amortize overhead (e.g., SIMD, “ASIC”-in-an-instruction)
- **Custom Numeric Types:** Trade off accuracy and efficiency with data types that use smaller bit widths or simpler arithmetic
- **Custom Memory Hierarchy:** Exploit data access patterns to reduce energy per memory operation
- **Custom Communication Architecture:** Tailor on-chip networks to data movement patterns

Customizing Compute Units: An Intuitive View



A simple single-cycle CPU

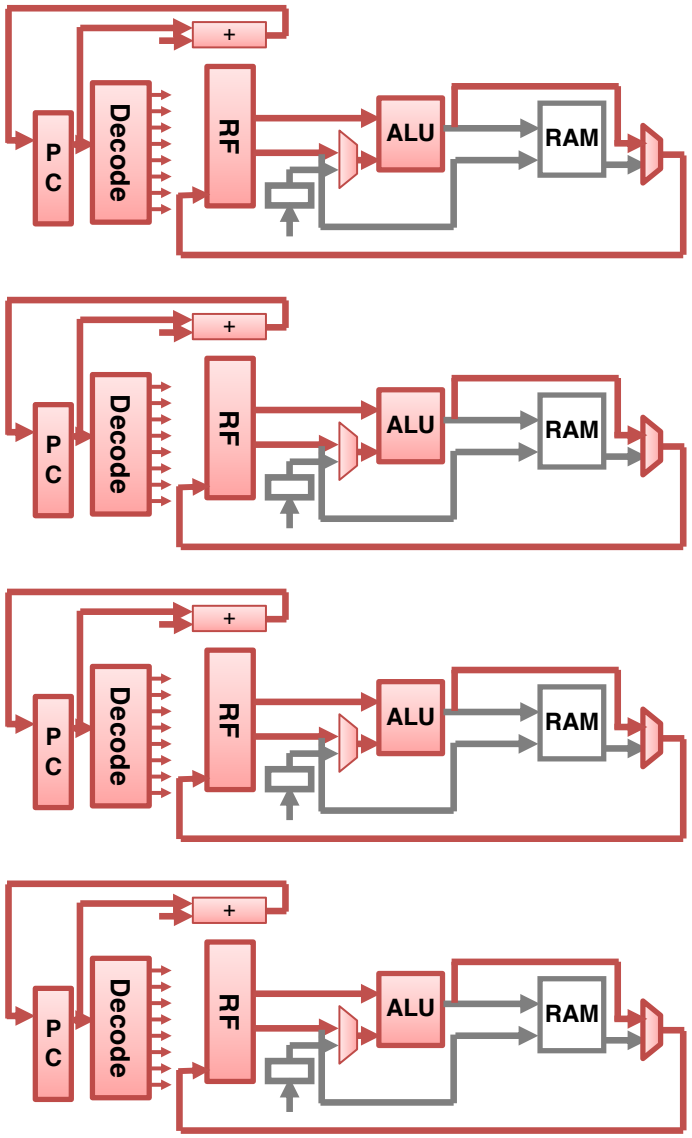
Evaluating a Simple Expression on CPU

R5 <= R1 * R3

R6 <= R2 * R4

R7 <= R5 - R6

R8 <= R9 + R7



Cycle-by-cycle CPU activities

Source: Adapted from Desh Singh's talk at HCP'14 workshop

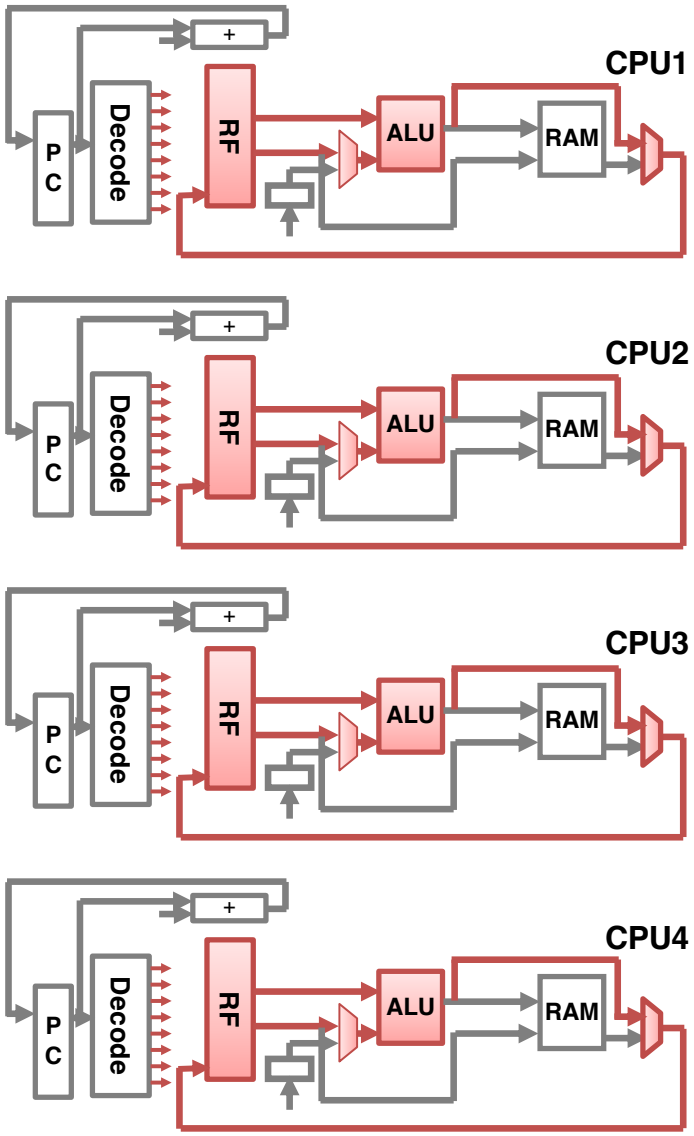
“Unrolling” the Instruction Execution

R5 <= R1 * R3

R6 <= R2 * R4

R7 <= R5 - R6

R8 <= R9 + R7



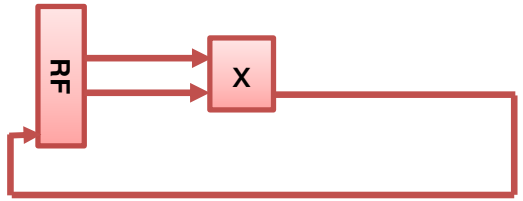
1. Replicate the CPU hardware
Instruction fixed => disable fetch & decode logic

Space

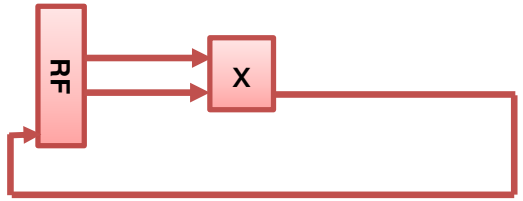
Source: Adapted from Desh Singh’s talk at HCP’14 workshop

Removing Unused Logic

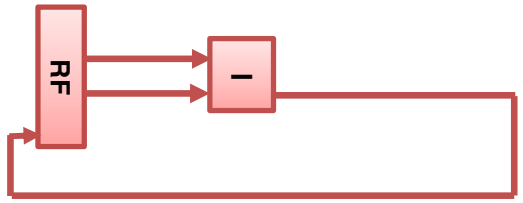
R5 <= R1 * R3



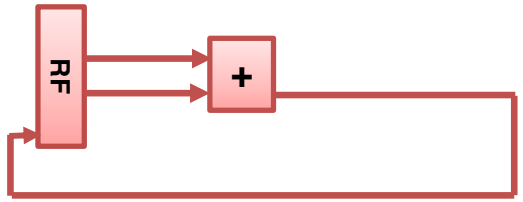
R6 <= R2 * R4



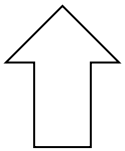
R7 <= R5 - R6



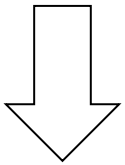
R8 <= R9 + R7



2. Removing unused logic
=> ALU also simplified



Space



Source: Adapted from Desh Singh's talk at HCP'14 workshop

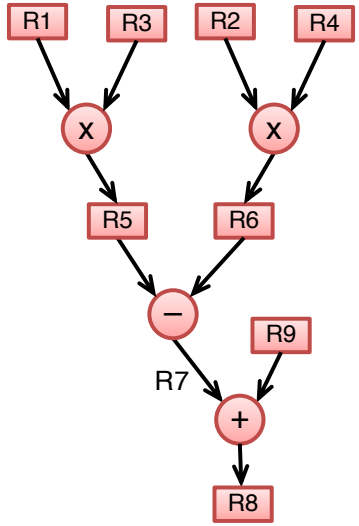
An Application-Specific Compute Unit

R5 <= R1 * R3

R6 <= R2 * R4

R7 <= R5 - R6

R8 <= R9 + R7



3. Wire up registers and functional units

Use combinational connections when timing constraints allow (e.g., R7)

Source: Adapted from Desh Singh's talk at HCP'14 workshop

Common Hardware Specialization Techniques: A Taxonomy

- **Custom Compute Units:** Use complex instructions to amortize overhead (e.g., SIMD, “ASIC”-in-an-instruction)

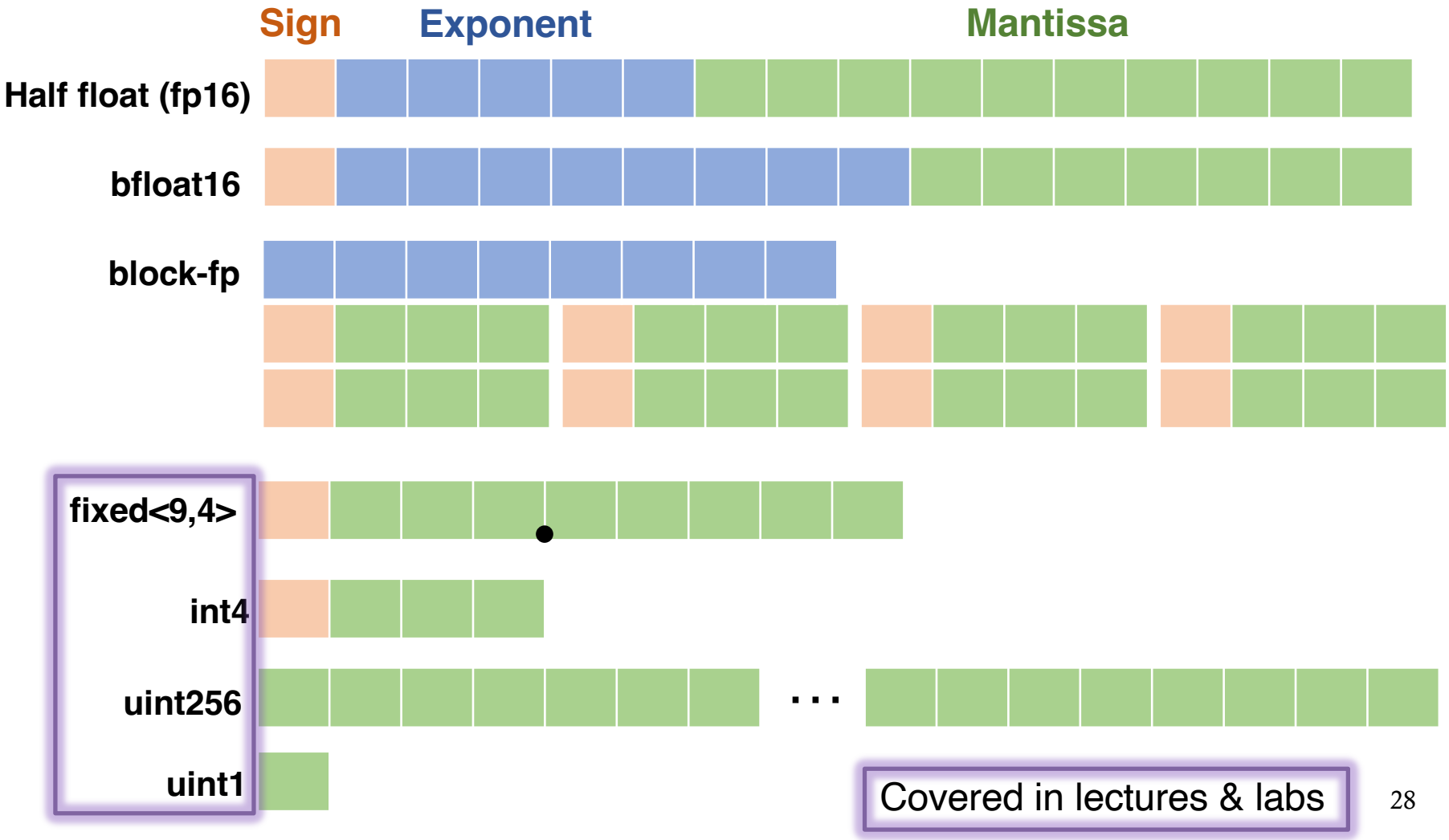
- **Custom Numeric Types:** Trade off accuracy and efficiency with data types that use smaller bit widths or simpler arithmetic

- **Custom Memory Hierarchy:** Exploit data access patterns to reduce energy per memory operation

- **Custom Communication Architecture:** Tailor on-chip networks to data movement patterns

Customized Data Types

- Using custom numeric types tailored for a given application/domain improves performance & efficiency



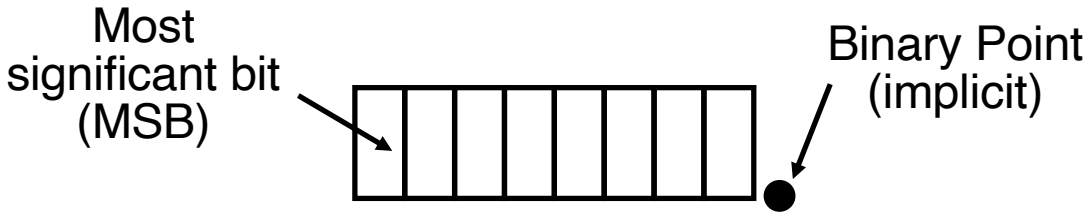
Binary Representation – Positional Encoding

Unsigned number

- MSB has a place value (weight) of 2^{n-1}

Two's complement

- MSB weight = -2^{n-1}

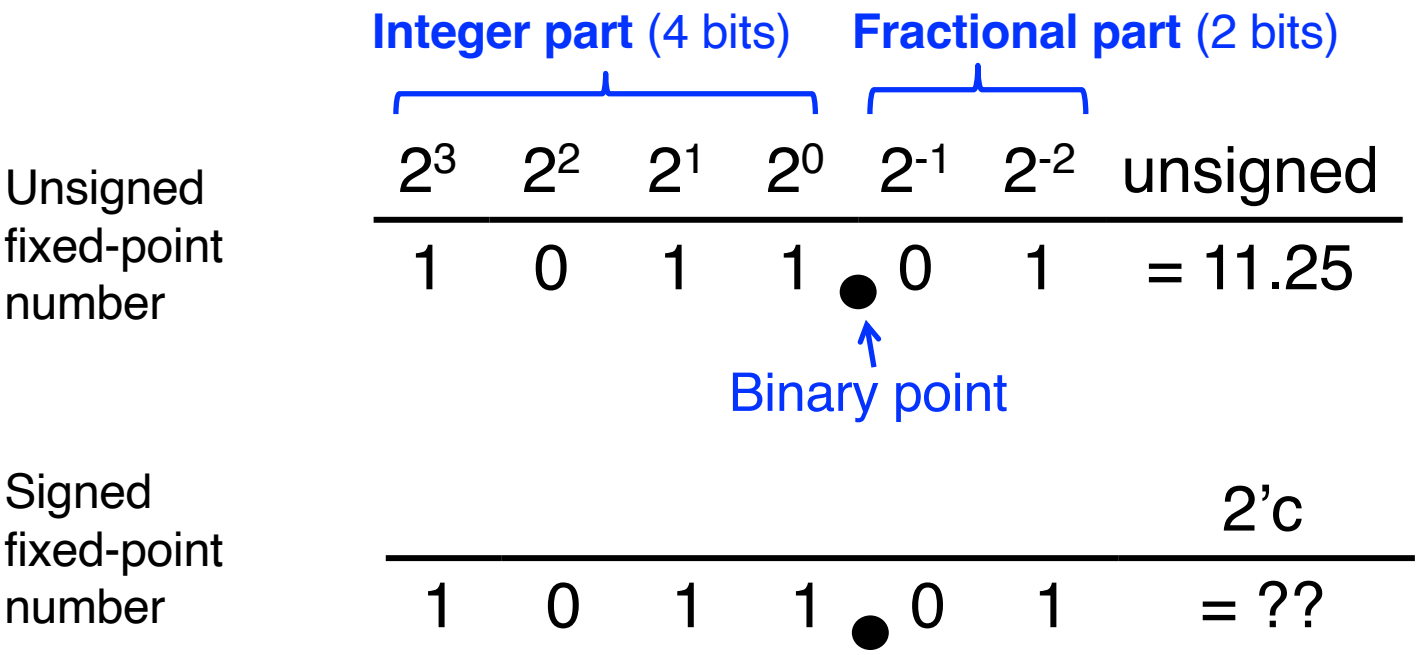


2^3	2^2	2^1	2^0	unsigned
1	0	1	1	= 11

-2^3	2^2	2^1	2^0	2'c
1	0	1	1	= -5

Fixed-Point Representation of Fractional Numbers

- The positional binary encoding can also represent fractional values, by using a **fixed** position of the binary point and place values with negative exponents
  - (–) Less convenient to use in software, compared to floating point
  - (+) Much more efficient in hardware



Overflow and Quantization Issues

- When working with fixed-point types, overflow and quantization issues often arise due to the limited range and precision of representation

- **Overflow** occurs when a value exceeds the maximum representable range for the fixed-point format
  - In our class, overflow mainly concerns the integer part of the fixed-point number

- **Quantization** is needed when converting real numbers from a higher-precision format (like float) to fixed point

Common Modes for Handling Overflow

- **Wrapping** or wraparound: The value wraps around within the range using modulo arithmetic
  - Efficient in hardware, as it involves simply dropping the MSB(s) of the original number

-2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2'c
1	0	1	1	0	1	= -4.75

Dropping MSB when integer width is reduced

					2'c
0	1	1	0	1	= ??

Wrapping can cause a negative number to become positive, or a positive to negative

- **Saturation**: Limits the value to the maximum (or minimum) representable value, preventing wraparound
  - What would be the saturated result for the example above?

Common Quantization Modes

- **Truncation**: Cuts off the excess bits that don't fit in the target precision
  - Efficient in hardware, as it involves simply dropping the LSB(s) of the original number

- **Rounding:** Rounds the value to a representable fixed-point number, potentially reducing quantization error compared to truncation
  - Round to Nearest: The most common method, where the value is rounded to the nearest fixed-point representation
  - Round Toward Zero: Rounds towards zero, effectively truncating the fractional part
  - Round Toward Infinity: Rounds away from zero, towards positive or negative infinity

Common Hardware Specialization Techniques: A Taxonomy

- **Custom Compute Units:** Use complex instructions to amortize overhead (e.g., SIMD, “ASIC”-in-an-instruction)

- **Custom Numeric Types:** Trade off accuracy and efficiency with data types that use smaller bit widths or simpler arithmetic

- **Custom Memory Hierarchy:** Exploit data access patterns to reduce energy per memory operation

- **Custom Communication Architecture:** Tailor on-chip networks to data movement patterns

Next lecture

Next Lecture

- More Hardware Specializatoin

Acknowledgements

- These slides contain/adapt materials developed by
  - Bill Dally, NVIDIA
  - System for AI Education Resource by Microsoft Research