

Computer Architecture

Yipeng Huang

Rutgers University

January 19, 2021

Table of contents

What this class is about: abstractions

- Low-level programming

- The memory hierarchy

- Digital logic

Class mechanics

- Class staff

- Accessing the class & resources

- Lecture and weekly short quizzes

- Programming assignments

- Recitation code review study groups

A New Golden Age for Computer Architecture

- History: computer architecture abstractions drove digital revolution

- Present: power constraints driving diverse computer architectures

- Future: post-Moore's Law computer architectures

What this class is about: abstractions

Intermediate courses in computer science:

CS 112: Data structures

learned about how to store data and manipulate data with algorithms

CS 205/206: Discrete structures

learn about the mathematics that govern computer science

CS 211: Computer architecture

learn about the abstractions that make programs run on computer building blocks

CS 213: Software methodology

learn how to organize complex programs

CS 214: Systems programming

learn how to interact with the operating system

What this class is about

Learning goal

Throughout the course, students will learn about important computing abstractions such as low-level programming, the memory hierarchy, and digital logic via case studies that are representative of real-world computer systems.

Important computing abstractions

Abstractions

A way to hide the details of an underlying system so you (users & programmers) can be more creative.

Low-level programming

C, assembly language, machine code, instruction set architecture

The memory hierarchy

File system, main memory, caches, data representations

Digital logic

Pipelines, registers, flip flops, arithmetic units, gates

Low-level programming: C

Learn a new and foundational programming language.

```
int main() {  
    printf("Hello, World!");  
    return 0;  
}
```

Low-level programming: assembly

Study the interface between software and hardware.

```
MOV [ESI+EAX], CL ; Move the contents of CL into the byte at address  
ESI+EAX
```

```
MOV DS, DX ; Move the contents of DX into segment register DS
```

The memory hierarchy

Computer Memory Hierarchy

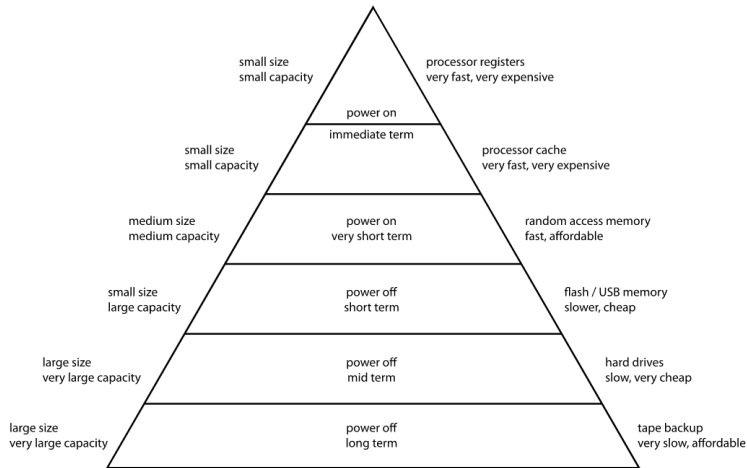
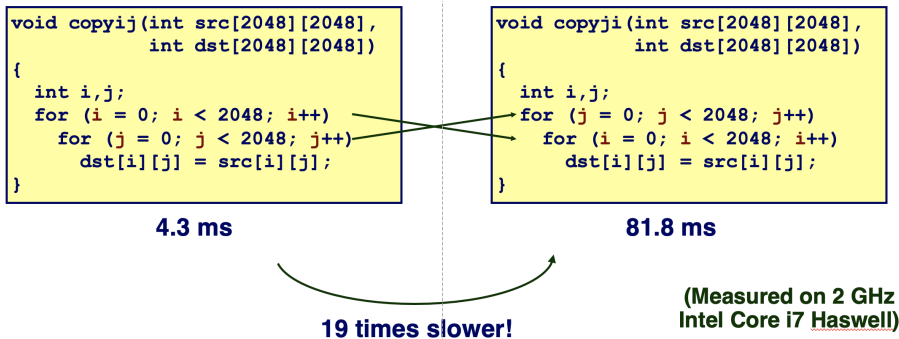


Figure: Credit: wikimedia

The memory hierarchy



- Hierarchical memory organization
- Performance depends on access patterns
 - Including how step through multi-dimensional array

Figure: Credit: Computer Systems: A Programmer's Perspective

Data representations

```
void show_squares()
{
    int x;
    for (x = 5; x <= 5000000; x*=10)
        printf("x = %d x^2 = %d\n", x, x*x);
}
```

x =	5	x ² =	25
x =	50	x ² =	2500
x =	500	x ² =	250000
x =	5000	x ² =	25000000
x =	50000	x ² =	-1794967296
x =	500000	x ² =	891896832
x =	5000000	x ² =	-1004630016

- Numbers are represented using a finite word size
- Operations can overflow when values too large
 - But behavior still has clear, mathematical properties

Figure: Credit: Computer Systems: A Programmer's Perspective

Table of contents

What this class is about: abstractions

- Low-level programming

- The memory hierarchy

- Digital logic

Class mechanics

- Class staff

- Accessing the class & resources

- Lecture and weekly short quizzes

- Programming assignments

- Recitation code review study groups

A New Golden Age for Computer Architecture

- History: computer architecture abstractions drove digital revolution

- Present: power constraints driving diverse computer architectures

- Future: post-Moore's Law computer architectures

Class staff

Prof. Yipeng Huang
yipeng.huang@rutgers.edu

Teaching assistants

- ▶ Marie Petitjean
- ▶ Neeraj Mula
- ▶ Abhinav Sirohi
- ▶ Prince Rawal
- ▶ Azita Nouri
- ▶ Gabrielle Capulong

<https://rutgers.instructure.com/courses/104725/pages/recitation-and-office-hour-information>

Prof. Yipeng Huang

<http://cs.rutgers.edu/~yh804>

My research is in abstractions that allow us to use novel computer architectures such as quantum and analog computers.

- ▶ I am looking for students who want to pursue research projects.
- ▶ In the fall I teach CS 583—Quantum Computing: Programs and Systems
- ▶ Worked with DARPA to investigate feasibility of using analog electronic circuits for scientific computation.
- ▶ PhD Dissertation: Hybrid Analog-Digital Co-Processing for Scientific Computation.
- ▶ MICRO Top Picks 2016, honorable mention 2017.
- ▶ Served on program committees for top computer architecture conferences: MICRO, HPCA, and reviewed papers for ASPLOS, ISCA.

Accessing the class & resources

Canvas

Announcements, lecture slides, videos, quizzes, assignments, submissions.

`https://rutgers.instructure.com/courses/104725/assignments/syllabus`

Textbooks

- ▶ Bryant and O'Hallaron. Computer Systems: A Programmer's Perspective. Prentice Hall. 3rd edition.
- ▶ Modern C: `https://modernc.gforge.inria.fr/`

Lecture and weekly short quizzes

Lectures

- ▶ You are encouraged to tune in live to keep up with the class and to ask questions.
- ▶ It continues to be a challenging time for students due to remote learning. I am not taking attendance.
- ▶ Videos will be posted within one day of lecture to YouTube, access link on Canvas.

Weekly short quizzes

- ▶ Ensure that you keep up with the class and check on basic concepts from previous week, and to collect feedback.
- ▶ 30 minutes for quiz, max two attempts, open for set time window.
- ▶ 2% of course grade each. No makeups, will have to recover points by doing well on assignments.

Programming assignments

Learning goal

Students will apply essential knowledge about computer systems to modify and create new low-level software and hardware implementations via hands-on programming exercises.

Programming assignments

ilab

- ▶ All students need access to ilab to compile, run, and test programs in Linux.
- ▶ If you do not have access, sign up immediately: `https://services.cs.rutgers.edu/accounts/activate/activate`

Piazza

- ▶ Ask all questions if possible on Piazza.
- ▶ If you send the instructor or any of the TAs an email that is better addressed on Piazza, we will kindly ask you to repost your question on Piazza and we will answer it there.
- ▶ Sign up now: `https://piazza.com/class/kjw3smy31un5hy`

Programming assignments

Automatic compiling, testing, and grading

- ▶ It is important that you carefully follow the specified output formats so that the testing framework can validate your program.
- ▶ New for this class: more incremental points, stricter validation, and more feedback from the grading system.

Submit on Canvas

- ▶ Start early.
- ▶ You can submit as many times as you wish.
- ▶ We will not accept late assignments; deadline will be enforced by Canvas.

Importance of writing your own code

INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):  
  IF LENGTH(LIST) < 2:  
    RETURN LIST  
  PIVOT = INT(LENGTH(LIST) / 2)  
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])  
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])  
  // UMMMMM  
  RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):  
  // AN OPTIMIZED BOGOSORT  
  // RUNS IN  $O(N \log N)$   
  FOR N FROM 1 TO LOG(LENGTH(LIST)):  
    SHUFFLE(LIST):  
    IF ISSORTED(LIST):  
      RETURN LIST  
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBININTERVIEWQUICKSORT(LIST):  
  OK SO YOU CHOOSE A PIVOT  
  THEN DIVIDE THE LIST IN HALF  
  FOR EACH HALF:  
    CHECK TO SEE IF IT'S SORTED  
    NO, WAIT, IT DOESN'T MATTER  
    COMPARE EACH ELEMENT TO THE PIVOT  
    THE BIGGER ONES GO IN A NEW LIST  
    THE EQUAL ONES GO INTO, UH  
    THE SECOND LIST FROM BEFORE  
  HANG ON, LET ME NAME THE LISTS  
  THIS IS LIST A  
  THE NEW ONE IS LIST B  
  PUT THE BIG ONES INTO LIST B  
  NOW TAKE THE SECOND LIST  
  CALL IT LIST, UH, A2  
  WHICH ONE WAS THE PIVOT IN?  
  SCRATCH ALL THAT  
  IT JUST RECURSIVELY CALLS ITSELF  
  UNTIL BOTH LISTS ARE EMPTY  
  RIGHT?  
  NOT EMPTY, BUT YOU KNOW WHAT I MEAN  
  AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):  
  IF ISSORTED(LIST):  
    RETURN LIST  
  FOR N FROM 1 TO 10000:  
    PIVOT = RANDOM(0, LENGTH(LIST))  
    LIST = LIST[PIVOT:] + LIST[:PIVOT]  
  IF ISSORTED(LIST):  
    RETURN LIST  
  IF ISSORTED(LIST):  
    RETURN LIST  
  IF ISSORTED(LIST): // THIS CAN'T BE HAPPENING  
    RETURN LIST  
  IF ISSORTED(LIST): // COME ON COME ON  
    RETURN LIST  
  // OH JEEZ  
  // I'M GONNA BE IN SO MUCH TROUBLE  
  LIST = [ ]  
  SYSTEM("SHUTDOWN -H +5")  
  SYSTEM("RM -RF ./")  
  SYSTEM("RM -RF ~/./*")  
  SYSTEM("RM -RF /")  
  SYSTEM("RD /S /Q C:\*") // PORTABILITY  
  RETURN [1, 2, 3, 4, 5]
```

Importance of writing your own code

Study programming smartly

- ▶ You are encouraged to discuss the homework with your classmates on Piazza.
- ▶ You are encouraged to research and study concepts online.

Importance of writing your own code

- ▶ But, you must not disclose your code or see your classmates' code.
- ▶ Finding your own solution and writing and debugging your own code is vital to your learning. Copying someone else's code short-circuits this process.
- ▶ We will use automatic tools to detect identical or similar submissions.

Rutgers Academic Integrity Policy

- ▶ <https://nbprovost.rutgers.edu/academic-integrity-students>
- ▶ Every offense will be reported to office of student conduct.

Recitation code review study groups

Goals

- ▶ Increase interactions during remote learning.
- ▶ Give stylistic code review and feedback (avoid coding to satisfy autograder).
- ▶ Boost recitation attendance and give structure to recitation sections.

Mechanics

- ▶ Teams of 5 students.
- ▶ Review and discuss code from previous assignment.
- ▶ As a team, present findings in 5 minute short summary.
- ▶ Recitation TAs have full discretion to award a portion of assignment grades for participating.

Table of contents

What this class is about: abstractions

- Low-level programming
- The memory hierarchy
- Digital logic

Class mechanics

- Class staff
- Accessing the class & resources
- Lecture and weekly short quizzes
- Programming assignments
- Recitation code review study groups

A New Golden Age for Computer Architecture

- History: computer architecture abstractions drove digital revolution
- Present: power constraints driving diverse computer architectures
- Future: post-Moore's Law computer architectures

A New Golden Age for Computer Architecture¹

Learning goal

At the end of this course, students should have the preliminary skills to design and evaluate solutions involving the computer software-hardware interface to address new problems.

¹<https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext>

History: computer architecture abstractions drove digital revolution



	1940s	1950s	1960s	1970s	1980s	1990s	2000s	2010s
Analog continuous-time computing	Analog computers for rocket and artillery controllers.	Analog computers for field problems. 	1 st transistorized analog computer. 	Analog-digital hybrid computers.	...			
Digital discrete-time computing	Turing's <i>Bomba</i> .	1 st transistorized digital computer.	Moore's law projection for transistor scaling.	Dennard's scaling for transistor power density.	VLSI democratized.			
	Stored program computer.	Microprogram ming.	Instruction set architecture.	Reduced instruction set computers.	Architecture abstraction milestones			
Transistor scaling and architectural abstractions drive digital revolution, make analog alternatives irrelevant								

Figure: Emerging Architectures for Humanity's Grand Challenges, Yipeng Huang

Present: power constraints driving diverse computer architectures

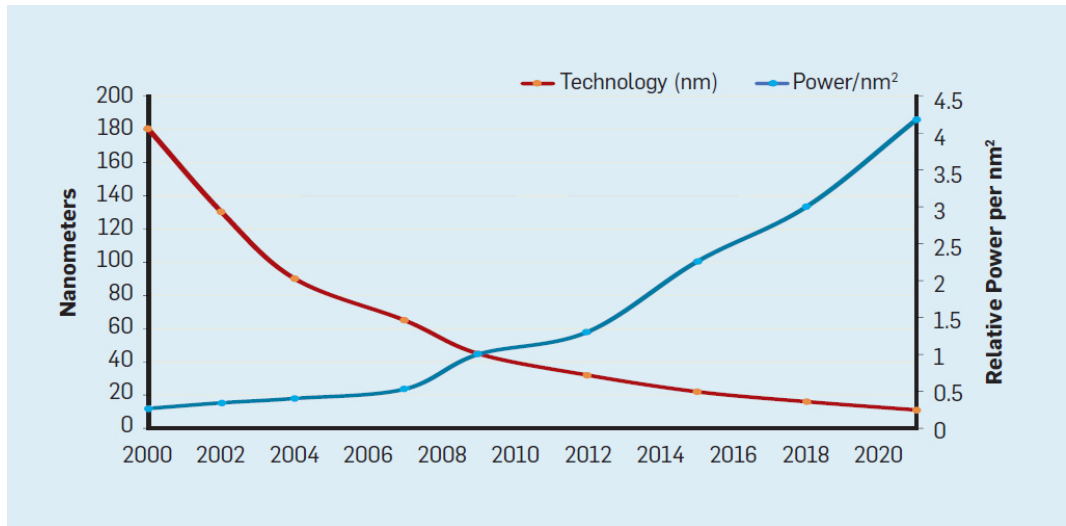


Figure: Credit: A New Golden Age for Computer Architecture

Present: power constraints driving diverse computer architectures

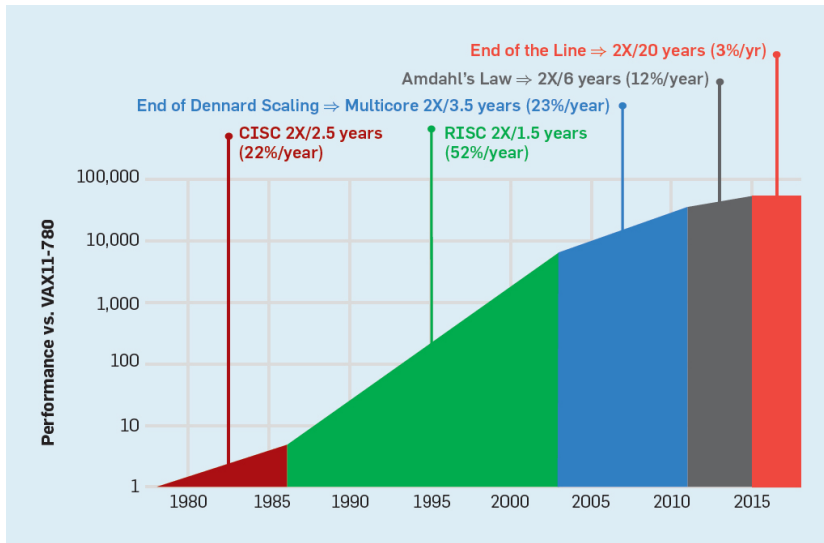


Figure: Credit: A New Golden Age for Computer Architecture

Present: power constraints driving diverse computer architectures



	1940s	1950s	1960s	1970s	1980s	1990s	2000s	2010s
Analog continuous-time computing	Analog computers for rocket and artillery controllers.	Analog computers for field problems. 	1 st transistorized analog computer. 	Analog-digital hybrid computers.	...			
Digital discrete-time computing	Turing's <i>Bomba</i> . Stored program computer.	1 st transistorized digital computer. Microprogramming.	Moore's law projection for transistor scaling. Instruction set architecture.	Dennard's scaling for transistor power density. Reduced instruction set computers.	VLSI democratized.	FPGAs introduced. GPUs introduced.	End of Dennard's scaling. CPUs go multicore. Nvidia introduces CUDA.	Cloud FPGAs: Microsoft Catapult. Amazon F1. ASICs: Google TPUs. DE Shaw Research Anton.
Transistor scaling and architectural abstractions drive digital revolution, make analog alternatives irrelevant					Scaling challenges drive heterogeneous architectures			

Figure: Emerging Architectures for Humanity's Grand Challenges, Yipeng Huang

Present: a rapidly evolving and influential field of study

Heterogeneity

Multicore CPUs, GPUs, FPGAs, ASICs, TPUs

Energy conservation

Laptop and phone battery life, datacenter energy consumption

Security

Spectre / Meltdown

Virtualization

Docker, Amazon AWS

Are abstractions always good? What are examples of deliberately breaking abstractions?

Python calling C binary

Breaking interpreted high level PL abstraction

Assembly code routines

Breaking structured programming abstraction

FPGAs

Breaking ISA abstraction

ASICs

Breaking von Neumann abstraction

Quoting Steve Jobs:

Everything around you that you call life was made up by people that were no smarter than you. And you can change it, you can influence it... Once you learn that, you'll never be the same again.

Future: post-Moore's Law computer architectures

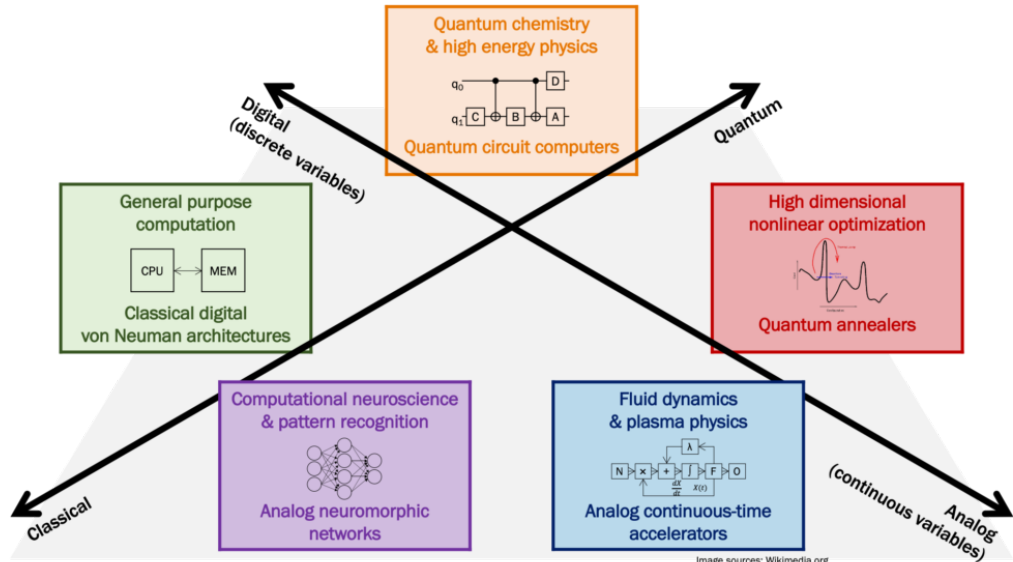


Image sources: Wikimedia.org

Future: post-Moore's Law computer architectures

Tutorial: differential equations →
accelerator configuration → analog solutions

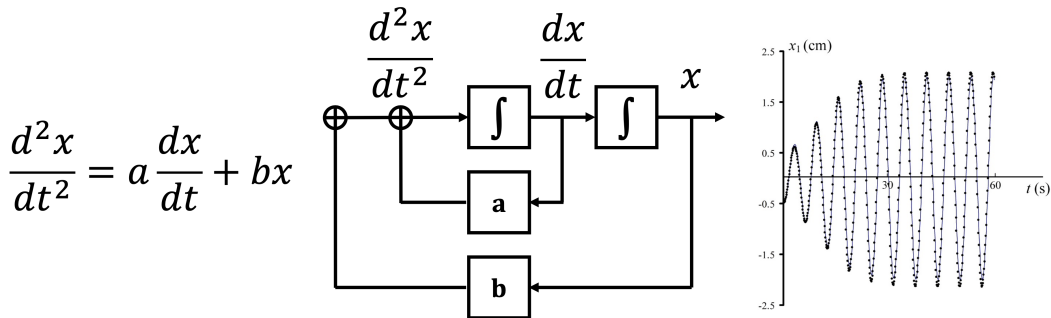
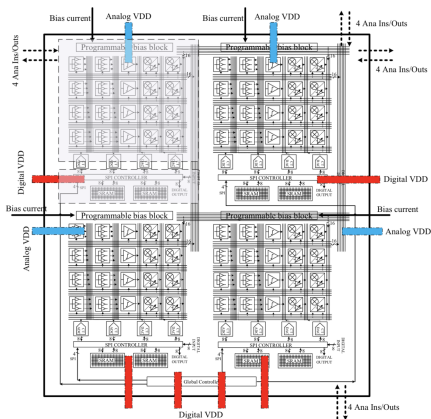


Figure: Emerging Architectures for Humanity's Grand Challenges, Yipeng Huang

Future: post-Moore's Law computer architectures

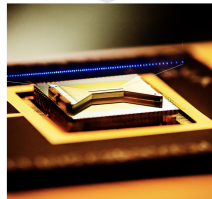
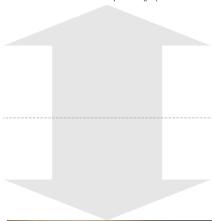
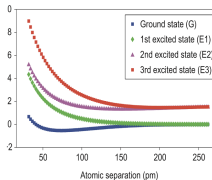
Columbia University prototype analog accelerator



Digital-to-analog converters (DACs)	8-bit, 16 total
Integrators	16 total
Variable-variable multipliers	32 total
Fanout current mirrors	32 total
Analog-to-digital converters (ADCs)	8-bit, 8 total
Programmable crossbar	All-to-all local, sparse global
Control & data interface logic	To config., calibrate, & handle exceptions

Figure: Emerging Architectures for Humanity's Grand Challenges, Yipeng Huang

Future: post-Moore's Law computer architectures



Awe-inspiring quantum algorithms

Chemistry simulations from governing equations

Quantum computers as quantum mechanics simulator

Shor's algorithm for factoring integers

Surpasses any known classical algorithm

Hundreds more near-term and far-future algorithms

QuantumAlgorithmZoo.org

My work in bridging quantum software-hardware gap

Assertions for quantum program patterns and bugs

ISCA 2019.
IBM Qiskit open-source contribution.

Graphical model inference for quantum program simulation and analysis

Analog computing support for quantum control & measurement

Now-viable quantum prototypes

Superconducting qubits

IBM, Google, [Rigetti](#), ...

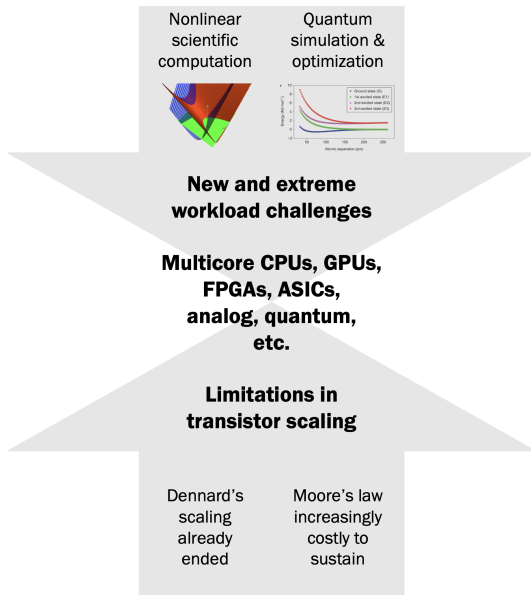
Trapped ion qubits

[IonQ](#), UMD, ...

Dozens of candidate qubit technologies

May yet surpass current leaders in capacity and reliability

Future: post-Moore's Law computer architectures



Open challenges in emerging architectures:

Problem abstractions

- How do you accurately solve big problems?

Programming abstractions

- Can you borrow ideas from conventional computing?

Architecture abstractions

- How to interface with the unconventional hardware?