

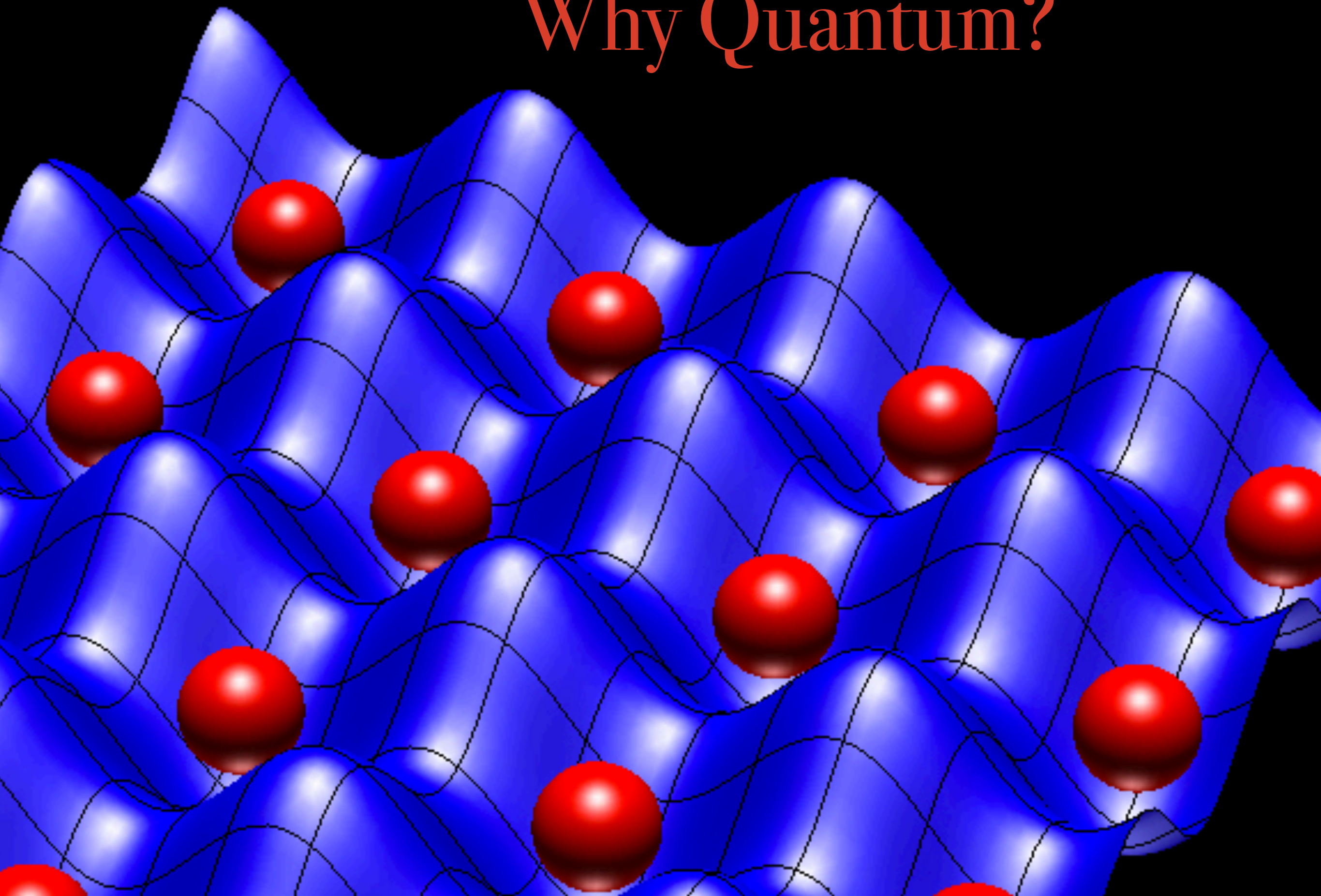
Quantum Computing for the Common Man



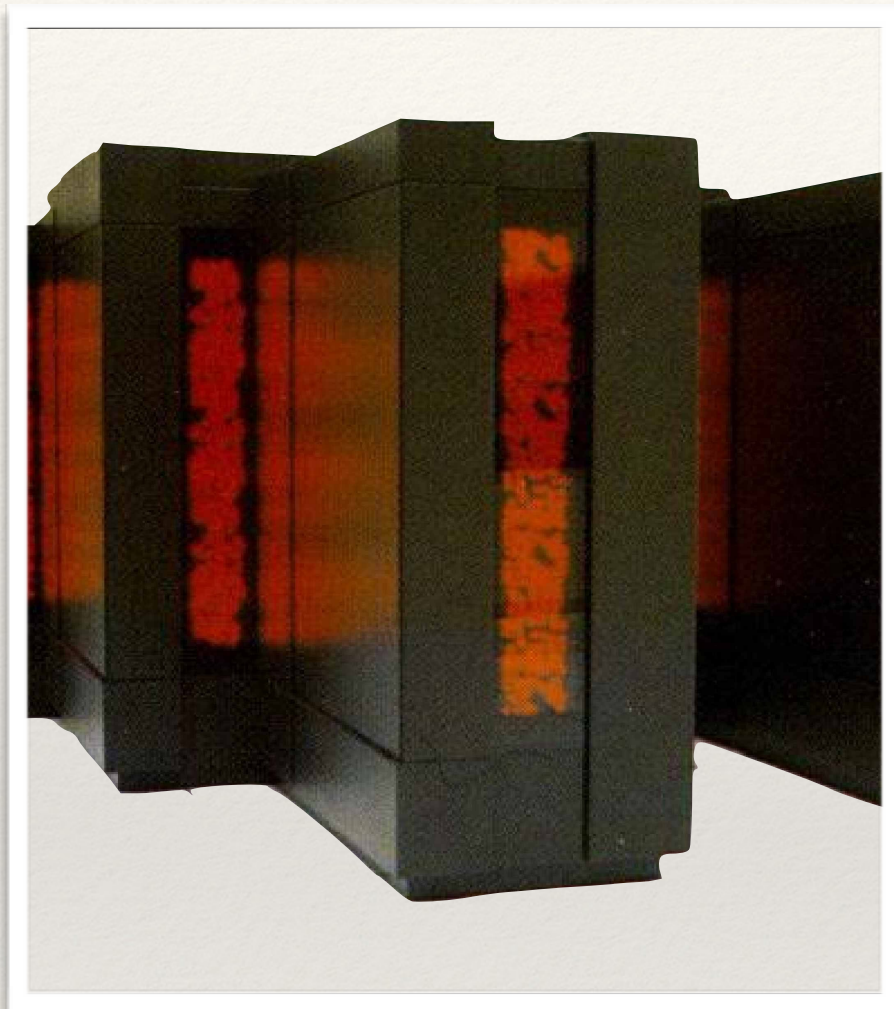
William S. Moses

8.370 Project Presentations
May 7, 2018

Why Quantum?



Moore's Law



Connection Machine CM-5

- 60 GFLOPS on LINPACK
- \$47 million in 1993



Apple 13" MacBook Pro

- 70 GFLOPS on LINPACK
- \$1500 in 2015

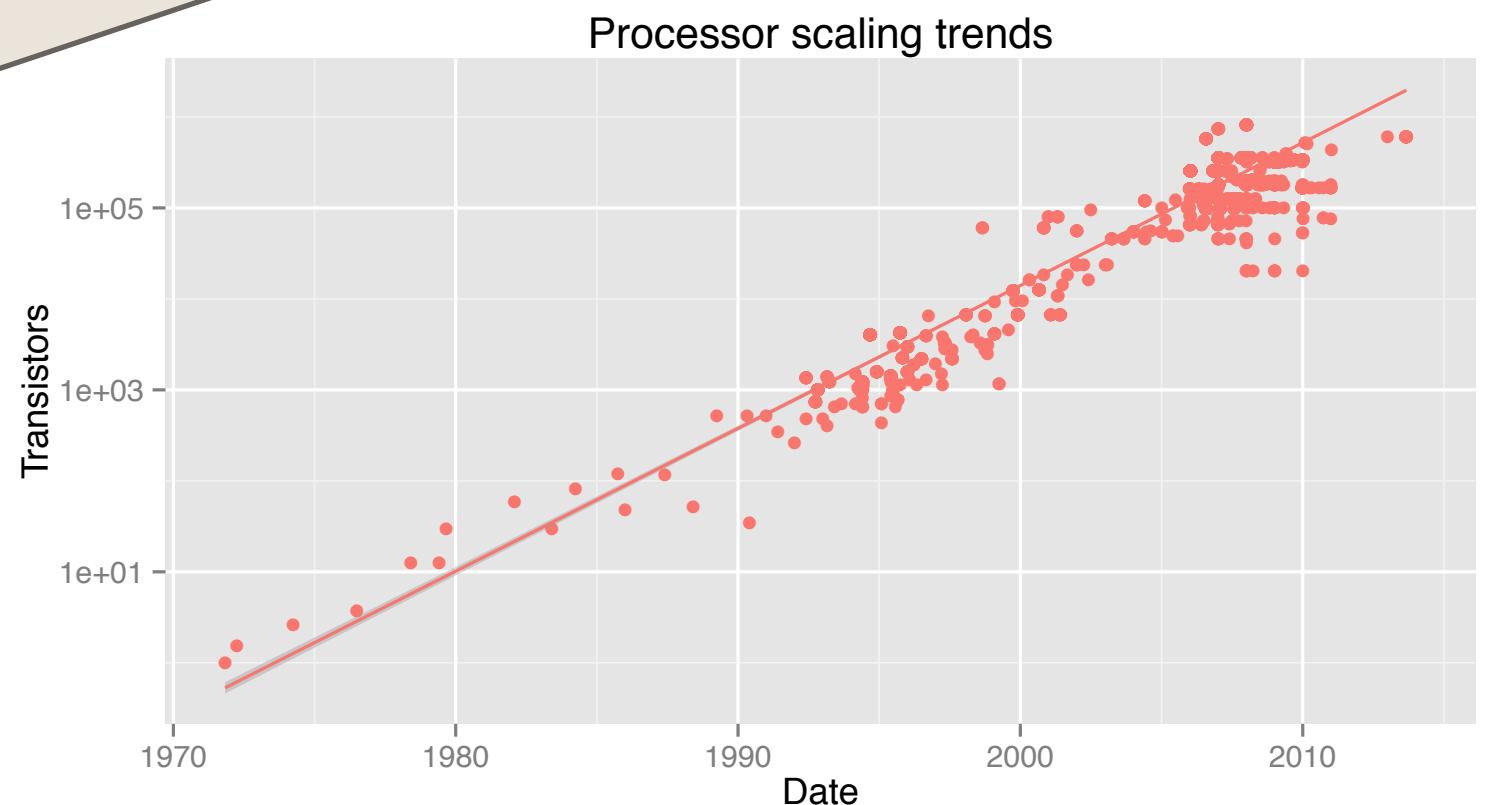
Moore's Law [M65, M75]

“The complexity for minimum component costs has increased at a rate of roughly a factor of two per year.” [M65]



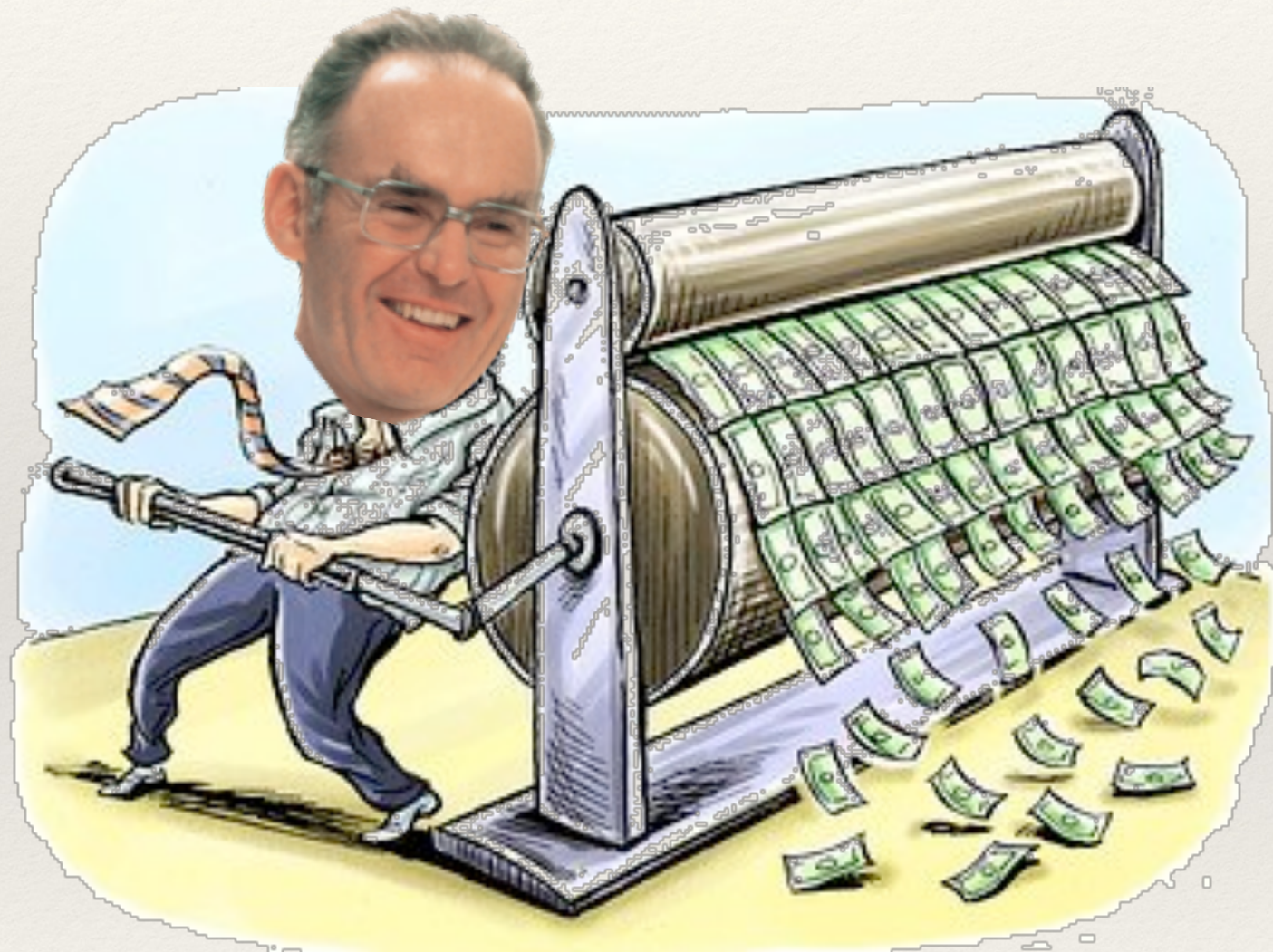
Moore

“The new slope might approximate a doubling every two years, rather than every year, by the end of the decade.” [M75]



50-Year Impact of Moore's Law

More transistors means cheaper computing.



Moore's Law is a printing press for processor cycles.

50-Year Impact of Moore's Law

More transistors mean cheaper computing.



Moore's Law is a printing press for processor cycles.

Moore's Law Will End

Robert Colwell, chief architect for the Intel Pentium Pro, Pentium II, Pentium III, and Pentium 4 processors, and former director of the Microsystems Technology Office at DARPA, said in 2013:



Colwell

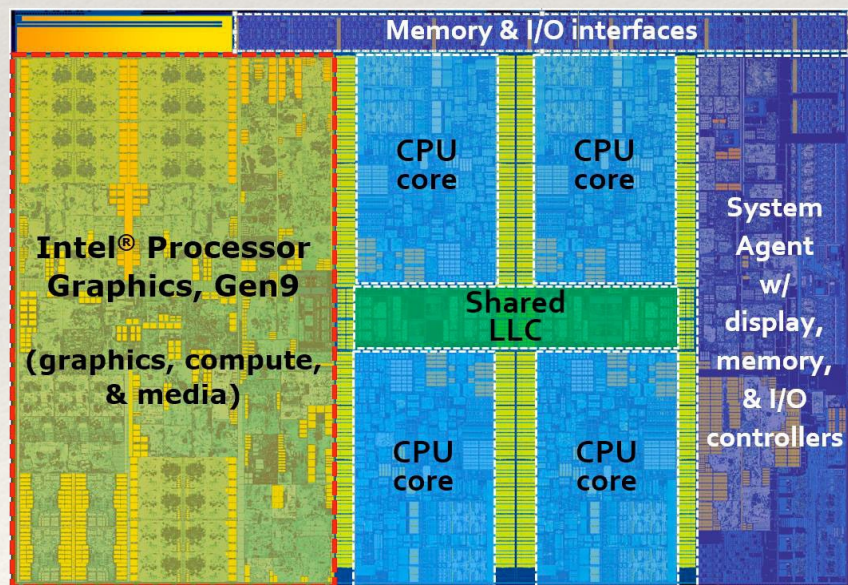
“For planning horizons, I pick 2020 as the earliest date where I think we could call [Moore's Law] dead.”

“You can talk me into 2022.”

Why Must It End Now?

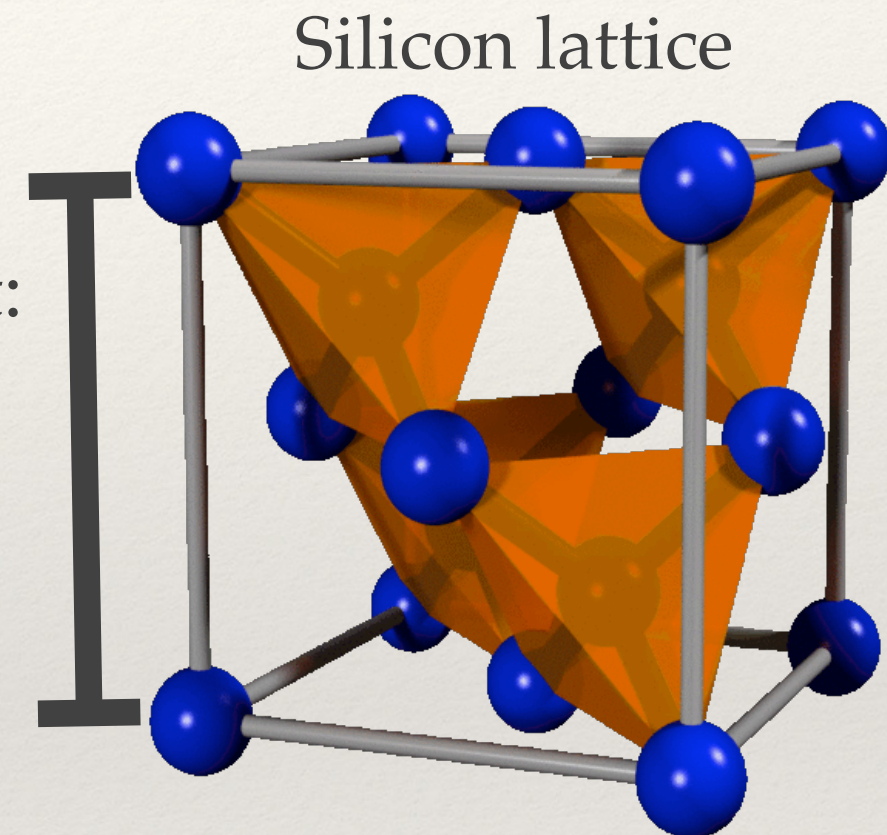
We're running out of atoms.

Intel Skylake processor, 2015



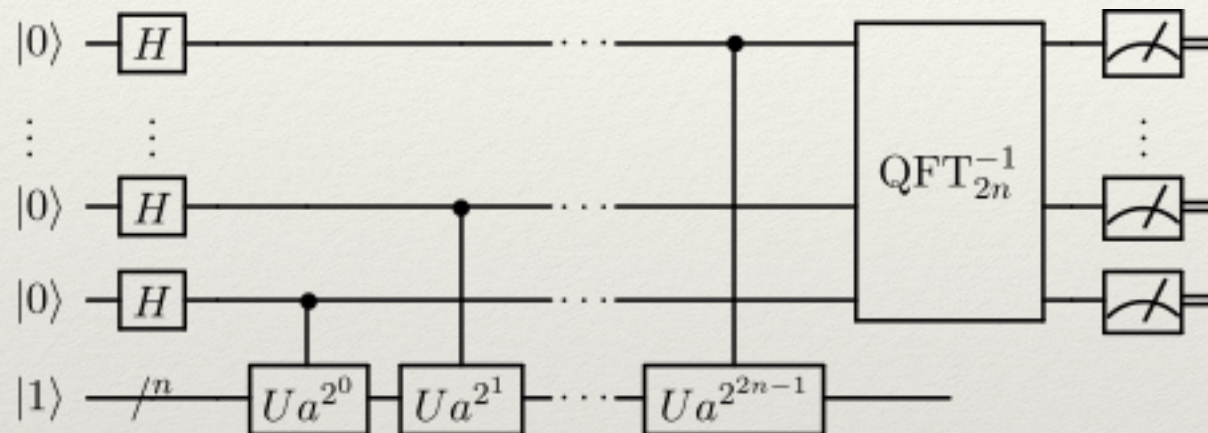
14 nanometer transistors

Silicon lattice constant:
0.543 nanometers
(5.43 angstroms)

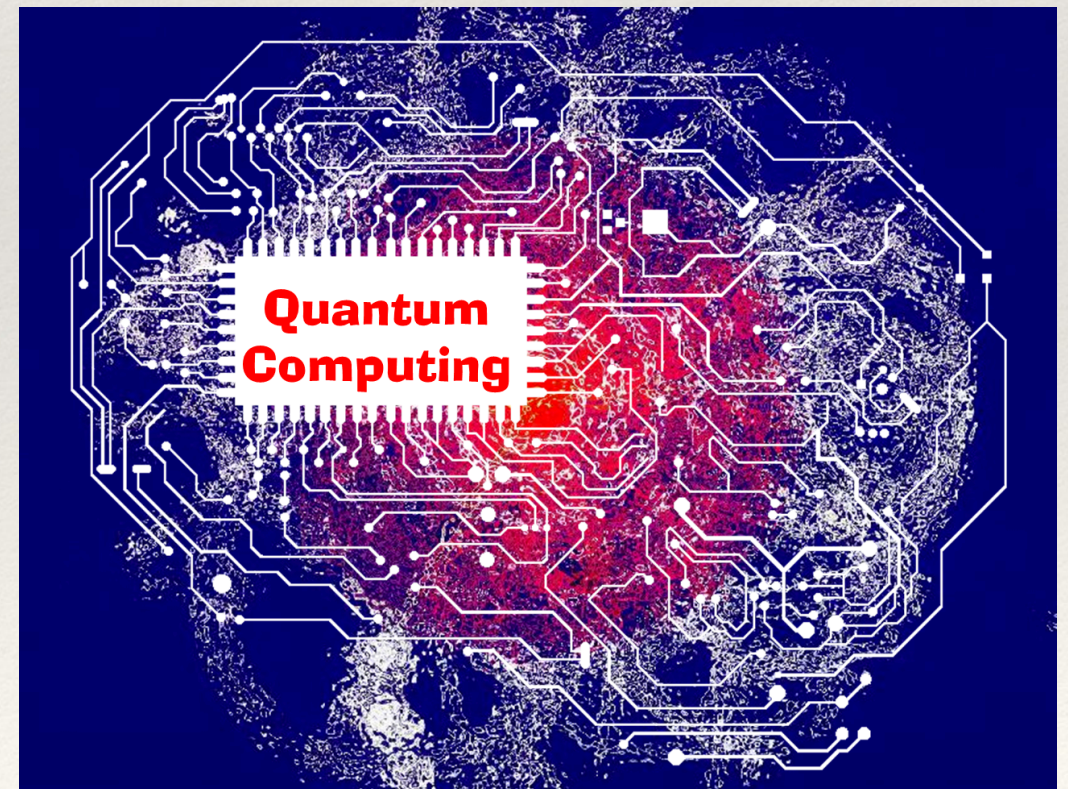


Transistors are now
25 atoms wide.

Won't Quantum Computing Save The Day?

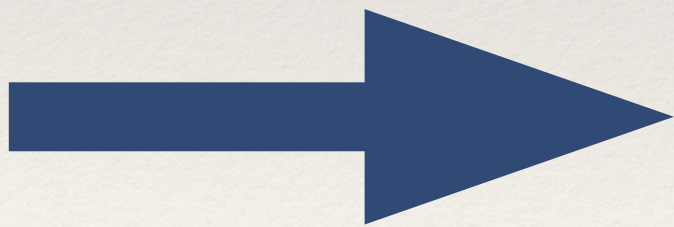


$$O\left(\sqrt{N}\right)$$



Not Quite Yet...

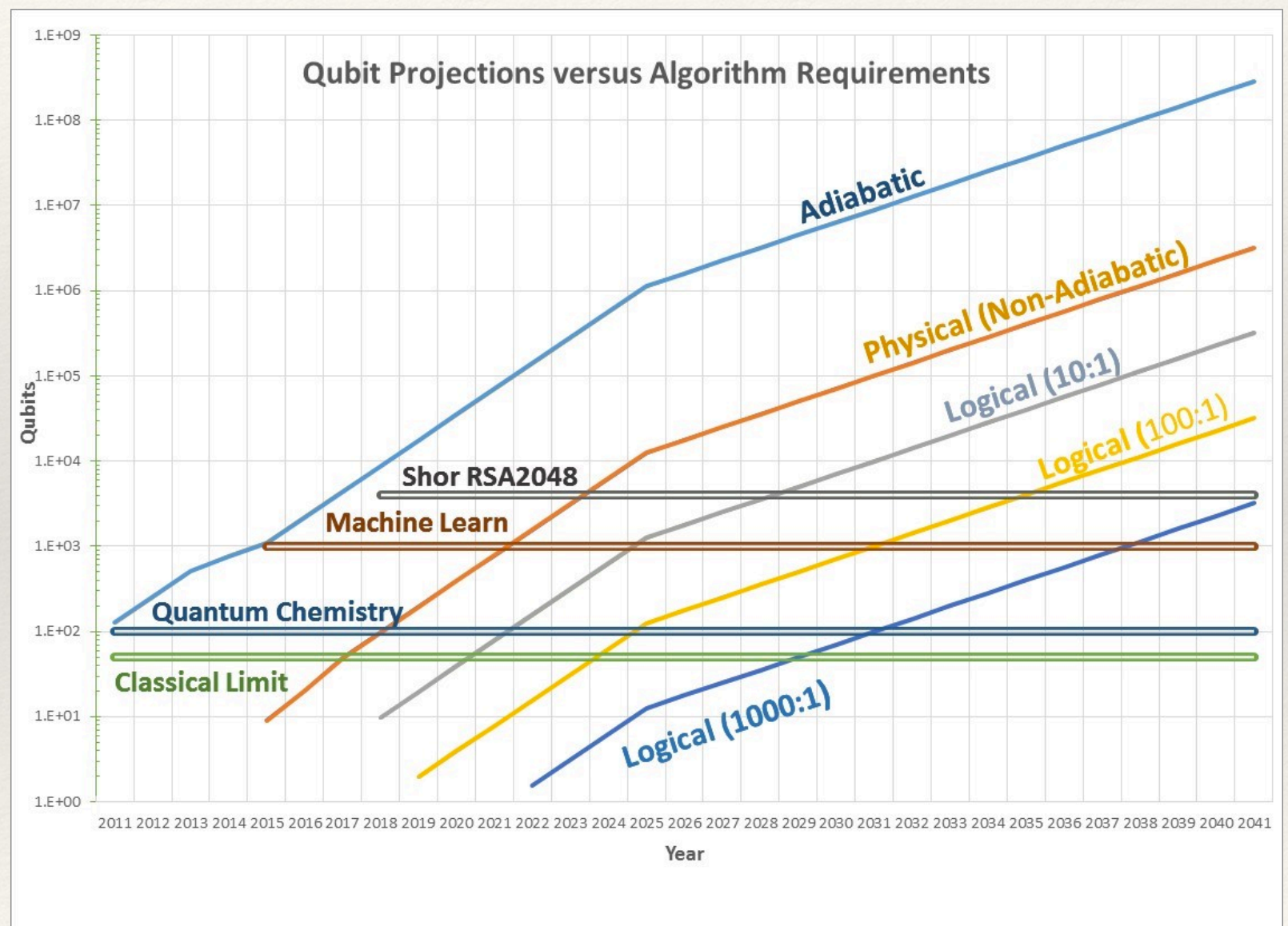
- ❖ Not enough error-corrected qubits to run “heavy” algorithms (though still plenty for doing interesting things)
- ❖ Lack of infrastructure for building and optimizing quantum programs



Hard to use quantum computing
for modern-day speedups

Qubit Counts [QCR16]

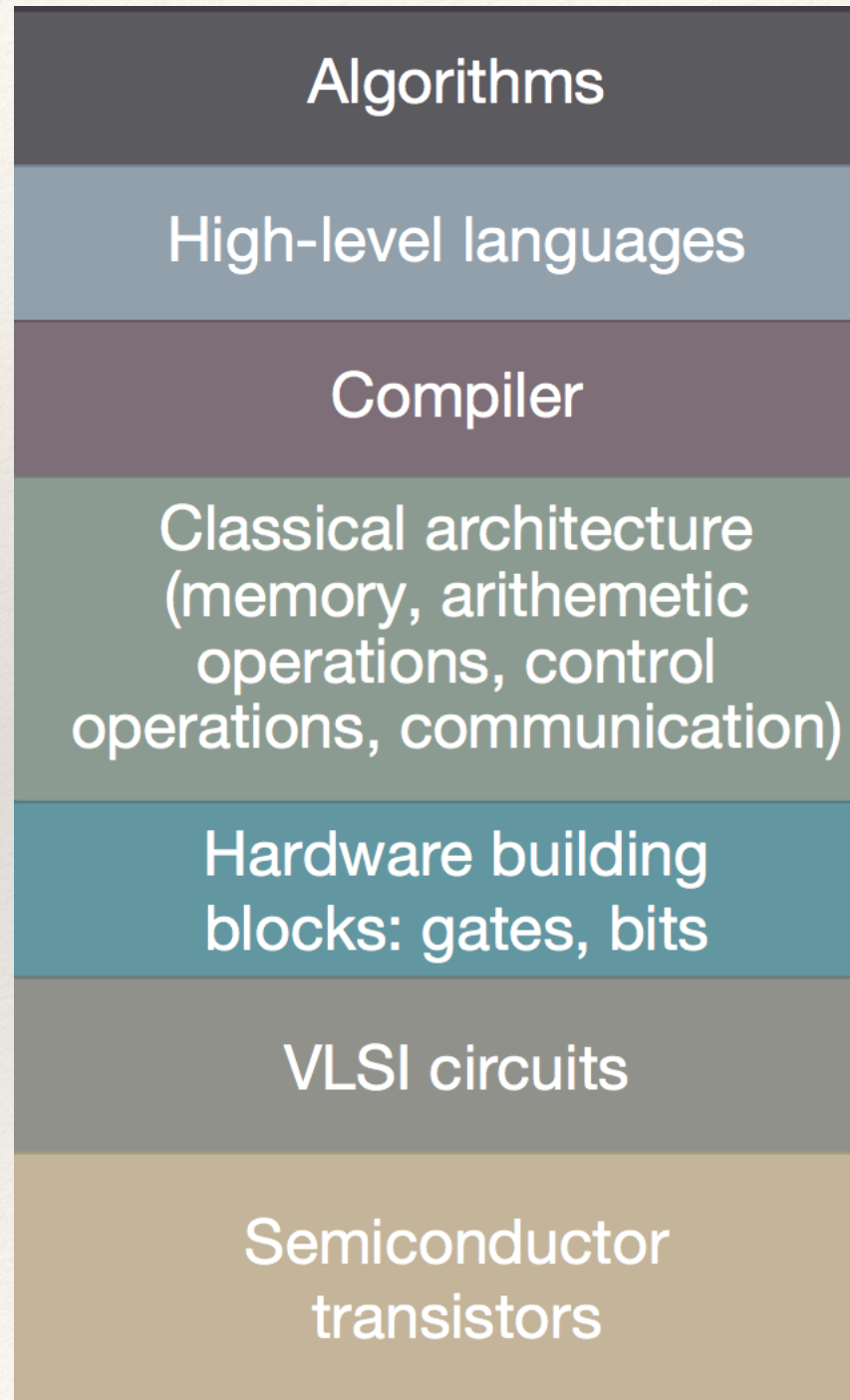
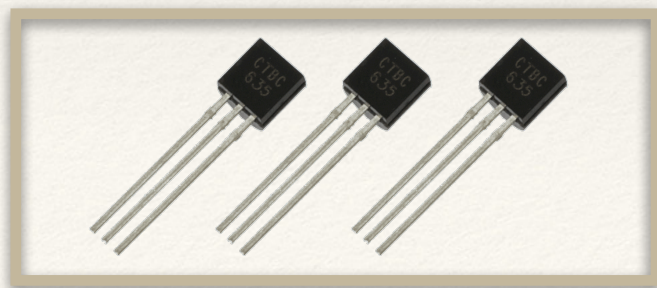
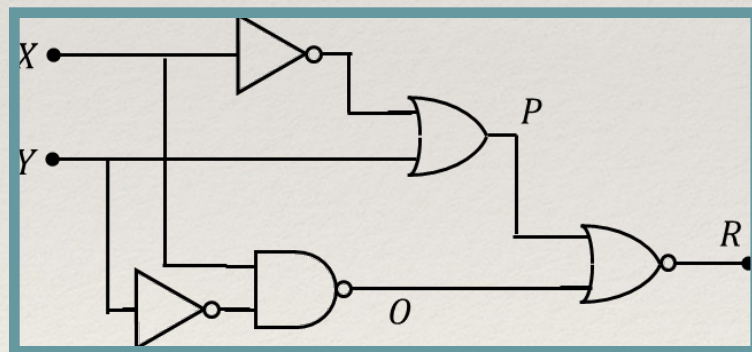
- ❖ The best we can hope from quantum computing in the near future are clever algorithms on relatively few qubits



Classical Infrastructure

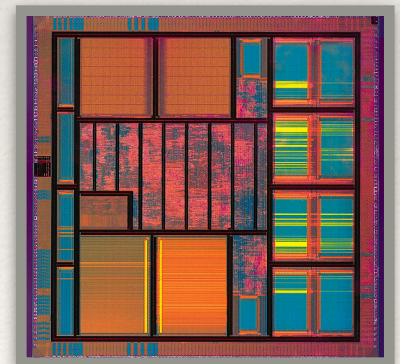
```
def normalize(out, in):  
    for i in range(len(in)):  
        out[i] = in[i] / norm(in)
```

```
def normalize(out, in):  
    n = norm(in)  
    for i in range(len(in)):  
        out[i] = in[i] / n
```



Normalize Vector

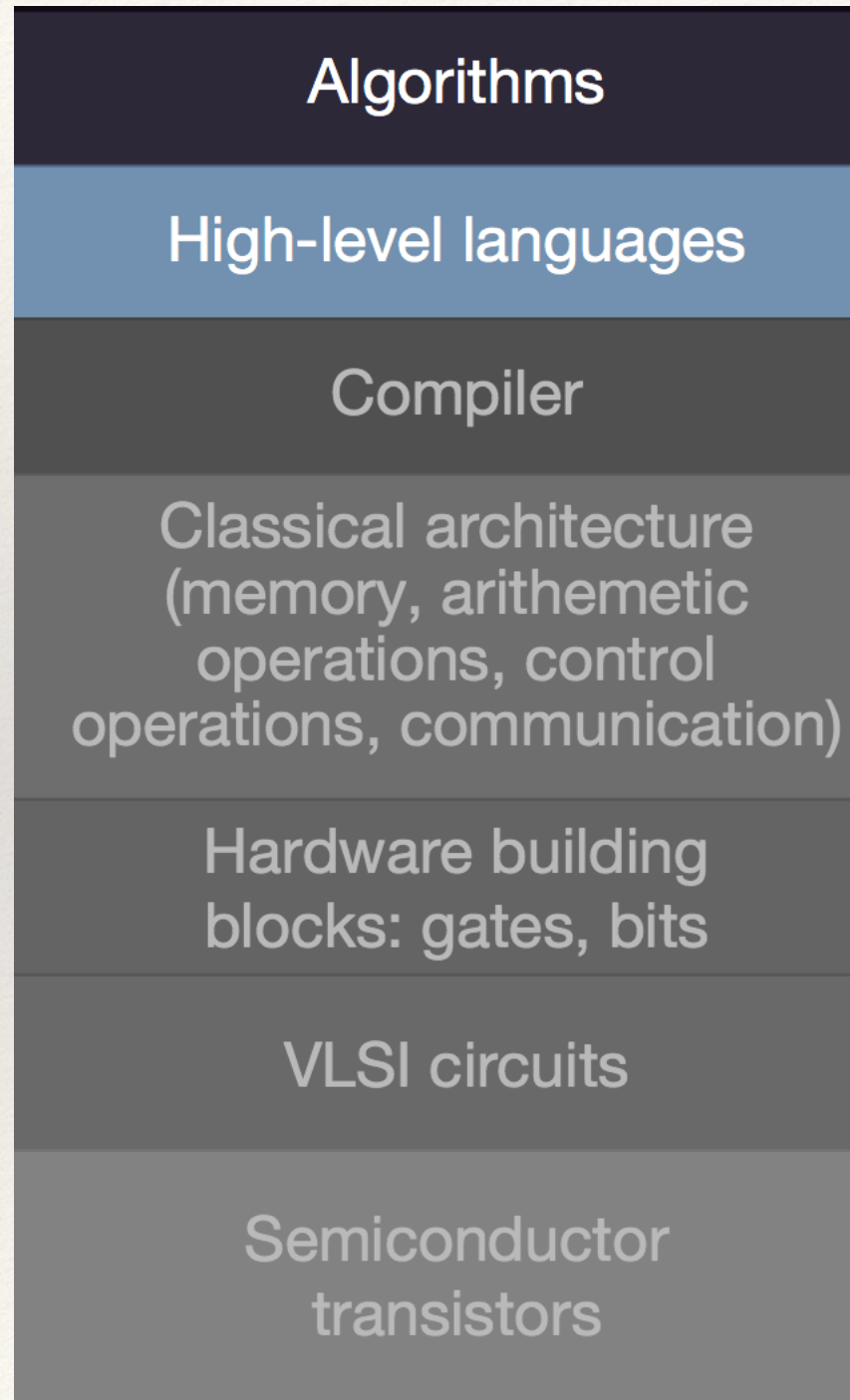
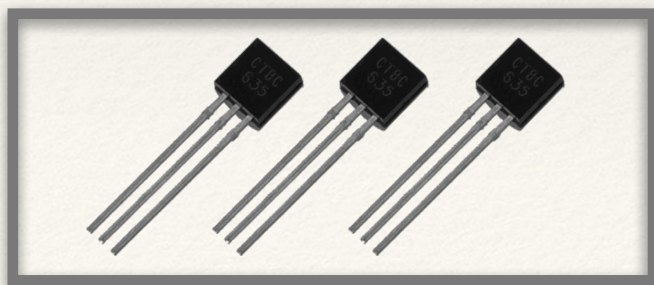
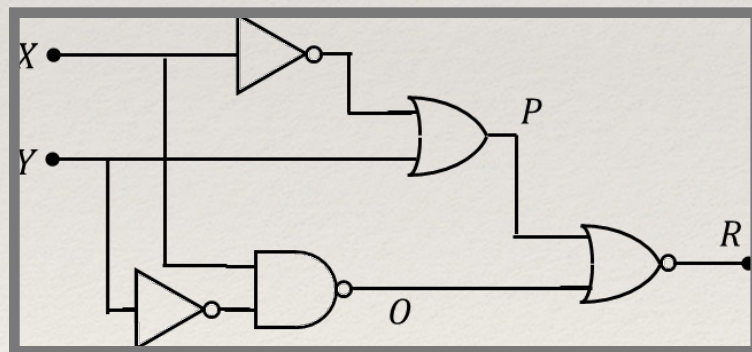
```
loop:  
    %divq %rax, %r10  
    %inc %rbx  
    %jne loop, %rbx, %r9
```



Classical Infrastructure

```
def normalize(out, in):  
    for i in range(len(in)):  
        out[i] = in[i] / norm(in)
```

```
def normalize(out, in):  
    n = norm(in)  
    for i in range(len(in)):  
        out[i] = in[i] / n
```

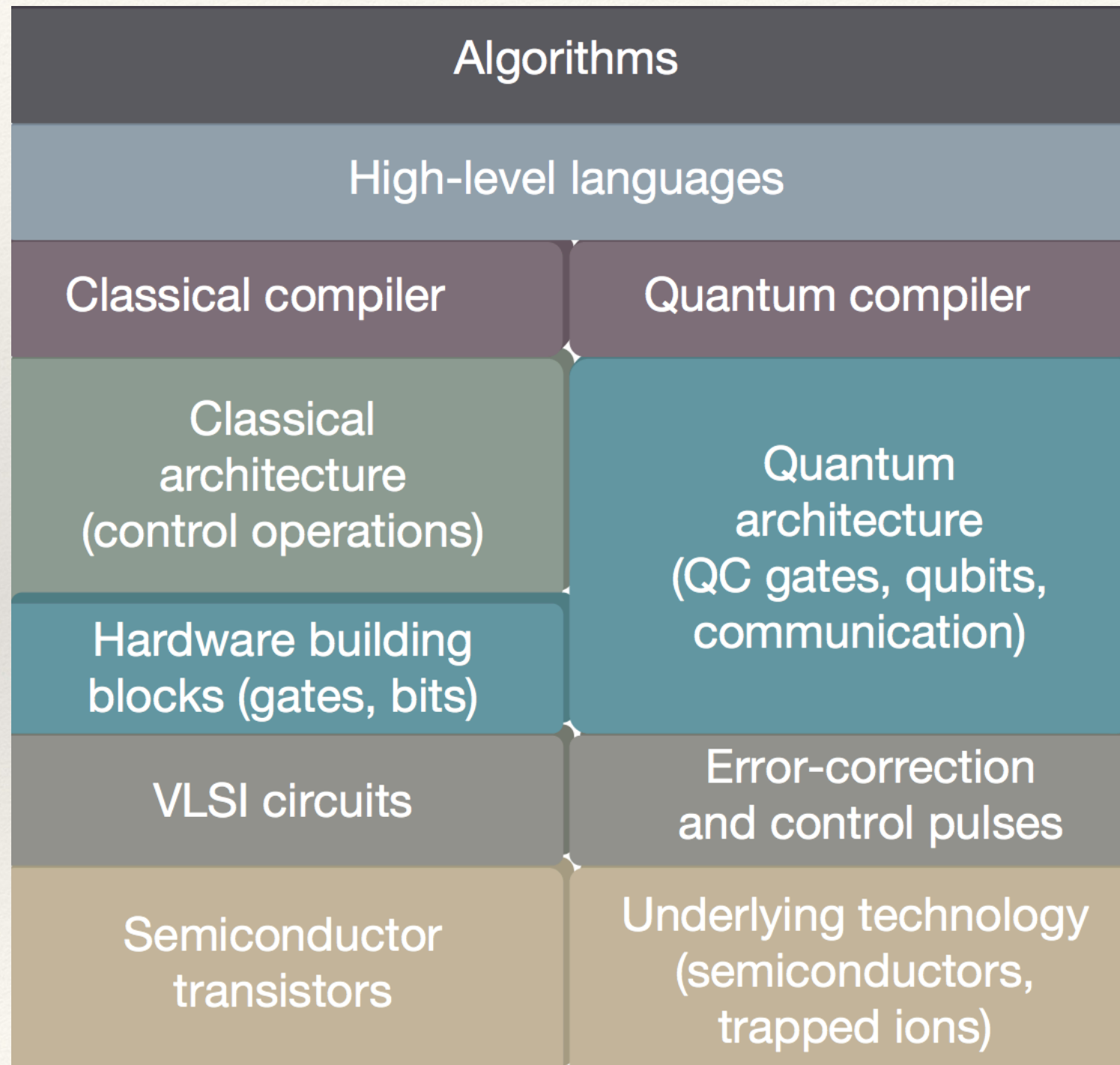


Normalize Vector

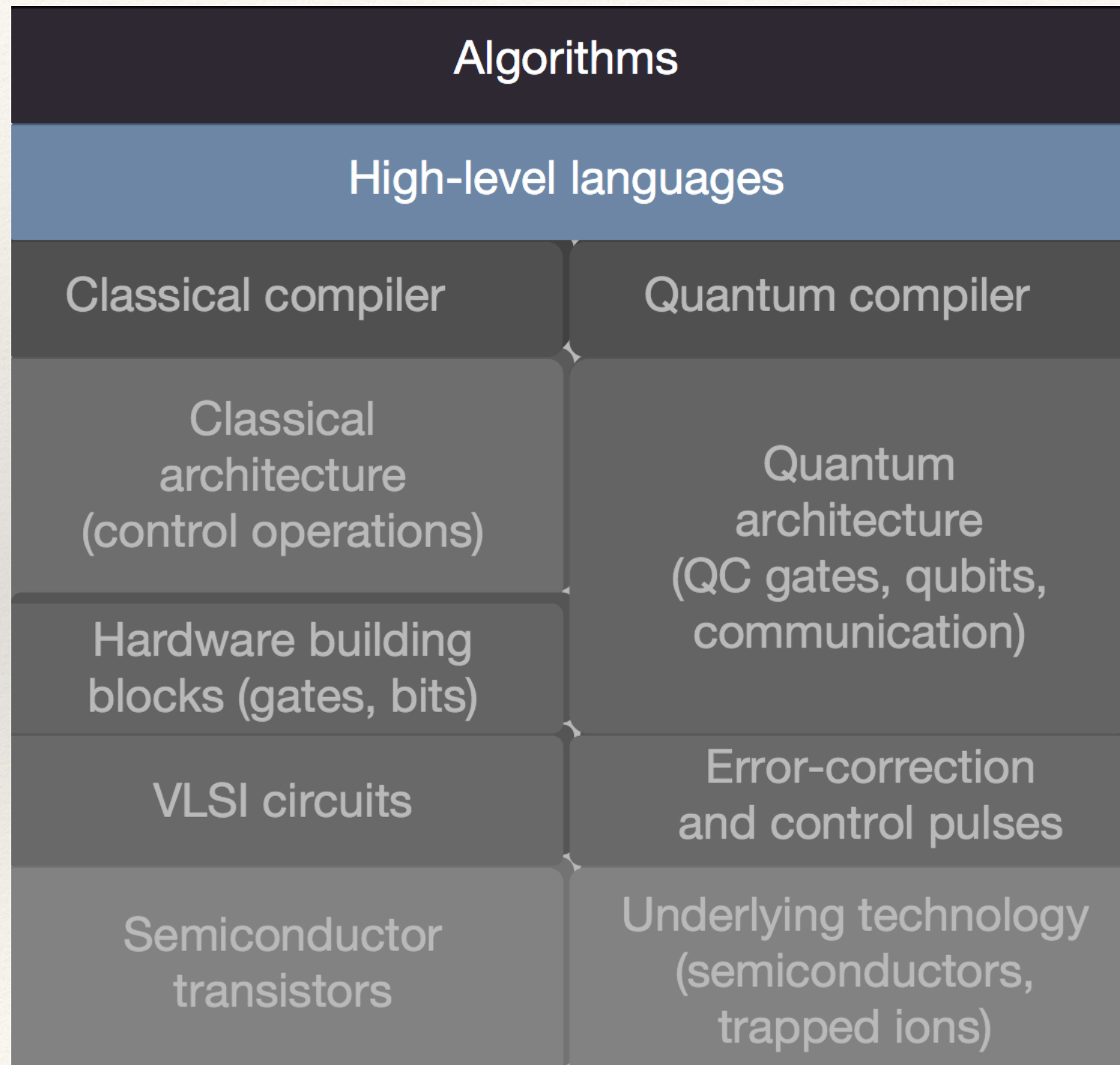
```
loop:  
    %divq %rax, %r10  
    %inc %rbx  
    %jne loop, %rbx, %r9
```



Quantum Infrastructure [CFM17]



Quantum Infrastructure [CFM17]



In Other Words,

MITx Decomposition exercise: 3-qubit x Billy

Secure | <https://lms...>

$U =$ [H(2),X(2),H(2),CNOT(1,2),Z(2),H(2),S(1),S(1),S(1),H(1),CNOT(0,1),H(1)] ✓

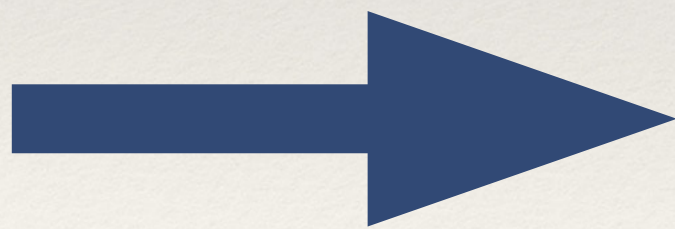
Graphical rendition of your circuit :

```
graph LR
    Q2[Qubit 2] -- H --> C1((CNOT))
    Q1[Qubit 1] -- S --> S2[S]
    S2 -- S --> H1[H]
    H1 -- CNOT --> Q2
    Q2 -- H --> Z[Z]
    Z -- H --> C2((CNOT))
    Q1 -- S --> C2
    C2 -- H --> Q1
    Q1 -- H --> C3((CNOT))
    Q0[Qubit 0] -- CNOT --> C3
```

No need to build an optimized
cryptosystem with MITx circuit builder

Conflicting Problems

- ❖ We need to build high-level primitives to help people develop and code quantum algorithms
- ❖ Quantum programs need to make clever use of few qubits to run at all (seemingly requires low-level knowledge of the quantum stack)



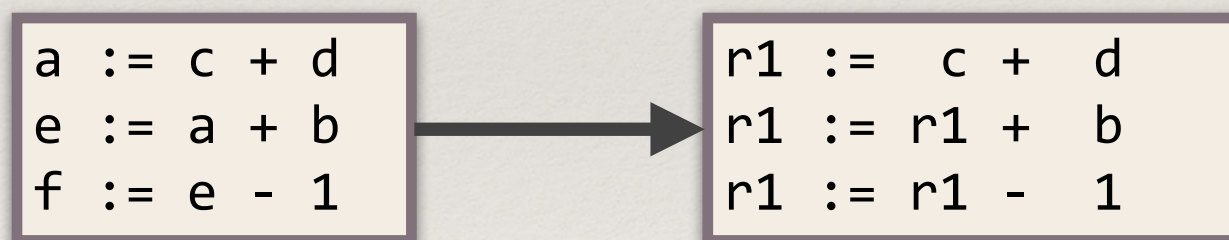
Have a (classical) computer
optimize our program for us

Rest of Talk

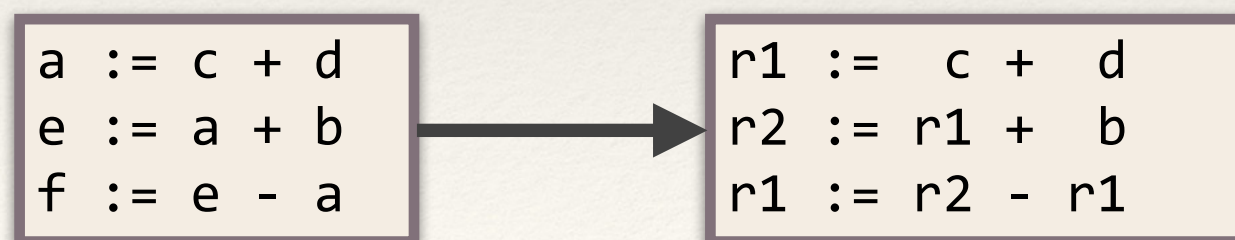
- ❖ Three quantum optimization success stories
 - ❖ Quantum Register Allocation [leveraging classical CS]
 - ❖ Quantum Time Evolution [leveraging physics]
 - ❖ Quantum Scheduling
- ❖ Glimpse at some original research

(Quantum) Register Allocation

- ❖ Have few (physical or logical) qubits
- ❖ Optimize an input program on n qubits to use as few physical qubits as possible.
- ❖ Key insight: when a qubit is dead, it can be reused



Reducible to one register

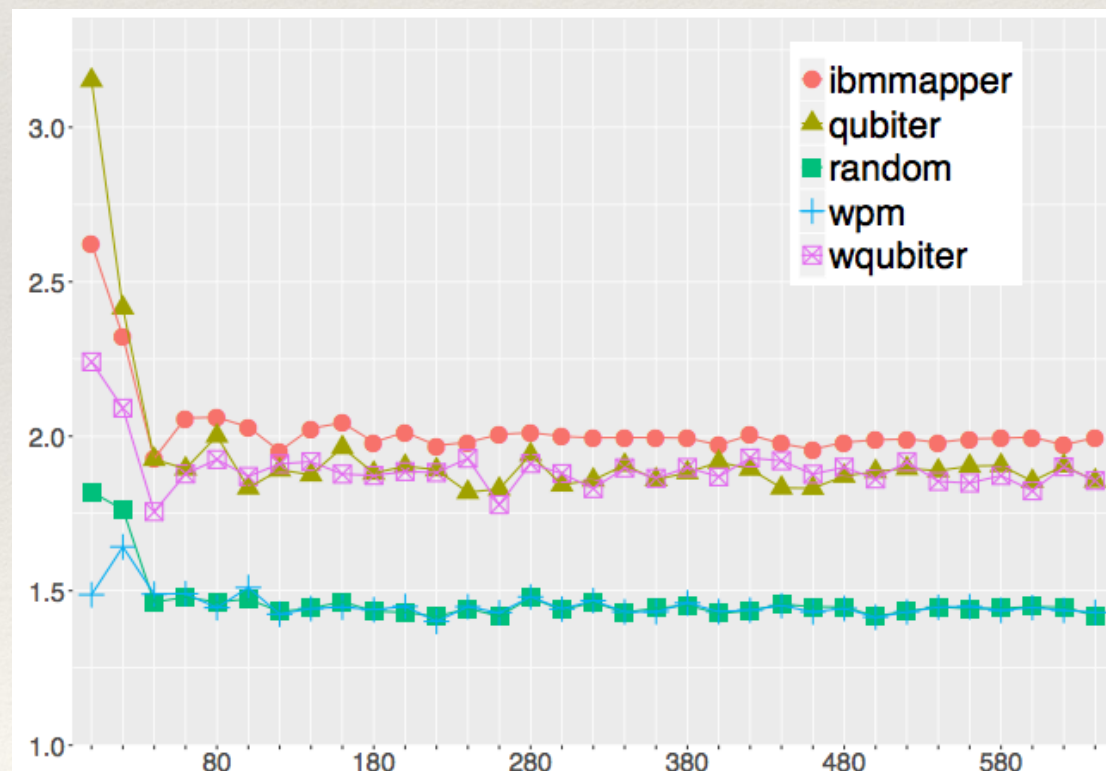


Requires two registers

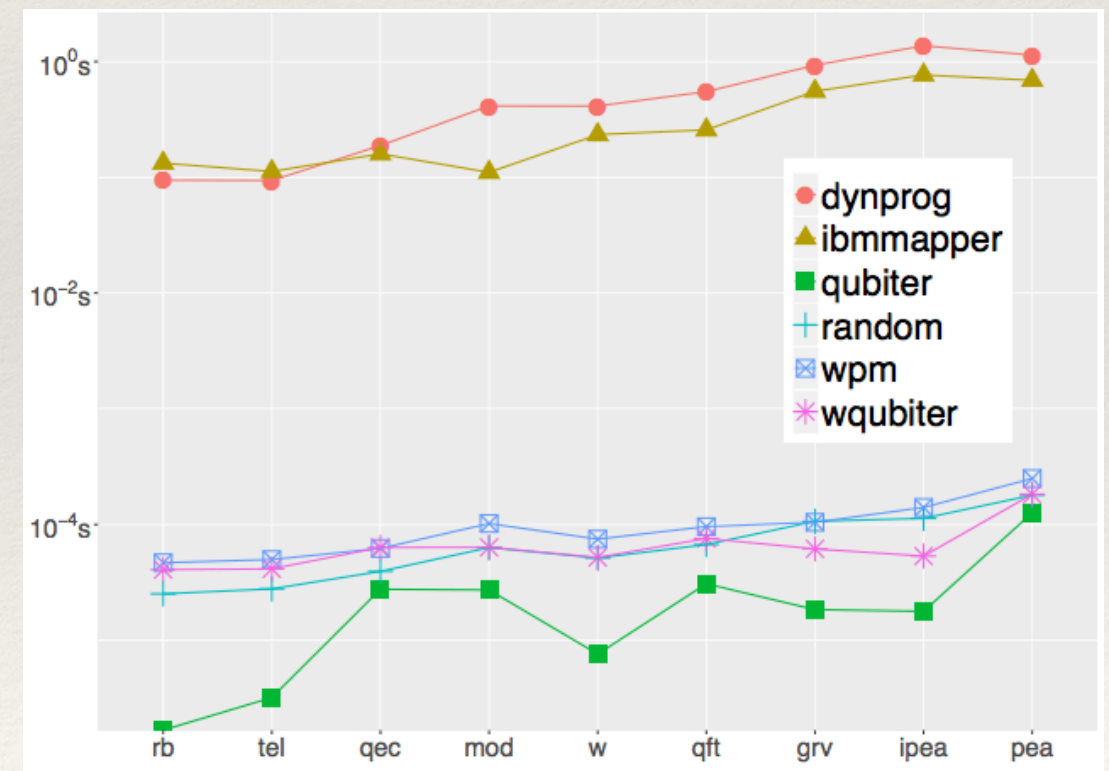
(Quantum) Register Allocation

[SFCQ18]

- ❖ Classically equivalent to finding optimal graph coloring on registers (NP-complete).
- ❖ For classical compilers, non-optimal heuristics are used.



Cost (~required qubits) relative to optimal

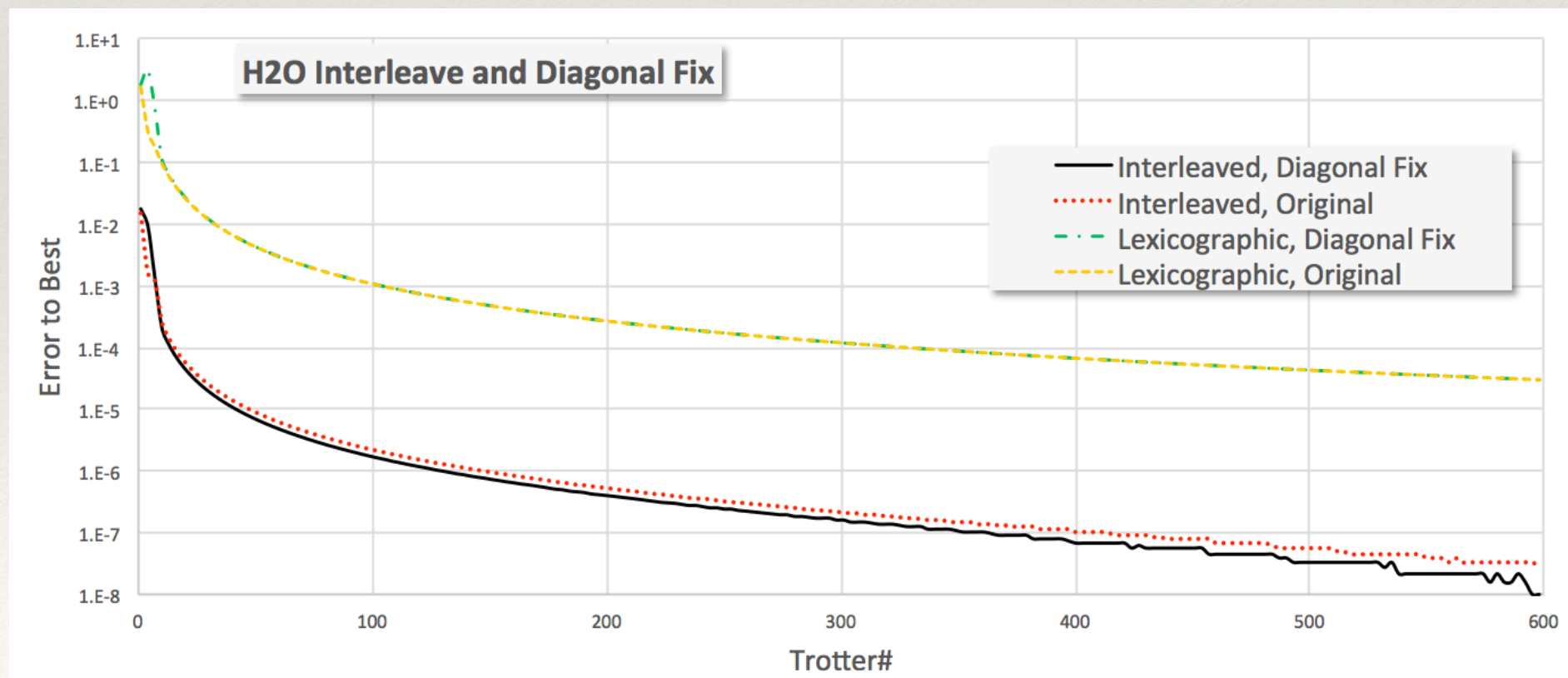


Time to solve for allocation
(dynprog is optimal)

Chemistry Time Evolution [HWBT14]

$$H = \sum_{pq} t_{pq} c_p^\dagger c_q + \frac{1}{2} \sum_{pqrs} V_{pqrs} c_p^\dagger c_q^\dagger c_r c_s$$

- ❖ Quantum chemistry problem has N non-commuting terms
- ❖ Similar to PSET, performing each term on a small time step.
- ❖ By taking advantage of certain properties present in the terms of the Hamiltonian, can choose a better (interleaved) ordering. [Additional, less dramatic “fix” via perturbation theory].



Quantum Scheduling [HPJHKBFCM15]

```
def bit_H(x, y, i):  
    H(x[i], y[i], i)  
  
for i in range(N):  
    X(x[i], y[i])  
  
for i in range(N):  
    bit_H(x, y, i)
```

```
X(x[0], y[0])  
...  
X(x[N-1], y[N-1])  
  
bit_H(x[0], y[0])  
...  
bit_H(x[N-1], y[N-1])
```

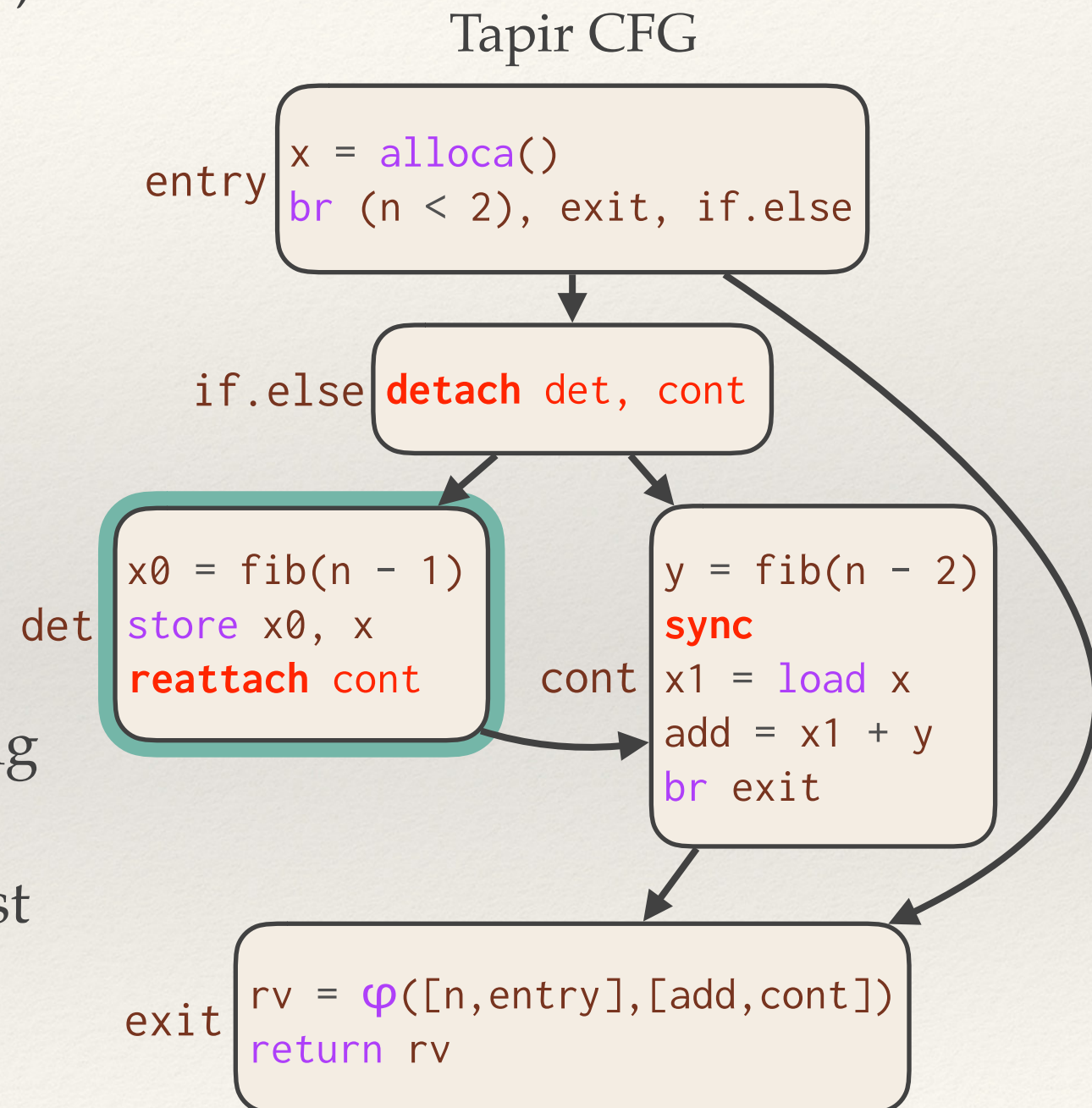
- ❖ “Quantum” programmer specifies gates (either directly or via library functions) to execute in imperative fashion.
- ❖ “Classical” control flow (loops, if, etc) separates any quantum operations. This example uses $2N$ quantum cycles.
- ❖ Unrolling all loops, the runtime can attempt (and here succeed) in detecting the parallelism and coalesce the X operations to one quantum cycle — however, the bit_H operations still take N quantum cycles, as hindered by classical function call.
- ❖ Deciding when to do such inlining is a difficult problem, as it can allow for dramatic quantum performance gains, however, at the risk of exponentially increasing code size.

Quantum Scheduling [HPJHKBFCM15]

- ❖ In 2015, Heckey et al introduced a set of optimization and analysis passes on top of a serial classical compiler (LLVM) to perform some of this optimization.
- ❖ In doing so they also introduced the Longest Path-First Scheduling (LPFS) algorithm to reduce communication overhead and increase parallelism (such as in the matter presented in prior slide).
- ❖ They found a 3%-308% improvement.

New Research in Q. Scheduling

- ❖ In 2017, Schardl, Moses, and Leiserson, were able to extend classical compiler infrastructure to natively support (optimize and analyze) *parallel* programs through the use of new instructions called Tapir, to much benefit.
- ❖ There has been success in using the polyhedral model (a mechanism for greatly optimizing code by considering it as an iteration on a lattice) for optimizing schedules with a given cost model or by simply auto-tuning [VZTGDMVAC18].



New Research in Q. Scheduling

- ❖ In 2017, Schardl, Moses, and Leiserson, were able to extend classical compiler infrastructure to natively support (optimize and analyze) *parallel* programs through the use of new instructions called Tapir, to much benefit.
- ❖ There has been success in using the polyhedral model (a mechanism for greatly optimizing code by considering it as an iteration on a lattice) for optimizing schedules with a given cost model or by simply auto-tuning [VZTGDMVAC18].

$$\text{Domain} \left[\begin{array}{l} \{S(i, j) \mid 0 \leq i < N \wedge 0 \leq j < K\} \\ \{T(i, j, k) \mid 0 \leq i < N \\ \wedge 0 \leq j < K \wedge 0 \leq k < M\} \end{array} \right.$$

Sequence

Filter $\{S(i, j)\}$

Band $\{S(i, j) \rightarrow (i, j)\}$

Filter $\{T(i, j, k)\}$

Band $\{T(i, j, k) \rightarrow (i, j, k)\}$



Fuse S, T loops

$$\text{Domain} \left[\begin{array}{l} \{S(i, j) \mid 0 \leq i < N \wedge 0 \leq j < K\} \\ \{T(i, j, k) \mid 0 \leq i < N \wedge 0 \leq j < K \wedge 0 \leq k < M\} \end{array} \right.$$

Band

$$\left[\begin{array}{l} \{S(i, j) \rightarrow (i, j)\} \\ \{T(i, j, k) \rightarrow (i, j)\} \end{array} \right.$$

Sequence

Filter $\{S(i, j)\}$

Filter $\{T(i, j, k)\}$


Band $\{T(i, j, k) \rightarrow (k)\}$


New Research in Q. Scheduling

- ❖ As some novel research for the class, I am presently working on using a combination of Tapir (to allow for efficient representation and optimization of the parallel aspects of the quantum operations) along with the use of the polyhedral model to optimize scheduling.
- ❖ Presently integrating of Tapir/Polly with ScaffCC (Quantum based LLVM).

 wsmoses / Tapir-LLVM

 Code

 Issues 11

 Pull requests 4

 Projects

Tapir extension to LLVM for optimizing Parallel Programs

 epiqc / ScaffCC

 Code

 Issues 8

 Pull requests 1

 Projects 0

Compilation, analysis and optimization framework for the Scaff

Conclusions

- ❖ We need to develop new technologies to help mitigate the effects of Moore's Law ending
- ❖ Quantum computing, while promising, isn't yet ready for prime time.
- ❖ We don't have enough qubits to run large algorithms, yet require high-level abstractions to allow regular programmers to use it.
- ❖ It is crucial to quantum computing's success in the near future to develop systems that are able to automatically make quantum programs more efficient (and thus help fit more abstract, yet less efficient quantum programs into current hardware).
- ❖ There have been a number of success stories applying work from across both classical CS and physics to resolving this issue.
- ❖ Yet, there is still a lot of work to be done!