Programming models for hybrid HPC-QPU applications: the deeper issues

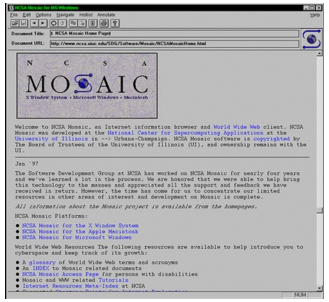
Santiago Núñez-Corrales, PhD

Quantum Lead Research Scientist, National Center for Supercomputing Applications
Faculty Affiliate, Illinois Quantum Information Science and Technology Center
Core Faculty, Program on Arms Control & Domestic and International Security
University of Illinois Urbana-Champaign

National Center for Supercomputing Applications
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

National Center for Supercomputing Applications

Mission: Bring people, computing and data together to benefit society

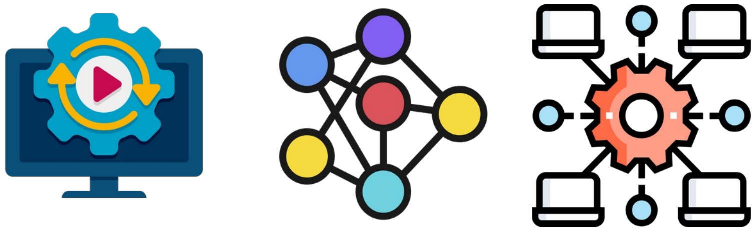


NCSA's mission in quantum computing

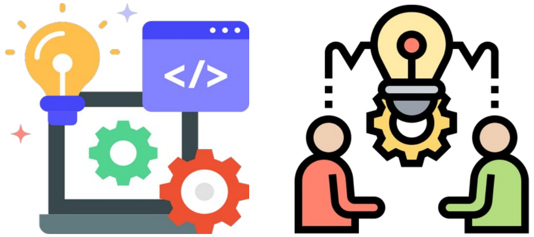
CI for Quantum Information Science and Engineering → Quantum Simulators and QPU access → Dependable HPC-QPU integration → **Productive Quantum Programming Models** → Utility-Driven Quantum Applications → Training the community → Broadening Quantum adoption

Expertise Areas

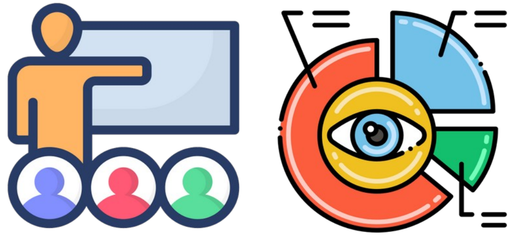
Simulation, networking and cyberinfrastructure



Scientific software development and research consulting



Training and advanced visualization



Icons created by Freepik - Flaticon

Kernels/Libs

QML

Q-UI/UX

Applications

NCSA

Quantum High Level Languages

Quantum Instruction Set Architectures

Quantum Abstract Machines

Distributed Classical-Quantum Computing

Quantum Micro/Nano programming

Quantum Circuit Execution

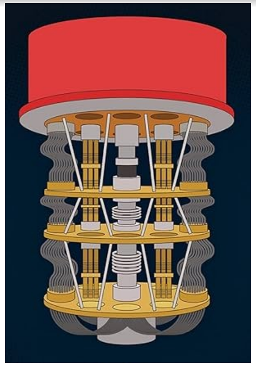
Quantum Circuit Compilation

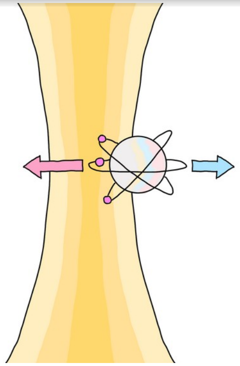
Quantum Error Correction

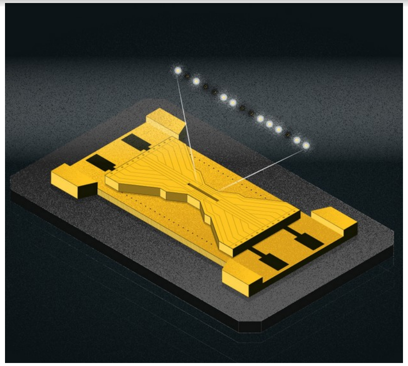
HPC-QPU integration

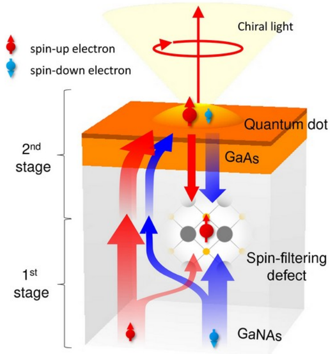
Quantum Pulse-Level Programming

Quantum Networks









HQAN

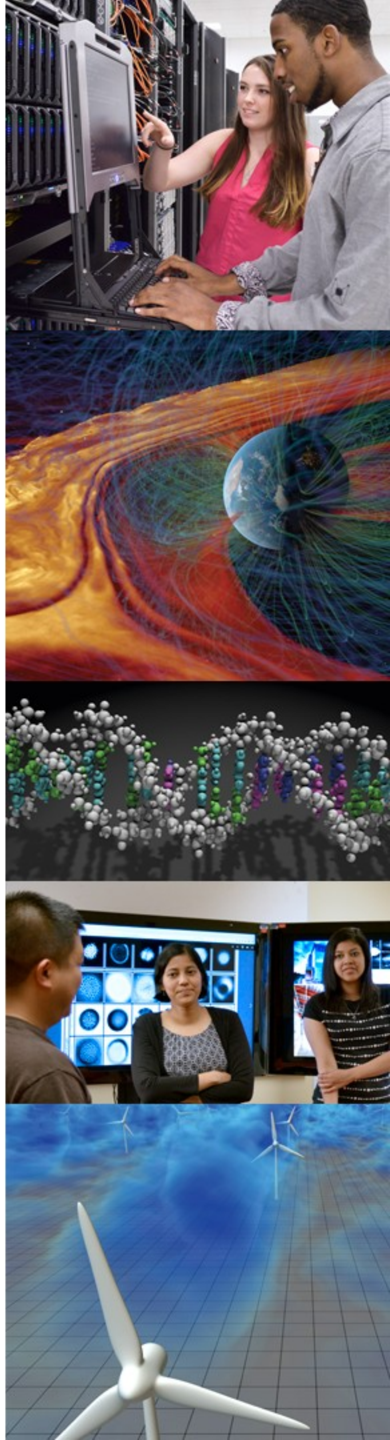
IQUIST

What have we learned in 80 years of classical computing that remains useful for programming utility-scale HPC-QPU systems?

National Center for Supercomputing Applications
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

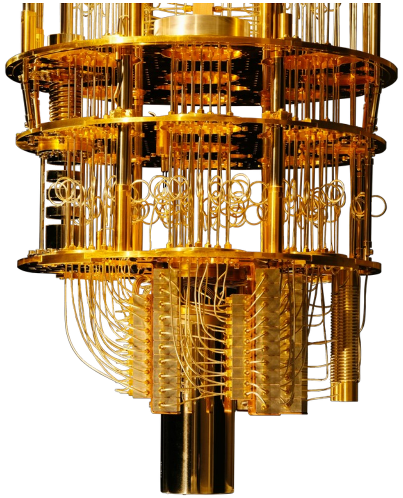
Food for thought: why do build these systems and how should we help people program them?

- Marvin Minsky (1967): *“programming is a good medium for expressing poorly understood and sloppily formulated ideas”*
- Alan J. Perlis, Foreword to SICP (1985): *“a programmer should acquire good algorithms and idioms.”*
- Harold Abelson, SICP (1985): *“Programs must be written for people to read, and only incidentally for machines to execute.”*



Most pivotal advances come from abstract understanding of resources

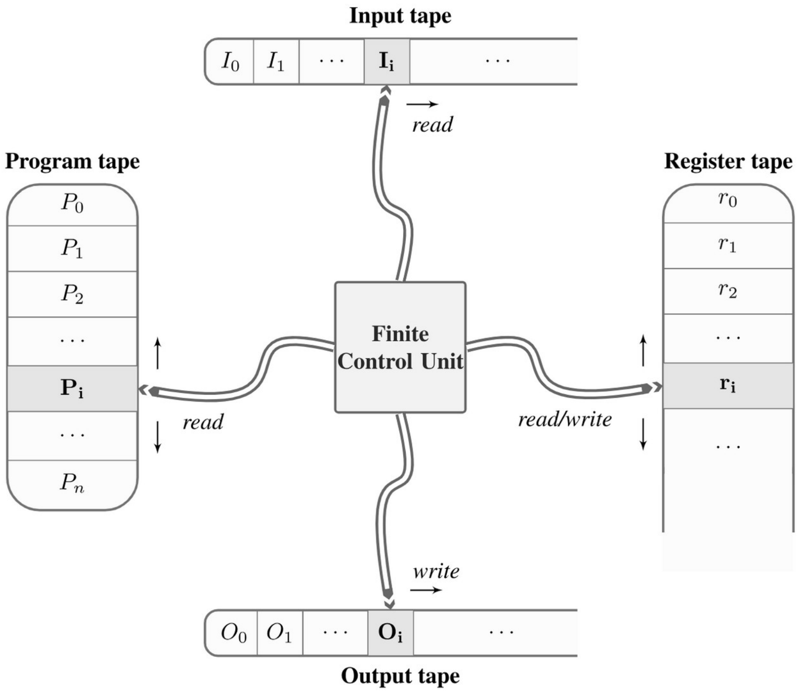
Space, Time



Space, Time
Superposition, Entanglement, Interference

The theory of quantum computation and quantum computational complexity need to become substantially more streamlined to address upcoming needs beyond 10^4 logical qubits. Much harder, urgent, underfunded and unattended problem.

Lesson 1: good *abstract* machines solve 80% of the algorithm development problem



Random Abstract Machines

Table 2. SQRAM Machine Quantum Instruction Set

Instruction	Effect
AQBIT	$qst \leftarrow qst + 1$ $pc \leftarrow pc + 1$
CNOT tar cont inv	$QR[tar] \leftarrow tar \times cnot(cont, inv, \ldots)$ $pc \leftarrow pc + 1$
GATE tar a b c d	$QR[tar] \leftarrow tar \times gate(a, b, c, d)$ $pc \leftarrow pc + 1$
HDMD tar	$QR[tar] \leftarrow tar \times gate(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})$ $pc \leftarrow pc + 1$
MSRE tar	$DS[st] \leftarrow measure(tar)$ $st \leftarrow st + 1$ $pc \leftarrow pc + 1$
PHASE tar	$QR[tar] \leftarrow tar \times gate(1, 0, 0, i)$ $pc \leftarrow pc + 1$
PI tar	$QR[tar] \leftarrow tar \times gate(1, 0, 0, e^{i\pi/4})$ $pc \leftarrow pc + 1$

Good abstract machines have instructions referring to functions and high-level objects. QRAM/QRASP are hardware simulators.

Núñez-Corrales, S., 2023. *arXiv:2307.08422*.

But: none of the existing quantum abstract machines are adequate!

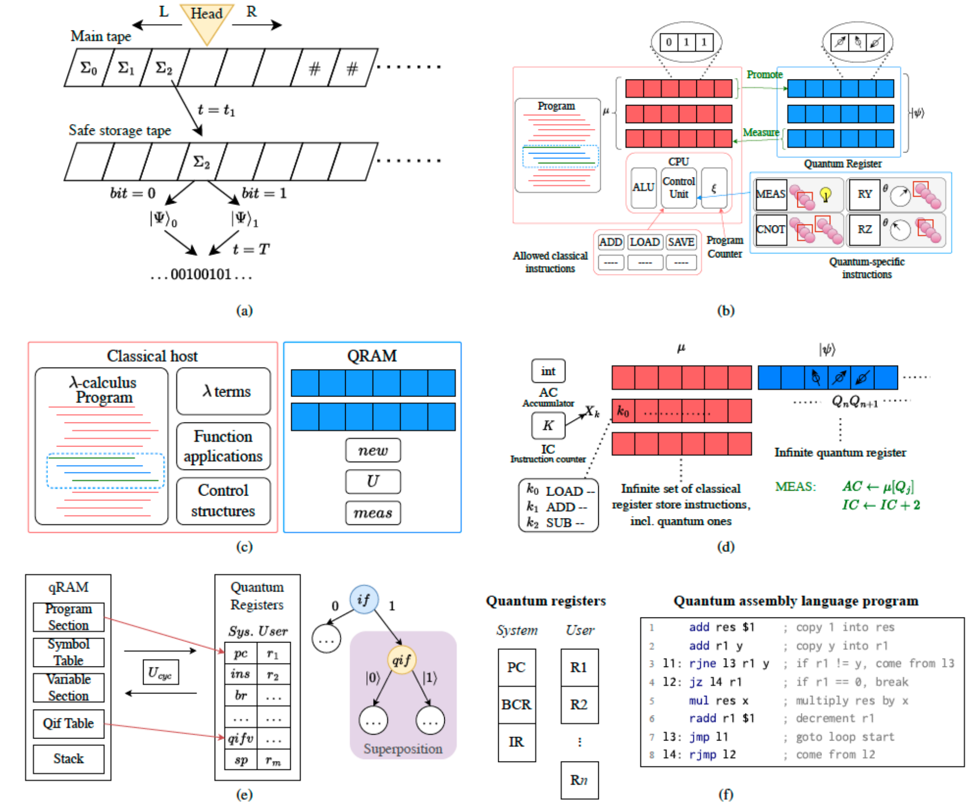


Fig. 1. Graphical representations of: (a) the Quantum Turing Machine [15], (b) the Quantum Random Access Machine [16], [17], (c) the Quantum Lambda Calculus Machine [18], (d) the Quantum Random Access Stored Program Machine [19], (e) the Quantum Register Machine [20], and (f) the Quantum Control Machine (with the code example taken verbatim from the QCM paper) [21]

TABLE I

ANALYSIS OF PREVAILING QUANTUM ABSTRACT MACHINES

Criterion	Description	QTM [19]	QRAM [19]	QRASP [19]	QRM [20]	QCM [21]	QLC [37]
1	Turing-complete & universal	✓	✓	✓	✓	✓	✓
2	Finite symbolic state				✓	✓	
3	Symbolic denotational semantics						
4	Representation-independent data types						
5	Stable instruction set architecture						
6	Verifiable formal content	✓	✓	✓	✓	✓	✓
7	Classical-quantum regularity	✓			✓†	✓†	
8	Compact instruction representation		✓	✓	✓	✓	✓
9	Degeneracy of implementation	✓	✓	✓	✓	✓	✓
10	Predictable procedural composability		✓	✓	✓	✓	
11	Intrinsic ensemble semantics	✓	✓	✓			
12	Resource-constructible functions	✓	✓	✓	✓	✓	✓
13	Standard instruction cycle		✓	✓	✓	✓	
14	Classical control flow			✓	✓	✓	✓
15	Quantum/hybrid control flow				✓	✓	
Total		6✓	8✓	9✓	11✓	11✓	6✓

† partial satisfaction due to explicit mention of unitary gates

Núñez-Corrales, S., Di Matteo, O., Dumbell, J., Edwards, M., Giusto, E., Pakin, S., Stirbu, V.Stęchły, M. (2025, submitted). Productive Quantum Programming Needs Better Abstract Machines. *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE/arXiv (submitted).

Lesson 2: the performance-expressiveness trade-off is *universal* and *unavoidable*

Expressive power:

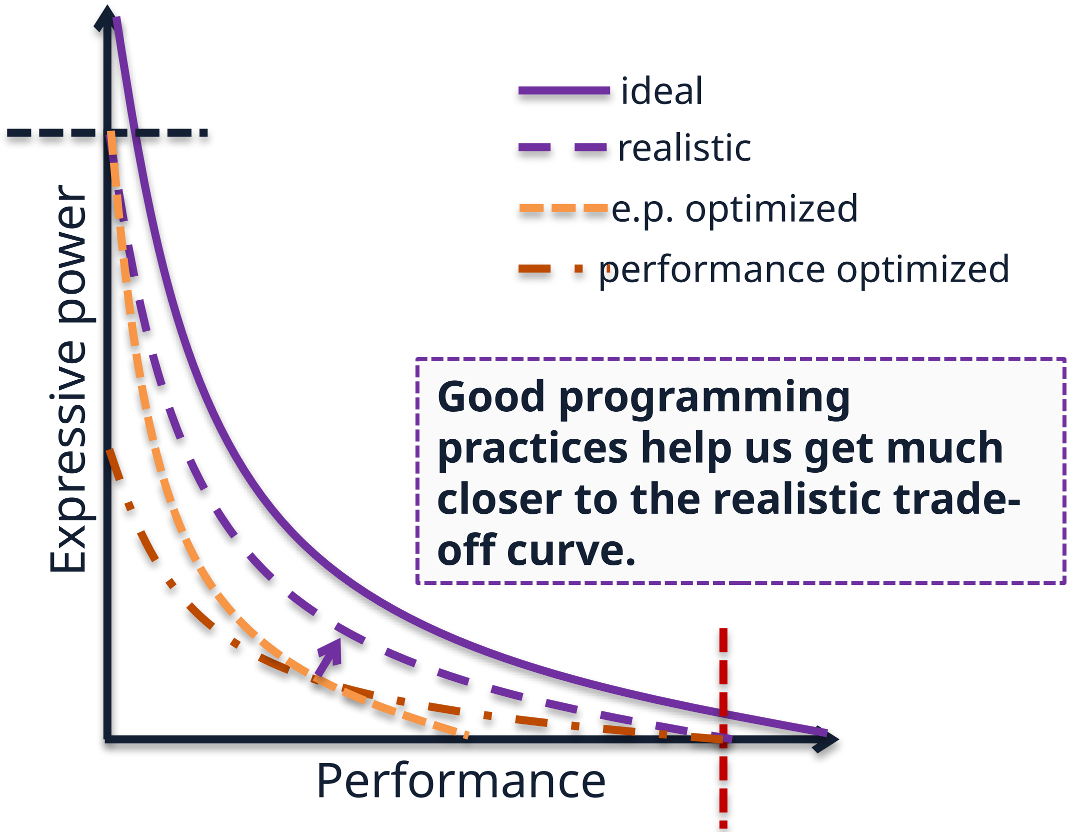
How many different problems can I describe (and solve!) with tool X?

Harder to measure:

- # of use cases
- # of instances of code reuse
- # of lines of code

Limited by:

- Theoretical bounds
- Problem features (e.g., size)
- Human needs (e.g., code maintainability and readability)



Good programming practices help us get much closer to the realistic trade-off curve.

We lack programming models that induce good practices in quantum computing. Pulse-level and circuit-level programming are likely not one of them.

Performance:

How few resources can I use to solve a specific problem with tool X?

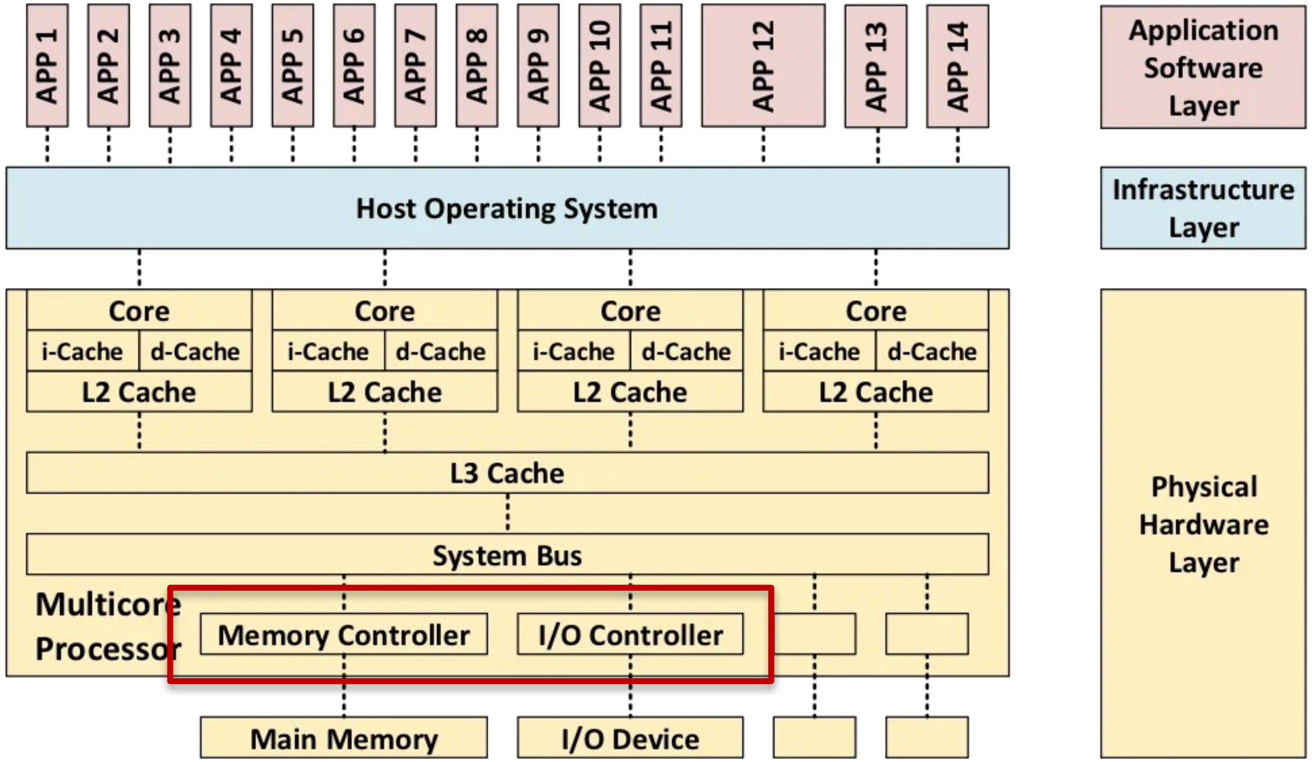
Easier to measure:

- Time
- Resource usage/pressure

Limited by

- Technology
- Available hardware resources
- Hidden costs of compilation / interpretation + OS

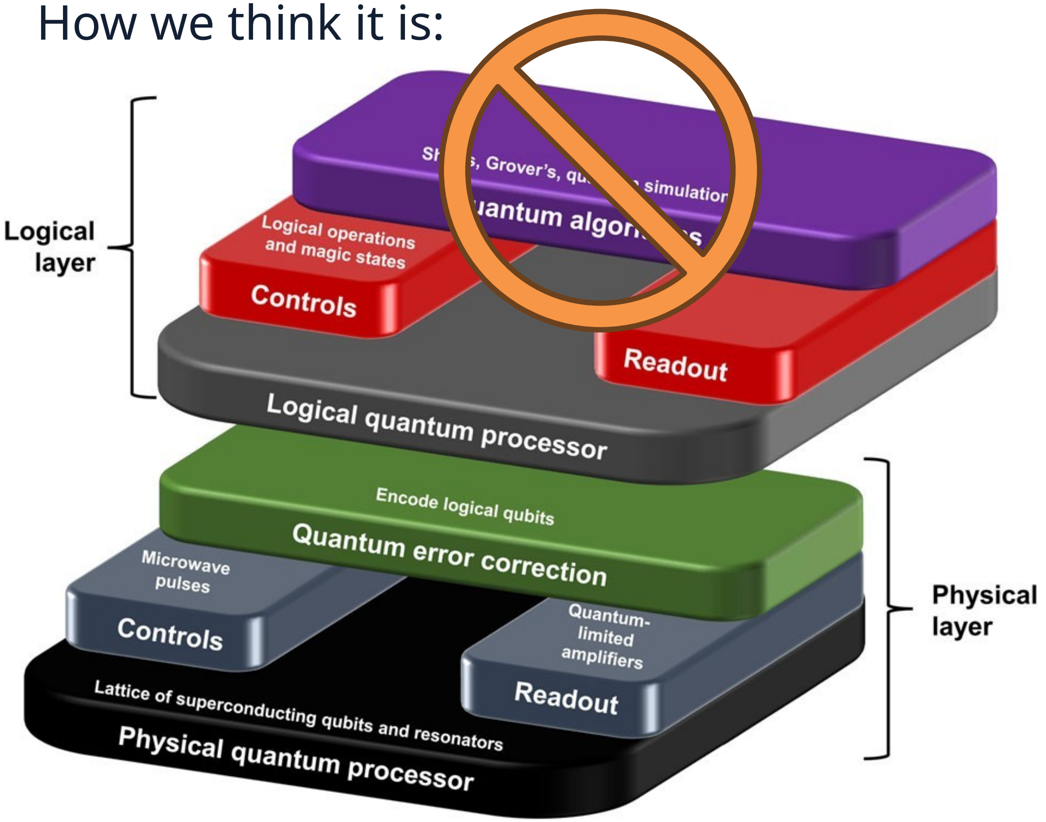
Lesson 3: control software becomes control hardware with time



Pulse-level synthesis and even higher quantum control primitives will likely become part of an SoC-like architecture.

Lesson 4: good stacks enable opportunistic refinement for hw-sw co-design

How we think it is:



Current quantum stacks focus too much on qubit function/performance, not enough on how the interfaces across layers should communicate.

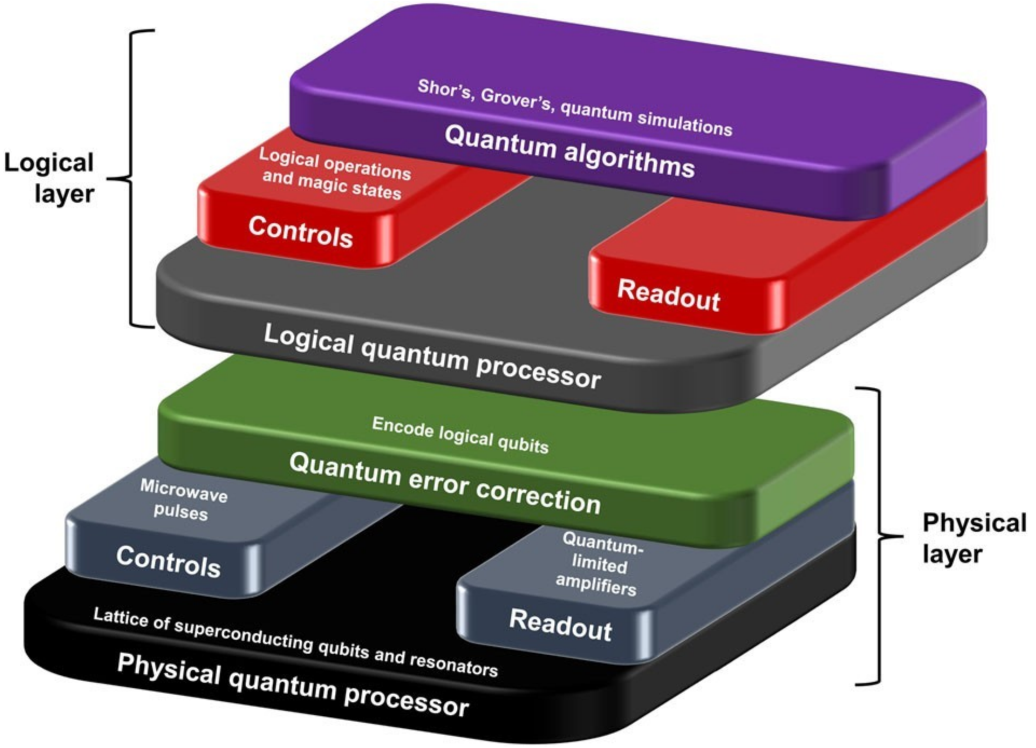
What we should aim for:

Hardware detail free zone

Applications
Algorithms (aka Libraries)
Languages
Orchestration (aka OS)
Classical-quantum ISA
Classical-quantum organization
Logical qubits
QECC+QEM
Pulse-level synthesis
Physical qubits

Lesson 5: good stacks separate concerns efficiently for programmers

How we think it is:



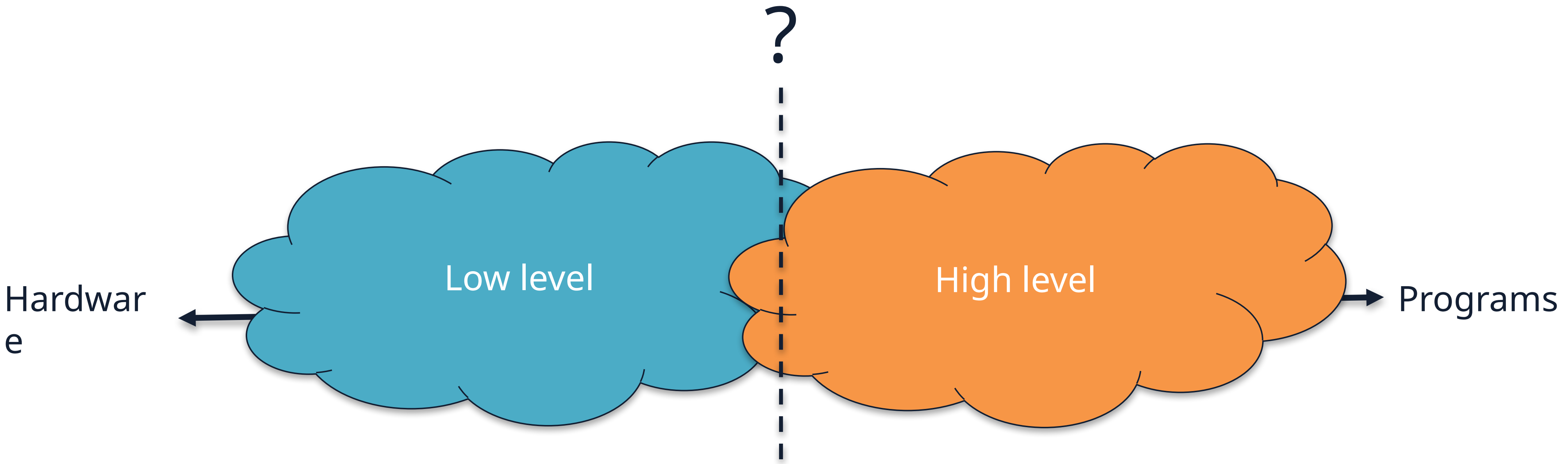
What we should aim for:

Hardware detail free zone

- Applications
- Algorithms (aka Libraries)
- Languages
- Orchestration (aka OS)
- Classical-quantum ISA
- Classical-quantum organization
- Logical qubits
- QECC+QEM
- Pulse-level synthesis
- Physical qubits

Heuristic: the difficulty of programming scales roughly proportional to the cube of the number of hardware details required to write code.

Lesson 6: circuits are <u>not</u> high-level constructs

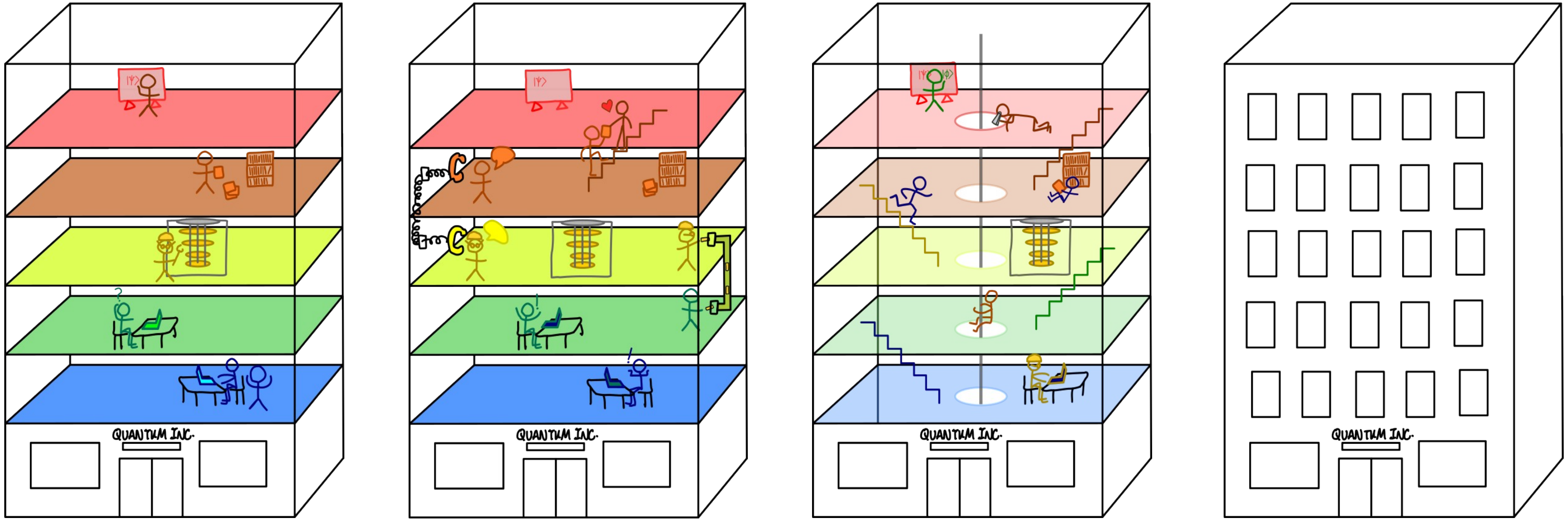


A. Denotational semantics: constructs isomorphic to functions within a space of objects w/ a closed algebra
B. Representation independent: constructs should not vary if the “digital” representation changes
C. Compositionality: the effect of large constructs is understandable from composition of smaller ones without abandoning representation independence

Quantum algorithms and applications will be found more quickly once we find true high-level constructs. Not there yet.

Núñez-Corrales, S., Frenkel, M. and Abreu, B., QCE 23; Di Matteo O, Núñez-Corrales S, Stęchły M, Reinhardt SP, Mattson T. arXiv:2405.13918.

Why do we want good abstractions?



Separation of concerns

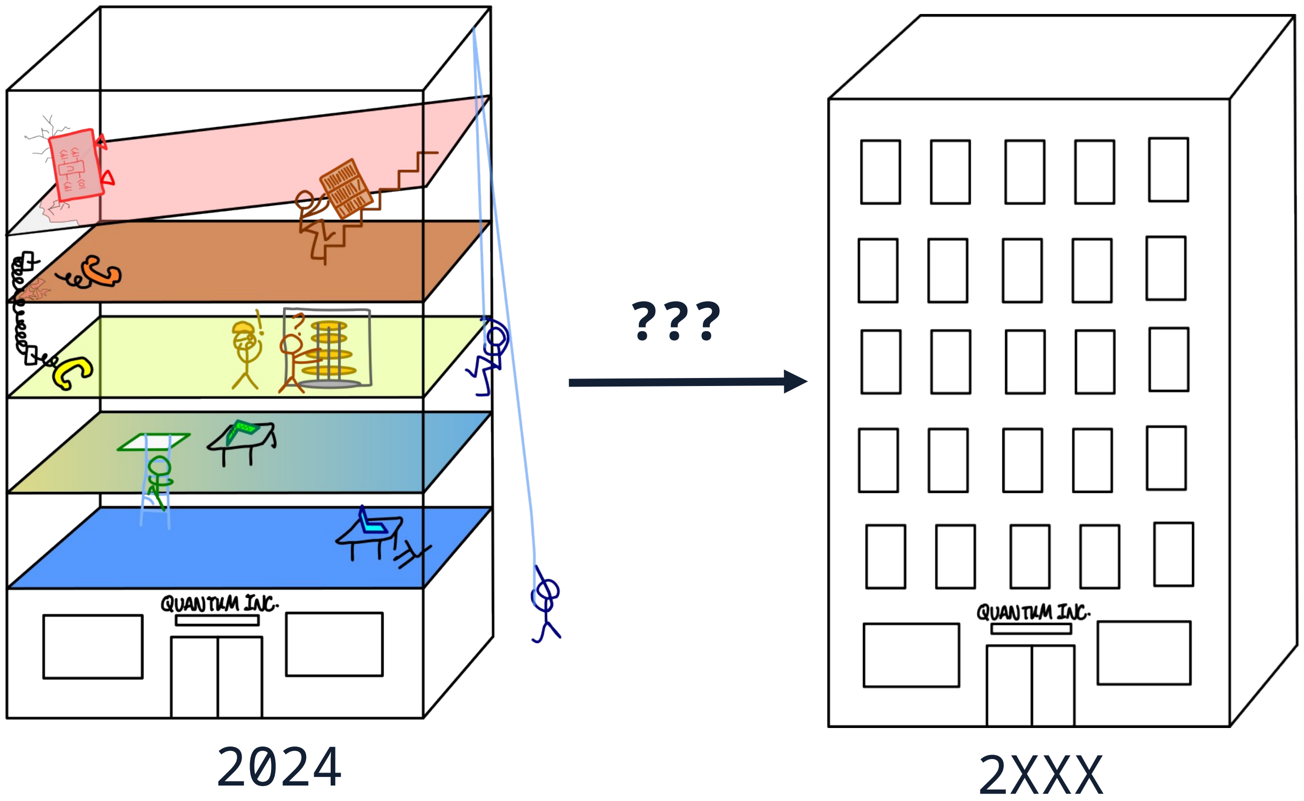
Well-defined interactions between adjacent layers

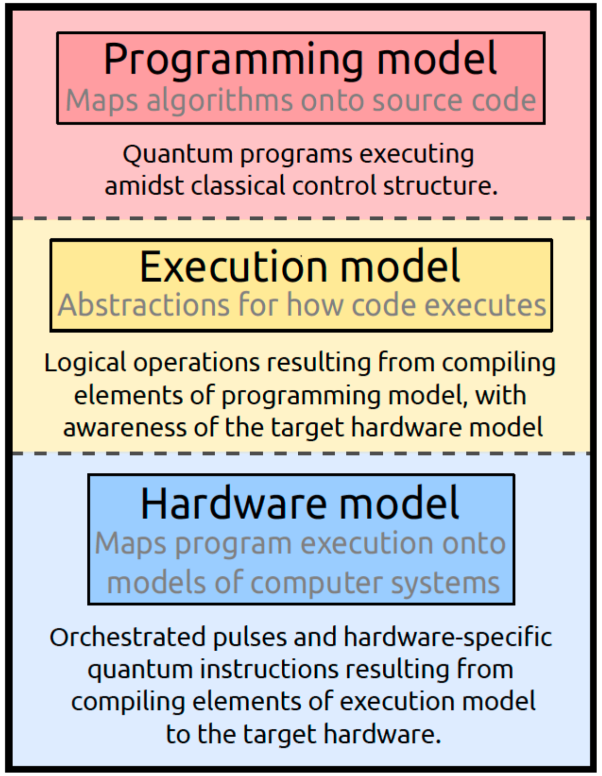
Opportunistic refinement

...

Di Matteo, O., Núñez-Corrales, S., Stęchły, M., Reinhardt, S.P. and Mattson, T., 2024, September. An abstraction hierarchy toward productive quantum programming. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)* (Vol. 1, pp. 979-989). IEEE.

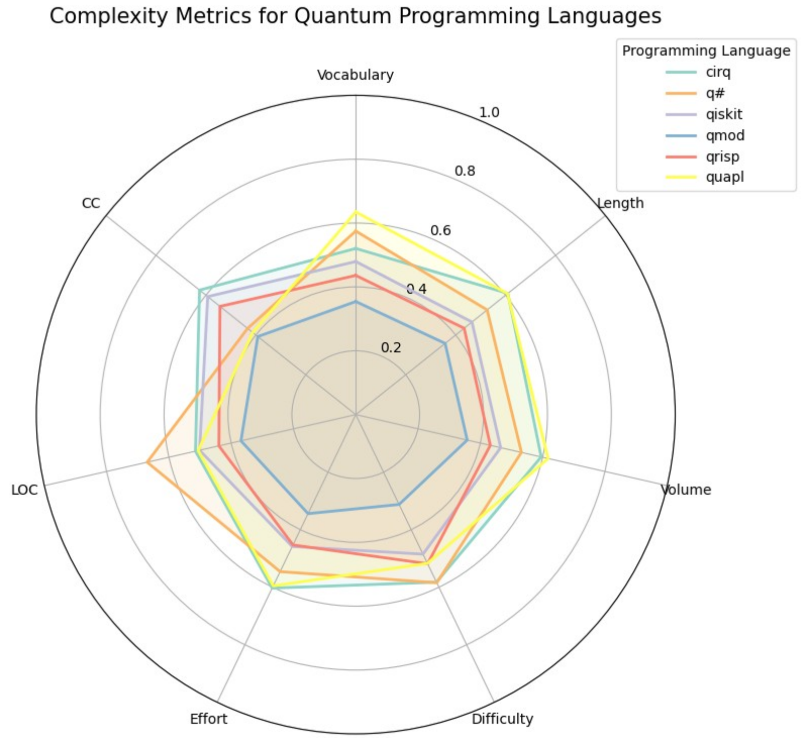
The state of quantum programming today

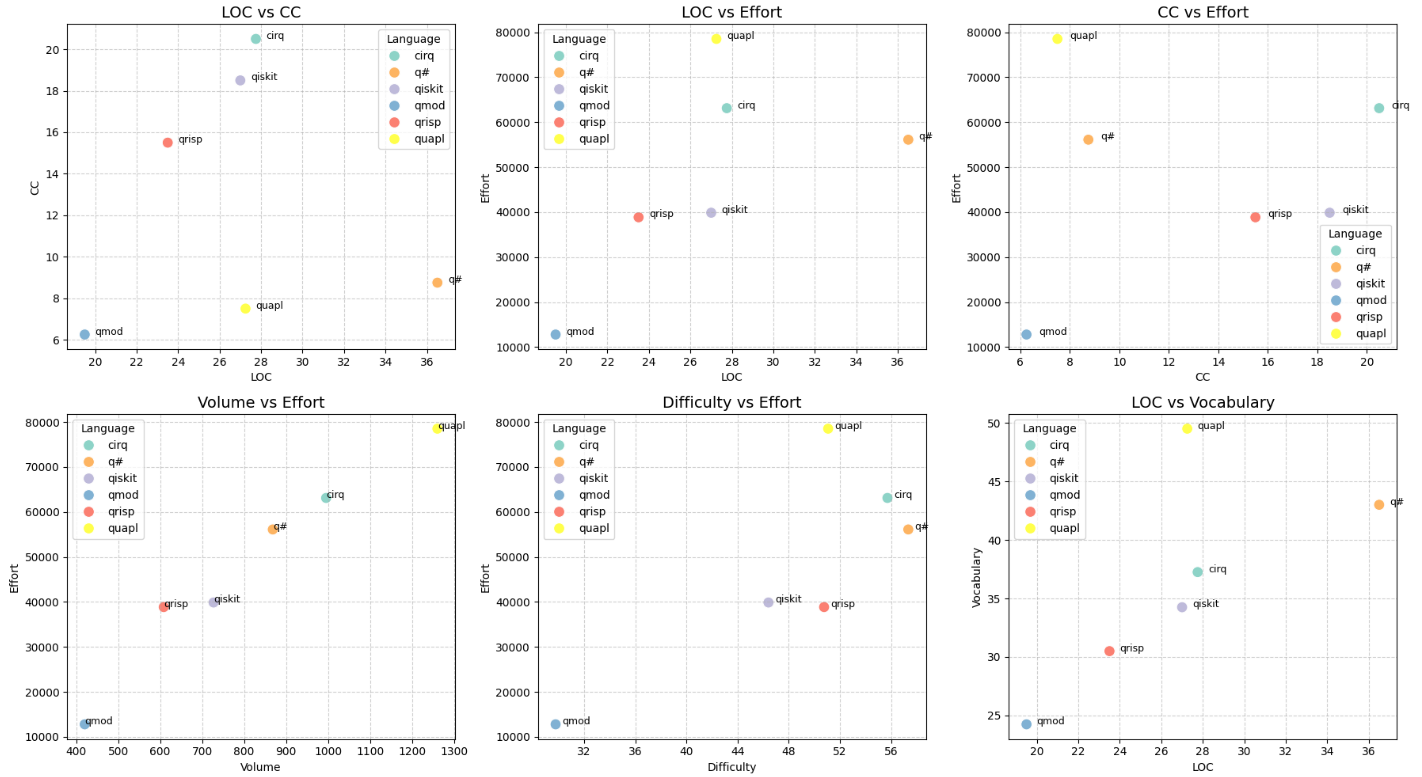




Di Matteo, O., Núñez-Corrales, S., Stęchły, M., Reinhardt, S.P. and Mattson, T., 2024, September. An abstraction hierarchy toward productive quantum programming. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)* (Vol. 1, pp. 979-989). IEEE.

Quantum programming languages lack sufficient expressiveness and productivity

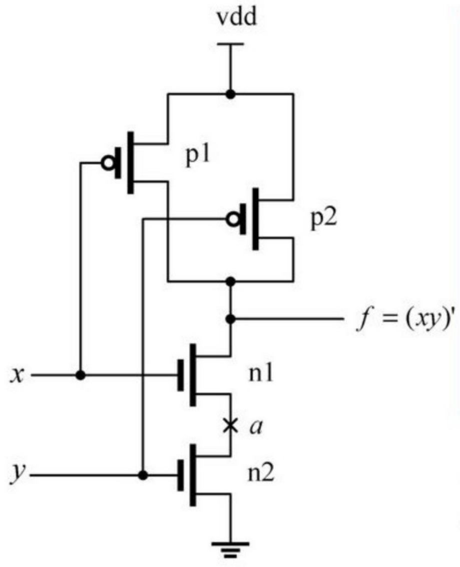




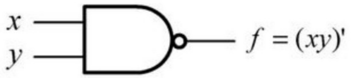
Corrales-Garro, F., Valerio-Ramírez, D., Núñez-Corrales, S. (2025) Is Productivity in Quantum Programming Equivalent to Expressiveness? arXiv:2504.08876

Lesson 7: generation/validation replace programming at very large hardware scales

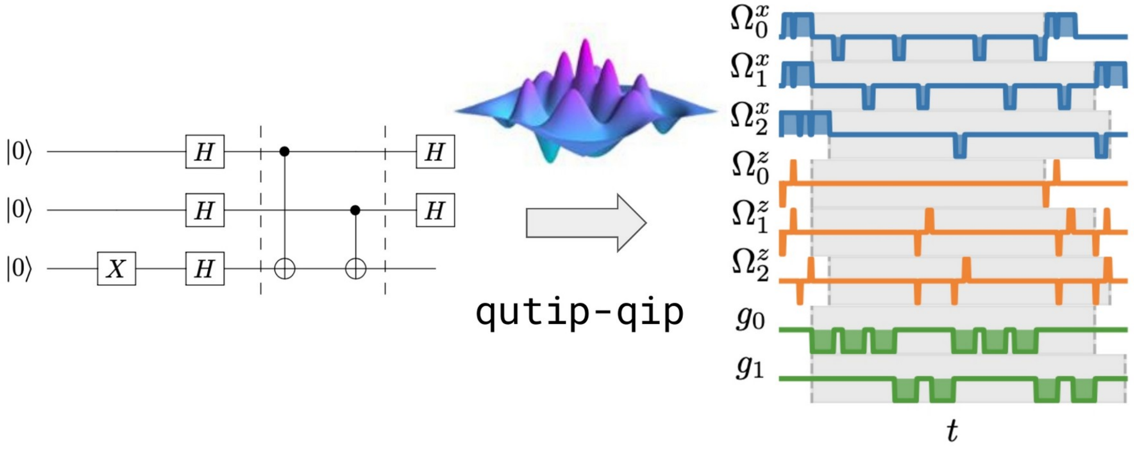
VLSI: generate and optimize -> validate




```
module my_nand (input x, y, output f);
supply1  vdd;
supply0  gnd;
wire     a;
// NAND gate body
   pmos p1 (f, vdd, x);
   pmos p2 (f, vdd, y);
   nmos n1 (f, a, x);
   nmos n2 (a, gnd, y);
endmodule
```

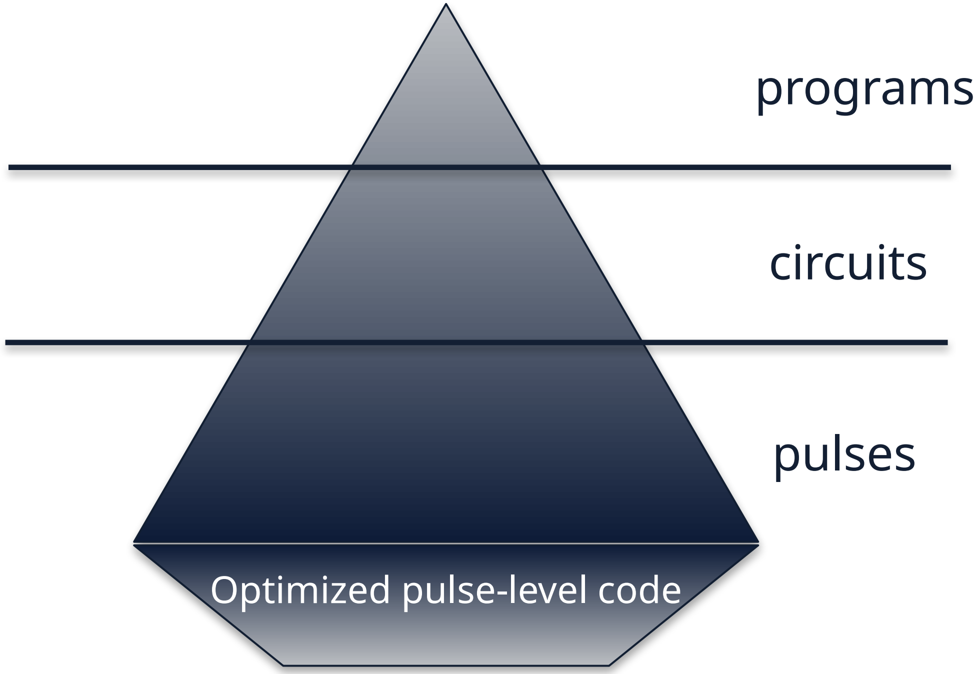


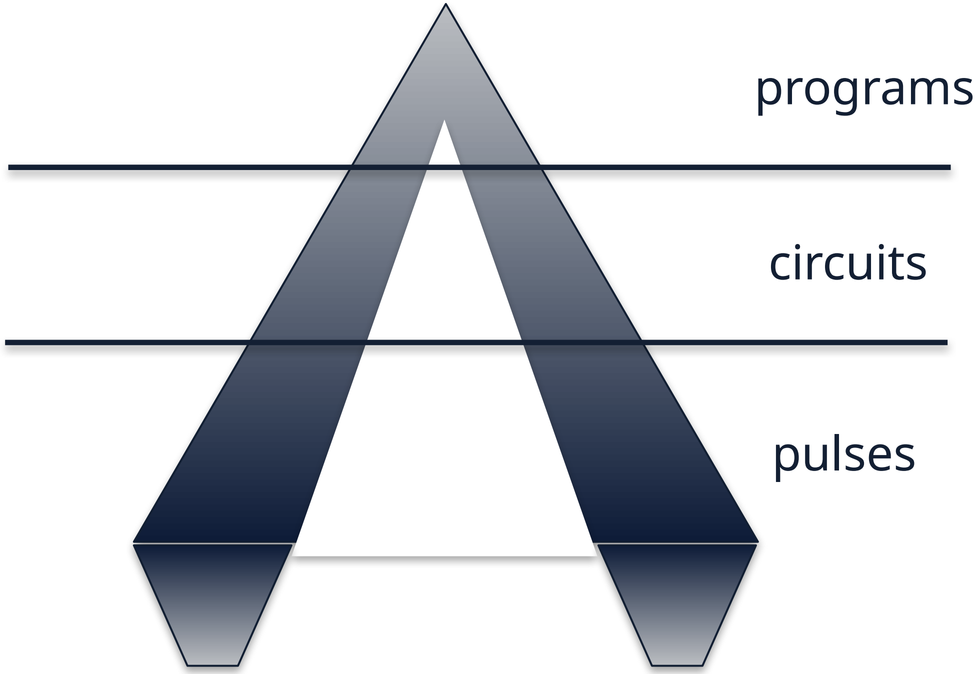
VLQI: very large quantum integration



Utility-scale, fault-tolerant quantum computers pose a wicked control problem for humans. Most likely, many of these are NP-HARD. VLQI will be self-bootstrapping.

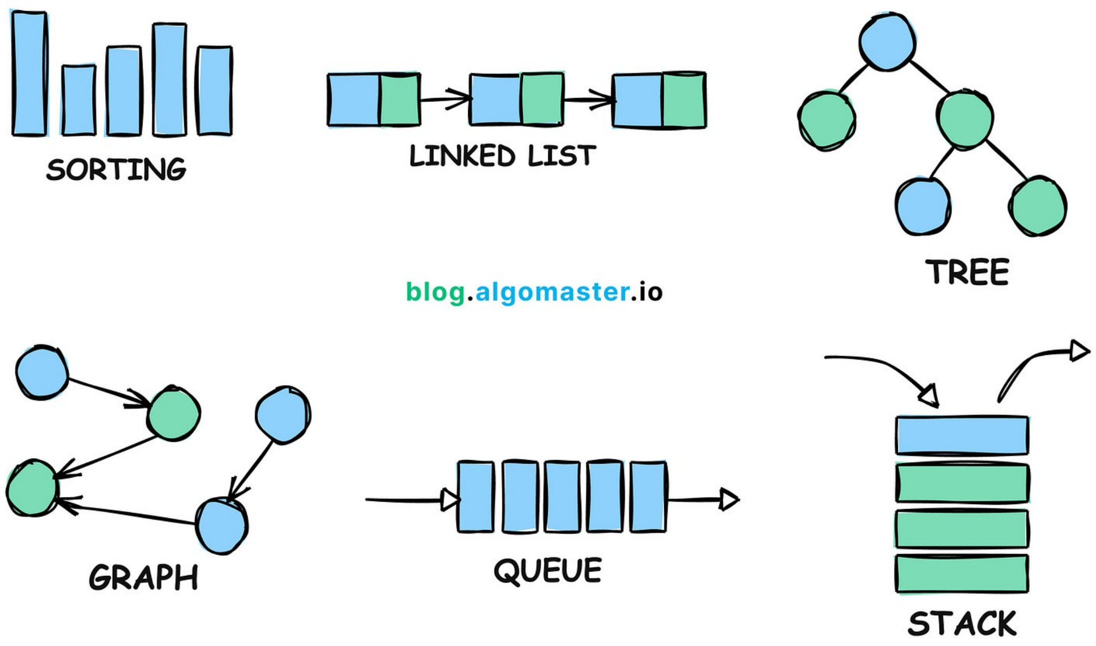
Lesson 8: resist to optimize within differences that make no difference





As quantum computers become larger (>10^4 logical qubits), optimizations must occur as high up as possible.

Lesson 9: modularity and indirection organize complexity



Few quantum data structures, more needed to scale up to utility-scale systems.

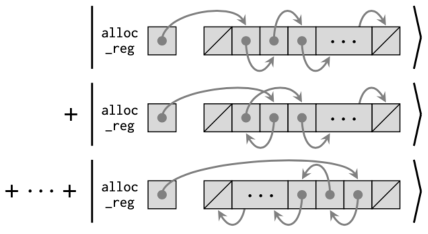


Fig. 17. Result of symmetrization on the initial program state from Figure 12 (normalizing amplitudes not shown). The symmetrized free list exists in a superposition of all possible permutations.

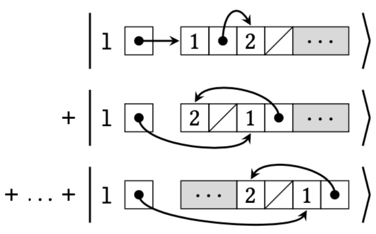


Fig. 18. Unique physical representation of state $1 \hookrightarrow [1,2]$ (normalizing amplitudes not shown), which stores data in a superposition of all possible allocation sites and is history-independent.

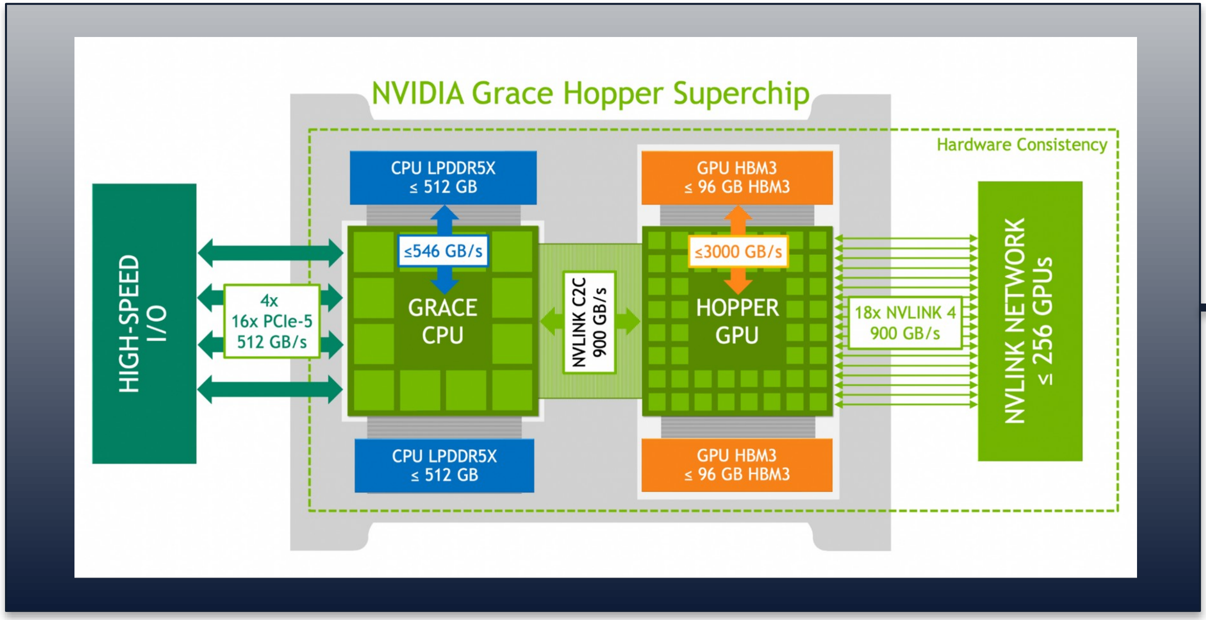
Data Structure	Reversible	Recursion	Mutation	Complexity	LoC	Qubits	Gates
List							
– `length`	Yes	Yes	No	$O(n)$	20	$34n+32$	$23n+3$
– `sum`	Yes	Yes	No	$O(n)$	20	$34n+40$	$21n+3$
– `find_pos`	Yes	Yes	No	$O(n)$	20	$42n+31$	$19n+3$
– `remove`	Yes	Yes	Yes	$O(n)$	48	$26n+56$	$42n+3$
Stack (list)							
– `push_front`	Yes	No	Yes	$O(1)$	8	40	4
– `pop_front`	Yes	No	Yes	$O(1)$	8	48	4
Queue (list)							
– `push_back`	Yes	Yes	Yes	$O(n)$	21	$34n+32$	$24n$
– `pop_front`	Yes	No	Yes	$O(1)$	8	48	4
String (word)							
– `is_empty`	Yes	No	No	$O(1)$	2	25	3
– `length`	Yes	No	No	$O(1)$	2	24	1
– `get_prefix`	Yes	No	No	$O(k)$	8	$11k$	52
– `get_substring`	Yes	No	No	$O(k)$	8	$12k$	54
– `get`	Yes	No	No	$O(k)$	7	$6k+1$	19
– `is_prefix`	Yes	Yes	No	$O(\text{poly}(k))$	26	k^2+11k	$98k+3$
– `num_matching`	Yes	Yes	No	$O(\text{poly}(k))$	42	$k^2+13k+4$	$110k+127$
– `equal`	Yes	No	No	$O(k)$	8	$6k+3$	5
– `concat`	Yes	No	No	$O(k)$	9	$11k$	8
– `compare`	Yes	Yes	No	$O(\text{poly}(k))$	27	$5k^2+12k$	$108k+3$
Set (radix tree)							
– `insert`	Yes	Yes	Yes	$O(\text{poly}(k))$	136	$13k^2+21k+9$	$1440k^2+5056k$
– `contains`	Yes	Yes	No	$O(\text{poly}(k))$	334	$17k^2+18k+2$	$784k^2+1612k+1$
Set (hash table)*							
– `insert`	Yes	Yes	Yes	$O(n)$	63	$52n+72$	$68n+15$
– `contains`	Yes	Yes	No	$O(n)$	7	$52n+81$	$136n+39$

* Hash table-based sets are not history-independent.

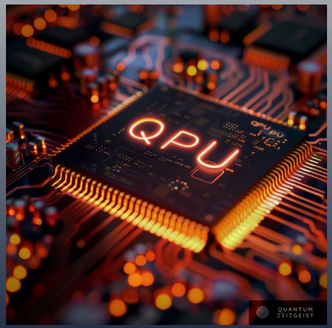
Yuan, C. and Carbin, M., 2022. *OOPSLA2*.

Leadership Class Compute Facility - LCCF (TACC+NCSA+IQUIST)

HPC+AI @ NCSA

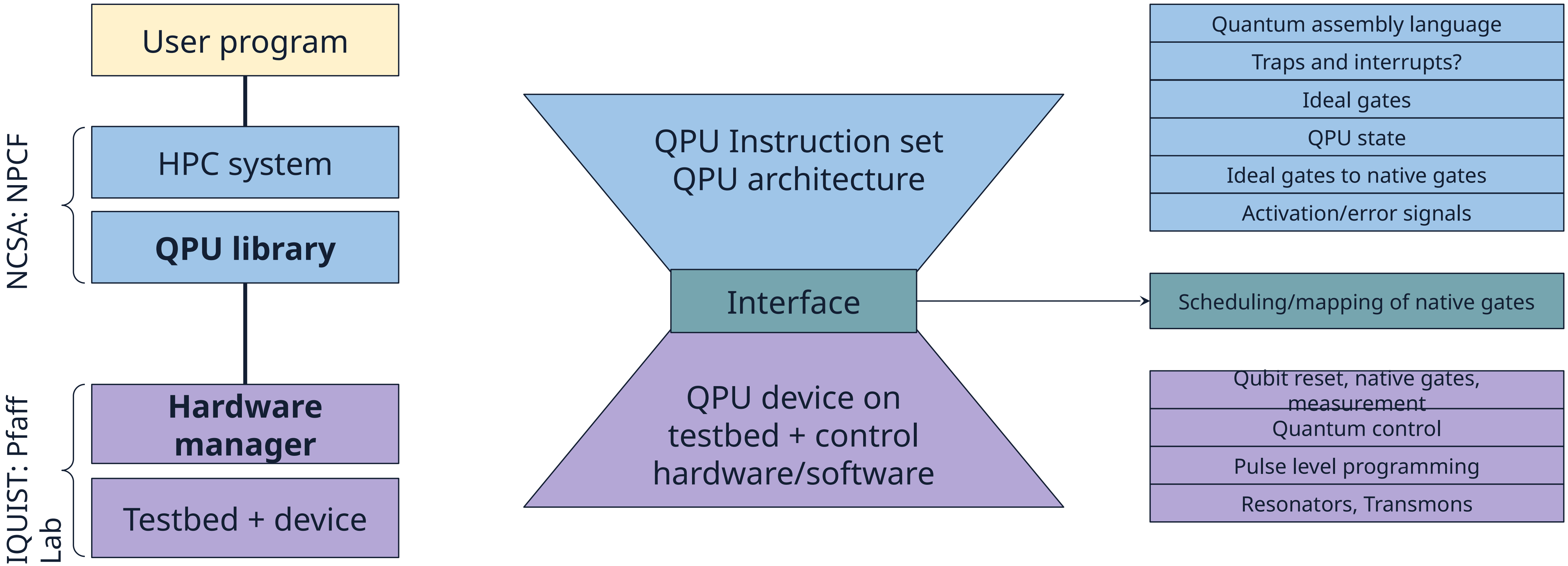


QPU @ IQUIST

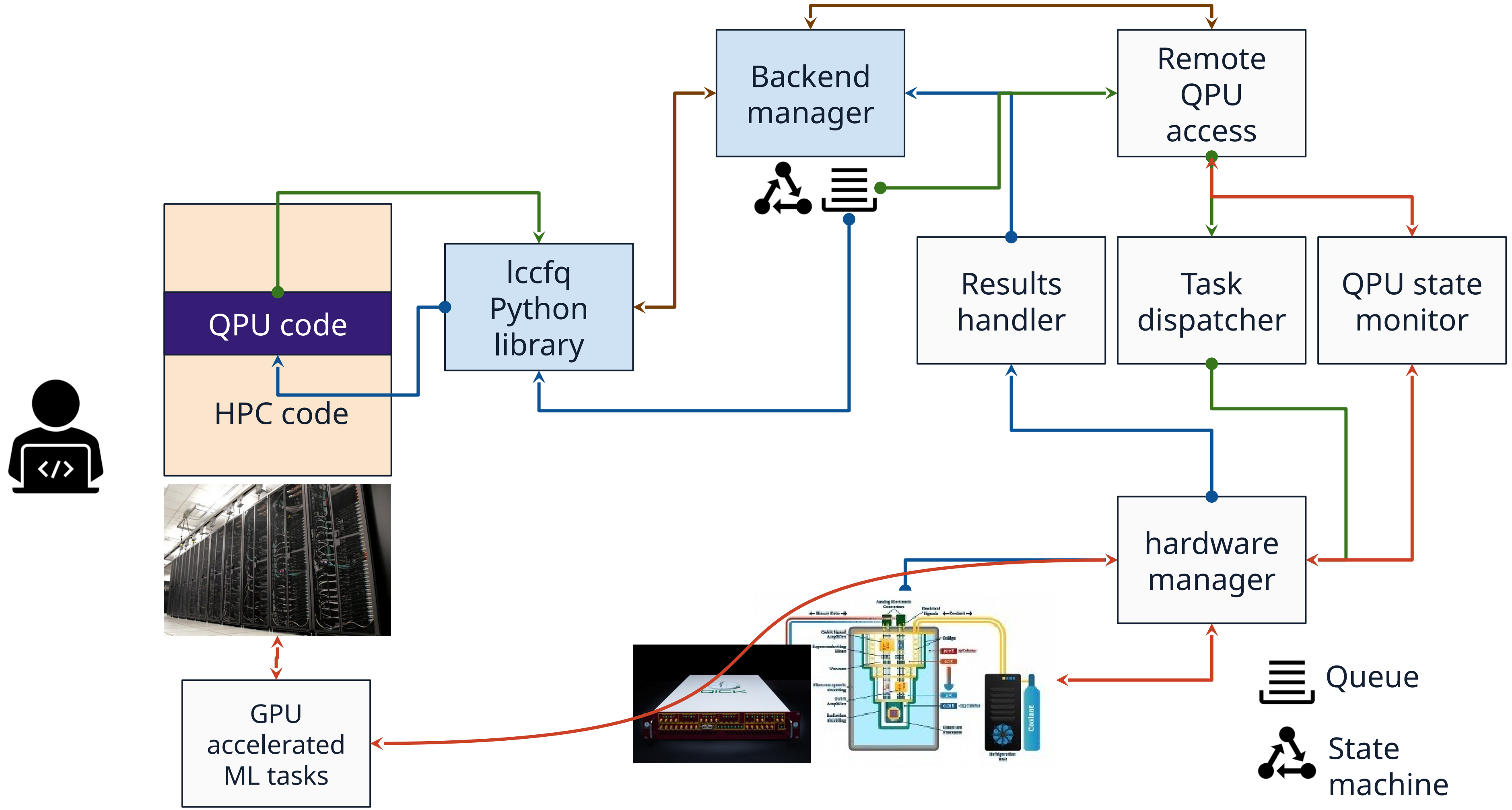


Tight HPC-AI-QPU integration	Development of quantum cyberinfrastructure	Deploy research and user access

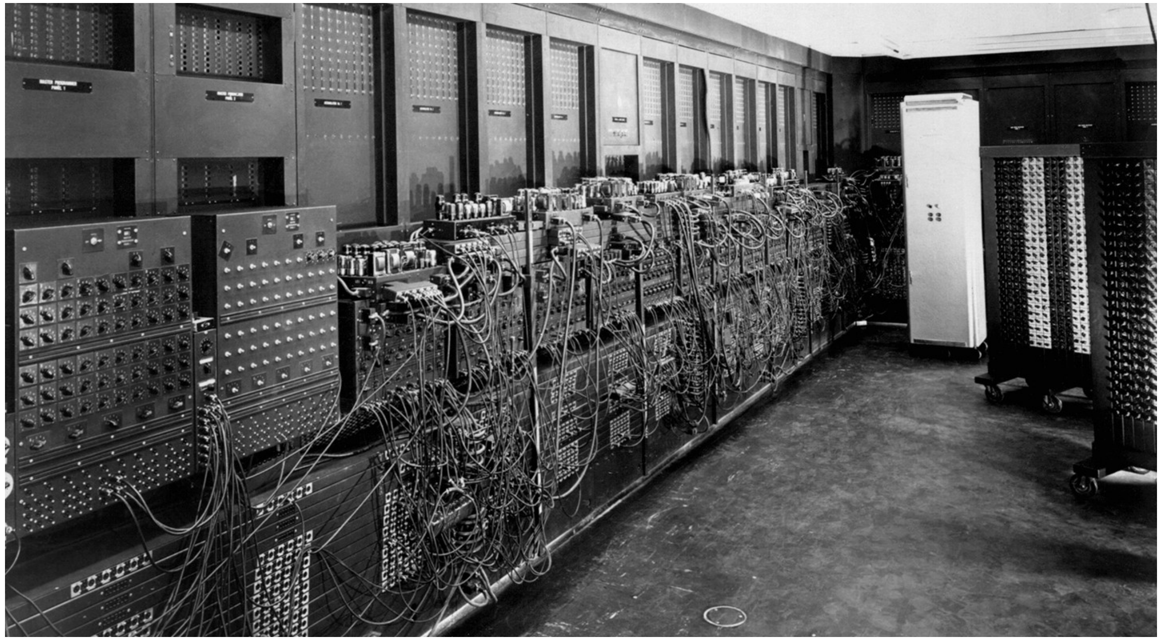
Separation of concerns to promote opportunistic refinement



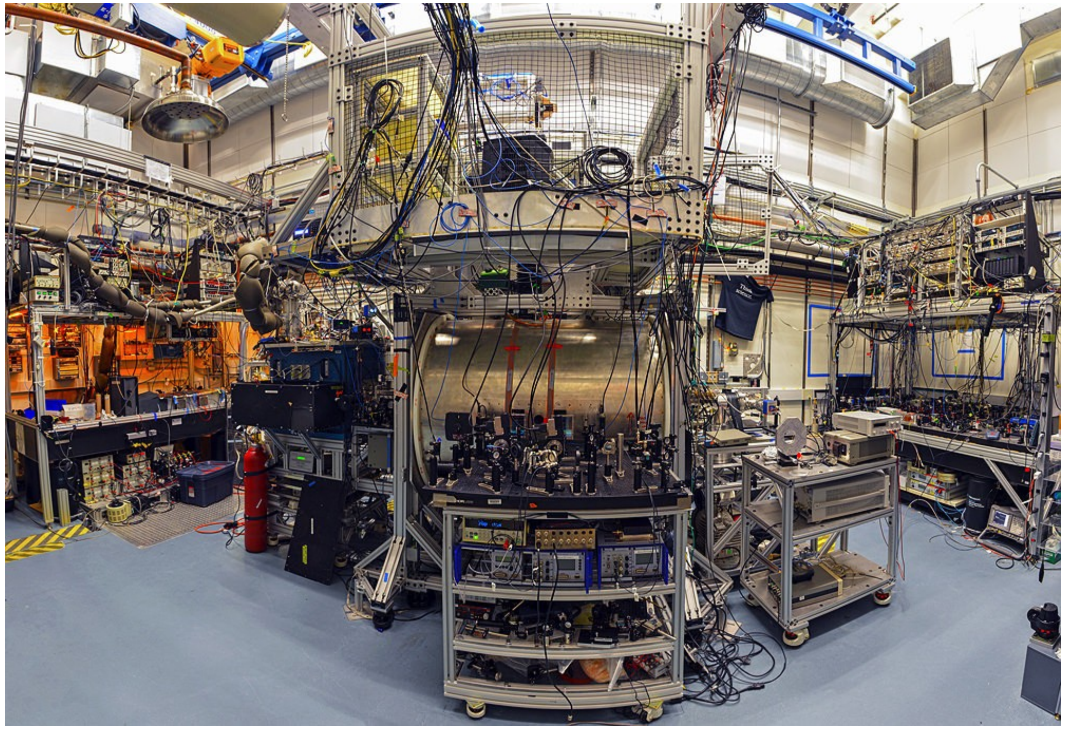
… differs from how implementation looks like



Academic hardware - Back to the 1940-1950



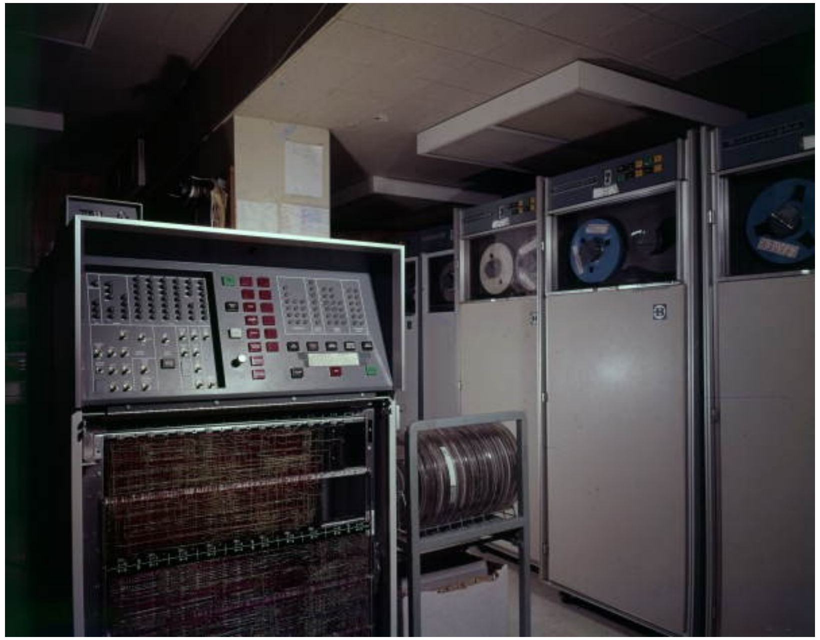
UPenn 1945 (ENIAC)



Harvard 2024 (HQI)

Staying at the forefront is messy. <u>But:</u> not all mess is unavoidable.

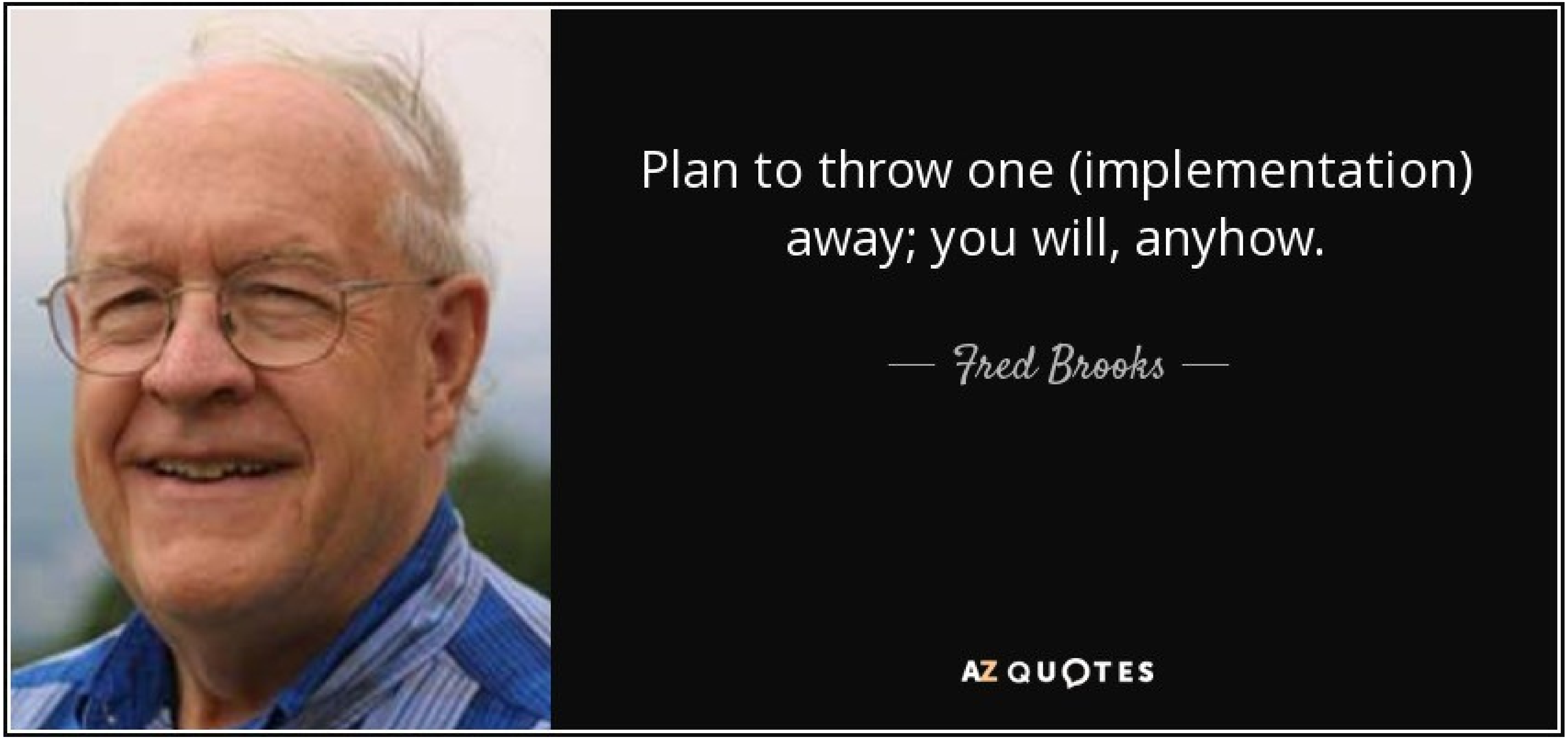
Vendor hardware - Back to the 1950-1960



UF Gainesville 1968 (Burroughs)

Munich Valley 2024 (IQM)

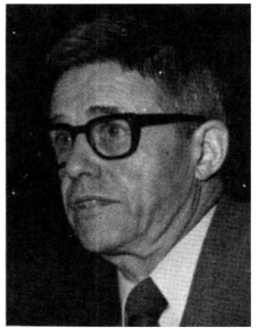
Market pressures drive innovation fast. Market pressures explain technology gaps.


Plan to throw one (implementation) away; you will, anyhow.
— Fred Brooks —
AZ QUOTES

Back to basics: seeking expressiveness

Notation as a Tool of Thought

Kenneth E. Iverson
IBM Thomas J. Watson Research Center



Key Words and Phrases: APL, mathematical notation
CR Category: 4.2

Author's present address: K.E. Iverson, I.P Sharp Associates, 145 King Street West, Toronto, Ontario, Canada M5H1J8.

The importance of nomenclature, notation, and language as tools of thought has long been recognized. In chemistry and in botany, for example, the establishment of systems of nomenclature by Lavoisier and Linnaeus did much to stimulate and to channel later investigation. Concerning language, George Boole in his *Laws of Thought* [1, p.24] asserted "That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted."

Mathematical notation provides perhaps the best-known and best-developed example of language used consciously as a tool of thought. Recognition of the important role of notation in mathematics is clear from the quotations from mathematicians given in Cajori's *A History of Mathematical Notations* [2, pp.332,331]. They are well worth reading in full, but the following excerpts suggest the tone:

> By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.
>
> A.N. Whitehead

1. Important Characteristics of Notation

In addition to the executability and universality emphasized in the introduction, a good notation should embody characteristics familiar to any user of mathematical notation:

- Ease of expressing constructs arising in problems.
- Suggestivity.
- Ability to subordinate detail.
- Economy.
- Amenability to formal proofs.

The foregoing is not intended as an exhaustive list, but will be used to shape the subsequent discussion.

Conclusion: we need prescriptive, abstraction-driven design toward HPC-QPU programmability