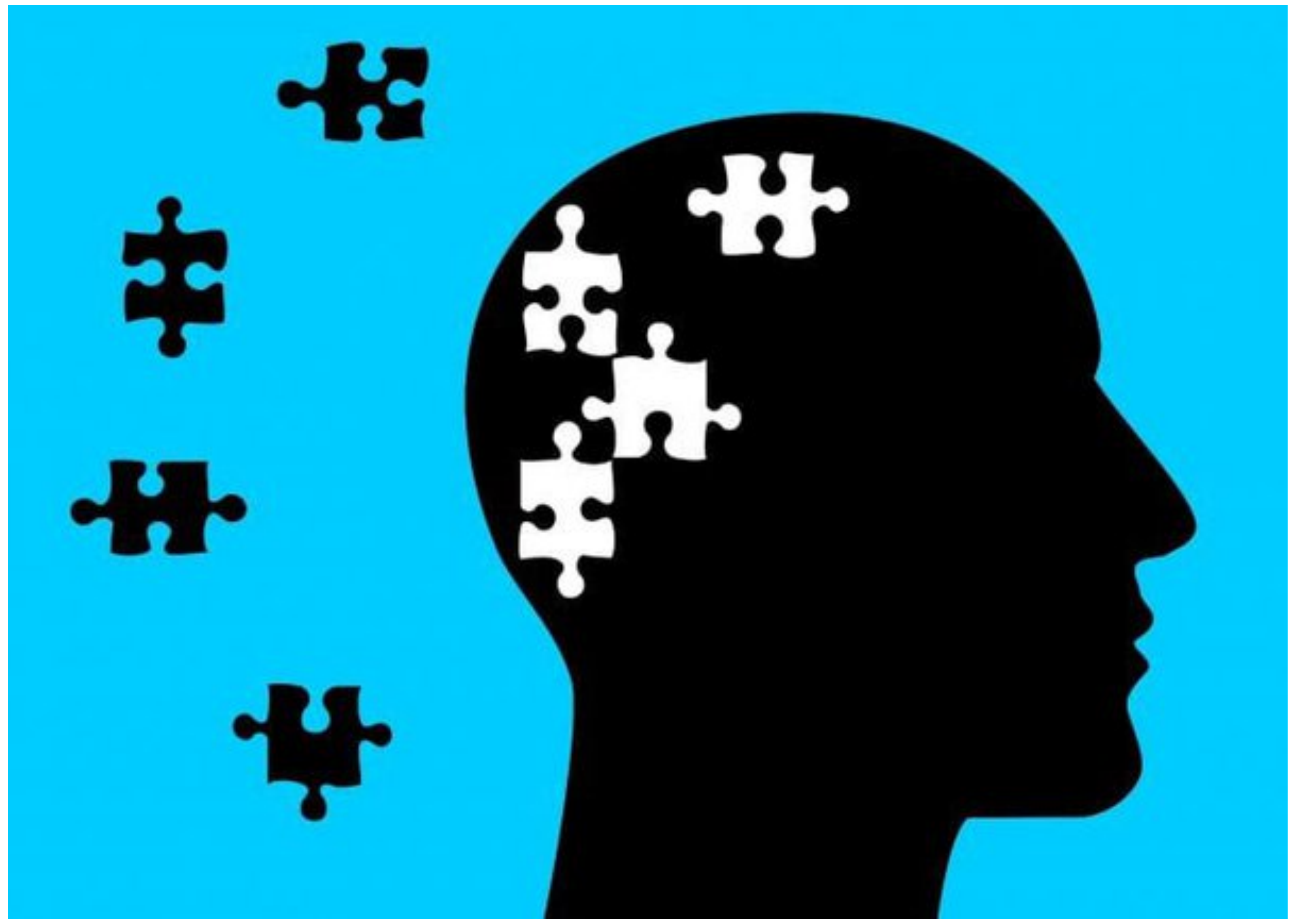
A Computer Science Perspective on the Foundations of Quantum Computing

Amr Sabry
Department of Computer Science



A few CS concepts
(to get in the right state of mind)

Key CS Concept I

Notation

1979 ACM Turing Award Lecture
Delivered at ACM '79, Detroit, Oct. 29, 1979

Notation as a Tool of Thought

Kenneth E. Iverson
IBM Thomas J. Watson Research Center

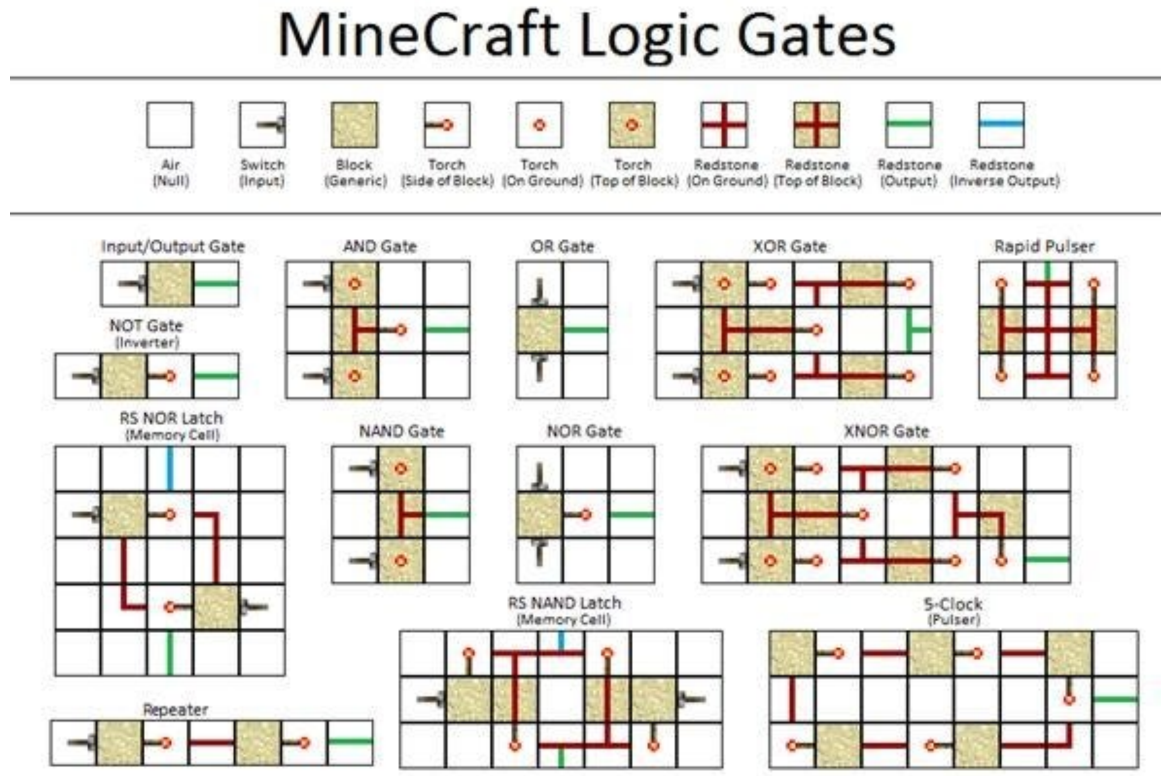
What is?

MCMLXXXIII * DCCXIII

- Complexity of algorithms depends on the notation used
- Not to speak of readability, ease of understanding, maintainability, potential for errors, etc.

Key CS Concept II

Encoding



Example: complex numbers “uninteresting” as they can be efficiently encoded

$$V = \begin{pmatrix} a+ib \\ c+id \end{pmatrix} \quad \mapsto \quad V^{\mathbb{R}} = \begin{pmatrix} a \\ b \\ -d \\ c \end{pmatrix}$$

$$M = \begin{pmatrix} a+ib & \cdots \\ \cdots & \end{pmatrix} \quad \mapsto \quad M^{\mathbb{R}} = \begin{pmatrix} \begin{pmatrix} a & -b \\ b & a \end{pmatrix} & \cdots \\ \cdots & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ i \end{pmatrix}$$

$$\Big\downarrow (.)^{\mathbb{R}} \qquad \Big\downarrow (.)^{\mathbb{R}} \qquad \Big\downarrow (.)^{\mathbb{R}}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

Key CS Concept III

Symbolic execution

```
power a 0 = 1
power a n = a * power a (n-1)

-- Normal execution
power 2 3 = 8

-- Symbolic execution / Partial evaluation
power a 3 = a * a * a * 1
```

2. Solve the Equation of Motion where F = 0

Solve the equation of motion using `dsolve` in the case of no external forces where $F = 0$. Use the initial conditions of unit displacement and zero velocity.

```
vel = diff(x,t);
cond = [x(0) == 1, vel(0) == 0];
eq = subs(eq,F,0);
sol = dsolve(eq, cond)
```

Get

sol =

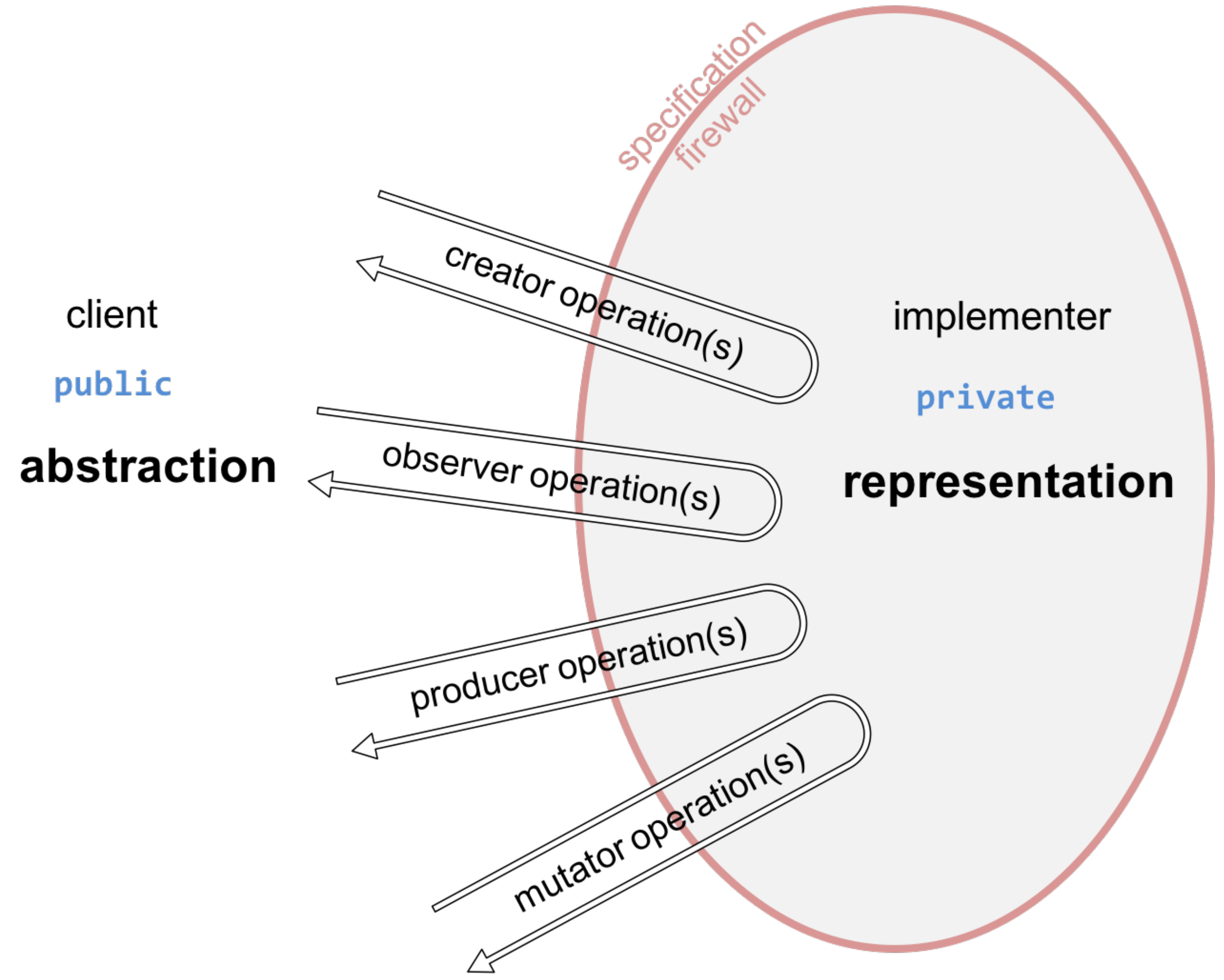
$$\frac{e^{-t\left(\frac{\gamma}{2}-\frac{\sigma_1}{2}\right)}(\gamma+\sigma_1)}{2\,\sigma_1} - \frac{e^{-t\left(\frac{\gamma}{2}+\frac{\sigma_1}{2}\right)}(\gamma-\sigma_1)}{2\,\sigma_1}$$

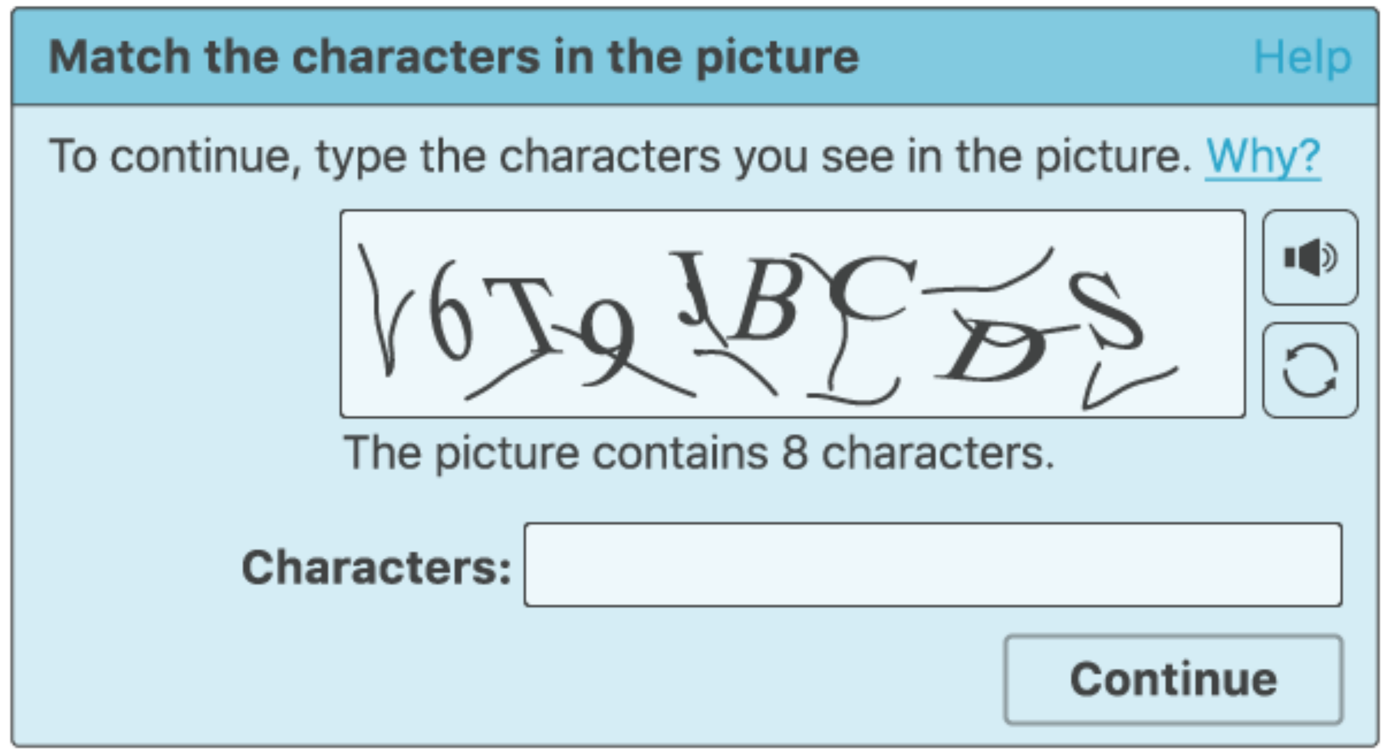
where

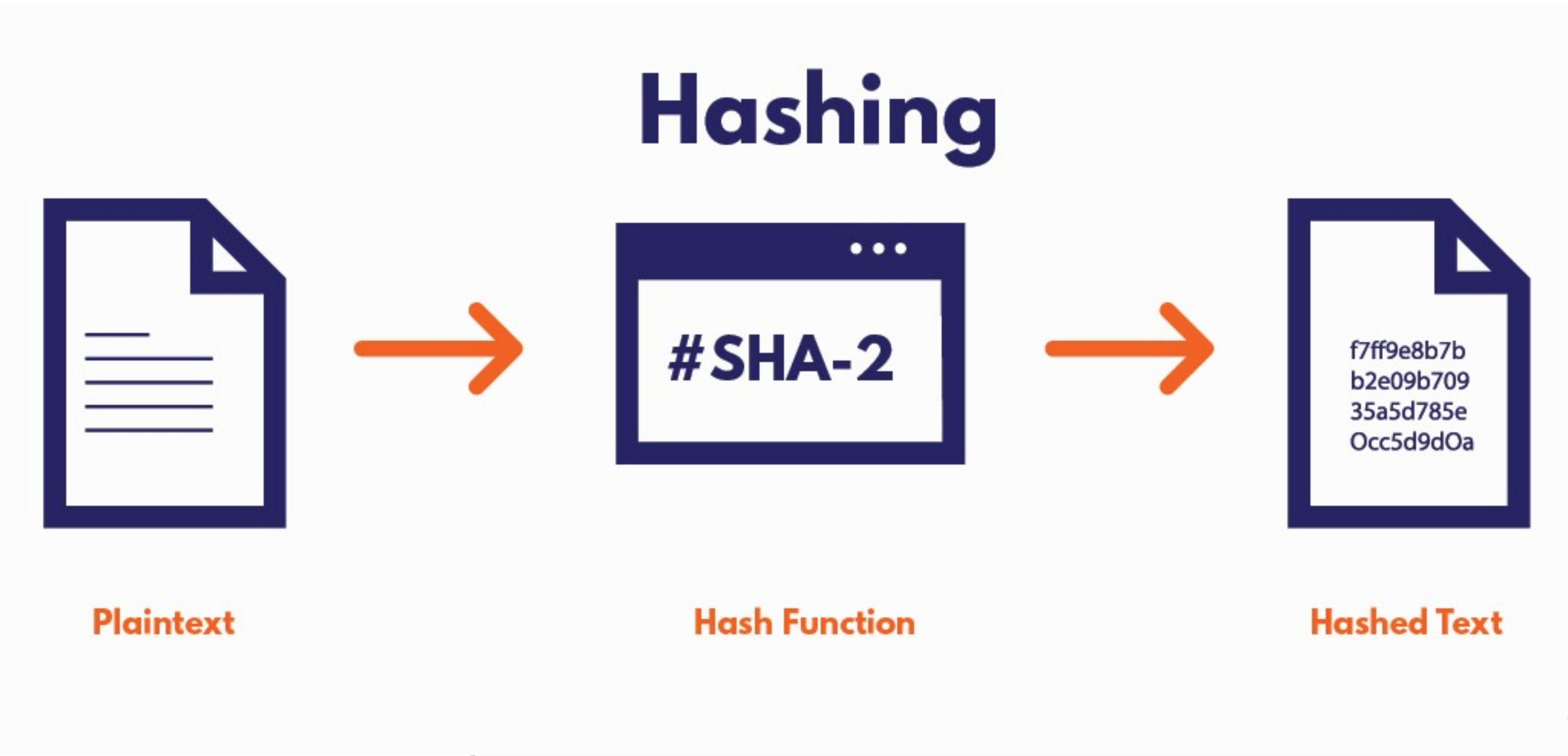
$$\sigma_1 = \sqrt{(\gamma - 2\,\omega_0)\,(\gamma + 2\,\omega_0)}$$

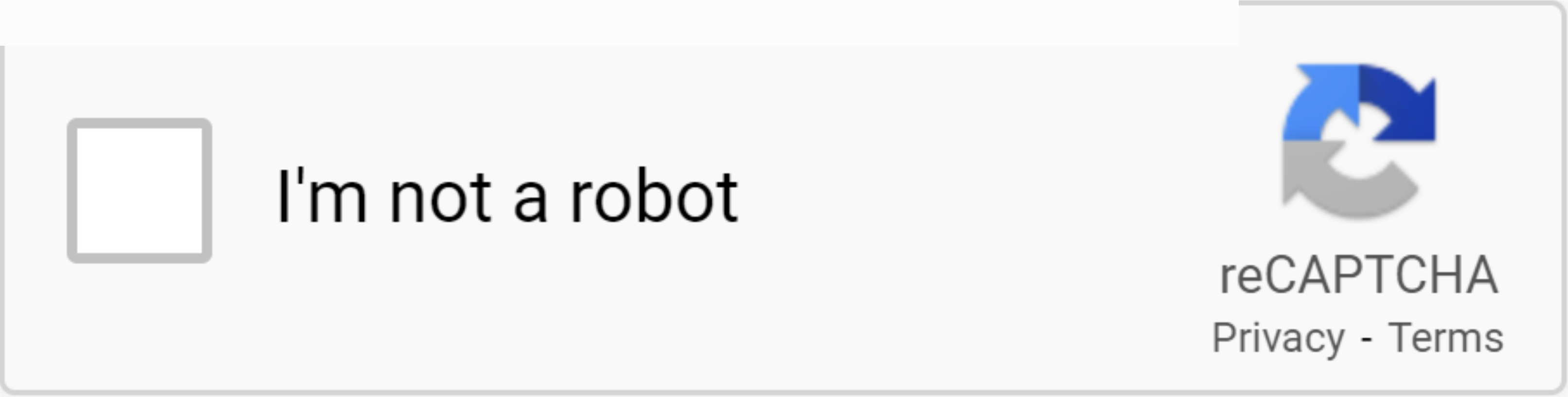
Key Concept IV

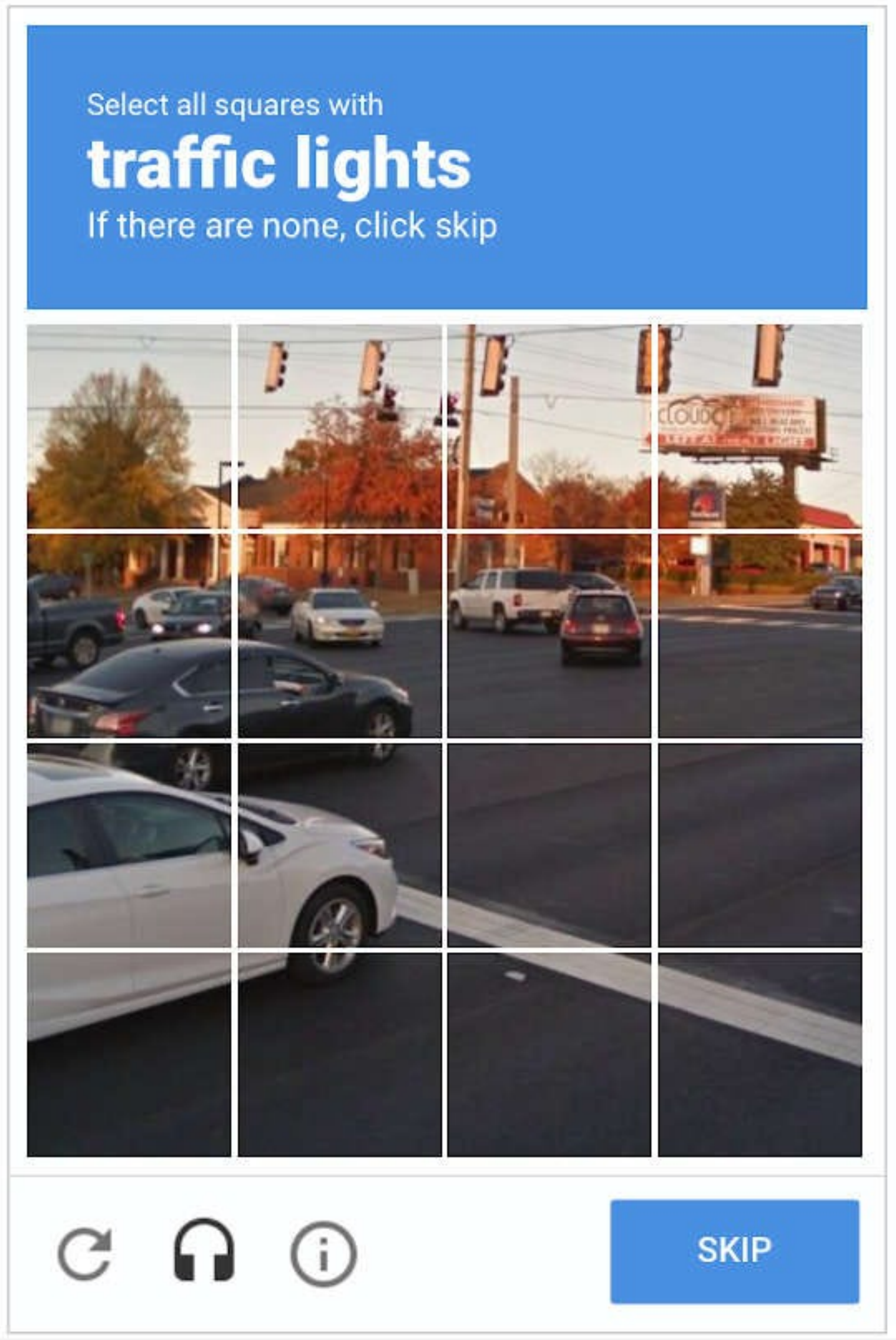
Encapsulation; Representation Independence







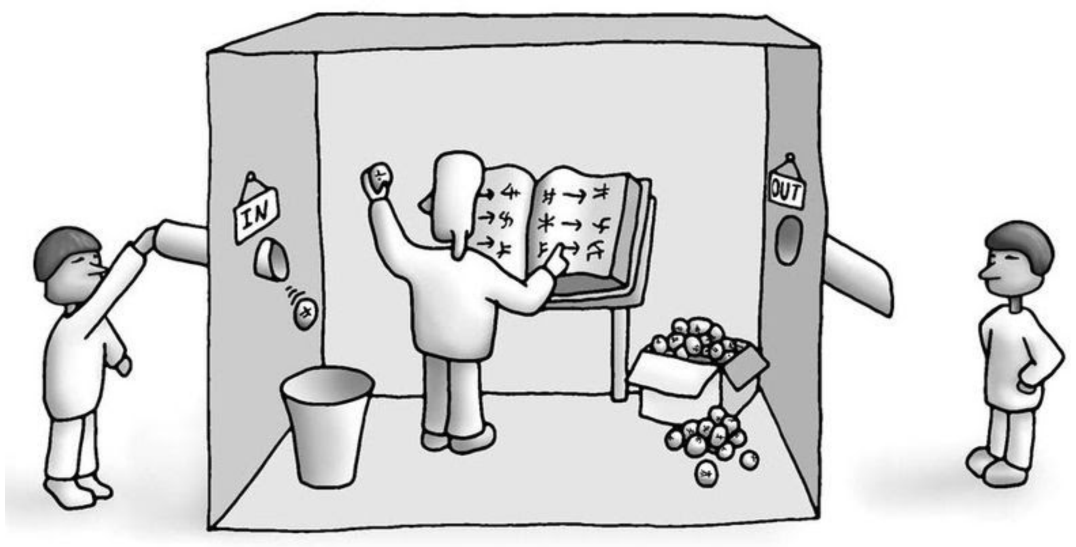




AMAZON / TECH / ARTIFICIAL INTELLIGENCE

Amazon insists Just Walk Out isn't secretly run by workers watching you shop

/ Amazon says human reviewers only annotate shopping data for its cashierless tech.



Key CS Concept V

Complexity Bounds

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Heap Sort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Quick Sort	O(n log(n))	O(n log(n))	$O(n^2)$	O(n)
Merge Sort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Bucket Sort	O(n +k)	O(n +k)	$O(n^2)$	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)

- Sorting a deck of 52 cards:
  - Find the Ace of spades, put it in position 1
  - Find the King of spades, put it in position 2
  - Find the Queen of spaces, put it in position 3
- Worst-case complexity: 52 + 51 + 50 + … = 1378 comparisons
- If deck had N cards, O(N^2)

Quantum Sorting

Theorem 2. *Any comparison-based quantum algorithm for sorting that errs with probability at most* $\epsilon \geq 0$ *requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{N}{2\pi}(H_N - 1) \qquad (2)$$

comparisons. In particular, any exact quantum algorithm requires more than $\frac{N}{2\pi}(\ln(N) - 1) \approx 0.110 N \log_2 N$ *comparisons.*

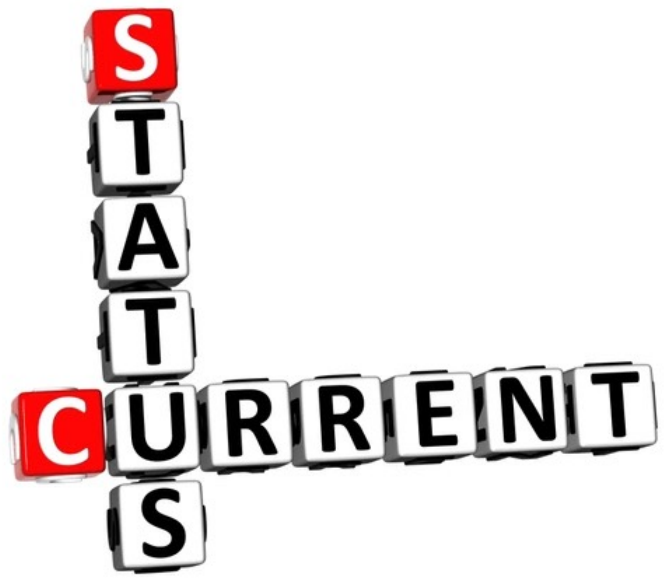
- Quantum advantage for sorting?
- Absolutely not
- There exist other classical sorting algorithms with O($NlogN$) complexity

Integer Factorization

- When the numbers are sufficiently large, no efficient non-quantum integer factorization algorithm is known.

- However, it has not been proven that such an algorithm does not exist.

Current Status



It has been 42 years since Feynman envisioned the use of quantum devices to efficiently simulate physics.

It has been 27 years since Shor developed a quantum polynomial-time prime factorization algorithm.

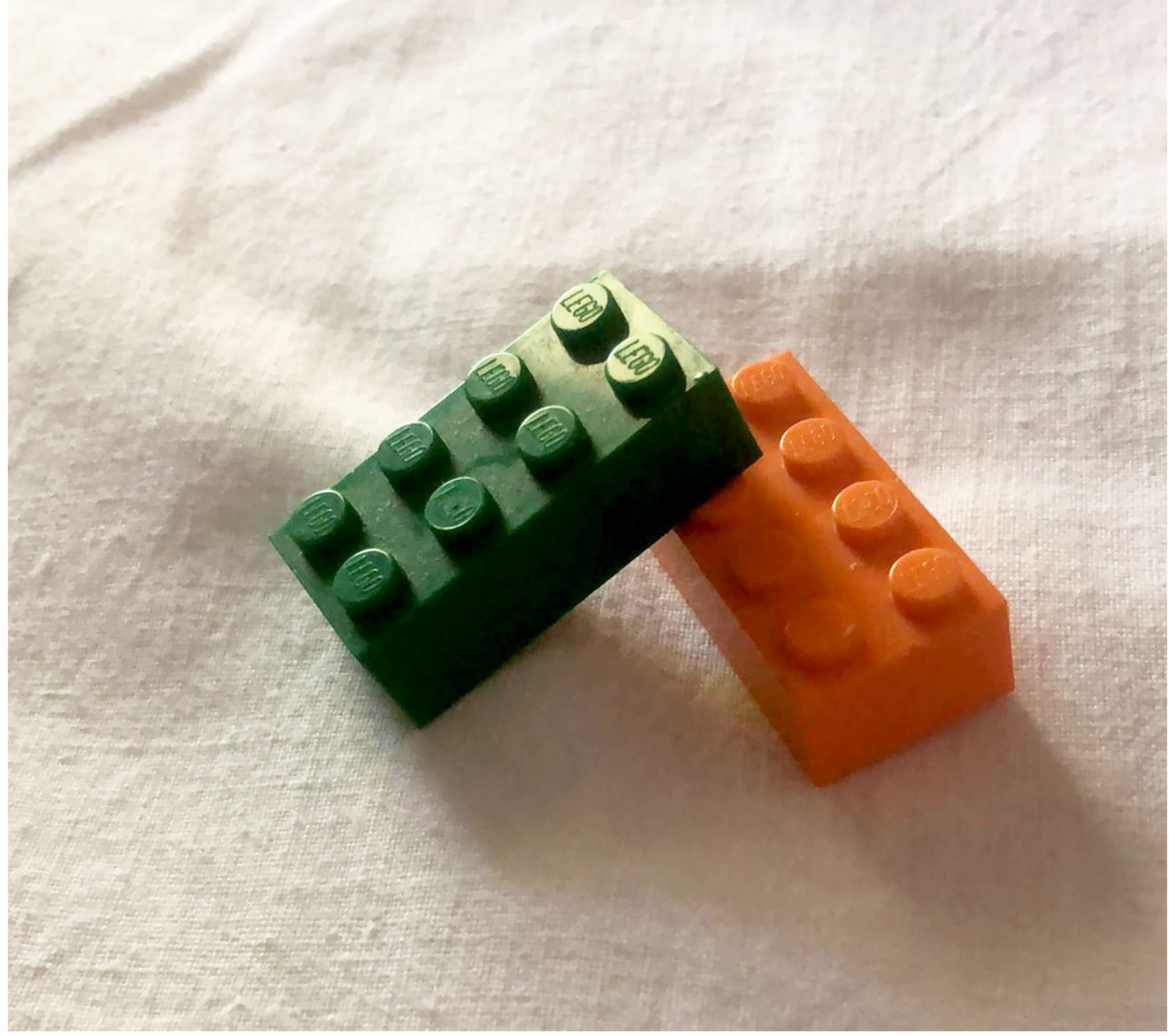
Despite impressive technological advances in the design and realization of quantum devices, there is yet not a single conclusive demonstration of a computational quantum advantage.

Why CS Perspective?

- Pragmatic: Reuse huge computational infrastructure to perform simulations, experiments, explore algorithms, and develop applications.

- Foundational: Examine the boundary between classical and quantum computing to gain insights about potential sources of quantum advantage

- Retrospective: As early as 1992, some CS researchers predicted “a physics revolution is brewing in CS.” Anytime now ???



Everything can be encoded using Toffoli and Hadamard

Formal Result

Theorem 1 (Shi / Aharonov).

The set consisting of just the ***Toffoli*** *and* ***Hadamard*** *gates is computationally universal for quantum computing.*

By computationally universal, *we mean the set can simulate, to within ϵ-error, an arbitrary quantum circuit of n qubits and t gates with only poly-logarithmic overhead in $(n, t, 1/\epsilon)$.*

The Hadamard Mystery

One conclusion:

The difference is all about Hadamard

Or if you prefer:

It's all about QFT (the Quantum Fourier Transform)

Hadamard $\simeq$ QFT

An Approximate Fourier Transform Useful in Quantum Factoring

We define an approximate version of the Fourier transform on 2**L elements, which is computationally attractive in a certain setting, and which may find application to the problem of factoring integers with a quantum computer as is currently under investigation by Peter Shor.

By: *Don Coppersmith*

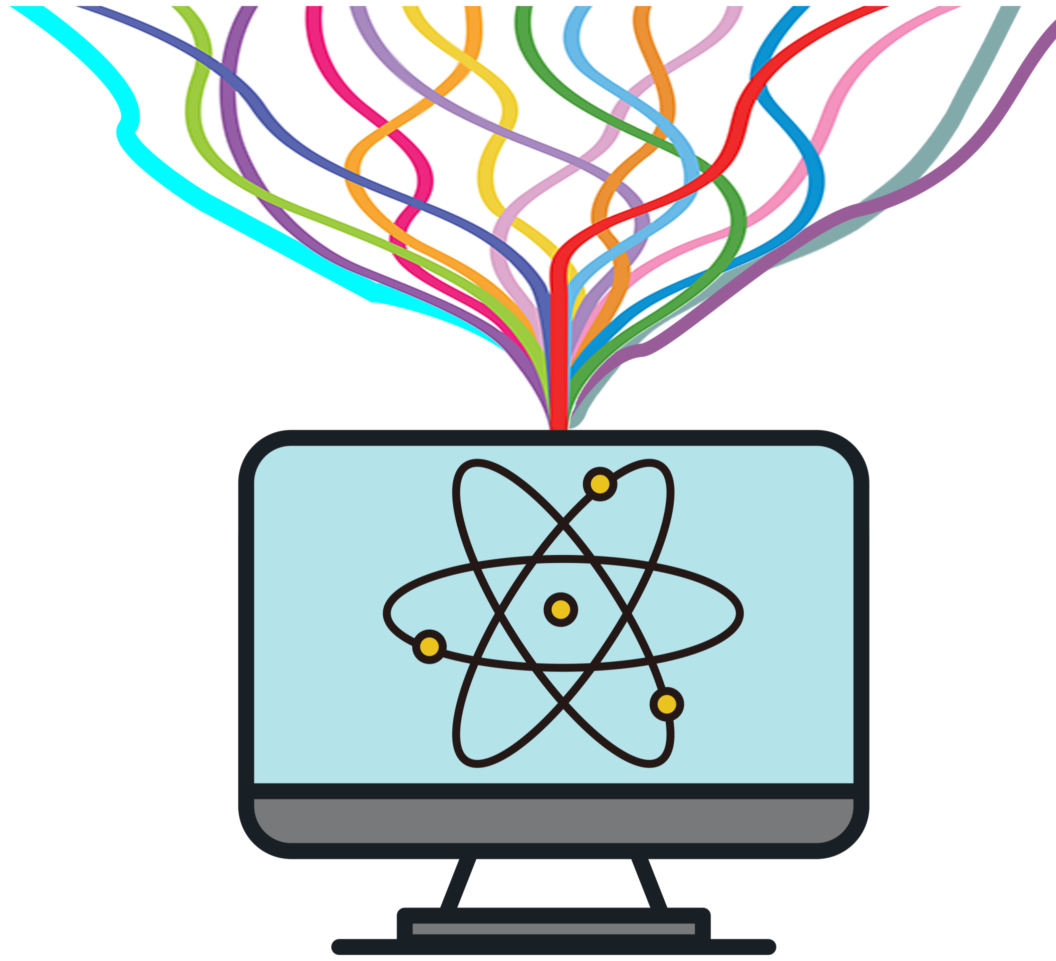
Published in: RC19642 in 1996

Focus on the Essence

- The Toffoli gate (CCX) is just a reference to (reversible) classical computing. Easy!
- The Hadamard gate (H) is a reference to any or all of the following:
    - the (quantum) Fourier transform,
    - a change of basis (from Z basis to X basis and back),
    - a square root of the boolean negation gate (the X gate)
    - or perhaps another perspective?

Plan

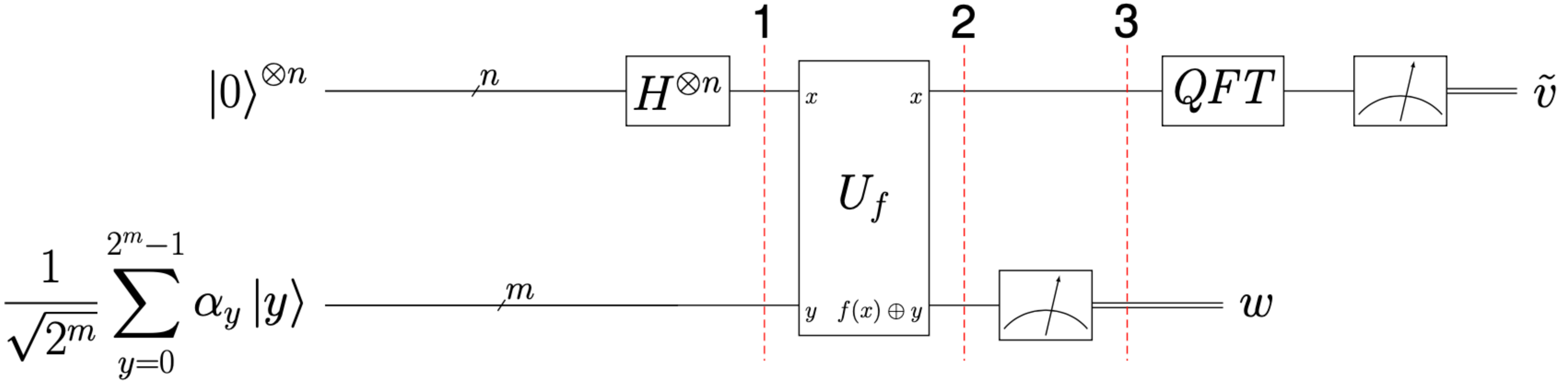
- Start with a “good” model of reversible classical computing
- Explore ways to express Hadamard-like functionality



Textbook Quantum Algorithms

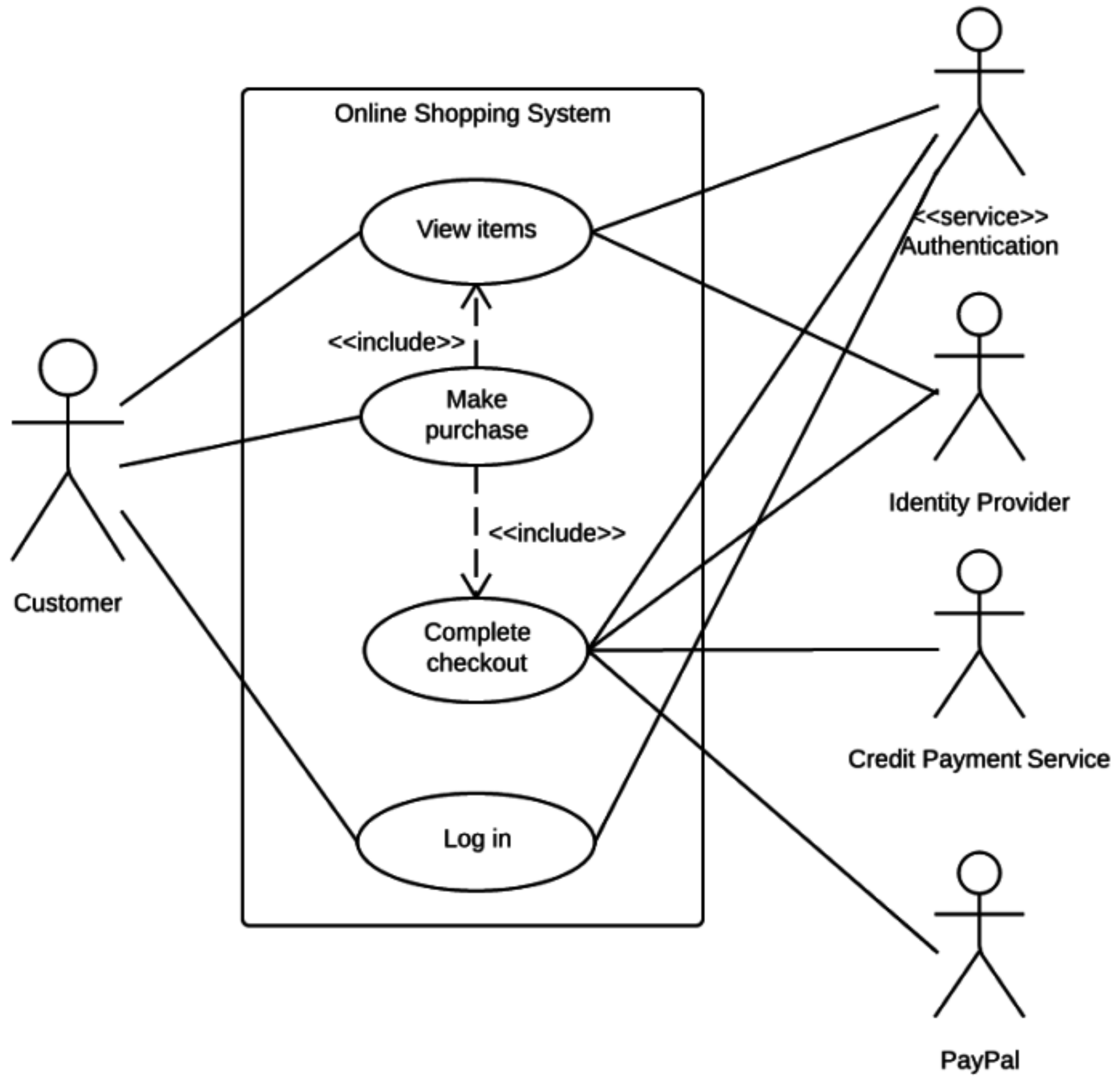
Circuits for Hidden Subgroup Problems

Class includes Deutsch-Jozsa, Bernstein-Vazirani, Simon, Grover and Shor algorithms



- Hadamard only after initialization
- Hadamard on $|0\rangle$ only
- QFT (generalized Hadamard) only before measurement

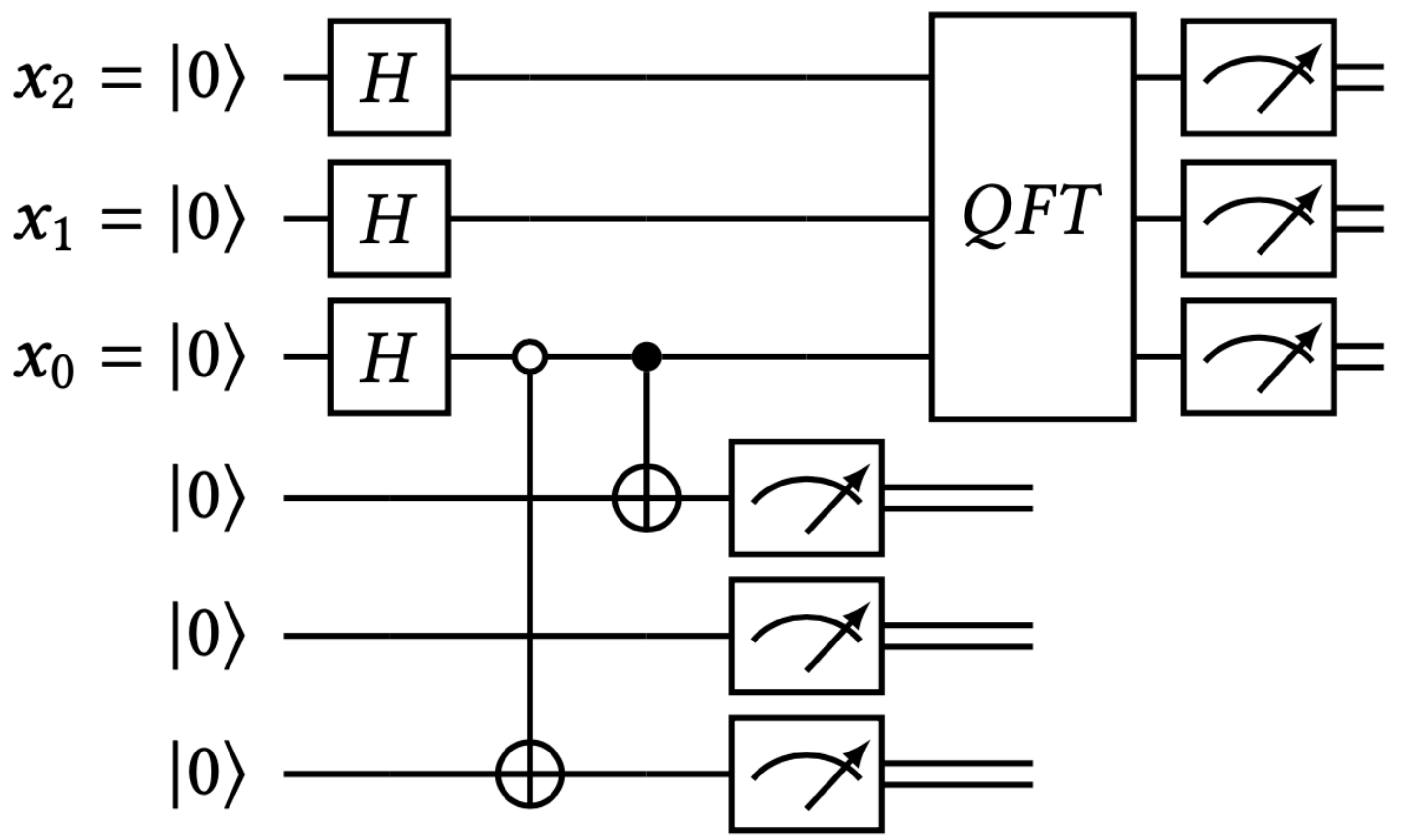
How Hadamard is actually used



- After initialization to introduce a uniform superposition
- Before measurement to extract spectral properties
- No uses of Hadamard in the middle !

Example

Factor 15 by computing period of $4^x \mod 15$



$$|000000\rangle + |001000\rangle + |010000\rangle + |011000\rangle + |100000\rangle + |101000\rangle + |110000\rangle + |111000\rangle$$

$$\Rightarrow$$

$$|000001\rangle + |001000\rangle + |010001\rangle + |011000\rangle + |100001\rangle + |101000\rangle + |110001\rangle + |111000\rangle$$

$$\Rightarrow$$

$$|000001\rangle + |001100\rangle + |010001\rangle + |011100\rangle + |100001\rangle + |101100\rangle + |110001\rangle + |111100\rangle$$

$$=$$

$$(|000001\rangle + |010001\rangle + |100001\rangle + |110001\rangle) \quad + \quad (|001100\rangle + |011100\rangle + |101100\rangle + |111100\rangle)$$

Bottom 3 qubits can be measured as:

001 so input to QFT = $|000\rangle + |010\rangle + |100\rangle + |110\rangle$ **(period = 2)**

100 so input to QFT = $|001\rangle + |011\rangle + |101\rangle + |111\rangle$ **(period = 2)**

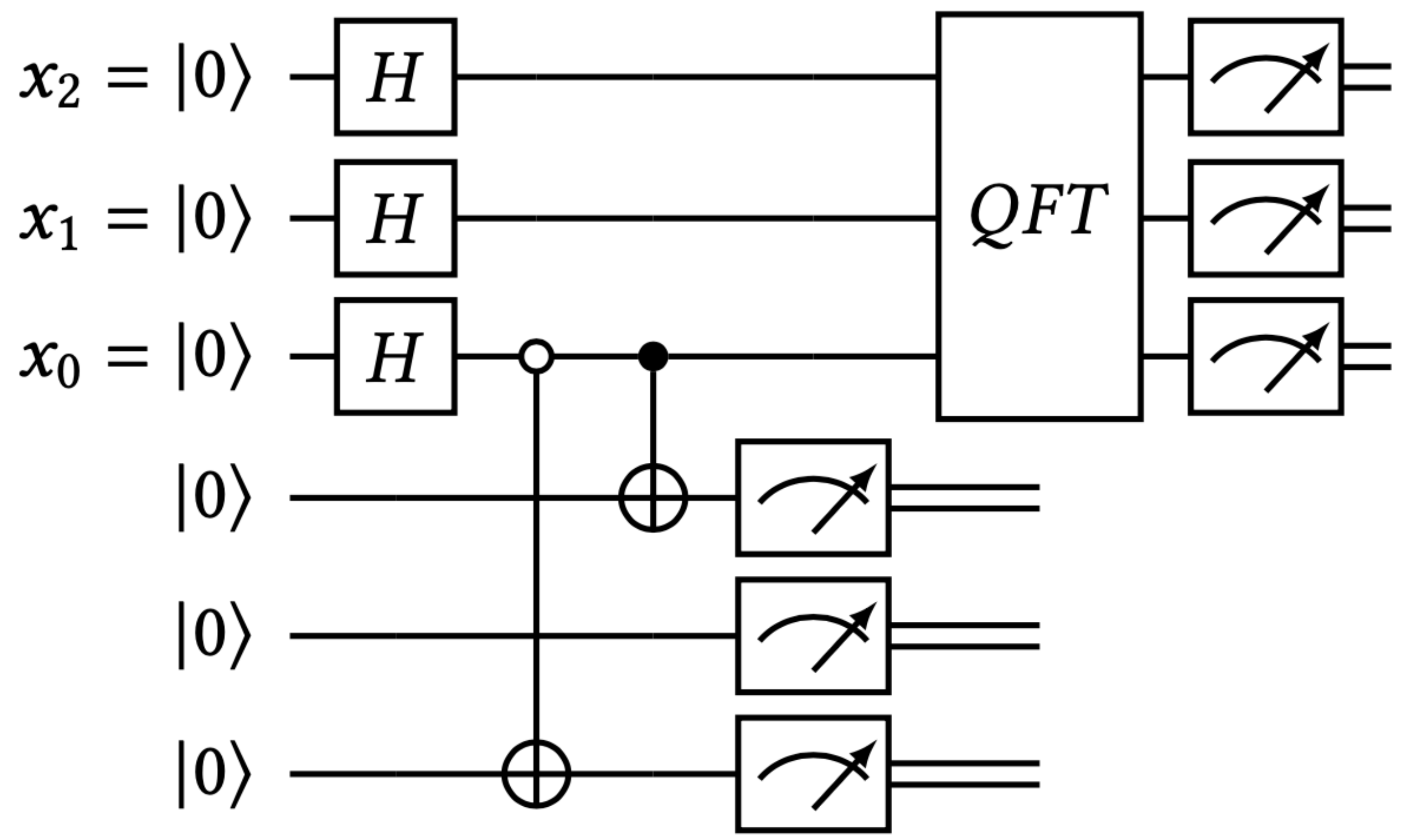
Symbolic Execution ?

$$
\begin{array}{rcl}
(\neg\neg x) & \rightsquigarrow & x \\
(\neg(x \vee y)) & \rightsquigarrow & ((\neg x) \wedge (\neg y)) \\
(\neg(x \wedge y)) & \rightsquigarrow & ((\neg x) \vee (\neg y)) \\
(x \wedge (y \vee z)) & \rightsquigarrow & ((x \wedge y) \vee (x \wedge z)) \\
((x \vee y) \wedge z) & \rightsquigarrow & ((x \wedge z) \vee (y \wedge z))
\end{array}
$$

- $H|0\rangle$ creates a unknown boolean variable
- We can compute symbolically, e.g.,
    - $CX(a, b) = (a, a \oplus b)$
- Initial and final conditions will constrain the variable

Example: symbolic execution

Factor 15 by computing period of $4^x \mod 15$



$|x_2x_1x_0001\rangle$

$\Leftarrow$

$|x_2x_1x_0x_001\rangle$

$\Leftarrow$

$|x_2x_1x_0x_00x_0\rangle$

Boundary conditions:

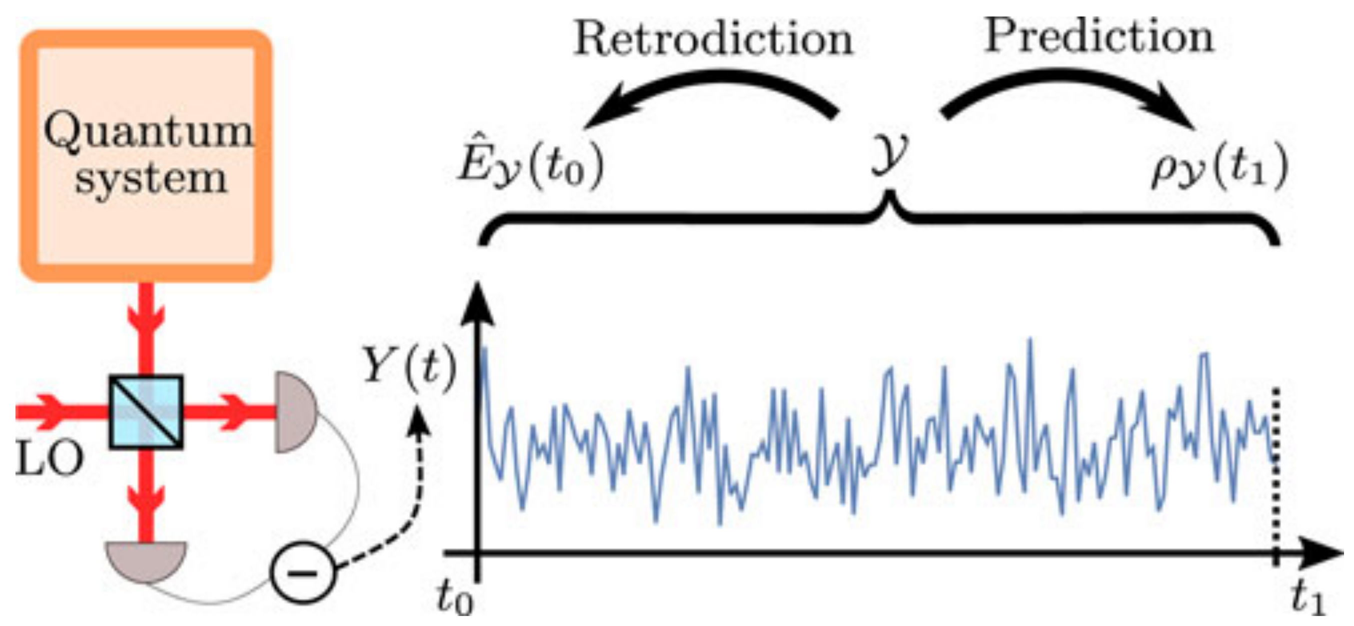
- $x_0 = 0$
- $0 = 0$
- $x_0 = 0$

Input to QFT:

x_2x_10

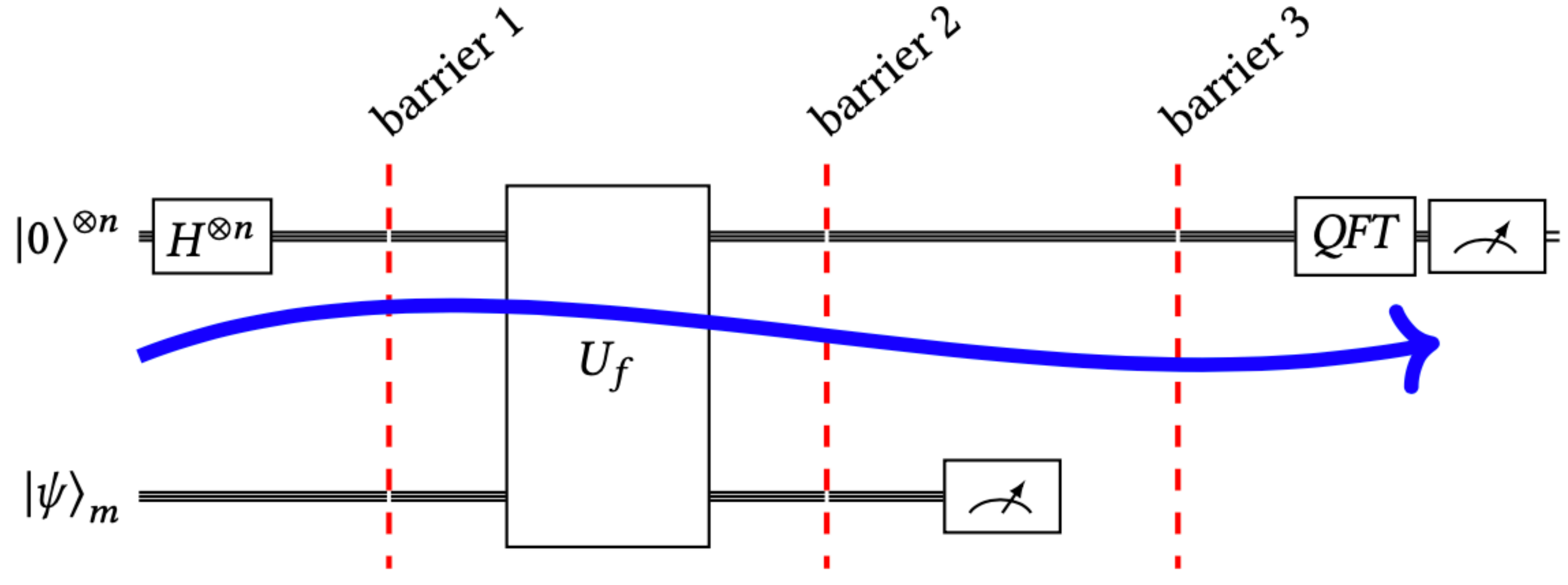
Period is 2
(even numbers)

Retrodictive Classical Execution

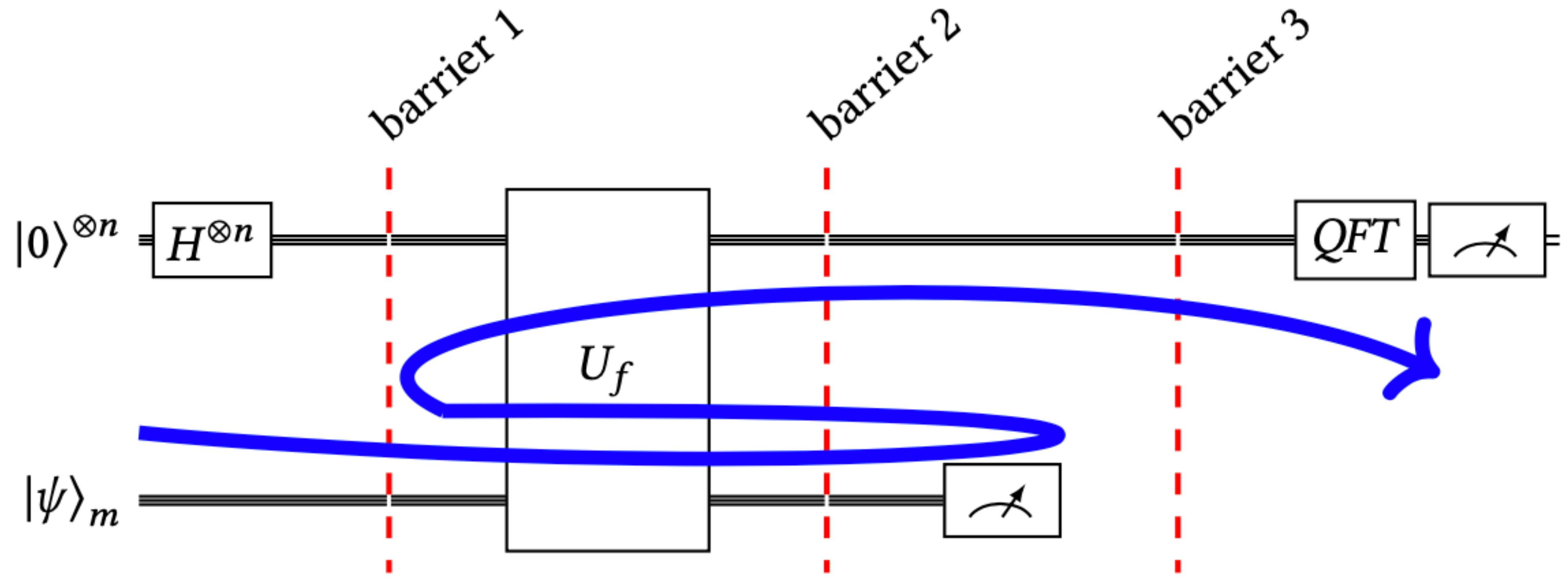


Instead of conventional forward execution:

- Run with one fixed input to determine a possible value for output register
- Run backwards with symbols for input register
- Use initial conditions to constrain symbolic values



(a) Conventional Flow



(b) Retrodictive Flow

```
> runRetroShor Nothing (Just 4) (Just 1) 15
n=8; a=4

Generalized Toffoli Gates with 3 controls = 2916
Generalized Toffoli Gates with 2 controls = 27378
Generalized Toffoli Gates with 1 controls = 26244
```

$$1 \oplus x_0 = 1$$
$$x_0 = 0$$

```
> runRetroShor Nothing Nothing (Just 1) 15
n=8; a=11

Generalized Toffoli Gates with 3 controls = 2916
Generalized Toffoli Gates with 2 controls = 27378
Generalized Toffoli Gates with 1 controls = 26244

x0 = 0
x0 = 0
```

```
> runRetroShor Nothing Nothing (Just 1) 51
n=12; a=37

Generalized Toffoli Gates with 3 controls = 8788
Generalized Toffoli Gates with 2 controls = 86866
Generalized Toffoli Gates with 1 controls = 81796
```

$$1 \oplus x_0 \oplus x_2 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_3 \oplus x_0x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_2x_3 = 1$$
$$x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_0x_1x_2x_3 = 0$$
$$x_0x_1 \oplus x_2 \oplus x_1x_2 \oplus x_0x_3 \oplus x_0x_1x_3 = 0$$
$$x_0 \oplus x_0x_2 \oplus x_1x_2 \oplus x_0x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3 = 0$$
$$x_1 \oplus x_0x_1 \oplus x_0x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_2x_3 \oplus x_0x_1x_2x_3 = 0$$
$$x_0 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_2x_3 = 0$$

Boolean + Fourier: Classic CS topic

Connections to learning; many roadblocks and open problems

CSE 291 - Fourier analysis of boolean functions (Winter 2017)

Time: Mondays & Wednesdays 5:00-6:20pm
Room: CSE (EBU3B) 4258
Instructor: Shachar Lovett, email: slovett@ucsd.edu

Overview:

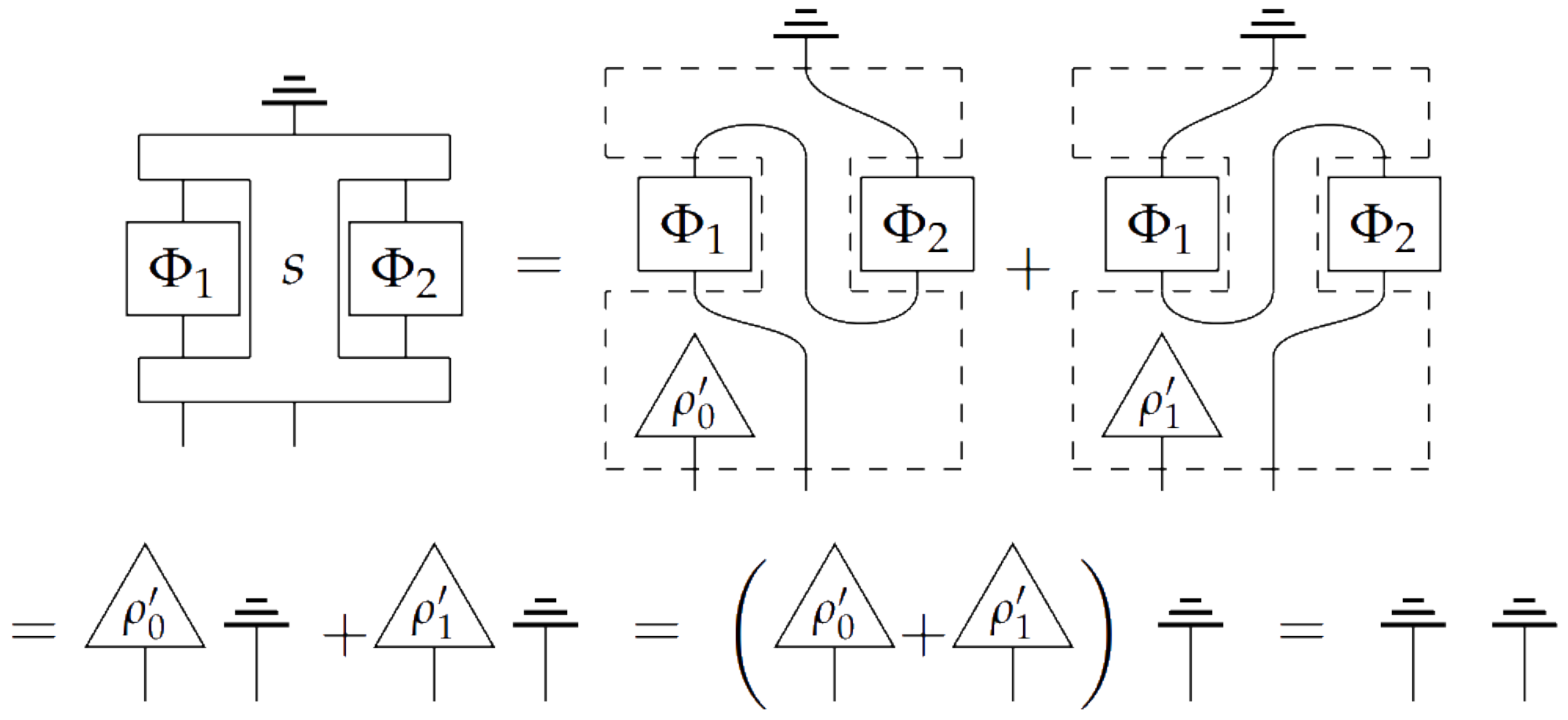
Fourier analysis is a powerful tool used to study boolean f applications in computer science, for example in learning theor cryptography, complexity theory and more. This class will mathematical background, as well as explore many applications.

COMP 760 (Fall 2011): Harmonic Analysis of Boolean Functions

Instructor's contact: See Here
Lectures: MW 11:35-12:55 in McConnell Engineering Building 103 (Starting from tomorrow, Wednesday, the class is 11:35-12:55)
Office Hours: By appointment (hatami at cs mcgill ca)

Course description:

This course is intended for graduate students in theoretical computer science or mathematics. Its purpose is to study Boolean functions via Fourier analytic tools. This analytic approach plays an essential role in modern theoretical computer science and combinatorics (e.g. in circuit complexity, hardness of approximation, machine learning, communication complexity, graph theory), and it is the key to understanding many fundamental concepts such as pseudo-randomness.



Characterize H using Categorical Semantics

Abstract Data Type: Bool

Public state v = false

Public interface { false, true, not, copy, inv copy, ...}

Hidden representation

true	1
false	0
not	v = v + 1 mod 2
copy	return (v,v)
inv copy (v,w)	$\begin{cases} \text{return v} & \text{if } v = w \\ \text{undefined} & \text{otherwise} \end{cases}$

...

Abstract Data Type: Bool

Public state v = false

Public interface { false, true, not, copy, inv copy, …}

Hidden representation

true	0
false	1
not	v = v + 1 mod 2
copy	return (v,v)
inv copy (v,w)	$\begin{cases} \text{return } v & \text{if } v = w \\ \text{undefined} & \text{otherwise} \end{cases}$

…

Abstract Data Type: Bool

Public state v = false

Public interface { false, true, not, copy, inv copy, …}

Hidden representation

true	2
false	0
not	v = v + 2 mod 4
copy	return (v,v)
inv copy(v,w)	$\begin{cases} \text{return v} & \text{if } v = w \\ \text{undefined} & \text{otherwise} \end{cases}$

…

Abstract Data Type: Bool

Public state v = false

Public interface { false, true, not, copy, inv copy, …}

Hidden representation

true	$\vert 1\rangle$
false	$\vert 0\rangle$
not	v = Xv
copy	return v $\otimes$ v
inv copy (v,w)	$\begin{cases} \text{return v} & \text{if v} = \text{w} \\ \text{undefined} & \text{otherwise} \end{cases}$

…

Abstract Data Type: Bool

Public state $v = \text{false}$

Public interface { false, true, not, copy, inv copy, …}

Hidden representation

true	$\vert - \rangle$
false	$\vert + \rangle$
not	$v = Zv$
copy	return $v \otimes v$
inv copy (v,w)	$\begin{cases} \text{return } v & \text{if } v = w \\ \text{undefined} & \text{otherwise} \end{cases}$

…

...bstract Data Type: Bool

Public state

Public interface ... ot, copy...

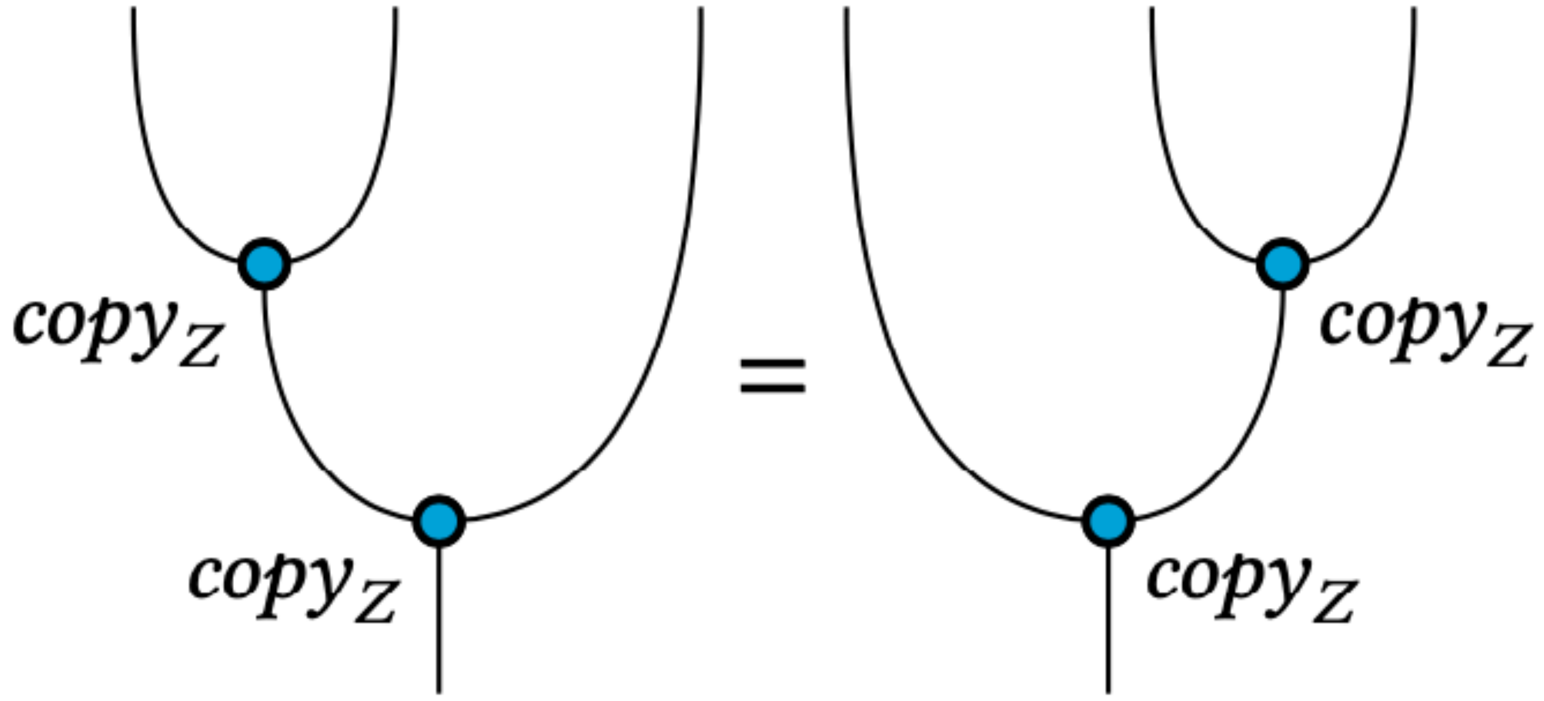
Hidden representatio...

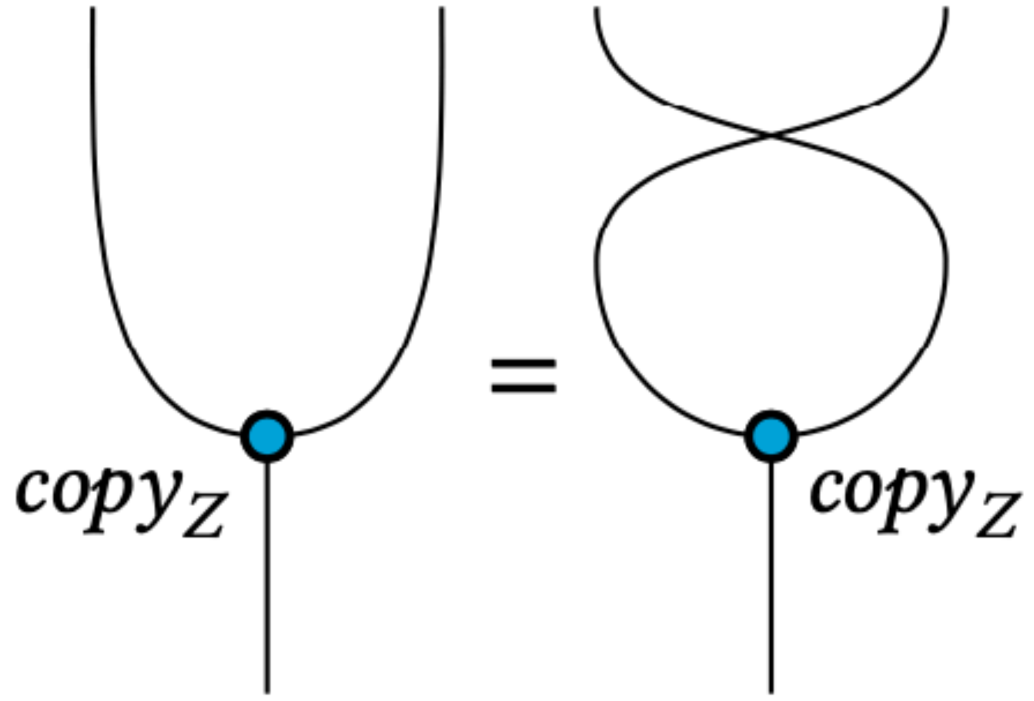
true	0		
false	1		
not	v = v...		
copy			
inv copy (v,w)		...v	if v = w
		...ined	otherwise

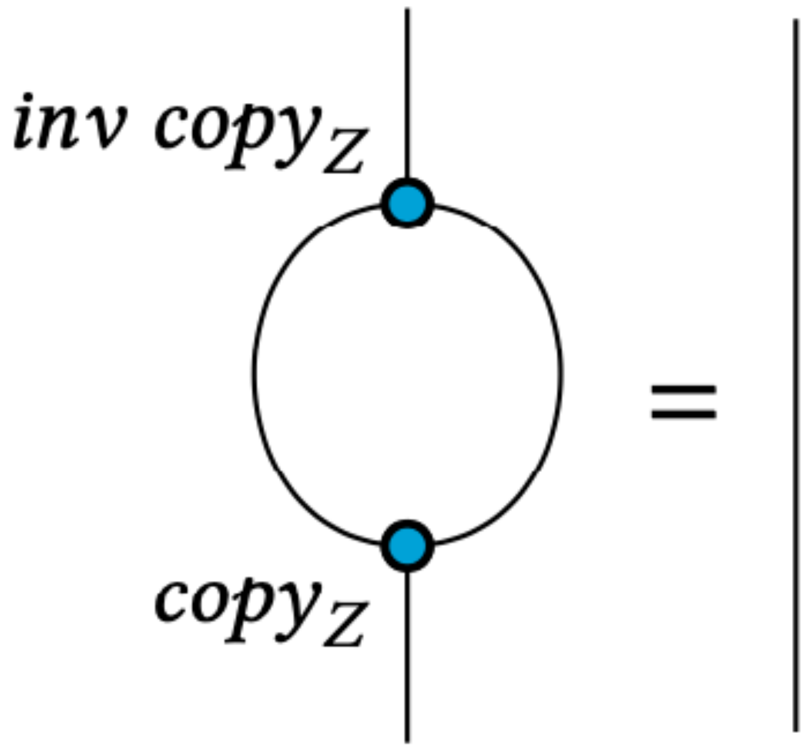
...

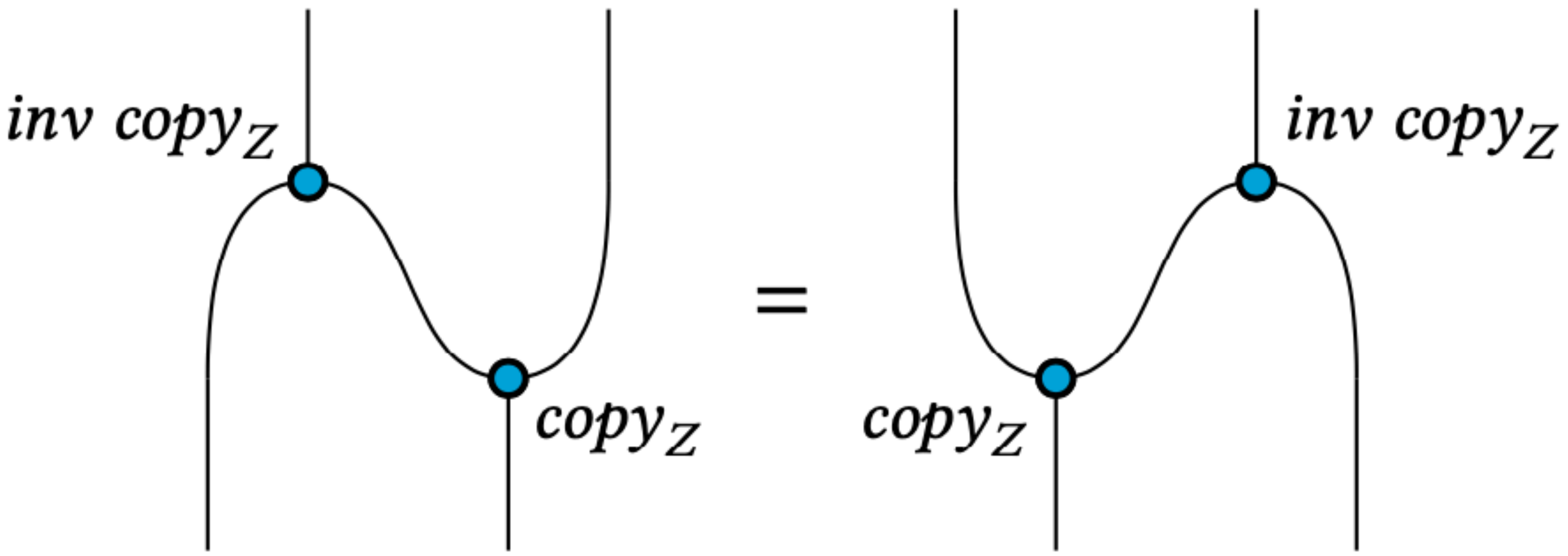
Hidden Implementation must satisfy

Equations I



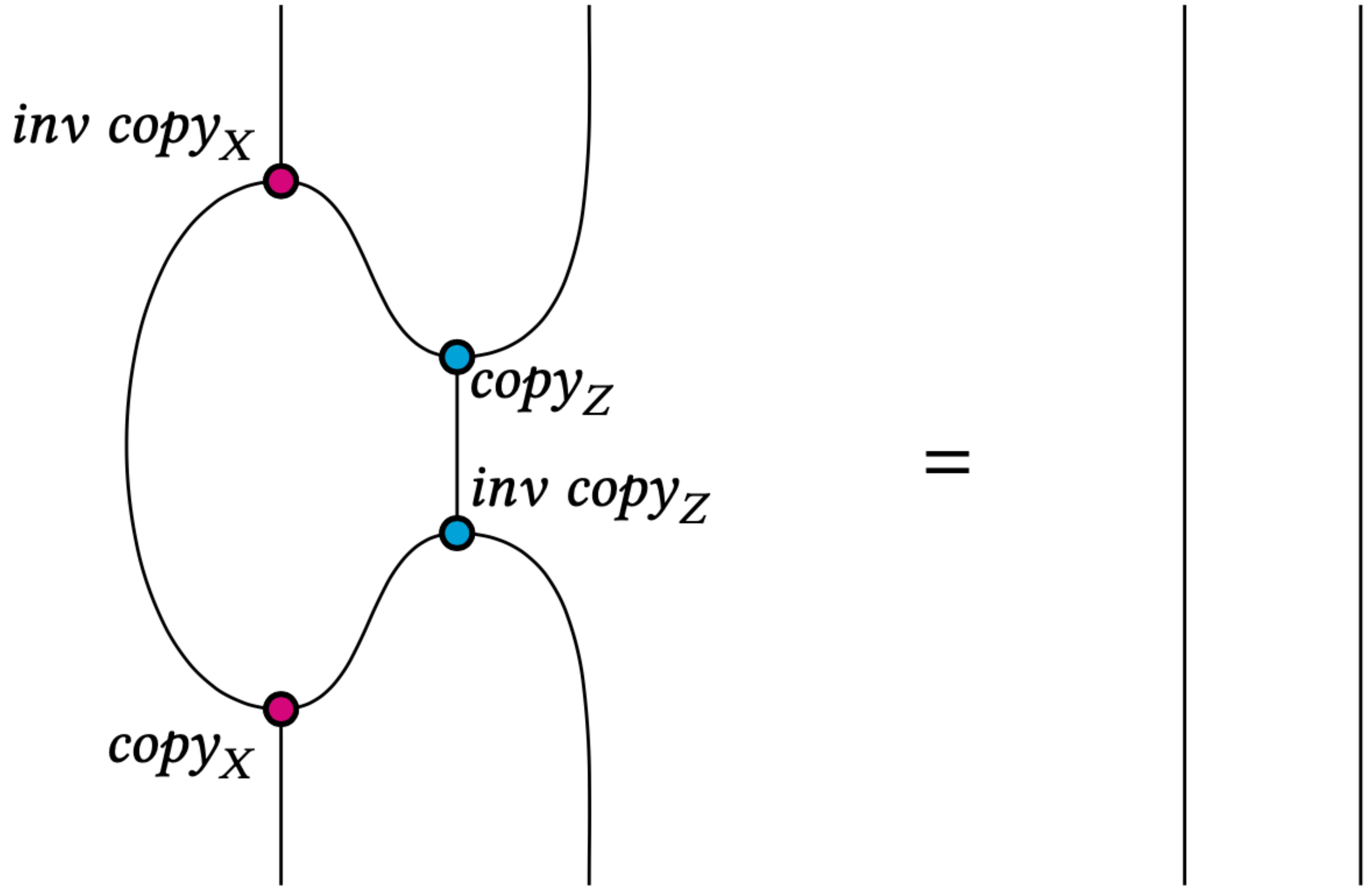




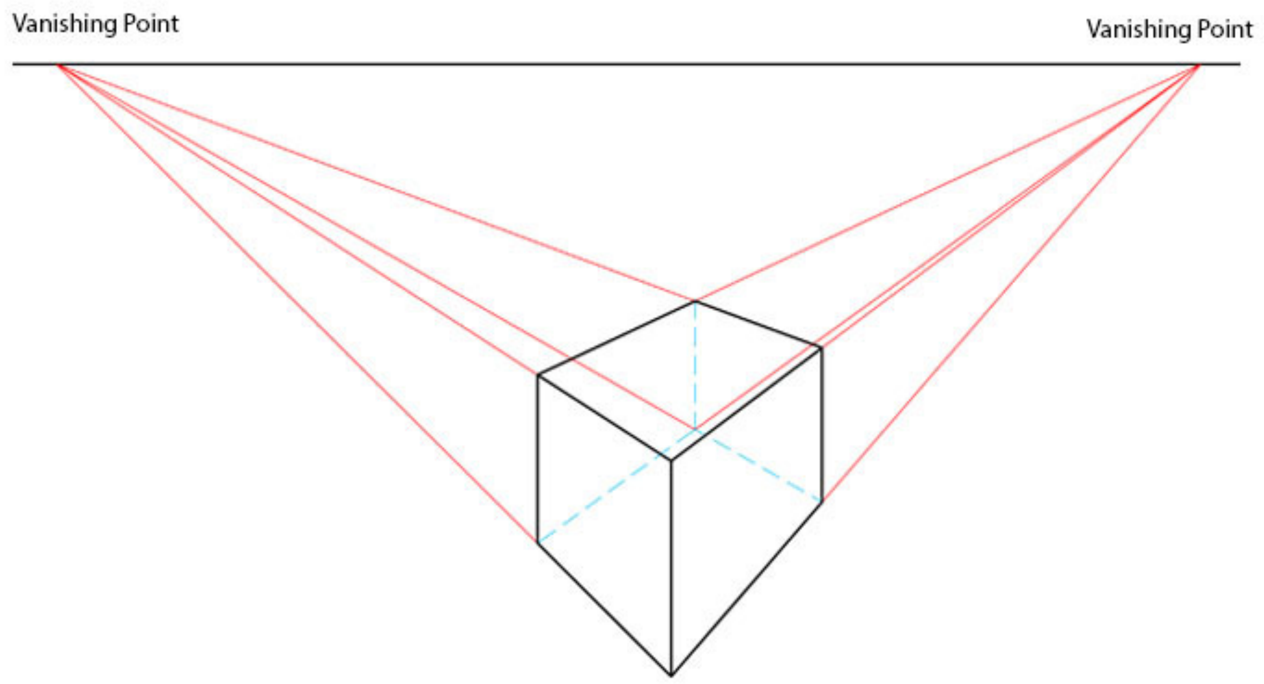


Hidden Implementation must satisfy

Equation II



It is Quantum !

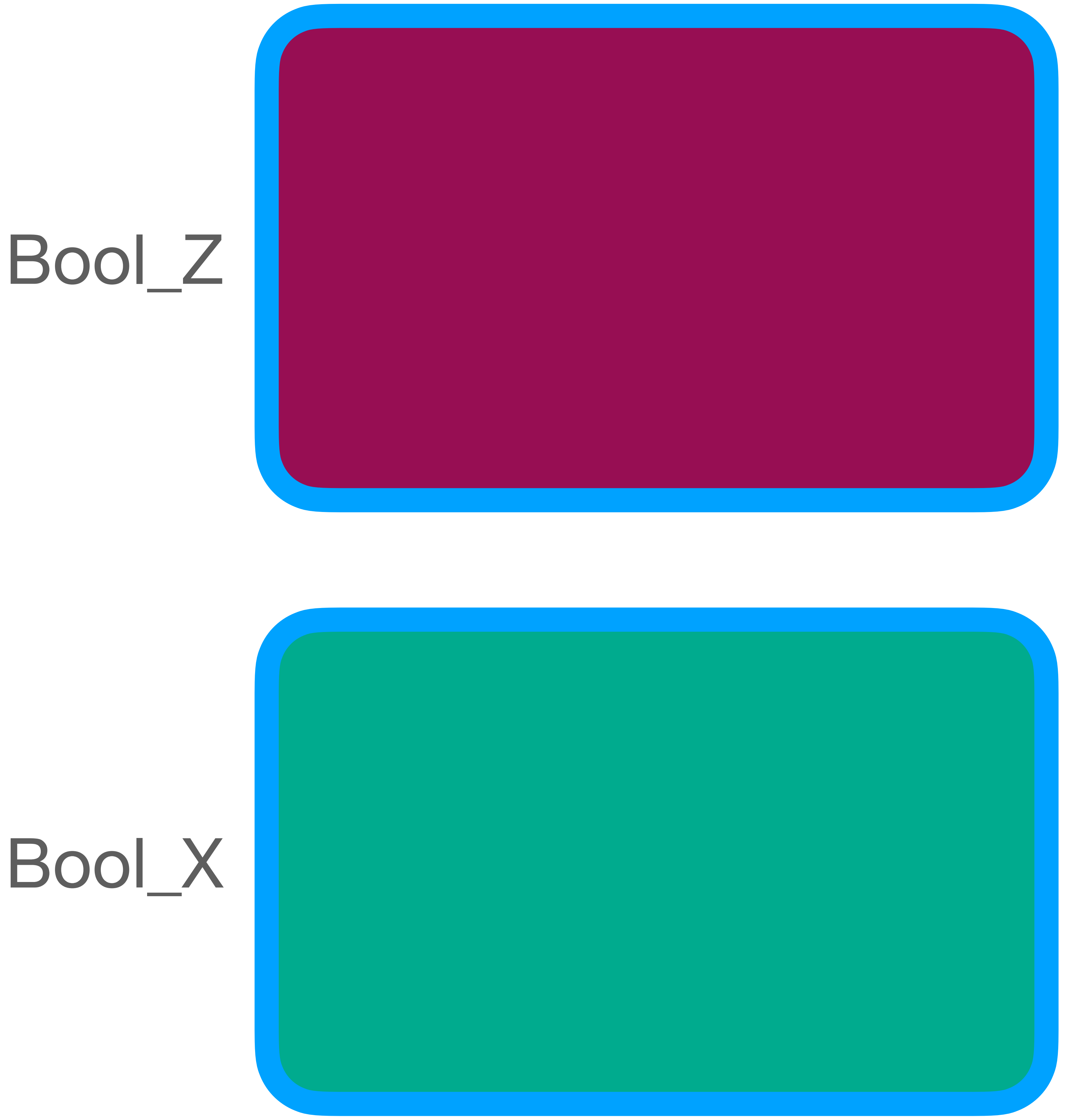


THEOREM 27 (CANONICITY). *If a categorical semantics* $[\![-]\!]$ *for* $\langle \Pi ⅁ \rangle$ *in* **Contraction** *satisfies the classical structure laws and the execution laws (defined in Prop. 24) and the complementarity law (Def. 26), then it must be the semantics of Sec. 7.3 with the semantics of* x_ϕ *being the Hadamard gate and:*

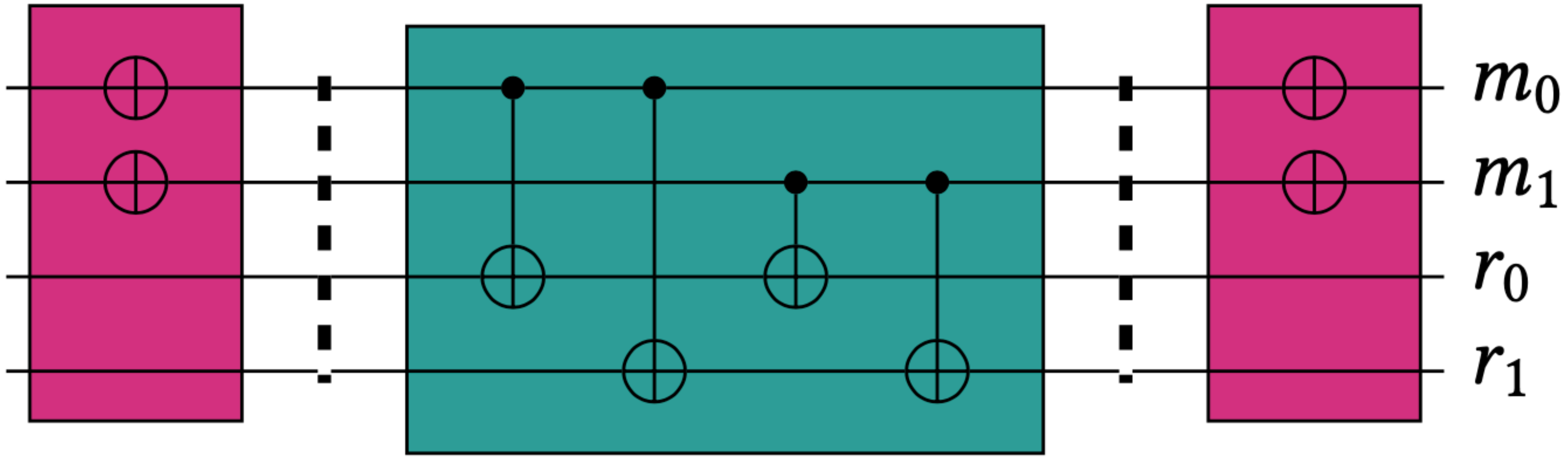
$$[\![copy_Z]\!]: \ |i\rangle \mapsto |ii\rangle \qquad\qquad [\![zero]\!] = |0\rangle$$

$$[\![copy_X]\!]: \ |\pm\rangle \mapsto |\pm\pm\rangle \qquad\qquad [\![assertZero]\!] = \langle 0|$$

Two instances of ADT Bool with unknown representation but constrained to satisfy some equations

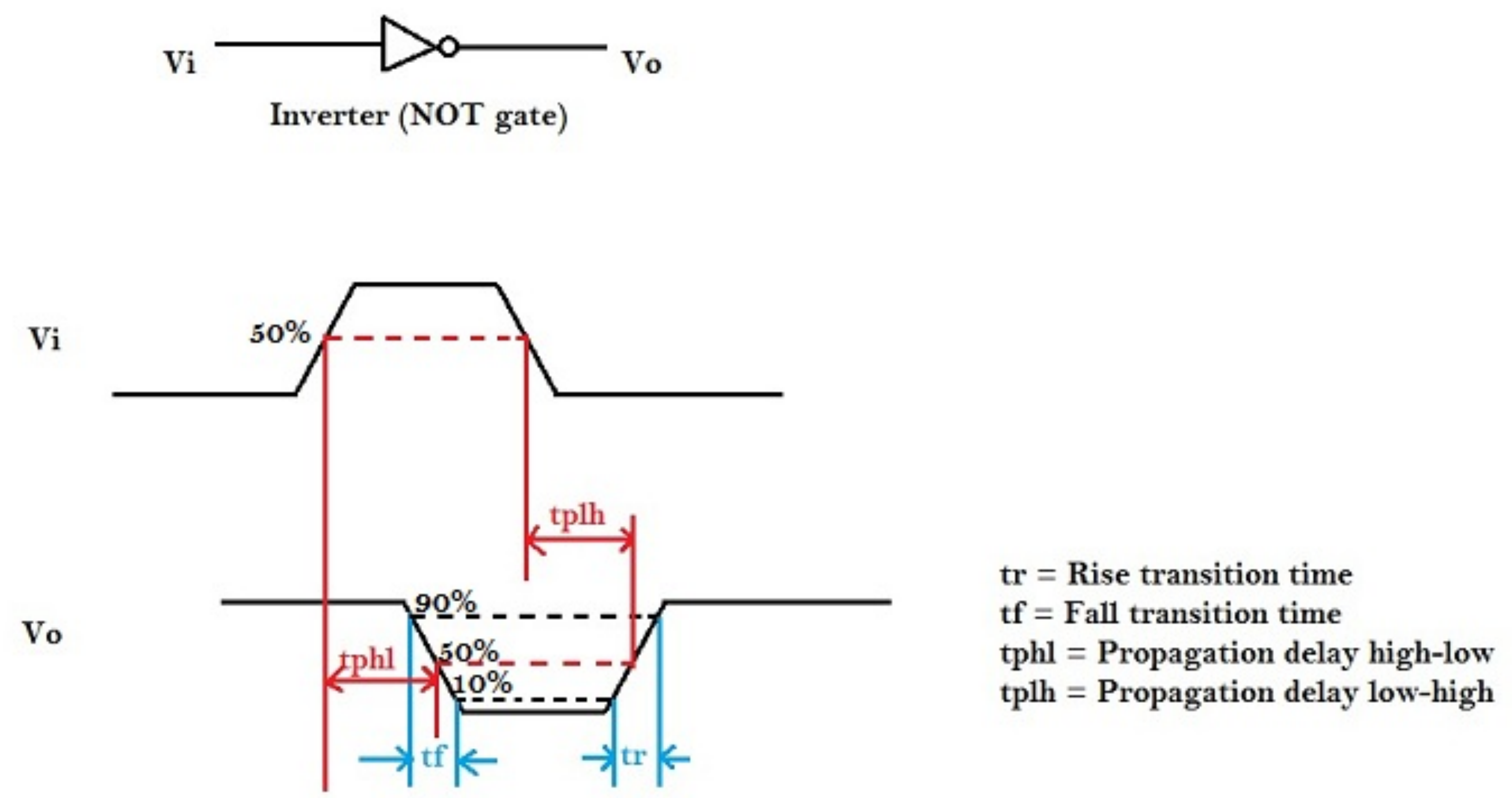


Allow interleaving of the two languages



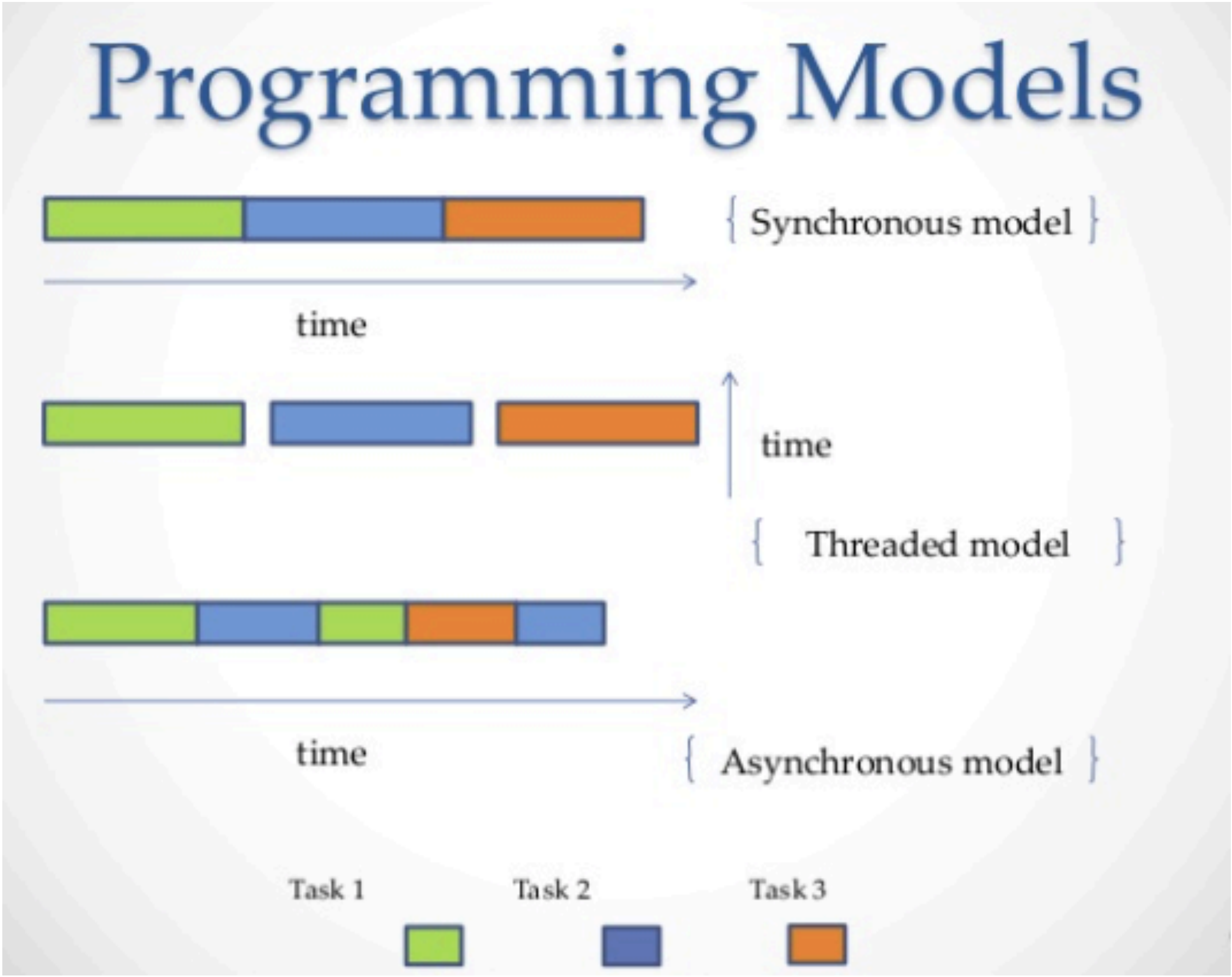
Hadamard from Square Roots

Clocked Digital Computation



- Simplified view of processor
- Clock defines smallest unit of time
- Every operation takes one or more clock cycle
- In particular, boolean negation takes one clock cycle

Half a clock cycle?



- What if we split the action of the NOT gate in two steps
- Some operations take a full clock cycle
- Some take half a clock cycle
- Allow asynchronous interleaving

Formally…

Take a reversible classical programming language, extend it with:

Syntax

$iso ::= \cdots \mid \mathrm{v} \mid \mathrm{v\textsc{i}} \mid \mathrm{w} \mid \mathrm{w\textsc{i}}$ (isomorphisms)

Types

$$\begin{array}{ccccccc} \mathrm{v} & : & \mathbb{2} & \leftrightarrow & \mathbb{2} & : & \mathrm{v\textsc{i}} \\ \mathrm{w} & : & \mathbb{1} & \leftrightarrow & \mathbb{1} & : & \mathrm{w\textsc{i}} \end{array}$$

Equations

(E1) $\mathrm{v}^2 \leftrightarrow_2 \mathrm{x}$

(E2) $\mathrm{w}^8 \leftrightarrow_2 1$

(E3) $\mathrm{v} \fatsemi (id + \mathrm{w}^2) \fatsemi \mathrm{v} \leftrightarrow_2 uniti^{\times}l \fatsemi \mathrm{w}^2 \times ((id + \mathrm{w}^2) \fatsemi \mathrm{v} \fatsemi (id + \mathrm{w}^2)) \fatsemi unite^{\times}l$

It's Quantum again

Definition of the Quantum Model. The model consists of a rig groupoid $(\mathrm{C}, \otimes, \oplus, O, I)$ equipped with maps $\omega\colon I \longrightarrow I$ and $\mathrm{V}\colon I \oplus I \longrightarrow I \oplus I$ satisfying the equations:

$$\text{(E1)}\ \omega^8 = \mathrm{id} \qquad \text{(E2)}\ \mathrm{V}^2 = \sigma_\oplus \qquad \text{(E3)}\ \mathrm{V} \circ \mathrm{S} \circ \mathrm{V} = \omega^2 \bullet \mathrm{S} \circ \mathrm{V} \circ \mathrm{S}$$

where $\circ$ is sequential composition, $\bullet$ is scalar multiplication (cf. Def. 4), $\sigma_\oplus$ is the symmetry on $I \oplus I$, exponents are iterated sequential compositions, and $\mathrm{S}\colon I \oplus I \longrightarrow I \oplus I$ is defined as $\mathrm{S} = \mathrm{id} \oplus \omega^2$.

Theorem 25 (Full abstraction for Gaussian Clifford+T circuits). *Let c_1 and c_2 be $\sqrt{\Pi}$ terms representing Gaussian Clifford+T circuits. Then $[\![c_1]\!] = [\![c_2]\!]$ iff $(\!|c_1|\!) = (\!|c_2|\!)$.*

Conclusions

Immediate Consequences

- Programming quantum computers can leverage a lot of the infrastructure of classical programming
- Teaching quantum computing should be possible by appealing to just classical notions
- Tantalizing connections to well-established to classical notions
- New CS perspectives
- Quantum advantage ???

nature

Explo

nature

Article

Evi
fau

Young

Rosen

Natur

126k

PRX QUANTUM

a Physical Review journal

Highlights | Recent | Accepted | Authors | Referees | Search | About | Scope | Editorial Team

Featured in Physics **Open Access**

Efficient Tensor Network Simulation of IBM's Eagle Kicked Ising Experiment

Joseph Tindall, Matthew Fishman, E. Miles Stoudenmire, and Dries Sels

PRX Quantum **5**, 010308 – Published 23 January 2024

Physics See Research News: A Moving Target for Quantum Advantage

nature

Explore content About the journal Publish with us

RESEARCH-ARTICLE FREE ACCESS

nat

Arti

Q

su

Fra

Ser

Rob

Gid

Nat

1.0

Closing the "quantum supremacy" gap: achieving real-time simulation of a random quantum circuit using a new Sunway supercomputer

Authors: Yong (Alexander) Liu, Xin (Lucy) Liu, Fang (Nancy) Li, Haohuan Fu, Yuling Yang, Jiawei Song, Pengpeng Zhao, Zhen Wang, Dajia Peng, Huarong Chen, Chu Guo, Heliang Huang, + 2

Authors Info & Claims

SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis • November 2021 • Article No.: 3 • Pages 1–12 • https://doi.org/10.1145/3458817.3487399

Published: 13 November 2021 Publication History

Check for updates

Quantum Advantage?

Still no clue !

But:

- Ability to efficiently switch representation from Z-booleans to X-booleans and back would be sufficient

- Having multiple execution threads going at different speeds is known to provide speedups












(Some of) The Details

The Algebraic Nature of CCX

- CCX operates on collections of booleans.
- What are ‘booleans’ ?
- What do we mean by ‘collections’ ?

Booleans represent Choices

- A boolean represents a choice between two atomic values
- Generalize to zero or more choices among arbitrary values
- 0 represents ‘no choice’ and + introduces a choice between two alternatives
- $\tau ::= 0 \,|\, \tau + \tau$
- Choice is a **commutative monoid**

$$
\begin{aligned}
\tau + 0 &= \tau \\
\tau_1 + \tau_2 &= \tau_2 + \tau_1 \\
\tau_1 + (\tau_2 + \tau_3) &= \tau_1 + (\tau_2 + \tau_3)
\end{aligned}
$$

Collections / Registers / Tuples / Records

- Collections represent one or more ‘thing’ next to each other
- $\tau ::= 0 \mid \tau + \tau \mid 1 \mid \tau * \tau$
- Another **commutative monoid**

```
τ * 1             =  τ
τ₁ * τ₂           =  τ₂ * τ₁
τ₁ * (τ₂ * τ₃)  =  τ₁ * (τ₂ * τ₃)
```

Distributivity !

- cake and (tea or coffee) = (cake and tea) or (cake and coffee)
- cake or (tea and coffee) ≠ (cake or tea) and (cake or coffee)
- We get a **commutative rig** (ring without negatives)

$$
\begin{aligned}
\tau * 0 &= 0 \\
\tau * (\tau_1 + \tau_2) &= (\tau * \tau_1) + (\tau * \tau_2) \\
\\
\tau + 1 &\neq 1 \\
\tau + (\tau_1 * \tau_2) &\neq (\tau + \tau_1) * (\tau + \tau_2)
\end{aligned}
$$

Put it all Together in Category Theory: Symmetric Rig Groupoid

A programming language Π_0 and a logic Π_1 for reasoning about programs

$$
\begin{array}{rcrcllcl}
id & : & b & \leftrightarrow & b & & : & id \\
swap^{+} & : & b_1 + b_2 & \leftrightarrow & b_2 + b_1 & & : & swap^{+} \\
assocr^{+} & : & (b_1 + b_2) + b_3 & \leftrightarrow & b_1 + (b_2 + b_3) & & : & assocl^{+} \\
unite^{+}l & : & 0 + b & \leftrightarrow & b & & : & uniti^{+}l \\
swap^{\times} & : & b_1 \times b_2 & \leftrightarrow & b_2 \times b_1 & & : & swap^{\times} \\
assocr^{\times} & : & (b_1 \times b_2) \times b_3 & \leftrightarrow & b_1 \times (b_2 \times b_3) & & : & assocl^{\times} \\
unite^{\times}l & : & 1 \times b & \leftrightarrow & b & & : & uniti^{\times}l \\
dist & : & (b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) & & : & factor \\
absorbl & : & b \times 0 & \leftrightarrow & 0 & & : & factorzr
\end{array}
$$

$$
\frac{c_1 : b_1 \leftrightarrow b_2 \quad c_2 : b_2 \leftrightarrow b_3}{c_1 \fatsemi c_2 : b_1 \leftrightarrow b_3}
\qquad
\frac{c : b_1 \leftrightarrow b_2}{inv\ c : b_2 \leftrightarrow b_1}
$$

$$
\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 + c_2 : b_1 + b_2 \leftrightarrow b_3 + b_4}
\qquad
\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 \times c_2 : b_1 \times b_2 \leftrightarrow b_3 \times b_4}
$$

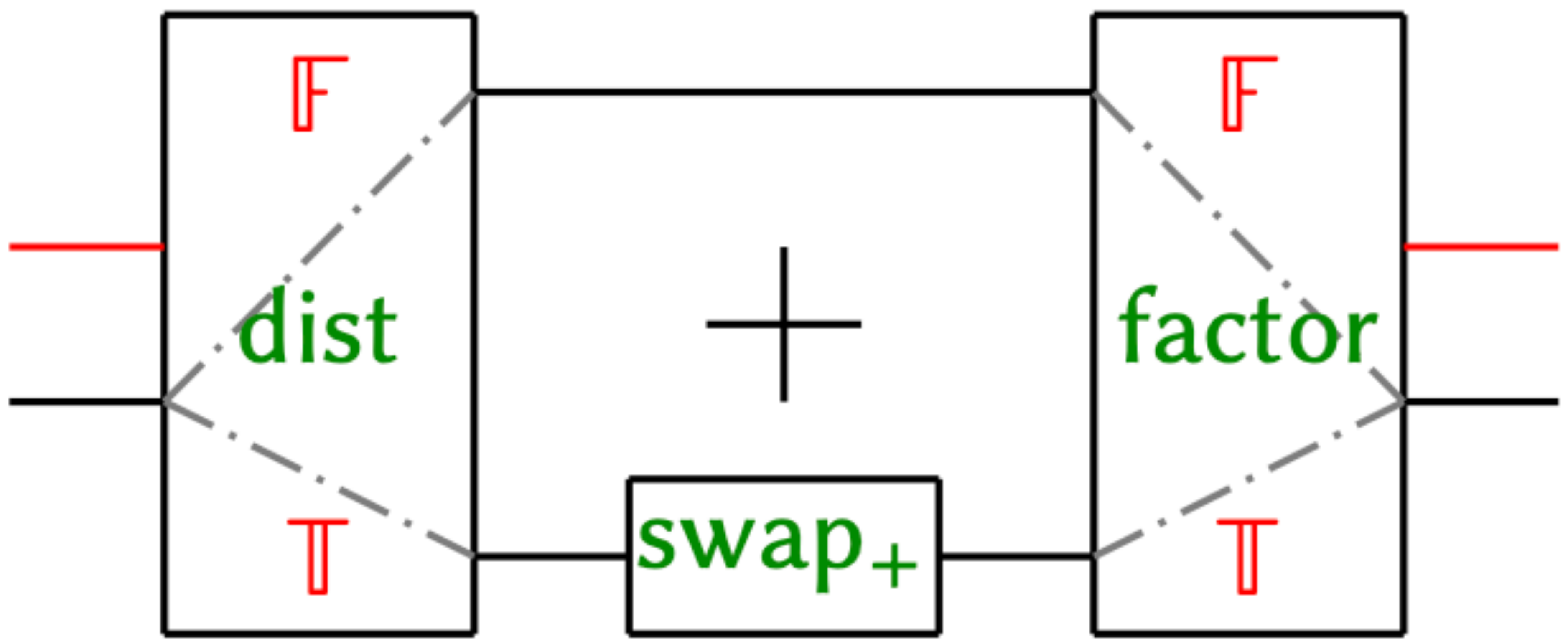
Programming in Π_0

ctrl $c = dist \ ; \ (id + id \times c) \ ; \ factor$

X $= swap^+$

CX = ctrl X

CCX = ctrl CX



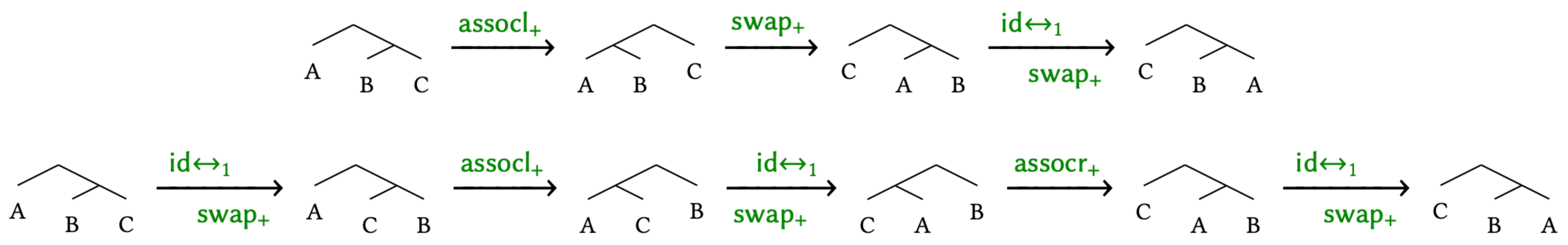
Reasoning in Π_1

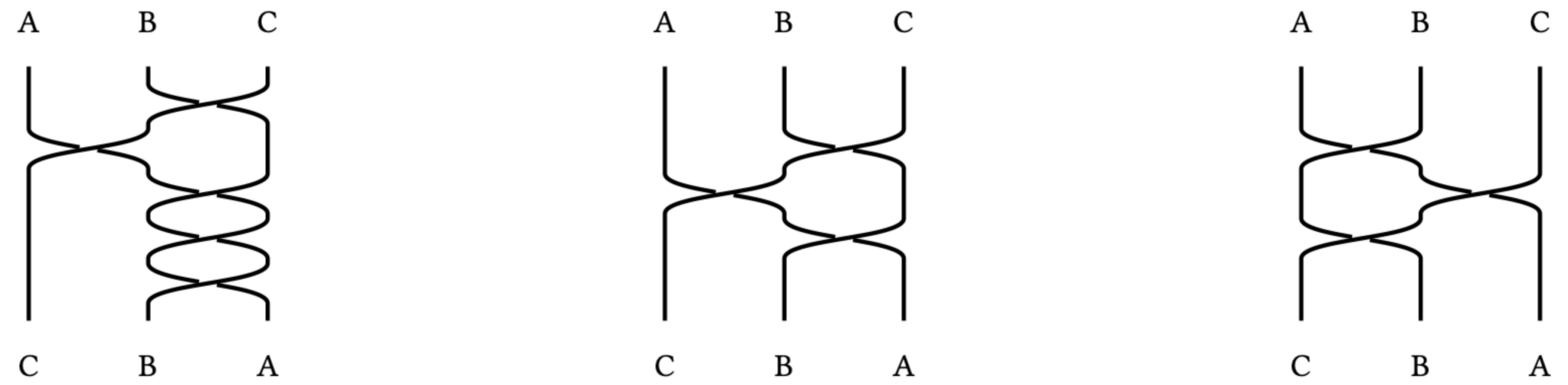
```
neg₁ neg₂ neg₃ neg₄ neg₅ : BOOL ⟷ BOOL
neg₁ = swap₊
neg₂ = id⟷ ◎ swap₊
neg₃ = swap₊ ◎ swap₊ ◎ swap₊
neg₄ = swap₊ ◎ id⟷
neg₅ = uniti⋆l ◎ swap⋆ ◎ (swap₊ ⊗ id⟷) ◎ swap⋆ ◎ unite⋆l
neg₆ = uniti⋆r ◎ (swap₊ {ONE} {ONE} ⊗ id⟷) ◎ unite⋆r

negEx : neg₅ ⇔ neg₁
negEx = (uniti⋆l ◎ (swap⋆ ◎ ((swap₊ ⊗ id⟷) ◎ (swap⋆ ◎ unite⋆l))))
          ⇔⟨ id⇔ ⊡ assoc◎l ⟩
        (uniti⋆l ◎ ((swap⋆ ◎ (swap₊ ⊗ id⟷)) ◎ (swap⋆ ◎ unite⋆l)))
          ⇔⟨ id⇔ ⊡ (swapl⋆⇔ ⊡ id⇔) ⟩
        (uniti⋆l ◎ (((id⟷ ⊗ swap₊) ◎ swap⋆) ◎ (swap⋆ ◎ unite⋆l)))
          ⇔⟨ id⇔ ⊡ assoc◎r ⟩
        (uniti⋆l ◎ ((id⟷ ⊗ swap₊) ◎ (swap⋆ ◎ (swap⋆ ◎ unite⋆l))))
          ⇔⟨ id⇔ ⊡ (id⇔ ⊡ assoc◎l) ⟩
        (uniti⋆l ◎ ((id⟷ ⊗ swap₊) ◎ ((swap⋆ ◎ swap⋆) ◎ unite⋆l)))
          ⇔⟨ id⇔ ⊡ (id⇔ ⊡ (linv◎l ⊡ id⇔)) ⟩
        (uniti⋆l ◎ ((id⟷ ⊗ swap₊) ◎ (id⟷ ◎ unite⋆l)))
          ⇔⟨ id⇔ ⊡ (id⇔ ⊡ idl◎l) ⟩
        (uniti⋆l ◎ ((id⟷ ⊗ swap₊) ◎ unite⋆l))
          ⇔⟨ assoc◎l ⟩
        ((uniti⋆l ◎ (id⟷ ⊗ swap₊)) ◎ unite⋆l)
          ⇔⟨ unitil⋆⇔l ⊡ id⇔ ⟩
        ((swap₊ ◎ uniti⋆l) ◎ unite⋆l)
          ⇔⟨ assoc◎r ⟩
        (swap₊ ◎ (uniti⋆l ◎ unite⋆l))
          ⇔⟨ id⇔ ⊡ linv◎l ⟩
        (swap₊ ◎ id⟷)
          ⇔⟨ idr◎l ⟩
        swap₊ ∎
```

Meta-Theoretical Results

- Thm: Π_0 is universal for classical reversible circuits.
- Thm: Π_1 is sound and complete with respect to permutations on finite sets





Π and $\rotatebox{45}{$\Pi$}$

- For Π we use the symmetric rig groupoid of finite sets and bijections
- For $\rotatebox{45}{$\Pi$}$ we rotate the reference semantics by $\begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$ for some ϕ
- We still just have two individual copies of the classical reversible language Π
- In one copy, the “booleans” are the usual booleans
- In the other copy, the “booleans” have a non-standard representation but this is completely invisible to the outside.

What Happened?

- Each copy of Π internalizes a choice of basis
- Modulo global phase, the required equation forces one copy to use the Z basis and the other copy to use the X basis
- Algebraic presentation of ***complementarity***
- The move from one language to the other is Hadamard
- All of that is hidden
- What is exposed is two classical languages and one equation that governs their interaction

Reasoning

```
minusZ≡plus : (minus >>> Z) ≡ plus
minusZ≡plus = begin
  (minus >>> Z)
    ≡⟨ id≡ ⟩
  ((plus >>> H >>> X >>> H) >>> H >>> X >>> H)
    ≡⟨ ((assoc>>>l ⊙ assoc>>>l) ⟩⨾⟨id ) ⊙ pullʳ assoc>>>l ⟩
  (((plus >>> H) >>> X) >>> (H >>> H) >>> X >>> H)
    ≡⟨ id⟩⨾⟨ ((hadInv ⟩⨾⟨id) ⊙ idl>>>l) ⟩
  (((plus >>> H) >>> X) >>> X >>> H)
    ≡⟨ pullʳ assoc>>>l ⟩
  ((plus >>> H) >>> (X >>> X) >>> H)
    ≡⟨ id⟩⨾⟨ (xInv ⟩⨾⟨id ⊙ idl>>>l) ⟩
  ((plus >>> H) >>> H)
    ≡⟨ cancelʳ hadInv ⟩
  plus ∎
```

Recall: Symmetric Rig Groupoid

A programming language Π_0 and a logic Π_1 for reasoning about programs

$$
\begin{array}{rcrcll}
\mathit{id} & : & b & \leftrightarrow & b & : \mathit{id} \\
\mathit{swap}^{+} & : & b_1 + b_2 & \leftrightarrow & b_2 + b_1 & : \mathit{swap}^{+} \\
\mathit{assocr}^{+} & : & (b_1 + b_2) + b_3 & \leftrightarrow & b_1 + (b_2 + b_3) & : \mathit{assocl}^{+} \\
\mathit{unite}^{+}l & : & 0 + b & \leftrightarrow & b & : \mathit{uniti}^{+}l \\
\mathit{swap}^{\times} & : & b_1 \times b_2 & \leftrightarrow & b_2 \times b_1 & : \mathit{swap}^{\times} \\
\mathit{assocr}^{\times} & : & (b_1 \times b_2) \times b_3 & \leftrightarrow & b_1 \times (b_2 \times b_3) & : \mathit{assocl}^{\times} \\
\mathit{unite}^{\times}l & : & 1 \times b & \leftrightarrow & b & : \mathit{uniti}^{\times}l \\
\mathit{dist} & : & (b_1 + b_2) \times b_3 & \leftrightarrow & (b_1 \times b_3) + (b_2 \times b_3) & : \mathit{factor} \\
\mathit{absorbl} & : & b \times 0 & \leftrightarrow & 0 & : \mathit{factorzr}
\end{array}
$$

$$
\frac{c_1 : b_1 \leftrightarrow b_2 \quad c_2 : b_2 \leftrightarrow b_3}{c_1 \fatsemi c_2 : b_1 \leftrightarrow b_3}
\qquad
\frac{c : b_1 \leftrightarrow b_2}{\mathit{inv}\ c : b_2 \leftrightarrow b_1}
$$

$$
\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 + c_2 : b_1 + b_2 \leftrightarrow b_3 + b_4}
\qquad
\frac{c_1 : b_1 \leftrightarrow b_3 \quad c_2 : b_2 \leftrightarrow b_4}{c_1 \times c_2 : b_1 \times b_2 \leftrightarrow b_3 \times b_4}
$$

Add two terms and three equations

It's Quantum again!

Syntax

$$iso ::= \cdots \mid \mathrm{v} \mid \mathrm{v}\textsc{i} \mid \mathrm{w} \mid \mathrm{w}\textsc{i} \qquad \text{(isomorphisms)}$$

Types

$$\begin{array}{ccccccc} \mathrm{v} & : & \mathbb{2} & \leftrightarrow & \mathbb{2} & : & \mathrm{v}\textsc{i} \\ \mathrm{w} & : & 1 & \leftrightarrow & 1 & : & \mathrm{w}\textsc{i} \end{array}$$

Equations

(E1) $\mathrm{v}^2 \leftrightarrow_2 \mathrm{x}$

(E2) $\mathrm{w}^8 \leftrightarrow_2 1$

(E3) $\mathrm{v} \fatsemi (id + \mathrm{w}^2) \fatsemi \mathrm{v} \leftrightarrow_2 \mathit{uniti}^{\times}l \fatsemi \mathrm{w}^2 \times ((id + \mathrm{w}^2) \fatsemi \mathrm{v} \fatsemi (id + \mathrm{w}^2)) \fatsemi \mathit{unite}^{\times}l$