Microsoft

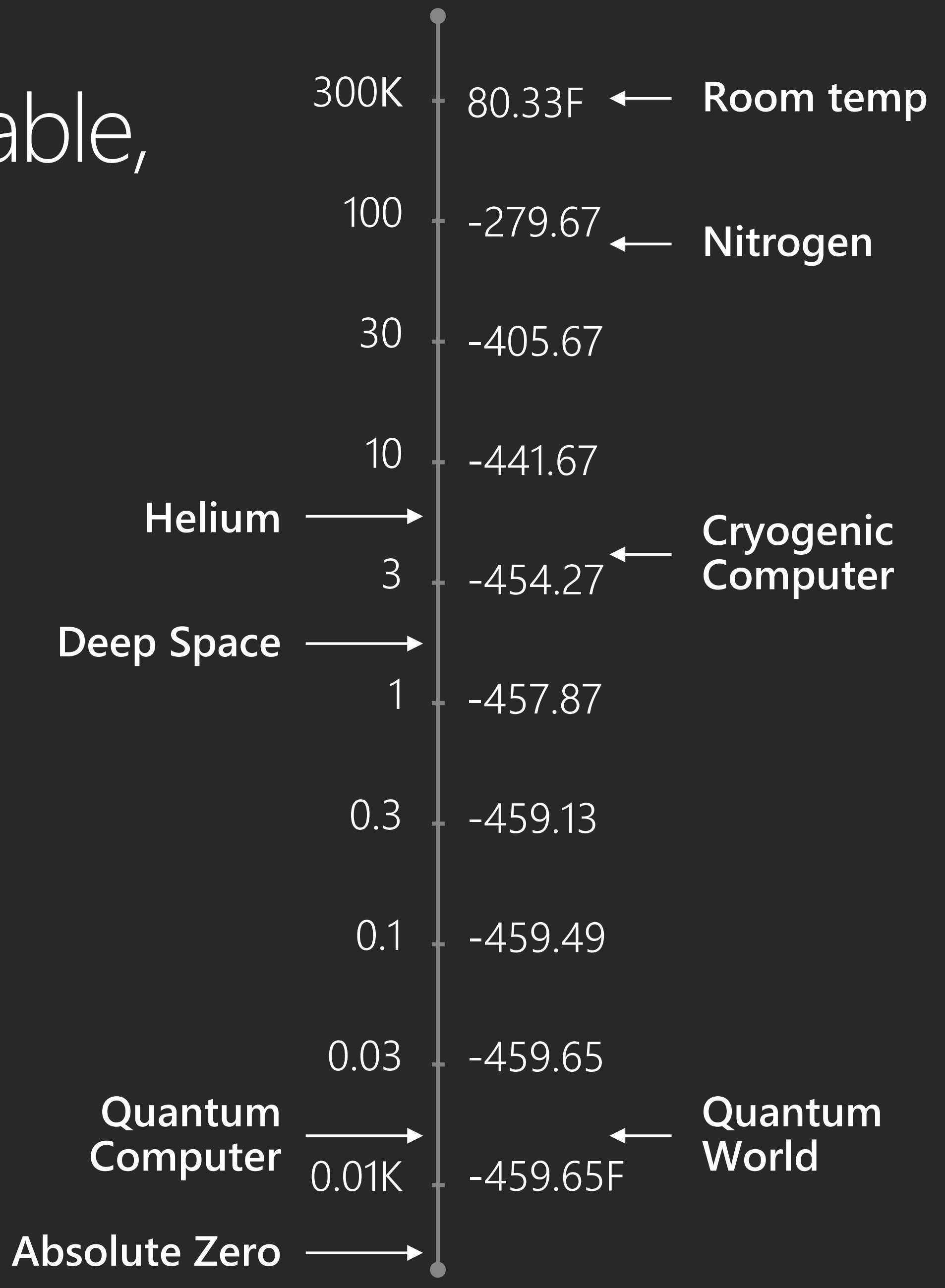
Ancilla management for quantum and reversible computation

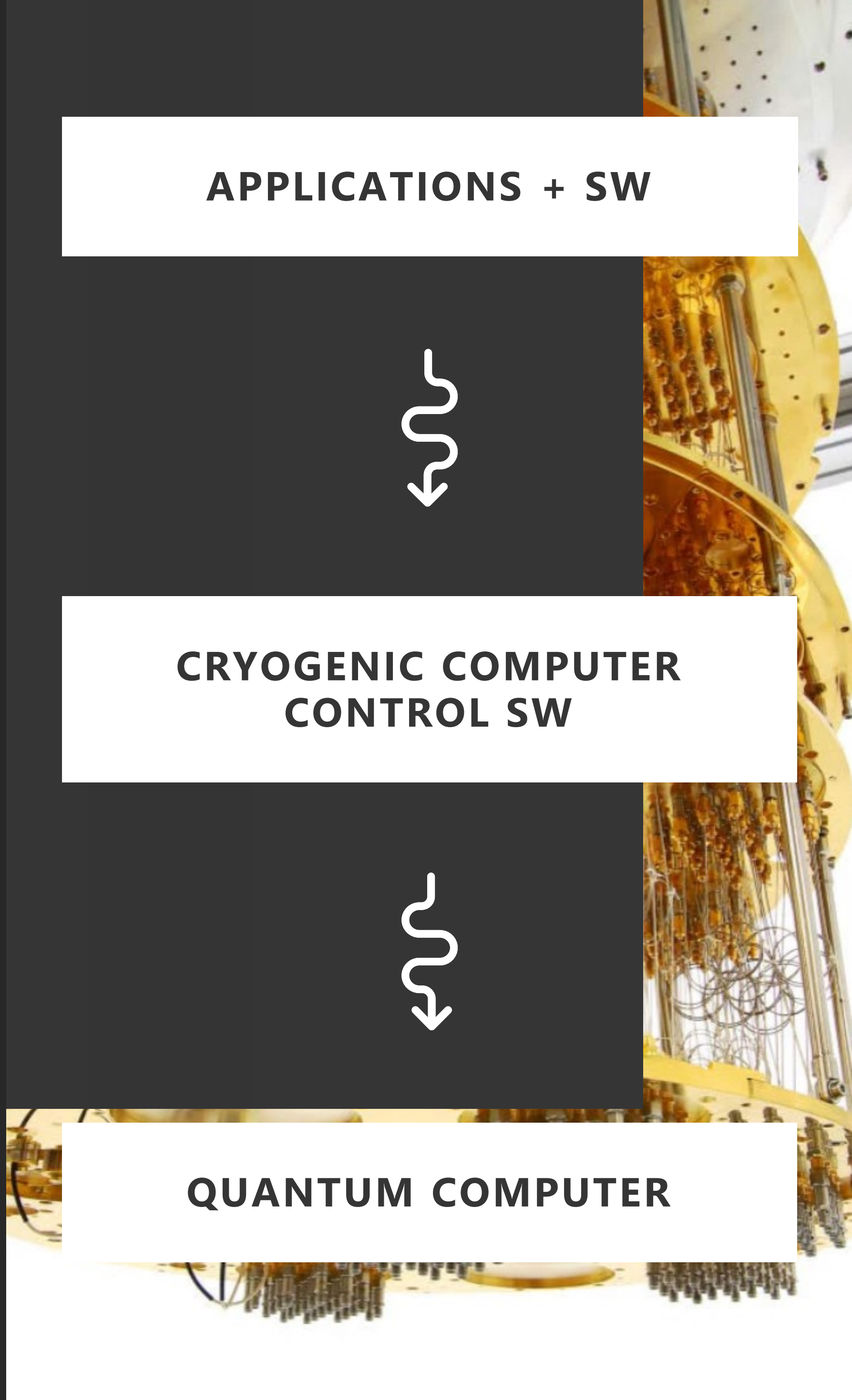
Martin Roetteler
Microsoft Research

EPFL Workshop on Logic Synthesis and Emerging Technologies
Lausanne, Switzerland, September 27, 2017

Igniting a quantum revolution

A complete, scalable, quantum system





Nitrogen fixation

Carbon capture

Materials science

Machine learning

Quantum "Hello, World"

```
operation () EPR (Qubit q1, Qubit q2) {
    Body {
        H (q1)
        CNOT (q1,q2)
    }
}

operation () Teleport (Qubit msg, Qubit here, Qubit there) {
    Body {
        EPR (here, there)
        CNOT (msg, here)
        H (msg)

        let m_here = M (here)
        if (m_here == One) {
            X (there)
        }

        let m_msg = M (msg)
        if (m_msg == One) {
            Z (there)
        }
    }
}

operation (Result) TeleportTest (Result msg) {
    Body {
        mutable res = Zero
        using (qubits = Qubit[3]) {
            let msgQ = qubits[0]

            // Set msgQ to message state
            SetQubit (msg, msgQ)

            Teleport (msgQ, qubits[1], qubits[2])

            set res = M (qubits[2])
        }
        return res
    }
}
```

Overview

Quantum entanglement and interference

Reversible computing

- Avoiding garbage by “un”-computing
- Design space exploration

Quantum memory management

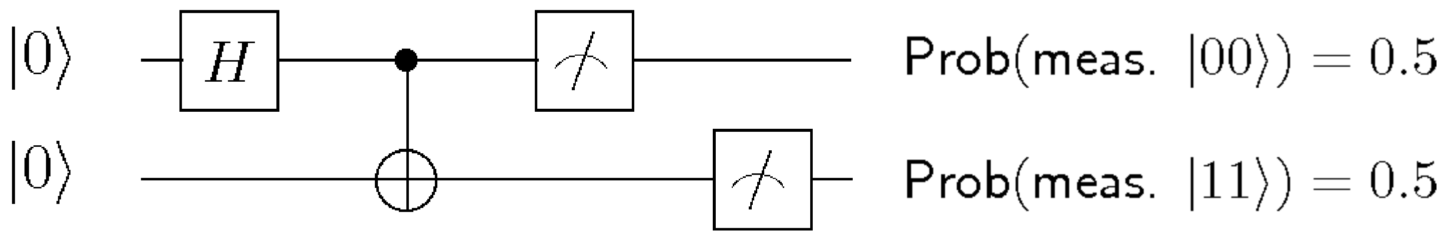
- Working with “dirty” qubits

Cryptanalysis of ECC signatures

- Libraries for modular arithmetic

Quantum circuit example

$$H \otimes \mathbf{1}_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \mathbf{1}_2$$



$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$$

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

$$|00\rangle \to |00\rangle$$
$$|01\rangle \to |01\rangle$$
$$|10\rangle \to |11\rangle$$
$$|11\rangle \to |10\rangle$$

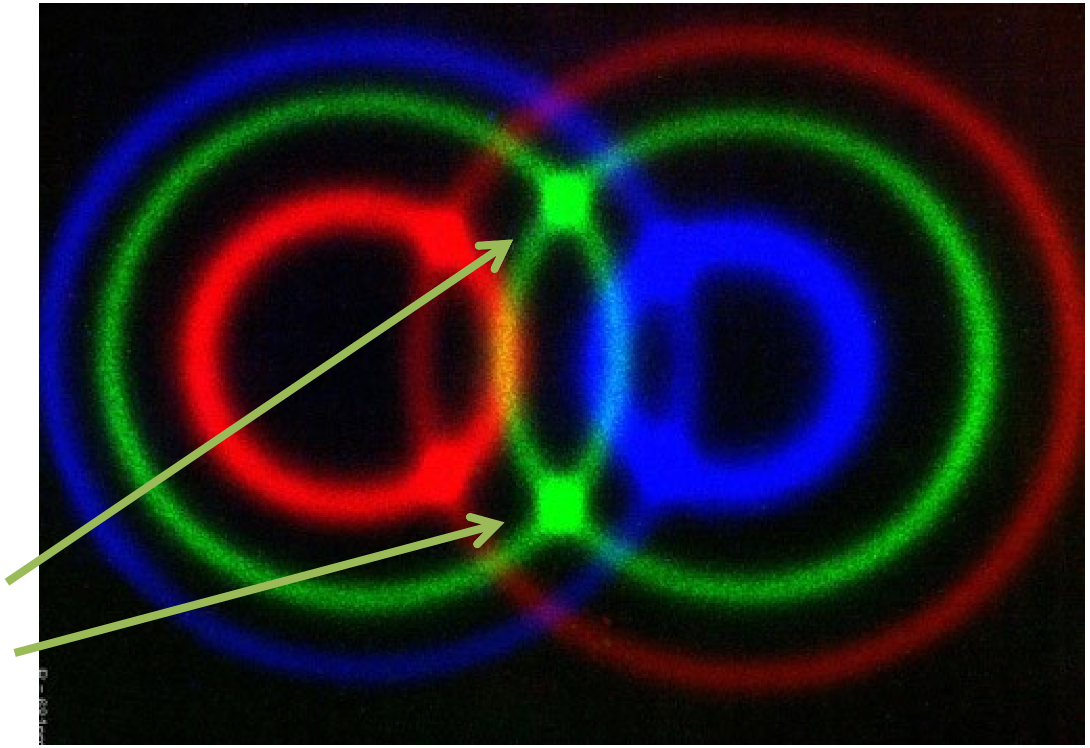
Entangled states

- States that cannot be characterized using only local correlations

- Example: the EPR state $\frac{|01\rangle - |10\rangle}{\sqrt{2}}$ (after Einstein, Podolsky, and Rosen)

  (also called "EPR pair")

- In optics, EPR pairs can be generated e.g. using parametric downconversion



Entangled state between photons emitted with same wavelength (i.e., same color) but with orthogonal polarizations H and V. At the intersection points the polarization is undefined, but different, resulting in a state of the form $1/\sqrt{2}(|H\rangle|V\rangle + e^{i\alpha}|V\rangle|H\rangle)$.

[Picture courtesy of IQOQI (Zeilinger,Weinfurter), Innsbruck]

Interference: the double-slit experiment

Experiment A ("Lee Marvin style"): a gun shoots bullets through 2 holes. Probability for outcome at x is given by summation of two probabilities.

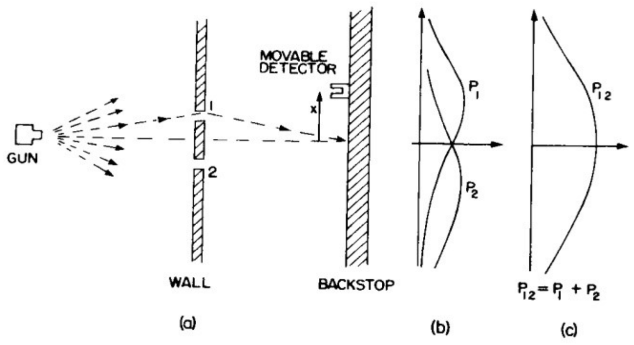


Fig. 1–1. Interference experiment with bullets.

Experiment B ("Water waves"): a source emits water waves. Probability for final outcome at point x also depends on **phases** of incoming waves.

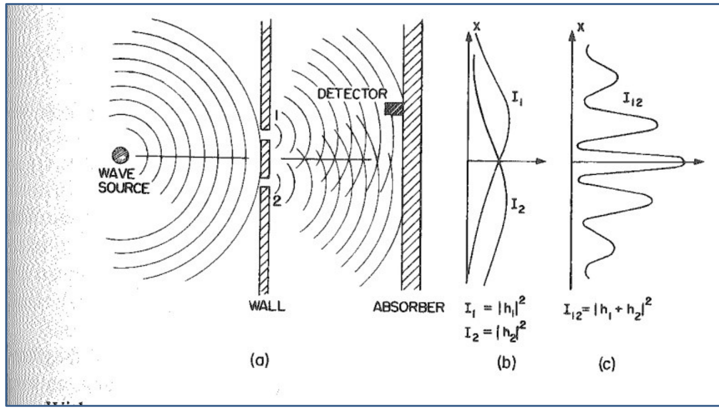
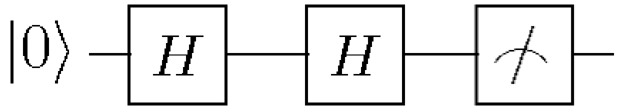


Fig. 1–2. Interference experiment with water waves.

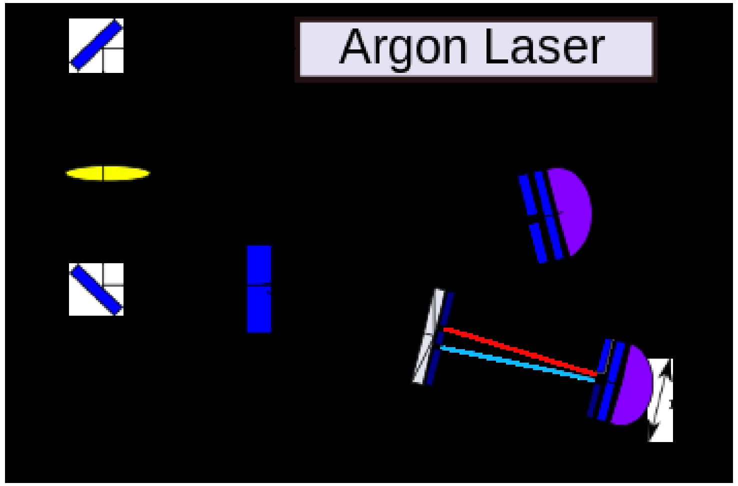
Interference example as circuit:

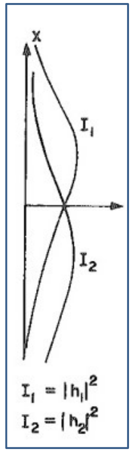


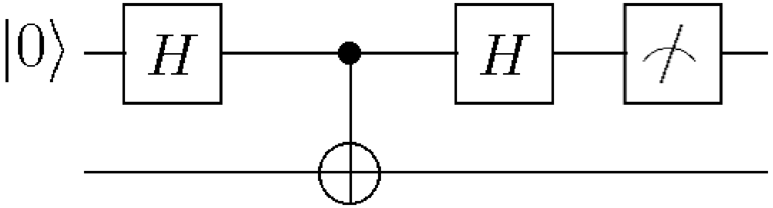
[Pictures credit: R. Feynman, FLP vol I]

Why garbage is fatal for interference

- By inserting polarization filters, the paths can be made distinguishable. The interference pattern disappears.

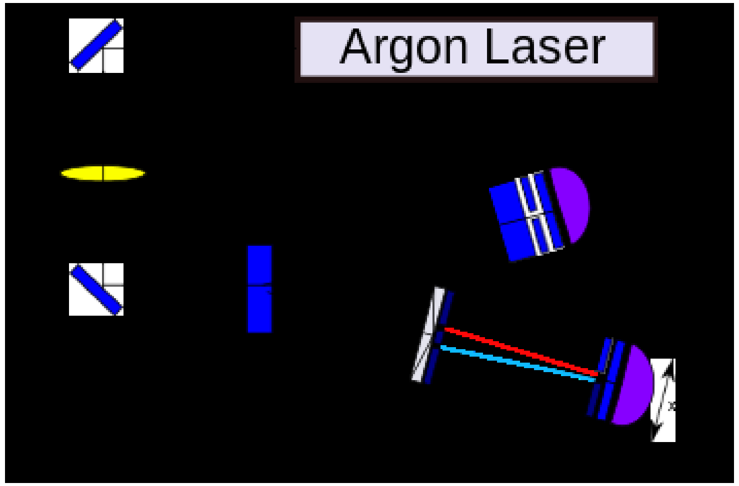


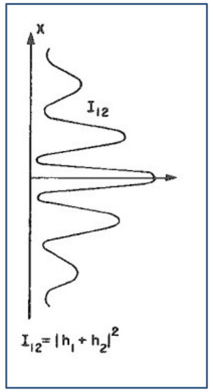


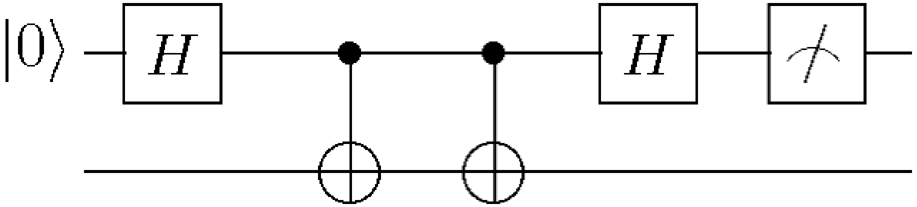


Example using reversible functions:
$|x\rangle|0\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle|g(x)\rangle$

- Quantum eraser experiment: ([Wheeler '78], [Scully et al, '82 and '99]): "Erase" polarization information **after** the photon passed the slits. The interference pattern re-appears!



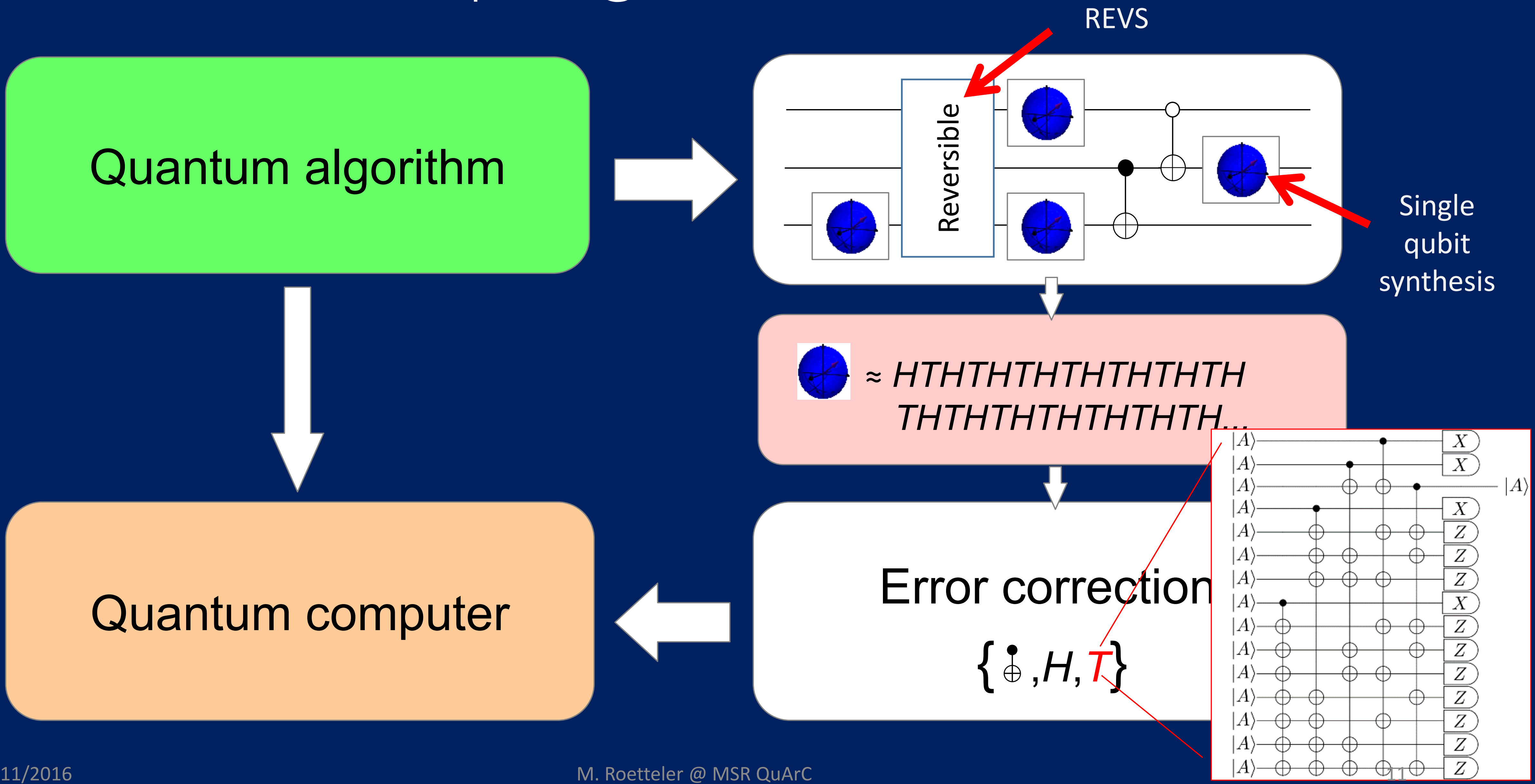




Example using reversible functions:
$|x\rangle|0\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle|0\rangle$

[Pictures credit: Wikipedia]

Quantum compiling



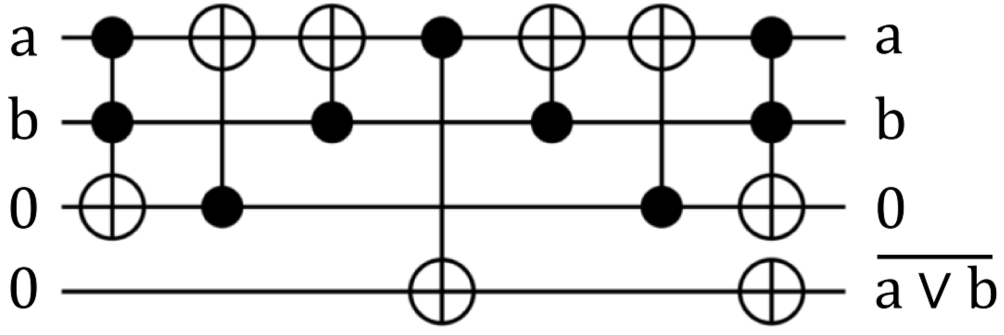
Reversible computing: why do we care?

- Arithmetic:
  - Factoring: just needs “constant” modular arithmetic
  - ECC dlogs: need generic modular arithmetic
  - HHL: need integer inverses; Newton type methods
- Amplitude amplification:
  - Implementation of the “oracles”, e.g., for search, collision etc.
  - Implementation of walk operators on data structures
- Quantum simulation:
  - Addressing/indexing functions for sparse matrices
  - Computing Hamiltonian terms on the fly

Universal reversible gate set: Toffoli gates

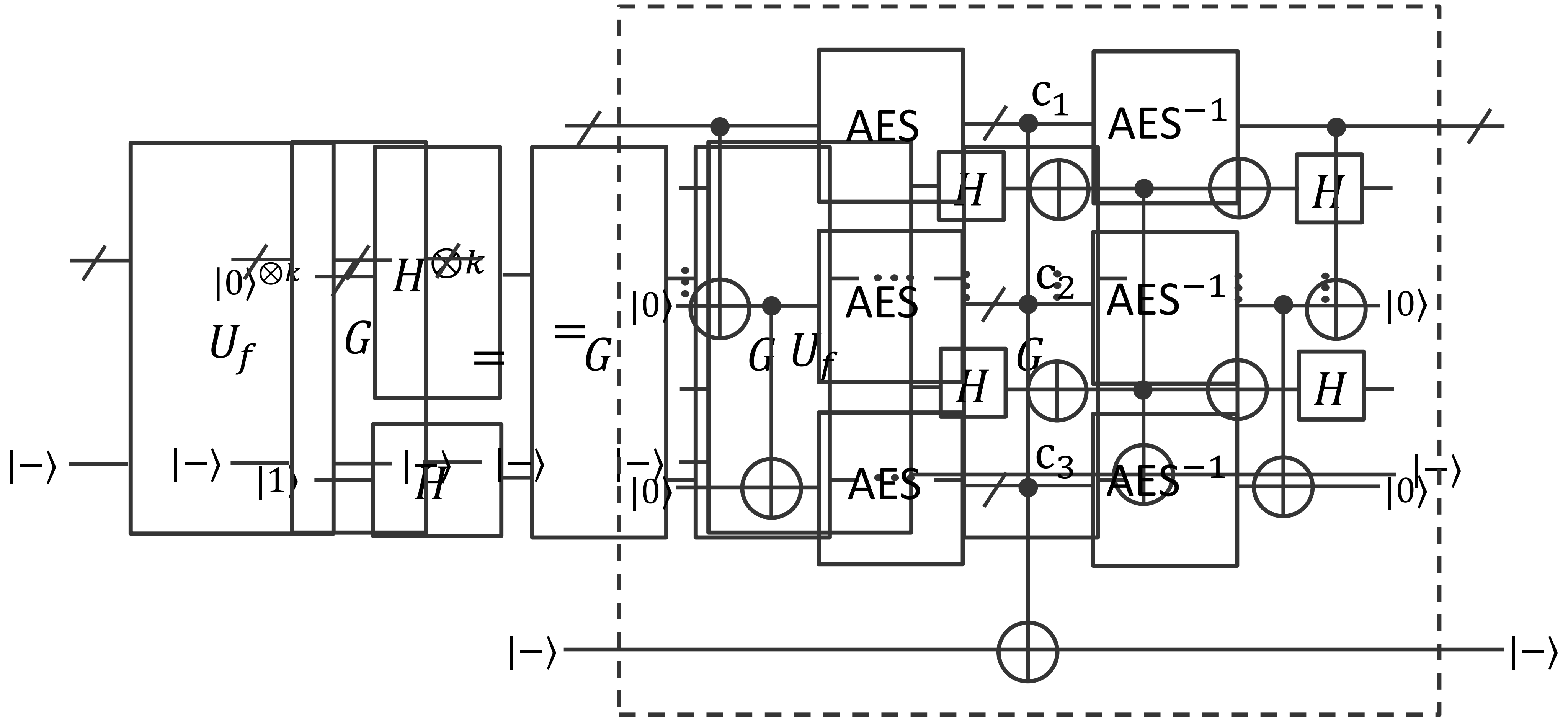
Fact: The set {Toffoli, CNOT, NOT} is universal for reversible computing: any *even* permutation on n qubits can be written as a sequence of Toffoli, CNOT, and NOT gates. **[Toffoli'80], [Fredkin/Toffoli'82]**

Example:



Main motivation: How can we find efficient implementations of reversible circuits in terms of efficient Toffoli networks?
How can we do this starting from irreversible descriptions in a programming language like Python or Haskell or F# or C?
Can we trade time (circuit depth) for space (#qubits) in a meaningful way?

Zooming into a quantum algorithm: Grover



$$f(K) := (\mathrm{AES}_K(m_1) = c_1) \wedge \ldots \wedge (\mathrm{AES}_K(m_r) = c_r)$$

[Grassl, Langenberg, R., Steinwandt, PQCrypto'16]

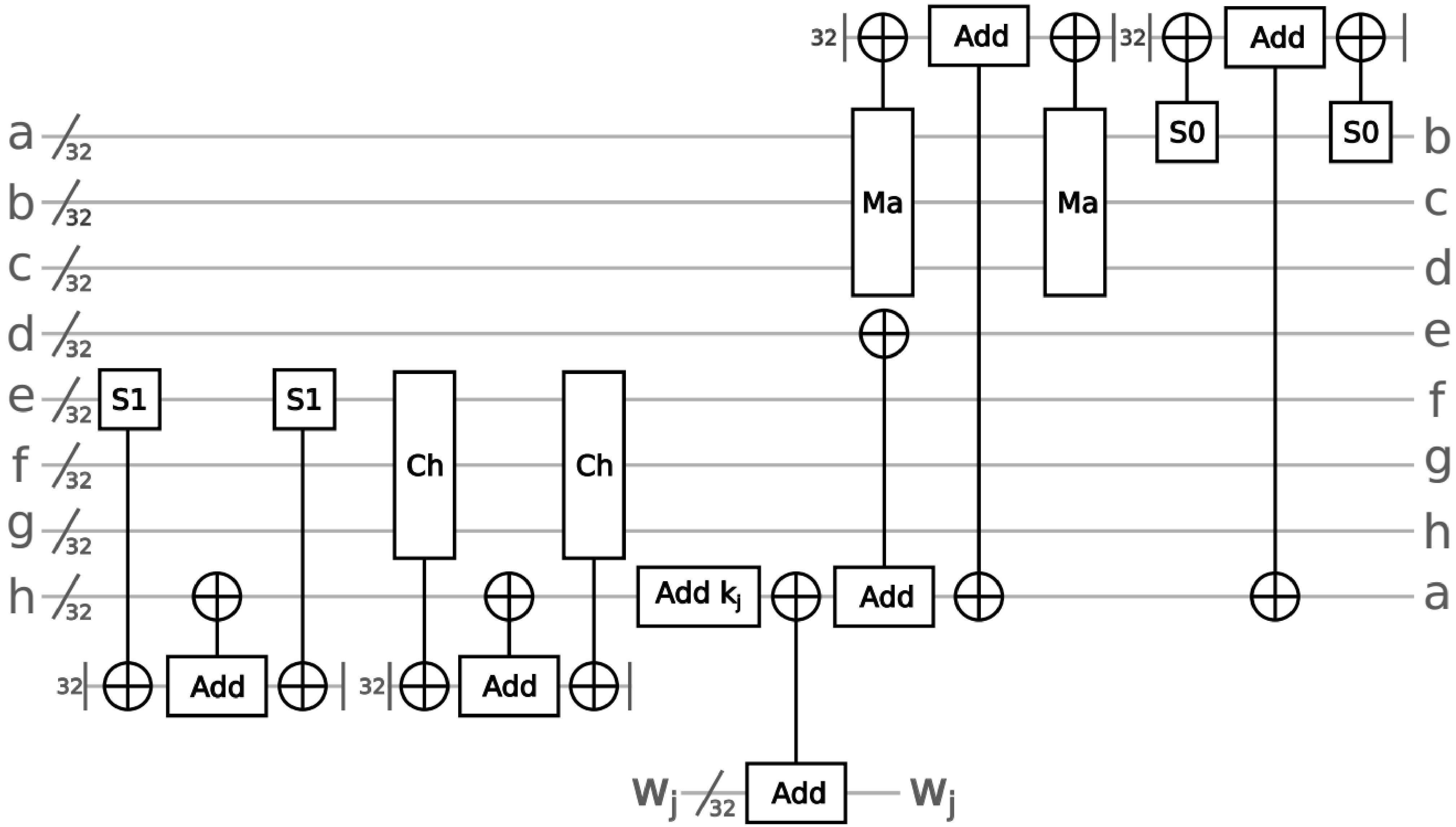
An example at scale: SHA-2

Hash function:

```
Initialize hash values
h0 := 0x6a09e667
h1 := 0xbb67ae85
…
h7 := 0x5be0cd19
Initialize constants
k[0..63] := 0x428a2f98, 0x71374491, 0xb5c0fbcf, …
Do preprocessing
break message into 512-bit chunks (16 32bit ints)
Expand to 64 32 bit ints as follows:
Create W: a 64 entry array of 32 bit ints
Copy the massage into w[0..15] and do:
for each chunk
          for i from 16 to 63
                    s0 := (w[i-15] ⋙ 7) ⊕ (w[i-15] ⋙ 18) ⊕ (w[i-15] ≫ 3)
                    s1 := (w[i-2] ⋙17) ⊕ (w[i-2] ⋙19) ⊕ (w[i-2] rshift 10)
                    w[i] := w[i-16] + s0 + w[i-7] + s1
          Initialize working variables to current hash value:
          a := h0
          …
          h := h7 Compression function main loop:
          Do compression rounds
          Add the compressed chunk to the current hash value:
          h0 := h0 + a
          …
          h7 := h7 + h
digest := hash := h0 :: h1 :: h2 :: h3 :: h4 :: h5 :: h6 :: h7
```

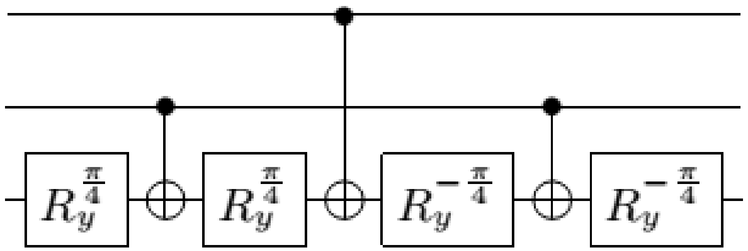
[Source: Wikipedia]

SHA-2: hand-optimized reversible circuit



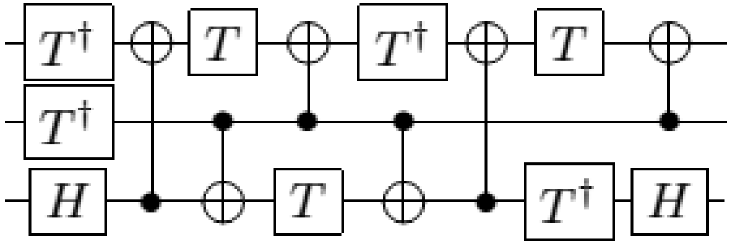
Benchmark “algorithms”

a



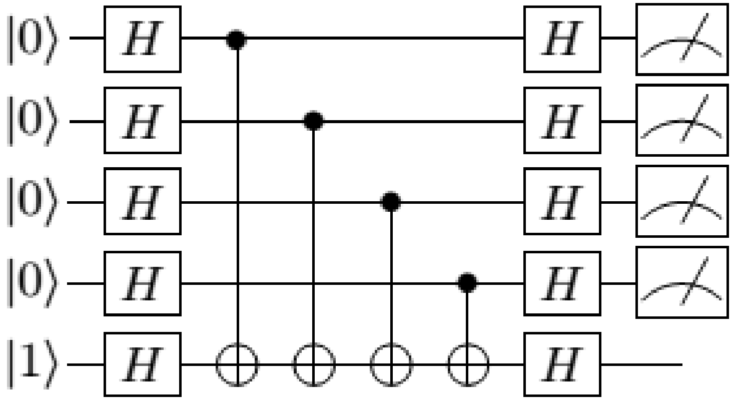
Margolus gate

b



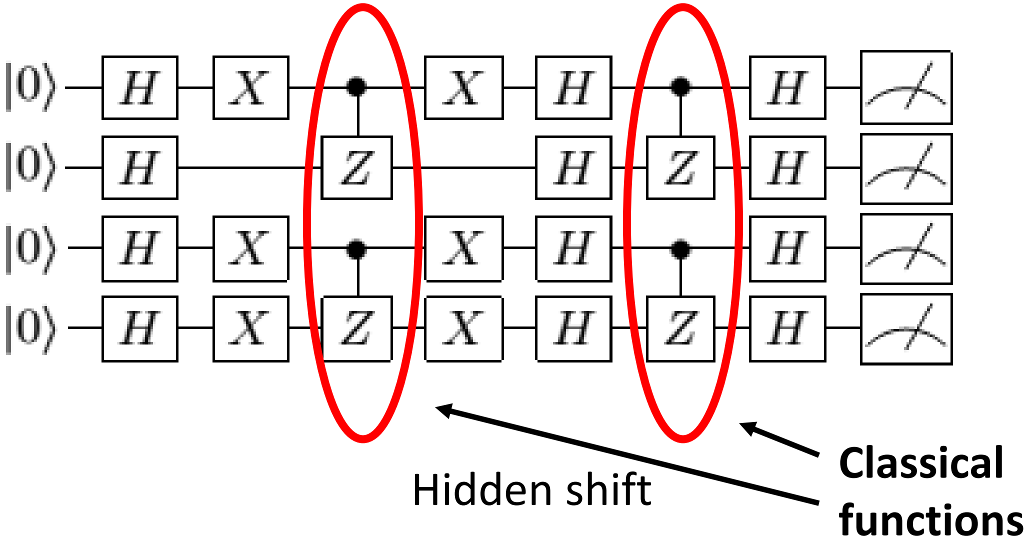
Toffoli gate

c



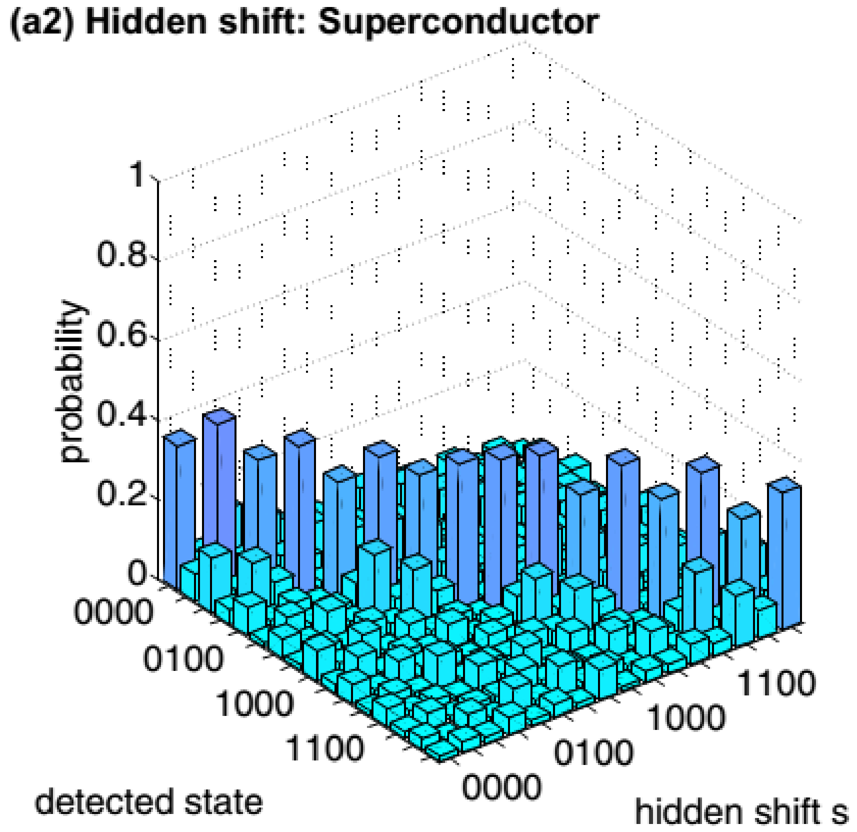
Bernstein-Vazirani

d

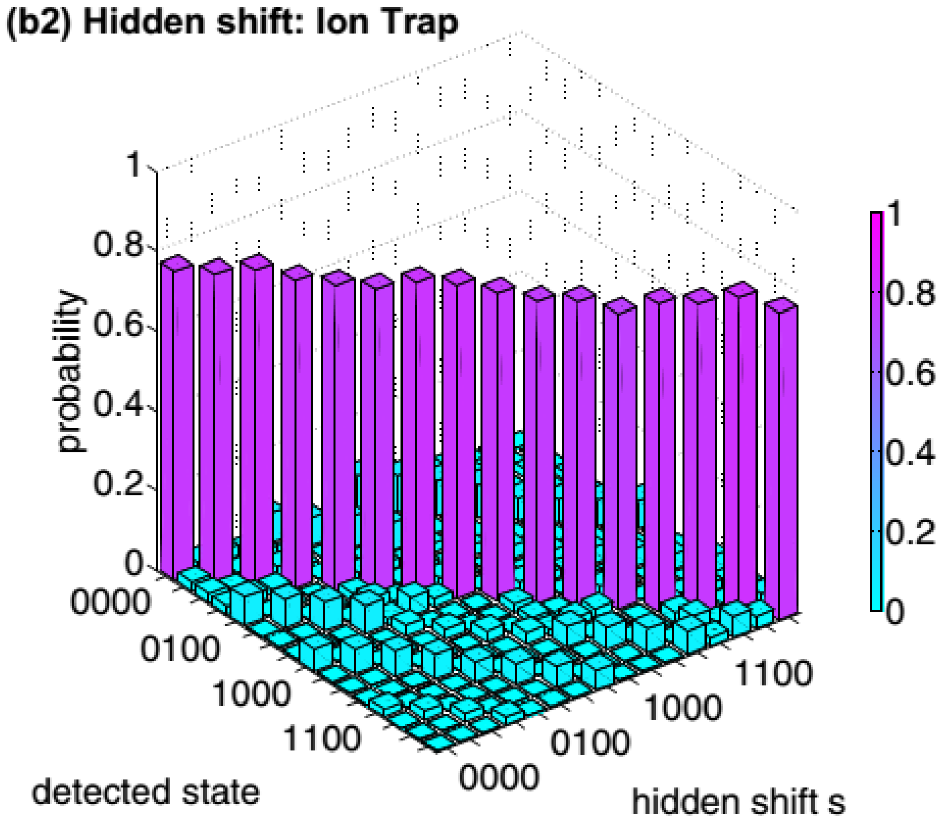


Hidden shift

Experiments on quantum HW: hidden shifts



$\overline{P}_{success} = 0.35$

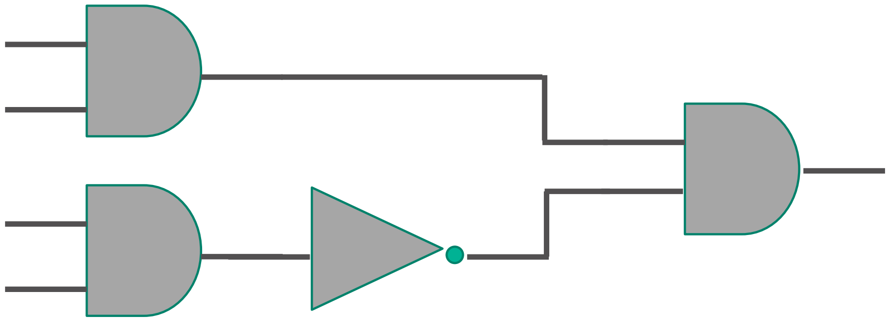


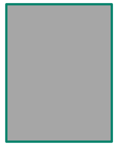
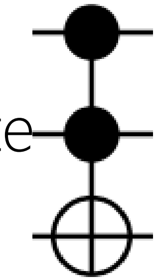
$\overline{P}_{success} = 0.77$

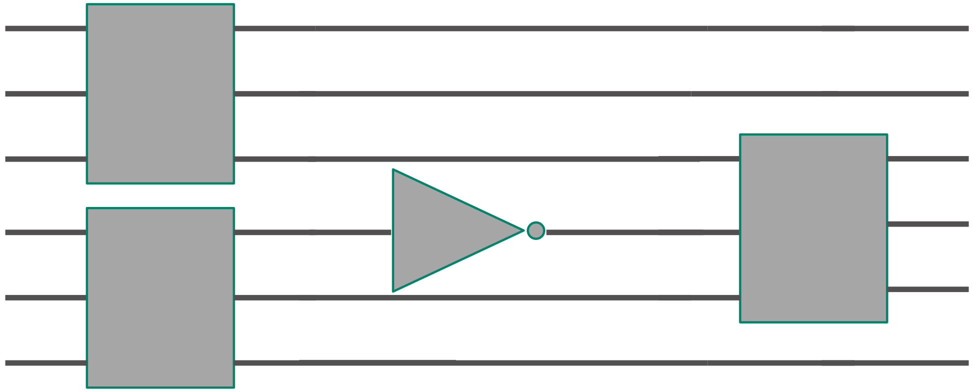
[Linke et al, Proc. Nat. Acad. Science, 2017]

Reversible embeddings

Example:



Replace each gate with a reversible one: (e.g.,  = Toffoli gate )



How to avoid garbage?

- Replacing each gate with a reversible one works fine, however, it produces "garbage", i.e., help registers will be in a state different from 0 at the end.

- There is a way out of this dilemma: the Bennett trick

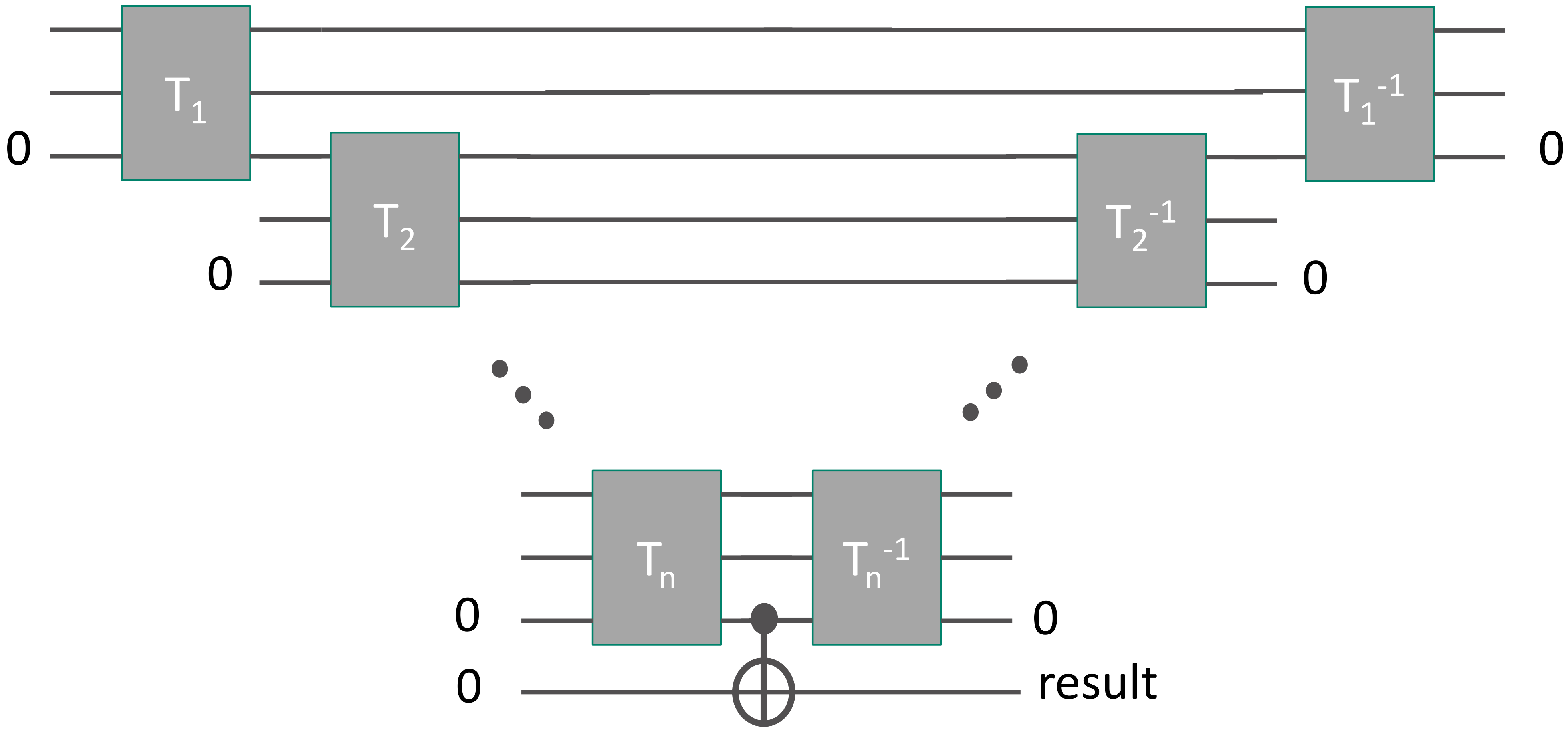
$$\begin{aligned} |x\rangle\,|0\rangle\,|0\rangle\,|0\rangle \;&\mapsto\; |x\rangle\,|f(x)\rangle\,|garbage(x)\rangle\,|0\rangle \\ &\mapsto\; |x\rangle\,|f(x)\rangle\,|garbage(x)\rangle\,|f(x)\rangle \\ &\mapsto\; |x\rangle\,|0\rangle\,|0\rangle\,|f(x)\rangle \end{aligned}$$

Idea: compute forward, copy the result, "uncompute" the garbage by running the computation backwards.

Problem: this leads to a marge quantum memory footprint.

Cleaning up the ancilla (scratch) qubits

- Replace each gate with a reversible one [Bennett, IBM JRD'73]:

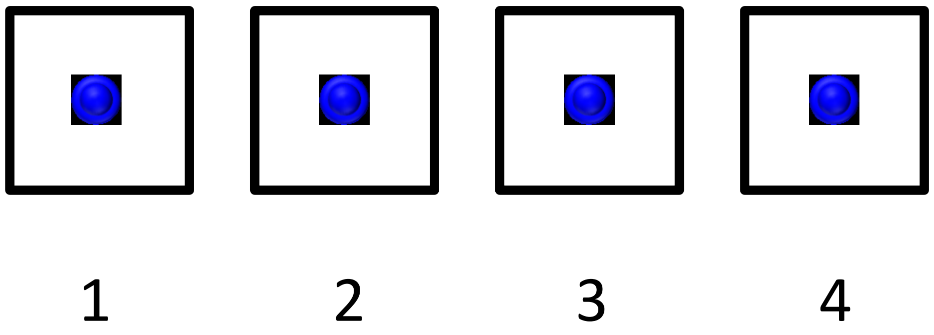


Pebble game: case of 1D chain

Rules of the game: [Bennett, SIAM J. Comp., 1989]

- n boxes, labeled i = 1, …, n
- in each move, either add or remove a pebble
- a pebble can be added or removed in i=1 at any time
- a pebble can be added of removed in i>1 if and only if there is a pebble in i-1
- 1D nature arises from decomposing a computation into “stages”

Example:



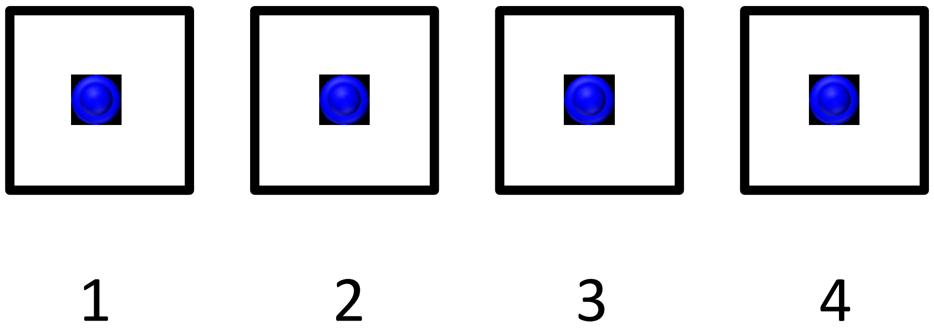
#	i
1	1
2	2
3	3
4	4
5	3
6	2
7	1

Pebble game: 1D chain w/space constraints

Imposing resource constraints:

- only a total of S pebbles are allowed
- corresponds to reversible algorithm with at most S ancilla qubits

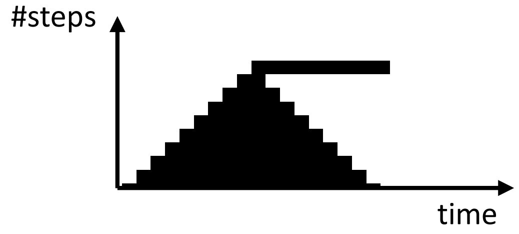
Example: (n=3, S=3)



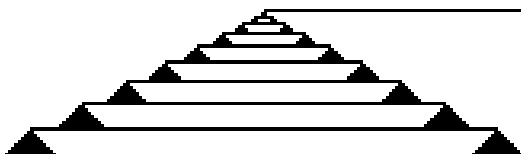
#	i
1	1
2	2
3	3
4	1
5	4
6	3
7	1
8	2
9	1

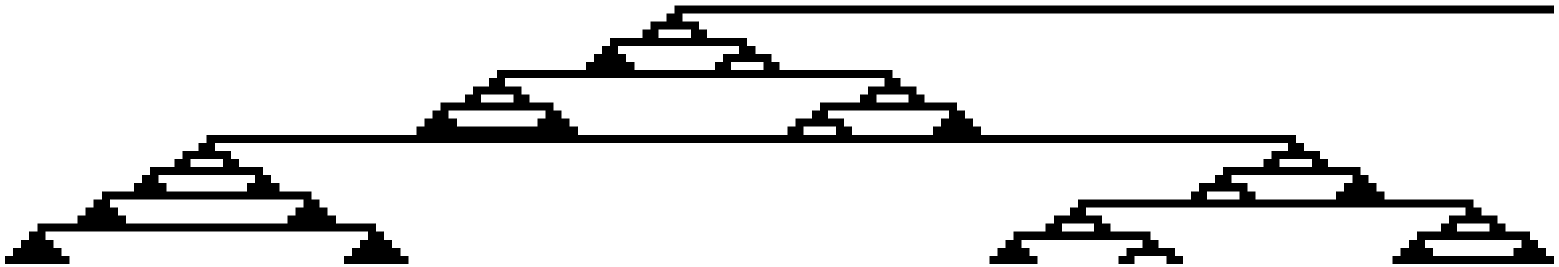
Optimal pebbling strategies: 1D chains

Dynamic programming: Allows to find best strategy for given number of steps n to be performed and given space resource constraint S which is the number of available pebbles.

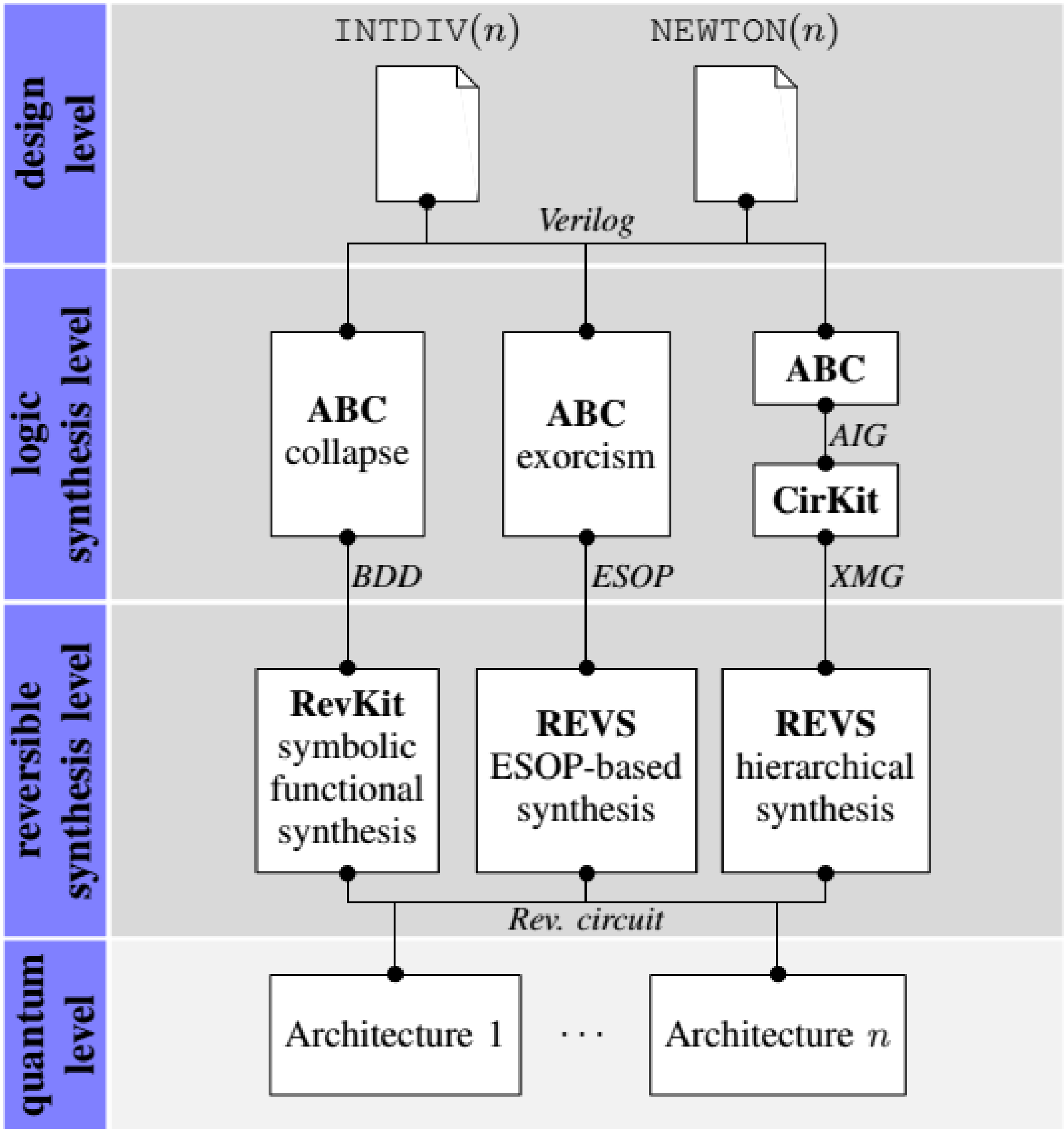


This works ok for 1D chains. For general graphs the problem of finding the optimal strategy is difficult (PSPACE complete problem) -> need heuristics





Circuit synthesis for classical subroutines



Example: compute integer division $x \mapsto 2^n/x$, Where x is an n-bit (unsigned) integer and the result is rounded to the closest integer.

At design level: start from high-level implementations of division function in Verilog. We considered:

- Integer long division (divide $2^n = qx + r$)
- Newton-Raphson

At logic synthesis level:

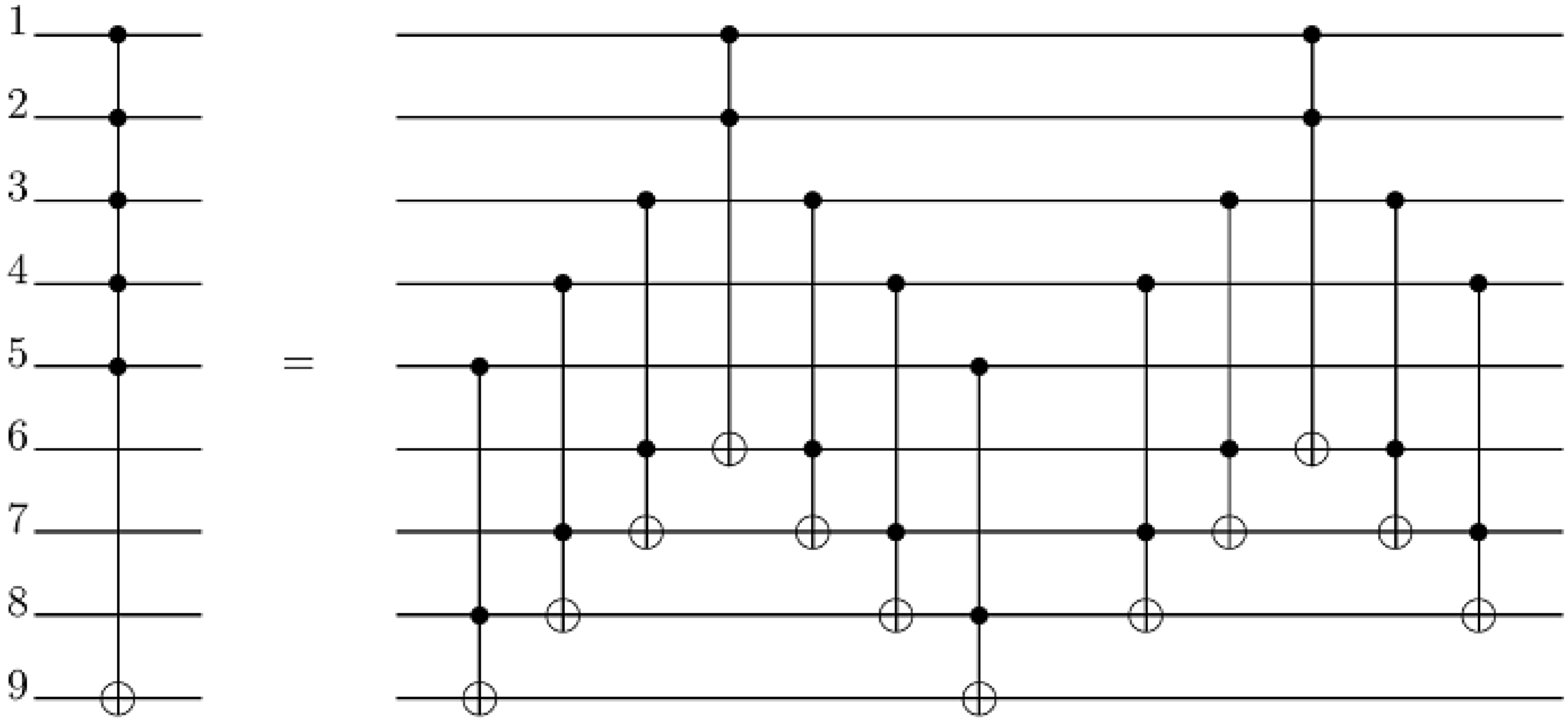
- Convert Verilog to logical netlist in AIG format (And-Inverter Graphs) using tool ABC
- Convert AIG to ESOP format (Exclusive Sums of Products) using tool XOR-cism
- Convert ESOP to Toffoli networks using different tools (REVS, RevKit)
- Also, we considered LUT based synthesis

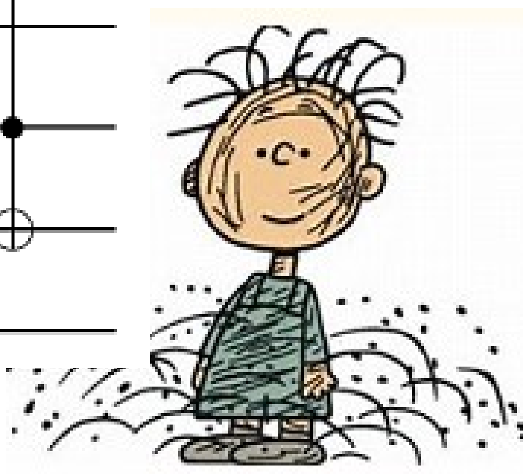
Several passes through the above for various parameter settings that allow T-count/space/compile time tradeoffs.

Design space: **[Soeken, R, Wiebe, De Micheli, DATE'17]**, LUTs: **[DAC'17]**

Dirty qubits

Implementation of Shor's algorithm on 2n+2 qubits

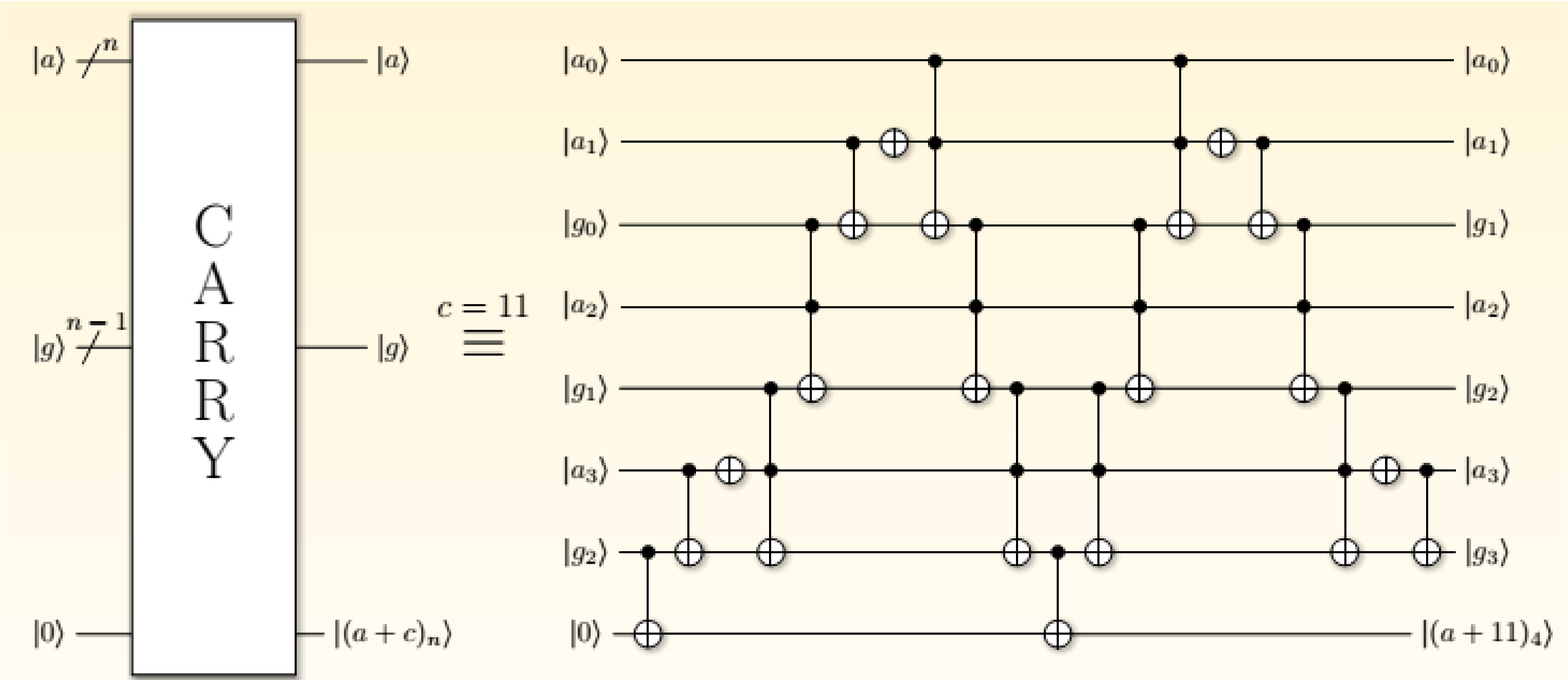




Basic inspiration: apply tricks similar to the above to use "dirty" ancillas for optimization (Barenco et al, PRA'95)

arXiv:1611.07995

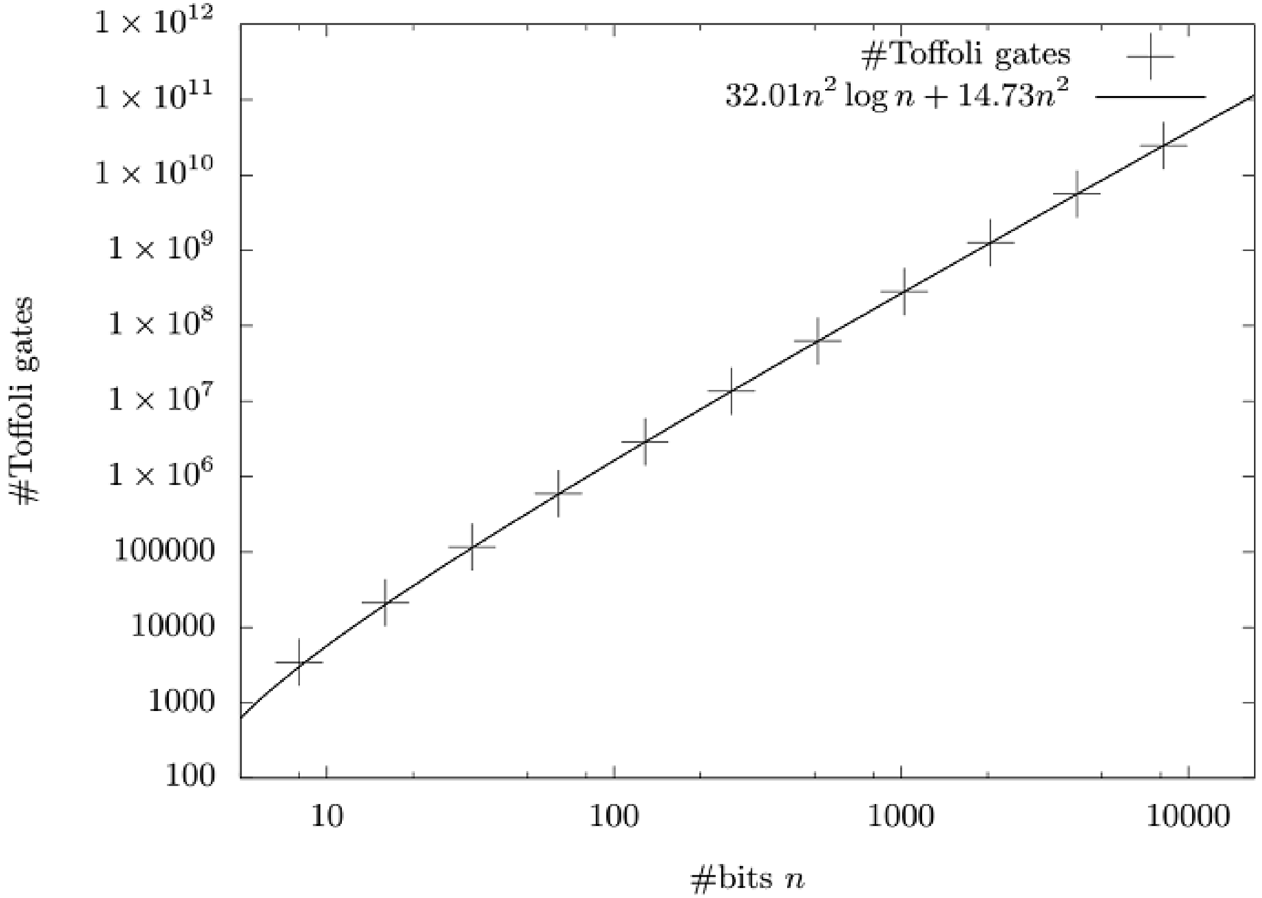
Carry prediction with dirty ancillas

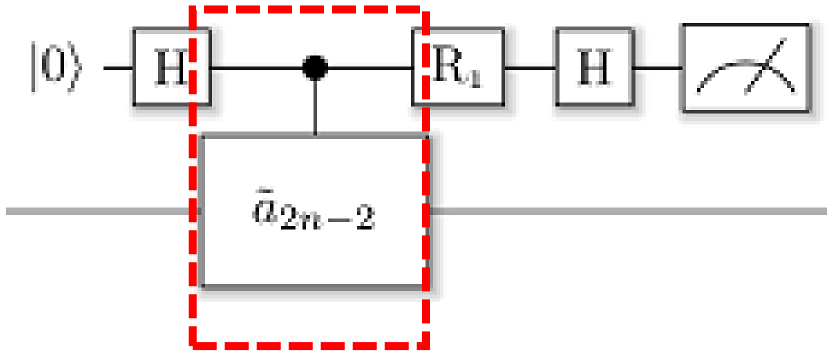


[Haener, R., Svore, QIC 2017]
[Gidney, arXiv:1706.07884]

Based on this, on can build constant folded modular arithmetic (+,*,exp)

Simulating the entire modular multiplication





controlled modular multiplication

[Haener, R., Svore, QIC 2017]

- Built Toffoli network for modular-multiplication for bit sizes relevant for RSA (1024-8192)
- Simulated networks in LIQUi|> using Toffoli simulator
- Metrics for entire Shor algorithm: #qubits = $2n + 2$, #Toffoli-gates = $64n^3 \log(n) + 29.45n^3$

Breaking ECC crypto

Or: why we need powerful quantum software libraries

Breaking RSA and elliptic curve signatures

Integer factorization

27997833911221327870829467638722601621
07044678695542853756000992932612840010
76093456710529553608560618223519109513
65788637105954482006576775098580557613
57909873495014417886317894629518723786
9221823983

= 35324619344027701212726049781984643686
71197400197625[illegible]76121253679
4232000585479[illegible]349
???

× 79258699544783330333470858414800596877
37975857364[illegible]145576787281
8152135381409304[illegible]467
???

Best known methods to factor n-bit numbers*:

Classical: $O\left(\exp\left(c\, n^{1/3} (\log n)^{2/3}\right)\right)$

Quantum: $O\left(n^2 \log n\, 2^{\log^* n}\right)$

*: In practice, Shor's quantum algorithm scales as $O(n^3 \log n)$.

ECC: Discrete logarithms

Basis for RSA/SSL encryption

E

eals

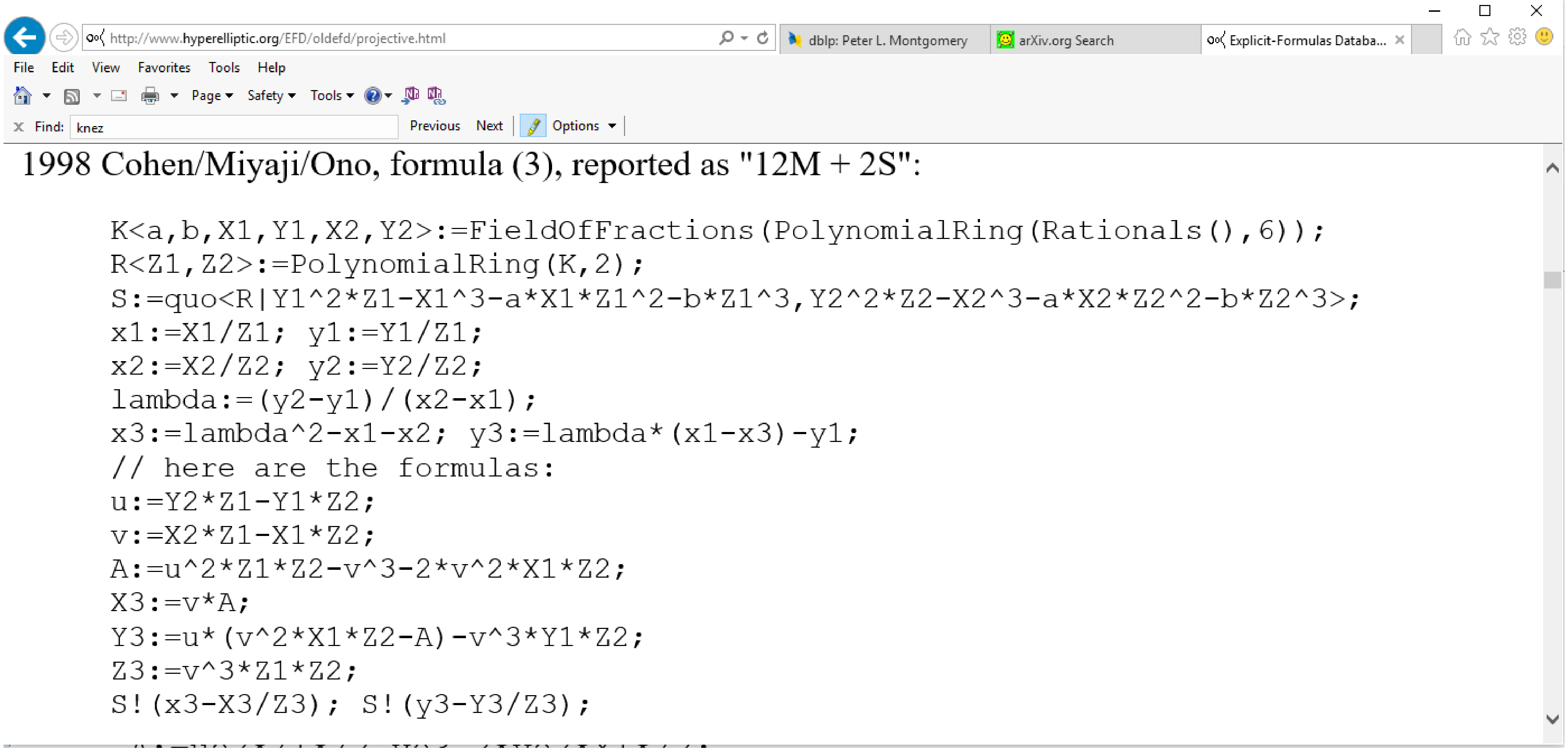
ue).

Recent result: circuit for Shor's

gs:

[Roetteler, Naehrig, Svore, Lauter, arxiv: 1706.06752]

Example: ECC point addition



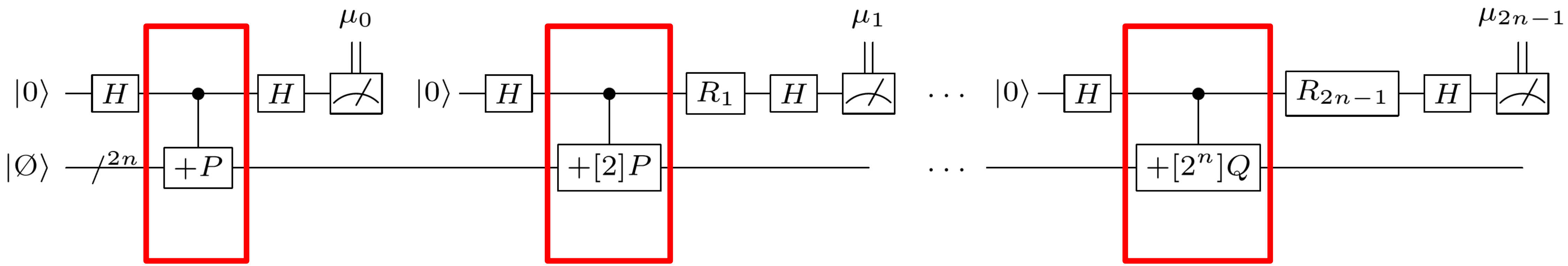
1998 Cohen/Miyaji/Ono, formula (3), reported as "12M + 2S":

```
K<a,b,X1,Y1,X2,Y2>:=FieldOfFractions(PolynomialRing(Rationals(),6));
R<Z1,Z2>:=PolynomialRing(K,2);
S:=quo<R|Y1^2*Z1-X1^3-a*X1*Z1^2-b*Z1^3,Y2^2*Z2-X2^3-a*X2*Z2^2-b*Z2^3>;
x1:=X1/Z1; y1:=Y1/Z1;
x2:=X2/Z2; y2:=Y2/Z2;
lambda:=(y2-y1)/(x2-x1);
x3:=lambda^2-x1-x2; y3:=lambda*(x1-x3)-y1;
// here are the formulas:
u:=Y2*Z1-Y1*Z2;
v:=X2*Z1-X1*Z2;
A:=u^2*Z1*Z2-v^3-2*v^2*X1*Z2;
X3:=v*A;
Y3:=u*(v^2*X1*Z2-A)-v^3*Y1*Z2;
Z3:=v^3*Z1*Z2;
S!(x3-X3/Z3); S!(y3-Y3/Z3);
```

[Bernstein, Lange: Database of explicit ECC formulas: http://www.hyperelliptic.org/EFD/]

High-level structure of the quantum algorithm

Phase estimation framework:



- Except for Hadamard H, rotations R_i and measurements, all gates in these circuits can be implemented over the Toffoli gate set.
- This allows to construct the circuit and simulate it on a classical machine.
- Precise resource estimates can be obtained from reference implementation.

Montgomery inversion

Algorithm MONTINVERSE
Inputs: a, b, n, where a is odd, $a > b > 0$, and n is the number of bits in a
Output: "Not relatively prime," or $b^{-1}2^n \bmod a$

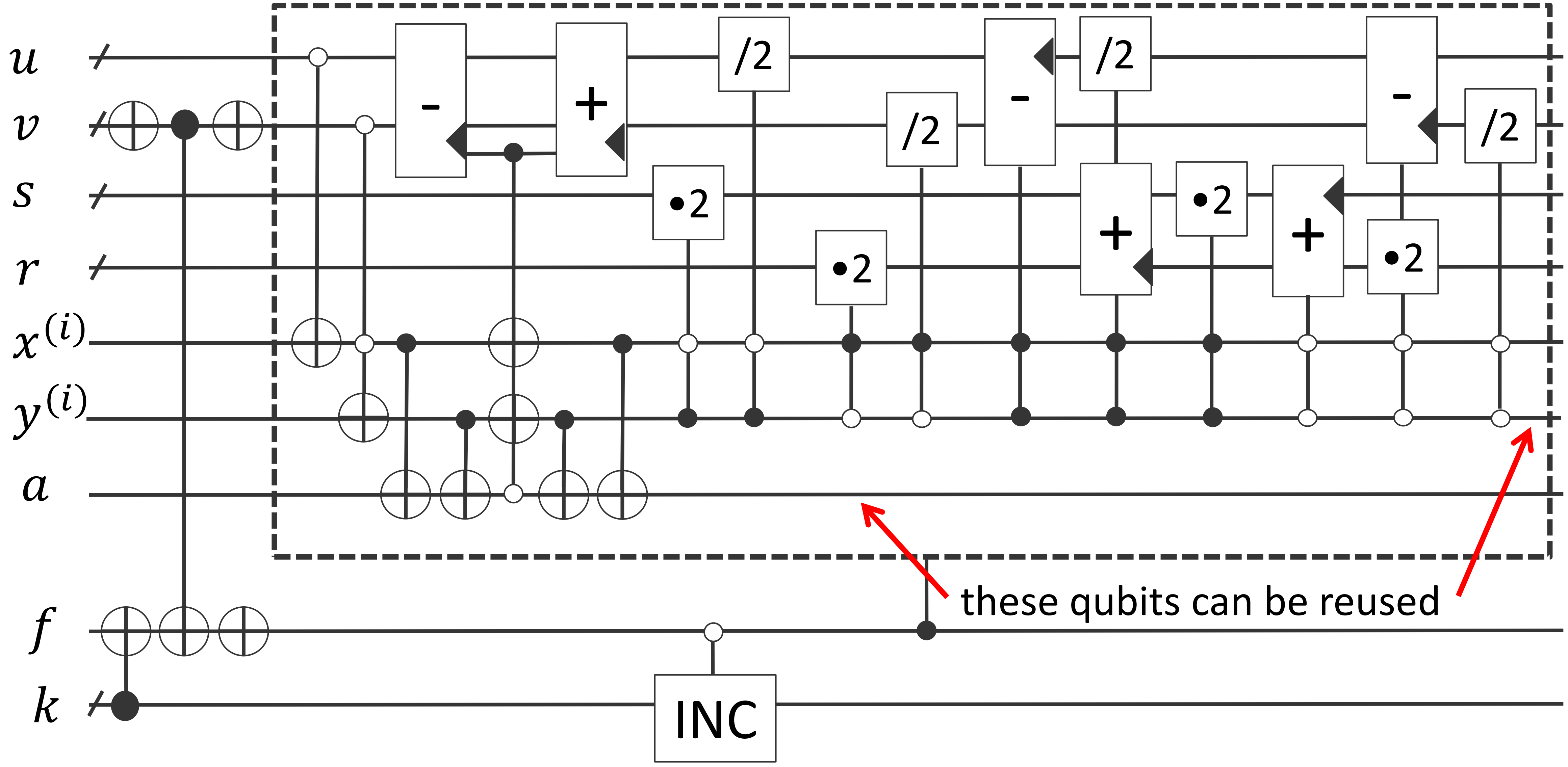
First phase
$u \leftarrow a$, $v \leftarrow b$, $r \leftarrow 0$, $s \leftarrow 1$
$k \leftarrow 0$
while $v > 0$ **do**
  if u is even **then** $u \leftarrow u/2$, $s \leftarrow 2s$
  else if v is even **then** $v \leftarrow v/2$, $r \leftarrow 2r$
  else if $u > v$ **then** $u \leftarrow (u - v)/2$, $r \leftarrow r + s$, $s \leftarrow 2s$
  else $v \leftarrow (v - u)/2$, $s \leftarrow r + s$, $r \leftarrow 2r$
  $k \leftarrow k + 1$
if $u \neq 1$ **then return** "Not relatively prime"
if $r \geq a$ then $r \leftarrow r - a$

Second phase
for $i \leftarrow 1$ to $k - n$ **do**
  if r is even **then** $r \leftarrow r/2$
  else $r \leftarrow (r + a)/2$
return $a - r$

[B. Kaliski, IEEE Trans. Comp. 44(8), 1995]

- Requires to handle a WHILE loop (with known upper bound (here: 2n))
- Implemented in LIQ*Ui*|>, including P-192, P-224, P-256, P-384, P-521

Quantum Montgomery arithmetic



Quantum circuits to attack ECC dlog

Implementing point addition: affine Weierstrass form

```
sub_const_modp x1 x2;                 //  x1 ← x1 − x2
ctrl_sub_const_modp y1 y2 ctrl;       //  y1 ← [y1 − y2]1, [y1]0
inv_modp x1 t0;
mul_modp y1 t0 λ;
mul_modp λ x1 y1;
inv_modp x1 t0;
squ_modp λ t0;
ctrl_sub_modp x1
ctrl_add_const_m
squ_modp λ t0;
mul_modp λ x1 y1;
inv_modp x1 t0;
mul_modp t0 y1 λ;
inv_modp x1 t0;
ctrl_neg_modp x1
ctrl_sub_const_m
add_const_modp x
```

Modular arithmetic circuit	# of qubits total	# of qubits ancillas	# Toffoli gates
add_const_modp, sub_const_modp	$2n$	n	$16n\log_2(n) - 26.9n$
ctrl_add_const_modp, ctrl_sub_const_modp	$2n+1$	n	$16n\log_2(n) - 26.9n$
ctrl_sub_modp	$2n+4$	3	$16n\log_2(n) - 23.8n$
ctrl_neg_modp	$n+3$	2	$8n\log_2(n) - 14.5n$
mul_modp (dbl/add)	$3n+2$	2	$32n^2\log_2(n) - 59.4n^2$
mul_modp (Montgomery)	$5n+4$	$2n+4$	$16n^2\log_2(n) - 26.3n^2$
squ_modp (dbl/add)	$2n+3$	3	$32n^2\log_2(n) - 59.4n^2$
squ_modp (Montgomery)	$4n+5$	$2n+5$	$16n^2\log_2(n) - 26.3n^2$
inv_modp	$7n+2\lceil\log_2(n)\rceil+9$	$5n+2\lceil\log_2(n)\rceil+9$	$32n^2\log_2(n)$

Comparing quantum attacks

ECDLP in $E(\mathbb{F}_p)$ simulation results					Factoring of RSA modulus N interpolation from [18]		
$\lceil \log_2(p) \rceil$ bits	#Qubits	#Toffoli gates	Toffoli depth	Sim time sec	$\lceil \log_2(N) \rceil$ bits	#Qubits	#Toffoli gates
110	1014	$9.44 \cdot 10^{9}$	$8.66 \cdot 10^{9}$	273	512	1026	$6.41 \cdot 10^{10}$
160	1466	$2.97 \cdot 10^{10}$	$2.73 \cdot 10^{9}$	711	1024	2050	$5.81 \cdot 10^{11}$
192	1754	$5.30 \cdot 10^{10}$	$4.86 \cdot 10^{10}$	1 149	–	–	–
224	2042	$8.43 \cdot 10^{10}$	$7.73 \cdot 10^{10}$	1 881	2048	4098	$5.20 \cdot 10^{12}$
256	2330	$1.26 \cdot 10^{11}$	$1.16 \cdot 10^{11}$	3 848	3072	6146	$1.86 \cdot 10^{13}$
384	3484	$4.52 \cdot 10^{11}$	$4.15 \cdot 10^{11}$	17 003	7680	15362	$3.30 \cdot 10^{14}$
521	4719	$1.14 \cdot 10^{12}$	$1.05 \cdot 10^{12}$	42 888	15360	30722	$2.87 \cdot 10^{15}$

- Our implementation of ECC dlog (ECDLP) for size n scales as $448n^3 \log_2 n + O(n^3)$ Toffoli gates and $9n + \lceil \log_2 n \rceil + 10$ qubits. For RSA the scaling is $64n^3 \log_2 n + O(n^3)$ and $2n + 2$ qubits.
- Timings with respect to LIQUi|⟩ running on HP ProLiant DL580 (4 Xeons @ 2.30GHz and 3TB memory)
- For example, bit size $n = 256$ corresponds to ECDLP used in bitcoin curve `secp256k1`
- Confirms estimates by Proos & Zalka and implies that ECDLP is easier quantum target than RSA.

WWW.MICROSOFT.COM/QUANTUM

Learn more about our approa

Get started with Quantum

Invent the future