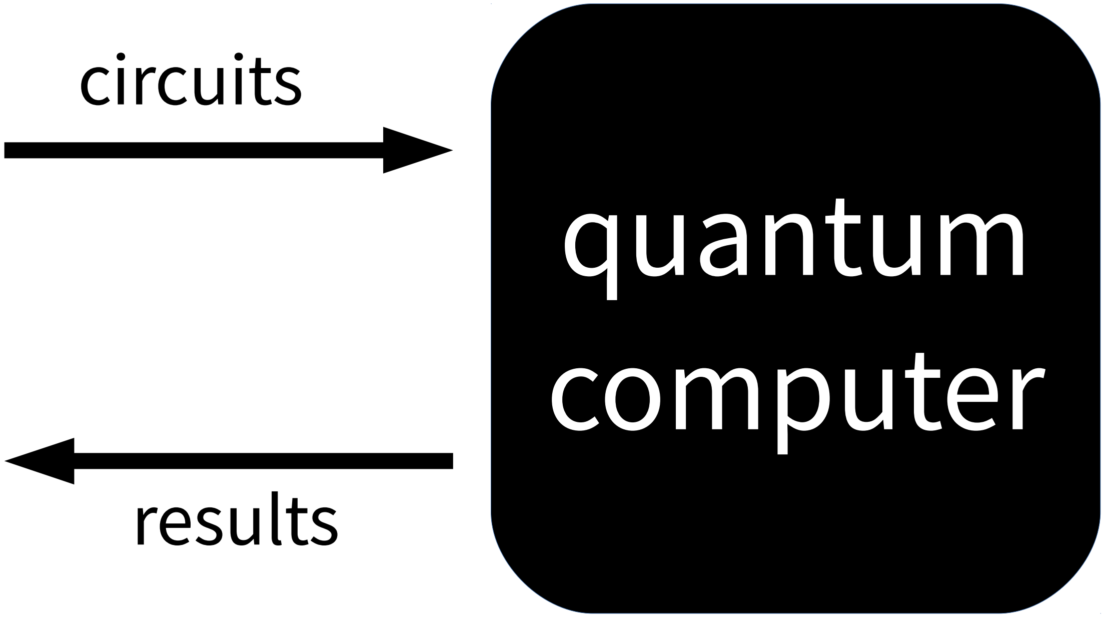
QWIRE:

A core language for quantum circuits

Jennifer Paykin, Robert Rand, Steve Zdancewic
University of Pennsylvania

POPL 2017, Paris, France

The Circuit Model



circuits

quantum computer

results

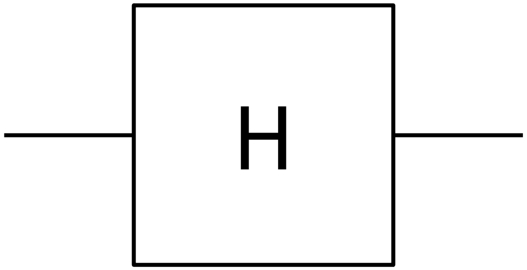
Building Blocks of Quantum Computing

Building Blocks of Quantum Computing

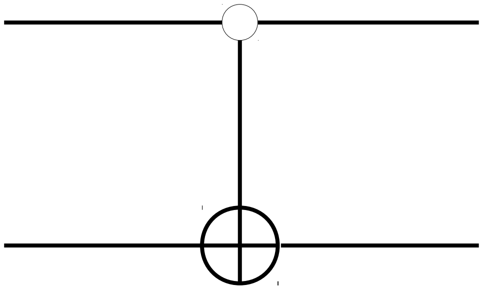
qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

Building Blocks of Quantum Computing

qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$



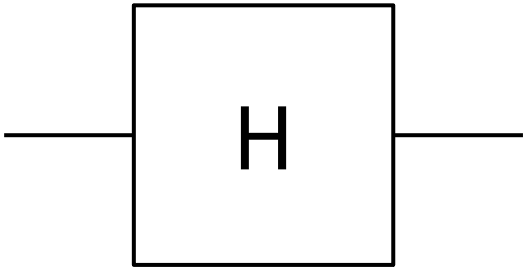
Hadamard

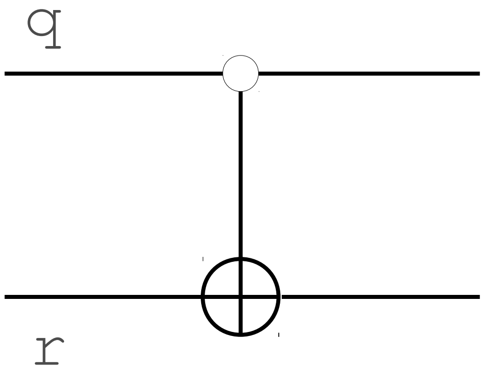


controlled not (CNOT)

Building Blocks of Quantum Computing

qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$





Hadamard

controlled not (CNOT)

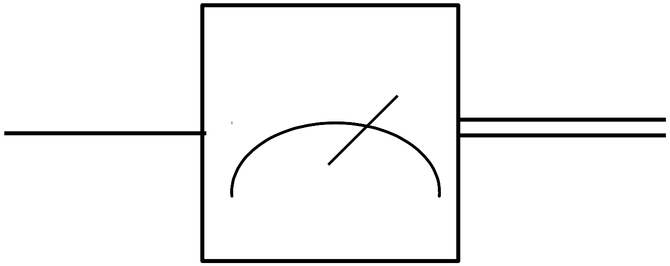
“`qif q then not r`”

Building Blocks of Quantum Computing

qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

Building Blocks of Quantum Computing

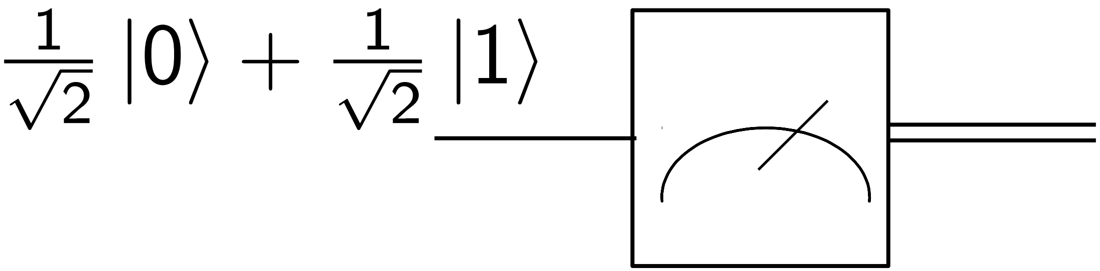
qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$



measurement

Building Blocks of Quantum Computing

qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

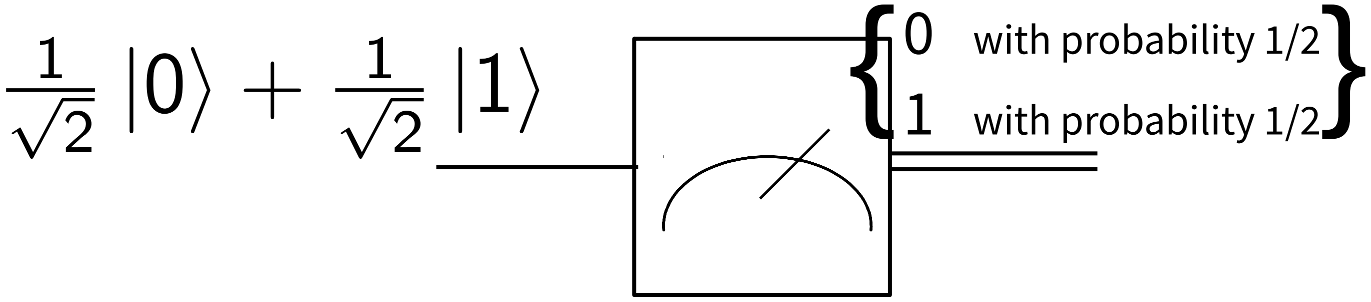


$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$

measurement

Building Blocks of Quantum Computing

qubits $|0\rangle$ or $|1\rangle$ or $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$



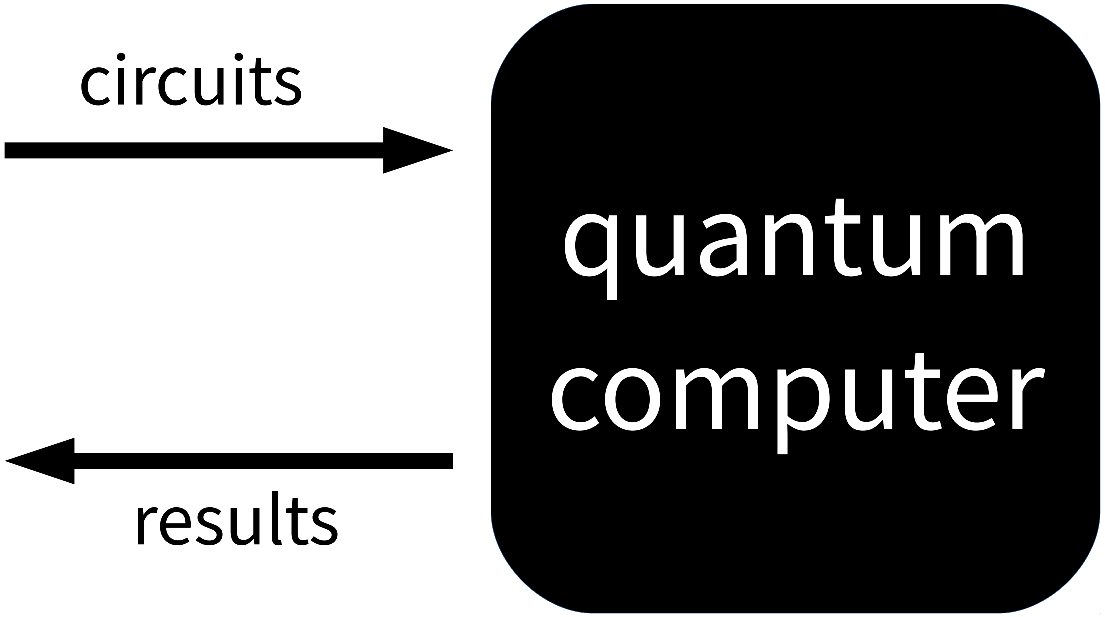
$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

$$\left\{\begin{matrix} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{matrix}\right\}$$

measurement

The QRAM Model of Quantum Computing

Knill, 1996



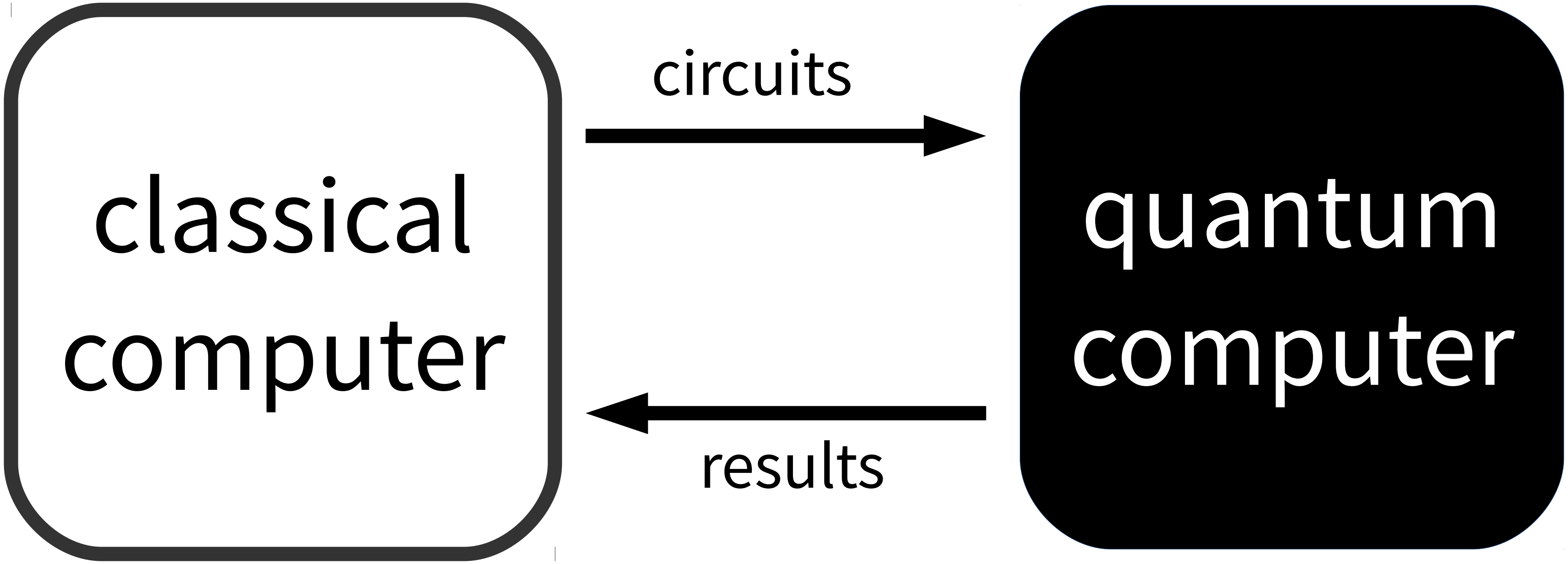
circuits

quantum computer

results

The QRAM Model of Quantum Computing

Knill, 1996



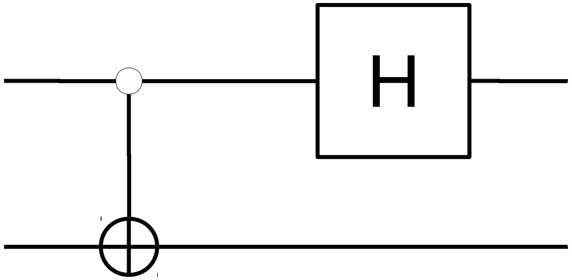
classical computer

circuits

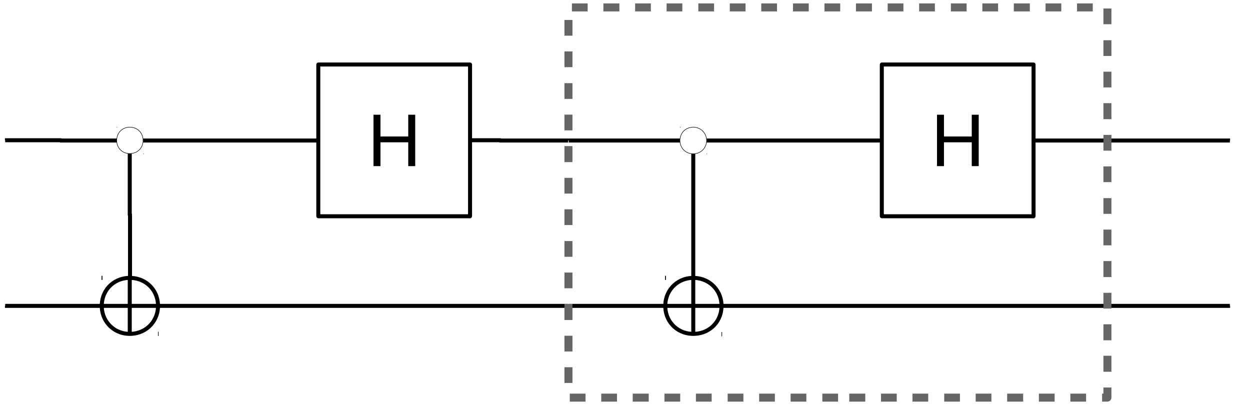
results

quantum computer

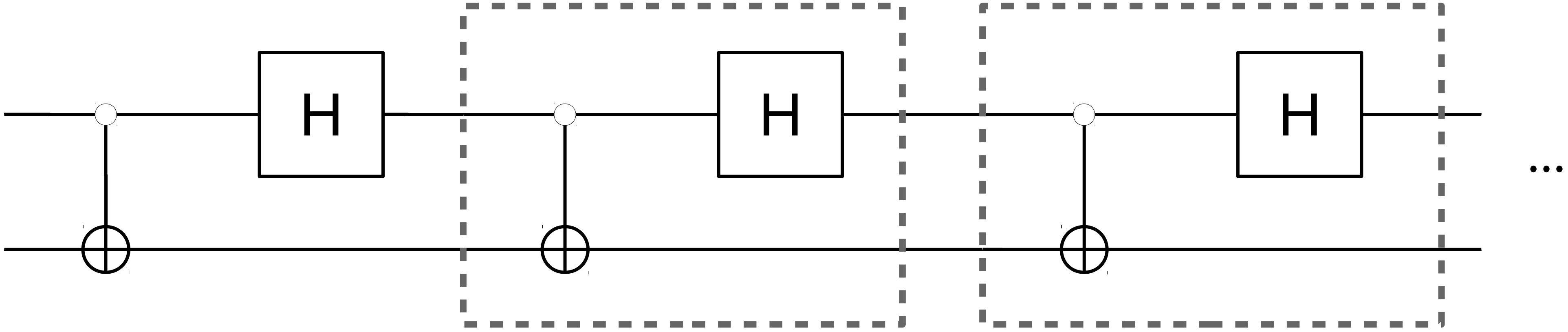
n-ary Composition



n-ary Composition

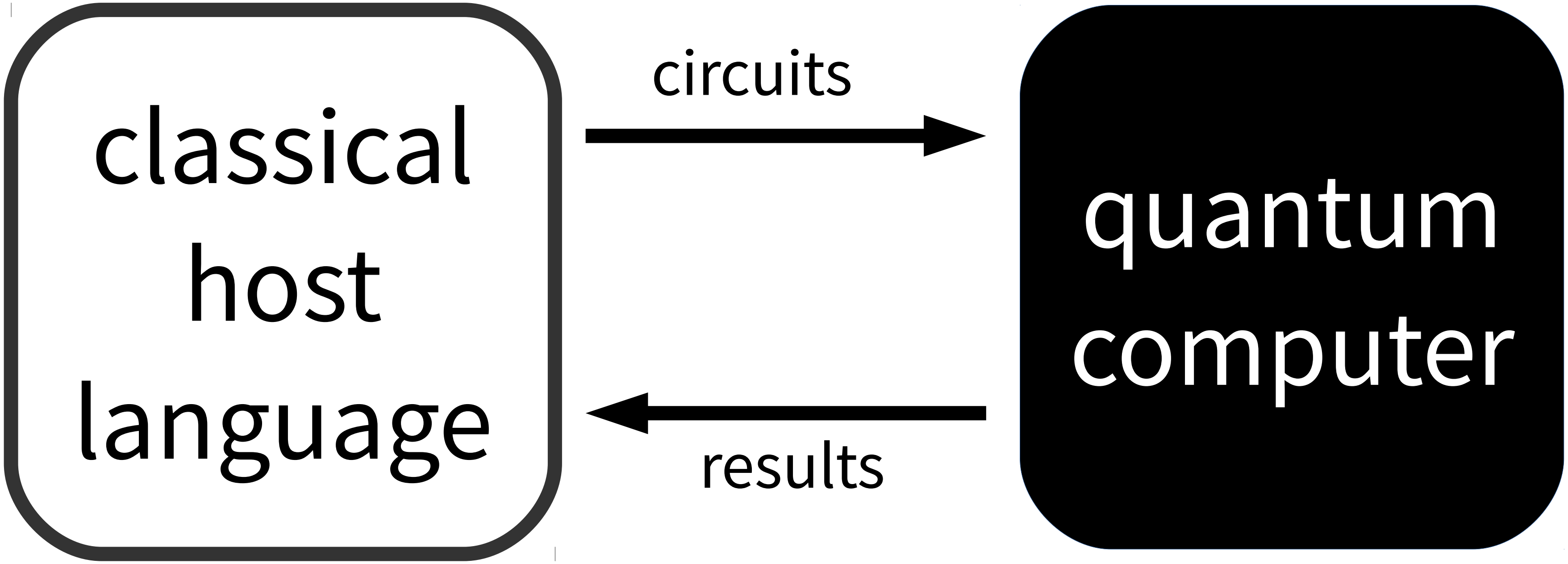


n-ary Composition



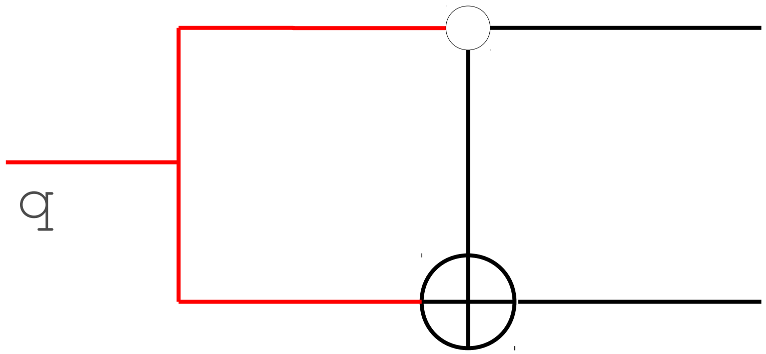
The QRAM Model of Quantum Computing

Knill, 1996



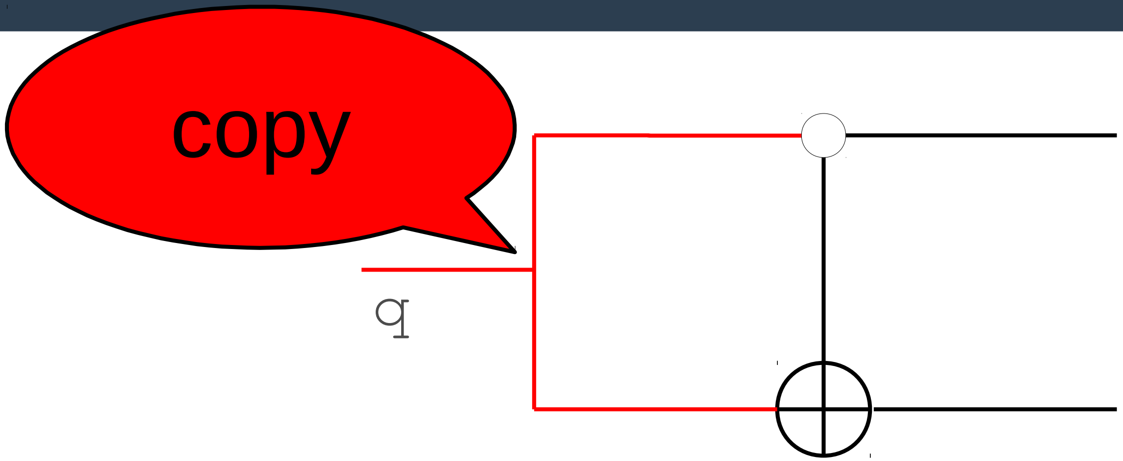
Quipper (Green et al, 2013)
LIQ$Ui|\rangle$ (Wecker and Svore, 2014)

Problem: Ill-Formed Circuits



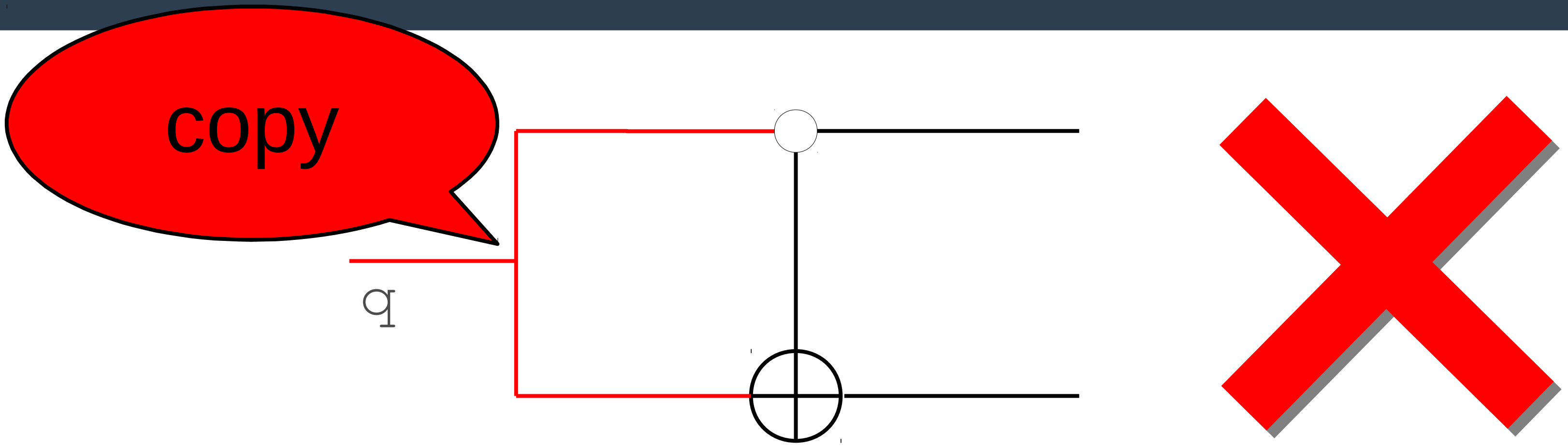
“qif q then not q”

Problem: Ill-Formed Circuits



“qif q then not q”

Problem: Ill-Formed Circuits

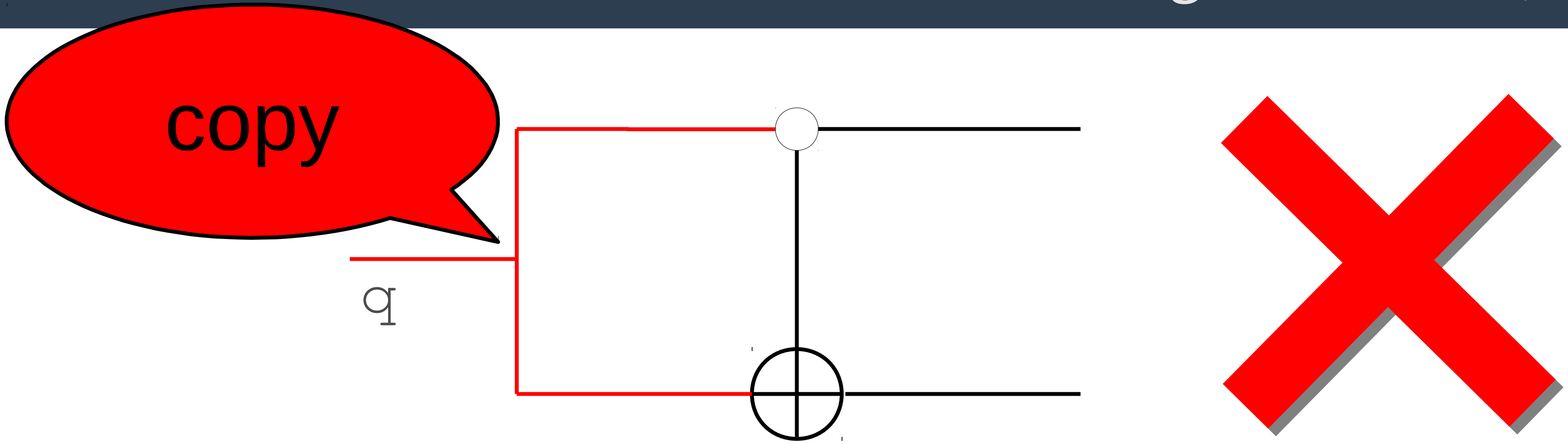


“`qif q then not q`”

no-cloning theorem: a qubit cannot be copied

Problem: Ill-Formed Circuits

Selinger & Valiron, 2009

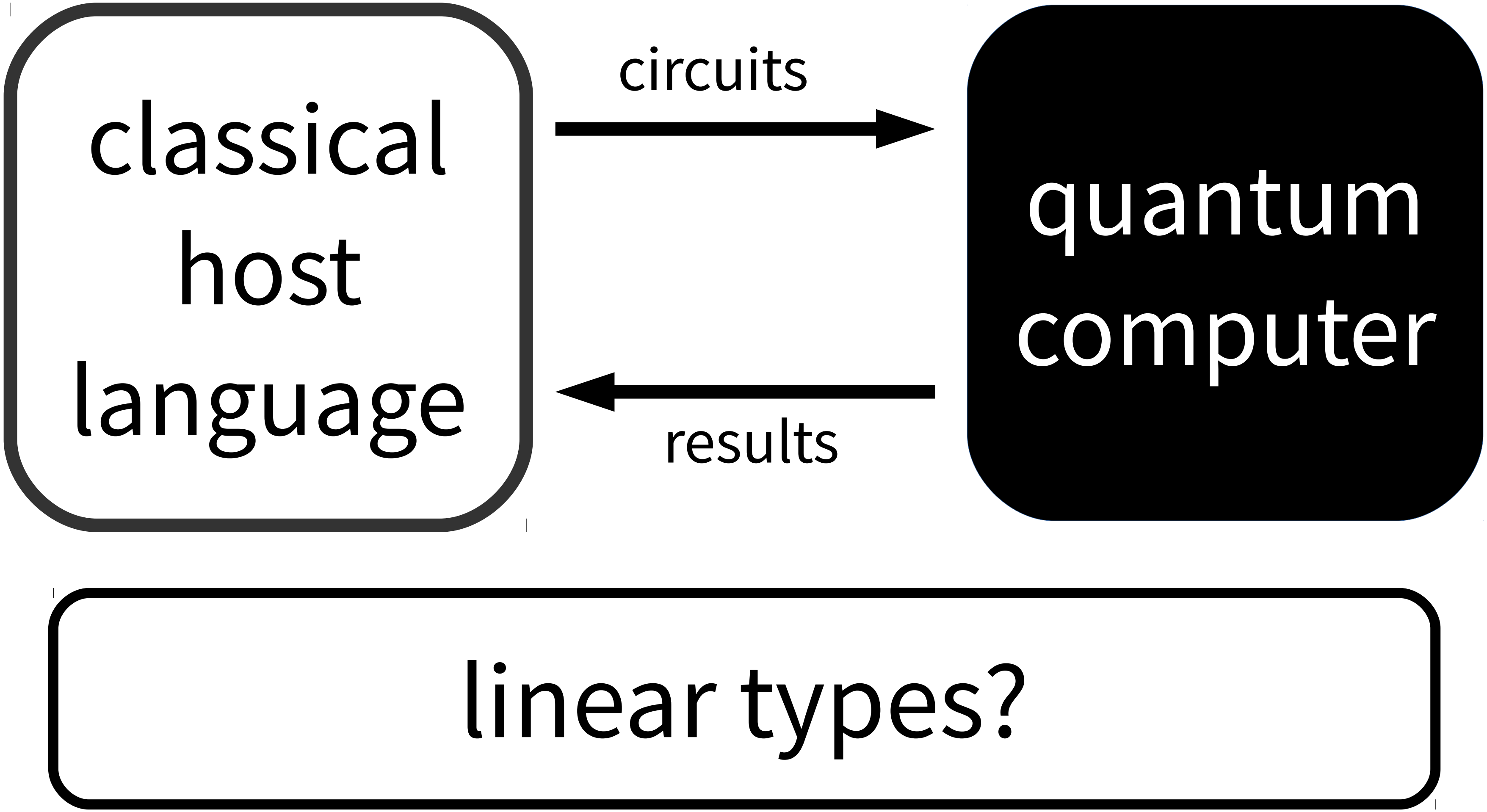


`“qif q then not q”`

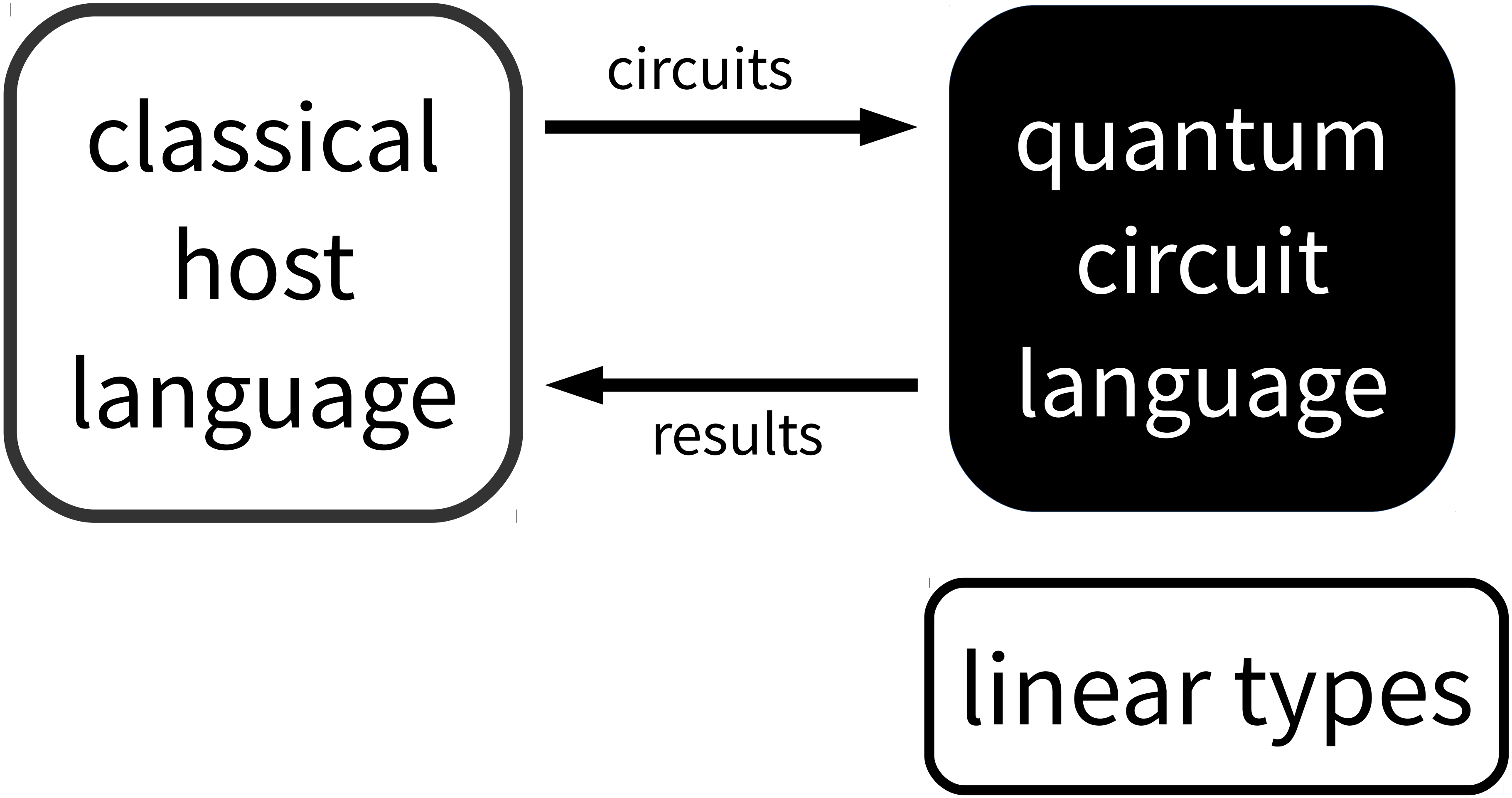
no-cloning theorem: a qubit cannot be copied

linear types ⇒ no cloning

A linear QRAM model?



Introducing $\mathcal{Q}$WIRE...



classical host language

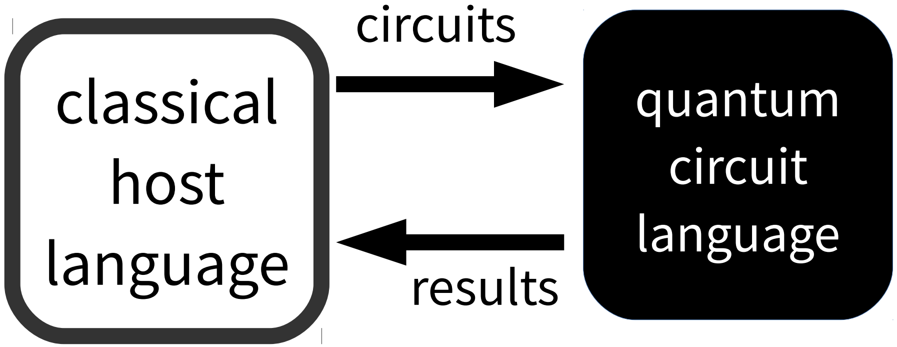
circuits

quantum circuit language

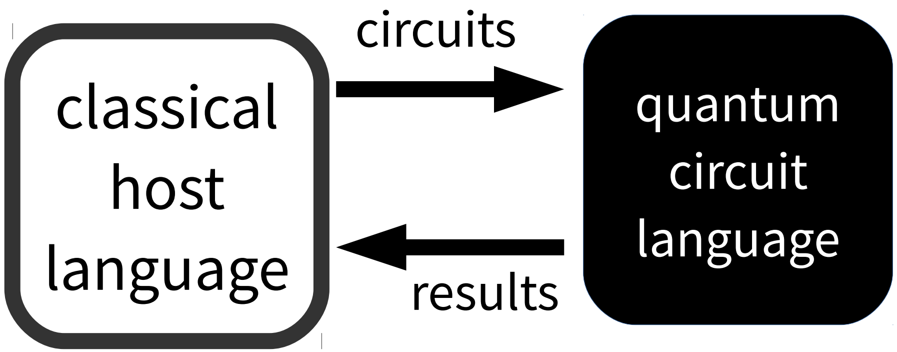
results

linear types

Introducing $\mathcal{Q}$WIRE…

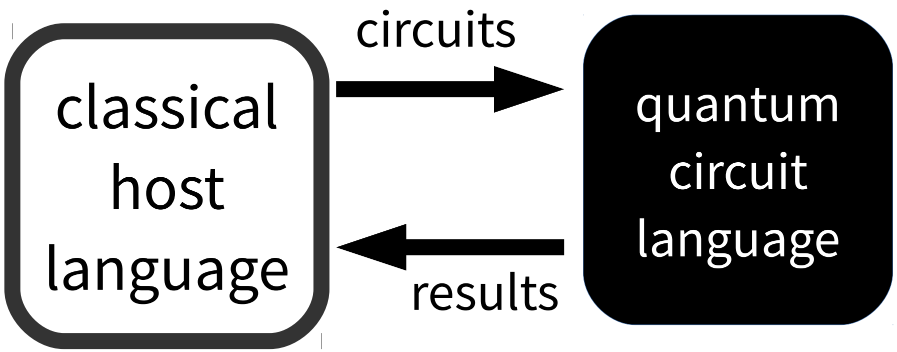


Introducing $\mathcal{Q}$WIRE…



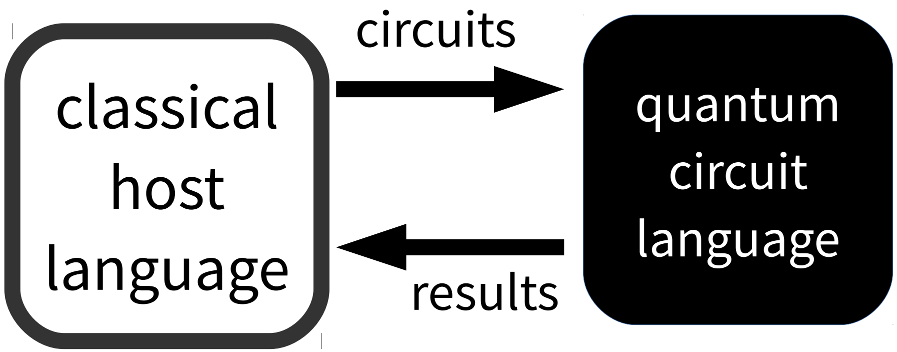
- A core language for quantum circuits

Introducing $\mathcal{Q}$WIRE…



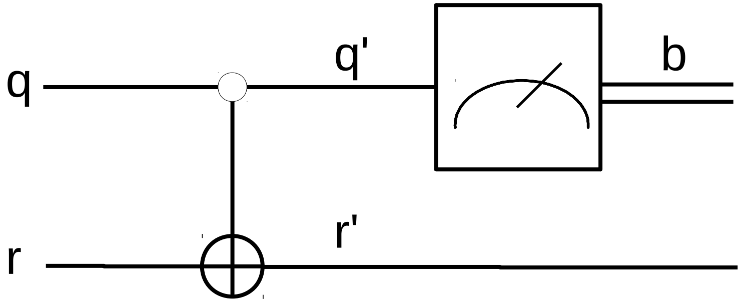
- A core language for quantum circuits
- Safe
  - linear types for wires
  - type safety & strong normalization
  - denotational semantics

Introducing $\mathcal{Q}$WIRE…



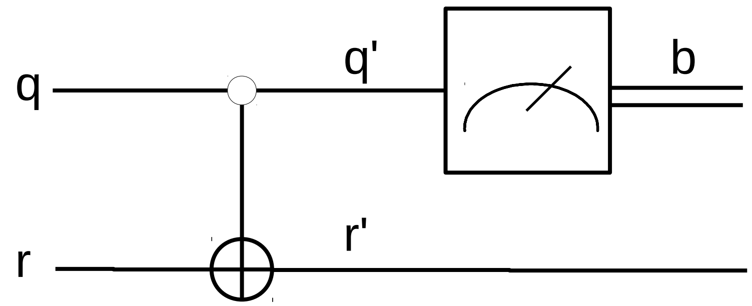
- A core language for quantum circuits
- Safe
  - linear types for wires
  - type safety & strong normalization
  - denotational semantics
- Expressive
  - structure based on the QRAM model
  - embedded in your favorite classical host language

Quantum Circuits



```
(q',r') ← gate CNOT (q,r);
b       ← gate meas q';
output (b,r')
```

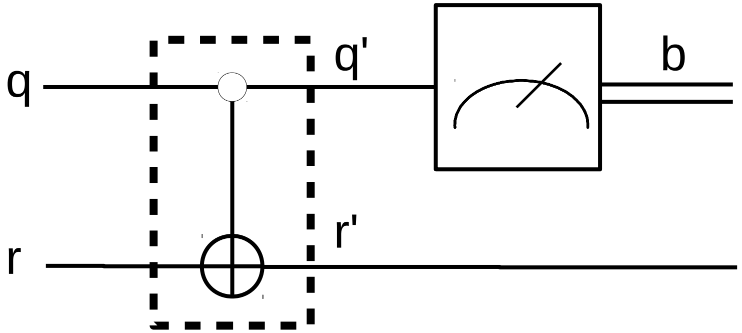
Quantum Circuits



wire names

```
(q',r') ← gate CNOT (q,r);
b       ← gate meas q';
output (b,r')
```

Quantum Circuits

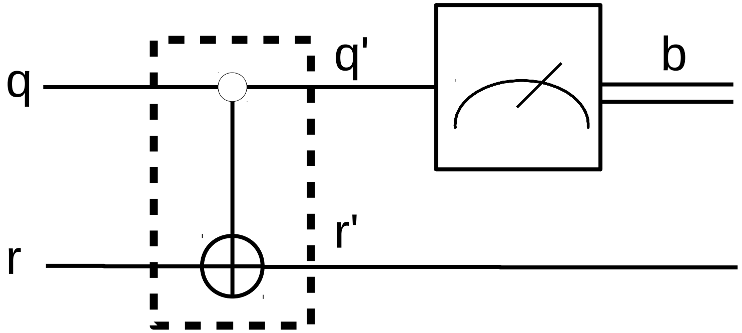


wire names

```
(q',r')  ← gate CNOT (q,r);
b        ← gate meas q';
output (b,r')
```

Quantum Circuits

let binding

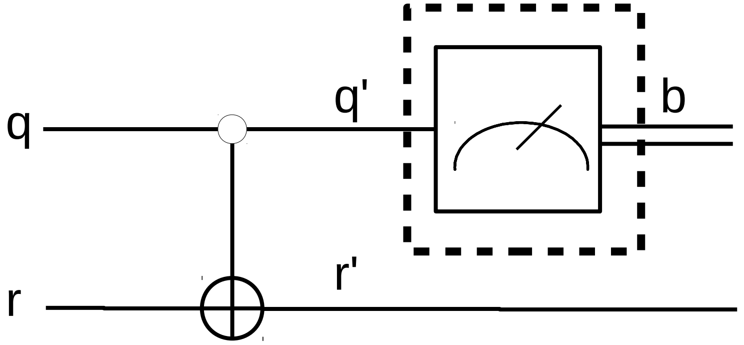


wire names

```
(q',r')  ← gate CNOT (q,r);
b        ← gate meas q';
output (b,r')
```

Quantum Circuits

let binding

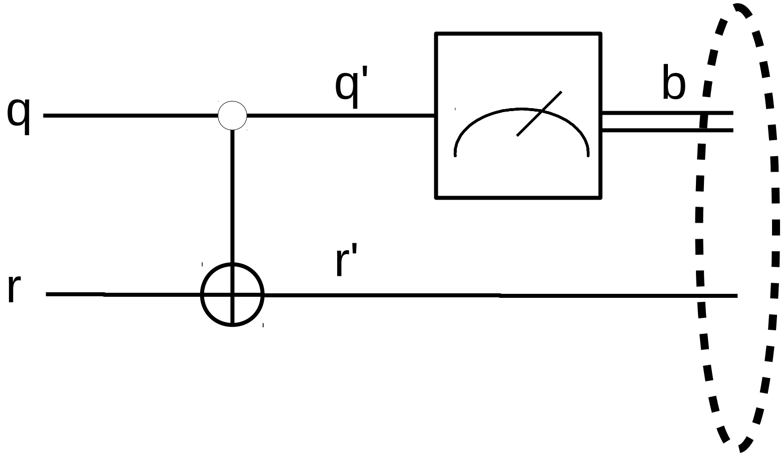


wire names

```
(q',r') ← gate CNOT (q,r);
b       ← gate meas q';
output (b,r')
```

Quantum Circuits

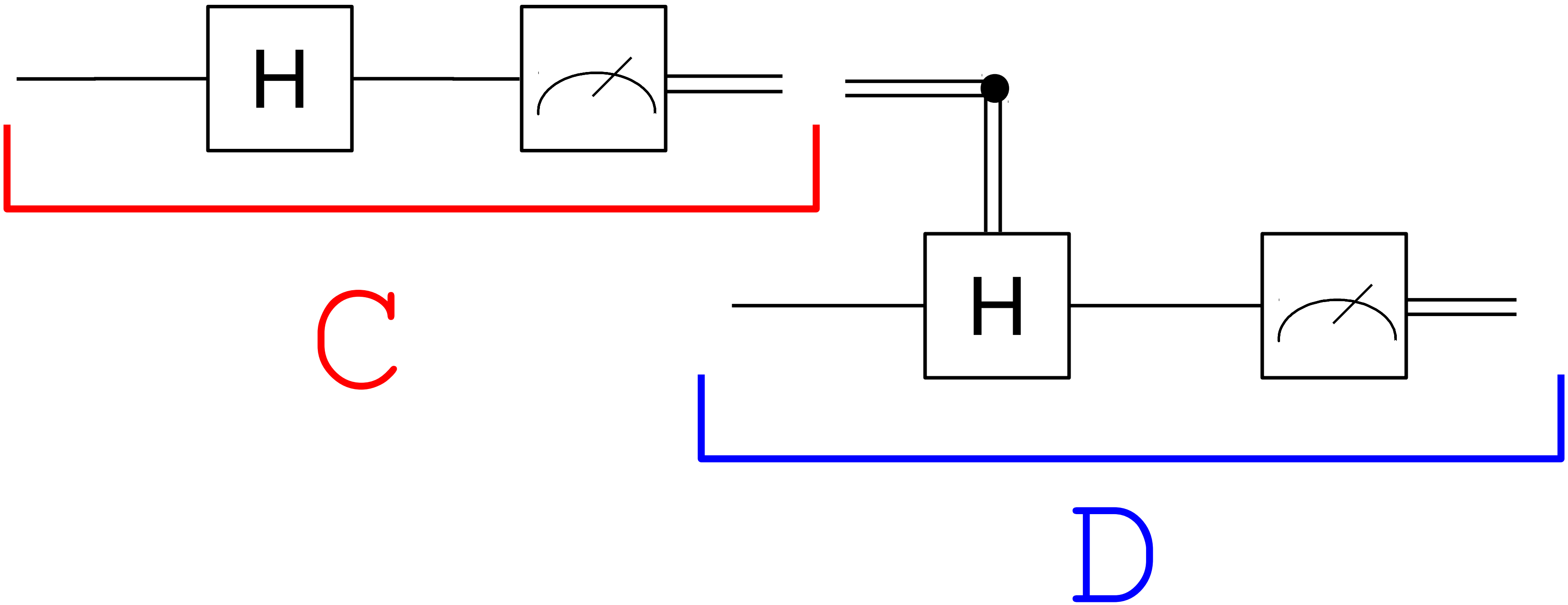
let binding



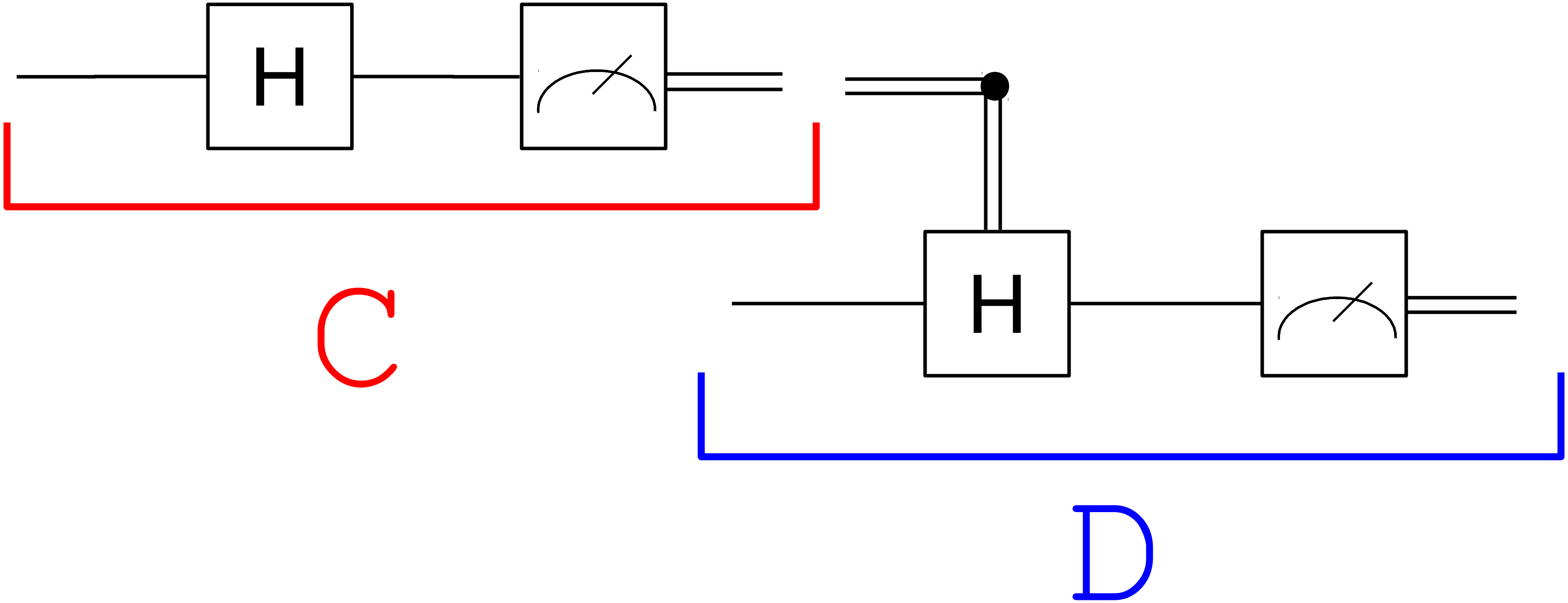
wire names

```
(q',r')  ← gate CNOT (q,r);
b        ← gate meas q';
output (b,r')
```

Composition

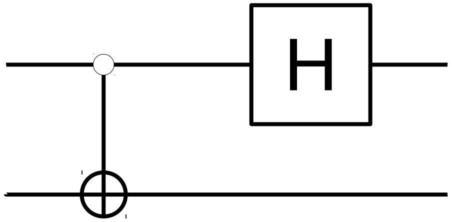


Composition

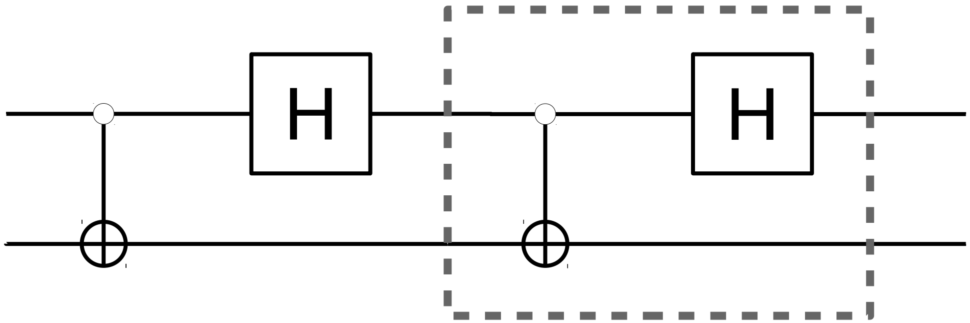


```
b ← C; D
```

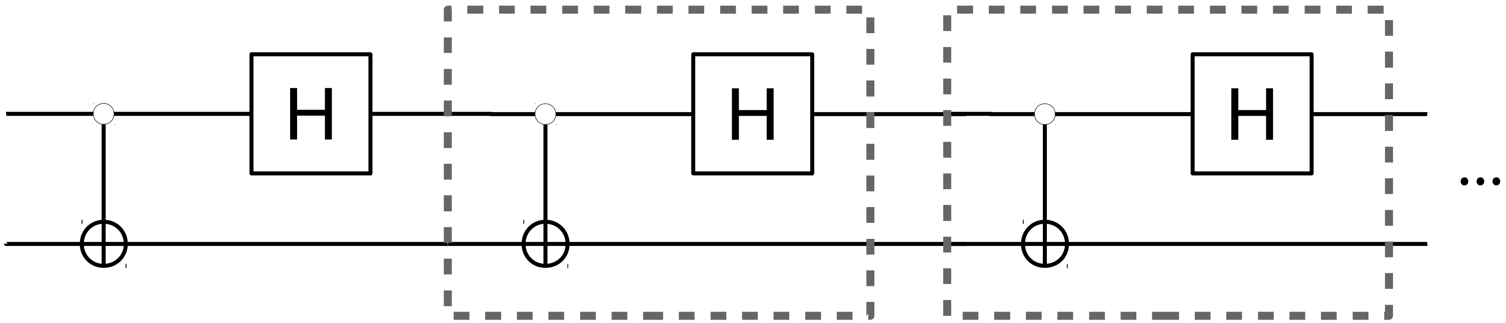
n-ary Composition?



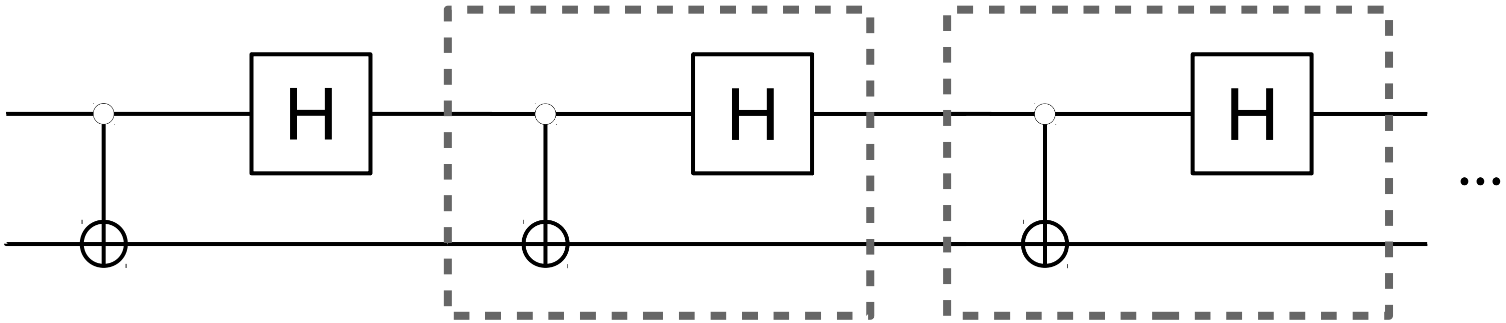
n-ary Composition?



n-ary Composition?

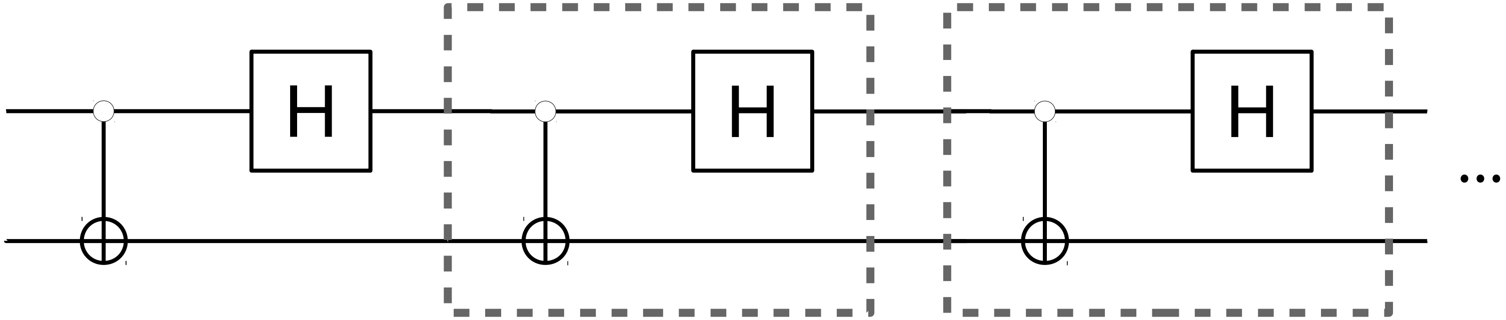


n-ary Composition?



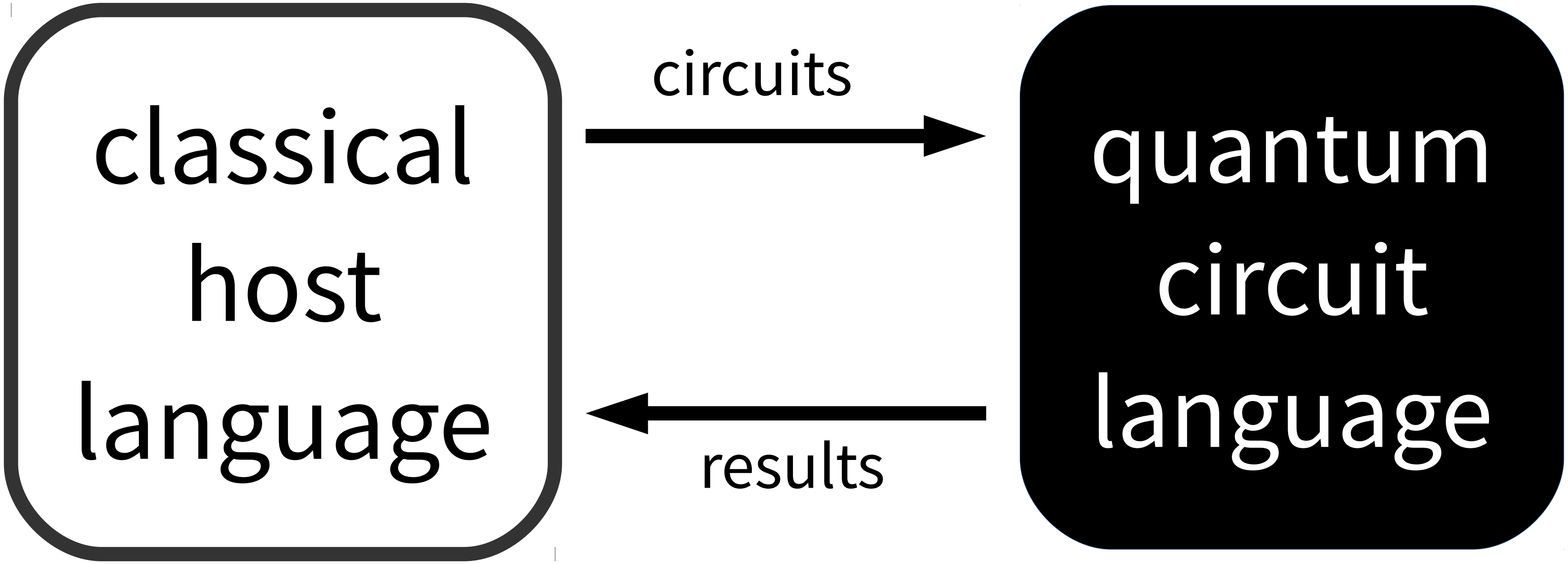
```
(q,r)  ←  gate CNOT (q,r);
q      ←  gate H q;
```

n-ary Composition?



```
(q,r) ← gate CNOT (q,r);
q     ← gate H q;
(q,r) ← gate CNOT (q,r);
q     ← gate H q;
...
```

Typing Judgments in $\mathcal{Q}$WIRE



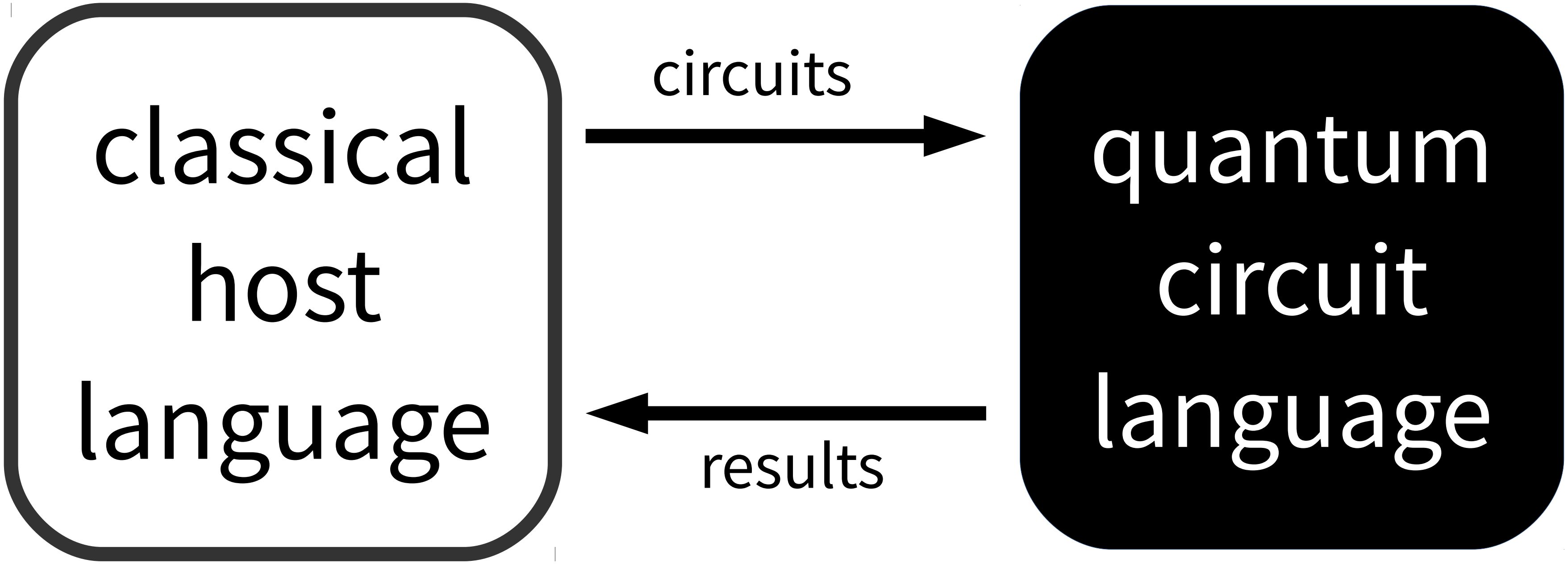
classical host language

circuits

results

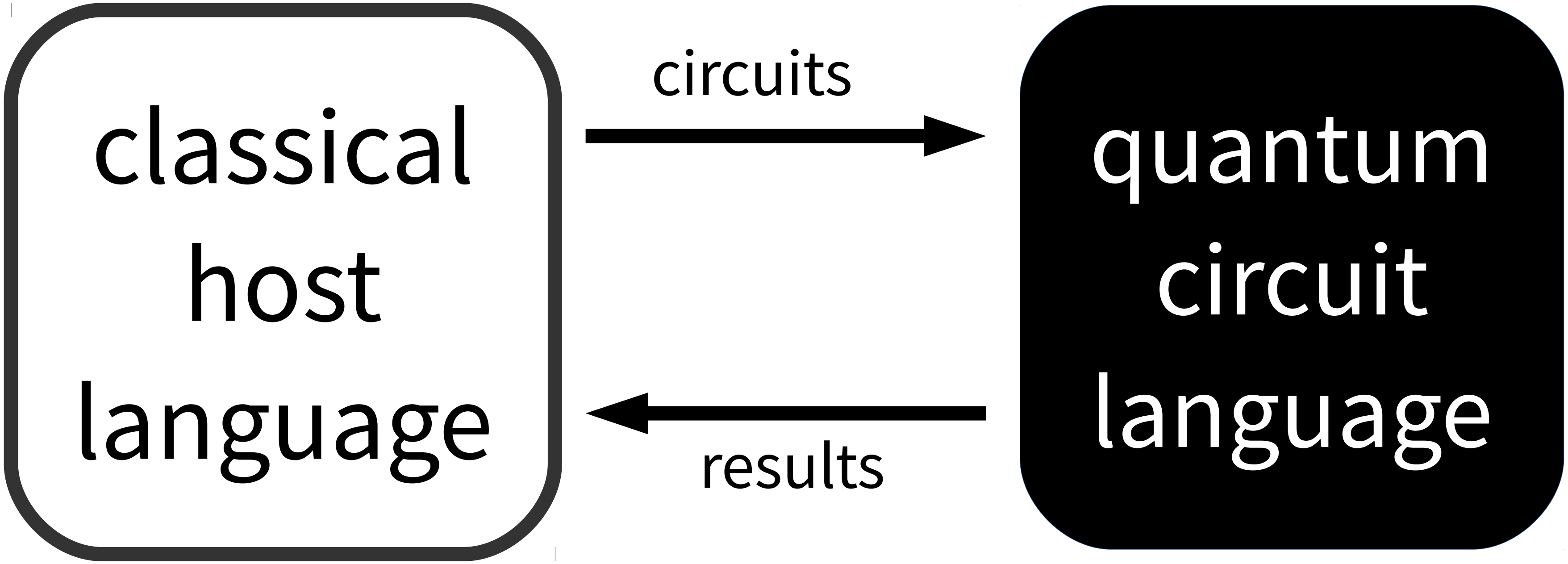
quantum circuit language

Typing Judgments in $\mathcal{Q}$WIRE



$$\Gamma \vdash t : A$$

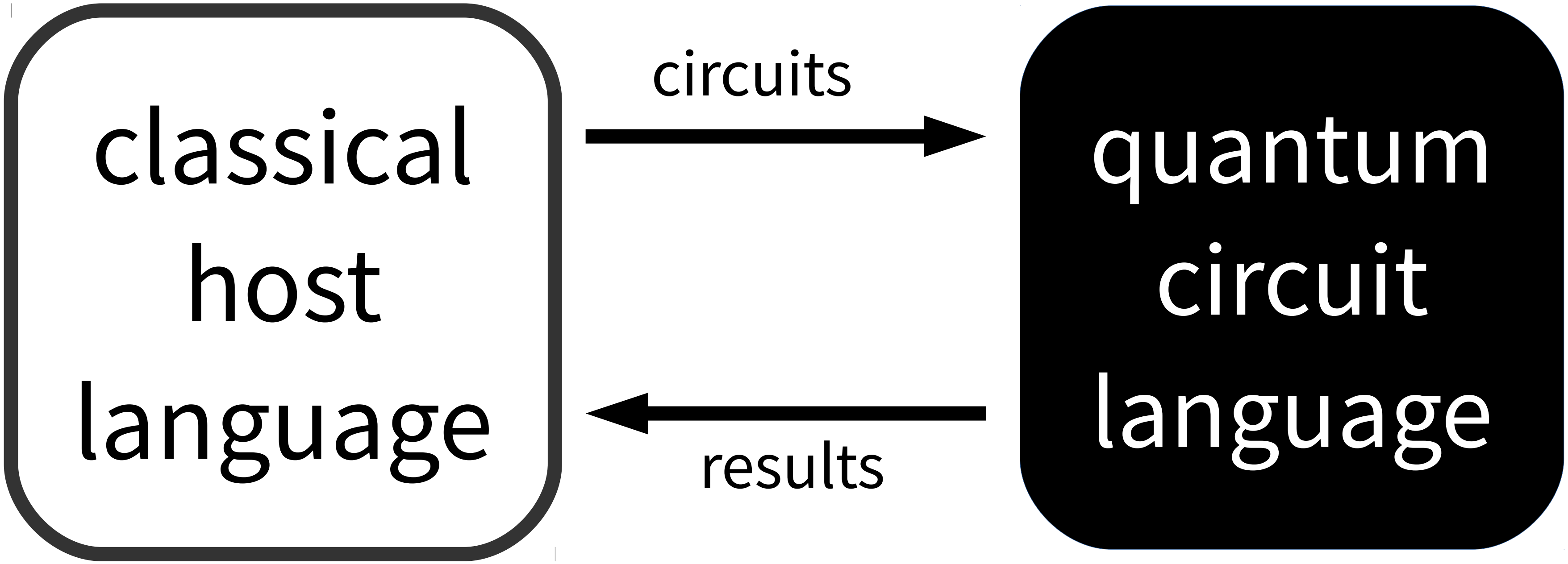
Typing Judgments in $\mathcal{Q}$WIRE



$$\Gamma \vdash t : A$$

$$\Gamma; \Delta \vdash C : W$$

Typing Judgments in $\mathcal{Q}$WIRE


classical host language

circuits

results

quantum circuit language

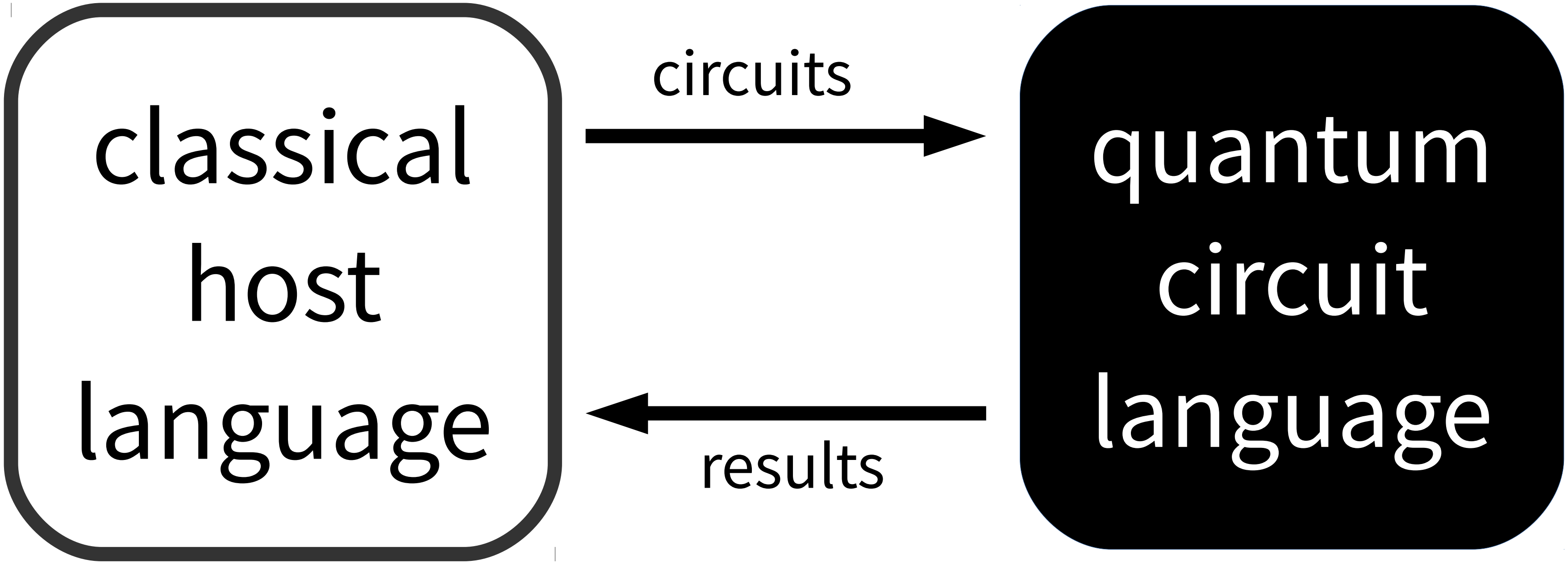
$$\Gamma \vdash t : A$$

$$\Gamma ; \underline{\Delta} \vdash C : \underline{W}$$

linear wire types

Typing Judgments in $\mathcal{Q}$WIRE

Benton, 1995


classical host language

circuits

results

quantum circuit language

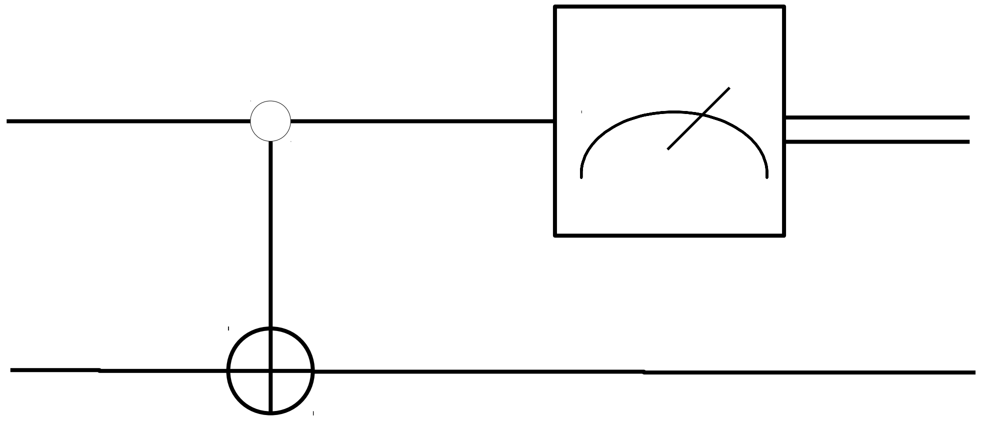
$$\underline{\Gamma} \vdash t : \underline{A}$$

$$\underline{\Gamma}; \underline{\Delta} \vdash C : \underline{W}$$

host language types

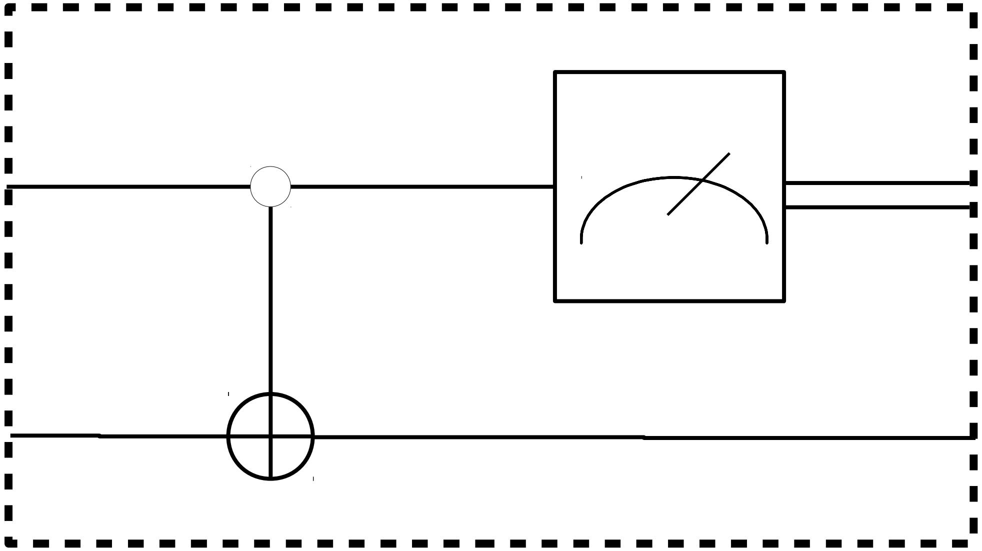
linear wire types

Boxes



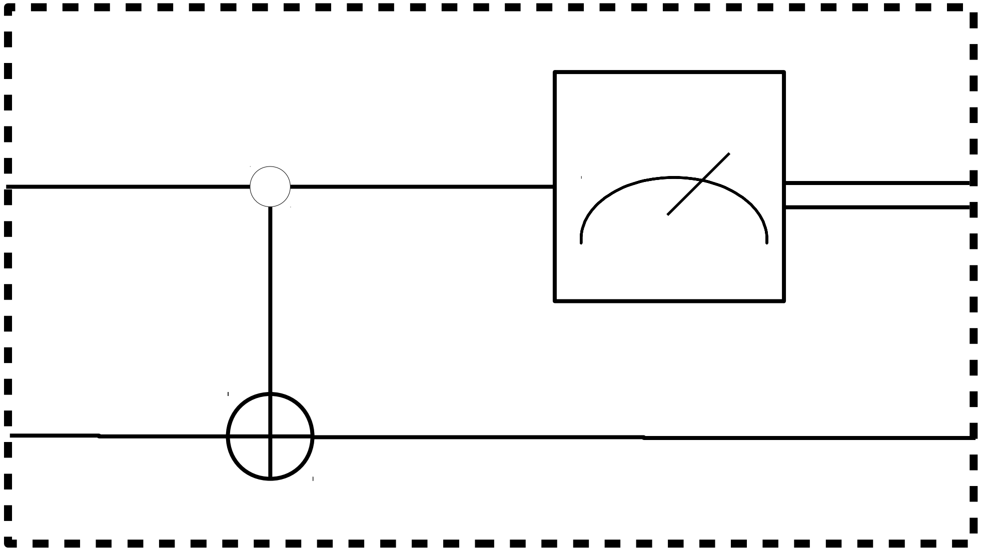
$$\Gamma; q : \texttt{qubit}, r : \texttt{qubit} \vdash C : \texttt{bit} \otimes \texttt{qubit}$$

Boxes



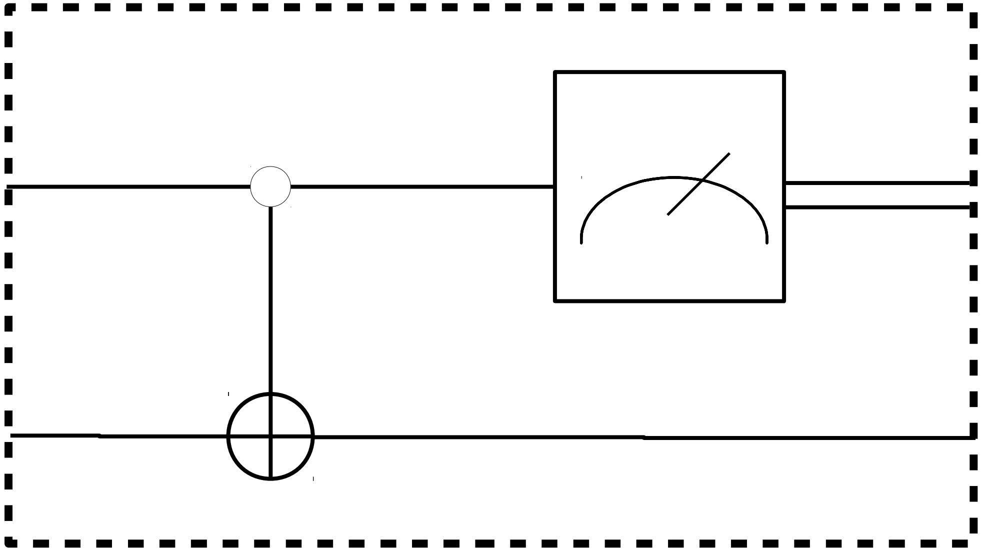
$$\frac{\Gamma; q : \mathtt{qubit}, r : \mathtt{qubit} \vdash C : \mathtt{bit} \otimes \mathtt{qubit}}{\Gamma \vdash \mathsf{box}\,(q,r) \Rightarrow C \\ : \mathsf{Circ}(\mathtt{qubit} \otimes \mathtt{qubit}, \mathtt{bit} \otimes \mathtt{qubit})}$$

Boxes



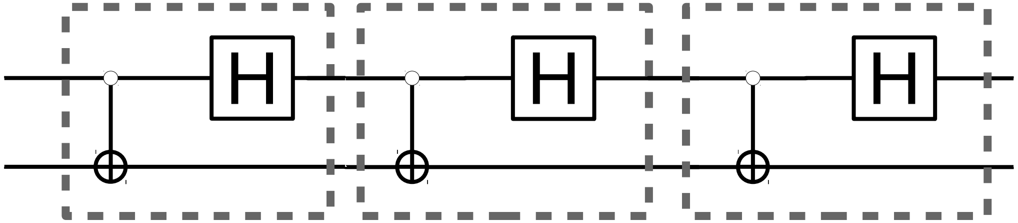
$$\frac{\Gamma; q : \underline{\texttt{qubit}}, r : \underline{\texttt{qubit}} \vdash C : \texttt{bit} \otimes \texttt{qubit}}{\begin{array}{l}\Gamma \vdash \mathsf{box}\,(q, r) \Rightarrow C \\ \qquad : \mathsf{Circ}(\underline{\texttt{qubit} \otimes \texttt{qubit}}, \texttt{bit} \otimes \texttt{qubit})\end{array}}$$

Boxes

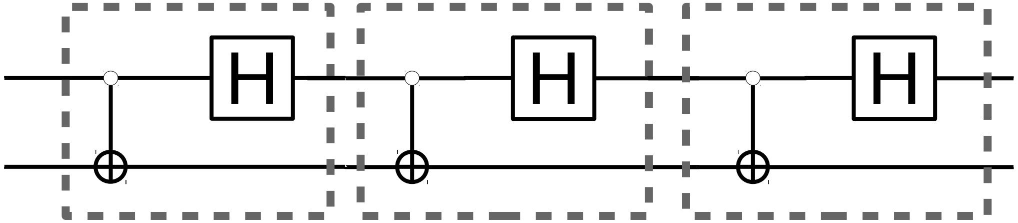


$$\frac{\Gamma; q : \underline{\texttt{qubit}}, r : \underline{\texttt{qubit}} \vdash C : \underline{\texttt{bit} \otimes \texttt{qubit}}}{\Gamma \vdash \mathsf{box}\,(q, r) \Rightarrow C \; : \mathsf{Circ}(\underline{\texttt{qubit} \otimes \texttt{qubit}}, \underline{\texttt{bit} \otimes \texttt{qubit}})}$$

n-ary Composition

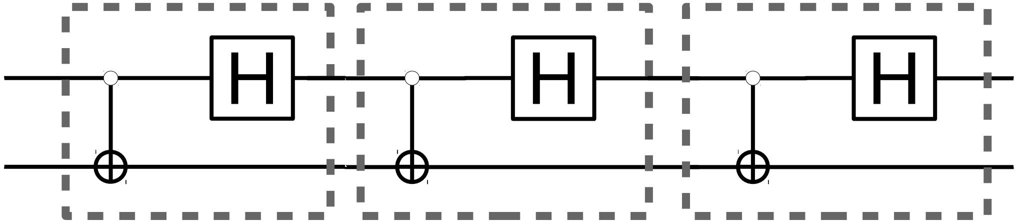


n-ary Composition



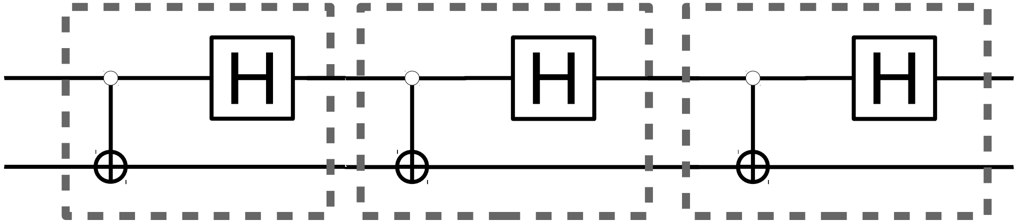
```
inSeqN (n : Nat) (c : Circ(W,W))
                     : Circ(W,W) =
  case n of
```

n-ary Composition



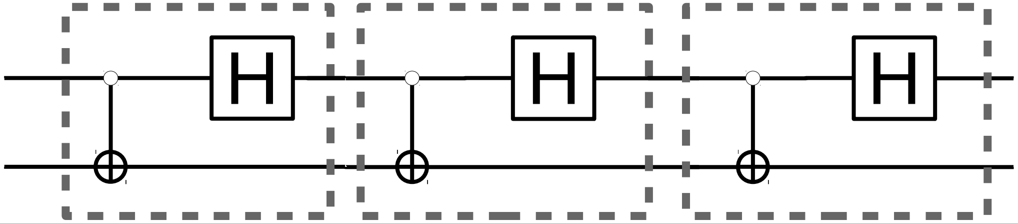
```
inSeqN (n : Nat) (c : Circ(W,W))
                    : Circ(W,W) =
  case n of

    | 0   => box w ⇒ output w
```

n-ary Composition



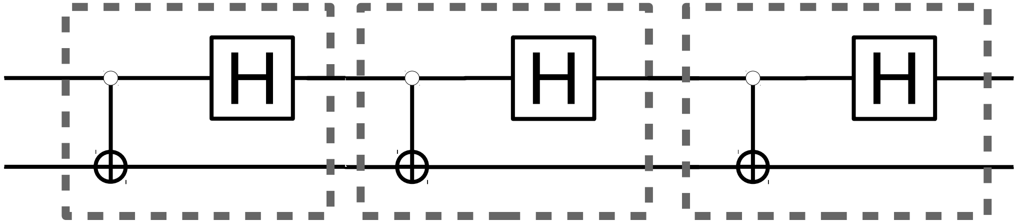
```
inSeqN (n : Nat) (c : Circ(W,W))
                     : Circ(W,W) =
  case n of

    | 0    => box w ⇒ output w

    | m+1  => box w ⇒
```

n-ary Composition



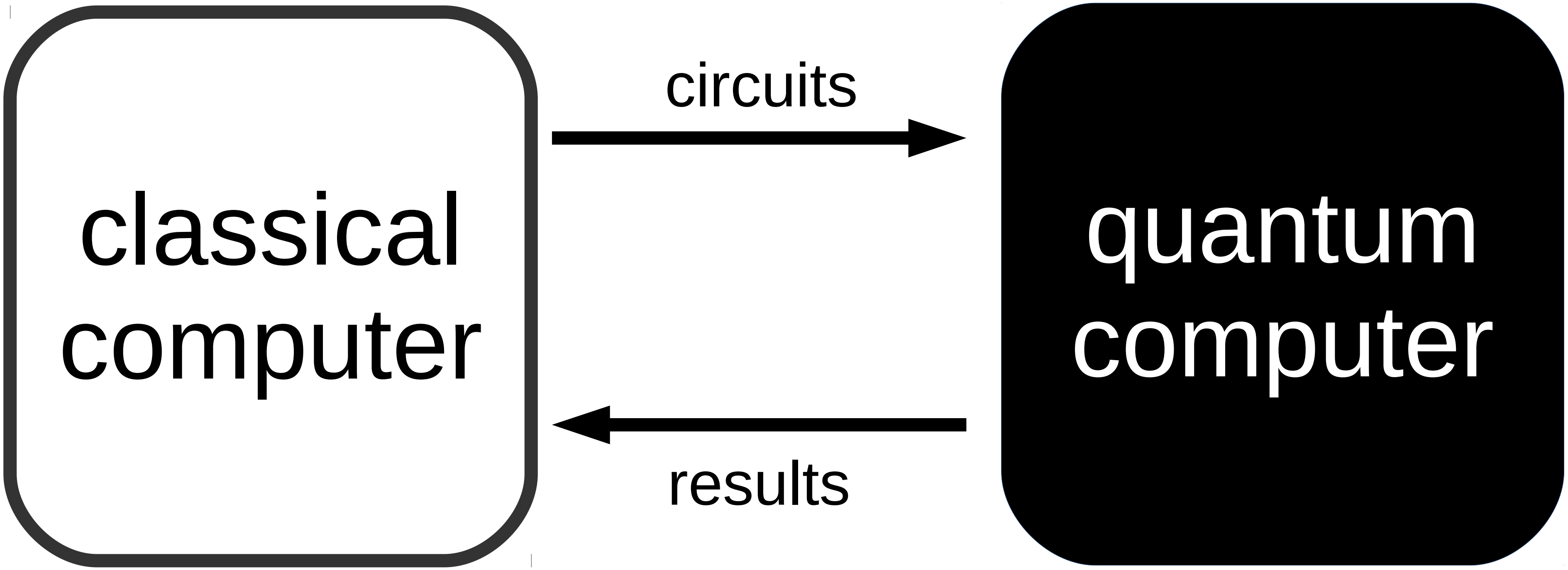
```
inSeqN (n : Nat) (c : Circ(W,W))
                    : Circ(W,W) =
  case n of

    | 0    => box w ⇒ output w

    | m+1  => box w ⇒
                  w' ← unbox c w;
```

n-ary Composition

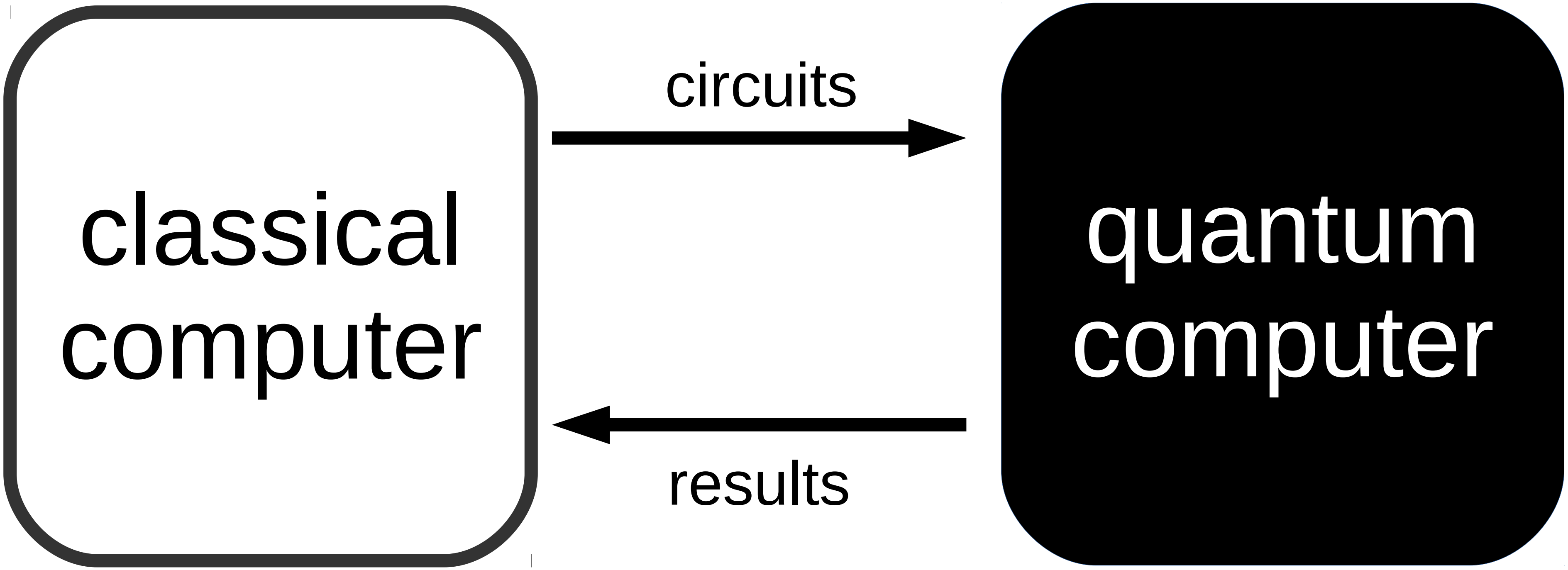


```
inSeqN (n : Nat) (c : Circ(W,W))
                     : Circ(W,W) =
  case n of

    | 0    => box w ⇒ output w

    | m+1  => box w ⇒
                w' ← unbox c w;
                unbox (inSeqN m) w'
```

Communication



Running Circuits



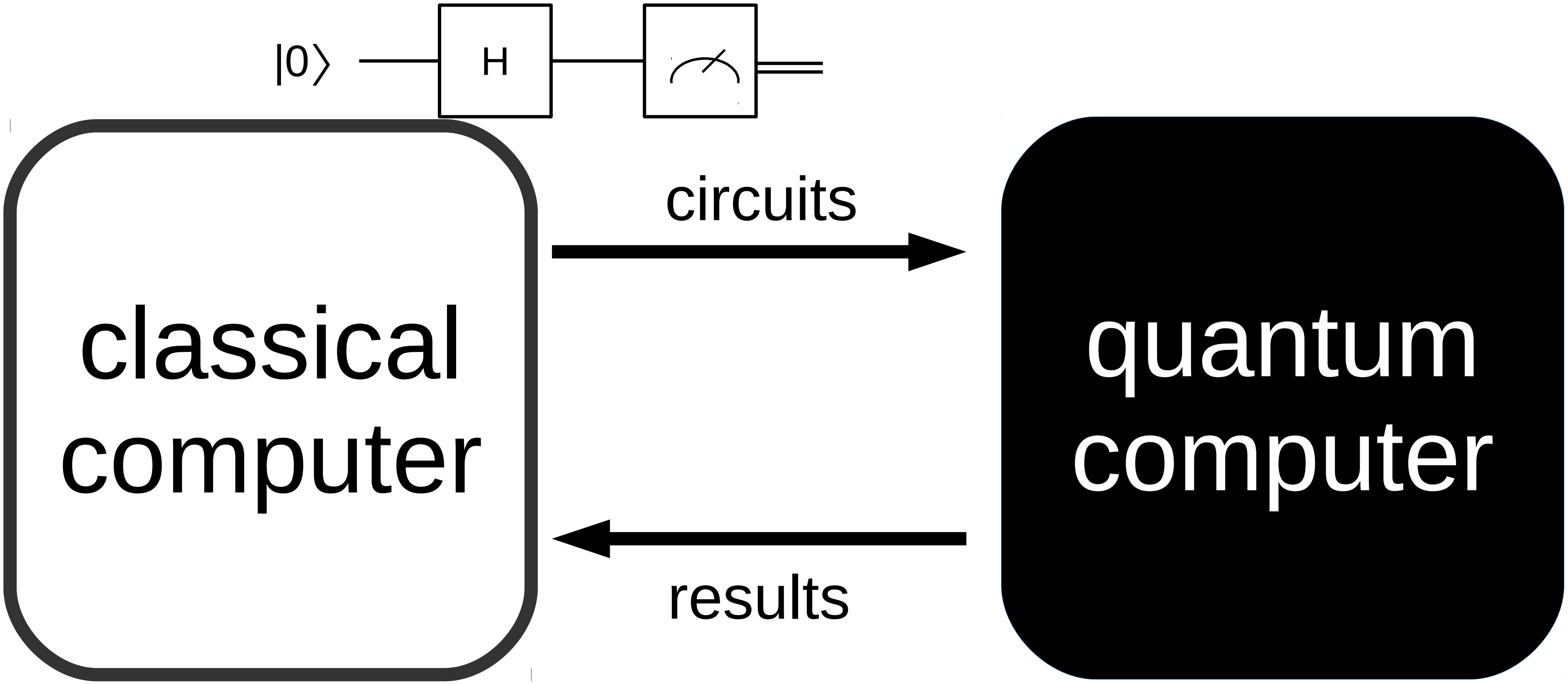
classical computer

circuits

results

quantum computer

Running Circuits



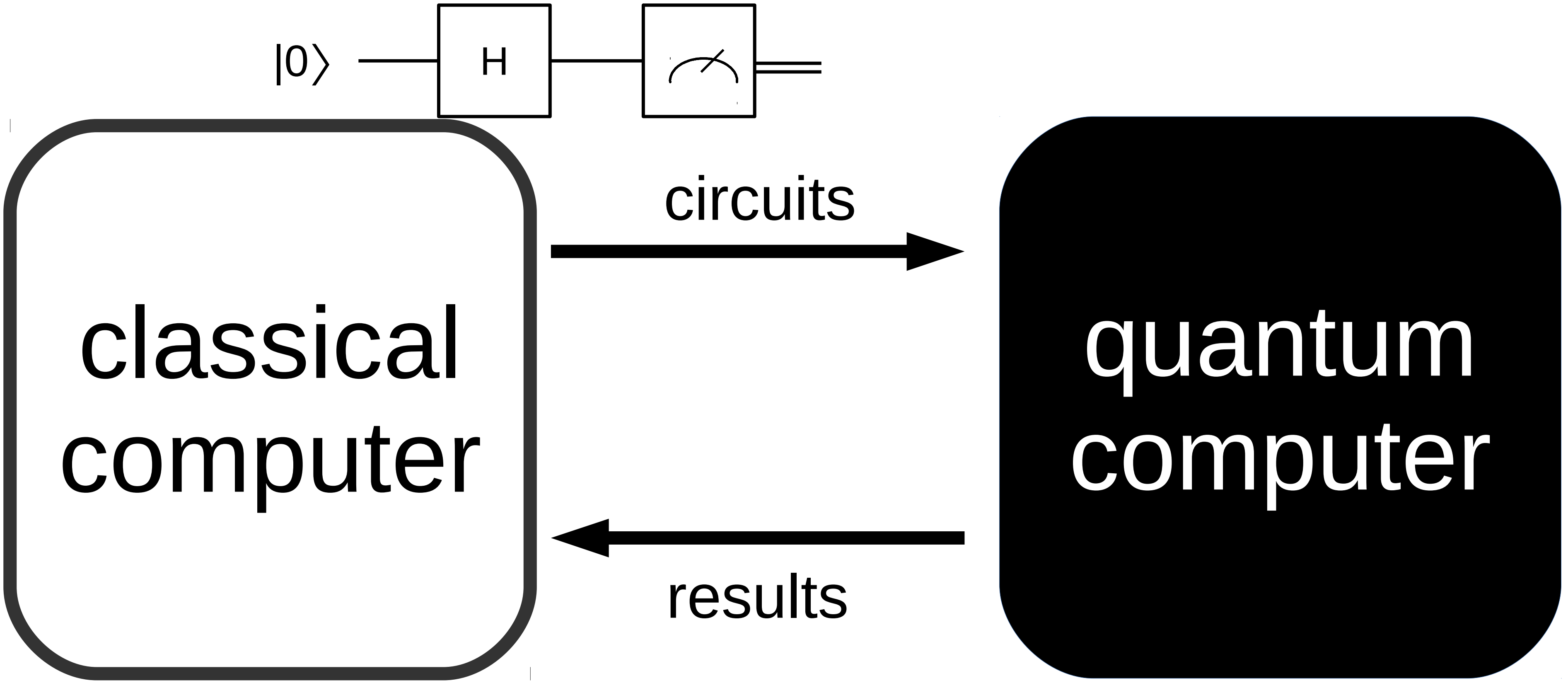
circuits

classical computer

quantum computer

results

Running Circuits



classical computer

circuits

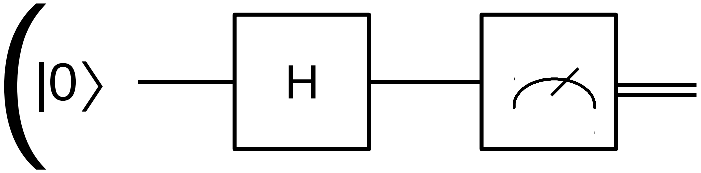
quantum computer

results

$$\left\{ \begin{array}{ll} \text{F} & \text{with probability } 1/2 \\ \text{T} & \text{with probability } 1/2 \end{array} \right\}$$

Running Circuits

$$\frac{\Gamma; \cdot \vdash C : \mathtt{bit}}{\Gamma \vdash \mathsf{run}\ C : \mathsf{Bool}}$$

$\mathsf{run}\Big($ 

$\Big) = \left\{ \begin{array}{ll} \mathrm{F} & \text{with probability } 1/2 \\ \mathrm{T} & \text{with probability } 1/2 \end{array} \right\}$

Dynamic Lifting

Dynamic Lifting

```
b ← gate meas q
```

Dynamic Lifting

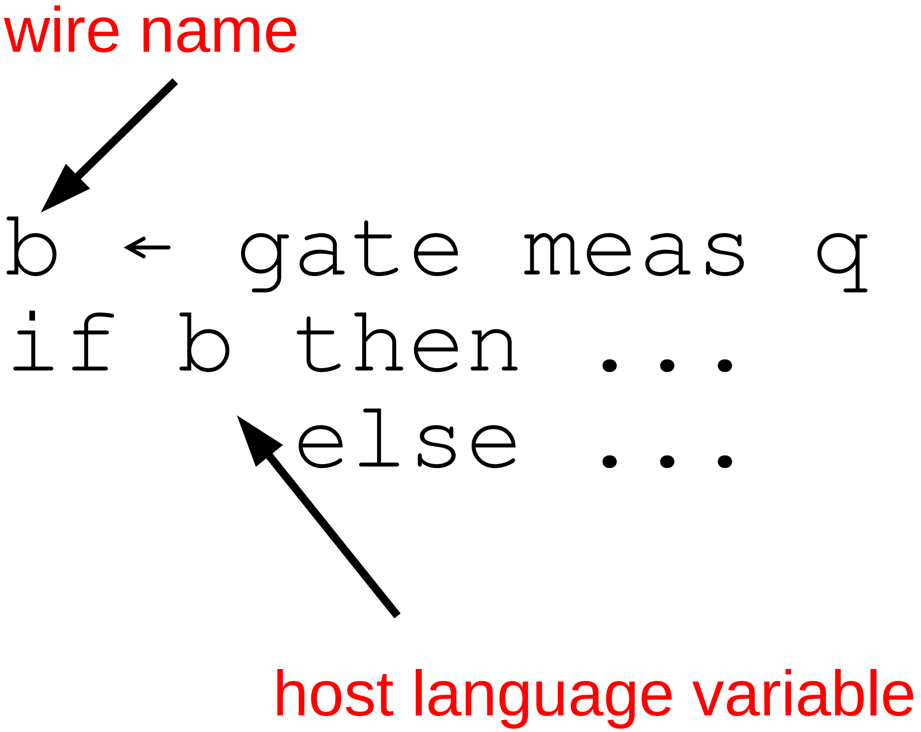
```
b ← gate meas q
if b then ...
     else ...
```

Dynamic Lifting

wire name

```
b ← gate meas q
if b then ...
     else ...
```

Dynamic Lifting



wire name

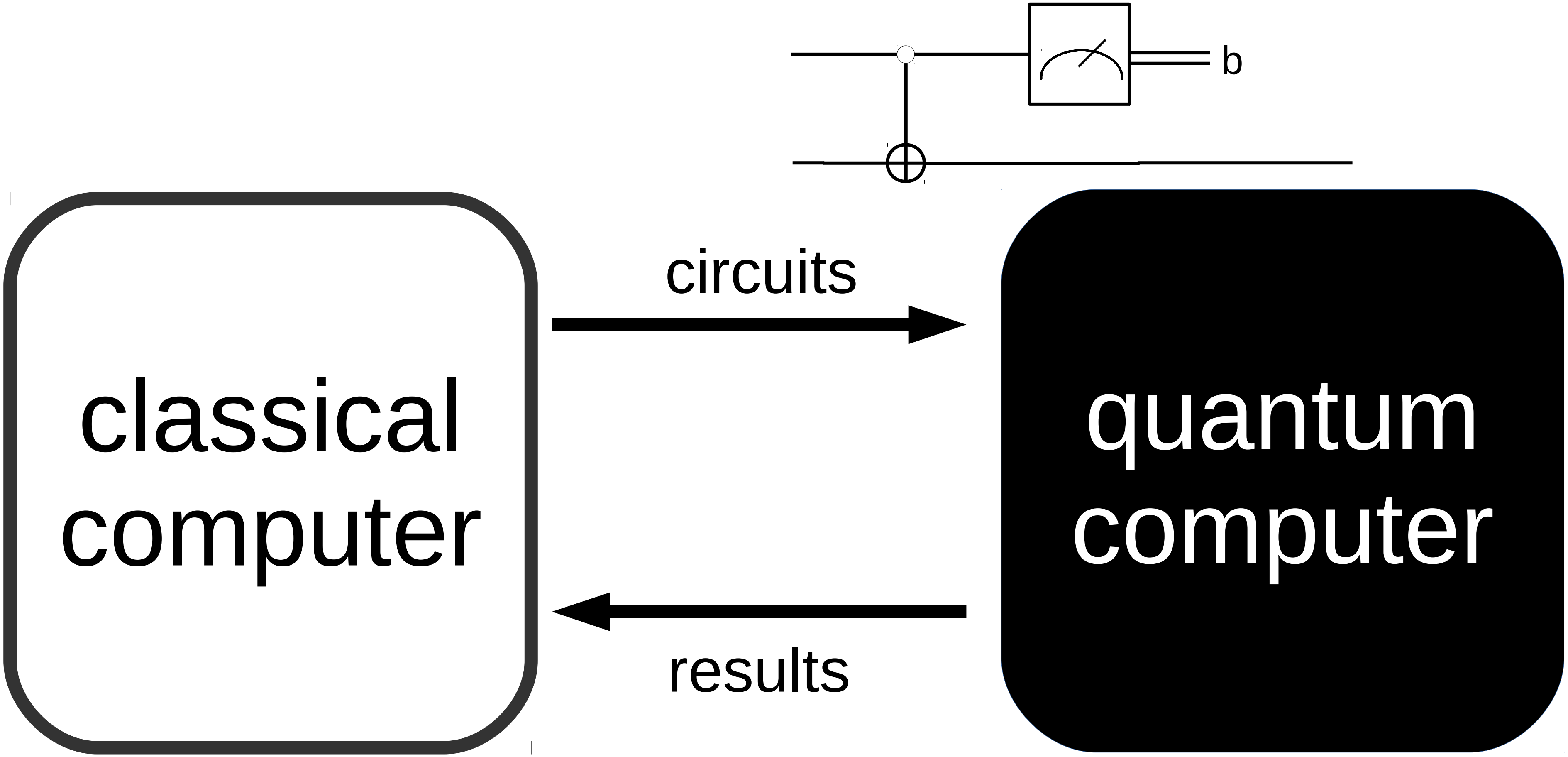
```
b ← gate meas q
if b then ...
     else ...
```

host language variable

Dynamic Lifting

```
b ← gate meas q
x ⇐ lift b;
unbox (if x then ...
            else ...) q'
```

Dynamic Lifting



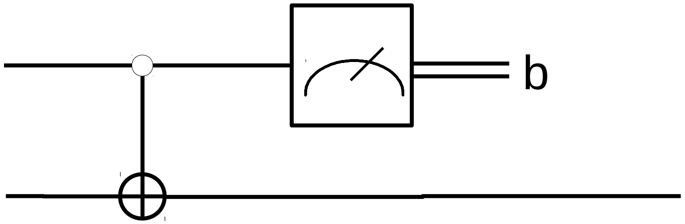
classical computer

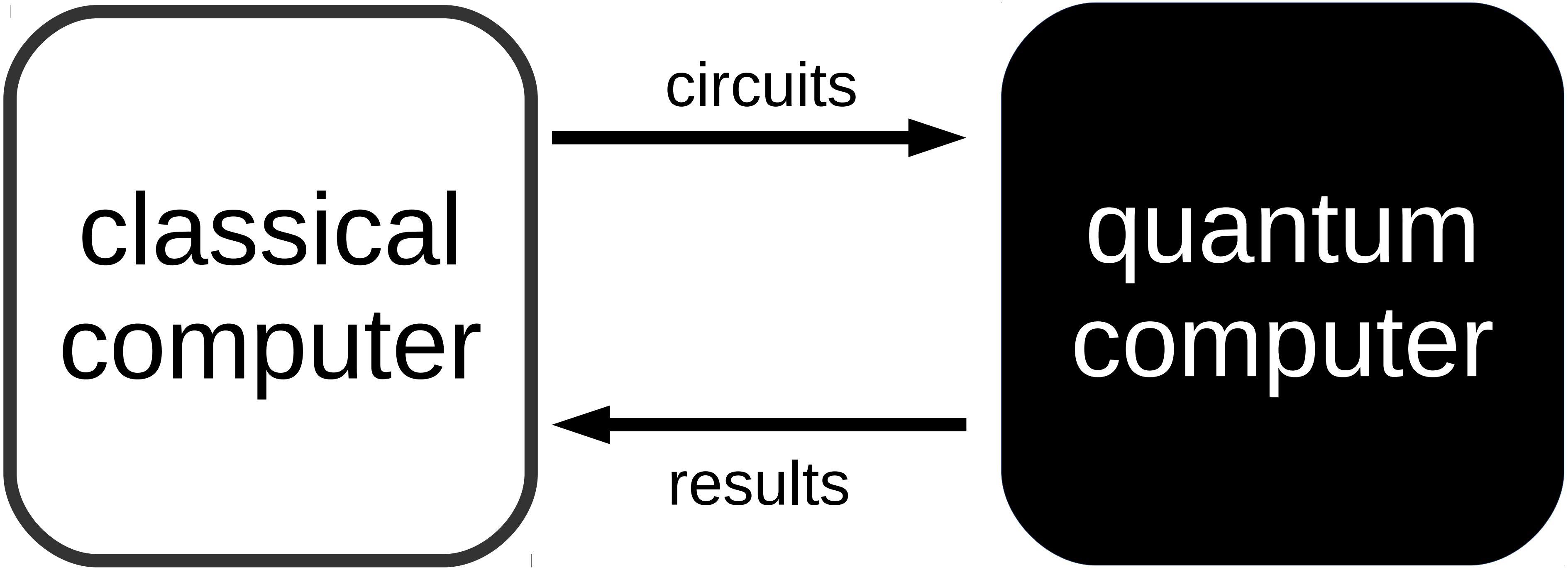
circuits

results

quantum computer

Dynamic Lifting





circuits

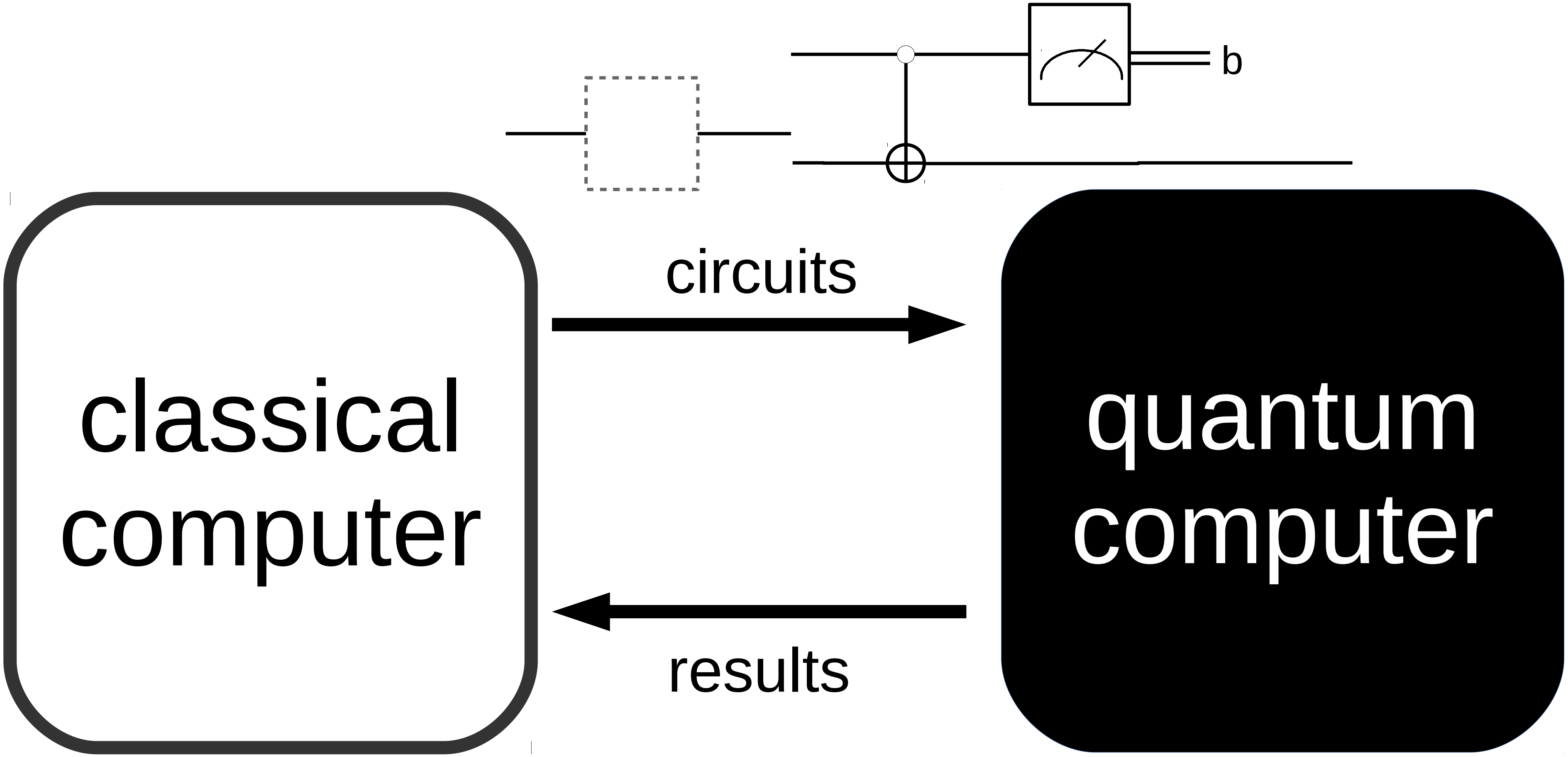
classical computer

quantum computer

results

$$\left\{ \begin{array}{ll} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{array} \right\}$$

Dynamic Lifting



circuits

classical computer

quantum computer

results

$$\left\{ \begin{array}{ll} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{array} \right\}$$

Summary

```
C ::= output p
    | p' ← gate g p; C
    | p  ← C; C'
    | unbox t p
    | x ⇐ lift p; C

t ::= …
    | box p ⇒ C
    | run C
```

Summary

```
C ::= output p
    | p' ← gate g p; C
    | p  ← C; C'
    | unbox t p
    | x ⇐ lift p; C

t ::= …
    | box p ⇒ C
    | run C
```

Summary

```
C ::= output p
    | p' ← gate g p; C
    | p  ← C; C'
    | unbox t p
    | x ⇐ lift p; C

t ::= …
    | box p ⇒ C
    | run C
```

Summary

```
C ::= output p
    | p' ← gate g p; C
    | p  ← C; C'
    | unbox t p
    | x ⇐ lift p; C

t ::= …
    | box p ⇒ C
    | run C
```

Summary

```
C ::= output p
    | p' ← gate g p; C
    | p  ← C; C'
    | unbox t p
    | x ⇐ lift p; C

t ::= ...
    | box p ⇒ C
    | run C
```

In the paper

In the paper

- operational semantics of circuits

In the paper

- **operational semantics of circuits**
  - proof of type safety
  - proof of strong normalization

In the paper

- **operational semantics of circuits**
  - proof of type safety
  - proof of strong normalization
- **denotational semantics of circuits**

Denotational Semantics

Selinger 2004

quantum computations

=

superoperators over density matrices

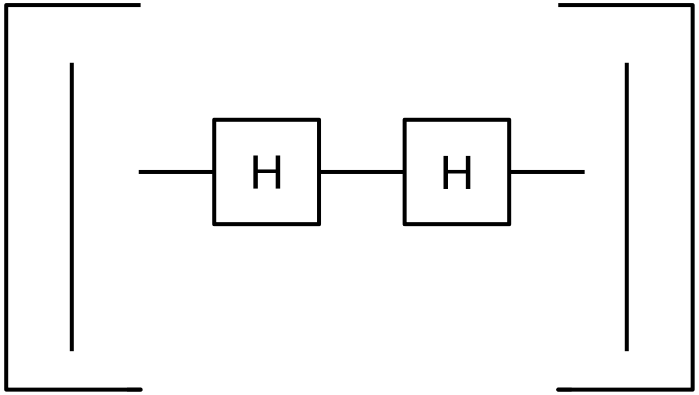
Denotational Semantics

Selinger 2004

quantum computations

=

superoperators over density matrices



$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

In the paper

In the paper

- **operational semantics of circuits**
  - proof of type safety
  - proof of strong normalization
- **denotational semantics of circuits**

In the paper

- **operational semantics of circuits**
  - proof of type safety
  - proof of strong normalization
- **denotational semantics of circuits**
  - proof of soundness

In the paper

- **operational semantics of circuits**
  - proof of type safety
  - proof of strong normalization
- **denotational semantics of circuits**
  - proof of soundness
- **dependently-typed circuits**

Dependent types

```
qubits (n : Nat) =
  case n of
  | 0   => 1
  | m+1 => qubit⊗(qubits m)
```

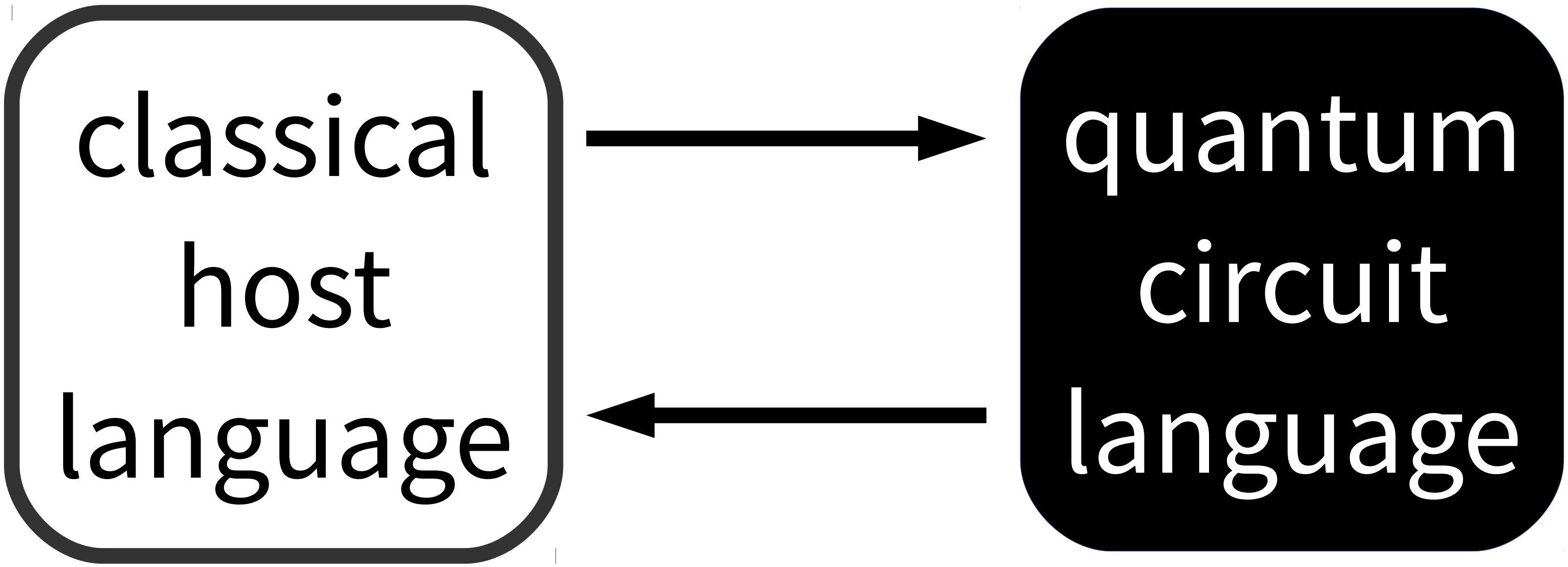
Dependent types

```
qubits (n : Nat) =
  case n of
  | 0    => 1
  | m+1  => qubit⊗(qubits m)


fourier : ∀(n:nat).
     Circ(qubits n, qubits n)
```

In the paper

- **operational semantics of circuits**
  - proof of type safety
  - proof of strong normalization
- **denotational semantics of circuits**
  - proof of soundness
- **dependently-typed circuits**
- **...and more!**

Thank you! Questions?



classical host language

quantum circuit language

References

- C. Badescu and P. Panangaden. Quantum alternation: prospects and problems. QPL 2015.
- M. Ying. Quantum recursion and second quantisation. Available on arXiv, 2014.
- E. H. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, 1996.
- A. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. PLDI 2013.
- D. Wecker and K. M. Svore. LIQUi|⟩: A software design architecture and domain-specific language for quantum computing. URL https://www.microsoft.com/en-us/research/publication/liqui-a-software-design-architecture-and-domain-specific-language-for-quantum-computing/. 2014.
- S. Bettelli, T. Calarco, and L. Serafini. Toward an Architecture for Quantum Programming. The European Physical Journal 2003.
- N. Benton. A mixed linear and non-linear logic. CSL 1995.
- M. A. Nielsen and I. L. Chuang. Quantum computation and quantum information. 2010

Normal Forms

```
C ::= output p
   | p' ← gate g p; C
   | p   ← C; C'
   | unbox t p
   | x ⇐ lift p; C
```

Normal Forms

```
C ::= output p
    | p' ← gate g p; C
    | p  ← C; C'
    | unbox t p
    | x ⇐ lift p; C
```

```
N ::= output p
    | p' ← gate g p; N
    | x ⇐ lift p; C
```

Types

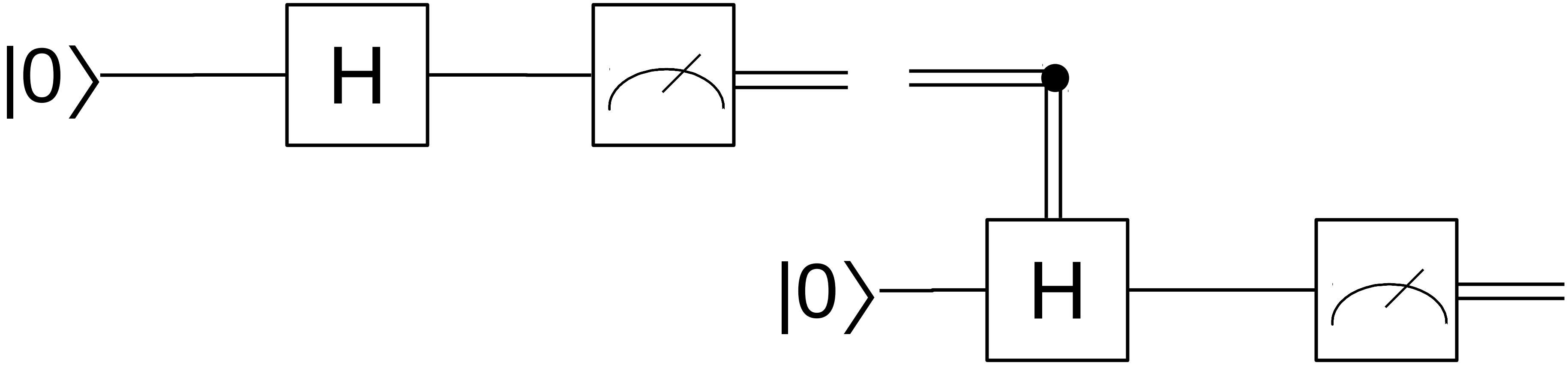
$$t ::= \cdots \mid \mathsf{Circ}(W_1, W_2)$$

Types

$$W ::= \texttt{qubit} \mid \texttt{bit} \mid 1 \mid W_1 \otimes W_2$$

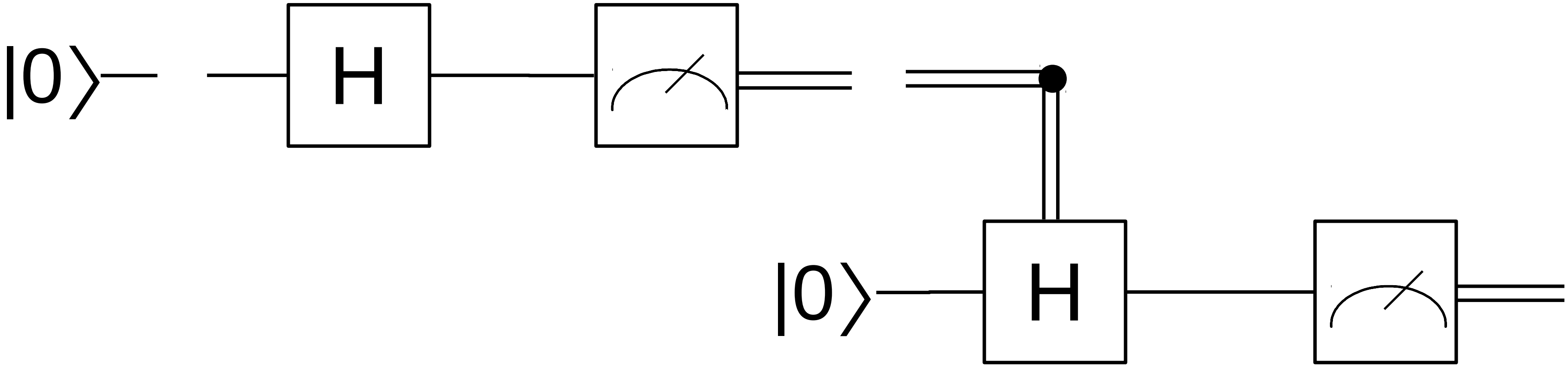
$$t ::= \cdots \mid \mathsf{Circ}(W_1, W_2)$$

Composition is associative



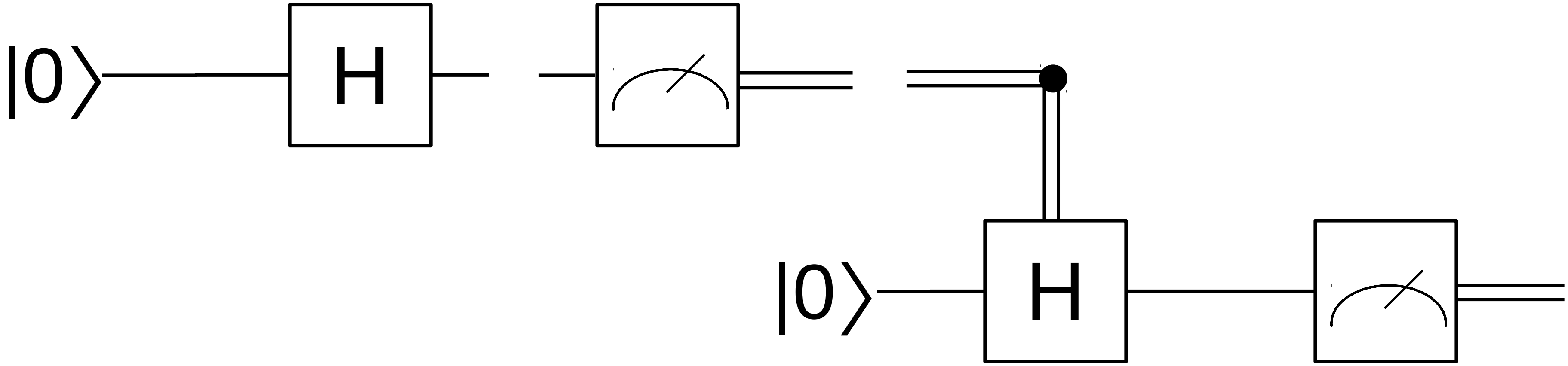
```
b ← (q  ← gate new0 ();
     q' ← gate H    q ;
     b' ← gate meas q';
     output b');
r ← new0 ();
...
```

Composition is associative



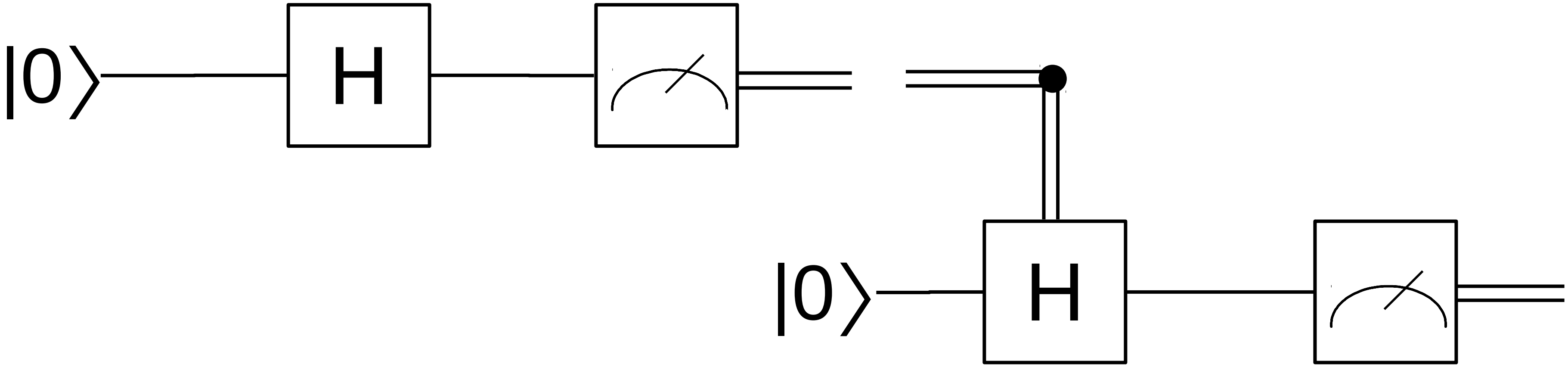
```
q ← gate new0 ();
b ← (q' ← gate H     q ;
     b' ← gate meas q';
     output b');
r ← new0 ();
...
```

Composition is associative



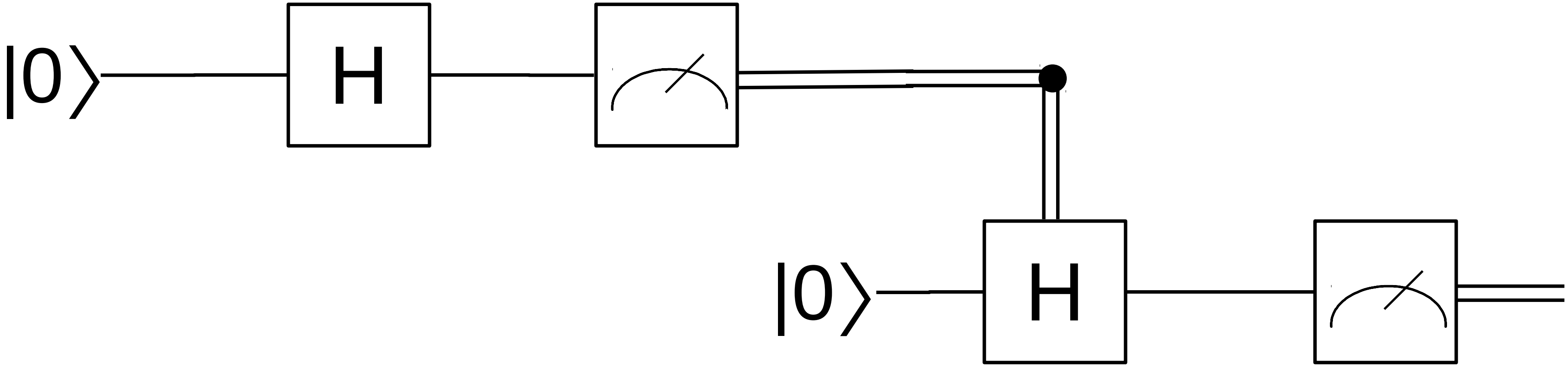
```
q ← gate new0 ();
q'← gate H     q ;
b ← (b' ← gate meas q';
     output b');
r ← new0 ();
...
```

Composition is associative



```
q ← gate new0 ();
q'← gate H    q ;
b'← gate meas q';
b ← (output b');
r ← new0 ();
...
```

Composition is associative



```
q ← gate new0 ();
q'← gate H    q ;
b'← gate meas q';
r ← new0 ();
...
```