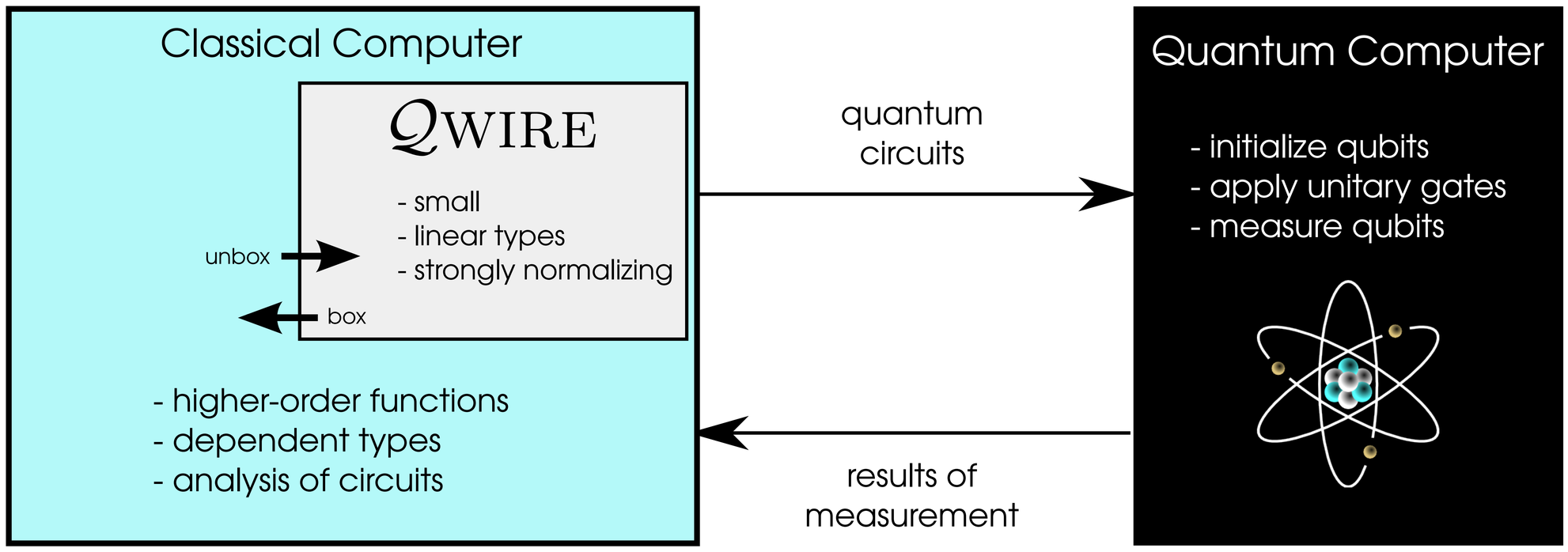
QWIRE: A QRAM-Inspired Quantum Circuit Language

Jennifer Paykin, Robert Rand, Steve Zdancewic
University of Pennsylvania



Host Language

$$
\begin{array}{rcl}
t & ::= & \lambda x.t \mid t_1 t_2 \\
& \mid & \cdots \\
& \mid & \text{box } p \Rightarrow C \\
A & ::= & A_1 \to A_2 \\
& \mid & \cdots \\
& \mid & \mathsf{CIRC}(W_1, W_2)
\end{array}
$$

host types = arbitrary use

QWIRE

$$
\begin{array}{rcll}
C & ::= & \text{output } p & \mid \text{gate } u \text{ on } p; C \\
& \mid & p \leftarrow \text{new } t; C & \mid x \leftarrow \text{measure } p; C \\
& \mid & x \leftarrow C; C' & \mid \text{unbox } t\ p
\end{array}
$$

$$
p ::= () \mid w \mid (p_1, p_2)
$$

$$
\begin{array}{rcl}
W & ::= & 1 \\
& \mid & \text{qubit} \\
& \mid & W_1 \otimes W_2
\end{array}
$$

wire types = linear use

Normalization

$$\text{unbox } (\text{box } p \Rightarrow C)\ p' \longrightarrow C\{p'/p\}$$

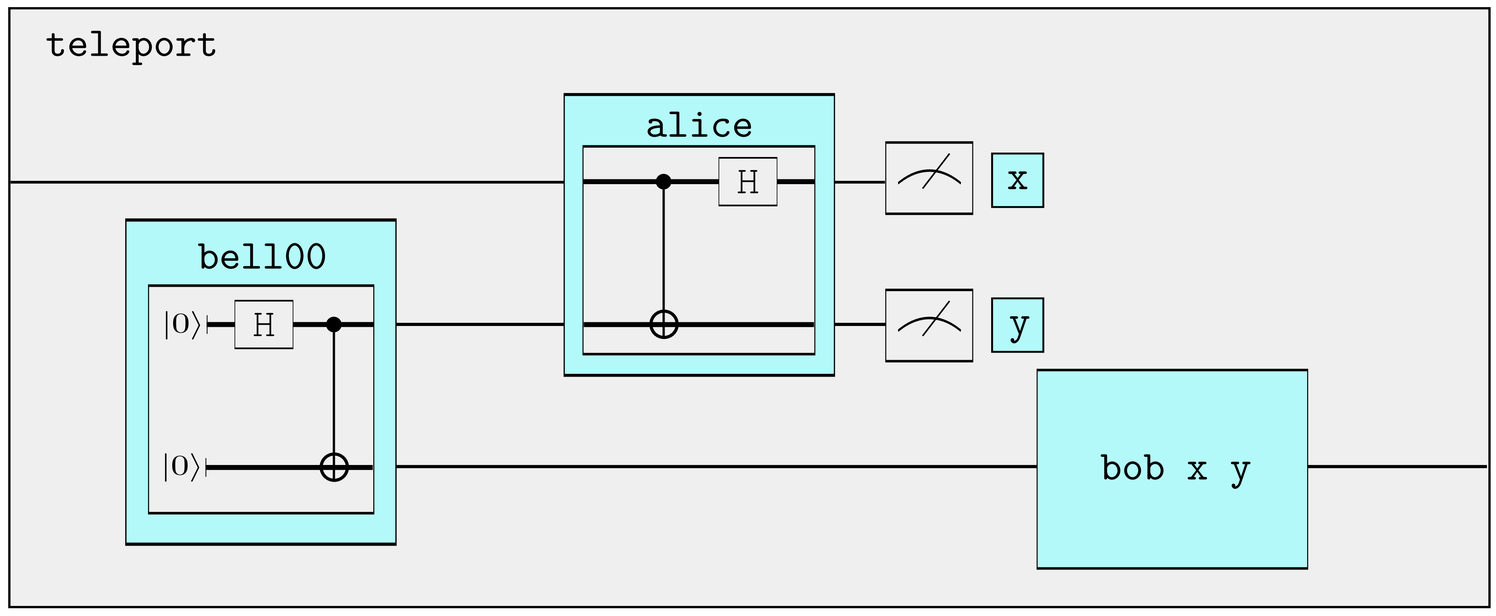
$$p \leftarrow \text{output } p'; C \longrightarrow C\{p'/p\}$$

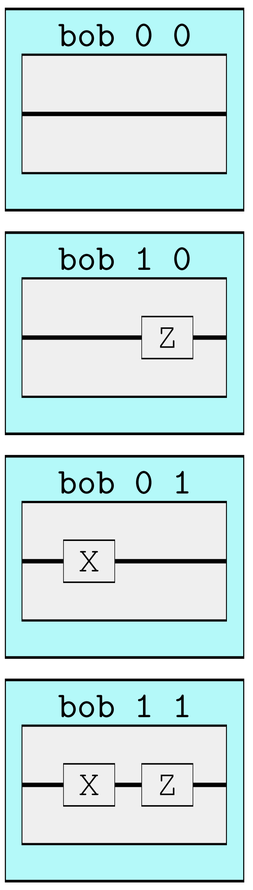
$$p \leftarrow (\text{gate } u \text{ on } p'; C'); C \longrightarrow \text{gate } u \text{ on } p'; p \leftarrow C'; C$$

$$
\begin{array}{rcl}
N & ::= & \text{output } p \\
& \mid & \text{gate } u \text{ on } p; N \\
& \mid & p \leftarrow \text{new } v; N \\
& \mid & x \leftarrow \text{measure } p; C
\end{array}
$$

Quantum Teleportation

```
teleport :: CIRC(qubit,qubit)
teleport = box q =>
  (a,b) <- unbox bell00 ();
  (q,a) <- unbox alice (q,a);
  (x,y) <- measure (q,a);
  unbox (bob x y) b
```





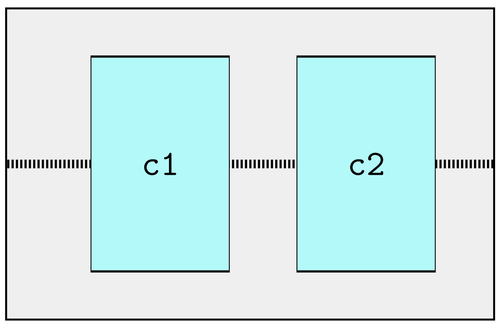

```
bell00 :: CIRC(1,qubit⊗qubit)
bell00 = box () =>
  (a,b) <- new (0,0);
  gate H on a;
  gate CNOT on (a,b);
  output (a,b)
```

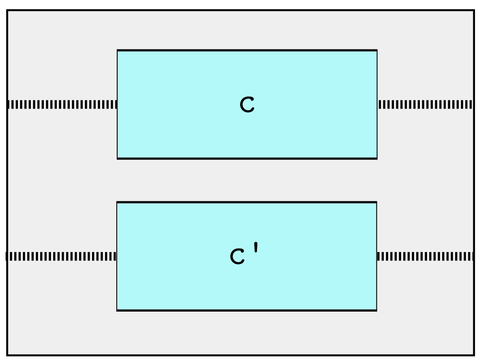
```
alice :: CIRC(qubit⊗qubit,qubit⊗ qubit)
alice = box (q,a) =>
  gate CNOT on (q,a);
  gate H on q;
  output (q,a)
```

```
bob :: Bit -> Bit -> CIRC(qubit,qubit)
bob x y = box b =>
  b <- unbox (if y then X_CIRC else ID) b
  b <- unbox (if x then Z_CIRC else ID) b
  output b
```

Meta-Programming with Circuits

```
inSequence :: CIRC(W1,W2) -> CIRC(W2,W3)
           -> CIRC(W1,W3)
inSequence c1 c2 = box w1 =>
  w2 <- unbox c1 w1;
  w3 <- unbox c2 w2;
  output w3
```






```
inParallel :: CIRC(W1,W2) -> CIRC(W1',W2')
           -> CIRC(W1⊗W1',W2⊗W2')
inParallel c c' = box (w1,w1') =>
  w2  <- unbox c  w1;
  w2' <- unbox c' w1';
  output (w2,w2')
```

Host Language Dependent Types

```
fourier :: forall (n :: Nat+). CIRC(Qubits n, Qubits n)
fourier 1      = box w => gate H on w; output w
fourier (n'+1) = box (q,w) =>
    w     <- unbox fourier n' w;
    unbox rotations (n'+1) n' (q,w)
```