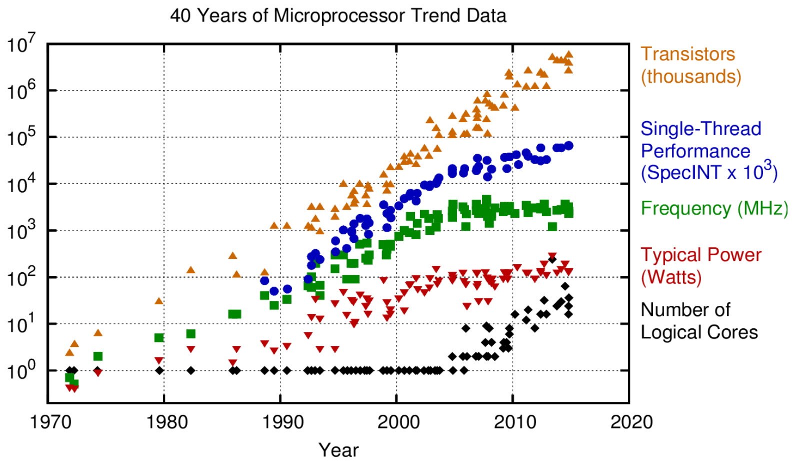
From Functional Programs to Pipelined Dataflow Networks

Richard Townsend
Martha A. Kim   Stephen A. Edwards

Columbia University

IBM PL Day, December 5, 2016

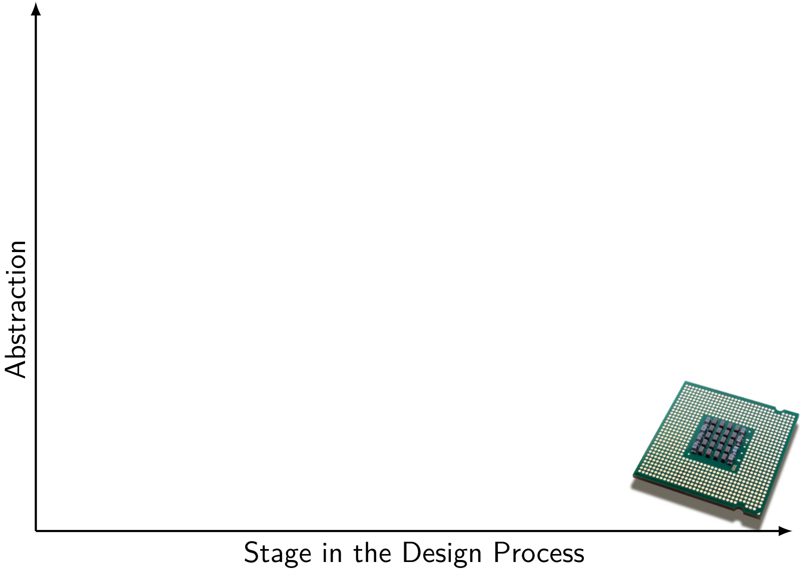
The Future of Hardware



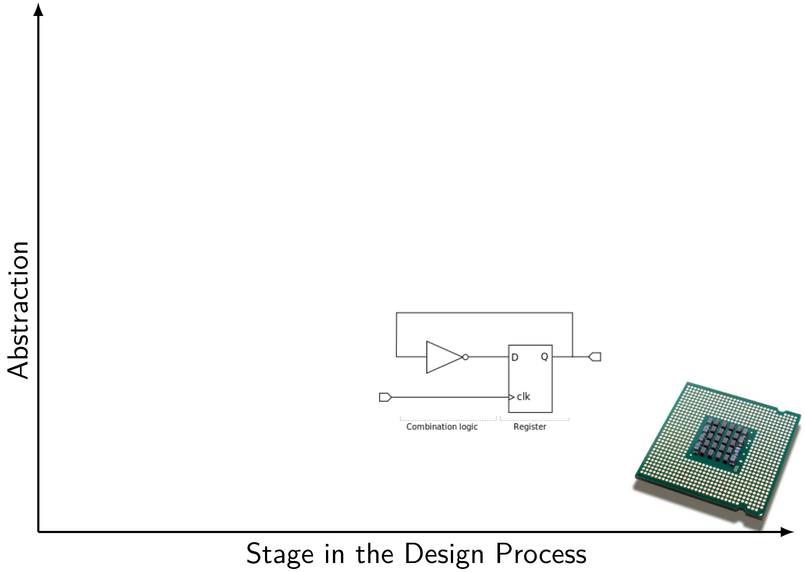
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Rupp, Karl. 40 Years Of Microprocessor Trend Data. 2015. Web. 28 Nov. 2016.

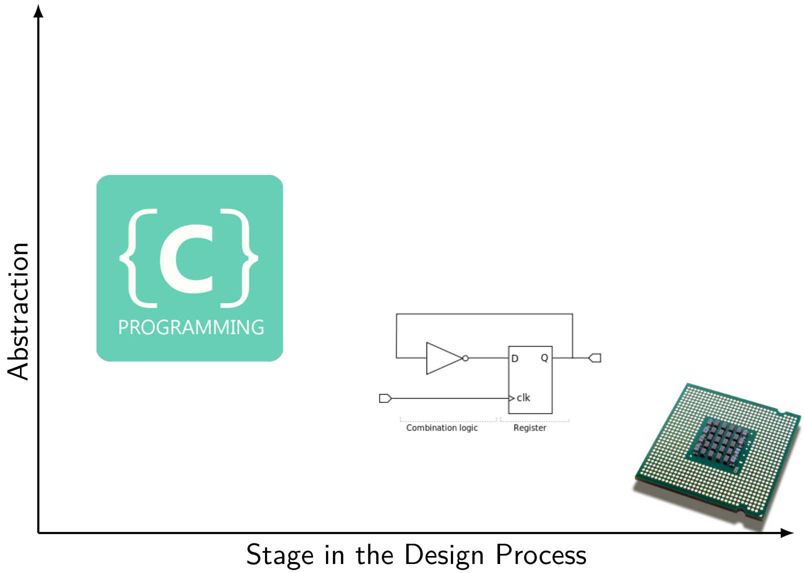
Designing Specialized Hardware



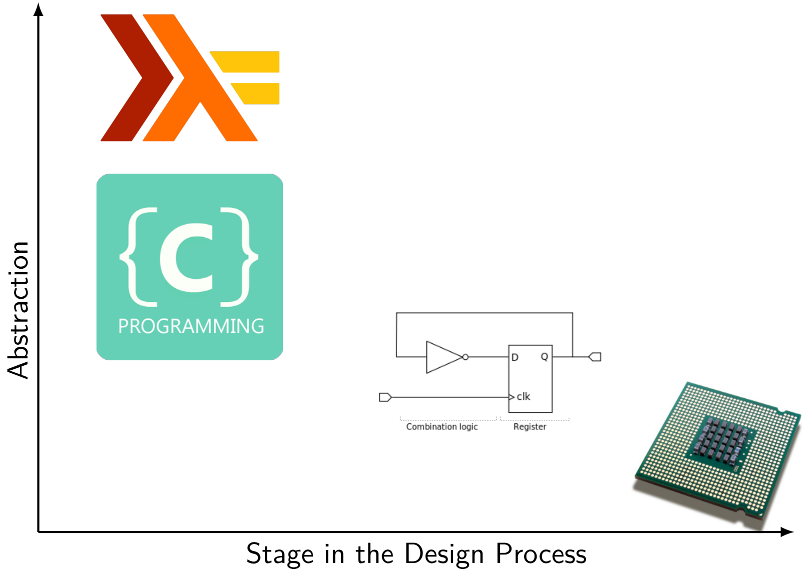
Designing Specialized Hardware



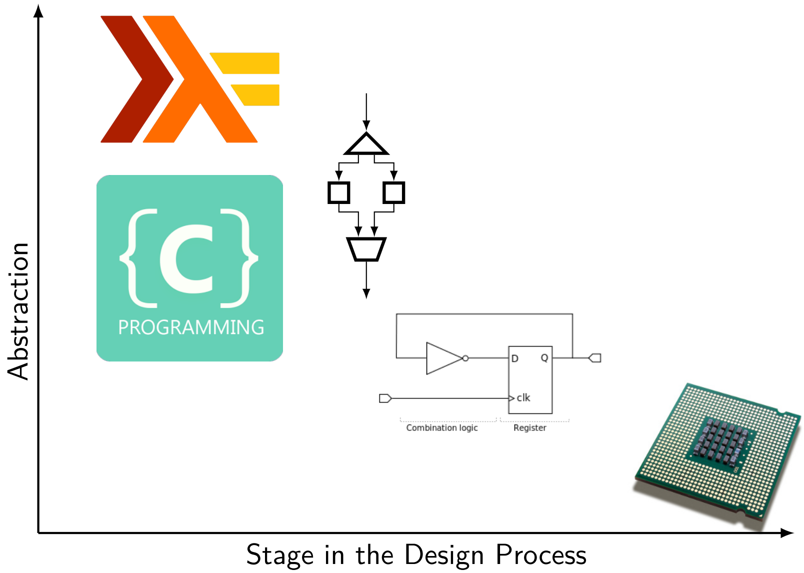
Designing Specialized Hardware



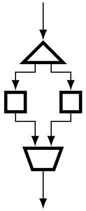
Designing Specialized Hardware



Designing Specialized Hardware



Overview



Floh IR

Dataflow Networks

Hardware Simulation

McGreggor, Duncan. Prefix Operators in Haskell. 2014. Web. 30 Nov. 2016.
Kuper, Jan. CλaSH: From Haskell to Hardware. 2015. Web. 30 Nov. 2016.

Floh (Functional Language on Hardware)

```
data List = Nil | Cons Int List




recMath :: List → Int → Int
recMath l x =
  case l of
    Nil         → add x 1
    Cons y xs → recMath xs (mul x y)
```

Floh (Functional Language on Hardware)

```
data List = Nil | Cons Int List



recMath :: List → Int → Int
recMath l x =
  case l of
    Nil       → add x 1
    Cons y xs → recMath xs (mul x y)
```

- True recursion → stack

Floh (Functional Language on Hardware)

```
data List = Nil Go | Cons Int List
data Go   = Go


recMath :: List → Int → Go → Int
recMath l x g =
  case l of
    Nil  _     → add x (1 g)
    Cons y xs  → recMath xs (mul x y) g
```

- True recursion → stack
- Trigger constants with **Go**

Floh (Functional Language on Hardware)

```
data List   = Nil Go | Cons Int ListPtr
data Go     = Go
data ListPtr = ListPtr Int

recMath :: ListPtr → Int → Go → Int
recMath lp x g =
  case readList lp of
    Nil  _     → add x (1 g)
    Cons y xs → recMath xs (mul x y) g
```

- True recursion → stack
- Trigger constants with **Go**
- Recursive types → explicit memory operations

Floh (Functional Language on Hardware)

```
data List    = Nil Go | Cons Int ListPtr
data Go      = Go
data ListPtr = ListPtr Int

recMath :: ListPtr → Int → Go → Int
recMath lp x g =
  case readList lp of
     Nil _      → add x (1 g)
     Cons y xs  → recMath xs (mul x y) g
```

- True recursion → stack
- Trigger constants with **Go**
- Recursive types → explicit memory operations

Floh (Functional Language on Hardware)

```
data List  = Nil Go | Cons Int  ListPtr
data Go   = Go
data ListPtr  = ListPtr  Int

recMath ::  ListPtr  → Int  → Go → Int
recMath lp x  g =
  case  readList  lp  of
     Nil   _     → add x (1  g)
     Cons y xs  → recMath xs (mul x y)  g
```

- True recursion → stack
- Trigger constants with **Go**
- Recursive types → explicit memory operations

Strictness Policies

Floh (Functional Language on Hardware)

```
data List  = Nil Go | Cons Int  ListPtr
data Go   = Go
data ListPtr = ListPtr Int

recMath :: ListPtr → Int → Go → Int
recMath lp x g =
  case readList lp of
    Nil  _     → add x (1 g)
    Cons y xs → recMath xs (mul x y) g
```

- True recursion → stack
- Trigger constants with **Go**
- Recursive types → explicit memory operations

Strictness Policies

- Data Constructors: **strict** – evaluate *all* arguments

Floh (Functional Language on Hardware)

```
data List    = Nil Go | Cons Int ListPtr
data Go      = Go
data ListPtr = ListPtr Int

recMath :: ListPtr → Int → Go → Int
recMath lp x g =
  case readList lp of
    Nil _      → add x (1 g)
    Cons y xs  → recMath xs (mul x y) g
```

- True recursion → stack
- Trigger constants with **Go**
- Recursive types → explicit memory operations

Strictness Policies

- Data Constructors: **strict** – evaluate *all* arguments
- Functions: **non-strict** – evaluate *first* argument

Floh (Functional Language on Hardware)

```
data List  = Nil Go | Cons Int  ListPtr
data Go   = Go
data ListPtr  = ListPtr  Int

recMath ::  ListPtr  → Int  → Go → Int
recMath lp x  g =
  case  readList  lp  of
    Nil   _     → add x (1  g)
    Cons y xs  → recMath xs (mul x y)  g
```

- True recursion → stack
- Trigger constants with **Go**
- Recursive types → explicit memory operations

Strictness Policies

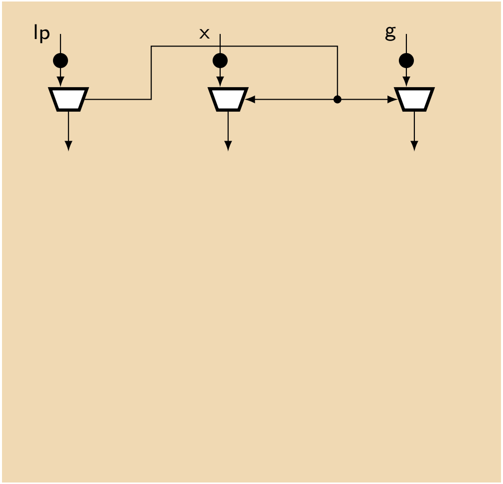
- Data Constructors: **strict** – evaluate *all* arguments
- Functions: **non-strict** – evaluate *first* argument
- Enables pipeline parallelism!

From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _    → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```

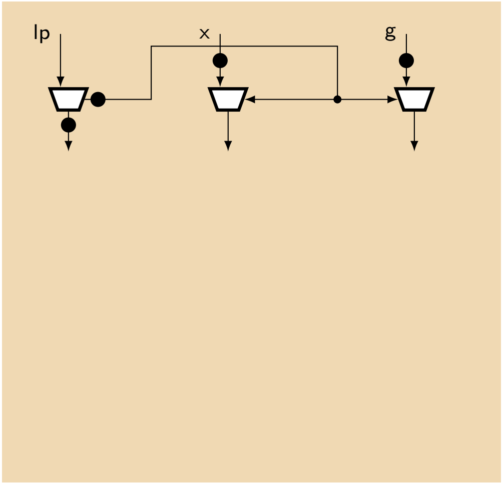
From Floh to Dataflow

```
recMath lp x g =
 case readList  lp  of
   Nil   _     → add x (1  g)
   Cons y xs →
     recMath xs (mul x y)  g
```



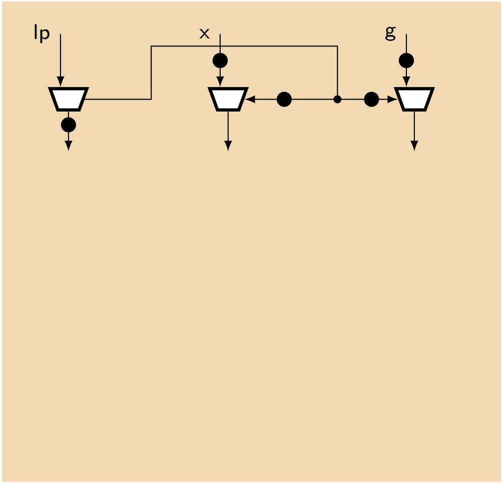
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
   Nil   _      → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



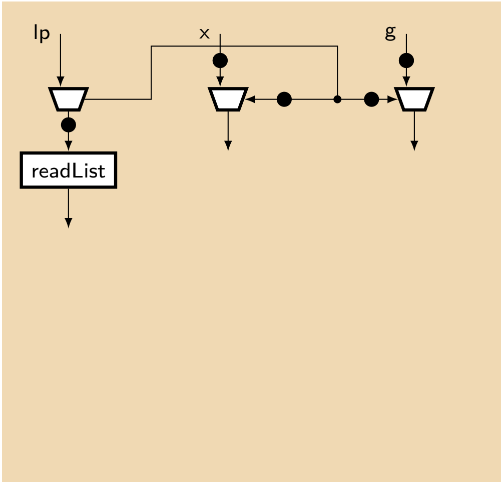
From Floh to Dataflow

```
recMath lp x g =
 case readList  lp  of
   Nil    _      → add x (1  g)
   Cons y xs  →
      recMath xs (mul x y)  g
```



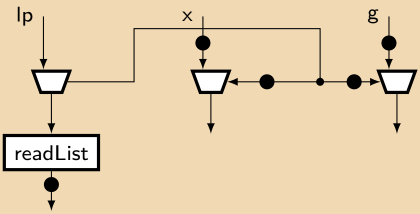
From Floh to Dataflow

```
recMath lp x g =
 case readList  lp of
   Nil   _      → add x (1  g)
   Cons y xs →
     recMath xs (mul x y)  g
```



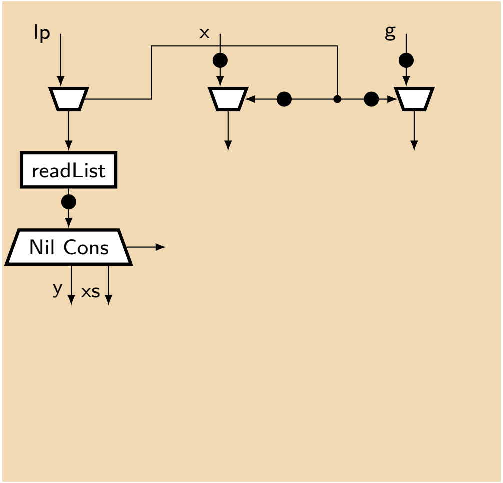
From Floh to Dataflow

```
recMath lp x g =
 case readList  lp of
   Nil   _      → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



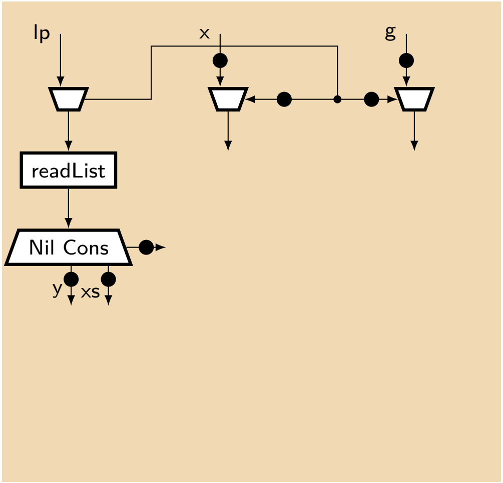
From Floh to Dataflow

```
recMath lp x g =
 case readList  lp of
   Nil   _      → add x (1 g)
   Cons y xs →
       recMath xs (mul x y) g
```



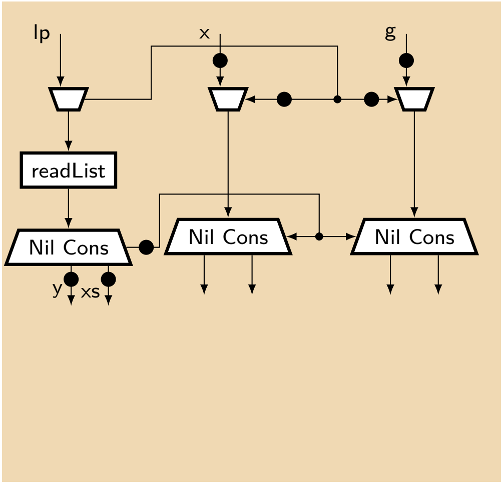
From Floh to Dataflow

```
recMath lp x g =
 case readList  lp  of
   Nil    _      → add x (1  g)
   Cons y xs →
       recMath xs (mul x y)  g
```



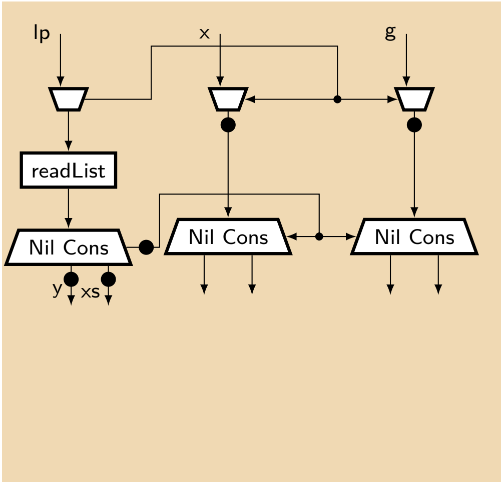
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil _        → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



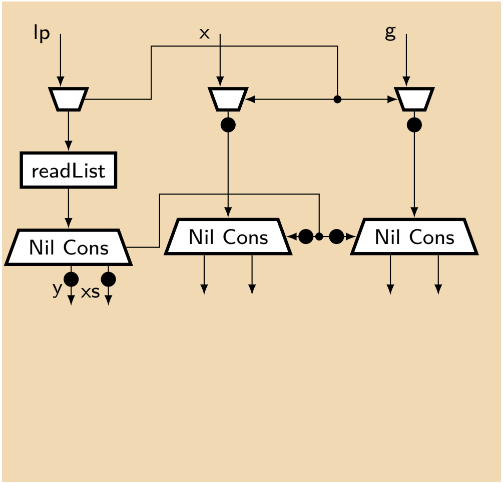
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _    → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



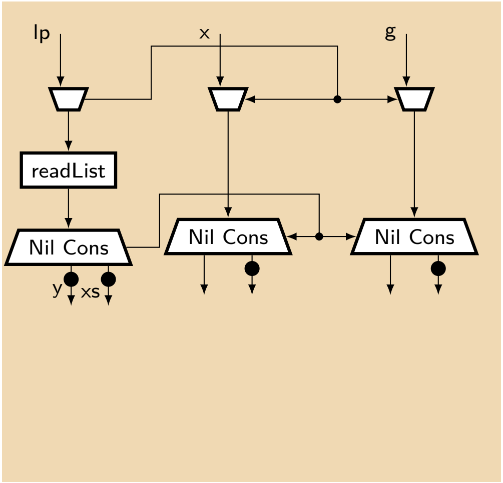
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _     → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



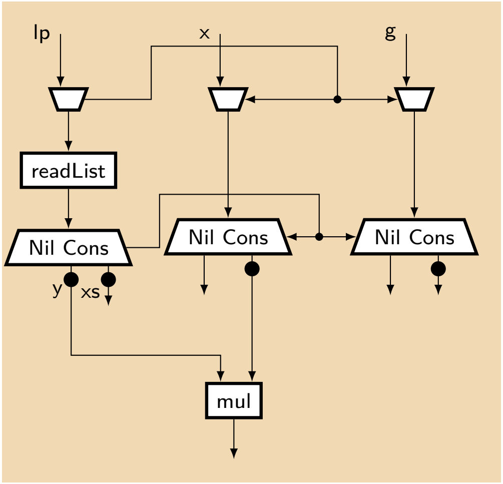
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
   Nil  _     → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



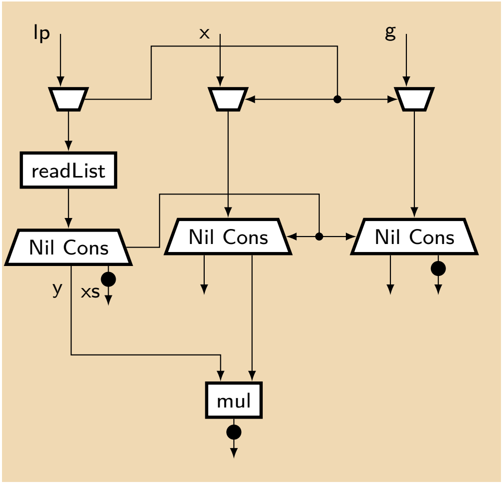
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
   Nil  _    → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



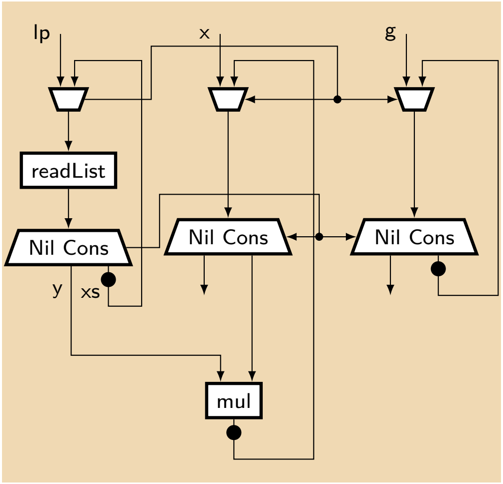
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
   Nil _     → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



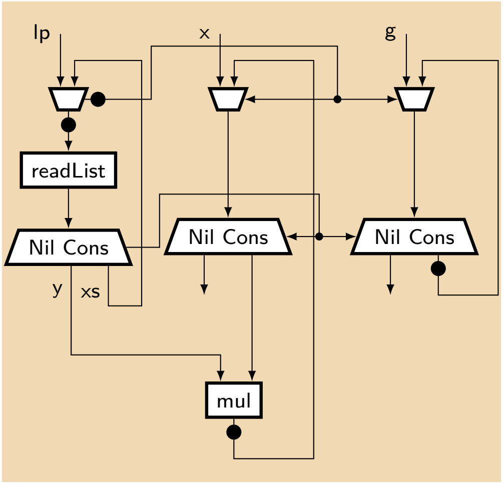
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil   _    → add x (1 g)
  Cons y xs →
      recMath xs (mul x y) g
```



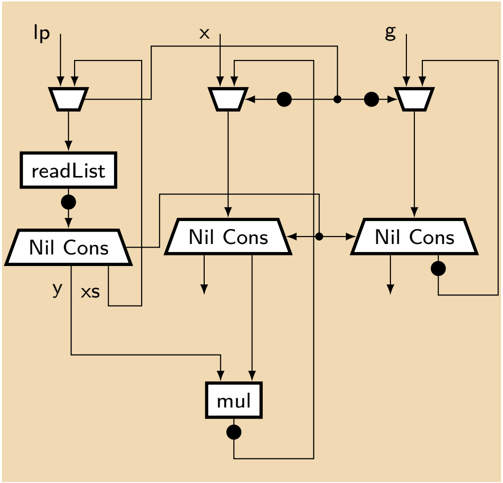
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _      → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



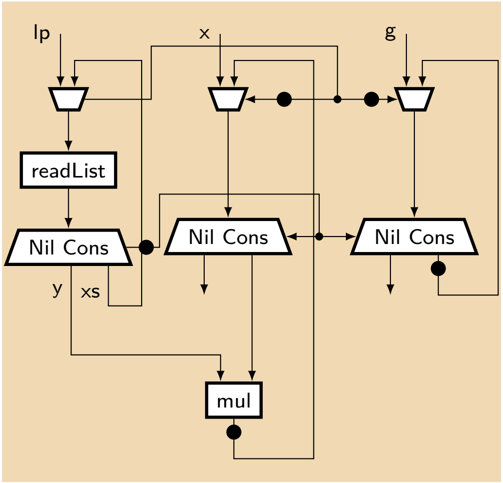
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _     → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



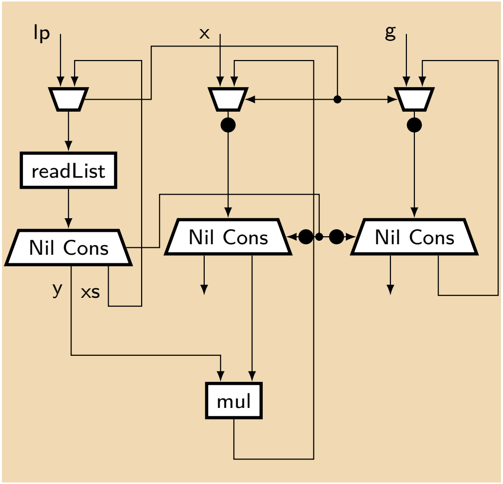
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _    → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



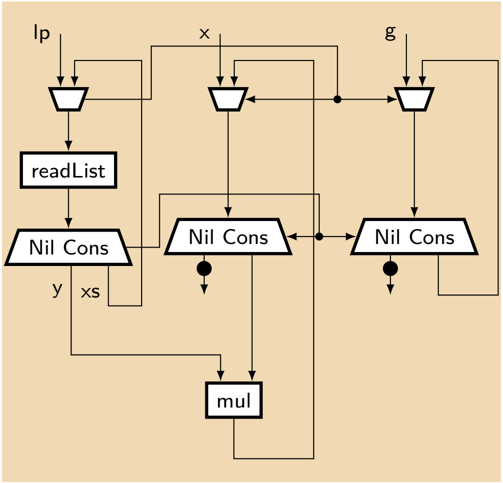
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
   Nil  _      → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



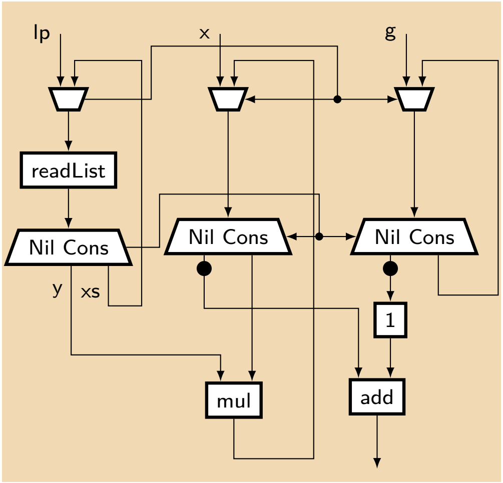
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil  _    → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



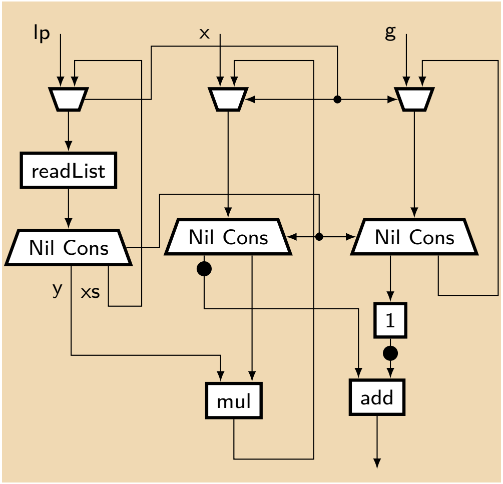
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil _      → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```



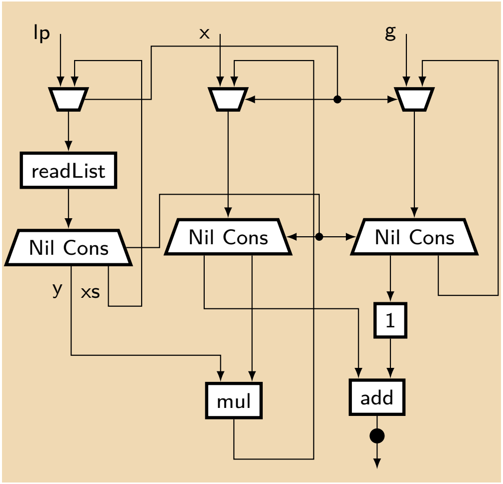
From Floh to Dataflow

```
recMath lp x g =
 case readList lp of
  Nil _     → add x (1 g)
  Cons y xs →
    recMath xs (mul x y) g
```

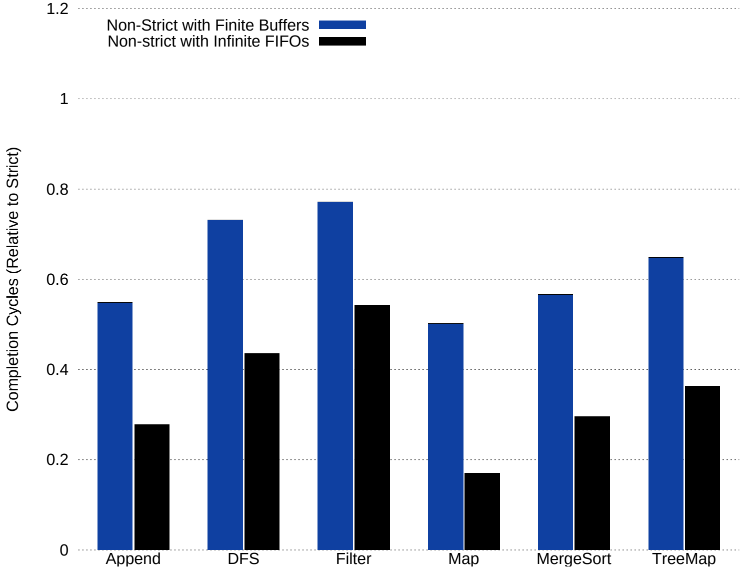


From Floh to Dataflow

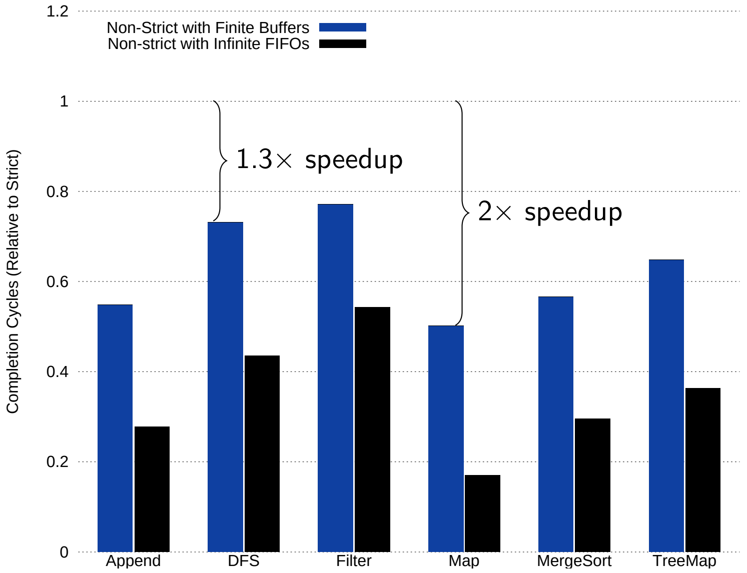
```
recMath lp x g =
 case readList lp of
   Nil   _     → add x (1 g)
   Cons y xs →
     recMath xs (mul x y) g
```



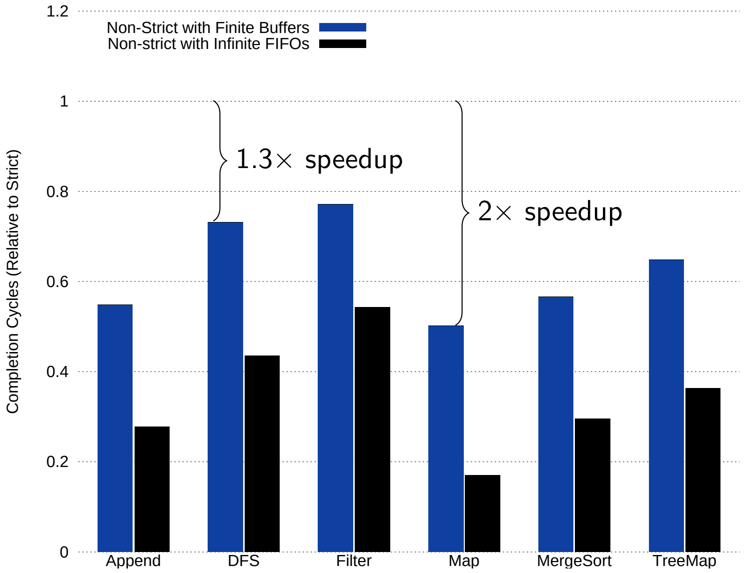
Non-strictness Exploits Pipeline Parallelism



Non-strictness Exploits Pipeline Parallelism



Non-strictness Exploits Pipeline Parallelism


Non-Strict with Finite Buffers
Non-strict with Infinite FIFOs
Completion Cycles (Relative to Strict)
1.3× speedup
2× speedup
Append
DFS
Filter
Map
MergeSort
TreeMap