Compiling with Effects and Handlers

A New Compiler Architecture

Jaro Reinders

TU Delft

Contents

1. Why write compilers differently?
2. What are effects and handlers?
3. How to compile with effects and handlers?

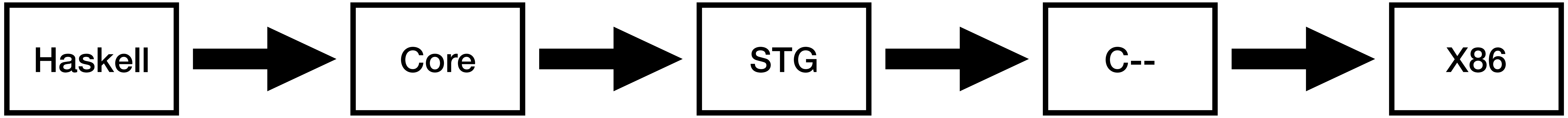
1. Why write compilers differently?
2. What are effects and handlers?
3. How to compile with effects and handlers?

Goal: Modularity

- Most existing compilers (such as GHC) are monolithic
- Modularity could improve:
  - Maintenance
  - Extensibility
  - Understanding (reasoning)

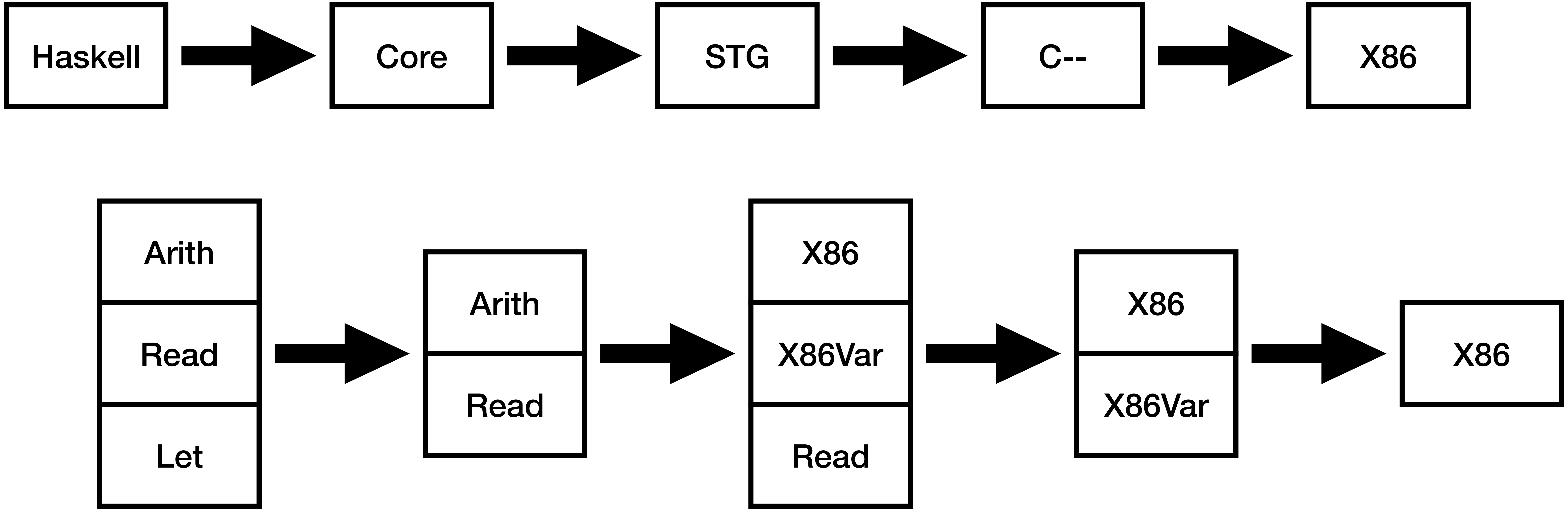
A New Compiler Architecture

Inspired by nanopass compilers



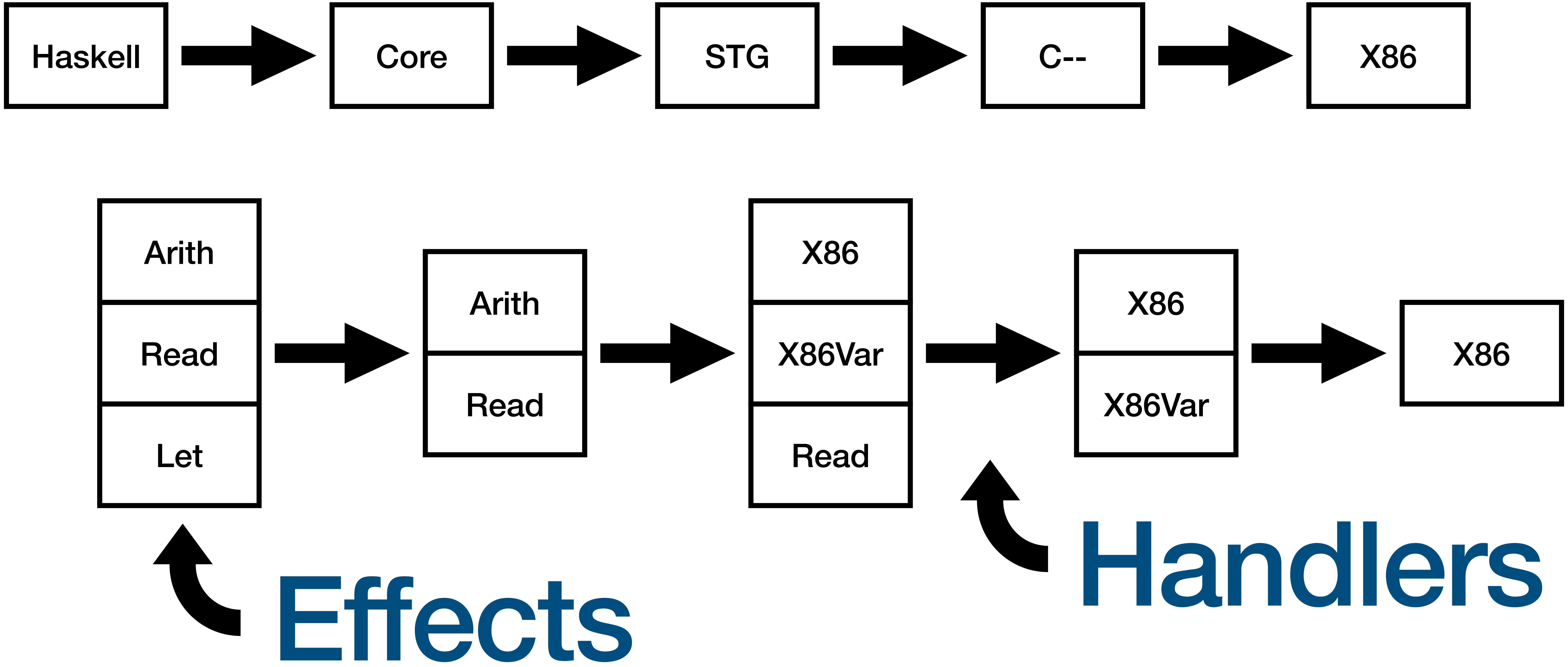
A New Compiler Architecture

Inspired by nanopass compilers



A New Compiler Architecture

Inspired by nanopass compilers



1. Why write compilers differently?
2. What are effects and handlers?
3. How to compile with effects and handlers?

Effects and Handlers

- Historically,
    - an improvement upon Moggi’s work on monads and monad transformers
    - avoiding the need for lifting functions
- Practically,
    - a set of **abstract operations** forming an interface
    - with the superpower to **manipulate control-flow**
    - and a mechanism to give **concrete meaning** to these abstract operations

Examples

Maybe

```haskell
data Maybe a
  = Nothing | Just a


instance Monad Maybe where
  Just x >>= k = k x
  Nothing >>= k = Nothing
  return x = Just x
```

Examples

Maybe

```
data Maybe a
  = Nothing | Just a
```

```
instance Monad Maybe where
  Just x >>= k = k x
  Nothing >>= k = Nothing
  return x = Just x
```

```
effect Abort where
  abort : m a
```

```
handle
  abort k -> Nothing
  return x -> Just x
```

Examples

State

```
data State s a
  = MkState (s → a × s)


instance Monad (State s) where
  MkState p >>= k = MkState \s →
    let (x, s’) = p s in
    let MkState q = k x in
    q s’
  return x = MkState \s → (x, s)
```

Examples

State

```
data State s a
  = MkState (s → a × s)

instance Monad (State s) where
  MkState p >>= k = MkState \s →
    let (x, s’) = p s in
    let MkState q = k x in
    q s’
  return x = MkState \s → (x, s)
```

```
effect State s where
  get : m s
  put : s → m ()

handle [s := s0]
  get      k s → k s s
  (put s’) k s → k () s’
  (return x) s → (x, s)
```

1. Why write compilers differently?
2. What are effects and handlers?
3. How to compile with effects and handlers?

A Simple Language

```
effect Arith v where
  int : Integer → m v
  add : v × v   → m v

effect Read v where
  read : m v

effect Let v where
  let  : m v × (v → m v)
    → m v
```

A Simple Language

```
effect Arith v where
  int : Integer → m v
  add : v × v   → m v

effect Read v where
  read : m v

effect Let v where
  let   : m v × (v → m v)
      → m v
```

```
⟦n⟧           = int n
⟦e₁ + e₂⟧     = do x ← ⟦e₁⟧
                   y ← ⟦e₂⟧
                   add x y

⟦read⟧        = read


⟦let x = e₁ in e₂⟧ = let  ⟦e₁⟧ \x →  ⟦e₂⟧
```

X86

```
effect X86 where
  imm   : Integer  → m v
  reg   : Register → m v
  deref : Register × Integer → m v
  movq  : v × v    → m ()
  addq  : v × v    → m ()
  callq : Label    → m ()
```

```
effect X86Var where
  x86var : m v
```

Compiling Let, Arith, Read

```
handle
  (let e f) k → do
    x ← e
    z ← f x
    k z
```

```
handle
  (int n) k → do
    x ← imm n
    z ← x64var
    movq x z
    k z
  (add x y) k → do
    z ← x64var
    movq y z
    addq x z
    k z
```

```
handle
  read k → do
    callq _read_int
    x ← reg %rax
    z ← x64var
    movq x z
    k z
```

Stack Allocation

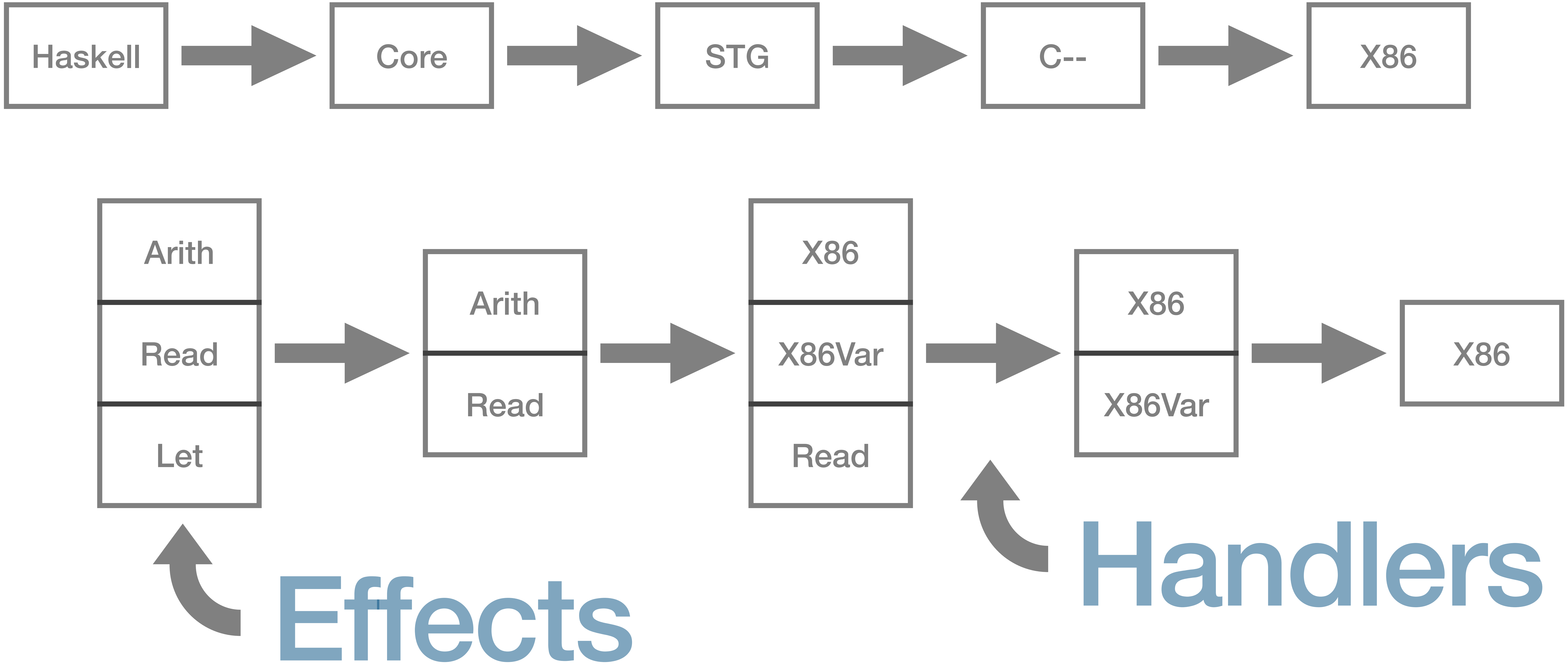
```
handle [n := 1]
  x64var k n → do
    z ← deref %rbp (-8 * n)
    k z (n + 1)
```

Pretty Printing

```
handle
  (imm n)     k → k (showInt n)
  (reg r)     k → k (showReg n)
  (deref r i) k → k (showInt i ++ “(“ ++ showReg r ++ “)”)
  (movq x y)  k → “movq “ ++ x ++ “, “ ++ y ++ “\n” ++ k ()
  (addq x y)  k → “addq “ ++ x ++ “, “ ++ y ++ “\n” ++ k ()
  (callq l)   k → “callq “ ++ l ++ “\n” ++ k ()
```

A New Compiler Architecture

Inspired by nanopass compilers


Haskell
Core
STG
C--
X86
Arith
Read
Let
Arith
Read
X86
X86Var
Read
X86
X86Var
X86
Effects
Handlers