Compiling
Dynamic Information Flow Control

Michael Greenberg
NJPLS Fall 2012

DIFC

```
let y = secret + x

let z = if secret then true else false
```

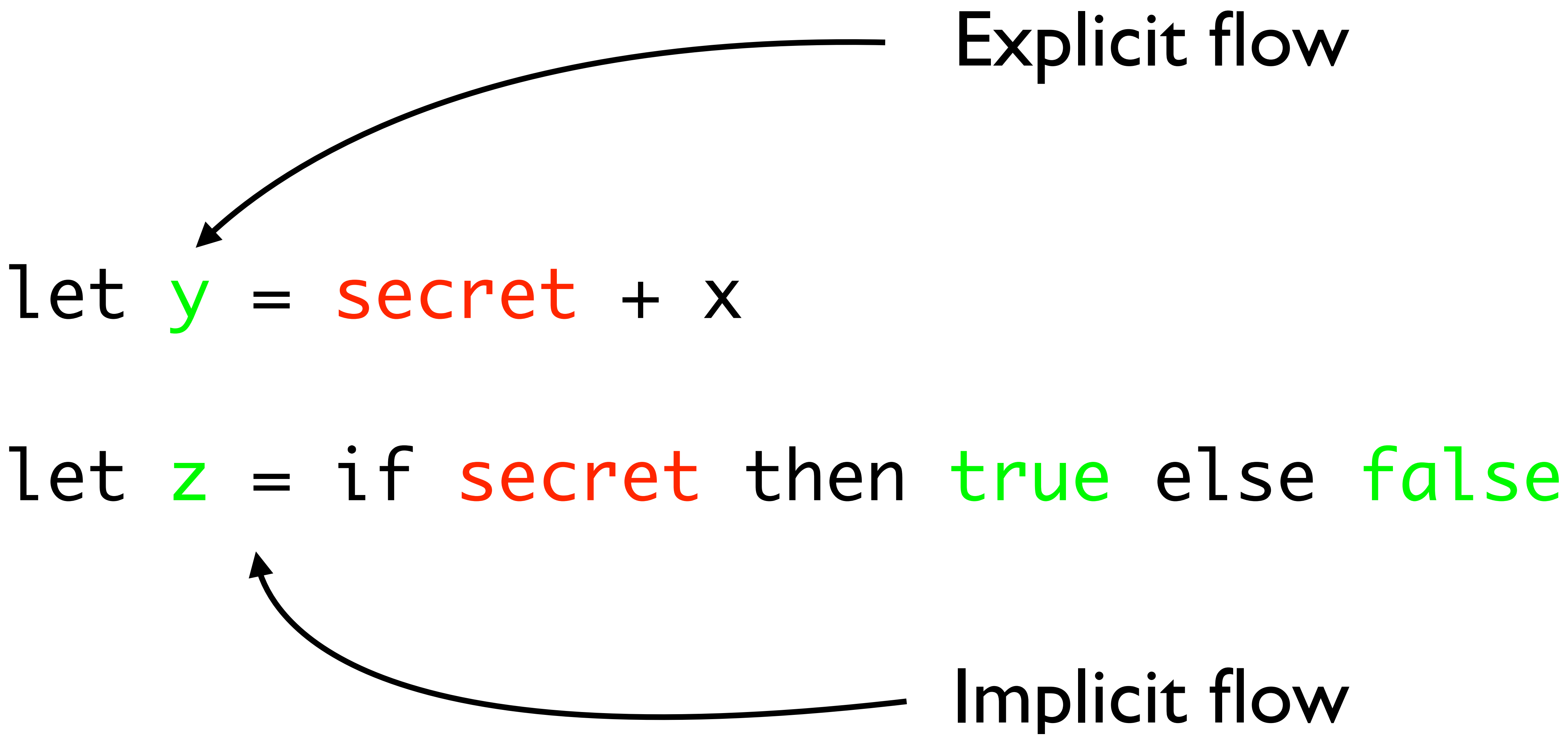
DIFC

Explicit flow

```
let y = secret + x

let z = if secret then true else false
```

DIFC



Explicit flow

```
let y = secret + x

let z = if secret then true else false
```

Implicit flow

Label lattices

Let (L,⊑,⊔,⊥,⊤) be a bounded lattice

Operations join their arguments

```
pc, 5@l + 6@h ⟶ pc, 11@h
```

Labels go on values and on the program counter (*pc*)

```
pc, if true@h then 1 else 0 ⟶ pc ⊔ h, 1
```

Not-a-Value values (NaVs)

- Catalin’s talk?

Not-a-Value values (NaVs)

- Fine-grained DIFC ⇒ delayed exceptions
- NaVs are first-class and labeled
- NaVs propagate via dataflow

3 + (5/0) ⇒ NaV("divide by zero")

(4,2).3 + (5/0) ⇒ NaV("out of bounds")

Compiling DIFC

- What's labeled?
- Where are labels kept?
- Interoperability
  - With labeled programs
  - With unlabeled programs

The SAFE ISA:
A DIFC Architecture

- Every word of memory labeled
- Richly configurable tagged architecture
  - OS support for DIFC
- No escape hatch
- No direct interoperation with unlabeled code

Compiling DIFC on Stock Hardware

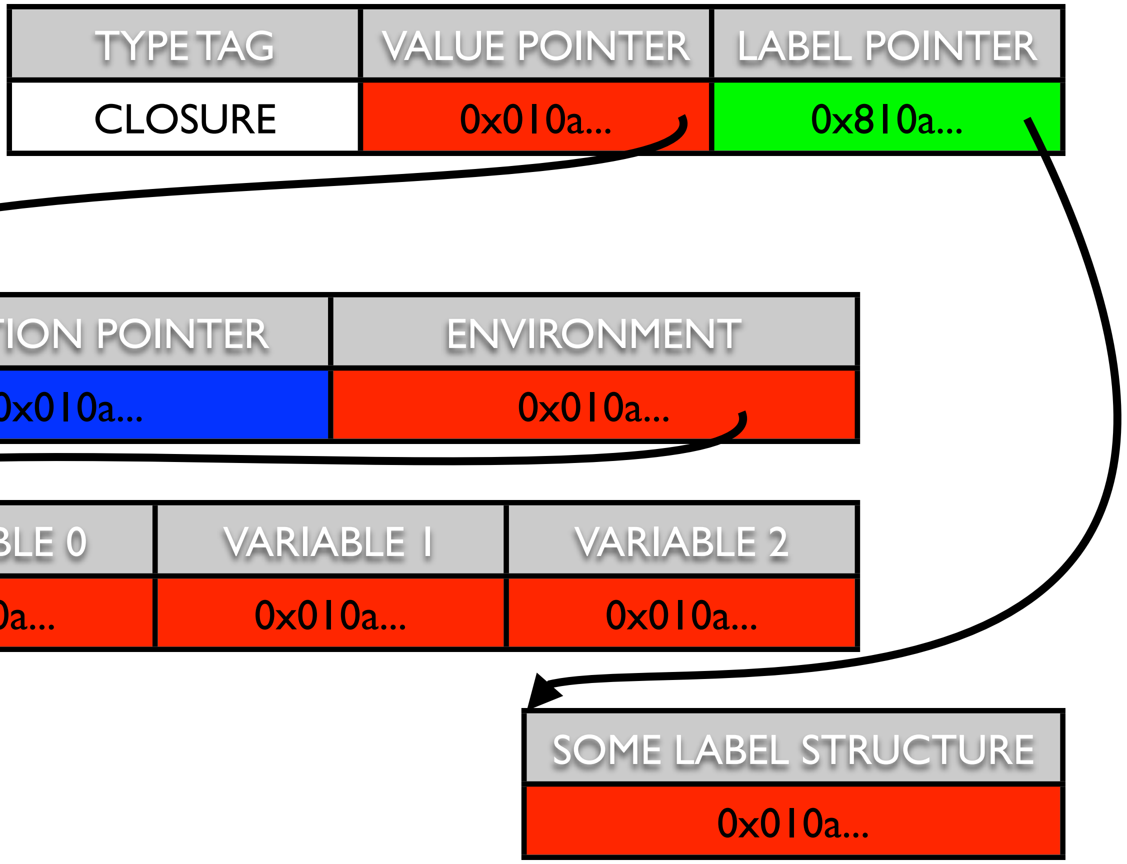
- IFC rules are an abstraction
  - Implementation can vary
  - Observably play by the rules
  - Do what you like when nobody's looking!

What am I doing?

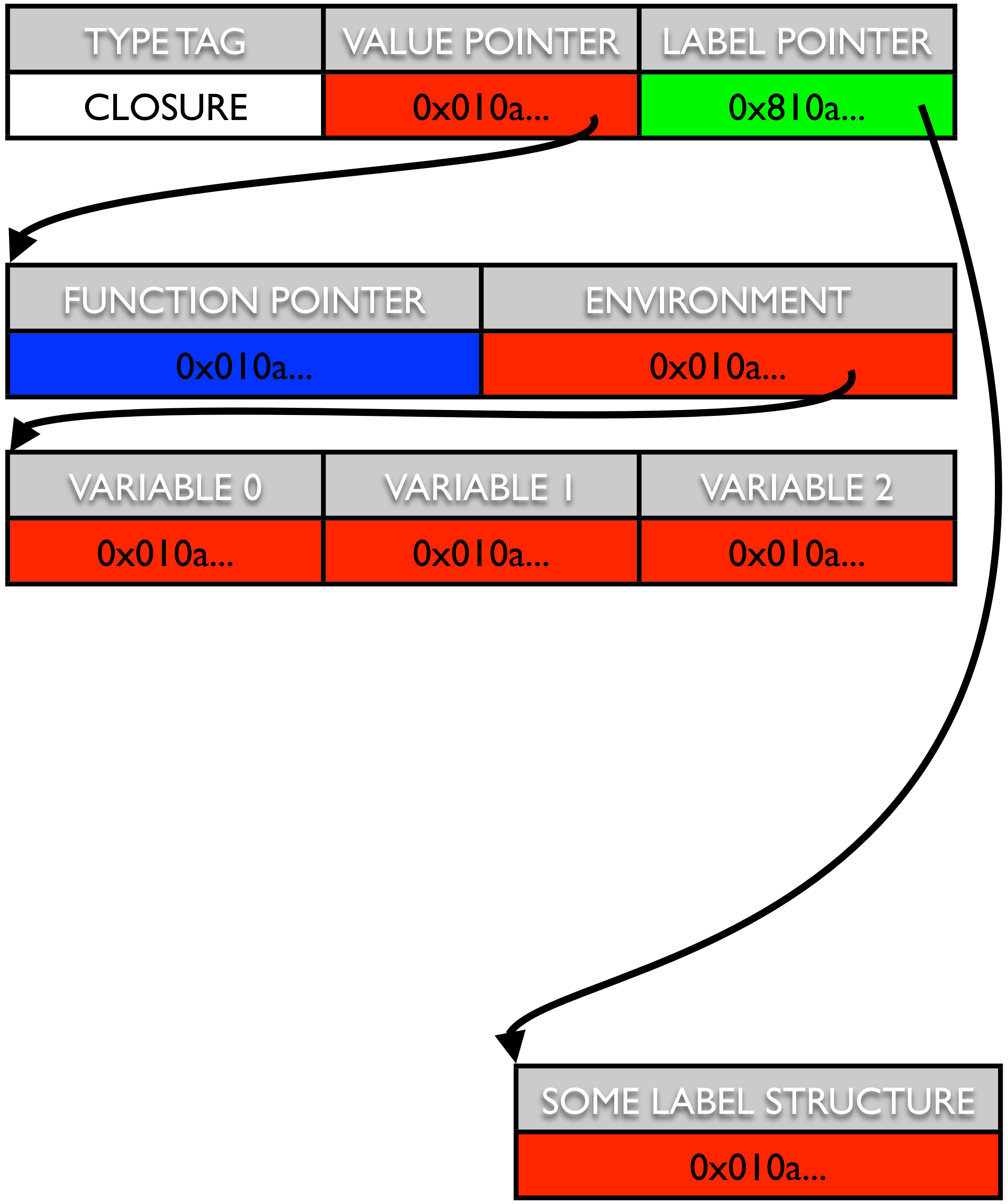
- Compiling a simple DIFC language on x86
  - Pure functional, non-interfering variant of Breeze
- LLVM back-end

What’s labeled?

- Language values are labeled
    - Stored in-line, with the type tag
- Unlabeled things:
    - Closures
    - Environment
    - Tuple arrays



Inline tags



TYPE TAG	VALUE POINTER	LABEL POINTER
CLOSURE	0x010a...	0x810a...

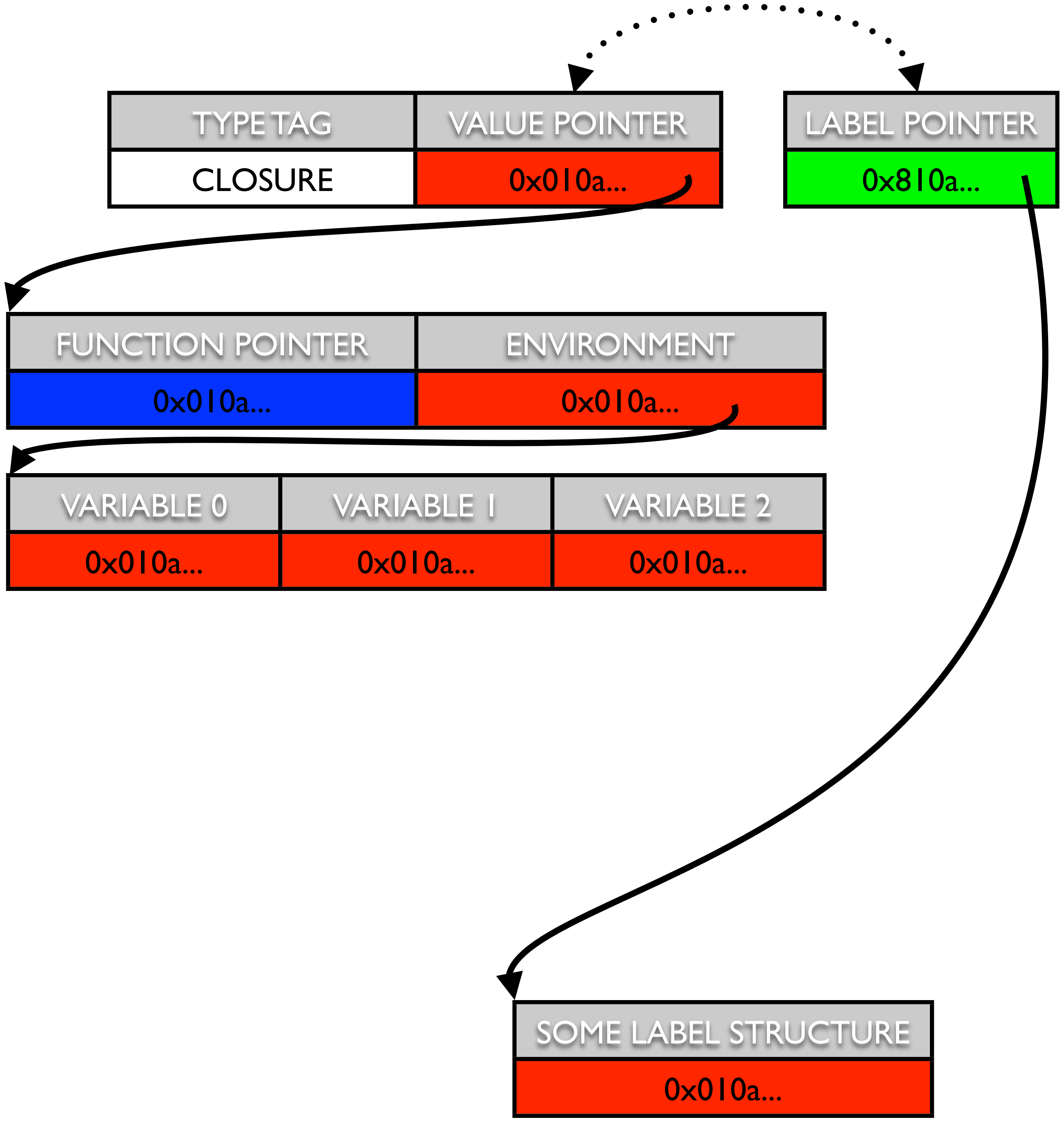
FUNCTION POINTER	ENVIRONMENT
0x010a...	0x010a...

VARIABLE 0	VARIABLE 1	VARIABLE 2
0x010a...	0x010a...	0x010a...

SOME LABEL STRUCTURE
0x010a...

Shadow space

Hash? Address + constant?



TYPE TAG	VALUE POINTER
CLOSURE	0x010a...

LABEL POINTER
0x810a...

FUNCTION POINTER	ENVIRONMENT
0x010a...	0x010a...

VARIABLE 0	VARIABLE 1	VARIABLE 2
0x010a...	0x010a...	0x010a...

SOME LABEL STRUCTURE
0x010a...

Let’s compute $x + y$

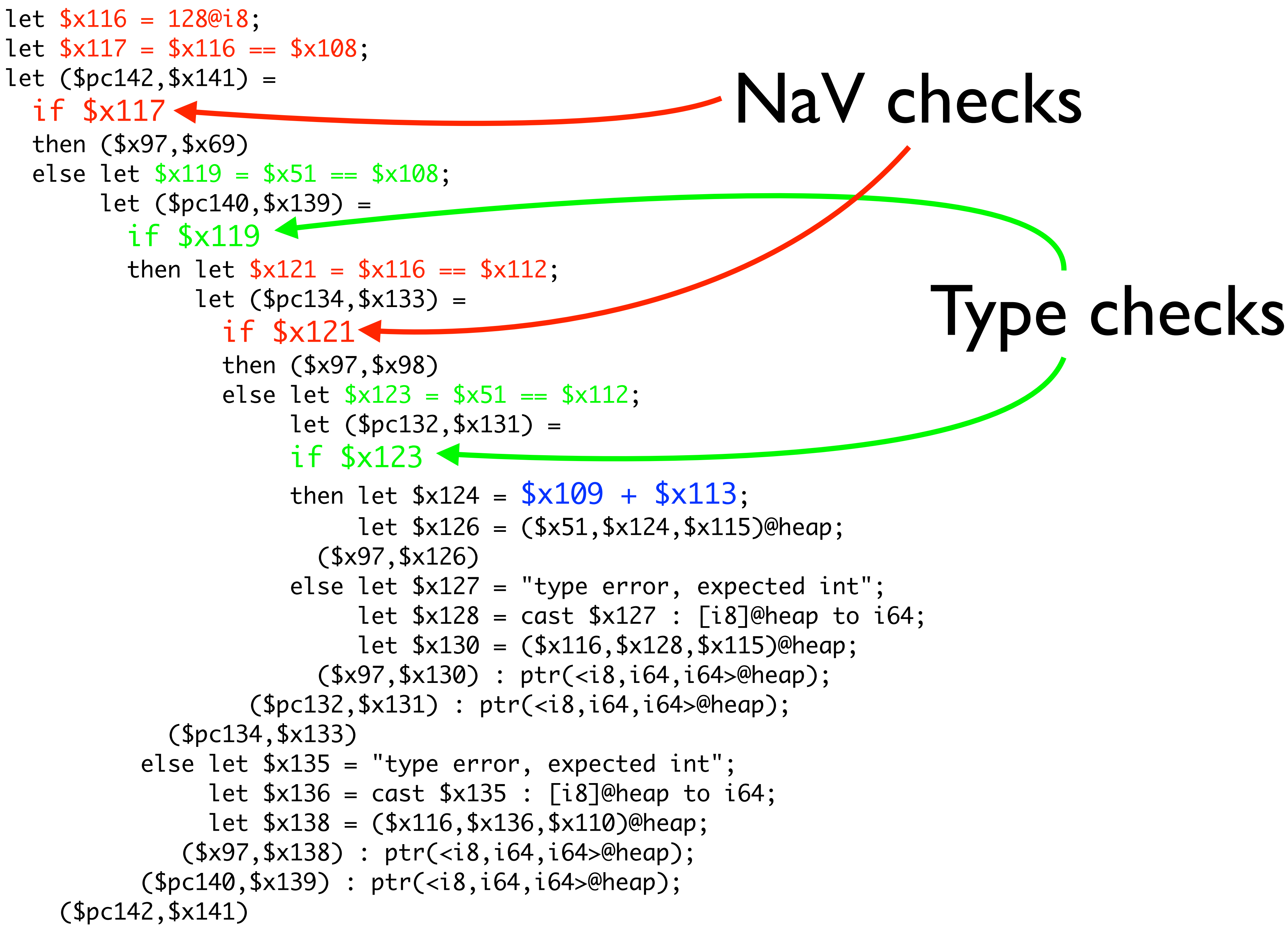
```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

Let’s compute x + y

```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

Type checks

Let’s compute x + y



```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

NaV checks

Type checks

Let’s compute x + y

Let’s compute x + y

```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

NaV checks

Type checks

Actual operation!

Optimization

Low level strategies

- Sound implementations of IFC abstractions
- Optimize!
  - Typed ANF IR that explicitly unpacks labels
  - LLVM backend
- Exploit NaVs

High level strategies

- Label lattice properties are optimizations
  - $\bot$ is the identity for $\sqcup$
  - $\sqcup$ is idempotent; $\sqsubseteq$ is reflexive
- *Abstract interpretation in the label lattice*
- *Contracts*

Work in progress!

Optimizations in action

```
f = \x. (x * x) + 5;
```

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
    let $x1 = $x0.0;
    let $x2 = $x0.1;
    let $x3 = $x0.2;
    let $x4 = load x;
    let $x5 = $x4.0;
    let $x6 = $x4.1;
    let $x7 = $x4.2;
    let $x8 = $x3 \/ $x7;
    let $x9 = 2@i8;
    let $x10 = $x9 == $x1;
    let ($pc27,$x26) =
      if $x10
      then let $x11 = 2@i8;
           let $x12 = $x11 == $x5;
           let ($pc21,$x20) = if $x12 then let $x13 = $x2 * $x6;
                                           let $x14 = 2@i8;
                                           let $x15 = ($x14,$x13,$x8)@heap;
                                             ($pc,$x15) else let $x16 = "type error, expected int";
                                                             let $x17 = cast $x16 : [i8]@heap to i64;
                                                             let $x18 = 128@i8;
                                                             let $x19 = ($x18,$x17,$x8)@heap;
                                                               ($pc,$x19) : ptr(<i8,i64,i64>@heap);
             ($pc21,$x20) else let $x22 = "type error, expected int";
                               let $x23 = cast $x22 : [i8]@heap to i64;
                               let $x24 = 128@i8;
                               let $x25 = ($x24,$x23,$x3)@heap;
                                 ($pc,$x25) : ptr(<i8,i64,i64>@heap);
    let $anf0 = $x26;
    let $x28 = 5@i64;
    let $x29 = bottom;
    let $x30 = 2@i8;
    let $x31 = ($x30,$x28,$x29)@heap;
    let $anf1 = $x31;
    let $x32 = load $anf0;
    let $x33 = $x32.0;
    let $x34 = $x32.1;
    let $x35 = $x32.2;
    let $x36 = load $anf1;
    let $x37 = $x36.0;
    let $x38 = $x36.1;
    let $x39 = $x36.2;
    let $x40 = $x35 \/ $x39;
    let $x41 = 2@i8;
    let $x42 = $x41 == $x33;
    let ($pc59,$x58) =
      if $x42
      then let $x43 = 2@i8;
           let $x44 = $x43 == $x37;
           let ($pc53,$x52) = if $x44
           then let $x45 = $x34 + $x38;
                let $x46 = 2@i8;
                let $x47 = ($x46,$x45,$x40)@heap;
                  ($pc27,$x47)
           else let $x48 = "type error, expected int";
                let $x49 = cast $x48 : [i8]@heap to i64;
                let $x50 = 128@i8;
                let $x51 = ($x50,$x49,$x40)@heap;
                  ($pc27,$x51) : ptr(<i8,i64,i64>@heap);
             ($pc53,$x52)
      else let $x54 = "type error, expected int";
           let $x55 = cast $x54 : [i8]@heap to i64;
           let $x56 = 128@i8;
           let $x57 = ($x56,$x55,$x35)@heap;
             ($pc27,$x57) : ptr(<i8,i64,i64>@heap);
    let $anf2 = $x58;
      ($pc59,$anf2);
```

Optimizations in action

Unoptimized

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
    let $x1 = $x0.0;
    let $x2 = $x0.1;
    let $x3 = $x0.2;
    let $x4 = load x;
    let $x5 = $x4.0;
    let $x6 = $x4.1;
    let $x7 = $x4.2;
    let $x8 = $x3 ⊔ $x7;
    let $x9 = 2@i8;
    let $x10 = $x9 == $x1;
    let ($pc27,$x26) =
      if $x10
      then let $x11 = 2@i8;
           let $x12 = $x11 == $x5;
           let ($pc21,$x20) = if $x12 then let $x13 = $x2 * $x6;
                                           let $x14 = 2@i8;
                                           let $x15 = ($x14,$x13,$x8)@heap;
                                             ($pc,$x15) else let $x16 = "type error, expected int";
                                                             let $x17 = cast $x16 : [i8]@heap to i64;
                                                             let $x18 = 128@i8;
                                                             let $x19 = ($x18,$x17,$x8)@heap;
                                                               ($pc,$x19) : ptr(<i8,i64,i64>@heap);
             ($pc21,$x20) else let $x22 = "type error, expected int";
                               let $x23 = cast $x22 : [i8]@heap to i64;
                               let $x24 = 128@i8;
                               let $x25 = ($x24,$x23,$x3)@heap;
                                 ($pc,$x25) : ptr(<i8,i64,i64>@heap);
    let $anf0 = $x26;
    let $x28 = 5@i64;
    let $x29 = bottom;
    let $x30 = 2@i8;
    let $x31 = ($x30,$x28,$x29)@heap;
    let $anf1 = $x31;
    let $x32 = load $anf0;
    let $x33 = $x32.0;
    let $x34 = $x32.1;
    let $x35 = $x32.2;
    let $x36 = load $anf1;
    let $x37 = $x36.0;
    let $x38 = $x36.1;
    let $x39 = $x36.2;
    let $x40 = $x35 ⊔ $x39;
    let $x41 = 2@i8;
    let $x42 = $x41 == $x33;
    let ($pc59,$x58) =
      if $x42
      then let $x43 = 2@i8;
           let $x44 = $x43 == $x37;
           let ($pc53,$x52) = if $x44
           then let $x45 = $x34 + $x38;
                let $x46 = 2@i8;
                let $x47 = ($x46,$x45,$x40)@heap;
                  ($pc27,$x47)
           else let $x48 = "type error, expected int";
                let $x49 = cast $x48 : [i8]@heap to i64;
                let $x50 = 128@i8;
                let $x51 = ($x50,$x49,$x40)@heap;
                  ($pc27,$x51) : ptr(<i8,i64,i64>@heap);
             ($pc53,$x52)
      else let $x54 = "type error, expected int";
           let $x55 = cast $x54 : [i8]@heap to i64;
           let $x56 = 128@i8;
           let $x57 = ($x56,$x55,$x35)@heap;
             ($pc27,$x57) : ptr(<i8,i64,i64>@heap);
    let $anf2 = $x58;
      ($pc59,$anf2);
```

Optimized

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  let $x9 = 2@i8;
  let $x10 = $x9 == $x1;
  let ($pc27,$x26) =
    if $x10
    then let $x13 = $x2 * $x2;
         let $x15 = ($x9,$x13,$x3)@heap;
           ($pc,$x15)
    else let $x22 = "type error, expected int";
         let $x23 = cast $x22 : [i8]@heap to i64;
         let $x24 = 128@i8;
         let $x25 = ($x24,$x23,$x3)@heap;
           ($pc,$x25) : ptr(<i8,i64,i64>@heap);
  let $x32 = load $x26;
  let $x33 = $x32.0;
  let $x34 = $x32.1;
  let $x35 = $x32.2;
  let $x38 = 5@i64;
  let $x42 = $x9 == $x33;
  let ($pc59,$x58) =
    if $x42
    then let $x45 = $x34 + $x38;
         let $x47 = ($x9,$x45,$x35)@heap;
           ($pc27,$x47)
    else let $x54 = "type error, expected int";
         let $x55 = cast $x54 : [i8]@heap to i64;
         let $x56 = 128@i8;
         let $x57 = ($x56,$x55,$x35)@heap;
           ($pc27,$x57) : ptr(<i8,i64,i64>@heap);
    ($pc59,$x58);
```

Optimizations in action

Unoptimized

Joins eliminated!

Optimized

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
    let $x1 = $x0.0;
    let $x2 = $x0.1;
    let $x3 = $x0.2;
    let $x4 = load x;
    let $x5 = $x4.0;
    let $x6 = $x4.1;
    let $x7 = $x4.2;
    let $x8 = $x3 ⊔ $x7;
    let $x9 = 2@i8;
    let $x10 = $x9 == $x1;
    let ($pc27,$x26) =
      if $x10
      then let $x11 = 2@i8;
           let $x12 = $x11 == $x5;
           let ($pc21,$x20) = if $x12 then let $x13 = $x2 * $x6;
                                           let $x14 = 2@i8;
                                           let $x15 = ($x14,$x13,$x8)@heap;
                                             ($pc,$x15) else let $x16 = "type error, expected int";
                                                             let $x17 = cast $x16 : [i8]@heap to i64;
                                                             let $x18 = 128@i8;
                                                             let $x19 = ($x18,$x17,$x8)@heap;
                                                               ($pc,$x19) : ptr(<i8,i64,i64>@heap);
             ($pc21,$x20) else let $x22 = "type error, expected int";
                               let $x23 = cast $x22 : [i8]@heap to i64;
                               let $x24 = 128@i8;
                               let $x25 = ($x24,$x23,$x3)@heap;
                                 ($pc,$x25) : ptr(<i8,i64,i64>@heap);
    let $anf0 = $x26;
    let $x28 = 5@i64;
    let $x29 = bottom;
    let $x30 = 2@i8;
    let $x31 = ($x30,$x28,$x29)@heap;
    let $anf1 = $x31;
    let $x32 = load $anf0;
    let $x33 = $x32.0;
    let $x34 = $x32.1;
    let $x35 = $x32.2;
    let $x36 = load $anf1;
    let $x37 = $x36.0;
    let $x38 = $x36.1;
    let $x39 = $x36.2;
    let $x40 = $x35 ⊔ $x39;
    let $x41 = 2@i8;
    let $x42 = $x41 == $x33;
    let ($pc59,$x58) =
      if $x42
      then let $x43 = 2@i8;
           let $x44 = $x43 == $x37;
           let ($pc53,$x52) = if $x44
           then let $x45 = $x34 + $x38;
                let $x46 = 2@i8;
                let $x47 = ($x46,$x45,$x40)@heap;
                  ($pc27,$x47)
           else let $x48 = "type error, expected int";
                let $x49 = cast $x48 : [i8]@heap to i64;
                let $x50 = 128@i8;
                let $x51 = ($x50,$x49,$x40)@heap;
                  ($pc27,$x51) : ptr(<i8,i64,i64>@heap);
             ($pc53,$x52)
      else let $x54 = "type error, expected int";
           let $x55 = cast $x54 : [i8]@heap to i64;
           let $x56 = 128@i8;
           let $x57 = ($x56,$x55,$x35)@heap;
             ($pc27,$x57) : ptr(<i8,i64,i64>@heap);
    let $anf2 = $x58;
      ($pc59,$anf2);
```

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  let $x9 = 2@i8;
  let $x10 = $x9 == $x1;
  let ($pc27,$x26) =
    if $x10
    then let $x13 = $x2 * $x2;
         let $x15 = ($x9,$x13,$x3)@heap;
           ($pc,$x15)
    else let $x22 = "type error, expected int";
         let $x23 = cast $x22 : [i8]@heap to i64;
         let $x24 = 128@i8;
         let $x25 = ($x24,$x23,$x3)@heap;
           ($pc,$x25) : ptr(<i8,i64,i64>@heap);
  let $x32 = load $x26;
  let $x33 = $x32.0;
  let $x34 = $x32.1;
  let $x35 = $x32.2;
  let $x38 = 5@i64;
  let $x42 = $x9 == $x33;
  let ($pc59,$x58) =
    if $x42
    then let $x45 = $x34 + $x38;
         let $x47 = ($x9,$x45,$x35)@heap;
           ($pc27,$x47)
    else let $x54 = "type error, expected int";
         let $x55 = cast $x54 : [i8]@heap to i64;
         let $x56 = 128@i8;
         let $x57 = ($x56,$x55,$x35)@heap;
           ($pc27,$x57) : ptr(<i8,i64,i64>@heap);
    ($pc59,$x58);
```

CSE

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
    let $x1 = $x0.0;
    let $x2 = $x0.1;
    let $x3 = $x0.2;
    let $x4 = load x;
    let $x5 = $x4.0;
    let $x6 = $x4.1;
    let $x7 = $x4.2;
    let $x8 = $x3 ⊔ $x7;
    ...
```

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  let $x8 = $x3 ⊔ $x3;
  ...
```

```
f = \x. (x * x) + 5;
```

Reflexivity

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  let $x8 = $x3 ⊔ $x3;
  ...
```

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  let $x8 = $x3;
  ...
```

```
f = \x. (x * x) + 5;
```

Variable reduction

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  let $x8 = $x3;
  ...
```

```
fun $clo0 (x,$env,$pc) =
  let $x0 = load x;
  let $x1 = $x0.0;
  let $x2 = $x0.1;
  let $x3 = $x0.2;
  ...[$x8 ↦ $x3]
```

```
f = \x. (x * x) + 5;
```

Constant folding

```
...
let $anf0 = $x26;
let $x28 = 5@i64;
let $x29 = bottom;
let $x30 = 2@i8;
let $x31 = ($x30,$x28,$x29)@heap;
let $anf1 = $x31;
let $x32 = load $anf0;
let $x33 = $x32.0;
let $x34 = $x32.1;
let $x35 = $x32.2;
let $x36 = load $anf1;
let $x37 = $x36.0;
let $x38 = $x36.1;
let $x39 = $x36.2;
let $x40 = $x35 ⊔ $x39;
...
```

```
f = \x. (x * x) + 5;
```

Constant folding

```
...
let $anf0 = $x26;
let $x28 = 5@i64;
let $x29 = bottom;
let $x30 = 2@i8;
let $x31 = ($x30,$x28,$x29)@heap;
let $anf1 = $x31;
let $x32 = load $anf0;
let $x33 = $x32.0;
let $x34 = $x32.1;
let $x35 = $x32.2;
let $x36 = load $anf1;
let $x37 = $x36.0;
let $x38 = $x36.1;
let $x39 = $x36.2;
let $x40 = $x35 ⊔ $x39;
...
```

```
...
let $x32 = load $x26;
let $x33 = $x32.0;
let $x34 = $x32.1;
let $x35 = $x32.2;
let $x38 = 5@i64;
let $x40 = $x35;
...
```

Bottom is a lattice identity

```
let $x40 = $x35 ⊔ bottom;
```

```
f = \x. (x * x) + 5;
```

Variable reduction

```
...
let $x32 = load $x26;
let $x33 = $x32.0;
let $x34 = $x32.1;
let $x35 = $x32.2;
let $x38 = 5@i64;
let $x40 = $x35;
...
```

```
...
let $x32 = load $x26;
let $x33 = $x32.0;
let $x34 = $x32.1;
let $x35 = $x32.2;
let $x38 = 5@i64;
...[$40 ↦ $x35]
```

```
f = \x. (x * x) + 5;
```

Lattice operations in the IR

- Most IR constructs correspond to LLVM
- EBottom, EJoin, EFlowsTo AST nodes don't
  - Abstract representation
  - Enables optimizations
    - Constant folding has a VBottom

Stupid NaV tricks

- Dedicated "is a NaV" bit in type tags
- For safe operations on values:
  - Perform the operation regardless
  - Bitwise AND the tags
  - Check the NaV bit
    - Error info stored on the side (DWARF)

NaV branch minimization*

```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

Computing x + y

* *Work in progress*

NaV branch minimization*

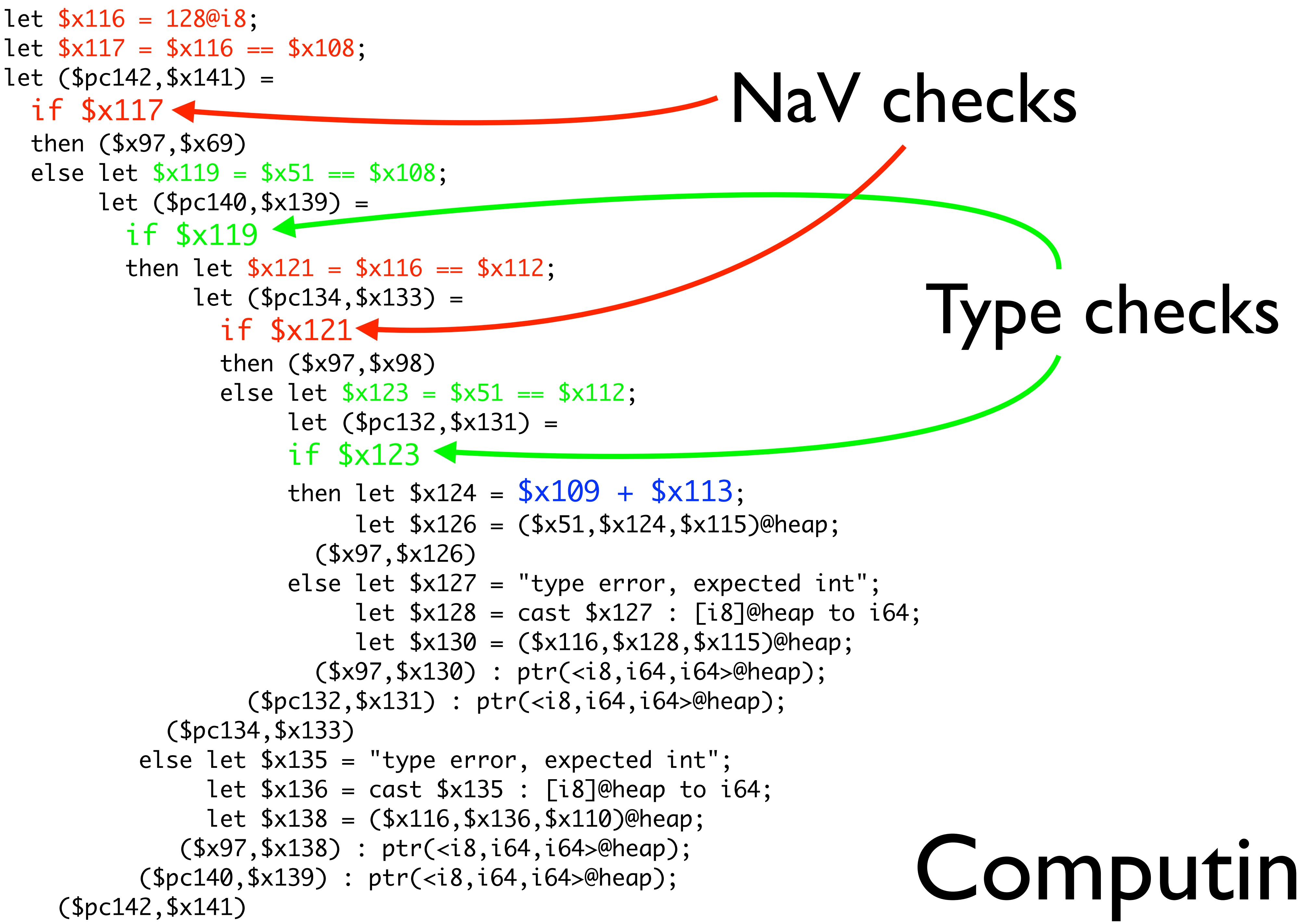
```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

Type checks

Computing x + y

* *Work in progress*

NaV branch minimization*



```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

NaV checks

Type checks

Computing x + y

* *Work in progress*

NaV branch minimization*

```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                          let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                          let $x128 = cast $x127 : [i8]@heap to i64;
                          let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

NaV checks

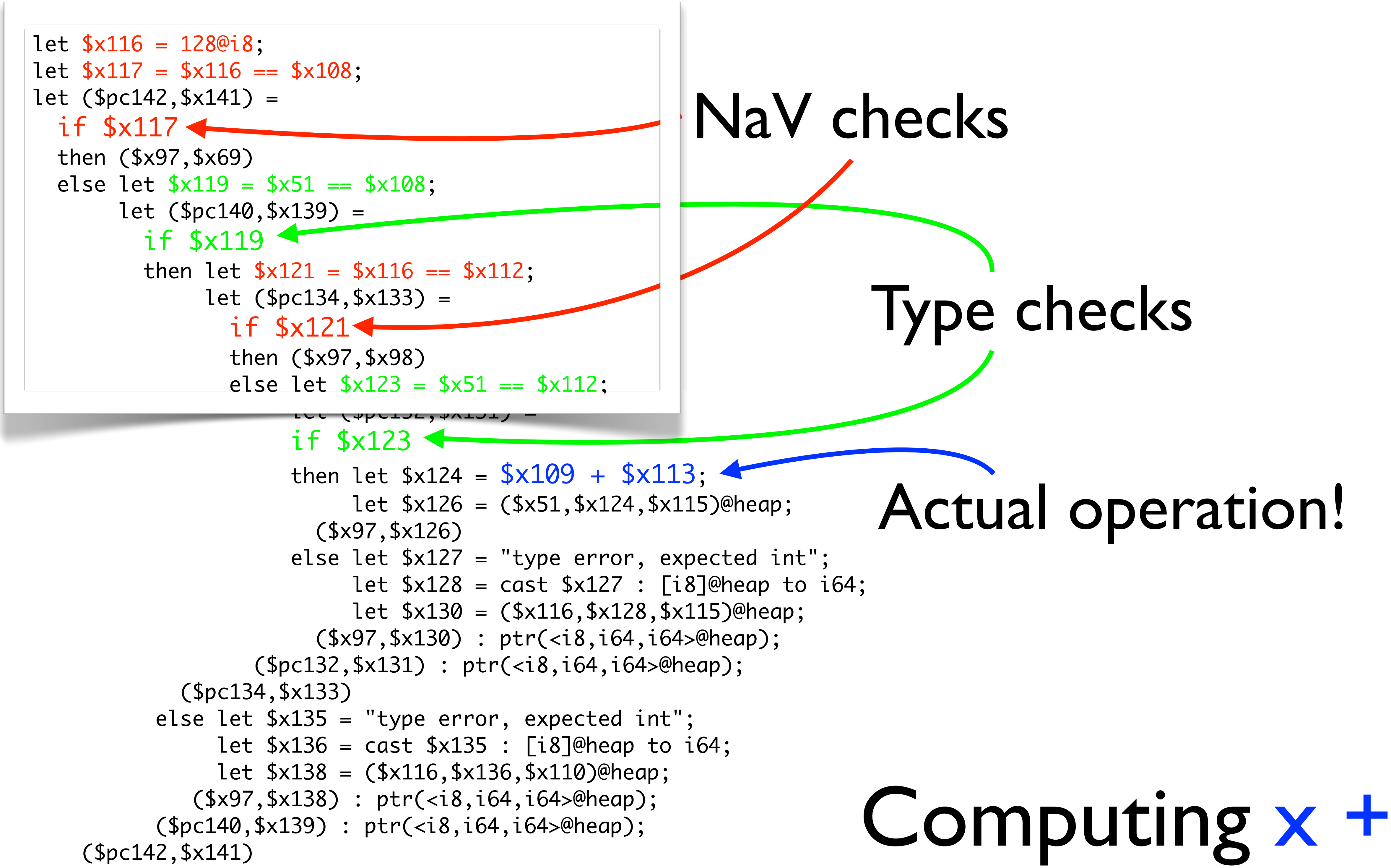
Type checks

Actual operation!

Computing x + y

* *Work in progress*

NaV branch minimization*



```
let $x116 = 128@i8;
let $x117 = $x116 == $x108;
let ($pc142,$x141) =
  if $x117
  then ($x97,$x69)
  else let $x119 = $x51 == $x108;
       let ($pc140,$x139) =
         if $x119
         then let $x121 = $x116 == $x112;
              let ($pc134,$x133) =
                if $x121
                then ($x97,$x98)
                else let $x123 = $x51 == $x112;
                     let ($pc132,$x131) =
                     if $x123
                     then let $x124 = $x109 + $x113;
                              let $x126 = ($x51,$x124,$x115)@heap;
                       ($x97,$x126)
                     else let $x127 = "type error, expected int";
                              let $x128 = cast $x127 : [i8]@heap to i64;
                              let $x130 = ($x116,$x128,$x115)@heap;
                       ($x97,$x130) : ptr(<i8,i64,i64>@heap);
                  ($pc132,$x131) : ptr(<i8,i64,i64>@heap);
            ($pc134,$x133)
          else let $x135 = "type error, expected int";
               let $x136 = cast $x135 : [i8]@heap to i64;
               let $x138 = ($x116,$x136,$x110)@heap;
             ($x97,$x138) : ptr(<i8,i64,i64>@heap);
          ($pc140,$x139) : ptr(<i8,i64,i64>@heap);
    ($pc142,$x141)
```

NaV checks

Type checks

Actual operation!

Computing x + y

*Work in progress

NaV branch minimization

Single branch in non-NaV case

```
let $x51 = 2@i8; // int tag
...
let $x124 = $x109 + $x113;
let $fstok = $x108 == $x51;
let $sndok = $x112 == $x51;
let $bothok = $fstok && $sndok;
let ($pc142,$x141) =
  if $bothok
  then let $x126 = ($x51,$x124,$x115)@heap;
     ($x97,$x126)
  else ... // check tags, figure out which NaV to return
```

Computing x + y

NaV branch minimization

```
let $x116 = 128@i8; // nav tag
let $navbits = $x108 | $x112;
let $havenav = $x116 & $navbits;
let ($pc142,$x141) =
  if $havenav
  then ... choose between ($x97,$x69)  and ($x97,$x98) ...
  else ... // some parametric operation
```

NaV-strict,
parametric
operations

Outlook

- Finishing up compiler now
- Lots of evaluation to do
- Beyond performance: questions!
    - Interoperability
    - Contracts