



Refinement Types for Haskell

Niki Vazou

University of Maryland

Software bugs are everywhere



Airbus A400M crashed due to a software bug.
— May 2015

Software bugs are everywhere



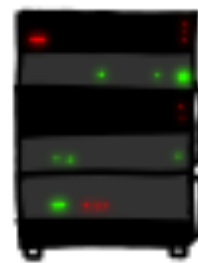
The Heartbleed Bug.
Buffer overflow in OpenSSL. 2015

HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).



...ns pages about "boats". User Erica requests
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H



...ns pages about "boats". User Erica requests
secure connection using key "4538538374224"
User Meg wants these 6 letters: **POTATO**. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H

POTATO



SERVER, ARE YOU STILL THERE?
IF SO, REPLY "BIRD" (4 LETTERS).



User Olivia from London wants pages about "nan
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bfff8d



HMM...



User Olivia from London wants pages about "nan
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: **BIRD**. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bfff8d

BIRD



SERVER, ARE YOU STILL THERE?

a connection. Jake requested pictures of deer.
User Meg wants these 500 letters: HAT. Lucas

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "HAT" (500 LETTERS).

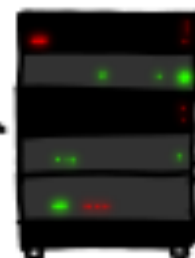


a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User



HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User

a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User



Make bugs difficult to express

Using Modern Programming Languages

F#, Ocaml, Erlang, Scala, Haskell

Because of

Strong Types + λ -Calculus

Make bugs difficult to express

Using Mode

*Well Typed Programs
cannot go wrong!*

languages

F#, Ocaml

many, Scala, Haskell



Because of

Strong Types + λ -Calculus



VS.



```
λ> :m +Data.Text Data.Text.Unsafe
```

```
λ> let pack = "hat"
```

```
λ> :t takeWord16
```

```
takeWord16 :: Text -> Int -> Text
```



VS.



```
λ> :m +Data.Text Data.Text.Unsafe
```

```
λ> let pack = "hat"
```

```
λ> takeWord16 pack True
```

```
Type Error: Cannot match Bool vs Int
```



VS.



```
λ> :m +Data.Text Data.Text.Unsafe
```

```
λ> let pack = "hat"
```

```
λ> takeWord16 pack 500  
"hat\58456\2594\SOH\NUL..."
```



VS.



Valid Values for takeWord16?

```
takeWord16 :: t:Text -> i:Int -> Text
```

All Ints

..., -2, -1, 0, 1, 2, 3, ...

Valid Values for takeWord16?

`takeWord16 :: t:Text -> i:Int -> Text`

Valid Ints

0, 1 ... , len t

Invalid Ints

len t + 1, ...

Refinement Types

`take :: t:Text -> {v:Int | v < len t} -> Text`

Valid Ints

0, 1 ... , len t

Invalid Ints

len t + 1, ...

Refinement Types

`take :: t:Text -> {v:Int | v < len t} -> Text`

```
λ> :m +Data.Text Data.Text.Unsafe
```

```
λ> let pack = "hat"
```

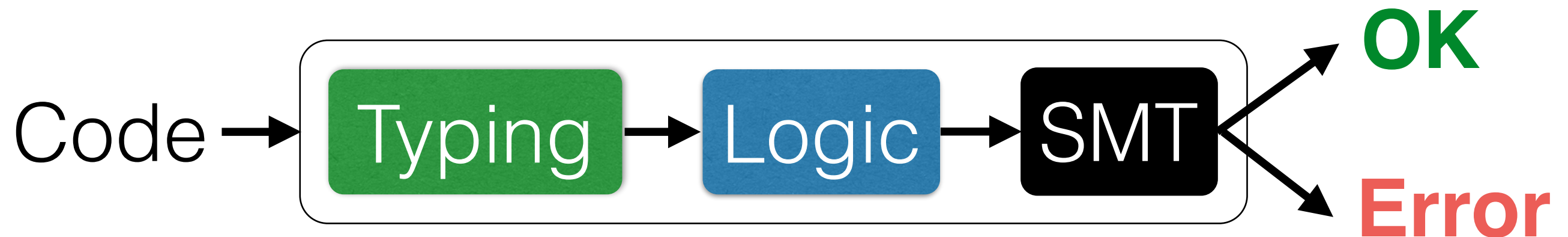
```
λ> take pack 500  
Refinement Type Error
```



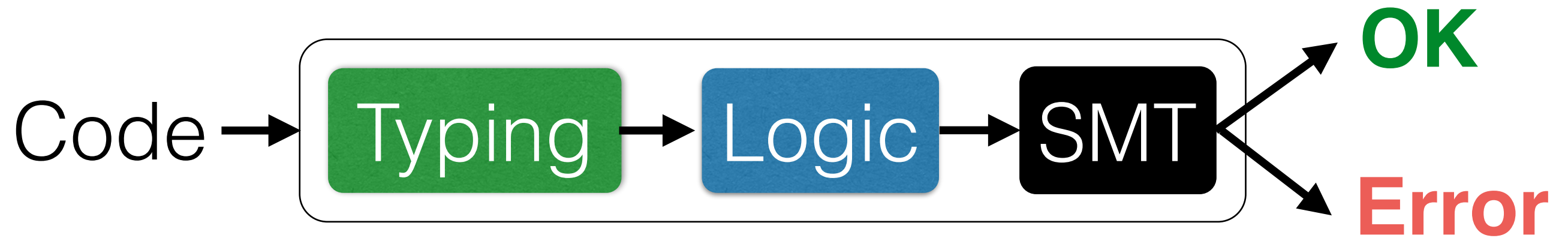
Refinement Types



Refinement Types



1. Source Code to **Type constraints**
2. **Type Constraints** to **Verification Condition (VC)**
3. Check **VC validity** with **SMT Solver**



```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
              in take x 8
```

Code → Typing

```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
              in take x 8
```

$x : \{v \mid \text{len } v = 3\} \vdash \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$

Code \rightarrow Typing

```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
              in take x 8
```

$x:\{v \mid \text{len } v = 3\} \vdash \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$

Code → Typing

```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
             in take x 8
```

$x : \{v \mid \text{len } v = 3\} \quad |- \quad \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$

Code \rightarrow Typing

```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
             in take x 8
```

$x : \{v \mid \text{len } v = 3\} \quad |- \quad \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$

Code \rightarrow Typing

```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
             in take x 8
```

$x : \{v \mid \text{len } v = 3\} \vdash \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$



Encode Subtyping ...

$x : \{v \mid \text{len } v = 3\} \vdash \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$

... as Logical VC

$\text{len } x = 3 \Rightarrow (v = 8) \Rightarrow (v \leq \text{len } x)$

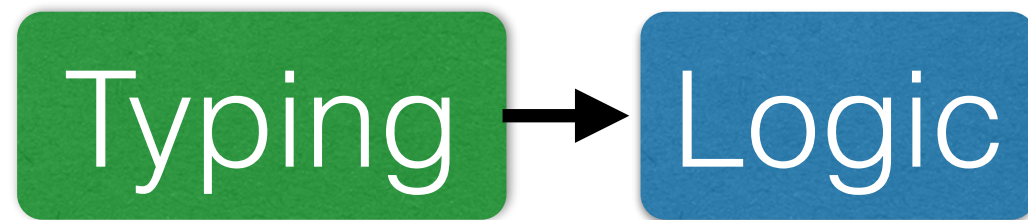


Encode Subtyping ...

$x : \{v \mid \text{len } v = 3\}$ $\vdash \{v \mid v = 8\} <: \{v \mid v \leq \text{len } x\}$

... as Logical VC

$\text{len } x = 3$ $\Rightarrow (v = 8) \Rightarrow (v \leq \text{len } x)$

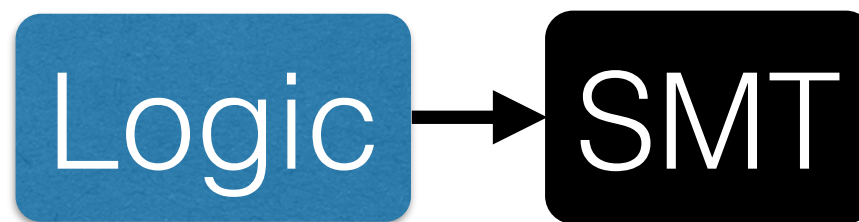


Encode Subtyping ...

$x : \{v \mid \text{len } v = 3\} \vdash \underline{\{v \mid v = 8\}} <: \{v \mid v \leq \text{len } x\}$

... as Logical VC

$\text{len } x = 3 \Rightarrow \underline{(v = 8)} \Rightarrow (v \leq \text{len } x)$



$\text{len } x = 3 \Rightarrow (v = 8) \Rightarrow (v \leq \text{len } x)$

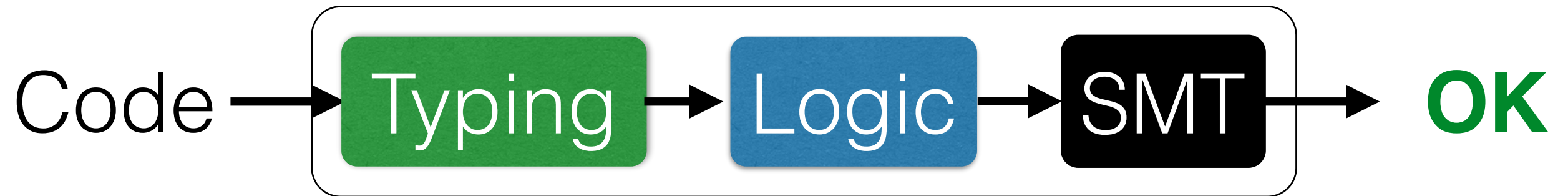


$\text{len } x = 3 \Rightarrow (v = 8) \Rightarrow (v \leq \text{len } x)$



```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
              in take x 8
```

$\text{len } x = 3 \Rightarrow (v = 8) \Rightarrow (v \leq \text{len } x)$



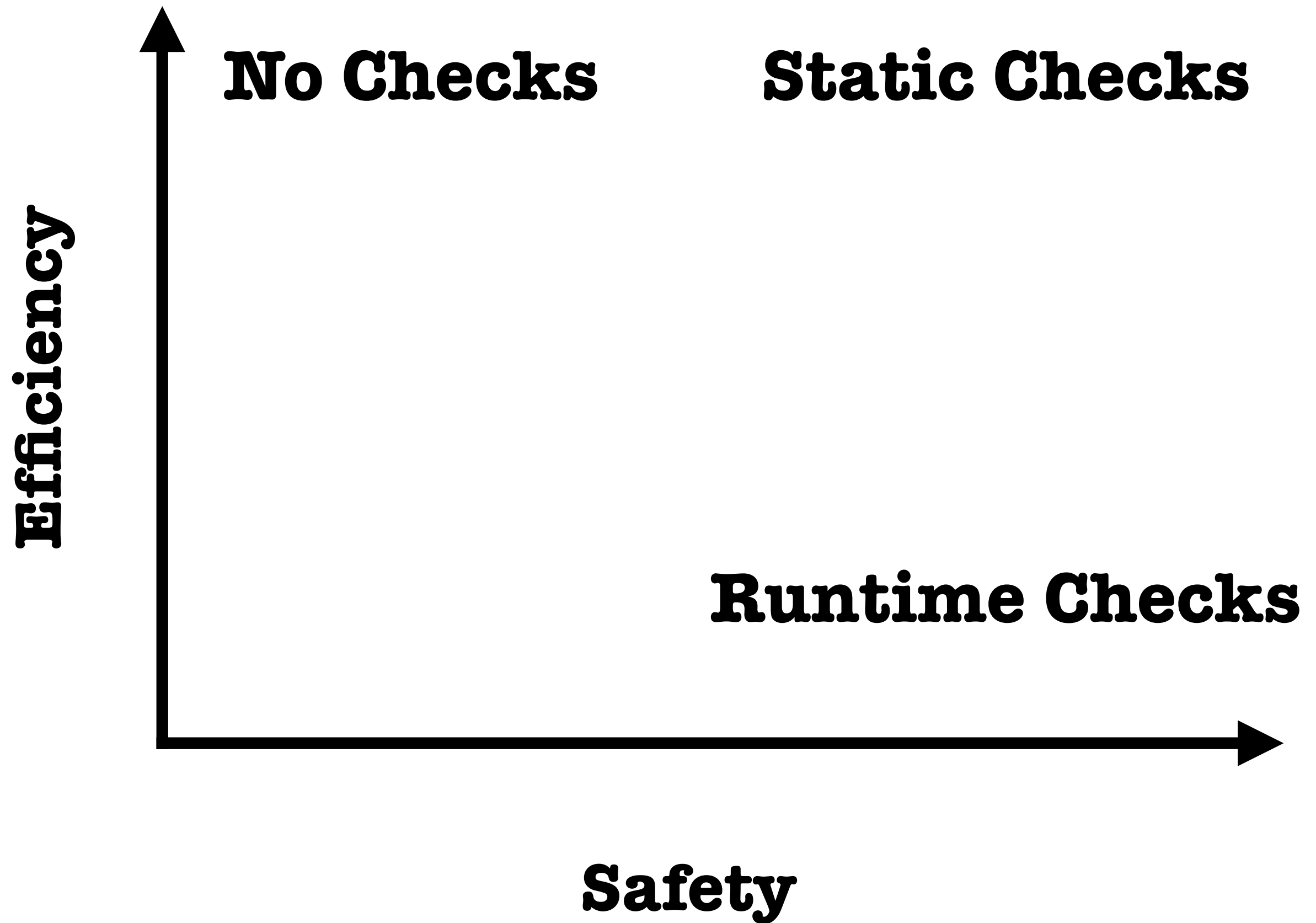
```
take :: t:Text -> {v | v <= len t} -> Text
heartbleed = let x = "hat"
              in take x 2
```

$\text{len } x = 3 \Rightarrow (v = 2) \Rightarrow (v \leq \text{len } x)$



Checks valid arguments, under facts.

Static Checks



No Checks

```
take :: t:Text -> i:Int -> Text  
take i t  
    = Unsafe.takeWord16 i t
```

No Checks

```
take :: t:Text -> i:Int -> Text  
take i t  
    = Unsafe.takeWord16 i t
```

```
heartbleed = take "hat" 500
```



OK

No Checks

```
take :: t:Text -> i:Int -> Text  
take i t  
    = Unsafe.takeWord16 i t
```

```
heartbleed = take "hat" 500
```

OK

```
λ> heartbleed
```

```
λ> “hat\58456\2594\SOH\NUL...
```

UNSAFE

Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

```
heartbleed = take "hat" 500
```

OK

Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

```
heartbleed = take "hat" 500
```

OK

```
λ> heartbleed
```

```
λ> *** Exception: Out Of Bounds!
```

SAFE

Runtime Checks are expensive

```
take :: t:Text -> i:Int -> Text  
take i t | i < len t  
    = Unsafe.takeWord16 i t  
take i t  
    = error "Out Of Bounds!"
```

Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
```

```
take i t | i < len t
```

```
= Unsafe.takeWord16 i t
```

```
take i t
```

```
= error "Out Of Bounds!"
```

Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
```

```
take i t
```

```
= Unsafe.takeWord16 i t
```

Static Checks

```
take :: t:Text -> i:{i < len t} -> Text  
take i t  
    = Unsafe.takeWord16 i t
```

UNSAFE

```
heartbleed = take "hat" 500
```




Static Checks

Safe & Efficient Code!