Gradual Information Flow Typing

Tim Disney
Cormac Flanagan

University of California Santa Cruz
January 29th, 2011 - STOP 2011

Combining gradual typing with information flow

Gradual Typing + Information Flow

=

More secure software

Perfect world

Security designed upfront

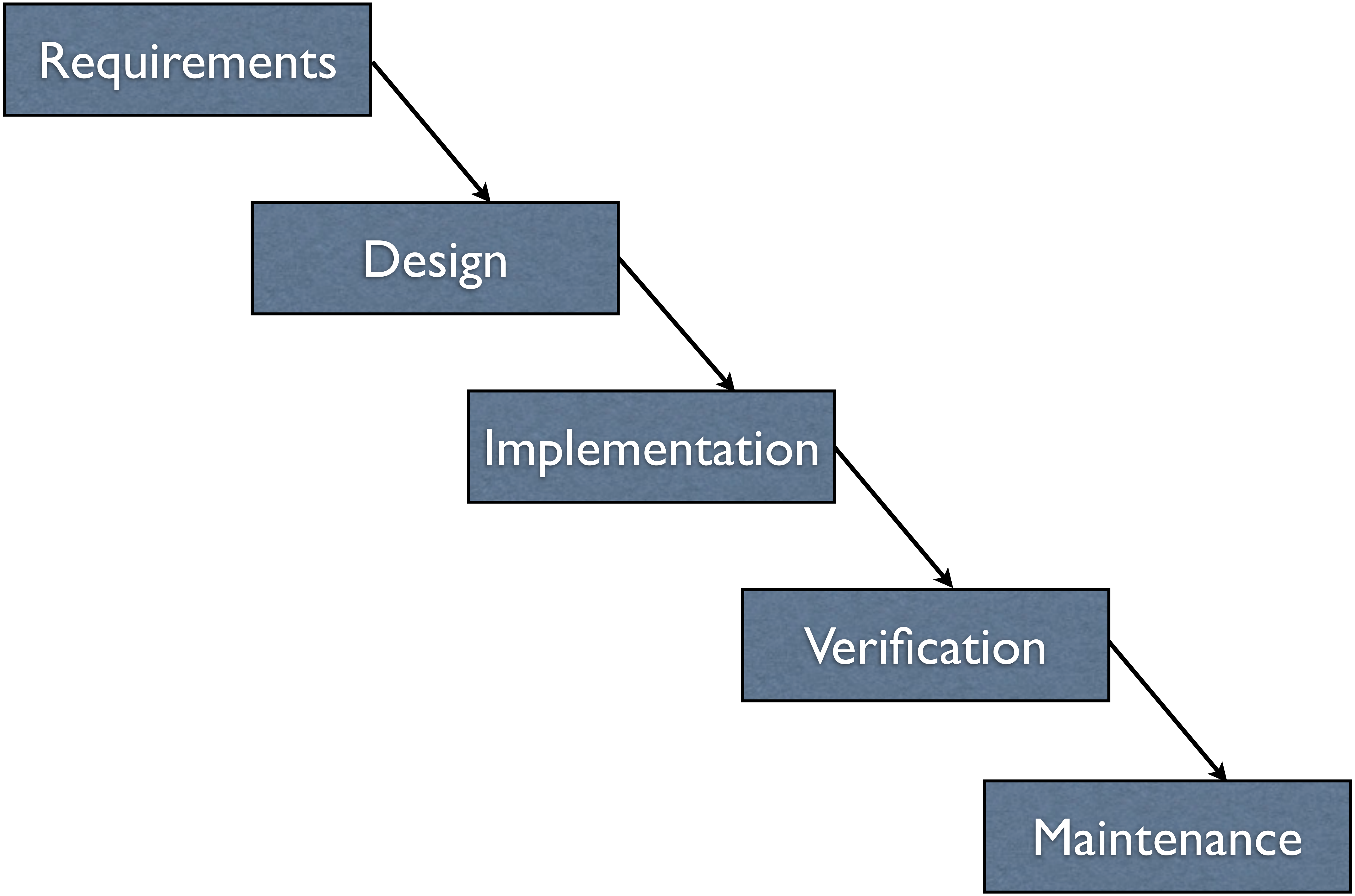
Perfect world

Security designed upfront

vs.

Broken world

Security bolted on after the fact


Requirements
Design
Implementation
Verification
Maintenance

Finding security requirements upfront is often not economically feasible

Perfect world

Broken world

Perfect world

Br… world

Do not want

Per[illegible] World

Does not exist

Br[illegible] World

Do not want

Real world

Security evolves with program

Perf… world — Does not exist

Bro… world — Do not want

Information Flow

Confidentiality: keeping sensitive data private

&

Integrity: protect against untrusted data

Information Flow

Confidentiality: keeping sensitive data private

&

Integrity: protect against untrusted data

For this talk we focus on confidentiality

Information Flow

Labels

H (private)

↑

L (public)

Information Flow

Labels

H (private)

↑

L (public)

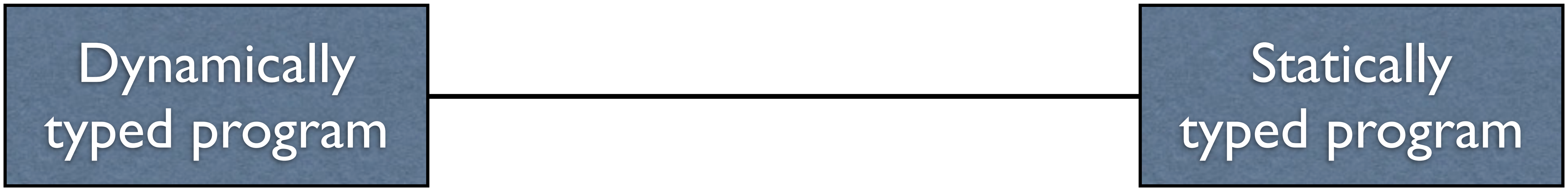
Labeled Values

42^{L}

$58{,}000^{H}$

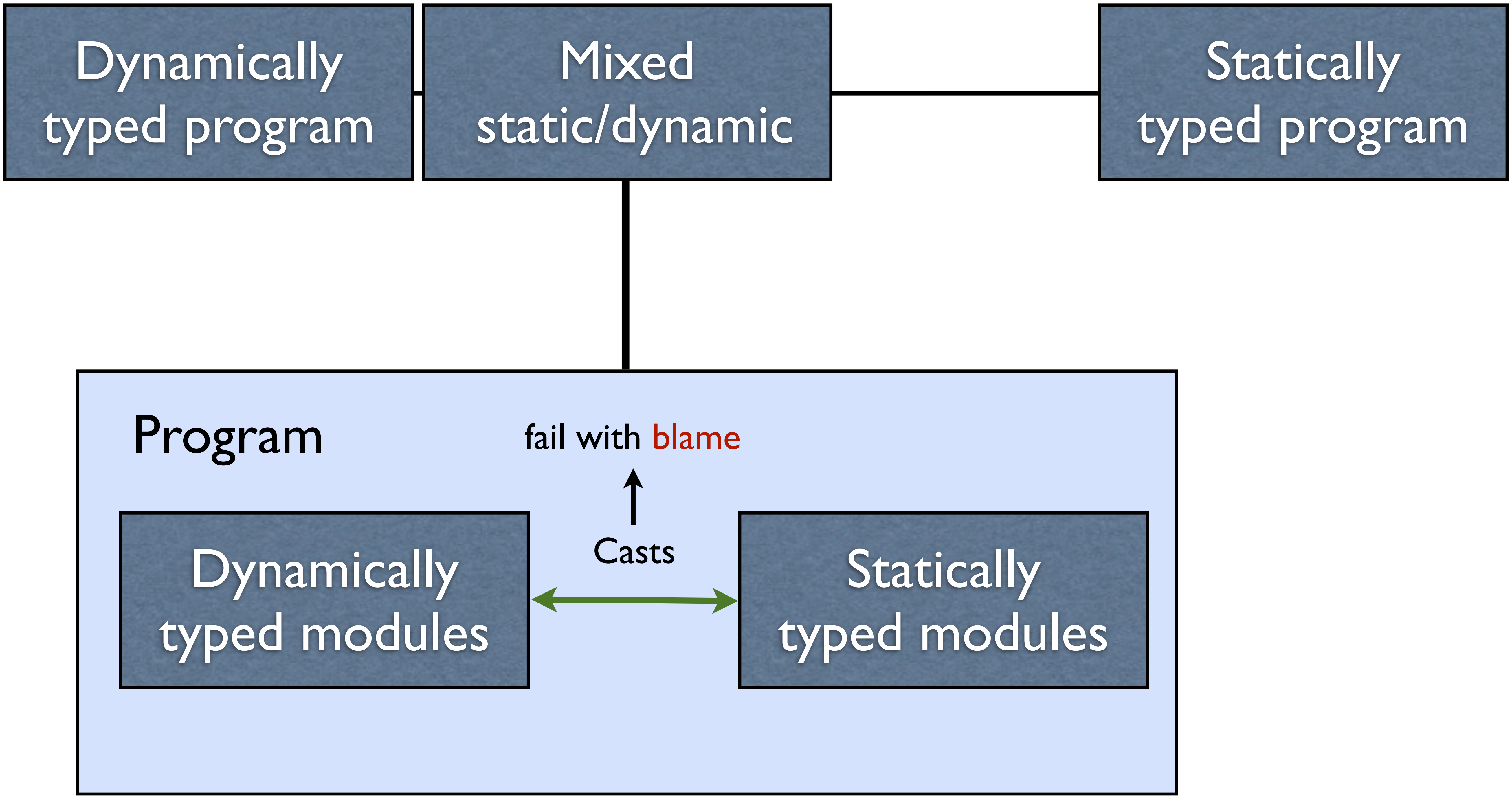
“hello”^{L}

Gradual Typing



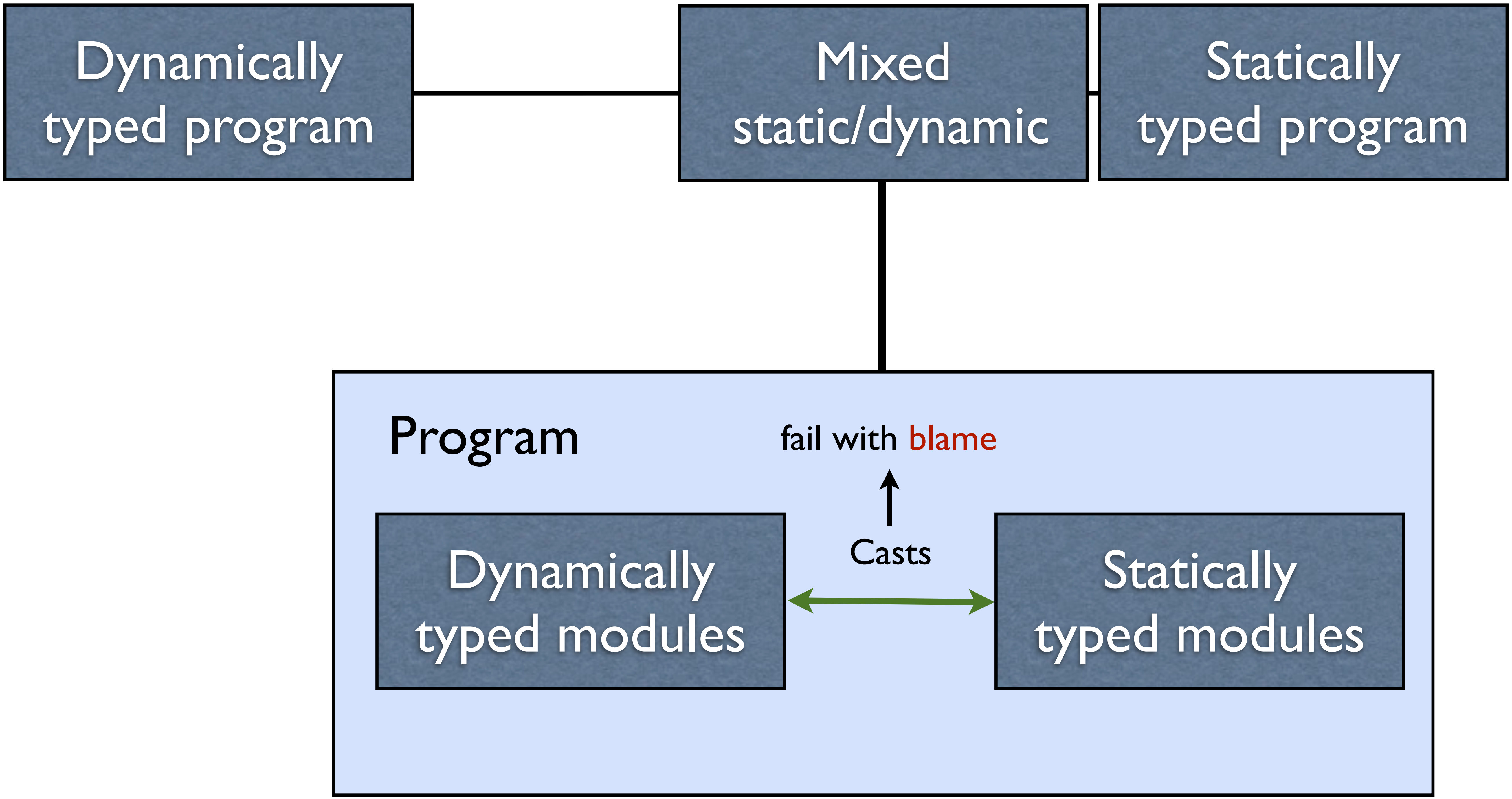
Siek and Taha (2006), Findler and Felleisen (2002), Wadler and Findler (2009), Ahmed et al. (2011), etc.

Gradual Typing



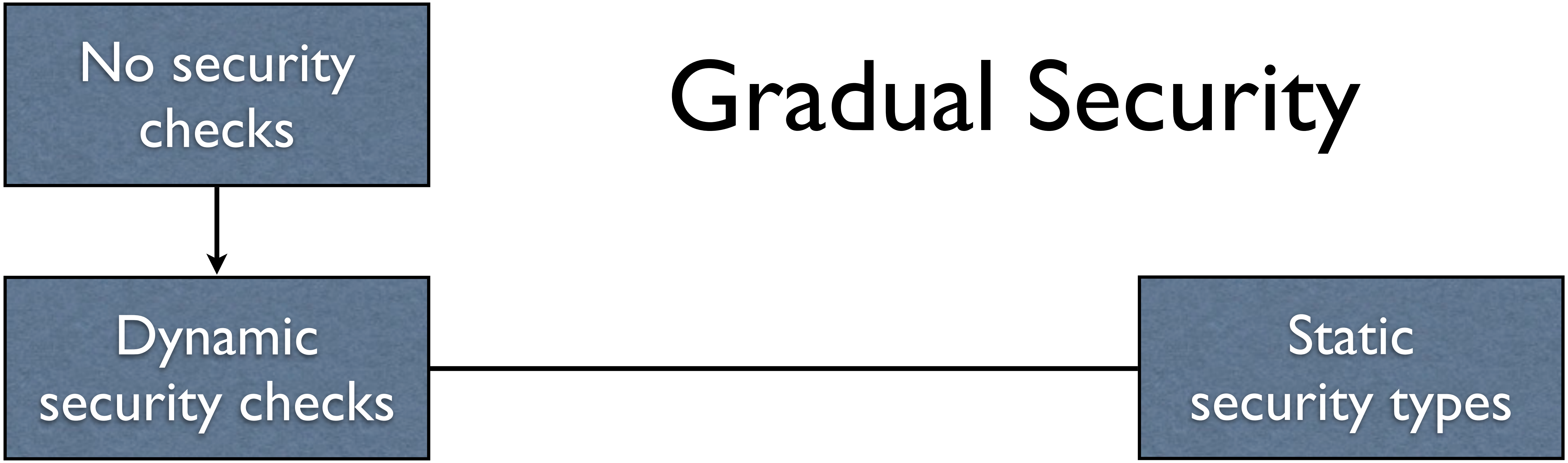
Siek and Taha (2006), Findler and Felleisen (2002), Wadler and Findler (2009), Ahmed et al. (2011), etc.

Gradual Typing

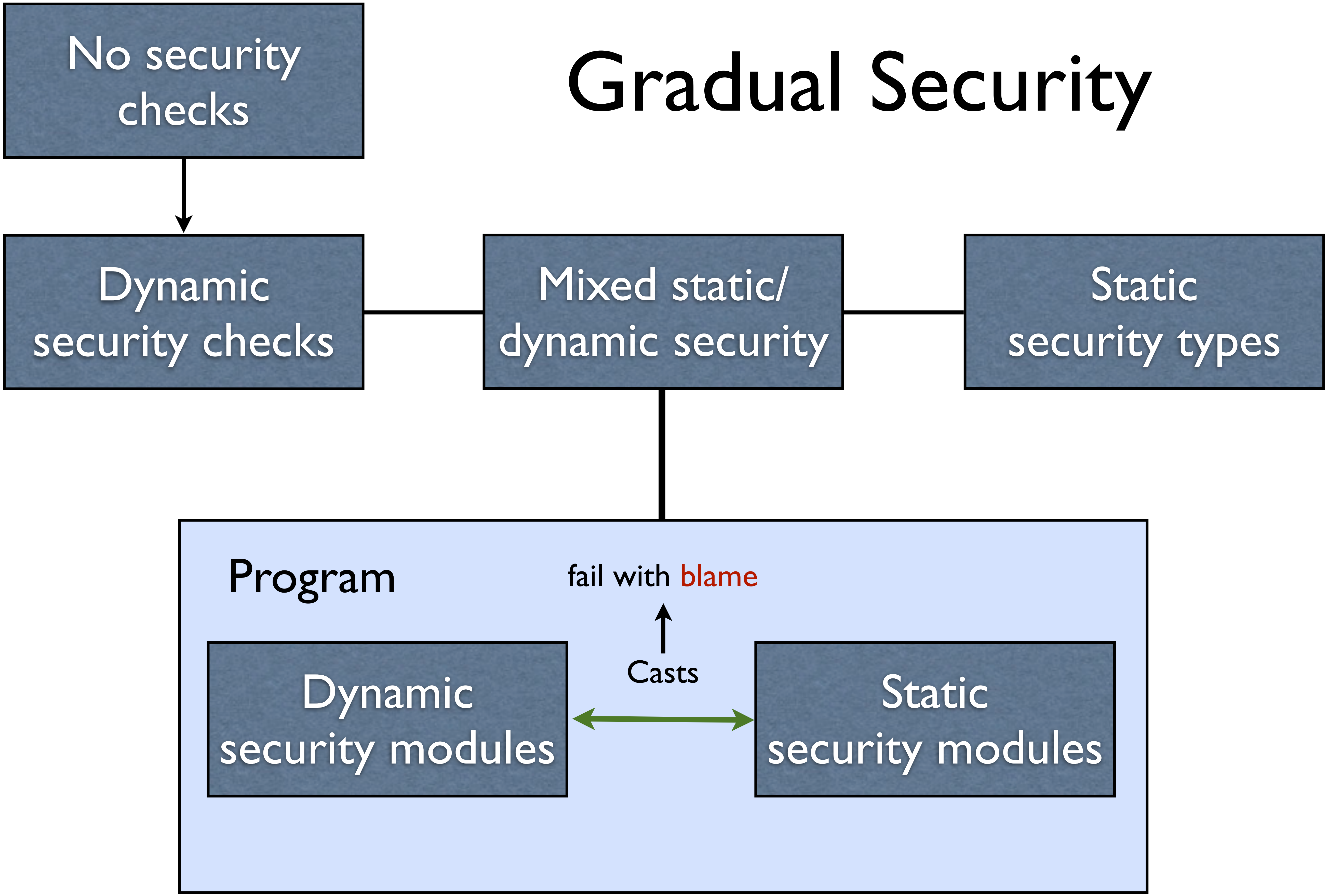


Siek and Taha (2006), Findler and Felleisen (2002), Wadler and Findler (2009), Ahmed et al. (2011), etc.

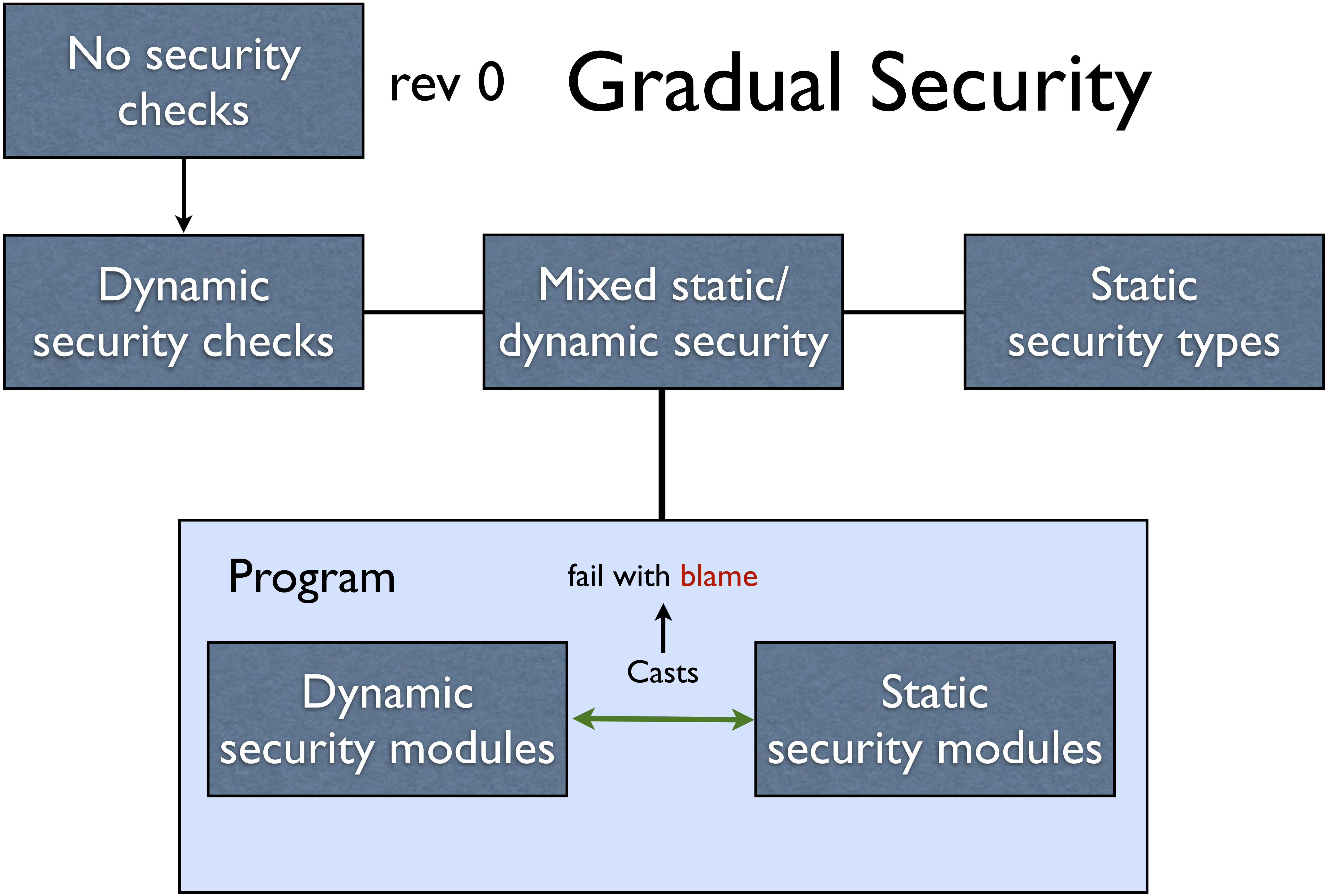
Gradual Security

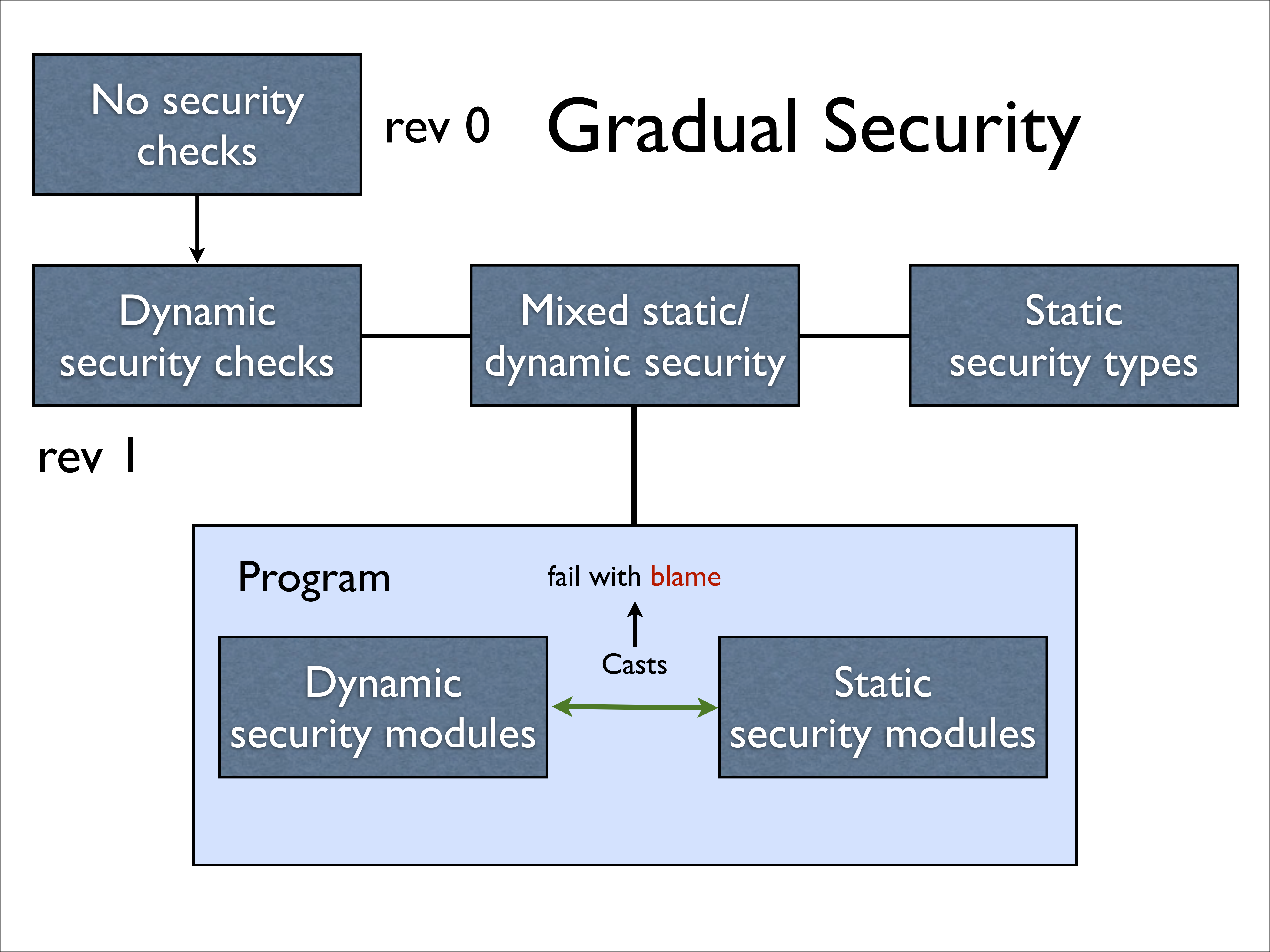


Gradual Security

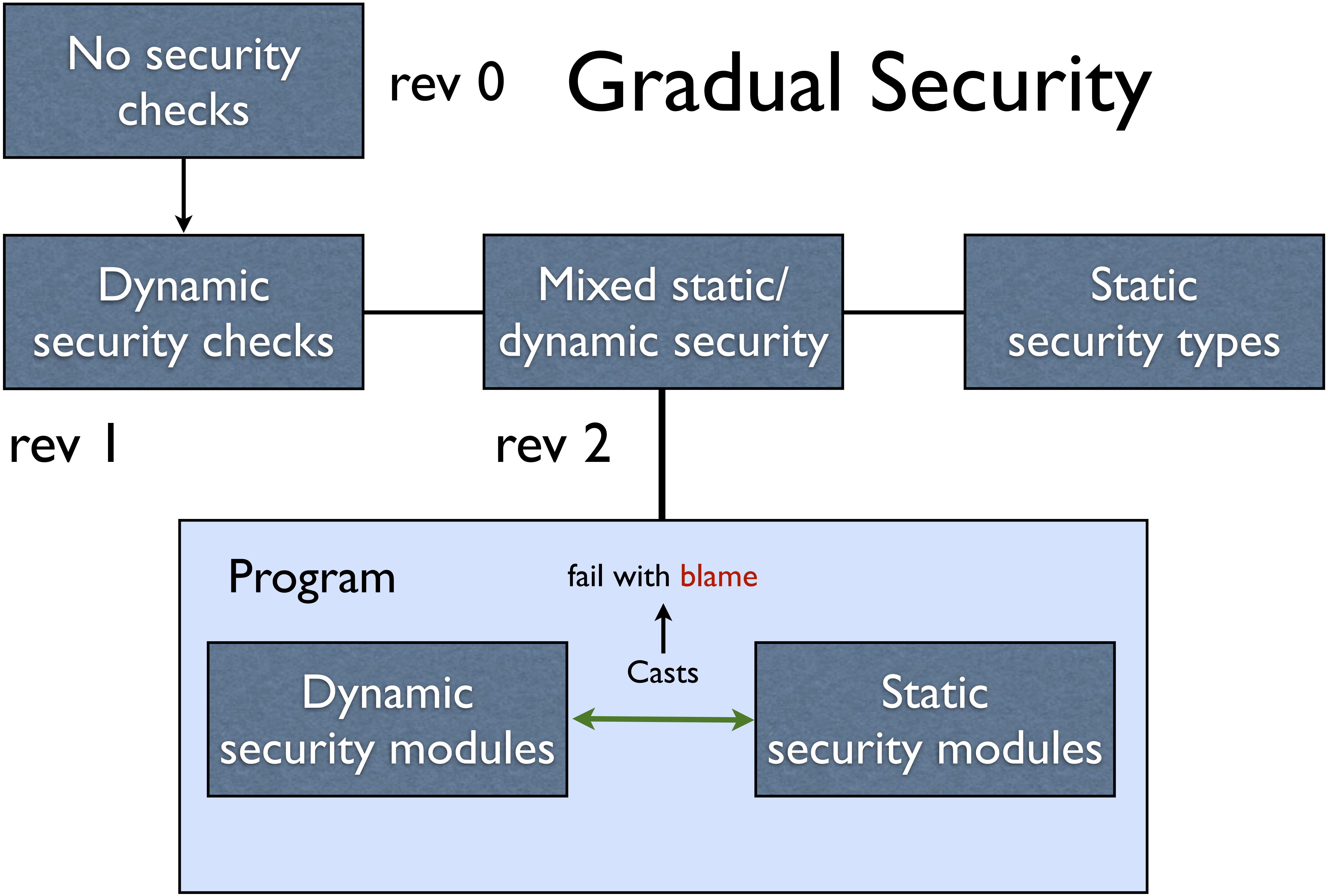


Gradual Security

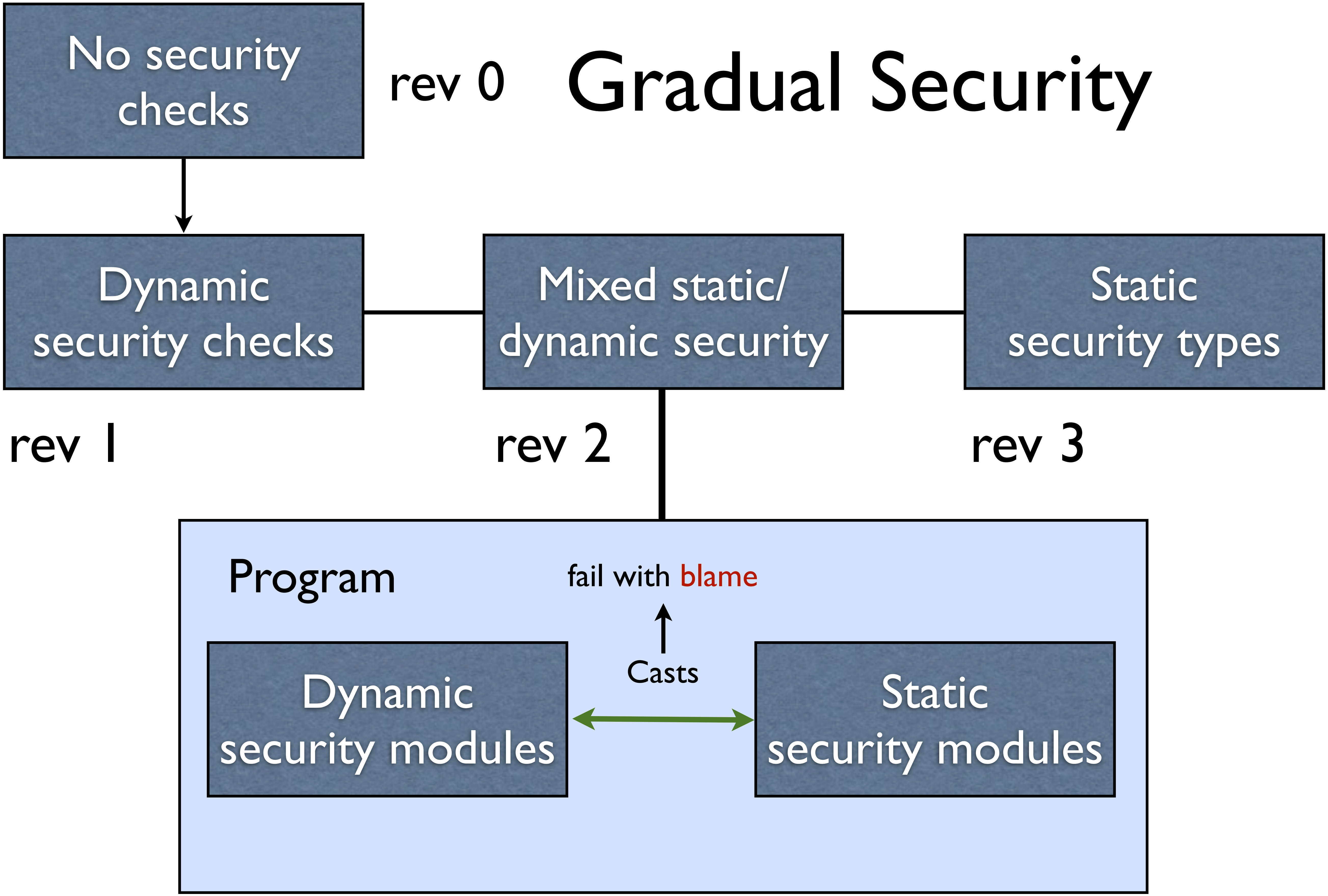



Gradual Security
No security checks
rev 0
Dynamic security checks
Mixed static/ dynamic security
Static security types
rev 1
Program
fail with blame
Casts
Dynamic security modules
Static security modules

Gradual Security



Gradual Security



λ_{gif}

A language for

gradual **i**nformation **f**low

The language λ_{gif}

Values (r):	$42, (\lambda x{:}\, A.\ t)$
Labeled Values (v):	$42^{H}, (\lambda x{:}\, A.\ t)^{L}$
Types (a, b):	$\texttt{Int}, \texttt{Bool}, A \rightarrow B$
Labeled Types (A, B):	$\texttt{Int}^{L}, \texttt{Bool}^{L}, (A \rightarrow B)^{H}$

The language λ_{gif}

Private types:
potentially private

$$\texttt{Int}^{H} = \{0^{L}, 0^{H}, 1^{L}, 1^{H}, \ldots\}$$

The language λ_{gif}

Private types:
potentially private

$$\texttt{Int}^{H} = \{0^{L}, 0^{H}, 1^{L}, 1^{H}, \ldots\}$$

Public types:
definitely public

$$\texttt{Int}^{L} = \{0^{L}, 1^{L}, 2^{L}, \ldots\}$$

The language λ_{gif}

Private types:
potentially private

$$\mathtt{Int}^{H} = \{0^{L}, 0^{H}, 1^{L}, 1^{H}, \ldots\}$$

Public types:
definitely public

$$\mathtt{Int}^{L} = \{0^{L}, 1^{L}, 2^{L}, \ldots\}$$

Subtypes

$$\mathtt{Int}^{L} <: \mathtt{Int}^{H}$$

The language λ_{gif}

Default labels are permissive

The language λ_{gif}

Default labels are permissive

$$42 = 42^{L}$$

The language λ_{gif}

Default labels are permissive

$$42 = 42^{L} \qquad \mathtt{Int} = \mathtt{Int}^{H}$$

The language λ_{gif}

Default labels are permissive

$$42 = 42^{L} \qquad \texttt{Int} = \texttt{Int}^{H}$$

$$42 : \texttt{Int}$$

The language λ_{gif}

Default labels are permissive

$$42 = 42^{L} \qquad \texttt{Int} = \texttt{Int}^{H}$$

$$42 : \texttt{Int}$$

$$42 = 42^{L} : \texttt{Int}^{L} <: \texttt{Int}^{H} = \texttt{Int}$$

Casting checks runtime labels

$$t : A \Rightarrow^{p} B$$

Syntax similar to "Blame For All" by Ahmed et al.

Casting checks runtime labels

$$t : A \Rightarrow^{p} B$$

$$42^{L} : \mathtt{Int}^{H} \Rightarrow^{p} \mathtt{Int}^{L} \longrightarrow 42^{L}$$

Syntax similar to "Blame For All" by Ahmed et al.

Casting checks runtime labels

$$t : A \Rightarrow^p B$$

$$42^{L} : \texttt{Int}^{H} \Rightarrow^p \texttt{Int}^{L} \longrightarrow 42^{L}$$

$$42^{H} : \texttt{Int}^{H} \Rightarrow^p \texttt{Int}^{L} \longrightarrow \mathit{blame}\ p$$

Syntax similar to "Blame For All" by Ahmed et al.

Higher-order casting: cast at fault

$$(fn) : (\texttt{Int}^{L} \rightarrow \texttt{Int}^{H}) \Rightarrow^{p} (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L})$$

$$\longrightarrow wrap\_fn$$

Higher-order casting: cast at fault

$$(fn) : (\mathtt{Int}^{L} \rightarrow \mathtt{Int}^{H}) \Rightarrow^{p} (\mathtt{Int}^{L} \rightarrow \mathtt{Int}^{L})$$

$$\longrightarrow wrap\_fn$$

$$fn = \lambda x{:}\,\mathtt{Int}^{L}.\ x + 1^{L}$$

$$(wrap\_fn)\ 42^{L} \longrightarrow 43^{L}$$

Higher-order casting: cast at fault

$$(fn) : (\texttt{Int}^{L} \to \texttt{Int}^{H}) \Rightarrow^{p} (\texttt{Int}^{L} \to \texttt{Int}^{L})$$

$$\longrightarrow wrap\_fn$$

$$fn = \lambda x{:}\,\texttt{Int}^{L}.\ x + 1^{L}$$

$$(wrap\_fn)\ 42^{L} \longrightarrow\ 43^{L}$$

$$fn = \lambda x{:}\,\texttt{Int}^{L}.\ x + 1^{H}$$

$$(wrap\_fn)\ 42^{L} \longrightarrow\ blame\ p$$

cast blamed

Higher-order casting: context at fault

$$(fn) : (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L}) \Rightarrow^{p} (\texttt{Int}^{H} \rightarrow \texttt{Int}^{L})$$

$$\rightarrow wrap\_fn$$

Higher-order casting: context at fault

$$(fn) : (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L}) \Rightarrow^{p} (\texttt{Int}^{H} \rightarrow \texttt{Int}^{L})$$

$$\longrightarrow wrap\_fn$$

$$(wrap\_fn)\ 42^{L} \longrightarrow 24^{L}$$

Higher-order casting: context at fault

$$(fn) : (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L}) \Rightarrow^{p} (\texttt{Int}^{H} \rightarrow \texttt{Int}^{L})$$

$$\rightarrow wrap\_fn$$

$$(wrap\_fn)\ 42^{L} \rightarrow 24^{L}$$

$$(wrap\_fn)\ 42^{H} \rightarrow blame\ \overline{p}$$

context blamed

Labeling $\Rrightarrow$ adds runtime labels

$$(58000^{L} : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H})$$

$$\longrightarrow\ 58000^{H}$$

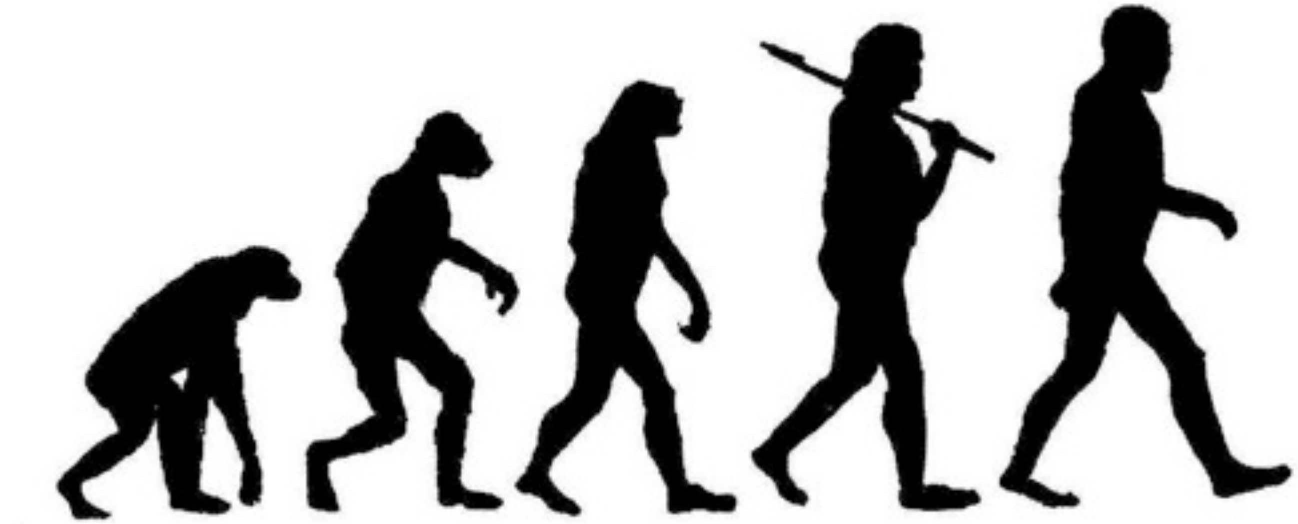
Labeling $\Rrightarrow$ adds runtime labels

$$(58000^{L} : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H})$$

$$\longrightarrow \; 58000^{H}$$

$$disk\_read : (\texttt{Int}^{L} \to \texttt{Int}^{L}) \Rrightarrow (\texttt{Int}^{L} \to \texttt{Int}^{H})$$

$$\longrightarrow \; wrap\_fn$$



Evolution Example

Rev 0 (no security)

$$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$$

Rev 0 (no security)

$$\texttt{let}\ intToString : \texttt{Int} \to \texttt{Str} = \ldots$$

$$\texttt{let}\ age : \texttt{Int} = 42$$

Rev 0 (no security)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000$ // confidential!

Rev 0 (no security)

$\texttt{let}\ intToString : \texttt{Int} \to \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000$ // confidential!

$\texttt{let}\ print : \texttt{Str} \to \texttt{Unit} = \lambda s{:}\,\texttt{Str}.\ \ldots$

Rev 0 (no security)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000$ // confidential!

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Rev 0 (no security)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000$ // confidential!

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\,\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Prints “58000”

Rev 0 (no security)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000$ // confidential!

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\,\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Prints "58000"

Add dynamic enforcement

Rev 1 (dynamic enforcement)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000 \colon \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Rev 1 (dynamic enforcement)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Rev 1 (dynamic enforcement)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Still prints "58000"^{H}

Rev 1 (dynamic enforcement)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\,\texttt{Str}.\ \ldots$

$print(intToString(salary))$

Still prints "58000"^{H}

Rev 1 (dynamic enforcement)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$

$\quad \texttt{let}\ s = (s : \texttt{Str}^{H} \Rrightarrow^{p} \texttt{Str}^{L})\ \texttt{in}\ \ldots$

$print(intToString(salary))$

Rev 1 (dynamic enforcement)

$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$

$\texttt{let}\ age : \texttt{Int} = 42$

$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$

$\quad \texttt{let}\ s = \boxed{(s : \texttt{Str}^{H} \Rightarrow^{p} \texttt{Str}^{L})}\ \texttt{in}\ \ldots$

$print(intToString(salary))$

Rev 1 (dynamic enforcement)

$$\texttt{let}\ intToString : \texttt{Int} \to \texttt{Str} = \ldots$$

$$\texttt{let}\ age : \texttt{Int} = 42$$
$$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rightarrow \texttt{Int}^{H}$$
$$\texttt{let}\ print : \texttt{Str} \to \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$$
$$\quad \texttt{let}\ s = \boxed{(s : \texttt{Str}^{H} \Rightarrow^{p} \texttt{Str}^{L})}\ \texttt{in}\ \ldots$$
$$print(intToString(salary))$$

Fails and blames p since Str^{H} can't be cast to Str^{L}

Rev 1 (dynamic enforcement)

$$\texttt{let}\ intToString : \texttt{Int} \rightarrow \texttt{Str} = \ldots$$

$$\texttt{let}\ age : \texttt{Int} = 42$$
$$\texttt{let}\ salary : \texttt{Int} = 58000 : \texttt{Int}^{L} \Rightarrow \texttt{Int}^{H}$$
$$\texttt{let}\ print : \texttt{Str} \rightarrow \texttt{Unit} = \lambda s{:}\texttt{Str}.\ \ldots$$
$$\texttt{let}\ s = (s : \texttt{Str}^{H} \Rightarrow^{p} \texttt{Str}^{L})\ \texttt{in}\ \ldots$$
$$print(intToString(salary))$$

✓ Fails and blames p since Str^{H} can't be cast to Str^{L}

Add static enforcement

Rev 2 (static enforcement)

$$\texttt{let}\ intToString : \texttt{Int}^{H} \to \texttt{Str}^{H} = \ldots$$

$$\texttt{let}\ age : \texttt{Int}^{L} = 42$$

$$\texttt{let}\ salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$$

$$\texttt{let}\ print : \texttt{Str}^{L} \to \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$$

$$print(intToString(salary))$$

Rev 2 (static enforcement)

$\texttt{let}\ intToString : \texttt{Int}^{H} \rightarrow \texttt{Str}^{H} = \ldots$

$\texttt{let}\ age : \texttt{Int}^{L} = 42$

$\texttt{let}\ salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str}^{L} \rightarrow \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$

$print(intToString(salary))$

intToString causes compile error

Rev 2 (static enforcement)

$$\texttt{let}\ intToString : \texttt{Int}^{\color{red}H} \rightarrow \texttt{Str}^{\boxed{\color{red}H}} = \ldots$$

$$\texttt{let}\ age : \texttt{Int}^{\color{blue}L} = 42$$

$$\texttt{let}\ salary : \texttt{Int}^{\color{red}H} = 58000 : \texttt{Int}^{\color{blue}L} \Rightarrow \texttt{Int}^{\color{red}H}$$

$$\texttt{let}\ print : \texttt{Str}^{\boxed{\color{blue}L}} \rightarrow \texttt{Unit}^{\color{blue}L} = \lambda s{:}\texttt{Str}^{\color{blue}L}.\ \ldots$$

$$print(intToString(salary))$$

intToString causes compile error

Rev 2 (static enforcement)

$\texttt{let}\ intToString : \texttt{Int}^{\color{red}H} \to \texttt{Str}^{\color{red}H} = \ldots$

$\texttt{let}\ age : \texttt{Int}^{\color{blue}L} = 42$
$\texttt{let}\ salary : \texttt{Int}^{\color{red}H} = 58000 \colon \texttt{Int}^{\color{blue}L} \Rrightarrow \texttt{Int}^{\color{red}H}$
$\texttt{let}\ print : \texttt{Str}^{\color{blue}L} \to \texttt{Unit}^{\color{blue}L} = \lambda s{:}\texttt{Str}^{\color{blue}L}.\ \ldots$
$\texttt{let}\ intToStringL : \texttt{Int}^{\color{blue}L} \to \texttt{Int}^{\color{blue}L} =$
$\quad intToString \colon (\texttt{Int}^{\color{red}H} \to \texttt{Int}^{\color{red}H}) \Rrightarrow^{p} (\texttt{Int}^{\color{blue}L} \to \texttt{Int}^{\color{blue}L})$
$print(intToStringL(salary))$

Rev 2 (static enforcement)

$\texttt{let}\ intToString : \texttt{Int}^{\color{red}H} \to \texttt{Str}^{\color{red}H} = \ldots$

$\texttt{let}\ age : \texttt{Int}^{\color{blue}L} = 42$

$\texttt{let}\ salary : \texttt{Int}^{\color{red}H} = 58000 \colon \texttt{Int}^{\color{blue}L} \Rrightarrow \texttt{Int}^{\color{red}H}$

$\texttt{let}\ print : \texttt{Str}^{\color{blue}L} \to \texttt{Unit}^{\color{blue}L} = \lambda s{:}\texttt{Str}^{\color{blue}L}.\ \ldots$

$\texttt{let}\ intToStringL : \texttt{Int}^{\color{blue}L} \to \texttt{Int}^{\color{blue}L} =$

$\quad intToString \colon (\texttt{Int}^{\color{red}H} \to \texttt{Int}^{\color{red}H}) \Rightarrow^{p} (\texttt{Int}^{\color{blue}L} \to \texttt{Int}^{\color{blue}L})$

$print(\boxed{intToStringL}(salary))$

Rev 2 (static enforcement)

$\texttt{let}\ intToString : \texttt{Int}^{H} \rightarrow \texttt{Str}^{H} = \ldots$

$\texttt{let}\ age : \texttt{Int}^{L} = 42$

$\texttt{let}\ salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str}^{L} \rightarrow \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$

$\texttt{let}\ intToStringL : \texttt{Int}^{L} \rightarrow \texttt{Int}^{L} =$

$\quad intToString : (\texttt{Int}^{H} \rightarrow \texttt{Int}^{H}) \Rrightarrow^{p} (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L})$

$print(\boxed{intToStringL}(salary))$

salary causes compile error

Rev 2 (static enforcement)

$\texttt{let } intToString : \texttt{Int}^H \rightarrow \texttt{Str}^H = \ldots$

$\texttt{let } age : \texttt{Int}^L = 42$

$\texttt{let } salary : \texttt{Int}^H = 58000 \colon \texttt{Int}^L \Rightarrow \texttt{Int}^H$

$\texttt{let } print : \texttt{Str}^L \rightarrow \texttt{Unit}^L = \lambda s{:}\texttt{Str}^L.\ \ldots$

$\texttt{let } intToStringL : \texttt{Int}^L \rightarrow \texttt{Int}^L =$

$\quad intToString \colon (\texttt{Int}^H \rightarrow \texttt{Int}^H) \Rightarrow^p (\texttt{Int}^L \rightarrow \texttt{Int}^L)$

$print(intToStringL(salary))$

✓ *salary* causes compile error

Rev 2 (static enforcement)

$\texttt{let}\ intToString : \texttt{Int}^{H} \rightarrow \texttt{Str}^{H} = \ldots$

$\texttt{let}\ age : \texttt{Int}^{L} = 42$

$\texttt{let}\ salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let}\ print : \texttt{Str}^{L} \longrightarrow \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$

$\texttt{let}\ intToStringL : \texttt{Int}^{L} \rightarrow \texttt{Int}^{L} =$

$\quad intToString : (\texttt{Int}^{H} \rightarrow \texttt{Int}^{H}) \Rightarrow^{p} (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L})$

$print(intToStringL(age))$

Rev 2 (static enforcement)

$$\texttt{let}\ intToString : \texttt{Int}^{H} \rightarrow \texttt{Str}^{H} = \ldots$$

$$\texttt{let}\ age : \texttt{Int}^{L} = 42$$
$$\texttt{let}\ salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$$
$$\texttt{let}\ print : \texttt{Str}^{L} \longrightarrow \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$$
$$\texttt{let}\ intToStringL : \texttt{Int}^{L} \longrightarrow \texttt{Int}^{L} =$$
$$\quad intToString : (\texttt{Int}^{H} \longrightarrow \texttt{Int}^{H}) \Rightarrow^{p} (\texttt{Int}^{L} \longrightarrow \texttt{Int}^{L})$$
$$print(intToStringL(age))$$

Rev 2 (static enforcement)

$\texttt{let } intToString : \texttt{Int}^{H} \rightarrow \texttt{Str}^{H} = \ldots$

$\texttt{let } age : \texttt{Int}^{L} = 42$

$\texttt{let } salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rrightarrow \texttt{Int}^{H}$

$\texttt{let } print : \texttt{Str}^{L} \longrightarrow \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$

$\texttt{let } intToStringL : \texttt{Int}^{L} \rightarrow \texttt{Int}^{L} =$

$\quad intToString : (\texttt{Int}^{H} \rightarrow \texttt{Int}^{H}) \Rightarrow^{p} (\texttt{Int}^{L} \rightarrow \texttt{Int}^{L})$

$print(intToStringL(age))$

Compiles successfully

Rev 2 (static enforcement)

$$\texttt{let}\ intToString : \texttt{Int}^{H} \to \texttt{Str}^{H} = \ldots$$

$$\texttt{let}\ age : \texttt{Int}^{L} = 42$$

$$\texttt{let}\ salary : \texttt{Int}^{H} = 58000 : \texttt{Int}^{L} \Rightarrow \texttt{Int}^{H}$$

$$\texttt{let}\ print : \texttt{Str}^{L} \to \texttt{Unit}^{L} = \lambda s{:}\texttt{Str}^{L}.\ \ldots$$

$$\texttt{let}\ intToStringL : \texttt{Int}^{L} \to \texttt{Int}^{L} =$$

$$intToString : (\texttt{Int}^{H} \to \texttt{Int}^{H}) \Rightarrow^{p} (\texttt{Int}^{L} \to \texttt{Int}^{L})$$

$$print(intToStringL(age))$$

✓ Compiles successfully

Safety Theorems

Theorem:
Termination Insensitive Non-Interference

Private inputs cannot affect public outputs

See paper for details

Subtyping

$L \sqsubseteq L \qquad L \sqsubseteq H \qquad H \sqsubseteq H$

Subtyping

$$L \sqsubseteq L \qquad L \sqsubseteq H \qquad H \sqsubseteq H$$

Subtype

$$\frac{l \sqsubseteq k}{\mathtt{Int}^l <: \mathtt{Int}^k} \qquad \frac{l \sqsubseteq k \quad A' <: A \quad B <: B'}{(A \rightarrow B)^l <: (A' \rightarrow B')^k}$$

Subtyping

$$L \sqsubseteq L \qquad L \sqsubseteq H \qquad H \sqsubseteq H$$

Subtype

$$\frac{l \sqsubseteq k}{\mathtt{Int}^l <: \mathtt{Int}^k} \qquad \frac{l \sqsubseteq k \quad A' <: A \quad B <: B'}{(A \to B)^l <: (A' \to B')^k}$$

Positive Subtype

$$\frac{l \sqsubseteq k}{\mathtt{Int}^l <:^+ \mathtt{Int}^k} \qquad \frac{l \sqsubseteq k \quad A' <:^- A \quad B <:^+ B'}{(A \to B)^l <:^+ (A' \to B')^k}$$

Subtyping

$$L \sqsubseteq L \qquad L \sqsubseteq H \qquad H \sqsubseteq H$$

Subtype

$$\frac{l \sqsubseteq k}{\texttt{Int}^l <: \texttt{Int}^k} \qquad \frac{l \sqsubseteq k \quad A' <: A \quad B <: B'}{(A \to B)^l <: (A' \to B')^k}$$

Positive Subtype

$$\frac{l \sqsubseteq k}{\texttt{Int}^l <:^+ \texttt{Int}^k} \qquad \frac{l \sqsubseteq k \quad A' <:^- A \quad B <:^+ B'}{(A \to B)^l <:^+ (A' \to B')^k}$$

Negative Subtype

$$\frac{k \sqsubseteq l}{\texttt{Int}^l <:^- \texttt{Int}^k} \qquad \frac{k \sqsubseteq l \quad A' <:^+ A \quad B <:^- B'}{(A \to B)^l <:^- (A' \to B')^k}$$

Blame Theorem

If two types are subtypes, casting cannot cause blame

Blame Theorem

If two types are subtypes, casting cannot cause blame

1. If $t\colon A \Rightarrow^p B$ and $A <: B$ then never blames p or $\overline{p}$

2. If $t\colon A \Rightarrow^p B$ and $A <:^+ B$ then never blames p

3. If $t\colon A \Rightarrow^p B$ and $A <:^- B$ then never blames $\overline{p}$

Conclusion

- Gradually evolve security
- From dynamic info-flow to static info-flow
- Provide language features to allow security evolution