COMPLETE MONITORS FOR GRADUAL TYPES

✦ Ben Greenman
at Northeastern

Matthias Felleisen
at Northeastern

Christos Dimoulas
at Northwestern

BU POPV
Fall 2020

Type soundness is not enough

Complete monitoring* is crucial
for **meaningful** gradual types

"Incomplete" monitoring provides a way to
measure the quality of blame errors

*from ESOP 2012

Sound? Dyn? Micro?

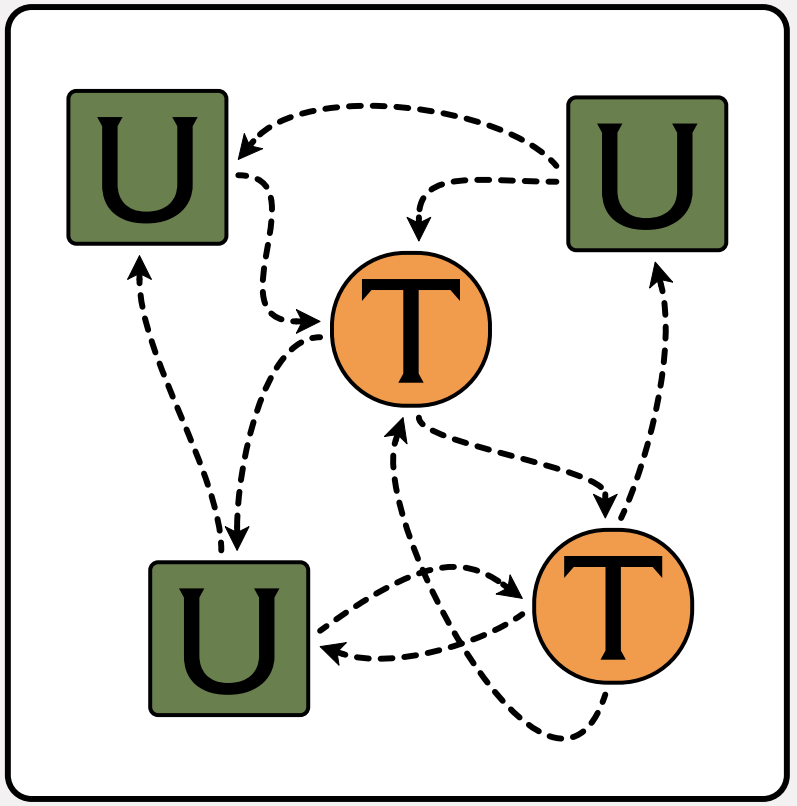
Mixed-Typed Language

Mixed-Typed Language

U = untyped code

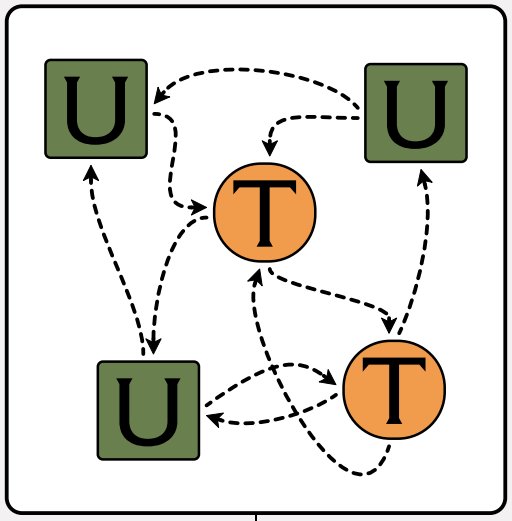
T = simply-typed code

(no 'Dynamic' type)



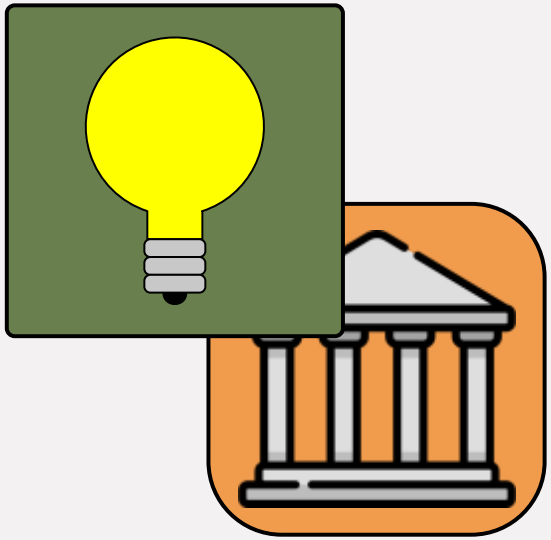
Untyped/Typed mix

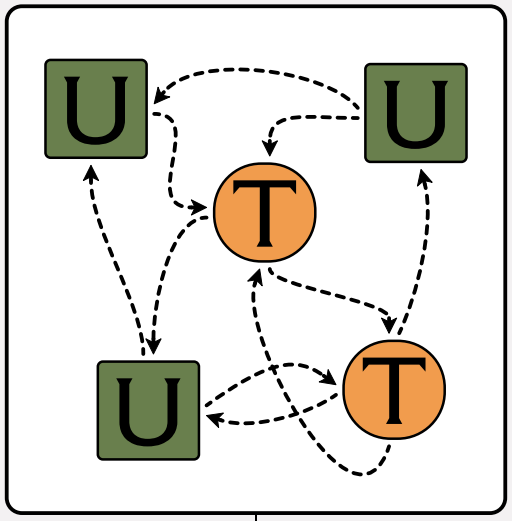
A Few Motivations



A Few Motivations

Prototyping

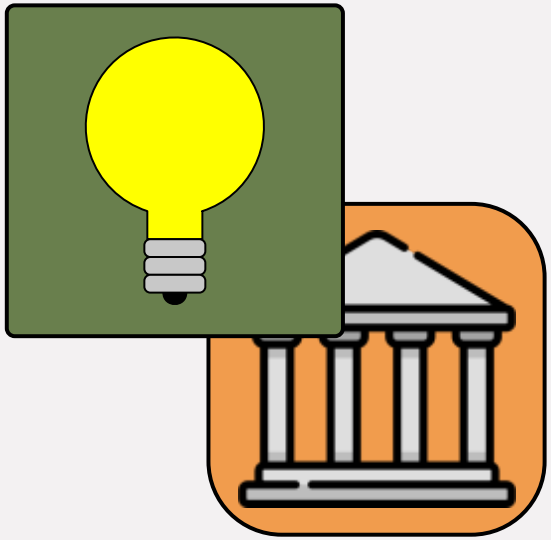




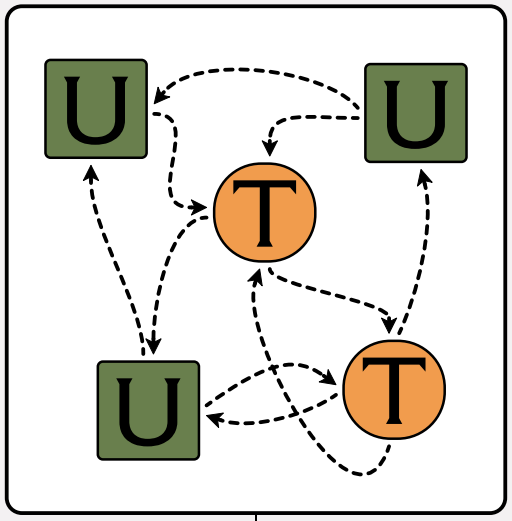
write untyped code,
rely on types

A Few Motivations

Prototyping



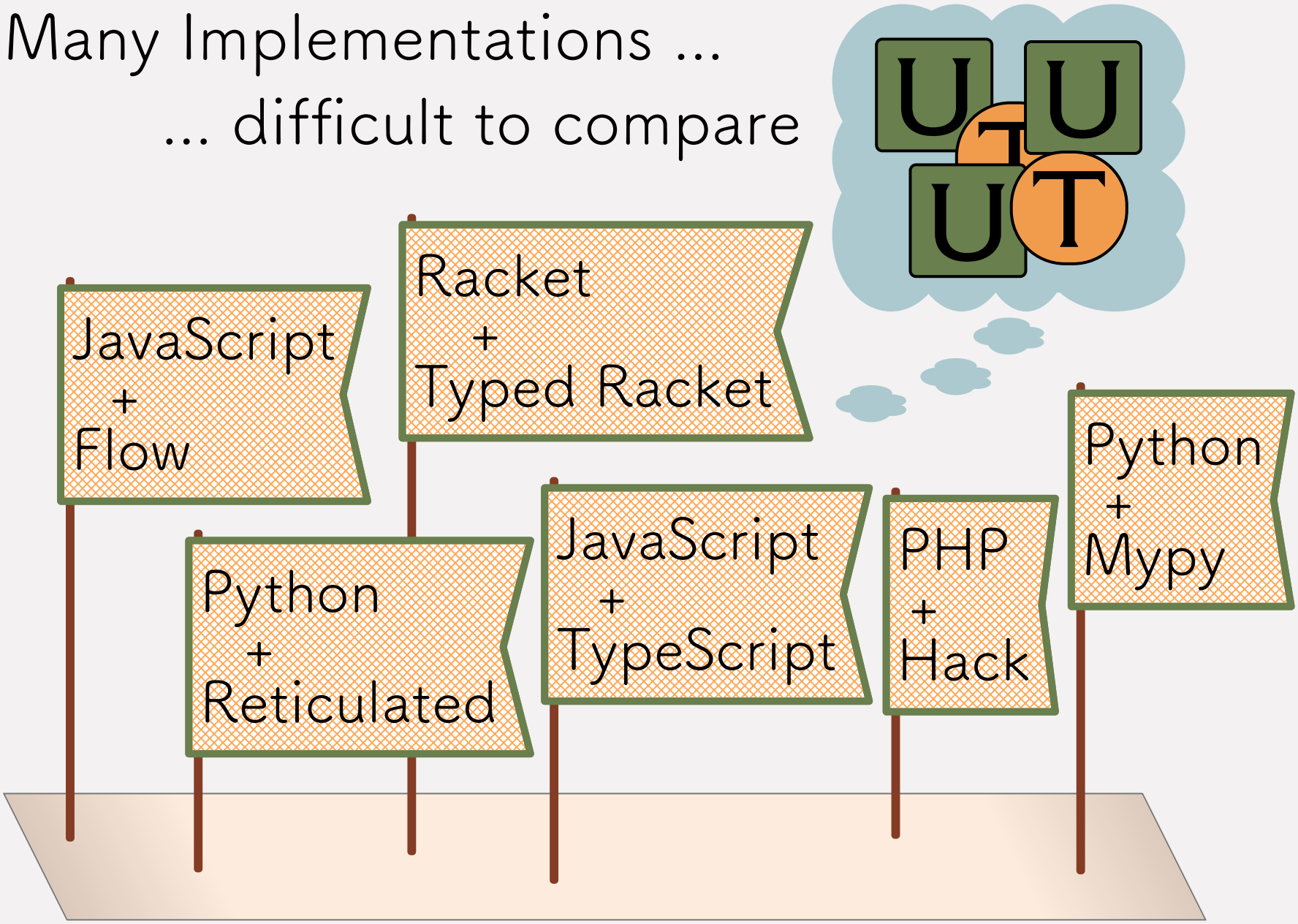
write untyped code,
rely on types



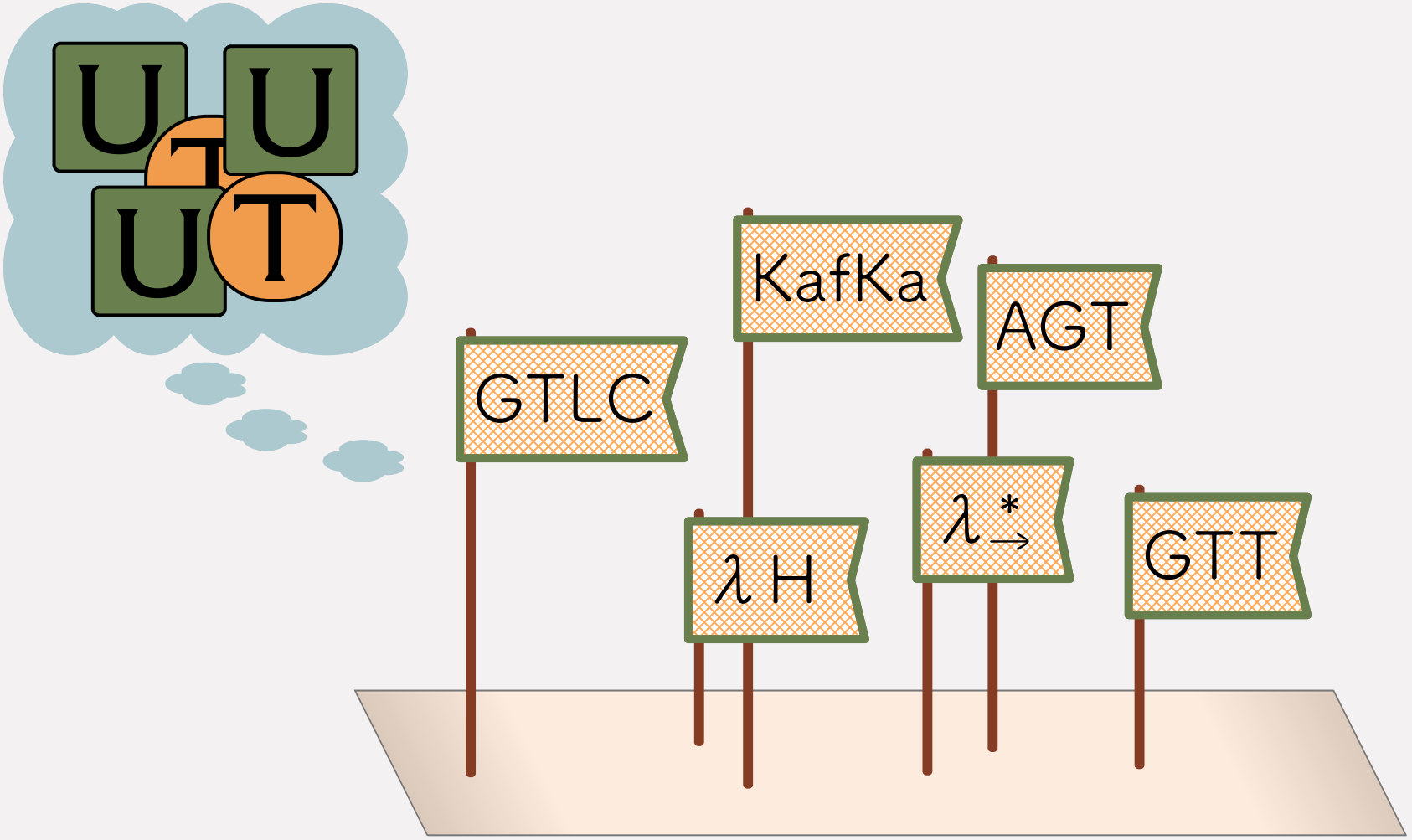
Re-Use

write typed code,
use old libraries

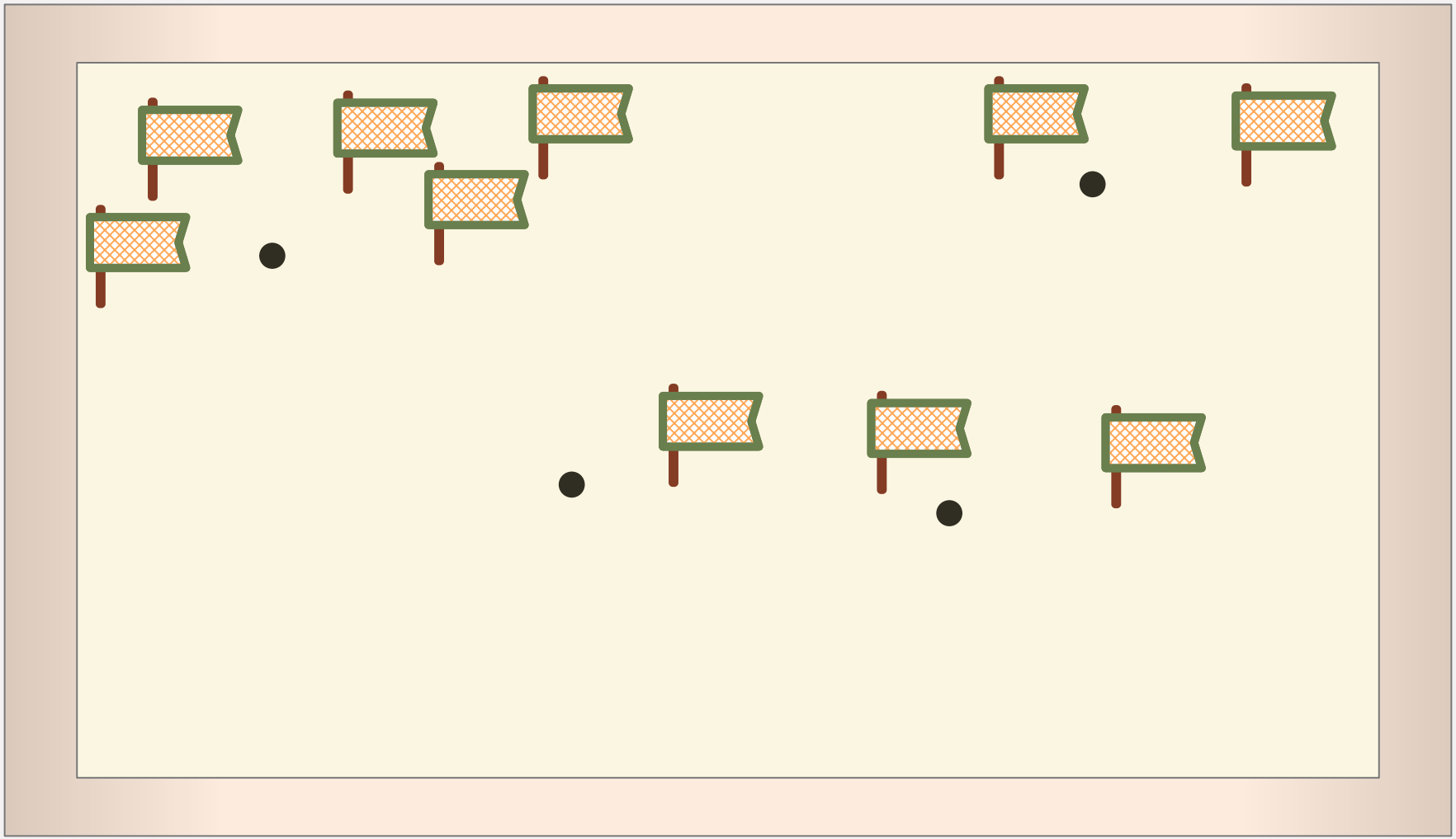
Many Implementations ...
... difficult to compare



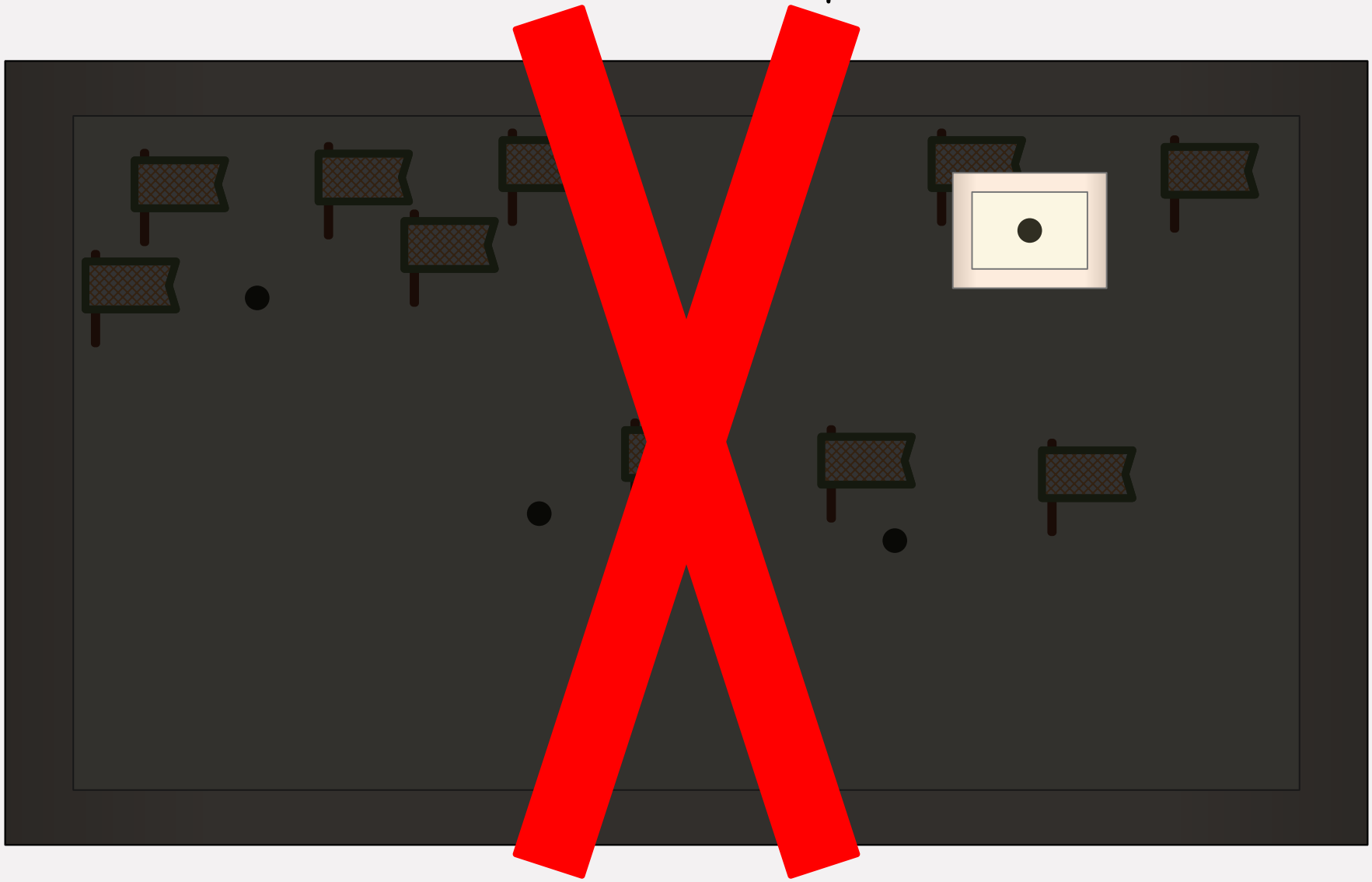
Many Models, too



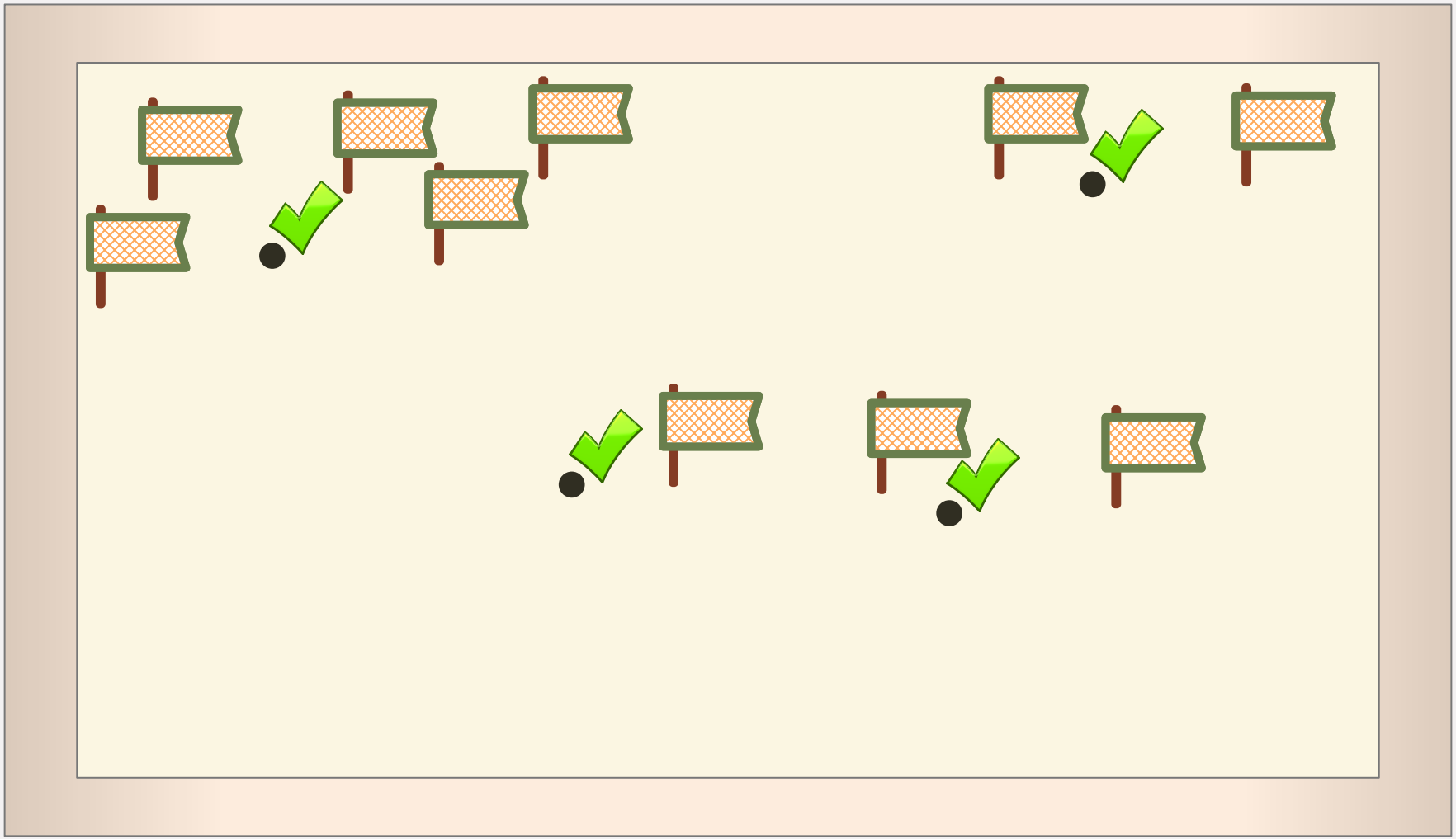
Goal: Characterize the Landscape



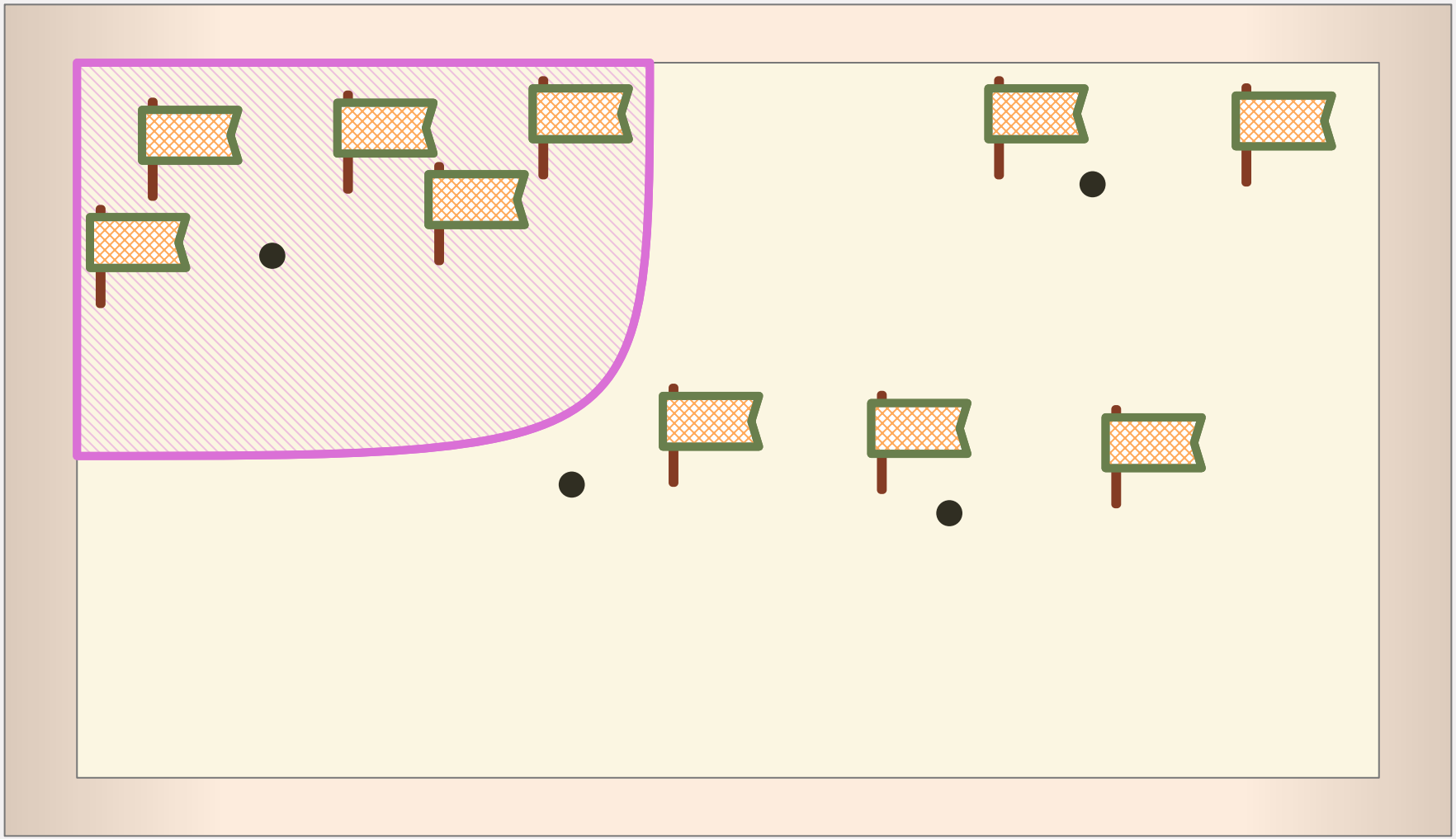
Non-Goal: Restrict Landscape



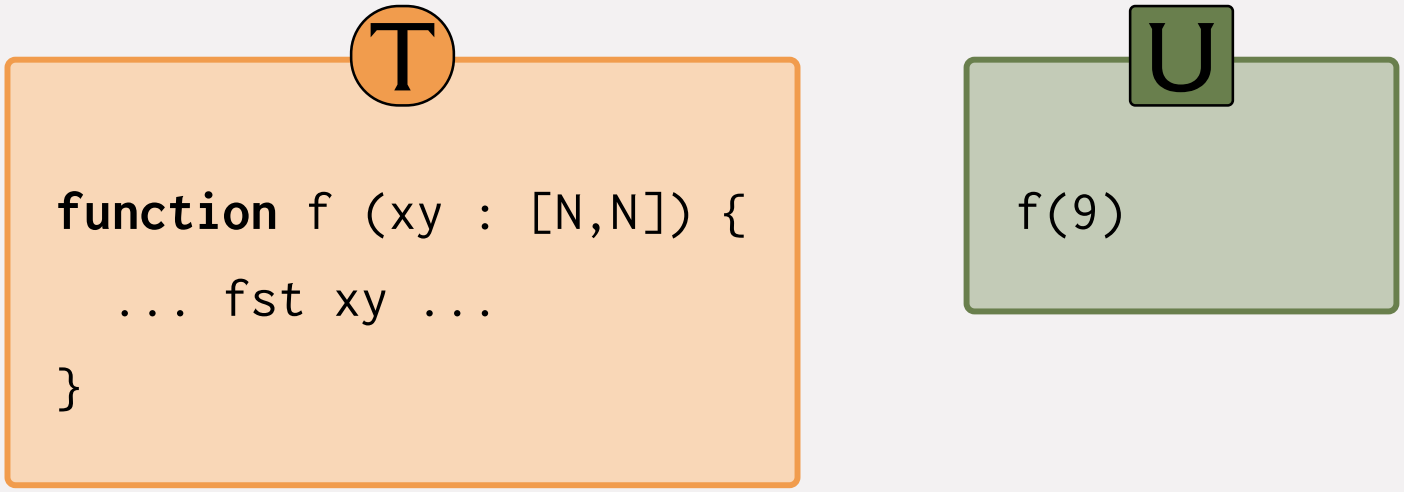
Want a Positive Characterization



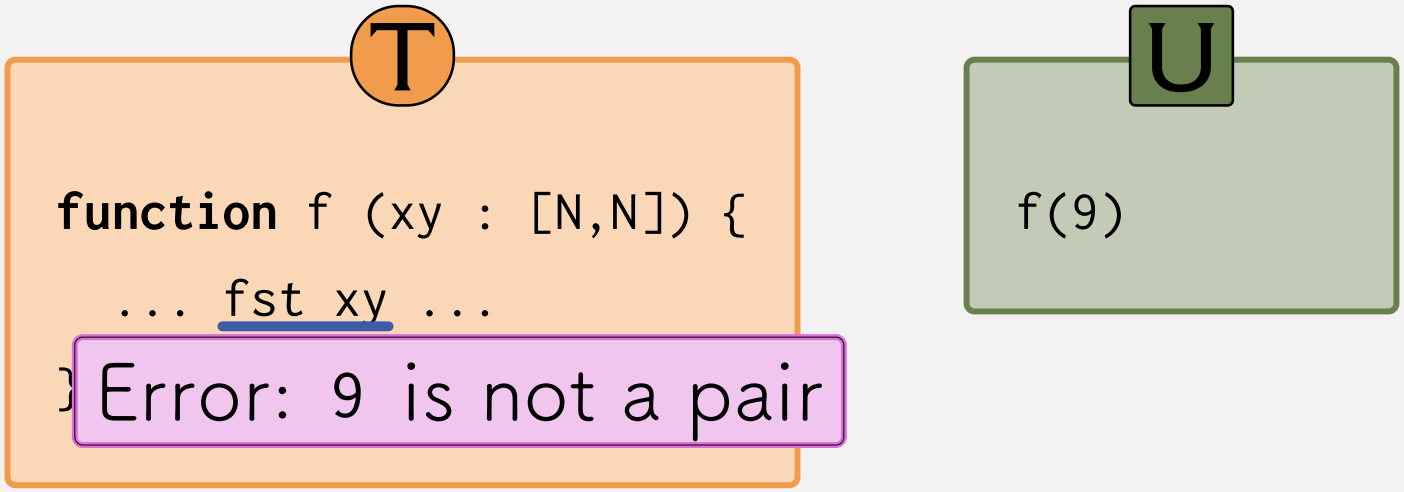
Optional Typing



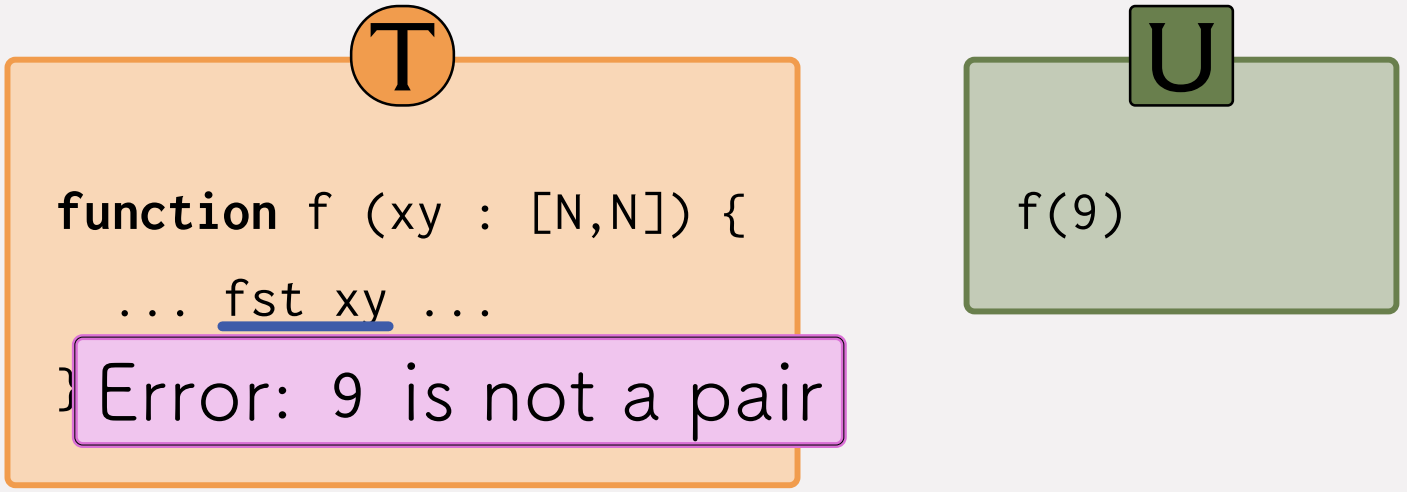
Example: Optional Typing



Example: Optional Typing

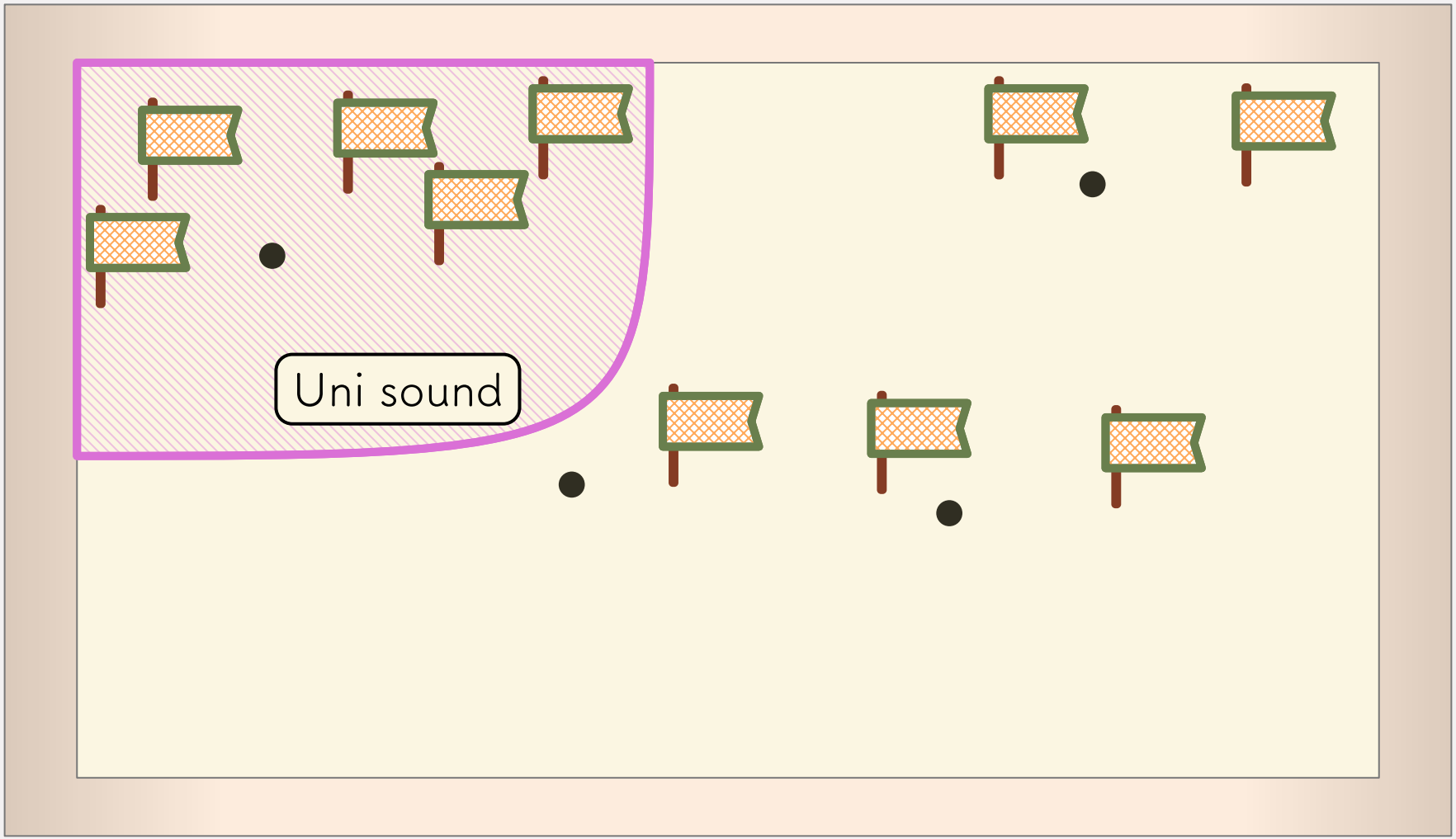


Example: Optional Typing

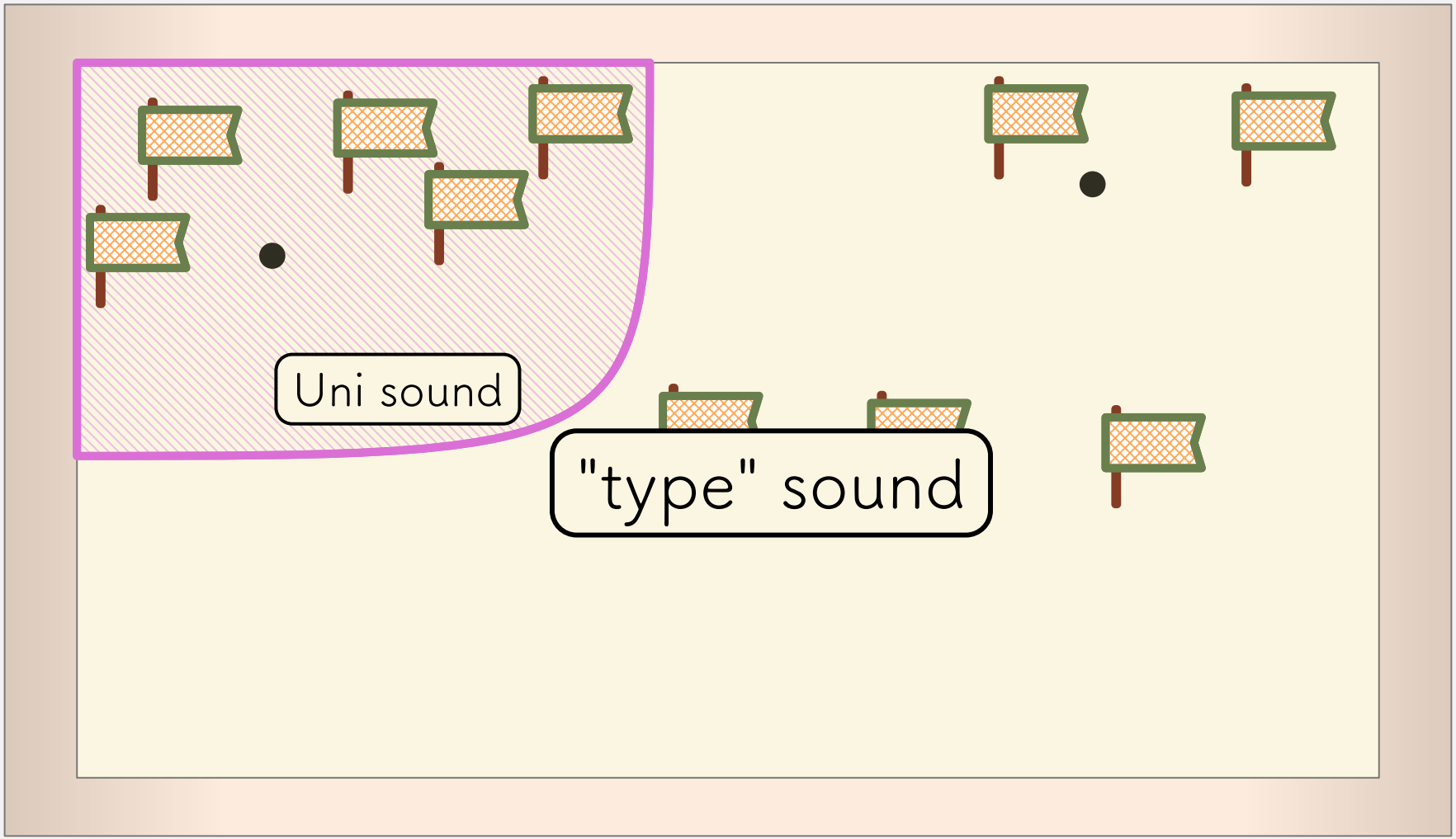


types are **meaningless** at run-time, and
cannot help debug a faulty program

= Does Not Preserve Types



= Does Not Preserve Types



ICFP '18 : A Spectrum of Type Soundness

ICFP '18 : A Spectrum of Type Soundness

Optional semantics

- types predict nothing

ICFP '18 : A Spectrum of Type Soundness

Optional semantics

- types predict nothing

Transient semantics

- types predict the top-level shape of values

ICFP '18 : A Spectrum of Type Soundness

Optional semantics

- types predict nothing

Transient semantics

- types predict the top-level shape of values

Natural semantics

- types predict the full behavior of values

ICFP '18 : A Spectrum of Type Soundness

Optional semantics

- types predict nothing

Transient semantics

- types predict the top-level shape of values

Natural semantics

- types predict the full behavior of values

Type Soundness

Definition (classic type soundness)**.**
If $\vdash e_0 : \tau_0$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash v_0 : \tau_0$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

Type Soundness

Definition (classic type soundness)**.**
If $\vdash e_0 : \tau_0$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash v_0 : \tau_0$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

Definition (untyped soundness)**.**
If $\vdash e_0 : \mathcal{U}$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash v_0 : \mathcal{U}$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

Type Soundness

Definition (classic type soundness).
If $\vdash e_0 : \tau_0$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash v_0 : \tau_0$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

Definition (untyped soundness).
If $\vdash e_0 : \mathcal{U}$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash v_0 : \mathcal{U}$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

Definition (F-type soundness).
If $\vdash e_0 : \mathsf{T}$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash_{\mathsf{F}} v_0 : \mathsf{F}(\mathsf{T})$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

F = 0 = always untyped

F = s = type-tags only

F = 1 = full types

ICFP '18 : A Spectrum of Type Soundness

Optional semantics

Uni sound

- types predict nothing

Transient semantics

⌊T⌋ sound

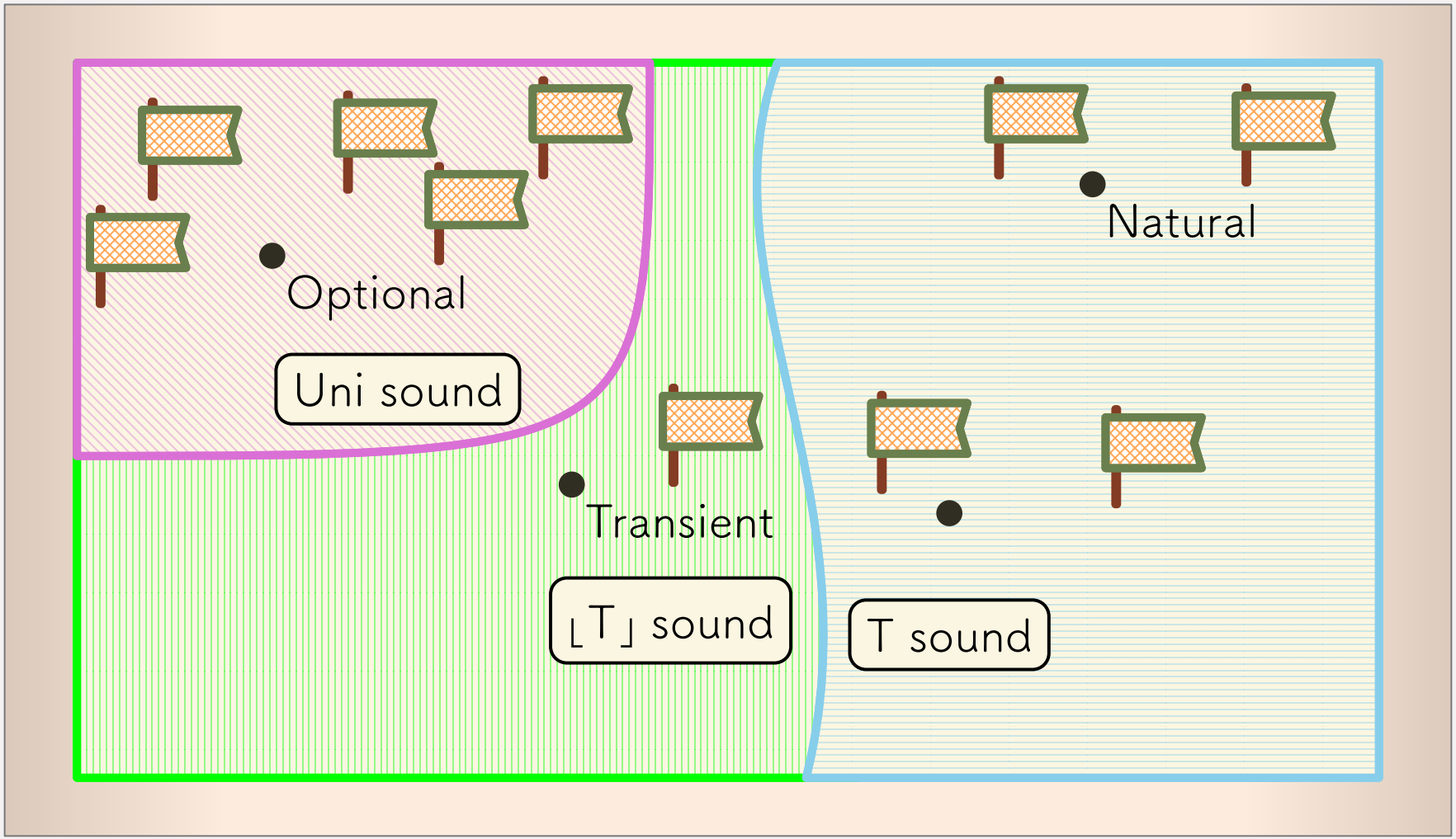
- types predict the top-level shape of values

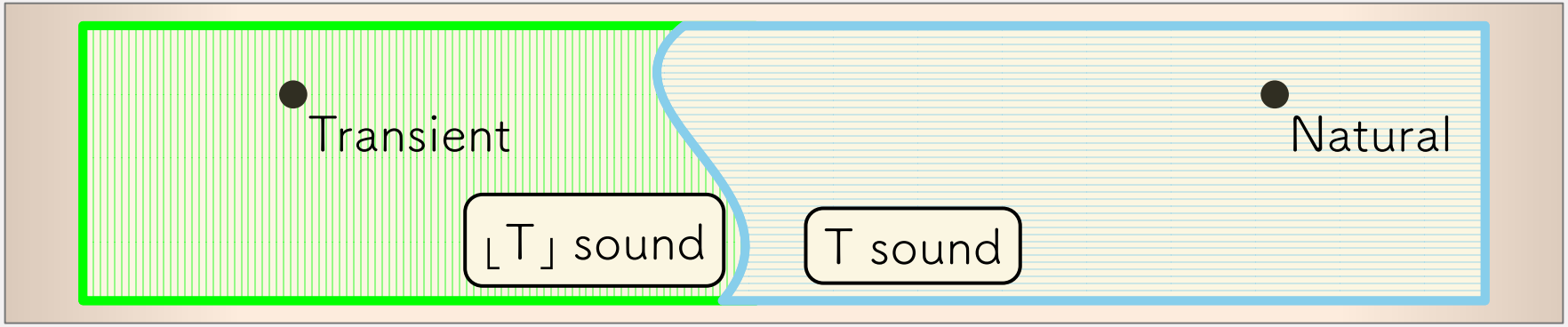
Natural semantics

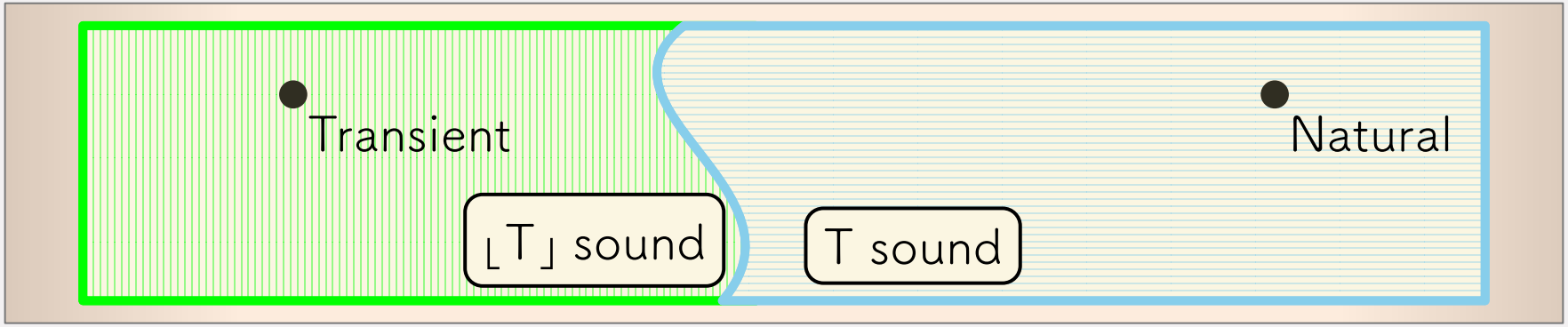
T sound

- types predict the full behavior of values

ICFP '18 : A Spectrum of Type Soundness




Transient
Natural
「T」sound
T sound



Transient semantics

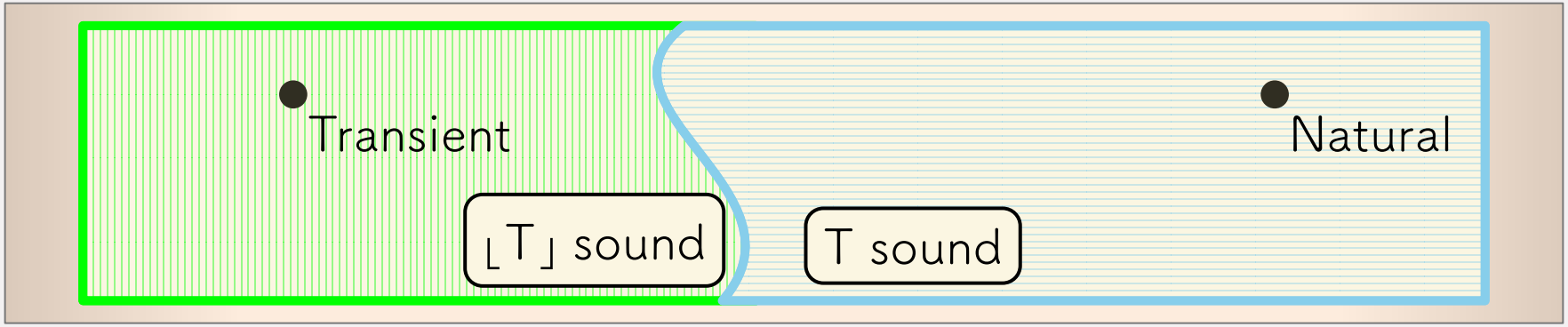
⌊T⌋ sound

- types predict the top-level shape of values

Natural semantics

T sound

- types predict the full behavior of values

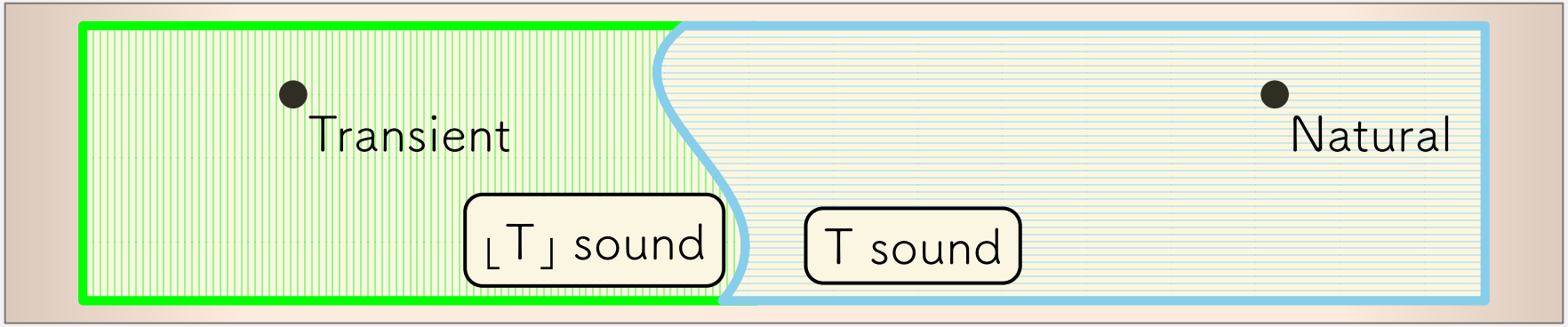


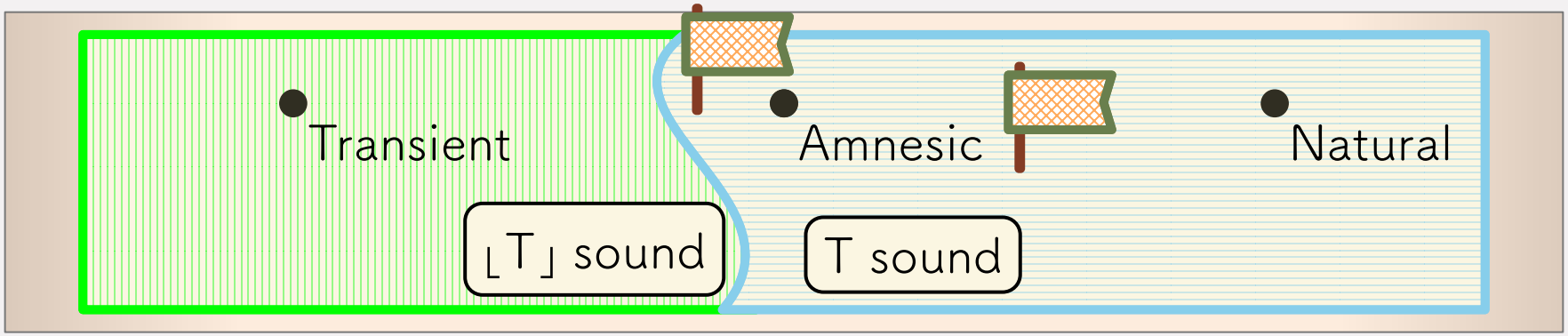
Transient semantics ⌊T⌋ sound

- types predict the top-level shape of values
- enforced by **tag checks**

Natural semantics T sound

- types predict the full behavior of values
- enforced by higher-order **wrappers**


Transient
Natural
「T」sound
T sound



OOPSLA '19

Amnesic semantics

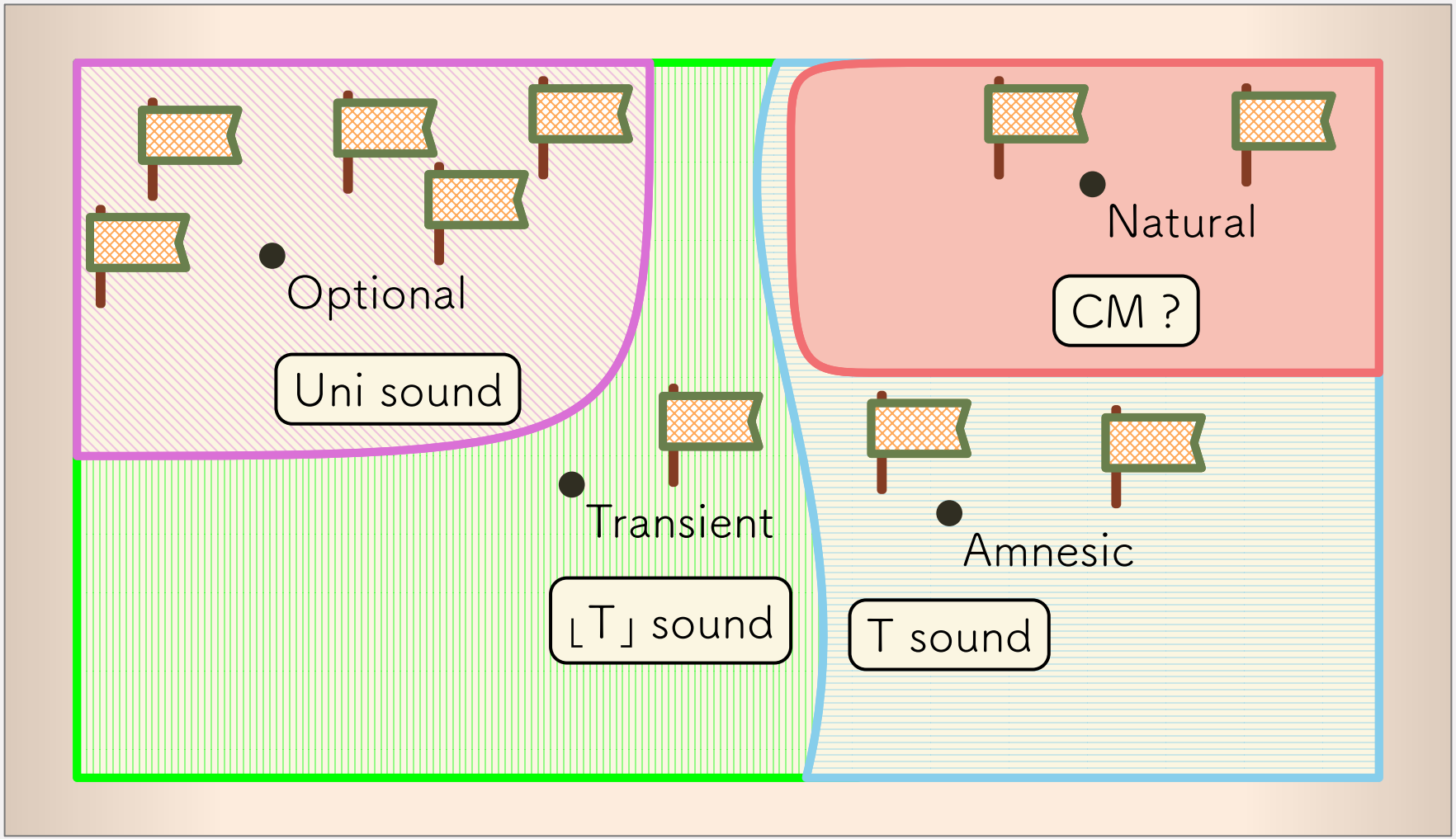
T sound

- enforce **tag checks** ⌊T⌋ with higher-order **wrappers**
- same behavior as **Transient**
- same type soundness as **Natural**

Greenberg POPL '15

Castagna, Lanvin ICFP '17

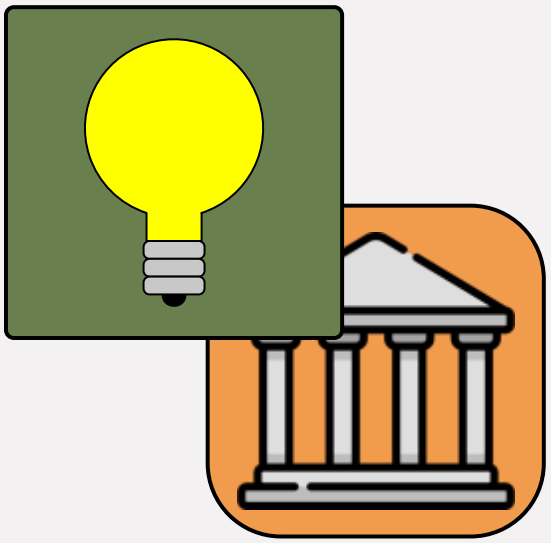
Type Soundness is Not Enough



Example: Transient/Amnesic vs. Natural

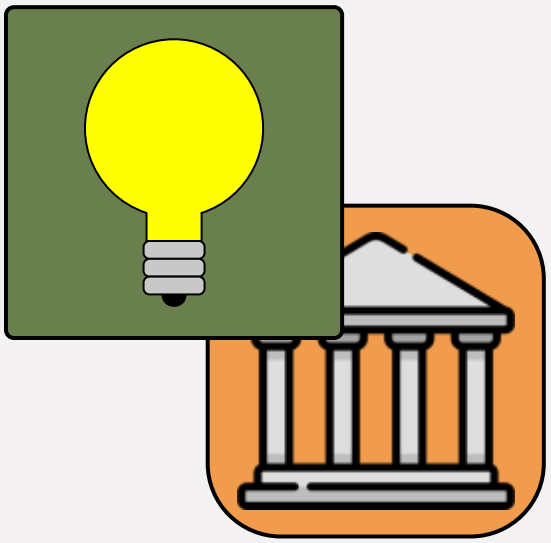
Example: Transient/Amnesic vs. Natural

Prototyping



Example: Transient/Amnesic vs. Natural

Prototyping

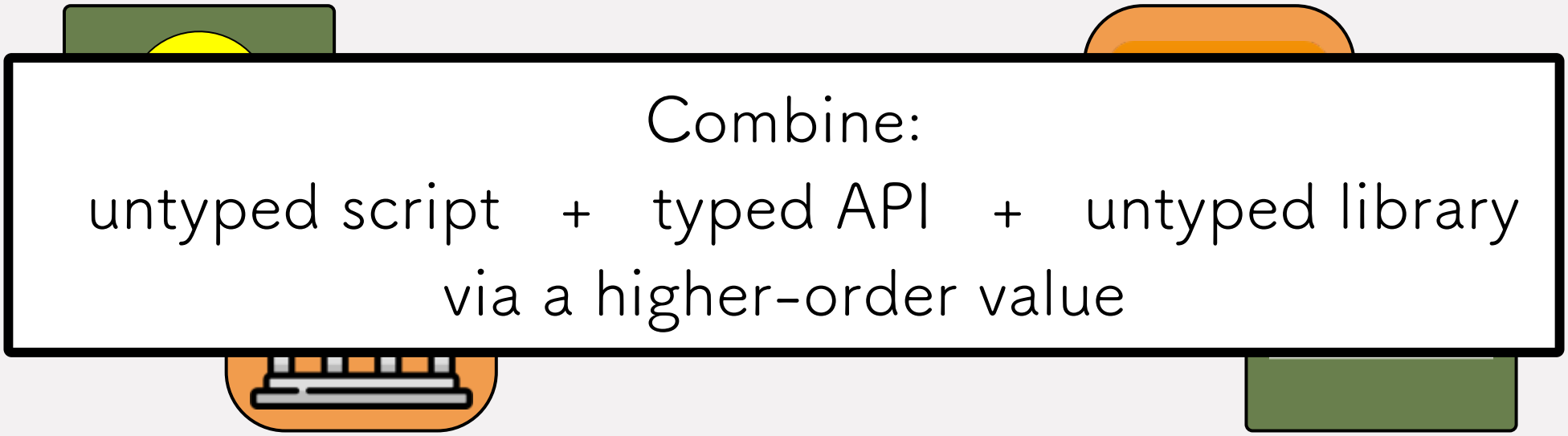


Library Re-Use

Example: Transient/Amnesic vs. Natural

Prototyping

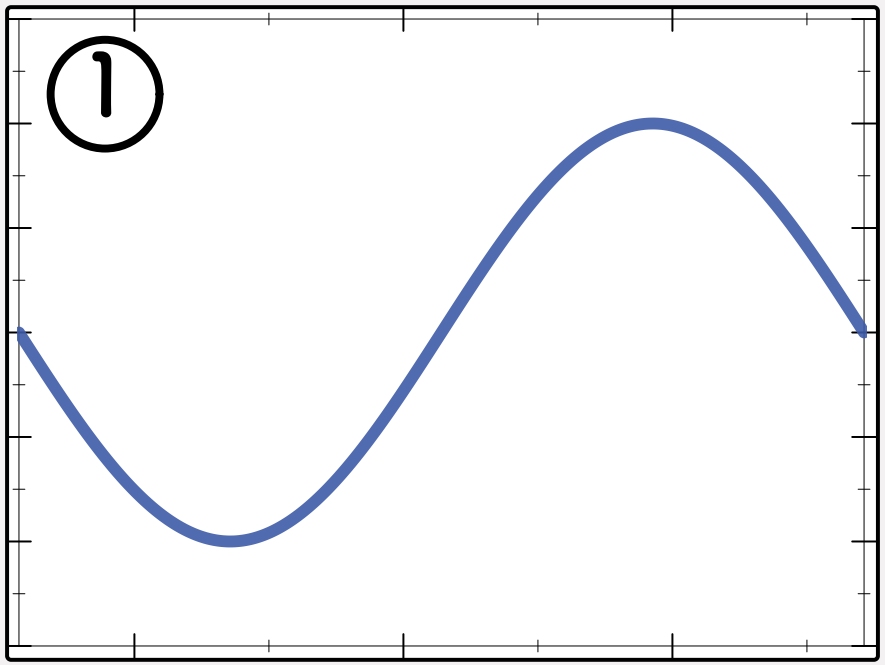
Library Re-Use



Example: Transient/Amnesic vs. Natural

1. plot data
2. listen for a click
3. draw an image

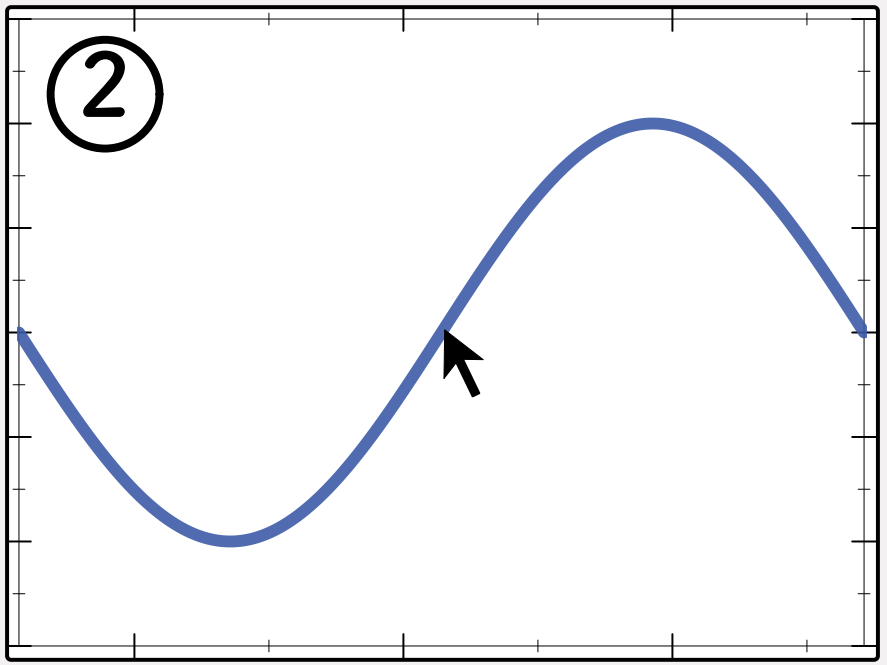
Clickable Plot



Example: Transient/Amnesic vs. Natural

1. plot data
2. listen for a click
3. draw an image

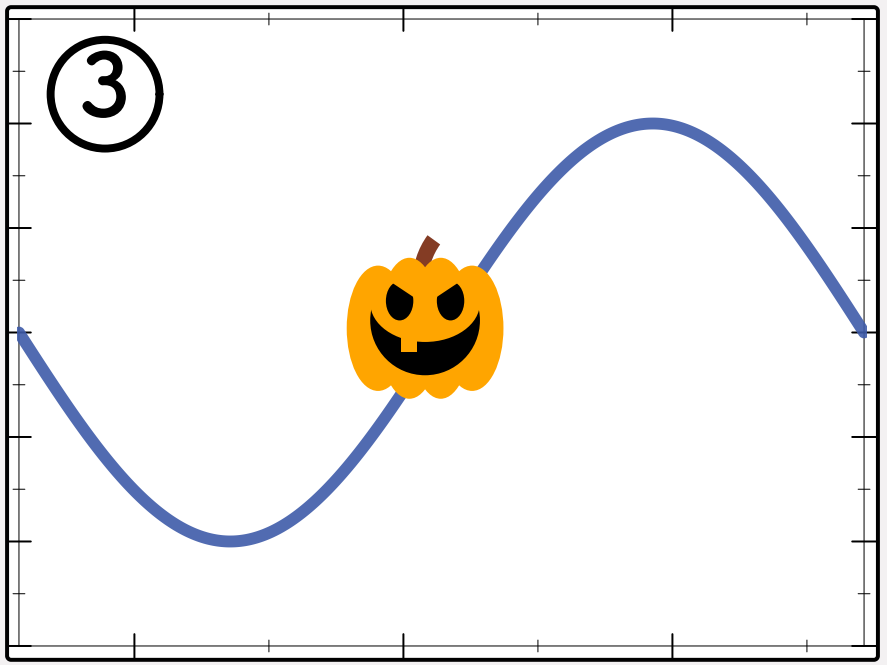
Clickable Plot



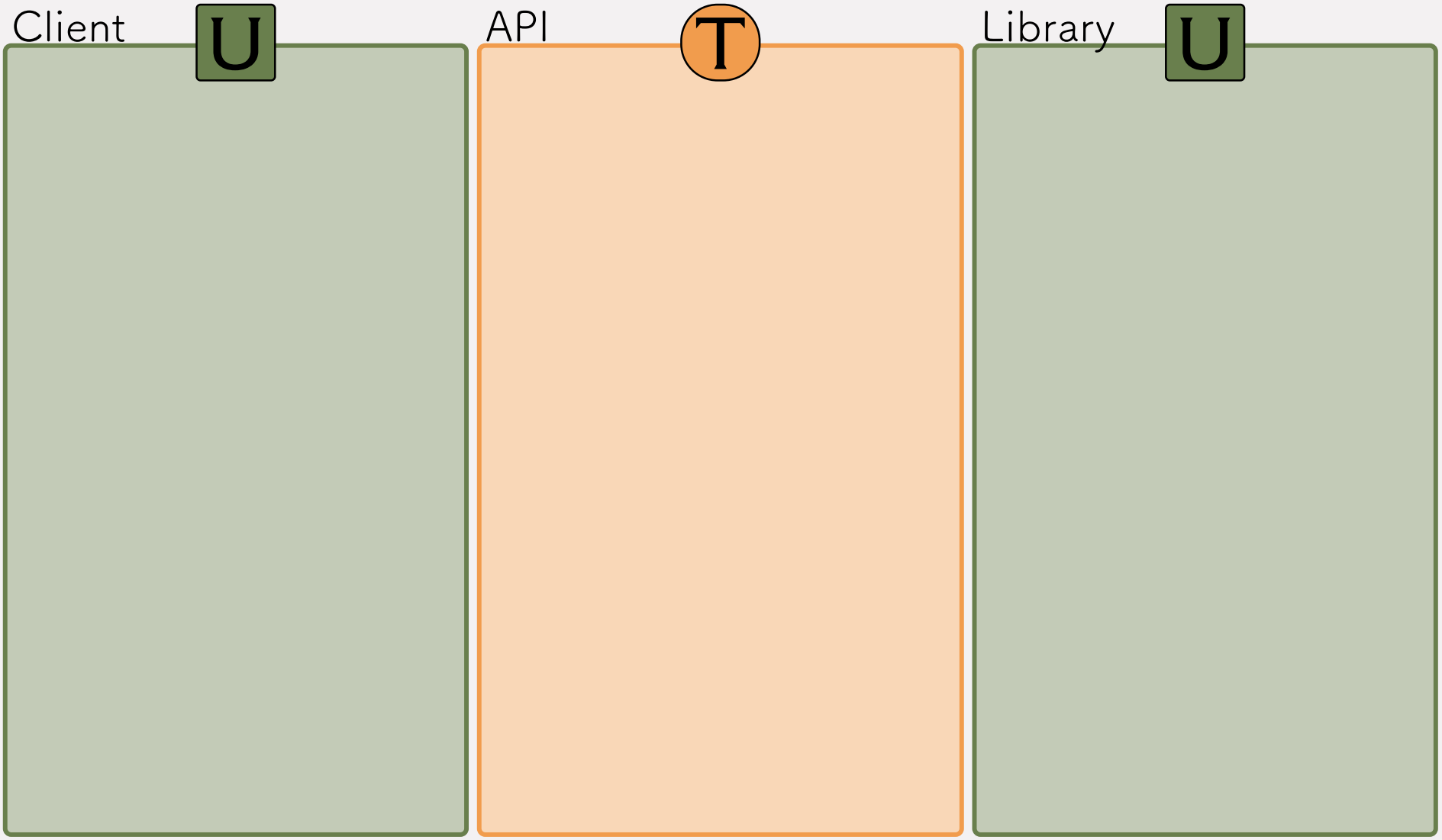
Example: Transient/Amnesic vs. Natural

1. plot data
2. listen for a click
3. draw an image

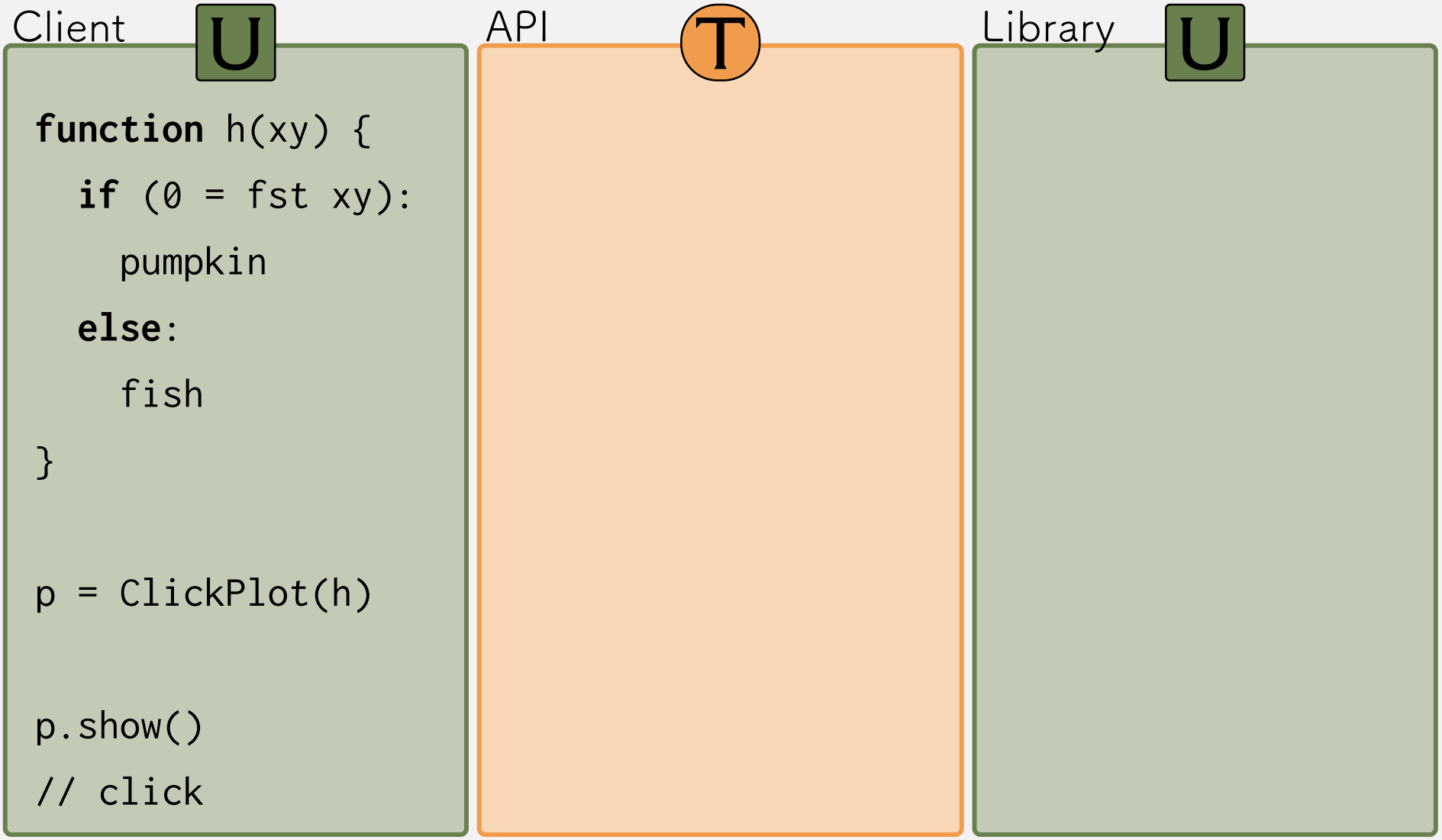
Clickable Plot



Example: interactive plot



Example: interactive plot



Example: interactive plot

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

API T

Library U

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

Example: interactive plot

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

API T

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

Library U

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

Example: interactive plot

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

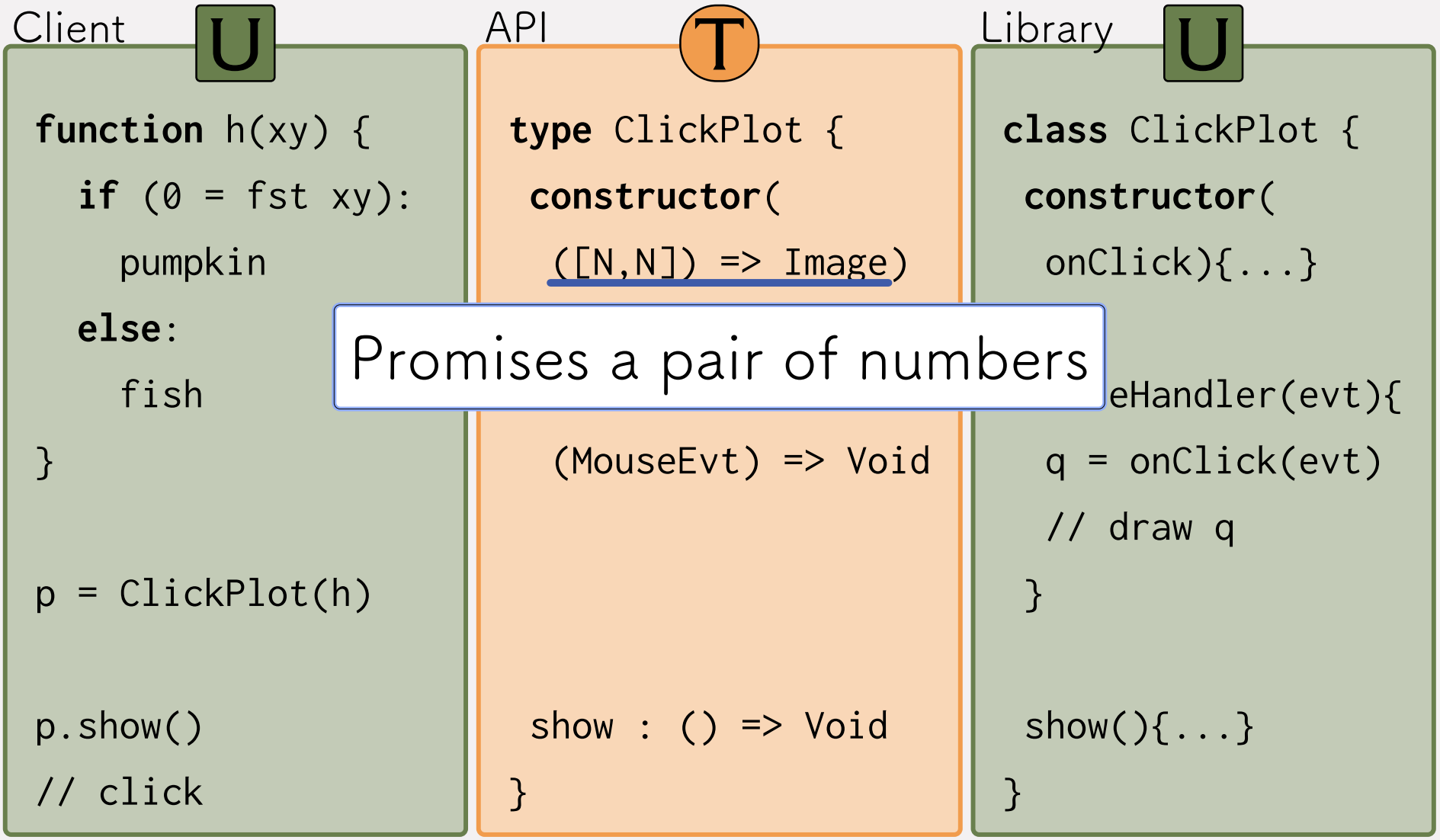
API T

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

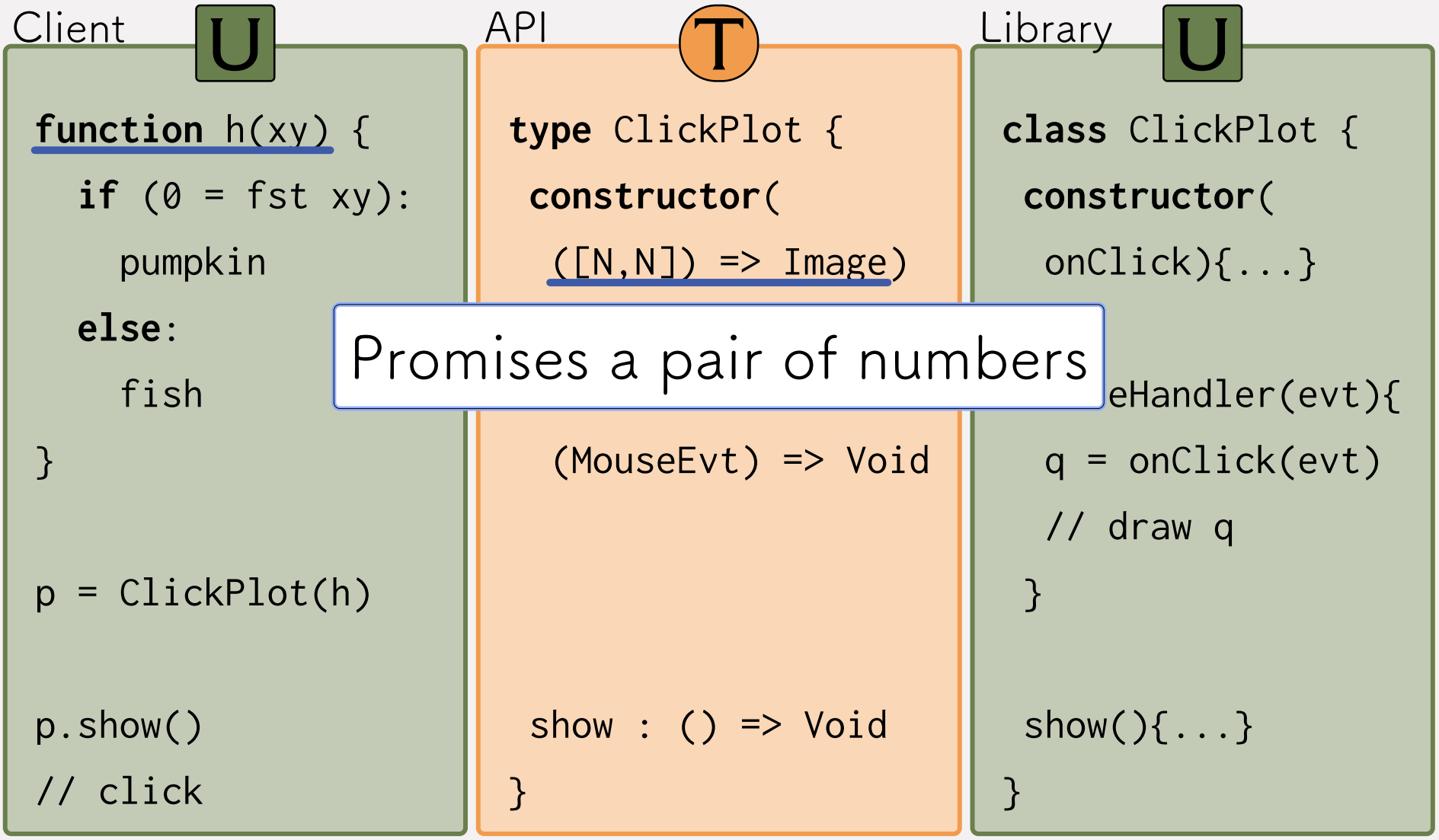
Library U

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

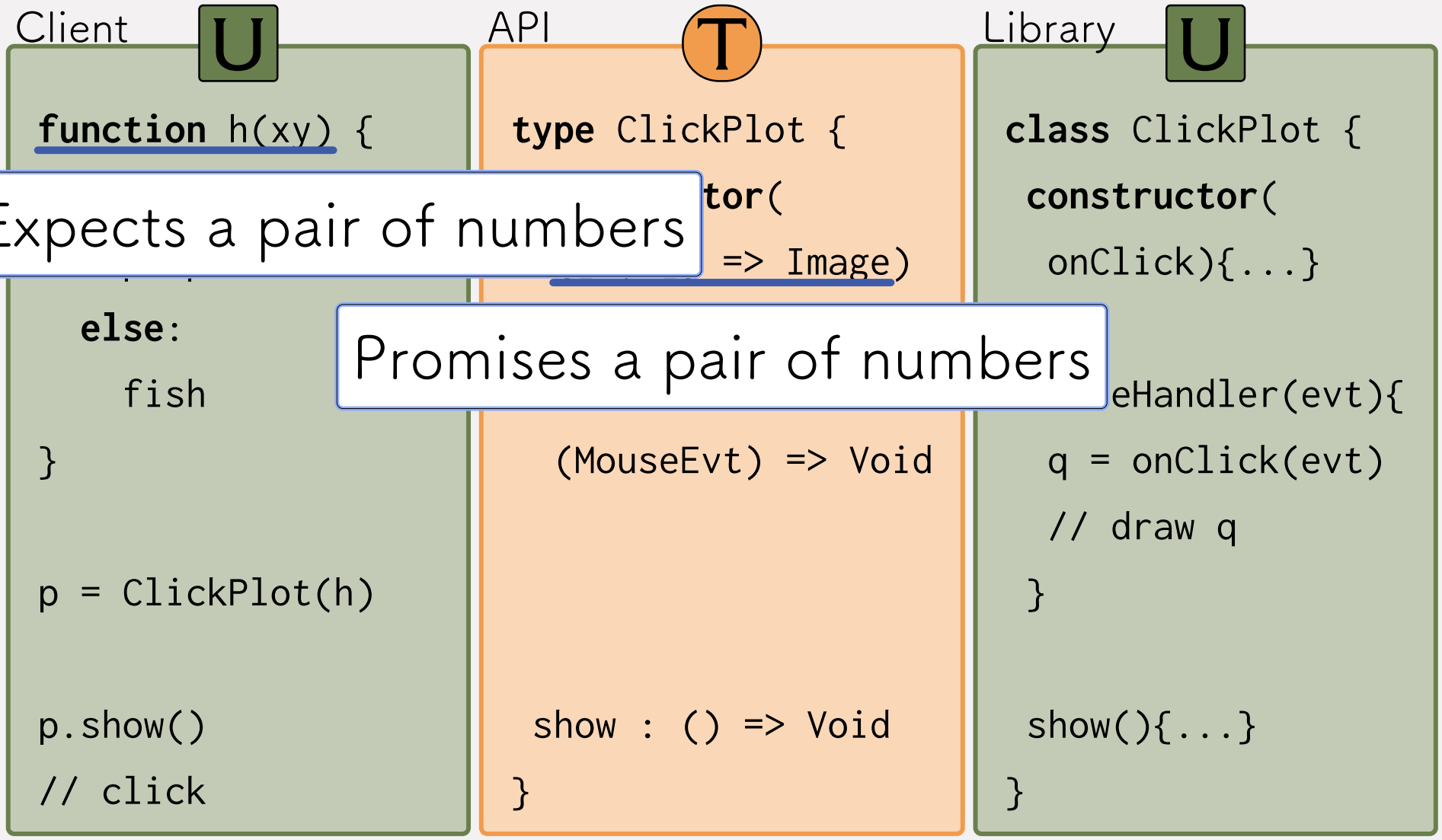
Example: interactive plot



Example: interactive plot



Example: interactive plot



Example: interactive plot

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

API T

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

Library U

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

Example: interactive plot

Client 

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

API 

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

Library 

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

Example: interactive plot

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

API T

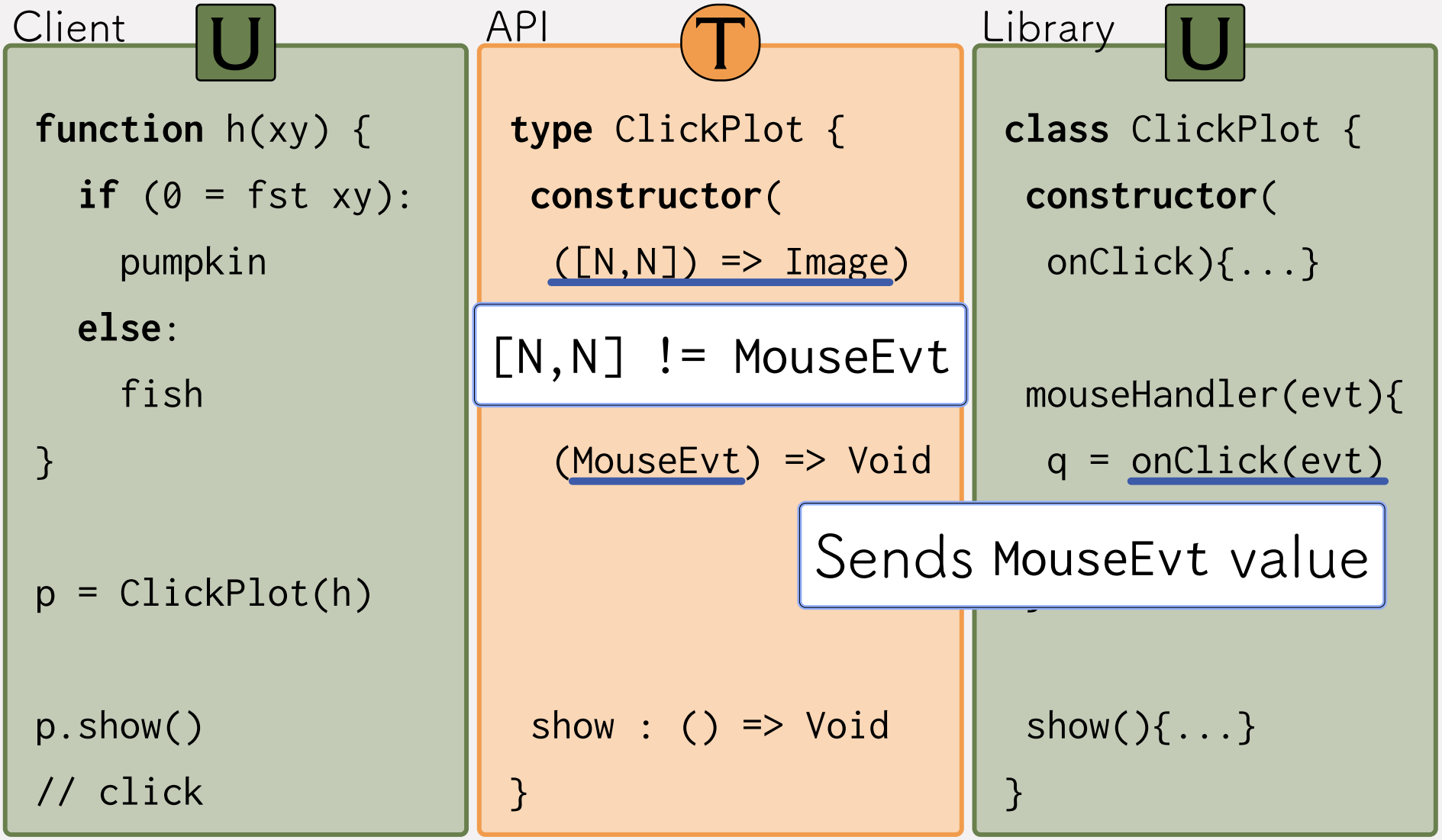
```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

Library U

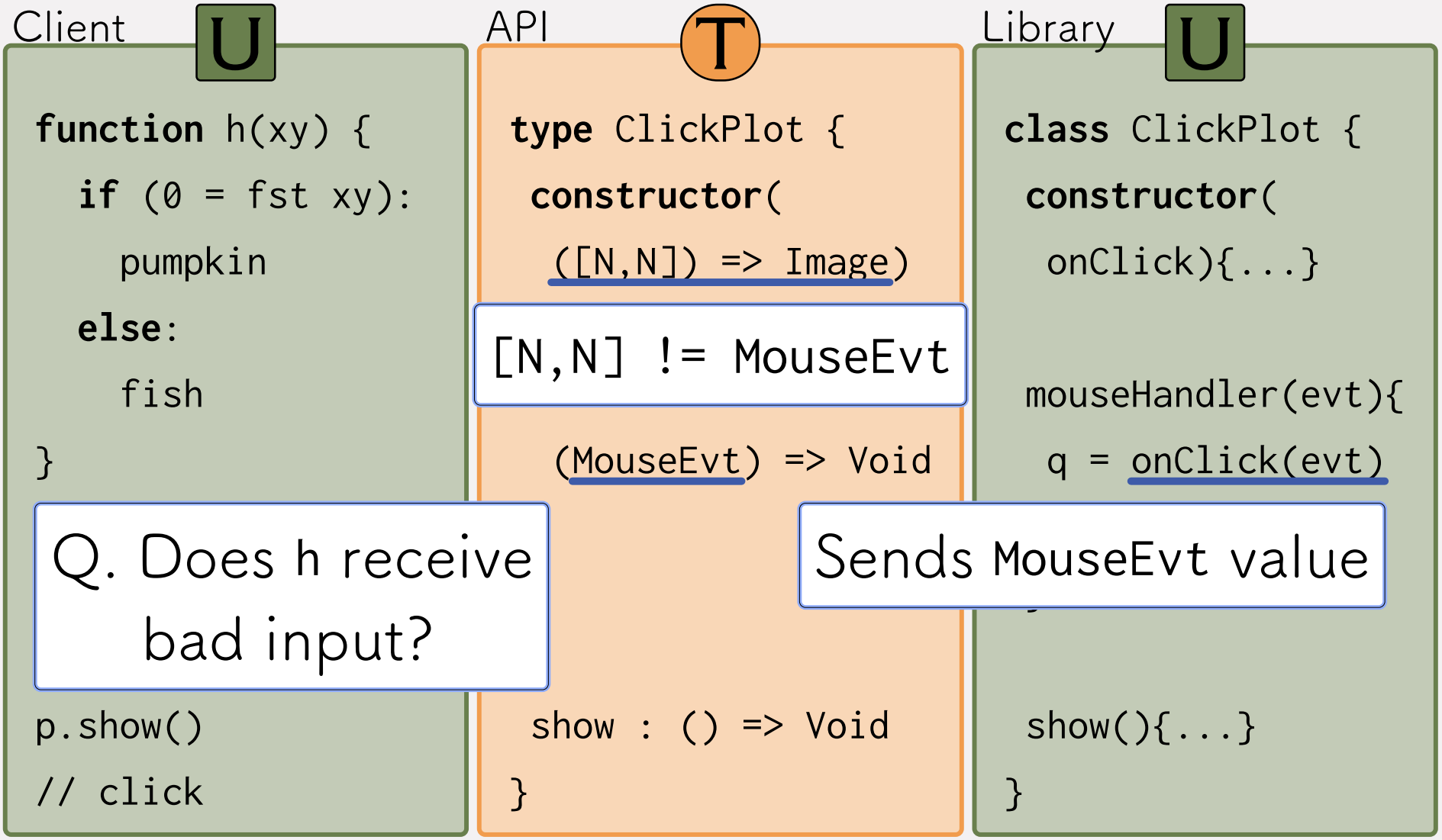
```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)

 show(){...}
}
```

Sends MouseEvt value

Example: interactive plot



Example: interactive plot



Example: interactive plot

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}
```

Q. Does h receive bad input?

```
p.show()
// click
```

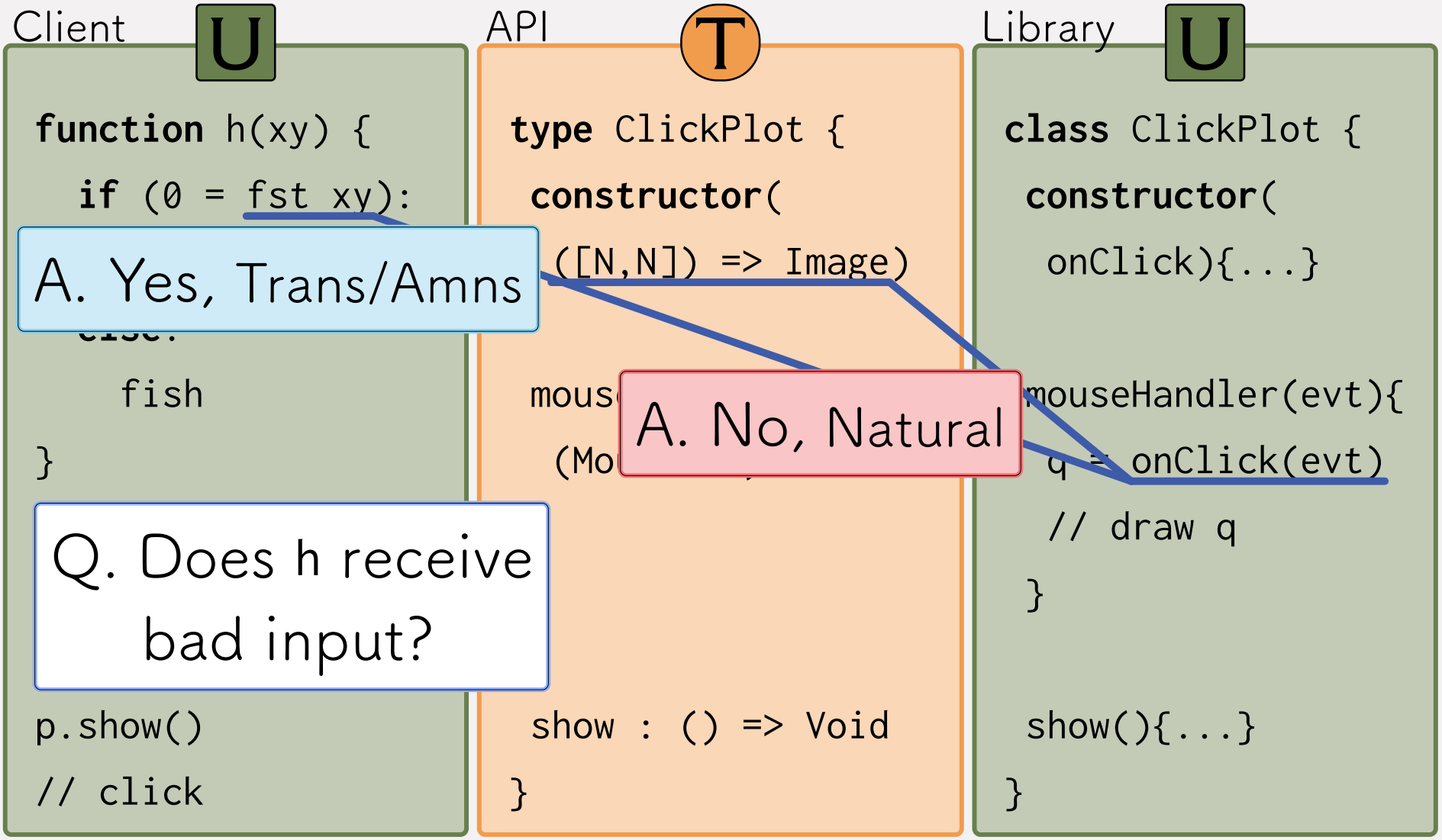
API T

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

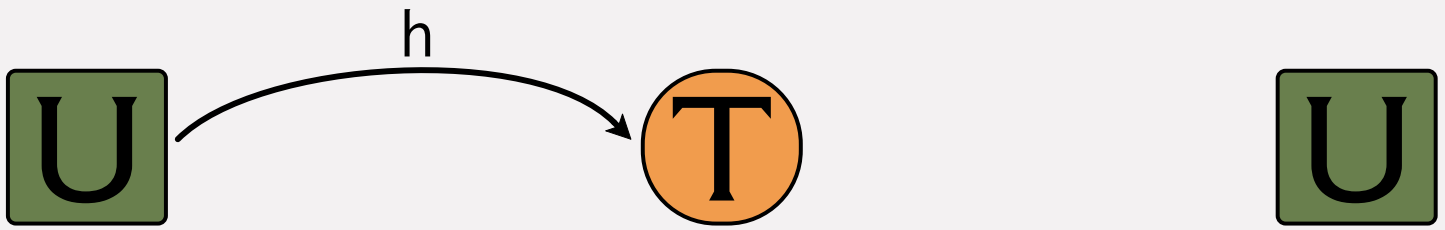
Library U

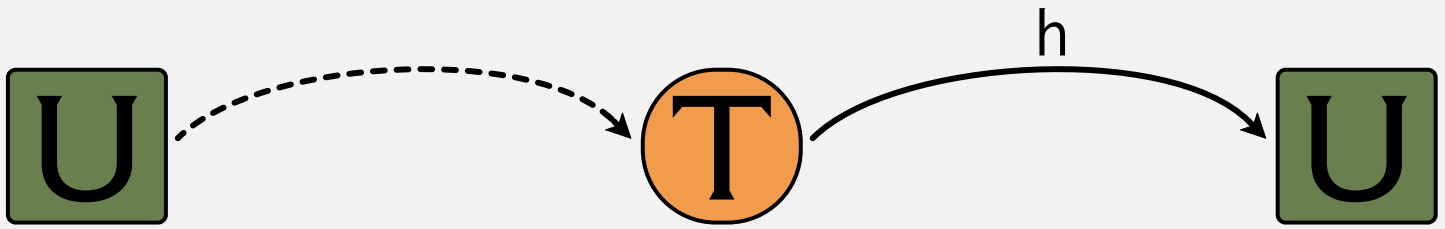
```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

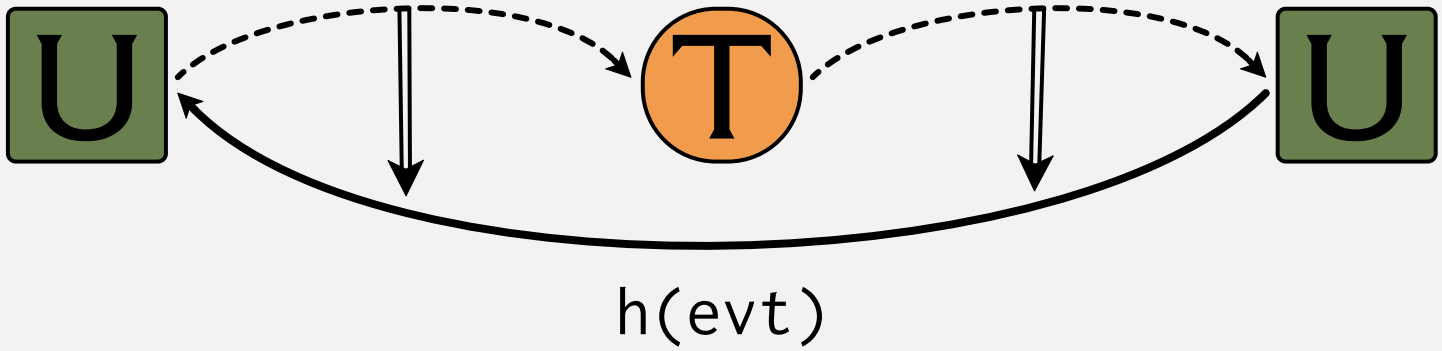
Example: interactive plot



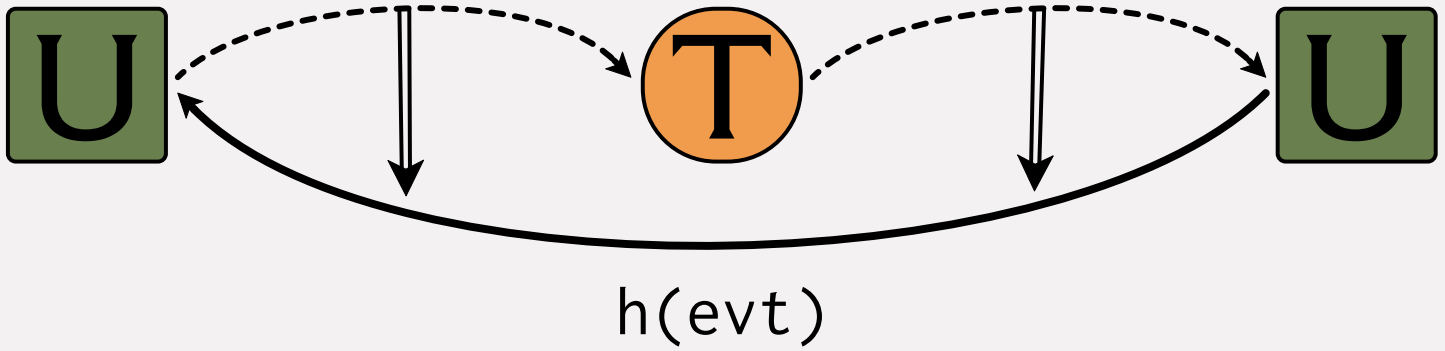

U
T
U


h
U
T
U

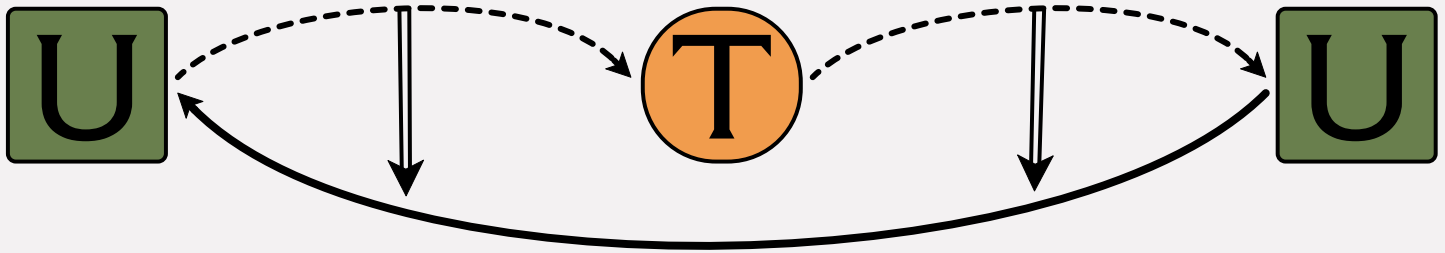

h
U
T
U


U
T
U
h(evt)

Q. Do types guard the **callback** channel?



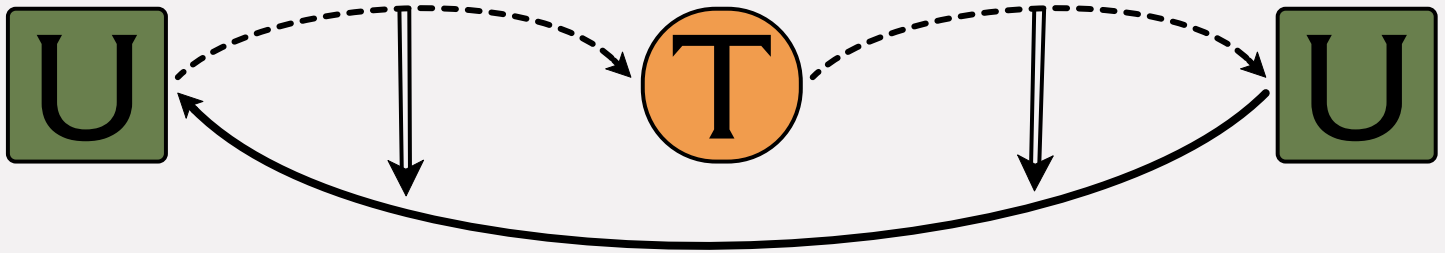
Q. Do types guard the **callback** channel?



Transient/Amnesic: no, because the channel is between two untyped components

Natural: yes, because the channel was created via typed code

Q. Do types guard the **callback** channel?



Type Soundness $\not\Rightarrow$ yes

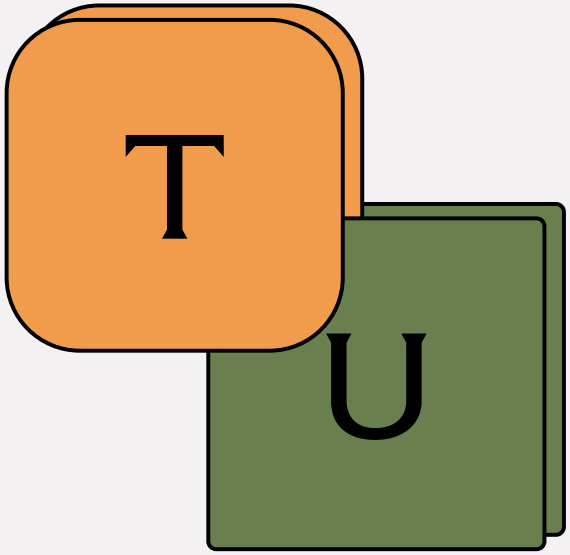
Complete Monitoring $\Rightarrow$ yes

!!!

Every Typed Language is Mixed-Typed

Many typed languages
trust untyped code

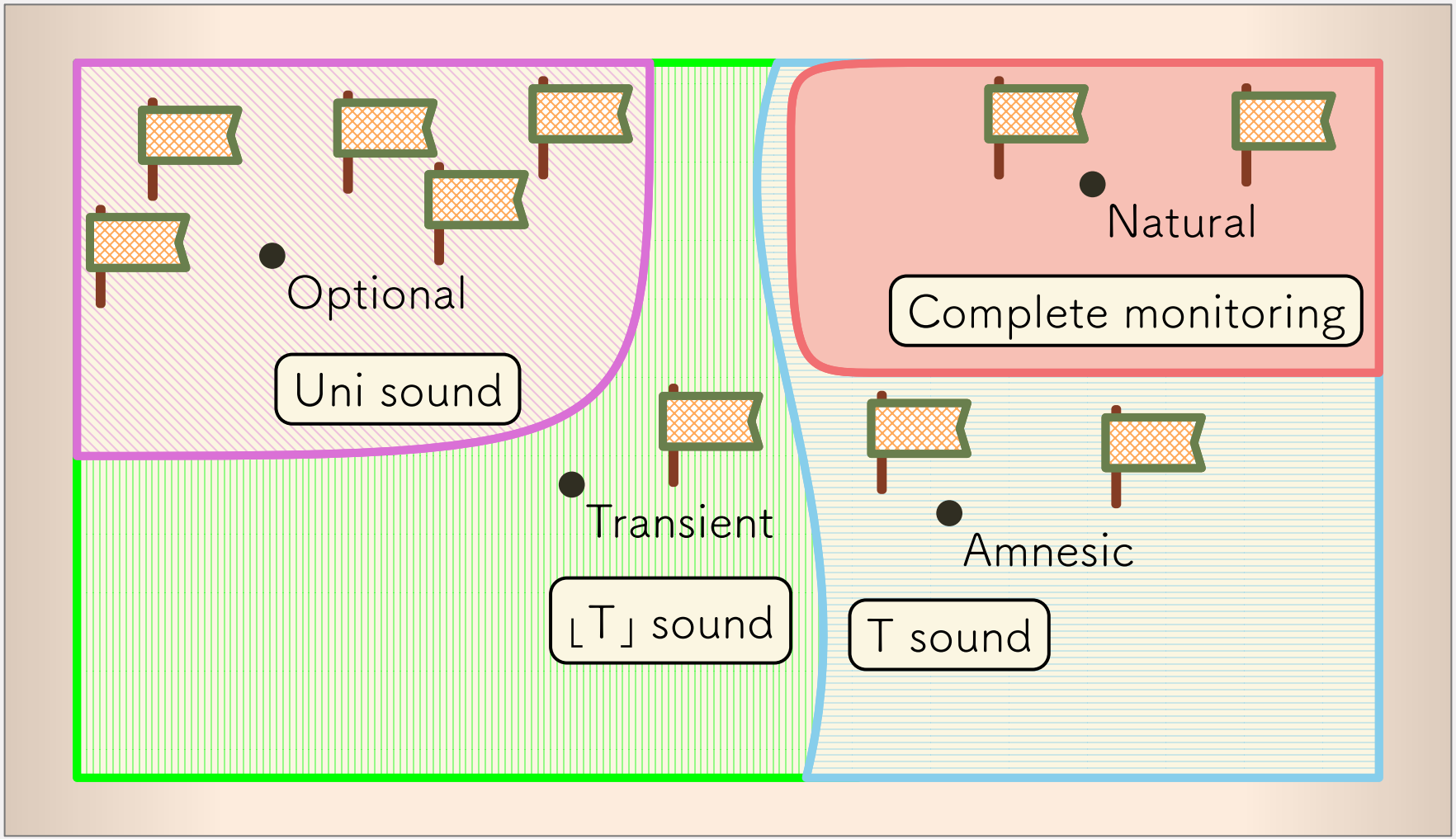
Every Typed Language is Mixed-Typed



Many typed languages **trust** untyped code

Gradual typing makes these boundaries **visible** ...

... and **challenges** our notions of types and what types mean


Optional
Uni sound
Natural
Complete monitoring
Transient
⌊T⌋ sound
Amnesic
T sound

	Natural	Transient	Amnesic
type soundness	T	⌊T⌋	T
complete monitoring			

	Natural	Transient	Amnesic
type soundness	T	⌊T⌋	T
complete monitoring	✓	✗	✗
BLAME			

Natural, Blame

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

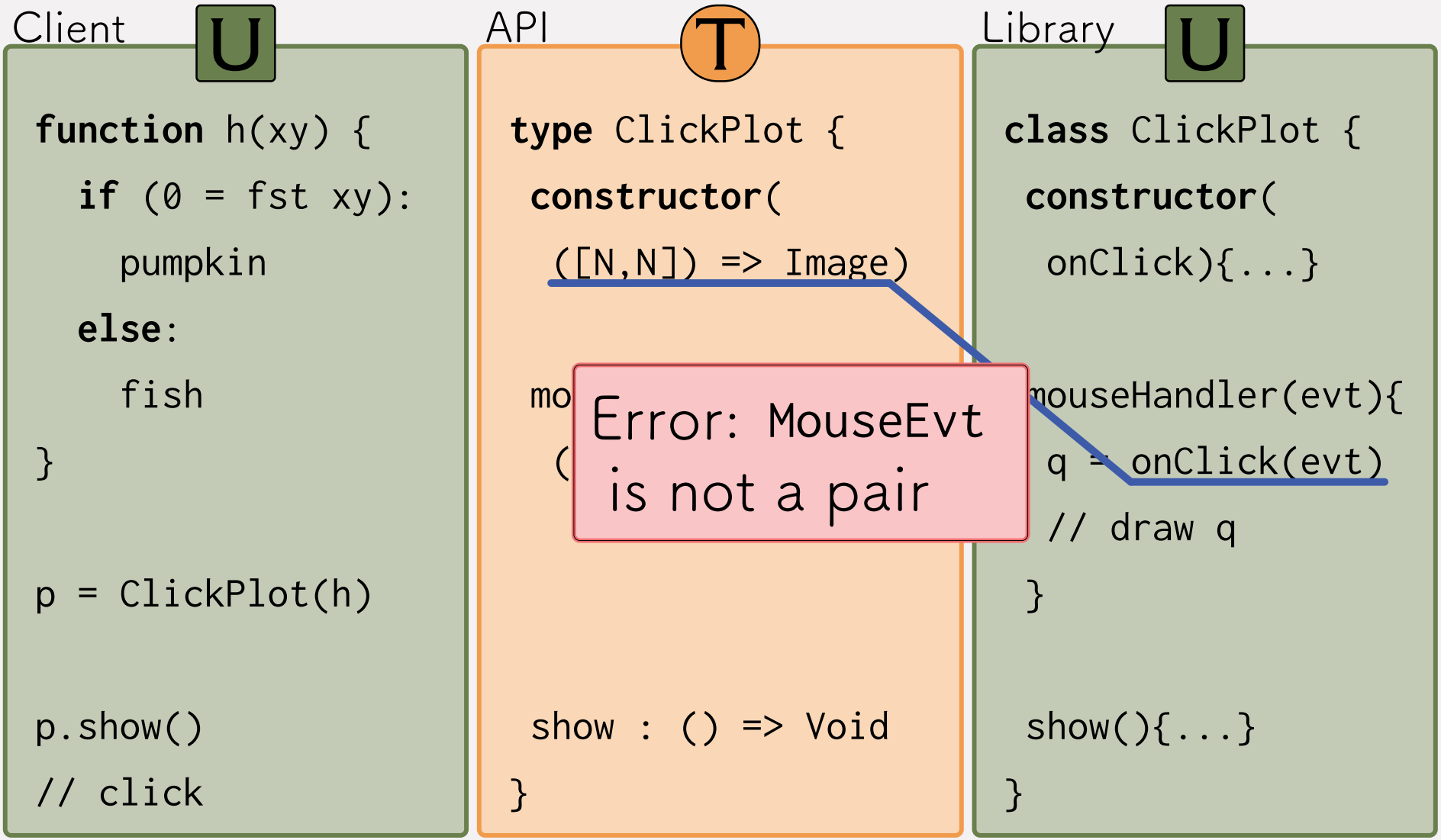
API T

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

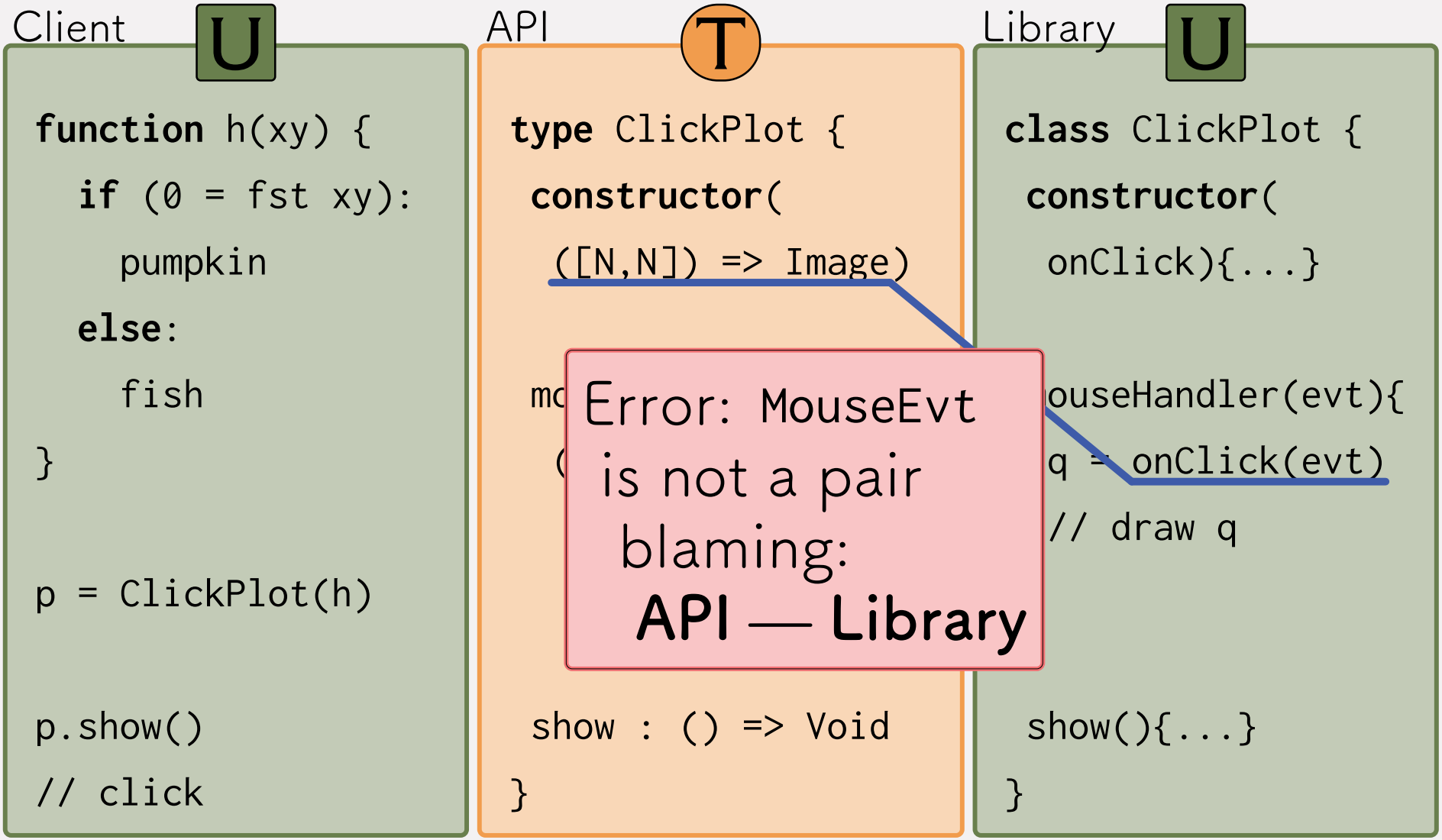
Library U

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

Natural, Blame



Natural, Blame



Transient/Amnesic, Blame (Best Case)

Client U

```
function h(xy) {
  if (0 = fst xy):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

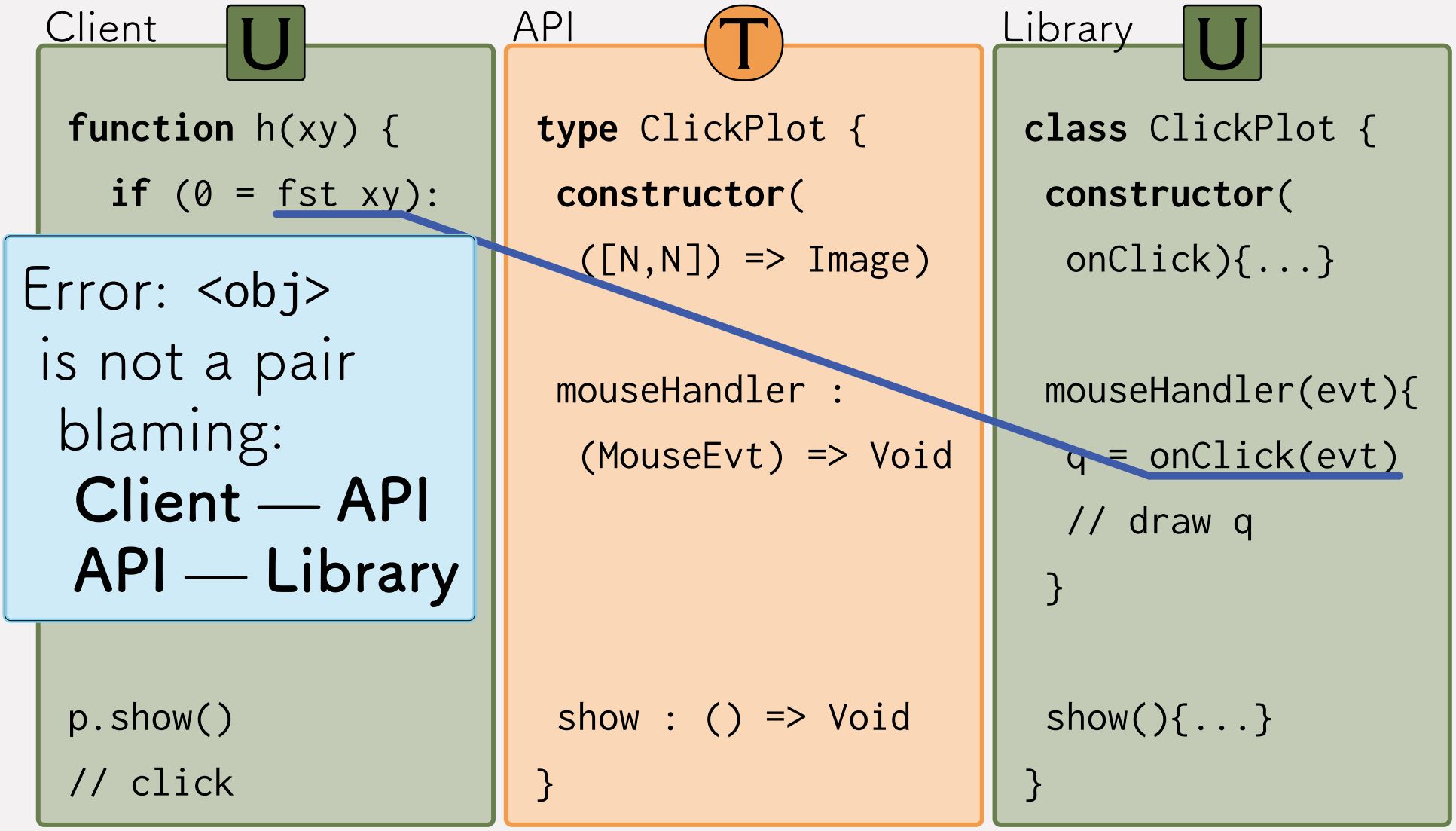
API T

```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

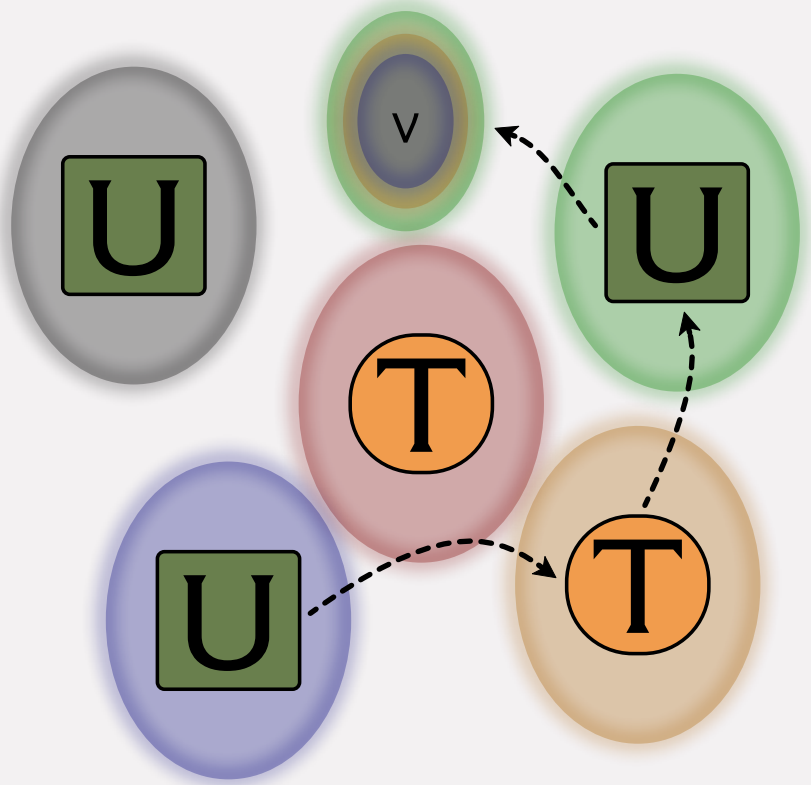
Library U

```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  q = onClick(evt)
  // draw q
 }

 show(){...}
}
```

Transient/Amnesic, Blame (Best Case)



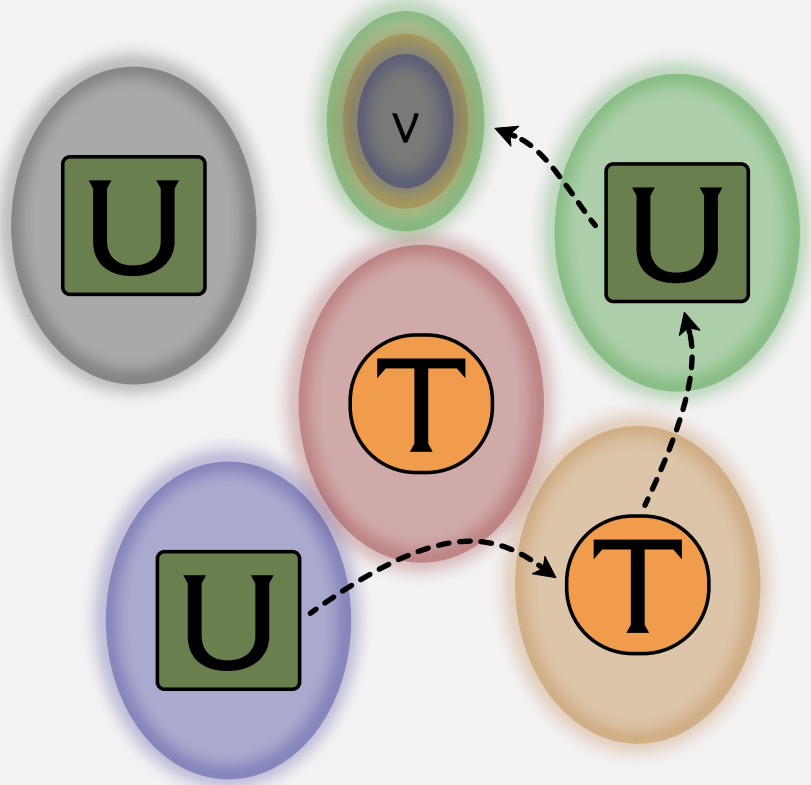
Blame Properties



Blame Properties

1. blame **only** responsible edges

2. blame **all** responsible edges



3. blame **exactly** the responsible edges

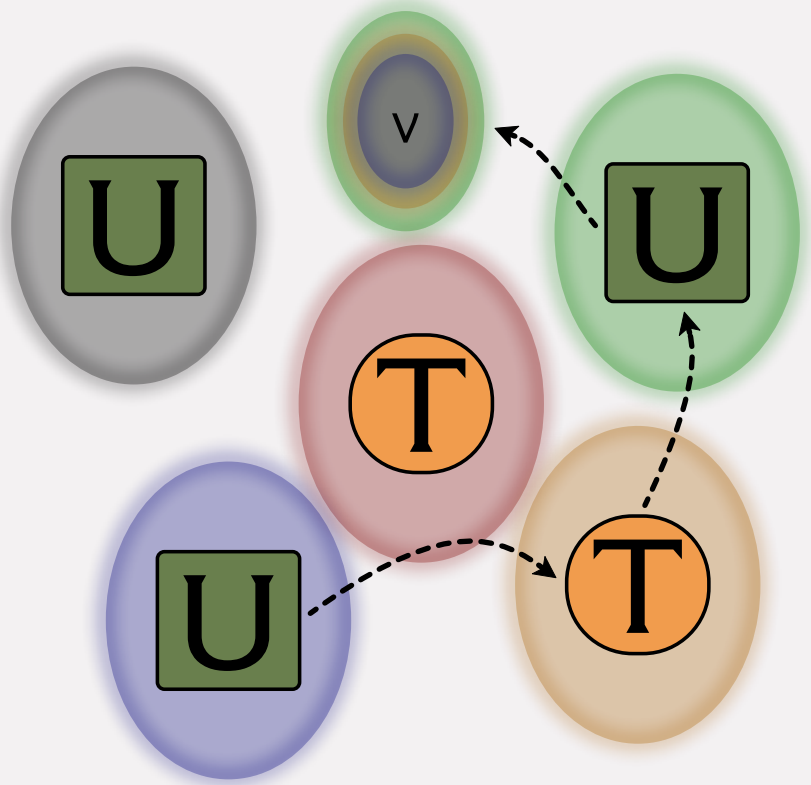
Blame Properties

Blame Soundness

1. blame **only** responsible edges

Blame Completeness

2. blame **all** responsible edges



B. Soundness + B. Completeness

3. blame **exactly** the responsible edges

	Natural	Transient	Amnesic
type soundness	T	⌊T⌋	T
complete monitoring	✓	✗	✗
blame soundness	✓		
blame completeness	✓		

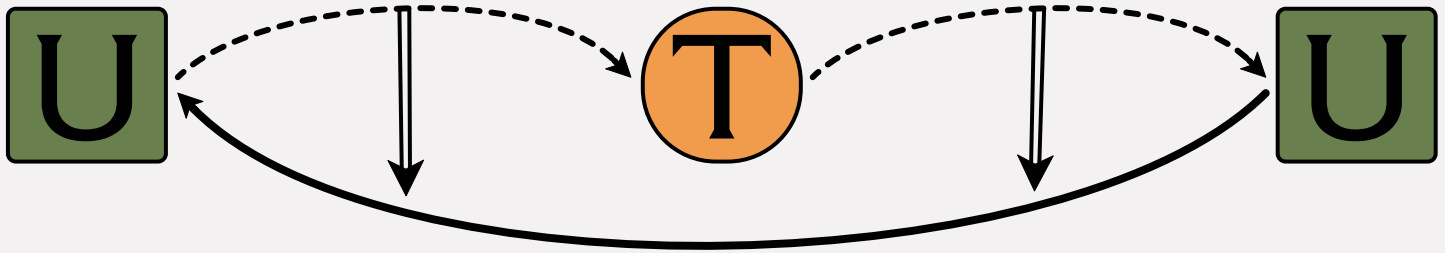
	Natural	Transient	Amnesic
type soundness	T	⌊T⌋	T
complete monitoring	✓	✗	✗
blame soundness	✓	✗	✓
blame completeness	✓	✗	✓

Complete monitoring **strengthens** type soundness for programs that **compose** typed and untyped

the proof framework **enables** precise statements about the quality of blame

Formal Details

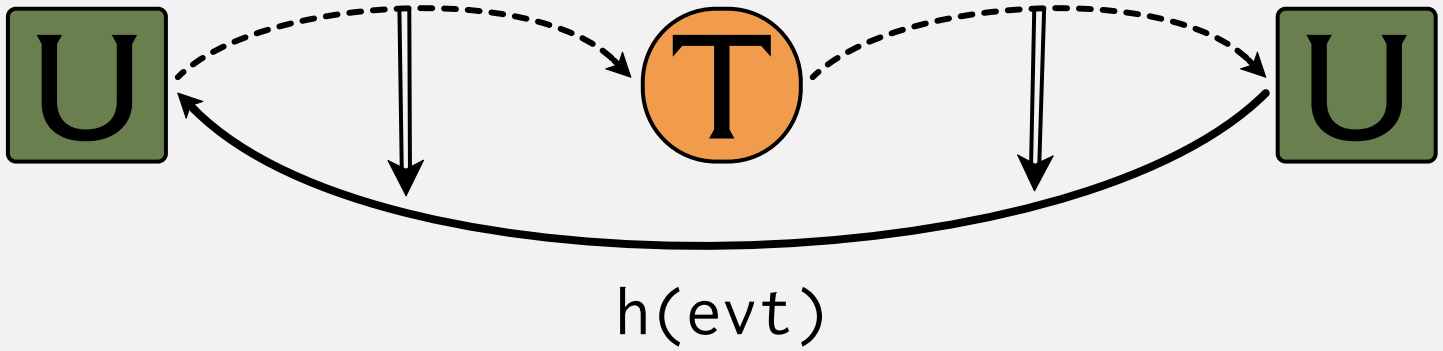
Q. Do types guard the **callback** channel?



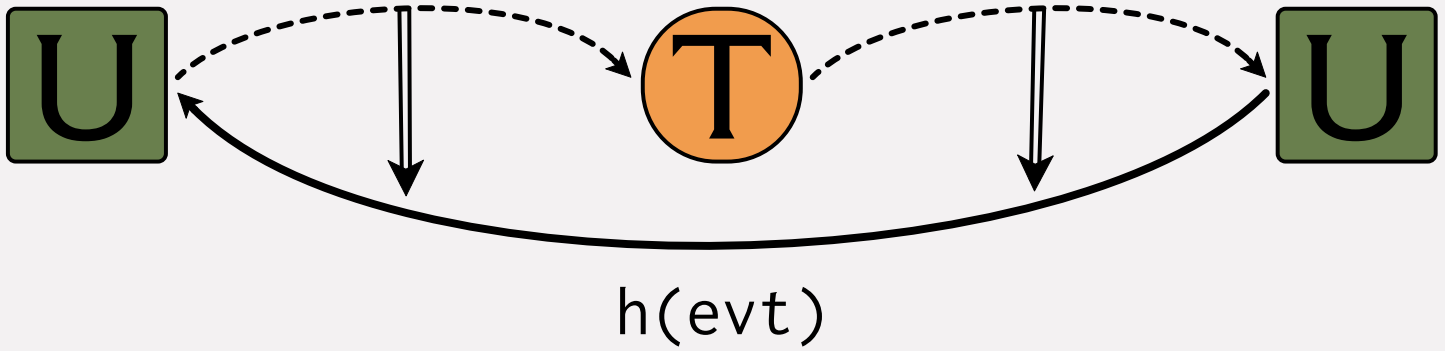
Type Soundness $\not\Rightarrow$ yes

Complete Monitoring $\Rightarrow$ yes

Complete Monitoring, Intuitions

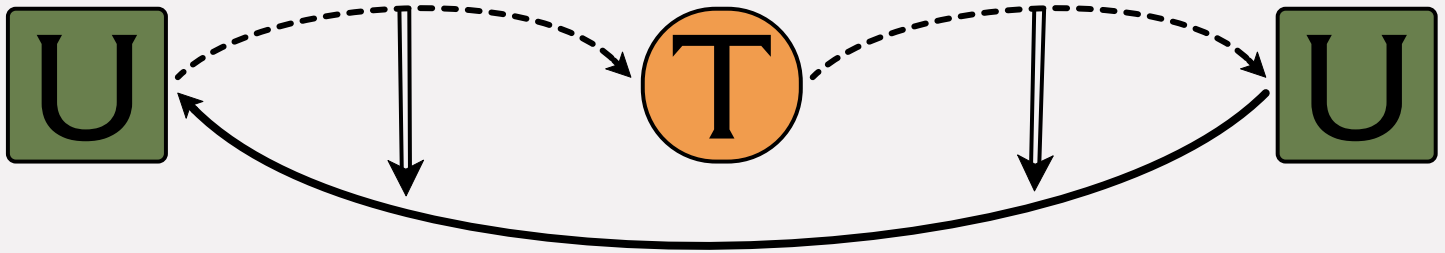


Complete Monitoring, Intuitions



- need to record paths
- need to test whether types are fully enforced
- want a syntactic technique

Complete Monitoring, Intuitions

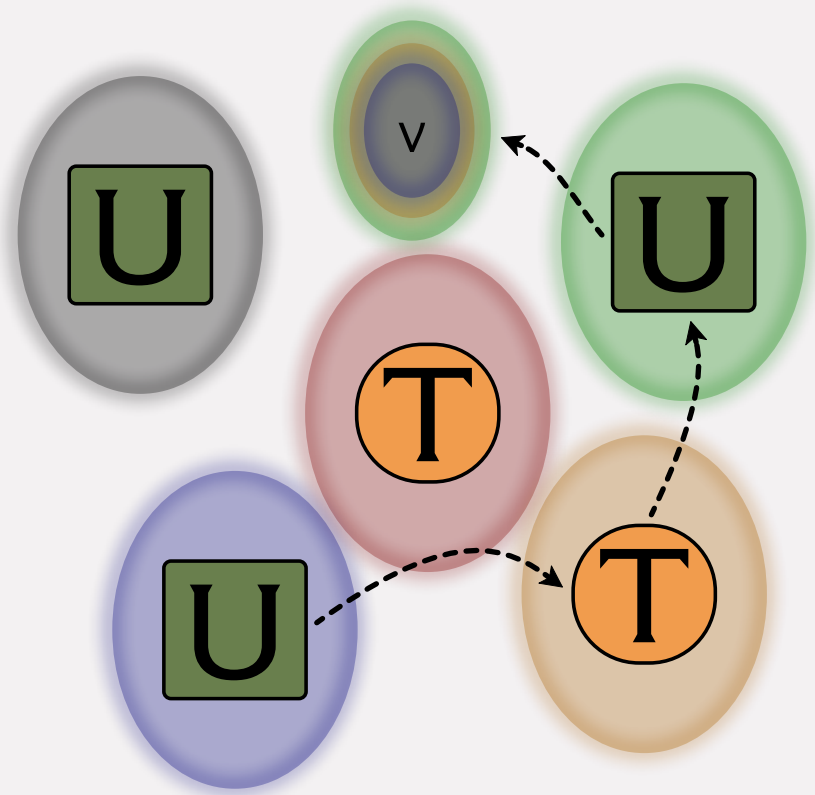


- ne
- ne ced
- wa

Plan:

- enrich syntax with 'invisible' labels
- track communications during reduction
- multi labels = shared ownership

Tracking Communications



Syntax

$$
\begin{array}{lcl}
e & = & x \mid i \mid n \mid \langle e, e \rangle \mid \lambda x.\, e \mid \lambda(x:\tau).\, e \mid \mathsf{app}\{\tau/\mathcal{U}\}\ e\ e \mid \mathit{unop}\{\tau/\mathcal{U}\}\ e \mid \mathit{binop}\{\tau/\mathcal{U}\}\ e\ e \mid \\
 & & \mathsf{dyn}\ b\ e \mid \mathsf{stat}\ b\ e \\
\tau & = & \mathsf{Int} \mid \mathsf{Nat} \mid \tau \Rightarrow \tau \mid \tau \times \tau \\
\tau/\mathcal{U} & = & \tau \mid \mathcal{U}
\end{array}
$$

$$
\begin{array}{lcl}
b & = & (\ell \blacktriangleleft \tau \blacktriangleleft \ell) \\
\ell & = & \text{countable set of names}
\end{array}
$$

Syntax

$$
\begin{array}{lcl}
e & = & x \mid i \mid n \mid \langle e, e \rangle \mid \lambda x.\, e \mid \lambda(x:\tau).\, e \mid \mathsf{app}\{\tau/\mathcal{U}\}\, e\, e \mid \mathit{unop}\{\tau/\mathcal{U}\}\, e \mid \mathit{binop}\{\tau/\mathcal{U}\}\, e\, e \mid \\
 & & \mathsf{dyn}\ b\ e \mid \mathsf{stat}\ b\ e \\
\tau & = & \mathsf{Int} \mid \mathsf{Nat} \mid \tau \Rightarrow \tau \mid \tau \times \tau \\
\tau/\mathcal{U} & = & \tau \mid \mathcal{U}
\end{array}
$$

$$
\begin{array}{lcl}
b & = & (\ell \blacktriangleleft \tau \blacktriangleleft \ell) \\
\ell & = & \text{countable set of names}
\end{array}
$$

Labeled Syntax

$$
\begin{array}{lcl}
e & = & x \mid i \mid n \mid \langle e, e \rangle \mid \lambda x.\, e \mid \lambda(x:\tau).\, e \mid \mathsf{app}\{\tau/\mathcal{U}\}\, e\, e \mid \mathit{unop}\{\tau/\mathcal{U}\}\, e \mid \mathit{binop}\{\tau/\mathcal{U}\}\, e\, e \mid \\
 & & \mathsf{dyn}\ b\ (e)^{\ell} \mid \mathsf{stat}\ b\ (e)^{\ell} \mid (e)^{\ell}
\end{array}
$$

Labeled Examples

Valid

$$(\lambda x_0.\, x_0)^{\ell_0}$$

$$((\lambda x_0.\, (x_0)^{\ell_0})^{\ell_1})^{\ell_2}$$

$$((\langle (1)^{\ell_0}, (2)^{\ell_1} \rangle)^{\ell_2})^{\ell_3}$$

$$(\mathsf{stat}\ (\ell_0 \blacktriangleleft \mathsf{Nat} \blacktriangleleft \ell_1)\ ((0)^{\ell_2})^{\ell_1})^{\ell_0}$$

Invalid

$$(\mathsf{stat}\ (\ell_0 \blacktriangleleft \mathsf{Nat} \blacktriangleleft \ell_1)\ 0)^{\ell_0}$$

Shorthand ((e))^l

$$(((42)^{\ell_0})^{\ell_1})^{\ell_2} = (\!(42)\!)^{\ell_0\ell_1\ell_2} \sim (\!(42)\!)^{\overline{\ell}_3}$$

Natural Reduction, Examples

$$\begin{array}{ll}
\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\lambda x_0.\, e_0) & \triangleright_N\ \mathbb{G}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\lambda x_0.\, e_0) \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \times \tau_1) \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle & \triangleright_N\ \langle \mathsf{dyn}\ (\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1)\ v_0, \mathsf{dyn}\ (\ell_0 \blacktriangleleft \tau_1 \blacktriangleleft \ell_1)\ v_1 \rangle \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ i_0 & \triangleright_N\ i_0 \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle & \triangleright_N\ \mathsf{BoundaryErr}\ ((\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1), \langle v_0, v_1 \rangle)
\end{array}$$

Natural Reduction, Examples

$$\begin{array}{ll}
\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\lambda x_0.\, e_0) & \triangleright_{\mathsf{N}}\ \mathbb{G}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\lambda x_0.\, e_0) \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \times \tau_1) \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle & \triangleright_{\mathsf{N}}\ \langle \mathsf{dyn}\ (\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1)\ v_0, \mathsf{dyn}\ (\ell_0 \blacktriangleleft \tau_1 \blacktriangleleft \ell_1)\ v_1 \rangle \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ i_0 & \triangleright_{\mathsf{N}}\ i_0 \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle & \triangleright_{\mathsf{N}}\ \mathsf{BoundaryErr}\ ((\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1), \langle v_0, v_1 \rangle)
\end{array}$$

Labeled

$$\begin{array}{ll}
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\!(\lambda x_0.\, e_0)\!)^{\overline{\ell}_0})^{\ell_2} & \triangleright_{\overline{\mathsf{N}}}\ (\mathbb{G}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\!(\lambda x_0.\, e_0)\!)^{\overline{\ell}_0})^{\ell_2} \\
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \times \tau_1) \blacktriangleleft \ell_1)\ (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0})^{\ell_2} & \triangleright_{\overline{\mathsf{N}}}\ (\langle \mathsf{dyn}\ (\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1)\ (\!(v_0)\!)^{\overline{\ell}_0}, \mathsf{dyn}\ (\ell_0 \blacktriangleleft \tau_1 \blacktriangleleft \ell_1)\ (\!(v_1)\!)^{\overline{\ell}_0} \rangle)^{\ell_2} \\
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ (\!(i_0)\!)^{\overline{\ell}_0})^{\ell_2} & \triangleright_{\overline{\mathsf{N}}}\ (i_0)^{\ell_2} \\
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0})^{\ell_2} & \triangleright_{\overline{\mathsf{N}}}\ (\mathsf{BoundaryErr}\ ((\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1), (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0}))^{\ell_2}
\end{array}$$

Transient Reduction, Simplified Examples

$$\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\lambda x_0.\, e_0) \rhd_T \lambda x_0.\, e_0$$

$$\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \times \tau_1) \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle \rhd_T \langle v_0, v_1 \rangle$$

$$\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ i_0 \rhd_T i_0$$

$$\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle \rhd_T \mathsf{BoundaryErr}\ (\{(\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1)\}, \langle v_0, v_1 \rangle)$$

Transient Reduction, Simplified Examples

$$\begin{array}{ll}
\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\lambda x_0.\, e_0) & \rhd_{\mathsf{T}}\ \lambda x_0.\, e_0 \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \times \tau_1) \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle & \rhd_{\mathsf{T}}\ \langle v_0, v_1 \rangle \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ i_0 & \rhd_{\mathsf{T}}\ i_0 \\
\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ \langle v_0, v_1 \rangle & \rhd_{\mathsf{T}}\ \mathsf{BoundaryErr}\ (\{(\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1)\}, \langle v_0, v_1 \rangle)
\end{array}$$

Labeled

$$\begin{array}{ll}
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \Rightarrow \tau_1) \blacktriangleleft \ell_1)\ (\!(\lambda x_0.\, e_0)\!)^{\overline{\ell}_0})^{\ell_2} & \rhd_{\mathsf{T}}\ (\!(\lambda x_0.\, e_0)\!)^{\overline{\ell}_0 \ell_2} \\
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft (\tau_0 \times \tau_1) \blacktriangleleft \ell_1)\ (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0})^{\ell_2} & \rhd_{\mathsf{T}}\ (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0 \ell_2} \\
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ (\!(i_0)\!)^{\overline{\ell}_0})^{\ell_2} & \rhd_{\mathsf{T}}\ (\!(i_0)\!)^{\ell_2} \\
(\mathsf{dyn}\ (\ell_0 \blacktriangleleft \mathsf{Int} \blacktriangleleft \ell_1)\ (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0})^{\ell_2} & \rhd_{\mathsf{T}}\ (\mathsf{BoundaryErr}\ (\{(\ell_0 \blacktriangleleft \tau_0 \blacktriangleleft \ell_1)\}, (\!(\langle v_0, v_1 \rangle)\!)^{\overline{\ell}_0}))^{\ell_2}
\end{array}$$

Definitions

Definition (F-type soundness)**.**
If $\vdash e_0 : \mathsf{T}$ *then one of the following holds:*

- $e_0 \rightarrow_{\mathsf{X}}^* v_0$ *and* $\vdash_{\mathsf{F}} v_0 : \mathsf{F}(\mathsf{T})$
- $e_0 \rightarrow_{\mathsf{X}}^*$ *an allowed error*
- $e_0 \rightarrow_{\mathsf{X}}^*$ *diverges*

Definitions

Definition (F-type soundness)**.**
If $\vdash e_0 : T$ *then one of the following holds:*

- $e_0 \rightarrow_X^* v_0$ *and* $\vdash_F v_0 : F(T)$
- $e_0 \rightarrow_X^*$ *an allowed error*
- $e_0 \rightarrow_X^*$ *diverges*

Definition 6.10 (complete monitoring)**.** *A semantics* X *satisfies* **CM** *if for all well-formed* e_0 *and all* e_1 *such that* $e_0 \longrightarrow_{\overline{X}}^* e_1$*, the contractum is single-owner consistent:* $\ell_0 \Vdash e_1$.

Definitions

Definition (F-type soundness)**.**
If $\vdash e_0 : \mathsf{T}$ *then one of the following holds:*

- $e_0 \to_{\mathsf{X}}^* v_0$ *and* $\vdash_{\mathsf{F}} v_0 : \mathsf{F}(\mathsf{T})$
- $e_0 \to_{\mathsf{X}}^*$ *an allowed error*
- $e_0 \to_{\mathsf{X}}^*$ *diverges*

Definition 6.10 (complete monitoring)**.** *A semantics* X *satisfies* **CM** *if for all well-formed* e_0 *and all* e_1 *such that* $e_0 \longrightarrow_{\overline{\mathsf{X}}}^* e_1$*, the contractum is single-owner consistent:* $\ell_0 \Vdash e_1$.

Definition 6.11 (path-based blame soundness and blame completeness)**.** *For all well-formed* e_0 *such that* $e_0 \longrightarrow_{\overline{\mathsf{X}}}^* \mathsf{BoundaryErr}\,(b_0^*, v_0)$*:*

- X *satisfies* **BS** *iff senders* $(b_0^*) \subseteq$ *owners* (v_0)
- X *satisfies* **BC** *iff senders* $(b_0^*) \supseteq$ *owners* (v_0)

	Natural	Transient	Amnesic
type soundness	T	⌊T⌋	T
complete monitoring	✓	✗	✗
blame soundness	✓	✗	✓
blame completeness	✓	✗	✓

	N	$\lesssim$	C	$\lesssim$	F	$\lesssim$	T	$\simeq$	A	$\lesssim$	E
type soundness	**1**		**1**		**1**		**s**†		**1**		**0**
complete monitoring	✓		✓		×		×		×		×
blame soundness	✓		✓		✓		*heap*		✓		∅
blame completeness	✓		✓		×‡		×		✓		×
no wrappers	×		×		×		✓		×		✓

† indirectly satisfies **TS**(**1**) by a bisimulation to A (theorem 6.31)

‡ satisfiable by adding A-style trace wrappers, see supplement

Extra

Gradual Guarantee

▶ **Theorem 5** (Gradual Guarantee). *Suppose* $e \sqsubseteq e'$ *and* $\vdash e : T$.

1. $\vdash e' : T'$ *and* $T \sqsubseteq T'$.
2. *If* $e \Downarrow v$, *then* $e' \Downarrow v'$ *and* $v \sqsubseteq v'$.
   If $e \Uparrow$ *then* $e' \Uparrow$.
3. *If* $e' \Downarrow v'$, *then* $e \Downarrow v$ *where* $v \sqsubseteq v'$, *or* $e \Downarrow \mathtt{blame}_T\, l$.
   If $e' \Uparrow$, *then* $e \Uparrow$ *or* $e \Downarrow \mathtt{blame}_T\, l$.

Siek, Vitousek, Cimini, Boyland SNAPL 2015

Gradual Guarantee

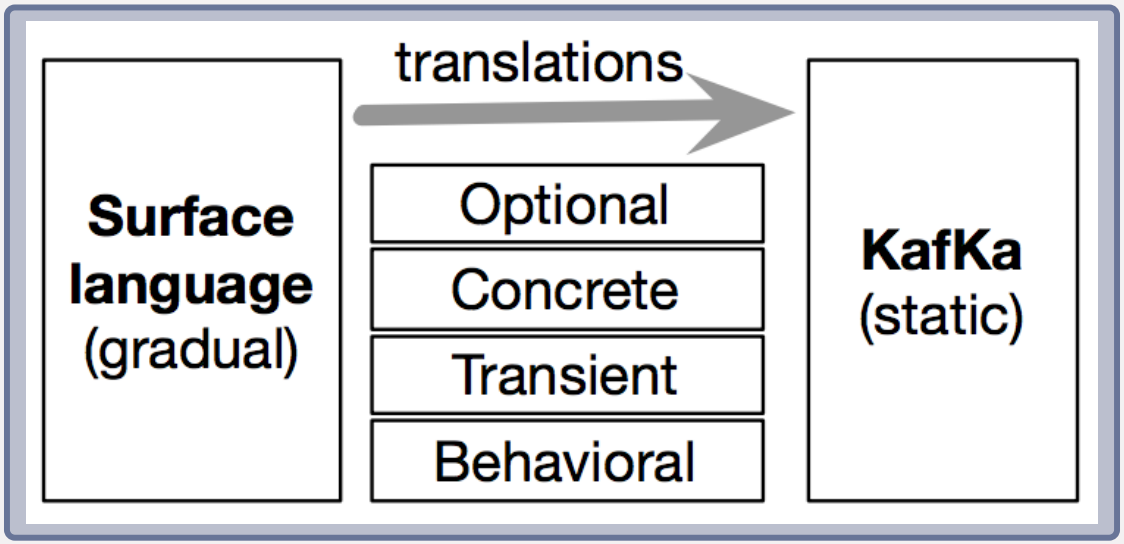
▶ **Theorem 5** (Gradual Guarantee). *Suppose* $e \sqsubseteq e'$ *and* $\vdash e : T$.

1. $\vdash e' : T'$ *and* $T \sqsubseteq T'$.
2. *If* $e \Downarrow v$, *then* $e' \Downarrow v'$ *and* $v \sqsubseteq v'$.
   If $e \Uparrow$ *then* $e' \Uparrow$.
3. *If* $e' \Downarrow v'$, *then* $e \Downarrow v$ *where* $v \sqsubseteq v'$, *or* $e \Downarrow \mathtt{blame}_T\, l$.
   If $e' \Uparrow$, *then* $e \Uparrow$ *or* $e \Downarrow \mathtt{blame}_T\, l$.

Siek, Vitousek, Cimini, Boyland SNAPL 2015

- concerns only the Dyn type
- satisfied by Natural, Transient, and Optional

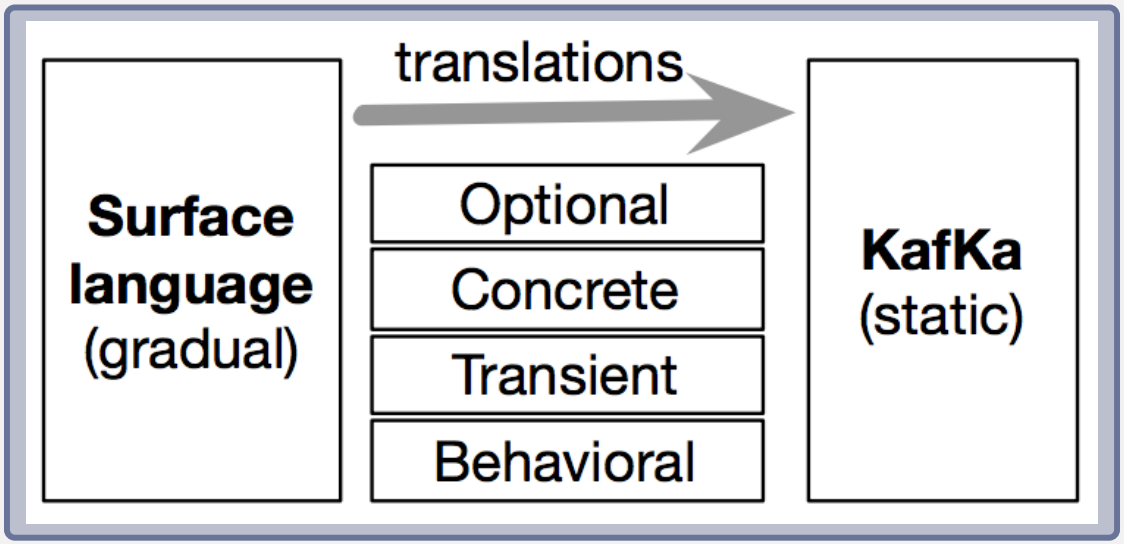
KafKa: Gradual Typing for Objects



Chung, Li, Zappa Nardelli, Vitek ECOOP 2018

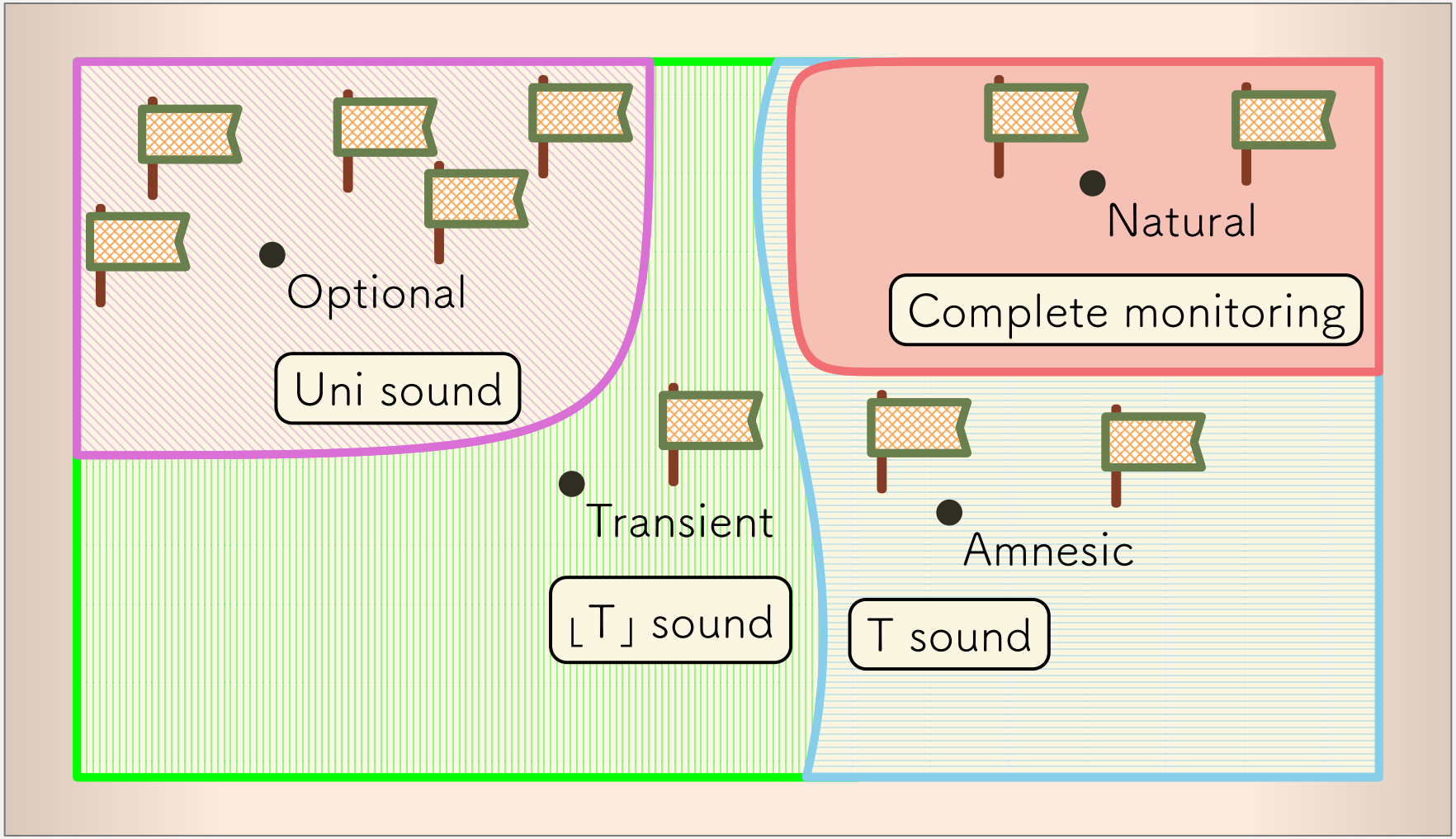
- different behaviors as different translations

KafKa: Gradual Typing for Objects



Chung, Li, Zappa Nardelli, Vitek ECOOP 2018

- different behaviors as different translations
- KafKa is mechanized, type-sound
- lacks a formal comparison of the translations


Optional
Uni sound
Natural
Complete monitoring
Transient
⌊T⌋ sound
Amnesic
T sound

Type soundness is not enough

Complete monitoring is crucial
for **meaningful** gradual types

"Incomplete" monitoring provides a way to
measure the quality of blame errors