COMPLETE MONITORS FOR GRADUAL TYPES

- Ben Greenman
  at Northeastern

Matthias Felleisen
at Northeastern

Christos Dimoulas
at Northwestern

COMPLETE MONITORS FOR GRADUAL TYPES

a careful analysis of the mixed-typed design space

✦ Ben Greenman
at Northeastern

Matthias Felleisen
at Northeastern

Christos Dimoulas
at Northwestern

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

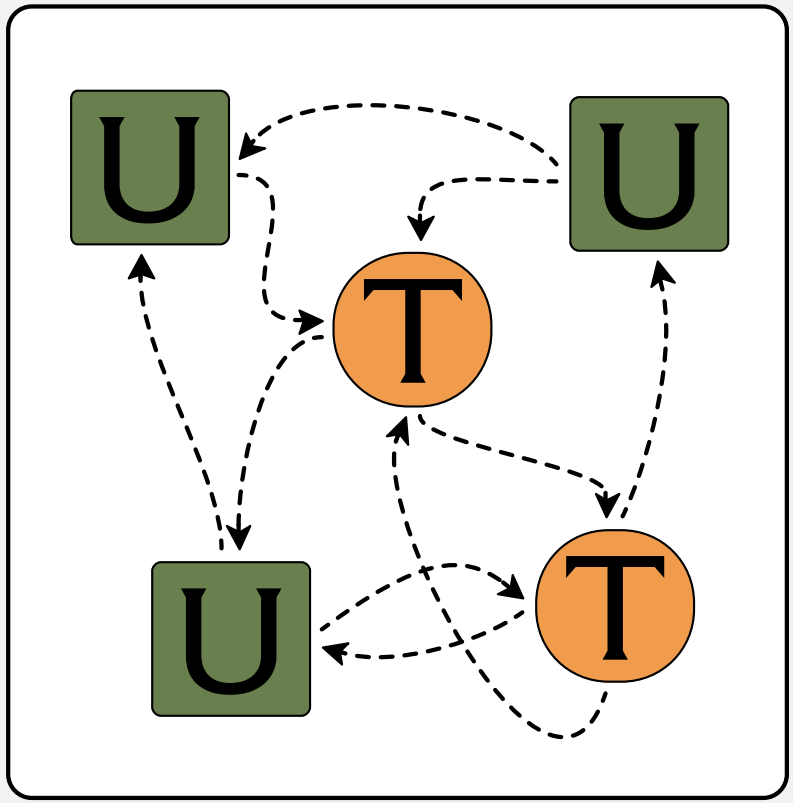
Mixed-Typed Language

Mixed-Typed Language

U = untyped code

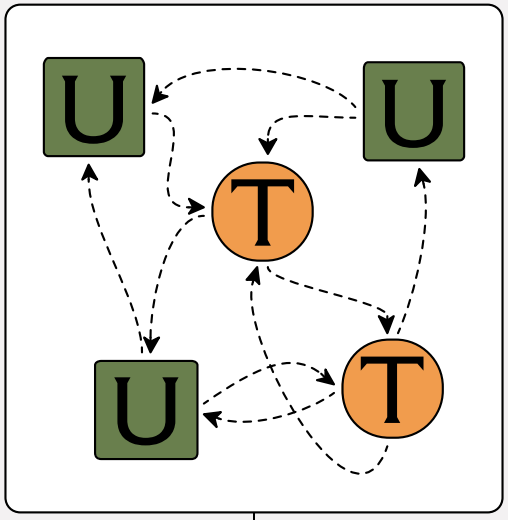
T = simply-typed code

(no 'Dynamic' type)



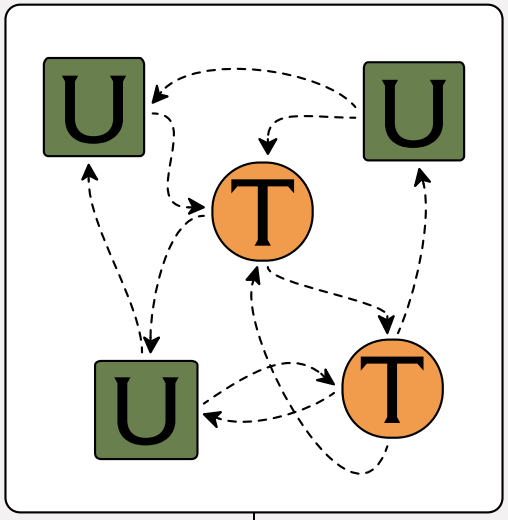
Untyped/Typed mix

A Few Motivations



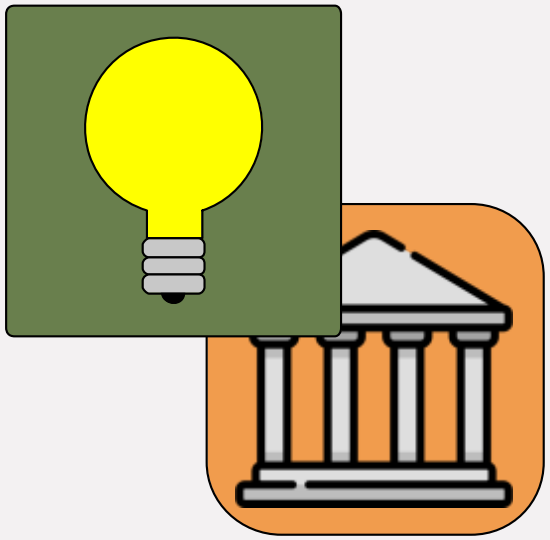
A Few Motivations

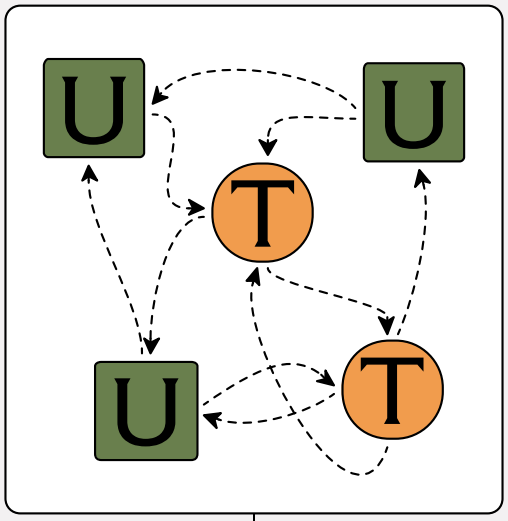
Prototyping



A Few Motivations

Prototyping

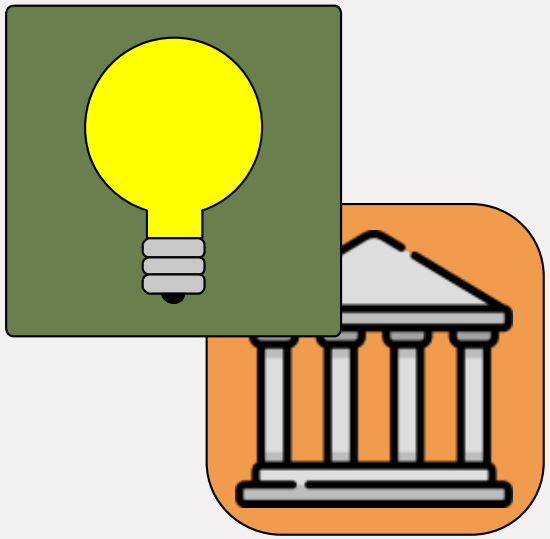




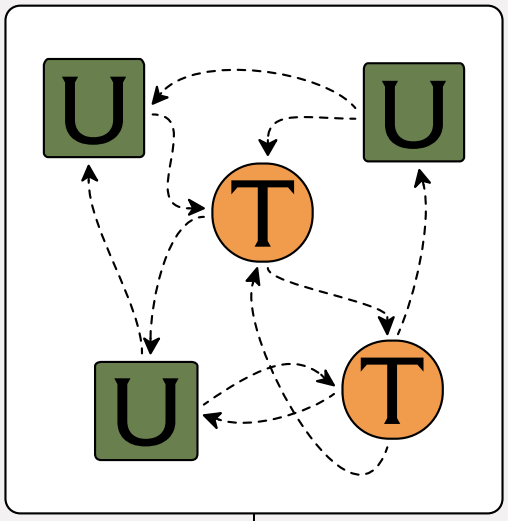
write untyped code,
rely on types

A Few Motivations

Prototyping



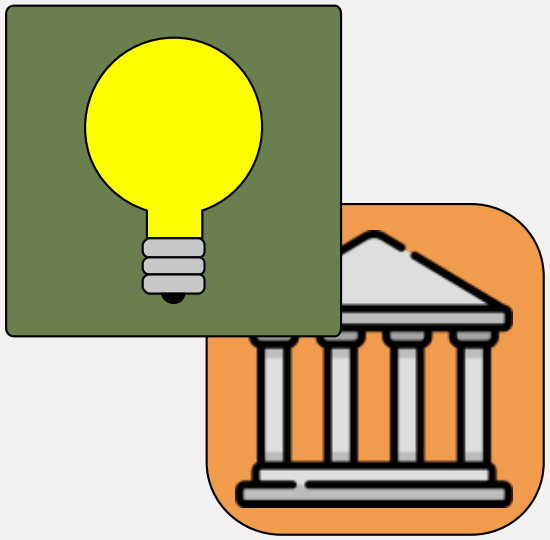
write untyped code,
rely on types



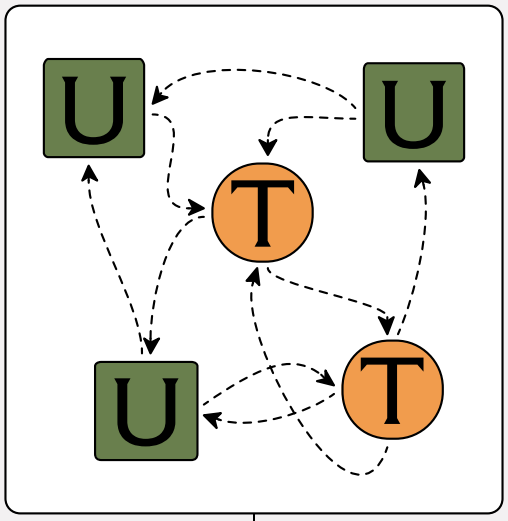
Re-Use

A Few Motivations

Prototyping



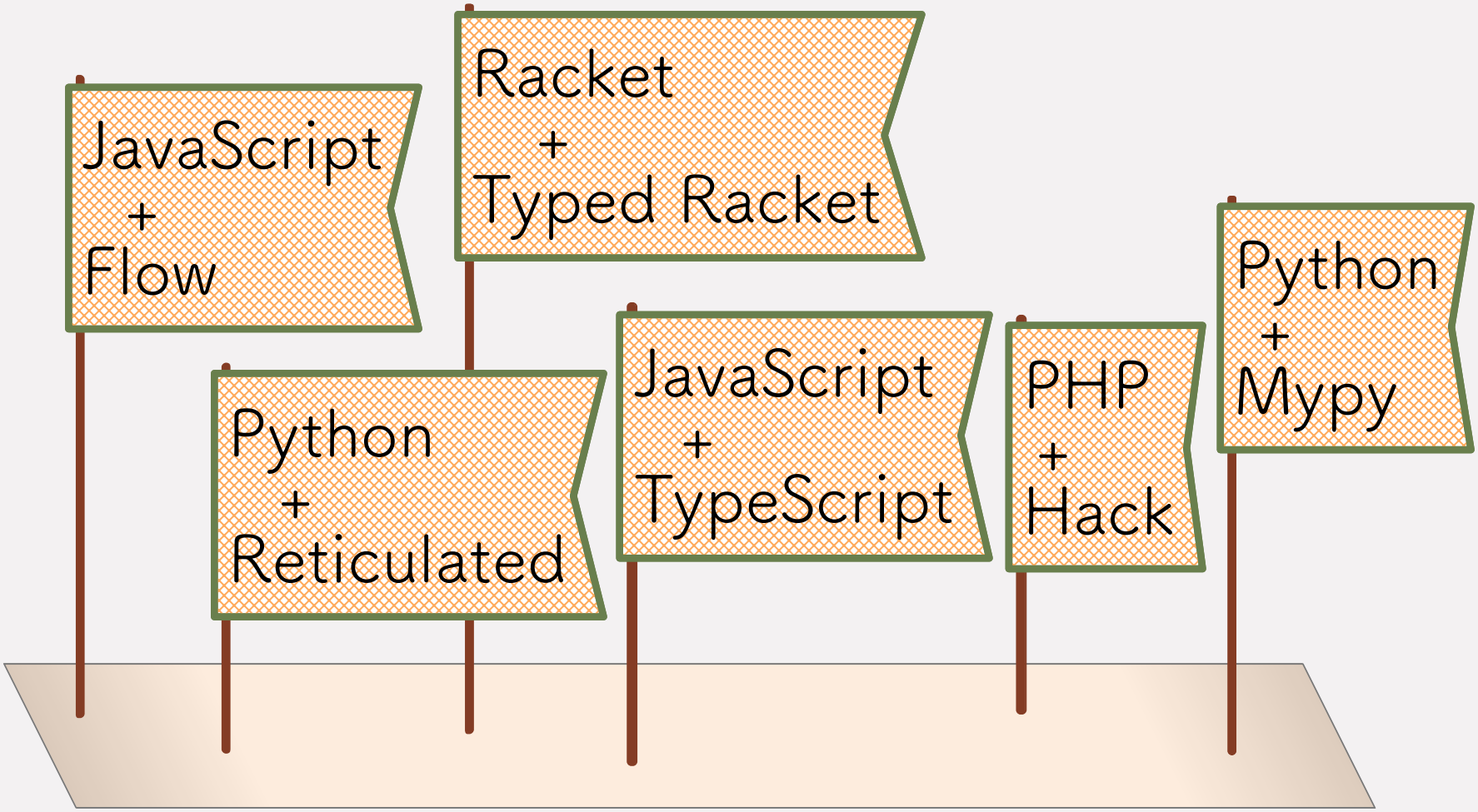
write untyped code,
rely on types



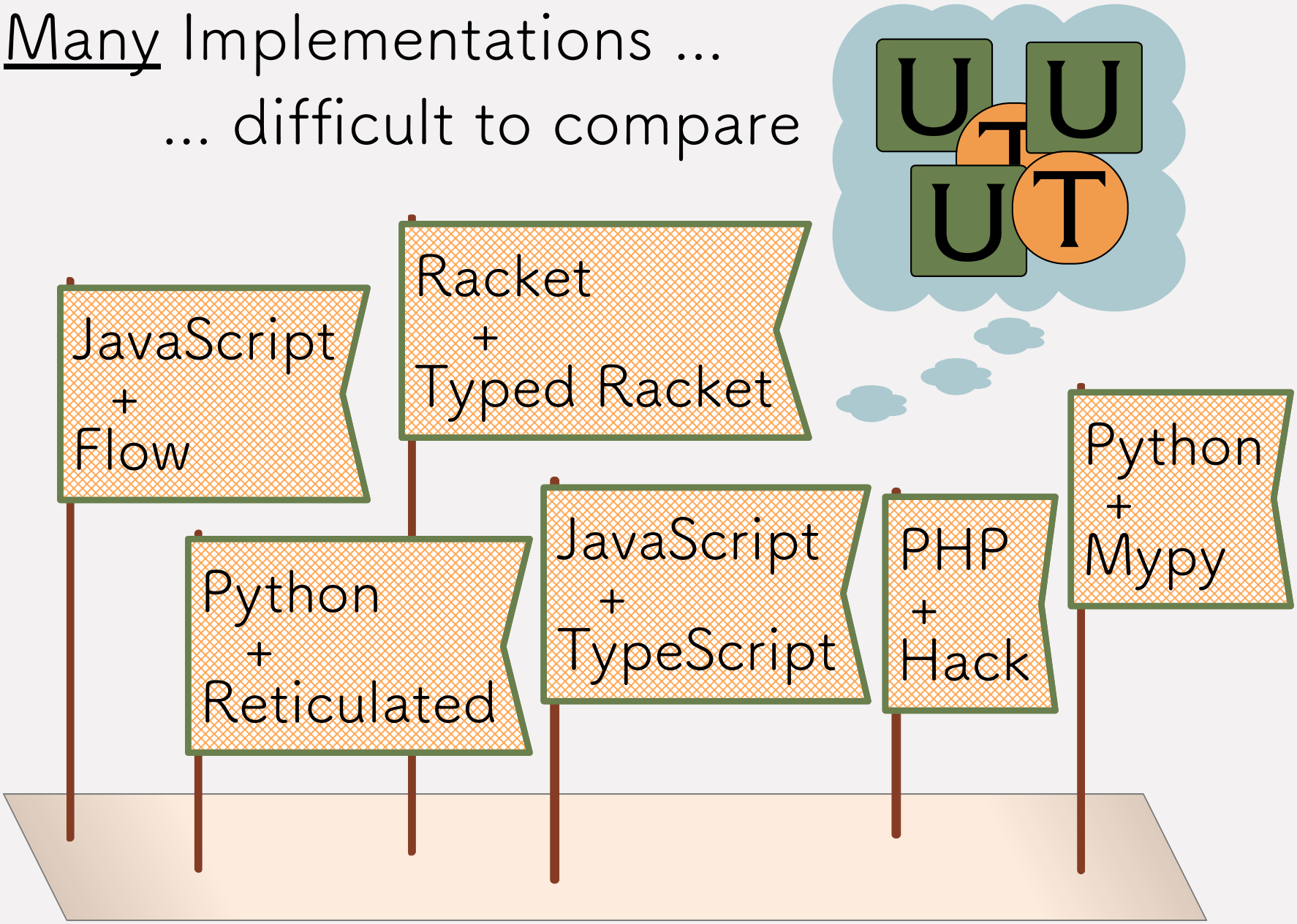
Re-Use

write typed code,
use old libraries

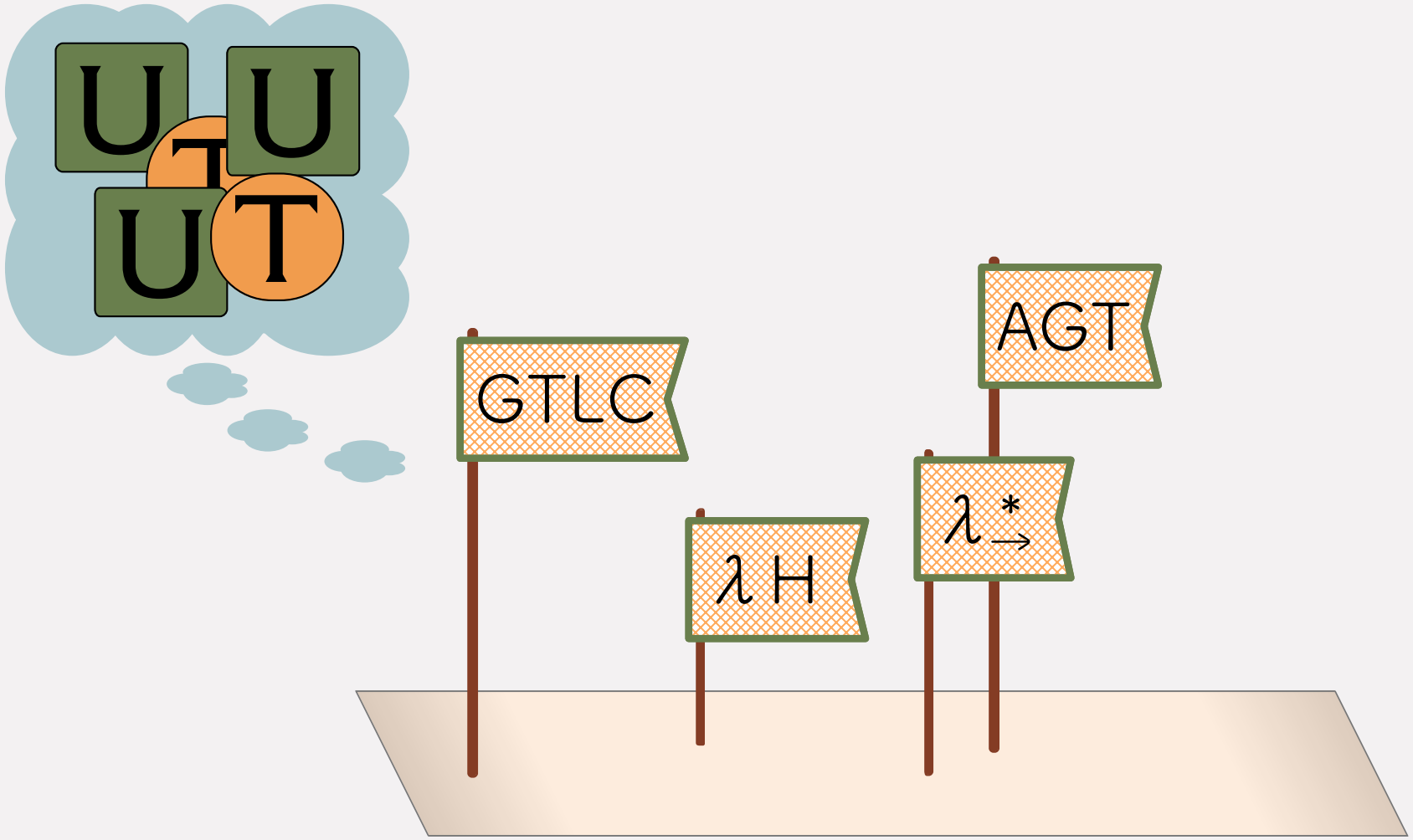
Many Implementations ...



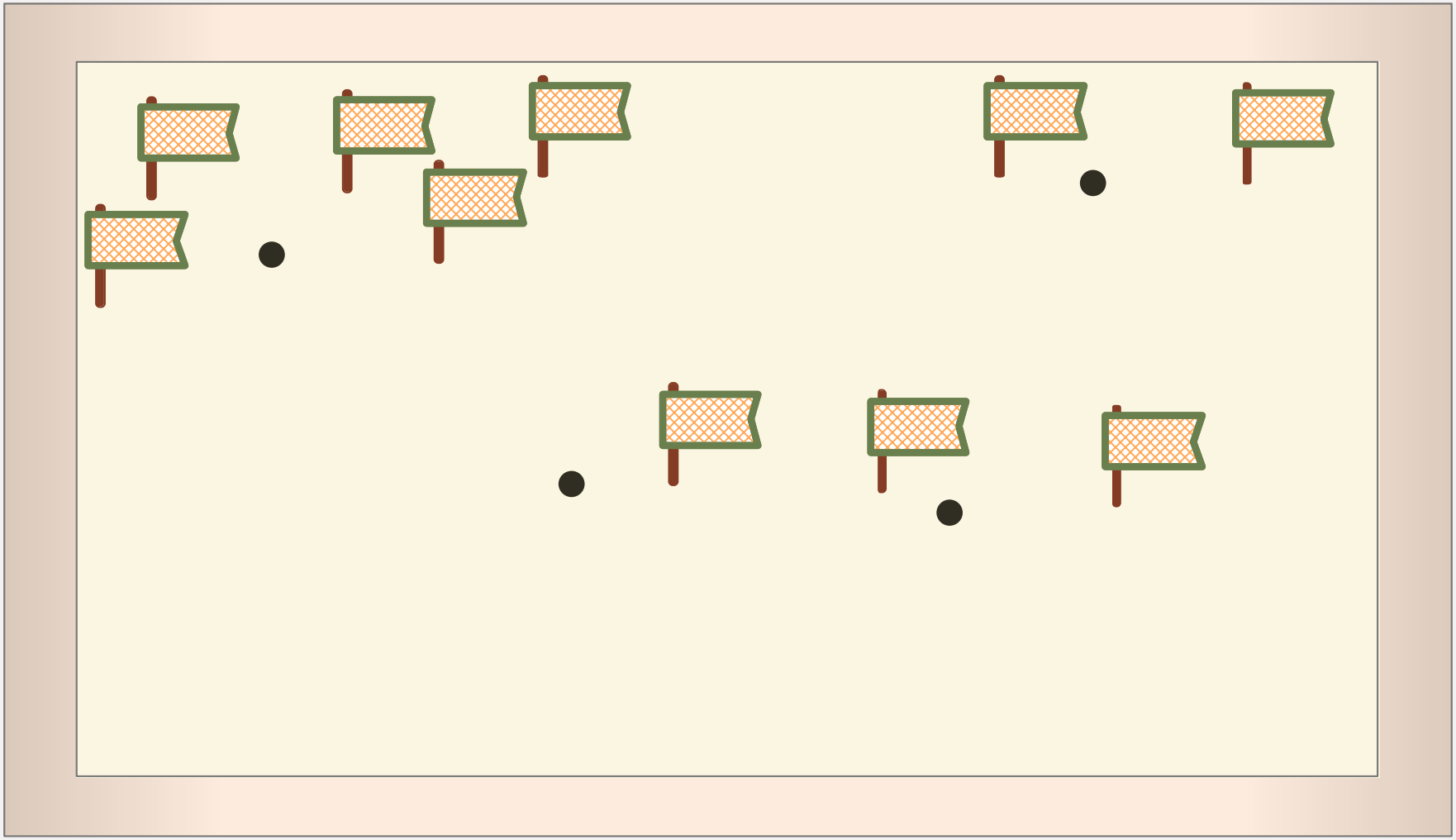
Many Implementations ...
... difficult to compare



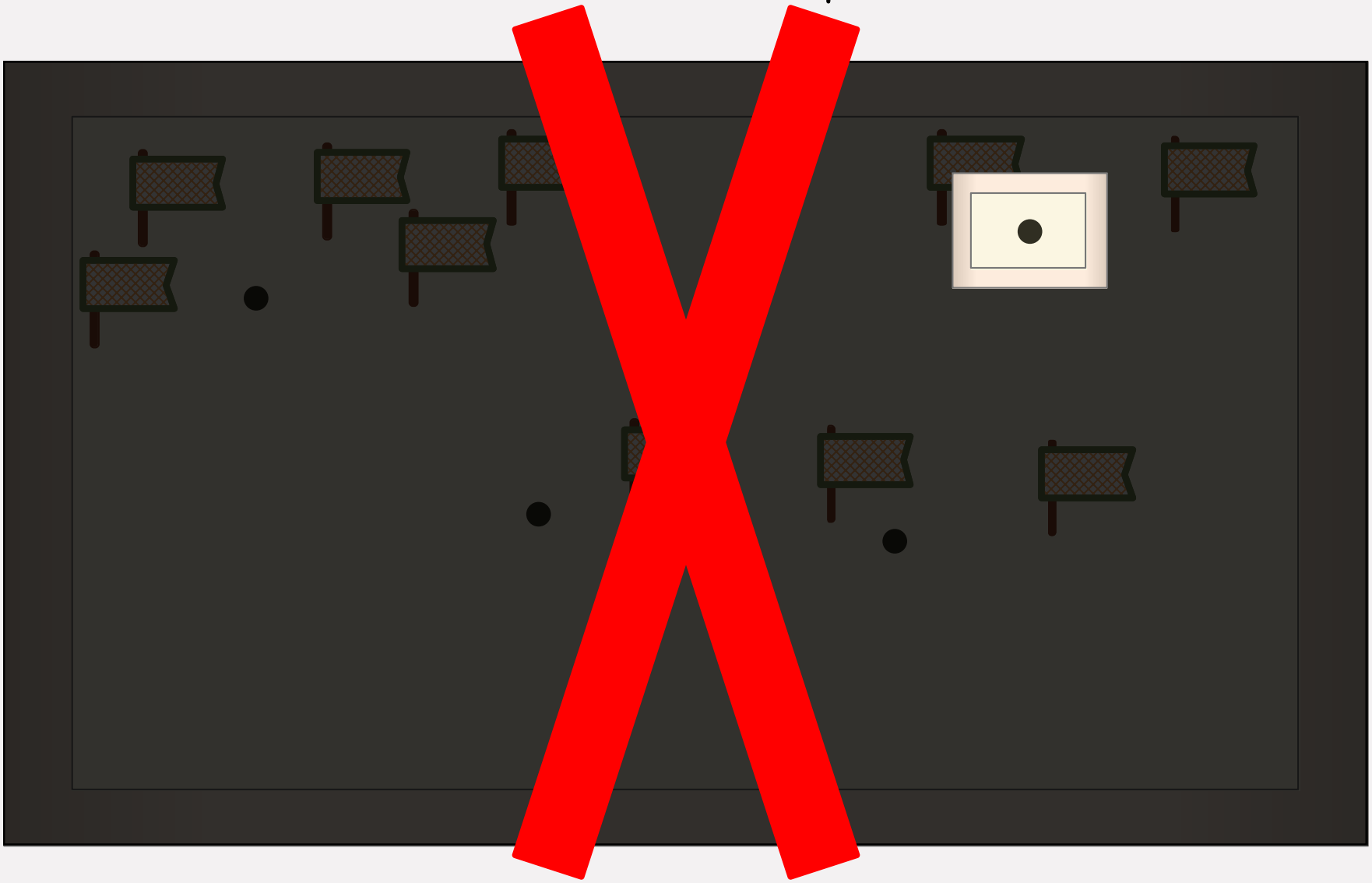
Many Models, too

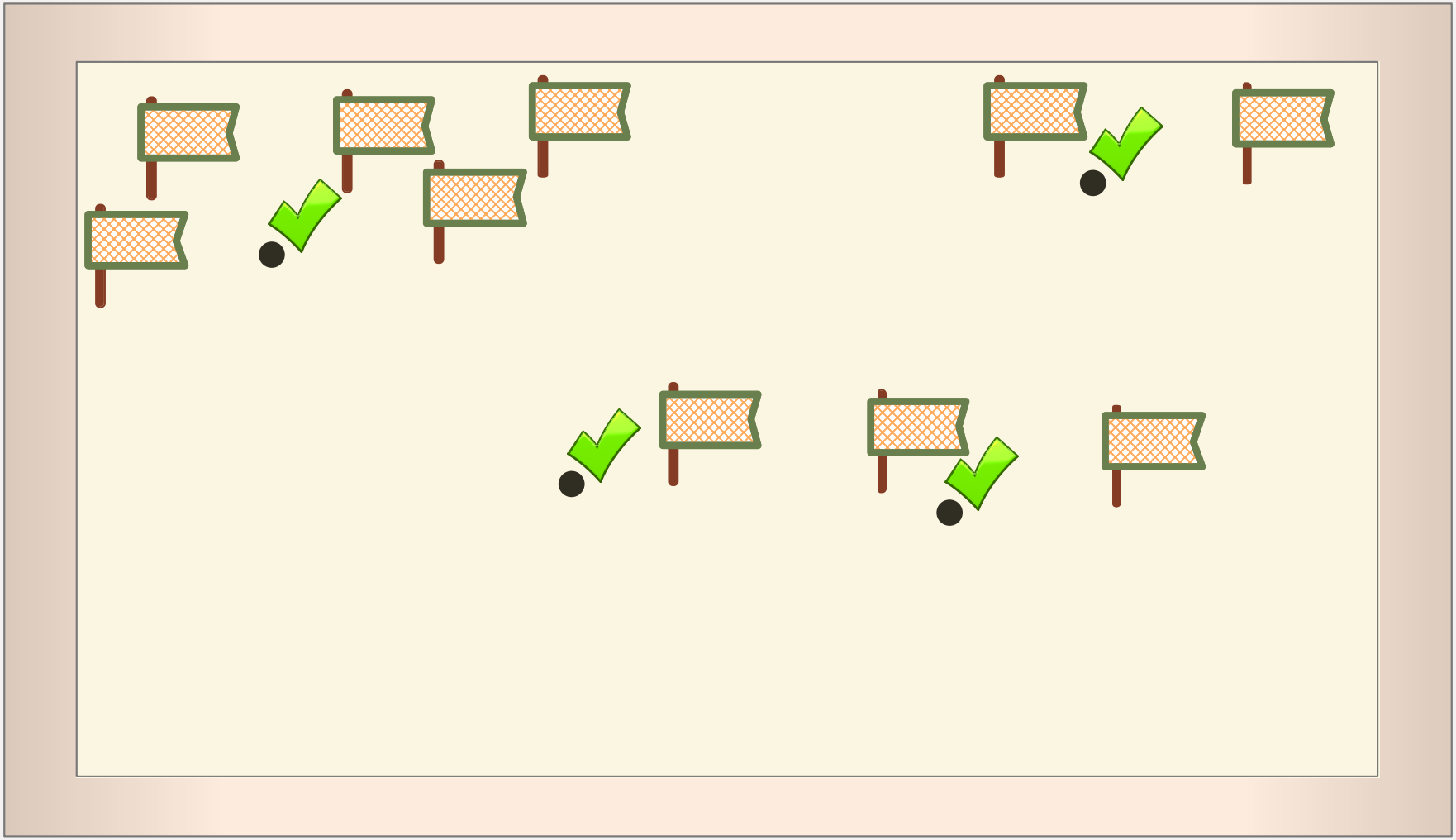


Goal: Characterize the Landscape

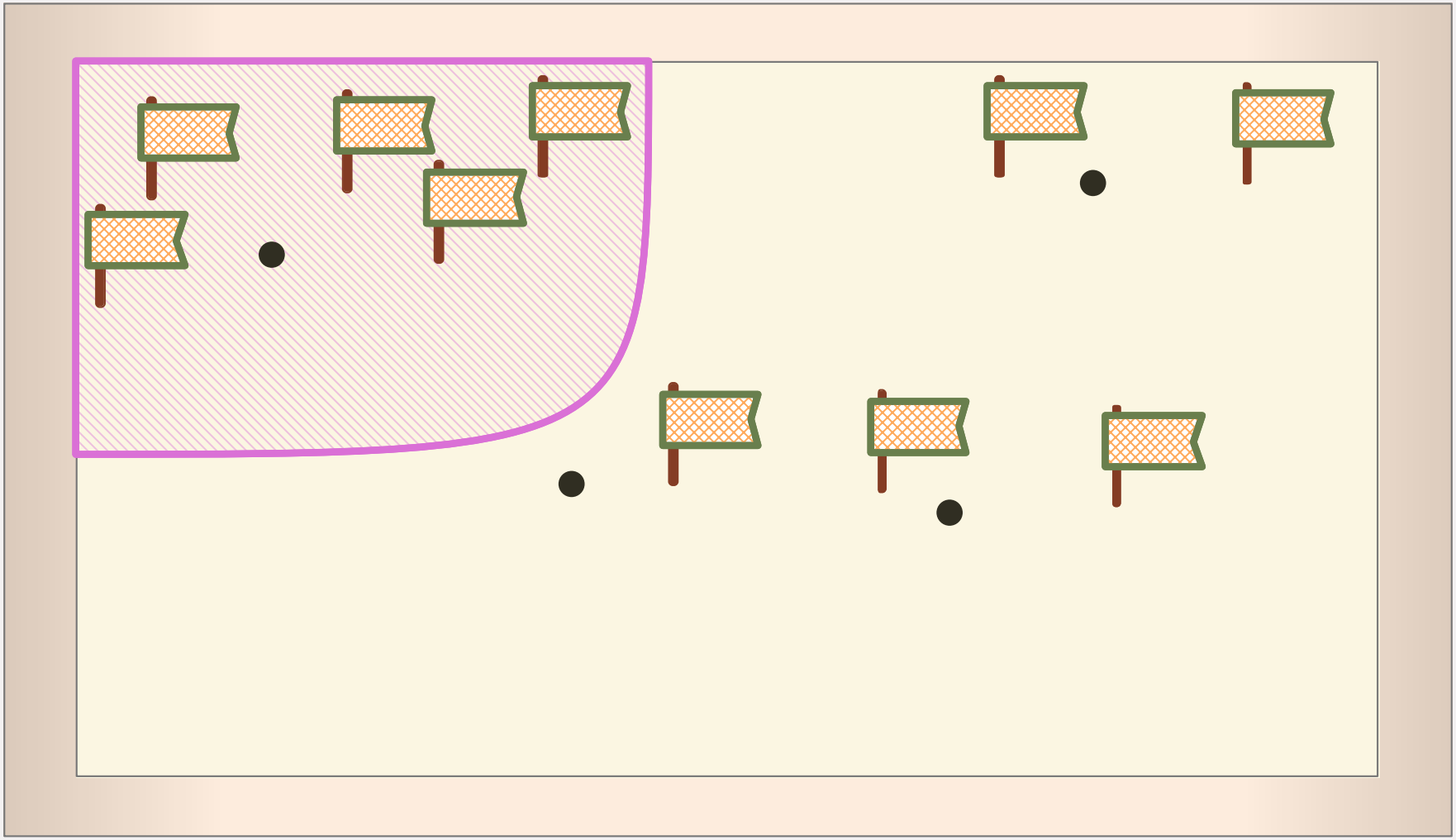


Non-Goal: Restrict Landscape

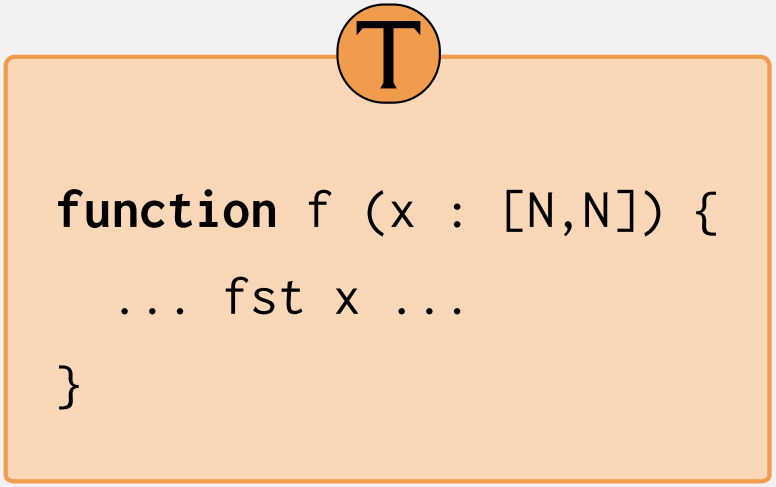




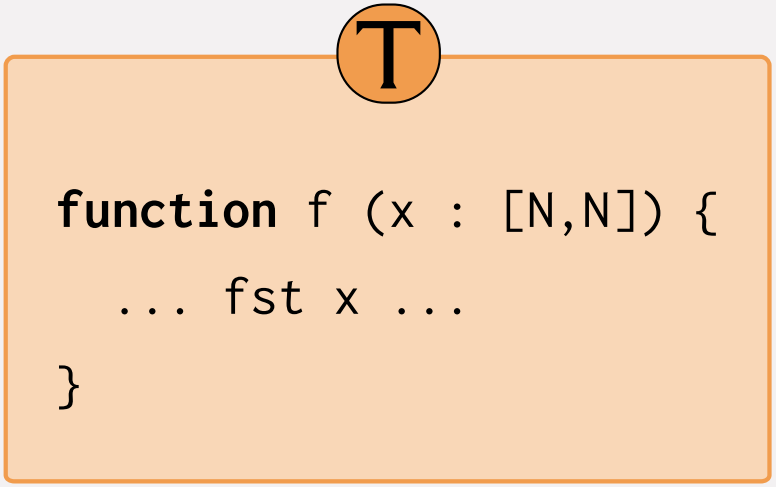
Warmup: Optional Typing / Erasure

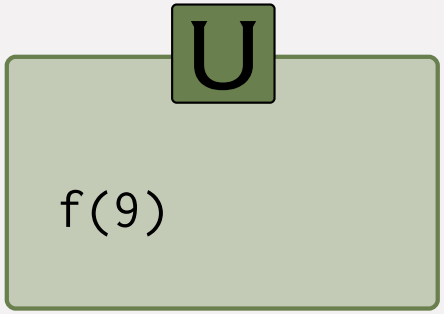


Example: Optional Typing

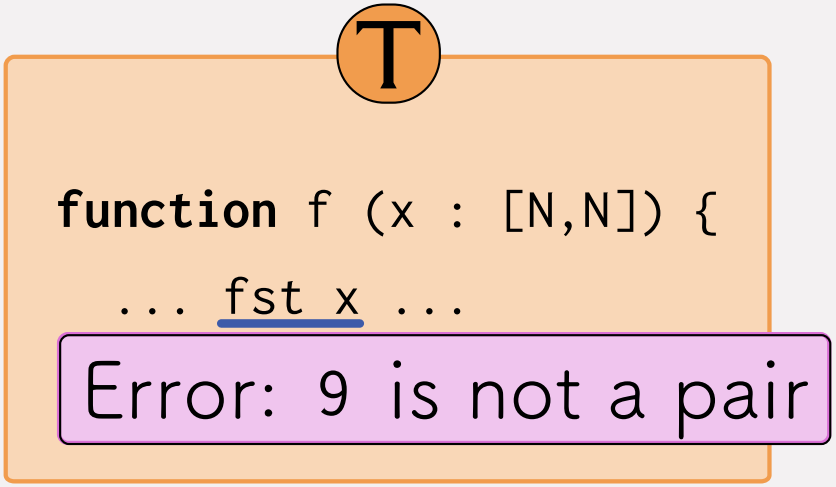


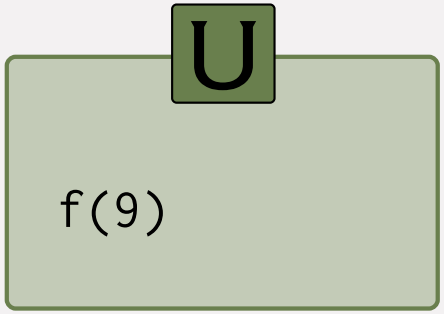
Example: Optional Typing



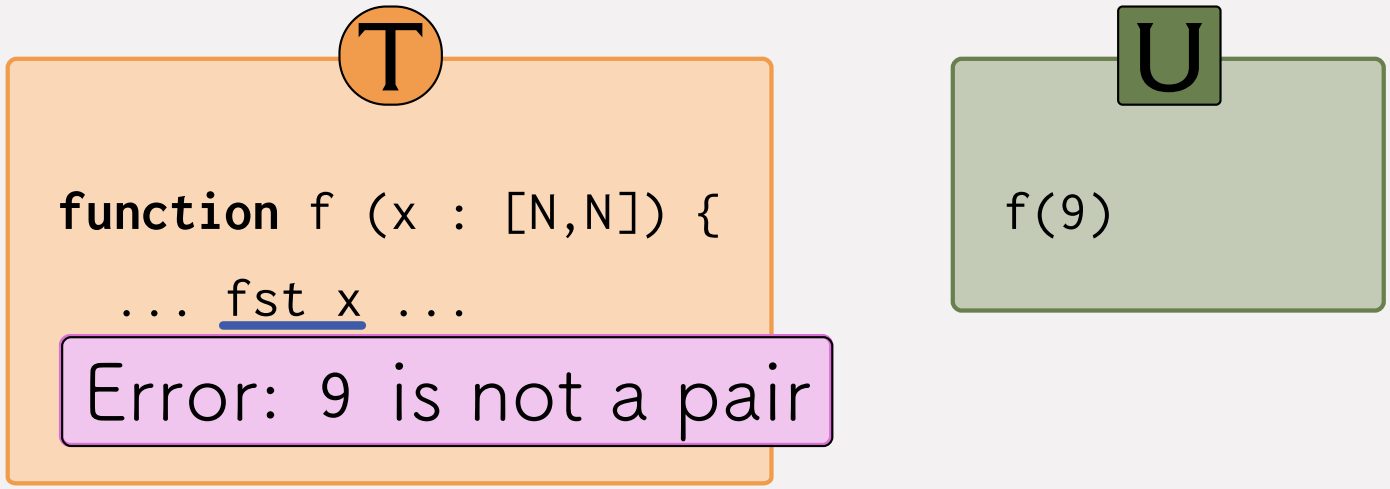


Example: Optional Typing





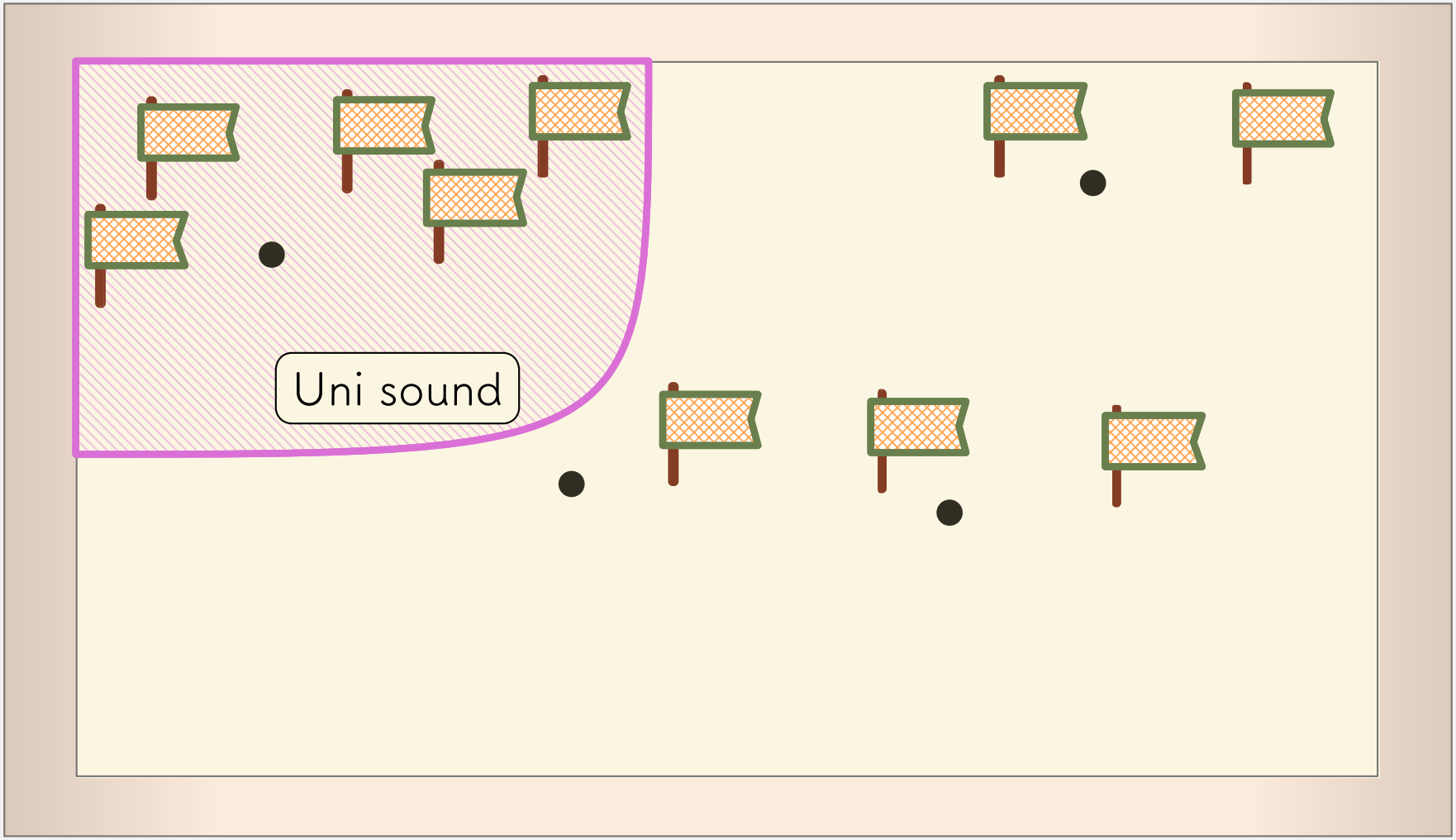
Example: Optional Typing



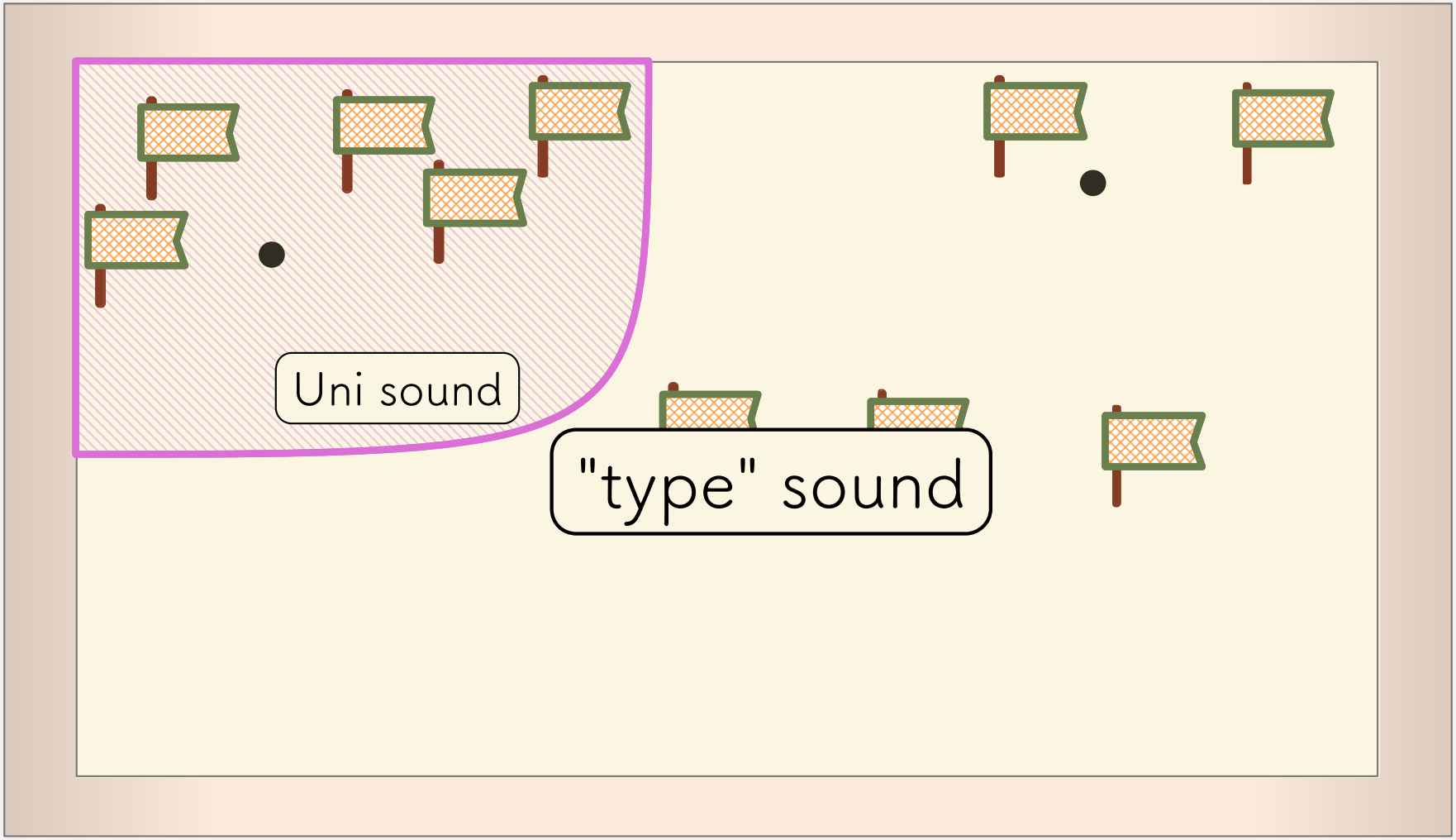
types are **meaningless** at run-time

cannot help debug a faulty program

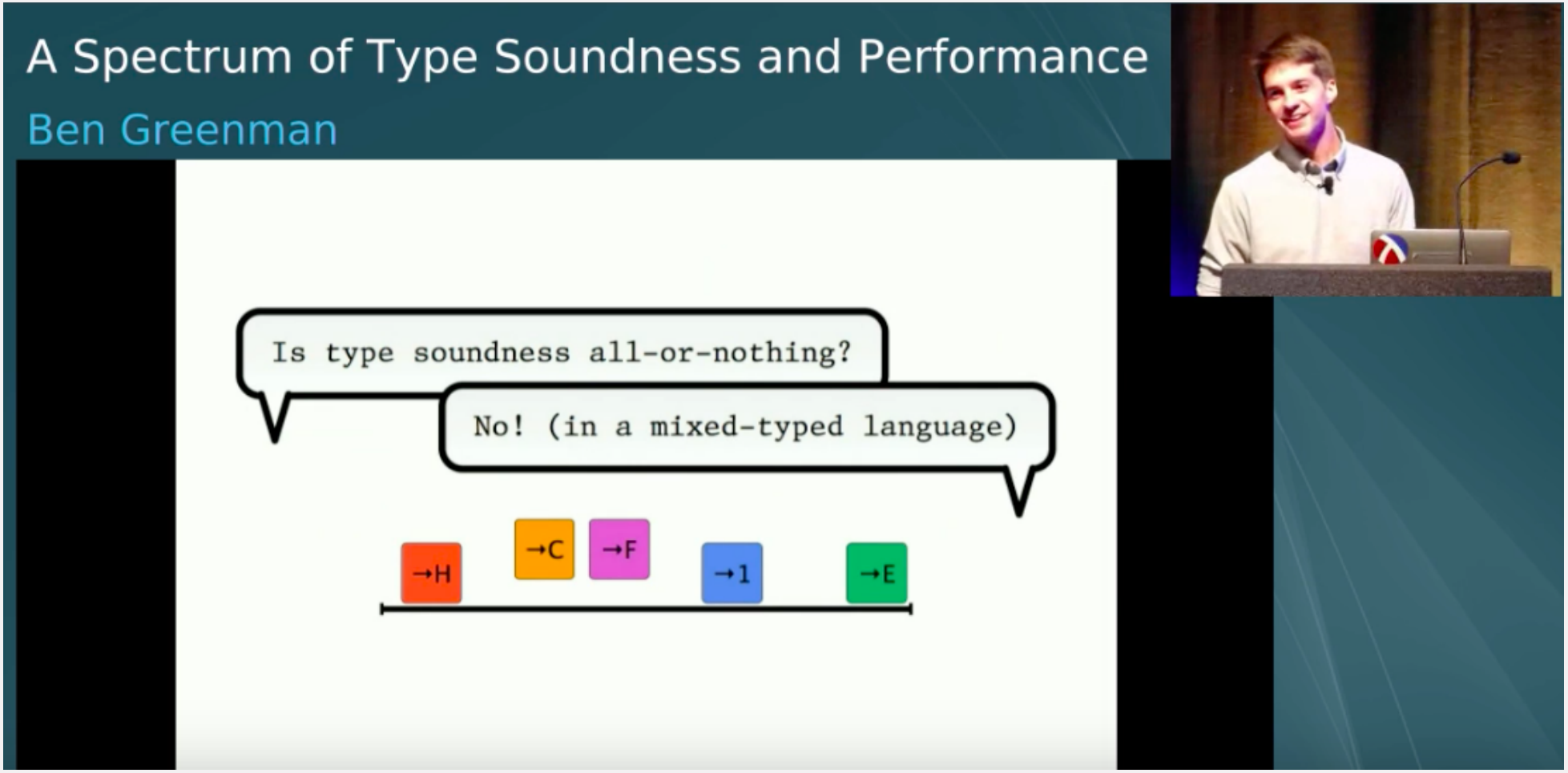
= Does Not Preserve Types



= Does Not Preserve Types



ICFP '18


A Spectrum of Type Soundness and Performance
Ben Greenman
Is type soundness all-or-nothing?
No! (in a mixed-typed language)
→H
→C
→F
→1
→E

ICFP '18 : Three Semantics, Soundnesses

ICFP '18 : Three Semantics, Soundnesses

Erasure semantics

- types predict nothing

ICFP '18 : Three Semantics, Soundnesses

Erasure semantics

- types predict nothing

Transient semantics

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

ICFP '18 : Three Semantics, Soundnesses

Erasure semantics
- types predict nothing

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

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

ICFP '18 : Three Semantics, Soundnesses

Erasure semantics (Uni sound)

- types predict nothing

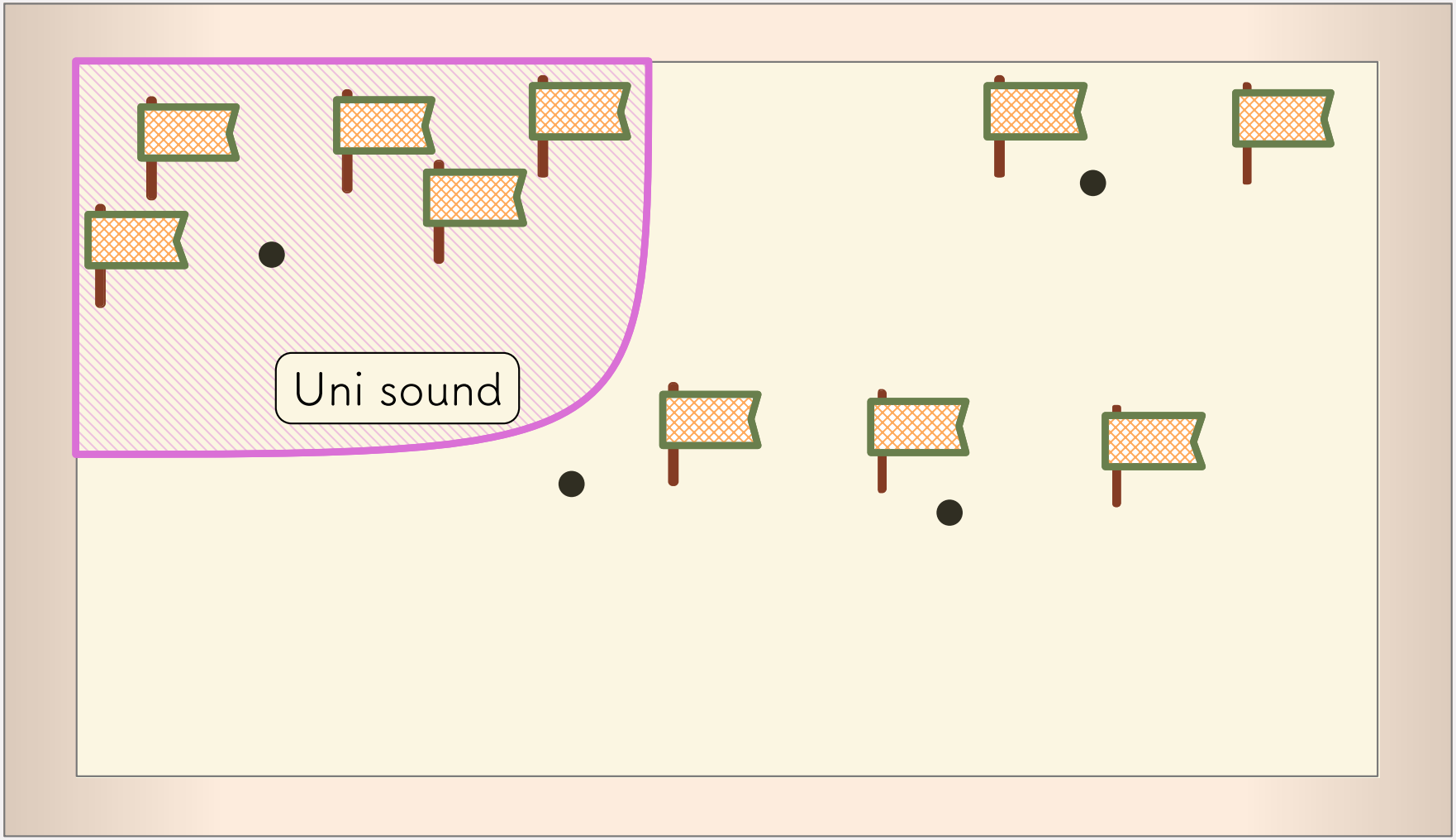
Transient semantics (⌊T⌋ sound)

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

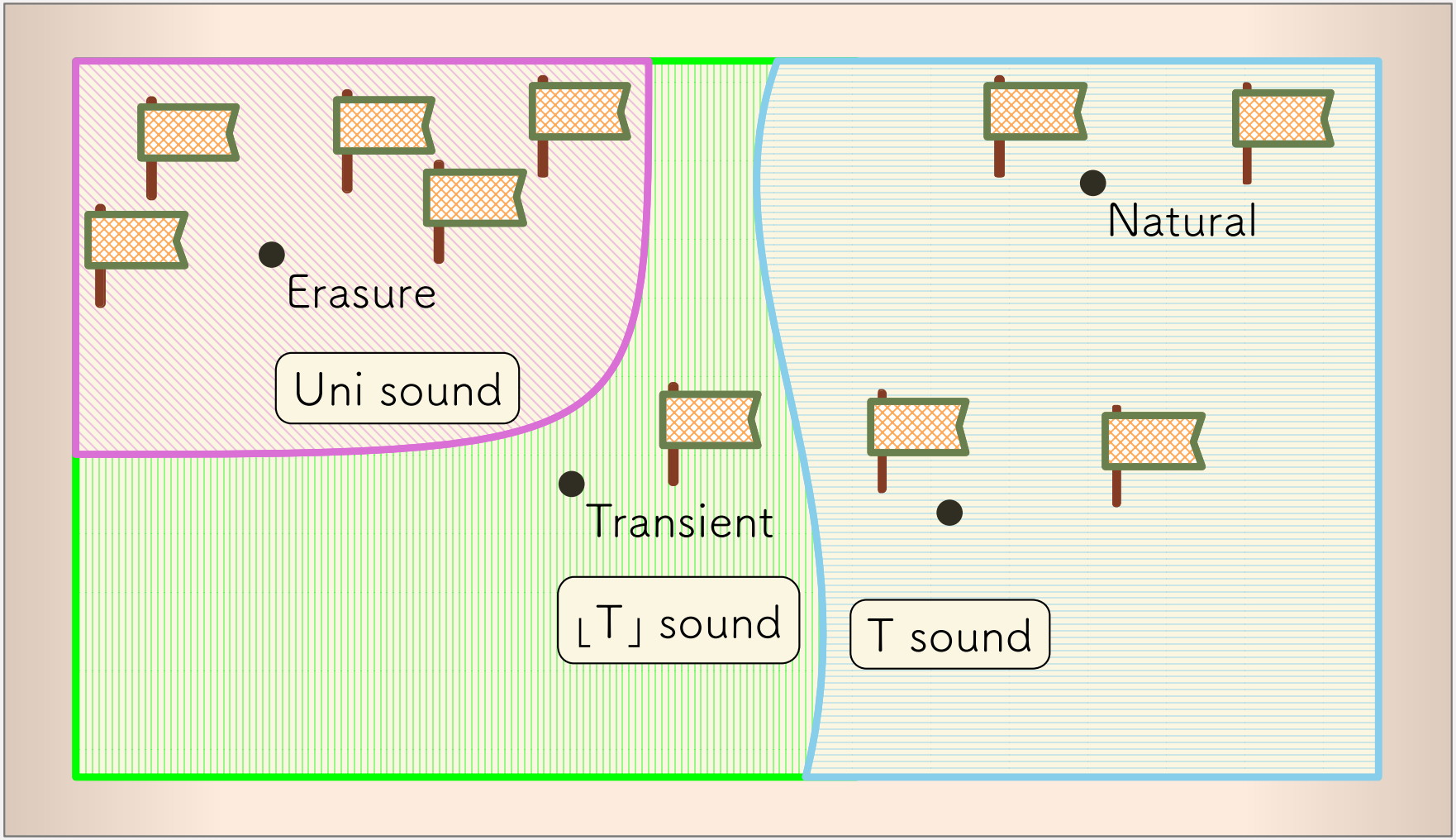
Natural semantics (T sound)

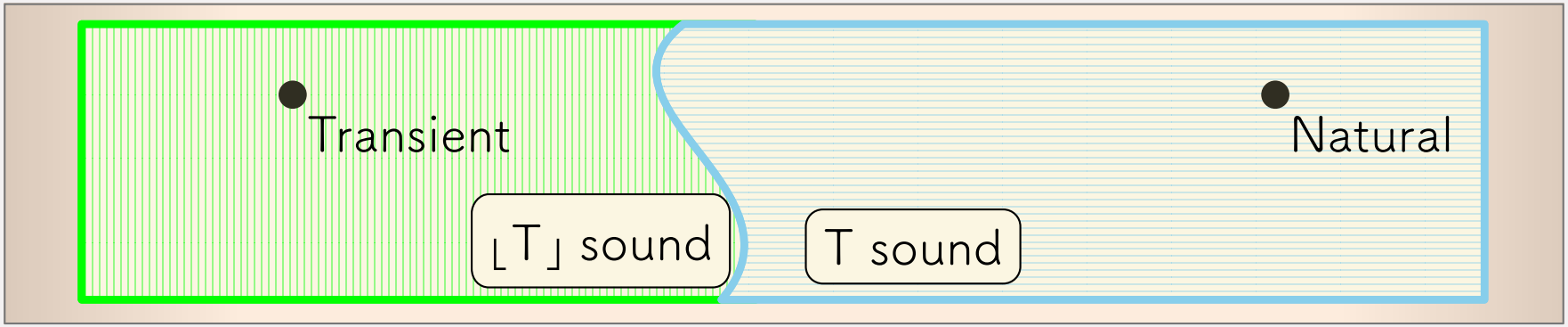
- types predict the full behavior of values
- enforced by higher-order **wrappers**

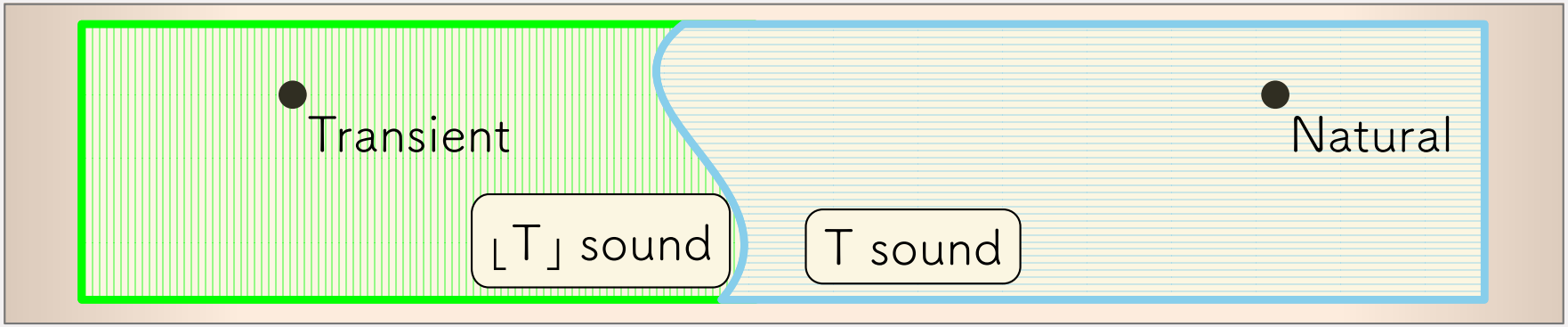
ICFP '18: A Spectrum of Type Soundness



ICFP '18: A Spectrum of Type Soundness




Transient
Natural
「T」sound
T sound

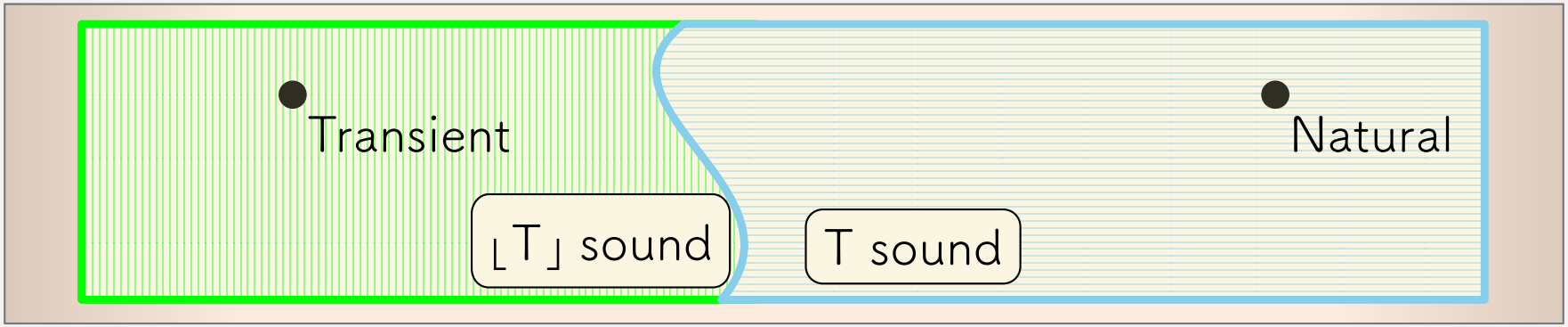


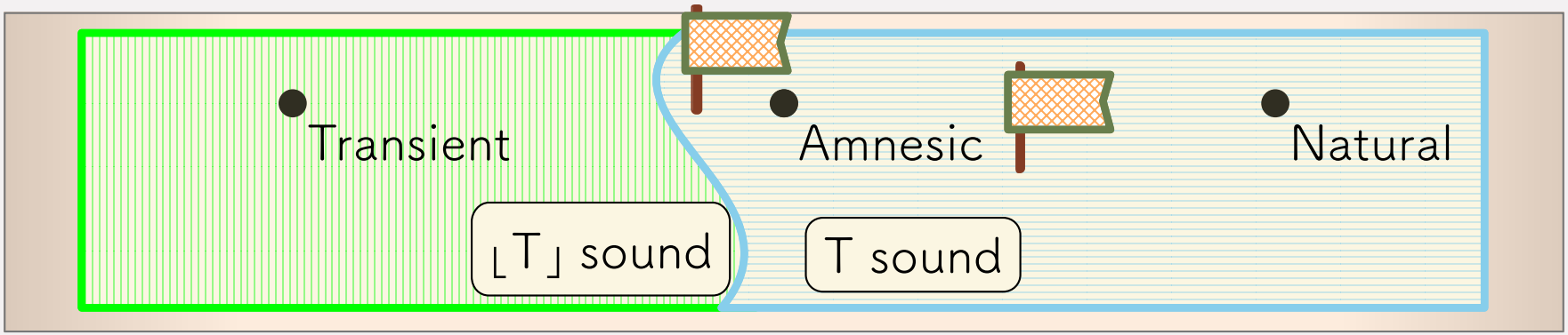
Transient semantics ⌊T⌋ sound

- types predict the top-level shape of values
- 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

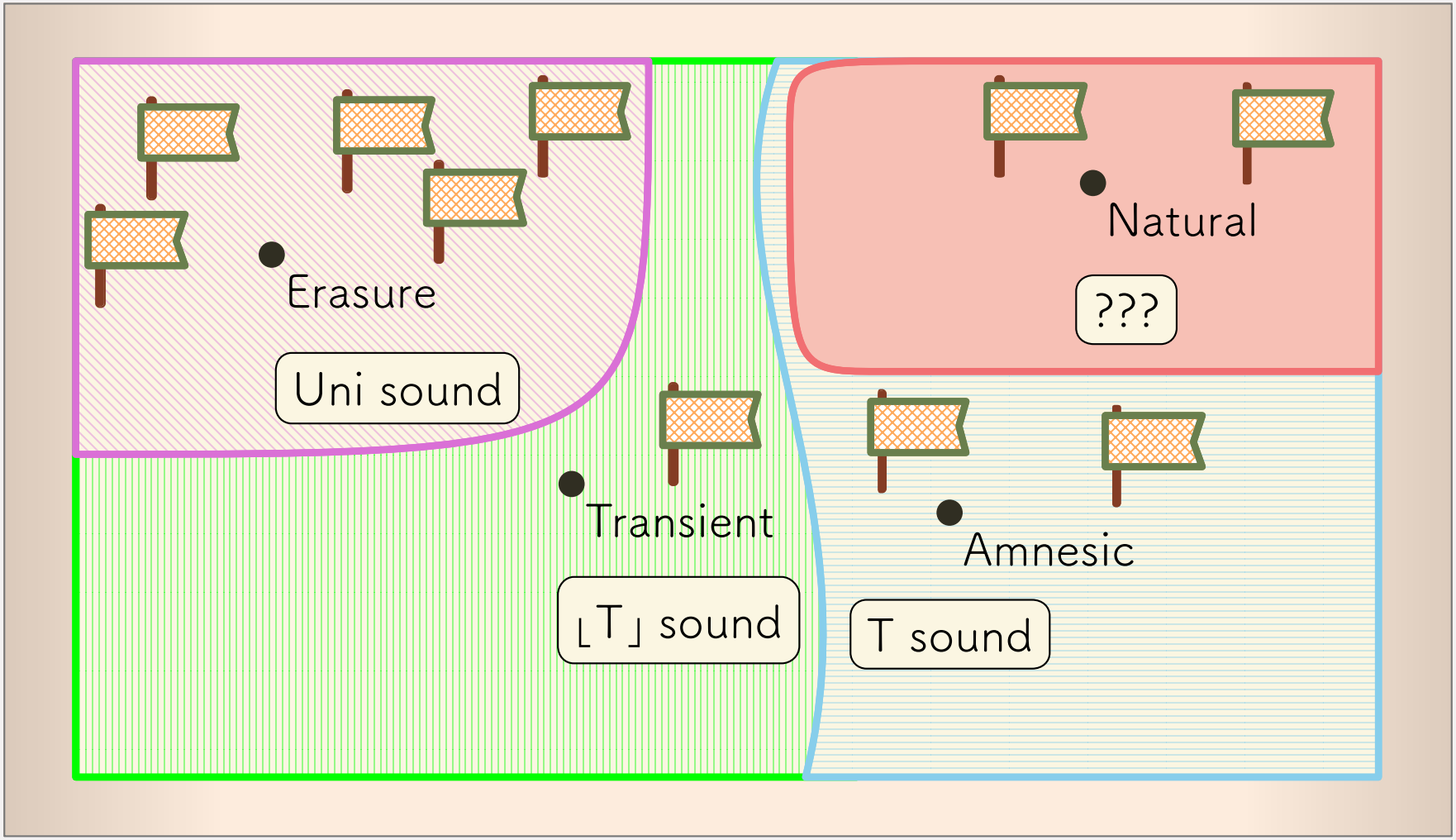
Amnesic semantics

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

Greenberg POPL '15

Castagna, Lanvin ICFP '17

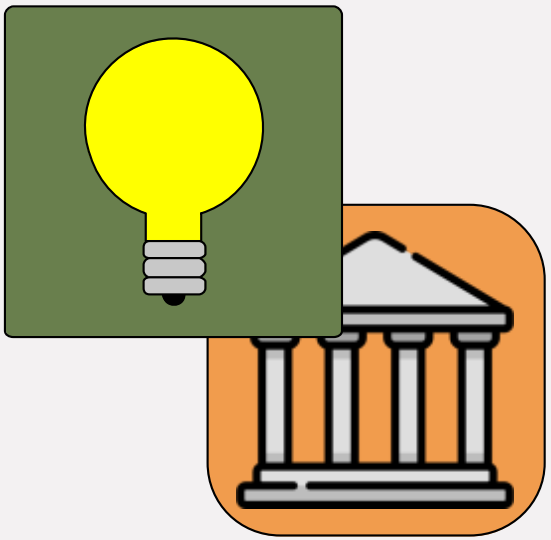
Type Soundness is NOT ENOUGH



Example: Transient/Amnesic vs. Natural

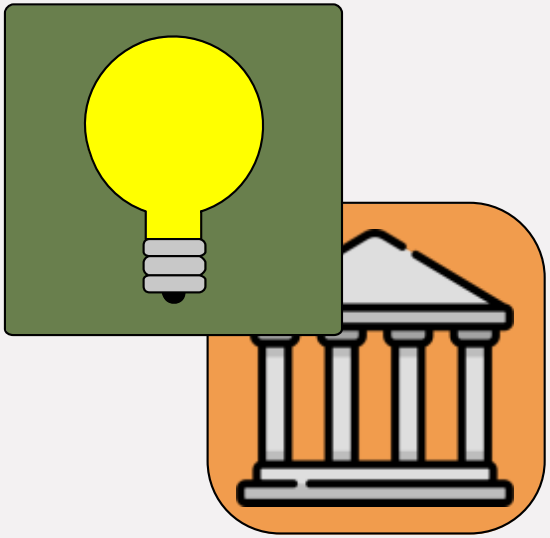
Example: Transient/Amnesic vs. Natural

Prototyping



Example: Transient/Amnesic vs. Natural

Prototyping

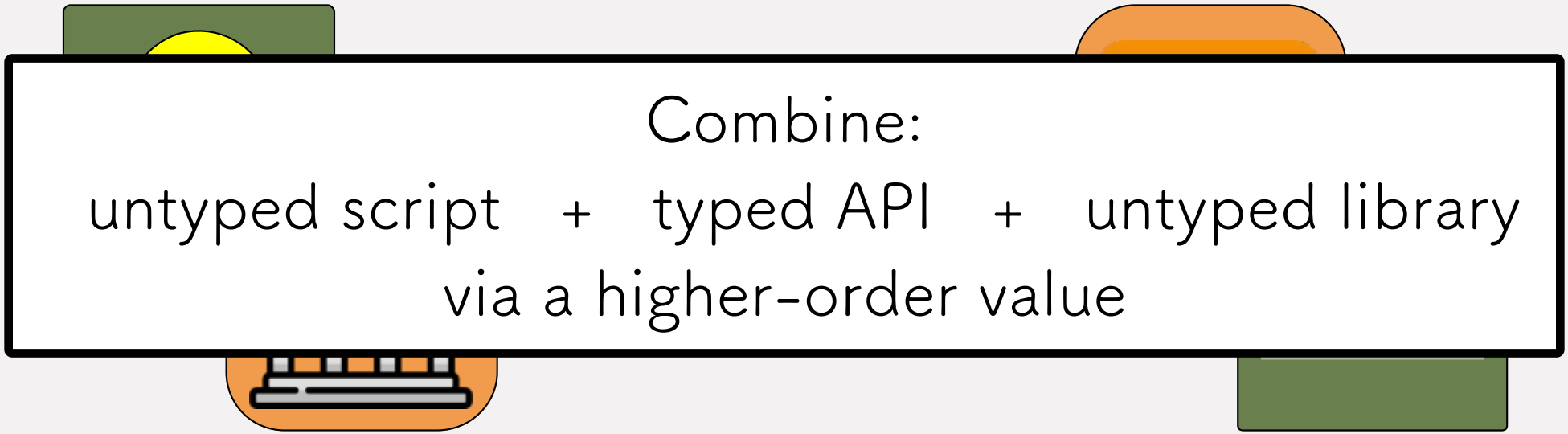


Library Re-Use

Example: Transient/Amnesic vs. Natural

Prototyping

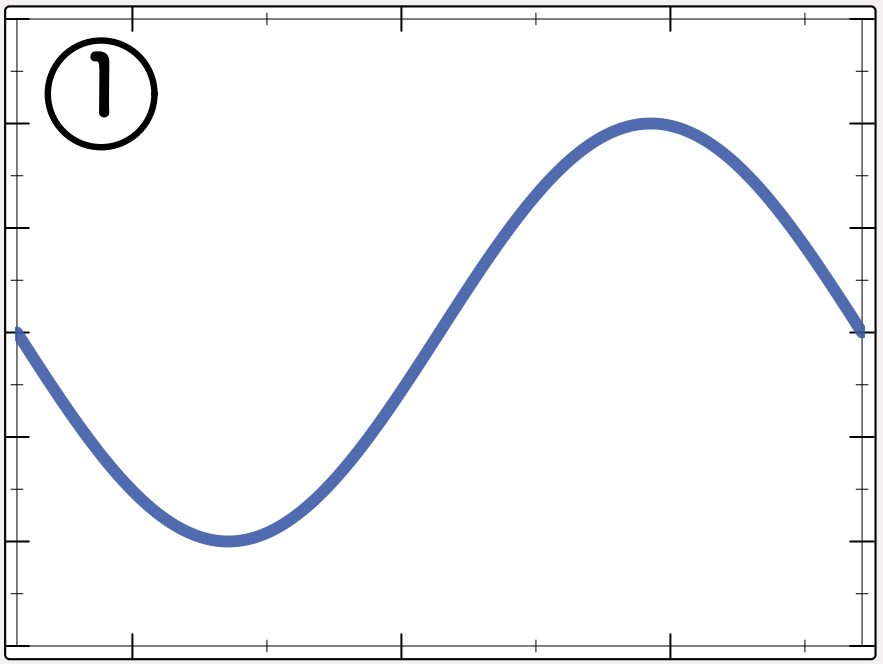
Library Re-Use



Example: Transient/Amnesic vs. Natural

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

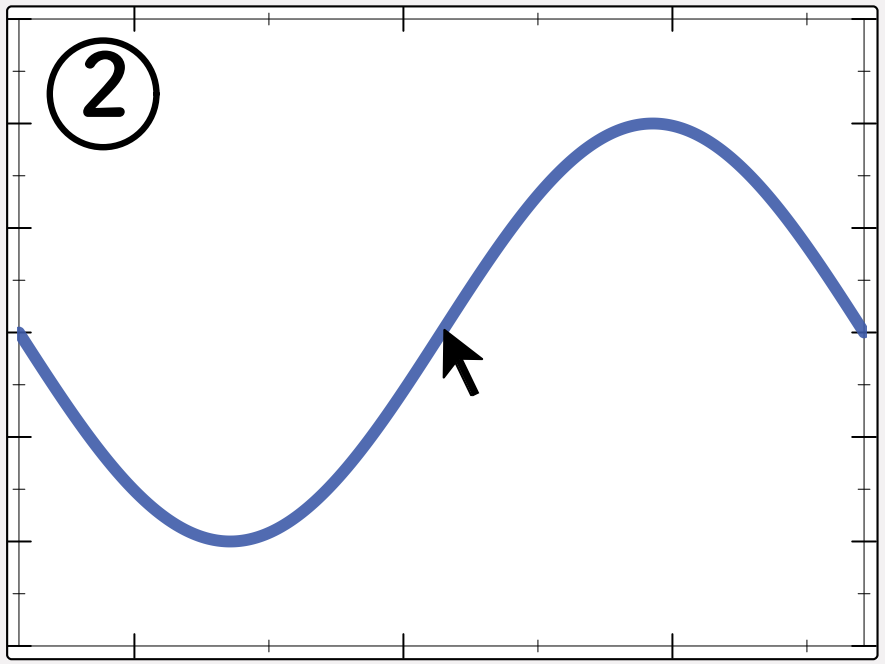
Clickable Plot



Example: Transient/Amnesic vs. Natural

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

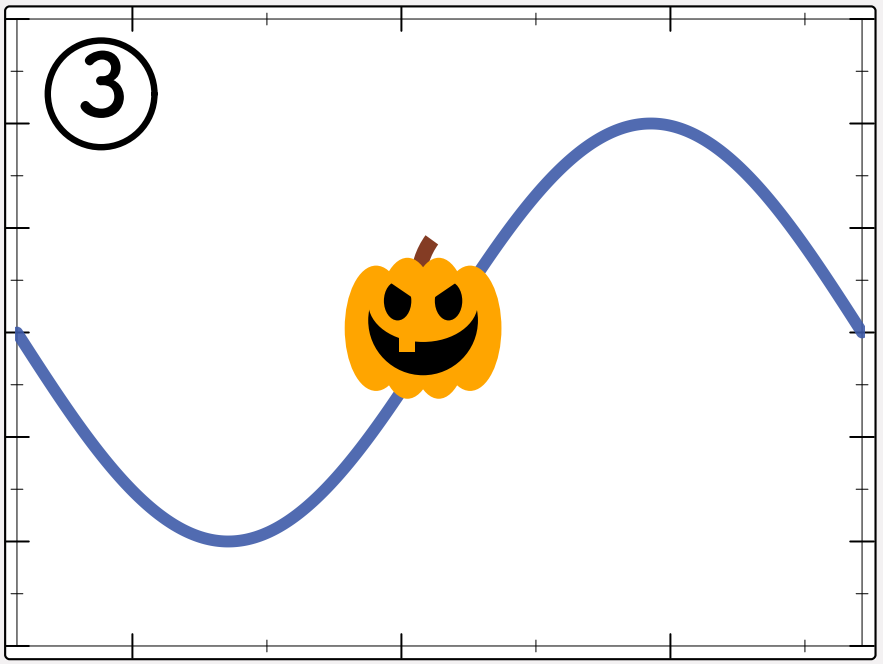
Clickable Plot



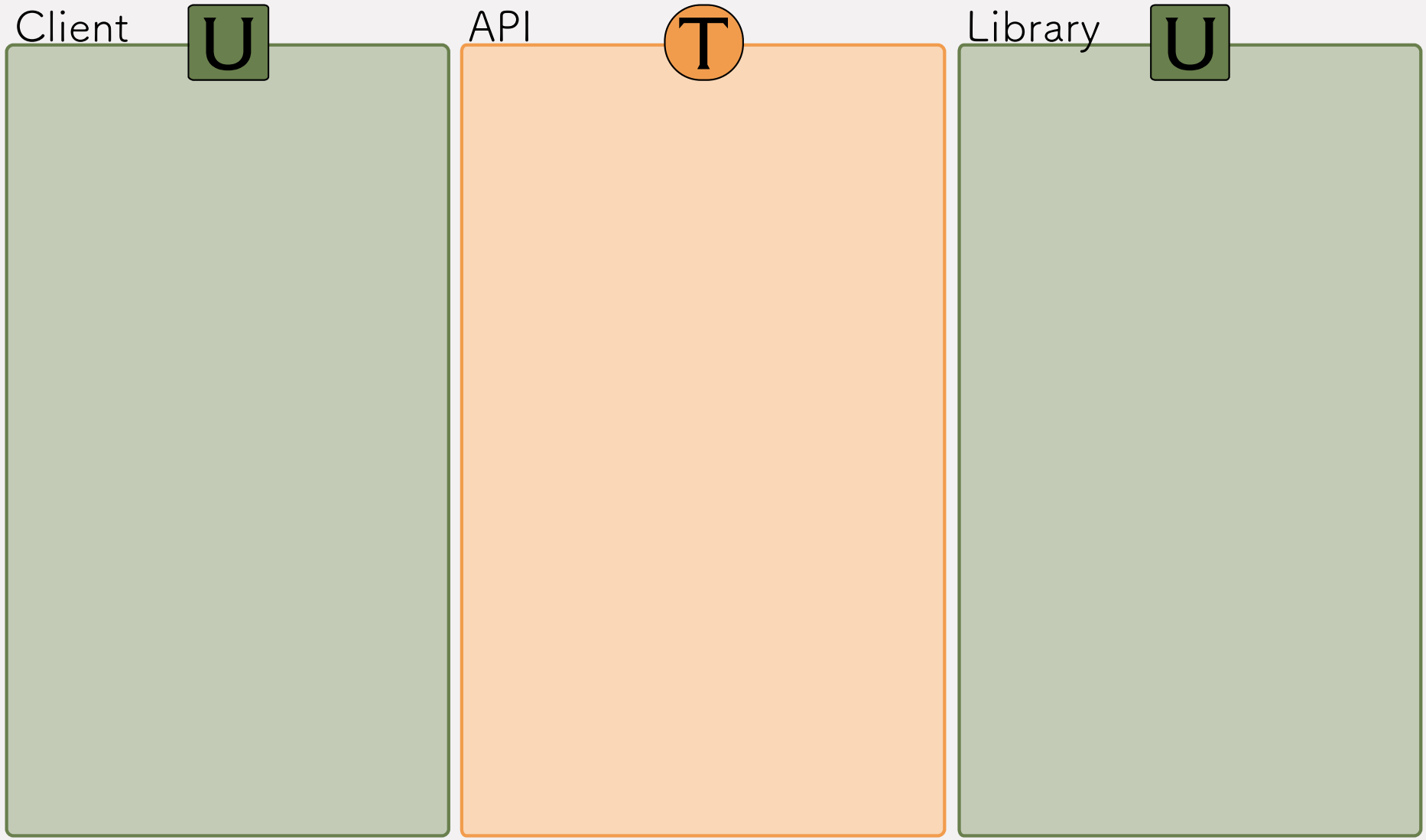
Example: Transient/Amnesic vs. Natural

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

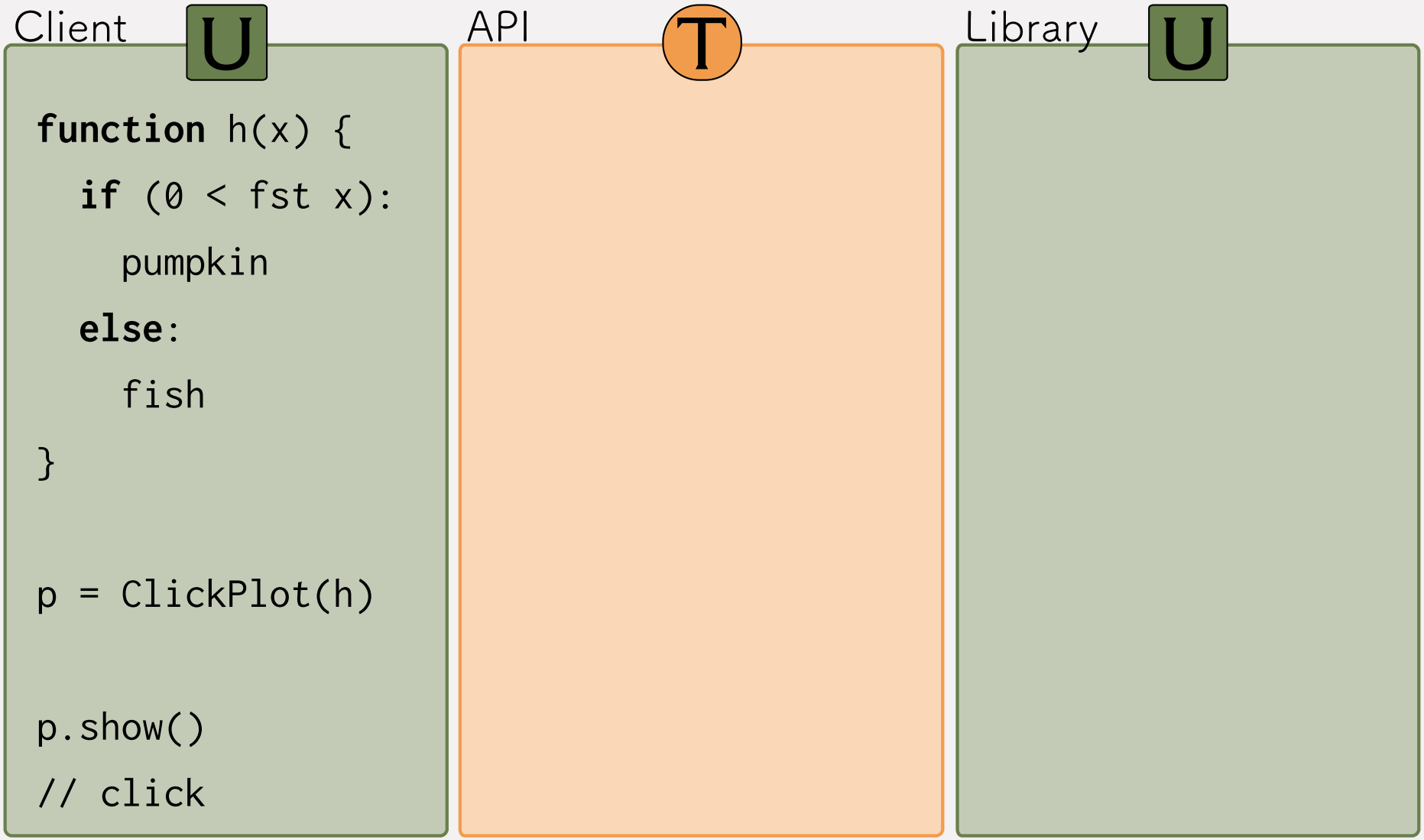
Clickable Plot



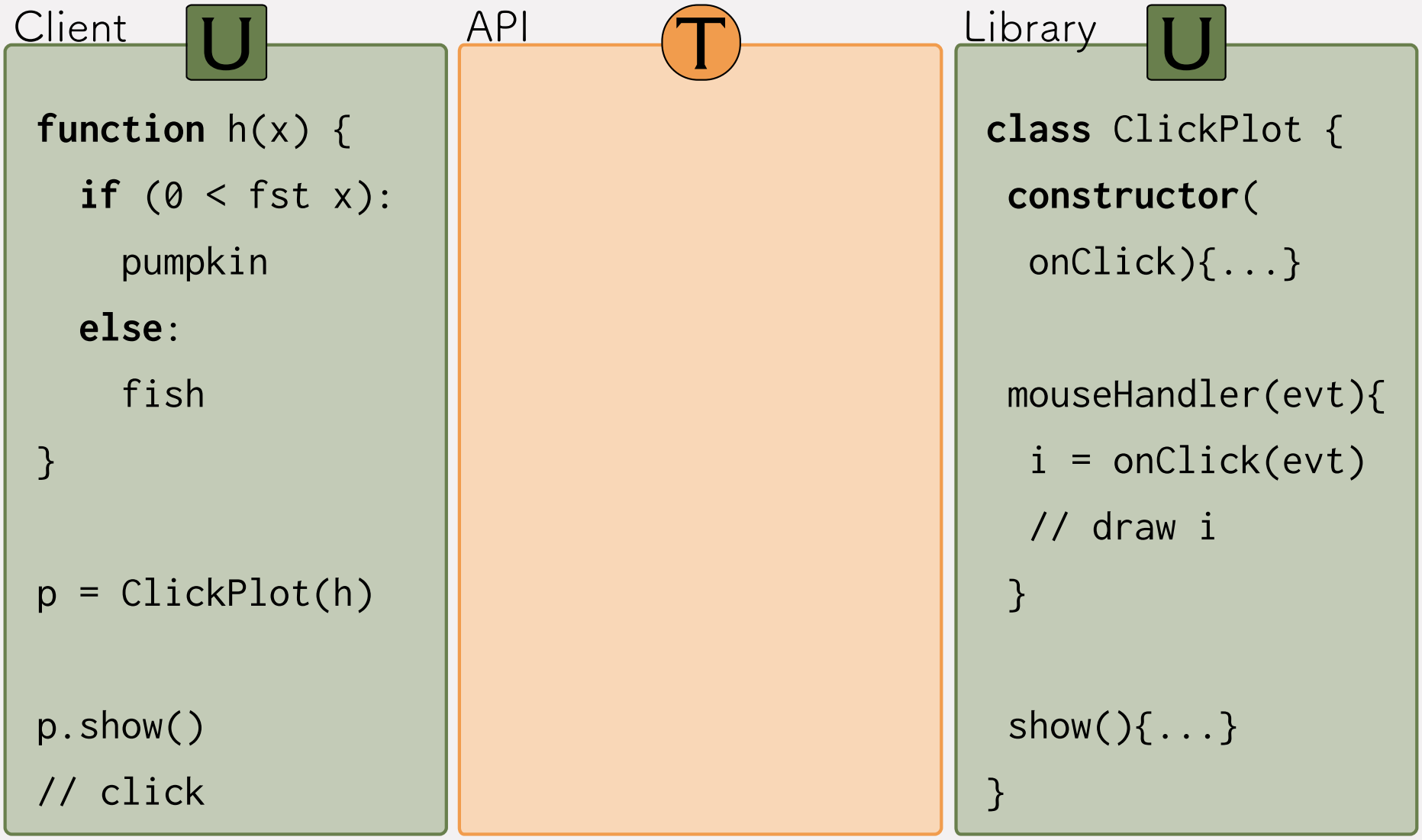
Example: interactive plot



Example: interactive plot



Example: interactive plot



Example: interactive plot

Client U

```
function h(x) {
  if (0 < fst x):
    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){
  i = onClick(evt)
  // draw i
 }

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

Example: interactive plot

Client U

```
function h(x) {
  if (0 < fst x):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

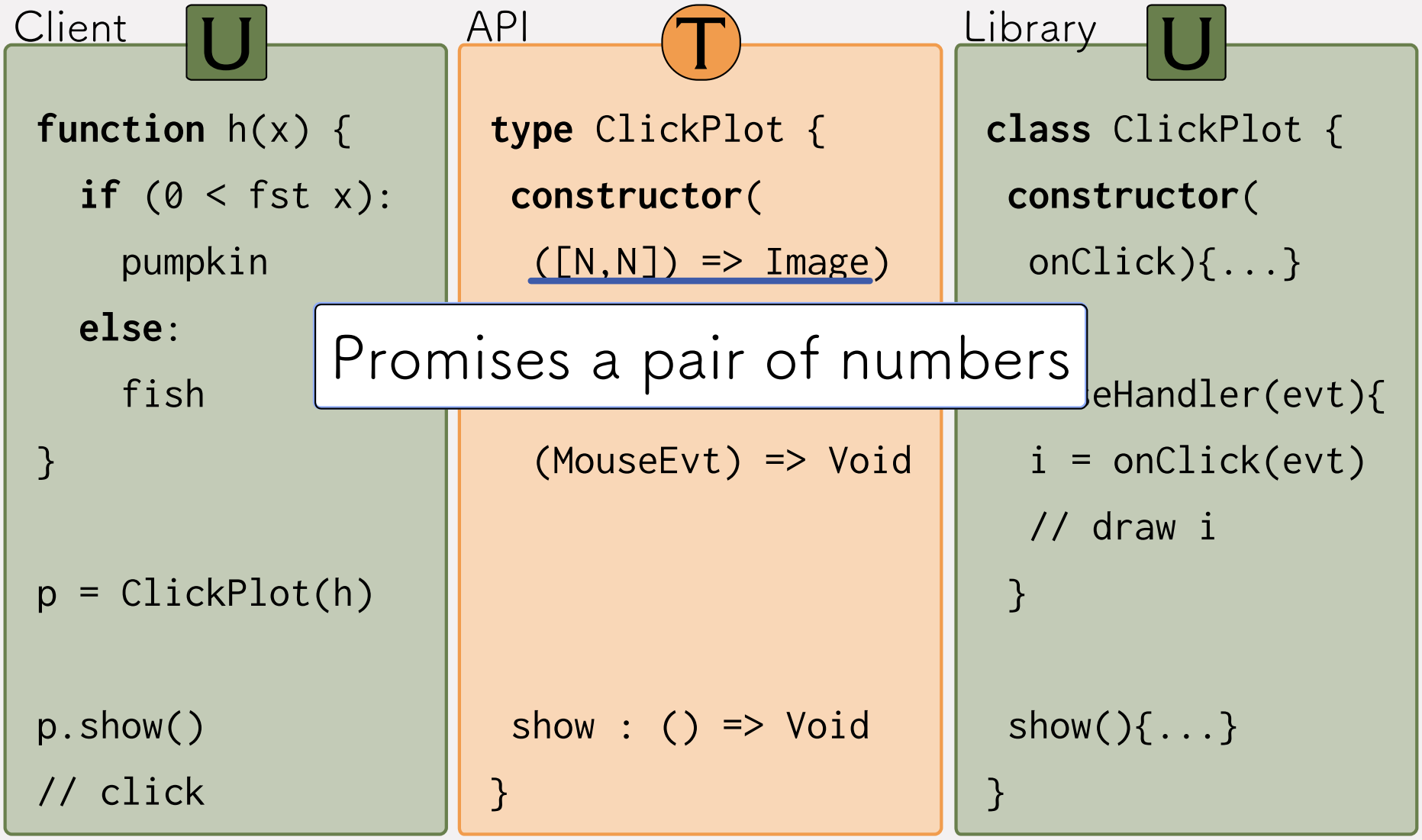
API T

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

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

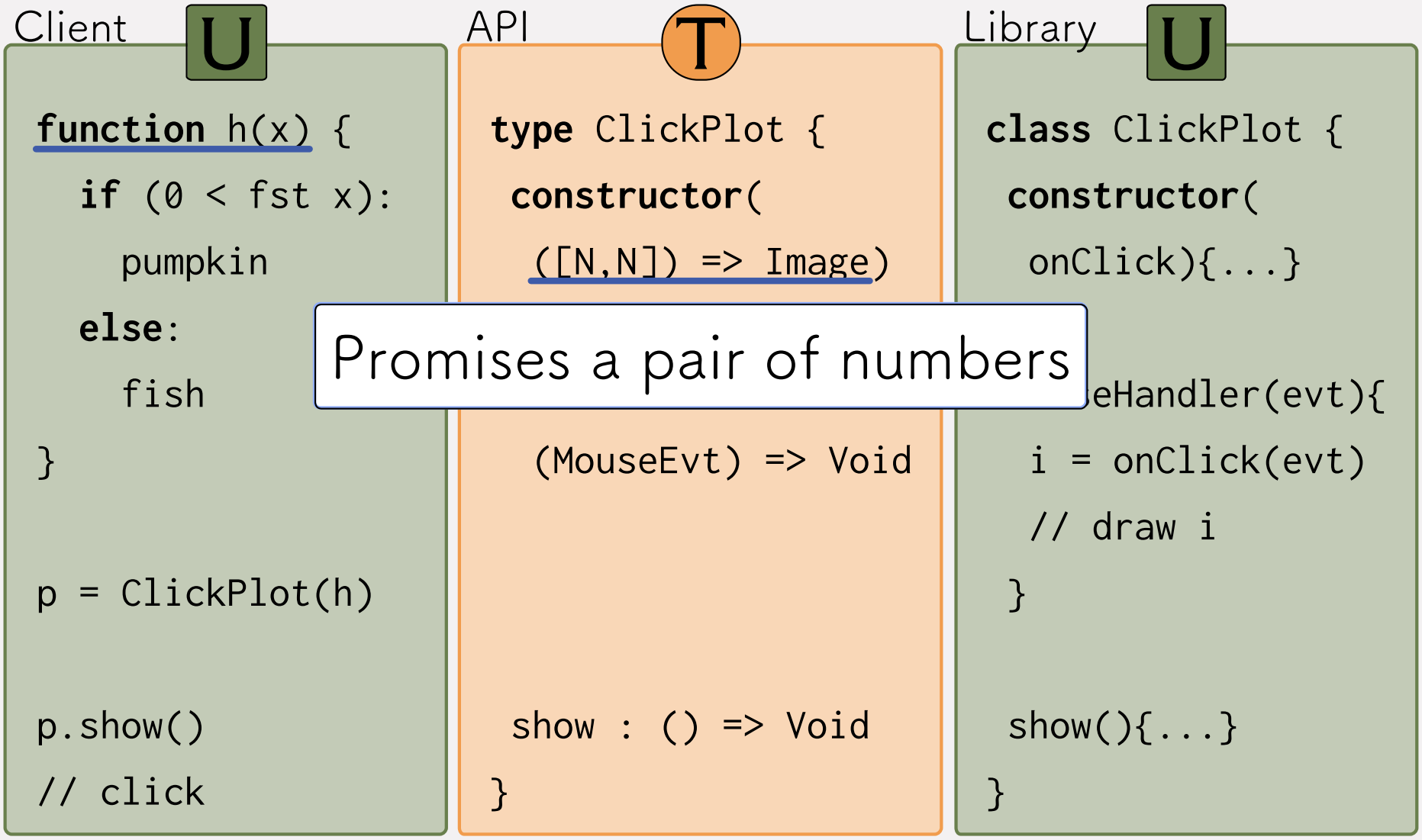
Library U

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

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

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

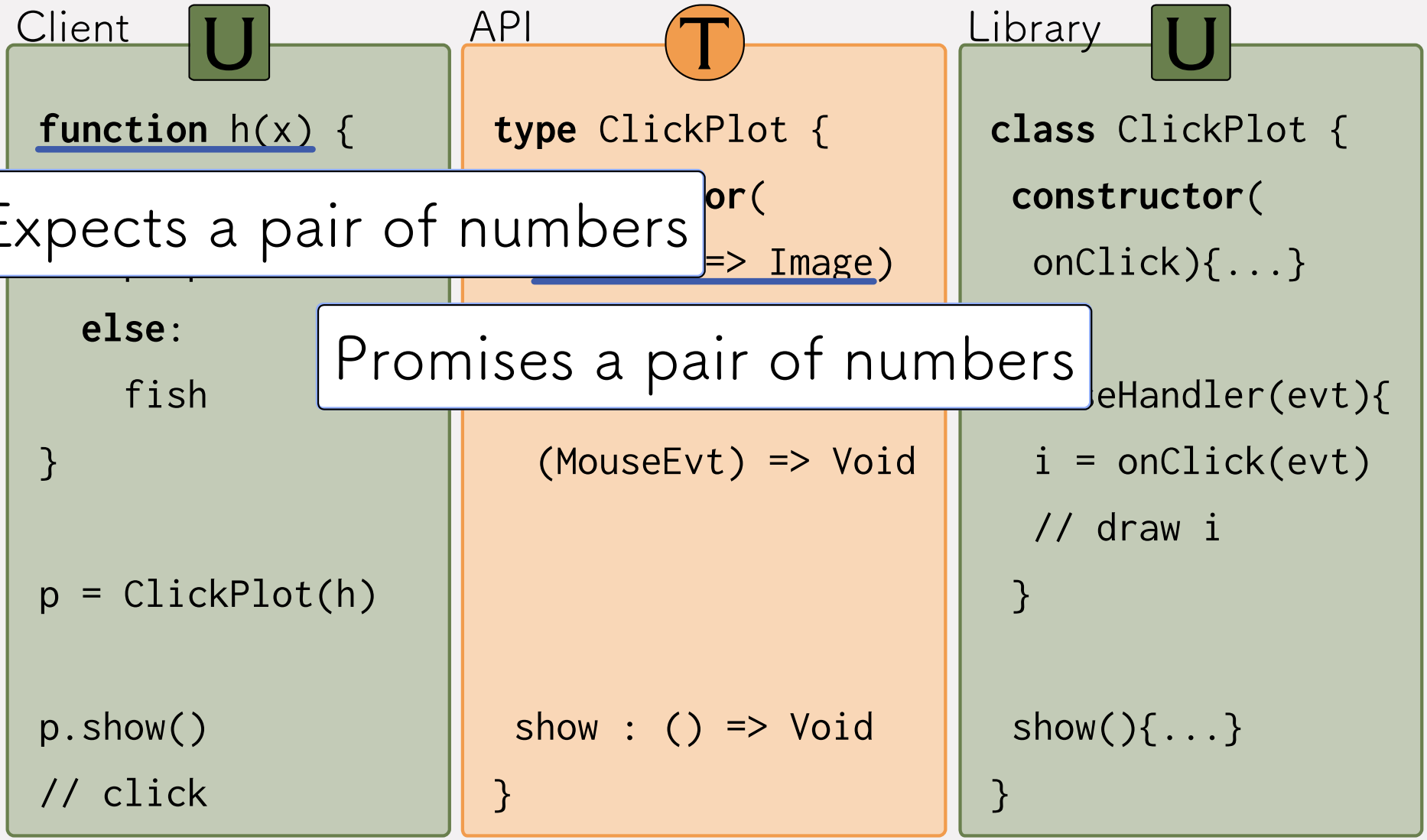
Example: interactive plot



Example: interactive plot



Example: interactive plot



Example: interactive plot

Client U

```
function h(x) {
  if (0 < fst x):
    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){
  i = onClick(evt)
  // draw i
 }

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

Example: interactive plot

Client U

```
function h(x) {
  if (0 < fst x):
    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){
  i = onClick(evt)
  // draw i
 }

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

Example: interactive plot

Client U

```
function h(x) {
  if (0 < fst x):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

API T

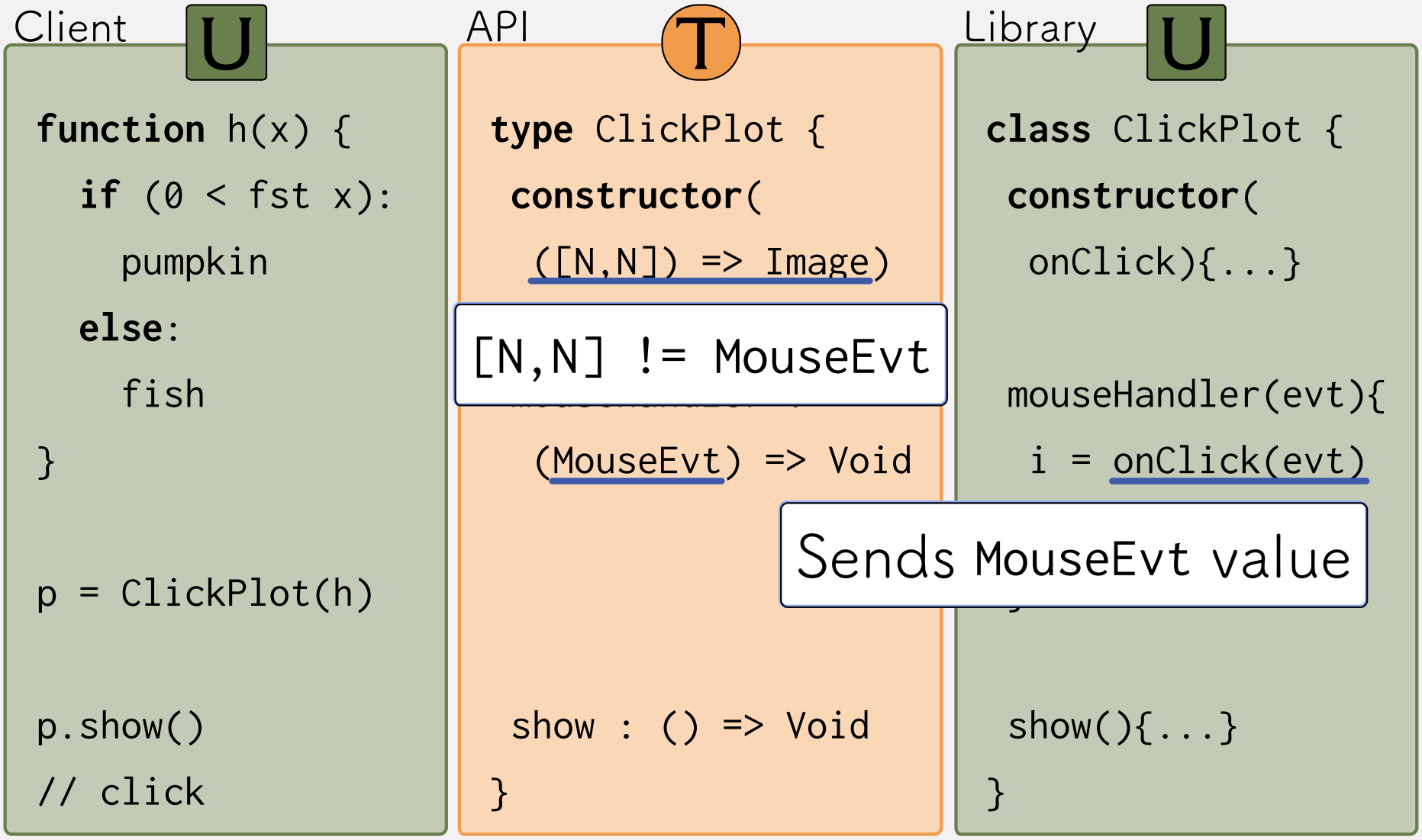
```
type ClickPlot {
 constructor(
  ([N,N]) => Image)

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

Library U

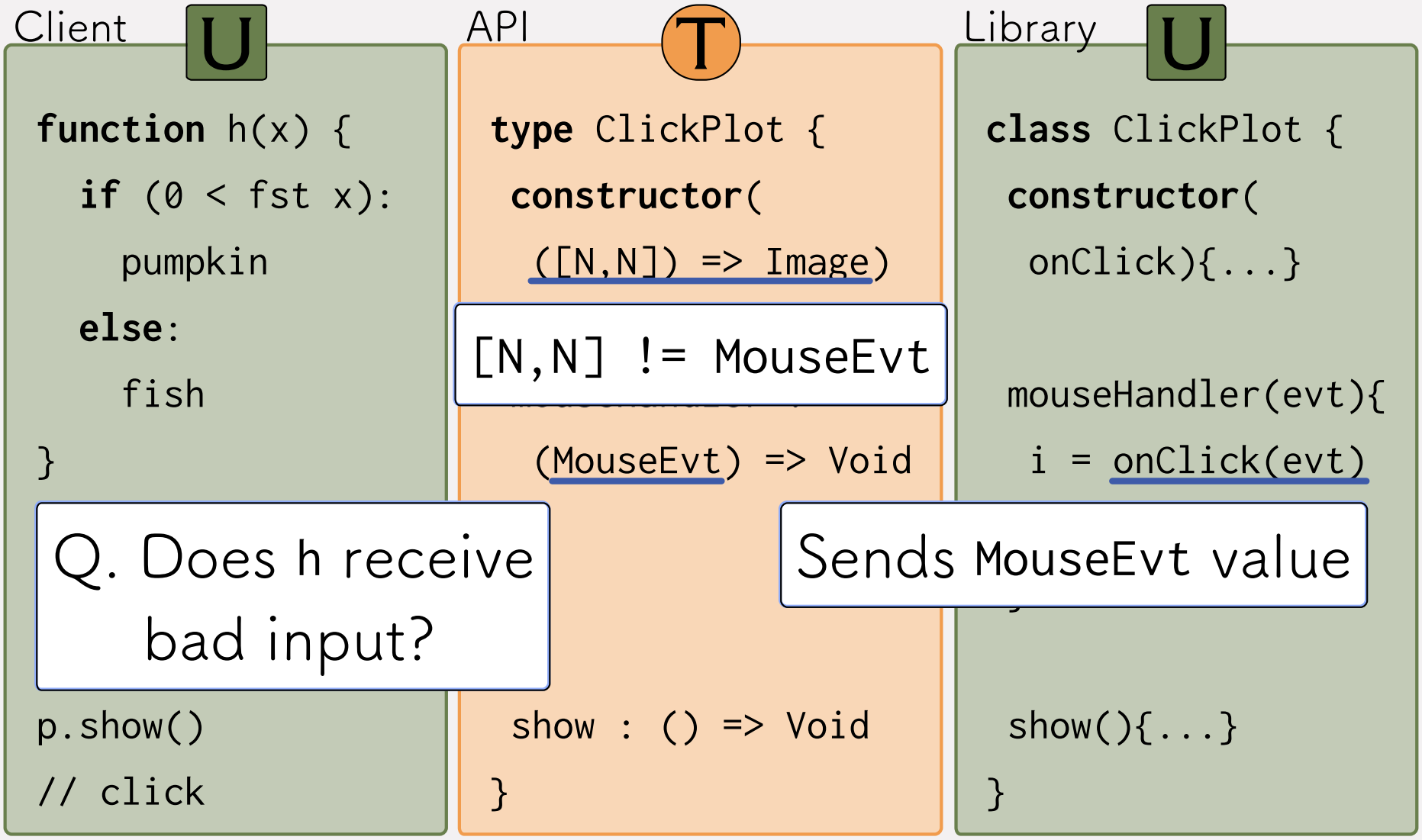
```
class ClickPlot {
 constructor(
  onClick){...}

 mouseHandler(evt){
  i = onClick(evt)

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

Sends MouseEvt value

Example: interactive plot



Example: interactive plot



Example: interactive plot

Client 

```
function h(x) {
  if (0 < fst x):
    pumpkin
  else:
    fish
}
```

Q. Does h receive bad input?

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

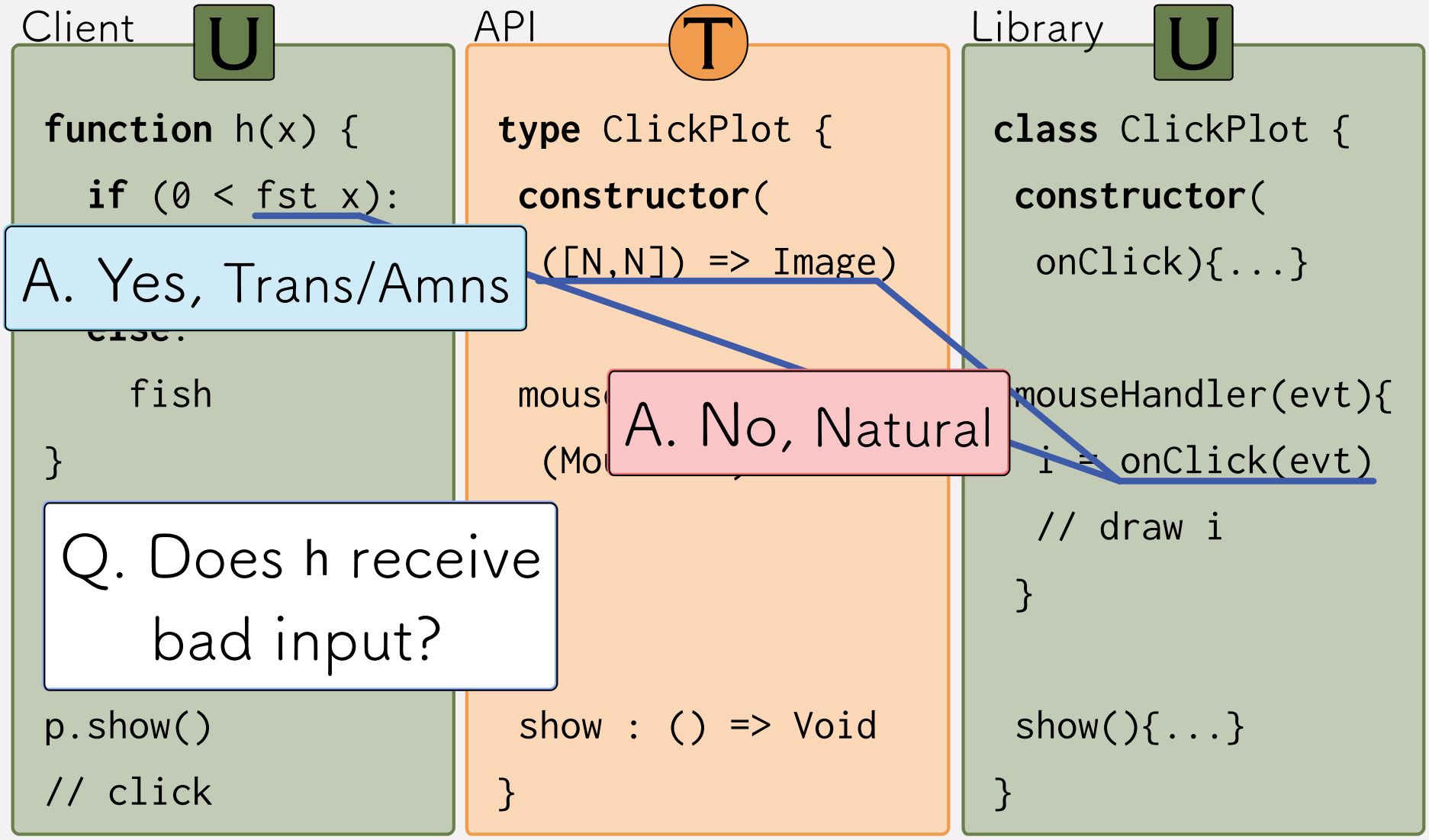
API 

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

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

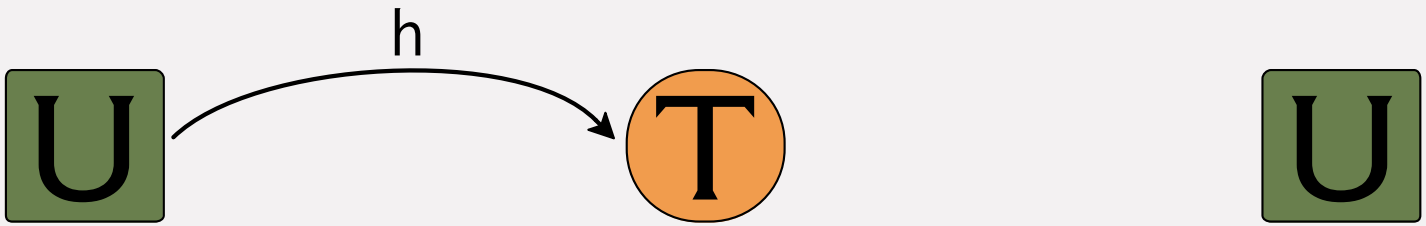
Library 

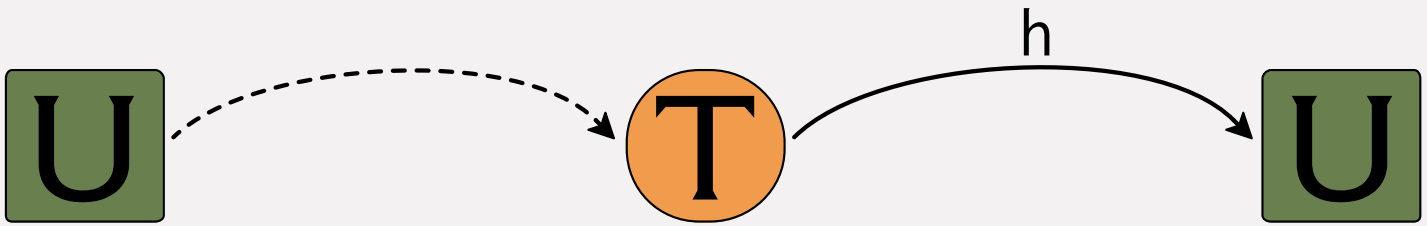
```
class ClickPlot {
 constructor(
  onClick){...}

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

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

Example: interactive plot

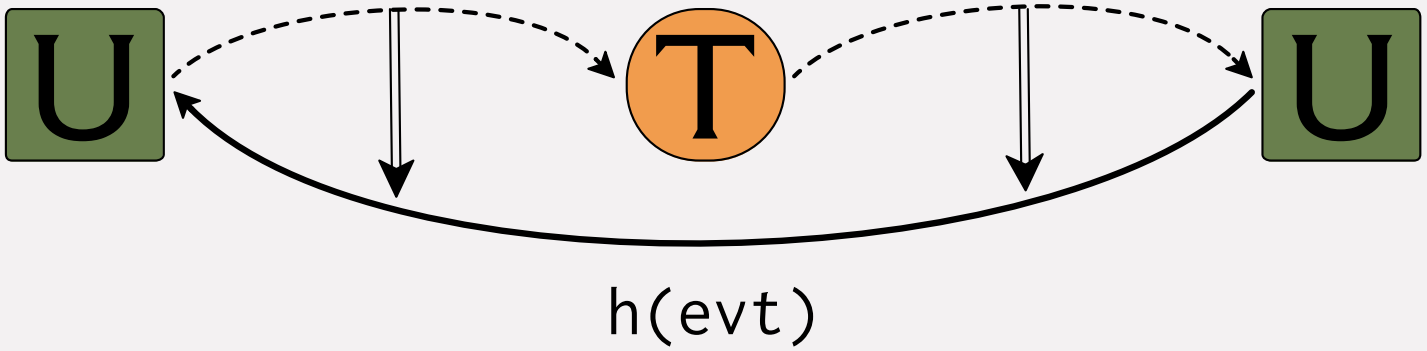



U
T
U

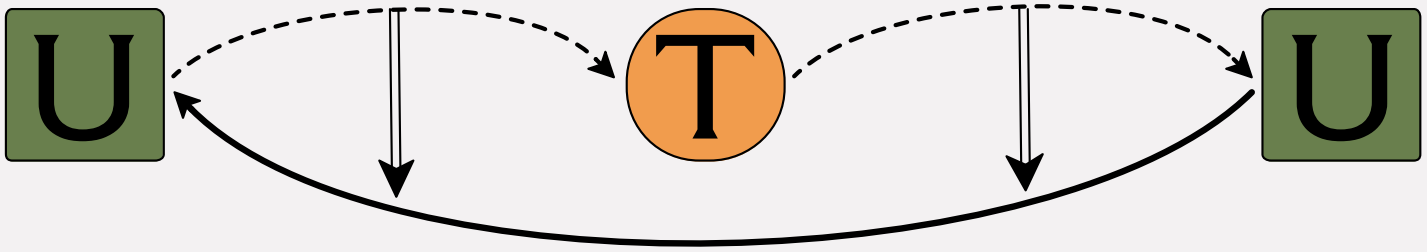

h
U
T
U


U
T
h
U

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



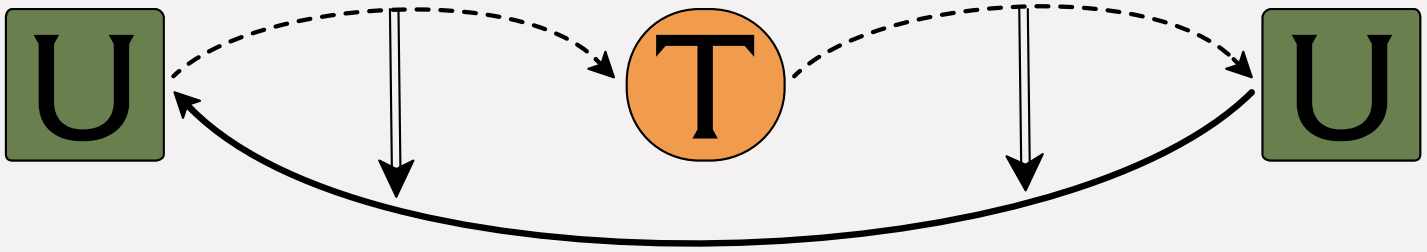
Q. Do types guard the **callback** channel?



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

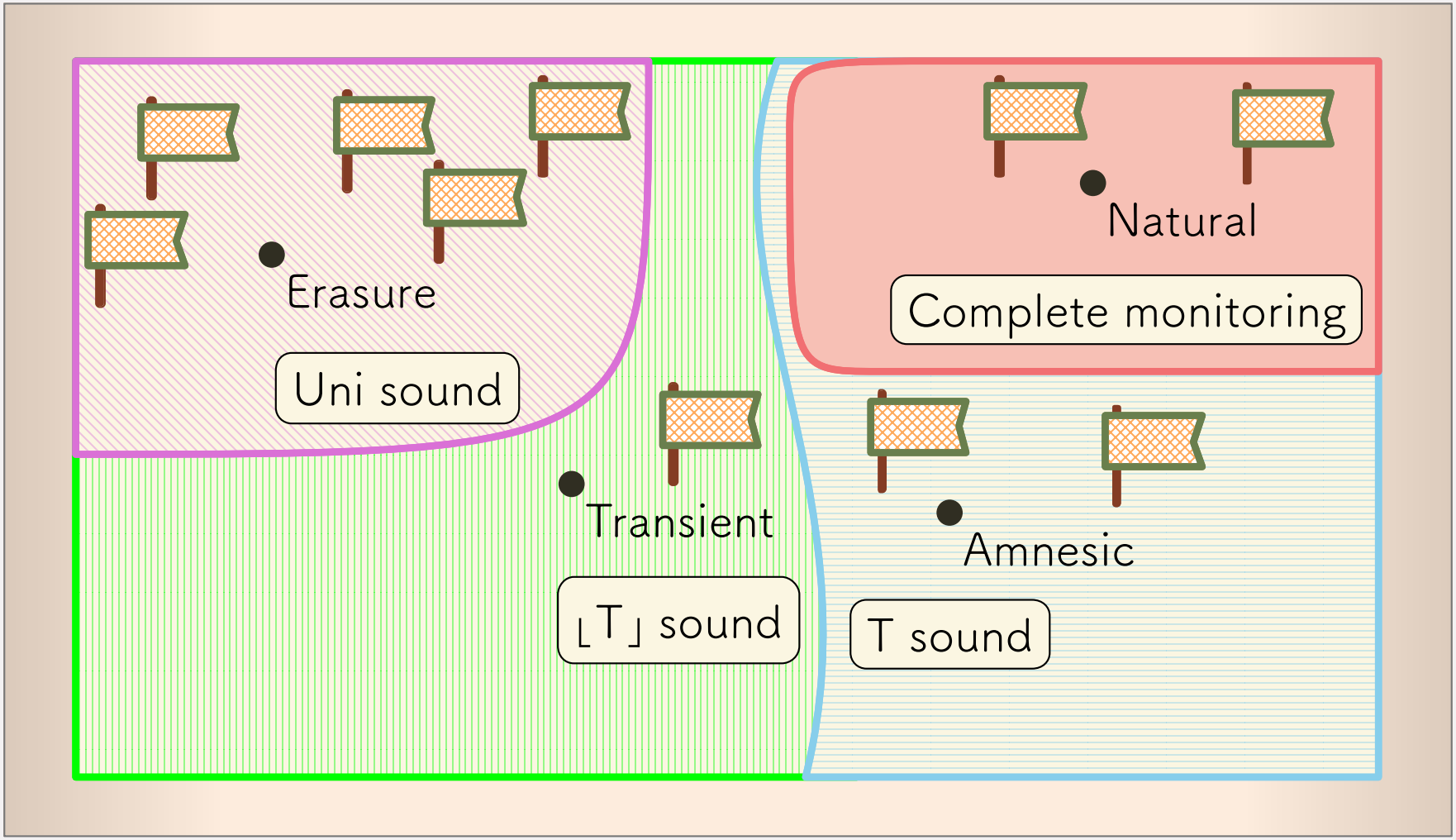
Natural: yes, because the channel was created via typed code

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



Type Soundness $\not\Rightarrow$ yes

Complete Monitoring $\Rightarrow$ yes


Erasure
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(x) {
  if (0 < fst x):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

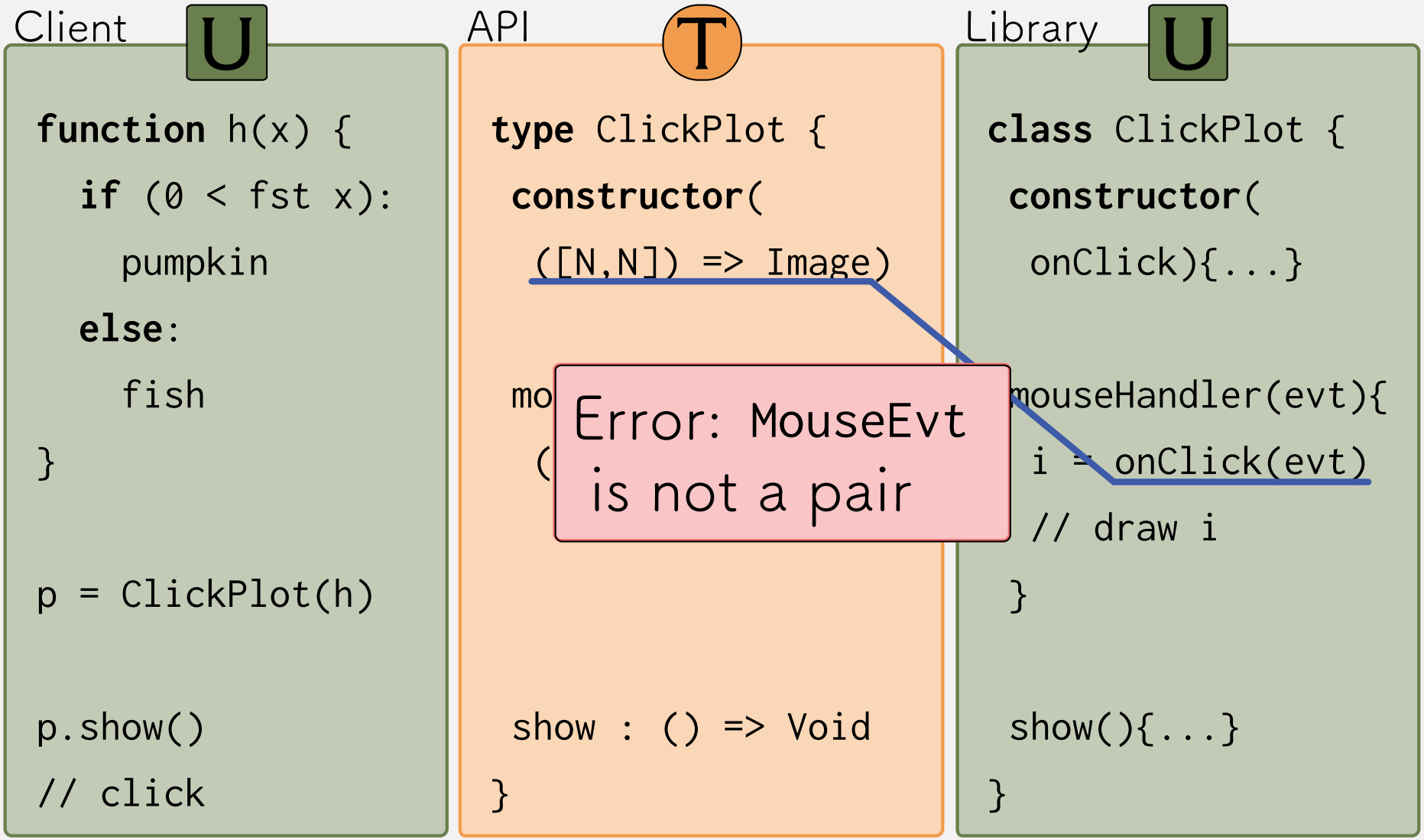
API T

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

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

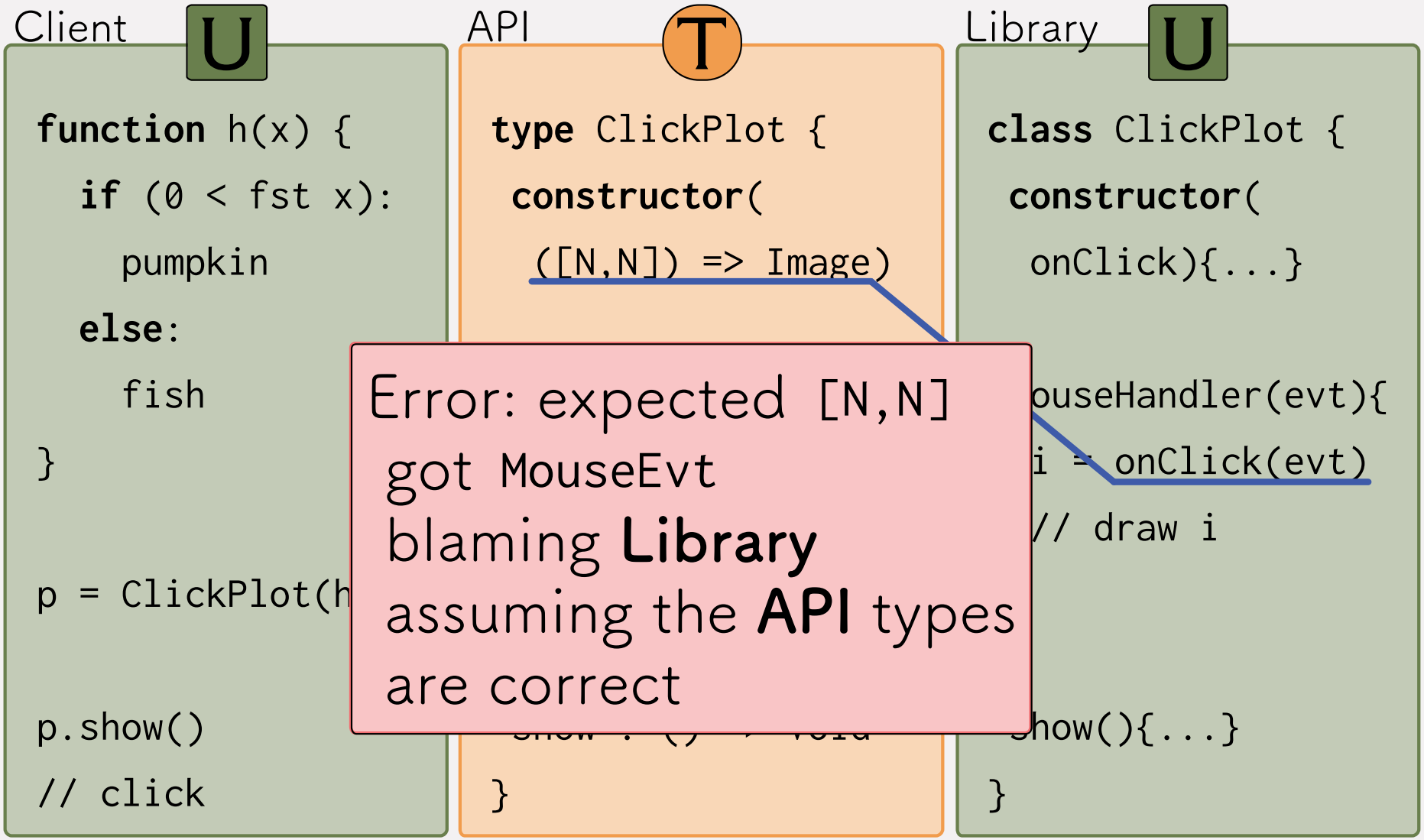
Library U

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

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

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

Natural, Blame



Natural, Blame



Transient/Amnesic, Blame

Client U

```
function h(x) {
  if (0 < fst x):
    pumpkin
  else:
    fish
}

p = ClickPlot(h)

p.show()
// click
```

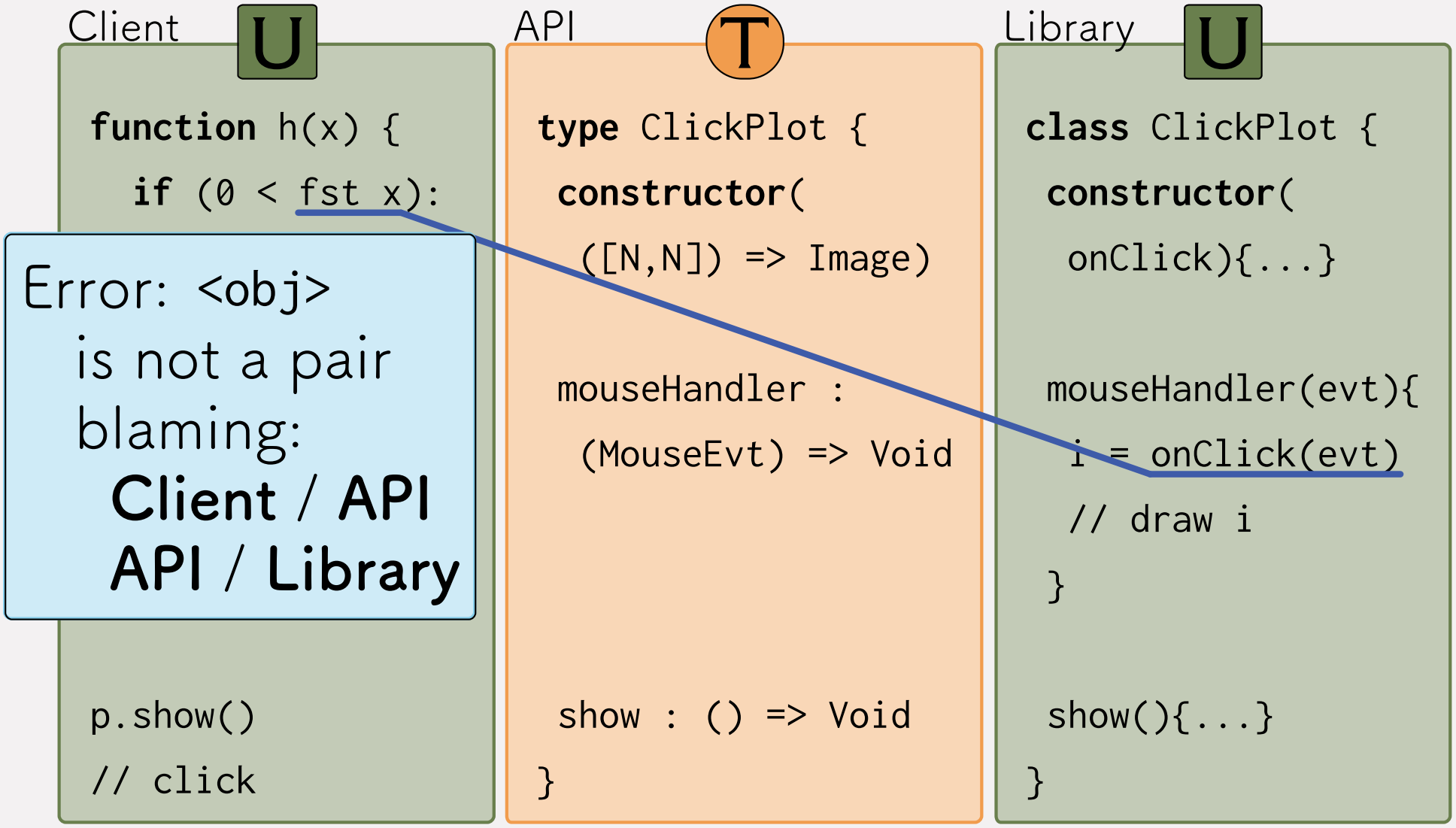
API T

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

 mouseHandler :
  (MouseEvt) => Void

 show : () => Void
}
```

Library U

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

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

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

Transient/Amnesic, Blame

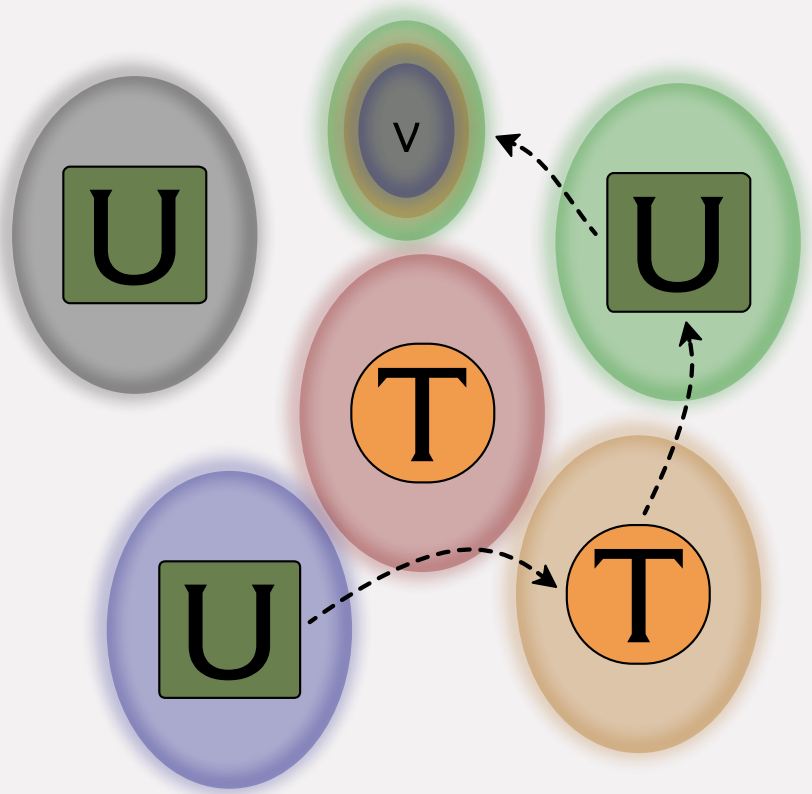


Blame Properties

1. blame **only** responsible edges

2. blame **all** responsible edges

3. blame **exactly** the responsible edges



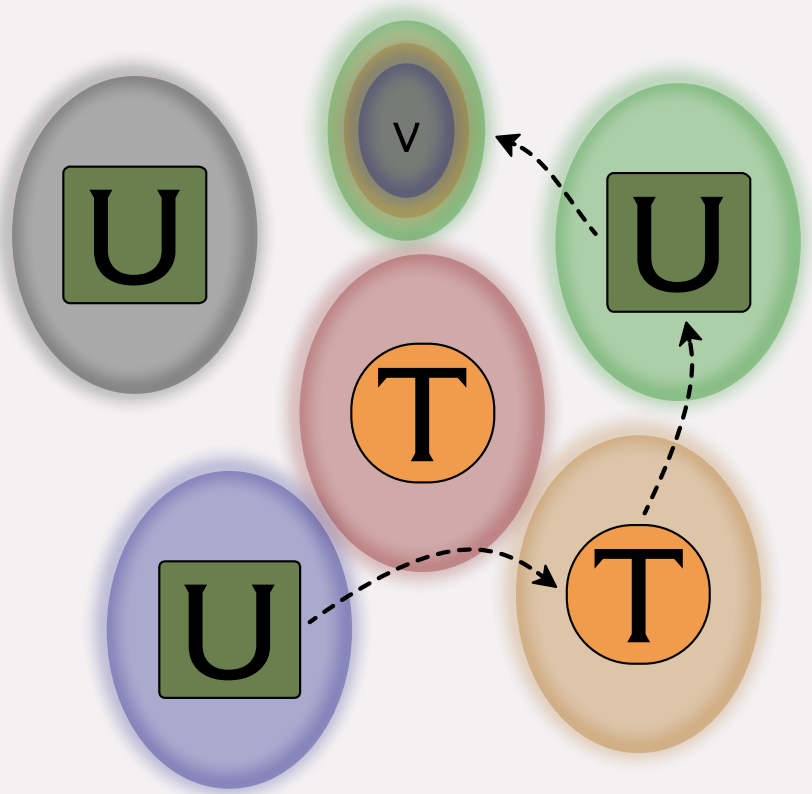
Blame Properties

Blame Soundness

1. blame **only** responsible edges

Blame Completeness

2. blame **all** responsible edges



B. Soundness + B. Completeness

3. blame **exactly** the responsible edges

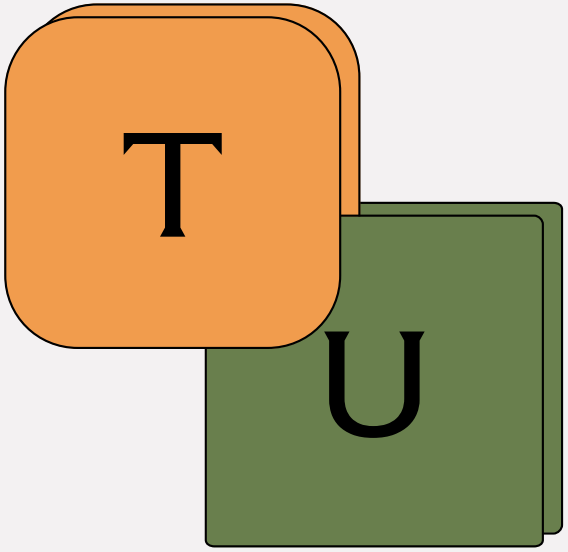
	Natural	Transient	Amnesic
type soundness	T	⌊T⌋	T
complete monitoring	✓	✗	✗
blame soundness	✓		
blame completeness	✓		

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

Every Typed Language is Mixed-Typed

Many typed languages
trust untyped code

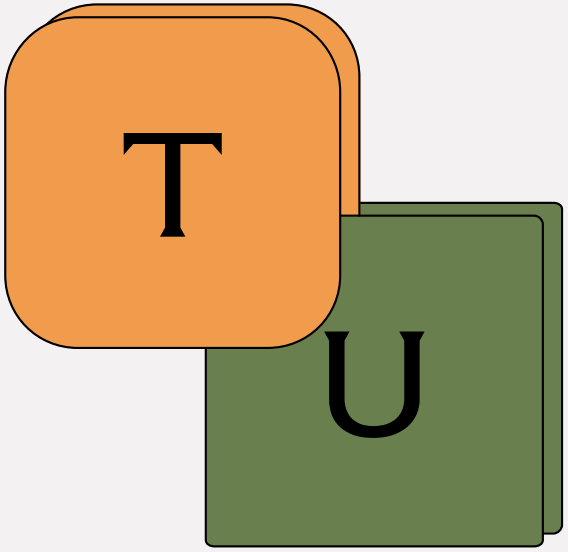
Every Typed Language is Mixed-Typed



Many typed languages **trust** untyped code

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

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

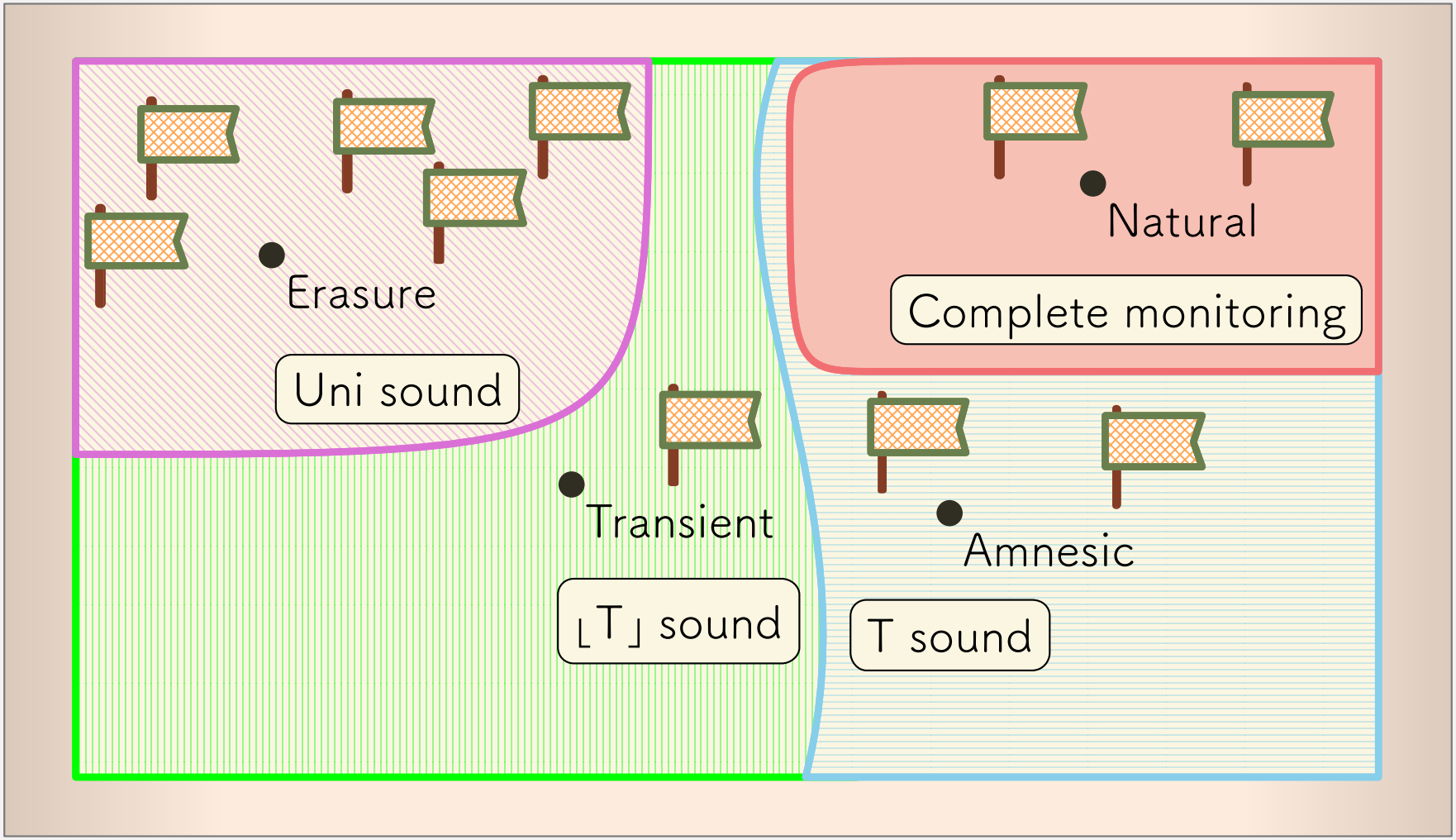
Complete monitoring **strengthens** type soundness
for programs that **compose** typed and untyped

and **enables** precise statements about
the quality of blame

Code + Proofs:

github.com/nuprl/gfd-oopsla-2019

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


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