IMPROVING ABSTRACT GRADUAL TYPING SEMANTICS

Ronald Garcia, Felipe Bañados Schwerter, Khurram Jafery, Alison Clark
University of British Columbia

EXECUTIVE SUMMARY

- Abstracting Gradual Typing (AGT): systematically construct gradually-typed languages from statically-typed ones
    - Semantics satisfy desirable properties (safety, soundness, gradual criteria)
    - But! saturate programs with runtime checks, no matter how static
        - Hand-crafted gradual languages sprinkle checks sparsely
            - Fully static programs have no runtime checks
    - **Question: Can we systematically derive sparse checking regimes?**

GRADUAL TYPING

Less Typed

More Typed

Sound Reasoning Principles

GRADUAL TYPING

Less Typed ⟷ More Typed

Less Typed		More Typed
Unityped	[Siek and Taha 06]	Simple
Unityped	[Siek and Taha 08]	Subtyping
Unityped	[Siek and Vachharajani 08]	Hindley/Milner
Simple	[Lehman et al. 17] [Dunfield et al. 17]	Refinements
Simple	[Bañados et al. 14]	Type&Effect
Simple	[Disney and Flanagan 11]	Security

GRADUAL TYPING

"Dynamic": Most Studied

Less Typed — More Typed

Unityped	[Siek and Taha 06]	Simple
Unityped	[Siek and Taha 08]	Subtyping
Unityped	[Siek and Vachharajani 08]	Hindley/Milner
Simple	[Lehman et al. 17] [Dunfield et al. 17]	Refinements
Simple	[Bañados et al. 14]	Type&Effect
Simple	[Disney and Flanagan 11]	Security

GRADUAL TYPING

Less Typed		More Typed
Unityped	[Siek and Taha 06]	Simple
Unityped	… and Taha 08]	Subtyping
Unityped	[Siek and Vachharajani 08]	Hindley/Milner
Simple	[Lehman et al. 17] [Dunfield et al. 17]	Refinements
Simple	[Bañados et al. 14]	Type&Effect
Simple	[Toro et al. 18]	Security

“Dynamic” is Relative

MIXED PROGRAMMING

```
def f(x:int) = x + 2
def h(g) = g(true)
h(f)
```

FROM MIXED TO GRADUAL

```
def f(x:int) = x + 2
def h(g) = g(true)
h(f)
```

desugar

```
def f(x:int) = x + 2
def h(g:?) =g(true:?):?
h(f:?)
```

Imprecise Types

[Siek & Taha, 2006]

[Wadler & Findler, 2009]

? = "The Unknown Type"

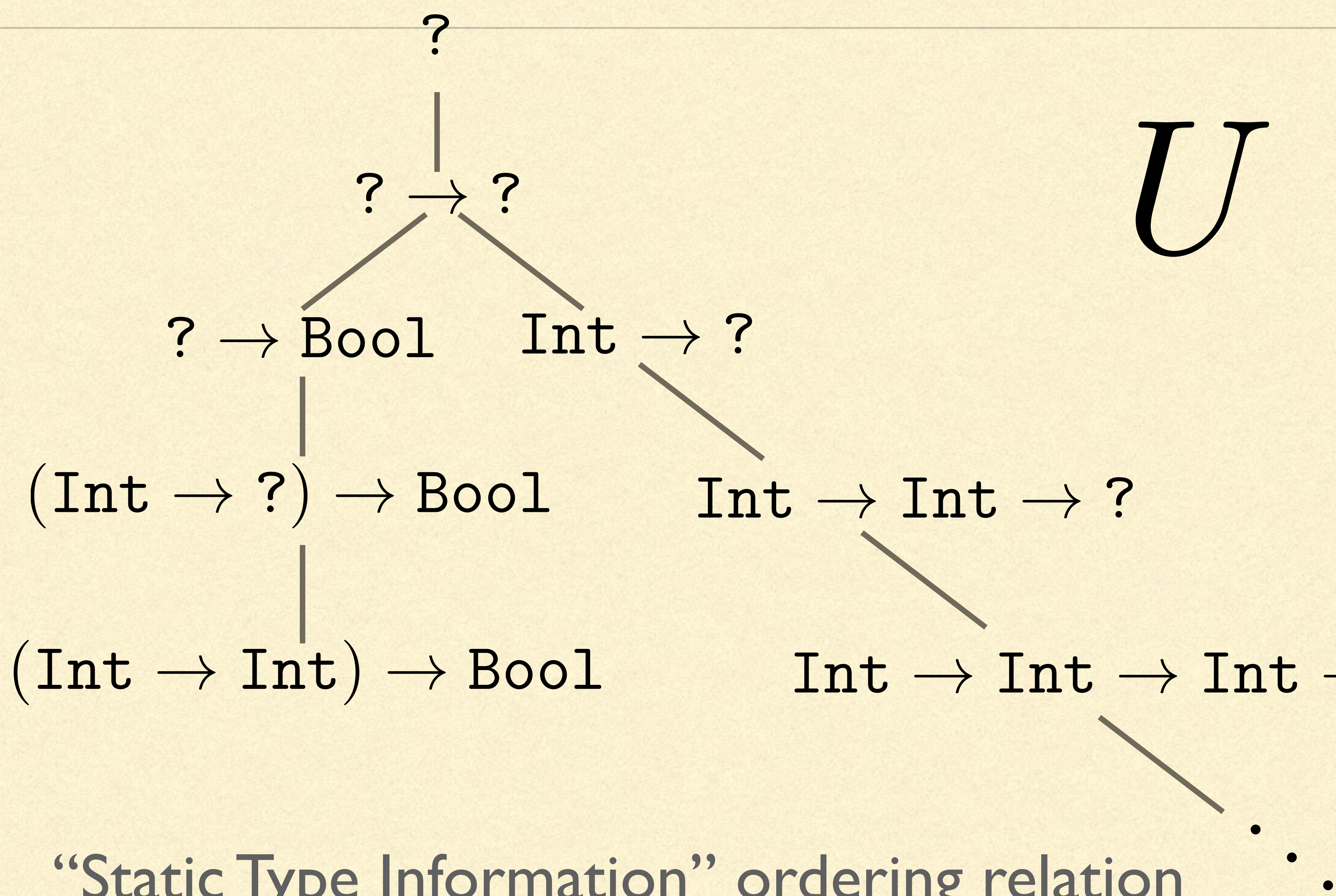
STATIC AND GRADUAL TYPES

Static Types (Type) $T ::= B \mid T \to T$

Gradual Types (GType) $U ::= \texttt{?} \mid B \mid U \to U$

$$\textsc{Type} \subseteq \textsc{GType}$$

GRADUAL TYPE PRECISION



$$U \sqsubseteq U$$

“Static Type Information” ordering relation

CONSISTENT LIFTING(*)

$$U_1 \sim U_2$$ Gradual Type Consistency

if and only if $\sqcup\!| \qquad \sqcup\!|$

$$T_1 = T_2$$ Static Type Equality

For some T1 and T2

(*) Reformulation of original definition

STATIC CHECKING

static type equality

$$\mathsf{Int} = \mathsf{Int}$$
$$\mathsf{Bool} = \mathsf{Bool}$$
$$\mathsf{Int} \to \mathsf{Bool} \neq \mathsf{Bool} \to \mathsf{Int}$$

extend

gradual type consistency

$$\mathsf{Int} \sim \mathsf{Int}$$
$$\mathsf{Bool} \sim \mathsf{Bool}$$
$$\mathsf{Int} \to \mathsf{Bool} \not\sim \mathsf{Bool} \to \mathsf{Int}$$
$$? \sim \mathsf{Bool}$$
$$? \to \mathsf{Bool} \sim \mathsf{Bool} \to ?$$

Consistency conservatively extends equality

CONSISTENT LIFTING

$$U_1 \lesssim U_2$$ Consistent Subtyping

if and only if $\quad \sqsubseteq \qquad \sqsubseteq$

$$T_1 <: T_2$$ Static Subtyping

For some T1 and T2

CONSISTENT LIFTING

Conservatively Extends <:

"unknown" is not the "top" type

$$\texttt{Int} \lesssim \texttt{Int}$$

$$\texttt{Int} \not\lesssim \texttt{Bool}$$

$$\texttt{Int} \lesssim \top$$

$$\top \not\lesssim \texttt{Int}$$

$$\texttt{Int} \lesssim \texttt{?}$$

$$\texttt{?} \lesssim \texttt{Int}$$

LIFT TYPING RULES

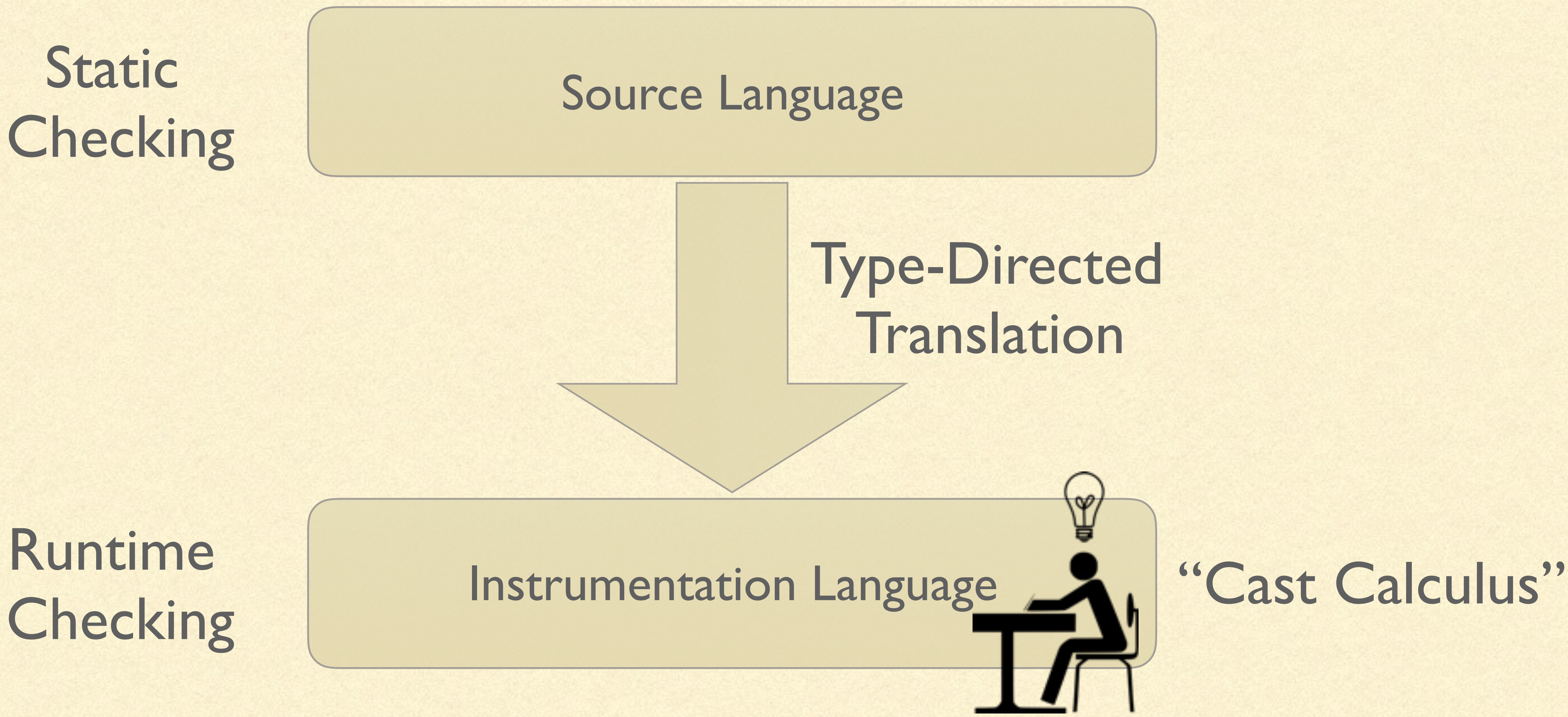
Static Type System

$$\frac{\Gamma \vdash t_1 : T_1 \qquad T_1 = \texttt{Int} \qquad \Gamma \vdash t_2 : T_2 \qquad T_2 = \texttt{Int}}{\Gamma \vdash t_1 + t_2 : \texttt{Int}}$$

Gradual Type System

$$\frac{\Gamma \vdash t_1 : U_1 \qquad U_1 \sim \texttt{Int} \qquad \Gamma \vdash t_2 : U_2 \qquad U_2 \sim \texttt{Int}}{\Gamma \vdash t_1 + t_2 : \texttt{Int}}$$

CLASSIC DYNAMIC SEMANTICS



THE JOY OF NOT THINKING

"It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle---they are strictly limited in number, they require fresh horses, and must only be made at decisive moments."

Alfred North Whitehead. "An Introducti athematics"

static type system &
type safety proof

interpretation of
gradual types

Abstracting Gradual Typing

Ronald Garcia [*] Alison M. Clark [†]
Software Practices Lab
Department of Computer Science
University of British Columbia, Canada
{rxg,amclark1}@cs.ubc.ca

Éric Tanter [‡]
PLEIAD Laboratory
Computer Science Department (DCC)
University of Chile, Chile
etanter@dcc.uchile.cl

POPL 2016

gradual language

TYPE SYSTEM | DYNAMIC SEMANTICS

SEMANTICS OF GRADUAL TYPES

e.g.,

$$\gamma : \textsc{GType} \rightarrow \mathcal{P}^{+}(\textsc{Type})$$

$$\gamma(\mathsf{Int}) = \{\, \mathsf{Int} \,\}$$

$$\gamma(\mathsf{Int} \rightarrow ?) = \{\, \mathsf{Int} \rightarrow T \mid T \in \textsc{Type} \,\}$$

$$\gamma(?) = \textsc{Type}$$

Concretization Function

$$U_1 \sqsubseteq U_2 \;\equiv\; \gamma(U_1) \subseteq \gamma(U_2)$$

SEMANTICS OF GRADUAL TYPES

e.g.,

$$\alpha : \mathcal{P}^{+}(\textsc{Type}) \rightarrow \textsc{GType}$$

$$\alpha(\gamma(U)) = U$$

$$\alpha(\{\,\texttt{Int}, \texttt{Bool}\,\}) = \texttt{?}$$

$$\alpha(\{\,\texttt{Int} \rightarrow \texttt{Int}, \texttt{Bool} \rightarrow \texttt{Int}\,\}) = \texttt{?} \rightarrow \texttt{Int}$$

Abstraction Function
(induced by Concretization)

$$U_1 \,\widetilde{\ddot{\vee}}\, U_2 = \alpha(\gamma(U_1)\, \ddot{\vee}^{*}\, \gamma(U_2))$$

CLASSIC DYNAMIC SEMANTICS

Static Checking

Source Language

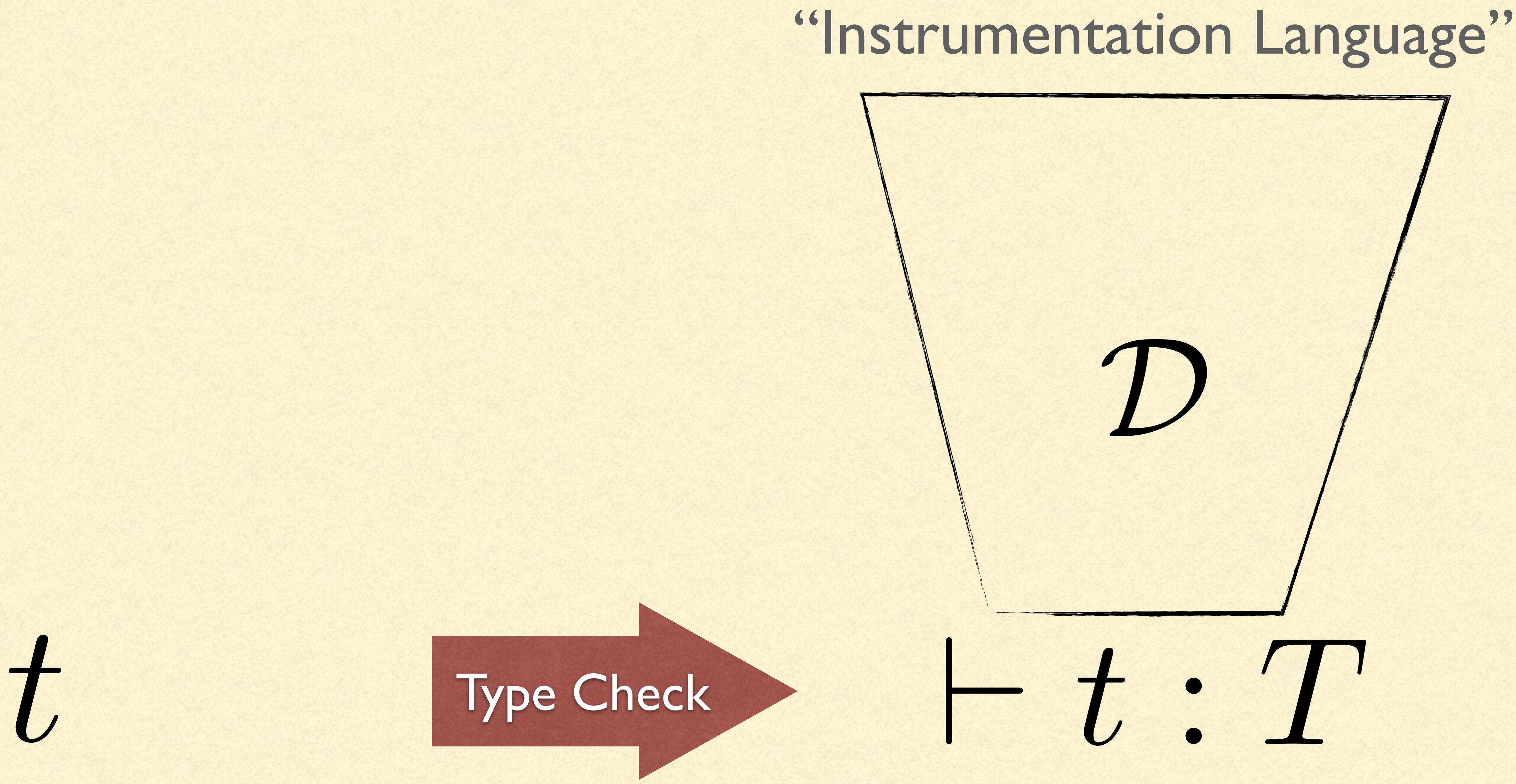
Type-Directed Translation

Runtime Checking

Instrumentation Language

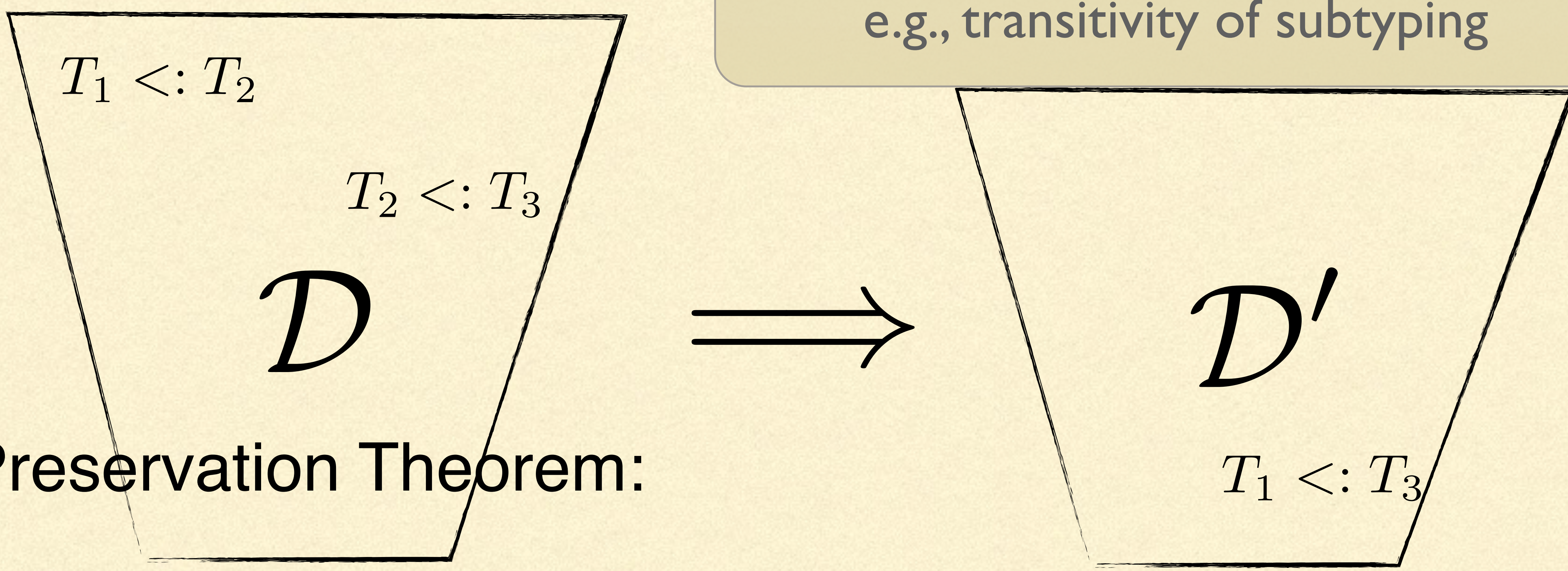
"Cast Calculus"

AGT DYNAMIC SEMANTICS



(SUBJECT) REDUCTION

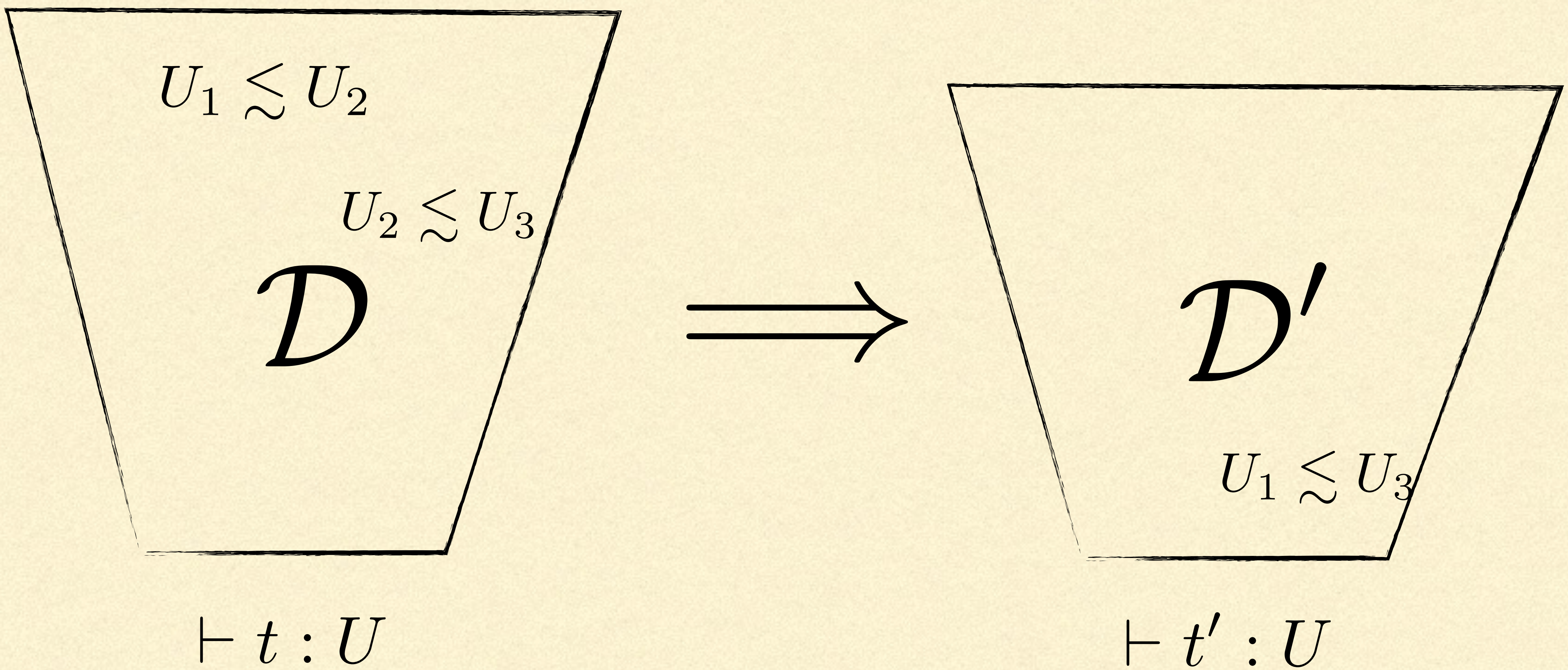
reasoning about types,
e.g., transitivity of subtyping



Preservation Theorem:

$\vdash t : T$ $\quad t \longmapsto t'$ $\quad \vdash t' : T$

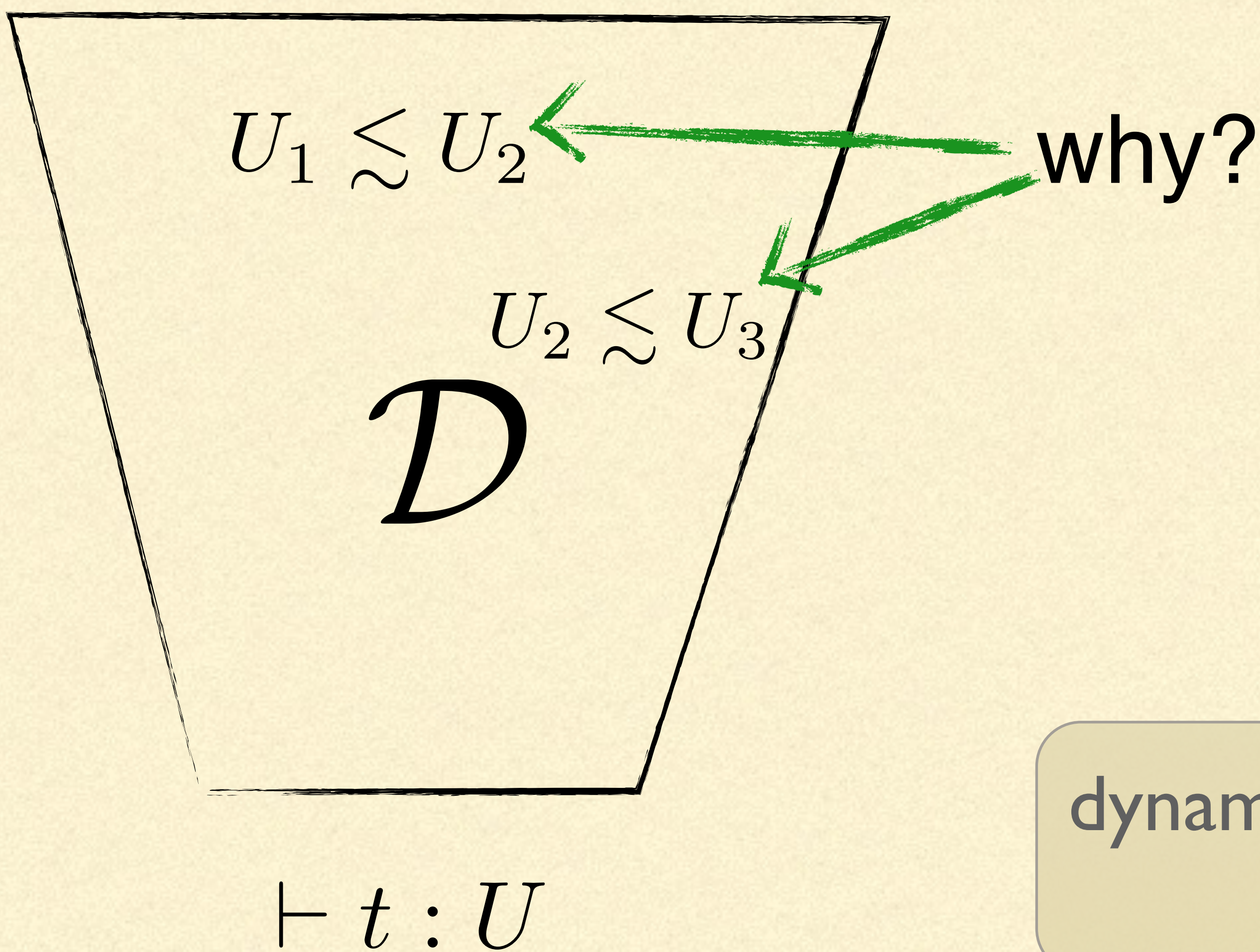
REDUCE TYPE DERIVATIONS!



consistent subtyping is *not* transitive!

$\mathsf{Int} \lesssim ?$ and $? \lesssim \mathsf{Bool}$ but $\mathsf{Int} \not\lesssim \mathsf{Bool}$

EVIDENCE OF CONSISTENT JUDGMENTS



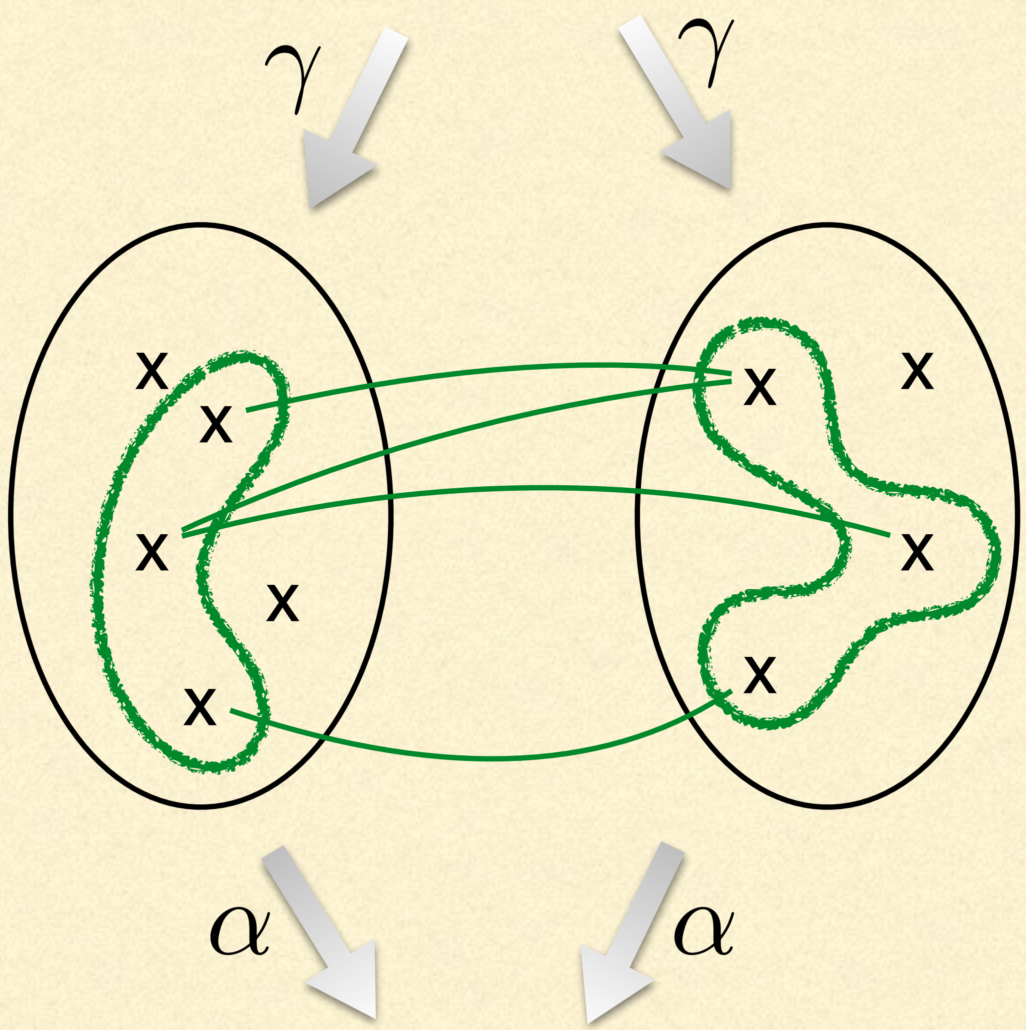
introduce **evidence**

$$\varepsilon_{12} \vdash U_1 \lesssim U_2$$

dynamic “local justification” for consistent subtyping

INITIAL EVIDENCE

$$U_1 \lesssim U_2$$



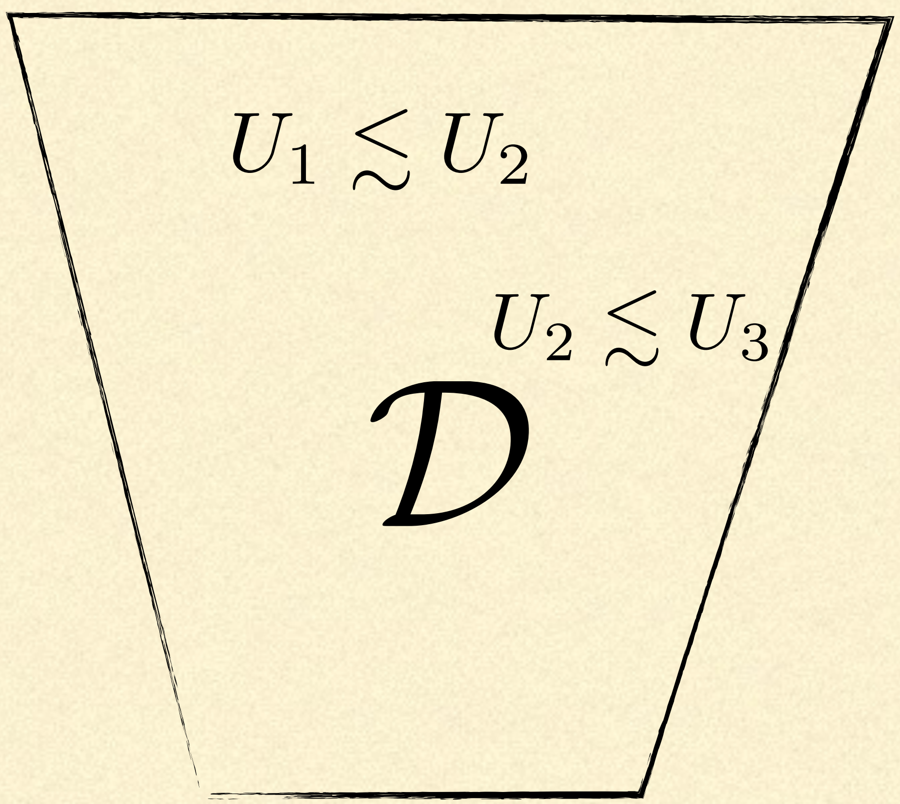
$$\langle U_1', U_2' \rangle$$

$$\varepsilon_{12} \vdash U_1 \lesssim U_2$$

Corresponds to Threesome middle type

[Siek & Wadler, 2010]

EVIDENCE

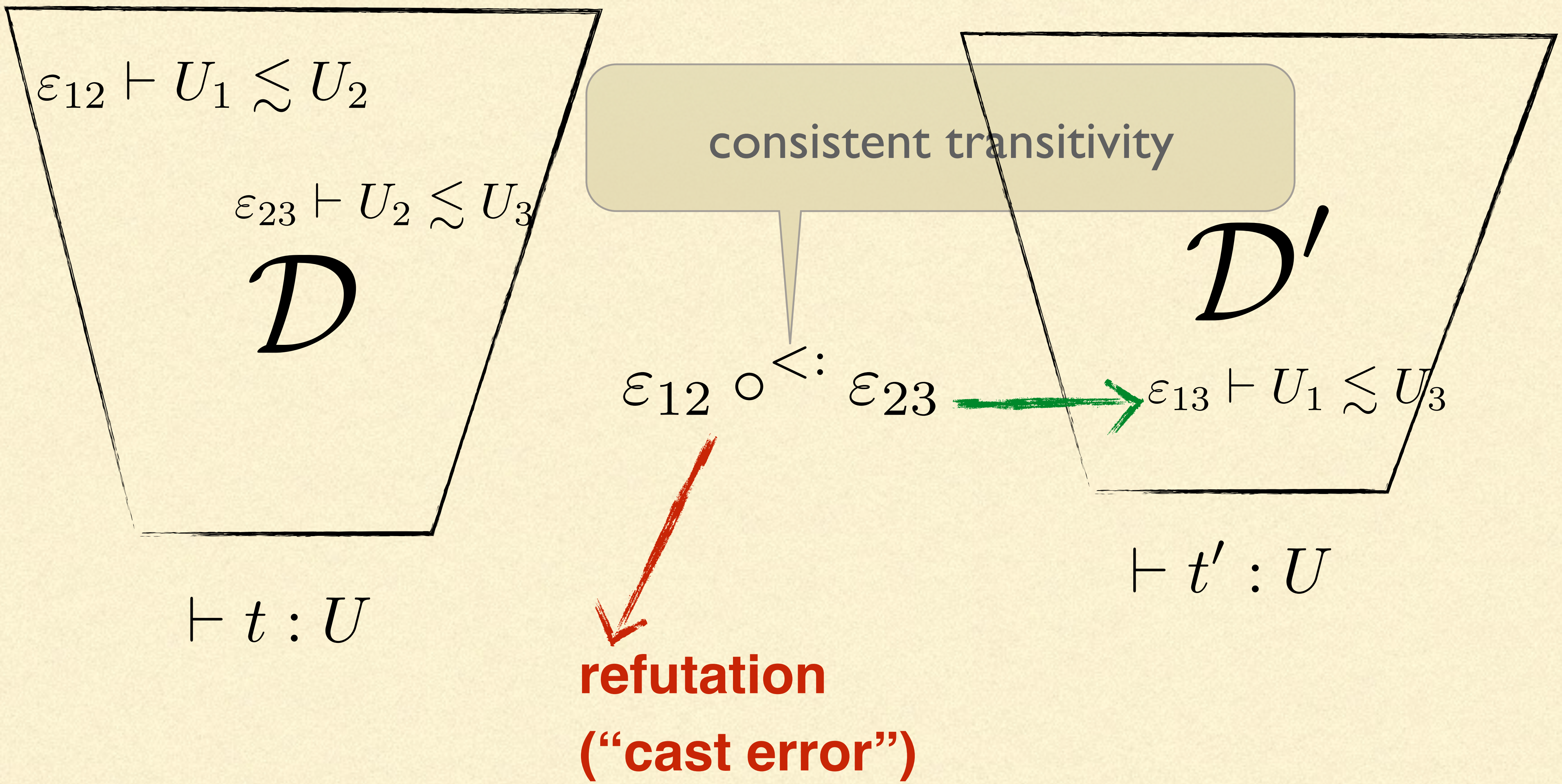


$\vdash t : U$

$$\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$$

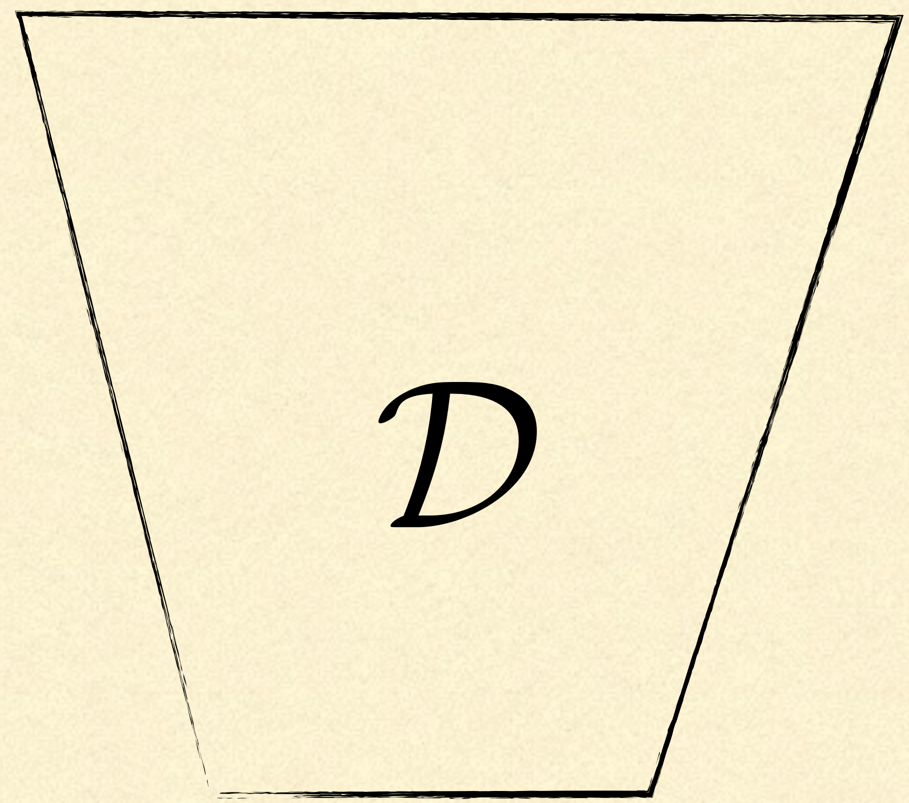
$$\varepsilon_{23} \vdash U_2 \lesssim U_3$$

CONSISTENT TRANSITIVITY



ABUNDANT EVIDENCE!

“Instrumentation Language”



t — Type Check → $\vdash t : T$

ABUNDANT EVIDENCE!

$$(\widetilde{T}\text{x})\ \frac{x : \widetilde{T} \in \Gamma}{\Gamma \vdash x : \widetilde{T}} \qquad (\widetilde{T}\text{n})\ \frac{}{\Gamma \vdash n : \mathsf{Int}} \qquad (\widetilde{T}\text{b})\ \frac{}{\Gamma \vdash b : \mathsf{Bool}}$$

$$(\widetilde{T}\text{app})\ \frac{\Gamma \vdash \widetilde{t}_1 : \widetilde{T}_1 \quad \Gamma \vdash \widetilde{t}_2 : \widetilde{T}_2 \quad \widetilde{T}_2 \sim \widetilde{dom}(\widetilde{T}_1)}{\Gamma \vdash \widetilde{t}_1\ \widetilde{t}_2 : \widetilde{cod}(\widetilde{T}_1)}$$

$$(\widetilde{T}+)\ \frac{\Gamma \vdash \widetilde{t}_1 : \widetilde{T}_1 \quad \Gamma \vdash \widetilde{t}_2 : \widetilde{T}_2 \quad \widetilde{T}_1 \sim \mathsf{Int} \quad \widetilde{T}_2 \sim \mathsf{Int}}{\Gamma \vdash \widetilde{t}_1 + \widetilde{t}_2 : \mathsf{Int}}$$

$$(\widetilde{T}\text{if})\ \frac{\Gamma \vdash \widetilde{t}_1 : \widetilde{T}_1 \quad \Gamma \vdash \widetilde{t}_2 : \widetilde{T}_2 \quad \Gamma \vdash \widetilde{t}_3 : \widetilde{T}_3 \quad \widetilde{T}_1 \sim \mathsf{Bool}}{\Gamma \vdash \mathsf{if}\ \widetilde{t}_1\ \mathsf{then}\ \widetilde{t}_2\ \mathsf{else}\ \widetilde{t}_3 : \widetilde{T}_2 \sqcap \widetilde{T}_3}$$

$$(\widetilde{T}\lambda)\ \frac{\Gamma, x : \widetilde{T}_1 \vdash \widetilde{t} : \widetilde{T}_2}{\Gamma \vdash (\lambda x : \widetilde{T}_1 . \widetilde{t}) : \widetilde{T}_1 \to \widetilde{T}_2} \qquad (\widetilde{T}::)\ \frac{\Gamma \vdash \widetilde{t} : \widetilde{T} \quad \widetilde{T} \sim \widetilde{T}_1}{\Gamma \vdash (\widetilde{t} :: \widetilde{T}_1) : \widetilde{T}_1}$$

SPARSE CASTING

Siek & Taha: Gradual Typing for Objects (2008)

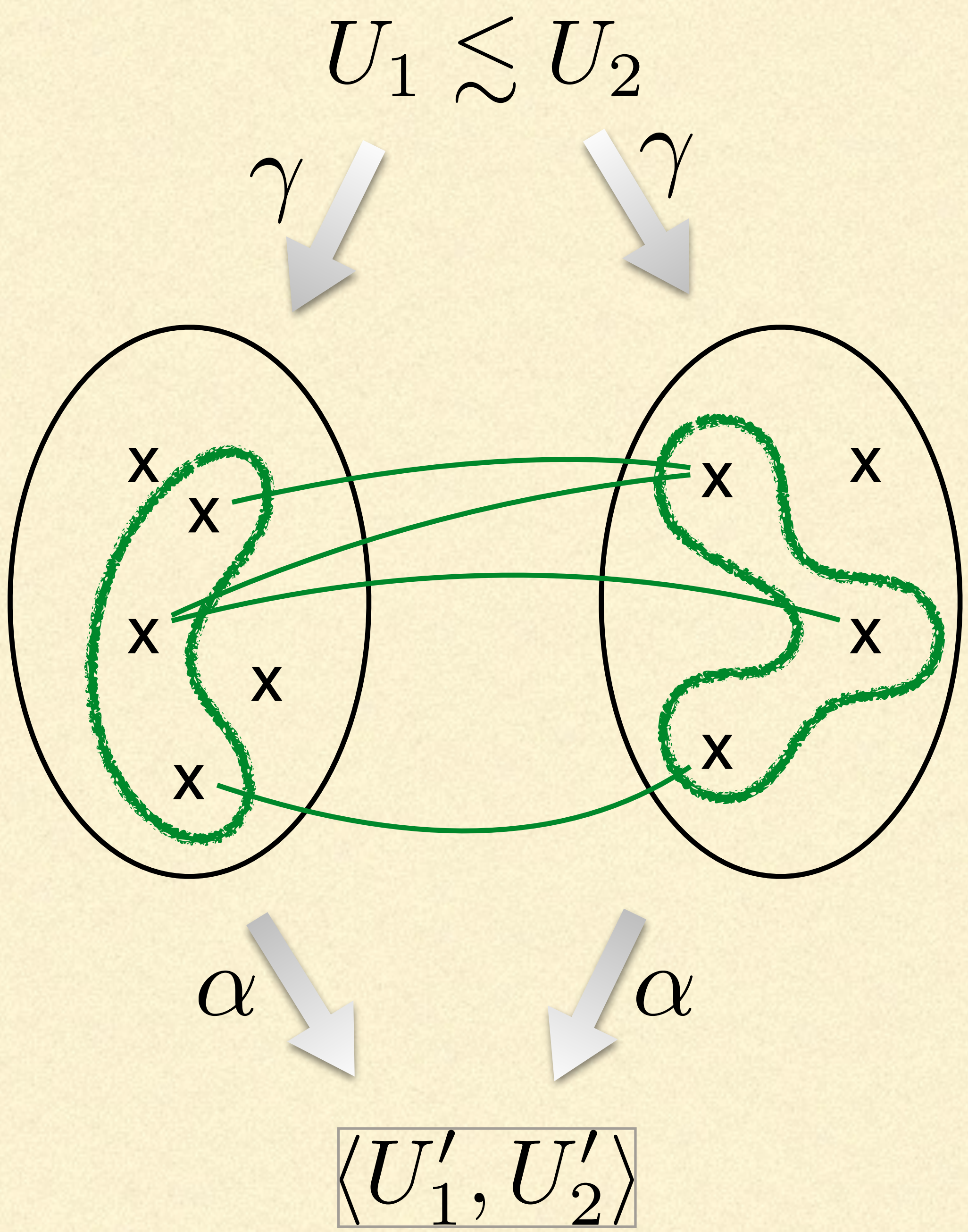
In the case when $\sigma' = \sigma$, we do not insert a cast, which is why we use the following helper function.

$$\langle\!\langle \tau \Leftarrow \sigma \rangle\!\rangle e \equiv \textsf{if } \sigma = \tau \textsf{ then } e \textsf{ else } \langle \tau \Leftarrow \sigma \rangle e$$

Plus: Operational Semantics
Discards "Useless" Casts

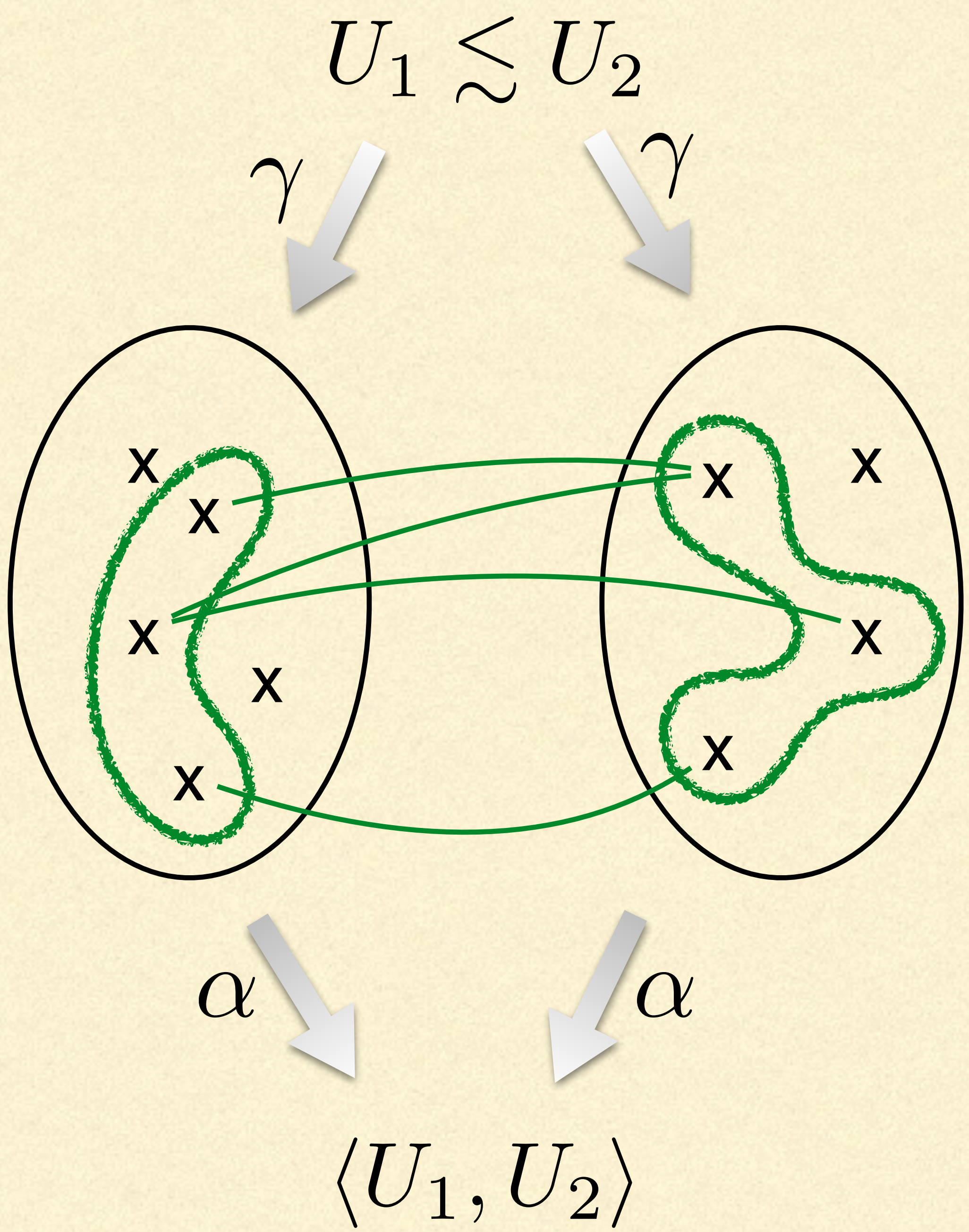
Question: when is evidence "useless"?

PROP 1: IRRELEVANT EVIDENCE



$\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$

PROP 1: IRRELEVANT EVIDENCE



$\boxed{\langle U_1, U_2 \rangle} \vdash U_1 \lesssim U_2$

Mirrored Runtime Evidence
Adds Nothing Locally

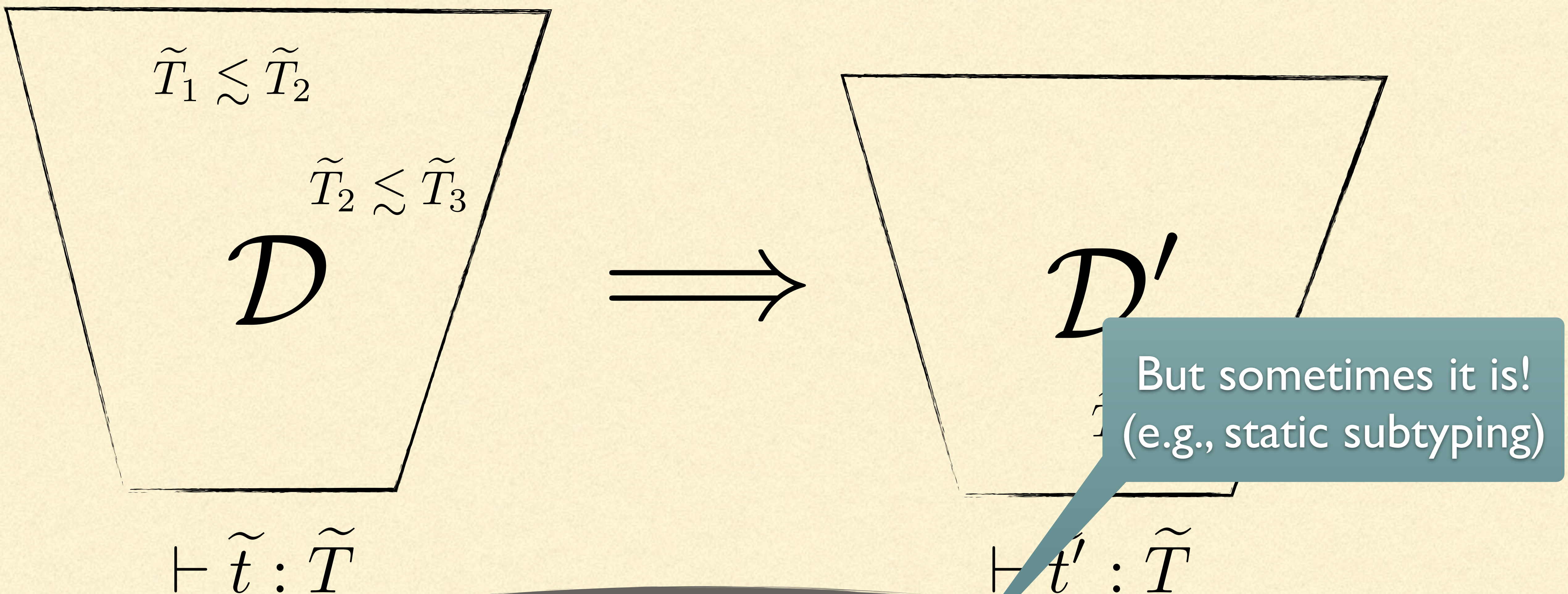
But may matter *globally*

$[x : \texttt{Int}] \lesssim \texttt{?}$

$? \lesssim [x : \texttt{Bool}]$

$[x : \texttt{Int}] \not\lesssim [x : \texttt{Bool}]$

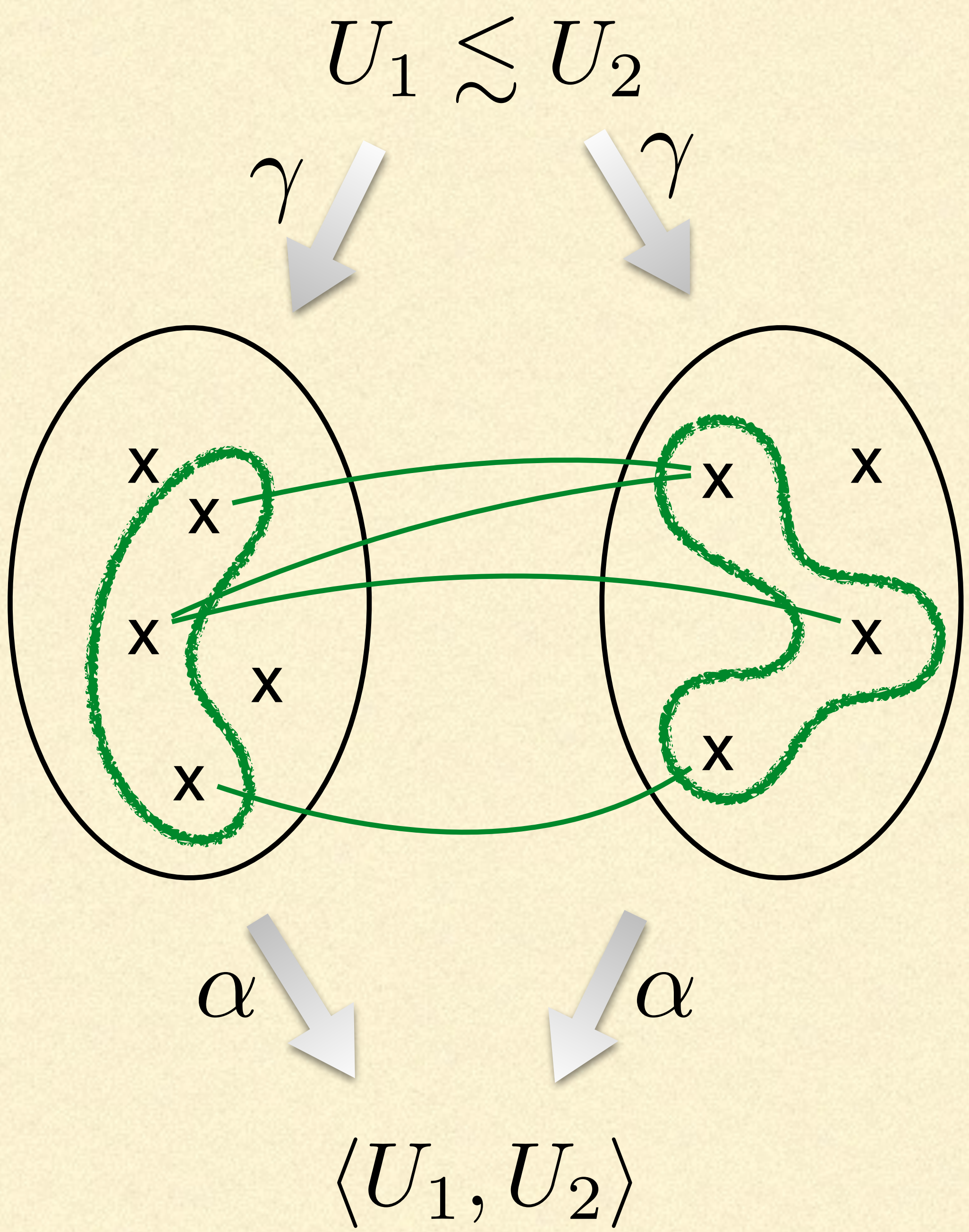
PROP 2: TRANSITIVE "KERNEL"



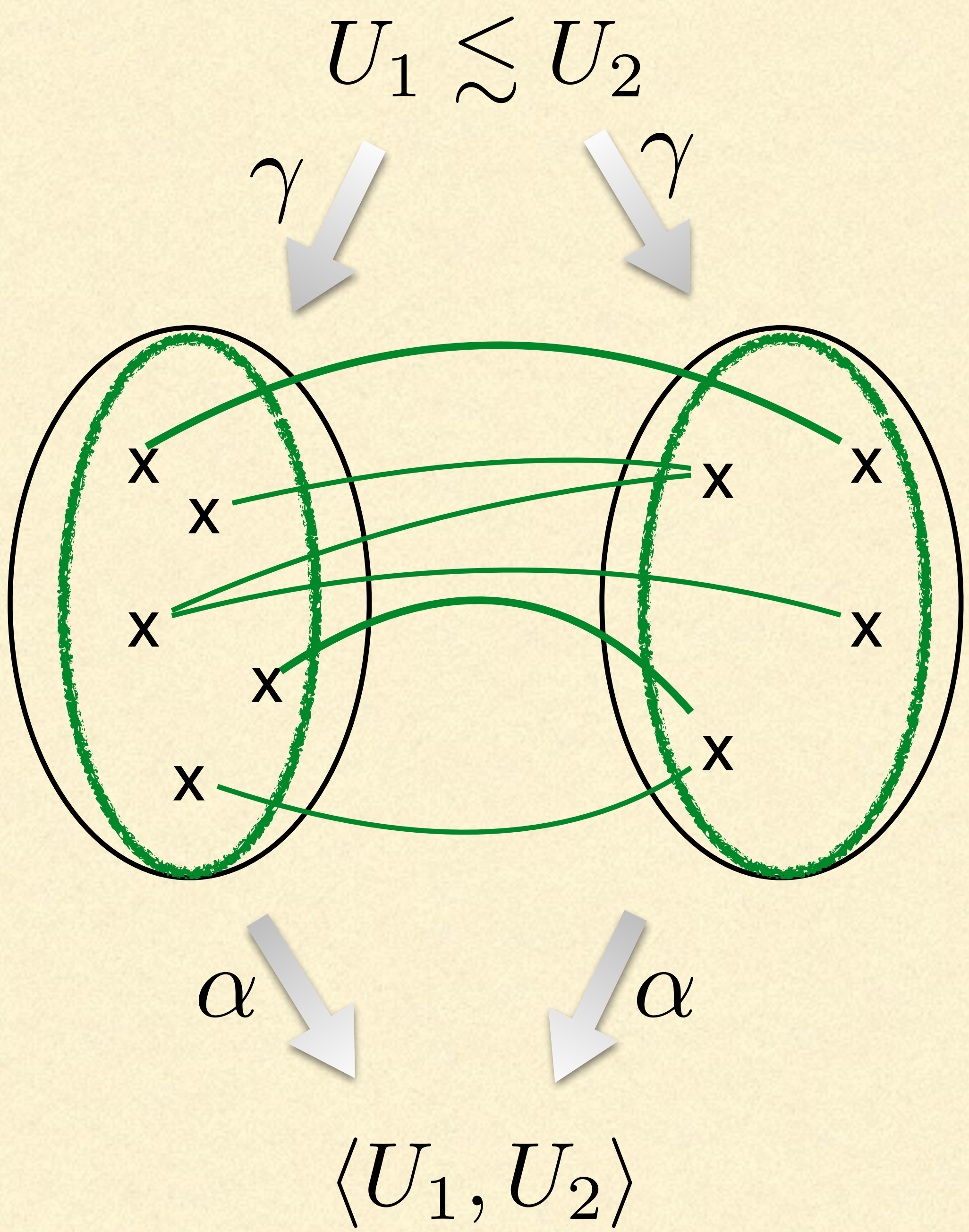
consistent subtyping is *not* transitive!

$\mathsf{Int} \lesssim ?$ and $? \lesssim \mathsf{Bool}$ but $\mathsf{Int} \not\lesssim \mathsf{Bool}$

PROP 2: TRANSITIVE "KERNEL"



PROP 2: TRANSITIVE "KERNEL"

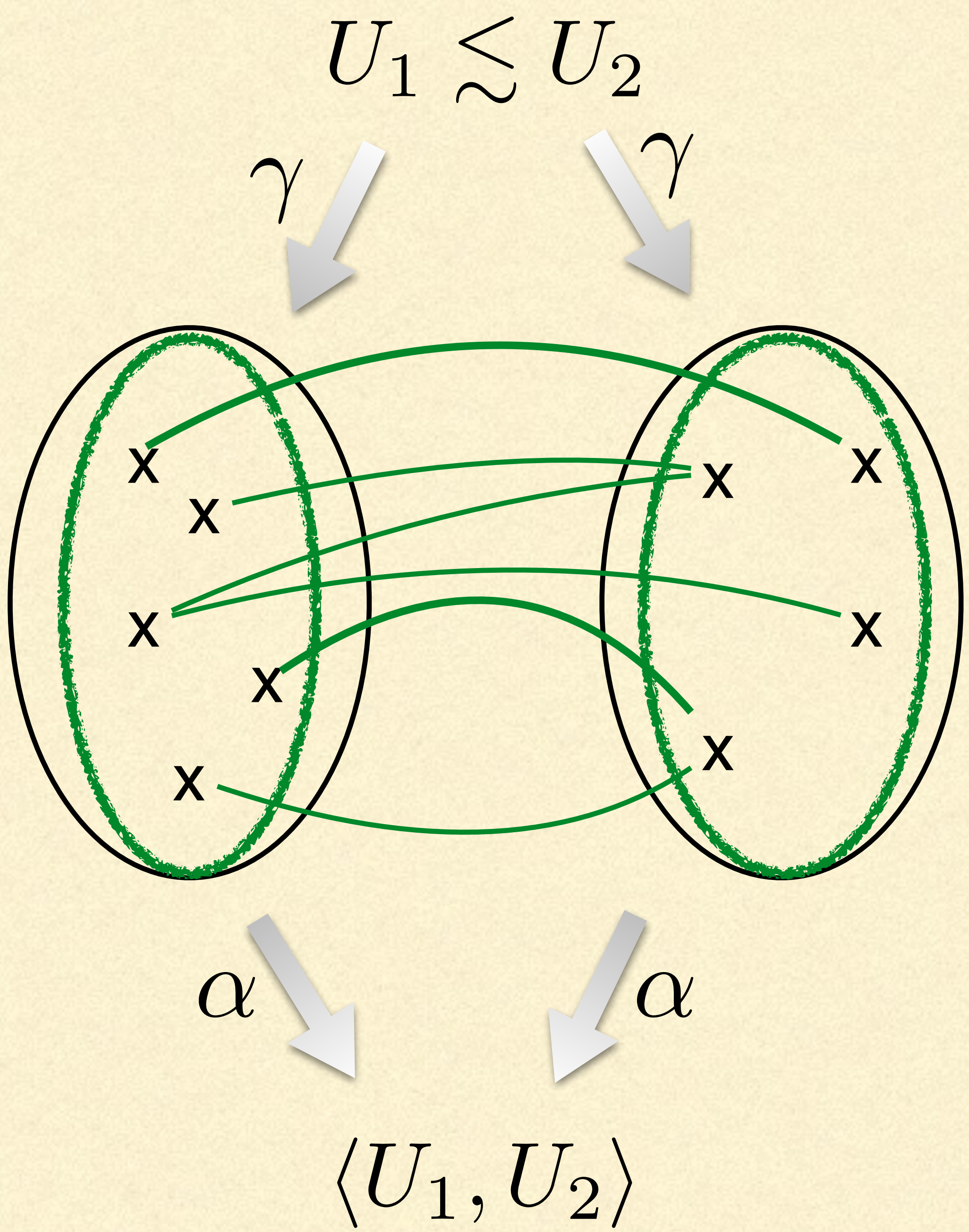


No "stray points" in concretization

$U_1 <: U_2$

Static Subtyping for Gradual Types!

PROP 2: TRANSITIVE "KERNEL"



But instances may acquire relevant evidence at runtime!

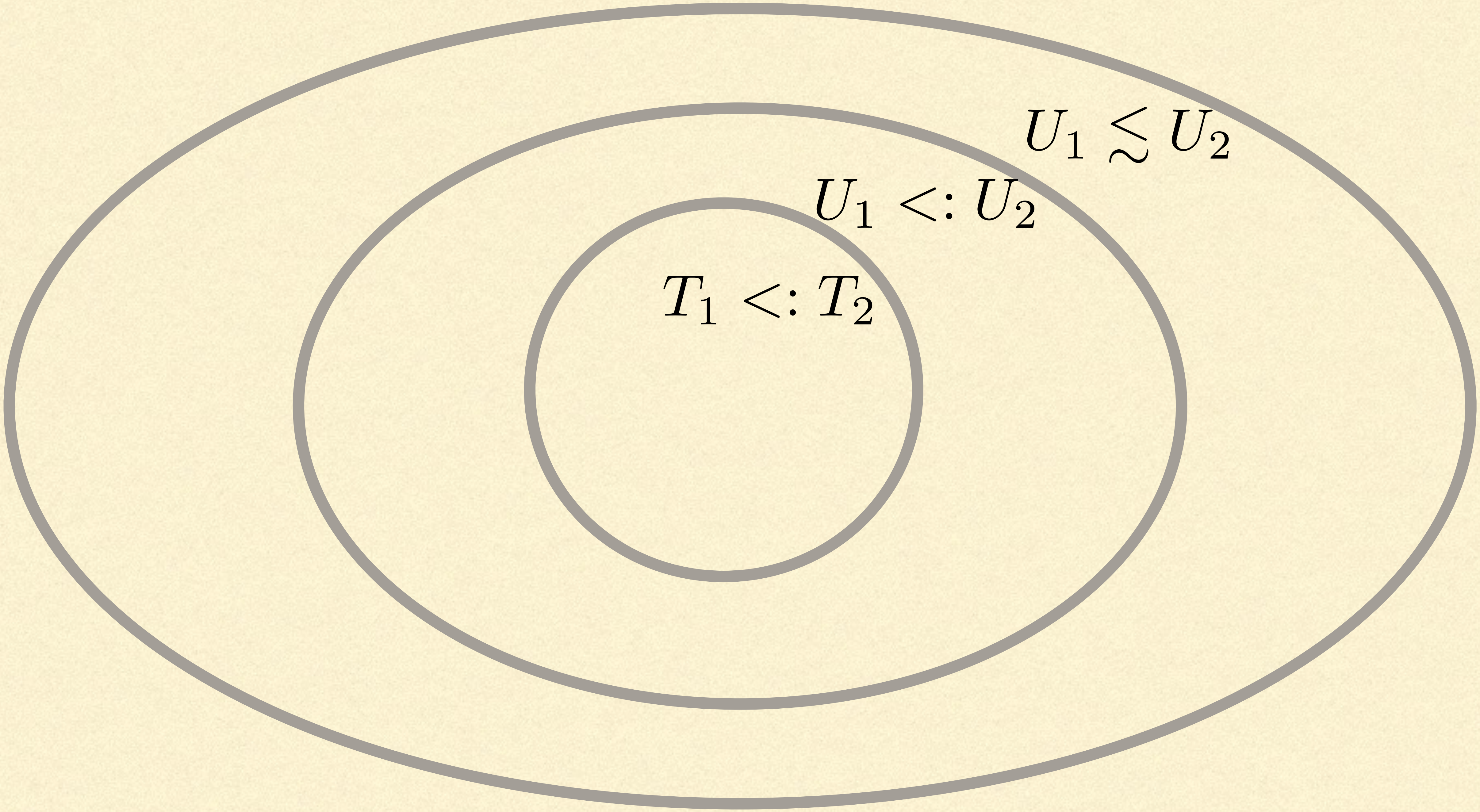
$\langle \texttt{Int}, \texttt{Int} \rangle \vdash \texttt{?} \lesssim \texttt{Int}$

$\langle \texttt{Int}, \texttt{Int} \rangle \vdash \texttt{Int} \lesssim \texttt{?}$

$\Downarrow$

$\langle \texttt{Int}, \texttt{Int} \rangle \vdash \texttt{?} \lesssim \texttt{?}$

HAPPY MEDIUM



INTO THE WAYBACK MACHINE

[Siek 08] Gradual Typing for Objects

Proposition 2 (Properties of Consistent-Subtyping). *The following are equivalent:*

1. $\sigma \lesssim \tau$,
2. $\sigma <: \sigma'$ *and* $\sigma' \sim \tau$ *for some* σ', *and*
3. $\sigma \sim \sigma''$ *and* $\sigma'' <: \tau$ *for some* σ''.

Same "Gradual Subtyping" Relation

2 => 1 and 3 => 1 are trivial
Need to think about converses

SPARSE EVIDENCE INSERTION

$U_1 <: U_2$ ➡ $\bullet \vdash U_1 <: U_2$

$U_1 \lesssim U_2$ ➡ $\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$

SPARSE REASONING

$$\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$$

$$\bullet \vdash U_2 <: U_3$$

$$\bullet \vdash U_3 <: U_4$$

$$\vdots$$

$$\bullet \vdash U_{k-2} <: U_{k-1}$$

$$\langle U_{k-1}', U_k' \rangle \vdash U_{k-1} \lesssim U_k$$

transitivity

$$\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$$

$$\bullet \vdash U_2 <: U_{k-1}$$

$$\langle U_{k-1}', U_k' \rangle \vdash U_{k-1} \lesssim U_k$$

SPARSE REASONING

$$\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$$
$$\bullet \vdash U_2 <: U_{k-1}$$
$$\langle U_{k-1}', U_k' \rangle \vdash U_{k-1} \lesssim U_k$$

$$\langle U_1', U_2' \rangle \vdash U_1 \lesssim U_2$$
$$\langle U_{k-1}', U_k' \rangle \vdash U_{k-1} \lesssim U_k$$

Need to reconstruct the "gap":
Threesome Calculus!
(eager checking semantics)

FURTHER WORK IN PROGRESS

- Greatest-Fixed Point Characterization of Gradual Subtyping
  - Directly captures “transitive kernel” concept
  - May help with proofs/semantics
- Proof of sparse/dense evaluation equivalence

CAST CALCULI <=> GRADUAL?

Threesomes, With and Without Blame *

Jeremy G. Siek
University of Colorado at Boulder
jeremy.siek@colorado.edu

Philip Wadler
University of Edinburgh
wadler@inf.ed.ac.uk

Gradual Session Types*

ATSUSHI IGARASHI, Kyoto University, Japan
PETER THIEMANN, University of Freiburg, Germany
VASCO T. VASCONCELOS, LaSIGE, Faculty of Sciences, Universit
PHILIP WADLER, University of Edinburgh, Scotlan