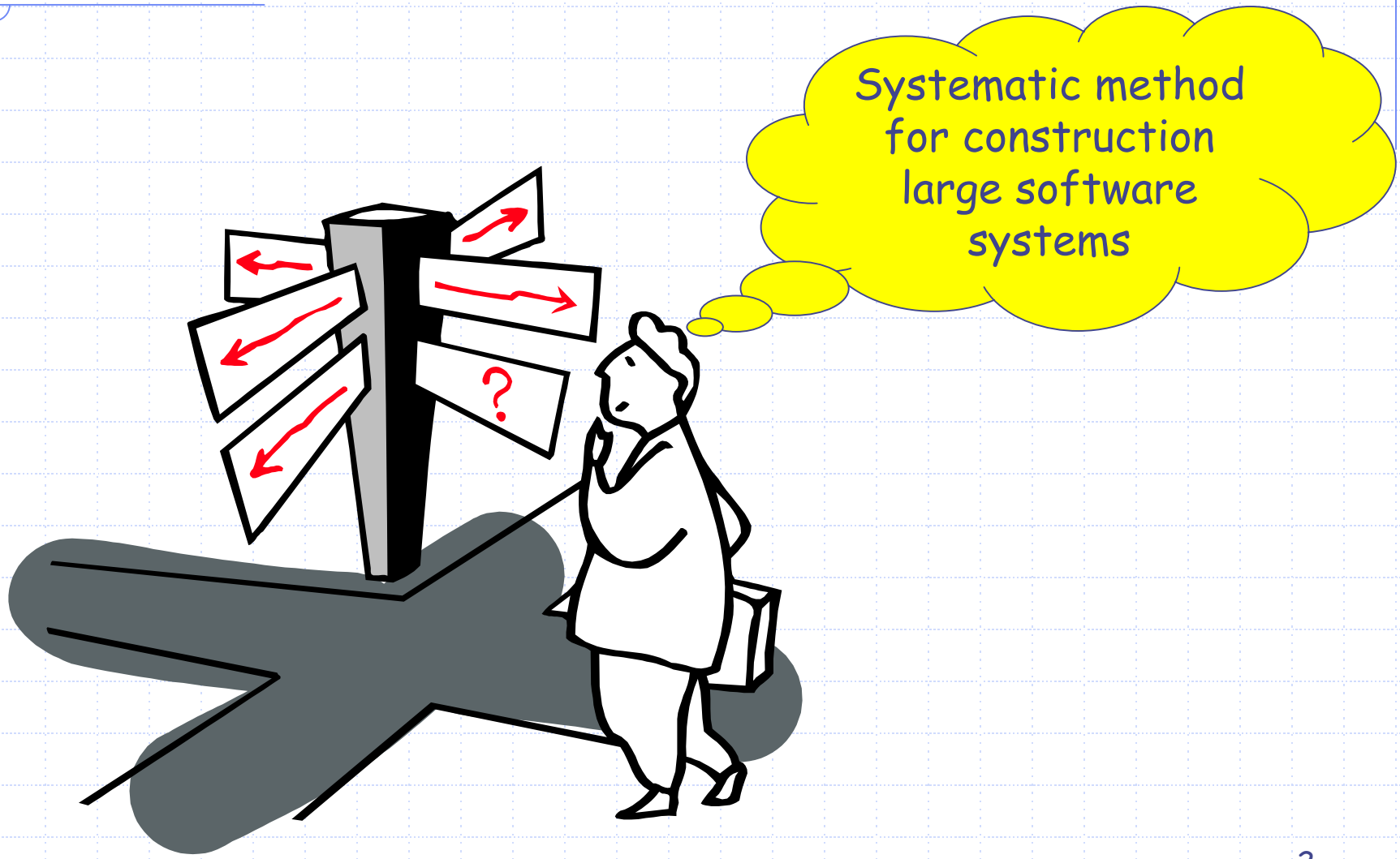
Classboxes:
An Experiment in Modeling Compositional Abstractions using Explicit Contexts

Markus Lumpe
Department of Computer Science
Iowa State University

Jean-Guy Schneider
Faculty of Information & Communication Technologies
Swinburne University of Technology

Do we really need a specially designed composition language?



What are the obstacles?

- Dependence on position and arity
- Changes may affect the whole system
- The world is dynamic

And the winner is …

Forms

What are forms?

- First-class namespaces with a small set of purely asymmetric operators
- Component interfaces, components, and composition mechanisms
- Compile-time and run-time entities

What else can we say about forms?

- Forms are not bound to a particular computational model.
- Forms have to be combined with a concrete target system.

The $\lambda\mathcal{F}$-Calculus

$F, G, H ::= \langle\rangle$		*empty form*
	$\mid X$	*form variable*
	$\mid F\langle l = V\rangle$	*binding extension*
	$\mid F \oplus G$	*form extension*
	$\mid F \setminus G$	*form restriction*
	$\mid F \rightarrow l$	*form dereference*
	$\mid F[G]$	*form context*

$V ::= \varepsilon$		*empty value*
	$\mid a$	*abstract value*
	$\mid M$	*$\lambda\mathcal{F}$-value*

$M, N ::= F$		*form*
	$\mid M.l$	*projection*
	$\mid \lambda(X)\, M$	*abstraction*
	$\mid M\ N$	*application*
	$\mid M[F]$	*$\lambda\mathcal{F}$-context*

How can we represent objects in the $\lambda\mathcal{F}$-calculus?

C_α =

let

$\Delta_C = \lambda(\text{State})\,(\boxed{\text{Methods}_C})\,[\text{State}]$

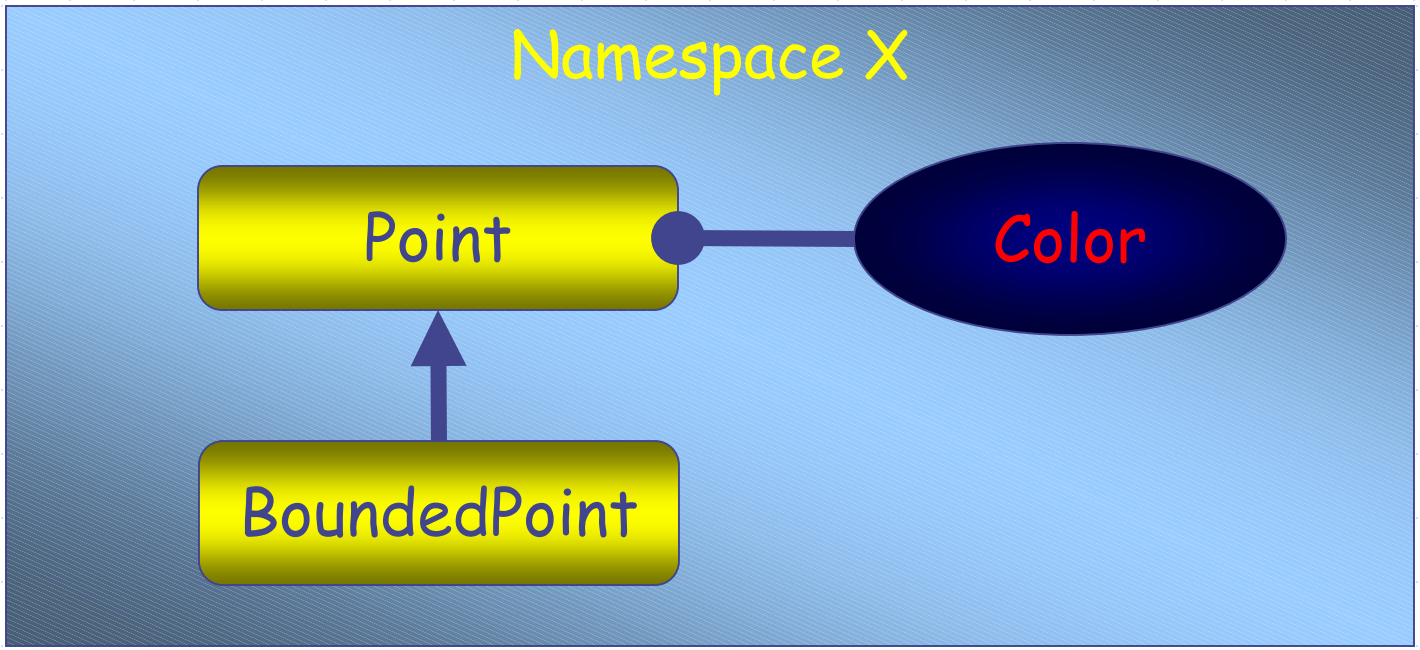
$G_C = \lambda(\gamma)\,\lambda(I)\,P_\beta \oplus \Delta_C\langle I \oplus (\boxed{\text{State}_C})\rangle$

$W_C = \lambda(\gamma)\,\mu_{self}\langle((\gamma \to C_\alpha).G\,(\beta \oplus \gamma))\,[self]\rangle$

in

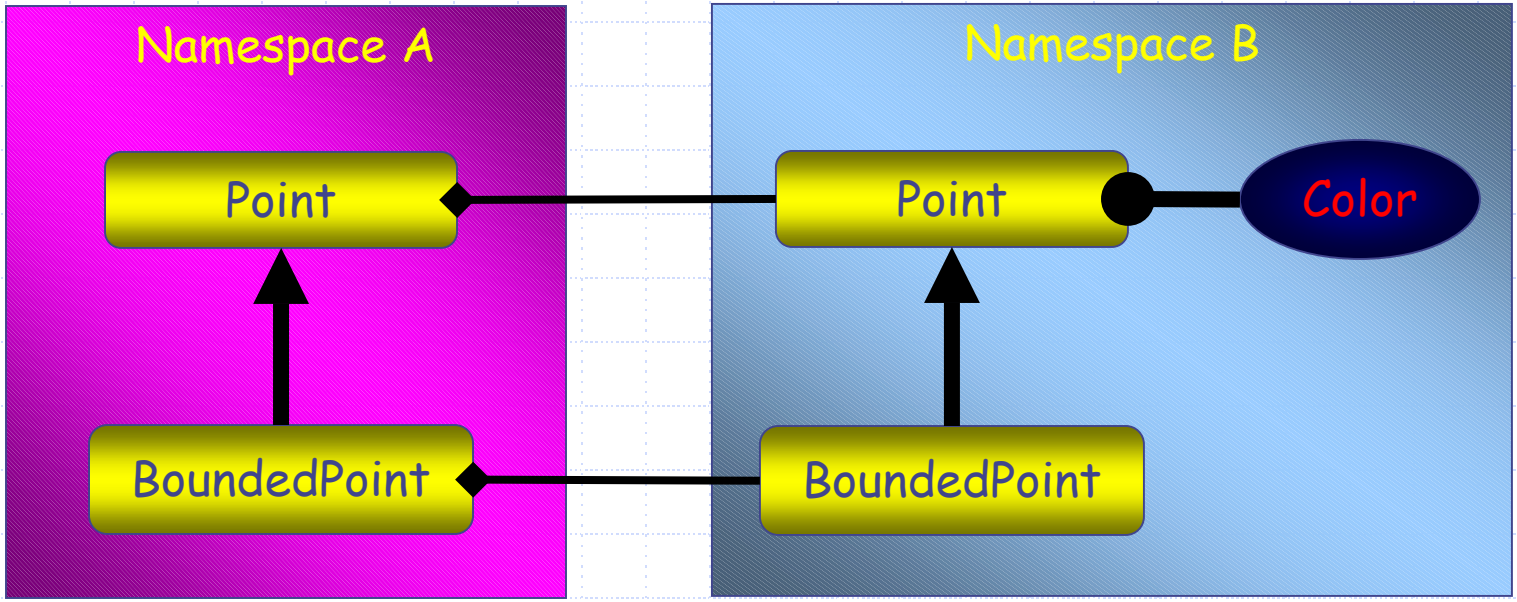
$\langle G = G_C,\ W = W_C\rangle$

How can we denote changes?



Class extensions are used to add or refine features of existing classes in a namespace.

How can we manage changes?



Class extensions are only visible in the namespace in which they are defined.

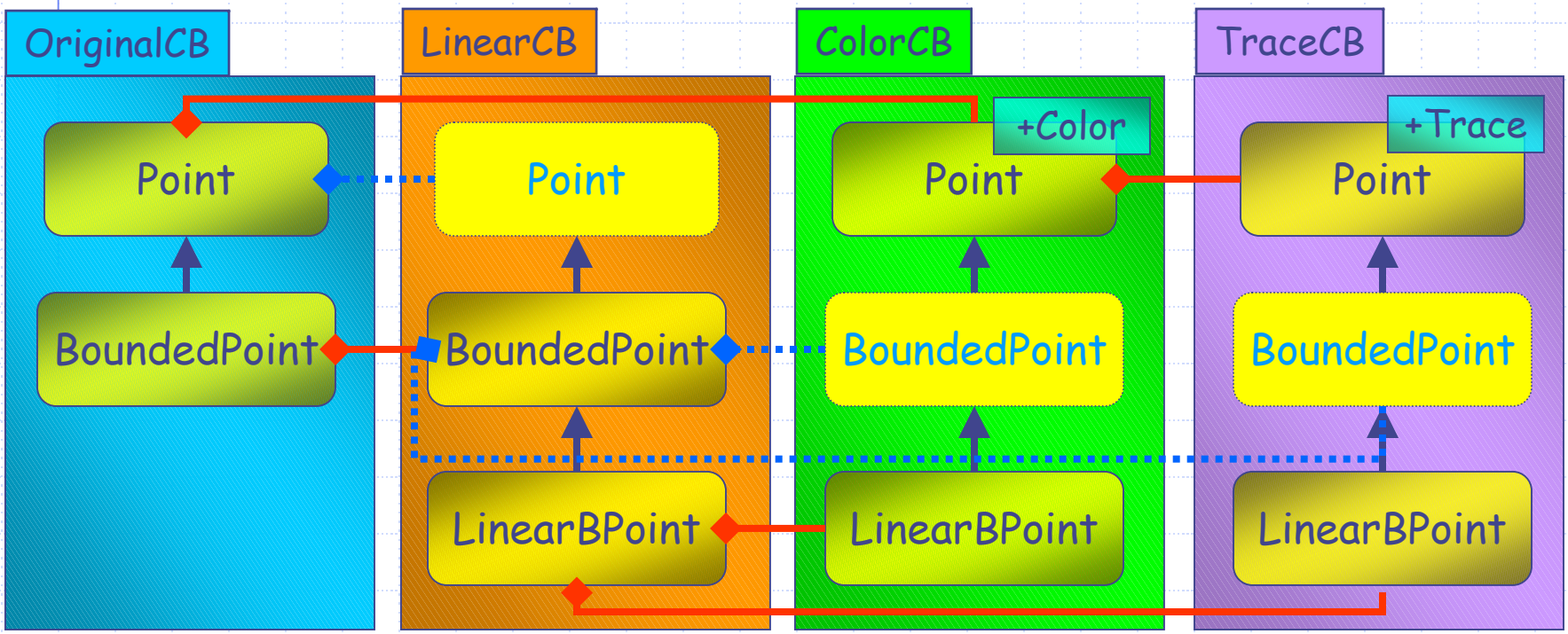
A new module system: **Classboxes**

- Classboxes define explicitly named scopes.
- Classboxes support import and local refinement of classes.

What are the available operations?

- Import a classes
- Introduction of subclasses
- Extension of classes
- Inclusion of new behavior

A Point Class Hierarchy



Extension

B^{E}_{β} =

let

$\Delta_B = \lambda(State)\,(\boxed{Methods_B})\,[State]$

$G_B = \lambda(Class)$

$\lambda(\gamma)\,\lambda(I)$

let

$P = \mu_{self}\langle(Class.G\ \gamma)\ [self]\rangle\ I$

in

$P \oplus \Delta_B\langle I \oplus (\boxed{State_B}\langle super = P\rangle)\rangle$

in

$\langle G = G_B\rangle$

Inclusion

B^{I}_{β} =

let

Δ_B = λ(State) ($\boxed{\text{Methods}_B}$) [State]

G_B = λ(Class)
$\lambda(\gamma)\ \lambda(I)$
let
P = (Class.G γ) I
in
$P \oplus \Delta_B\langle I \oplus (\boxed{\text{State}_B}\langle \text{original} = P\rangle)\rangle$

in

$\langle G = G_B \rangle$

How can we apply extension and inclusion?

Extension:

$$C_\alpha = (\text{lookupClass}\langle C, \alpha'\rangle)\langle G = B^E_\beta.G\ (\text{lookupClass}\langle C, \alpha'\rangle)\ \rangle$$

Inclusion:

$$C_\alpha =$$

let

$$G_C = \lambda(\gamma)\ (B^I_\beta.G\ (\text{lookupClass}\langle C, \alpha'\rangle))\ \gamma$$

$$W_C = \lambda(\gamma)\ (\text{lookupClass}\langle C, \alpha'\rangle).W\ (\beta \oplus \gamma)$$

in

$$\langle G = G_C, W = W_C\rangle$$

Why is the encoding of classboxes in the $\lambda\mathcal{F}$-calculus useful?

- Expressiveness of the $\lambda\mathcal{F}$-calculus
- Precise semantics of classboxes
- Discovery of new operations

Can our results be applied to an industry-strength language?

- Yes, C# with explicit class extensions:

```
namespace ColorCB
{
    using System.Drawing;
    using Point = OriginalCS.Point append Color;
    using LinearBoundedPoint = LinearCB.LinearBoundedPoint;

    extension Color {
        private Color color;

        public Color Color { get { return color; }
                             set { color = value; } } }
}
```