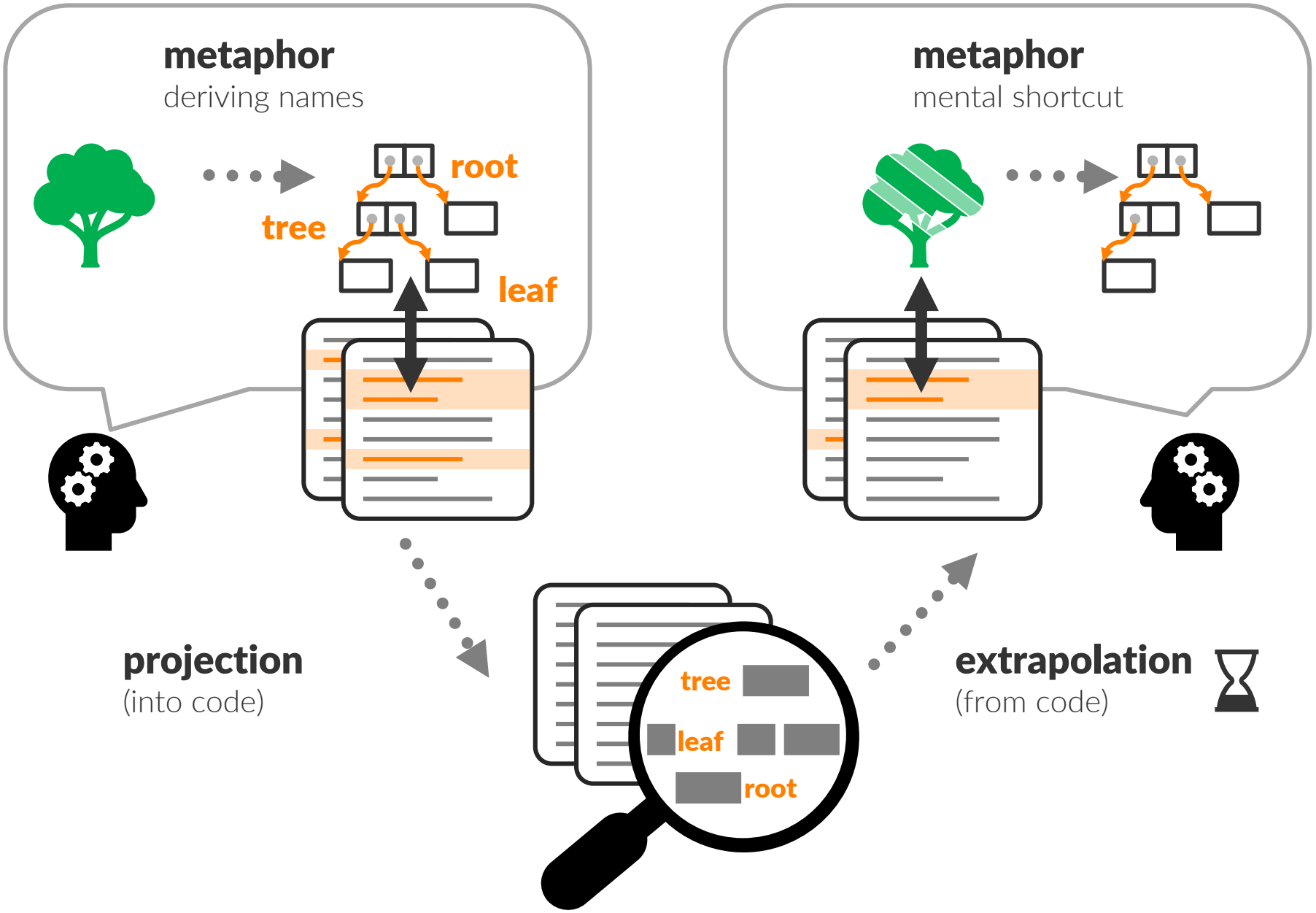
Towards
Concept-aware Programming Environments
for Guiding Software Modularity

Toni Mattis, Patrick Rein, Stefan Ramson,
Jens Lincke, Robert Hirschfeld

Software Architecture Group
Hasso Plattner Institute, University of Potsdam, Germany

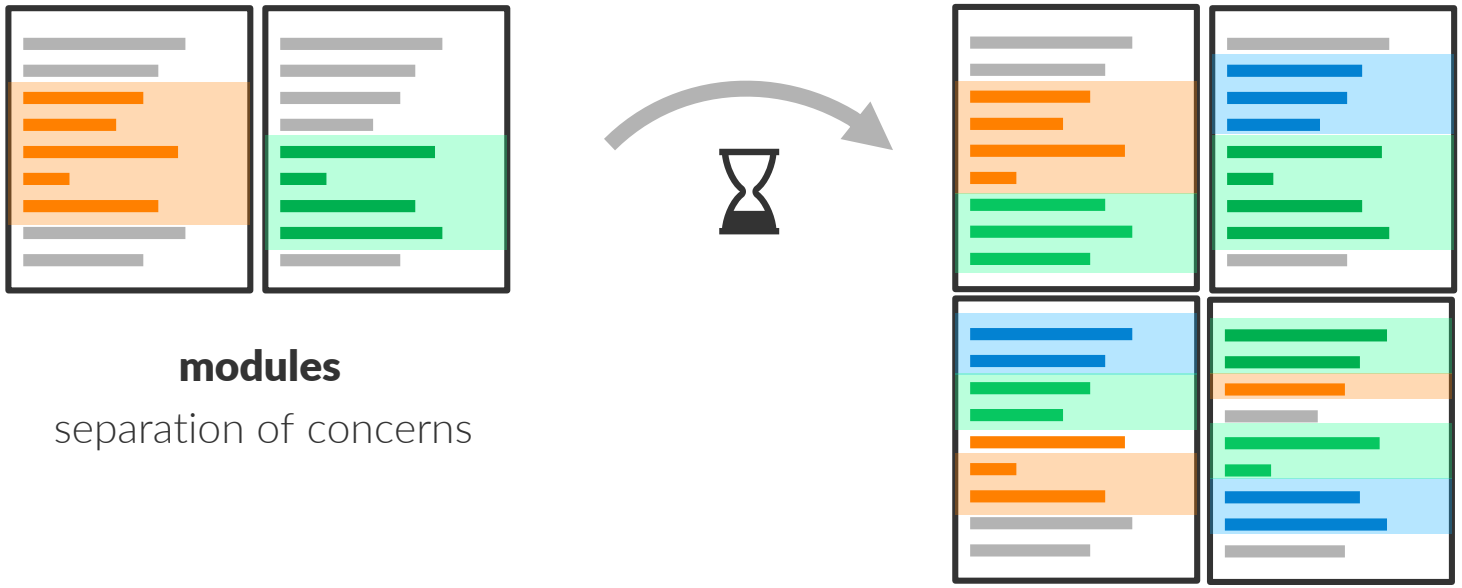
PX/17.2 22 Oct. 2017, Vancouver


metaphor
deriving names
root
tree
leaf
metaphor
mental shortcut
projection
(into code)
tree
leaf
root
extrapolation
(from code)

Problem Statement

With growing code bases…

- Concepts tend to scatter and entangle
- Programmers need more time to recognize concepts



modules

separation of concerns

scattering / tangling

Problem Statement

With growing code bases…

» Concepts tend to scatter and entangle

» Programmers need more time to recognize concepts

Consequences of incomplete recognition

» Architectural Drift: Code written in the wrong place

» Duplication: Missed existing functionality

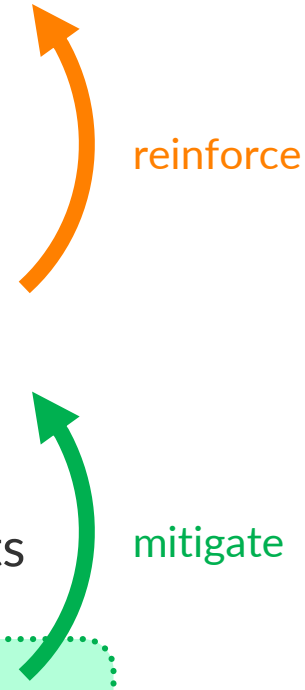
» Inconsistent Naming: Metaphor misunderstood

Approaches

» Proactive: Tools/Language features to maintain concepts
[e.g. AOP/COP/ … discipline during development!]

» Retroactive: Tools to recover concepts

» Proactive: Tools to support concept maintenance



Basic Concept Model

concept labels
which concept a name belongs to

concepts
prevalent names

```
Canvas » draw: anObject
  ^ anObject drawOn: self
```

draw, canvas, fill, ...

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

```
Morph » bounds: newBounds
  self position: newBounds topLeft;
  extent: newBounds extent.
```

bounds, position, extent, ...

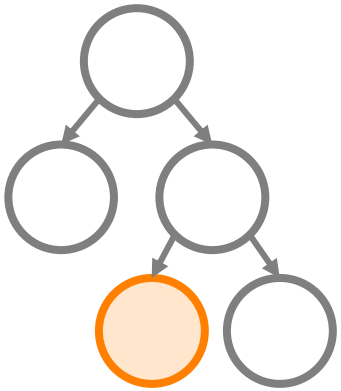
Names

- Typical **identifiers** can consist of **multiple names**
  - Camel Case:
    `fillRectangle` → `fill, rectangle`
  - Underscore:
    `fill_rectangle` → `fill, rectangle`
  - Acronyms:
    `HTTPServer` → `http, server`
  - Multi-part message names:
    `fillRectangle:color:` → `fill, rectangle, color`

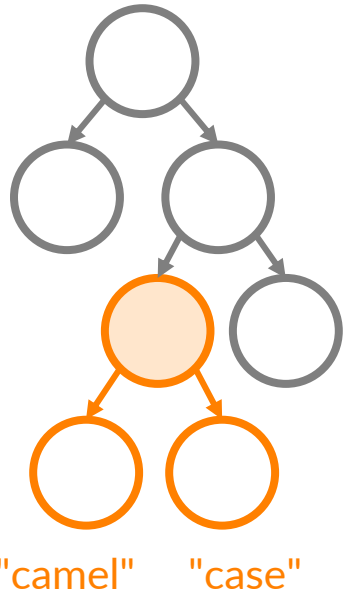
- Constant **strings** (or symbols) can be relevant, too:
  - `config['backgroundColor']`
  - `config at: #backgroundColor`
    → `background, color`

AST-based View

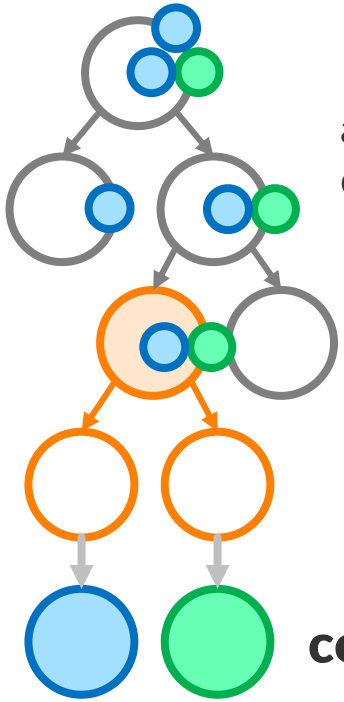
AST
Abstract Syntax Tree



Expanded Identifiers



Concept AST



aggegated concept distribution

concept labels

Maintaining Concepts

1. Automated bootstrap phase ("concept mining")
   - Deciding which names belong to **the same / a different** concept
   - Setting the **granularity**
   - Selecting useful **data**/features
2. User refinements
   - Types of **operations** provided to users
   - (Partially) **re-running** concept mining
   - **Synchronizing** refinements between team members

Maintaining Concepts

1. Automated bootstrap phase ("concept mining")

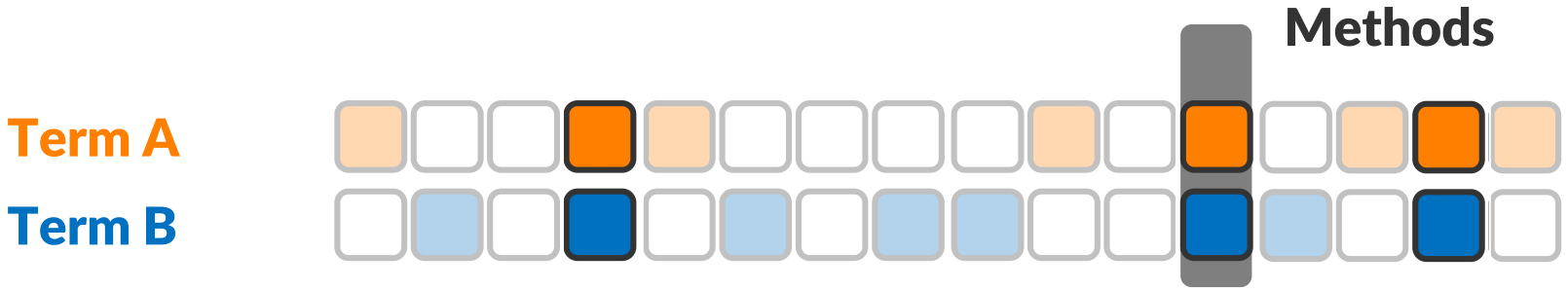
   - Deciding which names belong to **the same / a different** concept
   - Setting the initial **granularity**
   - Selecting useful **data**/features

2. User refinements

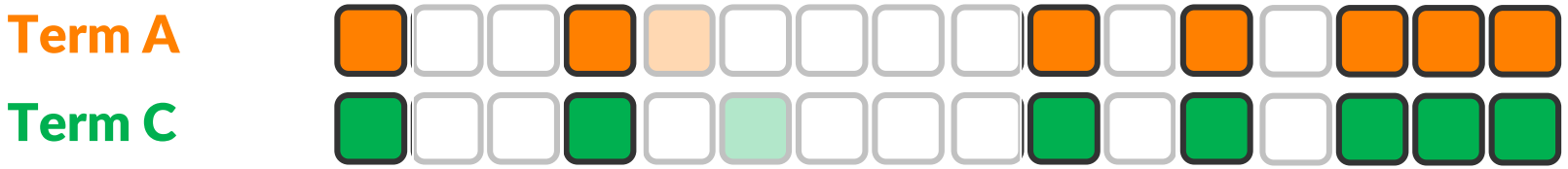
   - Types of **operations** provided to users
   - (Partially) **re-running** concept mining
   - **Synchronizing** refinements between team members

Distributional Hypothesis

» Lexical tokens with a similar **distribution** have a similar meaning



$$f(A \wedge B) = 3/16$$



$$f(A \wedge C) = 7/16$$

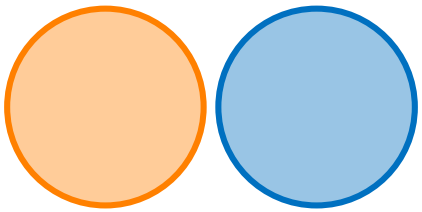
$E[f(A \wedge B)] = 4/16$ (if both were random)

Concept Mining: Co-occurrence

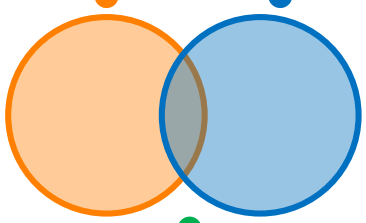
» Names belonging to the same concept co-occur more frequently in the same scope

$$\hat{f}(a,b) = \frac{f(a \wedge b)}{f(a)f(b)} = \cdots$$

proportion of… methods containing **a** methods containing **b**

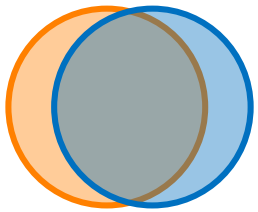


≈ 0.0 **(mutually exclusive)**



≈ 1.0 **(random)**

methods containing **a and b**



$\gg 1.0$ **(intentional?)**

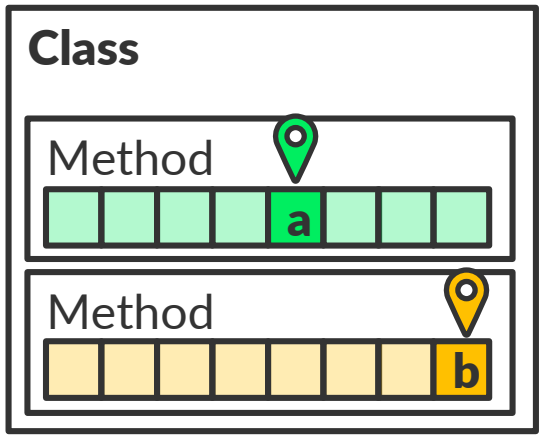
Co-occurrence

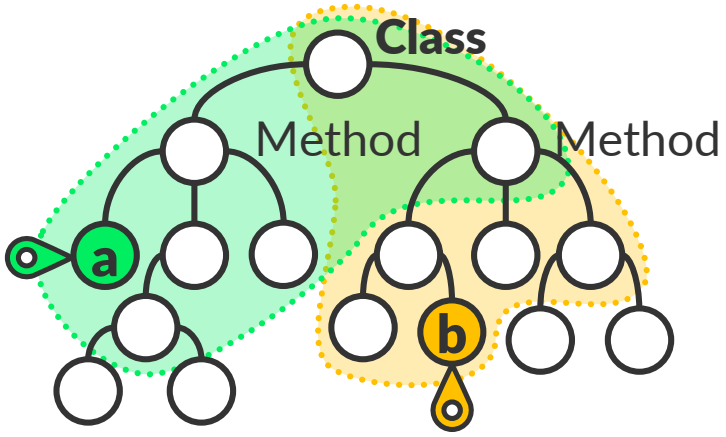
Examples (Squeak/Smalltalk Image)

a	b	$\hat{f}(a,b)$	
visit	accept	**70.1**	same design pattern
bounds	draw	**15.4**	geometry & drawing
collect	select	**6.8**	same API
parse	next	**2.2**	parsing & streams
collect	color	**1.5**	incidental
visitor	color	**0.0**	mutually exclusive

Co-occurrence Relations

- Same module
  - class, **method**, package
  - file
  - lexical scope
- Within certain distance
  - **… in the AST**
  - … in text
  - … in execution
- Edited close in time
  - Git commits
  - IDE interactions





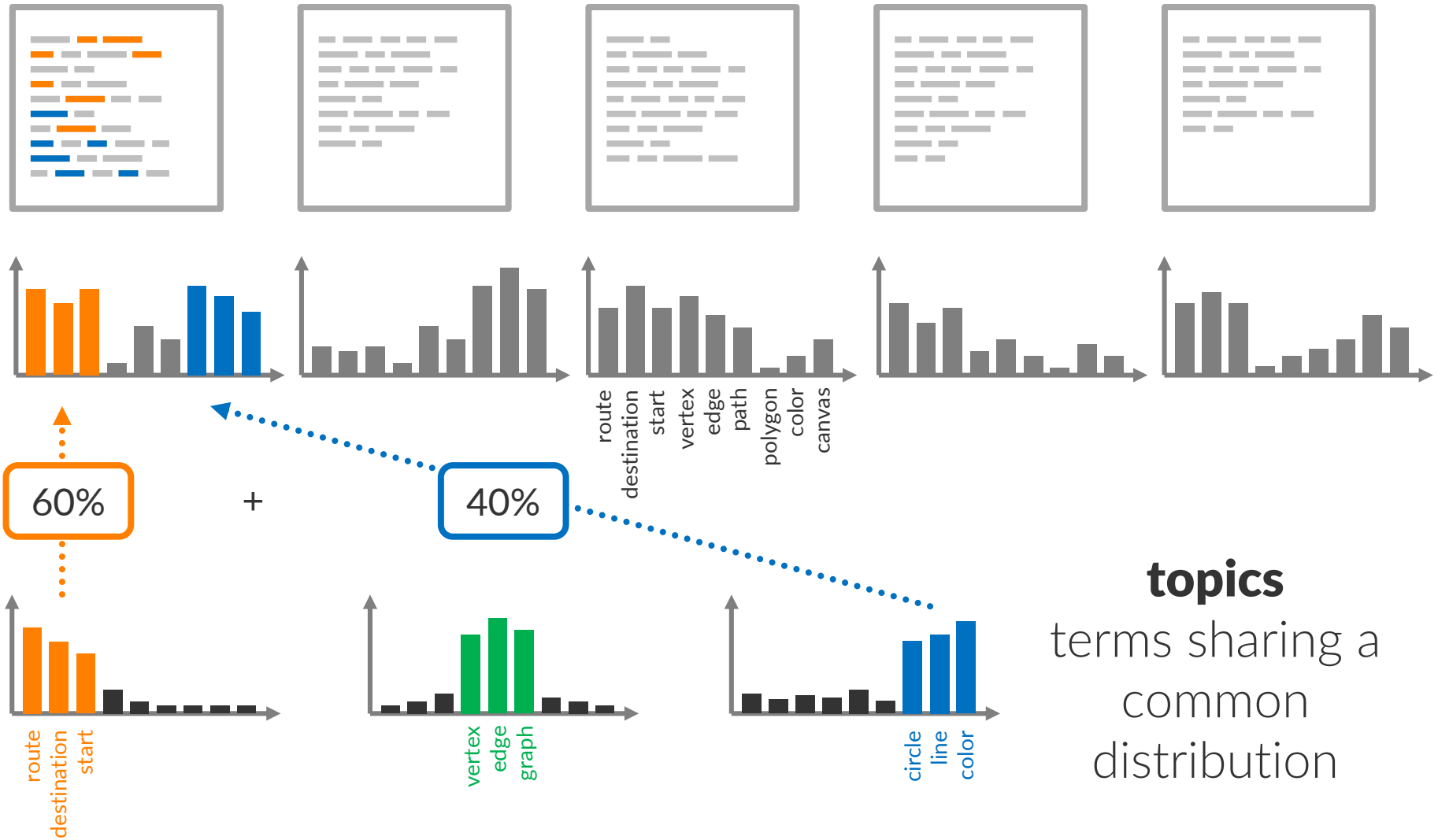
Concept Mining

- Clustering
  - Maximize intra-cluster **similarity**
  - Minimize inter-cluster **similarity**
  - One concept per name

co-occurrence, pointwise mutual information, cosine similarity, …

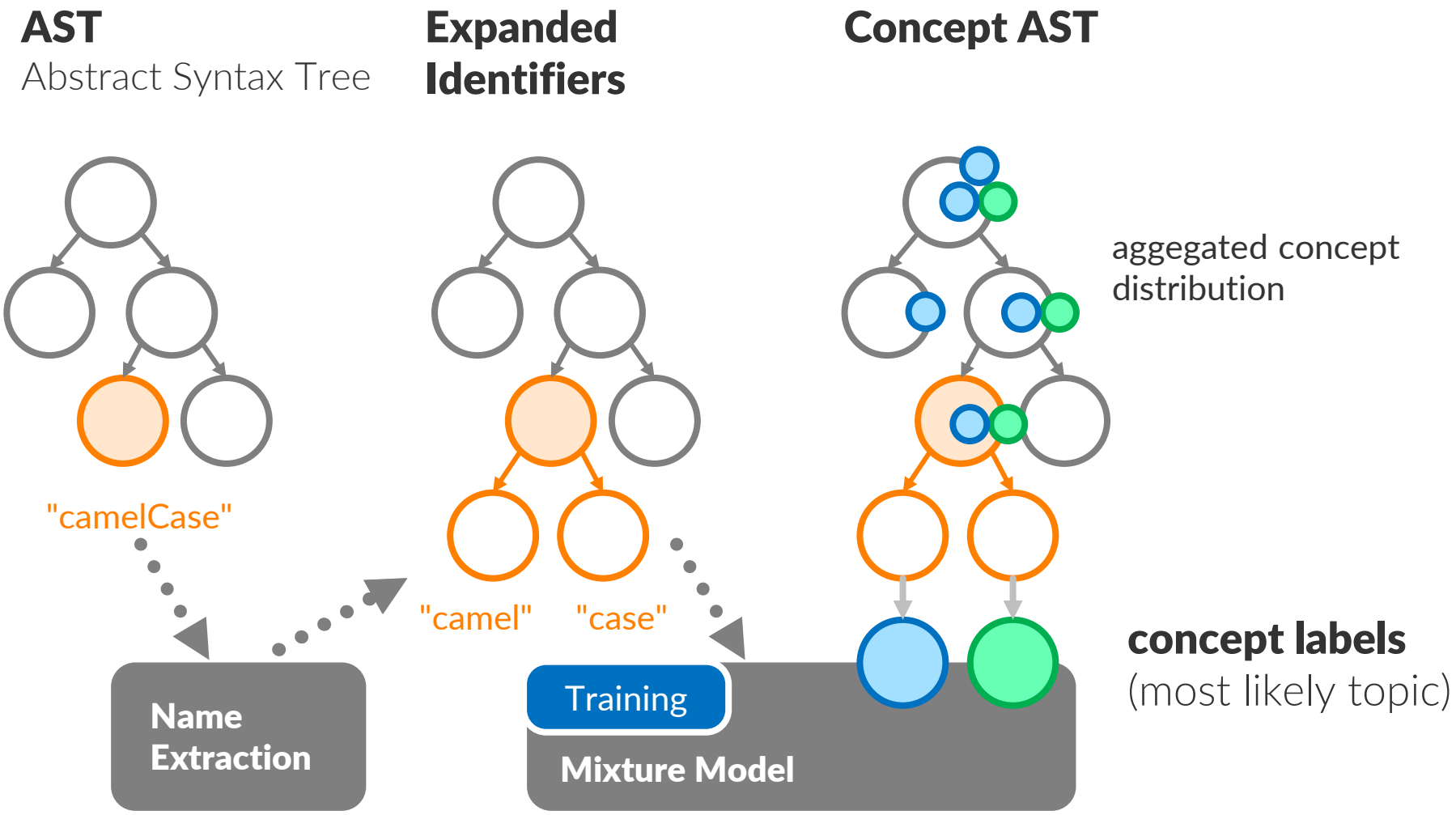
- Mixture Models
  - Every name has a **probability** of occurring in each concept
  - Bag-of-words (Topic Models)
  - Graph-based (Stochastic Block Models)

Topic Models



topics
terms sharing a common distribution

AST-based View



Composition & Abstraction Barriers

```
Canvas » draw: anObject
  ^ anObject drawOn: self
```

draw, canvas, fill, ...

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

mixing

uses
(implemented through)

```
Morph » bounds: newBounds
  self position: newBounds topLeft;
  extent: newBounds extent.
```

bounds, position, extent, ...

Composition & Abstraction Barriers

abstraction (concepts being defined)

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

concepts

use in their implementation

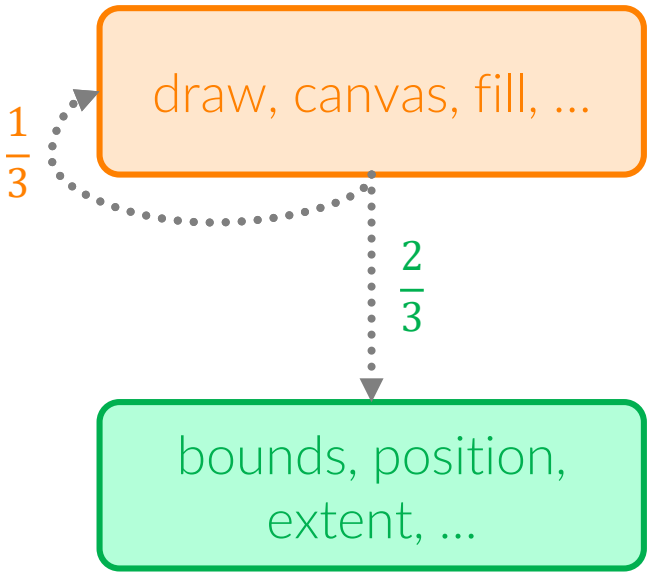
implementation (defining concepts)

Composition & Abstraction Barriers

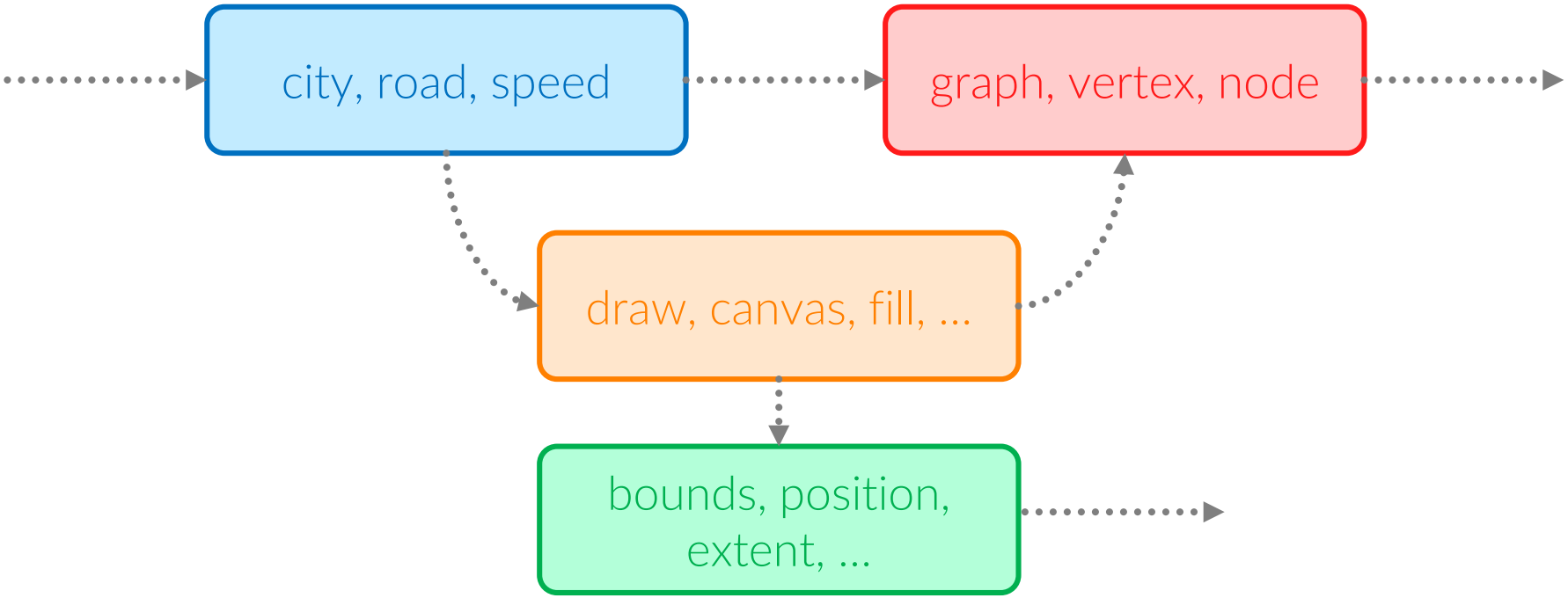
abstraction (concepts being defined)

```
Morph » drawOn: aCanvas
  aCanvas fillRectangle: self bounds.
```

implementation (defining concepts)



Concept Graph



Maintaining Concepts

1. Automated bootstrap phase ("concept mining")

   - Deciding which names belong to **the same / a different** concept
   - Setting the initial **granularity**
   - Selecting useful **data**/features

2. User refinements

   - Types of **operations** provided to users
   - (Partially) **re-running** concept mining
   - **Synchronizing** refinements between team members

Operations on Concepts

Reassign concept label:

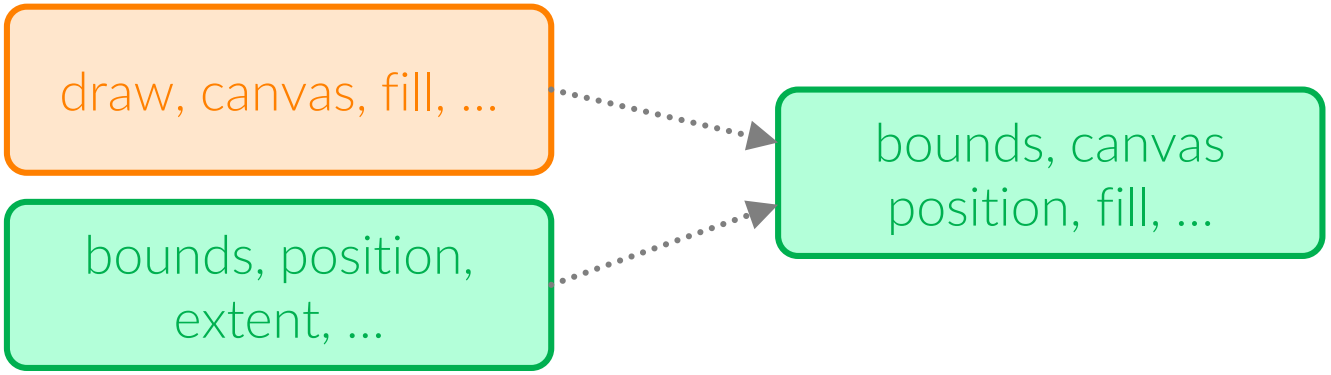
```
Morph » drawOn: aCanvas
    aCanvas fillRectangle: self bounds.
```

Challenges

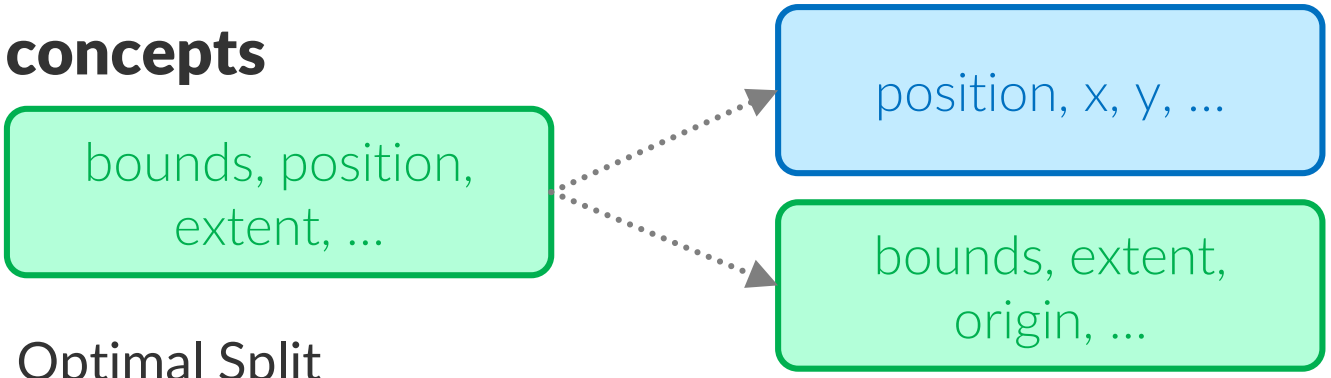
- Inconsistencies
  - Re-computing clustering might avalanche into many other concepts being **re-assigned** to restore optimality
  - Not doing so might leave programmers with lots of **manual re-assignment** work
- Synchronization
  - Share manual refinements across a team of programmers

Operations on Concepts

Merge concepts

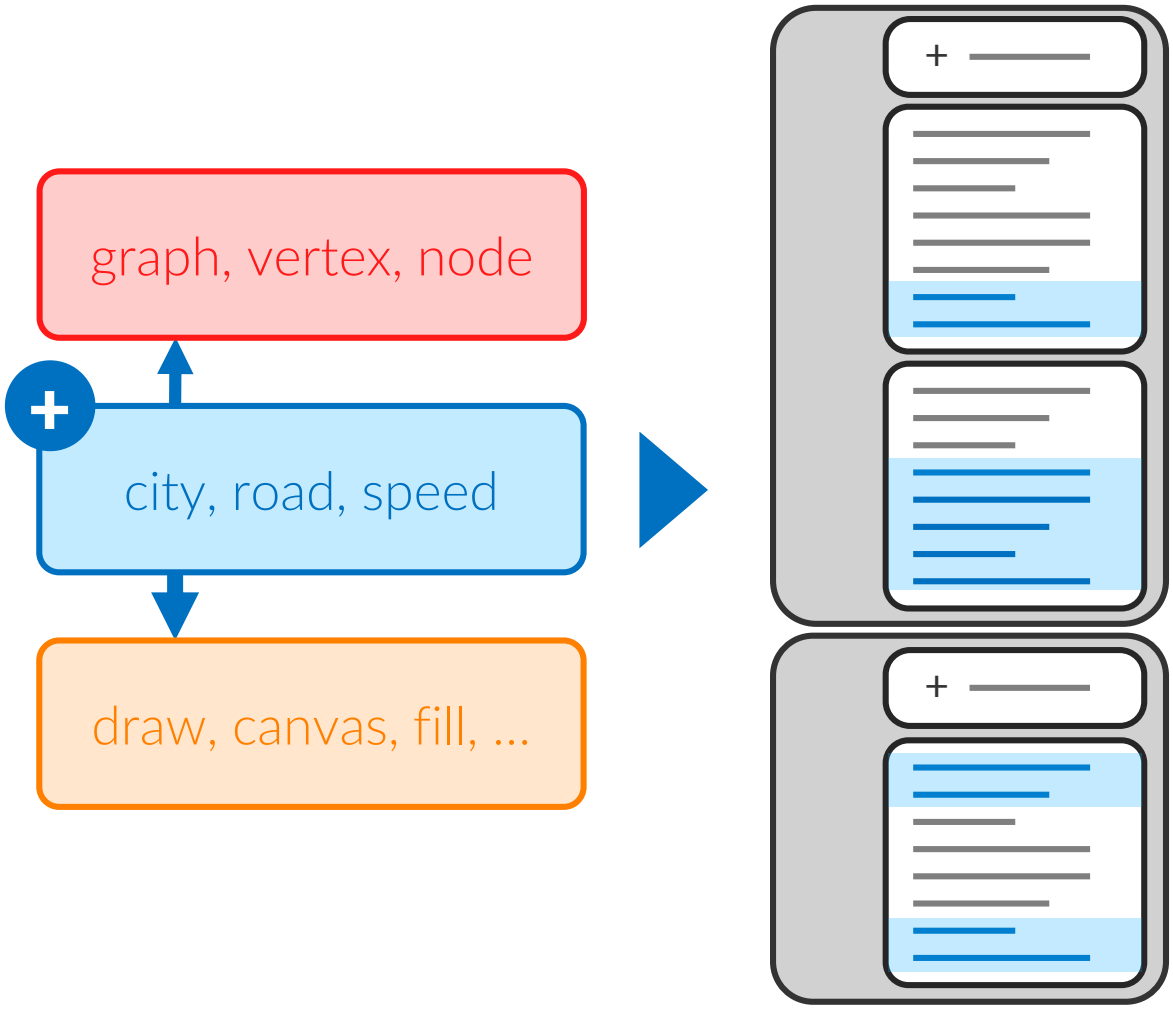


Split concepts

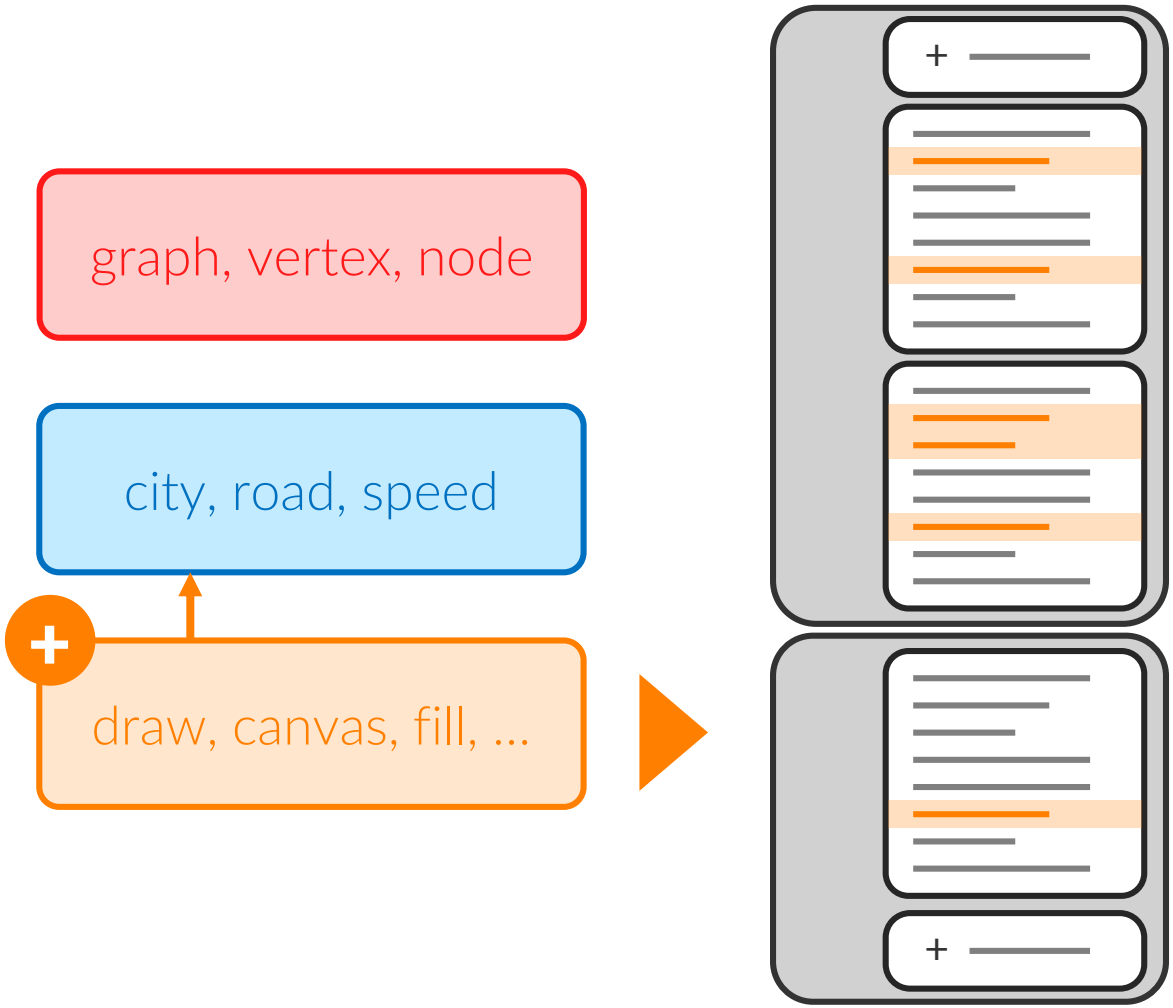


- Optimal Split
- Hierarchical Topic Model
- Manual / Semi-supervised

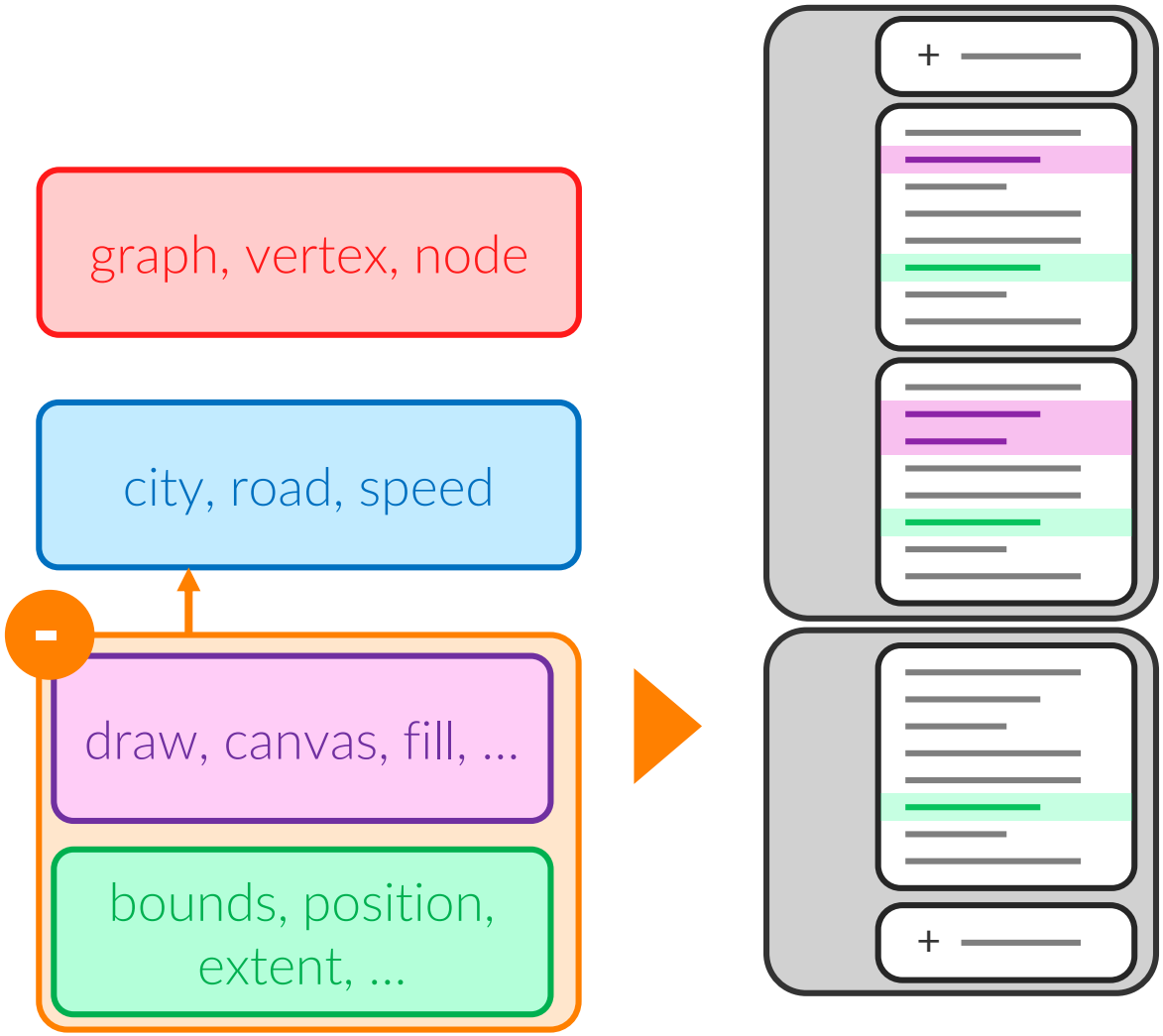
Exploring the Concept Graph



Exploring the Concept Graph

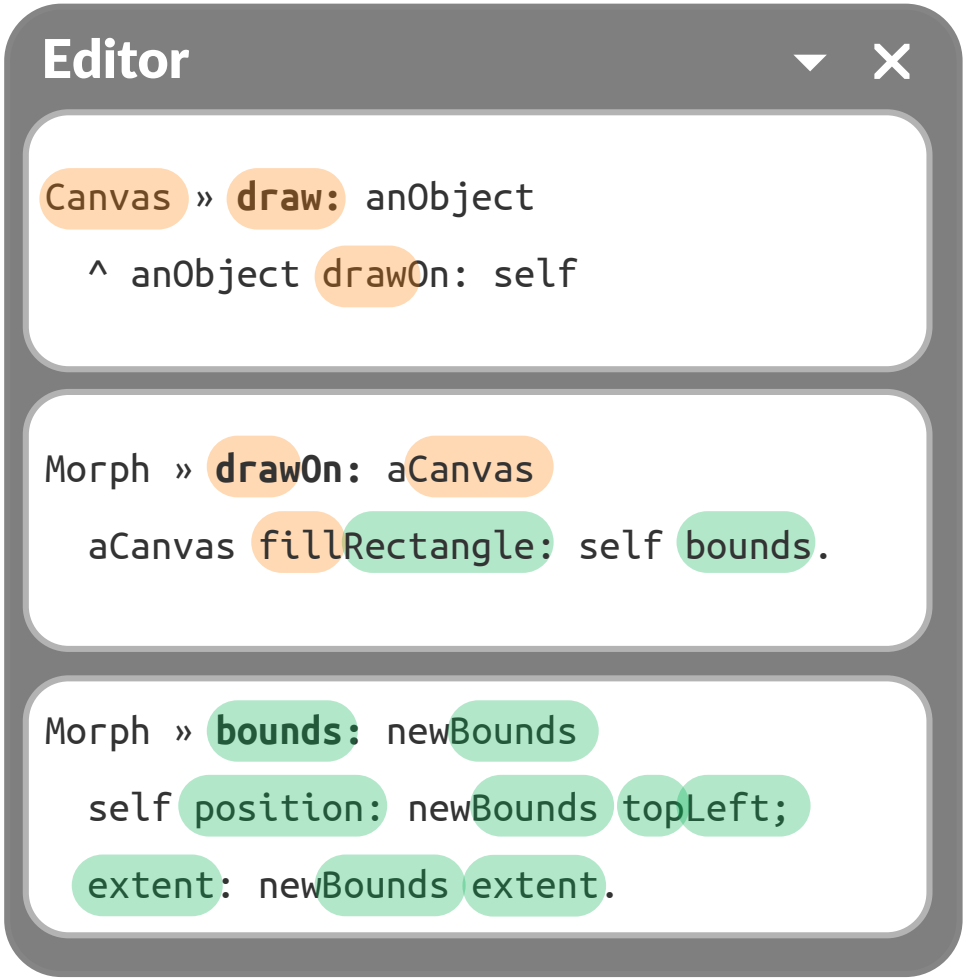


Exploring the Concept Graph



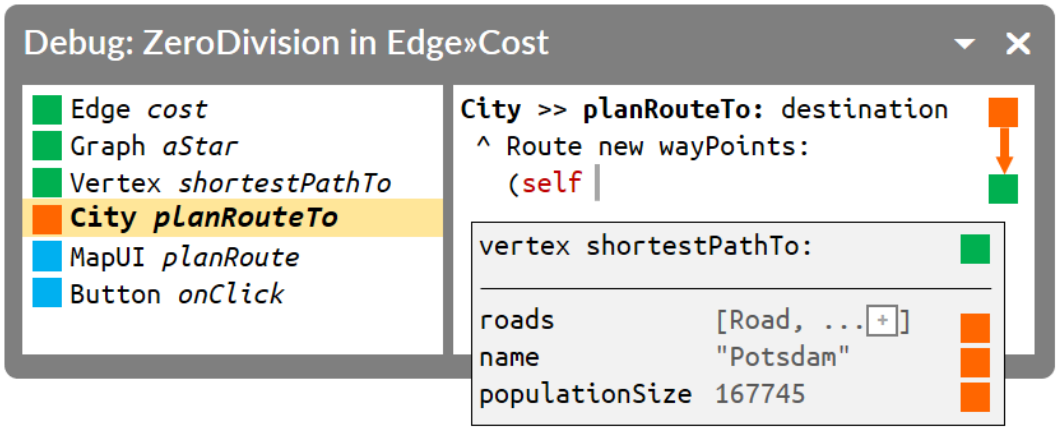
Concept-aware Tooling

- Highlight concepts in Code



Concept-aware Tooling

- Improve relevance of information displayed during
  - code completion
  - debugging

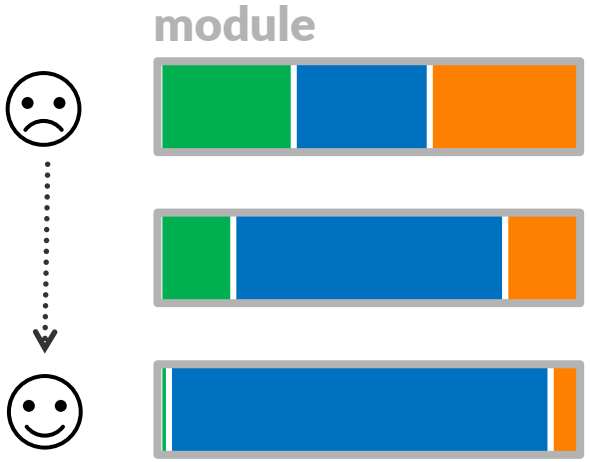


- In **live programming**
  - Arranging and prioritizing live objects and meta-objects
  - Live feedback on modularity, name choices, recommended code artifacts, …

A Perspective on Modularity

module entropy:

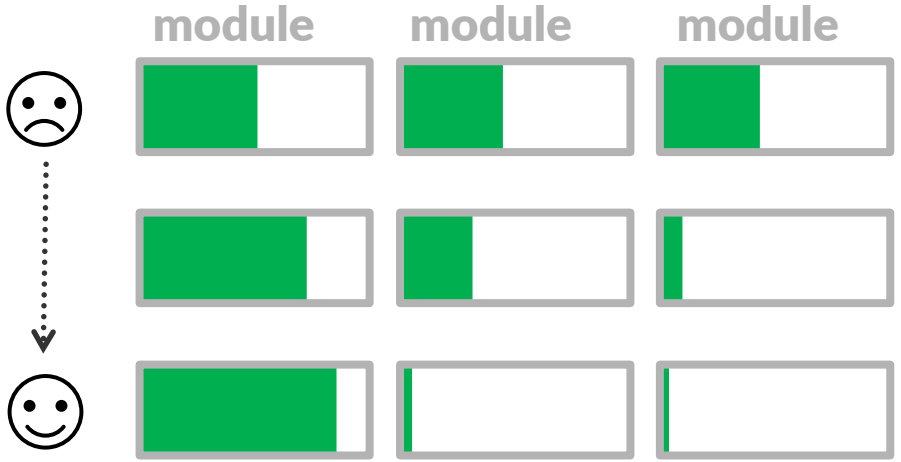
tangling



$$H(m) = -\sum_{c} p(c|m) \log_2 p(c|m)$$

concept entropy:

scattering



$$H(c) = -\sum_{m} p(m|c) \log_2 p(m|c)$$

…high values indicate need for refactoring or cross-cutting concerns

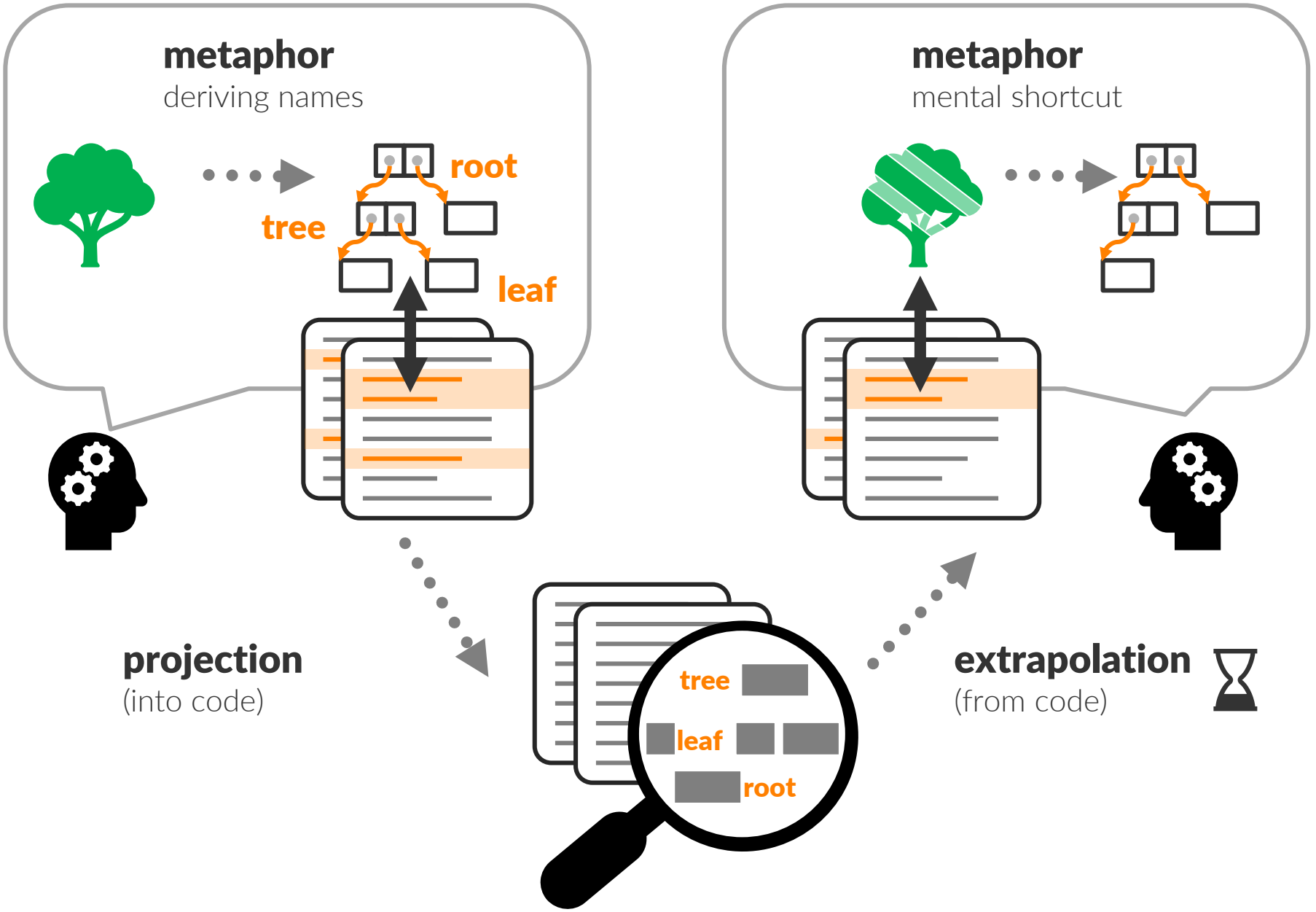
E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining Concepts from Code with Probabilistic Topic Models," *ASE*, 2007

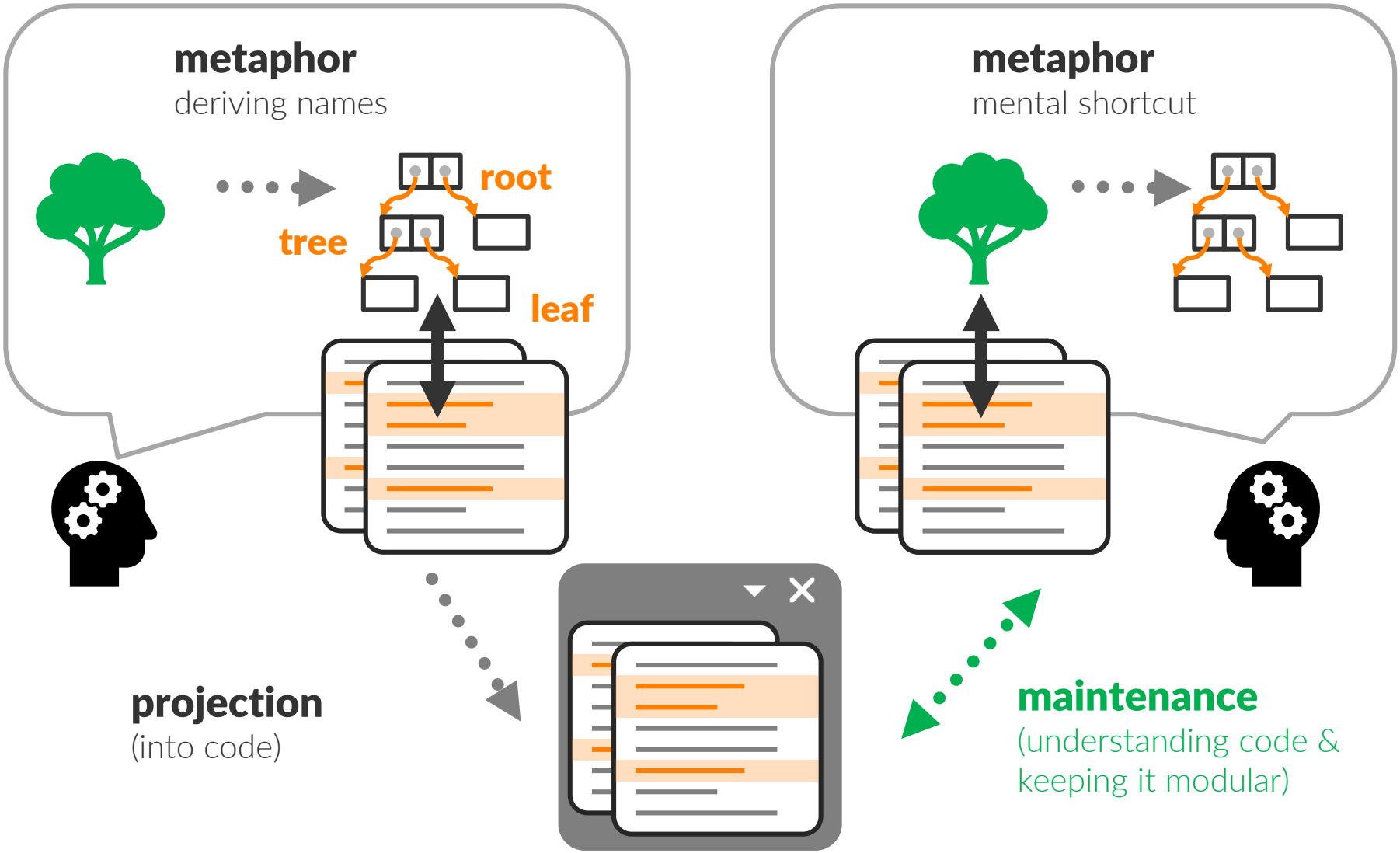
Counteracting Architectural Drift

Architectural Drift:

Conceptual model misaligned with module structure

- Quantifiable using **entropy over time**
- Motivates integration into **version control**
- Hypothesis: Awareness can help programmers to fix modularity issues before incurring **technical debt**


metaphor
deriving names
root
tree
leaf
metaphor
mental shortcut
projection
(into code)
tree
leaf
root
extrapolation
(from code)


metaphor
deriving names
root
tree
leaf
metaphor
mental shortcut
projection
(into code)
maintenance
(understanding code &
keeping it modular)

Open Questions

- Which **additional** information needs can be assessed using our concept model?
- How do our tools need to look like to keep programmers **aware of modularity issues** without distracting them?
- How can we balance the trade-off between **automated** (potentially surprising) and **manual** concept assignment?
- How can the proposed concept model be maintained **collectively**?

Conclusion

1. First-class concepts are complementary to language features to manage concepts

2. Existing tools can be extended to include concept information, new tools can navigate and manipulate concepts

3. Concepts are not restricted to **reverse engineering**, but support modularity during **forward engineering**