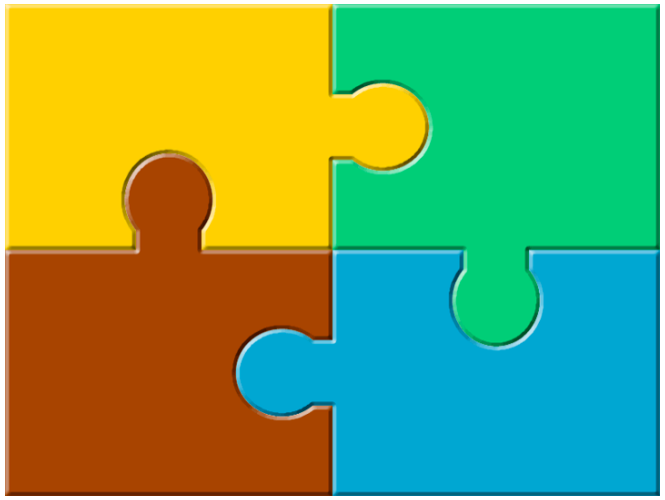


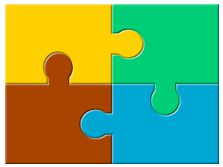
Traits and Aspects

Roel Wuyts
Université Libre de Bruxelles

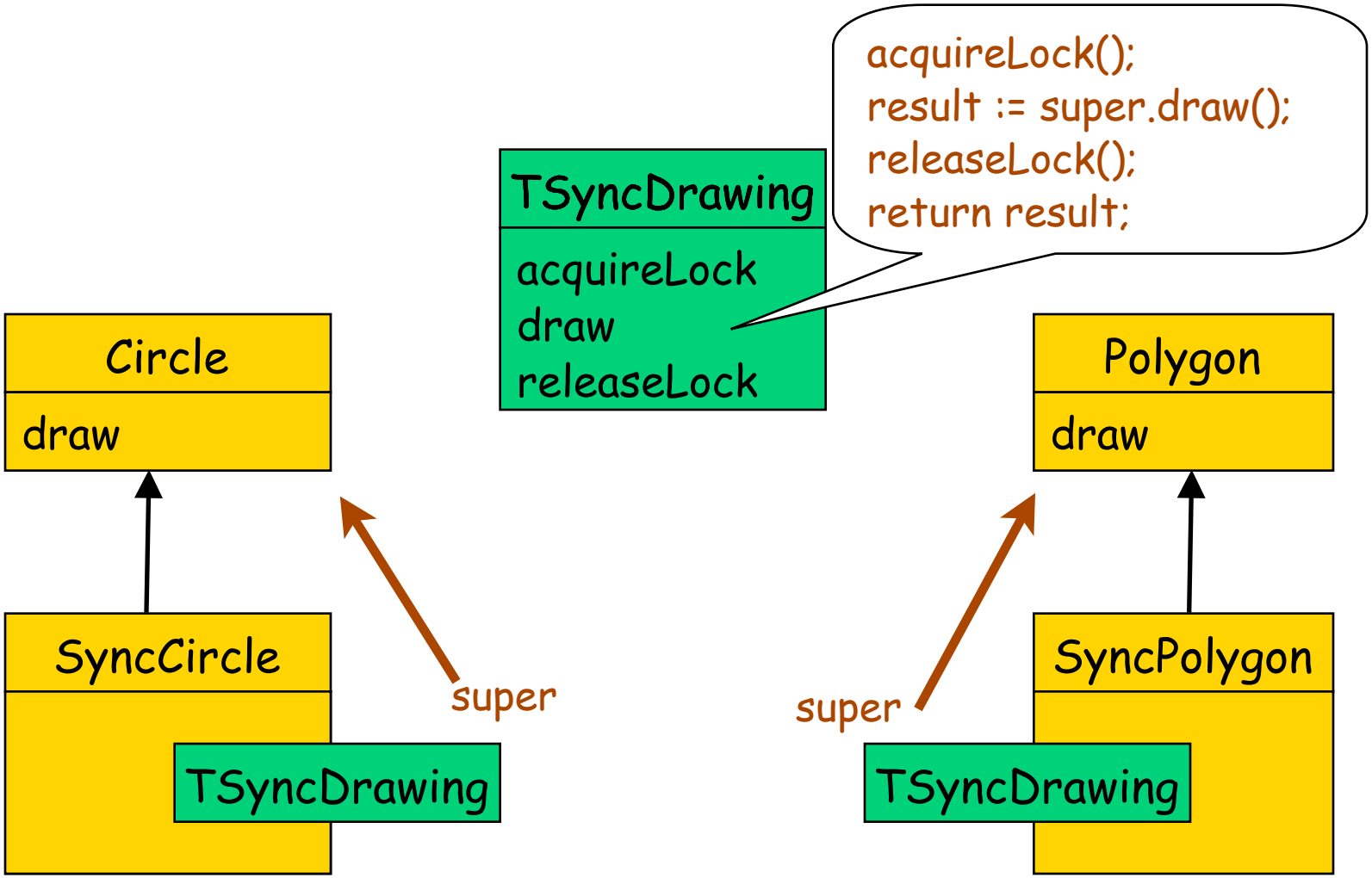
Software Evolution and Aspect-Oriented Programming
03/05/04

Inheritance Problems

- Single: not expressive enough
- Multiple:
  - Complex solution → Hard to understand
  - Explicit class references → Fragile
  - Sometimes requires code duplication
- Mixins:
  - Implicit conflict resolution → Surprises!
  - Composite entity is not in full control
    - → Dispersal of glue code
    - → Fragile hierarchies

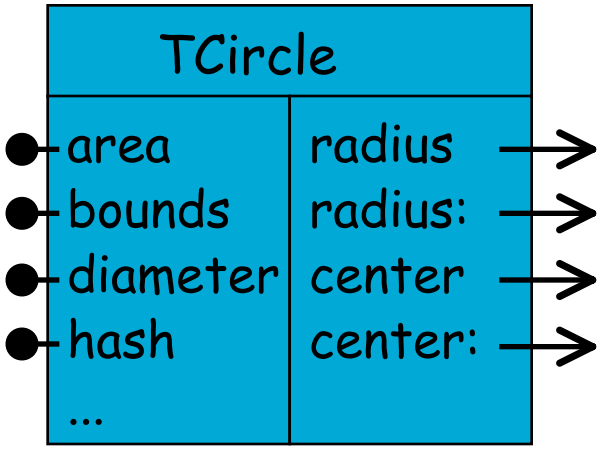


What we want..


acquireLock();
result := super.draw();
releaseLock();
return result;
TSyncDrawing
acquireLock
draw
releaseLock
Circle
draw
Polygon
draw
SyncCircle
TSyncDrawing
super
SyncPolygon
TSyncDrawing
super

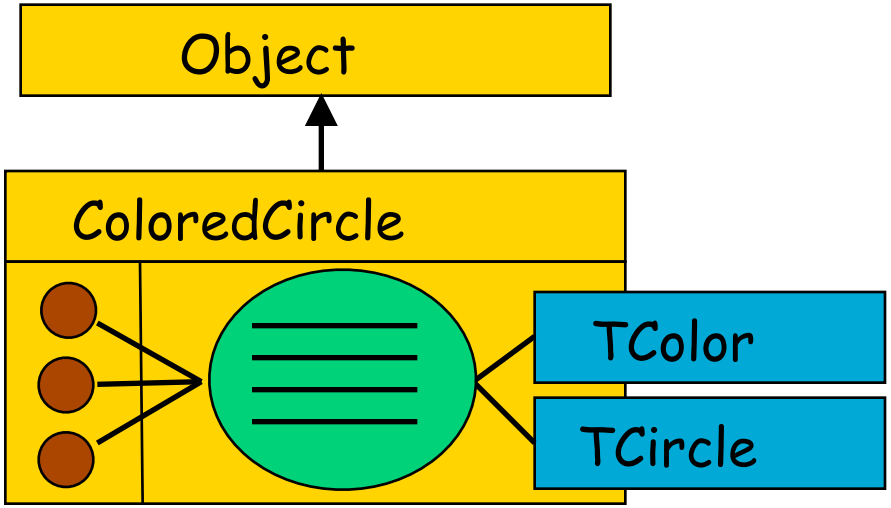
What are Traits?

- Traits are parameterized behaviors
  - Traits provide a set of methods (●—)
  - Traits require a set of methods (→)
  - Traits are purely behavioral (no state)



How are Traits Used?

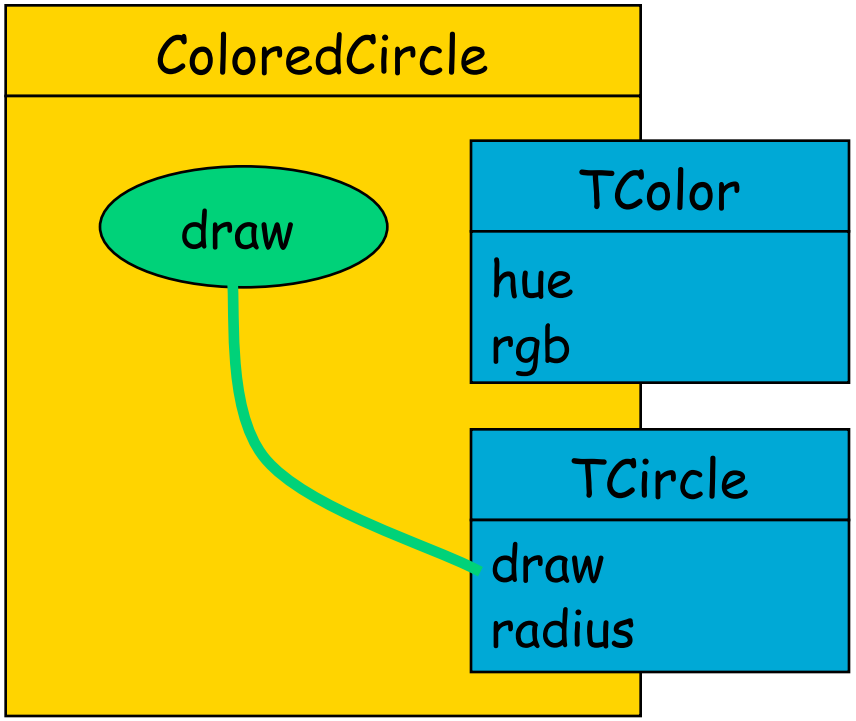
- Traits are the building blocks of classes
  - Class = superclass + state + traits + glue



- Traits do not replace single inheritance
  - They provide modularity *within* classes

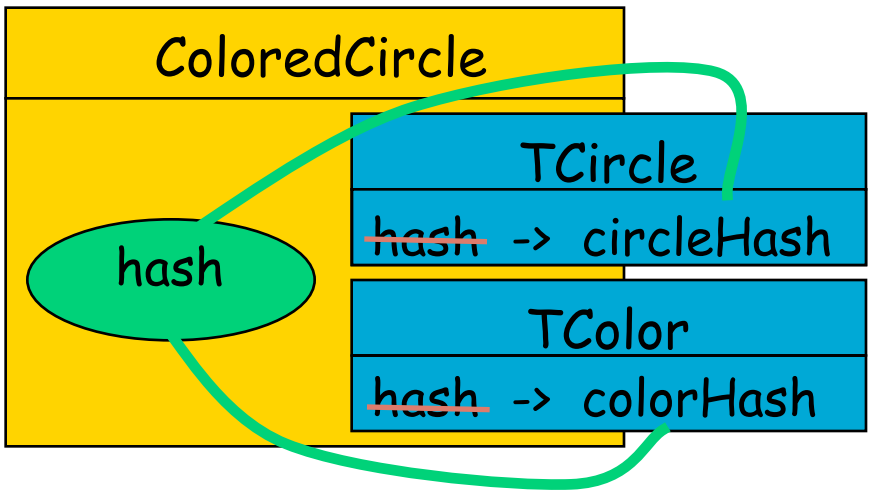
Composition Rules

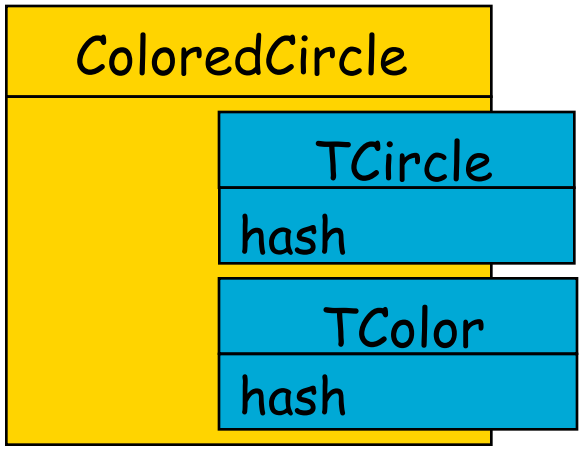
- Class methods take precedence over trait methods



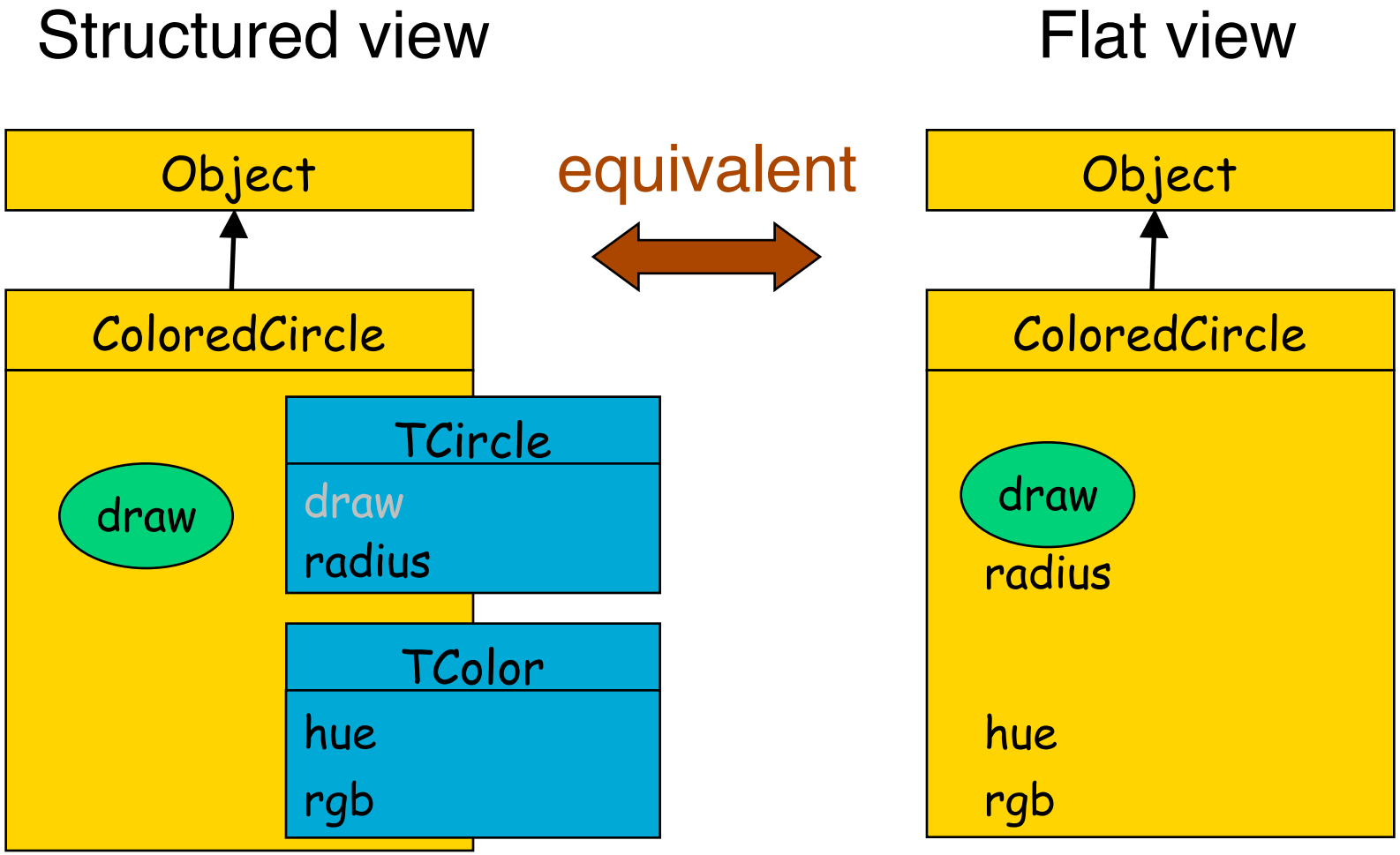
Explicit Conflict Resolution

- Override the conflict with a glue method
  - Aliases to access to the conflicting methods
- Avoid the conflict
  - Exclude conflicting method from one trait





Flattening Property (2 views on code)



Validation

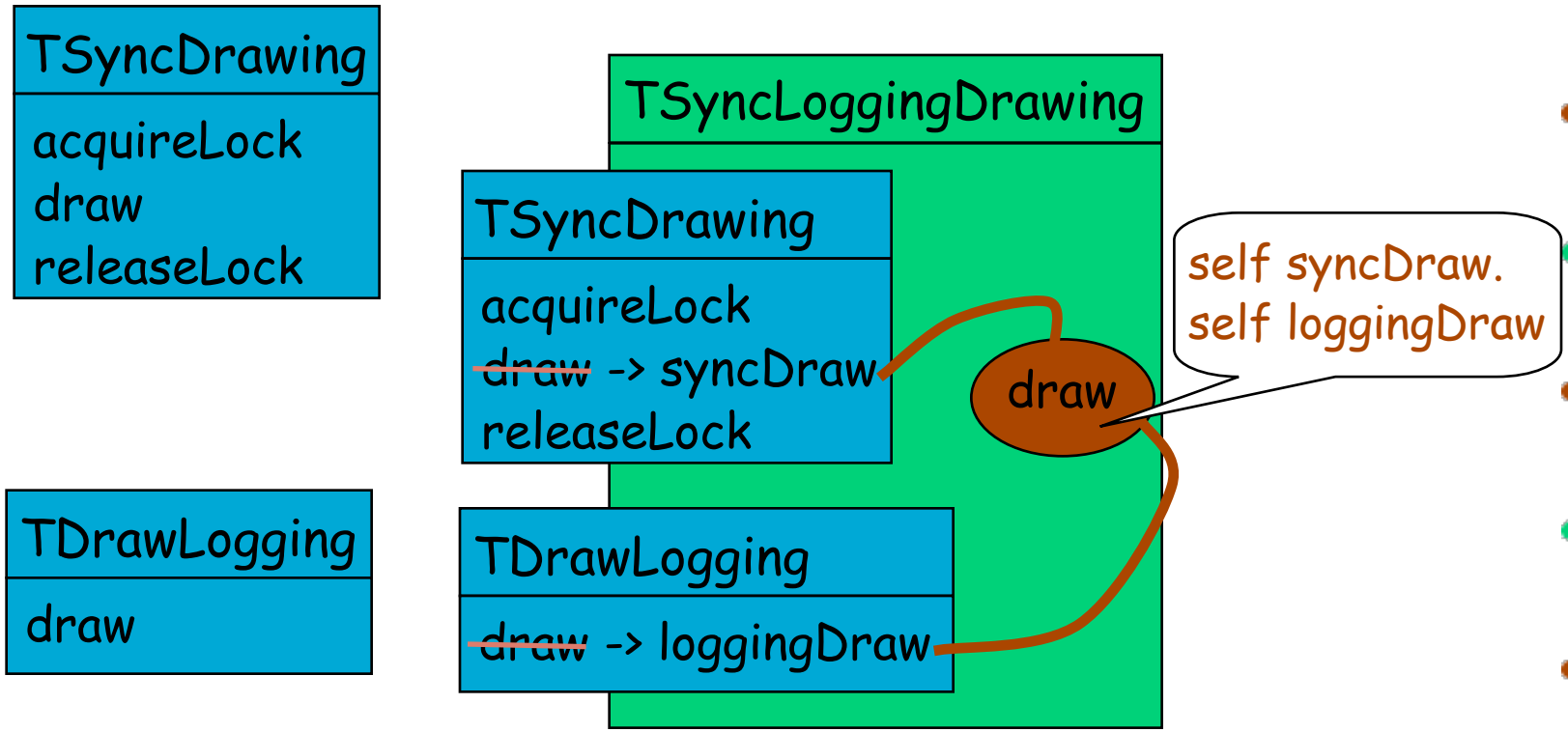
- Implemented in Smalltalk
  - Smart method dictionary manipulation
  - Development environment extended
- Set-theoretic formalization to prove the flattening property
- Applications
  - Refactored the collection hierarchy
  - applied on metaclass composition
- Broad impact

Links with Aspects

- Trait = concern
  - behavior belonging together that can be aplied on classes
  - can require other behavior/concerns to be present (required information)
- Explicit composition
  - + support for cancellation, aliasing, overriding
- Of course: no pointcut language

Traits AOPy example

- make traits for distribution, security, ...
  - and make "composite aspects"!

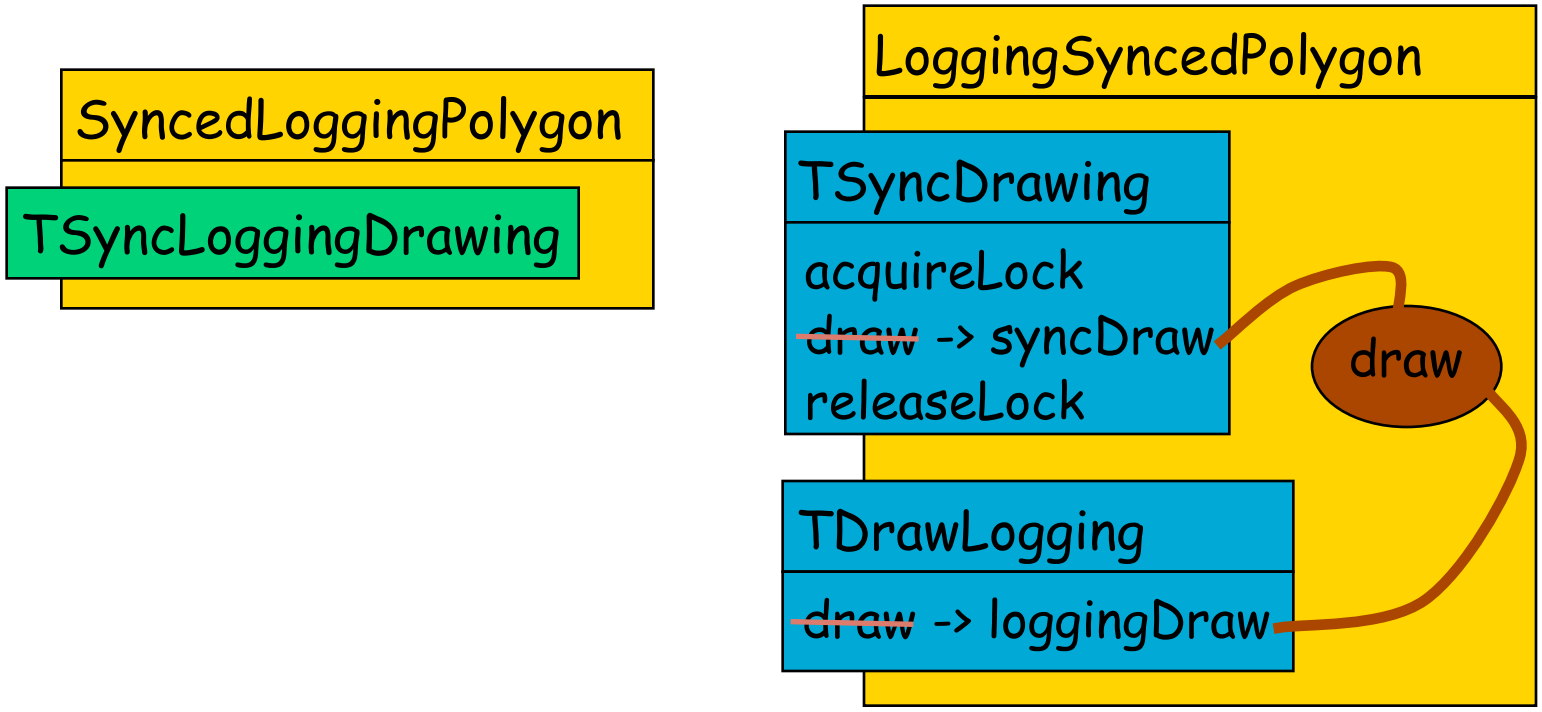


Overlapping advices

- What happens when advices from different aspects target the same join point?
  - AspectJ: declare preferences
    - declare precedence: *..*Security*,Logging+,*;
  - *implicit* conflict resolution
- Rule-based approaches
  - explicit composition (composition rules)
  - partial conflict detection (constraints)

Overlapping Advices with Traits

- With Traits: automatic conflict detection + explicit conflict resolution (not only ordering conflicts can be solved!)



Conclusion

- Traits: composition of methods in classes
  - Or to compose new traits
- Automatic conflict detection
- Explicit (manual) conflict resolution
  - overriding, cancellation, aliasing

More Information...

- Andrew P. Black, Nathanael Schärli, Stéphane Ducasse, *Applying Traits to the Smalltalk Collection Hierarchy*, Proceedings of OOPSLA 2003, Springer-Verlag, pp. 47--64, 2003.
- Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, Andrew P. Black, *Traits: Composable Units of Behavior*, Proceedings of ECOOP 2003
- Stephane Ducasse, Nathanael Schaerli, Oscar Nierstrasz, Roel Wuyts and Andrew Black, *Traits: A Mechanism for fine-grained Reuse*, Submittet to TOPLAS.