

# A taste of **trygve**

Jim Coplien  
Former C++ Programmer & Author

**Gertrud & Cope**  
Helsingør, Danmark  
jcoplien@gmail



# Pointers...

- DCI documentation & downloads:
  - <http://fulloo.info>
- **trygve** on GitHub:
  - <https://github.com/jcoplien/trygve>
- Upcoming 2-day tutorial in Frankfurt (& others):
  - <https://sites.google.com/a/gertrudandcope.com/www/training/ddd-with-dci>



My greatest contribution to computing is that I  
never invented a programming language.

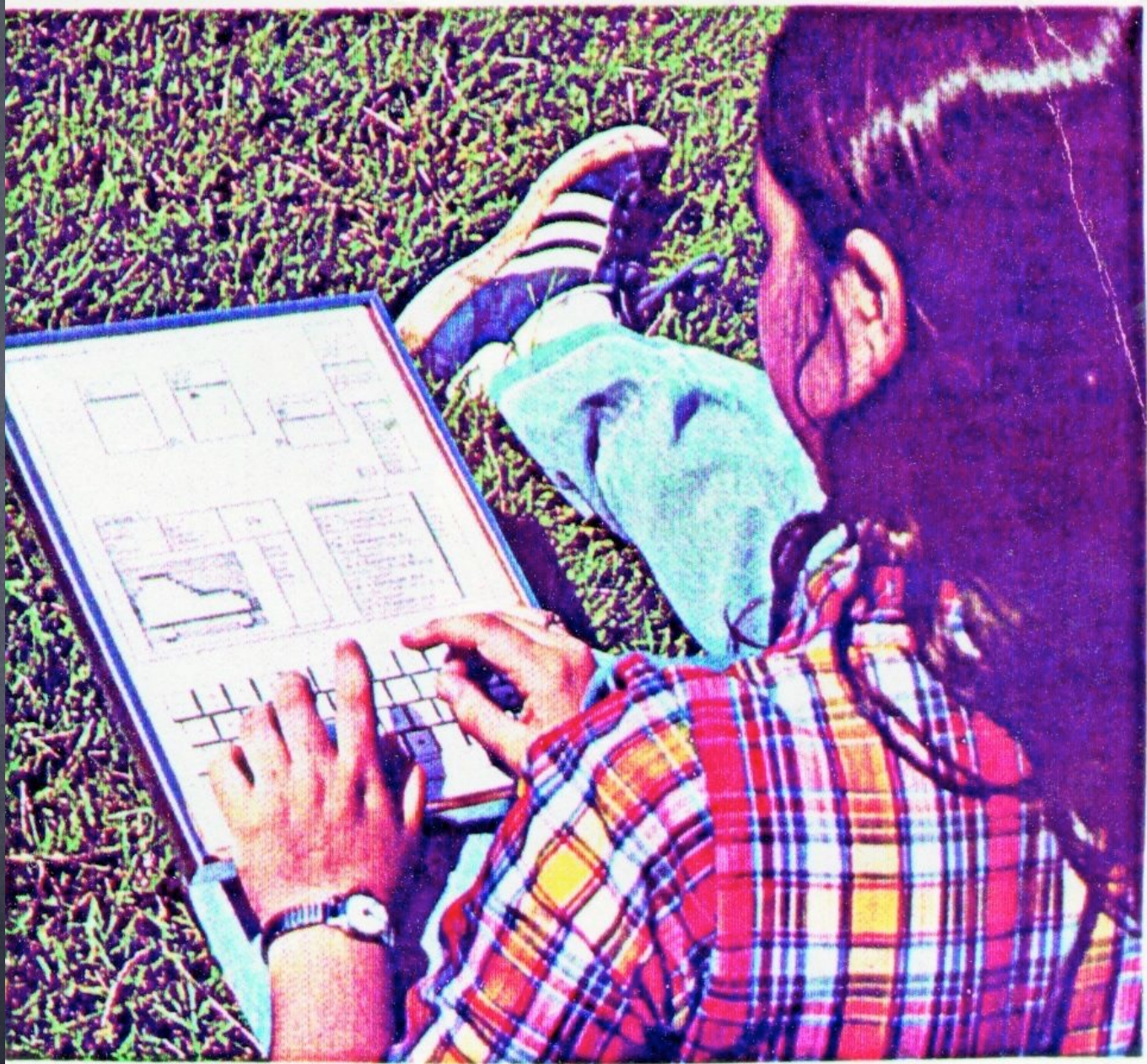
– Jerry Weinberg



# In 1972, Kay coined the term: “Object-Oriented Programming”

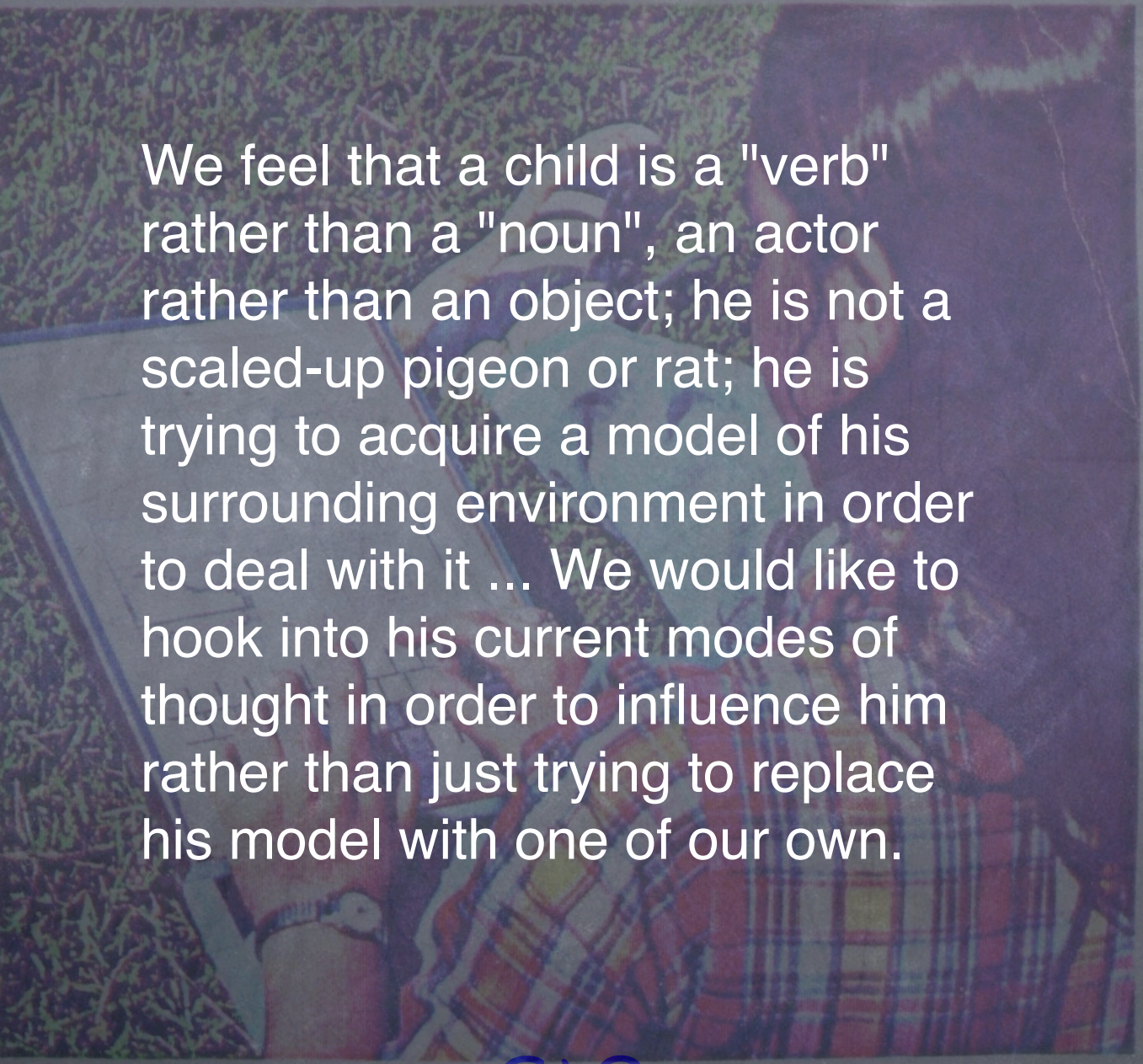
- In his 1972 paper the word “class” doesn’t appear once
- Objects: operational models, in the machine, to extend the capabilities of the human mind
- Classes came into Smalltalk ca. 1976 (from Simula 67)
- **trygve**: conceived to address the largest gaps between current OOP and the benefits of the original vision, through DCI





GFC

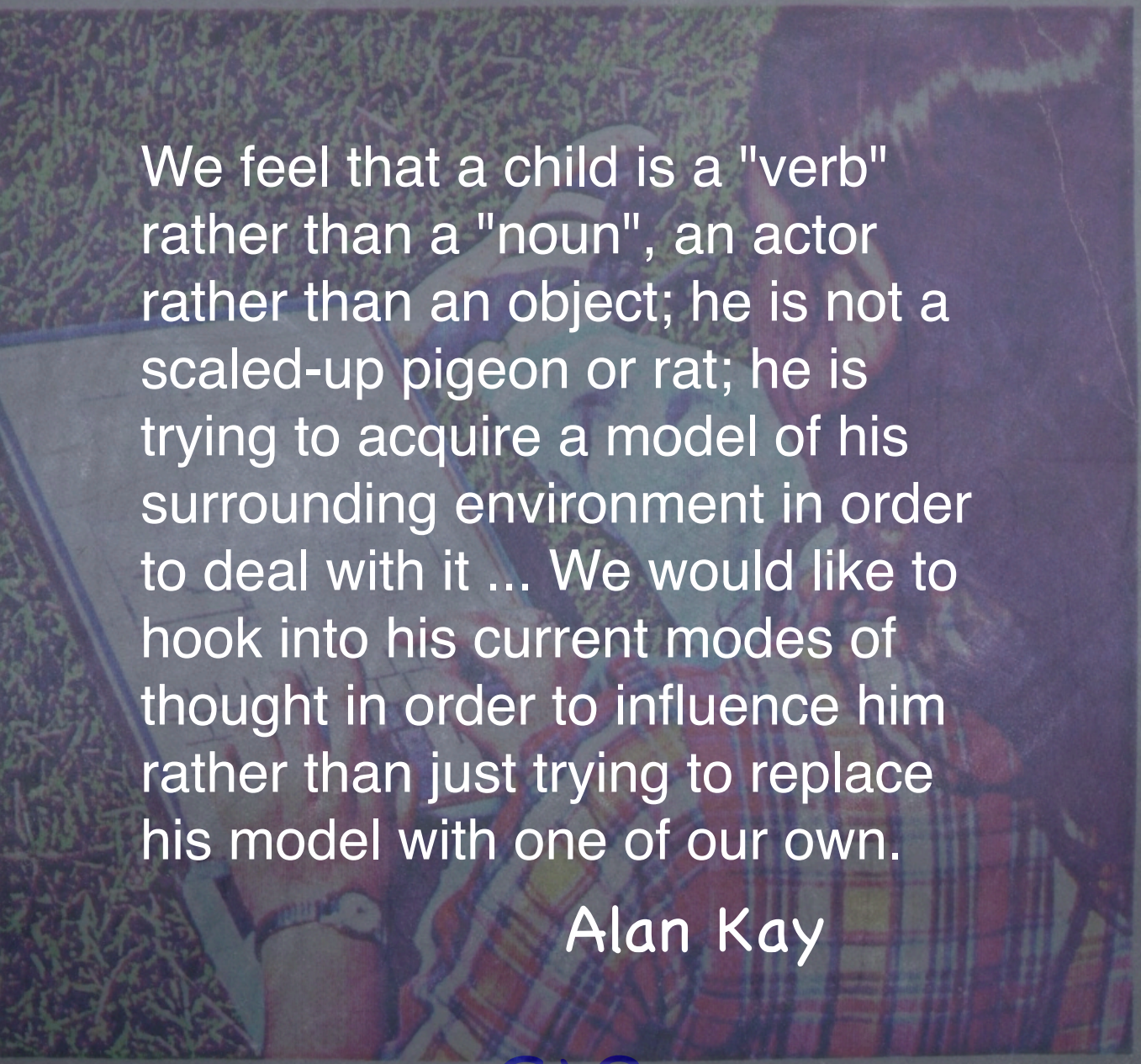




We feel that a child is a "verb" rather than a "noun", an actor rather than an object; he is not a scaled-up pigeon or rat; he is trying to acquire a model of his surrounding environment in order to deal with it ... We would like to hook into his current modes of thought in order to influence him rather than just trying to replace his model with one of our own.

G+C





We feel that a child is a "verb" rather than a "noun", an actor rather than an object; he is not a scaled-up pigeon or rat; he is trying to acquire a model of his surrounding environment in order to deal with it ... We would like to hook into his current modes of thought in order to influence him rather than just trying to replace his model with one of our own.

Alan Kay

G&C





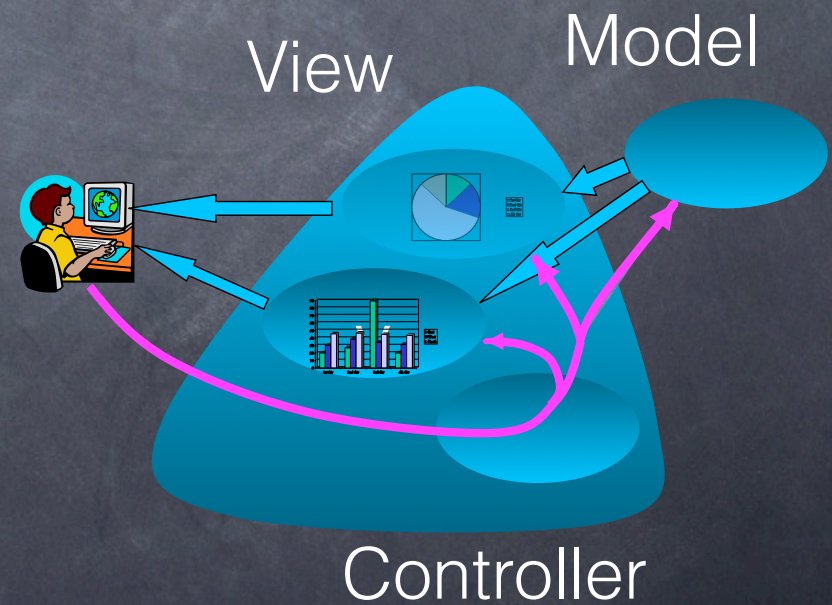
G+C







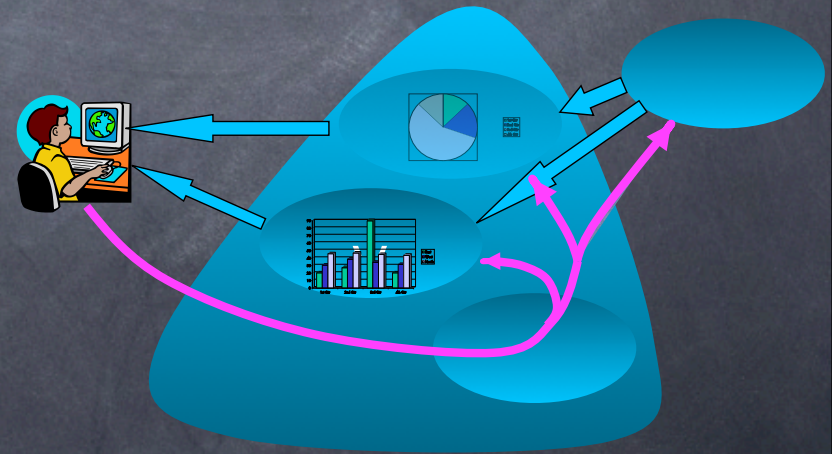
# Action Between Objects



GFC



# Action Between Objects



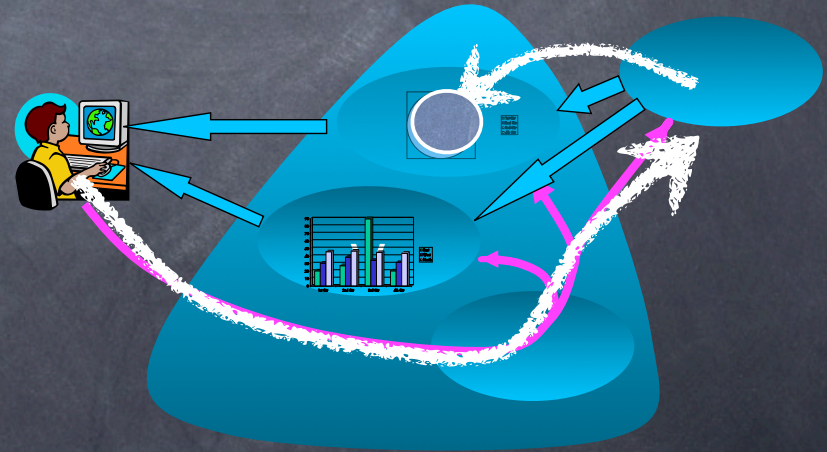
G+C



# Action Between Objects



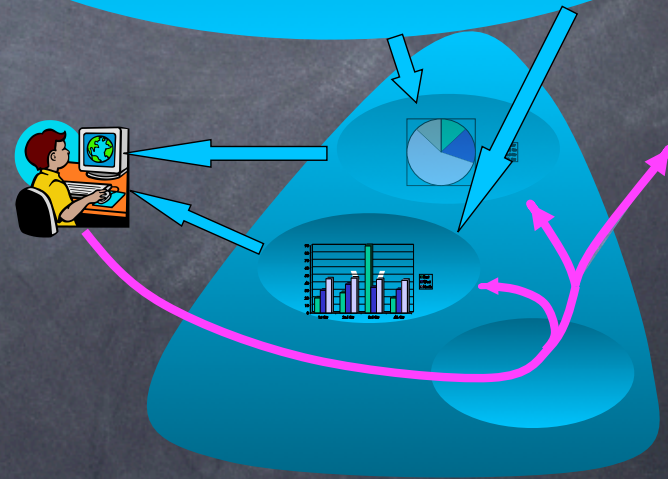
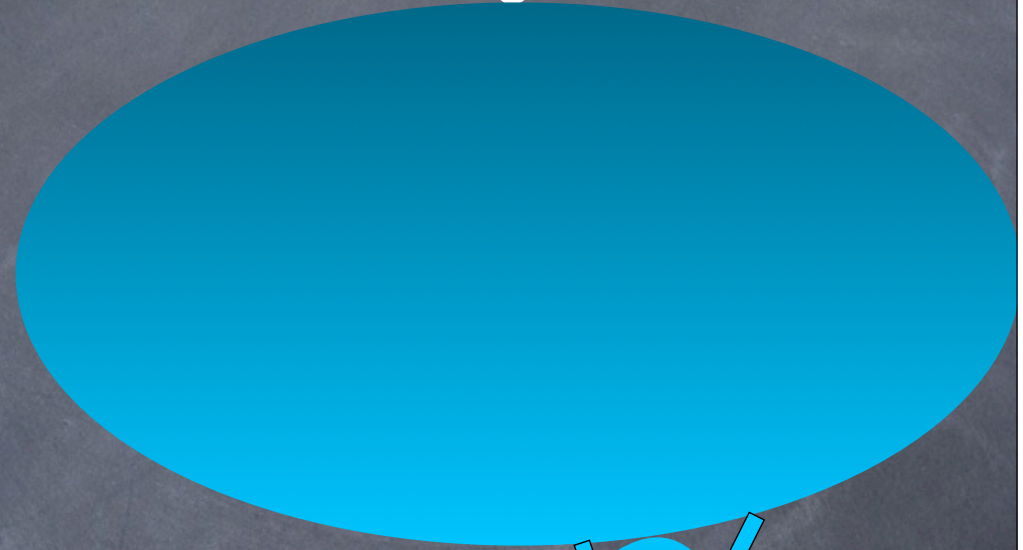
The programmer  
must consider the  
**system:**  
action **between**  
objects



G&C



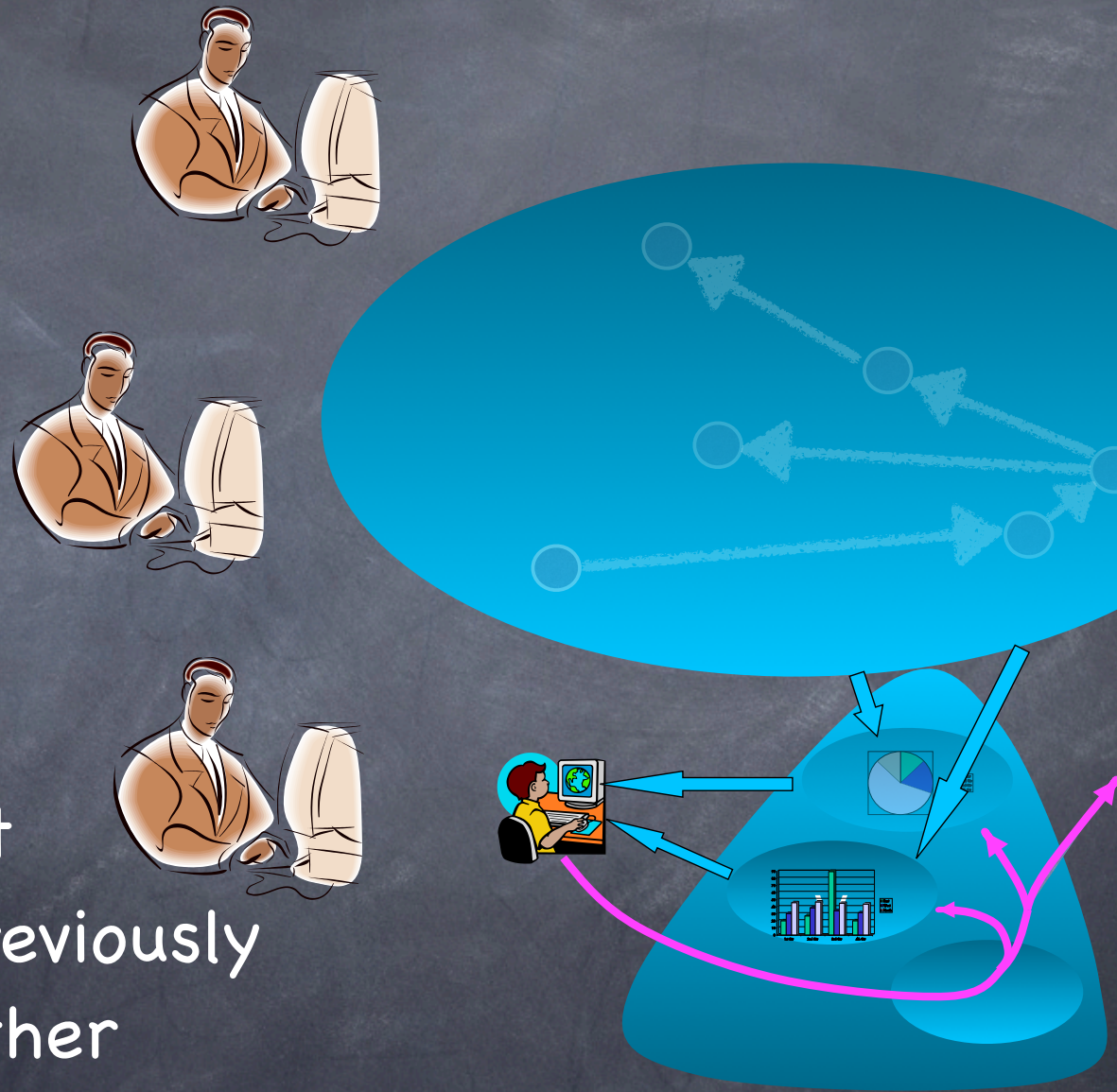
# Action Between Objects



GFC



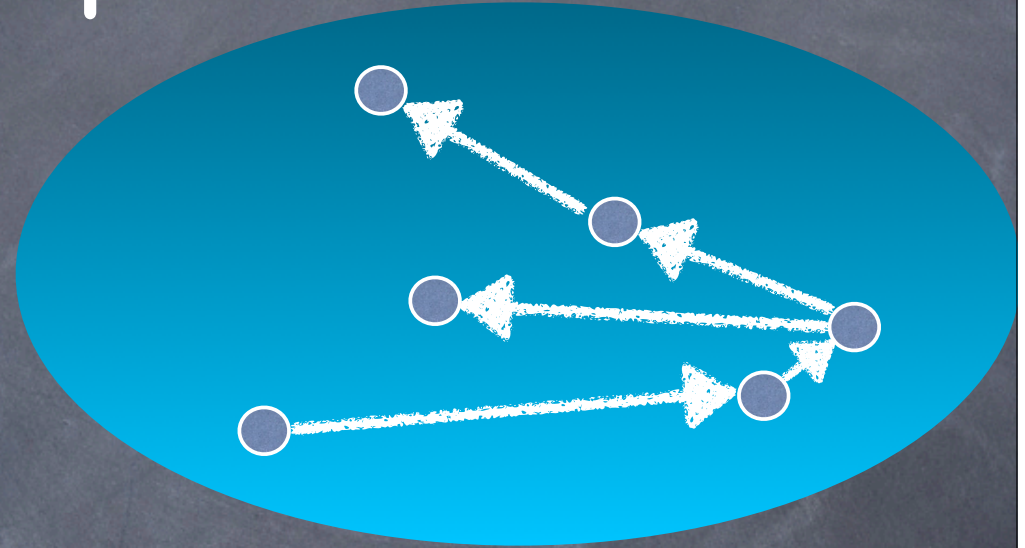
Programmers' objects interact with objects previously conceived by other programmers



G&C



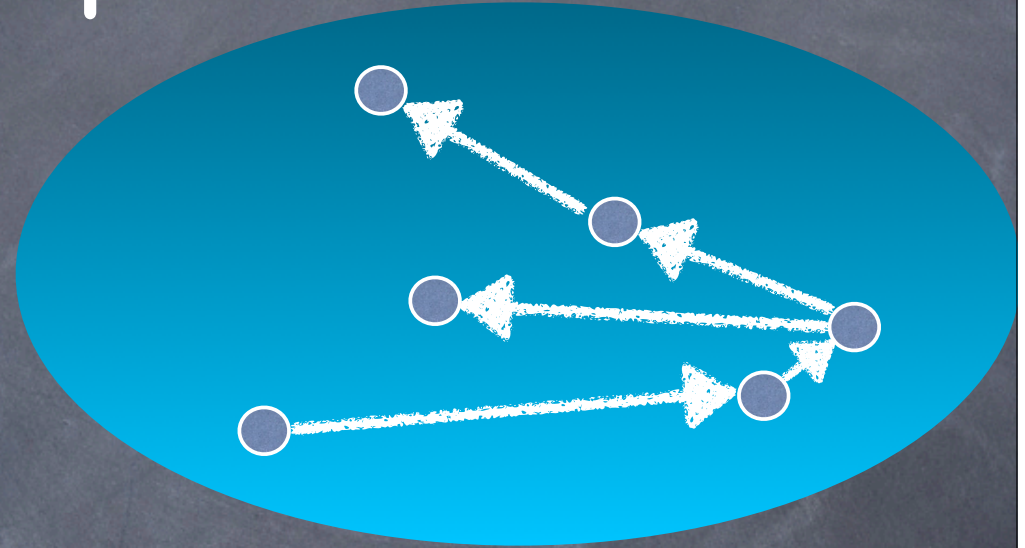
# System Operations



GFC



# System Operations

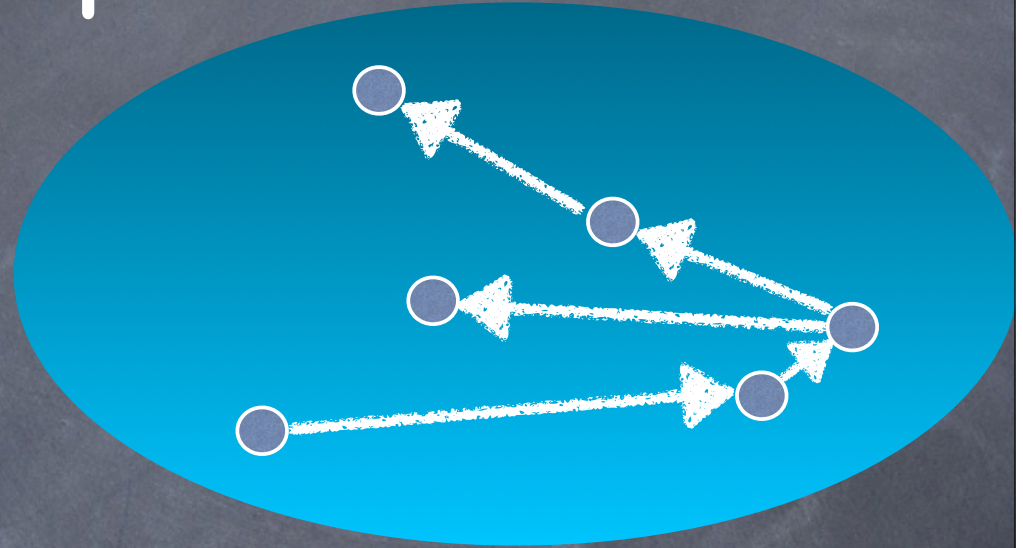


While the specific interactions are emergent, the **form** of the interactions is **designed**.

G&C



# System Operations

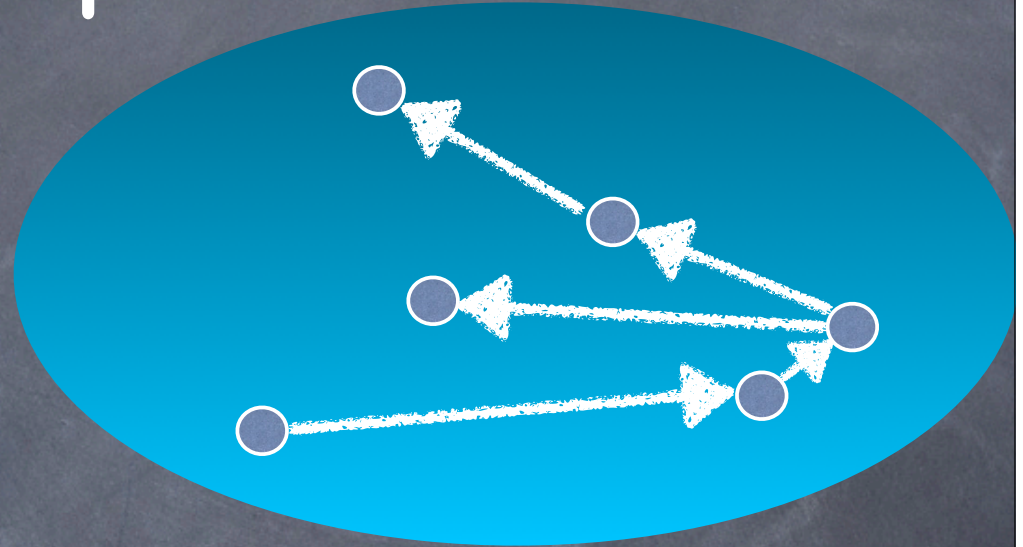


This form lives  
within no single  
object or class.

GFC



# System Operations

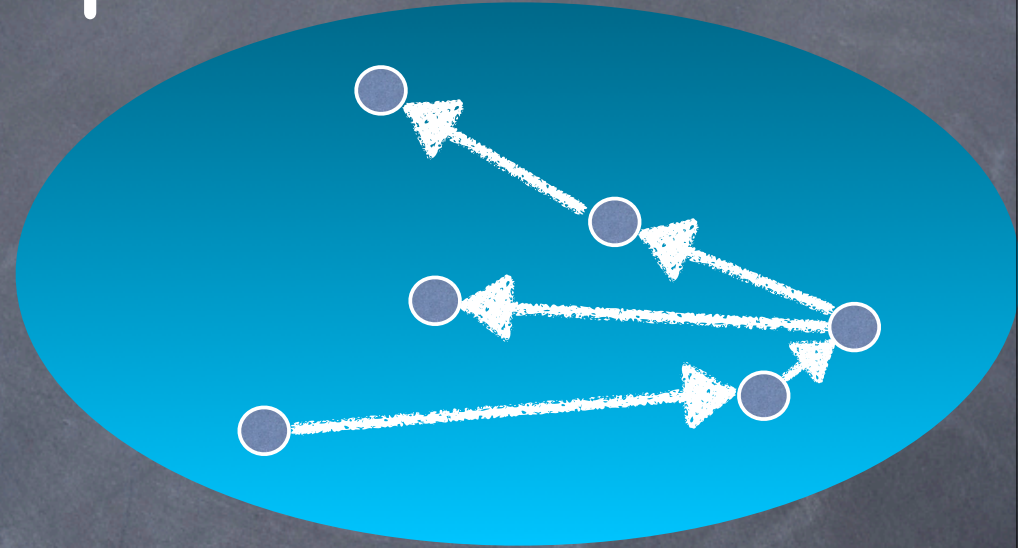


Objects are an overly  
small concept,  
conceived for the  
revenge of the nerds,  
each owning their  
individual classes

GFC



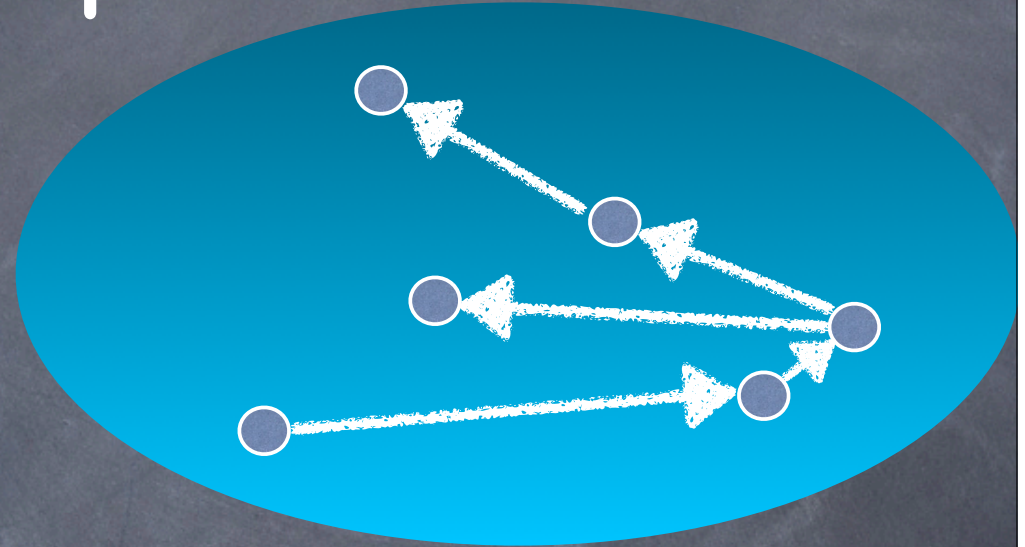
# System Operations



GFC



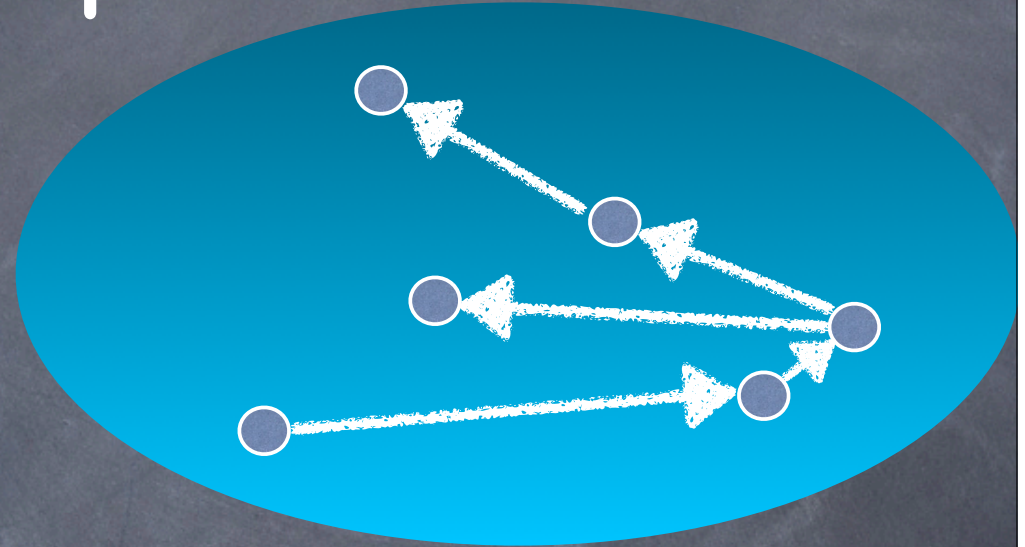
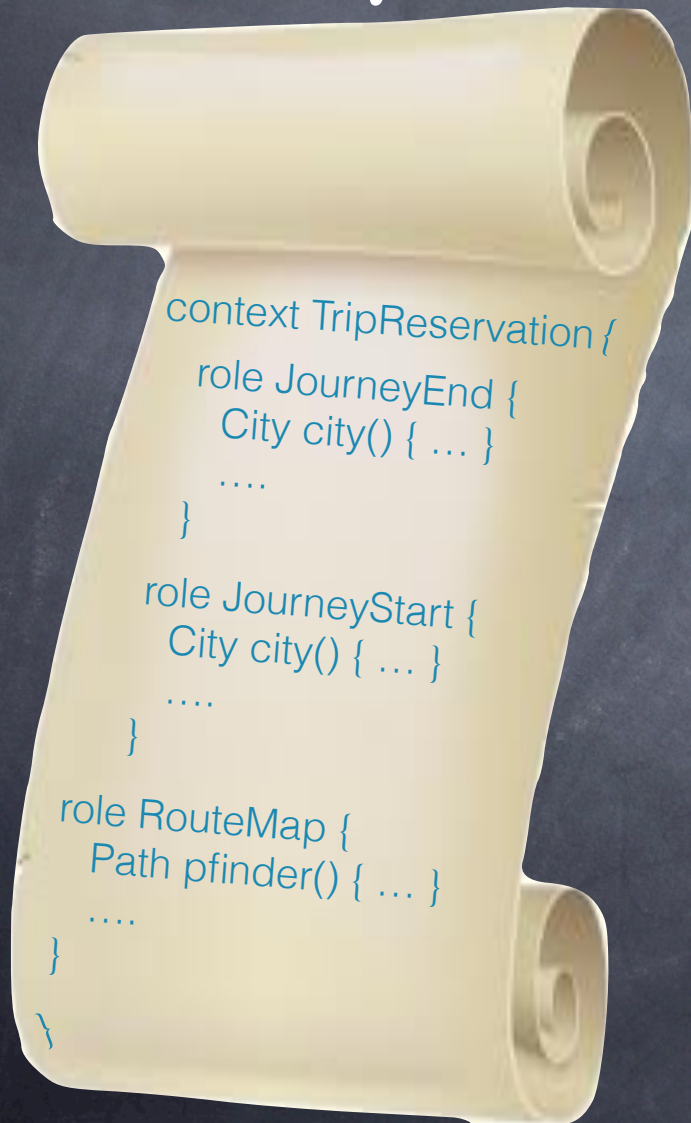
# System Operations



GFC



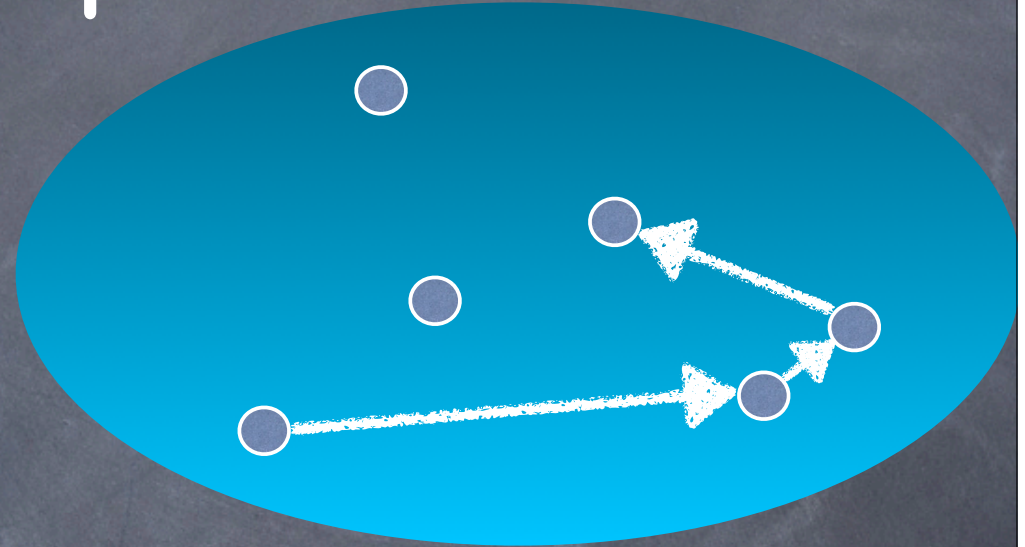
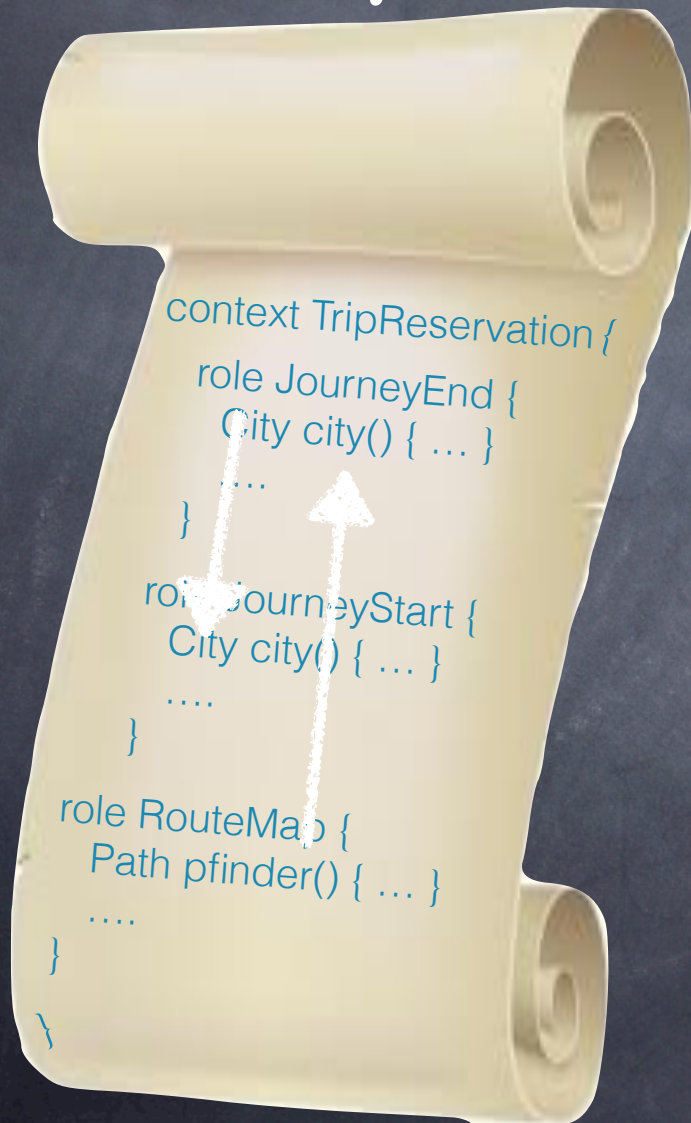
# System Operations



GFC



# System Operations



G+C



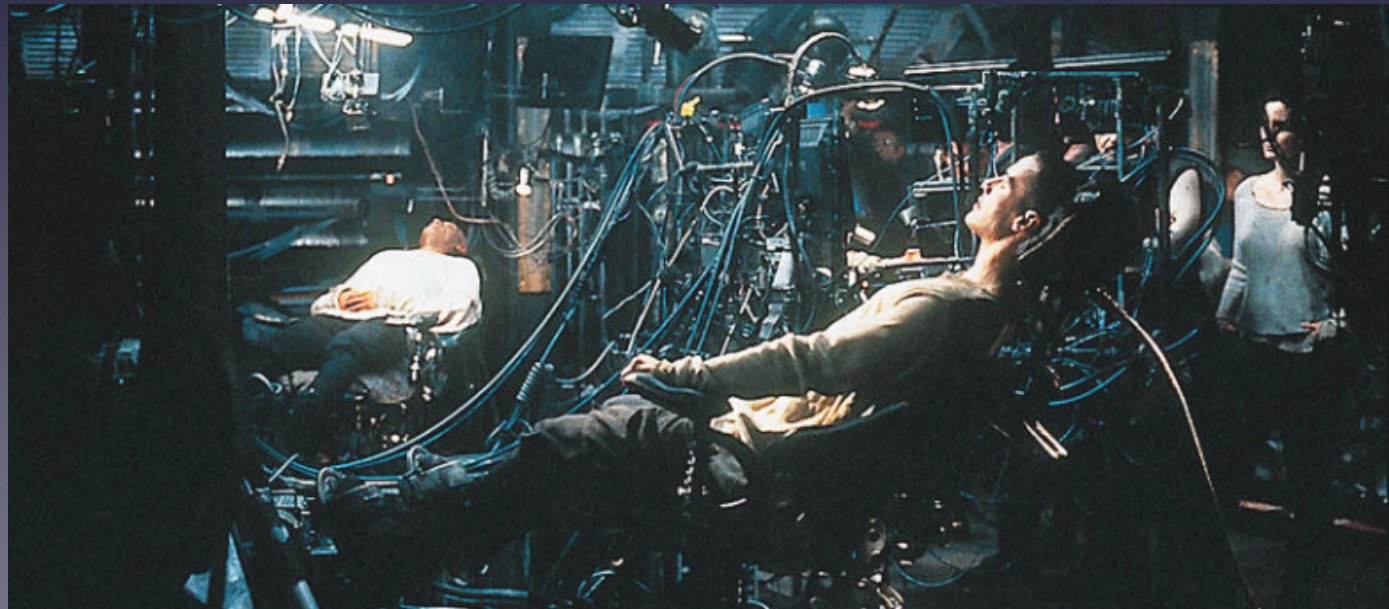
# Teaching Actors their Scripts

```
context TripReservation {  
  role JourneyStart { ... }  
  role JourneyEnd { ... }  
  public TripReservation(Object jStart, Object jEnd){  
    JourneyStart = jStart;  
    JourneyEnd = jEnd  
  }  
}
```



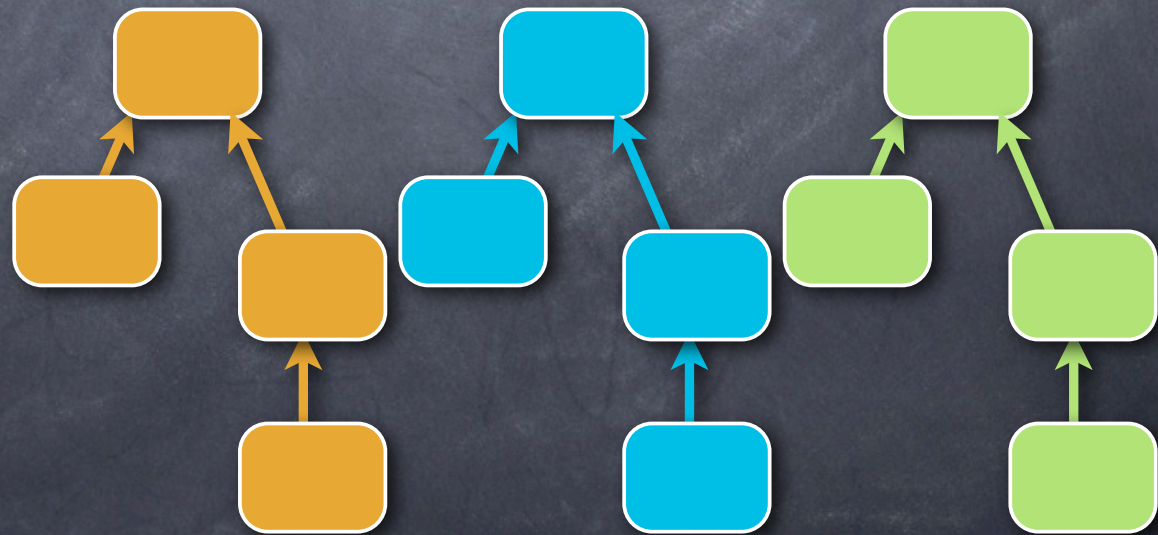
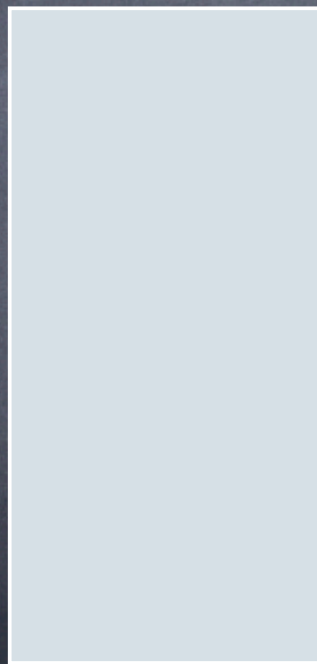
# Teaching Actors their Scripts

```
context TripReservation {  
  role JourneyStart { ... }  
  role JourneyEnd { ... }  
  public TripReservation(Object jStart, Object jEnd){  
    JourneyStart = jStart;  
    JourneyEnd = jEnd  
  }  
}
```



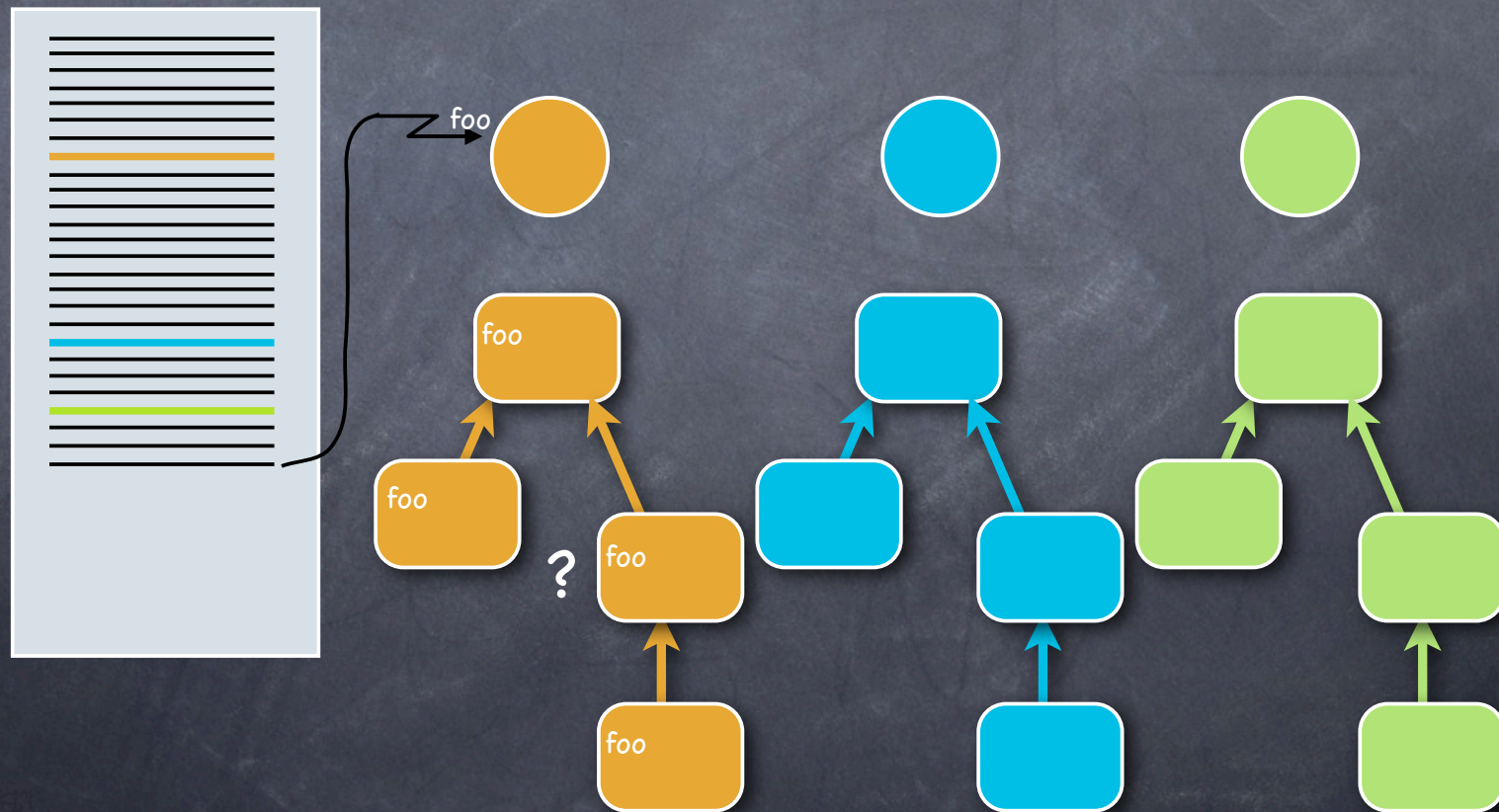


# Contextualized Polymorphism



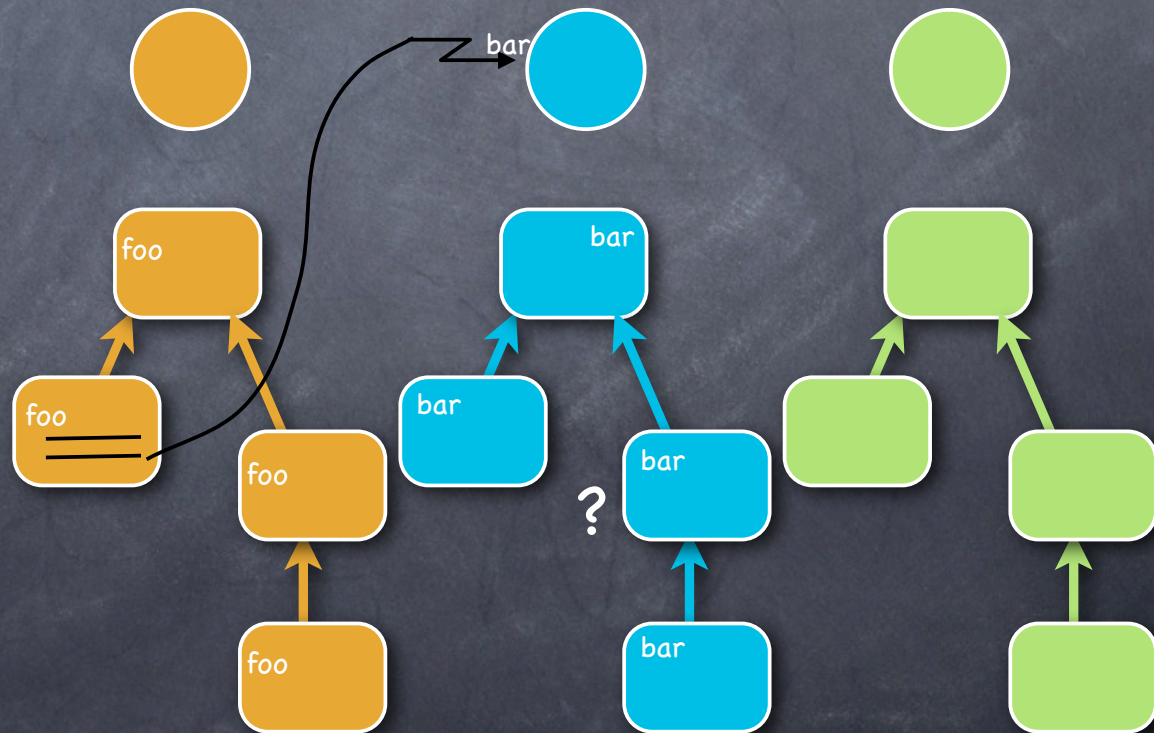


# Contextualized Polymorphism



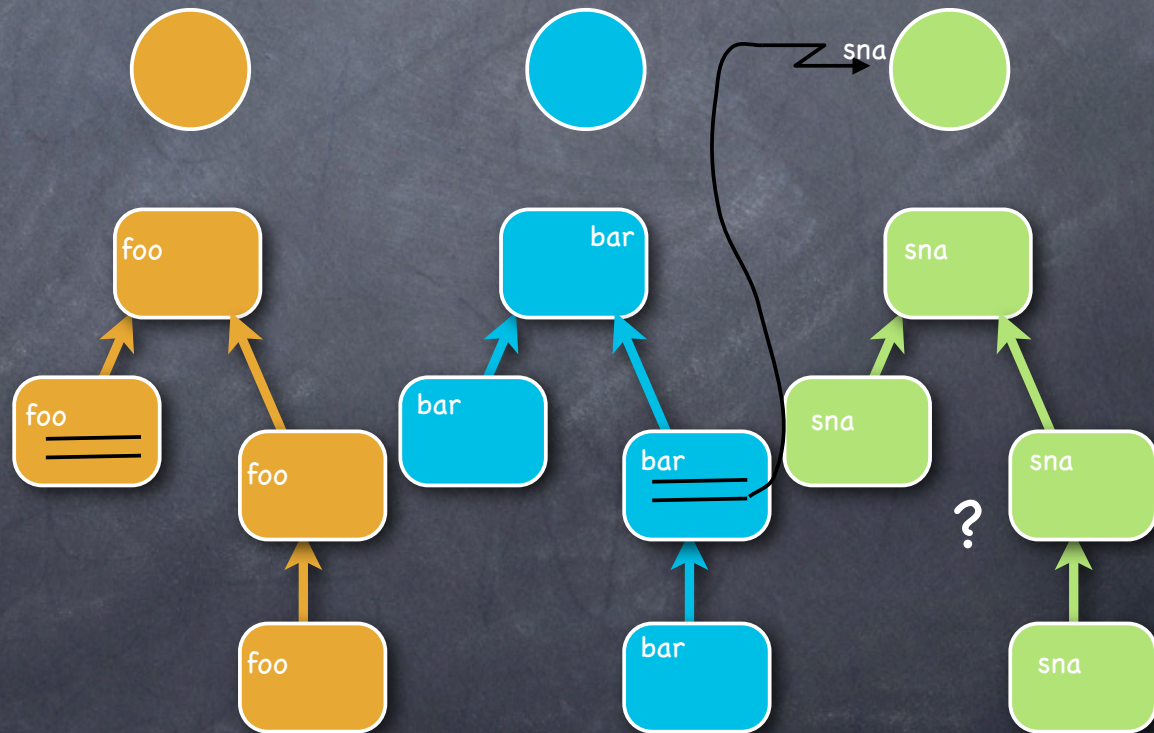
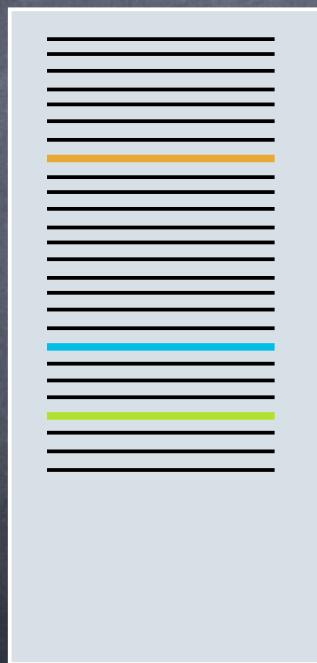


# Contextualized Polymorphism



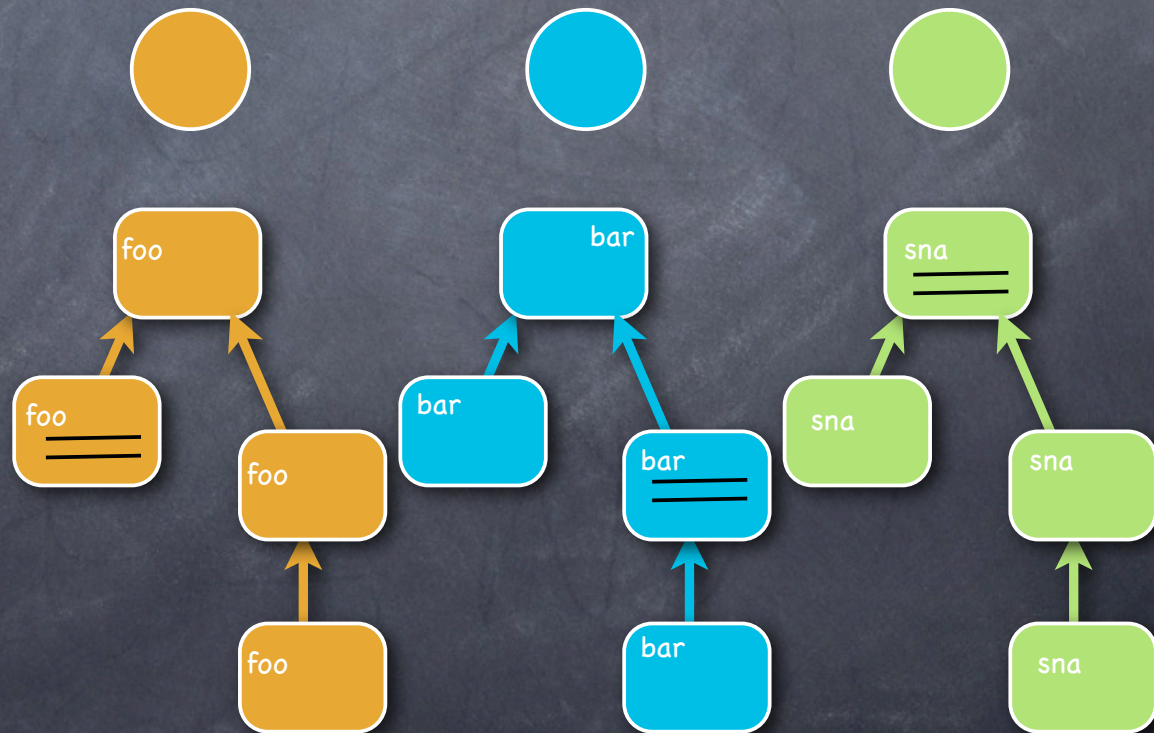
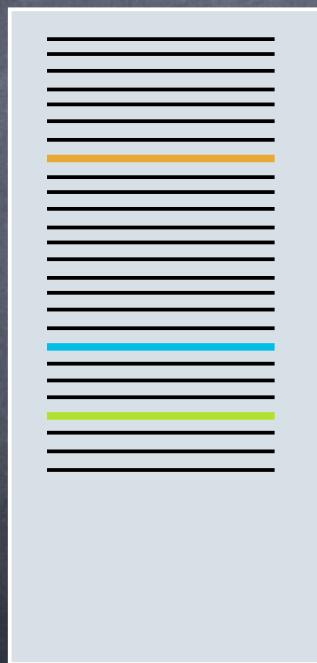


# Contextualized Polymorphism





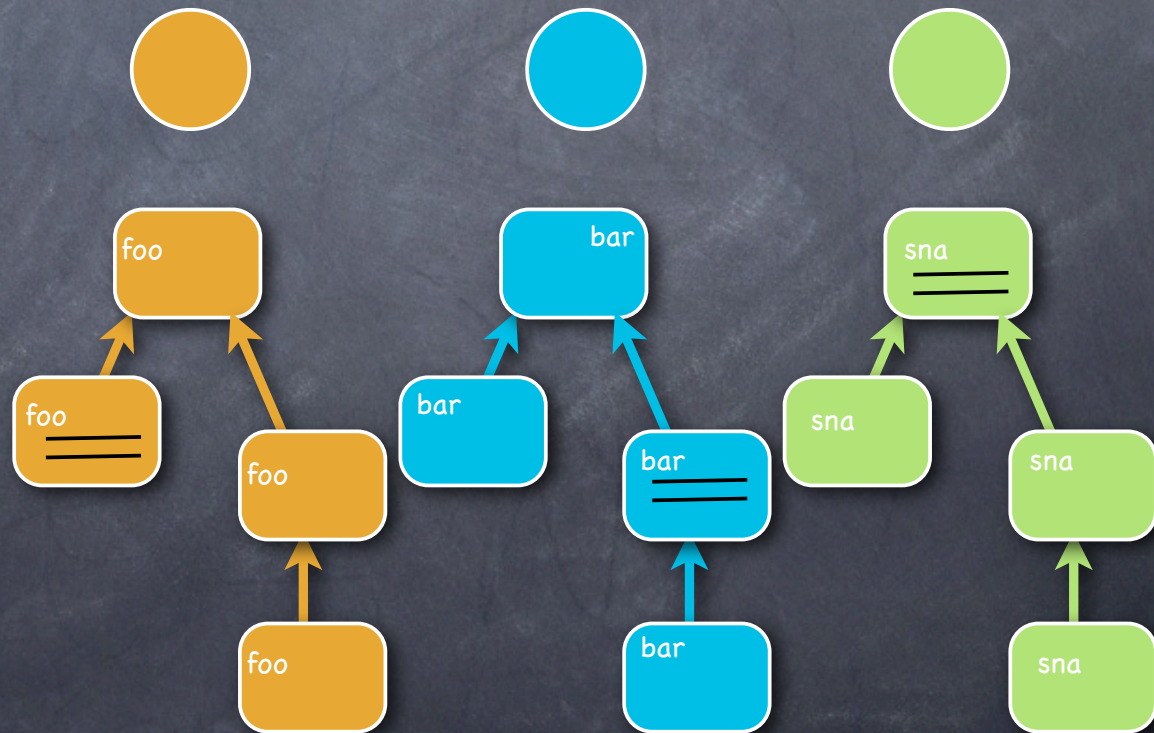
# Contextualized Polymorphism





“Just trust the objects to do the right thing and everything will be fine.” — Smalltalk

# Contextualized Polymorphism

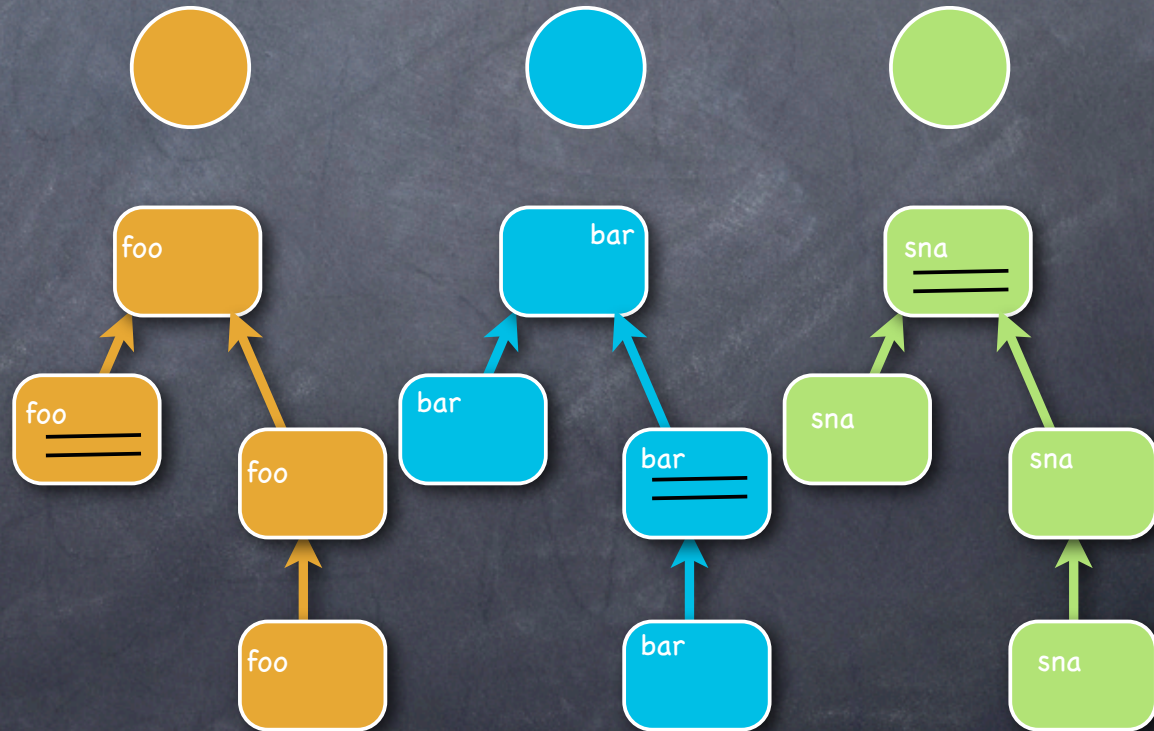




“You believe in things you don't understand, you may suffer.”

– Stevie Wonder

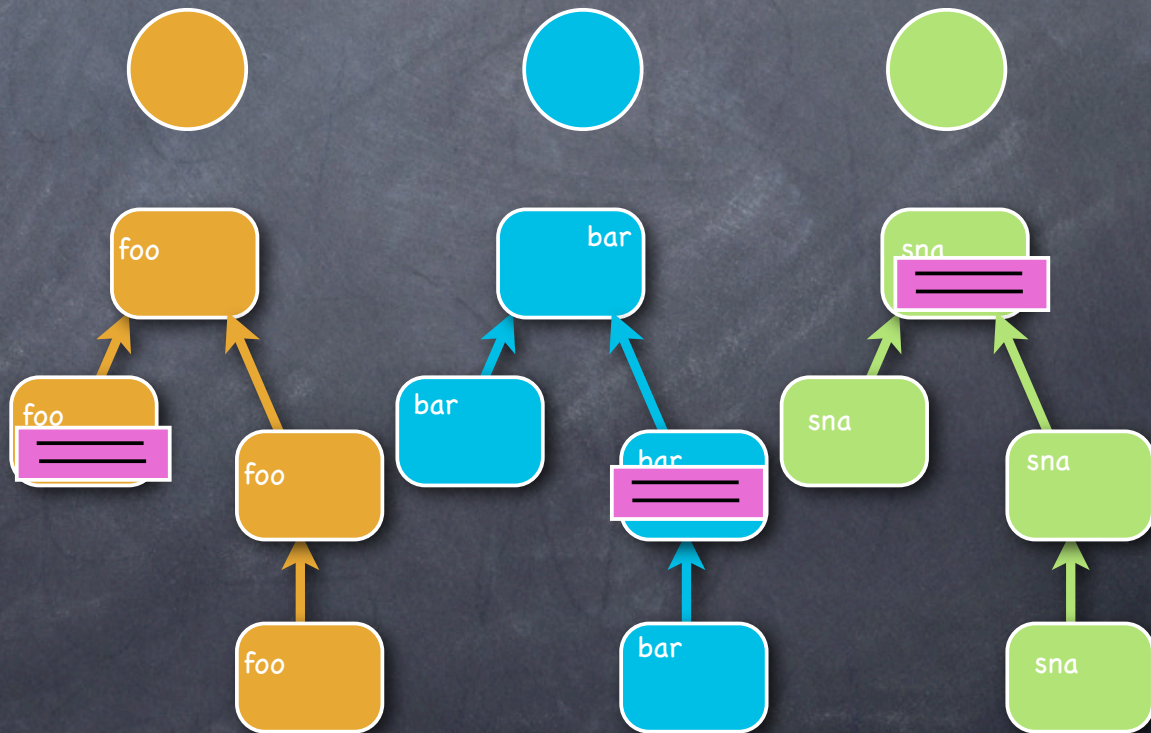
# Contextualized Polymorphism





# Contextualized Polymorphism

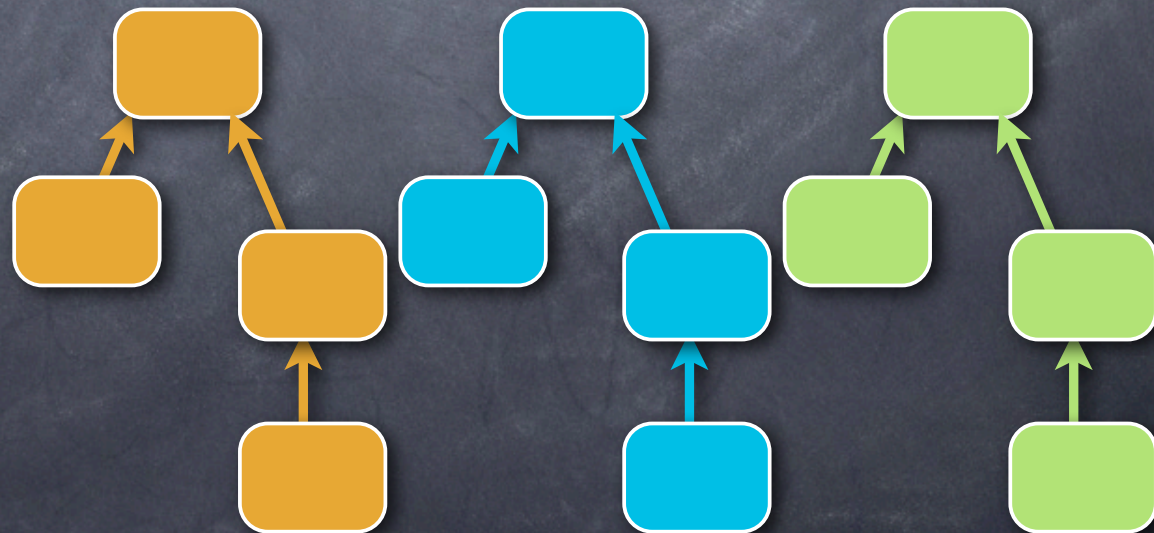
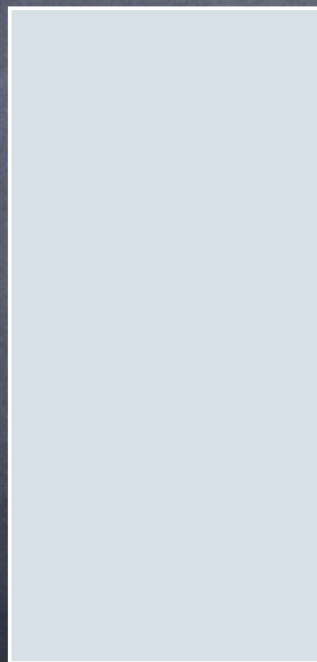
Where is the use case?





# Contextualized Polymorphism

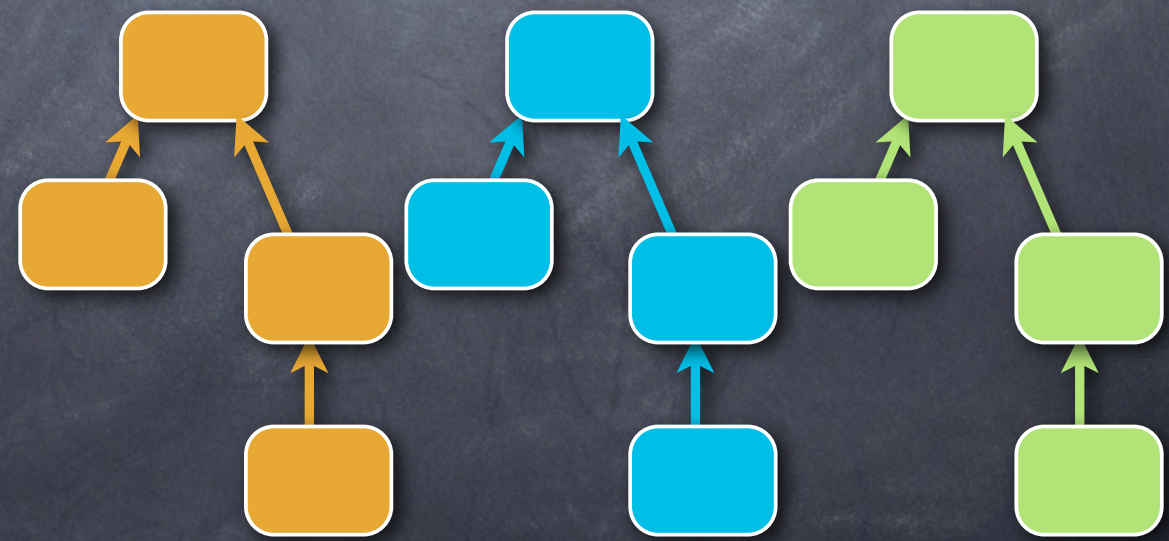
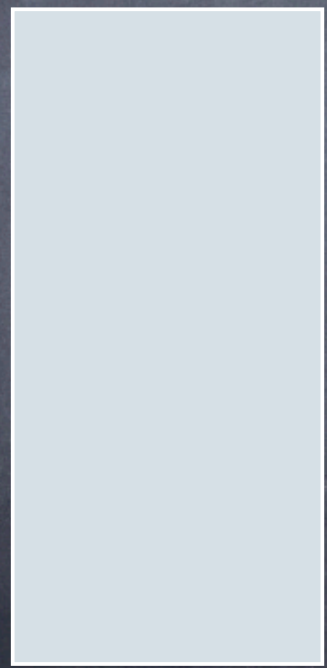
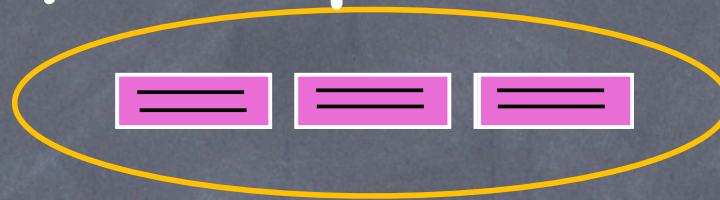
**Roles:** A new  
concept





# Contextualized Polymorphism

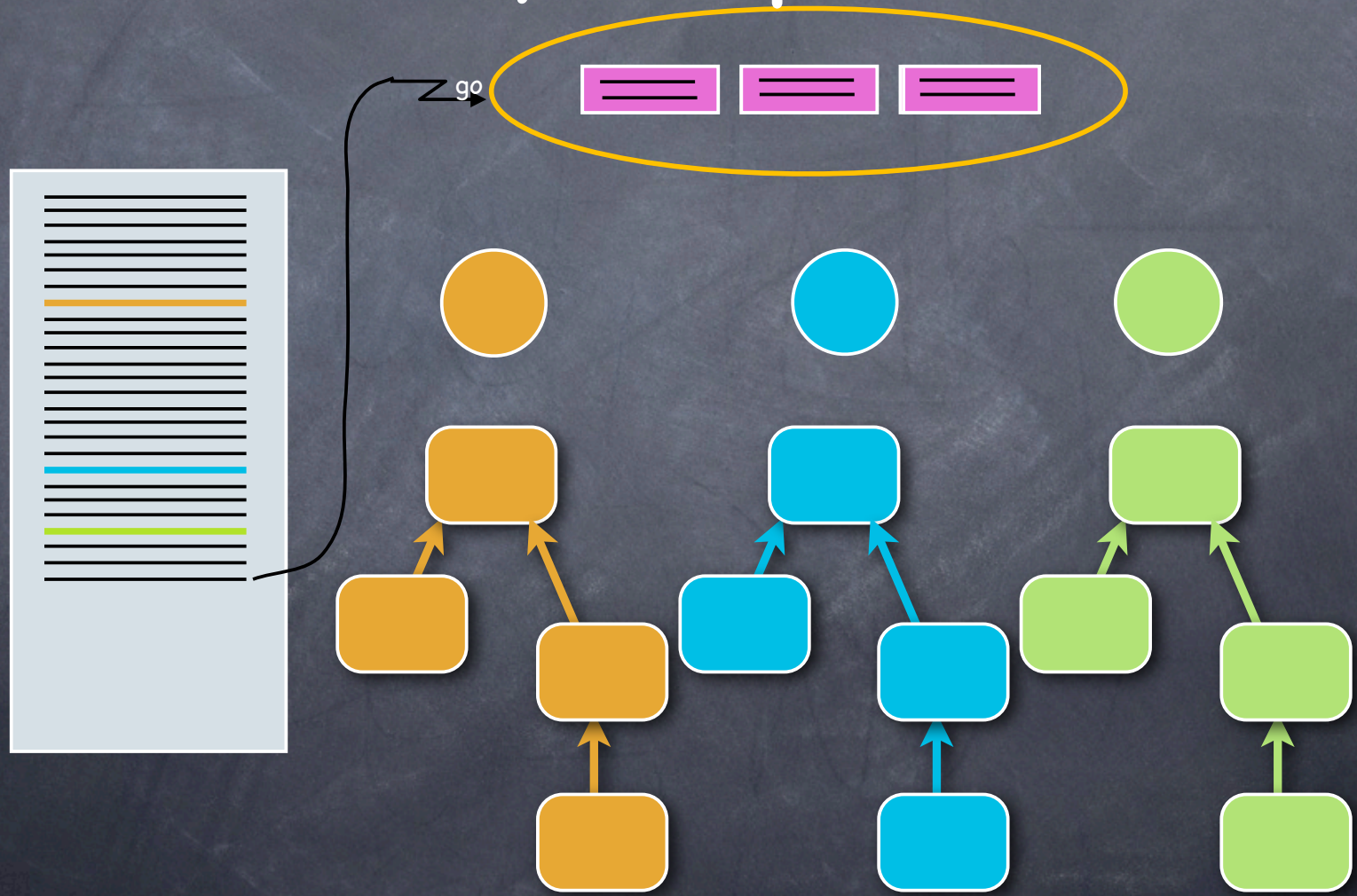
Context:  
Another  
new  
concept





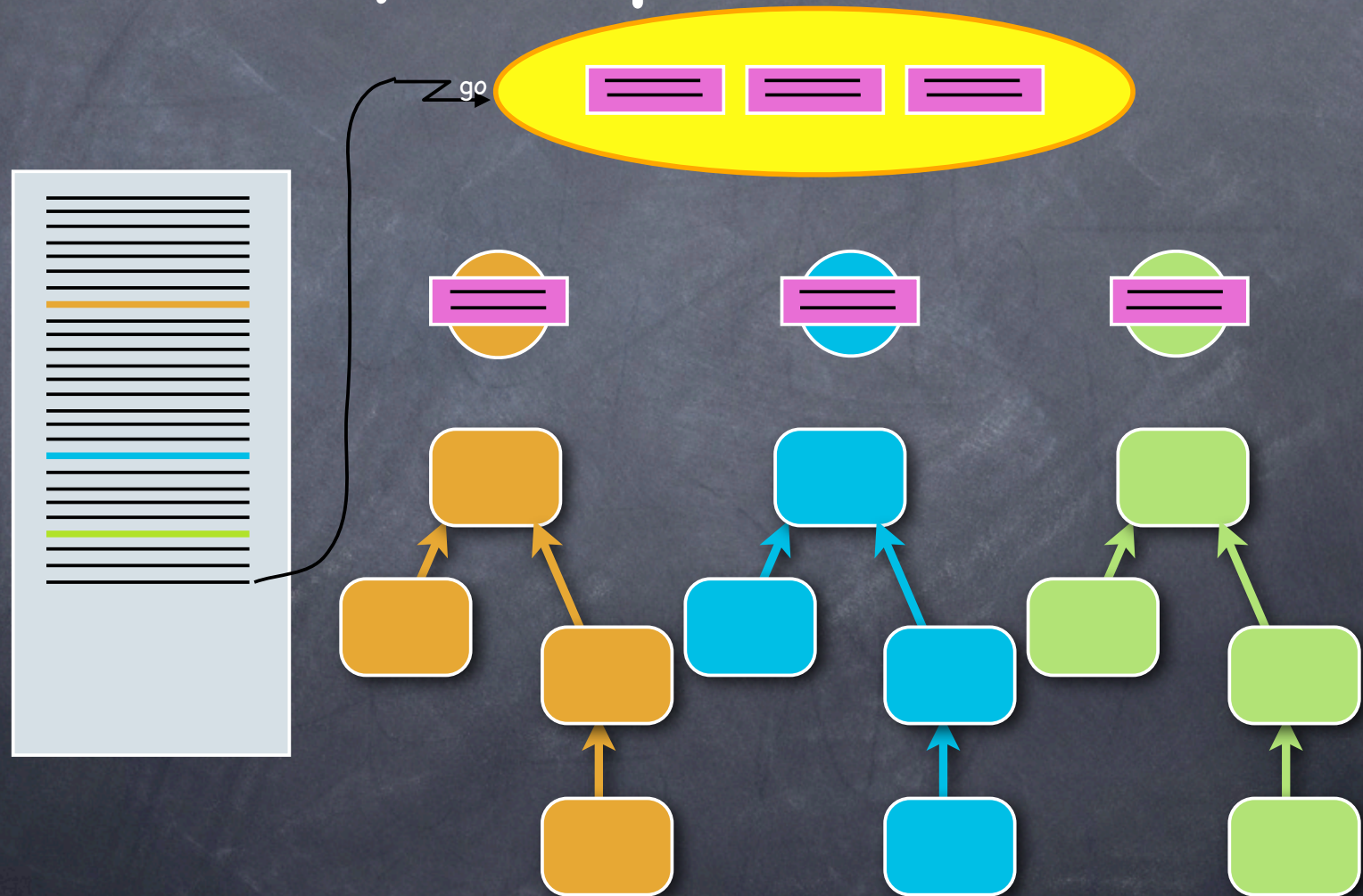
# Contextualized Polymorphism

Context:  
Another  
new  
concept



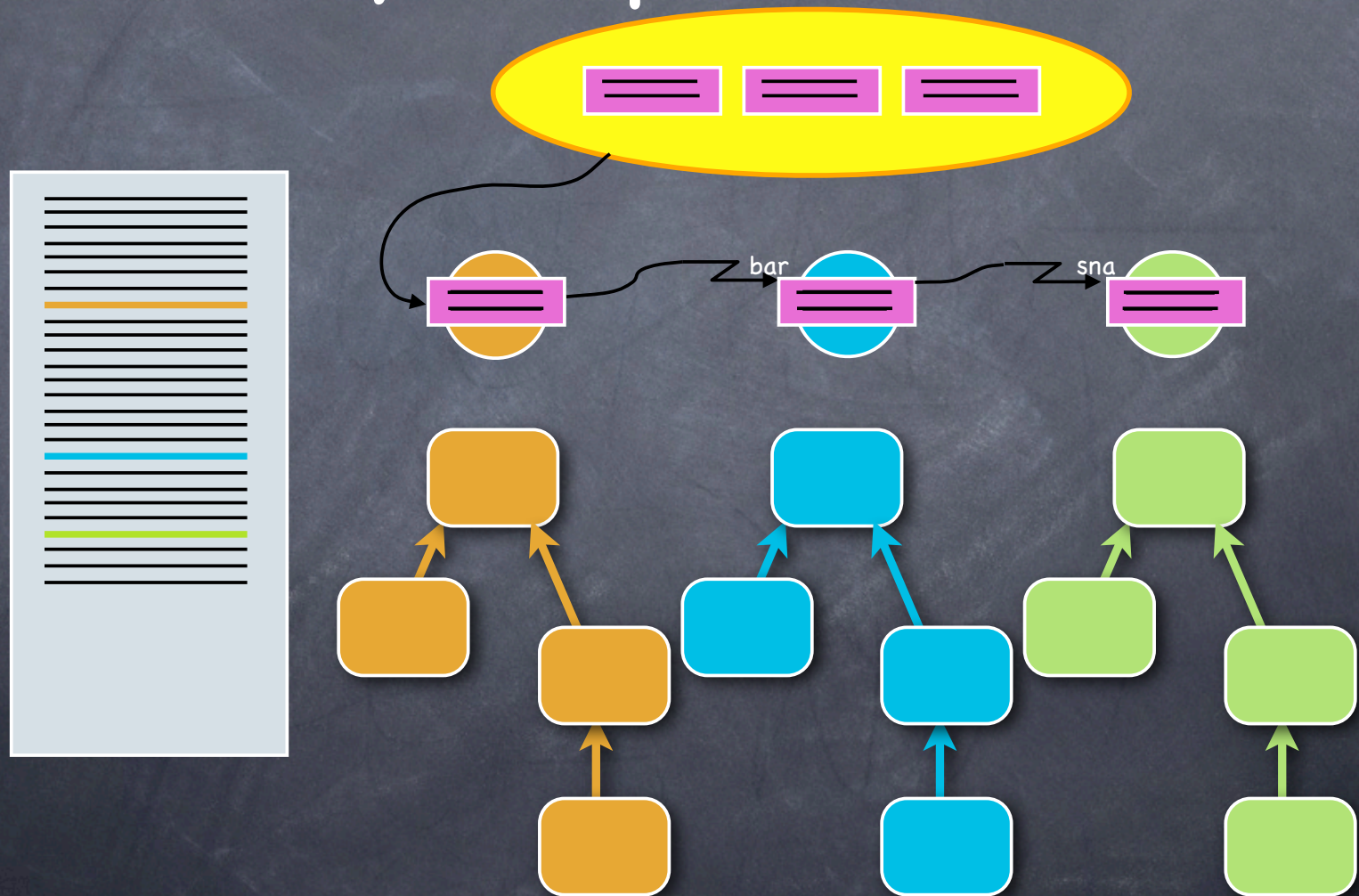


# Contextualized Polymorphism





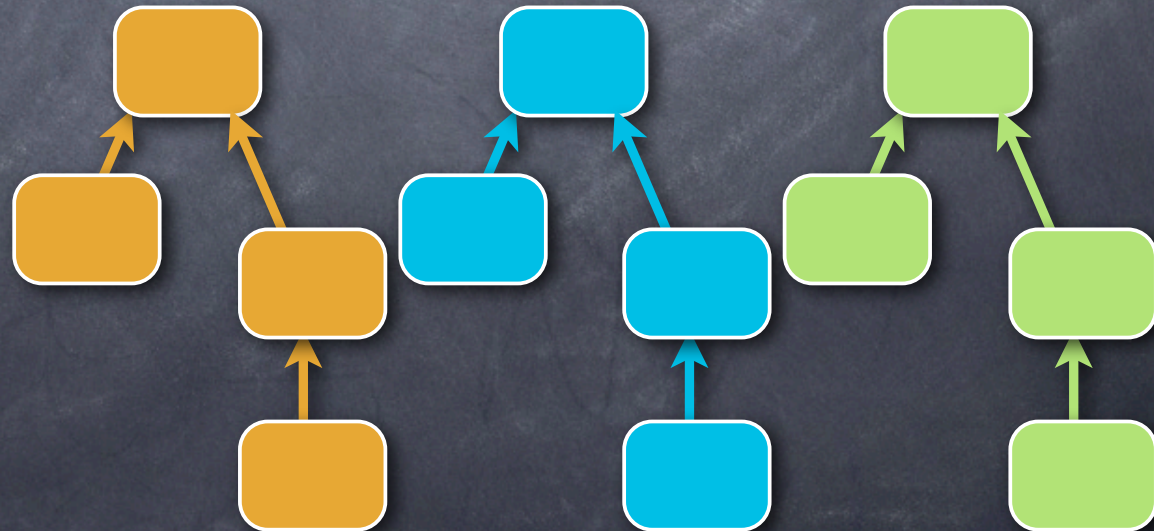
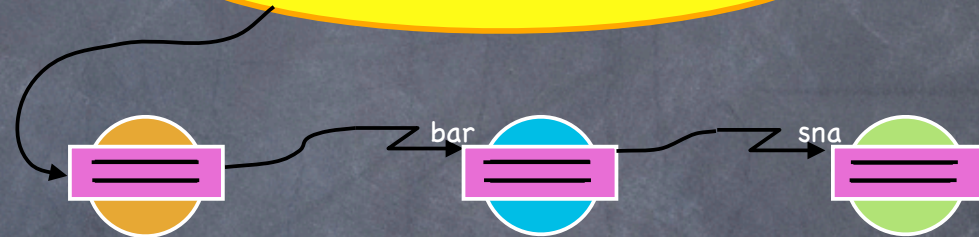
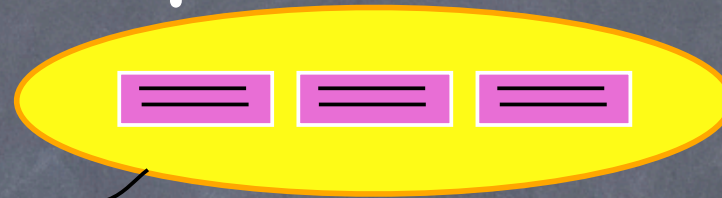
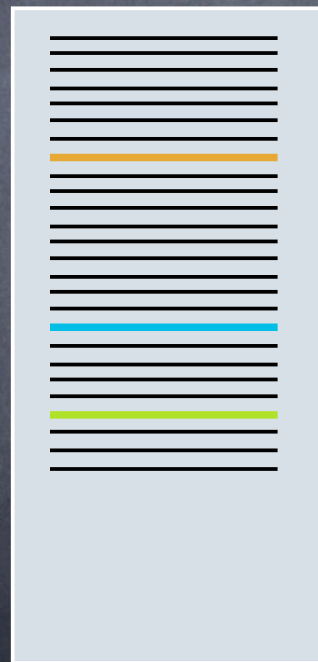
# Contextualized Polymorphism





# Contextualized Polymorphism

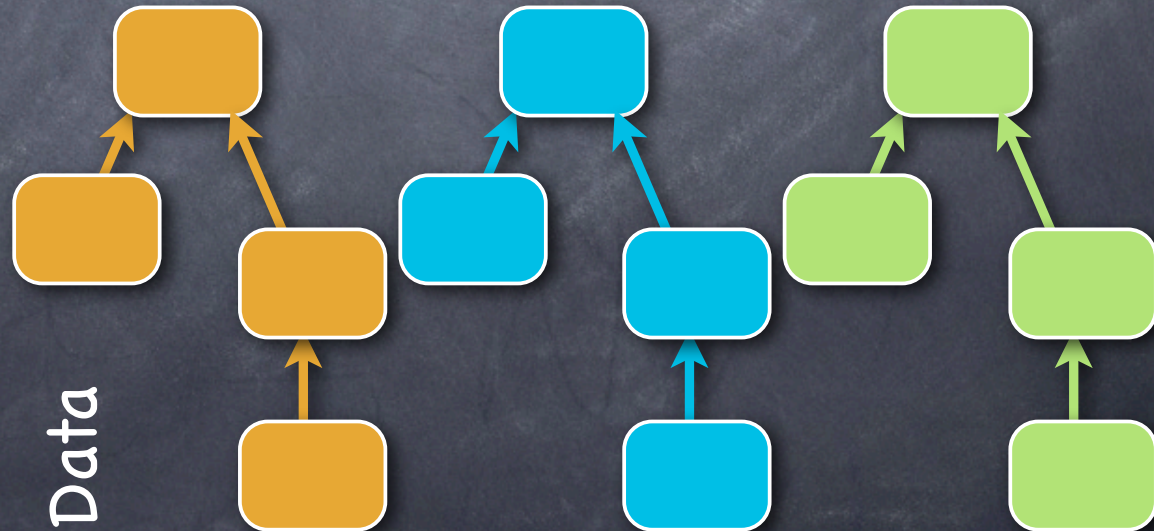
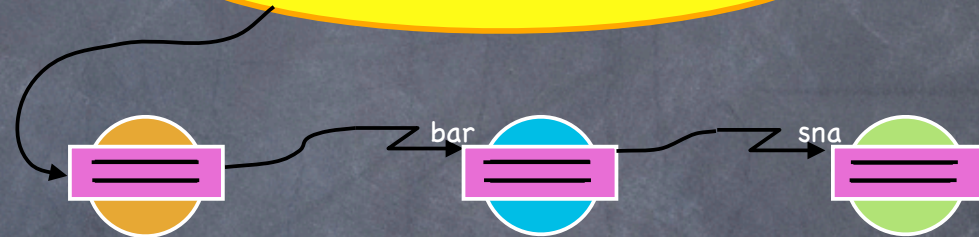
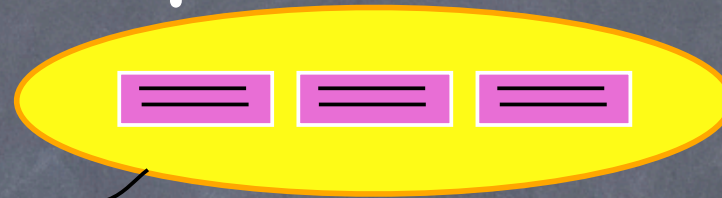
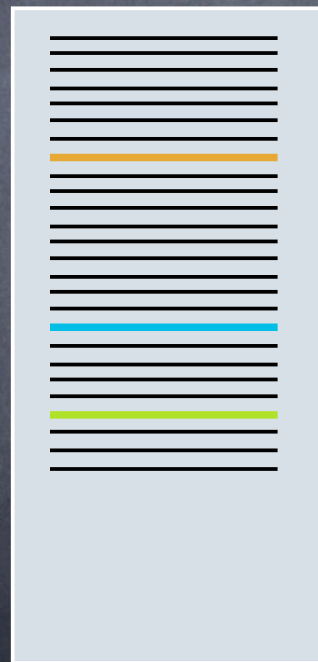
System Operations





# Contextualized Polymorphism

System  
Operations

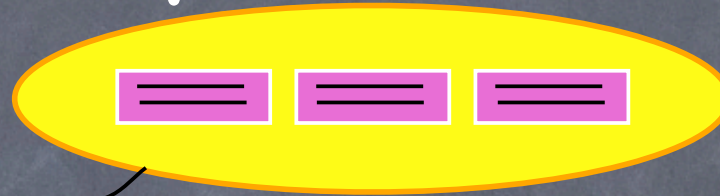
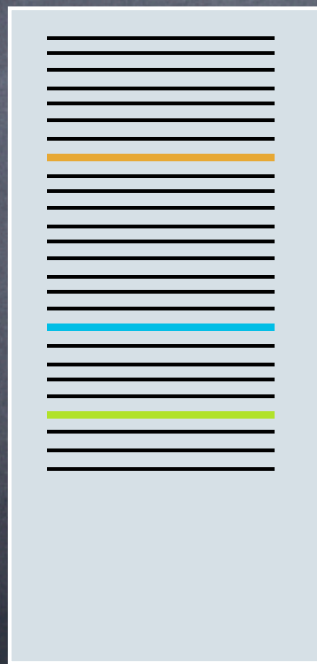


Data

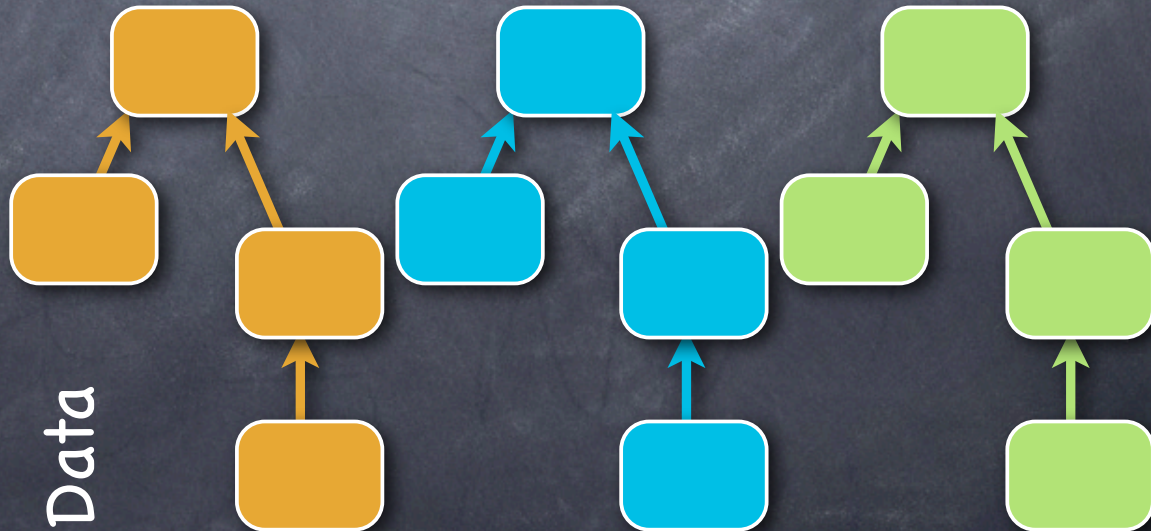
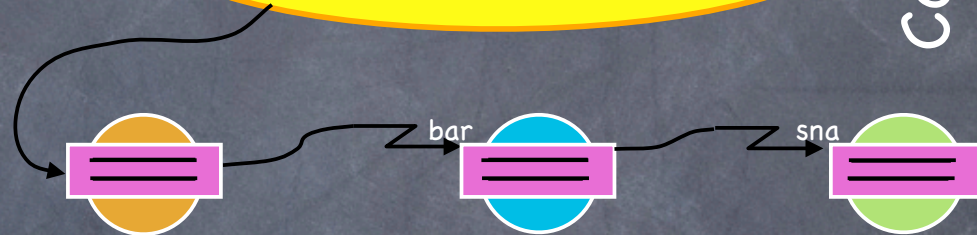


# Contextualized Polymorphism

System Operations



Context

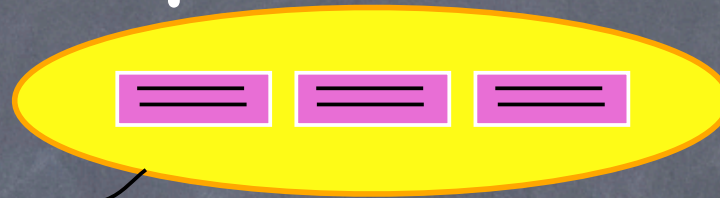
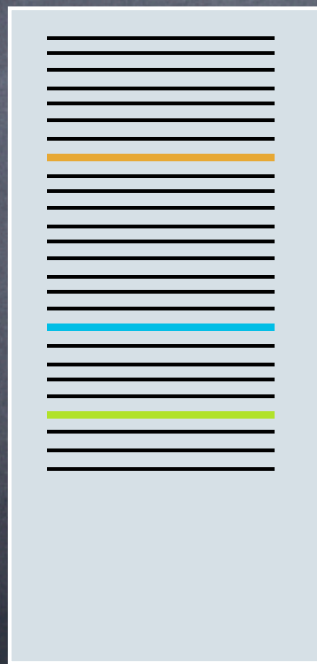


Data

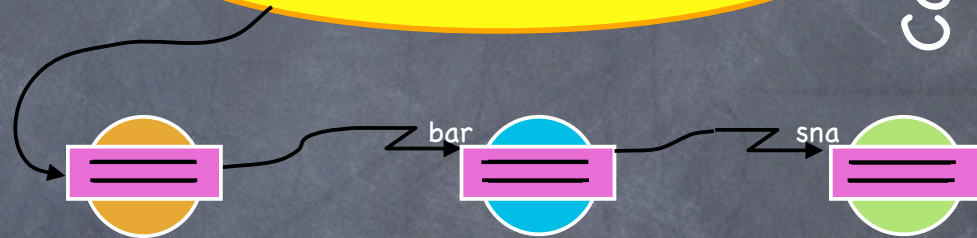


# Contextualized Polymorphism

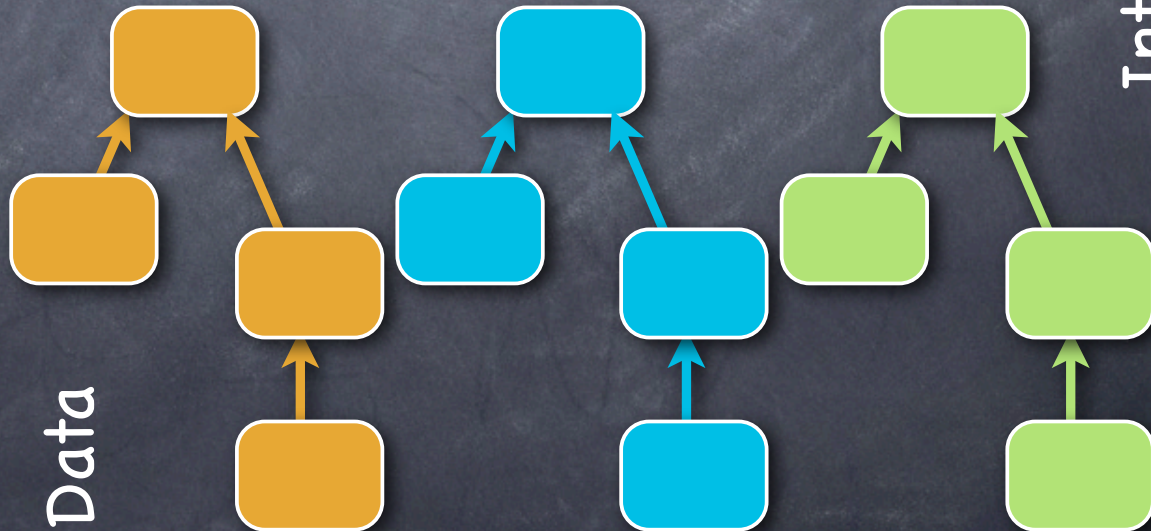
System Operations



Context



Interaction



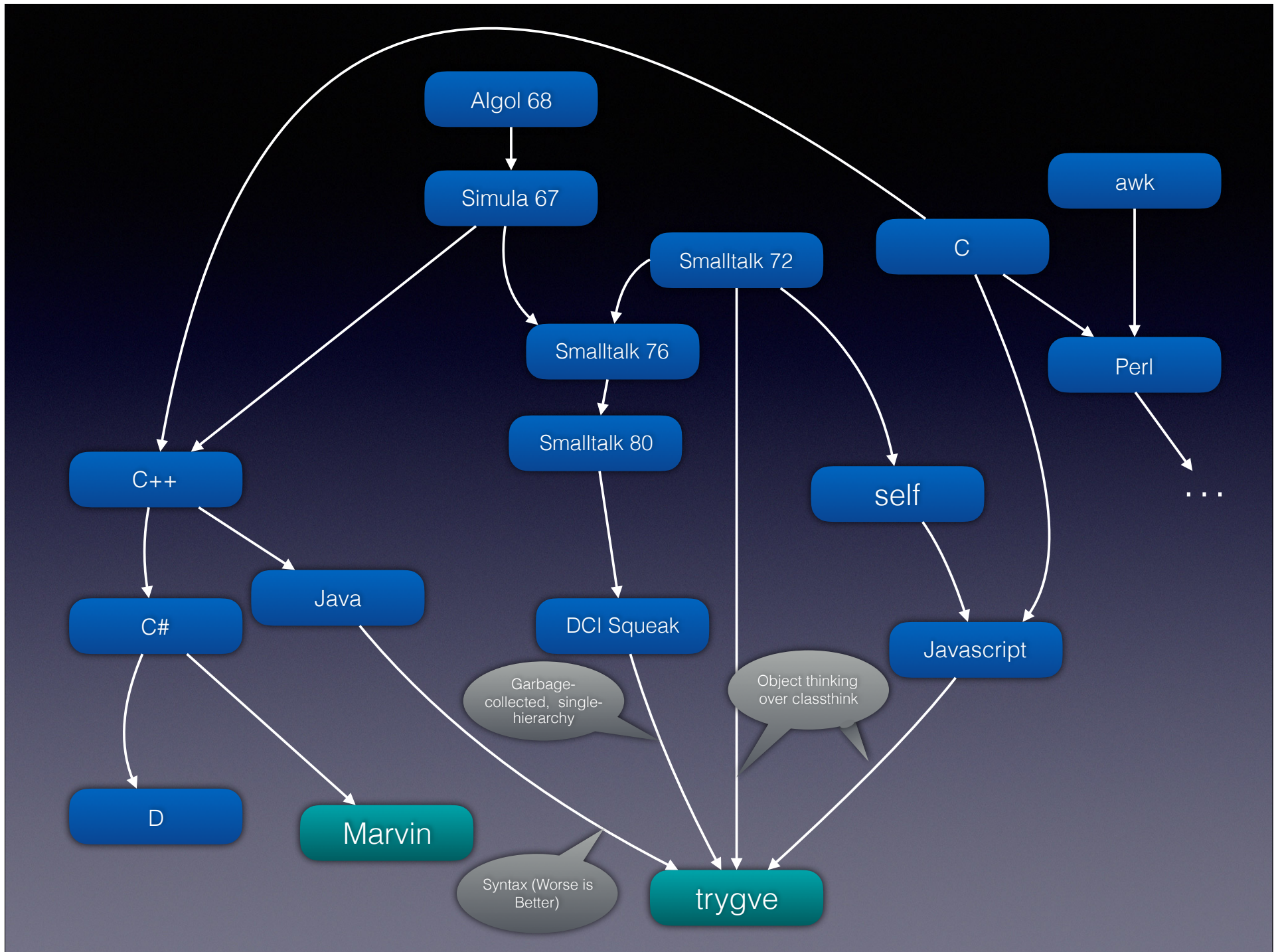
Data



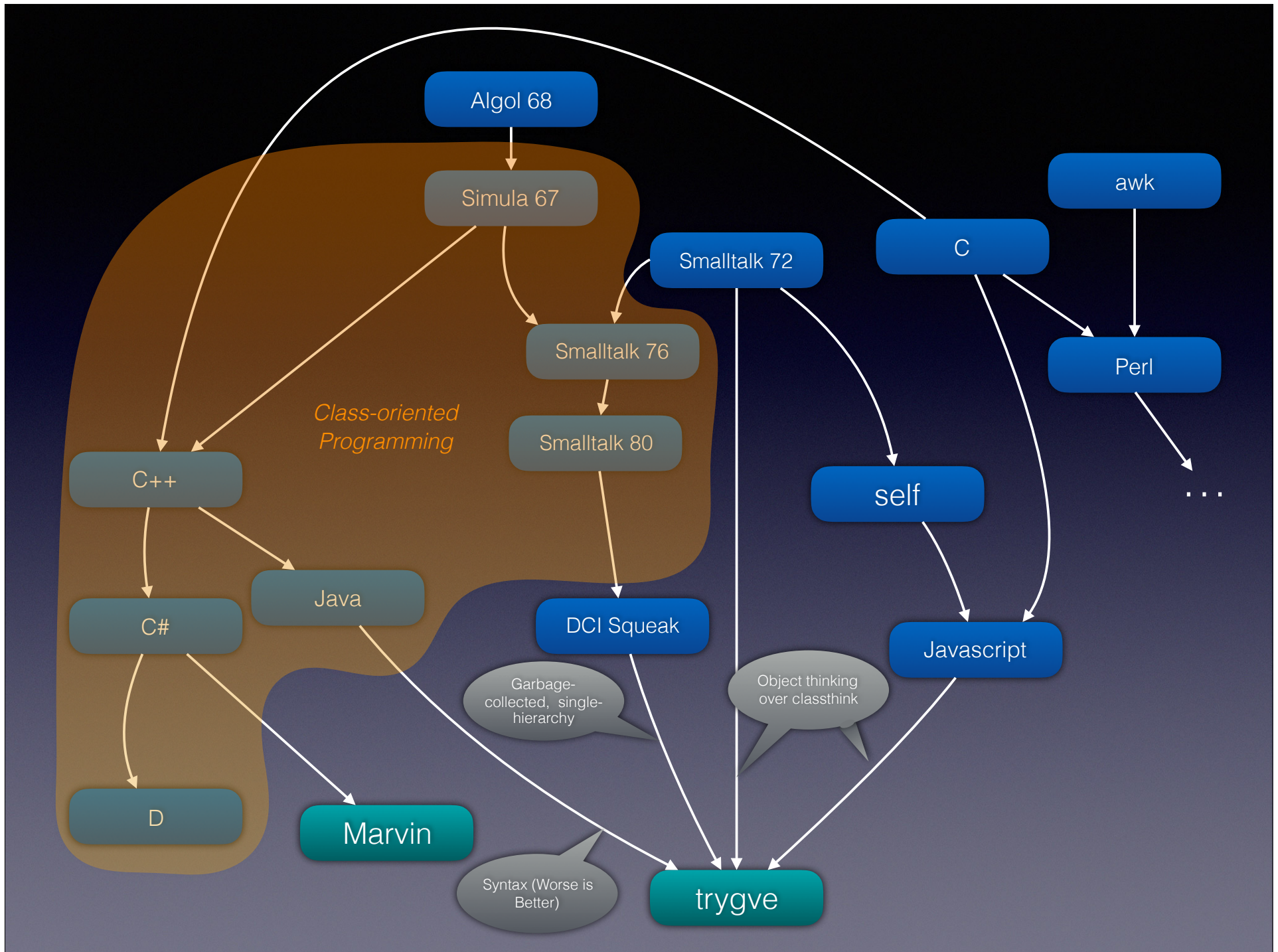
# Hoare's Insight

- “There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.” — Tony Hoare
- The primary goal of **trygve** is to make system operation code readable
- It is in many ways a language for 7-year-olds

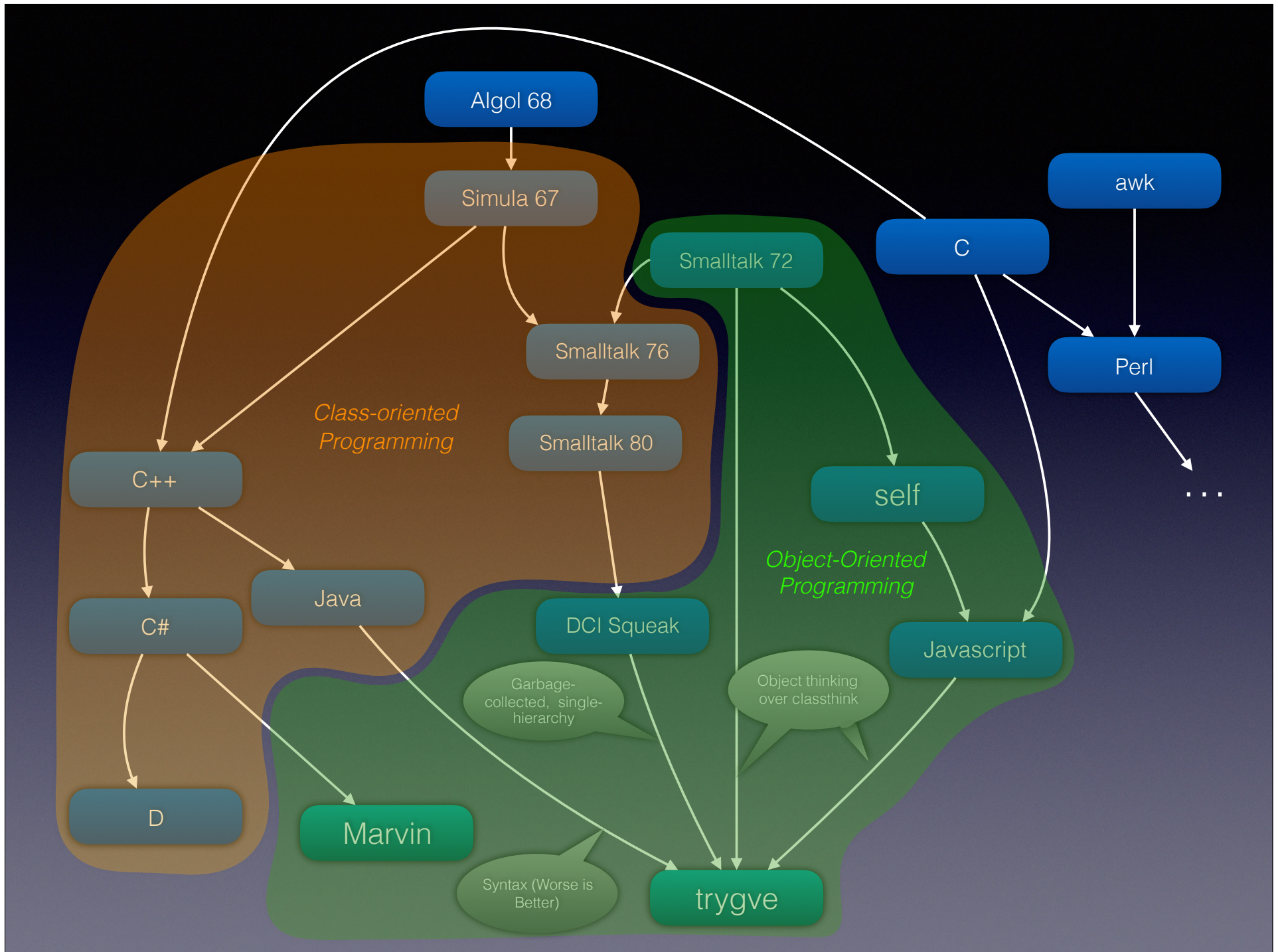














# The **trygve** language

- Contexts: System use cases in code
- The goal: readable code
- System use case steps are demarcated along Role boundaries
- The Context chooses objects — “Role-players” — for each Role (one-time binding)
- *Many forms* of object can play each Role — duck-typed through a contract specification
- Role-players are dumb, and Role / instance contracts should be simple and primitive



# trygve building blocks

- Declarations and expressions — and one value
- Parser swallows most naive Java syntax
- Contexts: use cases — mainly a set of Roles
- Classes: the “domain model” (dumb): actor DNA
  - Classify objects by how they are *built*
- Roles: scripts for the actors — stateless
  - Classify objects by how they *act*



- So this works:

```
int fact(int n) {  
    int retval = 1;  
    if (n > 1) retval = n * fact(n - 1);  
    return retval  
}
```

- but this is orthodox:

```
int fact(int n) {  
    return if (n <= 1) 1 else n * fact(n-1)  
}
```



# Details

- Semicolons optional
- Classes, but few class-oriented features (e.g., there is no **protected**)
- Strongly type-checked at run-time-typed; Roles are duck-typed
- Rudimentary templates
- No exceptions, RTTI, concurrency



# More going on here than meets the eye

- The **trygve** language is a stepping stone to side-effect free programming
- States make it difficult for a Role-player to play several Roles (the MI problem)
- Stateless computation transcends the problem



# Both class and Context instances can play Roles

- ... in which case, *we really don't need classes any more*
- *They never were part of the OO vision*
- The **trygve** language supports more or less arbitrary scope nesting (anything inside anything)
- Classes become truly *operational models*: exactly the Piagetian ideal that Kay strove for



# Both class and Context instances can play Roles

- ... in which case, *we really don't need classes any more*
- *They never were part of the OO vision*
- The **trygve** language supports more or less arbitrary scope nesting (anything inside anything)
- Classes become truly *operational models*: exactly the Piagetian ideal that Kay strove for



# On babies and bathwater

- Classes are still part of the business vocabulary
- They are at least part of the learned business mental model
- They are certainly part of the programmer mental model — and programmers are people, too
- So the “classless movement” is really a fad



# Reflection

- Even common inclusion polymorphism is overly general — reflection takes it far outside human mental models
- **trygve** slices reflection with a precisely honed scimitar
- Reflection is the Turing Machine of types
- It shows a lack of design discipline in the articulation of a paradigm
- See “Reflections on Reflection,” SPLASH 2013 keynote



# Conclusion: Everybody Wins

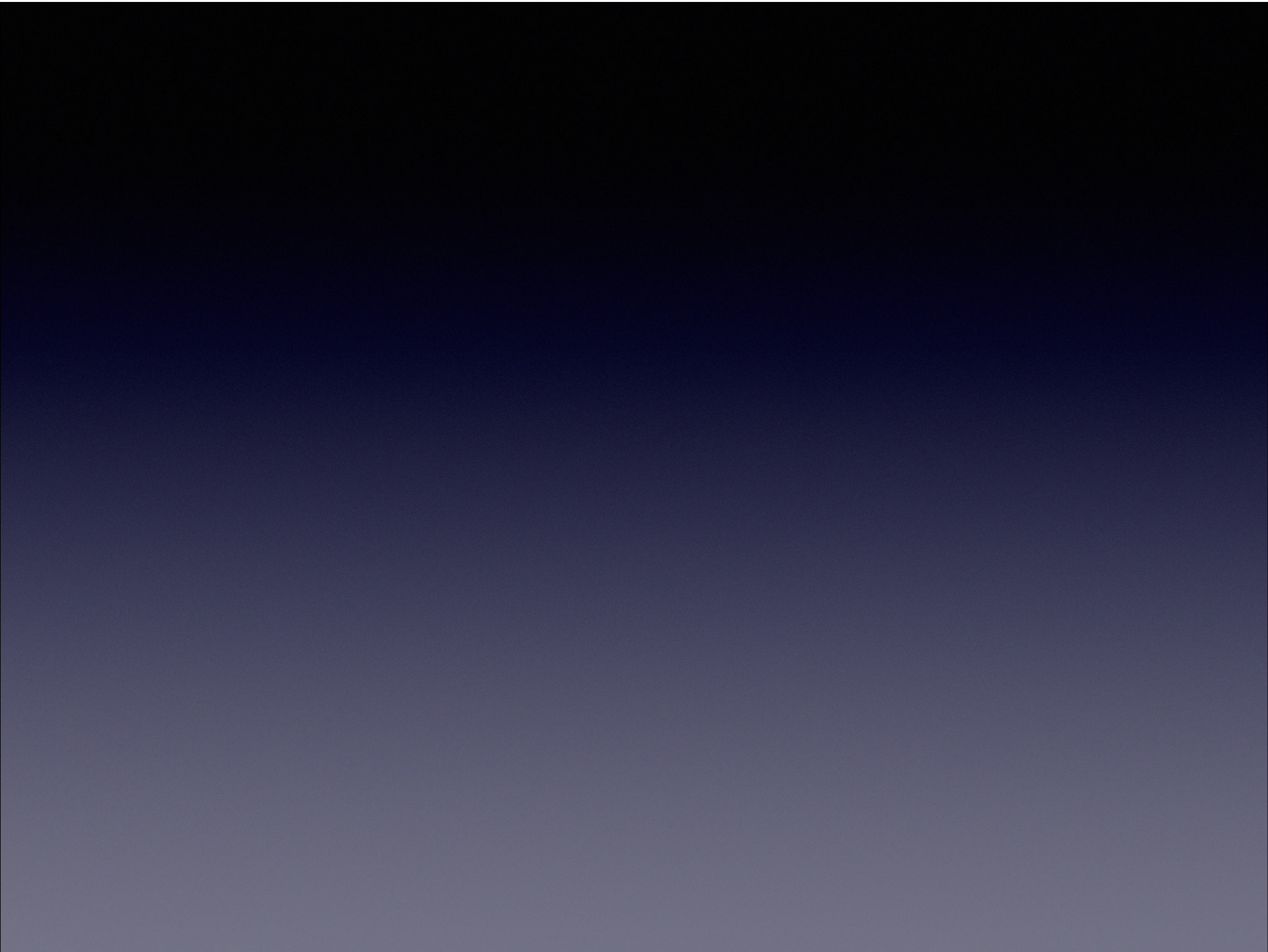
- Good for humans...
  - Both cognitive & volitive models in code
  - Organizes, rather than suppresses, complexity
- ... and software engineering
  - Incremental addition of new use cases
  - Reduced discovery cost
  - Separates rate-of-change shearing layers



# Postlog

- **trygve** is an open-source community research effort — join it (GitHub jcoplien / trygve)
- Also a binary download and documentation at [fulloo.info](http://fulloo.info).
- We've been stuck gilding the lily and making much ado about nothing for far too long
- **trygve** does not aspire to be king: it exists only to combat ignorance and stimulate thinking & dialog







# Other assorted goodies

- Templates (very minimal — just enough so people can use familiar Java containers)
- Very basic Frames, Panels, Events, Colors
- Rudimentary InputStream & OutputStream I/O
- No exceptions (goal is readability)



# Look, Ma, no semicolons

```
interface EventHandler {
    public void handleEvent(Event e)
}

class MyPanel extends Panel {
    int XSIZE = 1000
    int YSIZE = 600

    public int xsize() { return XSIZE }
    public int ysize() { return YSIZE }

    public MyPanel() {
        Panel()
        eventHandler_ = null
        frame_ = new Frame("Bouncy")
        frame_.add("Center", this)
        frame_.resize(XSIZE, YSIZE)
        frame_.setVisible(true)
        drawRect(0, 0, xsize(), ysize())
        repaint()
    }

    public boolean handleEvent(Event event) {
        boolean retval = true
        if (event.id == Event.MOUSE_MOVE) {
            if (eventHandler_ != null) {
                eventHandler_.handleEvent(event)
            }
        }
        return retval
    }
}
```



# Everything's an expression

```
context SpellCheck {
  role Utilities {
    public boolean isDelim(String c) const {
      return switch (c) {
        case "ø": case "Ø": case "æ": case "Æ": case "å": case "Å":
          false; break
        default: (c < "a" || c > "z") && (c < "A" || c > "Z")
      }
    }
  }
  . . . .
}
```



# Duck-typed Role / Object Contracts

```
role ThePanel {
  public void drawCircle(int x, int y, int r) {
    fillOval(x+r, y+r, r, r)
  }
  public void drawPaddle(int xs, int ys, int h, int w) {
    drawRect(xs, ys, h, w)
  }
  public int maxX() { return xsize() }
  public int maxY() { return ysize() }
  public void setColor(Color c) { setForeground(c) }
  public void clearObjects() { removeAll() }
  public void clear() {
    setColor(new Color(227, 221, 240));
    fillRect(0, 0, maxX() - 1, maxY() - 1 )
  }
} requires {
  void fillOval(int x, int y, int h, int w);
  void drawRect(int x, int y, int h, int w);
  void fillRect(int x, int y, int h, int w);
  int xsize();
  int ysize();
  void removeAll();
  void setForeground(Color color)
}
```



# A Puzzle

```
class ArrayDupTest {
    public void test() {
        int [] intArray = new int[5];
        for (int i = 0; i < 5; i++) {
            intArray[i] = i
        }
        for (int i = 0; i < 5; i++) {
            System.out.println(intArray[i])
        }
    }
}

new ArrayDupTest().test()
```



# Most class-oriented features have been removed

- `protected`
- `super`
- `Class::`
- **ABCs**
- `static` (though it exists internally)



# Can an object play more than one Role?

- In series, yes: that's the whole idea
- In parallel, no...
  - One Context could instantiate another Context
  - That Context could share Role-players with the original
- ... unless it's in the same Context so that there is no confusion





# The Object Machine

- Classes are run-time objects (but not in the language)
- Two scratch stacks: main, and event
- Two activation record stacks
- Uses Java GC + Context reference counting
- Like Smalltalk in that everything is an object