



Research Consortium in Speckled Computing

# Multiple Dispatch and Code Diffusion

Andy Koppe

University of Edinburgh

[andy.koppe@ed.ac.uk](mailto:andy.koppe@ed.ac.uk)



# Method Dispatch

- essential feature of object-oriented programming languages
- “methods” can have multiple implementations
  - contrasts with functions or procedures in imperative languages like C or Pascal
- method dispatch selects the most appropriate implementation when method is invoked
  - not to be confused with method or operator overloading which is resolved at compile-time

# Single Dispatch

- example

```
abstract class Animal {  
    abstract String noise();  
}  
class Dog extends Animal {  
    String noise() { return "Woof!"; }  
}  
class Cat extends Animal {  
    String noise() { return "Meeow!"; }  
}
```

- selected method depends on receiving object

```
Animal animal = new Dog();  
print(animal.noise());
```

⇒ "Woof!"

# Limits of Single Dispatch

- lack of functional extensibility
  - standard single-dispatch languages only provide type extensibility (through subclassing)
  - new methods cannot be added to an existing class without changing and recompiling its source code
  - workarounds like the Visitor pattern are cumbersome and sacrifice type extensibility
  - ‘partial classes’ are one solution

# Limits of Single Dispatch

- difficulty of modelling problems where answer depends on types of multiple objects

```
abstract class Animal {
    abstract String react(Animal other);
}
class Cat extends Animal {
    String react(Animal other) {
        if (other instanceof Cat) return "Meeow!"
        else if (other instanceof Dog) return "Run!"
    }
}
```

- not extendable with new classes

# Multiple Dispatch

- method selection can depend on all arguments
  - consequently, there is no distinguished receiver object in method calls
    - `react( animal, other )` instead of `animal.react( other )`
  - also, methods do not belong to a class anymore and are defined outside class declarations
    - this provides an elegant solution to the functional extensibility problem

- “type-oriented programming”
- implements multiple dispatch
- syntax examples

- type declaration

```
! Point { x : Int, y : Int };
```

- method declaration

```
! draw_line (Point, Point) : ();
```

- method definition

```
draw_line (Point p, Point p') = ...
```

# TOP Type System

- supports parametric types
- ensures that methods are completely and unambiguously implemented
- variables and fields are immutable
  - mutable storage has to be declared explicitly
- no ‘null’ references
  - avoids the dreaded NullPointerException

# Example

- animal example in “Top”

```
! ¬Animal;  
! noise Animal : String;  
! react (Animal,Animal) : String;
```

```
! Dog < Animal;  
! Cat < Animal;
```

```
noise Dog = "Woof!";  
noise Cat = "Meeow!";
```

```
react (Dog,Dog) = "Bark!";  
react (Cat,Cat) = "Meeow!";  
react (Cat,Dog) = "Run!";  
react (Dog,Cat) = "Chase!";
```

# Extensibility

- type extensibility

```
! Mouse < Animal;
```

```
noise Mouse = "Feep?";
```

```
react (Mouse, Mouse) = "";
```

```
react (Mouse, Dog) = "Hide.";
```

```
react (Dog, Mouse) = "Bark!";
```

```
react (Mouse, Cat) = "Run!";
```

```
react (Cat, Mouse) = "Hunt!";
```

- functional extensibility

```
! food Animal : String;
```

```
food Dog = "Bones";
```

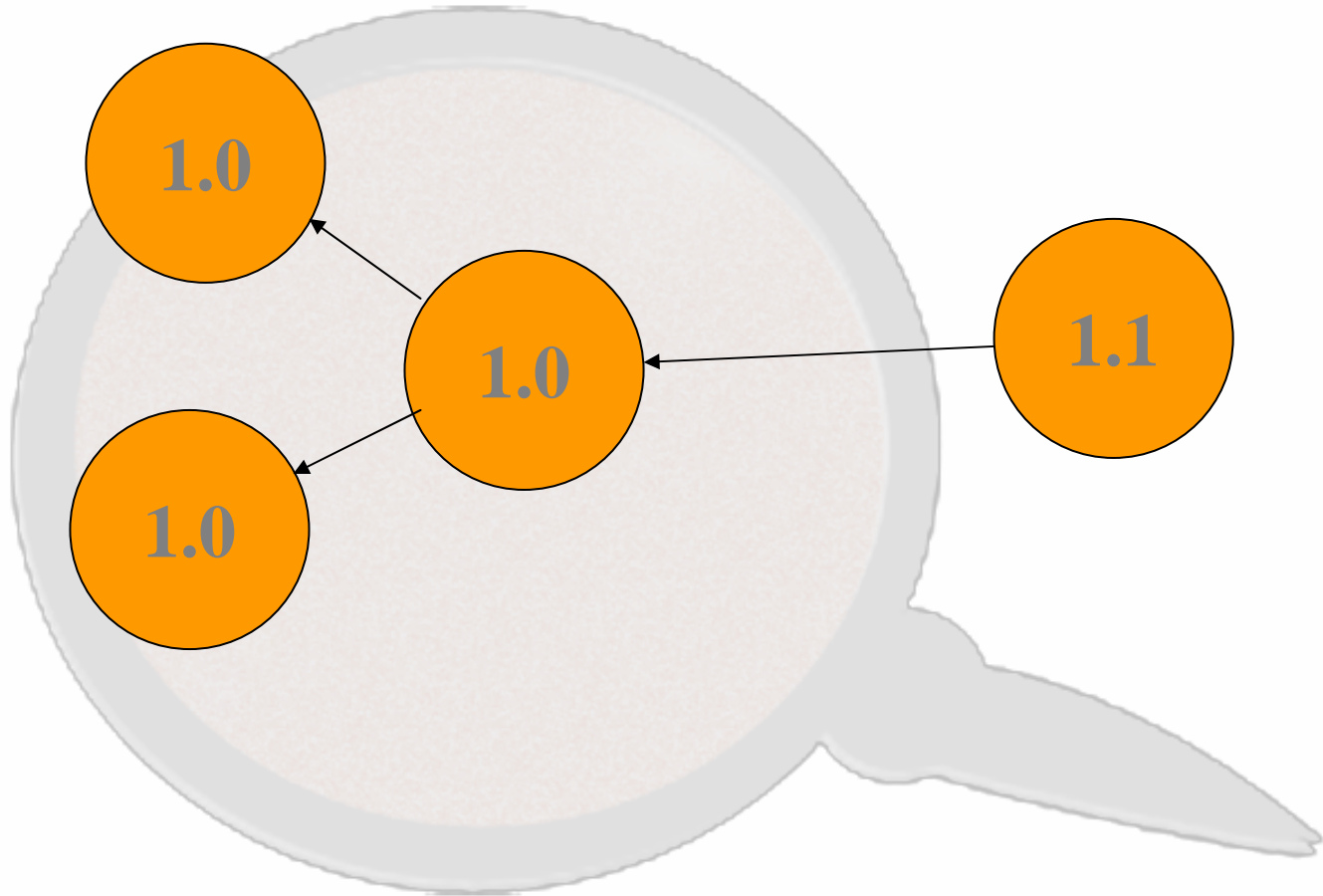
```
food Cat = "Mice";
```

```
food Mouse = "Grains";
```

# Code Diffusion

- approach for distributing “Top” program code in a network
- nodes communicate by sending objects in a serialised form
- diffusion of code through on-demand transmission
- no need for central programming station

# Code Diffusion



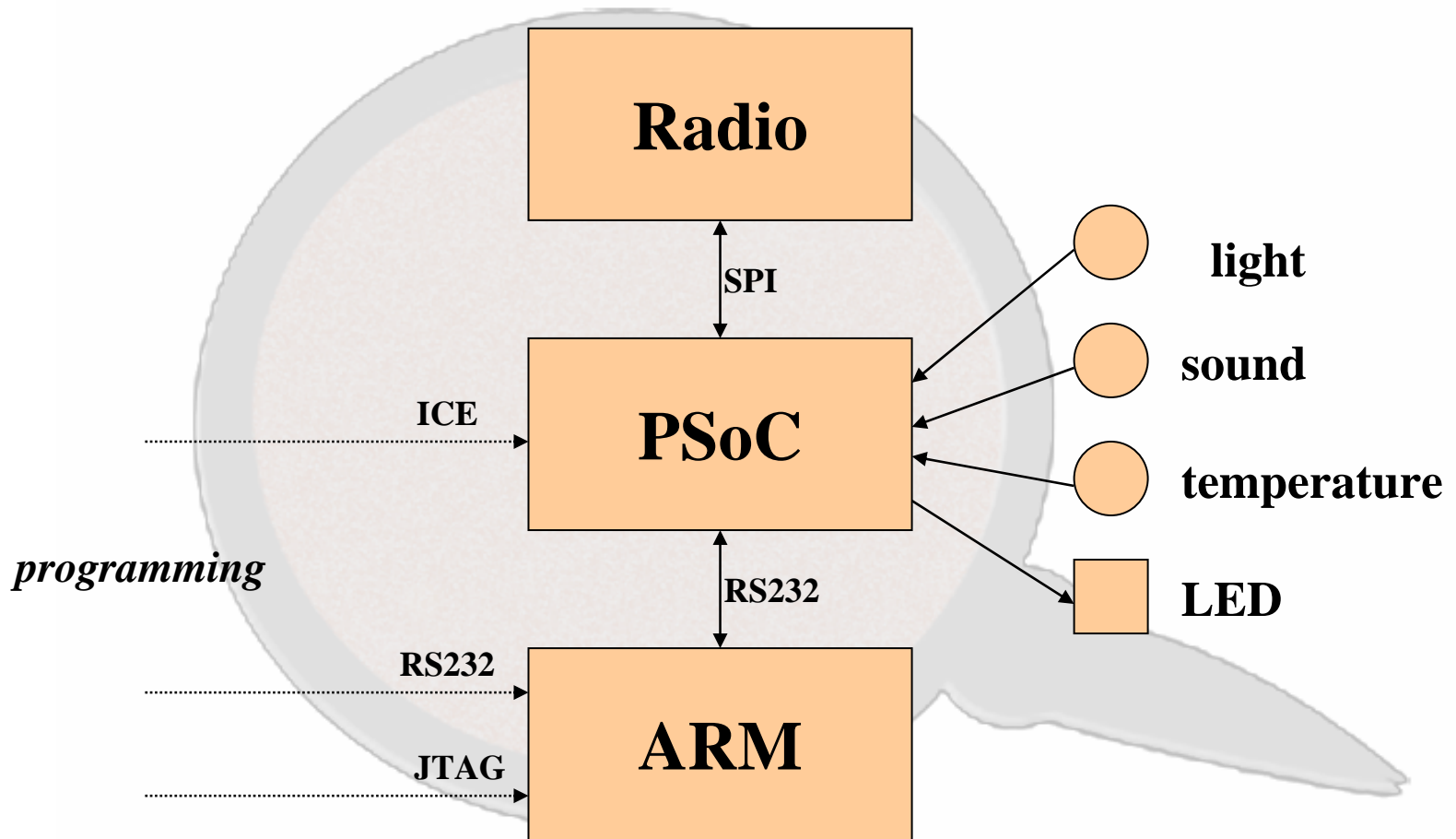
# Code diffusion

- three cases
  - on receiving an object with unknown type
    - type definition is requested from network
  - on dispatching a method on new type
    - suitable method implementations are requested
  - on calling an unknown method in new implementation
    - method declaration is requested

# ProSpeckzIII

- Atmel AT91FR40162 processor
  - 60 MHz ARM core
  - 256 kB RAM, 2 MB Flash
- ChipCon CC2420 2.4 GHz ZigBee radio
- Cypress CY8C27643 PSoC
  - 256 bytes RAM, 16kB Flash
  - configurable analog circuitry
  - sensors: light, sound, temperature

# ProSpeckzIII



# PSoC Library

- no direct PSoC programming required
- PSoC controls radio and sensors
- ARM directs PSoC through command-response protocol on serial line

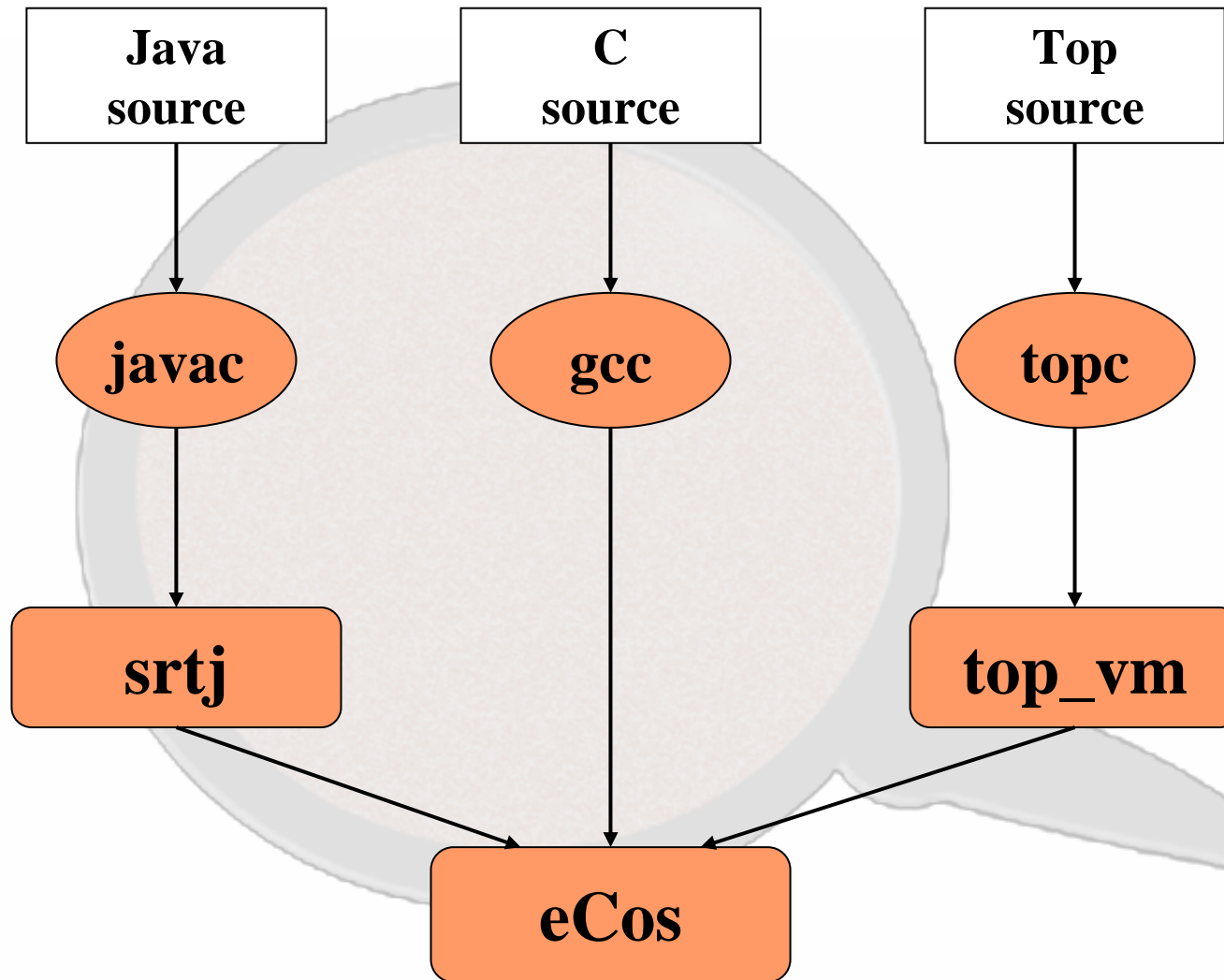
- real-time operating system
- small memory footprint (few tens of kB)
- runs on a large number of processors and platforms
- widely configurable to suit particular platform and application
- provides input/output drivers, dynamic memory management, thread-level parallelism, timers



# Simple Real Time Java

- small memory footprint (around 50k)
- portable to a range of microcontrollers
- simplified virtual machine
  - no 64-bit support and no sandboxing
  - simplified native interface
- limited class libraries
  - only essential classes included (java.lang)
  - platform-specific I/O

# Programming ProSpeckzIII



# Future Work

- finishing code diffusion implementation
- finishing the compiler, in particular the typechecker
- investigate interaction between code diffusion and type system
- better demos 😊

# Questions

???

