# Objects in Oz

Objects in Oz

Promotionskolloquium

Martin Henz

# Overview

**Part 1 (5 min)**

**Objects**

# Overview

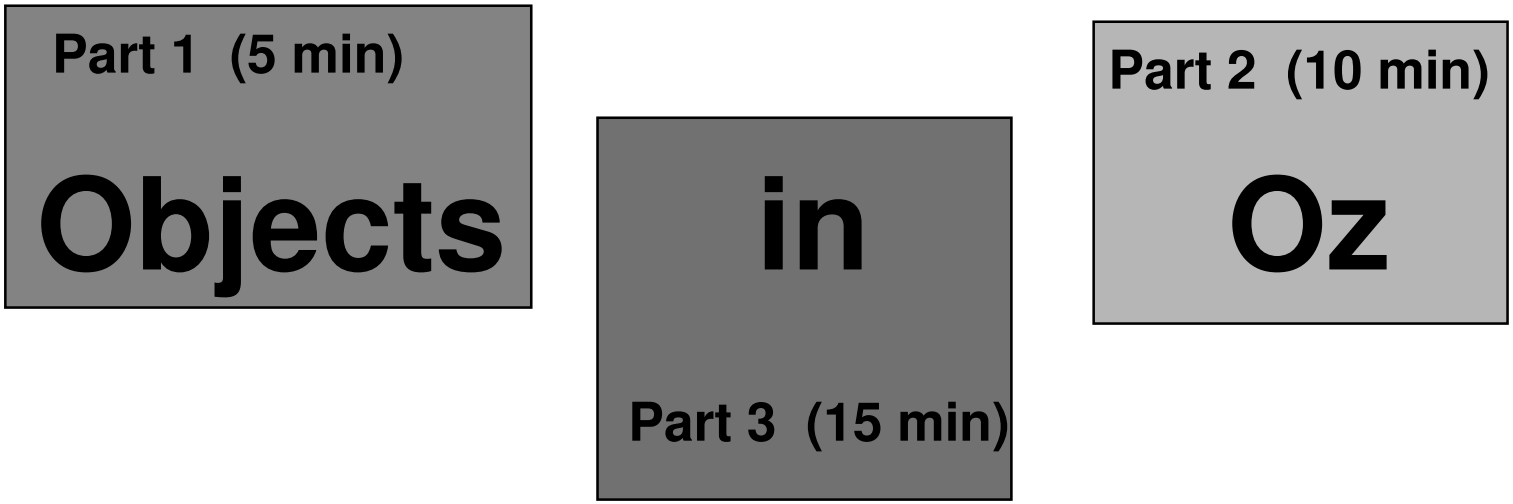
**Part 1 (5 min)**

**Objects**

**Part 2 (10 min)**

**Oz**

# Overview

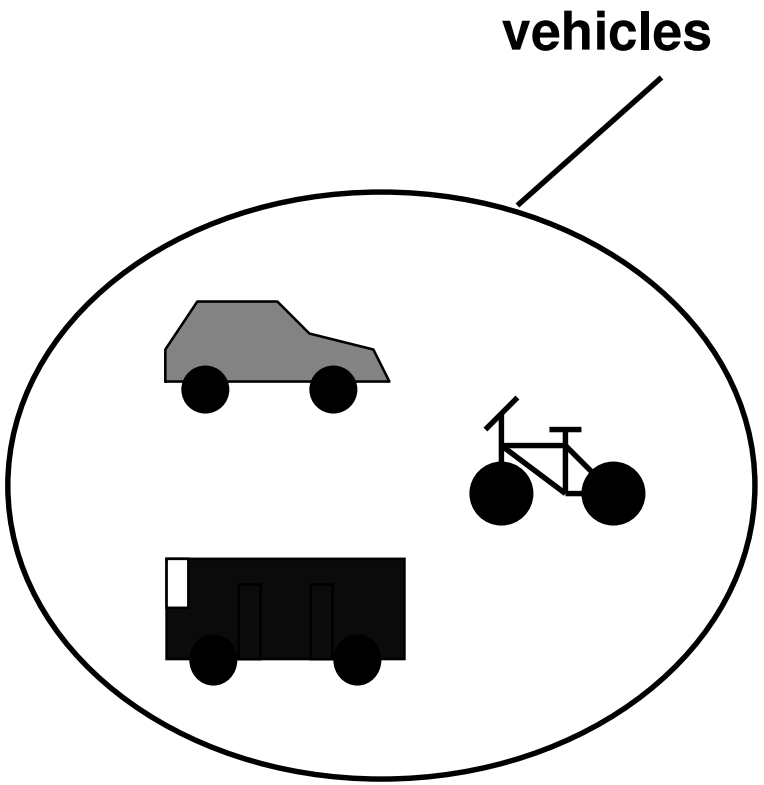


# Part 1: Objects in Oz

- data abstraction principles
- software engineering support

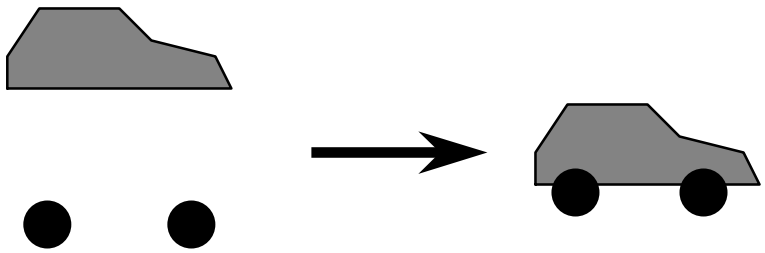
# Data Abstraction Principles

- classification
- aggregation
- specialization



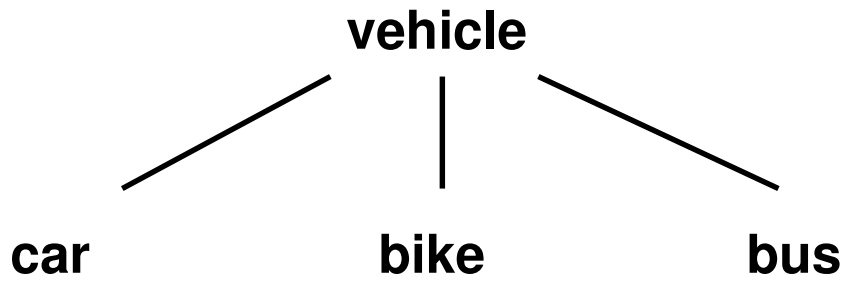
# Data Abstraction Principles

- classification
- aggregation
- specialization



# Data Abstraction Principles

- classification
- aggregation
- specialization



# Data Abstraction Principles

- classification
- aggregation
- specialization

```
class Car
   from Vehicle
   attr tyres
        body
end

MyCar={New Car}
```

# Software Engineering Aspects

- encapsulation
- inheritance
- late binding

# Late Binding

```
class Account
   attr balance
   meth transf(X)
      balance <- @balance+X
   end
   meth transfWithRec(X)
      Konto , überw(X)
      {Receipt @balance}
   end
end
```

# Late Binding

```
class Account
   attr balance
   meth transf(X)
      balance <- @balance+X
   end
   meth transfWithRec(X)
      Konto , überw(X)
      {Receipt @balance}
   end
end
```

```
class AccountWithFee
   from Account
   attr fee : 5
   meth transf(X)
      Account , transf(X-@fee)
   end
end
```

# Late Binding

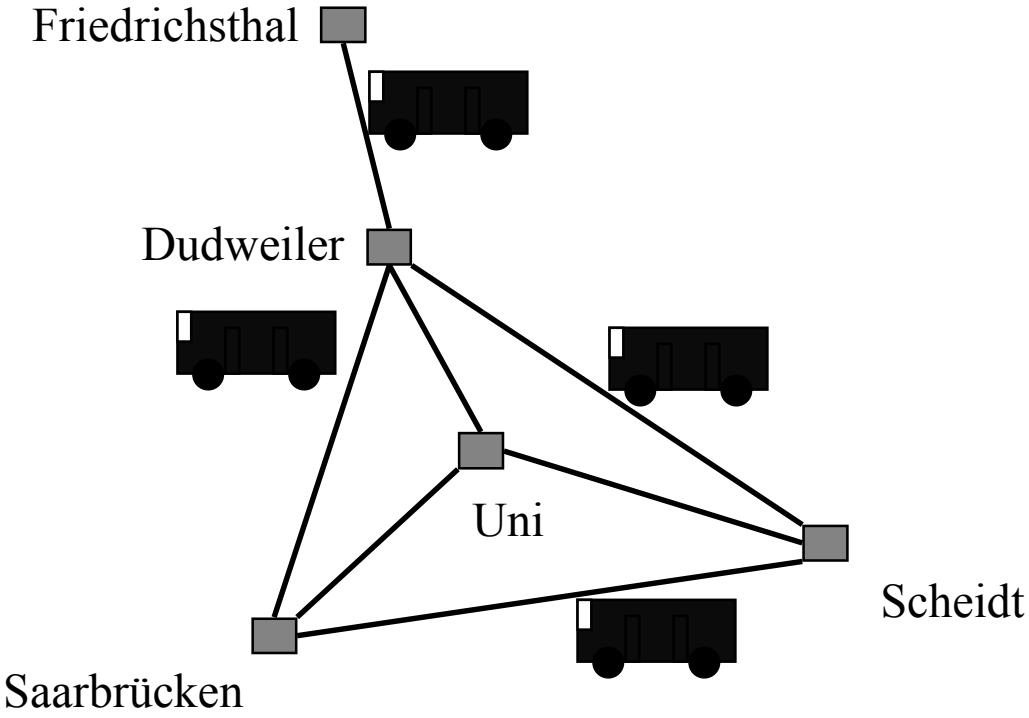
```
class Account
   attr balance
   meth transf(X)
      balance <- @balance+X
   end
   meth transfWithRec(X)
      {self transf(X)}
      {Receipt @balance}
   end
end
```

```
class AccountWithFee
   from Account
   attr fee : 5
   meth transf(X)
      Account , transf(X-@fee)
   end
end
```

# Object-Oriented Programming Languages

| | | |
|---|---|---|
| **Simula** | **Dahl, Nygaard** | **1966** |
| **Smalltalk** | **Goldberg, Kay** | **1972** |
| **CLOS** | **CL Committee** | **1980** |
| **C++** | **Stroustrup** | **1986** |
| **Eiffel** | **Meyer** | **1988** |
| **Java** | **Arnold, Gosling** | **1995** |

# Sequentiality and Concurrency

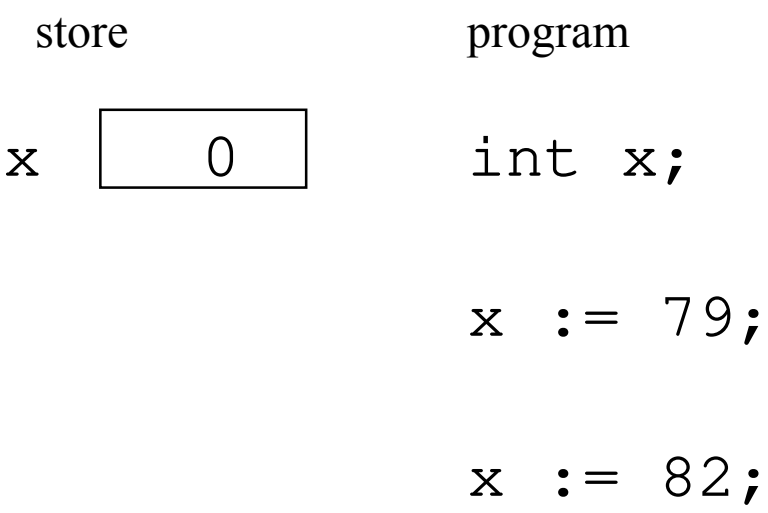


# Part 2: Objects in Oz

- synchronization and communication through logic variables
- integration in modern programming language
- constraint programming

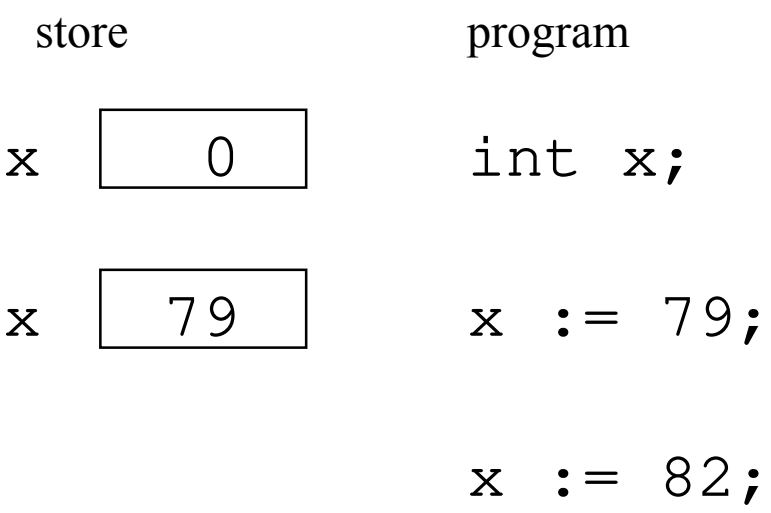
# Partial Information through Logic Variables

conventional
program variables



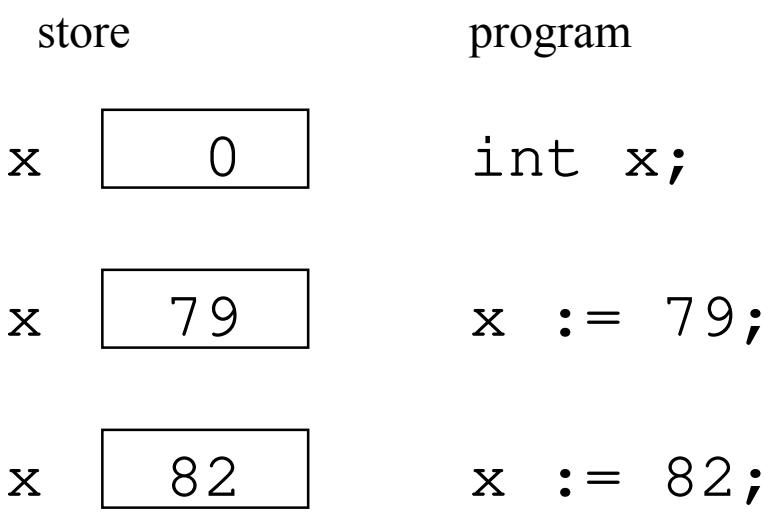
# Partial Information through Logic Variables

conventional
program variables



# Partial Information through Logic Variables

conventional
program variables



# Partial Information through Logic Variables

conventional program variables

| store | | program |
|---|---|---|
| x | 0 | `int x;` |
| x | 79 | `x := 79;` |
| x | 82 | `x := 82;` |

logic variables

| store | | program |
|---|---|---|
| X | | `declare X` |
| | | `X = 79` |
| | | `X = 82` |

# Partial Information through Logic Variables

convenitional program variables

| store | program |
|---|---|
| x 0 | `int x;` |
| x 79 | `x := 79;` |
| x 82 | `x := 82;` |

logic variables

| store | program |
|---|---|
| X | `declare X` |
| X 79 | `X = 79` |
| | `X = 82` |

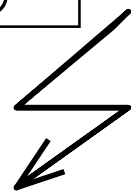
# Partial Information through Logic Variables

conventional program variables

| store | | program |
|---|---|---|
| x | 0 | `int x;` |
| x | 79 | `x := 79;` |
| x | 82 | `x := 82;` |

logic variables

| store | | program |
|---|---|---|
| X | | `declare X` |
| X | 79 | `X = 79` |
| X | 79 | `X = 82` |



# Oz: Logic Variables in a Modern Programming Language

- state
- explicit concurrency
- compositionality

# Synchronized Statements

Example: procedure application

```
declare P in
thread {P 1} end
thread proc {P X} ... end end
```

other synchronized statements

conditionals `if B then...else...end`

basic operations `S = X + Y`

`B = X < Y`

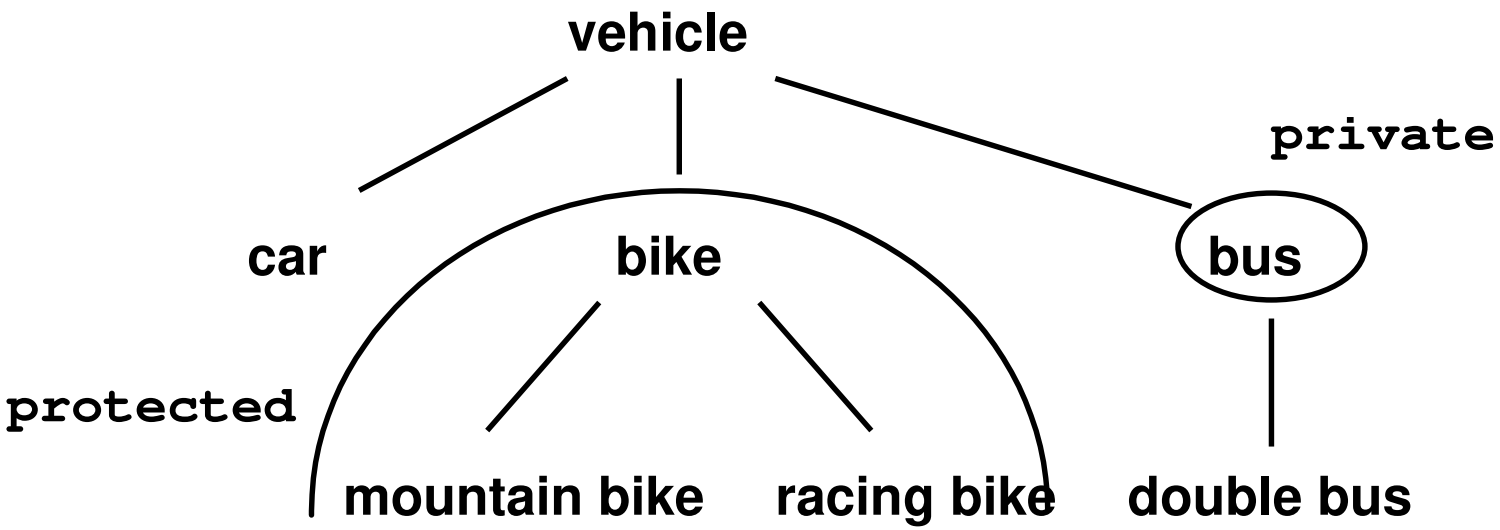
# Part 3: Objects in Oz

- design
- concurrent programming techniques
- implementation

# Design

- conventional objects
- integration into compositional language
- encapsulation through scoping

# Encapsulation



# Encapsulation with Names

```
private

local PrivAttr in
   {NewName PrivAttr}
   class C1
      attr PrivAttr
      ...
   end
end
```

```
protected

local ProtMeth in
   {NewName ProtMeth}
   class C2
      attr protMeth:ProtMeth
      meth !ProtMeth
         ...
      end
      meth otherMethod
         {self @protMeth}
      end
   end
end
```

# The Design in Context

- conventional object-oriented languages
- concurrent logic languages
- functional languages

# Concurrent Programming Techniques

- synchronization techniques
- active objects
- concurrent meta-object protocol

# Synchronization Techniques

- mutual exclusion
- bounded buffer
- time behavior
- locks
- reentrant locks

# Active Objects

- orthogonal concept
- enabled by first-class messages
- application in simulation

# Meta-Object Protocol

describing the concurrent behavior of objects by meta-classes

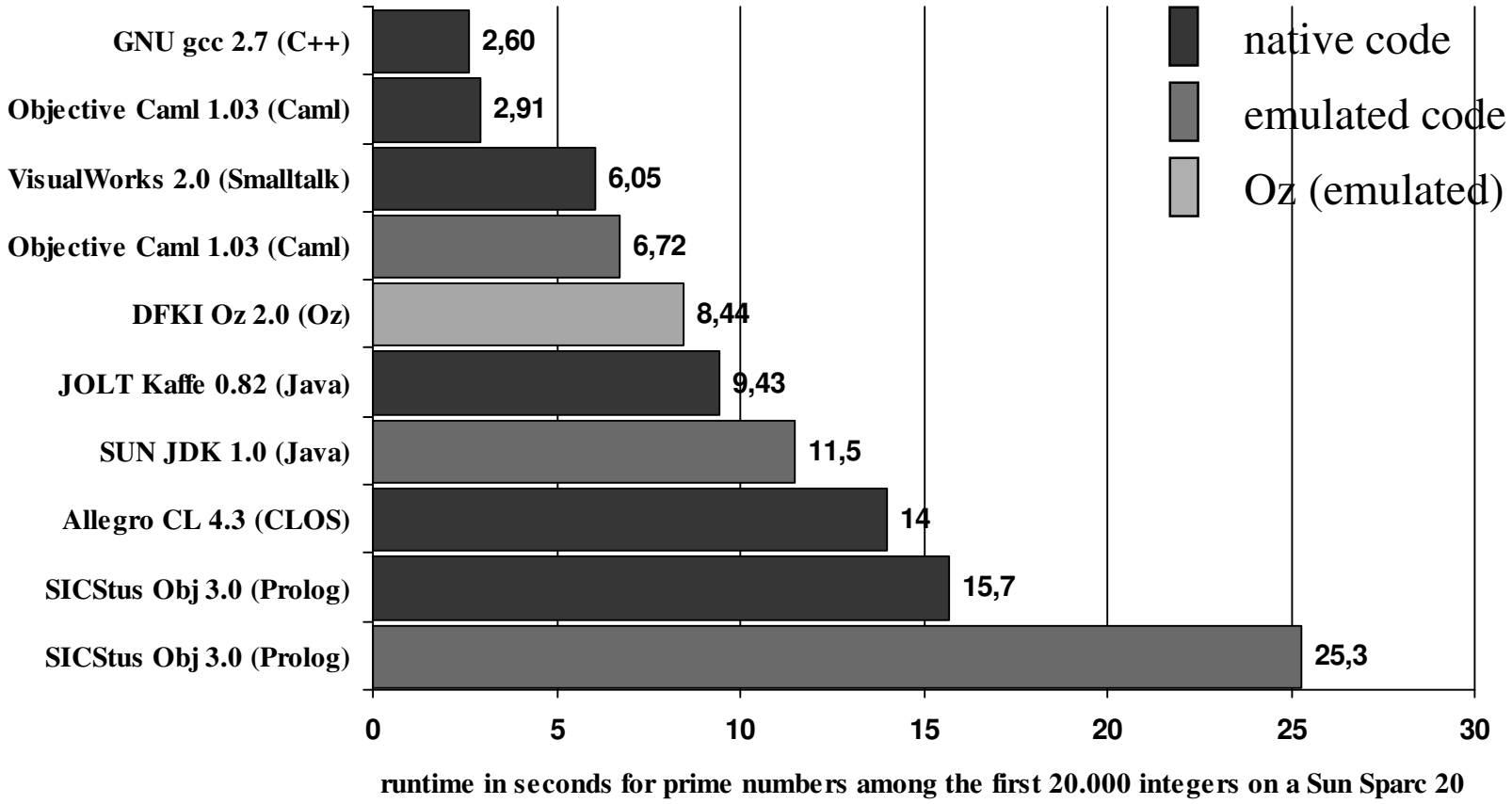
application areas

- debugging
- active objects
- hierarchical locking

# Implementation

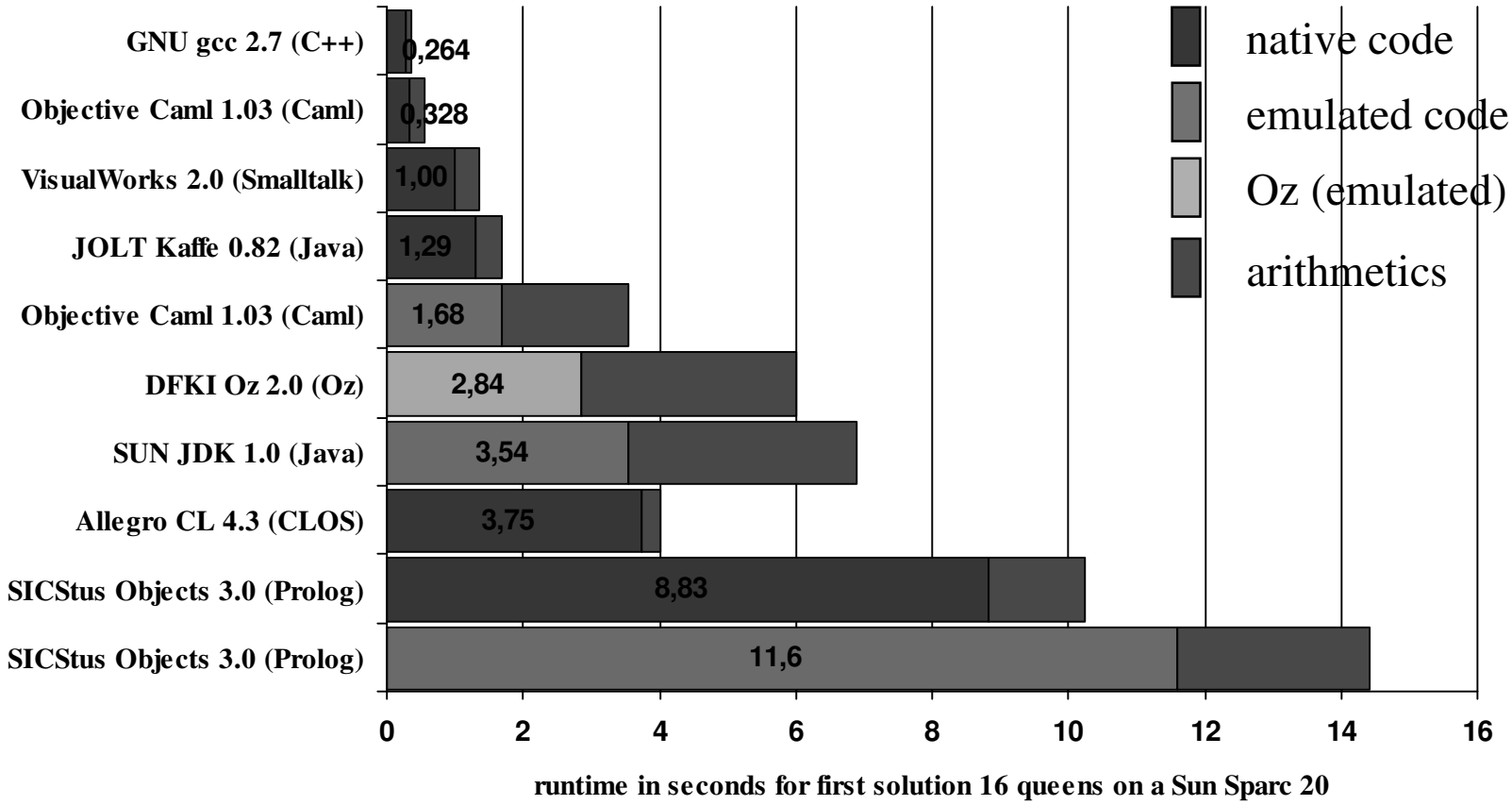
- integration in existing abstract machine
- surgical approach for optimization
  - lookup caches
  - inline caches
  - memory layout for objects
  - first-class messages

# Performance Study 1: Sieve of Erathosthenes



# Performance Study 2:

# N-Queens



# Contributions

- design
- concurrent programming techniques
- implementation

# Possible Further Directions

- other languages?
- concurrent design patterns?
- implementation:

surgical approach

integrated approach