Portable Resource Control in Java
The J-SEAL2 Approach

Walter Binder w.binder@coco.co.at
CoCo Software Engineering GmbH
Austria

Jarle Hulaas Jarle.Hulaas@cui.unige.ch
Alex Villazón Alex.Villazon@cui.unige.ch
Rory Vidal
University of Geneva
Switzerland

Overview

- Motivation
- Portable resource control layer
  - Architecture and functions
  - Memory accounting
  - CPU accounting
- Example
- Evaluation
- Integration with J-SEAL2

Motivation: Java and Mobile Code

- Most mobile code environments are based on Java
  - Applets
  - Mobile objects (agents)
- Protection of host requires resource control
- Java has no support for resource control

Benefits of Resource Control

- Security
  - Prevention against denial-of-service attacks
- E-commerce
  - Billing for resource consumption
  - Quality of service guarantees
- Software development
  - Monitoring and profiling of distributed applications

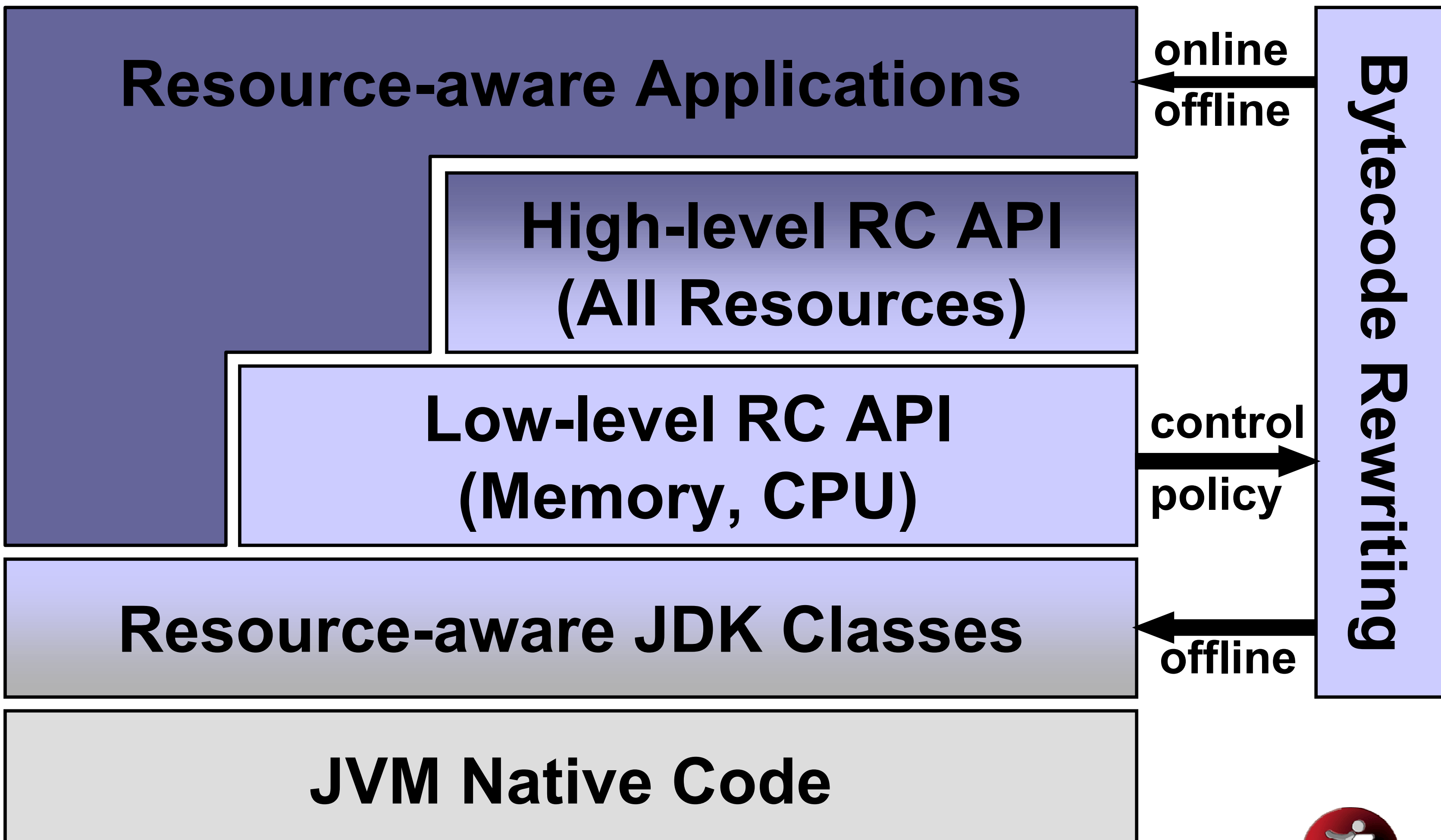
Different Approaches to Resource Control in Java

- Special JVM
  - KaffeOS (based on Kaffe VM)
  - Alta
  - Aroma
- Native code
  - JRes (CPU control)
  - JUM (Java Usage Monitor; JVMPI)
- Bytecode transformations
  - JRes (memory control)
  - J-RAF (Java Resource Accounting Facility)

J-RAF: Portable Resource Control Layer for Java

- Resource control
  - Resource accounting
  - Enforcing resource limits
- Resources
  - Physical (e.g., memory, CPU)
  - Logical (e.g., threads, tasks)
  - Communication (e.g., with services)
- Pure Java
  - No dependence on operating system
  - Standard JVMs (state-of-the-art JITs)
  - Java processors (e.g., embedded systems)

Layered Architecture for Resource Control



Functions of the Low-level Resource Control Layer

- **Memory accounting and control**
  - **Pre-accounting, enforcement of limits**
  - **Pluggable control strategy**
    - Single thread
    - Multi-threaded task (synchronization!)
    - Multiple tasks (consistency after termination!)
- **CPU accounting**
  - **Pre-accounting for each thread**
  - **Pre-accounting for estimated future GC costs during object allocation**

Functions of the High-level Resource Control Layer

- Integration with hierarchical task model
- Pluggable control strategies
  - Memory control policies
  - Scheduling for CPU control
- Uniform API for all resources
- Extensible API
- Charging for resource consumption

Reification of Physical Resources by Bytecode Rewriting

- Introduction of account objects
  - Associated with threads (thread-locals)
  - Passed as extra argument (optimization)
  - Wrapper methods with unmodified signature (called by native code)
- Memory accounting and control
  - Allocated bytes on heap (estimation)
  - Before allocation
- CPU accounting
  - Executed bytecode instructions
  - In every basic block of code

Memory Account (Simplified)

- Supports aggregation (single account for multiple threads/tasks)
- Weak references to allocated objects

```
public final class Mem {
   public static Mem getOrCreate();

   public void setLimit(long limit);
   public void checkAllocation(int size)
         throws ResourceOveruseException;
   public void register(Object o);
}
```

CPU Account

- Separate account for each thread
- No synchronization
- Accessed by high-priority scheduler thread (high-level RC layer)

```
public final class CPU {
   public static CPU getOrCreate();

   public volatile int usage;
}
```

Rewriting Example: Passing Account Objects

- **Unmodified method**

```
Object f(int x) {
   if (x < 0) return null;
   else return new Foo(g(x));
}
```

- **Added account objects**

```
Object f(int x, Mem mem, CPU cpu) {
   if (x < 0) return null;
   else return new Foo(g(x, mem, cpu),
                           mem, cpu);
}
```

Rewriting Example: Memory Accounting (Simplified)

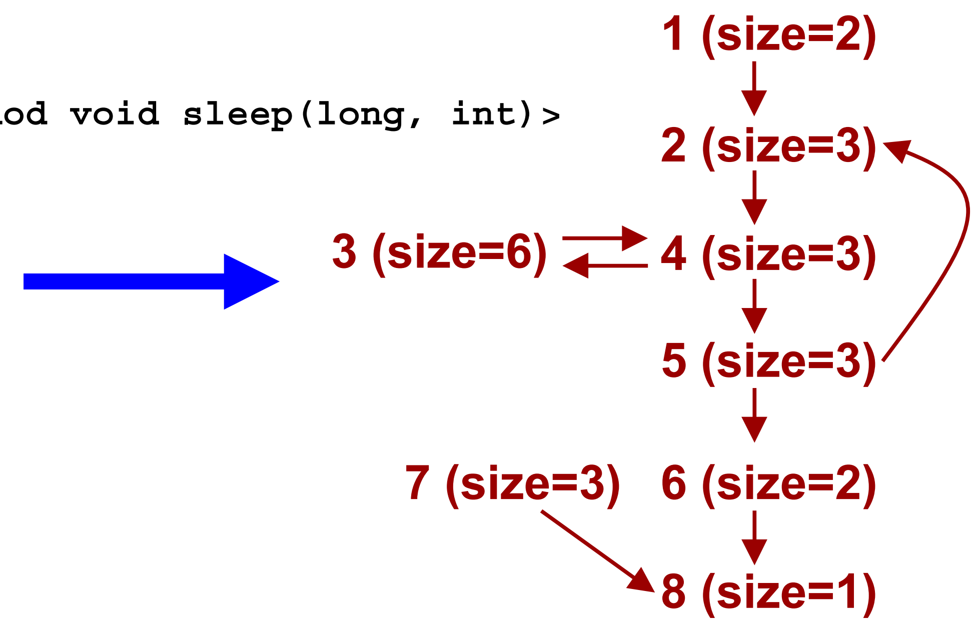
```
Object f(int x, Mem mem, CPU cpu) {
   if (x < 0) return null;
   else {
      int y = g(x, mem, cpu);
      mem.checkAllocation(SIZEOF_FOO);
      Object o = new Foo(y, mem, cpu);
      mem.register(o);
      return o;
   }
}
```

Rewriting Example: CPU Accounting

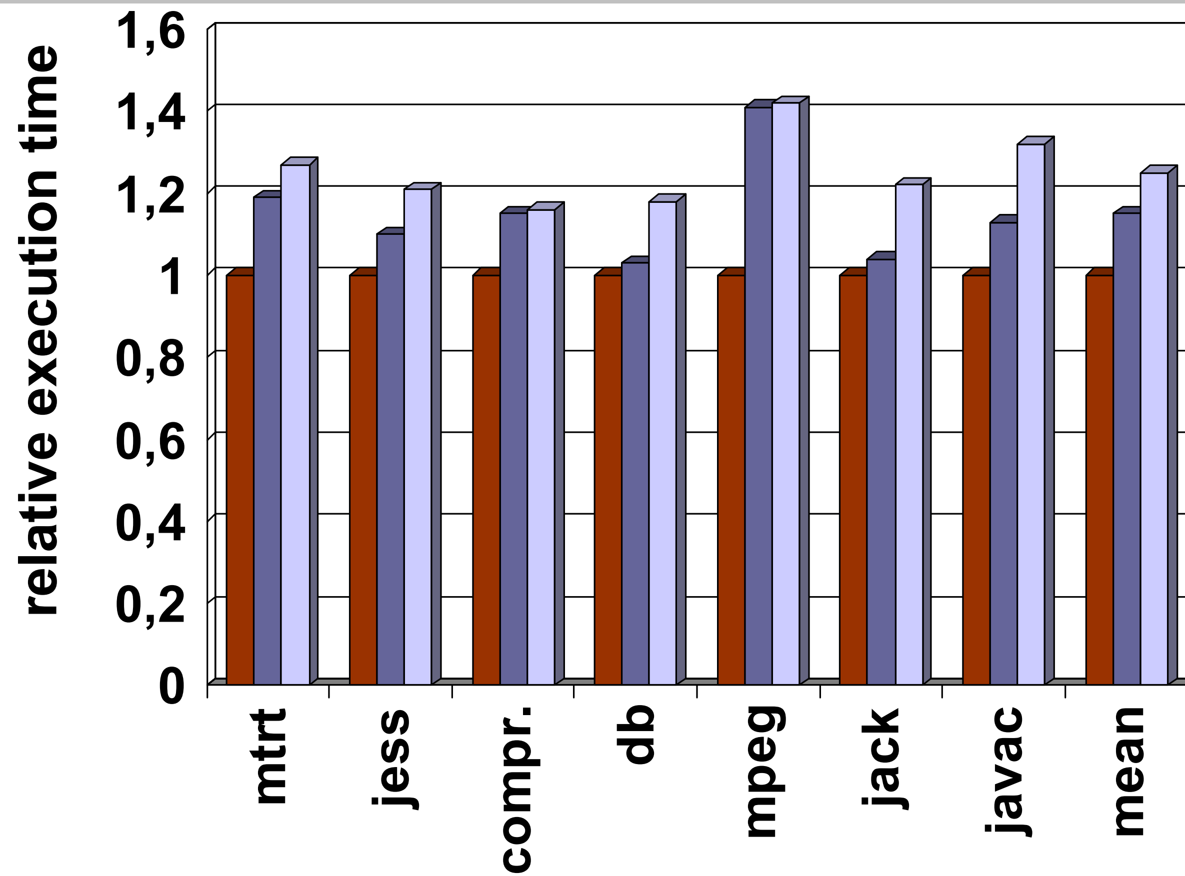
```
Object f(int x, Mem mem, CPU cpu) {
   cpu.usage += 8;
   if (x < 0) {
      cpu.usage += 8;
      return null;
   }
   else {
      cpu.usage += 26;
      int y = g(x, mem, cpu);
      mem.checkAllocation(SIZEOF_FOO);
      Object o = new Foo(y, mem, cpu);
      mem.register(o);
      return o;
   }
}
```

Accounting Block Analysis

```
1   0 bipush 10
    2 istore_1
2   3 iconst_1
    6 istore_3
    7 goto 25
3   10 invokestatic #2 <Method java.lang.Thread currentThread()>
    13 pop
    14 ldc2_w #3 <Long 100>
    17 bipush 100
    19 invokestatic #5 <Method void sleep(long, int)>
    22 iinc 3 1
4   25 iload_3
    26 iconst_1
    27 if_icmple 10
5   30 iload_2
    31 iconst_1
    32 if_icmpl1 3
6   35 iinc 2 -1
    38 goto 44
7   41 astore_2
    42 iconst_0
    43 istore 3
8   44 return
    Exception table:
      from   to   target   type
       3     38    41     <Class java.lang.Exception>
```



Overhead of CPU Accounting



Application JDK	Unmodified Unmodified	Rewritten Unmodified	Rewritten Rewritten
Mean	100%	115%	125%

Limitations

- Requires Java 2 platform
- No accounting for resources consumed by native code
- Restrictions (compensation for native code)
  - Reflection
  - Deserialization
  - Class loading
  - Object cloning

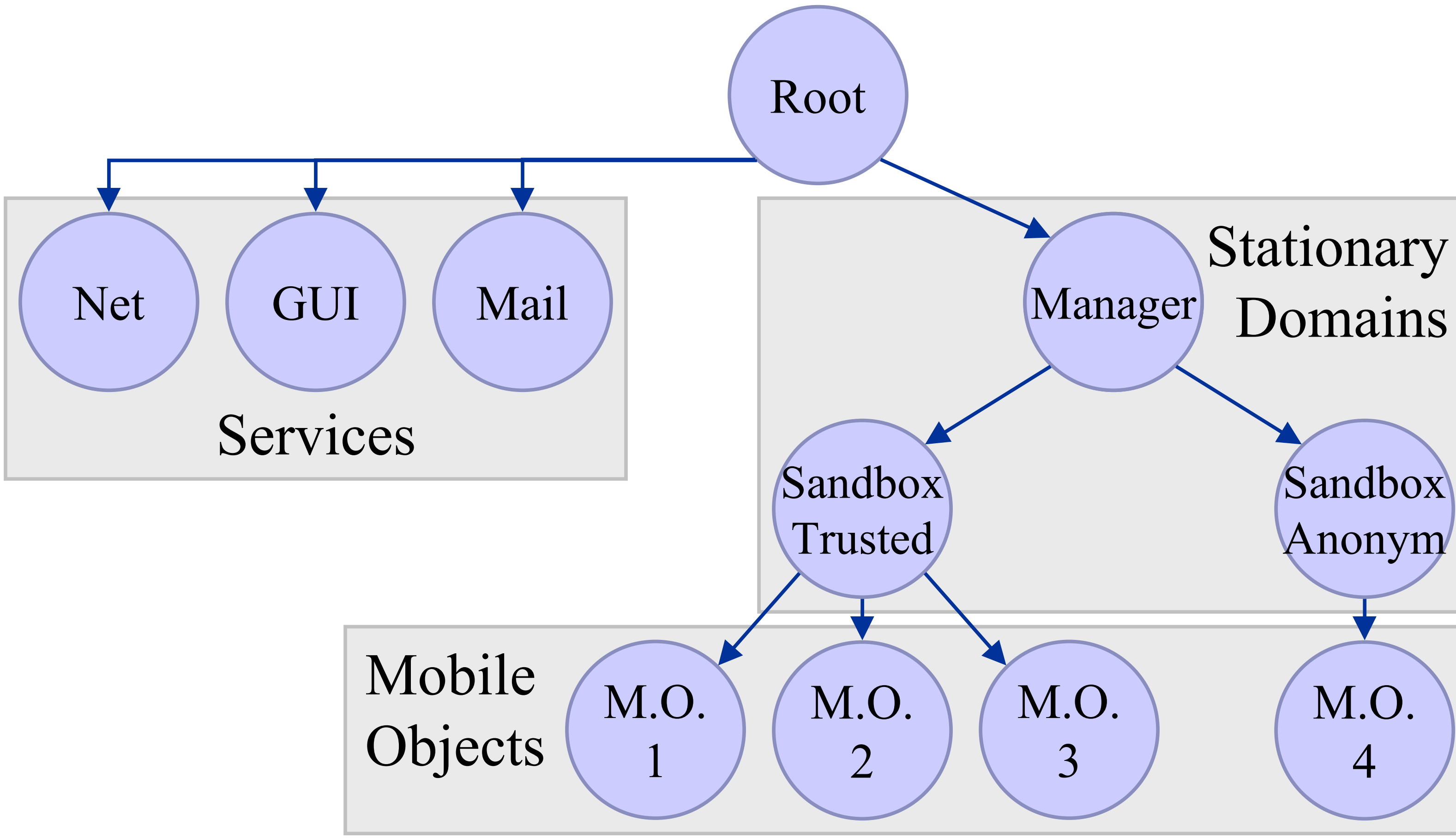
Case study: The J-SEAL2 Mobile Object Kernel

- Operating system structure
  - Isolated tasks
  - Safe task termination
  - Mediated communication
  - Resource control
- Small micro-kernel (< 150 KB)
- Portable (pure Java)
- Flexible and extensible
- Efficient and scalable

Nested Tasks in J-SEAL2

- Hierarchy of tasks
- Parent task controls children
  - Communication control
  - Resource control
  - Transformation control
  - Termination of sub-hierarchies
- Sandboxes with different privileges

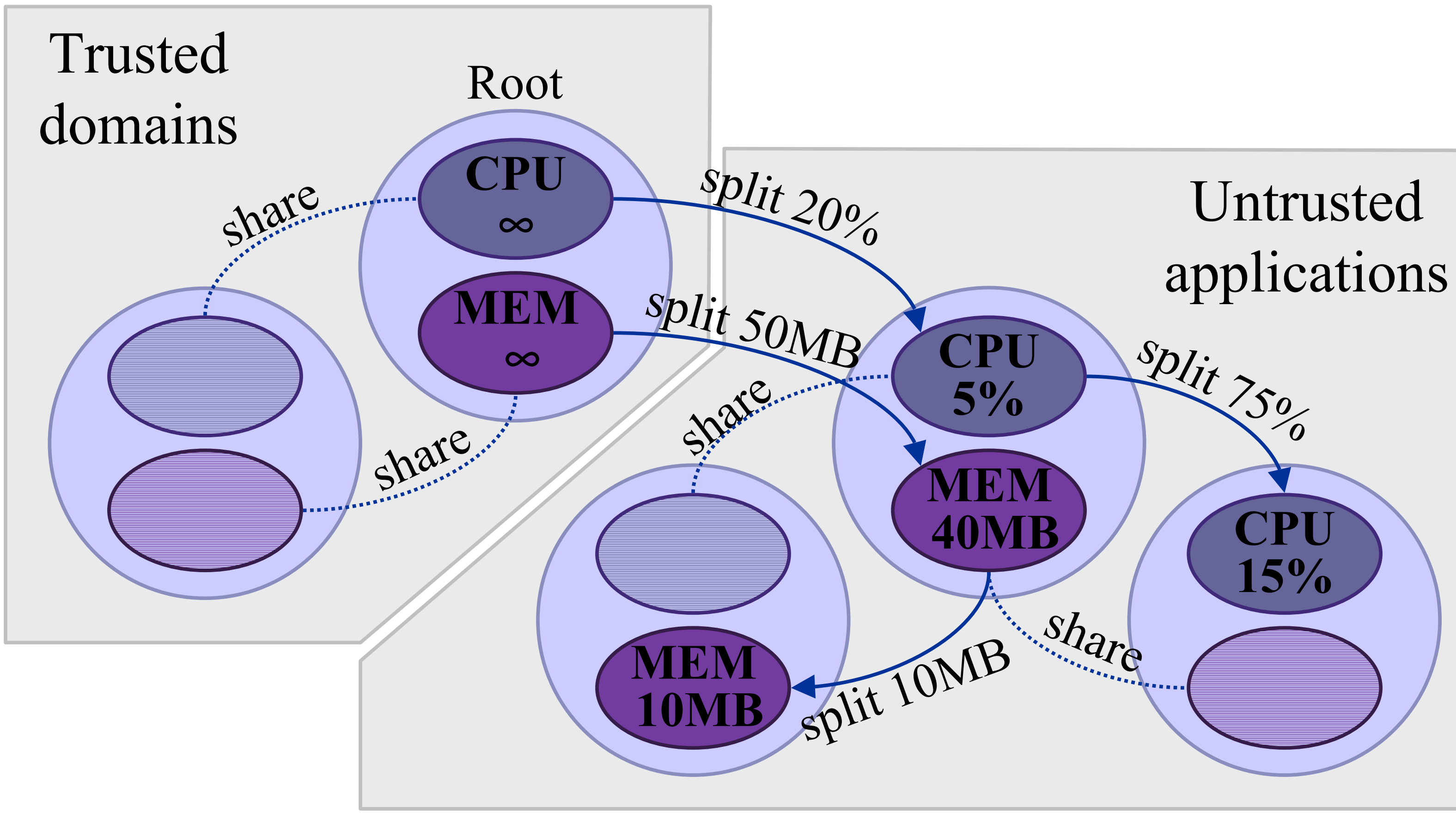
J-SEAL2 System Structure



Hierarchical Resource Control in J-SEAL2

- Startup
  - Root task owns all resources
- Sub-task creation
  - Parent may donate resources to child
  - Parent may share resources with child

Resource Donation and Sharing in J-SEAL2



Summary

- Resource control is crucial for host security, but missing in Java
- Resource control layer for Java
  - Portable (pure Java)
  - Reification of physical resources
  - Bytecode rewriting
  - Moderate overhead

Get More Information!

- J-SEAL2
  - URL: http://www.jseal2.com/
  - Email: w.binder@coco.co.at
- Java Resource Accounting Facility
  - Collaboration with TiOS (Uni Geneva)
  - URL: http://abone.unige.ch/