

RTSJ Into the Future

Morgan Deters
mdeters@cs.wustl.edu



www.cs.wustl.edu/~doc/

**Fall 2003 Seminar on
Programming Languages**

10 October 2003

What About Java??

Java for Real-Time?

- **Difficulty in accepting Java as a real-time platform**
 - **memory allocation**

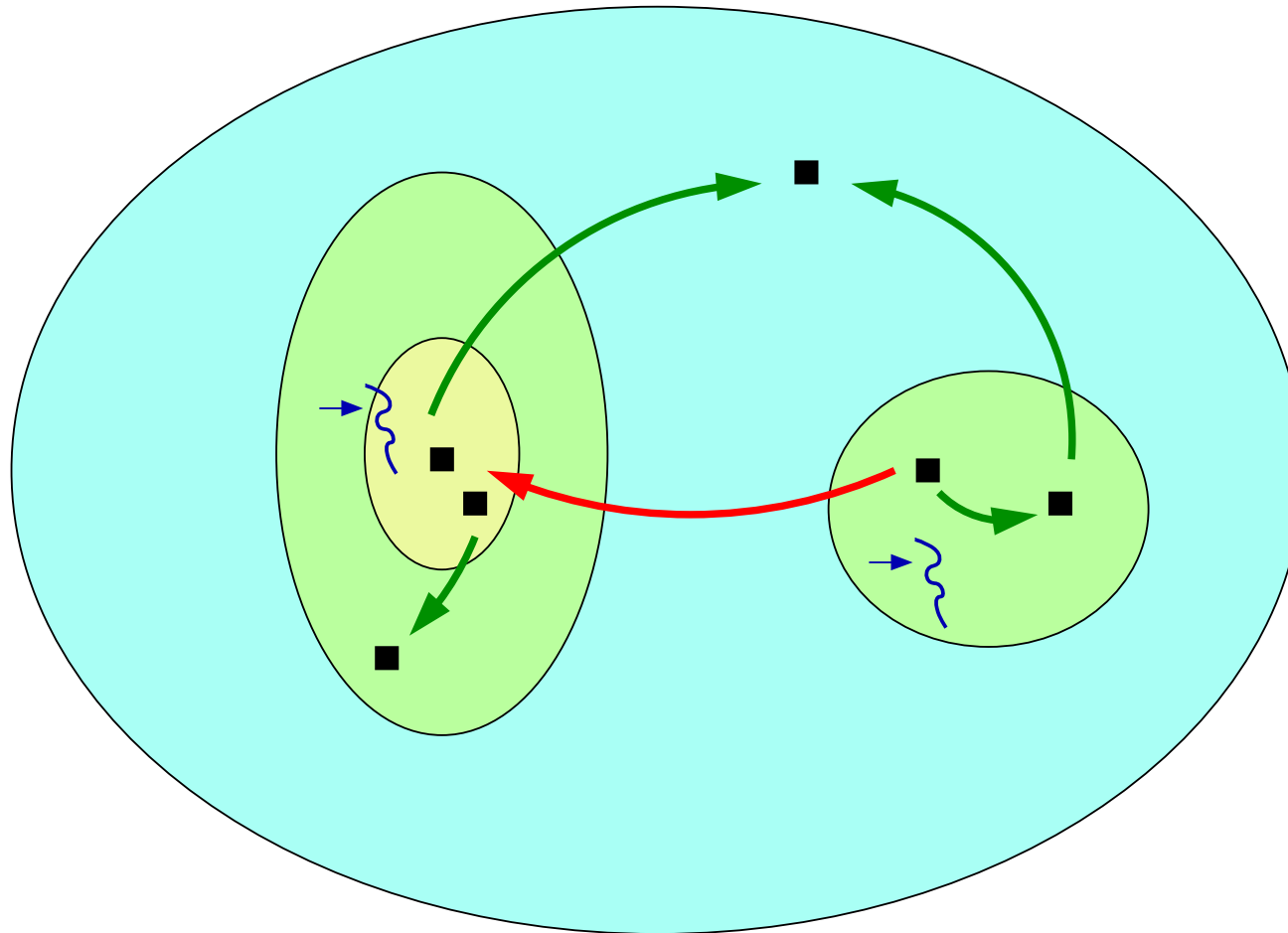
```
nuclearReactor.on();  
new Foo();  
nuclearReactor.off();
```
- ***Real-Time Specification for Java (RTSJ)* addresses real-time issues**
 - **Real-time schedulability**
 - **Asynchronous transfer of control**
 - **Non-garbage collected regions of memory**
 - ◇ **expert group didn't believe any suitable GC for real-time**
 - ★ **this was before Metronome [Bacon 2003], others**

RTSJ Memory Areas

- Associate memory areas with a particular scope of execution
- Objects allocated from the memory area are collected when the scope exits
- Scoped memory areas may be nested
- Objects within a memory area may not reference objects that are in shorter-lived memory areas

```
ScopedMemory lt = new LTMemory(1024, 1024); // 1 KiB
lt.enter(new Runnable() {
    public void run() {
        System.out.println("Hello from inside a scope!");
    }
});
System.out.println("Hello from outside a scope!");
```

A pretty picture of scopes

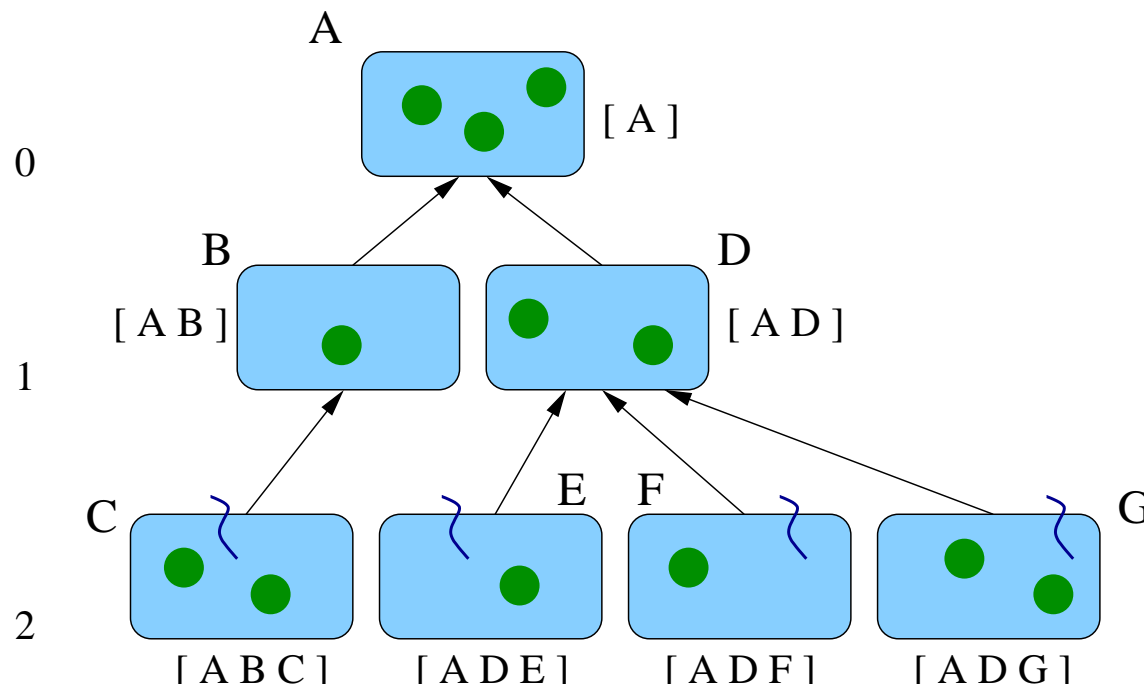


Implementing scopes

- **The RTSJ requires runtime checks**
 - **inter-object references**
 - **single parent rule**
- **These can't all be satisfied at compile-time...**
 - **...but some can [Sălcianu 2001]**
 - **...and they needn't be expensive [Corsaro 2003]**
 - **...or you could always scope-annotate your program such that compile-time checking of these annotations (and therefore your scopes) is possible [Boyapati 2003]**

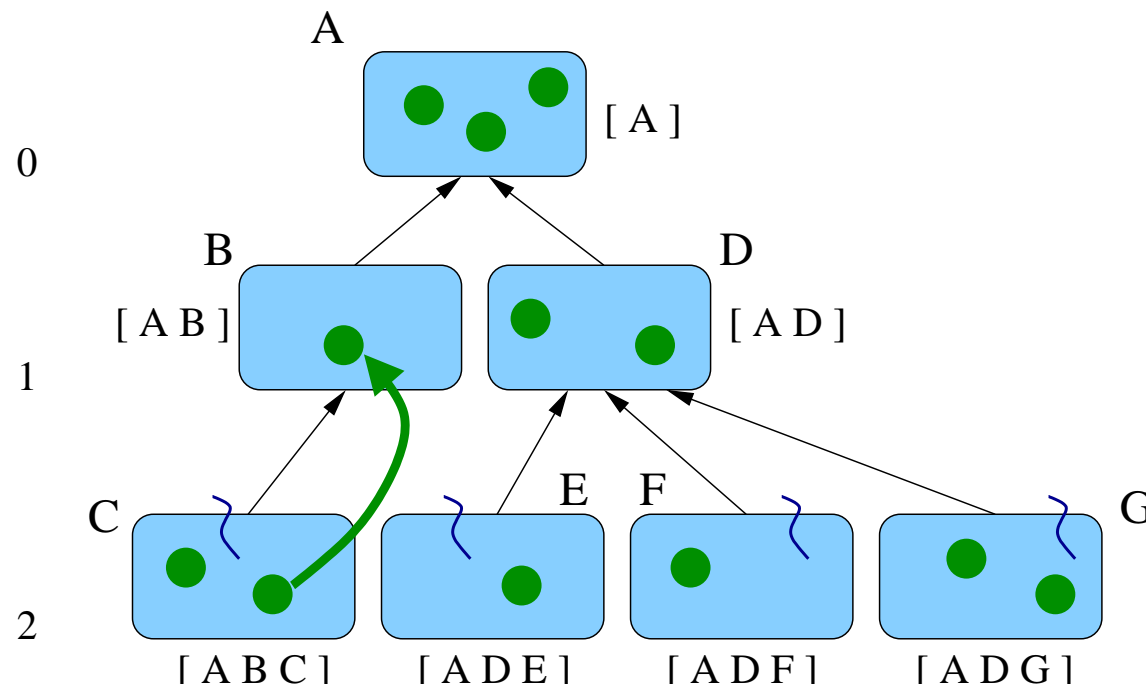
$O(1)$ runtime scope checks [Corsaro 2003]

- Use displays at nodes in the scope tree
 - Each scope has a depth and a display of ancestors



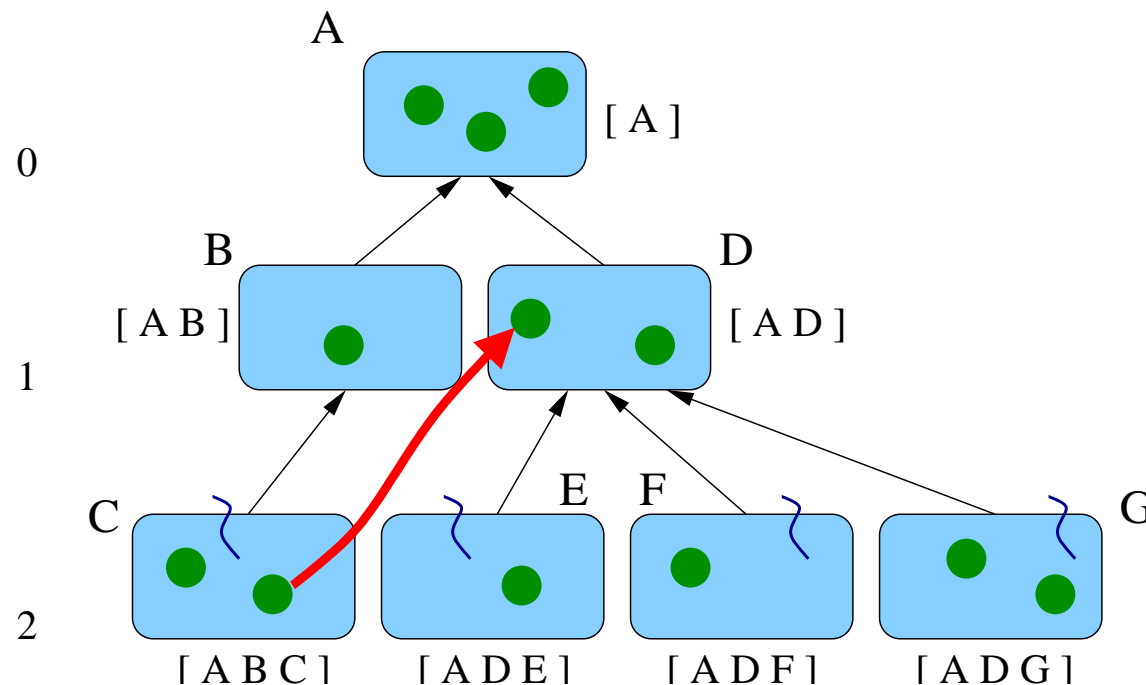
$O(1)$ runtime scope checks [Corsaro 2003]

- **(X ref Y):** `X.scope.depth >= Y.scope.depth && X.scope.display[Y.scope.depth] == Y`



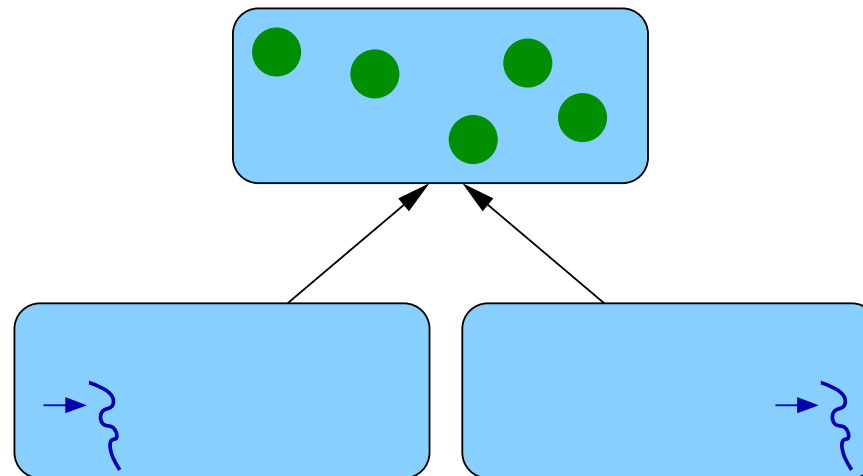
$O(1)$ runtime scope checks [Corsaro 2003]

- **(X ref Y):** `X.scope.depth >= Y.scope.depth && X.scope.display[Y.scope.depth] == Y`



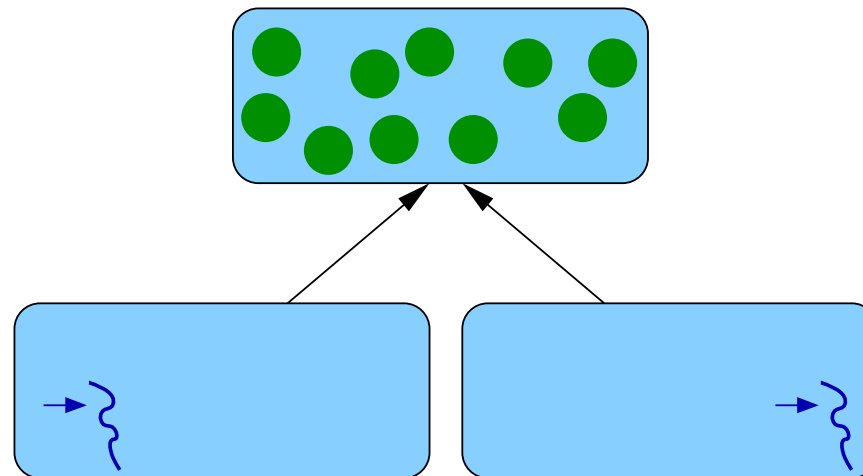
Design Problems with Scopes

- **Code is ultimately:**
 - **unwieldy**
 - **not maintainable**
 - **has its concerns all tangled up**
- **Manual memory management and cleanup**
 - **consider communication among threads**



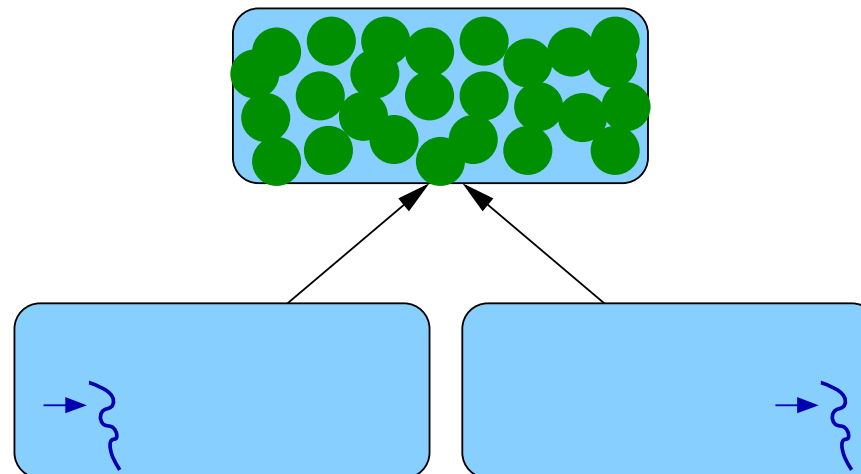
Design Problems with Scopes

- **Code is ultimately:**
 - unwieldy
 - not maintainable
 - has its concerns all tangled up
- **Manual memory management and cleanup**
 - consider communication among threads



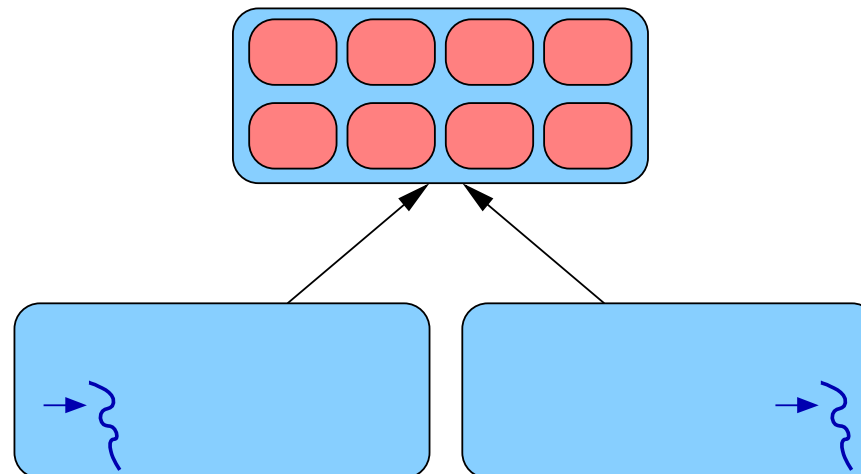
Design Problems with Scopes

- **Code is ultimately:**
 - unwieldy
 - not maintainable
 - has its concerns all tangled up
- **Manual memory management and cleanup**
 - consider communication among threads



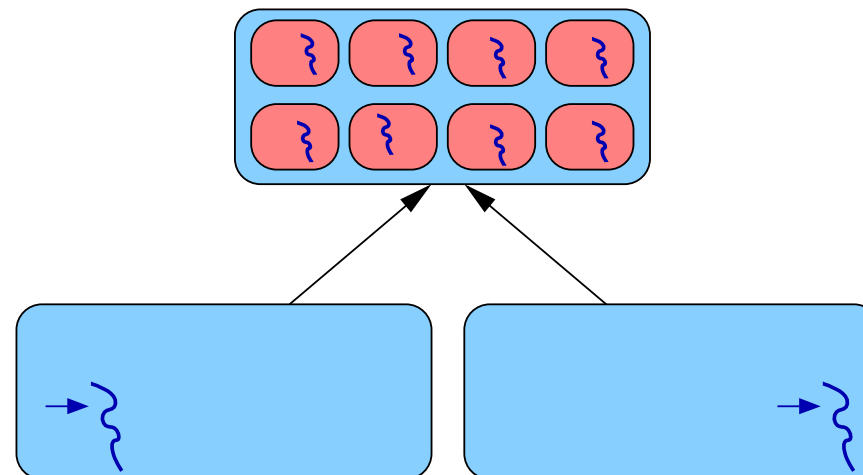
Design Problems with Scopes

- **Code is ultimately:**
 - unwieldy
 - not maintainable
 - has its concerns all tangled up
- **Manual memory management and cleanup**
 - consider communication among threads



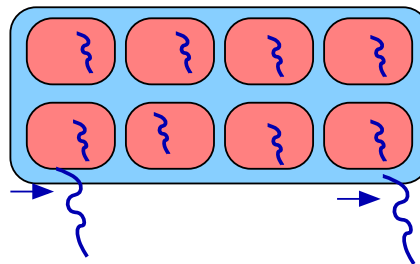
Design Problems with Scopes

- **Code is ultimately:**
 - unwieldy
 - not maintainable
 - has its concerns all tangled up
- **Manual memory management and cleanup**
 - consider communication among threads



Design Problems with Scopes

- **Code is ultimately:**
 - unwieldy
 - not maintainable
 - has its concerns all tangled up
- **Manual memory management and cleanup**
 - consider communication among threads



Design Problems with Scopes (There's More)

- **What scopes are return values in?**
 - **analogous to the problem of allocation of return values in C**
 1. **`malloc()`, then `return` (caller calls `free()`)**
 2. **pass-a-pointer (to a stack-allocated object)**
 3. **use a `static` variable in the function (or maybe TSS)**
 - **what are our options with RTSJ scopes?**
 1. **`new`, then `return`**
 - ◇ **might be allocated in a scope that's too short-lived!**
 2. **pass-a-scope**
 - ◇ **maybe... but too restrictive in general**
 3. **call with a properly-scoped object**
 - ◇ **requires object re-initialization, and breaks subclassing**

Design Problems with Scopes (Oh Yes, Even More!)

- **What about third-party libraries?**
 - **common to link libraries together, manages sharing of objects between them**
 - **now libraries have to know each others' memory requirements?!**

References

- [**Bacon 2003**] Bacon, David F., Perry Cheng, and V.T. Rajan. A real-time garbage collector with low overhead and consistent utilization. In POPL 2003.
- [**Boyapati 2003**] Boyapati, Chandrasekhar, Alexandru Sălcianu, William Beebee, Jr., and Martin Rinard. Ownership types for safe region-based memory management in real-time Java. In PLDI 2003.
- [**Corsaro 2003**] Corsaro, Angelo and Ron K. Cytron. Efficient memory-reference checks for real-time Java. In LCTES 2003.
- [**Sălcianu 2001**] Sălcianu, Alexandru and Martin C. Rinard. Pointer and escape analysis for multithreaded programs. In PPOPP 2001.

Discussion

www.cs.wustl.edu/~doc/



Washington
University in St. Louis

**Fall 2003 Seminar on
Programming Languages**

10 October 2003