

On the connection between functional programming languages and RTSJ

Delvin C. Defoe
Rose-Hulman Inst. of Technology
5500 Wabash Ave
Terre Haute, IN

Rob Legrand, and Ron K Cytron
Washington University in St Louis
1 Brookings Dr
Saint Louis, MO

Acknowledgements

- ▶ Research funded by
 - DARPA under contract F33615-00-C-1697 and
 - The Chancellor's Graduate Fellowship Program at Washington University

Outline

Part I

Memory management options

Part II

Cost analysis for Real-time scopes

Part III

Connection with RTSJ and pure functional programming

Part IV

Reclaiming Scopes

Memory management options

- ▶ Manual or explicit
[Kelley & Pohl]
- ▶ Garbage collection (GC)
[Collins, McCarthy]
- ▶ Real-time garbage collection
[Cheng et al., Bacon et al.]
- ▶ Real-time Specification for Java scoped-memory areas
[Bollella et al.]

Memory management options

- **Manual or explicit**
[Kelley & Pohl]
- **Garbage collection (gc)**
[Collins, McCarthy]
- **Real-time GC**
[Cheng et al., Bacon et al.]
- **RTSJ scoped memory**
[Bollella et al.]

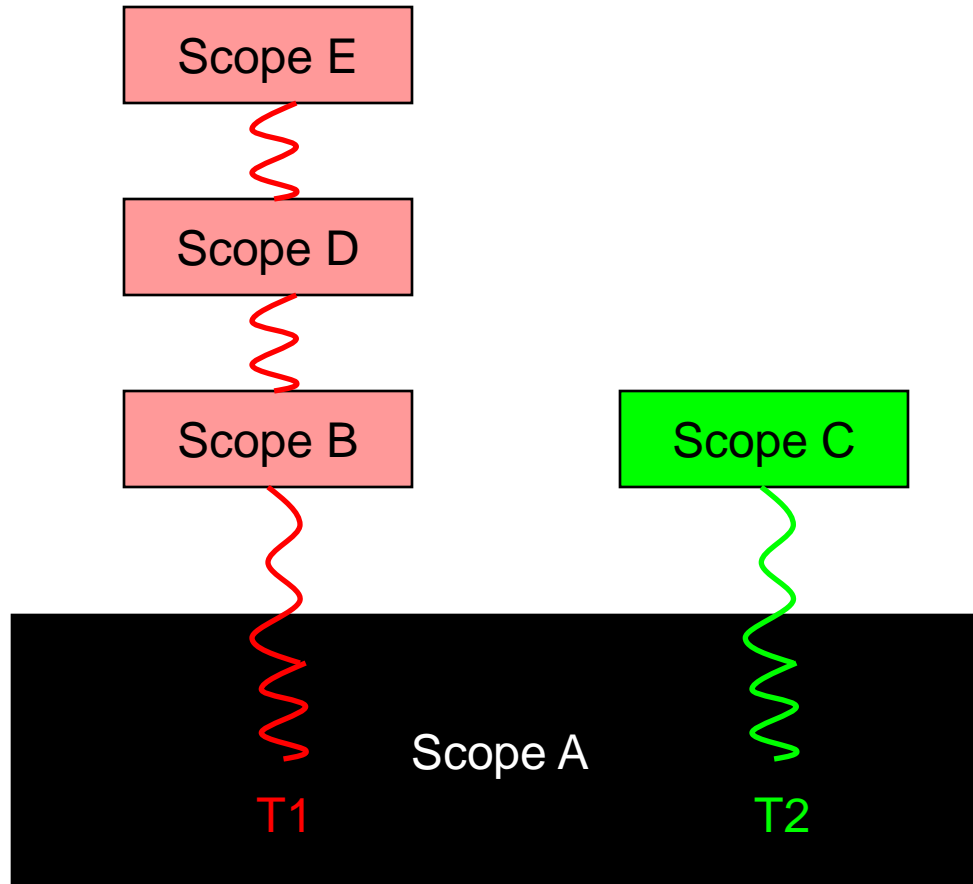
- **Semi-manual option**
 - **Scopes: regions of memory**
 - **Scopes: limited life times**
 - **Threads allocate from current scope**
 - **Predictable allocation**
 - **Predictable deallocation**
 - **No dangling pointers**

Memory management options

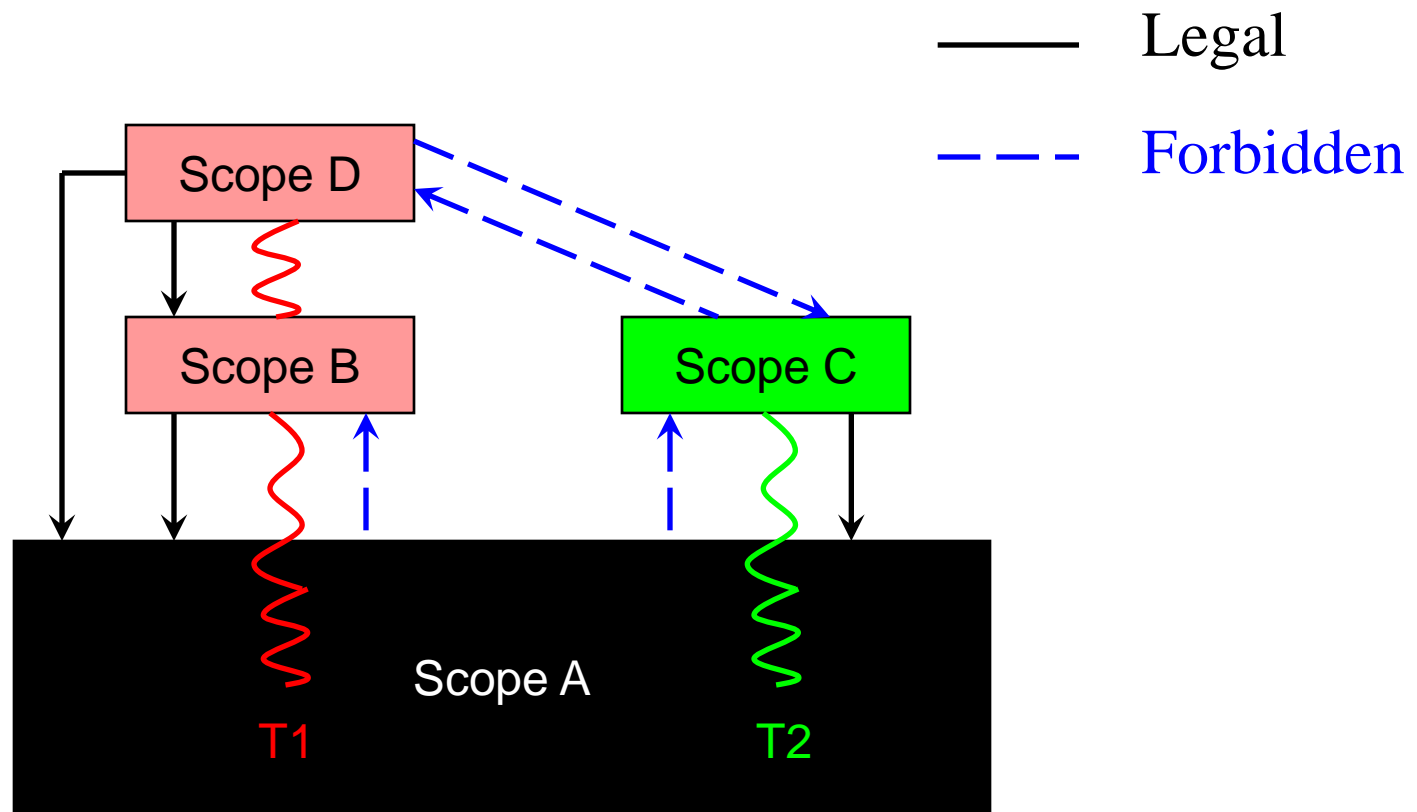
- **Manual or explicit**
[Kelley & Pohl]
- **Garbage collection (gc)**
[Collins, McCarthy]
- **Real-time GC**
[Cheng et al., Bacon et al.]
- **RTSJ scoped memory**
[Bollella et al.]

```
ScopedMemory
  scope = new ScopedMemory(1024);
scope.enter(new Runnable() {
  public void run() {
    // do some stuff
    someObj o = new someObj();
    // do some more stuff
    someObj s = new someObj();
  }
});
// scope is collected
```

Overview of RTSJ Scopes



Legal inter-scope references



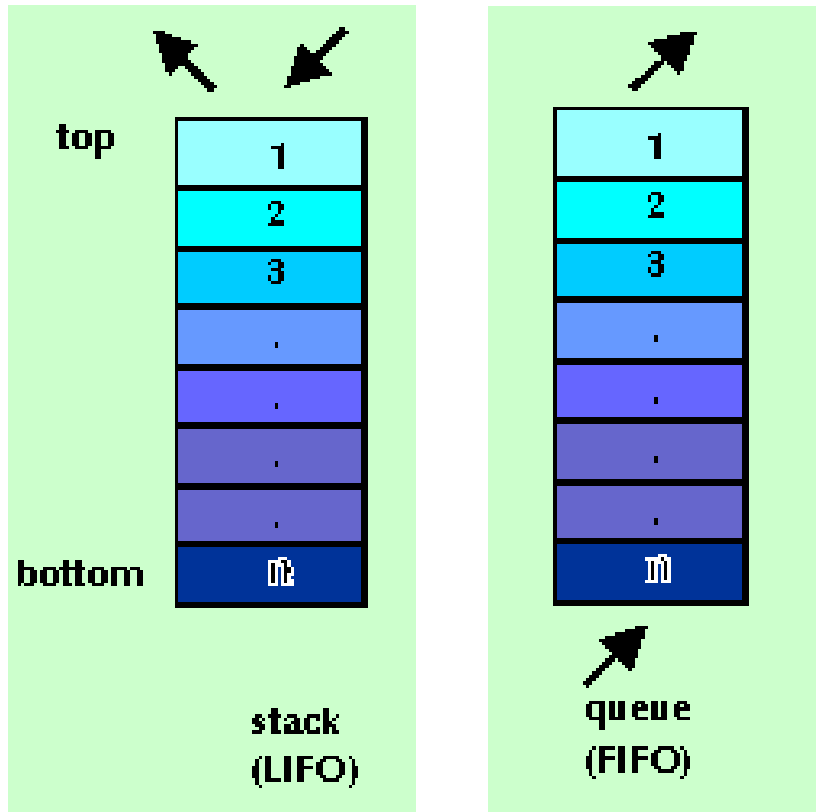
Can scopes be implemented?

- ▶ Existing implementations for RTSJ
 - TimeSys [TimeSys Corporation, 2002]
 - jRate [Corsaro et al., 2002]
 - Sun's Java Real-time System
- ▶ No independent cost analysis for scopes

How do we evaluate scopes?

- ▶ Is a large implementation necessary?
 - Execute it on particular platform
 - Do empirical studies
- ▶ Do we use asymptotic methods?
 - Objective
 - System independent

Asymptotic analysis of scopes



▶ Uses of stack

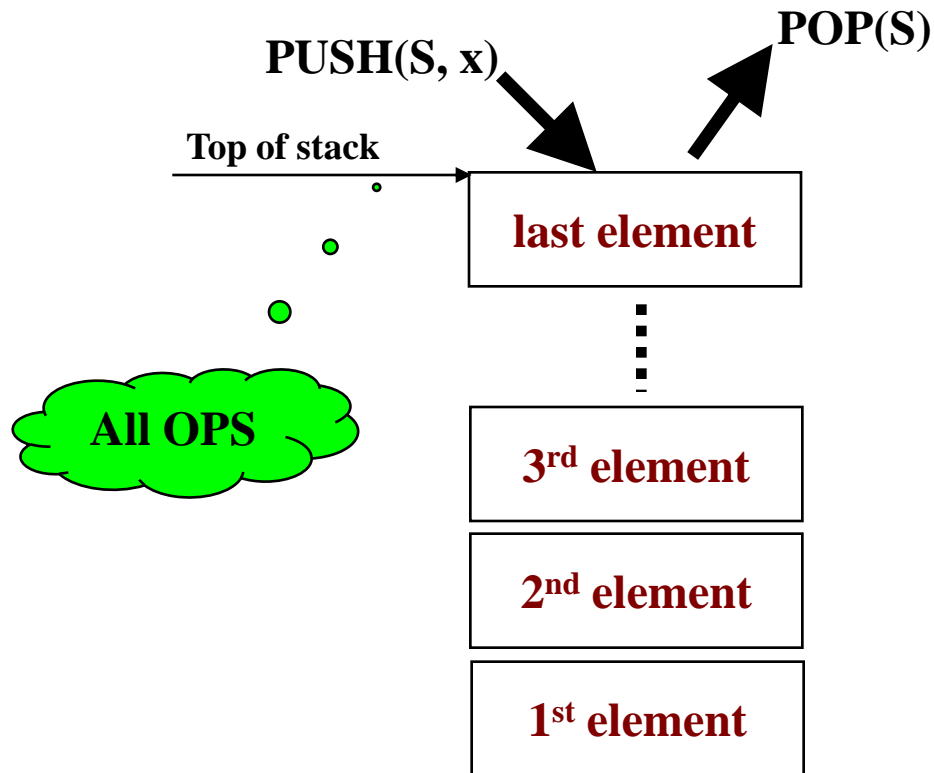
- Expression evaluation
- Syntax parsing
- Solving search problems

▶ Uses of queue

- Scheduling
- Buffering

Scope: cost using stack ADT

LIFO ADT

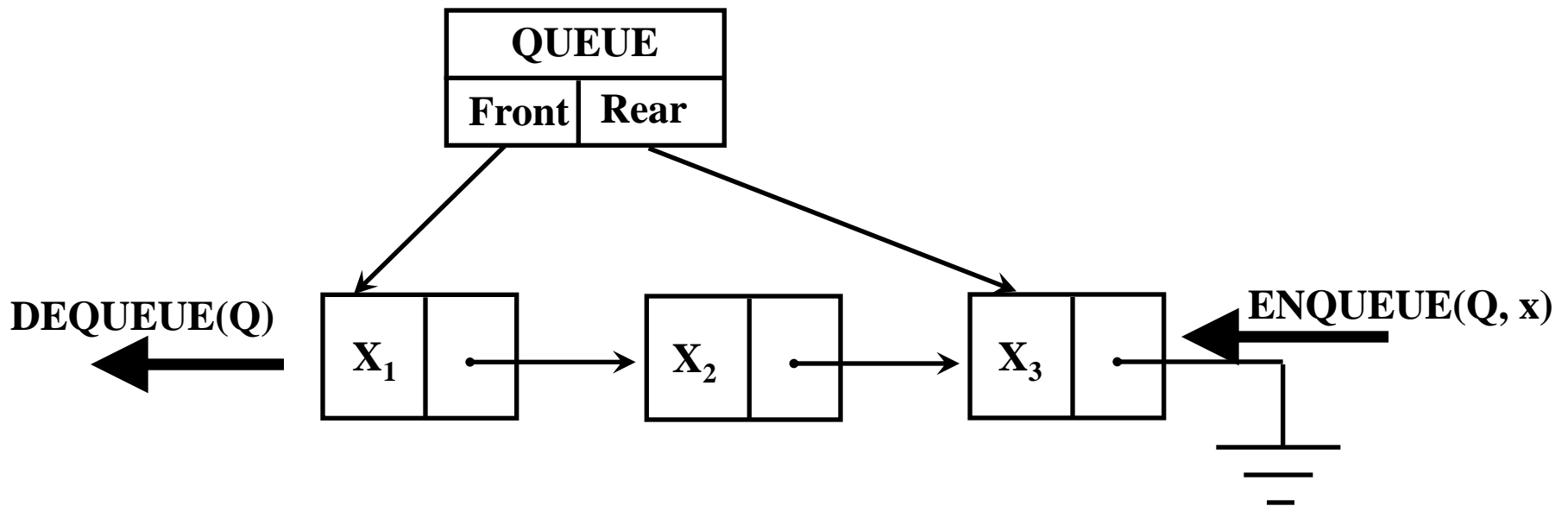


Findings: cost using stack ADT

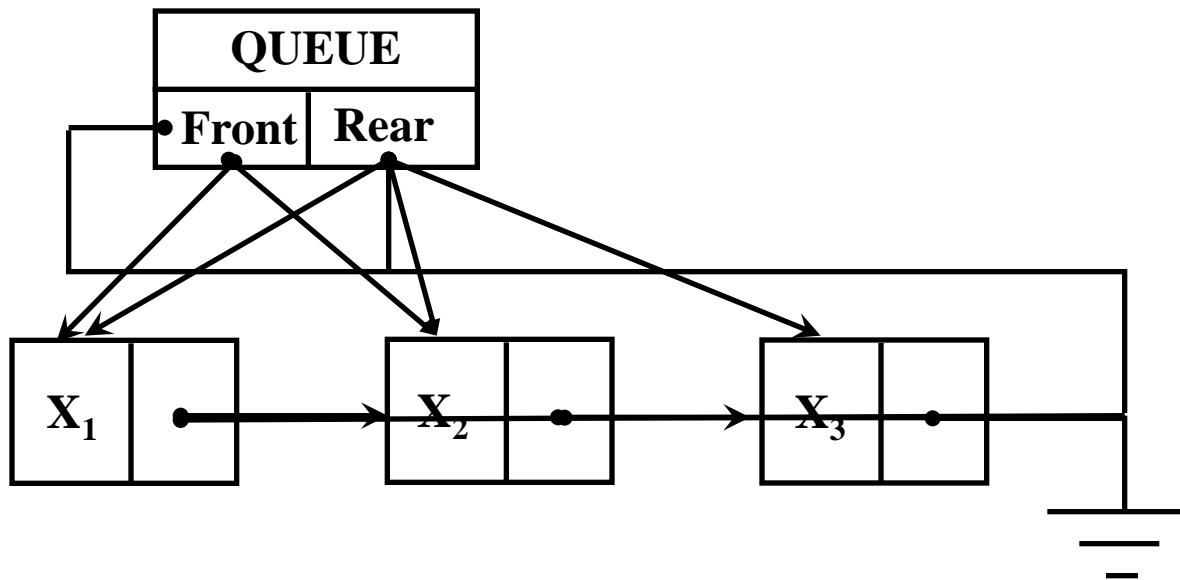
Ops	Java	RTSJ scopes
IS-EMPTY	$O(1)$	$O(1)$
PUSH	$O(1)$	$O(1)$
POP	$O(1)$	$O(1)$
n Ops	$O(n)$	$O(n)$

Scope: cost using queue ADT

FIFO ADT



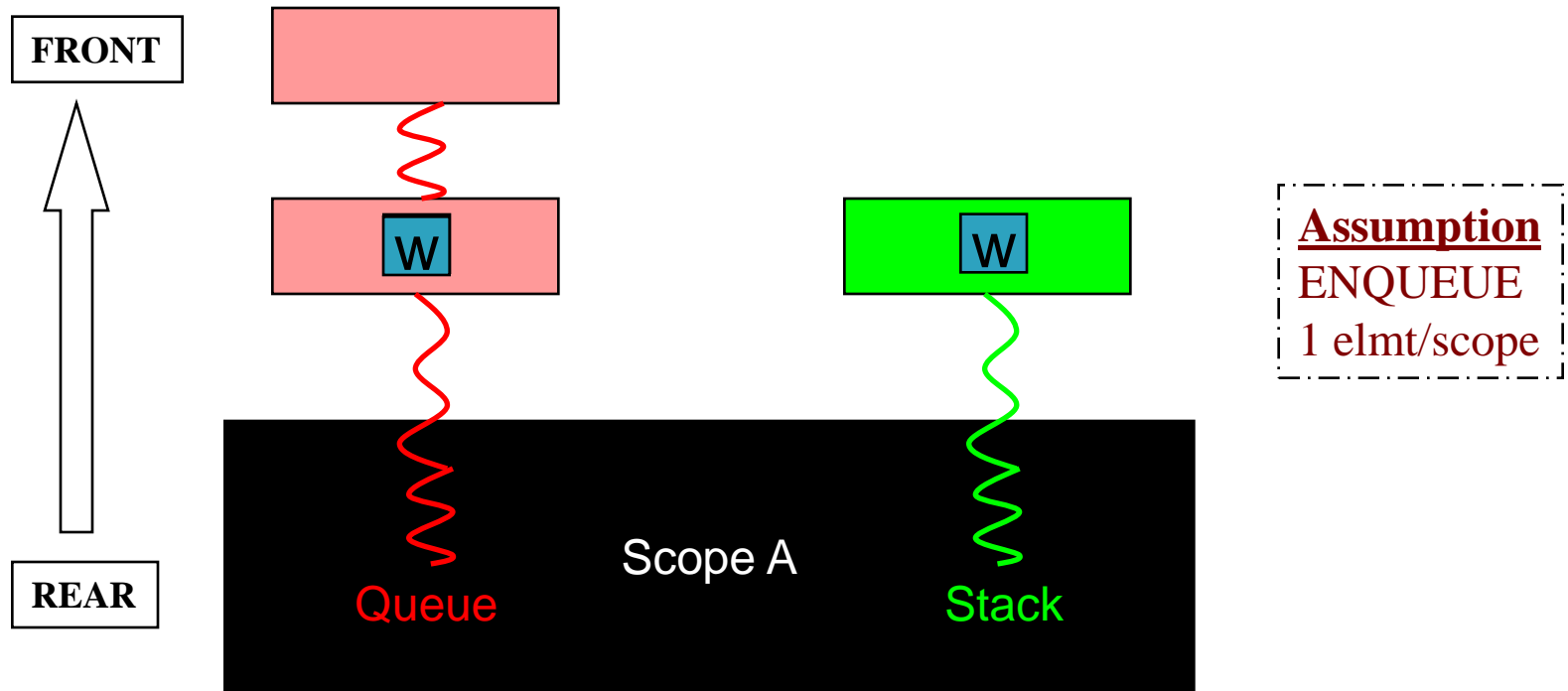
Queue: Java implementation



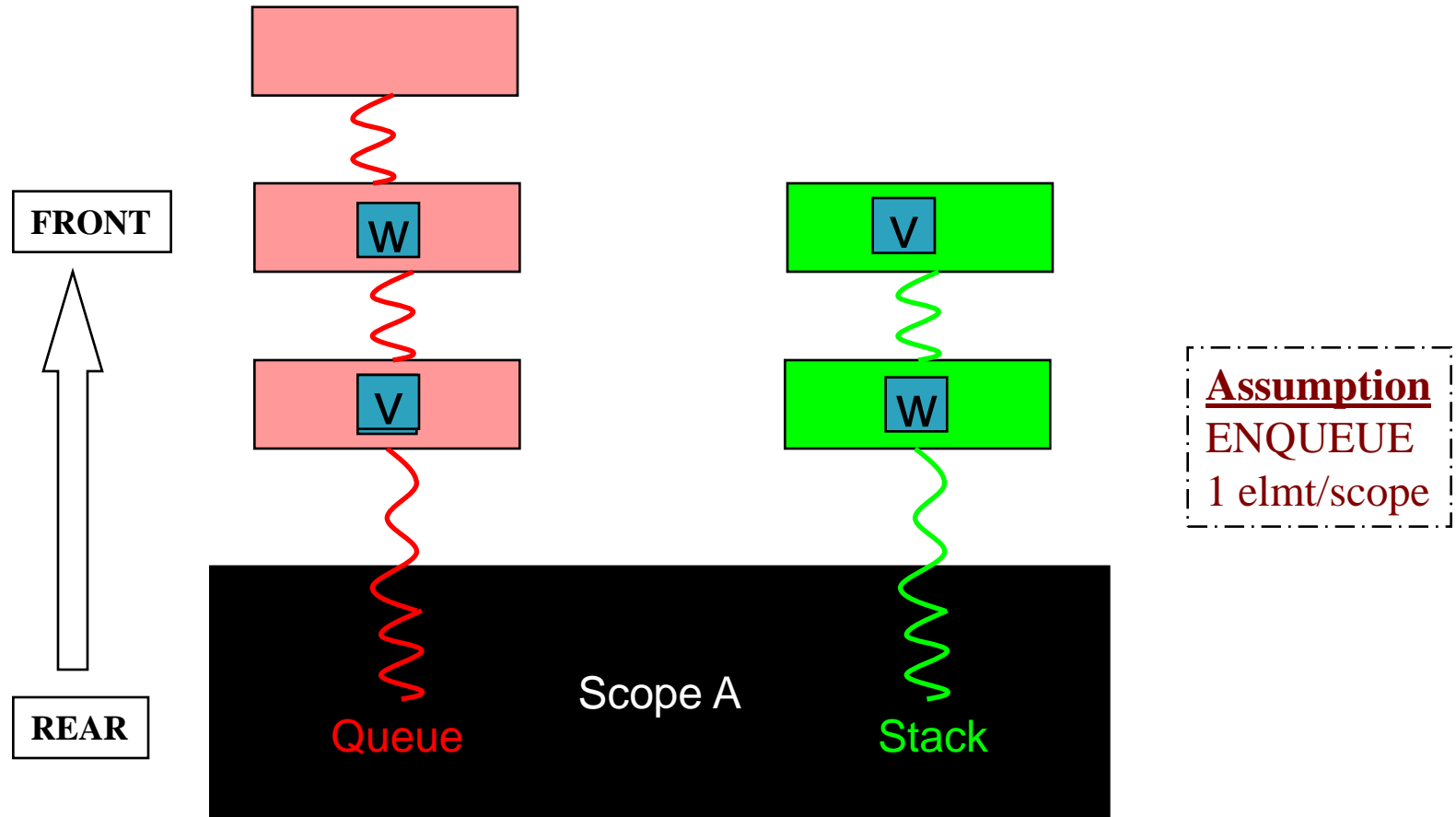
Scope: cost using queue ADT

Ops	Java	RTSJ scopes
ISQ-EMPTY	$O(1)$???
ENQUEUE	$O(1)$???
DEQUEUE	$O(1)$???
n Ops	$O(n)$???

Queue: scope implementation



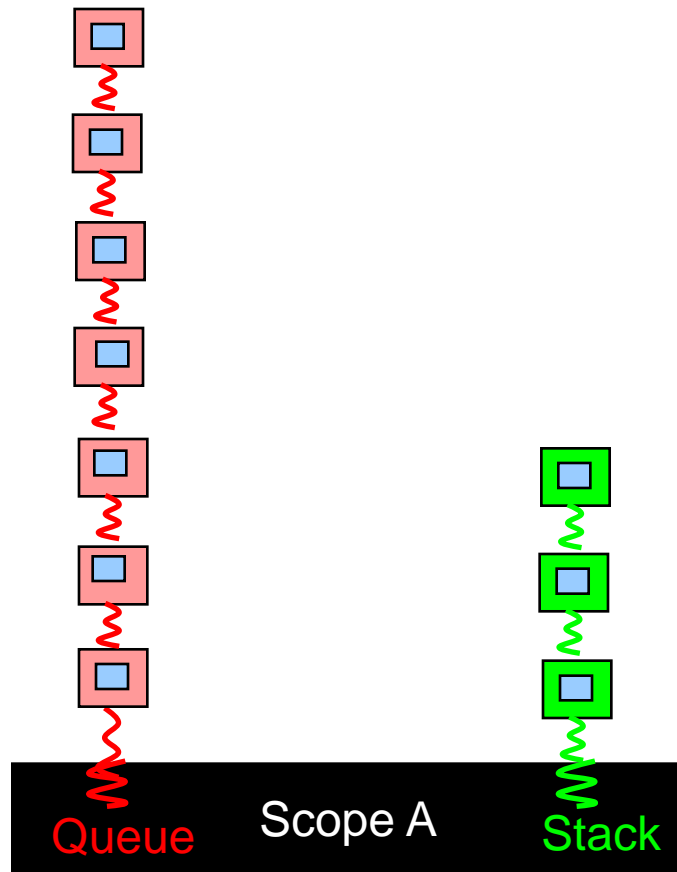
Queue: scope implementation



Scope: cost using queue ADT

Ops	Java	RTSJ scopes
ISQ-EMPTY	$O(1)$	$O(1)$
ENQUEUE	$O(1)$	$O(n)$
DEQUEUE	$O(1)$	$O(1)$
n Ops	$O(n)$	$O(n^2)$

Queue: scope implementation (2)



- ▶ Reduce complexity by doubling # of scopes
- ▶ Need $2*n$ scopes to store n objects – space–time tradeoff
- ▶ Enqueue op that requires new scope takes $O(n)$ time
- ▶ All other objects enqueued in $O(1)$ time

Scope: cost using queue ADT

Ops	Java	RTSJ scopes
ISQ-EMPTY	$O(1)$	$O(1)$
ENQUEUE	$O(1)$	$O(n)$
DEQUEUE	$O(1)$	$O(1)$
n Ops	$O(n)$	amortized $O(n)$

RTSJ & pure functional programs

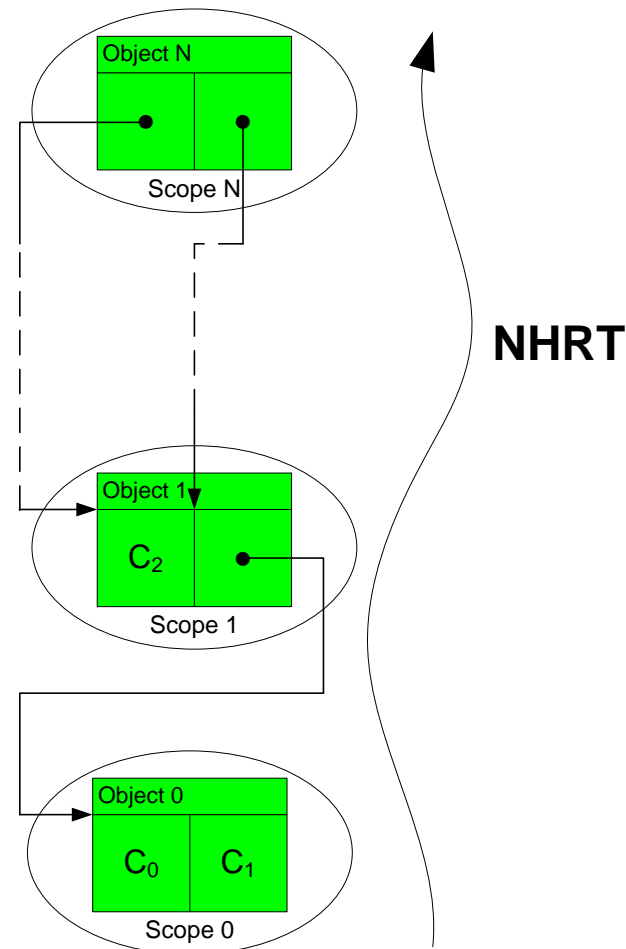
- ▶ New and interesting relationship between the two
 - **Mathematical transparency:**
Named expressions can only ref names strictly older than assigned name
 - Scopes can only ref ancestor scopes

From pure functional programs to RTSJ

- ▶ RTSJ developers migrate implementations from functional languages to RTSJ
 - RTSJ program creates a NHRT
 - Simulate *cons* cell by creating and entering new scope
 - Populate scope with precisely references for cons cell
- ▶ Runtime analyses follow

Pure functional program in RTSJ

- **RTSJ simulation of pure Lisp program**
 - **Scopes are exactly sized (cons cell)**
 - **By construction program is scope-safe**
 - **Lists and Heaps give good complexity**
[Okasaki, 1996]
 - **Scopes never exited**



Scope: cost using List ADT

Ops	Java	RTSJ scopes
INSERT	$O(1)$	$O(1)$
DELETE-ITEM	$O(1)$	$O(1)$
LOOKUP	$O(n)$	$O(\log n)$
n Ops	$O(mn)$	$O(m \log n)$

Scope: cost using Heap ADT

Ops	Java	RTSJ scopes
INSERT	$O(\log n)$	$O(\log n)$
FIND-MIN	$O(\log n)$	$O(\log n)$
DELETE-MIN	$O(\log n)$	$O(\log n)$
n Ops	$O(m \log n)$	$O(m \log n)$

Reclaiming Scopes

- ▶ Using the connection scopes are never exited
 - Not reclaimed
- ▶ Use ideas from garbage collection
 - Copying garbage collection
 - Reference counting can identify scopes no longer in use
 - Contaminated garbage collection
 - Tree of scopes instead of chain of scopes

Conclusions

- ▶ Have shown that there are hidden costs associated with scopes
- ▶ Presented interesting connection between functional program and RTSJ program

Future work

- ▶ Implement data structure packages in RTSJ that give runtime complexities similar to typical implementations
- ▶ Explore memory management issues associated with data structure implementation

Thank you

Questions?