Safety-Critical Java for Low-End Embedded Platforms

Stephan E. Korsholm & Hans Søndergaard
VIA University College, Horsens, DK

Anders P. Ravn
CISS, Aalborg University, DK

JTRES October 2012

The Problem

- Low-End Industrial Platforms
  - KT4585 from Polycom
  - ATMega2560 from AVR
  - NEC-V850 e.g. used by Grundfos
  - Typical memory resources
    - 16 kB RAM, 256 kB ROM

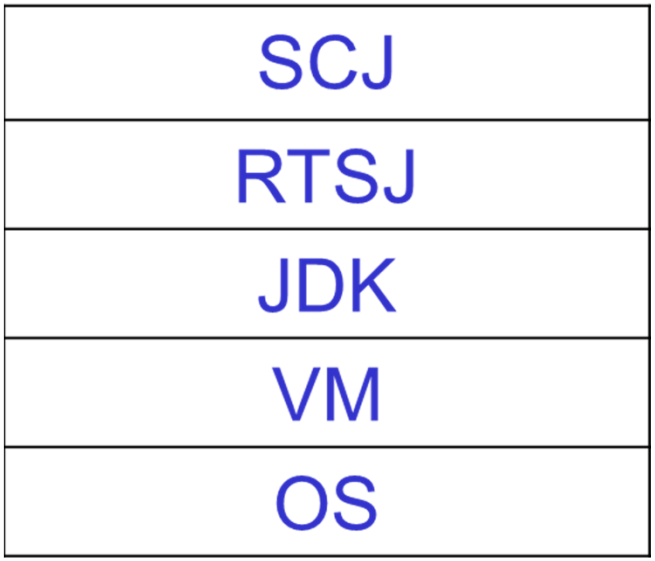
- Safety-Critical Java impl. using RTSJ
  - Based on Java RTS (SUN)
  - Recommended Requirements
    - CPU system with 512 MB
    - Real-Time OS: Linux

Plan to Solve the Problem

VIA UNIVERSITY COLLEGE

Centre for Embedded Software Systems

Reduce each layer of the architecture

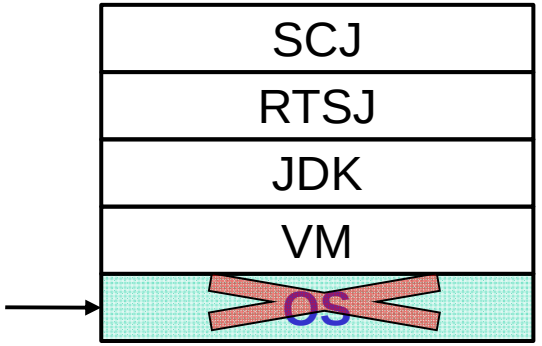


Operating System

No Operating System

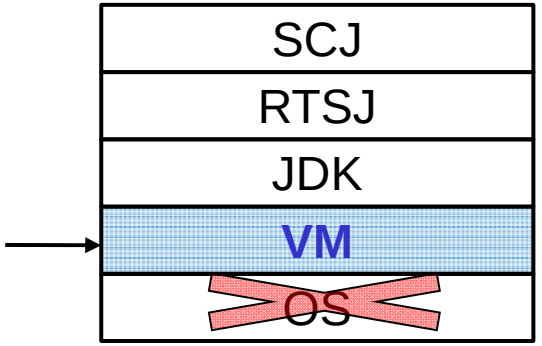
Instead:

- Hardware Objects for device control
- 1st level interrupt handling in Java space
- Minimal native layer for context switch between tasks



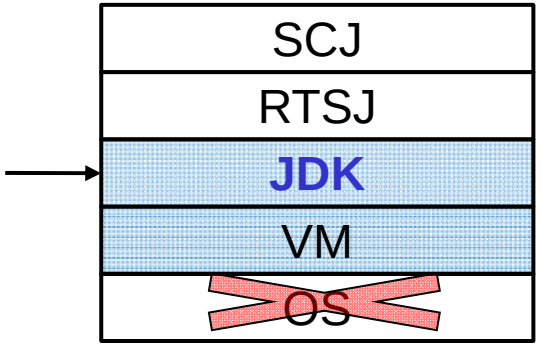
VM:
Hardware near Virtual Machine (HVM)

- Lean
  - Java-to-C compiler with embedded interpreter
  - Program specialization
    - Classes & methods
    - Bytecode selection
- No dependencies on external libraries
- Portable
  - Strict ANSI-C
  - All usual C compilers can be used
  - Simple build procedure



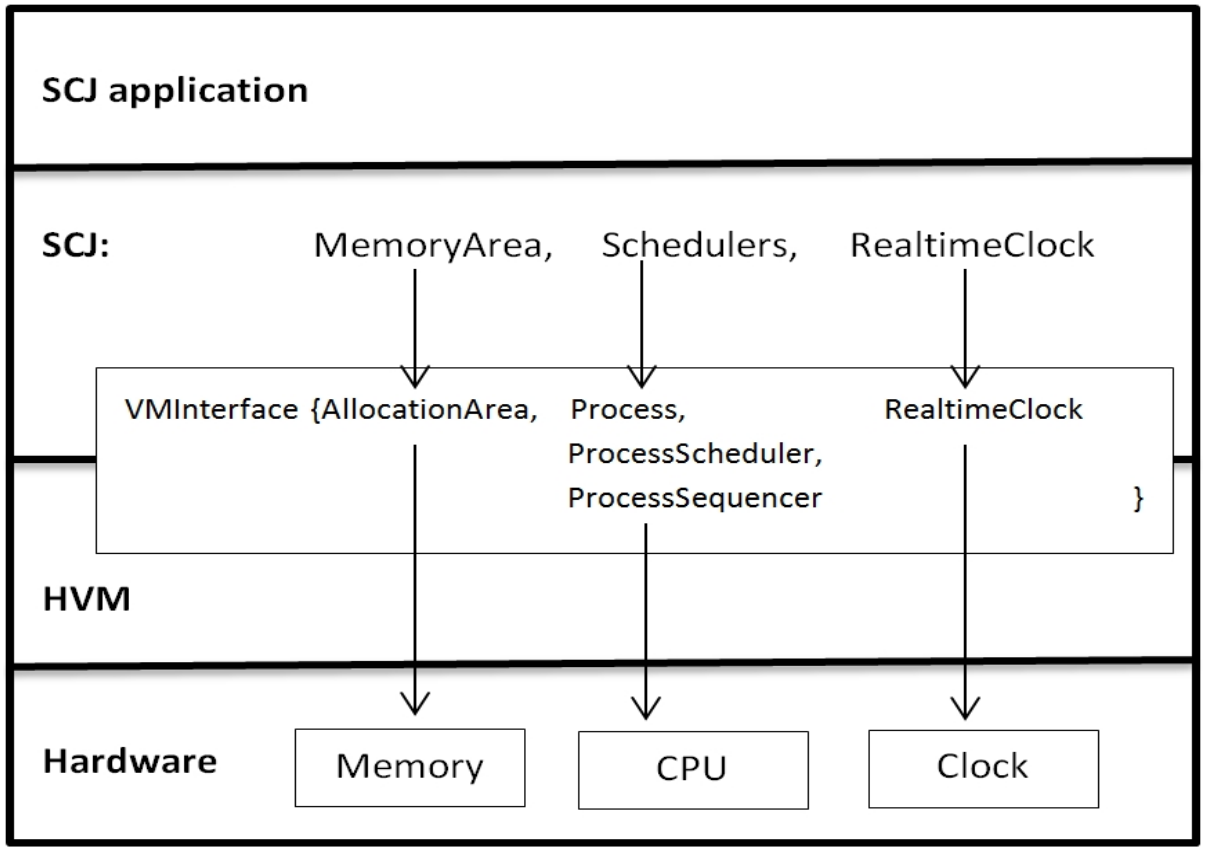
JDK

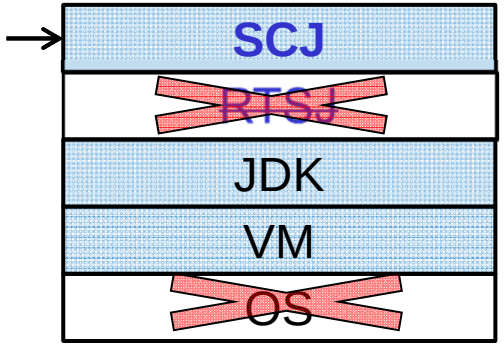
- No special JDK required
  - Uses Java 1.6 (Other JDKs supported as well)
  - Reduced through program specialization
- Dependency leaks
  - System.out.println leaks, but
  - Collection classes (e.g. ArrayList) do not



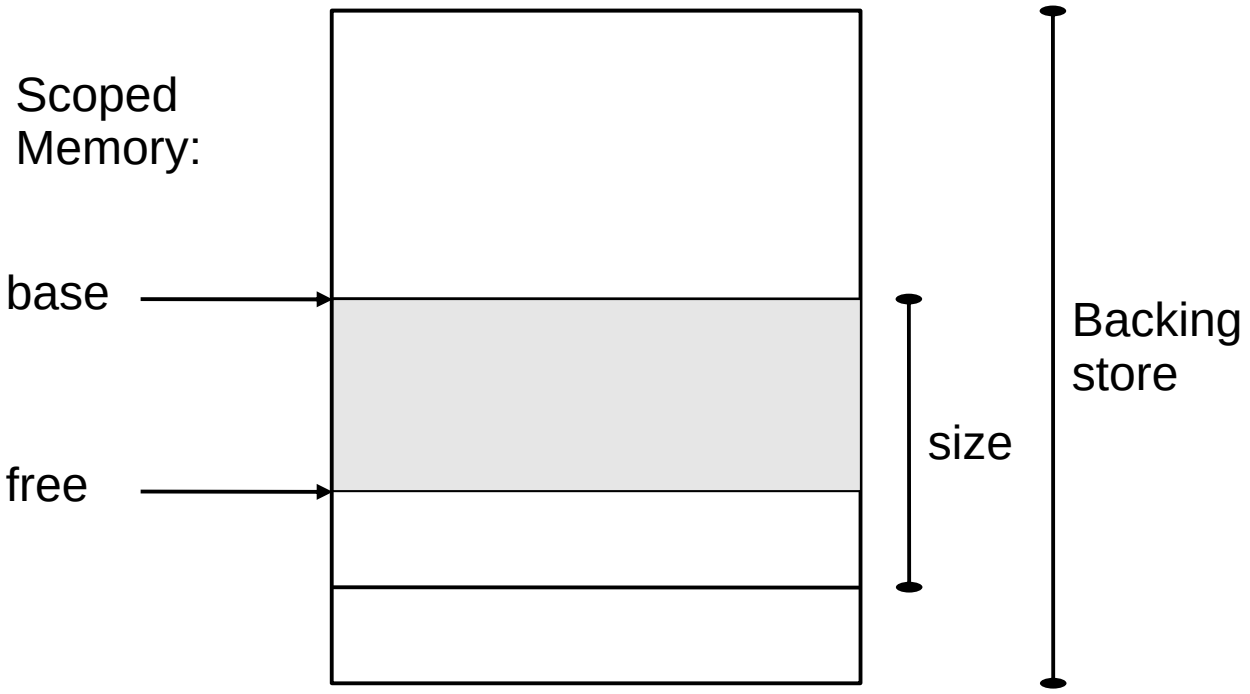
SCJ

- A bare metal implementation
  - No RTSJ
  - The VM interface





Scoped Memory



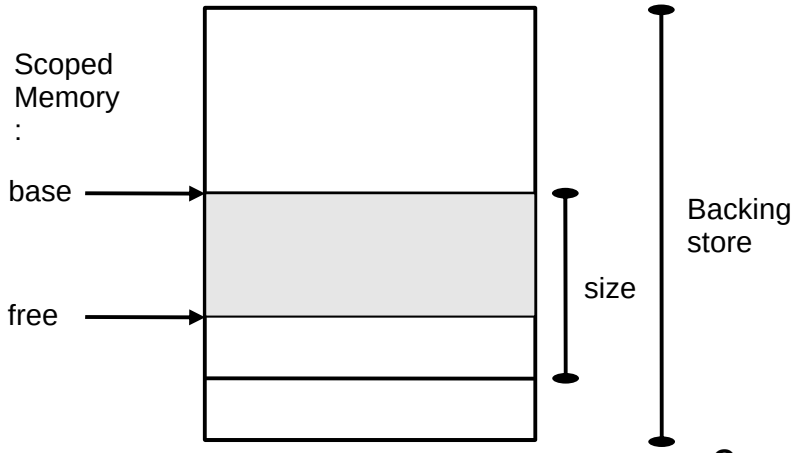
Scoped Memory

Java:

```
public class AllocationArea {
  protected int base;
  protected int size;
  protected int free;

  @IcecapCVar
  private static int HVMbase;
  @IcecapCVar
  private static int HVMfree;
  @IcecapCVar
  private static int HVMsize;

  @IcecapCompileMe
  public static void switchAllocationArea(AllocationArea newScope,
                                          AllocationArea oldScope) {
     oldScope.base = HVMbase;
     oldScope.free = HVMfree;
     oldScope.size = HVMsize;

     HVMbase = newScope.base;
     HVMfree = newScope.free;
     HVMsize = newScope.size;
  }
  ...
}
```

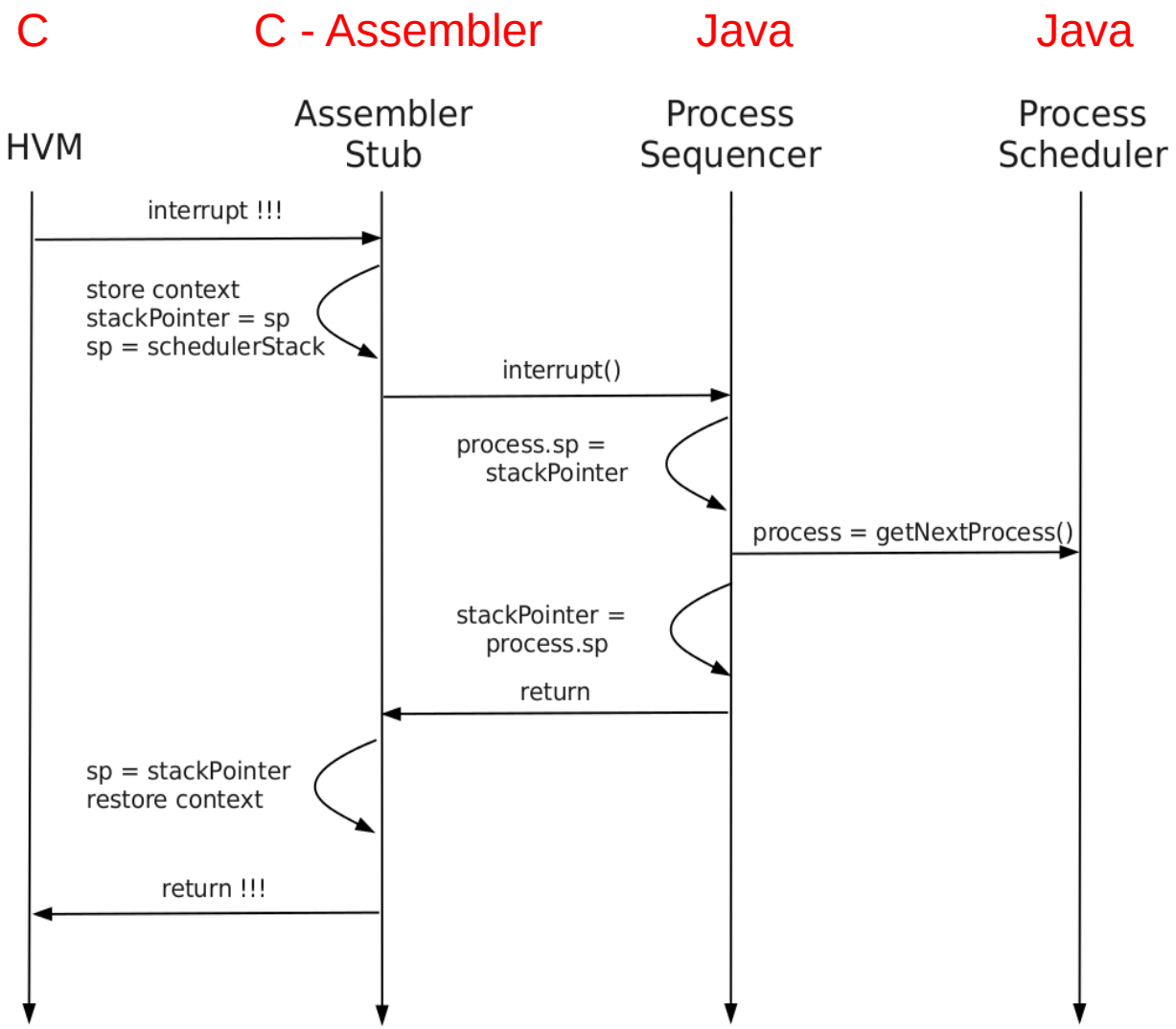
C:

```
unsigned char* HVMbase;
uint32 HVMfree;
uint32 HVMsize;
```



Scheduling

- Context switch through the layers



Real-Time Clock

- Platform specific
  - E.g. KT4585,

```
@IcecapCVar
private static int systemTick;
```

  - ATMega2560
    - Hardware clock
      - Configured using Hardware Objects
      - Tick interrupt handled in Java

Evaluation

- SCJ Level 1:

  1 Mission, 3 Handlers, KT4585

  - ROM: 35 kB
  - RAM: 10 kB

SCJ related	bytes
'Main' stack	1024
Mission sequencer stack	1024
Scheduler stack	1024
Idle task stack	256
3xHandler stack	3072
Immortal memory	757
Mission memory	1042
3xHandler memory	3x64 = 192
HVM infrastructure	
Various	959
Class fields	557
Total	9907

Evaluation

- MiniCDj, ATMega2560
  - ROM

	Classes	Methods
`java.lang.*`	46	171
`java.util.*`	10	42
`javax.safetycritical.*`	46	185
`minicdj.*`	49	216
Total	151	614

miniCDj benchmark	**ROM** (bytes)
Mostly interpreted	94682
Compilation only	282166

  - RAM, more than 300 kB

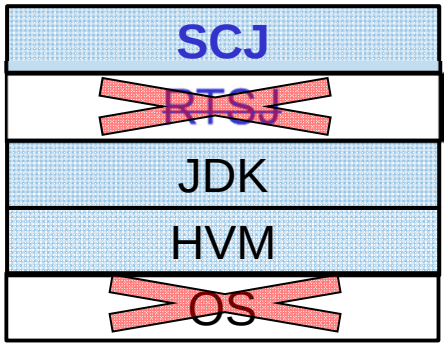
Related JVMs

- JamaicaVM
  - Hard real-time execution guarantees
  - Real-time GC
  - SCJ on top of RTSJ
  - High-end embedded platforms
- FijiVM
  - Efficient Java-to-C compiler
  - Real-time GC
  - SCJ Level 0 with native function layer
  - High-end embedded platforms
- KESO VM
  - Lean VM. Efficient Java-to-C compiler
  - GC support
  - HVM SCJ ported to KESO ?
  - Low-end embedded platforms

Conclusion

A SCJ Level 0 + 1 implementation for low-end platforms by means of:

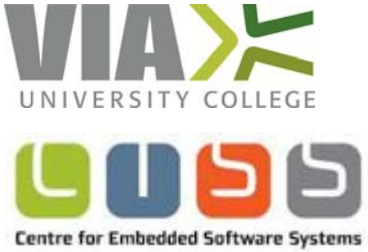
- A bare metal implementation of SCJ using a VMInterface
- No special JDK required
- A lean and portable HVM, no library dependencies
- Hardware near features like Hardware Objects



Typical memory resources

16 kB RAM, 256 kB ROM

Are we happy now?



- Ensure SCJ compatibility
- Development environment
- Improve Java SCJ infrastructure
- Learn efficient compilation from Fiji