# Private Memory Allocation Analysis for Safety-Critical Java

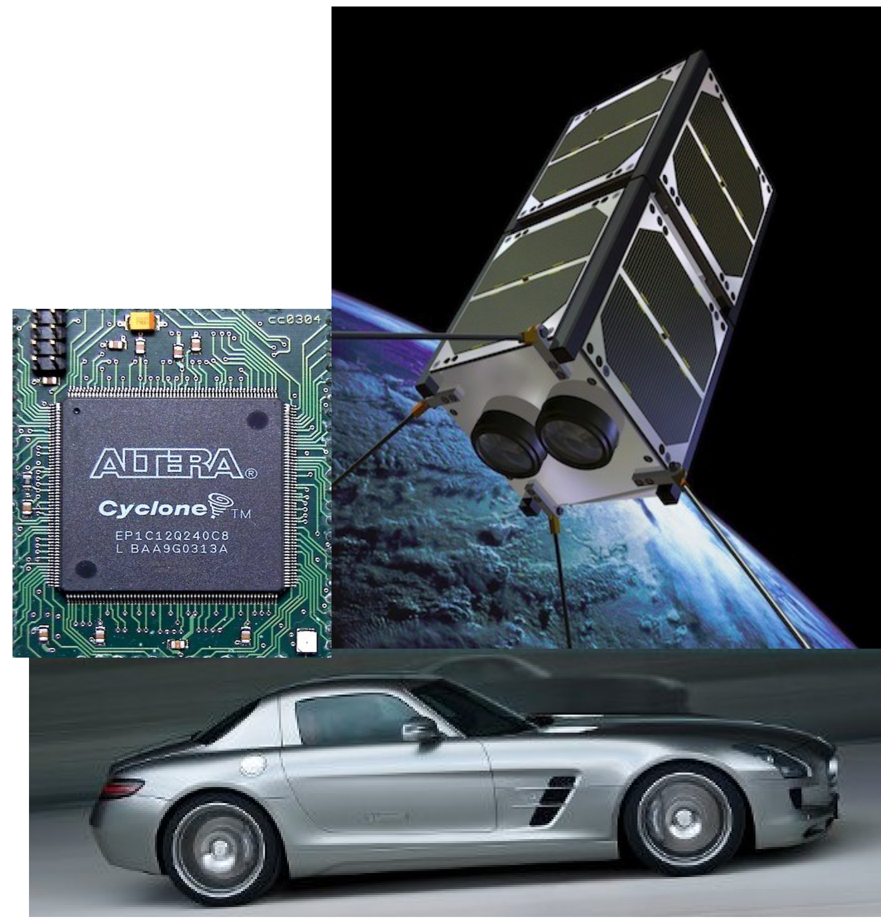
**Andreas E. Dalsgaard**
René R. Hansen
Martin Schöberl

Department of Computer Science, Aalborg University {andrease,rrh}@cs.aau.dk
Informatics and Mathematical Modeling Technical University of Denmark masca@imm.dtu.dk

# Introduction to CJ4ES

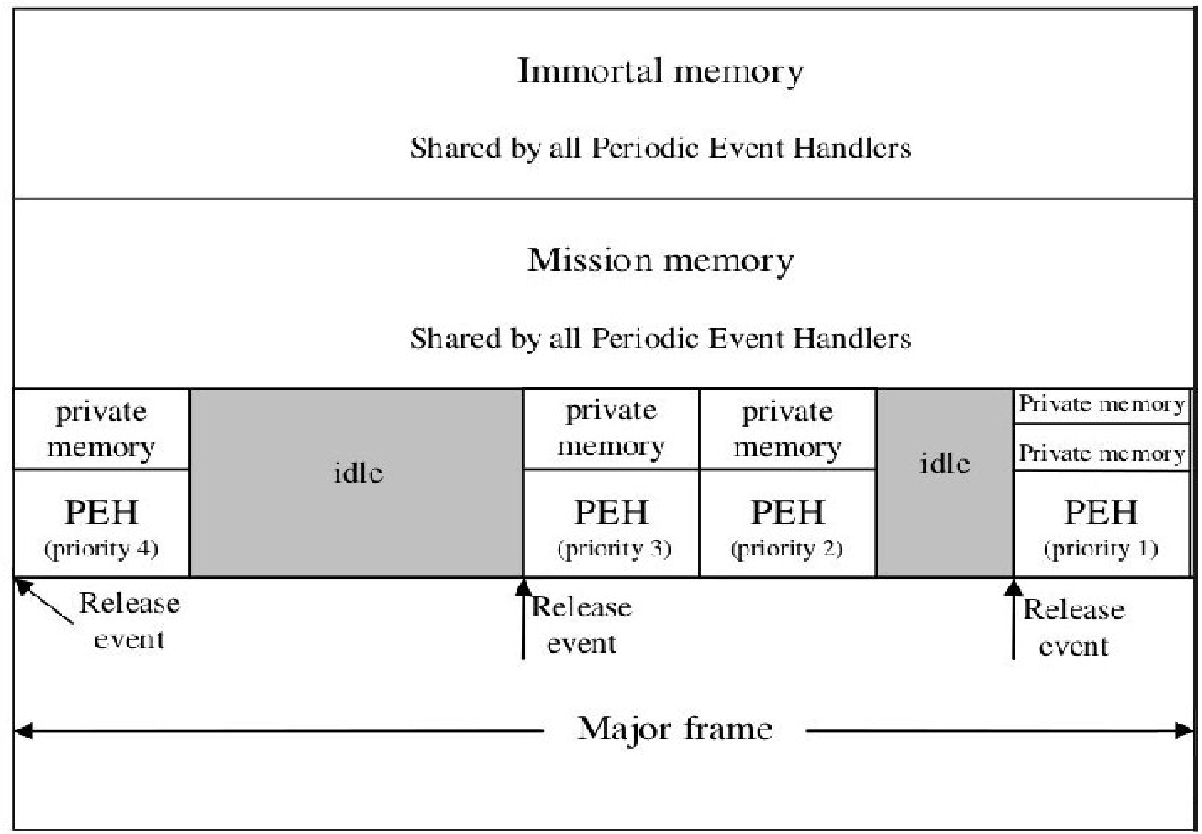
- CJ4ES
  - Target Safety Critical Java (SCJ)
  - JOP

- SCJ
  - Predictability
  - No garbage collection
  - Scoped memory
  - Large API
    - RTSJ



# SCJ - Execution Model

- Immortal Memory
- Missions
  - EventHandlers
- Private Memory
- Nested PM

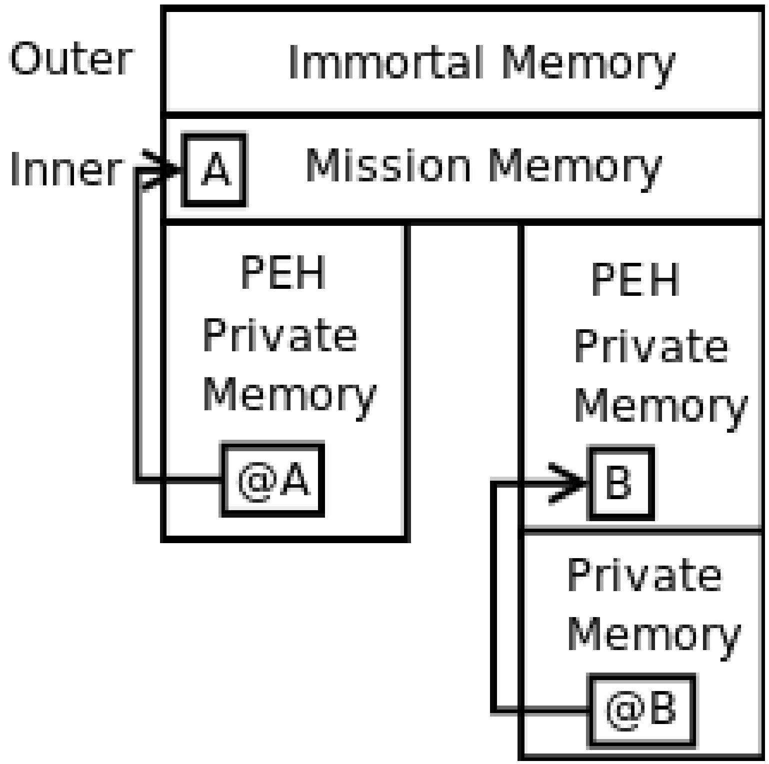


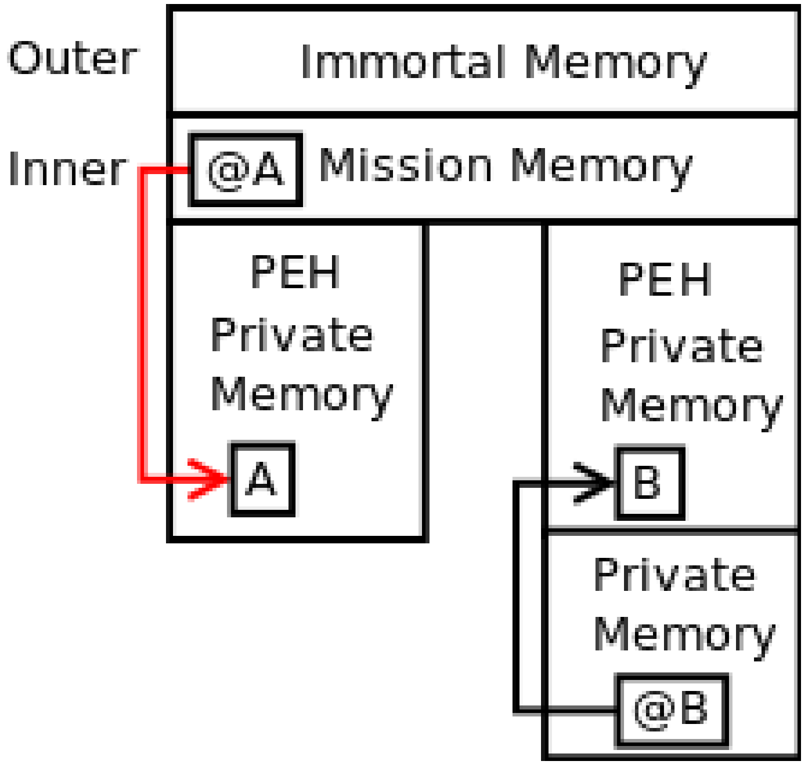
# Example of SCJ Application

```
public class InOutParameter extends Mission implements Safelet {
        @Override protected void initialize() {
                ... //initialize output stream (variable out)
                PeriodicEventHandler peh = new PeriodicEventHandler(...) {
                        public void handleAsyncEvent() {
                                InParam ip = new InParam();
                                StringBuilder outParam = new StringBuilder(30);
                                Worker w = new Worker(ip, outParam);
                                for (int i=0; i<10; ++i) {
                                        ip.s = "iter ";
                                        ip.i = i;
                                        ManagedMemory.enterPrivateMemory(500, w);
                                        out.println(outParam);
                                }
                };
                peh.register();
        }
        public static void main(String[] args) {
                JopSystem.startMission(new InOutParameter(););
        }
        ... // Safelet methods
}
... // InParam and Worker class definition
```

# SCJ - Memory Model

- References from outer scope to objects in an inner scope is not permitted
- References between scope stacks is not permitted



# Related Work on SCJ

- SCJ-Checker
  - Use annotations as a type system
  - Implemented using the Checker Framework
  - Works well for all levels
- Problems with annotations
  - Programmers have to write them
  - Class duplication

# Current Solution

```
@DefineScope(name="H", parent="M") @SCJAllowed(members=true)
@Scope("M") class Handler extends PeriodicEventHandler {

  Table st;

  @SCJAllowed(SUPPORT) @RunsIn("H") void handleAsyncEvent() {
    Sign s = ... ;
        @Scope("M") V3d old_pos = st.get(s);
        if (old_pos == null) {
          @Scope("M") Sign n_s = mkSign(s);
          st.put(n_s);
        } else ...
  }

  @RunsIn("H") @Scope("M") Sign mkSign(@Scope("M") Sign s) {
    @Scope(IMMORTAL) @DefineScope(name="M",parent="IMMORTAL")
    ManagedMemory m = (ManagedMemory) MemoryArea.getMemoryArea(s);

    @Scope("M") Sign n_s = ManagedMemory.newInstance(Sign.class);
    n_s.b = (byte[]) MemoryArea.newArrayInArea(s, byte.class, s.length);
    for (int i : s.b.length) n_s.b[i] = s.b[i];
    return n_s
  }
}
```

Figure 1.3: CD$x$ Handler implementation.

# Current Solution

```
@DefineScope(name="H", parent="M") @SCJAllowed(members=true)
@Scope("M") class Handler extends PeriodicEventHandler {

  Table st;

  @SCJAllowed(SUPPORT) @RunsIn("H") void handleAsyncEvent() {
    Sign s = ... ;
        @Scope("M") V3d old_pos = st.get(s);
        if (old_pos == null) {
          @Scope("M") Sign n_s = mkSign(s);
          st.put(n_s);
        } else ...
  }

  @RunsIn("H") @Scope("M") Sign mkSign(@Scope("M") Sign s) {
    @Scope(IMMORTAL) @DefineScope(name="M",parent="IMMORTAL")
    ManagedMemory m = (ManagedMemory) MemoryArea.getMemoryArea(s);

    @Scope("M") Sign n_s = ManagedMemory.newInstance(Sign.class);
    n_s.b = (byte[]) MemoryArea.newArrayInArea(s, byte.class, s.length);
    for (int i : s.b.length) n_s.b[i] = s.b[i];
    return n_s
  }
}
```

Figure 1.3: CD$x$ Handler implementation.

- 14 rules need to be checked for memory assignments

# Strategy

- Analyse on bytecode level
  - Precision over analysis run-time
  - Aid in verification process
    - Provide immediate feedback to developers
- Application + SCJ implementation library
  - Stubs
  - JOP SCJ
    - Extended JOP
  - Illegal assignments in SCJ implementation

# Analysis

- Perform a context sensitive pointer analysis
  - Build call graph – Dynamic dispatch
  - Stack of scopes used as context
    - Identify when contexts should change
  - Distinguish instances based on allocation site
- Perform check of result of pointer analysis
  - Compare scope stack of pointer and instance
  - Scope stack of instance $\sqsubseteq$ scope stack of pointer

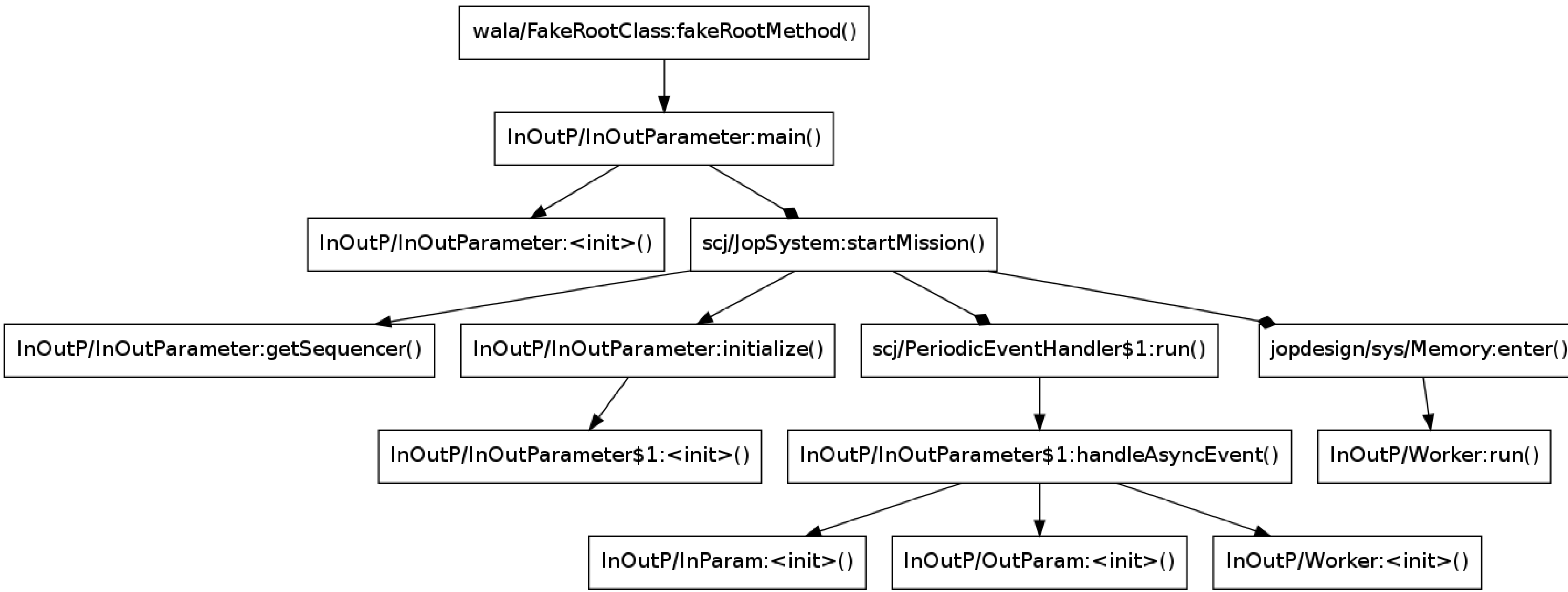
# Identify Context/Scope Change

- Inferred from call graph
  - StartMission – SCJ library specific
  - handleAsyncEvent
  - enterPrivateMemory

# Applying the Analysis

```java
public class InOutParameter extends Mission implements Safelet {
	@Override protected void initialize() {
		... //initialize output stream (variable out)
		PeriodicEventHandler peh = new PeriodicEventHandler(...) {
			public void handleAsyncEvent() {
				InParam ip = new InParam();
				StringBuilder outParam = new StringBuilder(30);
				Worker w = new Worker(ip, outParam);
				for (int i=0; i<10; ++i) {
					ip.s = "iter ";
					ip.i = i;
					ManagedMemory.enterPrivateMemory(500, w);
					out.println(outParam);
				}
		};
		peh.register();
	}
	public static void main(String[] args) {
		JopSystem.startMission(new InOutParameter(););
	}
	... // Safelet methods
}
... // InParam and Worker class definition
```

# Call Graph of InOutParameter




(ImmortalMemory, IM) $\sqsubseteq$ (ImmortalMemory, IM) : (InOutP/InOutParameter, MISSION)

# Identify Context/Scope Change(2)

- Memory reference to scopes
  - GetCurrentManagedMemory
  - GetMemoryArea
  - executeInArea

# Identify Context/Scope Change(2)

```
public class InOutParameter extends Mission implements Safelet {
        @Override protected void initialize() {
                ... //initialize output stream (variable out)
                PeriodicEventHandler peh = new PeriodicEventHandler(...) {
                        public void handleAsyncEvent() {
                                ... //initiaize variables
                                ManagedMemory pehpm = ManagedMemory.getCurrentManagedMemory()
                                Worker w = new Worker(ip, pehpm, this);
                                ... //for-loop
                };
                peh.register();
        }
        public void saveResult(Object result)
        {...}
        ... // Safelet and main methods
}
class Worker implements Runnable {
... // Fields and constructor
        @Override public void run() {
                ... // Compute varible result based on ip
                this.pehpm.executeInArea(new CopyToPeh(result, this.peh));
        }
}
class CopyToPeh implements Runnable {
... // Fields and constructor
        @Override public void run() {
                this.peh.saveResult(this.result)
        }
}
```

# Implementation

- Use T. J. Watson Libraries for Analysis (WALA)
  - Provide static analysis of bytecode
  - Support customising context changes
  - Support separating application and run-time-library
- Took more time than expected

# Tracking Context Change with WALA

- Context changes inferred from call graph
- Result of heap graph analysis unavailable from customised context selector
- Observation
- Remember last leaked memory reference
  - getCurrentManagedMemory
  - getMemoryArea

# Overview

- Build call-graph of SCJ app. and JOP SCJ impl.
  - Identify context changes
- Annotate call graph nodes with contexts
- Build Basic HeapGraph
  - PointerKeys and InstanceKeys get contexts
- Compare scope stacks of PointerKeys and InstanceKeys

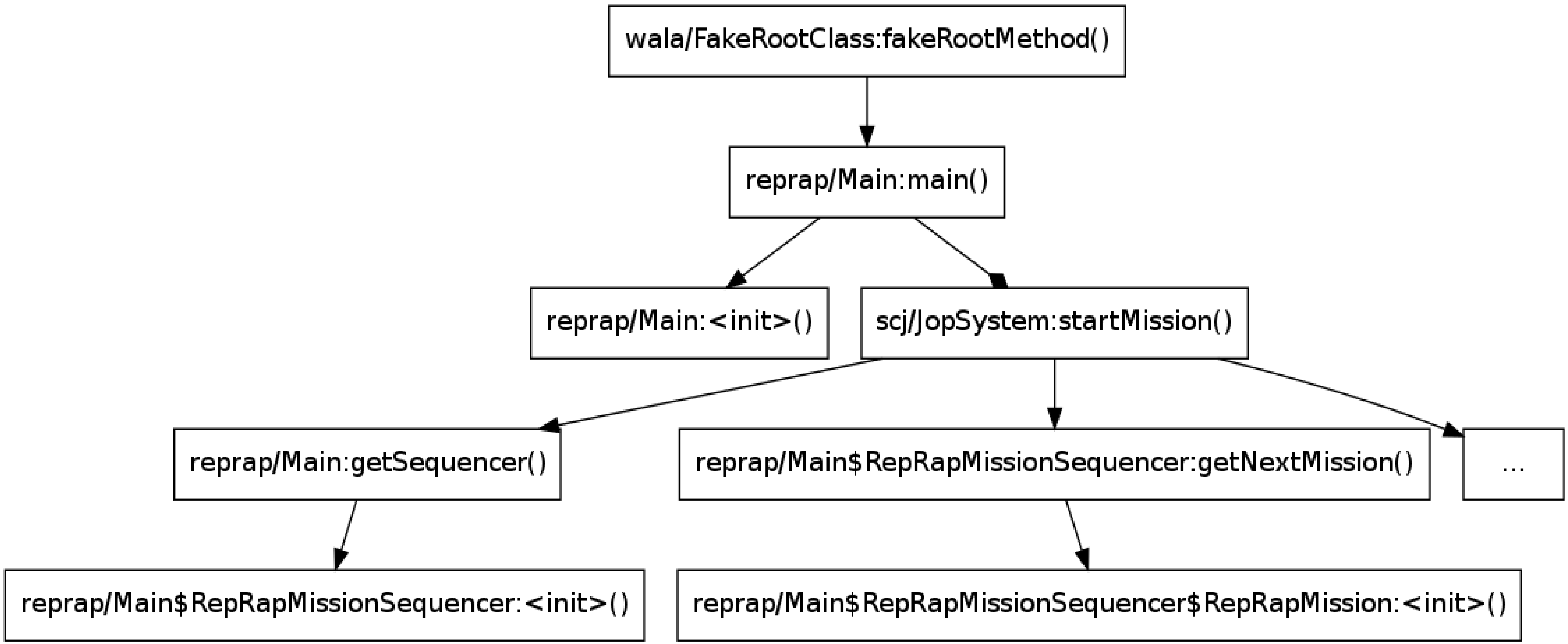
# Experiments

- Lines of code(LOC)
- Bytecode size in byte SCJ library/SCJ application

| Test case | LOC | Bytecode | Illegal Assignments | Reported |
|---|---|---|---|---|
| scjminepump | 1465 | 239884/18519 | 0 | 0 |
| scjminepumplog | 1490 | 239884/20511 | 1 | 1 |
| pmFFTcpResult | 545 | 247854/11577 | 0 | 0 |
| InOutParameter | 155 | 264949/6285 | 1 | 2 |
| scjreprap | 1758 | 242561/27730 | 4 | 5 |

- False positive in scjreprap
  - Due to implementation details of JOP SCJ
- False positive in InOutParameter
  - Clever reuse of space in a StringBuilder

# False positive in scjreprap - getSequencer()



# Example: Clever Reuse of StringBuilder

```
class InParam {
        String s;
        int i;
}
class Worker implements Runnable {
        InParam in;
        StringBuilder outParam;

        public Worker(InParam in, StringBuilder outParam) {
                this.in = in;
                this.outParam = outParam;
        }

        @Override
        public void run() {
                String s = in.s + in.i; // Concatenation generate garbage
                outParam.setLength(0);
                outParam.append(s); // Avoid allocating a new buffer
        }
}
```

# Experiences using WALA

- Can analyse real Java programs/bytecode
- Many different analyses
- Hard to get an overview
  - To use it – read the code
  - Lot of subclassing
- Performance optimisations
  - Makes debugging difficult
- No documentation of what is ensured by analyses

# Conclusion

- SCJ illegal assignment analysis tool
- More benchmarks
  - Real world examples
- Formalisation of the analysis
- More analyses tools of SCJ applications

- Links:
  - http://www.soc.tuwien.ac.at/jop.git
  - https://github.com/andreasDalsgaard/privmem

# Questions?