Rapid Development of Dynamic Analysis Tools for the Java Virtual Machine

Walter Binder

University of Lugano
Switzerland
walter.binder@usi.ch

Motivation

- Bytecode instrumentation is commonly used for dynamic analysis tools
  - Profiling
  - Debugging
  - Testing
  - Reverse engineering
- However, low-level bytecode instrumentation is error-prone, time-consuming, and expensive
  - Resulting tools are difficult to maintain and extend

Our Goals

- High-level specification of dynamic analysis tools
  - Compact tools
  - Easy to develop and extend
- Full method coverage
  - Including the Java class library
  - Including dynamically generated code
- Efficient tools
  - Use of idle CPU cores
  - Static analysis to optimize inserted code
- Portability and platform independence
  - Compatibility with state-of-the-art JVMs

Our Approach

Aspect-oriented programming (AOP)

+

Extensions

+

Advanced bytecode instrumentation techniques

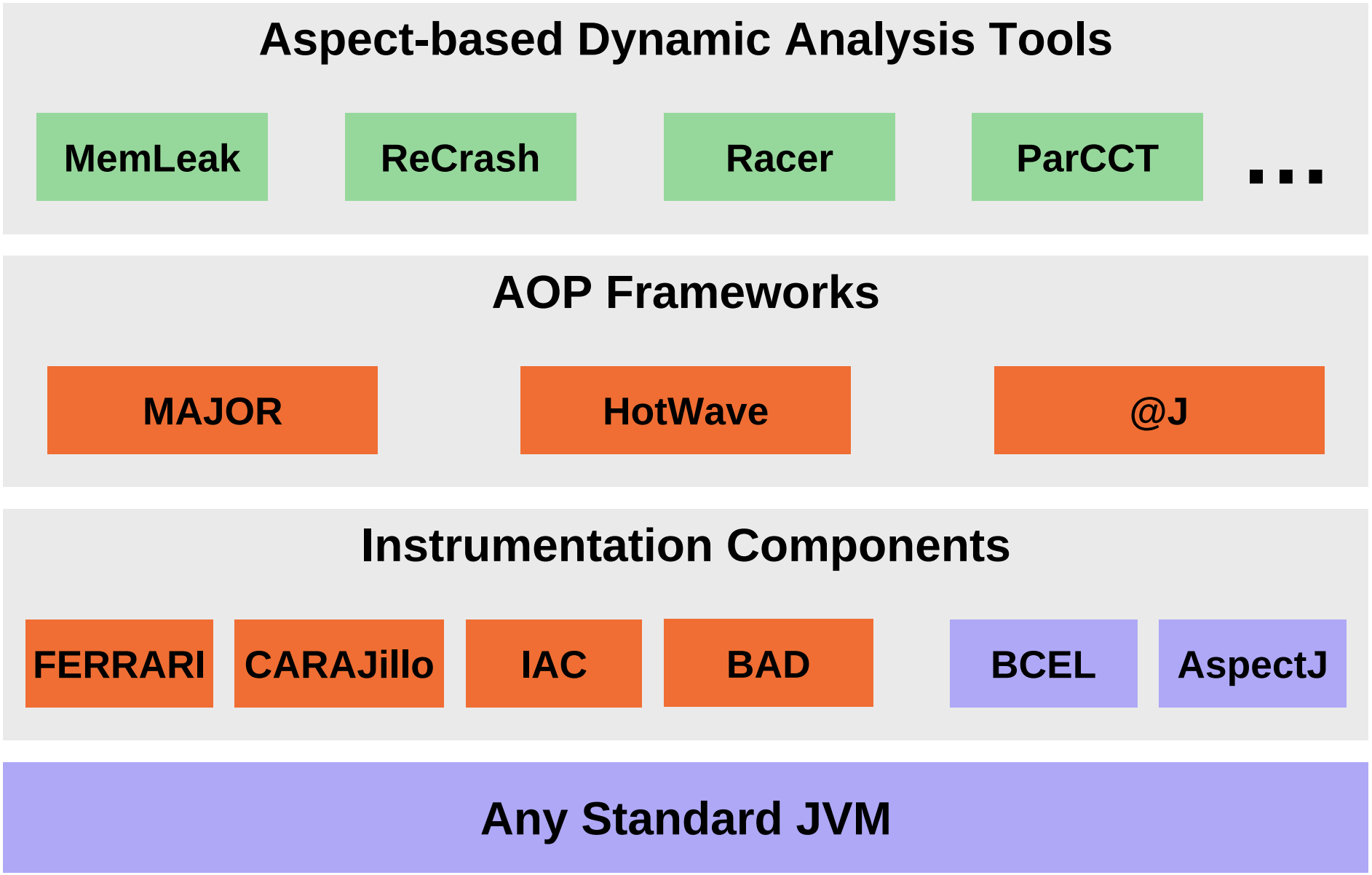
Example Aspect

- Count number of object allocations

```
aspect AllocCounter {
  final AtomicLong counter = new AtomicLong();

  after returning(Object o) : call(*.new(..)) {
    counter.incrementAndGet();
  }
  …
}
```

Limitations of Current AOP Frameworks

- Methods in Java class library cannot be woven
- Data passing between woven advice in local variables is not supported
- Limited set of join point. E.g., basic blocks of code are not supported as join points
- No support for user-defined static analyses at weaving time to customize the weaving process
- No dedicated support for executing advice on idle CPU cores

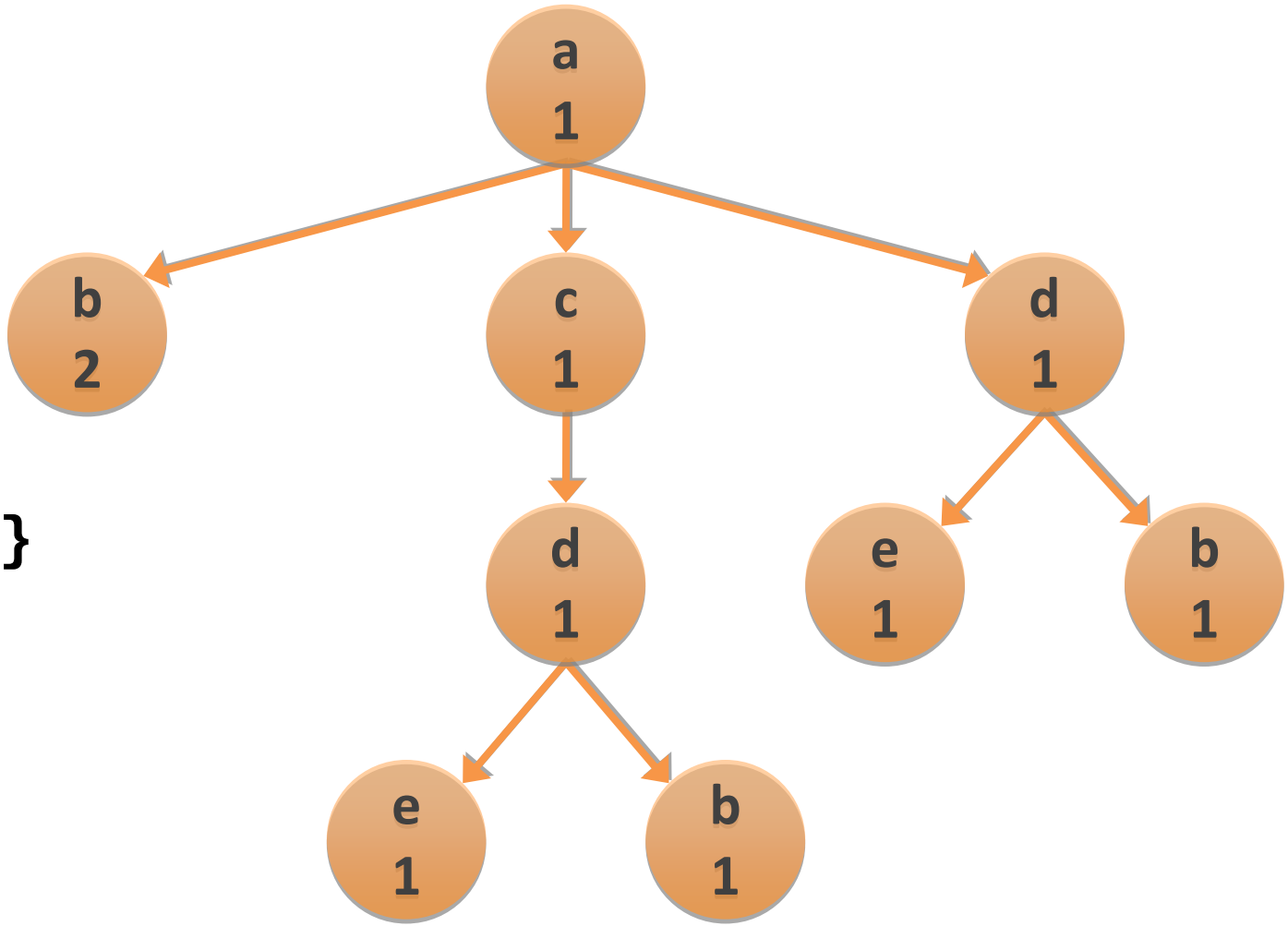

Aspect-based Dynamic Analysis Tools
MemLeak
ReCrash
Racer
ParCCT
...
AOP Frameworks
MAJOR
HotWave
@J
Instrumentation Components
FERRARI
CARAJillo
IAC
BAD
BCEL
AspectJ
Any Standard JVM

Case Study: Calling Context Profiling

- Analysis of the dynamic inter-procedural control flow of applications
- Dynamic metrics for each calling context
- Useful for
  - Performance analysis
  - Program optimization
  - Program comprehension
  - Reverse engineering

Calling Context Tree (CCT)

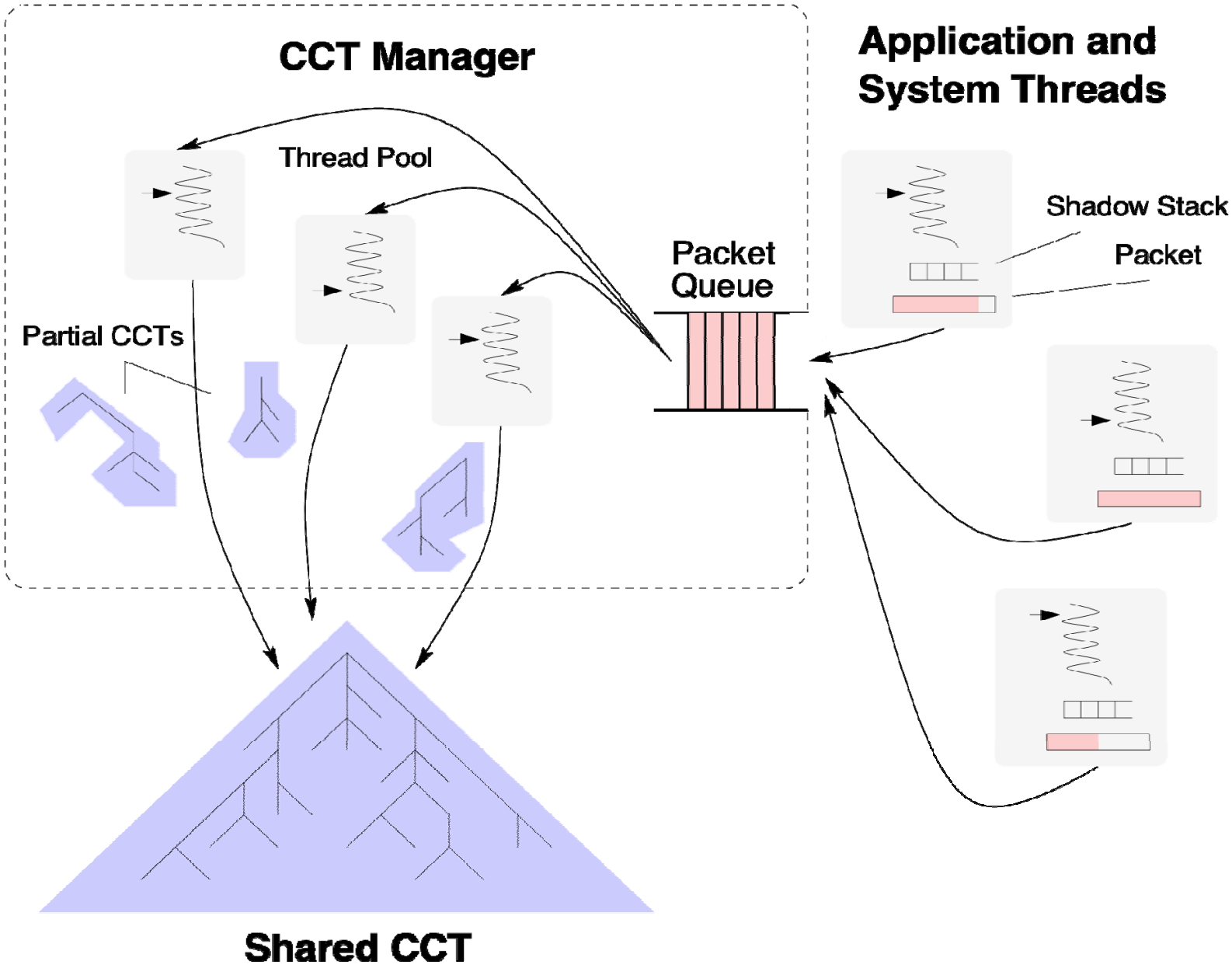
```
a() {
  b();
  c();
  b();
  d();
}

b() { }

c() { d(); }

d() {
  e();
  b();
}

e() { }
```



Aspect for Naïve CCT Creation

```
public aspect NaiveCCT {
   private static final CCTNode root = new CCTNode();

   private static final ThreadLocal<CCTNode> currentNode =
      new ThreadLocal<CCTNode>() {
         protected CCTNode initialValue() { return root; }
   };

   pointcut execs() : execution(* *.*(..)) && !within(NaiveCCT);

   before() : execs() {
      CCTNode caller = currentNode.get();
      CCTNode callee = caller.getCallee(thisJoinPointStaticPart);
      currentNode.set(callee);
   }

   after() : execs() {
      CCTNode callee = currentNode.get();
      CCTNode caller = callee.getParent();
      currentNode.set(caller);
   }
   …
}
```

Efficient CCT Creation (ParCCT)



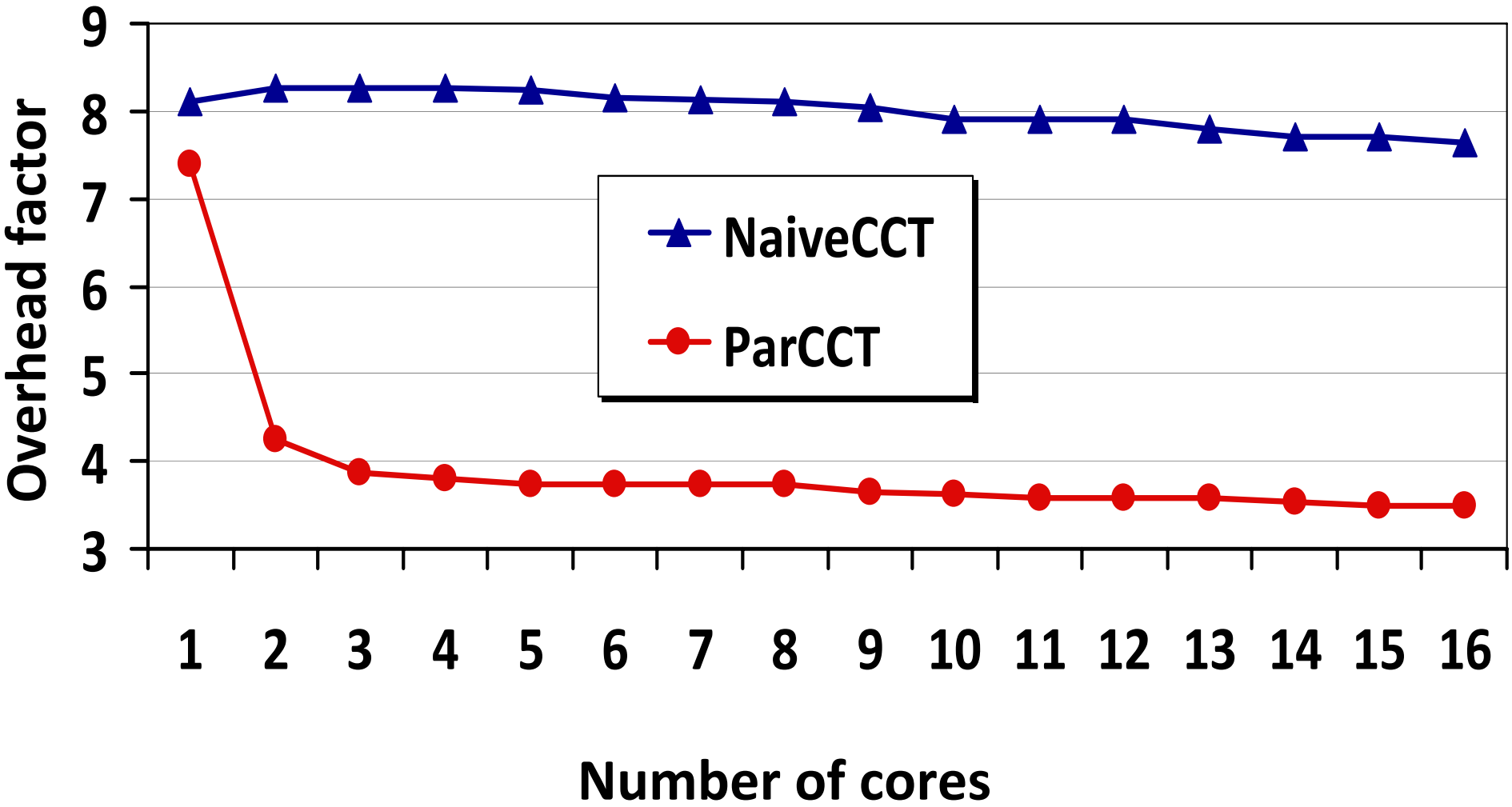
ParCCT Aspect (1)

```
public class TC { // thread context
   // shadow stack
   public static final int STACK_SIZE = 10000;
   public final Object[] stack = new Object[STACK_SIZE];
   public int sp = 0; // next free entry on shadow stack

   // current packet
   public static final int PACKET_SIZE = 40000;
   public Object[] packet = new Object[PACKET_SIZE];
   public int nextFree = 1; // next free entry in packet
}
```

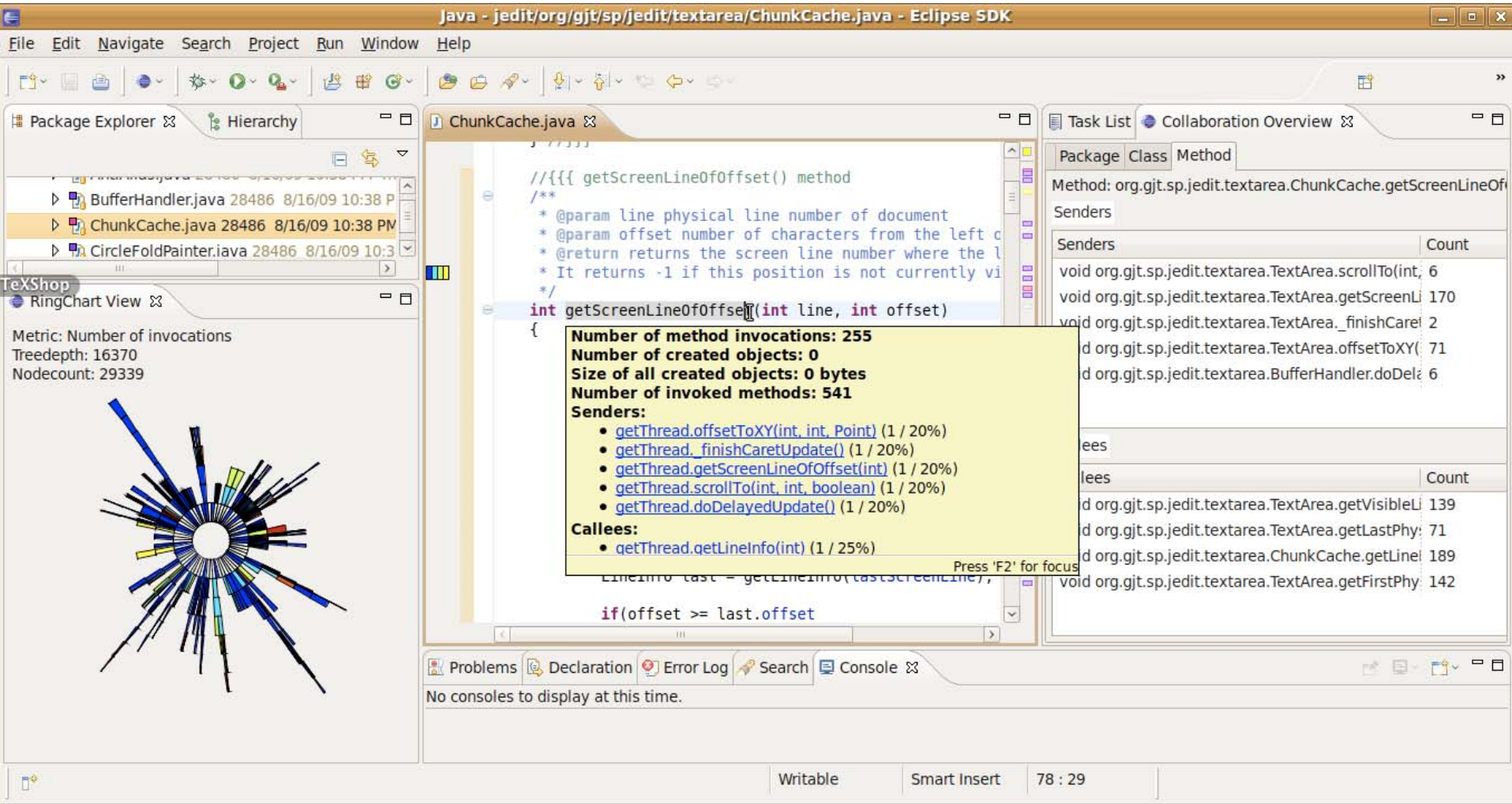
ParCCT Aspect (2)

```
public aspect ParCCT {
   private static final ThreadLocal<TC> currentTC =
      new ThreadLocal<TC>() {
         protected TC initialValue() { return new TC(); }
   };

   pointcut execs() : execution(* *.*(..)) && !within(ParCCT);

   before() : execs() {
      TC tc = currentTC.get();
      if (tc.nextFree >= TC.PACKET_SIZE) { // current packet full
         CCTManager.submitPacket(tc.packet);
         tc.packet = new Object[TC.PACKET_SIZE];
         for (int i = 0; i < tc.sp; ++i)   // create packet header
            tc.packet[i] = tc.stack[i];
         tc.nextFree = tc.sp + 1;
      }
      tc.stack[tc.sp++] = thisJoinPointStaticPart;
      tc.packet[tc.nextFree++] = thisJoinPointStaticPart;
   }

   after() : execs() {
      TC tc = currentTC.get(); tc.sp--; tc.nextFree++;
   }
   …
}
```

DaCapo Overhead (1–16 Cores, Geo. Mean)



Senseo Plugin for Eclipse



Joint work with the Software Composition Group, University of Bern

Conclusion http://www.inf.usi.ch/projects/ferrari/

- Building dynamic analyses with AOP helps reduce development effort and eases extension
- Current AOP frameworks lack some essential features
- We are extended the AspectJ weaver with the needed features (e.g., full method coverage)
- Case studies show that existing low-level tools can be recast as aspects, and new tools can be rapidly developed

My Team

- Danilo Ansaloni (PhD student)
- Philippe Moret (PhD student)
- Alex Villazón (postdoc)