Applied Static Analysis
2016

Dr. Michael Eichberg (Organizer)
Johannes Lerch, Ben Hermann, Sebastian Proksch, Karim Ali Ph.D.

Software Technology Group
TECHNISCHE UNIVERSITÄT DARMSTADT

Abstract Interpretation

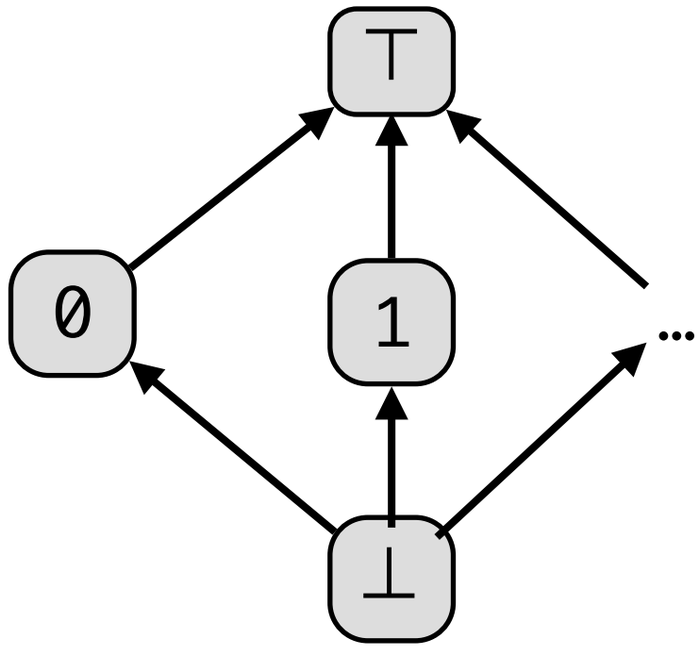
- A general methodology for calculating analyses.
- The idea is to perform an abstract interpretation of a computer program to gain information about its semantics (data- and control-flow) using monotonic functions over lattices. Using abstract semantics it will compute a superset of the concrete program semantics (sound).

Lattices

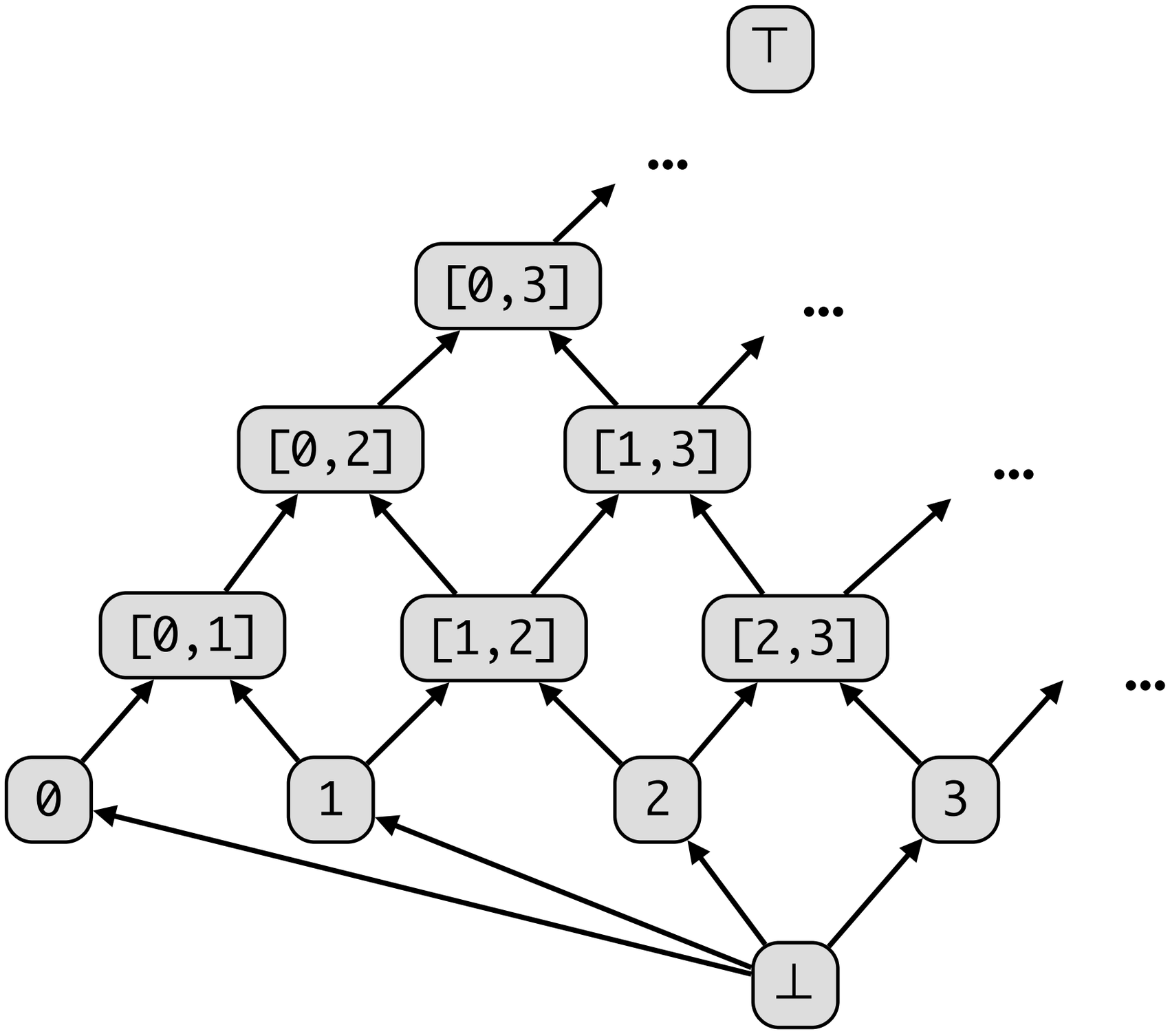
- Partially ordered set is a set equipped with a partial ordering. A partial ordering is a relation $\sqsubseteq$: L x L → {true, false} that is reflexive, transitive and anti-symmetric.

- A complete Lattice is a partially ordered set such that all subsets have least upper bounds as well as greatest lower bounds. Furthermore, bottom ($\bot$) is the least element and top ($\top$) is the greatest element.

Abstract Domain: Constants



Abstract Domain: Intervals

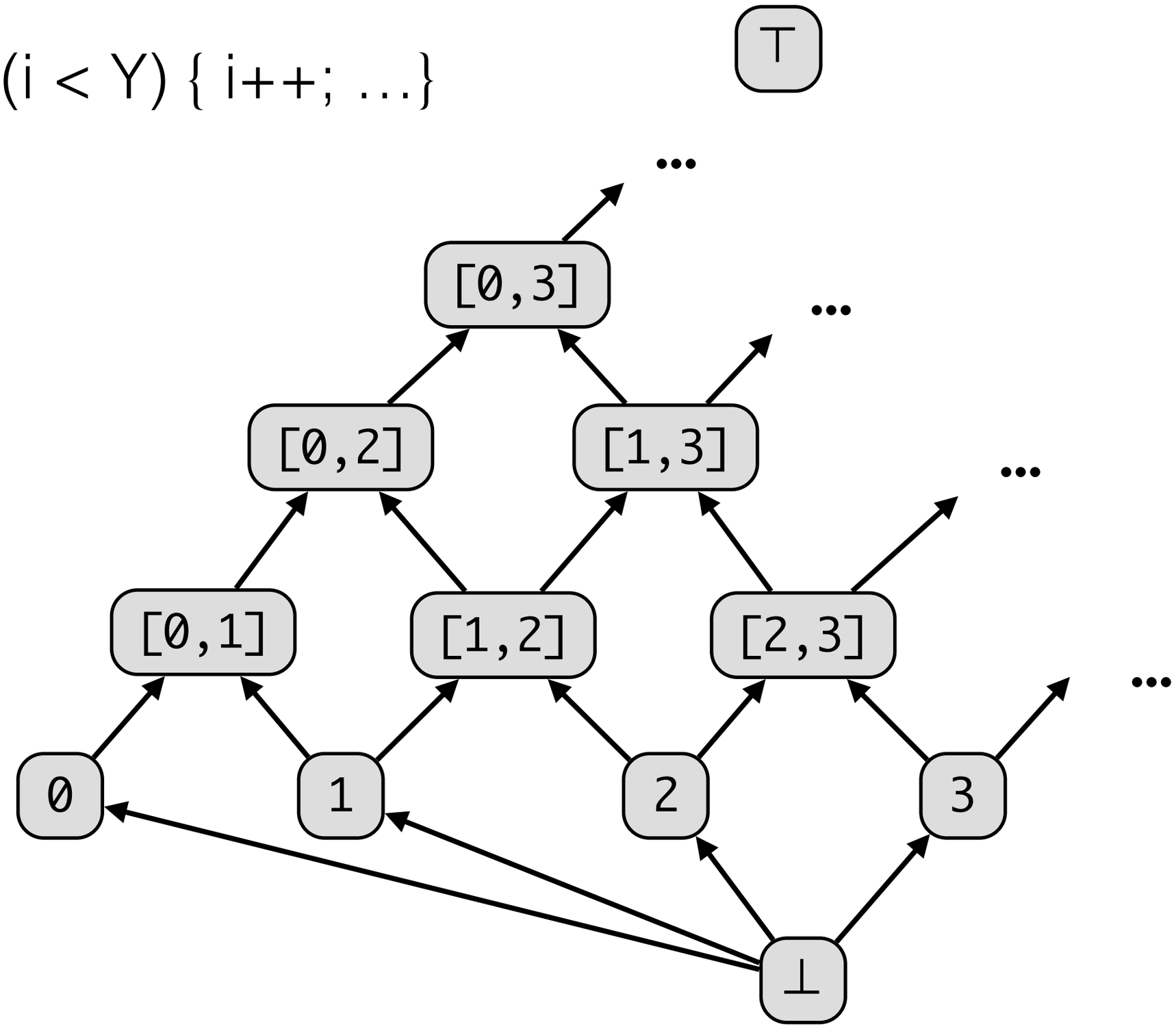


Computation on Intervals

```
int i = [1,4]
int j = [-1,2]
int z = i * j; // int z = [1,4] * [-1,2] === [-4,8]
```

Interval Lattice

while(i < Y) { i++; …}



Intra- and Interprocedural Analyses

- An ***intra***-*procedural* analysis analyzes one method at a time; not considering the behavior of other methods.

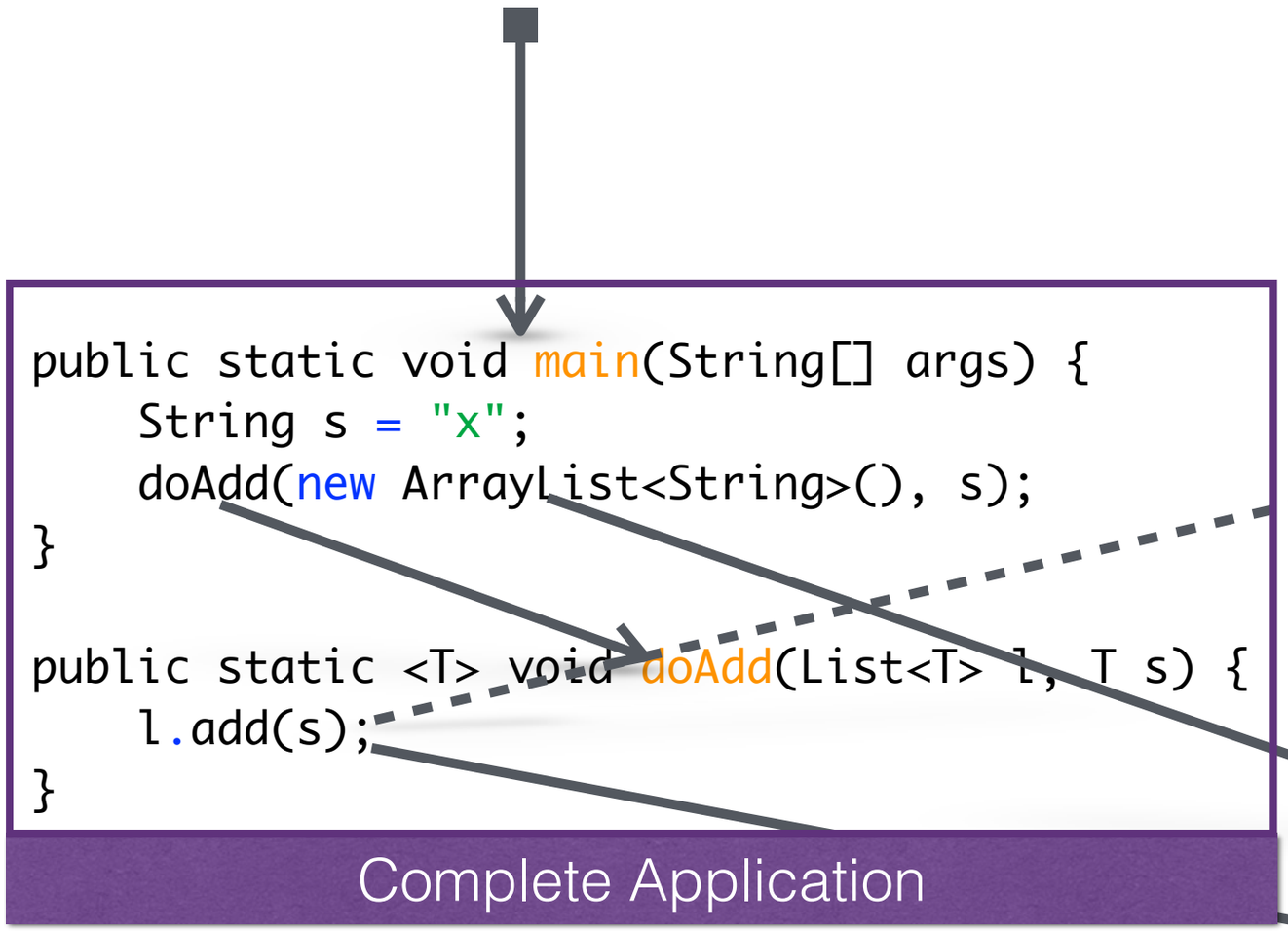
- An ***inter***-*procedural* analysis generally spans multiple methods and takes the behavior of other methods into account.
  This enables more precise analysis information.

Call Graph

- Records the information which method (*Caller*) calls which other methods (*Callees*) and where.

- The **Call Graph** is a key data-structure explicitly or implicitly used by most inter-procedural analyses.

Call Graph - Example




```
interface List<T> {
    void add(T t);
}

class LinkedList<T> implements List<T>
{
    public void add(T t) { /**/ }
}

class ArrayList<T> implements List<T>
{

    public ArrayList() { /* ... */ }

    public void add(T t) { /**/ }
}
```

Whether the edge to `LinkedList.add` is part of the Call Graph depends on the algorithm used for constructing the Call Graph.

JDK 1.8.0_25

Call Graph for the JDK

“Modified CHA”	
Method	**Number of CallEdges**
`toString`	1.825.762
`hashCode`	139.384
`equals`	504.636
Iterator related (`next`, `hasNext`, `remove`)	1.641.438
Sum	**4.111.220**
Total	**6.925.872**

many “toString” calls have 2078 call targets, because the receiver type is “Object”

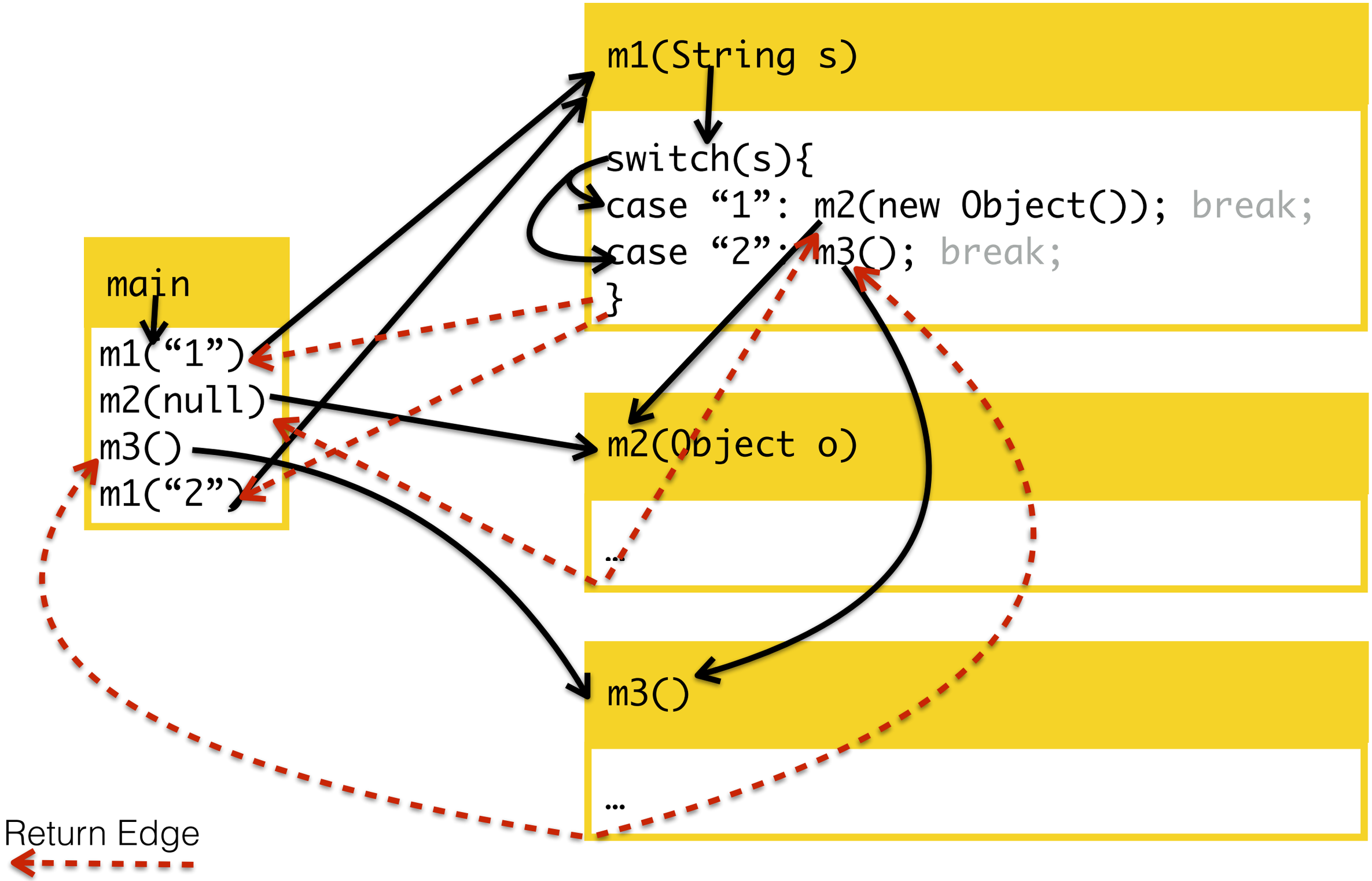
~60% of all edges are related to the 6 shown methods.

Interprocedural Control Flow Graph (ICFG)

- The ICFG is the result of combining the Call Graph with each method's individual CFG.

- … but how to do it?

Interprocedural Control Flow Graph



Context-Sensitive

- A context sensitive analysis *does not allow information from multiple calling contexts to interfere with each other.*

- Multiple possibilities exist how to distinguish the calling contexts; all have different cost/precision trade-offs.

Reference Analysis

- An analysis that seeks to **determine the set of objects to which a reference variable or field may** point during execution.

- *There are real tradeoffs between the usability of the analysis results in terms of precision and cost of obtaining them, the time and memory required.*

Class Analysis

Reference Analysis

- Per reference variable/field we calculate the set of classes (types) to which the variable/field may refer to.
  Here, one “abstract object” represents all instantiation of a class.

Pointer Analysis/Points-to Analyses

- A pointer analysis attempts to statically determine whether two variables may point to the same storage location at runtime.

Reference Analysis

Characteristics/Dimensions

Flow-Sensitive

- A flow sensitive analysis takes the **overall (intra-procedural)** control flow into account while determining points to relations at various program points. An analysis could be flow sensitive for just scalars or for object fields too.

Object Representation

- Two (most common) approaches:
  - One abstract object is used to represent **all** instantiations of a class.
  - A representative is used for each creation site to represent **all** objects created at that site.

Reference Analysis

- *Client Analyses:*
  - Escape Analyses
  - Redundant Synchronization Identification
  - Side-Effect Analyses
  - …

Pure vs. Side Effect Free

- Side-effect free
  - The execution must not have any visible effect other than to generate a result.
  - A method may still create and modify new objects and return those as the result.
- Pure:
  (Interesting in the context of verification of security properties)
  - side-effect free
  - deterministic - the result of a method must only depend on its arguments and given the same arguments return the same result.