MACHINE LEARNING FOR SOFTWARE MAINTAINABILITY

Anna Corazza, Sergio Di Martino, Valerio Maggio

Alessandro Moschitti, Andrea Passerini, Giuseppe Scanniello,

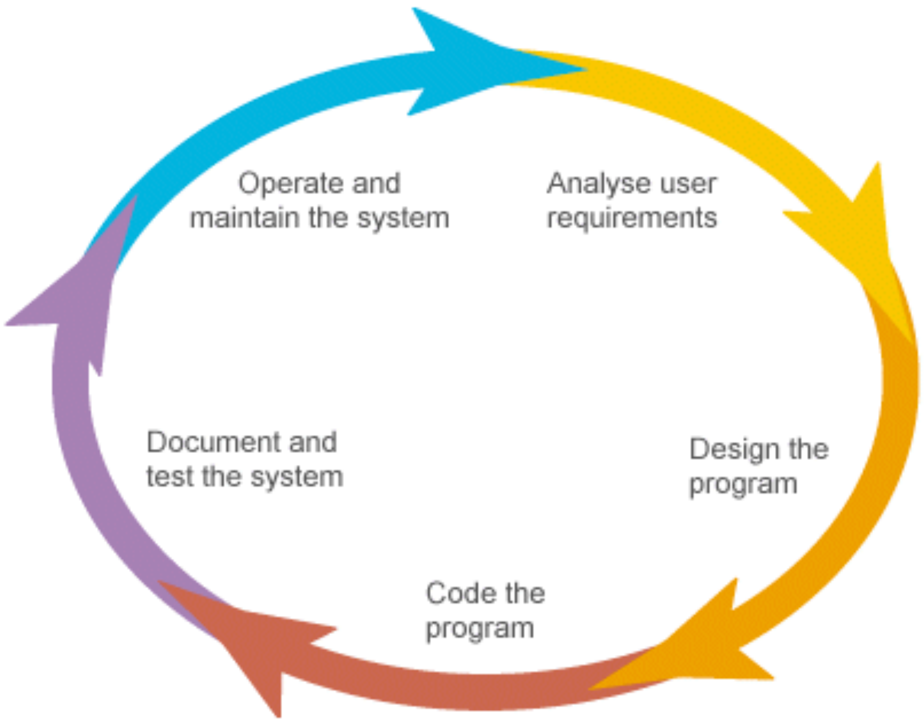
Fabrizio Silverstri

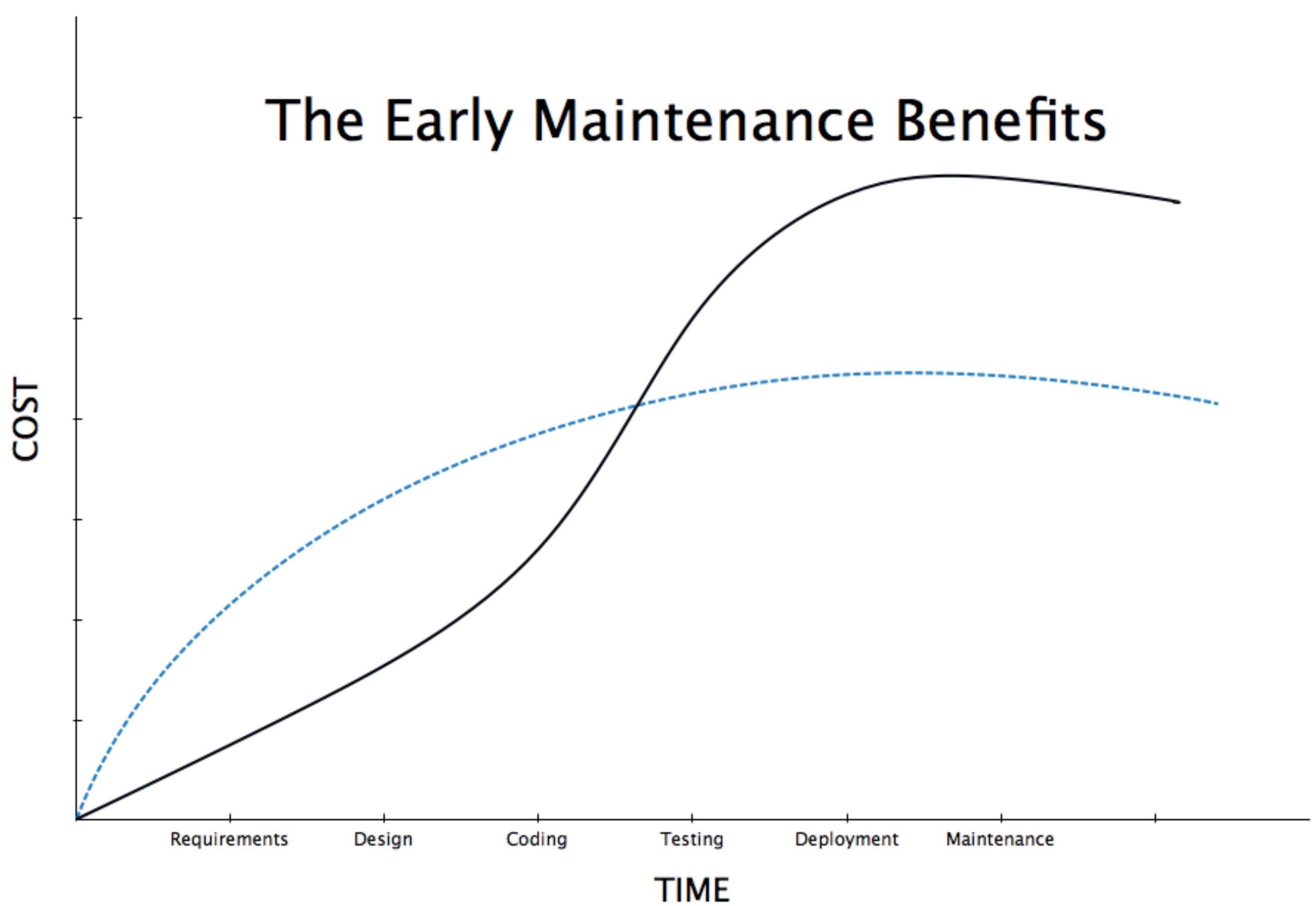
JIMSE 2012

August 28, 2012 Montpellier, France

SOFTWARE MAINTENANCE

"A software system must be continuously adapted during its overall life cycle or it progressively becomes less satisfactory"
(cit. Lehman's Law of Software Evolution)

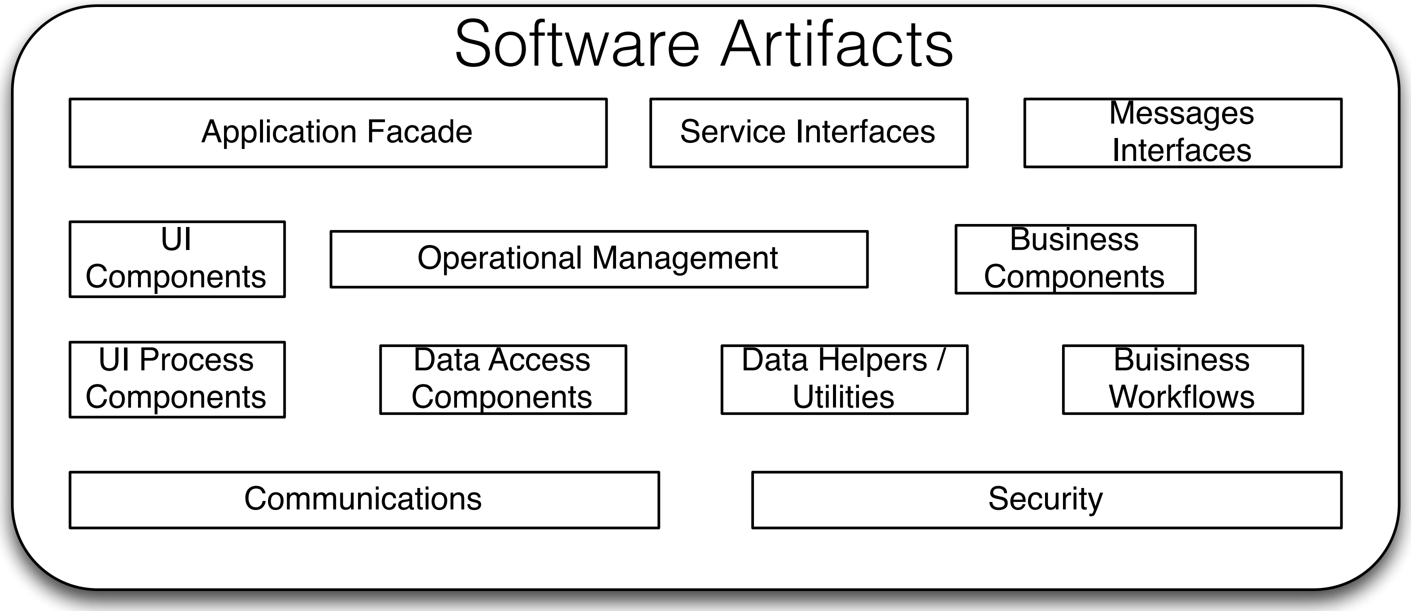




- Software Maintenance is one of the most expensive and time consuming phase of the whole life cycle
  - Anticipating the Maintenance operations reduces the cost
  - 85%-90% of the total cost are related to the effort necessary to comprehend the system and its source code [Erlikh, 2000]

SOFTWARE ARCHITECTURE

- Provide *models* and *views* representing the relationships among different **software artifacts**
  - *Clustering of Software Artifacts*



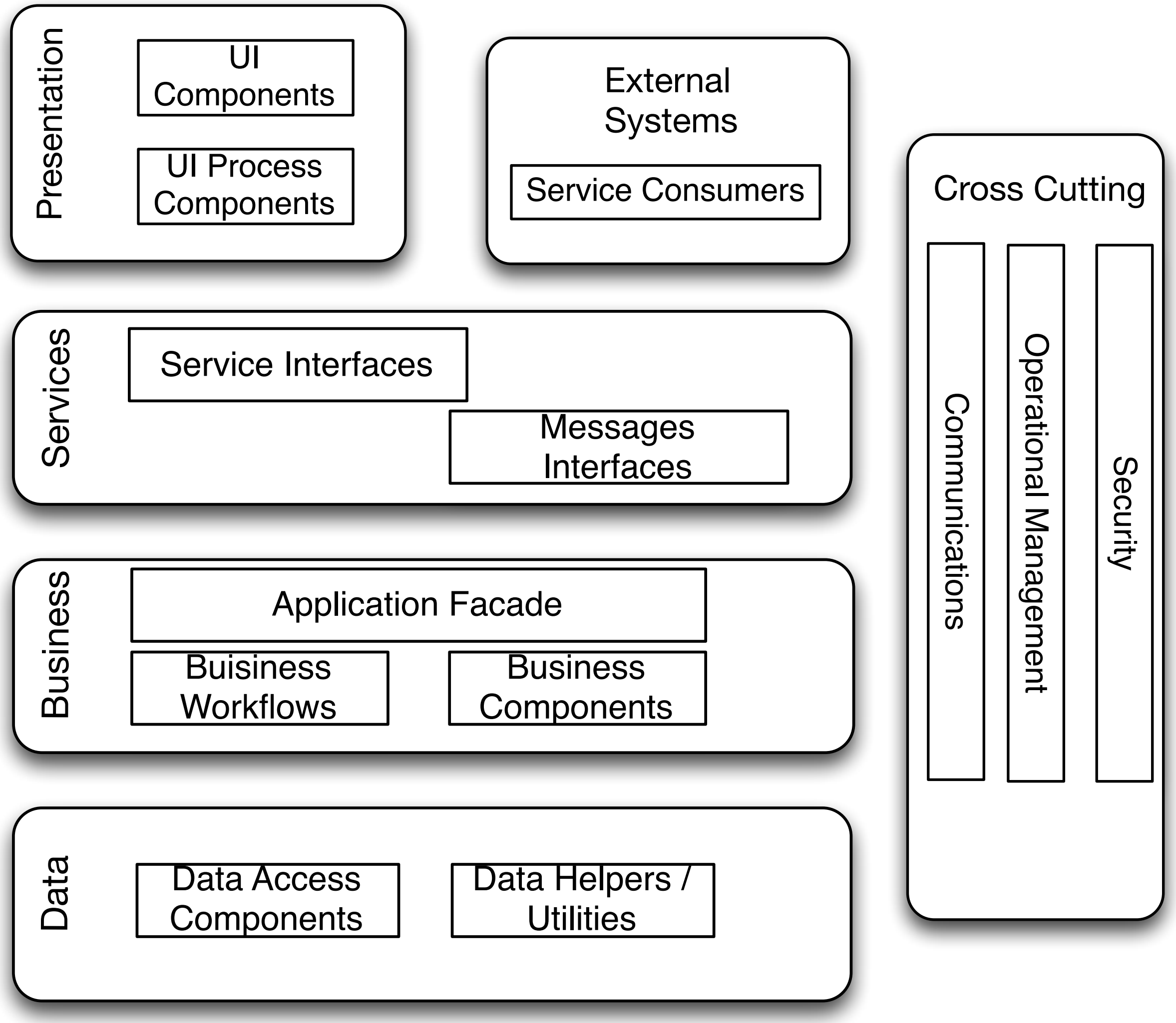
- **Advantages:**
  - To aid the comprehension
  - To reduce maintenance effort

SOFTWARE ARCHITECTURE

- Provide *models* and *views* representing the relationships among different **software artifacts**
  - *Clustering of Software Artifacts*

- **Advantages:**
  - To aid the comprehension
  - To reduce maintenance effort

Clusters of Software Artifacts

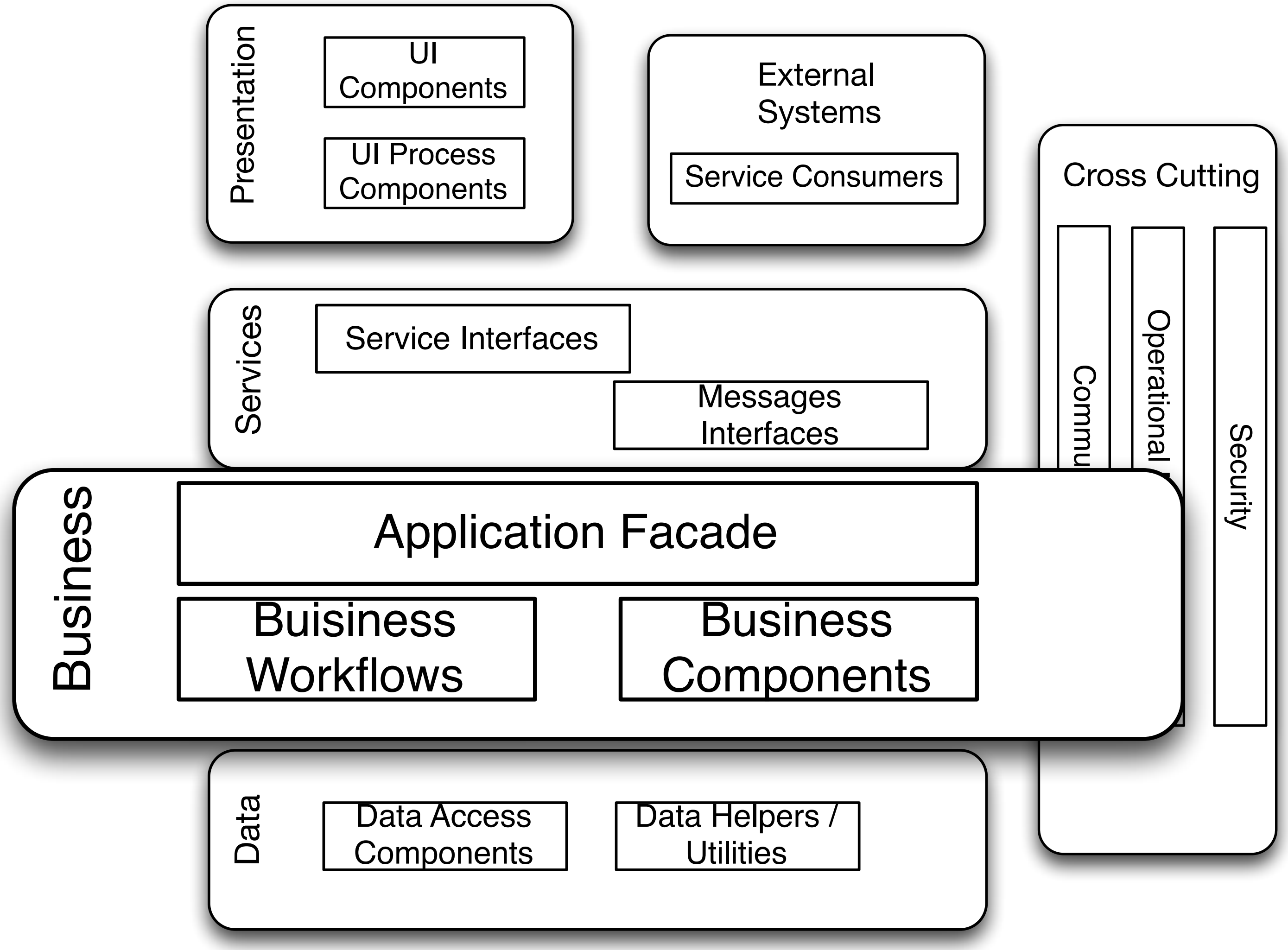


SOFTWARE ARCHITECTURE

- Provide *models* and *views* representing the relationships among different **software artifacts**
  - *Clustering of Software Artifacts*

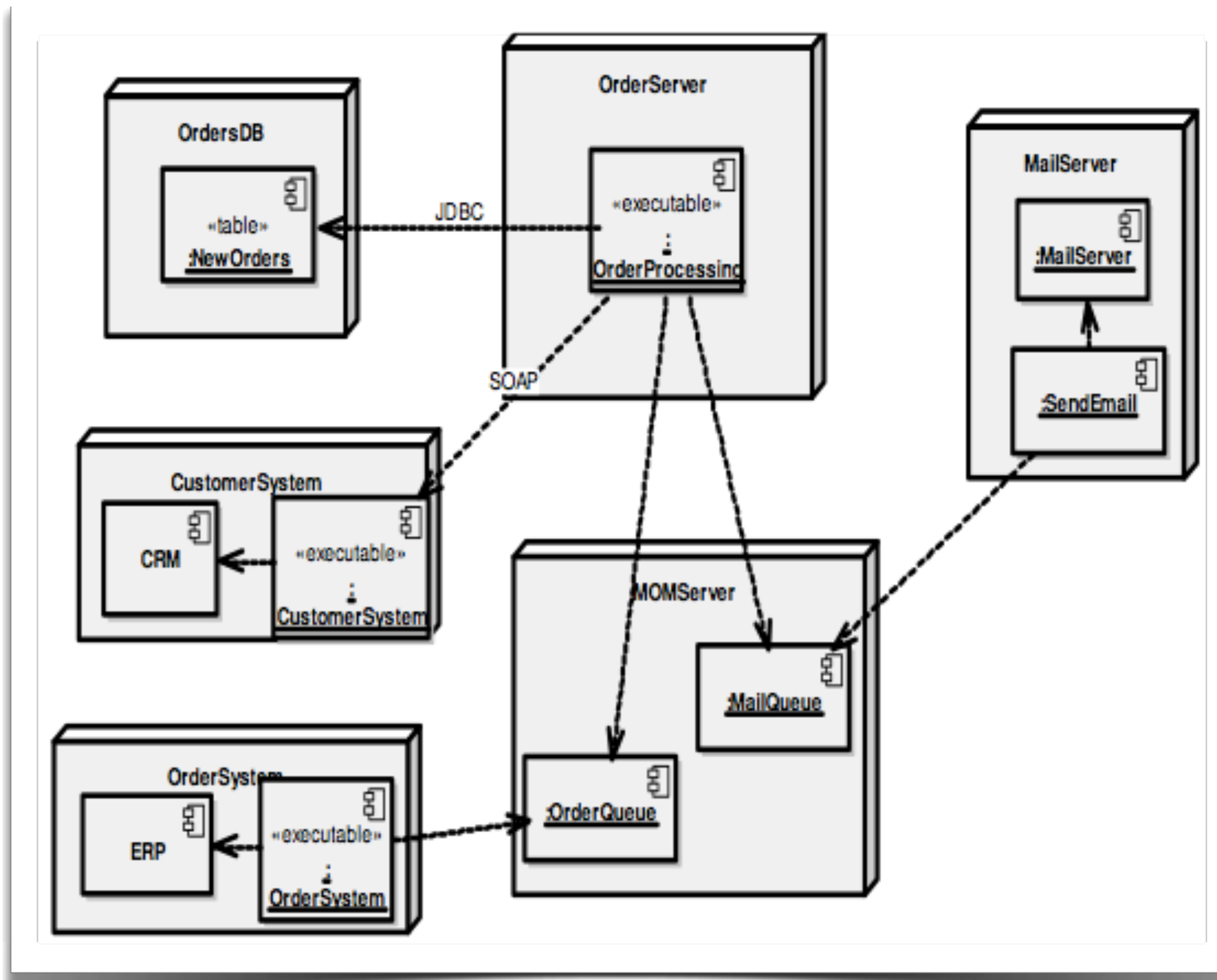
- **Advantages:**
  - To aid the comprehension
  - To reduce maintenance effort

Clusters of Software Artifacts



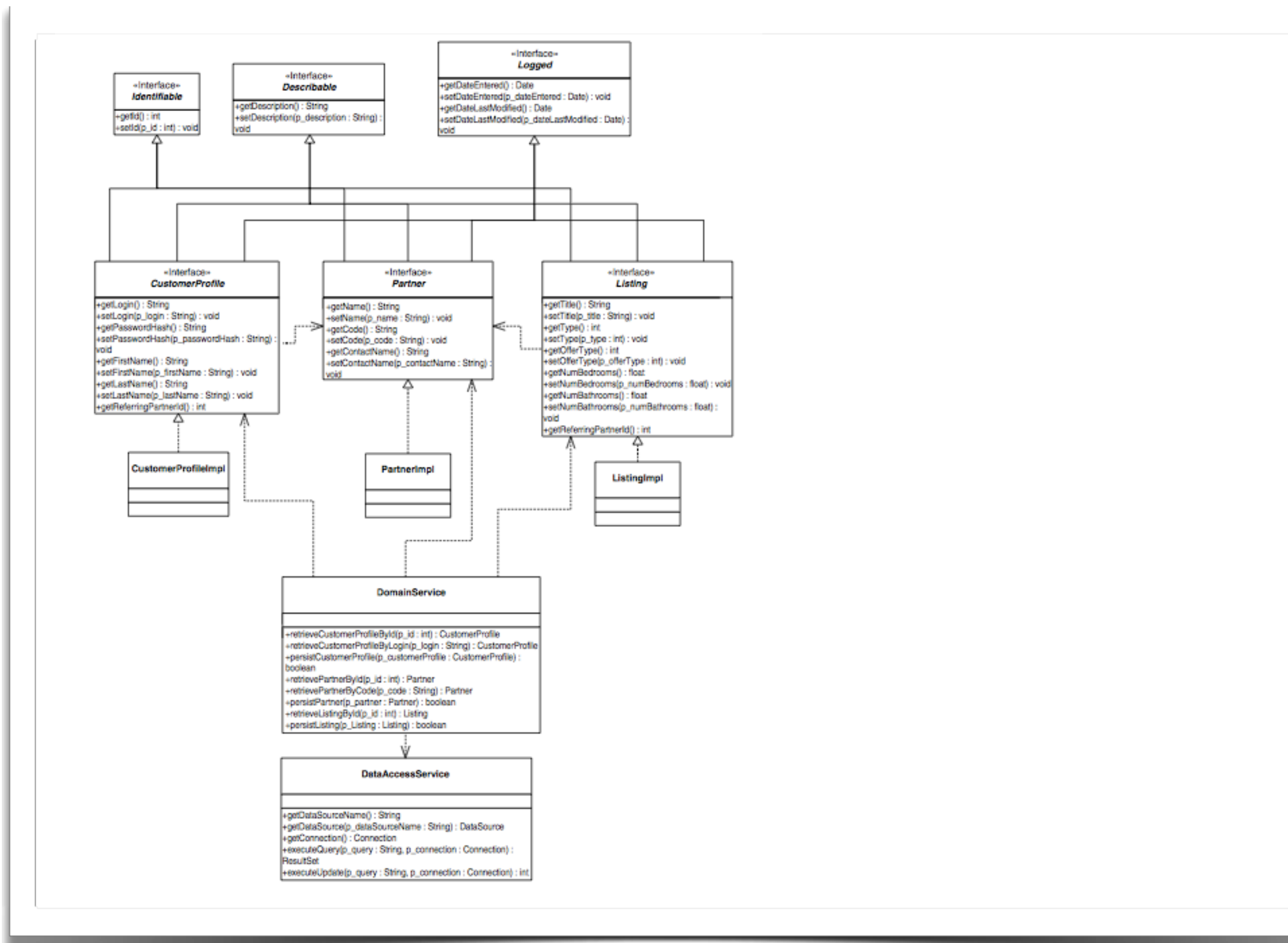
SOFTWARE ARTIFACTS

Software Artifacts may be analyzed at **different levels of abstractions**



SOFTWARE ARTIFACTS

Software Artifacts may be analyzed at **different levels of abstractions**



SOFTWARE ARTIFACTS

Software Artifacts may be analyzed at **different levels of abstractions**

The different levels of abstractions lead to different **analysis tasks:**

- *Identification of functional modules and their hierarchical arrangement*
  - i.e., Clustering of Software classes
- *Identification of Code Clones*
  - i.e., Clustering of Duplicated code fragments (blocks,

```
class WrappedClassLoader extends ClassLoader {
    private Bundle bundle;
    public WrappedClassLoader(Bundle bundle) {
        super();
        this.bundle = bundle;
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findClass(java.lang.String)
     */
    public Class findClass(String name) throws ClassNotFoundException {
        return bundle.loadClass(name);
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findResource(java.lang.String)
     */
    public URL findResource(String name) {
        return bundle.getResource(name);
    }

    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findResources(java.lang.String)
     */
    protected Enumeration findResources(String name) throws IOException {
        return bundle.getResources(name);
    }
```

SOFTWARE ARTIFACTS CLUSTERING

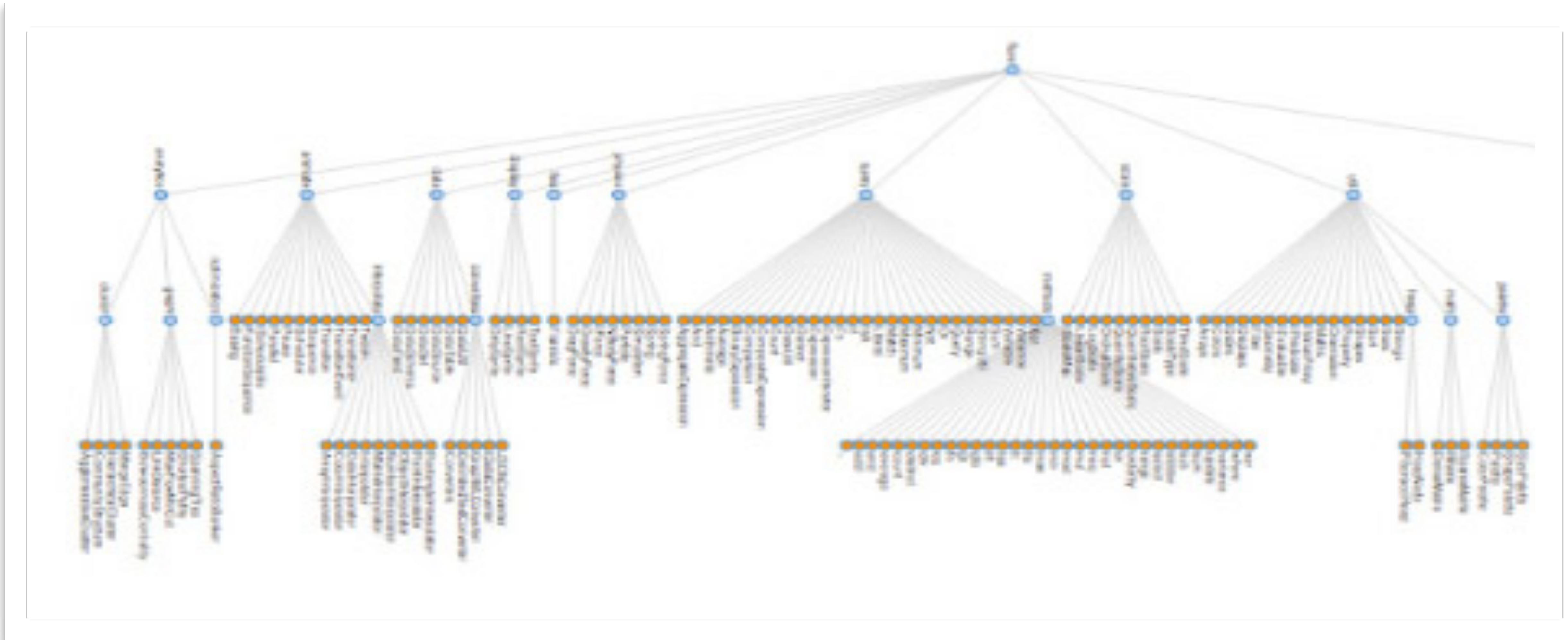
Problem: *Definition of a proper similarity measure to apply in the clustering analysis, which is able to exploit the considered representation of software artifacts*

- Mine information directly from the source code:
  - Exploit the **syntactic/lexical information** provided in the source code text
  - Exploit the **relational information** between artifacts
    - e.g., *Program Dependencies*

```
class WrappedClassLoader extends ClassLoader {
    private Bundle bundle;
    public WrappedClassLoader(Bundle bundle) {
        super();
        this.bundle = bundle;
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findClass(java.lang.String)
     */
    public Class findClass(String name) throws ClassNotFoundException {
        return bundle.loadClass(name);
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findResource(java.lang.String)
     */
    public URL findResource(String name) {
        return bundle.getResource(name);
    }

    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findResources(java.lang.String)
     */
    protected Enumeration findResources(String name) throws IOException {
        return bundle.getResources(name);
    }
```

MINING LARGE REPOSITORIES



- Analysis of **large** and **complex systems**
- Solutions and algorithms must be able to **scale** efficiently (*in the large* and *in the many*)

ADVANCED MACHINE LEARNING FOR SOFTWARE MAINTENANCE

Idea: *Definition of Machine Learning techniques to mine information from the source code*

- Combine different kind of information (lexical and structural)
    - Application of **Kernel Methods** to software artifacts
- Provide flexible and computational effective solutions to analyze large data sets

ADVANCED MACHINE LEARNING FOR SOFTWARE MAINTENANCE

Idea: *Definition of Machine Learning techniques to mine information from the source code*

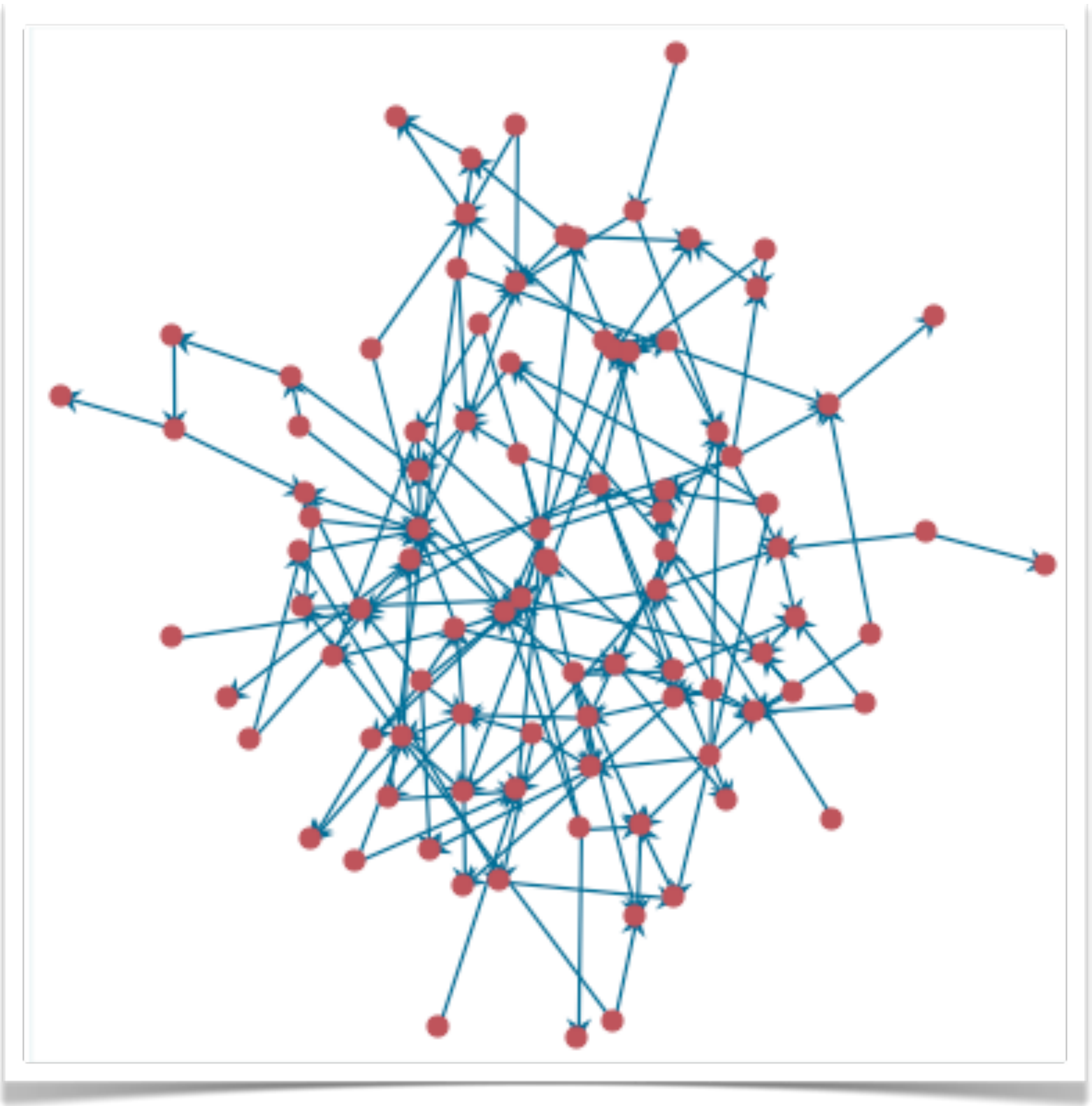
- Combine different kind of information (lexical and structural)
  - Application of **Kernel Methods** to software artifacts
- Provide flexible and computational effective solutions to analyze large data sets

Advanced Machine Learning

- Learning with syntactic/semantic information (Natural Language Processing)
- Learning in relational domains (Structured-output learning, Logic Learning, Statistical Relational Learning)

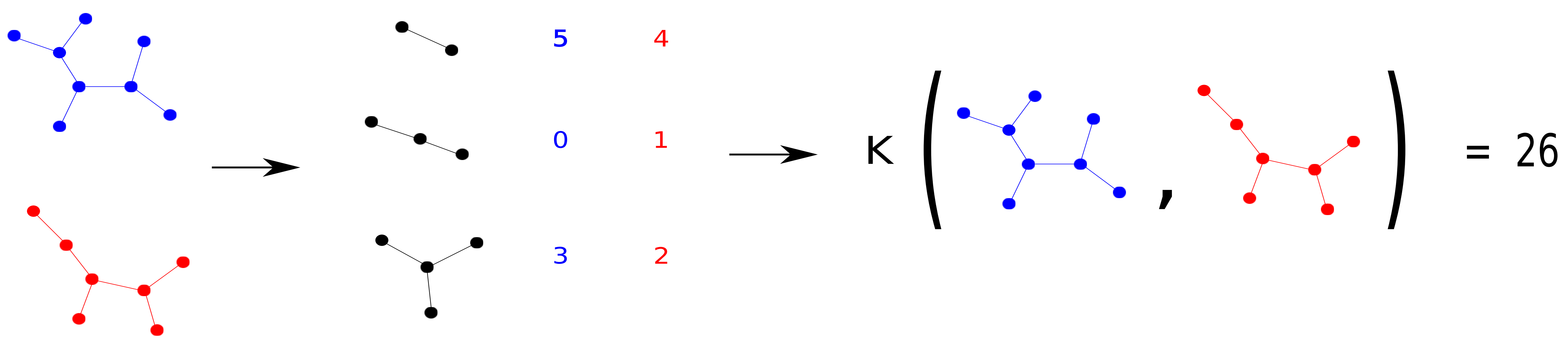
KERNEL METHODS FOR STRUCTURED DATA

- A **Kernel** is a function between (arbitrary) pairs of entities
    - It can be seen as a kind of **similarity measure**
- Based on the idea that structured objects can be described in terms of their constituent parts
- Generalize the computation of the dot product to arbitrary domains
- Can be easily tailored to specific domains
    - **Tree Kernels**
    - **Graph Kernels**
    - **....**



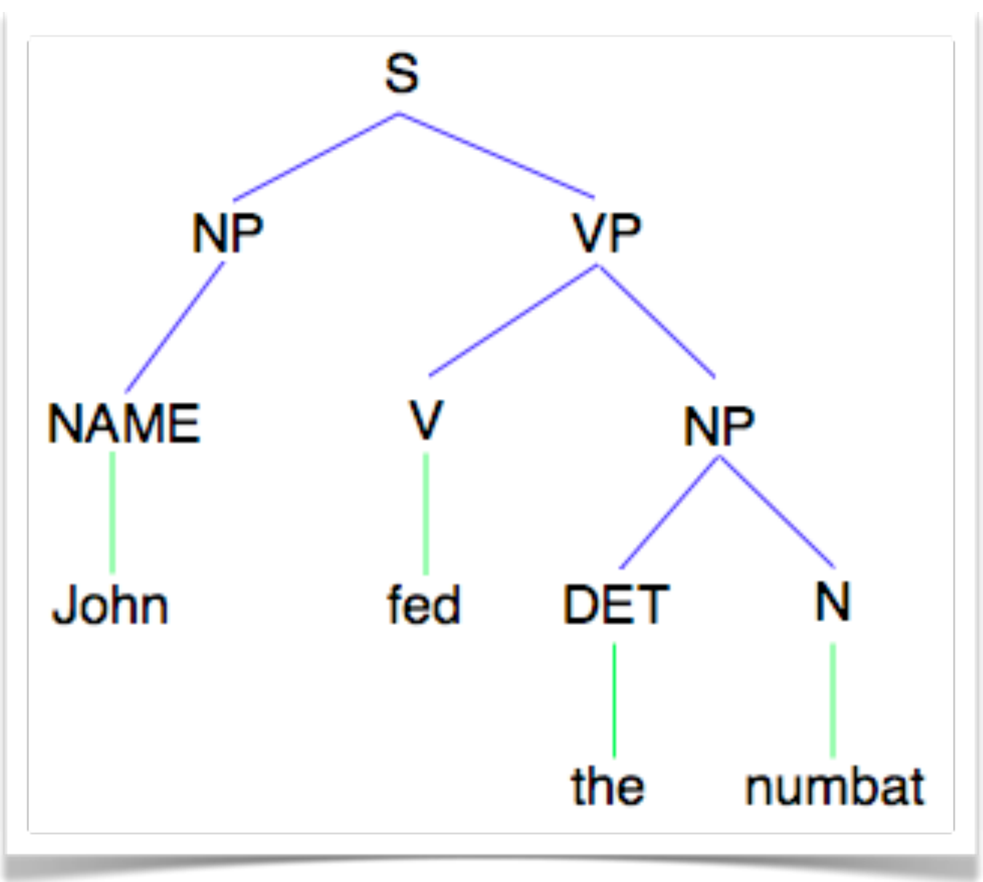
KERNELS FOR STRUCTURES

Computation of the **dot product** between (Graph) Structures



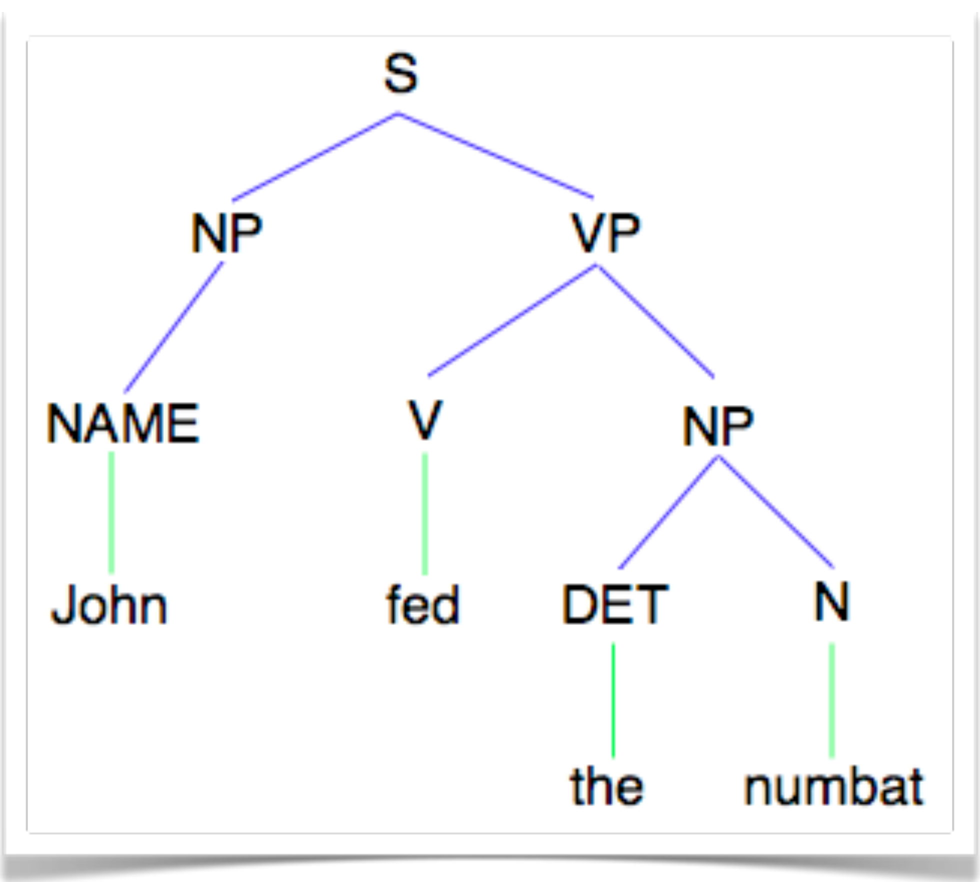
KERNELS FOR LANGUAGES

- Parse Trees represent the syntactic structure of a sentence
- Tree Kernels can be used to measure the similarity between parse trees



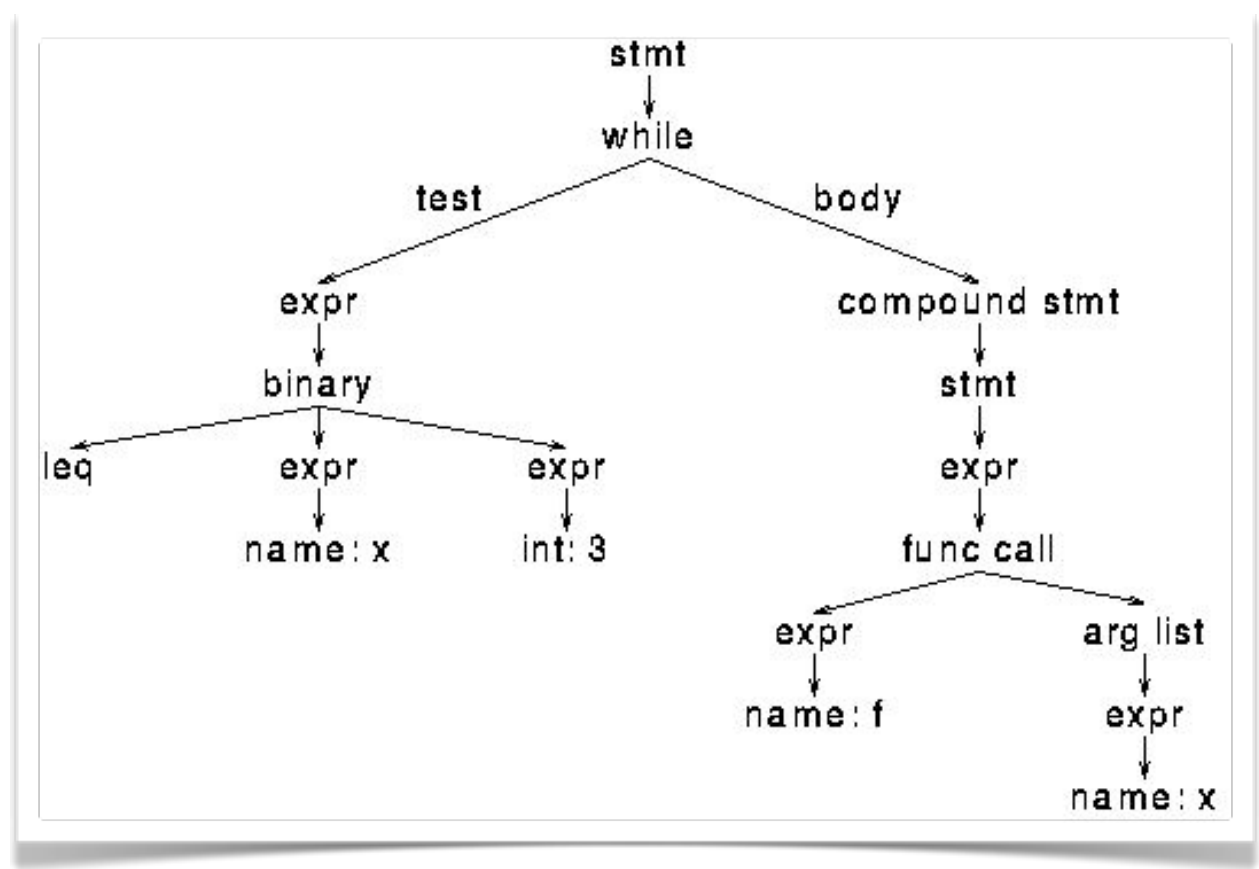
KERNELS FOR LANGUAGES

- Parse Trees represent the syntactic structure of a sentence
- Tree Kernels can be used to measure the similarity between parse trees



KERNELS FOR SOURCE CODE

- Abstract Syntax Trees (AST) represent the syntactic structure of a piece of code
- Research on Tree Kernels for NLP carries over to AST *(with adjustments)*



KERNELS FOR PARSE TREE

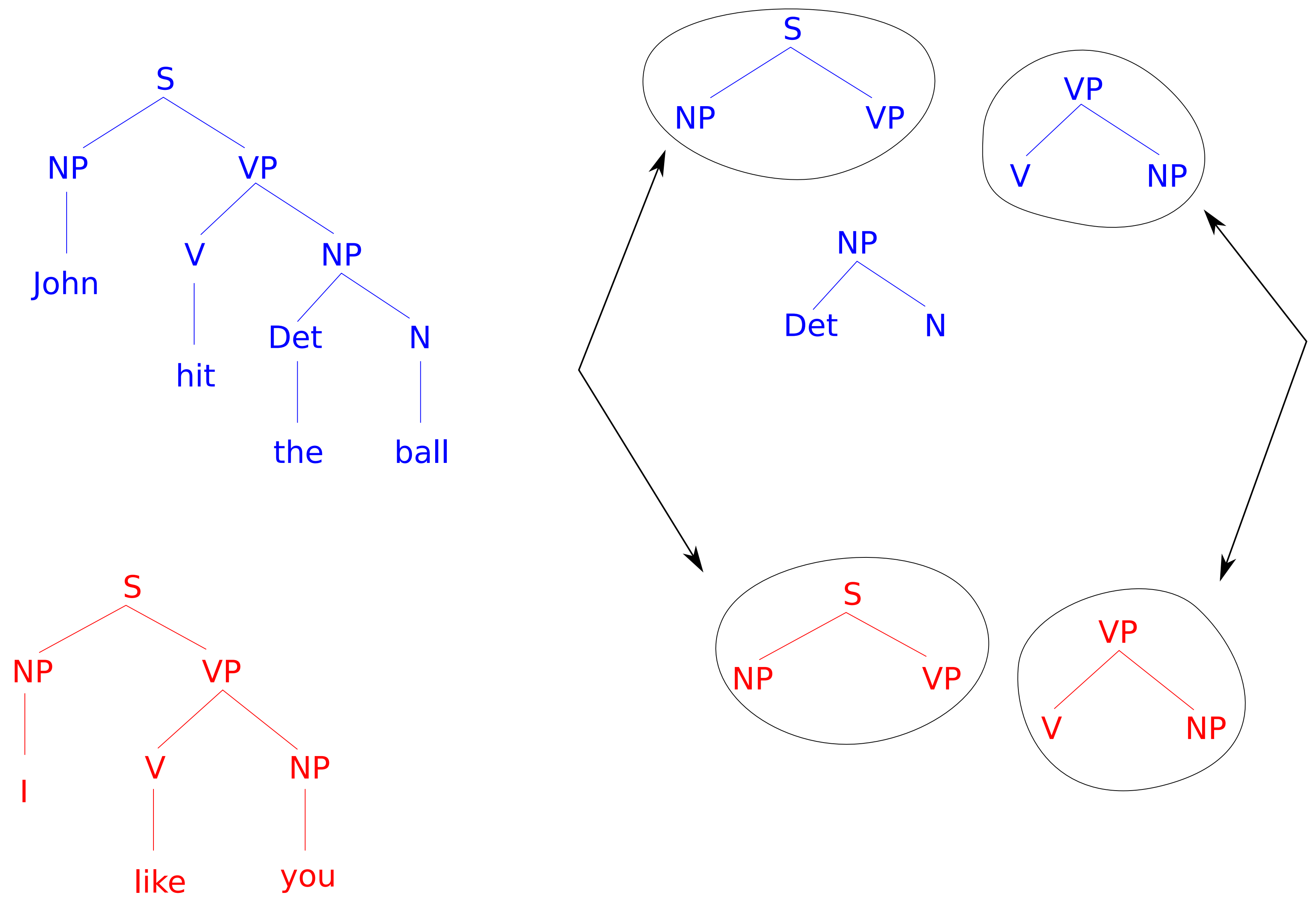
Sentence

Parse Tree

Parse Tree Kernel

John hit the ball

I like you



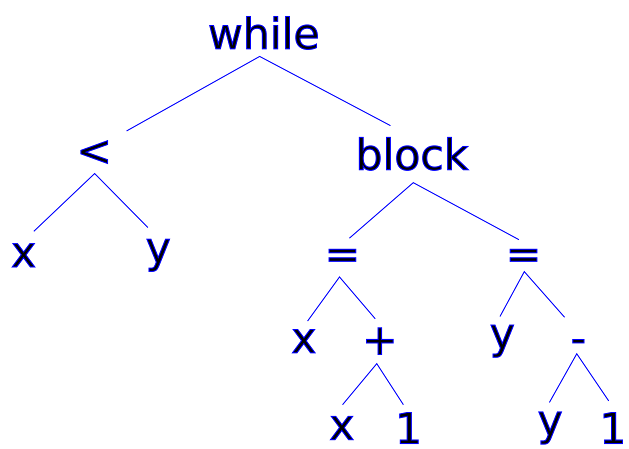
KERNELS FOR AST

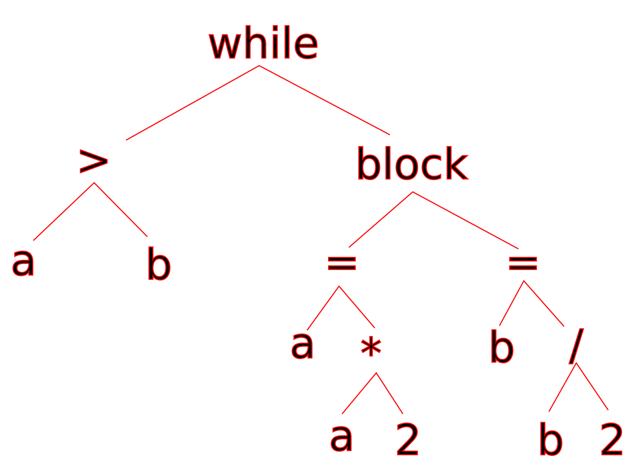
CODE

```
while (x < y) {
   x = x + 1
   y = y - 1
}
```

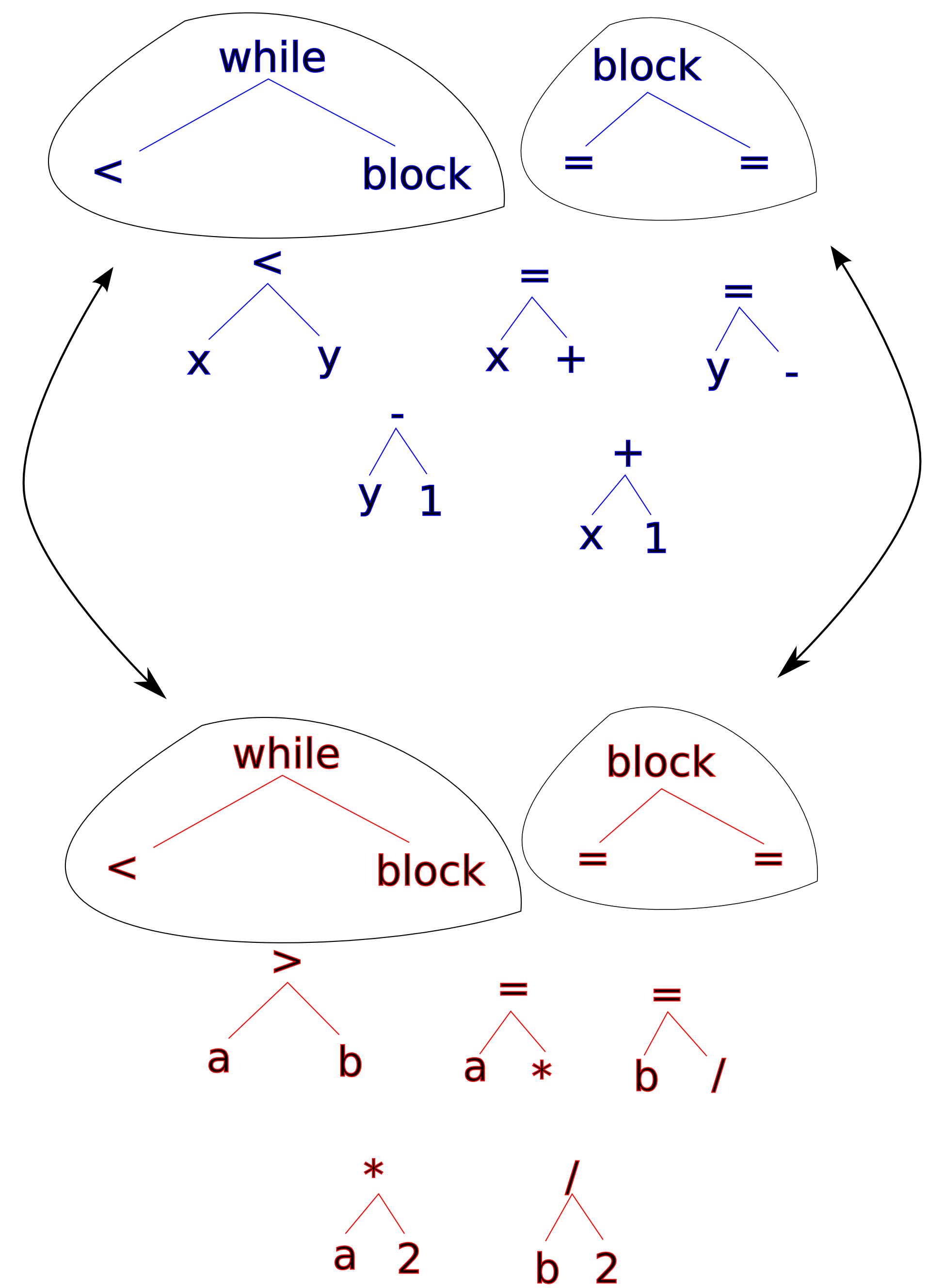
```
while (a < b) {
   a = a * 2
   b = b / 2
}
```

AST





AST KERNEL



KERNEL MACHINES

Idea: Any learning algorithm relying on similarity measure can be used

- **Supervised Learning**
  - Binary Classification
  - Multi-class Classification
  - Ranking
- **Unsupervised Learning**
  - Clustering
  - Anomaly Detection

KERNEL MACHINES FOR CONE DETECTION

- **Supervised Learning**
  - Pairwise classifier: predict if a pair of fragments is clone
- **Unsupervised Learning**
  - Clustering: cluster together all candidate clones

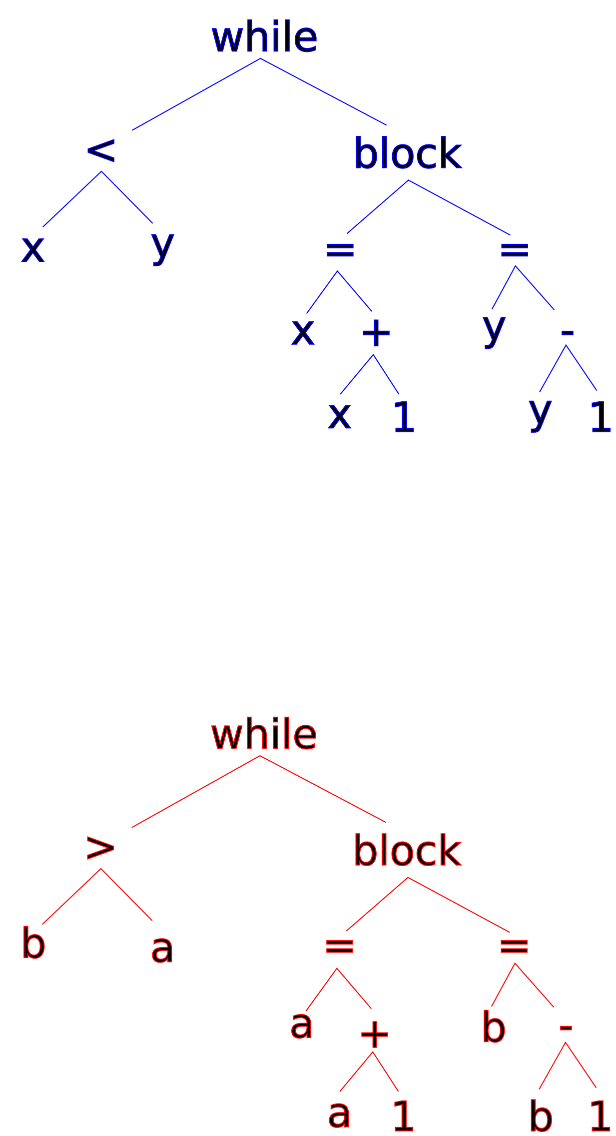
KERNEL FOR CLONES

CODE

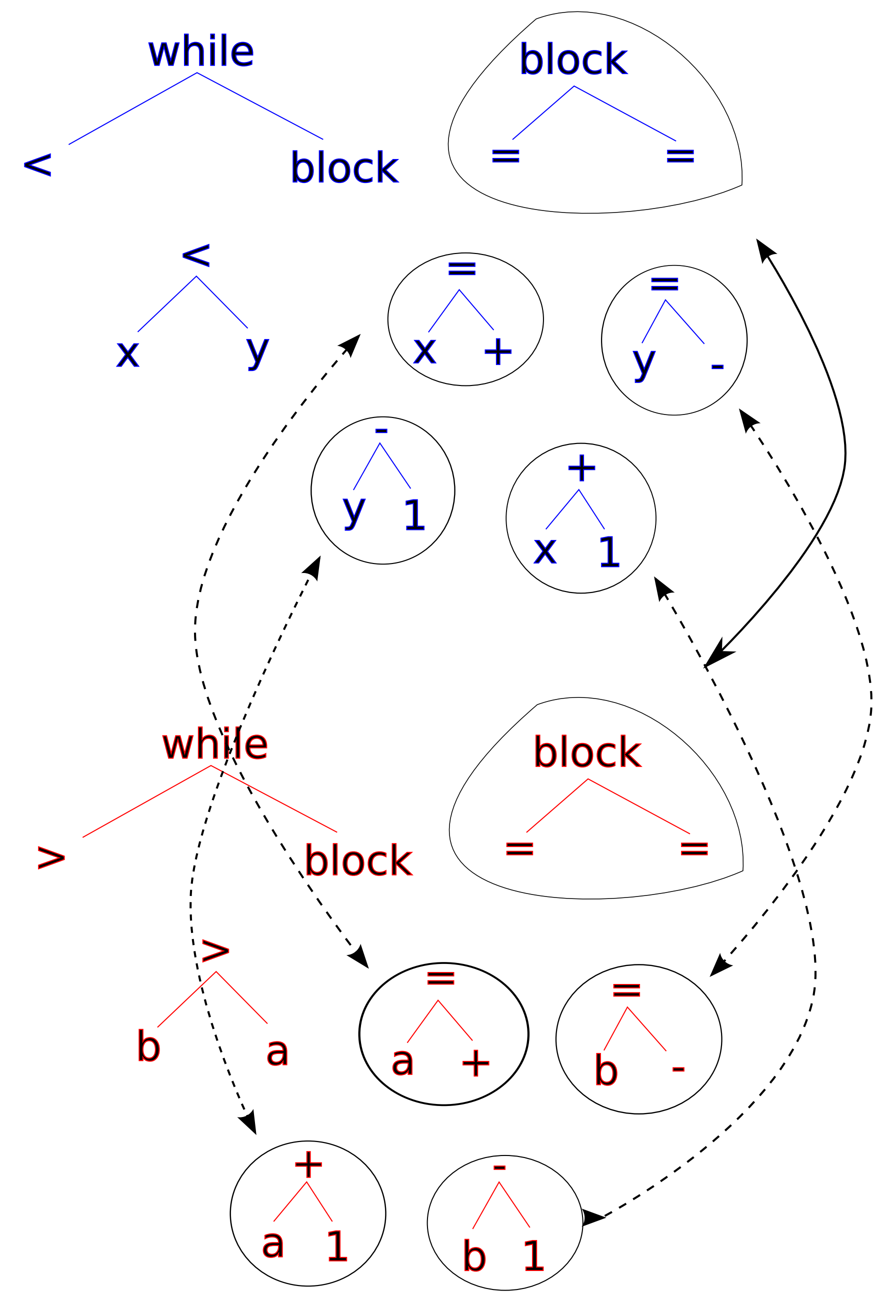
```
while (x < y) {
  x = x + 1
  y = y - 1
}
```

```
while (b > a) {
  a = a + 1
  b = b - 1
}
```

AST



AST KERNEL



LEARNING SIMILARITIES

KERNEL LEARNING

- Construct a number of candidate kernels with different characteristics
  - *e.g., Ignore variables names or not*
- Employ kernel learning approaches which learn a weighted combination of candidate kernels
- Useless/harmful kernels will get zero weight and will be discarded in the final model

STRUCTURED-OUTPUT LEARNING

Supervised Clustering

- Exploit information on already annotated pieces of software
- Training examples are software projects/portions with annotation on existing clones (clustering)
- A learning model uses training examples to refine the similarity measure for correctly clustering novel examples

SUMMARY

- Software has a rich structure and heterogeneous information
- Advanced Machine learning approaches are promising for exploiting such information
- **Kernel Methods** are natural candidate
  - e.g., see the analogy between NLP parse trees and AST
- Many applications:
  - architecture recovery, code clone detection, vulnerability detection ....

CASE STUDY: KERNELS FOR CLONES

CODE CLONE DETECTION

```
d_setitem(arrayobject *ap, Py_ssize_t i, PyObject *v)
{
        double x;
        if (!PyArg_Parse(v, "d;array item must be float", &x))
                return -1;
        if (i >= 0)
                    ((double *)ap->ob_item)[i] = x;
        return 0;
}
```

```
i_setitem(arrayobject *ap, Py_ssize_t i, PyObject *v)
{
        int x;
        /* 'i' == signed int, maps to PyArg_Parse's 'i' formatter */
        if (!PyArg_Parse(v, "i;array item must be integer", &x))
                return -1;
        if (i >= 0)
                    ((int *)ap->ob_item)[i] = x;
        return 0;
}
```

- **Goal**: *"Identify and group all duplicated code fragments/functions"*
  - *Copy&Paste programming*
- Taxonomy of **4 different types of clones**
  - *Program Text* similarities and *Functional* similarities
- Clones affect the reliability and the maintainability of a software system

KERNELS FOR CLONES

Kernels for Structured Data:

- The source code could be represented by many different data structures

- **Abstract Syntax Tree** (AST)
- Tree structure representing the syntactic structure of the different instructions of a program (function)

- **Program Dependencies Graph** (PDG)
  - (Directed) Graph structure representing the relationship among the different statement of a program

CODE
STRUCTURES
AST

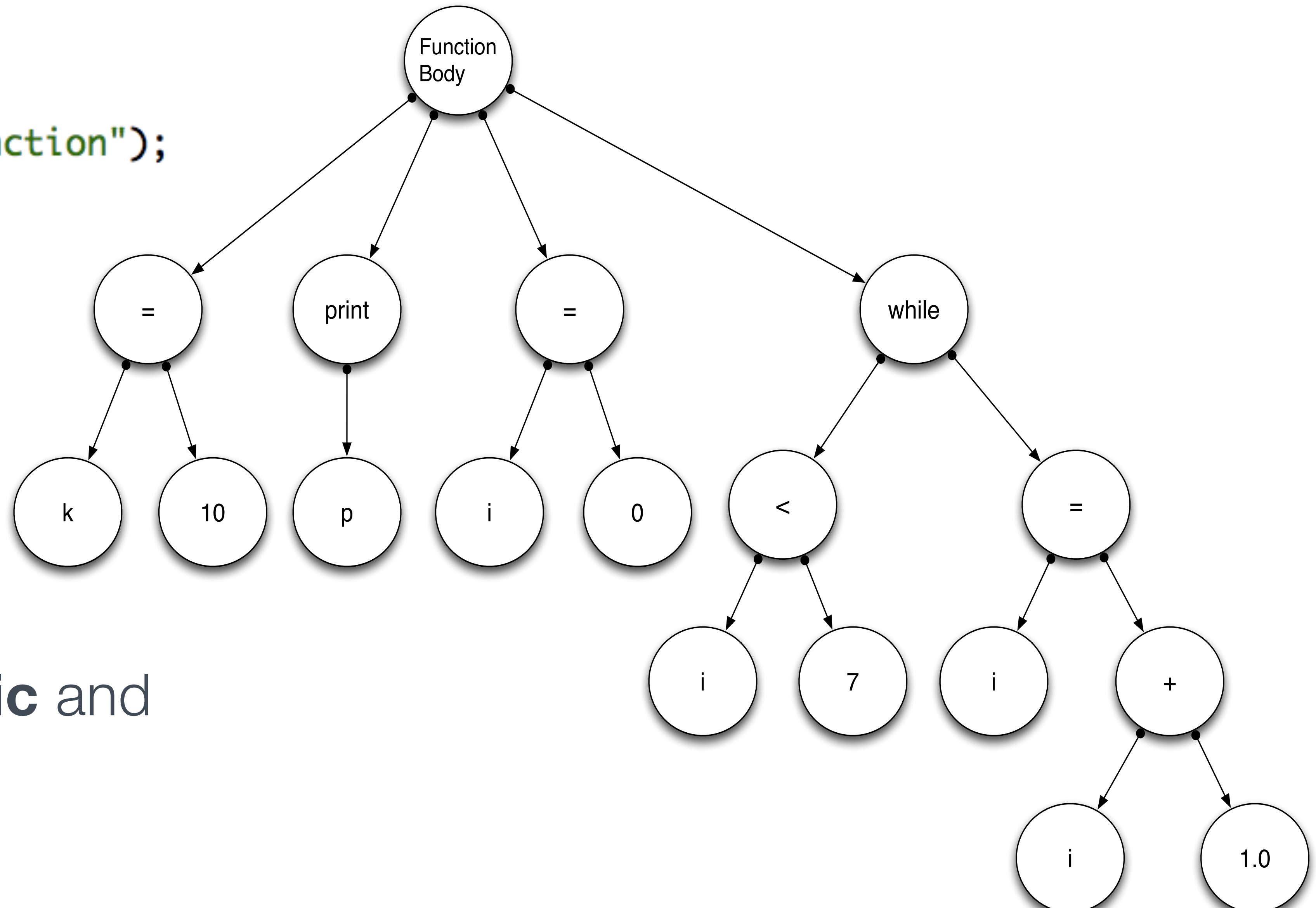
ABSTRACT SYNTAX TREE (AST)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



AST embeds both **Syntactic** and **Lexical** Information

- Program Instructions
- Name of Variables, Literals...

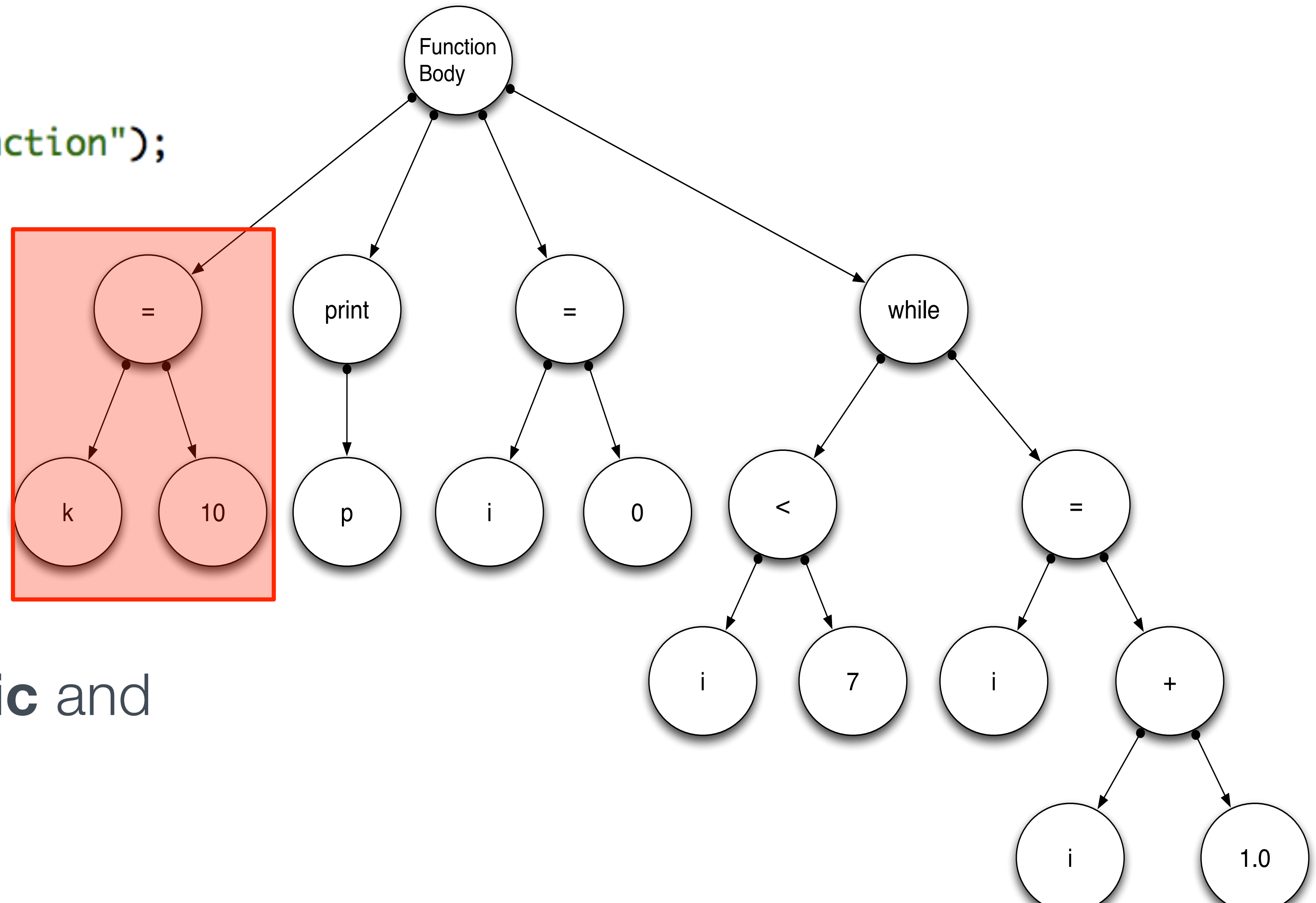
ABSTRACT SYNTAX TREE (AST)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



AST embeds both **Syntactic** and **Lexical** Information

- Program Instructions
- Name of Variables, Literals...

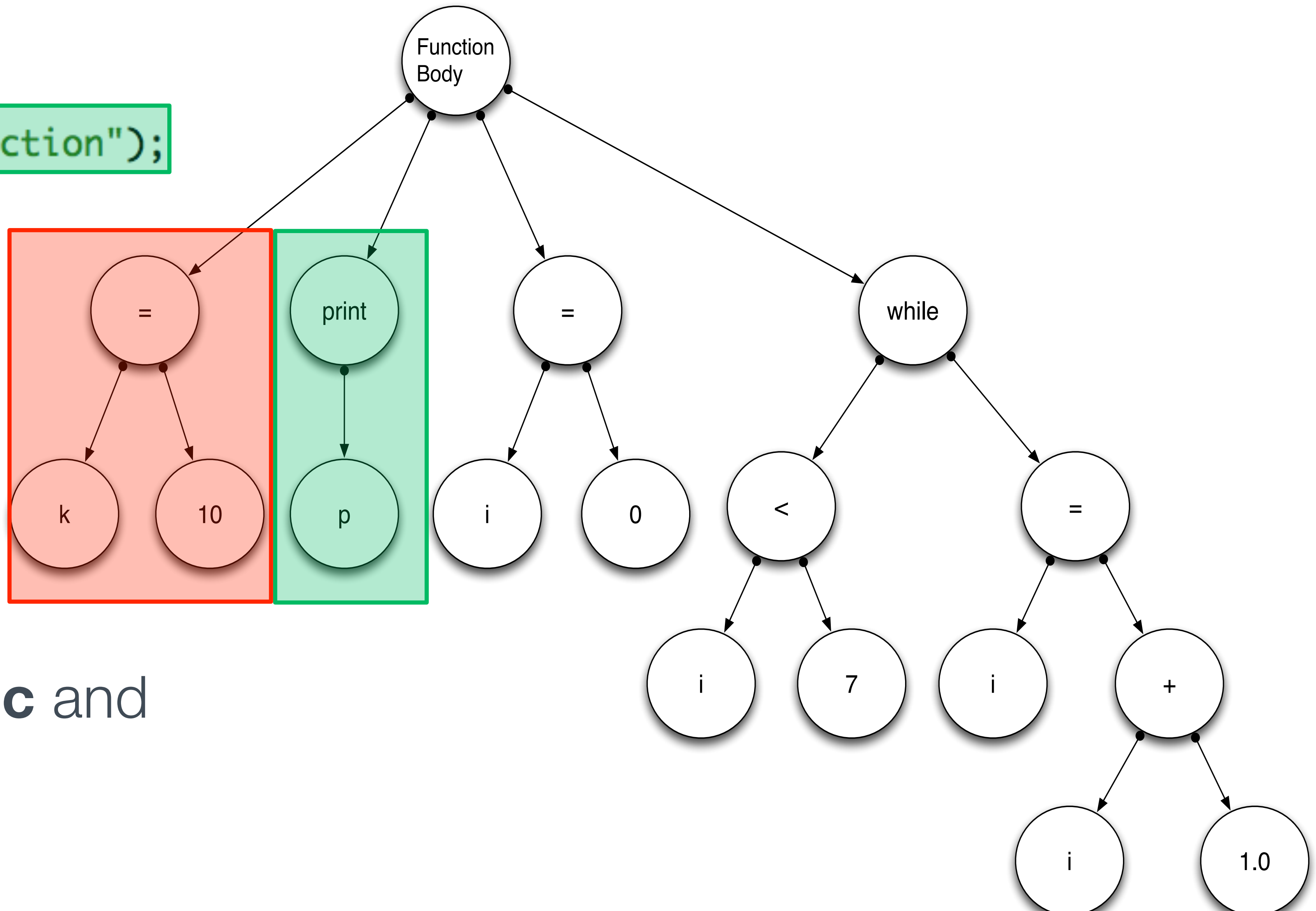
ABSTRACT SYNTAX TREE (AST)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



AST embeds both **Syntactic** and **Lexical** Information

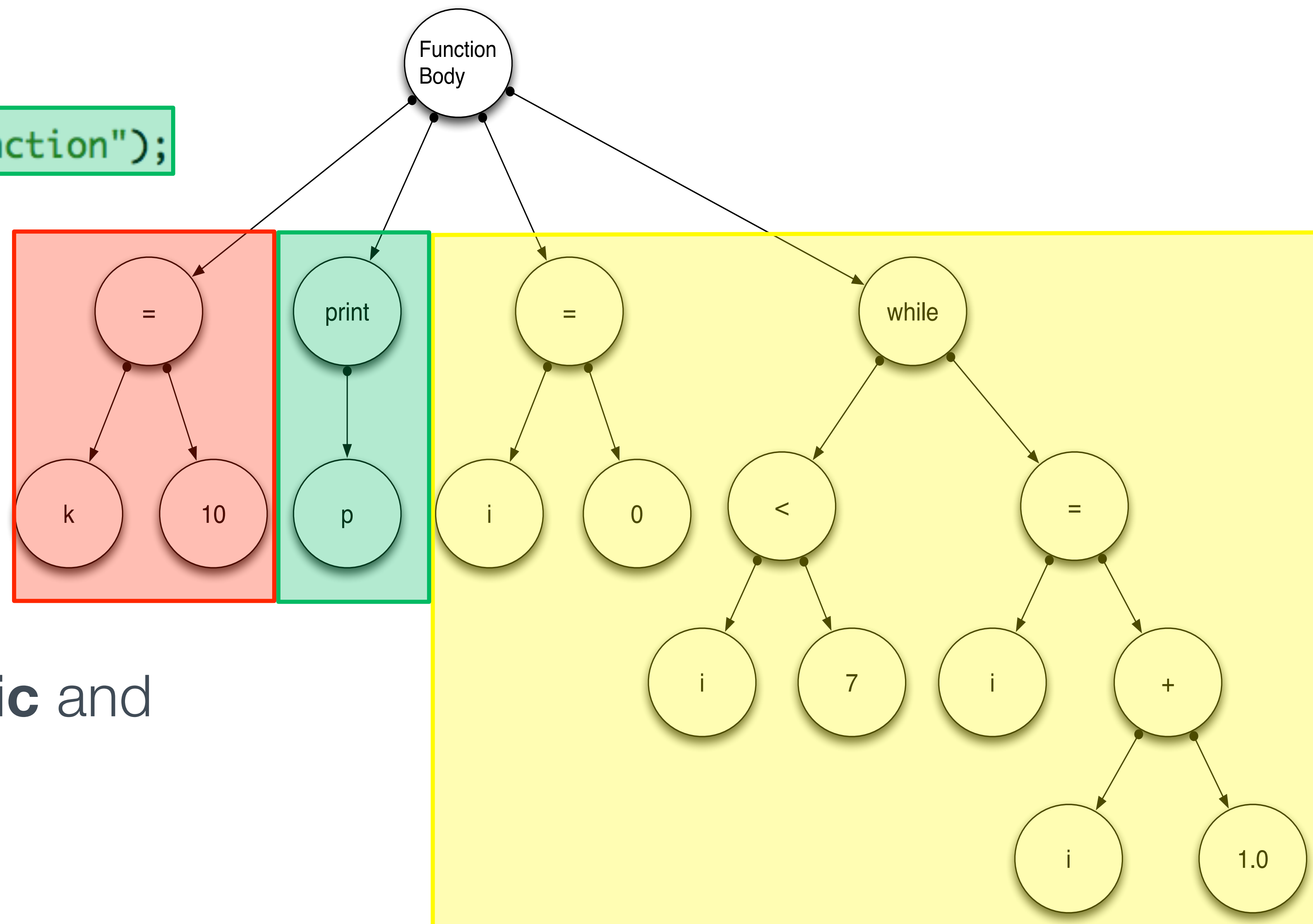
- Program Instructions
- Name of Variables, Literals...

ABSTRACT SYNTAX TREE (AST)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}
```



AST embeds both **Syntactic** and **Lexical** Information

- Program Instructions
- Name of Variables, Literals...

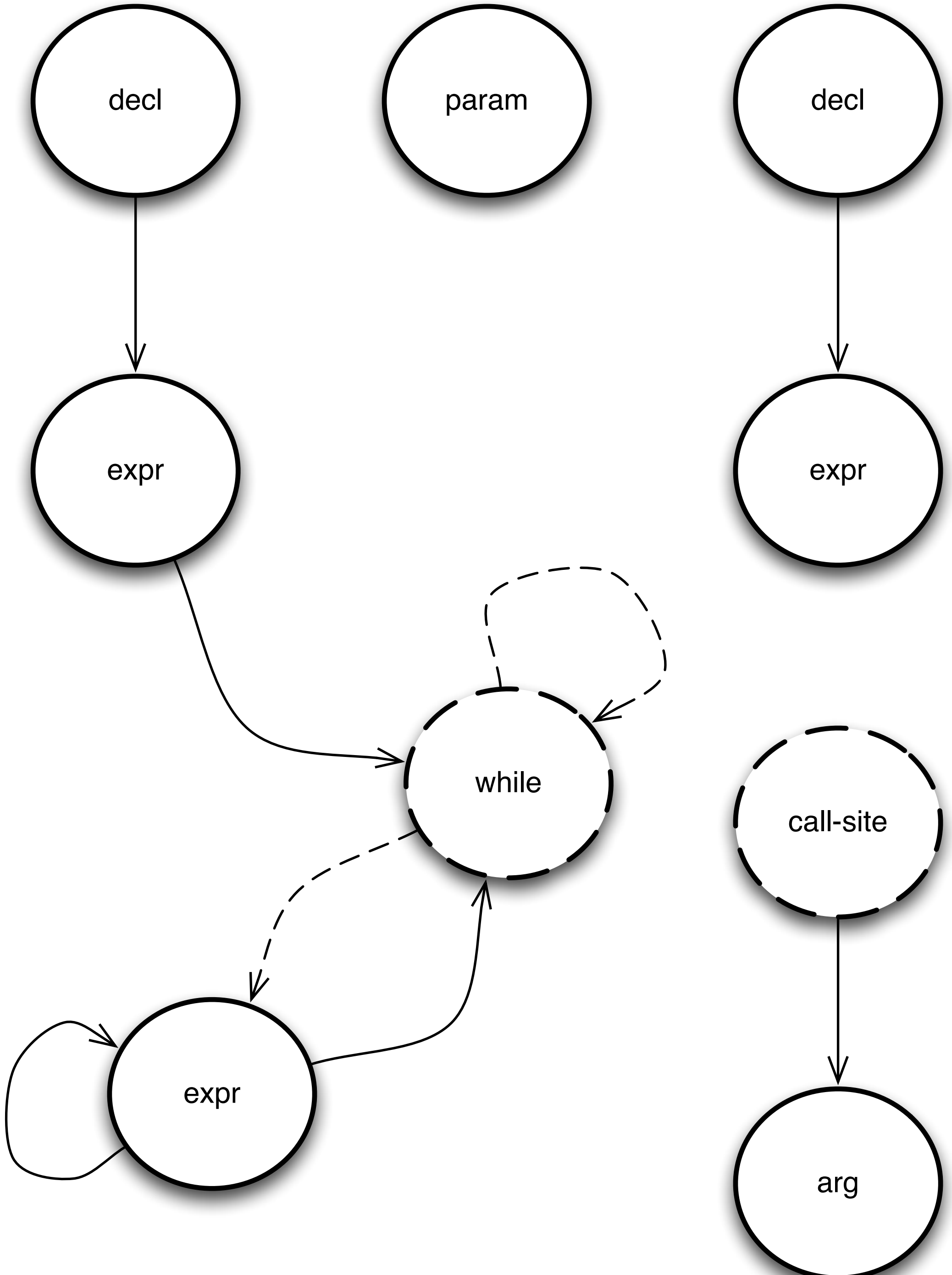
PROGRAM DEPENDENCIES GRAPH (PDG)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes

CODE
STRUCTURES
PDG

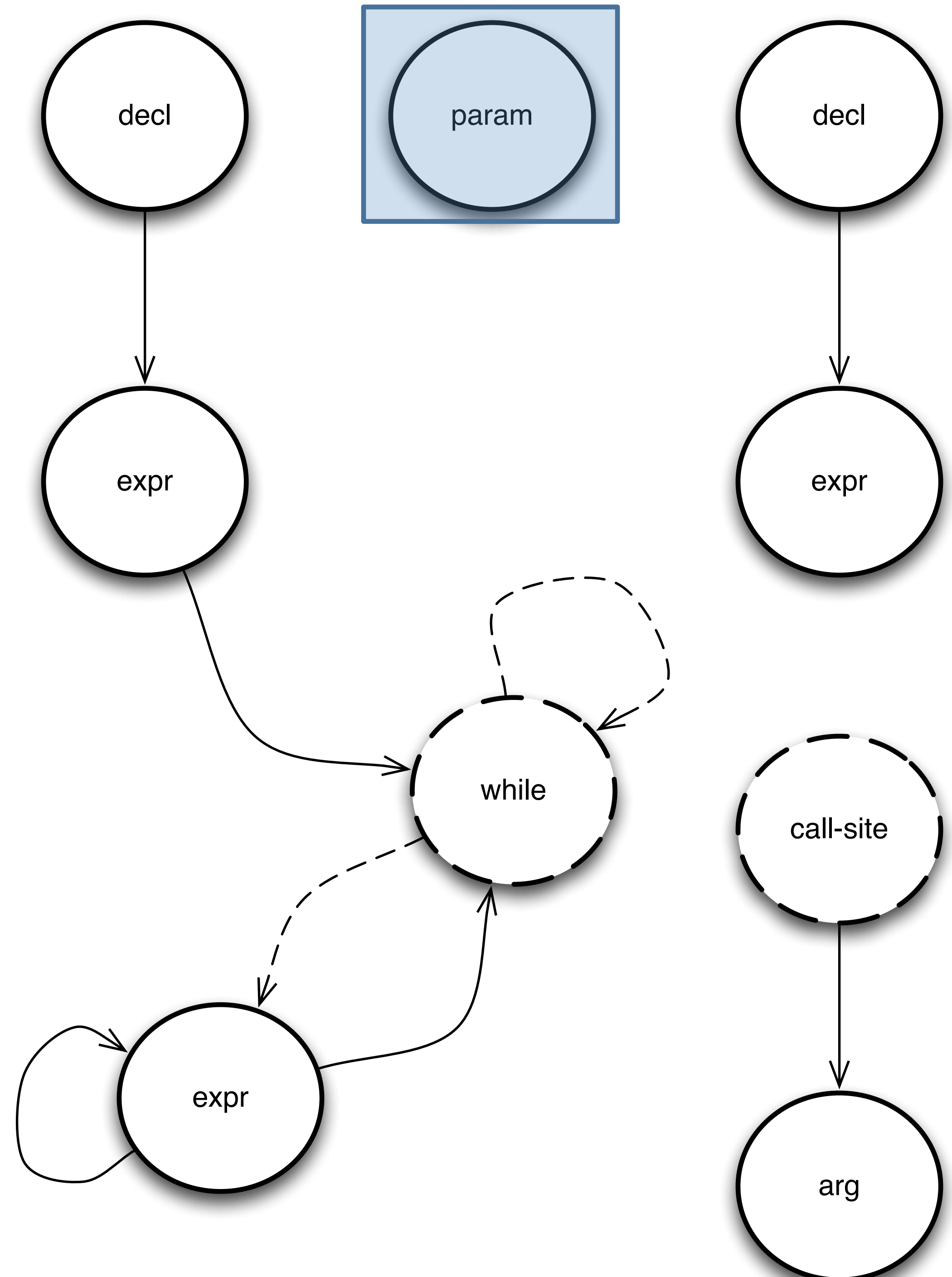
PROGRAM DEPENDENCIES GRAPH (PDG)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes

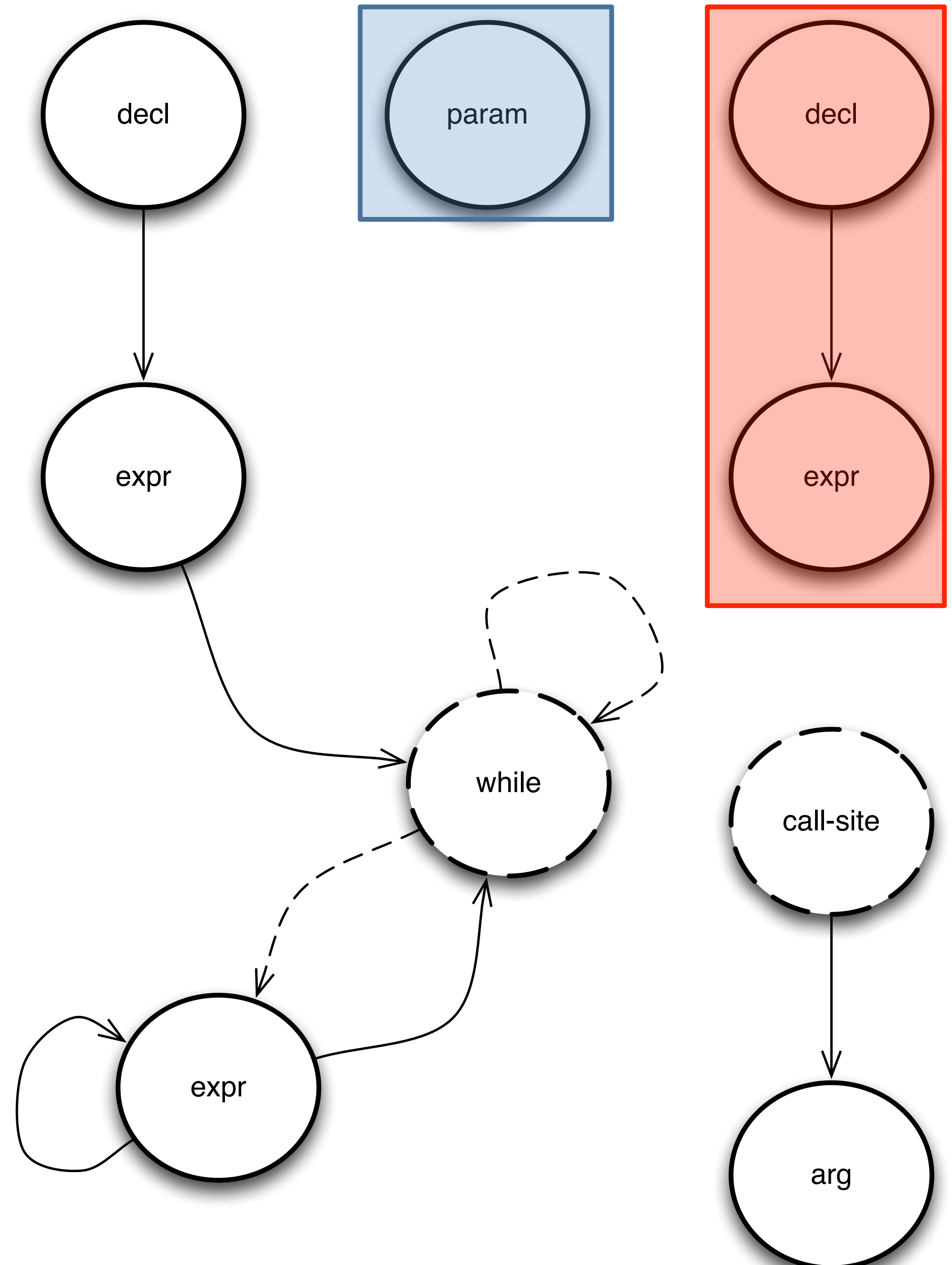
PROGRAM DEPENDENCIES GRAPH (PDG)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes

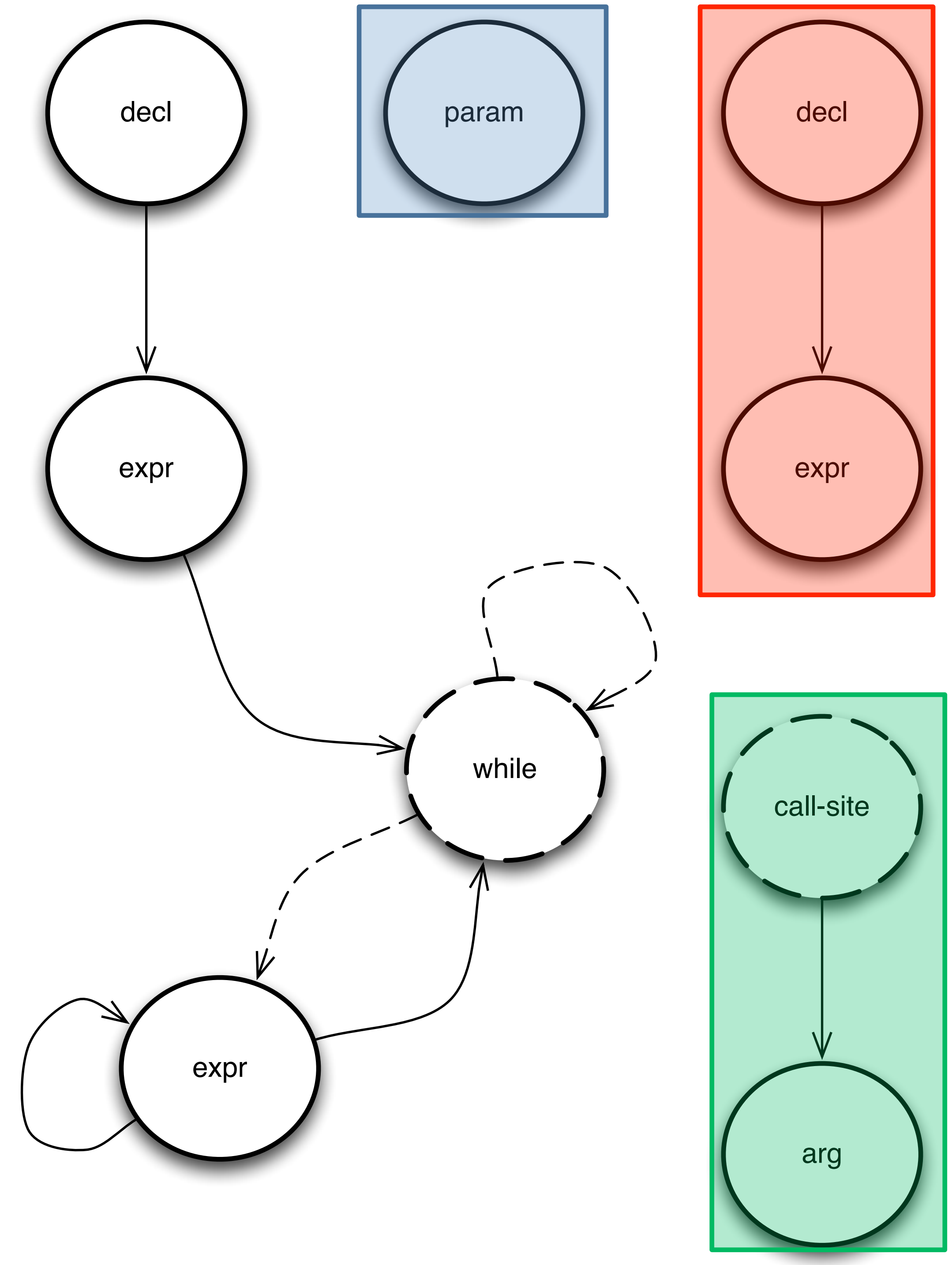
PROGRAM DEPENDENCIES GRAPH (PDG)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```



- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes

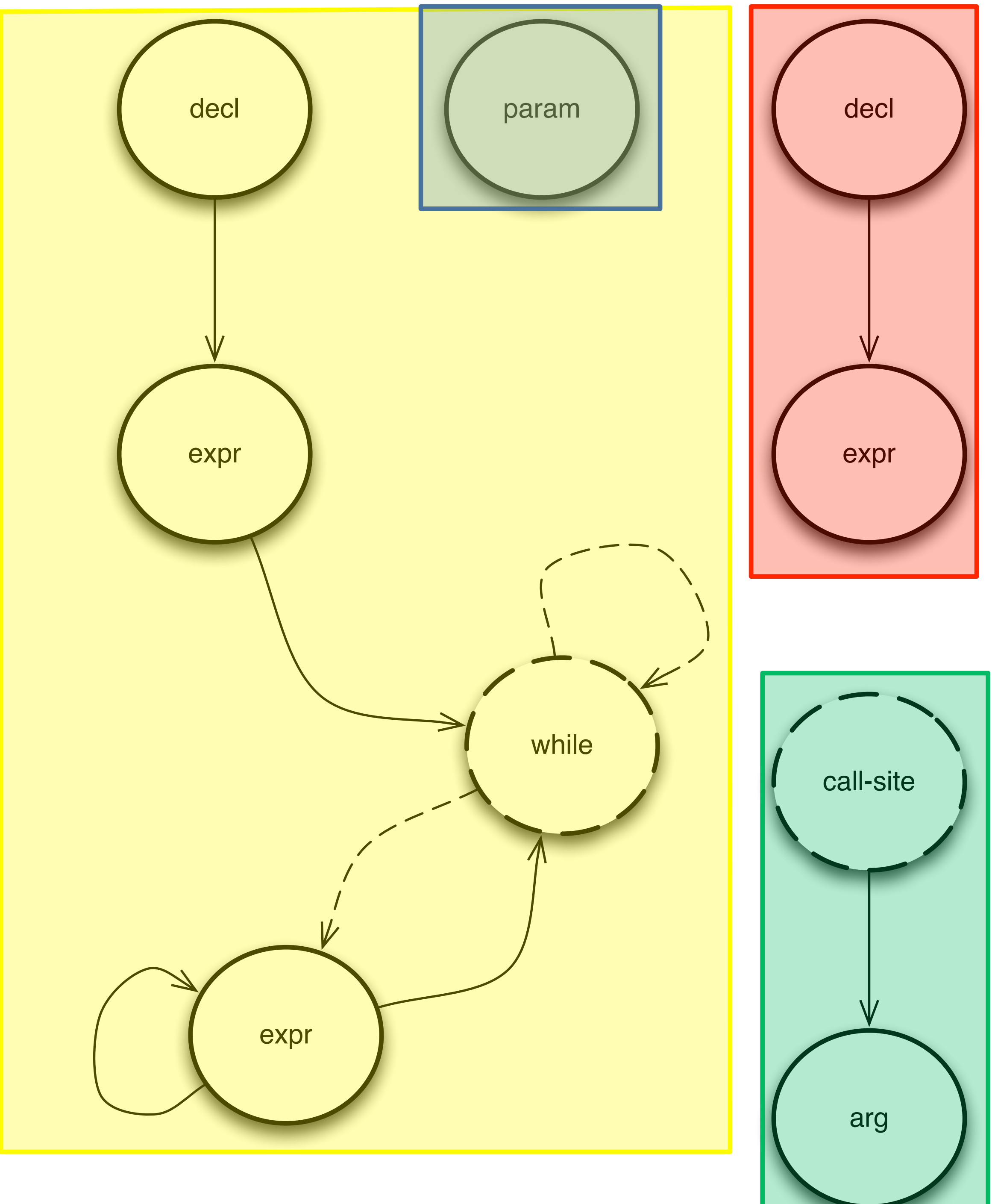
PROGRAM DEPENDENCIES GRAPH (PDG)

```
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }

}
```

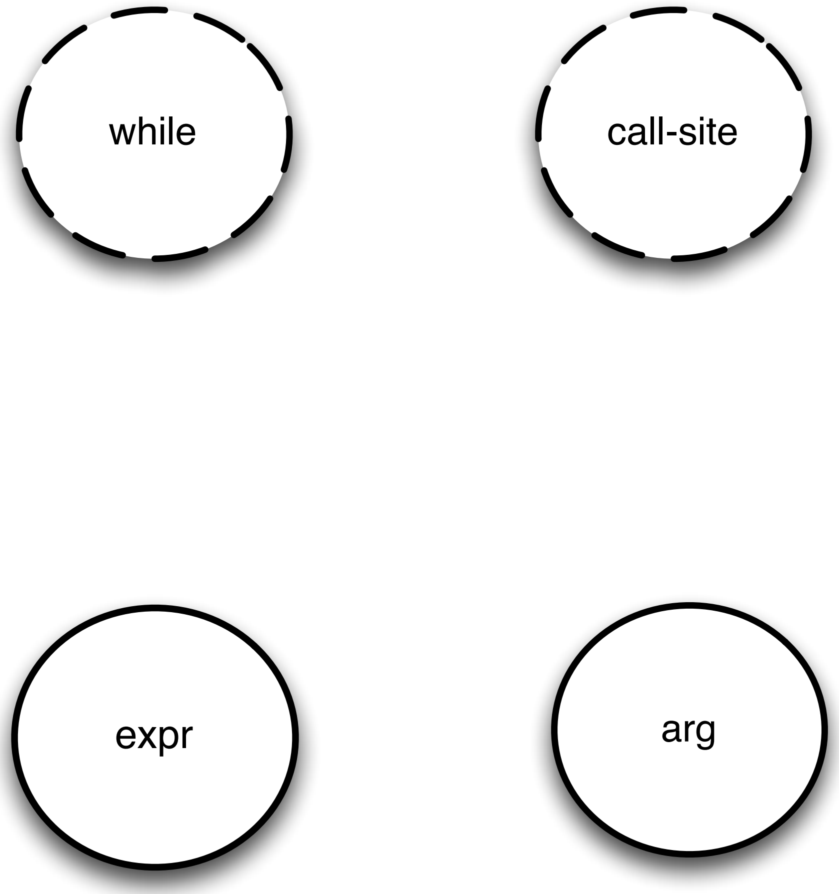


- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes

CODE
STRUCTURES
PDG

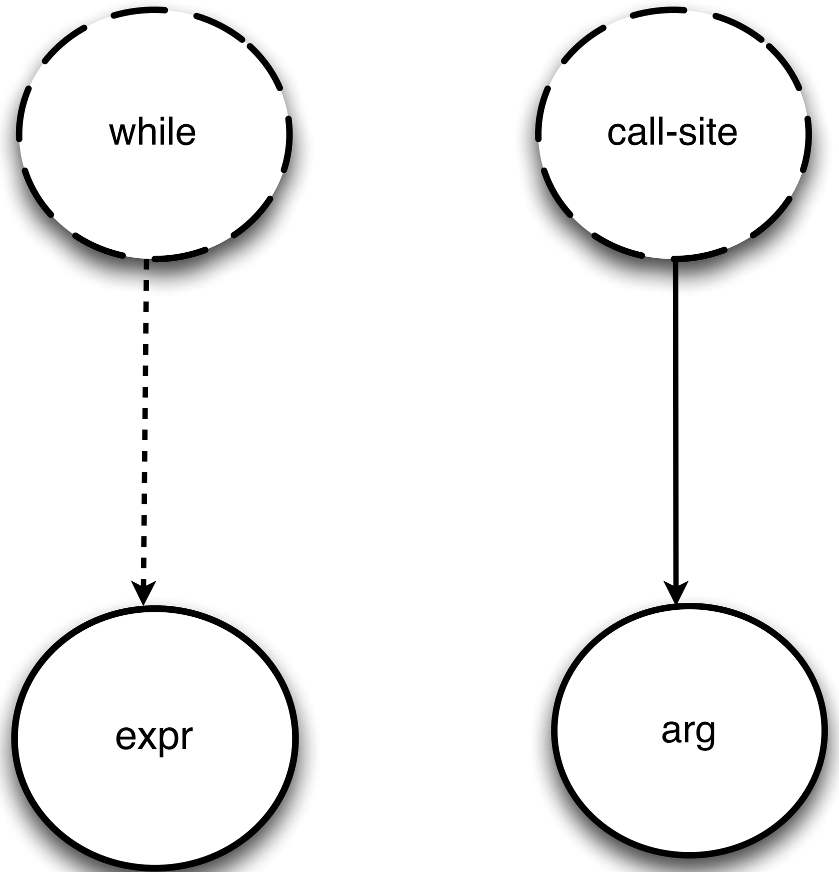
NODES AND EDGES

- **Two Types** of Nodes
  - *Control Nodes* (Dashed ones)
    - e.g., `if - for - while - function calls...`
  - *Data Nodes*
    - e.g., `expressions - parameters...`



NODES AND EDGES

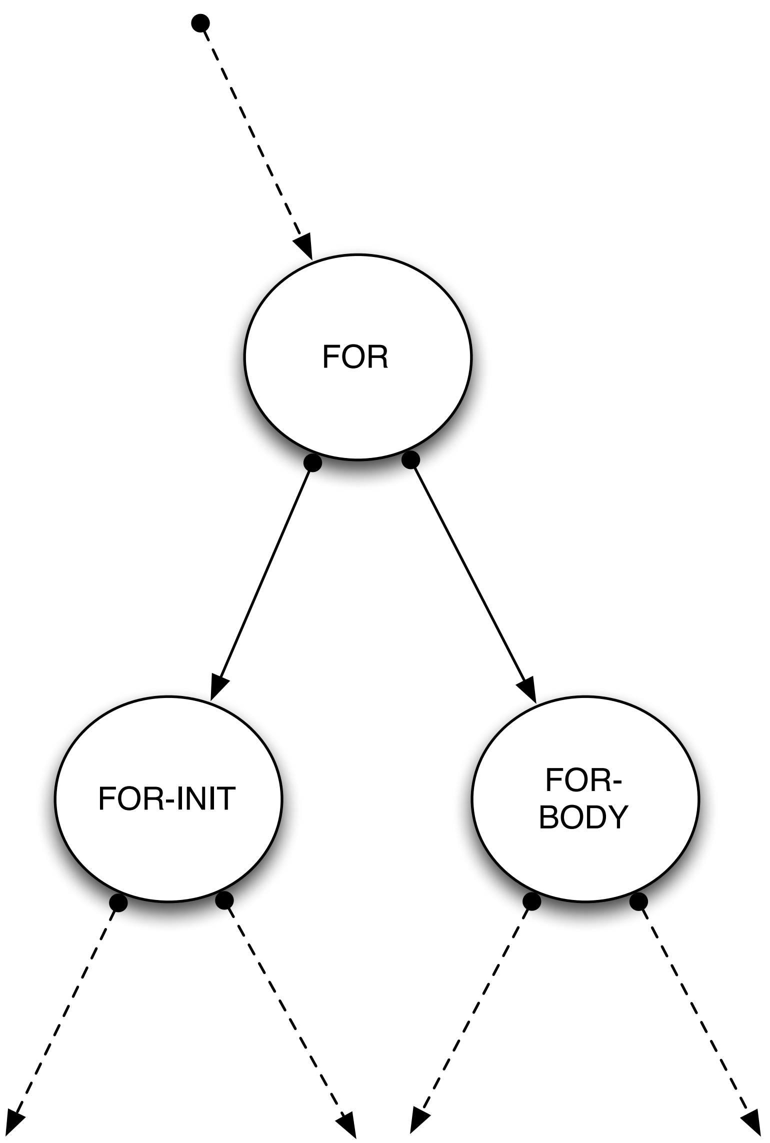
- **Two Types** of Nodes
  - *Control Nodes* (Dashed ones)
    - e.g., `if - for - while - function calls...`
  - *Data Nodes*
    - e.g., `expressions - parameters...`
- **Two Types** of Edges (i.e., *dependencies*)
  - *Control edges* (Dashed ones)
  - *Data edges*



DEFINING KERNELS FOR STRUCTURED DATA

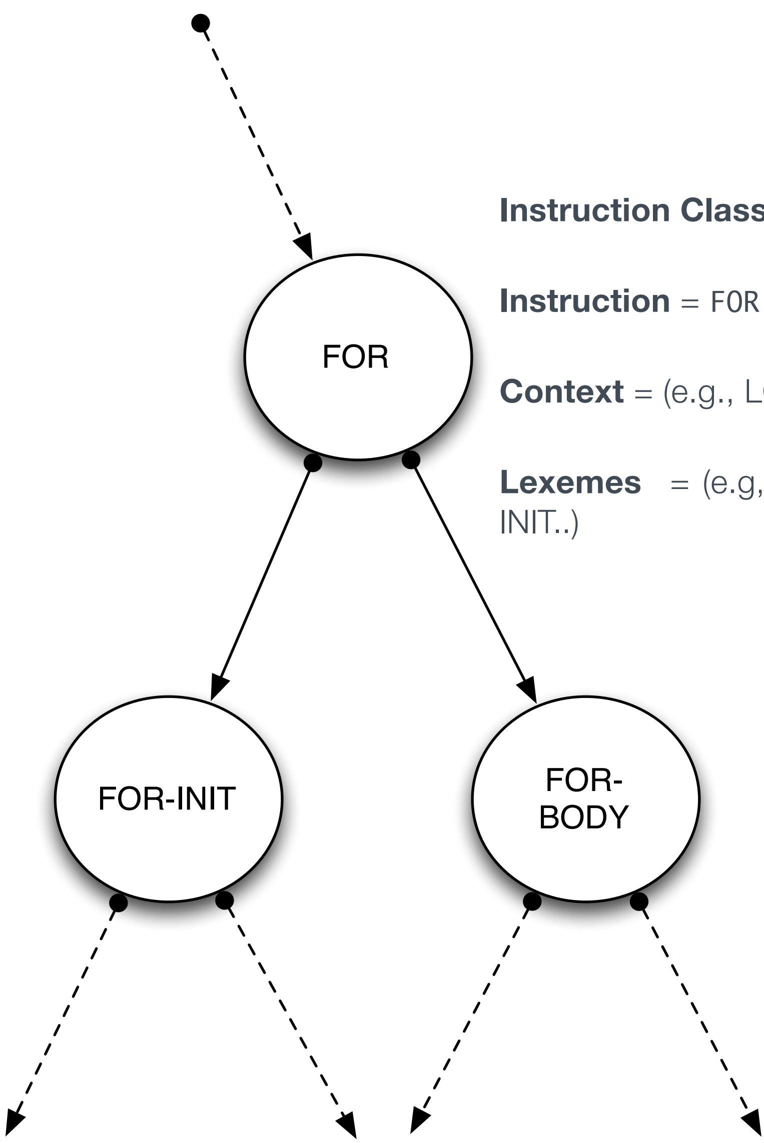
- The definition of a new Kernel for a Structured Object requires the definition of:
- Set of features to annotate each part of the object
- A Kernel function to measure the similarity on the smallest part of the object
  - *e.g., Nodes of AST and Graphs*
- A Kernel function to apply the computation on the different (sub)parts of the structured object

TREE KERNELS FOR AST



- **Features**: each node is characterized by a set of 4 features
  - **Instruction Class**
    - i.e., `LOOP, CONDITIONAL_STATEMENT, CALL`
  - **Instruction**
    - i.e., `FOR, IF, WHILE, RETURN`
  - **Context**
    - i.e., Instruction Class of the closer statement node
  - **Lexemes**
    - Lexical information gathered (recursively) from leaves

TREE KERNELS FOR AST



Instruction Class = LOOP

Instruction = FOR

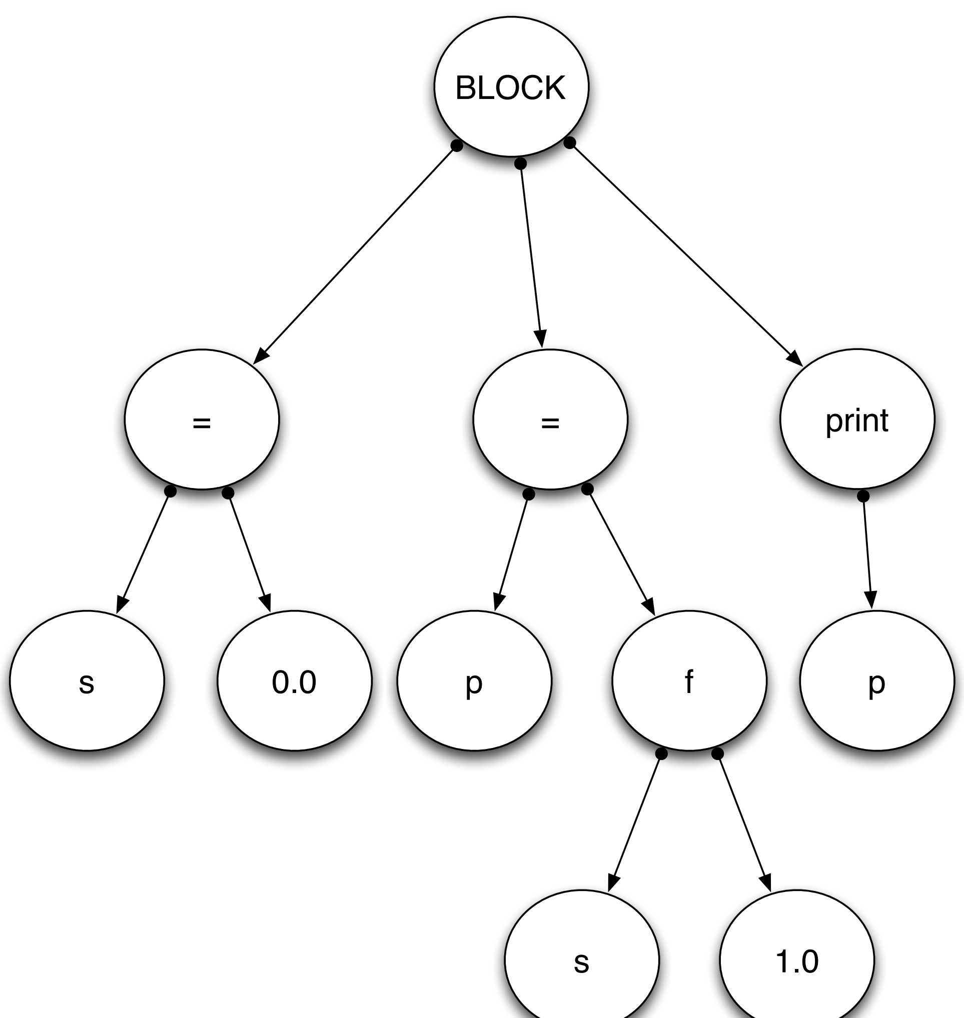
Context = (e.g., LOOP)

Lexemes = (e.g, name of variables in FOR-INIT..)

- **Features**: each node is characterized by a set of 4 features
  - **Instruction Class**
    - i.e., `LOOP, CONDITIONAL_STATEMENT, CALL`
  - **Instruction**
    - i.e., `FOR, IF, WHILE, RETURN`
  - **Context**
    - i.e., Instruction Class of the closer statement node
  - **Lexemes**
    - Lexical information gathered (recursively) from leaves

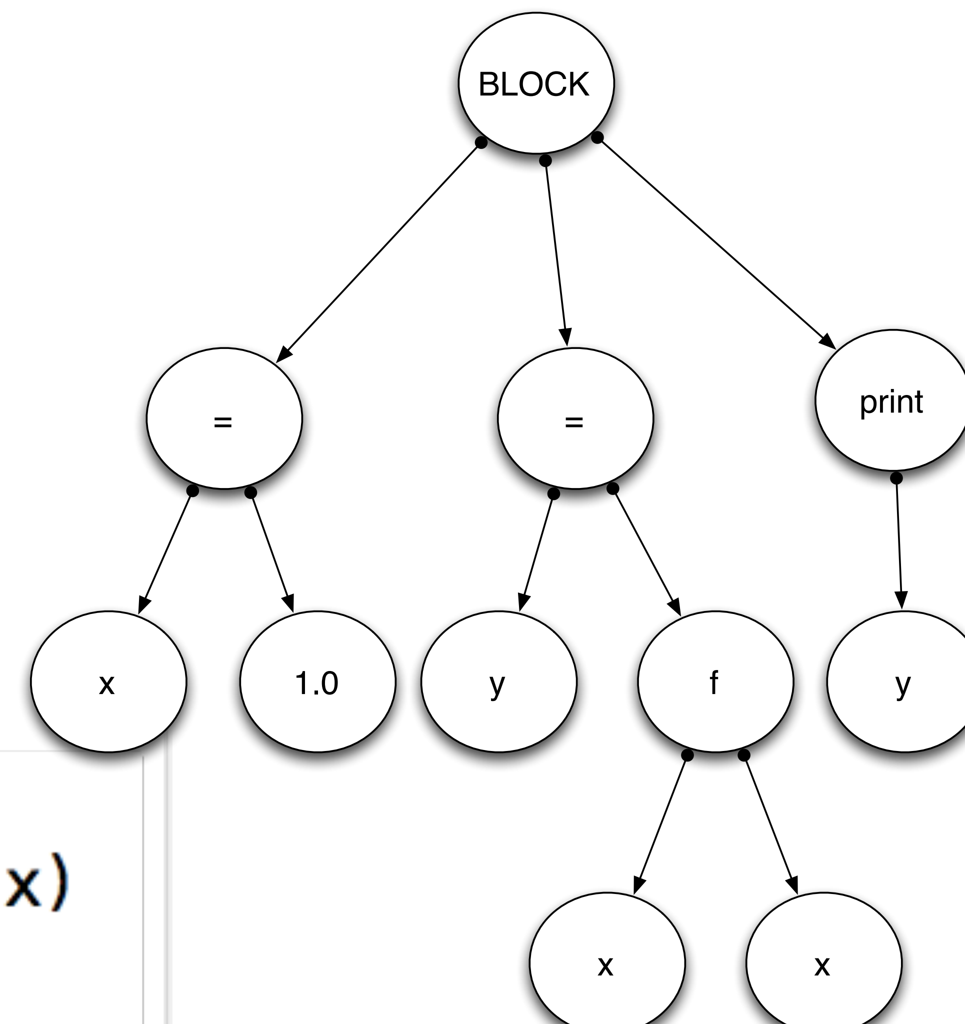
TREE KERNELS FOR AST

- **Goal:** *Identify the maximum isomorphic Tree/Subtree*
- *Comparison of blocks to each other*
- ***Blocks:*** *Atomic unit for (sub) tree considered*



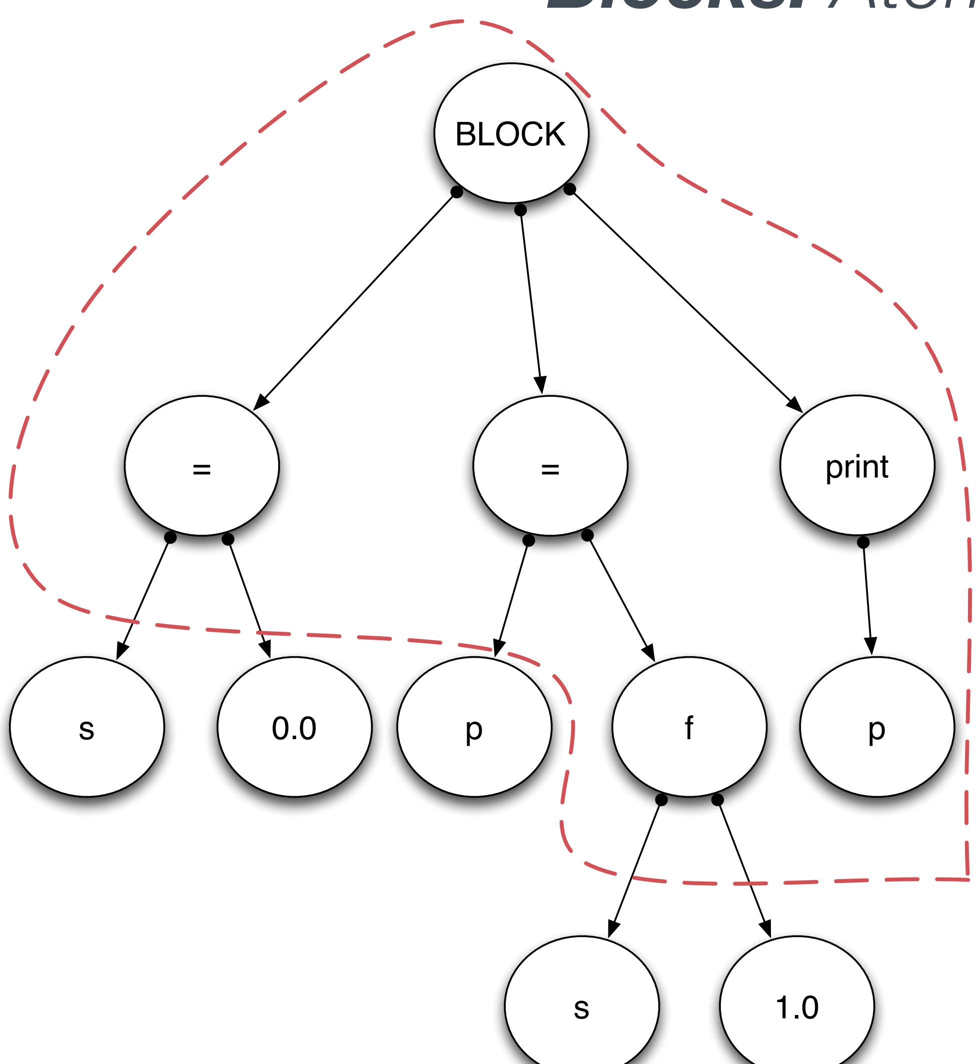

```
s = 0.0
p = f(s, 1.0)
print p
```

```
x = 1.0
y = f(x, x)
print y
```



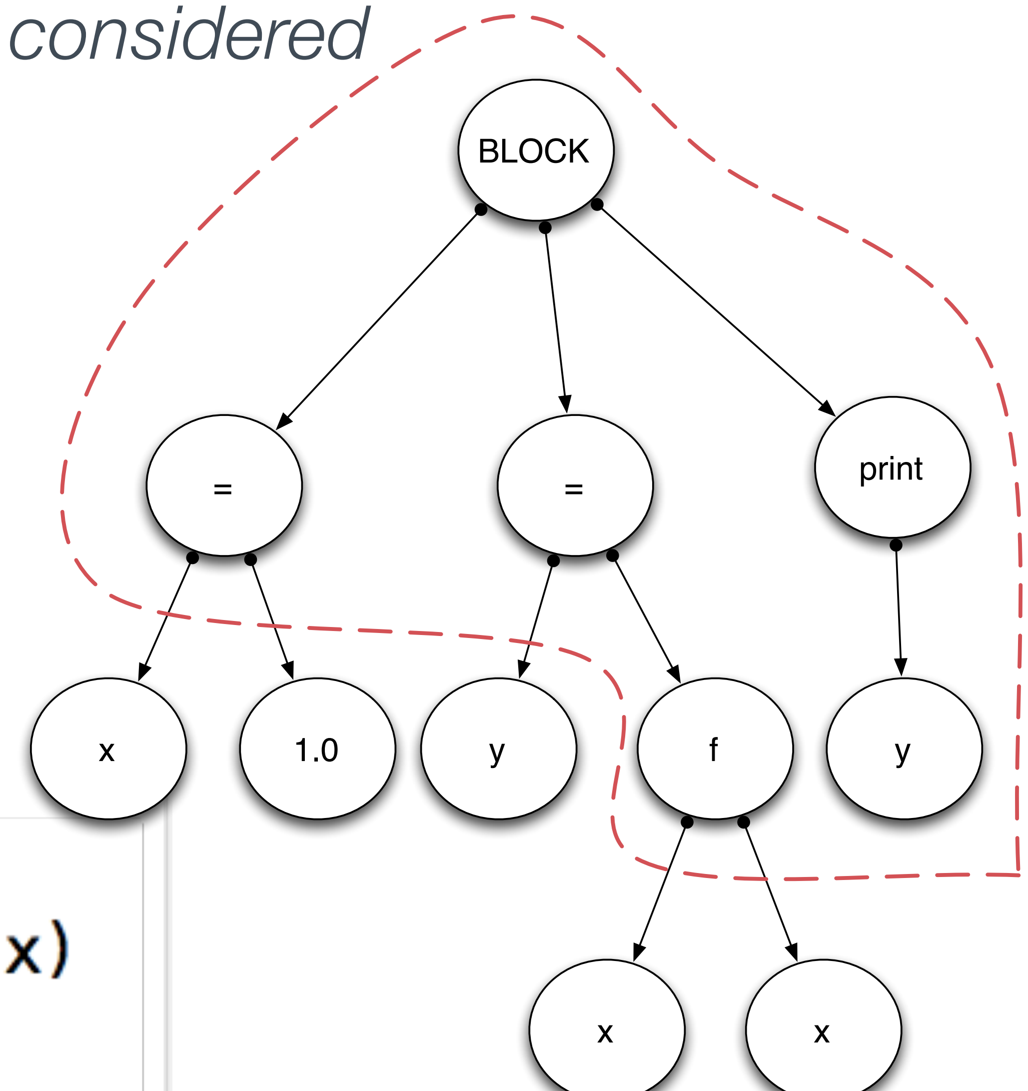
TREE KERNELS FOR AST

- **Goal:** *Identify the maximum isomorphic Tree/Subtree*
- *Comparison of blocks to each other*
- ***Blocks:*** *Atomic unit for (sub) tree considered*

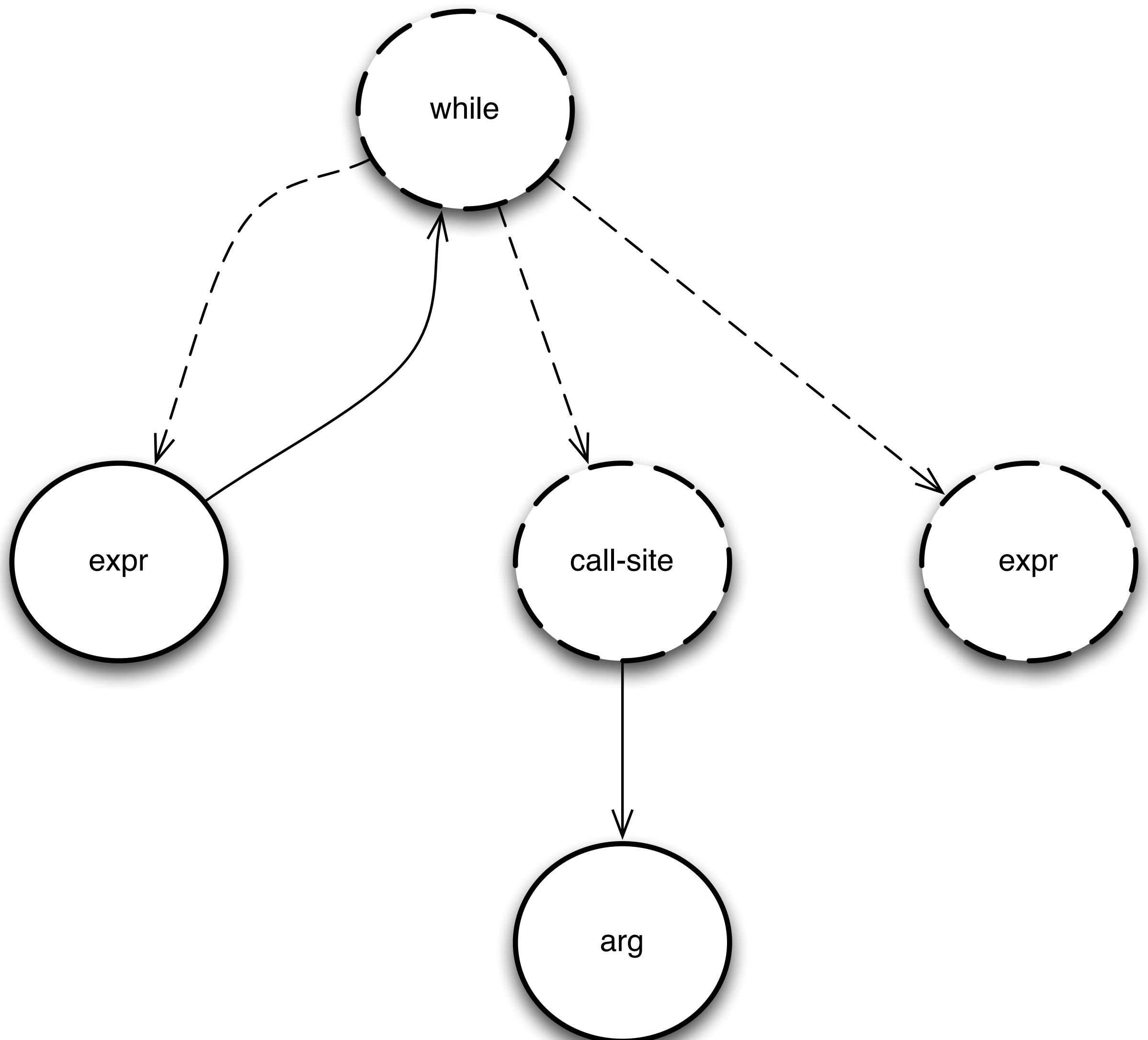



```
s = 0.0
p = f(s, 1.0)
print p
```

```
x = 1.0
y = f(x, x)
print y
```



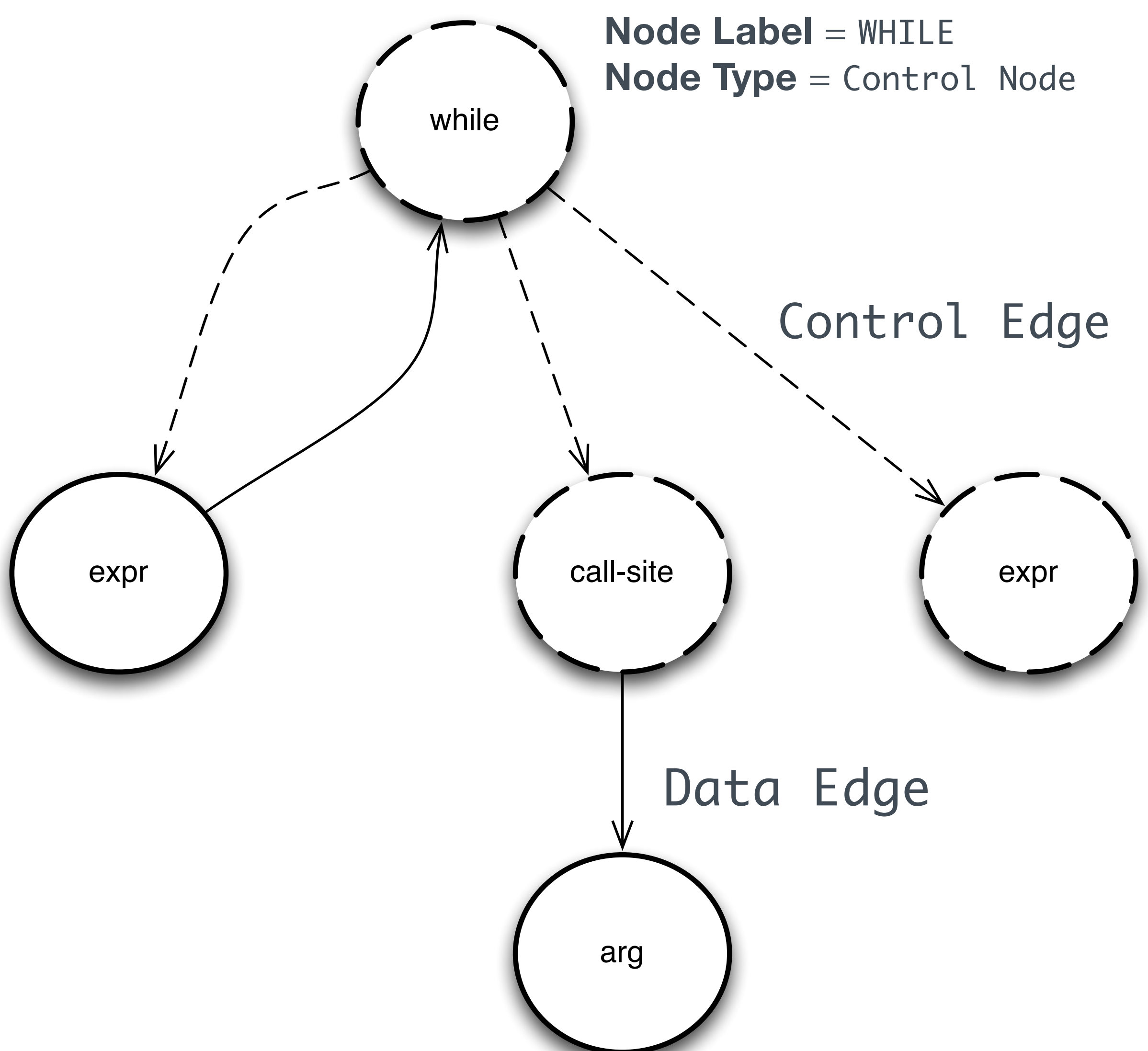
GRAPH KERNELS FOR PDG



- **Features of nodes**:
  - **Node Label**
    - i.e., , `WHILE, CALL-SITE, EXPR, ...`
  - **Node Type**
    - i.e., `Data Node or Control Node`
- **Features of edges:**
  - **Edge Type**
    - i.e., `Data Edge or Control Edge`

KERNELS
FOR CODE STRUCTURES:
PDG

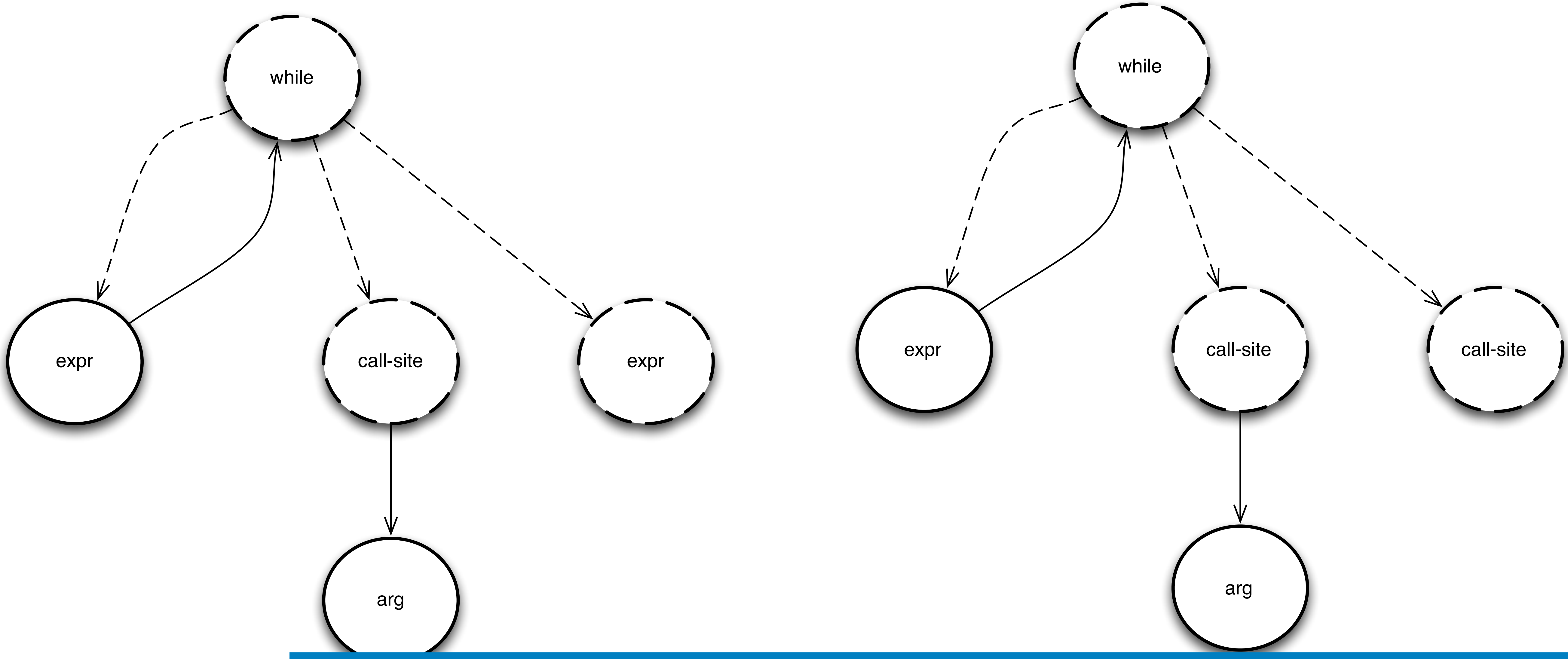
GRAPH KERNELS FOR PDG



- **Features of nodes**:
  - **Node Label**
    - i.e., , WHILE, CALL-SITE, EXPR, ...
  - **Node Type**
    - i.e., Data Node or Control Node
- **Features of edges:**
  - **Edge Type**
    - i.e., Data Edge or Control Edge

GRAPH KERNELS FOR PDG

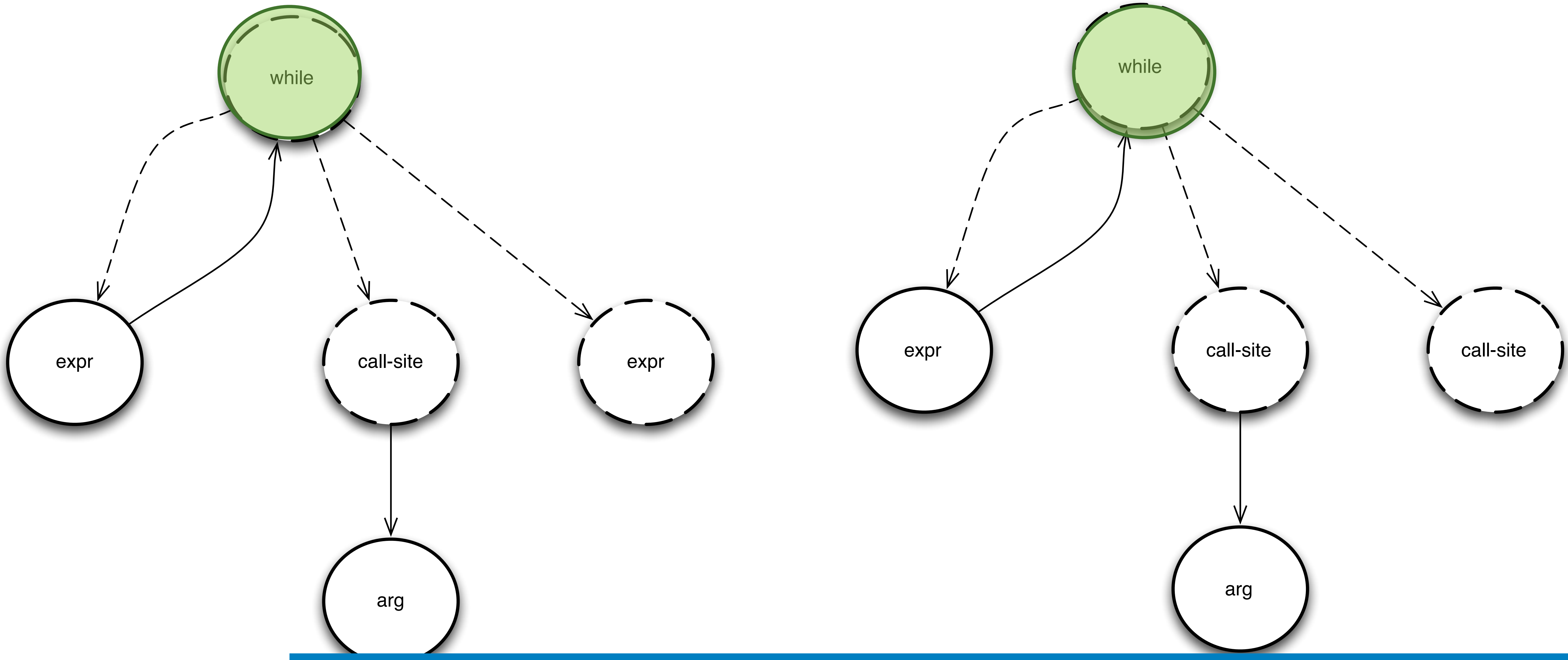
- **Goal:** *Identify common subgraphs*
- ***Selectors:*** *Compare nodes to each others and explore the subgraphs of only "compatible" nodes (i.e., Nodes of the same type)*
- ***Context:*** *The subgraph of a node (with paths whose lengths are at most L to avoid loops)*



KERNELS FOR CODE STRUCTURES: **PDG**

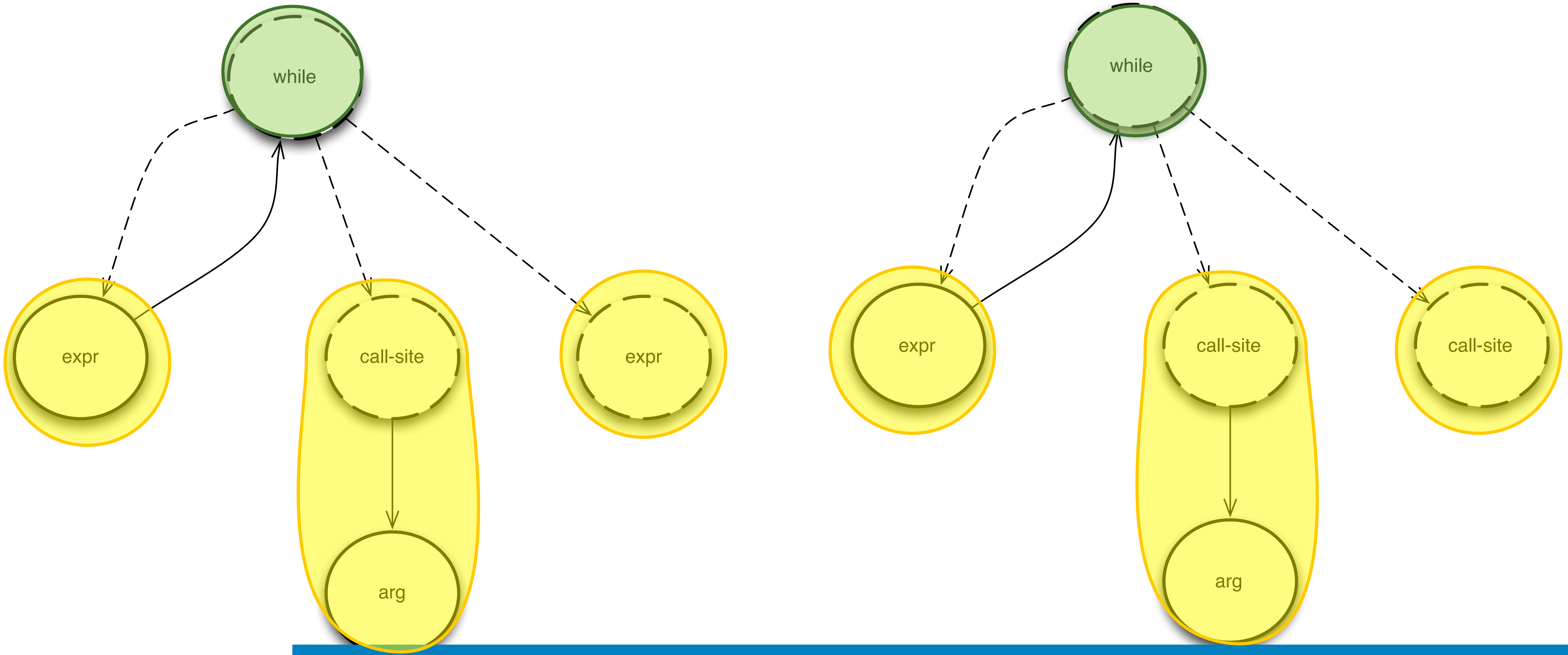
GRAPH KERNELS FOR PDG

- **Goal:** *Identify common subgraphs*
- ***Selectors:*** *Compare nodes to each others and explore the subgraphs of only "compatible" nodes (i.e., Nodes of the same type)*
- ***Context:*** *The subgraph of a node (with paths whose lengths are at most L to avoid loops)*



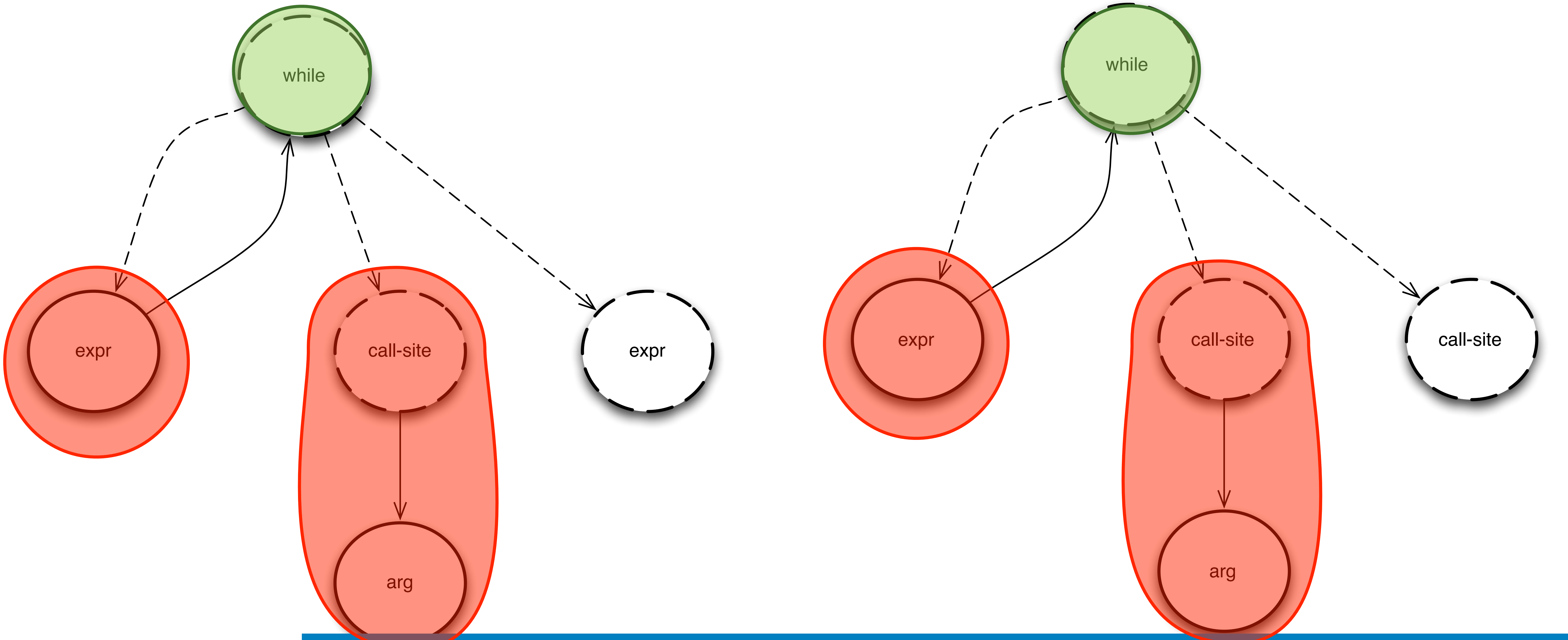
GRAPH KERNELS FOR PDG

- **Goal:** *Identify common subgraphs*
- ***Selectors:*** *Compare nodes to each others and explore the subgraphs of only "compatible" nodes (i.e., Nodes of the same type)*
- ***Context:*** *The subgraph of a node (with paths whose lengths are at most L to avoid loops)*



GRAPH KERNELS FOR PDG

- **Goal:** *Identify common subgraphs*
- ***Selectors:*** *Compare nodes to each others and explore the subgraphs of only "compatible" nodes (i.e., Nodes of the same type)*
- ***Context:*** *The subgraph of a node (with paths whose lengths are at most L to avoid loops)*



EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
  - AST-based Clone detector
  - PDG-based Clone Detector

EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
    - AST-based Clone detector
    - PDG-based Clone Detector
- No publicly available clone detection dataset

EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
  - AST-based Clone detector
  - PDG-based Clone Detector
- No publicly available clone detection dataset
  - No unique set of analyzed open source systems

EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
  - AST-based Clone detector
  - PDG-based Clone Detector
- No publicly available clone detection dataset
  - No unique set of analyzed open source systems
  - Usually clone results are not available

EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
    - AST-based Clone detector
    - PDG-based Clone Detector
- No publicly available clone detection dataset
    - No unique set of analyzed open source systems
    - Usually clone results are not available
- Two possible strategies:

EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
  - AST-based Clone detector
  - PDG-based Clone Detector
- No publicly available clone detection dataset
  - No unique set of analyzed open source systems
  - Usually clone results are not available
- Two possible strategies:
  - To automatically modify an existing system with randomly generated clones

EVALUATION PROTOCOL

- Comparison of results with other two clone detector tools:
  - AST-based Clone detector
  - PDG-based Clone Detector
- No publicly available clone detection dataset
  - No unique set of analyzed open source systems
  - Usually clone results are not available
- Two possible strategies:
  - To automatically modify an existing system with randomly generated clones
  - Manual classification of candidate results

EMPIRICAL
EVALUATION

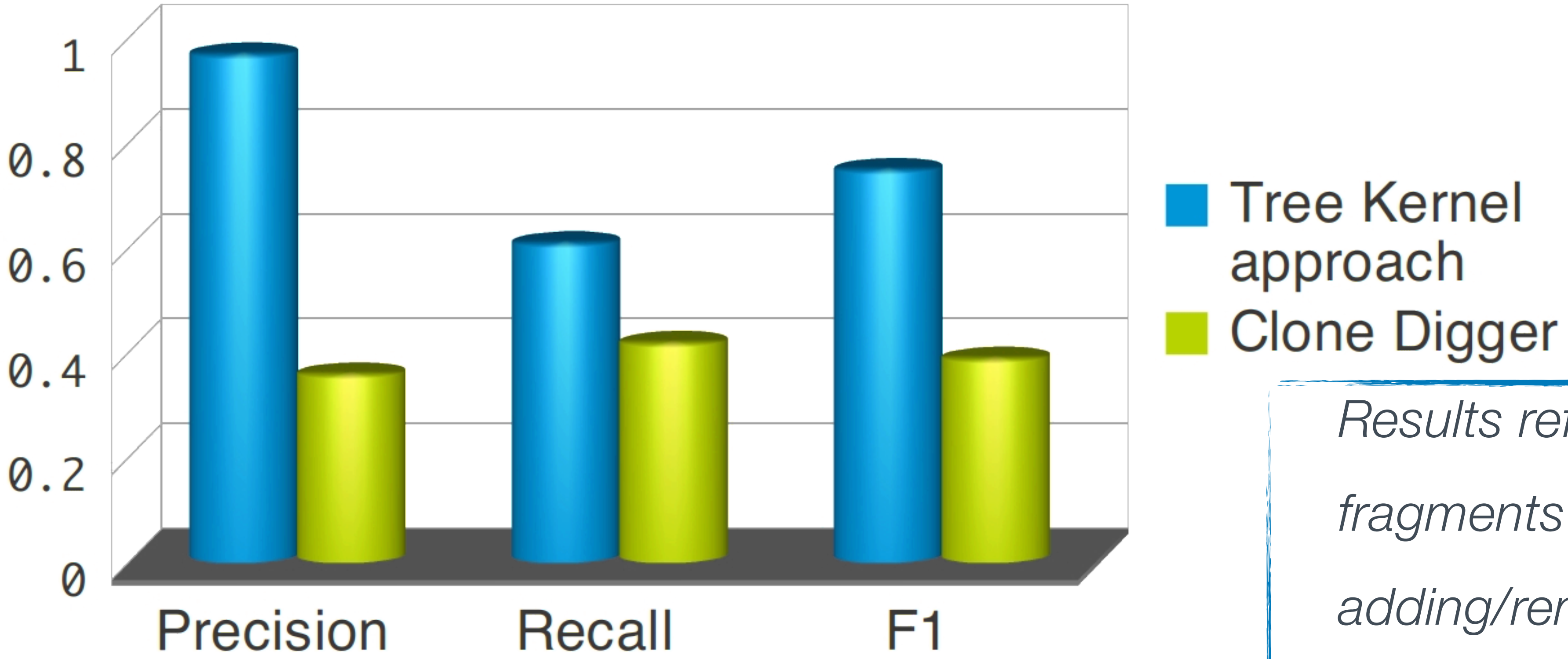
BENCHMARKS AND DATASET

Project	Size (KLOC)	# PDGS
Apache-2.2.14	343	3017
Python-2.5.1	435	5091

- Comparison with another Graph-based clone detector
  - MeCC (ICSE2011)
- **Baseline** Dataset
  - Results provided by MeCC
- **Extended** Dataset
  - Extension of Clones results by manual evaluation of candidate clones
    - Agreement rate calculation between the evaluators

EMPIRICAL EVALUATION OF TREE KERNEL FOR AST

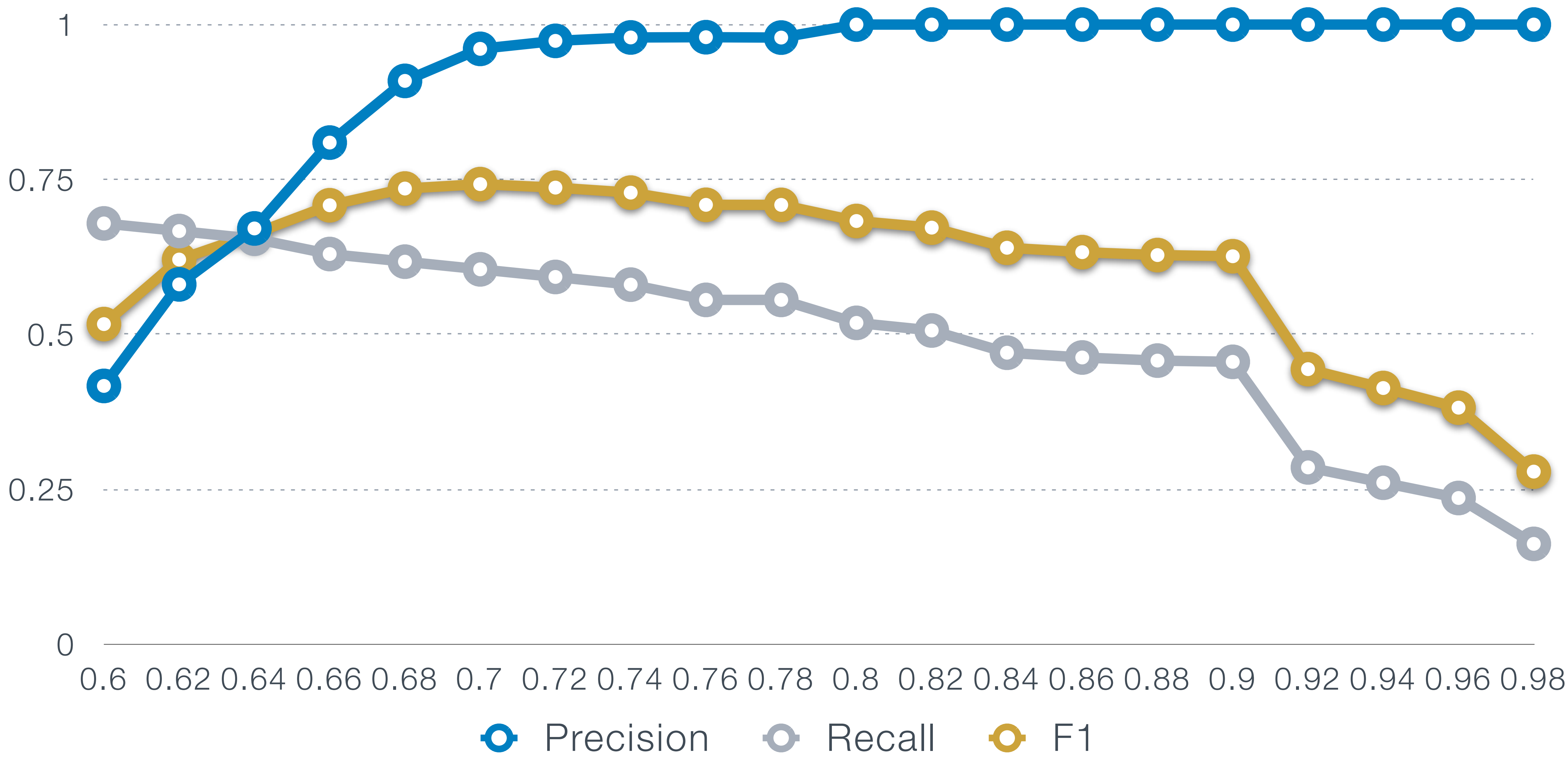
- Comparison with another (pure) AST-based clone detector
  - Clone Digger http://clonedigger.sourceforge.net/
- Comparison on a system with randomly seeded clones



Results refer to clones where code fragments have been modified by adding/removing or changing code statements

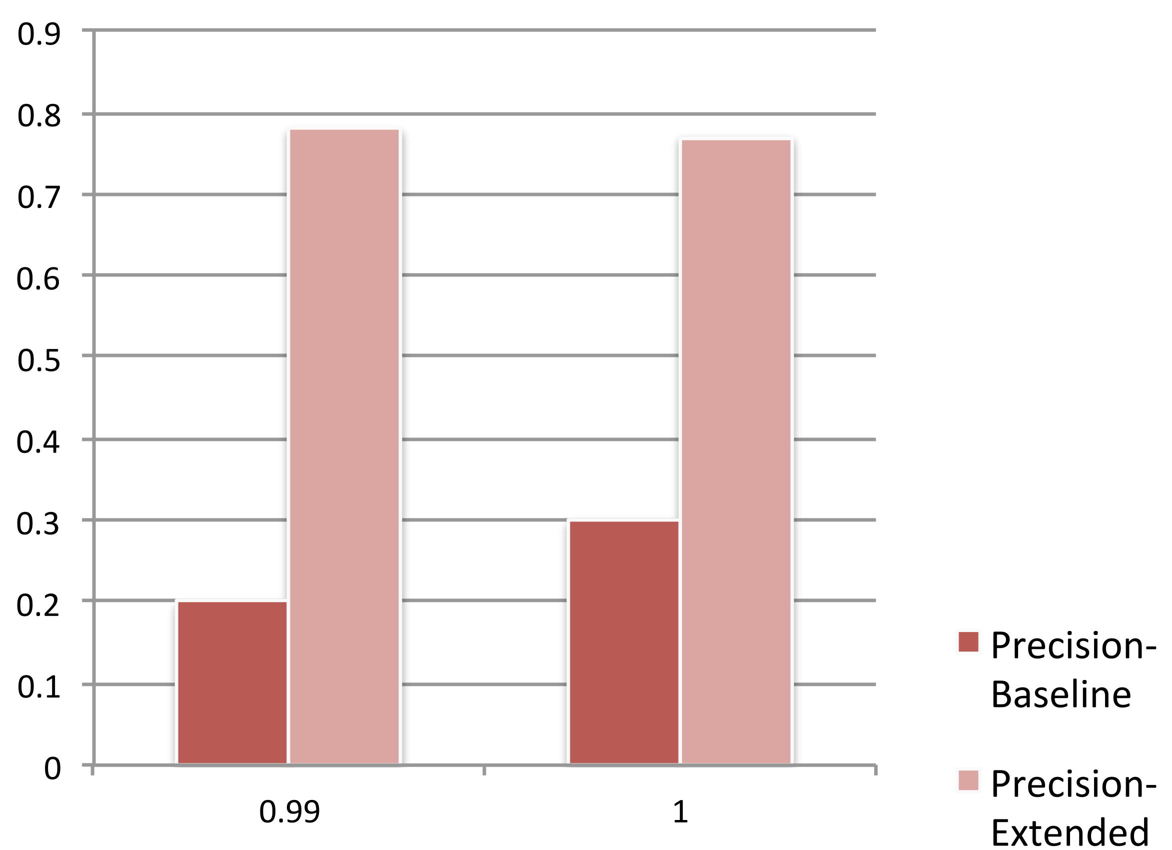
PRECISION, RECALL AND F1 PLOT

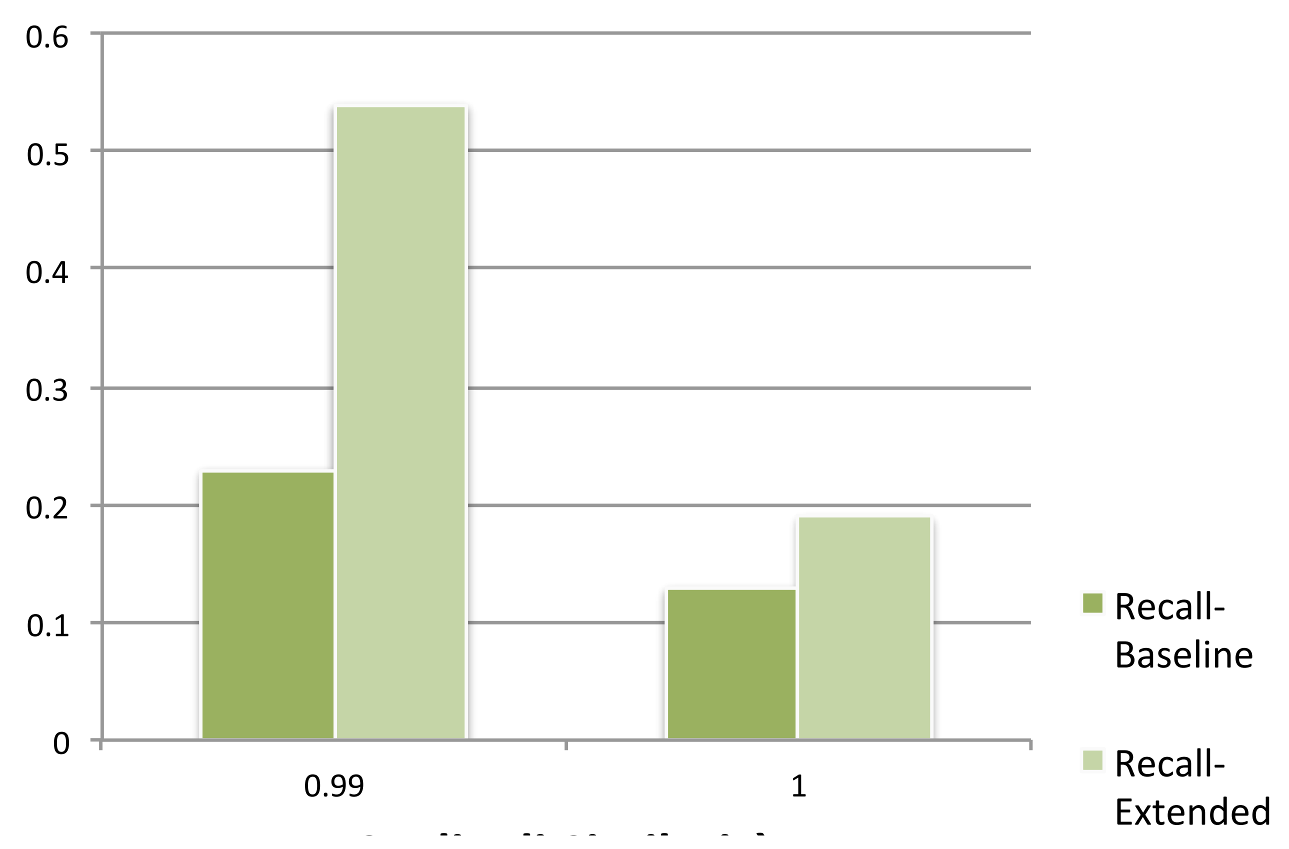
Clone results with different similarity thresholds



RESULTS WITH APACHE 2.2.14

Threshold	#Clones in the Baseline	#Clones in the Extended Dataset
1.00	874	1089
0.99	874	1514

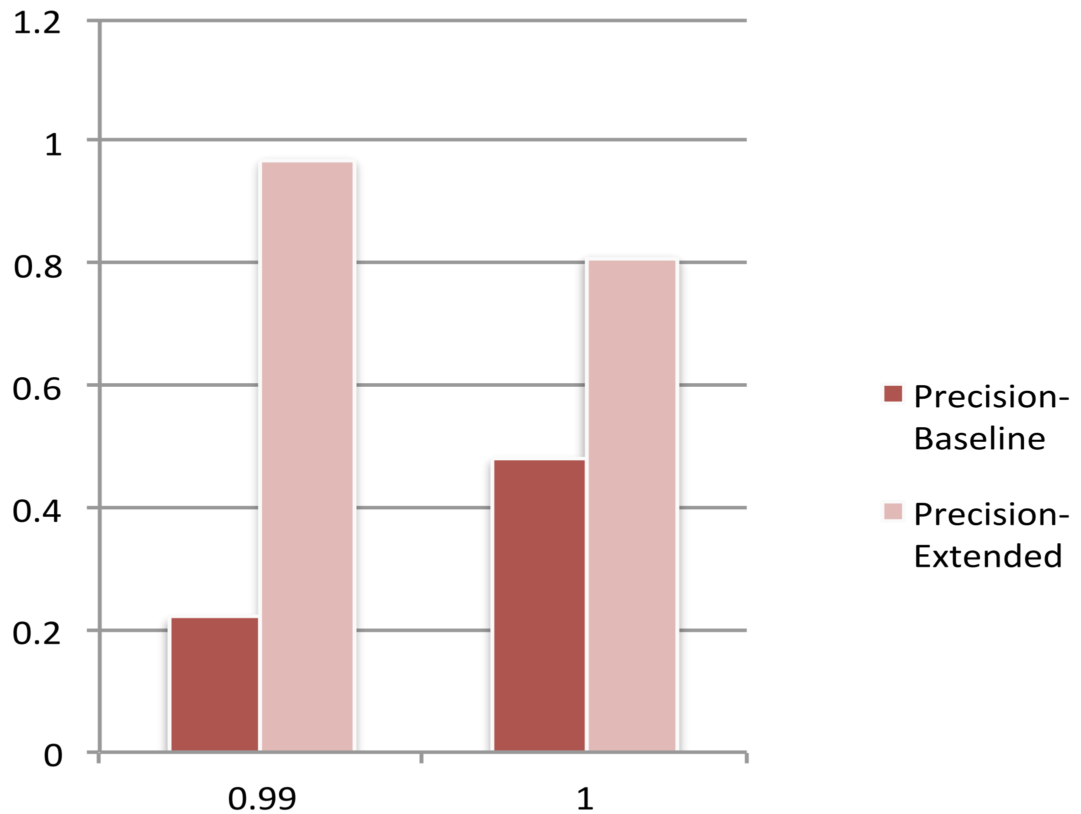


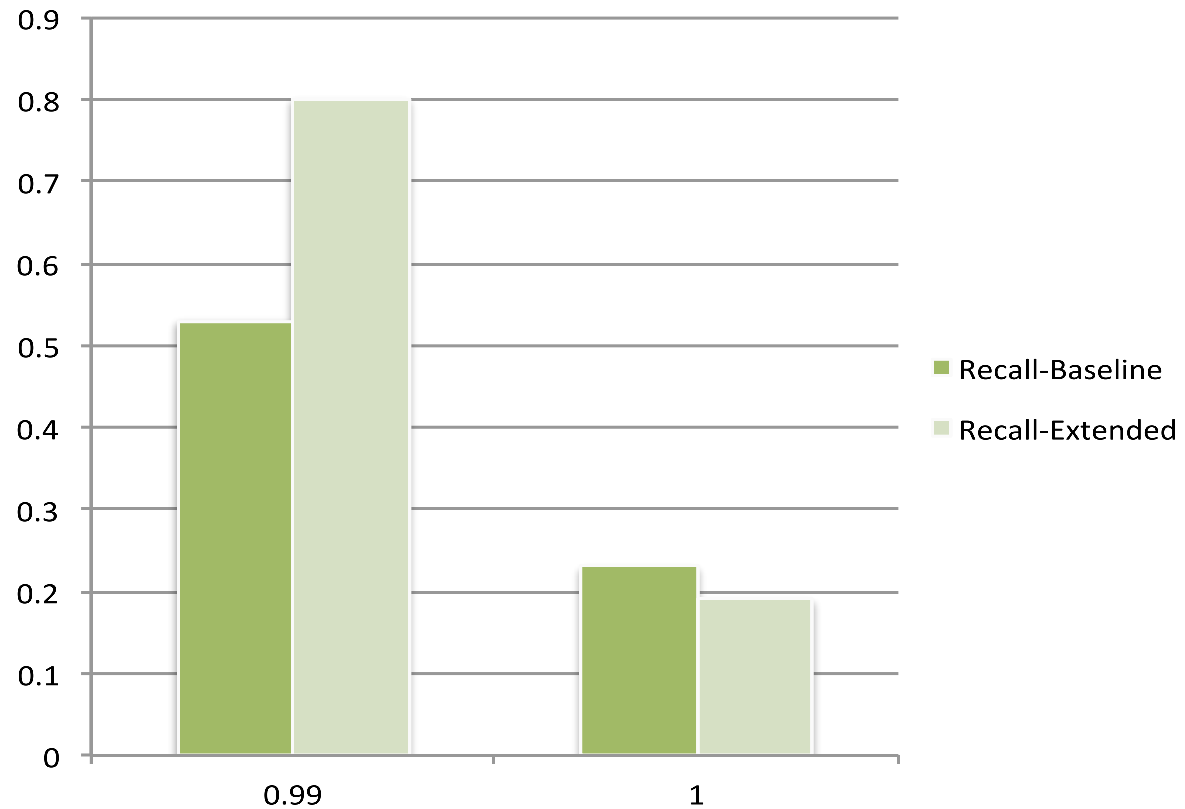


EVALUATION
GRAPH KERNELS FOR PDG

RESULTS WITH PYTHON 2.5.2

Threshold	#Clones in the Baseline	#Clones in the Extended Dataset
1.00	858	1066
0.99	858	2119





CHALLENGES AND OPPORTUNITIES

- Learning Kernel Functions from Data Set
  - Kernel Methods **advantages**:
    - flexible solution to be tailored to specific domain
    - efficient solution easy to parallelize
    - combinations of multiple kernels
  - Provide a publicly available data set

THANK YOU

FOR YOUR KIND ATTENTION