Research on NLP for RE at Università della Svizzera italiana: a Report

Arianna Blasi♦ · Mauro Pezzè♦ · Alessandra Gorla♥ · Michael D. Ernst♠

♦USI Università della Svizzera italiana, Switzerland

♥IMDEA Software Institute, Spain

♠University of Washington Seattle, WA, USA

Automatic generation of test cases

Test case =

Test case = input

Test case = input +

Test case = input + *expected* output

Test case = input + *expected* output

- How to know inputs - valid, invalid, etc. ?

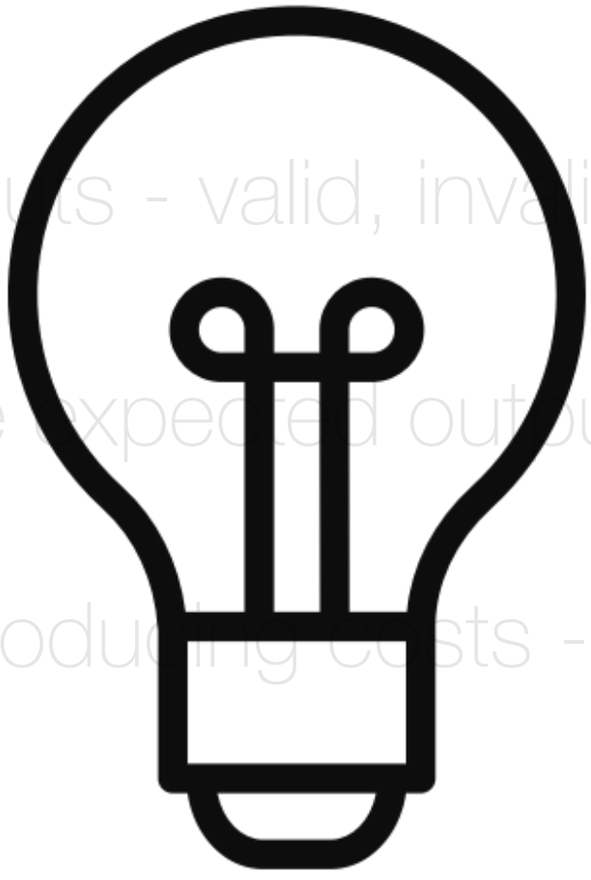
Test case = input + *expected* output

- How to know inputs - valid, invalid, etc. ?
- How to know the expected output given an input?

Test case = input + *expected* output

- How to know inputs - valid, invalid, etc. ?
- How to know the expected output given an input?
- How to avoid introducing costs - e.g., human aid?

Test case = input + *expected* output



- How to know inputs - valid, invalid, etc. ?
- How to know the expected output given an input?
- How to avoid introducing costs - e.g., human aid?

Exploiting available information

- How to know inputs - valid, invalid, etc. ?
- How to know the expected output given an input?
- How to avoid introducing costs - e.g., human aid?

Exploiting available information

Exploiting available information

- *Semantically relevant* information

Exploiting available information

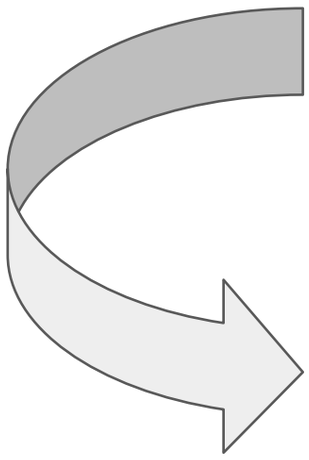
- *Semantically relevant* information
- Often expressed in Natural Language

Exploiting available information

- *Semantically relevant* information
- Often expressed in Natural Language
- ...code comments, documentation, requirements...

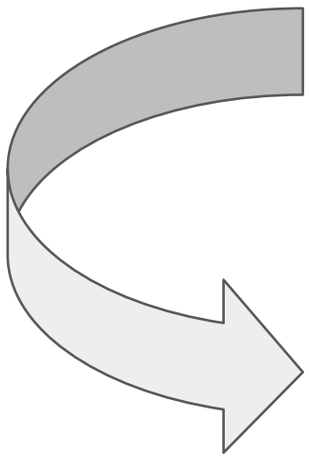
```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```



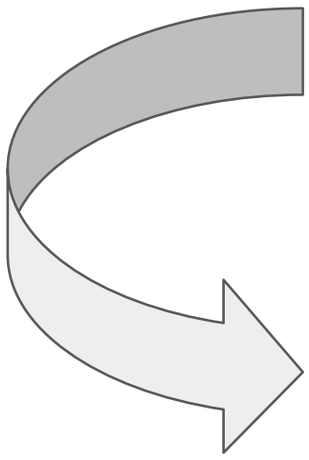
```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```



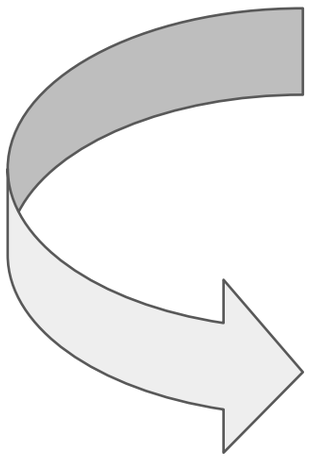
```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```



```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```

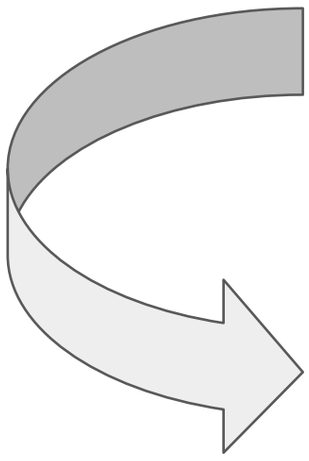


```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

Test fails

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```

BUG!



```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

Test fails

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x < 0;
    }
}
```

FIXED!

```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

```java
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x < 0;
    }
}
```

FIXED!

```java
public class SimpleMathTest {
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

Test passes

```
// returns true if x is less than 0

public class SimpleMath {
    public boolean isNegative(int x){
        return x > 0;
    }
}
```

FIXED!

Test cases help you detecting bugs!

```
    @Test
    public void testIsNegative(){
        SimpleMath simpleMathInstance = new SimpleMath();
        assertTrue(simpleMathInstance.isNegative(-1));
    }
}
```

Test passes

Can we automatically translate natural language information into test cases?

Can we automatically translate natural language information into test cases?

Let’s try with Javadoc comments!

```
/**
 * @param  v, the vertex, must not be null
 *
 * @param  nList, list of possible neighbors
 *
 * @return true if vertexes in nList are neighbors of
 *         vertex. Always false if the list is empty
 *
 * @throws IllegalArgumentException if vertex is not
 *         found in the graph
 */

public boolean neighborsOf(Vertex v, List nList){…}
```

```
/**
 * @param  v, the vertex, must not be null
 *
 * @param  nList, list of possible neighbors
 *
 * @return true if vertexes in nList are neighbors of
 *         vertex. Always false if the list is empty
 *
 * @throws IllegalArgumentException if vertex is not
 *         found in the graph
 */

public boolean neighborsOf(Vertex v, List nList){…}
```

v != null

```
/**
 * @param  v, the vertex, must not be null
 *
 * @param  nList, list of possible neighbors
 *
 * @return true if vertexes in nList are neighbors of
 *         vertex. Always false if the list is empty
 *
 * @throws IllegalArgumentException if vertex is not
 *         found in the graph
 */

public boolean neighborsOf(Vertex v, List nList){…}
```

v != null

nList.isEmpty() ? returnValue==false

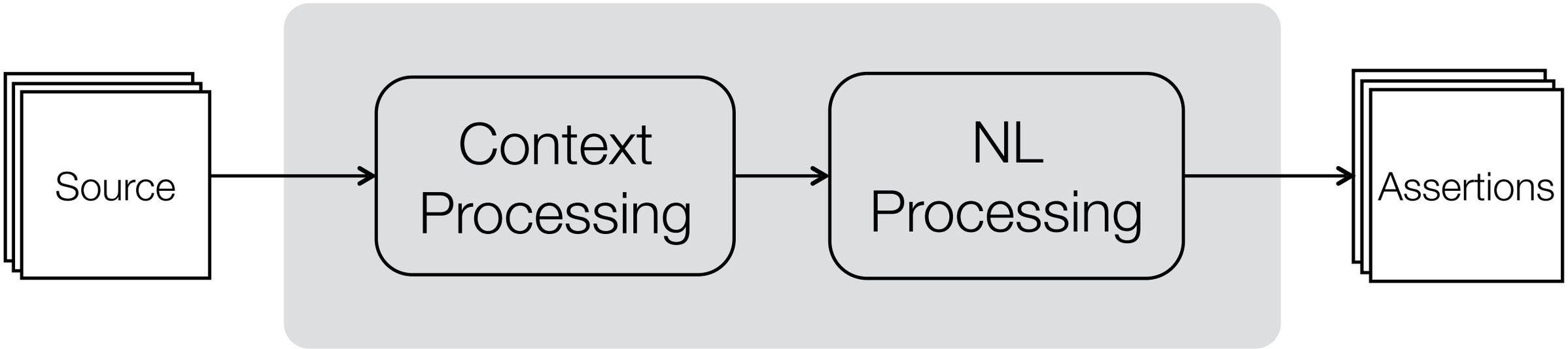
```
/**
 * @param  v, the vertex, must not be null
 *
 * @param  nList, list of possible neighbors
 *
 * @return true if vertexes in nList are neighbors of
 *         vertex. Always false if the list is empty
 *
 * @throws IllegalArgumentException if vertex is not
 *         found in the graph
 */

public boolean neighborsOf(Vertex v, List nList){…}
```

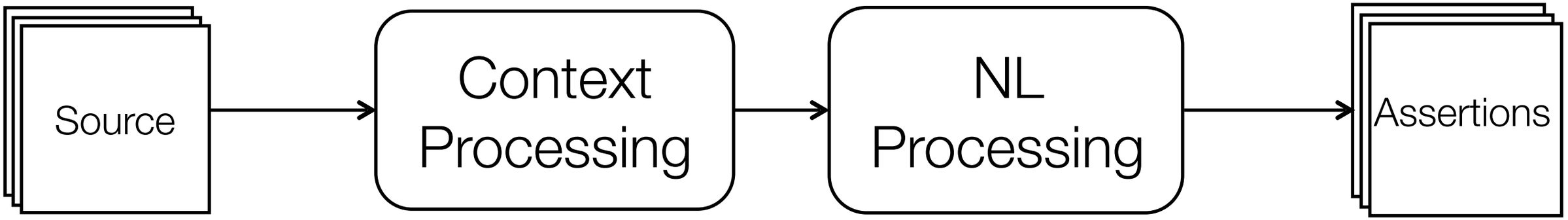
v != null

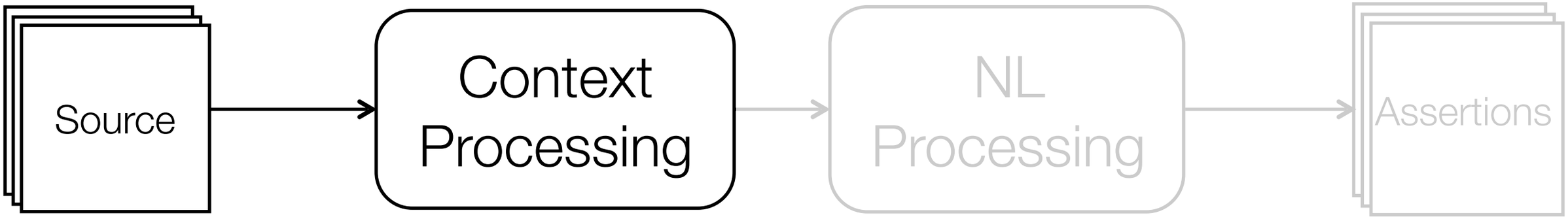
nList.isEmpty() ? returnValue==false

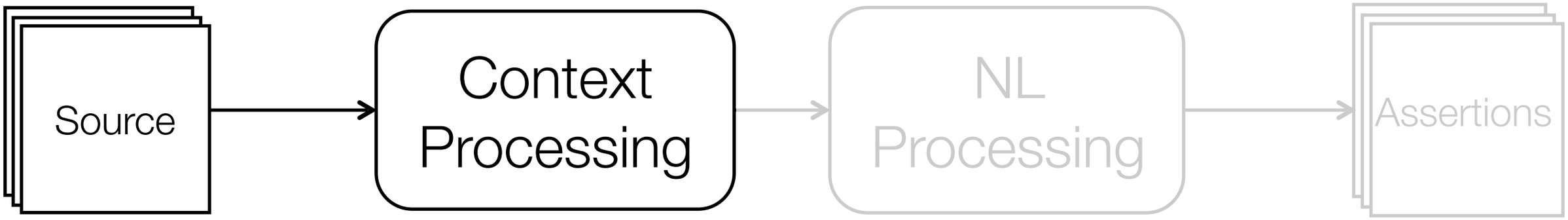
!graph.contains(v) -> IllegalArgumentException

Jdoctor



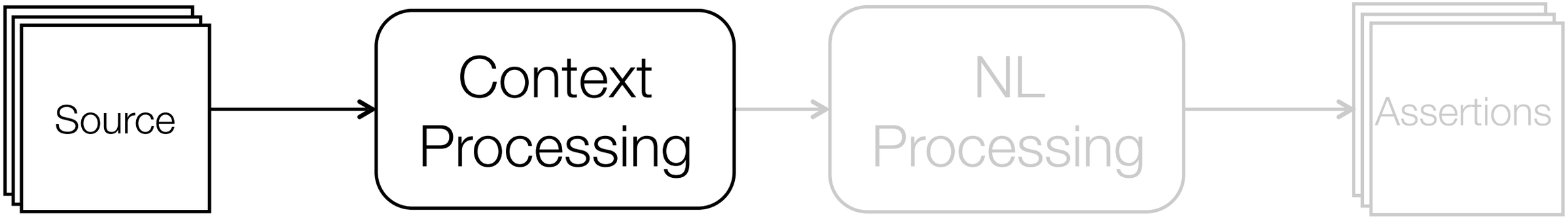

Source
Context Processing
NL Processing
Assertions


Source
Context Processing
NL Processing
Assertions



@param

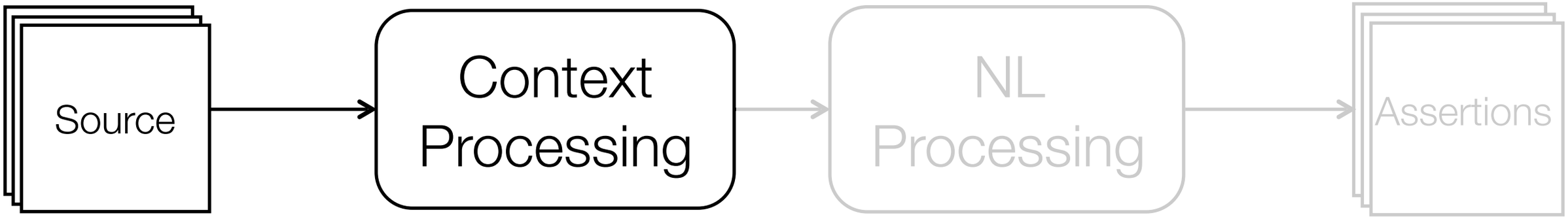
1. Identifiers
2. Condition



@param

1. Identifiers
2. Condition

```
@param v, the vertex, must not
       be null
```



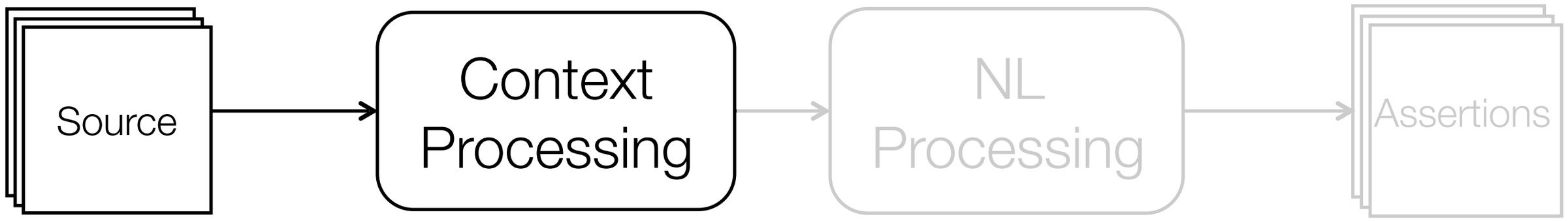
@param

1. Identifiers
2. Condition

```
@param v, the vertex, must not
       be null
```

@return

1. Expected result
2. Condition



@param

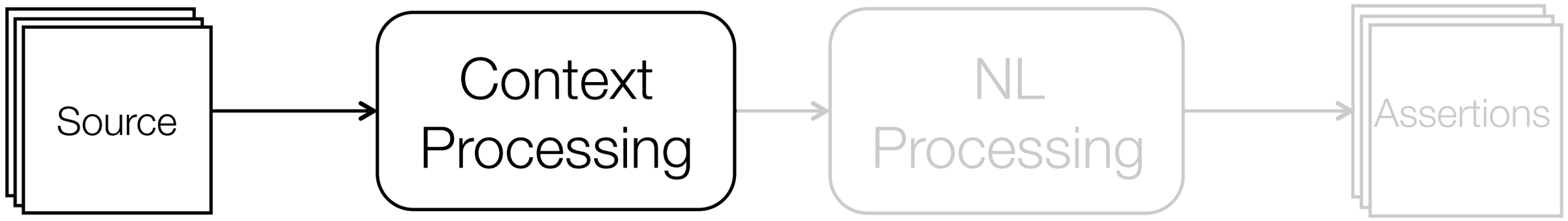
1. Identifiers
2. Condition

```
@param v, the vertex, must not
       be null
```

@return

1. Expected result
2. Condition

```
@return always false if the
        list is empty
```



@param

1. Identifiers
2. Condition

```
@param v, the vertex, must not
       be null
```

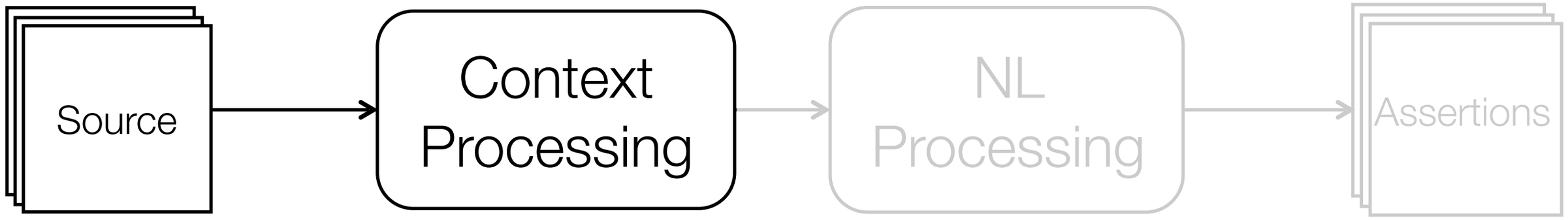
@return

1. Expected result
2. Condition

```
@return always false if the
        list is empty
```

@throws

1. Expected exception
2. Condition



@param

1. Identifiers
2. Condition

```
@param v, the vertex, must not
       be null
```

@return

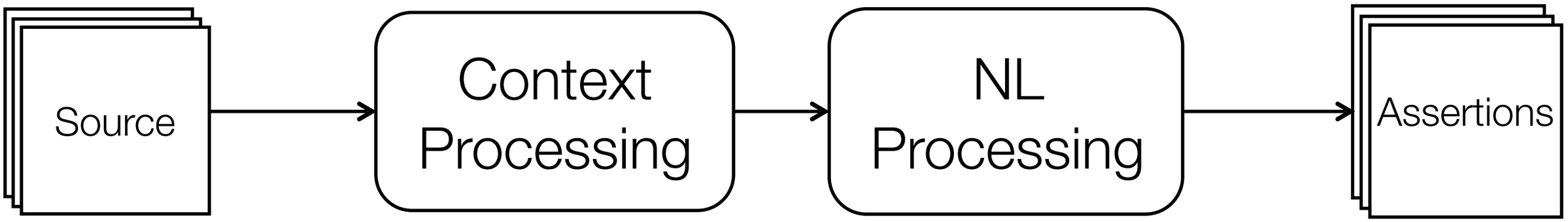
1. Expected result
2. Condition

```
@return always false if the
        list is empty
```

@throws

1. Expected exception
2. Condition

```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```



@param

1. Identifiers
2. Text

```
@param v, the vertex, must not
       be null
```

@return

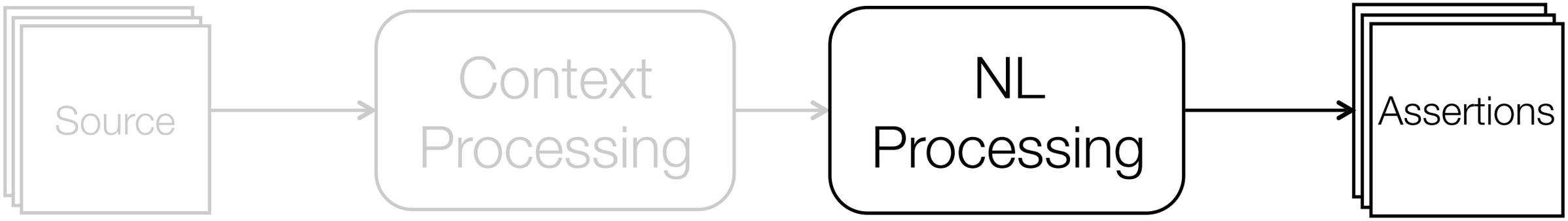
1. Expected result
2. Text

```
@return always false if the
        list is empty
```

@throws

1. Expected exception
2. Text

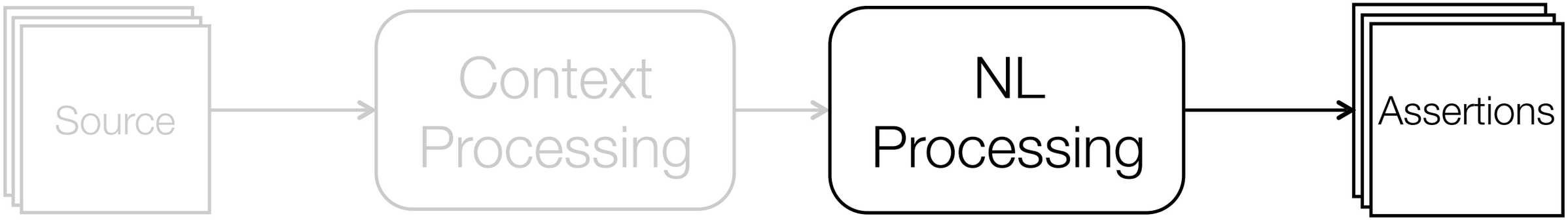
```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```




```
@param v, the vertex, must not
       be null
```

```
@return always false if the
        list is empty
```

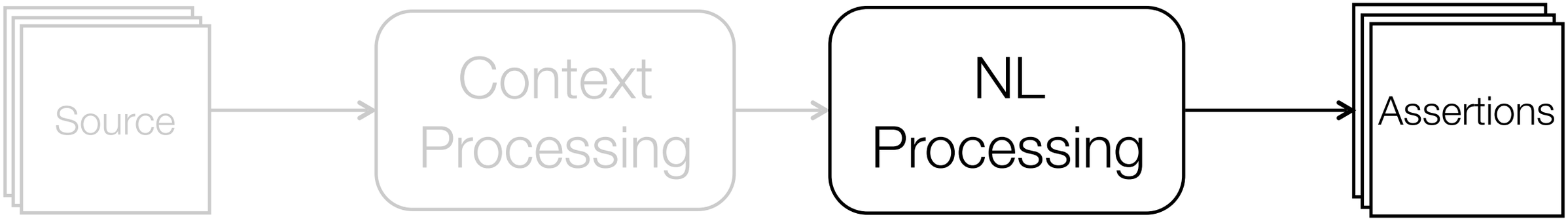
```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```




```
@param v, the vertex, must not
       be null
```

```
@return always false if the
        list is empty
```

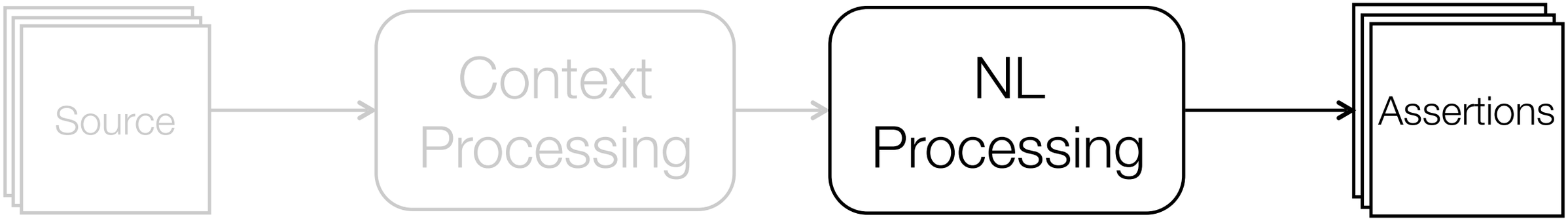
```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```




```
@param v, the vertex, must not
       be null
```

```
@return always false if the
        list is empty
```

```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```




```
@param v, the vertex, must not
       be null
```

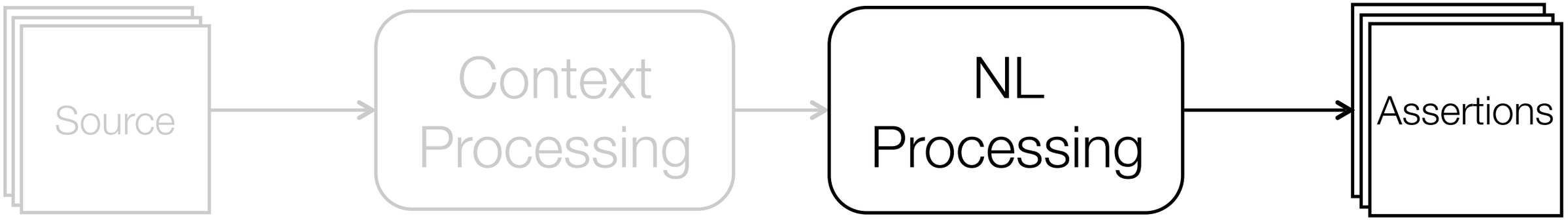
PATTERN MATCH

"vertex not null"

`v != null`

```
@return always false if the
        list is empty
```

```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```




```
@param v, the vertex, must not
       be null
```

PATTERN MATCH

"vertex not null"

`v != null`

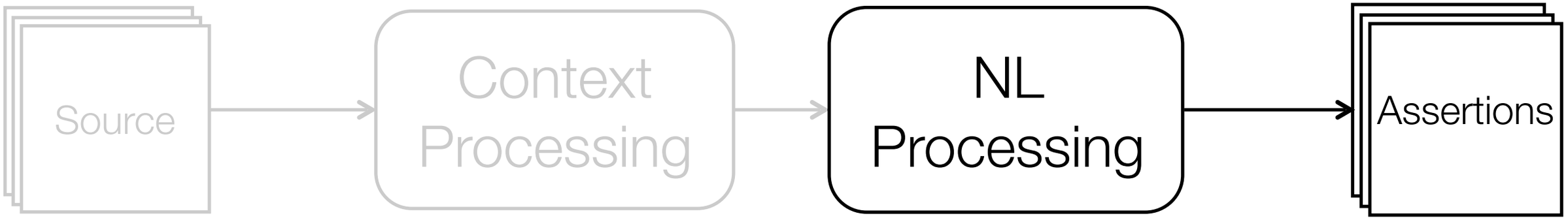
```
@return always false if the
        list is empty
```

SYNTAX MATCH

"list is empty"

`nList.isEmpty()`

```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```




```
@param v, the vertex, must not
       be null
```

PATTERN MATCH

"vertex not null"

`v != null`

```
@return always false if the
        list is empty
```

SYNTAX MATCH

"list is empty"

`nList.isEmpty()`

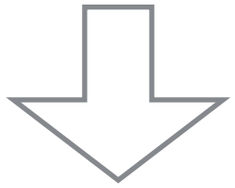
```
@throws IllegalArgumentException
        if vertex is not found
        in the graph
```

SEMANTIC MATCH

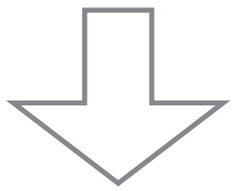
"vertex not found in graph"

`!graph.contains(v)`

Natural Language Specification

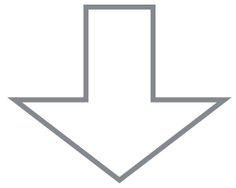


NL
Processing

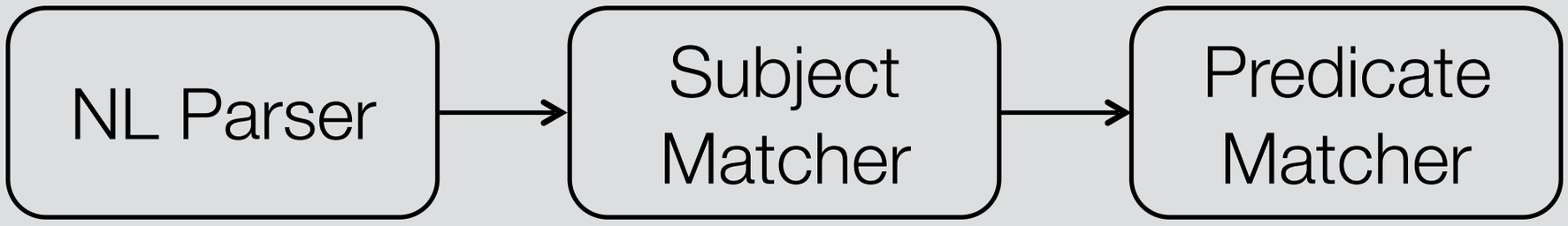


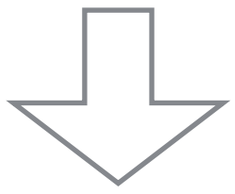
Code Translation

Natural Language Specification



Natural Language Processing





Code Translation

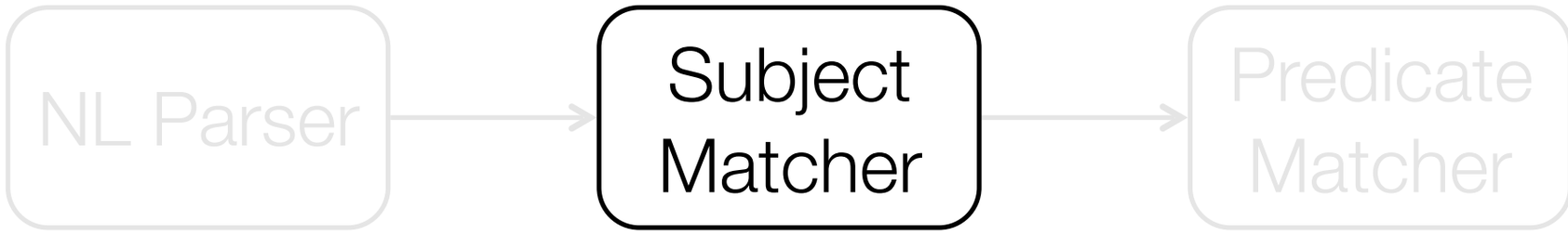
@return

“ *(false)* if the list is empty ”

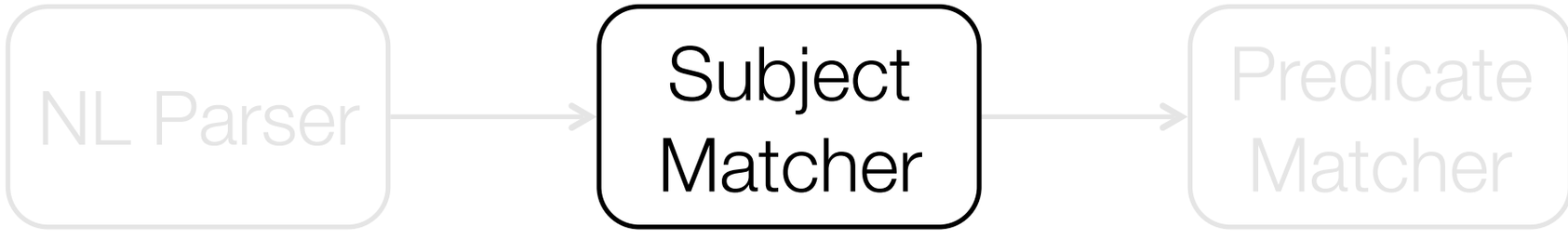
@return

`" (false) if the list is empty "`

Subject	Predicate
`"list"`	`"is empty"`



Subject: `“list”`



Subject: `“list”`

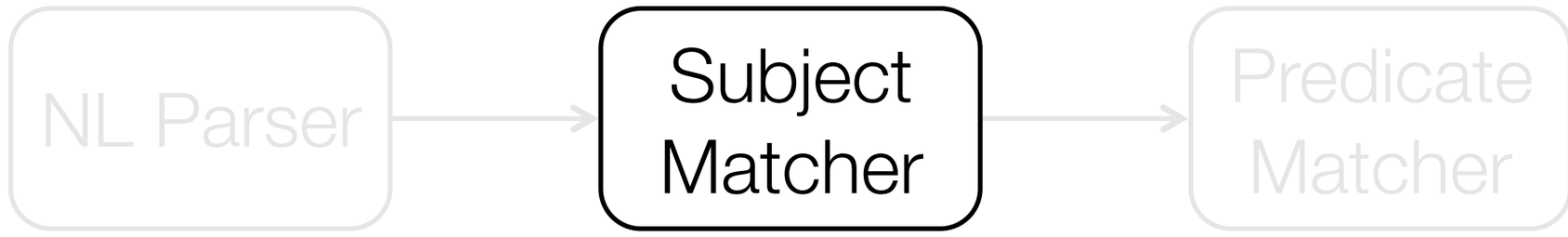
Candidates

Formal Parameters

Class Name

Methods

Fields

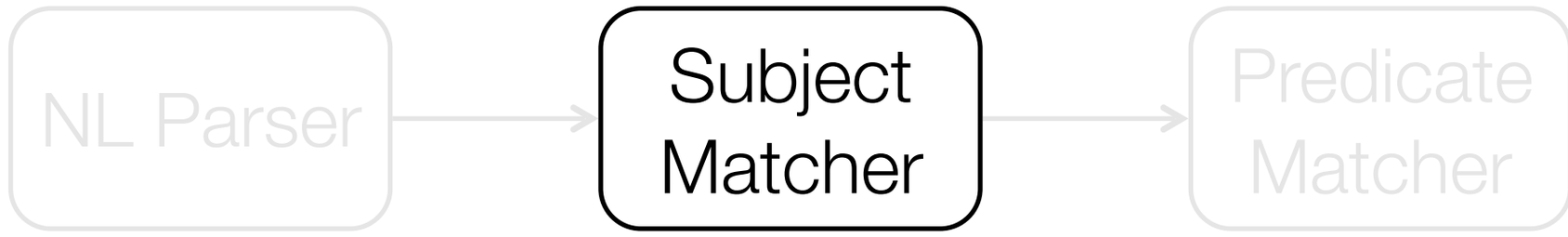


Subject: `"list"`

Candidates

Formal Parameters
Class Name
Methods
Fields

Candidate	Edit Distance
`nList`	1
`isEmpty`	7
`iterator`	8
`size`	4
...	

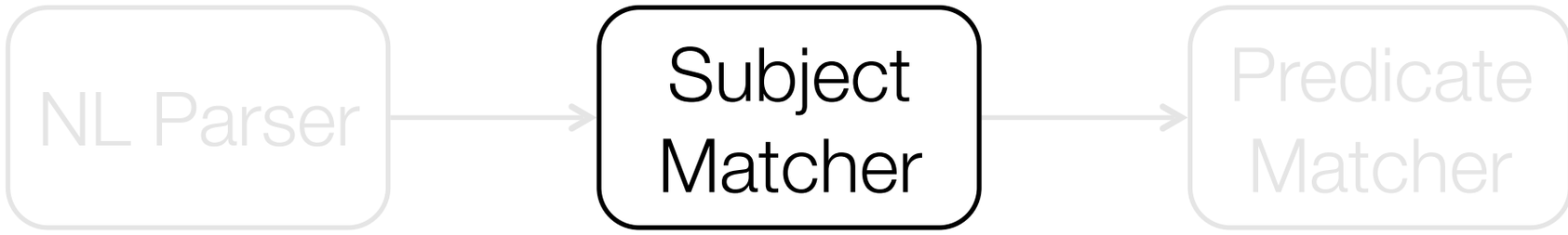


Subject: `"list"`

Candidates

Formal Parameters
Class Name
Methods
Fields

Candidate	Edit Distance
`nList`	**1**
`isEmpty`	7
`iterator`	8
`size`	4
...	



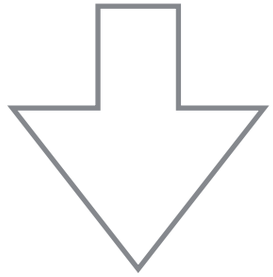
Subject: `"list"` => **`nList`**

Candidates

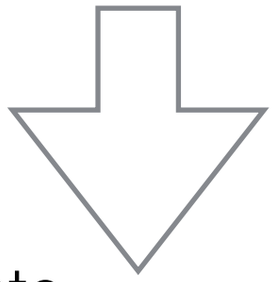
Formal Parameters
Class Name
Methods
Fields

Candidate	Edit Distance
`nList`	**1**
`isEmpty`	7
`iterator`	8
`size`	4
...	

Subject	Predicate
`"list"`	`"is empty"`

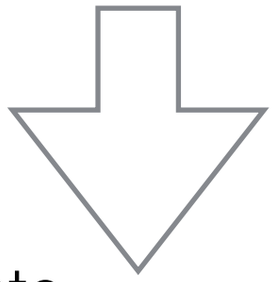


Subject	Predicate
`"list"`	`"is empty"`



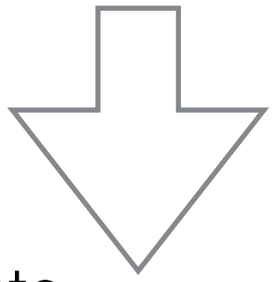
Candidate	Edit Distance
`nList`	8
`isEmpty`	1
`iterator`	6
`size`	7
...	

Subject	Predicate
`"list"`	`"is empty"`



Candidate	Edit Distance
`nList`	8
`isEmpty`	1
`iterator`	6
`size`	7
...	

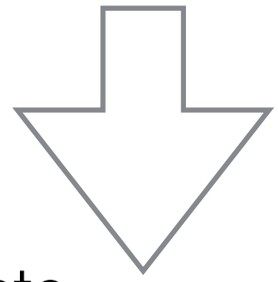
Subject	Predicate
`"list"`	`"is empty"`



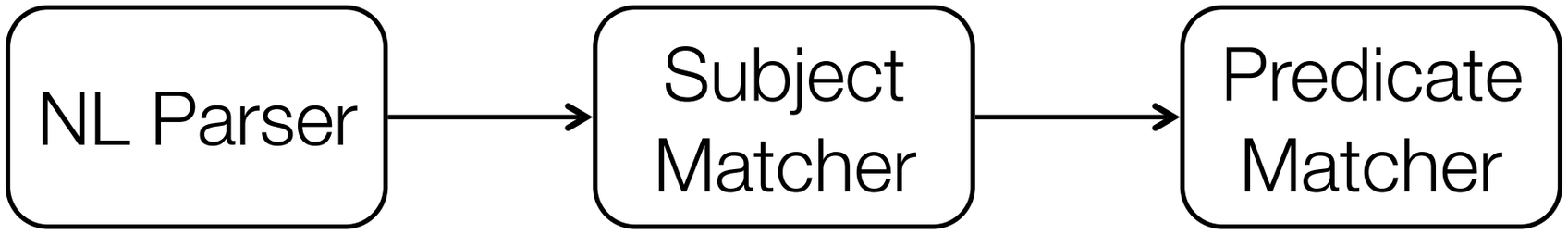
Candidate	Edit Distance
`nList`	8
`isEmpty`	**1**
`iterator`	6
`size`	7
...	

Subject	Predicate
`"list"`	`"is empty"`

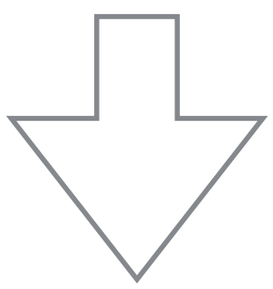
Syntax match



Candidate	Edit Distance
`nList`	8
`isEmpty`	**1**
`iterator`	6
`size`	7
...	



Subject	Predicate
`"list"`	`"is empty"`



```
nList.isEmpty()
```

Experimental setup

6 Popular open-source Java systems

Experimental setup

829 Manually-written Java conditions

92

%

Precision

92%

Precision

83%

Recall

92%

Precision

83%

Recall

Translating Code Comments to Procedure Specifications, Blasi et al., ISSTA'18

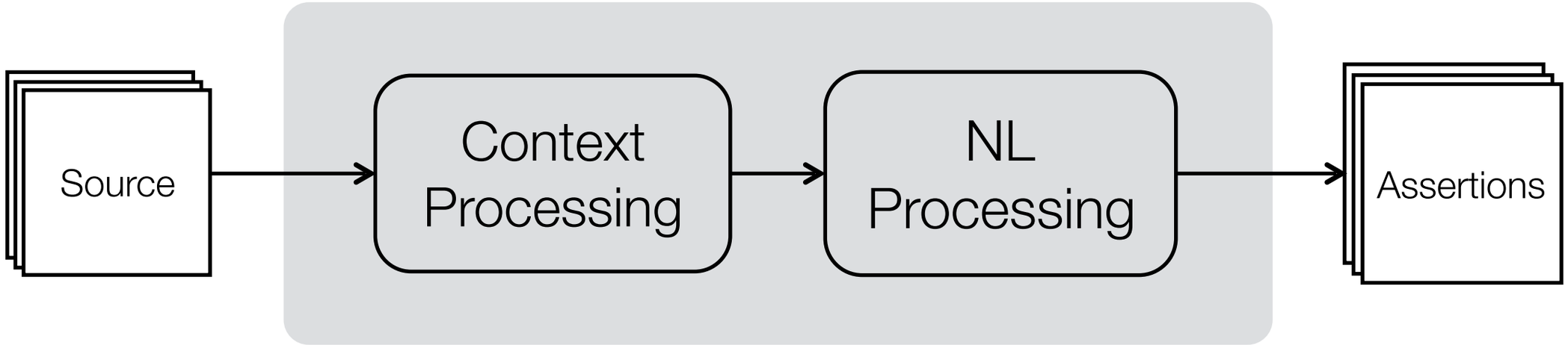
Past

Past

```
/**
 * @return always false if the list is empty
 * @param  v, the vertex, must not be null
 * @throws IllegalArgumentException if vertex is not found in the graph
 */
```

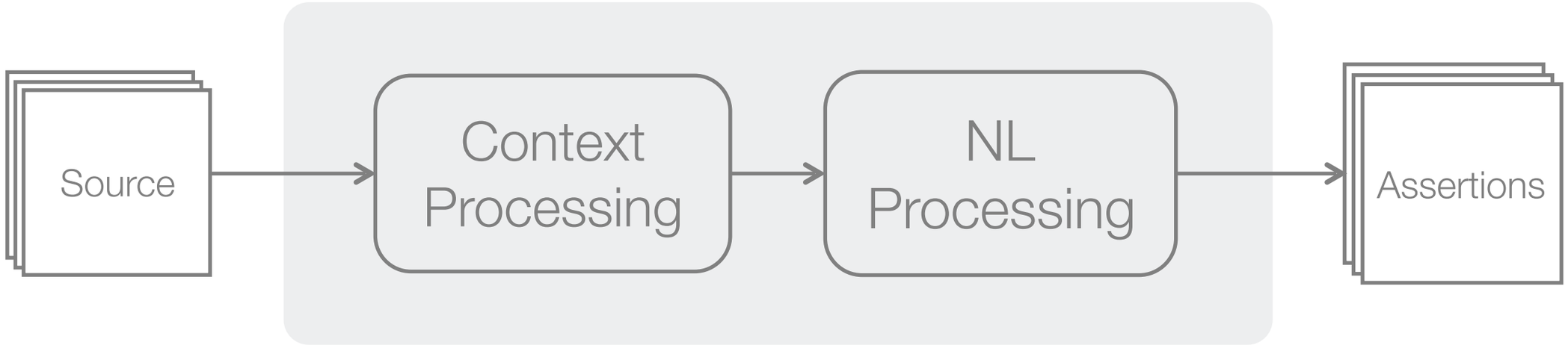
Past

```
/**
 * @return always false if the list is empty
 * @param  v, the vertex, must not be null
 * @throws IllegalArgumentException if vertex is not found in the graph
 */
```



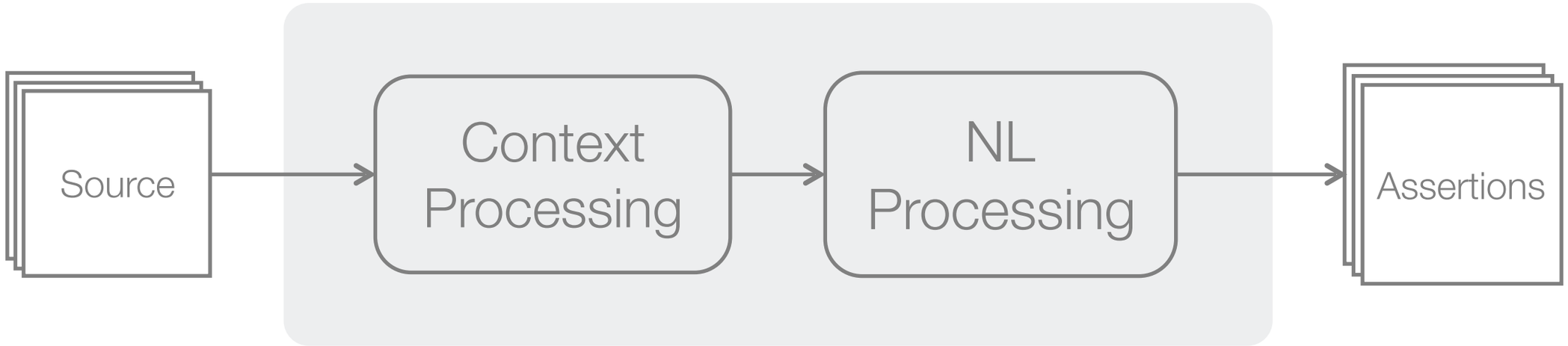
Past

```
/**
 * @return always false if the list is empty
 * @param  v, the vertex, must not be null
 * @throws IllegalArgumentException if vertex is not found in the graph
 */
```



Past

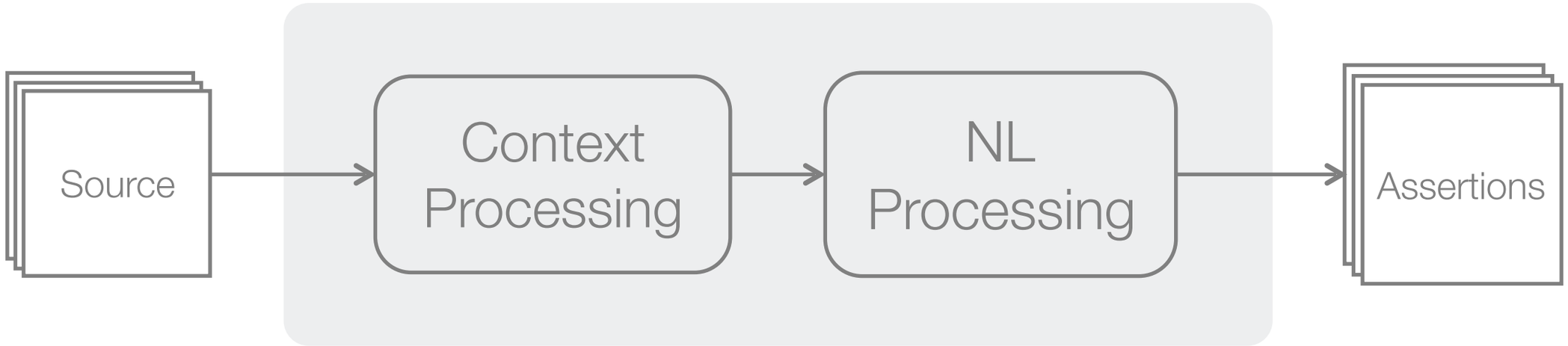
```
/**
 * @return always false if the list is empty
 * @param  v, the vertex, must not be null
 * @throws IllegalArgumentException if vertex is not found in the graph
 */
```



- Exploiting **semi-structured** natural language information

Past

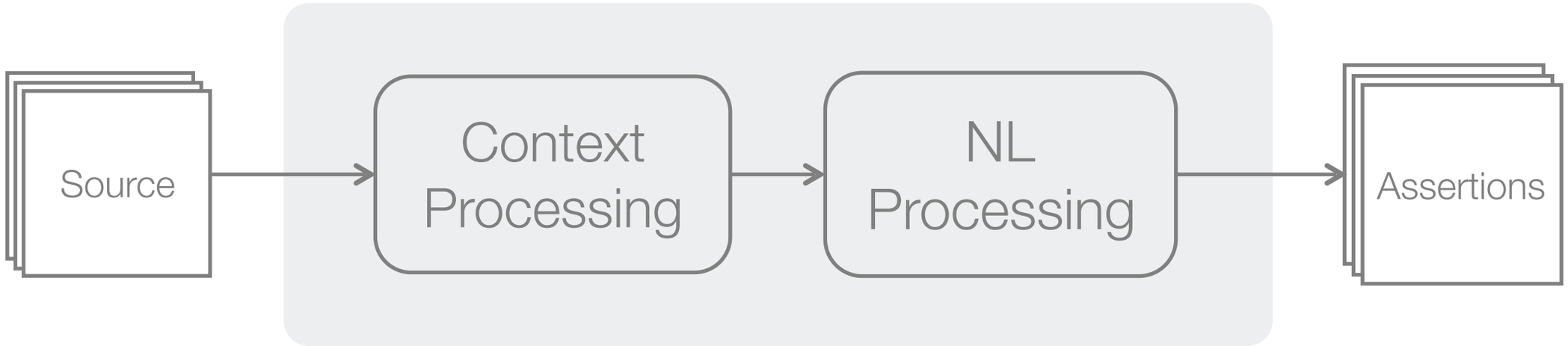
```
/**
 * @return always false if the list is empty
 * @param  v, the vertex, must not be null
 * @throws IllegalArgumentException if vertex is not found in the graph
 */
```



- Exploiting **semi-structured** natural language information
- For generating test cases at the **unit** level

Past

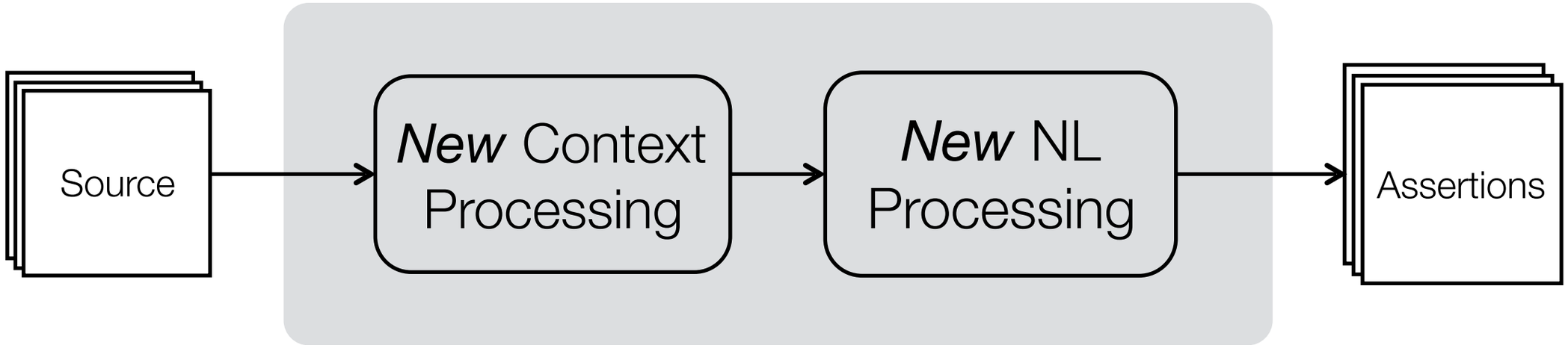
```
/**
 * @return always false if the list is empty
 * @param  v, the vertex, must not be null
 * @throws IllegalArgumentException if vertex is not found in the graph
 */
```



- Exploiting **semi-structured** natural language information
- For generating test cases at the **unit** level
- Testing **functional** properties only

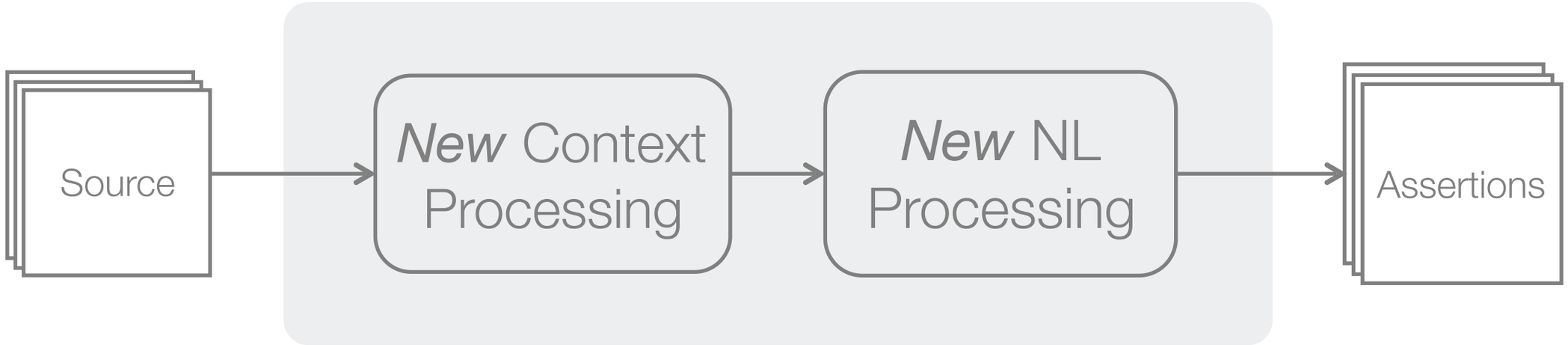
Present

```
/**
 * Note that it is not possible to delete the last element by
 * immediately calling remove() on the iterator, as a call to remove()
 * before a call to next() will result in an illegal state.
 */
```



Present

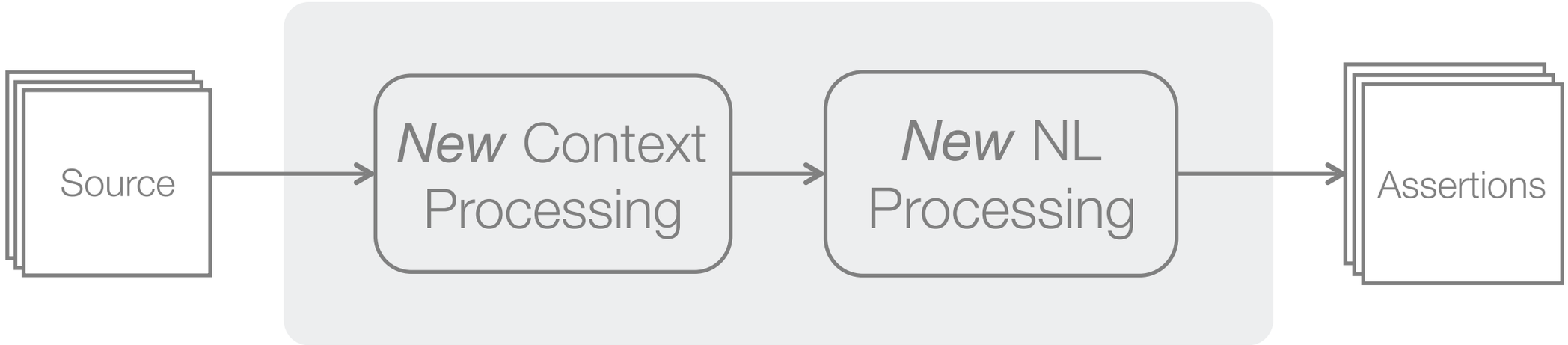
```
/**
 * Note that it is not possible to delete the last element by
 * immediately calling remove() on the iterator, as a call to remove()
 * before a call to next() will result in an illegal state.
 */
```



- Exploiting **unstructured** natural language information

Present

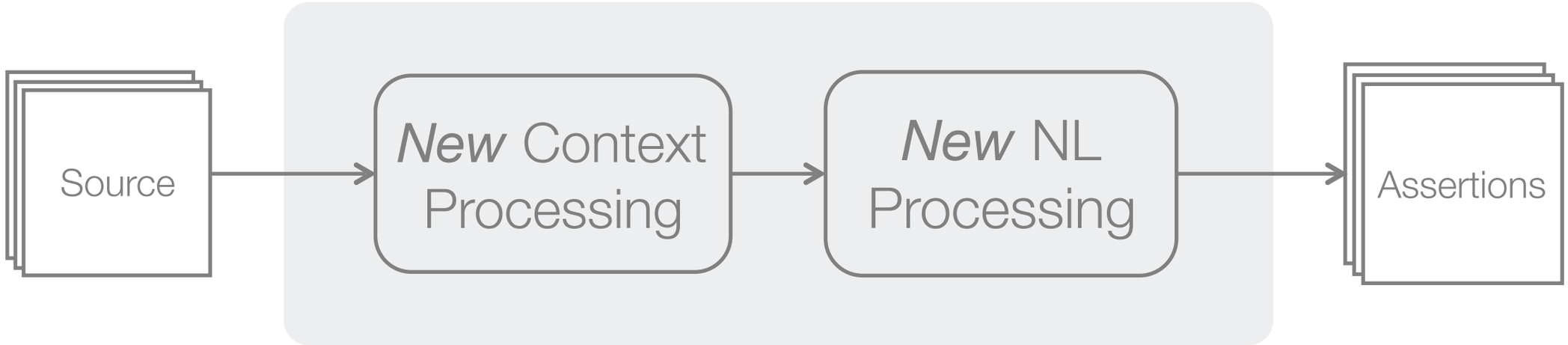
```
/**
 * Note that it is not possible to delete the last element by
 * immediately calling remove() on the iterator, as a call to remove()
 * before a call to next() will result in an illegal state.
 */
```



- Exploiting **unstructured** natural language information
- For testing both at the unit and the **integration** level

Present

```
/**
 * Note that it is not possible to delete the last element by
 * immediately calling remove() on the iterator, as a call to remove()
 * before a call to next() will result in an illegal state.
 */
```



- Exploiting **unstructured** natural language information
- For testing both at the unit and the **integration** level
- Testing functional properties only

Future

Future

- Testing at the **system** level for complex systems

Future

- Testing at the **system** level for complex systems
- Testing both functional and **non-functional properties**

Future

- Testing at the **system** level for complex systems
- Testing both functional and **non-functional properties**
- Exploiting **any kind** of natural language information