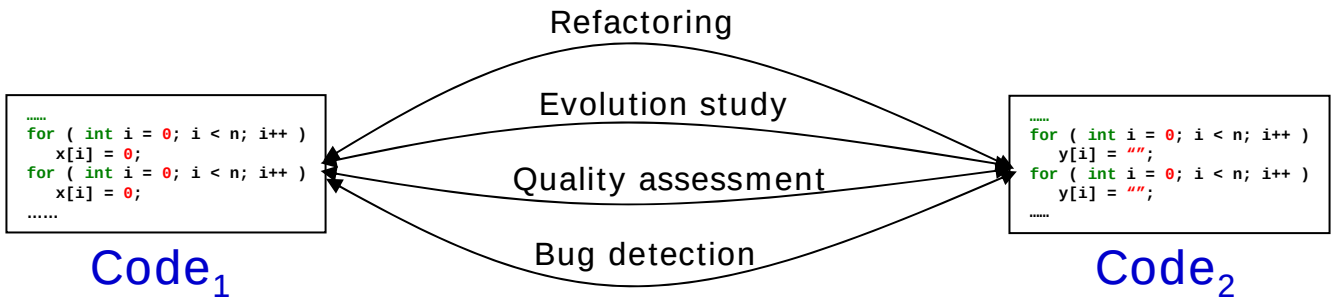
Deckard:
Scalable and Accurate Tree-based Detection of Code Clones

Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, Stephane Glondu

The Problem

- Find "similar" code in large code bases, often referred to as Clone Detection.

- Many applications in software engineering



Clone Detection Techniques

- String-based
  - E.g., Baker (1995, 1997)
- Token-based
  - E.g., CCFinder (2002), CP-Miner (2004)
- Tree-based
  - E.g., CloneDR (1998, 2004)
- Semantic-based
  - E.g., Komondoor & Horwitz (2001)

Limitations of Existing techniques

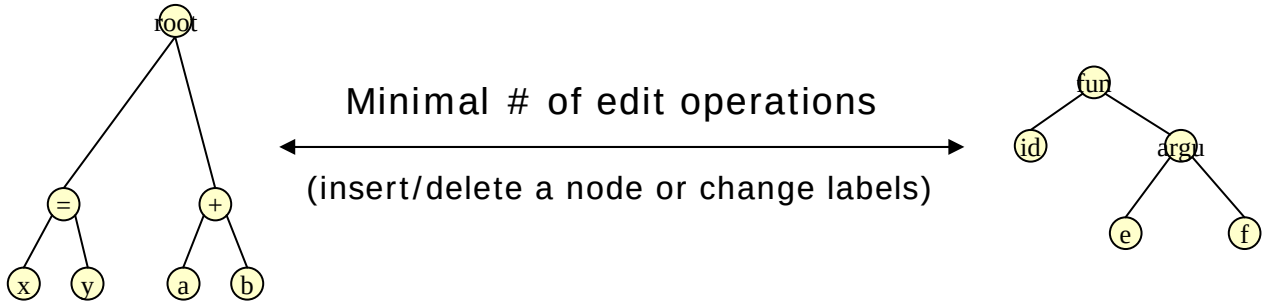
- String-based, token-based
  - Sensitive to code format
  - E.g., `x = a*b+c` vs. `z = c+a*b`
- Tree-based, semantic-based
  - Not scalable

Deckard's Goal

- More robust against code modification
- Scalable to million-line programs

Tree Similarity and Code Clones

- Well defined tree similarity problem
  - Tree editing distance $\delta(T_1, T_2)$:



  - T_1 and T_2 are *σ-similar* if $\delta(T_1, T_2) < \sigma$

- Code clones based on δ

```
……
for ( int i = 0; i < n; i++ )
   x[i] = 0;
for ( int i = 0; i < n; i++ )
   x[i] = 0;
……
```

Code$_1$

clone pair, if $\delta(T_1, T_2) < \sigma$

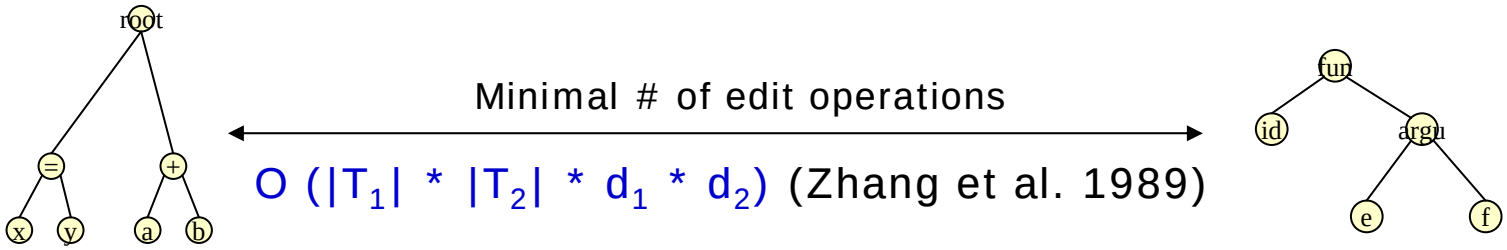
```
……
for ( int i = 0; i < n; i++ )
   y[i] = "";
for ( int i = 0; i < n; i++ )
   y[i] = "";
……
```

Code$_2$

Not Scalable

A clone detection directly based on the definition would not scale:

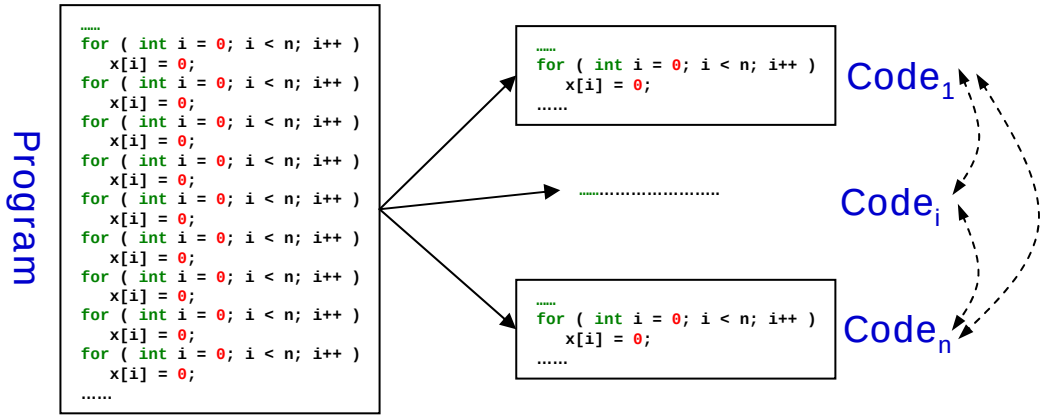
- Computing tree editing distance is expensive



$O(|T_1| * |T_2| * d_1 * d_2)$ (Zhang et al. 1989)

where d_i is the minimum of the depth of T_i and the number of leaves of T_i

- Need to compare many pairs of subtrees

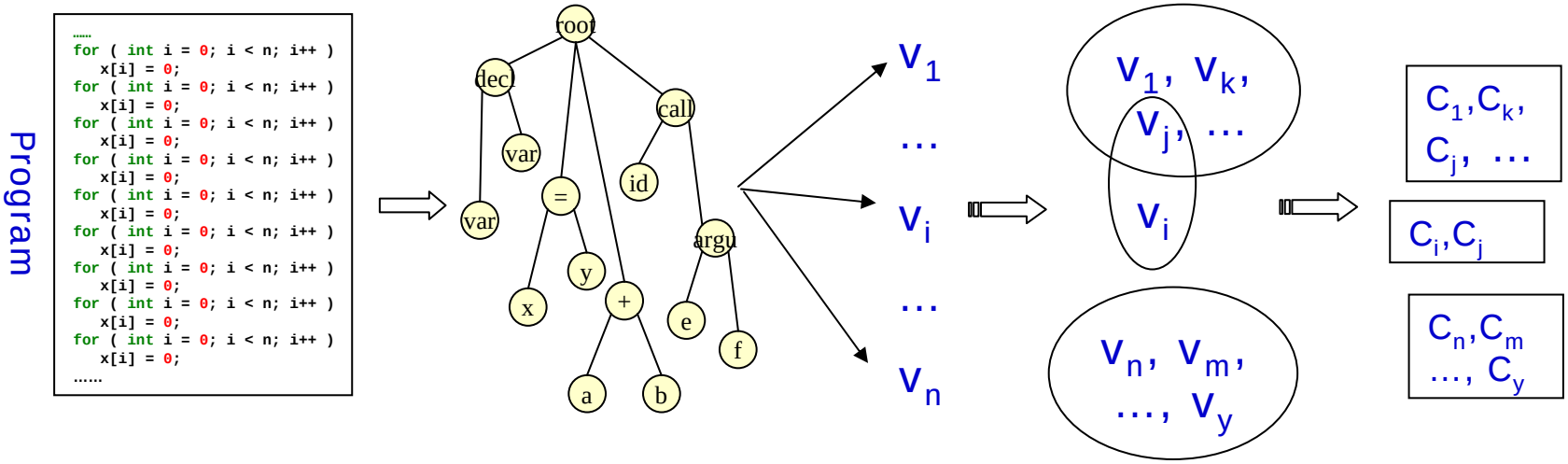


Quadratic # of Tree Comparison

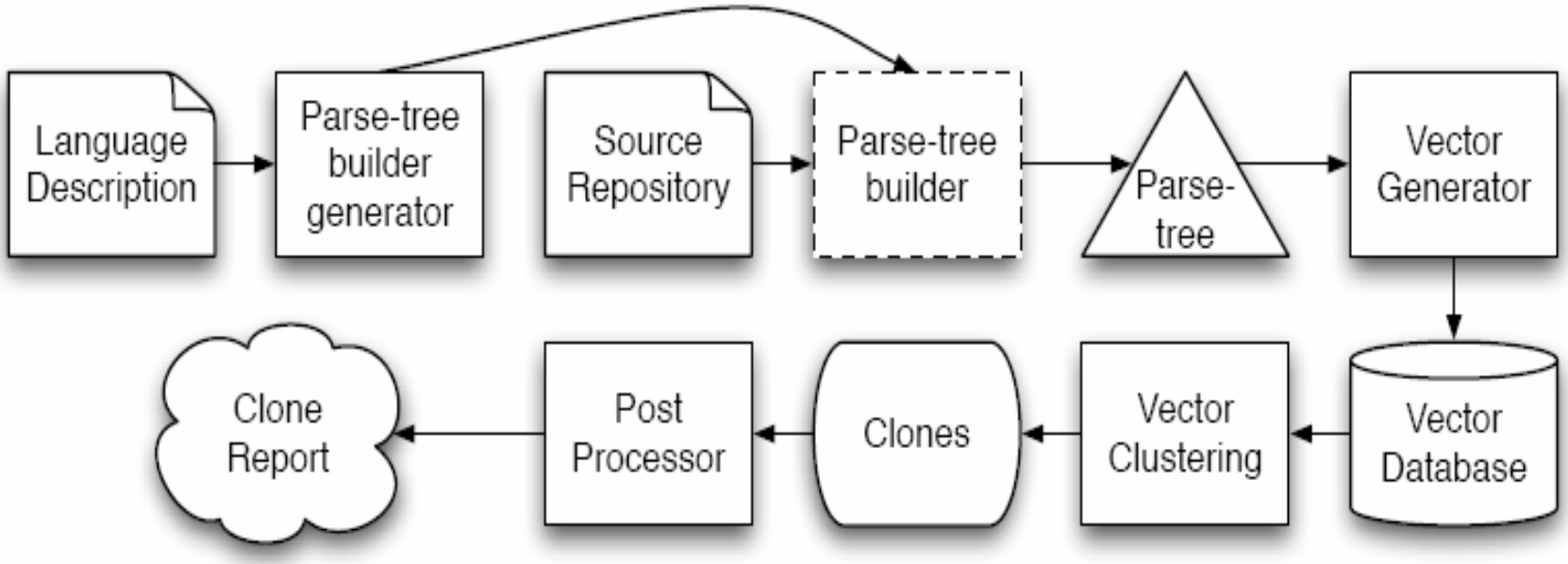
The Idea – Numerical Vectors

- Numerical vectors are much easier to compare than trees
  - Suppose $v_1 = \langle x_1, \ldots, x_n \rangle$ and $v_2 = \langle y_1, \ldots, y_n \rangle$
  - Hamming distance (or the l_1 *norm*): $\mathcal{H}(v_1, v_2) = \sum^{n} |x_i - y_i|$
  - Euclidean distance (or the l_2 *norm*): $\mathcal{D}(v_1, v_2) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

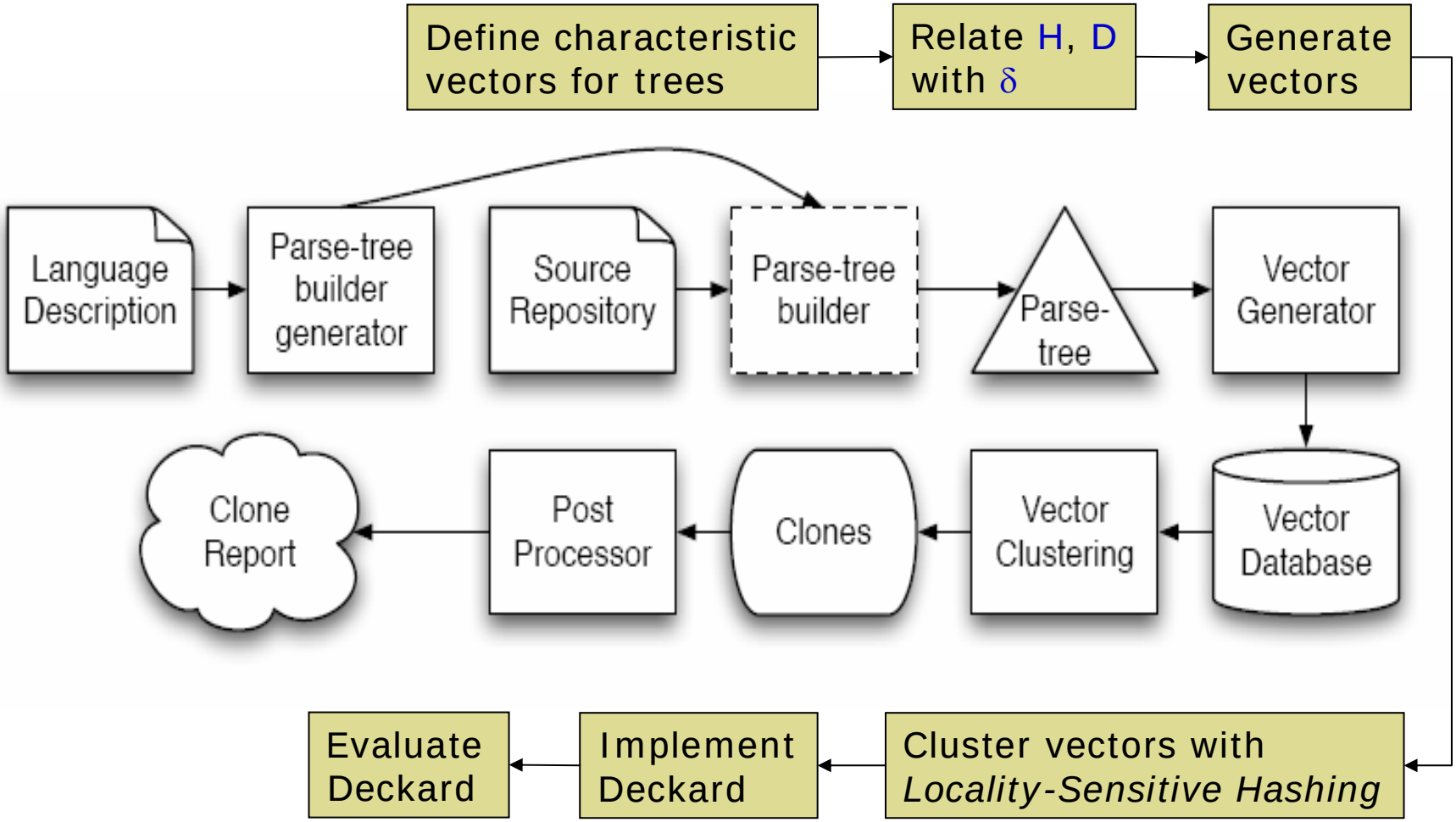
- Characterize parse trees as numerical vectors
  - Cluster vectors based on numerical distances
  - Code corresponding to vectors within a same cluster are clones



Deckard's High Level Architecture



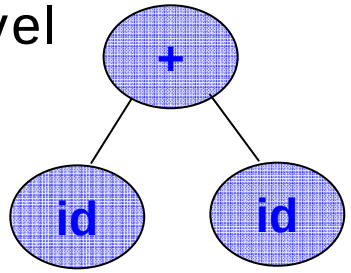
Outline


Define characteristic vectors for trees
Relate H, D with δ
Generate vectors
Language Description
Parse-tree builder generator
Source Repository
Parse-tree builder
Parse-tree
Vector Generator
Clone Report
Post Processor
Clones
Vector Clustering
Vector Database
Evaluate Deckard
Implement Deckard
Cluster vectors with *Locality-Sensitive Hashing*

Characteristic Vectors for Trees

Definitions

- *q-level atomic patterns*
  - Labeled complete binary trees of height q
  - E.g., 2-level



  - At most $|L|^{2^q-1}$ patterns, where L is the set of labels

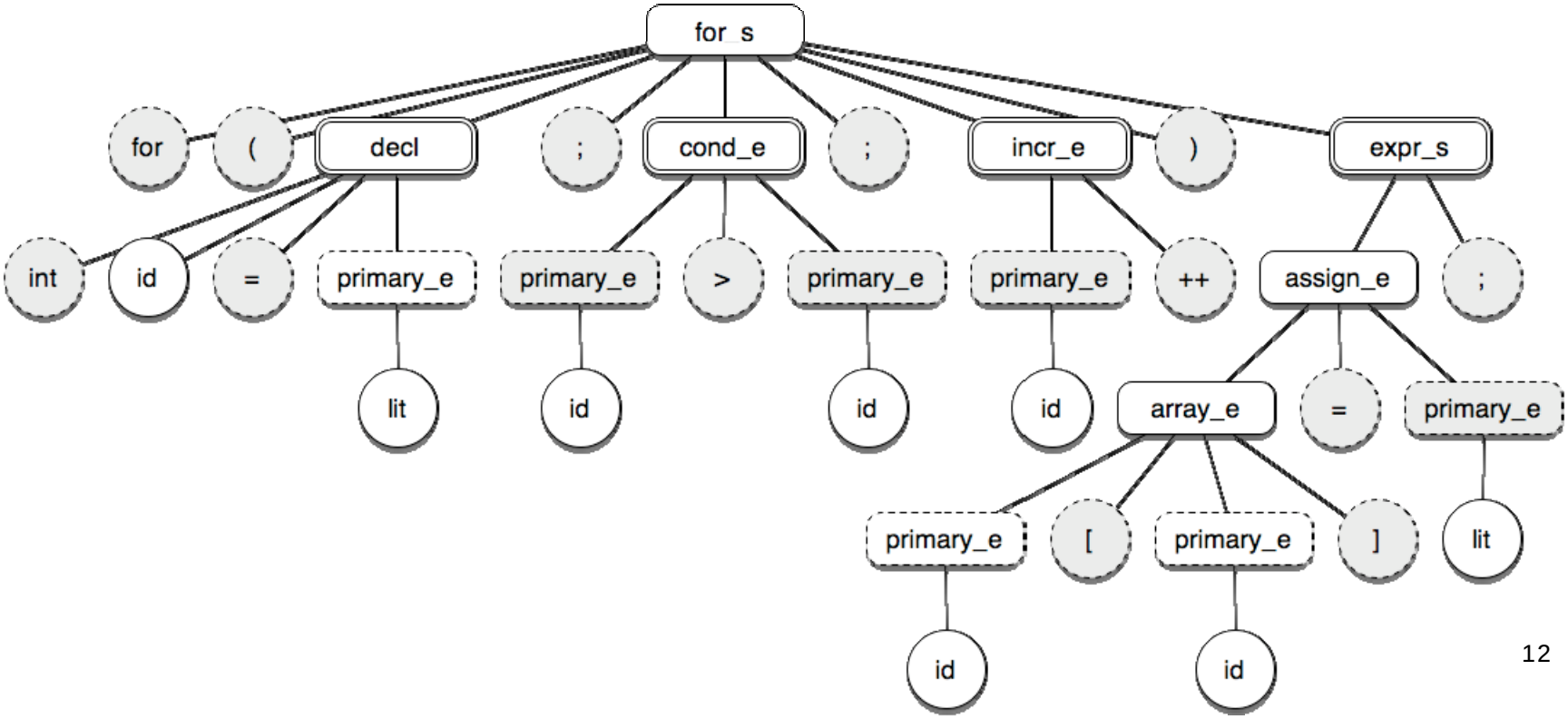
- *q-level vector* for a tree or tree-forest T
  - <b_1, …, b_k>, where $k \leq |L|^{2^q-1}$
  - Each b_i counts # of i-th pattern in T

An Sample Characteristic Vector

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

1-level patterns:

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

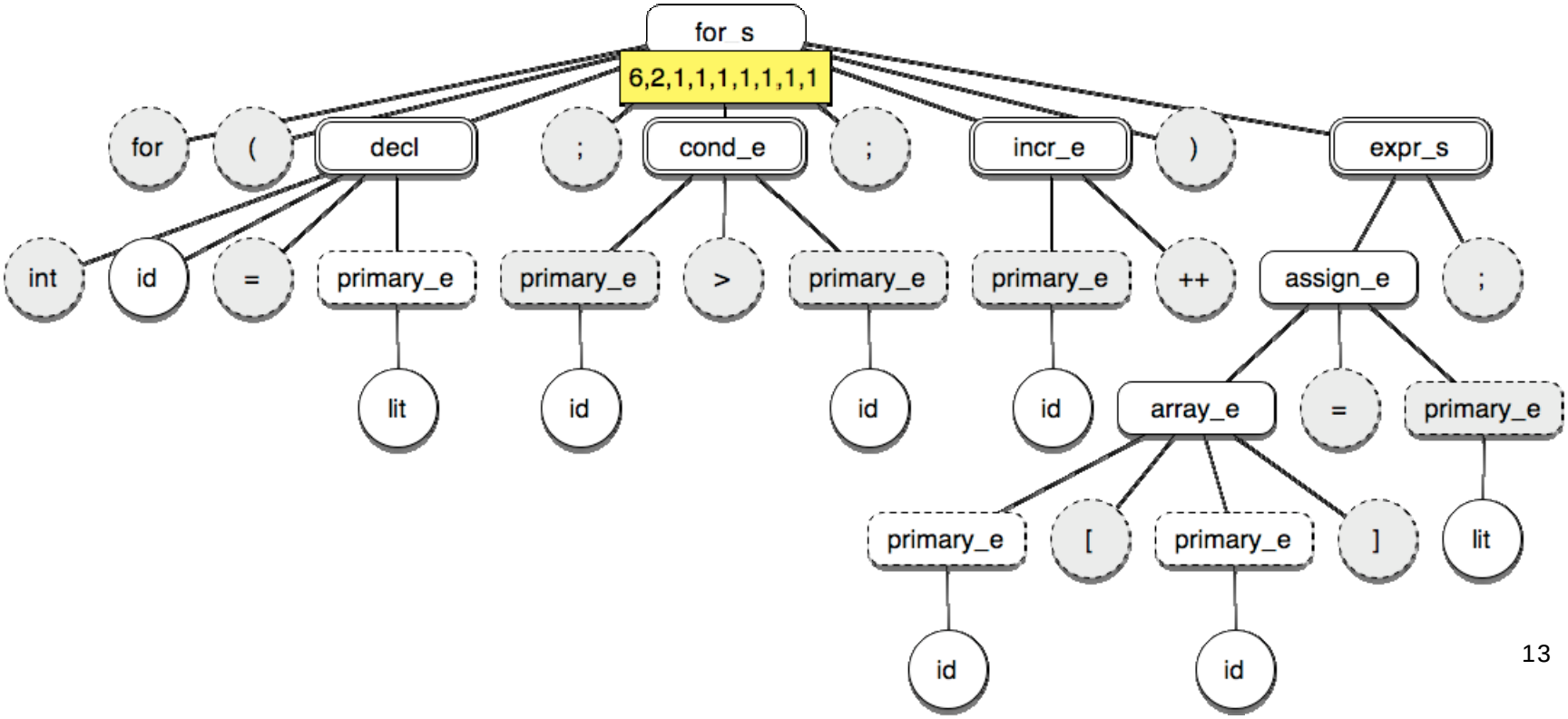


An Sample Characteristic Vector

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

1-level patterns:

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>



Relate $H(T_1,T_2)$, $D(v_1,v_2)$, and $\delta(v_1,v_2)$

- Theorem 1 (Yang et al. 2005):

  If $\delta(T_1,T_2) = k$, then $H(v_1,v_2) \le (4q-3)k$

- Theorem 2: for integer vectors,

  $$\sqrt{H(v_1,v_2)} \le D(v_1,v_2) \le H(v_1,v_2)$$

- Theorem 3: for tree characteristic vectors,

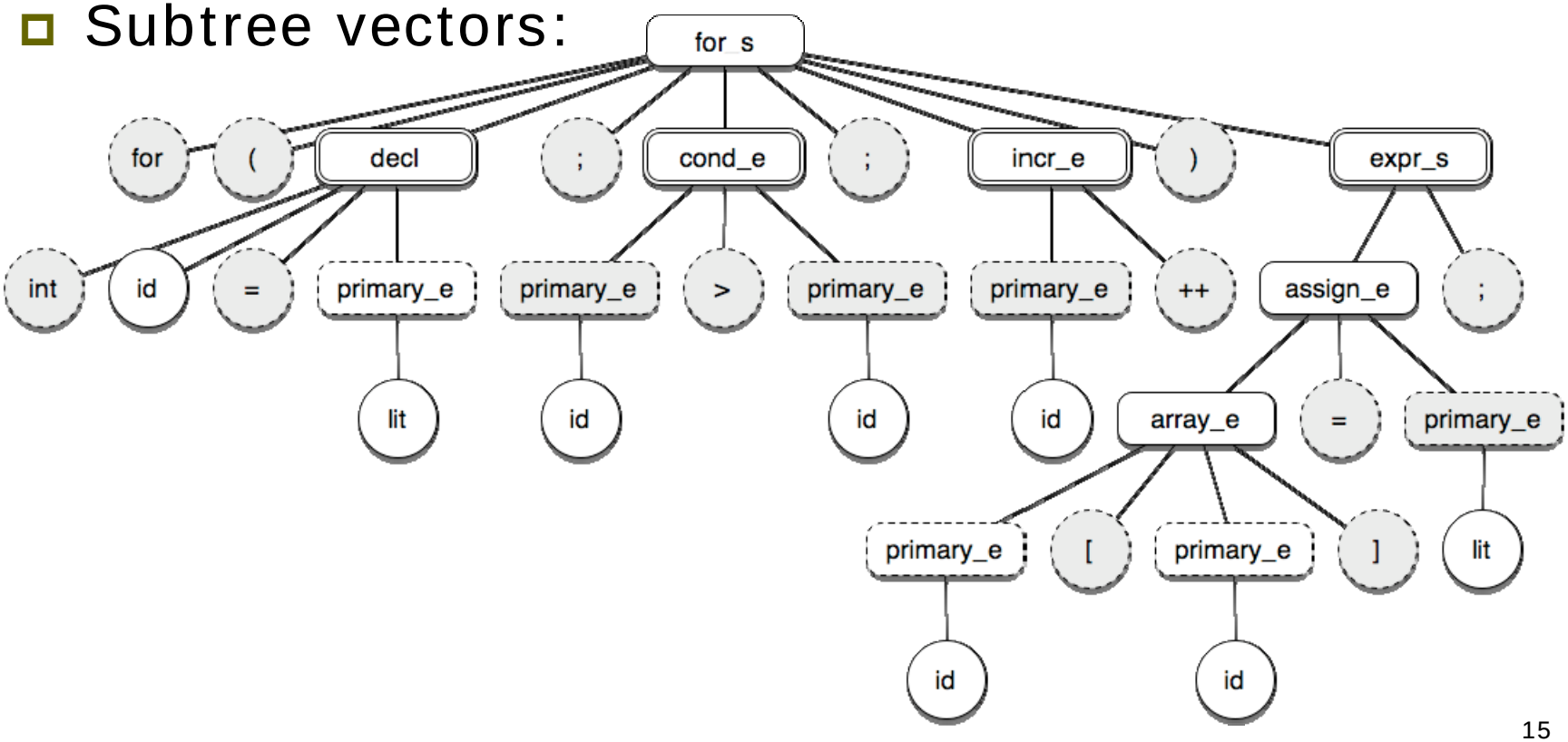
  $$\frac{\sqrt{H(v_1,v_2)}}{4q-3} \le \frac{D(v_1,v_2)}{4q-3} \le \frac{H(v_1,v_2)}{4q-3} \le k$$

Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

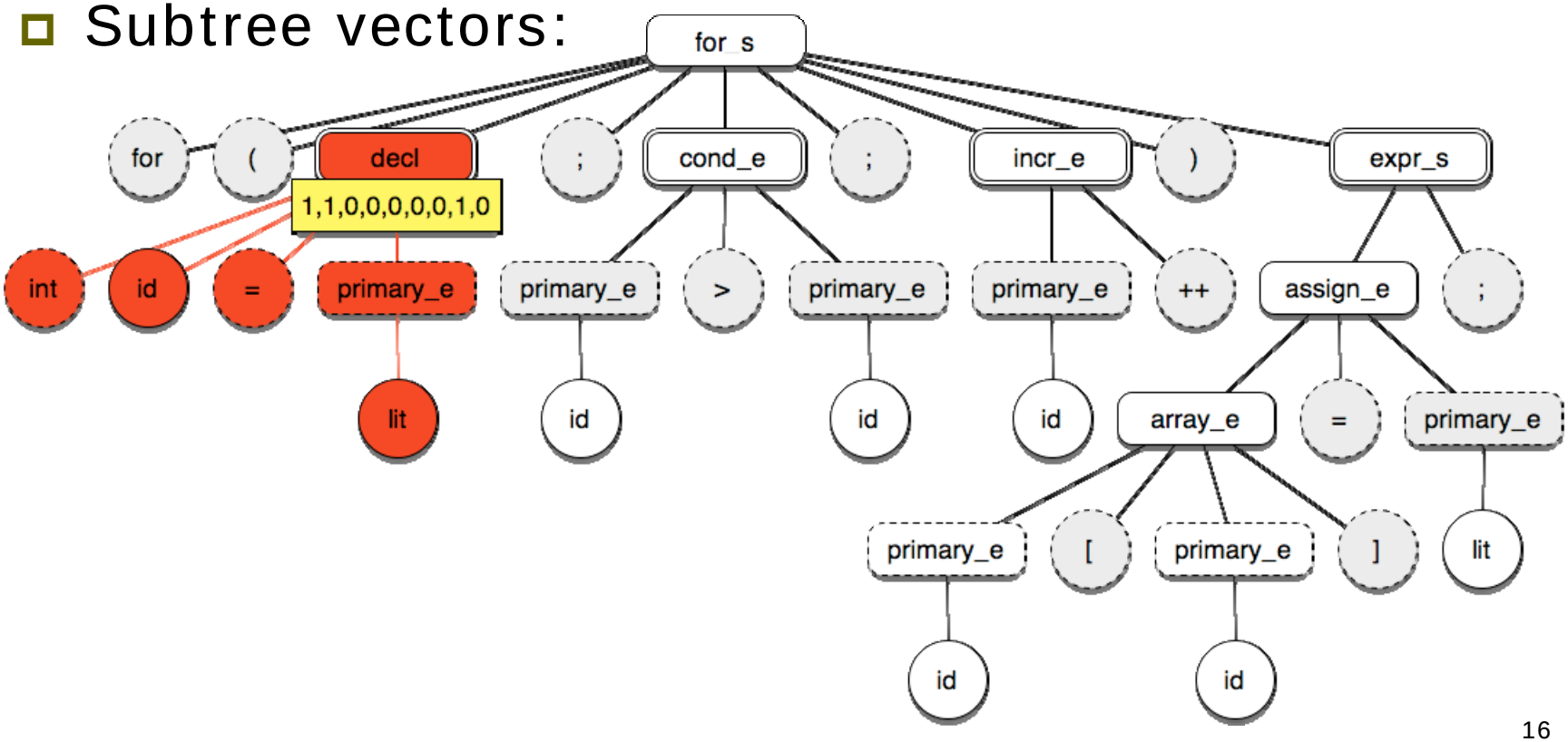


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

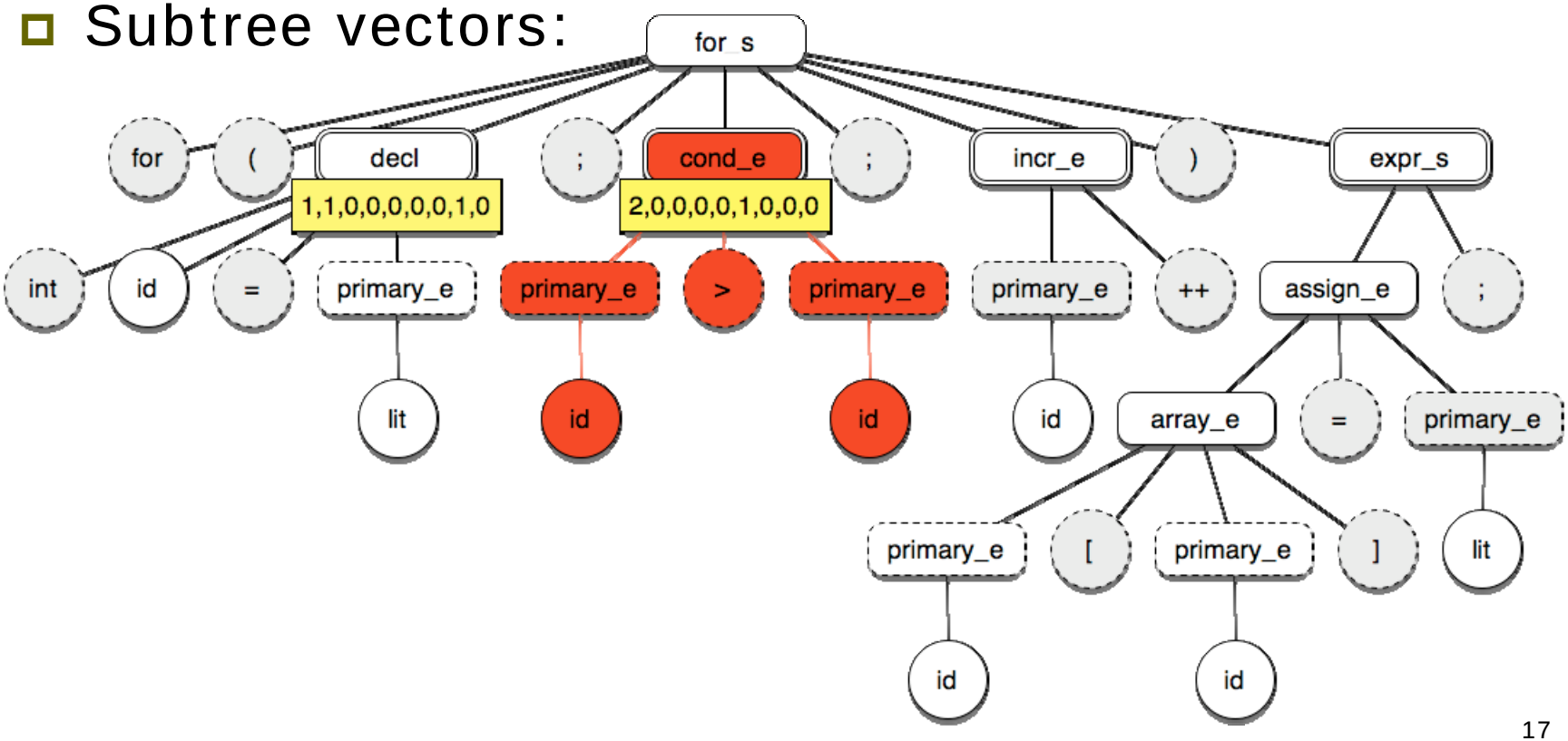


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

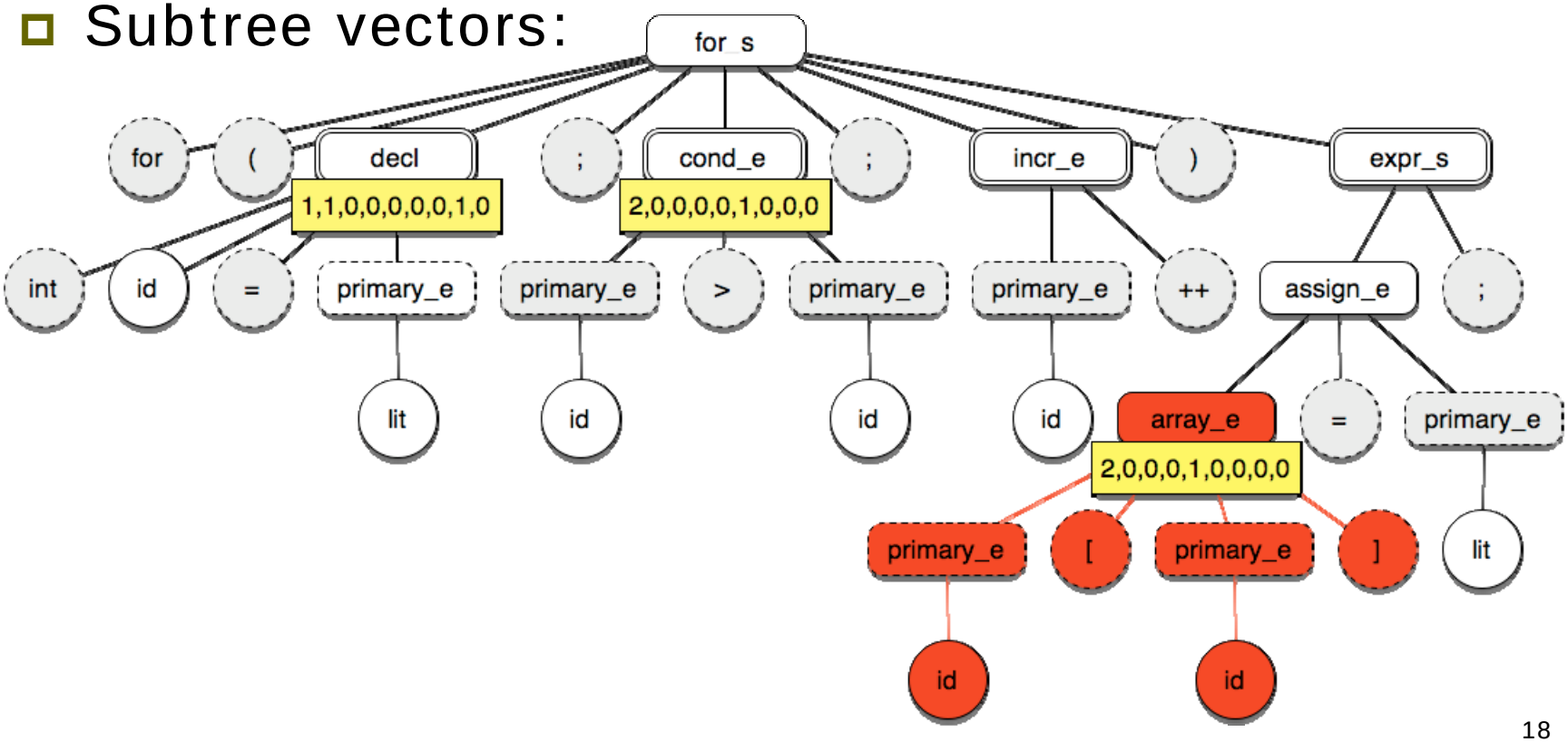


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

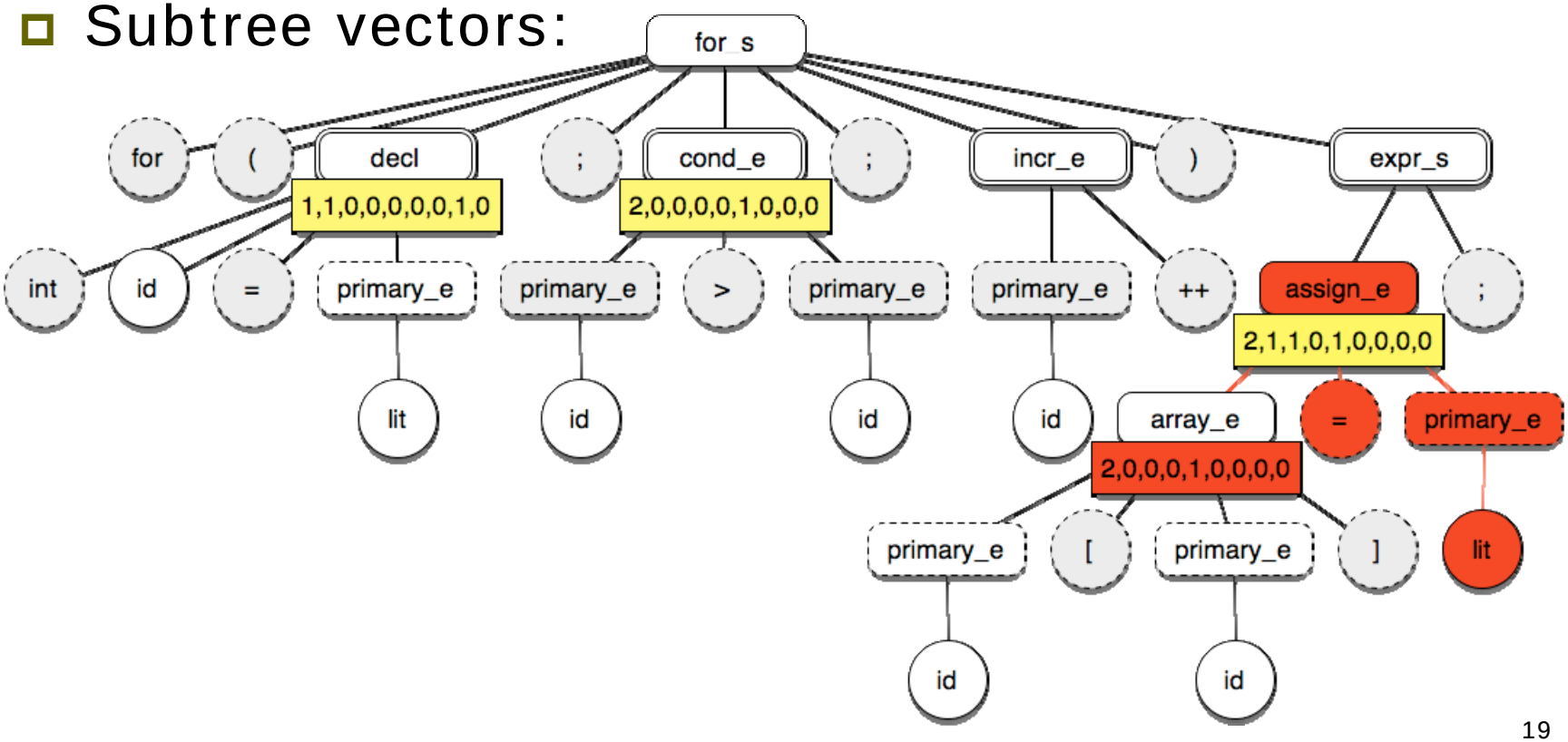


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

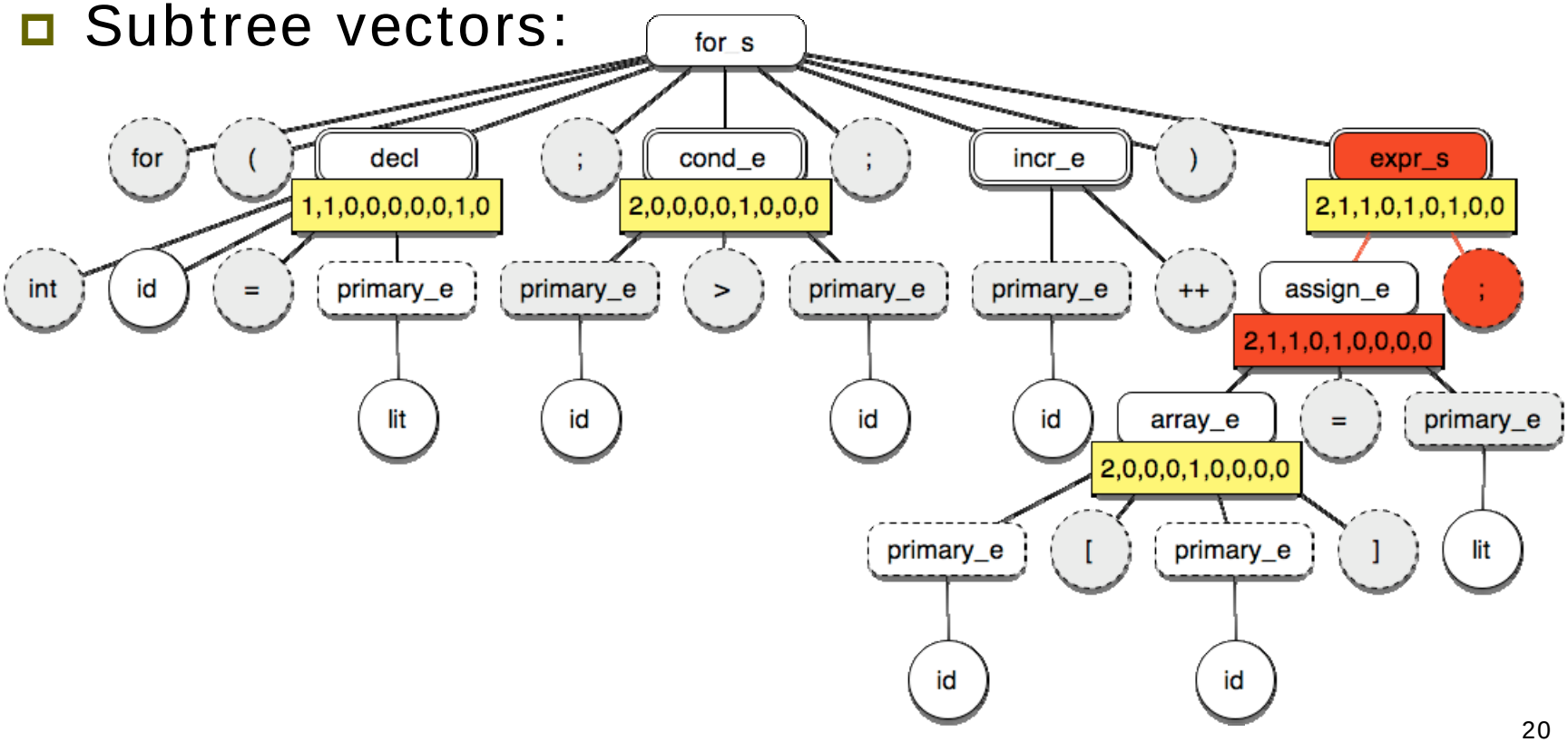


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

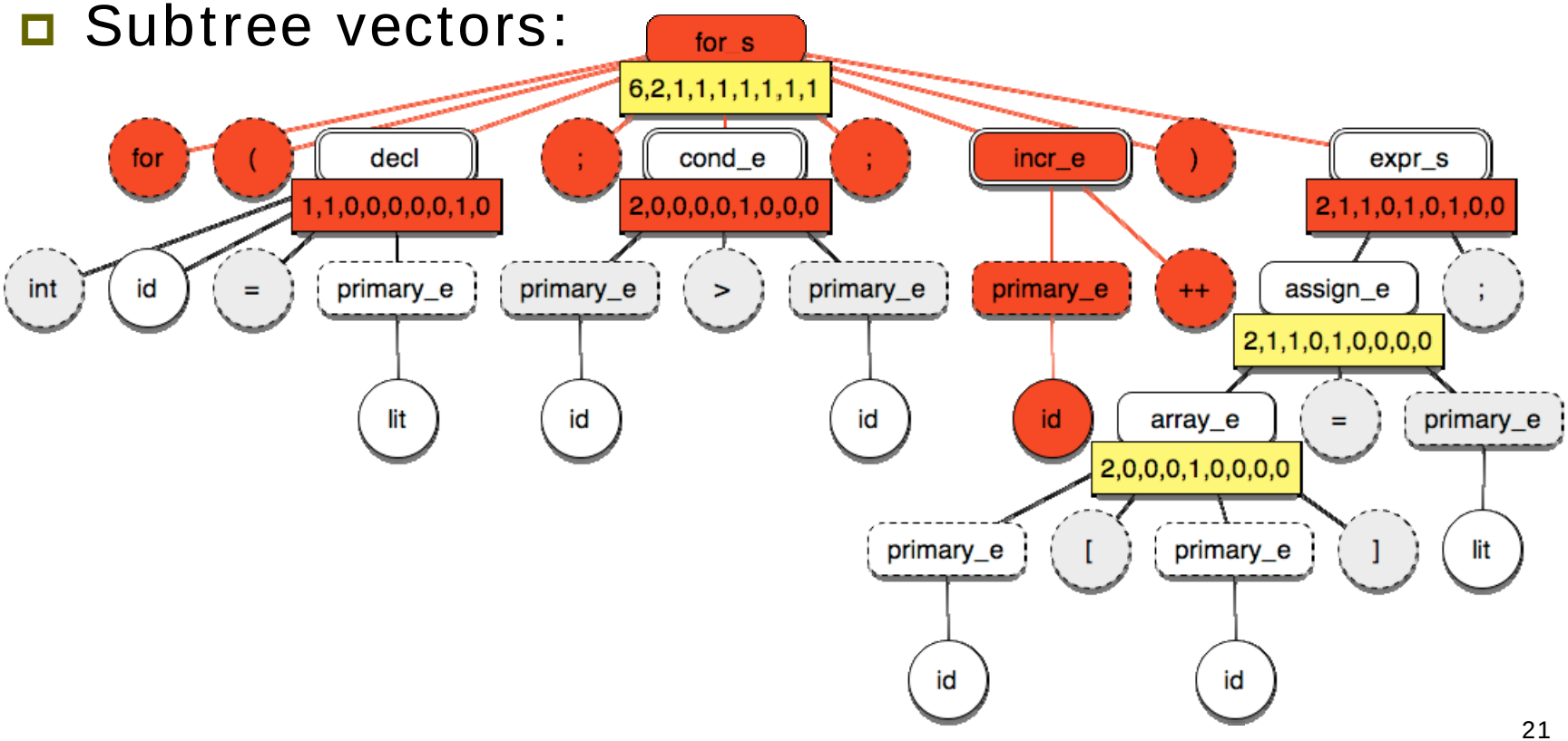


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Subtree vectors:

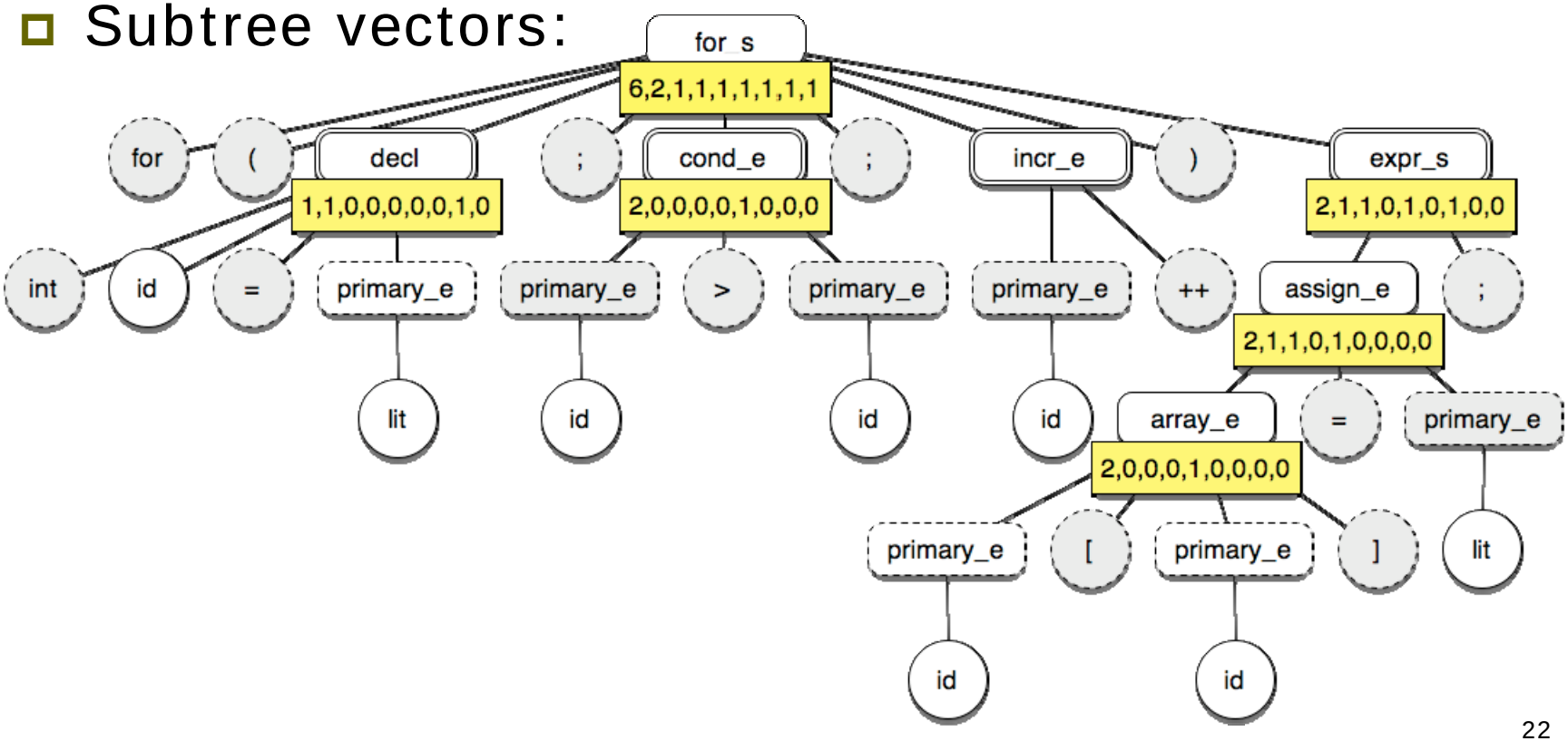


Map Trees to Numerical Vectors (1)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

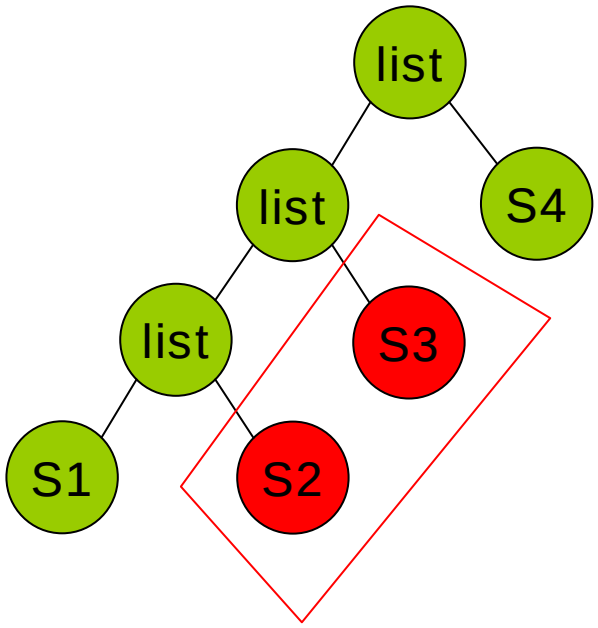
<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

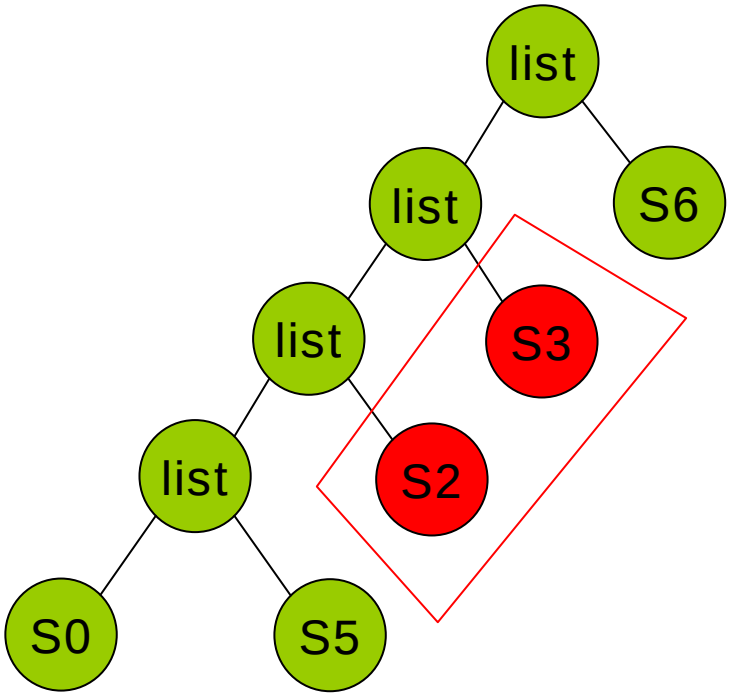
- Subtree vectors:



Map Trees to Numerical Vectors (2)

- Forest vectors
  - For sequences of program elements
  - Why needed?
    - S1; S2; S3; S4;
    - S0; S5; S2; S3; S6



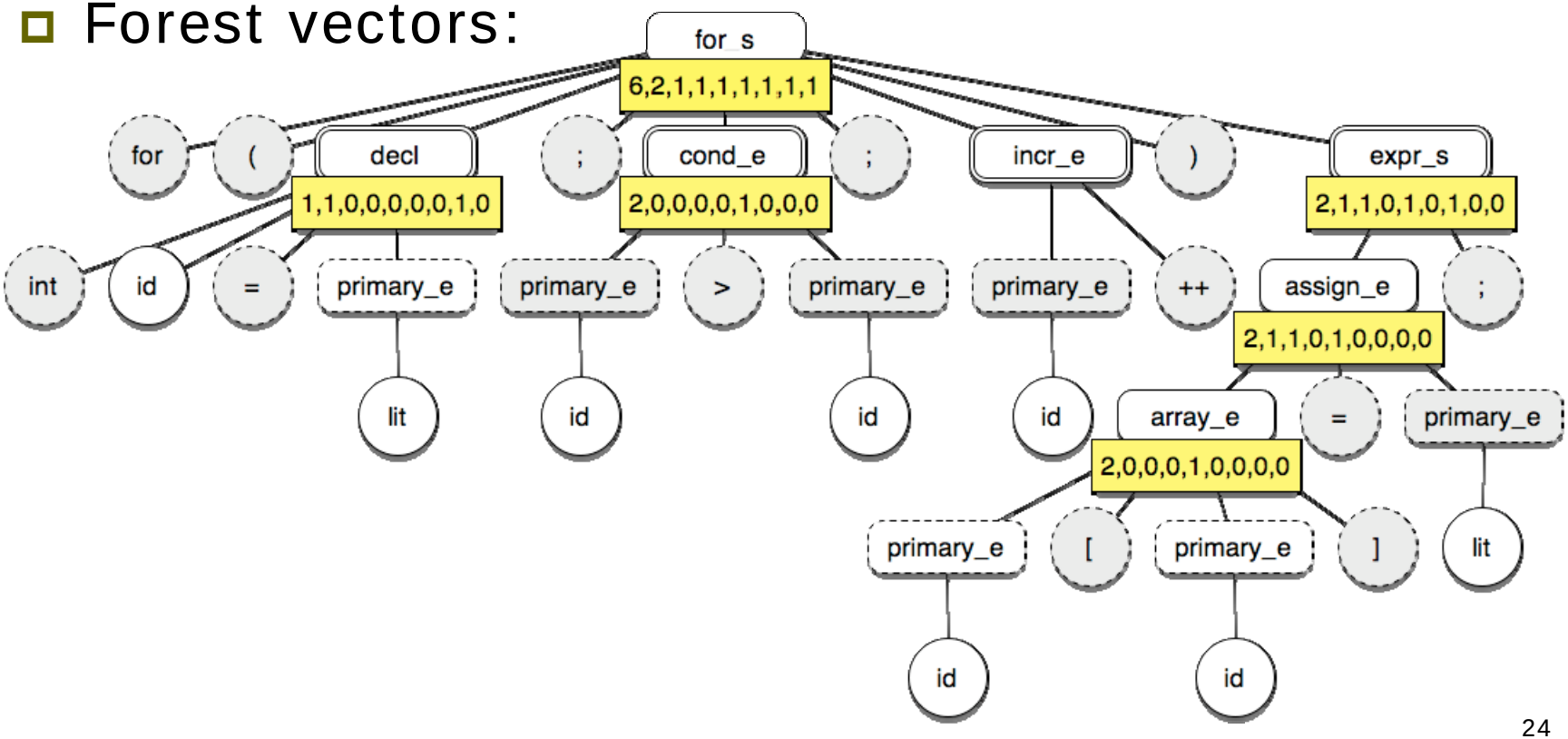


Map Trees to Numerical Vectors (2)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Forest vectors:

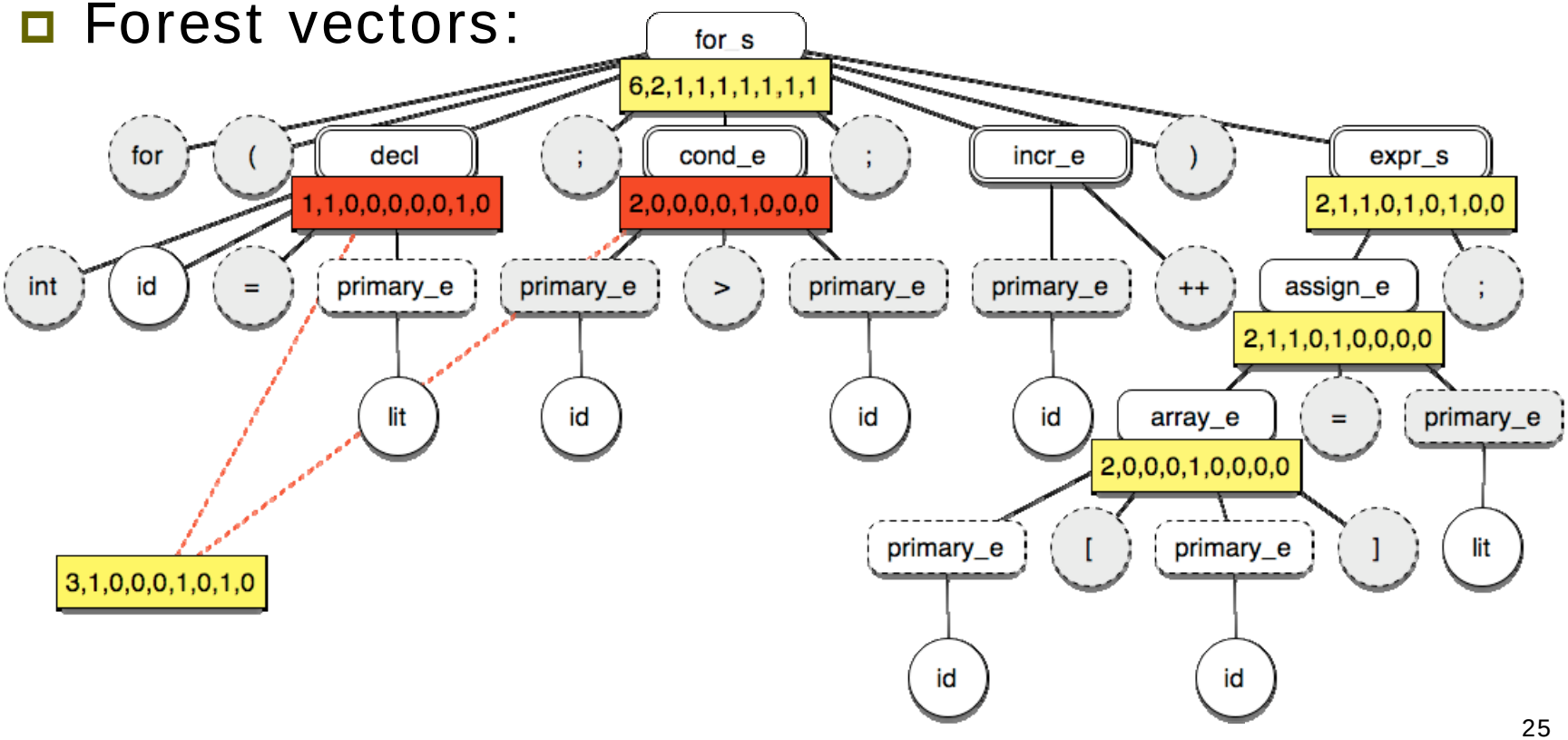


Map Trees to Numerical Vectors (2)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Forest vectors:

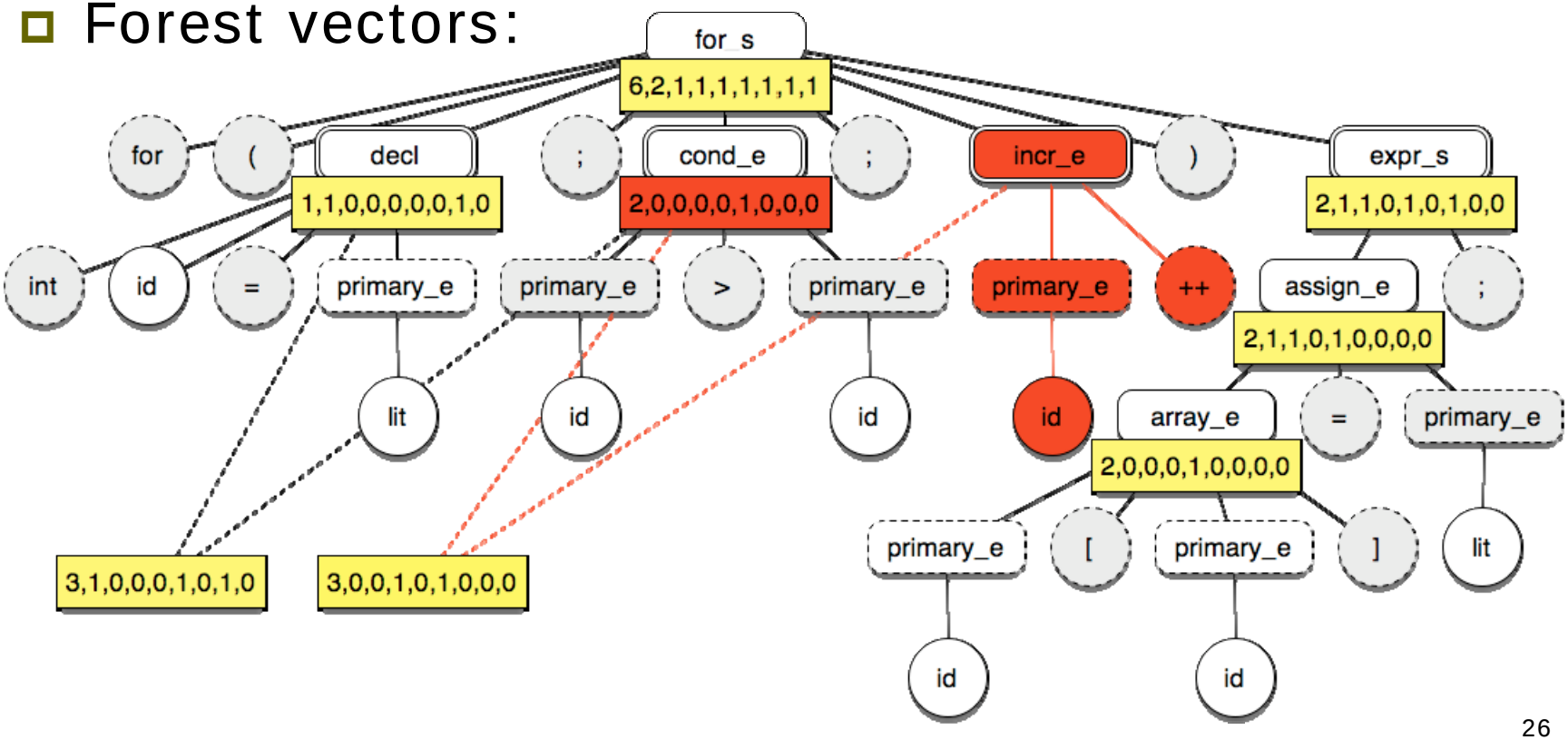


Map Trees to Numerical Vectors (2)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Forest vectors:

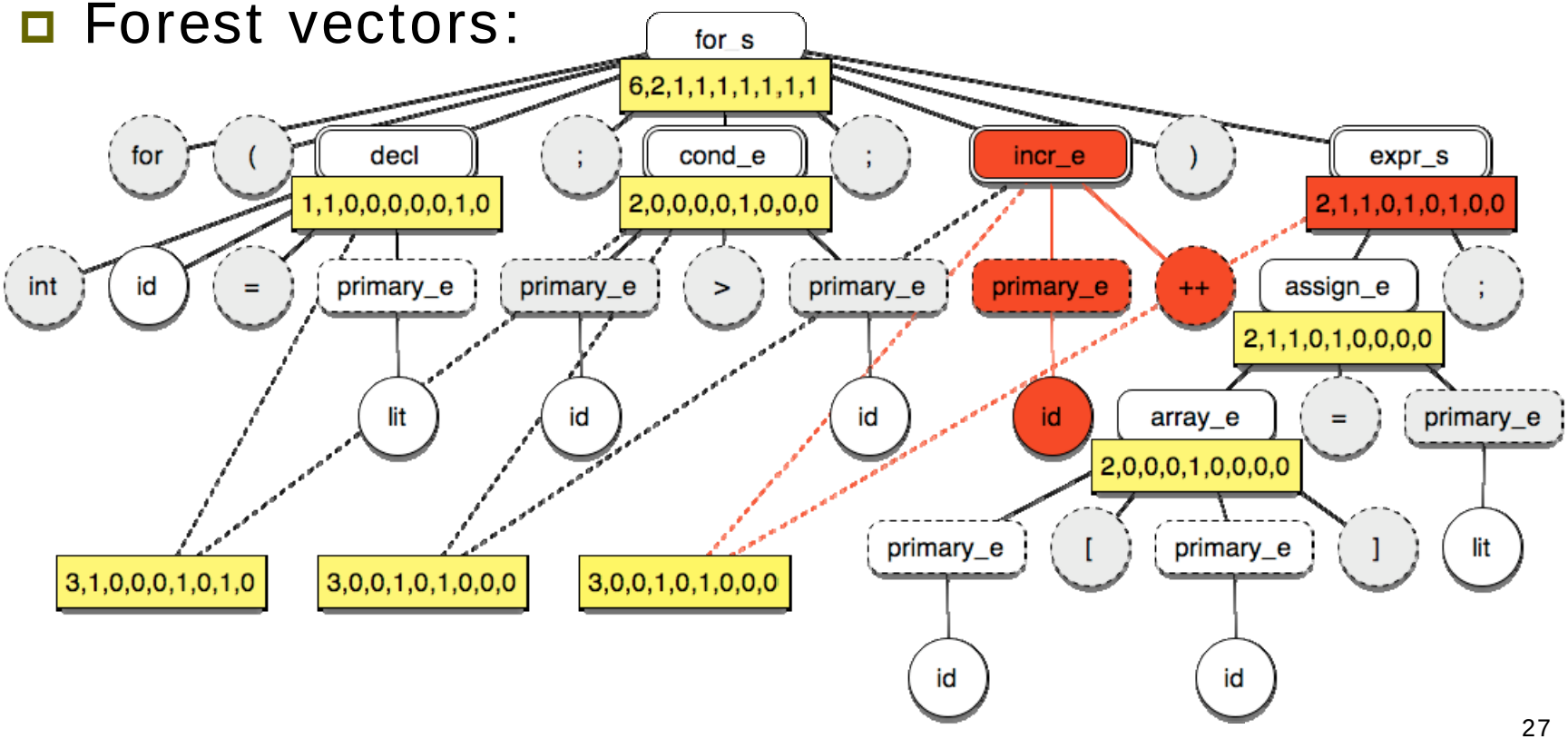


Map Trees to Numerical Vectors (2)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- Forest vectors:

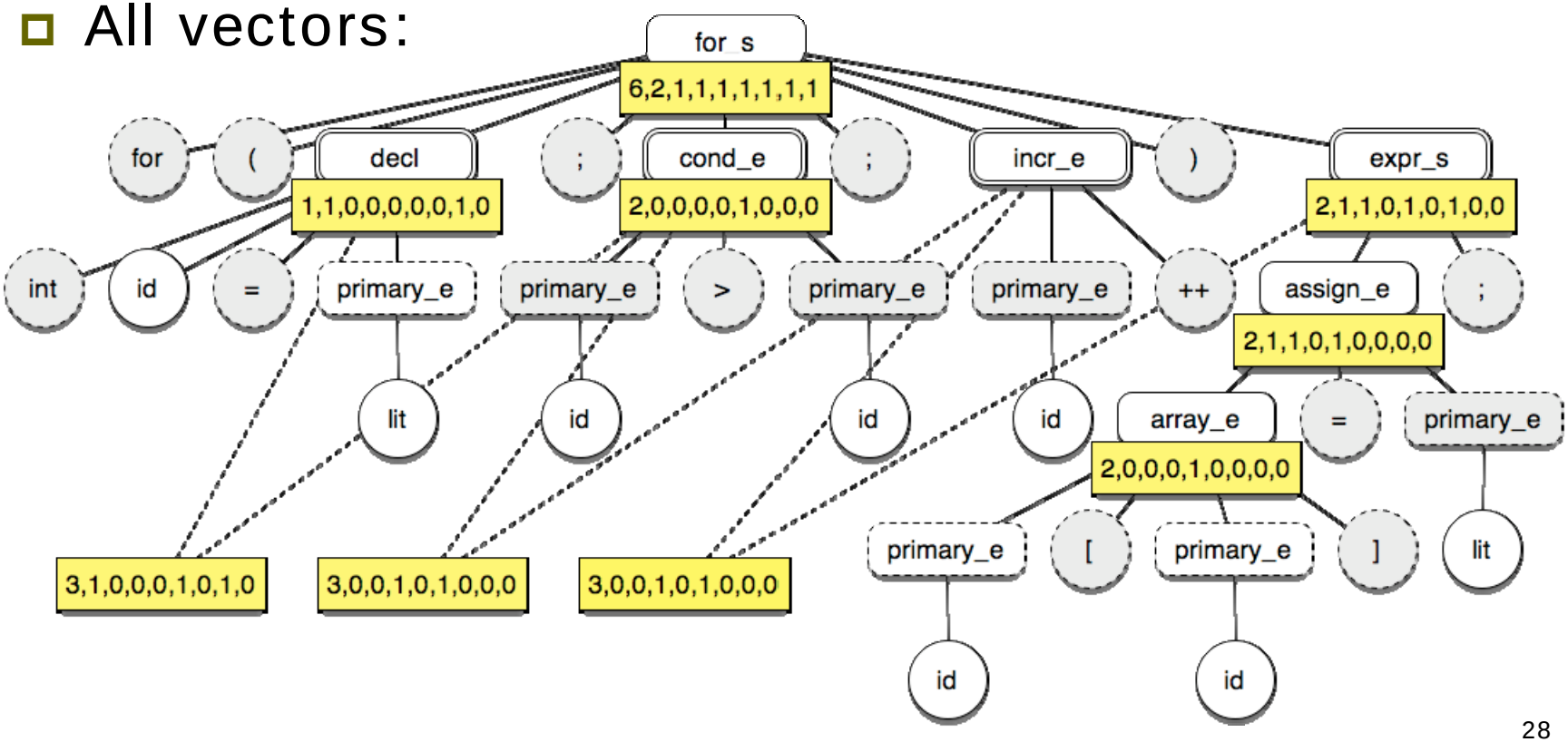


Map Trees to Numerical Vectors (2)

```
for ( int i = 0; i < n; i++ )
   x[i] = 0;
```

<id, lit, assign_e, incr_e, array_e, cond_e, expr_s, decl, for_s>

- All vectors:

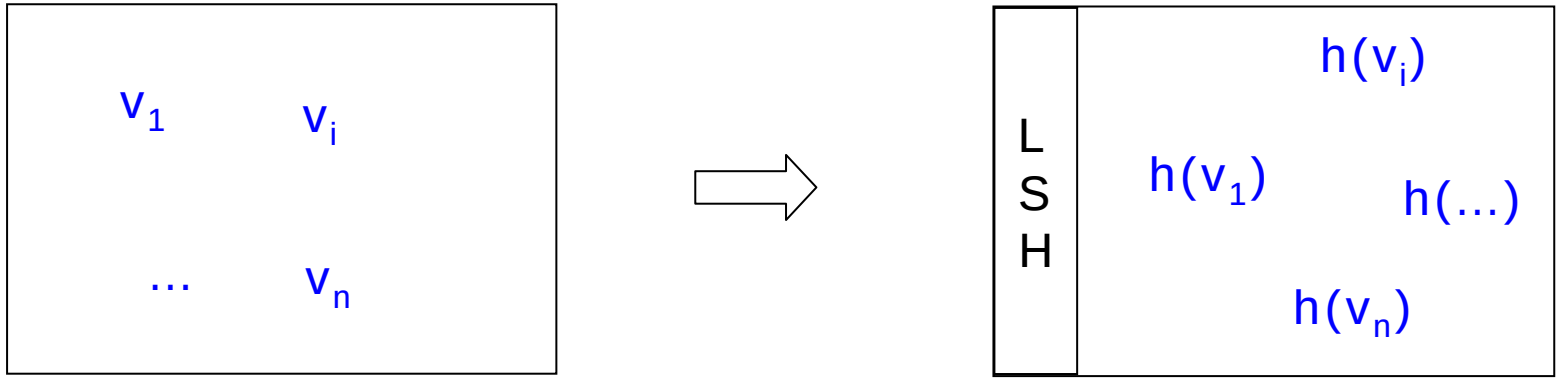


Vector Clustering

- Have millions of generated vectors
- Need to efficiently cluster “close” vectors together
  - Hashing
- Locality-sensitive hashing (LSH)
  - A family of hash functions, s.t., with high probability,
    - Similar vectors hashed to a same hash value
    - Distant vectors hashed to different hash values
  - http://web.mit.edu/andoni/www/LSH/index.html

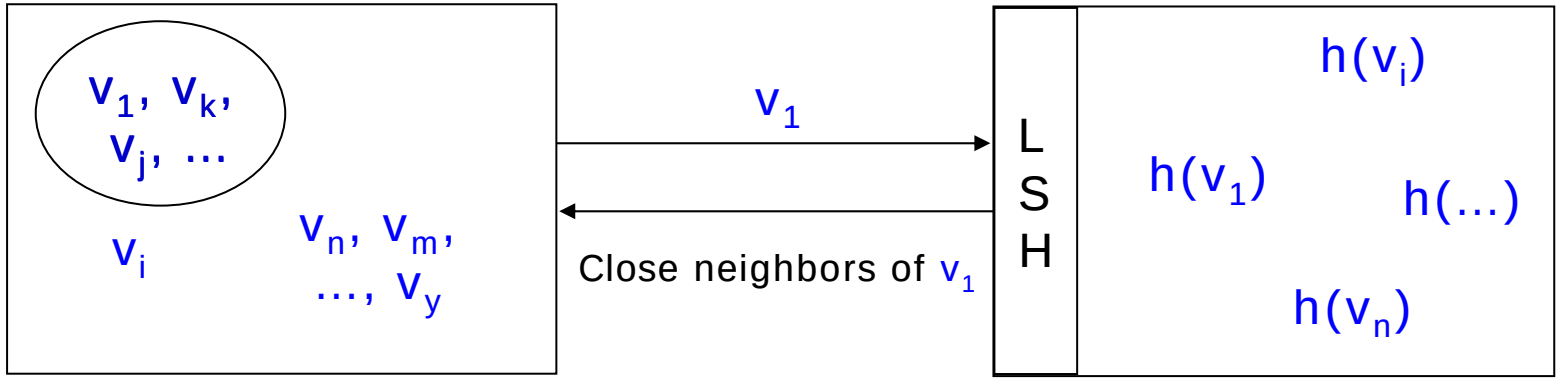
LSH-based Clone Detection (1)

- Rely on LSH to construct hash tables for vectors



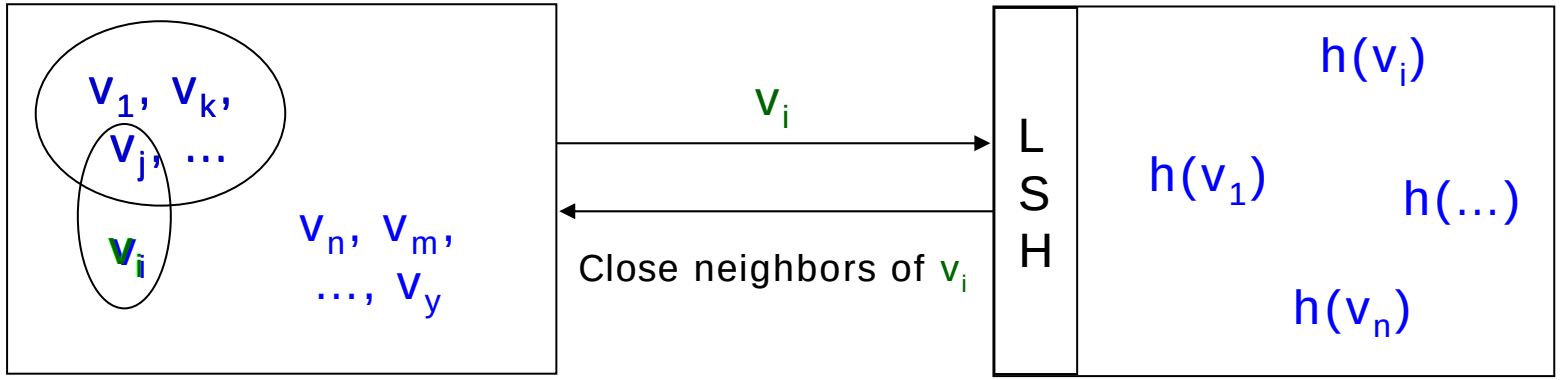
LSH-based Clone Detection (2)

- Query LSH for vector clusters



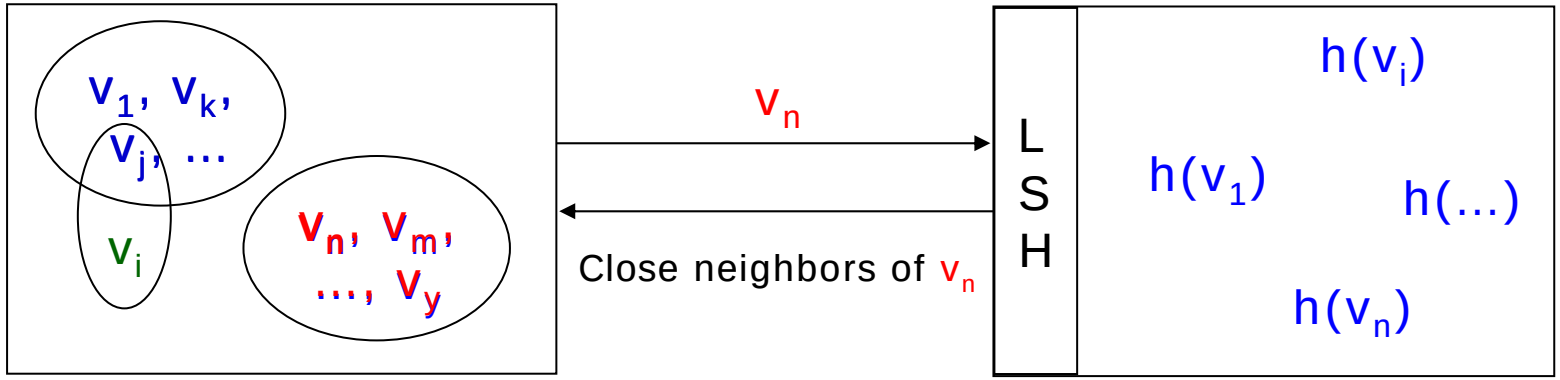
LSH-based Clone Detection (2)

- Query LSH for vector clusters



LSH-based Clone Detection (2)

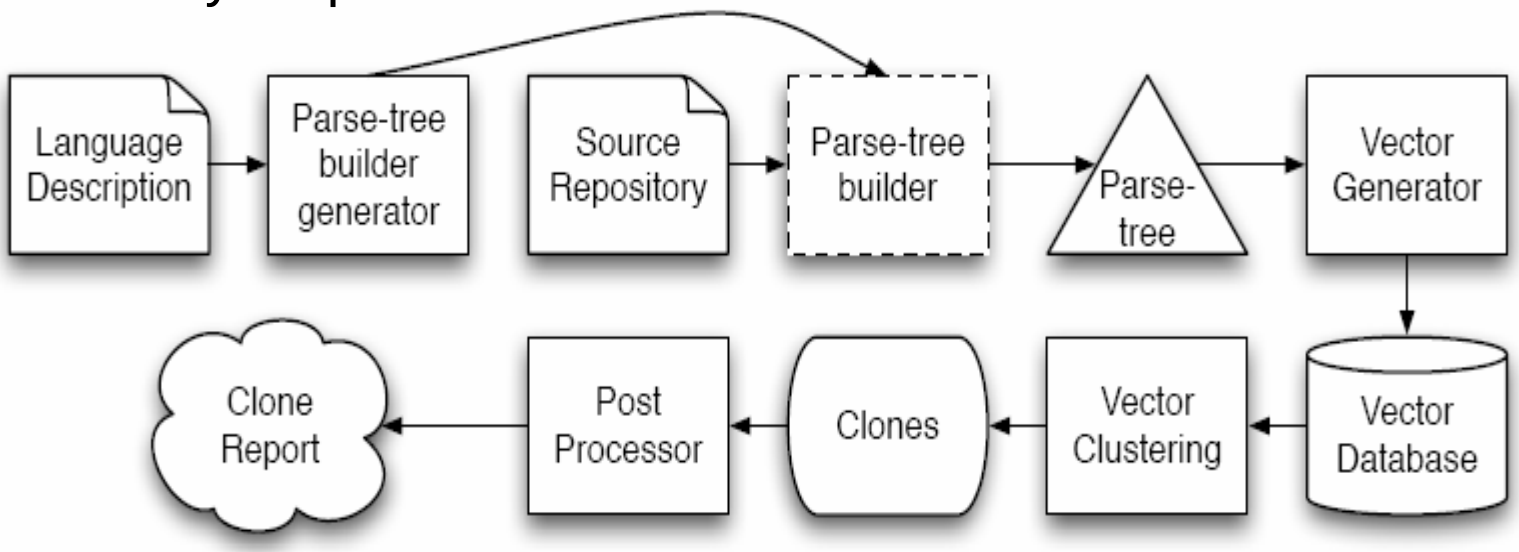
- Query LSH for vector clusters



- Post-process to reduce "false" clones
  - Small clones
  - Overlapping/duplicate clones

Implementation

- Modified bison for constructing parse trees
- 1-level characteristic vectors for trees
- Group vectors according to their sizes
  - Able to handle size-sensitive clones
  - Less stress on LSH
  - Easy to parallelize



Evaluation Results

- Comparison with two state-of-the-art tools
  - Tree-based: CloneDR (for Java)
    - An evaluation version of a commercial tool
  - Token-based: CP-Miner (for C)
- Evaluation metrics
  - Clone quantity (% of cloned code)
  - Clone quality (% of false clones)
  - Scalability

Test Programs

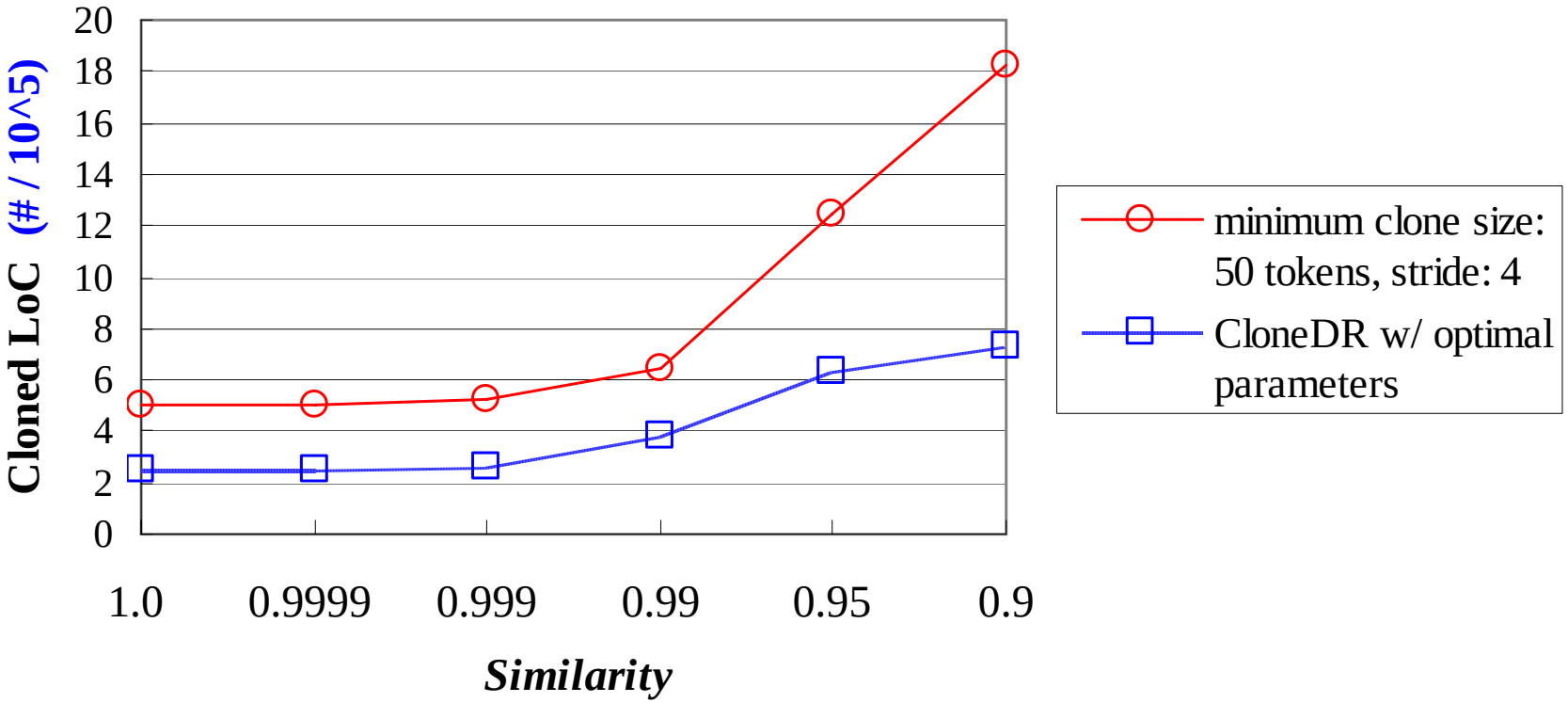
Application	Language	# files	# lines
Linux kernel 2.6.16	C	7,988	5,287,090
JDK 1.4.2	Java	8,534	2,418,767
GCC 3.3.6	C	7,828	1,246,322
PostgreSQL 8.1.0	C	725	606,015
Derby 10.0.2.1	Java	1,418	523,939
Apache 2.2.0	C	537	269,287

Test Programs

Application	Language	# files	# lines
Linux kernel 2.6.16	C	7,988	5,287,090
JDK 1.4.2	Java	8,534	2,418,767
GCC 3.3.6	C	7,828	1,246,322
PostgreSQL 8.1.0	C	725	606,015
Derby 10.0.2.1	Java	1,418	523,939
Apache 2.2.0	C	537	269,287

Detected More Clones (1)

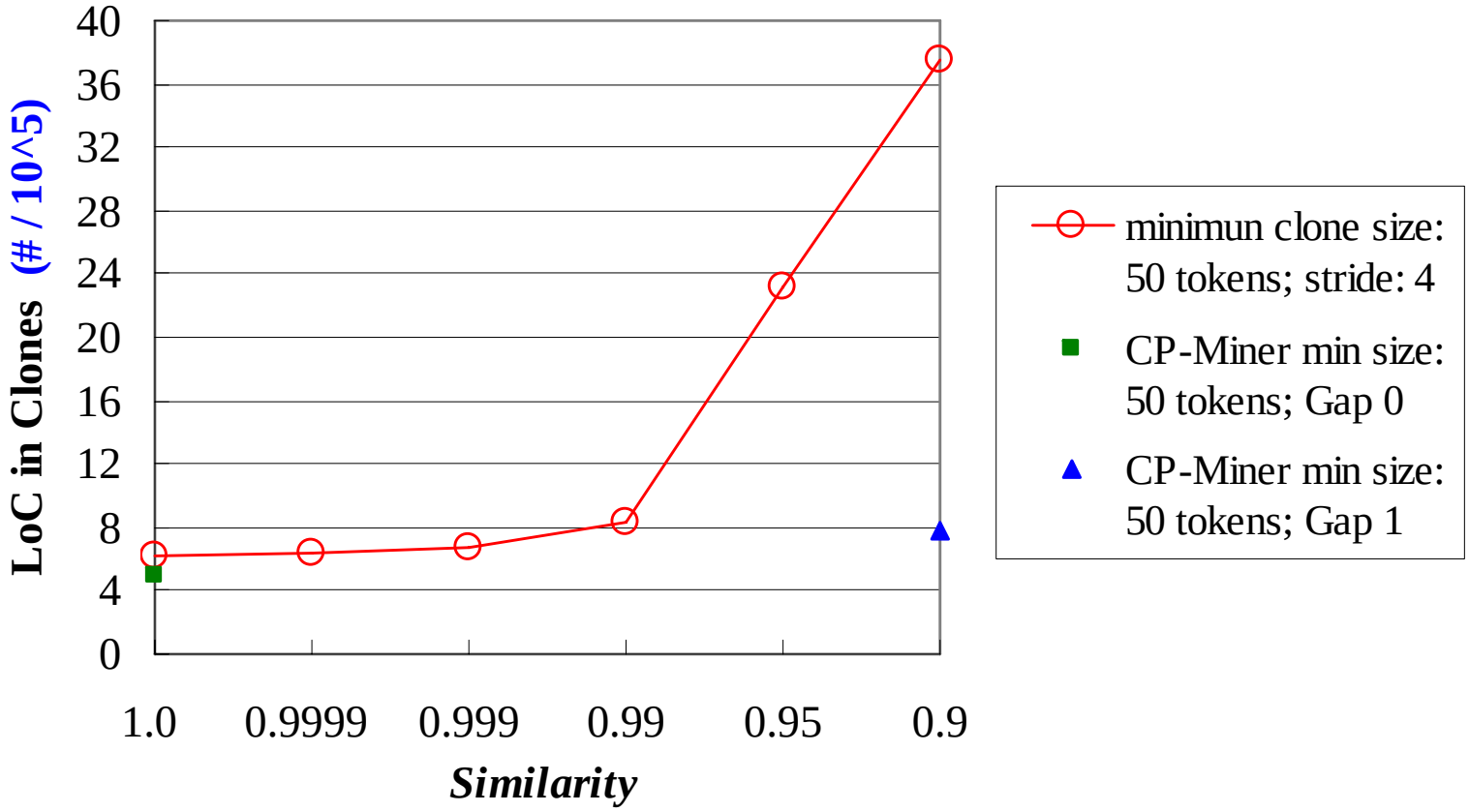
JDK 1.4.2 (Deckard vs. CloneDR)



Detected More Clones (2)

Linux kernel 2.6.16 (Deckard vs. CP-Miner)

Gap: # of statement insertions, deletions, and modifications



Clone Quality (1)

- Examined 100 clone clusters
  - Randomly selected from JDK 1.4.2
  - Used parameters
    - Similarity: 1.0
    - Minimum clone size in tokens: 50
    - Stride: 4
- Among the 100 clusters
  - 93 are clearly real clones
  - The remaining 7 are difficult to assess
    - But they are structurally similar

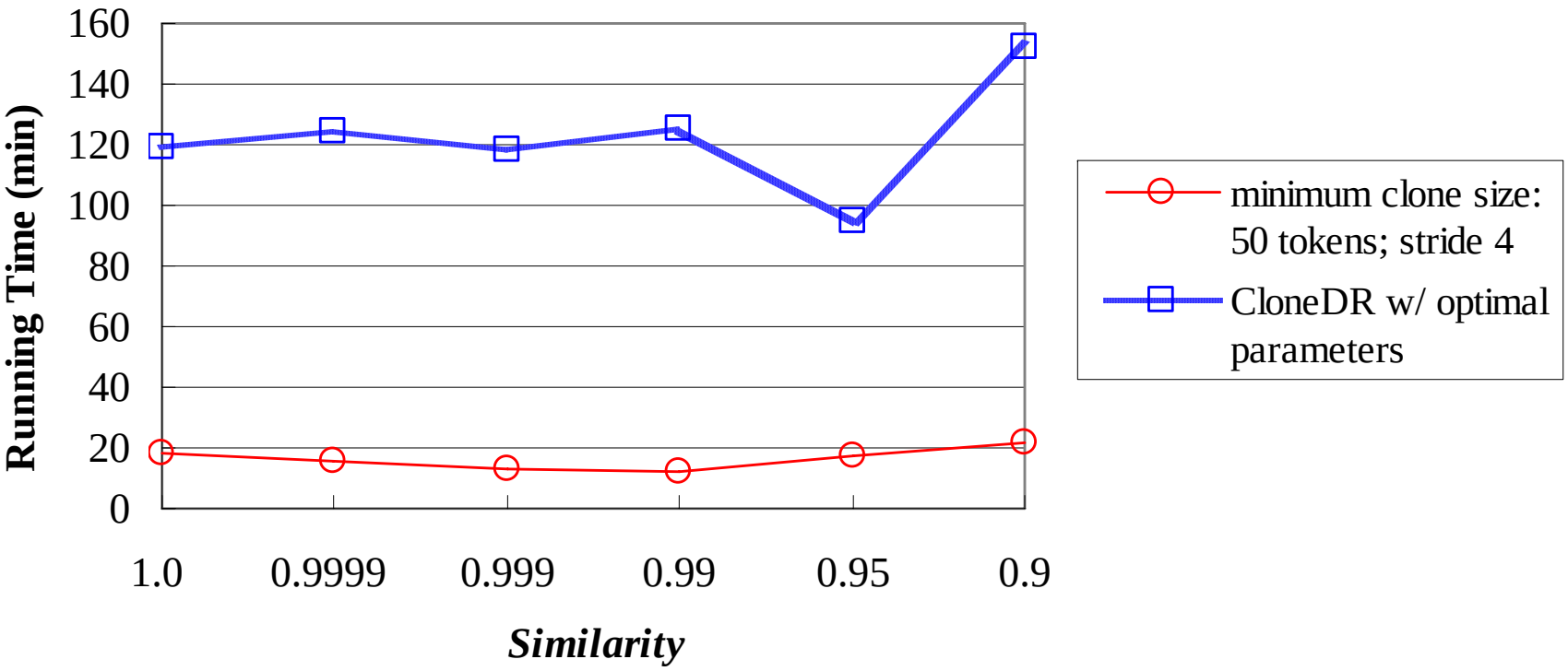
Clone Quality (2)

```
if ......
......
else if (option.equalsIgnoreCase(``basic'')) {
        bBasicTraceOn = true;
} else if (option.equalsIgnoreCase(``net'')) {
        bNetTraceOn = true;
} else if (option.equalsIgnoreCase(``security'')) {
        bSecurityTraceOn = true;
} else ......
......
```

```
if ......
......
else if (opt.equals(``-nohelp'')) {
      nohelp = true;
} else if (opt.equals(``-splitindex'')) {
      splitindex = true;
} else if (opt.equals(``-noindex'')) {
      createindex = false;
} else ......
......
```

Scalability (1)

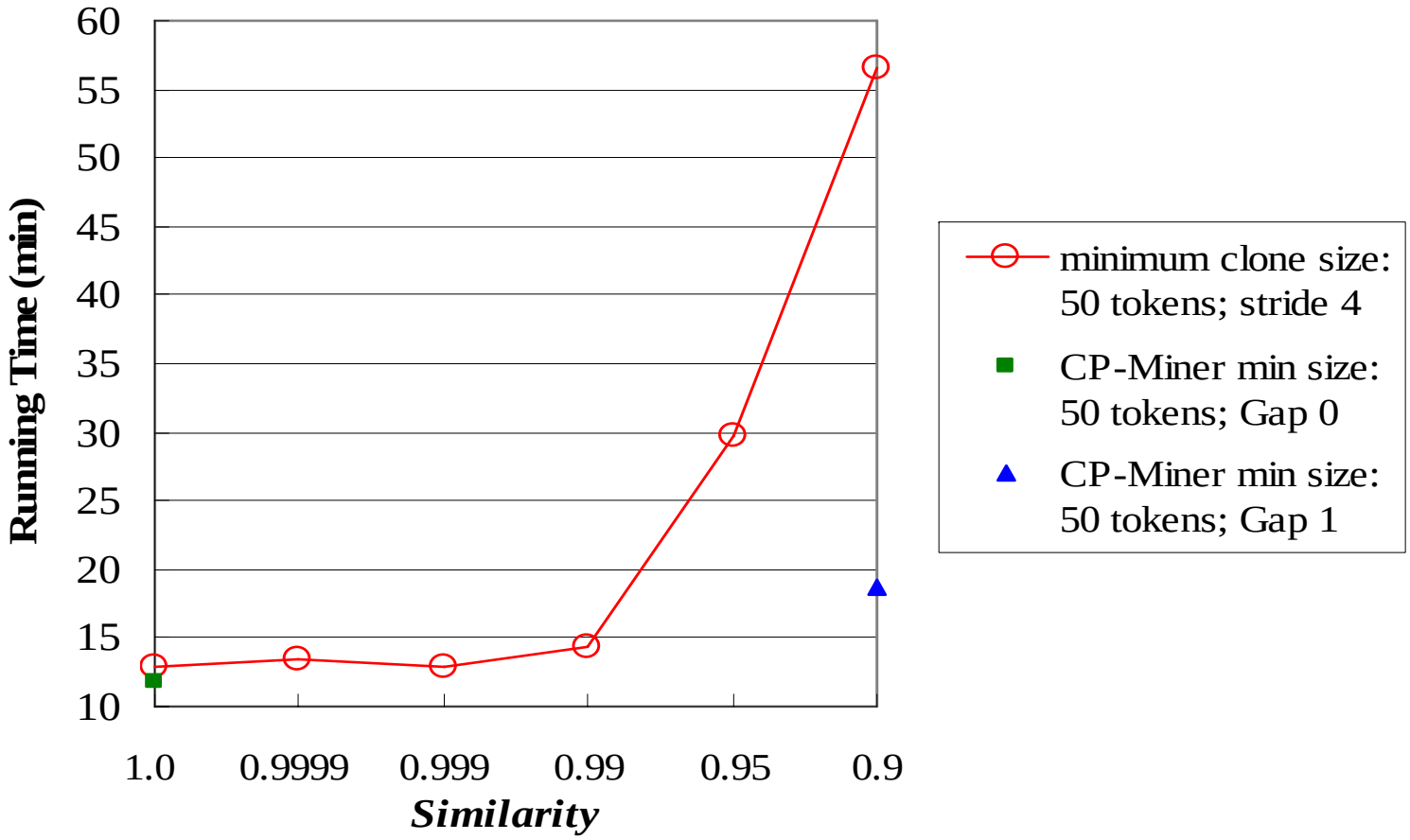
JDK 1.4.2 (Deckard vs. CloneDR)



Scalability (2)

Linux kernel 2.6.16 (Deckard vs. CP-Miner)

Gap: # of statement insertions, deletions, and modifications



Conclusion

- New algorithm for tree similarity checking
  - Map trees to numerical vectors
  - Cluster similar vectors efficiently
- Application of the algorithm on clone detection
  - Similarity checking on parse trees
  - Promising results: scalable and accurate

Thank you!

Questions?

jiangl@cs.ucdavis.edu