

# Instant Code Clone Detection

---

Mu-Woong Lee, Jong-Won Roh, Seung-won Hwang  
POSTECH

Sunghun Kim  
HKUST

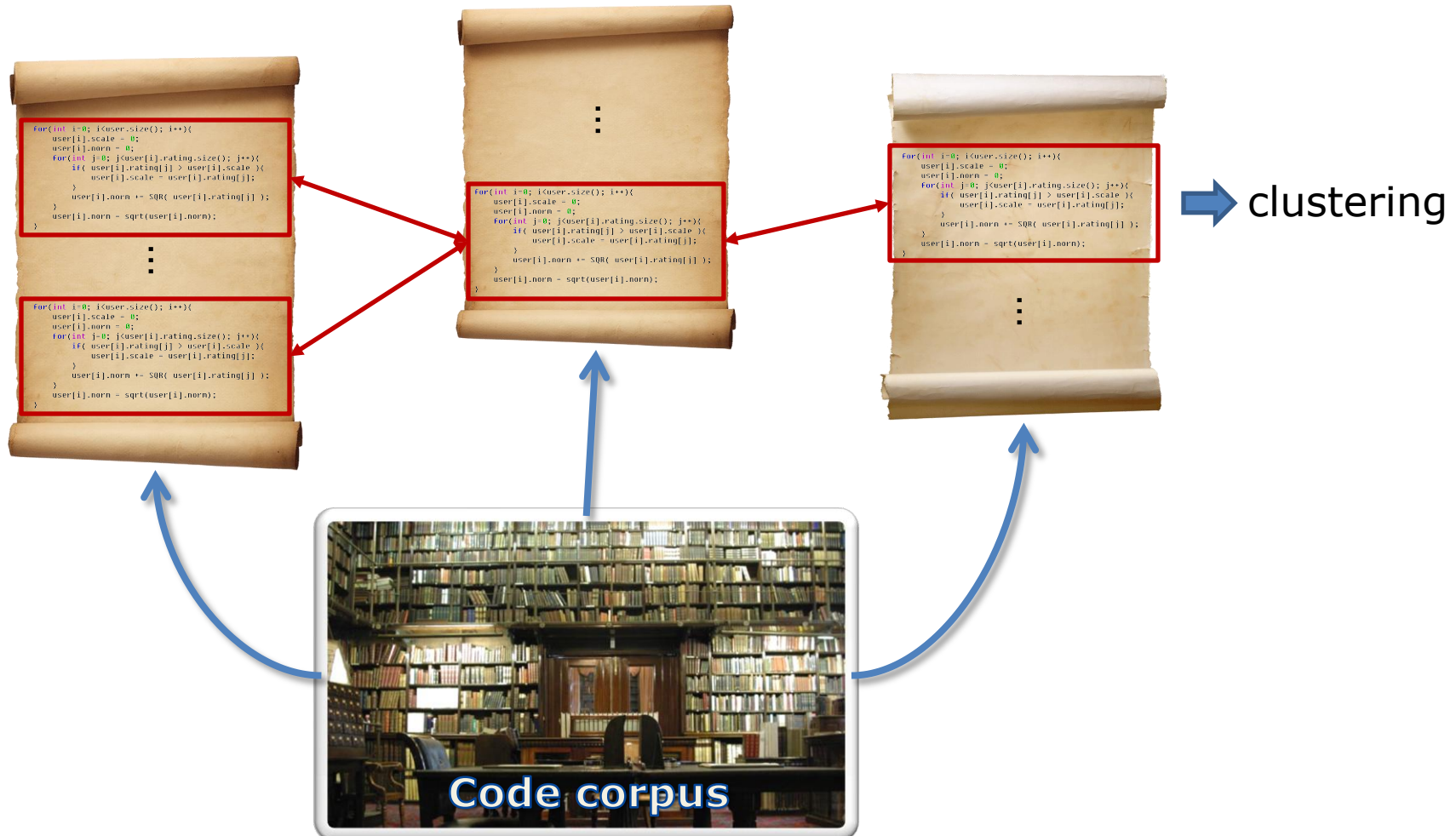
# Outline

---



- Code Clone Detection
- Motivation
  - Our Goal
- System Overview
- Characteristic Vectors
- Dimensionality Reduction
- Indexing
- Filtering-then-Ranking Clone Detection
- Interleaved Clone Detection
- Experimental Evaluation

# Code Clone Detection



# Code Clone Detection

---

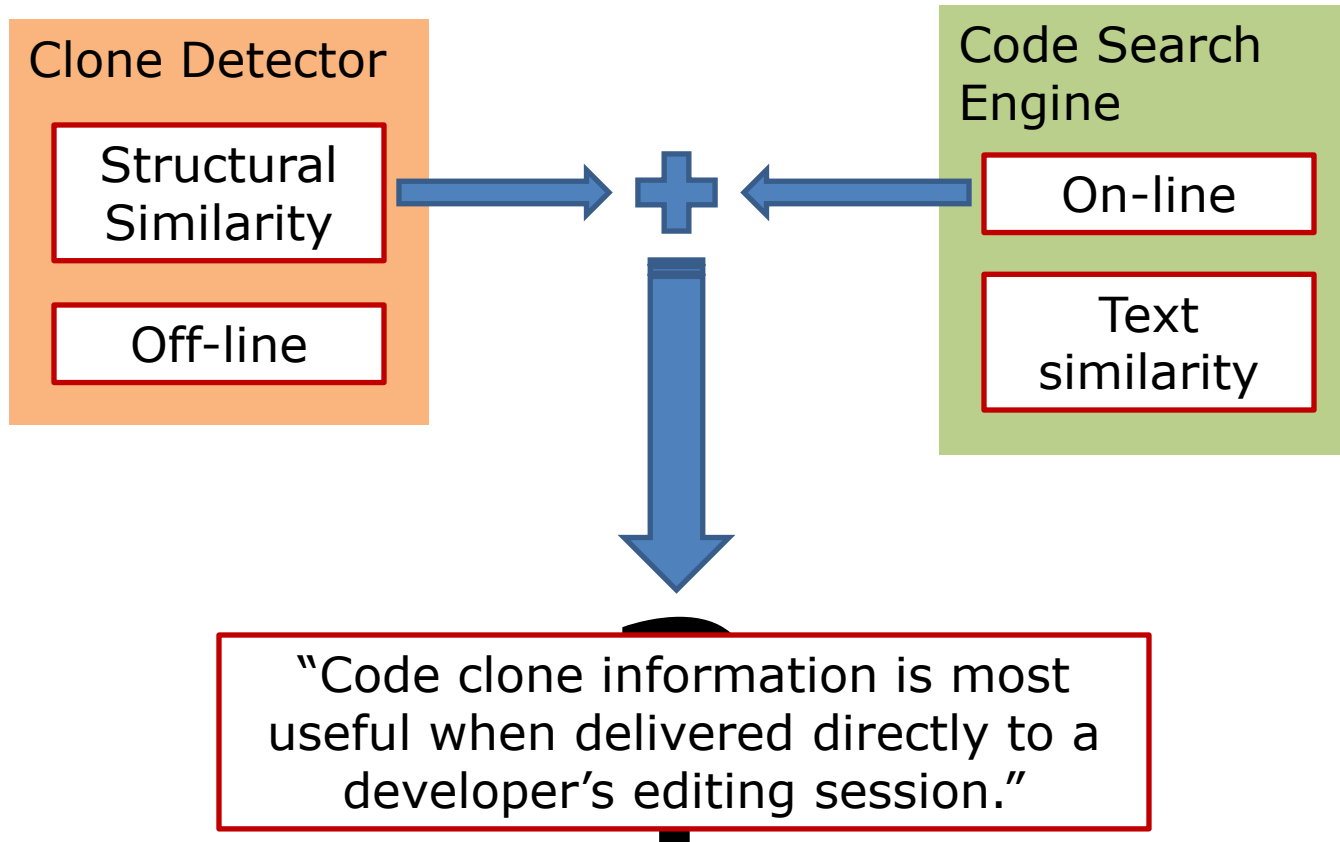


- Similarity measure:
  - Textual similarities using tokens
  - Abstract Syntax Tree (AST)
  - Program Dependence Graph (PDG)
  
- Application:
  - Code refactoring
  - Cheating detection...
  
- Most clone detectors work as an “off-line” manner...

# Motivation



- Why it should be an off-line process?



# Our Goal



- Find clones of a given code ASAP!

```
for(int i=0; i<user.size(); i++){
    user[i].scale = 0;
    user[i].norm = 0;
    for(int j=0; j<user[i].rating.size(); j++){
        if (user[i].rating[j] > user[i].scale){
            user[i].scale = user[i].rating[j];
        }
        user[i].norm += SQR( user[i].rating[j] );
    }
    user[i].norm = sqrt(user[i].norm);
}
```

Query code fragment

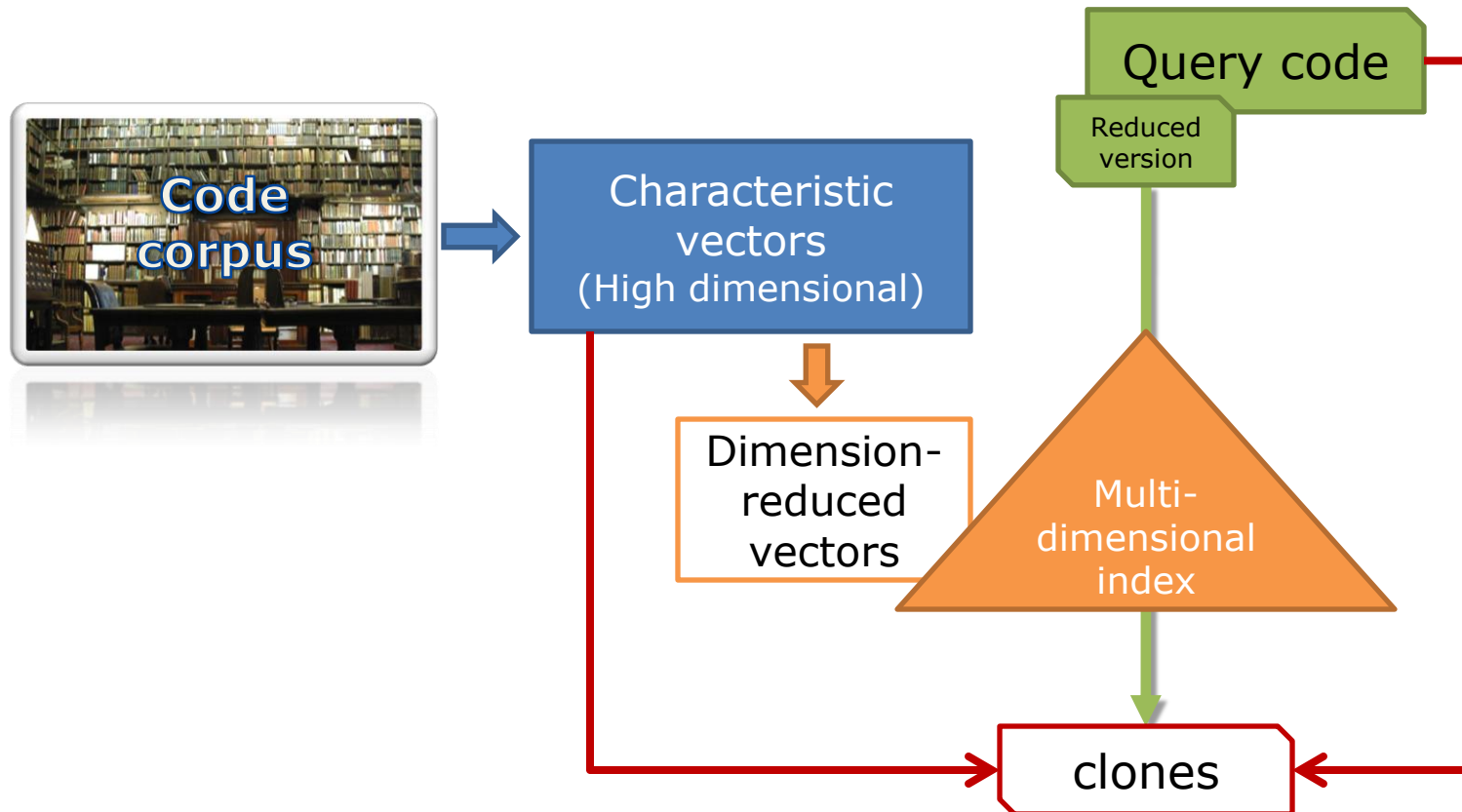


```
for(int i=0; i<user.size(); i++){
    user[i].scale = 0;
    user[i].norm = 0;
    for(int j=0; j<user[i].rating.size(); j++){
        if (user[i].rating[j] > user[i].scale){
            user[i].scale = user[i].rating[j];
        }
        user[i].norm += SQR( user[i].rating[j] );
    }
    user[i].norm = sqrt(user[i].norm);
}
```

Top-*k* clones



# System Overview

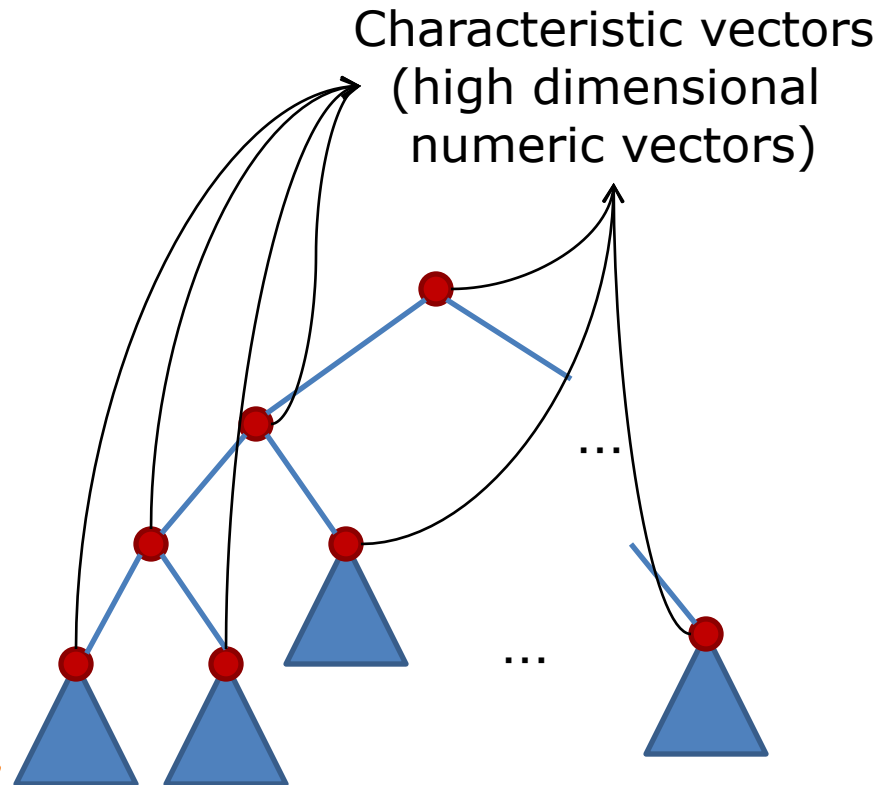


# Characteristic Vectors



□ Tree similarity  $\rightarrow$  vector similarity

```
darkkhaki - Xshell 3.0
파일(F) 편집(E) 보기(V) 도구(T) 창(W) 도움말(H) | 콘솔 | F10:Sys | 12
| darkkhaki
48 public final class HomepageRedirector extends Controller {
49
50     /**
51     * logger.
52     */
53     private static final Logger LOGGER = Logger
54         .getLogger(HomepageRedirector.class);
55
56     /**
57     * redirects to the first visible <code>SitemapNode</code> in th
58     * <code>Locale</code>.
59     *
60     * @param bundle
61     *     the bundle
62     * @param request
63     *     the current request
64     * @return an <code>EmptyResult</code>
65     */
66     @Action(value = "homepage", generate = true)
67     @Permission("user")
68     public Result goToHomepage(final Bundle bundle, final ServiceReq
69         Locale locale;
70         if (request.getCommand().getParameter("country") != null) {
71             locale = findLocale(request.getCommand().getParameter("c
72                 .getFirstValue().split(" "), request);
73             redirect(bundle, request, locale);
74             return new EmptyResult();
75         }
76
77         locale = request.getLocale();
78         if (locale != null) {
79             redirect(bundle, request, locale);
80         }
81         return new EmptyResult();
82     }
83
84     /**
```



Each sub-tree  
containing at  
least  $T$  nodes

AST



# Dimensionality Reduction

---



- Why we reduce dimensionality of the characteristic vectors?
  - **Curse of dimensionality!**
  - Characteristic vectors → 261D integer vectors
  - On high dimensional data
    - Index search time > sequential search time

# Dimensionality Reduction



## □ Optimal subspace selection

- Preserve the lower-bounding property

$$\|v'_i, v'_j\| \leq \|v_i, v_k\|$$

- Preserve the distance relations between two vectors as much as possible

$$\Delta = \sum_{\forall i, \forall j, i \neq j} \delta_{i,j} = \sum_{\forall i, \forall j, i \neq j} \|v_i, v_j\| - \|v'_i, v'_j\|$$

- NP-Hard

# Dimensionality Reduction



- Greedy vs. Variance-based
  - Top-10 selected dimensions:

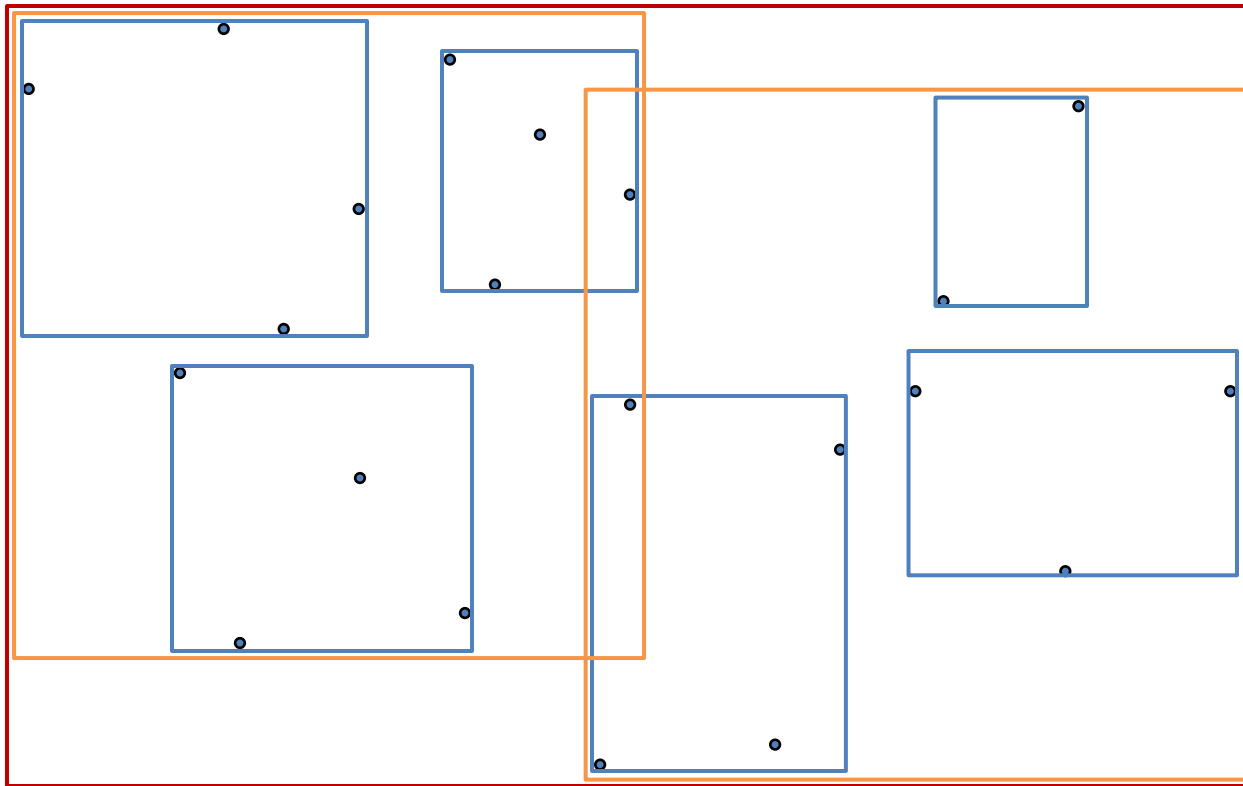
	Greedy strategy	Variance-based
1	Identifier	Identifier
2	ID_TK	ID_TK
3	Unary expression	Unary expression
4	Multiplicative expression	Multiplicative expression
5	Additive expression	Additive expression
6	Shift expression	Relational expression
7	Relational expression	Shift expression
8	Equality expression	Equality expression
9	Conditional expression	Conditional expression
10	Assignment expression	Assignment expression

# Indexing



## □ R-tree

- 2D example (node capacity: 4)

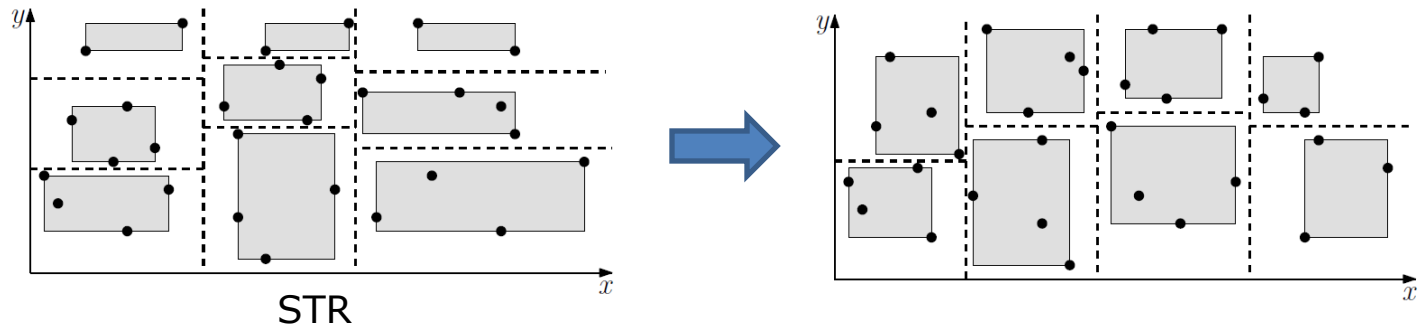


# Indexing



## □ “bulkloading”

- The variance of each dimension varies a lot

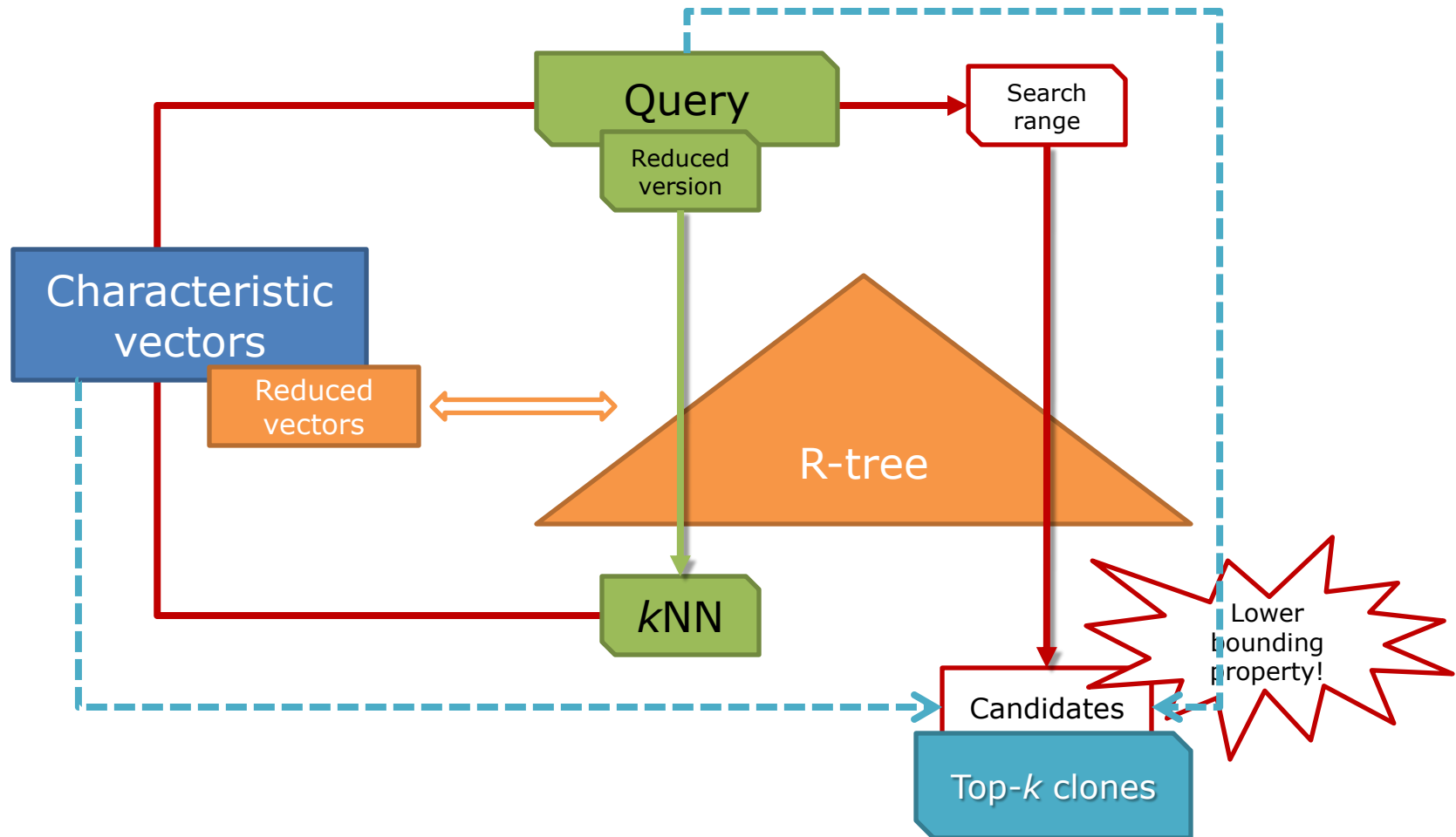


- Still high dimensional (20D?)
  - Decrease the number of slice for each dimension
- Bottom-up index building

# Filtering-then-Ranking Clone Detection



- Top-k code clones?



# Filtering-then-Ranking Clone Detection

---



## □ Performance

- Faster than the sequential scan (7~10 times)

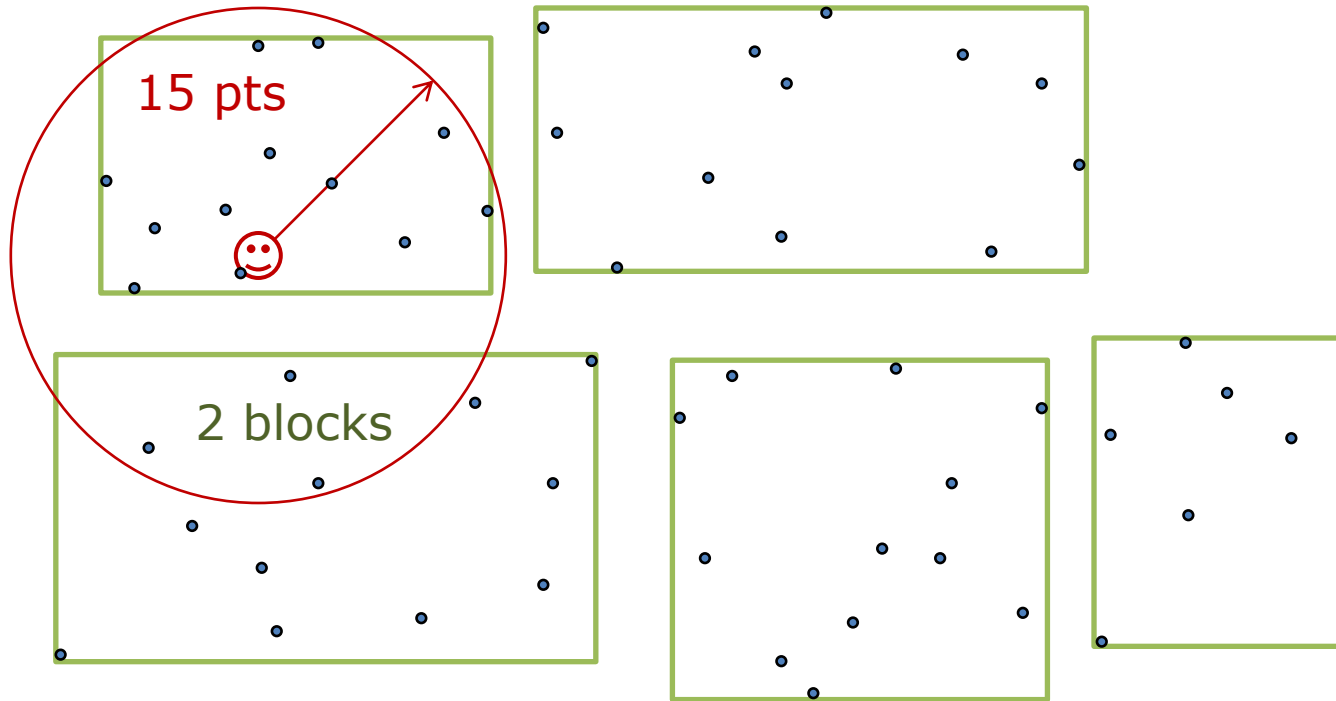
## □ Issue

- Expensive I/Os
  - Random I/Os for traversing the tree
  - Random I/Os for reading original characteristic vectors

# Interleaved Clone Detection



## □ Vector packing



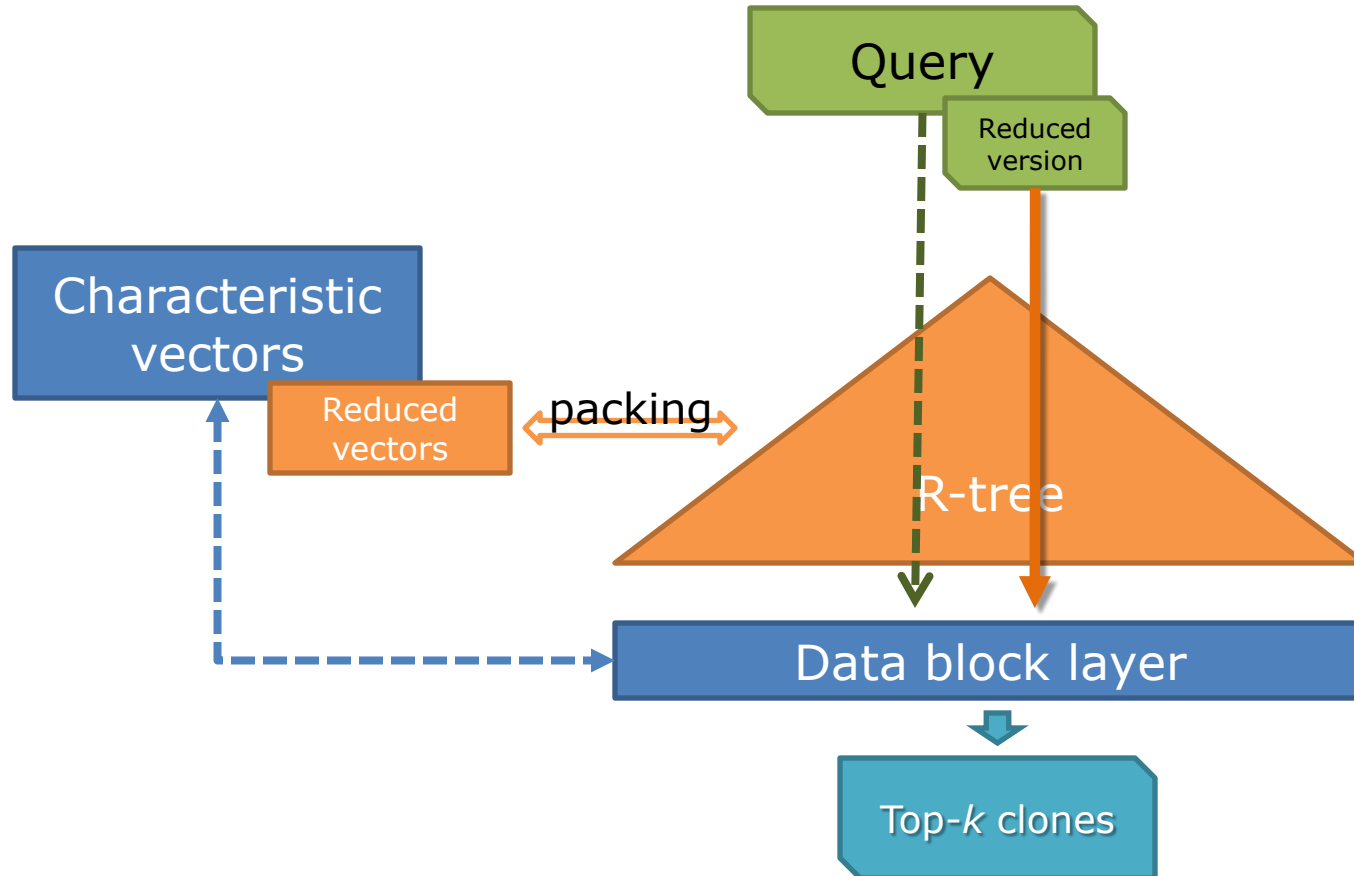
- Many random I/Os → few random I/Os + scan



# Interleaved Clone Detection



## □ Interleaved index traversal



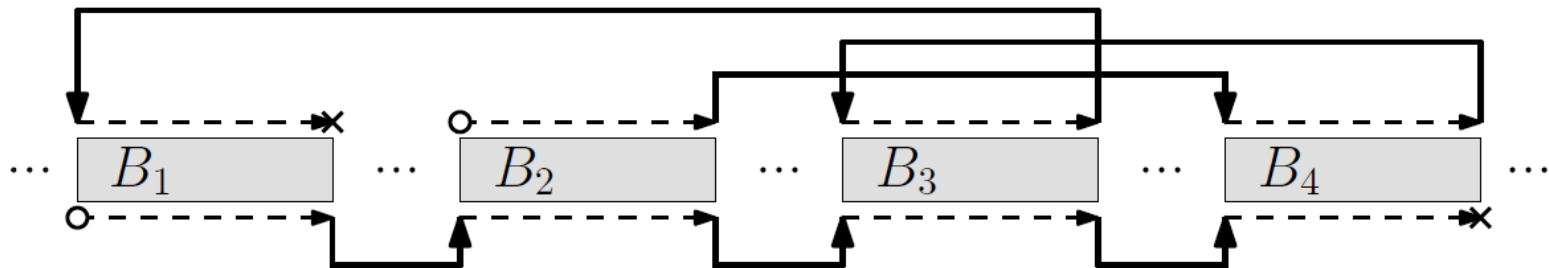
# Interleaved Clone Detection



## □ Delayed loading

- “Circular SCAN disk scheduling”
- Eg) loading the blocks  $2 \rightarrow 4 \rightarrow 3 \rightarrow 1$ :

Without delayed loading



With delayed loading

--> Sequential access  
—> Random access

- Four random access  $\rightarrow$  one random + sequential access

# Experimental Evaluation

---



## □ Environment

- Pentium IV 3.2 GHz CPU
- 1GB memory
- P-ATA HDD
- Linux, gcc

## □ Dataset

- JDK 1.6.0 update 13
  - 7,195 java files, 2,075,573 LOC
- 400 Java open source projects
  - Hosted on SourceForge, Tigris.org, and GoogleCode
  - 288,846 java files, 54,709,384 LOC
- DECKARD characteristic vector generator

# Experimental Evaluation



## □ Index building time

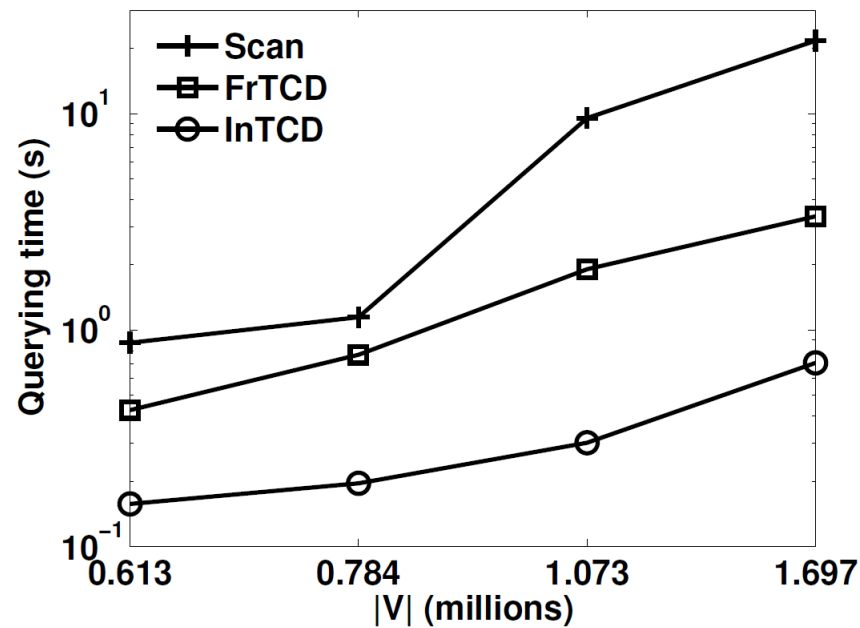
Dataset	minT	# vectors	Building time (s)	
			Filtering-then-ranking	Interleaved
JDK <sub>5</sub>	50	36,658	0.563	0.867
JDK <sub>3</sub>	30	60,582	0.793	1.517
OSP <sub>9</sub>	90	612,926	8.968	34.055
OSP <sub>7</sub>	70	783,933	11.619	46.725
OSP <sub>5</sub>	50	1,072,598	16.939	72.903
OSP <sub>3</sub>	30	1,696,806	27.653	128.118

# Experimental Evaluation



## □ Querying time over varying # of vectors

■  $k = 20$

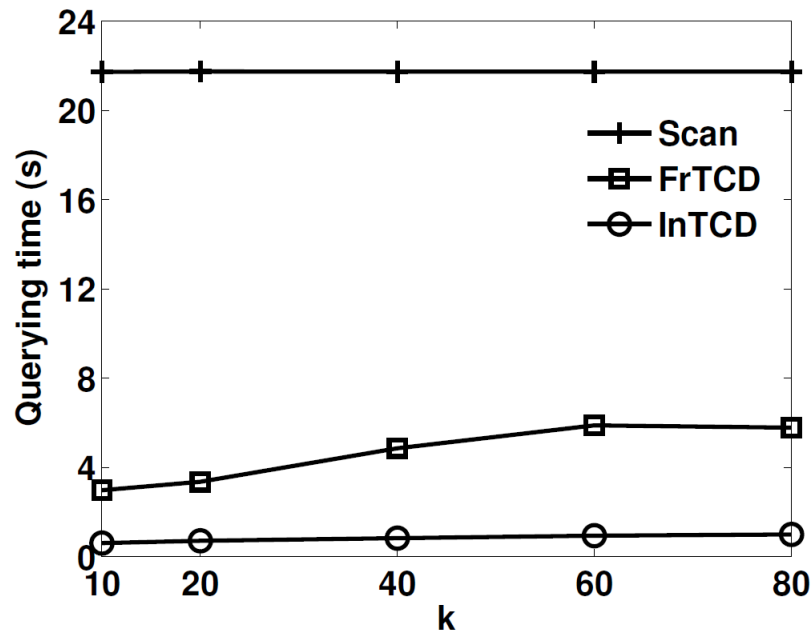


# vectors (millions)	# results
0.613	27.26
0.784	40.55
1.073	56.12
1.697	73.23

# Experimental Evaluation



- Querying time over varying  $k$ 
  - # of vectors = 1,697K



$k$	# results
10	63.60
20	73.23
40	94.92
60	113.78
80	135.93

# Conclusion

---



- Our proposed algorithm
  - Detects clones among 1.7 million code fragments in sub-second response time
  - Supports top- $k$  queries
- We also proposed an approximation algorithm
  - Dozens times faster / 70% accurate
- To do
  - Comparisons with the state-of-the-art tools

# Q&A

---

