Instant Code Clone Detection

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

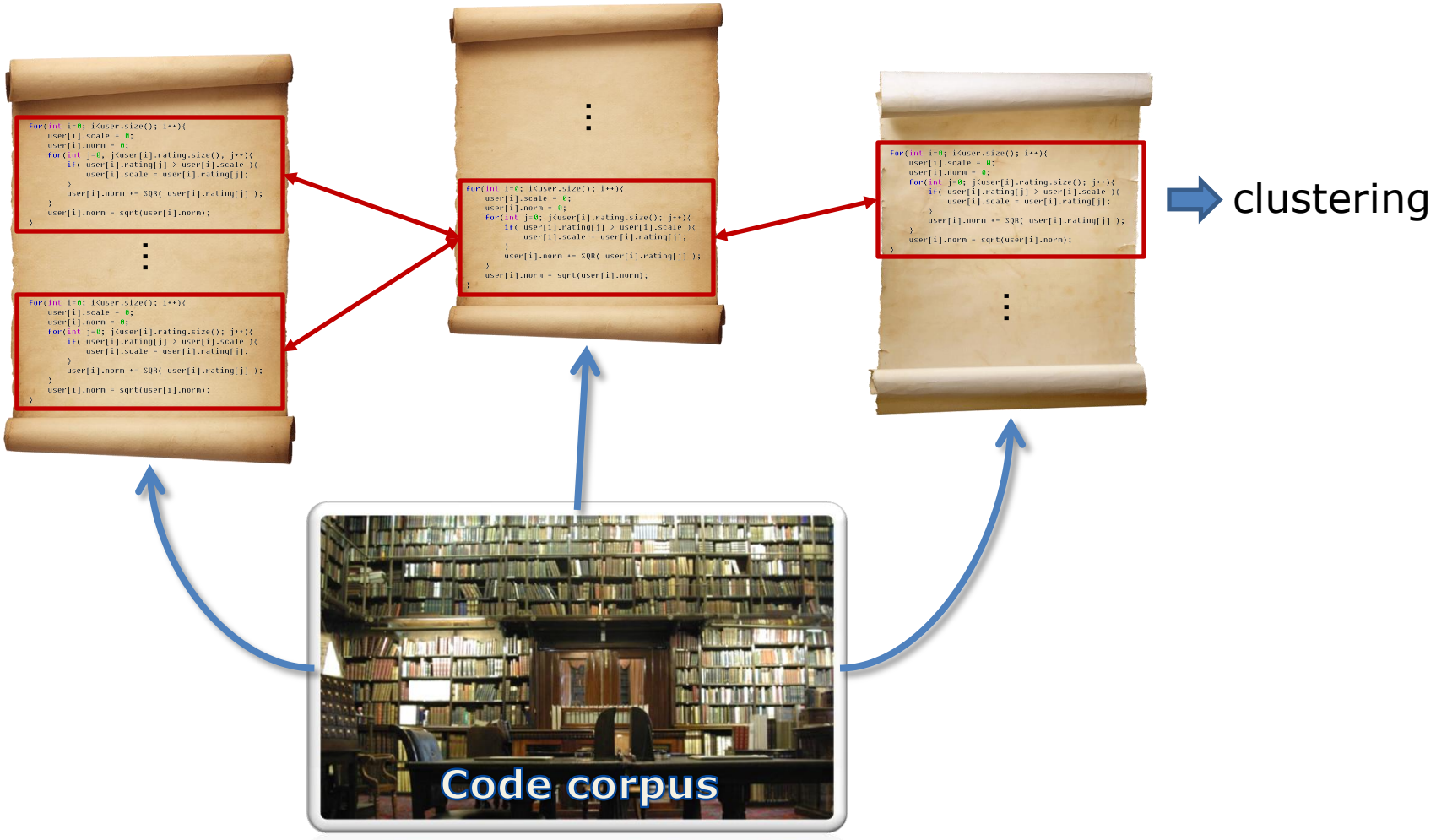
Sunghun Kim
HKUST

POSTECH

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

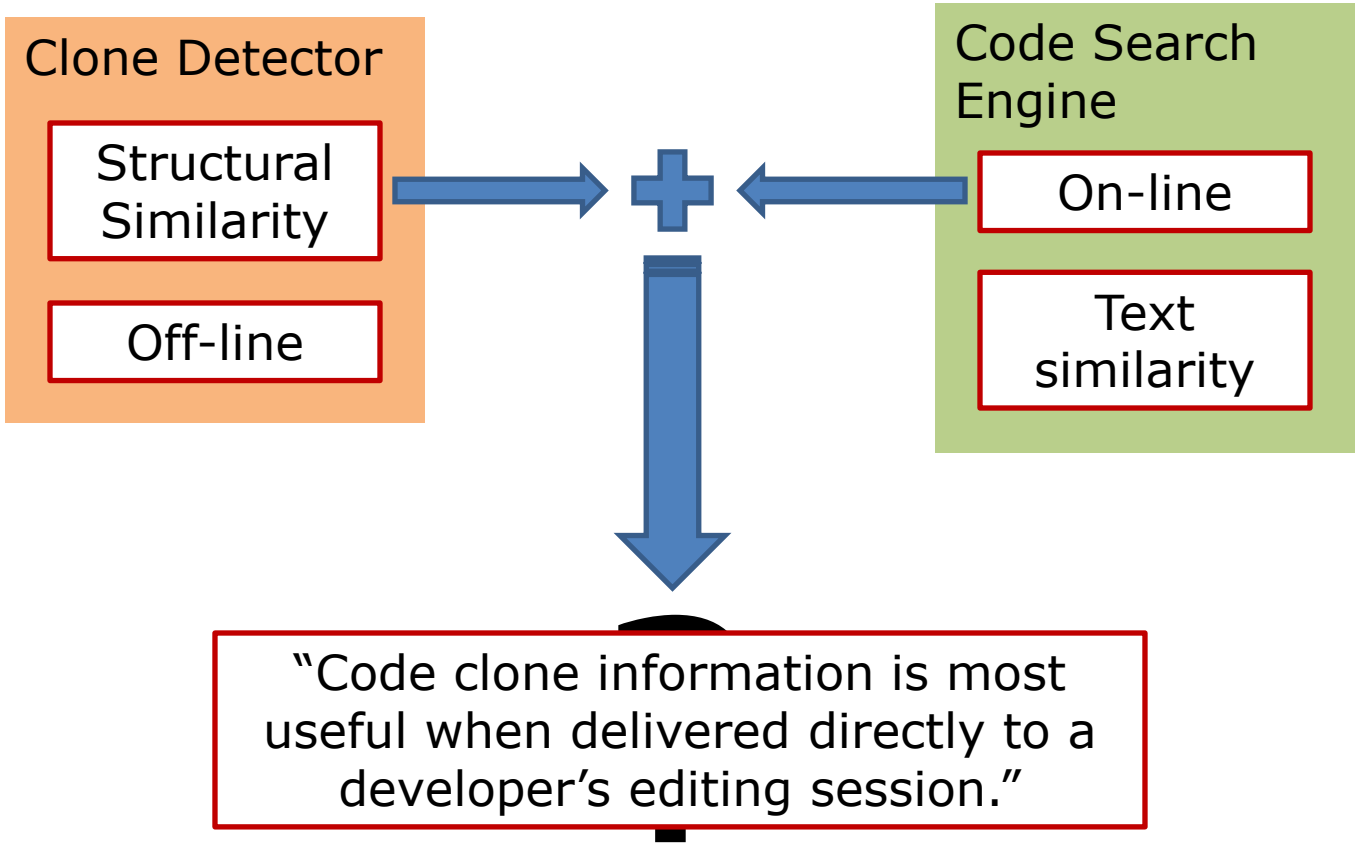


Code Clone Detection

- 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…

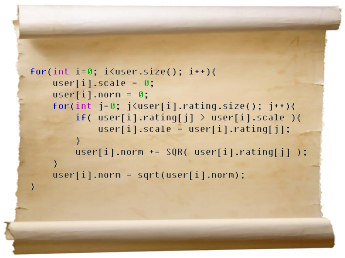
Motivation

- Why it should be an off-line process?

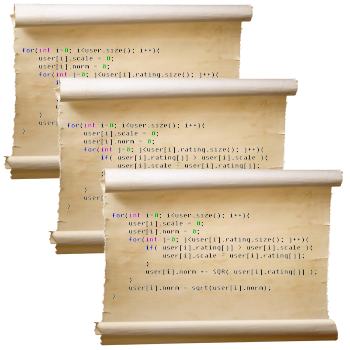


Our Goal

- Find clones of a given code ASAP!

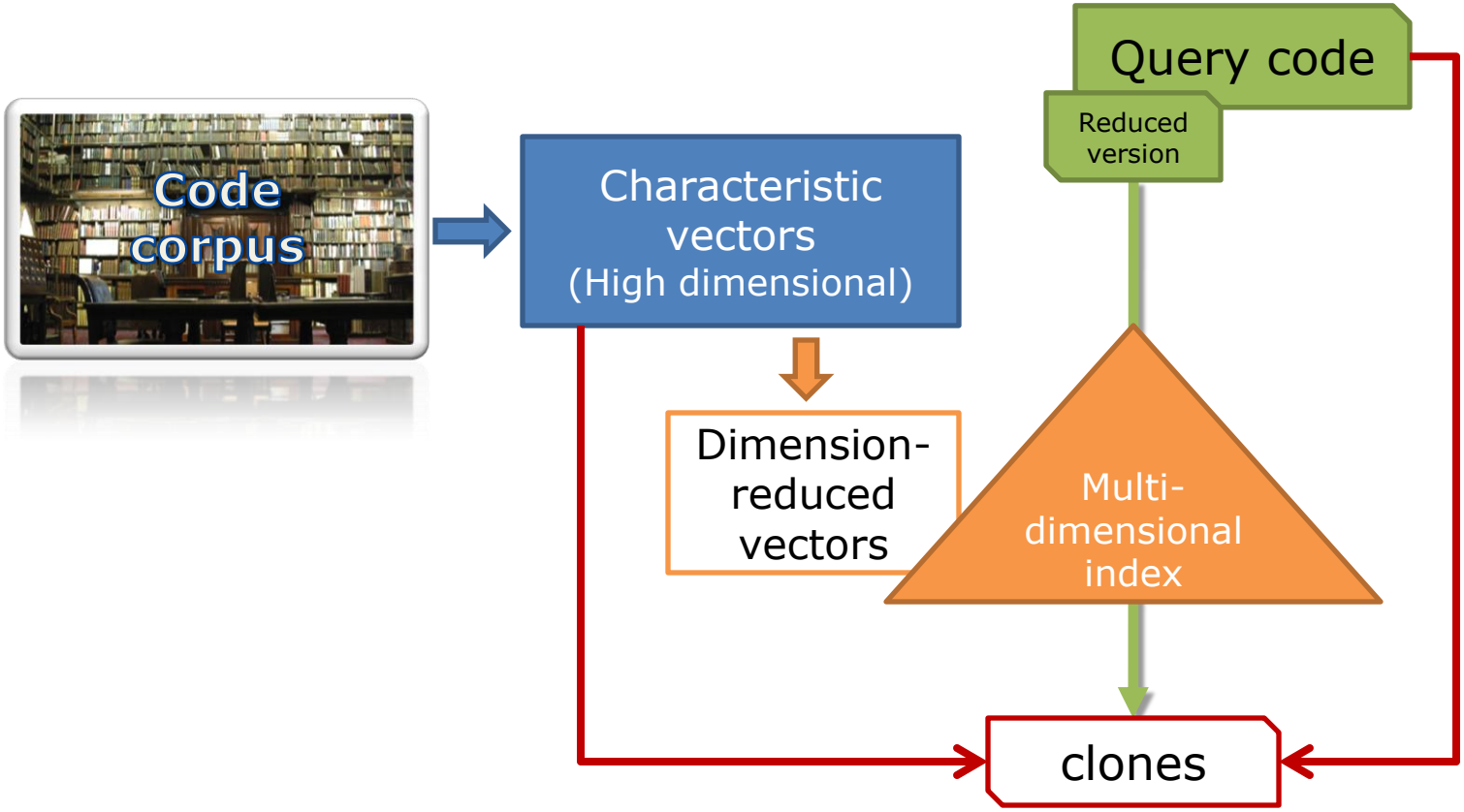


Query code fragment



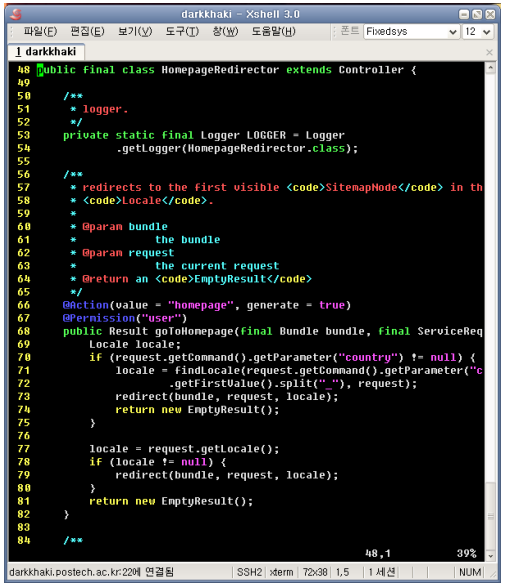
Top-*k* clones

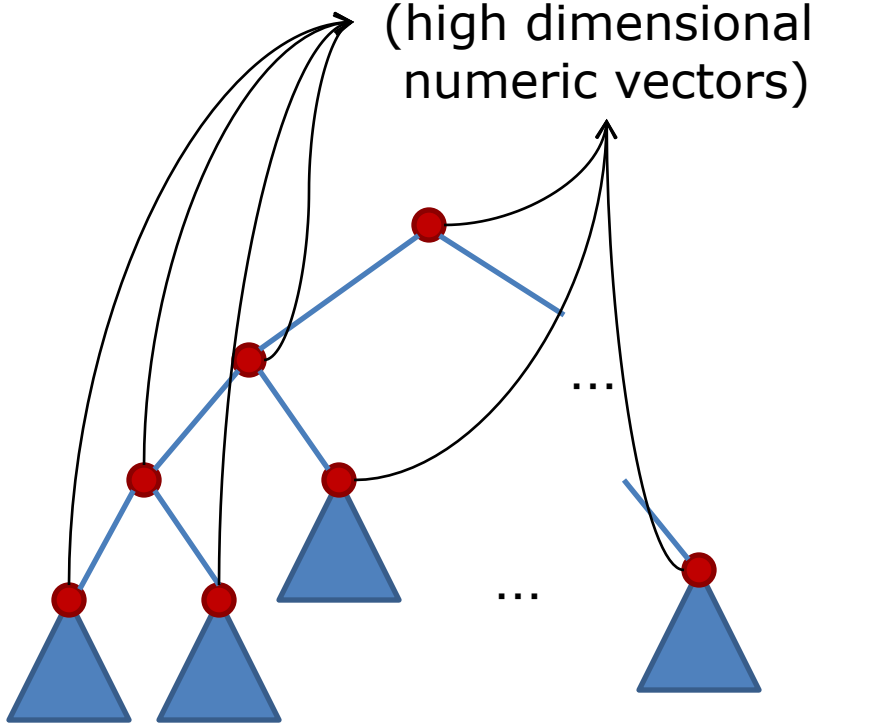
System Overview



Characteristic Vectors

- Tree similarity → vector similarity





Each sub-tree containing at least *T* nodes

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

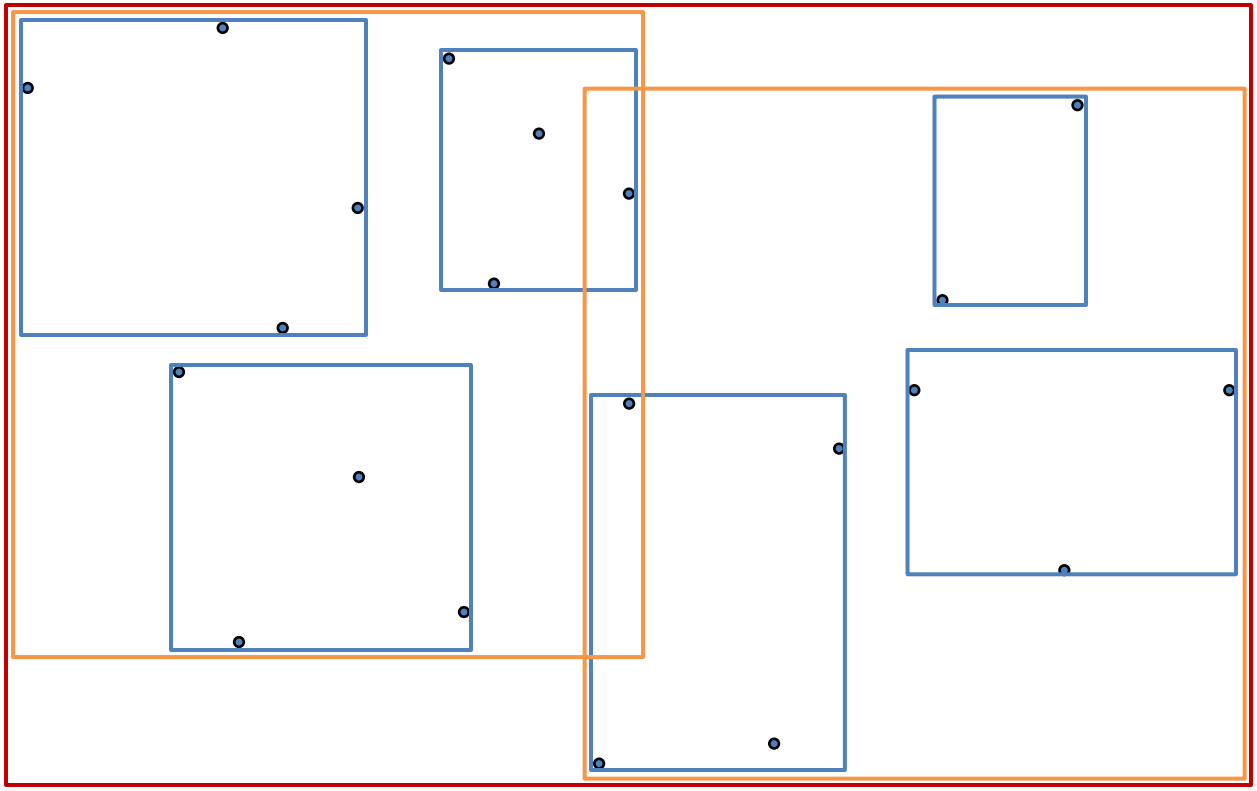
Dimensionality Reduction

- 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

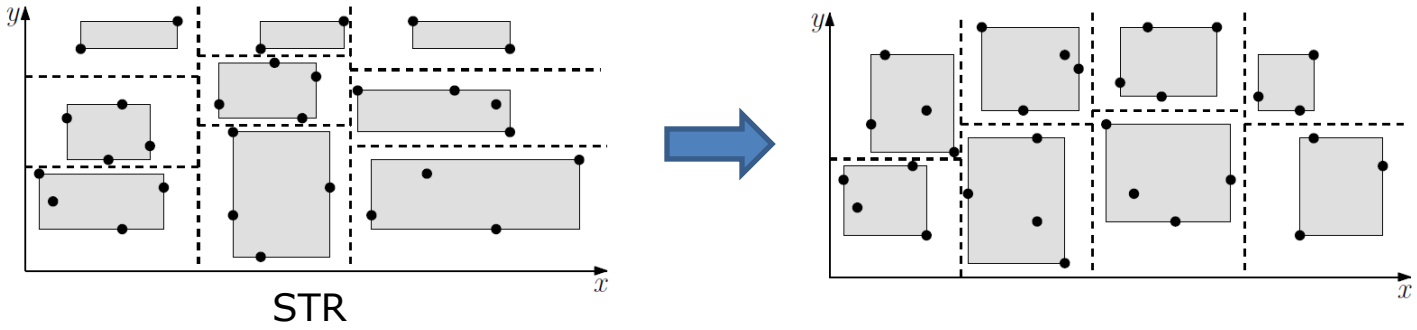
Indexing

- R-tree
  - 2D example (node capacity: 4)



Indexing

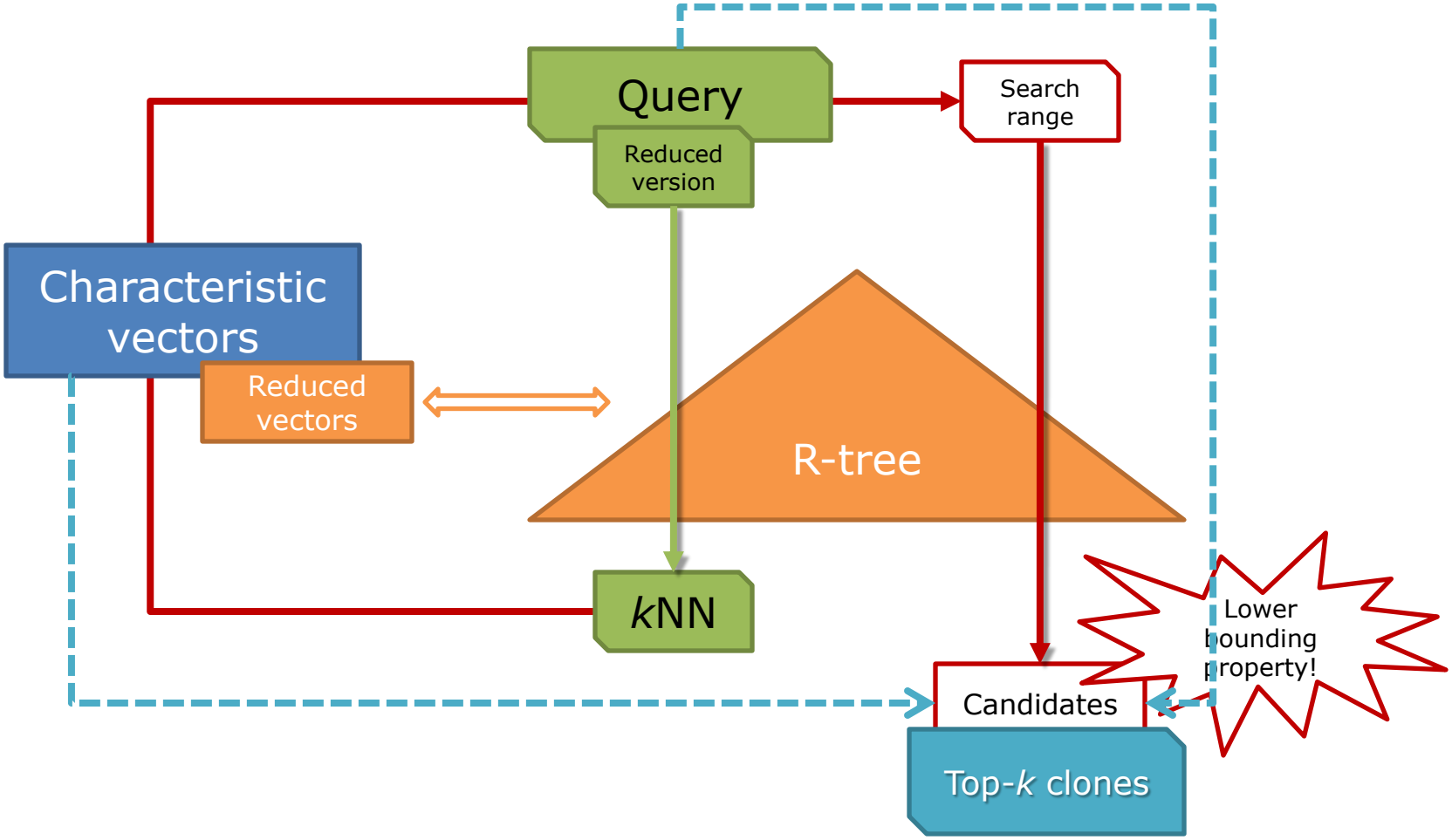
- "bulkloading"
  - The variance of each dimension varies a lot



  - 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?



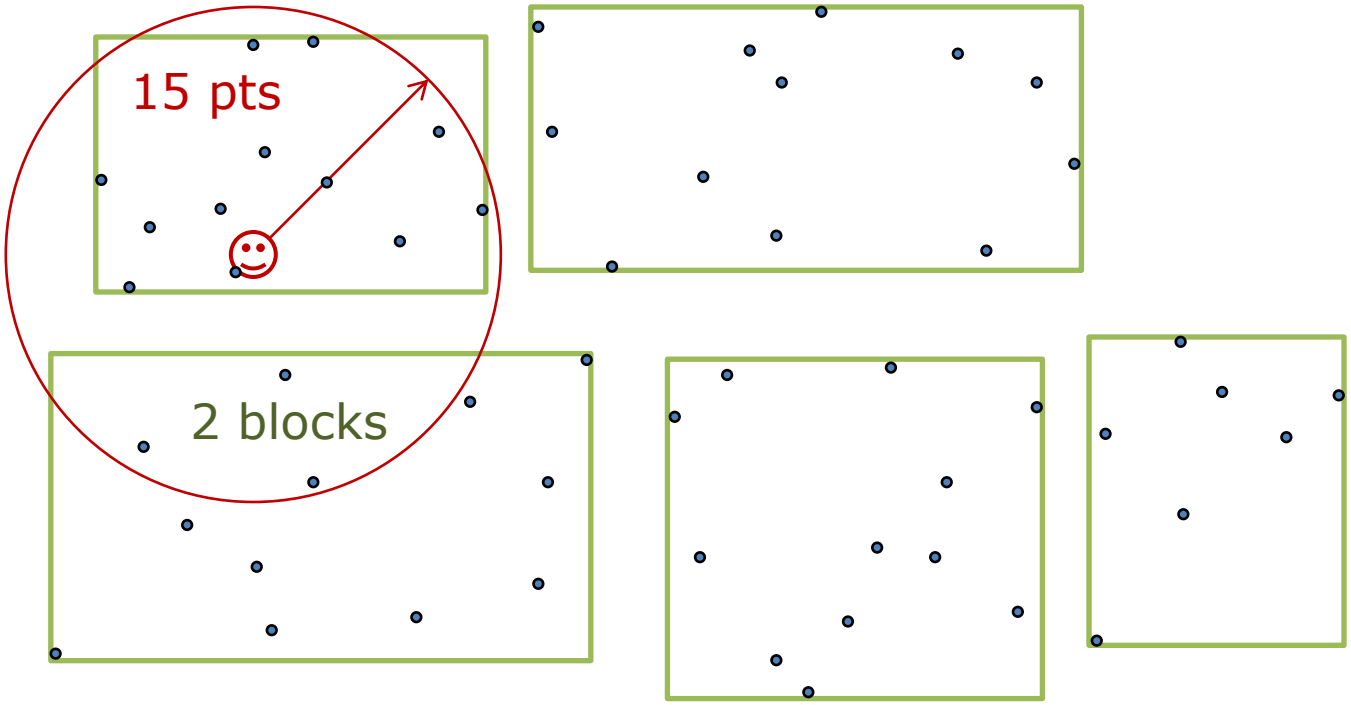
POSTECH

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

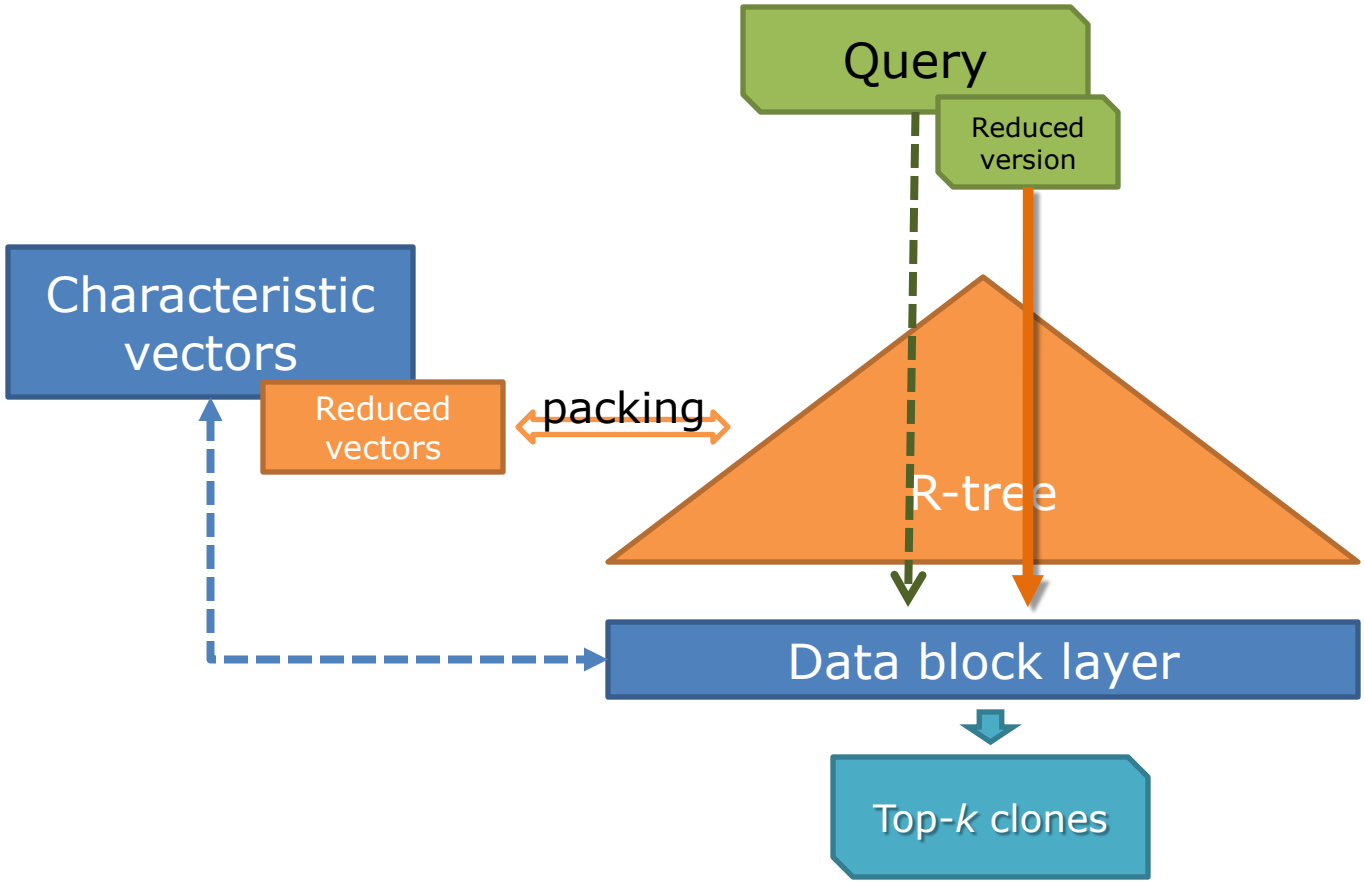
- Vector packing



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

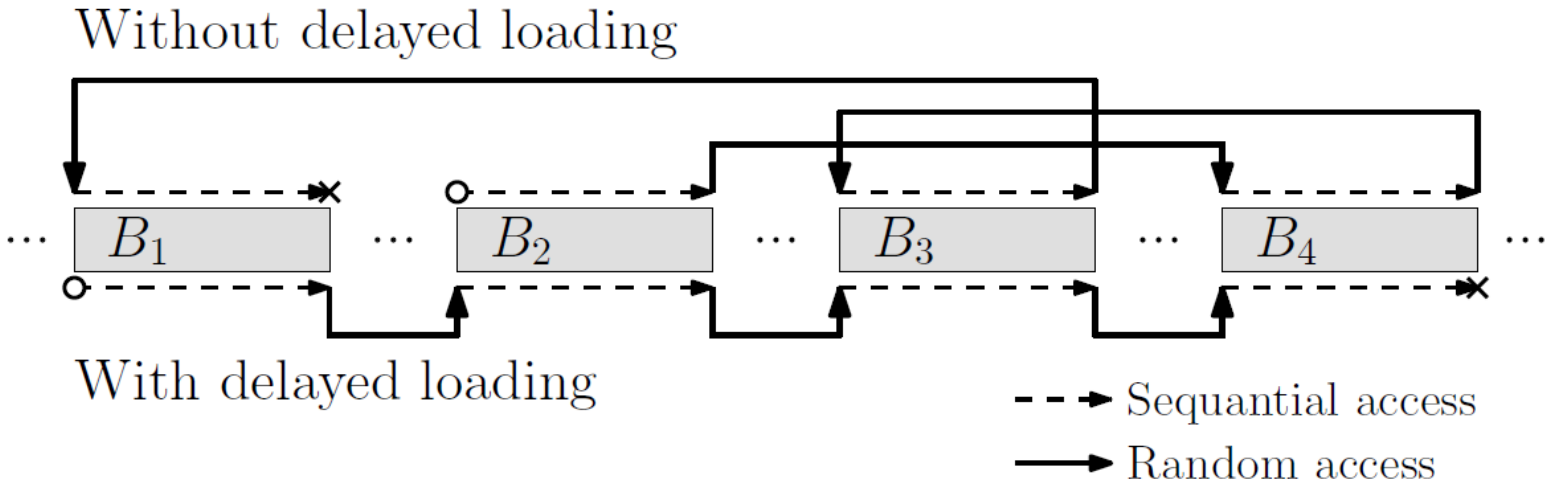
Interleaved Clone Detection

- Interleaved index traversal



Interleaved Clone Detection

- Delayed loading
  - "Circular SCAN disk scheduling"
  - Eg) loading the blocks 2 → 4 → 3 → 1:



  - Four random access → 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

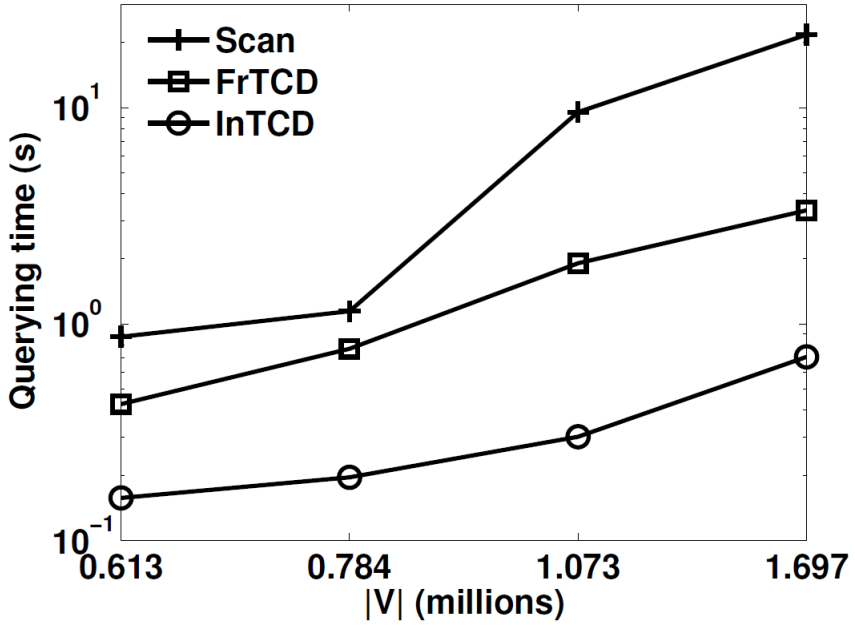
Experimental Evaluation

- Index building time

Dataset	minT	# vectors	Building time (s)	
			Filtering-then-ranking	Interleaved
JDK_5	50	36,658	0.563	0.867
JDK_3	30	60,582	0.793	1.517
OSP_9	90	612,926	8.968	34.055
OSP_7	70	783,933	11.619	46.725
OSP_5	50	1,072,598	16.939	72.903
OSP_3	30	1,696,806	27.653	128.118

Experimental Evaluation

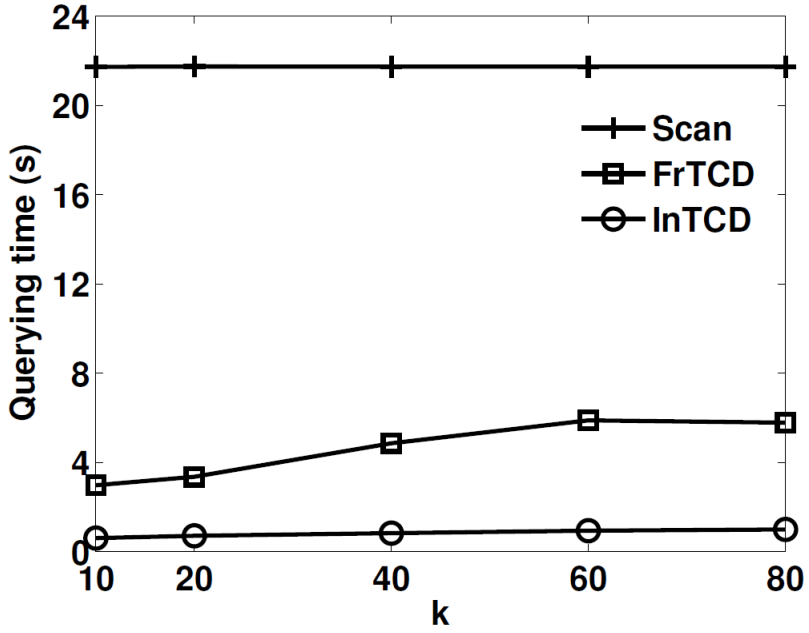
- Querying time over varying # of vectors
  - k = 20



# 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