MOBS:
Multi-Operator Observation-Based Slicing
using Lexical Approximation of Program Dependence

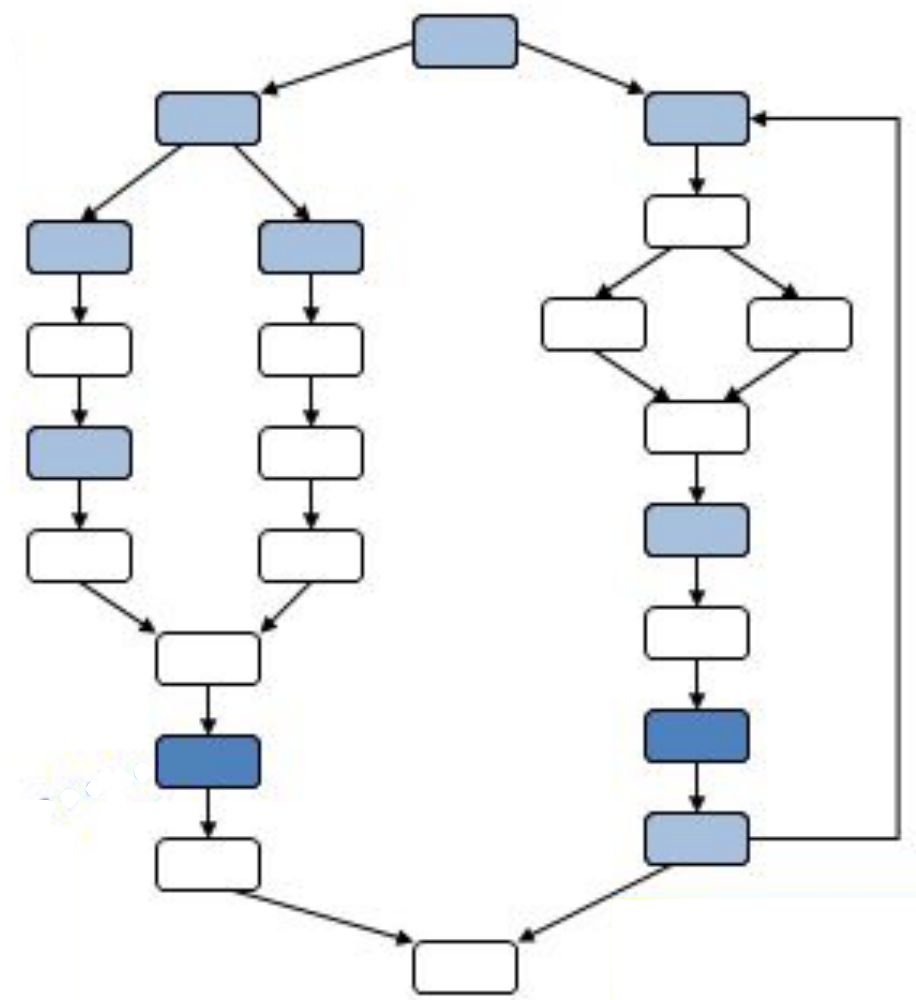
Seongmin Lee

COINSE, KAIST

Program Slicing

- Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.

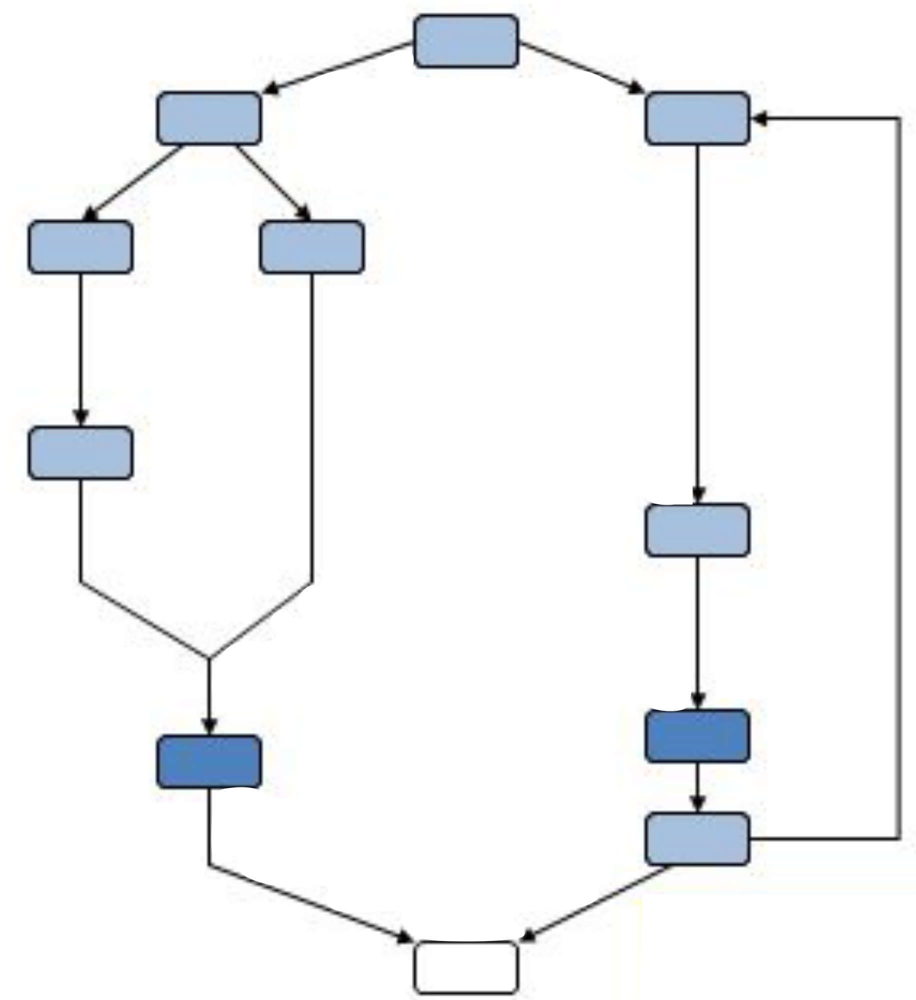
- Specific behavior: Slicing Criterion < i , V >
(i : line number, V : variable name)



Program Slicing

- Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.

- Specific behavior: Slicing Criterion $< i , V >$ (i : line number, V : variable name)

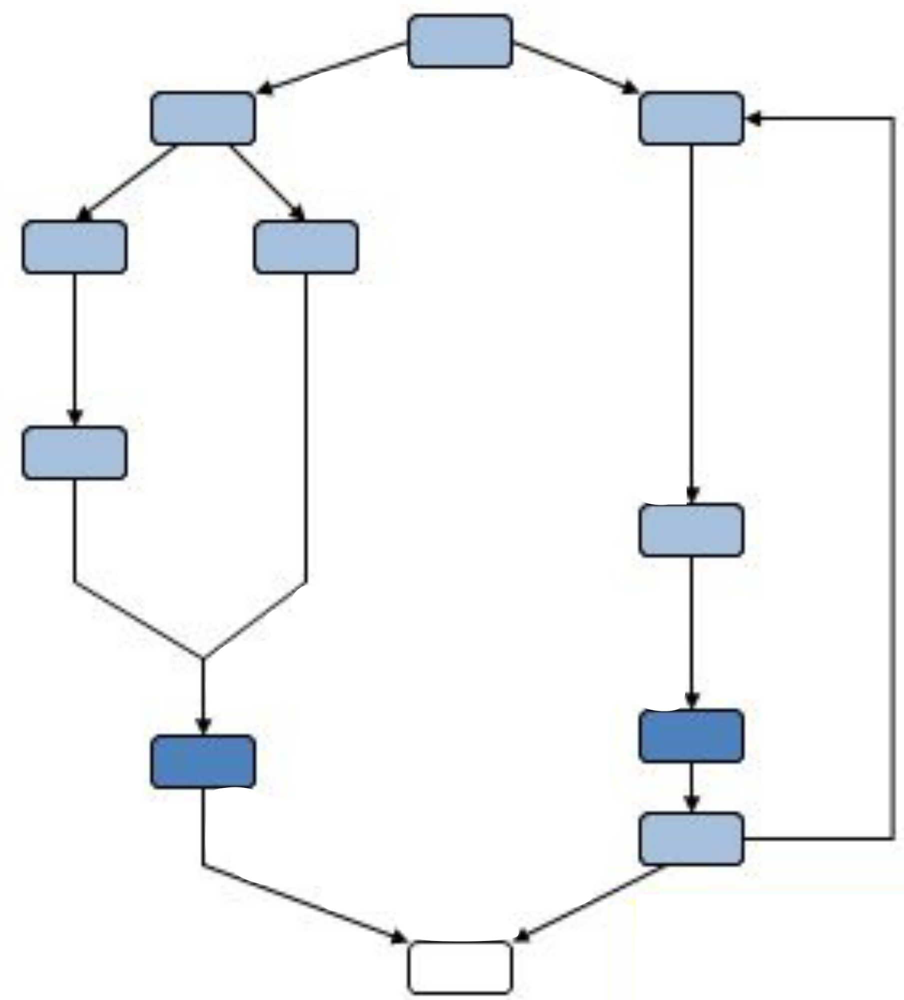


Program Slicing

- Generates a **subset** of the original program, while preserving the **specific behavior** of the original program.

- Specific behavior: Slicing Criterion $< i , V >$ (i : line number, V : variable name)

- Limitations: lack of supports on
  - code unrecognizable dependency
  - multi-lingual systems



Dependency analysis

Observation-Based Slicing (ORBS)

- Makes a series of deletions of code lines, which

    1) leaves the code (still) compilable, and

    2) preserves the trajectory of the slicing criterion.

➡ Purely dynamic & **Language Independent**

- Able to slice programs on which

    - static slicers are unable to make the minimal slice,

      (*Binkley et al. 2015, ORBS and the Limits of Static Slicing*)

    - have highly unconventional semantics.

      (*Yoo et al. 2017, Observational slicing based on visual semantics*)

Observation-Based Slicing (ORBS)

- Makes a series of deletions of code lines, which

  1) leaves the code (still) compilable, and

Approximates the program dependence via observation of test executions

- static slicers are unable to make the minimal slice,

  (*Binkley et al. 2015, ORBS and the Limits of Static Slicing*)

- have highly unconventional semantics.

  (*Yoo et al. 2017, Observational slicing based on visual semantics*)

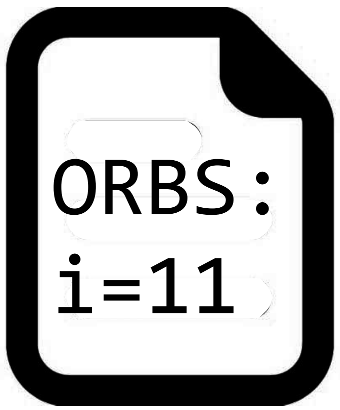
Observation-Based Slicing (ORBS)

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("ORBS: %d\n", i);
}
```


"ORBS: Language-Independent Program Slicing", FSE14

Observation-Based Slicing (ORBS)

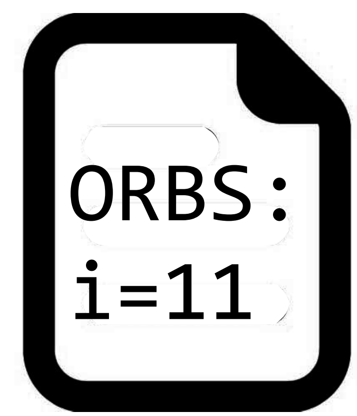
```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("ORBS: %d\n", i);
}
```



"ORBS: Language-Independent Program Slicing", FSE14

Observation-Based Slicing (ORBS)

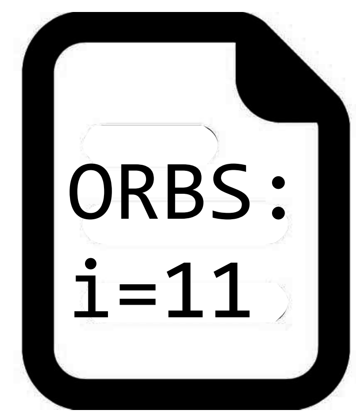
```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
    printf(“ORBS: %d\n”, i);
```



“ORBS: Language-Independent Program Slicing”, FSE14

Observation-Based Slicing (ORBS)

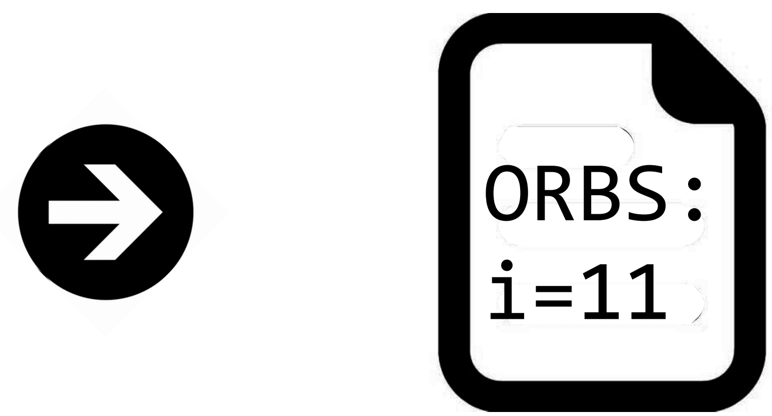
```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
```



“ORBS: Language-Independent Program Slicing”, FSE14

Observation-Based Slicing (ORBS)

```c
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
}
```

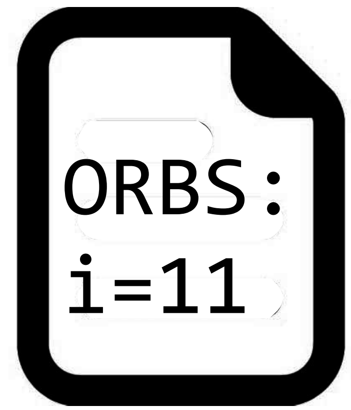


“ORBS: Language-Independent Program Slicing”, FSE14

Observation-Based Slicing (ORBS)

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("ORBS: %d\n", i);
}
```

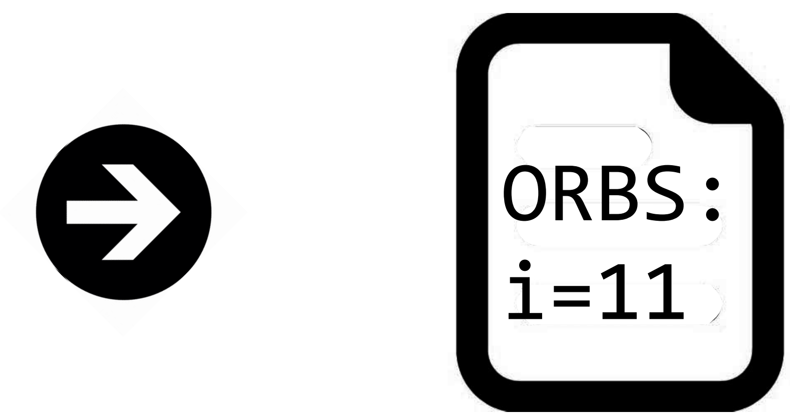


"ORBS: Language-Independent Program Slicing", FSE14

Observation-Based Slicing (ORBS)

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("ORBS: %d\n", i);
}
```

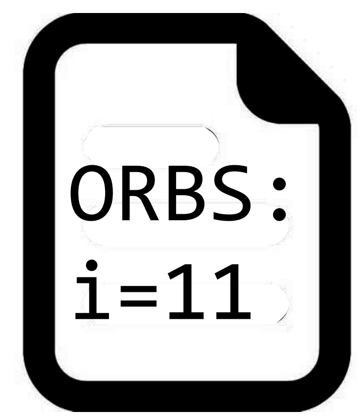


"ORBS: Language-Independent Program Slicing", FSE14

Observation-Based Slicing (ORBS)

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("ORBS: %d\n", i);
}
```



Window-Deletion

"ORBS: Language-Independent Program Slicing", FSE14

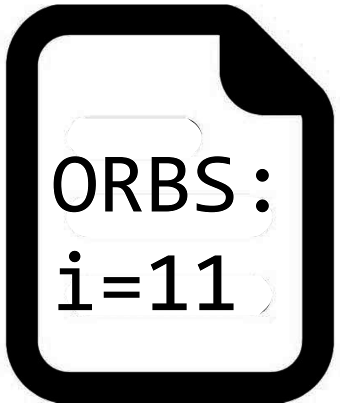
Observation-Based Slicing (ORBS)

```
int main() {

    int i = 1;
    while (i < 11) {

        i = i + 1;
    }

    printf("ORBS: %d\n", i);
}
```



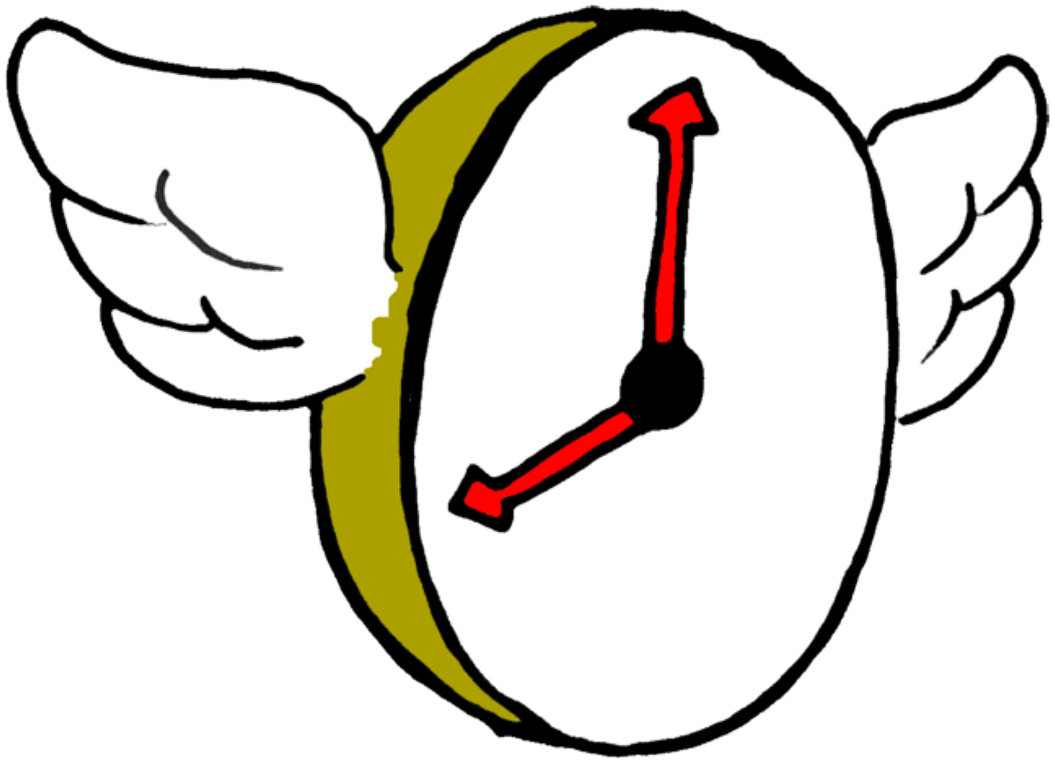
Window-Deletion

"ORBS: Language-Independent Program Slicing", FSE14

Limitations of ORBS

- Scalability
  - Takes around 7200 s to delete 220 lines.
    $\Rightarrow$ **0.03 del/s**
    $\Rightarrow$ **32.7 s/del**

(‘`escape`’ package on Guava)



Scalability

Efficiency

$$\textbf{Efficiency} = \frac{\textbf{Number of Deleted Lines}}{\textbf{Time spent}}$$

Efficiency

$$\text{Efficiency} = \frac{\text{Number of Deleted Lines}}{\text{Deletion Attempt}}$$

Efficiency

=

↑ Number of Deleted Lines / Deletion Attempt

$\uparrow$ **Efficiency** $=$ $\dfrac{\uparrow \textbf{Number of Deleted Lines}}{\textbf{Deletion Attempt}}$

```java

private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNar

private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finalize

```

```java
      try {
        ((FinalizableReference) reference).finalizeReferent();
      } catch (Throwable t) {
        logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
      }
```

```java

  private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNa

  private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finaliz

```

```java
      try {
        ((FinalizableReference) reference).finalizeReferent();
      } catch (Throwable t) {
        logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
      }
```

```python
        except Exception:
            if not from_error_handler:
                raise
            self.logger.exception('Request finalizing failed with an ' 'error while handling
        return response
```

“ Delete all lines of code that are related to a word ‘log’! ”

```

private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNa

private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finaliz

```

```
try {
  ((FinalizableReference) reference).finalizeReferent();
} catch (Throwable t) {
  logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
}
```

```
except Exception:
    if not from_error_handler:
        raise
    self.logger.exception('Request finalizing failed with an ' 'error while handling
return response
```

“ Delete all lines of code that are related to a word ‘log’*! ”*

Dependence Approximation

Spatial
Neighborhood

“ Delete all lines of code that are related to a word ‘log’*! ”*

Dependence Approximation

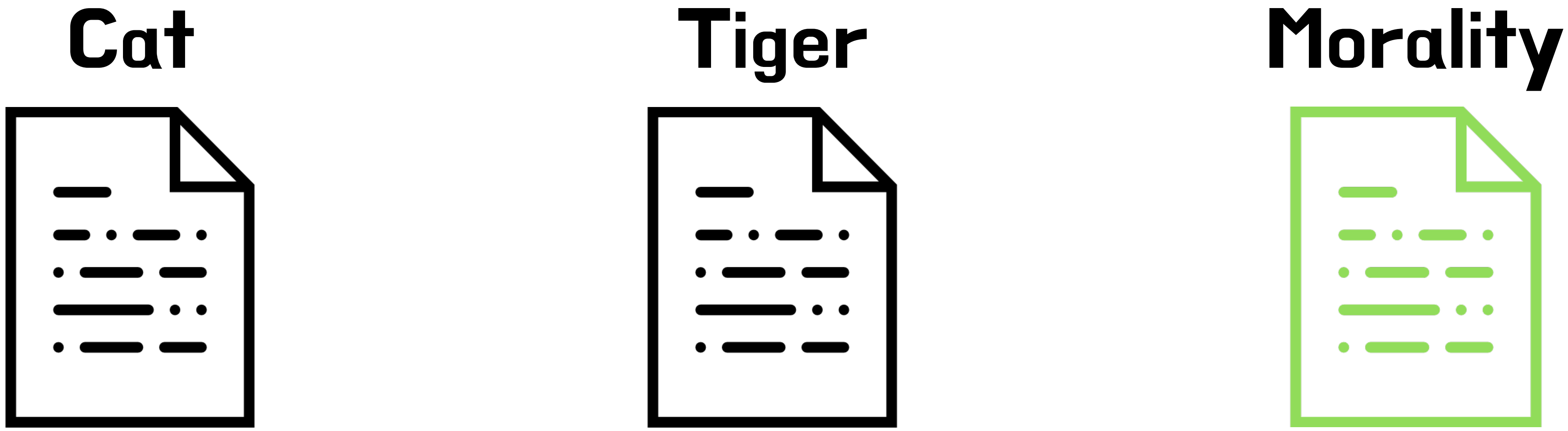
Spatial Neighborhood → Lexical Neighborhood

Deletion based on Lexical Similarity

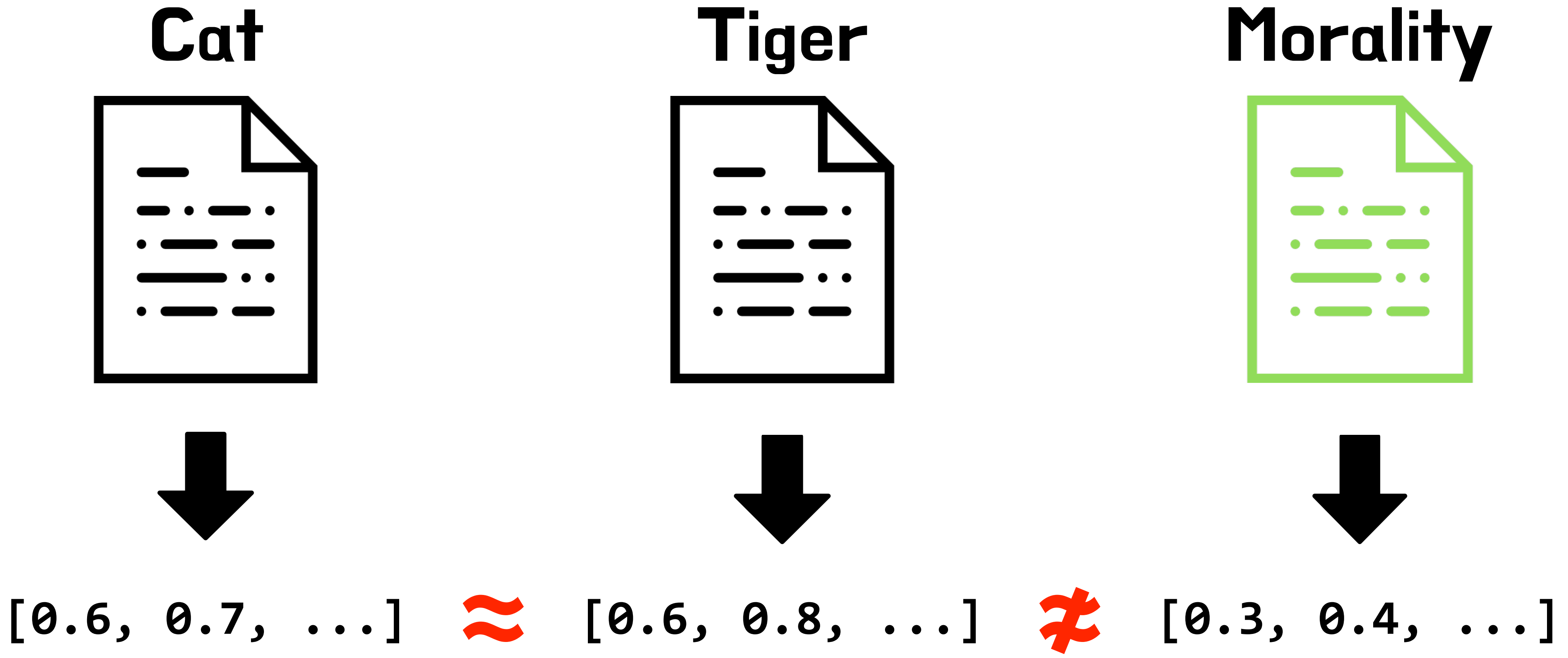
- Vector Space Model
    - Traditional method for calculating distances between text documents and a query.

- Latent Dirichlet Allocation
    - Probabilistic model that describes which topics are presented in a given document.

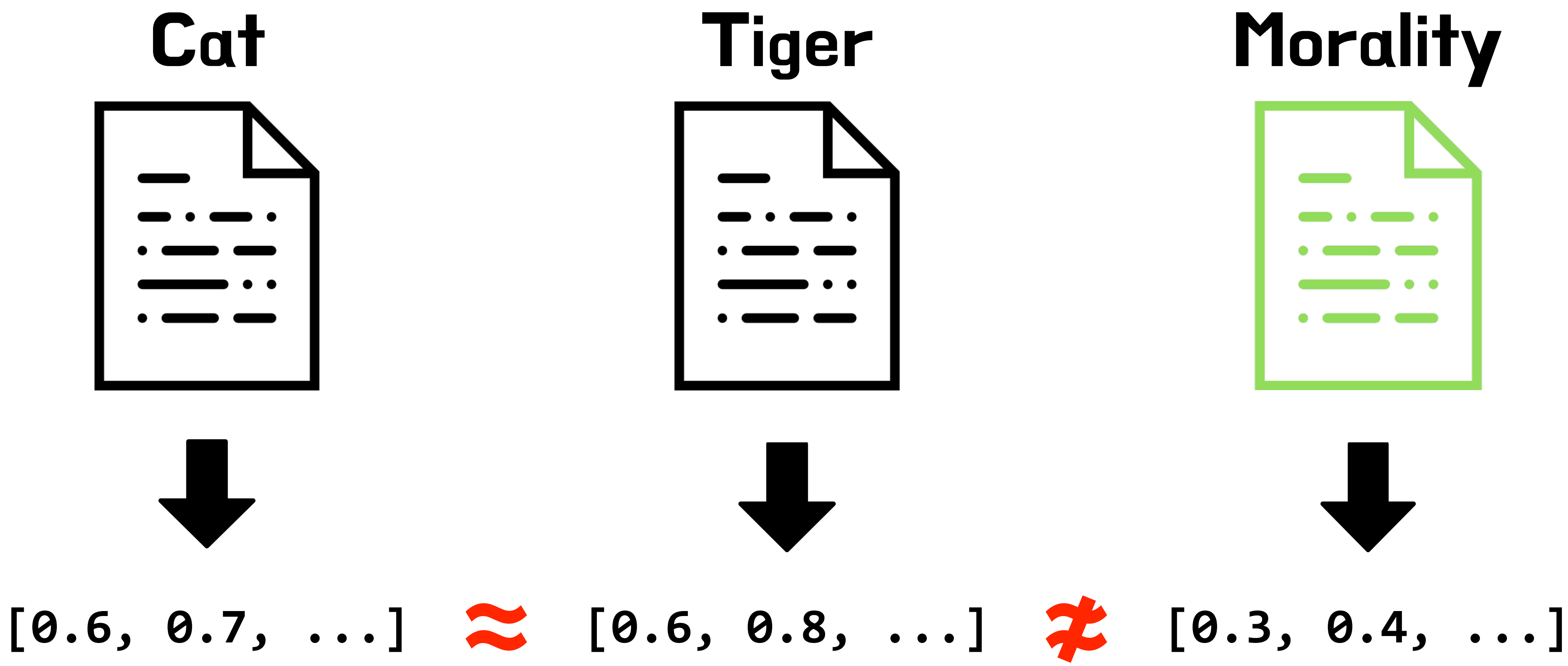
Deletion based on Lexical Similarity



Deletion based on Lexical Similarity



Deletion based on Lexical Similarity



- Consider each code lines as a document.
- Attempts to delete a set of code lines whose similarity is above certain threshold.

Deletion based on Lexical Similarity

- Vector Space Model

  ➡ **VSM-Deletion**

- Latent Dirichlet Allocation

  ➡ **LDA-Deletion**

  - Consider each code lines as a document.
  - Attempts to delete a set of code lines whose similarity is above certain threshold.

Deletion based on Lexical Similarity

- Vector Space Model
  ➡ **VSM-Deletion**
- Latent Dirichlet Allocation
  ➡ **LDA-Deletion**

⇒ Lexical Similarity based ORBS (LS-ORBS)

(*Lee et al. 2016, Using source code lexical similarity to improve efficiency of observation based slicing*)

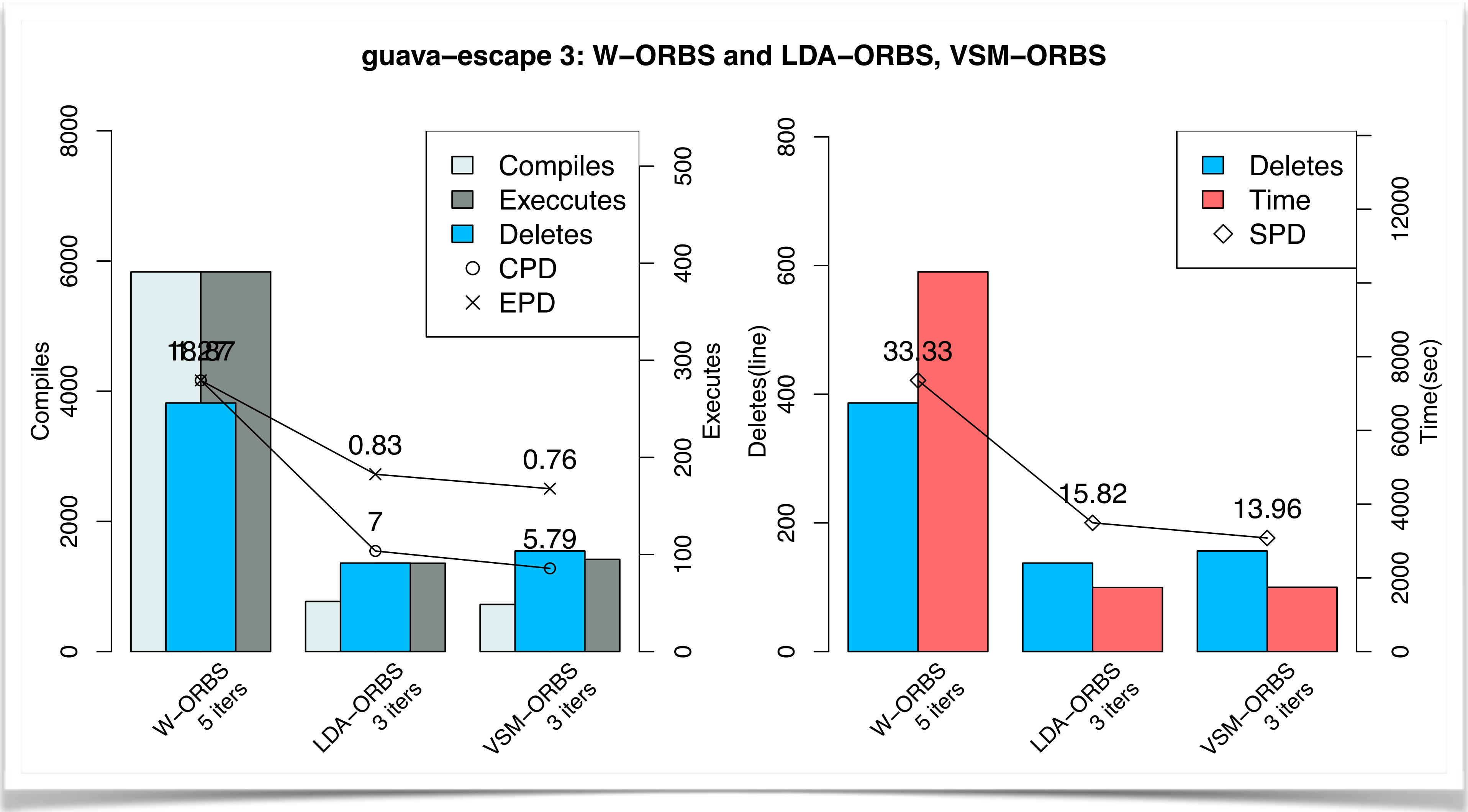
Advantage of LS-ORBS w.r.t ORBS

ORBS (Window)	LS-ORBS (VSM, LDA)
Can delete only **up to *k* lines** per deletion attempt (*k* = window size)	Can delete an **arbitrary number** of similar lines with a single deletion
Can delete **consecutive** lines only	Can delete **non-consecutive** lines
Multiple deletion attempt on a single line at each iteration	Single line - **Single attempt** at each iteration

LS-ORBS - Subjects

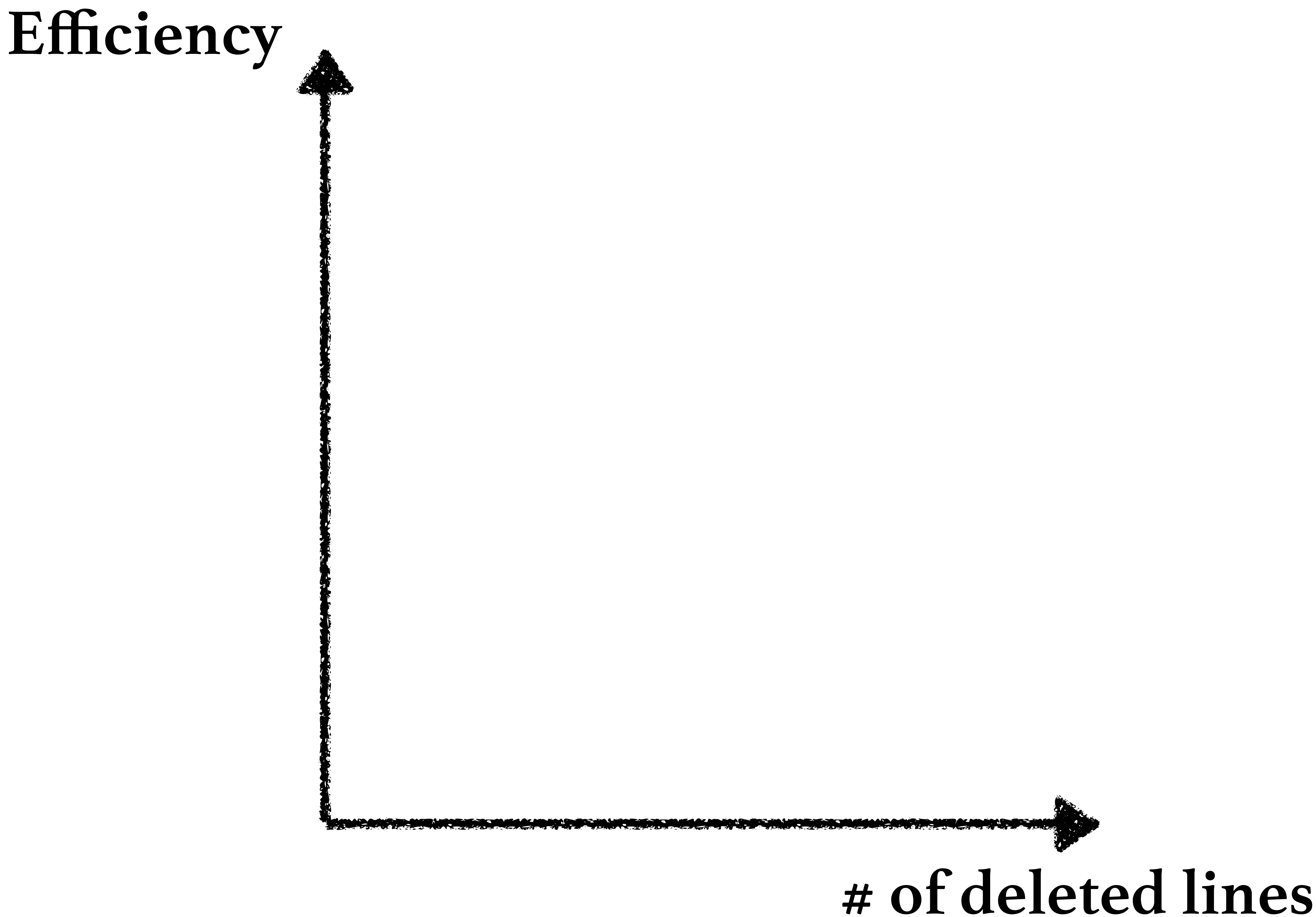
- **Java:** 12 slice criteria from 3 open source java projects
  - 3 slice criteria for each of subpackage ‘`escape`’ and ‘`net`’ from `Guava`.
  - 3 slice criteria for each of `apache-commons-csv`, `cli`
- **C**: Siemens Suite (except `tcas`)
  - 1 slice criteria for each program (total 6)

LS-ORBS - Results

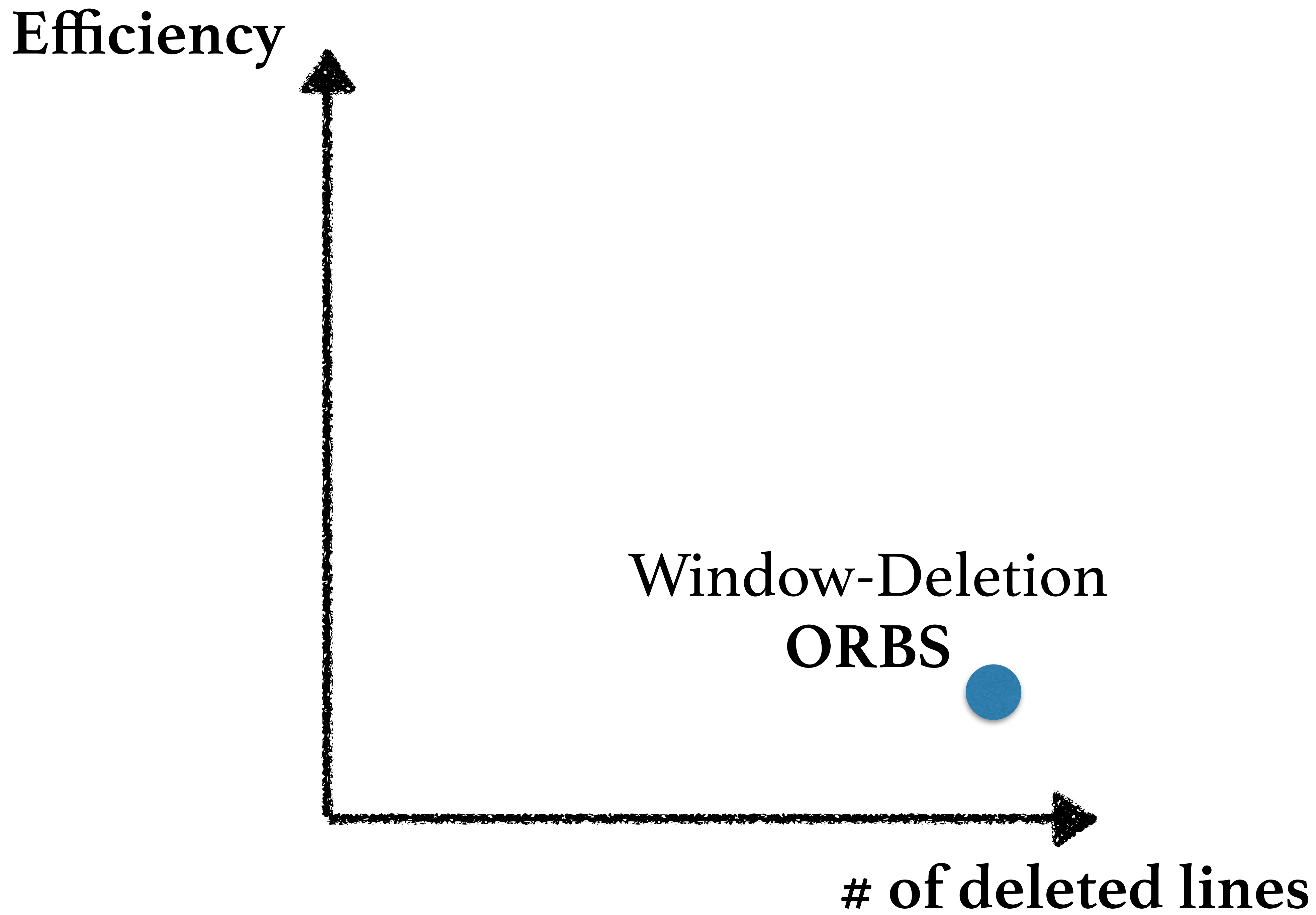


44.79% # of compilations, **69.56%** # of executions, **63.74%** time taken per deleted line.

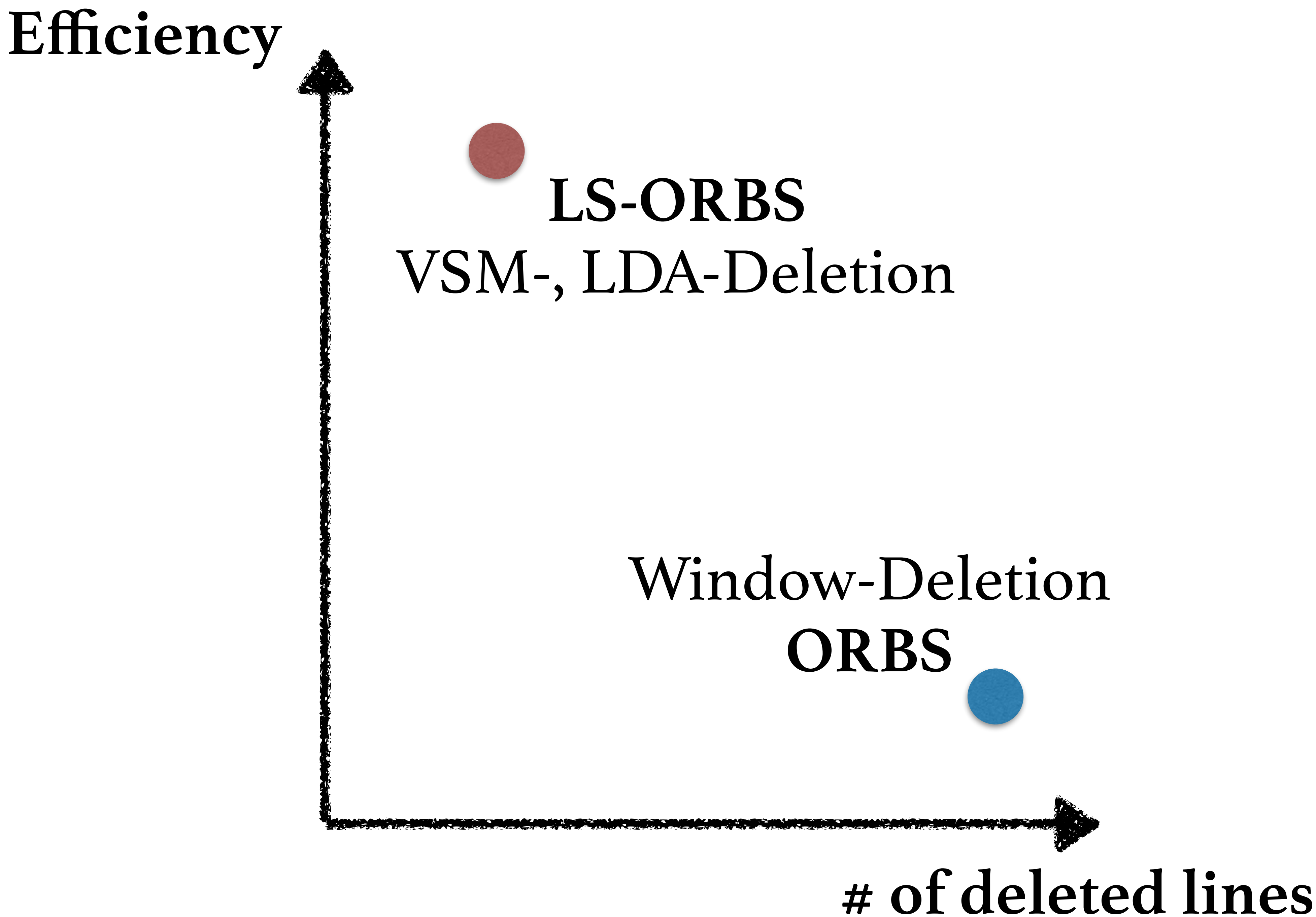
Compare Strategies



Compare Strategies



Compare Strategies



Compare Strategies

LS-ORBS

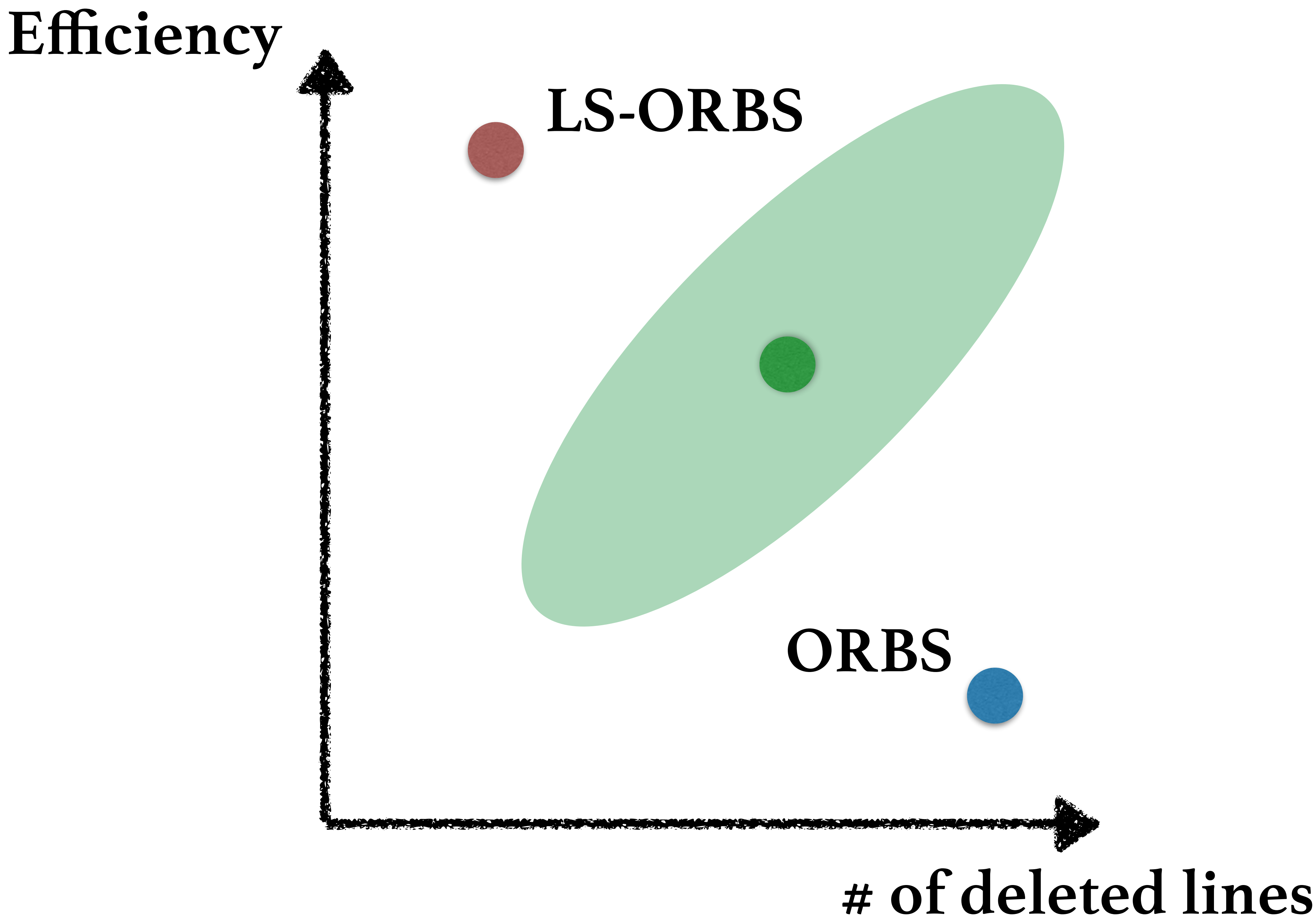
ORBS

Compare Strategies

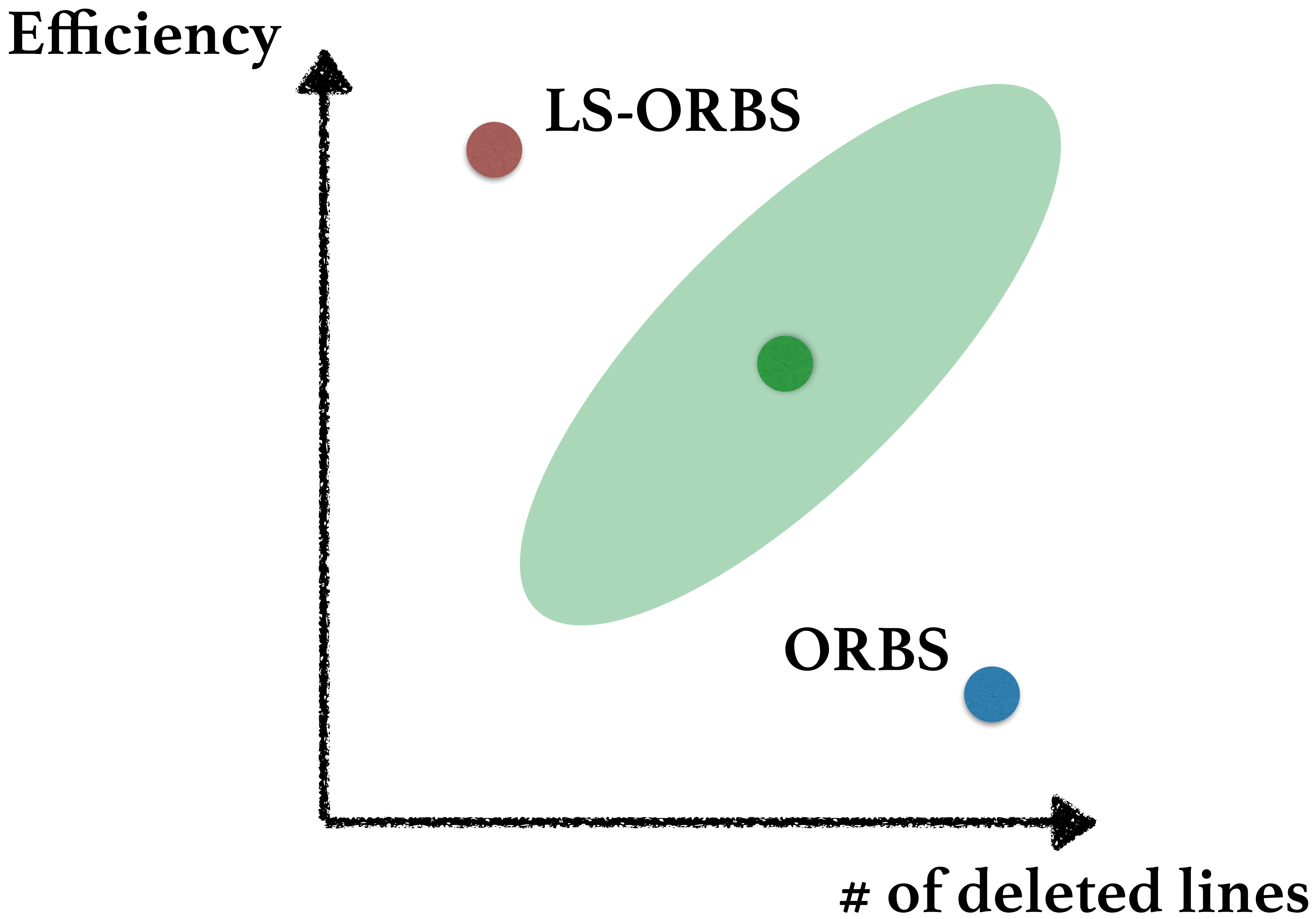
VSM-, LDA-Deletion
+
Window-Deletion

LS-ORBS **ORBS**

Compare Strategies

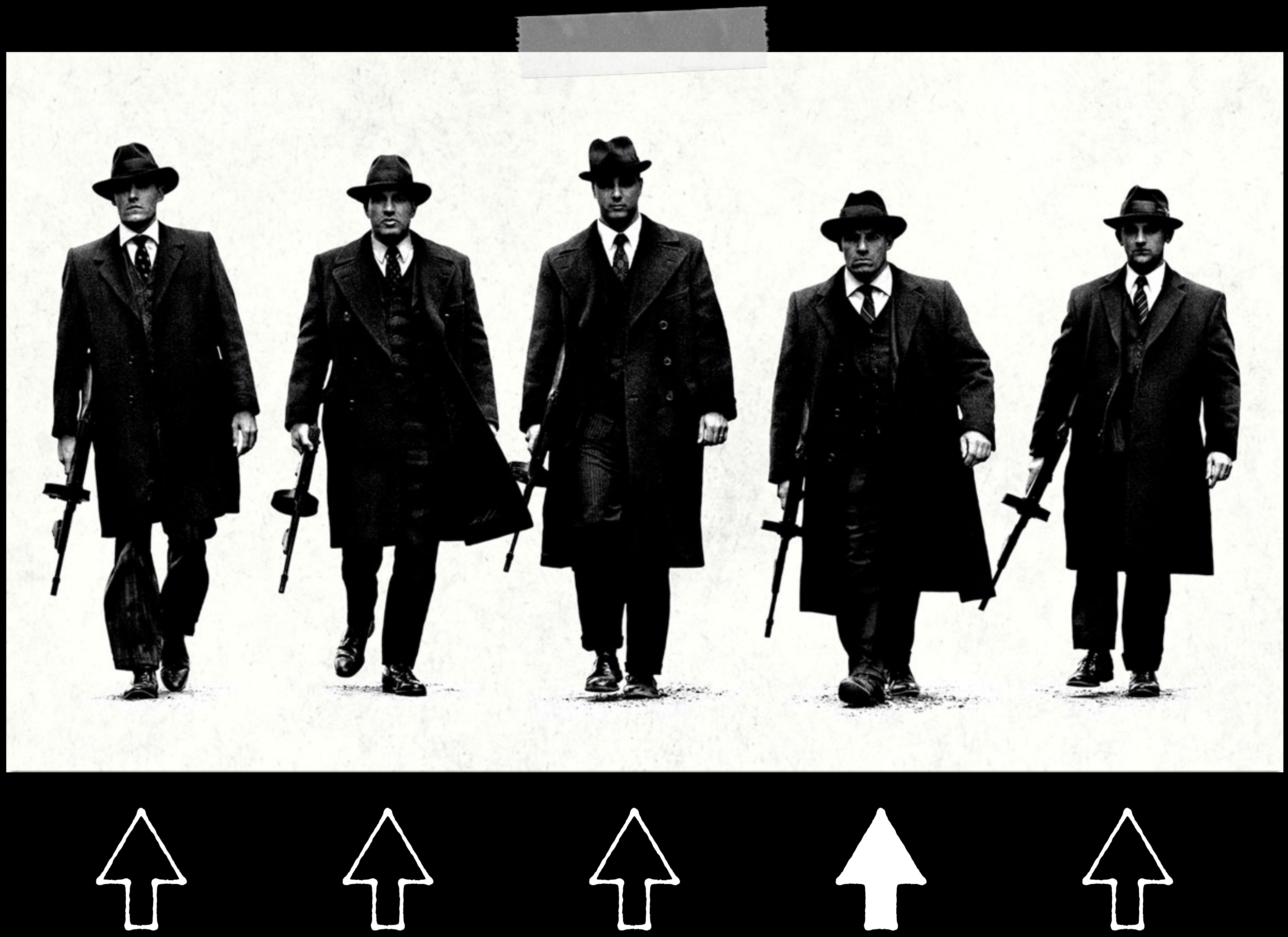


Multi-operator Observation-Based Slicing



MOBS

Q. How to select the operator among various kind of deletion operators ?



Operator Selection Strategies

Operator Selection Strategies

Operator Selection Strategies

Fixed Operator Selection (FOS)

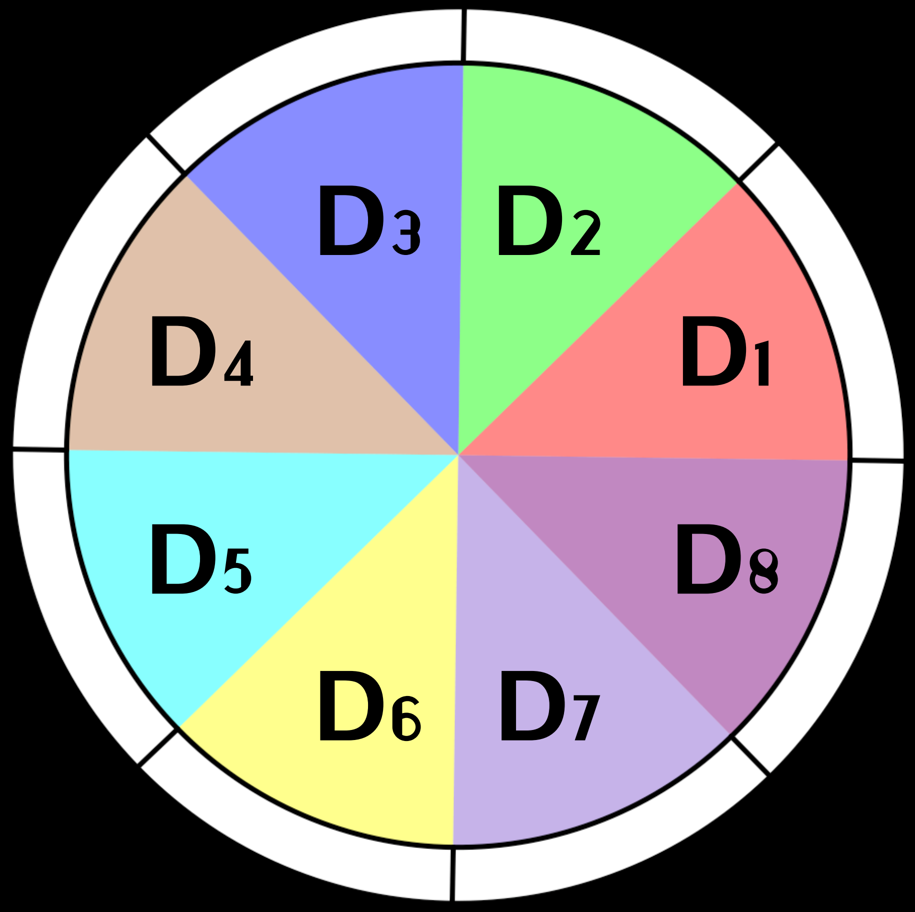
- Uniform
- Applicability
  - Success rate
- Affect
  - # of deletable lines

Operator Selection Strategies

Fixed Operator Selection (FOS)

- Uniform
- Applicability
  - Success rate
- Affect
  - # of deletable lines

Rolling Operator Selection (ROS)

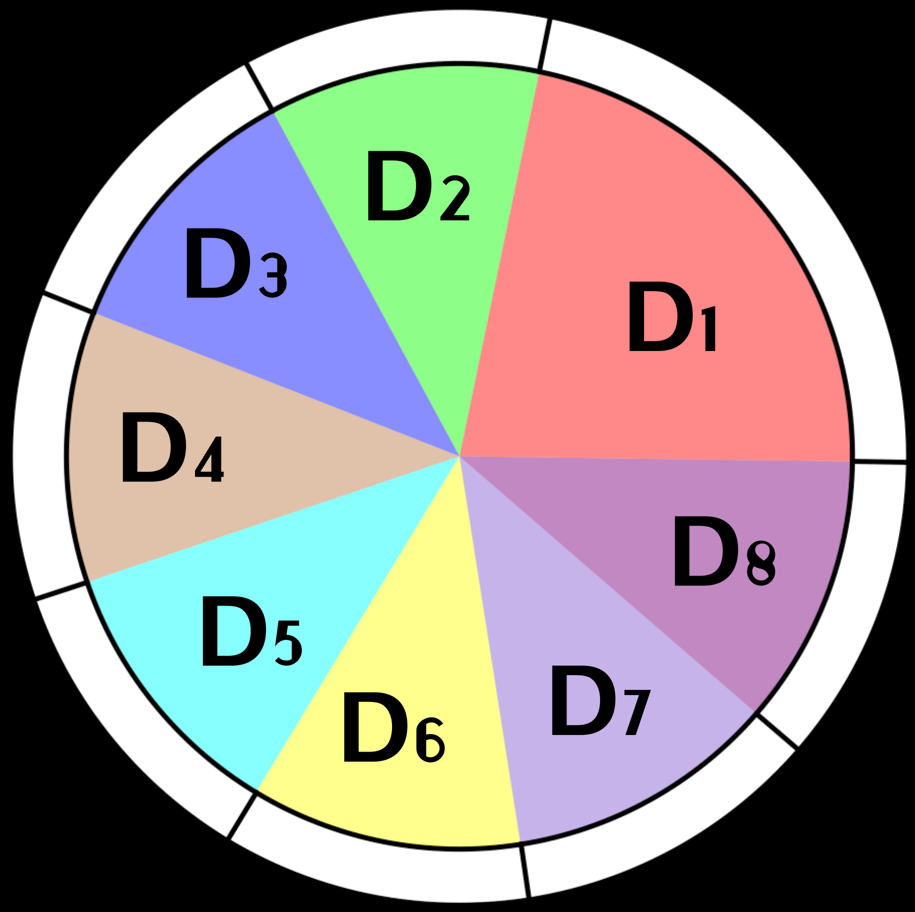


Operator Selection Strategies

Fixed Operator Selection (FOS)

- Uniform
- Applicability
  - Success rate
- Affect
  - # of deletable lines

Rolling Operator Selection (ROS)

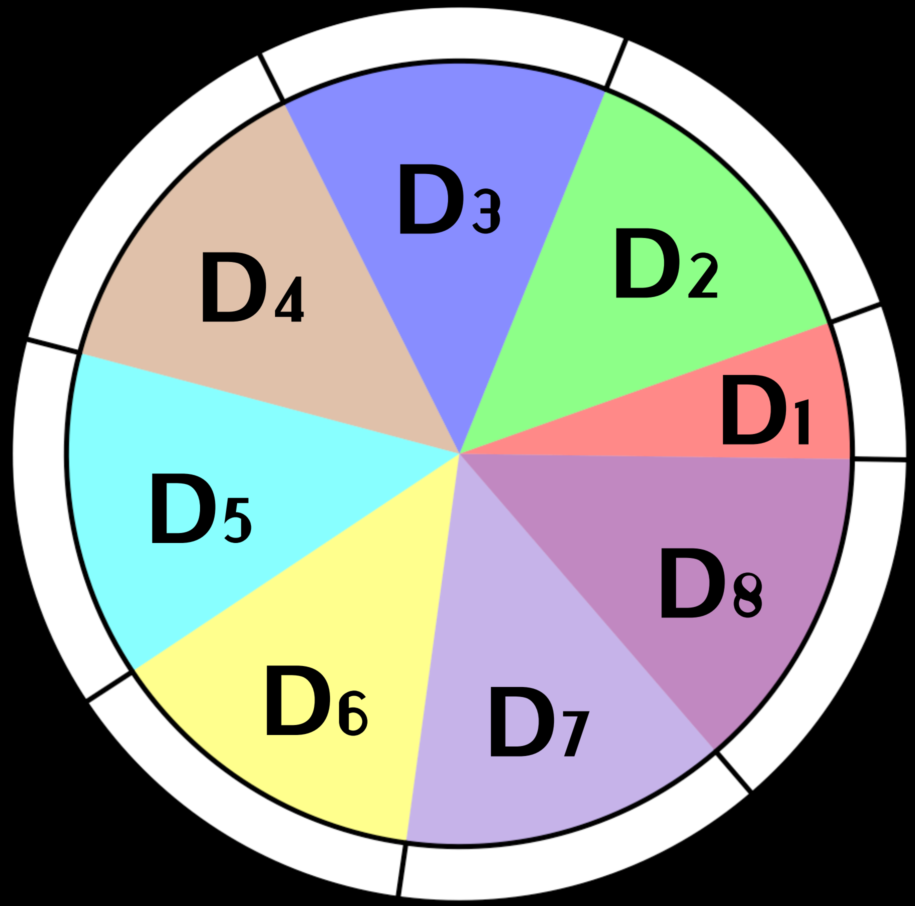


Operator Selection Strategies

Fixed Operator Selection (FOS)

- Uniform
- Applicability
  - Success rate
- Affect
  - # of deletable lines

Rolling Operator Selection (ROS)

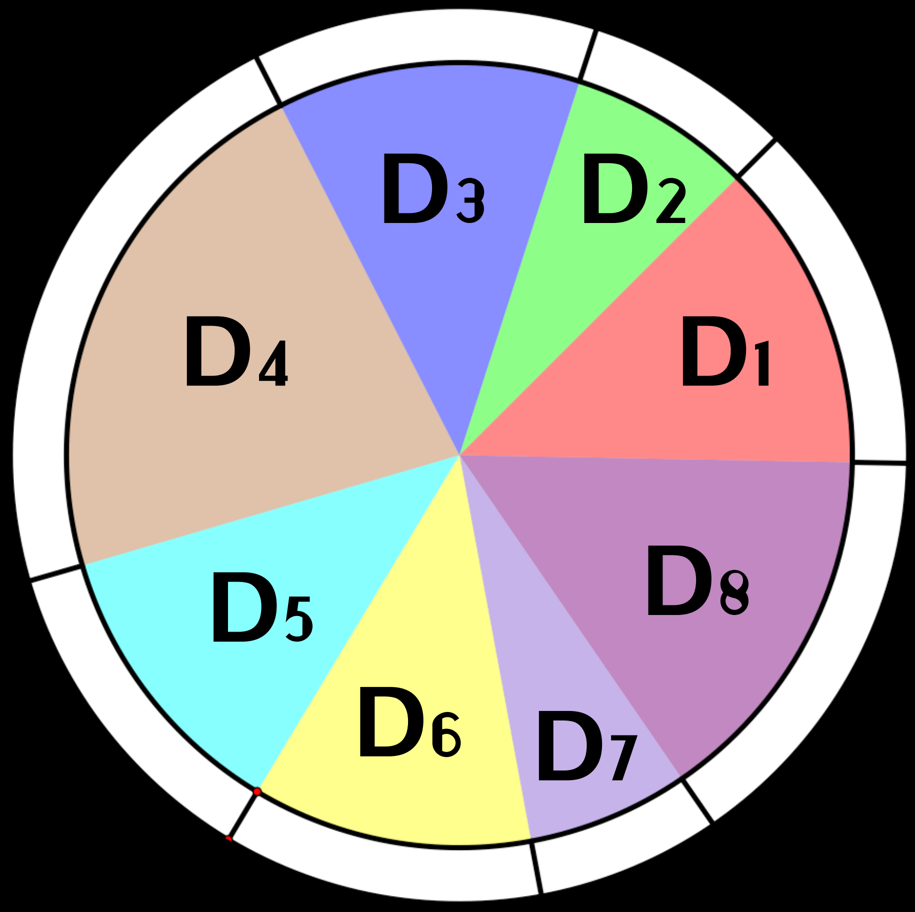


Operator Selection Strategies

Fixed Operator Selection (FOS)

- Uniform
- Applicability
  - Success rate
- Affect
  - # of deletable lines

Rolling Operator Selection (ROS)



MOBS - Configuration

- 12 Studied Deletion Operators
  - Window-Deletion of size 1, 2, 3, 4.
  - VSM-, LDA-Deletion of threshold 0.6, 0.7, 0.8, 0.9.
- Repeats 20 times for each operator selection strategy.

MOBS - Results

Table 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

Criteria	Strategy	μ_{del}	μ_{time}	μ_{spd}	Speedup
commons-cli	ROS-MOBS	1051	20533	19.89	2.76
	FOS-app-MOBS	957	23697	25.32	2.40
	FOS-aff-MOBS	969	21690	22.89	2.62
	FOS-uni-MOBS	951	23653	25.31	2.40
	W-ORBS	1255	56897	46.01	1.00
commons-csv	ROS-MOBS	665	12850	19.86	3.61
	FOS-app-MOBS	618	14862	24.55	3.11
	FOS-aff-MOBS	625	14103	22.97	3.26
	FOS-uni-MOBS	606	13531	22.68	3.39
	W-ORBS	797	46008	58.78	1.00
guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

MOBS - Results

Table 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

Criteria	Strategy	μ_{del}	μ_{time}	μ_{spd}	Speedup
commons-cli	ROS-MOBS	1051	20533	19.89	2.76
	FOS-app-MOBS	957	23697	25.32	2.40
	FOS-aff-MOBS	969	21690	22.89	2.62
	FOS-uni-MOBS	951	23653	25.31	2.40
	W-ORBS	1255	56897	46.01	1.00
commons-csv	ROS-MOBS	665	12850	19.86	3.61
	FOS-app-MOBS	618	14862	24.55	3.11
	FOS-aff-MOBS	625	14103	22.97	3.26
	FOS-uni-MOBS	606	13531	22.68	3.39
	W-ORBS	797	46008	58.78	1.00
guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

MOBS - Results

Table 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

Criteria	Strategy	μ_{del}	μ_{time}	μ_{spd}	Speedup
commons-cli	ROS-MOBS	1051	20533	19.89	2.76
	FOS-app-MOBS	957	23697	25.32	2.40
	FOS-aff-MOBS	969	21690	22.89	2.62
	FOS-uni-MOBS	951	23653	25.31	2.40
	W-ORBS	1255	56897	46.01	1.00
commons-csv	ROS-MOBS	665	12850	19.86	3.61
	FOS-app-MOBS	618	14862	24.55	3.11
	FOS-aff-MOBS	625	14103	22.97	3.26
	FOS-uni-MOBS	606	13531	22.68	3.39
	W-ORBS	797	46008	58.78	1.00
guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

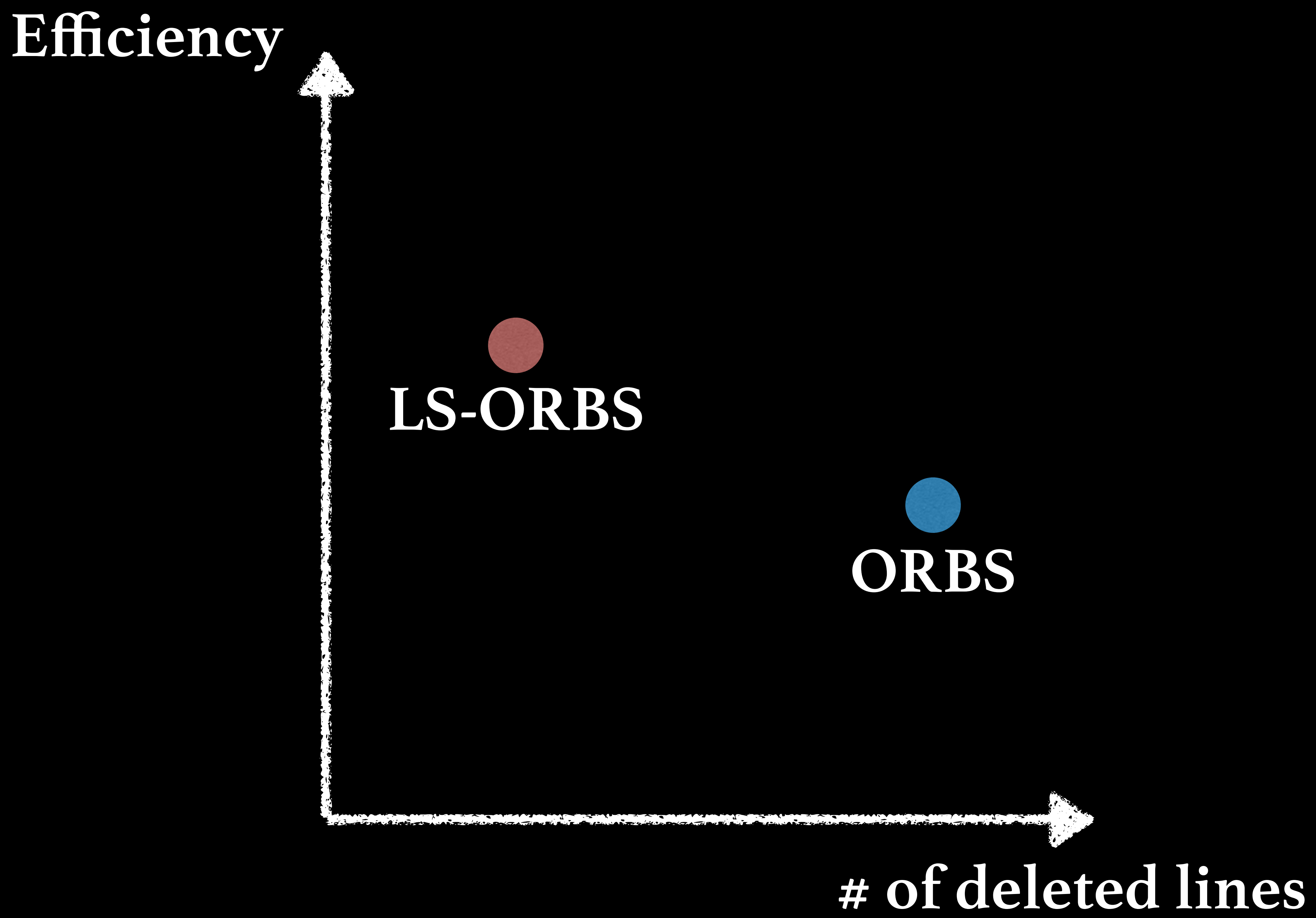
MOBS - Results

Table 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

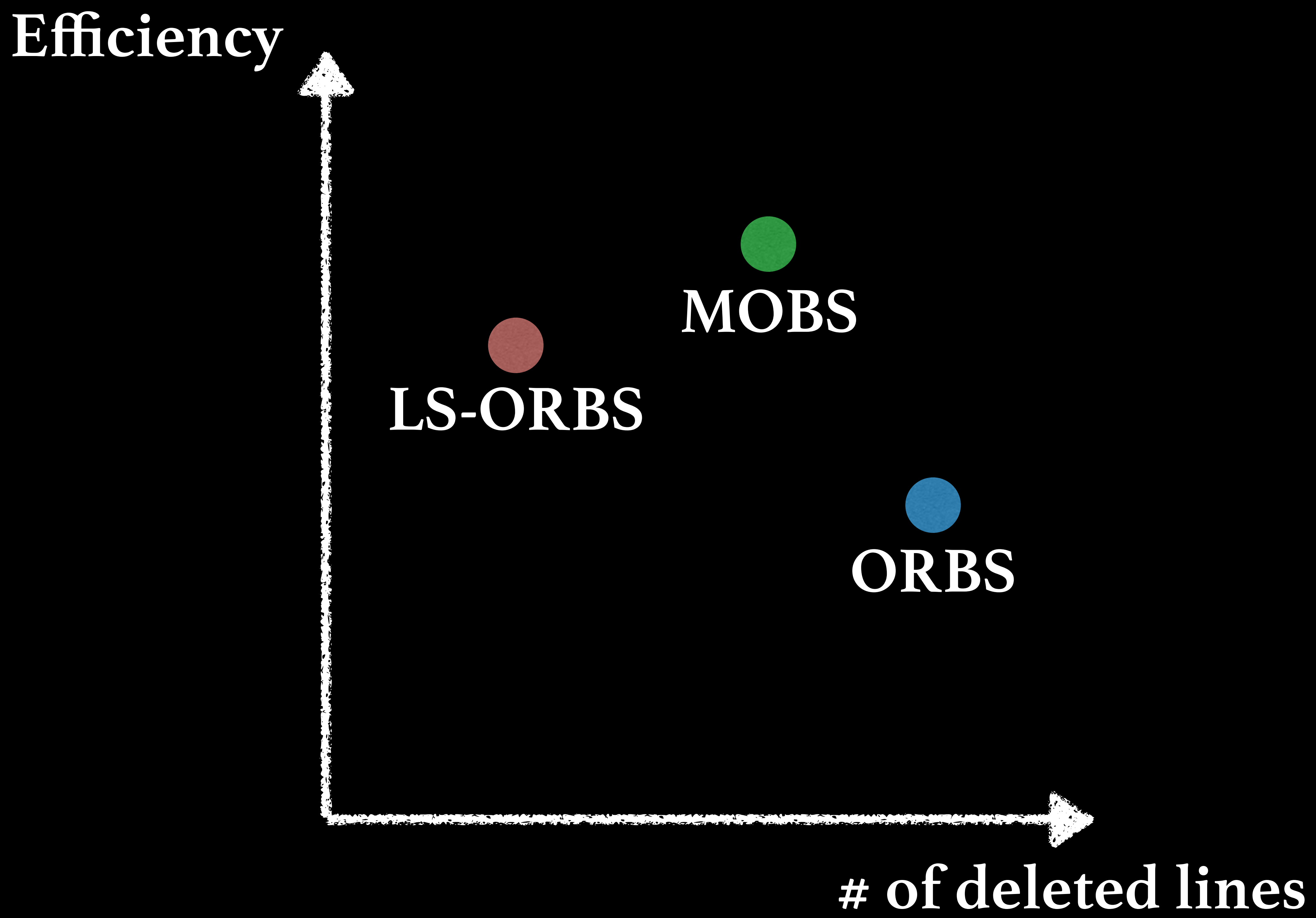
The results show that MOBS can delete up to 79% lines in less than 33% time compared to ORBS.

guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

Again, Compare Strategies



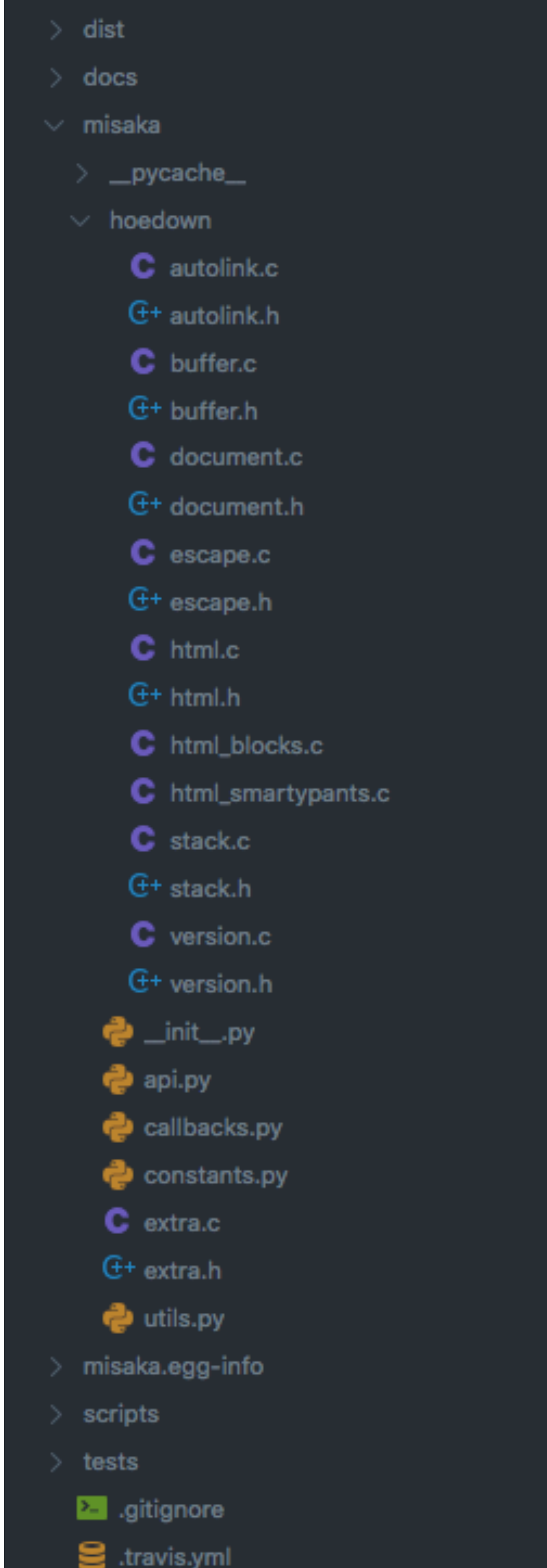
Again, Compare Strategies



Multi-language Subject

- Misaka(http://misaka.61924.nl)
    - A Python binding for Hoedown, a markdown parsing C library.
    - Programming language:
      C, Python

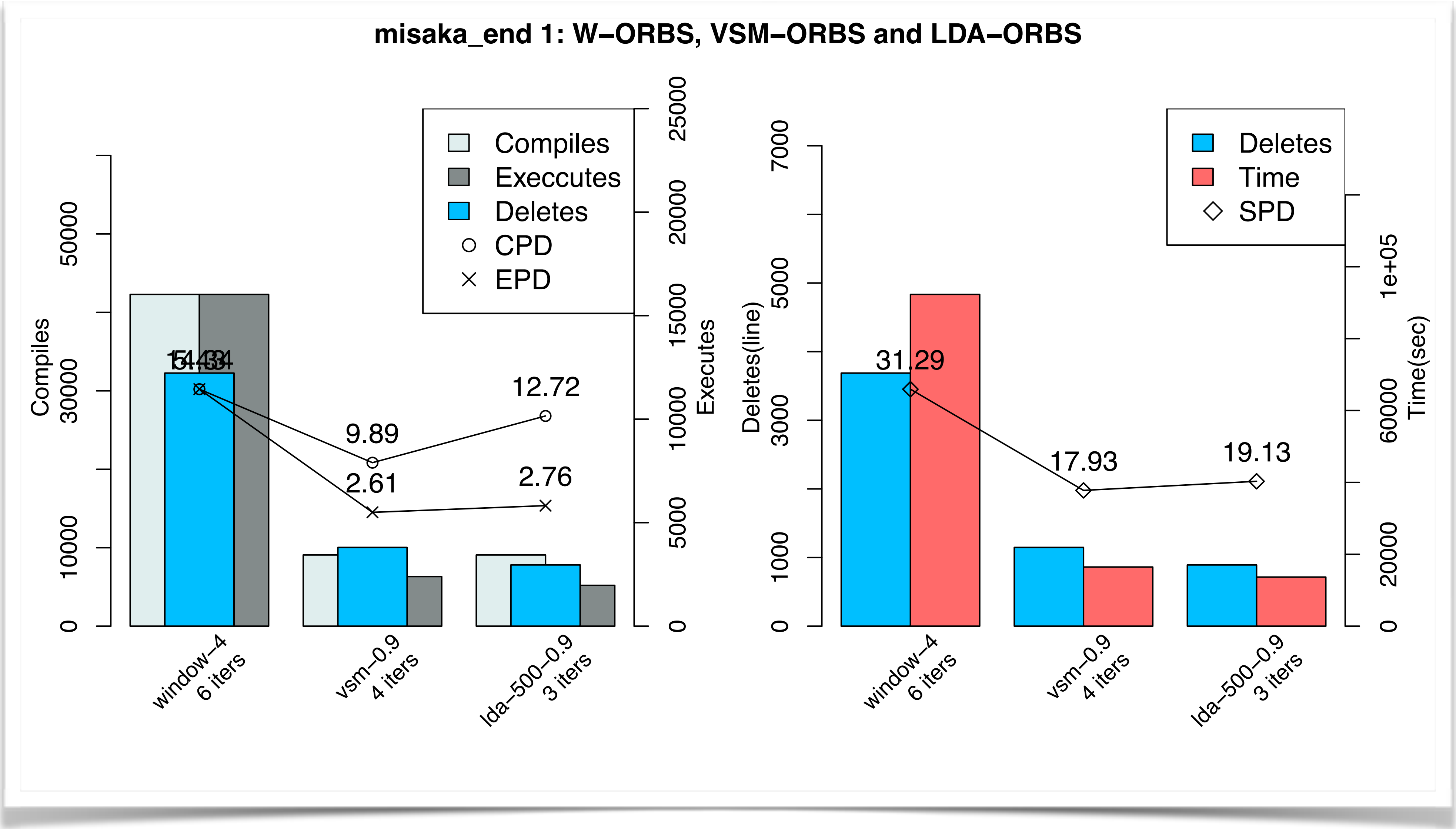
	NCLOC	FILES	TC
C	4360	10	
Python	473	5	
Total	4833	15	92



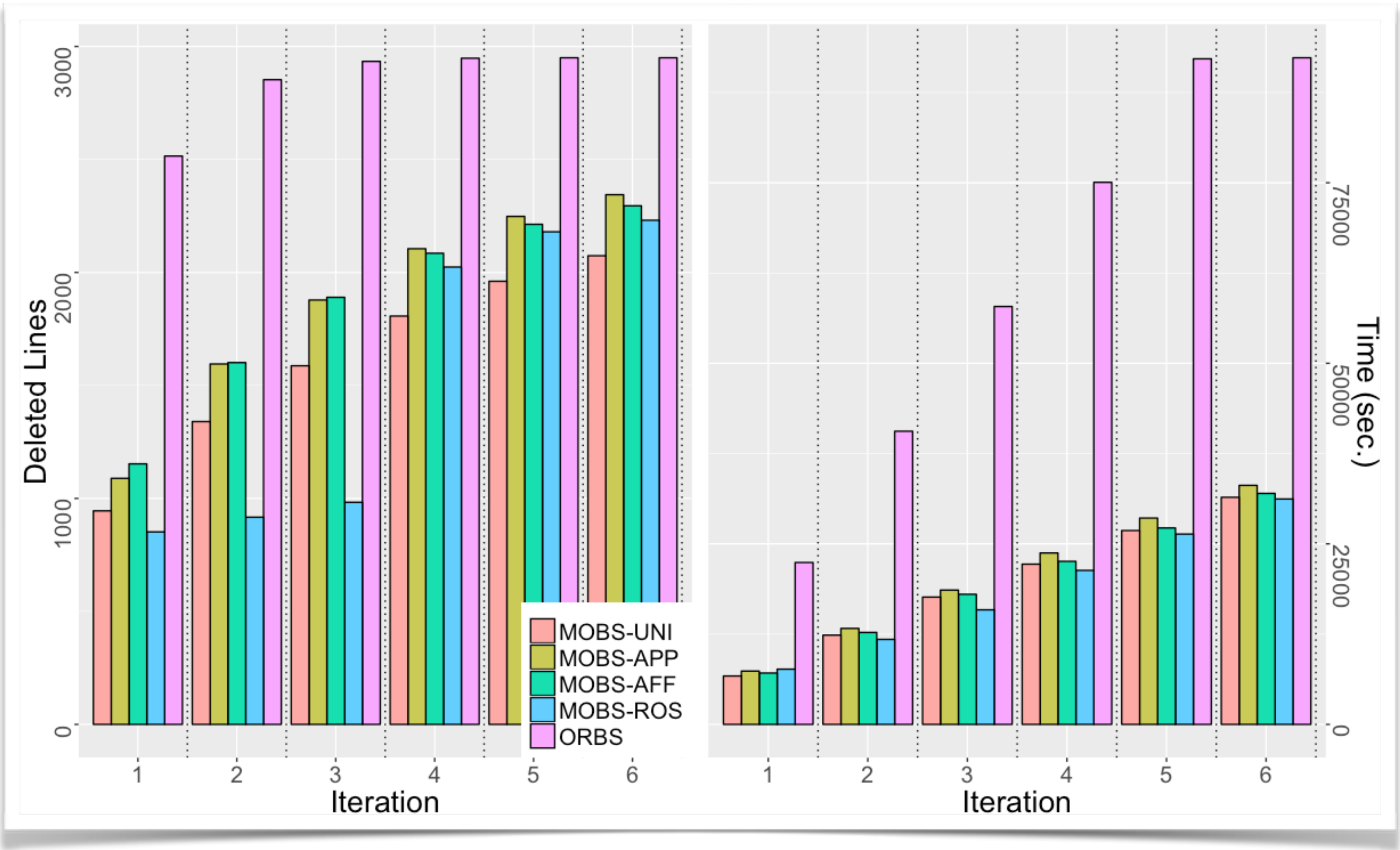
Multi-language Configuration

- **LS-ORBS**
  - Construct VSM and LDA model using the source code with both programming language at once.
- **MOBS**
  - Use the same deletion operators.
  - Run the experiments once for each probability distribution strategy.

LS-ORBS Results



MOBS Results



75% lines / 2.9X speed

Multi-lingual Deletion Examples

- VSM Deletion operator

```
┌ callbacks.py                (97)   > elif align_bit == TABLE_ALIGN_LEFT:
| callbacks.py                (98)   >     align = ‘left'
└ hoedown/html.c              (195)  > case HOEDOWN_TABLE_ALIGN_LEFT:
```

- LDA Deletion operator

```
┌ api.py                      (29)   > lib.hoedown_buffer_puts(ib, text.encode('utf-8'))
| hoedown/document.c          (2490) > hoedown_buffer_free(text);
└ hoedown/html_smartypants.c (195)  > hoedown_buffer_putc(ob, text[0]);
```

- Both LDA and VSM Deletion operator

```
┌ callbacks.py                (125)  > result = renderer.blockhtml(text)
└ hoedown/html.c              (635)  > renderer->blockhtml = NULL;
```

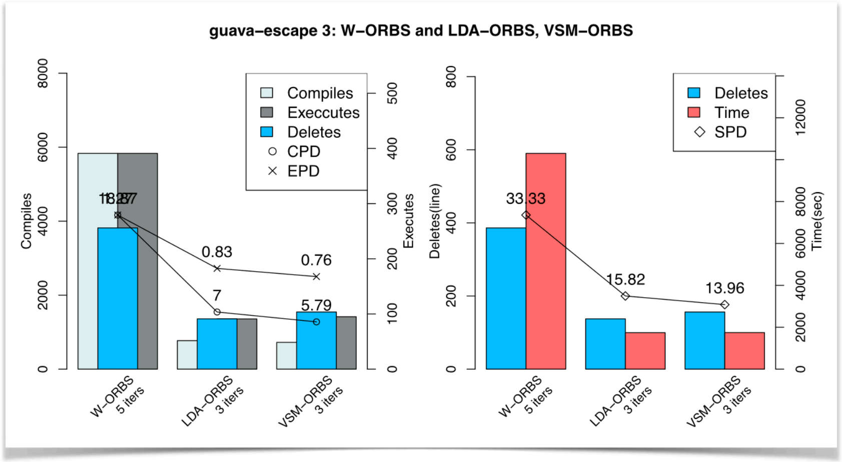
Deletion based on Lexical Similarity

“Delete all lines of code that are related to a word ‘log’!”

Dependence Approximation

Spatial Neighborhood → Lexical Neighborhood

LS-ORBS - Results



44.79% # of compilations, **69.56%** # of executions, **63.74%** time taken per deleted line.

MOBS

Multi-lingual Deletion Examples

- VSM Deletion operator

```
┌ callbacks.py                (97)   > elif align_bit == TABLE_ALIGN_LEFT:
| callbacks.py                (98)   >     align = 'left'
└ hoedown/html.c              (195)  > case HOEDOWN_TABLE_ALIGN_LEFT:
```

- LDA Deletion operator

```
┌ api.py                      (29)   > lib.hoedown_buffer_puts(ib, text.encode('utf-8'))
| hoedown/document.c          (2490) > hoedown_buffer_free(text);
└ hoedown/html_smartypants.c  (195)  > hoedown_buffer_putc(ob, text[0]);
```

- Both LDA and VSM Deletion operator

```
┌ callbacks.py                (125)  > result = renderer.blockhtml(text)
└ hoedown/html.c              (635)  > renderer->blockhtml = NULL;
```

Appendix A. Applicability & Affect

- Apply every deletion operators on each code line individually, record the comparability and # of deleted lines.

Appendix A. Applicability & Affect

Table 3: Comparison between Deletion Operators

Criteria	$\sqcup$	W	V	L	$W \setminus V$	$W \setminus L$	$V \setminus W$	$V \setminus L$	$L \setminus W$	$L \setminus V$	$\mathcal{V}_{DPS}$	$\mathcal{L}_{DPS}$
cli-1	540	539	210	162	330	377	1	83	0	35	5.60	3.36
cli-2	636	633	269	177	367	456	3	125	0	33	7.04	2.46
cli-3	638	634	275	241	362	396	3	84	3	50	6.99	4.09
csv-1	487	485	207	146	279	340	1	77	1	16	5.76	5.29
csv-2	418	418	183	119	235	299	0	75	0	11	6.08	3.21
csv-3	486	486	212	121	274	365	0	101	0	10	5.82	2.31
esc-1	144	144	87	82	57	62	0	8	0	3	6.78	2.53
esc-2	130	130	80	72	50	58	0	12	0	4	7.12	3.13
esc-3	163	163	98	99	65	64	0	10	0	11	6.39	3.49
net-1	567	563	378	212	189	351	4	175	0	9	3.53	3.16
net-2	542	538	373	209	169	329	4	170	0	6	3.57	3.74
net-3	582	578	387	192	195	386	4	205	0	10	3.52	2.29

Appendix A. Applicability & Affect

Table 4: Operator Proportions for FOS with Applicability

Criteria	DLDA with γ =				DVSM with γ =				DW with δ =			
	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9	1	2	3	4
commons-cli-1	0.026	0.039	0.071	0.090	0.033	0.049	0.081	0.116	0.215	0.092	0.110	0.079
commons-cli-2	0.029	0.035	0.058	0.085	0.033	0.052	0.093	0.125	0.213	0.085	0.121	0.072
commons-cli-3	0.030	0.041	0.069	0.103	0.030	0.047	0.083	0.116	0.199	0.094	0.105	0.082
commons-csv-1	0.050	0.057	0.070	0.078	0.035	0.055	0.081	0.111	0.160	0.092	0.118	0.093
commons-csv-2	0.041	0.053	0.066	0.077	0.037	0.055	0.091	0.118	0.169	0.091	0.115	0.087
commons-csv-3	0.043	0.048	0.058	0.067	0.038	0.064	0.093	0.118	0.164	0.092	0.123	0.094
guava-escape-1	0.089	0.091	0.100	0.103	0.069	0.073	0.094	0.109	0.140	0.043	0.049	0.040
guava-escape-2	0.088	0.094	0.097	0.099	0.072	0.070	0.099	0.110	0.146	0.046	0.043	0.036
guava-escape-3	0.093	0.093	0.102	0.109	0.063	0.070	0.093	0.108	0.136	0.043	0.048	0.041
guava-net-1	0.042	0.046	0.056	0.066	0.080	0.099	0.109	0.115	0.133	0.082	0.090	0.082
guava-net-2	0.045	0.049	0.061	0.066	0.080	0.098	0.109	0.115	0.130	0.081	0.085	0.082
guava-net-3	0.033	0.040	0.052	0.061	0.083	0.101	0.112	0.120	0.138	0.084	0.092	0.085

Appendix A. Applicability & Affect

Table 5: Operator Proportions for FOS with Affect

Criteria	DLDA with γ =				DVSM with γ =				DW with δ =			
	0.6	0.7	0.8	0.9	0.6	0.7	0.8	0.9	1	2	3	4
`commons-cli-1`	0.045	0.052	0.054	0.053	0.085	0.098	0.120	0.149	0.071	0.060	0.109	0.104
`commons-cli-2`	0.029	0.026	0.032	0.051	0.084	0.118	0.204	0.153	0.062	0.050	0.107	0.084
`commons-cli-3`	0.046	0.043	0.071	0.091	0.071	0.101	0.170	0.128	0.054	0.051	0.086	0.089
`commons-csv-1`	0.084	0.085	0.085	0.083	0.070	0.089	0.086	0.139	0.041	0.048	0.092	0.097
`commons-csv-2`	0.047	0.054	0.058	0.053	0.091	0.102	0.117	0.175	0.049	0.053	0.100	0.101
`commons-csv-3`	0.038	0.037	0.038	0.036	0.093	0.125	0.121	0.174	0.051	0.057	0.114	0.116
`guava-escape-1`	0.066	0.066	0.066	0.058	0.171	0.128	0.149	0.153	0.037	0.023	0.039	0.043
`guava-escape-2`	0.076	0.072	0.071	0.068	0.169	0.121	0.149	0.149	0.036	0.022	0.031	0.035
`guava-escape-3`	0.094	0.088	0.089	0.078	0.144	0.111	0.130	0.133	0.035	0.022	0.037	0.041
`guava-net-1`	0.057	0.057	0.054	0.052	0.137	0.120	0.111	0.105	0.046	0.056	0.093	0.112
`guava-net-2`	0.062	0.065	0.071	0.067	0.130	0.114	0.106	0.101	0.042	0.052	0.083	0.107
`guava-net-3`	0.031	0.036	0.039	0.044	0.148	0.130	0.121	0.115	0.050	0.061	0.101	0.123

Appendix B. ROS Formula

$$P_{new}(\mathrm{D}) = \begin{cases} \omega_{comp} \cdot P(\mathrm{D}) & \texttt{compilation-error} \\ \omega_{exec} \cdot P(\mathrm{D}) & \texttt{trajectory-change} \\ \left(1 + \log_{10}(line\_cnt + 1)\right) \cdot P(\mathrm{D}) & \texttt{success} \end{cases}$$

Appendix C. Statistics b/w MOBS Strategies

Table 6: Results of Mann-Whitney U Test on SPD between Selection Strategies after Bonferroni Correction

Criteria	p_{Bonf}											
	ROS < APP	ROS < AFF	ROS < UNI	APP < AFF	APP < UNI	AFF < UNI	ROS > APP	ROS > AFF	ROS > UNI	APP > AFF	APP > UNI	AFF > UNI
commons-cli-1	**<0.05**	**<0.05**	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	**<0.05**	1.0	1.0
commons-cli-2	0.10	1.0	**<0.05**	1.0	1.0	**<0.05**	1.0	1.0	1.0	0.81	1.0	1.0
commons-cli-3	**<0.05**	**<0.05**	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	**<0.05**	1.0	1.0
commons-csv-1	**<0.05**	**<0.05**	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
commons-csv-2	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**<0.05**	1.0
commons-csv-3	**<0.05**	0.40	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
guava-escape-1	1.0	0.11	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
guava-escape-2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.28	0.48
guava-escape-3	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**<0.05**	**<0.05**	1.0
guava-net-1	1.0	**<0.05**	**<0.05**	1.0	0.81	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0
guava-net-2	1.0	0.31	0.10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
guava-net-3	**<0.05**	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**<0.05**	0.19	1.0

Appendix C. Statistics b/w MOBS Strategies

Table 7: $\hat{A}_{12}$ on SPD between Selection Strategies

Criteria	Vargha-Delaney $\hat{A}_{12}$					
	ROS < APP	ROS < AFF	ROS < UNI	APP < AFF	APP < UNI	AFF < UNI
commons-cli-1	**0.04**	**0.08**	**0.08**	0.86	0.59	**0.29**
commons-cli-2	**0.21**	**0.29**	**0.00**	0.73	**0.43**	**0.11**
commons-cli-3	**0.10**	**0.19**	**0.16**	0.82	0.69	**0.32**
commons-csv-1	**0.11**	**0.12**	**0.15**	0.66	0.63	**0.45**
commons-csv-2	**0.19**	**0.42**	0.54	0.73	0.82	0.61
commons-csv-3	**0.15**	**0.25**	**0.12**	0.64	**0.48**	**0.29**
guava-escape-1	**0.39**	**0.22**	**0.38**	**0.34**	0.57	0.65
guava-escape-2	**0.28**	**0.30**	0.53	0.63	0.76	0.75
guava-escape-3	**0.04**	0.52	**0.39**	0.92	0.88	**0.39**
guava-net-1	**0.34**	**0.15**	**0.09**	**0.37**	**0.27**	**0.19**
guava-net-2	**0.42**	**0.24**	**0.21**	**0.44**	**0.37**	**0.39**
guava-net-3	**0.14**	**0.42**	**0.41**	0.87	0.77	**0.40**