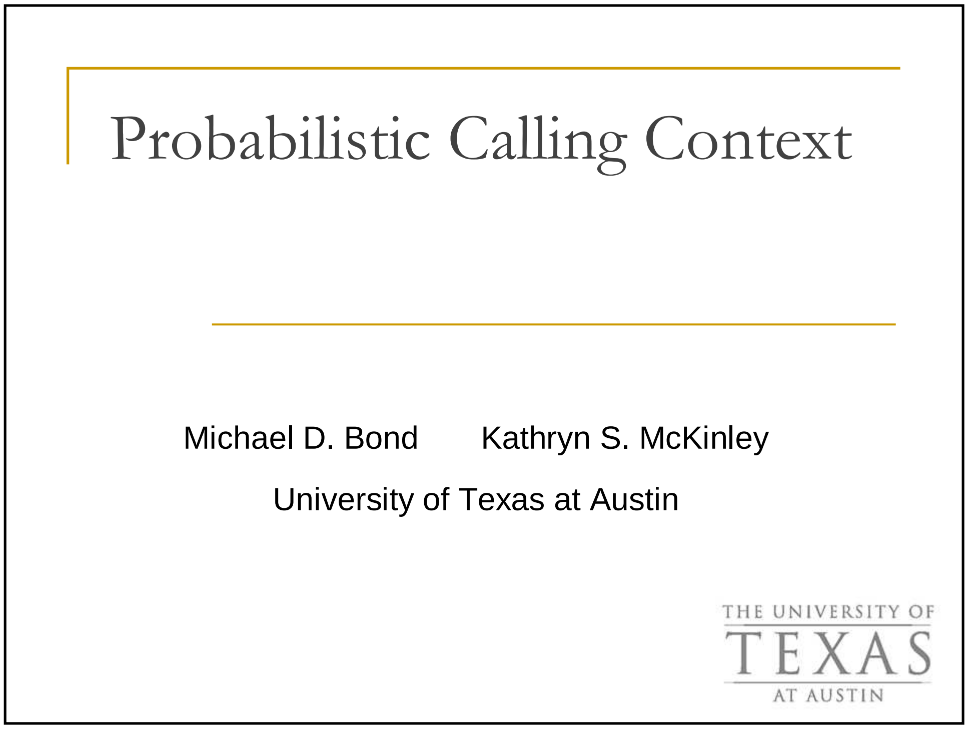
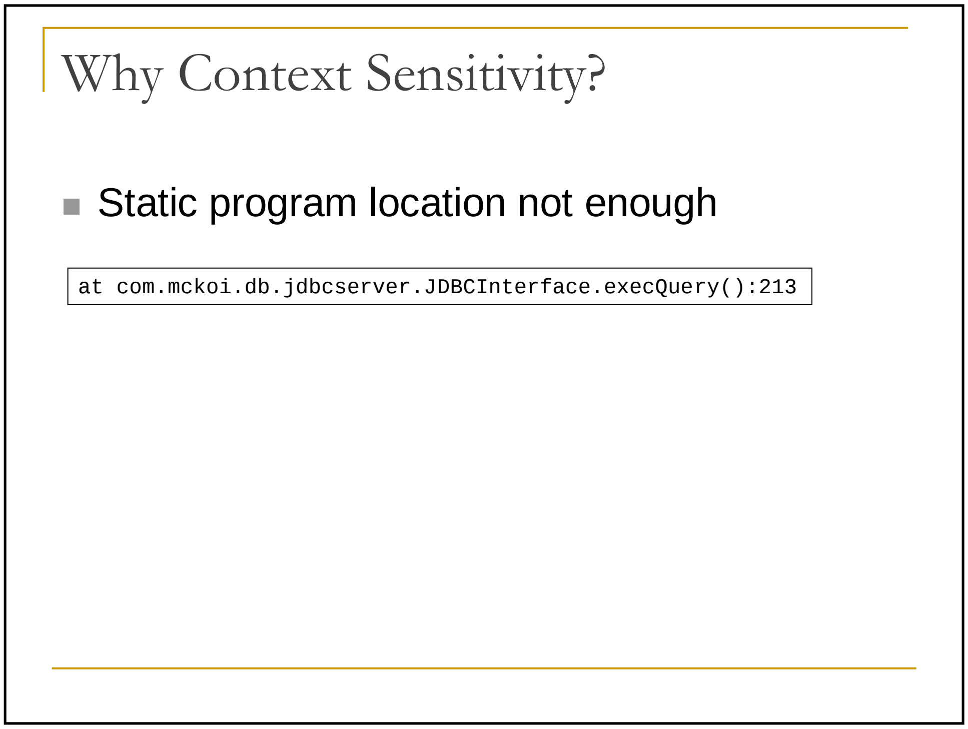


Probabilistic Calling Context

Michael D. Bond Kathryn S. McKinley

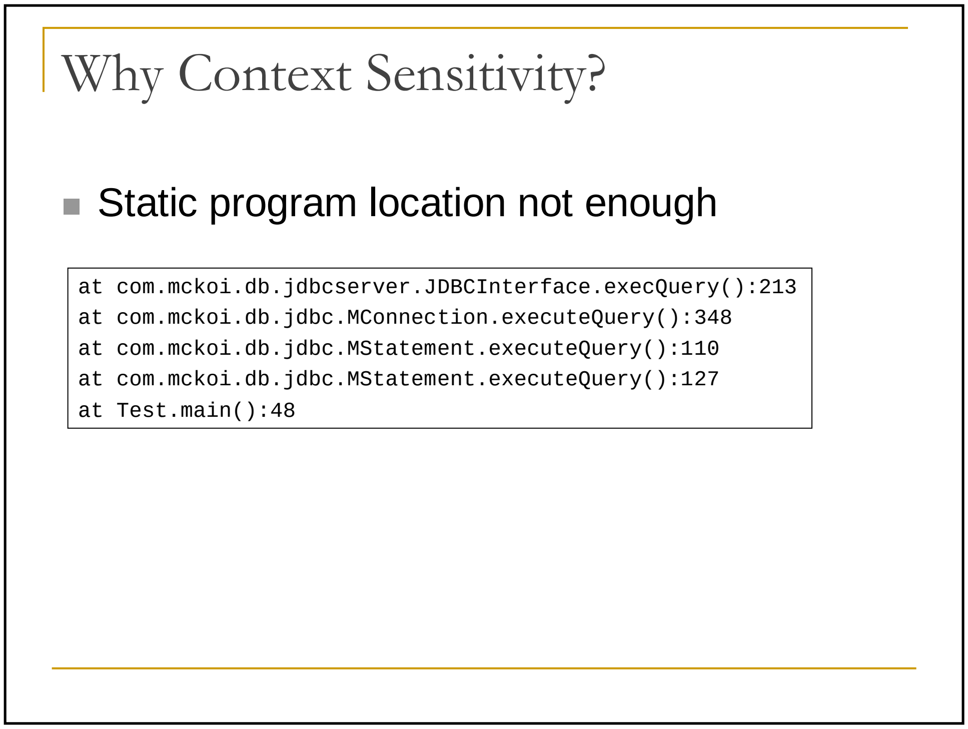
University of Texas at Austin



Why Context Sensitivity?

- Static program location not enough

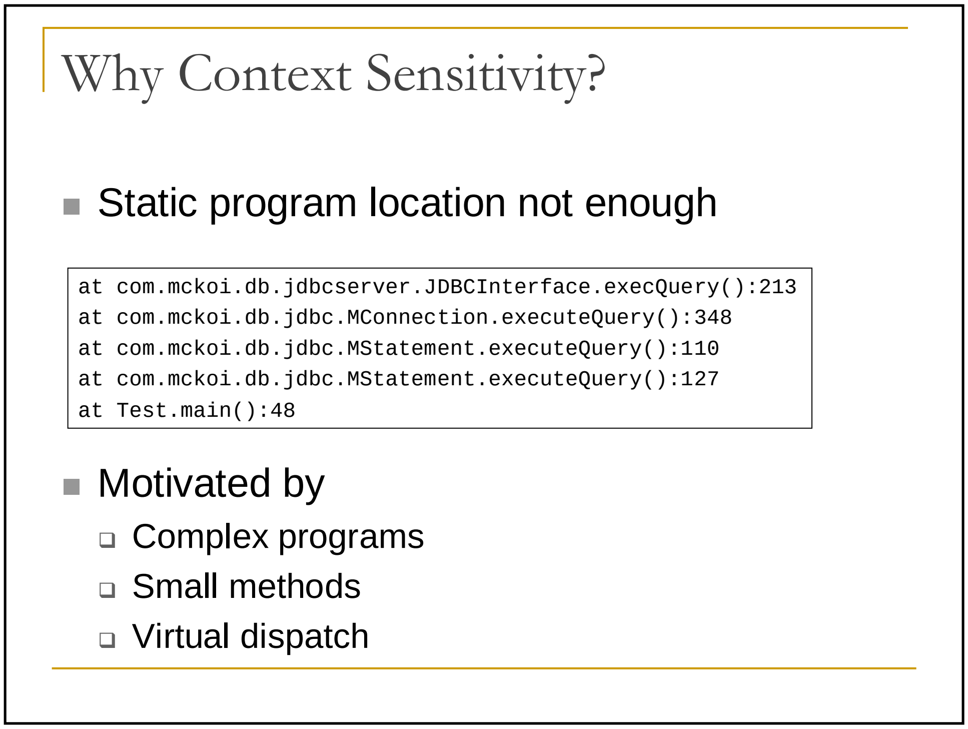
```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
```



Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48
```

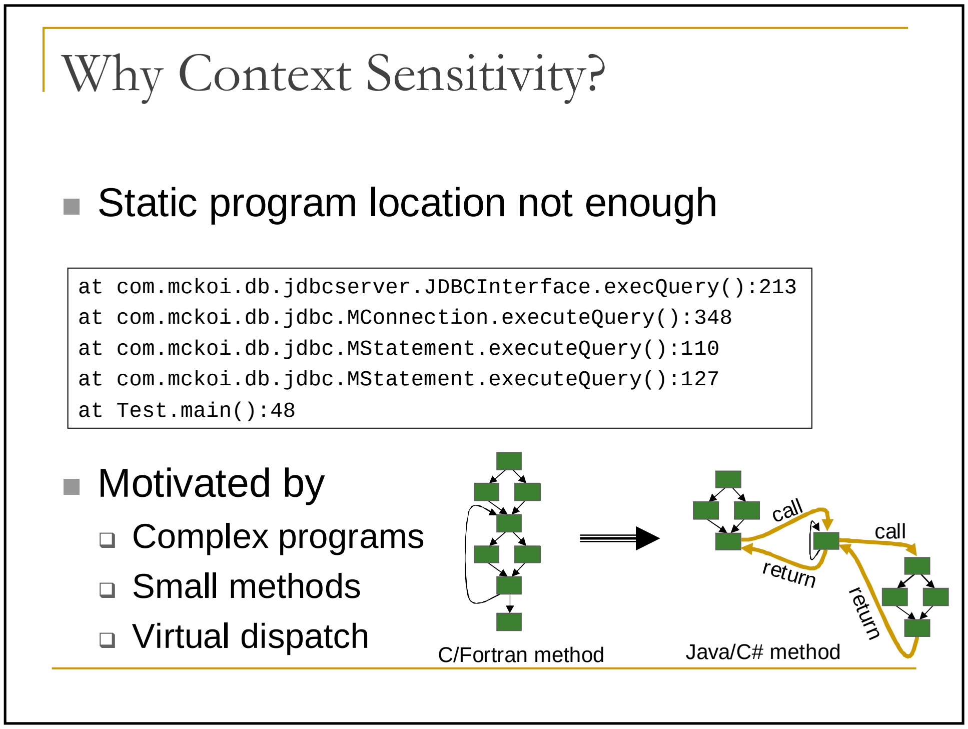


Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48
```

- Motivated by
  - Complex programs
  - Small methods
  - Virtual dispatch

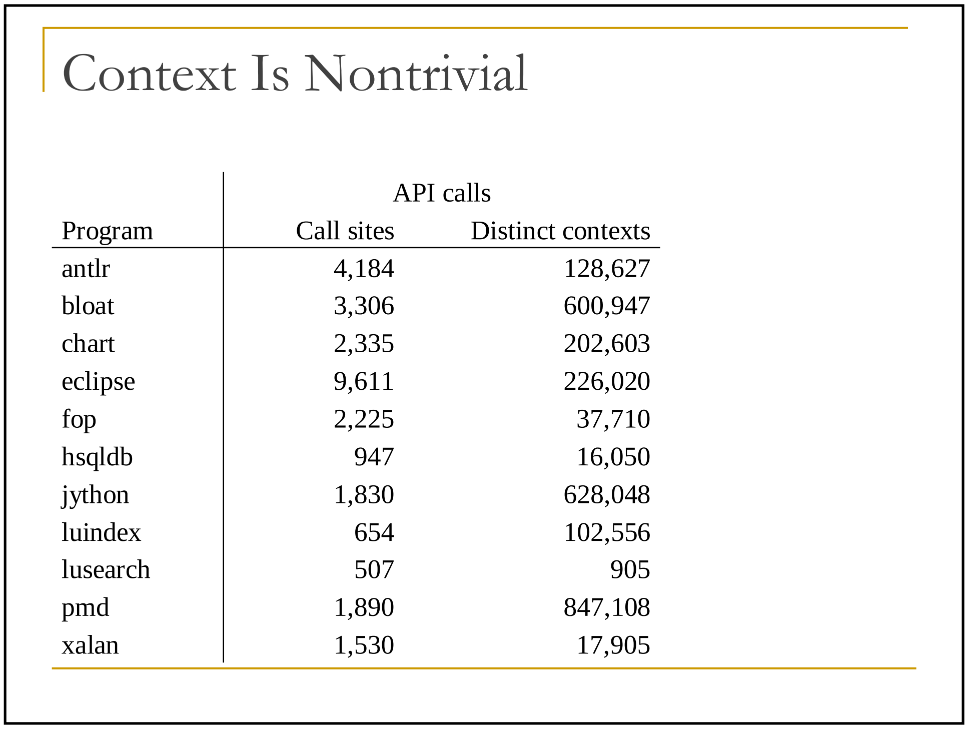


Why Context Sensitivity?

- Static program location not enough

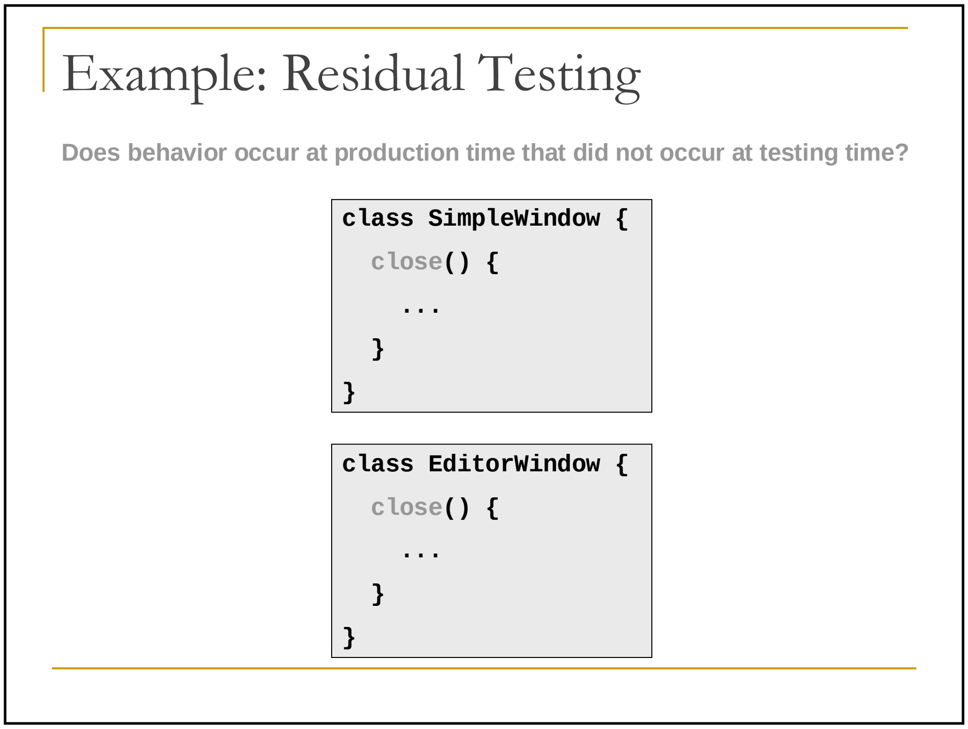
```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48
```

- Motivated by
  - Complex programs
  - Small methods
  - Virtual dispatch



Context Is Nontrivial

Program	API calls	
	Call sites	Distinct contexts
antlr	4,184	128,627
bloat	3,306	600,947
chart	2,335	202,603
eclipse	9,611	226,020
fop	2,225	37,710
hsqldb	947	16,050
jython	1,830	628,048
luindex	654	102,556
lusearch	507	905
pmd	1,890	847,108
xalan	1,530	17,905

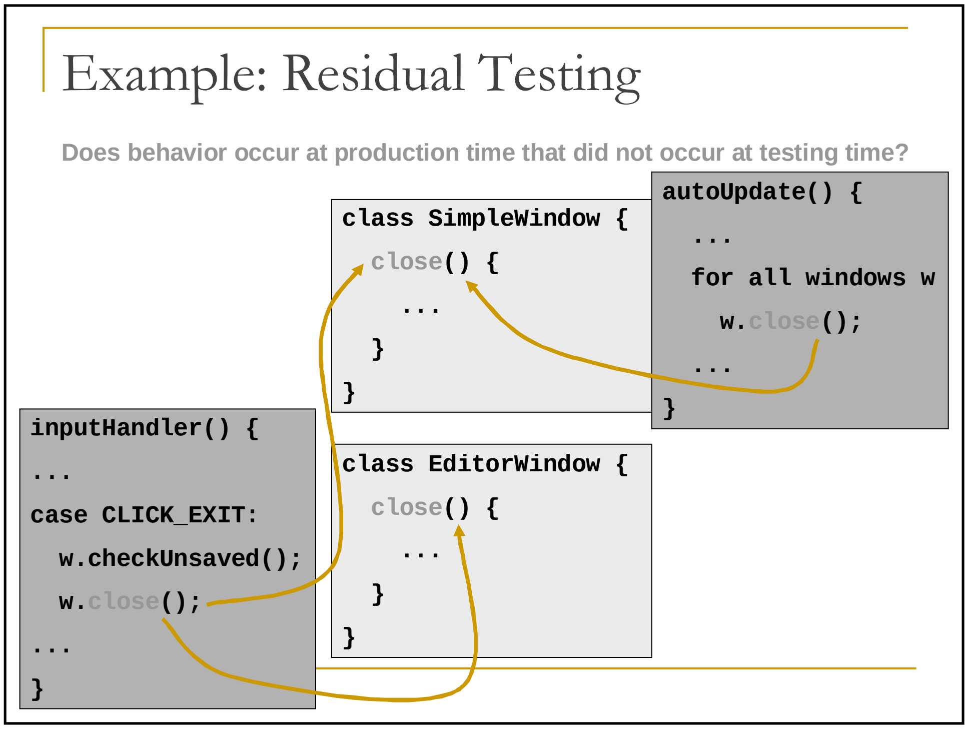


Example: Residual Testing

Does behavior occur at production time that did not occur at testing time?

```
class SimpleWindow {
  close() {
    ...
  }
}
```

```
class EditorWindow {
  close() {
    ...
  }
}
```



Example: Residual Testing

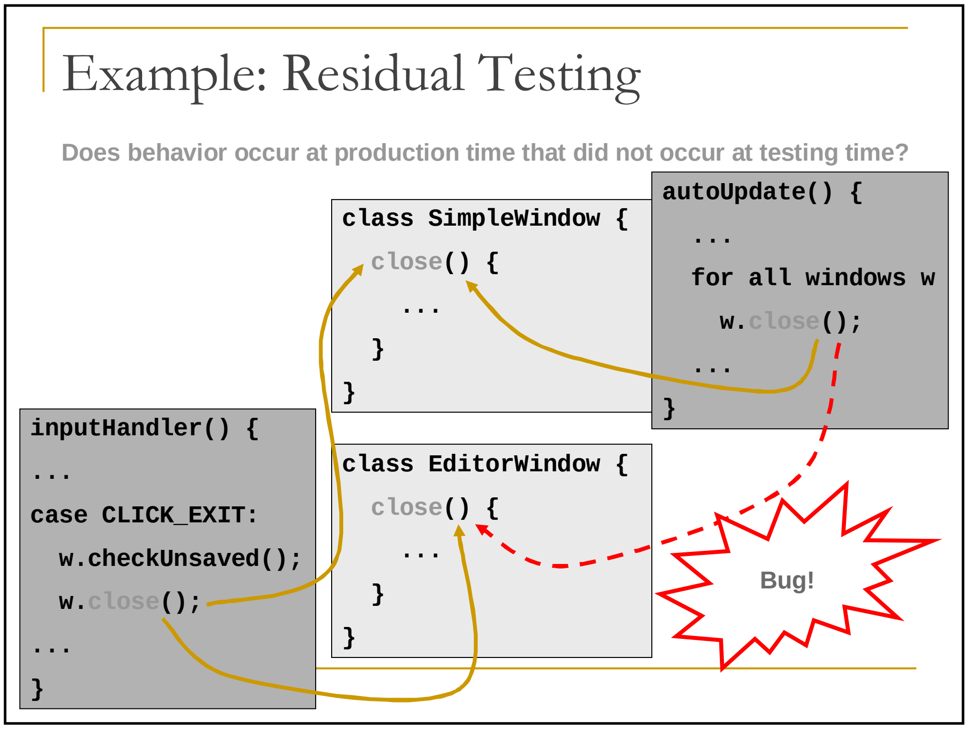
Does behavior occur at production time that did not occur at testing time?

```
class SimpleWindow {
  close() {
    ...
  }
}
```

```
autoUpdate() {
  ...
  for all windows w
    w.close();
  ...
}
```

```
inputHandler() {
...
case CLICK_EXIT:
  w.checkUnsaved();
  w.close();
...
}
```

```
class EditorWindow {
  close() {
    ...
  }
}
```



Example: Residual Testing

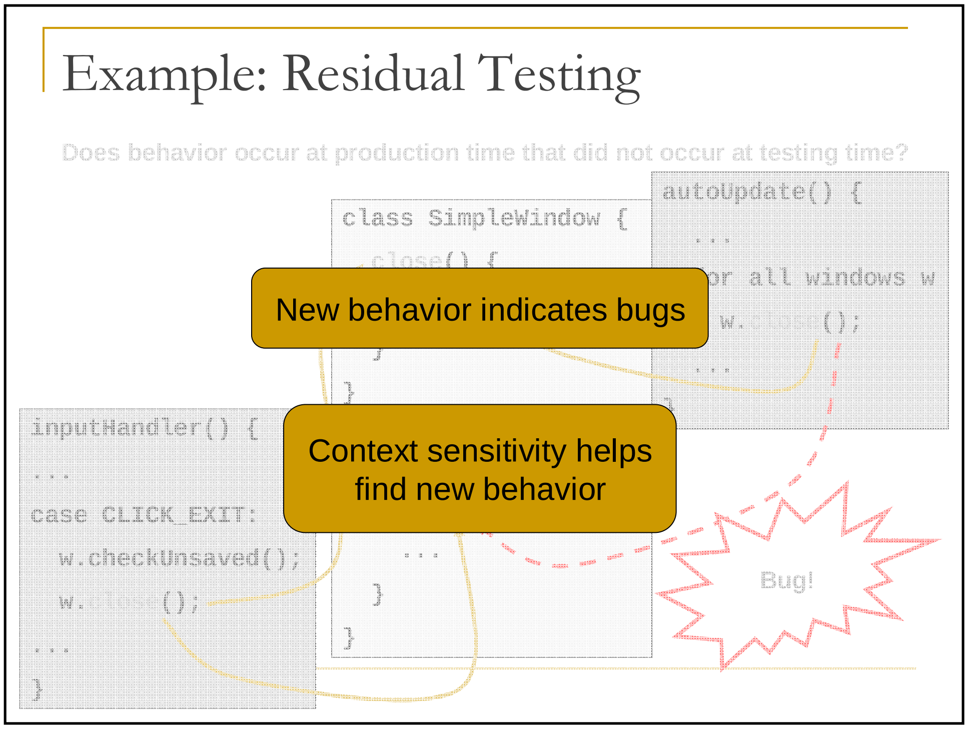
Does behavior occur at production time that did not occur at testing time?

```
class SimpleWindow {
  close() {
    ...
  }
}
```

```
autoUpdate() {
  ...
  for all windows w
    w.close();
  ...
}
```

```
inputHandler() {
...
case CLICK_EXIT:
  w.checkUnsaved();
  w.close();
...
}
```

```
class EditorWindow {
  close() {
    ...
  }
}
```

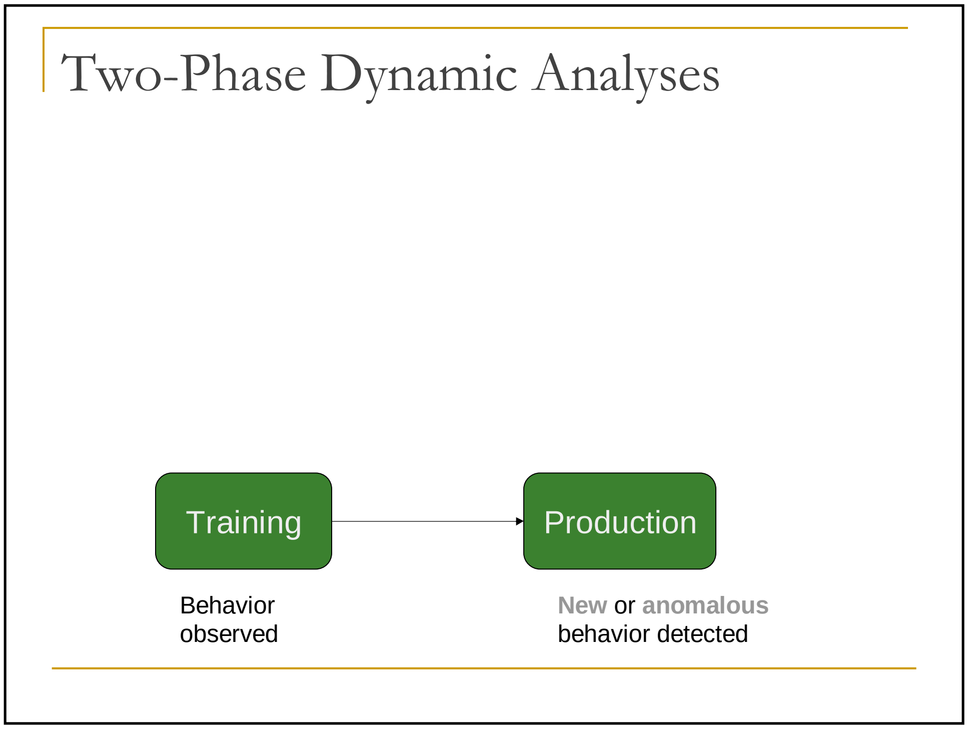


Example: Residual Testing

Does behavior occur at production time that did not occur at testing time?

New behavior indicates bugs

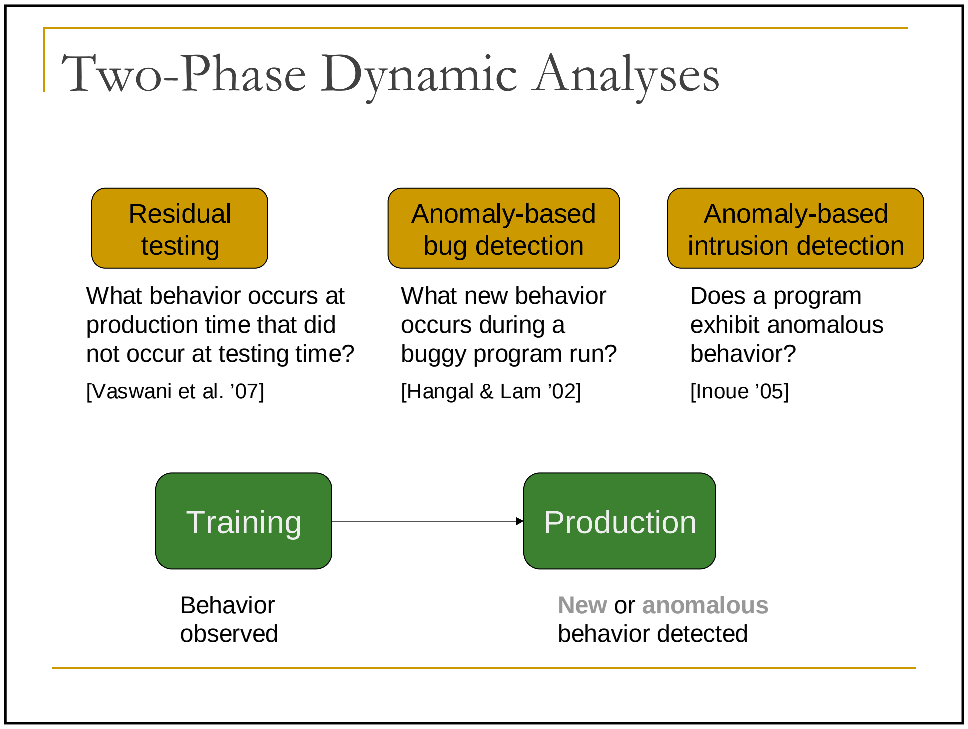
Context sensitivity helps find new behavior



Two-Phase Dynamic Analyses

Behavior
observed

New or anomalous
behavior detected



Two-Phase Dynamic Analyses

Residual testing
Anomaly-based bug detection
Anomaly-based intrusion detection
Training
Production

Residual testing

What behavior occurs at production time that did not occur at testing time?

[Vaswani et al. '07]

Anomaly-based bug detection

What new behavior occurs during a buggy program run?

[Hangal & Lam '02]

Anomaly-based intrusion detection

Does a program exhibit anomalous behavior?

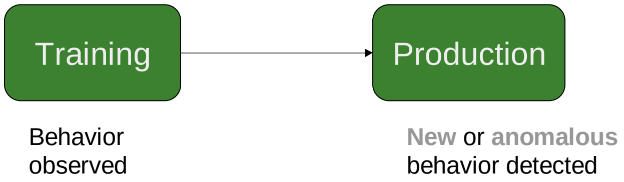
[Inoue '05]

Behavior
observed

New or anomalous
behavior detected

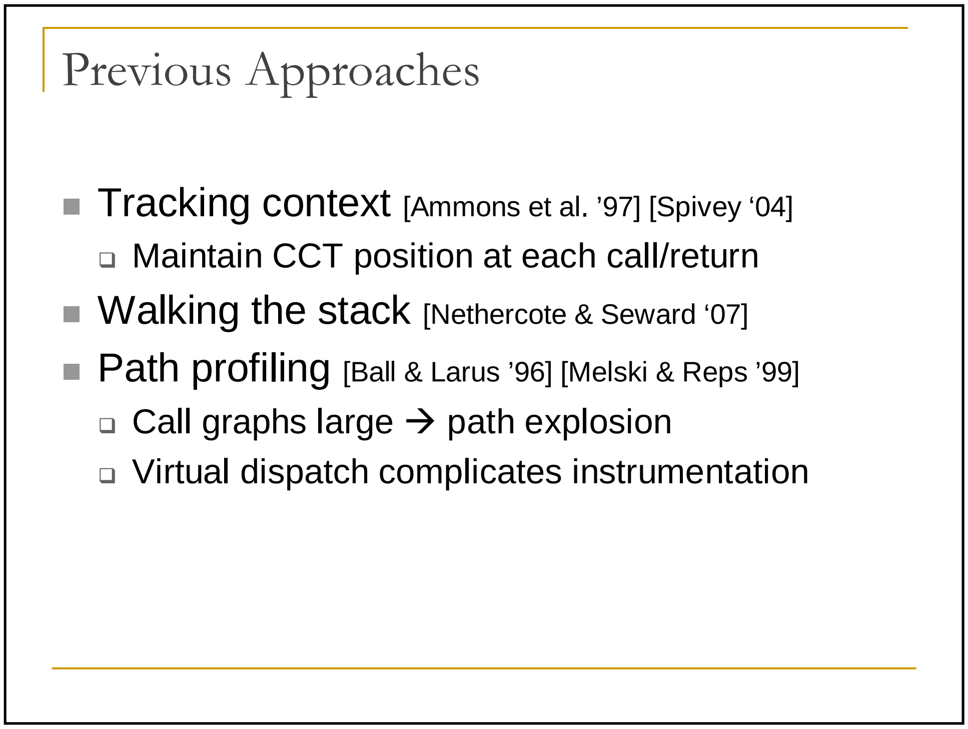
Probabilistic Calling Context

- Adds context sensitivity to dynamic analyses
- Maintains value representing context
  - Unique with high probability
  - New value → new context → walk stack
- High accuracy: <0.1% false negatives
- Low overhead: 3% overhead, 0-8% for clients



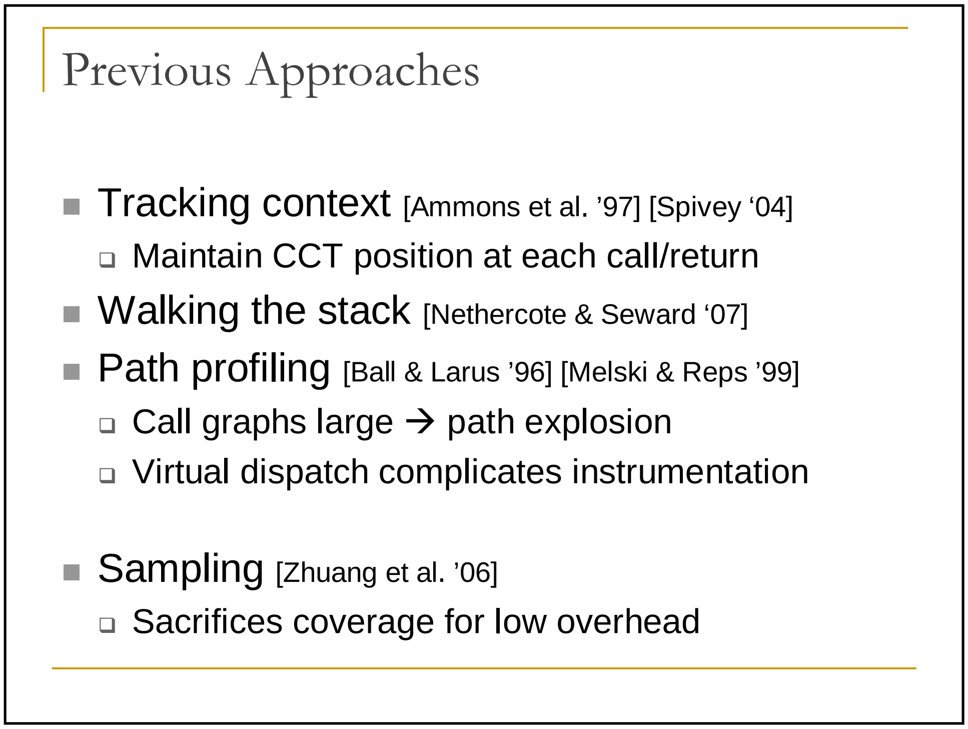
Outline

- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Accuracy
  - Performance



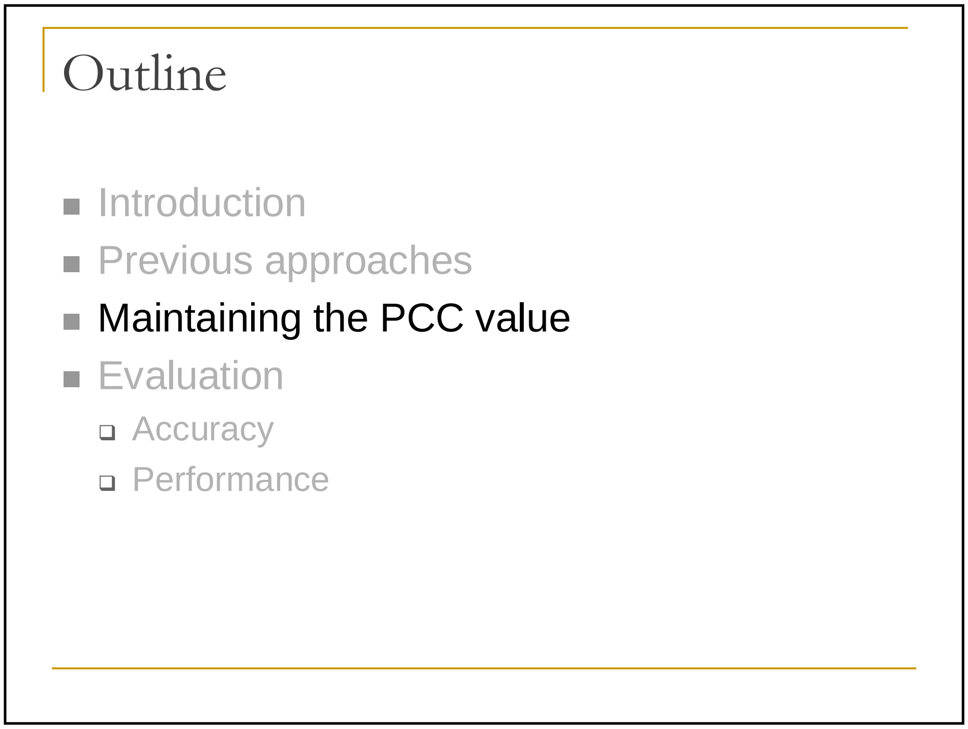
Previous Approaches

- Tracking context [Ammons et al. ’97] [Spivey ‘04]
  - Maintain CCT position at each call/return
- Walking the stack [Nethercote & Seward ‘07]
- Path profiling [Ball & Larus ’96] [Melski & Reps ’99]
  - Call graphs large → path explosion
  - Virtual dispatch complicates instrumentation



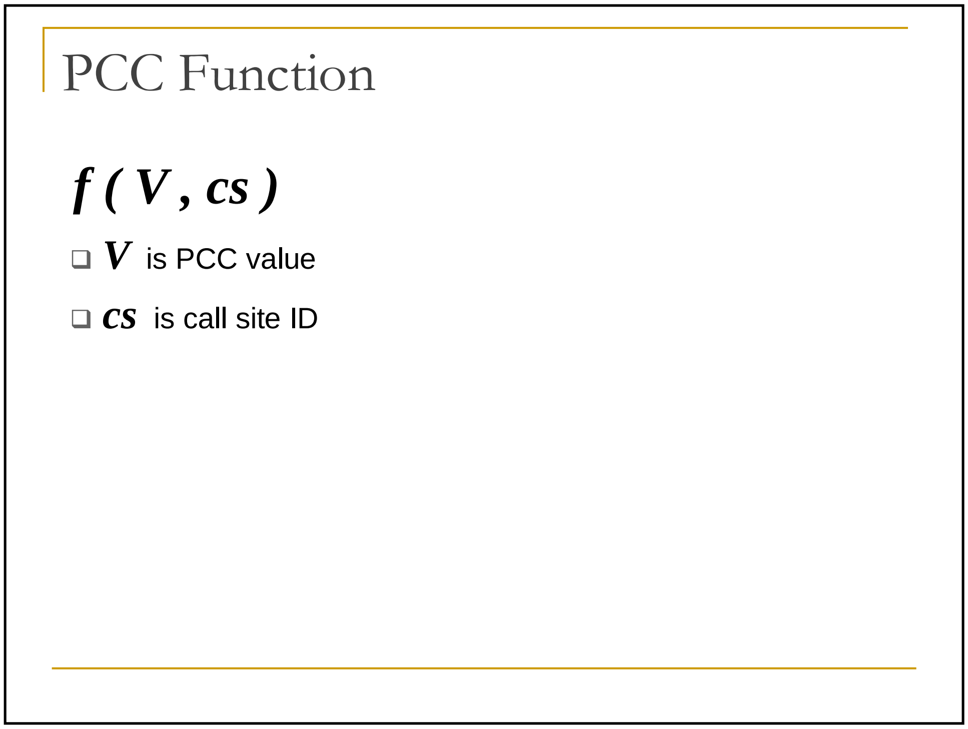
Previous Approaches

- Tracking context [Ammons et al. ’97] [Spivey ‘04]
  - Maintain CCT position at each call/return
- Walking the stack [Nethercote & Seward ‘07]
- Path profiling [Ball & Larus ’96] [Melski & Reps ’99]
  - Call graphs large → path explosion
  - Virtual dispatch complicates instrumentation
- Sampling [Zhuang et al. ’06]
  - Sacrifices coverage for low overhead



Outline

- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Accuracy
  - Performance



PCC Function

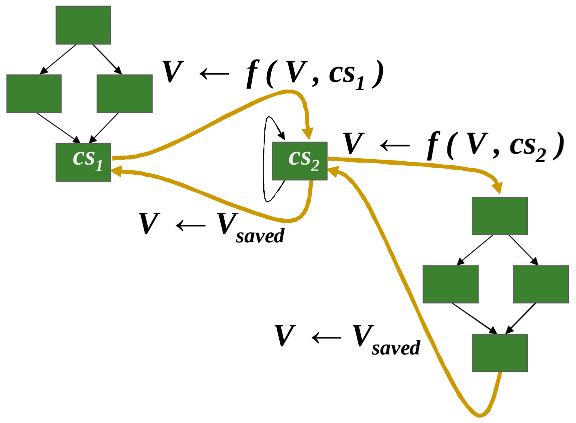
$$f\,(\,V\,,\,cs\,)$$

- ❑ V is PCC value
- ❑ cs is call site ID

PCC Function

$$f\,(\,V\,,\,cs\,)$$

- ❑ V is PCC value
- ❑ cs is call site ID


$V \leftarrow f\,(\,V\,,\,cs_1\,)$
cs_1
cs_2
$V \leftarrow f\,(\,V\,,\,cs_2\,)$
$V \leftarrow V_{saved}$
$V \leftarrow V_{saved}$

PCC Function

$$f\,(\,V\,,\,cs\,) \;\equiv\; 3V + cs \pmod{2^{32}}$$

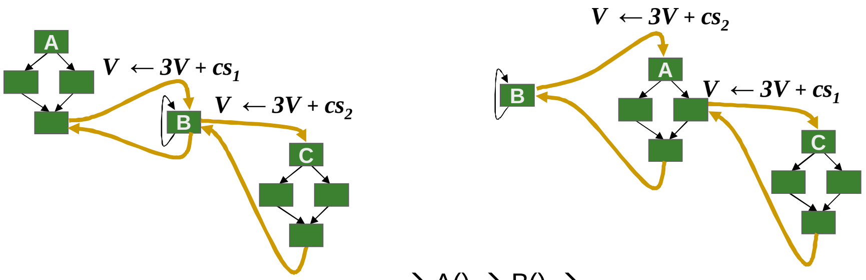
- ❑ V is PCC value
- ❑ cs is call site ID

PCC Function

$$f\,(\,V\,,\,cs\,)\ \equiv\ 3V + cs\ \ (mod\ 2^{32})$$

- Motivated by MPI datatype hashing
  [Langou et al. ’05] [Gropp ’00]
- Cheap to compute
- Desirable properties:
  - Non-commutative
  - Composition efficient to compute

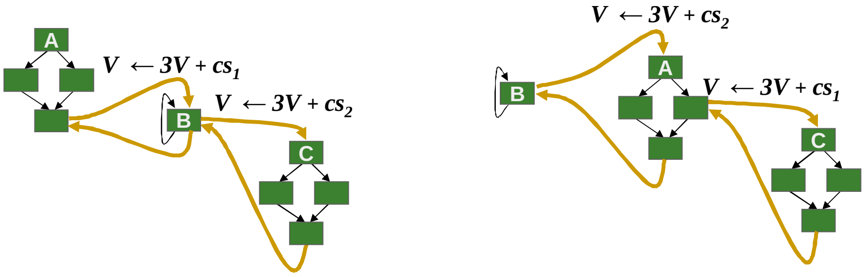
Differentiating Similar Contexts



… → A() → B() → …

… → B() → A() → …

Differentiating Similar Contexts



$V \leftarrow 3V + cs_1$

$V \leftarrow 3V + cs_2$

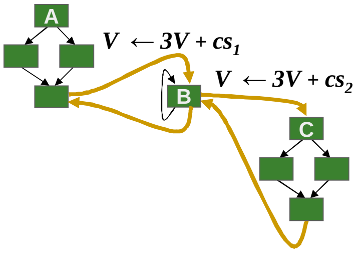
$V \leftarrow 3V + cs_2$

$V \leftarrow 3V + cs_1$

- Non-commutative

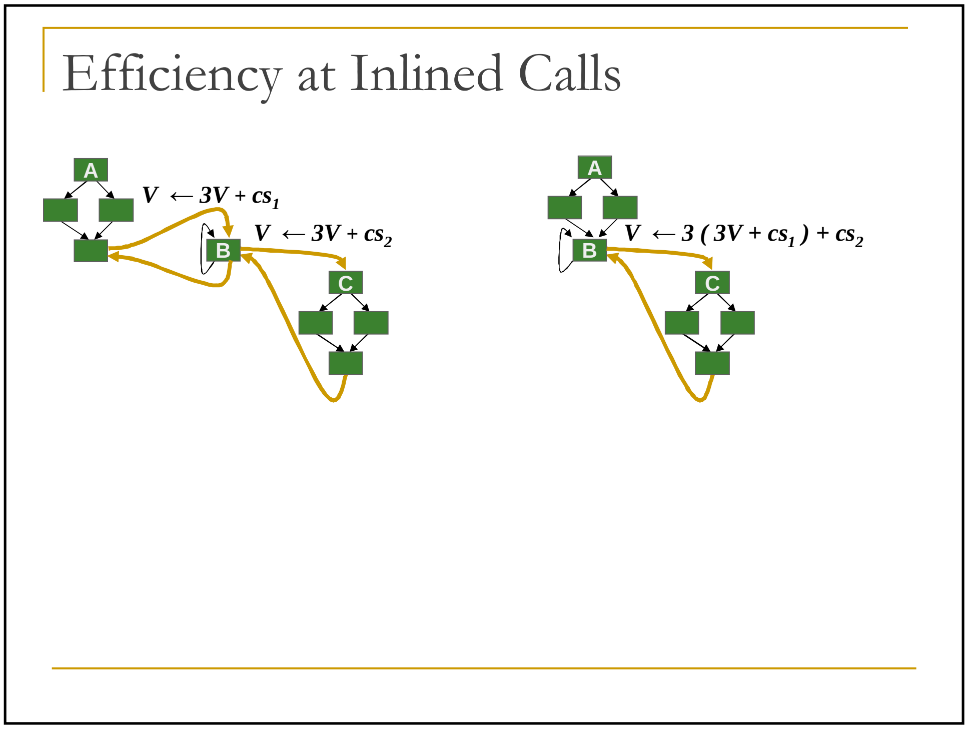
$$f\,(\,f\,(V\,,\,cs_1\,)\,,\,cs_2\,) \neq$$
$$f\,(\,f\,(V\,,\,cs_2\,)\,,\,cs_1\,)$$

Efficiency at Inlined Calls



$V \leftarrow 3V + cs_1$

$V \leftarrow 3V + cs_2$

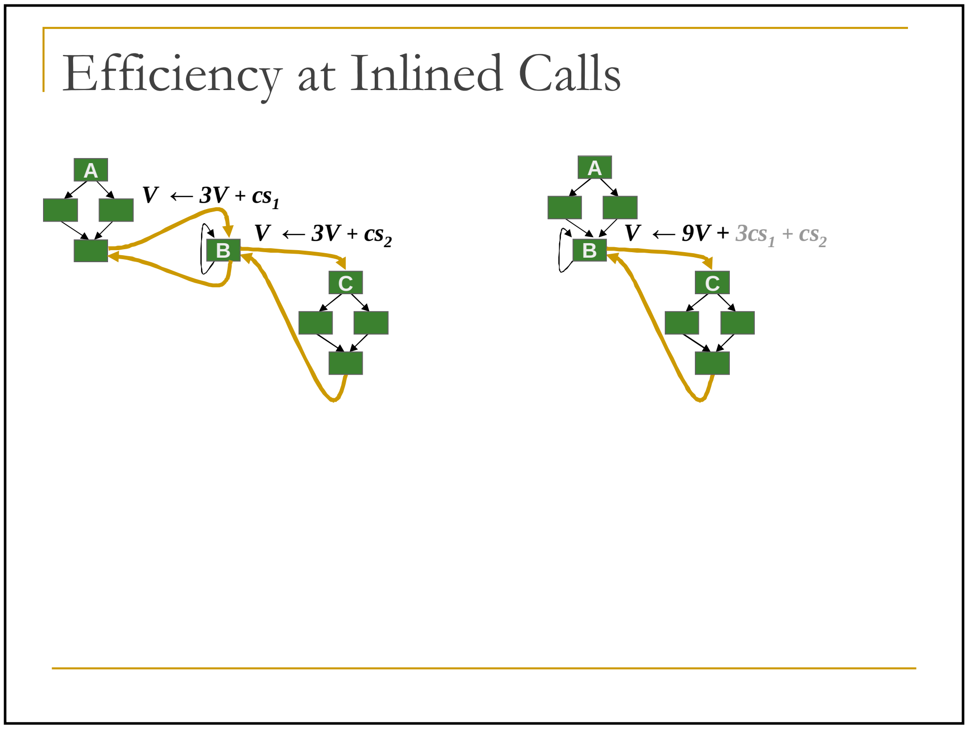


Efficiency at Inlined Calls

$V \leftarrow 3V + cs_1$

$V \leftarrow 3V + cs_2$

$V \leftarrow 3\,(\,3V + cs_1\,) + cs_2$

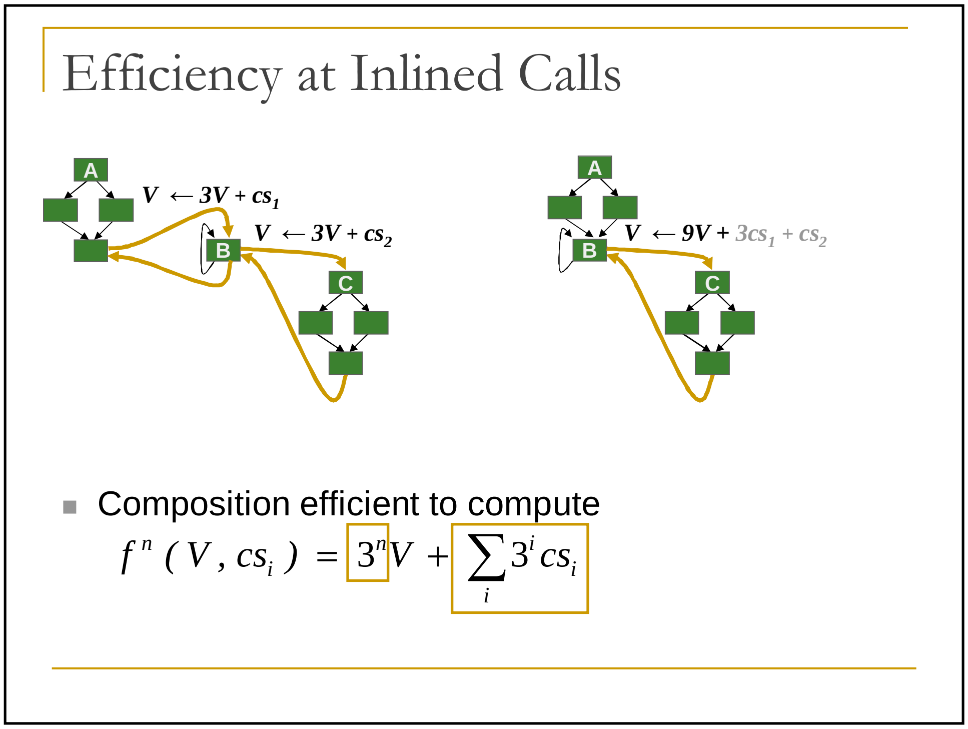


Efficiency at Inlined Calls

$V \leftarrow 3V + cs_1$

$V \leftarrow 3V + cs_2$

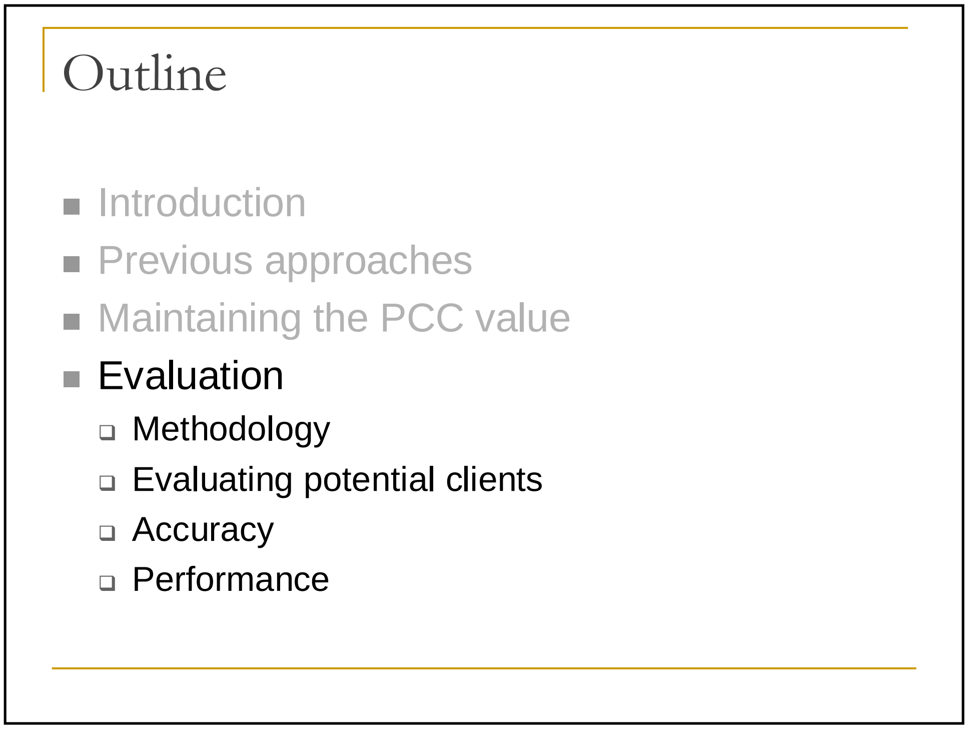
$V \leftarrow 9V + 3cs_1 + cs_2$



Efficiency at Inlined Calls

- Composition efficient to compute

$$f^{n}(V, cs_i) = \boxed{3^{n}}V + \boxed{\sum_i 3^{i} cs_i}$$



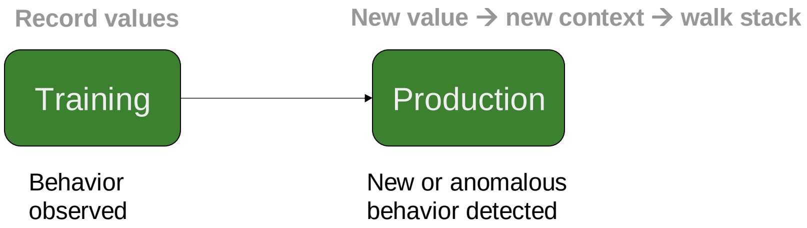
Outline

- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Methodology
  - Evaluating potential clients
  - Accuracy
  - Performance

Methodology

- Implementation in Jikes RVM 2.4.6
  - Available on Jikes RVM Research Archive
- Deterministic calling context profiling
  - Maintains CCT node at each call & return
- Benchmarks: DaCapo, SPEC JBB2000, SPEC JVM98
- Platform: 3.6 GHz Pentium 4 w/Linux

How Clients Use PCC



Record values

New value → new context → walk stack

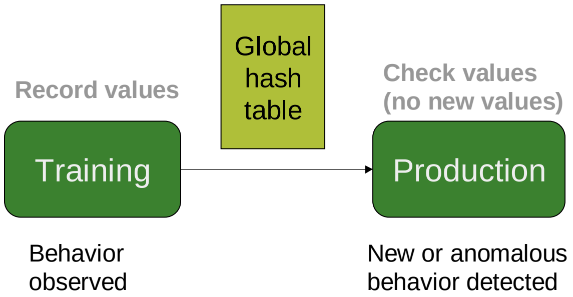
Training

Production

Behavior observed

New or anomalous behavior detected

Evaluating Potential Clients



Global hash table

Record values

Training

Behavior observed

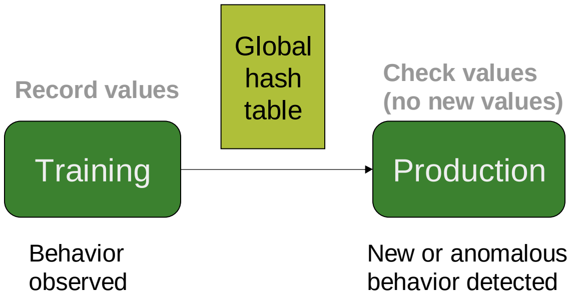
Check values (no new values)

Production

New or anomalous behavior detected

Evaluating Potential Clients

Memory overhead:
proportional to contexts



Global hash table

Record values

Training

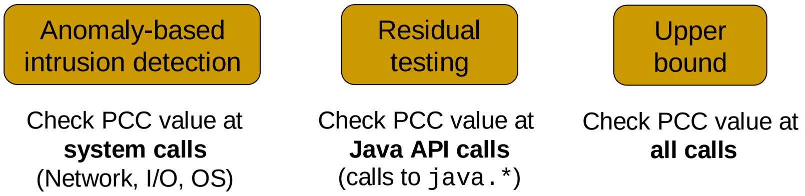
Behavior observed

Check values (no new values)

Production

New or anomalous behavior detected

Evaluating Potential Clients



Anomaly-based intrusion detection

Check PCC value at **system calls** (Network, I/O, OS)

Residual testing

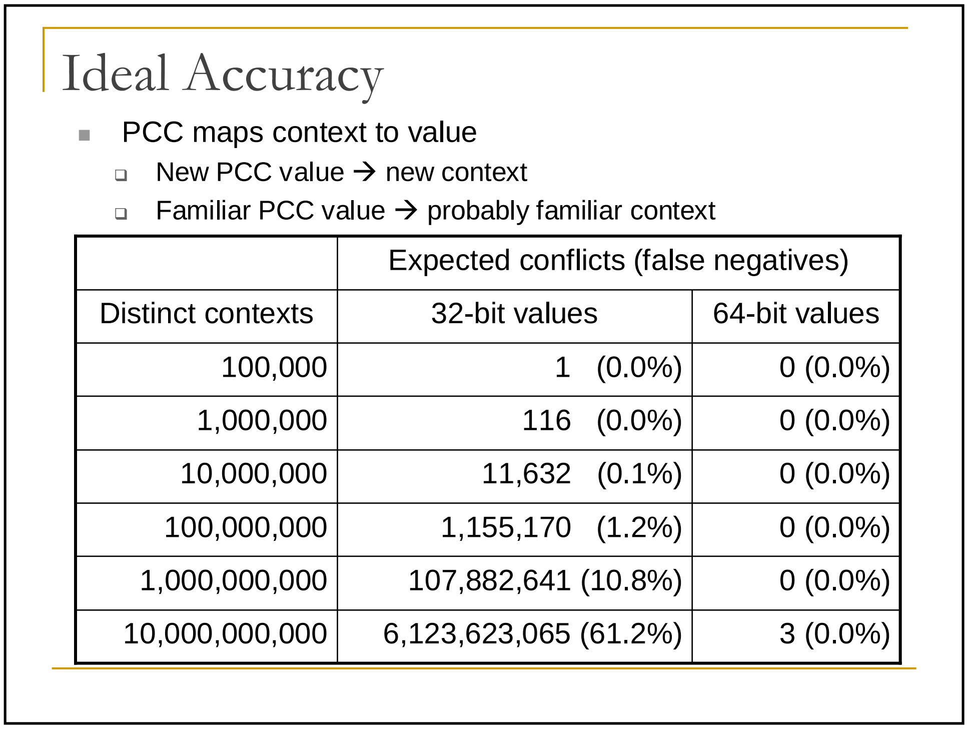
Check PCC value at **Java API calls** (calls to `java.*`)

Upper bound

Check PCC value at **all calls**

Ideal Accuracy

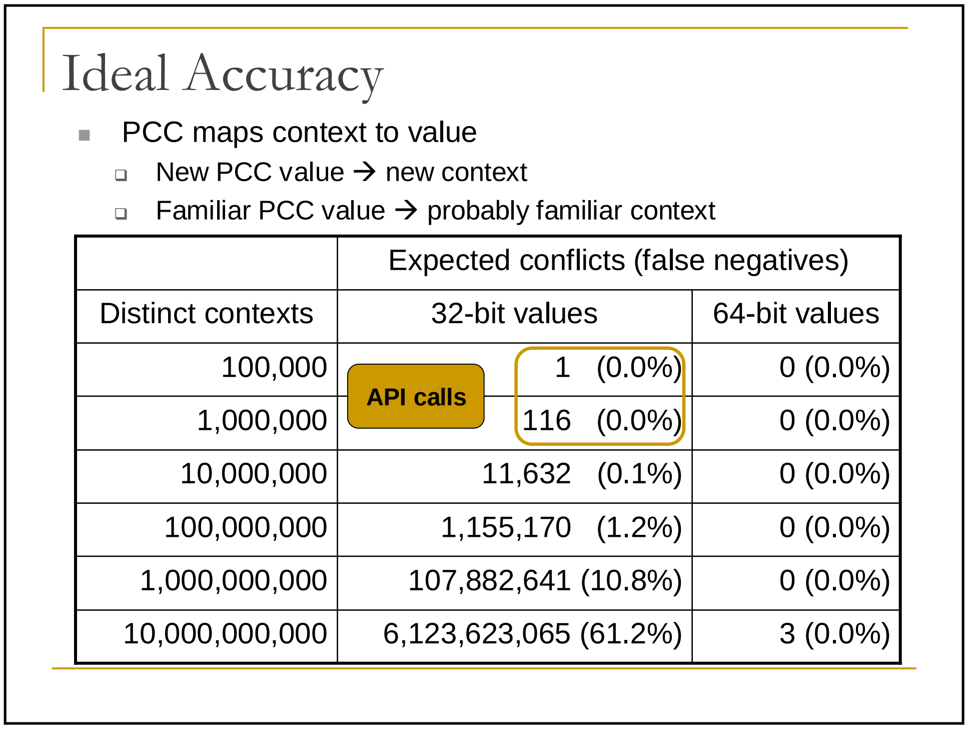
- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context



Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,632 (0.1%)	0 (0.0%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

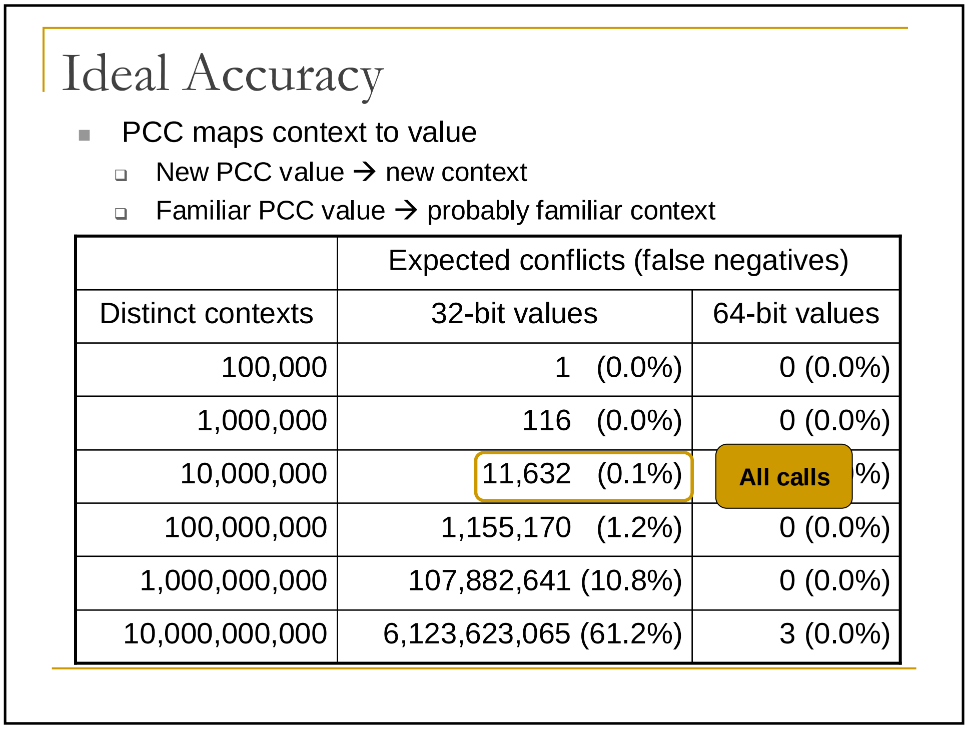


Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,632 (0.1%)	0 (0.0%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

API calls

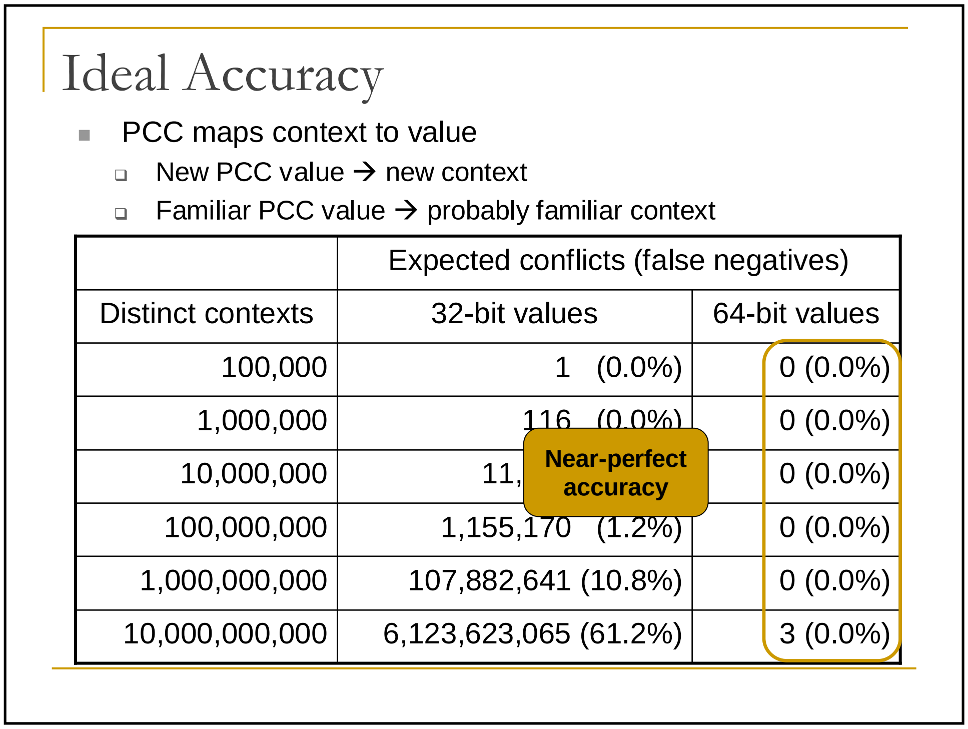


Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,632 (0.1%)	%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

All calls

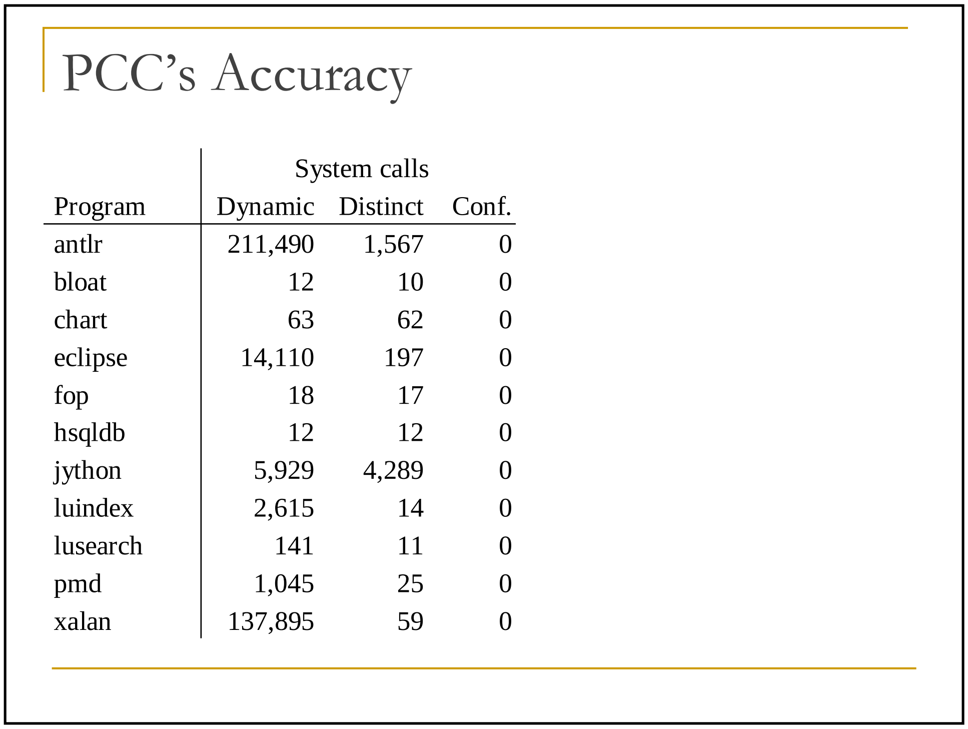


Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

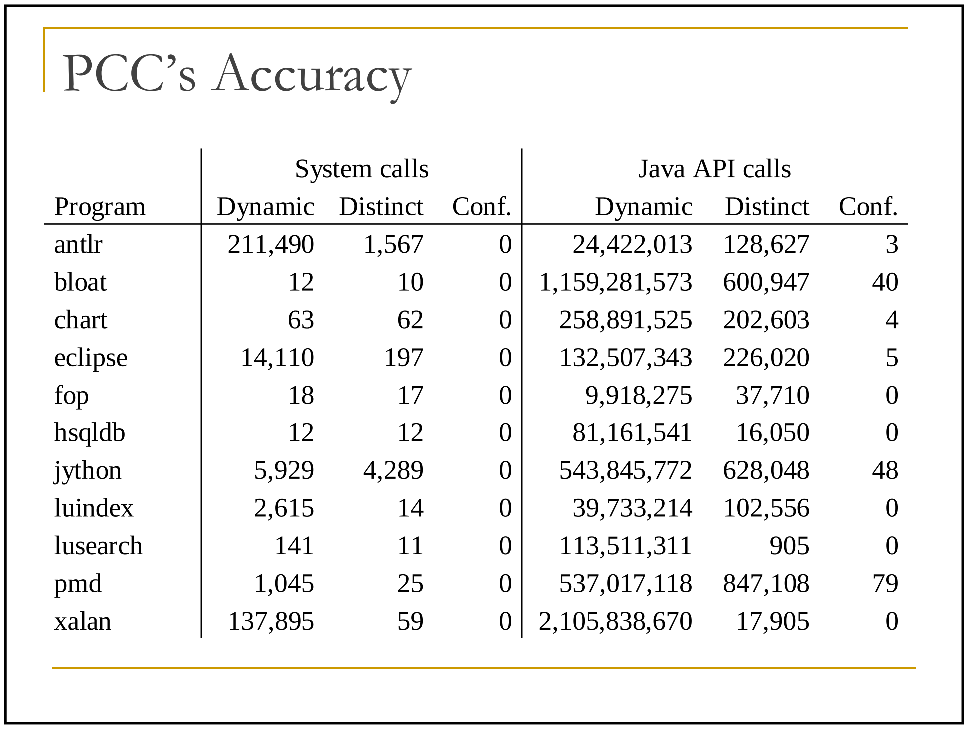
	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,	0 (0.0%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

Near-perfect accuracy



PCC's Accuracy

Program	System calls Dynamic	 Distinct	 Conf.
antlr	211,490	1,567	0
bloat	12	10	0
chart	63	62	0
eclipse	14,110	197	0
fop	18	17	0
hsqldb	12	12	0
jython	5,929	4,289	0
luindex	2,615	14	0
lusearch	141	11	0
pmd	1,045	25	0
xalan	137,895	59	0



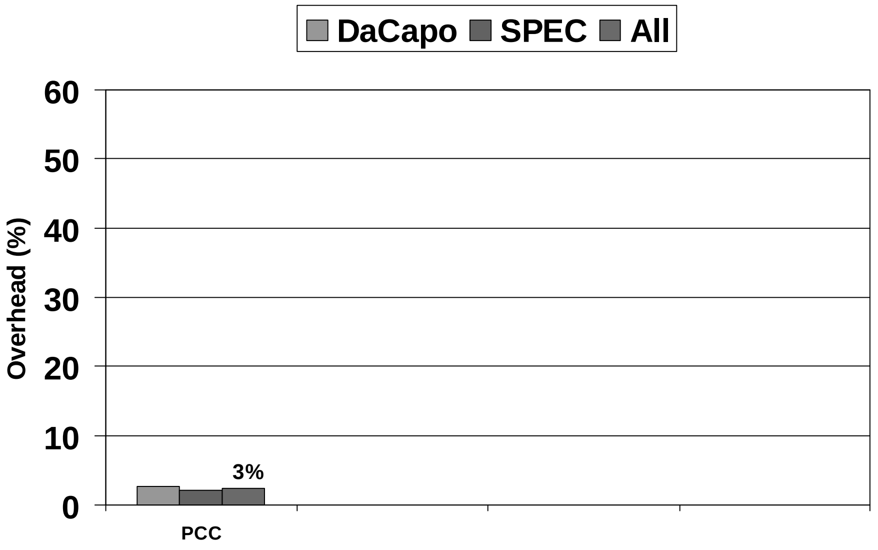
PCC's Accuracy

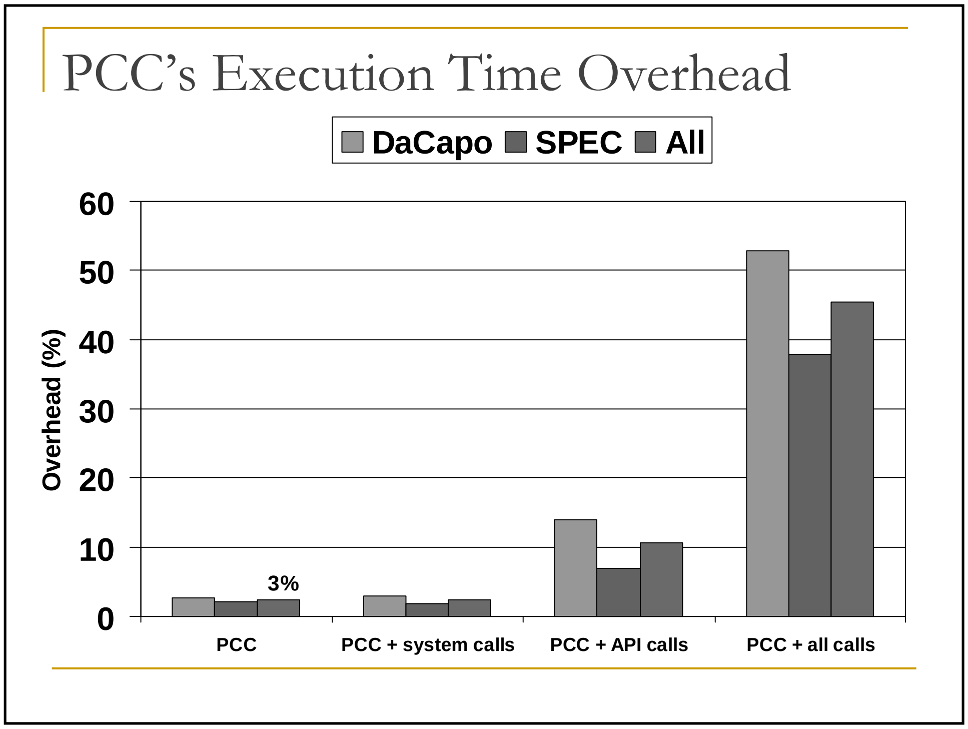
	System calls			Java API calls		
Program	Dynamic	Distinct	Conf.	Dynamic	Distinct	Conf.
antlr	211,490	1,567	0	24,422,013	128,627	3
bloat	12	10	0	1,159,281,573	600,947	40
chart	63	62	0	258,891,525	202,603	4
eclipse	14,110	197	0	132,507,343	226,020	5
fop	18	17	0	9,918,275	37,710	0
hsqldb	12	12	0	81,161,541	16,050	0
jython	5,929	4,289	0	543,845,772	628,048	48
luindex	2,615	14	0	39,733,214	102,556	0
lusearch	141	11	0	113,511,311	905	0
pmd	1,045	25	0	537,017,118	847,108	79
xalan	137,895	59	0	2,105,838,670	17,905	0

PCC’s Accuracy

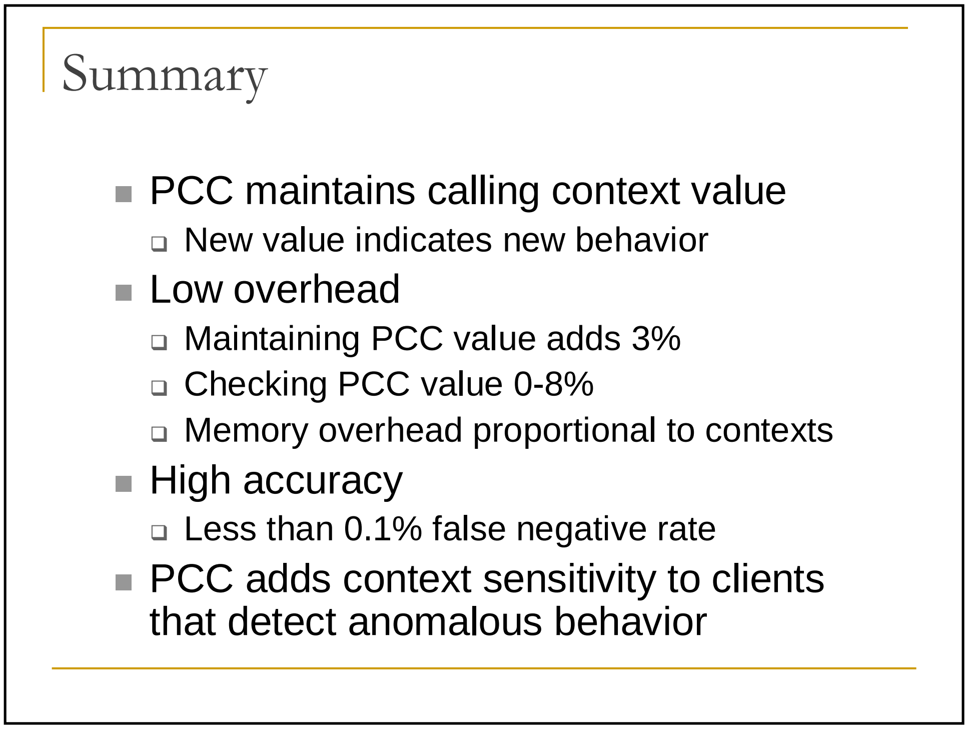
Program	All calls Dynamic	 Distinct	 Conf.
antlr	490,363,211	1,006,578	118
bloat	6,276,446,059	1,980,205	453
chart	908,459,469	845,432	91
eclipse	1,266,810,504	4,815,901	2,652
fop	44,200,446	174,955	2
hsqldb	877,680,667	110,795	1
jython	5,326,949,158	3,859,545	1,738
luindex	740,053,104	374,201	12
lusearch	1,439,034,336	6,039	0
pmd	2,726,876,957	8,043,096	7,653
xalan	10,083,858,546	163,205	6

PCC’s Execution Time Overhead



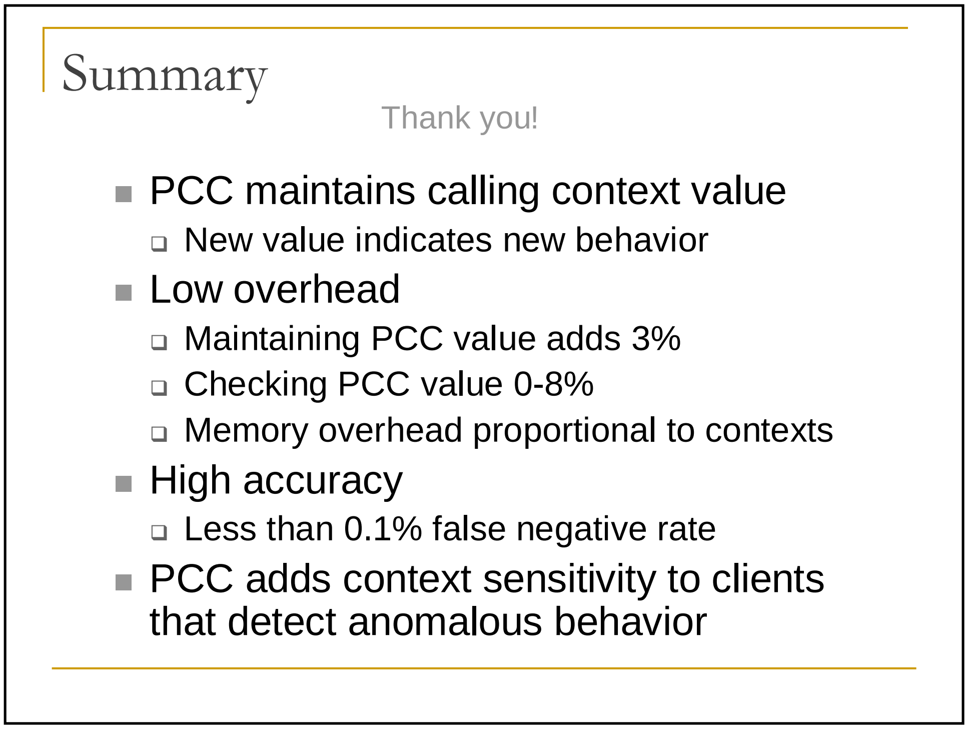


PCC's Execution Time Overhead



Summary

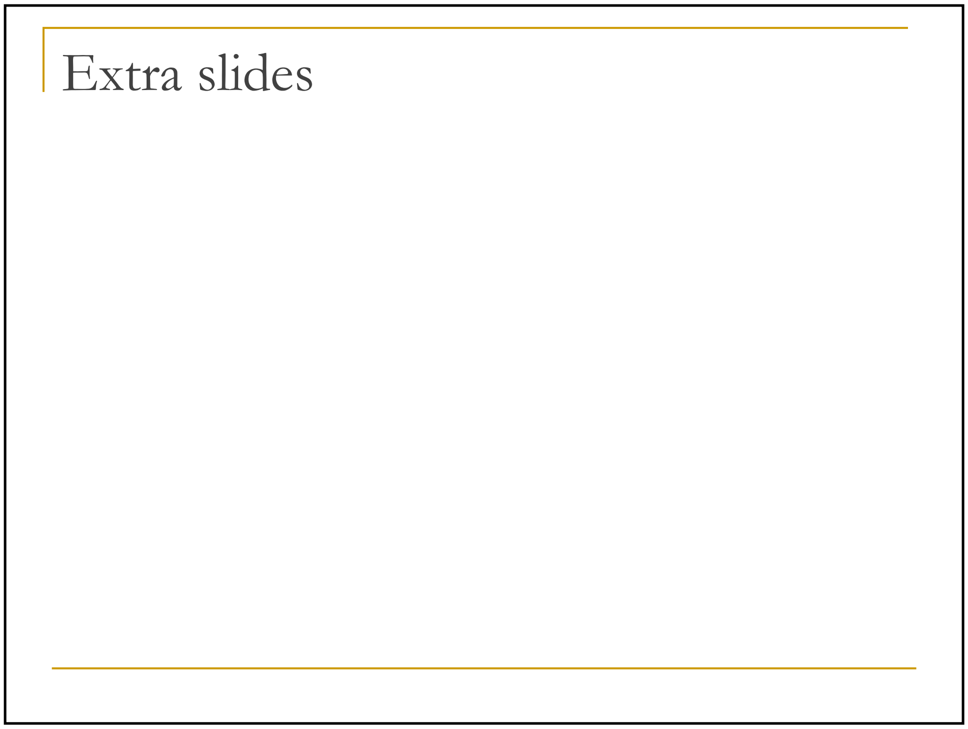
- PCC maintains calling context value
  - New value indicates new behavior
- Low overhead
  - Maintaining PCC value adds 3%
  - Checking PCC value 0-8%
  - Memory overhead proportional to contexts
- High accuracy
  - Less than 0.1% false negative rate
- PCC adds context sensitivity to clients that detect anomalous behavior



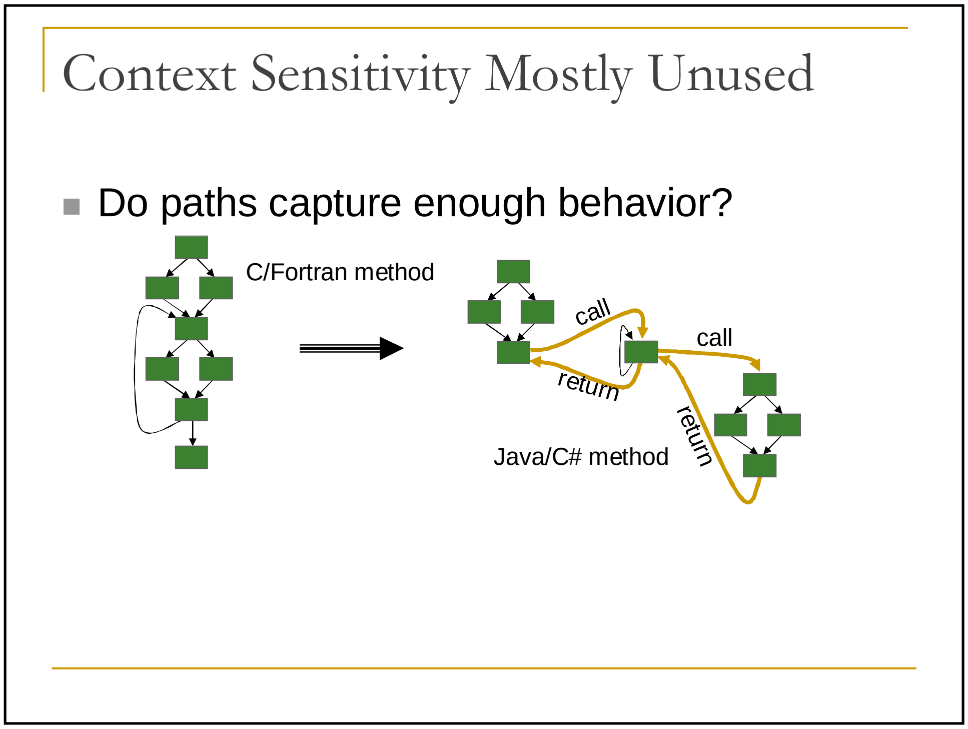
Summary

Thank you!

- PCC maintains calling context value
  - New value indicates new behavior
- Low overhead
  - Maintaining PCC value adds 3%
  - Checking PCC value 0-8%
  - Memory overhead proportional to contexts
- High accuracy
  - Less than 0.1% false negative rate
- PCC adds context sensitivity to clients that detect anomalous behavior



Extra slides



Context Sensitivity Mostly Unused

- Do paths capture enough behavior?