Precise Calling Context Encoding

Nick Sumner
Yunhui Zheng
Dasarath Weeratunge
Xiangyu Zhang

What Are Calling Contexts?

- Calling Contexts
  - Sequence of active functions on call stack
  - Precisely capture sequence of active call sites

Calling Contexts

- Calling Contexts
  - Sequence of active functions on call stack
  - Precisely capture sequence of active call sites

```
1)def a():
2)   print('Here')

3)def b():
4)   a()
5)   a()

6)def main():
7)   b()
```

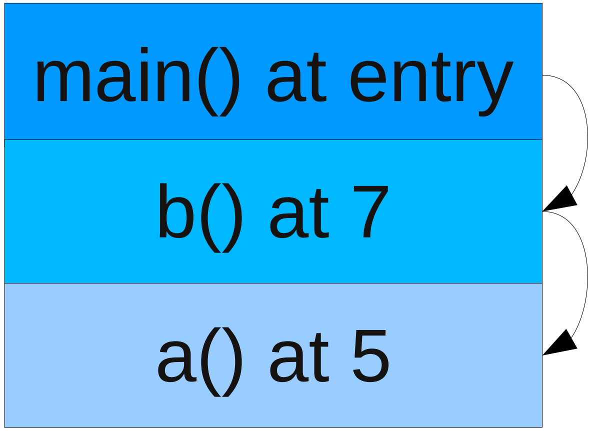
Calling Contexts

- Calling Contexts
  - Sequence of active functions on call stack
  - Precisely capture sequence of active call sites

```
1)def a():
2)   print('Here')

3)def b():
4)   a()
5)   a()

6)def main():
7)   b()
```



main() at entry

b() at 7

a() at 5

Why Calling Contexts?

- Context sensitive profiling
  - Identify subtle program behaviors

Why Calling Contexts?

- Context sensitive profiling
  - Identify subtle program behaviors
- Failure location
  - For bug reports and debugging tools

Why Calling Contexts?

- Context sensitive profiling
  - Identify subtle program behaviors
- Failure location
  - For bug reports and debugging tools
- Reverse engineering input formats
  - Contexts identify substructures

Why Calling Contexts?

- Context sensitive profiling
  - Identify subtle program behaviors
- Failure location
  - For bug reports and debugging tools
- Reverse engineering input formats
  - Contexts identify substructures
- Security
  - Tracking the sources of information

Existing Approaches

- Full Contexts
  - stack walking, calling context trees, ...

Existing Approaches

- Full Contexts
  - stack walking, calling context trees, ...
- Context IDs
  - probabilistic contexts, profile inferred contexts, ...

Existing Approaches

- Problems
  - Full contexts are too expensive

Existing Approaches

- Problems
  - Full contexts are too expensive
  - IDs don't allow reverse lookup

Given an ID, to what context does it belong?

Precise Context Features

- Encode many contexts to 1 integer
  - Uses multiple integers as necessary

Precise Context Features

- Encode many contexts to 1 integer
  - Uses multiple integers as necessary
- Reversible encoding

Precise Context Features

- Encode many contexts to 1 integer
  - Uses multiple integers as necessary

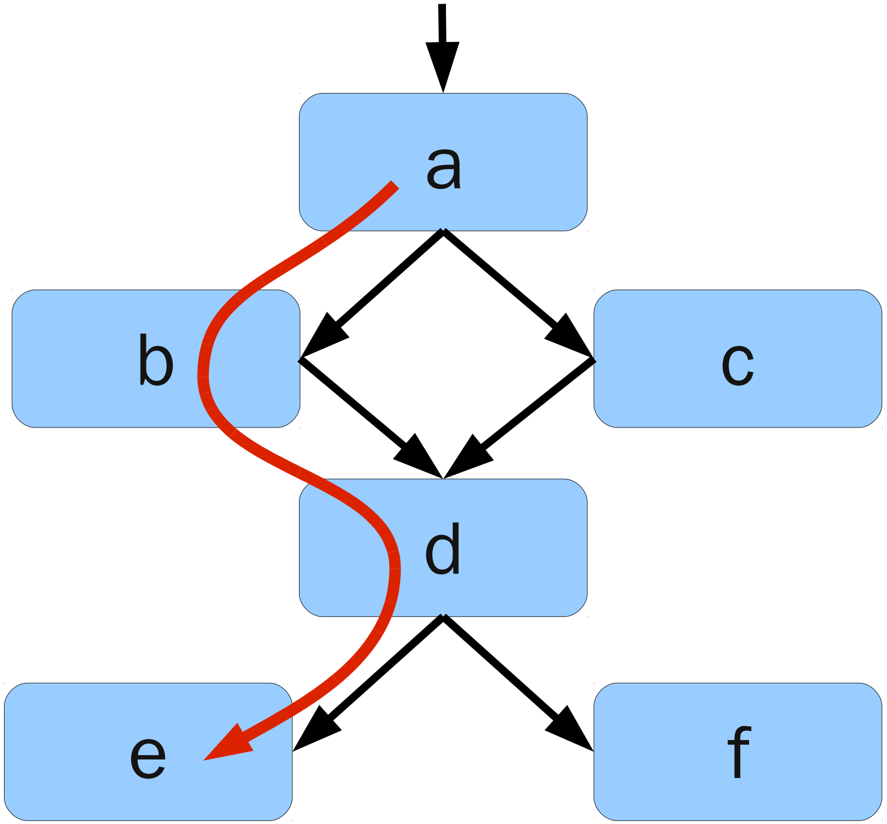
- Reversible encoding

- Robust
  - Recursion, indirection, exceptions, ...

Precise Context Features

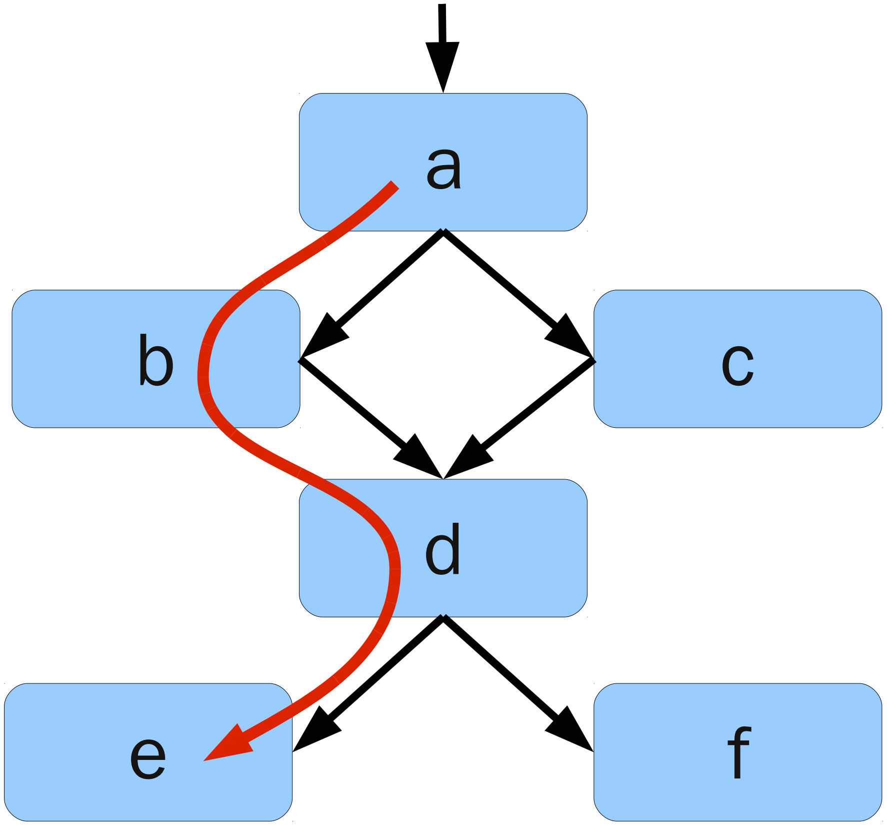
- Encode many contexts to 1 integer
  - Uses multiple integers as necessary
- Reversible encoding
- Robust
  - Recursion, indirection, exceptions, …
- Optimized using stack sizes and profiling
  - 1.9% - 3% overhead

Precise Context Encoding

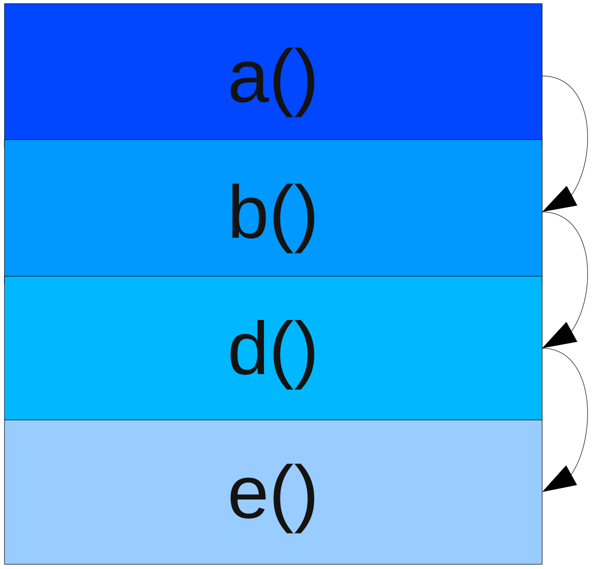


Each context is a path in the call graph

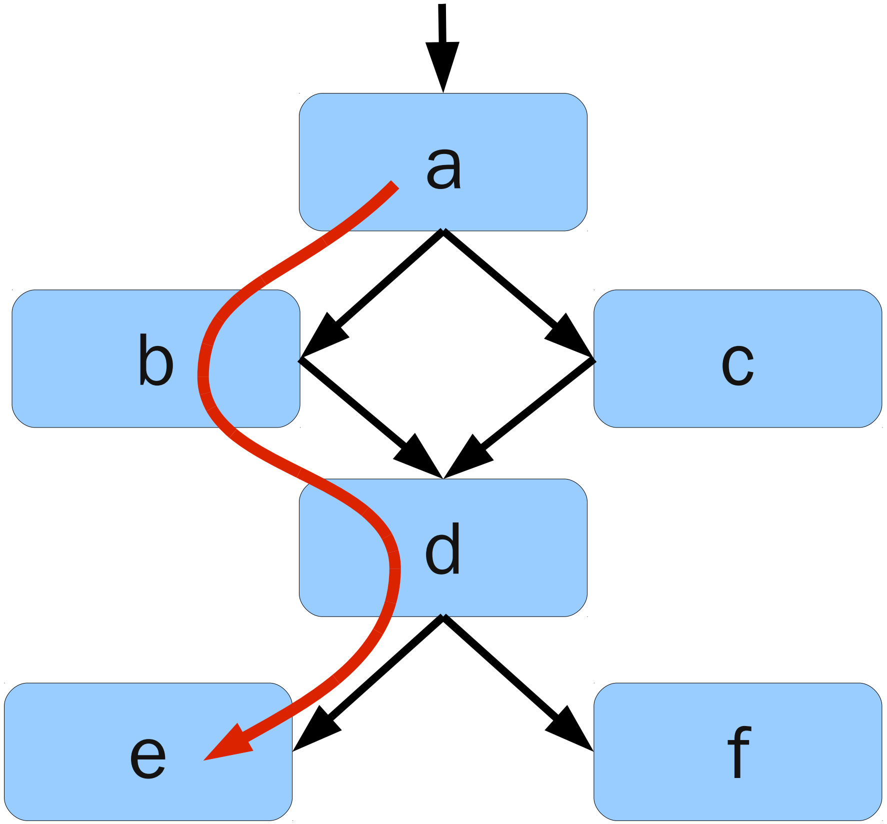
Precise Context Encoding



Each context is a path in the call graph

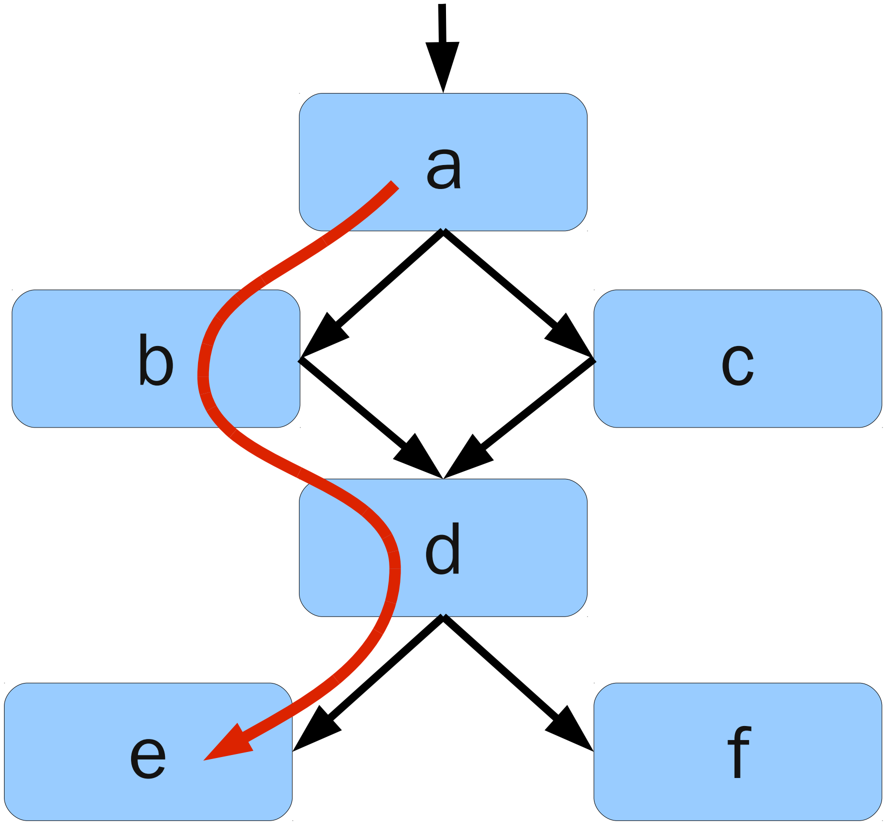


Precise Context Encoding



Use unique path numbering over the call graph

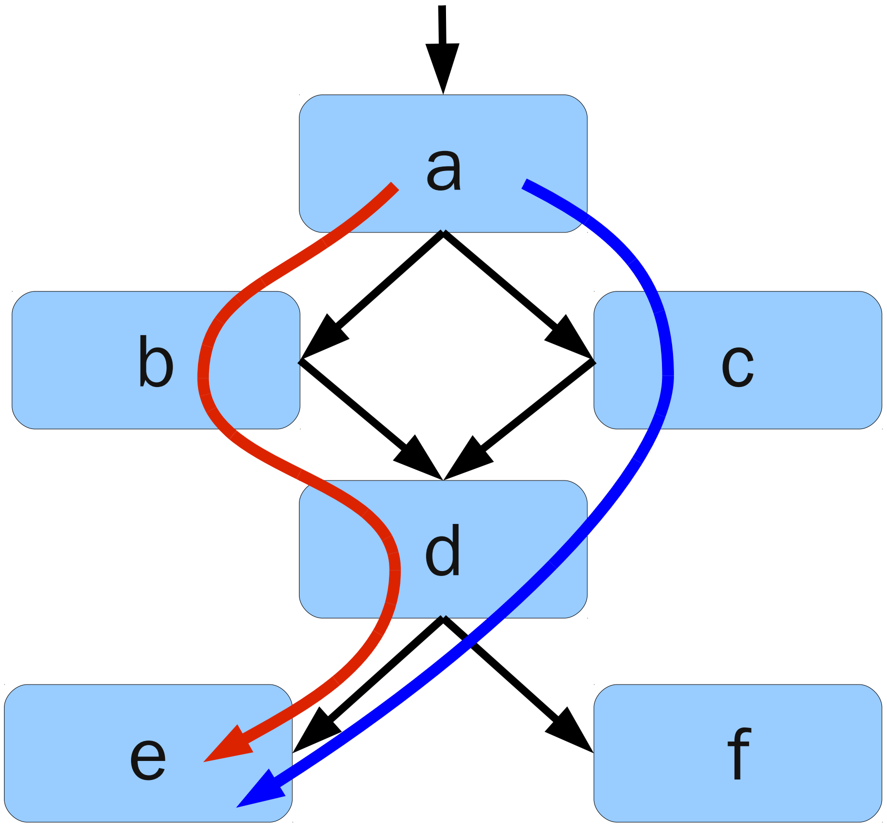
Precise Context Encoding



Use unique path numbering over the call graph

🟥 1

Precise Context Encoding



Use unique path numbering over the call graph

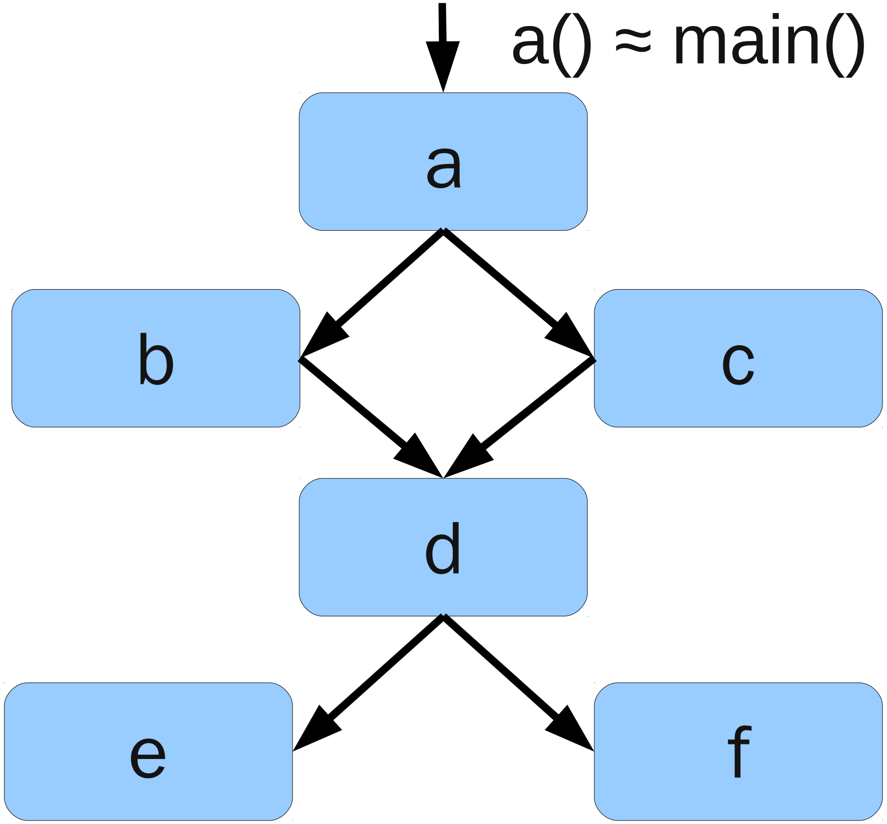
🟥 1

🟦 2

Precise Context Encoding

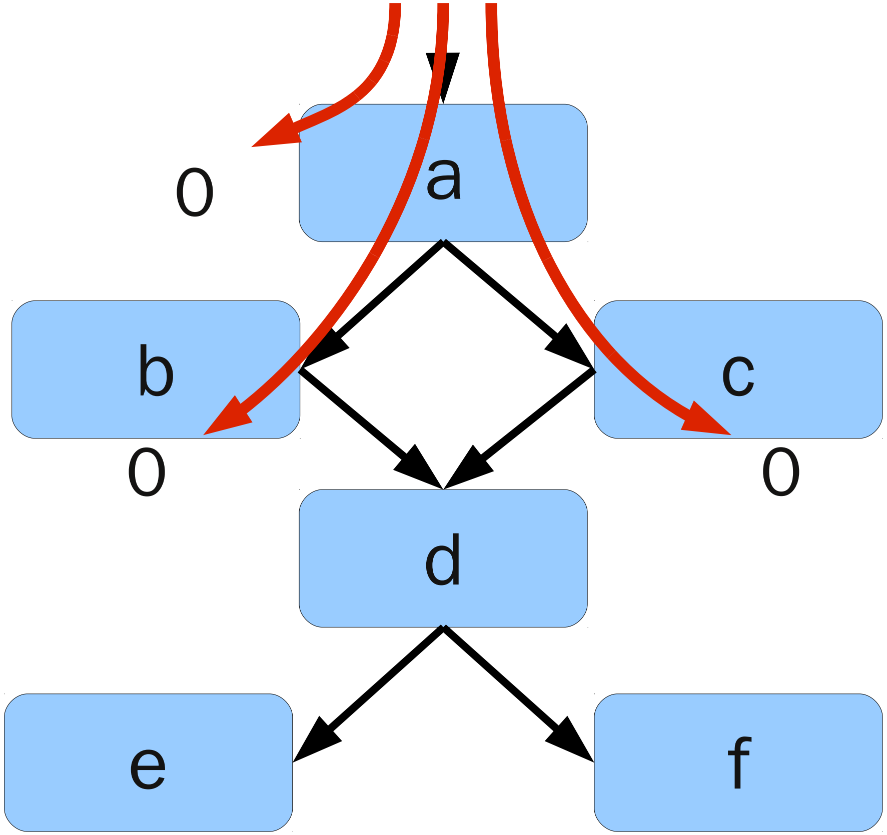
- Encode each context in a number
  - Compute the current context number online
  - Similar to Ball-Larus CFG path numbering

Basic Context Encoding



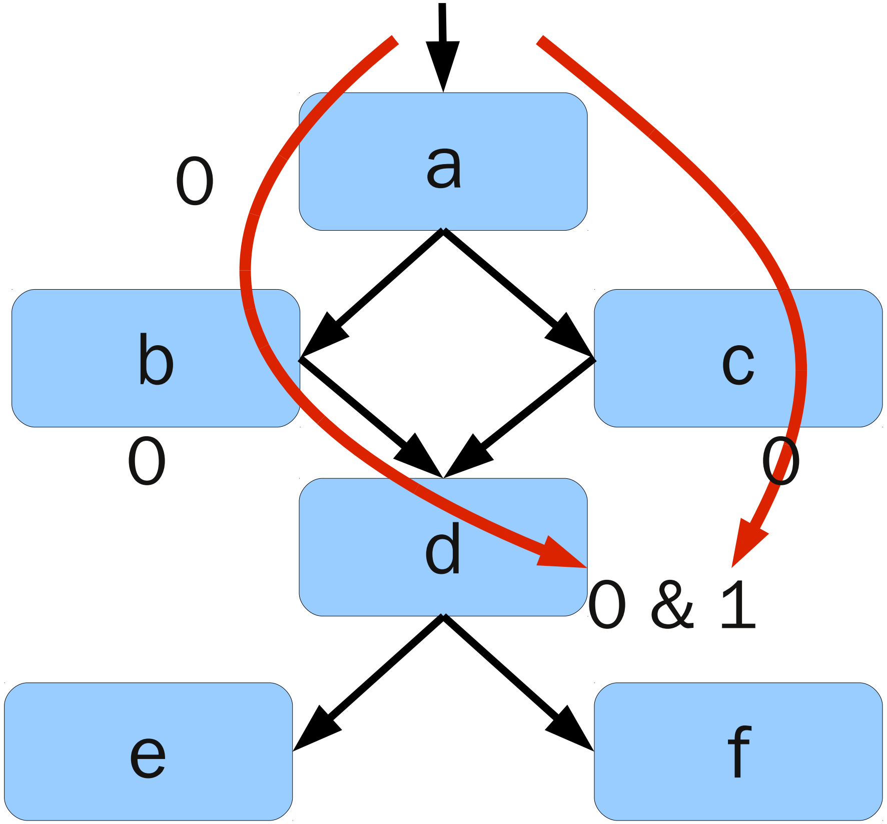
- Paths start at the root
- They may end **anywhere**

Basic Context Encoding



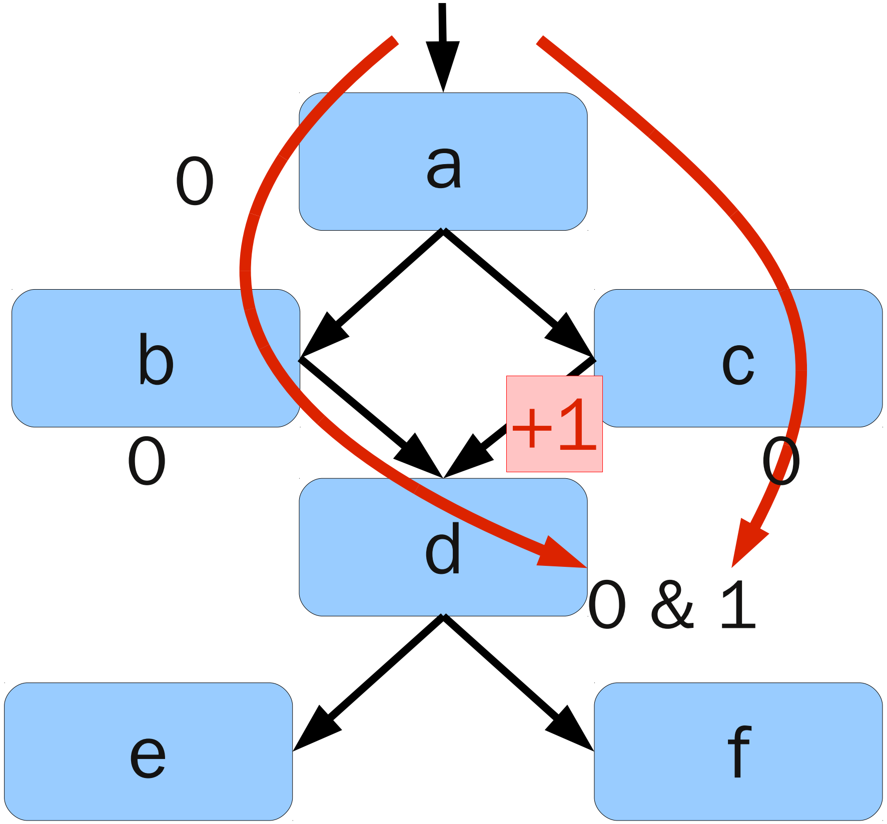
- Paths start at the root
- They may end **anywhere**

Basic Context Encoding



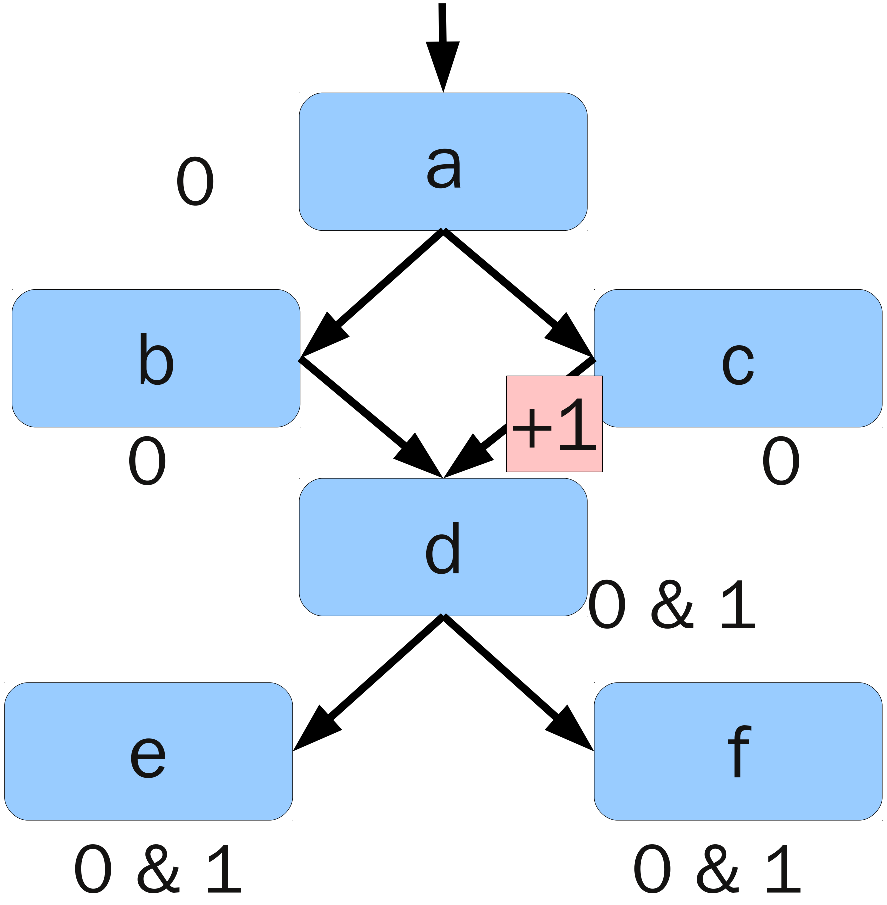
- Paths start at the root
- They may end **anywhere**

Basic Context Encoding



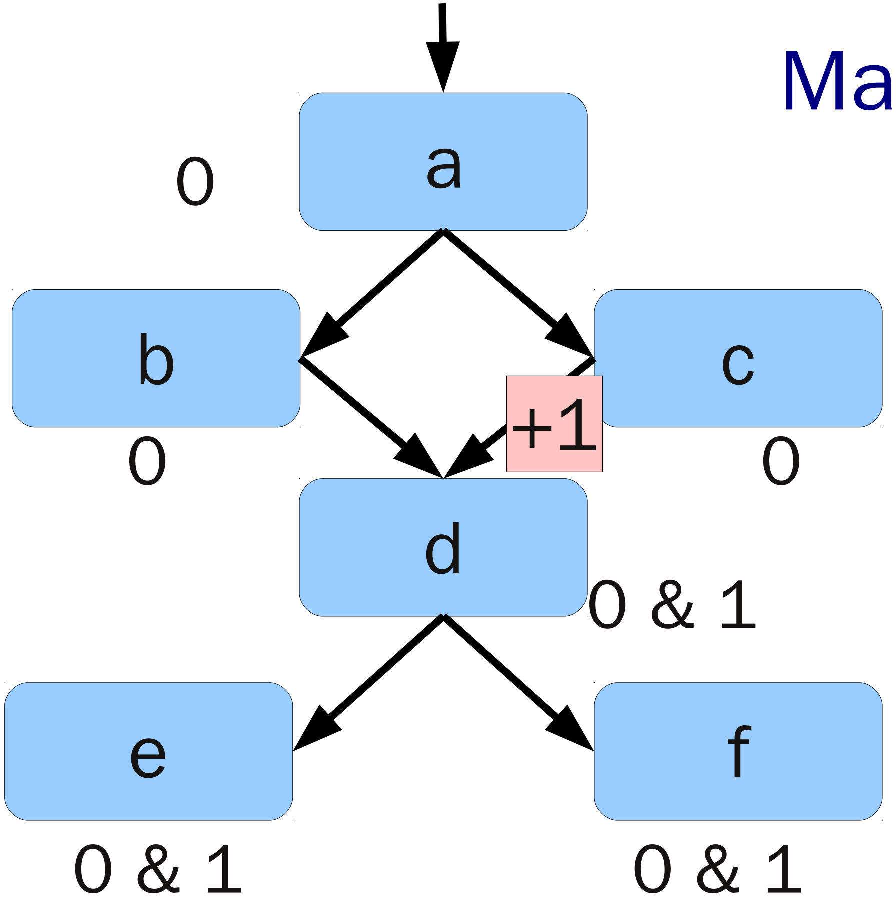
- Paths start at the root
- They may end **anywhere**

Basic Context Encoding



- Paths start at the root
- They may end **anywhere**
- We reuse the solutions for common subproblems

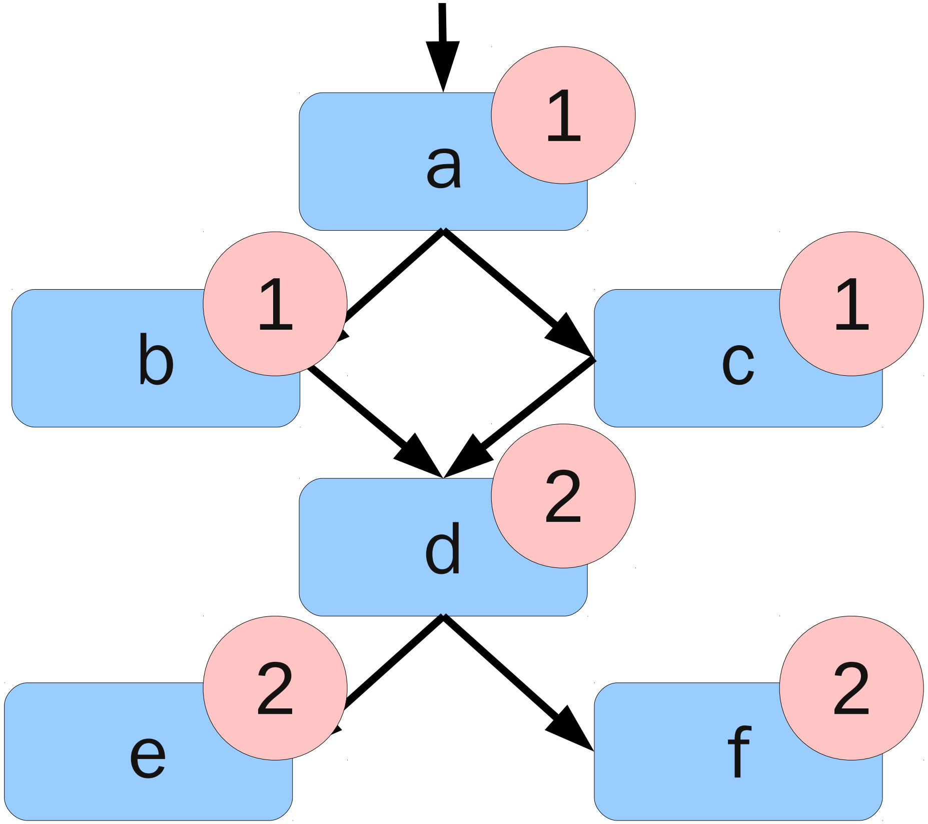
Basic Context Encoding



Maintain the current ID online

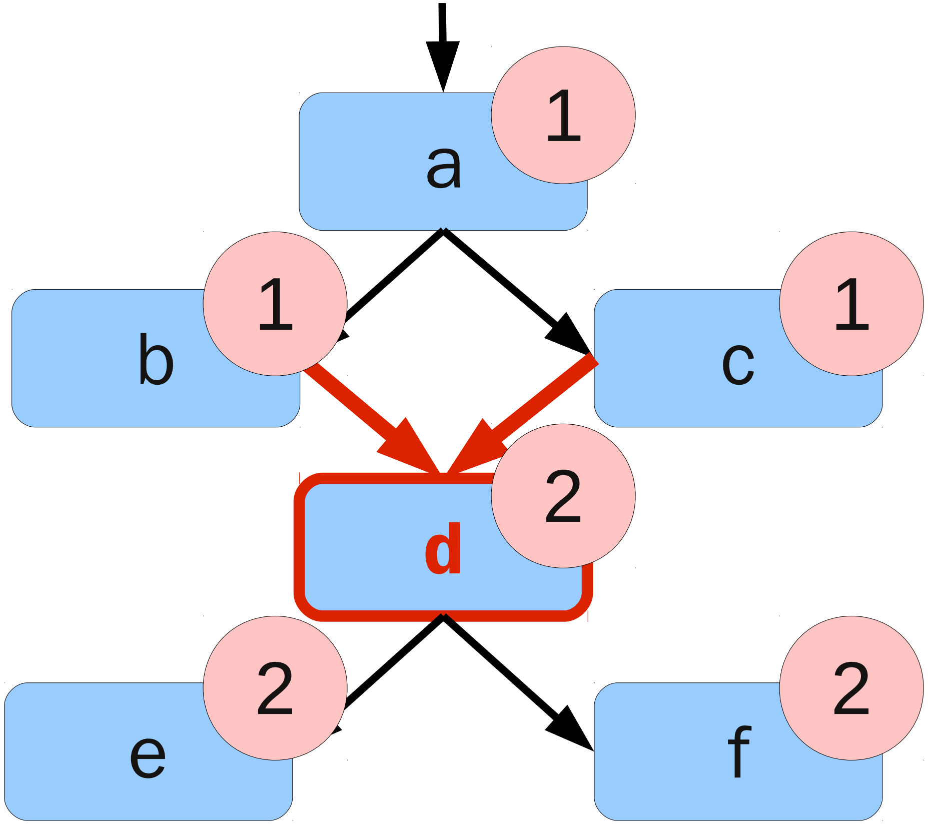
```
def c():
    ...
    contextID += 1
    d()
    contextID -= 1
    ...
```

Basic Context Encoding



- Count # of contexts per function

Basic Context Encoding

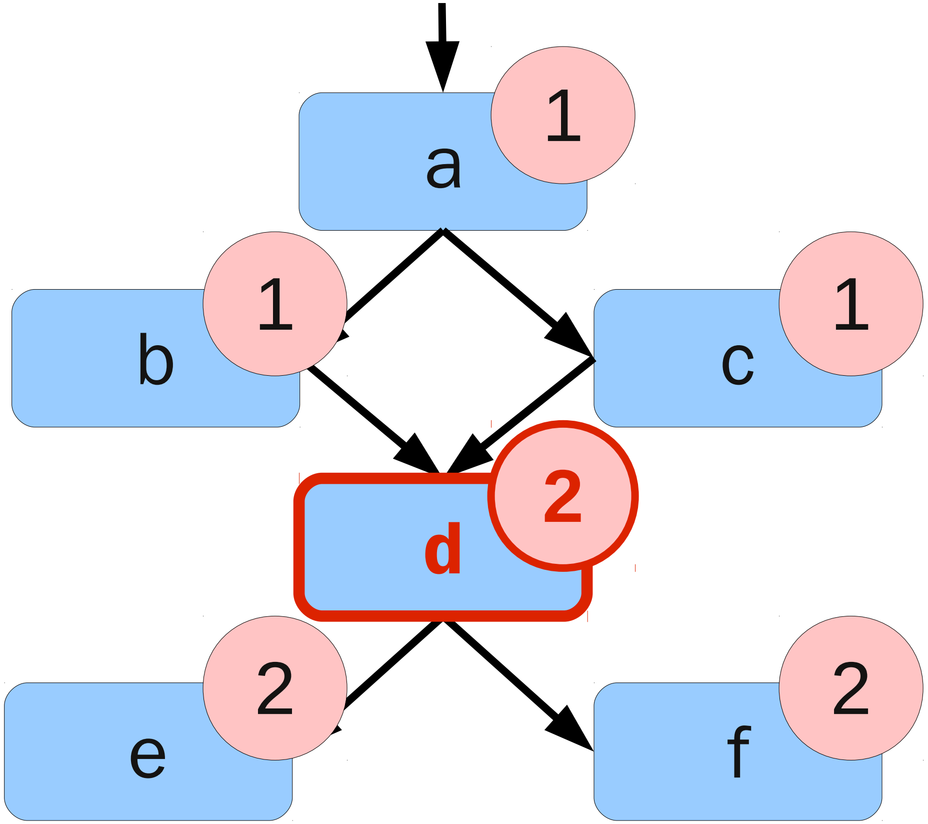


- Count # of contexts per function

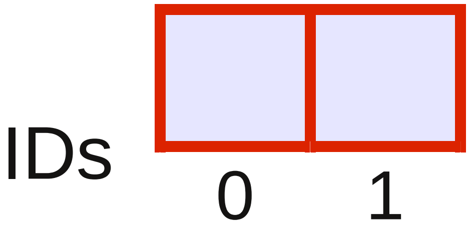
for each function:

Σ # contexts for each caller

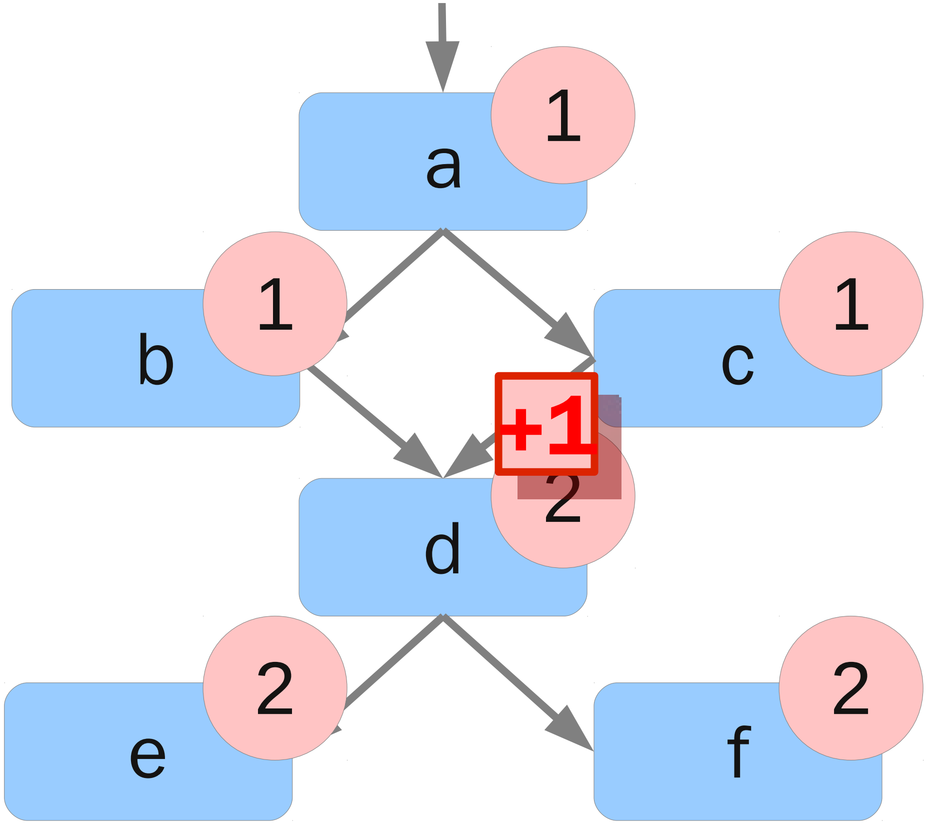
Basic Context Encoding



- Count # of contexts per function



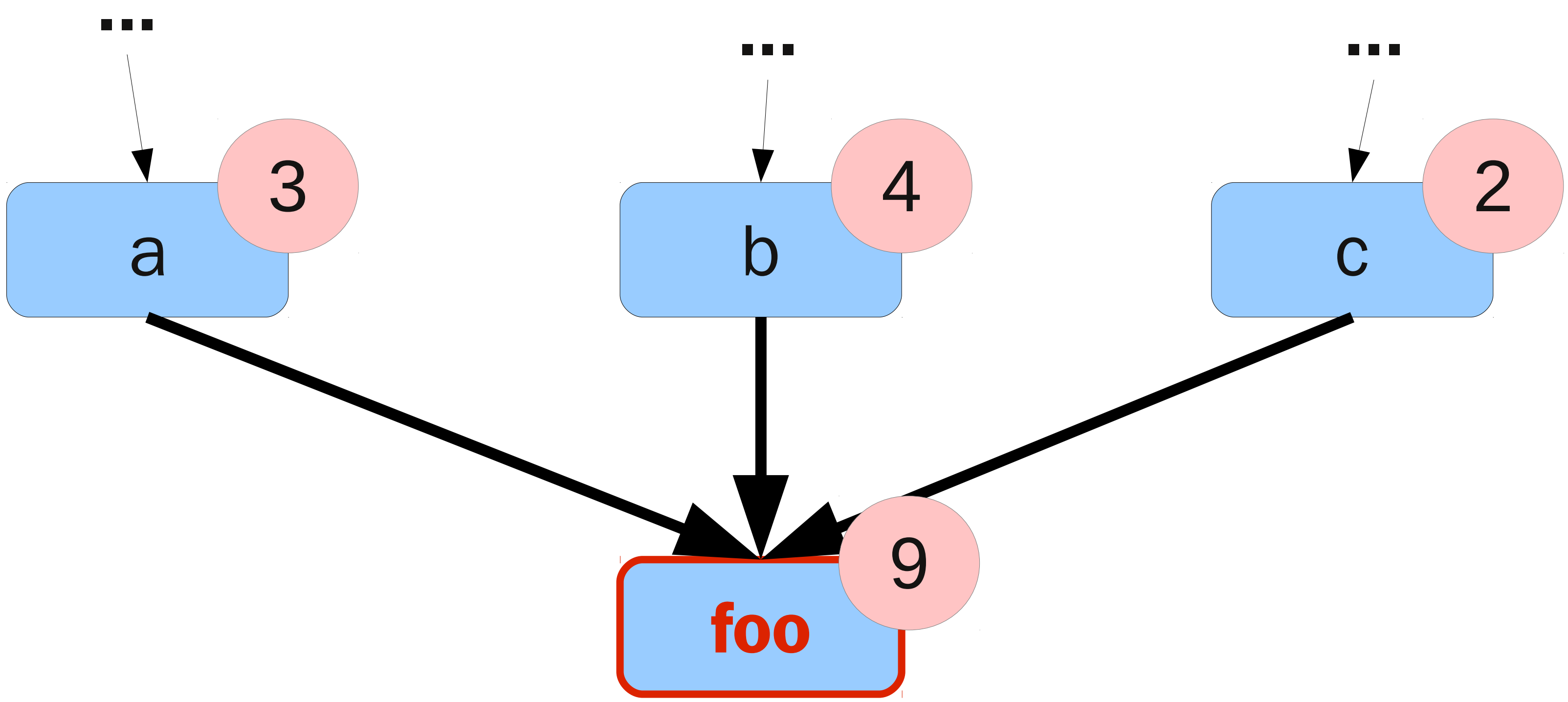
Basic Context Encoding



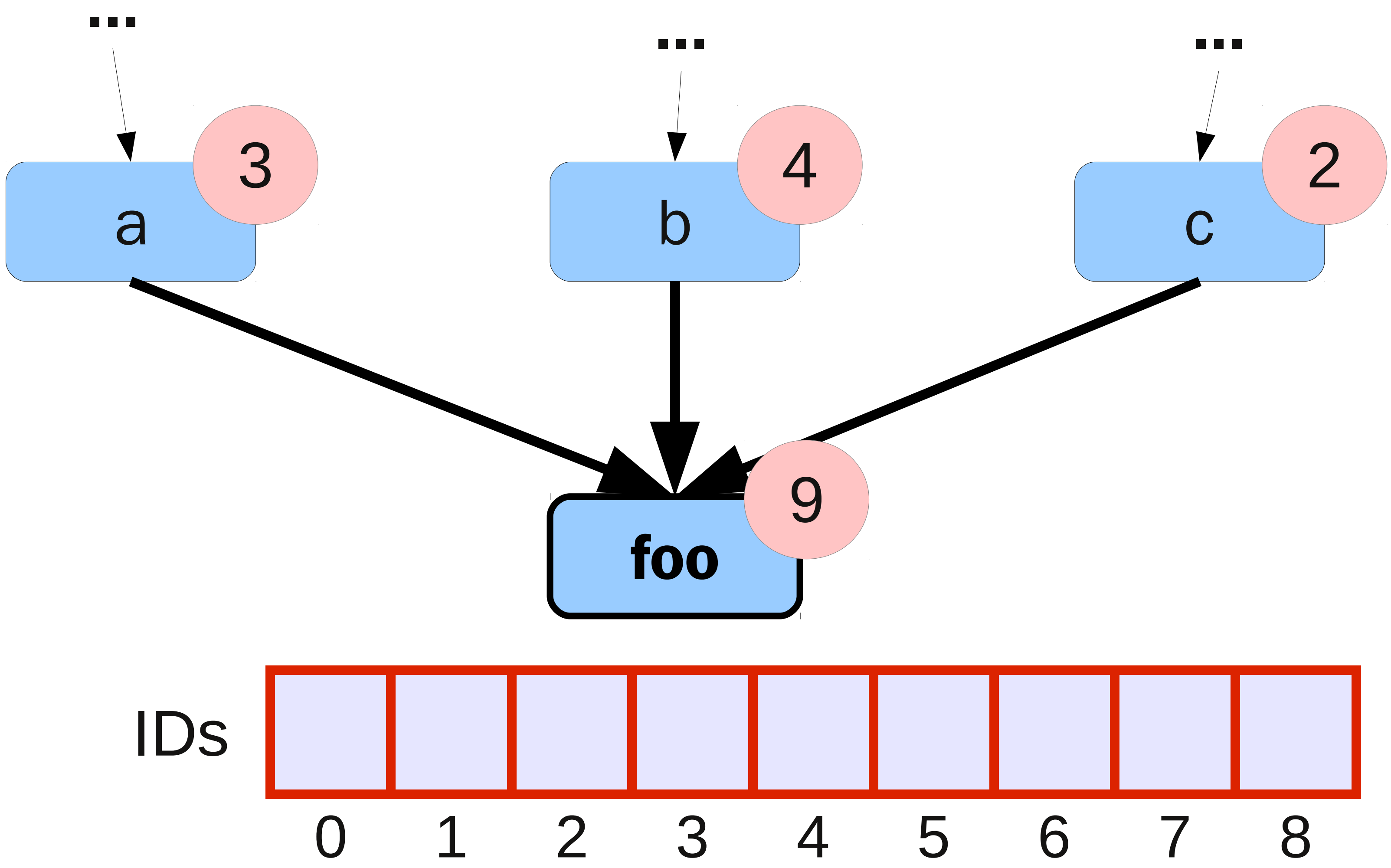
- Use instrumentation to partition ID space

	b	c
IDs	0	1

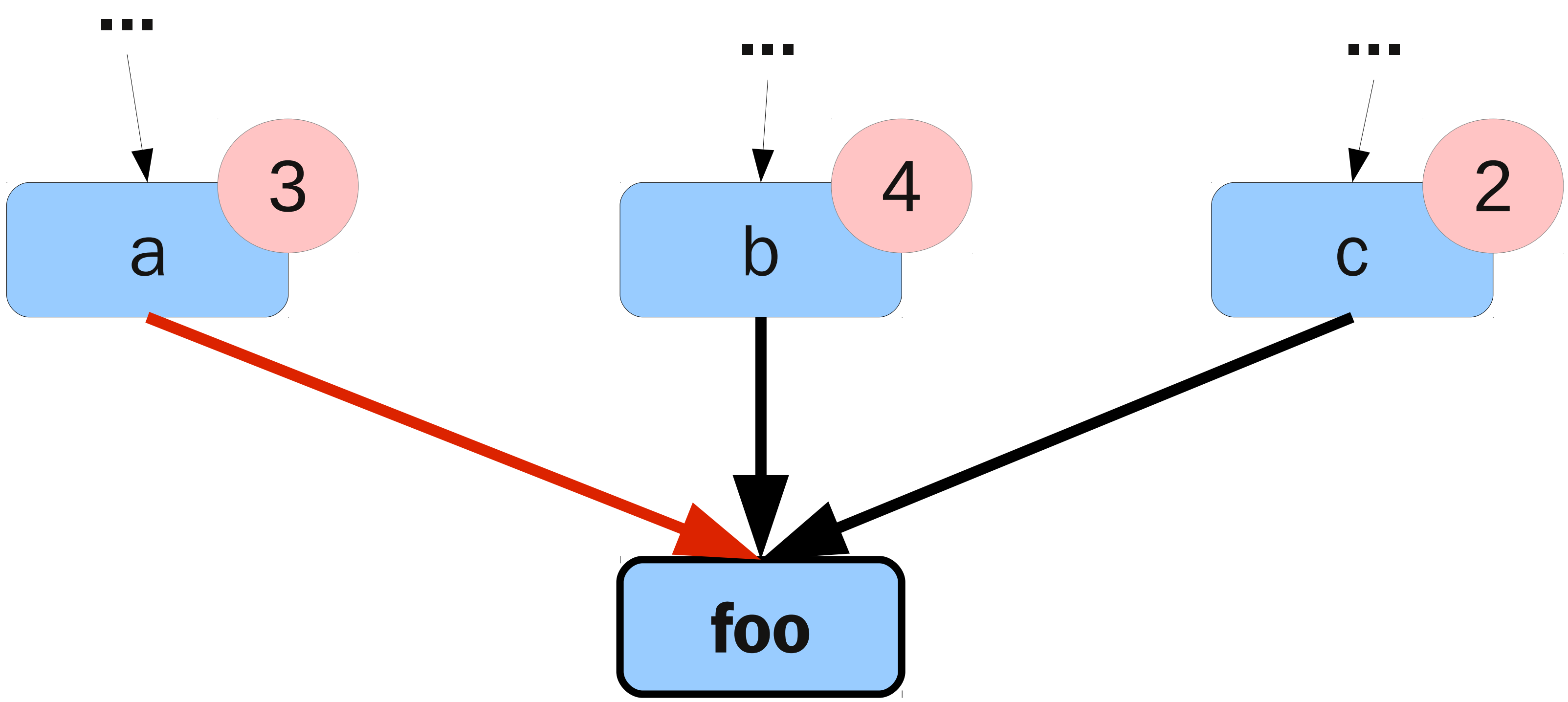
Basic Context Encoding



Basic Context Encoding

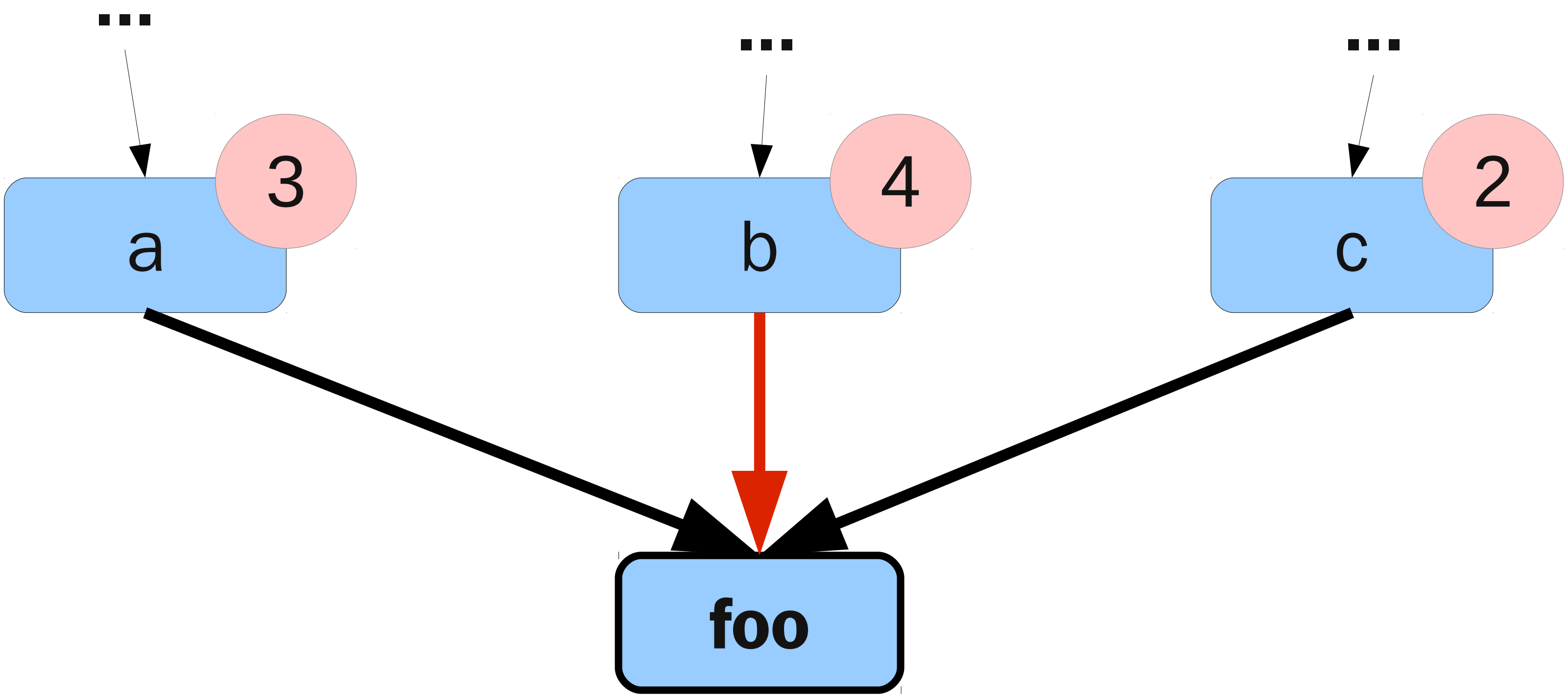


Basic Context Encoding



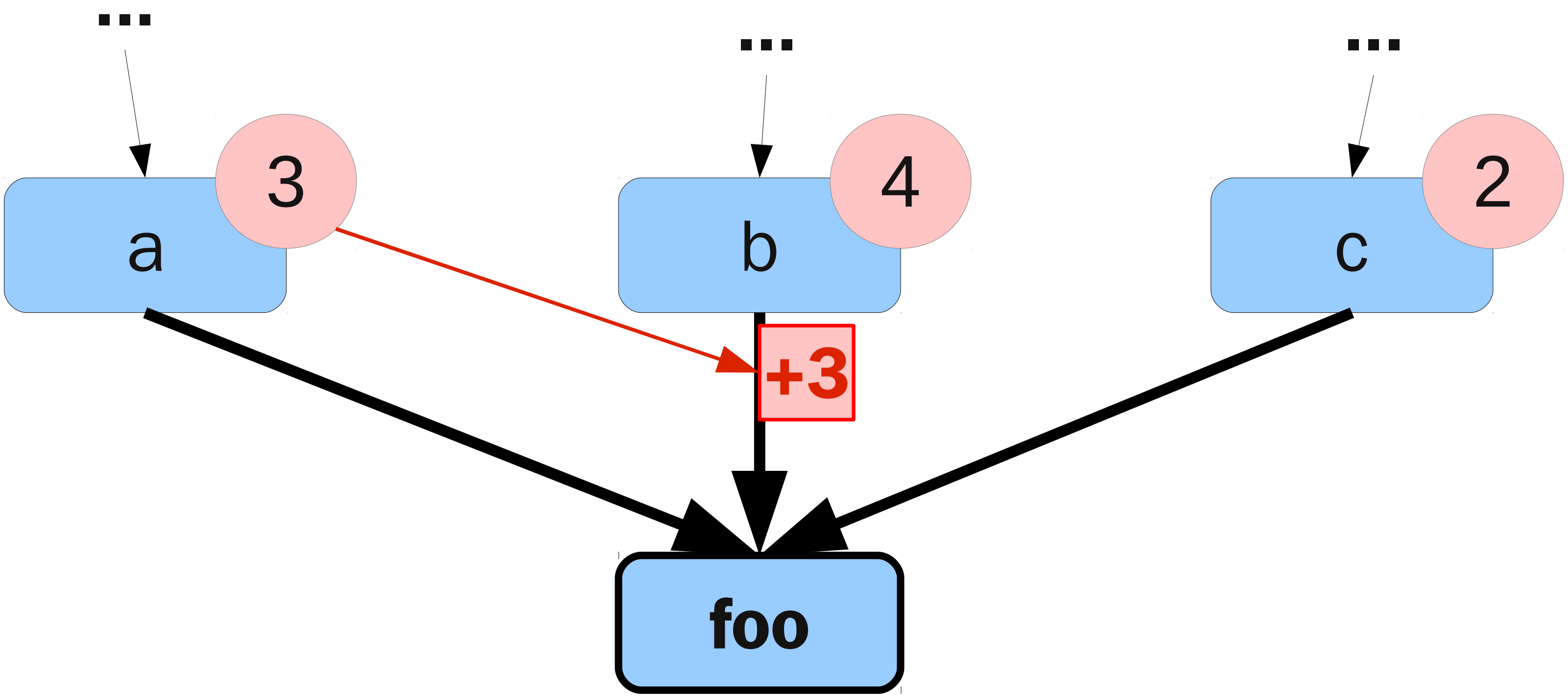
IDs	a	a	a						
	0	1	2	3	4	5	6	7	8

Basic Context Encoding



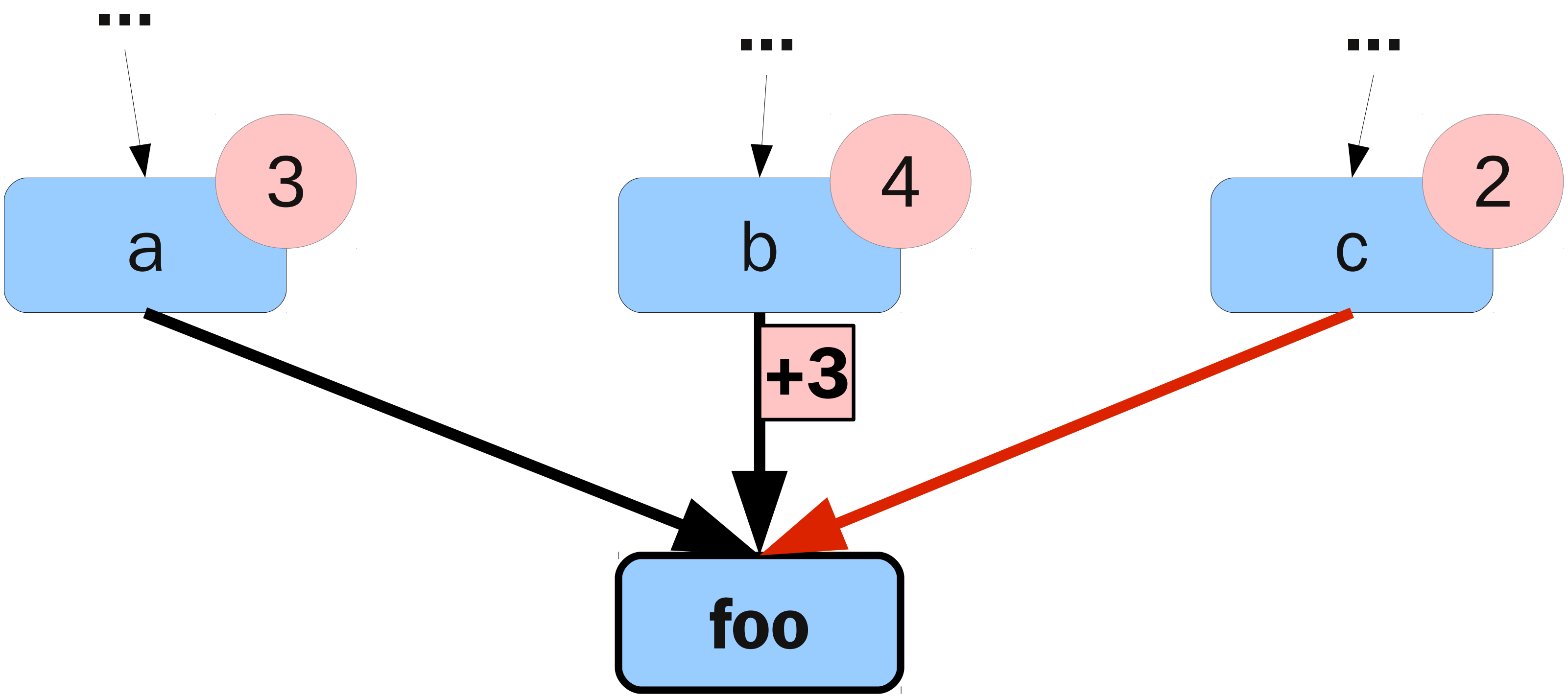
IDs	a	a	a	b	b	b	b		
	0	1	2	3	4	5	6	7	8

Basic Context Encoding



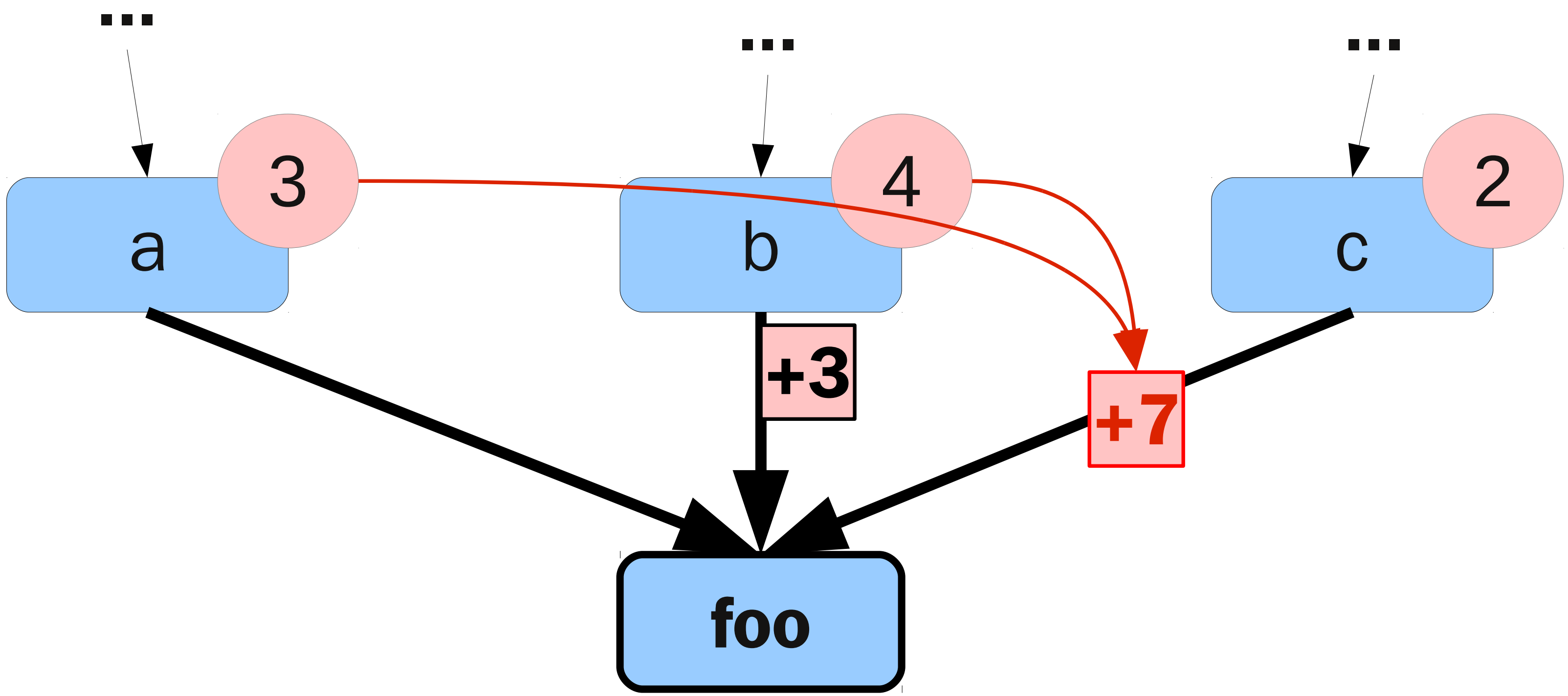
IDs	a	a	a	b	b	b	b		
	0	1	2	3	4	5	6	7	8

Basic Context Encoding



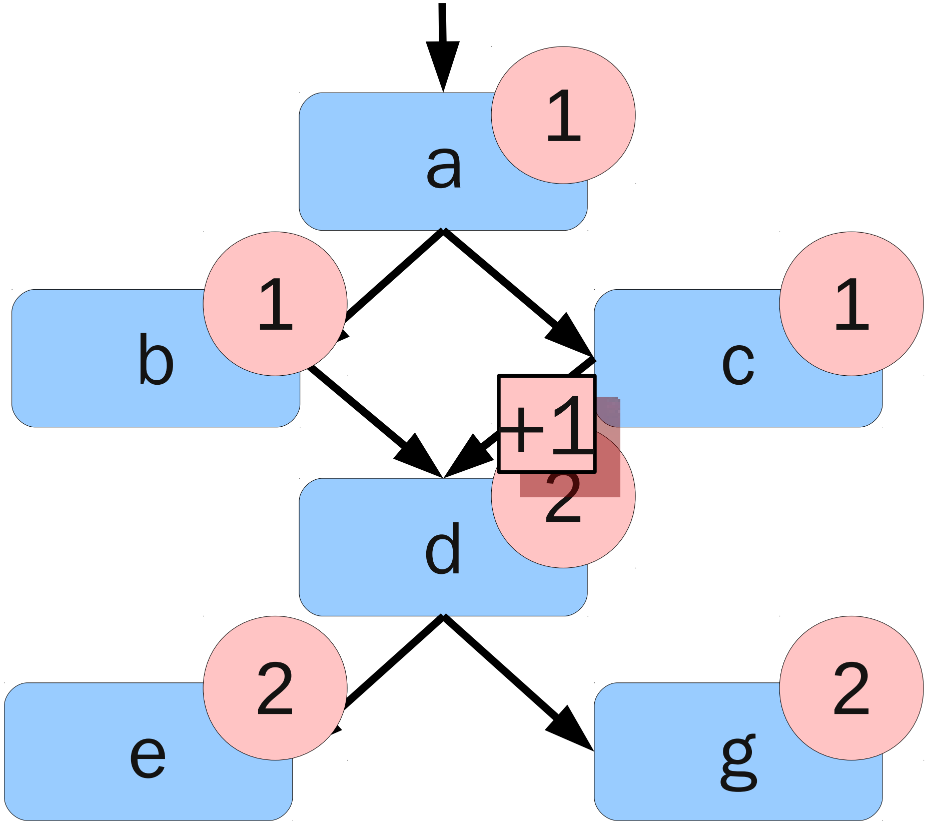
IDs	a	a	a	b	b	b	b	c	c
	0	1	2	3	4	5	6	7	8

Basic Context Encoding



IDs	a	a	a	b	b	b	b	c	c
	0	1	2	3	4	5	6	7	8

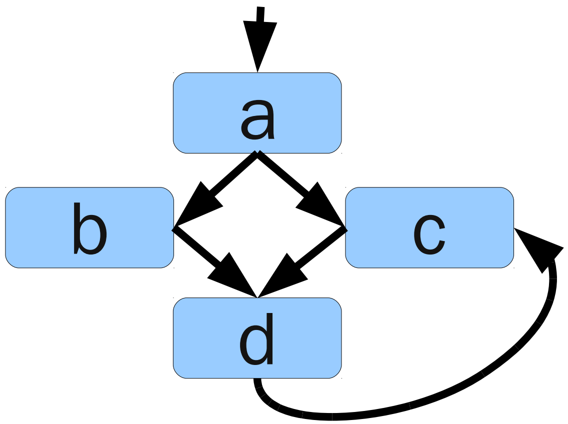
Basic Context Encoding



- Use instrumentation to partition ID space
- Decoding simply reverses the process

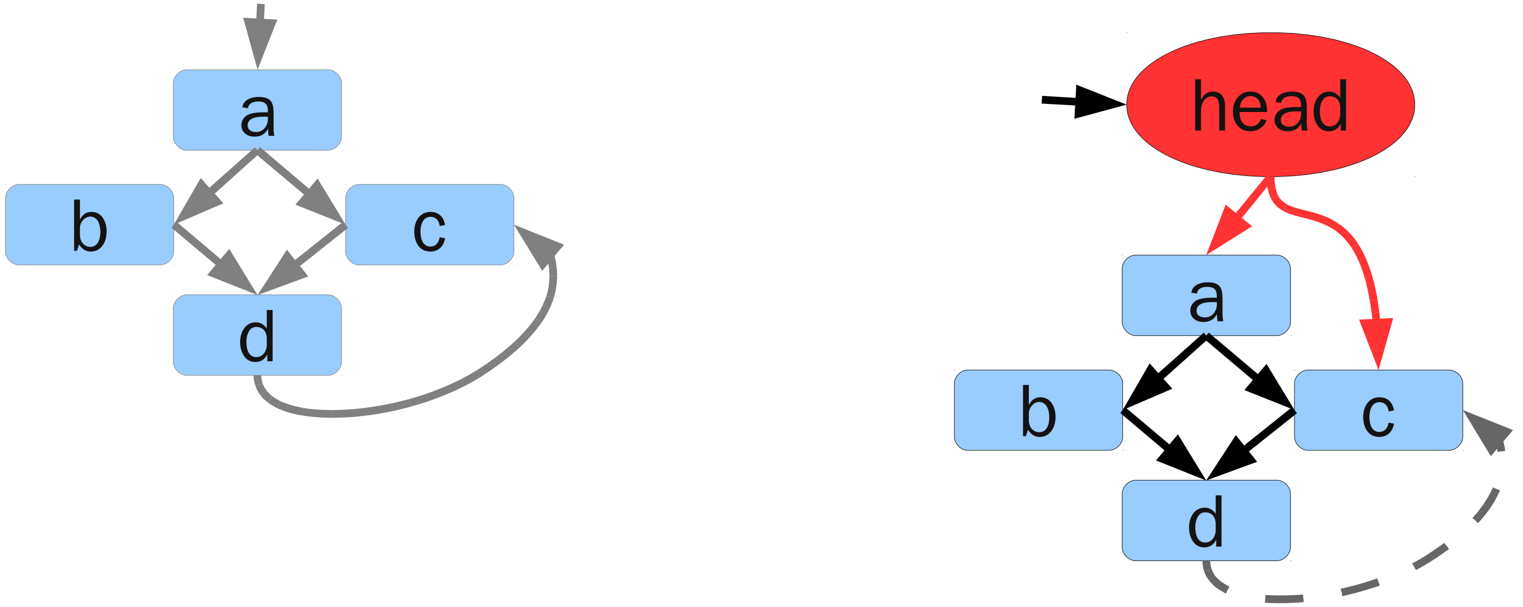
Recursion

- With recursion\cycles, numbering is unbounded.



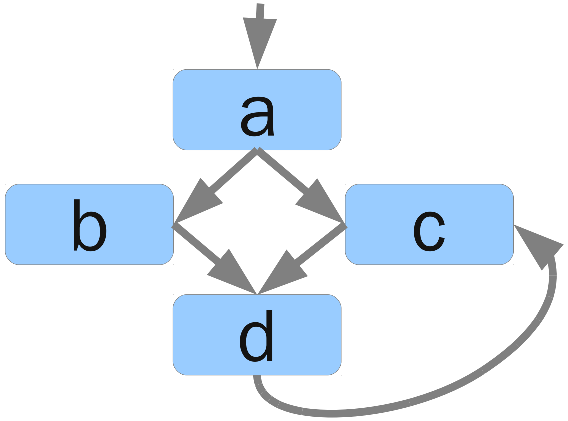
Recursion

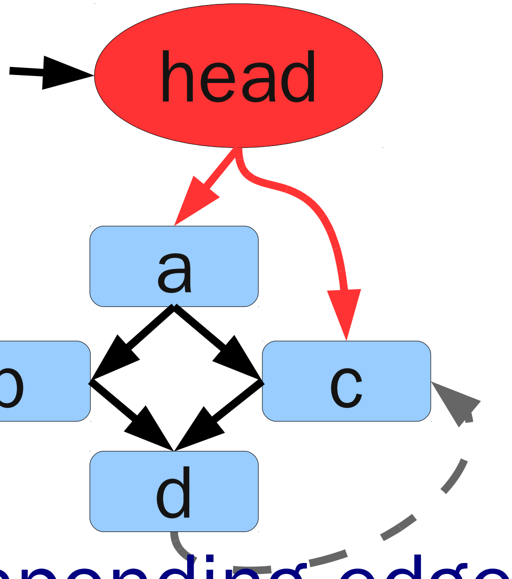
- With recursion\cycles, numbering is unbounded.
  - Transform them into acyclic graphs.



Recursion

- With recursion\cycles, numbering is unbounded.
  - Transform them into acyclic graphs.





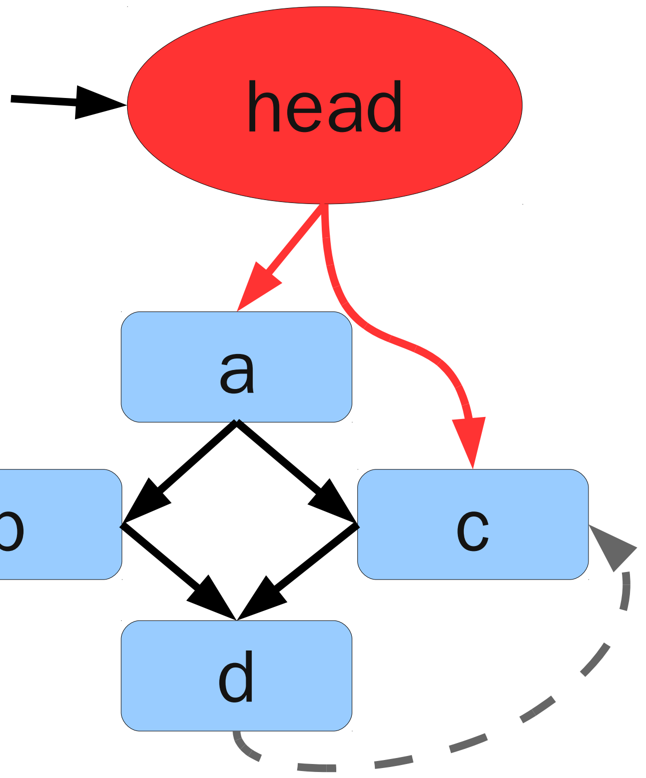
- Each back edge has a corresponding edge in the new acyclic graph.
  - Each cyclic path becomes a list of acyclic paths

Recursion

- Push the current ID onto a context stack before recursive calls.

Instrumentation:

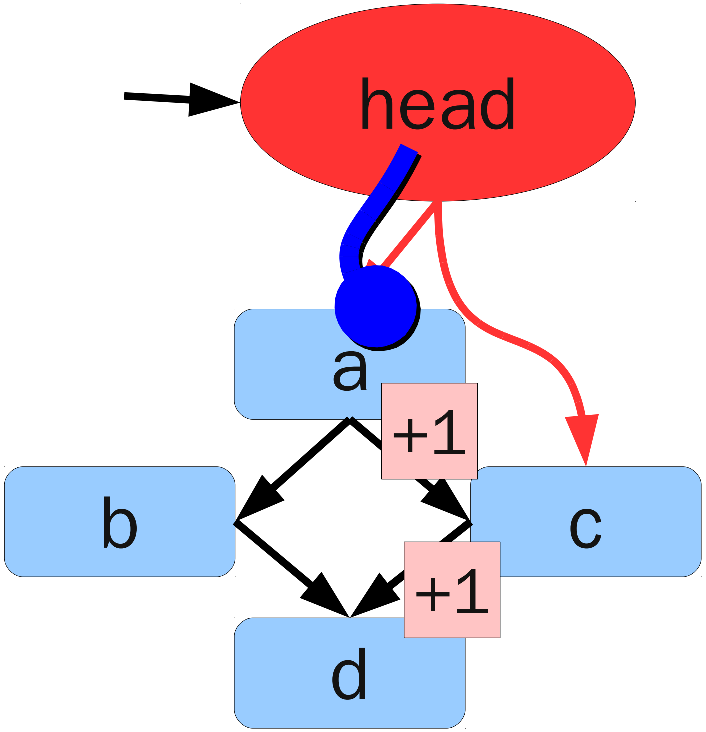
```
def d():
   ...
   push(d, contextID)
   contextID = 0
   c()
   contextID = pop()
   ...
```



Recursion

- In the series of calls: a→c→d→c→d

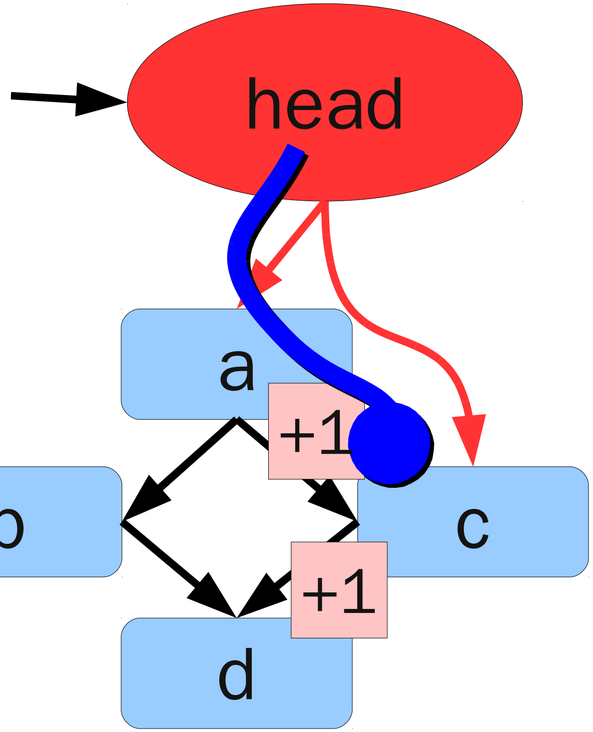
Last Called	ID
a	0



Recursion

- In the series of calls: a→c→d→c→d

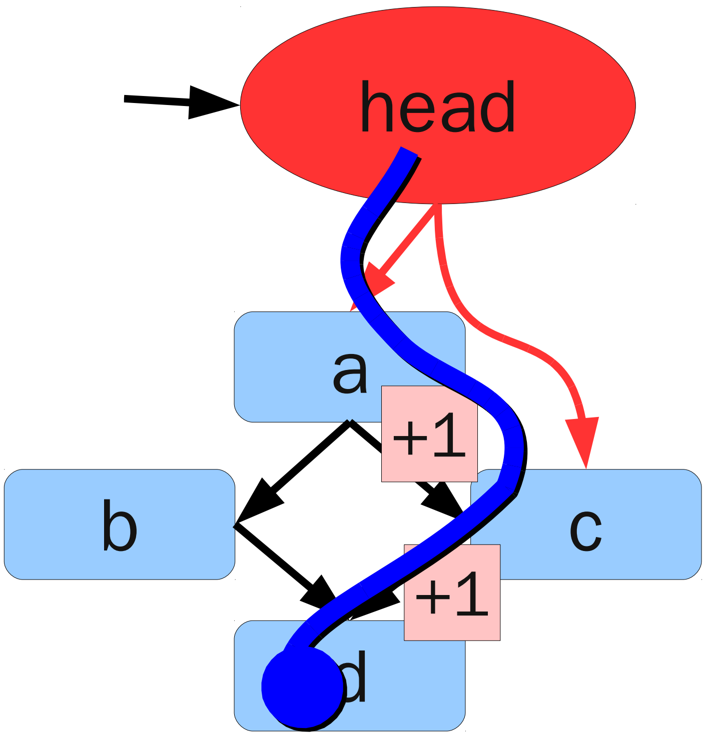
Last Called	ID
a	0
c	1



Recursion

- In the series of calls: a→c→d→c→d

Last Called	ID
a	0
c	1
d	2



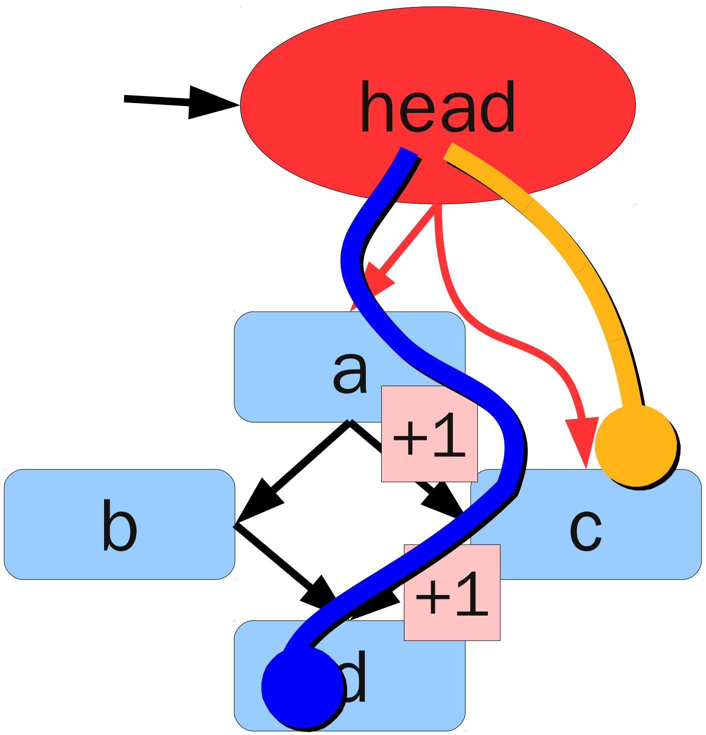
Recursion

- In the series of calls: a→c→d→c→d

Last Called	ID
a	0
c	1
d	2
c	0 || 2

ID

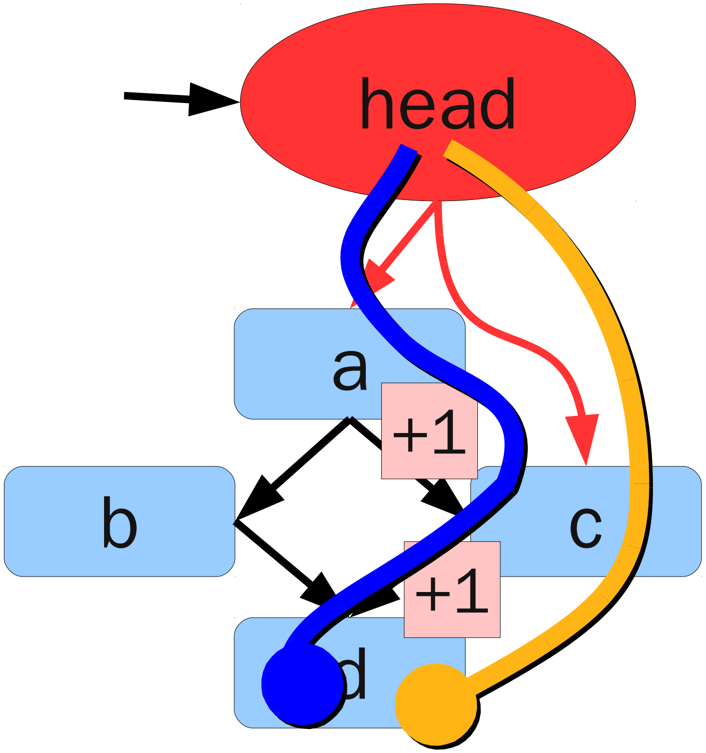
Context Stack



Recursion

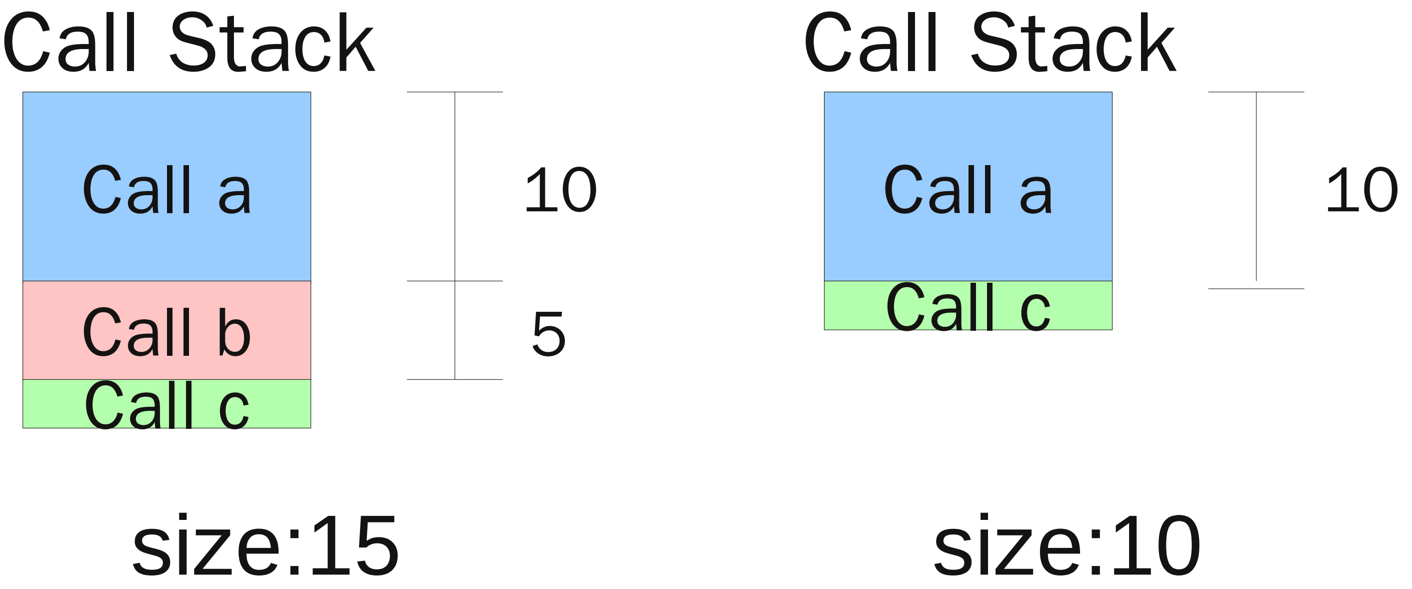
- In the series of calls: a→c→d→c→d

Last Called	ID
a	0
c	1
d	2
c	0 || 2
d	1 || 2



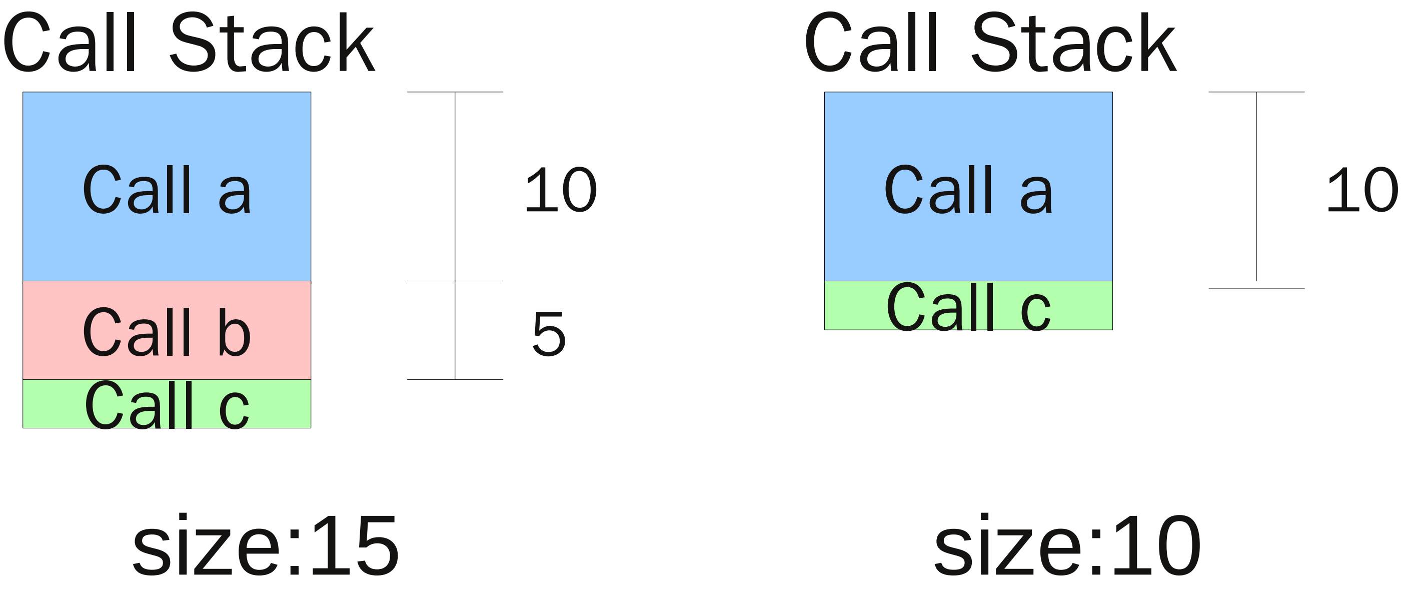
Precise Implicit Encoding

- Some contexts can be precisely identified by stack sizes



Precise Implicit Encoding

- Some contexts can be precisely identified by stack sizes
  - We can use these when possible and fall back on explicit encoding when necessary.



Call Stack

Call a

Call b

Call c

10

5

size:15

Call Stack

Call a

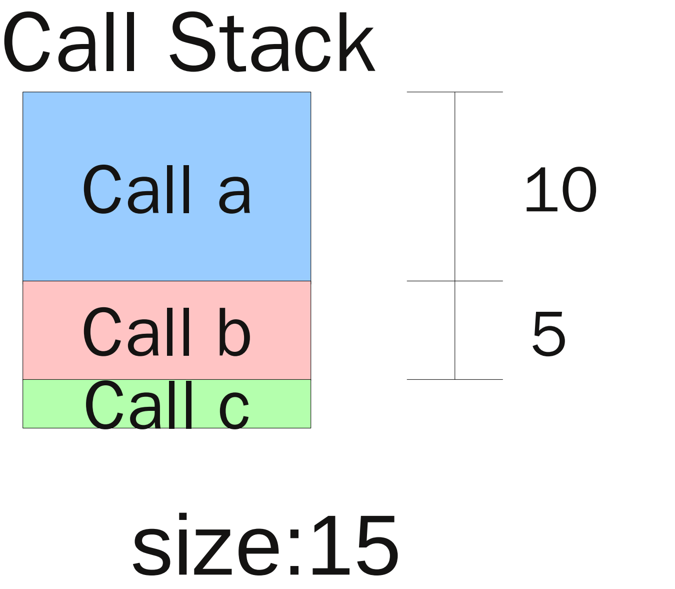
Call c

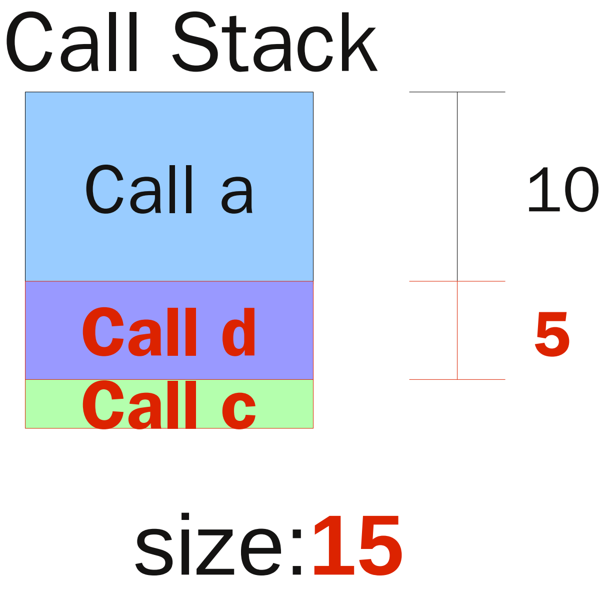
10

size:10

Precise Implicit Encoding

- Some contexts can be precisely identified by stack sizes
  - We can use these when possible and fall back on explicit encoding when necessary.





Precise Implicit Encoding

- Some contexts can be precisely identified by stack sizes
  - We can use these when possible and fall back on explicit encoding when necessary.

- Fall back on explicit encoding for contexts w/:
  - Variable stack allocation
  - Recursive paths
  - Conflicting contexts with the same size

Evaluation

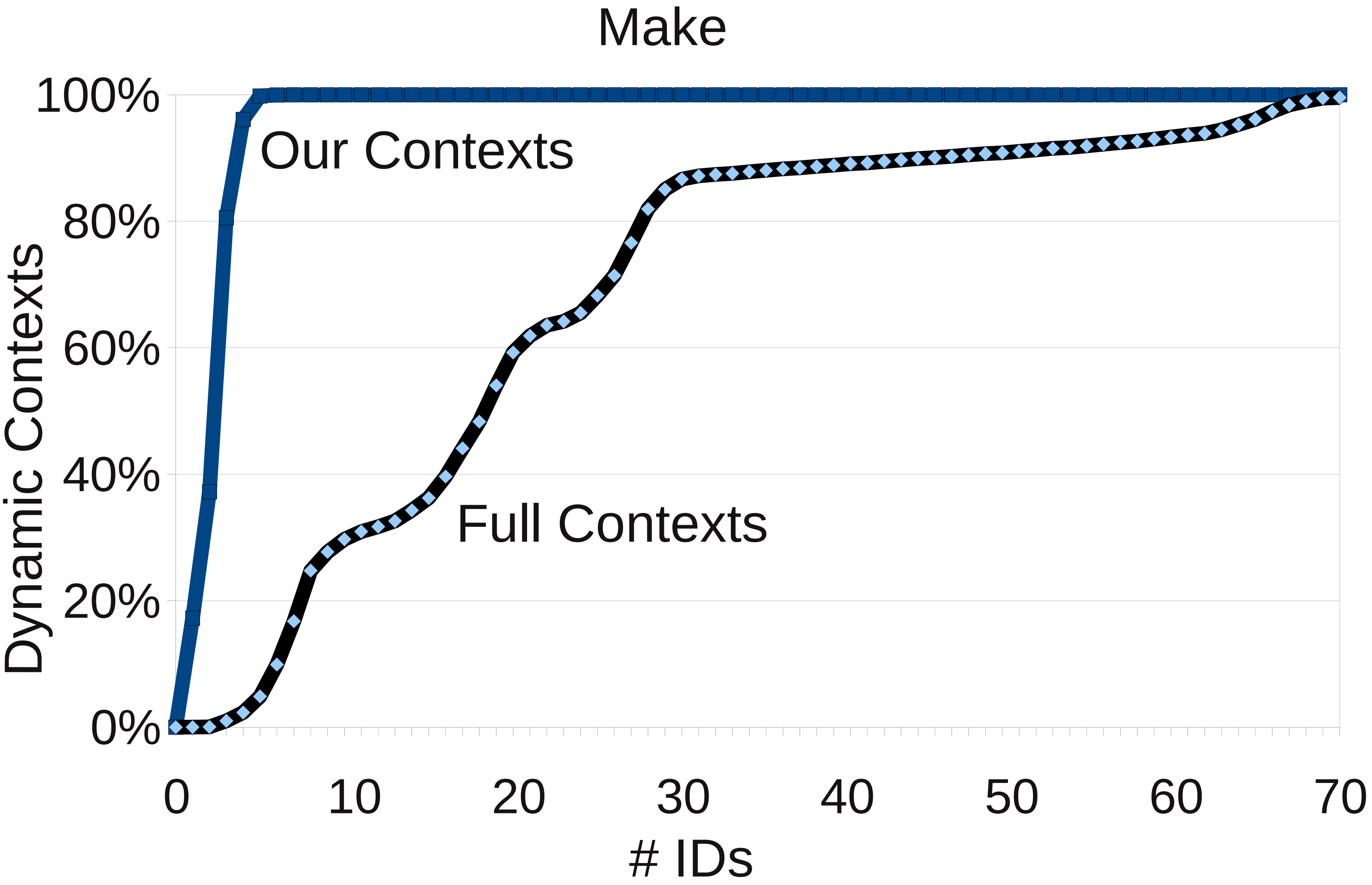
- Implemented prototype using CIL
- Examined results on SPEC 2000 and a set of real world programs
- 32-bit IDs

Evaluation: Context Attributes

Program	Max Size		90% Size		# Contexts
	Ours	Full	Ours	Full	
164.gzip	1	9	1	7	258
175.vpr	1	9	1	6	1553
176.gcc	20	136	3	15	169090
181.mcf	15	42	1	2	12920
186.crafty	35	41	11	23	27103471
197.parser	37	73	12	28	3023011
255.vortex	8	43	3	12	205004
256.bzip2	2	8	1	8	96
300.twolf	5	11	1	5	971

Program	Max Size		90% Size		# Contexts
	Ours	Full	Ours	Full	
cmp 2.8.7	1	3	1	3	9
diff 2.8.7	1	7	1	5	34
sdiff 2.8.7	1	5	1	4	44
find 4.4.0	3	12	2	12	186
locate 4.4.0	1	9	1	9	65
grep 2.5.4	1	11	1	8	117
tar 1.16	4	40	3	31	1346
make 3.80	7	82	4	43	1789
alpine 2.0	12	29	7	18	7575
vim 6.0	11	31	6	10	3226

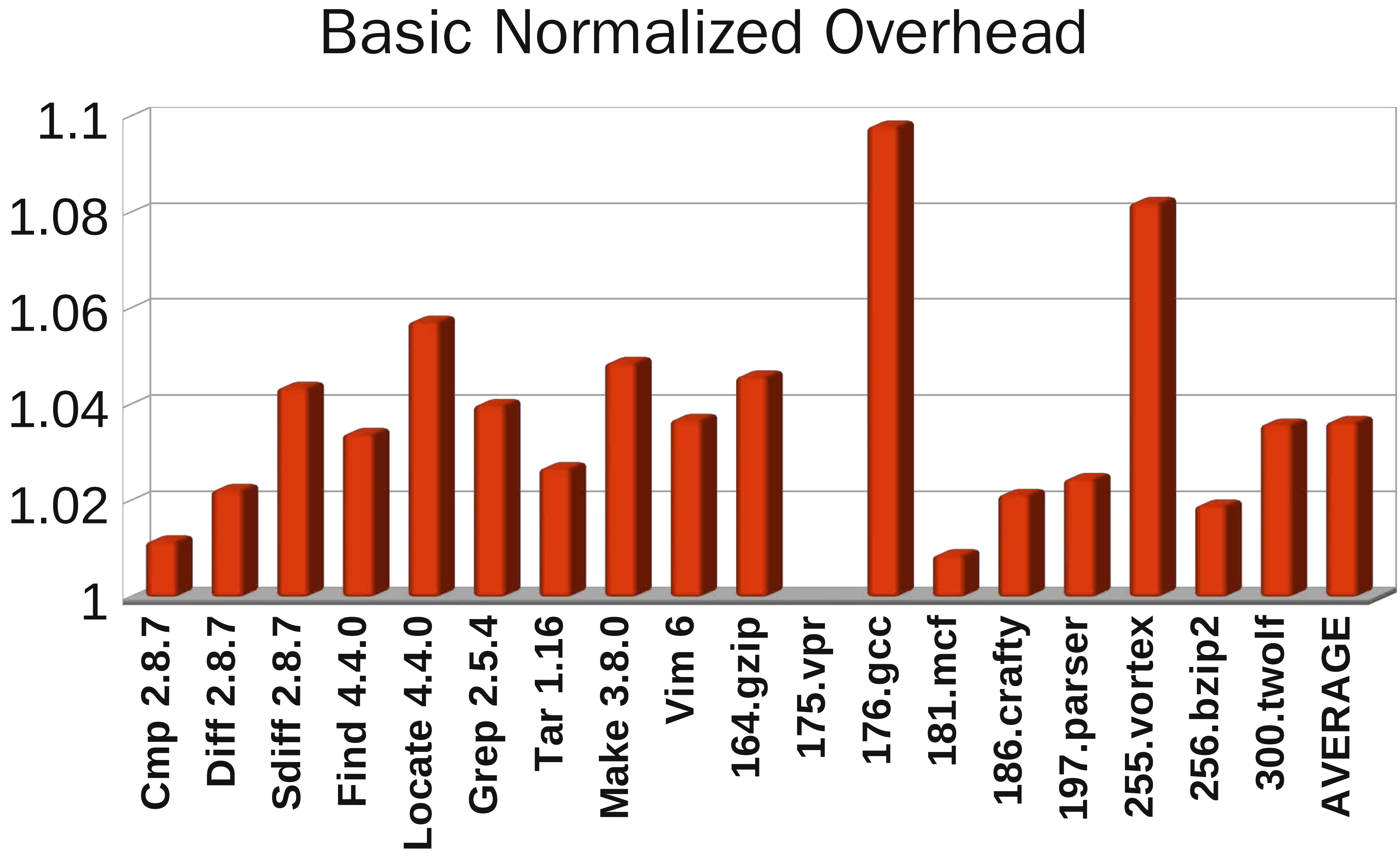
Context Stack Size Sufficiency



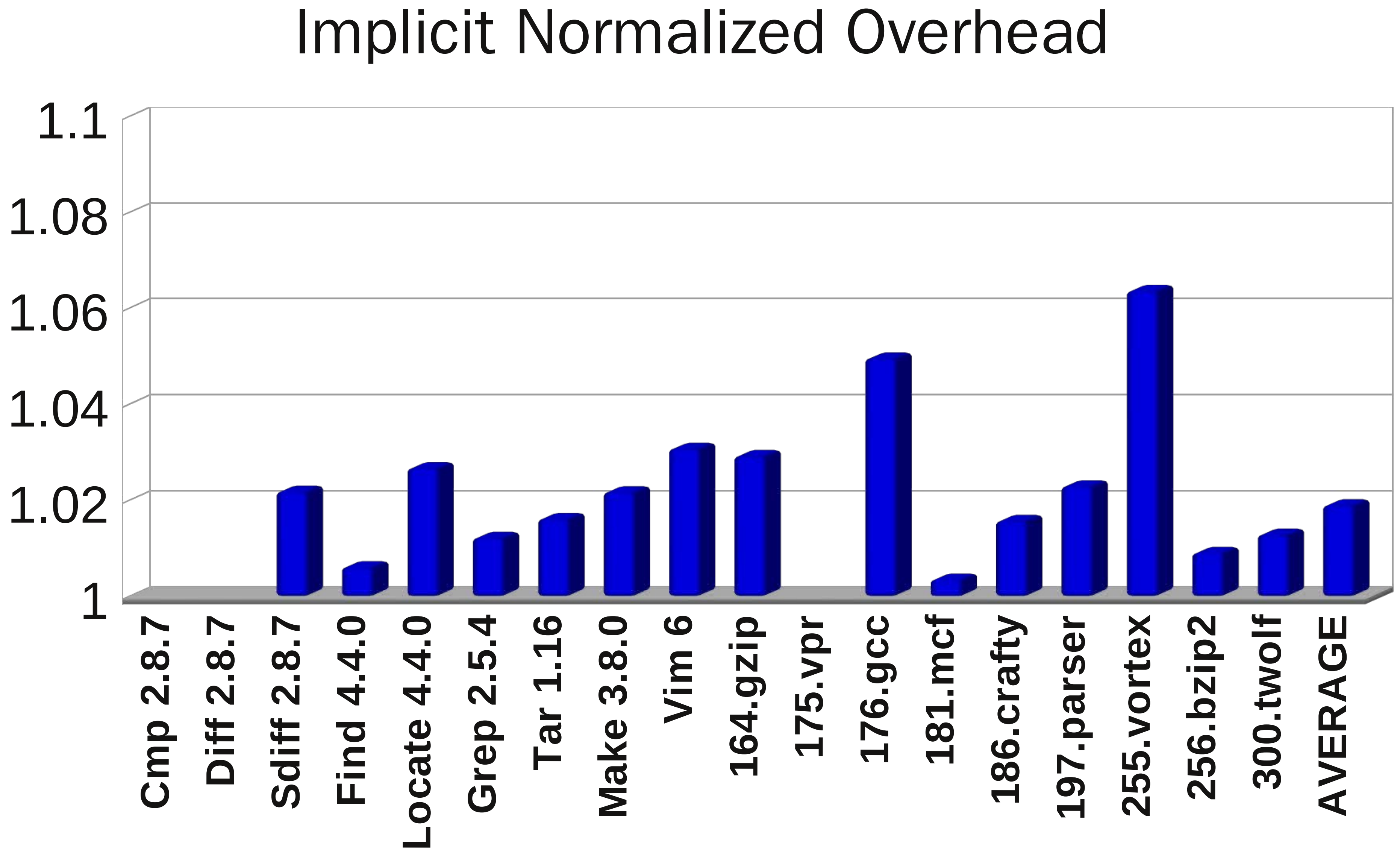
Evaluation: Context Attributes

Program	Max Size		90% Size	
	Ours	Full	Ours	Full
AVERAGE	8.7	39.2	3.2	13.7

Evaluation: Runtime



Evaluation: Runtime



Evaluation

- Our method
  - Basic: 3.6% overhead
  - Hybrid: 1.9% overhead
  - Reversible
  - Multiple integers (1-3 in most cases)
- Compared to Probabilistic:
  - 3% overhead
  - One way
  - One integer

Related Work

Probabilistic Calling Context

[Bond, McKinley OOPSLA'07]

Breadcrumbs

[Bond, Baker, Guyer PLDI'10]

Inferred Call Path Profiling

[Mytkowicz, Coughlin, Diwan OOPSLA'09]

Efficient Path Profiling

[Ball, Larus MICRO'96]

Conclusions

	Lower Overhead	Higher Overhead
Partial Context Info	PCC (Hashing) Breadcrumbs Inferred Call Paths	
Full Context Info		Stack Walking Calling Context Trees

Conclusions

	Lower Overhead	Higher Overhead
Partial Context Info	PCC (Hashing) Breadcrumbs Inferred Call Paths	
Full Context Info	**Precise Calling Context Encoding**	Stack Walking Calling Context Trees

Thank You