Query-Directed Adaptive Heap Cloning for Optimizing Compilers

Yulei Sui, Yue Li, Jingling Xue

Programming Languages and Compilers Group
School of Computer Science and Engineering
University of New South Wales, Australia

February 25, 2013

Heap cloning

Statically distinguishing heap objects by call paths

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```

main

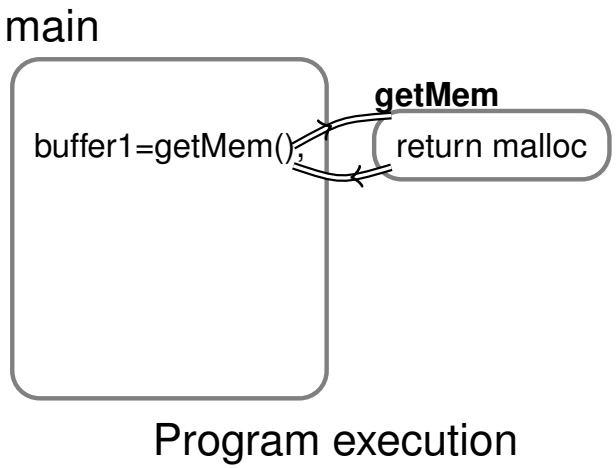
Program execution

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```



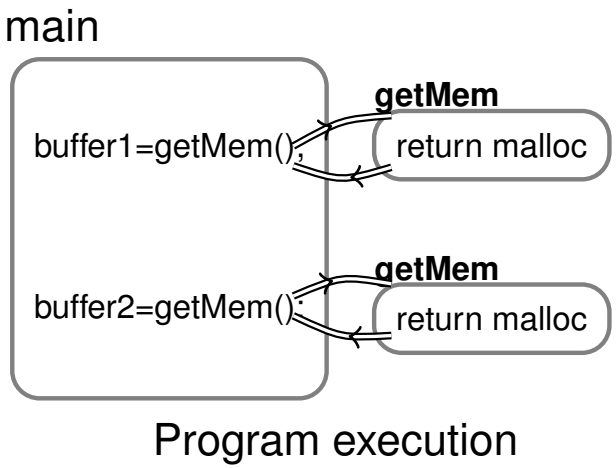
Program execution

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```



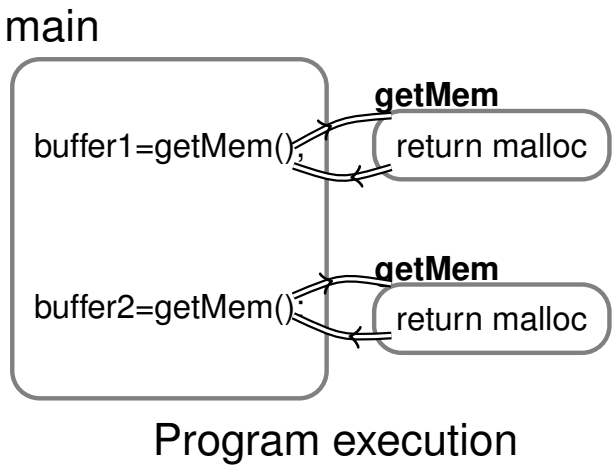
Program execution

Heap cloning

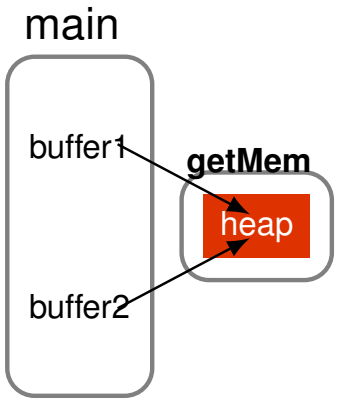
Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```



Program execution



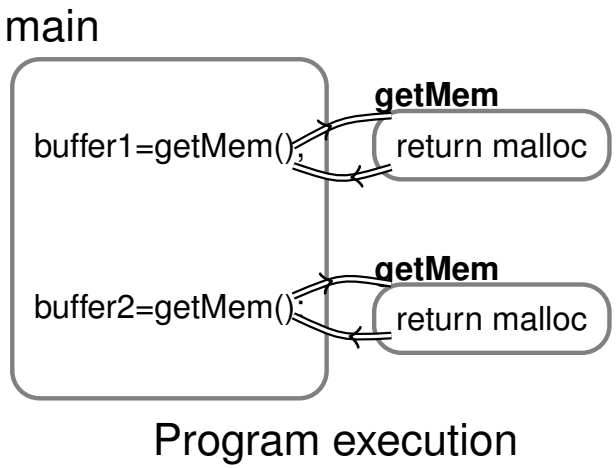
Points-to relations without heap cloning

Heap cloning

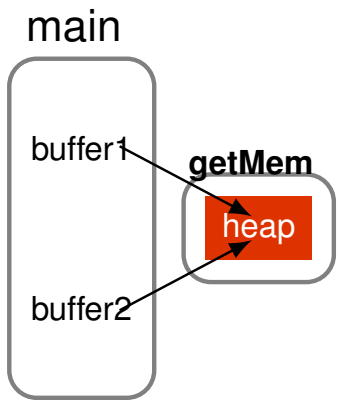
Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

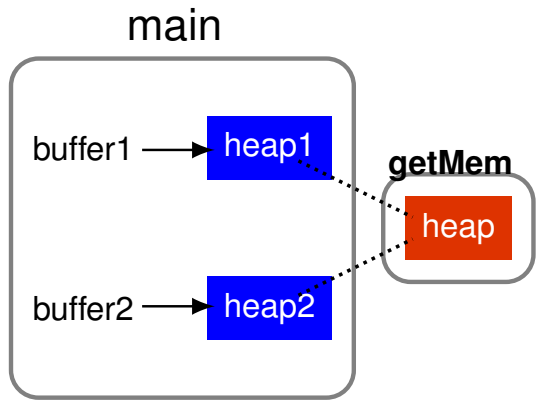
```
int* getMem(){
  return malloc(10);
}
```



Program execution

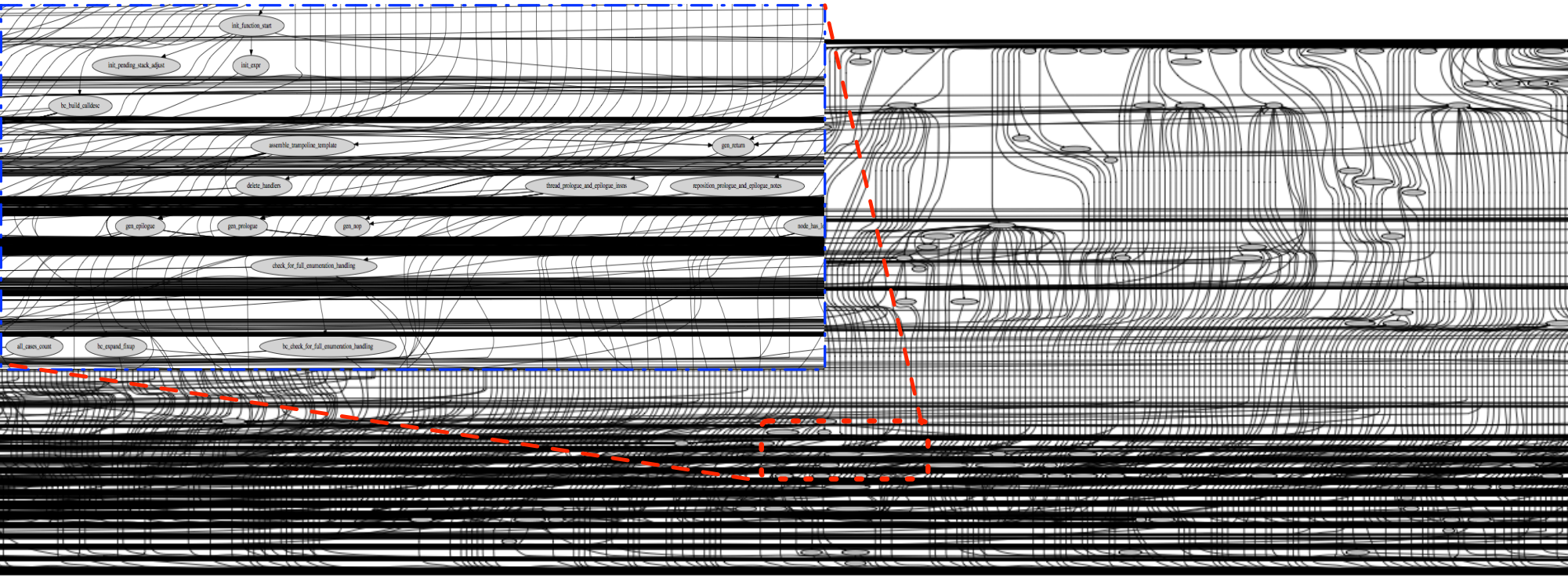


Points-to relations without heap cloning



Points-to relations with heap cloning

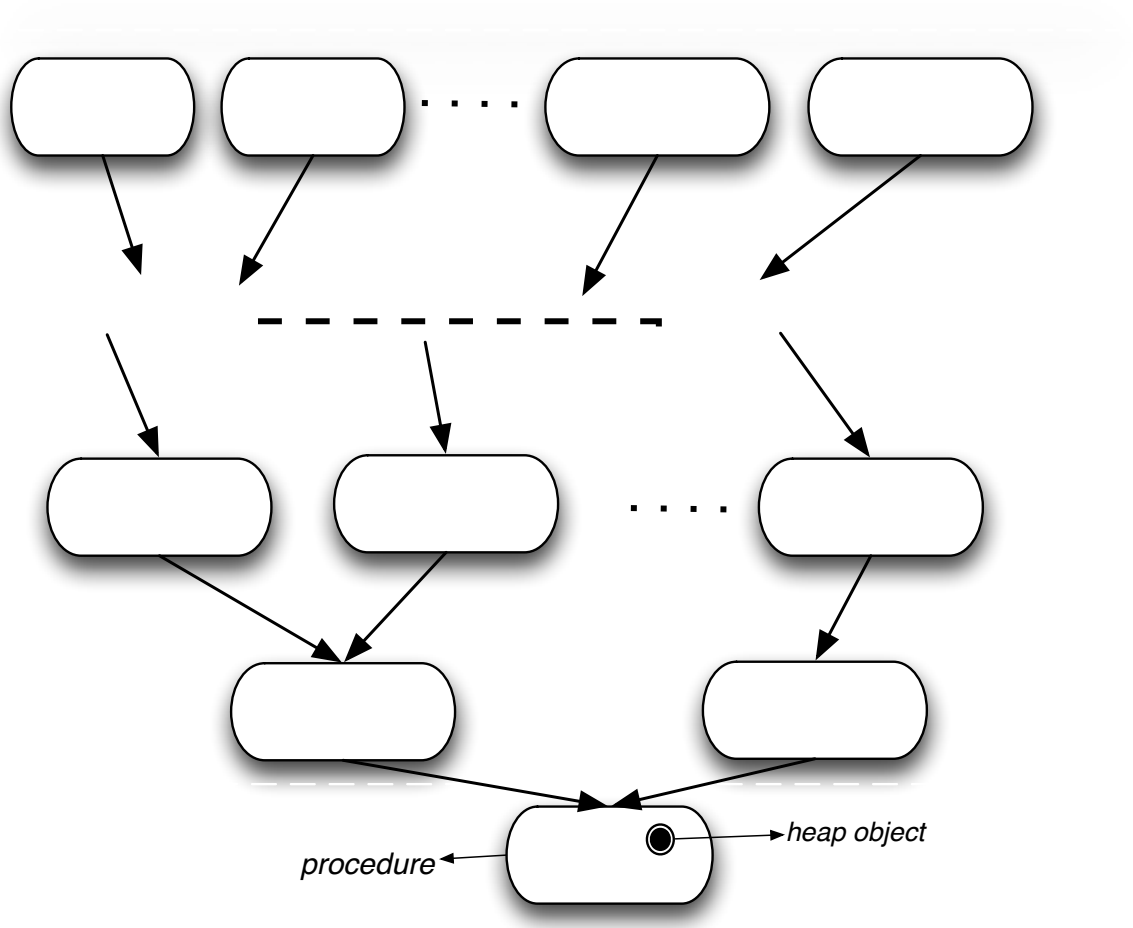
Call graph of 176.gcc (230.4KLOC)



#Procedures: 2256 #Pointers: 134380 #Calling Contexts: $1.2*10^5$

Context-sensitive heap cloning can be costly!

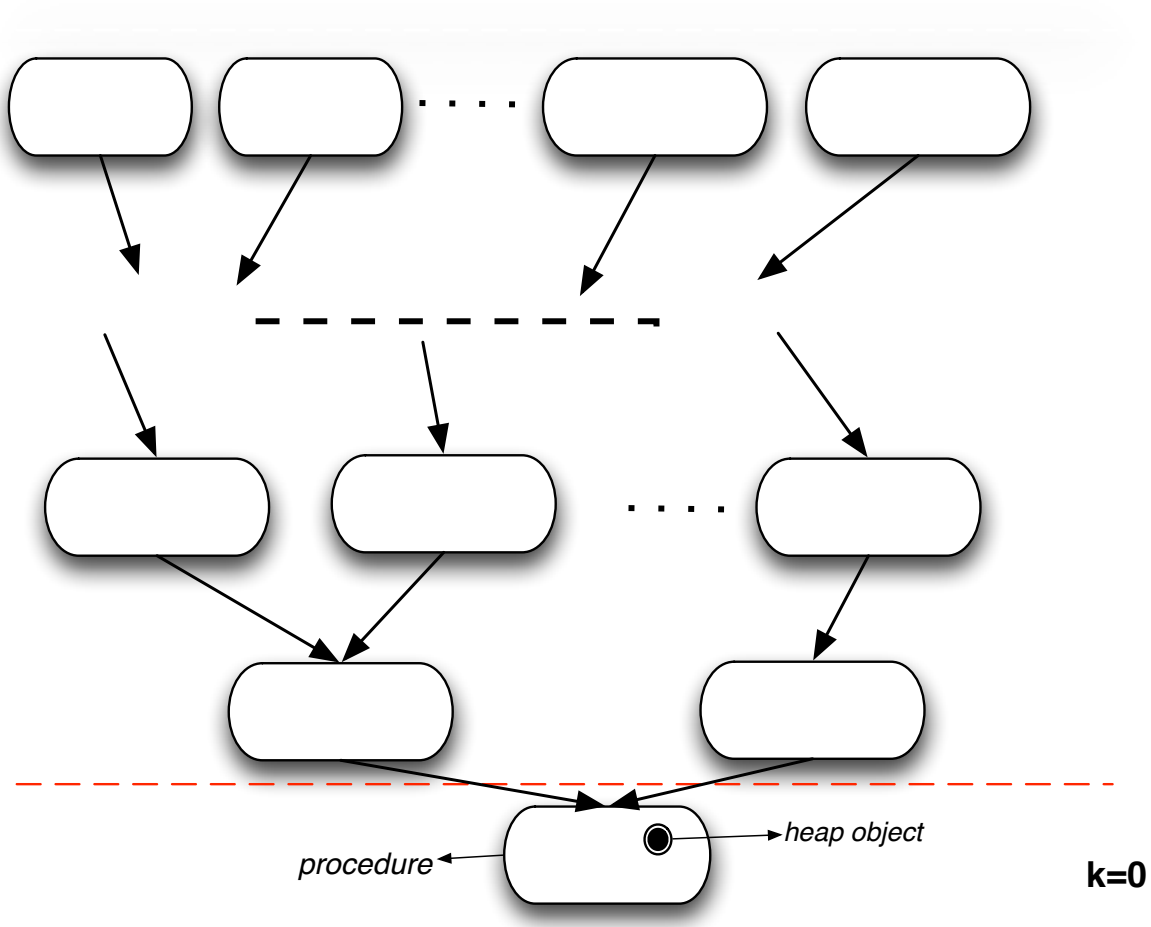
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

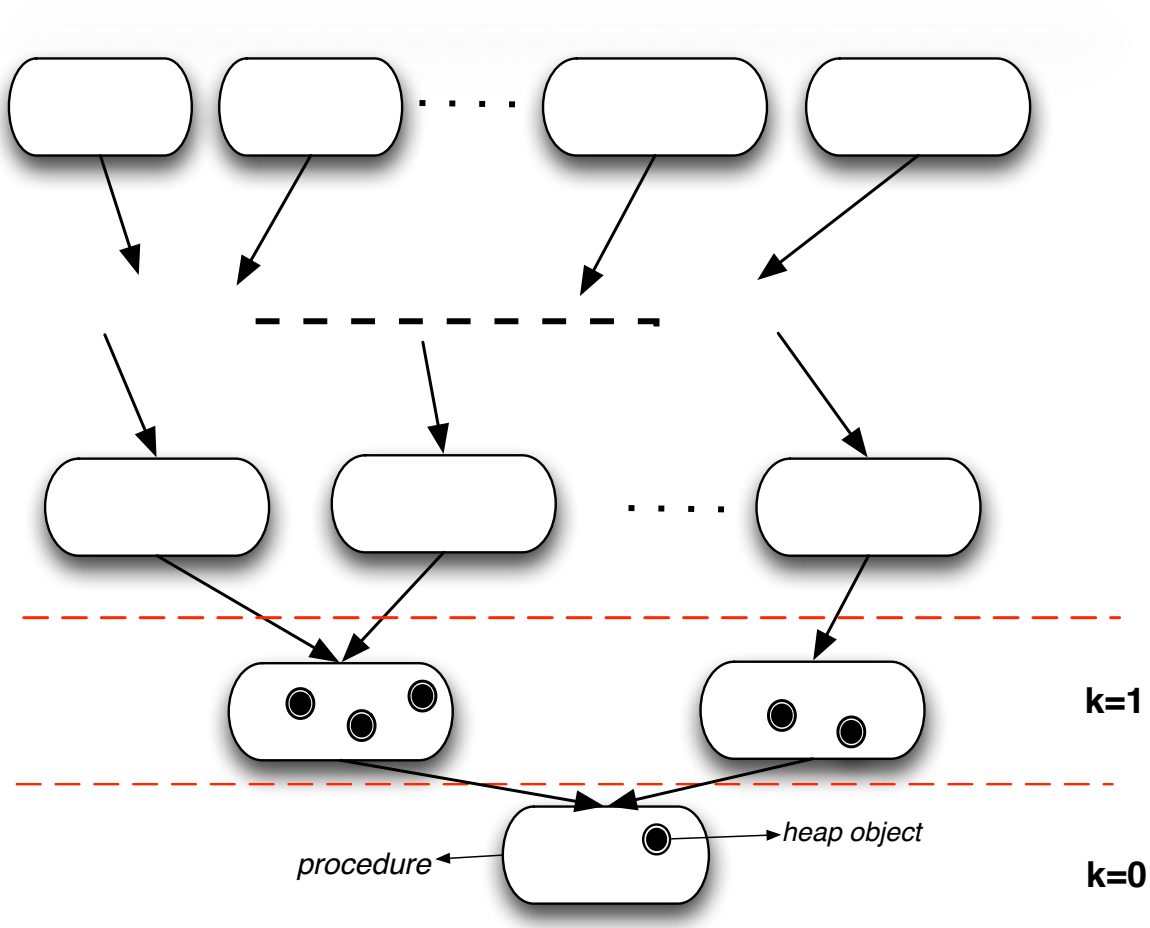
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

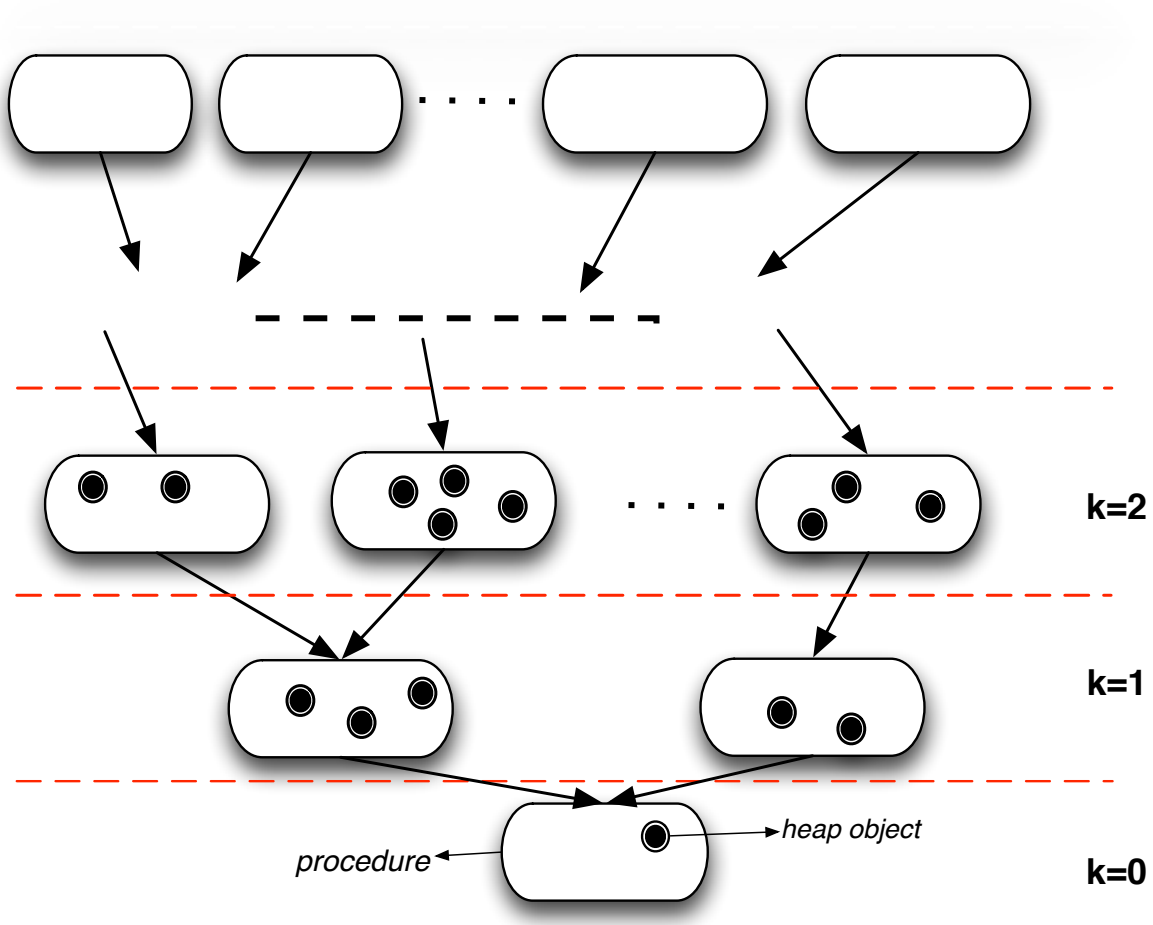
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

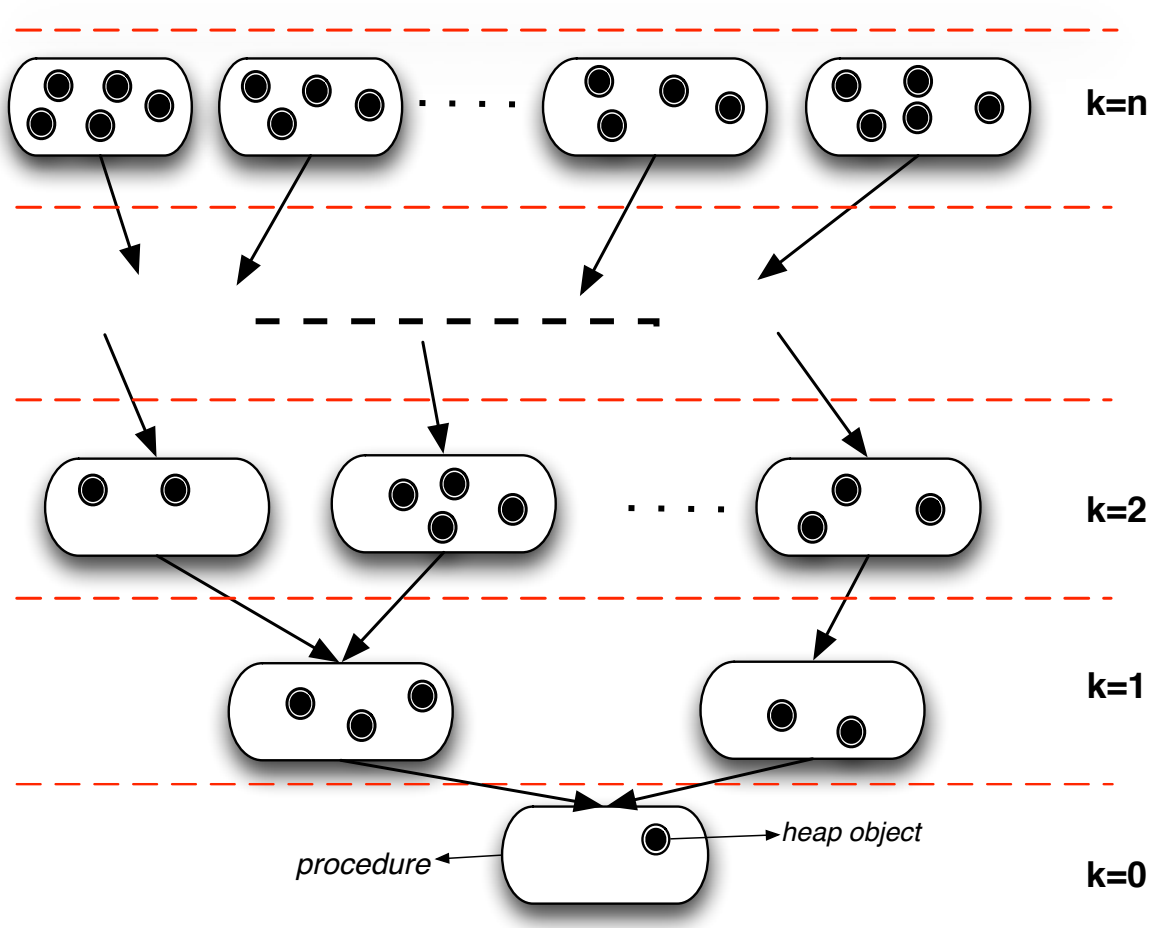
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

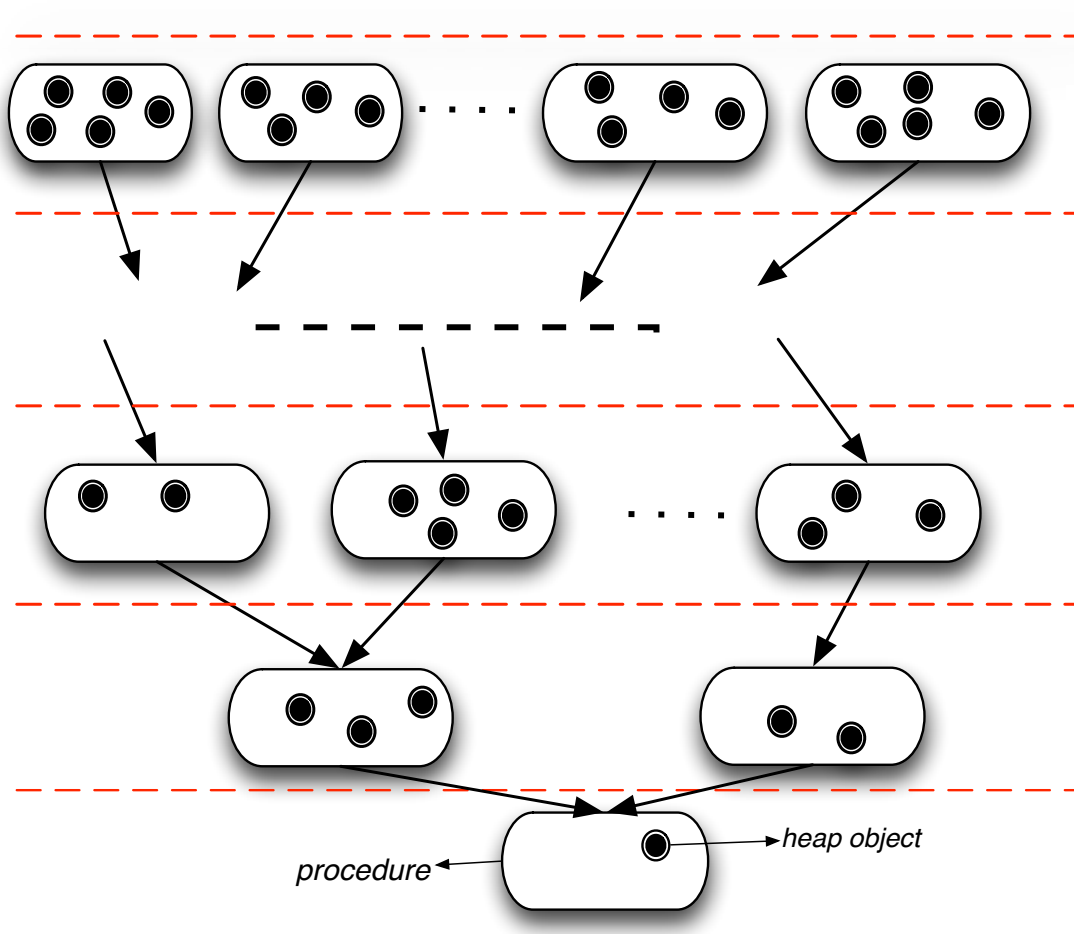
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

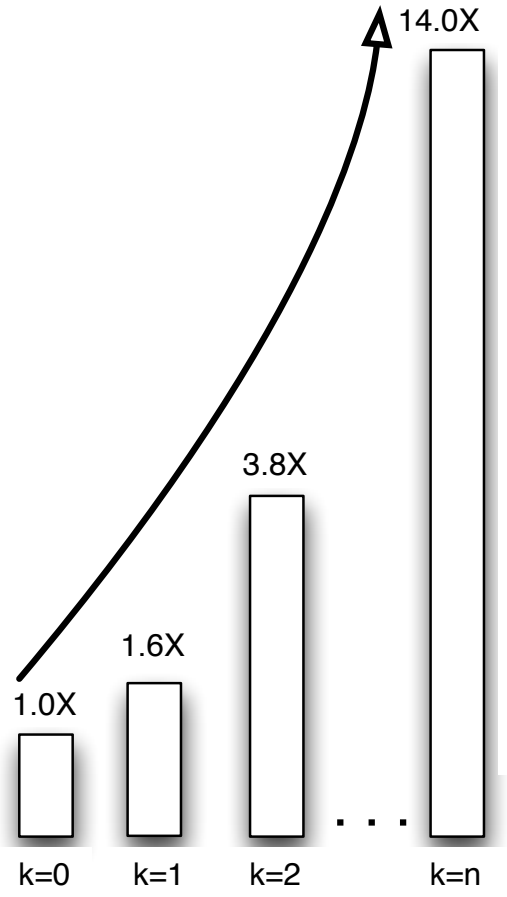
[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

K-callsite-sensitive heap cloning



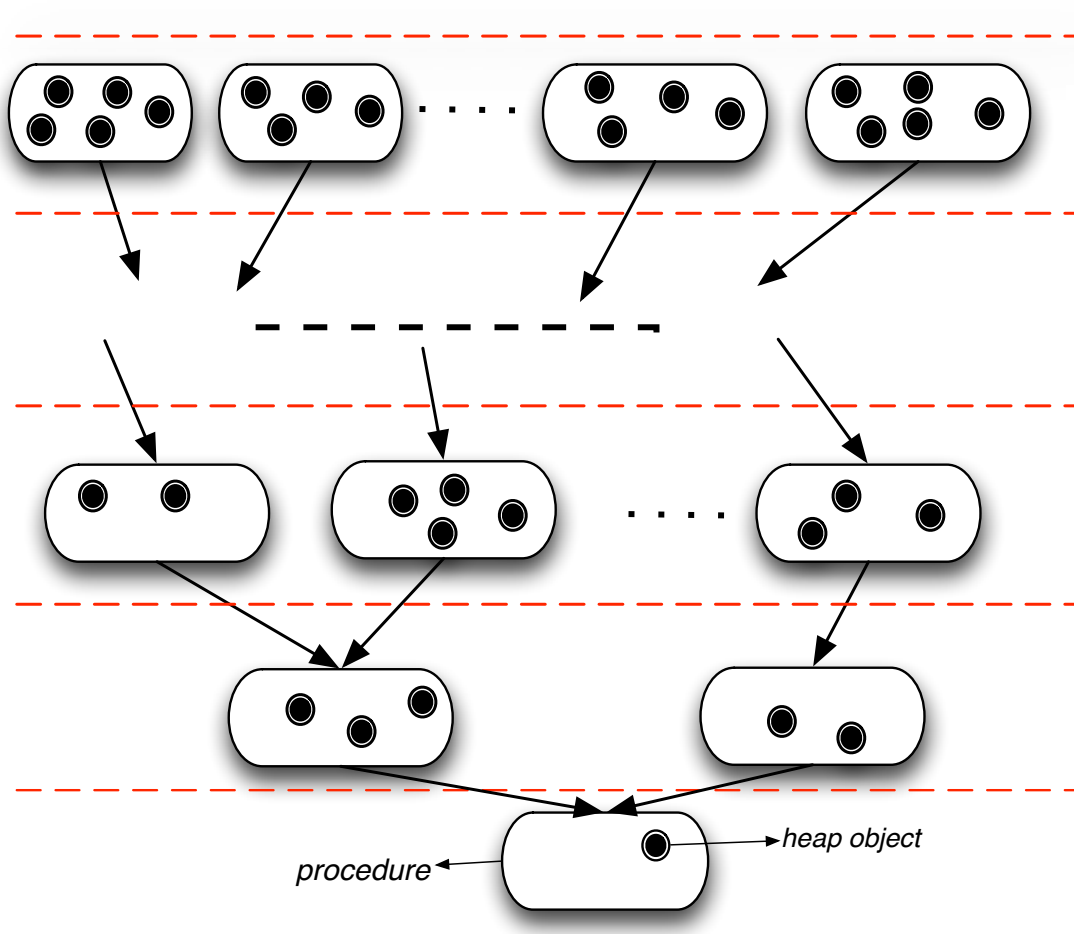
Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]



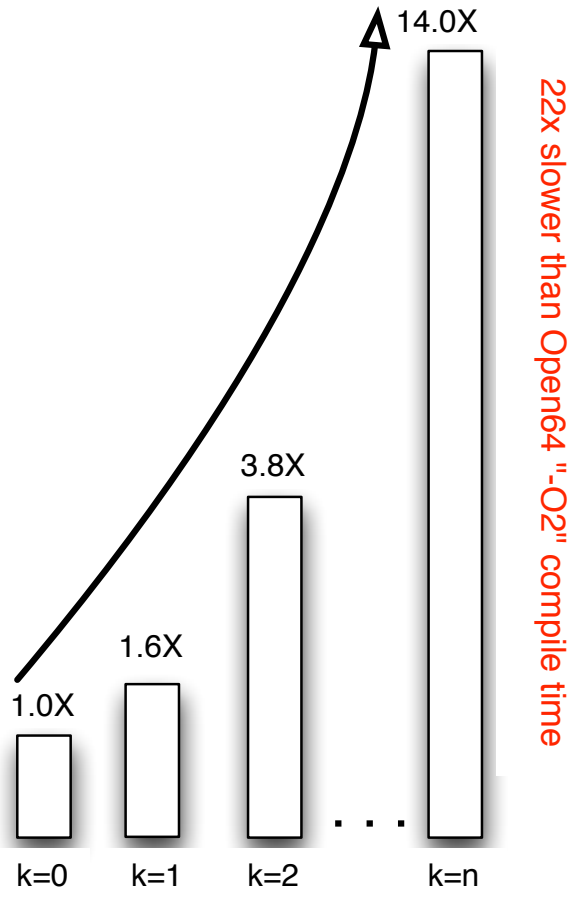
Analysis time of 176.gcc with Andersen-style context sensitive heap cloning

K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]



Analysis time of 176.gcc with Andersen-style context sensitive heap cloning

- Is *full heap cloning* overkill (relative to a client's needs)?

- Is *full heap cloning* overkill (relative to a client's needs)?

- Is *k-callsite sensitive cloning* the best solution?

Alias Query

- Whether two expressions may represent the same memory location.

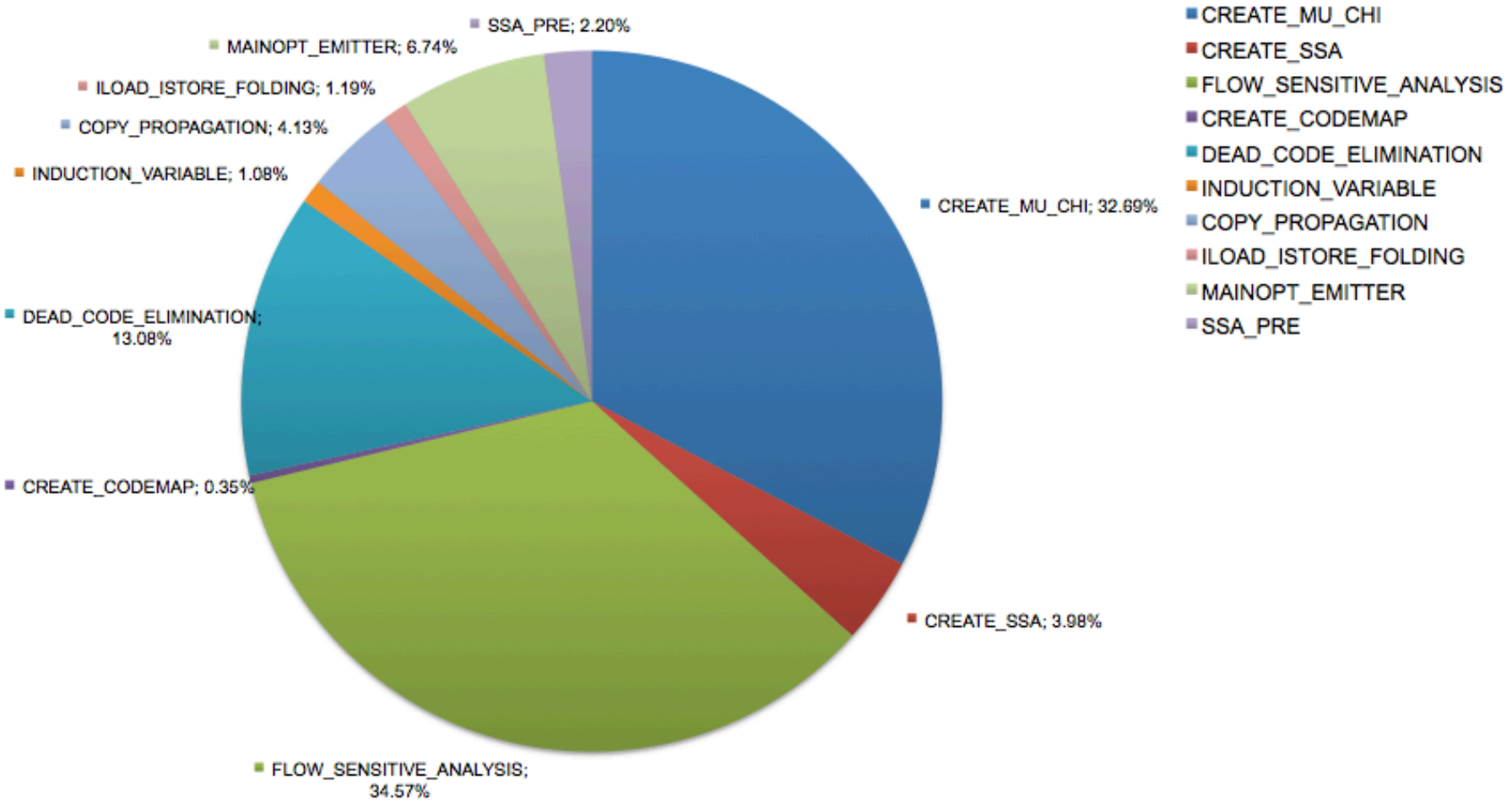
Alias Query

- Whether two expressions may represent the same memory location.
- For example: ⟨*buffer1,*buffer2⟩
  - Alias Without Heap Cloning

Alias Query

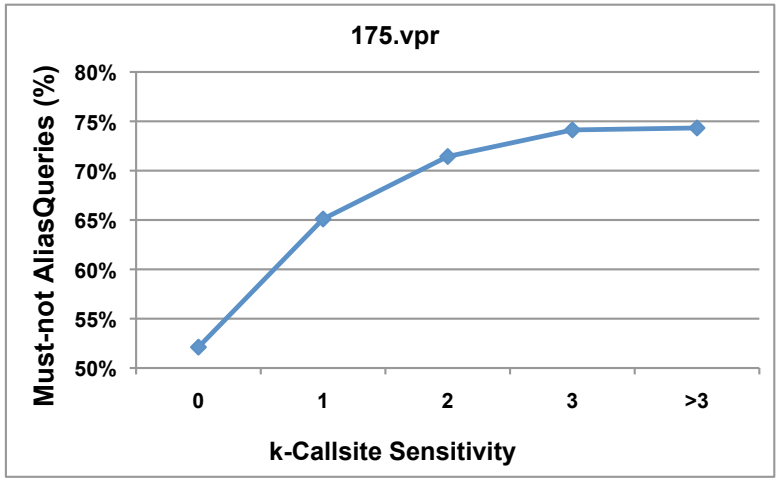
- Whether two expressions may represent the same memory location.
- For example: ⟨*buffer1,*buffer2⟩
  - Alias Without Heap Cloning
  - Not-Alias Heap Cloning

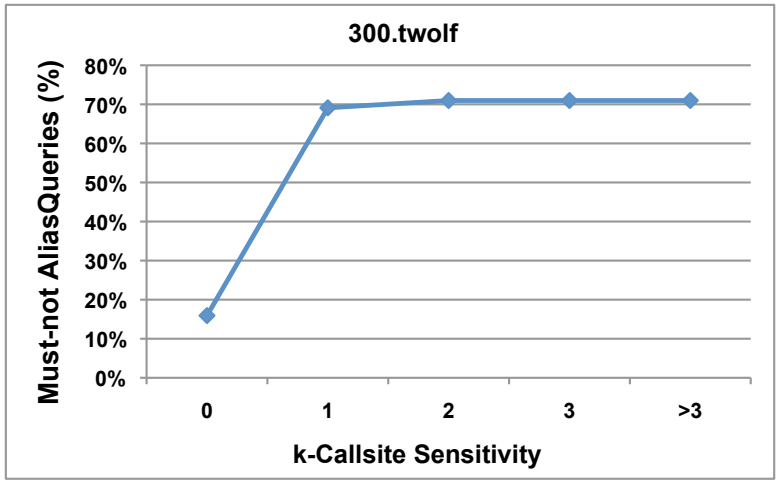
Alias queries in open64 WOPT

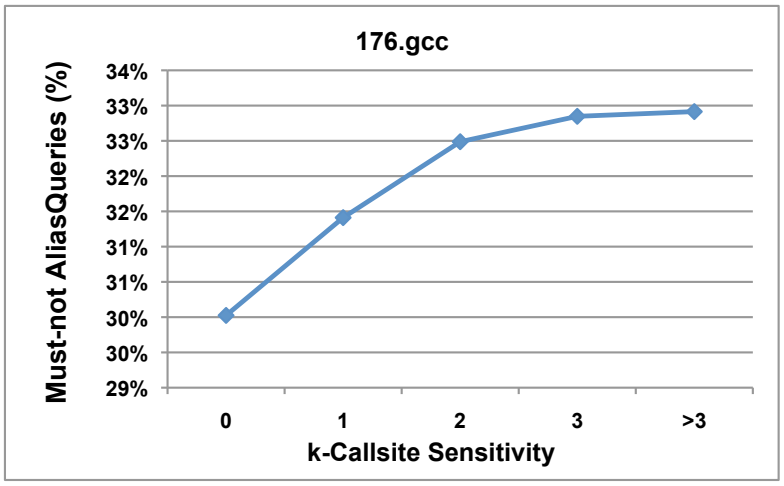


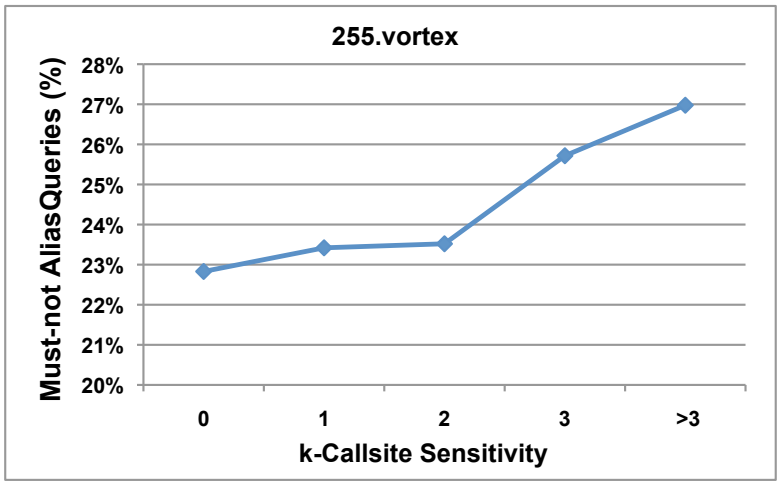
A total of 496304 alias queries generated from WOPT in Open64 for analyzing SPEC2000 C benchmarks

Analysis precision for answering alias queries



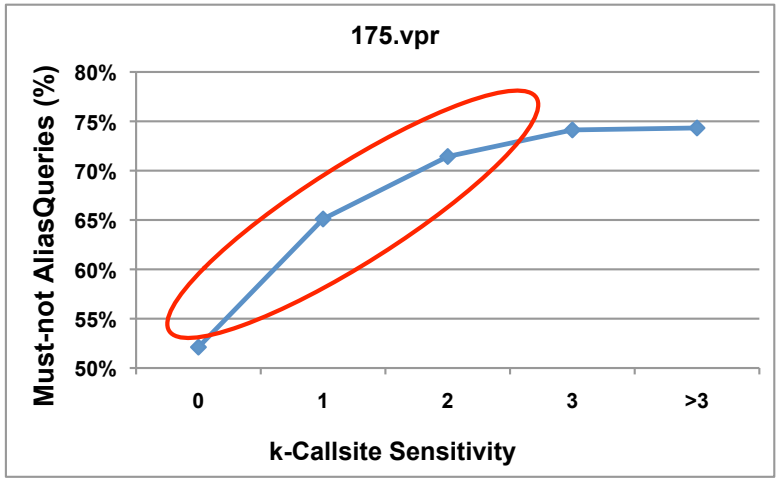


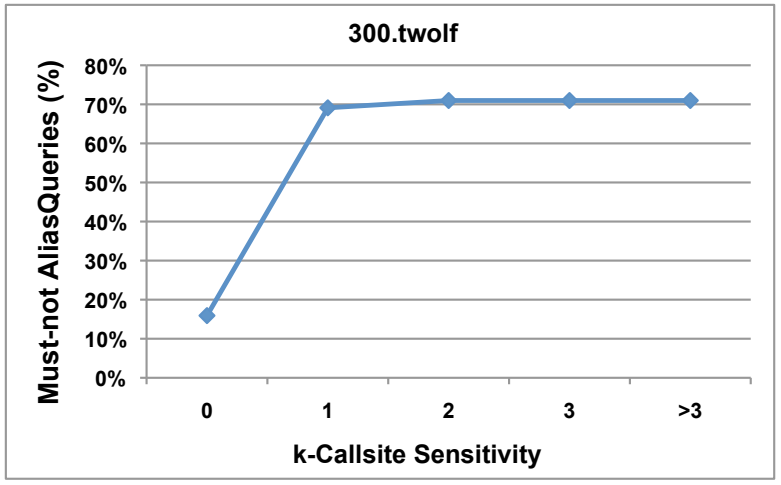


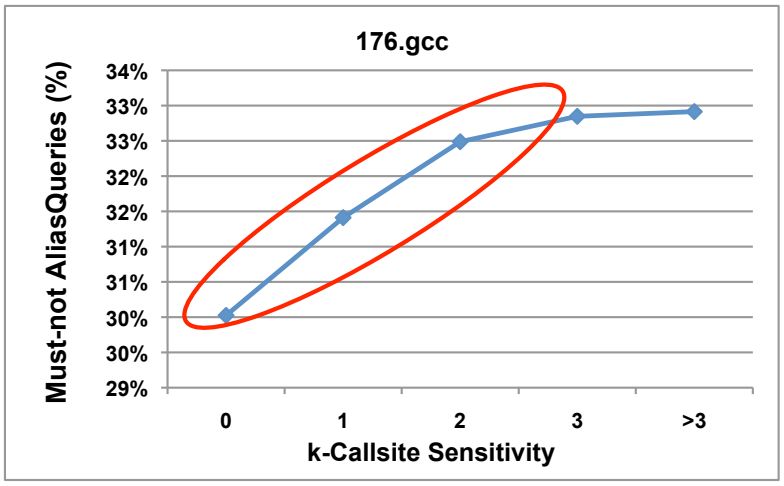


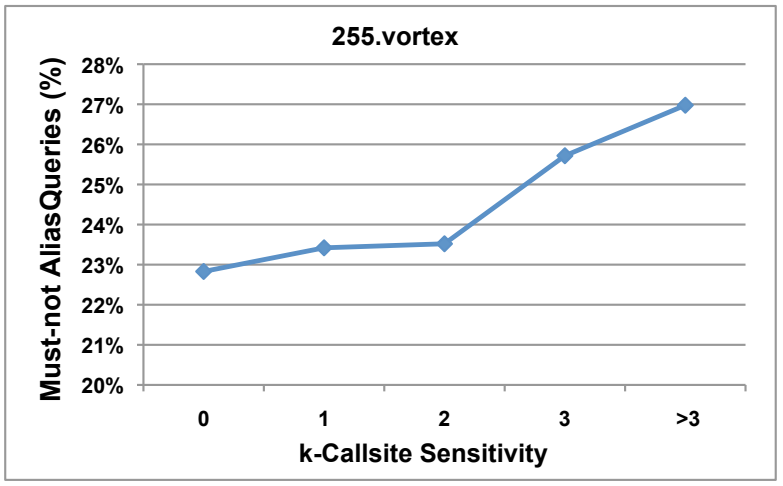
Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning

Analysis precision for answering alias queries



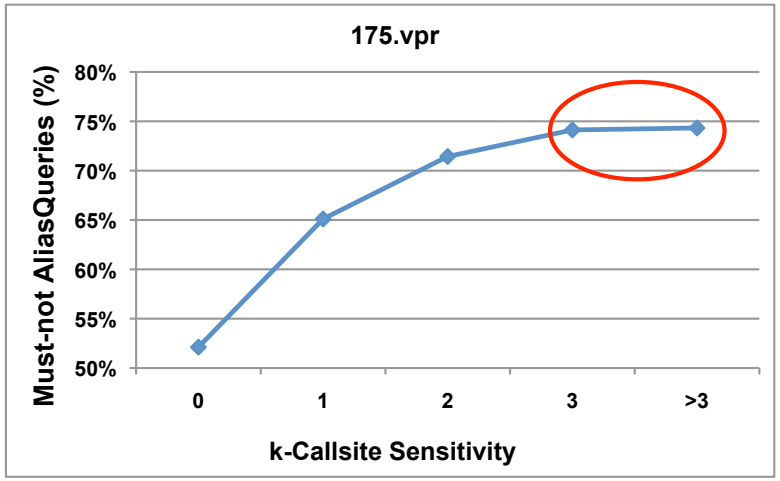


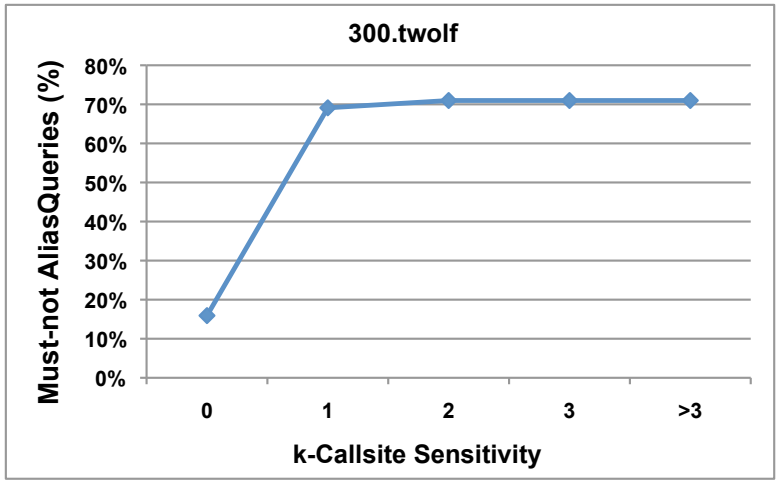


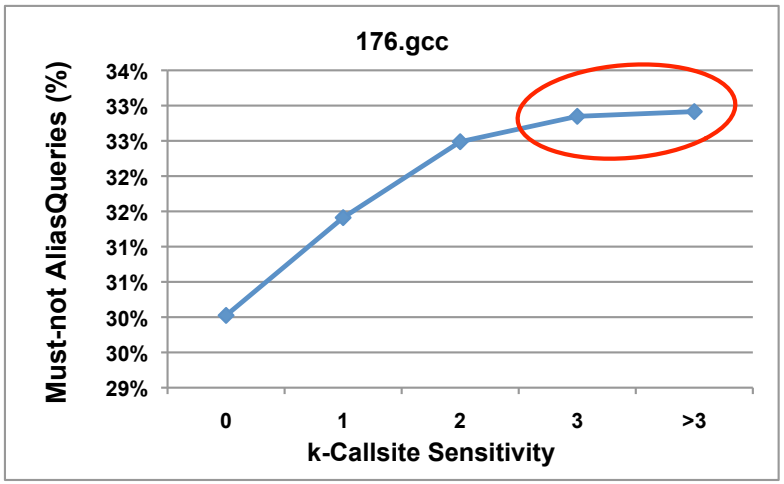


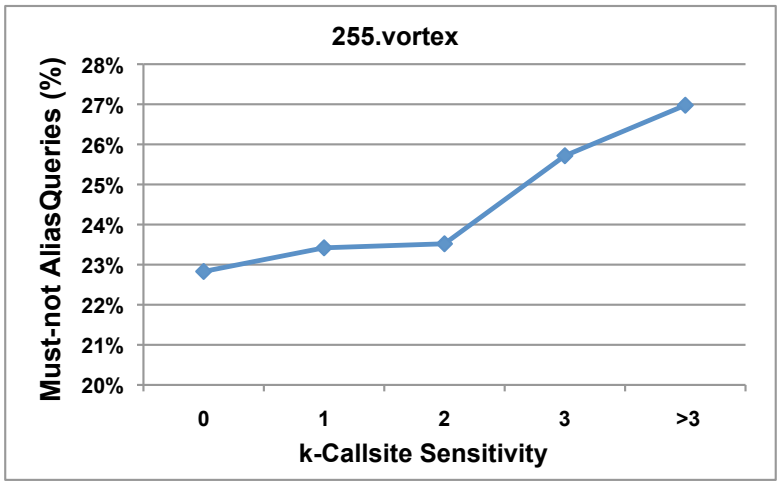
Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning.

Analysis precision for answering alias queries



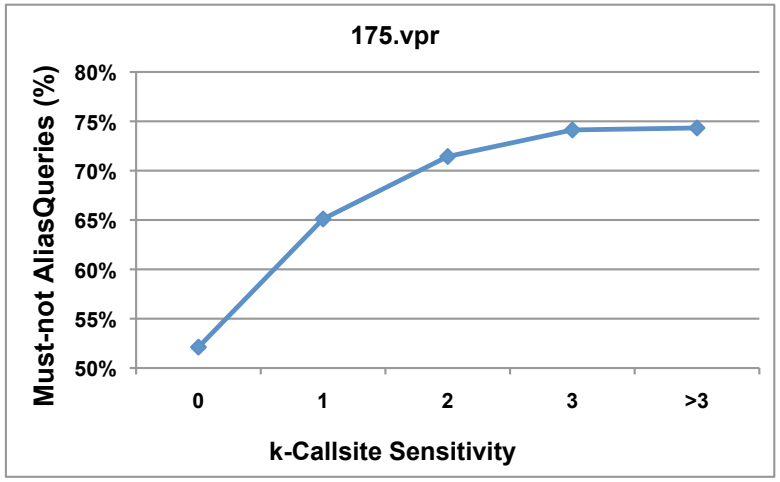


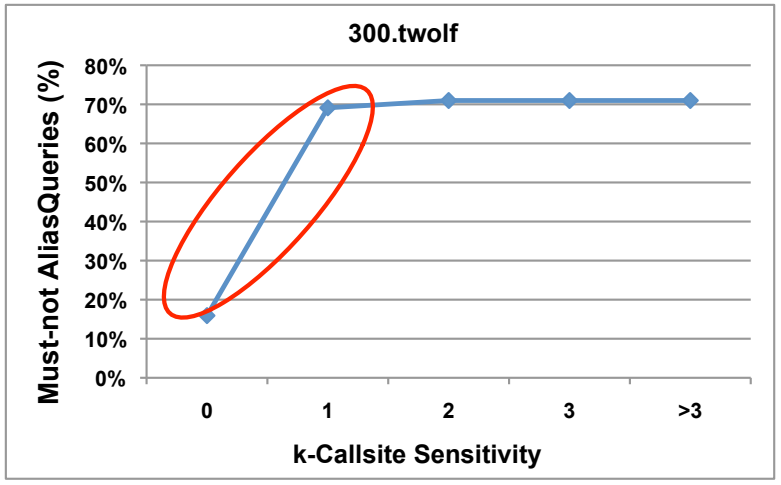


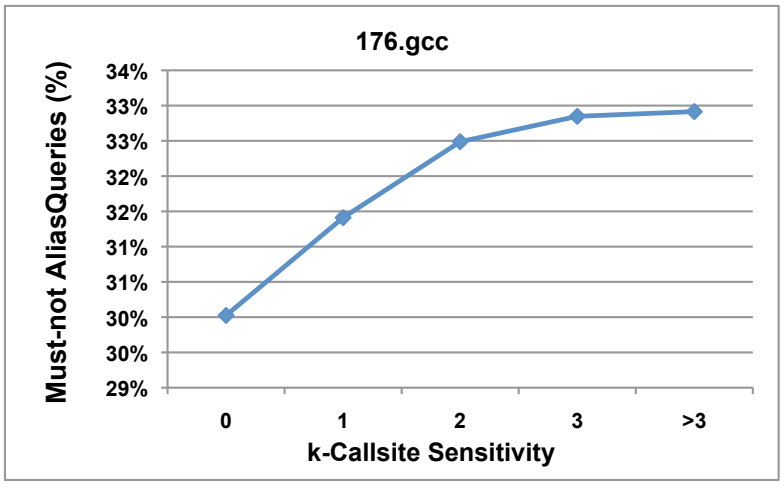


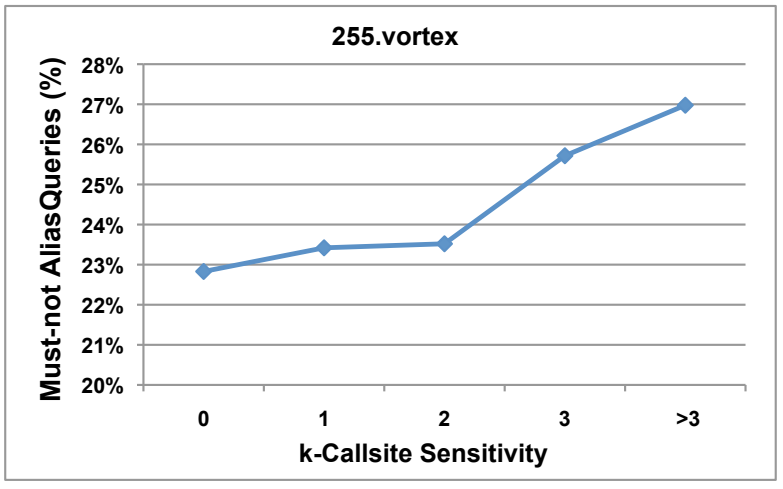
Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning.

Analysis precision for answering alias queries



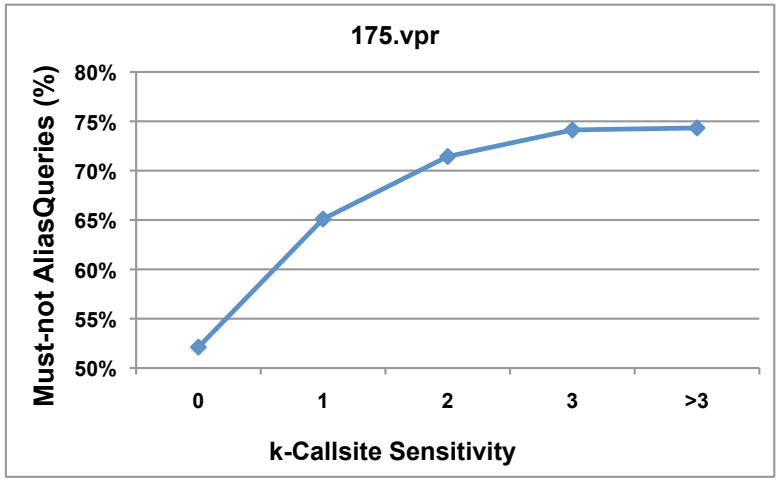


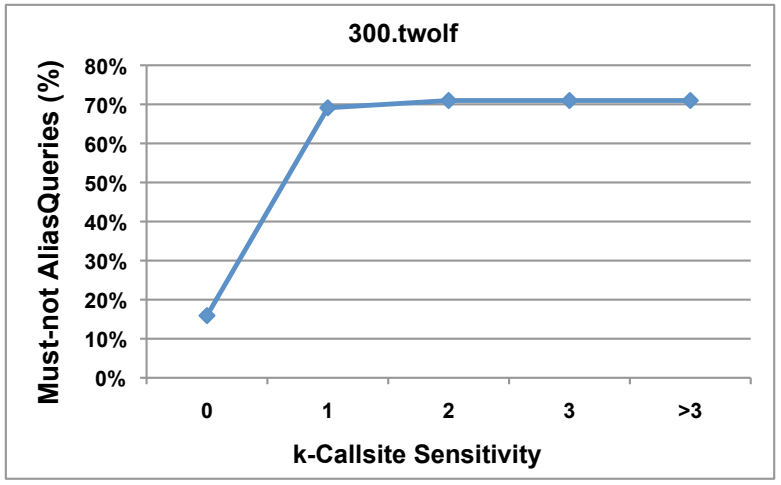


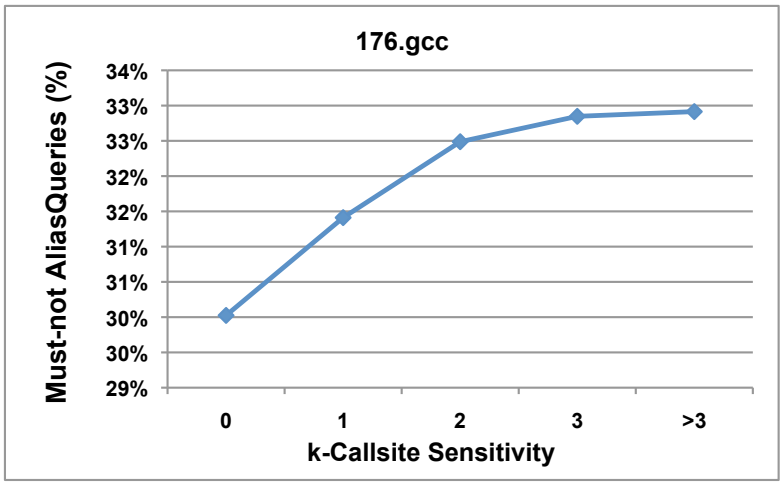


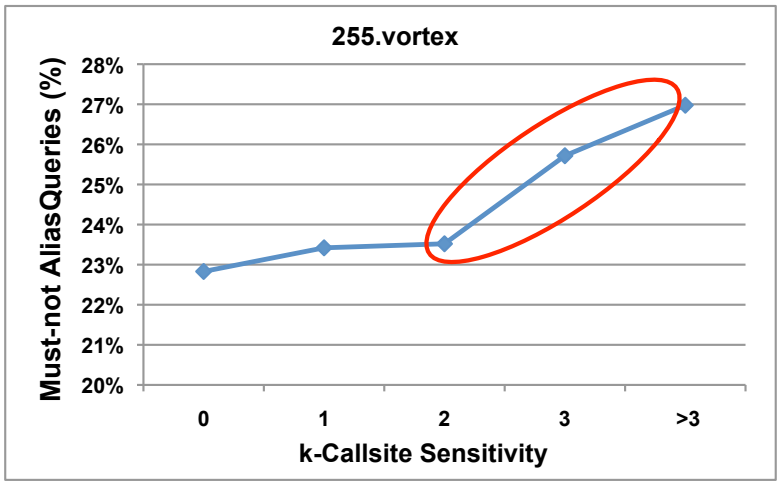
Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning.

Analysis precision for answering alias queries



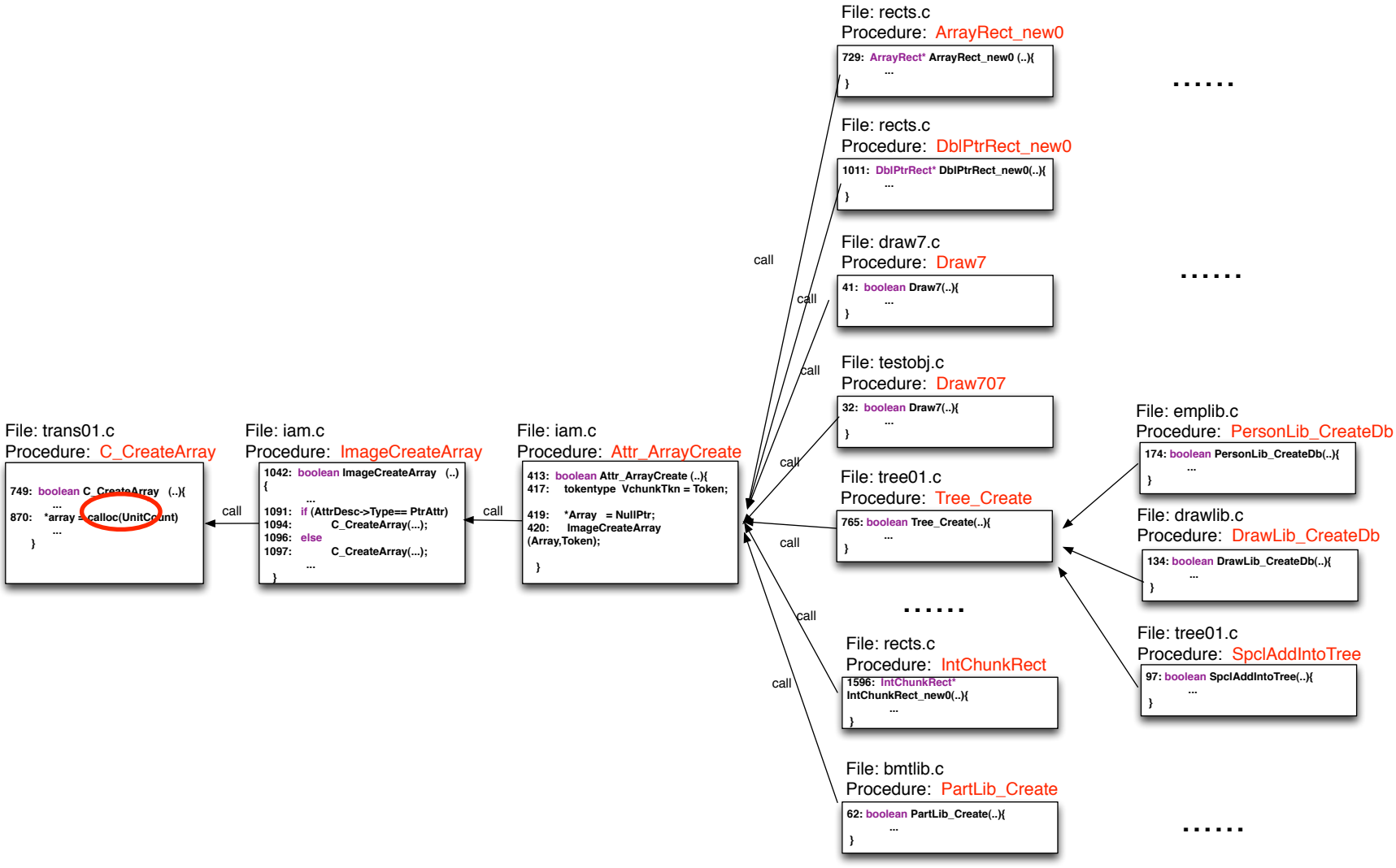




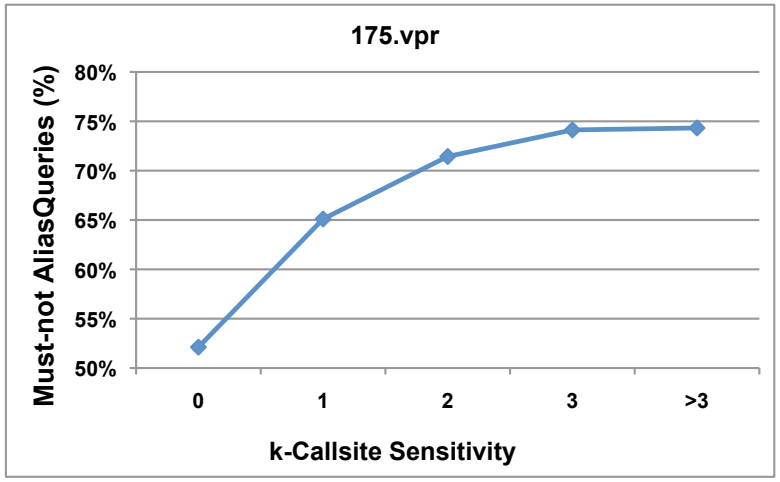


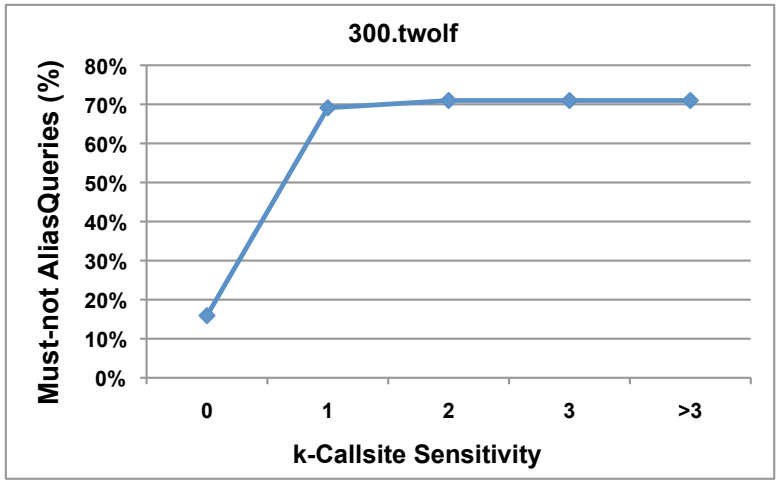
Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning.

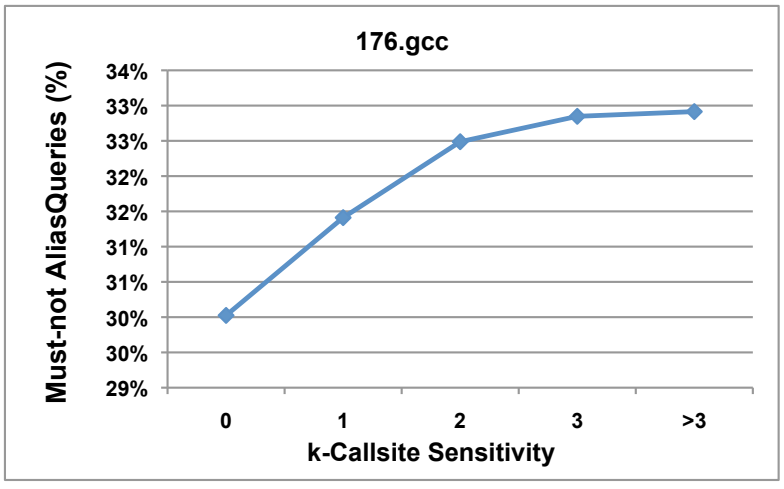
A close look at 255.vortex's call graph

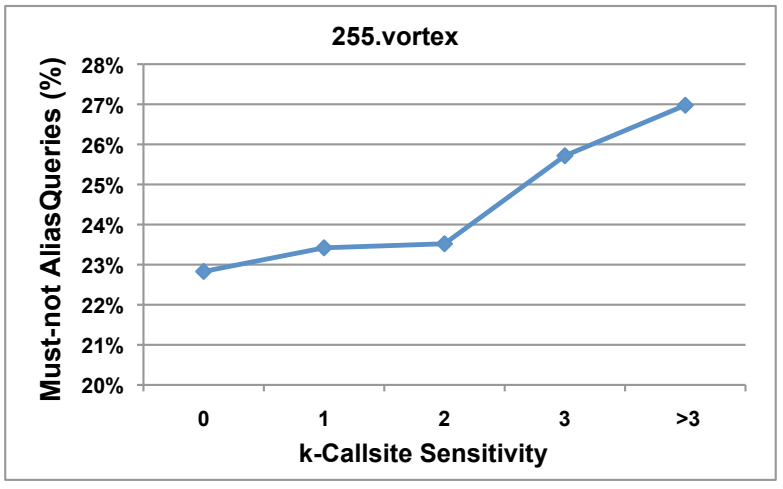


Analysis precision for answering alias queries









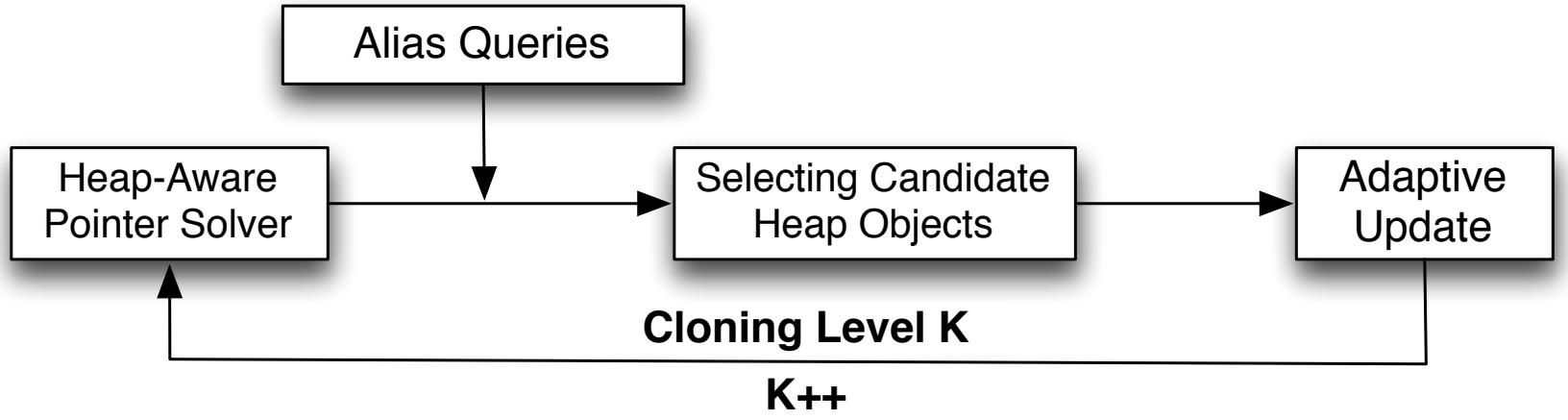
Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning.

Goal

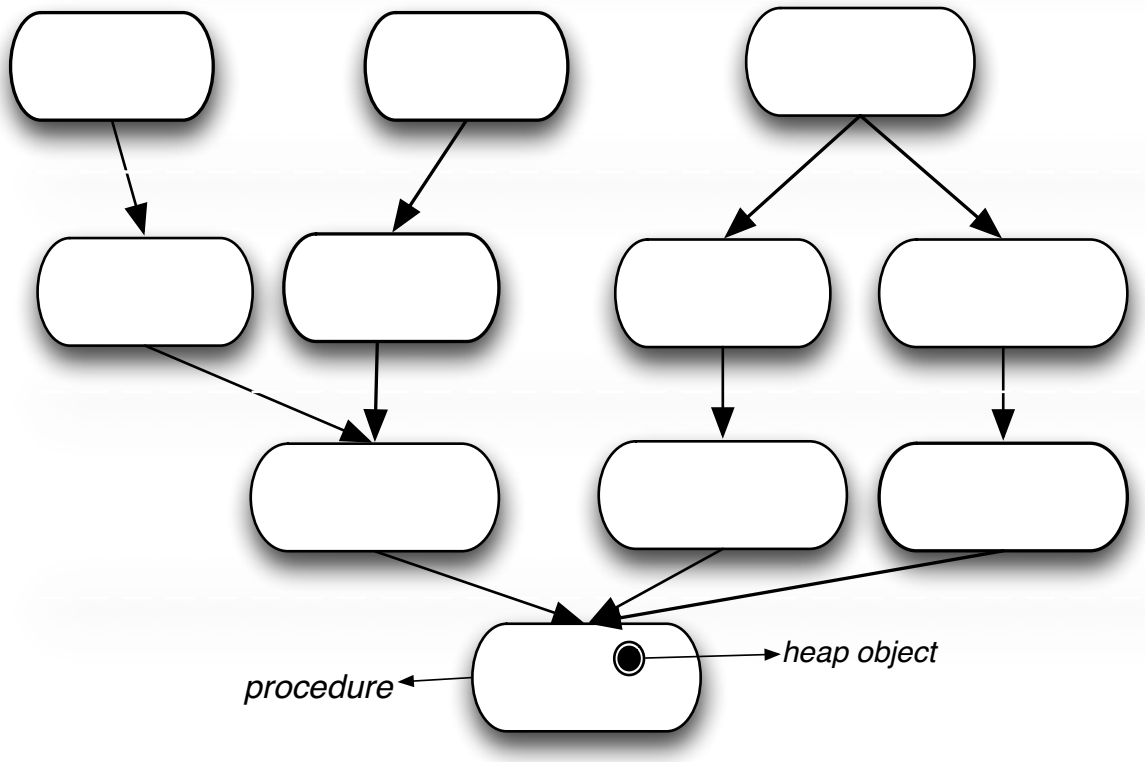
- Can we enable heap cloning only where it is necessary?
- Can we achieve the same precision as full heap cloning according to a client's needs?

Our QUDA framework

QUery-Directed Adaptive heap cloning

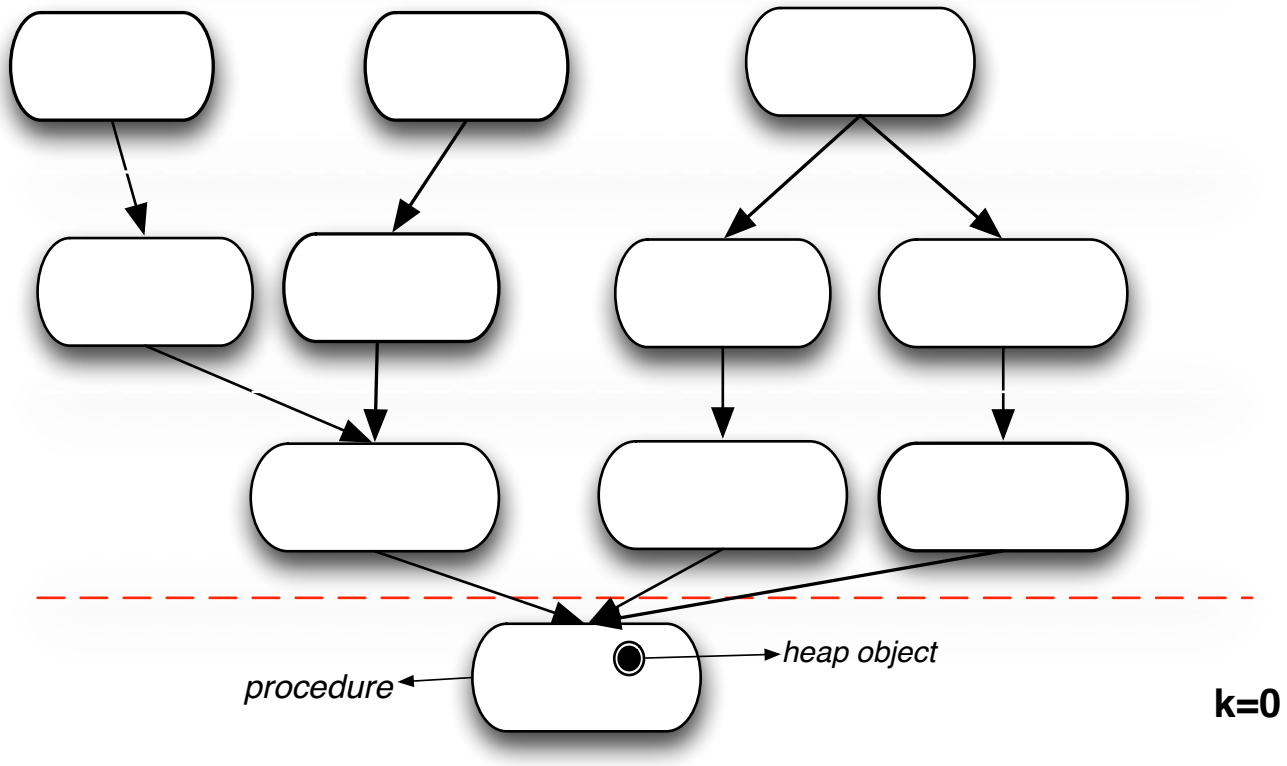


Query-directed adaptive heap cloning



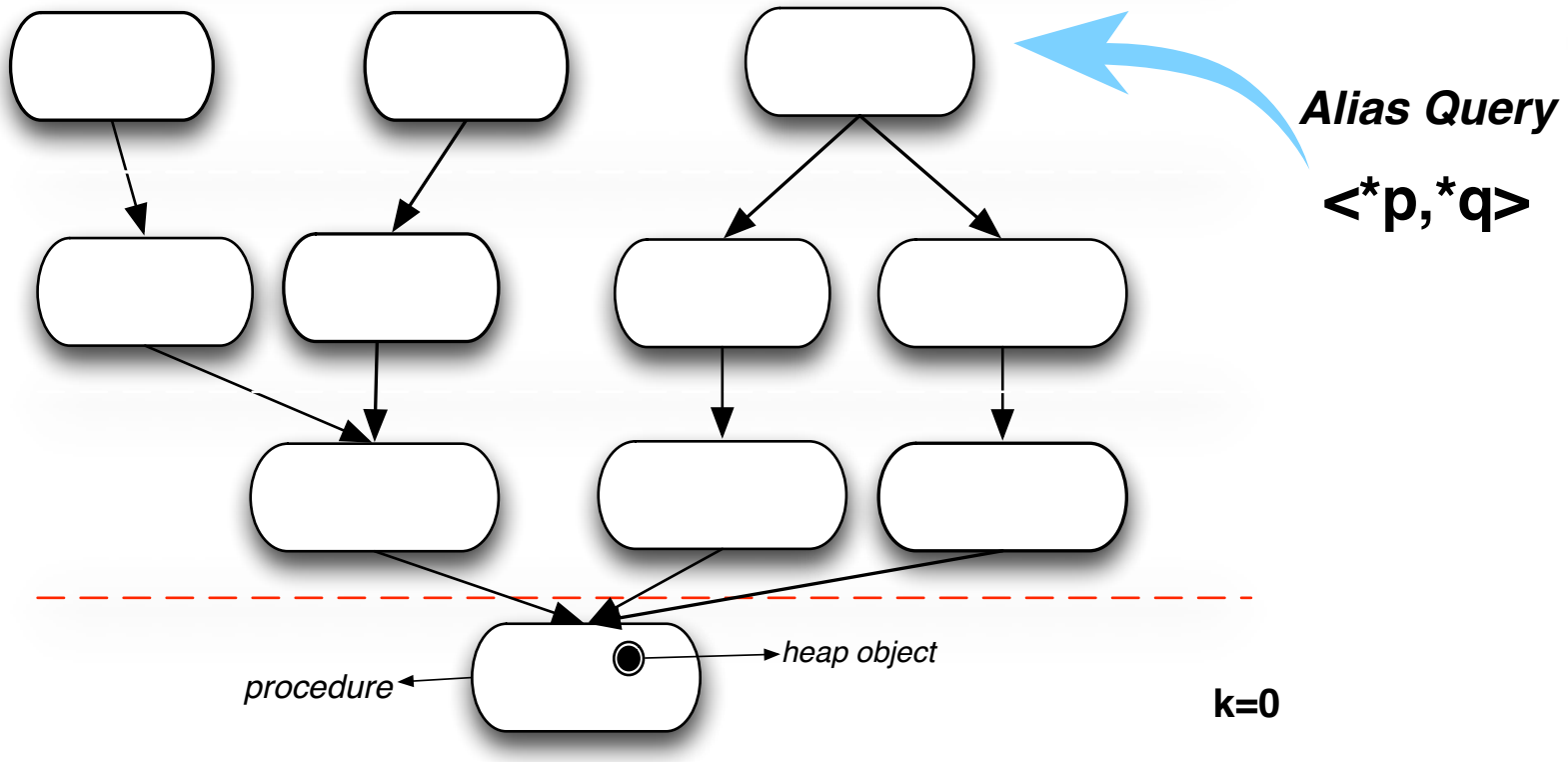
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



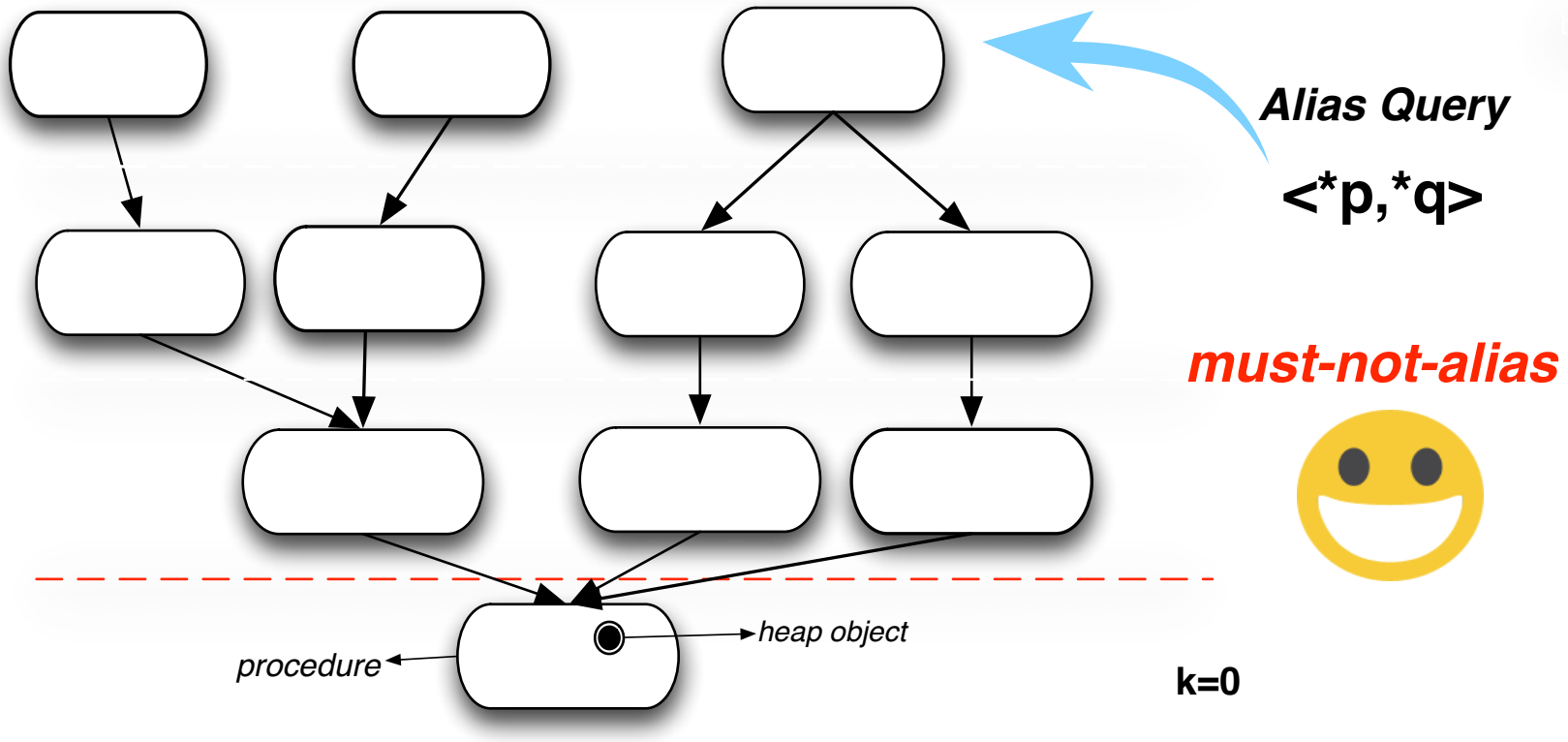
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



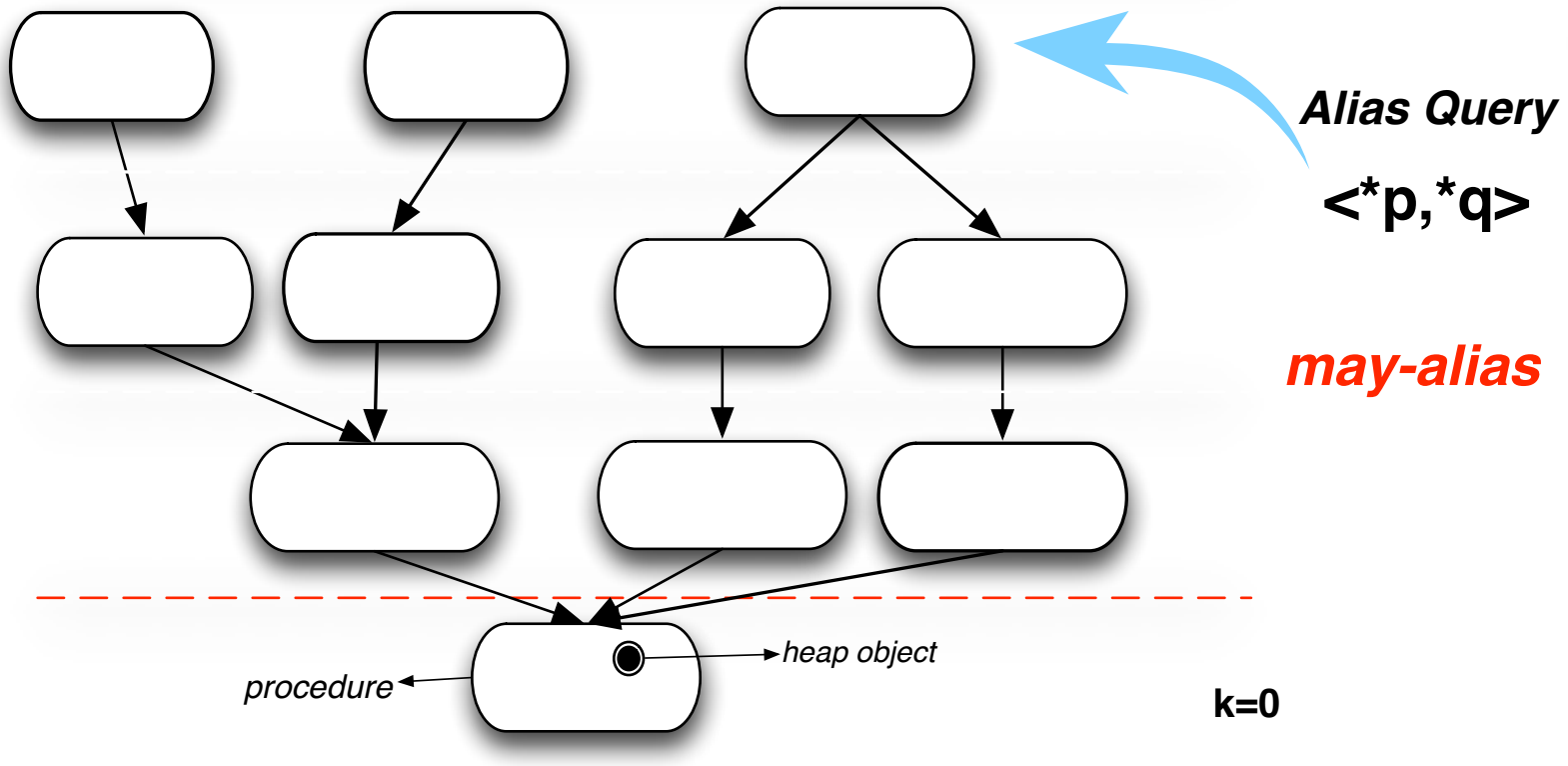
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



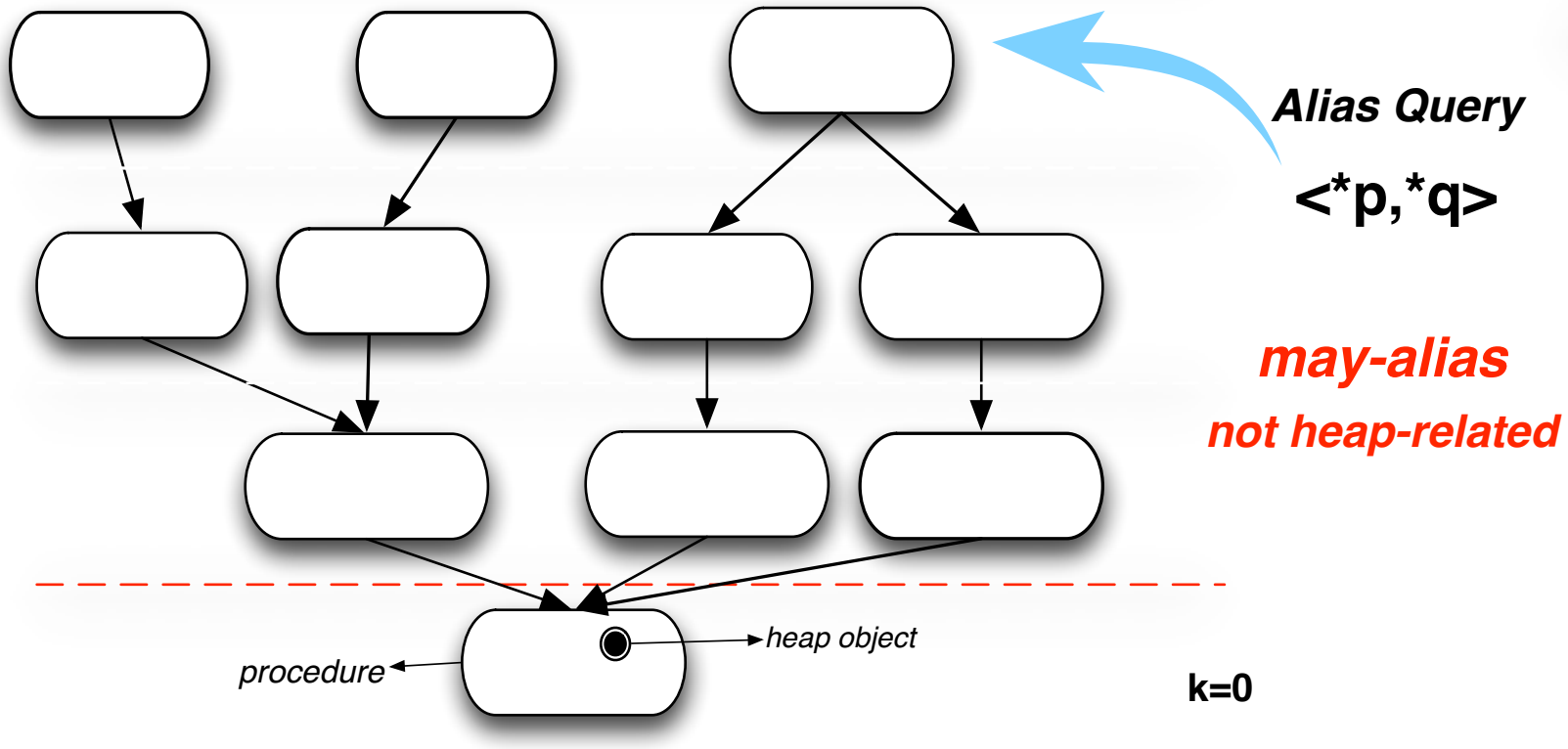
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



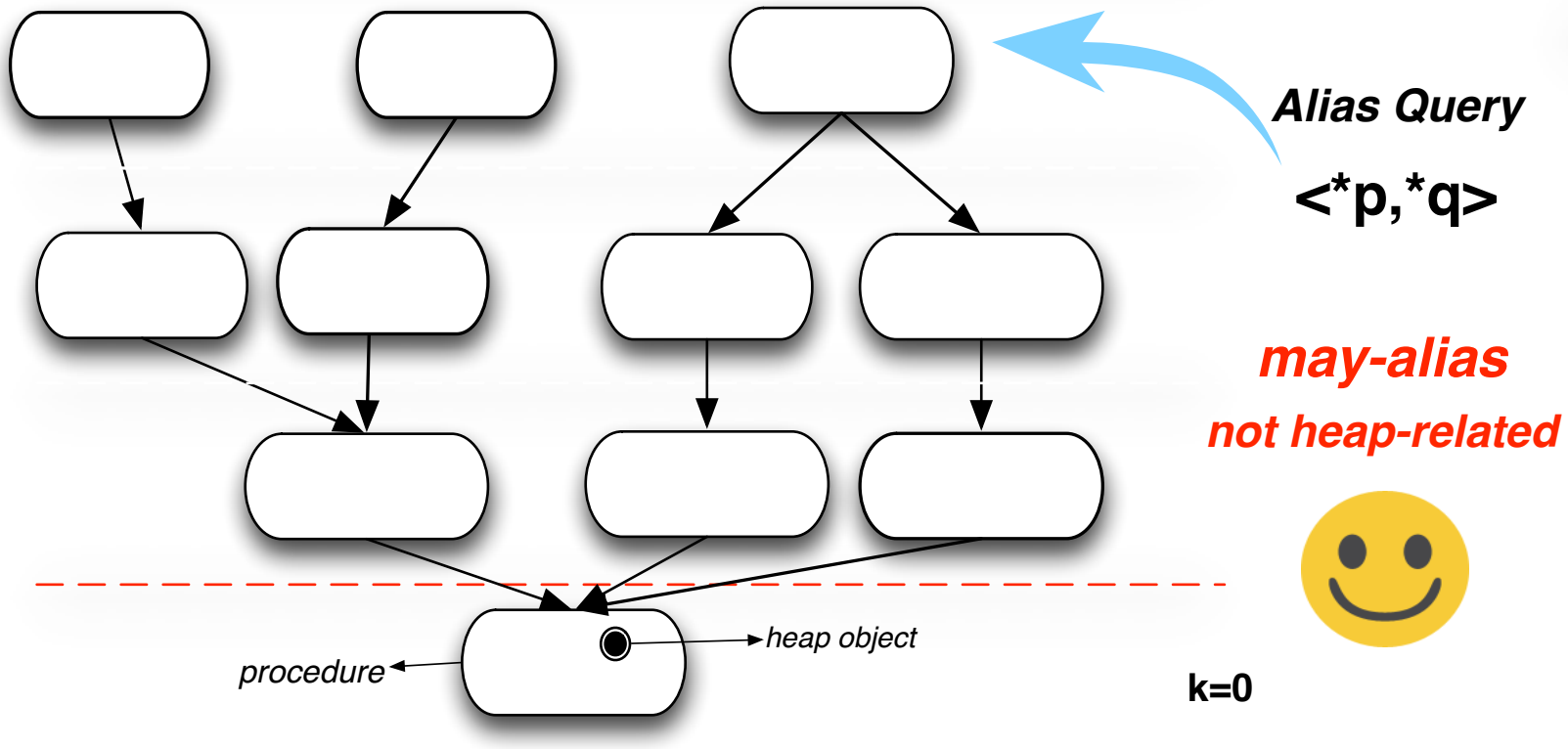
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



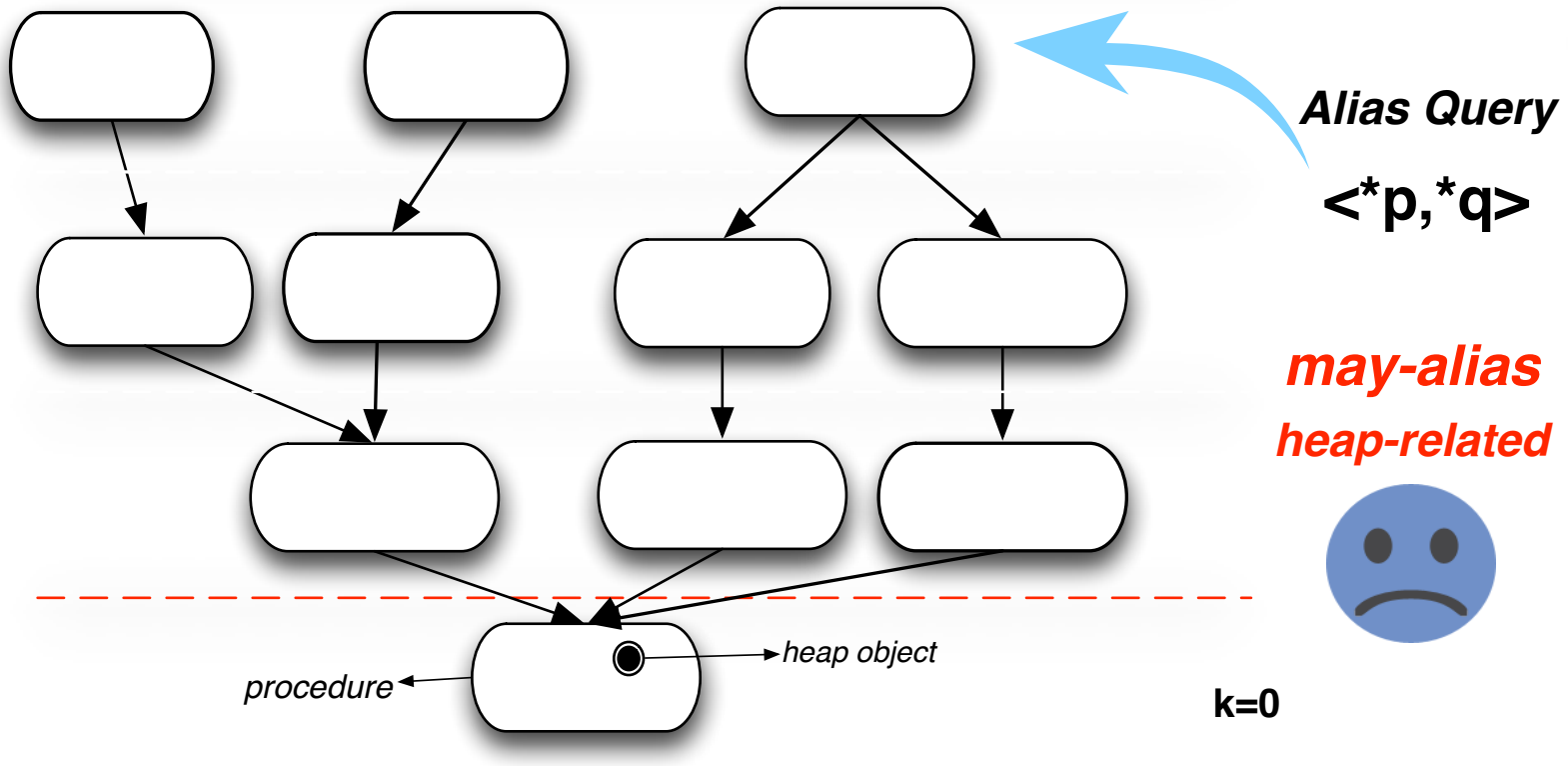
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



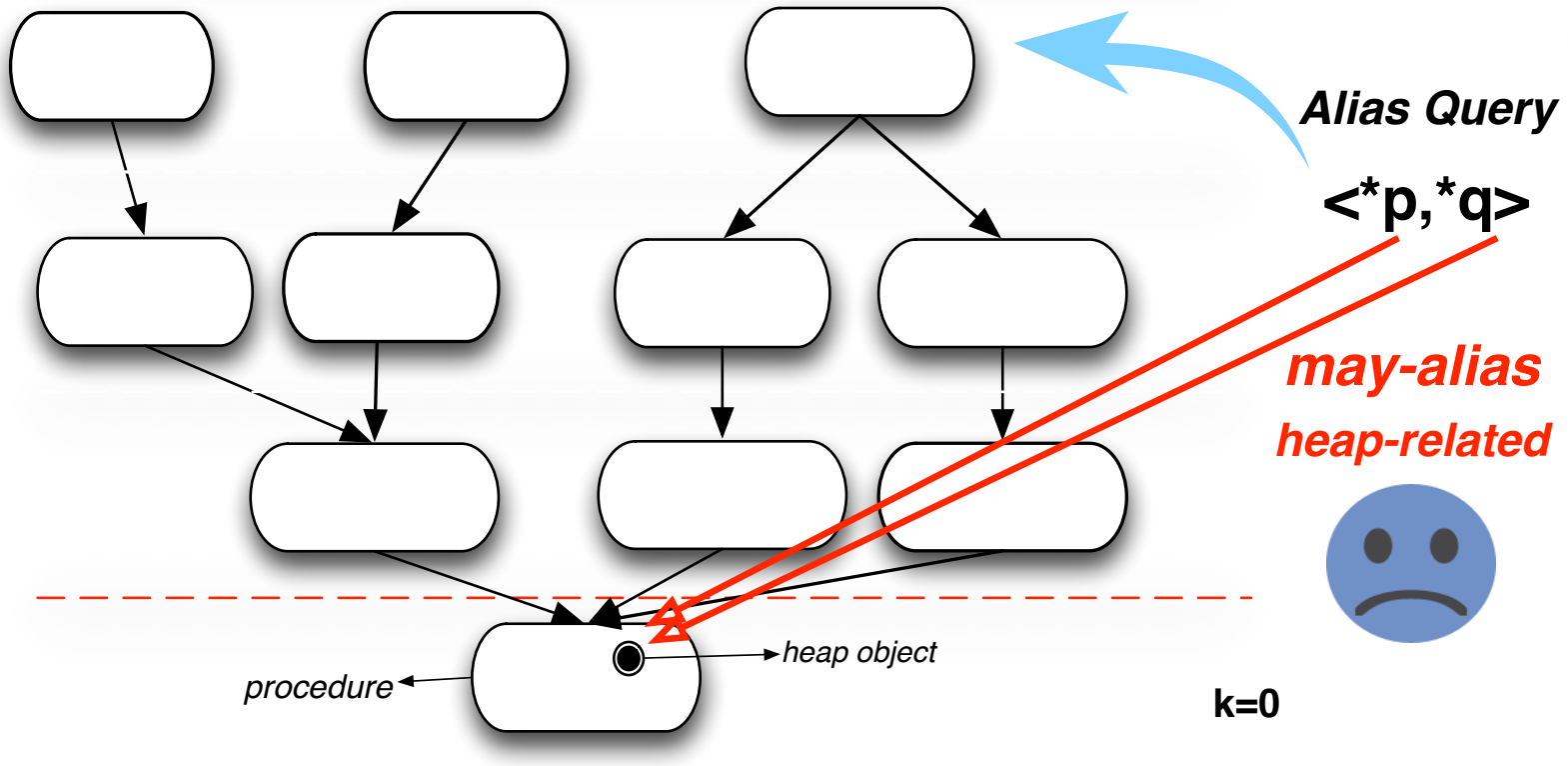
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



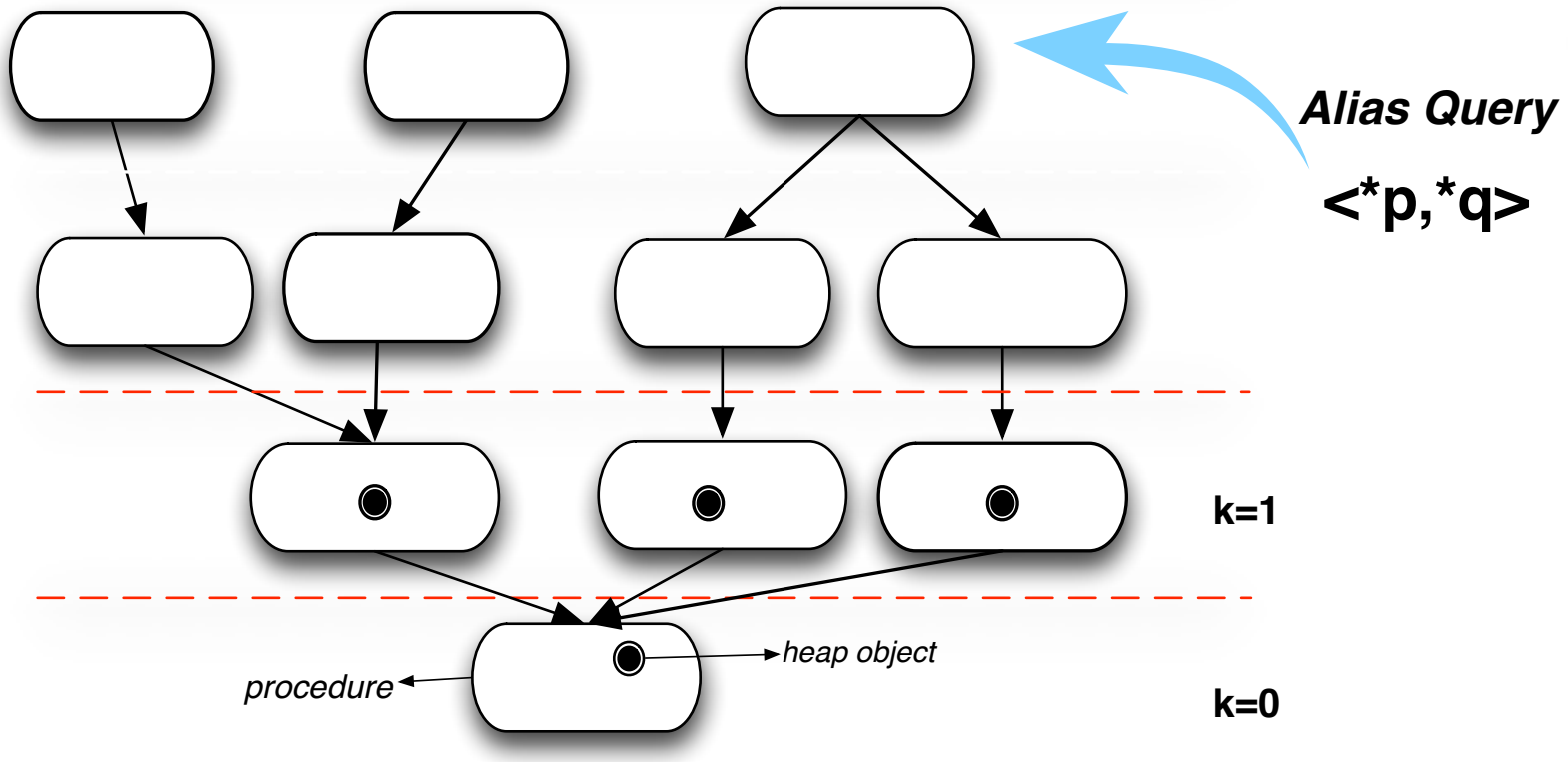
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



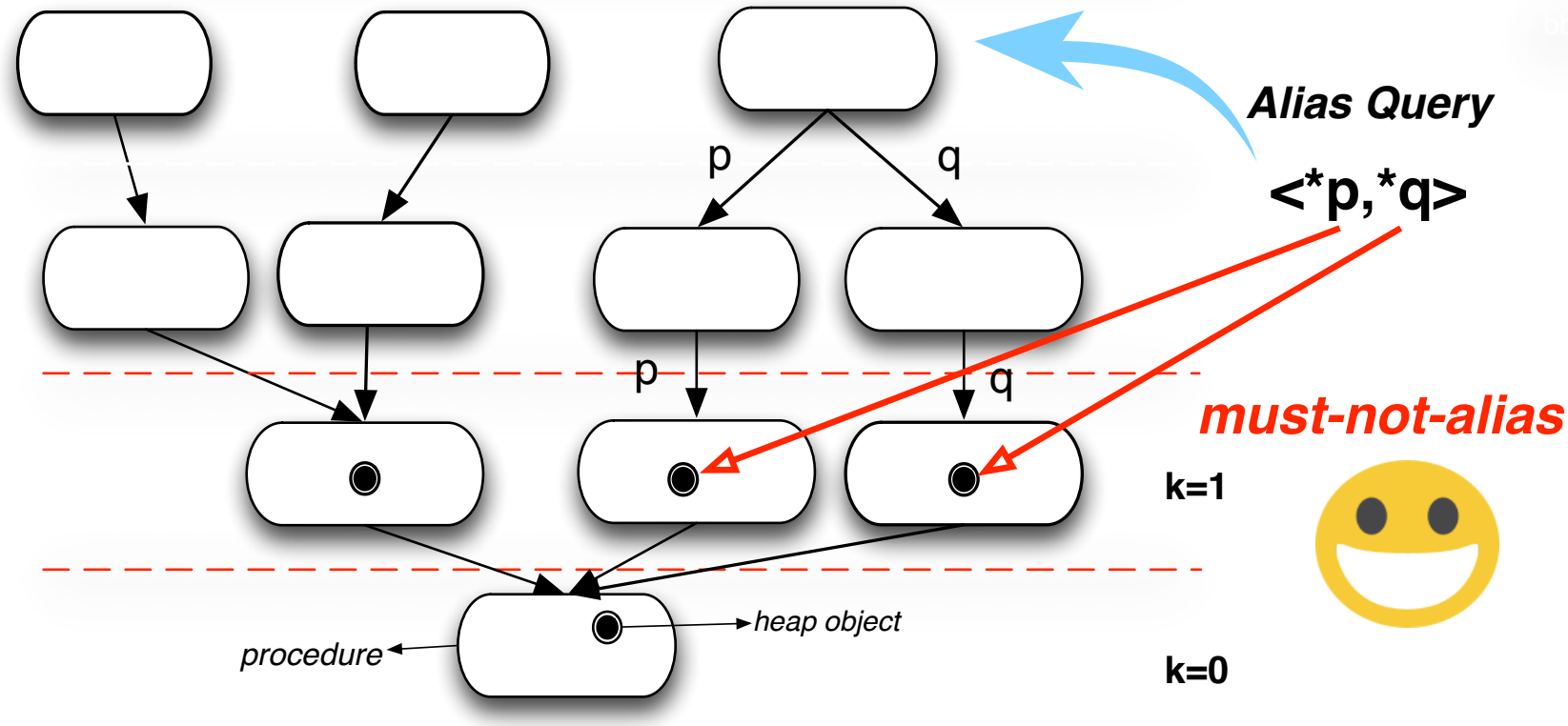
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



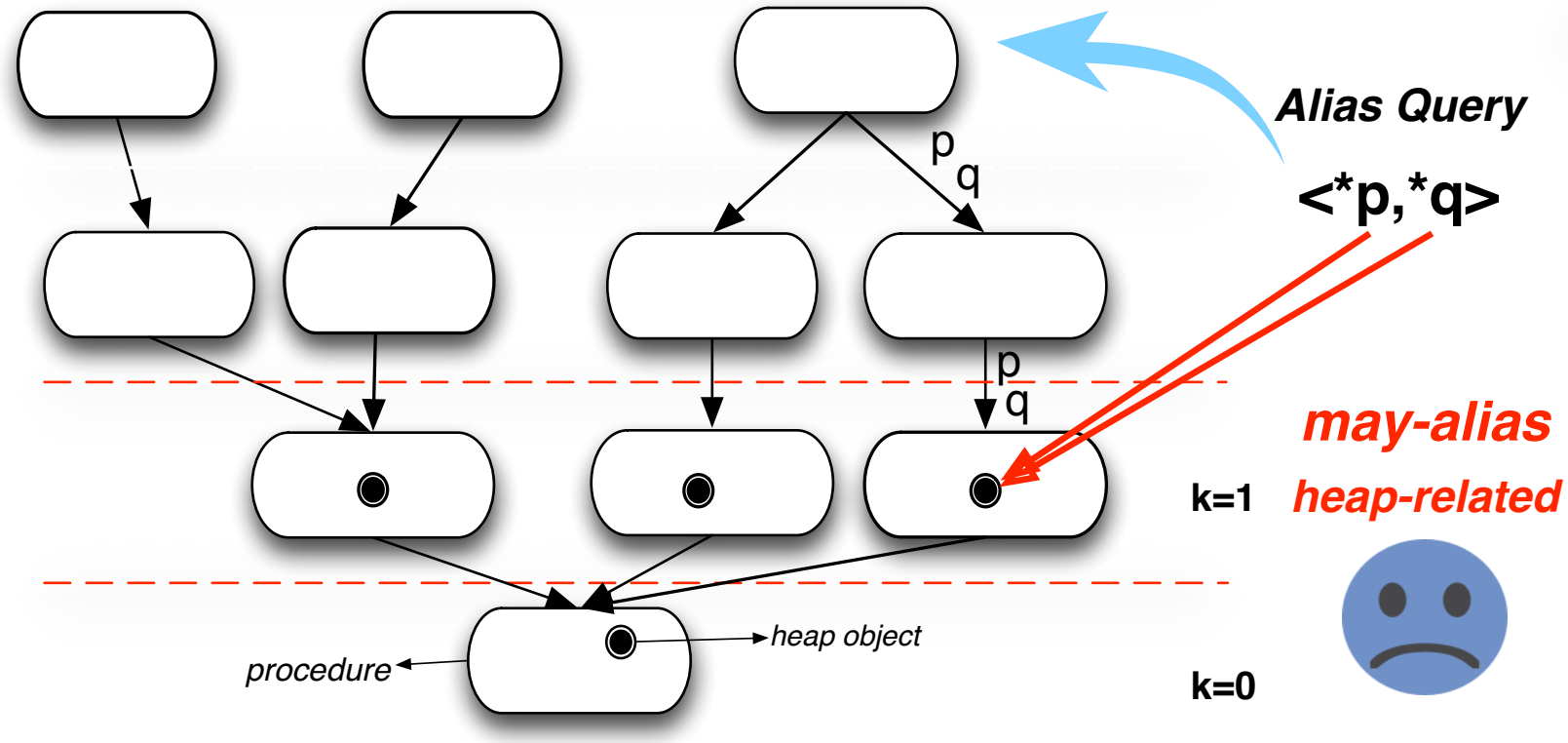
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



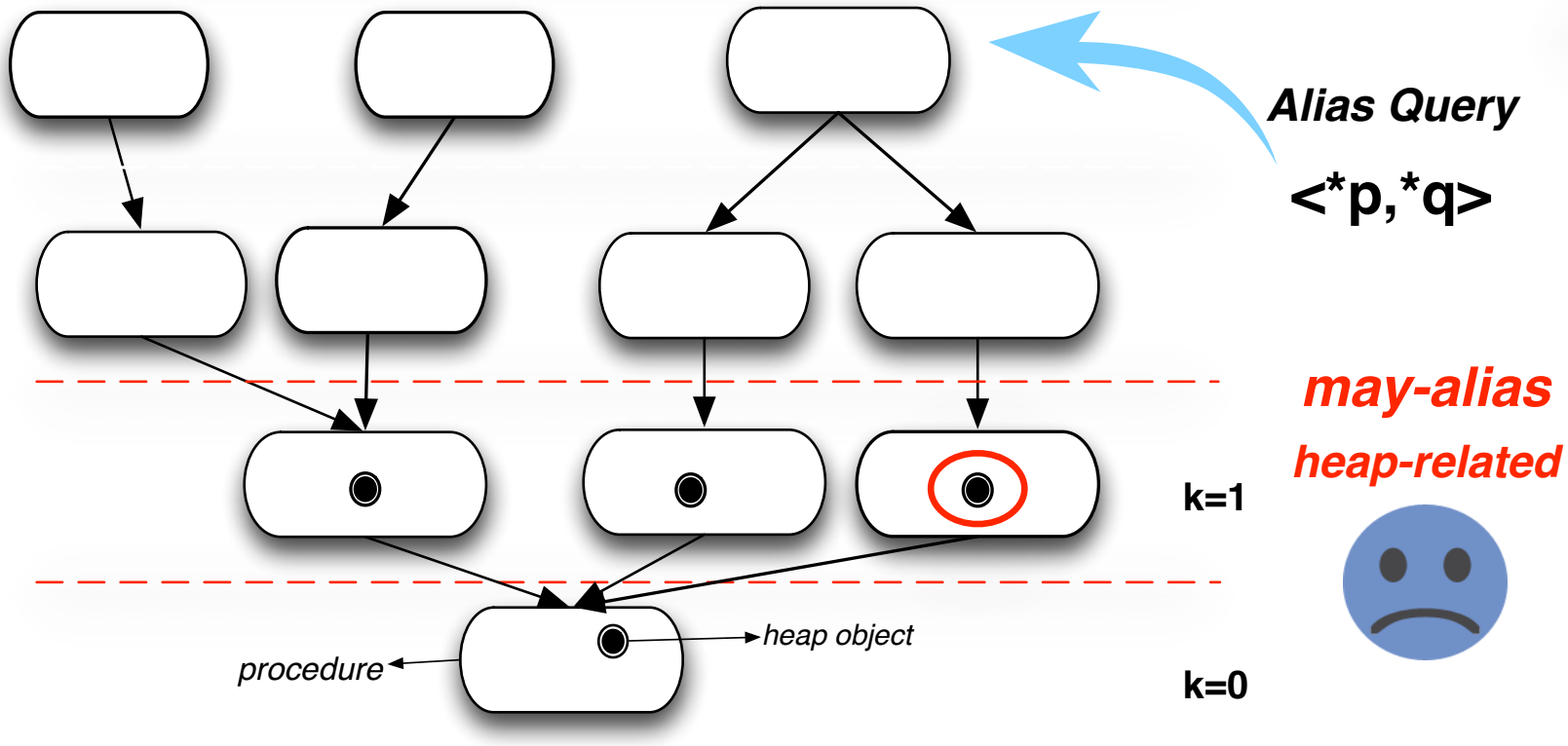
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



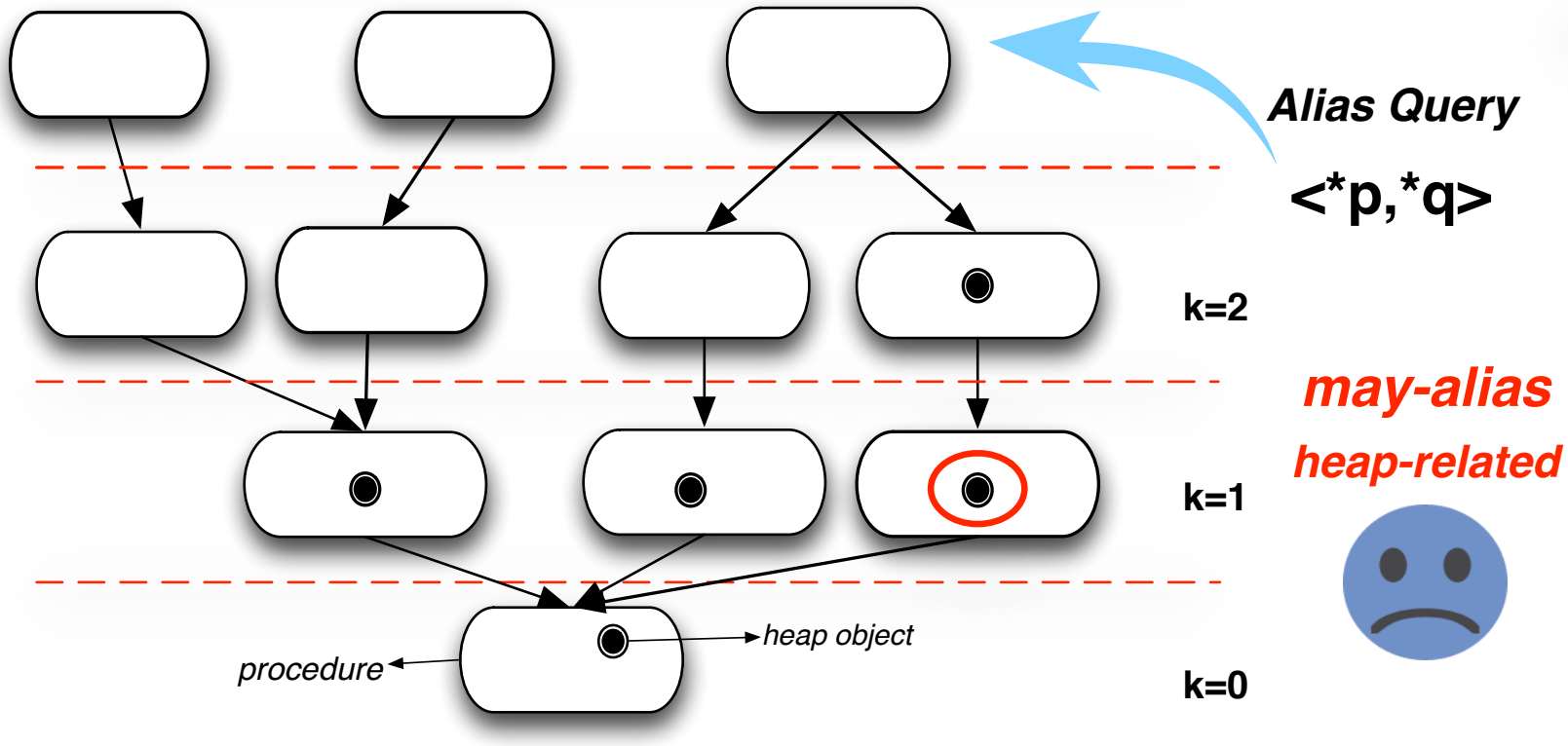
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



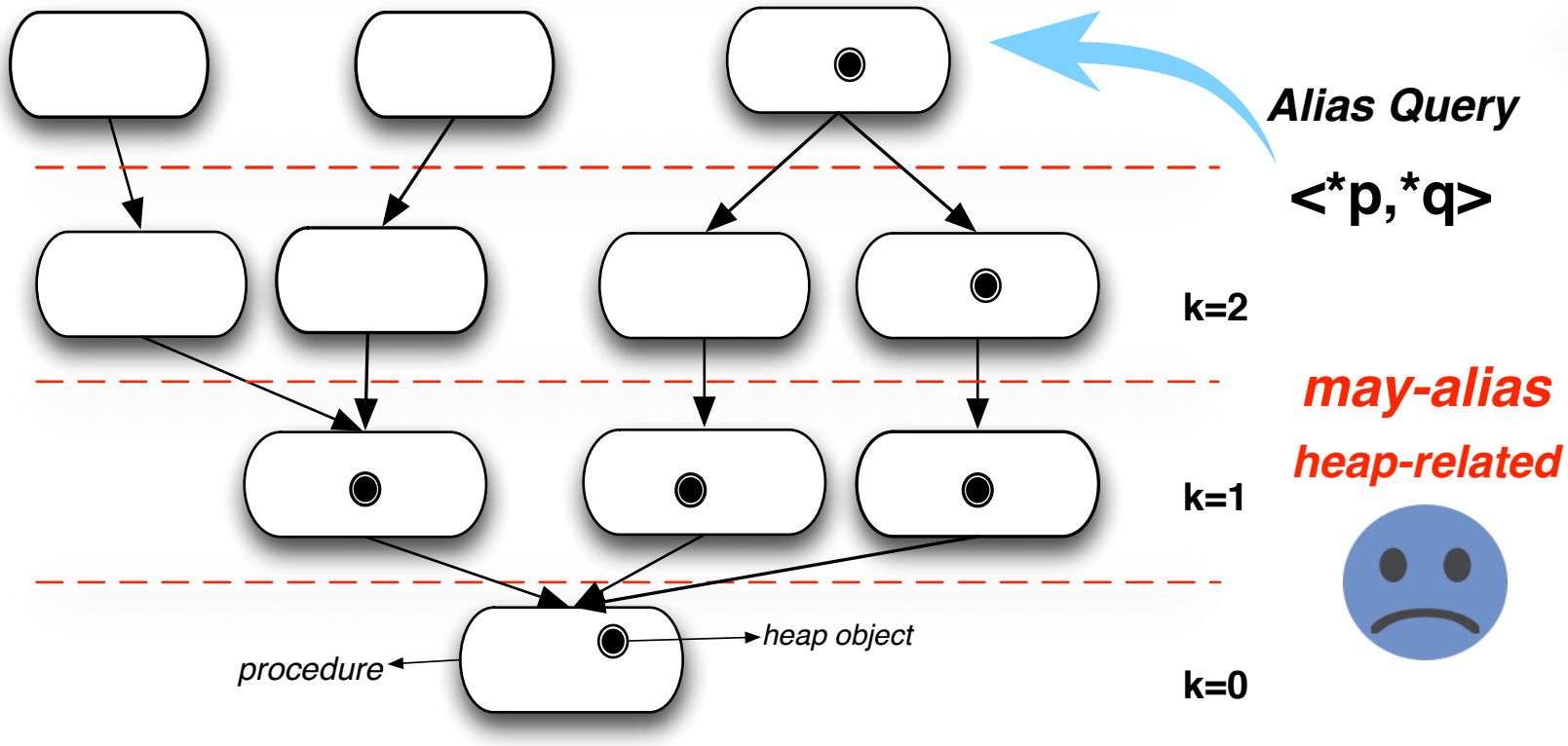
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



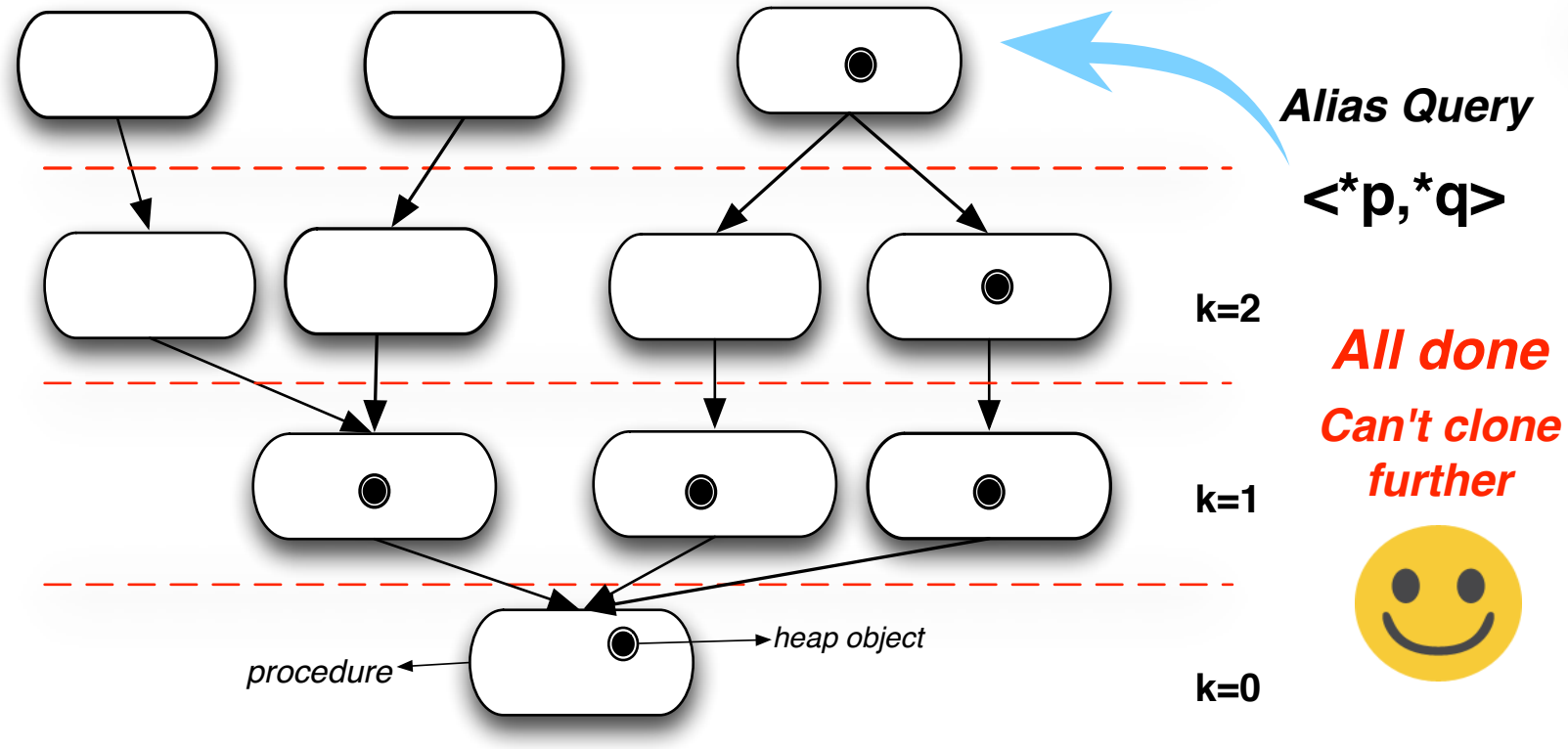
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



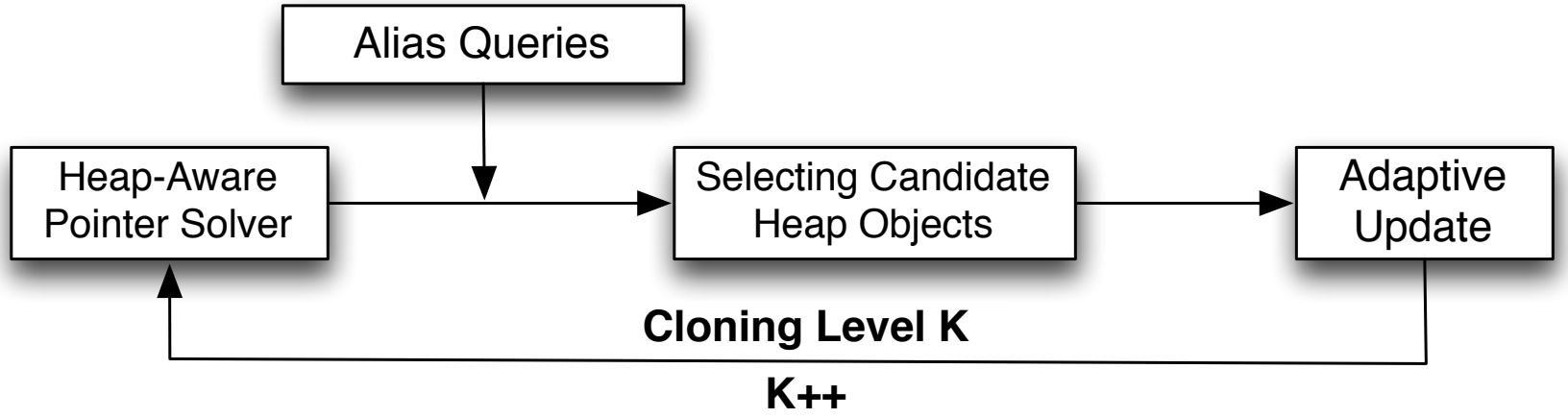
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



Query-Directed Adaptive Heap Cloning

QUDA:QUery-Directed Adaptive heap cloning



Heap-aware pointer analysis

$$x \rightarrow g$$
$$p \rightarrow o$$

Heap-aware pointer analysis

$$x \rightarrow (\text{true}, g)$$
$$p \rightarrow (h_o, o)$$

Heap-aware pointer analysis

$$
\begin{aligned}
\text{x} &\rightarrow (\text{true}, \ g) \\
\text{p} &\rightarrow (h_o, \quad o) \\
\text{q} &\rightarrow (h_o, \quad o)
\end{aligned}
$$

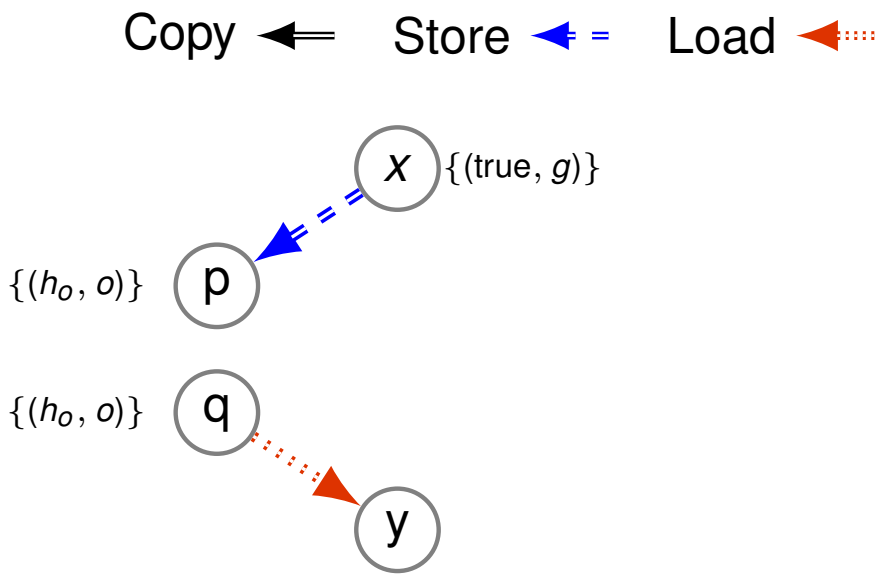
```
 *p  =  x;
y    =  *q;
```

Heap-aware pointer analysis

$$x \rightarrow (\text{true}, g)$$
$$p \rightarrow (h_o, o)$$
$$q \rightarrow (h_o, o)$$

```
*p = x;
y  = *q;
```

Constraint Graph:

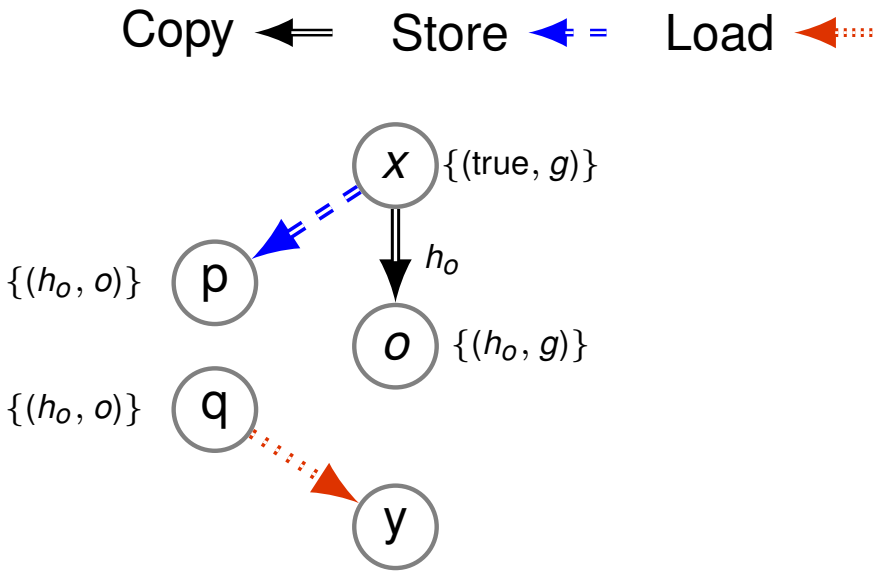


Heap-aware pointer analysis

$x \rightarrow (\text{true}, g)$
$p \rightarrow (h_o, o)$
$q \rightarrow (h_o, o)$

*p = x;
y = *q;

Constraint Graph:

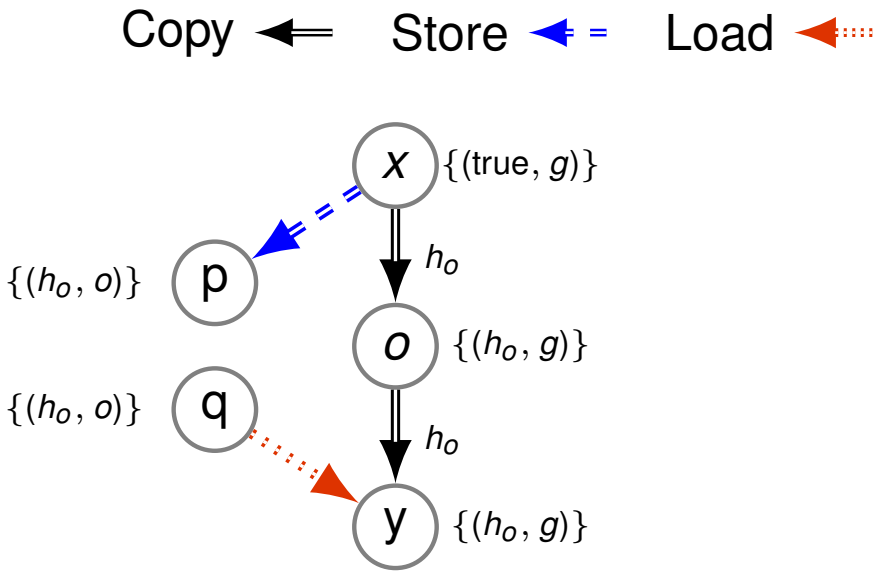


Heap-aware pointer analysis

$x \rightarrow (\text{true}, g)$
$p \rightarrow (h_o, o)$
$q \rightarrow (h_o, o)$

```
*p = x;
y  = *q;
```

Constraint Graph:



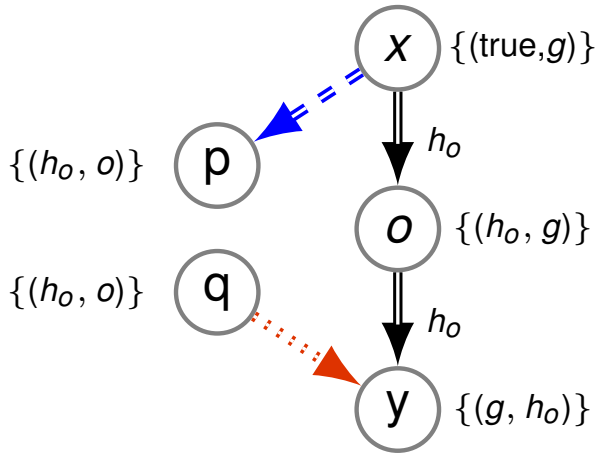
Candidate heap objects selection

Constraint Graph:

Copy ⟸ Store ⟸ Load

Alias Query

$\langle *x, *y \rangle$



Candidate heap objects selection

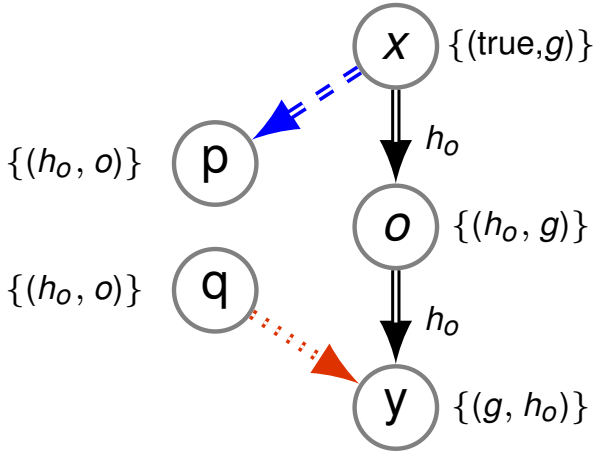
Constraint Graph:

Copy ⟸ Store ⟸ Load

Alias Query

$\langle *x, *y \rangle$

pts(x)={true,g}
pts(y)={h_o,g}



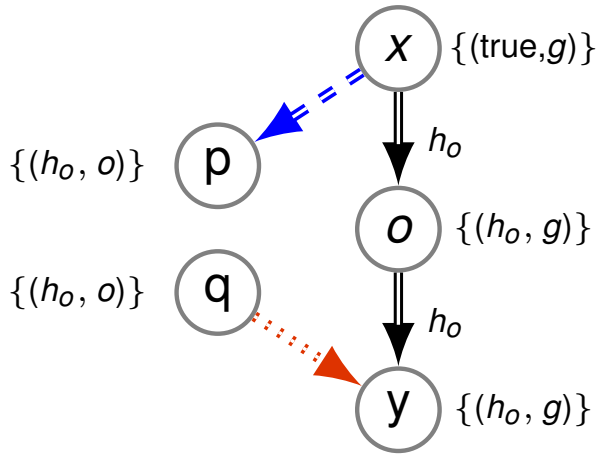
Candidate Heap Object $\{o\}$

Adaptive update

Constraint Graph:

Copy ⟸ Store ⟵= Load ⟵⋯

Candidate Heap Object $\{o\}$

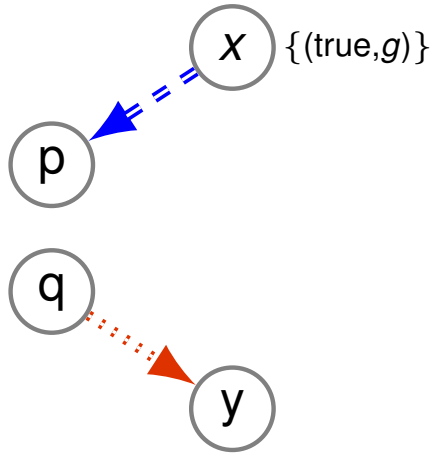


Adaptive update

Constraint Graph:

Copy ⟸ Store ⟵= Load ⟵⋯

Candidate Heap Object $\{o\}$



Next round resolution

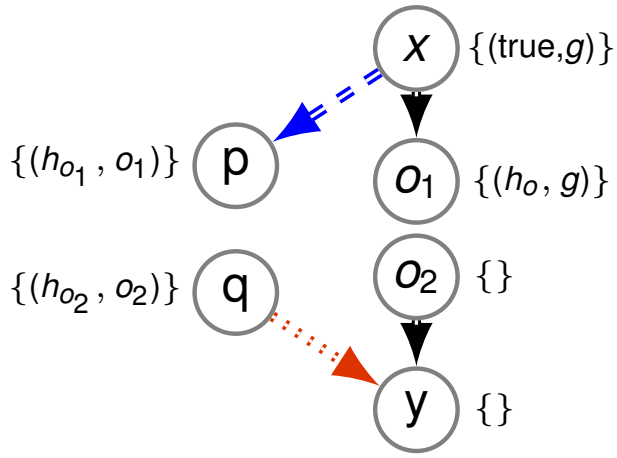
Constraint Graph:

Copy ⟸ Store ⟸ Load ⟸

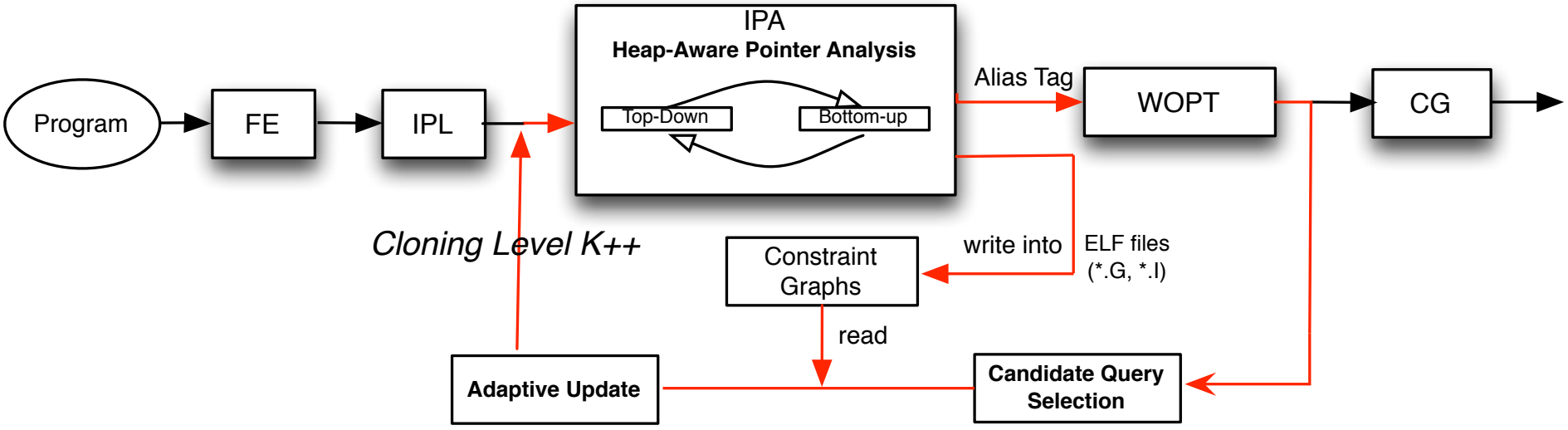
Alias Query

$\langle *x, *y \rangle$

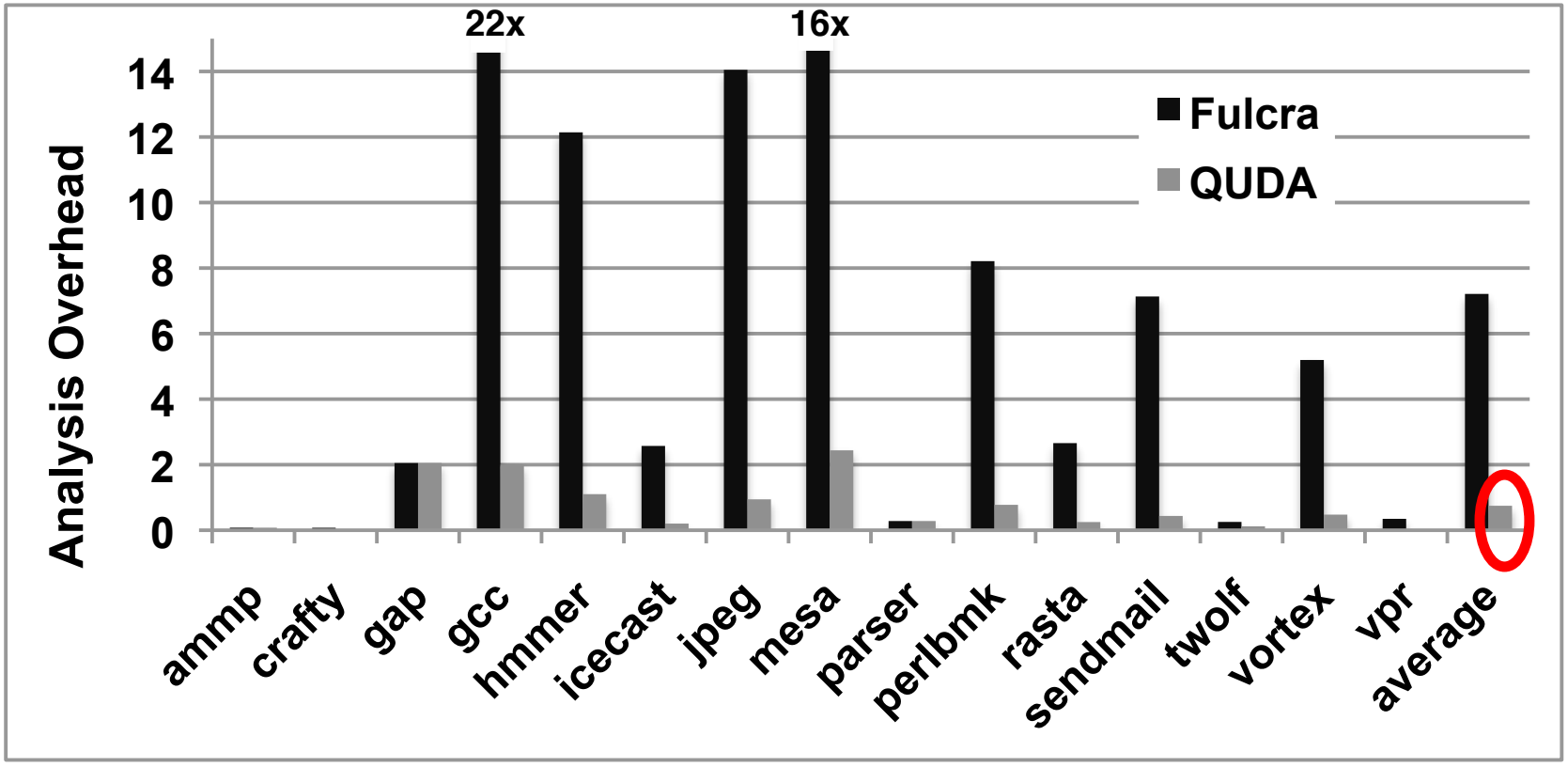
Not-alias!



QUDA framework in Open64

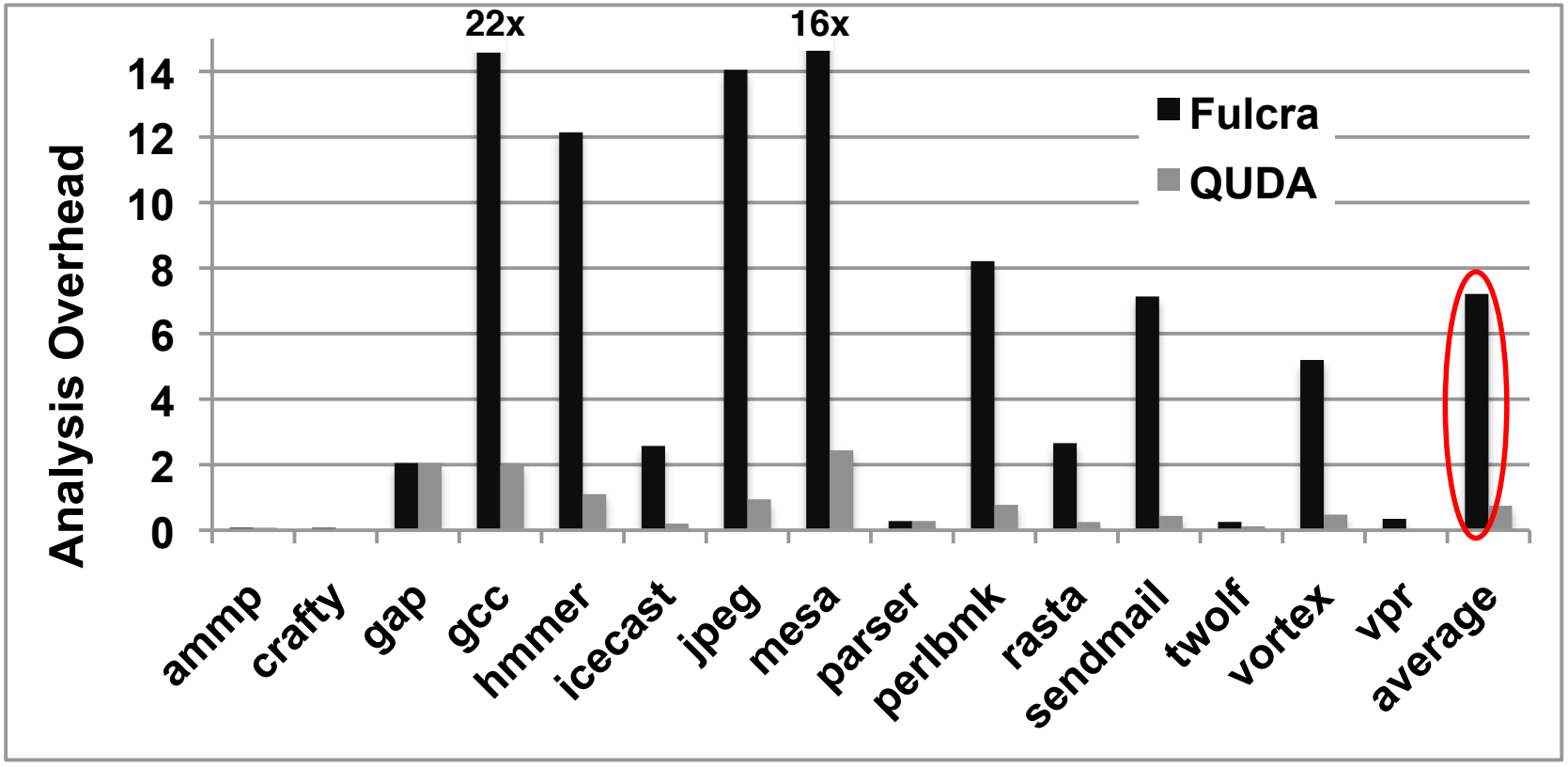


Analysis times of FULCRA and QUDA



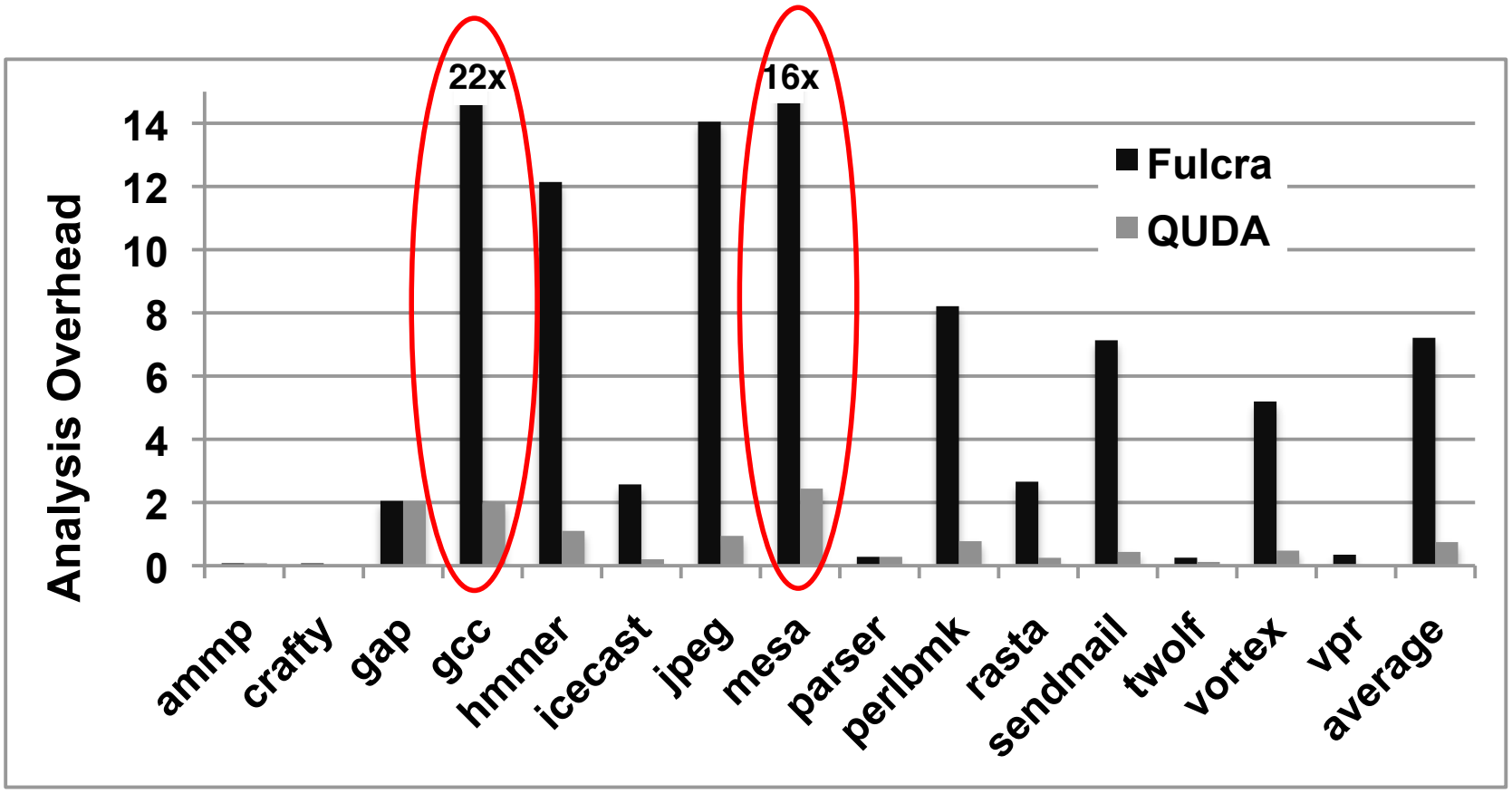
Analysis time normalized with respect to Open64's compile times (-O2)

Analysis times of FULCRA and QUDA



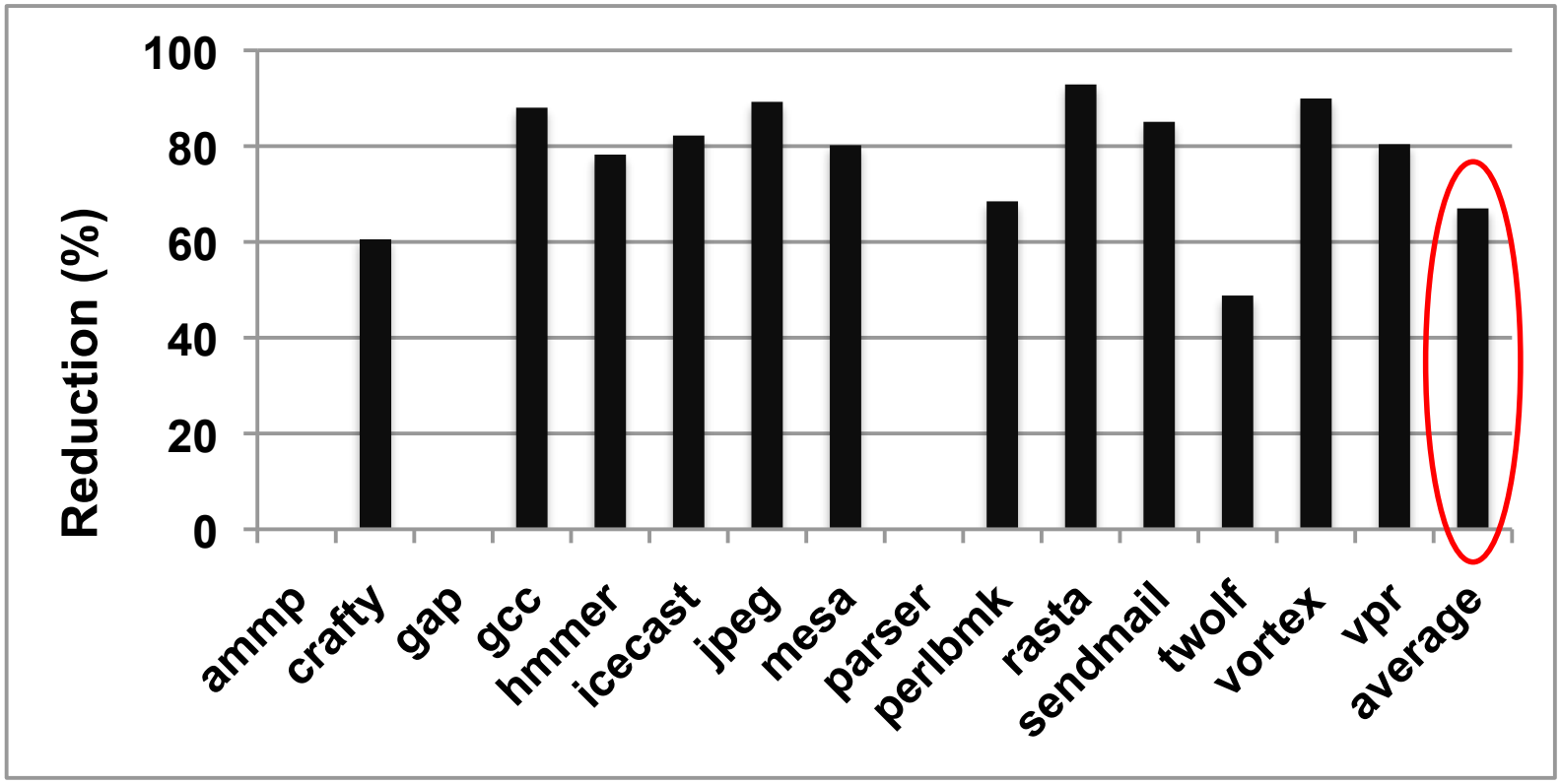
Analysis time normalized with respect to Open64's compile times (-O2)

Analysis times of FULCRA and QUDA



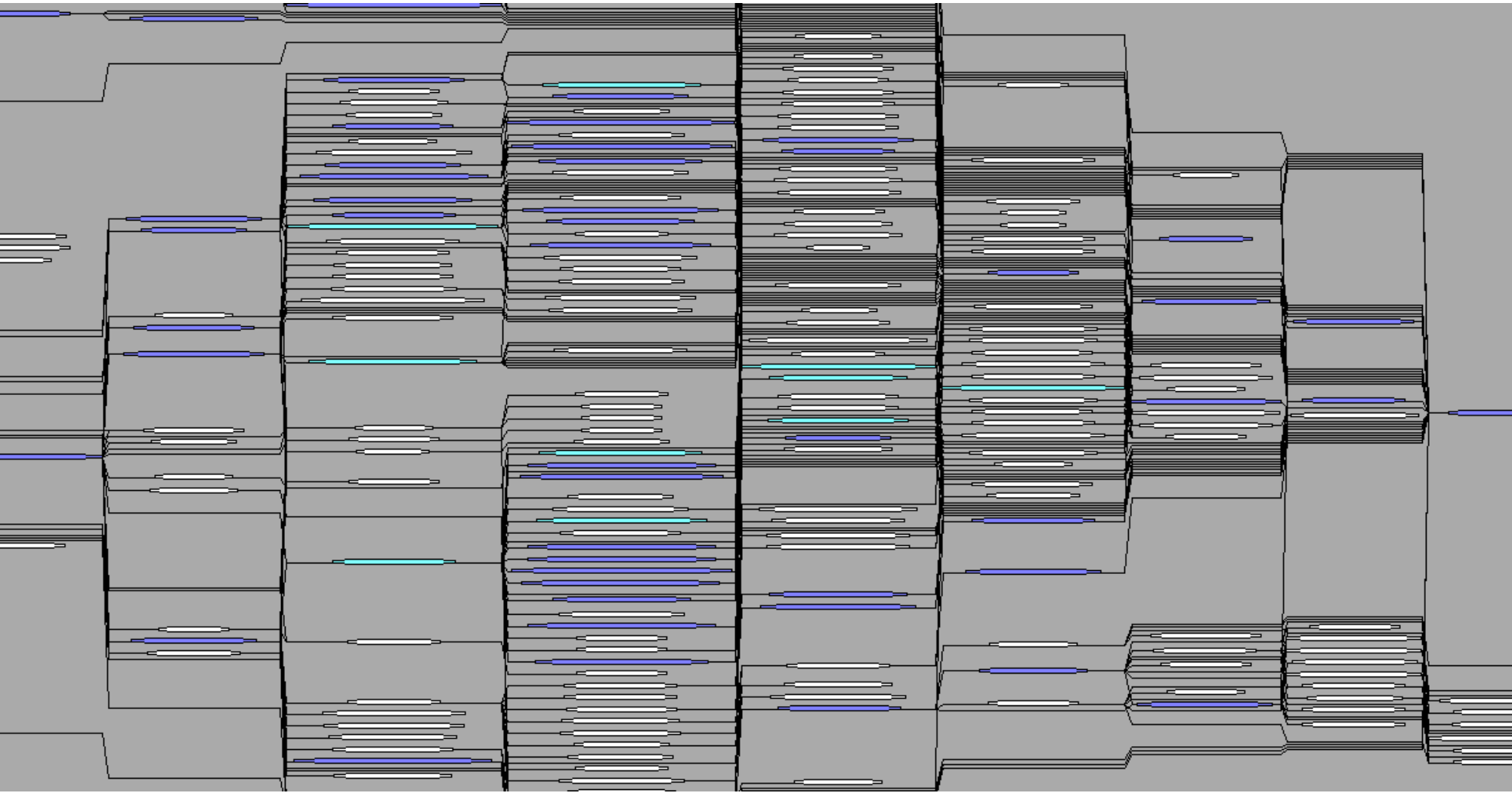
Analysis time normalized with respect to Open64's compile times (-O2)

Heap objects reduced by QUDA over FULCRA

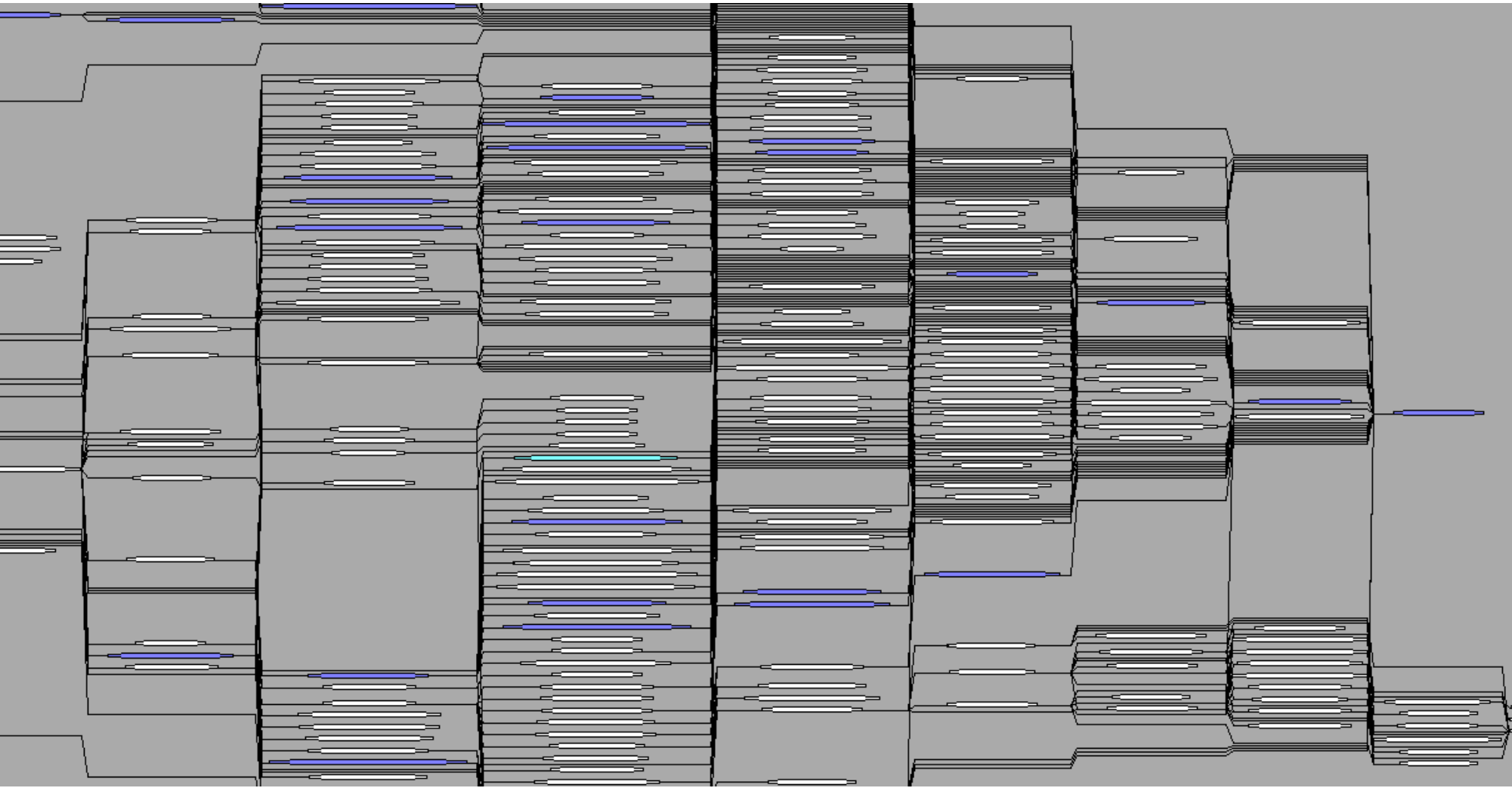


Number of heap objects reduced by QUDA over FULCRA in percentage terms

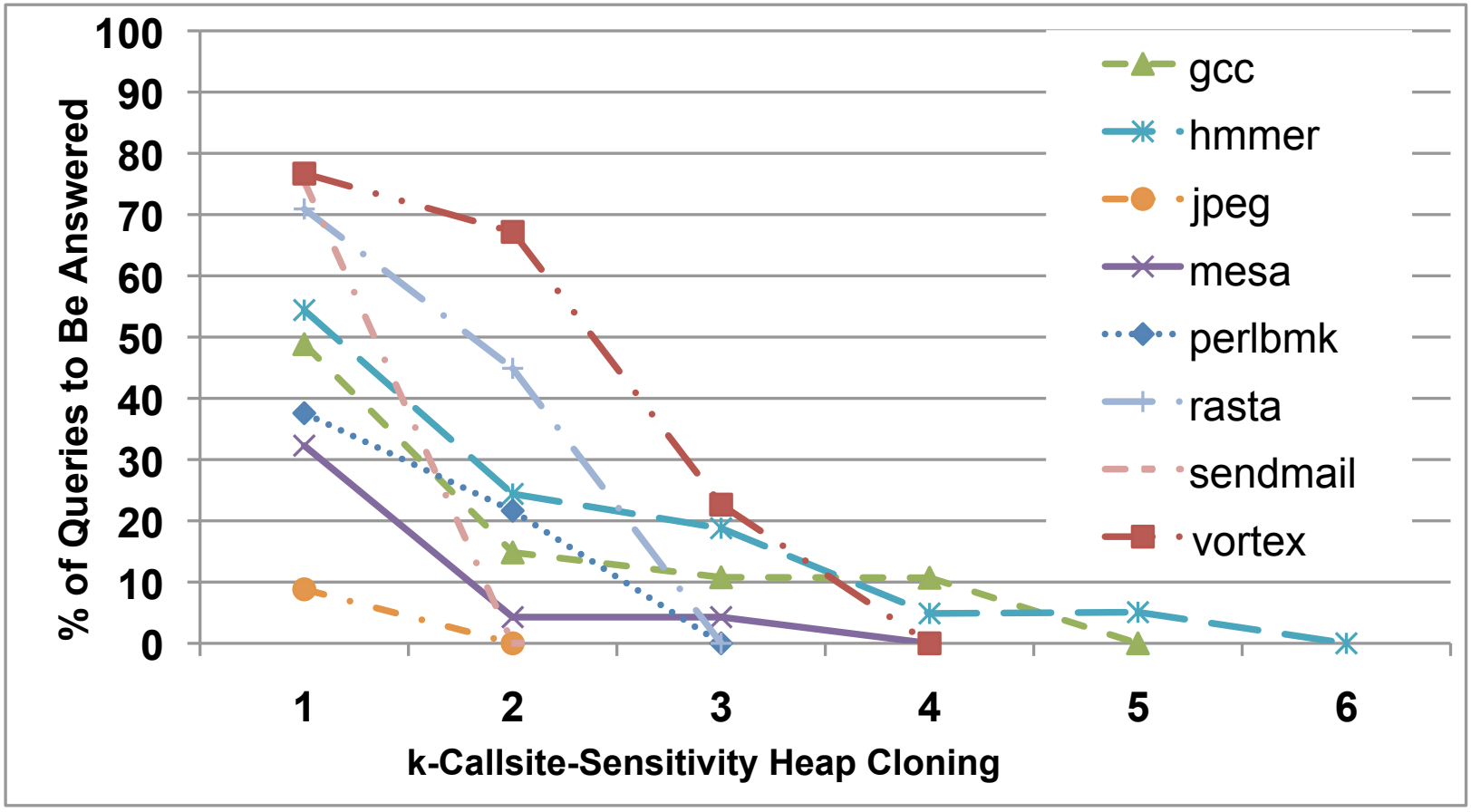
Heap distribution with full heap cloning (175.vpr)



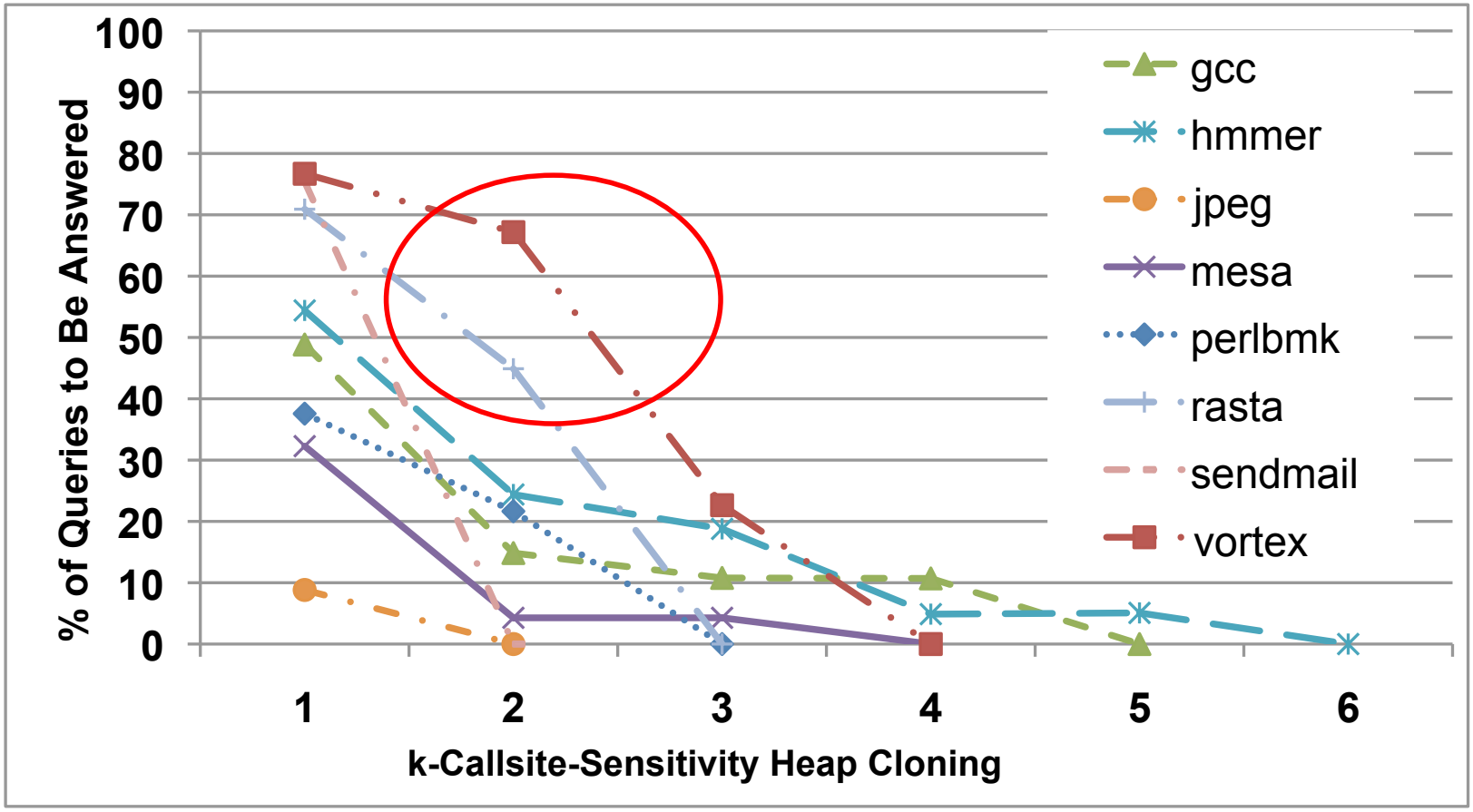
Heap distribution with QUDA (175.vpr)



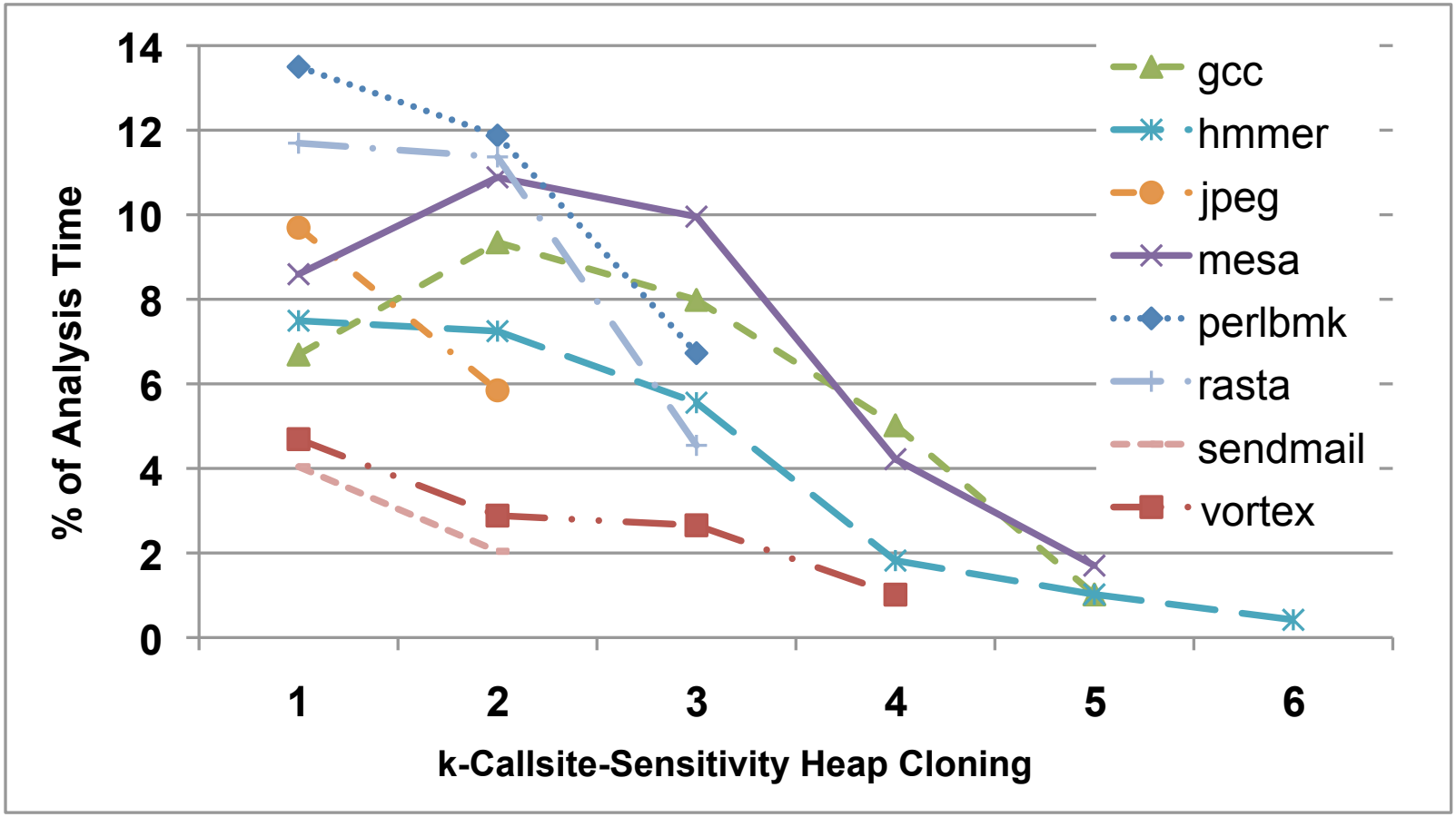
Alias queries to be answered at each iteration



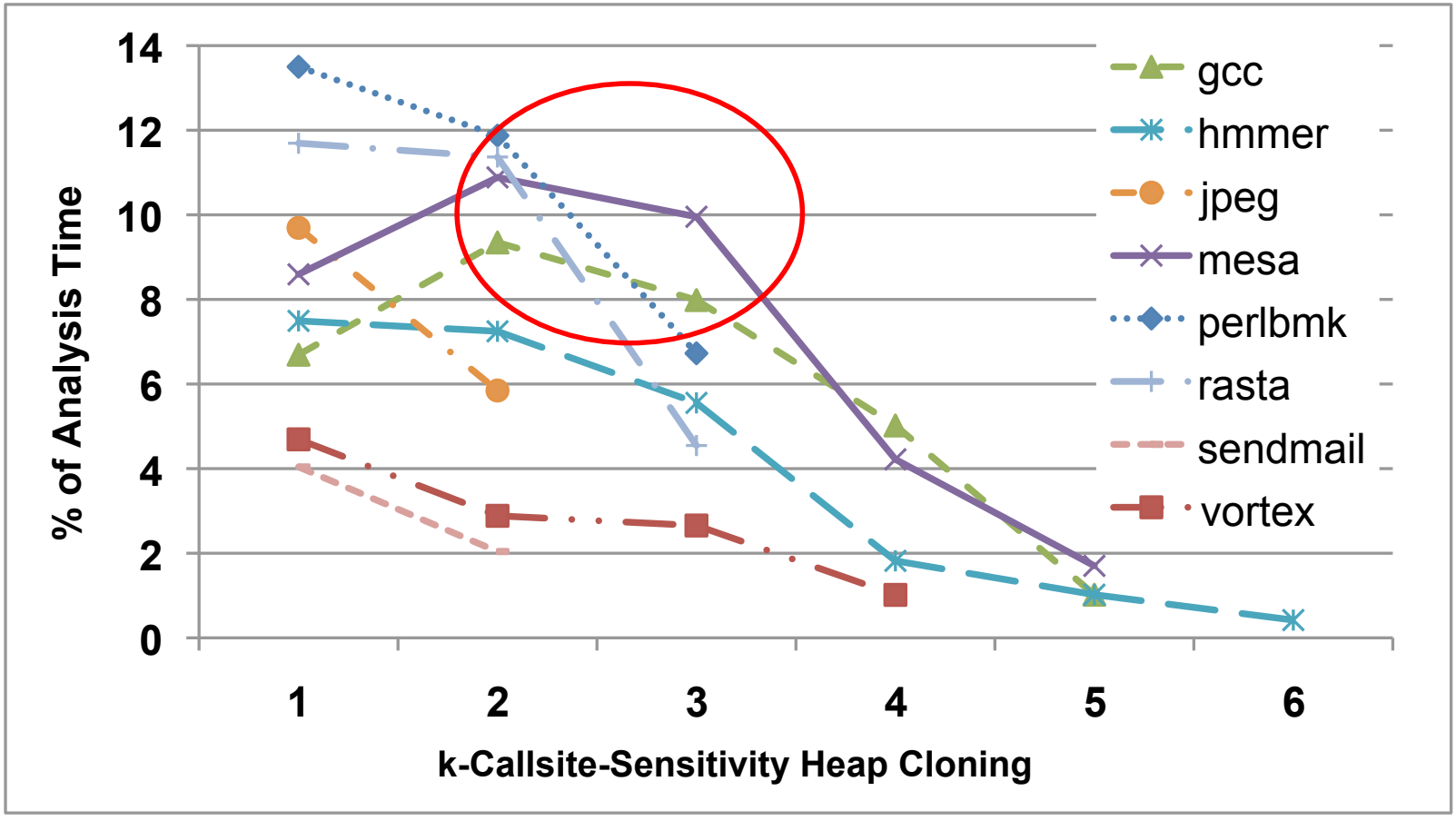
Alias queries to be answered at each iteration



Analysis time per iteration over the total



Analysis time per iteration over the total



Conclusion

Novel heap cloning approach: same precision as full heap cloning but significantly more scalable

- Heap-aware analysis
- Query-directed
- Adaptive

Challenges and opportunities:

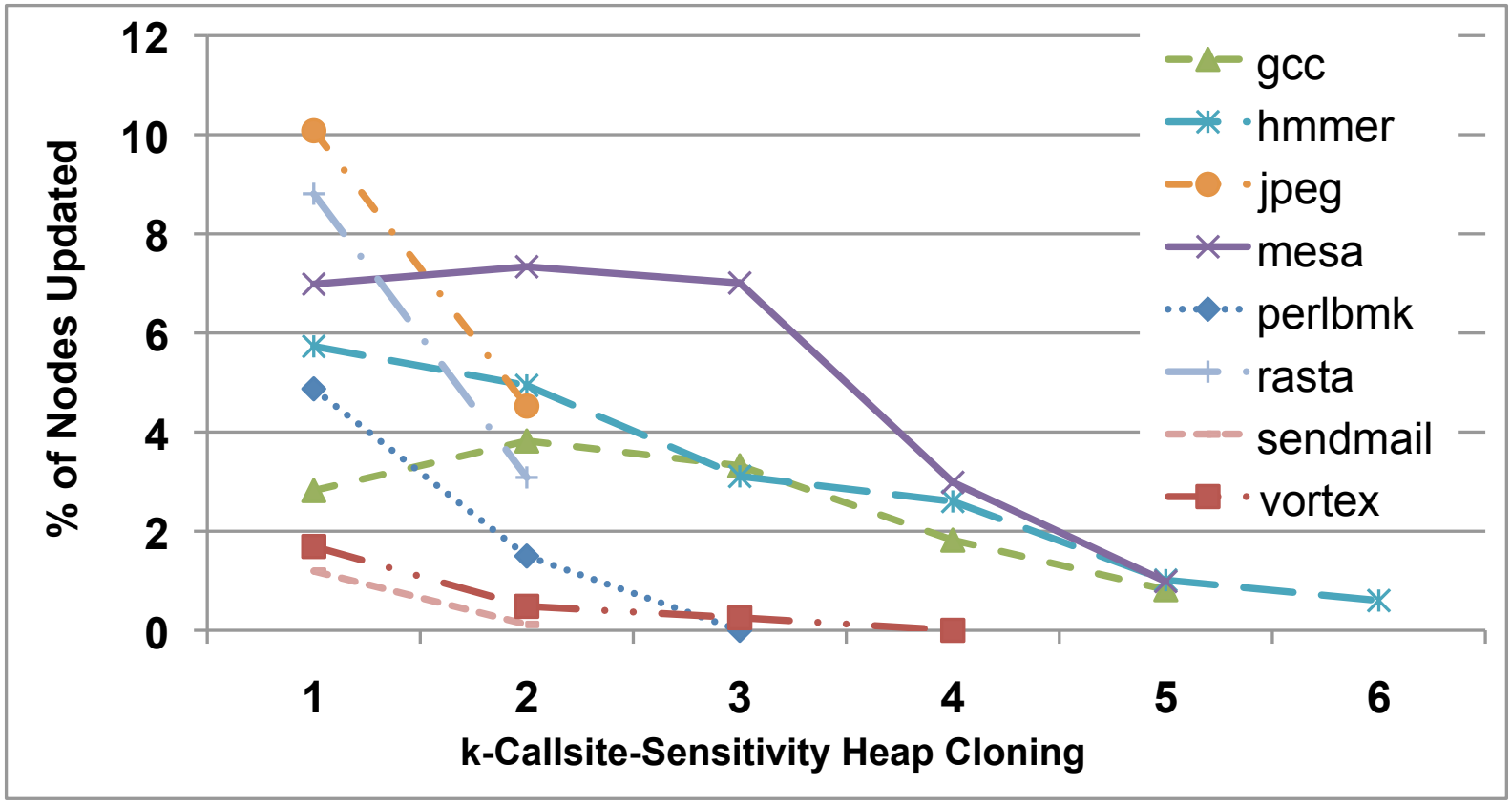
- Iterative compilation (prioritising queries in hot functions)
- Bug detection (scaling precise pointer analysis)

Thanks!

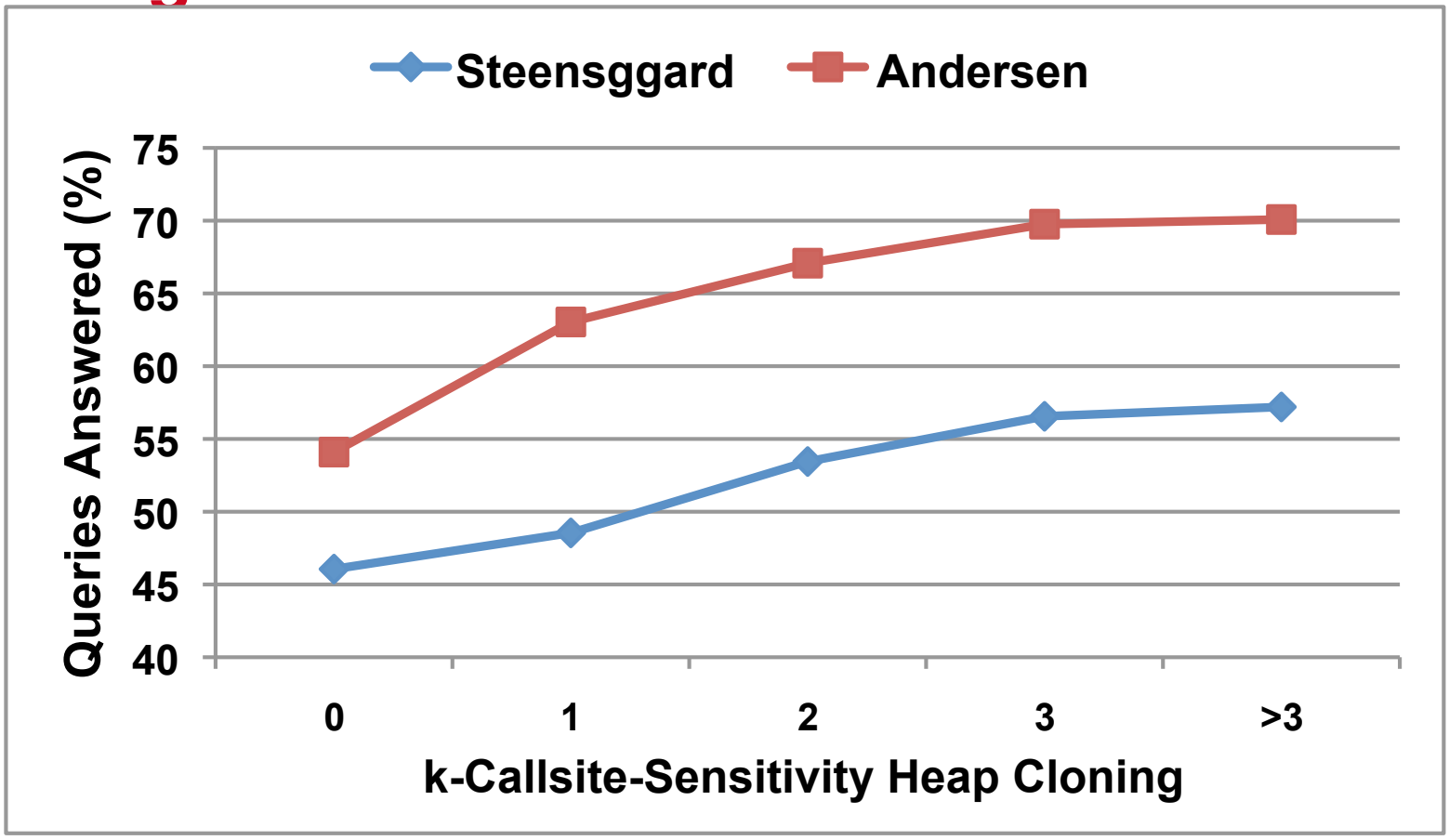
Backup Slides:Benchmarks

Program	Program Characteristics				QUDA				Analysis Times (secs)		
	KLOC	#Procs	#Pointers	#Callsites	#Queries	#Iters	#Nodes	#Edges	FULCRA	QUDA	Speedup
ammp	13.4	182	9829	1201	38893	2	11690	10267	0.38	0.38	1.00
crafty	21.2	112	11883	4046	15545	3	13725	10887	0.84	0.15	5.60
gap	71.5	857	61435	5980	101187	2	61856	70719	22.20	22.20	1.00
gcc	230.4	2256	134380	22353	128713	6	135785	153220	1321.33	92.78	14.24
hmmer	48.1	3868	645	2446	645	7	55285	63197	128.52	11.65	11.03
icecast	22.3	15098	603	468	7774	2	24085	24331	28.30	2.23	12.69
jpeg	26.4	377	19623	1177	8978	3	26884	43845	108.90	7.31	14.90
mesa	61.3	1109	44582	3611	144328	5	101228	141379	651.08	93.73	6.95
parser	11.4	327	8228	1782	9529	2	9155	8560	0.96	0.96	1.00
perlbmk	87.1	1079	54816	8470	98090	4	59341	64127	117.17	11.06	10.59
rasta	26.9	299	33387	2782	10892	4	33383	42963	30.10	2.83	10.64
sendmail	115.2	107242	2656	16973	144398	3	42887	46934	85.60	5.23	16.37
twolf	20.5	194	20773	2074	92917	2	24313	31672	2.56	1.20	2.13
vortex	67.3	926	40260	8522	81542	5	49678	50391	78.65	7.21	10.91
vpr	17.8	275	7930	1995	23557	4	9914	10902	1.32	0.10	13.20
total	840.8	134201	451030	83880	906988		659209	773394	2577.91	259.02	

Back Up Slides:Percentage of nodes updated

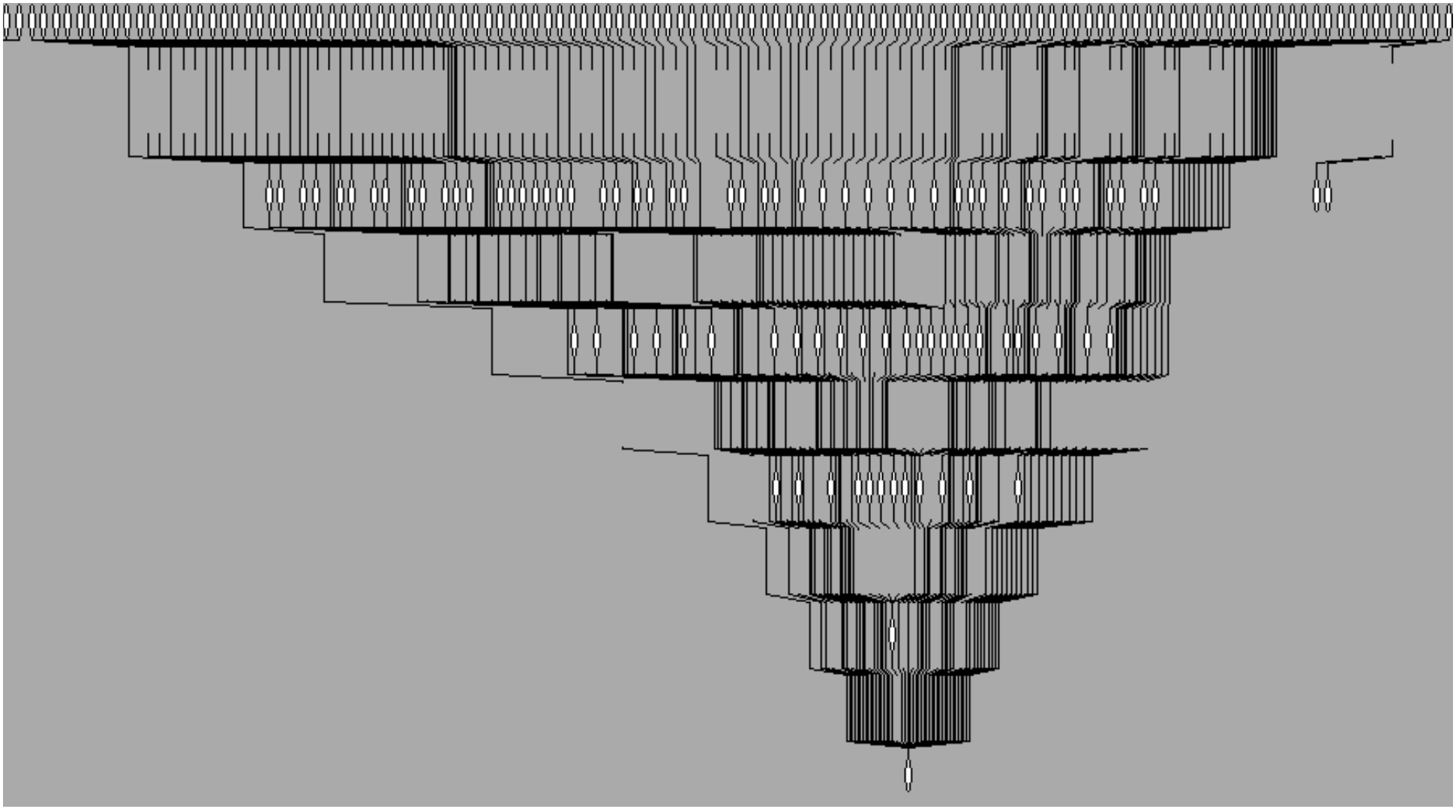


Back Up Slides: Percentage of must-not aliases disambiguated for hmmer

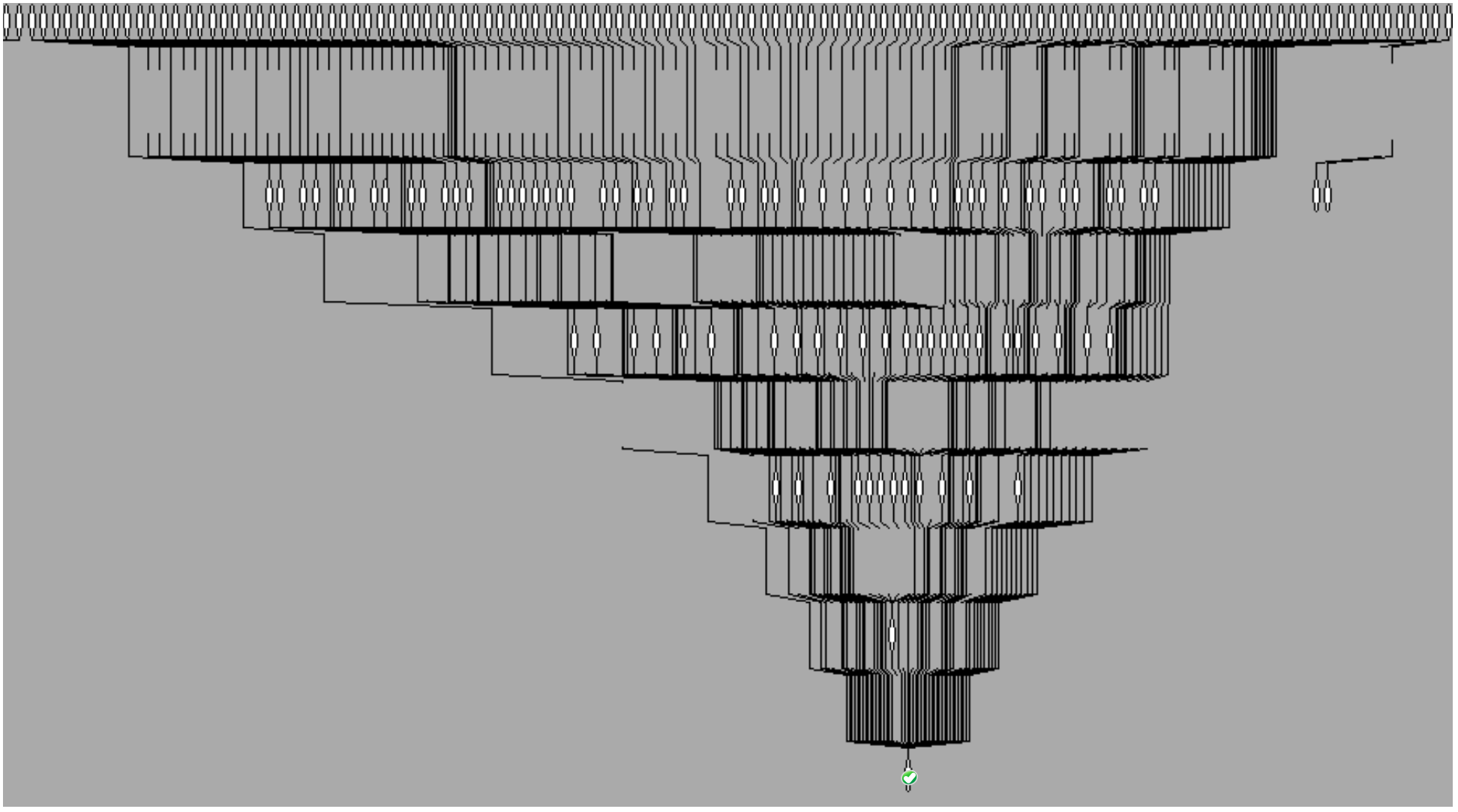


Queries issued by WOPT (in Open64s back- end) with k-callsite-sensitive heap cloning

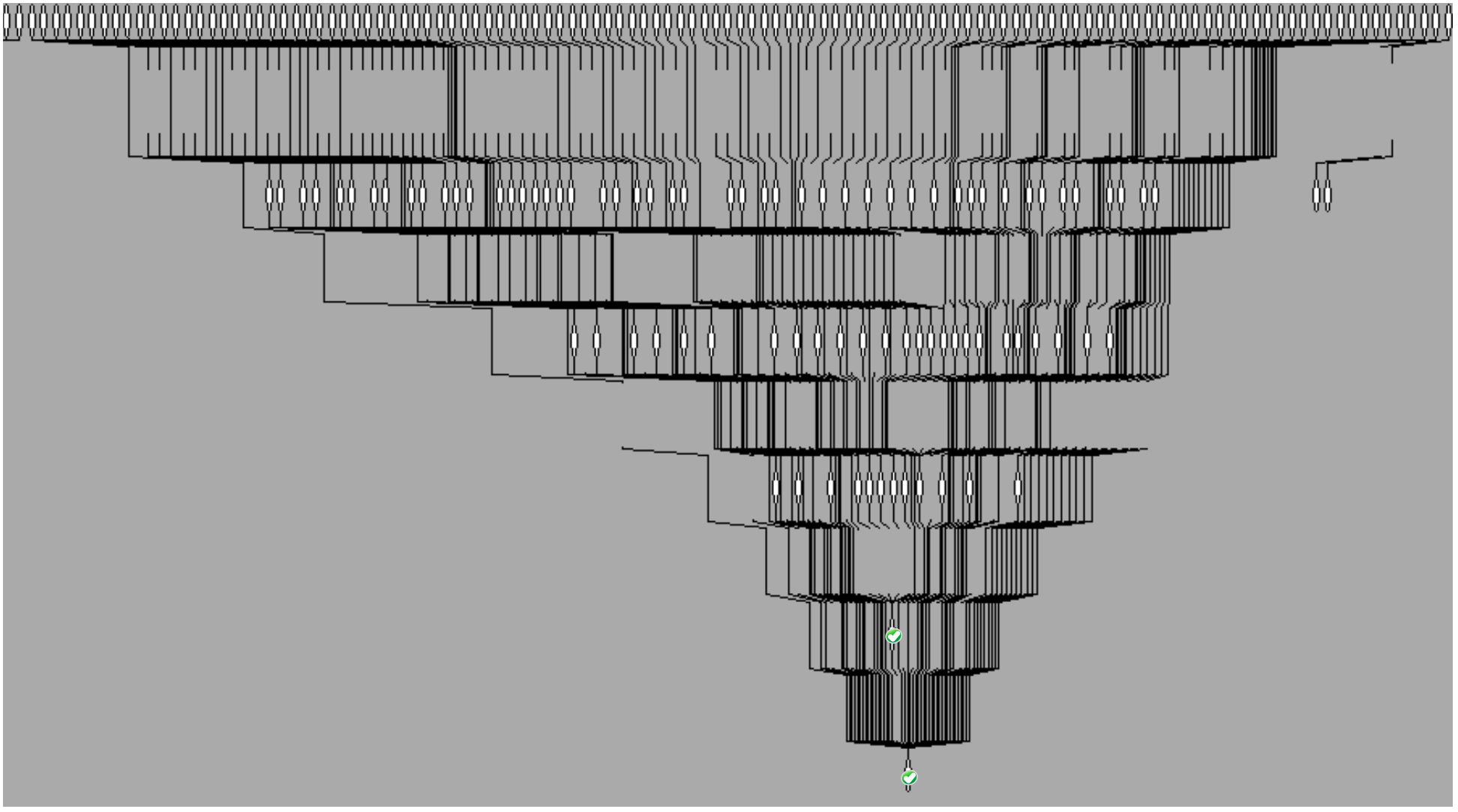
An exhaustive heap cloning tree (175.vpr)



Adaptive heap cloning tree construction (175.vpr)



Adaptive heap cloning tree construction (175.vpr)



Adaptive heap cloning tree construction (175.vpr)