Access-based Localization with Bypassing

Hakjoo Oh and Kwangkeun Yi

Seoul National University

APLAS 2011 @ Kenting, Taiwan

Challenge in Static Analysis

Precise, sound, scalable yet global static analyzers

Reality

Compromise either soundness or scalability

“bug-finders”	“verifiers”
scalable	sound
unsound	unscalable

Our Long-term Goal

Achieving scalable global static analyzers
without compromising precision and soundness

Overall Approach

- Design static analyzers by abstract interpretation
  - sound, precise, and global but unscalable
- Apply a set of cost-reduction techniques
  - scalable, preserving the precision and soundness

Localization

"local reasoning"
"framing" in separation logic

- Spatial localization [VMCAI'11]
- Temporal localization (submitted)
- Contextual localization [APLAS'09, SPE'10]

Localization

"local reasoning"
"framing" in separation logic

- Spatial localization [VMCAI'11, APLAS'11] — improved
- Temporal localization (submitted)
- Contextual localization [APLAS'09, SPE'10]

Performance of Sparrow

The Early Bird

Program	LOC	Baseline		Localize		Spd↑	Mem↓
		Time	Mem	Time	Mem		
gzip-1.2.4a	7 K	772	240	3	63	257 x	74 %
bc-1.06	13 K	1,270	276	7	75	181 x	73 %
less-382	23 K	9,561	1,113	33	127	289 x	86 %
make-3.76.1	27 K	24,240	1,391	21	114	1,154 x	92 %
wget-1.9	35 K	44,092	2,546	11	85	4,008 x	97 %
a2ps-4.14	64 K	∞	N/A	40	353	N/A	N/A
sendmail-8.13.6	130 K	∞	N/A	744	678	N/A	N/A
nethack-3.3.0	211 K	∞	N/A	16,373	5,298	N/A	N/A
emacs-22.1	399 K	∞	N/A	37,830	7,795	N/A	N/A
python-2.5.1	435 K	∞	N/A	11,039	5,535	N/A	N/A
linux-3.0	710 K	∞	N/A	33,618	20,529	N/A	N/A
gimp-2.6	959 K	∞	N/A	3,874	3,602	N/A	N/A
ghostscript-9.00	1,363 K	∞	N/A	14,814	6,384	N/A	N/A

Performance of Sparrow

The Early Bird

Program	LOC	Baseline		Localize		Spd↑	Mem↓
		Time	Mem	Time	Mem		
gzip-1.2.4a	7 K	772	240	3	63	257 x	74 %
bc-1.06	13 K	1,270	276	7	75	181 x	73 %
less-382	23 K	9,561	1,113	33	127	289 x	86 %
make-3.76.1	27 K	24,240	1,391	21	114	1,154 x	92 %
wget-1.9	35 K	44,092	2,546	11	85	4,008 x	97 %
a2ps-4.14	64 K	∞	N/A	40	353	N/A	N/A
sendmail-8.13.6	130 K	∞	N/A	744	678	N/A	N/A
nethack-3.3.0	211 K	∞	N/A	16,373	5,298	N/A	N/A
emacs-22.1	399 K	∞	N/A	37,830	7,795	N/A	N/A
python-2.5.1	435 K	∞	N/A	11,039	5,535	N/A	N/A
linux-3.0	710 K	∞	N/A	33,618	20,529	N/A	N/A
gimp-2.6	959 K	∞	N/A	3,874	3,602	N/A	N/A
ghostscript-9.00	1,363 K	∞	N/A	14,814	6,384	N/A	N/A

Performance of Sparrow

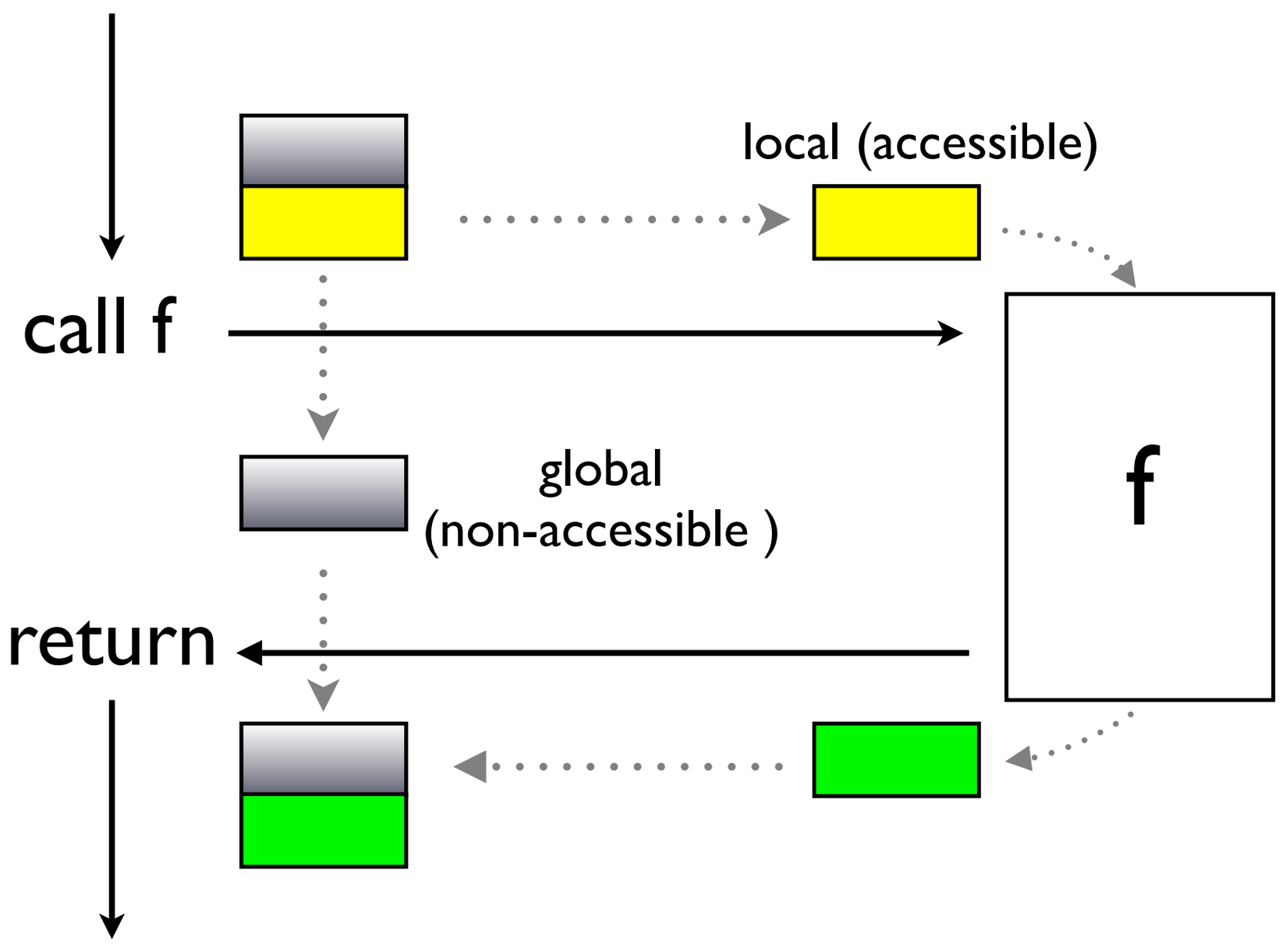
The Early Bird

Program	LOC	Baseline		Localize		Spd↑	Mem↓
		Time	Mem	Time	Mem		
gzip-1.2.4a	7 K	772	240	3	63	257 x	74 %
bc-1.06	13 K	1,270	276	7	75	181 x	73 %
less-382	23 K	9,561	1,113	33	127	289 x	86 %
make-3.76.1	27 K	24,240	1,391	21	114	1,154 x	92 %
wget-1.9	35 K	44,092	2,546	11	85	4,008 x	97 %
a2ps-4.14	64 K	∞	N/A	40	353	N/A	N/A
sendmail-8.13.6	130 K	∞	N/A	744	678	N/A	N/A
nethack-3.3.0	211 K	∞	N/A	16,373	5,298	N/A	N/A
emacs-22.1	399 K	∞	N/A	37,830	7,795	N/A	N/A
python-2.5.1	435 K	∞	N/A	11,039	5,535	N/A	N/A
linux-3.0	710 K	∞	N/A	33,618	20,529	N/A	N/A
gimp-2.6	959 K	∞	N/A	3,874	3,602	N/A	N/A
ghostscript-9.00	1,363 K	∞	N/A	14,814	6,384	N/A	N/A

VMCAI'11

Memory Localization

(spatial localization)



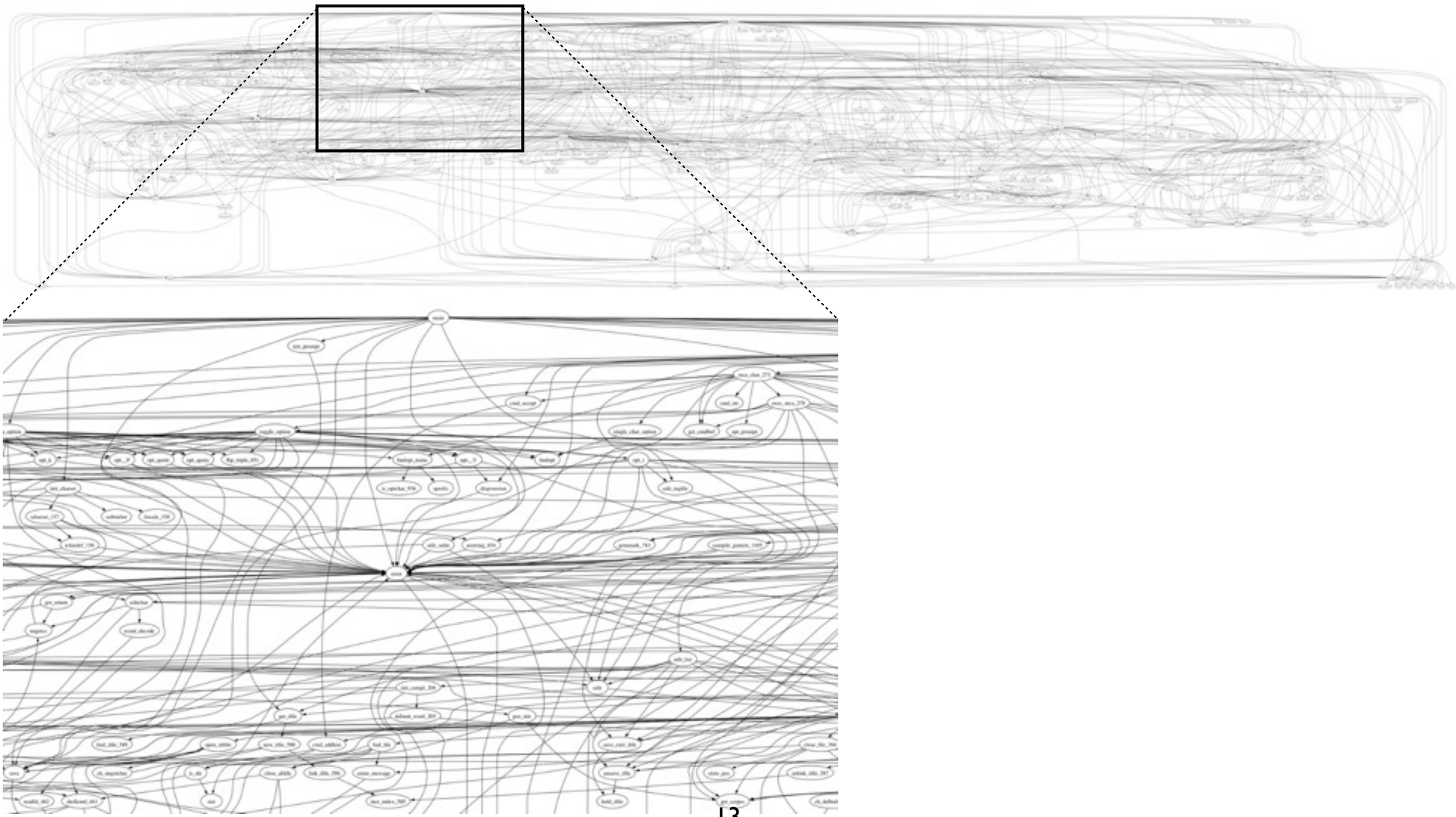
Benefits of Localization

```
int g;

int f() {...}

int main() {
  g = 0;
  f();

  g = 1;
  f();
}
```

f does not access g

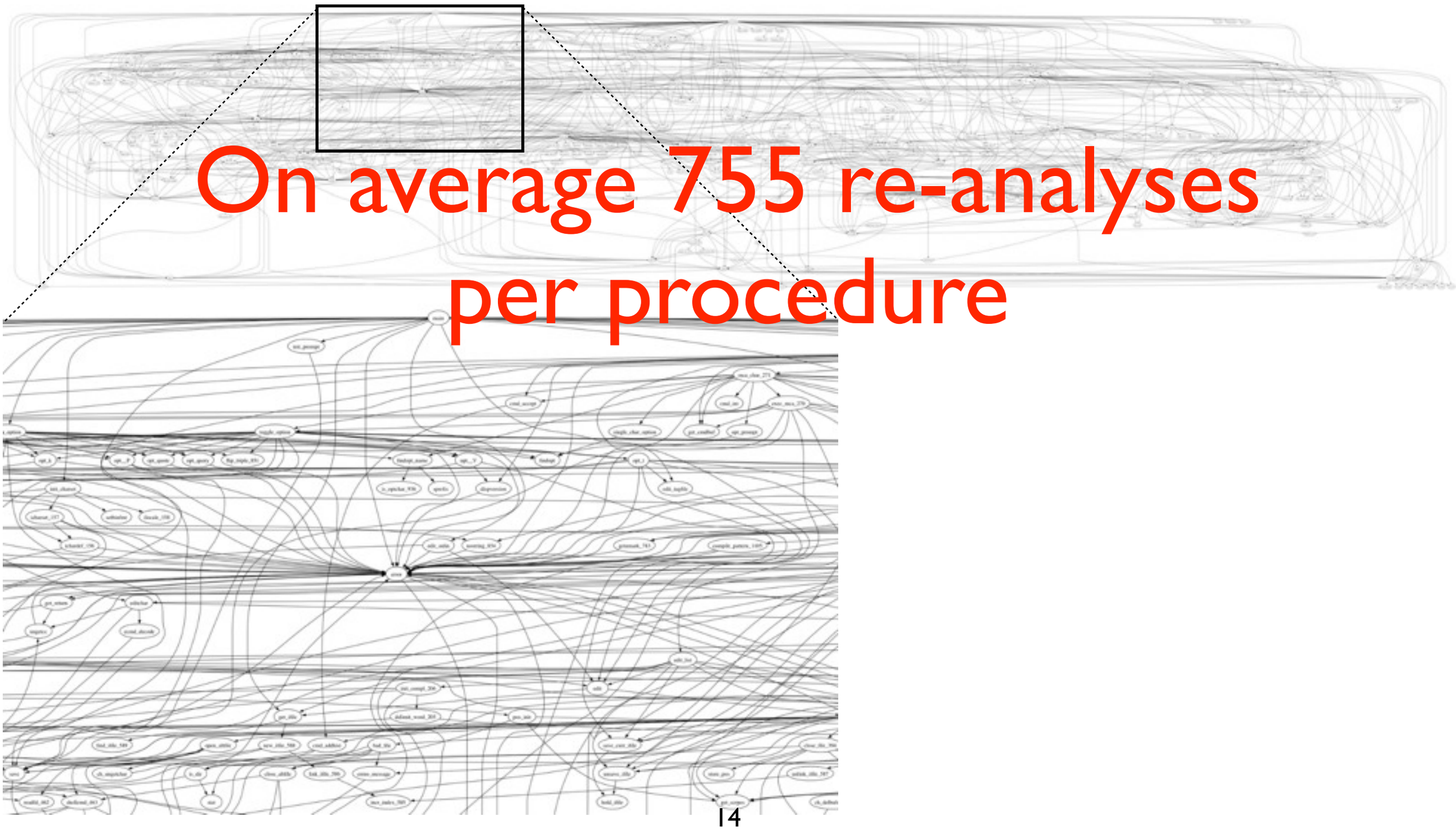
Localization Is Vital

less-382 (23,822 LOC)



Localization Is Vital

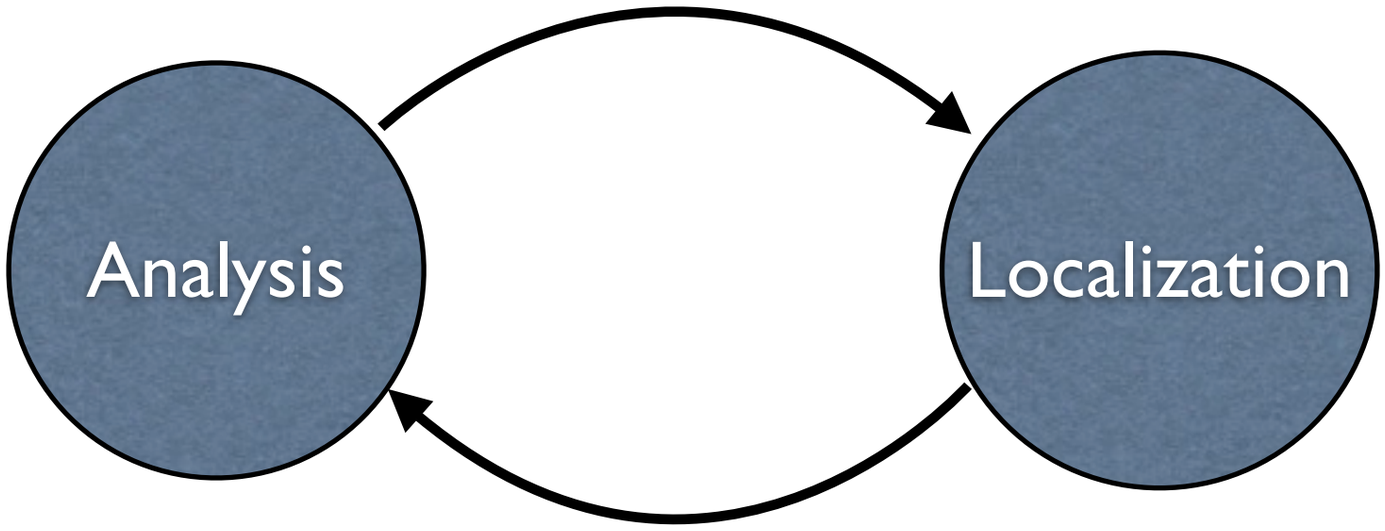
less-382 (23,822 LOC)



On average 755 re-analyses per procedure

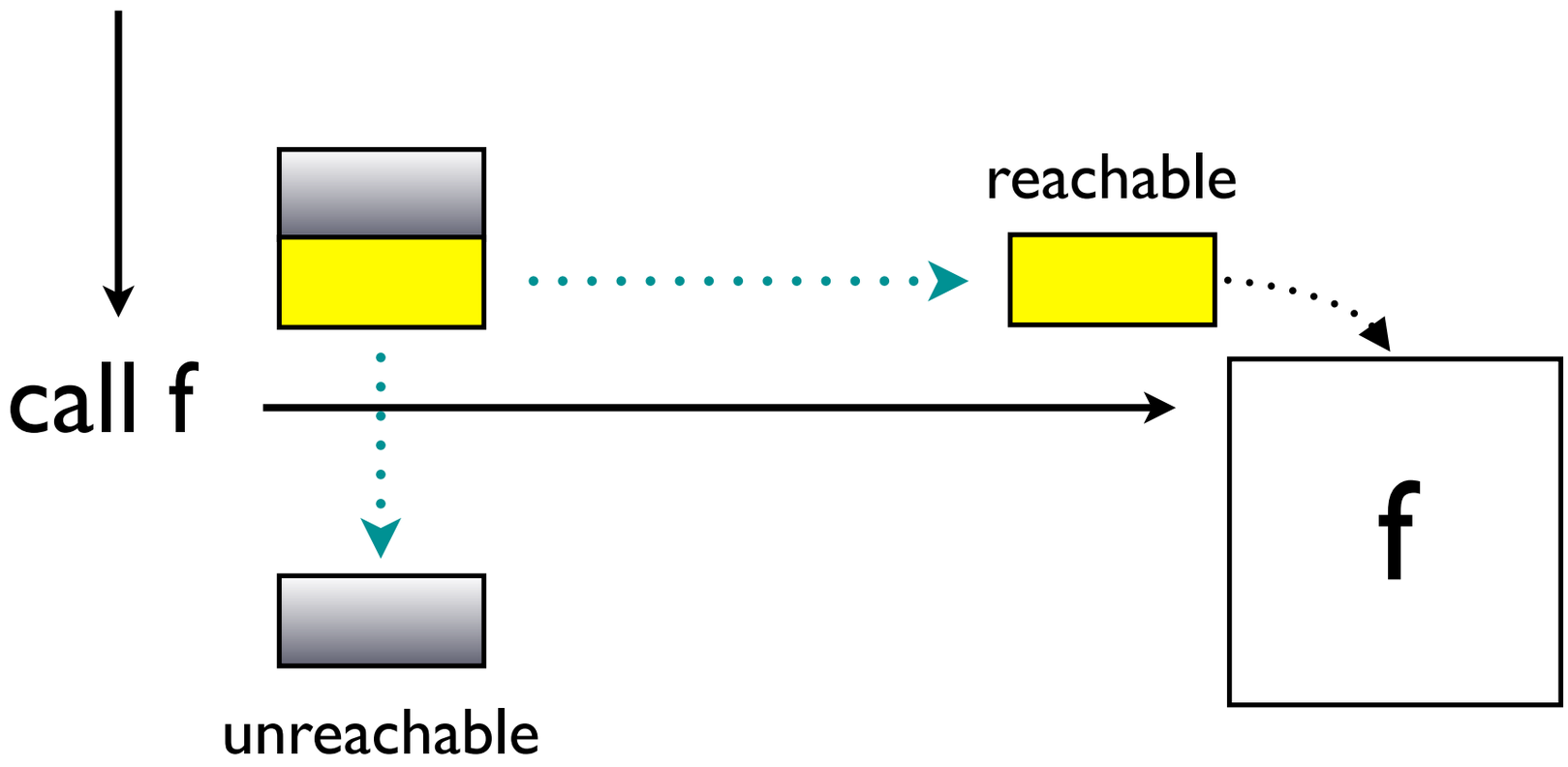
Challenge

The optimal localization is impossible



Reachability-based Localization
(abstract garbage collection)

- Remove the unreachable from params and globals



Reachability is Too Conservative

Program	LOC	accessed memory / reachable memory
spell-1.0	2,213	5 / 453 (1.1%)
barcode-0.96	4,460	19 / 1175 (1.6%)
httptunnel-3.3	6,174	10 / 673 (1.5%)
gzip-1.2.4a	7,327	22 / 1002 (2.2%)
jwhois-3.0.1	9,344	28 / 830 (3.4%)
parser	10,900	75 / 1787 (4.2%)
bc-1.06	13,093	24 / 824 (2.9%)
less-290	18,449	86 / 1546 (5.6%)

average : 4%

VMCAI'11

Access-based Localization*

pre-analysis —(conservative access information)→ actual analysis

Over-approximation of actual access info. {a,b,c}

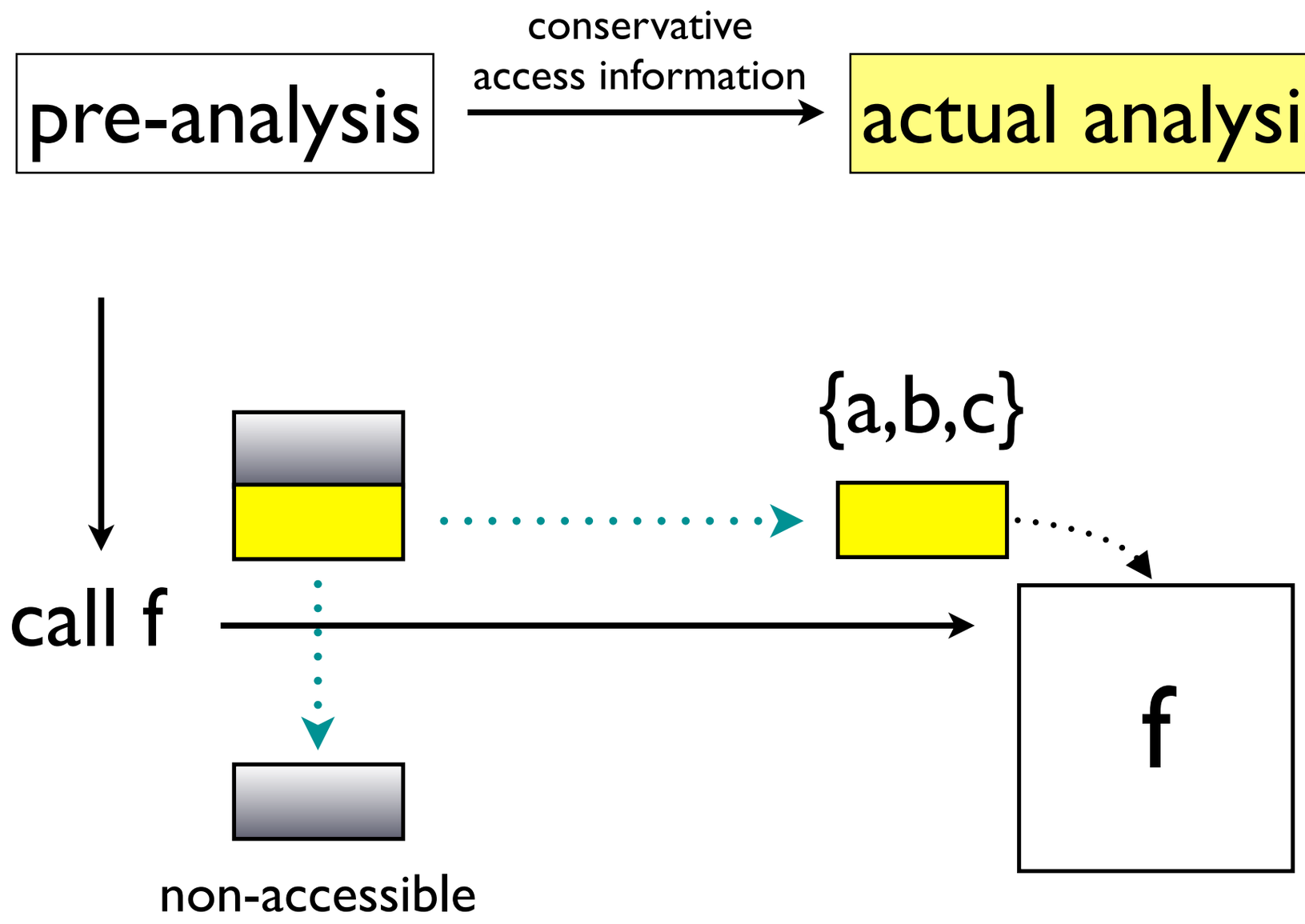
∪

actual access info. {a,b}

f

* Hakjoo Oh, Lucas Brutschy, Kwangkeun Yi, Access analysis-based tight localization of abstract memories, VMCAI'11

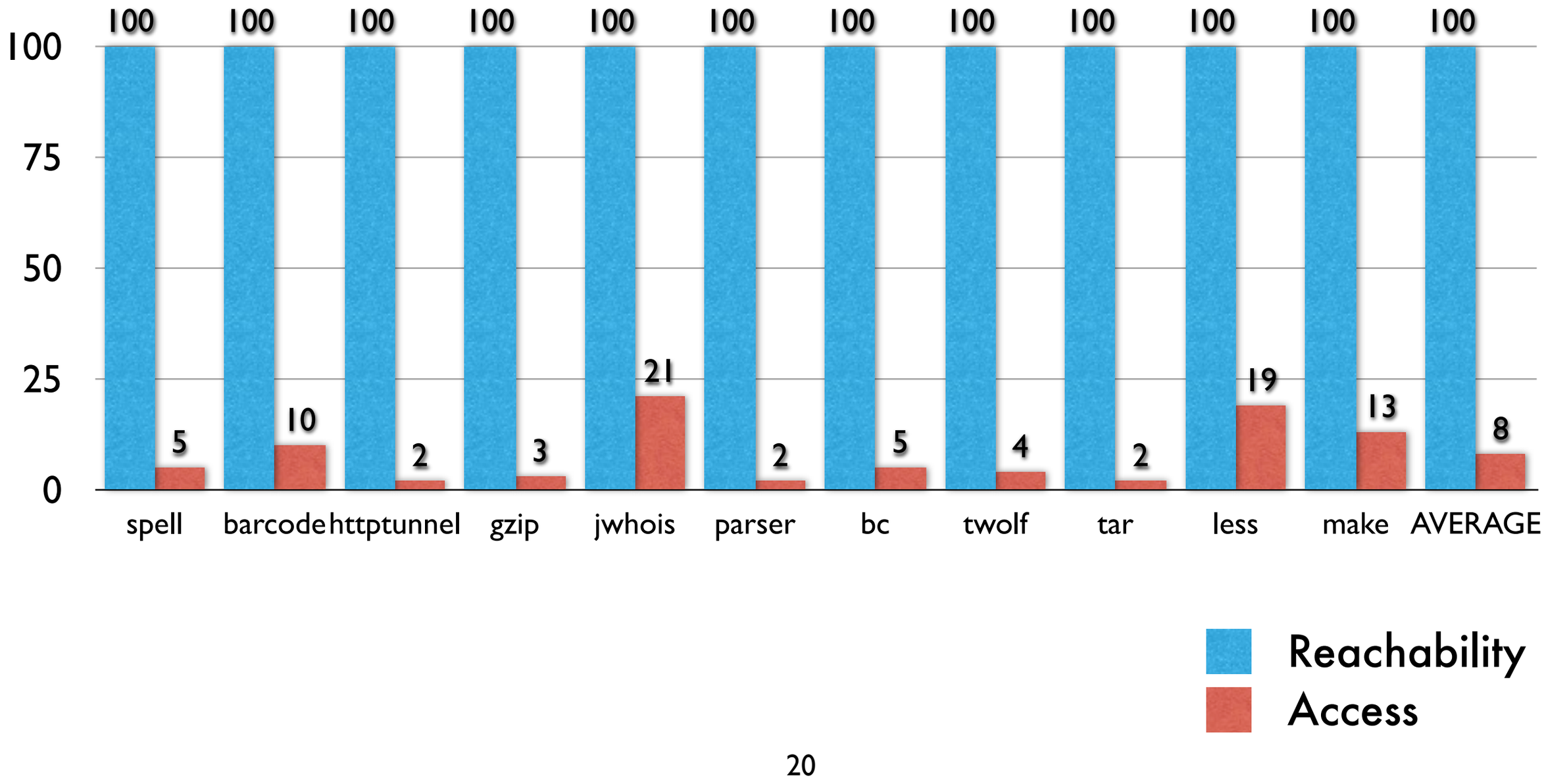
Access-based Localization



VMCAI'11

Performance

5x~50x speed-up over reachability



APLAS'11

Motivation

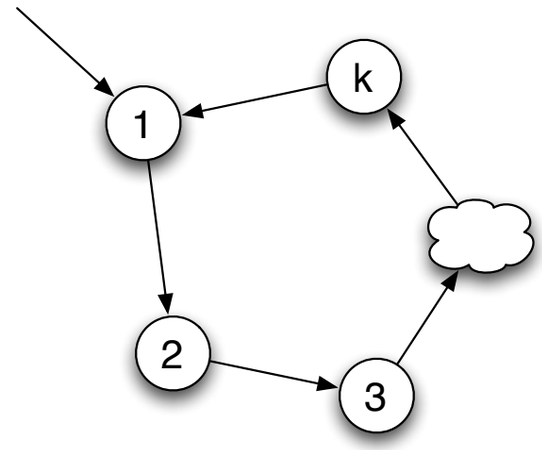
- Access-based localization is sometimes not much effective.

Program	LOC	Baseline	Localized	Speed-Up
twolf	19,700	27,230s	509s	53x
less-382	23,822	137,827s	14,720s	9x
make-3.76	27,304	126,908s	14,681s	8x
bash-2.05a	105,174	oo	391s	n/a

4 hours

Reason: Recursive Call Cycles

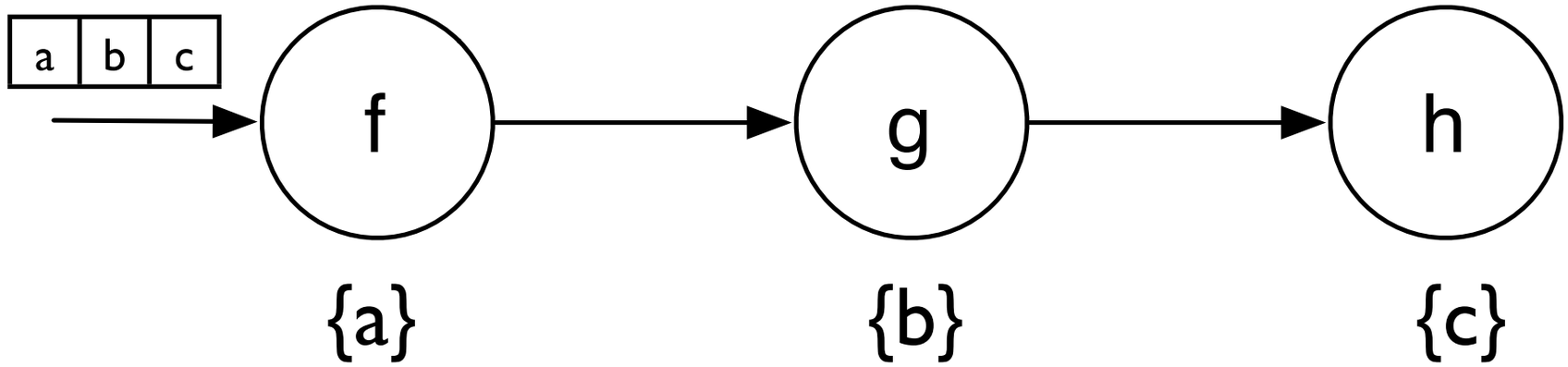
- They contain lots of recursive procedures.
- In particular, large recursive call cycles.



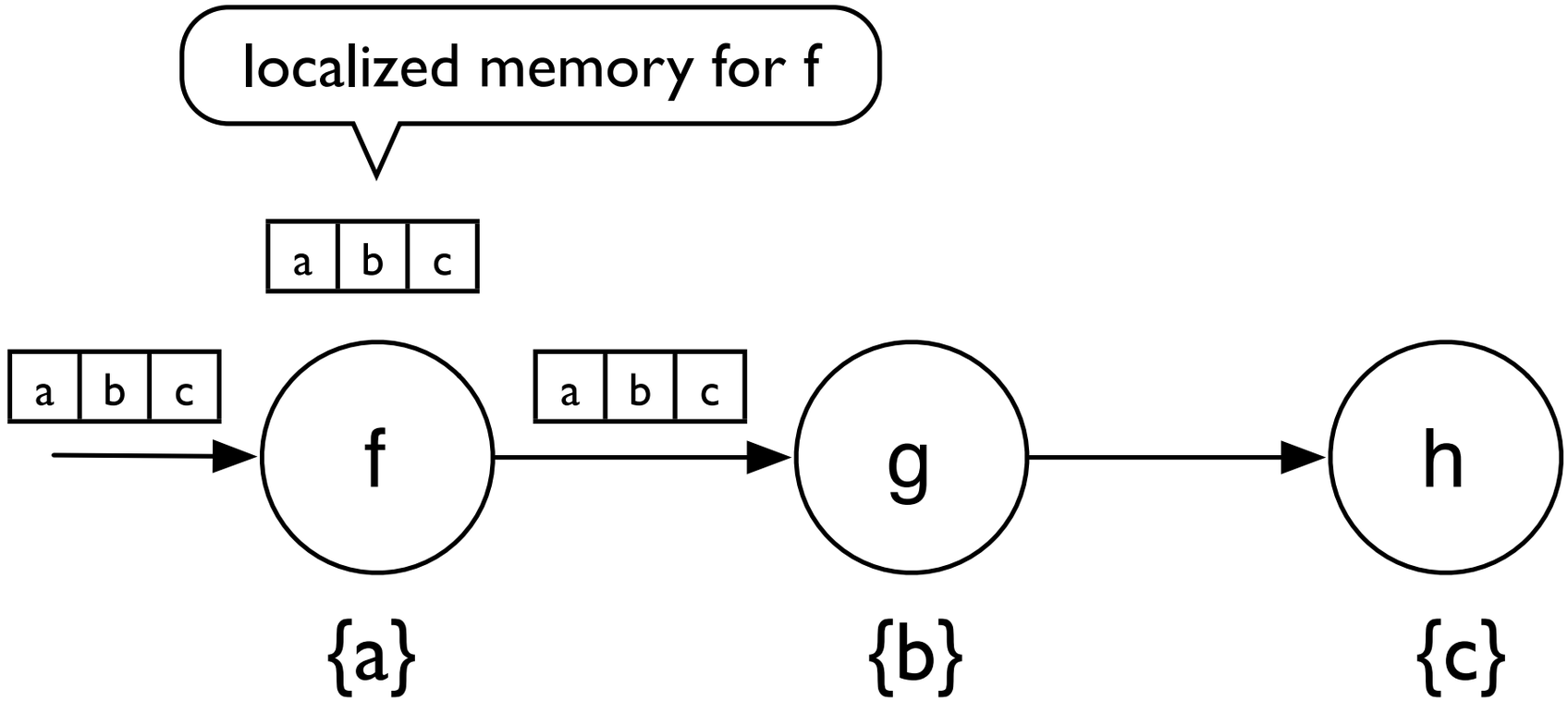
Sizes of the **L**argest **R**ecursive call **C**ycles

Program	LOC	Speed-Up	#procs	LRC
twolf	19,700	53x	192	1
less-382	23,822	9x	382	46
make-3.76	27,304	8x	191	61
bash-2.05a	105,174	n/a	959	4

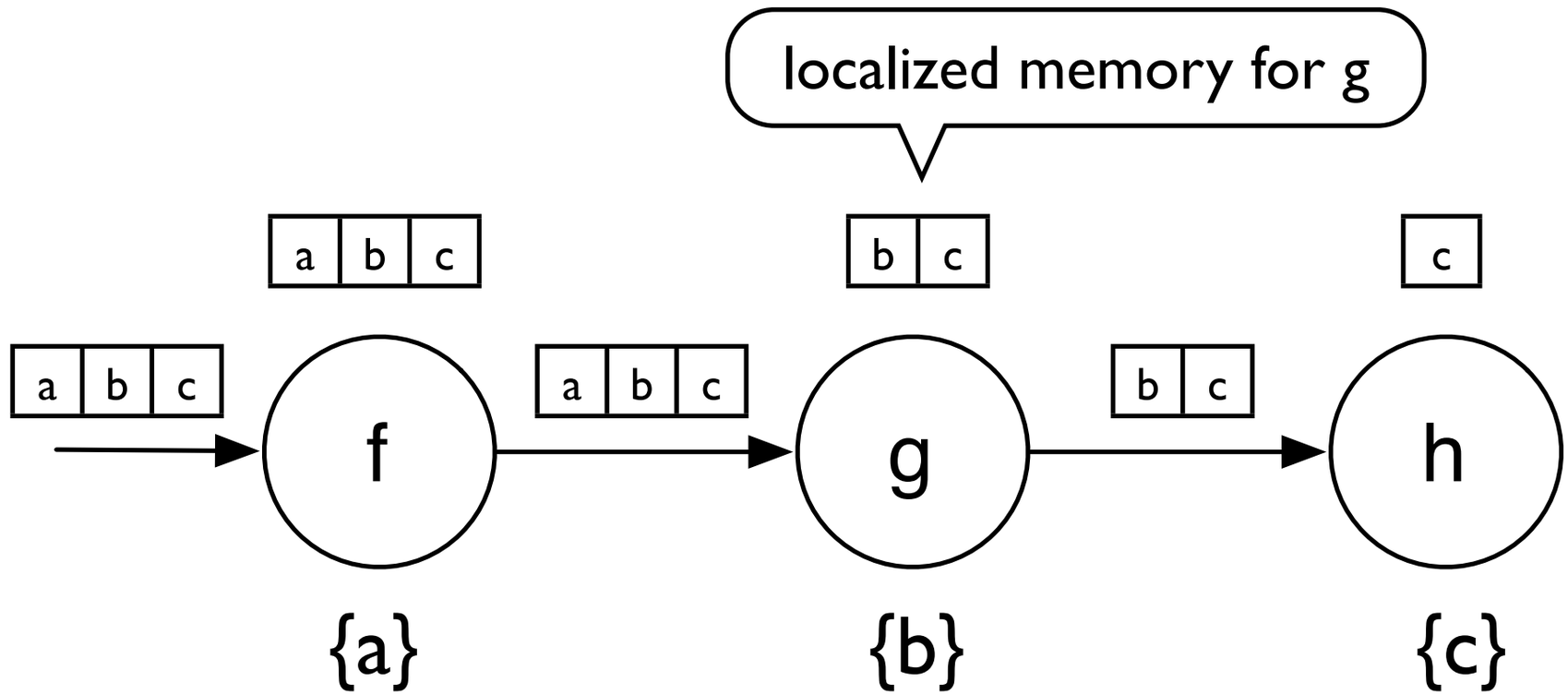
A Source of Inefficiency



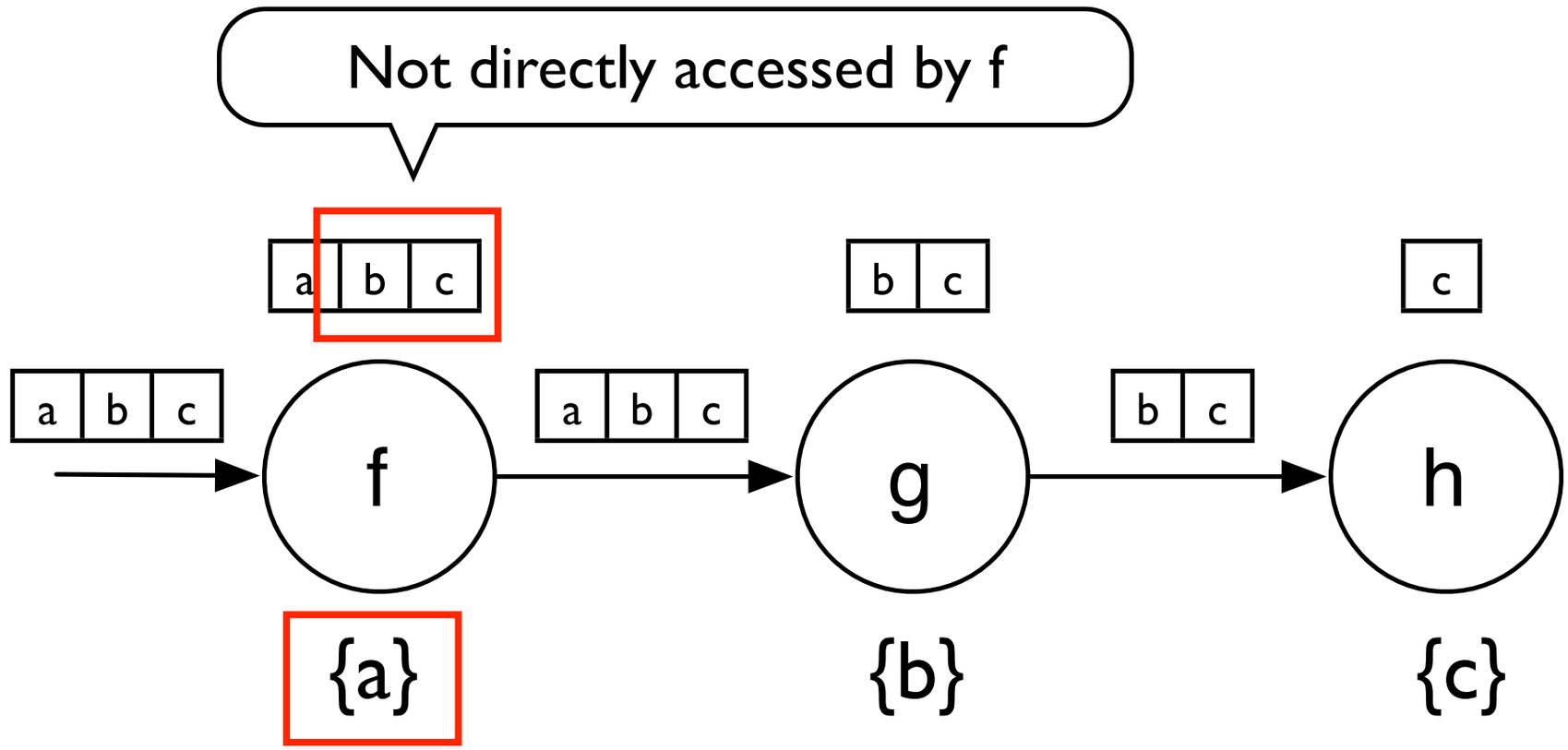
A Source of Inefficiency



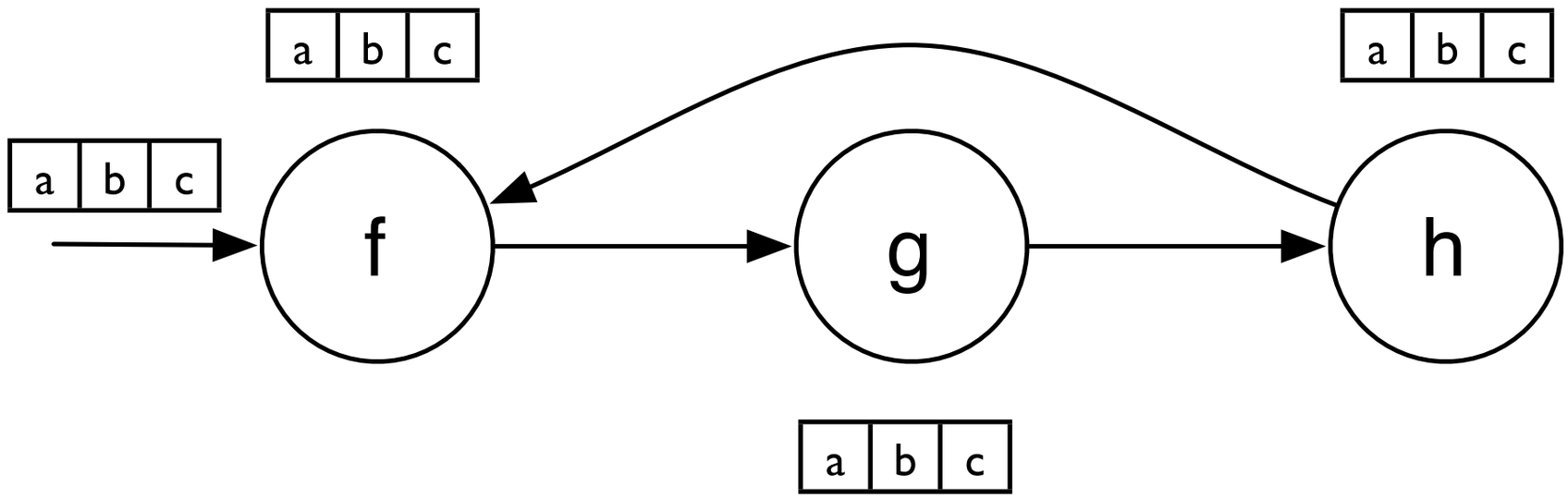
A Source of Inefficiency



A Source of Inefficiency



Recursive Call Cycle



Localization does not work inside recursive cycles!

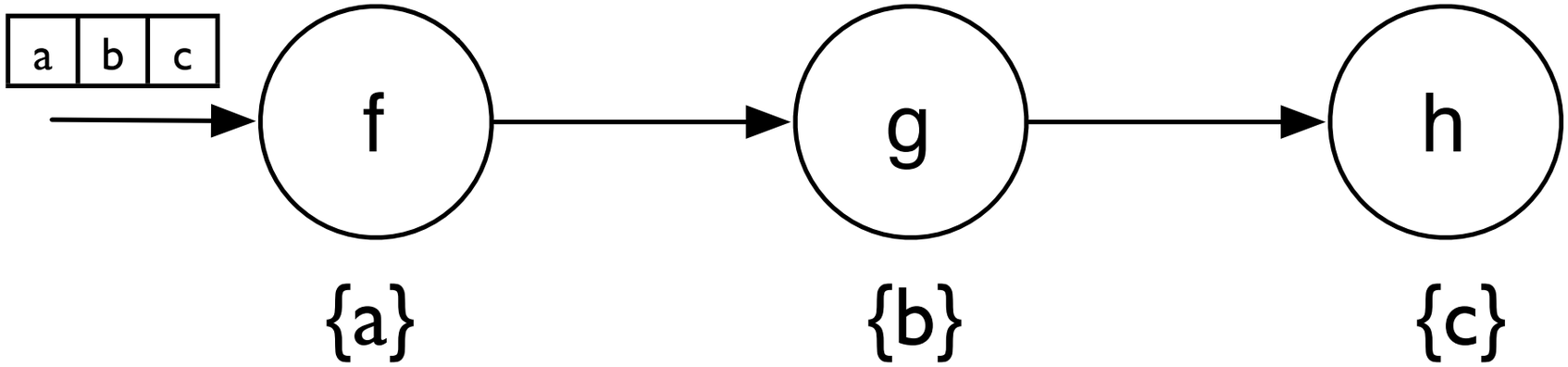
Efficient call cycle analysis is a key

Program	LOC	Functions	LRC
gzip-1.2.4a	7K	132	2
bc-1.06	13K	132	1
tar-1.13	20K	221	13
less-382	23K	382	46
make-3.76.1	27K	190	57
wget-1.9	35K	433	13
screen-4.0.2	45K	588	65
a2ps-4.14	64K	980	6
bash-2.05a	105K	955	4
lsh-2.0.4	111K	1,524	13
sendmail-8.13.6	130K	756	60
nethack-3.3.0	211K	2,207	997
vim60	227K	2,770	1,668
emacs-22.1	399K	3,388	1,554
python-2.5.1	435K	2,996	723
linux-3.0	710K	13,856	493
gimp-2.6	959K	11,728	2
ghostscript-9.00	1,363K	12,993	39

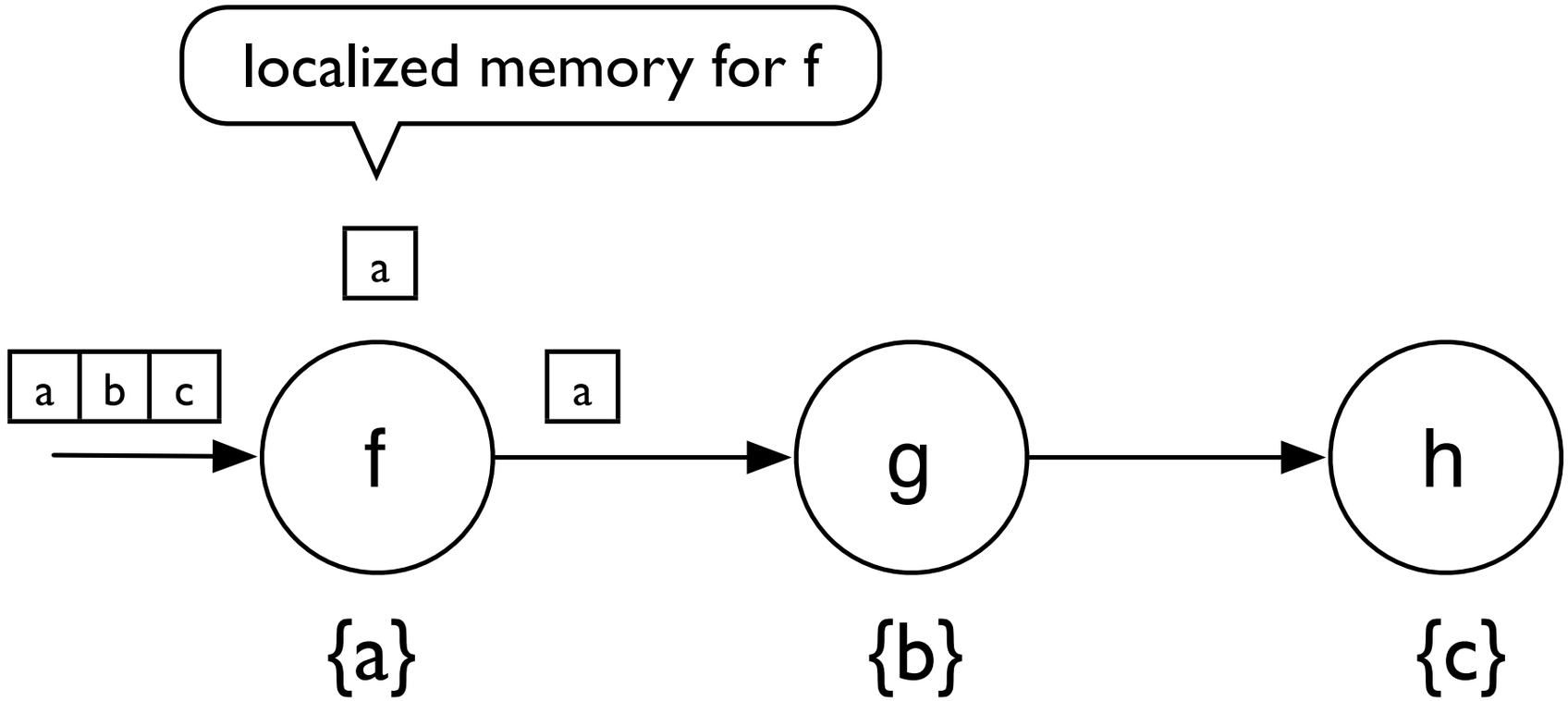
Sizes of the **L**argest **R**ecursive call **C**ycles

APLAS'11

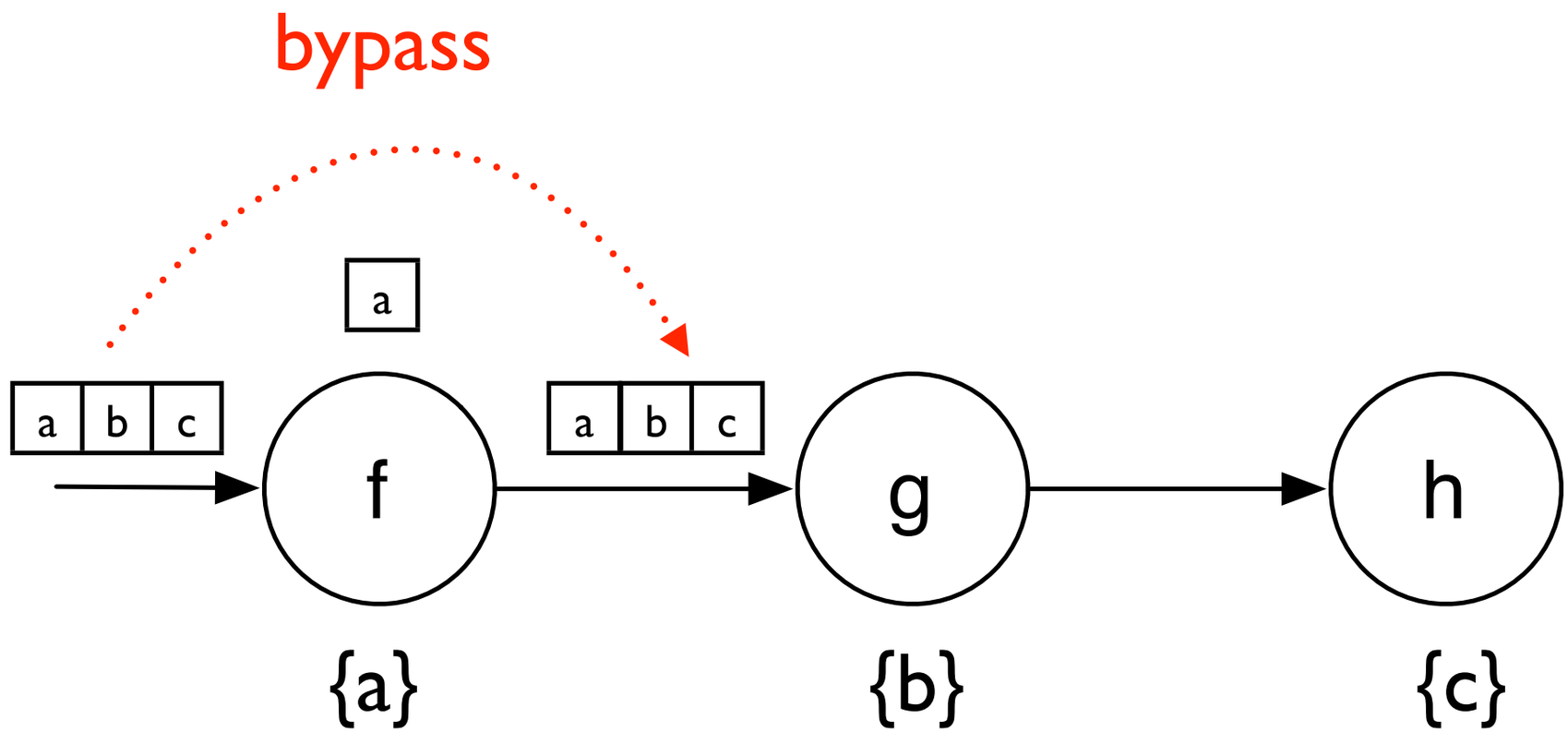
Localization with Bypassing



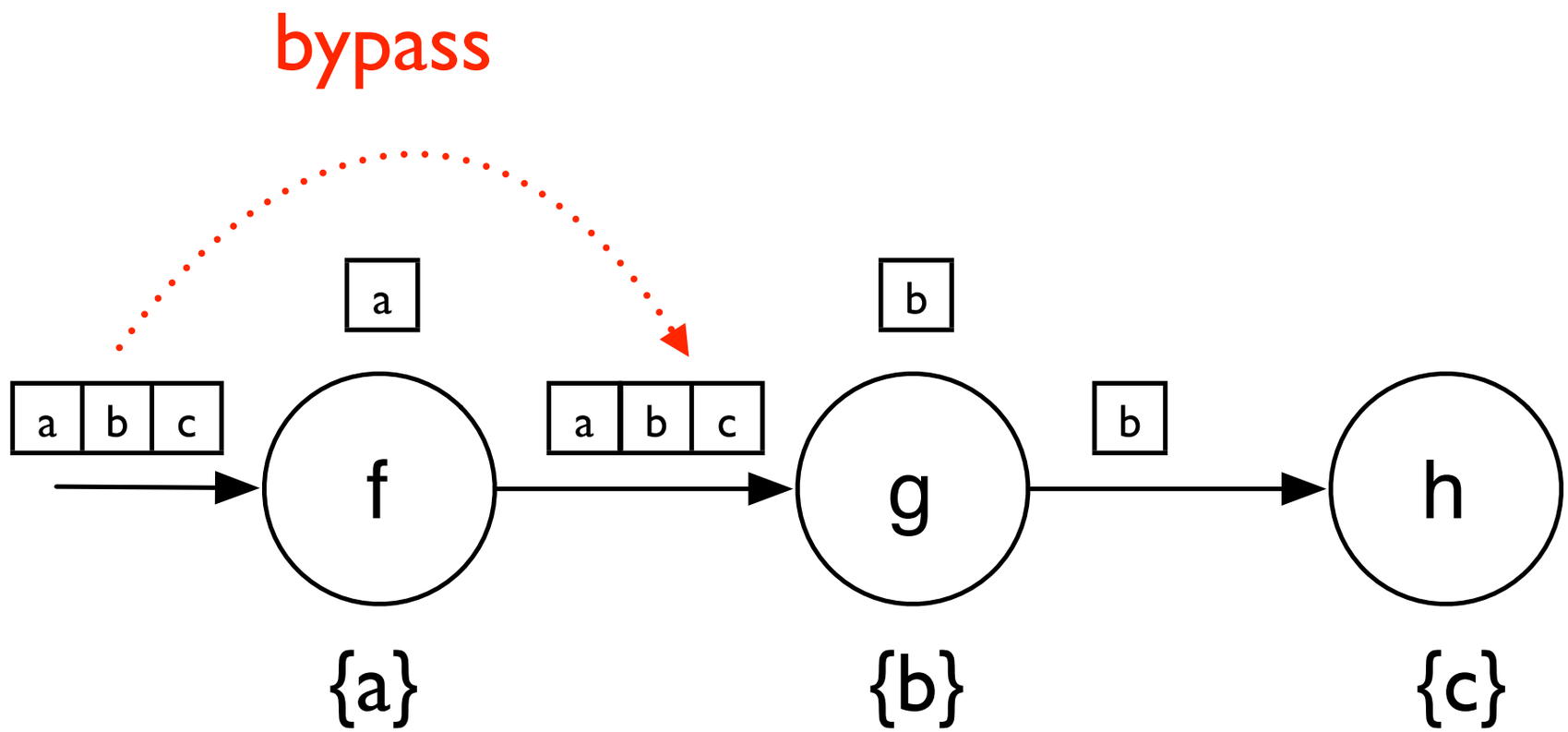
Localization with Bypassing



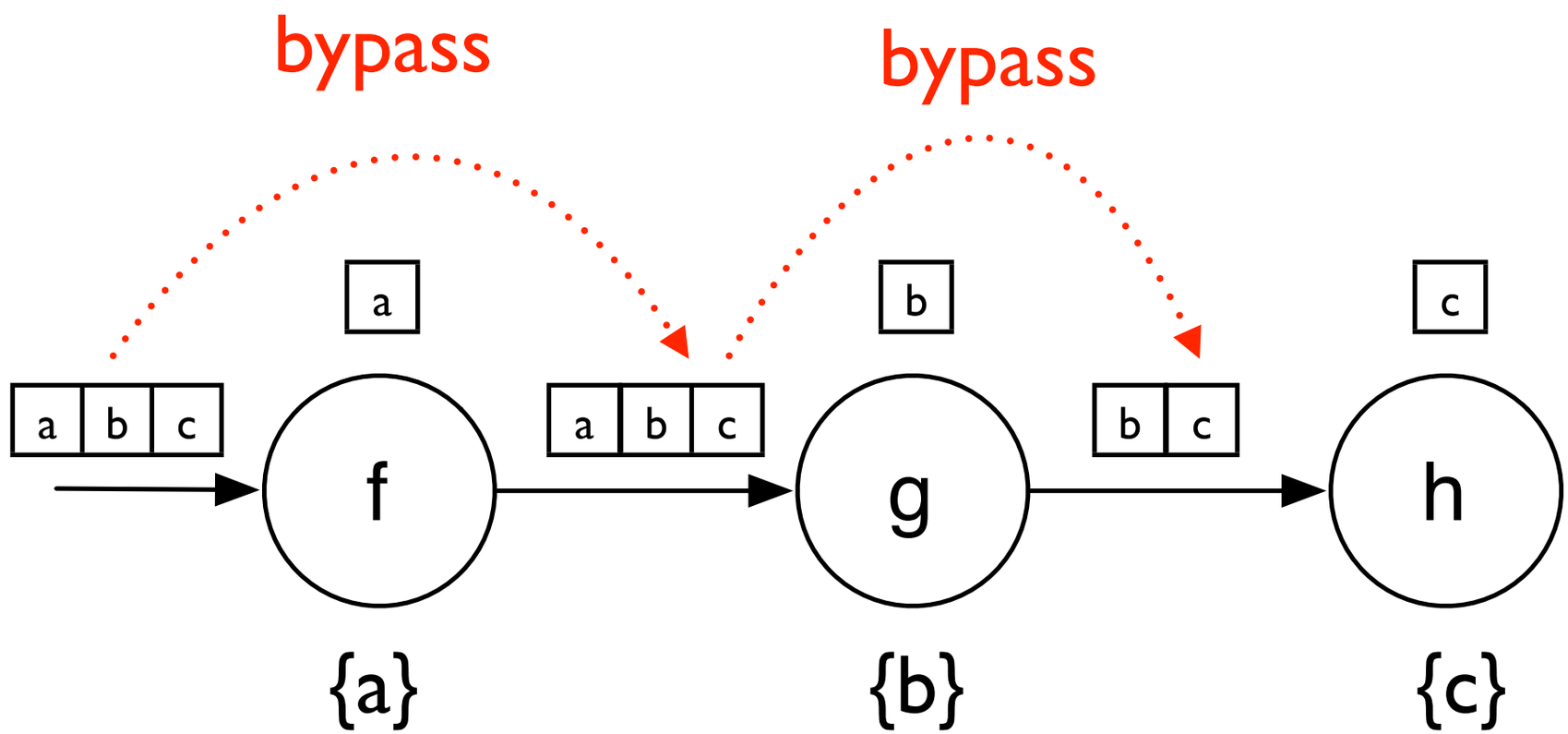
Localization with Bypassing



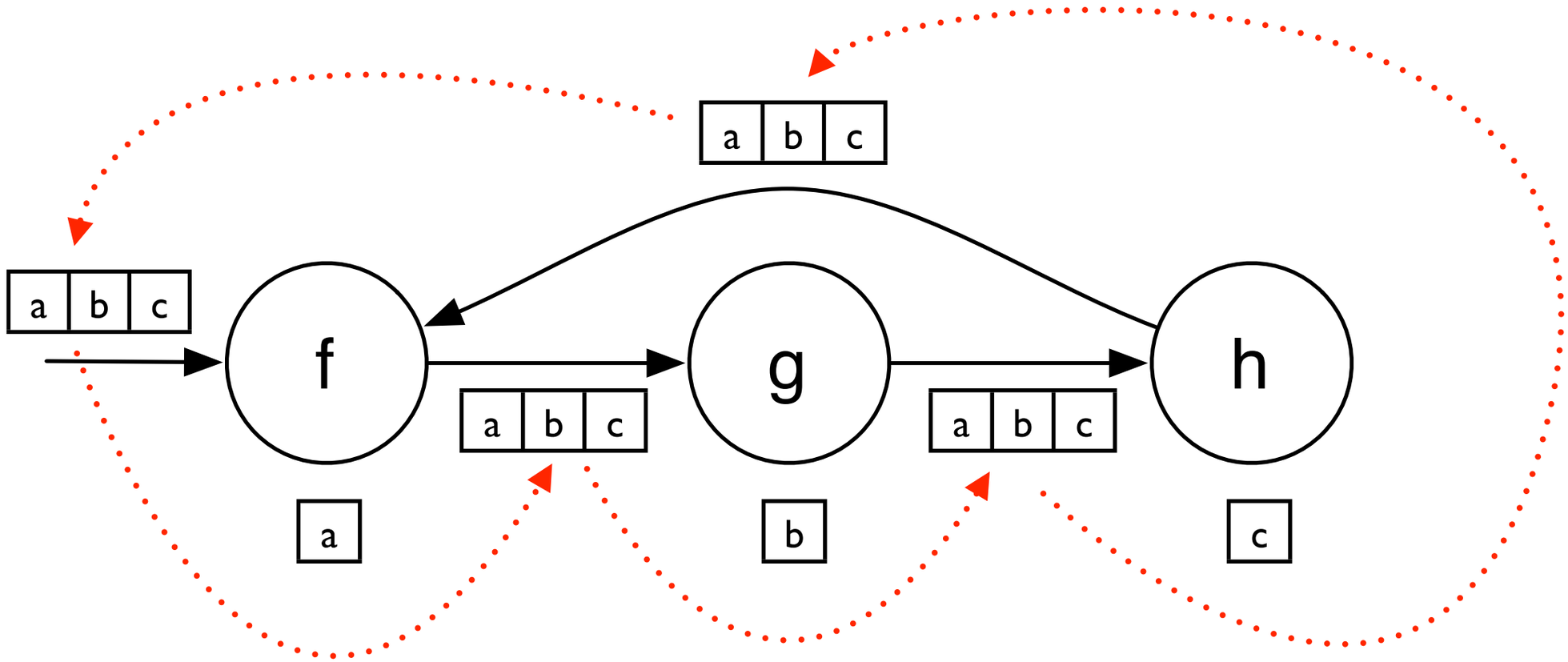
Localization with Bypassing



Localization with Bypassing



Bypassing Call Cycles



Reason for Cost Reduction

```
int a=0, b=0;
void g() { b++; }
void f() { a++; g(); }
int main () {
  b=1; f();          // first call to f
  b=2; f(); }        // second call to f
```

- Localization alone
  - both f and g are re-analyzed
- Localization with bypassing
  - only g is re-analyzed

Even Improve Precision

- In principle, aggressive localization leads to precision improvements.

```
int x;

void g() { x++; }

void f () {
  while (...) { ... }
  g ();
}

void main () {
  x = 0; f ();
  x = 1; f ();
}
```

f does not access x

$x : [0,0] \bigtriangledown [1,1] = [0,+\infty]$

$x : [0,+\infty]$ vs. $[2,2]$

Experiments

- Sparrow: an interval domain-based abstract interpreter
  - AccLoc: access-based localization
  - Bypass: access-based localization with bypassing
- 10 GNU / SPEC 2000 benchmarks
  - 2K~105K lines of code

Results

Sizes of the **L**argest **R**ecursive call **C**ycles

Program	**LOC**	**Proc**	**LRC**	Airac$_{\text{AccLoc}}$		Airac$_{\text{Bypass}}$		*Save*
				time(sec)	*MB*	*time*(sec)	*MB*	(*time*)
spell-1.0	2,213	31	0	2.4	10	1.6	10	31.6%
gzip-1.2.4a	7,327	135	2	51.9	65	37.7	64	27.4%
parser	10,900	325	3	571.6	206	319.4	245	44.1%
bc-1.06	13,093	134	1	496.9	131	318.4	165	35.9%
twolf	19,700	192	1	509.5	212	389.9	212	23.5%
tar-1.13	20,258	222	13	2,407.9	294	1,503.2	338	37.6%
less-382	23,822	382	46	14,720.8	490	4,906.4	427	66.7%
make-3.76.1	27,304	191	61	14,681.9	695	5,248.0	549	64.3%
wget-1.9	35,018	434	13	6,717.5	544	4,383.4	552	34.7%
screen-4.0.2	44,734	589	77	310,788.0	2,228	66,920.6	1,875	78.5%
bash-2.05a	105,174	959	4	1,637.6	272	1,492.4	265	8.9%

Some programs contain large recursive call cycles.

Results

Program	**LOC**	**Proc**	**LRC**	Airac$_{\text{AccLoc}}$		Airac$_{\text{Bypass}}$		*Save*
				time(sec)	*MB*	*time*(sec)	*MB*	(*time*)
spell-1.0	2,213	31	0	2.4	10	1.6	10	31.6%
gzip-1.2.4a	7,327	135	2	51.9	65	37.7	64	27.4%
parser	10,900	325	3	571.6	206	319.4	245	44.1%
bc-1.06	13,093	134	1	496.9	131	318.4	165	35.9%
twolf	19,700	192	1	509.5	212	389.9	212	23.5%
tar-1.13	20,258	222	13	2,407.9	294	1,503.2	338	37.6%
less-382	23,822	382	46	14,720.8	490	4,906.4	427	66.7%
make-3.76.1	27,304	191	61	14,681.9	695	5,248.0	549	64.3%
wget-1.9	35,018	434	13	6,717.5	544	4,383.4	552	34.7%
screen-4.0.2	44,734	589	77	310,788.0	2,228	66,920.6	1,875	78.5%
bash-2.05a	105,174	959	4	1,637.6	272	1,492.4	265	8.9%

For those programs, AccLoc is inefficient.

Results

Program	**LOC**	**Proc**	**LRC**	$\mathsf{Airac}_{\mathsf{AccLoc}}$		$\mathsf{Airac}_{\mathsf{Bypass}}$		*Save*
				time(sec)	*MB*	*time*(sec)	*MB*	(*time*)
spell-1.0	2,213	31	0	2.4	10	1.6	10	31.6%
gzip-1.2.4a	7,327	135	2	51.9	65	37.7	64	27.4%
parser	10,900	325	3	571.6	206	319.4	245	44.1%
bc-1.06	13,093	134	1	496.9	131	318.4	165	35.9%
twolf	19,700	192	1	509.5	212	389.9	212	23.5%
tar-1.13	20,258	222	13	2,407.9	294	1,503.2	338	37.6%
less-382	23,822	382	46	14,720.8	490	4,906.4	427	66.7%
make-3.76.1	27,304	191	61	14,681.9	695	5,248.0	549	64.3%
wget-1.9	35,018	434	13	6,717.5	544	4,383.4	552	34.7%
screen-4.0.2	44,734	589	77	310,788.0	2,228	66,920.6	1,875	78.5%
bash-2.05a	105,174	959	4	1,637.6	272	1,492.4	265	8.9%

For those programs, Bypass is especially effective.
(time reduction of 64~79%)

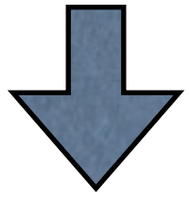
Results

Program	**LOC**	**Proc**	**LRC**	Airac$_{\text{AccLoc}}$		Airac$_{\text{Bypass}}$		*Save*
				time(sec)	*MB*	*time*(sec)	*MB*	(*time*)
spell-1.0	2,213	31	0	2.4	10	1.6	10	31.6%
gzip-1.2.4a	7,327	135	2	51.9	65	37.7	64	27.4%
parser	10,900	325	3	571.6	206	319.4	245	44.1%
bc-1.06	13,093	134	1	496.9	131	318.4	165	35.9%
twolf	19,700	192	1	509.5	212	389.9	212	23.5%
tar-1.13	20,258	222	13	2,407.9	294	1,503.2	338	37.6%
less-382	23,822	382	46	14,720.8	490	4,906.4	427	66.7%
make-3.76.1	27,304	191	61	14,681.9	695	5,248.0	549	64.3%
wget-1.9	35,018	434	13	6,717.5	544	4,383.4	552	34.7%
screen-4.0.2	44,734	589	77	310,788.0	2,228	66,920.6	1,875	78.5%
bash-2.05a	105,174	959	4	1,637.6	272	1,492.4	265	8.9%

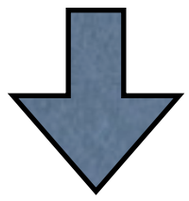
Bypass is also effective for other programs.
(time reduction of 9~44%)

Conclusion

Localization has a problem with recursive cycles



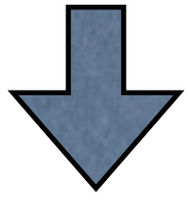
Bypassing mitigates the performance problem



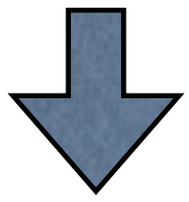
Key to scalability for real C programs

Conclusion

Localization has a problem with recursive cycles



Bypassing mitigates the performance problem



Key to scalability for real C programs

Thank you