An Effective Memory Localization

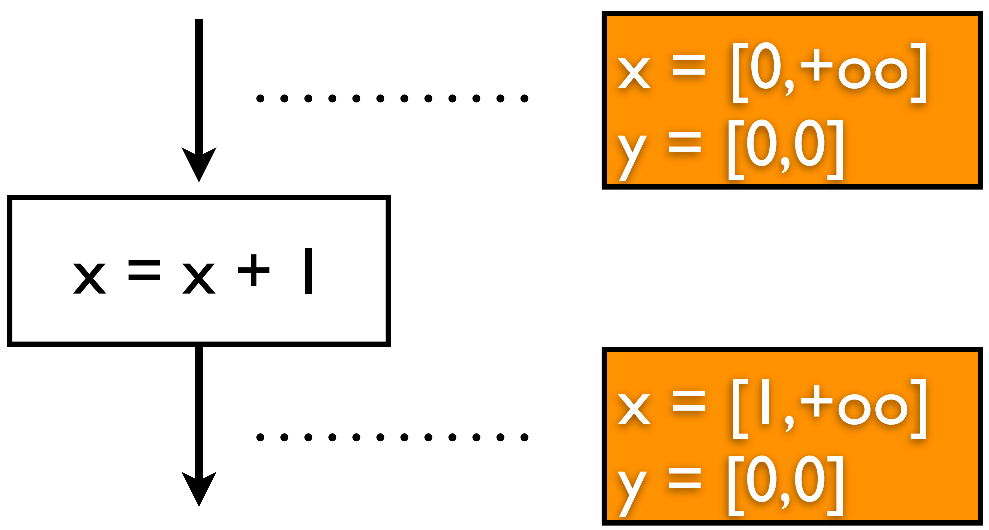
Hakjoo Oh, Lucas Brutschy, and Kwangkeun Yi
{pronto,lucas,kwang}@ropas.snu.ac.kr

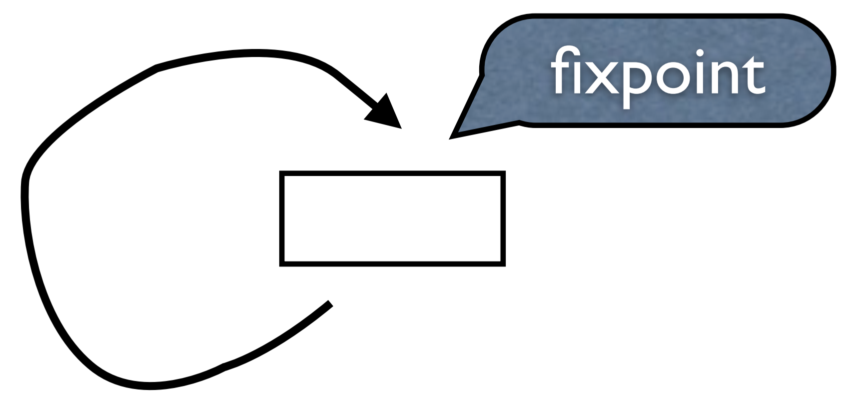
ROSAEC Workshop
2010.01

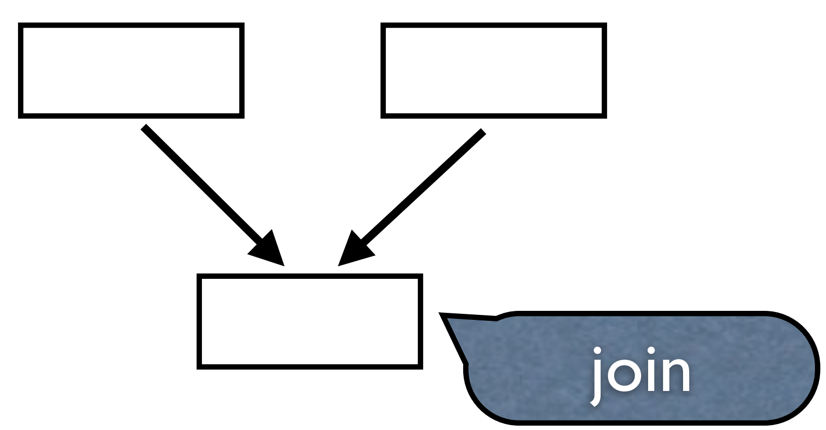
Background

Airac is

- one core engine of Sparrow
- an interval-domain-based abstract interpreter





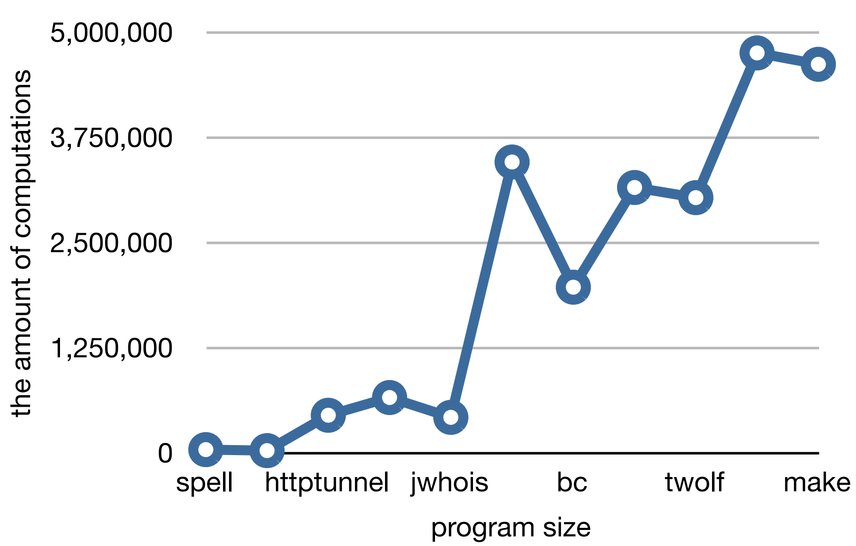


Motivation

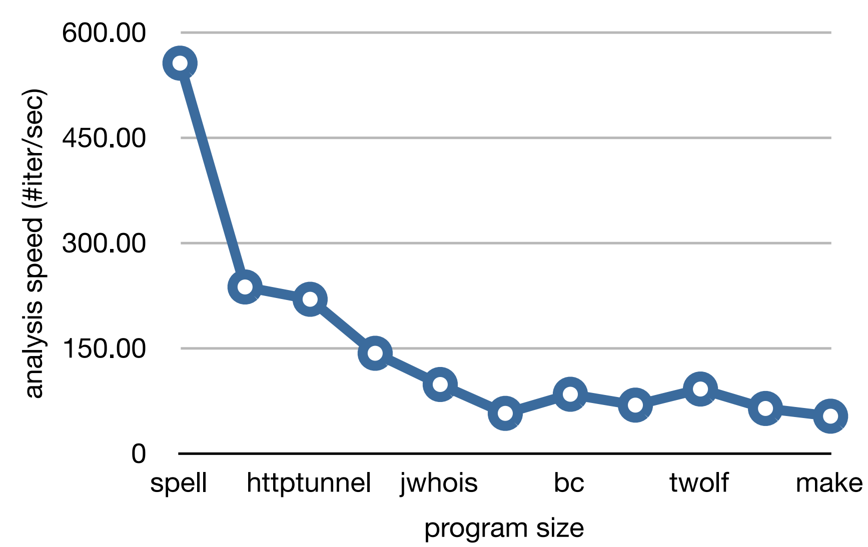
global analysis time

Program	LOC	#Basic-Blocks	Time(s) [1]	#Iterations
gzip-1.2.4a	7,327	6,541	4601.23	653,063
bc-1.06	13,093	9,298	23515.27	1,964,396
less-290	18,449	7,754	46274.67	3,149,284
tar-1.13	20,258	10,800	75013.88	4,748,749
make-3.76.1	27,304	11,061	88221.06	4,613,382

3sec/line

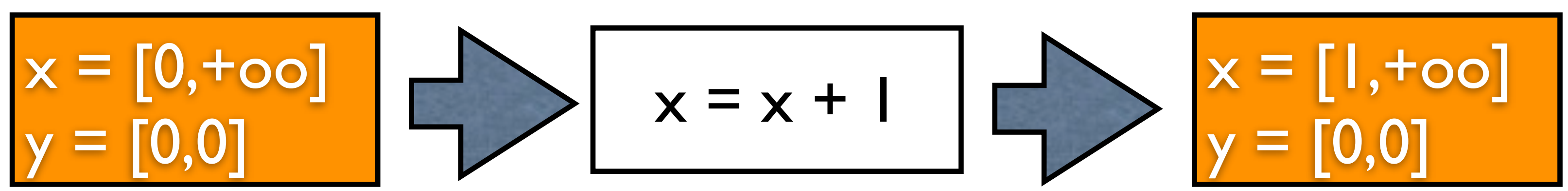


total #iteration



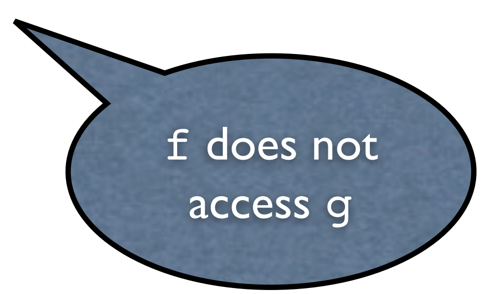
speed (#iteration/sec)

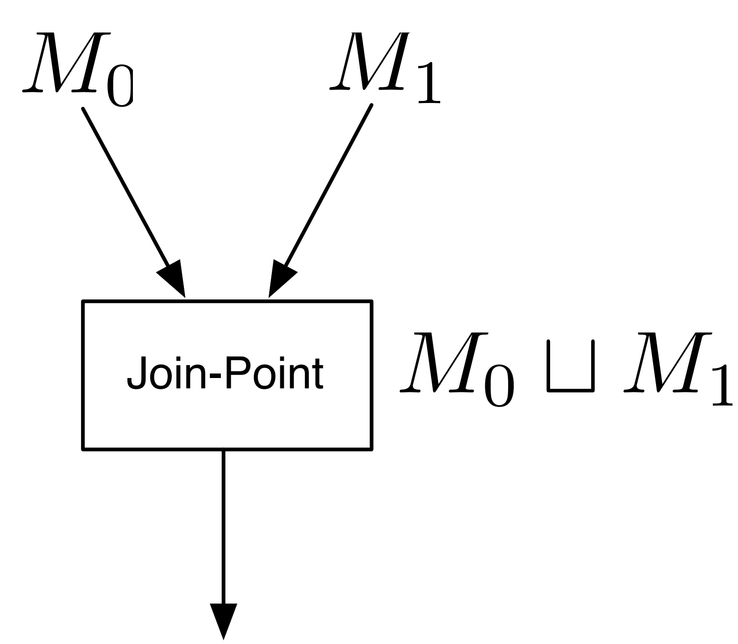
Lack of locality is the problem



- Unnecessary computation increases (#iteration↑)
- Memory operation gets more expensive (speed↓)

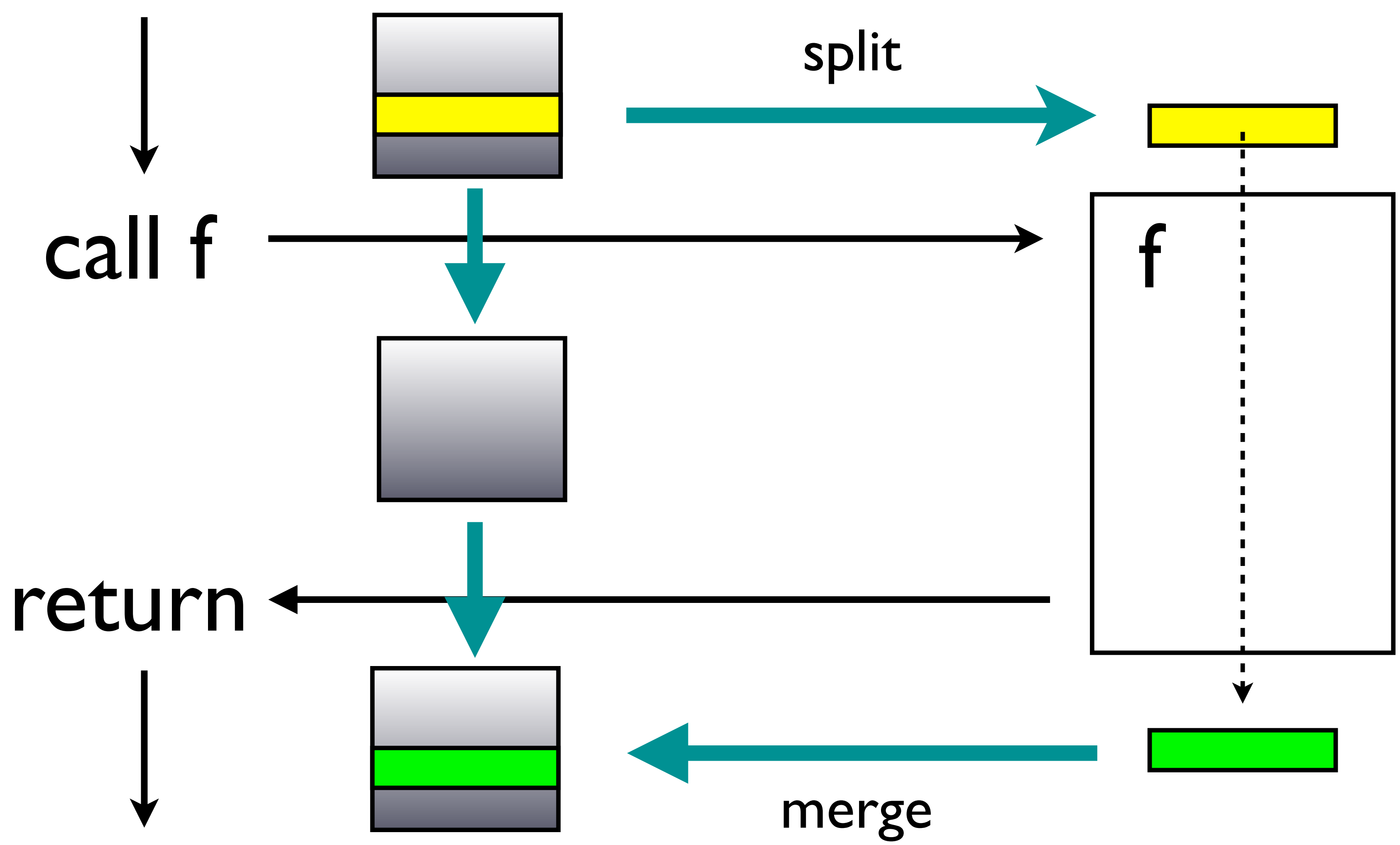
```
int f() {...}
int m() {
  g = 1;
  f();

  g = 2;
  f();
}
```





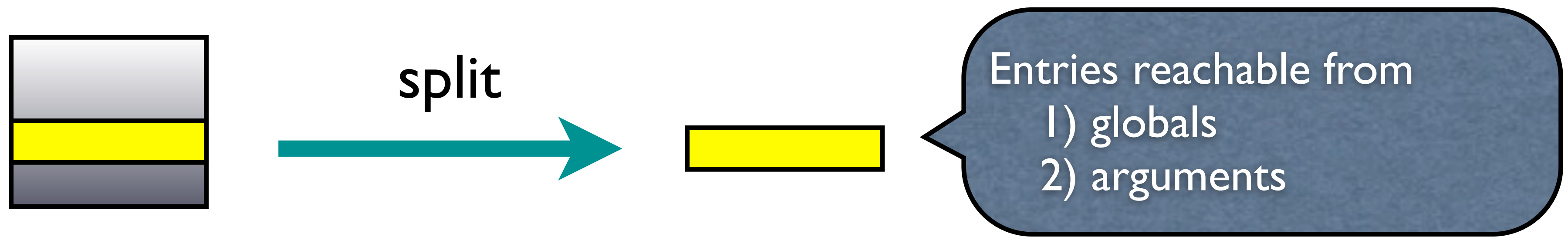
Memory Localization

(a.k.a, static garbage collection or framing in the separation logic)



The Current Standard

(Reachability-based localization)




```
struct S { int a; int b; }
int g;
void f (S* p) { p->a = 1; }
void main() {
    struct S *s = (S*)malloc(sizeof(struct S));
    g = 0;
    s->a = 0;
    s->b = 0;
    f(s); }
```

$$\begin{aligned} s &\mapsto \langle l_5, \{a, b\}\rangle \\ \langle l_5, a\rangle &\mapsto \bot \\ \langle l_5, b\rangle &\mapsto \bot \end{aligned}$$

$$\begin{aligned} p &\mapsto \langle l_5, \{a, b\}\rangle \\ s &\mapsto \langle l_5, \{a, b\}\rangle \\ \langle l_5, a\rangle &\mapsto [0, 0] \\ \langle l_5, b\rangle &\mapsto [0, 0] \\ g &\mapsto [0, 0] \end{aligned}$$

$$\begin{aligned} p &\mapsto \langle l_5, \{a, b\}\rangle \\ \langle l_5, a\rangle &\mapsto [0, 0] \\ \langle l_5, b\rangle &\mapsto [0, 0] \\ g &\mapsto [0, 0] \end{aligned}$$

The Current Standard

(Reachability-based localization)

split

Entries reachable from

Pgm	LOC	No Local	Reach
gzip	7,327	12315s	2182s
twolf	19,700	∞	19214s
bash	105,174	∞	∞

```
stru
int
void
void


    s->a = 0;
    s->b = 0;
    f(s); }
```

$\mapsto \langle l_5, \{a, b\}\rangle$
$\mapsto \bot$
$\mapsto \bot$

$$p \mapsto \langle l_5, \{a, b\}\rangle$$
$$s \mapsto \langle l_5, \{a, b\}\rangle$$
$$\langle l_5, a\rangle \mapsto [0, 0]$$
$$\langle l_5, b\rangle \mapsto [0, 0]$$
$$g \mapsto [0, 0]$$

$$p \mapsto \langle l_5, \{a, b\}\rangle$$
$$\langle l_5, a\rangle \mapsto [0, 0]$$
$$\langle l_5, b\rangle \mapsto [0, 0]$$
$$g \mapsto [0, 0]$$

Too Conservative in Practice

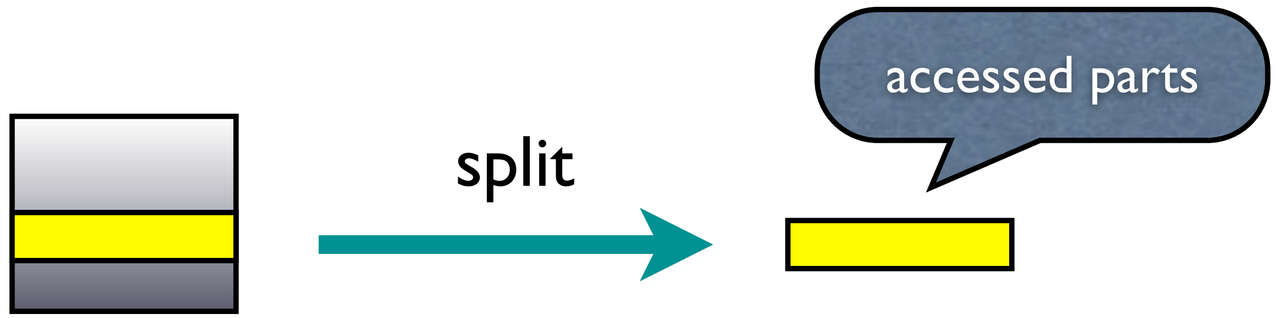
Just small number of

- global variables
- reachable heap locations

will be actually accessed by a procedure.

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%)

Our Approach

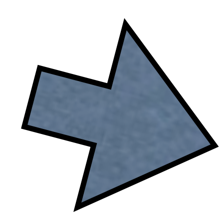


$$\{p, \langle l_5, a\rangle\}$$

```
1: struct S { int a; int b; }
2: int g;
3: void f (S* p) { p->a = 1; }
4: void main() {
5:     struct S *s = (S*)malloc(sizeof(struct S));
6:     g = 0;
7:     s->a = 0;
8:     s->b = 0;
9:     f(s); }
```

$$\begin{aligned} s &\mapsto \langle l_5, \{a, b\}\rangle \\ \langle l_5, a\rangle &\mapsto \bot \\ \langle l_5, b\rangle &\mapsto \bot \end{aligned}$$

$$\begin{aligned} p &\mapsto \langle l_5, \{a, b\}\rangle \\ s &\mapsto \langle l_5, \{a, b\}\rangle \\ \langle l_5, a\rangle &\mapsto [0, 0] \\ \langle l_5, b\rangle &\mapsto [0, 0] \\ g &\mapsto [0, 0] \end{aligned}$$



$$\begin{aligned} p &\mapsto \langle l_5, \{a, b\}\rangle \\ \langle l_5, a\rangle &\mapsto [0, 0] \end{aligned}$$

Our Approach

$\{p, \langle l_5, a\rangle\}$

Pgm	LOC	Reach	Access	Save
gzip	7,327	2182s	94s	95.7%
twolf	19,700	19214s	600s	96.9%
bash	105,174	∞	702s	n/a

```
f(s); }
```

$l_5, \{a, b\}\rangle$

$$
\begin{aligned}
p &\mapsto \langle l_5, \{a, b\}\rangle \\
s &\mapsto \langle l_5, \{a, b\}\rangle \\
\langle l_5, a\rangle &\mapsto [0, 0] \\
\langle l_5, b\rangle &\mapsto [0, 0] \\
g &\mapsto [0, 0]
\end{aligned}
$$

$$
\begin{aligned}
p &\mapsto \langle l_5, \{a, b\}\rangle \\
\langle l_5, a\rangle &\mapsto [0, 0]
\end{aligned}
$$

Computing Accessed Locations

- Staging the analysis

preliminary analysis → (access info) → actual analysis

```
struct S { int a; int b; }
int g;
void f (S* p) { p->a = 1; }
void main() {
    struct S *s = (S*)malloc(sizeof(struct S));
    g = 0;
    s->a = 0;
    s->b = 0;
    f(s); }
```

over-approximation of all the possible memory states

$$
\begin{aligned}
p &\mapsto \langle l_5, \{a, b\}\rangle \\
s &\mapsto \langle l_5, \{a, b\}\rangle \\
\langle l_5, a\rangle &\mapsto [-\infty, +\infty] \\
\langle l_5, b\rangle &\mapsto [-\infty, +\infty] \\
g &\mapsto [-\infty, +\infty]
\end{aligned}
$$

Collect the accessed locations using the memory

`p->a = 1`

$$\{p, \langle l_5, a\rangle\}$$

dereference

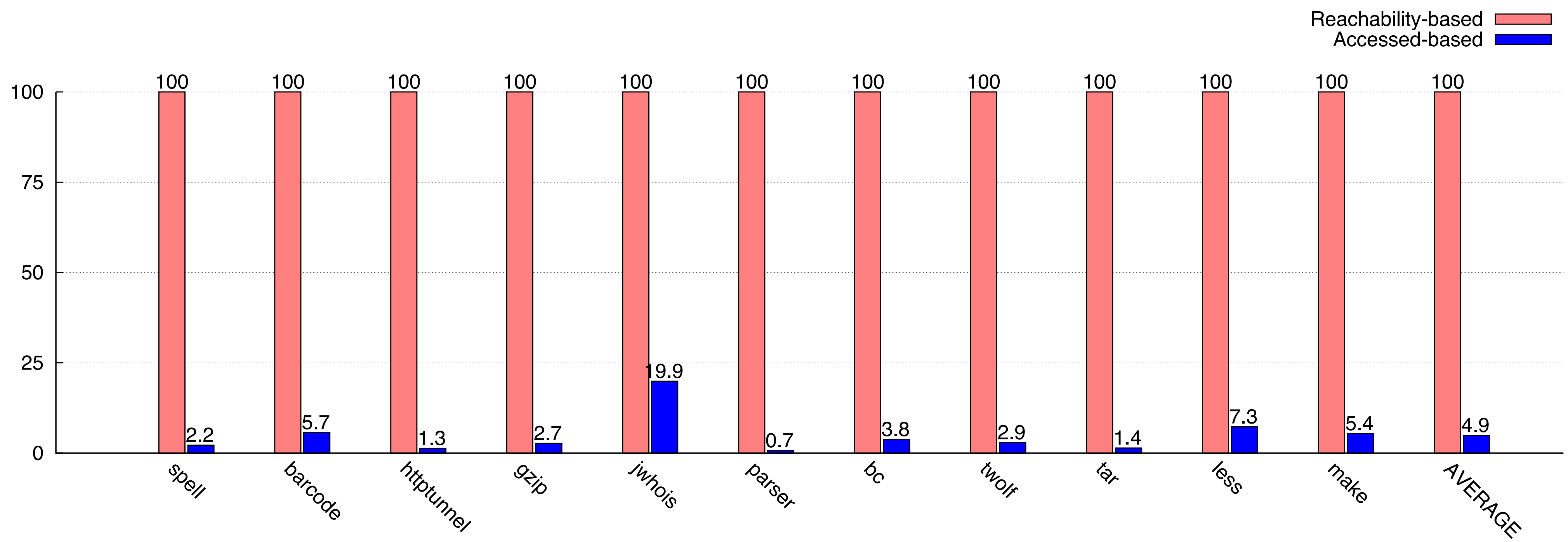
field access

Analysis Time

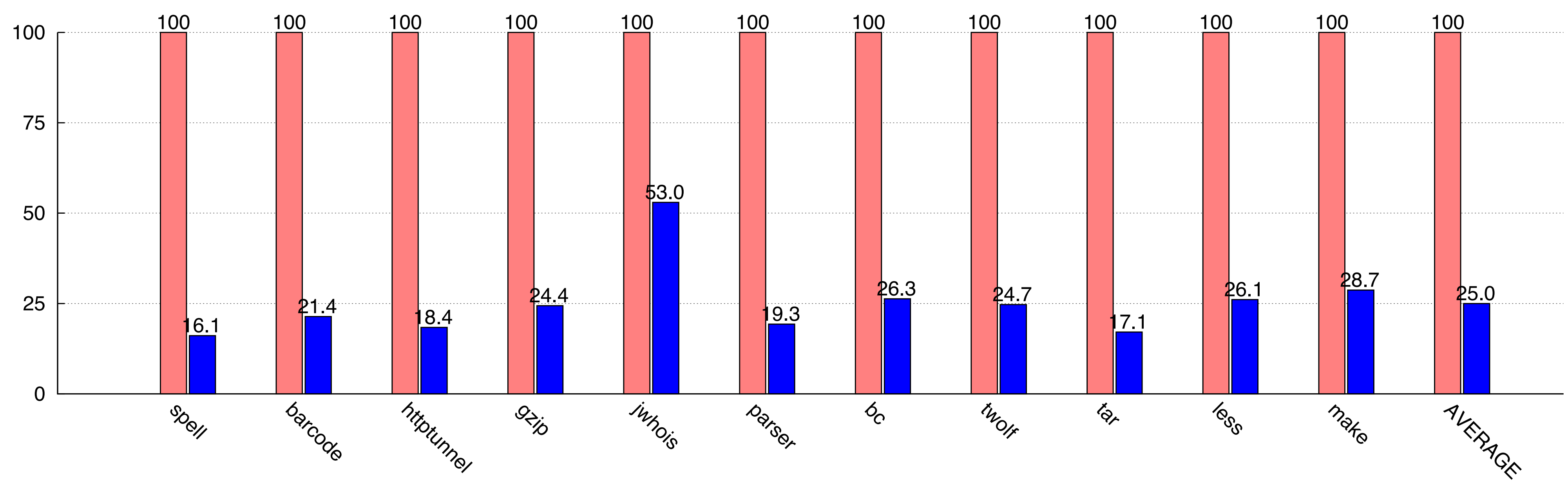
- Reach reduces the time by on average 45.0%
  - Sometimes, 7.7x speed-up
- Our approach reduces the time by on average 97.4%
  - Sometimes, 1000x speed-up

Reach vs. Ours

Time

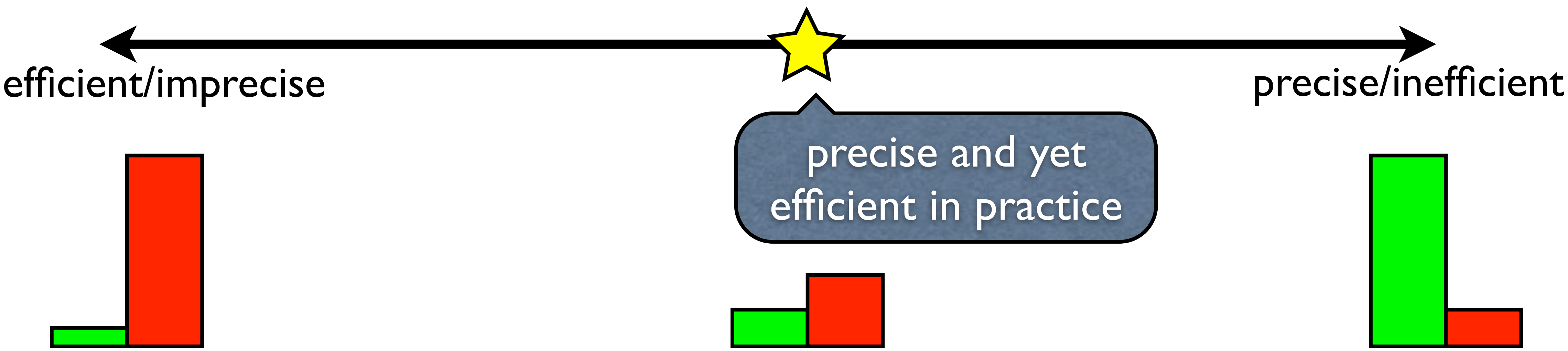


Peak Mem



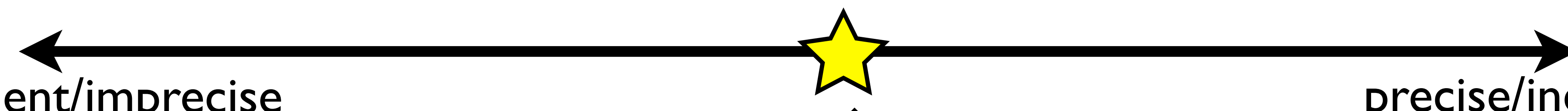
- Our approach reduces
  - the time by on average 95.1% (up to 144x speed-up)
  - peak memory by on average 75.0%

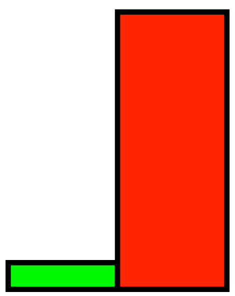
Balancing pre/actual Analysis



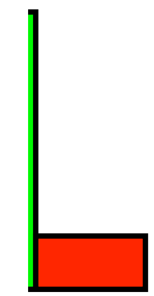
- Our pre-analysis is
  - efficient enough thanks to flow-insensitivity
  - precise enough because we do not abstract address domain
    - tracking variables as well as heap locations

Balancing pre/actual Analysis





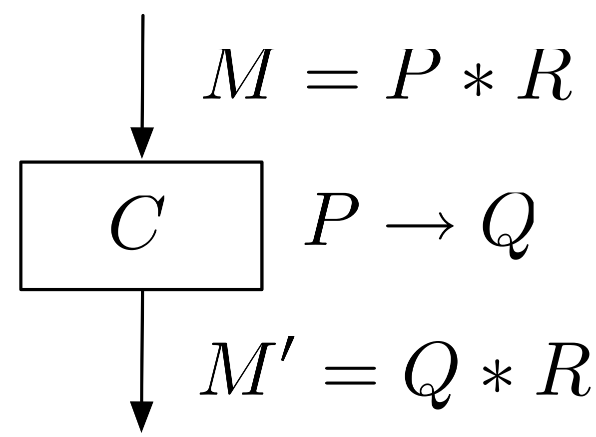
Pgm	LOC	Reach	Access	Pre
gzip	7,327	2182s	94s	1.6s
twolf	19,700	19214s	600s	5.7s
bash	105,174	∞	702s	19.0s



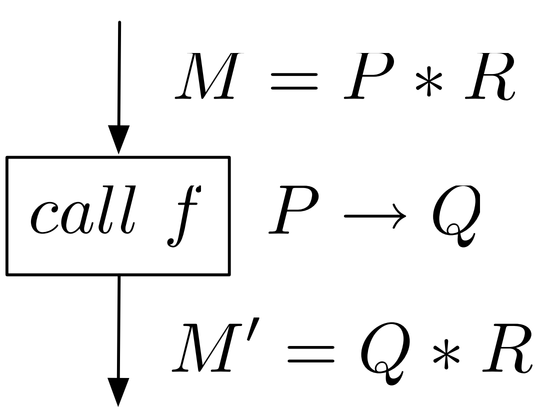
- Ou
  - efficient enough thanks to flow insensitivity
  - precise enough because we do not abstract address domain
    - tracking variables as well as heap locations

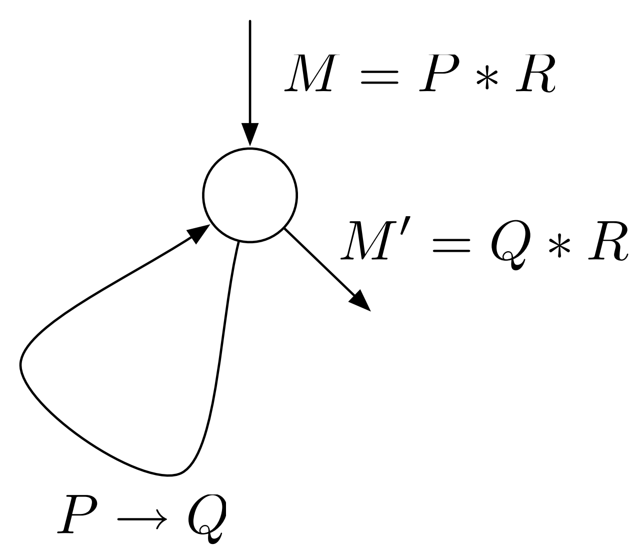
Generalized Localization

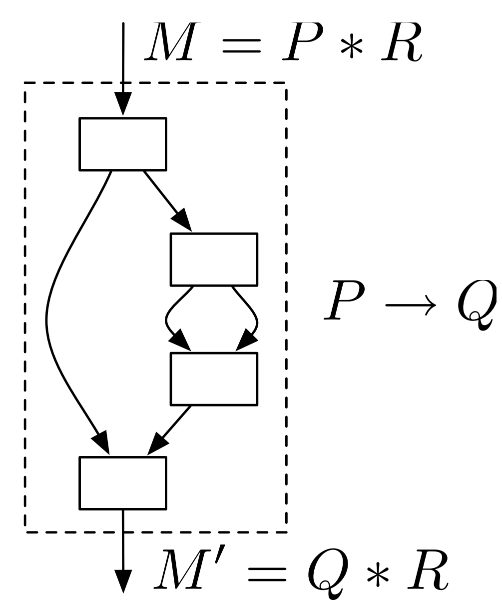
We can apply our localization for any code part



Examples







Example

```
int g;
int buf[10];

void f(int p) {
    g = p;
    for(i=0;i<9;i++)
    {
        buf[i] = 0;
        ...
    }
}

int main() {
    f(1);
    f(2);
}
```

f is analyzed twice even with localization

Example

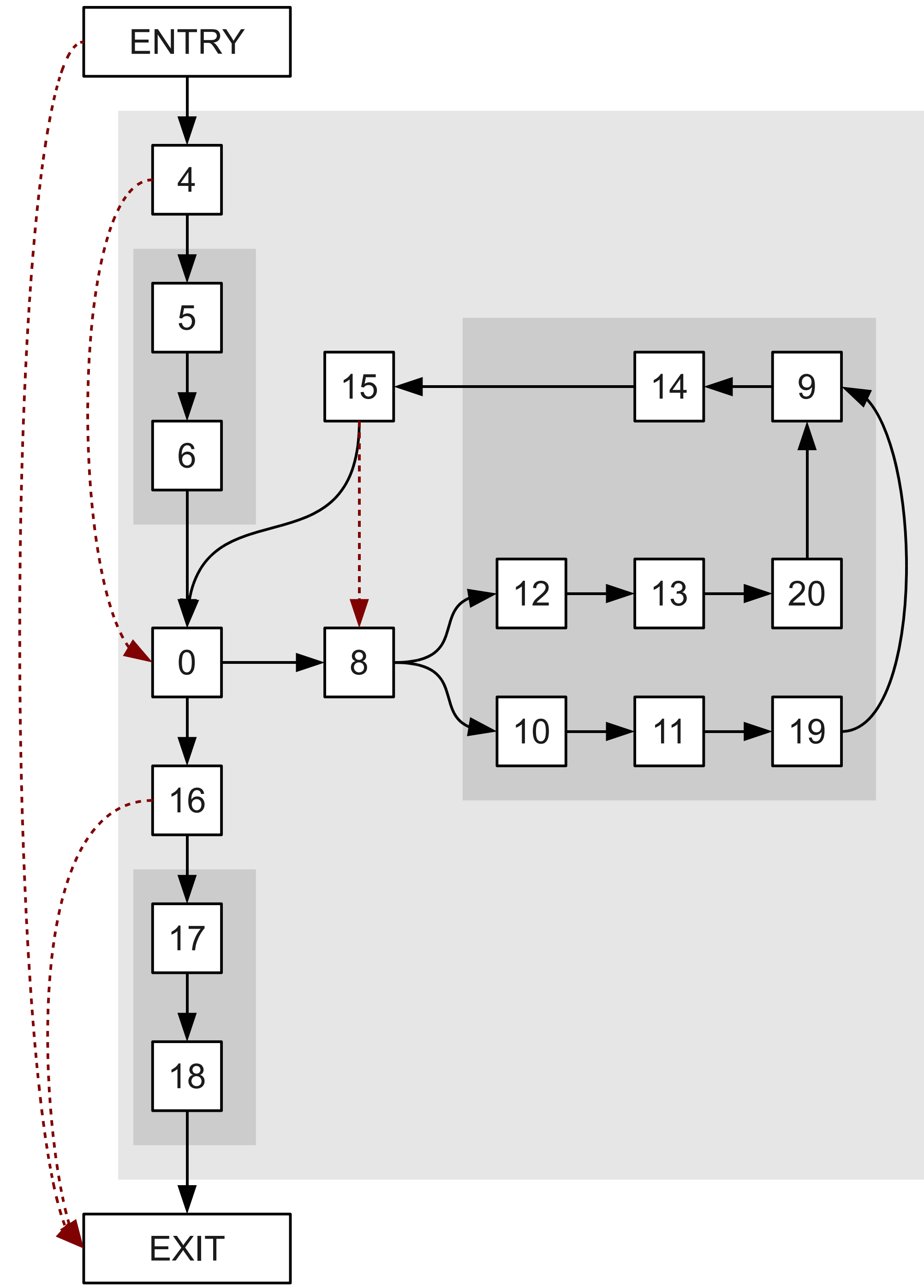
```
int g;
int buf[10];

void f(int p) {
    g = p;
    for(i=0;i<9;i++)
    {
        buf[i] = 0;
        ...
    }
}

int main() {
    f(1);
    f(2);
}
```

does not access g and p

f is analyzed twice even with localization

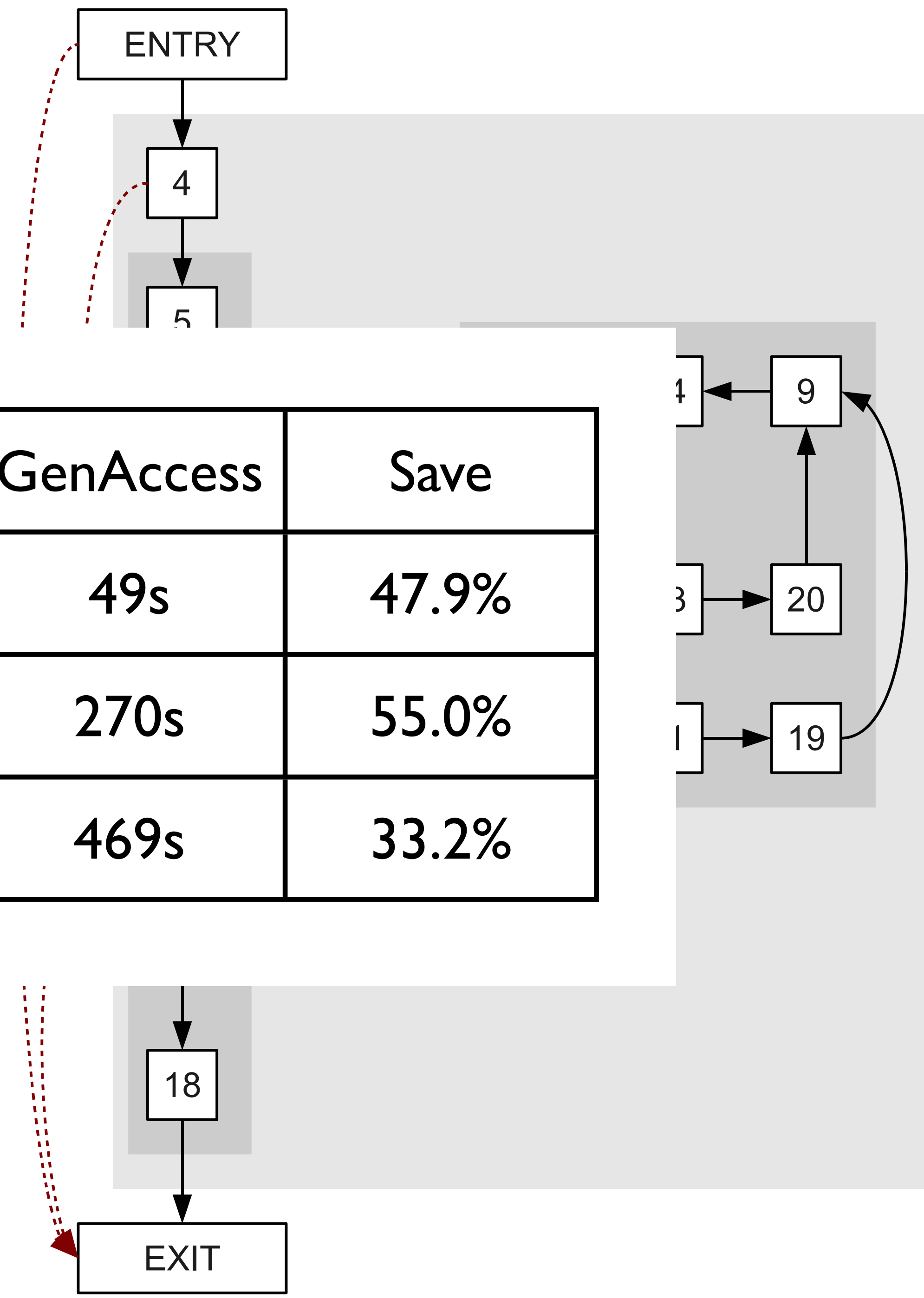
Selecting Localization Points

- The minimum size of blocks to be selected is parameterized.
- Blocks with one entry/exit point are selected.
- Blocks are selected recursively.



Selecting Localization Points

- The minimum size of bl
  pa
- Bl
  ex
- Bl
  recursively.



Pgm	LOC	Access	GenAccess	Save
gzip	7,327	94s	49s	47.9%
twolf	19,700	600s	270s	55.0%
bash	105,174	702s	469s	33.2%

Conclusion

- We need a more aggressive localization than reachability-based one.
  - Accessed-based localization passes only the memory parts possibly accessed by the called procedure bodies.
- Our localization can be effectively used for smaller entities than procedures.