Access Analysis-Based Tight Localization of Abstract Memories

Hakjoo Oh[1], Lucas Brutschy[2], and Kwangkeun Yi[1]

[1] Seoul National University, Korea
[2] RWTH Aachen University, Germany

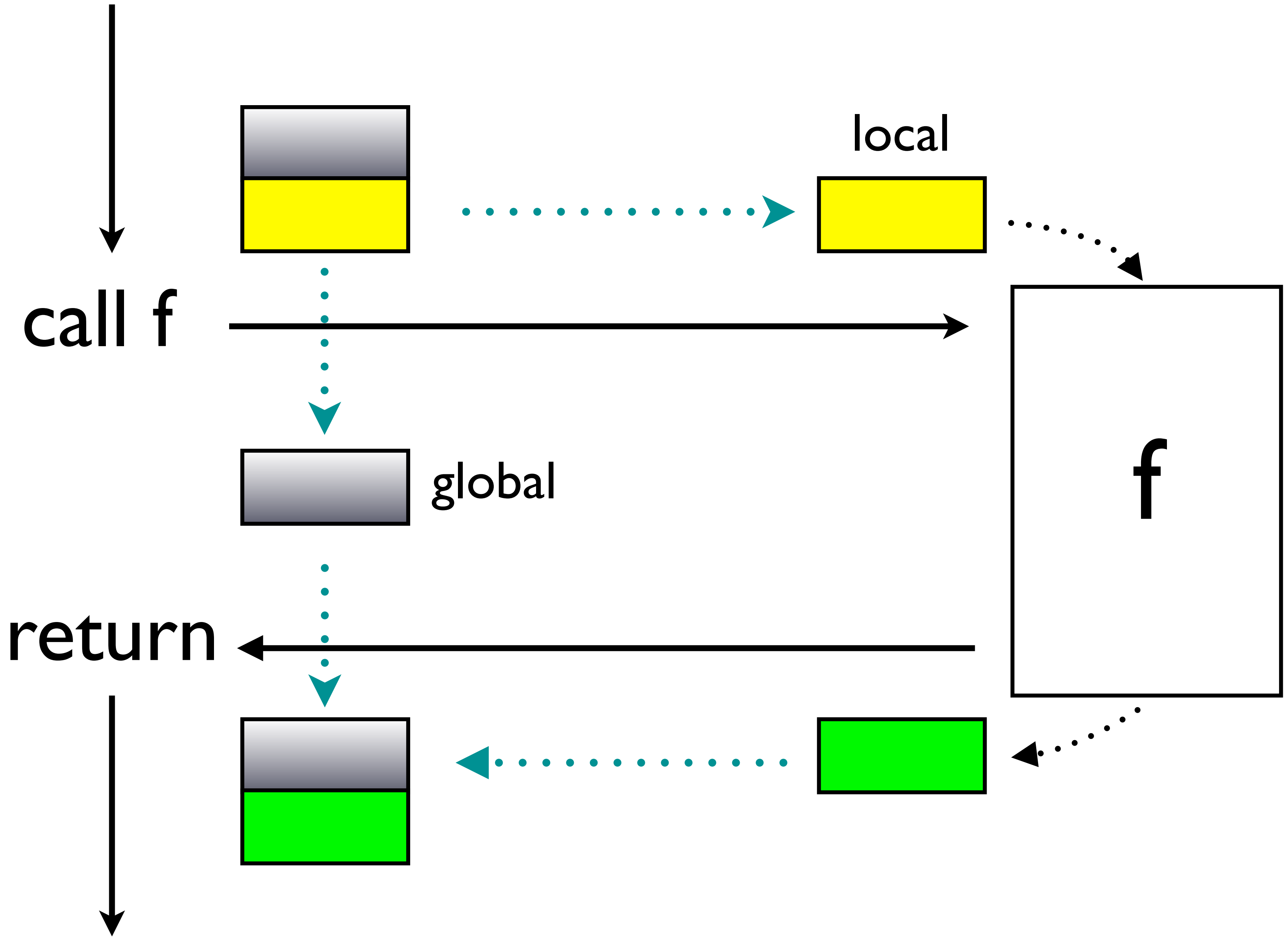
VMCAI 2011 @ Austin, Texas, USA

Localization

“framing”
“abstract garbage collection”

Key to scalability

Memory Localization



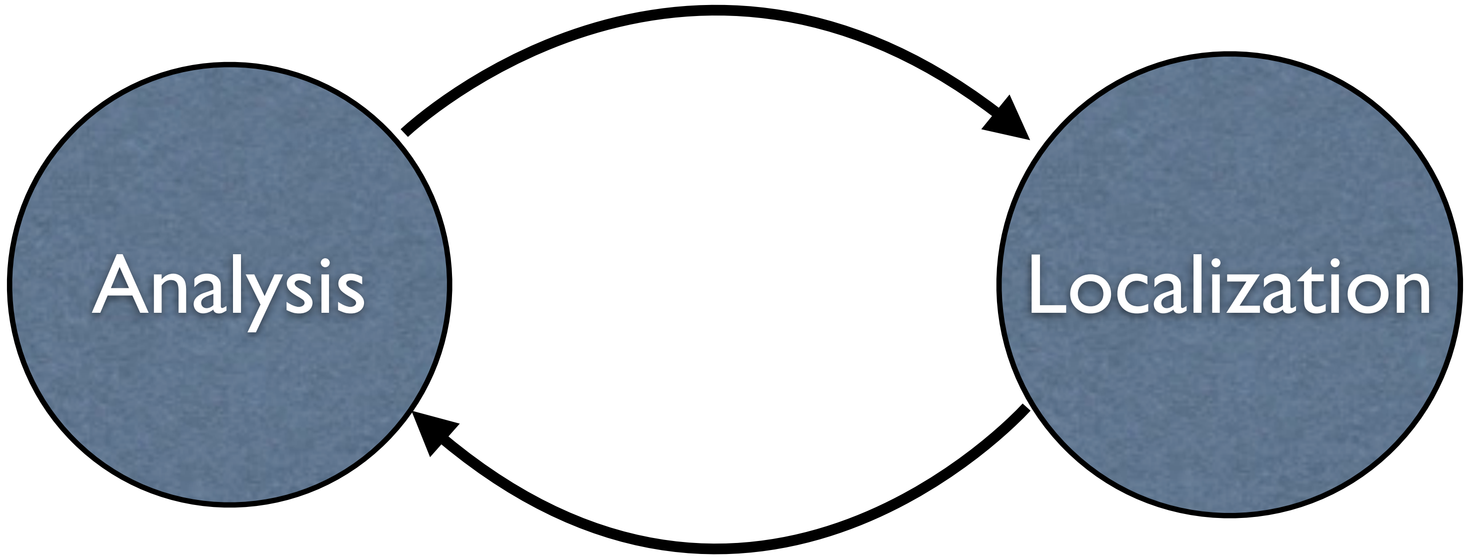
Benefits of Localization

```
int g;

int f() {...}

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

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

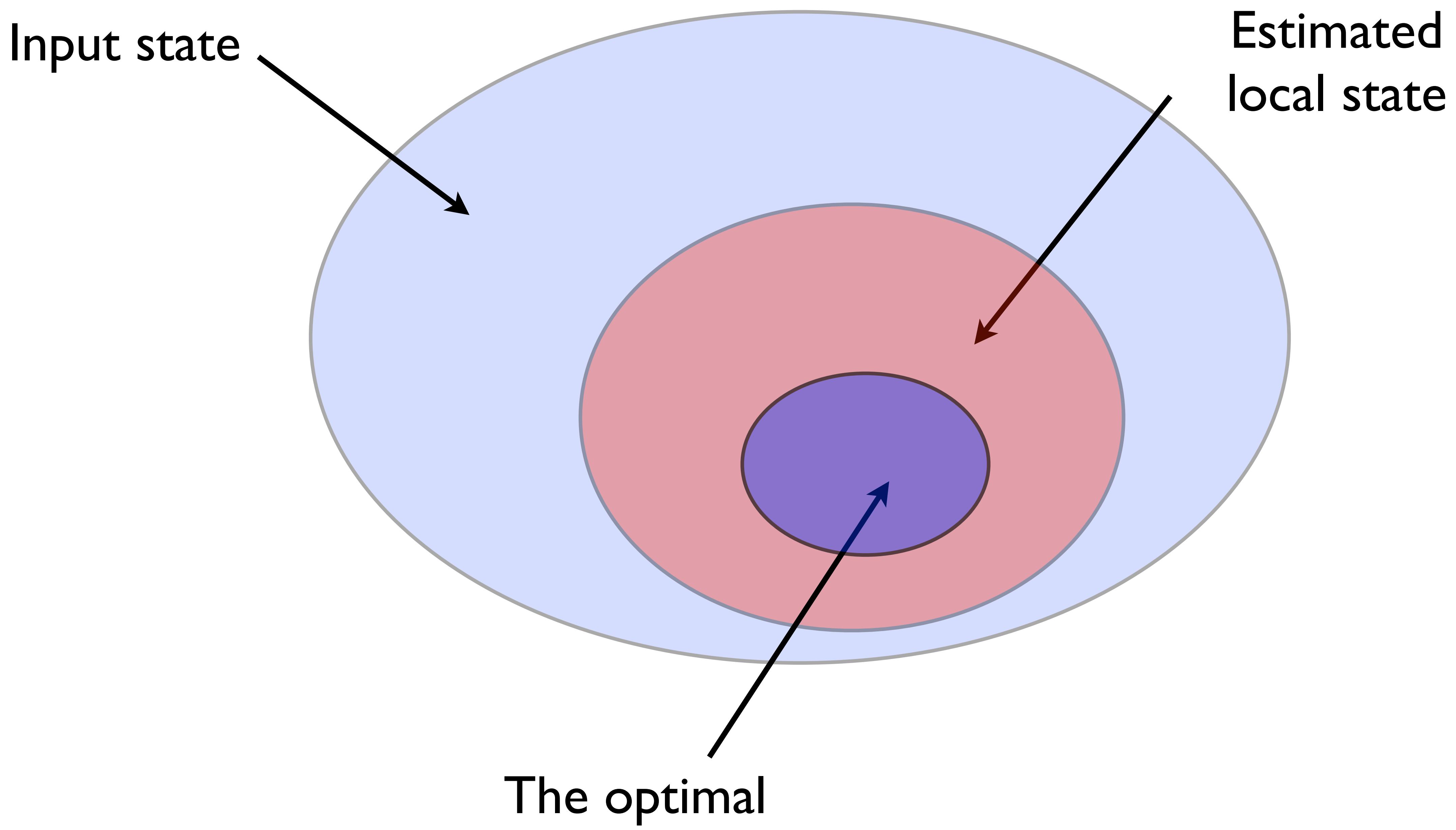
f does not access g

A Catch-22 Situation

The optimal localization is impossible

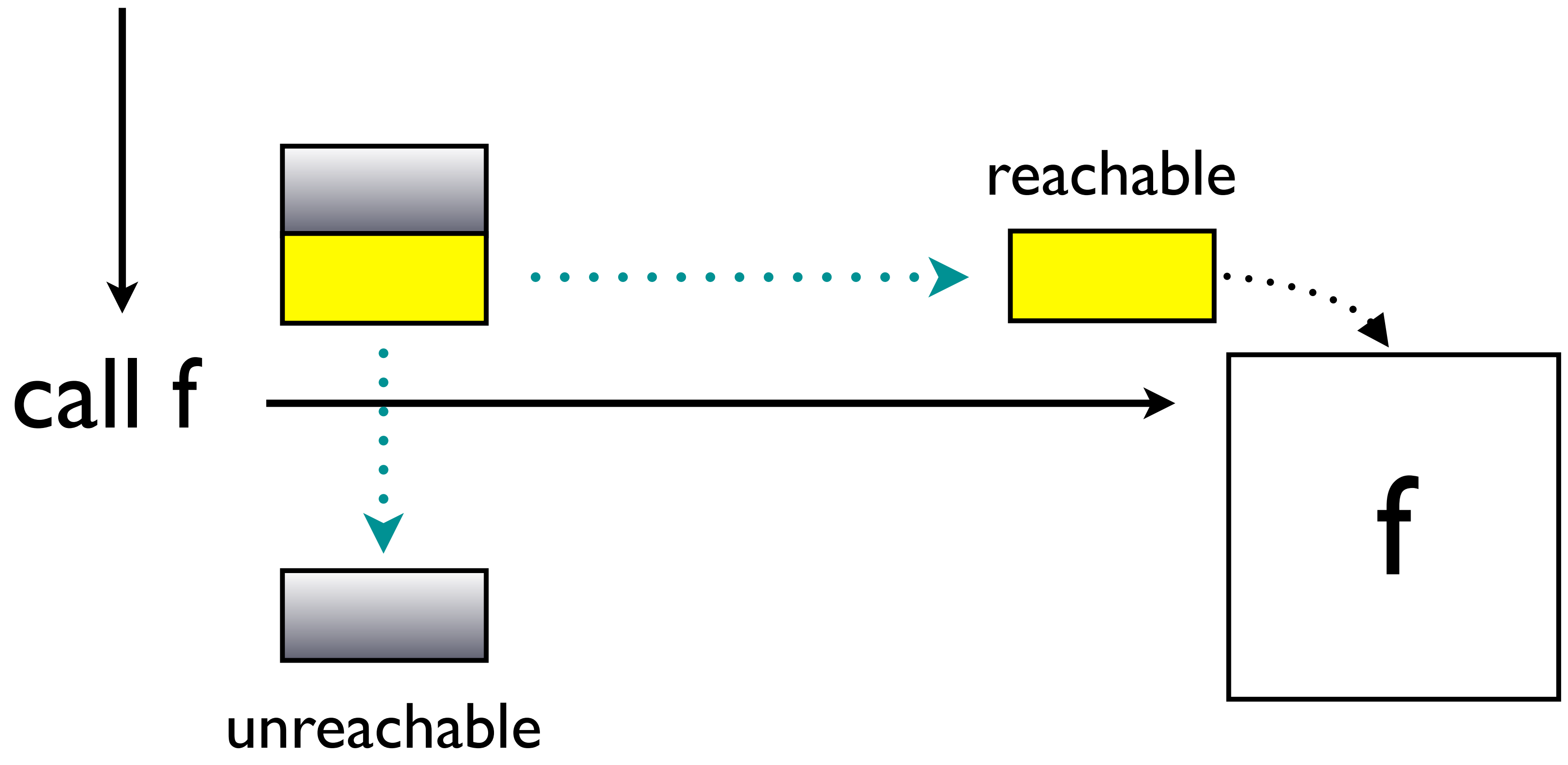


Need Approximation



Reachability-based Localization

- Remove the unreachable from params and globals



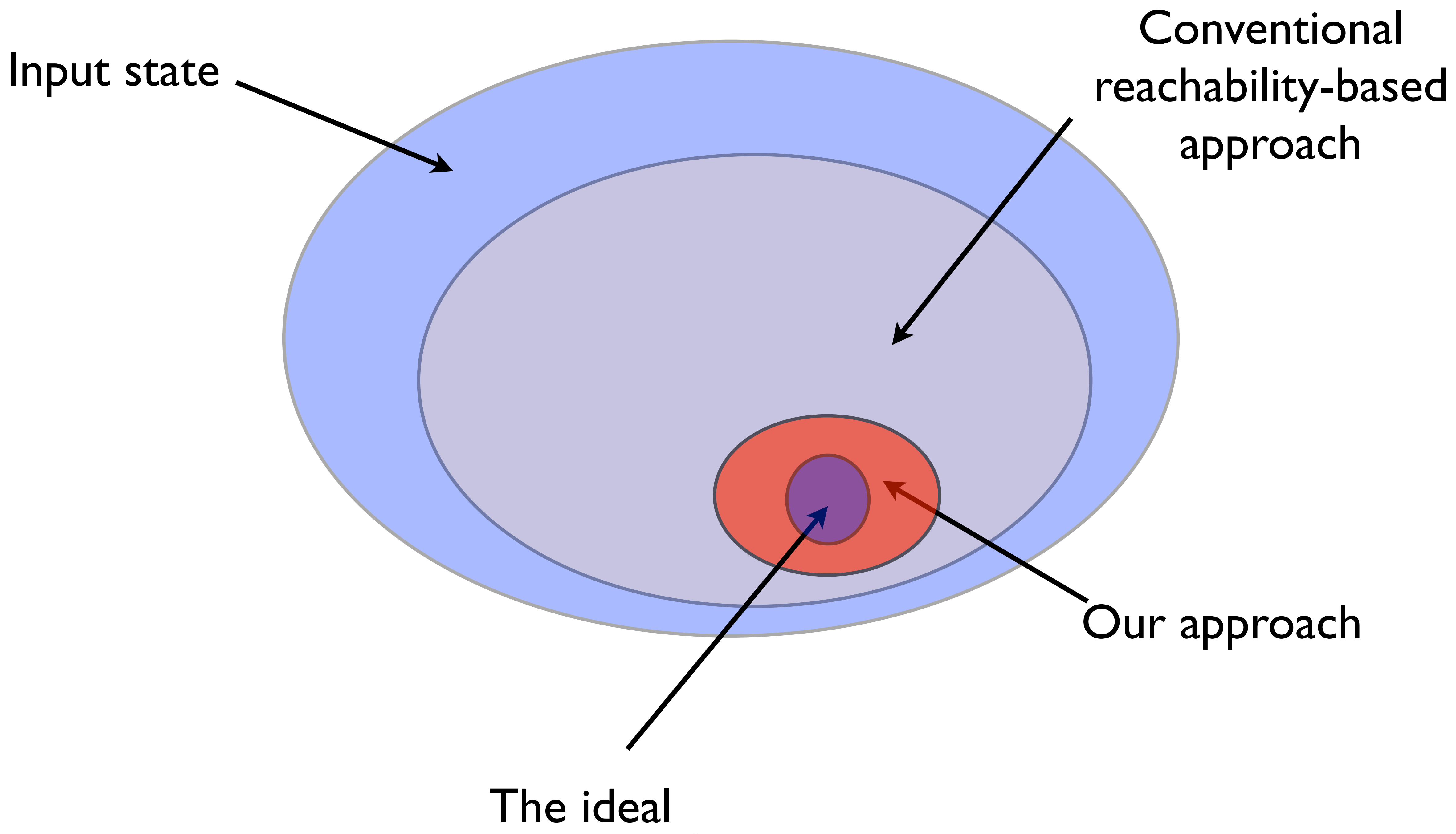
Key Observation

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%)

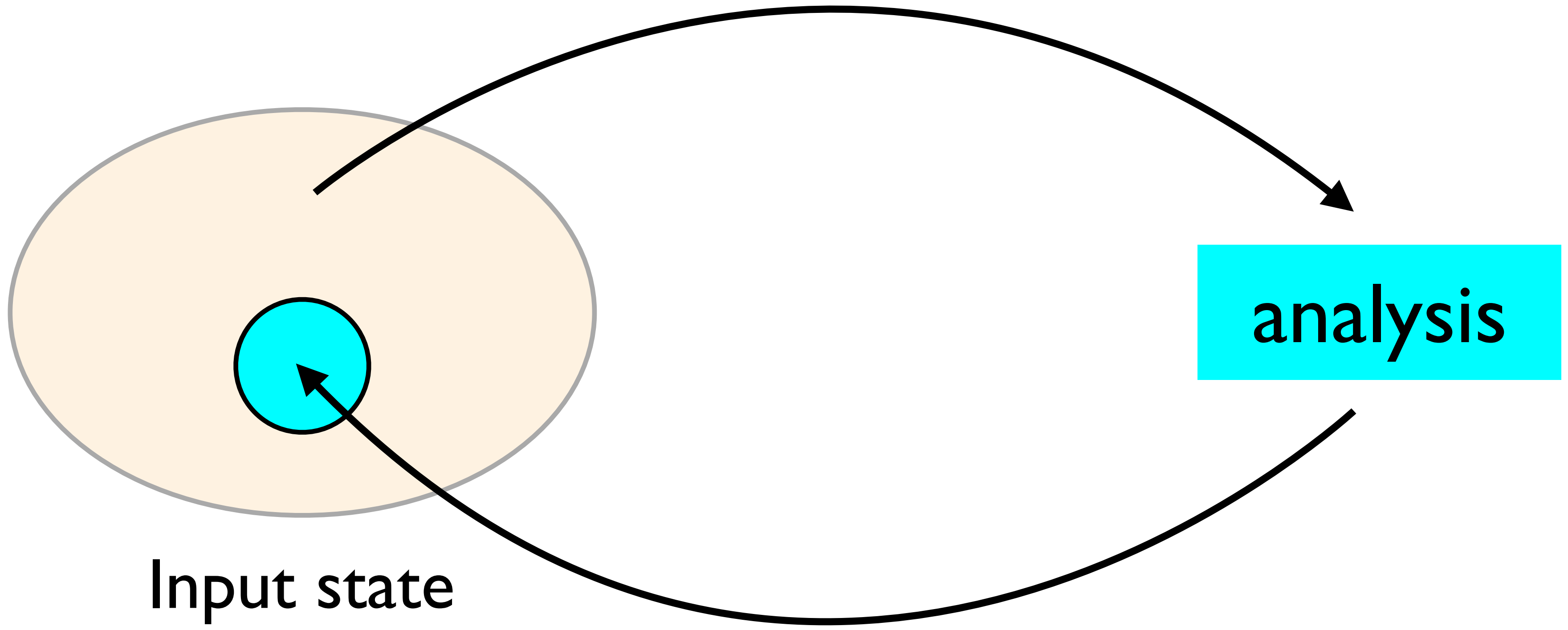
average : 4%

Goal


Input state
Conventional reachability-based approach
Our approach
The ideal

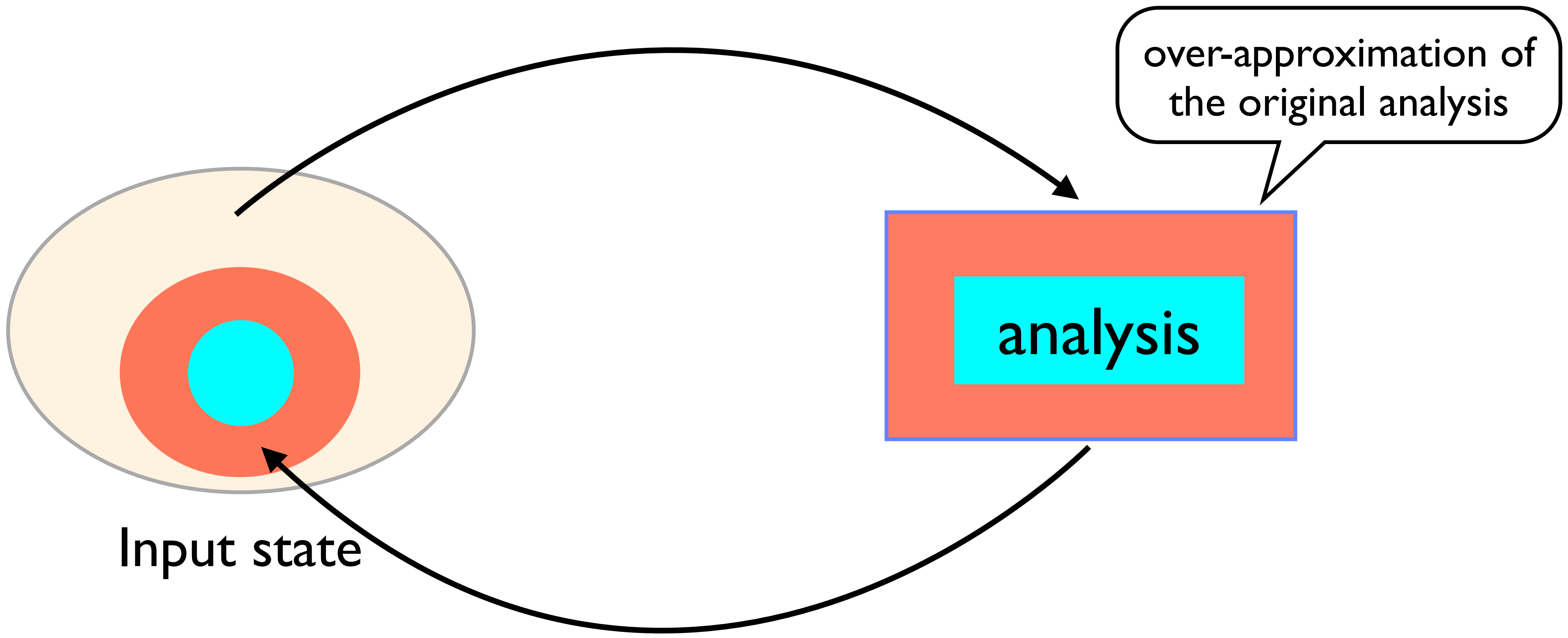
Basic Idea

Analyze the procedure and
observe which resources are used



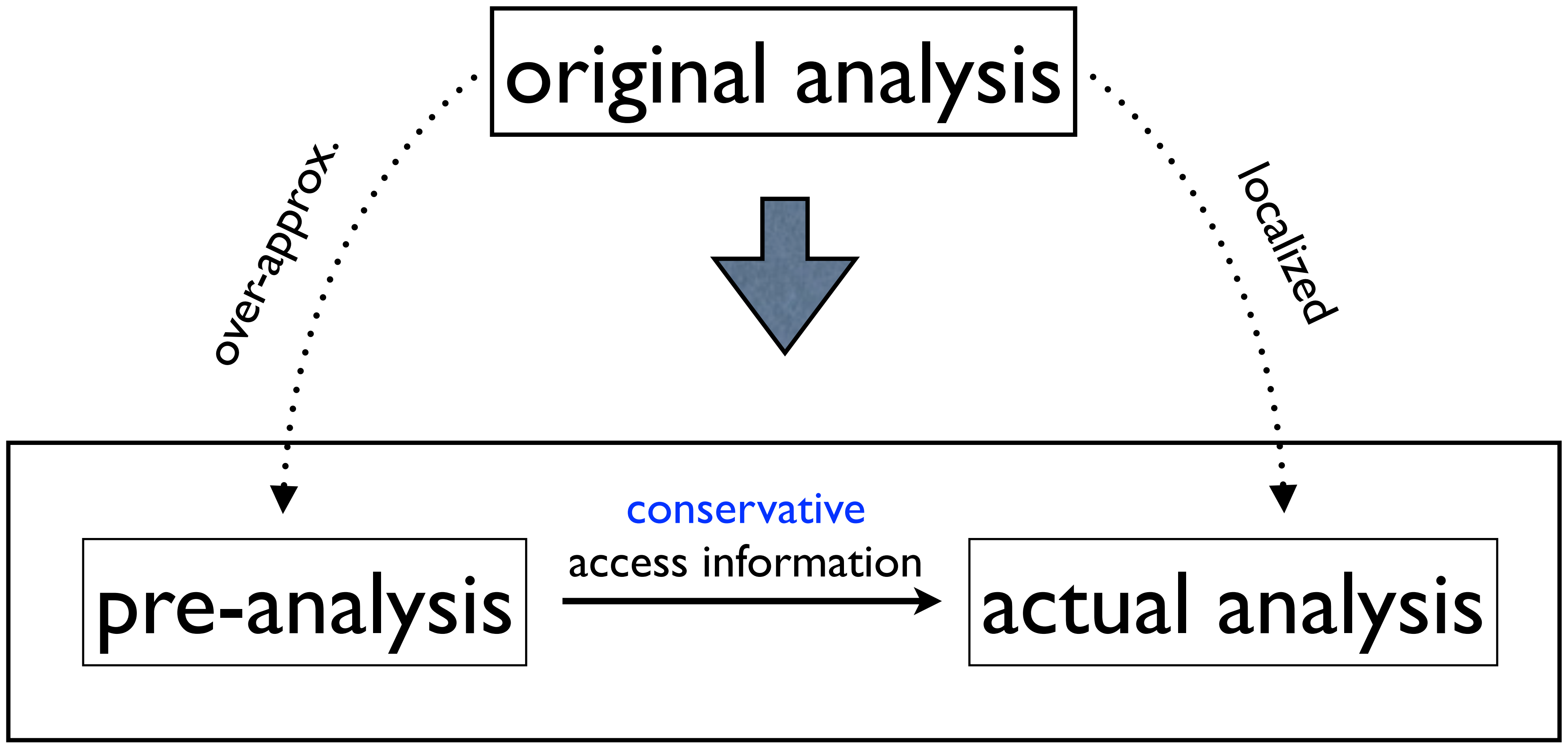
Basic Idea

Over-approximated access-info
from an over-approximated analysis



Access-based Localization

- Staging the analysis into two phases



Deriving a Pre-analysis

- Original analysis: abstract interpretation

$$(\hat{D}, \hat{F}) \qquad \mathsf{lfp}\hat{F} \quad (\hat{F} : \hat{D} \to \hat{D})$$

- Goal: finding an over-approximation

$$\mathsf{lfp}\hat{F} \sqsubseteq \text{⭐}$$

Deriving a Pre-analysis

- Pre-analysis is a further abstract interpretation
  - define $(\hat{D}_p, \hat{F}_p)$ such that

$$\hat{D} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \hat{D}_p$$

$$\alpha \circ \hat{F} \sqsubseteq \hat{F}_p \circ \alpha \quad (\hat{F}_p : \hat{D}_p \to \hat{D}_p)$$

$$\boxed{\mathsf{lfp}(\hat{F}) \sqsubseteq \gamma(\mathsf{lfp}(\hat{F}_p))}$$

Our Pre-analysis

$$\begin{tikzcd}
PgmPt \to \hat{Mem} & & \hat{Mem} \\
\hat{D} \arrow[u, phantom, "\rotatebox{90}{=}"] \arrow[rr, shift right, "\alpha=\lambda d.\bigsqcup_{p\in PgmPt} d(p)"'] & & \hat{D}_p \arrow[ll, shift right, "\gamma=\lambda m.\lambda n.m"'] \arrow[u, phantom, "\rotatebox{90}{=}"]
\end{tikzcd}$$

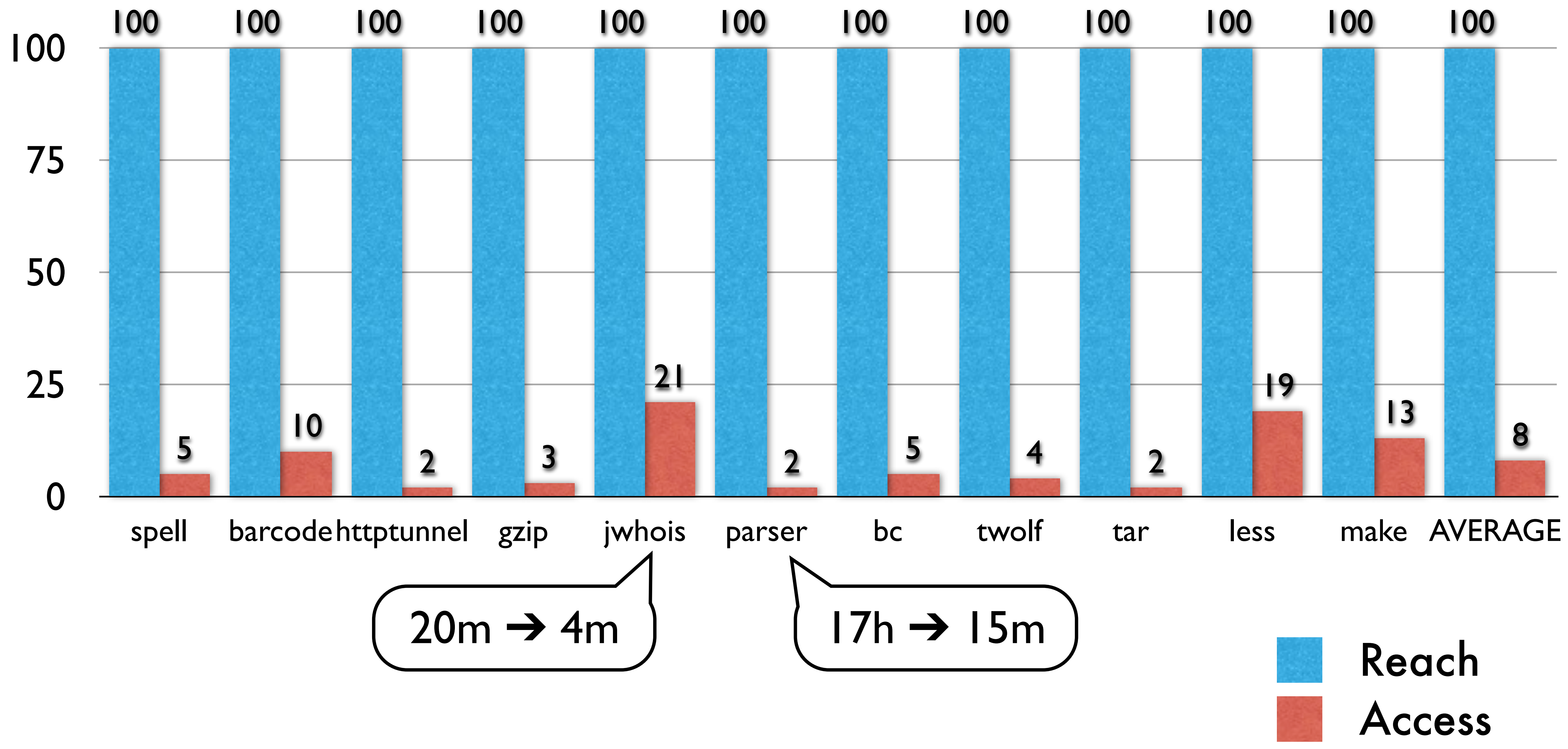
ignore statement orders

Experiments

- Interval-domain-based abstract interpreter
  - Baseline:no localization
  - Reach: Baseline with reachability-based localization
  - Access: Baseline with access-based localization
- 15 GNU / SPEC 2000 benchmarks

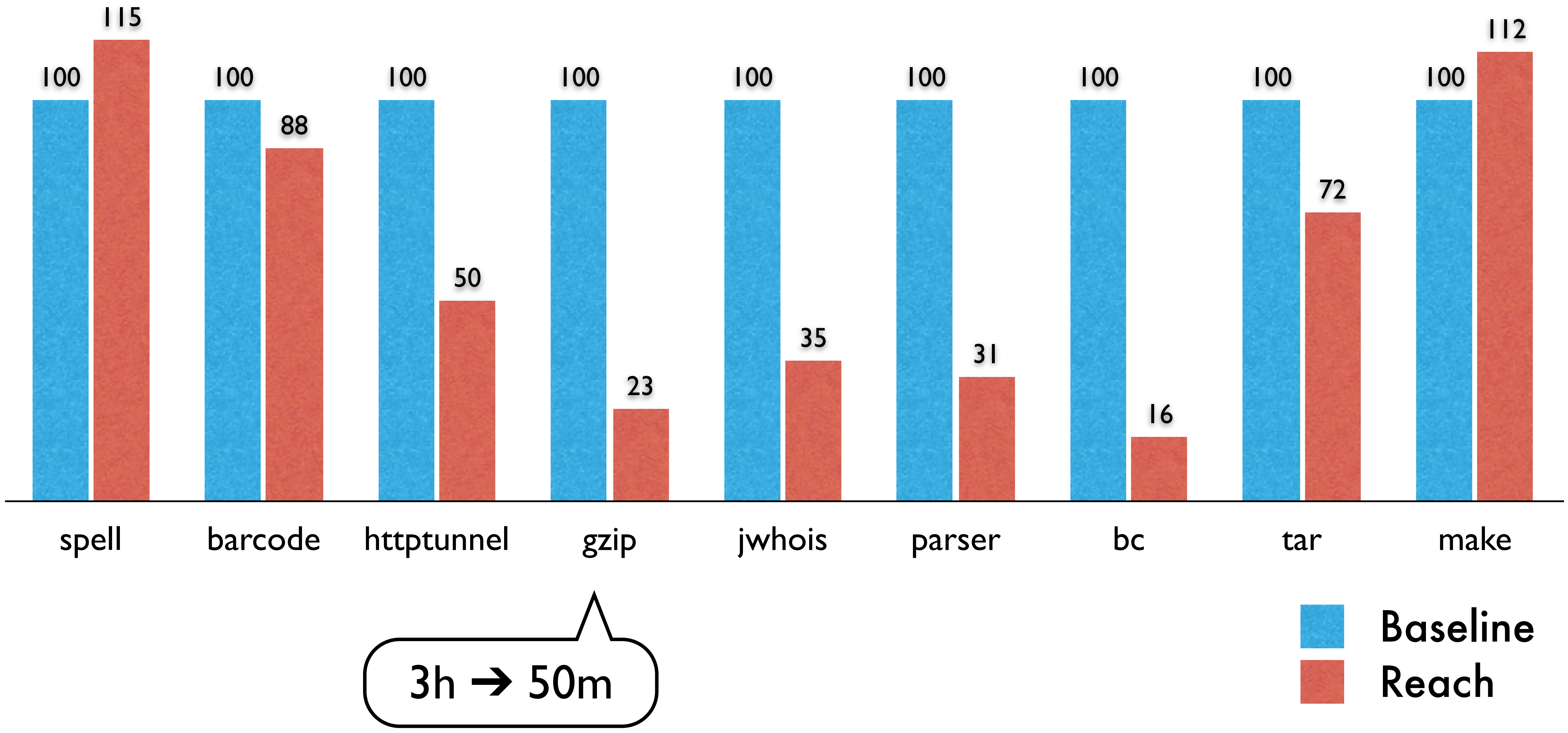
Reach vs. Access

78.5%-98.5% reduction
92.1% in average



Baseline vs. Reach

~6x speed-up



Pre-analysis Overhead

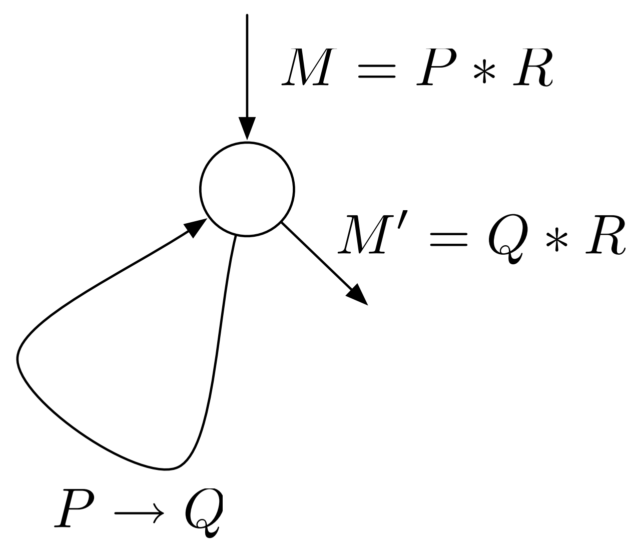
- Small overhead compared to the total analysis time
  - 0.1 ~ 8%

Program	LOC	Time		Overhead
		Total	Pre	
gzip	7,327	95s	1.3s	1.4%
bc	13,093	730s	4.1s	0.6%
bash	105,174	2011s	20.2s	1.0%

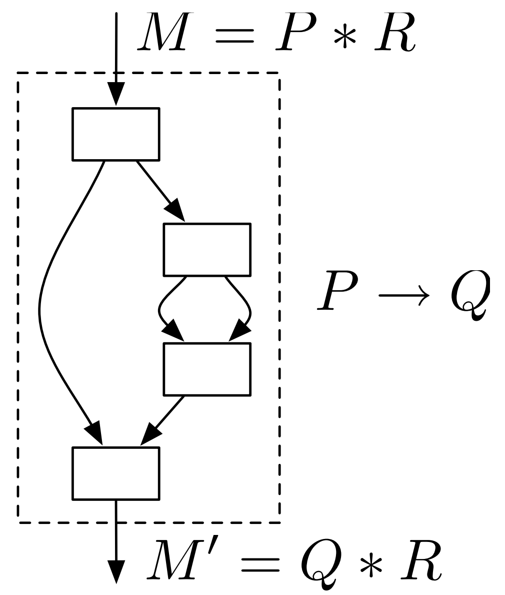
Block-level Localization

Access-based localization at any level

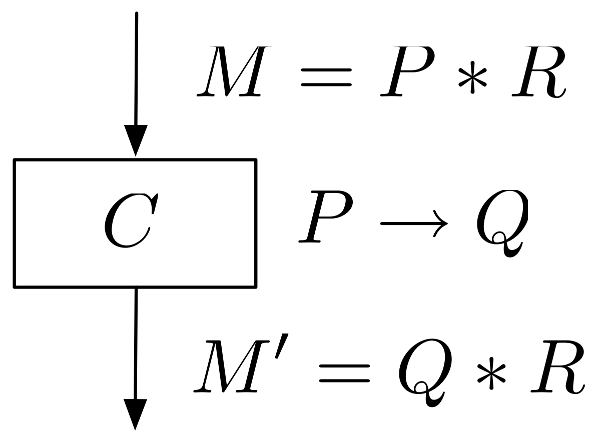
loops



branches

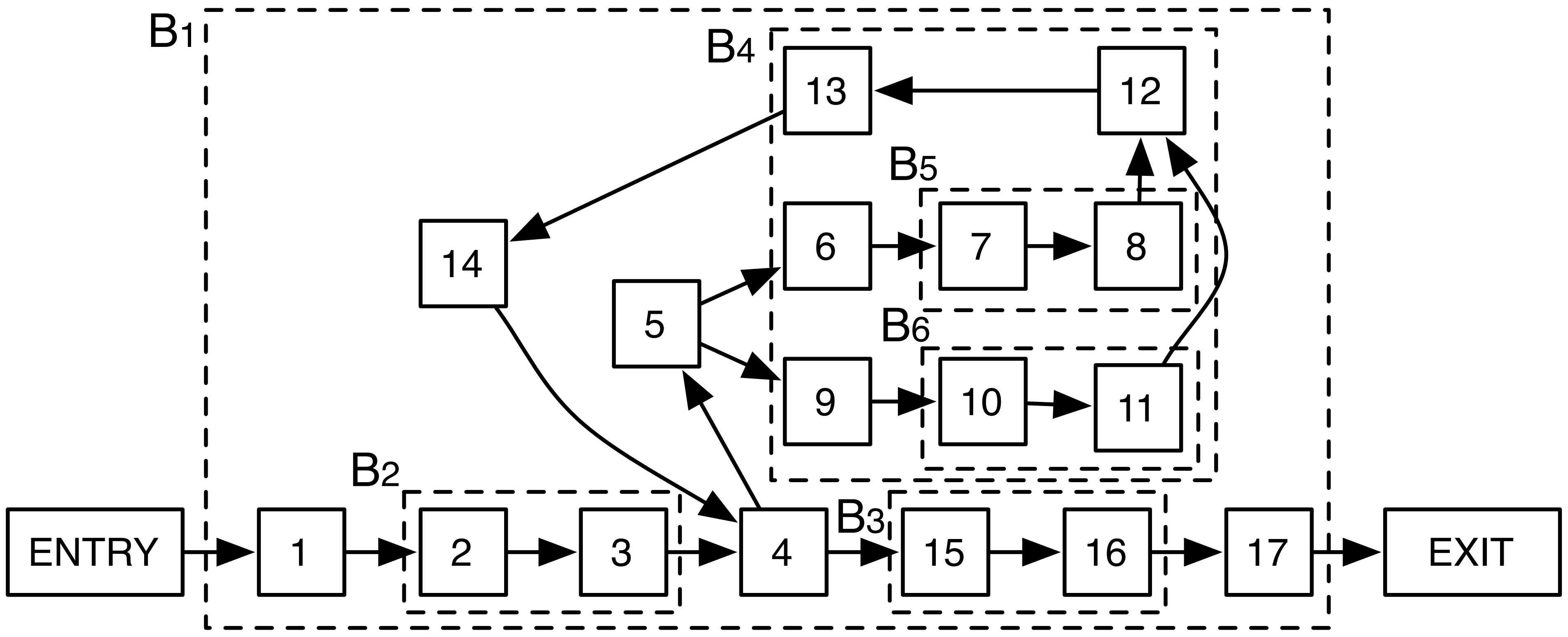


basic blocks



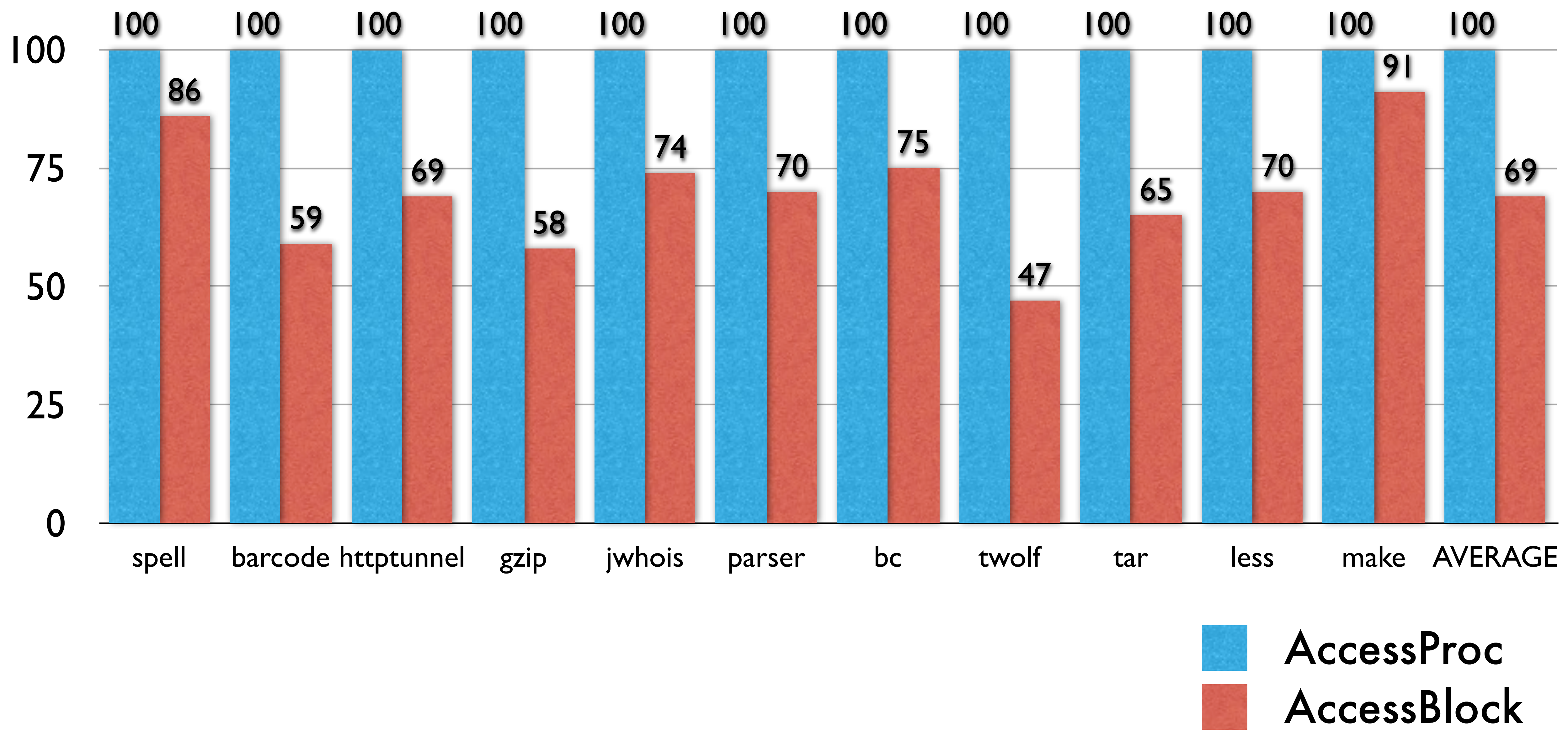
Block Selection Strategy

- In a nested way
- |block| ≧ *k*



Block-level Localization

On average 31% reduction in time (k=6)



Precision

- No precision loss
- Sometimes, even improved

f does not access g

```
int g;

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

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

g : [0,0] $\triangledown$ [1,1] = [0,+oo]

g : [0,+oo] vs. [1,1]

Conclusion

Reachability is too conservative

Access-based localization is a good alternative
“fast”
“extensible”

Thank you