Inlining for Size

Kyungwoo Lee
Ellis Hoag
Nathan Lanza

Meta

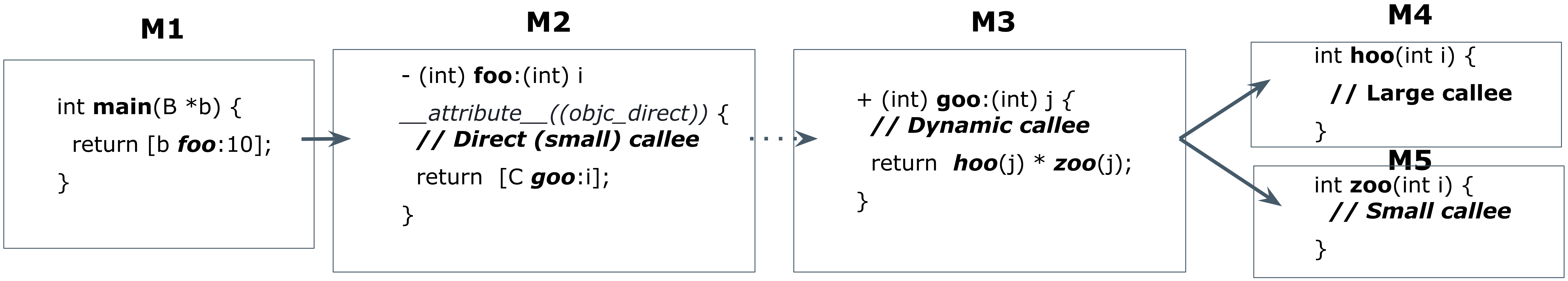
Mobile Apps Optimized for Size

- Mobile apps are mainly optimized for size (*-Oz*).
  - Demand more features on constrained devices.
  - Size and performance can be tuned with PGO [1].
- A (*Full*) link-time optimization (LTO) can minimize app size.
  - Practically, ThinLTO [2], a scalable LTO, is used for large apps.
- Inlining has been primarily considered a speed optimization.
  - Inlining is also critical for app size.
  - However, inlining for size with ThinLTO is suboptimal.

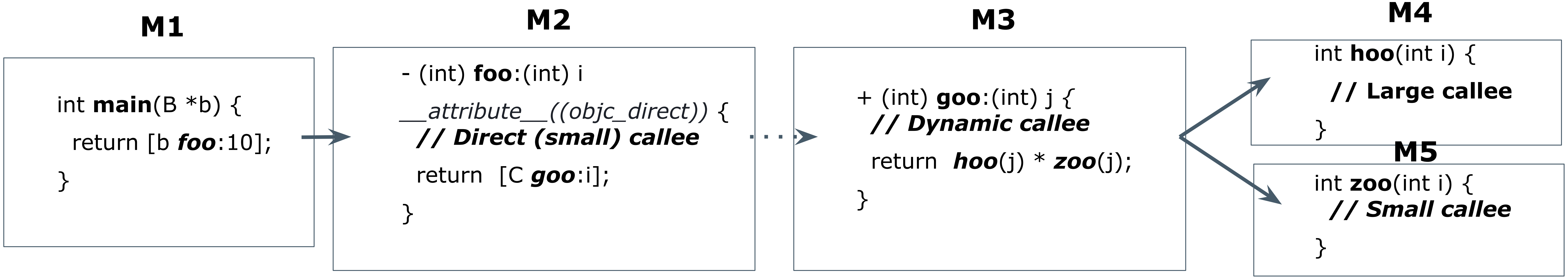
[1] Efficient Profile-Guided Size Optimization for Native Mobile Applications, Kyungwoo Lee, Ellis Hoag, and Nikolai Tillman. 2022.
https://dl.acm.org/doi/10.1145/3497776.3517764
[2] ThinLTO: Scalable and Incremental LTO, Teresa Johnson, Mehdi Amini, and Xinliang David Li. 2017.
https://dl.acm.org/doi/10.5555/3049832.3049845

Meta

NoLTO - No inlining occurs across modules

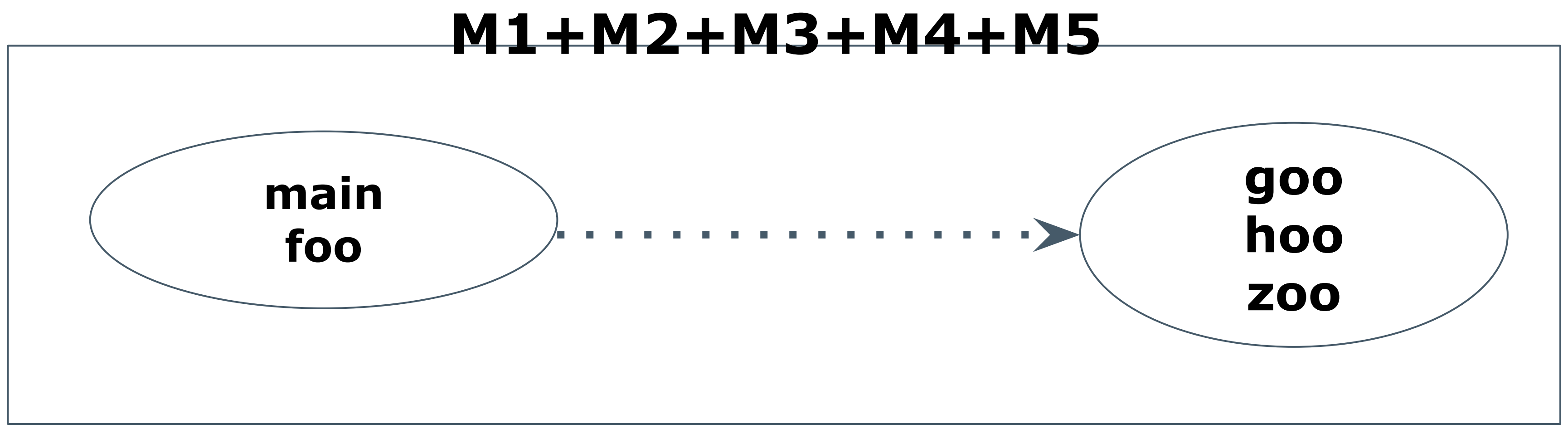


NoLTO - No inlining occurs across modules



(*Full*)LTO - All (direct) callees are inlined

After Inlining



Meta

(*Full*)LTO - Size-Cost Model

- The baseline inliner aggressively inlines a local function called from a ***single*** call-site.
  - Or, only tiny functions (<= *OptMinSizeThreshold*(5)) become inline candidates with -Oz.
- Our size-inliner uses the simple cost model, to find a candidate where $C_{before} > C_{after}$

$$C_{before} = C_{callee} + N * C_{call}$$

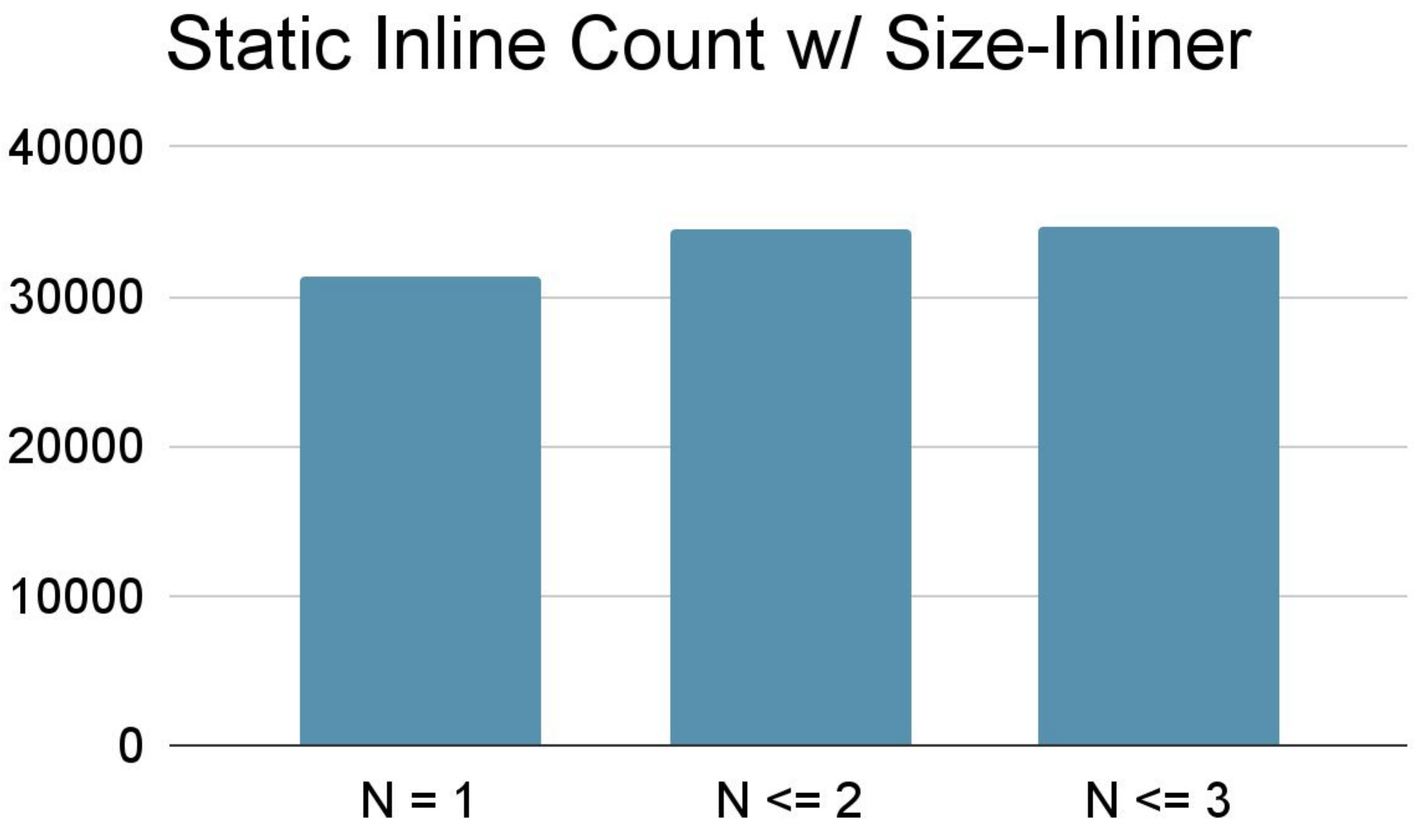
$$C_{after} = N * C_{callee}$$

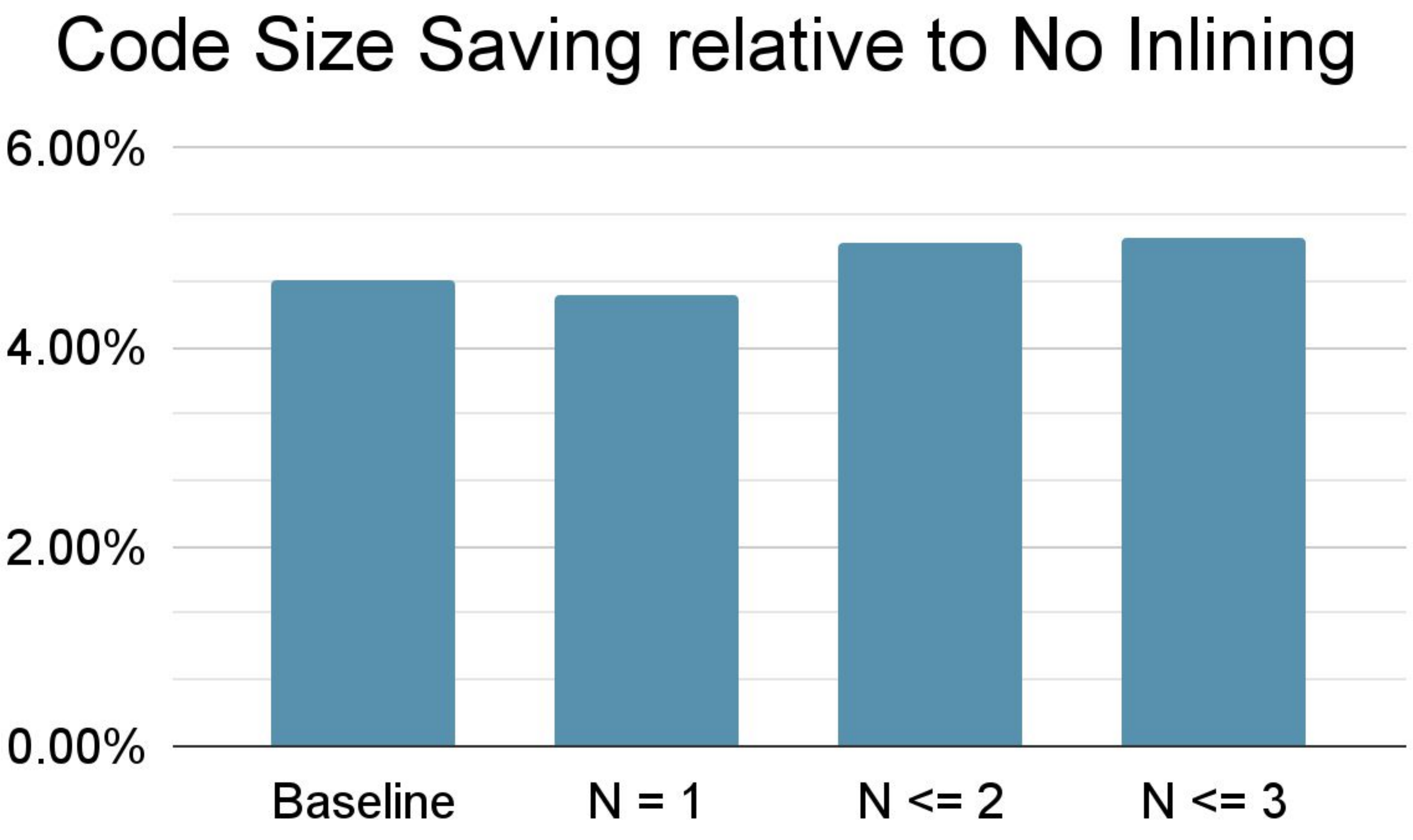
where N is # of call sites, C_{callee} is the callee size, and C_{call} is the call overhead

Meta

(*Full*)LTO - Limit Study

- The baseline inliner (-Oz) is close to our size inliner with # of Call Site, N = 1, which is the majority of size-win.
  - Our size-inliner can improve the size win further by 0.4% w/ up to # of Call Site, N = 3.





Meta

ThinLTO - Only small (eligible) callee inlined

Before Inlining

M1

```
int main(B *b) {
  return [b foo:10];
}
```

M2

```
- (int) foo:(int) i
__attribute__((objc_direct)) {
  // Direct (small) callee
  return  [C goo:i];
}
```

M3

```
+ (int) goo:(int) j {
  // Dynamic callee
  return  hoo(j) * zoo(j);
}
```

hoo is **large** and ThinLTO cannot determine its unique call-site.

M4

```
int hoo(int i) {
  // Large callee
}
```

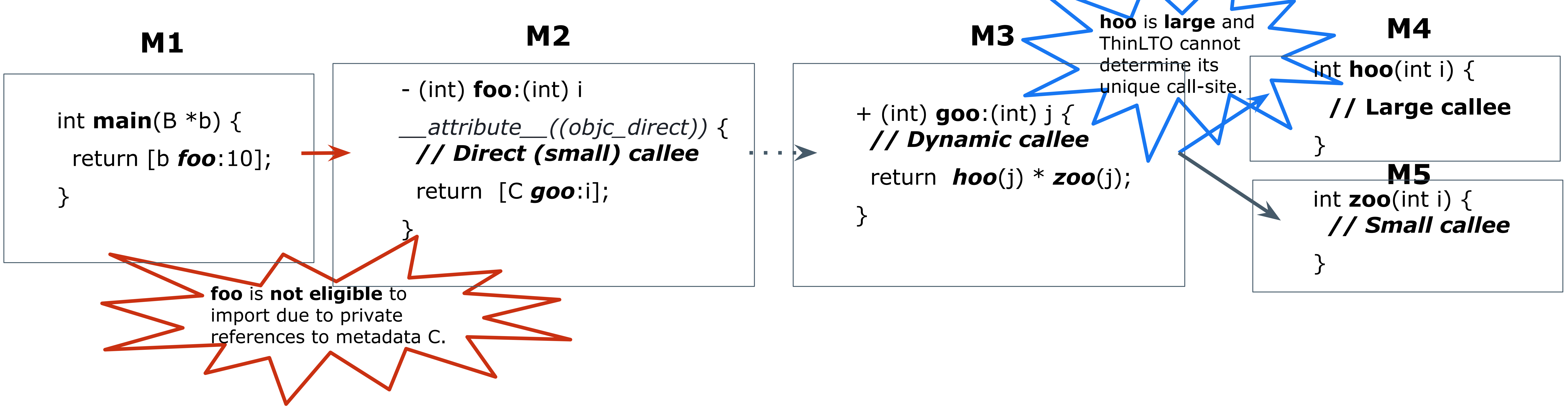
M5

```
int zoo(int i) {
  // Small callee
}
```

Meta

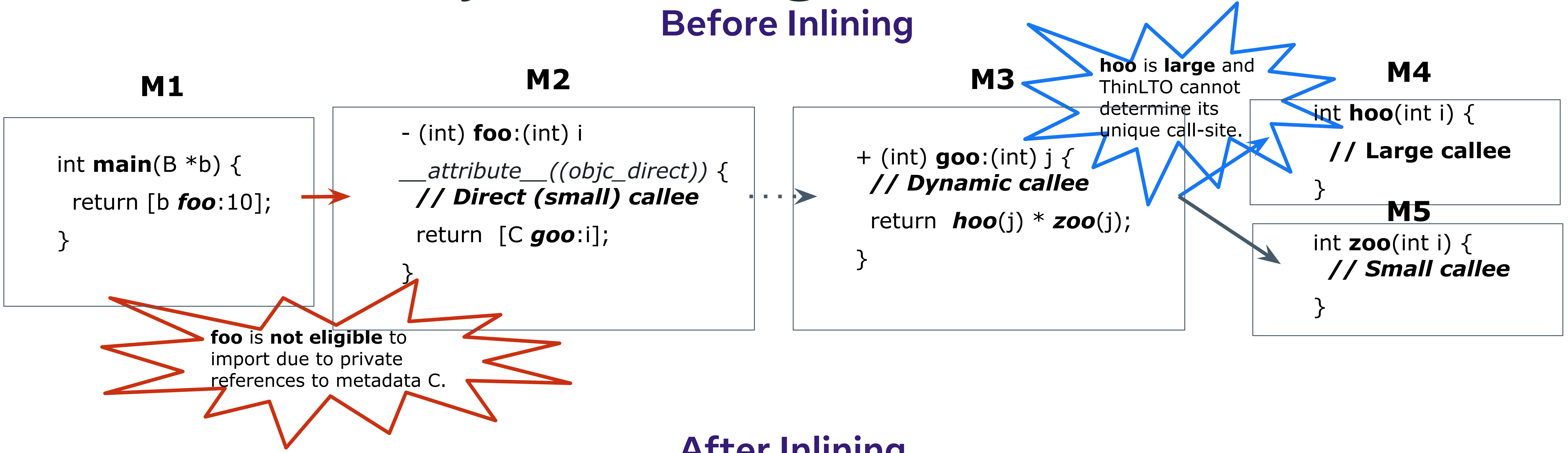
ThinLTO - Callee is not eligible to import.

Before Inlining

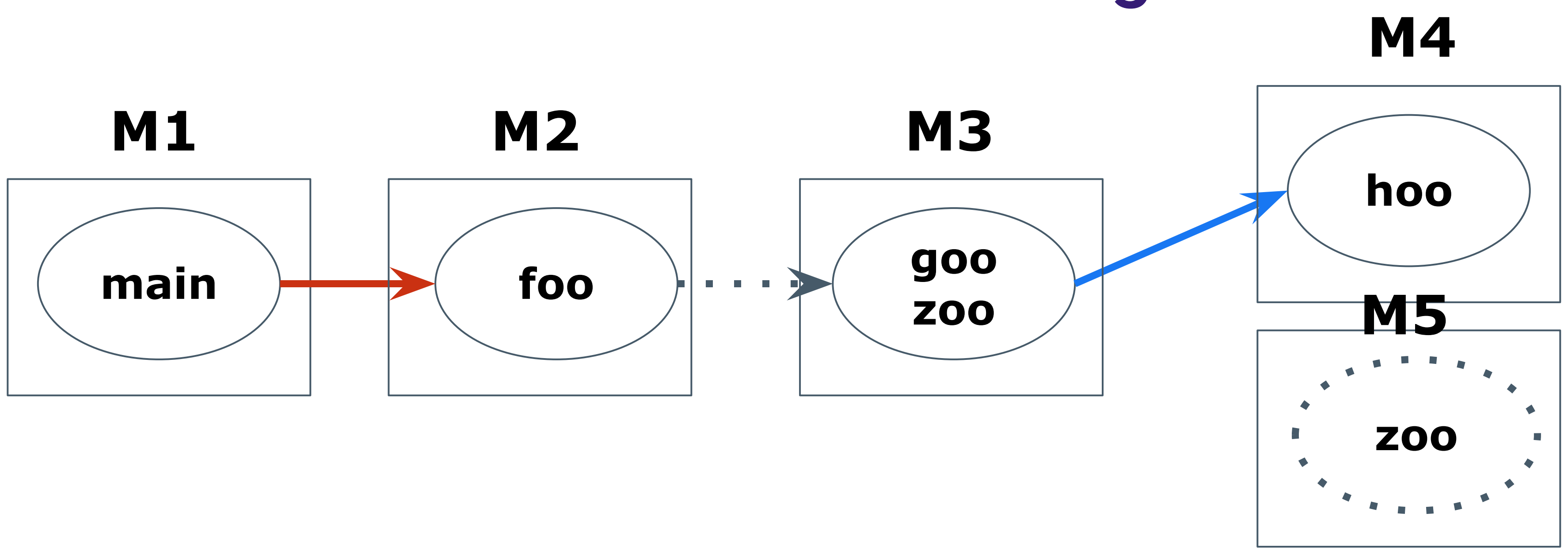


ThinLTO - Only small (eligible) callee inlined

Before Inlining



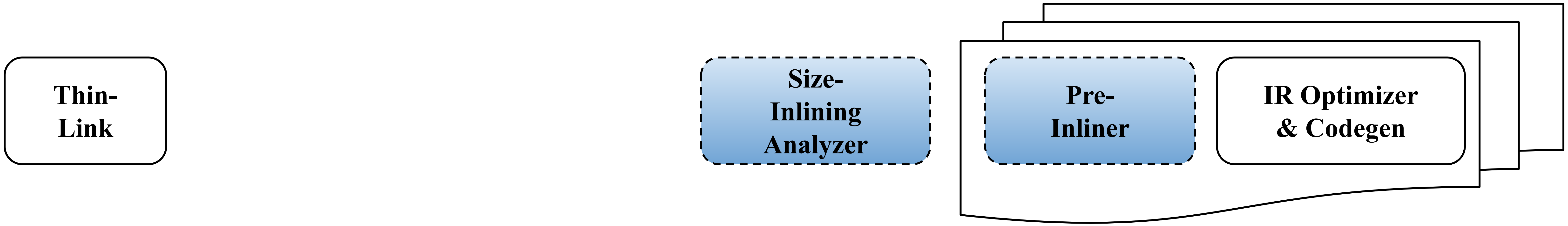
After Inlining



Meta

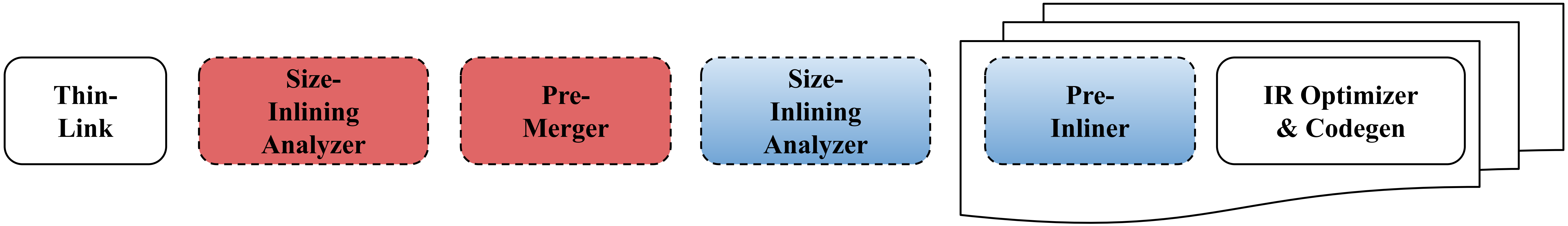
Our Contribution

Our Contribution



- Size-Inlining Analyzer + Pre-Inliner
  - Extend bitcode summary that reflects call-site counts (within each module)
  - Propagate summaries, and determine inline candidates
  - Force to import & inline those candidates to realize the size win ahead.

Our Contribution

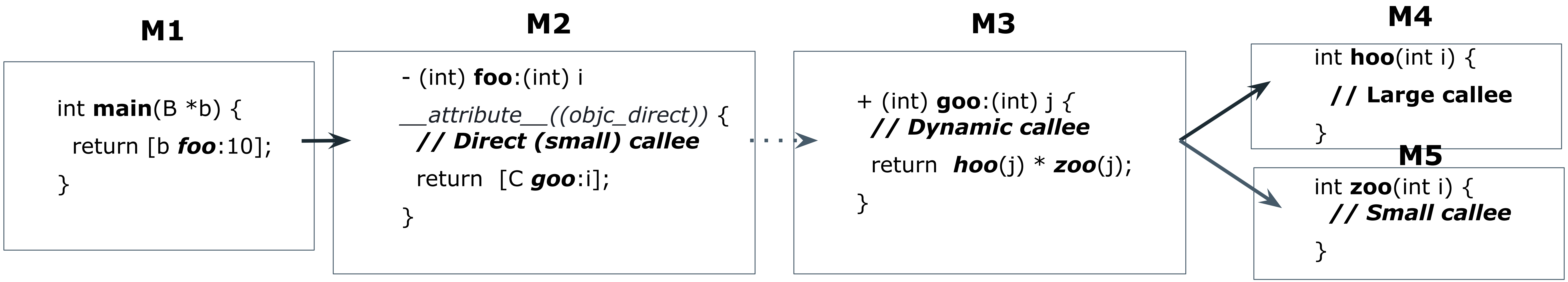


- Size-Inlining Analyzer + Pre-Inliner
  - Extend bitcode summary that reflects call-site counts (within each module)
  - Propagate summaries, and determine inline candidates
  - Force to import & inline those candidates to realize the size win ahead.

- Size-Inlining Analyzer + Pre-Merger
  - Find the inline candidates ***that are not eligible to import***.
  - Merge their parent bitcode modules to remove inline restrictions.

ThinLTO /w Size Inliner

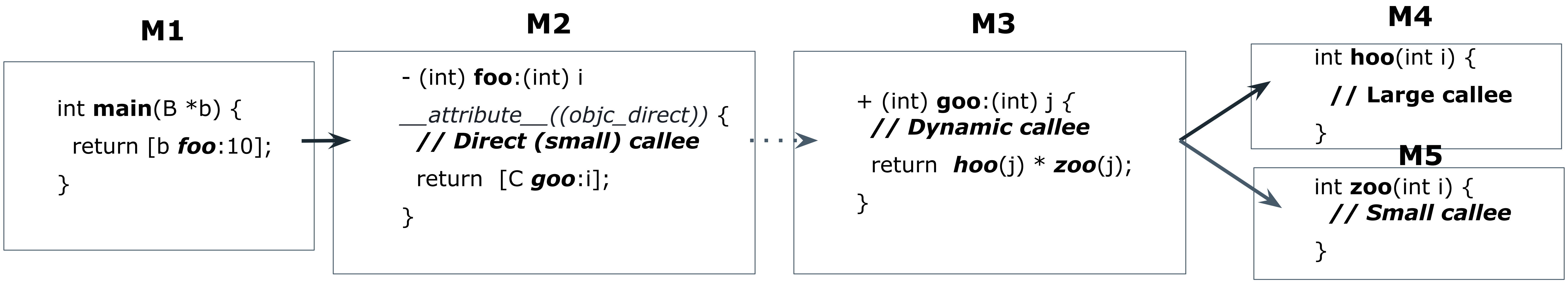
Before Inlining



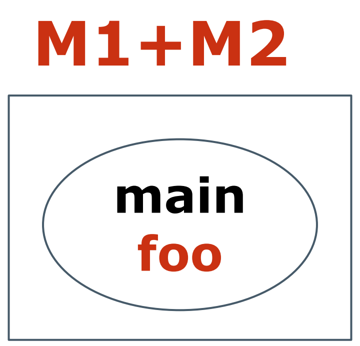
Meta

ThinLTO /w Size Inliner

Before Inlining

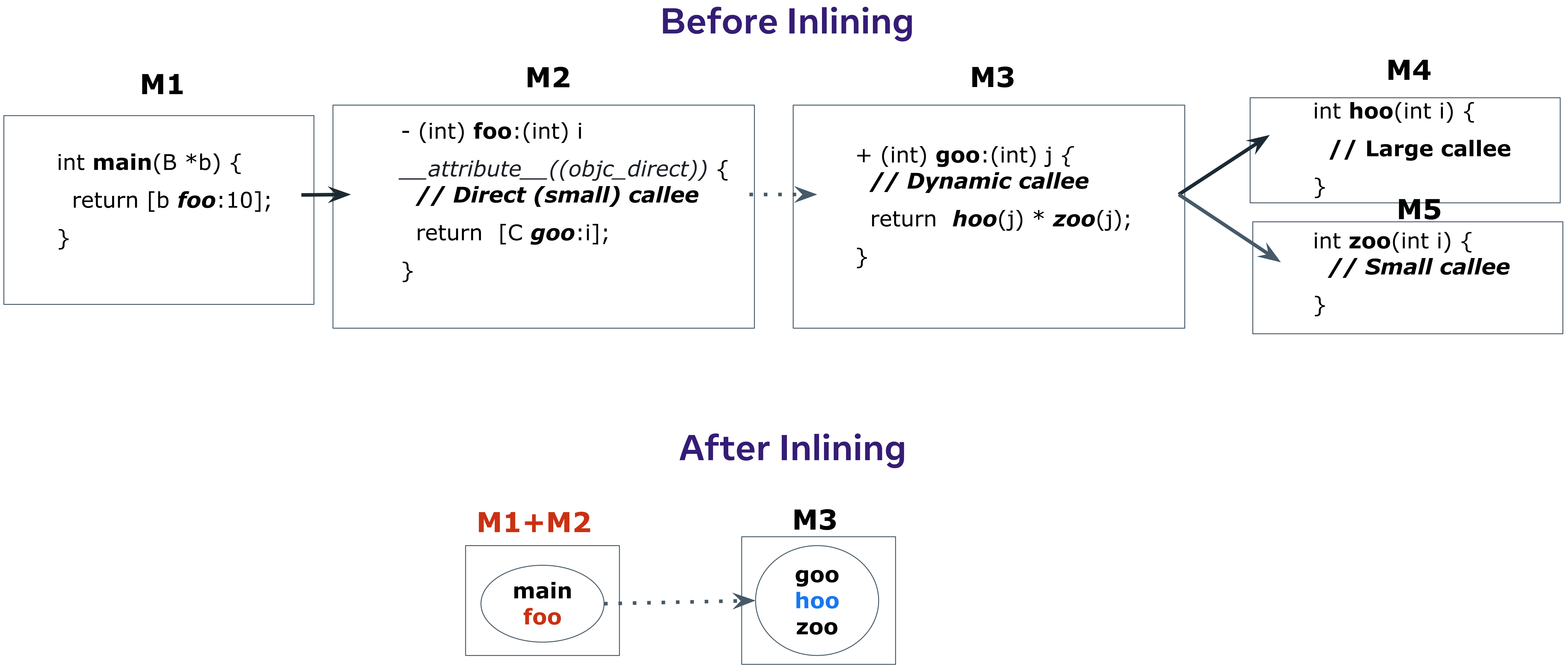


After Inlining



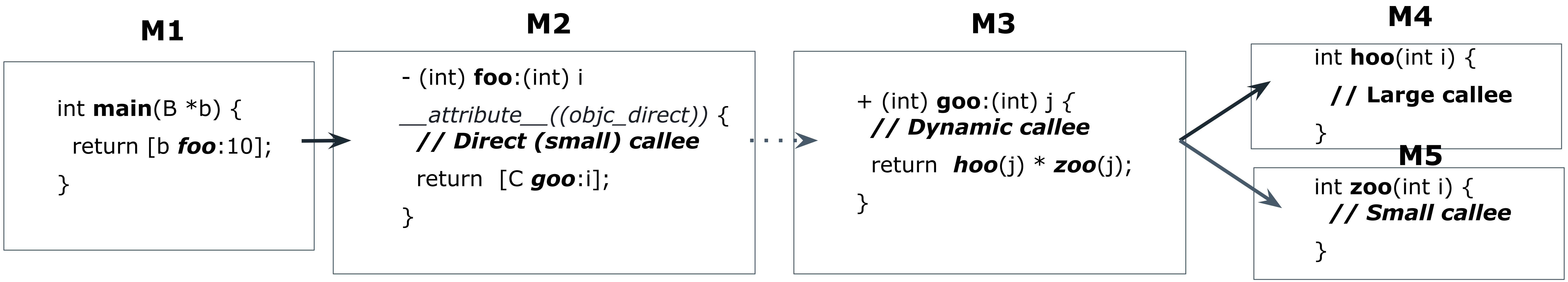
Meta

ThinLTO /w Size Inliner

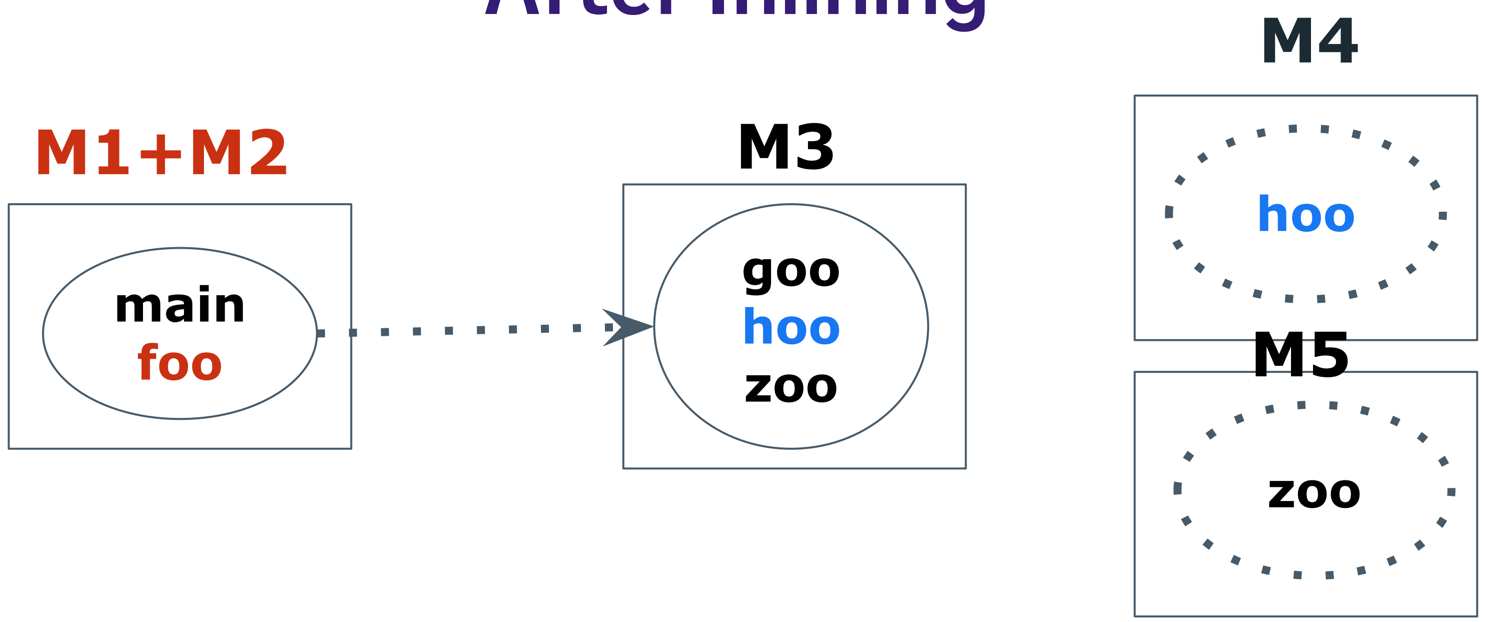


ThinLTO /w Size Inliner

Before Inlining

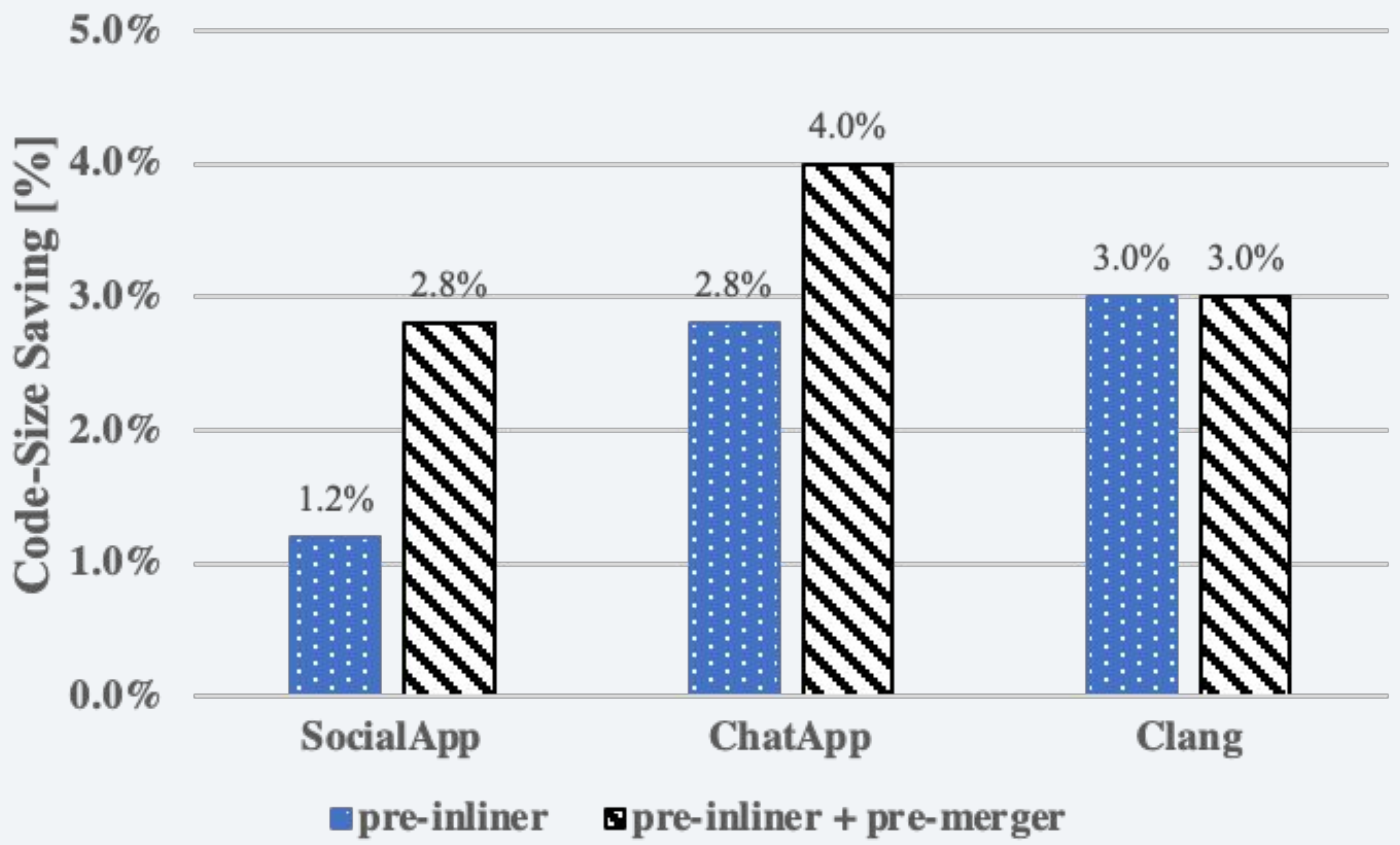


After Inlining



Code Size Impact

- SocialApp
  - A large app written in Objective-C/Swift
- ChatApp
  - A medium size app written in Objective-C/C++
- Clang
  - A compiler benchmark
  - Pre-merger has no impact



Conclusion

- The size inliner for (*Full*)LTO can still improve the size by 0.4% for Clang.
- The size inliner for ThinLTO improved [1]:
  - The code size, 2.8% for SocialApp, and 4.0% for ChatApp.
  - Clang became 3% smaller and 6.1% faster.

[1] Scalable size inliner for mobile applications (WIP), Kyungwoo Lee, Manman Ren, and Shane Nay. 2022.
https://dl.acm.org/doi/10.1145/3519941.3535074