Guided Linking:
Dynamic Linking
Without the Costs

Sean Bartell, Vikram Adve
University of Illinois
LLVM 2020

Motivating Example

```c
// foo.c
void foo(T* p) {
    int n = bar(p);
    // ...
}
```

```c
// bar.c
int bar(T* p) {
    return p->size;
}
```

Can be optimized
(w/ LTO)

Motivating Example

```
// plugin.so
void foo(T* p) {
    int n = bar(p);
    // ...
}
```

```
// library.so
int bar(T* p) {
    return p->size;
}
```

Can't be optimized

Motivating Example

```
// plugin.so
void foo(T* p) {
    int n = bar(p);
    // ...
}
```

```
// library.so
int bar(T* p) {
    return p->size;
}
```

~~Can't be optimized~~

Can be optimized w/ Guided Linking

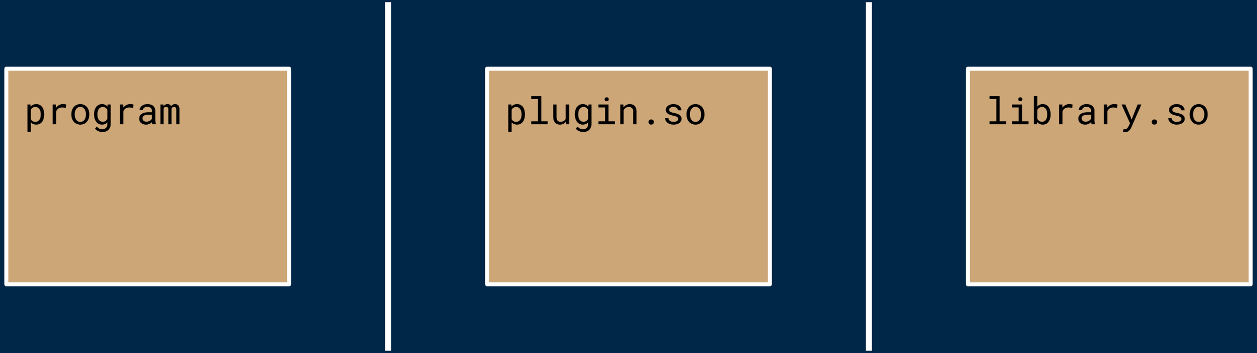
Guided Linking

- Works on existing software
- Requires no code changes
- Python 9% faster
- Boost 57% smaller (multiple versions optimized together)

Overview

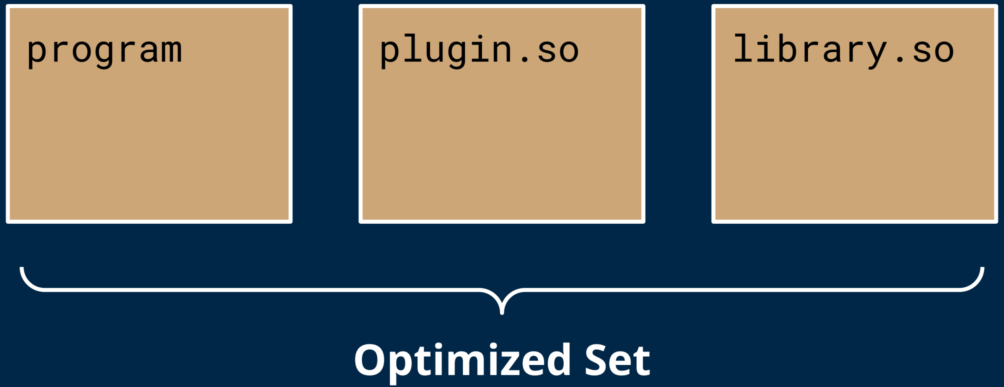
Problem #1

Each program/library is optimized **separately**



Solution #1

Optimize multiple programs/libraries at once



Problem #2

LD_DYNAMIC_WEAK?

/etc/ld.so.cache?

Set-user-ID?

Dynamic linking is **unpredictable**

Modified libraries?

Interposing definitions?

LD_PRELOAD?

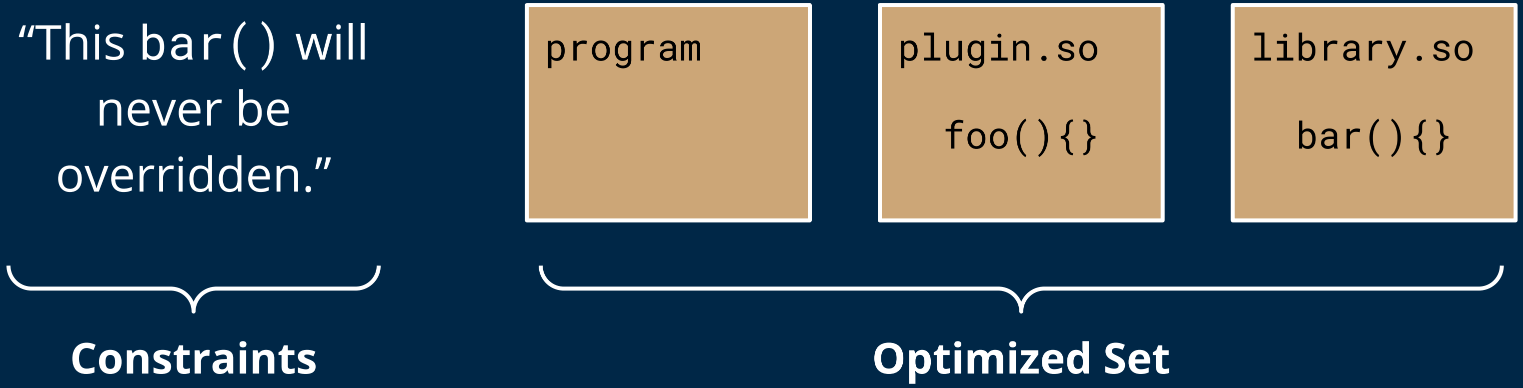
LD_LIBRARY_PATH?

Solution #2

Developer provides **constraints**

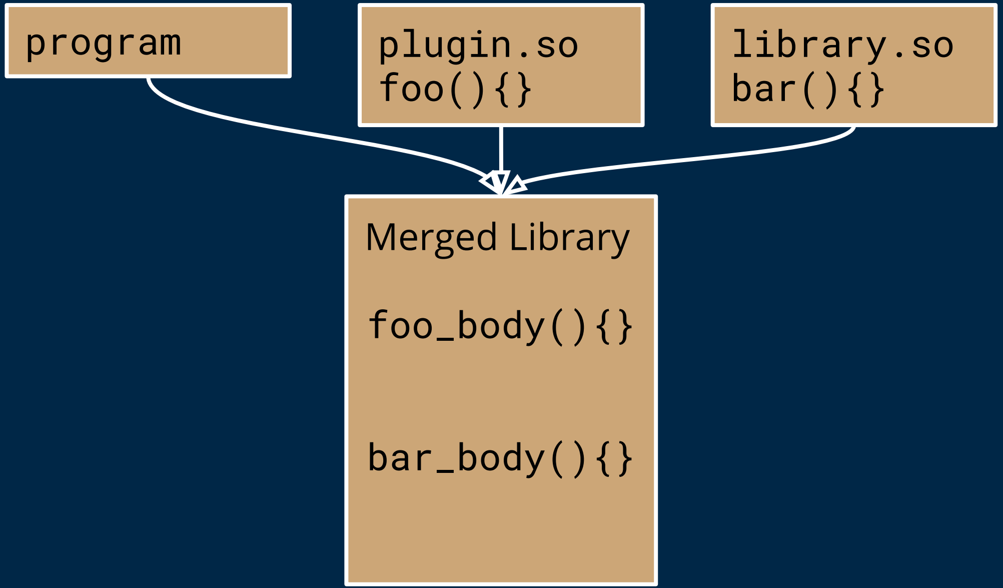
“This `bar()` will never be
overridden by a different `bar()`.”

Guided Linking



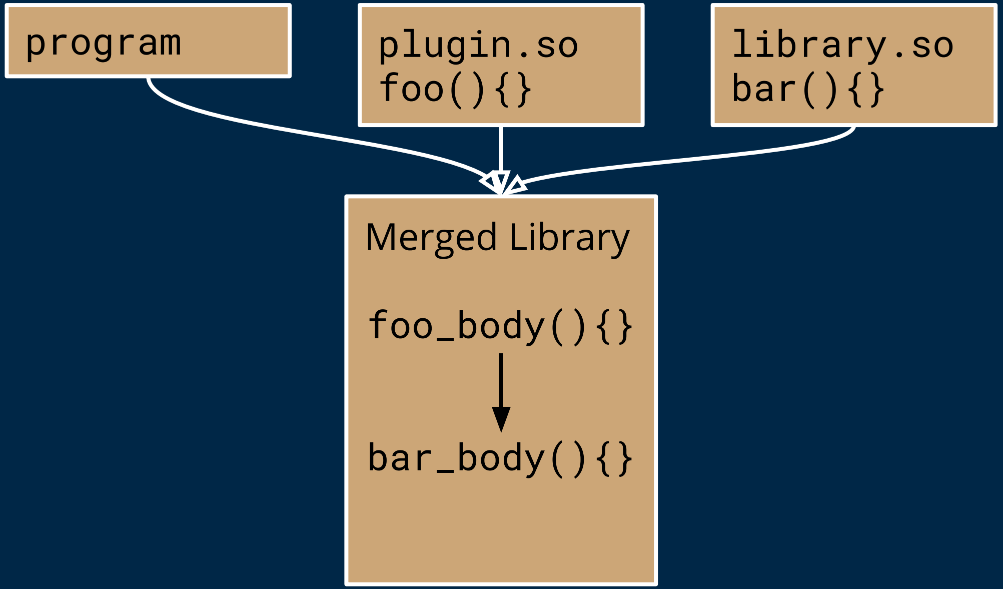
Step 1: move code to a merged library

"This `bar()` will never be overridden."



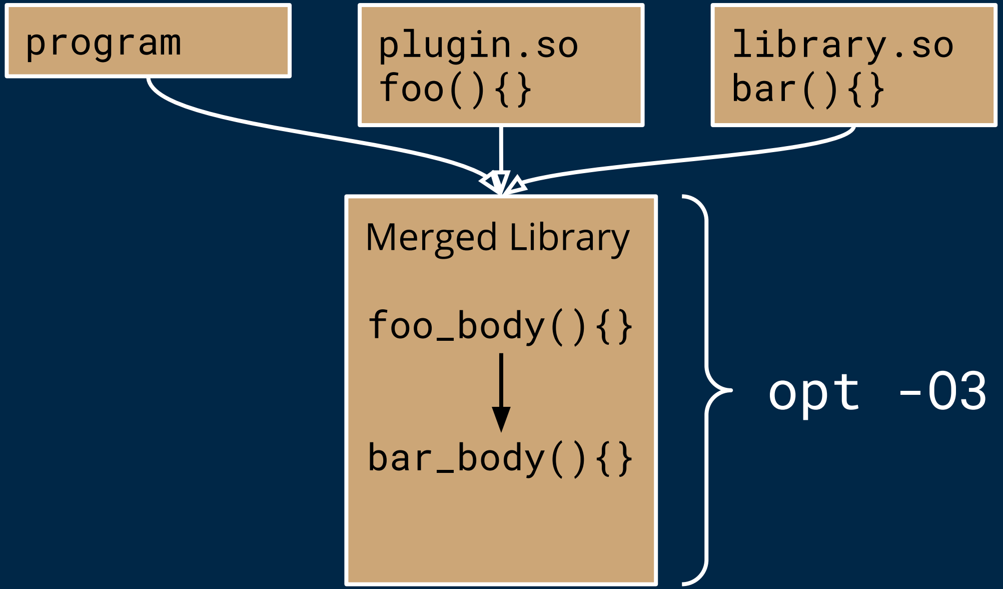
Step 2: static resolution (when possible)

"This `bar()` will never be overridden."



Step 3: LTO does the rest

"This `bar()` will never be overridden."



Choosing the Optimized Set

Choosing the Optimized Set

- Arbitrary set of programs, libraries, plugins
- Must be in bitcode form
- Must be distributed as a single unit
  - Slows down upgrades

Optimized Set Possibilities

- One package
- Entire Docker container
- Entire desktop computer
  - Ship software in bitcode form
  - Re-optimize whenever programs are added

Constraints

Available Constraints

NoOverride	No external overrides
NoUse	No external uses
NoPlugin	No use in plugin
NoWeak	No weak uses or external definitions

Specifying Constraints

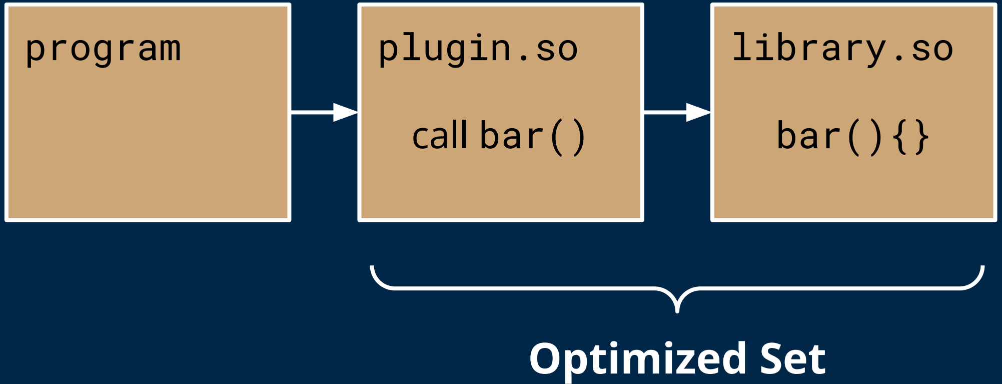
```
default: NoOverride, NoUse, NoPlugin, NoWeak

[Use]
fun:PyInit_*
```

NoOverride: No external overrides

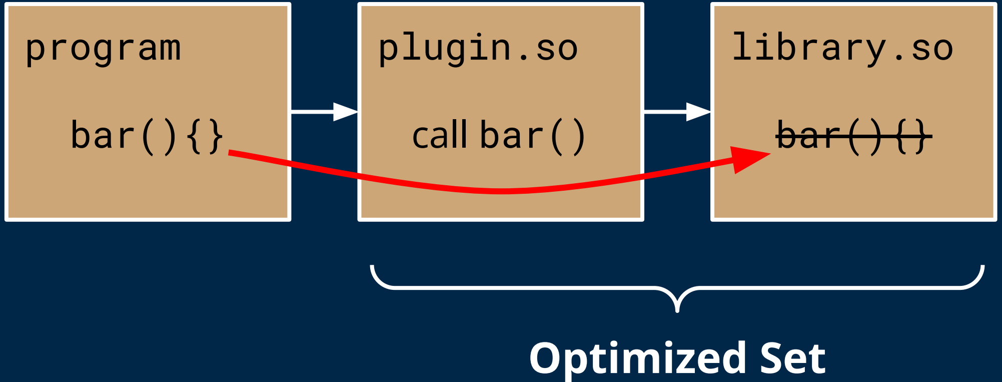
What We Want

We want to inline `bar()`.



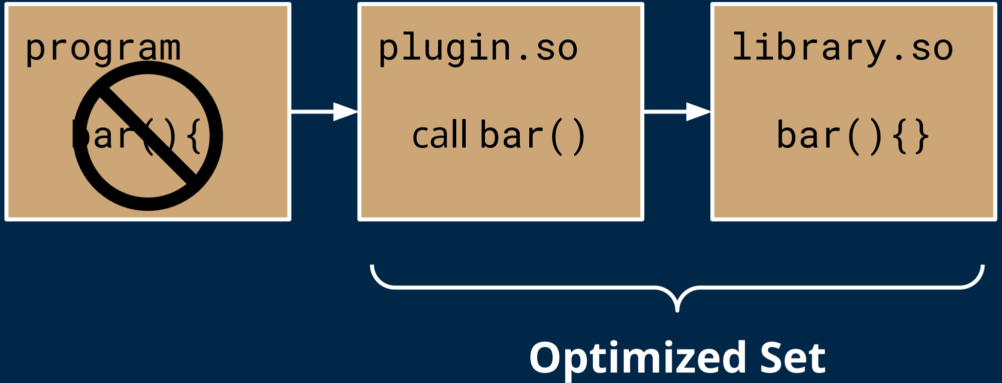
Without the Constraint

Unsafe to inline.
What if `bar()` is overridden?



With the Constraint

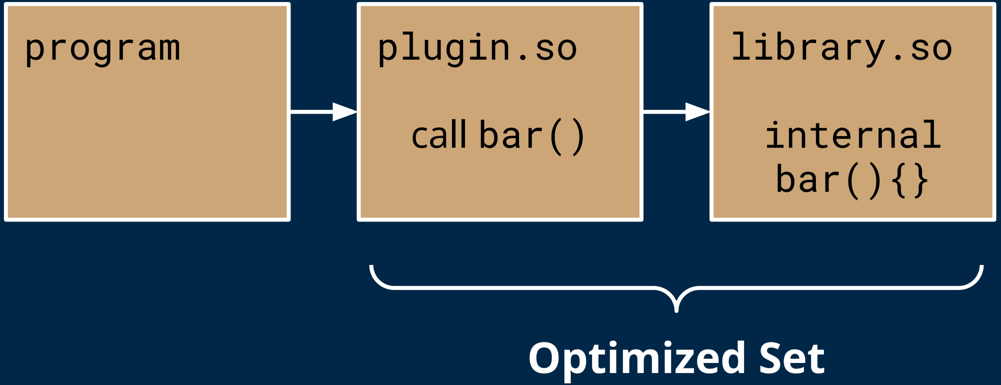
NoOverride guarantees this won't happen.
We can safely inline `bar()`.



NoUse: No external uses

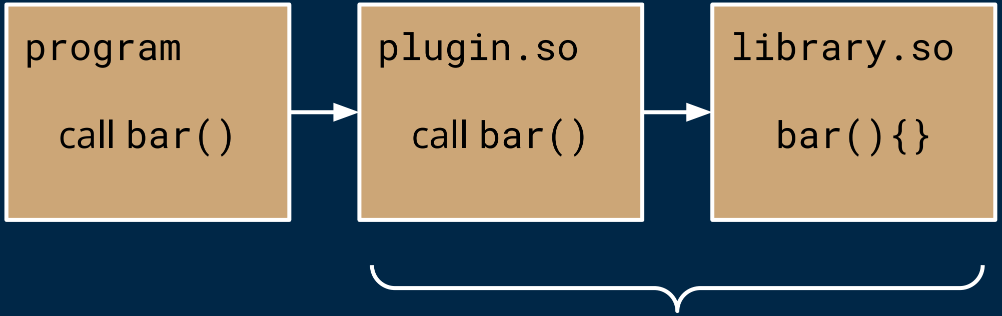
What We Want

We want to internalize `bar()`
in the merged library.



Without the Constraint 1

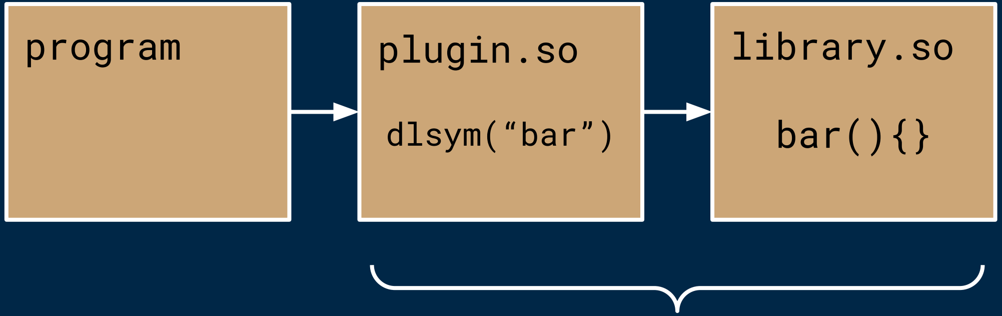
Unsafe to internalize.
Used outside the set.



Optimized Set

Without the Constraint 2

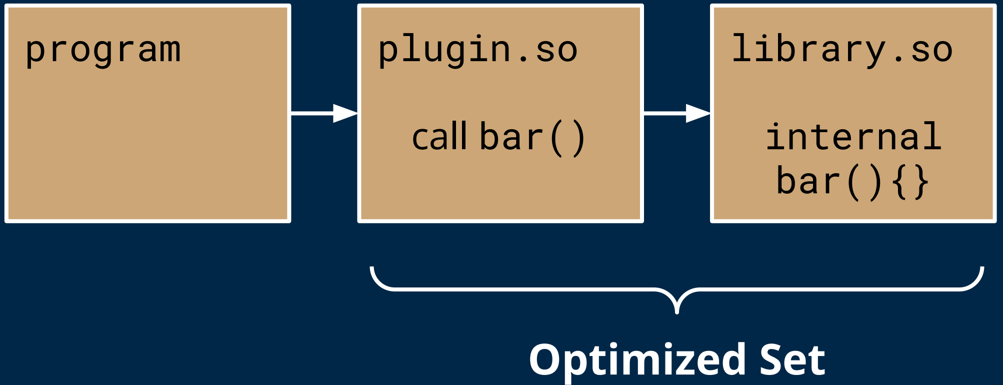
Unsafe to internalize.
Used through dynamic linker.



Optimized Set

With the Constraint

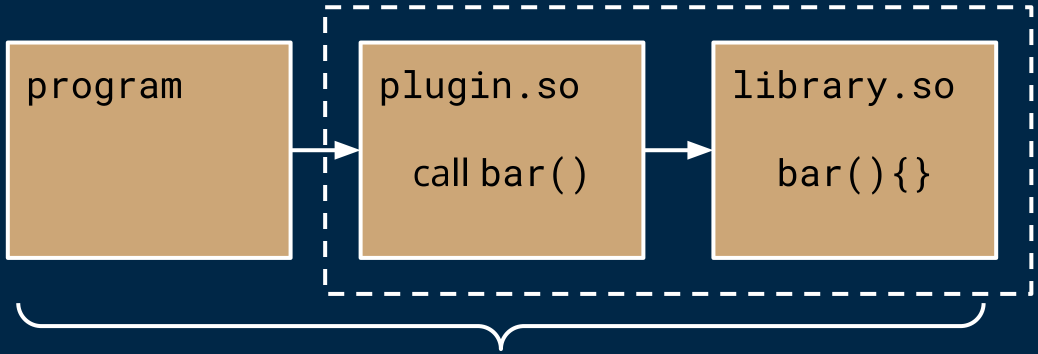
NoUse guarantees neither will happen.
Safe to internalize.



NoPlugin: no use in plugin

Background

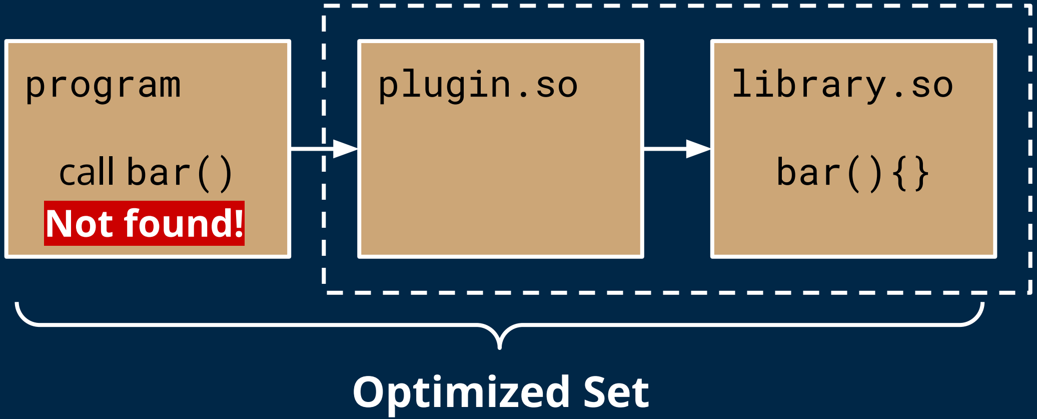
Each plugin has its own
lookup scope (RTLD_LOCAL)



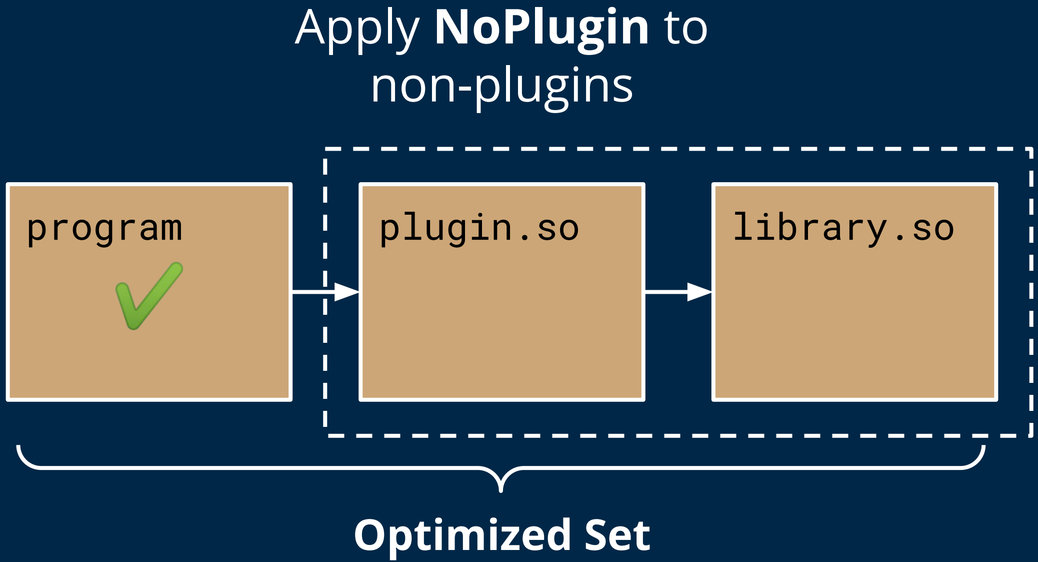
Optimized Set

Moving Code

Moving code that uses external functions requires extra work

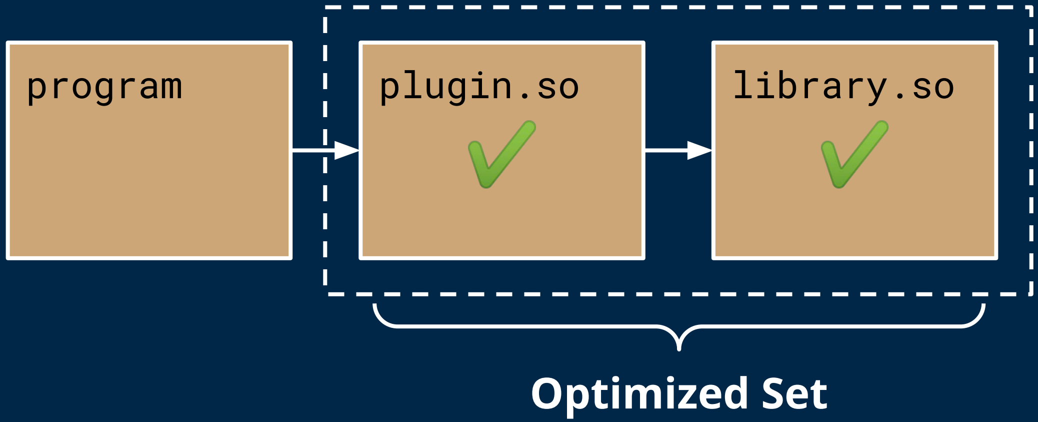


Applying the Constraint



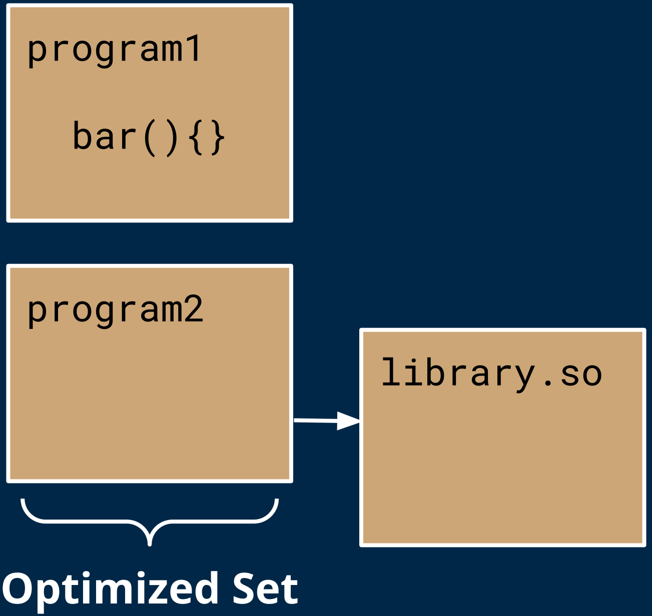
Applying the Constraint

Or, reduce optimized set
to only cover one plugin



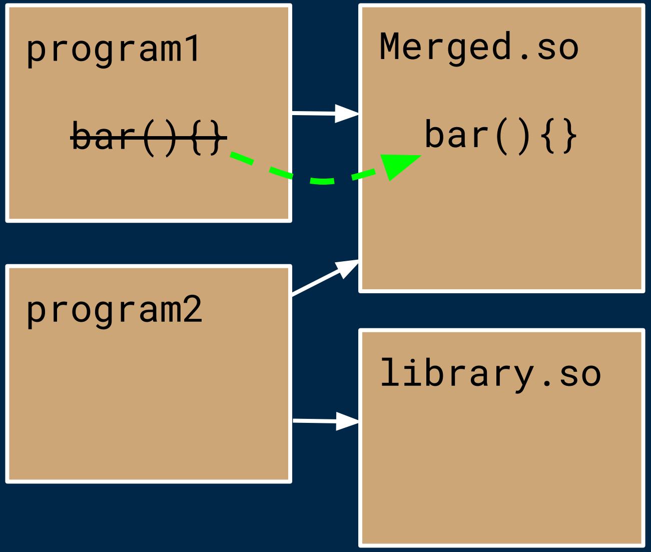
NoWeak: no weak uses or external definitions

What We Want



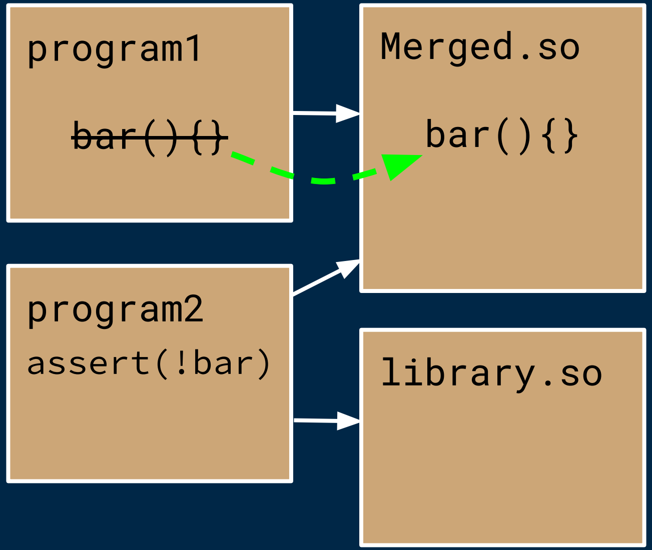
What We Want

We want to move defs to the merged lib



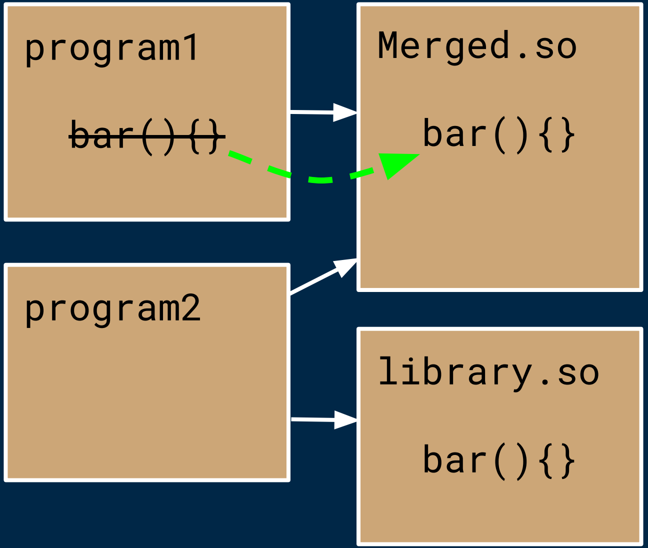
Without the Constraint

Unsafe with weak uses



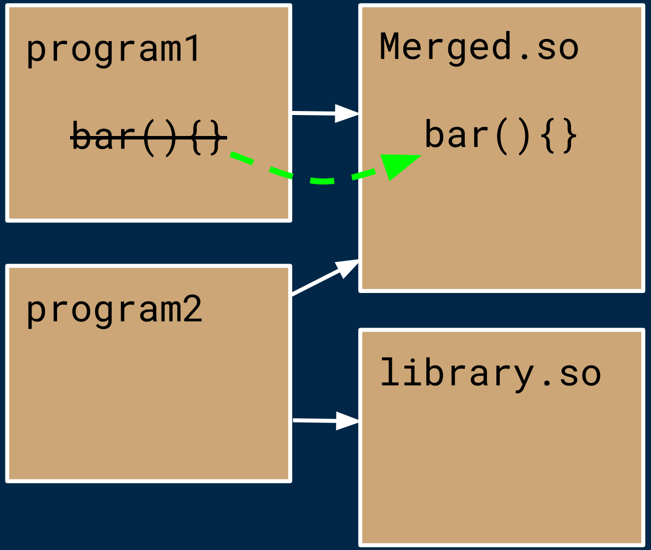
Without the Constraint

Unsafe with other defs



With the Constraint

Safe with **NoWeak**

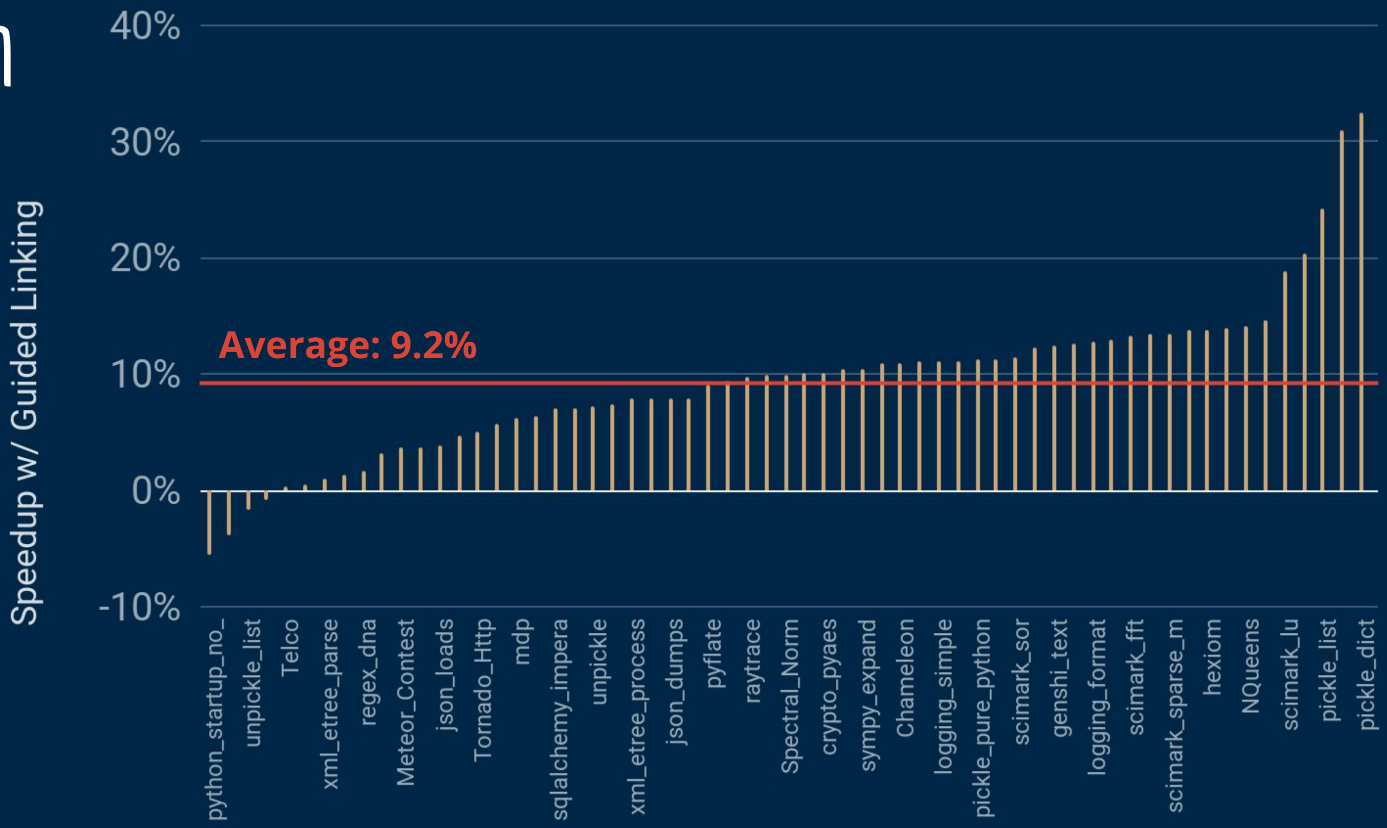


Results

Python

- Optimized set: Python, plus plugin modules
- Constraints: **NoOverride**+**NoUse**+**NoPlugin**+**NoWeak**
  - **NoPlugin** exceptions for plugins & their deps
  - **NoUse** exceptions for functions loaded with dlsym()
- Baseline: O3+LTO+PGO
- Benchmarks: pyperformance suite

Python



Speedup w/ Guided Linking
40%
30%
20%
10%
0%
-10%
Average: 9.2%
python_startup_no_
unpickle_list
Telco
xml_etree_parse
regex_dna
Meteor_Contest
json_loads
Tornado_Http
mdp
sqlalchemy_impera
unpickle
xml_etree_process
json_dumps
pyflate
raytrace
Spectral_Norm
crypto_pyaes
sympy_expand
Chameleon
logging_simple
pickle_pure_python
scimark_sor
genshi_text
logging_format
scimark_fft
scimark_sparse_m
hexiom
NQueens
scimark_lu
pickle_list
pickle_dict

Python Example 1

```
// _pickle.so
int save(...) {
    ...
    while (_PyDict_Next(...))
        ...
    ...
}

// libpython3.7m.so
int _PyDict_Next(...) { ... }
```

Guided Linking enables inlining here.

Python Example 2

```
// libpython3.7m.so
PyTypeObject PyFrame_Type = {
    ...
    (destructor)frame_dealloc,
    ...
};
```

- Guided Linking internalizes this variable
- LLVM determines it's never modified
- `frame_dealloc()` can be inlined.

Boost and Protobuf

- Optimized set: multiple versions of the same library
- Constraint: **NoPlugin**
- Function deduplication
  - Normalize functions
  - Merge bodies of identical functions
- Baseline: `clang -Oz -flto`

Size reductions

57%

11 versions of Boost

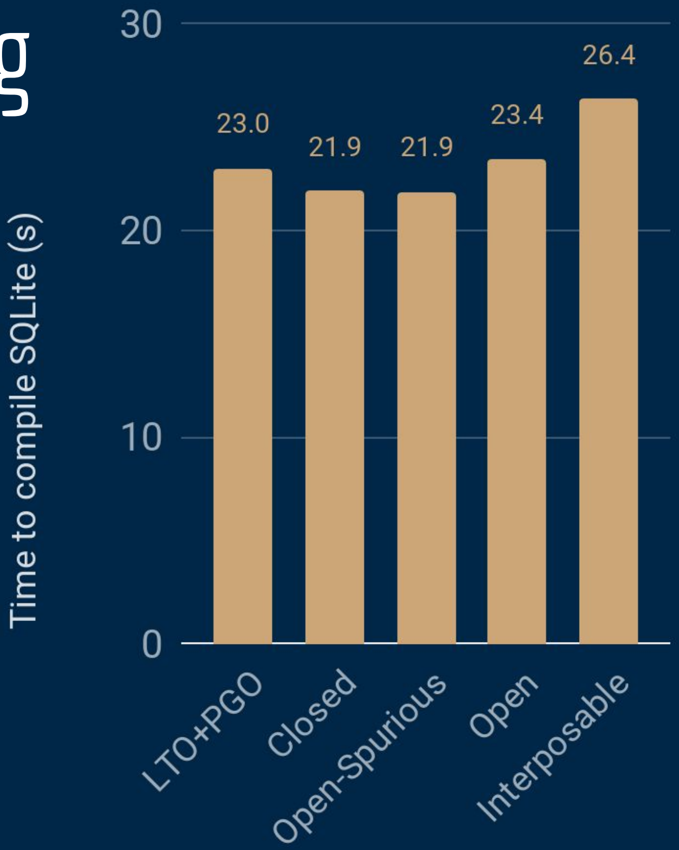
31%

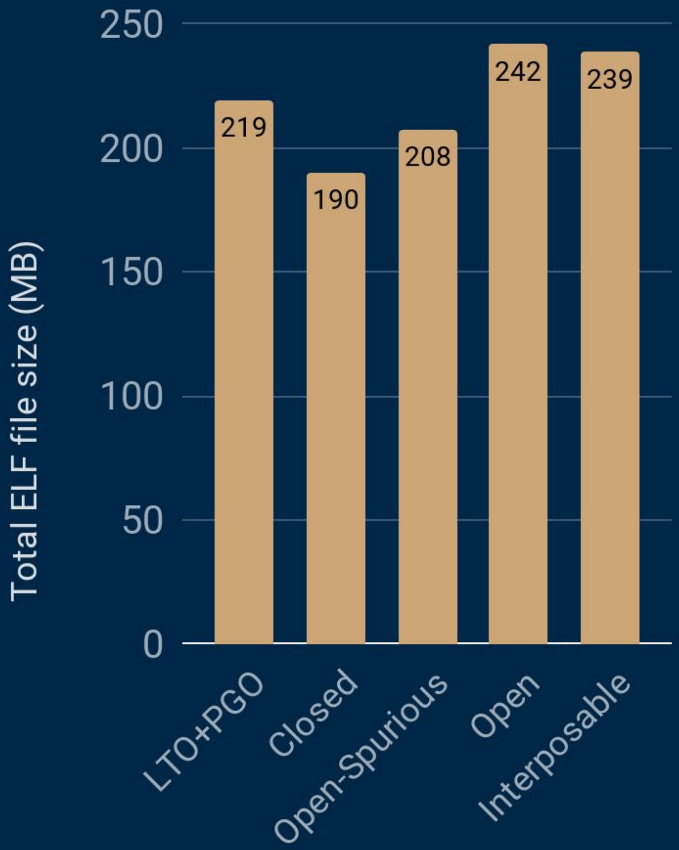
8 versions of Protobuf

Clang

- Optimized set: all of LLVM and Clang
  - Built with BUILD_SHARED_LIBS (94 programs, 227 libs)
  - Caveat: LLVM already has better options
- Constraints: 4 different levels
  - Closed: **NoPlugin**+**NoOverride**+**NoWeak**+**NoUse**
  - Open-Spurious: **NoPlugin**+**NoOverride**+**NoWeak**
  - Open: **NoPlugin**+**NoOverride**
  - Interposable: **NoPlugin**
- Baseline: O3+LTO+PGO
- Benchmark: compile SQLite

Clang


Time to compile SQLite (s)
30
20
10
0
23.0
21.9
21.9
23.4
26.4
LTO+PGO
Closed
Open-Spurious
Open
Interposable


Total ELF file size (MB)
250
200
150
100
50
0
219
190
208
242
239
LTO+PGO
Closed
Open-Spurious
Open
Interposable

Future Work

Automatic Multicall

- Automatically make a program like Busybox
- Get rid of dynamic linker entirely
- Will Dietz' Allmux did this in limited cases
    - No libraries left out
    - No plugins or dlsym
    - No multiply-defined symbols

Conclusion

- Guided Linking can optimize dynamically linked code
- Use an **optimized set** and **constraints**
- Speed up Python by **9%**
- Combine Boost/Protobuf versions and shrink by **31-57%**
- Speed up Clang+LLVM by **5%** and shrink by **13%**

Conclusion

- Guided Linking can optimize dynamically linked code
- Use an **optimized set** and **constraints**
- Speed up Python by **9%**
- Combine Boost/Protobuf versions and shrink by **31-57%**
- Speed up Clang+LLVM by **5%** and shrink by **13%**

Questions? Ideas? smbarte2@illinois.edu