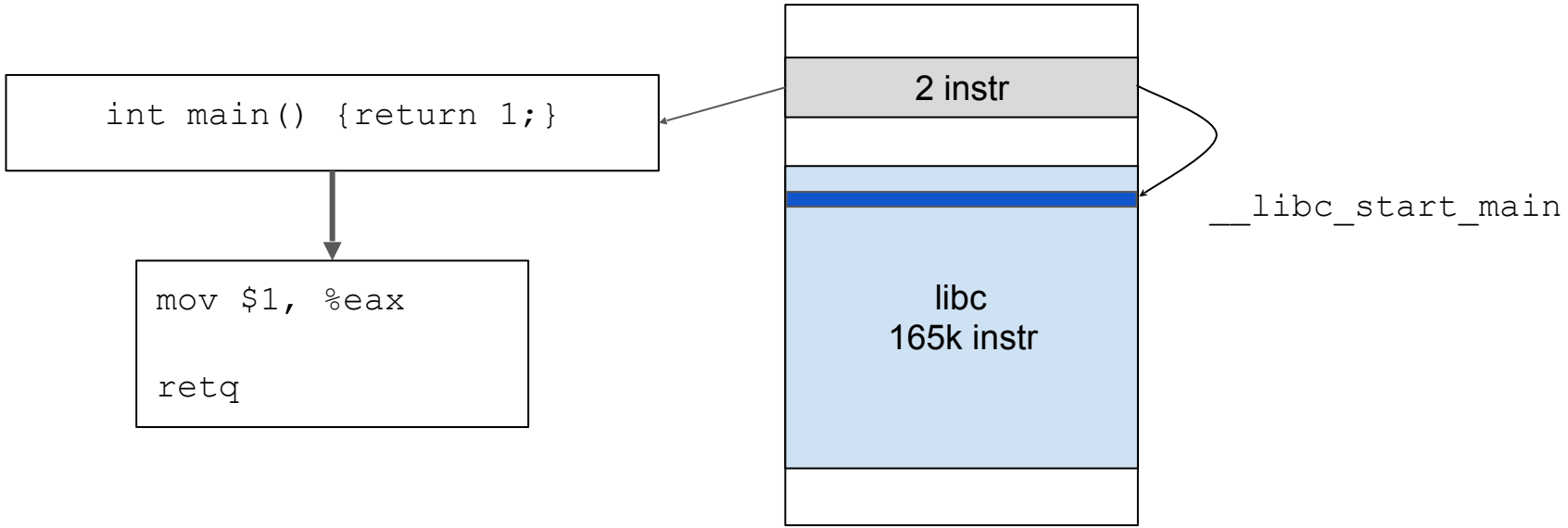
Debloating Software through Piece-Wise Compilation & Loading

Anh Quach, Aravind Prakash
Binghamton University

Lok Yan
Air Force Research Laboratory

Programs are bloated



glibc footprint (Ubuntu 16.04 LTS Desktop)

Program	% glibc functions Imported
vlc	21%
rhythmbox	20%
unpkg	19%
gst-xmlinspect-0.10	19%
kubuntu-debug-installer	19%
soffice.bin	19%
...	**...**
Mean	**5%**

Popular Shared Libraries (Ubuntu 16.04)

Library	# Programs Use the Library	Average % of Functions Used
glibc	1932	24.64
libm	284	7.06
libstdc++	266	37.77
...		
libXau	86	7.13
libselinux	72	8.57
Mean (top 15)	**279**	**10.22**

Static Dead Code Elimination

- Compiler: intra-procedural optimization
- Static Linker: inter-module optimization

Our Approach: Remove Unused Code

Piece-wise: Inter-module late stage debloating framework

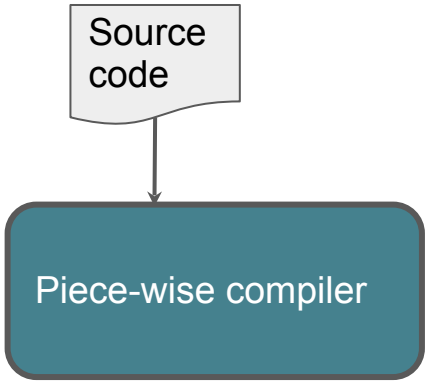
- Piece-wise compiler: generate intra-module dependencies (dependency graph)
- Piece-wise loader:
  - Identify inter-module dependency using dependency graph
  - Remove unused code

Bridge the gap between **early** (compilation) and **late** (loading) stages.

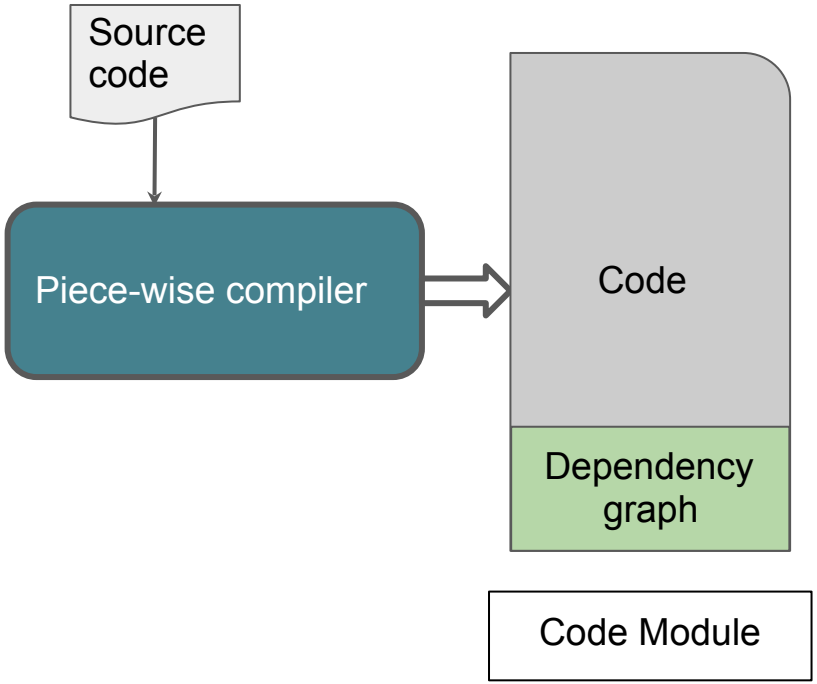
Challenges

- **Modular Interdependencies**: one module depends on multiple modules
- **Late Symbol Binding**: statically unknown, depends on runtime information
- **Code Pointers**: indirect branches
- **Hand-written Assembly**: assembly code not analyzed by compiler
- **Dynamically Loaded Libraries**: statically unknown dependencies

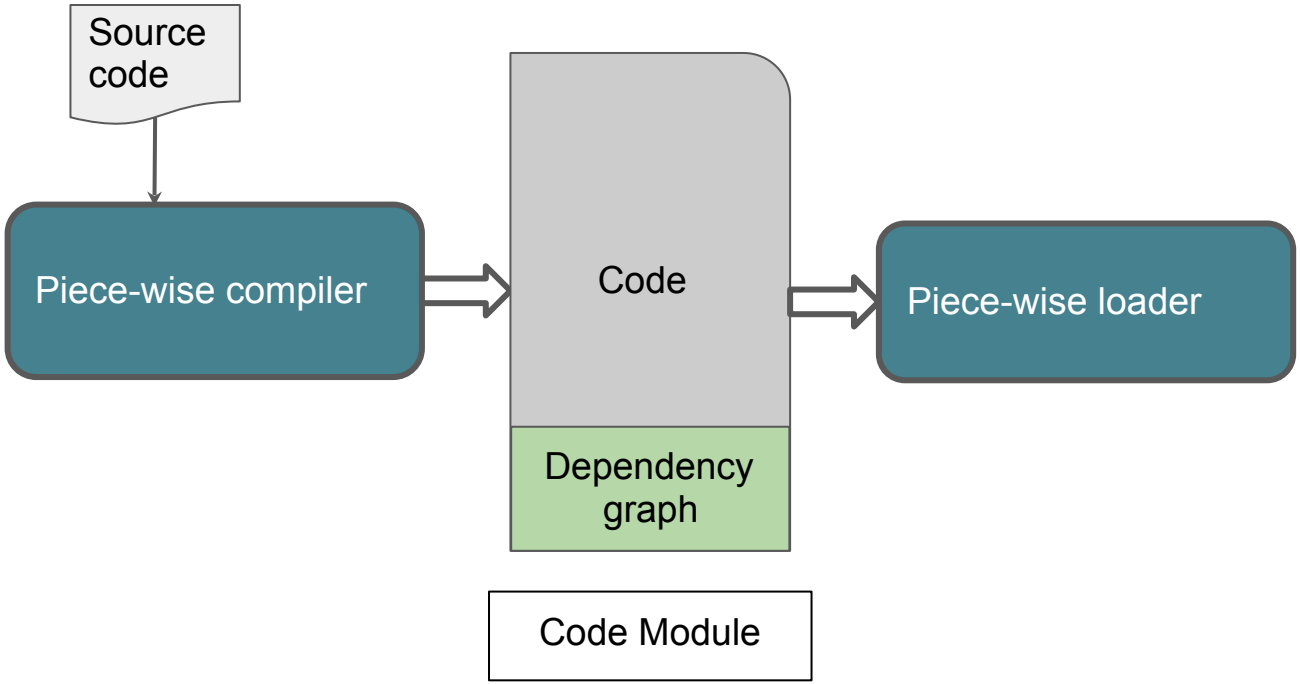
Piece-wise System Design



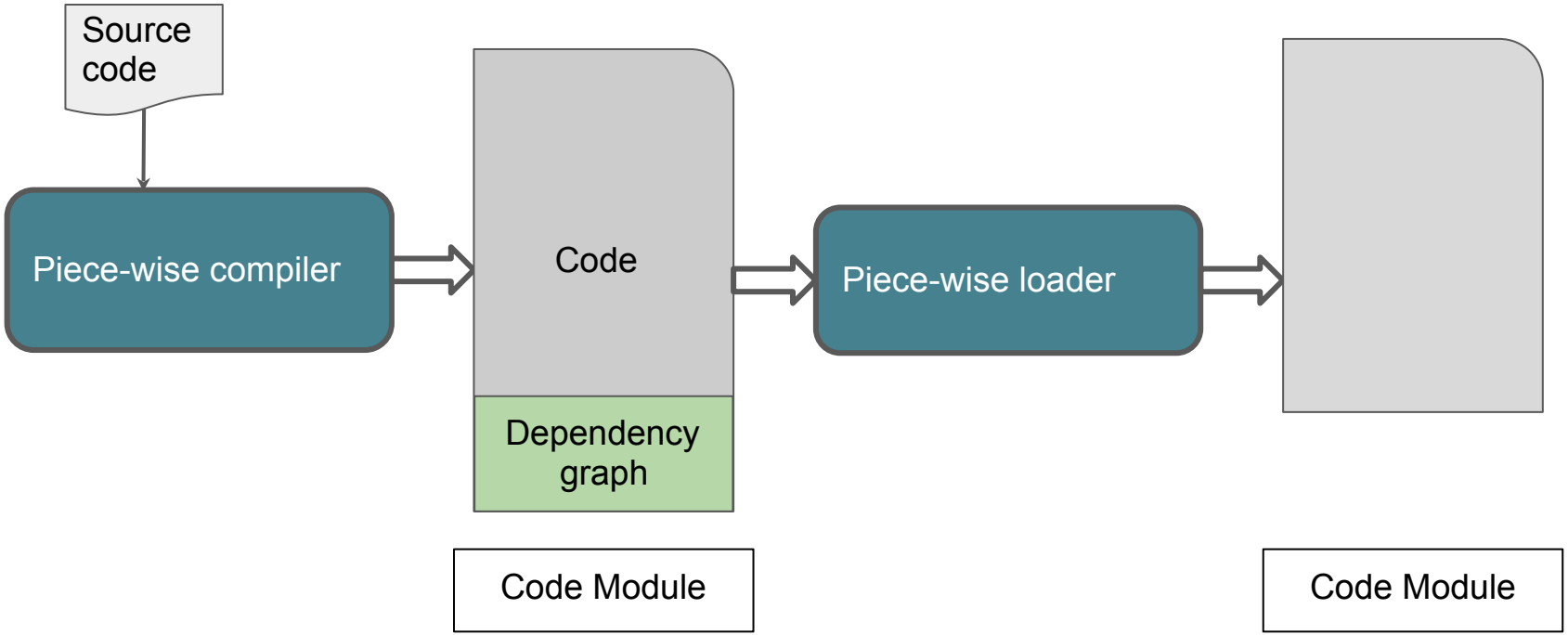
Piece-wise System Design



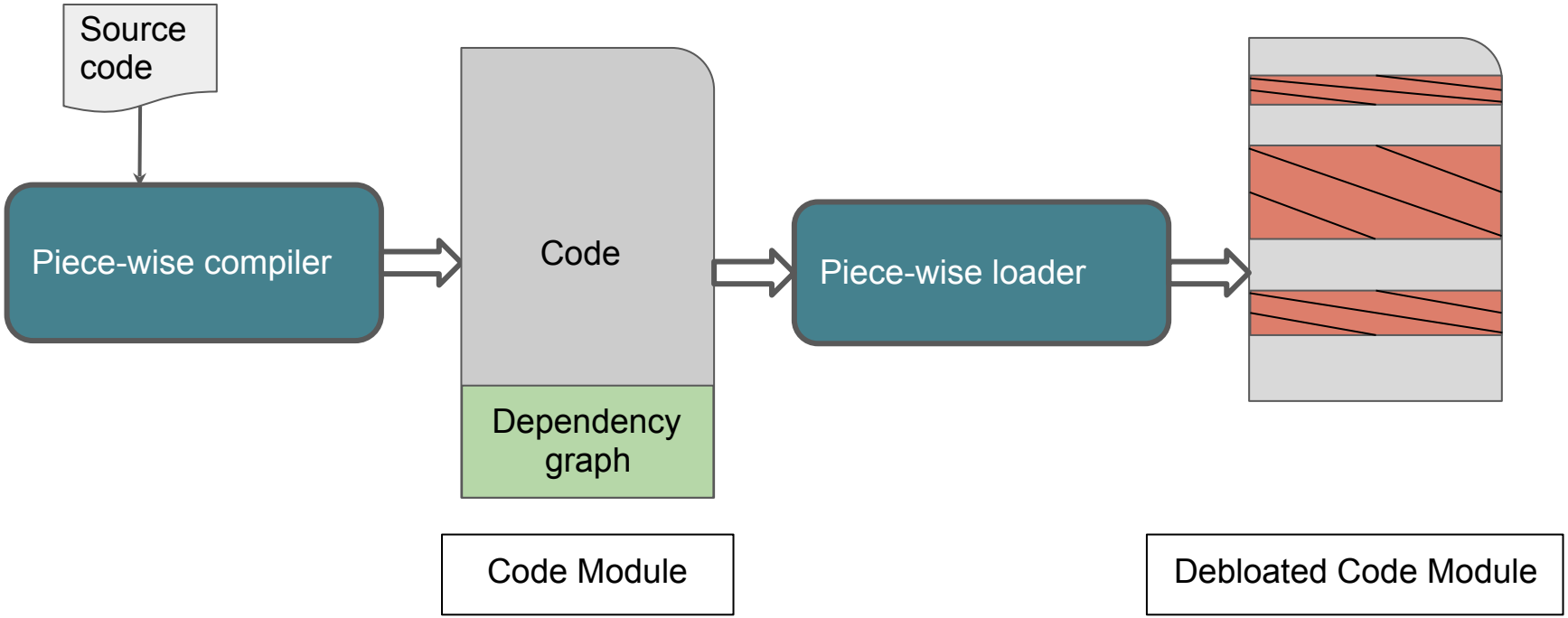
Piece-wise System Design



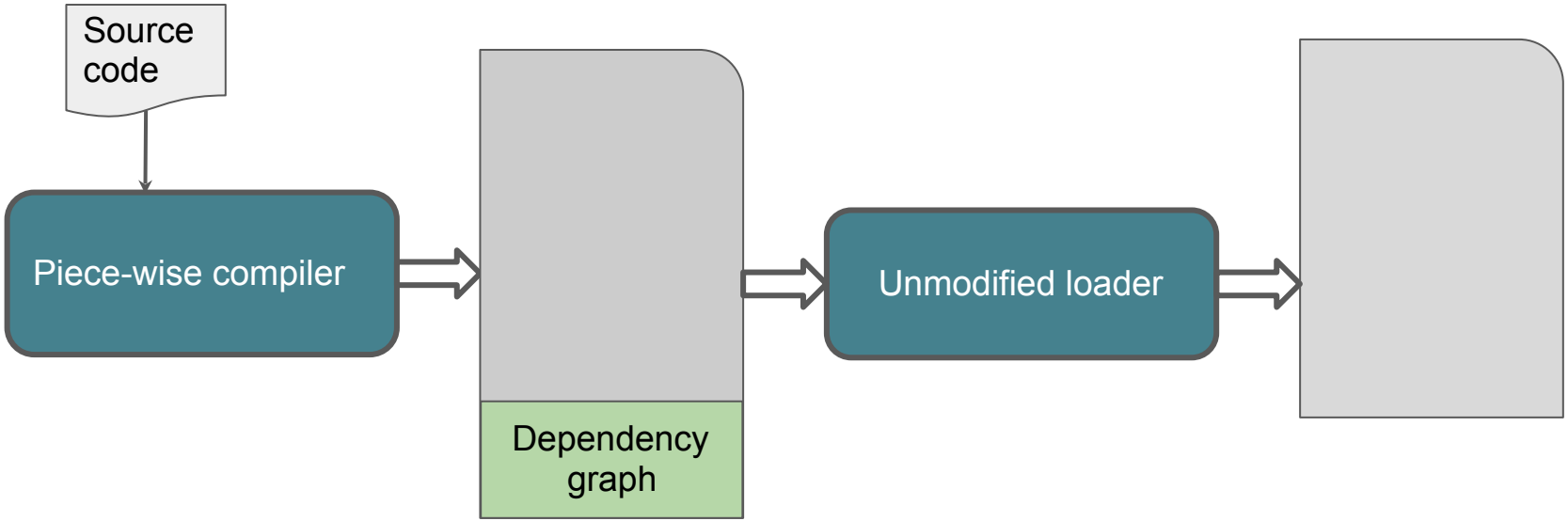
Piece-wise System Design



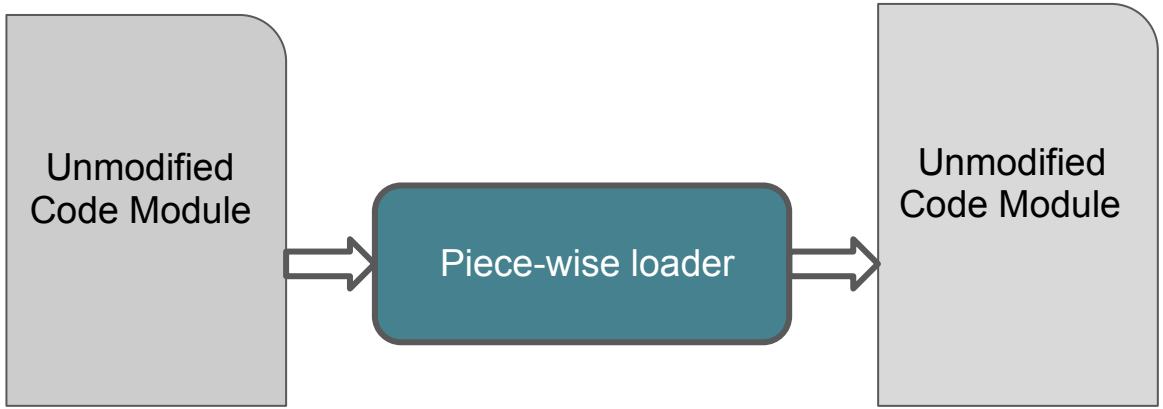
Piece-wise System Design



Backwards Compatibility



Backwards Compatibility



Piece-wise Compiler

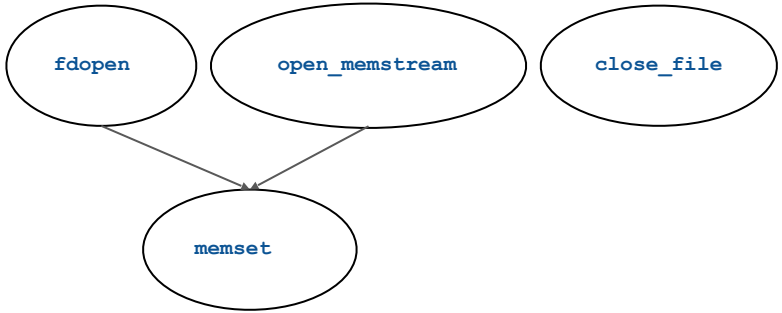
- Generate call graph
- Perform code pointer analysis
  - Global Scan
  - Localized Scan
  - Pointer Analysis
  - C++: object-sensitive analysis
- Analyze inlined/hand-written assembly
- Generate code dependency graph

Libc Example: Dependency Graph (Call Graph only)

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```

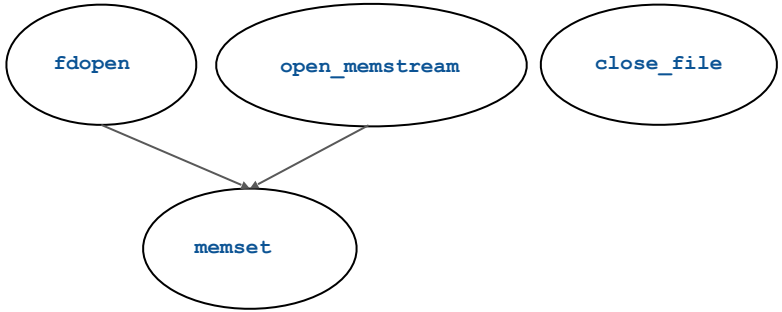


Libc Example: Dependency Graph (Call Graph only)

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```



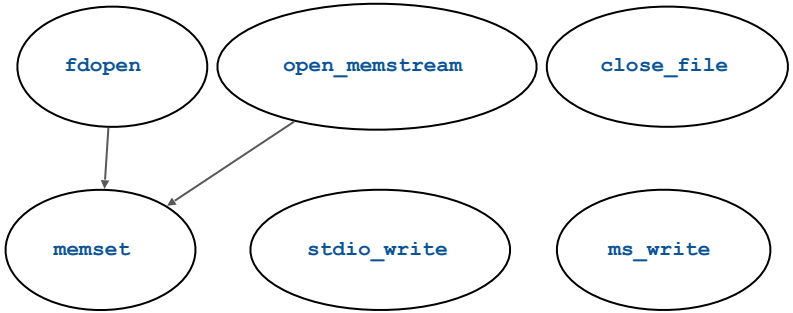
Missing:
stdio_write
ms_write

Libc Example: Global Scan

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```



Global Dependency: `stdio_write, ms_write`

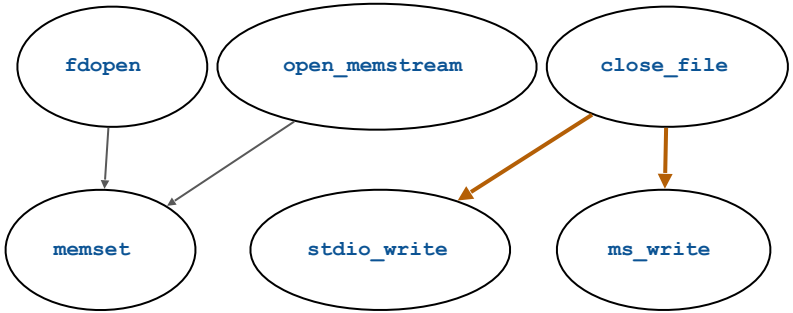
```
fdopen: memset
open_memstream: memset
close_file:
```

Libc Example: Pointer Analysis

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```



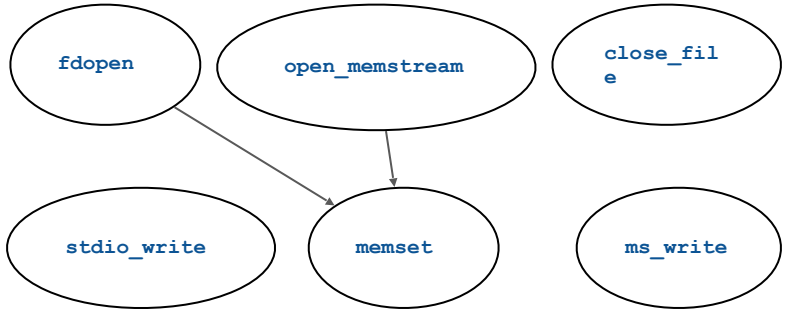

```
fdopen: memset
open_memstream: memset
close_file: stdio_write, ms_write
```

Libc Example: Localized Scan

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```



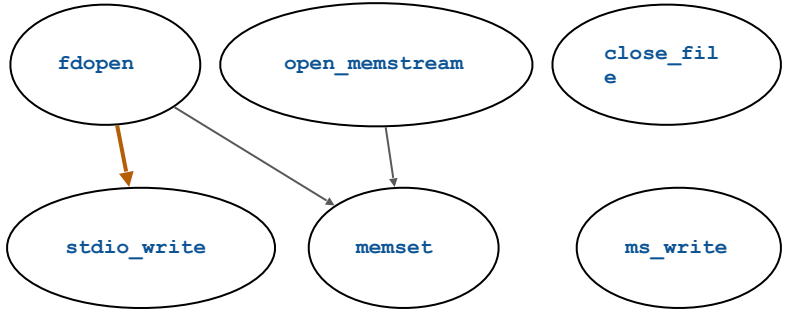

```
fdopen: stdio_write, memset
open_memstream: ms_write, memset
close_file:
```

Libc Example: Localized Scan

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```



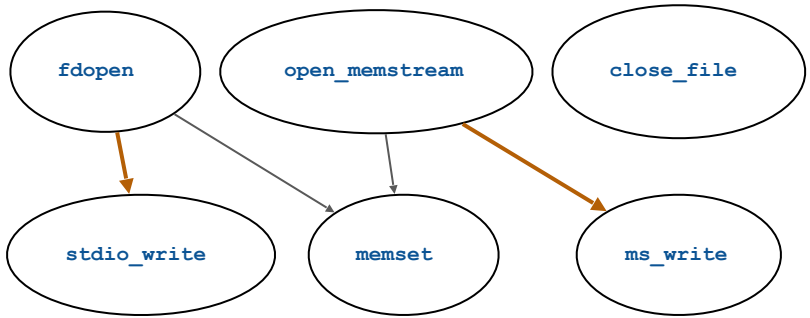

```
fdopen: stdio_write, memset
open_memstream: ms_write, memset
close_file:
```

Libc Example: Localized Scan

```
FILE *fdopen() {
  FILE *f = malloc(...);
  f->write = stdio_write;
  memset();
  ...
  return f;
}
```

```
FILE *open_memstream(){
  FILE *f = malloc(sizeof *f + UNGET + BUFSIZ);
  f->write = ms_write;
  memset();
  ...
  return f;
}
```

```
void close_file(FILE *f) {
  f->write();
}
```

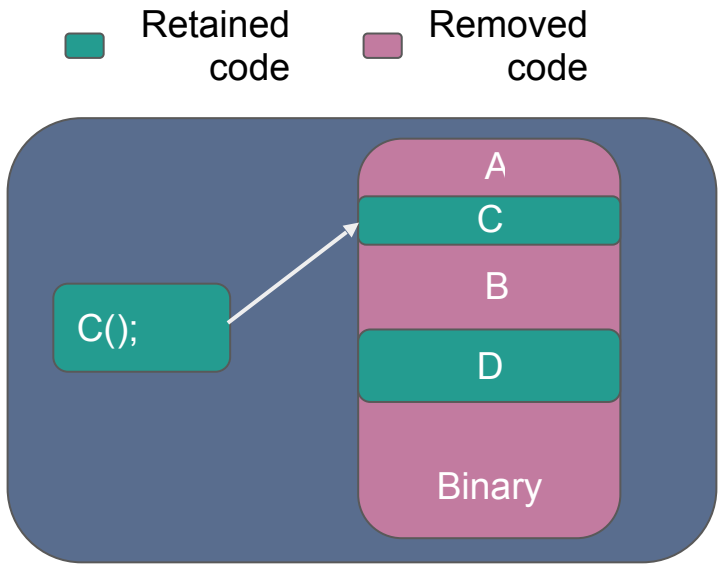



```
fdopen: stdio_write, memset
open_memstream: ms_write, memset
close_file:
```

Piece-wise Loader

- Reads dependency graph
- Preloads libraries
- Performs early binding
- Identifies unused code
- Removes dead code

A: B, C, D
B:
C: D
D:



CFI and Piece-wise

Piece-wise is complimentary to CFI:

- Reduces attack surface
- Provides post-compromise protection

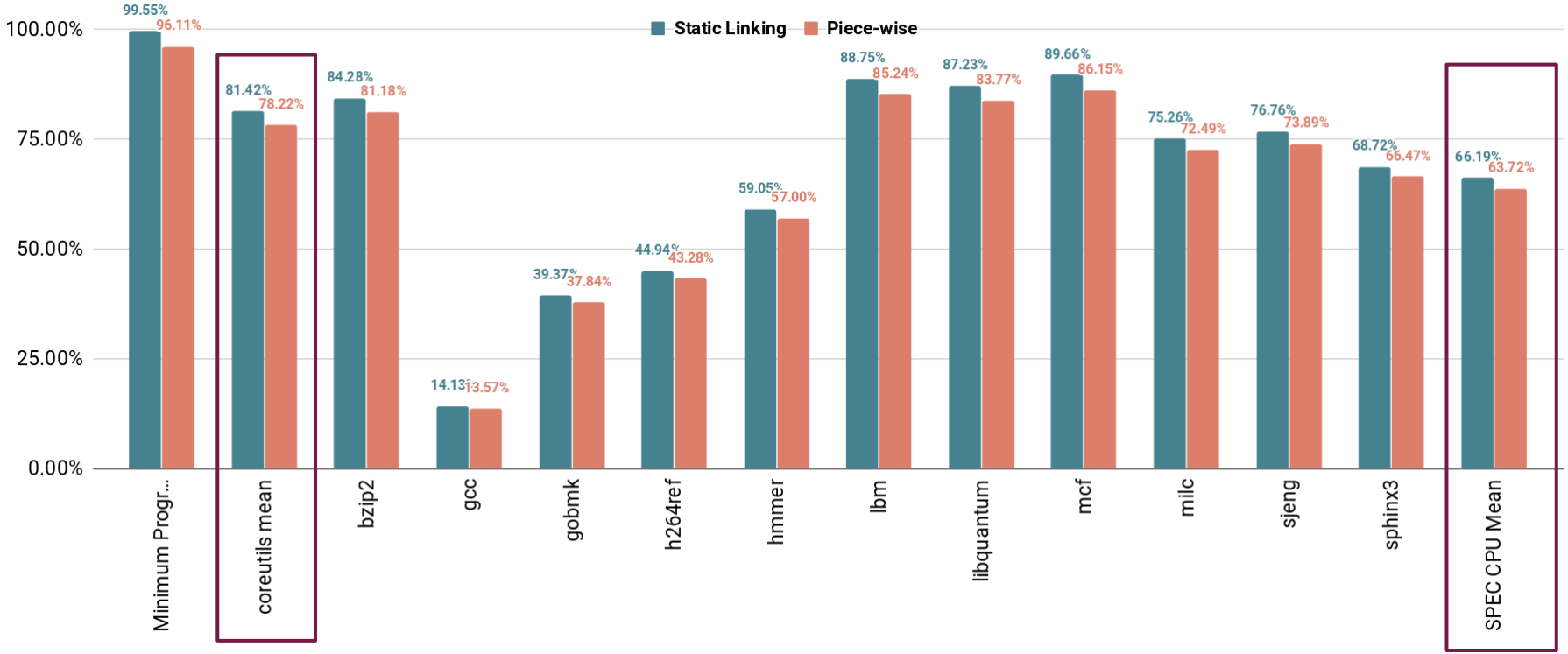
Evaluation

1. Compile-time Overhead
2. Debloating Capability
3. Security Impacts

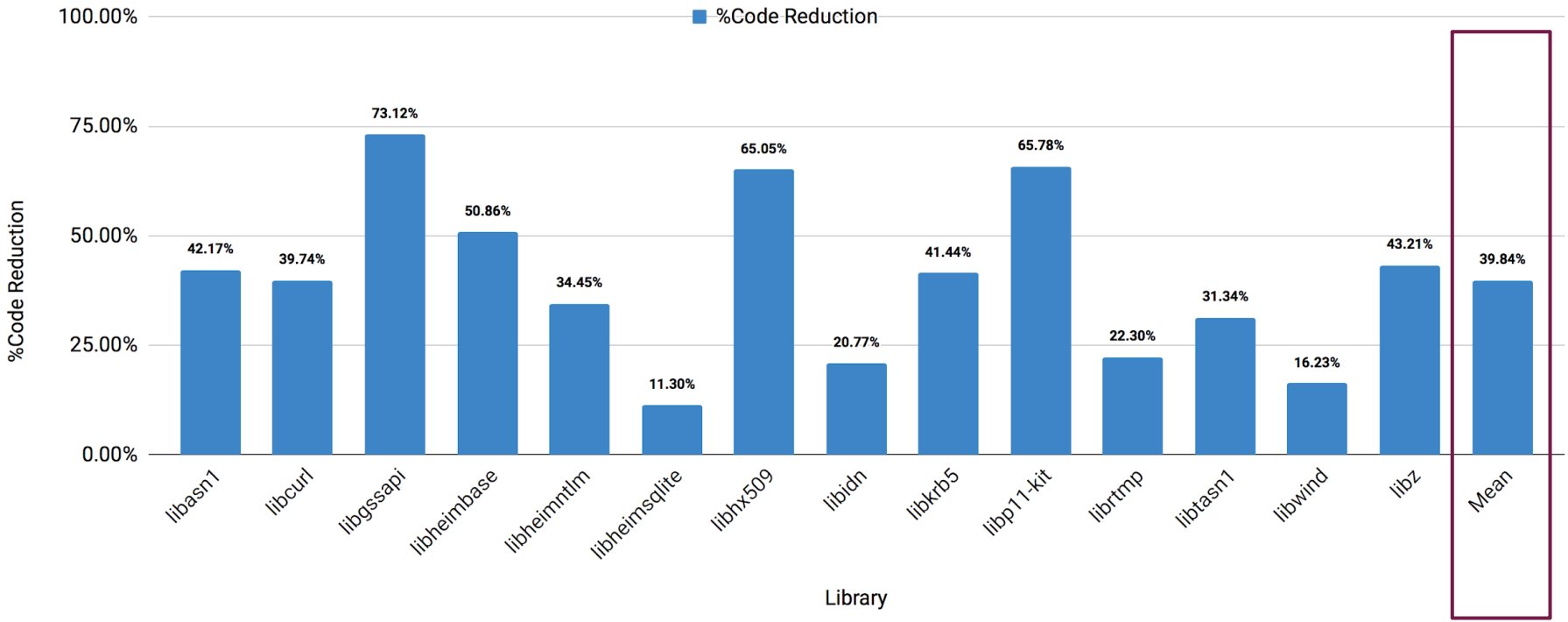
Compile Time (millisecond)

Library	Global Scan	Pointer Analysis	Localized Scan
musl-libc	73	28661	158
libasn1	40.80	16,000	41.40
libcurl	23	891	79.10
libgssapi	14.10	31,600	132
libheimbase	6.30	1,570	8.94
libheimntlm	0.81	275	1.02
libheimsqlite	406	241,000	3,380
libhx509	22.20	12,700	4.07
libidn	0.67	0.68	0.68
libkrb5	165	20,700	776
libp11-kit	6.95	4,330	0.89
librtmp	2.66	1,000	3.31
libtasn1	2.19	1,370	2.36
libwind	0.27	186	0.25
libz	1.20	1,530	7.63

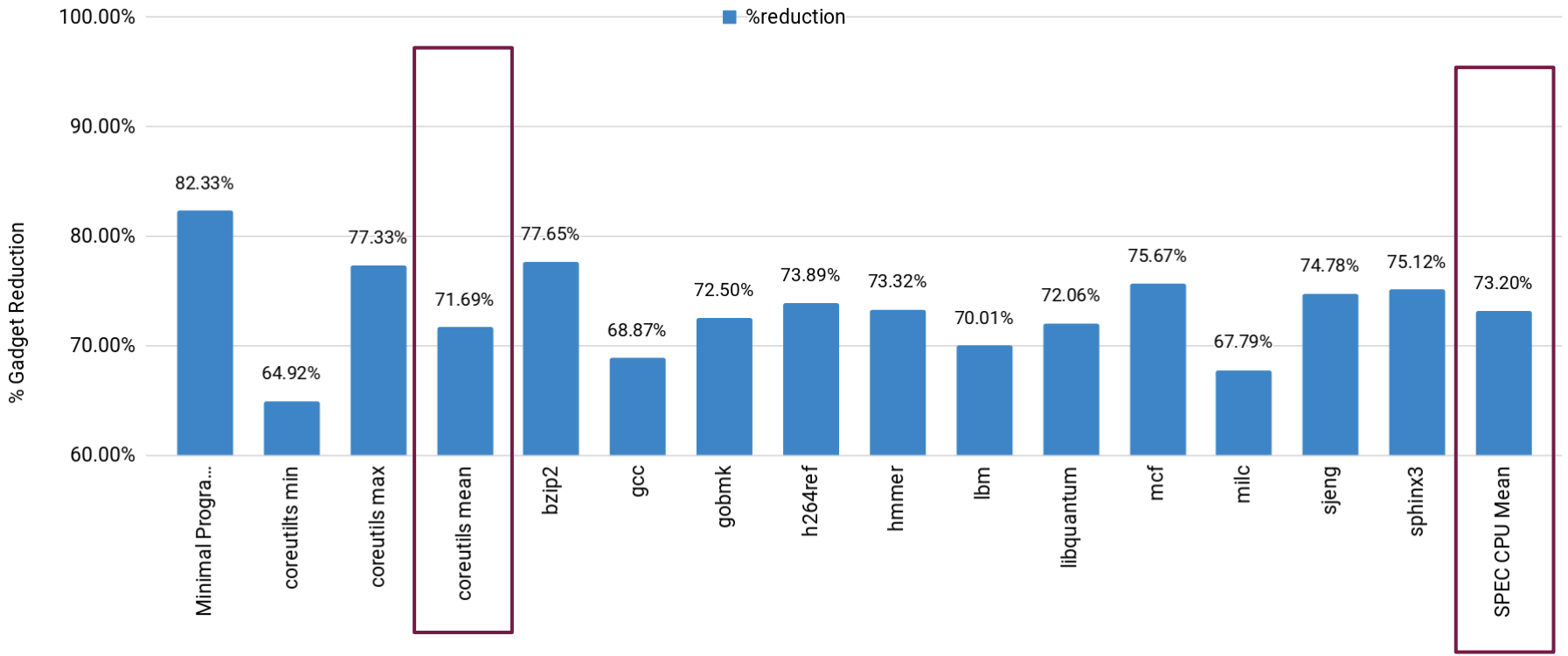
Piece-wise vs Static Linking for libc



Debloating curl



Gadget Reduction for libc



Vulnerability Elimination

Library	CVE	Functions	Program	Vulnerability Type
zlib	CVE-2016-9842	inflateMark	git, curl, LibreOffice, firefox	Undefined Behavior
libcurl	CVE-2016-7167	curl_escape, curl_easy_escape, curl_unescape, curl_easy_unescape	curl	Integer Overflow
	CVE-2014-3707	curl_easy_duphandle	curl, cmake	Out-of-bound Read
	CVE-2016-9586	curl_mprintf	cmake	Buffer Overflow

Artifact

A docker image of piece-wise toolchain is available at:

https://github.com/bingseclab/piecewise

Email: aquach1@binghamton.edu