~~Everything~~ I Wish They Told Me About Linkers

Some Stuff

Ofek Shilon

2023

And Linux/Windows diffs

Focusing on shared libs

And Loaders

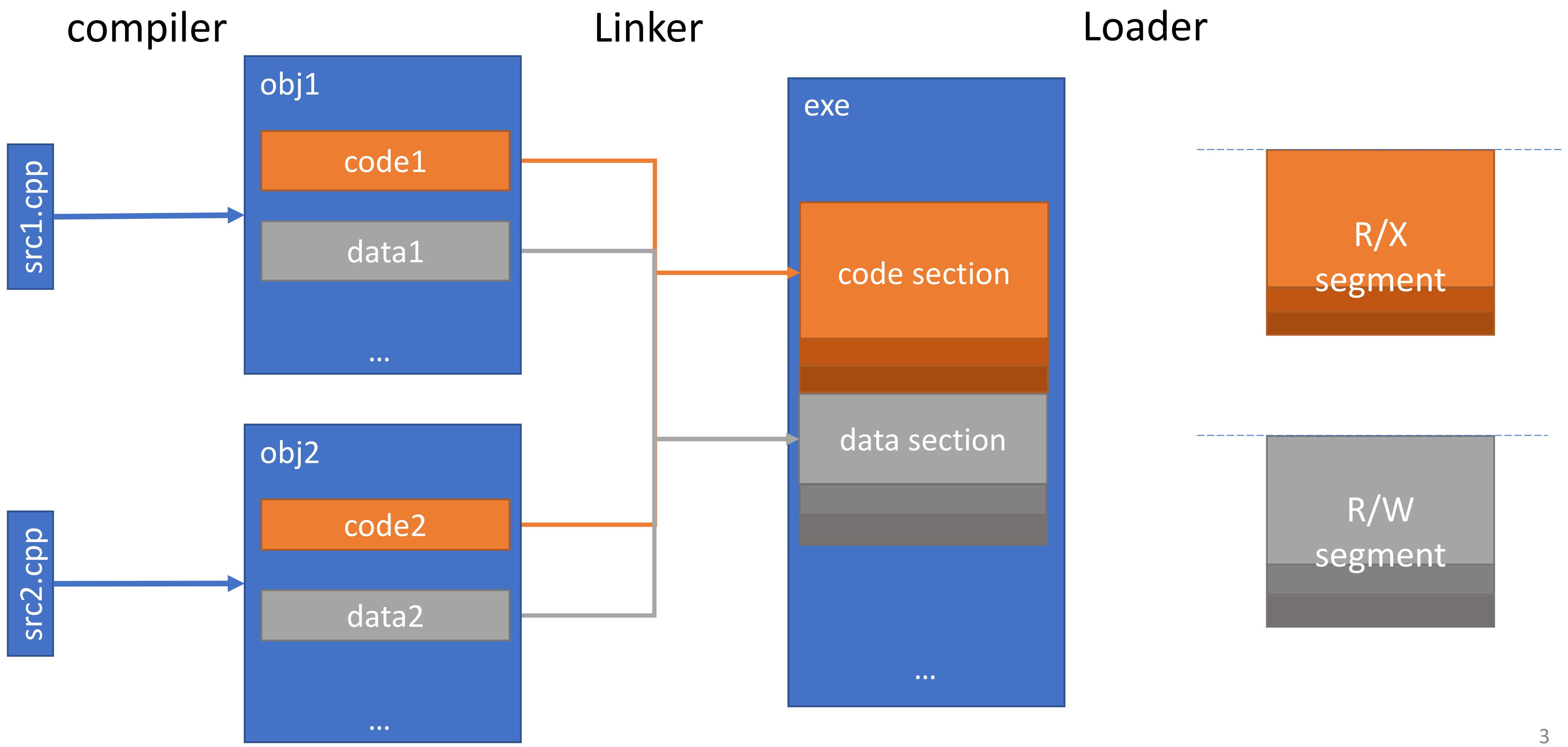
Ofek Shilon

Senior Developer
@Toga Networks
(a Huawei Company)

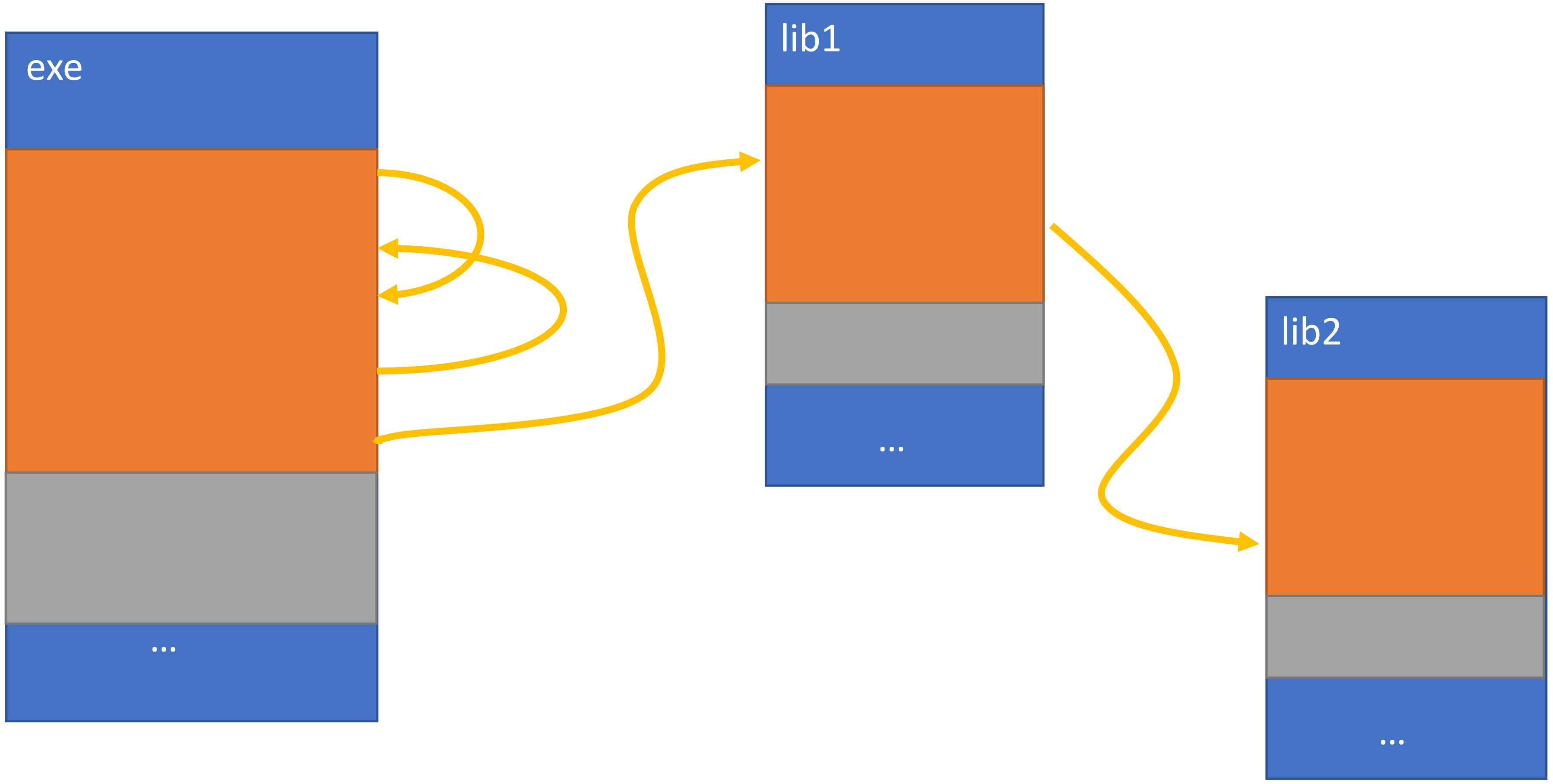
ofekshilon@gmail.com

OfekShilon

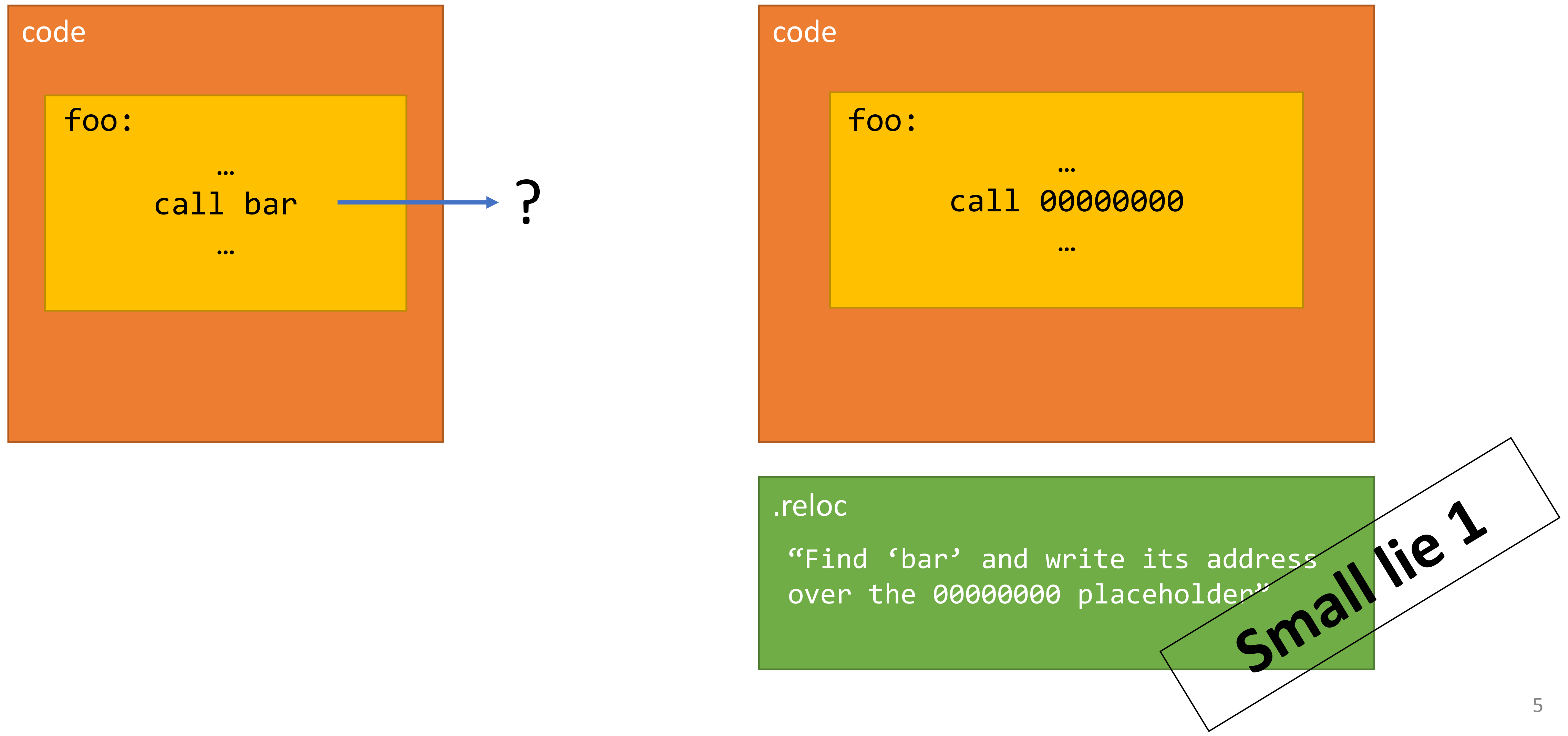
Intro to Linking in 3 slides, #1



Intro to Linking in 3 slides, #2



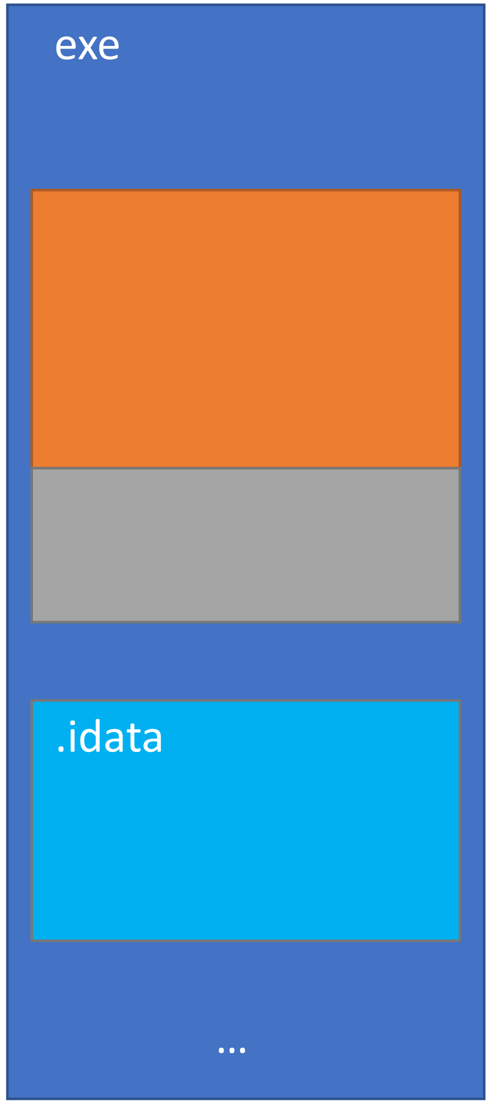
Intro to Linking in 3 slides, #3



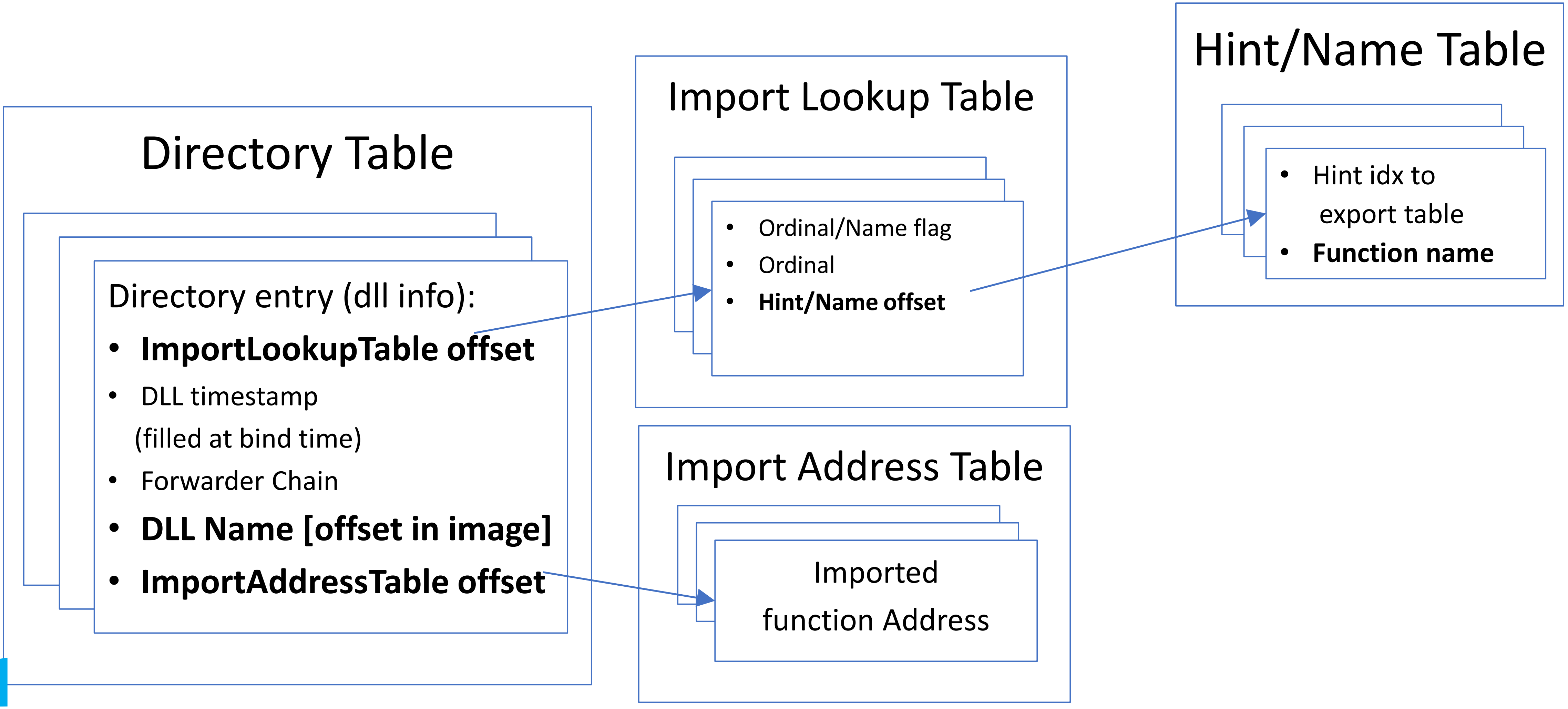
“Find ‘bar’ and write its address over the 00000000 placeholder”

Windows

Import data section



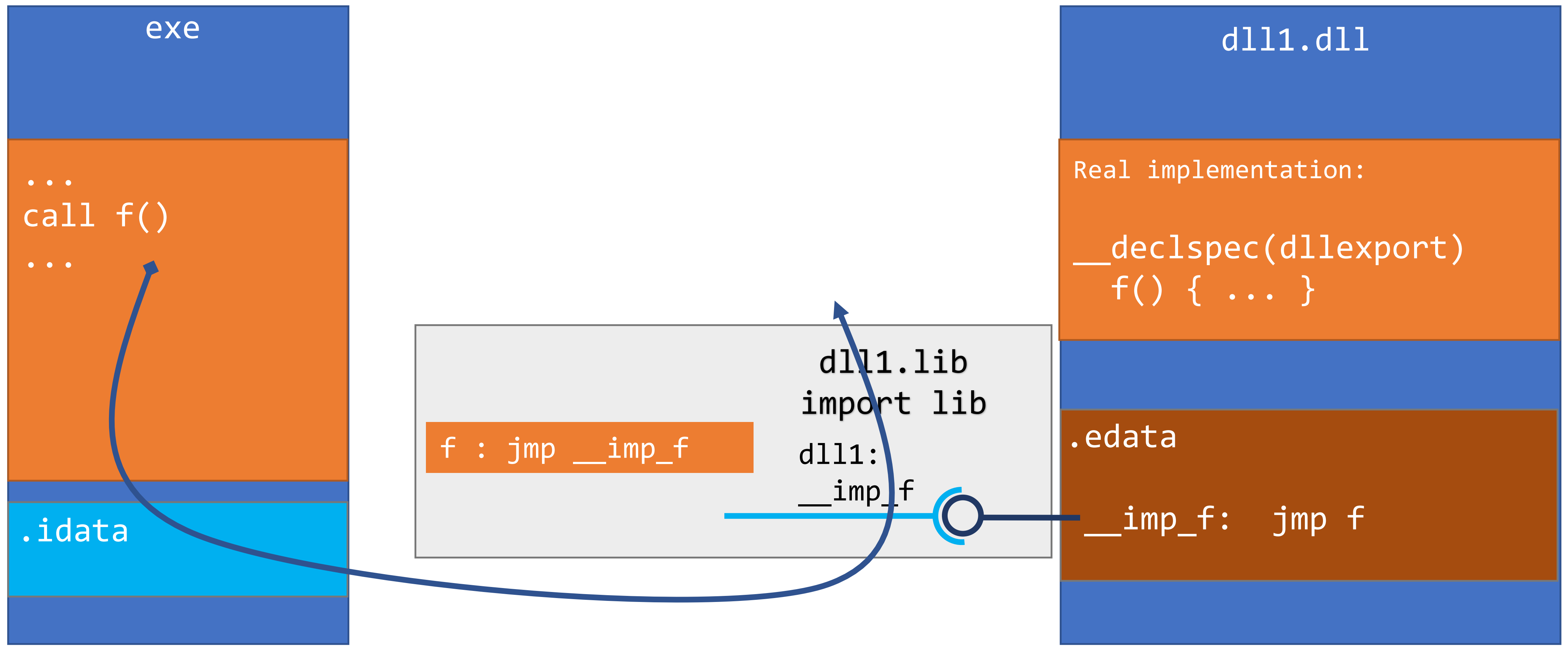
.idata section layout



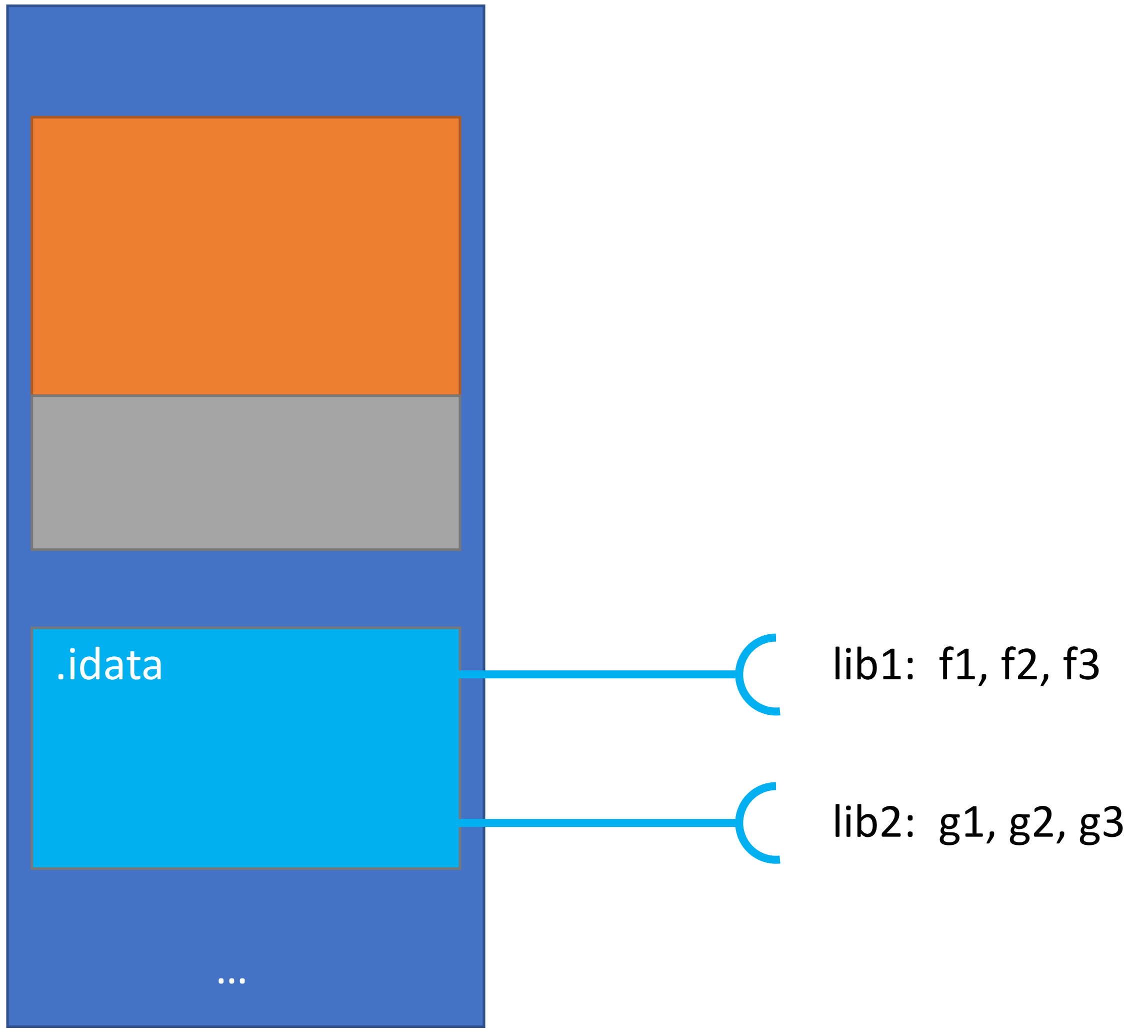
Directory Table
Directory entry (dll info):
• **ImportLookupTable offset**
• DLL timestamp (filled at bind time)
• Forwarder Chain
• **DLL Name [offset in image]**
• **ImportAddressTable offset**
Import Lookup Table
• Ordinal/Name flag
• Ordinal
• **Hint/Name offset**
Hint/Name Table
• Hint idx to export table
• **Function name**
Import Address Table
Imported function Address

https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#the-idata-section

Import Library



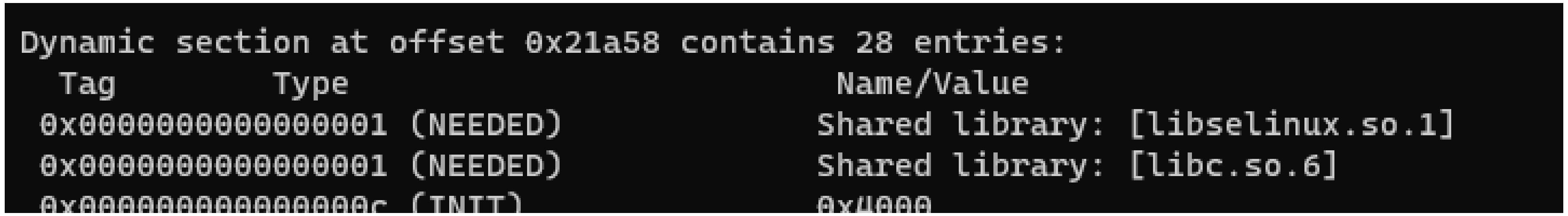
Windows Schematic Interface



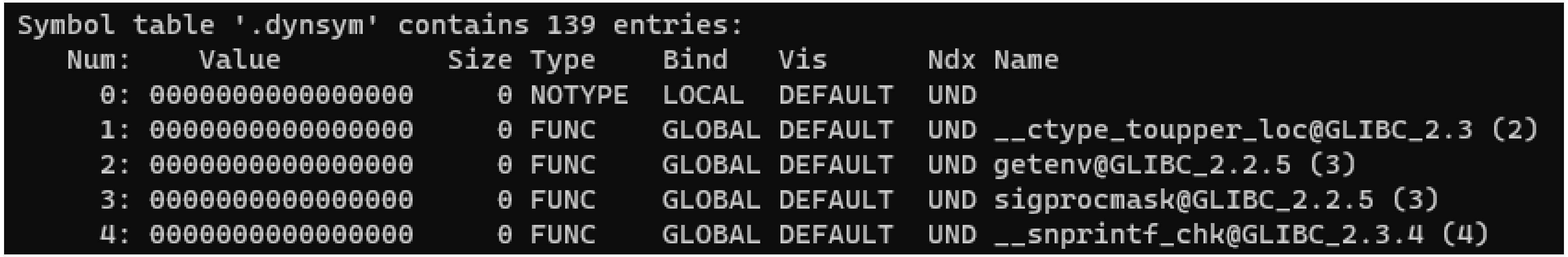
Linux

Linux import sections

- .dynamic /.dynsym: separate buckets of lib names and symbol names

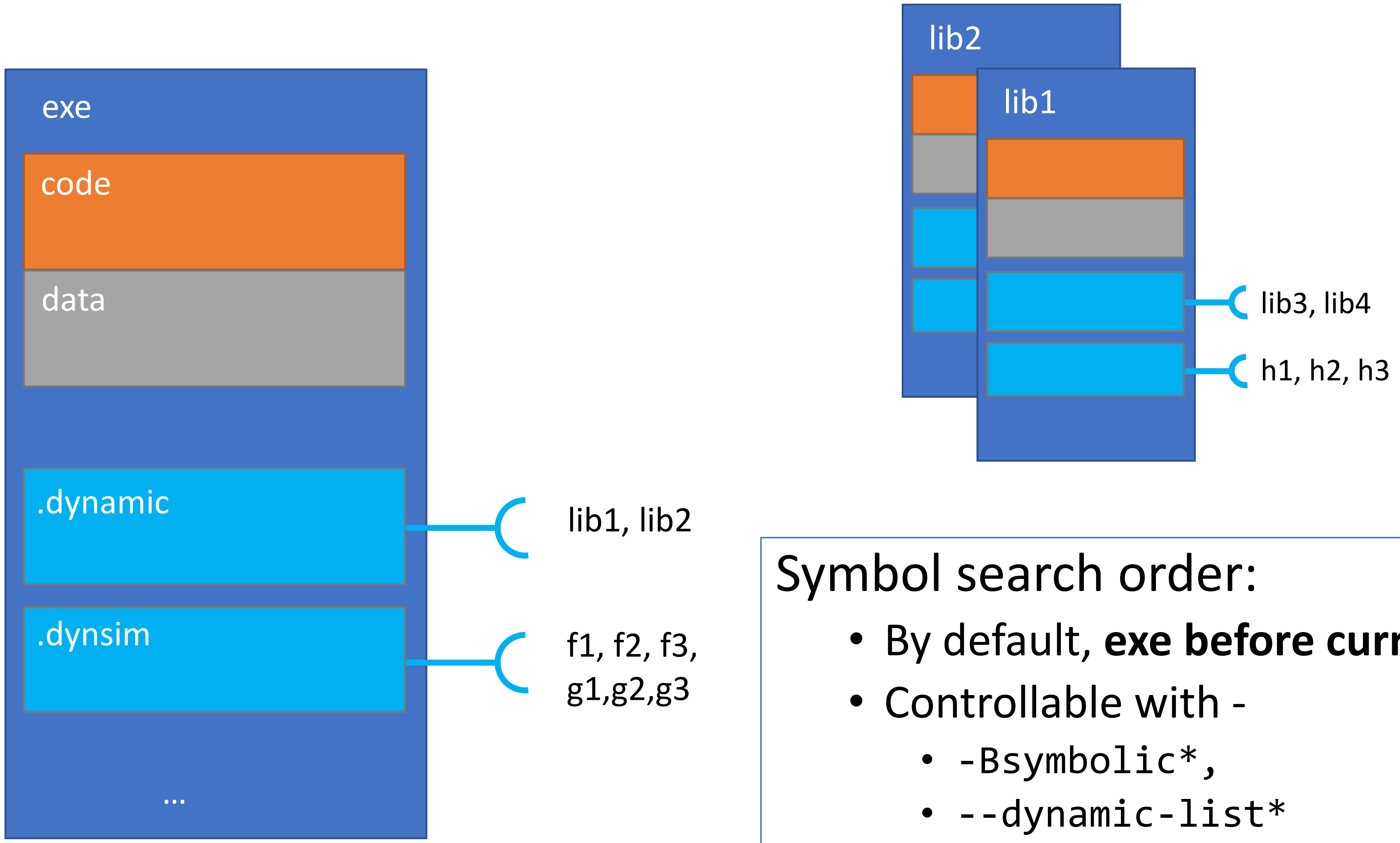


```
Dynamic section at offset 0x21a58 contains 28 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libselinux.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x4000
```



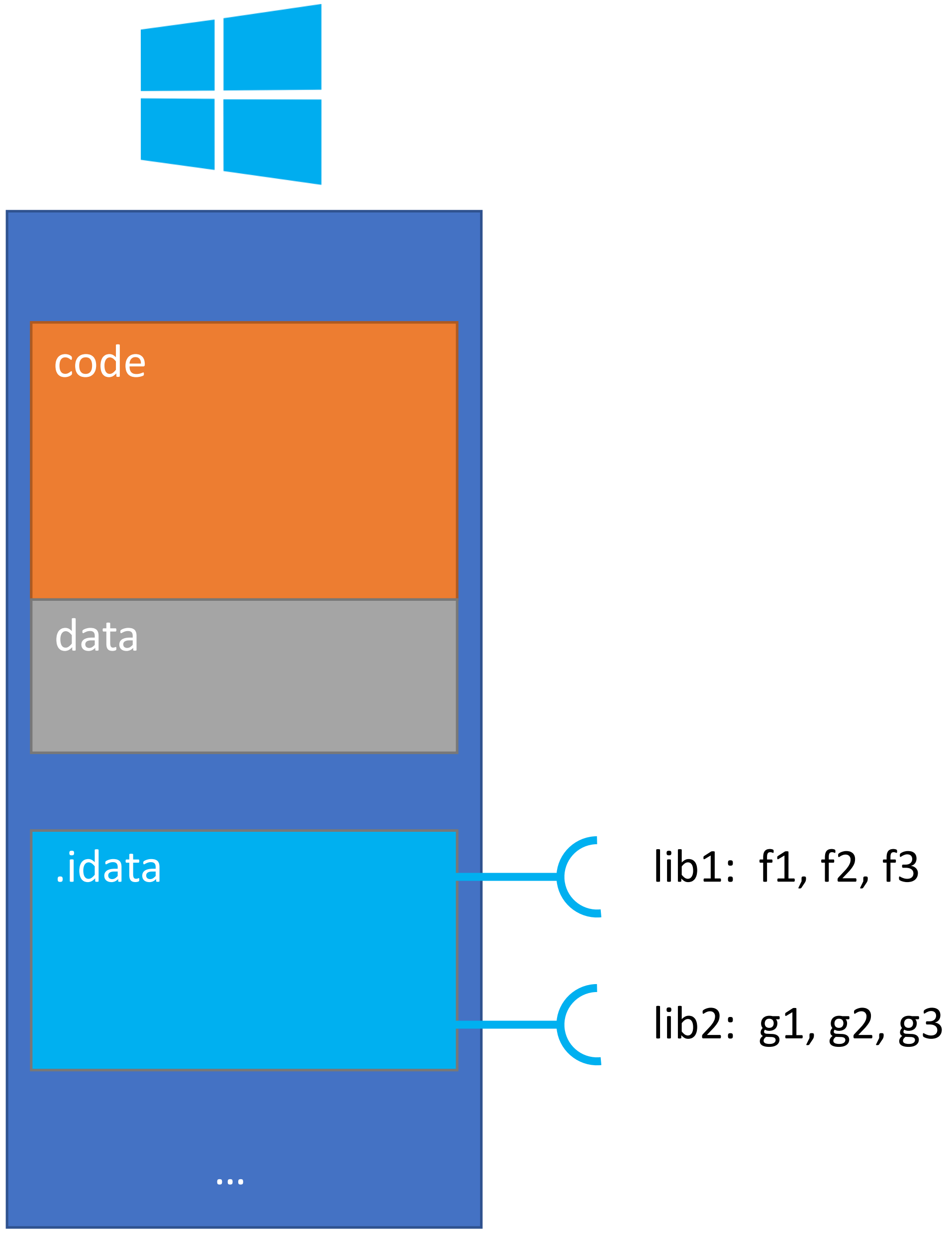
```
Symbol table '.dynsym' contains 139 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __ctype_toupper_loc@GLIBC_2.3 (2)
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND getenv@GLIBC_2.2.5 (3)
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND sigprocmask@GLIBC_2.2.5 (3)
     4: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __snprintf_chk@GLIBC_2.3.4 (4)
```

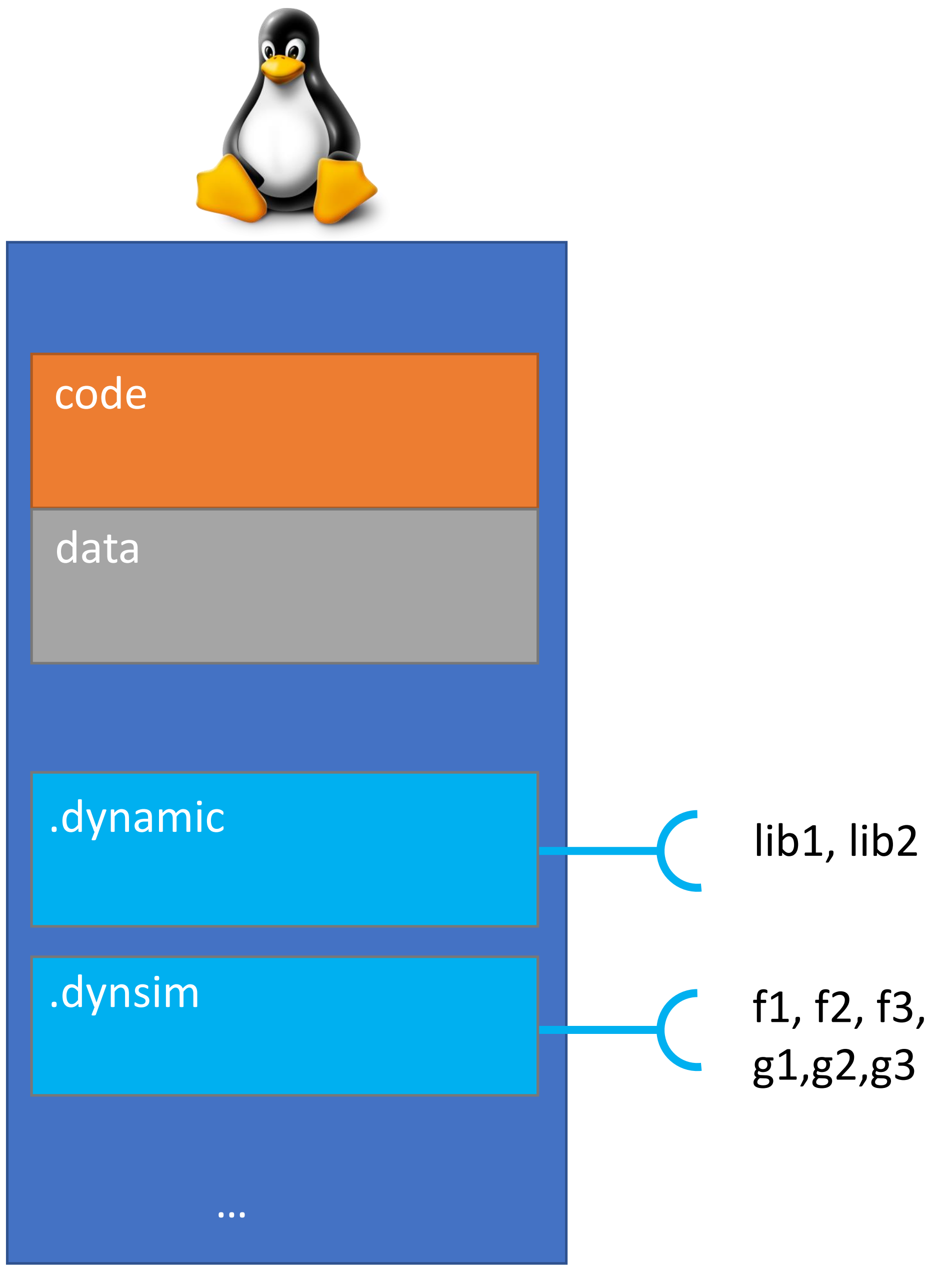
Linux Schematic Interface



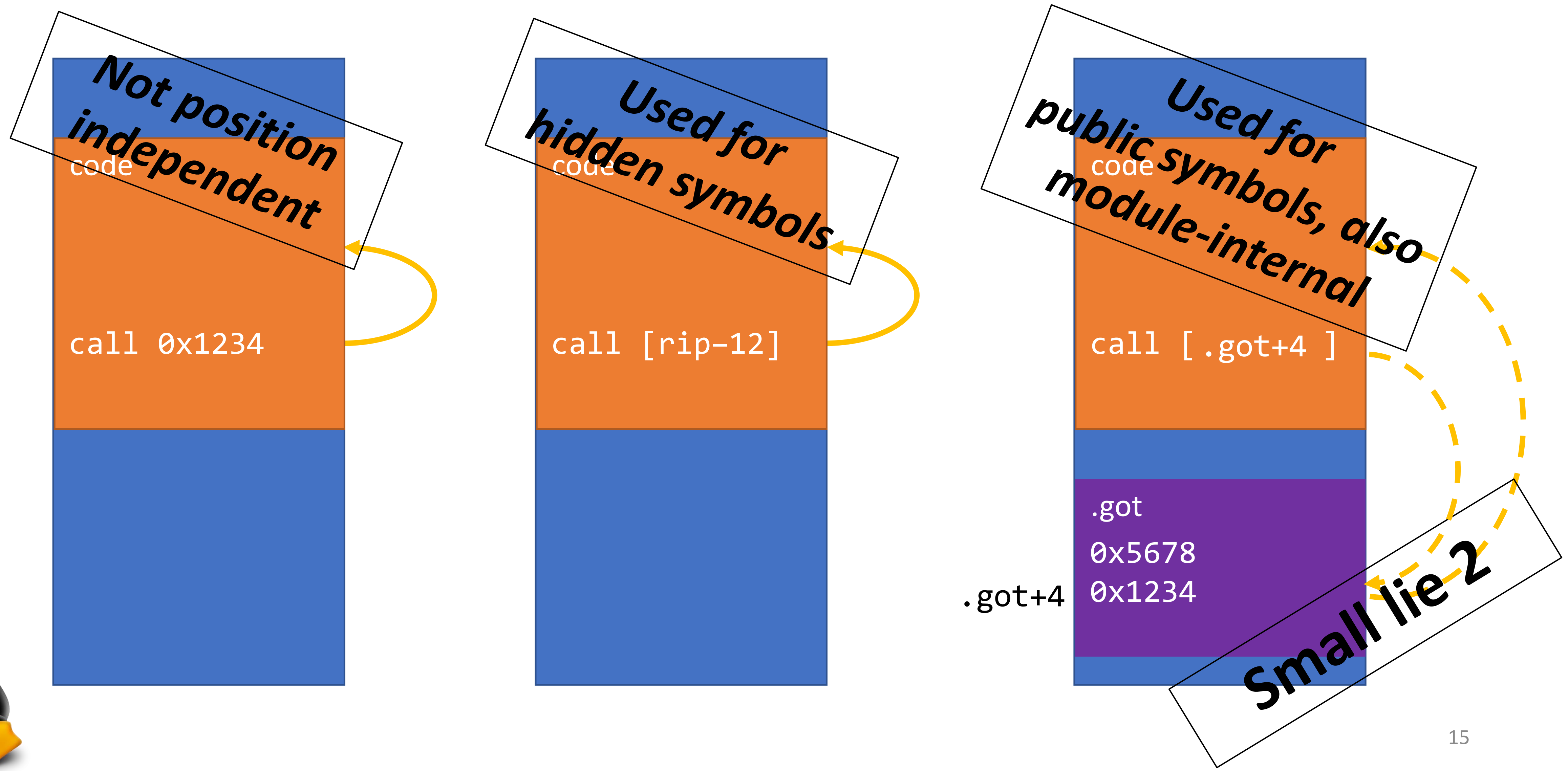
Symbol search order:

- By default, **exe before current lib**
- Controllable with -
  - `-Bsymbolic*,`
  - `--dynamic-list*`


code
data
.idata
lib1: f1, f2, f3
lib2: g1, g2, g3
...


code
data
.dynamic
.dynsim
lib1, lib2
f1, f2, f3,
g1,g2,g3
...

Position Independent Code



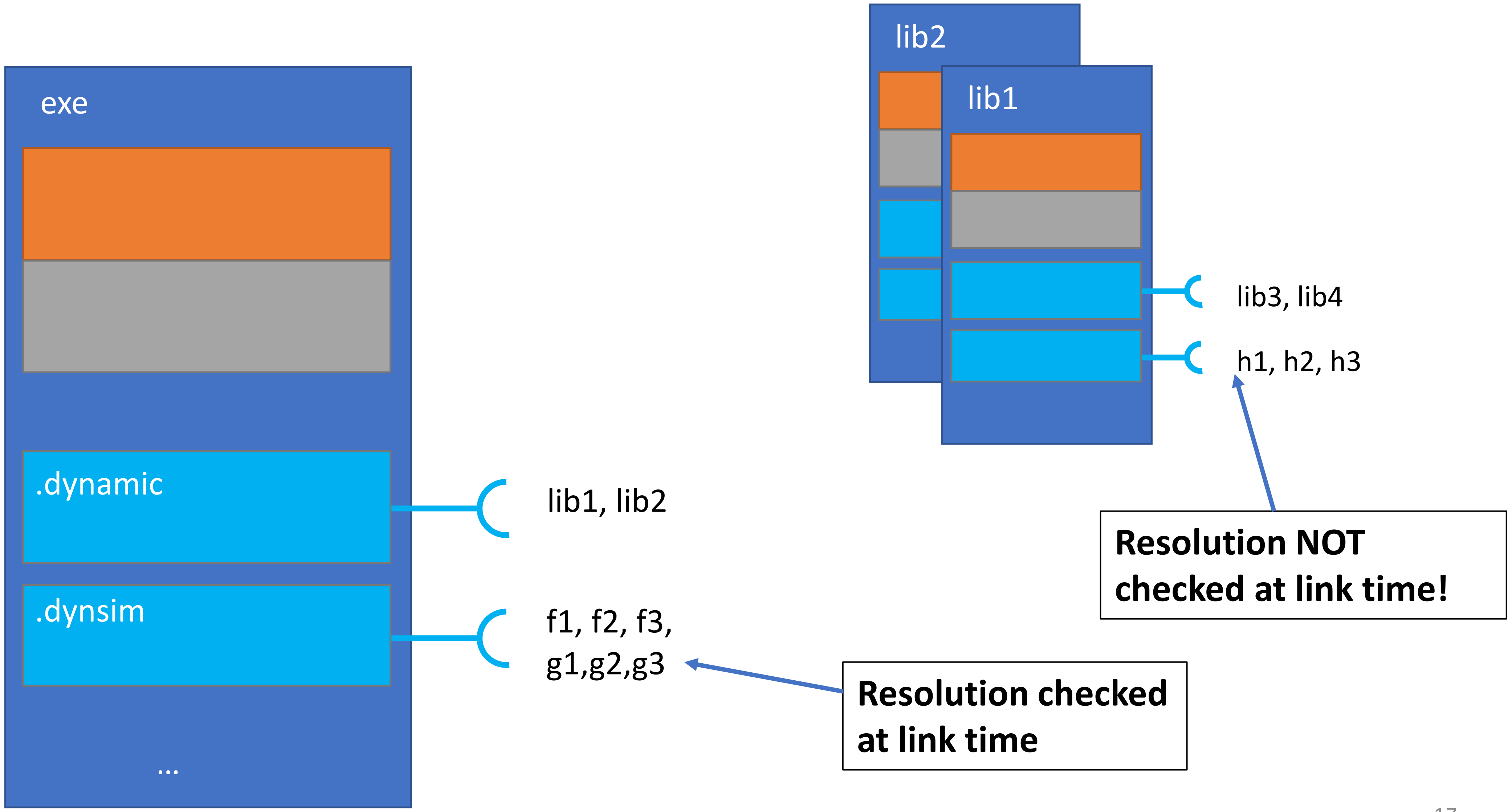
Position Independent Code - switches

- ***ALL*** shared lib code must be position independent
- Yet `-fPIC` is optional , not even implicit for `-shared`.
- If you try to link a shared lib from obj files not built with fPIC *and* using global vars:

```
error: relocation R_X86_64_PC32 against symbol `global' can not be
used when making a shared object; recompile with -fPIC
```

- -fpic vs -fPIC:
  - Help bypass built-in GOT size limit on some architectures (alpha, sparc).
- -fPIC vs –fPIE:
  - http://www.openbsd.org/papers/nycbsdcon08-pie/mgp00004.html

Resolution Time



Resolution NOT checked at link time!

Resolution checked at link time

Resolution Time

- Default: `--allow-shlib-undefined`
- Can be controlled with –
  - --no-allow-shlib-undefined
    - Note: operates recursively on ld, not on gold / lld.
  - `-z defs /--no-undefined :` forces link time resolution check

Linux: Intermediate Summary

- Symbol and Library dependencies are maintained separately
- By default ***all*** calls are indirected through `.got`
- By default, resolution in libraries (essentially populating the .got in every library) is deferred to load time, **and** the executable is searched first.

C++ Implication #1: How to form a process-wide singleton?

- (variable or function)
- Linux:
  - Just put it in the executable
- Windows:
  - Re-link the EXE and all DLLs against the single DLL that defines the singleton.

C++ Implication #2: Can you have circular library dependencies?

- Linux:
  - Yes
- Windows
  - No.
  - Well, you’d have to hack hard.

- The Linux design provides some flexibility, but ..
  - “This [allowed-shlib-undefined] is an unfortunate default for -shared. Changing it may be disruptive today. Mach-O and PE/COFF have many problems but this may be a place where they got right.”
    Rui Ueyama, author of LLD and MOLD linkers
    https://maskray.me/blog/2021-06-13-dependency-related-linker-options

C++ Implication #3: Can a shared-library symbol be overridden from an executable?

- “Interposition”
- Windows:
  - No.
  - Well –
    - Not from the executable. You’d have to re-link all components against a dll implementing the symbol to be overridden.
- Linux:
  - Yes.
    - Non-default build switches can intervene.

C++: `new`

[replacement.functions]: A C++ program may provide the definition for any of the following dynamic memory allocation function signatures declared in header <new> :

- operator new(std::size_t)
- operator new(std::size_t, std::align_val_t)
- …

The program's definitions are used instead of the default versions supplied by the implementation …

- On Windows, that's not what happens.

Lazy Binding

(a.k.a Delayed Loading)

Lazy Bind by Default?

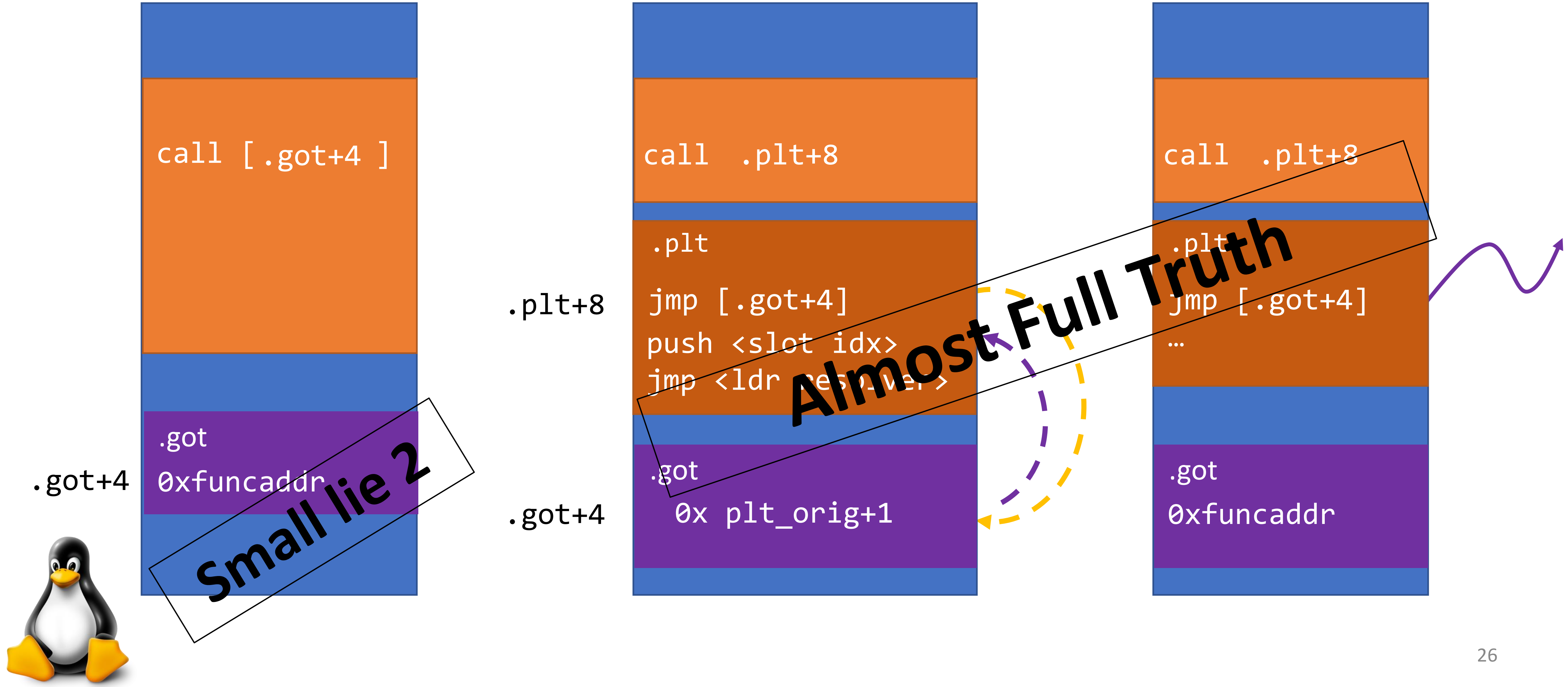
Yes.

Controllable with the env var
LD_BIND_NOW

No.

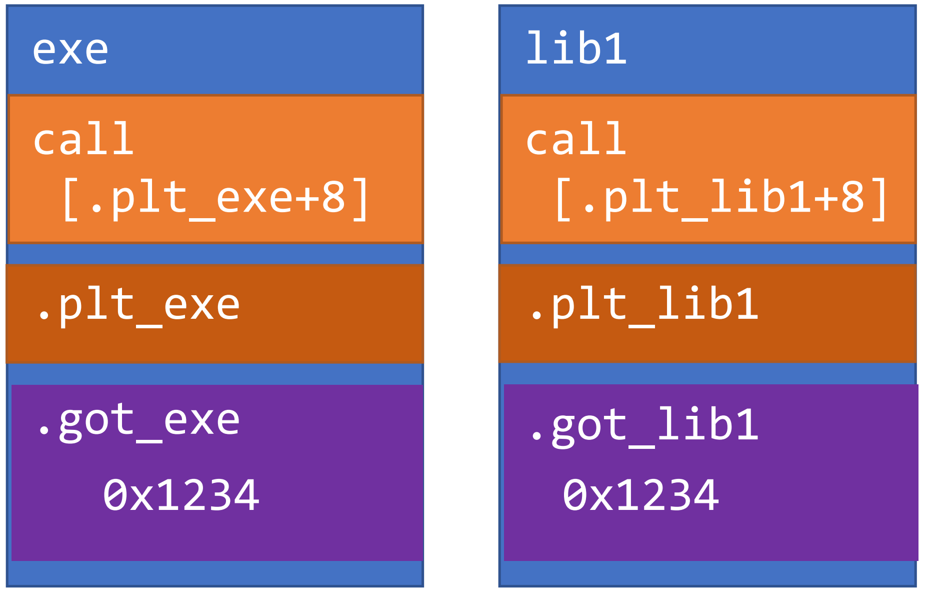
Controllable with linker switch
/DELAYLOAD:<your_dll.dll>

Procedure Linkage Table (PLT)



C++ Implication #4: Comparing Func Ptrs

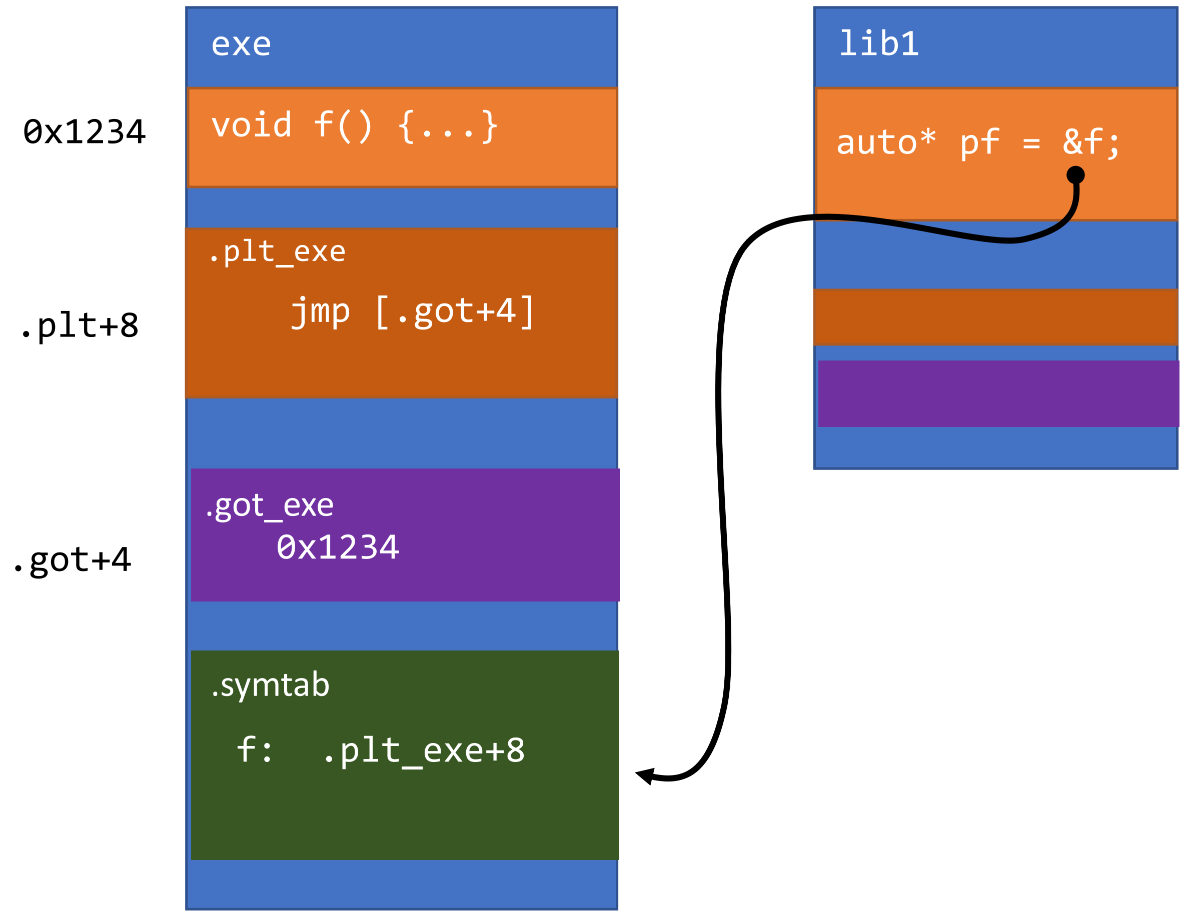
- C++ standard, [expr.eq]§3.2: "… if the pointers are both null, **both point to the same function**, or both represent the same address (6.8.2), they compare equal."
- Actual calls are made to a PLT entry.
- Different among libraries!



C++ Implication #4: Comparing Func Ptrs

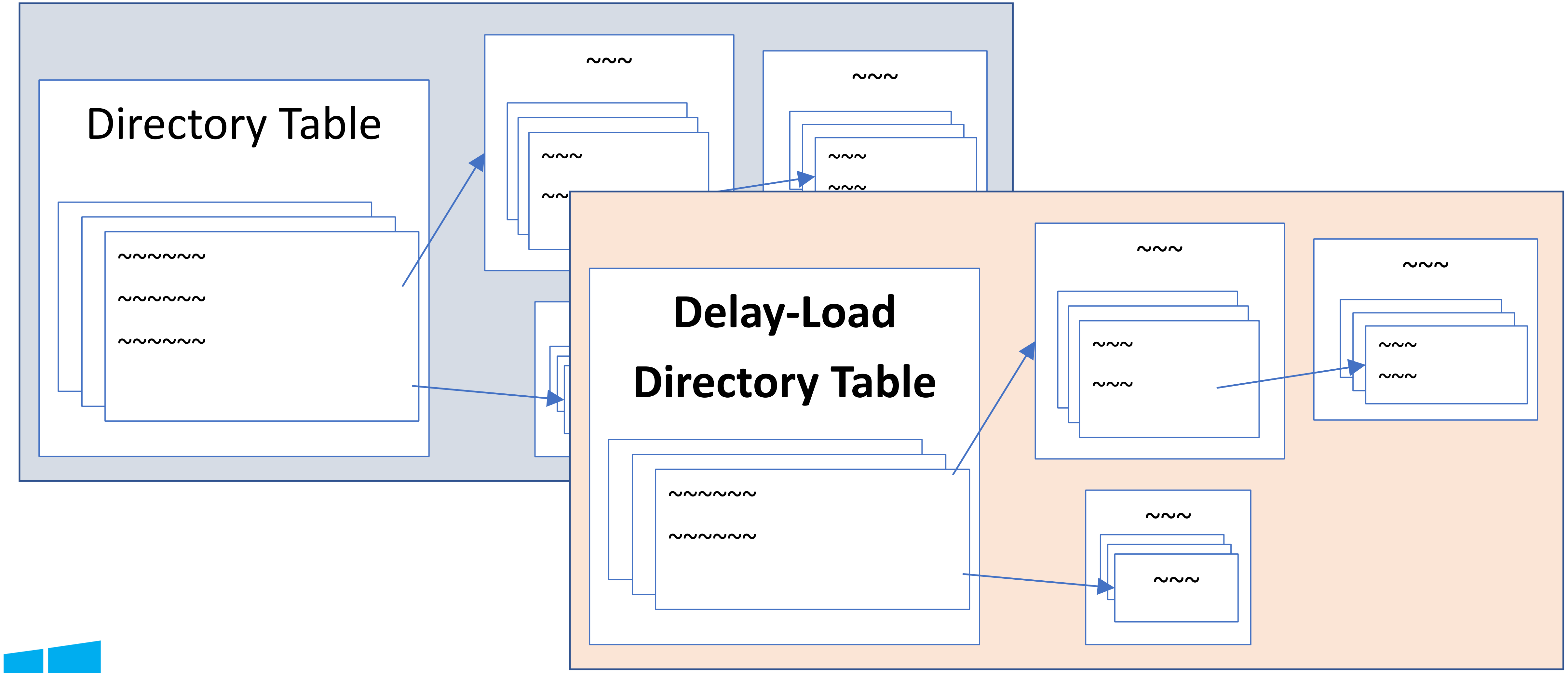
From the SystemV ABI:

To allow comparisons of function addresses to work as expected, if an executable file references a function defined in a shared object, the link editor will place the address of the procedure linkage table entry for that function in its associated symbol table entry. This will result in symbol table entries with section index of SHN_UNDEF but a type of STT_FUNC and a non-zero st_value. A reference to the address of a function from within a shared library will be satisfied by such a definition in the executable



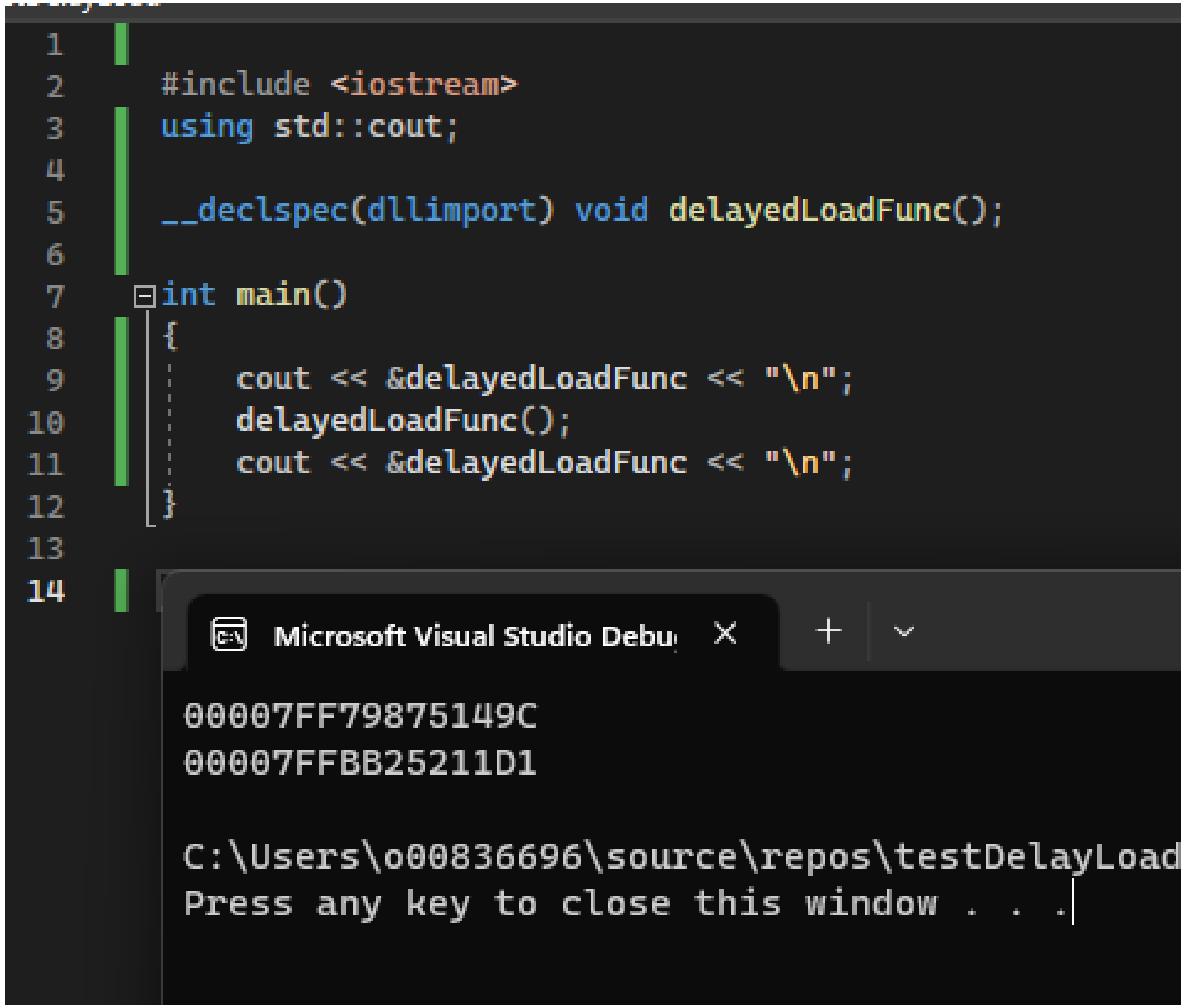
https://gitlab.com/x86-psABIs/x86-64-ABI/-/jobs/artifacts/master/raw/x86-64-ABI/abi.pdf?job=build

Windows .idata section


Directory Table
~~~~~~
~~~~~~
~~~~~~
~~~
~~~
~~~
Delay-Load
Directory Table
~~~~~~
~~~~~~
~~~
~~~
~~~
~~~
~~~
~~~
~~~
~~~

https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#the-idata-section

No analogue mechanism in Windows:



```cpp

#include <iostream>
using std::cout;

__declspec(dllimport) void delayedLoadFunc();

int main()
{
    cout << &delayedLoadFunc << "\n";
    delayedLoadFunc();
    cout << &delayedLoadFunc << "\n";
}


```

```
00007FF79875149C
00007FFBB25211D1

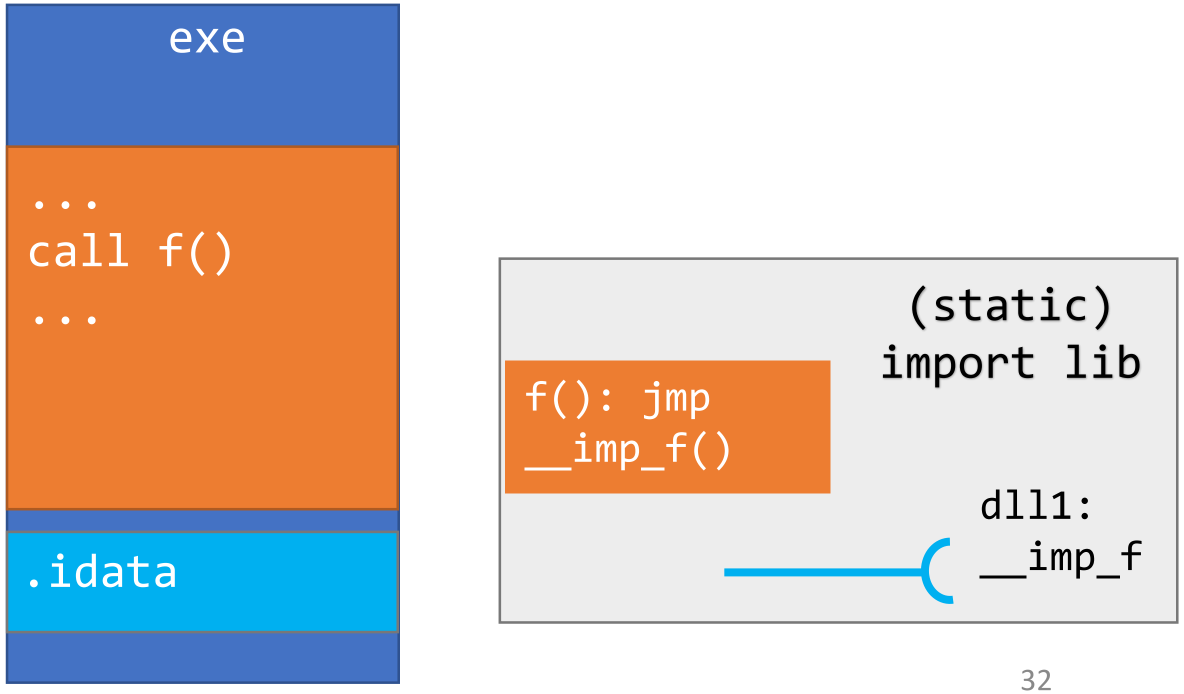
C:\Users\o00836696\source\repos\testDelayLoad
Press any key to close this window . . .
```

Symbol Visibility

Symbol Visibility - Windows

- `__declspec(export)` – add symbol to .edata
- `__declspec(import)` – doesn't do much…
  - Minor optimization that happens in Release anyway

- Most symbols are neither.



Symbol Visibility - Linux

- No import/export distinction! Only public/hidden
- When a symbol is public it –
  - Goes through the longer PLT/GOT route,
  - Is available to other libraries/executables (“exported”).
  - Potentially subject to interposition (“imported”), and allowed-undefined.
- ***Default visibility is public!***
  - Intervene through:
    - -fvisibility=hidden,
    - -fvisibility-inlines-hidden,
    - -fvisibility-ms-compat,
    - __attribute__ ((visibility ("hidden")))

Symbol Visibility - Linux

- “Using this feature [-fvisibility=hidden] can very substantially improve linking and load times of shared object libraries, produce more optimized code, provide near-perfect API export and prevent symbol clashes. It is strongly recommended that you use this in any shared objects you distribute.”

$ man gcc

- https://gcc.gnu.org/wiki/Visibility

Virtual functions overhead

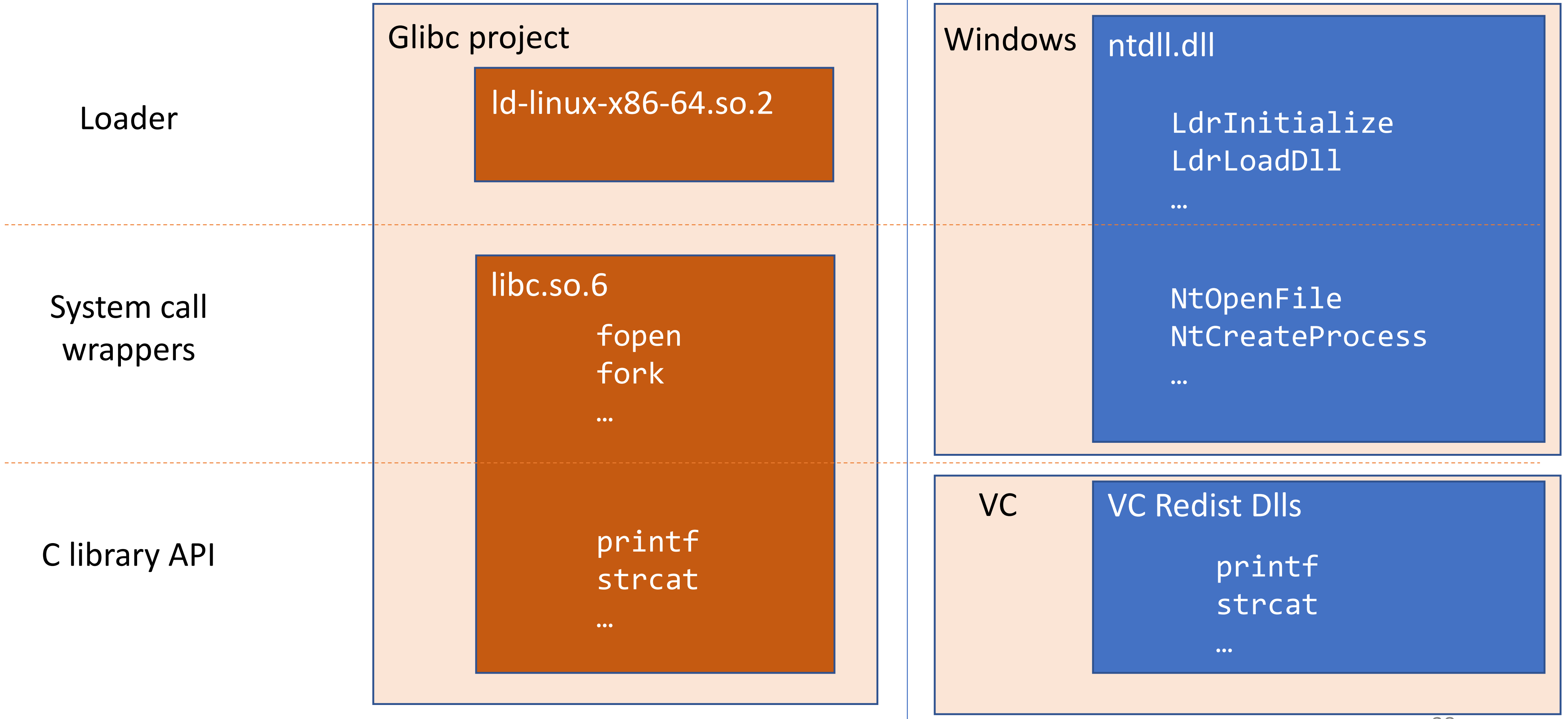
- Careful when measuring on linux…
- By default **ALL** calls are indirect calls, through the PLT/GOT.
  - In a library.
  - Sometimes in executables too (-fPIE)

Loader

Loader

- A.k.a Dynamic Linker, a.k.a Interpreter, a.k.a Image Loader (esp in windows)
- Runs in user mode – operates on regular process address space

Component Map



Selecting a different loader

```
$ gcc –v whatever.cpp
...
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/9/collect2 -plugin /usr/lib/gcc/x86_64-linux-gnu/9/liblto_plugin.so -
plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/9/lto-wrapper -plugin-opt=-fresolution=/tmp/ccfheLJQ.res -
plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s -plugin-opt=-pass-through=-lc -
plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s --build-id --eh-frame-hdr -m
elf_x86_64 --hash-style=gnu --as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie -z now -z
relro /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/Scrt1.o /usr/lib/gcc/x86_64-linux-
gnu/9/../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/9/crtbeginS.o -
L/usr/lib/gcc/x86_64-linux-gnu/9 -L/usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu -
L/usr/lib/gcc/x86_64-linux-gnu/9/../../../../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -
L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/9/../../..
/tmp/ccijfiIQ.o -lgcc --push-state --as-needed -lgcc_s --pop-state -lc -lgcc --push-state --as-
needed -lgcc_s --pop-state /usr/lib/gcc/x86_64-linux-gnu/9/crtendS.o /usr/lib/gcc/x86_64-linux-
gnu/9/../../../x86_64-linux-gnu/crtn.o
...
```

Selecting a different loader

```
$ readelf --program-headers /usr/bin/ls
...
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
...
```

Observing the Loader in action - Linux

```
$ LD_DEBUG=help cat
Valid options for the LD_DEBUG environment variable are:

  libs        display library search paths
  reloc       display relocation processing
  files       display progress for input file
  symbols     display symbol table processing
  bindings    display information about symbol binding
  versions    display version dependencies
  scopes      display scope information
  all         all previous options combined
  statistics  display relocation statistics
  unused      determined unused DSOs
  help        display this help message and exit
```

Observing the Loader in action - Windows

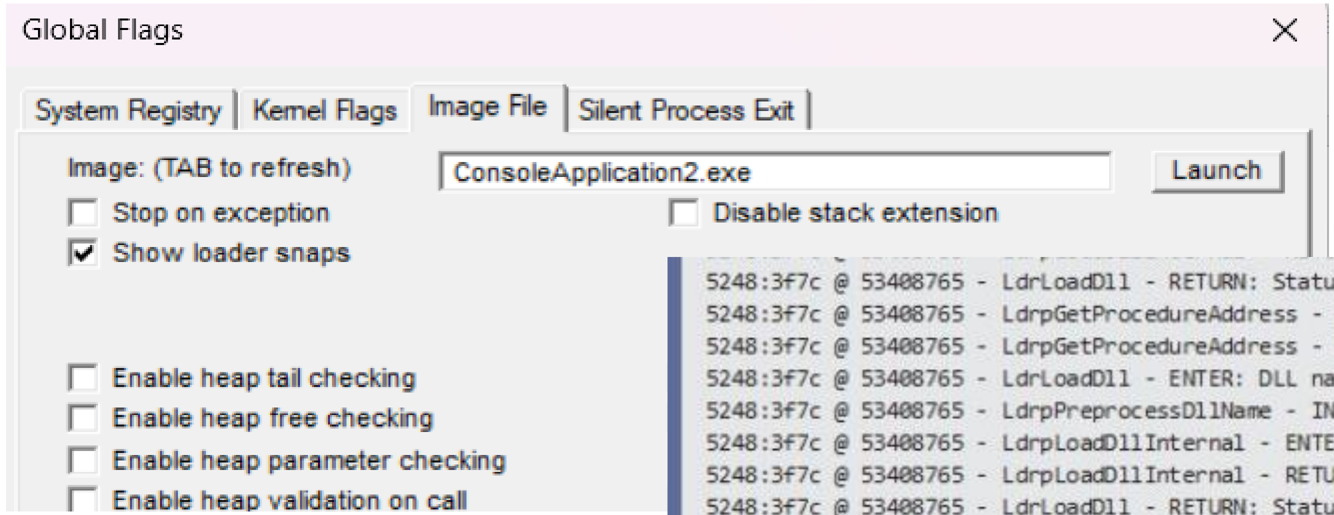
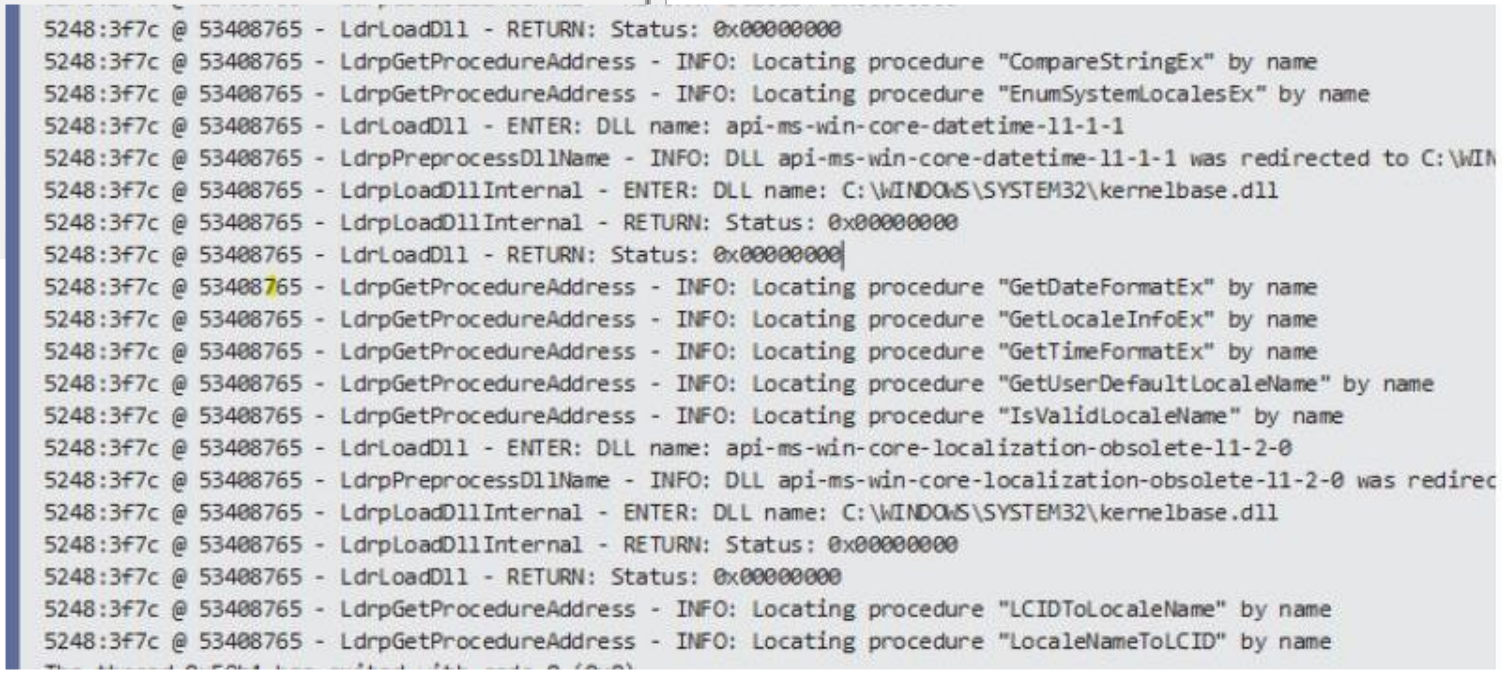


Image: (TAB to refresh) ConsoleApplication2.exe

- [ ] Stop on exception
- [ ] Disable stack extension
- [✓] Show loader snaps
- [ ] Enable heap tail checking
- [ ] Enable heap free checking
- [ ] Enable heap parameter checking
- [ ] Enable heap validation on call



```
5248:3f7c @ 53408765 - LdrLoadDll - RETURN: Status: 0x00000000
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "CompareStringEx" by name
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "EnumSystemLocalesEx" by name
5248:3f7c @ 53408765 - LdrLoadDll - ENTER: DLL name: api-ms-win-core-datetime-l1-1-1
5248:3f7c @ 53408765 - LdrpPreprocessDllName - INFO: DLL api-ms-win-core-datetime-l1-1-1 was redirected to C:\WIN
5248:3f7c @ 53408765 - LdrpLoadDllInternal - ENTER: DLL name: C:\WINDOWS\SYSTEM32\kernelbase.dll
5248:3f7c @ 53408765 - LdrpLoadDllInternal - RETURN: Status: 0x00000000
5248:3f7c @ 53408765 - LdrLoadDll - RETURN: Status: 0x00000000
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "GetDateFormatEx" by name
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "GetLocaleInfoEx" by name
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "GetTimeFormatEx" by name
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "GetUserDefaultLocaleName" by name
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "IsValidLocaleName" by name
5248:3f7c @ 53408765 - LdrLoadDll - ENTER: DLL name: api-ms-win-core-localization-obsolete-l1-2-0
5248:3f7c @ 53408765 - LdrpPreprocessDllName - INFO: DLL api-ms-win-core-localization-obsolete-l1-2-0 was redirec
5248:3f7c @ 53408765 - LdrpLoadDllInternal - ENTER: DLL name: C:\WINDOWS\SYSTEM32\kernelbase.dll
5248:3f7c @ 53408765 - LdrpLoadDllInternal - RETURN: Status: 0x00000000
5248:3f7c @ 53408765 - LdrLoadDll - RETURN: Status: 0x00000000
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "LCIDToLocaleName" by name
5248:3f7c @ 53408765 - LdrpGetProcedureAddress - INFO: Locating procedure "LocaleNameToLCID" by name
```

https://ofekshilon.com/2013/06/22/entry-point-not-found-and-other-dll-loading-problems/

C++ and Shared Libs

C++ and Shared Libs

[replacement.functions]: A C++ program may provide the definition for any of the following dynamic memory allocation function signatures declared in header <new> :

- operator new(std::size_t)
- operator new(std::size_t, std::align_val_t)
- …

The program’s definitions are used instead of the default versions supplied by the implementation

- “Shared libraries are out of scope for the standard”
- “ ‘Program’ and ‘implementation’ are undefined”

C++ and Shared Libs

- These clauses are dead letter.
  - Pragmatic suggestion: drop altogether statements that will never be followed.
    - replacement-function
    - func-ptr comparison
    - … ?
  - *The only way to make the standard applicable to real world programs.*

More Rabbit Holes for the Curious

- Relocation details
- Weak linkage
- Versioning (both of libraries and of API)
- Granularity
    - COMDAT, -ffunction-sections, -fdata-sections
- Optimizations
    - Identical Code Folding: /OPT:ICF, -icf=all, -icf=safe
    - Dead Code Elimination: /OPT:REF, -fvtable-gc, --gc-section
- Linker scripts

Resources

- Ulrich Drepper: “How to write shared libraries”
  - http://library.bagrintsev.me/CPP/dsohowto.pdf
- Eli Benderski:
  - https://eli.thegreenplace.net/2011/08/25/load-time-relocation-of-shared-libraries/
  - https://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-libraries/
- Rui Ueyama, author of LLD and MOLD linkers:
  - https://maskray.me/blog/
- Ian Lance Taylor, author of GOLD:
  - https://lwn.net/Articles/276782/
- John Levine, “Linkers and Loaders” book:
  - https://www.amazon.com/Linkers-Kaufmann-Software-Engineering-Programming/dp/1558604960
- Michael Kerrisk, Online Training:
  - https://www.man7.org/training/shlib/index.html