COLLABORA

libcapsule - Segregated Dynamic Linking

Vivek Das Mohapatra

Collabora

November 25, 2017

Open First

libcapsule - Segregated Dynamic Linking

Vivek Das Mohapatra

Collabora

November 25, 2017

libcapsule - Segregated Dynamic Linking

└─libcapsule - Segregated Dynamic Linking

libcapsule - Segregated Dynamic Linking

Introduction

The Pieces of the Puzzle

These Yaks Aren't Shaving Themselves

And Finally

COLLABORA

The Problem

- Applications ‘Containerised’
- Libraries come from a runtime

Open First

The Problem

- Applications ‘Containerised’
- Libraries come from a runtime

runtime makes promises about API/ABI/versions mesa tied to hw, no reasonable way for runtime to stay current

The Problem

- Applications ‘Containerised’
- Libraries *mostly* come from a runtime
- ...but *some* still need to come from the host

The Problem

- Applications ‘Containerised’
- Libraries *mostly* come from a runtime
- ...but *some* still need to come from the host

runtime makes promises about API/ABI/versions mesa tied to hw, no reasonable way for runtime to stay current

COLLABORA

The Problem

- Applications 'Containerised'
- Libraries *mostly* come from a runtime
- ...but *some* still need to come from the host
  - notably mesa (libGL & friends)

The Problem

- Applications 'Containerised'
- Libraries *mostly* come from a runtime
- ...but *some* still need to come from the host
  - notably mesa (libGL & friends)

runtime makes promises about API/ABI/versions mesa tied to hw, no reasonable way for runtime to stay current

COLLABORA

The Problem

- Applications 'Containerised'
- Libraries *mostly* come from a runtime
- ...but *some* still need to come from the host
  - notably mesa (libGL & friends)
- host libraries may have incompatible dependencies

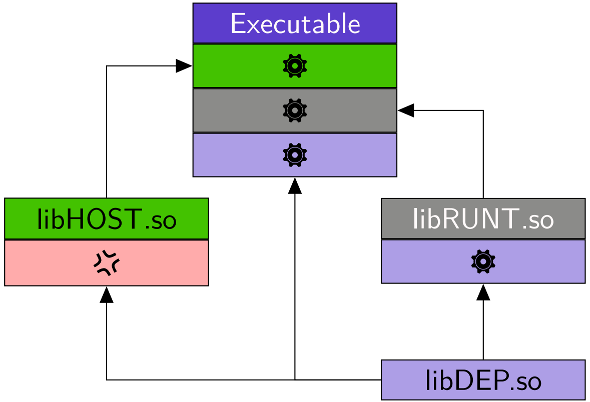
The Problem

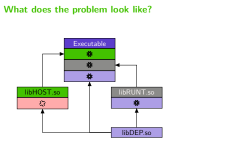
- Applications 'Containerised'
- Libraries *mostly* come from a runtime
- ...but *some* still need to come from the host
  - notably mesa (libGL & friends)
- host libraries may have incompatible dependencies

runtime makes promises about API/ABI/versions mesa tied to hw, no reasonable way for runtime to stay current

COLLABORA

What does the problem look like?

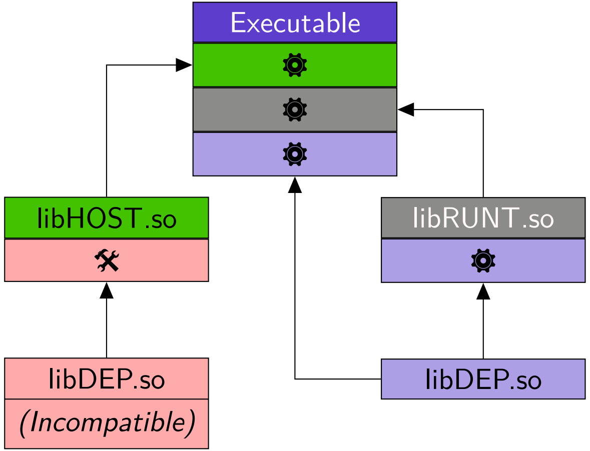




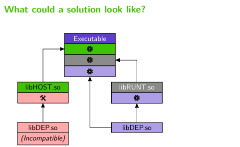
The linker uses sonames to decide if a library meets our requirements *but* sometimes we end up with incompatible libraries with the same soname... and it only allows one copy of the same soname in any link chain

COLLABORA

What could a solution look like?



Open First



We can see here two incompatible versions of libDEP from host and runtime: only libHOST sees the host version (and it does not see the runtime version)

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us

Objectives
- Expose only the library we want to isolate
  - its dependencies *not* exposed to us

What do we mean by minimal intervention?

In order of preference:

- Purely runtime isolation mechanism
- Some compilation required but basically automatic
- Manual intervention required to generate the isolating ‘thing’

COLLABORA

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes

What do we mean by minimal intervention?

In order of preference:

- Purely runtime isolation mechanism
- Some compilation required but basically automatic
- Manual intervention required to generate the isolating 'thing'

COLLABORA

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation

What do we mean by minimal intervention?

In order of preference:

- Purely runtime isolation mechanism
- Some compilation required but basically automatic
- Manual intervention required to generate the isolating ‘thing’

COLLABORA

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation
- No performance hit

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation
- No performance hit

What do we mean by minimal intervention?

In order of preference:

- Purely runtime isolation mechanism
- Some compilation required but basically automatic
- Manual intervention required to generate the isolating ‘thing’

COLLABORA

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation
- No performance hit
- Transparent use of nonstandard search-path

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation
- No performance hit
- Transparent use of nonstandard search-path

What do we mean by minimal intervention?

In order of preference:

- Purely runtime isolation mechanism
- Some compilation required but basically automatic
- Manual intervention required to generate the isolating ‘thing’

COLLABORA

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation
- No performance hit
- Transparent use of nonstandard search-path
- Minimal manual intervention

Objectives

- Expose only the library we want to isolate
  - its dependencies *not* exposed to us
- No code changes
- No recompilation
- No performance hit
- Transparent use of nonstandard search-path
- Minimal manual intervention

What do we mean by minimal intervention?

In order of preference:

- Purely runtime isolation mechanism
- Some compilation required but basically automatic
- Manual intervention required to generate the isolating ‘thing’

COLLABORA

libcapsule - Segregated Dynamic Linking

└─libcapsule - Segregated Dynamic Linking

libcapsule - Segregated Dynamic Linking

Introduction

The Pieces of the Puzzle

These Yaks Aren't Shaving Themselves

And Finally

1: Private Dependencies

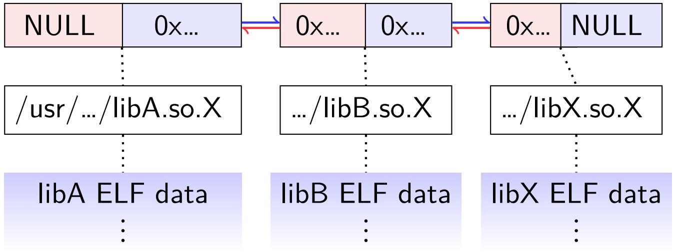
- Library should have isolated dependencies

1: Private Dependencies
- Library should have isolated dependencies

COLLABORA

1: Private Dependencies

- Library should have isolated dependencies
- Normally all dependencies in a single linked list



1: Private Dependencies
Library should have isolated dependencies
Normally all dependencies in a single linked list

1: Private Dependencies

- Library should have isolated dependencies
- Normally all dependencies in a single linked list
- So how do we do this?

1: Private Dependencies

- Library should have isolated dependencies
- Normally all dependencies in a single linked list
- So how do we do this?

COLLABORA

dlmopen()

- *cf* dlopen() but *can* create a new link map
- ... or can add a new entry to an existing link map
- more-or-less workable from at least glibc 2.19

dlmopen()

- *cf* dlopen() but *can* create a new link map
- ... or can add a new entry to an existing link map
- more-or-less workable from at least glibc 2.19

COLLABORA

2: Picking the right library versions

- dlmopen() automatically loads dependencies

2: Picking the right library versions

- dlmopen() automatically loads dependencies

COLLABORA

2: Picking the right library versions

- dlmopen() automatically loads dependencies
- Libraries are picked from the search path

2: Picking the right library versions

- dlmopen() automatically loads dependencies
- Libraries are picked from the search path

COLLABORA

2: Picking the right library versions

- dlmopen() automatically loads dependencies
- Libraries are picked from the search path
- First match wins

2: Picking the right library versions

- dlmopen() automatically loads dependencies
- Libraries are picked from the search path
- First match wins

2: Picking the right library versions

- dlmopen() automatically loads dependencies
- Libraries are picked from the search path
- First match wins
- We need to subvert this for isolated libraries

2: Picking the right library versions

- dlmopen() automatically loads dependencies
- Libraries are picked from the search path
- First match wins
- We need to subvert this for isolated libraries

COLLABORA

Controlling the link map

- Linker loads all listed dependencies

Controlling the link map

- Linker loads all listed dependencies

Effectively we populat the link map by hand - by doing dependency resolution by hand we prevent the linke's automatic searching from kicking in: A classic convenience *vs* control trade-off

COLLABORA

Controlling the link map

- Linker loads all listed dependencies
- The linker *won't* reload items already in the map

Open First

Controlling the link map
- Linker loads all listed dependencies
- The linker *won't* reload items already in the map

Effectively we populat the link map by hand - by doing dependency resolution by hand we prevent the linke's automatic searching from kicking in: A classic convenience *vs* control trade-off

Controlling the link map

- Linker loads all listed dependencies
- The linker *won't* reload items already in the map
  - Loading libraries explicitly
  - In reverse dependency order
  - We can control exactly what gets linked

Controlling the link map

- Linker loads all listed dependencies
- The linker *won't* reload items already in the map
  - Loading libraries explicitly
  - In reverse dependency order
  - We can control exactly what gets linked

Effectively we populat the link map by hand - by doing dependency resolution by hand we prevent the linke's automatic searching from kicking in: A classic convenience *vs* control trade-off

COLLABORA

3: Automatically exposing symbols

- Isolated the dlmopen()ed symbols completely

3: Automatically exposing symbols

- Isolated the dlmopen()ed symbols completely

3: Automatically exposing symbols

- Isolated the dlmopen()ed symbols completely
- Need callers to get to them automatically

3: Automatically exposing symbols

- Isolated the dlmopen()ed symbols completely
- Need callers to get to them automatically

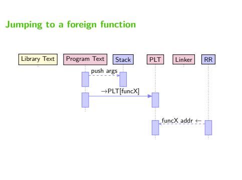
COLLABORA

3: Automatically exposing symbols

- Isolated the dlmopen()ed symbols completely
- Need callers to get to them automatically
- Need to understand dynamic library calls

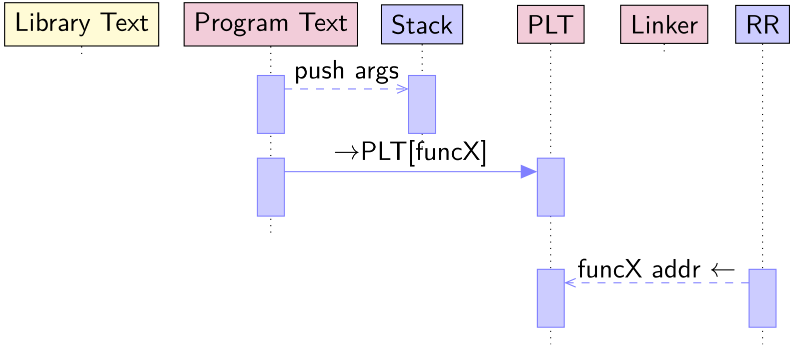
3: Automatically exposing symbols

- Isolated the dlmopen()ed symbols completely
- Need callers to get to them automatically
- Need to understand dynamic library calls



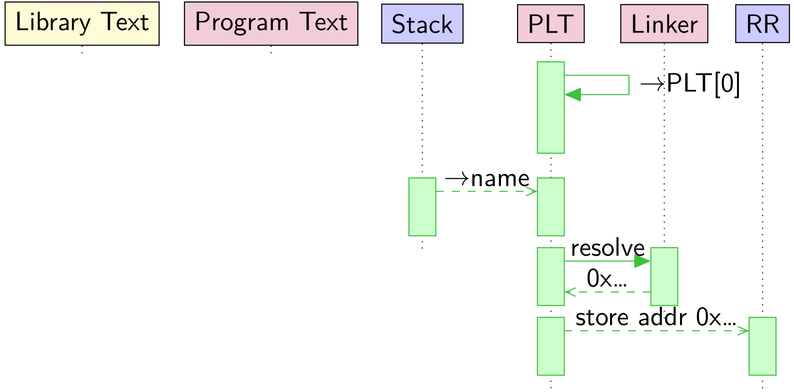
COLLABORA

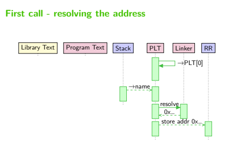
Jumping to a foreign function



- Calling code puts foreign function arguments on the stack
- Excution jumps to a fixed offset in the PLT (specific to this function)
- The PLT stub looks up the corresponding address in the RR and jumps to it

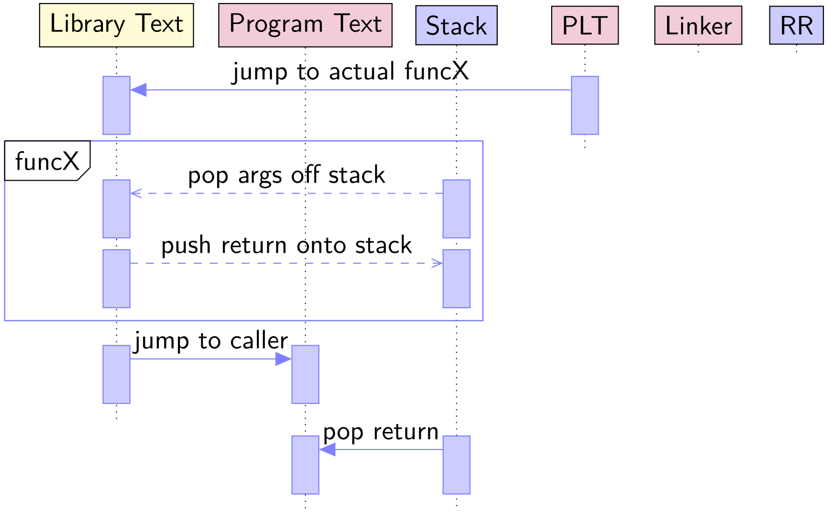
First call - resolving the address

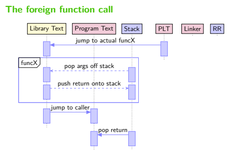




- — The fixup code pointed at by the RR asks the linker for the real address
- — The linker searches the calling DSO dependencies for the symbol
- — The fixup code writes the address into the RR slot
- — The fixup code jumps to the address in the RR slot

The foreign function call





- Jump to funcX
- The code in the foreign DSO pulls the arguments off the stack
- Function does whatver it does
- The return value is pushed onto the stack
- Execution jumps back to the caller

COLLABORA

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:

COLLABORA

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address

Control the RR, Control the call...
- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address

COLLABORA

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?

COLLABORA

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?
  - Yes — we can!

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?
  - Yes — we can!

COLLABORA

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?
  - Yes — we can!
  - The link map → to ELF data for each library

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?
  - Yes — we can!
  - The link map → to ELF data for each library

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?
  - Yes — we can!
  - The link map → to ELF data for each library
  - libelf can interrogate this

Control the RR, Control the call...

- If we scribble on the RR slot before the first call:
  - The PLT fixup will never be invoked
  - The linker never resolves the symbol's address
  - Total control over where the function call goes
- Key question — can we find the RR?
  - Yes — we can!
  - The link map → to ELF data for each library
  - libelf can interrogate this

COLLABORA

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path

Needs the list of exported symbols, but not their signatures
Needs to know its target's soname
Otherwise fully automated

COLLABORA

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:

Needs the list of exported symbols, but not their signatures
Needs to know its target's soname
Otherwise fully automated

COLLABORA

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies

Needs the list of exported symbols, but not their signatures
Needs to know its target's soname
Otherwise fully automated

COLLABORA

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies
    - Do this in reverse dependency order

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies
    - Do this in reverse dependency order

Needs the list of exported symbols, but not their signatures
Needs to know its target's soname
Otherwise fully automated

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies
    - Do this in reverse dependency order
    - Search the alternate library path

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies
    - Do this in reverse dependency order
    - Search the alternate library path

Needs the list of exported symbols, but not their signatures
Needs to know its target's soname
Otherwise fully automated

COLLABORA

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies
    - Do this in reverse dependency order
    - Search the alternate library path
  - Walk the link map & scribble on all the RRs

Putting the Pieces Together

- Make a shim library with the target's **soname**
- Put the shim on the search path
- During the shim's init:
  - dlmopen() the real library and its dependencies
    - Do this in reverse dependency order
    - Search the alternate library path
  - Walk the link map & scribble on all the RRs

Needs the list of exported symbols, but not their signatures

Needs to know its target's soname

Otherwise fully automated

libcapsule - Segregated Dynamic Linking

Open First

libcapsule - Segregated Dynamic Linking

Introduction

The Pieces of the Puzzle

These Yaks Aren't Shaving Themselves

And Finally

Some Terminology

- Call an isolated set of libraries a *capsule*
- Assume they come from an fs mounted at **/host**

Some Terminology

- Call an isolated set of libraries a *capsule*
- Assume they come from an fs mounted at **/host**

dlopen() in capsules

- dlopen() can't be called from inside a capsule

dlopen() in capsules

- dlopen() can't be called from inside a capsule

COLLABORA

dlopen() in capsules

- dlopen() can't be called from inside a capsule
  - replace capsule's dlopen with a wrapper that calls dlmopen()
  - remap paths to **/host** in the wrapper
  - dlmopen() doesn't accept RTLD_GLOBAL

dlopen() in capsules

- dlopen() can't be called from inside a capsule
  - replace capsule's dlopen with a wrapper that calls dlmopen()
  - remap paths to **/host** in the wrapper
  - dlmopen() doesn't accept RTLD_GLOBAL

COLLABORA

dlsym() now has a split personality

- dlsym() outside the capsule has to do extra work

Open First

dlsym() now has a split personality

- dlsym() outside the capsule has to do extra work

dlsym() now has a split personality

- dlsym() outside the capsule has to do extra work
  - Pretend that two separate dl handles are the same

dlsym() now has a split personality

- dlsym() outside the capsule has to do extra work
  - Pretend that two separate dl handles are the same

dlsym() now has a split personality

- dlsym() outside the capsule has to do extra work
  - Pretend that two separate dl handles are the same
  - Do this when we scribble on the RRs

dlsym() now has a split personality

- dlsym() outside the capsule has to do extra work
  - Pretend that two separate dl handles are the same
  - Do this when we scribble on the RRs

dlopen() outside capsules

- dlopen() outside capsule must trigger RR scribbling

dlopen() outside capsules

- dlopen() outside capsule must trigger RR scribbling

COLLABORA

dlopen() outside capsules

- dlopen() outside capsule must trigger RR scribbling
  - replace external dlopen with a wrapper that does this

dlopen() outside capsules

- dlopen() outside capsule must trigger RR scribbling
  - replace external dlopen with a wrapper that does this

COLLABORA

dlopen() outside capsules

- dlopen() outside capsule must trigger RR scribbling
  - replace external dlopen with a wrapper that does this
  - Again — when we scribble on the RRs

dlopen() outside capsules

- dlopen() outside capsule must trigger RR scribbling
  - replace external dlopen with a wrapper that does this
  - Again — when we scribble on the RRs

Extra Problems

- *alloc()/free() pairing

Extra Problems

- *alloc()/free() pairing

COLLABORA

Extra Problems

- *alloc()/free() pairing
  - Propose RTLD_*SOMETHING*

Extra Problems

- *alloc()/free() pairing
  - Propose RTLD_*SOMETHING*

COLLABORA

Extra Problems

- *alloc()/free() pairing
  - Propose RTLD_*SOMETHING*
  - For now, replace the *alloc/free cluster inside the capsule

Extra Problems

- *alloc()/free() pairing
  - Propose RTLD_*SOMETHING*
  - For now, replace the *alloc/free cluster inside the capsule

COLLABORA

libcapsule - Segregated Dynamic Linking

libcapsule - Segregated Dynamic Linking
Introduction
The Pieces of the Puzzle
These Yaks Aren't Shaving Themselves
And Finally

drumroll

- Does it *actually* work?

drumroll

- Does it *actually* work?

drumroll

- Does it *actually* work?
- Yes!

drumroll

- Does it *actually* work?
- Yes!

COLLABORA

drumroll

- Does it *actually* work?
- Yes!
  - glxinfo et al
  - openarena (SDL 1 & 2)
  - Dungeon Defenders (SDL 2)
  - And a Unity game whose name I forget...

drumroll

- Does it *actually* work?
- Yes!
  - glxinfo et al
  - openarena (SDL 1 & 2)
  - Dungeon Defenders (SDL 2)
  - And a Unity game whose name I forget...

COLLABORA

Any Questions... ?

Open First

Any Questions... ?