HARMEN STOPPELS / FOSDEM 23

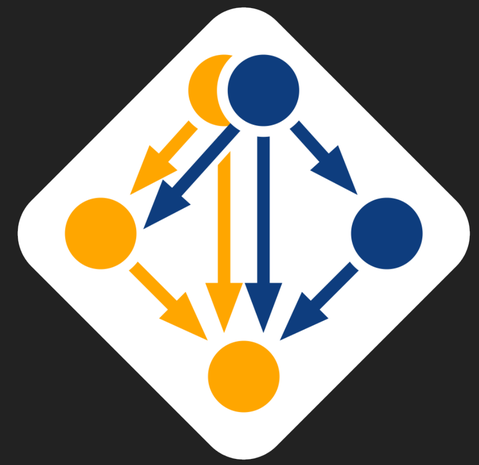
TAMING THE STAT STORM IN SPACK

WHAT IS THE STAT STORM AND WHY SHOULD IT BE TAMED?

- Term "coined" by Guix: https://guix.gnu.org/blog/2021/taming-the-stat-storm-with-a-loader-cache/
- Ingredients for the problem:
  1. A package manager that installs each package in its own prefix (Nix/Guix/Spack)
  2. A loader/interpreter that has to locate dependencies at application startup
  3. Your average slow, shared filesystem

WHAT IS SPACK?

- Flexible package manager primarily for HPC
- No root privileges required, build on top of your distro
- Supports installing multiple flavors of the same package
  - Versions, variants, dependencies, ...
- Powerful dependency solver
- Package recipes are written in Python
  - `depends_on("python@3.7:", when="@2: +python")`



CONCRETIZATION IN SPACK

```
$ spack spec fftw precision=float,double +mpi ^mpich@:3
Input spec
--------------------------------
fftw+mpi precision=double,float
    ^mpich@:3

Concretized
--------------------------------
fftw@3.3.10%gcc@7.5.0+mpi~openmp ... precision=double,float arch=linux-sles15-zen
    ^mpich@3.4.3%gcc@7.5.0~argobots~cuda ... patches=7326028 pmi=pmi arch=linux-sles15-zen
        ^findutils@4.9.0%gcc@7.5.0 patches=440b954 arch=linux-sles15-zen
        ^hwloc@2.9.0%gcc@7.5.0~cairo~cuda~gl ... libs=shared,static arch=linux-sles15-zen
            ^ncurses@6.4%gcc@7.5.0~symlinks+termlib abi=none arch=linux-sles15-zen
        ^libfabric@1.16.1%gcc@7.5.0~debug~kdreg fabrics=sockets,tcp,udp arch=linux-sles15-z
        ^libpciaccess@0.16%gcc@7.5.0 arch=linux-sles15-zen
            ^libtool@2.4.7%gcc@7.5.0 arch=linux-sles15-zen
                ^m4@1.4.19%gcc@7.5.0+sigsegv patches=9dc5fbd,bfdffa7 arch=linux-sles15-zen
                    ^diffutils@3.8%gcc@7.5.0 arch=linux-sles15-zen
                    ^libsigsegv@2.13%gcc@7.5.0 arch=linux-sles15-zen
```

WHERE DOES SPACK INSTALL PACKAGES

- Every package is installed in a unique directory
- Directory name contains a hash derived from the DAG
- Intentionally non-FHS* (root-level /bin, /lib, /etc) compliant

```
~/my-packages/fftw-3.3.10-5nnbgjggppj5rf2ilvtb6bmcqkbm2mze
```

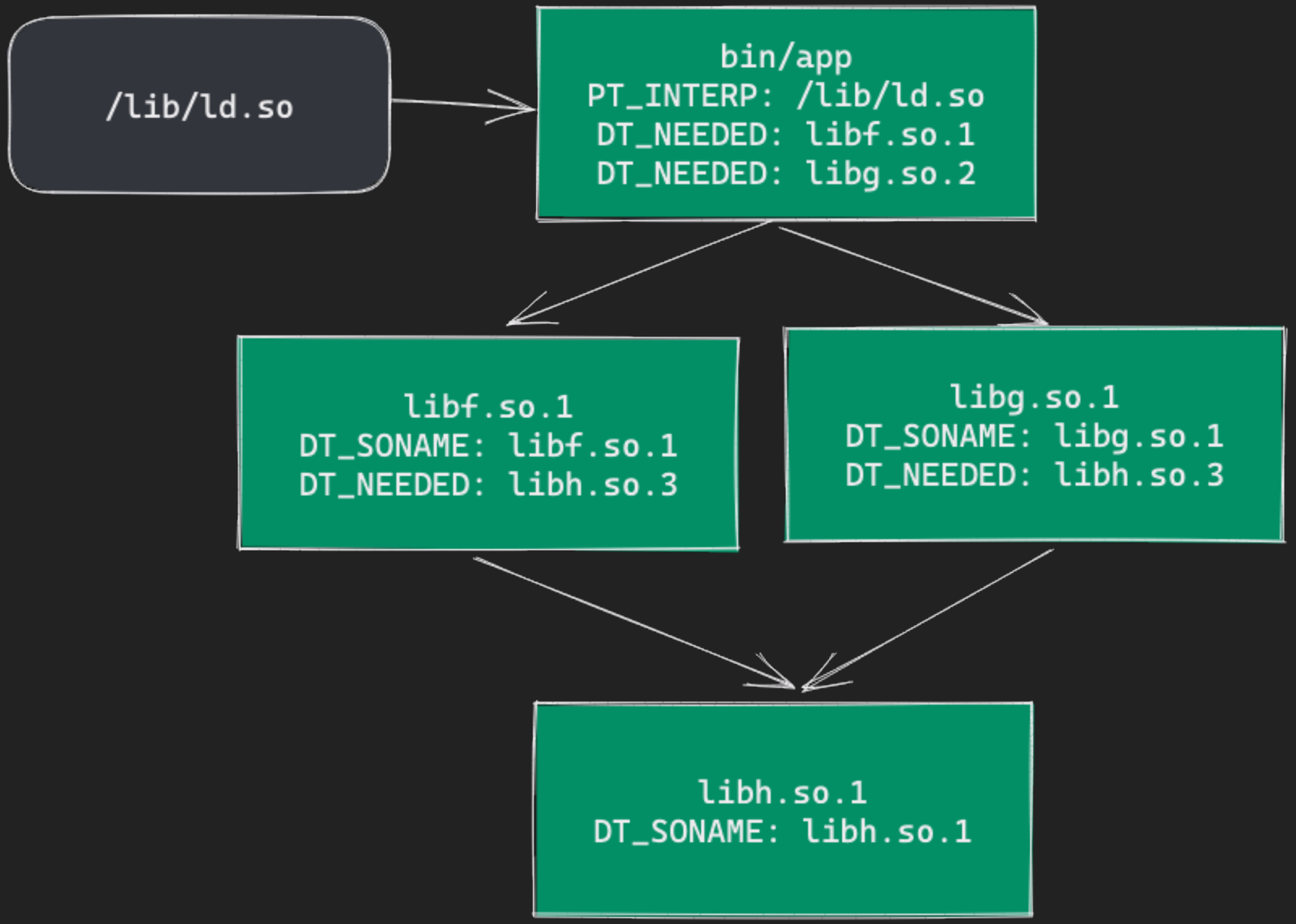
** Filesystem Hierarchy Standard*

LOCATING DEPENDENCIES AT RUNTIME

- The classical HPC solution: use multiple KBs of environment variables

- Binaries: `LD_LIBRARY_PATH`
- Python: `PYTHONPATH`
- Perl: `PERL5LIB`

- Too global, too opaque


/lib/ld.so
bin/app
PT_INTERP: /lib/ld.so
DT_NEEDED: libf.so.1
DT_NEEDED: libg.so.2
libf.so.1
DT_SONAME: libf.so.1
DT_NEEDED: libh.so.3
libg.so.1
DT_SONAME: libg.so.1
DT_NEEDED: libh.so.3
libh.so.1
DT_SONAME: libh.so.1

- We want users to be able to run executables without magic variables
- Typical solution: binary-local search paths through linker wrapper:
  `-Wl,-rpath,<prefix a>/lib:<prefix b>/lib:...:<prefix n>/lib`

- glibc: rpath > LD_LIBRARY_PATH > runpath > ld.so.cache > default paths
- musl libc: LD_LIBRARY_PATH > rpath=runpath > config paths > default paths

- The cost of rpath is search at runtime
- System executables w/o rpaths:

```
  for soname in needed
    path = ld.so.cache(soname, libc, arch)
```

- Spack executables w/ rpaths:

```
for soname in needed:
  for rpath in rpaths:
    for hwcap_dir in hwcap_dirs: # glibc specific, redundant in Spack-world
      open(rpath / hwcap_dir / soname)
```

```
$ libtree /usr/bin/git
/usr/bin/git
├── libz.so.1 [default path]
├── libpthread.so.0 [default path]
├── libpcre2-8.so.0 [default path]
│   └── libpthread.so.0 [default path]
└── libsha1detectcoll.so.1 [default path]
```

EXAMPLE 2: EMACS WITH GTK = 150+ NODES, 700+ EDGES, 20'000+ PATHS


```
$ libtree /usr/bin/emacs-gtk
/usr/bin/emacs-gtk
├── libz.so.1 [default path]
├── libpthread.so.0 [default path]
├── libacl.so.1 [default path]
│   └── libattr.so.1 [default path]
├── librt.so.1 [default path]
│   └── libpthread.so.0 [default path]
├── libtinfo.so.6 [default path]
├── libselinux.so.1 [default path]
│   └── libpcre.so.1 [default path]
├── libtiff.so.5 [default path]
│   ├── libz.so.1 [default path]
│   ├── liblzma.so.5 [default path]
│   │   └── libpthread.so.0 [default path]
│   ├── libjpeg.so.8 [default path]
│   └── libjbig.so.2 [default path]
├── libfreetype.so.6 [default path]
│   ├── libz.so.1 [default path]
│   ├── libbz2.so.1 [default path]
│   └── libpng16.so.16 [default path]
│       └── libz.so.1 [default path]
├── libfontconfig.so.1 [default path]
│   ├── libpthread.so.0 [default path]
│   ├── libfreetype.so.6 [default path]
│   └── libexpat.so.1 [default path]
├── libgconf-2.so.4 [default path]
│   ├── libpthread.so.0 [default path]
│   ├── libgmodule-2.0.so.0 [default path]
│   │   ├── libpthread.so.0 [default path]
│   │   └── libglib-2.0.so.0 [default path]
│   │       ├── libpthread.so.0 [default path]
│   │       └── libpcre.so.1 [default path]
│   ├── libglib-2.0.so.0 [default path]
│   ├── libgobject-2.0.so.0 [default path]
│   │   ├── libpthread.so.0 [default path]
│   │   ├── libglib-2.0.so.0 [default path]
│   │   └── libffi.so.7 [default path]
│   ├── libdbus-1.so.3 [default path]
│   │   ├── libpthread.so.0 [default path]
│   │   └── libsystemd.so.0 [default path]
│   │       ├── librt.so.1 [default path]
│   │       ├── libpthread.so.0 [default path]
│   │       ├── libselinux.so.1 [default path]
│   │       ├── libgcrypt.so.20 [default path]
│   │       │   └── libgpg-error.so.0 [default path]
│   │       ├── libcap.so.2 [default path]
│   │       ├── liblz4.so.1 [default path]
│   │       ├── libzstd.so.1 [default path]
│   │       └── liblzma.so.5 [default path]
│   └── libdbus-glib-1.so.2 [default path]
│       ├── libdbus-1.so.3 [default path]
│       ├── libglib-2.0.so.0 [default path]
│       ├── libgobject-2.0.so.0 [default path]
│       └── libgio-2.0.so.0 [default path]
│           ├── libz.so.1 [default path]
│           ├── libpthread.so.0 [default path]
│           ├── libresolv.so.2 [default path]
│           ├── libselinux.so.1 [default path]
│           ├── libglib-2.0.so.0 [default path]
│           ├── libmount.so.1 [default path]
│           │   ├── libselinux.so.1 [default path]
│           │   └── libblkid.so.1 [default path]
│           ├── libgmodule-2.0.so.0 [default path]
│           └── libgobject-2.0.so.0 [default path]
└── libotf.so.0 [default path]
```

STARTING EMACS BUILT WITH SPACK

```
$ strace -c ./emacs --version
GNU Emacs 28.2

% time     calls    errors syscall
------ --------- --------- ----------------
 77.89      3284      3213 openat
 12.26       868       780 stat
  3.53       191           mmap
  3.12        72           fstat
  1.69       141           mprotect
------ --------- --------- ----------------
100.00      4740      3994 total
```

STARTING EMACS BUILT WITH SPACK

```
Benchmark 1: /slow/fs/baseline/bin/emacs --version
  Time (mean ± σ):      45.3 ms ±   0.4 ms    [User: 10.4 ms, System: 27.4 ms]
  Range (min … max):    44.1 ms …  46.5 ms    63 runs
```

DYNAMIC LOADER OVERHEAD

- In Spack the bottleneck is loading objects, not relocation.
- Especially in HPC with slow fs:
  `#processes * #rpaths (* #hwcaps-dirs) * #libs` = many syscalls

HOW ABOUT STATIC LINKING

✅ Neither searching nor relocation

👎 Symbol clashes: shared libraries have public/private symbols with -fvisibility=hidden

👎 No LD_PRELOAD: it *is* convenient to swap out malloc, or an entire library like zlib with zlib-ng

👎 Static linking more likely to run into build issues

👎 Sometimes you just have stub libraries in the build environment

👎 Dynamic languages interface with dlopen(...)

GUIX'S PACKAGE-LOCAL LD.SO.CACHE

- Instead of `/etc/ld.so.cache`, make glibc use `$ORIGIN/../etc/ld.so.cache`

✅ Elegant

👎 Requires (patching) glibc

FILESYSTEM SYMLINK-BASED CACHE

- `<prefix-a>/lib/cache/libx.so.1 => <prefix-b>/lib/libx.so.1`
- Turn `n` rpaths into `1` rpath `<prefix-a>/lib/cache`

✅ Easy

✅ Works for glibc and musl libc

👎 Relative `$ORIGIN/xyz` rpaths becomes relative to the *symlink*

SHRINKWRAP (NIXOS/PATCHELF PULL REQUEST BY FARID ZAKARIA)

- Replace `DT_NEEDED` with absolute paths of the transitive closure (`ldd executable` output)
- When ordered properly, no recursion is required & deps are flattened.

✅ Interesting: "cache" is effectively baked into every executable

✅ Built on top of patchelf

👎 Patching ELF files has side-effects when not in-place

TYPICAL USER ISSUES ON HPC SYSTEMS

It builds fine but when I submit a job it ...

- ... can't find required libraries
- ... picks up the wrong `libstdc++.so` / `libgfortran.so` / etc

DISCREPANCY BETWEEN LINKER AND DYNAMIC LOADER

- `cc -shared f.c -o libf.so  # create a library`
- `cc hello.c -o hello lib.so  # executable links to it`
- `./hello # 😩`
  ```
  ./hello: error while loading shared libraries: libf.so:
  cannot open shared object file: No such file or directory
  ```

- Sure, I get it, but ...

ALTERNATIVE IDEA

- **Linker**: copies the soname of the library into `DT_NEEDED` the dependent
- **Dynamic loader**: searches `DT_NEEDED` *except* if it contains a `/`
  If `/` it is directly opened

- What happens if the soname contains a `/`? 🧠
  Actually, this trick is commonly used on macOS

CAN YOU JUST CHANGE SONAMES?

- Generally yes
- sonames are mostly a cache key
- GNU extension for introspection with `dlinfo(3)` is rarely used
  - *If* used (e.g. java), we simply exclude it

REPLACE SONAMES WITH LIBRARY'S ABSOLUTE PATH (POST-INSTALL)

- Opt-in from Spack 0.19, enable through
  `spack config add config:shared_linking:bind:true`

✅ WYLIWYG stability (what you link is what you get)

✅ Works outside of Spack (no wrappers / patches required)

👎 Intra-package linking (`curl` links to `libcurl.so`) does not benefit

HOW TO REPLACE SONAMES?

1. Nix's **patchelf** (currently used by Spack)
   ✅ can also fix intra-package linking
   👎 more `mmap` syscalls due to advanced ELF-shuffling

2. In-place updates (under consideration)
   Reserve space in dynamic section with placeholder rpath

   ```
   ld -rpath :::::<snip>:::
   ```

   like macOS's `ld -headerpad_max_install_names`

```
Benchmark 1: /slow/fs/baseline/bin/emacs --version
  Time (mean ± σ):      45.3 ms ±   0.4 ms    [User: 10.4 ms, System: 27.4 ms]
  Range (min … max):    44.1 ms …  46.5 ms    63 runs

Benchmark 2: /slow/fs/solution-5/bin/emacs --version
  Time (mean ± σ):      31.2 ms ±   0.3 ms    [User: 10.2 ms, System: 18.9 ms]
  Range (min … max):    30.7 ms …  32.0 ms    90 runs
```

GOT EVERYTHING BUT GLIBC

```
$ ldd emacs
  linux-vdso.so.1 (0x00007ffecaffb000)
  /path/to/spack/libtiff-4.4.0-yjm5bib4lkscmdc7hmvlodnkwstismle/lib64/libtiff.so.5.8.0 (0x00001551e08e3000)
  /path/to/spack/libjpeg-turbo-2.1.4-hijqvsrd7q4p44p7dclgairjxkxonssv/lib64/libjpeg.so.62.3.0 (0x00001551e0631000)
  /path/to/spack/libpng-1.6.37-y2tfsrm673ufdr66mbydbrwbguekguim/lib/libpng16.so.16.37.0 (0x00001551e03fa000)
  /path/to/spack/zlib-1.2.13-gbopbddrxdop7ea7ti4kfkqifrn7x3om/lib/libz.so.1.2.13 (0x00001551e01e1000)
  ...
  /path/to/spack/libx11-1.7.0-cxp4rycc5l6z3ganbmjipgx2geoqpsxw/lib/libX11.so.6.4.0 (0x00001551dcb8b000)
  /path/to/spack/libx11-1.7.0-cxp4rycc5l6z3ganbmjipgx2geoqpsxw/lib/libX11-xcb.so.1.0.0 (0x00001551dc988000)
  /path/to/spack/libxcb-1.14-7blk65qpx7m6hey6impgkriulqj4eqyh/lib/libxcb.so.1.1.0 (0x00001551dc75a000)
  /path/to/spack/libxrender-0.9.10-z3pkulzmcdq3ghv5glqxrj5w4bjcuzwl/lib/libXrender.so.1.3.0 (0x00001551dc54d000)
  /path/to/spack/librsvg-2.51.0-tasxznk7tiblyj3a4ie4uux76xwjzsjy/lib/librsvg-2.so.2.48.0 (0x00001551db9bd000)
  libm.so.6 => /lib64/libm.so.6 (0x00001551db672000)
  librt.so.1 => /lib64/librt.so.1 (0x00001551db469000)
  /path/to/spack/dbus-1.12.8-brt5wx3idou6ri3lobel4oine5wklgxe/lib/libdbus-1.so.3.19.7 (0x00001551db203000)
  /path/to/spack/libxrandr-1.5.0-4f4wle76giqfapfougvsdyj76qfvtrv2/lib/libXrandr.so.2.2.0 (0x00001551daff6000)
  /path/to/spack/libxfixes-5.0.2-2bn437dlgo4nu76cexmuftmcy6v3lfwo/lib/libXfixes.so.3.1.0 (0x00001551dadef000)
  /path/to/spack/libxext-1.3.3-cvqro27fb3pudhdmjit6clrqttlv6dfj/lib/libXext.so.6.4.0 (0x00001551dabda000)
  ...
```

- libc was not absolutified
- musl libc: the loader is libc, it doesn't need to be located
- glibc: loader spends ~400 syscalls to locate ... itself?!

```
Benchmark 1: /slow/fs/baseline/bin/emacs --version
  Time (mean ± σ):      45.3 ms ±   0.4 ms    [User: 10.4 ms, System: 27.4 ms]
  Range (min … max):    44.1 ms …  46.5 ms    63 runs

Benchmark 2: /slow/fs/solution-5/bin/emacs --version
  Time (mean ± σ):      31.2 ms ±   0.3 ms    [User: 10.2 ms, System: 18.9 ms]
  Range (min … max):    30.7 ms …  32.0 ms    90 runs

Benchmark 3: /slow/fs/solution-5-preload-glibc/bin/emacs --version
  Time (mean ± σ):      21.8 ms ±   0.2 ms    [User: 9.4 ms, System: 11.1 ms]
  Range (min … max):    21.3 ms …  22.5 ms    120 runs

openat from 3284 down to 110; stat from 868 down to 0
```

FURTHER LINKS

- https://github.com/spack/spack/
- https://github.com/NixOS/patchelf/pull/357 (shrinkwrap)
- https://guix.gnu.org/blog/2021/taming-the-stat-storm-with-a-loader-cache/
- https://github.com/NixOS/nixpkgs/pull/207061 (nix's relative ld.so.cache)
- https://github.com/haampie/libtree