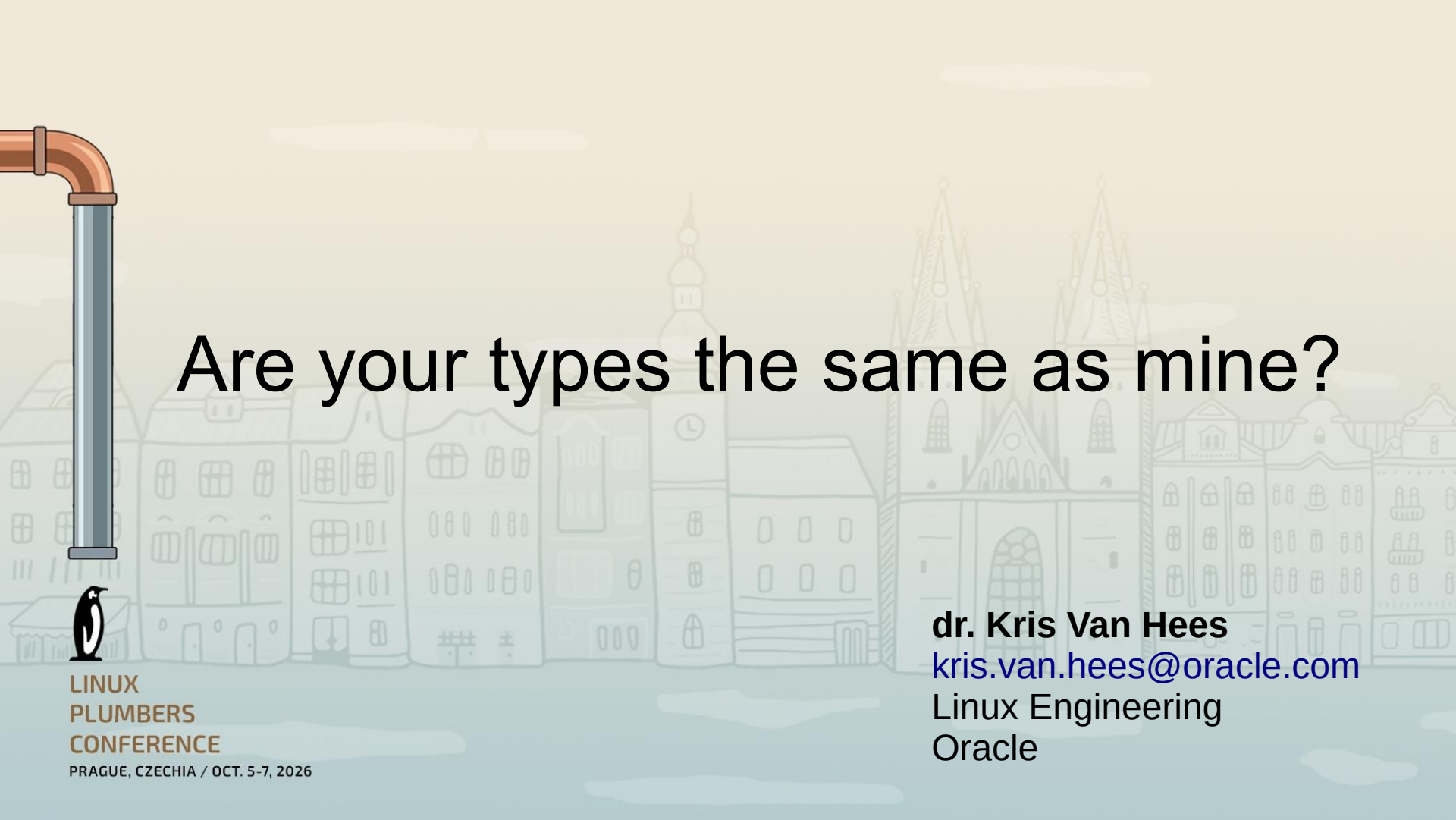




Are your types the same as mine?



LINUX
PLUMBERS
CONFERENCE

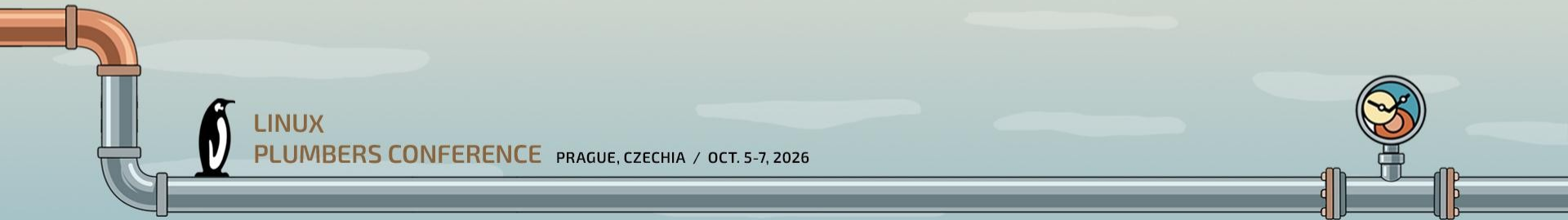
PRAGUE, CZECHIA / OCT. 5-7, 2026

dr. Kris Van Hees
kris.van.hees@oracle.com
Linux Engineering
Oracle

Overview

Tracers (and debuggers) depend on type information for various things:

- Access global data
- Access function arguments and return values
- Identify functions to attach to
- Manage session state
- Manage internal state



Where does type data come from?

Full-stack tracers (and debuggers) are different from compilers because:

- 1) You often do not have access to source code header files
- 2) You may not know which header files were used to compile the things you are tracing
- 3) The things you trace may have distinct versions of types with the same name
- 4) Trace scripts can declare their own types

One single type dictionary cannot represent all this

- Distinct types with the same name are possible
- Some complex tracing needs may require handling of new type data discovered while tracing is going on



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Where does type data come from?

Using DTrace as an example:

- Kernel
 - BTF, CTF
- Userspace
 - BTF, CTF
- Tracer's own data structures
 - CTF
- Trace session data
 - CTF

No DWARF because it is usually not available (or too big to use).



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Types used in tracing (“easy” case)

Example: Tracing `fstat()`

- Kernel
 - syscall `newfstat(int struct stat *)`
- Userspace
 - C-library `fstat(int, struct stat *)`
 - Application functions using `struct stat *`
 - Global data stored as `struct stat *`
- Tracing script
 - TLS or global data stored as `struct stat *`

All compiled using the same (or equivalent) `struct stat` declaration
... hopefully



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Are my types the same as yours?

How do I know whether the following types are the same:

- vmlinux BTF id 17127 (struct stat)
- a.out BTF id 31 -or- CTF id 32 (struct stat)
- D CTF id 2147490113 -or- CTF id 21270 (struct stat)

Deep comparison of the type definitions:

- Expensive
- May need conversion to a common format
- If they are the same, pick one, and replace references to the other ones with the most convenient representative copy
- But some types only exist in one or some type collections



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



All your type belong to me

One easy solution:

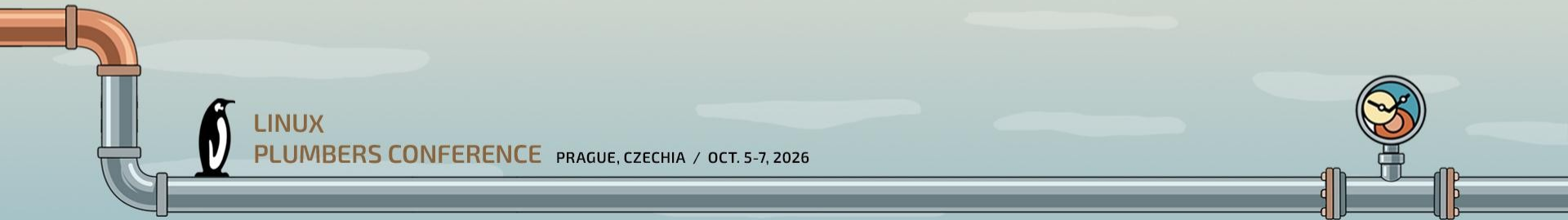
- Copy all types into a single type collection
- Perform all type lookups on that collection

Advantage:

- Only need to deal with a single authoritative type collections

Disadvantage:

- Expensive (deep copy of all types from all sources)
- Still need to compare types



All your type belong to me

Challenges:

- When types can come from distinct sources (BTF, CTF, ...), you need to convert between representations
- Depending on which you use for the unified type collection, you need to implement your own deep-copy function for type data
- This won't work if there are type conflicts
 - Less common for kernel UAPI types
 - Mostly seen when tracing two userspace programs
 - Occasionally seen when a trace script declares its own type for storing state, and it conflicts with some other type source



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



You can keep your types

When a unified type collection is a problem...
... don't unify them!

Advantage:

- No type conflicts
 - Types can be identified by (collection, type id) pairs

Disadvantages:

- Type comparison becomes expensive
 - If same collection, same type id → SAME
 - If same collection, different type id → check if assignment compatible
 - If different collection → deep (recursive) compare needed



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



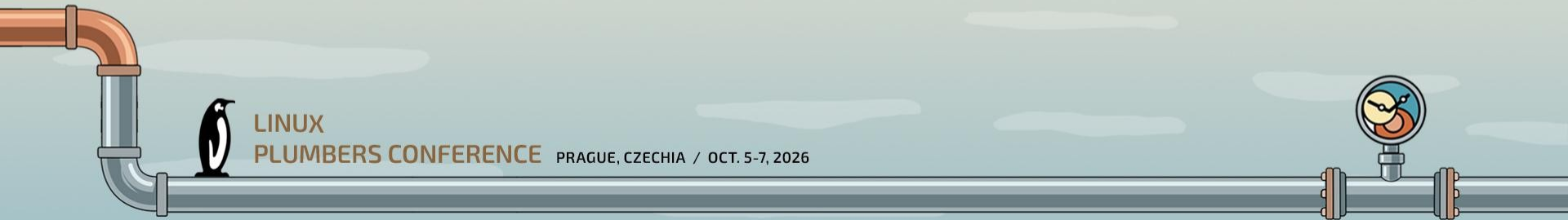
You use yours, I use mine

Still need to deal with different sources:

- 1) Pick one kind (BTF or CTF) and convert as needed
- 2) Support multiple kinds, and implement deep (recursive) comparison functions to e.g. compare a BTF type description against a CTF type description

Disadvantages:

- Conversion is expensive (on-demand is not practical)
- Multi-source deep comparison is expensive
 - Optimization is usually equivalent to just doing the conversion anyway



You use yours, I use mine

Hybrid solutions:

- 1) Add a comparison cache to the main type representation (e.g. CTF)
 - Record all (kind, collection, type id) triples that are equivalent
 - Can be done on-demand, as a comparison cache
- 2) Add a hashtable of (kind, collection, type id) triples
 - Store the main type representation as value



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



I want to use your types

libbpf

- Would be nice to have a libbtf
- Would be nice to have a deep (recursive) comparison function
- Would be nice to have some caching of (btf *, id) comparison results to avoid repeated deep comparisons

libctf

- Would be nice to have some caching of (ctf_dict_t *, id) comparison results to avoid repeated deep comparisons

(It would be REALLY nice if we could just have ONE type representation to deal with.)

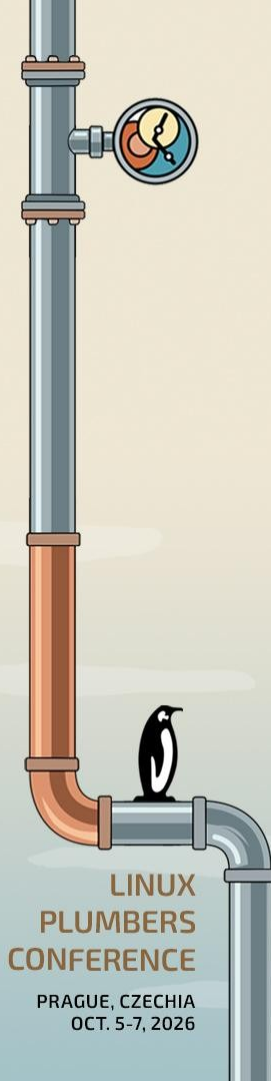


LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Thank you!



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026