CTF: future plans and use in the kernel

<nick.alcock@oracle.com>

The last year’s work

- Almost entirely one thing: link-time deduplication, and infrastructure needed by it.
- The result is not very optimized but is already fast and fairly compact (a bit over a minute to emit 10MiB of CTF for a typical 3000-module enterprise kernel, including subdivision of types by module and all function prototypes, and including unused types).

Future plans

- Now we can really get going on interesting stuff!
- My todo list: https://github.com/oracle/binutils-gdb/wiki/libctf-todo
- Literally just committed gettextization (yesterday). Currently working on symtab handling to map from symtab entries to types
- pahole can emit C code from BTF: we could do with a similar tool for CTF

A format bump

- Format v4 is the first format bump in a couple of years. Lots of plans! Read and write support for current format kept.
- Improved compactness via a better fundamental type encoding, various forms of delta-compression for big structures and perhaps for repetitive member names, and replacement of parent/child dicts for ambiguous types with a better representation using the unused “labels” feature of CTF: all provisional, dependent on whether it improves compactness after compression
- Representation of GCC attributes (as per request at last LPC)
- Drop the variable section: use symtab handling to allow mapping of names to variables *and functions* (as per request at last LPC)
- Record arg names in function prototypes (important for C code generation)
- LZMA compression (not sure what to do if users don’t have liblzma)

All plans change

- Last year’s future plans for CTF included reading and writing of all related formats, including BTF
- BTF’s rate of change seems far too high for direct writing to be practical: it seems to change multiple times per month
- binutils (and thus libctf) releases once every six months, so even if the kernel always forced building with the latest binutils (which it doesn’t), libctf could only ever emit an old BTF format
- Thus, direct emission of BTF from libctf is probably impractical: reading of BTF from current kernels is also impractical

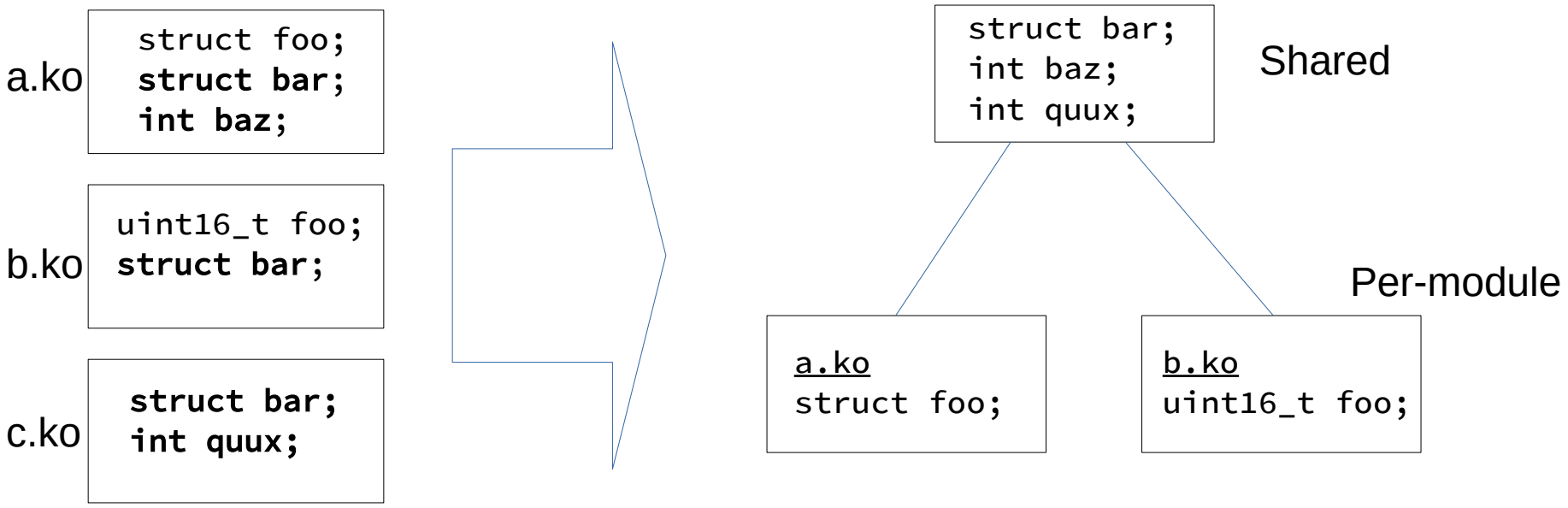
So, can libctf help at all?

I think it can!

Reusing CTF dedup in BTF

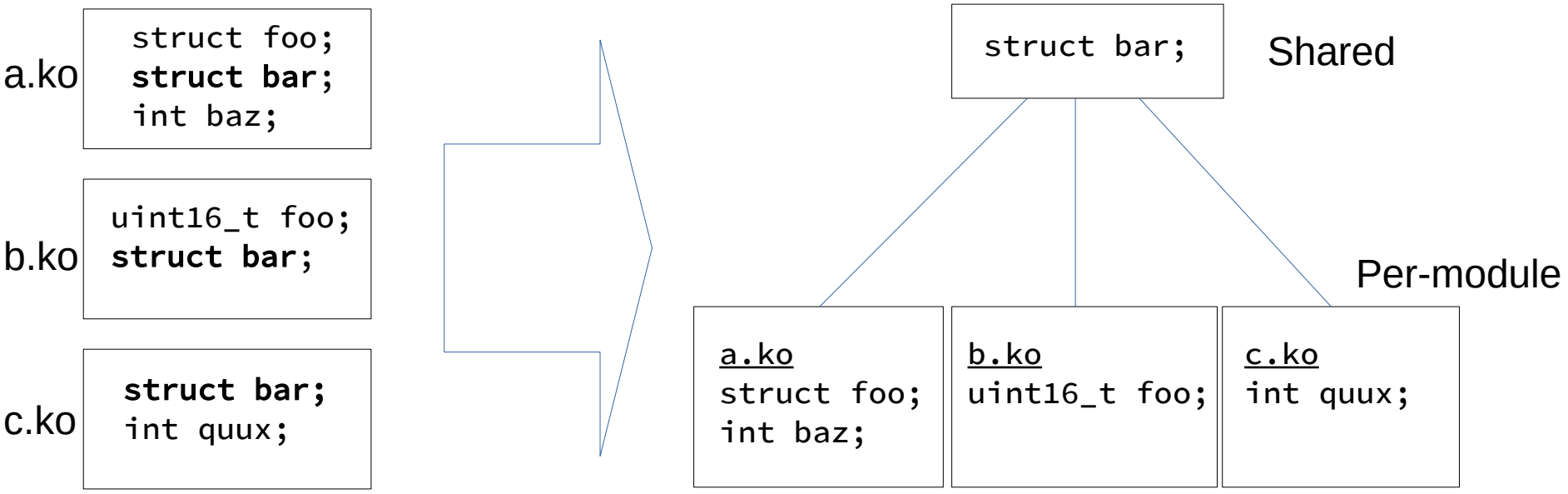
- Steal an idea from BTF
- Have an objdump option to generate C code from CTF in a linked vmlinux (and/or one or more modules?): you get nothing but types out, obviously. These are *already deduplicated* types.
- Parent/child relationships in the input CTF can be emitted as multiple TUs #including a parent TU (or just as a comment that some reading tool can parse?)
- Two ways of distributing types between TUs currently exist

Type distribution: share-unconflicted



This puts all types without ambiguous definitions into the shared parent.

Type distribution: share-duplicated



This puts only types shared between modules into the shared parent.

Reusing CTF dedup in BTF, contd

- You could then process these C files without needing a deduplicator
- If this idea turns out to be useful it means that CTF can pull in BTF-generated type C in the same fashion, so BTF and CTF can borrow each others' capabilities while retaining a fairly loose coupling, so that both can grow without restricting the other.

Does this seem useful?

Can we do anything else?