BPF Support in the GNU Toolchain

David Faust Cupertino Miranda

Oracle

Linux Plumbers Conference 2024
September 18-20 | Vienna

Outline

Status
- Recent work in binutils
- Recent work in GCC
- Kernel selftests

Discussion Topics

Discussion Topics: Future Works

Major Milestone

- Compile all kernel selftests
- ... and pass most of them
- Compile DTrace, systemd, others

Availability and Adoption

Distro availability:

- Oracle Linux `cross-gcc, cross-binutils`
- Debian `gcc-bpf, binutils-bpf`
- Gentoo `sys-devel/bpf-toolchain`
- Fedora `cross-gcc, cross-binutils`
- and derivatives

Projects building with GCC BPF:

- DTrace
- systemd
- bpftune
- ...

Availability and Adoption

Distro availability:

- Oracle Linux `cross-gcc, cross-binutils`
- Debian `gcc-bpf, binutils-bpf`
- Gentoo `sys-devel/bpf-toolchain`
- Fedora `cross-gcc, cross-binutils`
- and derivatives

Projects building with GCC BPF:

- DTrace
- systemd
- bpftune
- ...

Outline

New Instructions

- Support for all CPU v4 instructions:
  - Unconditional byte swapping instructions (`bswap{16,32,64}`)
  - Long gotos with 32-bit target displacement (`jal`)
  - Signed memory load instructions (`ldsxw`,...)
  - Signed register moves (`smov`,...)
  - Signed division and modulus (`sdiv`, `smod`,...)
- Support for relaxation with v4 instructions:
  - `ja disp16 -> jal disp32`
  - `jxx disp16 -> jxx +1; ja +1; jal disp32`
  - Option `-m[no-]relax`, enabled by default

Immediate Overflows

Consolidated handling of immediate overflow to work the same between GAS and LLVM BPF assembler

- For an immediate field of `N` bits, *any* written number whose two's complement encoding fits in `N` bits is accepted
- e.g. `-2` is the same as `0xfffffffe`
- Up to the instruction to decide how to interpret the value
- Do not relax immediate fields in jump instructions; relax to jumps with wider range only when expressions are involved

ELF Header Flags

Add `ELF_BPF_CPUVER` bits in the ELF machine-dependent header flags

- Encode the BPF CPU version for which the object file has been compiled
- A value of zero indicates "use latest supported version"
- Disassembler honors flags and uses appropriate ISA version if user did not specify on command line

```
$ readelf -h foo.o.bpf
ELF Header:
     ...
     Flags:                             0x4, CPU Version: 4
```

Outline

Assorted Changes and Improvements

- Compiler-shipped `bpf-helpers.h` workaround file has been removed!
- Complete BPF CO-RE implementation
- Generate BTF by default with `-g` for BPF target
- Emit pseudo-C asm syntax by default :(
  - Necessary due to inline asm prevalence in kernel BPF headers
- Inline `__builtin_{memmove,memcpy,memset}` into verifier-friendly sequences

BPF Feature Macros

GCC now defines BPF feature macros used in existing programs:

- `__BPF_CPU_VERSION__ (1, 2, 3, 4)`
- Enabled with `-mcpu=v3` or higher:

`__BPF_FEATURE_ALU32`	`-m[no-]alu32`
`__BPF_FEATURE_JMP32`	`-m[no-]jmp32`
`__BPF_FEATURE_JMP_EXT`	`-m[no-]jmp-ext`
`__BPF_FEATURE_BSWAP`	`-m[no-]bswap`
`__BPF_FEATURE_SDIV_SMOD`	`-m[no-]sdiv`
`__BPF_FEATURE_MOVSX`	`-m[no-]smov`

- Enabled with `-mcpu=v4` or higher:

`__BPF_FEATURE_LDSX`
`__BPF_FEATURE_GOTOL`
`__BPF_FEATURE_ST`

BTF Pruning: Context

selftests/bpf/progs/bpf_loop.c

GCC

```
$ objdump -h -j.BTF
Sections:
Idx Name          Size
  6 .BTF          00099c47

$ bpftool btf dump
...
[8766] DATASEC 'license'
```

???

clang

```
Sections:
Idx Name          Size
 11 .BTF          00000ce8

...
[56] DATASEC 'license'
```

BTF Pruning

`-g[no-]prune-btf`

- Prune BTF prior to emission, same algorithm as clang:
  - Start from only types actually used in the program
  - Avoid chasing pointers to struct/union types if otherwise unused
- Enabled by default for BPF target with `-g`
- Give users choice of "clang-like" or "pahole-like" BTF:
  - clang: prune: minimal BTF to load and run program
  - pahole: no-prune: translate DWARF to BTF
- GCC supports generating BTF for all targets

Outline

Kernel Selftests

453/3446 PASSED, 92 SKIPPED, 102 FAILED

- Missing `type_tag` and `decl_tag` in GCC
- Valid code patterns not understood by verifier
- Bleeding-edge BPF features not yet implemented

Outline

BPF CI with GCC

- Catch new testsuite additions accidentally relying on compiler-specific features or behaviors
- For now: compile only
- Once all selftests pass: run

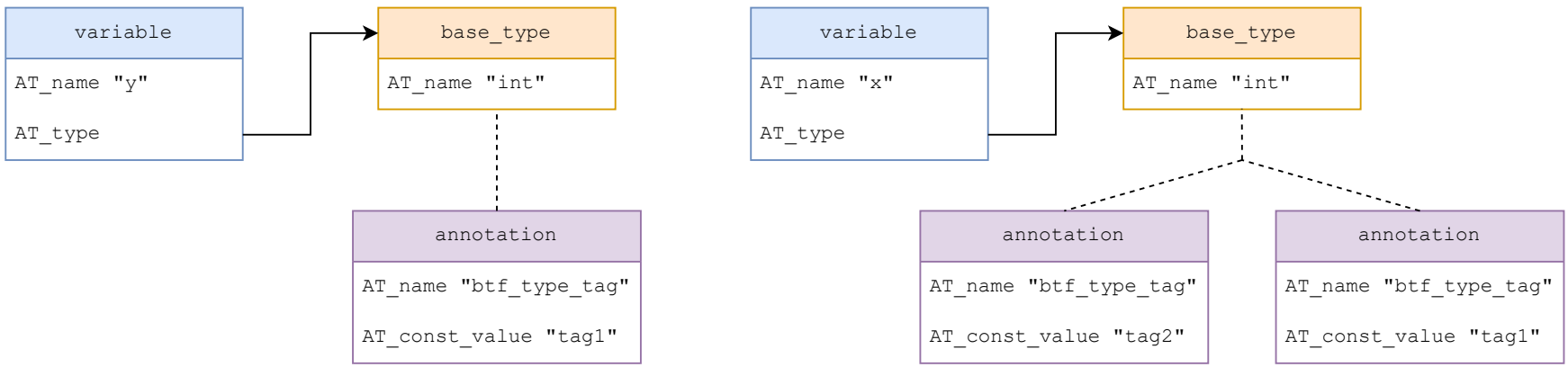
BPF Memory Model

WIP in both GCC and LLVM

- `__atomic_X` builtins accept a memory order argument
- For relaxed ordering, if return value unused, use lock insn
- BTF for `_Atomic`
  - `BTF_KIND_ATOMIC` to behave as cv-qual ?
  - GCC currently ignores `_Atomic` when generating BTF
- LLVM patches: `https://github.com/llvm/llvm-project/pull/107343`
- GCC tracking PR: `https://gcc.gnu.org/bugzilla/show_bug.cgi?id=116717`

DWARF BTF Tags: Pending Format

```
int __tag1 __tag2 x;
int __tag1 y;
```



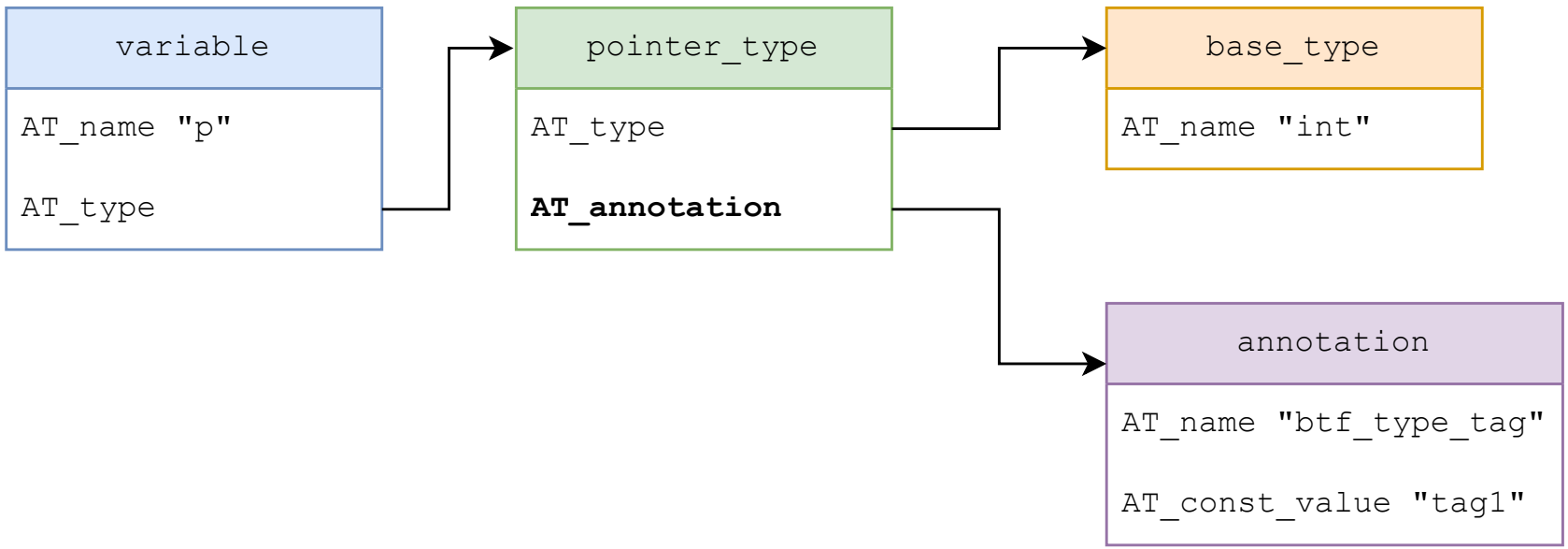
- Wasteful!
- Room to improve; blocker for GCC patches

DWARF BTF Tags: Proposal

- Add new `DW_AT_annotation` in addition to `DW_TAG_annotation`
- Holds a reference to annotation DIE for that item, if any
- Annotation DIEs may be chained by additional `AT_annotation`
- Can reuse (sub-)chains of annotations

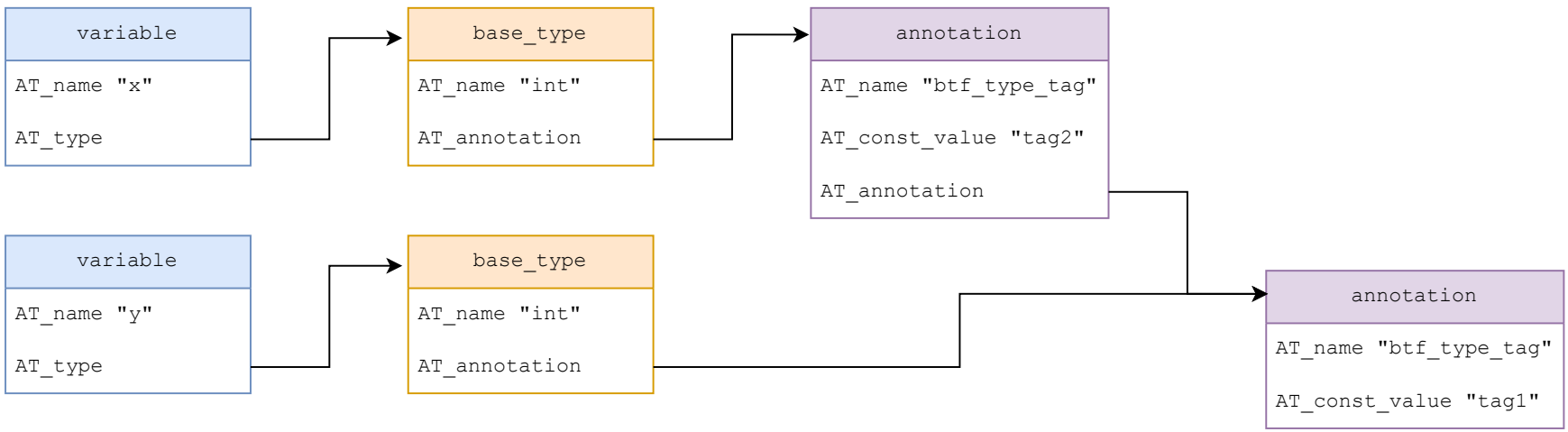
DWARF BTF Tags: Proposal

```
int * __tag1 p;
```



DWARF BTF Tags: Proposal

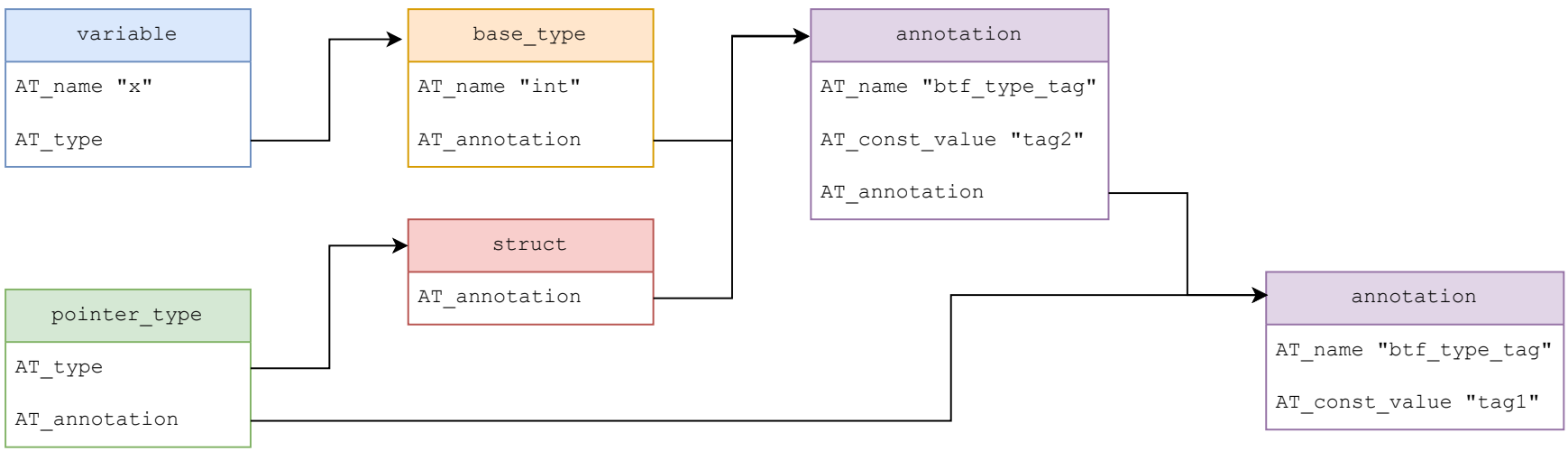
```
int __tag1 __tag2 x;
int __tag1 y;
```



- Annotations can be shared and reused

DWARF BTF Tags: Proposal

```
int         __tag1 __tag2 x;
struct S __tag1 __tag2 * __tag1 ...
```



- Ordering is preserved

DWARF BTF Tags: Proposal

- Still safe for consumers unaware of extension
- No objections from GCC DWARF maintainers
- Preserves ordering of tags
- Opinions?

CO-RE (in GCC)

- GCC CO-RE implementation on par with clang
- GCC builtins have different prototypes (abstracted in `bpf_core_read.h`)
- `#pragma clang attribute push ...` not supported
  - correct bpftool to properly attribute structs
- Possible next steps for CO-RE

CO-RE (clang example)

Input

```
struct Q {
  int x;
  int y;
} __attribute__((preserve_access_index));

struct P {
  struct Q qs[10];
} __attribute__((preserve_access_index));

void
bar (struct P *p, int i)
{
  p->qs[i].y = 6;
}
```

clang

```
bar:
  r3 = 0x0
  CO-RE <byte_off> [2] struct P::qs (0:0)
  r1 += r3
  w2 = w2
  r2 «= 0x20
  r2 s»= 0x20
  r2 «= 0x3       % multiplication by 8
  r1 += r2
  w2 = 0x2a
  *(u32 *)(r1 + 0x4) = w2
  CO-RE <byte_off> [3] struct Q::y (0:1)
  exit
```

Compiler explorer link: https://godbolt.org/z/7Gec6xeTG

CO-RE (gcc vs. clang)

gcc

```
bar:
  r2 = (s32) r2
.L3:
  r0 = 8 ll
  % CO-RE <sizeof> struct Q
.L4:
  r3 = 0 ll
  % CO-RE <byte_off> struct P::qs (0:0)
  r0 *= r2
  r1 += r3
.L5:
  r4 = 4 ll
  r1 += r4
  r1 += r0
  *(u32 *) (r1+0) = 42
  % CO-RE <byte_off> struct Q::y (0:1)
  exit
```

clang

```
bar:
  r3 = 0x0
  CO-RE <byte_off> [2] struct P::qs (0:0)
  r1 += r3
  w2 = w2
  r2 «= 0x20
  r2 s»= 0x20
  r2 «= 0x3       % multiplication by 8
  r1 += r2
  w2 = 0x2a
  *(u32 *)(r1 + 0x4) = w2
  CO-RE <byte_off> [3] struct Q::y (0:1)
  exit
```

- gcc is allowing non constant indexing of arrays using a sizeof CO-RE reloc.
- What about enum indexing?

Verifier and PTR to CTX restrictions

```
SEC("iter/task_vma")
int get_vma_offset(struct bpf_iter__task_vma *ctx) {
  struct vm_area_struct *vma = ctx->vma;
  struct seq_file *seq = ctx->meta->seq;
  struct task_struct *task = ctx->task;

  if (task == NULL || vma == NULL)
    return 0;
  ... }
```

```
reg type unsupported for arg#0 function get_vma_offset#8723
0: R1=ctx() R10=fp0
0: (18) r0 = 0x10                     ; R0_w=16
2: (bf) r2 = r1                       ; R1=ctx() R2_w=ctx()
3: (0f) r2 += r0                      ; R0_w=16 R2_w=ctx(off=16)
4: (79) r0 = *(u64 *)(r1 +8)          ; R0_w=ptr_or_null_task_struct(id=1) R1=ctx();
5: (15) if r0 == 0x0 goto pc+21       ; R0_w=ptr_task_struct()
6: (79) r2 = *(u64 *)(r2 +0)
dereference of modified ctx ptr R2 off=16 disallowed
```

- Is clang representing verifier restrictions?
- Can the verifier relax these rules?

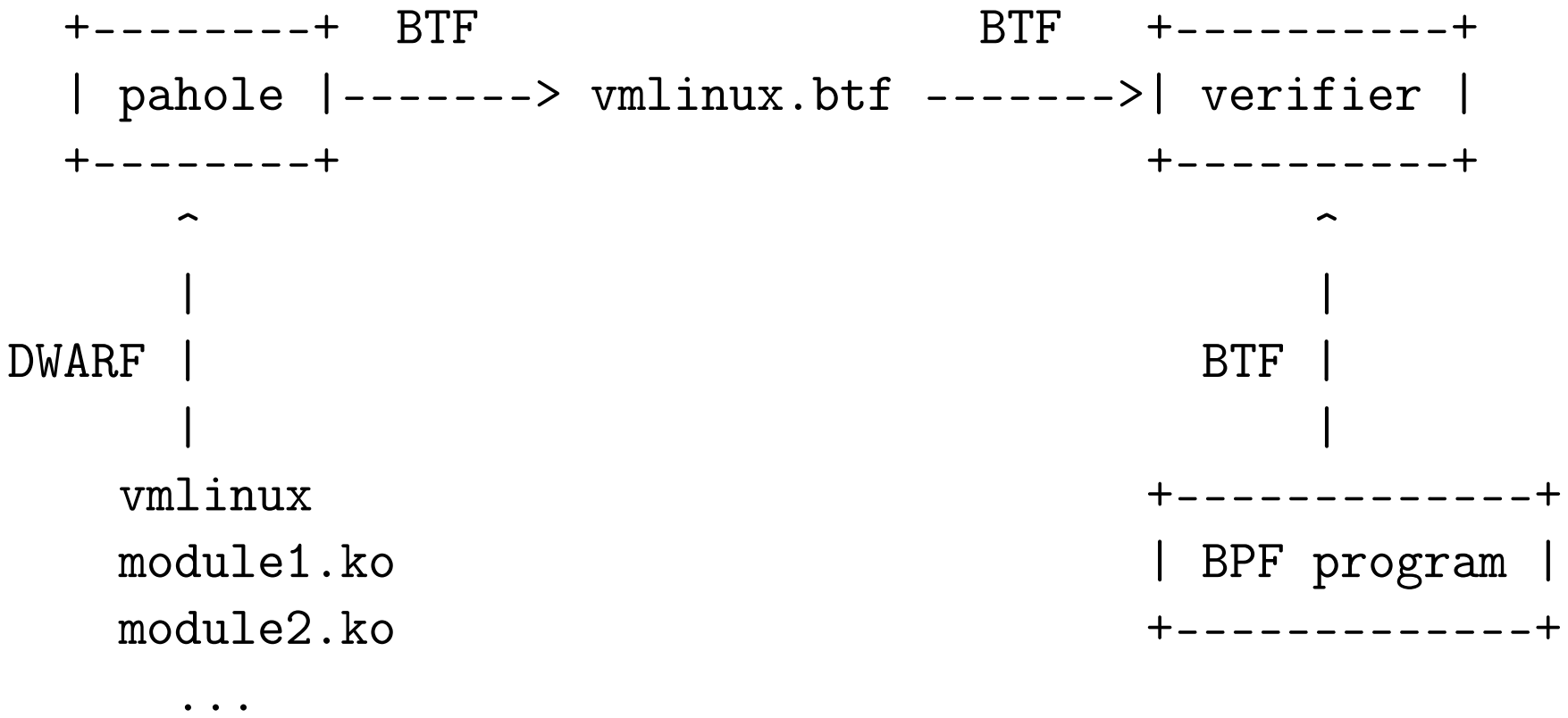
Outline

BTF support in toolchain

Kernel BTF build relies on `pahole` translating DWARF to BTF, introduces dependency on DWARF:

> *All information expressed in BTF shall be conveyable in standard DWARF or deducible from some source available to pahole*

BTF for kernel: now



Removing DWARF dependency for kernel BTF

All information expressed in BTF shall be conveyable in standard DWARF or deducible from some source available to pahole

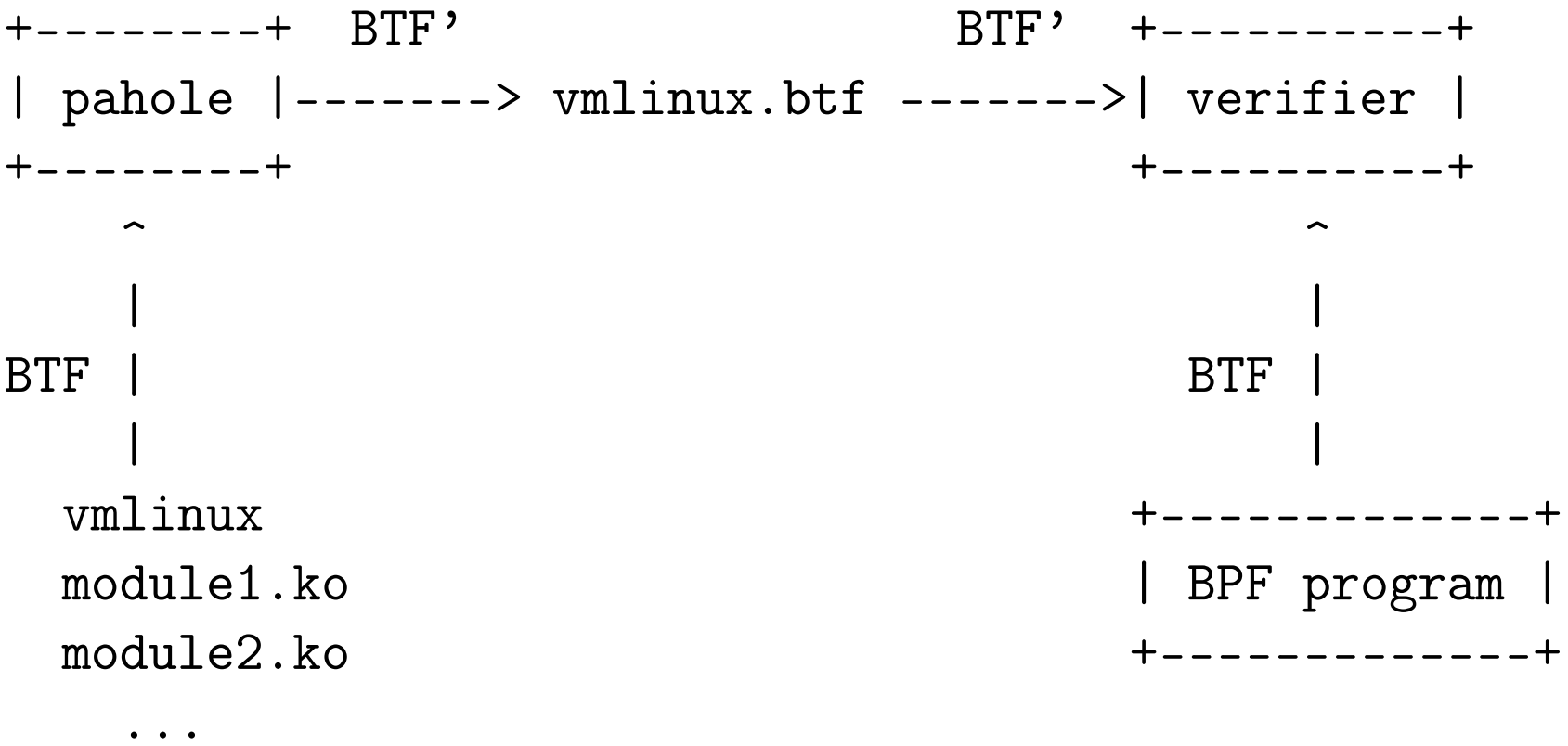
This is problematic because:

1. DWARF is difficult to extend without breaking it
2. Proper additions to DWARF shall be done through the standard
3. Coupling of the two formats is gratuitous
4. DWARF for kernel is hundreds of MiB (at least)

Toolchain BTF support

- Support merging and deduplicating BTF in the linker
- GNU `ld` already supports merging and deduplication for CTF; could be extended to merge and deduplicate BTF also
- Then: skip DWARF, pahole amends toolchain-produced BTF

BTF for kernel: next



Linking for BPF programs

- Already some static linking of BPF programs with libbpf
- Why not do it with toolchain proper?

BPF, BTF and Rust

Updates from Rust-for-Linux meeting at Kangrejos earlier this month:

- Most RfL people did not know about BTF until now
- BTF must reflect realized structs, after Rust compiler has reordered
- RfL people said they would look into BTF, and to CO-RE
- Confirmed that ORC can be reverse-engineered from compiled Rust, and it works
- CFI directives generated by Rust compiler are enough SFrame also

Approaches for compiling for verified targets

1. Do nothing
2. Disable all optimizations
3. Disable some optimizations
4. Target counterpasses
5. Target driven pass tailoring
6. Language level support e.g. “must pragmas”
7. Assembler support

- LPC 2023 decided: some combination of 2-7
- We can start with 7, gas now has control flow graphs via SCFI

32-bit sub-register support

Following compiler constraints agreed on at LSFMM; not yet implemented

For immediates:

```
"i":  imm64
"I":  imm32
"O":  off16
```

For registers:

32-bit sub-register support

Following compiler constraints agreed on at LSFMM; not yet implemented

For immediates:

```
"i":  imm64
"I":  imm32
"O":  off16
```

For registers:

32-bit sub-register support

"r": 64-bit register (rN) or 32-bit sub-register (wN), based on the mode of the operand

If 32-bit arithmetic available:

- char, short -> wN and warning
- int -> wN
- long int -> rN

Else:

- char, short, int -> rN and warning
- long int -> rN

32-bit sub-register support

"w": 32-bit sub-register (wN) regardless of the mode of the operand

If 32-bit arithmetic available:

- `char, short -> wN and warning`
- `int -> wN`
- `long int -> wN and warning`

Else:

- `char, short, int, long int -> wN and warning`

32-bit sub-register support

"R": 64-bit register (rN) regardless of mode of operand

- `char, short, int -> rN and warning`
- `long int -> rN`

Fast calls

`__attribute__((bpf_fastcall))`

- GCC: To-do `https://gcc.gnu.org/bugzilla/show_bug.cgi?id=116718`

Inclusion of system headers

- GCC provides standard headers e.g. stdint.h
- clang does not?
- Should harmonize both toolchains