WHEN GDB IS NOT ENOUGH

PAUL SEMEL

KEVIN TAVUKCIYAN

TALKING ABOUT DWARF

TALKING ABOUT DWARF

DEBUGGING WITH ATTRIBUTE RECORD FORMATS

WHAT IS DWARF?

WHAT IS DWARF?

- Format used to store debug informations
- Generated by the compiler
- Sections in the binary format

HOW DOES IT WORK?

INFO

- Contains all information on our types
- Informations are sorted by compilation units.
- Info entry are called DIE (Dwarf Info Entry).

COMPILATION UNIT

- Interesting projects have more than one source code file and are compiled separately and linked together
- They are called Compilation Unit in DWARF
- Each DIE are different and separated by compile units

DIE (SOUNDS APPEALING ALREADY HUH?)

- Basic description entry in DWARF
- Has a tag which precises what it describes
- Can describe data or functions/executable code

ABBREV

- Table of abbreviation
- Used to compress data inside the section
- Contains info on the content of the DIE

APPLICATIONS

GNU BINUTILS

- Multiple DWARF4 parsing implementation
- Too deeply merged in the rest of the projects
- 10k+ LOC in objdump / 25k+ LOC in gdb

OUR PROJECT

DWARF PRETTYPRINTER

- Get structure members.
- Print structure content.
- ~~1500 LOC : Deal with it Stallman !~~

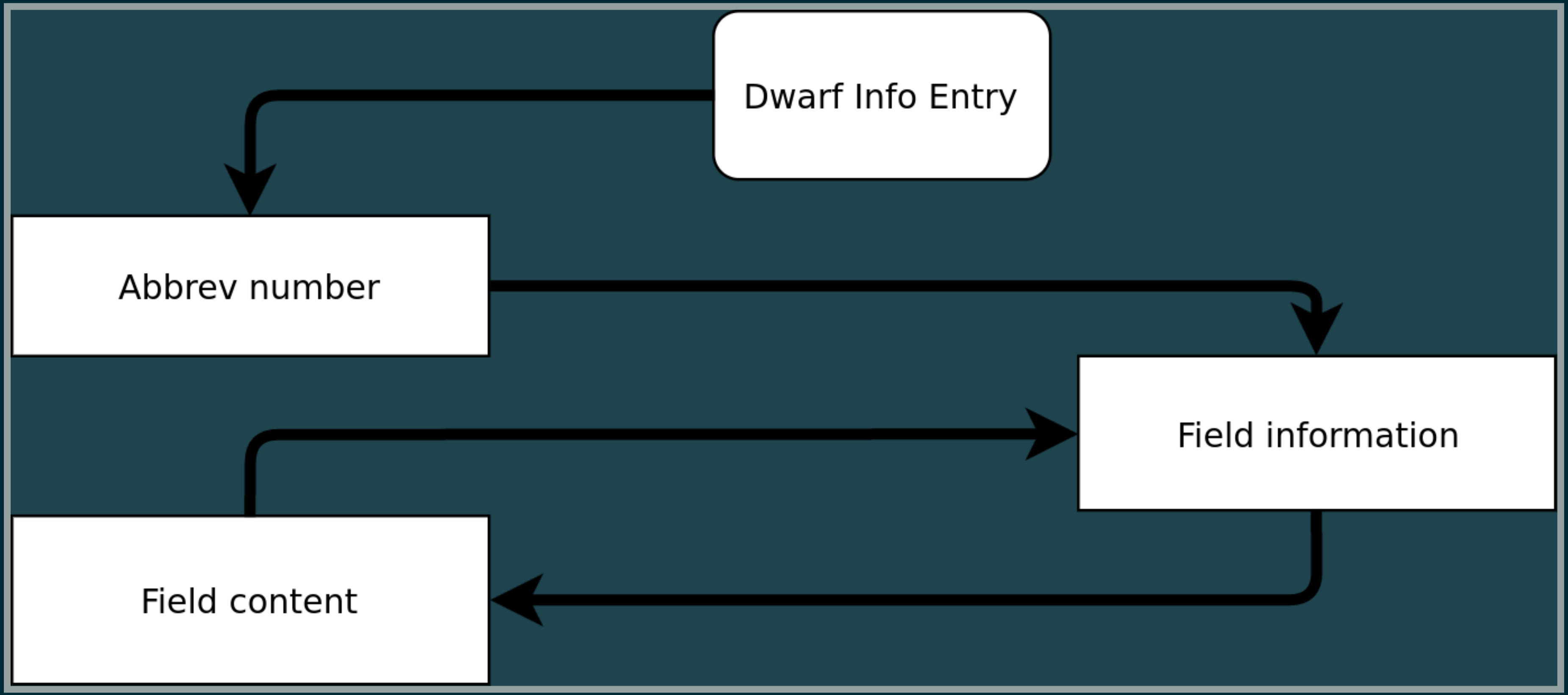
WHAT WAS THE POINT OF THE PROJECT?

- Create a lib that can print the structure when given a pointer to it.
- Useful for debug when gdb is too much.
- As fast and as lightweight as possible.

TECHNIQUES USED

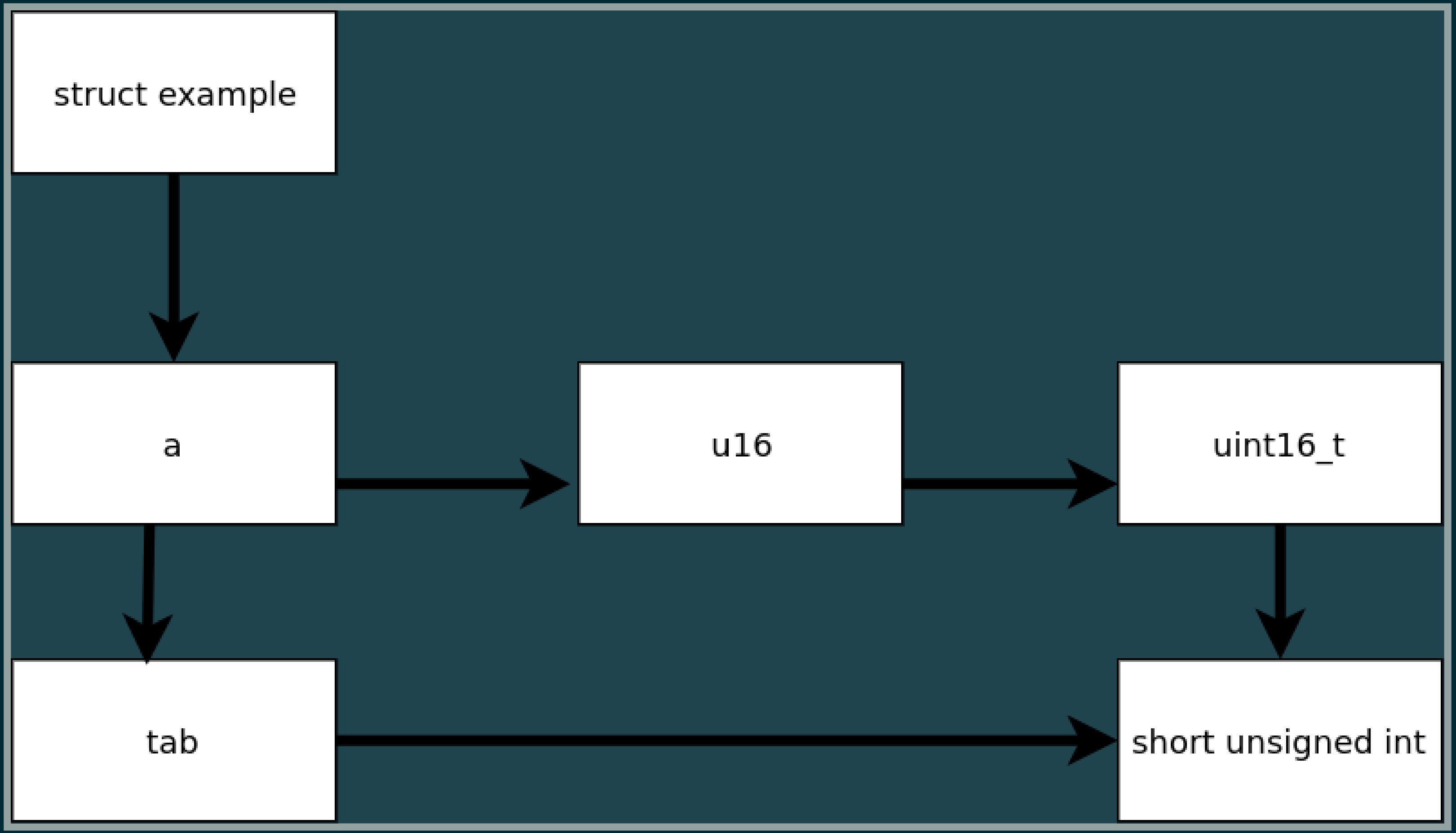
DWARF FORMAT PARSING

- DWARF format is against us.
- We needed a technique to avoid loading the whole DWARF tree in memory


Dwarf Info Entry
Abbrev number
Field information
Field content

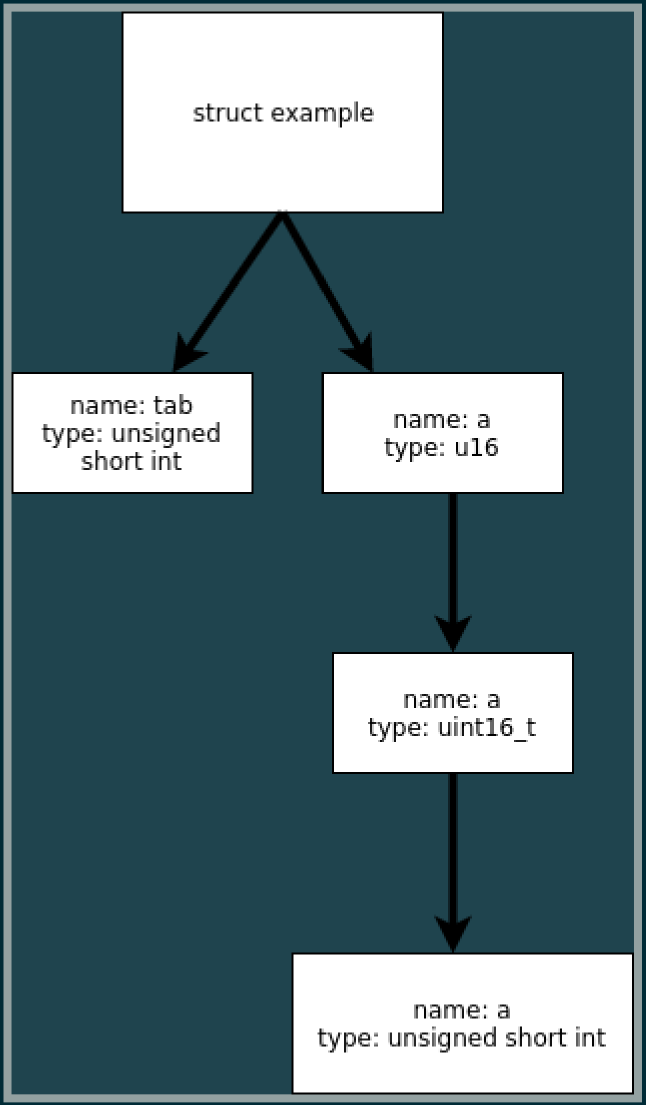
WHAT'S WRONG WITH THIS ?

- We need to parse the whole format until we find a DIE.
- The size of a tag can be variable.


struct example
a
u16
uint16_t
tab
short unsigned int

SO, HOW CAN WE OPTIMIZE THIS ?

- We need to fix the size of as most tag as we can.
- If we encounter a structure, we keep it in memory.
- We remain where we stopped parsing the last time.


struct example
name: tab
type: unsigned short int
name: a
type: u16
name: a
type: uint16_t
name: a
type: unsigned short int

VISITOR

WHAT DO WE NEED ?

- Give depth traversing control to the user.
- The user must be able to hook everywhere.
- By default depth traversing and printing functions.

HOW CAN WE DO THAT ?

- Each node has a default depth traversing function.
- We store printing functions in a hash table.
- Basic types must have a default printing function.
- The traversing is triggered by the printing function.

NODE STRUCTURE

```c
struct Die {
   const char *name;
   const char *type;
   void *data;
   size_t len;
   struct list children;
   struct list sibling;
   void (*next)(struct Die *, struct hash_control *);
};
```

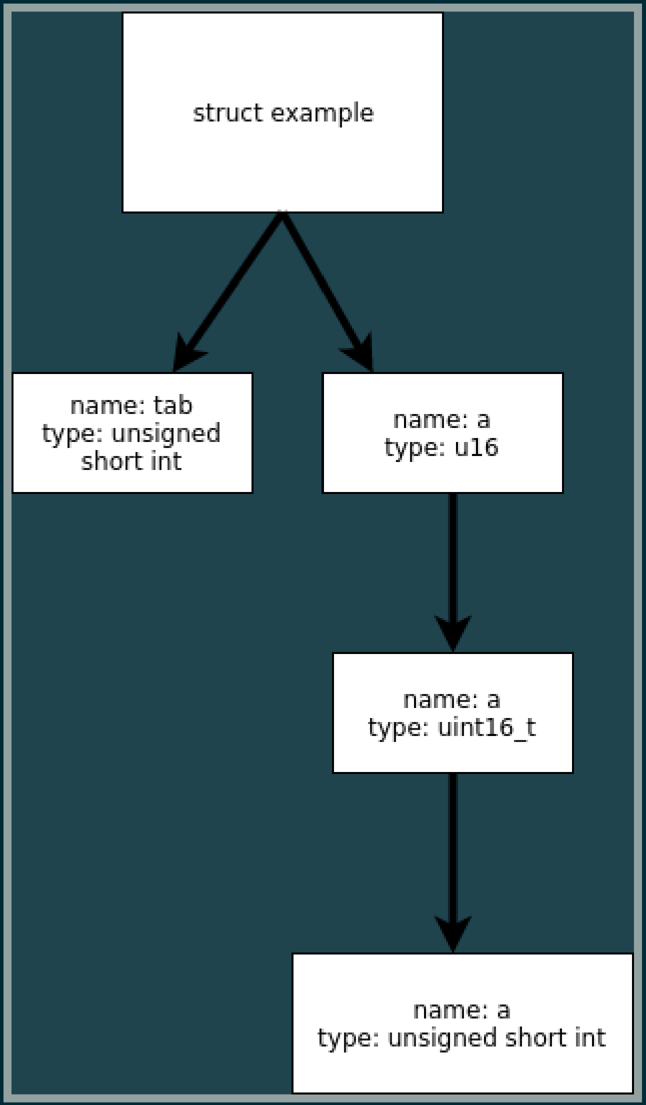
This is the structure given to the user

LET'S TRY IT !

THE TEST STRUCURE

```
typedef uint16_t u16;

struct example {
  unsigned short int a;
  u16 b;
};
```


struct example
name: tab
type: unsigned
short int
name: a
type: u16
name: a
type: uint16_t
name: a
type: unsigned short int

```c
void *init_dwarf(char *path);
int print_structure_content(void *address, char *name, void *di,
                            void* ctx);
/*
 * param 1 : ctx may be NULL the first time the function is calle
 * param 2 : name may be NULL if no type specification
 * param 3 : musn't be NULL
 * param 4 : print function
 *   * param 1 : Current Die
 *   * param 2 : User data
 *   * param 3 : Must be sent to next() function if called
 * param 5 : user data
 */
void *set_context(void *ctx, const char *name, const char *type,
                  void (*print)(Die *, void *, void *), void *dat
void print_uint16(Die *die, void *data, void *h) {
    printf("%s %s : %d\n", die->type, die->name,
    *(unsigned short int *)die->data);
}

int main(void) {
        struct example t;
        t.a = 1234;
        t.b = 42;
        void *te = init_dwarf("/proc/self/exe");
        void *ctx = NULL;
        ctx = set_context(NULL, NULL, "uint16_t", print_uint16,
        NULL);
        print_structure_content(&t, "example", te, ctx);
}
```

BY DEFAULT

```
struct example : t
short unsigned int b : 42
short unsigned int a : 1234
```

WITH OUR HOOK

```
struct example : t
uint16_t b : 42
short unsigned int a : 1234
```

What's next?

- Recursive parsing
- Parse faster when given a compile unit.

CONCLUSION

QUESTIONS?