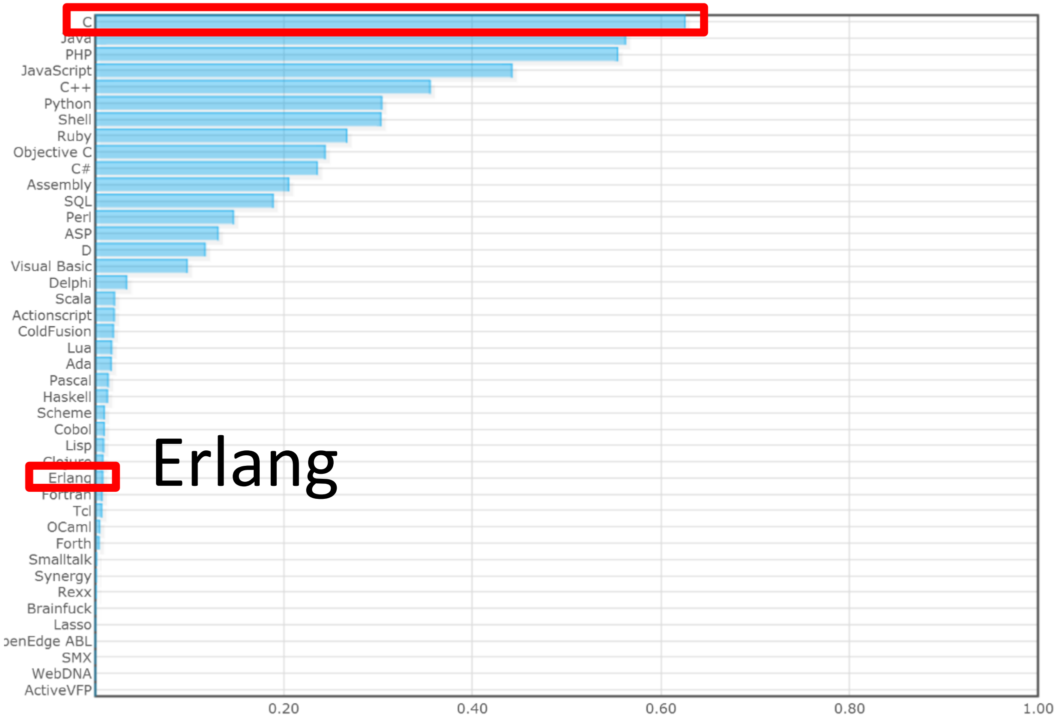
The Nifty Way to Call Hell from Heaven

ANDREAS LÖSCHER AND KONSTANTINOS SAGONAS

UPPSALA UNIVERSITY

There is a lot of C Code out there

C



Erlang

Source:
www.langpop.com
(Normalized statistic)

How can we use C libraries?

Port-Driver:

- C program connected over stdin/stdout

C Nodes:

- Socket communication

Native Implemented Functions (NIF)

- Shared library loaded by the VM
- Fast and dangerous

Simple Example:

Header File: heaven.h

```
extern int heaven(int value);
```

Source File: heaven.c

```
#include “heaven.h“

int
heaven(int value)
{
   return value + 42;
}
```

We need to write NIF:

- Translate argument
- call **heaven**()
- return result

Simple Example:

```c
static ERL_NIF_TERM nif_heaven(ErlNifEnv* env, int argc,
                               const ERL_NIF_TERM argv[]) {
  int err;
  int retval;
  int arg;
  err = enif_get_int(env, argv[0], &arg);
  if (!err) {
    return enif_make_badarg(env);
  }
  retval = heaven(arg);
  return enif_make_int(env, retval);
}
```

Simple Example:

```
static ErlNifFunc nif_functions[] = {
  {“heaven", 1, nif_heaven}
};

ERL_NIF_INIT(heaven, nif_functions, NULL, NULL, NULL, NULL);
```

Simple Example:

```
-module(heaven).
-export([heaven/1]).
-on_load(init/0).

init() ->
    ok = erlang:load_nif("heaven_nif", 0).

heaven(_) ->
    erlang:nif_error(nif_library_not_loaded).
```

Something more complex:

```
#include <stddef.h>
struct snappy_env {
  unsigned short *hash_table;
  void *scratch;
  void *scratch_output;
};

int
snappy_compress(struct snappy_env *env, const char *input,
                size_t input_length, char *compressed,
                size_t *compressed_length);
```

Something more complex:

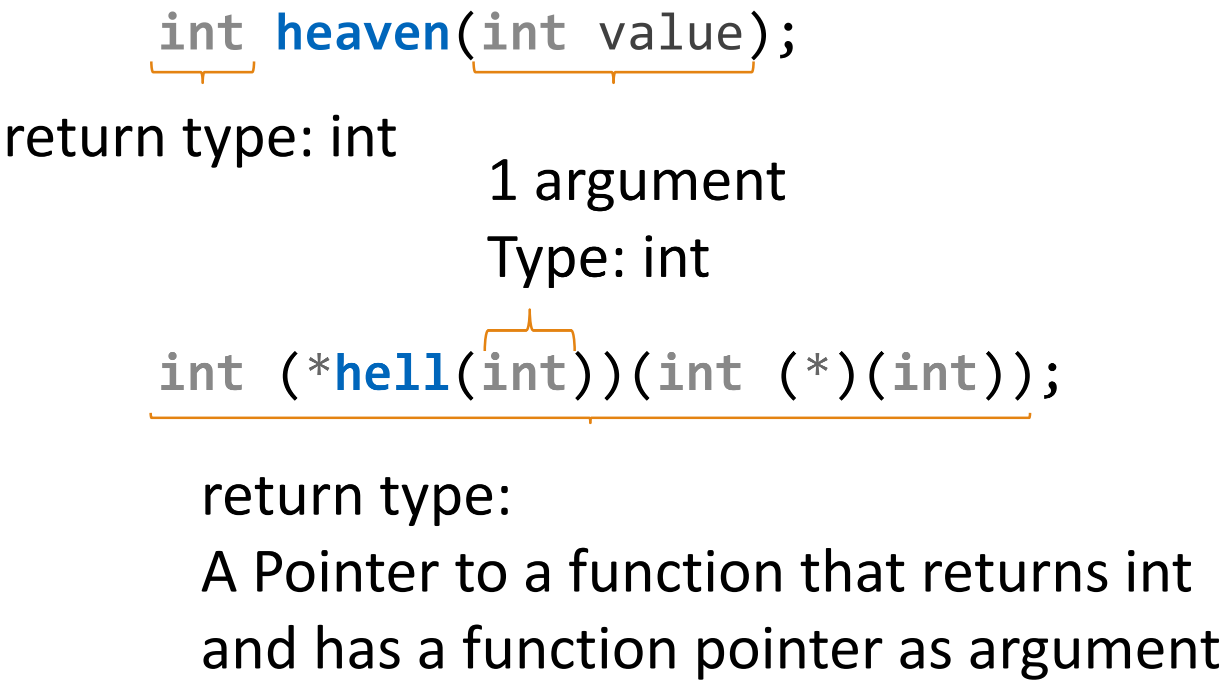
We need a lot of coding to wrap those functions.

Most of the code is tedious to write.

What is a C struct or a pointer in Erlang?

Let’s automate it!

All the information we need is given by the function declarations in header files:



```
int heaven(int value);
```

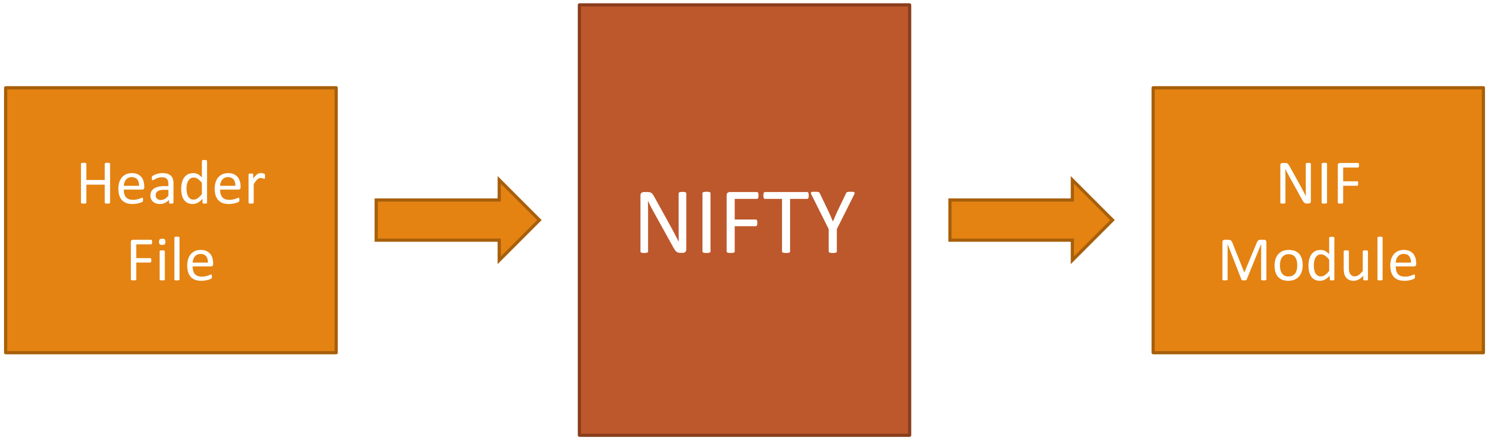
return type: int

1 argument
Type: int

```
int (*hell(int))(int (*)(int));
```

return type:
A Pointer to a function that returns int
and has a function pointer as argument

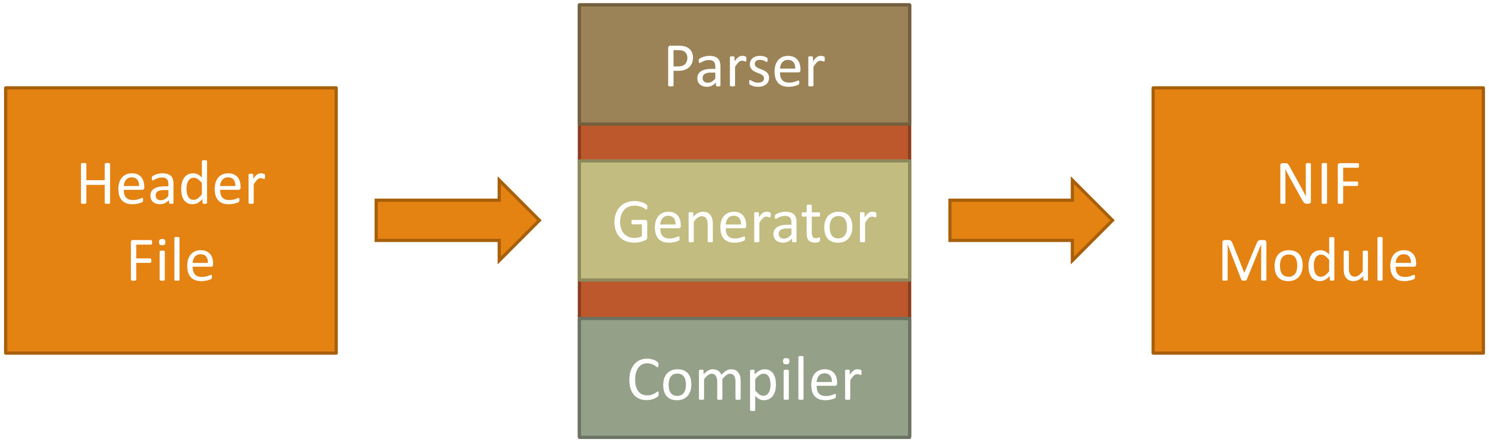
Nifty



Nifty - Heaven

```
1> nifty:compile(
     "heaven.h",
     heaven,
     nifty_utils:add_sources(["heaven.c"], [])).
...
ok
2> heaven:heaven(12).
54
```

Nifty



Header File

Parser

Generator

Compiler

NIF Module

Nifty – Parsing the Header

~~Implement a C Parser?~~

```
int (*hell(int))(int (*)(int));
```

LibClang

- High-Level Interface to the clang compiler
- Gives us access to the Abstract Syntax Tree
- Handles most of the uglyness of C

The AST is processed to a:

- Type Table
- Symbol Table

Type Table

Type	Entry
”char *”	{base, [”*”, ”char”, ”signed”, ”none”]}
”unsigned long”	{base, [”int”, ”unsigned”, ”long”]}
”size_t”	{typedef, ”unsigned long”}
”size_t *”	{userdef, [”*”, ”size_t”]}
”struct snapp_env”	{userdef, [{struct, ”snappy_env”}]}
…	…

- Base Types
- Typedefs
- User Defined Types

Type Table

Information about the used types

- Specifiers
- Pointers
- Structs

This type information is needed for:

- NIF module generation
- Usage of the NIF module

Symbol Table

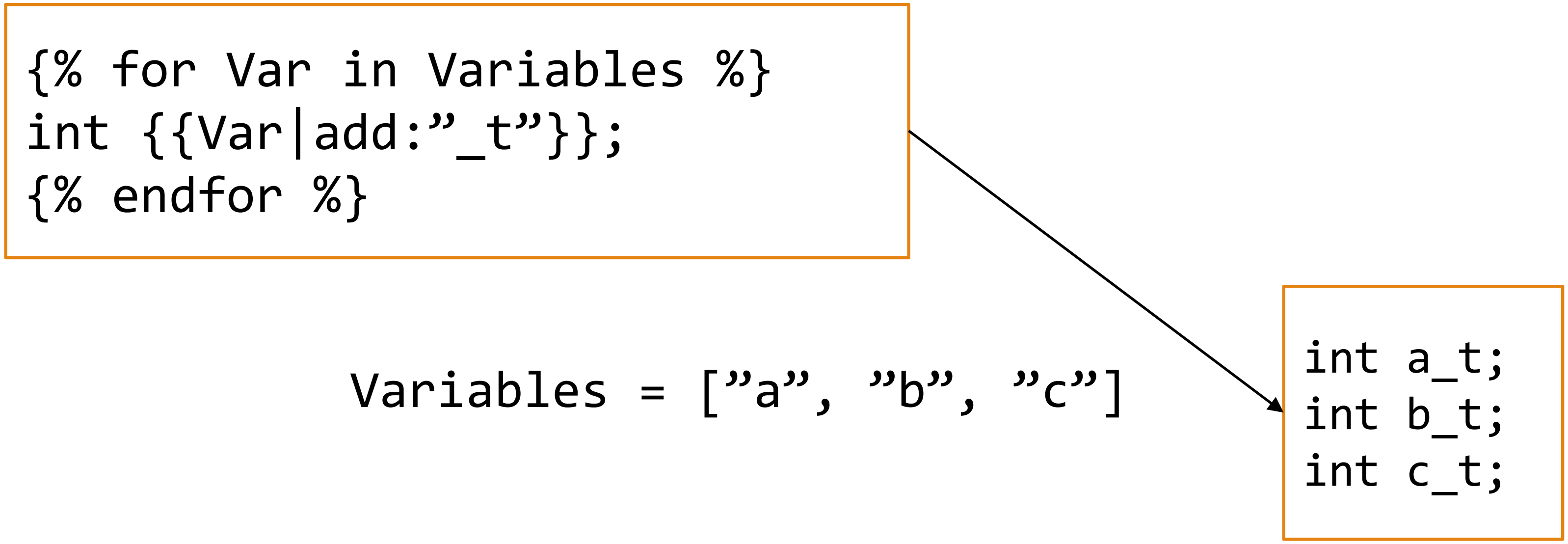
Symbol	Entry
”snappy_compress”	[{return, ”int”}, {argument, 0, ”struct snappy_env *”}, {argument, 1, ”const char *”}, …]

- Return Type
- Argument Type
- Argument Position

Nifty – Code Generation

ErlyDTL

- Django Template Language (DTL)
- Static and Variable parts can be described together

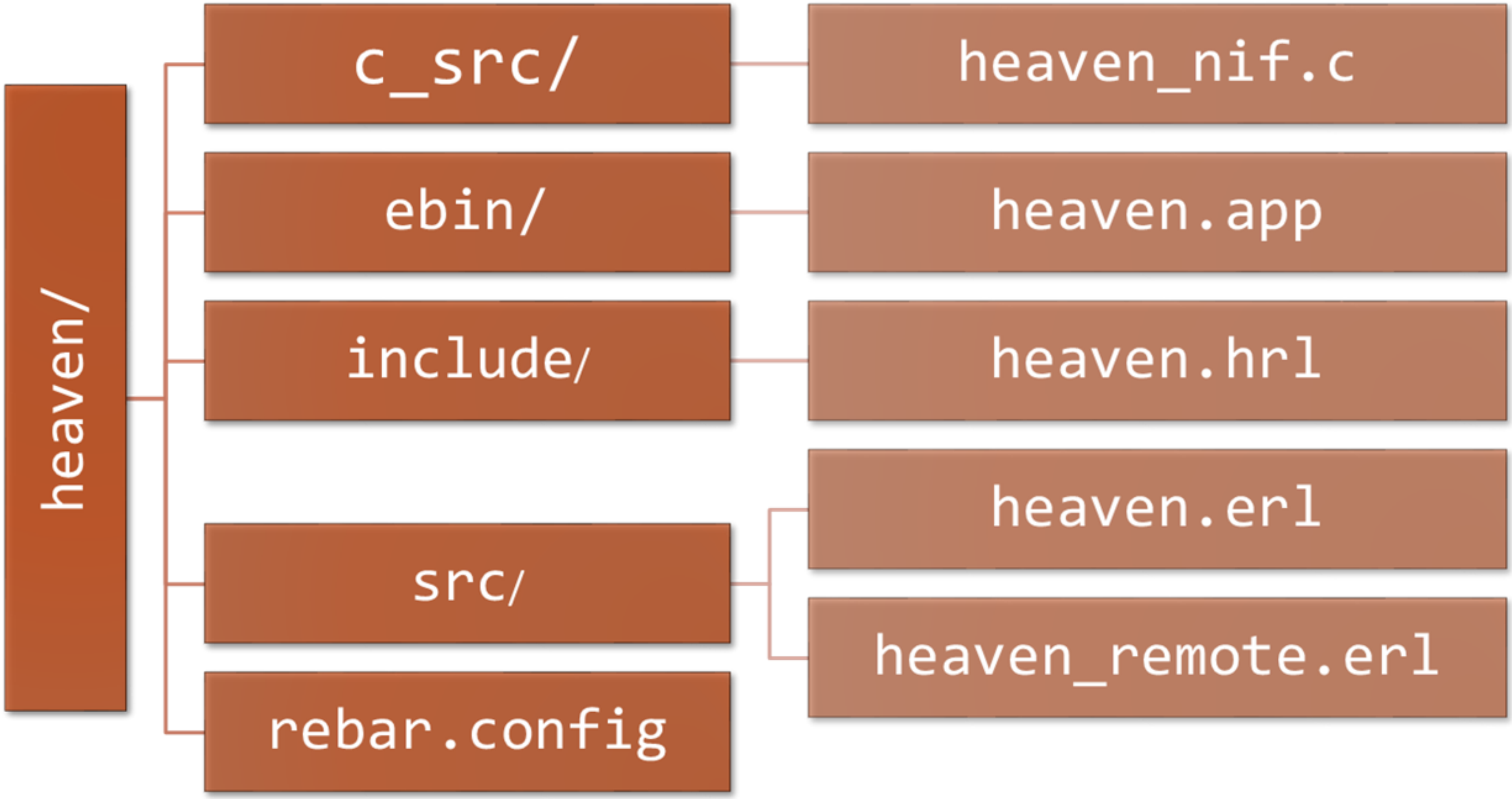


```
{% for Var in Variables %}
int {{Var|add:”_t”}};
{% endfor %}
```

```
Variables = [”a”, ”b”, ”c”]
```

```
int a_t;
int b_t;
int c_t;
```

Nifty - Package



Nifty – Compiling

Using an Erlang build tool:

- Rebar

Nifty generates:

- rebar.config
- Application resource file

Compiling is as ”simple” as ”`rebar compile`”

Nifty - Hell

```
1> nifty:compile(“hell.h", hell, …).
...
ok
2> hell:hell().
Segmentation fault (core dumped)
```

Buggy C Code crashes the VM!!

Nifty - Safe Execution

```
2> hell_remote:start().
ok
(p33_p0_master@cola-light) 3> hell_remote:hell().
** exception error: node_crashed
```

- NIF module gets loaded in a dedicated Erlang Node
- The error crashes only this Node

Limitations

Enums

Anonymous and Nested Structs

- Not implemented
- Each definition can be written in an equivalent way that is supported

```
typedef struct {
  struct nested {
    int x;
  } f1;
} my_struct;
```

```
struct nested {
  int x;
};
typedef struct my_struct {
  struct nested f1;
} my_struct;
```

Limitations

Function Pointers

- Are translated into void*
- There is no support in the NIF library of OTP to call Erlang functions from C

Variable Argument Lists

- ”…” is seen as no additional argument (printf/1)
- Functions using va_list can not be wrapped

Homepage

http://parapluu.github.io/nifty/

Nifty - Types

C	Erlang
int, long, short, char	integer()
float, double	float()
<type> *	{addr(), ”<module>.<type> *”}
struct name {…}	{name, …} (Erlang Record)