Command Line Interface / Application Programming Interface (cliapi)

Kevin Sheldrake
rtfc.org.uk

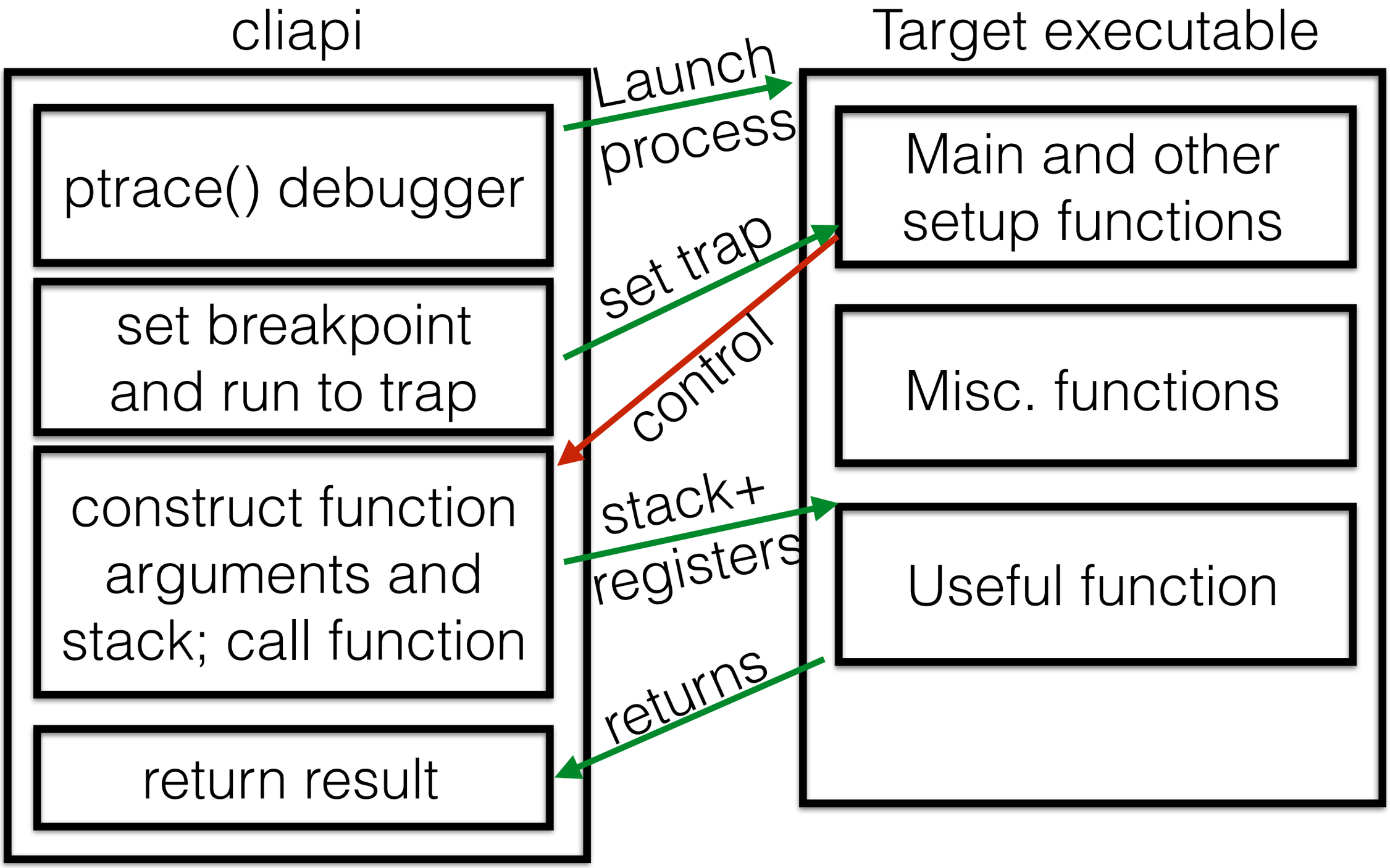
WTF?

- cliapi is a tool that runs individual functions in an executable or library on linux.
- Sometimes a function in an executable or library has some really useful functionality that you'd like to run out of context, but with your own arguments.
- You might only have a shell - no compiler, tool chain, python, perl, etc.
- cliapi is intended to solve this problem.

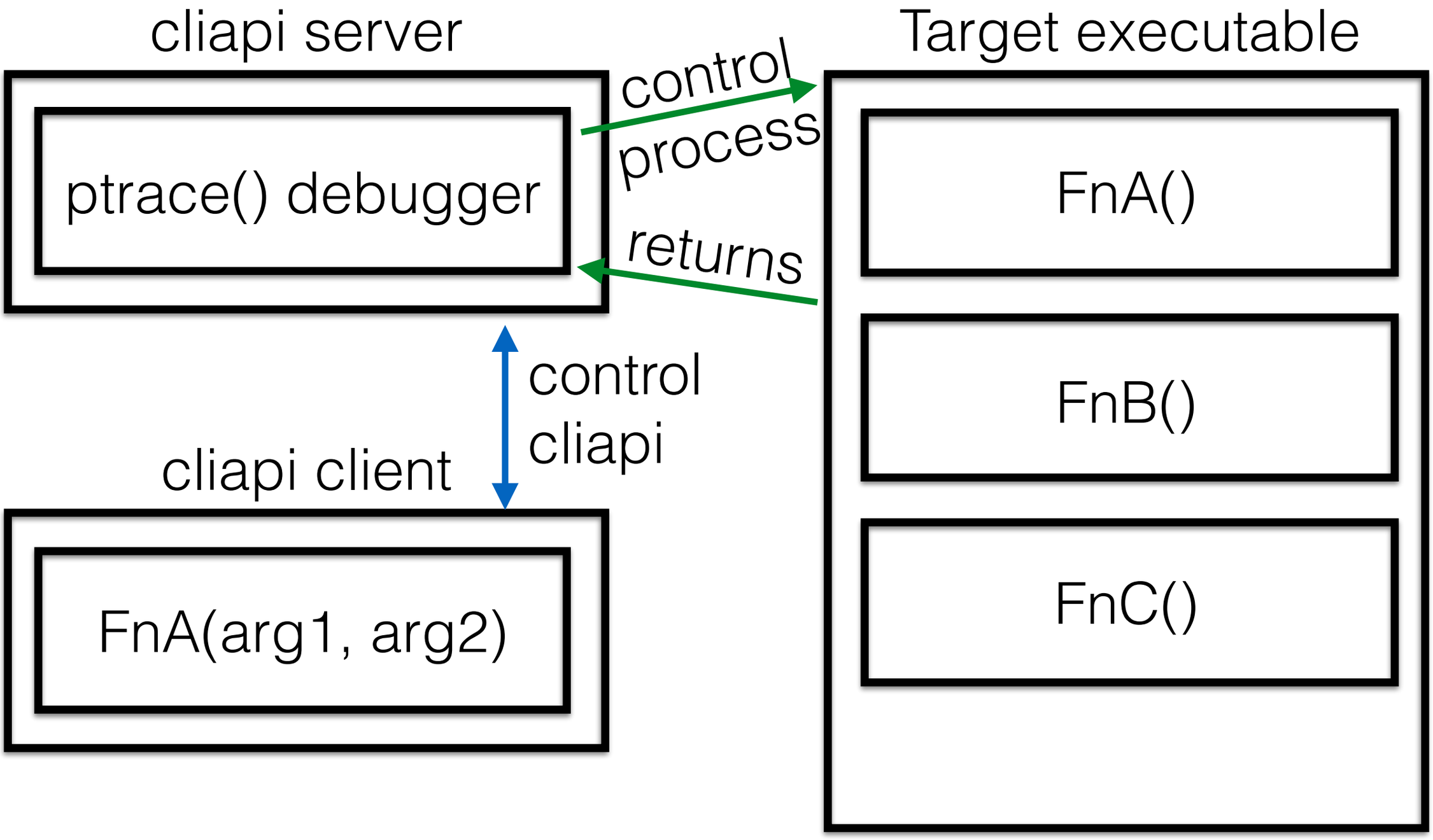
Inspiration

- Real embedded test.
- Device stored settings in the NVRAM and cli tools permitted reading and writing these settings.
- The password setting was encrypted using AES128-CBC with a key stored in one of the executables. The executable included functions to read and write the NVRAM plus special functions to read and write encrypted contents to it (using this key).
- A script could implement read/decrypt and encrypt/write but it was messy.
- Better solution: call the encrypted read/write functions directly.

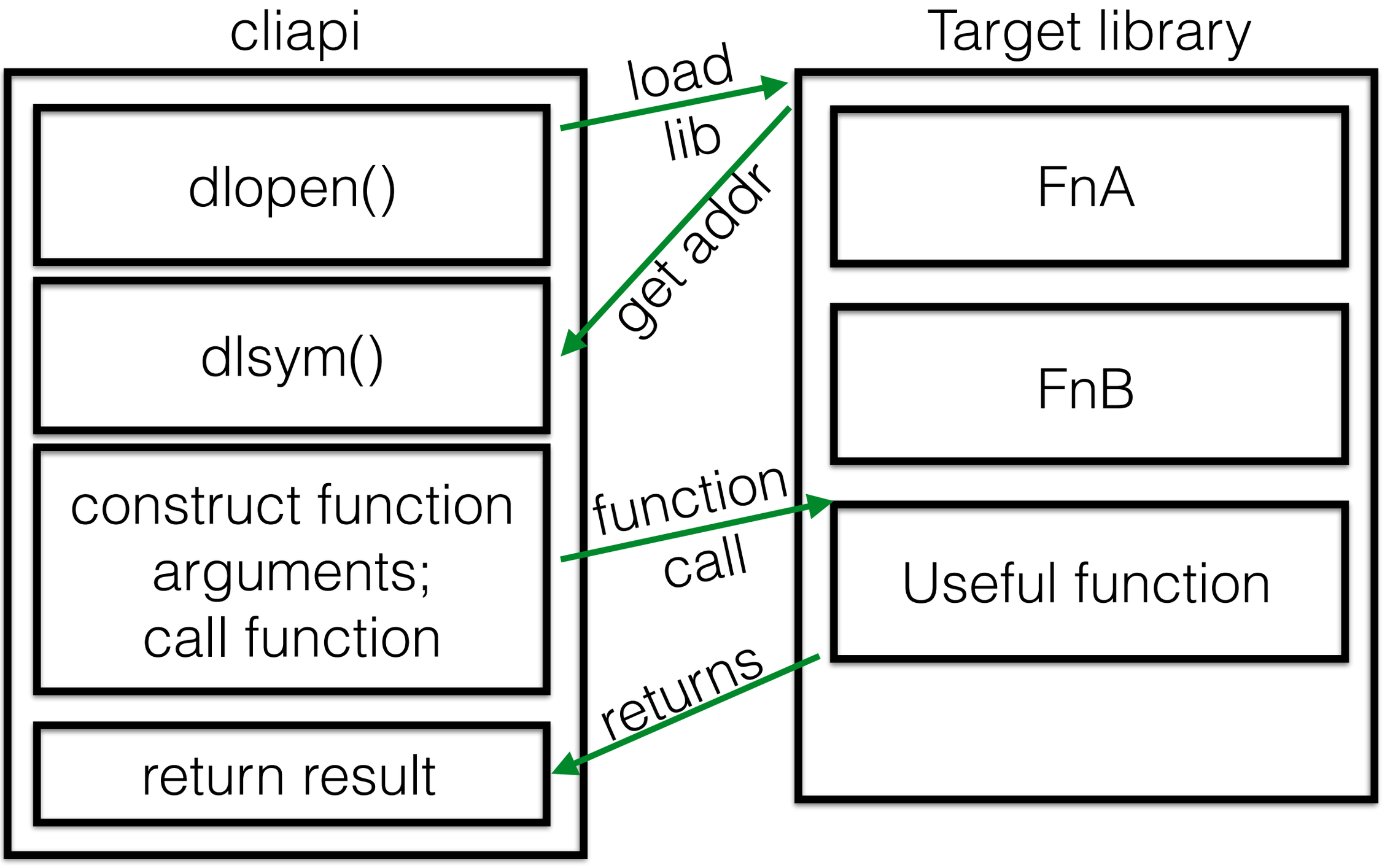
Example


cliapi
Target executable
ptrace() debugger
set breakpoint and run to trap
construct function arguments and stack; call function
return result
Launch process
set trap
control
stack+ registers
returns
Main and other setup functions
Misc. functions
Useful function

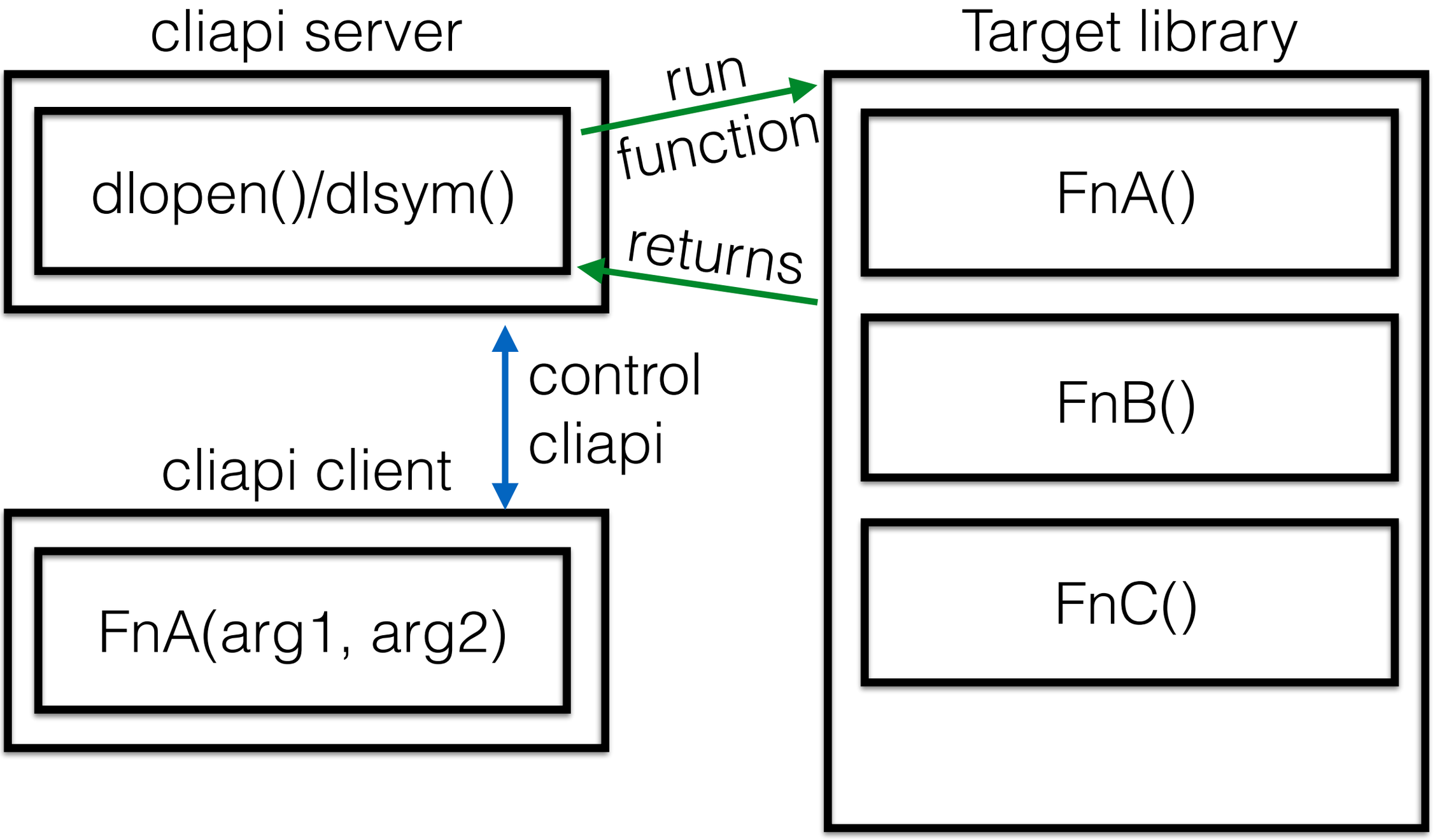
Pipe Server


cliapi server
ptrace() debugger
control
process
returns
Target executable
FnA()
FnB()
FnC()
control
cliapi
cliapi client
FnA(arg1, arg2)

With libraries too



Pipe Server for libraries



State of development

- x86 (32bit) version works for multi-threaded, multi-process executables and any shared libraries.
- Plans:
  - Port to ARMv6 and MIPS32.
  - Port to x64 and OS X.

Example

```
~/cliapi $ ./hello5
calling newfn
This is newfn: 5
after newfn
strfn, msg( 0x8048813 )='hello', msg2( 0x804880b )='goodbye'
output( 0x8cb2008 ) = 'hello - goodbye'
strfn returned 'hello - goodbye'
hello world

~/cliapi $ ./hello5 anyarg
calling newfn
This is newfn: 5
after newfn
strfn, msg( 0x8048813 )='hello', msg2( 0x804880b )='goodbye'
output( 0x98a1008 ) = 'hello - goodbye'
strfn returned 'hello - goodbye'
a = 0xbfdde0f8, b = 0x804882e
*a = 5, *b = 'hello world'
c = 11, d = 11
hello world
```

hello5.c (abridged)

```
int hidden(int *a, char *b) {

   printf("a = %p, b = %p\n", a, b);
   printf("*a = %d, *b = '%s'\n", *a, b);

   *a = strlen(b);

   return *a;
}
...
int main(int argc, char *argv[]) {

   int c = 5;
   int d;
...

   if (argc > 1) {

      d = hidden(&c, "hello world");
      printf("c = %d, d = %d\n", c, d);
   }
   printf("hello world\n");

   exit(0);
}
```

List functions

```
~/cliapi $ ./cliapi.0.6 -l ./hello5
0x08048460 LOCAL  deregister_tm_clones
0x08048490 LOCAL  register_tm_clones
0x080484d0 LOCAL  __do_global_dtors_aux
0x080484f0 LOCAL  frame_dummy
0x08048750 GLOBAL __libc_csu_fini
0x08048450 GLOBAL __x86.get_pc_thunk.bx
0x00000000 GLOBAL printf@@GLIBC_2.0
0x08048584 GLOBAL newfn
0x08048754 GLOBAL _fini
0x00000000 GLOBAL malloc@@GLIBC_2.0
0x00000000 GLOBAL puts@@GLIBC_2.0
0x00000000 GLOBAL system@@GLIBC_2.0
0x00000000 GLOBAL exit@@GLIBC_2.0
0x0804859f GLOBAL strfn
0x0804856c GLOBAL envfn
0x00000000 GLOBAL strlen@@GLIBC_2.0
0x00000000 GLOBAL __libc_start_main@@GLIBC_2.0
0x080486f0 GLOBAL __libc_csu_init
0x08048420 GLOBAL _start
0x08048635 GLOBAL main
0x00000000 GLOBAL sprintf@@GLIBC_2.0
0x0804851b GLOBAL hidden
0x08048350 GLOBAL _init
```

Run hidden function

```
int hidden(int *a, char *b) {

   printf("a = %p, b = %p\n", a, b);
   printf("*a = %d, *b = '%s'\n", *a, b);

   *a = strlen(b);

   return *a;
}
```

```
~/cliapi $ ./cliapi.0.6 -l ./hello5 -f hidden -I 8 -s "my string"
a = 0xbfad016c, b = 0xbfad0170
*a = 8, *b = 'my string'
run_controller ended
Returned int = 9
0: I 9
```

Usage

```
cliapi v0.6 K Sheldrake

Run functions in elf binaries
cliapi -l loadfile
  list functions
cliapi -l loadfile -f fn [-r type] [-v verbosity] [-i arg] [-s string] [-S len:string]
  -l path to elf executable or library
  -f function name
  -F function address
  -i integer argument
  -I pointer to integer argument
  -s pointer to string argument
  -S pointer to mutable string argument, specify length - -S len:string
  -h pointer to hexstring argument
  -H pointer to mutable hexstring argument, specify length - -H len:hexstring
  -b break function for exes (default is main())
  -B break address for exes (default is main())
  -D detach other processes and threads on break
  -r return value type; e.g. i, s or h
  -o output len for hexstring output
  -a cmdline argument for exe
  -n don't check for function
  -P path/to/pipe to listen on
  -p path/to/pipe to connect to
  -v verbose - use twice for more info
  -q quiet

Use -i, -I, -s, -S, -h, -H, -a multiple times to add extra arguments.
Numbers can be decimal (no prefix), hex (0x prefix) or octal (0 prefix)
Total string length must be specified for mutable strings.
Arguments specified with -I, -S or -H will be printed on exit.
```

Any Questions?

*

* Note: No Lenovo laptops were actually used in the making of this tool