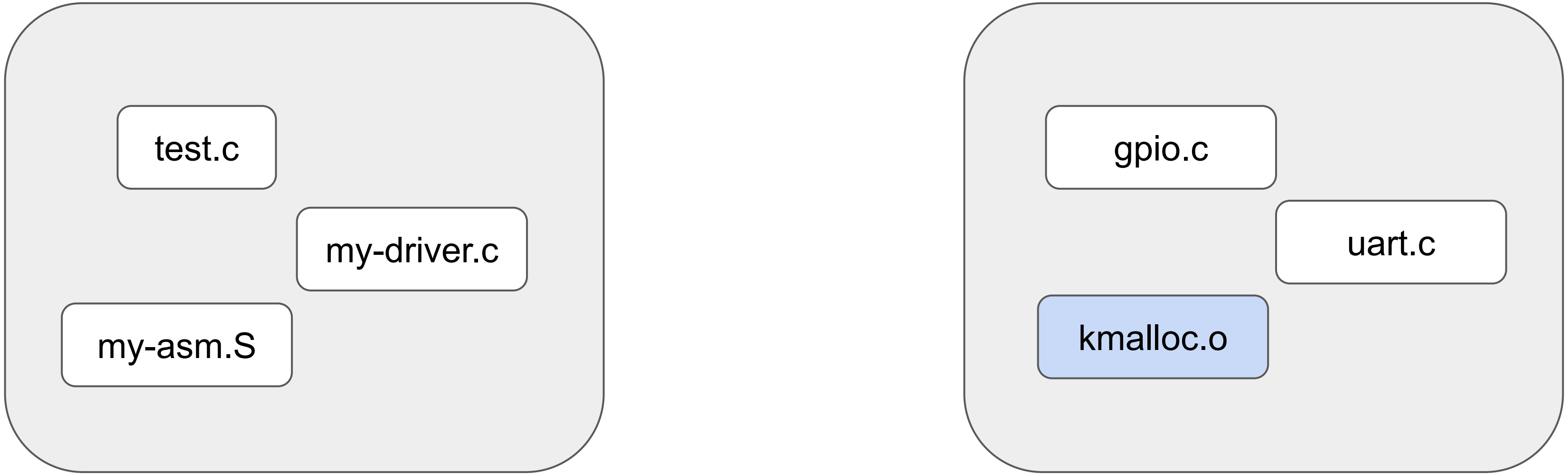
ELF & Dynamic Linker

CS 240LX

Motivation

How we ran programs on Pi so far



test.c

my-driver.c

my-asm.S

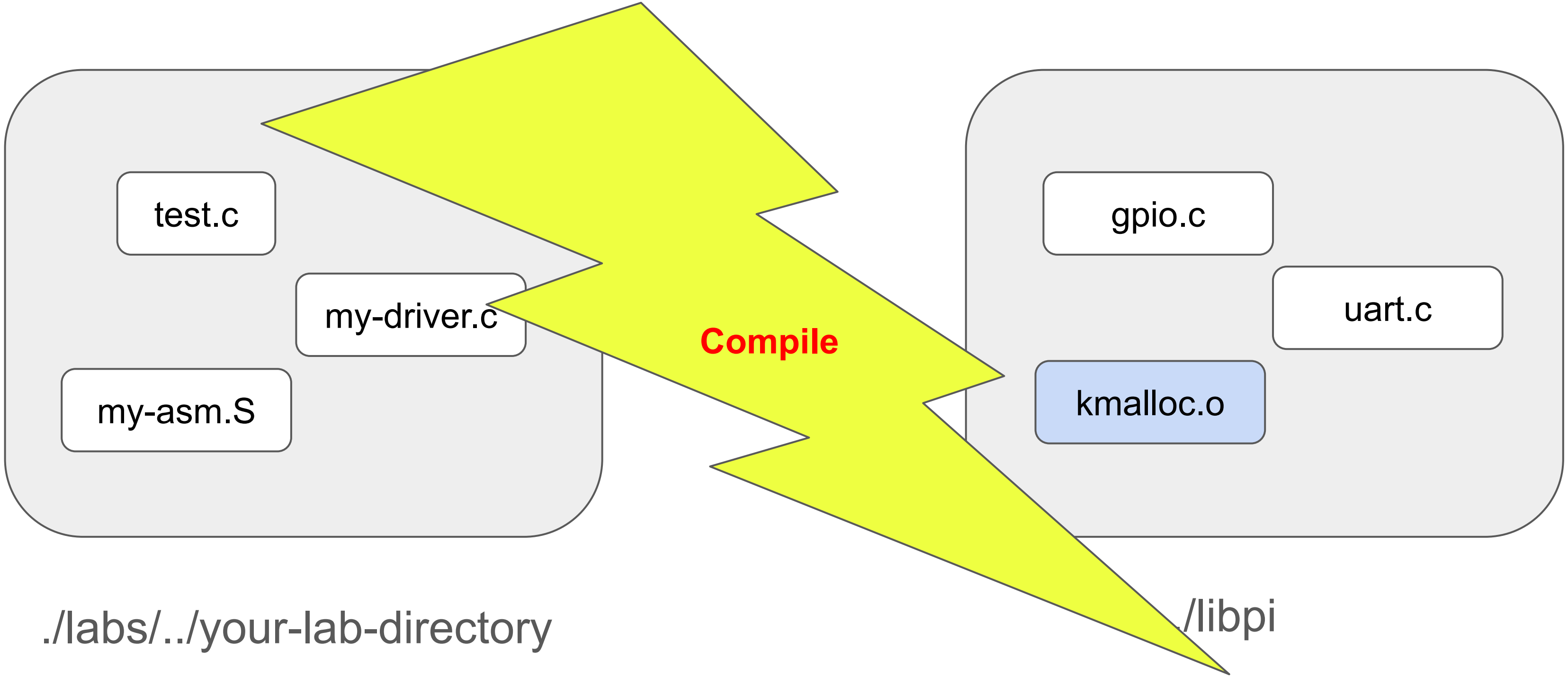
./labs/../your-lab-directory

gpio.c

uart.c

kmalloc.o

./libpi



test.c

my-driver.c

my-asm.S

Compile

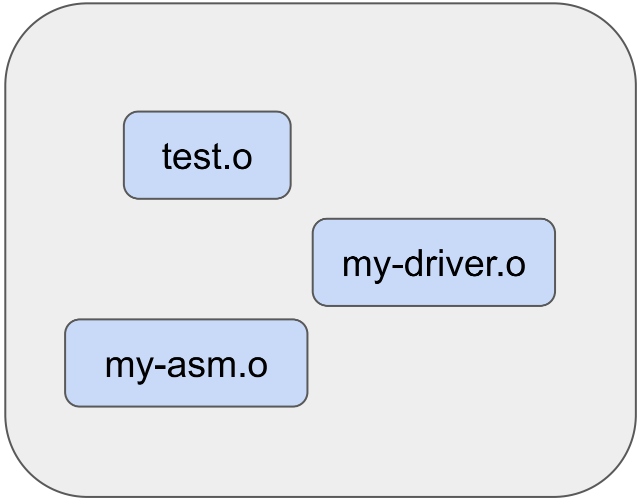
gpio.c

uart.c

kmalloc.o

./labs/../your-lab-directory

/libpi

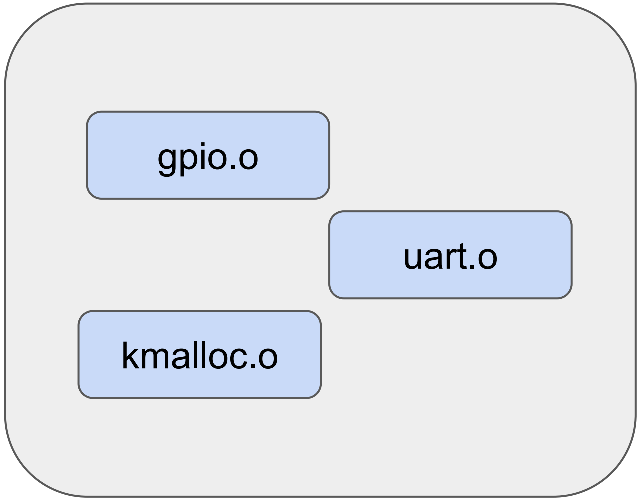


test.o

my-driver.o

my-asm.o

./labs/../your-lab-directory

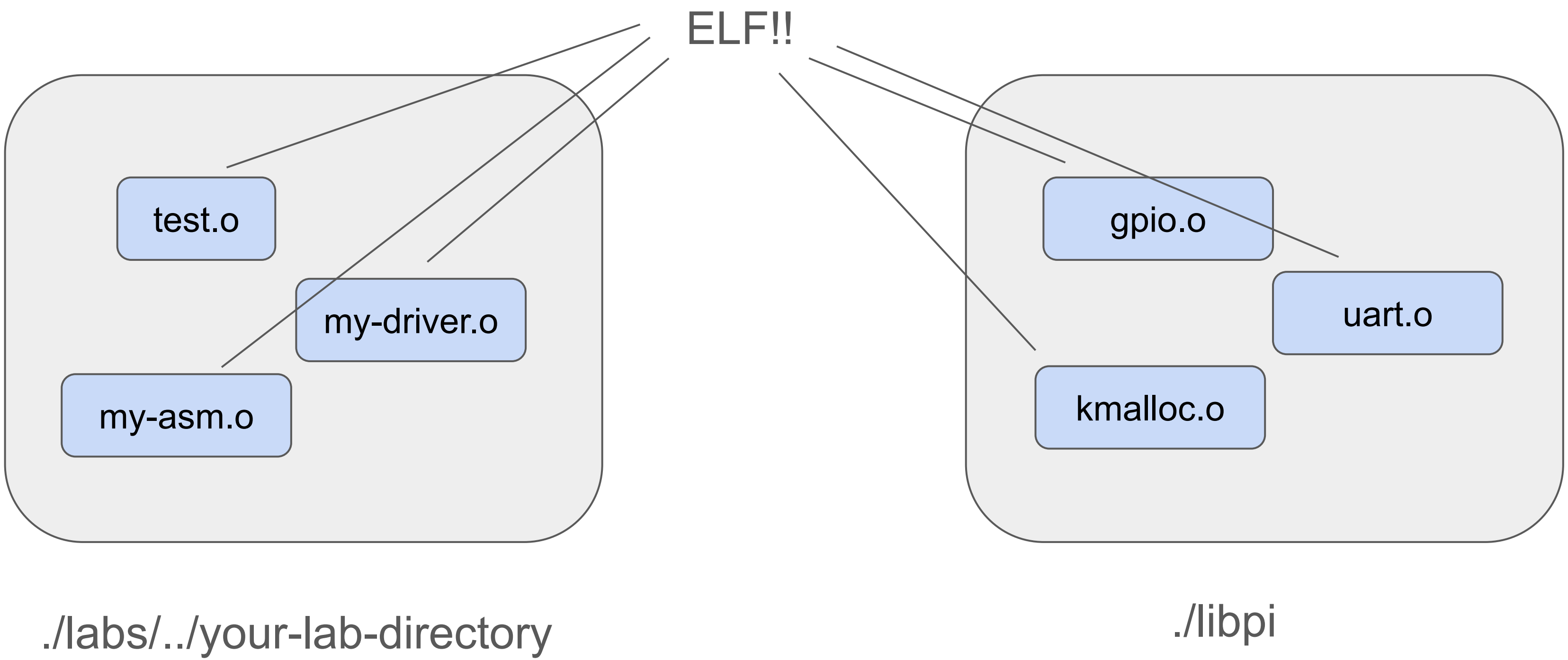


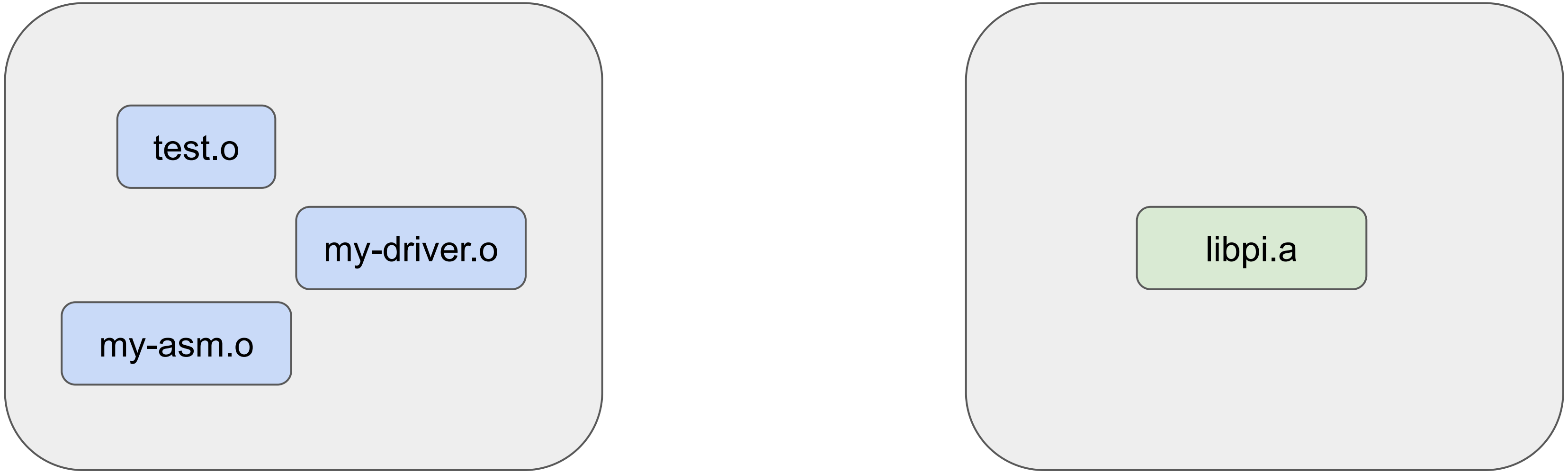
gpio.o

uart.o

kmalloc.o

./libpi


ELF!!
test.o
my-driver.o
my-asm.o
gpio.o
uart.o
kmalloc.o
./labs/../your-lab-directory
./libpi



test.o

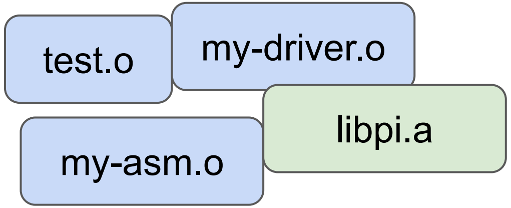
my-driver.o

my-asm.o

libpi.a

./labs/../your-lab-directory

./libpi



test.o

my-driver.o

libpi.a

my-asm.o

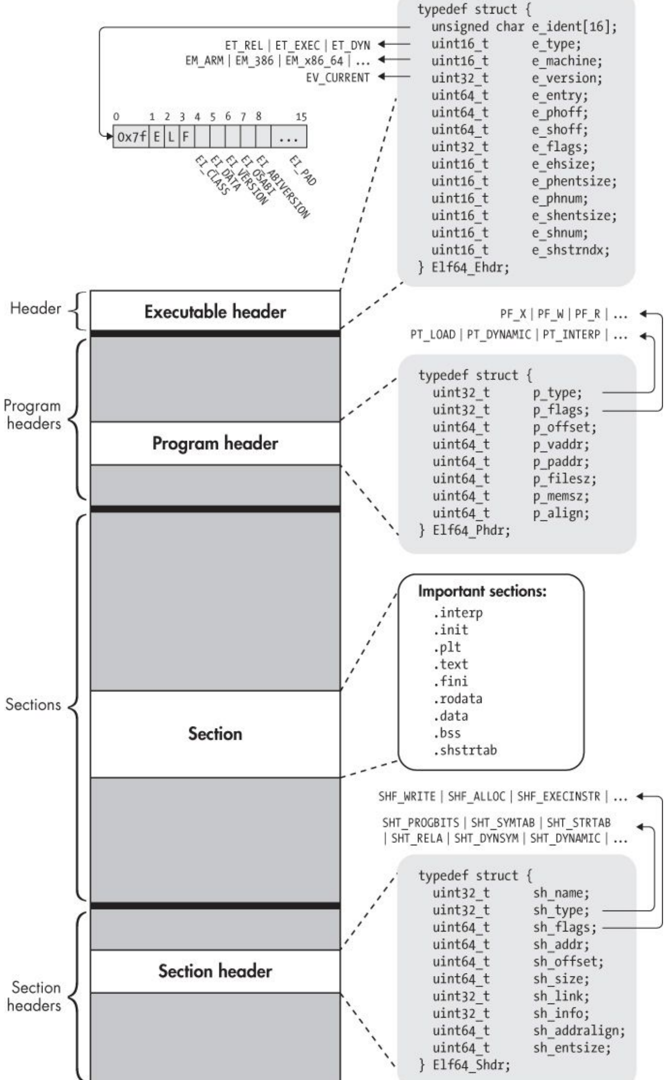
test

ELF Executable

test

ELF Executable

Up to this point, nothing different from
the “usual” way


ET_REL | ET_EXEC | ET_DYN
EM_ARM | EM_386 | EM_x86_64 | ...
EV_CURRENT
0 1 2 3 4 5 6 7 8 15
0x7f E L F ...
EI_CLASS
EI_DATA
EI_VERSION
EI_OSABI
EI_ABIVERSION
EI_PAD
Header
Executable header
Program headers
Program header
Sections
Section
Section headers
Section header
PF_X | PF_W | PF_R | ...
PT_LOAD | PT_DYNAMIC | PT_INTERP | ...
SHF_WRITE | SHF_ALLOC | SHF_EXECINSTR | ...
SHT_PROGBITS | SHT_SYMTAB | SHT_STRTAB | SHT_RELA | SHT_DYNSYM | SHT_DYNAMIC | ...

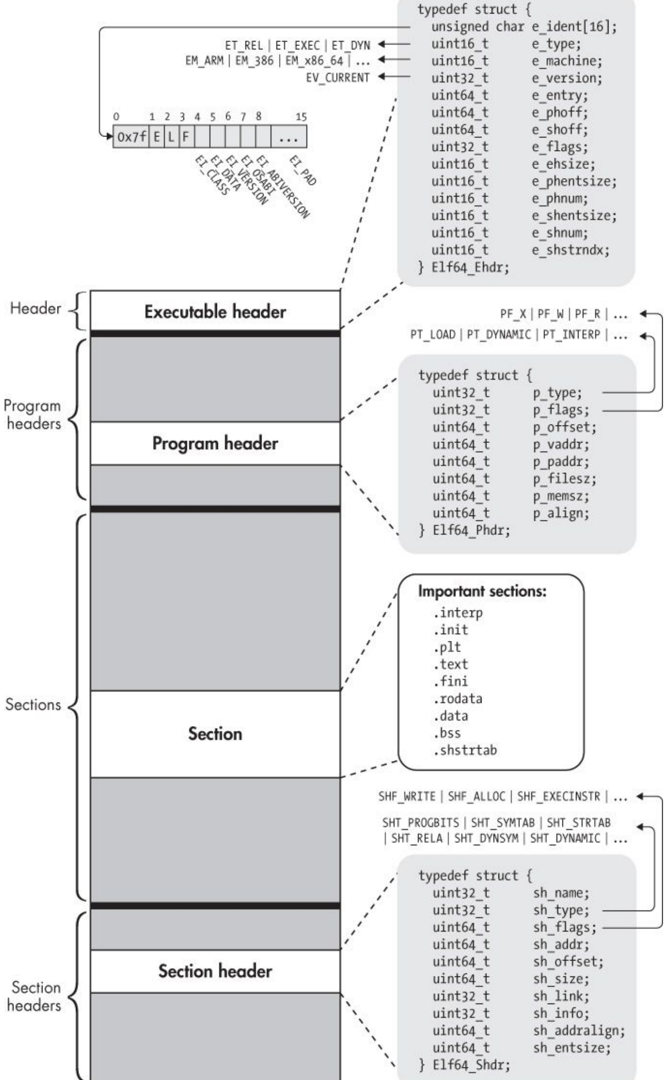
```
typedef struct {
  unsigned char e_ident[16];
  uint16_t      e_type;
  uint16_t      e_machine;
  uint32_t      e_version;
  uint64_t      e_entry;
  uint64_t      e_phoff;
  uint64_t      e_shoff;
  uint32_t      e_flags;
  uint16_t      e_ehsize;
  uint16_t      e_phentsize;
  uint16_t      e_phnum;
  uint16_t      e_shentsize;
  uint16_t      e_shnum;
  uint16_t      e_shstrndx;
} Elf64_Ehdr;
```

```
typedef struct {
  uint32_t      p_type;
  uint32_t      p_flags;
  uint64_t      p_offset;
  uint64_t      p_vaddr;
  uint64_t      p_paddr;
  uint64_t      p_filesz;
  uint64_t      p_memsz;
  uint64_t      p_align;
} Elf64_Phdr;
```

Important sections:

```
.interp
.init
.plt
.text
.fini
.rodata
.data
.bss
.shstrtab
```

```
typedef struct {
  uint32_t      sh_name;
  uint32_t      sh_type;
  uint64_t      sh_flags;
  uint64_t      sh_addr;
  uint64_t      sh_offset;
  uint64_t      sh_size;
  uint32_t      sh_link;
  uint32_t      sh_info;
  uint64_t      sh_addralign;
  uint64_t      sh_entsize;
} Elf64_Shdr;
```



Header
Executable header
Program headers
Program header
Sections
Section
Section headers
Section header
ET_REL | ET_EXEC | ET_DYN
EM_ARM | EM_386 | EM_x86_64 | ...
EV_CURRENT
0 1 2 3 4 5 6 7 8 15
0x7f E L F ...
EI_CLASS
EI_DATA
EI_VERSION
EI_OSABI
EI_ABIVERSION
EI_PAD
PF_X | PF_W | PF_R | ...
PT_LOAD | PT_DYNAMIC | PT_INTERP | ...
SHF_WRITE | SHF_ALLOC | SHF_EXECINSTR | ...
SHT_PROGBITS | SHT_SYMTAB | SHT_STRTAB
| SHT_RELA | SHT_DYNSYM | SHT_DYNAMIC | ...

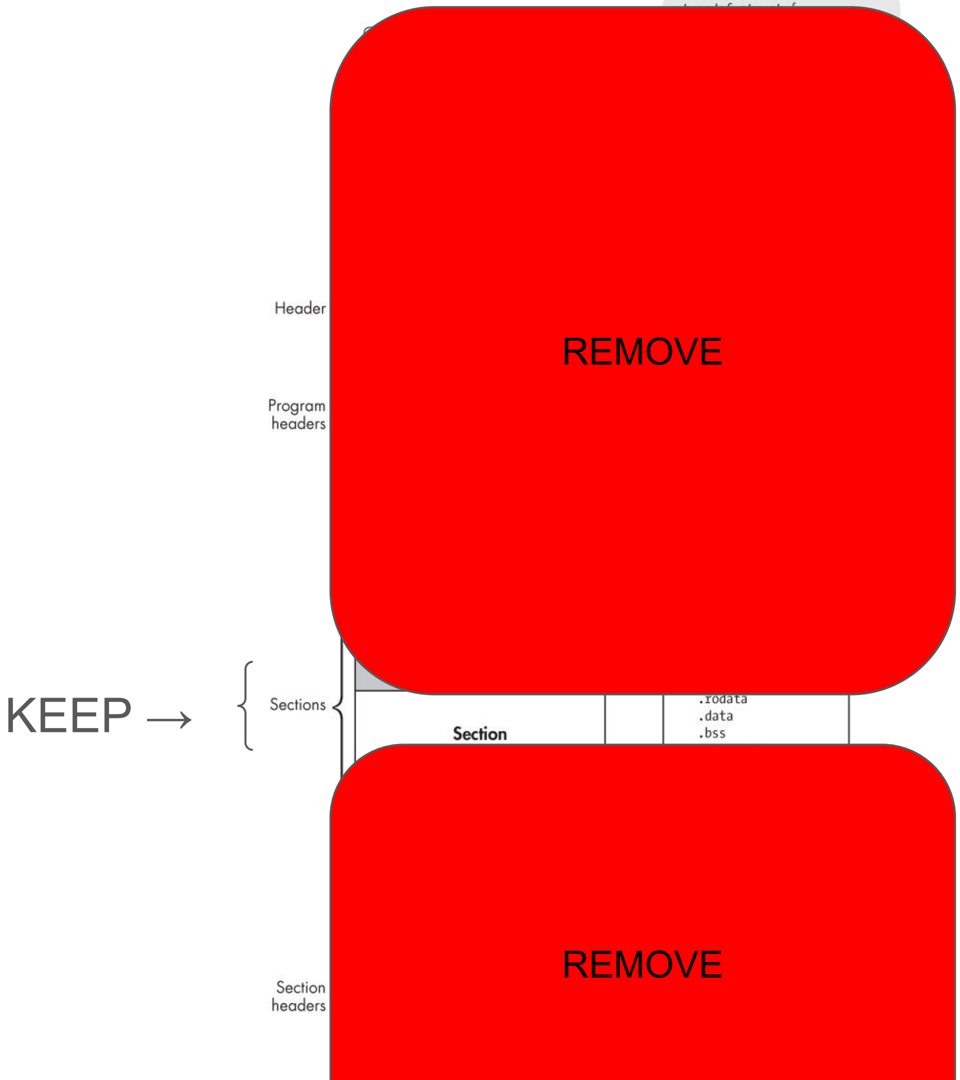
```
typedef struct {
  unsigned char e_ident[16];
  uint16_t      e_type;
  uint16_t      e_machine;
  uint32_t      e_version;
  uint64_t      e_entry;
  uint64_t      e_phoff;
  uint64_t      e_shoff;
  uint32_t      e_flags;
  uint16_t      e_ehsize;
  uint16_t      e_phentsize;
  uint16_t      e_phnum;
  uint16_t      e_shentsize;
  uint16_t      e_shnum;
  uint16_t      e_shstrndx;
} Elf64_Ehdr;
```

```
typedef struct {
  uint32_t      p_type;
  uint32_t      p_flags;
  uint64_t      p_offset;
  uint64_t      p_vaddr;
  uint64_t      p_paddr;
  uint64_t      p_filesz;
  uint64_t      p_memsz;
  uint64_t      p_align;
} Elf64_Phdr;
```

Important sections:

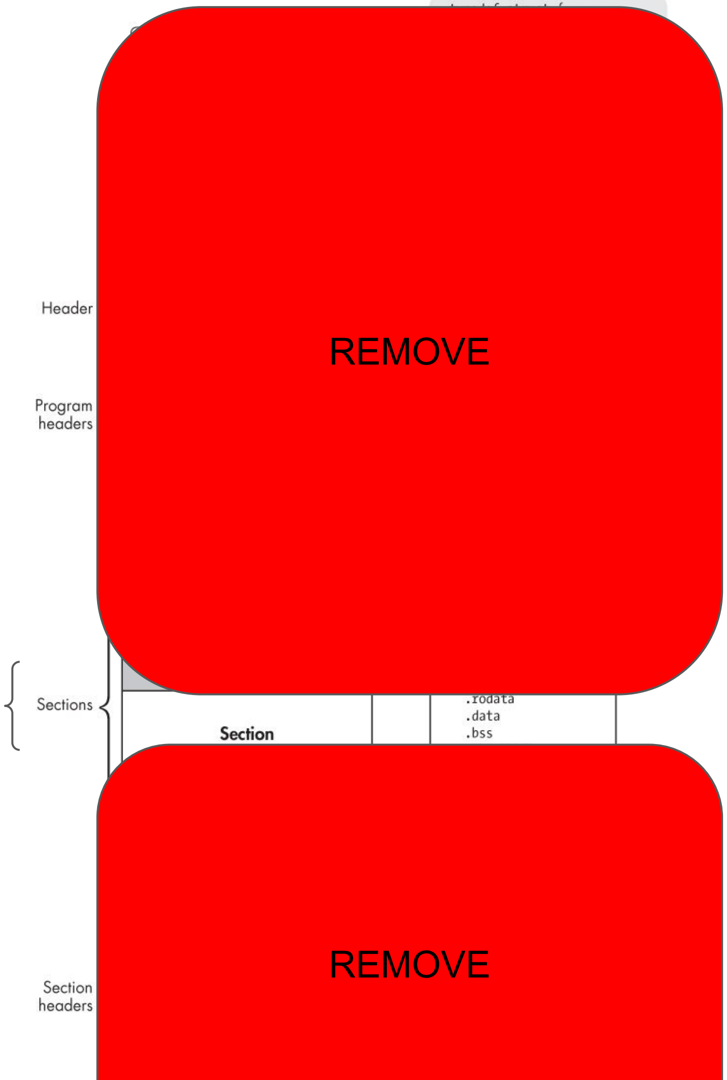
```
.interp
.init
.plt
.text
.fini
.rodata
.data
.bss
.shstrtab
```

```
typedef struct {
  uint32_t      sh_name;
  uint32_t      sh_type;
  uint64_t      sh_flags;
  uint64_t      sh_addr;
  uint64_t      sh_offset;
  uint64_t      sh_size;
  uint32_t      sh_link;
  uint32_t      sh_info;
  uint64_t      sh_addralign;
  uint64_t      sh_entsize;
} Elf64_Shdr;
```

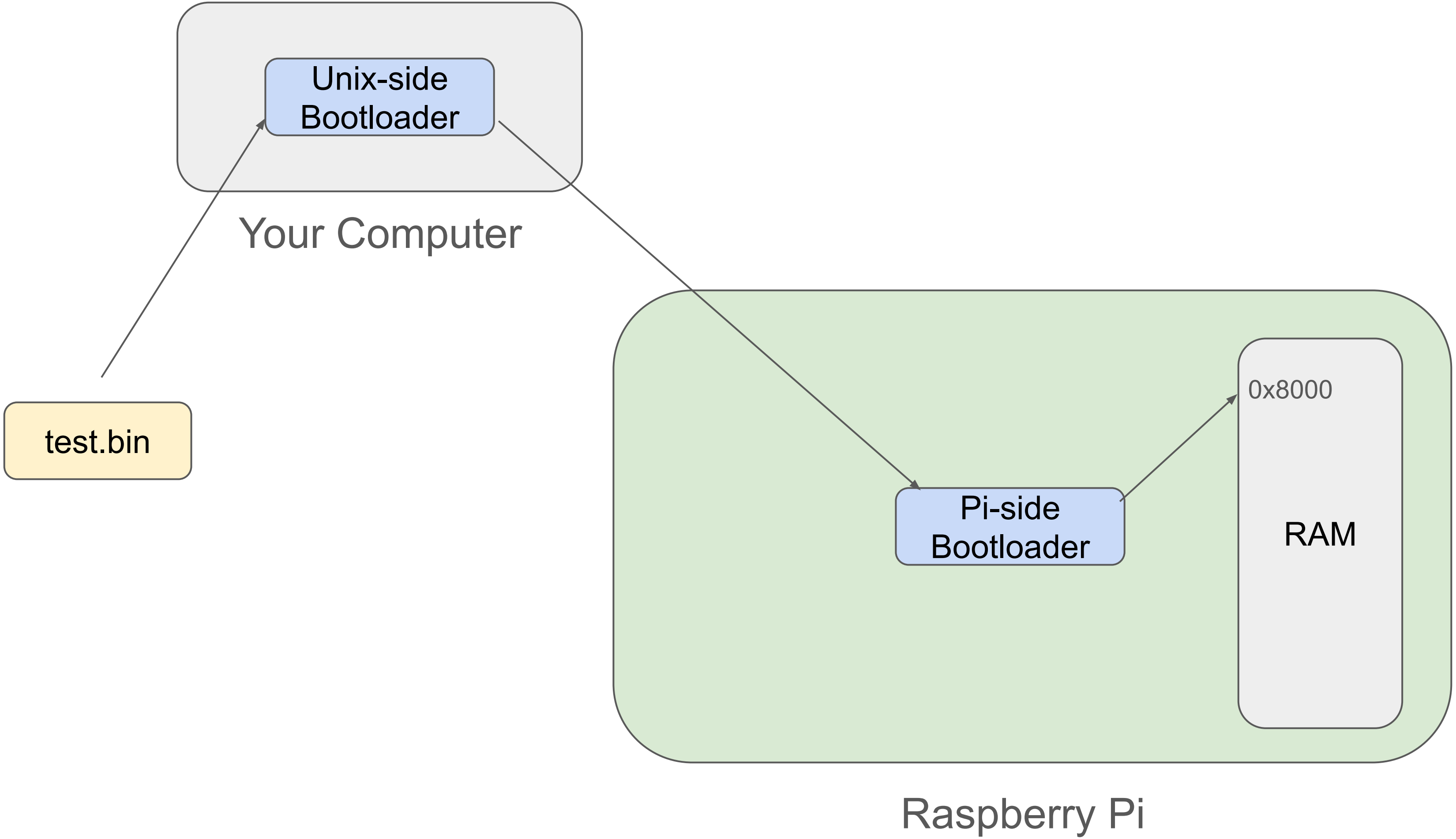

Header
Program headers
REMOVE
KEEP →
Sections
Section
.rodata
.data
.bss
REMOVE
Section headers

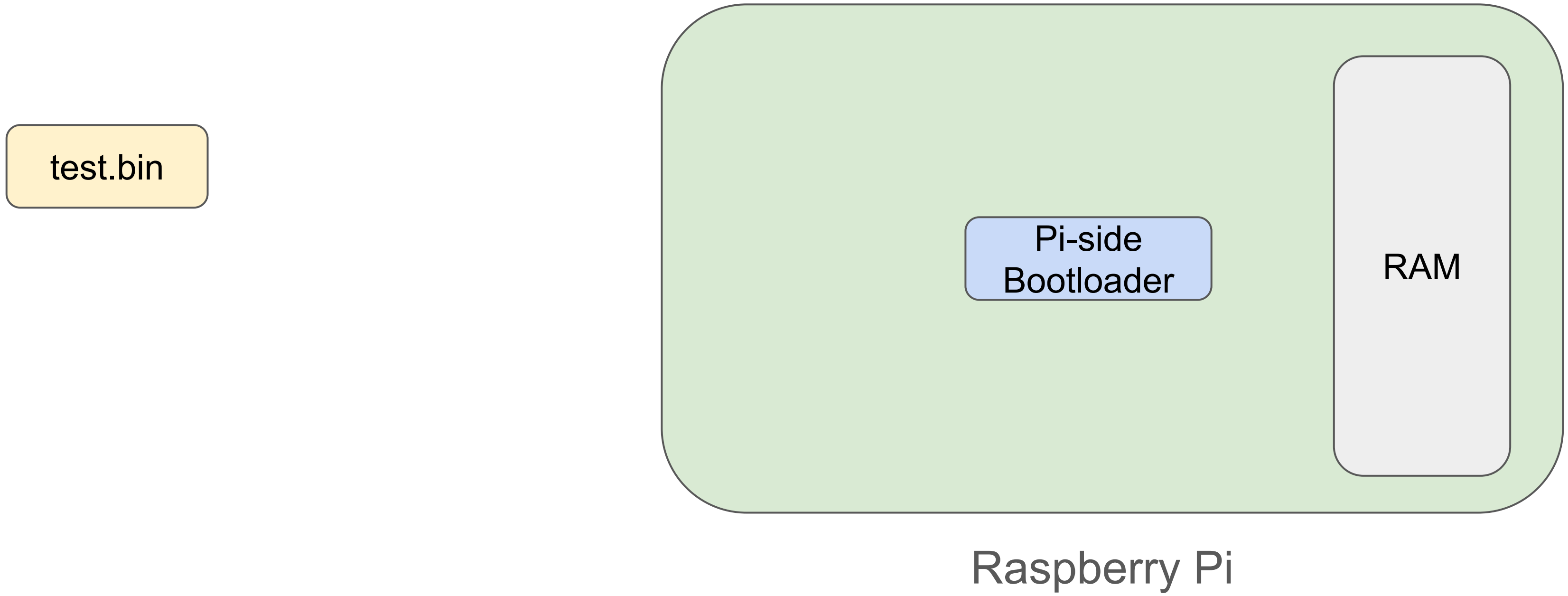
test.bin

Raw binary (just .text, .rodata, .data)



test.bin


Unix-side Bootloader
Your Computer
test.bin
Pi-side Bootloader
0x8000
RAM
Raspberry Pi



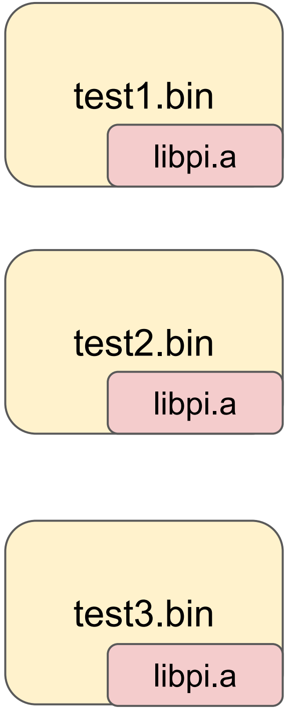
test.bin

Pi-side Bootloader

RAM

Raspberry Pi

Main Problem: Redundancy
(+ can’t run “normal” executables, can’t use external libraries, etc etc)


test1.bin

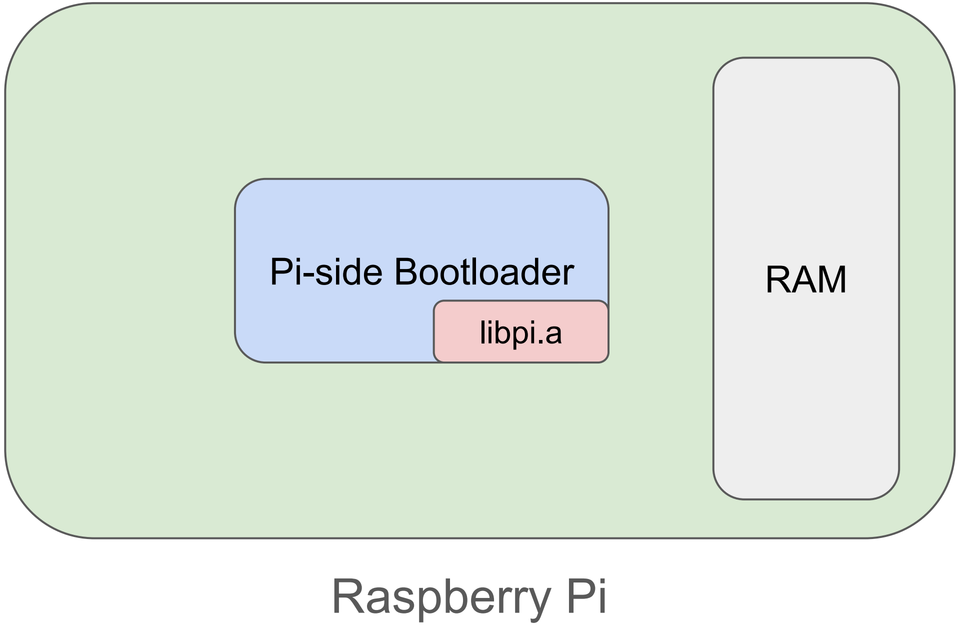
libpi.a

test2.bin

libpi.a

test3.bin

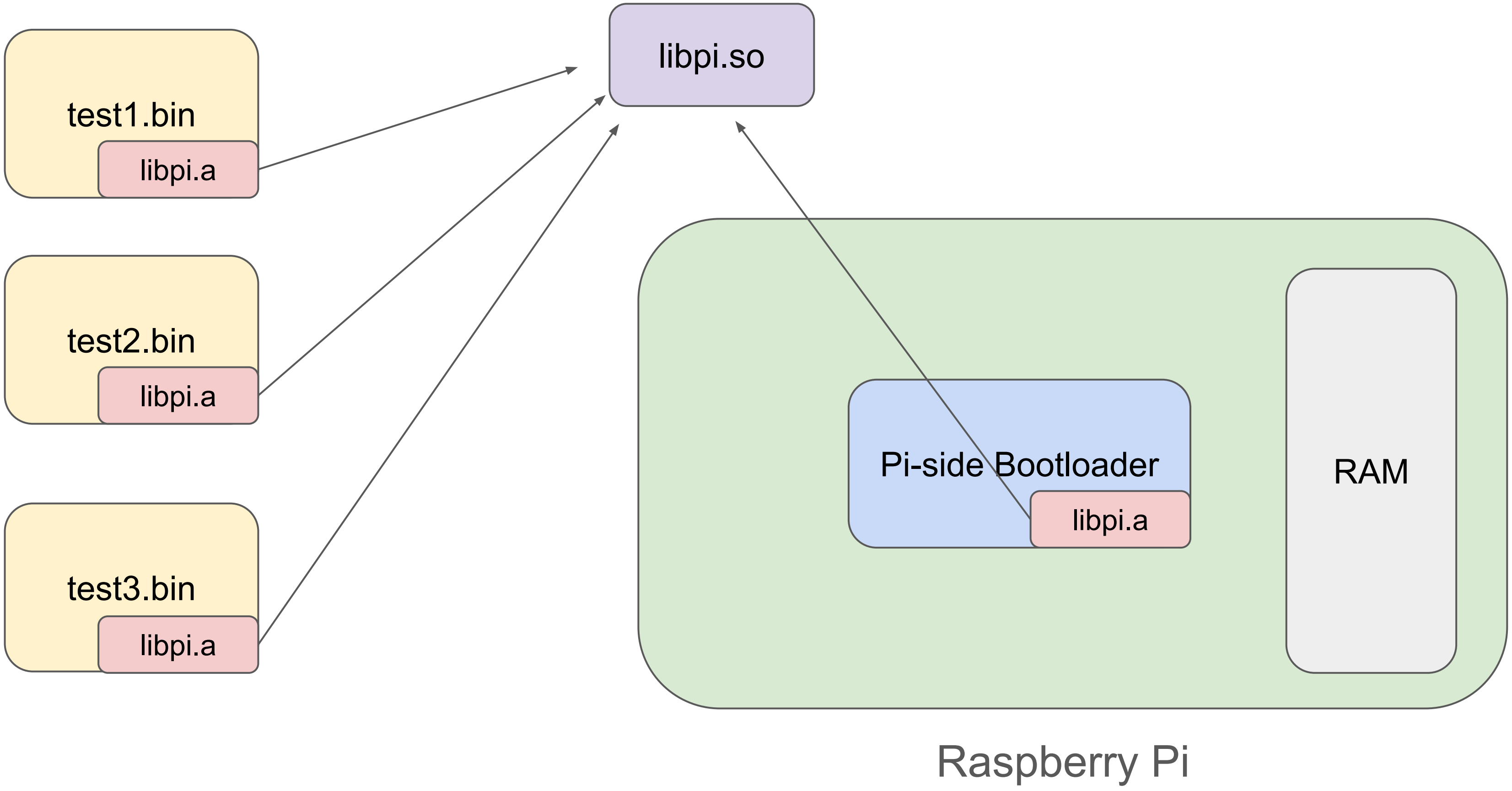
libpi.a

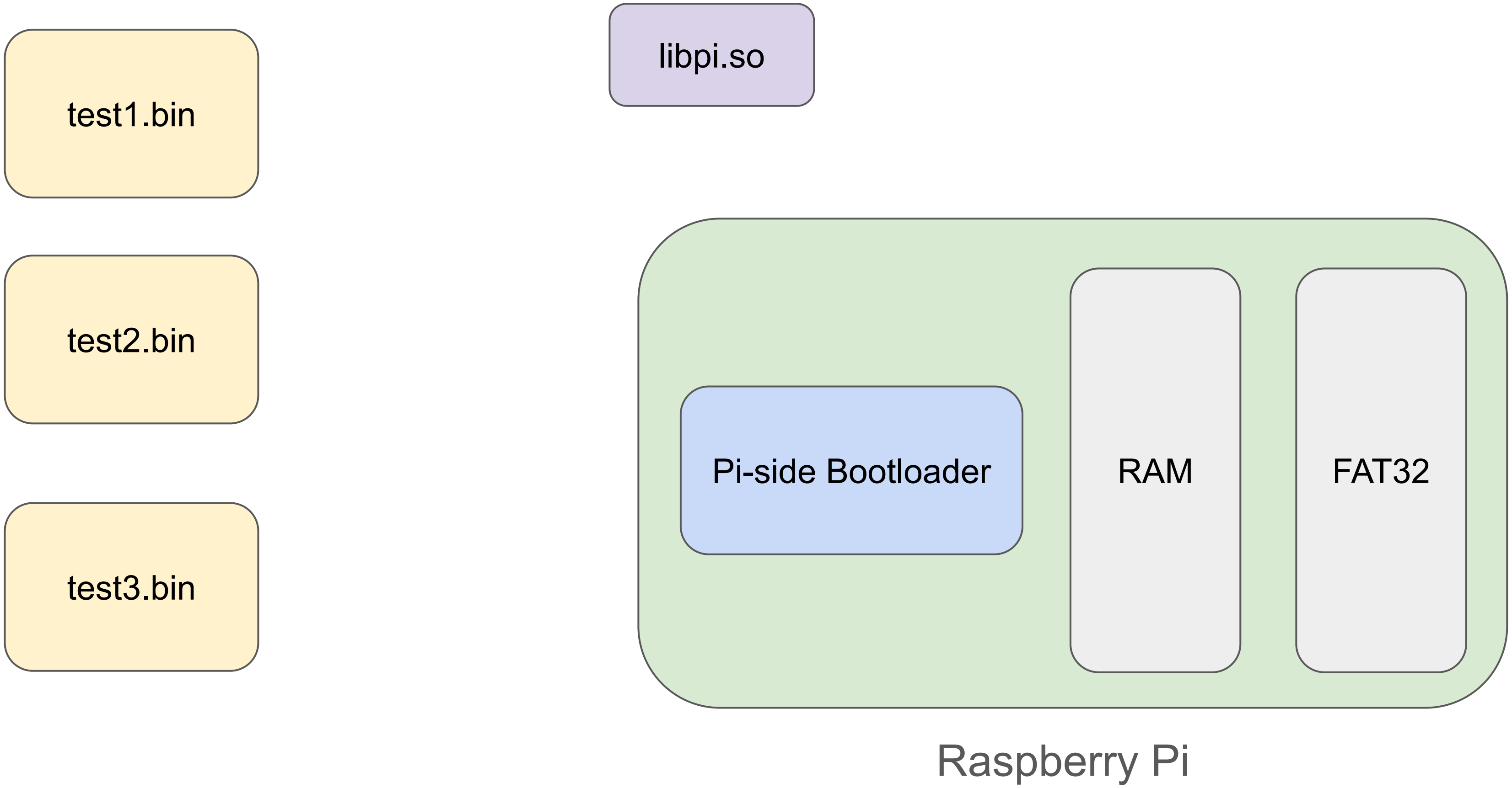

Pi-side Bootloader

libpi.a

RAM

Raspberry Pi


libpi.so
test1.bin
libpi.a
test2.bin
libpi.a
test3.bin
libpi.a
Pi-side Bootloader
libpi.a
RAM
Raspberry Pi



test1.bin

test2.bin

test3.bin

libpi.so

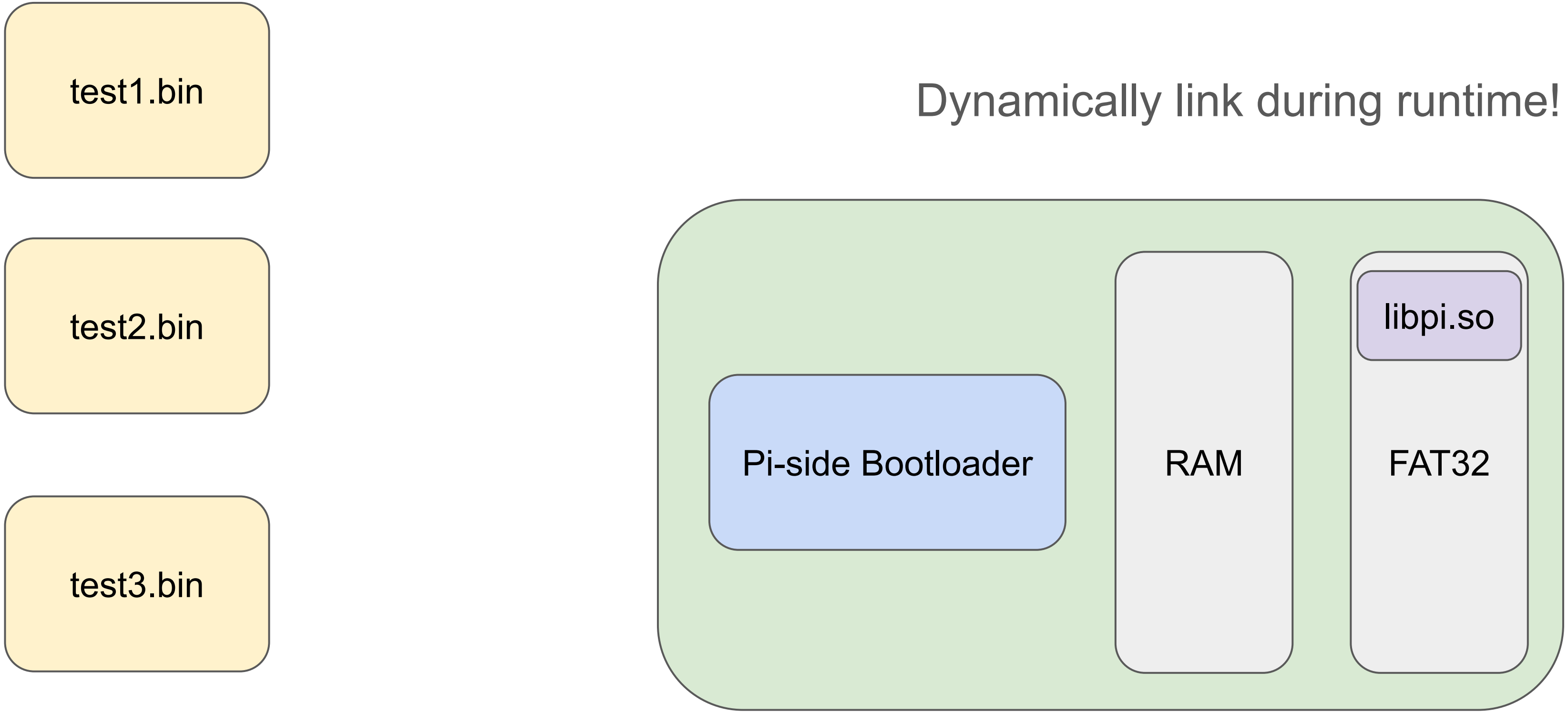
Pi-side Bootloader

RAM

FAT32

Raspberry Pi

Dynamically link during runtime!



test1.bin

test2.bin

test3.bin

Pi-side Bootloader

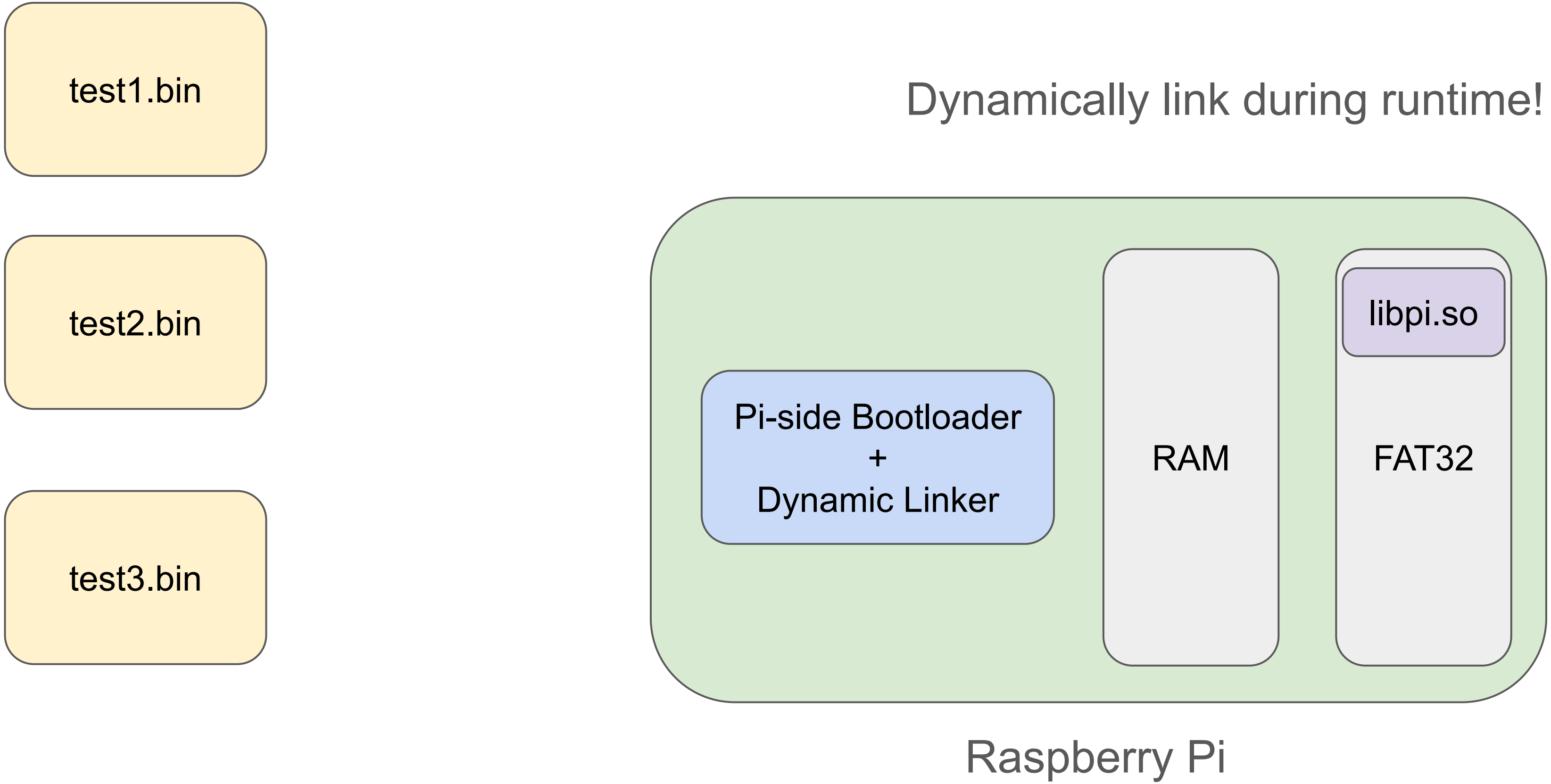
RAM

libpi.so

FAT32

Raspberry Pi

Dynamically link during runtime!



test1.bin

test2.bin

test3.bin

Pi-side Bootloader
+
Dynamic Linker

RAM

libpi.so

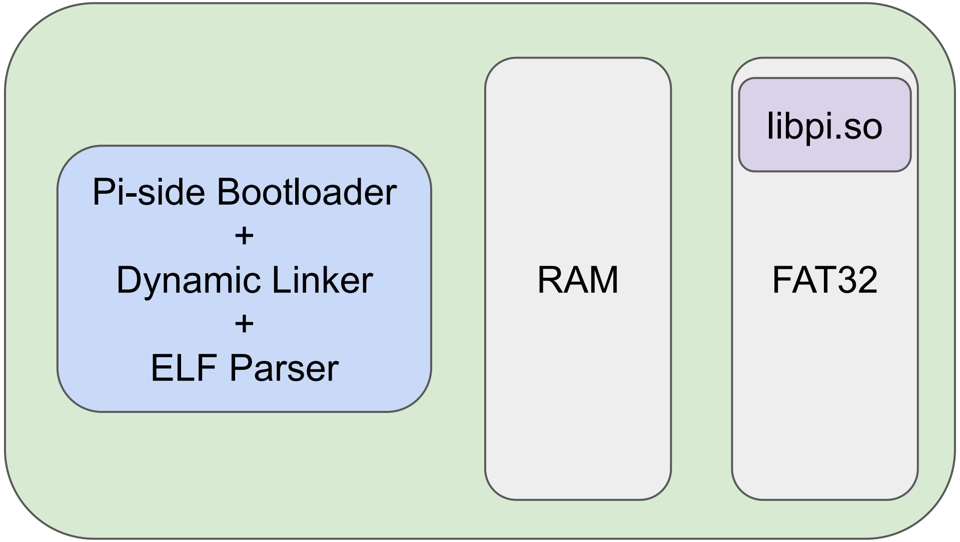
FAT32

Raspberry Pi

Full ELF executables

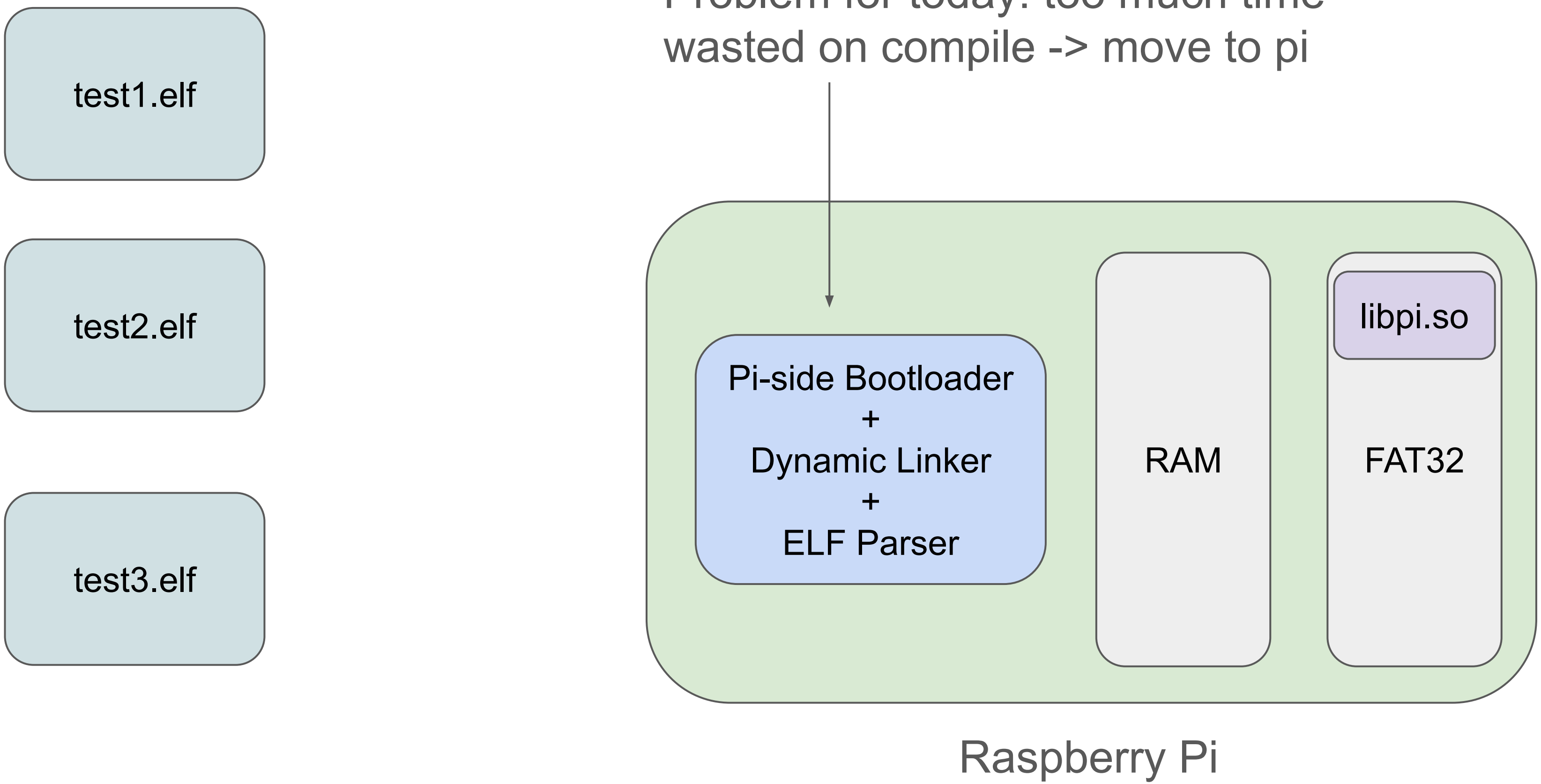
test1.elf
test2.elf
test3.elf

Dynamically link during runtime!
+
Read ELF files, not stripped binary!

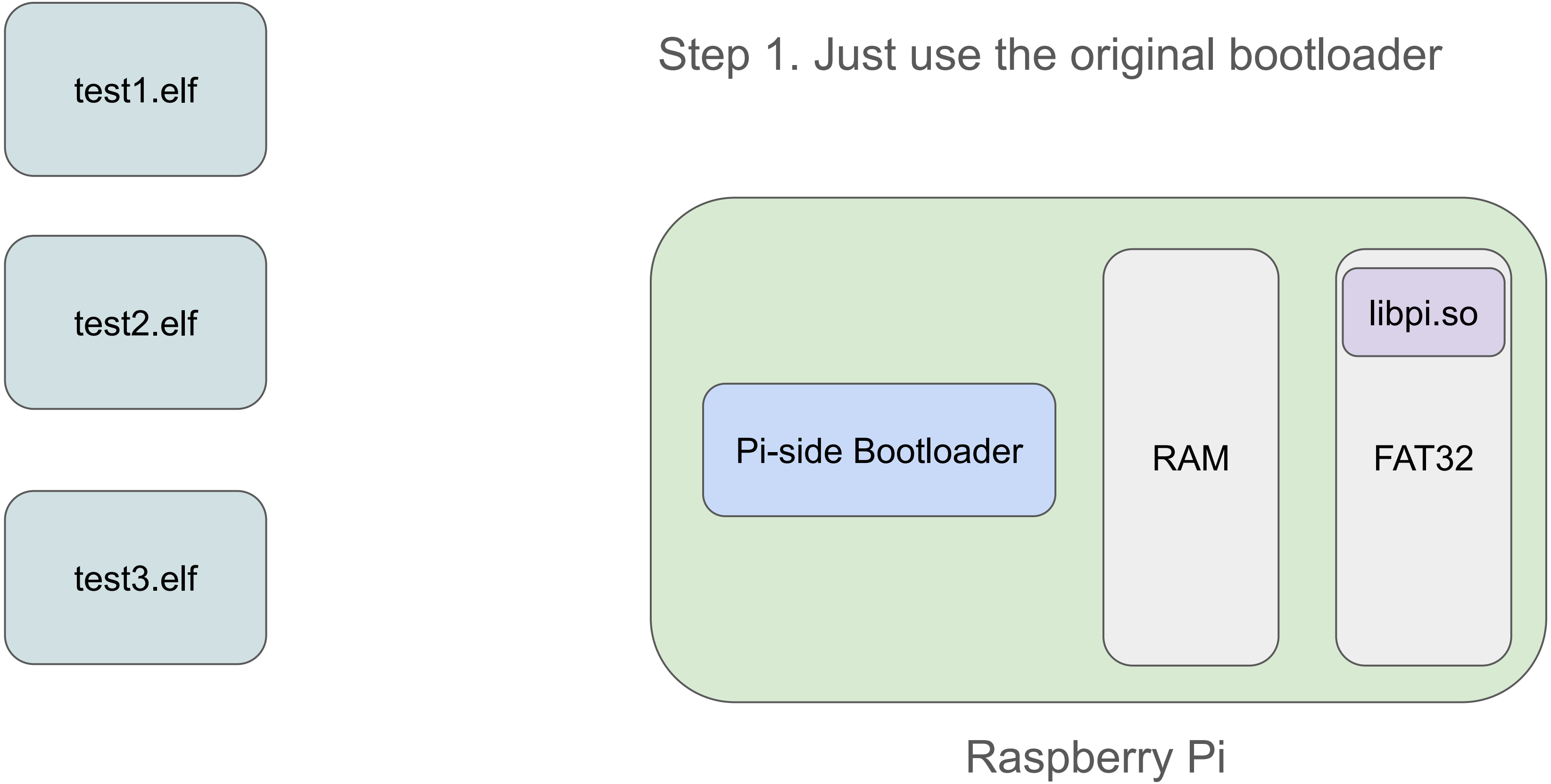


Raspberry Pi

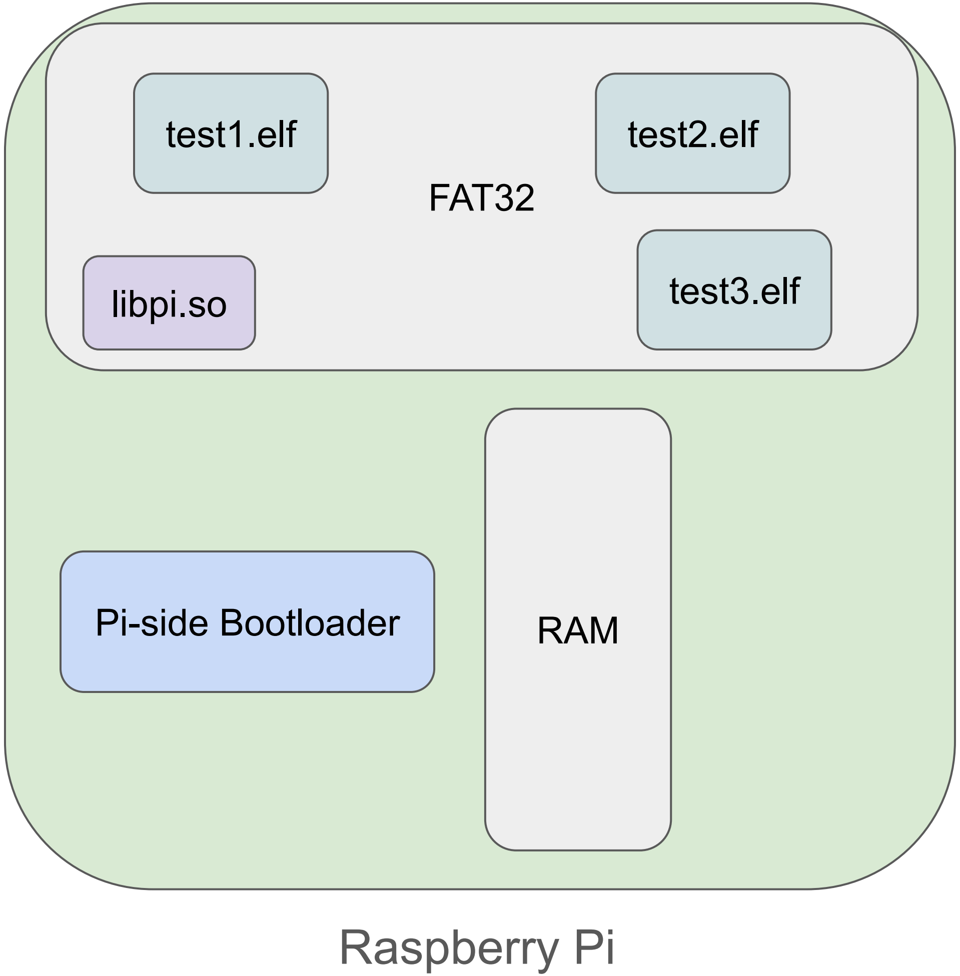
Problem for today: too much time wasted on compile -> move to pi



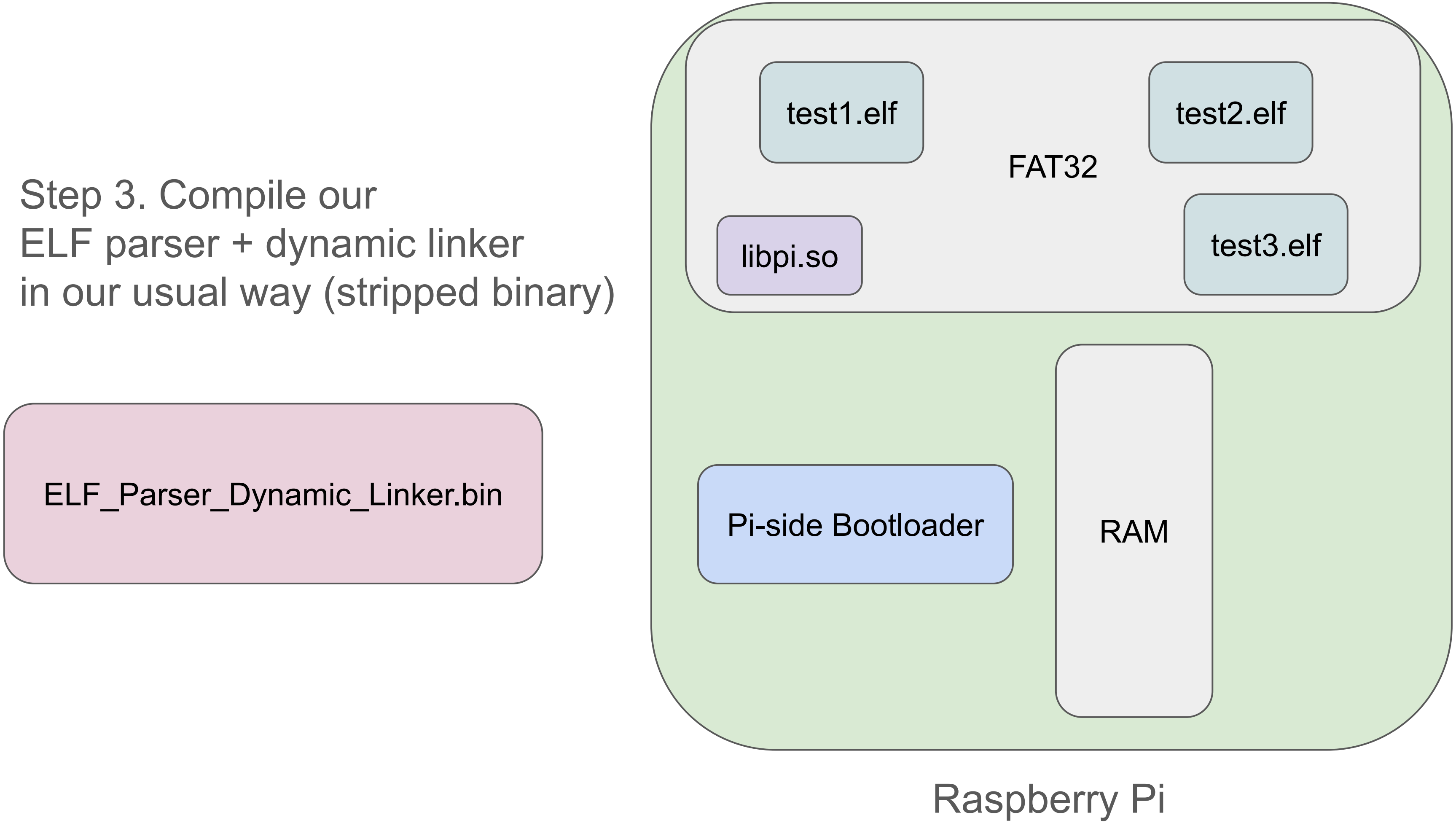
Step 1. Just use the original bootloader



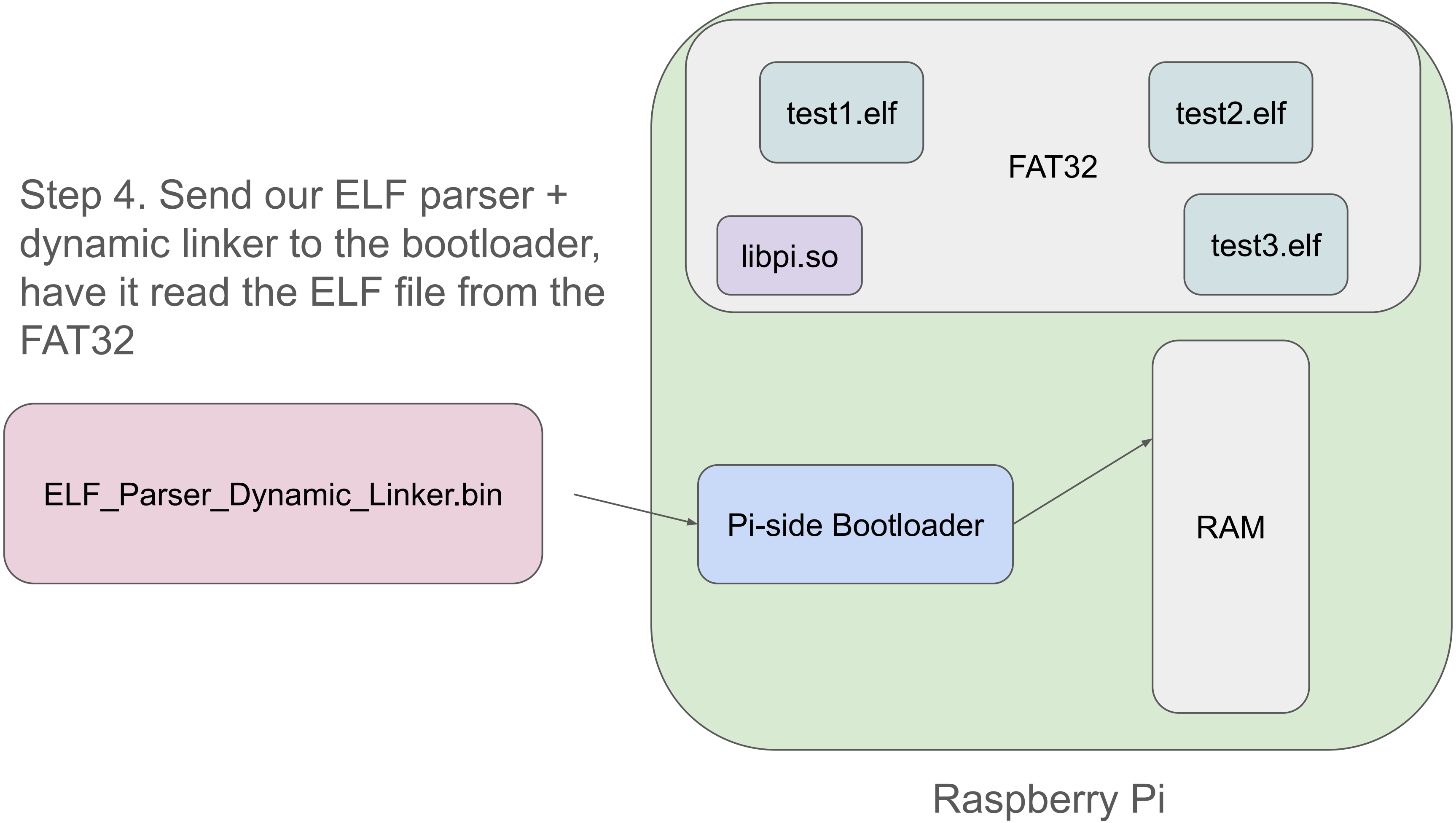
Step 2. Move the ELF executables to FAT32



Step 3. Compile our
ELF parser + dynamic linker
in our usual way (stripped binary)



Step 4. Send our ELF parser + dynamic linker to the bootloader, have it read the ELF file from the FAT32



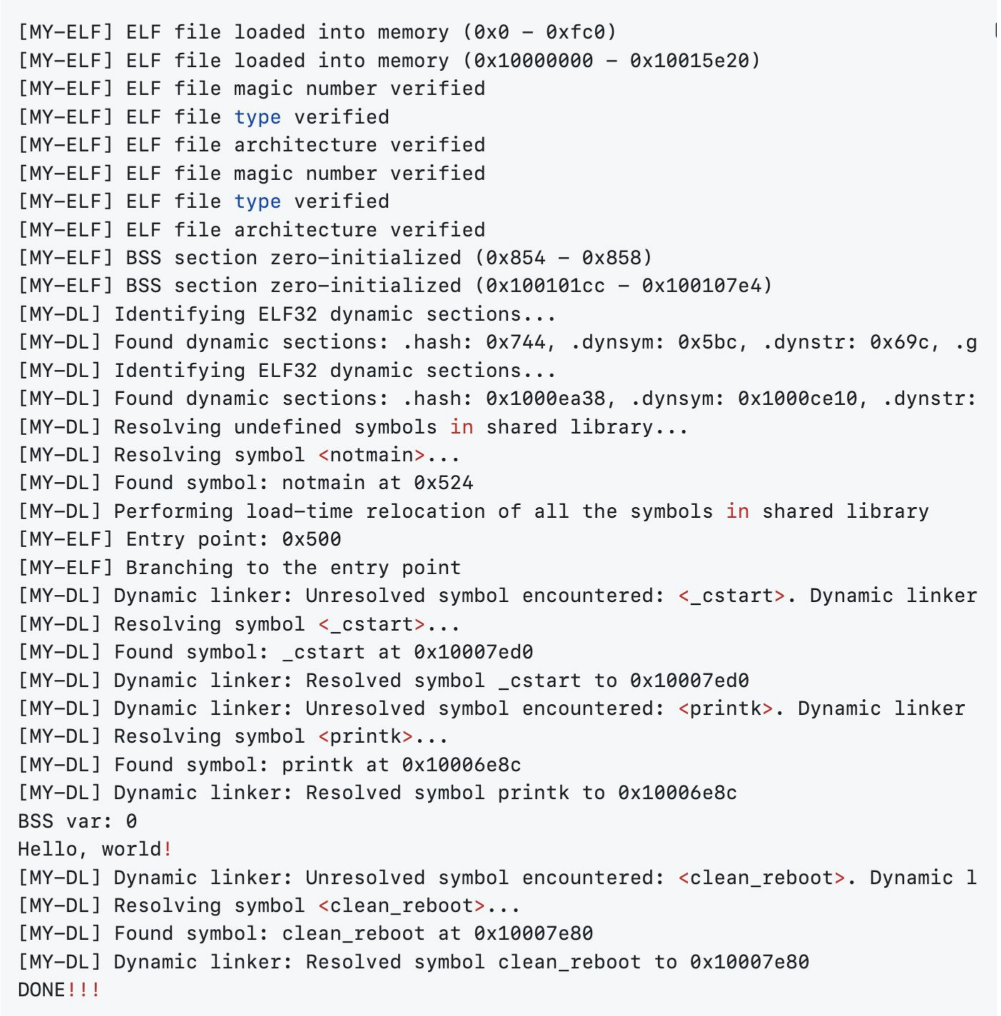
Results

ELF parsing

Shared lib loading

Jumping to entry point

Auto-resolving symbols during runtime (printf, clean_reboot, _cstart, ….)



```
[MY-ELF] ELF file loaded into memory (0x0 - 0xfc0)
[MY-ELF] ELF file loaded into memory (0x10000000 - 0x10015e20)
[MY-ELF] ELF file magic number verified
[MY-ELF] ELF file type verified
[MY-ELF] ELF file architecture verified
[MY-ELF] ELF file magic number verified
[MY-ELF] ELF file type verified
[MY-ELF] ELF file architecture verified
[MY-ELF] BSS section zero-initialized (0x854 - 0x858)
[MY-ELF] BSS section zero-initialized (0x100101cc - 0x100107e4)
[MY-DL] Identifying ELF32 dynamic sections...
[MY-DL] Found dynamic sections: .hash: 0x744, .dynsym: 0x5bc, .dynstr: 0x69c, .g
[MY-DL] Identifying ELF32 dynamic sections...
[MY-DL] Found dynamic sections: .hash: 0x1000ea38, .dynsym: 0x1000ce10, .dynstr:
[MY-DL] Resolving undefined symbols in shared library...
[MY-DL] Resolving symbol <notmain>...
[MY-DL] Found symbol: notmain at 0x524
[MY-DL] Performing load-time relocation of all the symbols in shared library
[MY-ELF] Entry point: 0x500
[MY-ELF] Branching to the entry point
[MY-DL] Dynamic linker: Unresolved symbol encountered: <_cstart>. Dynamic linker
[MY-DL] Resolving symbol <_cstart>...
[MY-DL] Found symbol: _cstart at 0x10007ed0
[MY-DL] Dynamic linker: Resolved symbol _cstart to 0x10007ed0
[MY-DL] Dynamic linker: Unresolved symbol encountered: <printk>. Dynamic linker
[MY-DL] Resolving symbol <printk>...
[MY-DL] Found symbol: printk at 0x10006e8c
[MY-DL] Dynamic linker: Resolved symbol printk to 0x10006e8c
BSS var: 0
Hello, world!
[MY-DL] Dynamic linker: Unresolved symbol encountered: <clean_reboot>. Dynamic l
[MY-DL] Resolving symbol <clean_reboot>...
[MY-DL] Found symbol: clean_reboot at 0x10007e80
[MY-DL] Dynamic linker: Resolved symbol clean_reboot to 0x10007e80
DONE!!!
```

Just 1 more thing

Symbol table (.symtab, .dynsym)

st_name	st_value
0	0x83ce
7	0x8
20	0x83ce

String table (.strtab, .dynstr)

printf\0clean_reboot\0
my_func\0…

Symbol table (.symtab, .dynsym)

st_name	st_value
0	0x83ce
7	0x8
20	0x83ce

Index in the string table + actual
symbol address

String table (.strtab, .dynstr)

printf\0clean_reboot\0
my_func\0…

List of \0-terminated characters

Enjoy!