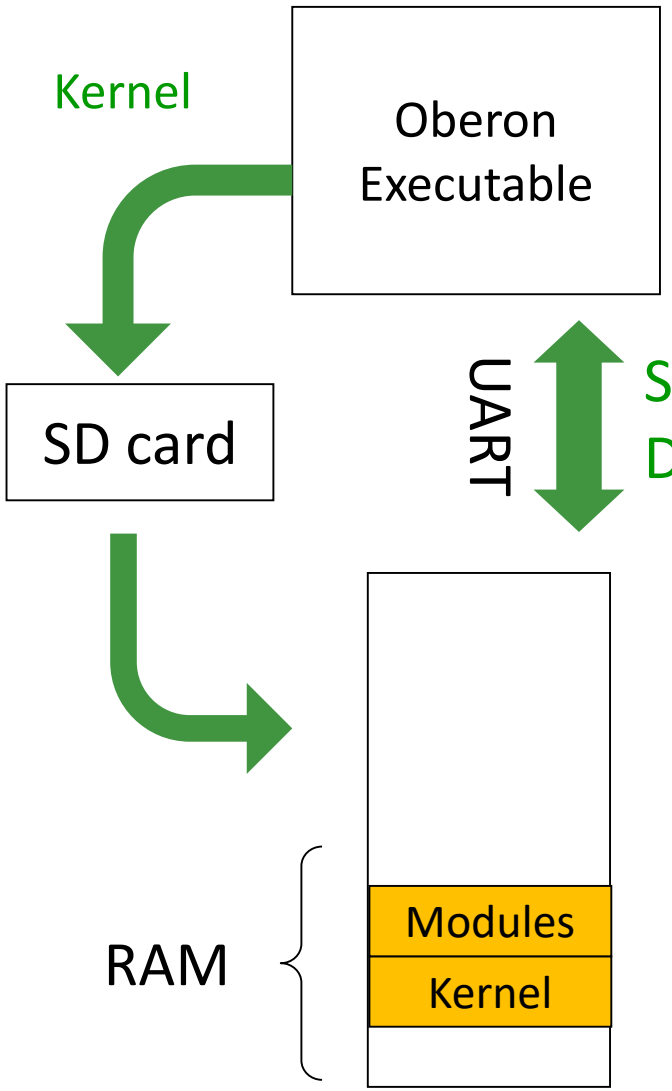
How to Cross-Develop, Build and Deploy a System

1.2. CROSS DEVELOPMENT, PROGRAMMING LANGUAGE AND RUNTIME SUPPORT

Cross Development Platform

used in the Exercises



- [Program Editor]
- Cross Compiler
- Linker
- Dialog

Developer Software

- Boot from SD Card (VC / ARM on RPI)
- Kernel
- Command Interpreter
- Log

Target (Minos OS)

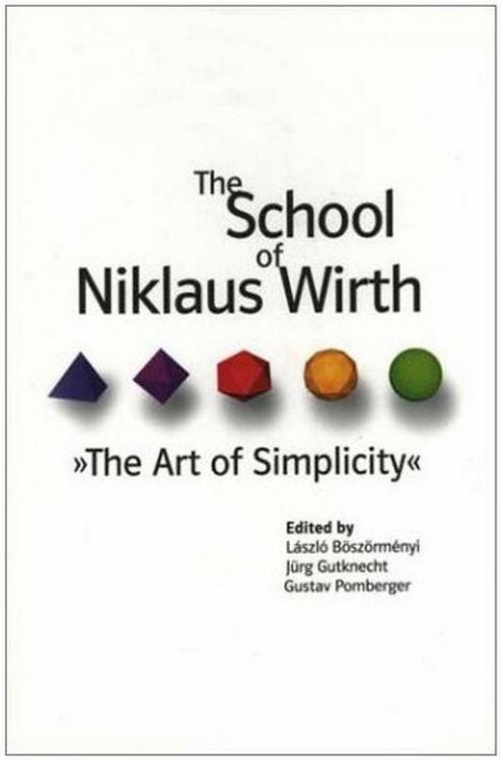
Programming Language Oberon

- Pascal family
- Modular with separate compilation
- Strongly typed
  - Static type checking at compile time
  - Runtime (dynamic) support for type guards / tests
- Consequently high level
  - Minimal assembler code (we used some in the first exercises)
  - Specific low level functions in a Pseudo-Module called SYSTEM

The art of simplicity

- Most recent Compilers by Prof. N. Wirth
  - restricted subset of Oberon (Oberon07), designed for one-pass compiler

part	size in lines of code
scanner:	300
parser/driver:	1000
types/symbols:	500
generator	1400
	--------
	ca 3k



- Fox Compiler, used in the exercises (including all backends, supporting various add-ons) ca. 50k lines of code
- gcc / llvm : Millions of lines of code

Where are the programs?

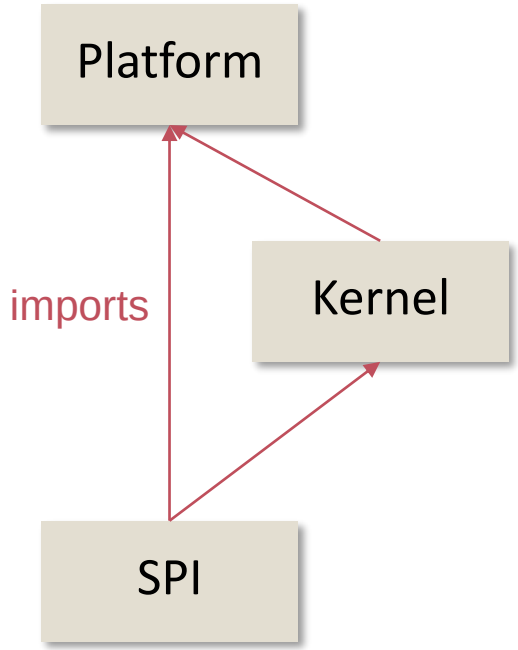
- There is no «program» in Oberon.
- There are modules.
  Modules can contain commands.
  Commands can be called.
- Modules can be statically linked to
  form a kernel (or executable if embedded in other OS)
- Modules can be *dynamically* linked (=loaded)

Example of a Module

```
MODULE SPI; (* Raspberry Pi 2 SPI Interface – Bitbanging *)
IMPORT Platform, Kernel;

CONST HalfClock = 100; (* microseconds -- very conservative*)

PROCEDURE SetGPIOs;
BEGIN
   Platform.ClearAndSetBits(Platform.GPFSEL0, {21..29},{21,24});
   Platform.ClearAndSetBits(Platform.GPFSEL1, {0..5},{0,3});
END SetGPIOs;

PROCEDURE Write* (CONST a: ARRAY OF CHAR);
VAR i: SIZE;
BEGIN
   Kernel.MicroWait(HalfClock);
   Platform.WriteBits(Platform.GPCLR0, SELECT); (* signal select *)
   Kernel.MicroWait(HalfClock);
   FOR i := 0 TO LEN(a)-1 DO
      WriteByte(a[i]);   (* write data, toggling the clock *)
   END;
   Kernel.MicroWait(HalfClock);
   Platform.WriteBits(Platform.GPSET0, SELECT); (* signal deselect *)
END Write;
...

BEGIN
   SetGPIOs;
END SPI;
```

*** = exported procedure**: can be used by importing modules

module body: executed first -- and only once -- when module is loaded



Example of a Module

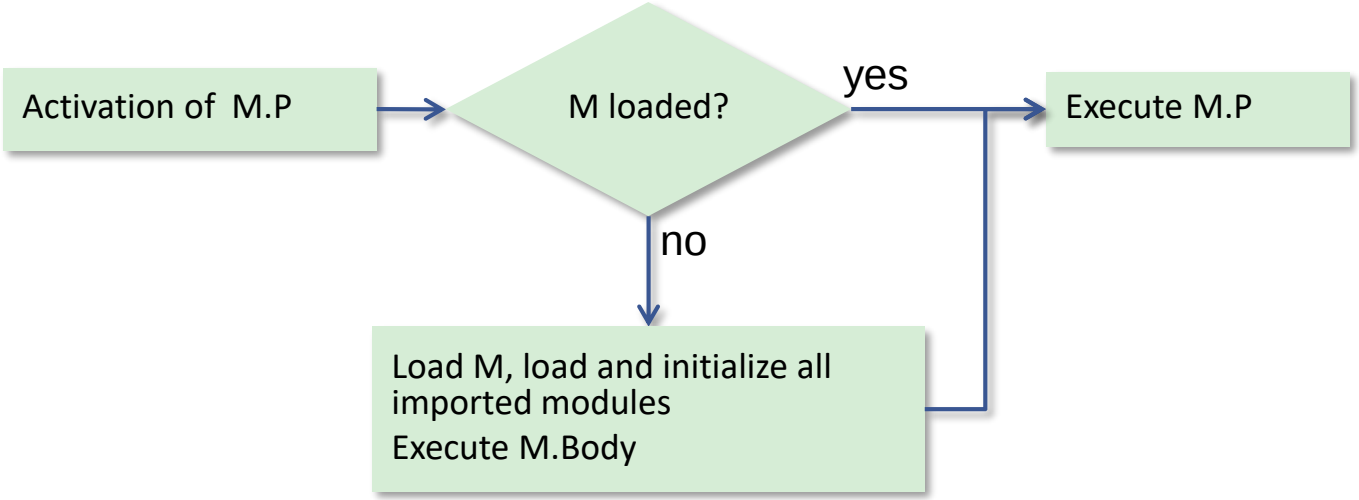
```
MODULE Timer;

IMPORT Kernel, Out := Log;

VAR global: INTEGER; factor: REAL;

    PROCEDURE Start*(VAR ticks: INTEGER);
    BEGIN time := Kernel.GetTicks();
    END Start;

    PROCEDURE Step*(VAR ticks: INTEGER): REAL;
    VAR previous: INTEGER;
    BEGIN previous := ticks; ticks :=  Kernel.GetTicks(); RETURN (ticks-previous)*factor
    END Step;

    PROCEDURE Tick*; BEGIN Start(global); END Tick;

    PROCEDURE Tock*;
    BEGIN Out.String("elapsed seconds: "); Out.Real(Step(global),20); Out.Ln;
    END Tock;

    PROCEDURE Calibrate; BEGIN … END Calibrate;

BEGIN Calibrate();
END Timer.
```

global symbols (variables)
in module context

exported procedure without parameters:
can be used as command

Commands and Module Loading

- Modules are *loaded on demand*
- *Statically linked* modules are loaded at system-startup
- Exported parameter-less Procedures can act as *commands*
- A modification of a compiled module becomes effective only after *re-loading* the module
- A module M can be *unloaded* only if no currently loaded module imports M and if M is not statically linked to the Kernel



Oberon Language

Program units

MODULE, PROCEDURE （Value, VAR and CONST parameters）

Data types

INTEGER, SIZE, ADDRESS, SIGNED8|16|32|64, UNSIGNED8|16|32|64,
REAL, FLOAT32, FLOAT64,
SET, SET8|16|32|64,
BOOLEAN, CHAR,

Structured types

ARRAY, RECORD, OBJECT, POINTER TO ARRAY, POINTER TO RECORD

Statements

ProcedureCall, Assignment, IF, WHILE, REPEAT, LOOP/EXIT, FOR, CASE, WITH,
AWAIT, RETURN, BEGIN ... END, CODE, IGNORE

Control Structures

IF

```
IF a = 0 THEN
        (* statement sequence *)
END
```

WHILE

```
WHILE x<n DO
        (* statement sequence *)
END
```

REPEAT

```
REPEAT
        (* statement sequence *)
UNTIL x=n;
```

FOR

```
FOR i := 0 TO 100 DO
        (* statement seq *)
END;
```

CASE

```
CASE c OF
'a'..'z':  …
| '0'..'9': …
ELSE
END;
```

WITH

```
WITH obj:BinaryExpression DO
        …
| UnaryExpression DO
        …
ELSE
END;
```

Fundamental Types

BOOLEAN

```
b := TRUE; IF b THEN END;
```

CHAR

```
c := ‘a’; c := 0AX;
```

SIGNED8 ⊂ UNSIGNED8 ⊂ SIGNED16 ⊂ UNSIGNED16 ⊂ SIGNED32 ⊂ UNSIGNED32 ⊂ SIGNED64 ⊂ UNSIGNED64

```
i := SIGNED8(s); l := 10; h := 01CH; h := 0x1a;
```

FLOAT32 ⊂ FLOAT64

```
r := 1.0; r := 10E0;
```

SET8 ⊂ SET16 ⊂ SET32 ⊂ SET64

```
s := {1,2,3}; s := s + {5};
s := s – {5}; s := s * {1..6};
INCL(s, 5); EXCL(s, 8);
s := SHL(s,2); s := ROL(s,2); s := SHR(s,3);
```

ADDRESS, SIZE, INTEGER, REAL

Builtin Functions

Increment and decrement

```
INC(x); DEC(x);
INC(x,n); DEC(x,n);
```

Sets

```
INCL(set, element);
EXCL(set, element);
```

Assert and Halt

```
ASSERT(b<0); HALT(100);
```

Allocation

```
NEW(x, …); x := NEW(T,...);
```

Shifts

```
ASH(x,y); LSH(x,y); ROT(x,y);
SHL(x,y); SHR(x,y); ROL(x,y); ROR(x,y);
```

Conversion

```
SETx(i); SIGNEDx(i); ORD(ch); CHR(i);
ENTIER(r);
```

Arrays

```
LEN(x); LEN(x,y); DIM(t);
```

Misc

```
ABS(x); MAX(type); MIN(type); ODD(i);
CAP(c);
```

Addresses and Sizes

```
ADDRESS OF x; ADDRESSOF(x);
SIZE OF t; SIZEOF(t);
```

Pseudo Module SYSTEM

Direct Memory Access Functions

```
SYSTEM.PUT (a, x), SYSTEM.GET (a, x),

SYSTEM.PUT8|16|32|64(a, x); x := SYSTEM.GET8|16|32|64(a);

SYSTEM.MOVE(src, dest, length);
```

Data Type

```
SYSTEM.BYTE
```

Type Cast (≠Conversion)

```
b := SYSTEM.VAL(a, t);
```

Register Access

```
sp := SYSTEM.GetStackPointer(); SYSTEM.SetStackPointer(sp);
sp := SYSTEM.GetFramePointer(); SYSTEM.SetFramePointer(sp);
```

ARM specific

```
SYSTEM.LNK(), SYSTEM.SETLR(x)
SYSTEM.LDPSR(b,x), SYSTEM.STPSR(b,x)
```

Example: Low-level access without Assembly

```
IMPORT SYSTEM;


PROCEDURE LetThereBeLight;
CONST GPSET0 = 03F20001CH;
BEGIN
    SYSTEM.PUT(GPSET0, {21});
END LetThereBeLight;
```

SYSTEM.PUT: write to address

Interrupt Procedures

```
PROCEDURE Handler {INTERRUPT, PCOFFSET=k};
BEGIN (* k is the offset to the next instruction
        cf. table of exceptions *)
END Handler;
```

special calling convention

Special System's Programming Flags and Features

PROCEDURE {PLAIN}

Procedure without procedure activation frame

PROCEDURE {OPENING}

Procedure that is linked to the beginning of a kernel

PROCEDURE {CLOSING}

Procedure that is linked after call to all module bodies

CODE … END

special statement block that can contain inline assembler code

Special System's Programming Flags and Features

POINTER {UNSAFE} TO ...

Unsafe pointer that is assignment compatible with type ADDRESS.
C-like behavior. Not GCed. Not safe!

symbol {ALIGNED(32)}

alignment of a symbol (e.g. variable)

symbol {OFFSET(0x8000)}

providing an offset (e.g. in a record)

symbol { UNTRACED }

symbol that is invisible to a Garbage Collector

a EXTERN adr : INTEGER;

pin variable to address or symbol

PROCEDURE EXTERN name P(INTEGER x);

pin procedure to extern name (linker hint)

System Programming with Oberon

Bits

Use built-in type `SET` for bitsets …

```
VAR s: SET;

INCL(s, 3);   -- include bit 3 in s

EXCL(s, 4);   -- exclude bit 4 from s

s := {0,2,5}; -- s consisting of bits 0, 2 and 5 (int value 37)

s := s + {1,3,5}; -- include bits 1,3,5 in s

s := s - {1,2,3}; -- exclude bits 1,2,3 from s

s := SHL(s,3); -- shift bits
```

```
PROCEDURE EnableIRQs*;
VAR cpsr: SET;
BEGIN SYSTEM.STPSR( 0, cpsr );
      cpsr := cpsr - {7};
      SYSTEM.LDPSR( 0, cpsr );
END EnableIRQs;
```

System Programming with Oberon

```
PROCEDURE EnableIRQs*;
VAR cpsr: SET;
BEGIN SYSTEM.STPSR( 0, cpsr );
      cpsr := cpsr - {7};
      SYSTEM.LDPSR( 0, cpsr );
END EnableIRQs;
```

Bits

and / or arithmetic operations and ODD

```
VAR i: LONGINT;
...
i := i DIV 10H;                 shift to right by 4
i := i MOD 10H;                 and with 0FH
IF ODD(i) THEN                  test if bit 0 is set
i DIV 10000H MOD 100H;          extract bits 20..27 from i
SHL(i,3);                       shift bits (left)
SHR(i,1);                       shift bits (right)
```

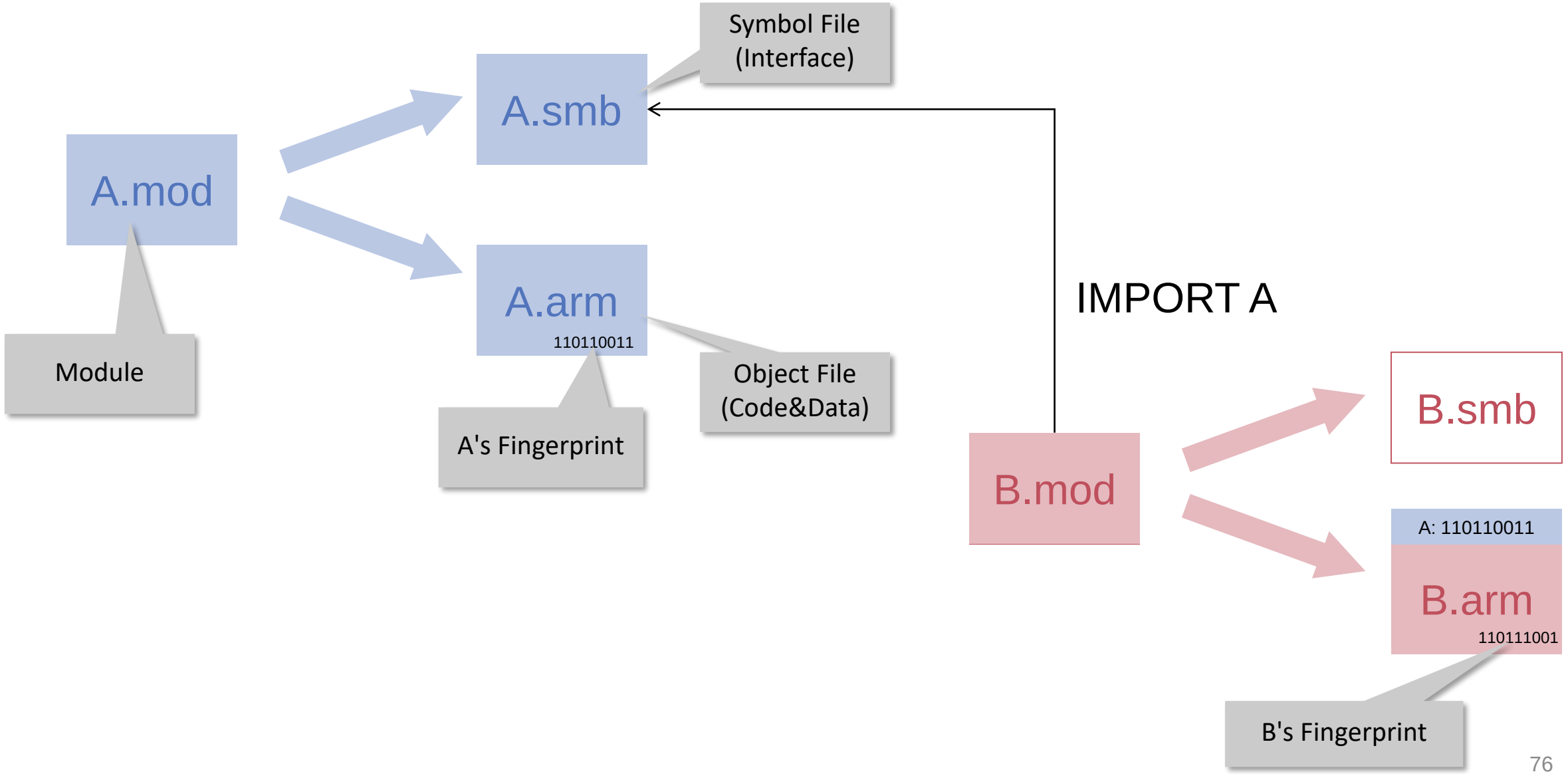
Example: Inline-Assembly within Modules

```
MODULE MinimalLED;
IMPORT SYSTEM; (* required for writing assembler code  *)

PROCEDURE {OPENING} Entry;
CODE
      ldr r0, [gpio]              ; load base address of gpio
      mov r1, #8             ; write 3rd bit ...
      str r1, [r0,#8]        ; ... to register at 3f200000 + 8
      mov r1, #0x200000           ; write 21st bit ...
      str r1, [r0, #0x1c]         ; ... to register at 3f200000 + 0x1c
      b end                  ; jump over data
      gpio: d32 0x3f200000   ; data
      end:
END Entry;

PROCEDURE {CLOSING} Exit;
CODE
      end:
      b end
END Exit;
END MinimalLED.
```

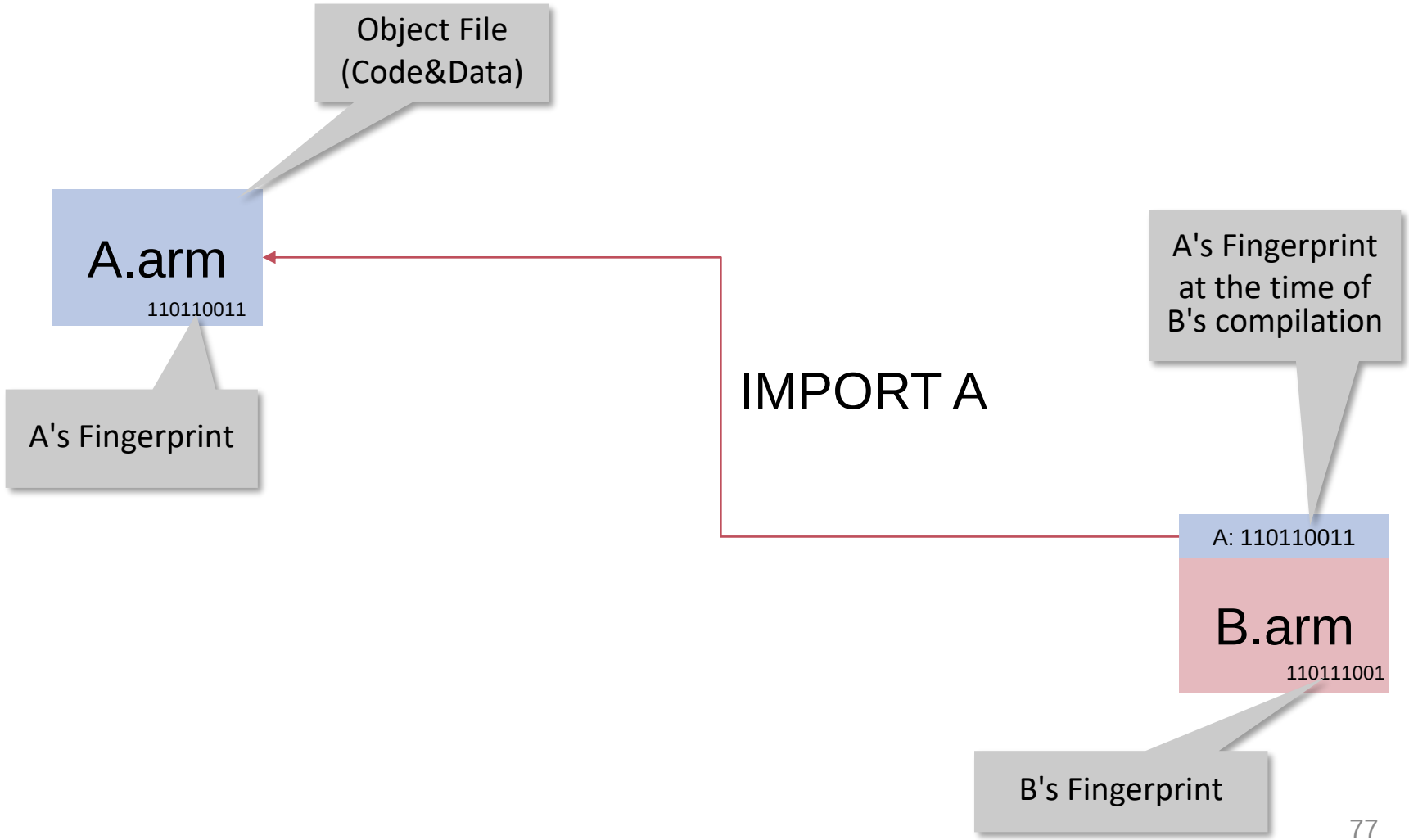
Example: Unsafe Pointers

```
MODULE TestLED;
IMPORT SYSTEM;

CONST GPIO = 03F200000H;
VAR
   gpio: POINTER {UNSAFE} TO RECORD
      GPFSEL: ARRAY 6 OF SET;
      res0: ADDRESS;
      GPFSET: ARRAY 2 OF SET;
      ...
   END;

   PROCEDURE SwitchOnLED;
   BEGIN
      gpio.GPFSEL[2] := {3};
      gpio.GPFSET[0] := {21};
   END SwichOnLED;

BEGIN
   gpio := GPIO;
   SwitchOnLED;
END TestLED.
```

Compilation Schema



Linking Schema



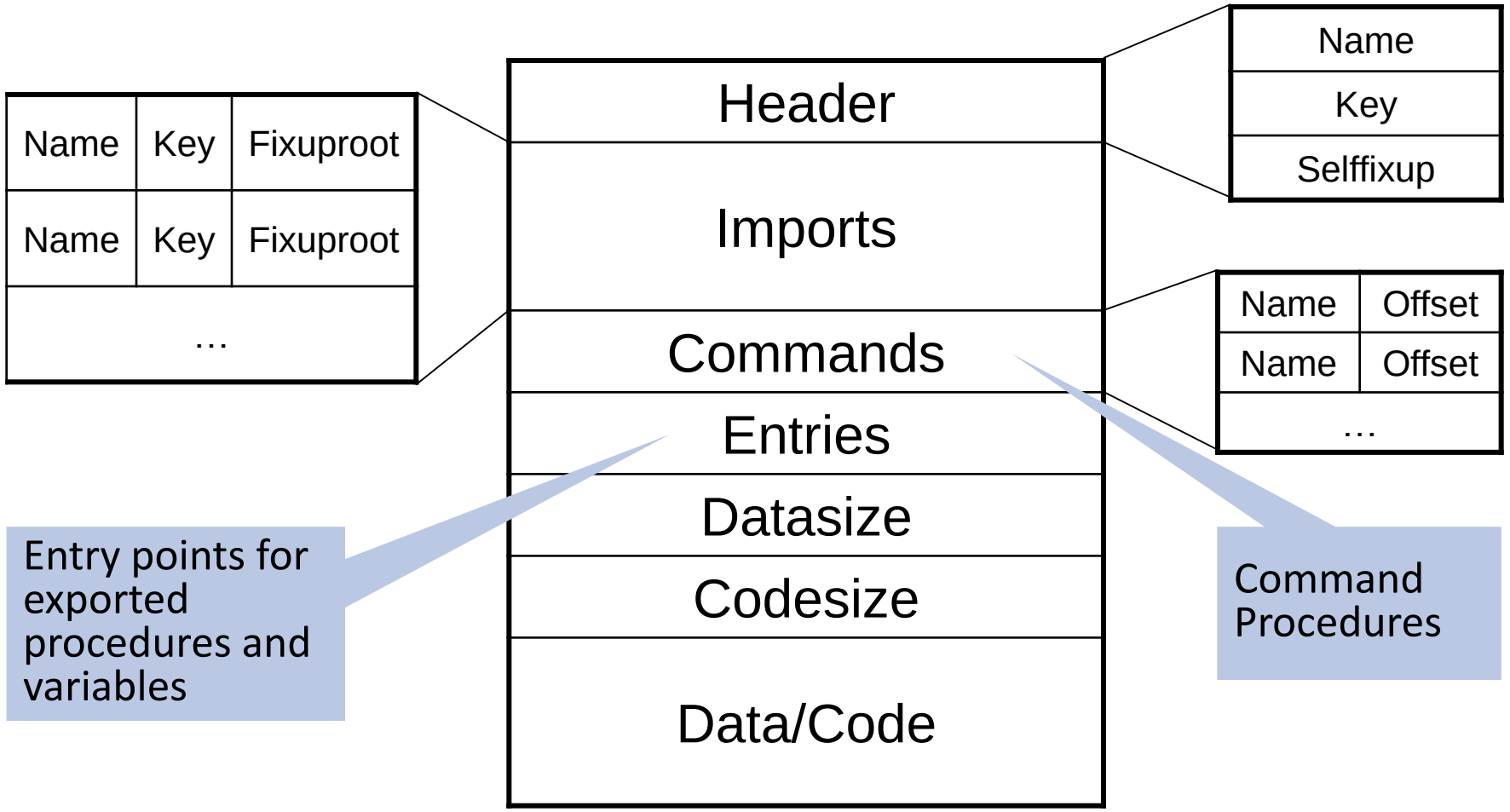
Linking Process

```
MODULE A;
   IMPORT B, C, ...;
BEGIN S (* initialize *)
END A.
```

- Link A =
  Link B; Link C; ...
  Fixup external call chains in A;
  Execute S

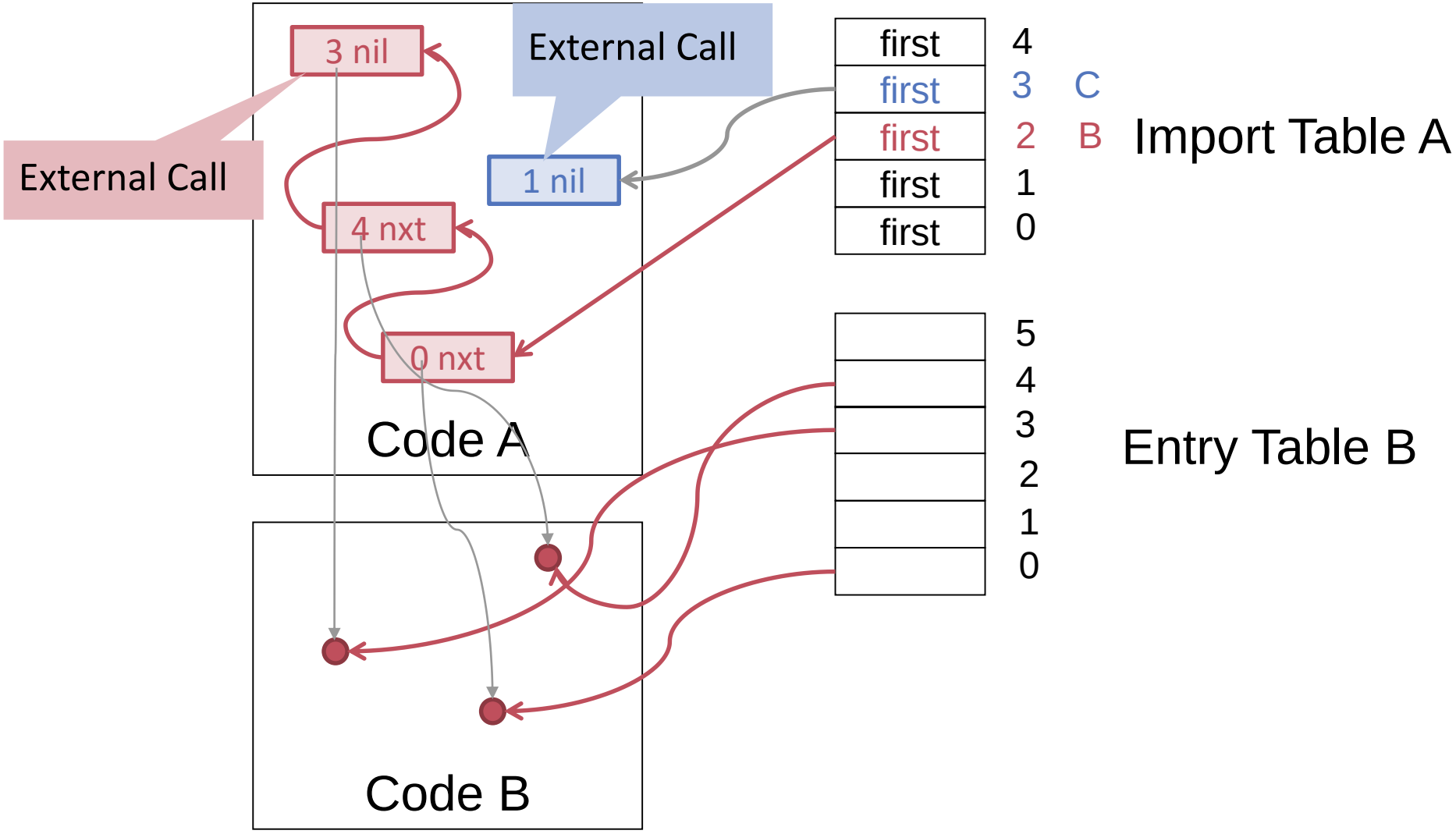
```
....
00008010:   B #134504
....

00028D80:   BL #-134508
00028D84:   BL #-133008
00028D88:   BL #-124984
00028D8C:   BL #-117280
00028D90:   BL #-113584
00028D94:   BL #-106772
00028D98:   BL #-98592
00028D9C:   BL #-98452
00028DA0:   BL #-90572
00028DA4:   BL #-85468
00028DA8:   BL #-38196
00028DAC:   BL #-35944
00028DB0:   BL #-32456
00028DB4:   BL #-28068
00028DB8:   BL #-25104
00028DBC:   BL #-22948
00028DC0:   BL #-17648
00028DC4:   B #-8
```

Binary Object File Format



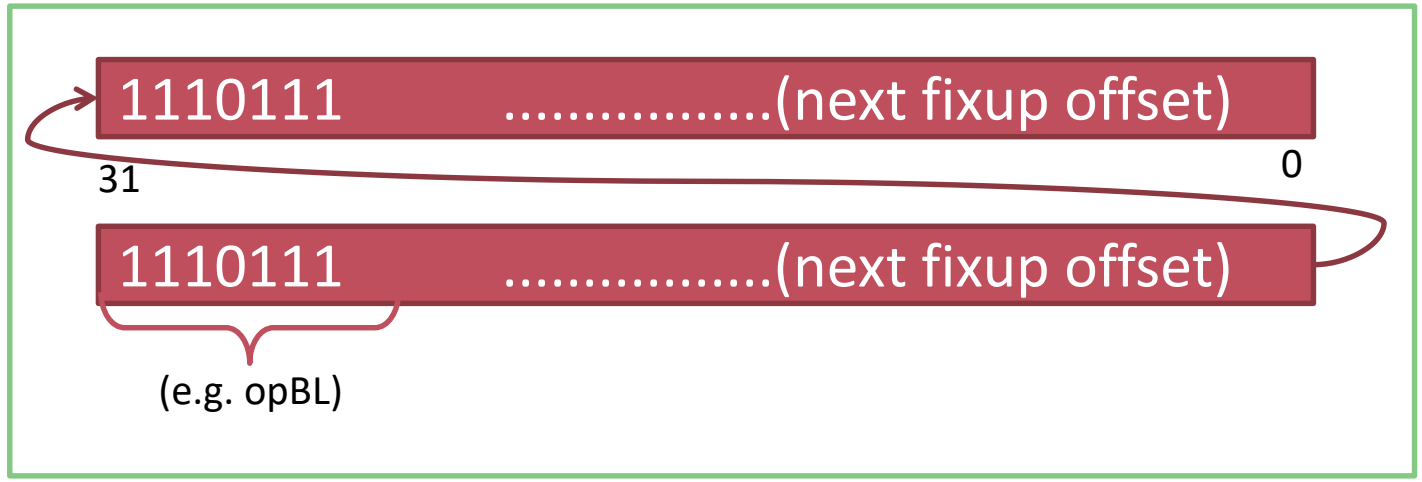
Compiler.Compile –p=Minos

Fixups


3 nil
External Call
External Call
1 nil
4 nxt
0 nxt
Code A
Code B
first
first
first
first
first
4
3
2
1
0
C
B
Import Table A
5
4
3
2
1
0
Entry Table B

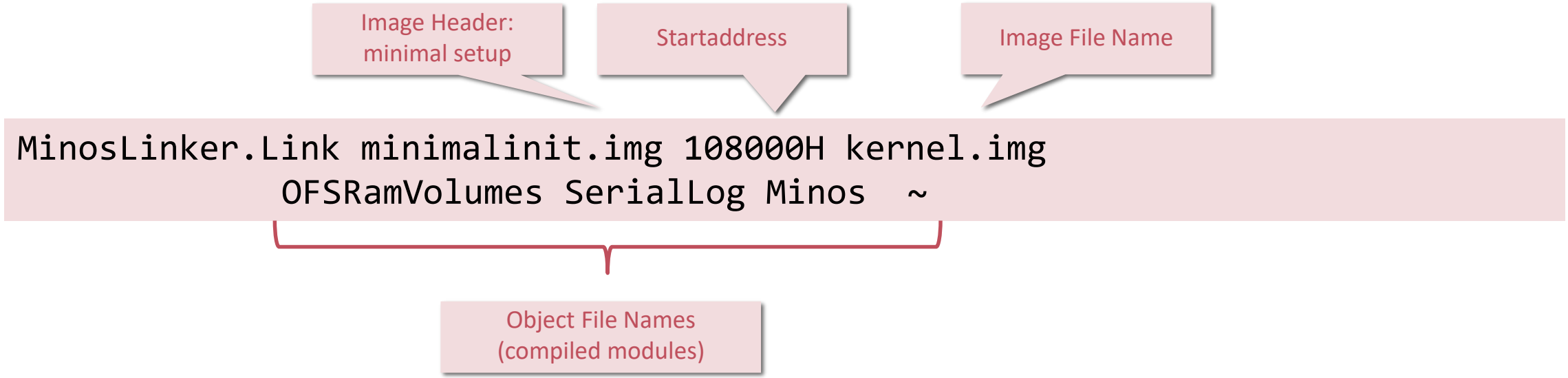
Fixup Chains (Module Modules)

```
WHILE (fixloc # 0) DO
       SYSTEM.GET(pbase + fixloc*4, instr);
       next := instr MOD 10000H;
       pno := instr DIV 10000H MOD 100H;
       op := instr DIV 100000H MOD 100H;
       ... (* fixup *)
       fixloc := next;
END;
```



Bootfile

- Linked module hierarchy of OS kernel
- Predefined loading address and entry point (0x8000 for RPI2)
- Bootlinking command in host system



Type Declarations

```
TYPE

Device *= POINTER TO DeviceDesc;
DeviceDesc* = RECORD
   id*: INTEGER;
   Open*: PROCEDURE (dev: Device);
   Close*: PROCEDURE(dev: Device);
   next*: Device;
END;
```

Pointer (to Record)
Reference Type

Record
Value Type

Record Fields

Type Declarations

```
TYPE

TrapHandler* = PROCEDURE(type,adr,fp: INTEGER;VAR res: INTEGER );

NumberType*= REAL;

DeviceName* = ARRAY DeviceNameLength OF CHAR;

Data*= POINTER TO ARRAY OF CHAR;
```

Procedure Type with Signature

Type Alias

Array Type

Dynamic Array Type

Inheritance (Example)

```
Task* = POINTER TO TaskDesc;
TaskDesc* = RECORD
   proc: PROCEDURE (me: Task);   (* This procedure is executed in the task *)
   next: Task;                   (* The next task in the list of tasks *)
END;

PeriodicTask* = POINTER TO PeriodicTaskDesc;
PeriodicTaskDesc* = RECORD (TaskDesc)
   priority: LONGINT;            (* The priority determines the execution order *)
   interval: LONGINT;            (* The task is executed every "interval" msecs *)
END;
```


```
IF task IS PeriodicTask THEN ... END;
IF task(PeriodicTask).priority = 1 THEN ... END;
WITH task: PeriodicTask DO
   ...
ELSE ... END;
```

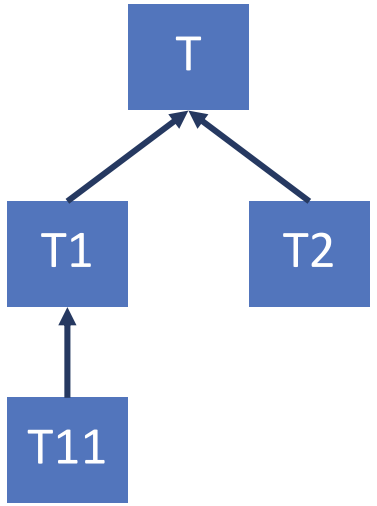
type test

type guard

type test + guard

Runtime Support: Inheritance Scenario

```
TYPE
  T = POINTER TO RECORD (* base type *)
    ... (* base fields *)
  END;
  T1 = POINTER TO RECORD (T) (* extended type *)
    ... (* additional fields *)
  END;
  T2 = POINTER TO RECORD (T)
    ...
  END;
  T11 = POINTER TO RECORD (T1)
    ...
  END;
```

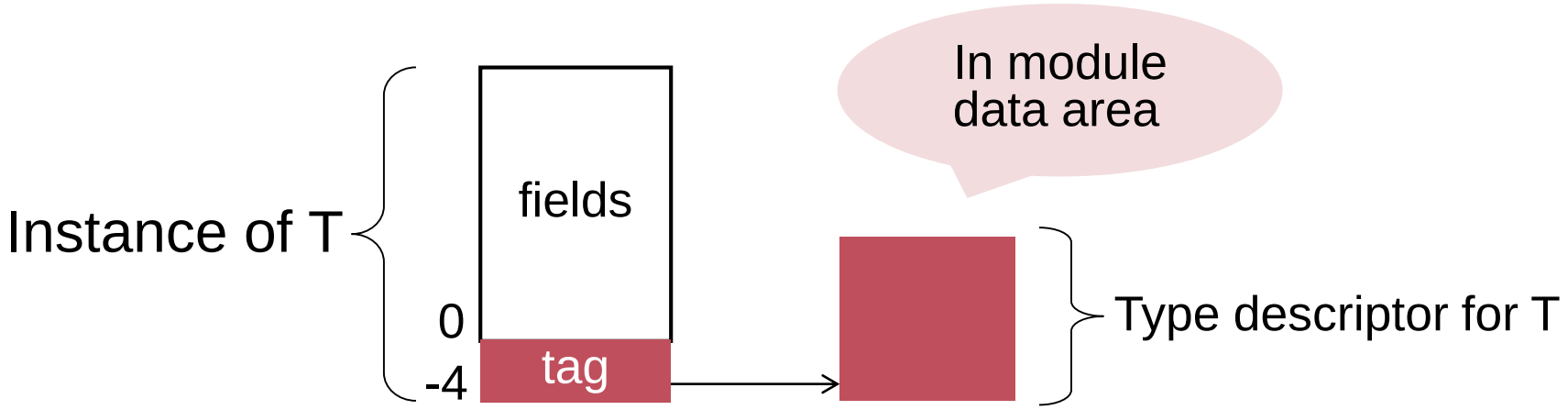


Runtime Support: Type Descriptors

Basic type descriptor

```
TDesc* = ARRAY 3 OF LONGINT;

(* ext[i] = pointer to TDesc
     of base type at level i + 1 *)
```

Type tag



Runtime Support: Type Test Code

Source code

```
VAR t: T; t11: T11; (* static types *)

BEGIN
   NEW(t11); t := t11;
   IF t = NIL THEN ... END; (* false *)
   IF t IS T11 THEN ... END; (* true *)
   IF t IS T1 THEN ... END; (* true *)
   IF t IS T2 THEN ... END; (* false *)
```

Compiled code (pseudocode)

```
   CMP t, 0
   CMP t.tag.ext[2], adr(typedesc T11)
   CMP t.tag.ext[1], adr(typedesc T1)
   CMP t.tag.ext[1], adr(typedesc T2)
```

also possible:

```
WITH t: T11 DO ...
| T2 DO ...
| T1 DO ...
ELSE ...
END;
```