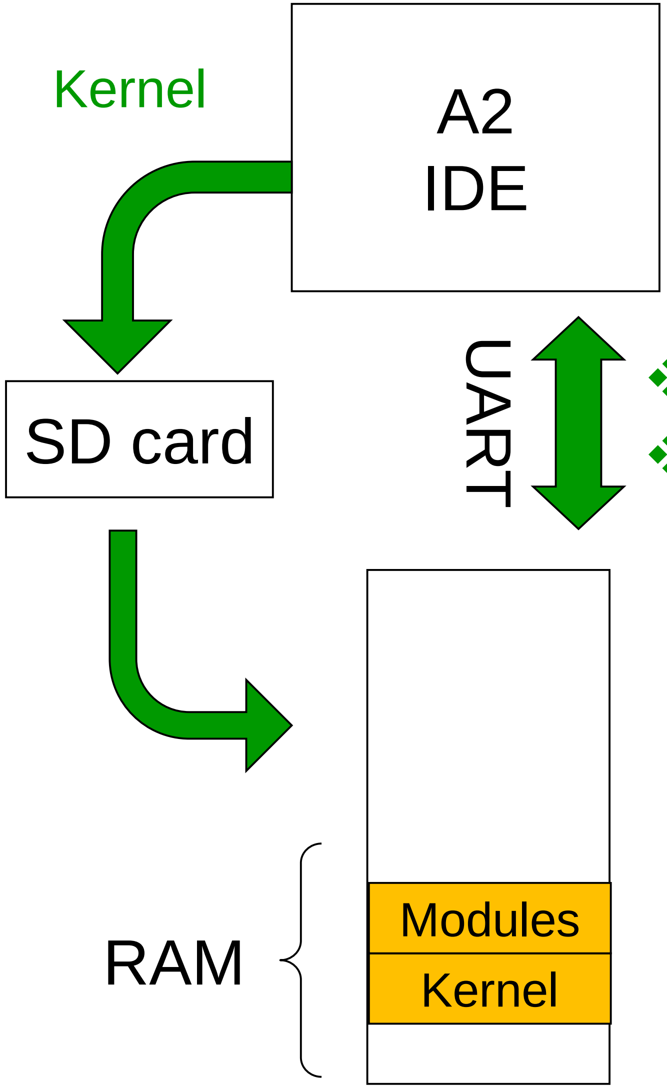
How to Cross-Develop and Build a System

1.2. CROSS DEVELOPMENT

Cross Development Platform

used in the Exercises



- Program Editor
- Cross Compiler
- Boot Linker
- Dialog

} Developer Software

❖Download Modules
❖Interprete Commands

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

} Target Software

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 will use some in the first exercises)
  - Specific low level functions in a Pseudo-Module called SYSTEM

Oberon07

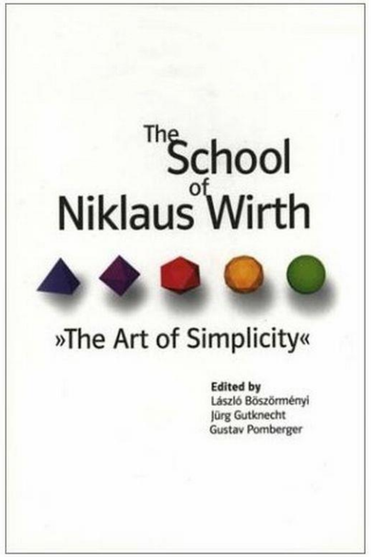
Dialect of Oberon

- Minimal
- Specifically designed for one-pass compilers
  Processor specific functions
- Interrupt procedures
- Pragmatic, predefined functions
- No type OBJECT*, no methods

The compiler used in this course implements Oberon07 as a subset.
Less restrictions apply.

*as opposed to Active Oberon

The art of simplicity



The School of Niklaus Wirth

»The Art of Simplicity«

Edited by
László Böszörményi
Jürg Gutknecht
Gustav Pomberger

- Most recent Compilers by Prof. N. Wirth

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

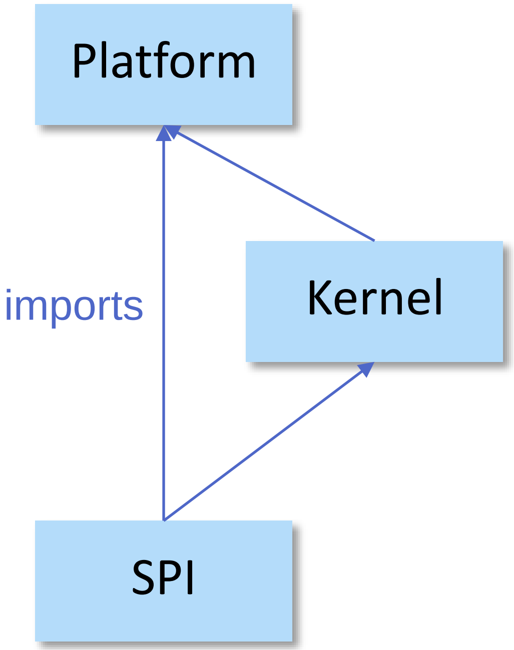
- Fox Compiler, used in the exercises (including all backends and various dialects) ca. 50k lines of code
- gcc / llvm : Millions of lines of code

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: LONGINT;
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;

VAR global: LONGINT; factor: REAL;

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

   PROCEDURE Step*(VAR ticks: LONGINT): REAL;
   VAR previous: LONGINT;
   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

Oberon Language

- Program units
  - MODULE, PROCEDURE （Value, VAR and CONST parameters）
- Data types
  - BOOLEAN, CHAR, SHORTINT, INTEGER, LONGINT, HUGEINT, REAL, LONGREAL, SET, ADDRESS, SIZE
- Structured types
  - ARRAY, RECORD (with type extension), POINTER TO ARRAY, POINTER TO RECORD
- Statements
  - ProcedureCall, Assignments , IF, WHILE, REPEAT, LOOP/EXIT, FOR, CASE, WITH, AWAIT, RETURN, BEGIN ... END

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;
```

Fundamental Types

- **BOOLEAN**
  ```
  b := TRUE; IF b THEN END;
  ```
- **CHAR**
  ```
  c := ‘a’; c := 0AX;
  ```
- **SHORTINT ⊂ INTEGER ⊂ LONGINT ⊂ HUGEINT**
  ```
  i := SHORT(s); l := 10; h := 010H;
  ```
- **REAL ⊂ LONGREAL**
  ```
  r := 1.0; r := 10E0; d := 1.0D2;
  ```
- **SET**
  ```
  s := {1,2,3}; s := s + {5};
  s := s - {5}; s := s * {1..6};
  ```
- **ADDRESS, SIZE**

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, …);`
- **Shifts**
  `ASH(x,y); LSH(x,y); ROT(x,y);`
- **Conversion**
  `SHORT(x); LONG(x); 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
  - `b := SYSTEM.VAL(a, t);`

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

Pseudo Module SYSTEM: ARM Specific

- Register Access
  - `SYSTEM.SP(), SYSTEM.FP(), SYSTEM.LNK()`
  - `SYSTEM.SETSP(x), SYSTEM.SETFP(x), SYSTEM.SETLR(x)`
  - `SYSTEM.LDPSR(b,x), SYSTEM.STPSR(b,x)`
  - `SYSTEM.LDCPR(a,b,c), SYSTEM.STCPR(a,b,c), SYSTEM.FLUSH(x)`

- Floating Point
  - `SYSTEM.NULL(x); SYSTEM.MULD(a,b,c);`

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 {NOTAG}`
  - Procedure without procedure activation frame
- `PROCEDURE {INITIAL}`
  - Procedure that is linked to the beginning of a kernel
- `CODE … END`
  - special statement block that can contain inline assembler code
- `symbol {ALIGNED(32)}`
  - alignment of a symbol (e.g. variable)
- `symbox {FIXED(0x8000))`
  - pinning of a symbol

Special System's Programming Flags and Features

- `POINTER {UNSAFE} TO ...`
  - Unsafe pointer that is assignment compatible with type ADDRESS
- symbol {UNTRACED}
  - Symbol that is invisible to a Garbage Collector

System Programming with Oberon

Bits

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

- 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
    ```

- and / or arithmetic operations and ODD

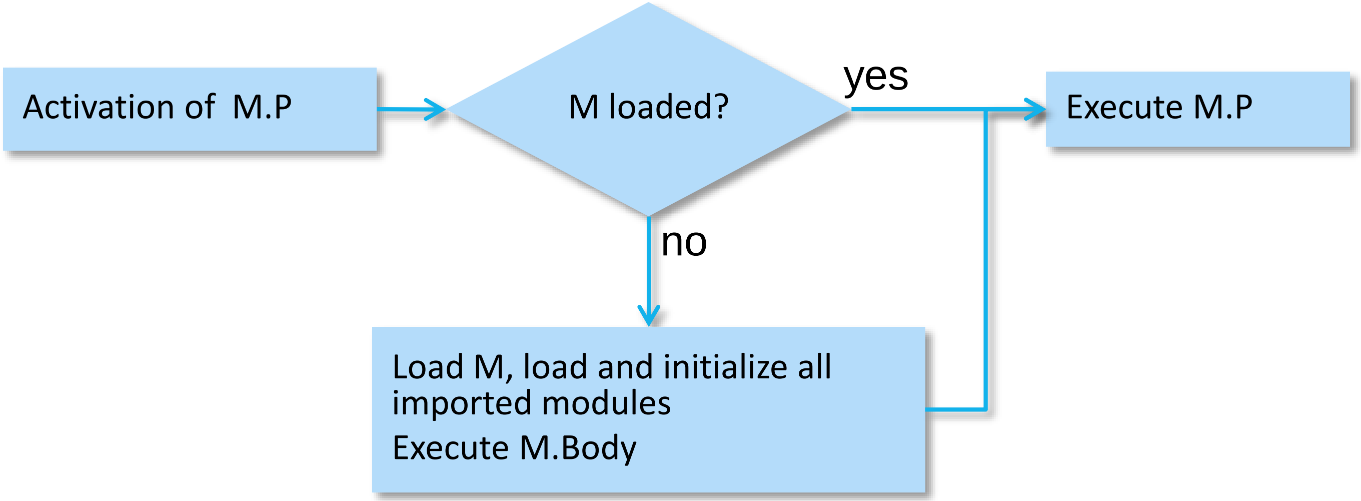
  -
    ```
    VAR i: LONGINT;
    i := i DIV 10H;  -- shift to right by 4
    i := i MOD 10H;  -- and with 0FH
    IF ODD(i) THEN  – test if bit 0 is set
    i DIV 10000H MOD 100H;  --  extract bits 20..27 from i
    ```

Example: Inline-Assembly within Modules

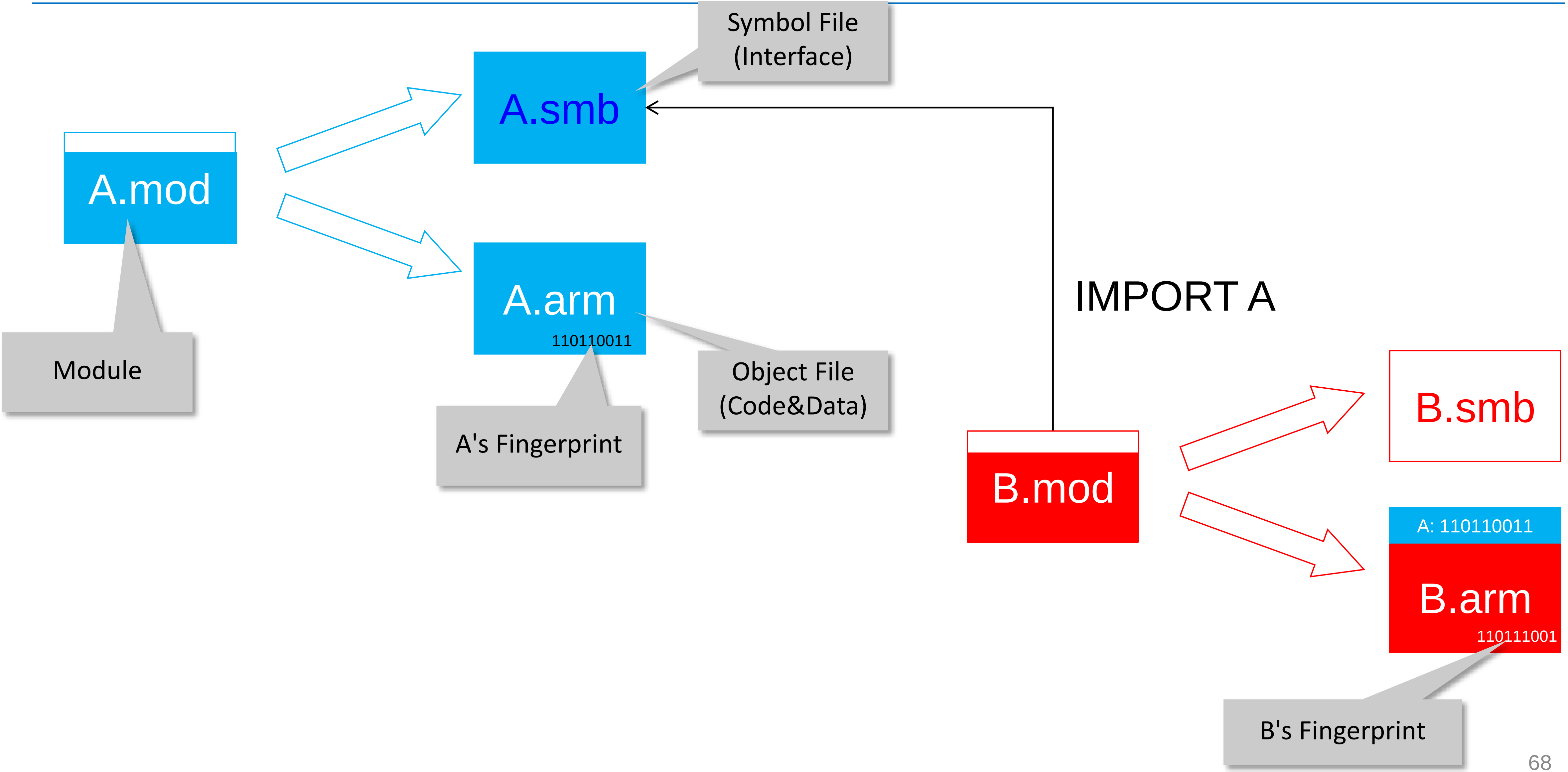
```
MODULE MinimalLED;

IMPORT SYSTEM;

PROCEDURE {INITIAL, NOPAF} Entry;
CODE
    ldr r0, [pc, #someNumber - $ - 8]
    mov r1, #0x30
    b end
    someNumber: d32 0x3f000000
    end:
END Entry;

PROCEDURE {FINAL, NOPAF} Exit;
CODE
    end:
    b end
END Exit;

END MinimalLED.
```

Commands and Module Loading

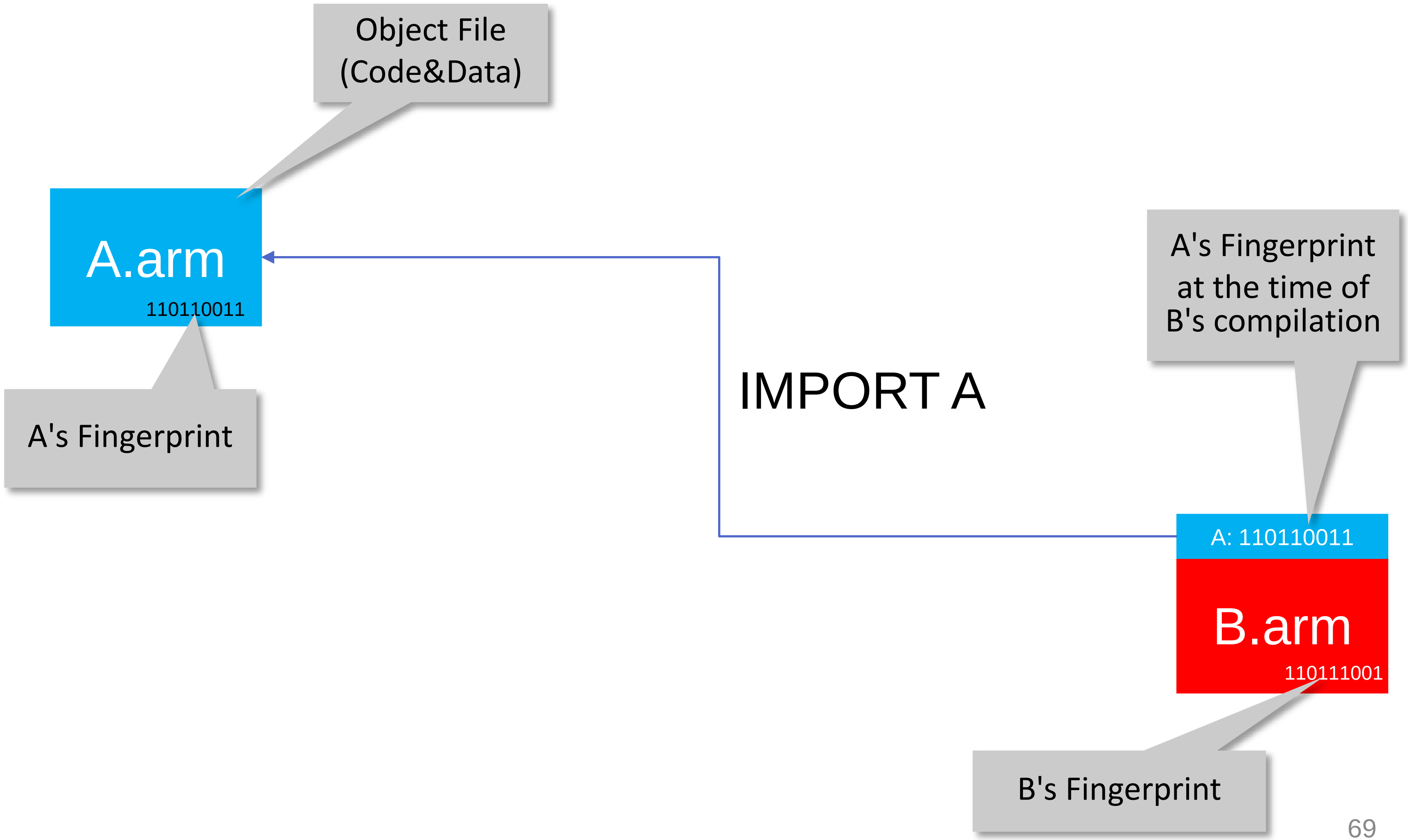
- Modules are loaded on demand
- Statically linked modules are loaded at system-startup
- Exported Procedures without parameters can act as commands
- A modification of a compiled module becomes effective only after (re-) loading the module
- A module M can be unloaded only if no currently loaded module imports M and if M is not statically linked to the Kernel



Compilation Schema



Linking Schema



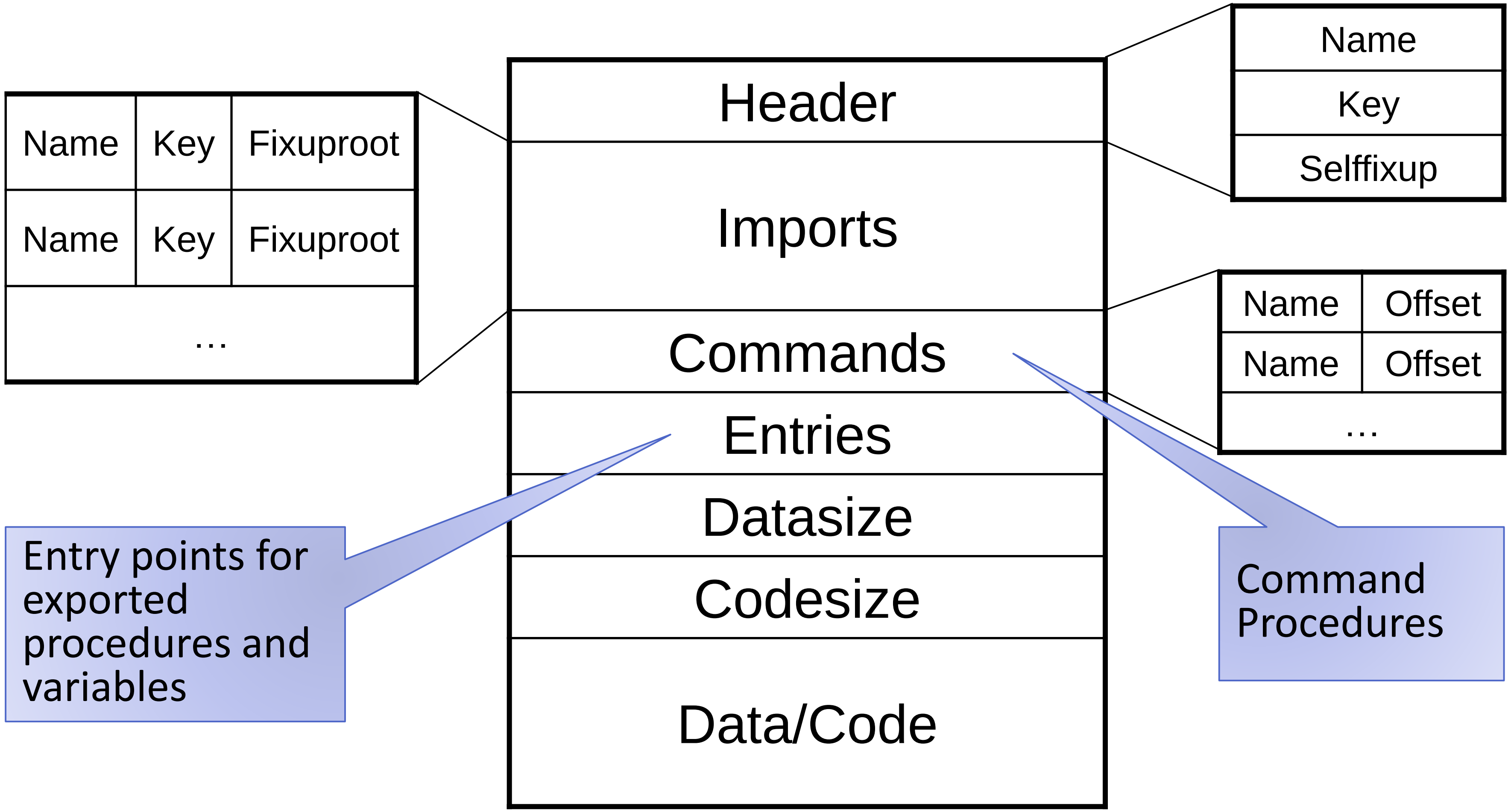
Linking Process

```
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

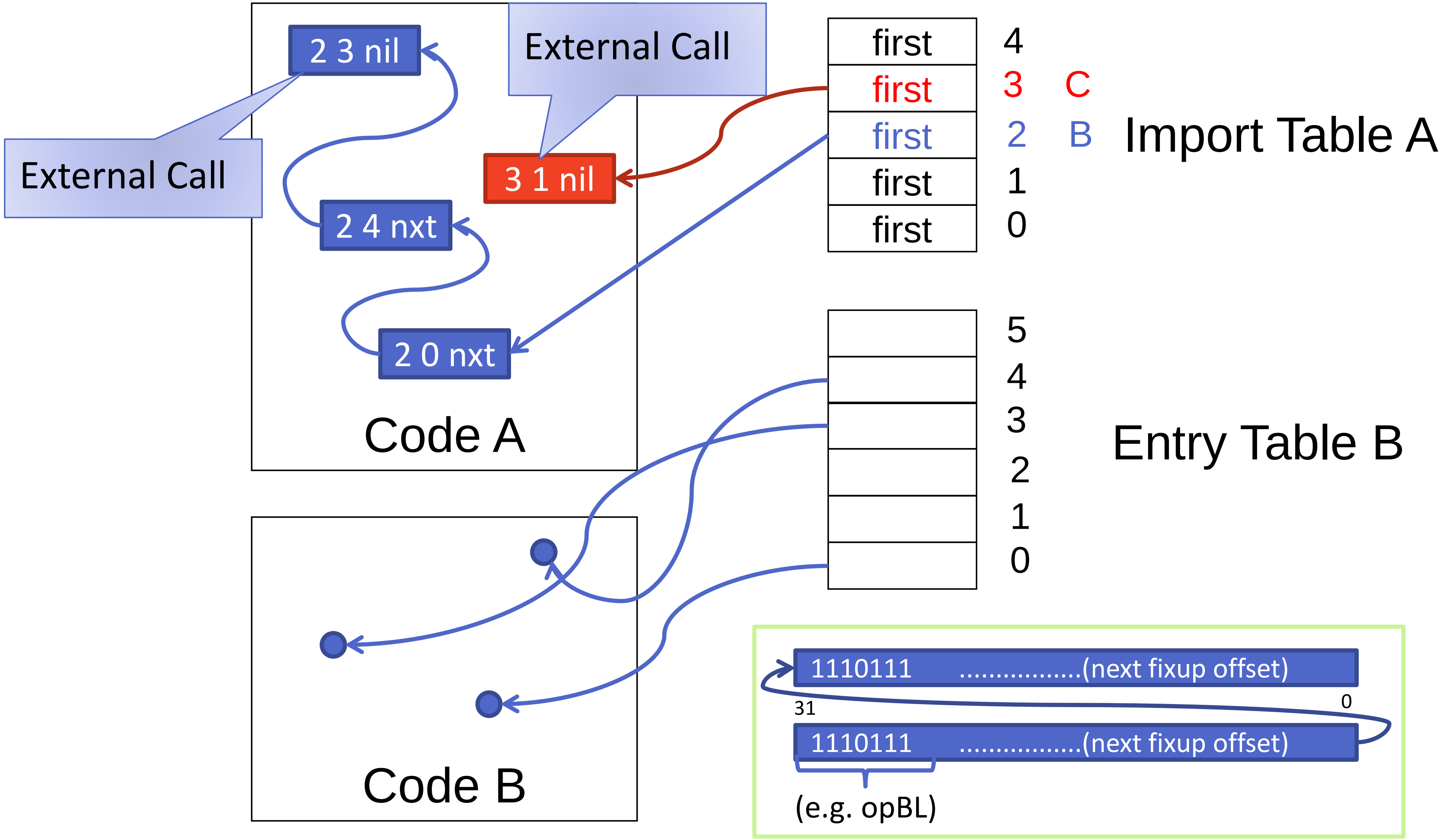


Entry points for exported procedures and variables

Command Procedures

Compiler.Compile -b=ARM --objectFile=Minos

Fixups



External Call

External Call

2 3 nil

3 1 nil

2 4 nxt

2 0 nxt

Code A

Code B

Import Table A

first	4	
first	3	C
first	2	B
first	1	
first	0	

Entry Table B

	5
	4
	3
	2
	1
	0

1110111 ....(next fixup offset)

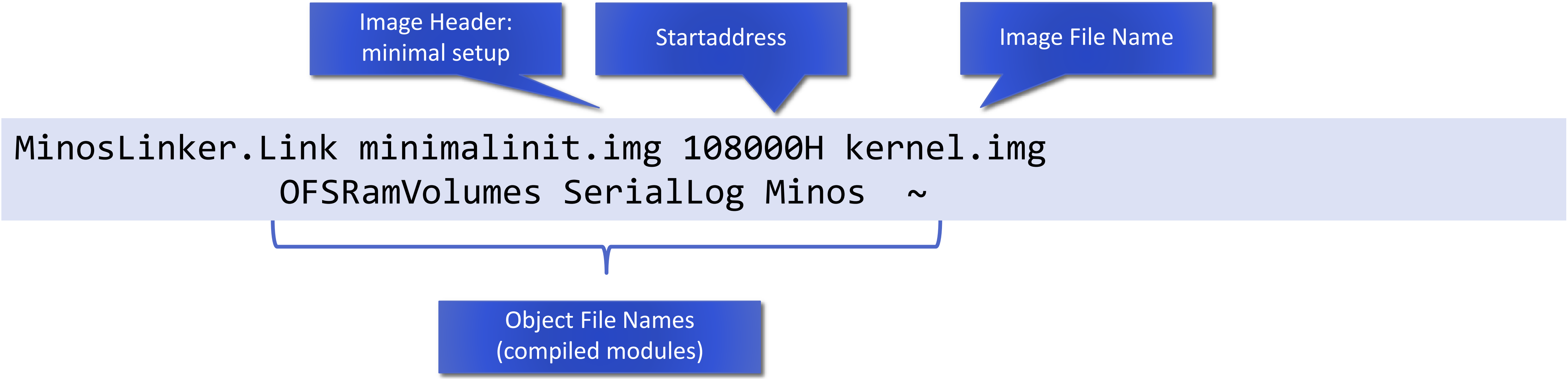
31 0

1110111 ....(next fixup offset)

(e.g. opBL)

Bootfile

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



```
MinosLinker.Link minimalinit.img 108000H kernel.img
                 OFSRamVolumes SerialLog Minos  ~
```

Type Declarations

```
TYPE

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

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

NumberType*= REAL;

DeviceName* = ARRAY DeviceNameLength OF CHAR;

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

Pointer (to Record)
Reference Type

Record
Value Type

Record Entries
(like Variables)

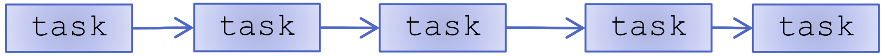
Procedure Type
with Signature

Type Alias

Array Type

Dynamic Array Type

Inheritance (Example)



task task task task task

```
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
   ...
END;
```

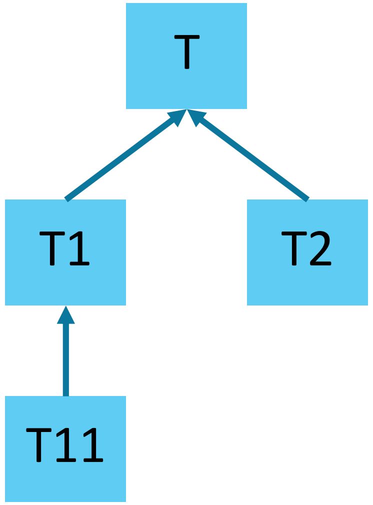
type test

type guard

type test + guard

Runtime Support: Inheritance Scenario

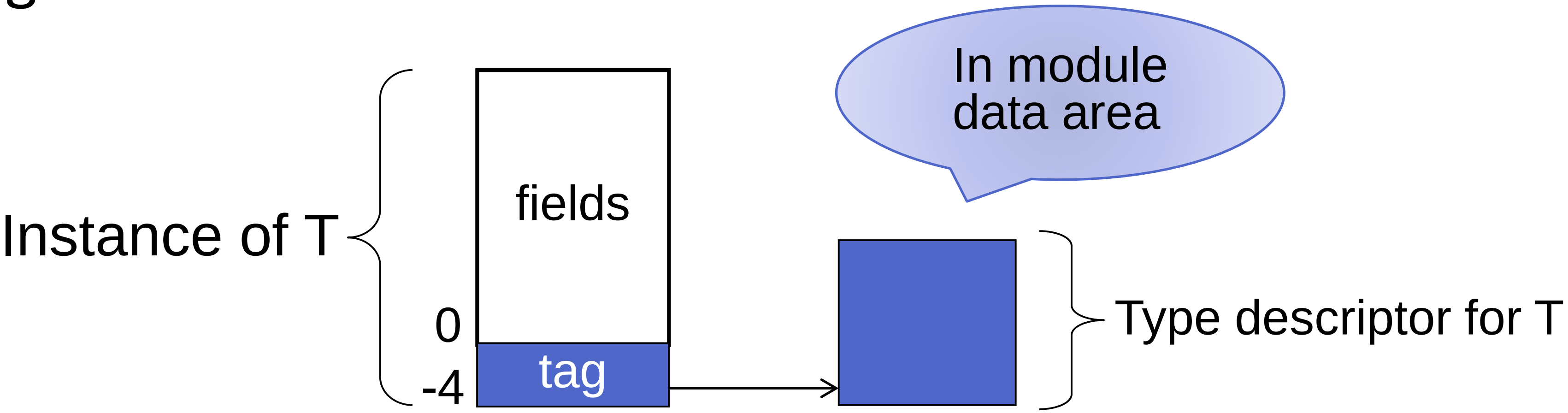
```
TYPE
  T = POINTER TO RECORD (* base type *)
    ... (* base fields *)
  END;
  T1 = POINTER TO RECORD (T) (* extended type *)
    ... (* additional fields *)
  END;
  T2 = POINTER TO RECORD (T)
    ...
  END;
  T11 = POINTER TO RECORD (T1)
    ...
  END;
```



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



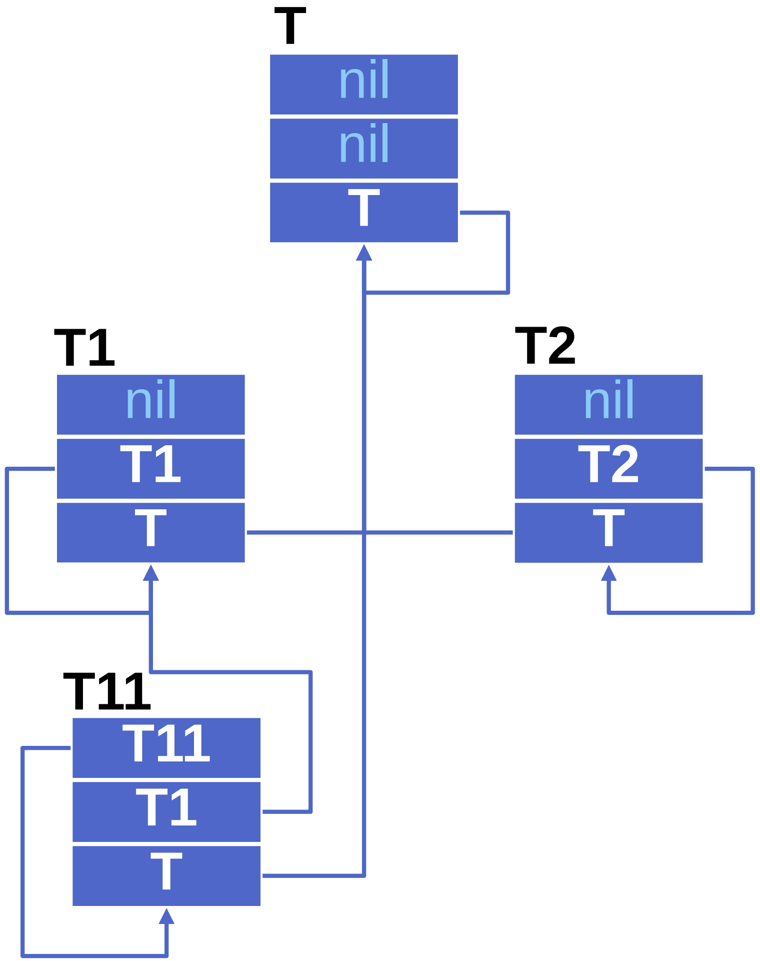
Runtime Support: Type Test Code

- 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

```
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)
```



1.3. MINOS OPERATING SYSTEM

System Startup

RPI (2) VC / firmware

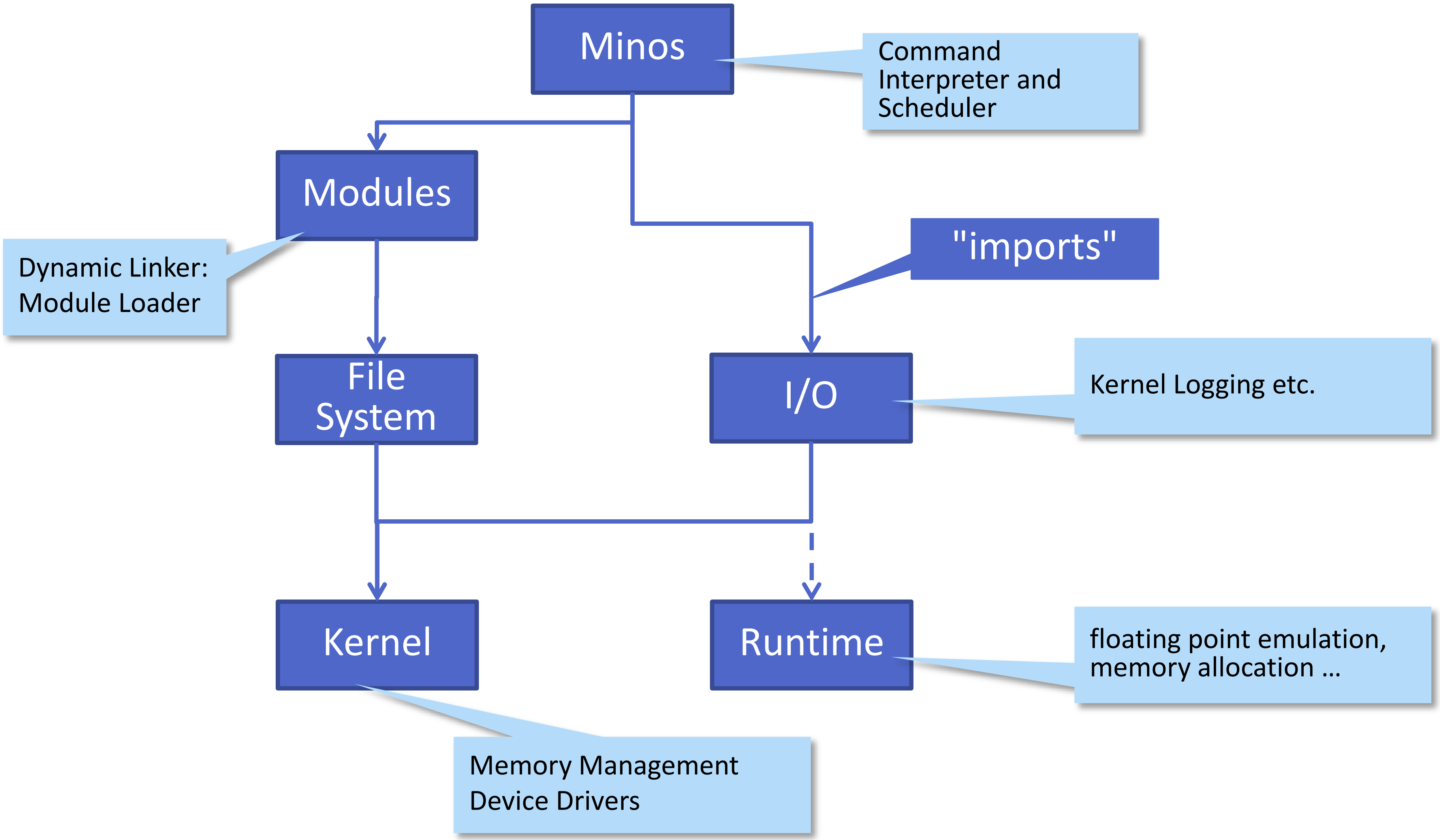
- Initialize hardware
- Copy boot image to RAM
- Jump to OS boot image (Initializer)

OS Initializer (we!)

- Set stack registers for all processor modes
- Setup free heap list and module list
- Initialize MMU & page table
- Setup interrupt handlers & runtime vectors
- Start timer & enable interrupts
- Initialize other runtime data structures
- Initialize UARTs
- Initialize RAM disk
- Enter scheduling loop on OS

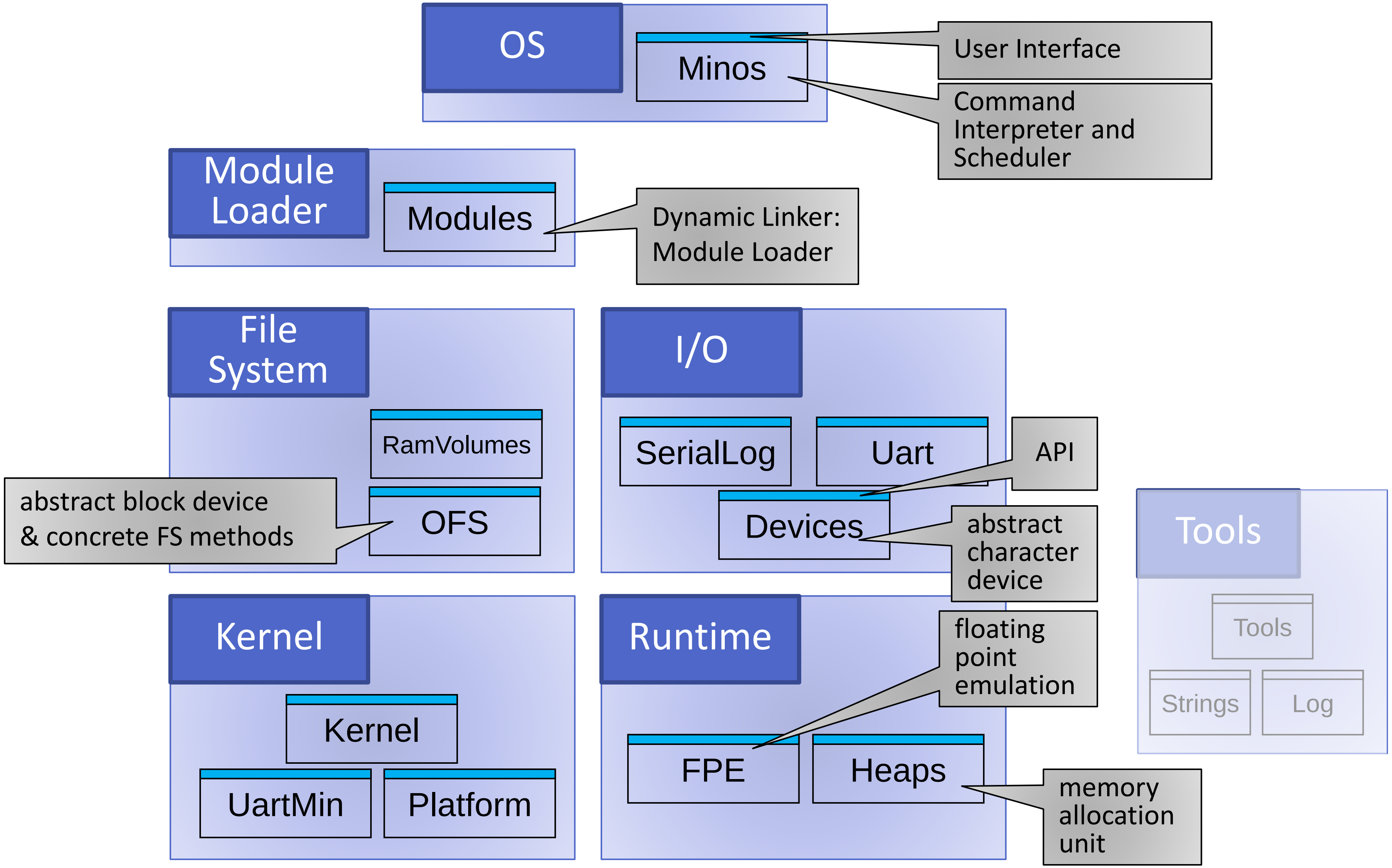
Modular Kernel Structure

The Big Picture


Minos
Command Interpreter and Scheduler
Modules
Dynamic Linker: Module Loader
"imports"
File System
I/O
Kernel Logging etc.
Kernel
Runtime
floating point emulation, memory allocation …
Memory Management
Device Drivers

Modular Kernel Structure

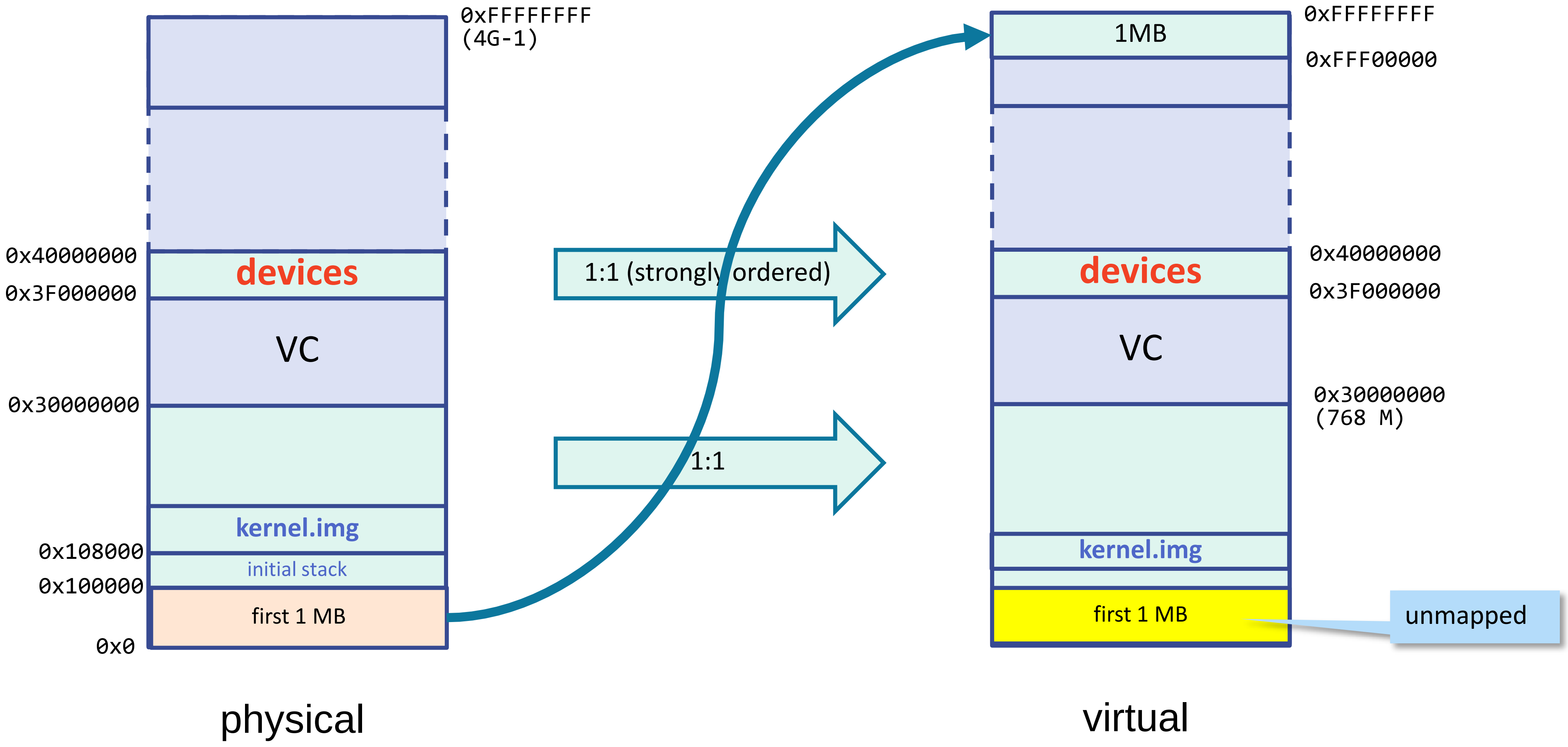
Minos Modules in More Detail



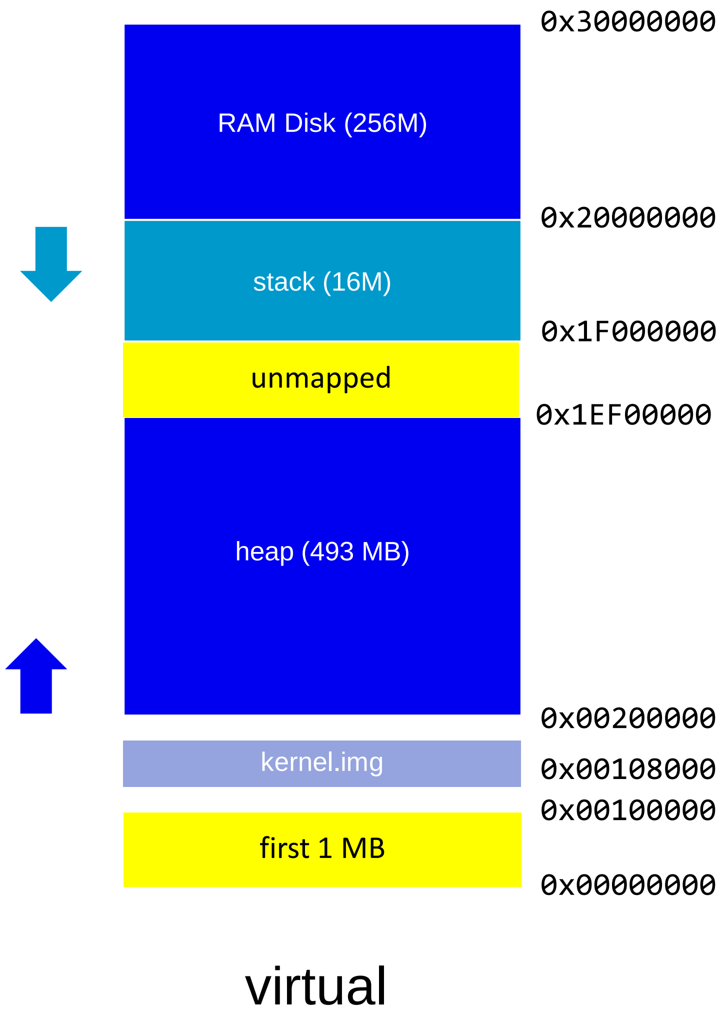
Kernel Module

```
MODULE Kernel;
  IMPORT SYSTEM, Platform;
  TYPE …
    (* types of runtime data structure *)
  VAR …
    (* global runtime data structures *)
  PROCEDURE P* (…); (* exported *)
  BEGIN …
    (* low level routine *)
  END …;
  PROCEDURE … (…); (* internal *)
    (* low level routine *)
  BEGIN …
  END …;
BEGIN …
  (* runtime initialization *)
END Kernel.
```

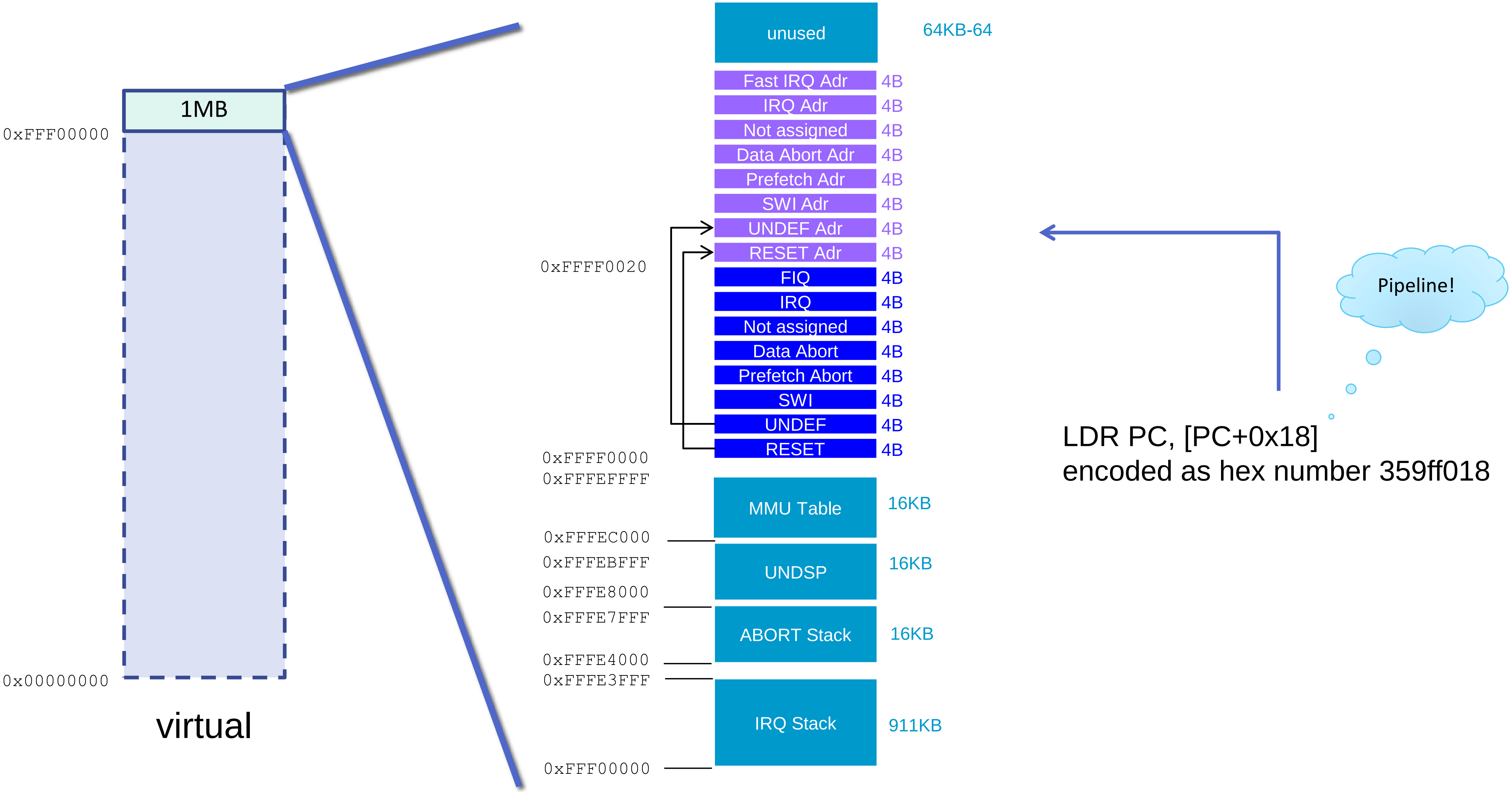
Memory Layout: big picture



Virtual Memory Layout: Heap, Stack, RAMDisk



Virtual Memory Layout: IRQ Table / MMU



LDR PC, [PC+0x18]
encoded as hex number 359ff018

Initialization

Kernel (body)

SYSTEM.LDPSR(u, src)	[instruction MSR]
u = 0	PSR of current processor mode
u = 1	PSR of saved processor mode
src	value to be loaded in PSR, an expression

```
VAR lnk: ADDRESS;
...
BEGIN    (* do not enter any call here --> link register consistency ! *)
   SYSTEM.PUT32(ADDRESSOF(lnk), SYSTEM.LNK());

   SYSTEM.LDPSR( 0, Platform.SVCMode + Platform.FIQDisabled + Platform.IRQDisabled );
   SYSTEM.SETSP(Platform.SVCSP);
   SYSTEM.SETFP(Platform.SVCSP);

   SYSTEM.LDPSR( 0, Platform.IRQMode + Platform.FIQDisabled + Platform.IRQDisabled );
   SYSTEM.SETSP(Platform.IRQSP);
   ...
   (* other modes omitted for brevity *)

   SYSTEM.LDPSR( 0, Platform.SVCMode + Platform.FIQDisabled + Platform.IRQDisabled );   (* Disable
   interrupts, init SP, FP *)

   Init; InitMMU; SetupInterruptVectors; InitHandlers; EnableIRQs;      OSTimer;
   lnk
END Kernel.
```

store link register globally – we are switching the stack!

disable IRQs, stay in SVC mode

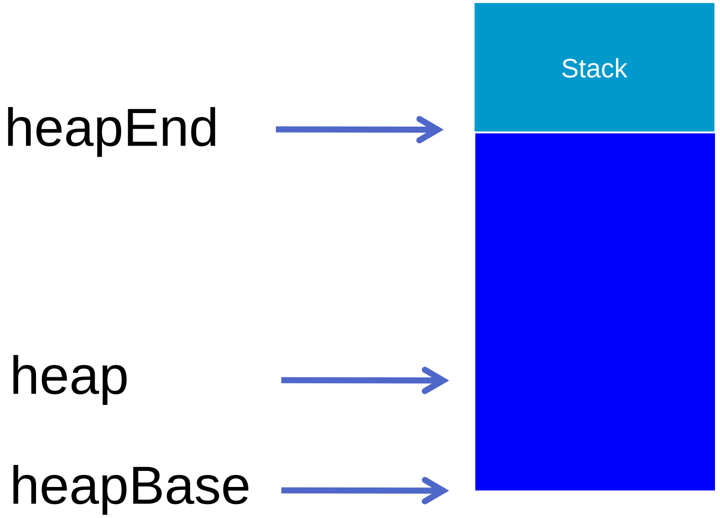
new stack top for this mode

continue execution (next body)

Initialization

Heap

```
MODULE Heaps;

...

BEGIN

  heapStart := Platform.HeapBase;
  heap := Platform.HeapBase;
  heapEnd := Platform.HeapEnd;

END Heaps.
```


Stack
heapEnd
heap
heapBase