

OCAML MEETING 2011  
PARIS, APRIL 15TH

# OCAPIC

programming PIC microcontrollers  
using Objective Caml

PHILIPPE WANG & BENOÎT VAUGON

# OCAPIC

programming PIC microcontrollers  
using Objective Caml

PHILIPPE WANG  
SUPERVISOR

&

BENOÎT VAUGON  
DEVELOPER

The background of the slide features a photograph of four PIC microcontrollers. One large PIC18F4550-44P-A is positioned at the top, showing its 44 pins. Below it, three smaller PIC16C55-20P-A microcontrollers are visible, each with 20 pins. The chips are black with gold-colored pins and are set against a light gray background.

# OCAPIC

programming PIC microcontrollers  
using Objective Caml

picture from <http://fr.wikipedia.org/wiki/Fichier:4pics.jpg>

1/20

# OUTLINE

- What PIC microcontrollers ( $\mu$ C) are.
- What we did with a PIC18  $\mu$ C.
- How we did it.
- What is OCaml in OCAPIC?
- Conclusion.



# PIC MICROCONTROLLERS

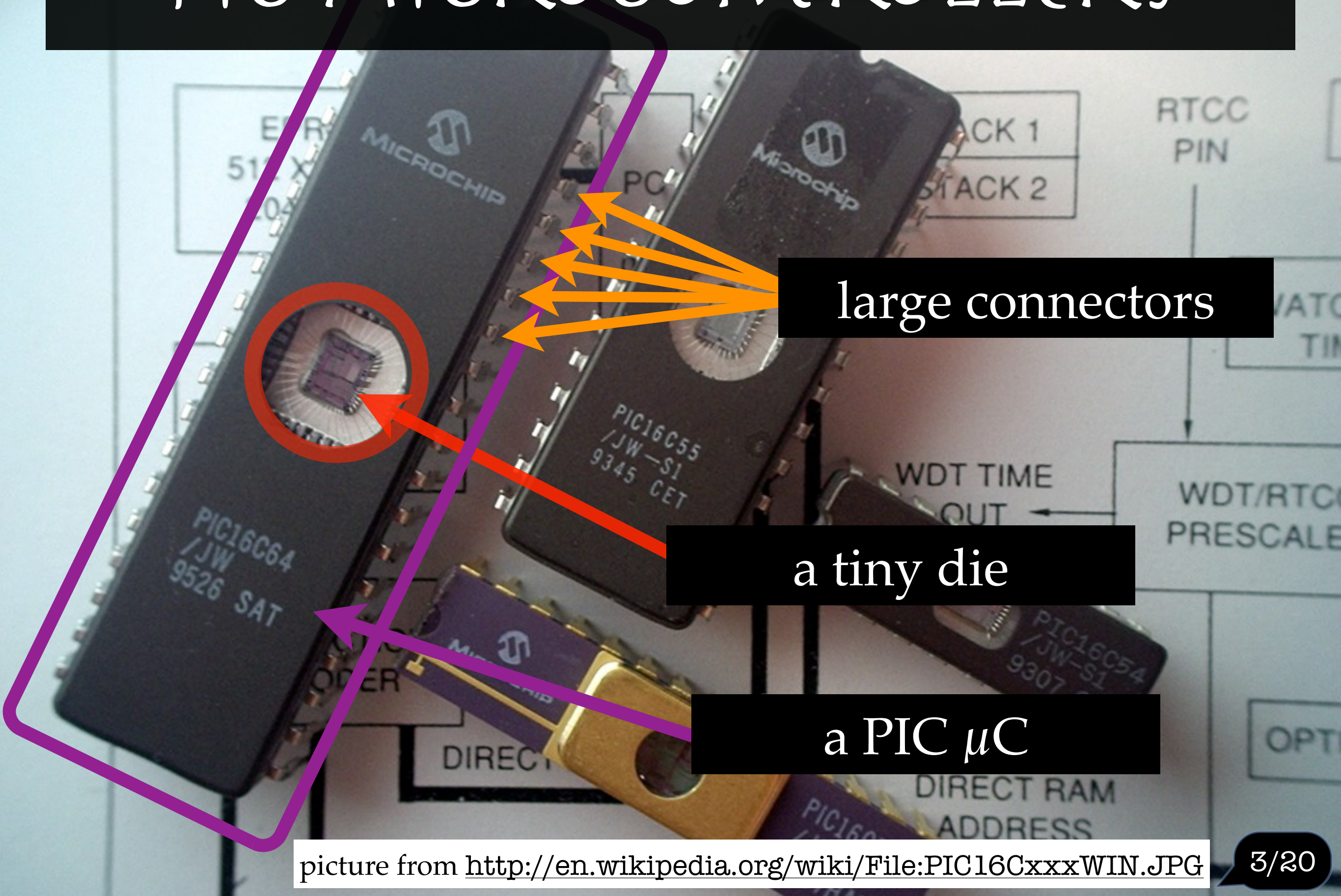


picture from <http://en.wikipedia.org/wiki/File:PIC16CxxxWIN.JPG>

3/20



# PIC MICROCONTROLLERS



picture from <http://en.wikipedia.org/wiki/File:PIC16CxxxWIN.JPG>

3/20



# PIC MICROCONTROLLERS

- programmable
- low-energy consumption
- low-cost (~ 2 €)
- amateur circuits + domestic appliances
- *no operating system*

picture from <http://en.wikipedia.org/wiki/File:PIC16CxxxWIN.JPG>



# PIC18 MICROCONTROLLERS

- 8-bit architecture
- Flash memory: 4 to 128 kB
- RAM: 256 B to 4kB
- Speed: 8 to 16 MIPS

picture from <http://fr.wikipedia.org/wiki/Fichier:4pics.jpg>



# PIC18 MICROCONTROLLERS

- 8-bit architecture *like old times...*
- Flash memory: 4 to 128 kB
- RAM: 256 B to 4kB
- Speed: 8 to 16 MIPS *like old times...*

picture from <http://fr.wikipedia.org/wiki/Fichier:4pics.jpg>

# PIC18 MICROCONTROLLERS

- 8-bit architecture *like old times...*
- Flash memory: 4 to 128 kB *stores the program*
- RAM: 256 B to 4kB
- Speed: 8 to 16 MIPS *like old times...*

picture from <http://fr.wikipedia.org/wiki/Fichier:4pics.jpg>

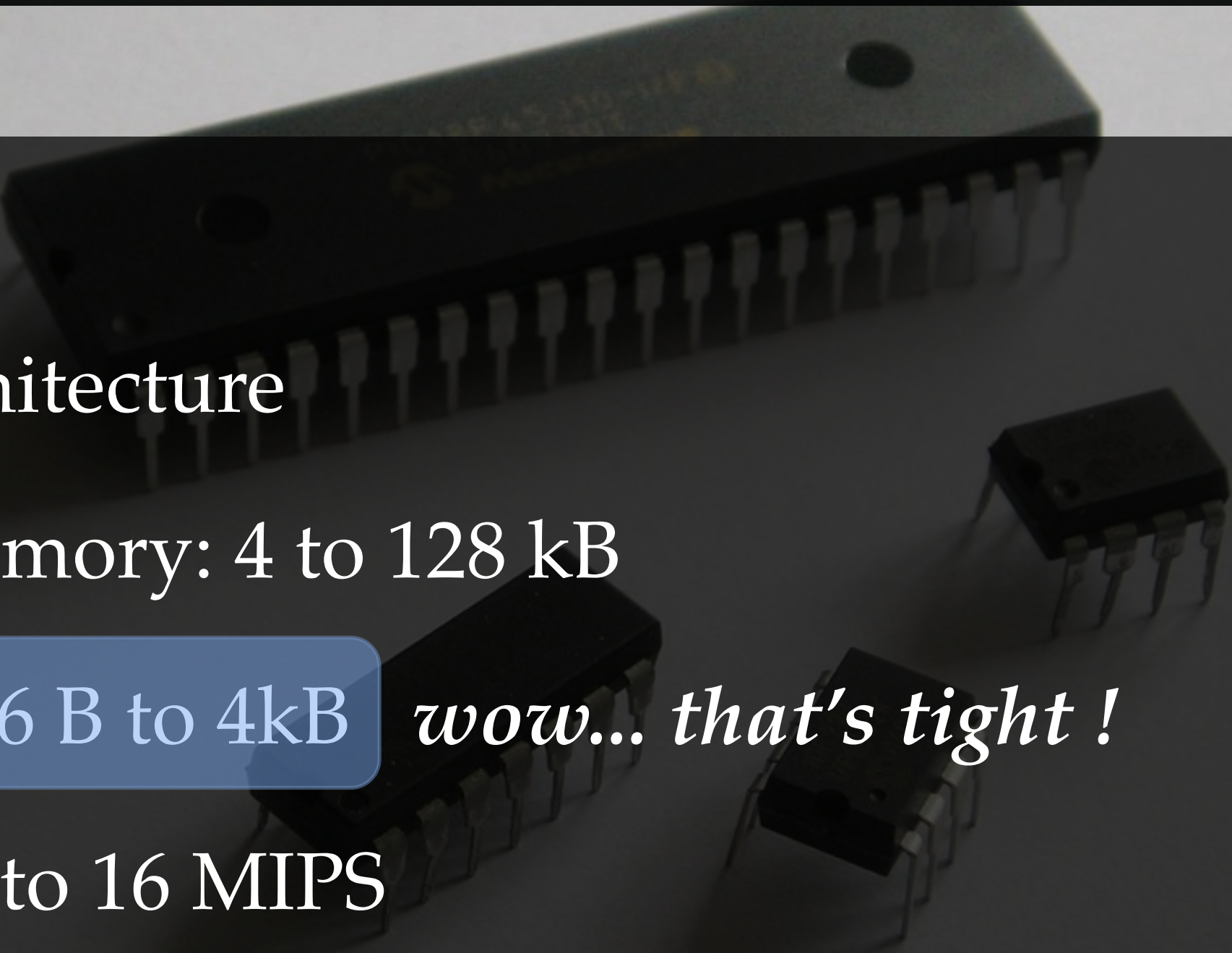


# PIC18 MICROCONTROLLERS

- 8-bit architecture *like old times...*
- Flash memory: 4 to 128 kB *stores the program*
- RAM: 256 B to 4kB *wow... that's tight !*
- Speed: 8 to 16 MIPS *like old times...*

picture from <http://fr.wikipedia.org/wiki/Fichier:4pics.jpg>

# PIC18 MICROCONTROLLERS

- 
- 8-bit architecture
  - Flash memory: 4 to 128 kB
  - RAM: 256 B to 4kB *wow... that's tight !*
  - Speed: 8 to 16 MIPS

picture from <http://fr.wikipedia.org/wiki/Fichier:4pics.jpg>



# APP EXAMPLE



picture taken at JFLA 2011

5/20



# APP EXAMPLE

- a 2-player strategy game
- human *versus* computer
- a board with 24 push-buttons + wiring
- an LCD display + a PIC18  $\mu$ C + a battery

picture taken at JFLA 2011



# APP EXAMPLE



picture taken at JFLA 2011

6/20



# APP EXAMPLE



picture taken at JFLA 2011



**wires**

EXAMPLE

**LCD display**

**PIC18F4620**

**homemade circuit**

pictures taken at JFLA 2011

6/20



# APP EXAMPLE



humans tend to lose...



pictures taken at JFLA 2011



# APP EXAMPLE

implementation in OCaml



```
$ wc -l *.ml*
  78 display.ml
 156 goblet.ml
 256 grid.ml
 117 ia.ml
  60 stacks.ml
  14 types.mli
 681 total
```

**but how to program PIC  $\mu$ Cs in OCaml?**

picture taken at JFLA 2011

# A NEW BACKEND FOR OCAML FOR PROGRAMMING PIC18 $\mu$ C

*let's keep in mind what PIC18  $\mu$ C are*

- 8-bit architecture
- Flash memory: 4 to 128 kB
- RAM: 256 B to 4kB *wow... that's tight !*
- Speed: 8 to 16 MIPS
- no operating system

OCAPIC provides  
*an OCaml VM on a PIC, to run them all*



# PIC & OCAML

- PIC18F4620
  - 3968 registers of 1 byte
  - 8-bit RISC architecture
  - 64 kB of static memory
- OCaml
  - multiparadigm, high-level features
  - automatic safe memory management
  - complex RT lib (*e.g.*, `Pervasives.compare`)

# PROGRAMMING PIC MICROCONTROLLERS WITHOUT OCAML

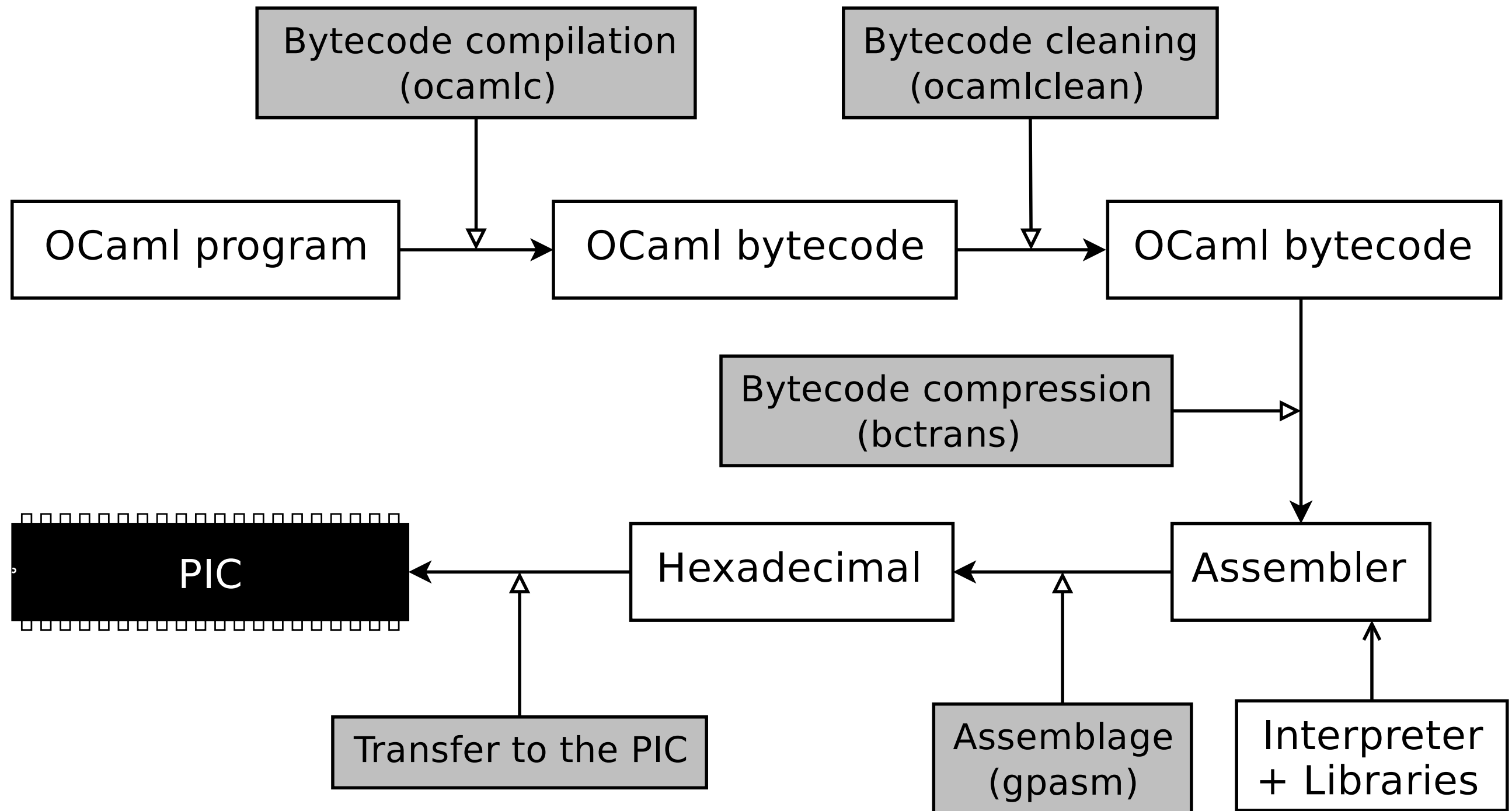
- Assembly
- Compiled languages
  - Basic (BASIC84plus)
  - C (sdcc, SIXPIC, MCC18, ...)
  - Forth (PicForth)
- Interpreted languages
  - Basic (BasicStamps), Forth (FlashForth)
- Virtual Machine
  - PICOBIT (a Scheme VM for PIC)

SAMPLE:

```
tblrd*+  
addwf FSR0L, W  
rcall SUB_PRGM  
btfsc STATUS, C  
incf FSR0H, F  
return
```



# PROGRAMMING PIC MICROCONTROLLERS WITH OCAML



OCAPIC usage overview

# OCAPIC

A NEW BACKEND FOR OCAML  
& A SET OF TOOLS

- OCamlClean
  - a tool for eliminating dead code
- BCtrans
  - compress bytecode for PIC (take back zeros)
- OCaml Virtual Machine
  - implemented in PIC18 ASM



# OCAML CLEAN

## EMPTY BYTECODE FOR EMPTY PROGRAMS

- takes a binary, gives back a lighter binary
- reduce heap occupation  
& remove unused closures (a lot from libraries)
  - ▶ reduce bytecode size & faster initialization
- reduce indirections (closures globalization)
  - ▶ better performance

(can be used in casual programming)

# OCAML BYTECODE → .HEX (COMPRESSION)

- converts an OCaml binary to PIC data
- take back zeros (divides size by 3,5)  
*OCaml bytecode binaries have a lot of zeros for good reasons, but it's a huge waste for PIC  $\mu$ C*
- change heap initialization technique  
*replace computations by a simple copy*
- external (ASM) functions link resolution



# OCAML RUN ON A PIC

## OCAML VM IN PIC ASM

```
wc -l interp.asm runtime.asm
2556 interp.asm
368 runtime.asm
2924 total
```

- uniform data representation, using **16 bits**
  - *15-bit integers, repr. as in orig. OCaml (tag on 1 bit)*
  - *block repr. ~ as in original OCaml : tag & size & data*
- memory garbage collector
  - *a simple Stop&Copy (150 lines of ASM)*
  - *a full cycle runs in less than 1.5ms (RAM is small)*

# OCAPIC LIBRARY

## STANDARD LIBRARY

```
wc -l stdlib.asm  
6757 stdlib.asm (+ ~180 lines of OCaml)
```

- some modules are copied

*Array, ArrayLabels, CamlinternalLazy, CamlinternalMod, Char, Hashtbl, Int32, Int64, Lazy, List, ListLabels, MoreLabels, OO, Queue, Set, Sort, Stack, StdLabels, String, StringLabels*

- some modules are modified

*Buffer+ , CamlinternalOO+ , Gc+, Map+, Obj+, Pervasives\* , Random, Printf\*, Std\_exit, Sys\**

- some modules are left behind

*Arg, Complex, Callback, Digest, Filename, Format, Genlex, Marshal, Nativeint, Parsing, Printexc, Scanf, Stream, Weak*



# OCA PIC LIBRARY

## PIC SPECIFIC LIBRARIES

```
external read_reg : reg -> int = "caml_pic_read_reg";;
```

```
(** [read_reg reg] reads value of  
    a Special Function Register [reg].  
    Return value between 0 and 255. *)
```

PIC I/O

```
external write_reg : reg -> int -> unit = "caml_pic_write_reg";;
```

```
(** [write_reg reg value] writes value in  
    the Special Function Register [reg].  
    [value] should be between 0 and 255. *)
```

```
external set_interruption_handler : (bit -> unit) -> unit =  
    "caml_set_interruption_handler";;
```

```
(** Register an interruption handler. *)
```

interruptions

```
external clear_interruption_handler : unit -> unit =  
    "caml_set_interruption_handler";;
```

```
(** Remove the interruption handler. *)
```

# OCA PIC LIBRARY

## PIC SPECIFIC LIBRARIES

```
module MyDisplay = Lcd.Connect (  
  struct  
    let bus_size = Lcd.Four (* databus mode *)  
    let e = Pic.LATD0 (* command connector *)  
    let rs = Pic.LATD2  
    let rw = Pic.LATD1  
    let bus = Pic.PORTC (* databus port *)  
  end);;  
MyDisplay.init ();;  
MyDisplay.config ();;  
MyDisplay.print_string "Hello world";;
```

LCD display

```
let (ic : t in_channel) = Serial.open_in () in  
  while true do  
    match Serial.receive ic with  
    | Clear -> clear ()  
    | Moveto (l, c) -> moveto l c  
    | Int i -> print_int i  
    | Text t -> print_string t  
  done
```

serial conn.



# OCAPIC + PIC18F4620 BENCHMARKS

execution  
speed

- ~400,000 VM instructions / second
- ~25 machine cycles / VM instruction  
(~18 for a standard processor)
- PICOBIT runs 37,000 VM instructions / second
  - less by 11 times, but for a **different** bytecode

- 64 kB of static memory on PIC18F4620
- Interpreter: 4954 B (7.6%)
- Runtime library: 4254 B (6,5%)
- Bytecode: up to 55kB (86%)

static  
memory  
occupation

# CONCLUSION

- Full OCaml frontend & Good performance **thanks to a virtual machine!** (+ a set of tools)
- Works with any PIC18 (*but check the memory size*)
- Other ideas
  - make other languages target OCaml VM
  - port OCaml VM on other architectures
  - try other garbage collection algorithms
  - try external RAM

[http://www.algo-prog.info/ocaml\\_for\\_pic/](http://www.algo-prog.info/ocaml_for_pic/)





THANKS!

*don't miss the demo session  
this afternoon ; - )*

ANY QUESTIONS?

picture taken at JFLA 2011

20/20