OCAML MEETING 2011

PARIS, APRIL 15TH

OCAPIC

programming PIC microcontrollers using Objective Caml

PHILIPPE WANG & BENOÎT VAUGON

UPMC SORBONNE UNIVERSITÉS

LIP6

ENSTA ParisTech

OCAPIC

programming PIC microcontrollers using Objective Caml

PHILIPPE WANG
SUPERVISOR

&

BENOÎT VAUGON
DEVELOPER

OCAPIC

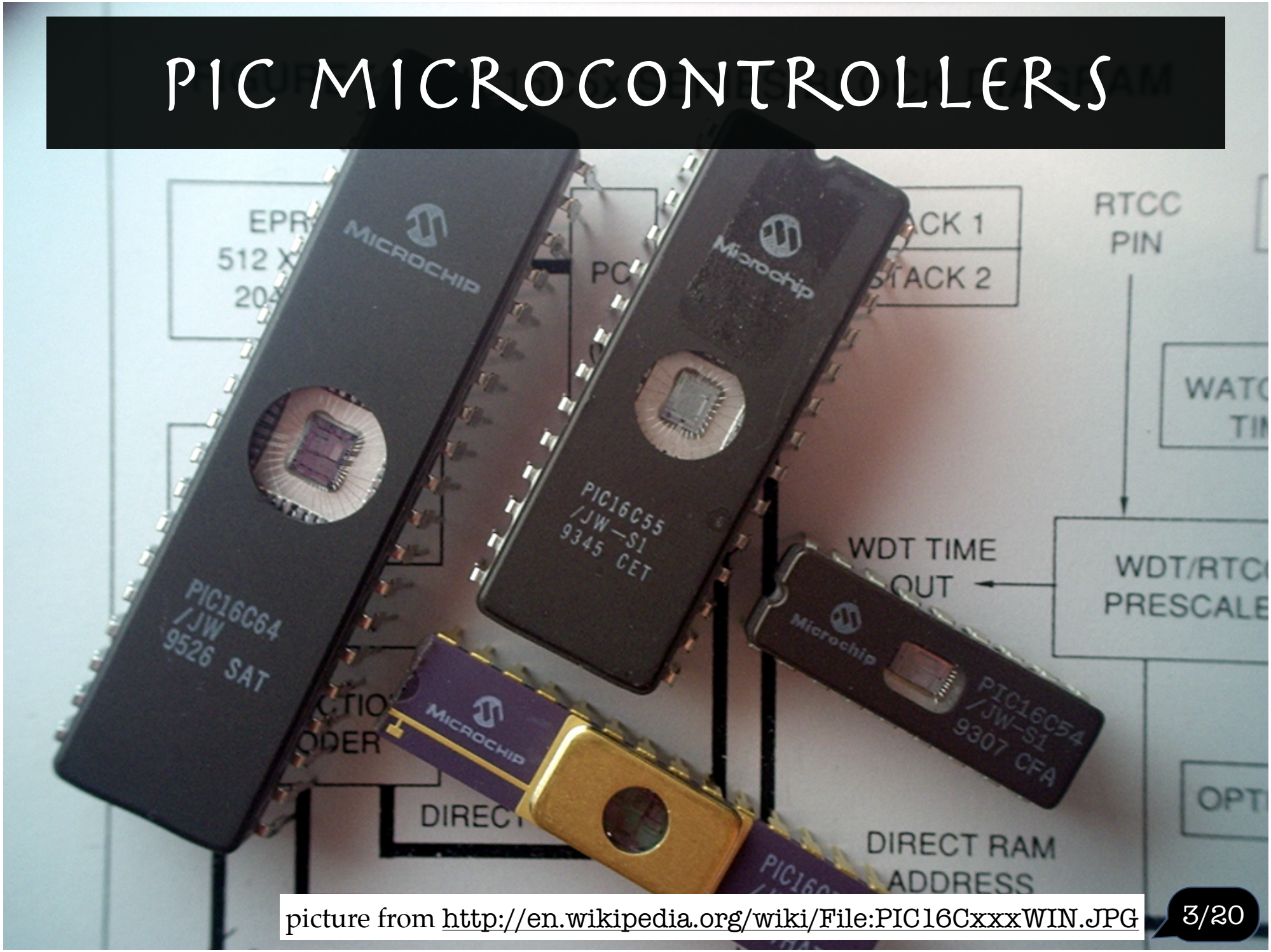
programming PIC microcontrollers
using Objective Caml

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

OUTLINE

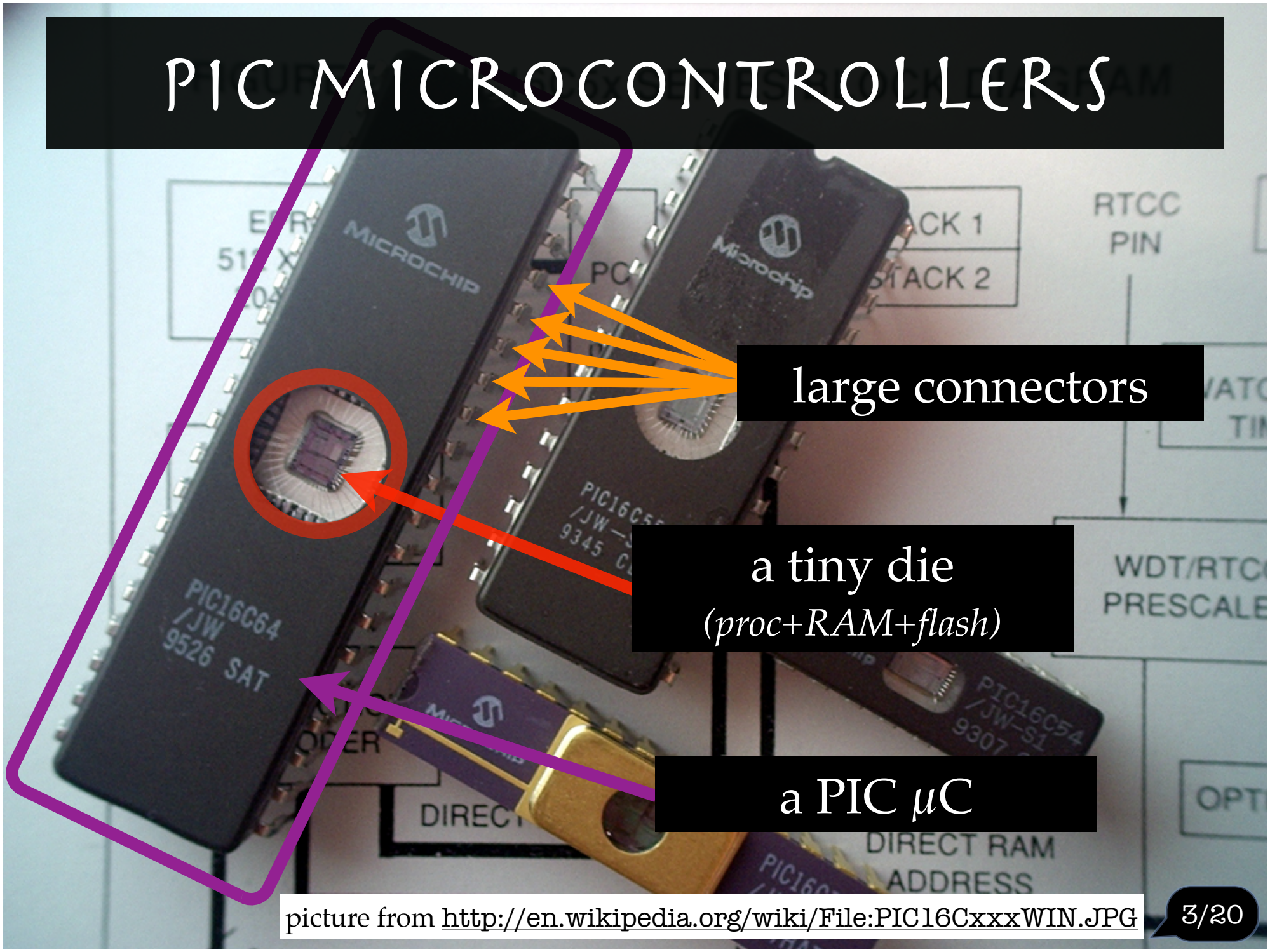
- What PIC microcontrollers (μC) are.
- What we did with a PIC18 μC.
- How we did it.
- What is OCaml in OCAPIC?
- Conclusion.

PIC MICROCONTROLLERS



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

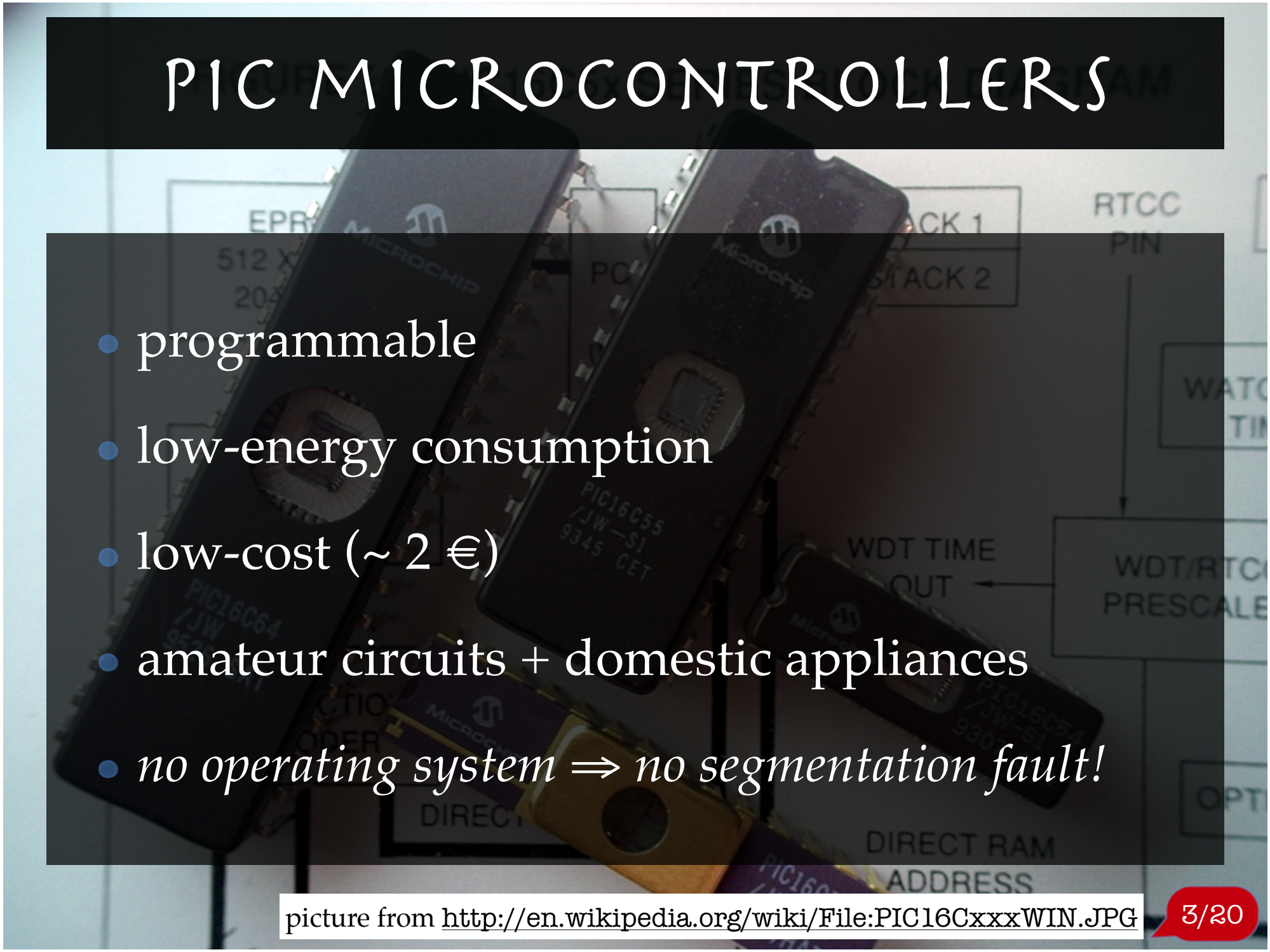
PIC MICROCONTROLLERS



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

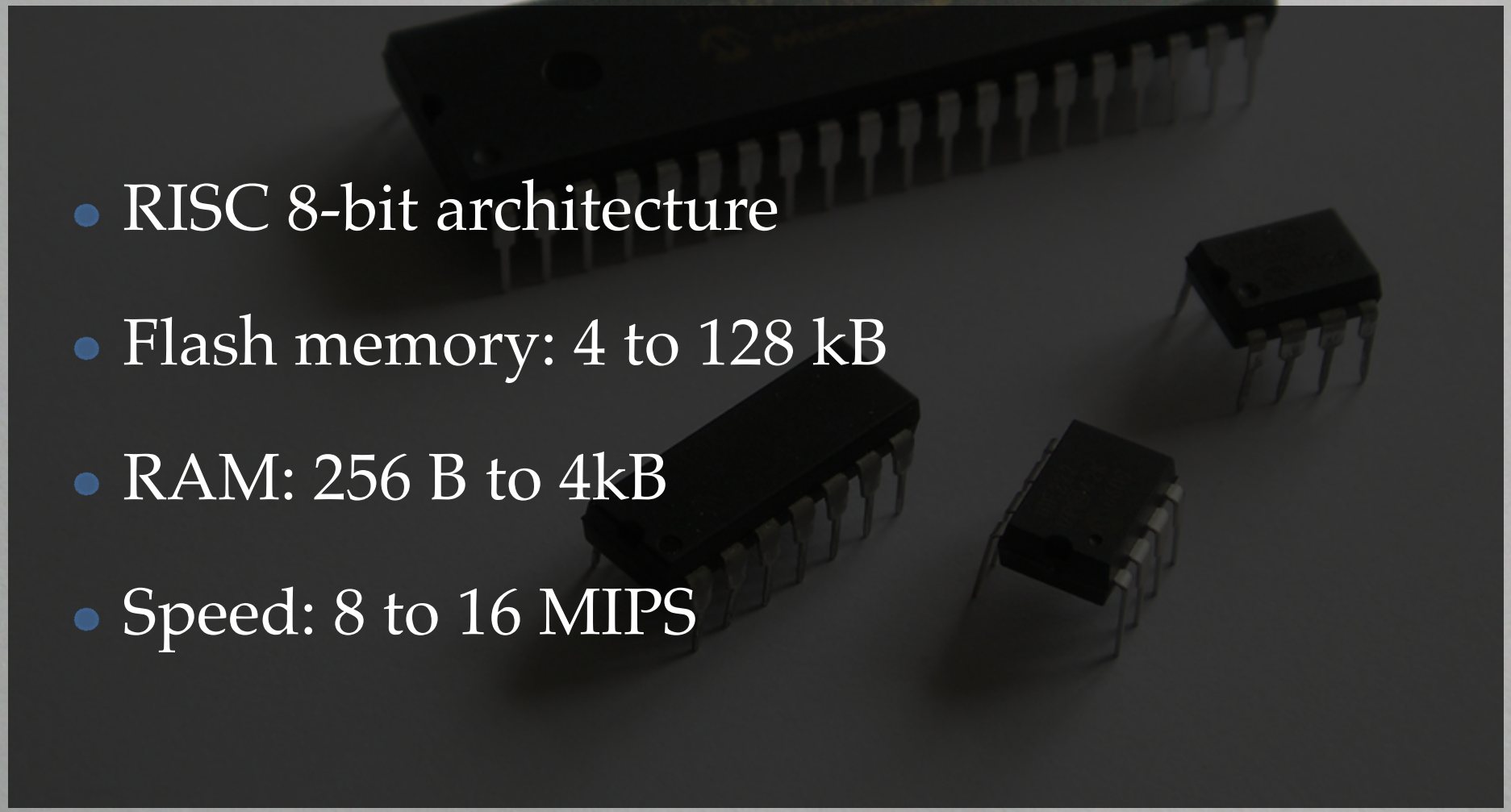
PIC MICROCONTROLLERS

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



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

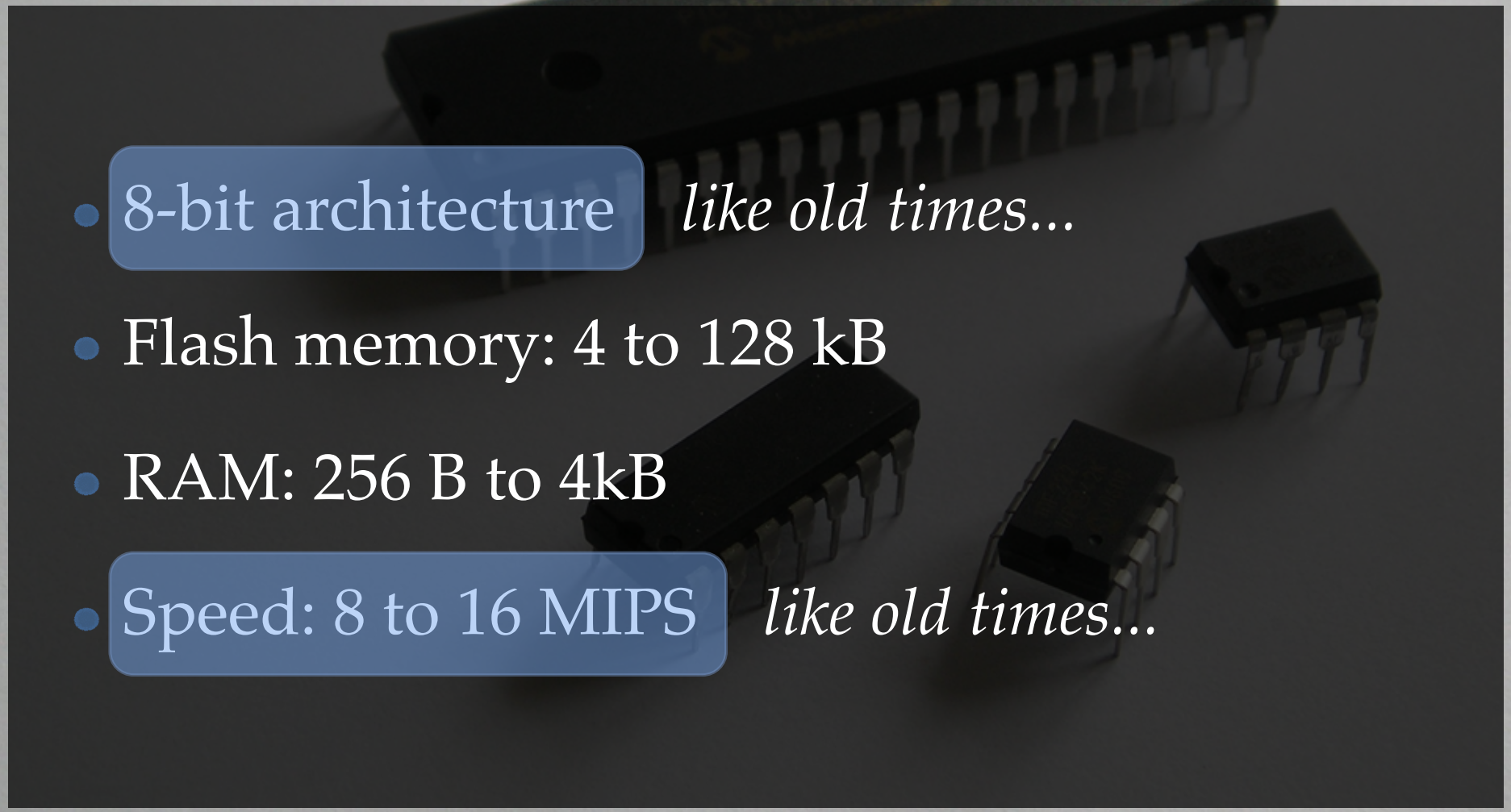
PIC18 MICROCONTROLLERS



- RISC 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...*

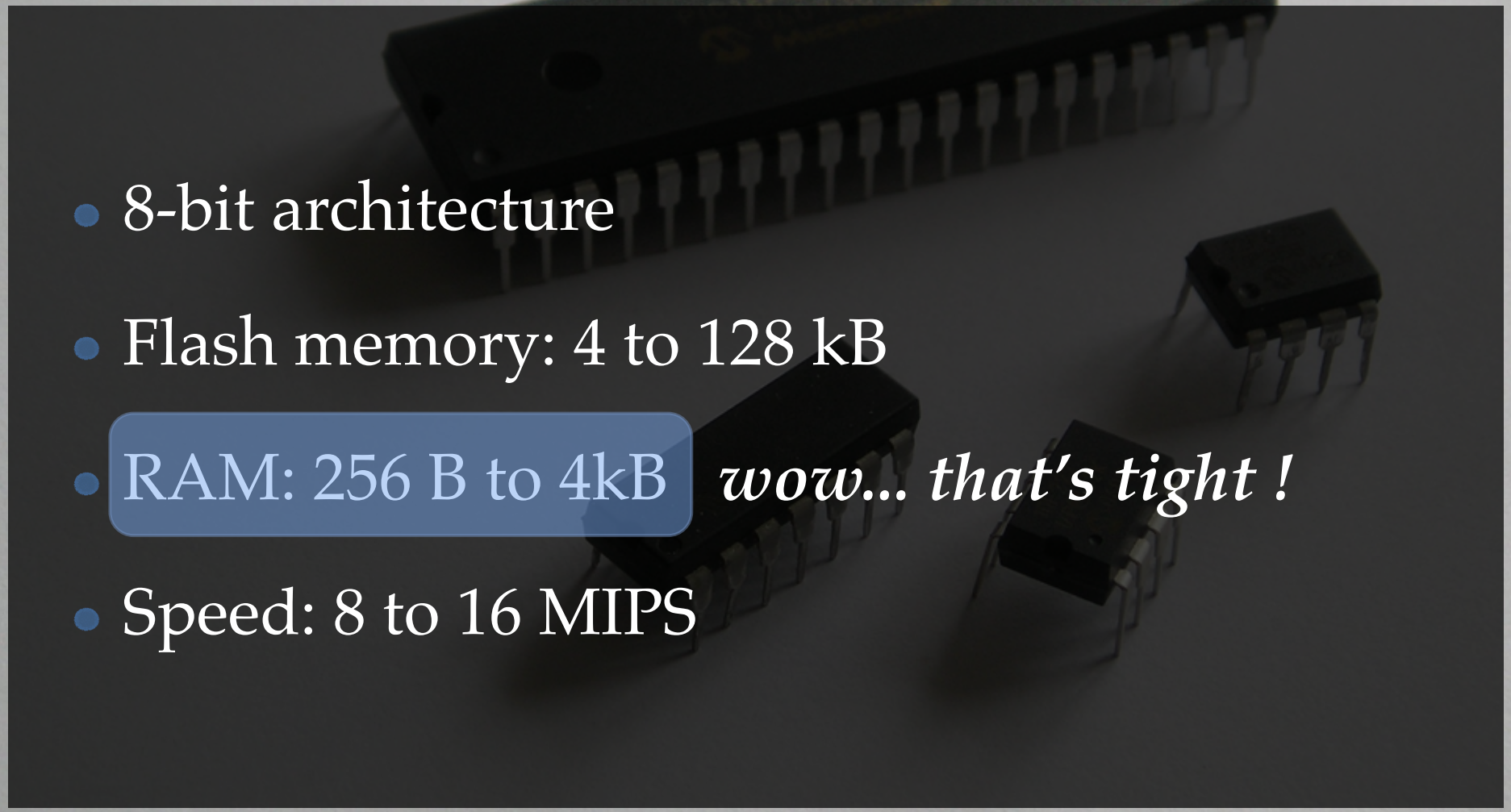
picture from http://fr.wikipedia.org/wiki/Fichier:4pics.jpg

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

APP EXAMPLE

- a 2-player strategy game
- human *versus* computer

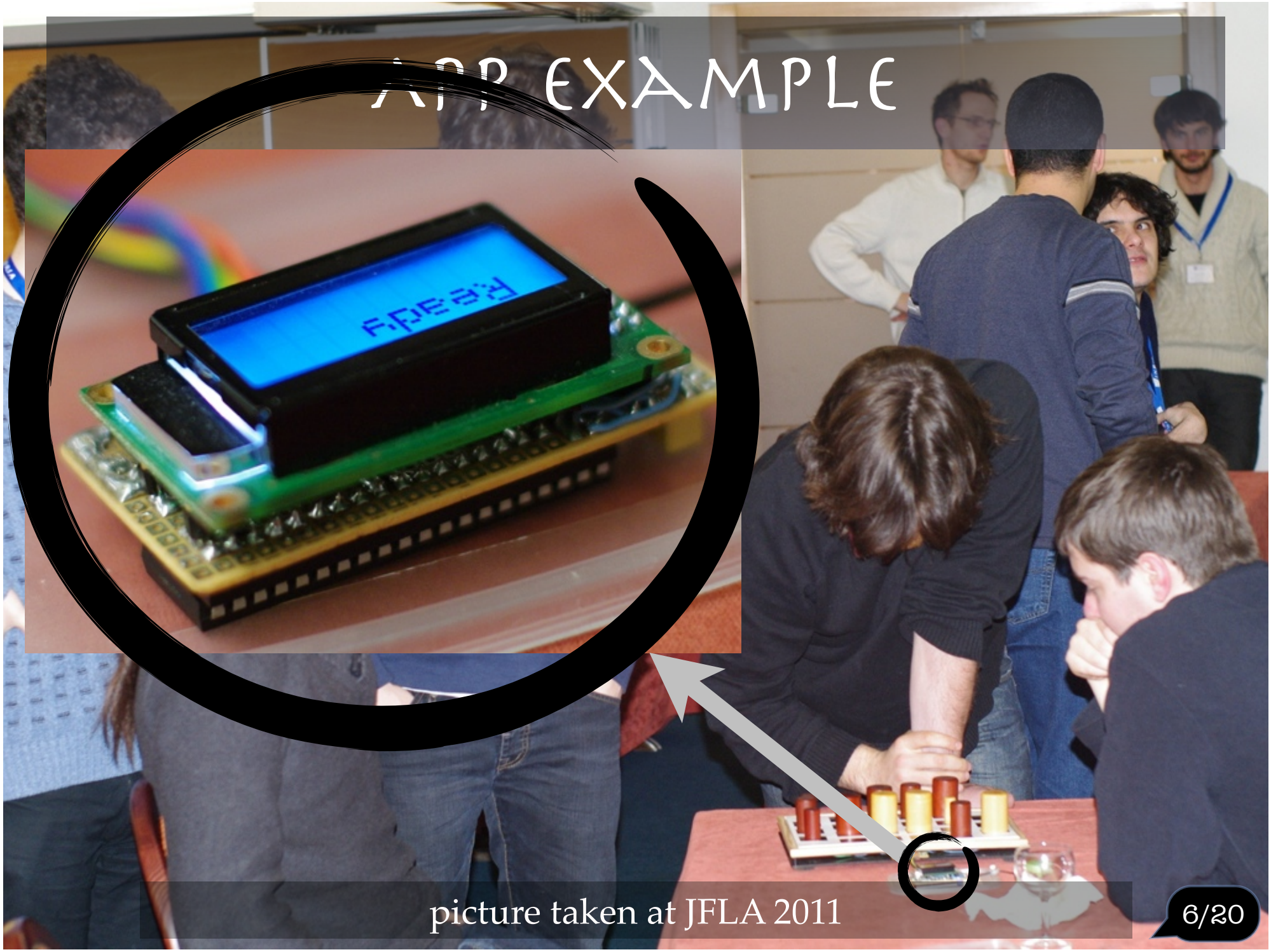
- a board with 24 push-buttons + wiring
- an LCD display + a PIC18 μC + a battery

picture taken at JFLA 2011

APP EXAMPLE

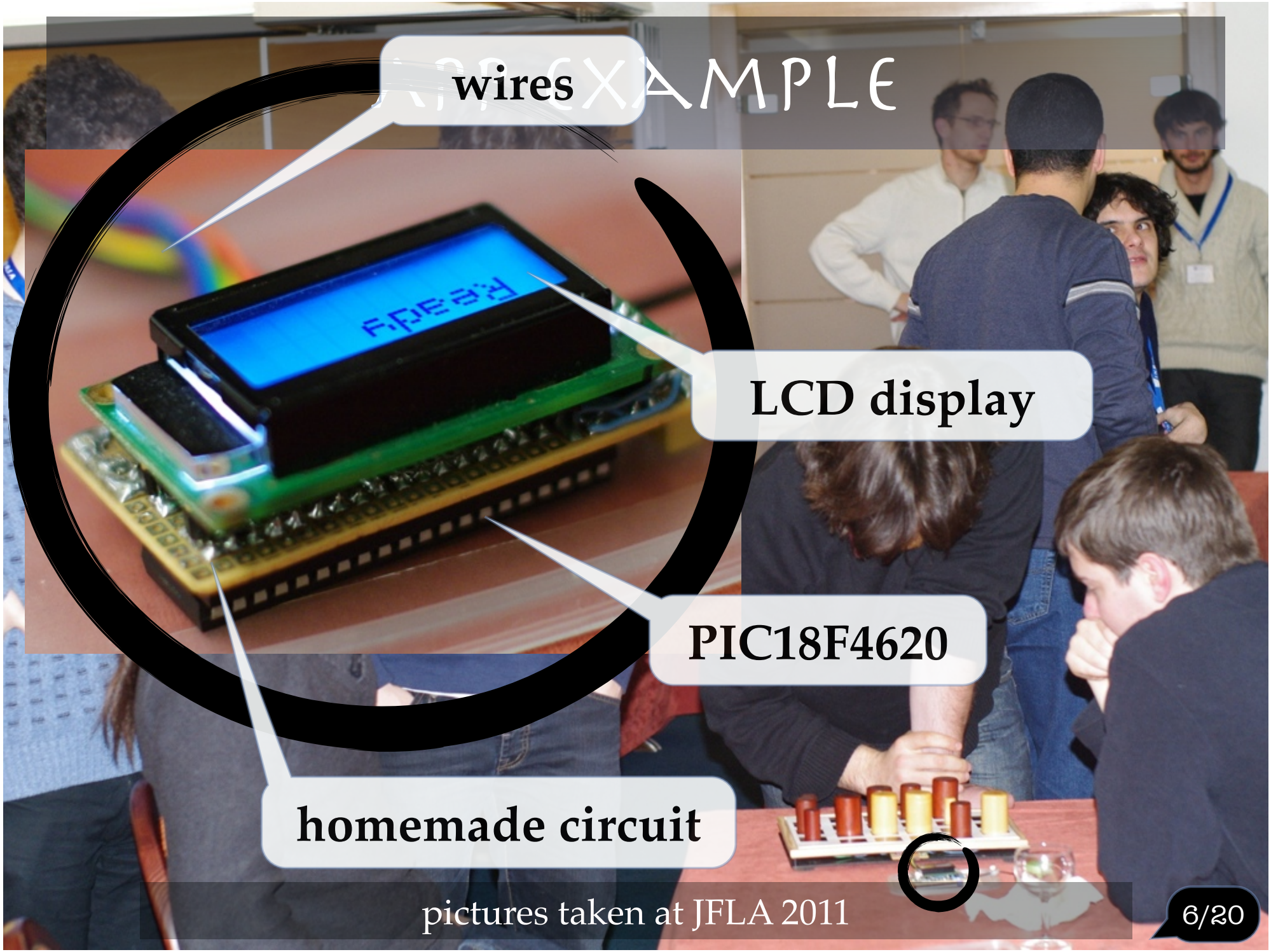
picture taken at JFLA 2011

APP EXAMPLE



picture taken at JFLA 2011

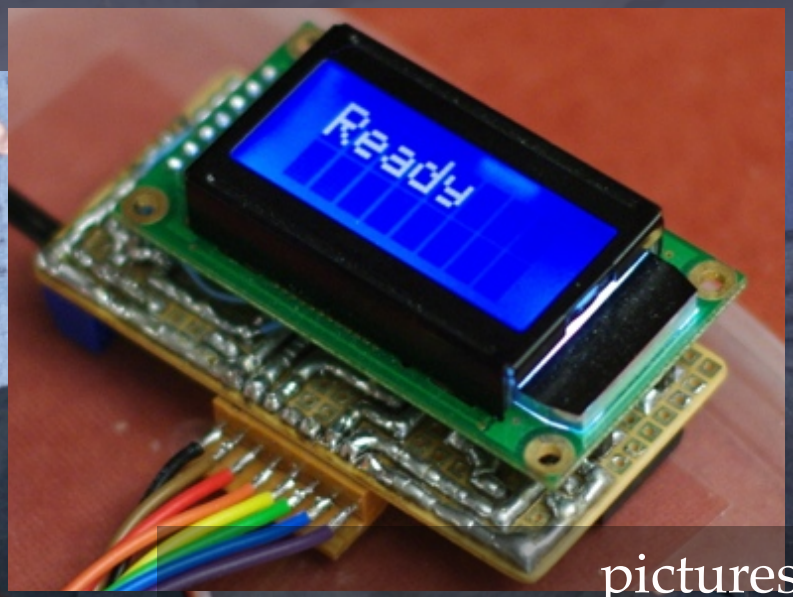
EXAMPLE



pictures taken at JFLA 2011

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 μCs in OCaml?

picture taken at JFLA 2011

A NEW BACKEND FOR OCAML FOR PROGRAMMING PIC18 µC

let's keep in mind what PIC18 µC 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 (among other things)
an OCaml VM on a PIC, to run them all

PIC & OCaml

- PIC18F4620
  - 3968 registers of 1 byte (→ RAM)
  - 8-bit RISC architecture
  - 64 kB of flash memory

- OCaml
  - multiparadigm, high-level features
  - automatic safe memory management
  - complex RT lib (*e.g.*, `Pervasives.compare`)

PIC & OCAML

- PIC18F4620
  - 3968 registers of 1 byte (→ RAM)
  - 8-bit RISC architecture
  - 64 kB of flash memory

tight resources

- OCaml
  - multiparadigm, high-level features
  - automatic safe memory management
  - complex RT lib (*e.g.*, `Pervasives.compare`)

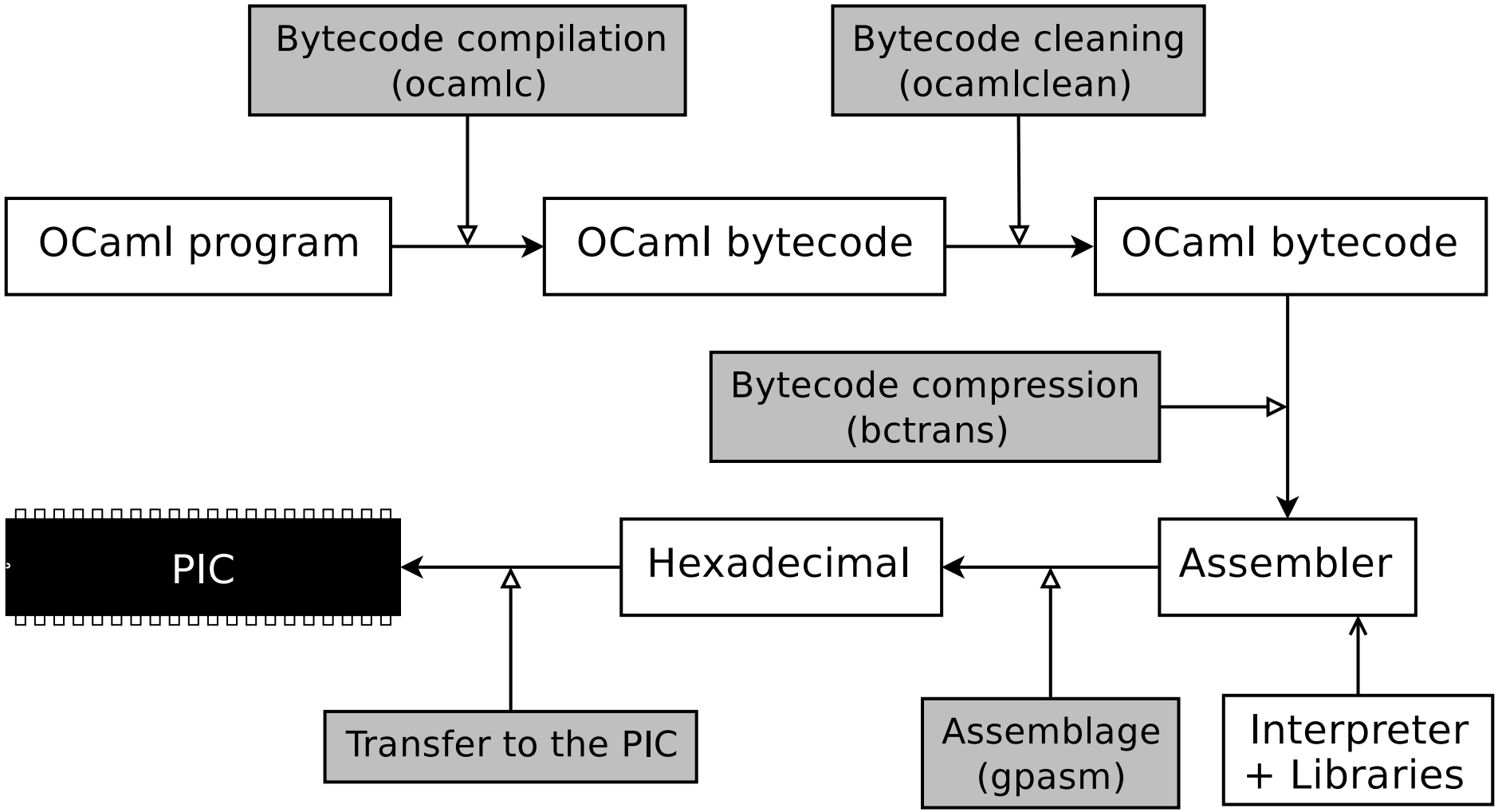
rich language

PROGRAMMING PIC MICROCONTROLLERS WITHOUT OCAML

- Assembly language
- 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
```

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
- **(+ a PIC environment simulator)**

OCAMLCLEAN
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)

COMPRESSION WITH BCTRANS
(OCAML BYTECODE → .HEX)

- converts an OCaml binary into 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 µC
- change heap initialization technique
  replace computations by a simple copy
- external (ASM) functions link resolution

OCAMLVM IN PIC ASM

OCAMLRUN ON A PIC

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

- memory garbage collector
  - *a simple Stop&Copy (150 lines of ASM)*
  - *a full cycle runs in less that 1.5ms (RAM is small)*
- 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*

OCAPIC LIBRARY

STANDARD LIBRARY

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

- part that can't be in OCaml: `stdlib.asm`

  Arithmetic operators, Pervasives.compare, ...

- some OCaml modules are copied

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

- some OCaml modules are modified

  Buffer+ , CamlinternalOO+ , Gc+, Map+, Obj+, Pervasives , Random, Printf*, Std_exit, Stream, Sys*,*

- some OCaml modules are left behind

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

OCAPIC 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. *)

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. *)
```

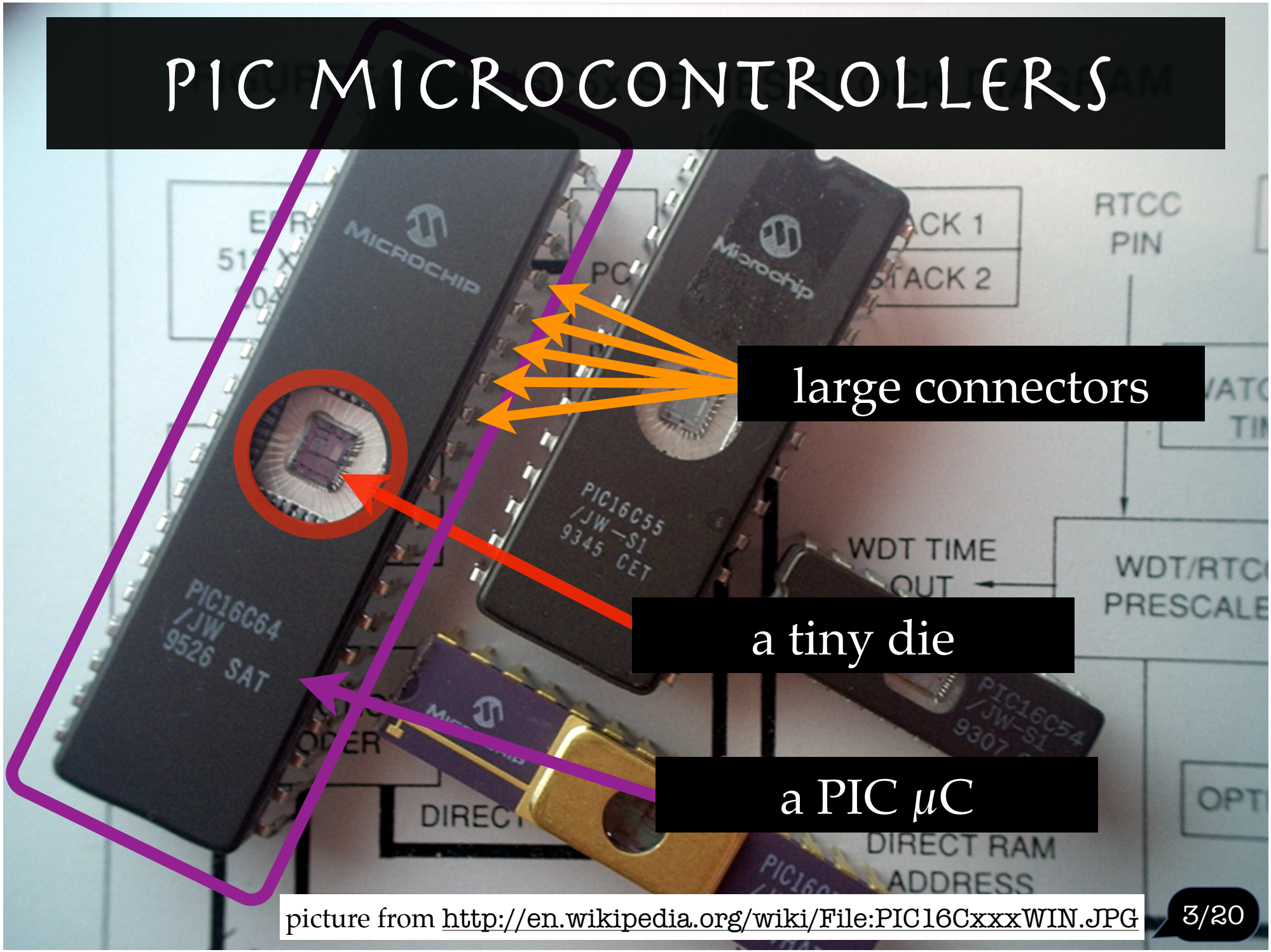
PIC I/O

```
external set_interruption_handler : (bit -> unit) -> unit =
  "caml_set_interruption_handler";;
(** Register an interruption handler. *)

external clear_interruption_handler : unit -> unit =
  "caml_set_interruption_handler";;
(** Remove the interruption handler. *)
```

interruptions

PIC MICROCONTROLLERS



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

OCAPIC 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. *)

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. *)
```

PIC I/O

```
external set_interruption_handler : (bit -> unit) -> unit =
  "caml_set_interruption_handler";;
(** Register an interruption handler. *)

external clear_interruption_handler : unit -> unit =
  "caml_set_interruption_handler";;
(** Remove the interruption handler. *)
```

interruptions

OCAPIC LIBRARY

PIC SPECIFIC LIBRARIES

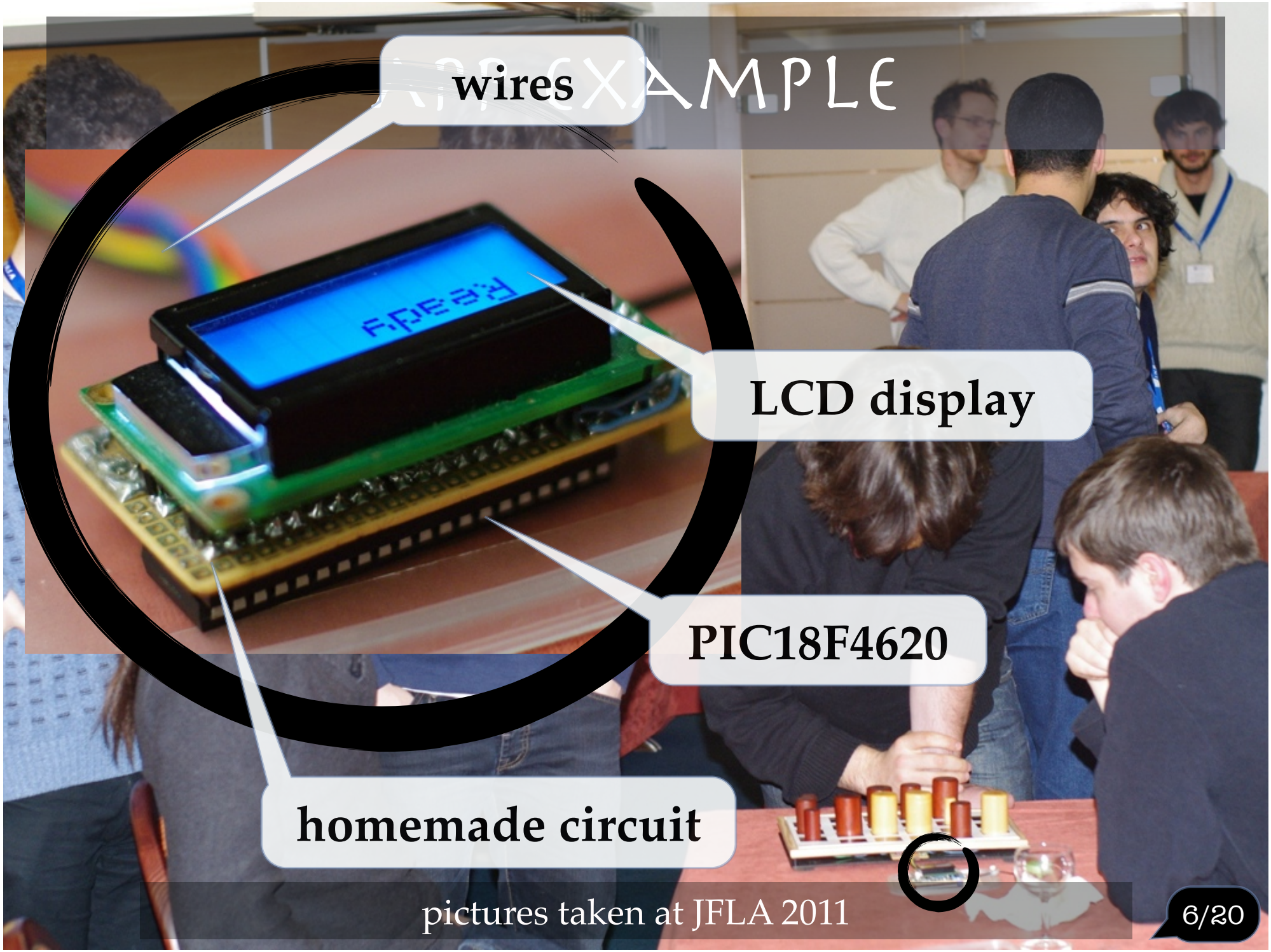
```
module MyDisplay = Lcd.Connect (
struct
  let bus_size = Lcd.Four (* databus mode *)
  let e  = Pic.LATD0      (* command connector *)
  let rs = Pic.LATD2
  let rw = Pic.LATD1
  let bus = Pic.PORTC     (* databus port *)
end);;
MyDisplay.init ();;
MyDisplay.config ();;
MyDisplay.print_string "Hello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

EXAMPLE



pictures taken at JFLA 2011

OCAPIC 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

flash memory occupation

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

CONCLUSION

- Full OCaml frontend & Good performance
  thanks to a virtual machine! (+ a set of tools)
- OCAPIC works with any PIC18 *(but check the memory size)*
- OCamlClean was necessary to allow the use of OOP
  (but not necessary for small programs without OOP)
- 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/

THANKS!

don't miss the demo session

this afternoon ;-)

ANY QUESTIONS?

picture taken at JFLA 2011