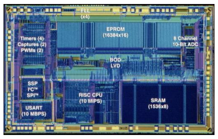
Running Scheme on a PIC microcontroller

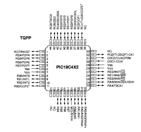
Marc Feeley (Université de Montréal) and Danny Dubé (Université Laval)

- Objective: create a Scheme system for PIC that is **R^{4}RS conformant** (except for file I/O)

- The PIC is an inexpensive **single-chip** general purpose computer with **little RAM**

Model	Pins	MIPS	ROM	RAM	Price
PIC12C508	8	1	512 × 12 bits	25 × 8 bits	$0.90
PIC16F628	18	5	2048 × 14 bits	224 × 8 bits	$2.00
PIC18F1320	18	10	4096 × 16 bits	256 × 8 bits	$3.25
PIC18F6520	64	10	16384 × 16 bits	2048 × 8 bits	$6.50
PIC18F8720	80	6.25	65536 × 16 bits	3840 × 8 bits	$11.03





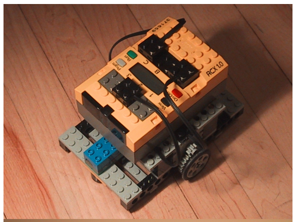
- Series of 3 different implementations that target different memory sizes:
  - BIT (1996): H8/3292 μcontr., 2.5 KB ≤ RAM ≤ 64 KB, 8.5 KB ≤ ROM
  - PICBIT (2003): PIC18F6720 μcontr., 0.25 KB ≤ RAM ≤ 3.5 KB, 22 KB ≤ ROM
  - PICOBIT (2003): PIC18F1320 μcontr., 0.1 KB ≤ RAM ≤ 0.5 KB, ≈4 KB ≤ ROM

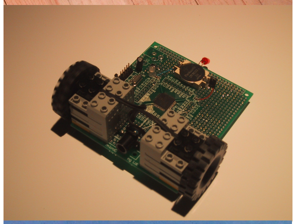
- All 3 systems have compilers that eliminate dead code through a **whole-program analysis** and the library is compact thanks to **higher-order functions**

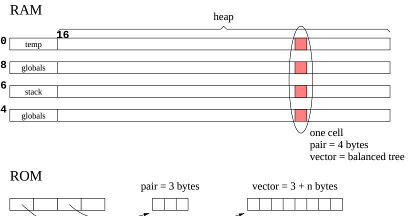
- **BIT**: was ported to the LEGO MINDSTORMS and Z8 Encore! platforms, rather **slow execution** (≈8000 byte-codes per second)
  VM: stack based, stack is a chain of heap cells (this causes high pressure on the GC)
  Memory management: **real-time GC**, mark-compact type, 16 bit words, smallest object = 8 bytes, stationary handles & movable bodies

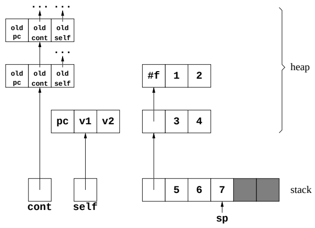
- **PICBIT**: targets high-end PICs, takes into account that RAM is the critical resource, **≈2 × faster than BIT**, requires little RAM but large ROM
  VM: **register based** (reduces allocation rate for continuations), safe-for-space, optimizing compiler
  Memory management: mark-sweep **blocking GC** (Deutsch-Schorr-Waite marking algo), 11 bit words, all objects = 3 bytes, 2 refs per object

- **PICOBIT**: ****work in progress****, targets mid-level PICs
  VM: written in assembler-level macro language, stack based (**stack cache** overflowing to the heap), not safe-for-space, simple compiler, **multi-threading** implemented on top of continuations
  Memory management: like PICBIT except: **8 bit words**, all objects = 4 bytes, **3 refs per object**, this takes **less RAM and ROM space**, RAM and ROM objects have **different representations**









⇒ Results from a simulator:

	24 bit cells		32 bit cells	
Program	Byte-code	RAM needed	Byte-code	RAM needed
`empty`	79	0	79	0
`traffic`	164	24	164	20
`photovore`	363	99	310	96
`(fib 20)`	113	294	113	228
`1-thread`	217	54	216	44
`10-thread`	253	84	252	84