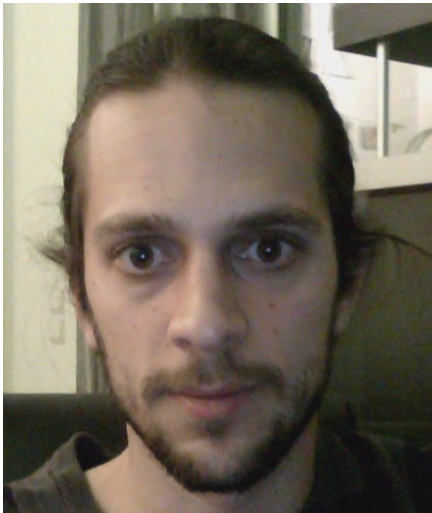
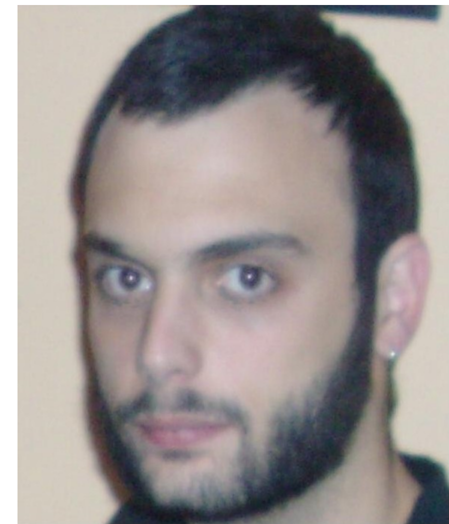
Kostis Sagonas



joint work with



Chris Stavrakakis and **Yiannis Tsiouris**

What is ErLLVM?

- A project aiming to provide multiple back-ends for the **High Performance Erlang (HiPE)** native code compiler of Erlang/OTP ...
- … using the **Low Level Virtual Machine (LLVM)** compiler infrastructure …
- … in order to *improve the performance* of Erlang applications …
- … and *ease the maintenance* of its native code compiler.

This talk

- Overview and design
  - HiPE native code compiler
  - LLVM compiler infrastructure
- Architecture and implementation of ErLLVM
  - LLVM extensions
  - New HiPE component
- Performance evaluation
  - vs. BEAM
  - vs. HiPE
  - vs. Erjang
- Current status and future work

HiPE: High Performance Erlang

- Project at Uppsala University started in 1997
- Developed the native code compiler for Erlang
  - Component of Erlang/OTP since 2001
- A *mature* compiler that is *robust* and produces *reasonably efficient* code
- Back ends for
  - SPARC V8+
  - x86 and x86_64 (AMD64)
  - PowerPC and PowerPC64
  - ARM

LLVM: Low Level Virtual Machine

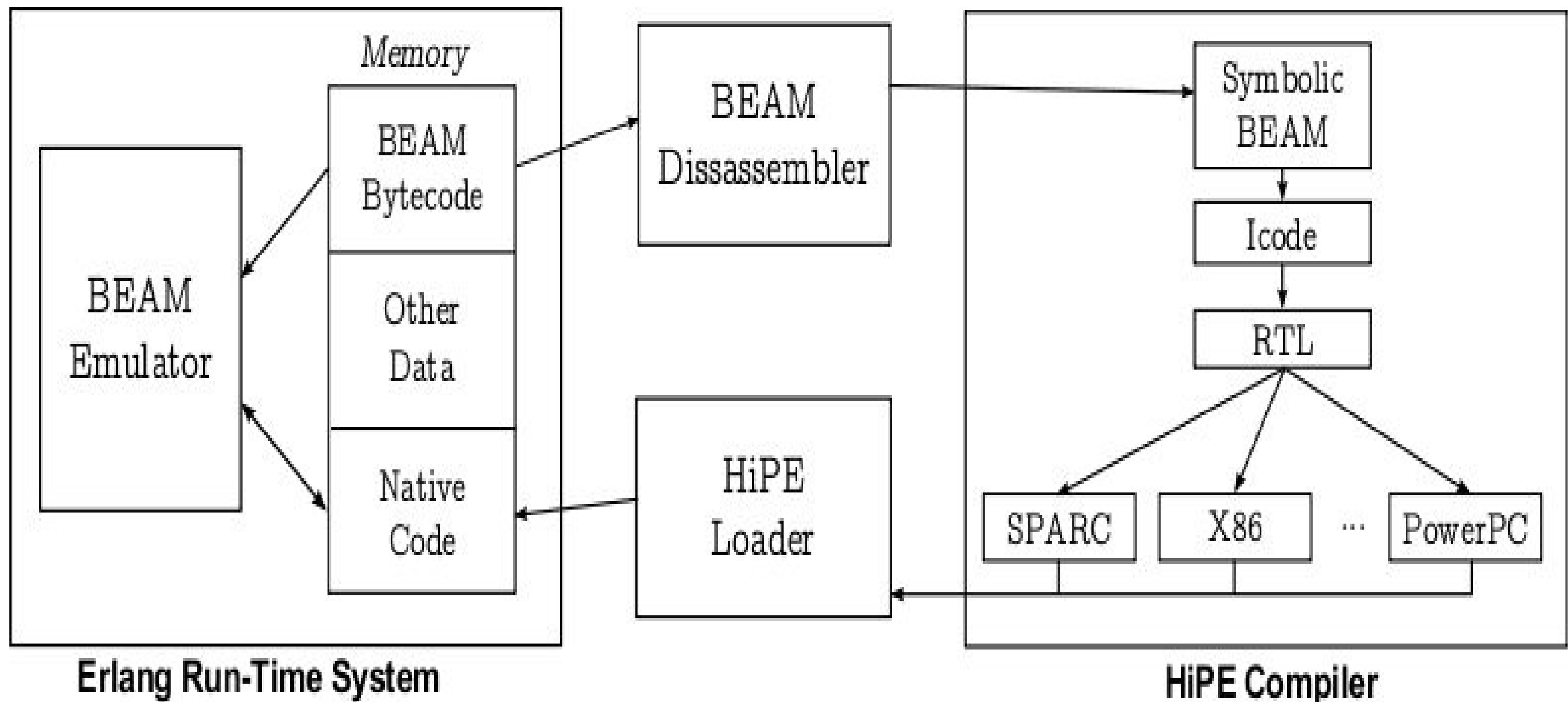
- Collection of industrial strength compiler technology
  - Language-independent *optimizer* and *code generator*
    - Many optimizations, many targets, generates good code
  - Clang C/C++/Objective-C front end
    - Designed for speed, reusability, compatibility with GCC
  - Debuggers, “binutils”, standard libraries
    - Providing pieces of low-level tools, with many advantages

- High-level portable LLVM assembly
  - RISC-like instruction set; static type system; SSA form
  - Three forms: human-readable, on-disk, in-memory

Why LLVM?

- Used as a static or JiT compiler and for static analysis
- State-of-the-art software with very active community of developers
- A new compiler = glue code + any components not yet available. Allows choice of the right components for the job, e.g. register allocator, scheduler, optimization order.
- Supports many architectures: x86, x86_64, ARM, PowerPC, SPARC, Alpha, MIPS, Blackfin, CellSPU, Mblaze, MSP430, XCore, …
- Open source with a *BSD-like License* and many contributors: industry, research groups, individuals

HiPE Architecture in Erlang/OTP



HiPE's Icode

```
1   lists:sum/2(v1, v2) ->
2   %% Info: ['Not a closure','Leaf function']
3   1:
4         v3 := phi({1, v1}, {3, v6})
5         v4 := phi({1, v2}, {3, v7})
6         _ := redtest() (primop)
7         goto 4
8   4:
9         if is_cons(v3) then 3 (0.50) else 6
10  3:
11        v5 := unsafe_hd(v3) (primop)
12        v6 := unsafe_tl(v3) (primop)
13        v7 := '+'(v5, v4) (primop)
14        goto 1
15  6:
16        if is_nil(v3) then 5 (0.50) else 2
17  5:
18        return(v4)
19  2:
20        v10 := function_clause
21        fail(error, [v10])
```

Figure 2. The Icode (in SSA form) of a `lists:sum/2` function

```
sum([H|T],A) -> sum(T,H+A);
sum([], A) -> A.
```

HiPE's RTL

```
1  {lists,sum,2}(v19, v20) ->
2  ;; Leaf function
3  ;; Info: []
4  .DataSegment
5  .CodeSegment
6  L1:
7      v21 <- v20
8      v22 <- v19
9      goto L2
10 L2:
11     v23 <- phi({1, v21}, {11, v32})
12     v24 <- phi({1, v22}, {11, v33})
13     %fcalls <- %fcalls sub 1 if lt then L4 (0.01) else L6
14 L4:
15      <- suspend_0() then L6
16 L6:
17     r25 <- v24 'and' 2 if eq then L7 (0.50) else L8
18 L7:
19     v26 <- [v24+-1]
20     v27 <- [v24+7]
21     r28 <- v26 'and' v23
22     r29 <- r28 'and' 15
23     if (r29 eq 15) then L13 (0.99) else L12
24 L13:
25     r34 <- v23 sub 15
26     v35 <- v26 add r34 if not_overflow
27                       then L11 (0.99) else L12
28 L11:
29     v31 <- phi({13, v35}, {12, v30})
30     v32 <- v31
31     v33 <- v27
32     goto L2
33 L12:
34     v30 <- '+'(v26, v23) then L11
35 L8:
36     if (v24 eq -5) then L15 (0.50) else L16
37 L15:
38     return(v23)
39 L16:
40     v36 <- atom_no('function_clause')
41      <- erlang:error(v36)
42     return(15)
```

Figure 3. RTL code (in SSA) produced by the Icode of Figure 2

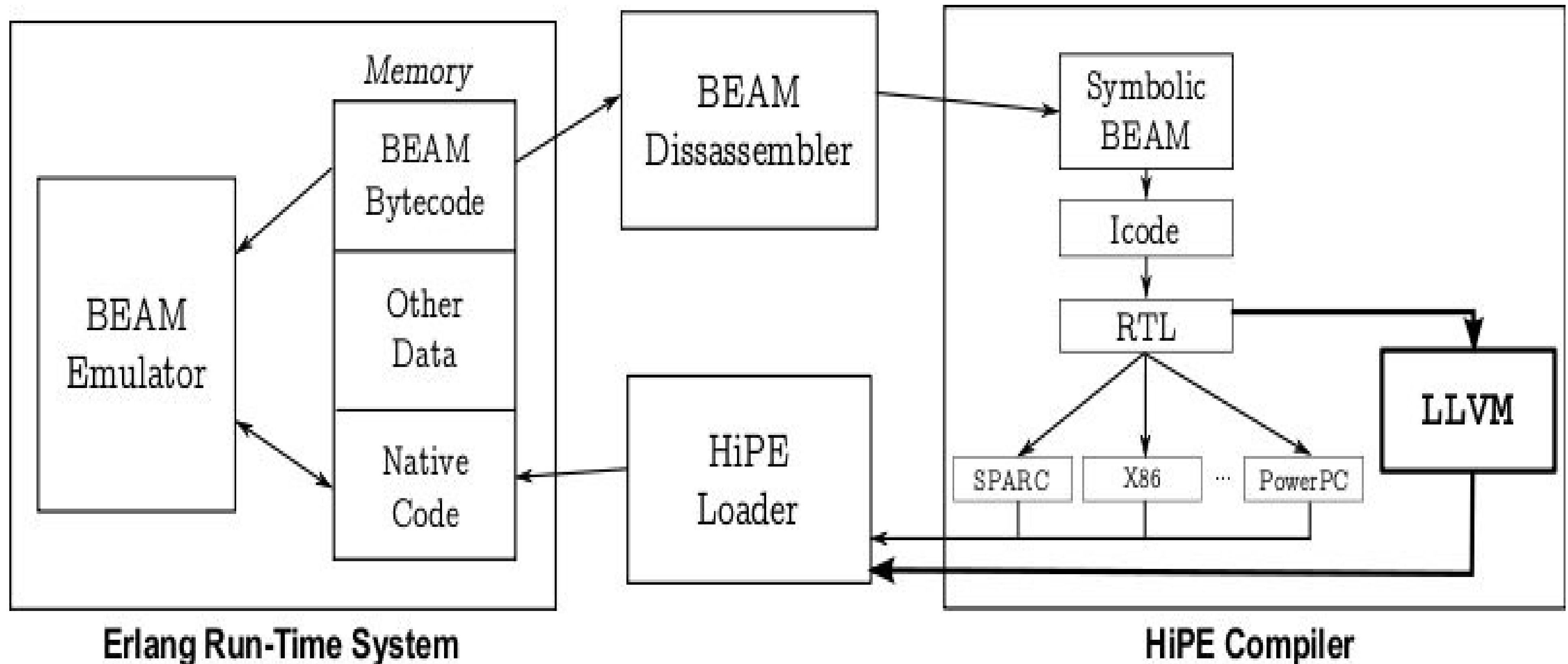
Structure of HiPE's Back-ends

- Register allocation
  - Many choices; default iterated register coalescing
- Frame management
  - Check for stack overflow
  - Set up frame
  - Create stack descriptors
  - Add “special” code for tailcalls
- Code linearization
- Assembly

Why use LLVM as a Back-end?

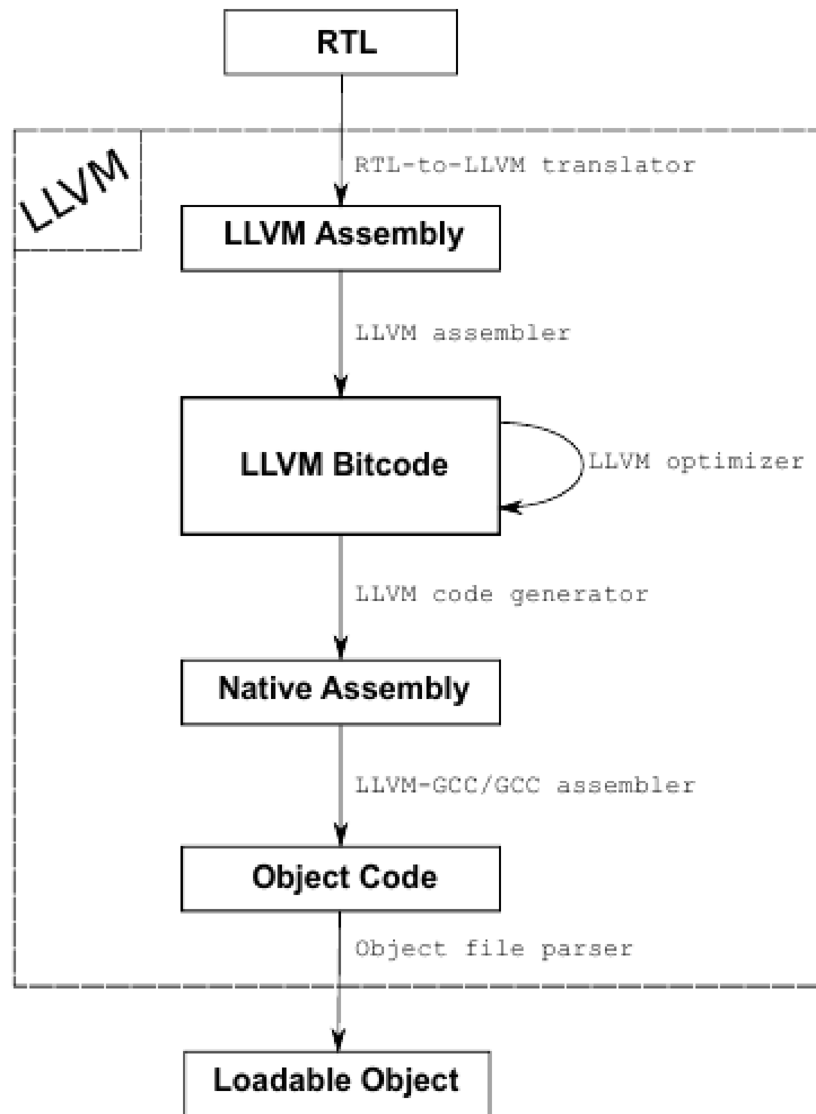
- Curiosity: perform a research experiment
- Easier maintenance of existing back-ends
  - One instead of six
  - Small-sized, straightforward code
  - Outsource implementation and further optimization
- Get more back-ends “for free” (well, almost...)
- Possibly improve performance
  - Outsource target-related optimizations

HiPE Architecture in ErLLVM



- Use existing HiPE Loader and ERTS support
  - Be ABI compatible!

The LLVM Component



hipe_rtl2llvm	Create human-readable LLVM assembly (.ll)
llvm-as	Human-readable assembly (.ll) → LLVM bitcode (.bc)
opt	Optimization *Passes*, supports standard groups (-O1, -O2, -O3) (.bc → .bc)
llc	Bitcode (.bc) → Native assembly (.s), impose rules about memory model, stack alignment, etc.
llvm-gcc	Create object file (.s → .o)
elf64_format	Extract executable code and relocations

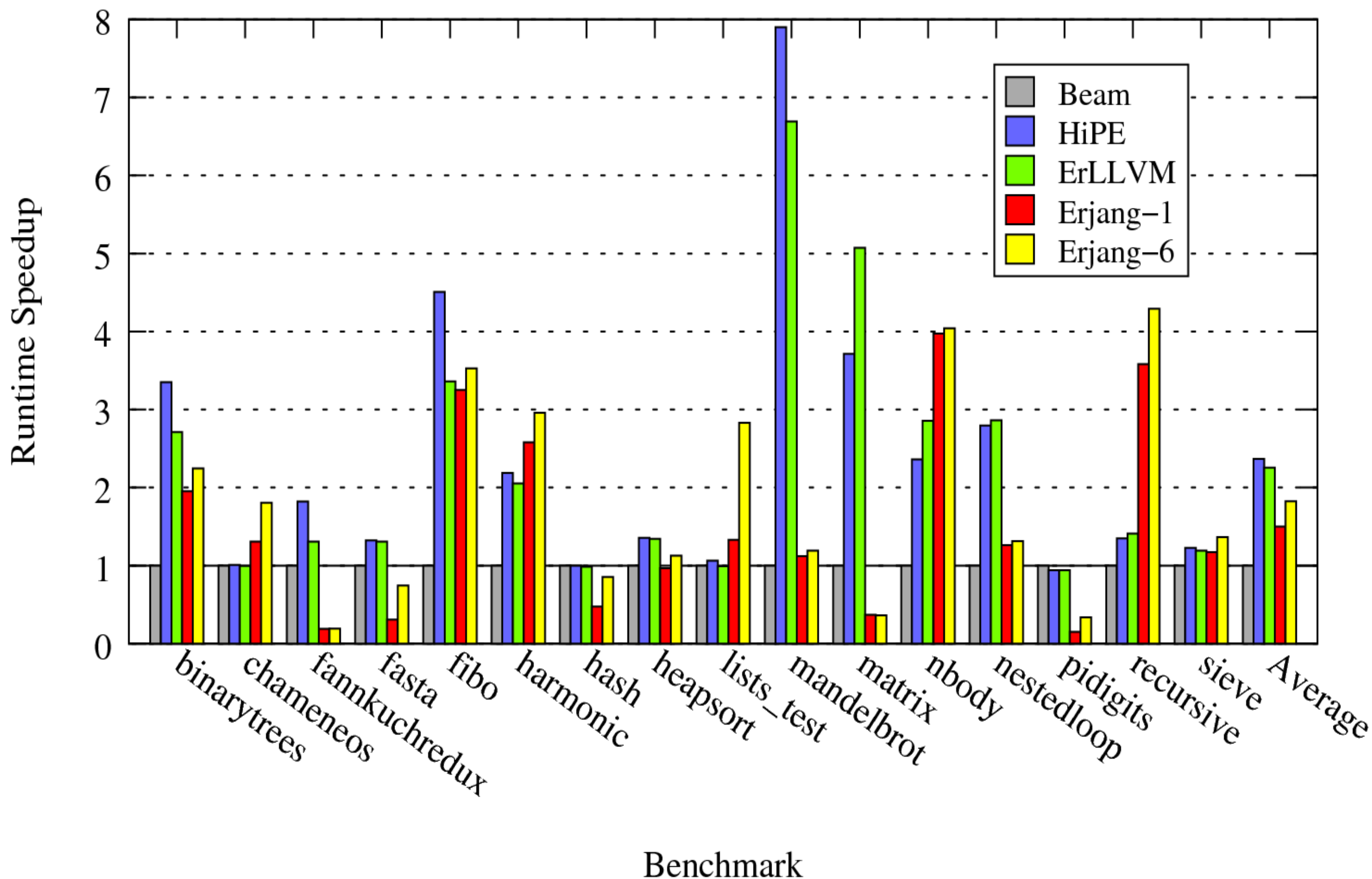
Subtle Points of Using LLVM

- Calling convention
  - VM “special” registers, args and return values
  - callee-/caller-save registers, callee pops args
- Explicit frame management
  - In-lined code for stack overflow checks in assembly prologue
- Stack descriptors
  - Exception handling
  - Precise garbage collection

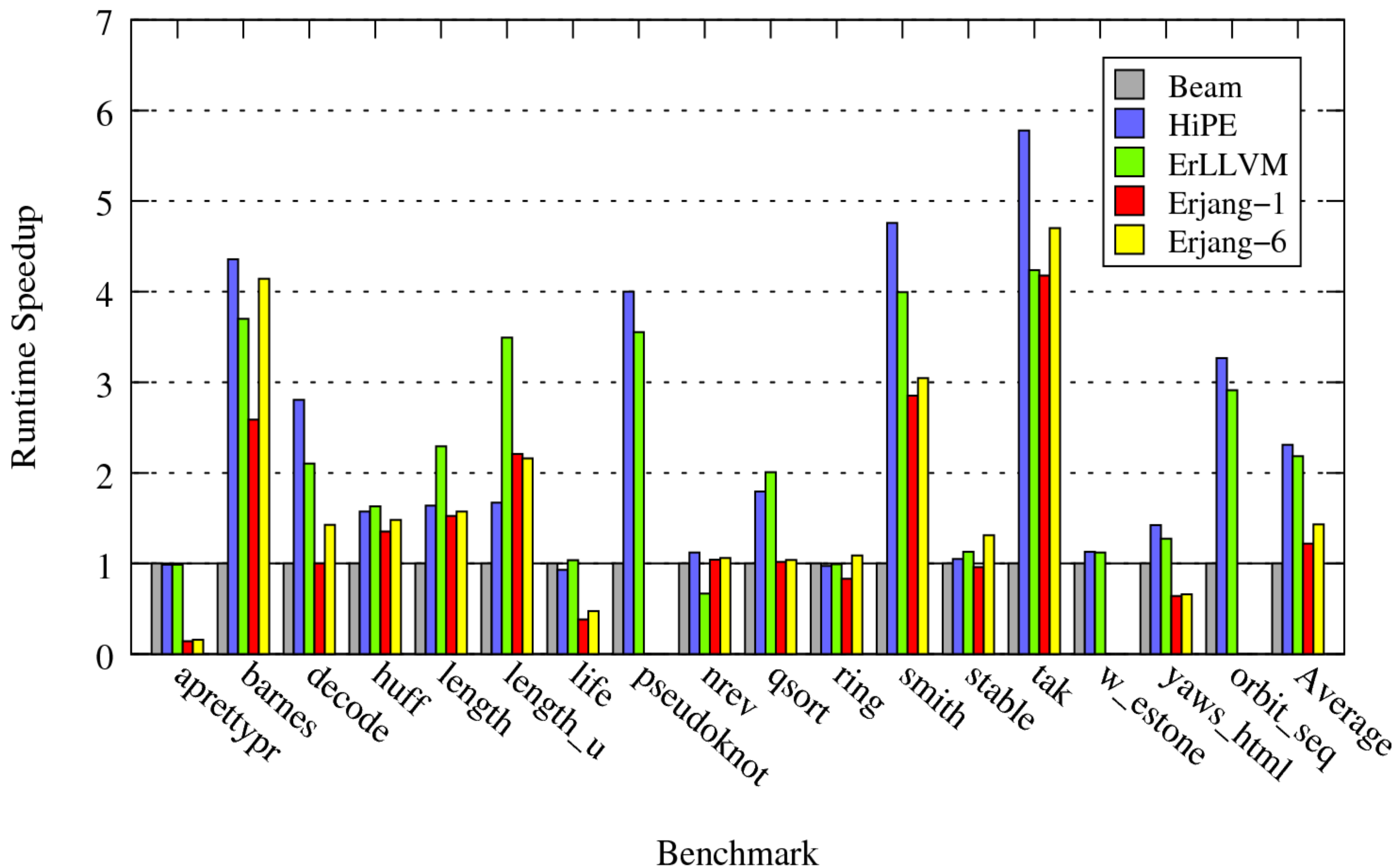
Current Status of ErLLVM

- Patches to LLVM
  - Custom calling convention & register pinning
    - (Already part of LLVM's code base!)
  - GC plugin to write GC information in object file
    - Use **`elf_format`** to parse .o file and extract the info
  - Function pass to emit custom prologue
- New HiPE component on top of R15B "maint"
  - Support for x86 and x86_64
  - Support for *accurate GC:* mark stack slots not live when variables that "inhabit" them are no longer live
  - About 5000 LOC
- Very robust and ready to use in production!

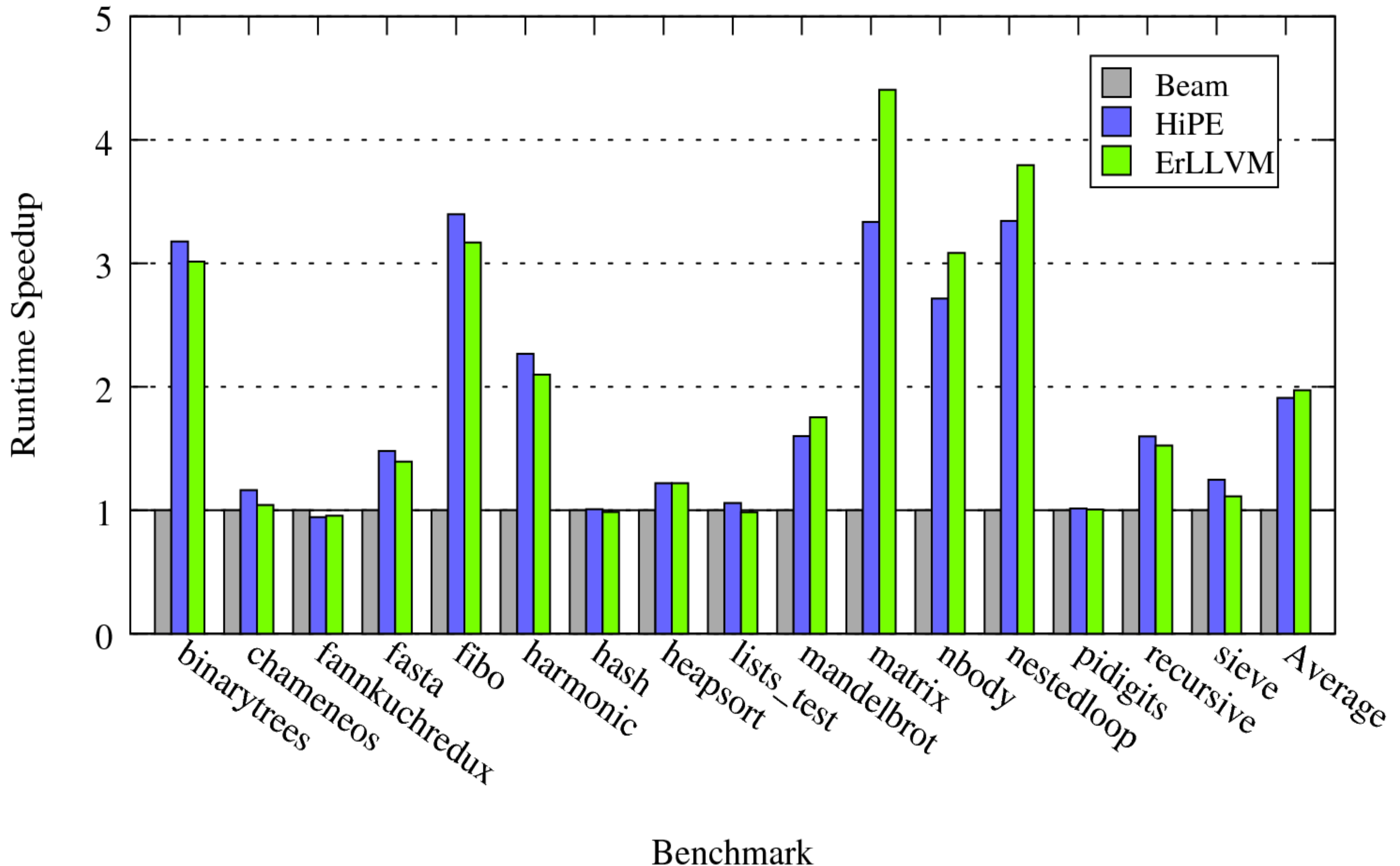
Performance on x86_64 (Beam=1)



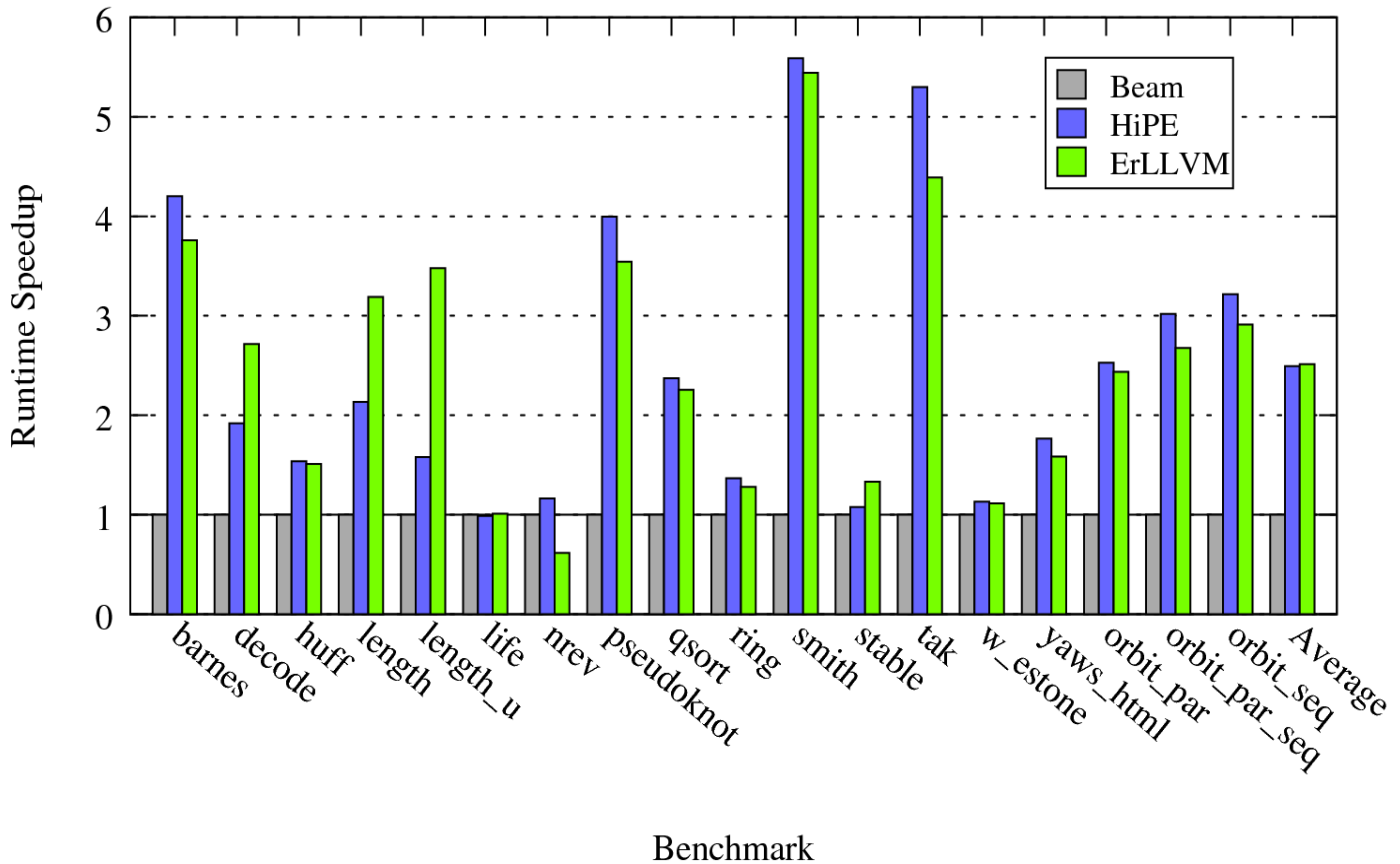
Performance on x86_64 (Beam=1)



Performance on x86 (Beam=1)



Performance on x86 (Beam=1)



Now what?

Demo time!

Current Status: Pros

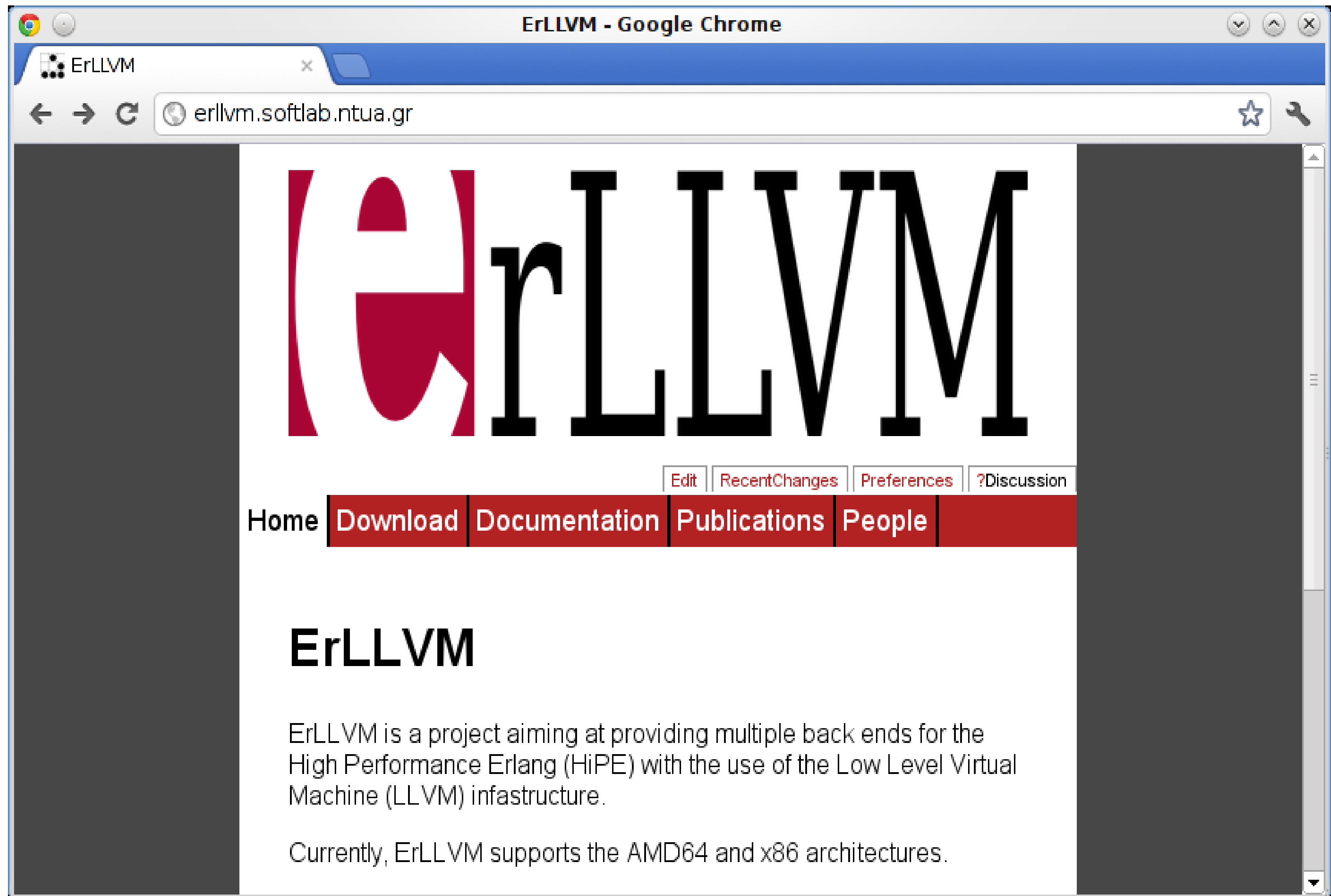
- Complete & robust: handles all Erlang programs
- ABI compatible:
  - smooth integration with BEAM and HiPE code
- Performance:
  - much better than BEAM
  - almost as good as HiPE
- Smaller and simpler code base for the back-ends
- Possibility to target more architectures
- LLVM back-end improvements now also improve performance of Erlang applications!

Current Status: Cons & Future Work

- Cons:
  - Need to download and install custom LLVM
  - Slightly longer compilation times
    - Not a problem in practice
    - Erlang LLVM bindings to the rescue??

- Future Work
  - Finish pushing LLVM patches upstream
  - Port to ARM
  - Improve GC support

Where can I find ErLLVM?



Users are now welcome!

- Install following the instructions at:

  `http://erllvm.softlab.ntua.gr`

- Grab code from `github`
- Test and measure!
- Report experiences
- Contribute to the project