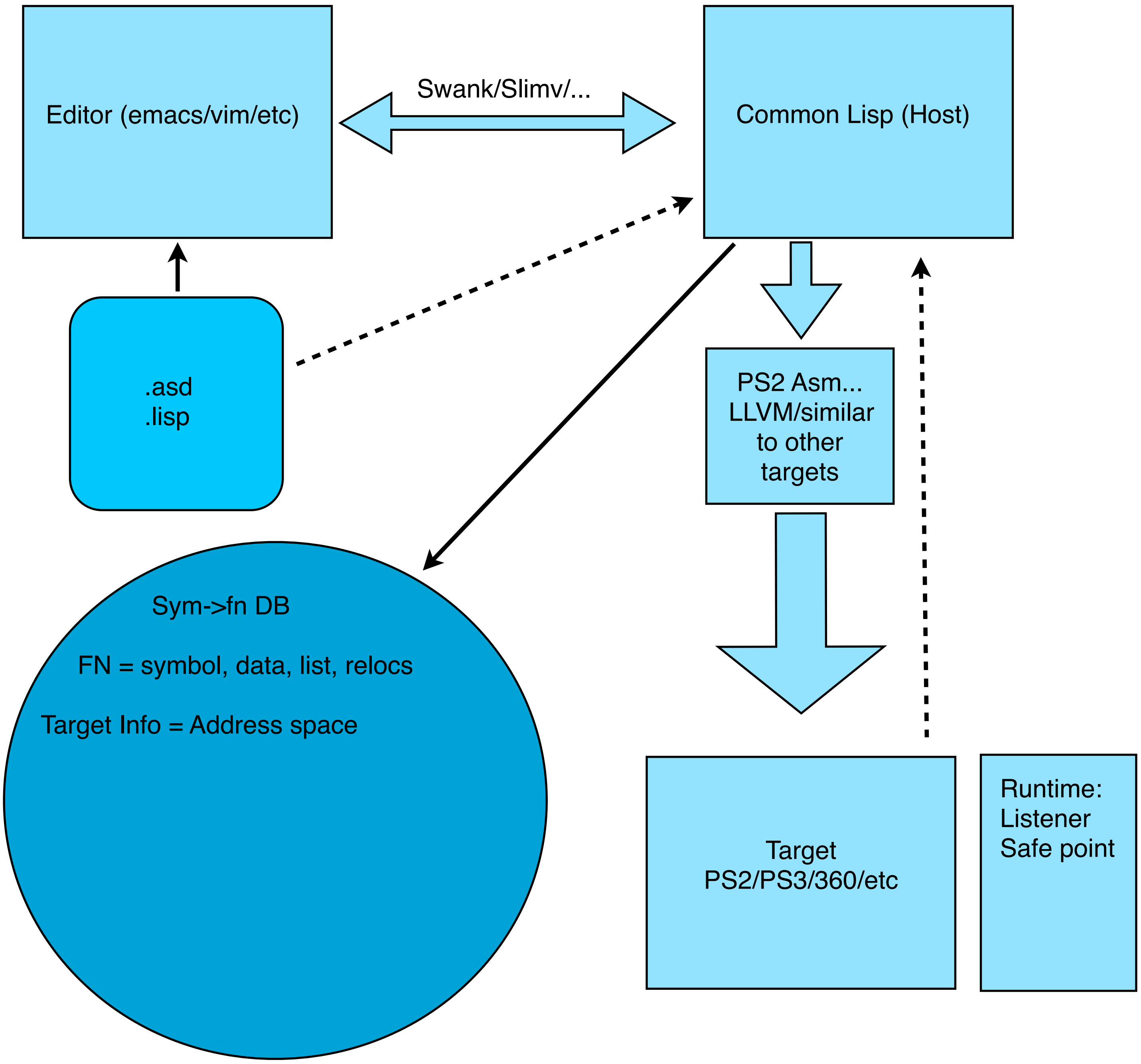
Overview of Score!

These are mostly ramblings and various ideas that me and Andreas Fredriksson has come up with during a brain storm session about Score!

High level flow-chart



So the way it will work (as of now) is

1. Score core is loaded into the Lisp.
2. Score code is actual .lisp files, but uses other constructs to define functions. For example

```
(def-ee-fun my-fun () ... )
```

The code that implements the def-ee-fun will assemble the code into native code for the target (it would also be possible to "run the assembler" code as lisp code if implemented)

3. Once the assemble is done it will be inserted into a global function table. The code is not sent directly to the target but needs to be done separately (this can also be chained so a function can be updated directly on the target)

4. The target will know very little about the world. The host will feed it with all information it need, the only exception is if you run a function and want to have some output (text) back to the host.

5. The host will do all allocation, gc, etc of code (data is a later issue) and the target will only be given a simple set of commands (write data, set pc, and may be a bit more)

6. The target needs (as of now at least) to implement a safe point where code can be updated, this way it makes it easier to update the code as things could get messy if we are running something at the same time as we try to update it.

Ideas of the assembler. As of now only focus on PS2 target. Ideas is:

Loop macros:

```
(dotimes (i 100)
   (add ...)
   (sub ...))
```

Automatic register allocator and explicit (with also ability to pick register)

```
(rlet ((foo :r7))
   ... )
```

This should be legal (automatic register allocation)

```
(let ((... ))
  (add test, r1, r2)
  (mul r2, test, r1)
```

Assign type to register, makes it possible to use generic construction on types

```
(let ((foo :u32 bar :u16))
  (inc foo)
  (dec bar))
```

High level constructs on low level assembler.

Register allocation, Form expand (macros)

Implement pseudo mips instructions (load 32-bit constant, etc)