

# Bang bang

or

Conceptual integrity and the meaning of the !!  
operator

Joe Armstrong

[joe@sics.se](mailto:joe@sics.se)

# Conceptual integrity

Everything is a process

!!

A !! B is an infix Remote Procedure Call operator

A can be a Pid, or a string or a list of Pids and strings

If A is a Pid then A !! B is short for:

```
A !! {self(), B},  
receive  
  {A, Reply} ->  
    Reply  
end
```

# Parallel RPC's

$[A_1, A_2, \dots, A_n] !! B$  returns  $[V_1, V_2, \dots, V_n]$  where

$V_1 = A_1 !! B,$

$V_2 = A_2 !! B,$

...

$V_n = A_n !! B$

and all the calls are done in parallel

# Files are processes

If F is a file

```
F !! read      reads the file,  
F !! {write, Bin}  writes to the file.
```

Read a file and copy to new location

```
{ok, Bin} = "/home/joe/foo" !! read,  
"/home/joe/bar" !! {write, Bin}
```

Read three files in parallel

```
[A,B,C] =
```

```
["/home/joe/foo", "http://www.erlang.org/a/b",  
 "ftp://bing.bong/pub/abc"] !! read
```

# Files are processes

Backup a local file on a remote machine

```
L = ["/home/joe/foo",  
      Remote="ftp://www.sics.se/joe/backup/foo"],  
case L !! read of  
  {X, X} -> true;  
  {{ok,Bin},_} -> Remote !! {write, Bin}  
end
```

# Distribution and !!

"erl://Node/Name !! X " means do Name !! X on Node and return the result.

Read a local file

```
"/home/joe/foo !! read"
```

Read a remote file

```
"erl://www.sics.se/home/joe/foo" !! read
```

# spawn and !!

"/dev/spawn !! Fun" is the same as spawn(Fun)

Create a local process

```
"/dev/spawn" !! fun() -> ... end
```

Create a remote process

```
"erl://www.sics.se/dev/spawn" !! fun() -> ... end
```

Note: We have replaced a BIF by a process.

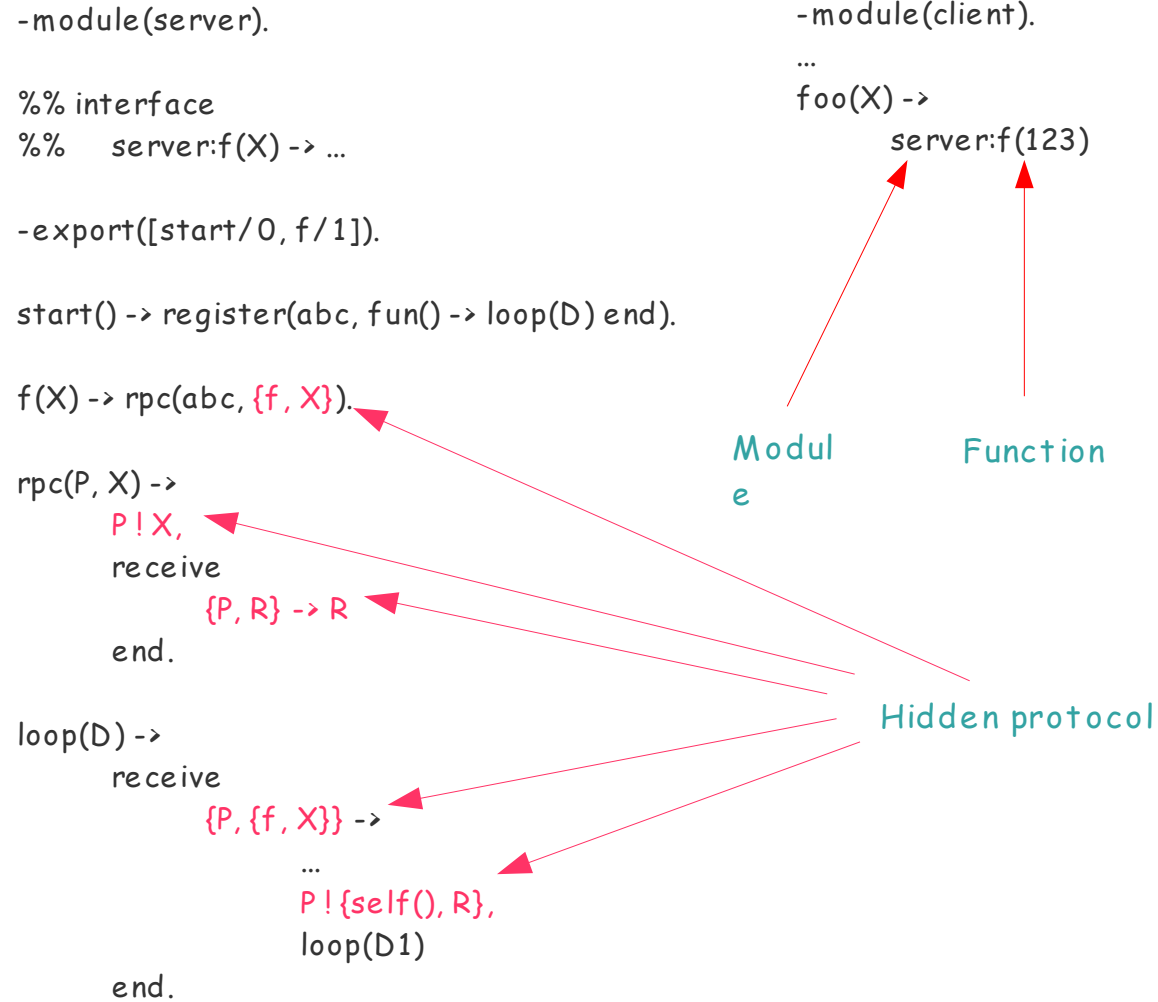
```
"/dev/spawn"
```

is a magic process - if you send it a fun it sends you back a Pid.

# Process names

"/dev/spawn"	process spawner
"/dev/code"	code server
"http://..."	read only files
"/home/..."	files
"ftp://..."	read-write files
"erl://Node/..."	remove nodes
"/proc/453612678"	local processes
"/dev/videoplayer"	video player
...	

# Information hiding



# Information revealing

```
-module(server).
```

```
-export([start/0, f/1]).
```

```
%% interface
```

```
%% "/dev/abc" !! {f, X} => ...
```

```
start() -> register("/dev/abc", fun() -> loop(D) end).
```

```
loop(D) ->
```

```
receive
```

```
{P, {f, X}} ->
```

```
...
```

```
...
```

```
P!{self(), R},  
loop(D1)
```

```
end.
```

```
-module(client).
```

```
...
```

```
foo(X) ->
```

```
"/dev/abc" !! {f, 123}
```

Process

Message

Revealed protocol

# Recap

Shorter code - Never a bad thing

We can often see the messages (and if they are remote or local) -  
Important for efficiency

Increases awareness of processes

Easier to understand

Many of the BIFs vanish

Encourages us to think in terms of protocol machines and NOT  
functions

# What's next?

~~How do we program?~~

Solved

How to we make systems?

How do we store stuff?

How to we find stuff?

# How do we make systems?



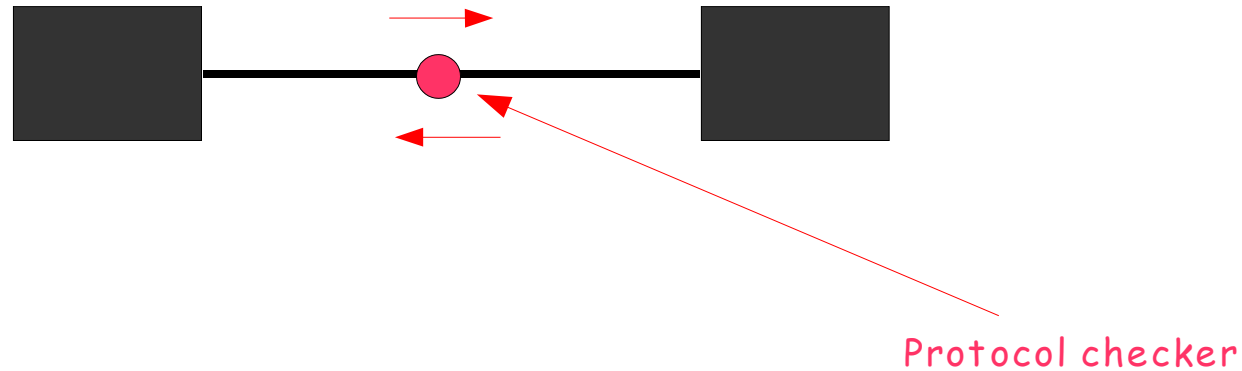
Systems are made of black boxes (components)

Black boxes execute concurrently

Black boxes communicate

How the black box works internally is irrelevant

# How do we fit the bits together?



Systems are made of black boxes (components)

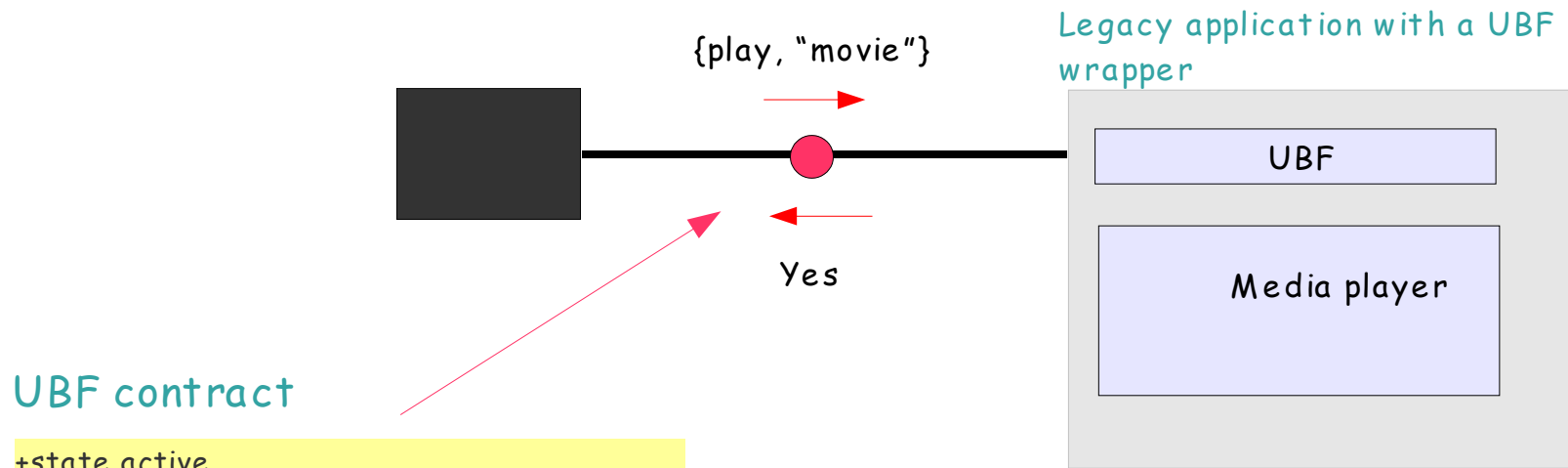
Black boxes execute concurrently

Black boxes communicate with defined (universal) protocols

The protocol is checked externally

How the black box works internally is irrelevant

# Putting things together



## UBF contract

```
+state active
  {play, string()} => yes x playing;
                    |   no x active
```

## Erlang with !!

```
"/dev/videoplayer" !!
{play, "movie"}
```

Now you see why I wanted !!

There is no `videoplayer:play(Movie)`

I want to expose the interface to the programmer

# What's left to do?

Write loads of wrappers for legacy applications

Implement UBF in all languages known to man

Increase the property of "isolation" in Erlang

Make decent OS's that obey principle of isolation

Add !! to core Erlang

Rewrite kernel libraries

Figure out how to store things

Figure out how to find things

- end -