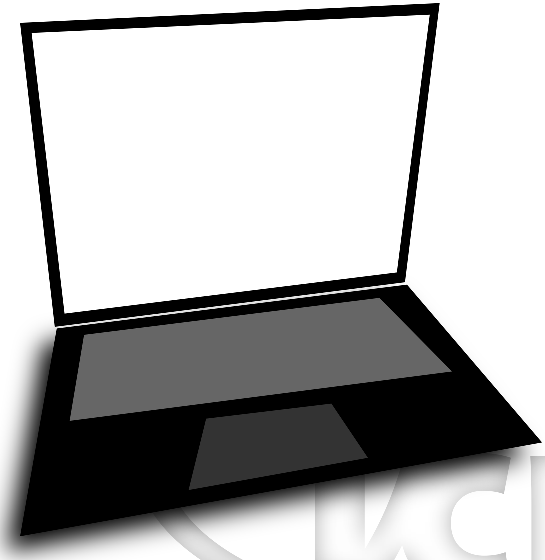
RPC w R

Miron Kursa

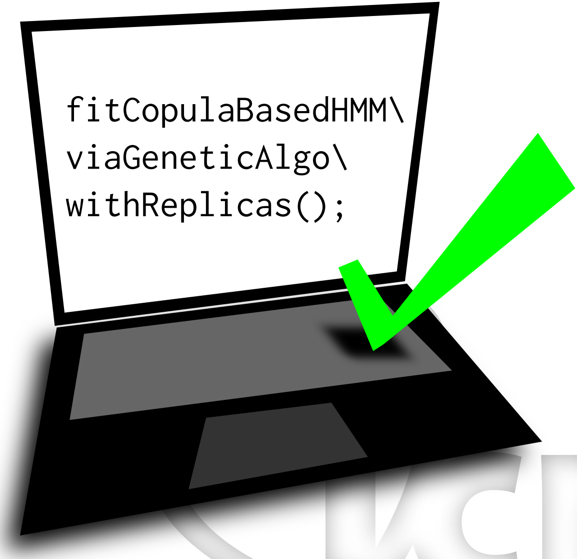
ICM UW

1 III 2012

Remote Procedure Call

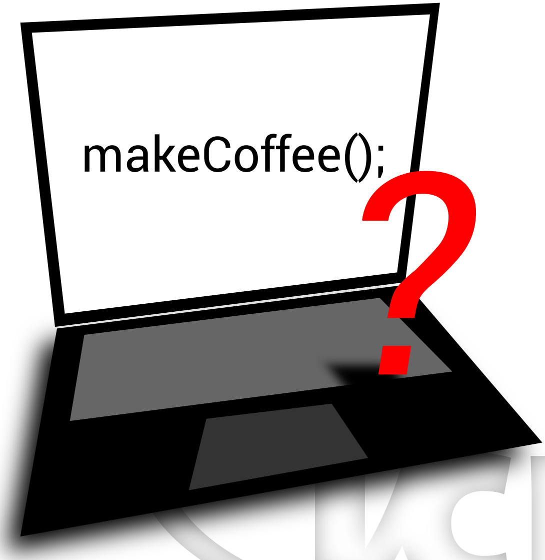


Remote Procedure Call



```
fitCopulaBasedHMM\
viaGeneticAlgo\
withReplicas();
```

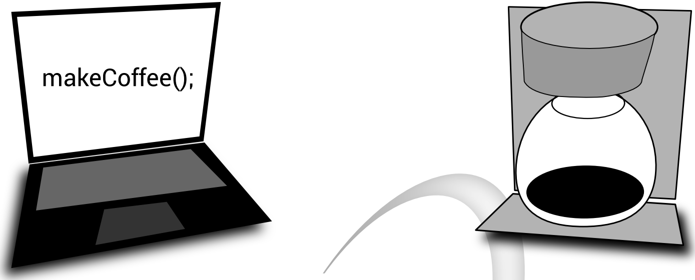
Remote Procedure Call



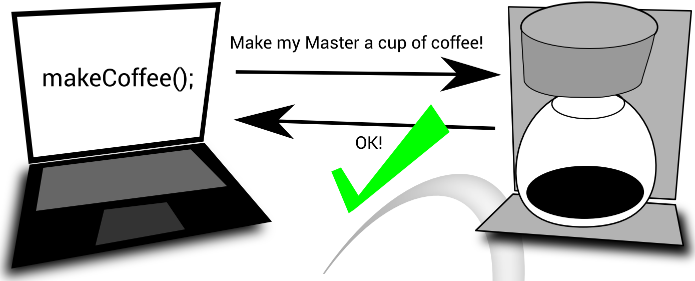
makeCoffee();

?

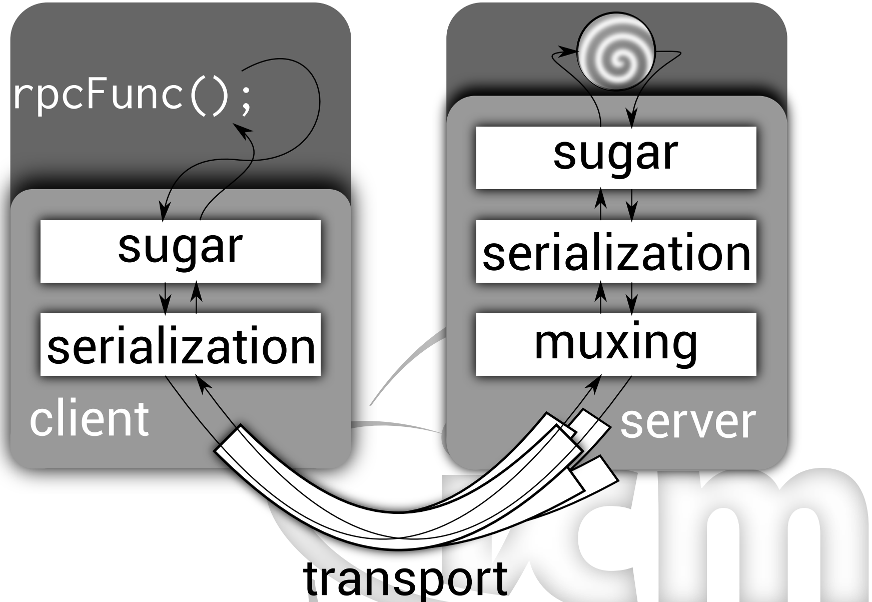
Remote Procedure Call



Remote Procedure Call



RPC architecture



Problems

- Muxing
  - R is strictly thread-dangerous.
  - R network capabilities are mostly limited to client use.
  - Interpreter wants to be in REPL loop.
- Serialization
  - Marshalling of R objects is very slow...
  - ...and is restricted to ASCII if *in* RAM*io*.
  - Fun fact is that it is rarely needed.

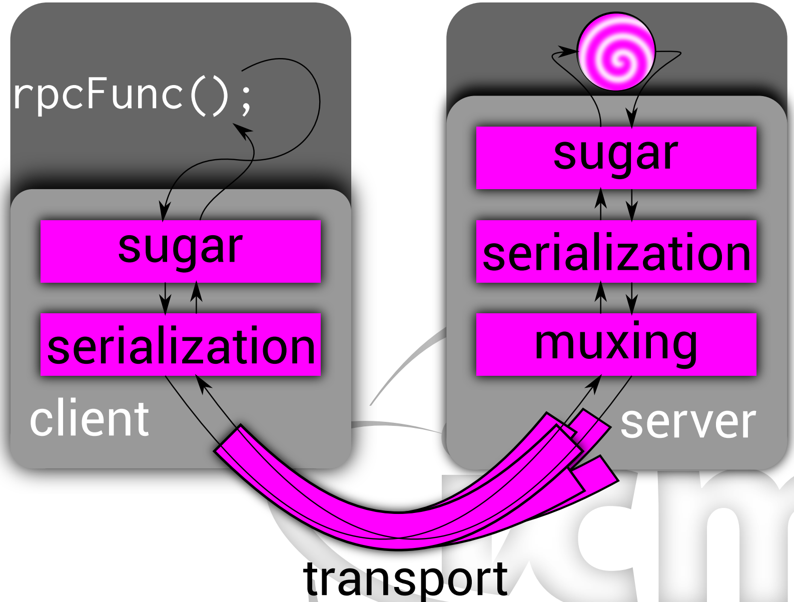
Solutions (servers!)

- Pipe (**rjson** package)
- Fork+pipes (**multicore**, **Rserve**)
- Prefork interpreters (**rApache**)
- Internal web server (**Rook**)
- CORBA, COM, etc. (omegahat, **rcom**)
- Smuggle real event loop into interpreter (**triggr**)

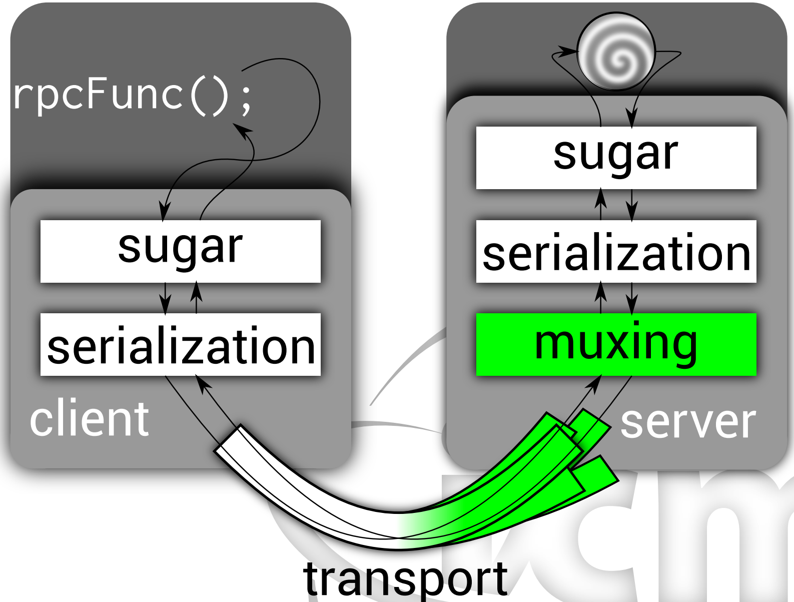
Rserve

- **Rserve** forks R session for each connection.
- User sends data and code via specific client (provided for C, Java, R and PHP).
- **Rserve** simply executes everything sent in.
- Connections exist in isolated environments which are wiped out after disconnect.
- Forking R is no fun because of the garbage collector.

Rserve


rpcFunc();
sugar
serialization
client
sugar
serialization
muxing
server
transport

triggr


rpcFunc();
sugar
serialization
client
sugar
serialization
muxing
server
transport

triggr

- User gives port and callback.
- **triggr** waits on a given port for messages defined as chunks of data separated by an empty lines (i.e. RNRNs).
- Callback is called on a chunk and its result is sent to client in the same format; callback can also terminate connection.
- When there are multiple messages as one time, **triggr** queues them in order of coming in.
- Each message processing is only isolated in an environment; connections *may* be tracked by connection ID.
- RNRN separation allows one to mimic HTTP traffic and use already done proxies and load balancers.

Live presentation

- Echo server
- Random number generator
- HTTP server
- ‘Real’ RPC (i.e. how the sugar part can be done in R)

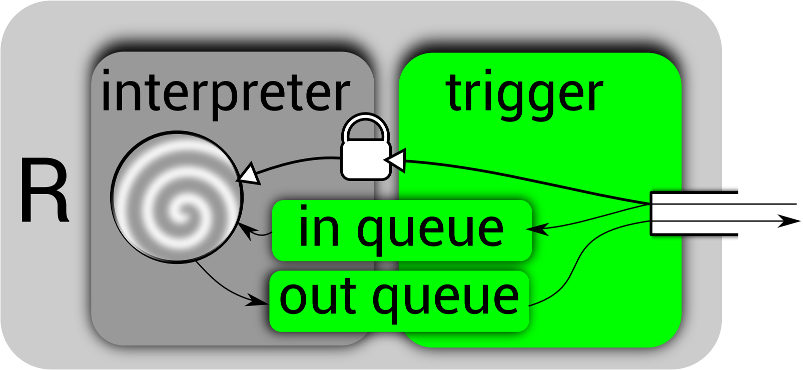
The end

`https://github.com/mbq/Triggr`

(can't be on CRAN because Windows[1] has real threads, OSX is broken and I'm too stupid to install gcc on Solaris since Oracle deopened it)

[1] Windows port is quite in reach, though.

Bonus for nerds: triggr internals


interpreter
trigger
R
in queue
out queue