```
:- module rpc_client.

new(Host, Port, RPC) :- true |
  klicio:klicio(S),
  S = [connect(inet(Host, Port), R)]
  new1(RPC, R).
new1(RPC, R) :- R = normal(Sock) |
  rpc(RPC, Sock).

rpc(RPC, Sock) :- RPC=[M:Call|RPC1] |
  io:put(Sock, Sock1, M:Call, Res),
  RPC = [Res | RPC1],
  rpc(RPC1, Sock1).
```

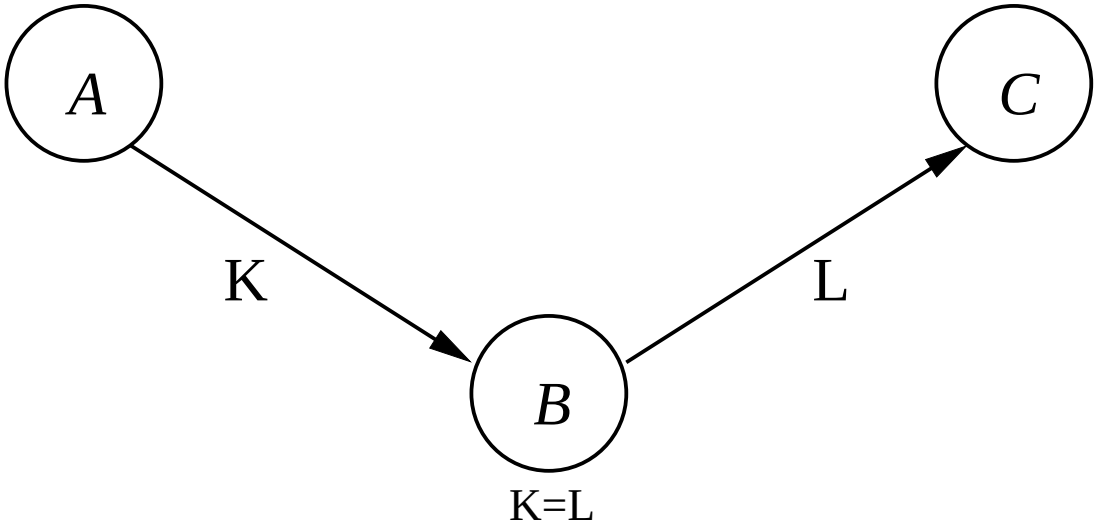
```
:- module io.

put(Sock,Sock_,Put,Get) :- wait(Put) |
  Sock = [putt(Put),
    fwrite("\n."), fflush(F) | Sock1],
  get(Sock1, Sock_, F, Get).

get(Sock, Sock_, F, Get) :- wait(F) |
  Sock = [gett(Get) | Sock_].
```

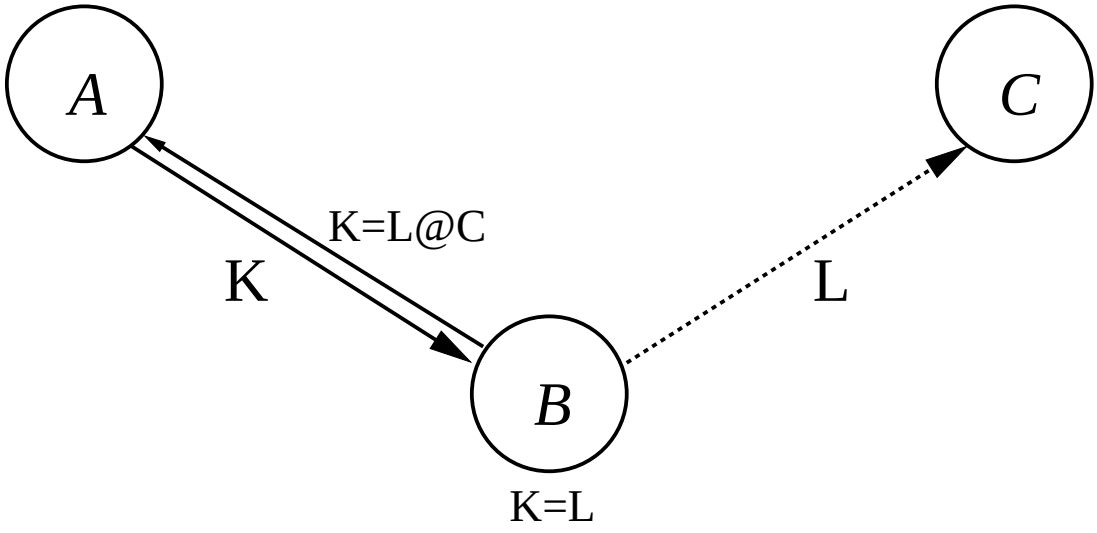
Remote Predicate Call

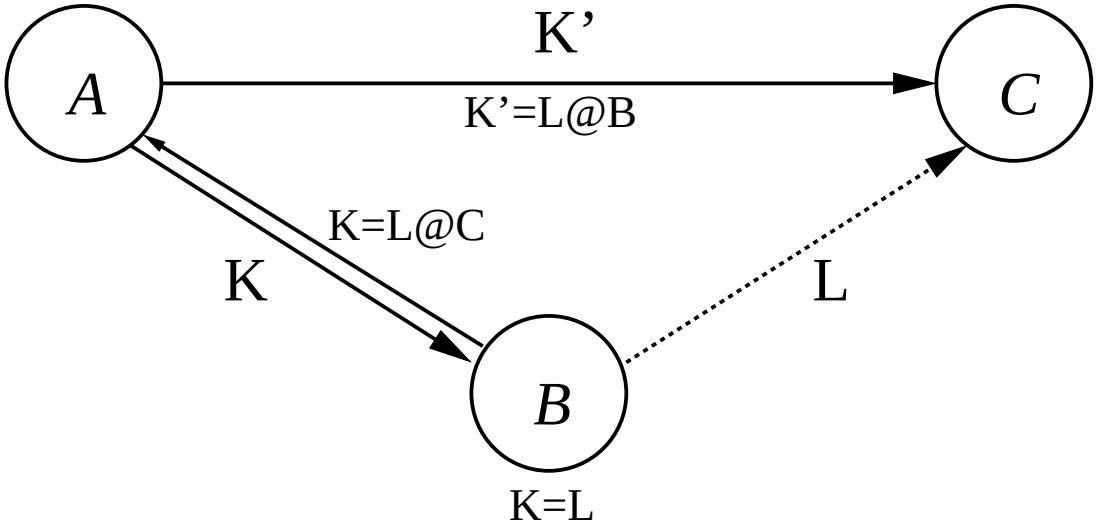
```
:- module rpc_server.

args_vector(F,Arity,V,L) :- Arity=:=0 |
  new_vector(V, L).
args_vector(F,Arity,V,L) :- Arity > 0 |
  arg(Arity, F, Arg),
  L1 = [Arg | L],
  Arity1 := Arity - 1,
  args_vector(F, Arity1, V, L1).

new(Port) :- true |
  klicio:klicio([bind(inet(Port),R)]),
  new1(R).
new1(R) :- R = normal(ServSock) |
  accept(ServSock).

accept(ServSock) :- true |
  ServSock = [accept(R) | ServSock1],
  accept1(ServSock1, R).
accept1(ServSock,R) :- R=normal(Sock) |
  io:get(Sock, Sock1, 0, Get),
  rpcd(Sock, Get),
  accept(ServSock).

rpcd(Sock, Get) :- Get = (M:Call) |
  functor(Call, Name, Arity),
  Arity1 := Arity + 1,
  generic:new(module, Modu, M),
  generic:new(predicate,
    Pred, Modu, Name, Arity1),
  args_vector(Call, Arity, V, [Res]),
  apply(Pred, V),
  io:put(Sock, Sock1, Res, Get),
  rpcd(Sock1, Get).
```

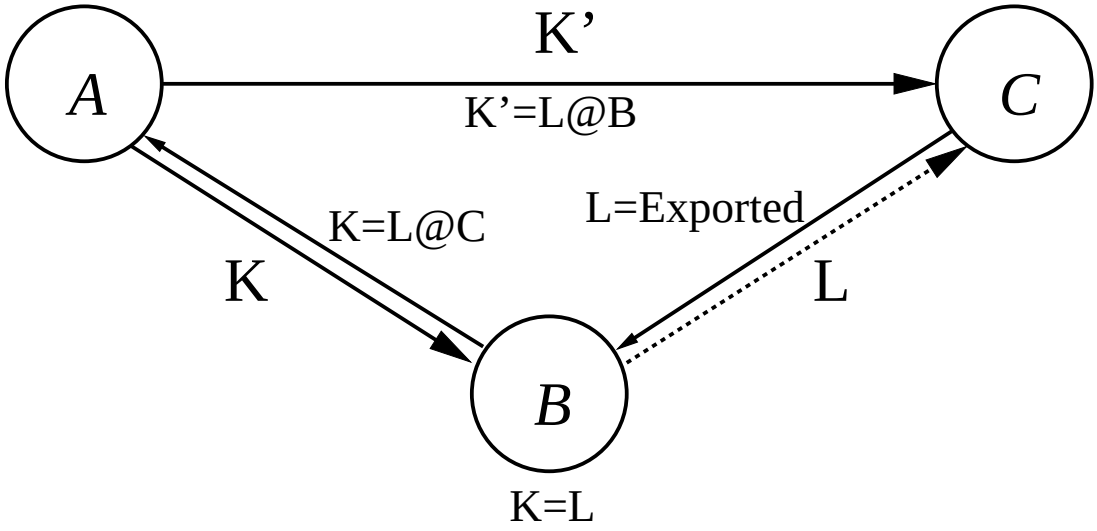
```
:- module rpc_client.

new(Host, Port, RPC) :- true |
  dklicio:connect(Host, Port, R),
  new1(RPC, R).
new1(RPC, R) :- R = normal(Sock) |
  RPC = Sock.
```

```
:- module rpc_server.

args_vector(F,Arity,V,L) :- Arity=:=0 |
  new_vector(V, L).
args_vector(F,Arity,V,L) :- Arity > 0 |
  arg(Arity, F, Arg),
  L1 = [Arg | L],
  Arity1 := Arity - 1,
  args_vector(F, Arity1, V, L1).

new(Port) :- true |
  dklicio:bind(Port, R),
  new1(R).
new1(R) :- R = normal(ServSock) |
  accept(ServSock).

accept(ServSock) :- true |
  ServSock = [accept(R) | ServSock1],
  accept1(ServSock1, R).
accept1(ServSock,R) :- R=normal(Sock) |
  rpcd(Sock),
  accept(ServSock).

rpcd(Sock) :- Sock = [M:Call|Sock1] |
  functor(Call, Name, Arity),
  generic:new(module, Modu, M),
  generic:new(predicate,
    Pred, Modu, Name, Arity),
  args_vector(Call, Arity, V, []),
  apply(Pred, V).
```


A
C
K
L
B
K=L


A
C
K=L@C
K
L
B
K=L





三者通信 — 中継の終了