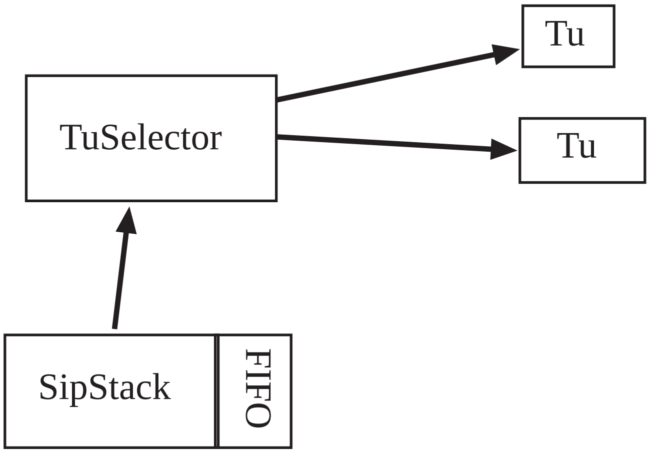
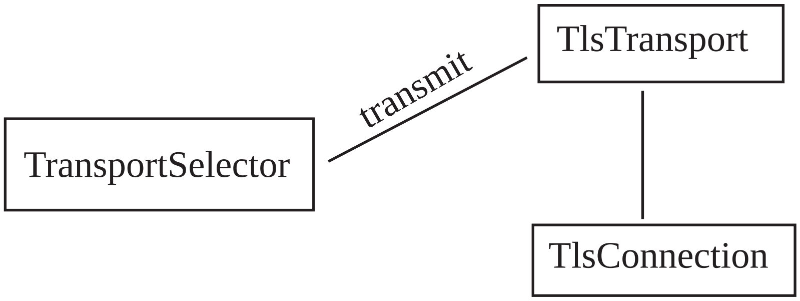
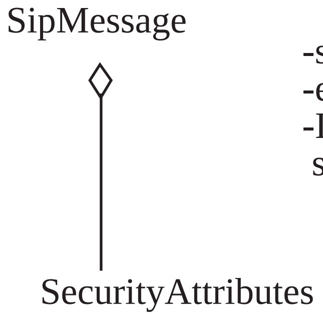

Tu
Tu
TuSelector
SipStack
FIFO

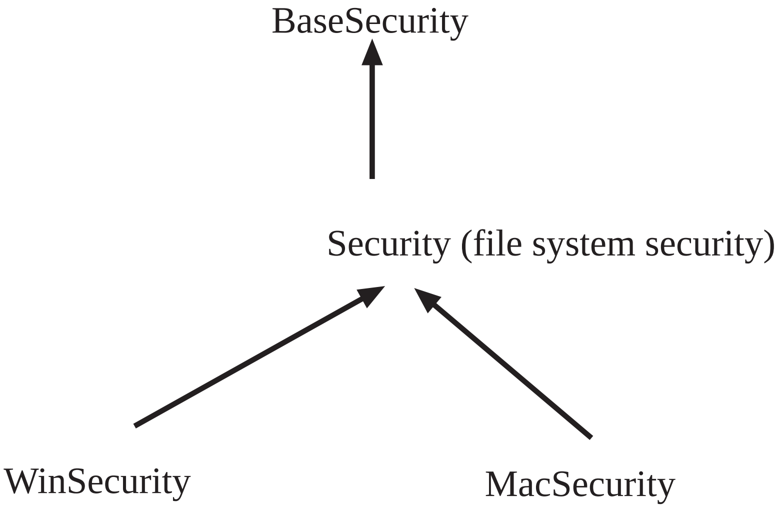
Dum

-Identity Checking
-s/mime manipulation

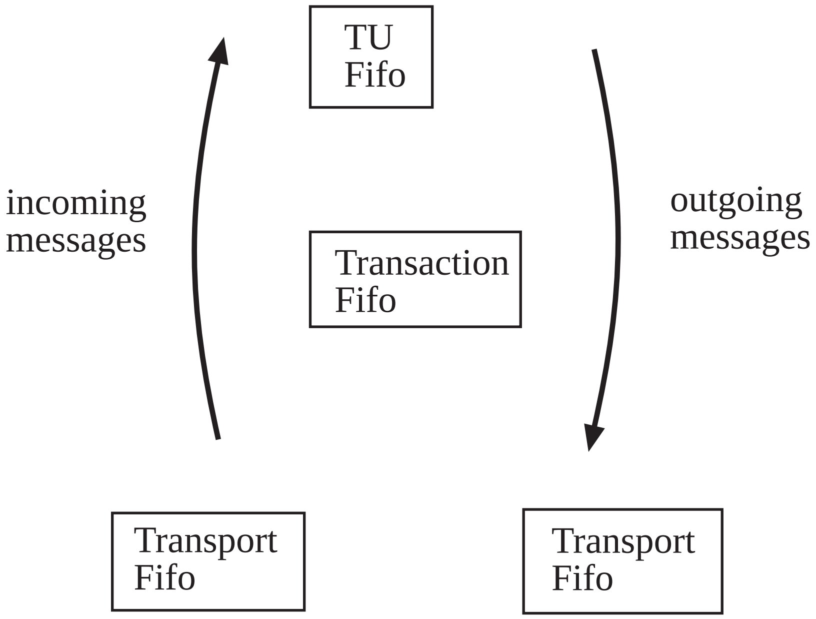

TransportSelector
transmit
TlsTransport
TlsConnection

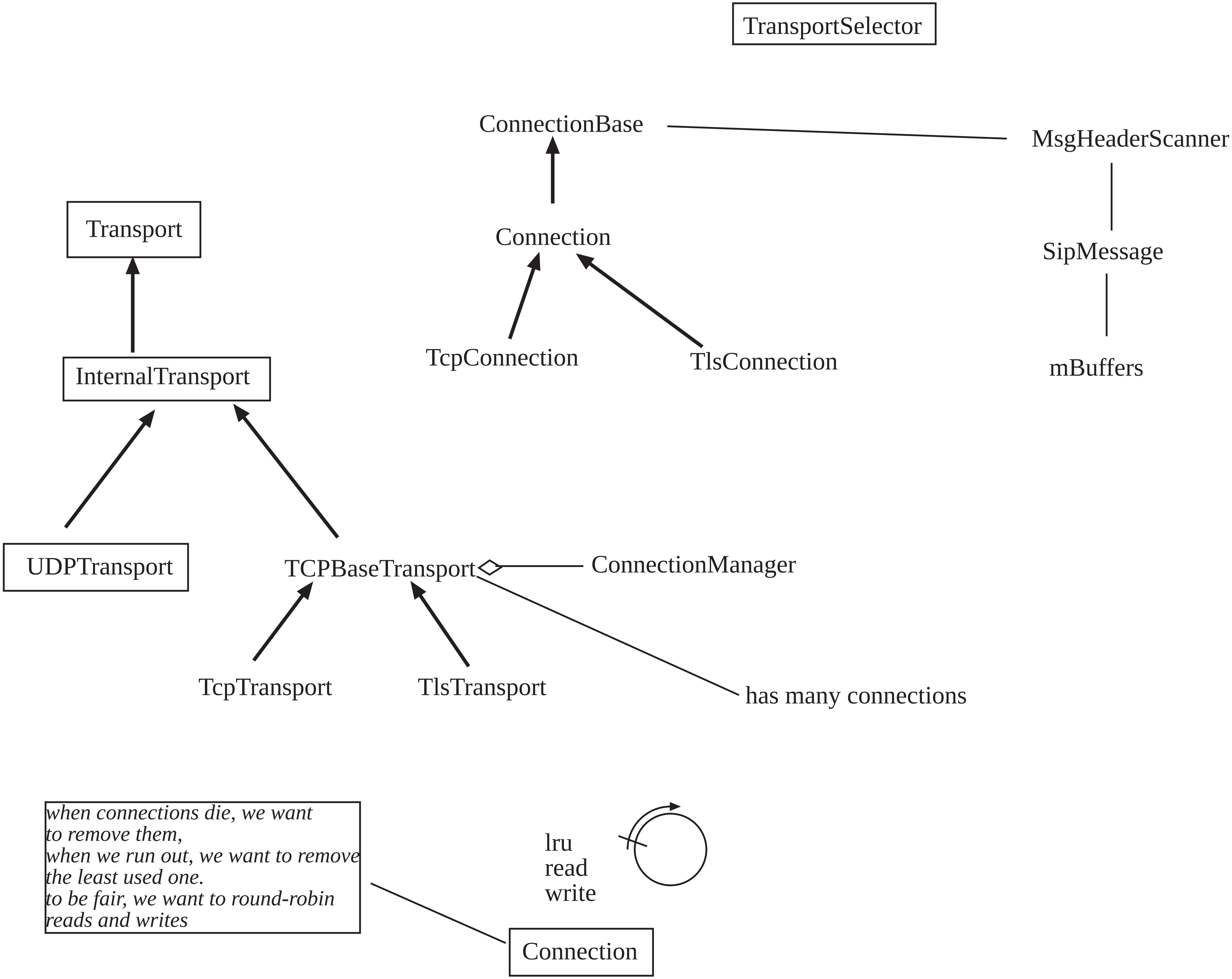

SipMessage
SecurityAttributes

-signal
-encrypted
-Indentity
strength


BaseSecurity
Security (file system security)
WinSecurity
MacSecurity

Fifo (often used as a way to communicate between threads)
add
getNext - it will block until things are in the Fifo,
there is also a method that will block only for so long.





when connections die, we want to remove them,
when we run out, we want to remove the least used one.
to be fair, we want to round-robin reads and writes

TLS domain is used to bind to the specific TLS interface.

You can’t receive connections for multiple domains on the same TSL connection.

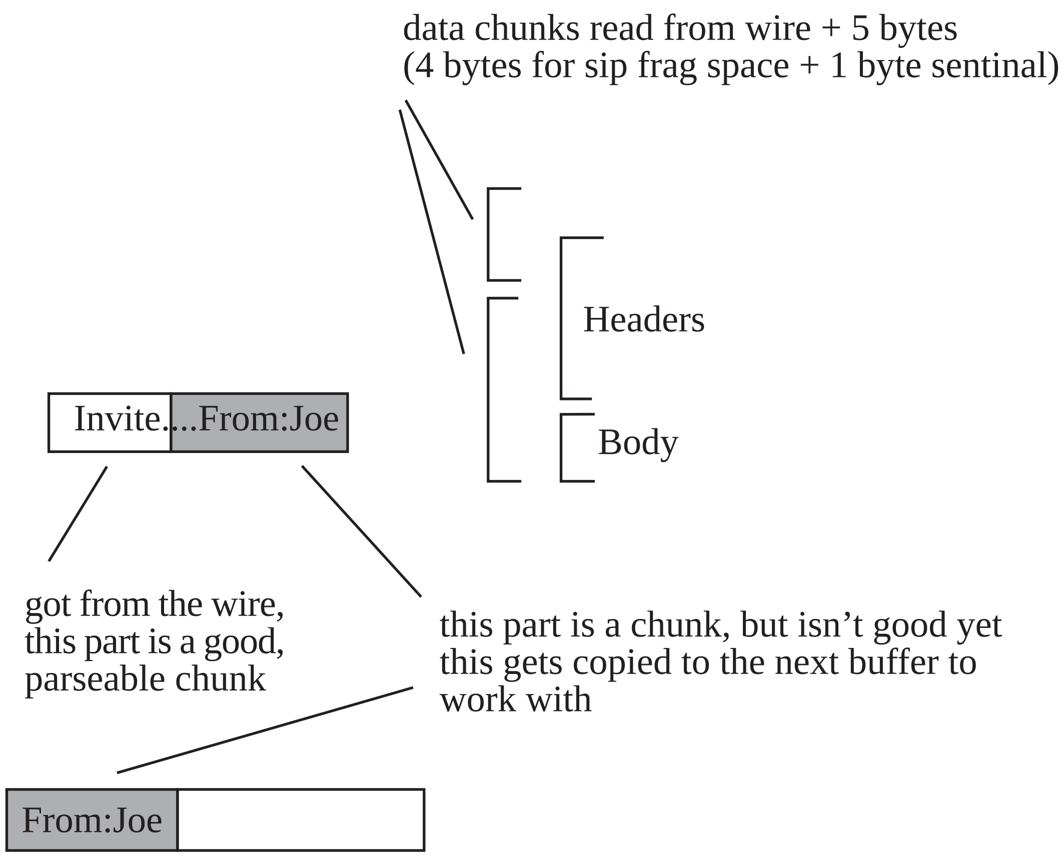
Internal is all the common code for the internal transports in ReSIP (file descriptor stuff and fifo live here now. They were originally in the Transport.
Transport is the contract for TransportSelectors

ConnectionBase has all the logic for framing messages coming in for conneciton oriented protocols.

Invite SIP 2.0

To
From
...
Contact
CRLF
CRLF
Body....

sent to
message
scanner

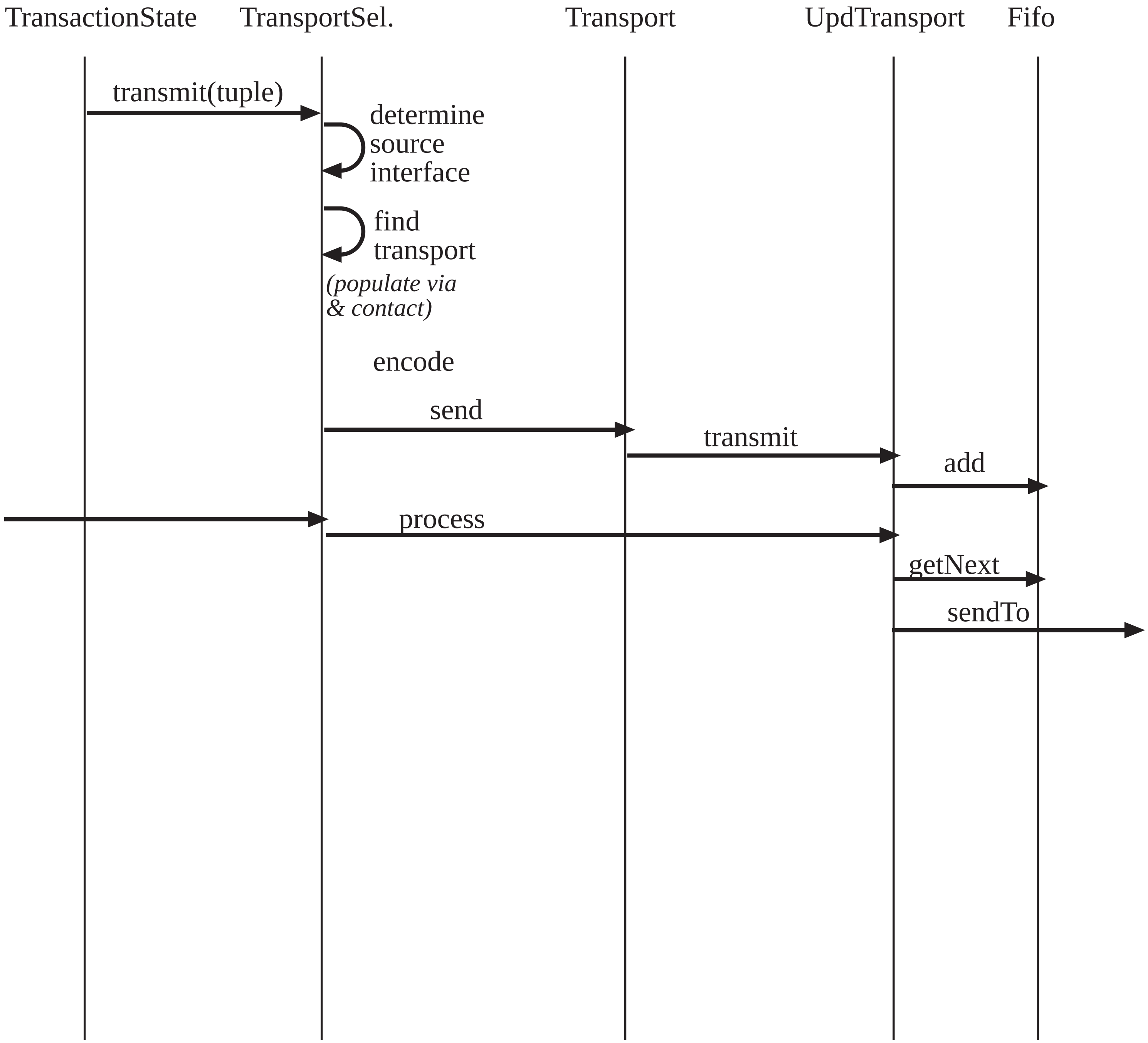


data chunks read from wire + 5 bytes
(4 bytes for sip frag space + 1 byte sentinal)

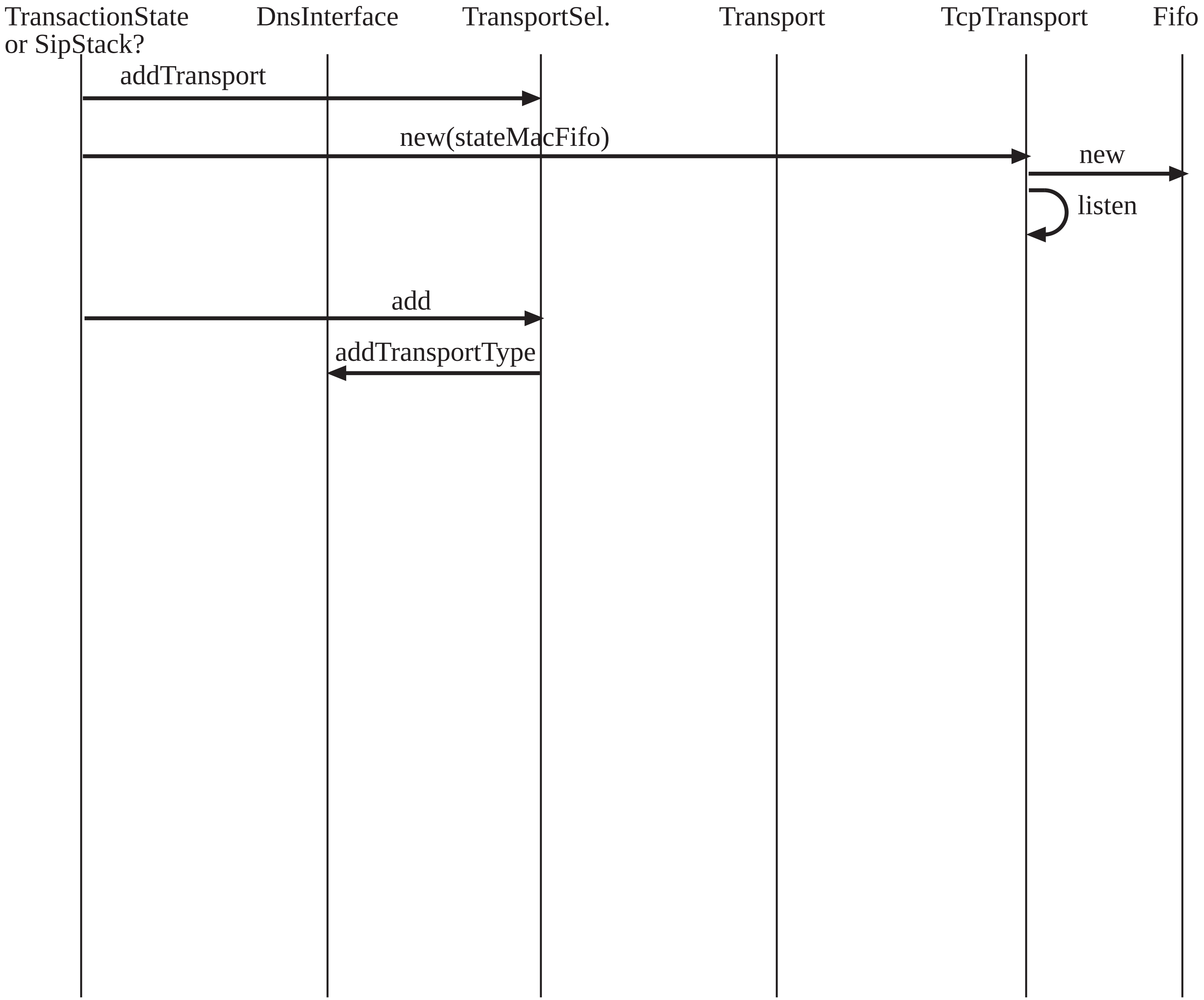
got from the wire,
this part is a good,
parseable chunk

this part is a chunk, but isn’t good yet
this gets copied to the next buffer to
work with

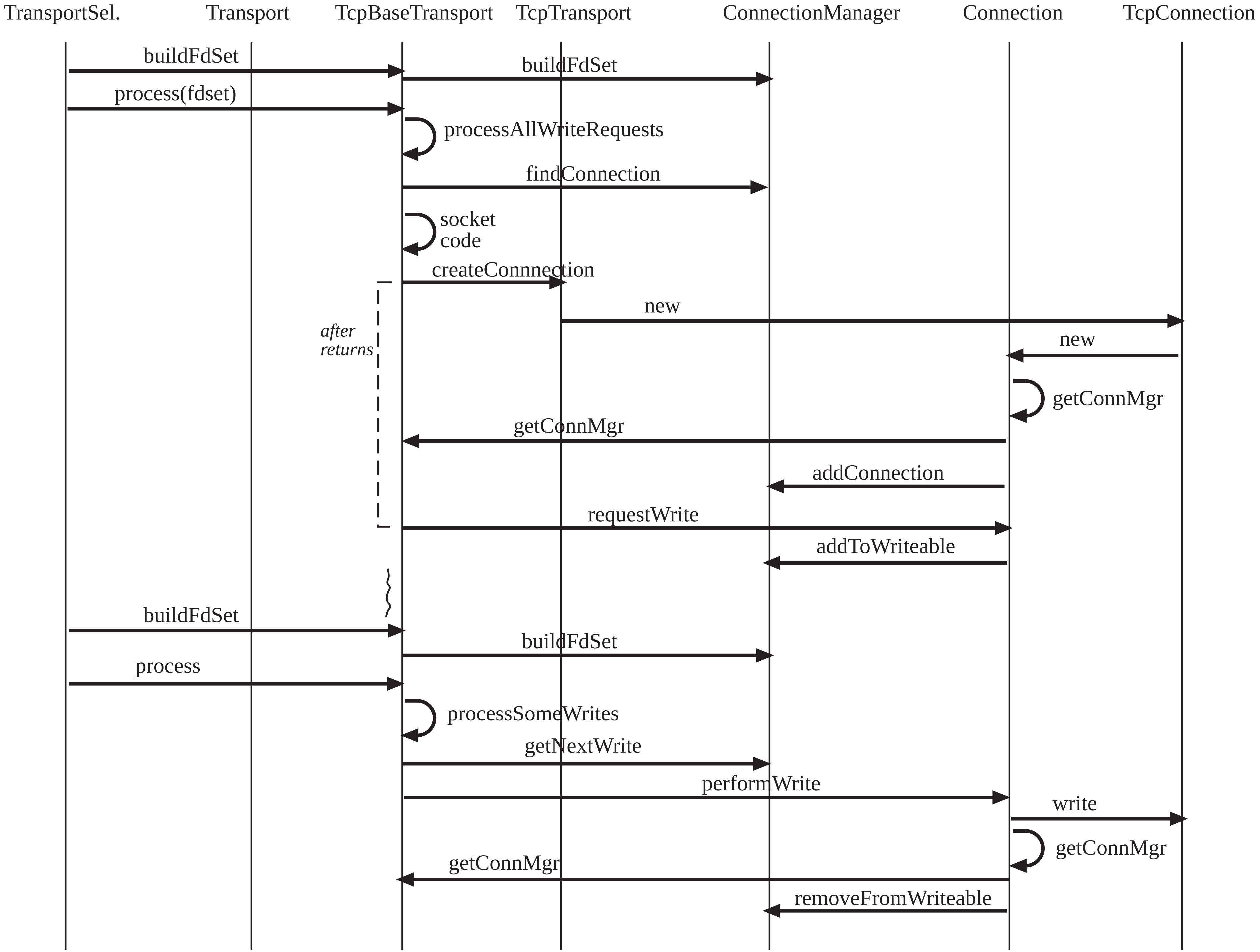
Sending datagram (we are after DNS resolution, no via hint)

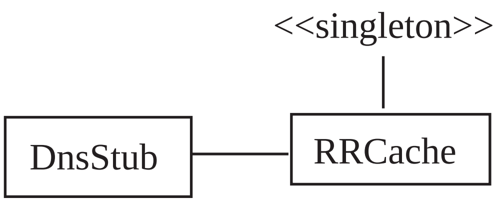


Adding a transport



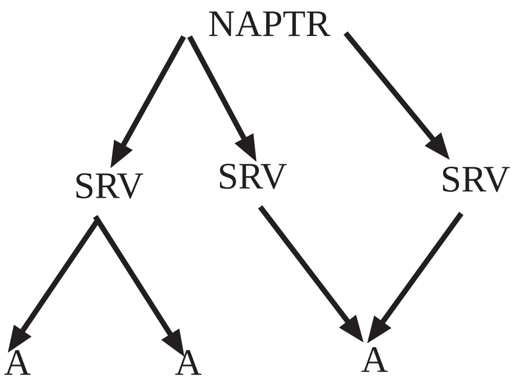
Receive msg over new tcp connection

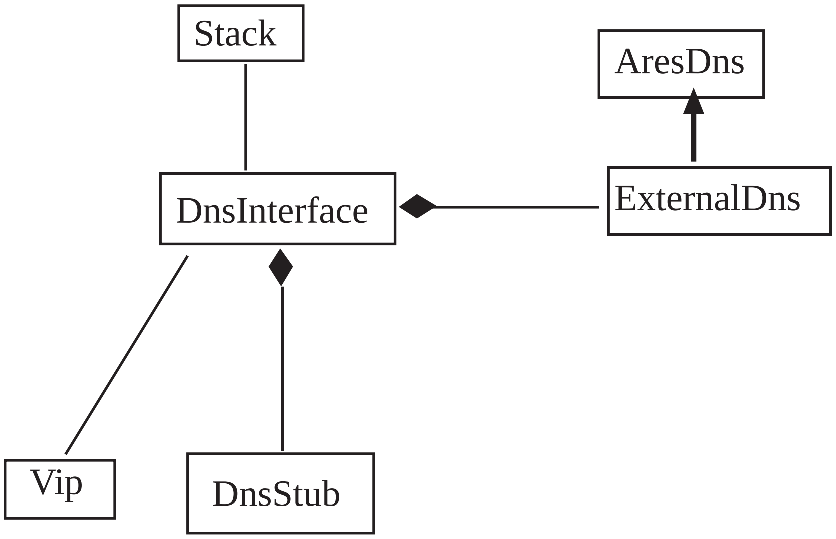


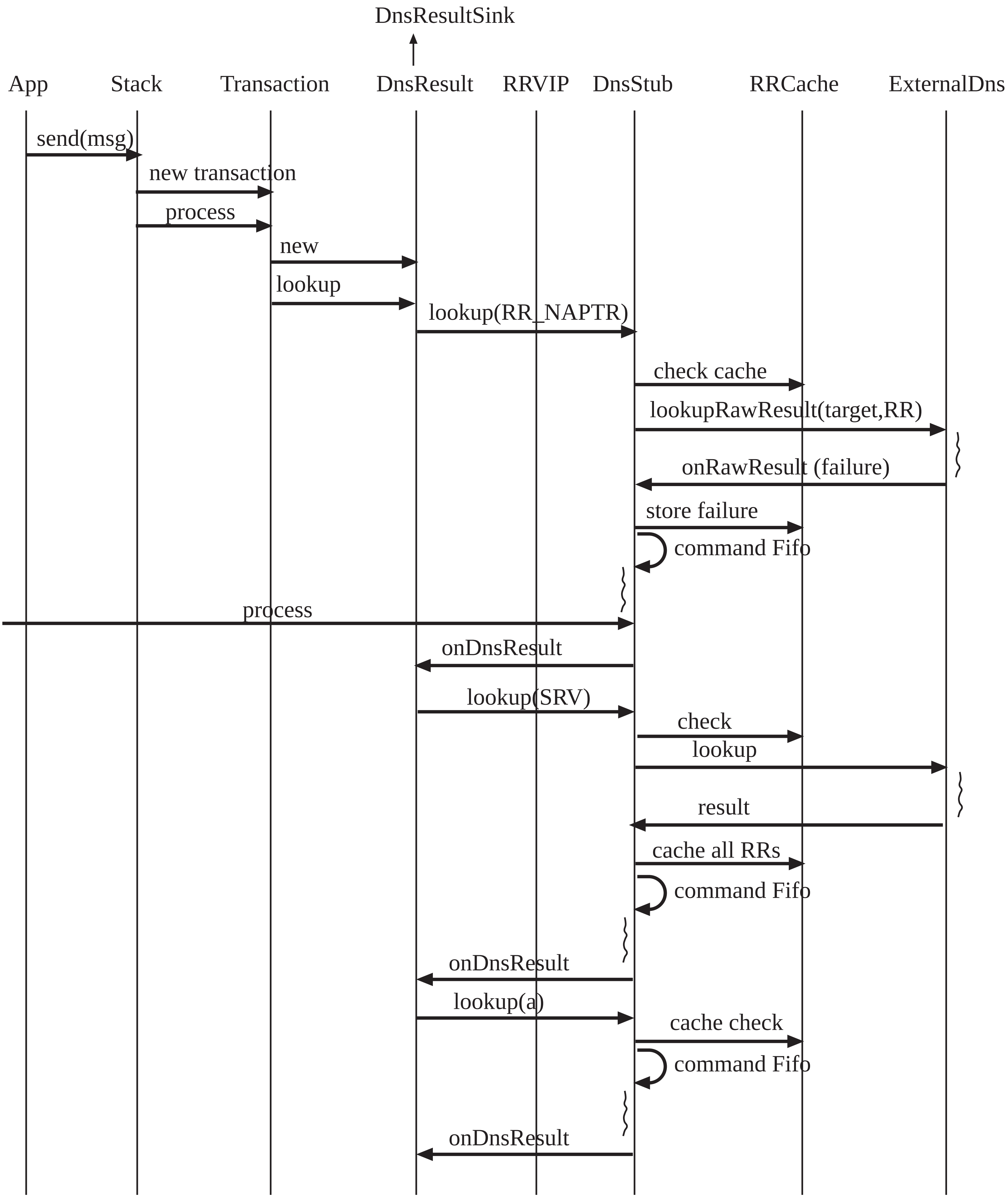


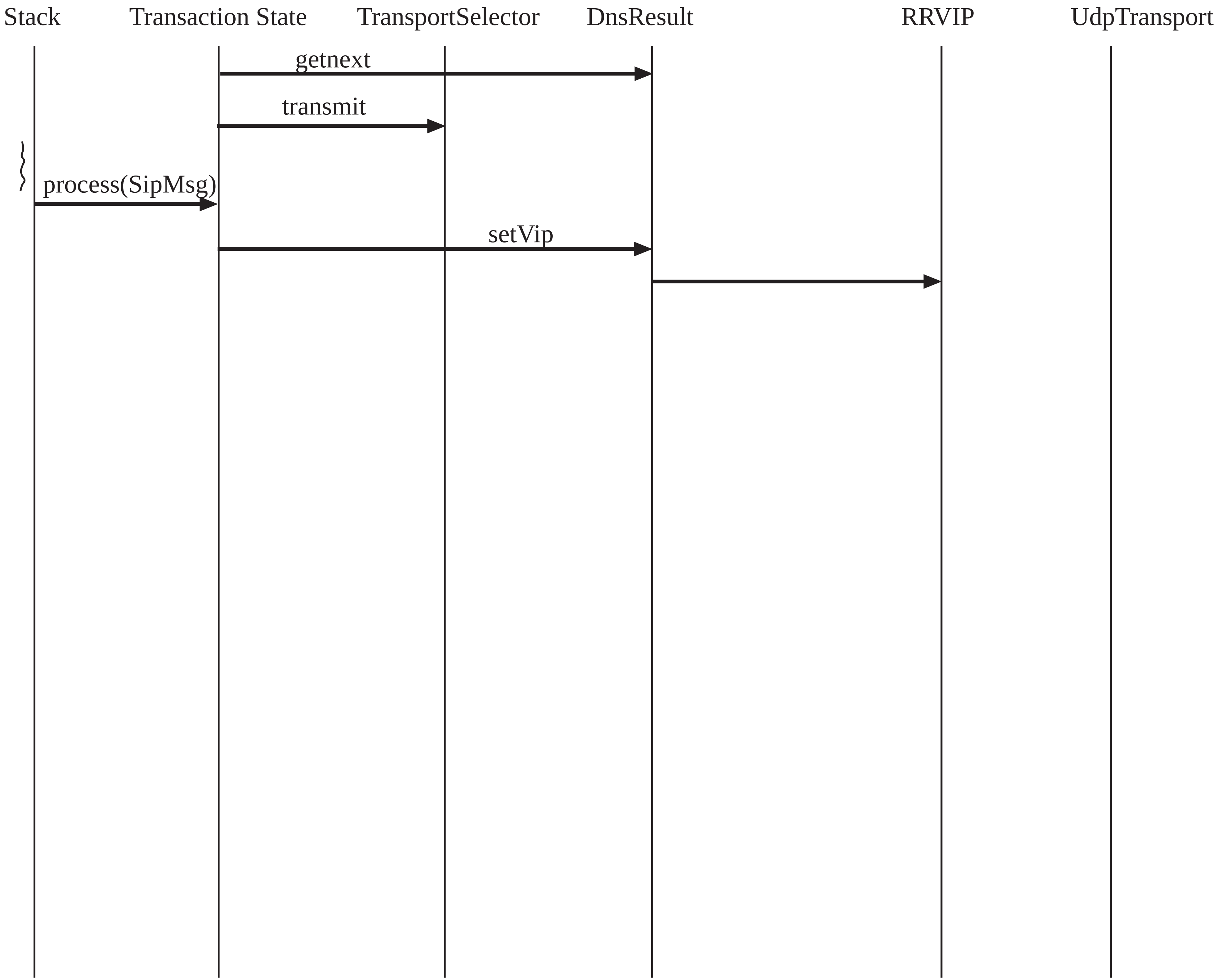
caching based on ttl

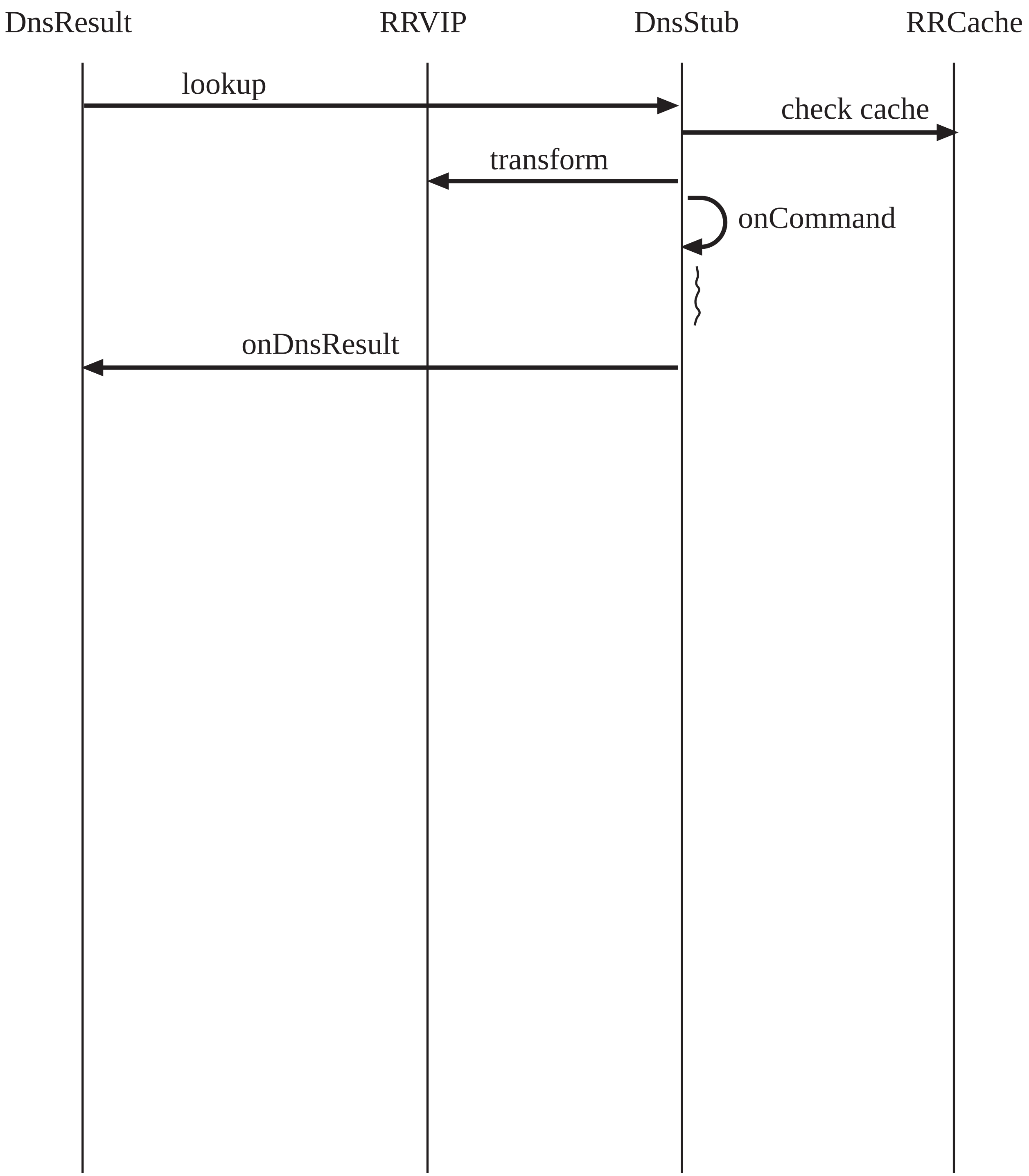
blacklisting interface (the path to the record, not just the record)

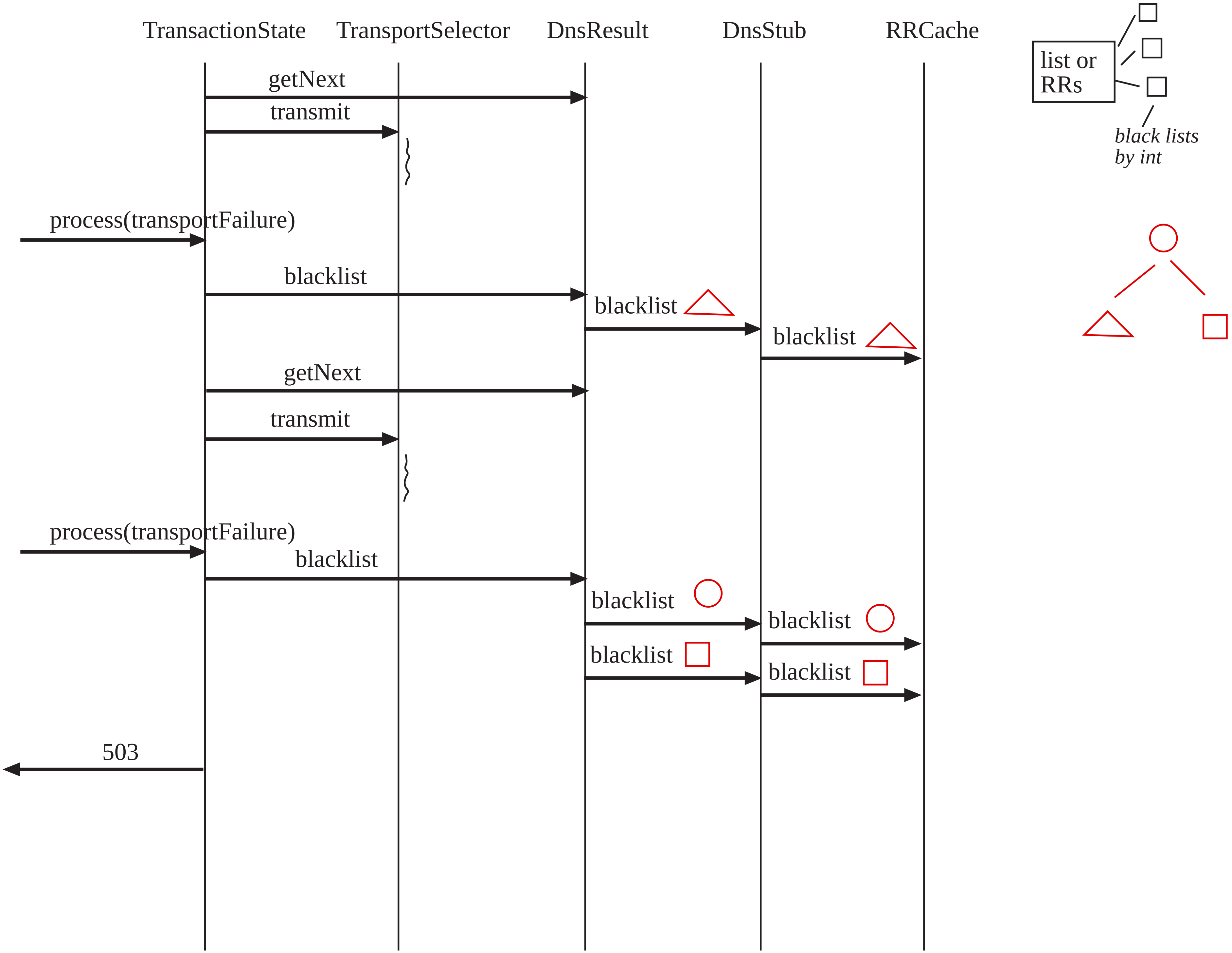


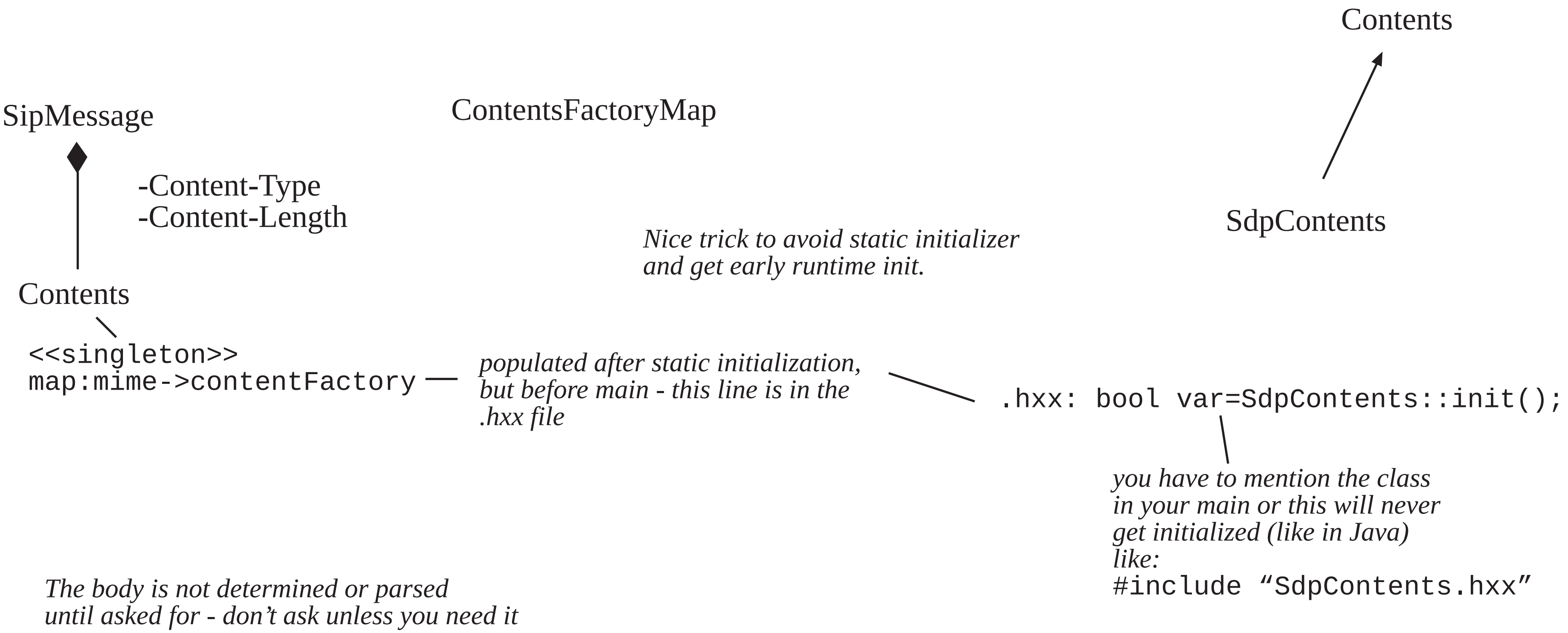



DnsResultSink
App
Stack
Transaction
DnsResult
RRVIP
DnsStub
RRCache
ExternalDns
send(msg)
new transaction
process
new
lookup
lookup(RR_NAPTR)
check cache
lookupRawResult(target,RR)
onRawResult (failure)
store failure
command Fifo
process
onDnsResult
lookup(SRV)
check
lookup
result
cache all RRs
command Fifo
onDnsResult
lookup(a)
cache check
command Fifo
onDnsResult


Stack
Transaction State
TransportSelector
DnsResult
RRVIP
UdpTransport
getnext
transmit
process(SipMsg)
setVip


DnsResult
RRVIP
DnsStub
RRCache
lookup
check cache
transform
onCommand
onDnsResult


TransactionState
TransportSelector
DnsResult
DnsStub
RRCache
list or RRs
black lists by int
getNext
transmit
process(transportFailure)
blacklist
blacklist
blacklist
getNext
transmit
process(transportFailure)
blacklist
blacklist
blacklist
blacklist
blacklist
503



Contents

SipMessage

ContentsFactoryMap

-Content-Type
-Content-Length

SdpContents

Nice trick to avoid static initializer
and get early runtime init.

Contents

```
<<singleton>>
map:mime->contentFactory
```

populated after static initialization,
but before main - this line is in the
.hxx file

```
.hxx: bool var=SdpContents::init();
```

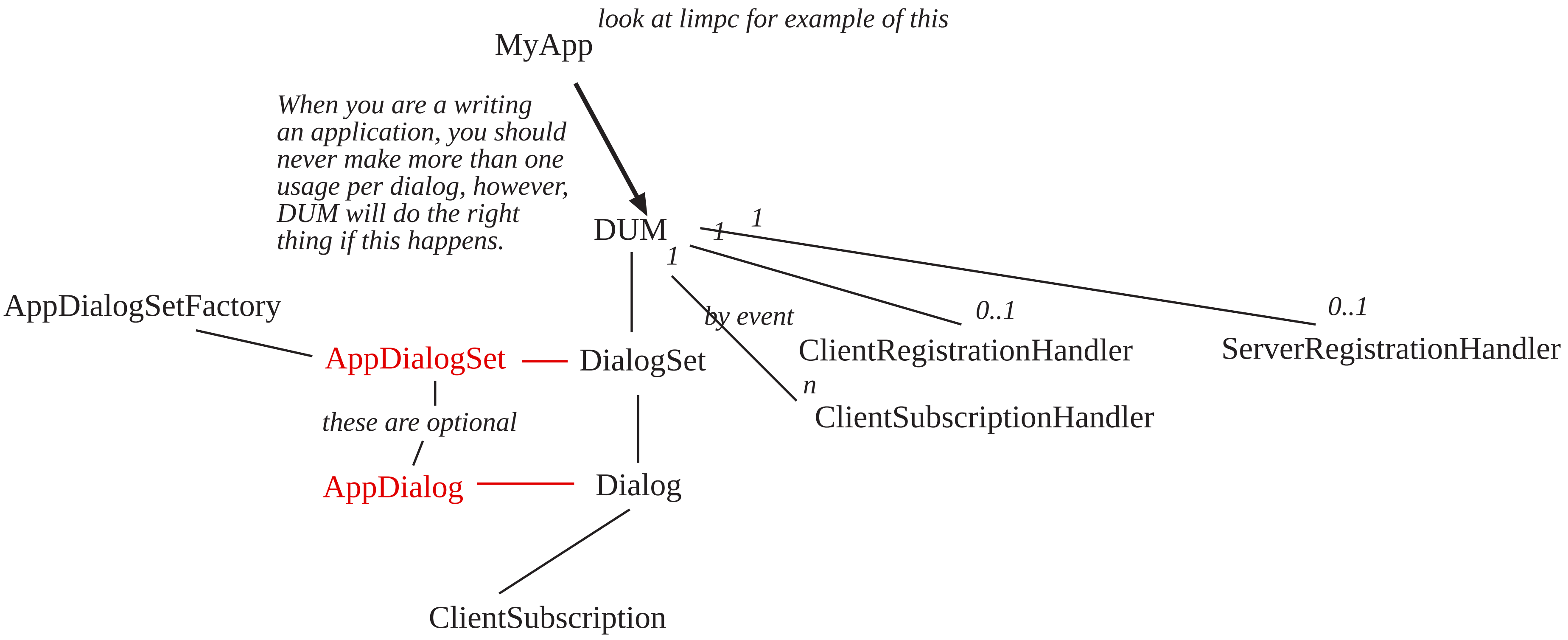
you have to mention the class
in your main or this will never
get initialized (like in Java)
like:

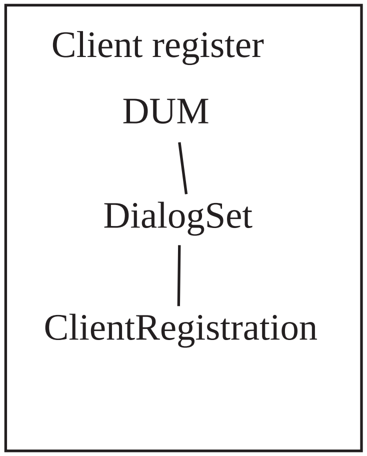
```
#include “SdpContents.hxx”
```

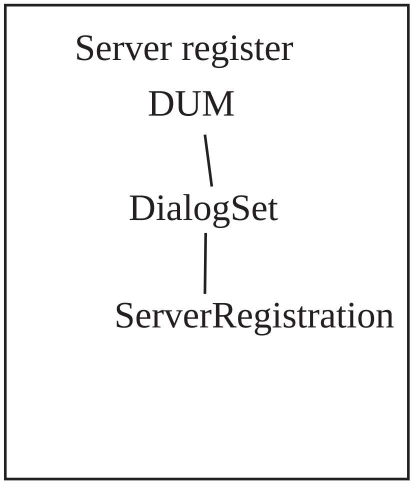
The body is not determined or parsed
until asked for - don’t ask unless you need it

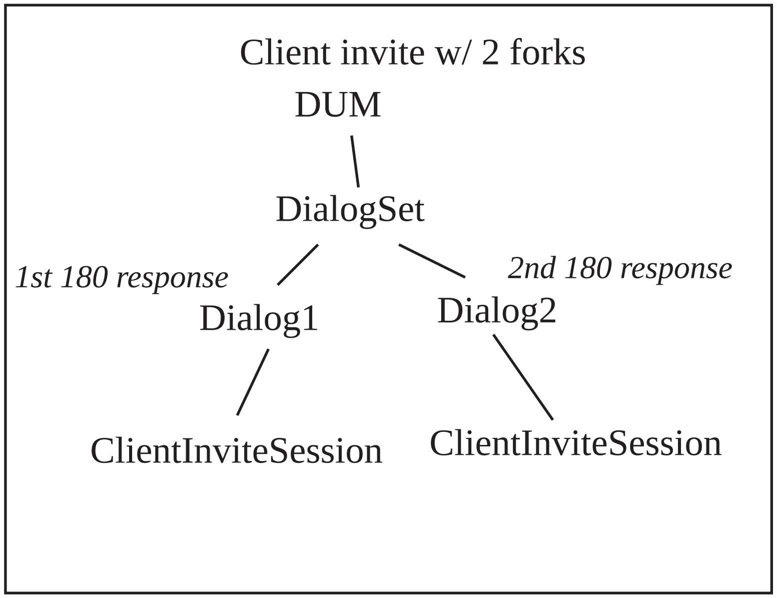
```
Contents* SipMessage::getContents
```

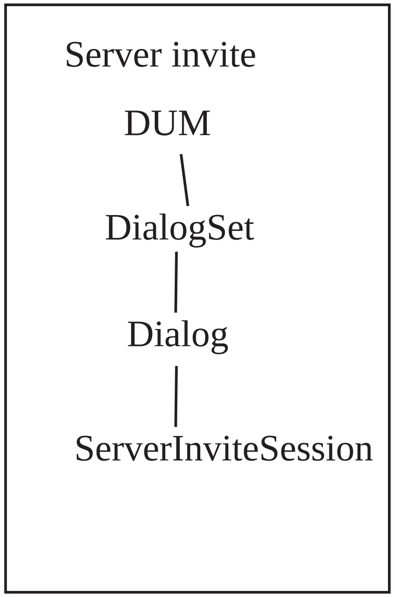
```
SdpContents*sdp =  dynamic_cast<SdpContents*>(msg.getContents());
```

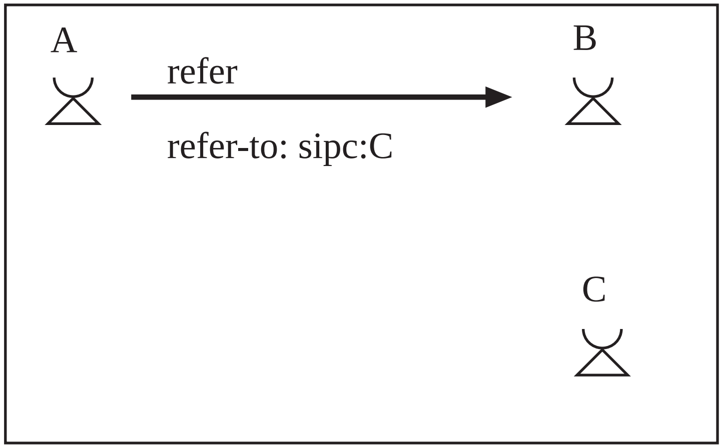

look at limpc for example of this
MyApp
When you are a writing an application, you should never make more than one usage per dialog, however, DUM will do the right thing if this happens.
DUM
1
1
1
AppDialogSetFactory
by event
0..1
0..1
ClientRegistrationHandler
ServerRegistrationHandler
AppDialogSet
DialogSet
n
ClientSubscriptionHandler
these are optional
AppDialog
Dialog
ClientSubscription

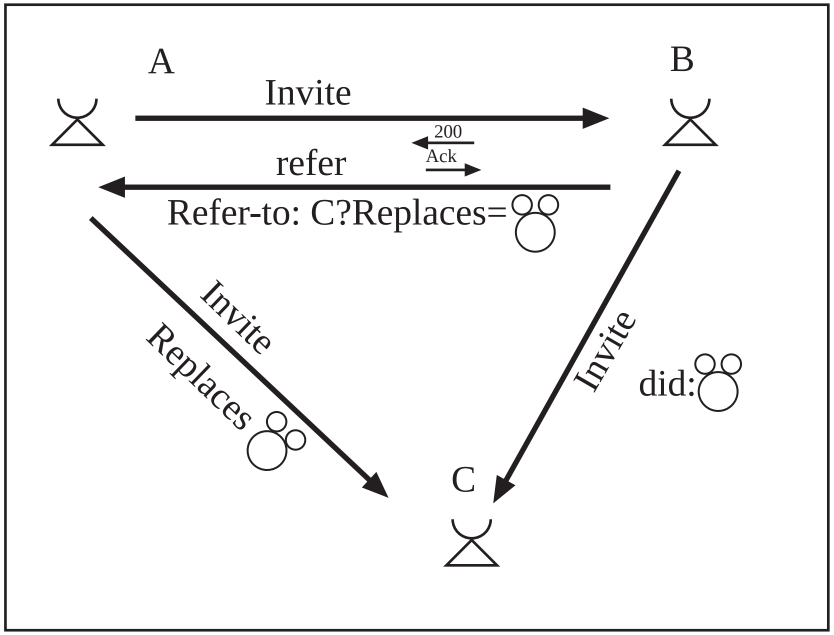

Client register
DUM
DialogSet
ClientRegistration

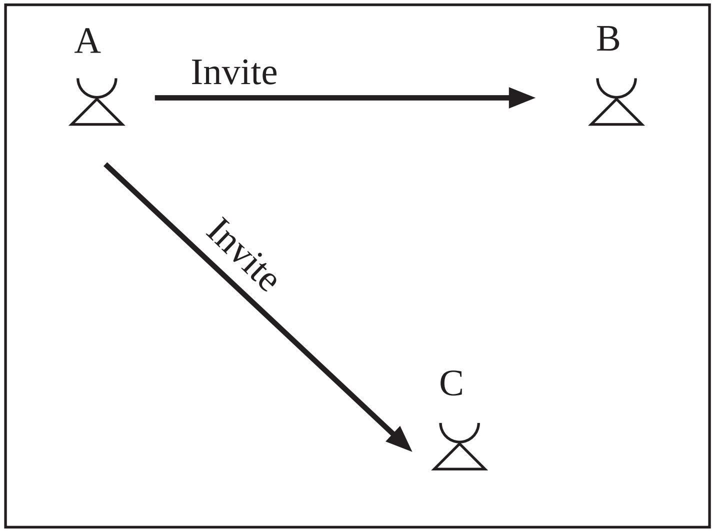

Server register
DUM
DialogSet
ServerRegistration

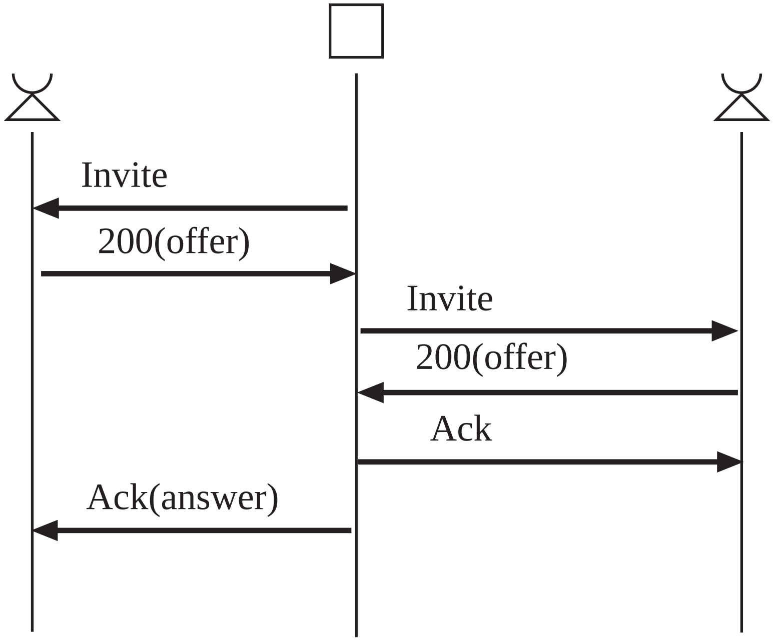

Client invite w/ 2 forks
DUM
DialogSet
1st 180 response
2nd 180 response
Dialog1
Dialog2
ClientInviteSession
ClientInviteSession

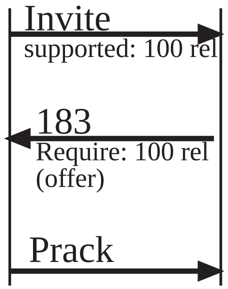

Server invite
DUM
DialogSet
Dialog
ServerInviteSession

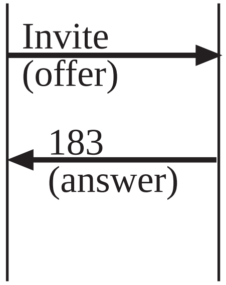

A
B
refer
refer-to: sipc:C
C


A
B
Invite
200
Ack
refer
Refer-to: C?Replaces=
Invite
Replaces
Invite
did:
C

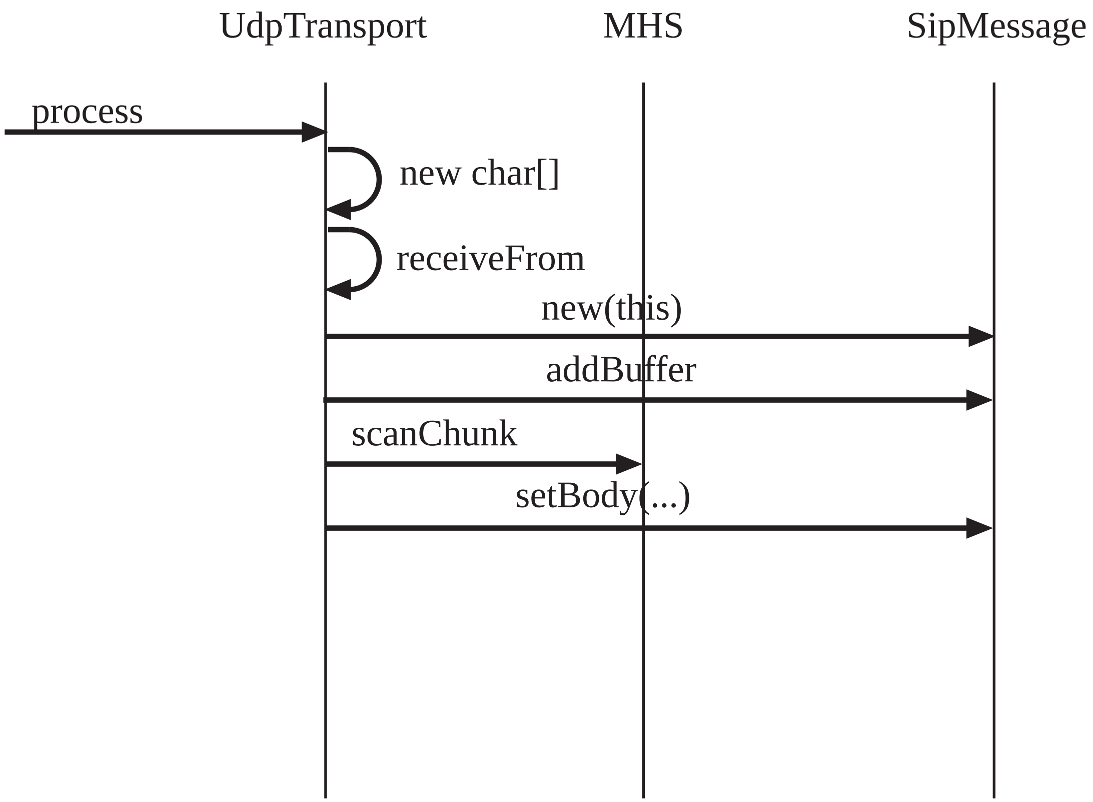

A
B
Invite
Invite
C


Invite
200(offer)
Invite
200(offer)
Ack
Ack(answer)

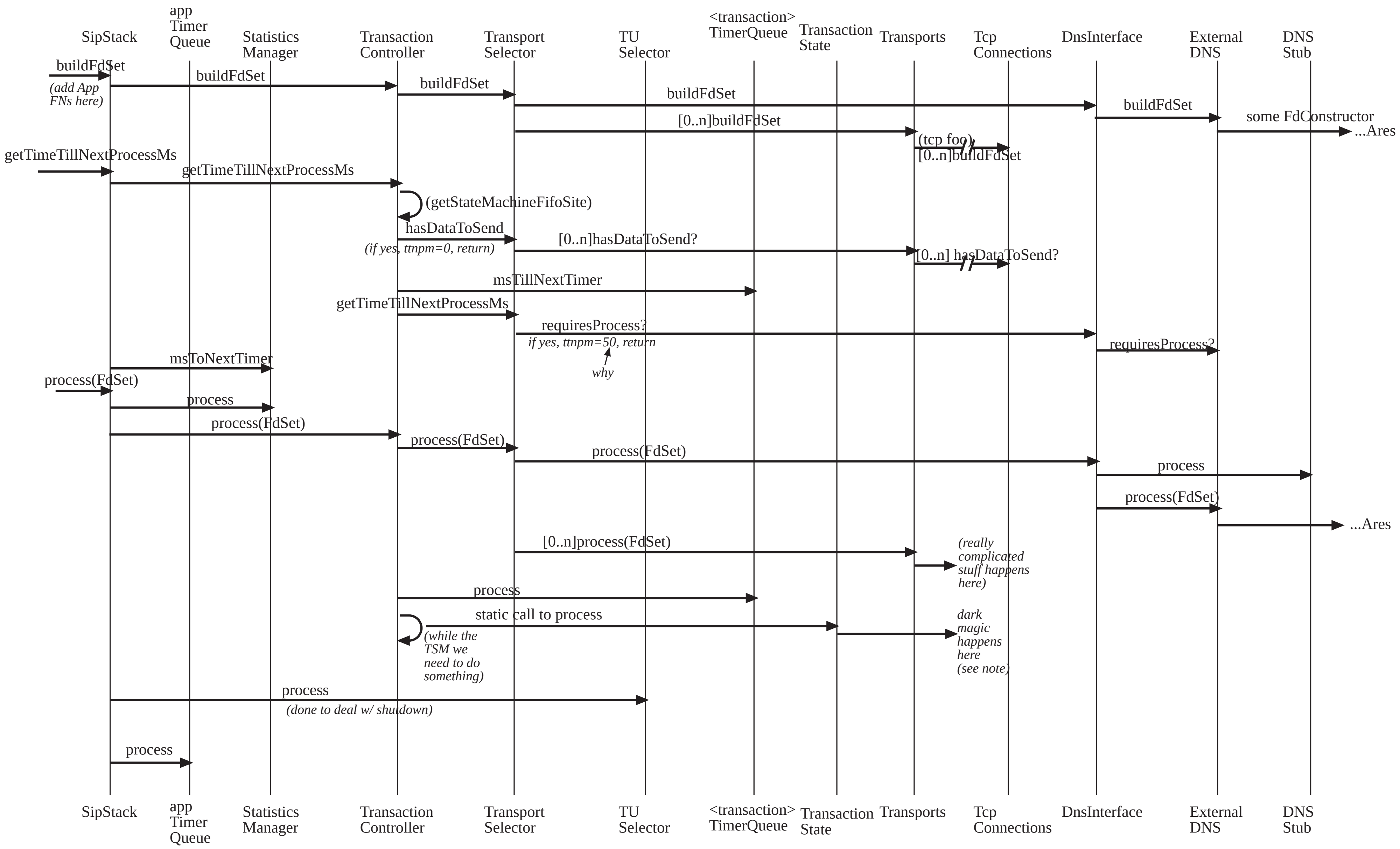

Invite
supported: 100 rel
183
Require: 100 rel
(offer)
Prack


Invite
(offer)
183
(answer)

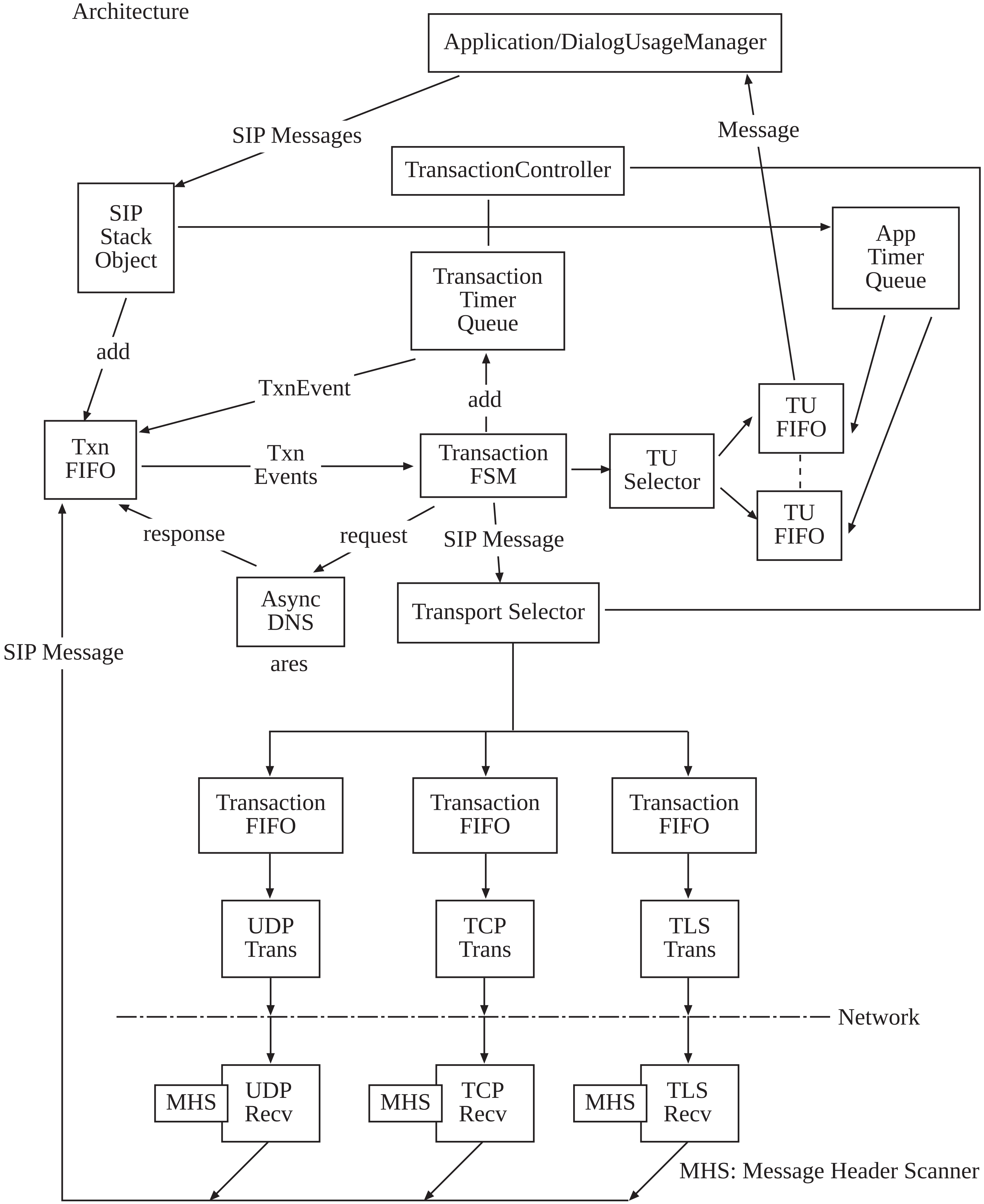
Process Incoming UDP



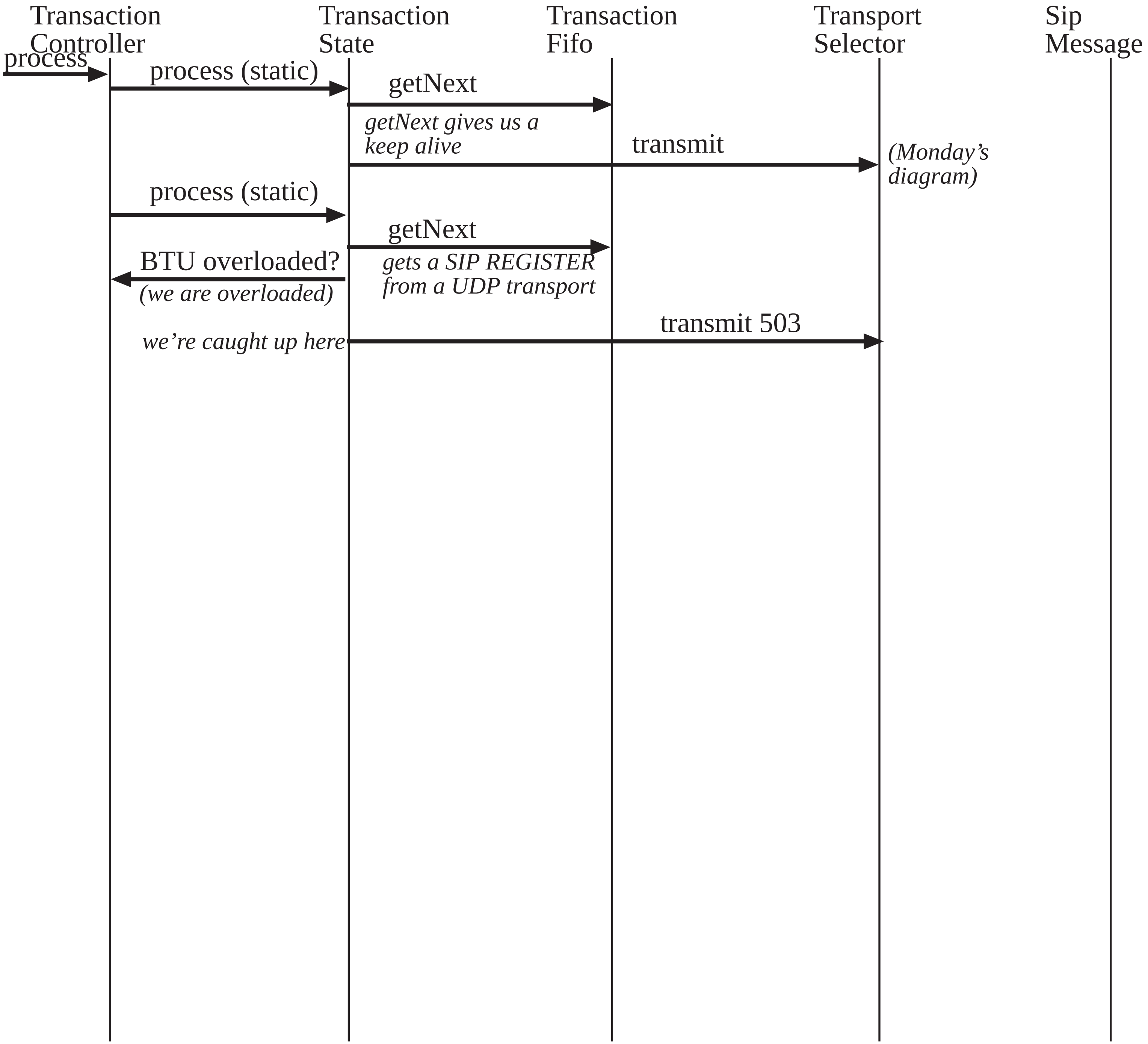
Main Event Loop



reSIProcate
Architecture



Transaction Nightmare



Transaction Controller

Transaction State

Transaction Fifo

Transport Selector

Sip Message

process

process (static)

getNext

getNext gives us a keep alive

transmit

(Monday's diagram)

process (static)

getNext

gets a SIP REGISTER from a UDP transport

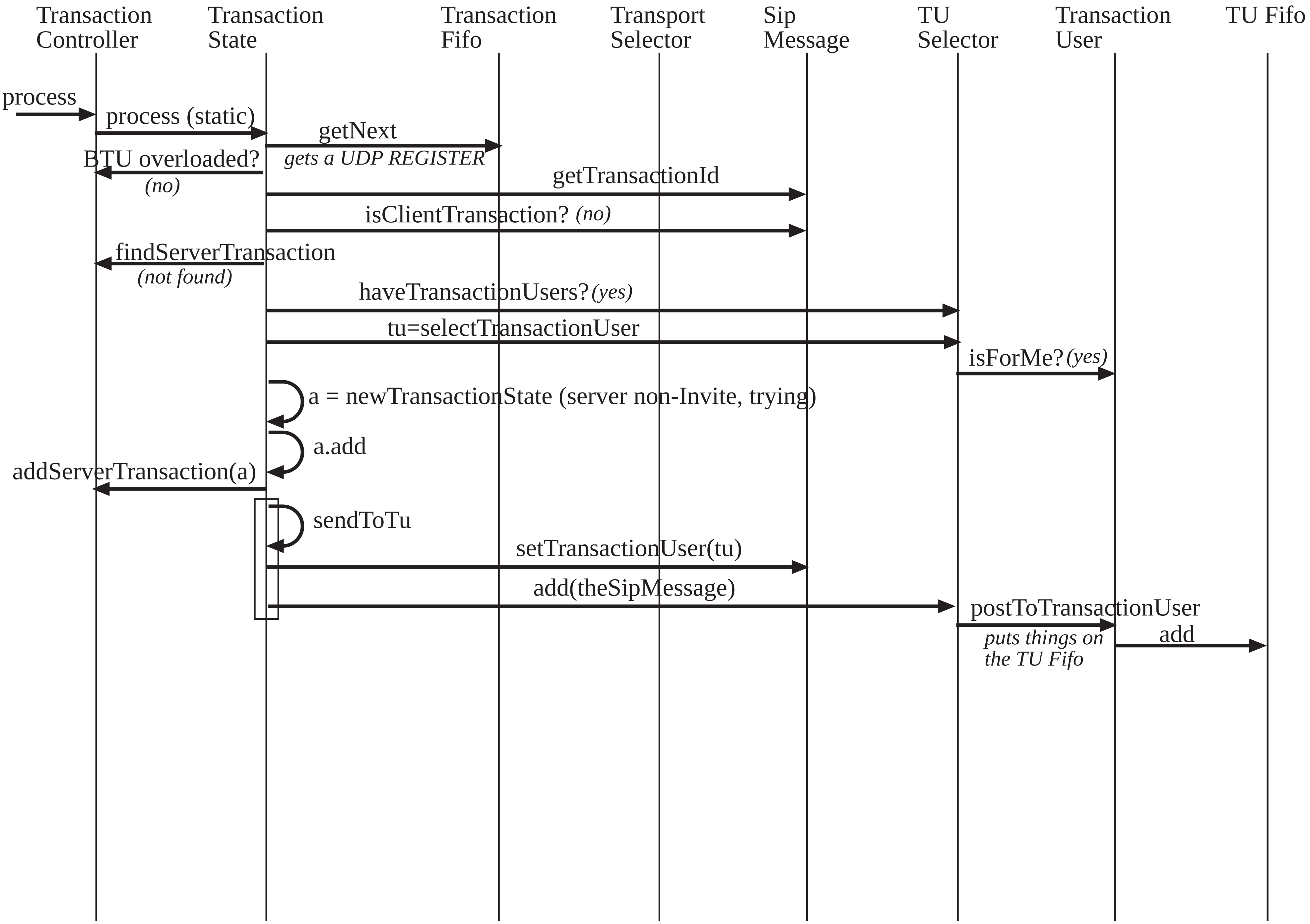
BTU overloaded?

(we are overloaded)

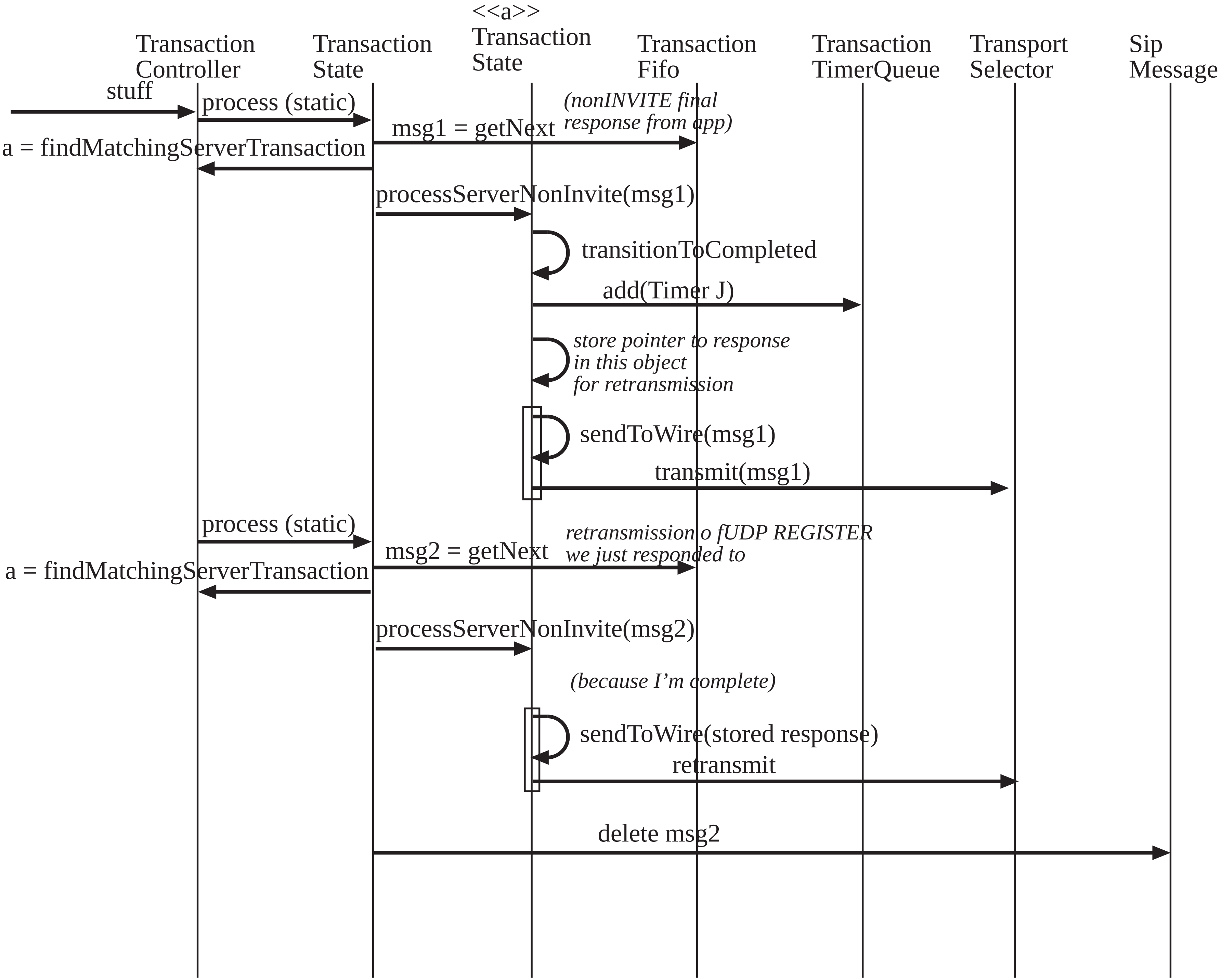
transmit 503

we're caught up here

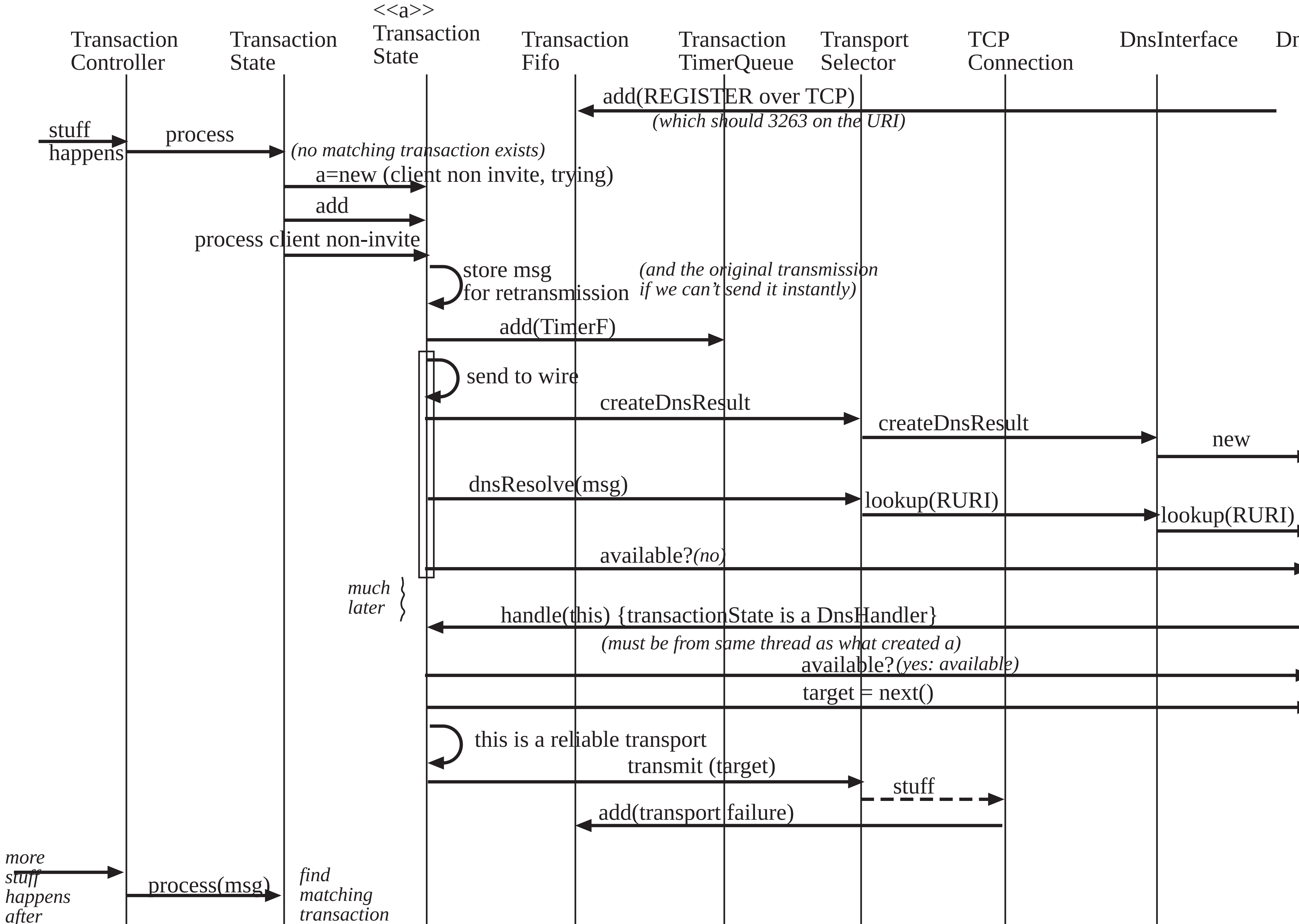
Server non-Invite w/UDP



TU sends response to non-invite request (UDP)



TU sends response to non-invite request (UDP)



<<a>>
Transaction Controller
Transaction State
Transaction State
Transaction Fifo
Transaction TimerQueue
Transport Selector
TCP Connection
DnsInterface
Dn
add(REGISTER over TCP)
(which should 3263 on the URI)
stuff happens
process
(no matching transaction exists)
a=new (client non invite, trying)
add
process client non-invite
store msg for retransmission
(and the original transmission if we can't send it instantly)
add(TimerF)
send to wire
createDnsResult
createDnsResult
new
dnsResolve(msg)
lookup(RURI)
lookup(RURI)
available?(no)
much later
handle(this) {transactionState is a DnsHandler}
(must be from same thread as what created a)
available?(yes: available)
target = next()
this is a reliable transport
transmit (target)
stuff
add(transport failure)
more stuff happens after
process(msg)
find matching transaction