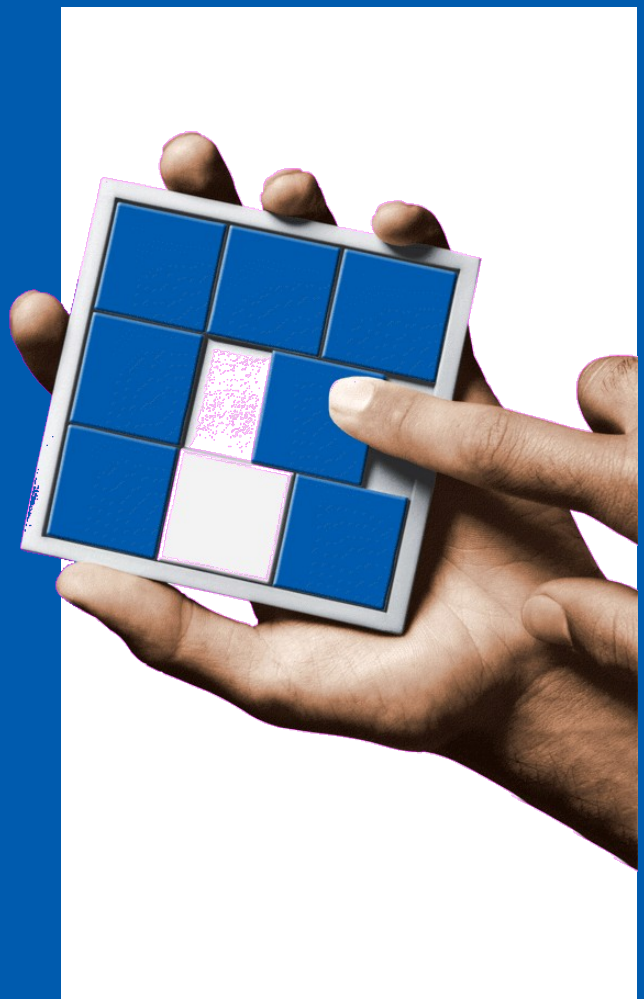
NetApp™

Go further, faster™

Linux NFSv4.1 Backchannel Work in Progress

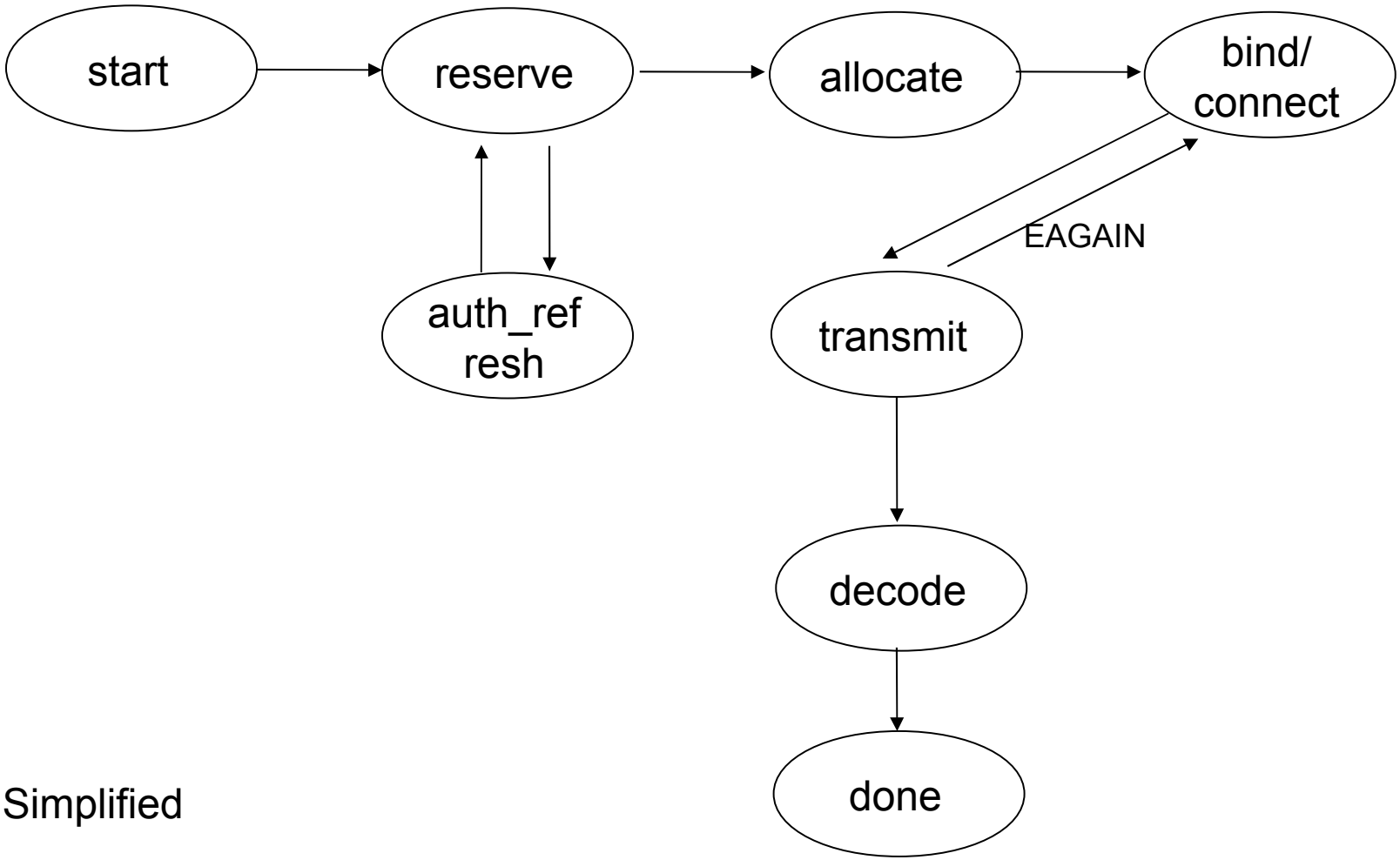
Ricardo Labiaga
ricardo dot labiaga at netapp dot com
Connectathon February 24, 2009



Agenda

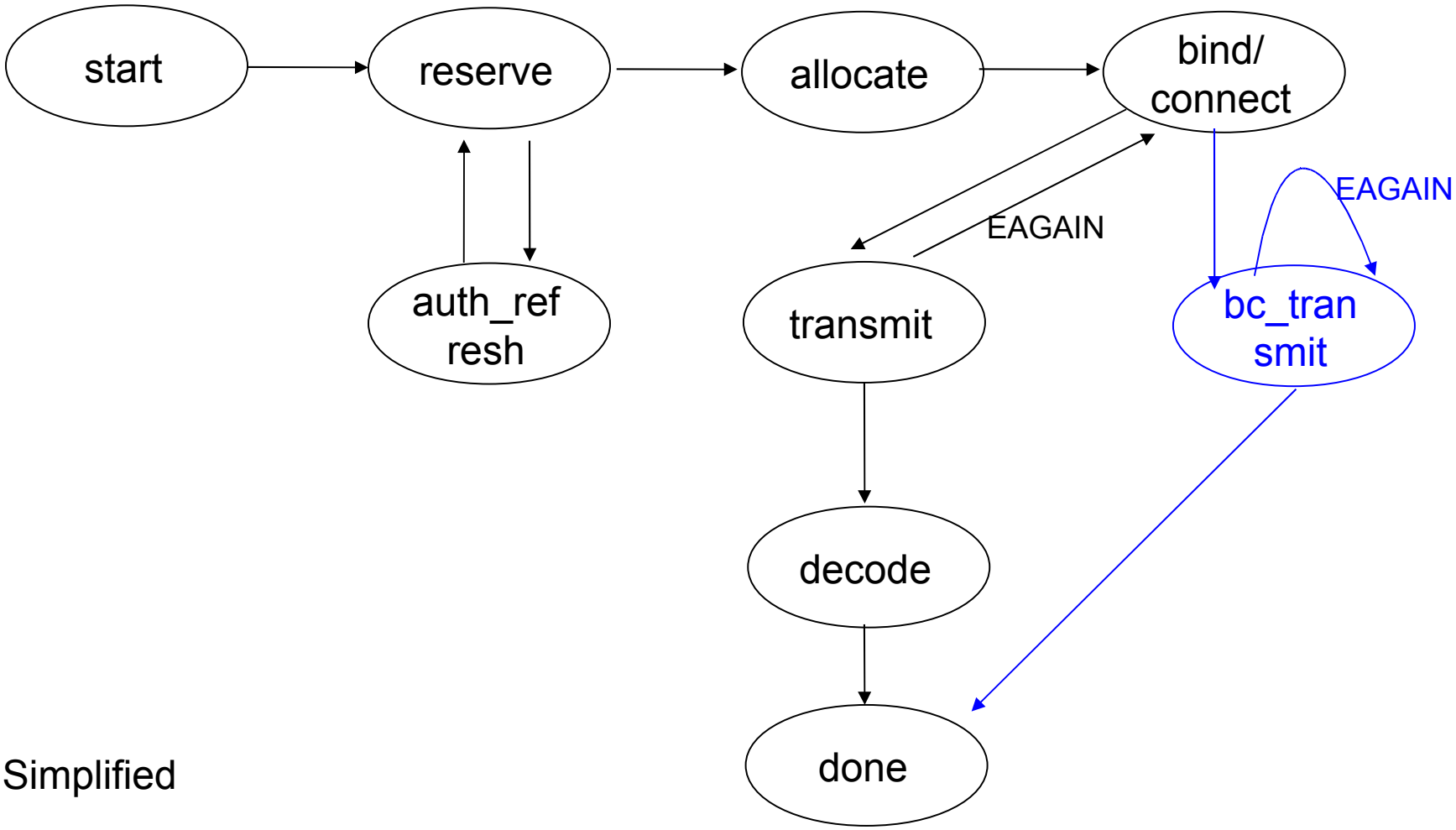
- Architecture Linux Backchannel Client Implementation
  - Initialization
  - Call Routing
  - Call Processing
- RPCSEC_GSS Authentication
- Todo's

RPC Client State Machine*



* Simplified

RPC Client State Machine*
and Backchannel Processing



* Simplified

Backchannel Initialization

- Initialized during Session setup
- Single slot
- Preallocates a '*struct rpc_rqst*', receive and send XDR buffers for each slot
- *Will* associate existing RPCSEC_GSS context with the backchannel
- If necessary, spawns new NFSv4.1 backchannel service (thread)
  - Backchannel service listens on callback wait queue

Callback Routing

- Callback arrives on the connection
- RPC TCP transport *xs_tcp_data_recv()* determines direction of the RPC
  - Reply
    - Find matching request and wake up waiting task
  - Callback Request
    - Obtain previously allocated '*struct rpc_rqst'* and read the data into the preallocated buffers
    - Place request in the callback queue

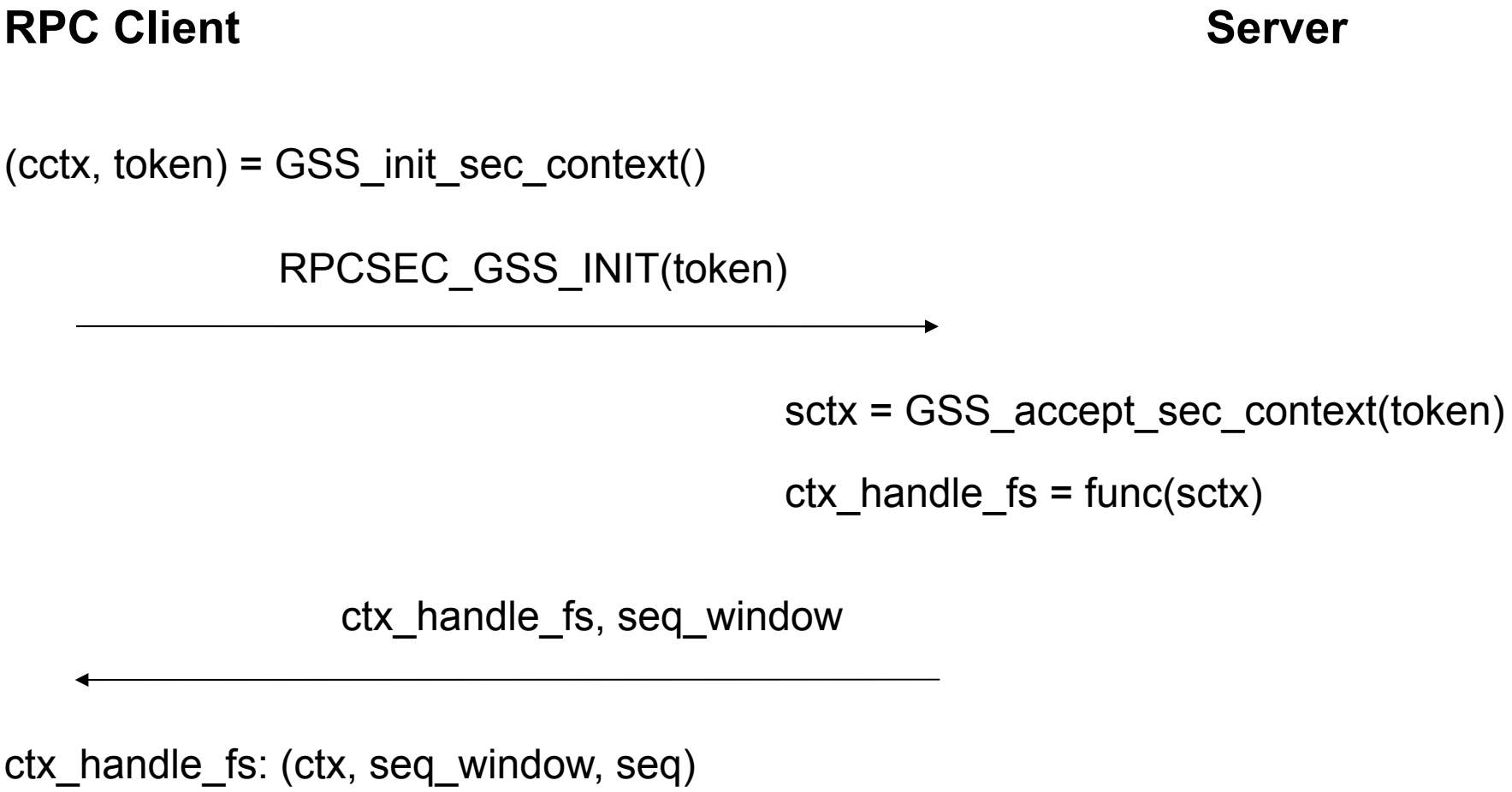
Callback Processing

- NFSv4.1 callback service grabs '*struct rpc_rqst'* from the callback wait queue and process it
- Common authentication and dispatch functionality factored out of *svc_process()* into *svc_common_process()*
- Backchannel's *bc_svc_process()* builds a new '*struct svc_rqst'* and invokes *svc_common_process()*

```
svc_process() {
   // Setup XDR response
   svc_common_process();
   svc_send();
}
```

```
bc_svc_process() {
   // Construct 'struct svc_rqst'
   svc_common_process();
   bc_send();
}
```

GSS Context Initialization



Backchannel GSS Context Setup

RPC Client **Server**

ctx_handle_fs: (cctx, seq_window, curr_seq)

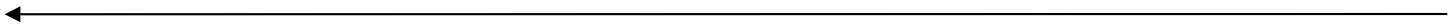
ctx_handle_fc: (cctx, seq_window, bc_seq_window_bitmap)

CREATE_SESSION(..., ctx_handle_fs, ctx_handle_fc)

sctx = $func^{-1}$(ctx_handle_fs)

Server associates *ctx_handle_fc with sctx* and *seq_window* :

sctx ~ $func^{-1}$(ctx_handle_fs) ~ $func^{-1}$(ctx_handle_fc)

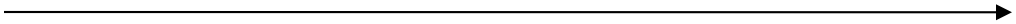


Data Exchange (Forechannel)

RPC Client **Server**

ctx_handle_fs: (cctx, seq_window, seq)

RPCSEC_GSS_DATA(ctx_handle_fs, seq)



sctx = $func^{-1}$(ctx_handle_fs)

GSS_VerifyMIC(sctx)

RPC processing...

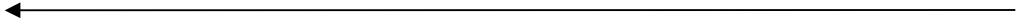
Callback

RPC Client **Server**

GSS_GetMIC(sctx)

ctx_handle_fc= $func^{-1}$(sctx)

RPCSEC_GSS_DATA(ctx_handle_fc, seq_fc)



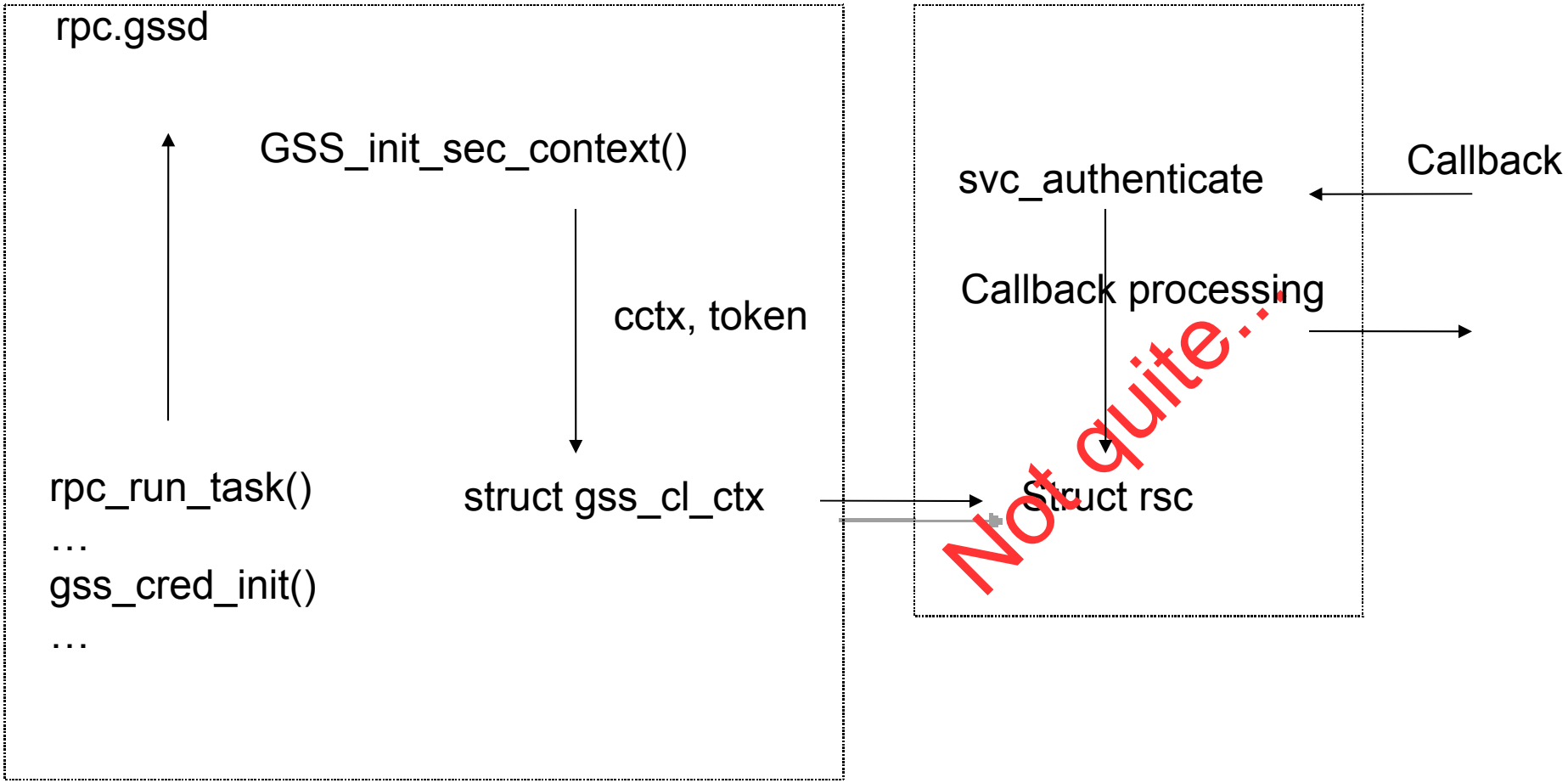
ctx_handle_fc: (cctx, seq_window, bc_seq_window_bitmap)
GSS_VerifyMIC(cctx)

Linux Client

NetApp

RPC Client

RPC Callback Service



Backchannel RPCSEC_GSS Context Caching

- Can't use server side caching as is
  - Assumes context is created by the server and unique
- Backchannel context handle is not generated by the client
- Different NFS servers may generate clashing context handles
- The client may in turn be an NFS server and generate a clashing context handle
- We know the credentials we used to create the session, compare directly with that

Backchanel on New Connection

- Unlike NFSv4, the NFSv4.1 backchannel connection is initiated by the client
  - Can't simply have backchannel service listening on socket: *svc_recv()*
- Backchannel communication needs to be received over the '*struct rpc_xprt*' and not the '*struct svc_xprt*'
- Allocate new '*struct rpc_xprt*' and a new '*struct rpc_clnt*' for every new connection
  - *'struct rpc_clnt'* needed for authentication information

Client Backchannel ToDo's

- Need SessionID and slotID verification
- Need slot replay cache – single slot
- Backchannel Only connection
  - Use existing mechanism that preallocates '*struct rpc_xprt*'s
- Implement RPCSEC_GSS backchannel

Thank you!

ricardo dot labiaga at netapp dot com