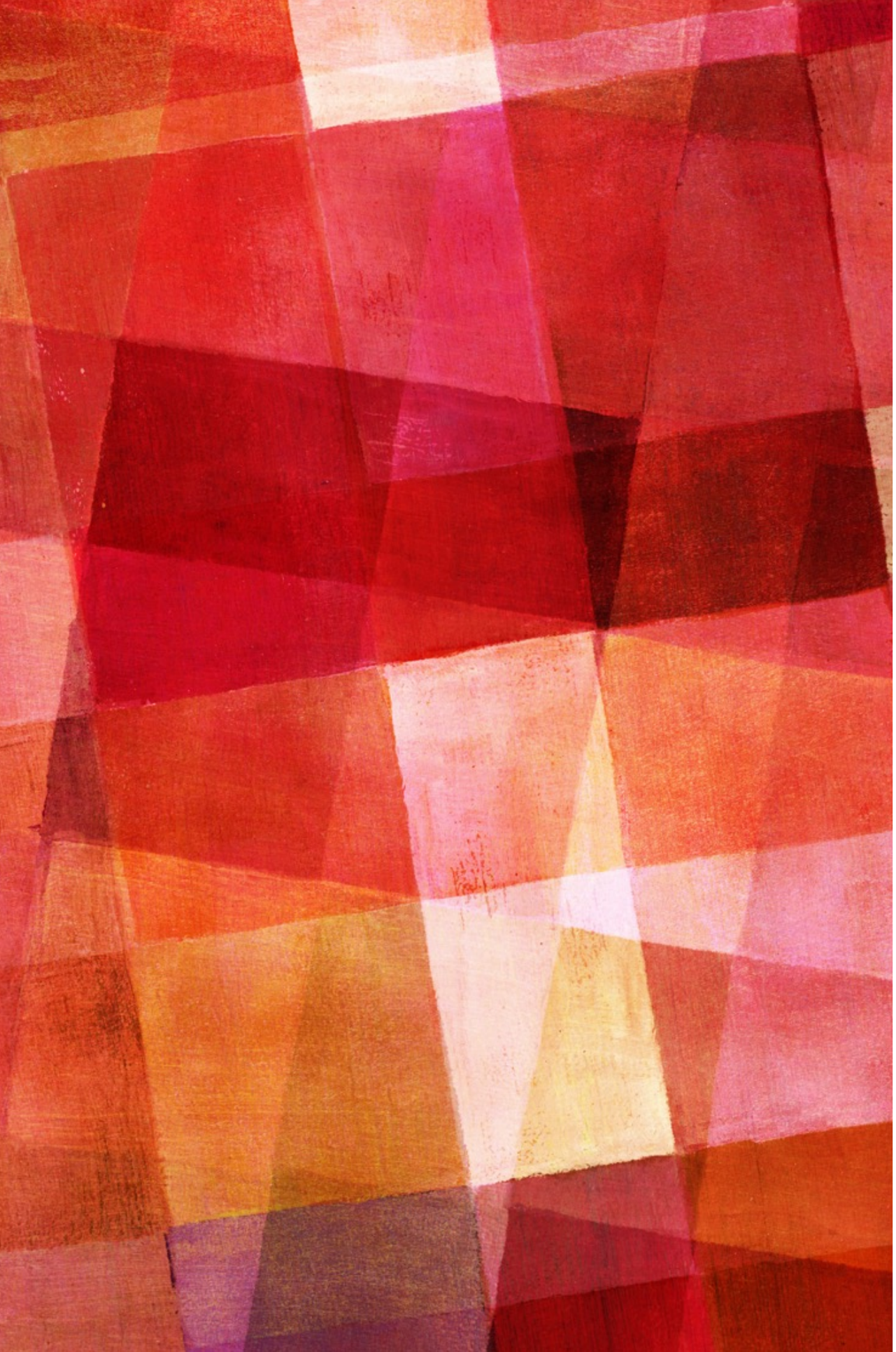
NFS/RDMA BASICS

2017 Westford NFS Bake-a-thon – Part Three



CODE ORGANIZATION

- Client transport overview
- Server transport overview
- NFSv4.1 backchannel operation

RDMA VERBS

- In the Linux kernel, the RDMA verbs API is provided by a set of function calls and data objects.
  - These work with any RDMA-enabled network fabric
- Verb names start with ib_ :
  - ib_post_send, ib_modify_qp, ib_sge
- RDMA core functionality uses names start with rdma_ :
  - rdma_resolve_addr, rdma_create_qp

UPCALLS

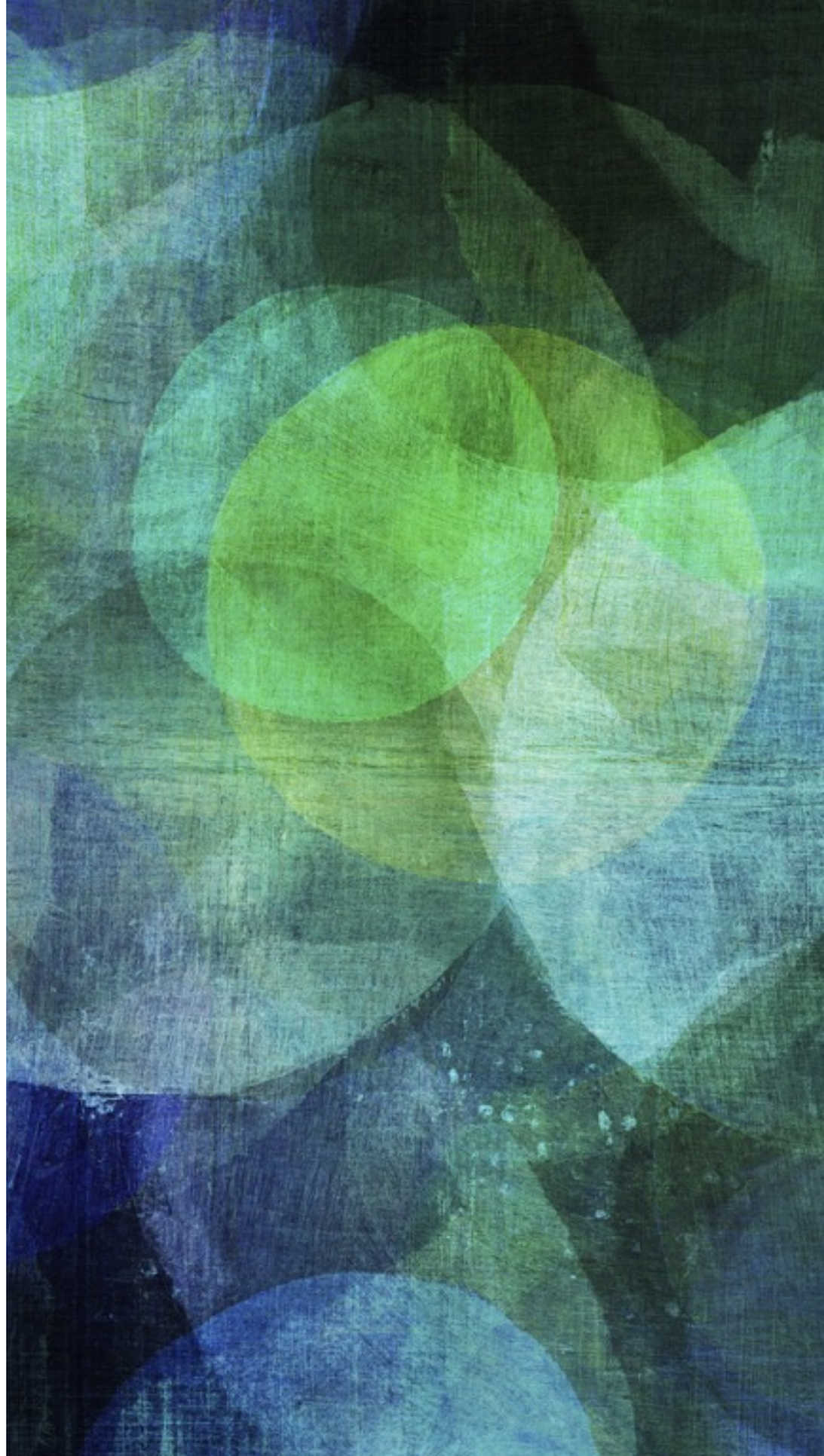
- In the Linux kernel, verbs provider upcalls are used:
  - When a Send or Receive completion fires
  - When reporting a connection-related event
  - When reporting a QP error

- Upcalls may occur in process or soft IRQ context.
  - Posting a Send or Receive WR can be done in either context.

THE RPCRDMA KERNEL MODULE

- Source code is located in net/sunrpc/xprtrdma:
  - Server code is in files names svc_rdma_*
  - Client code is everything else
- Currently one module, rpcrdma.ko, contains both the client and server transports.

CLIENT TRANSPORT OVERVIEW



CLIENT TRANSPORT SWITCH

- transport.c

Method	Purpose
reserve_xprt	Take write lock
release_xprt	Release write lock
connect	Establish a connection
close	Close a connection
buf_alloc	Allocate buffers for Call and Reply
buf_free	Release buffers
send_request	Send an RPC Call
timer	An RPC timeout occurred

MARSHALING RPC CALLS

- rpc_rdma.c
  - Main entry point is rpcrdma_marshal_req.
  - Decides whether to send each RPC Call inline, use scatter-gather, reduce data items, or use special chunks.
    - Uses NFS XDR reply size information
- Chunk lists are constructed and memory is registered.
- The Transport Header is built in a separate buffer, then this buffer plus the buffer containing the RPC message are Sent together.

HANDLING RPC CALLS WITH DATA PAYLOADS

- NFS sets a flag in the xdr_buf to indicate when the NFS operation is allowed to use a Read chunk. The xdr_buf's page list contains the data payload.
  - If the RPC Call is smaller than the inline threshold, the data buffer is made part of the Send message, using the Send WR's scatter-gather list.
  - If the RPC Call is large, the data buffer is registered as a Read chunk and added to the Read list.
- If the RPC Call is large and no Read chunk is allowed, the whole message is registered and added to the Read list as a Position Zero Read chunk.

PREPARING FOR RPC REPLIES WITH DATA PAYLOADS

- NFS sets a flag in the xdr_buf to indicate when the NFS operation is allowed to use a Write chunk. The xdr_buf's page list contains the data buffer.
  - If the estimated maximum size of the RPC Reply is smaller than the inline threshold, no additional action is needed.
  - If the estimated maximum size of the RPC Reply is large, the data buffer is registered as a Write chunk and added to the Write list.
- If the estimated maximum size of the RPC Reply is large and no Write chunk is allowed, a Reply chunk is registered and added to the Transport Header.

HANDLING REPLIES

- rpc_rdma.c
  - Receive upcall runs in soft IRQ context
    - DMA sync, process the credits field, queue work
  - rpcrdma_reply_handler runs in workqueue context
    - Fully parses the transport header
    - Invalidates and DMA unmaps memory associated with request
    - Pulls up and reconstructs the RPC Reply xdr_buf
    - Invokes xprt_complete_rqst

GENERIC RDMA HELPERS

- verbs.c
  - Send, Receive, QP error, and connect upcalls
  - Transport set up and tear-down
  - A *regbuf* is a memory region with an lkey and DMA mapping state
    - Registered for local access only
    - Used internally by the transport for RPC buffers
- Helpers for posting Send and Receive WRs

MEMORY REGISTRATION OPS

- Specific methods for performing memory registration and invalidation on memory that belongs to the upper layer

Method	Purpose
map	Register an MR
unmap_sync	Invalidate all MRs for an RPC
unmap_safe	Invalidate or recover all MRs for an RPC
recover_mr	Recover one MR
open	Compute registration parameters
maxpages	Return maximum pages per MR
init_mr	Prepare one MR for use by the transport
release_mr	Release MR before transport destruction

FRWR MEMORY REGISTRATION

- frwr_ops.c
  - Registering memory for one RDMA segment:
    - DMA map the region then post a FastReg WR to register it
      - WR is not signaled
- Invalidating memory for one RPC:
  - Post LocalInv WRs for all registered MRs
  - Wait for completion
  - DMA unmap all MRs

FMR MEMORY REGISTRATION

- fmr_ops.c
  - Registering memory for one RDMA segment:
    - DMA map the region
    - Use ib_map_phys_mr to register it
  - Invalidating memory for one RPC:
    - Build a list of all MRs
    - Use ib_unmap_fmr to invalidate them
    - DMA unmap all MRs

THE CONNECT WORKER

- transport.c and verbs.c
  - IP address is resolved to a GID/LID (native address)
  - Connecting a transport is serialized with sending RPC Calls
  - Connect worker also handles device unload events
  - Registered memory has to be “re-registered” after a reconnect
  - DMA mapped regbufs have to be remapped after a device unload

INTERESTING DATA STRUCTURES

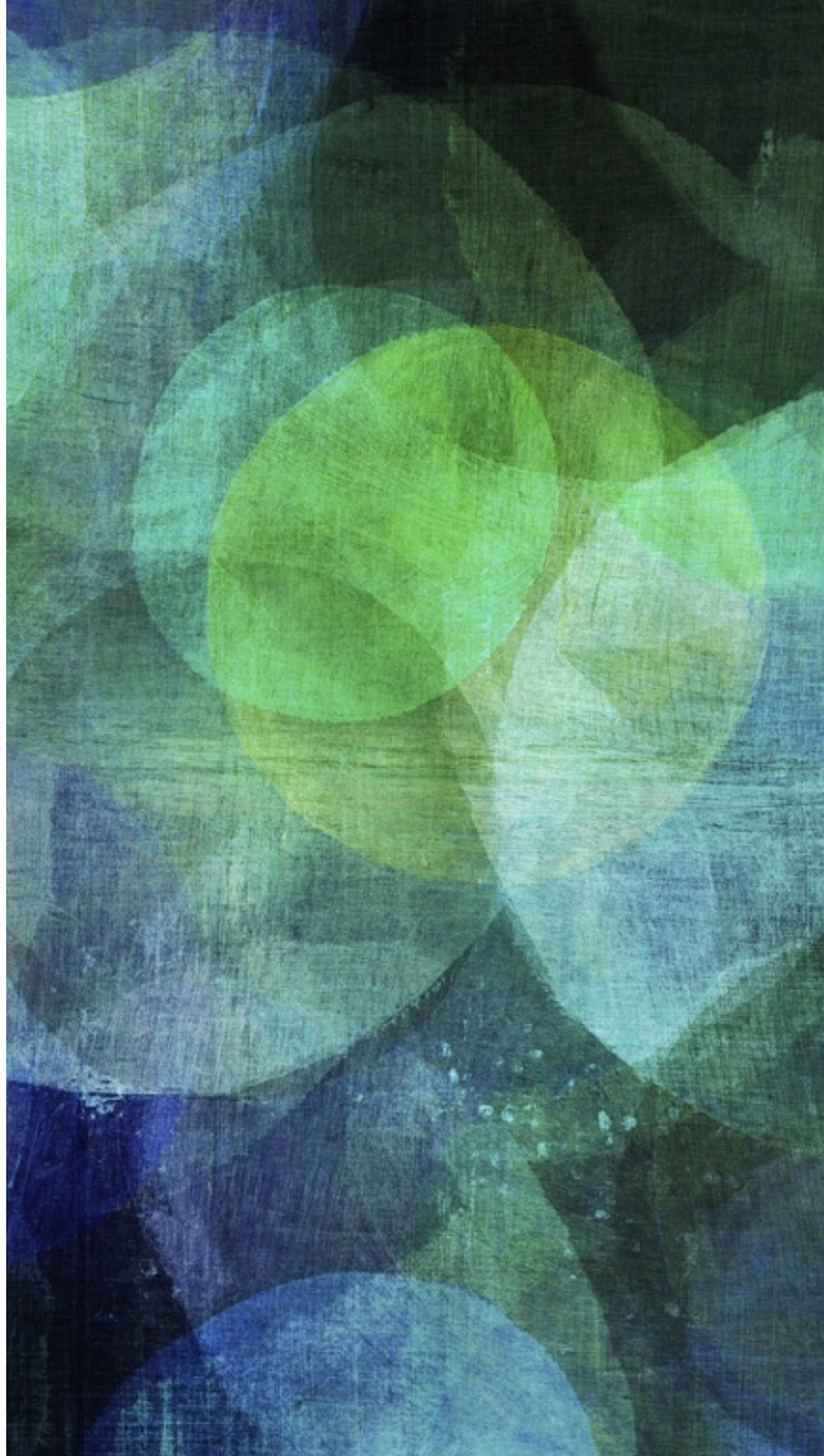
- xprt_rdma.h
  - rpcrdma_req – contains per-request state
  - rpcrdma_rep – state of a reply
  - rpcrdma_mw – state of one memory region
  - rpcrdma_xprt – per-transport state
  - rpcrdma_regbuf – internal buffer with DMA mapping state
  - rpcrdma_buffer – set of reqs and reps for one connection

LOCKING

- Implicit serialization
  - The RPC client serializes calls to ->send_request, ->connect, and ->close
  - The provider serializes calls to completion handlers

Spin lock	Protected list
rb_lock	rb_send_bufs, rb_recv_bufs, rb_pending
rb_mwlock	rb_mws, rb_all
rb_reqslock	rb_allreqs
rb_recoverylock	rb_stale_mrs

SERVER TRANSPORT OVERVIEW



SERVER TRANSPORT SWITCH

- svc_rdma_recvfrom.c
  - svc_rdma_recvfrom – called by an nfsd thread to receive an RPC message from a client and assemble it into an xdr_buf. Dequeues complete Receives, initiates RDMA Reads, dequeues complete Reads.
- svc_rdma_sendto.c
  - svc_rdma_sendto – called by an nfsd thread to send an RPC message in an xdr_buf to a client. Initiates RDMA Writes and Sends.
- RDMA Read and Write WRs are scheduled in svc_rdma_rw.c

ACCEPTING CONNECTIONS

- transport.c
  - Sets up a listener QP
  - New connections accepted in svc_rdma_accept, which allocates fixed per-connection resources
- Some completion upcall handlers live in this file
- And one helper that posts Send operations

INTERESTING DATA STRUCTURES

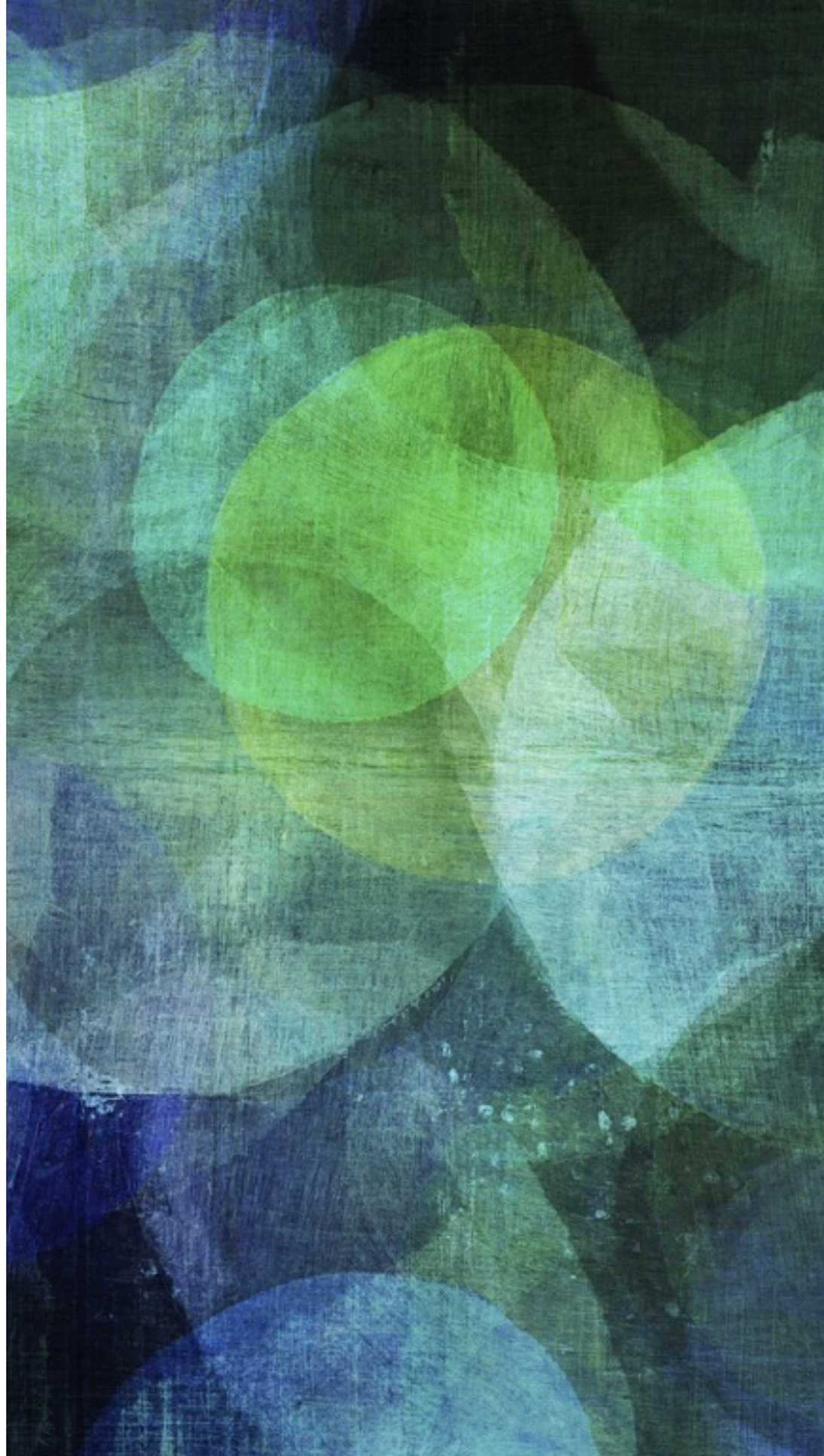
- svcrdma_xprt – per-connection state
- svc_rdma_op_ctxt – state for each Send and each Receive.
- svc_rdma_rw_ctxt – state for a set of RDMA Reads or Writes.
- svc_rdma_chunk_ctxt – completion context for one Read or Write chunk.
- svc_rdma_write_info – state for one Write chunk
- svc_rdma_read_info – state for one Read chunk

LOCKING

- Upper Layer locking
  - A per-transport mutex serializes calls to ->sendto
  - Everything runs in a kthread or workqueue except handle_connect_req

Spin lock	Protected list
sc_rq_dto_lock	sc_read_complete_q, sc_rd_dto_q
sc_ctxt_lock	sc_ctxts
sc_rw_ctxt_lock	sc_rw_ctxts
sc_lock	sc_accept_q

NFSV4.1 BACK CHANNEL OPERATION



NFSV4.1 BACKCHANNEL

- svc_rdma_backchannel.c
  - Plugs into client transport switch
  - Sends CB Calls from the server, handles CB Replies

- backchannel.c
  - Plugs into RPC server framework
  - Handles incoming CB Calls on the client, sends CB Replies

LOCKING

Spin lock	Protected list
bc_pa_lock	bc_pa_list
rb_reqslock	rb_allreqs