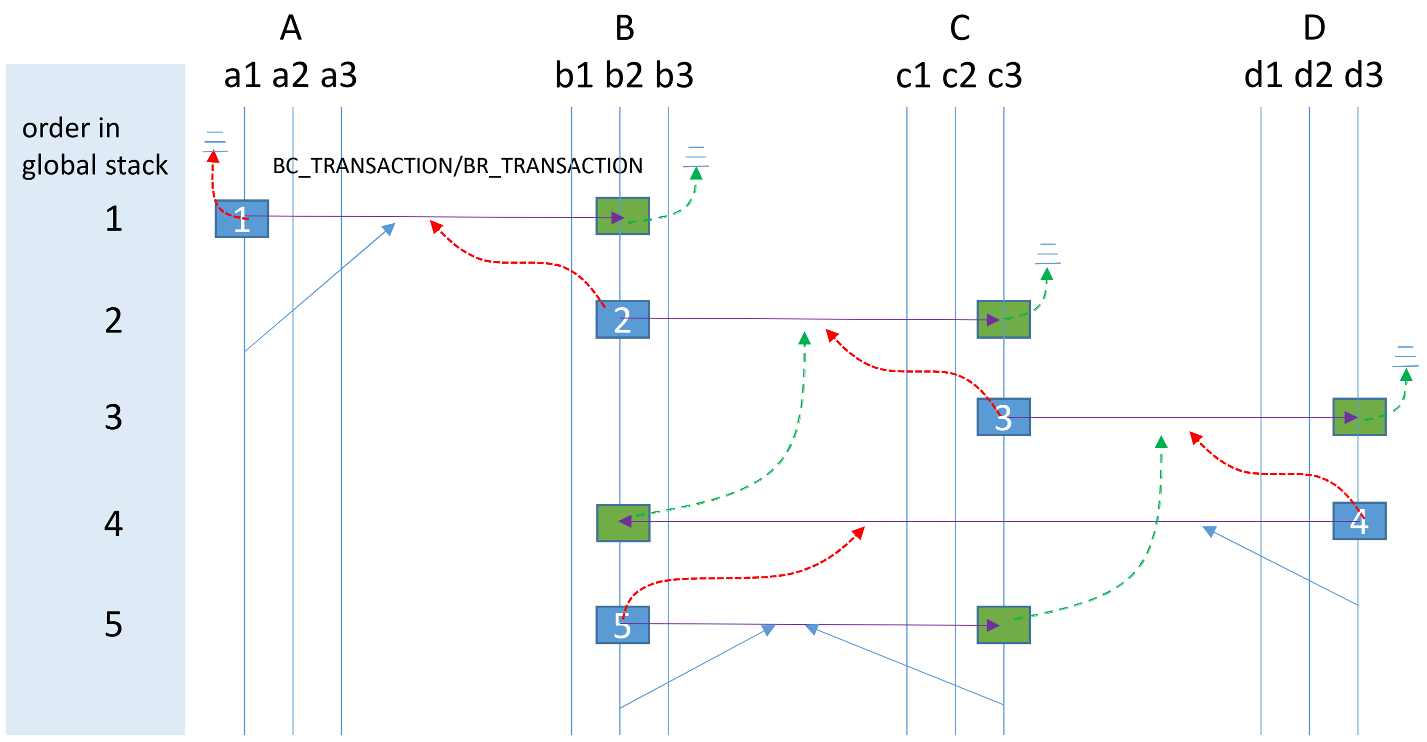
binder_transaction

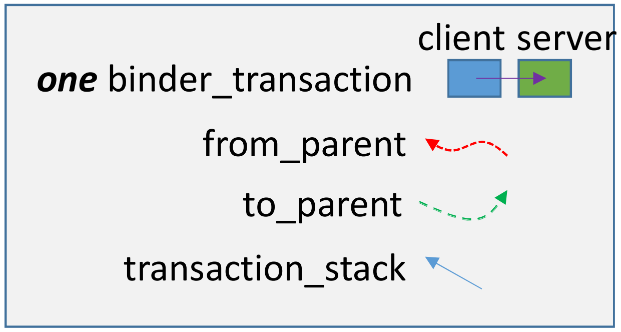
- Structure binder_transaction records the data of a transaction between a client and a server thread
  - Shared between the client and the server (all in kernel space), generated by BC_TRANSACTION processing, and passed to BR_TRANSACTION processing
  - BC_REPLY and BR_REPLY for the same transaction consume the data (destructing it)
  - binder_transaction behaves like a stack frame for local function call
- In order to serve its client, a server may initiate a new transaction
  - The server has to receive result from the new transaction, before it can return to its client. That is, A calls B, B calls C; C replies to B, B replies to A.
  - In this case, there are two binder_transactions (A->B and B->C) in the system that are serving a single task. They behave like a ”global call stack” for A->B->C.
  - A single task can have many binder_transactions for service calls across processes. They are chained together via its from_parent pointer, forming the “global call stack”.
    - One binder_transaction is a stack frame
    - The system can find all the involved threads by traversing the global call stack from stack top

transaction_stack

- All the binder_transactions to and from a same thread are linked together, forming a local transaction stack.
  - Newer transaction of the thread stay higher in the stack.
  - binder_thread->transaction_stack is the top pointer to the local task stack.
  - Since a transaction is shared by both client and server threads, it is always in the stacks of both client and server.
  - Not a real stack, because two consecutive elements do not have direct “call” relation. It is only a data structure to maintain the transactions of a thread.
- When a service call in a global task call chain comes to a process that is already in the call chain (i.e., a thread T is processing a transaction in the call chain), the system finds the thread T by traversing the call chain,
  - Use that same thread T to serve the newly incoming service call
  - This makes the global call chain A (in T)->B->...->X(in T) look like a local call A->X in T
  - Now the transaction_stack for T looks like a real stack

transaction_stack illustration





- A new transaction's from_parent points to the client thread transaction stack
- The new transaction's to_parent points to the server's thread transaction stack
- In this way, the new transaction is pushed into the stacks of both client and server

Traverse all transaction stack:

```
tmp = thread.transaction_stack
while (tmp != null){
    tmp = tmp->from_parent
}
```

Traverse thread's transaction stack:

```
tmp = thread.transaction_stack
while (tmp != null){
    if( tmp->to_thread == thread)
        tmp = tmp->to_parent
    else
        tmp = tmp->from_parent
}
```