Understanding Binder in Android

Haifeng Li

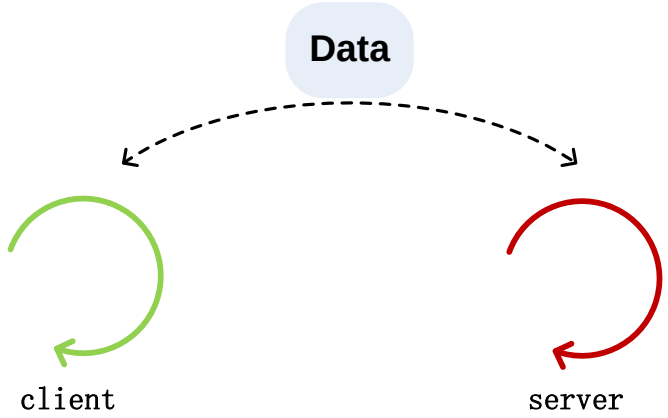
2014-9-2

Outline

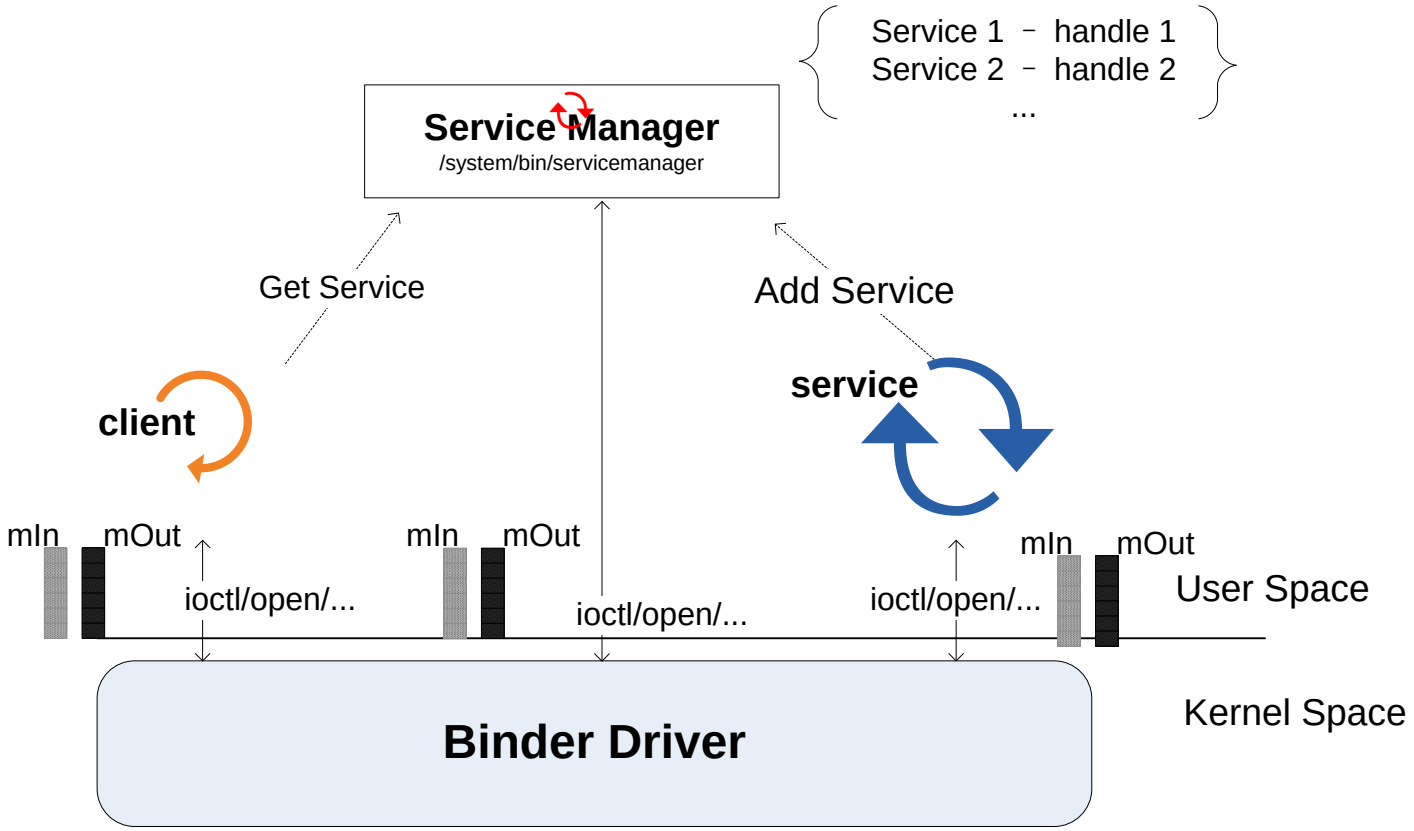
- Background
  - What is Binder
  - Binder Communication Model
  - Terminology
  - Binder Software Stack
- Client(user space)
- Binder driver (Kernel Space)
- Service(user space)

Background

- What is Binder?
  - An Inter-process communication system for developing object-oriented OS services.

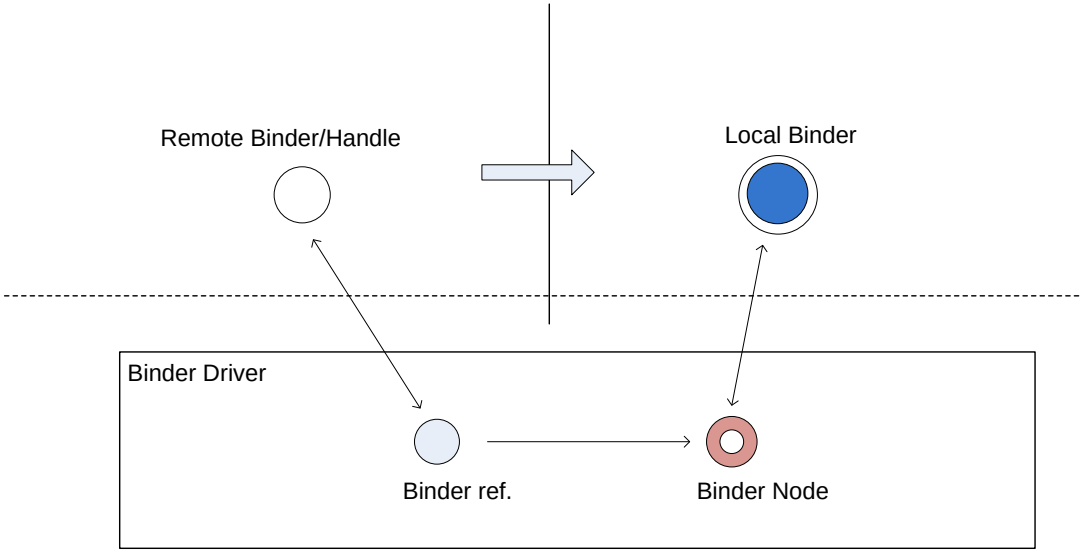


Binder Communication Model

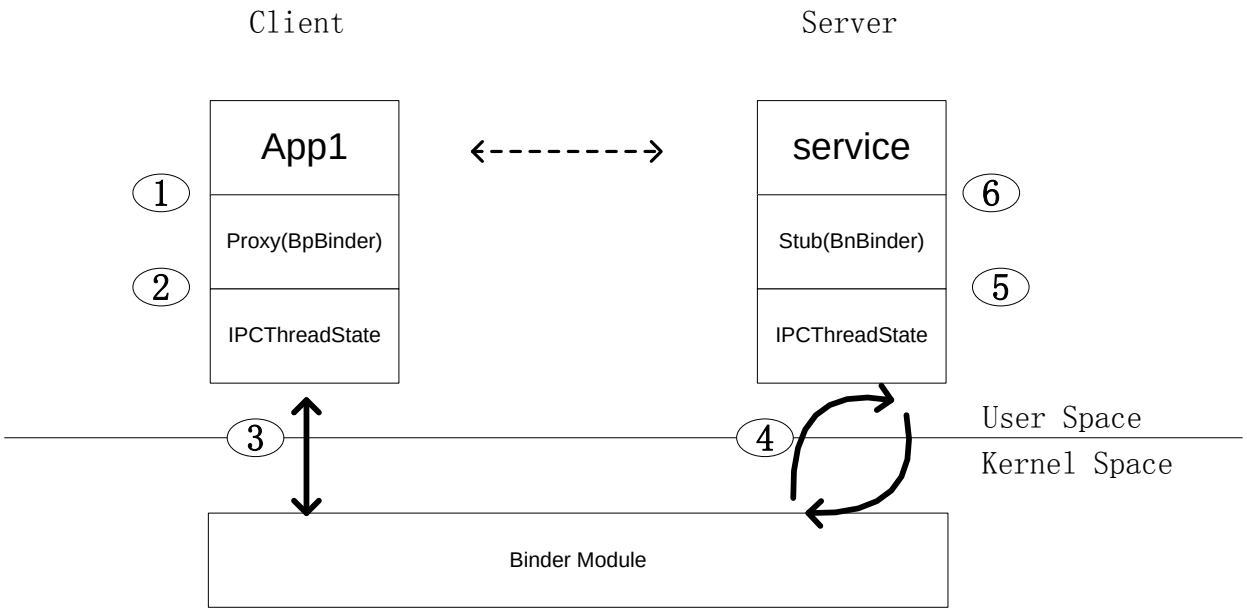


Terminology

- Service Handle
- Remote Binder
- Local Binder
- Binder node
- Binder reference
- Service Manager(Context Manager)
  - It’s handle is 0 in all binder client and server.



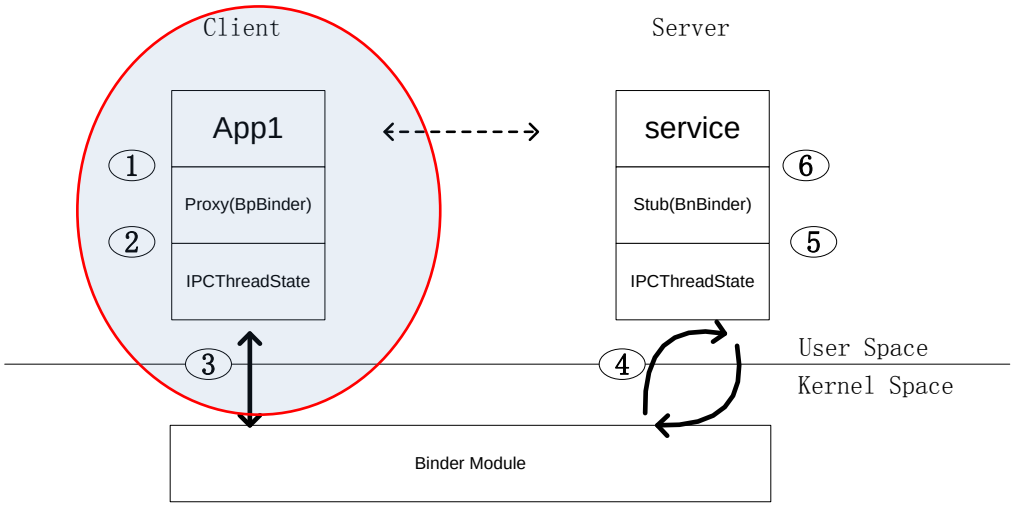
IPC Software Stack



1. BpBinder(n) -> transact(OP, data, &reply)
2. IPCThreadState::transact(handle, OP, data, reply)
3. ioctl(binder_fd, BINDER_OPERATION, &bwr)
4. IPCThreadState::getAndExecuteCommand()
5. Bnxxx::onTransact(OP, data, reply)

Client(user space)

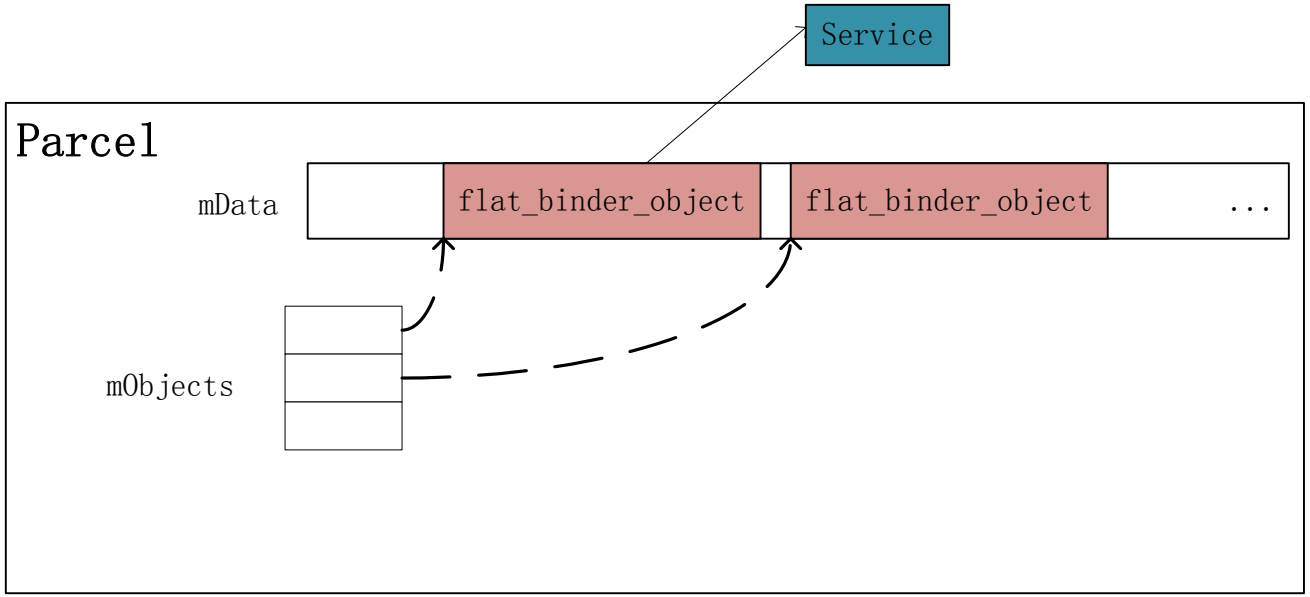
- Initialization
  - Call system call open(), which is binder_open in kernel. Open /dev/binder file and get a file description. Create some key data structures.
  - mmap 1MB-8KB virtual space for data transaction by binder_mmap in kernel.
- Get handle of service from service manager
- Sent request to Service by BpBinder(*handle*)
- Data transact to kernel by IPCThreadState.



Data Transaction in Client(1)

- Package in Client

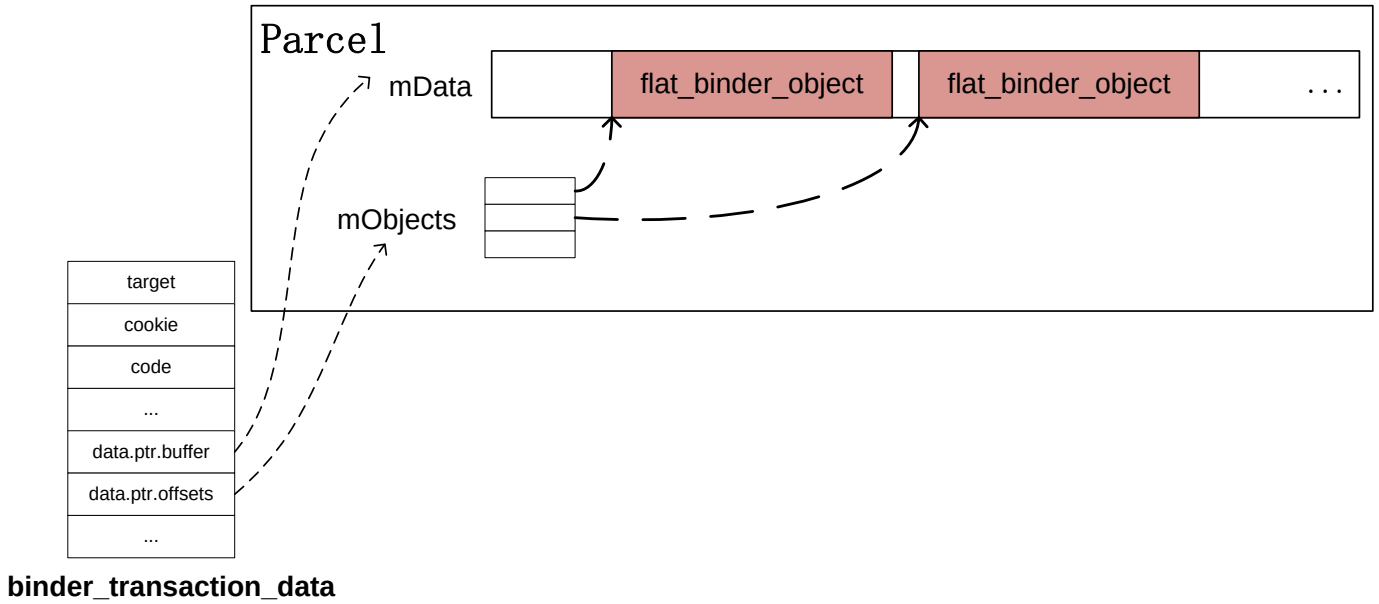
```
102     virtual void Client::Foo(int32_t push_data) {
103         Parcel data, reply;
104         data.writeInterfaceToken(IDemo::getInterfaceDescriptor());
105         data.writeInt32(push_data);//writeStrongBinder(service)
109
110         remote()->transact(OP, data, &reply);
```



Data Transaction in Client(2)

- Packaged to binder_transaction_data

IPCThreadState::writeTransactionData(int32_t cmd, …, int32_t handle, uint32_t code, const Parcel& data…)

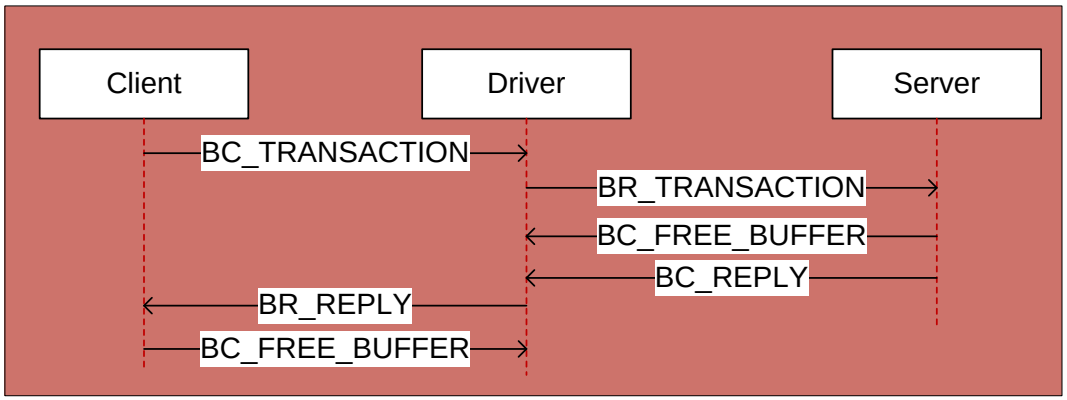


•cmd will add to mData.(BC_TRANSACTION, BC_REPLY,… Binder Command)
•Target: target handle
•Cookie: will be define according to handle in binder driver
•Code: **Op**eration of client.
•Offsets could help the binder driver to process binder object reference.

Data Transaction in Client(3)

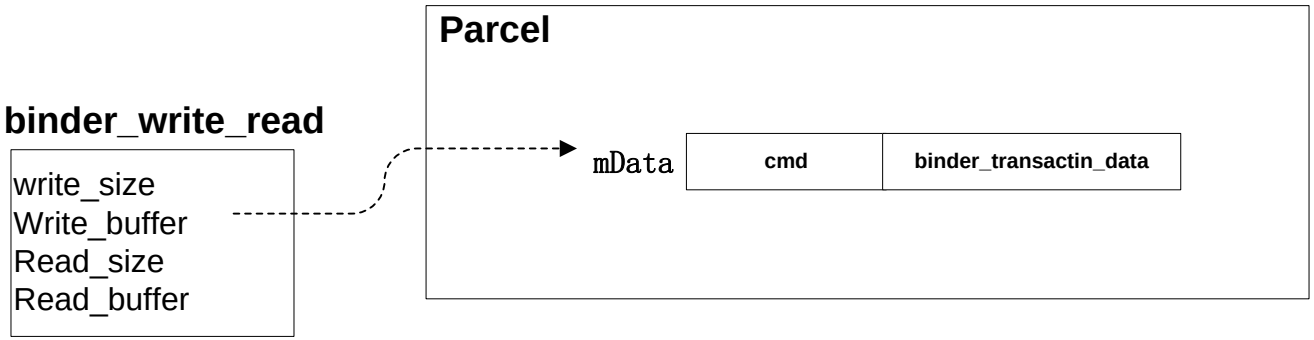
- Binder Command(User->Driver)
  - Binder Thread Support: BC_REGISTER_LOOPER, BC_ENTER LOOPER,BC_EXIT_LOOPER
  - Binder Transactions: BC_TRANSACTION, BC_REPLY
  - Further Mechanism: BC_INCREFS, BC_RELEASE , BC_DECREFS, BC_REQUEST_DEATH_NOTIFICIATION, BC_CLEAR_DEATH_NOTIFICATION, BC_DEAD_BINDER_DONE,…

- Binder Return Command (Driver -> User)
  - Binder Thread Support: BR_SPAWN_LOOPER
  - Binder Transactions: BR_TRANSACTION, BR_REPLY
  - Further Mechanism: BR_INCREFS,BR_ACQUIRE, BR_RELEASE , BR_DECREFS, BR_CLEAR_DEATH_NOTIFICATION_DONE, …



Data Transaction in Client(4)

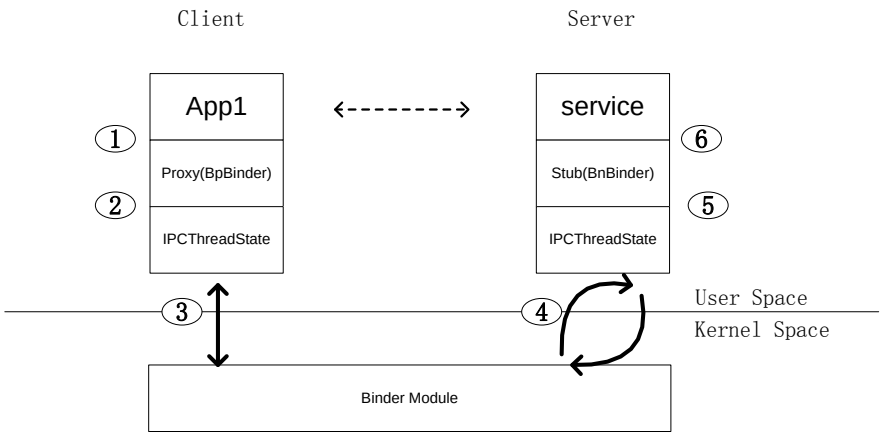
- Repackage to Parcel(mOut)



- Each working thread has two parcel: mOut and mIn. mOut is for write buffer, mIn for read buffer.

Outline

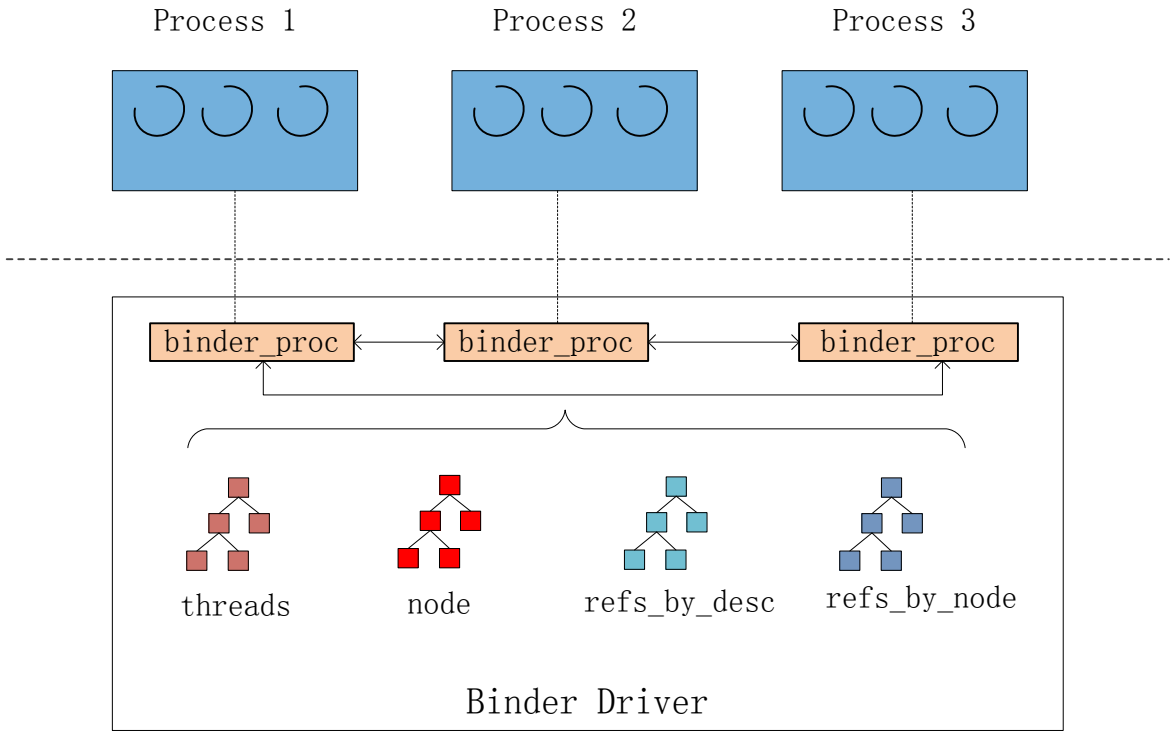
- Background
- Client(user space)
- Binder driver (Kernel Space)
- Service(user space)



Binder Driver:Binder Protocol

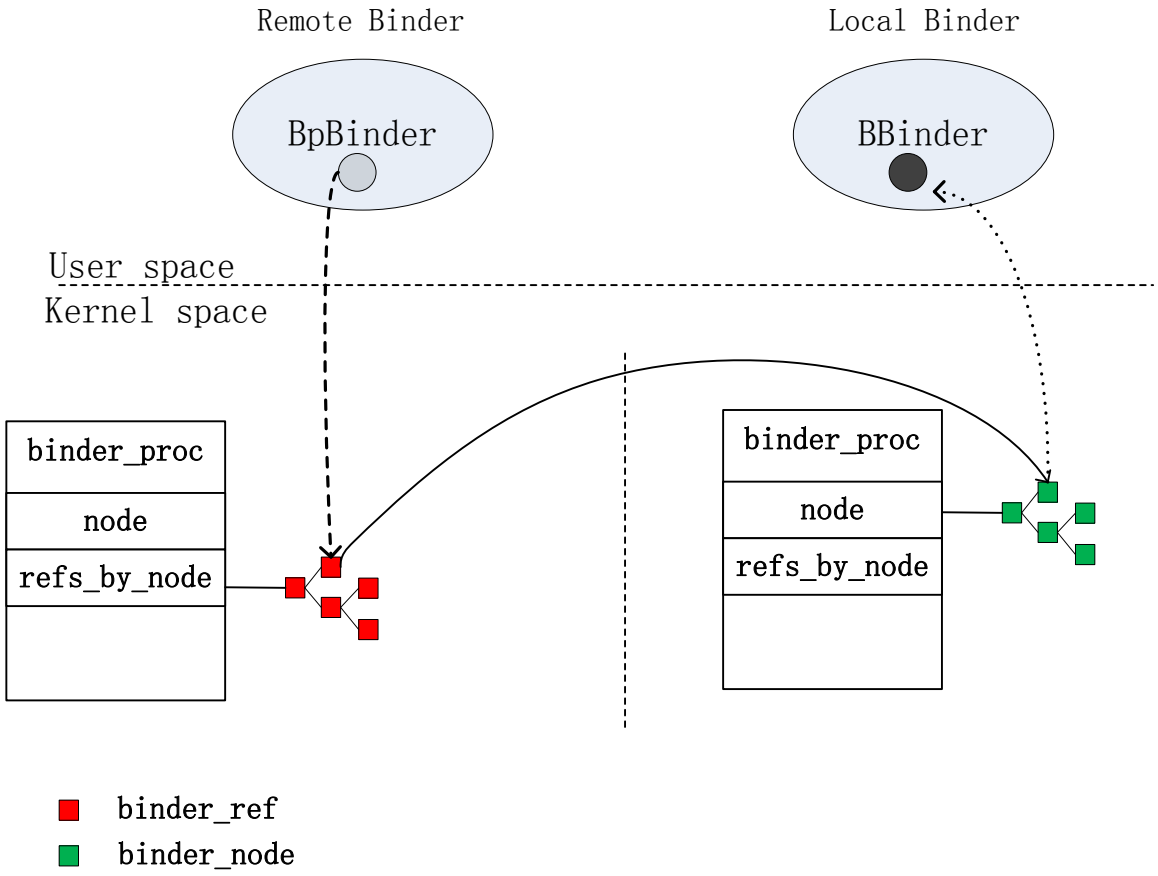
- The protocol used for ioctl() system call.
  - **BINDER_WRITE_READ**
  - BINDER_SET_MAX_THREADS
  - BINDER_SET_CONTEXT_MGR
  - BINDER_THREAD_EXIT
  - BINDER_VERSION
  - BINDER_SET_IDLE_TIMEOUT

Key Data Structure (1)

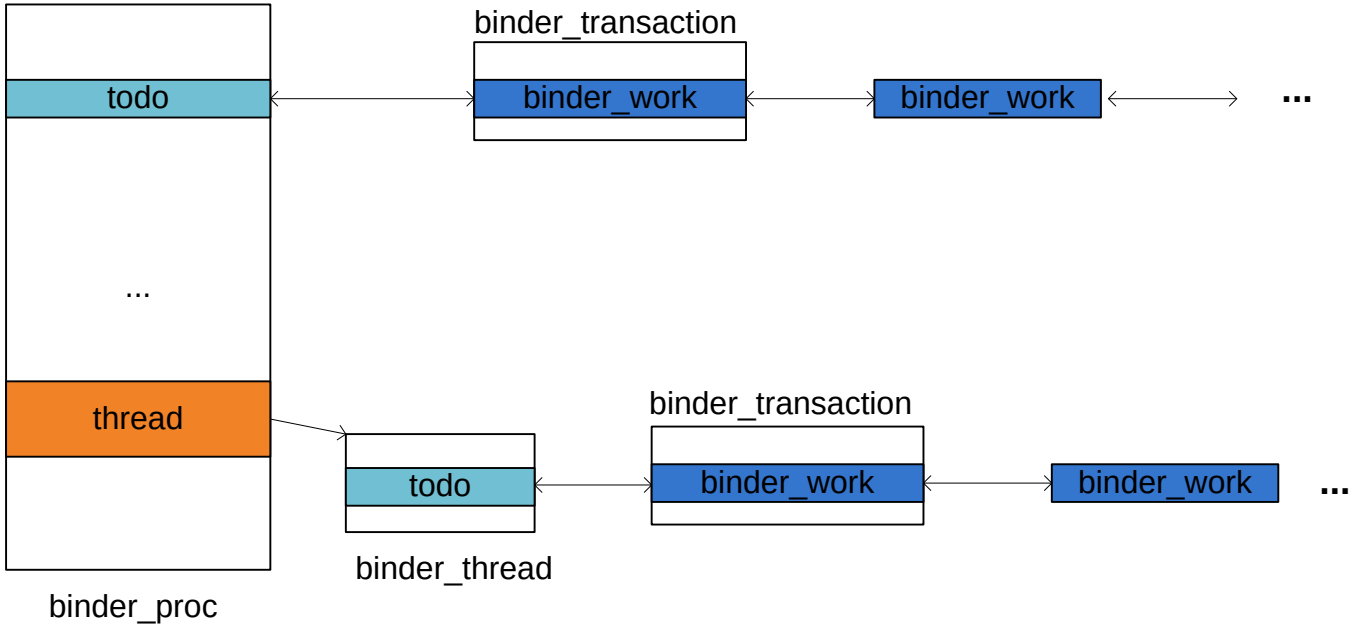


- The binder_proc is mapped to process by 1:1. It is created on binder_open().
- All binder_proc are listed in binder_procs.
- The binder_proc has 4 red-black tree.
- The binder_thread represents a working thread, inserted into threads rbtree.
- The binder_node represents a service, inserted into node rbtree.
- refs_by_desc and refs_by_node represent reference to a proc_node.

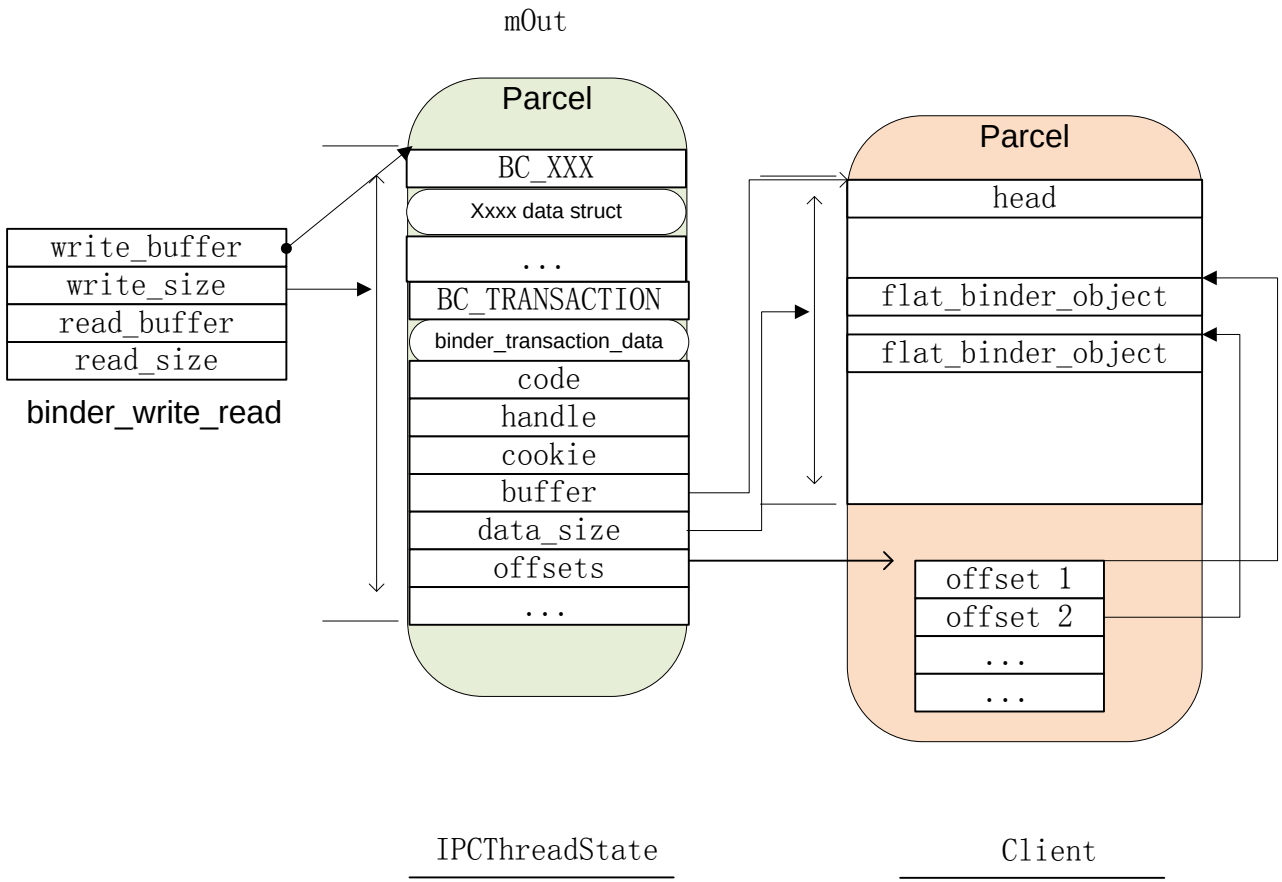
Key Data Structure (2)



Key Data Structure (3)

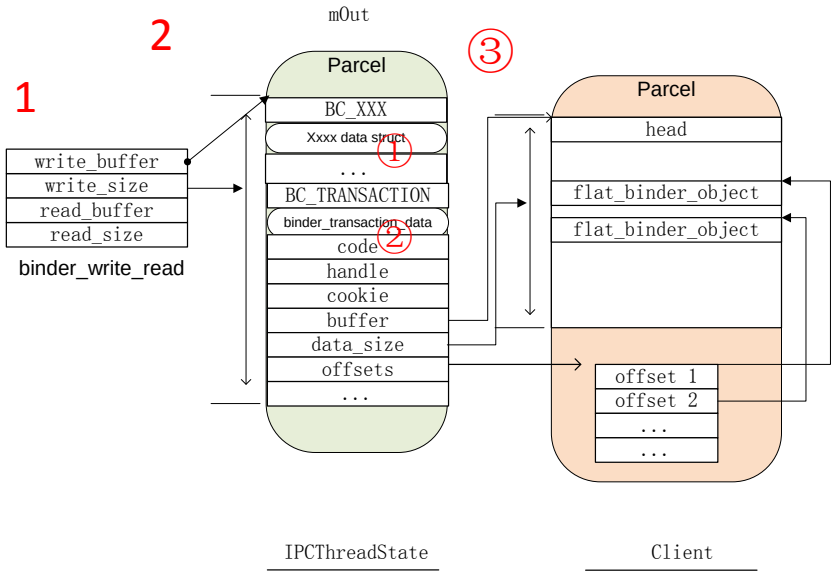


Input Data Format

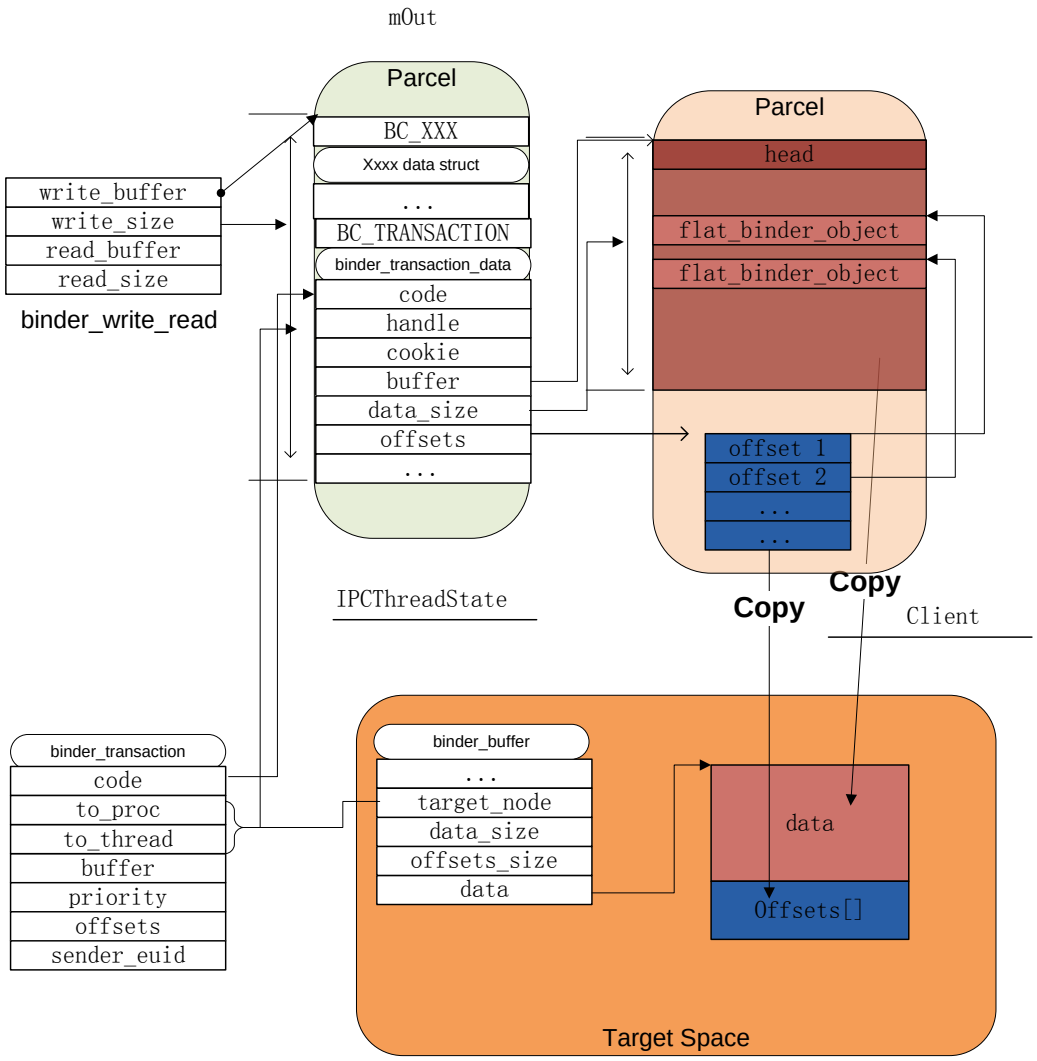


Transaction in Binder – Client (1)

1. Copy binder_write_read from user space
2. Copy *xxx* data to kernel space. Iterate the items:
   - ① Get Binder command from write_buffer(BC_XXX).
   - ② Get target thread/proc/node by handle.
   - ③ Allocate binder_buffer from target space, and copy effective data.
   - ④ Build a session(build_transaction).
   - ⑤ Mount the session to target_thread's todo list.
   - ⑥ Mount a BINDER_WORK_TRANSACTION_COMPLETE binder_work to source thread
   - ⑦ Wake up corresponding thread.

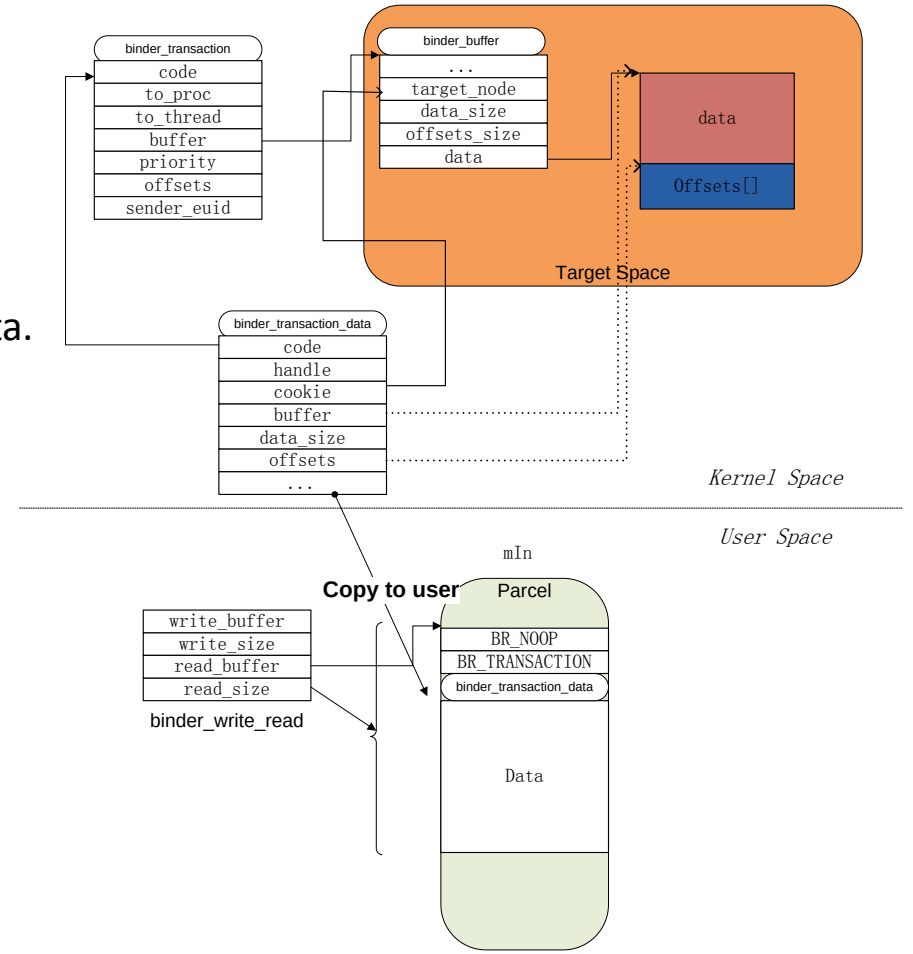


Transaction in Binder –Client (2)



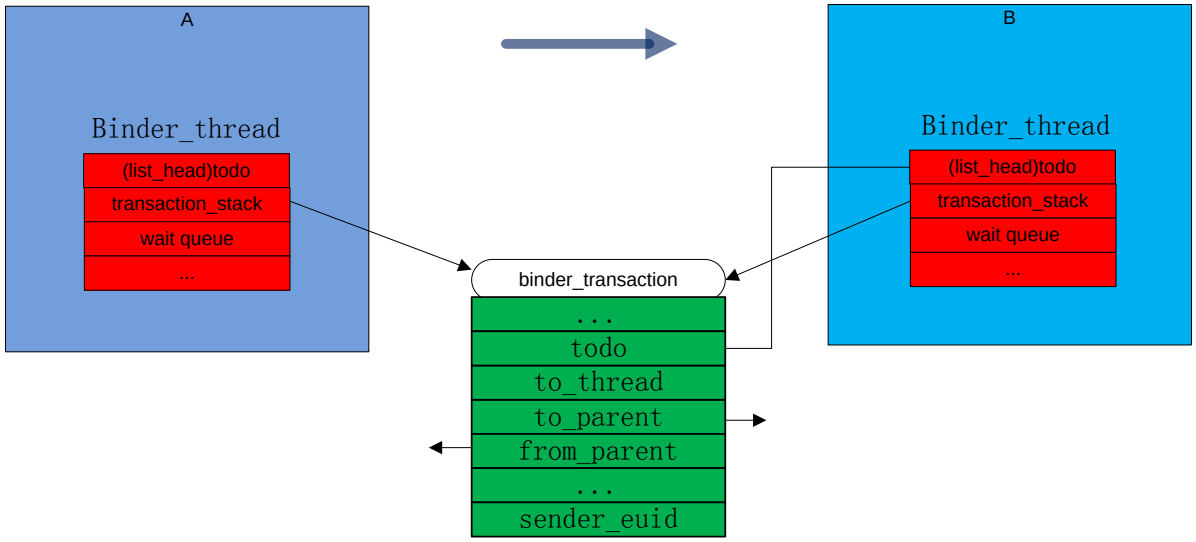
Transaction in Binder -- Server

1. Server thread wake up and get a binder_work(binder_transaction) from todo list
2. Build a binder_transaction_data from binder_transaction
3. Set priority of current thread of client.
4. The buffer and offsets will be virtual address
   - Virtual address = kernel address + address offset
5. Copy binder_transaction_data to mIn
6. Return from kernel
7. IPCThreadState iternate the command and data.
   - For BR_TRANSACTION, call stub's onTransact function, which will call server service function finally.



Transaction stack(1)

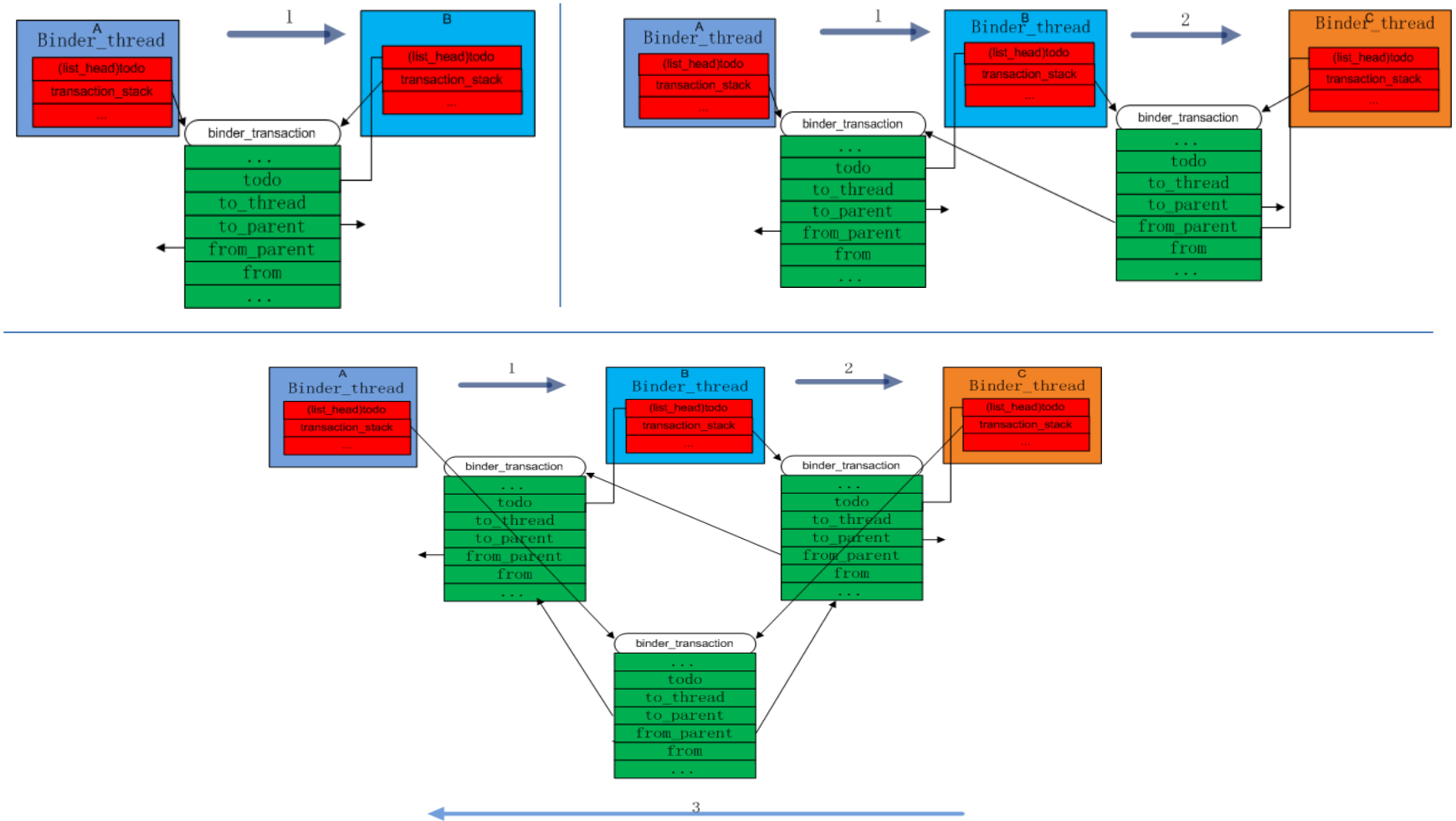
- The stack is for recording transaction session.



- Commonly, to_parent and from_parent is null. But, when the transaction rely on other session, what will happen?
  - The session in B will lost.

Transaction stack(2)

- For example, A -> B, B->C, C->A



Binder Workflow