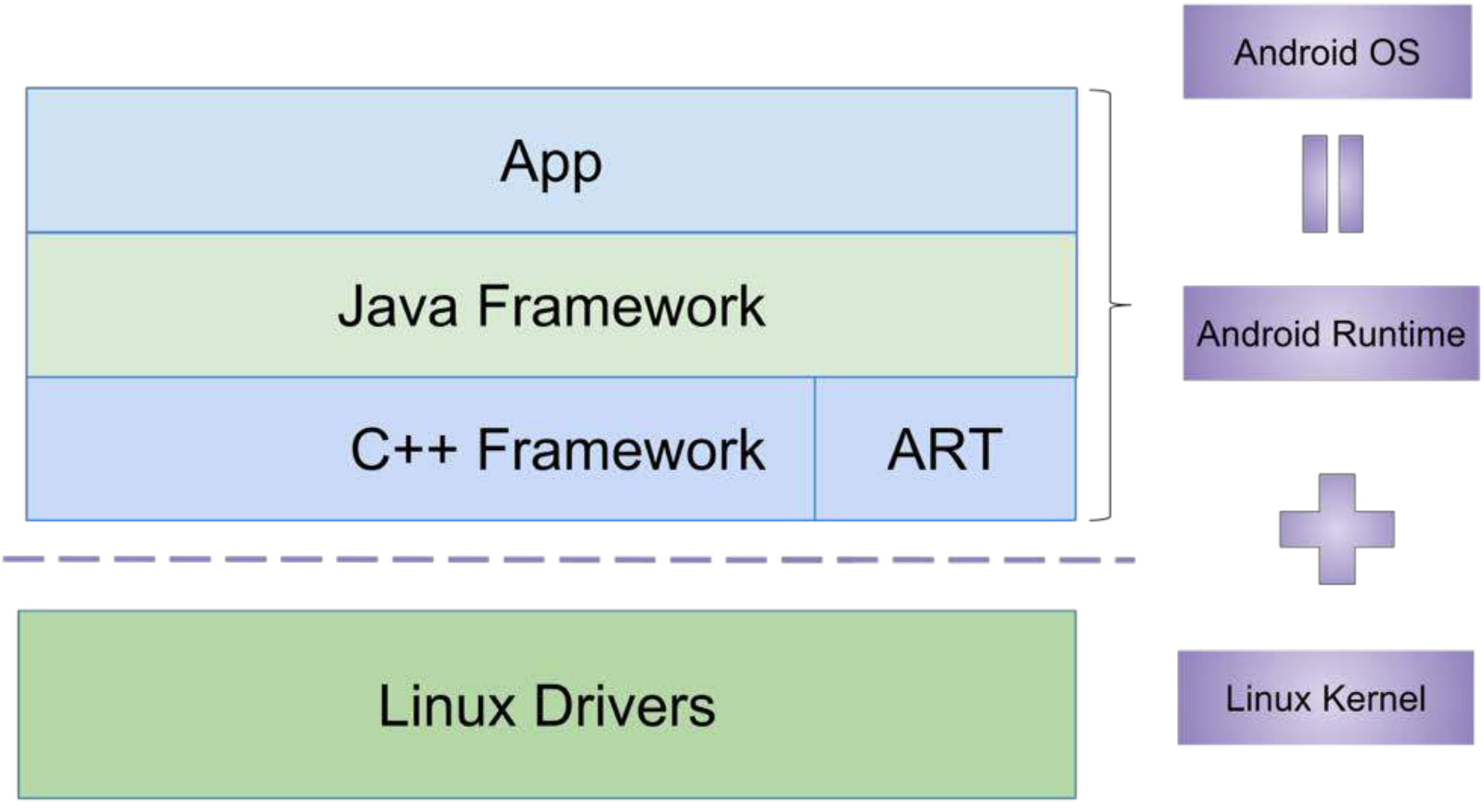
Android Binder设计之道

袁辉辉 GITYUAN


App
Java Framework
C++ Framework
ART
Linux Drivers
Android OS
Android Runtime
Linux Kernel

Linux kernel map

functionalities
layers

user space interfaces

system calls and system files

virtual

bridges

cross-functional modules

logical

functions implementations

device control

hardware interfaces

drivers, registers and interrupts

electronics

human interface

HI char devices

cdev_add
input_fops
snd_fops
video_fops
console_fops
sys_capset
fb_fops
sys_syslog

security

security/
security.h
linux/security.h
security_capset
may_open
security_socket_create
inode_permission
security_inode_create
security_ops
selinux_ops

debugging

sys_ptrace
log_buf
register_kprobe
printk
handle_sysrq
oprofile_start
kgdb_breakpoint
oprofile_init

HI subsystems

oss
alsa
video_device
tty
drivers/
abstract devices and HID class drivers
drivers/input/
drivers/media/
sound/
console
kbd
fb_ops
mousedev
drm_drv

HI peripherals device drivers

uvc_driver
vga_con
ac97_driver
atkbd_drv
psmouse

system

interfaces core

kernel/
System Call Interface
system files
linux/syscalls.h
linux/uaccess.h
copy_from_user
/proc /sysfs /dev
sysfs_ops
sys_ioctl
cdev_add
cdev_map
sys_epoll_create
cdev
sys_reboot
sys_init_module

Device Model

drivers/base/
driver_init
linux/kobject.h
kobject
kset
linux/device.h
bus_register
bus_type
device_create
device
class
device_type
driver_register
device_driver
probe
kvm
load_module
module
module_param
kernel_param
kobject_uevent_init
kobject_uevent

system run

init/
kernel/
boot, shutdown
power management
init/main.c
start_kernel
mount_root
do_initcalls
run_init_process
kernel_restart
kernel_power_off
hibernate
machine_ops

generic HW access

request_region
pci_driver
request_mem_region
pci_register_driver
pci_request_regions
usb_driver
usb_submit_urb
usb_hcd
ioremap

device access and bus drivers

ehci_irq
writew
readw
ehci_urb_enqueue
outw
inw
usb_hcd_irq
pci_read
pci_write

processing

processes

kernel/
processes
fs/exec.c
kernel/signal.c
sys_fork
sys_vfork
sys_clone
sys_execve
sys_kill
sys_signal
do_sigaction
linux/binfmt
sys_futex
sys_gettimeofday
sys_time
sys_times

threads

schedule_work
init_workqueues
init_work
queue_work
work_struct
workqueue_struct
kthread_create
kernel_thread
current
thread_info
do_fork

synchronization

lock_kernel
mutex_lock_interruptible
mutex_unlock
kernel_flag
wait_for_completion
complete
mutex
owner
add_timer
timer_list
run_timer_softirq
down_interruptible
up
semaphore
wait_event
wake_up
msleep
spin_lock_irqsave
spin_unlock_irqrestore

Scheduler

kernel/sched.c
task_struct
schedule_timeout
schedule
setup_timer
process_timeout
activate_task
irq
context_switch

interrupts core

request_irq
irqaction
irqaction_struct
jiffies_64++
softirq_action
do_timer
setup_irq
tick_periodic
timer_interrupt
do_softirq
do_IRQ
irq_desc
softirq_init

CPU specific

setup_arch
x86_init
trap_init
early_trap_init
start_thread
native_init_IRQ
switch_to
/proc/interrupts
set_intr_gate
system_call
show_regs
interrupt
pt_regs
atomic_t
cli
sti

memory

memory access

mm/
sys_brk
sys_mmap
shm_vm_ops
sys_shmctl
sys_shmat
sys_sysinfo
si_meminfo
sys_mprotect
sys_mincore
/dev/mem
mem_fops
mmap_mem
/proc/meminfo
/proc/self/maps
sys_msync

virtual memory

vmalloc_init
find_vma_prepare
vmalloc
vfree
vmlist
vm_struct
virt_to_page

memory mapping

mm/mmap.c
do_mmap
do_mmap_pgoff
kmem_cache_alloc
vma_link
mm_struct
vm_area_struct

logical memory

physically mapped memory
mm_init
kmalloc
kfree
pgd_t
pmd_t
pte_t

Page Allocator

/proc/slabinfo
mm/slab.c
mm/slub.c
mm/slab.h
kmem_cache
__free_pages
kmem_cache_init
kmem_cache_alloc
__free_one_page
__get_free_pages
page
__alloc_pages
vm_stat
totalram_pages
try_to_free_pages

physical memory operations

arch/x86/mm/
mem_init
get_page_from_freelist
zonelist
zone
free_area
free_list
out_of_memory
oom
num_physpages
do_page_fault

storage

files & directories access

fs/
sys_open
sys_read
sys_write
sys_lseek
sys_pipe
sys_select
sys_flock
sys_splice
sys_sendfile
pipe_fs_i.h
pipe_inode_info
alloc_file
sys_chown
sys_chmod
notify_change
sys_newstat
sys_fsync
inode_setattr
sys_mount
sys_symlink
sys_sync

Virtual File System

file
vfs_fsync
file_systems
vfs_read
vfs_stat
vfs_write
vfs_getattr
vfs_create
inode
fsnotify
inode_operations
file_operations
file_system_type
get_sb
super_block
ipc_ipcs_struct
smb_fs_i.h

page cache

address_space
bdi_writeback_thread
do_writepages

swap

si_swapinfo
swap_info
kswapd
do_swap_page
wakeup_kswapd

networking storage

nfs_file_operations
smb_fs_type
cifs_file_ops
iscsi_tcp_transport

logical file systems

ext4_file_operations
ext4_get_sb
ext4_readdir

block devices

block/
gendisk
block_device_operations
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver
usb_stor_host_template

disk controller drivers

scsi_host_alloc
libata
Scsi_Host
ahci_pci_driver
ata3xx_init

networking

sockets access

net/
sys_socketcall
sys_connect
sys_accept
sys_bind
sys_listen
sys_sendmsg
sys_recvmsg
sys_setsockopt
sys_socket
/proc/net/
tcp4_seq_show
tg_proc_tcp_show_tbl
rt_cache_seq_show
sock_ioctl

protocol families

inet_init
__sock_create
socket
inet_family_ops
inet_create
unix_family_ops
proto_ops
inet_dgram_ops
inet_stream_ops
socket_file_ops

socket splice

sock_sendpage
tcp_sendpage
udp_sendpage
sock_splice_read
tcp_splice_read

protocols

proto
/proc/net/protocols
udp_prot
tcp_prot
udp_sendmsg
tcp_sendmsg
tcp_v4_rcv
udp_rcv
tcp_transmit_skb
NF_HOOK
nf_hook()
ip_queue_xmit
ip_local_deliver
ip_rcv_finish
ip_route_input
alloc_skb
ip_rcv
sk_buff
ip_output

network interface

linux/netdevice.h
netif_receive_skb
dev_queue_xmit
net_rx
register_netdev
net_device
alloc_netdev_mq
ieee80211_alloc_hw
ether_setup
ieee80211_rx
netif_carrier_on
ieee80211_xmit
drivers/net/

network device drivers

usbnet_probe
pw2100_pci_init_one
zd1201_probe
e1000_xmit_frame
e1000_intr

electronics

user peripherals

keyboard
camera
mouse
graphics card
audio

I/O

I/O mem
I/O
PCI controller
I/O ports
ACPI
USB controller

CPU

registers
APIC
interrupt controller

memory

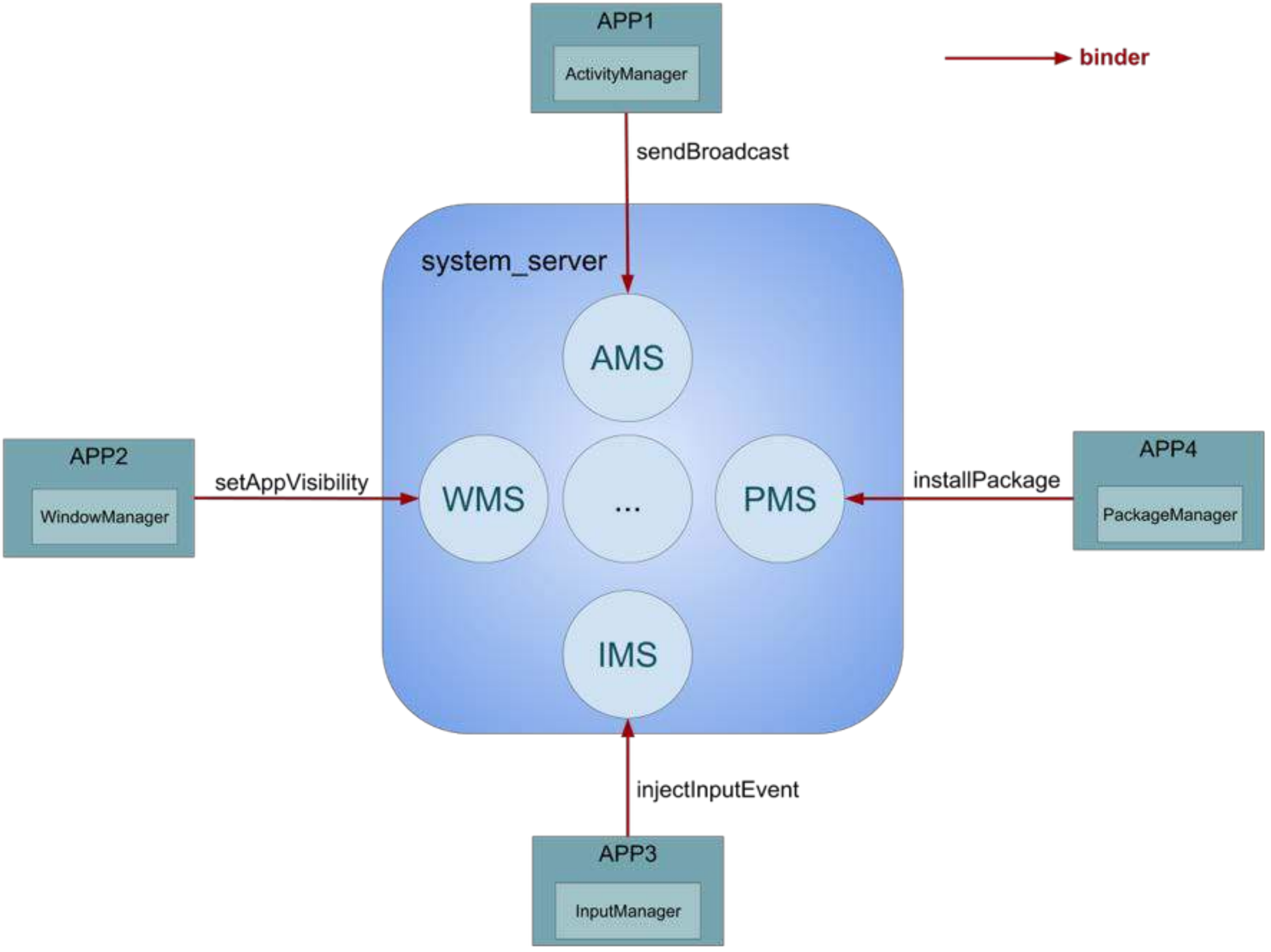
RAM
DMA
MMU

disk controllers

SCSI
SATA

network controllers

Ethernet
WiFi


APP1
ActivityManager
binder
sendBroadcast
system_server
AMS
WMS
...
PMS
IMS
APP2
WindowManager
setAppVisibility
installPackage
APP4
PackageManager
injectInputEvent
APP3
InputManager

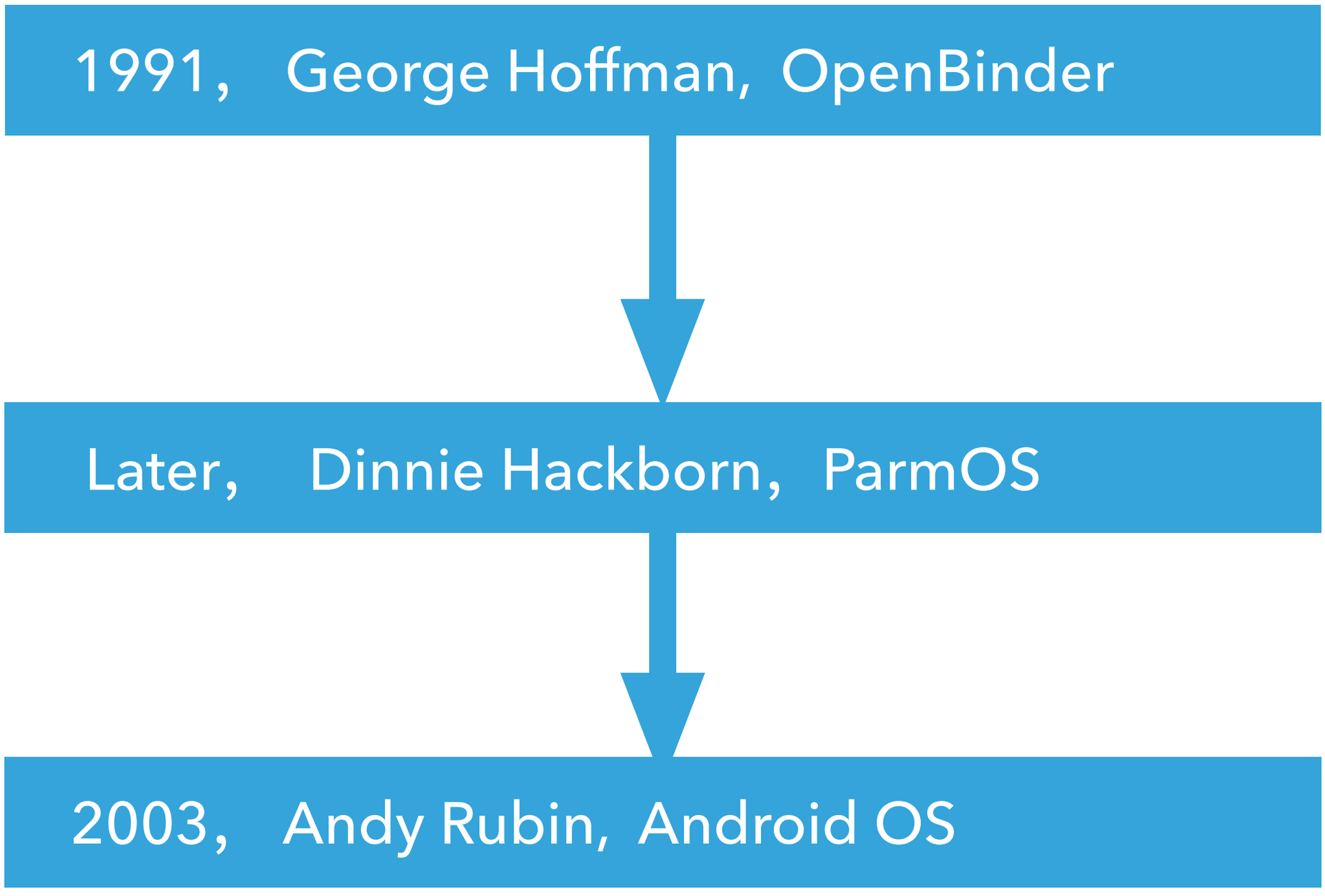
提纲

- Binder概述
- Binder设计思想
- Binder通信协议
- Binder死亡通知
- Binder线程池管理

提纲

- Binder概述
- Binder设计思想
- Binder通信协议
- Binder死亡通知
- Binder线程池管理

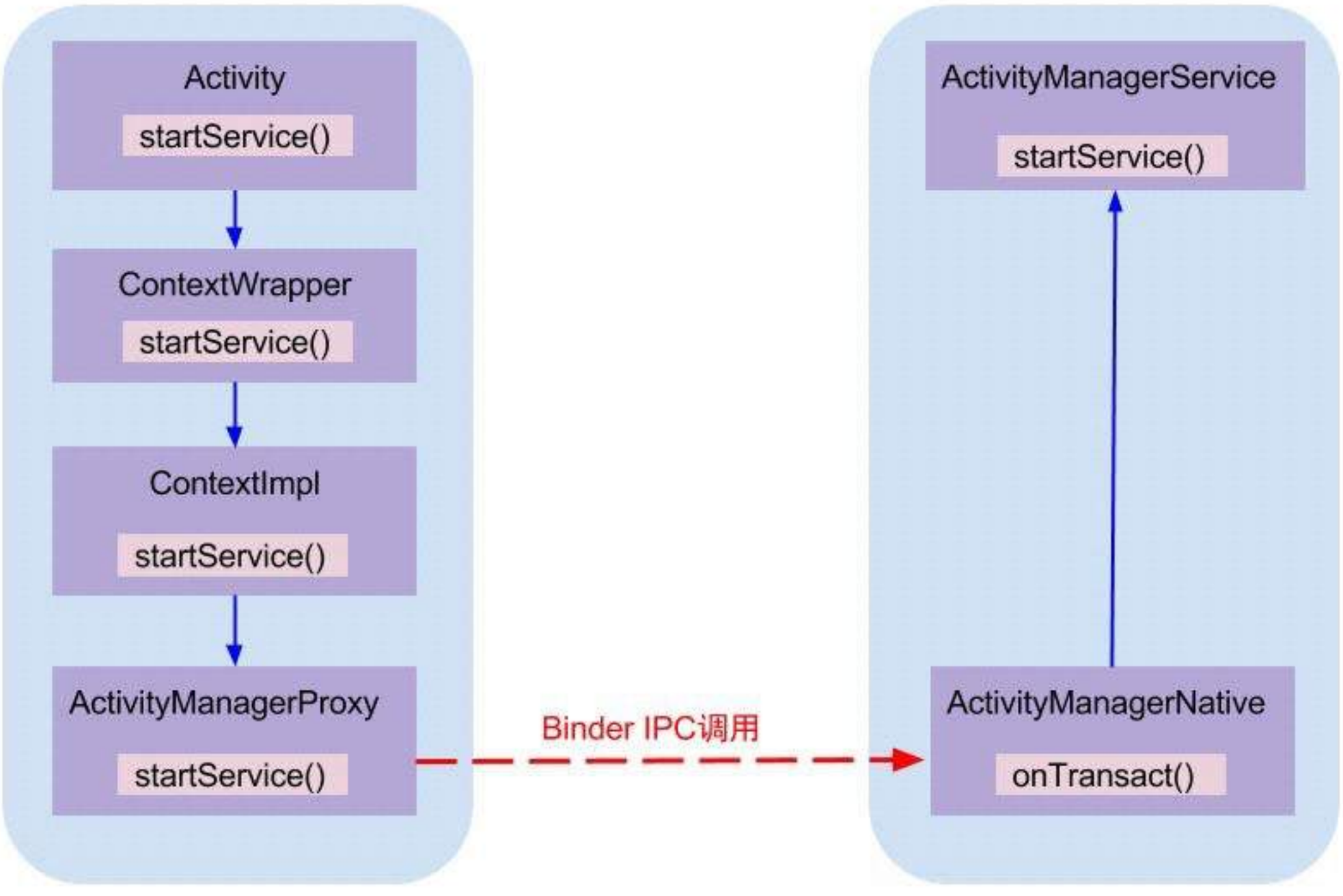
Binder 概述


1991， George Hoffman， OpenBinder
Later， Dinnie Hackborn， ParmOS
2003， Andy Rubin， Android OS

Binder 概述

- AIDL
- startService/ bindService
- send broadcast
- get contentProvider
- ...

例子：StartService(intent)

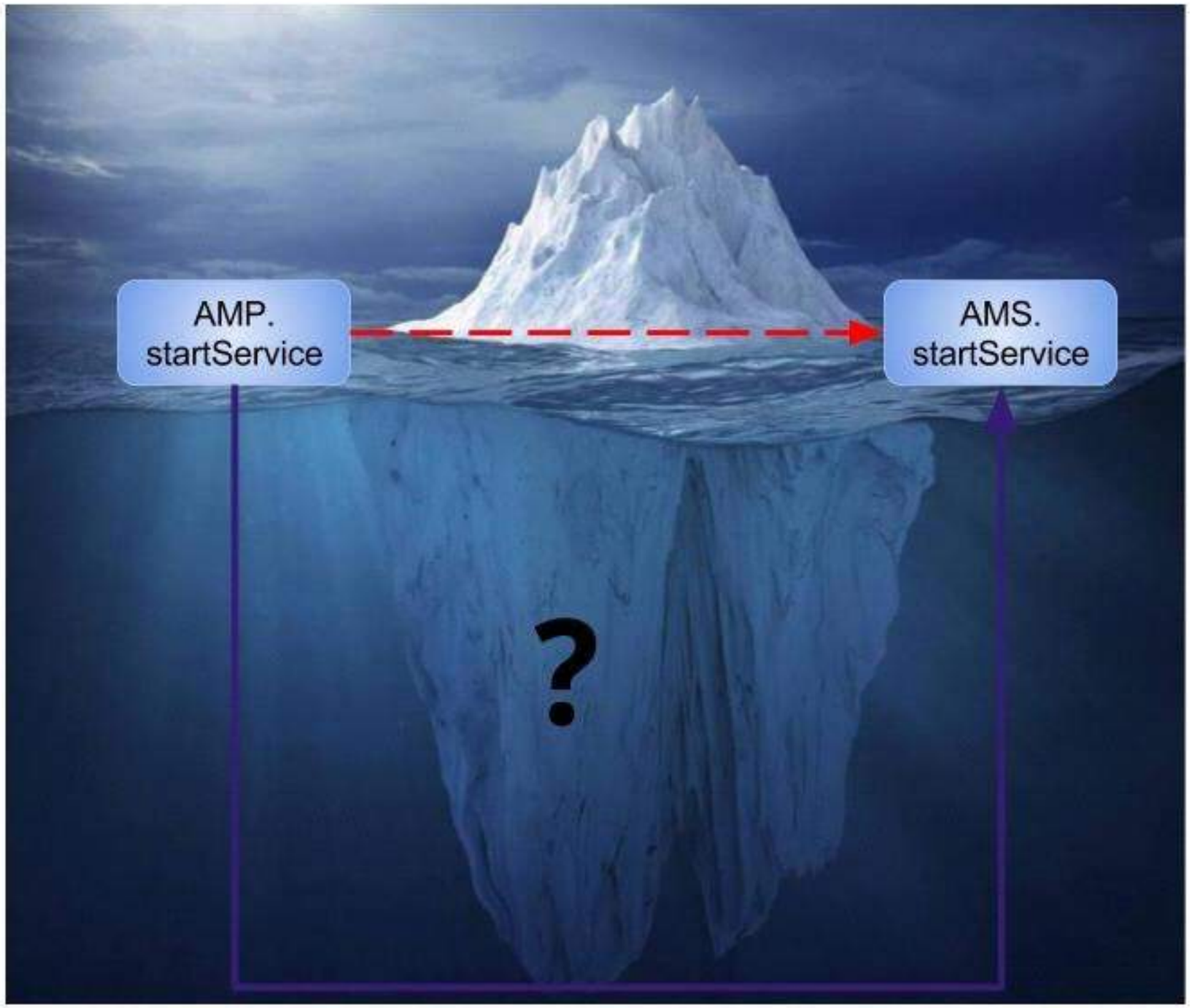


例子：StartService(intent)

```
public ComponentName startService(IApplicationThread caller, Intent service,
        String resolvedType, String callingPackage, int userId) throws RemoteException
{
    Parcel data = Parcel.obtain();
    Parcel reply = Parcel.obtain();
    data.writeInterfaceToken(IActivityManager.descriptor);
    data.writeStrongBinder(caller != null ? caller.asBinder() : null);
    service.writeToParcel(data, flags: 0);
    data.writeString(resolvedType);
    data.writeString(callingPackage);
    data.writeInt(userId);
    mRemote.transact(START_SERVICE_TRANSACTION, data, reply, flags: 0);
    reply.readException();
    ComponentName res = ComponentName.readFromParcel(reply);
    data.recycle();
    reply.recycle();
    return res;
}
```

```
case START_SERVICE_TRANSACTION: {
    data.enforceInterface(IActivityManager.descriptor);
    IBinder b = data.readStrongBinder();
    IApplicationThread app = ApplicationThreadNative.asInterface(b);
    Intent service = Intent.CREATOR.createFromParcel(data);
    String resolvedType = data.readString();
    String callingPackage = data.readString();
    int userId = data.readInt();
    ComponentName cn = startService(app, service, resolvedType, callingPackage, userId);
    reply.writeNoException();
    ComponentName.writeToParcel(cn, reply);
    return true;
}
```

例子：StartService(intent)



相关概念

- BpBinder, BBinder , handle
- BinderProxy
- IPCThreadState, ProcessState
- binder_proc, binder_thread
- binder_node, binder_ref
- binder_buffer, binder_transaction

提纲

- Binder概述
- Binder设计思想
- Binder通信协议
- Binder死亡通知
- Binder线程池管理

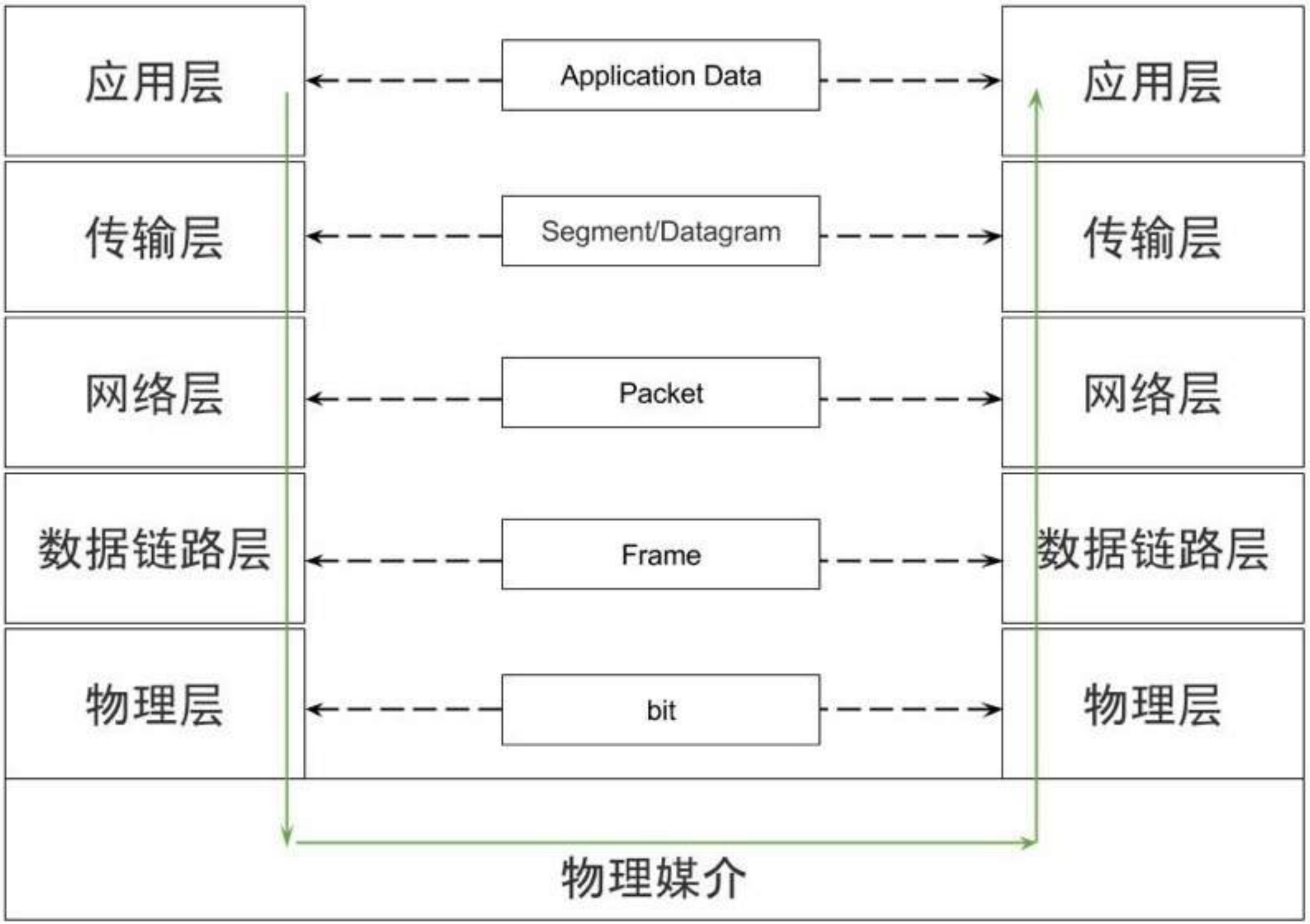
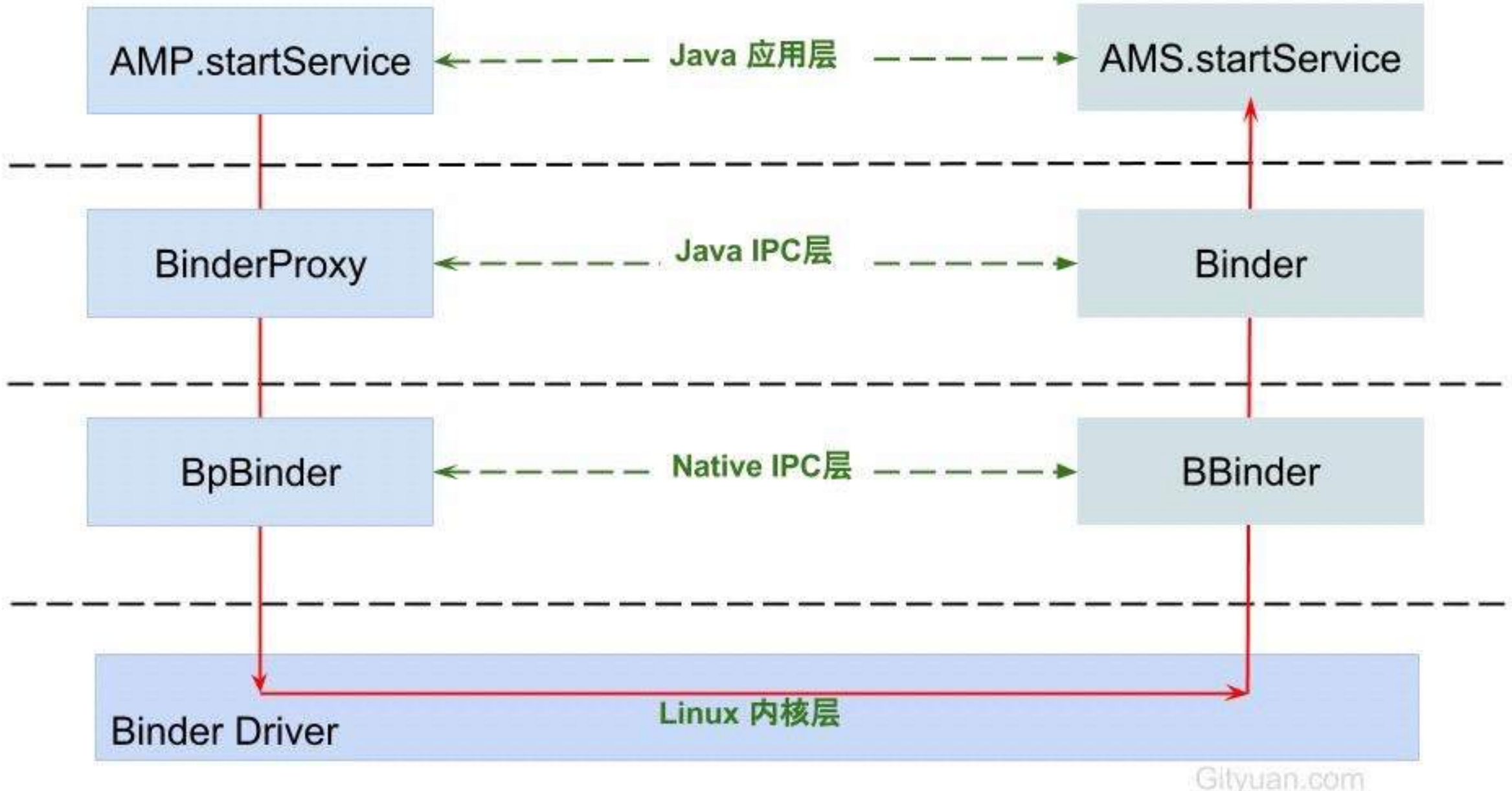


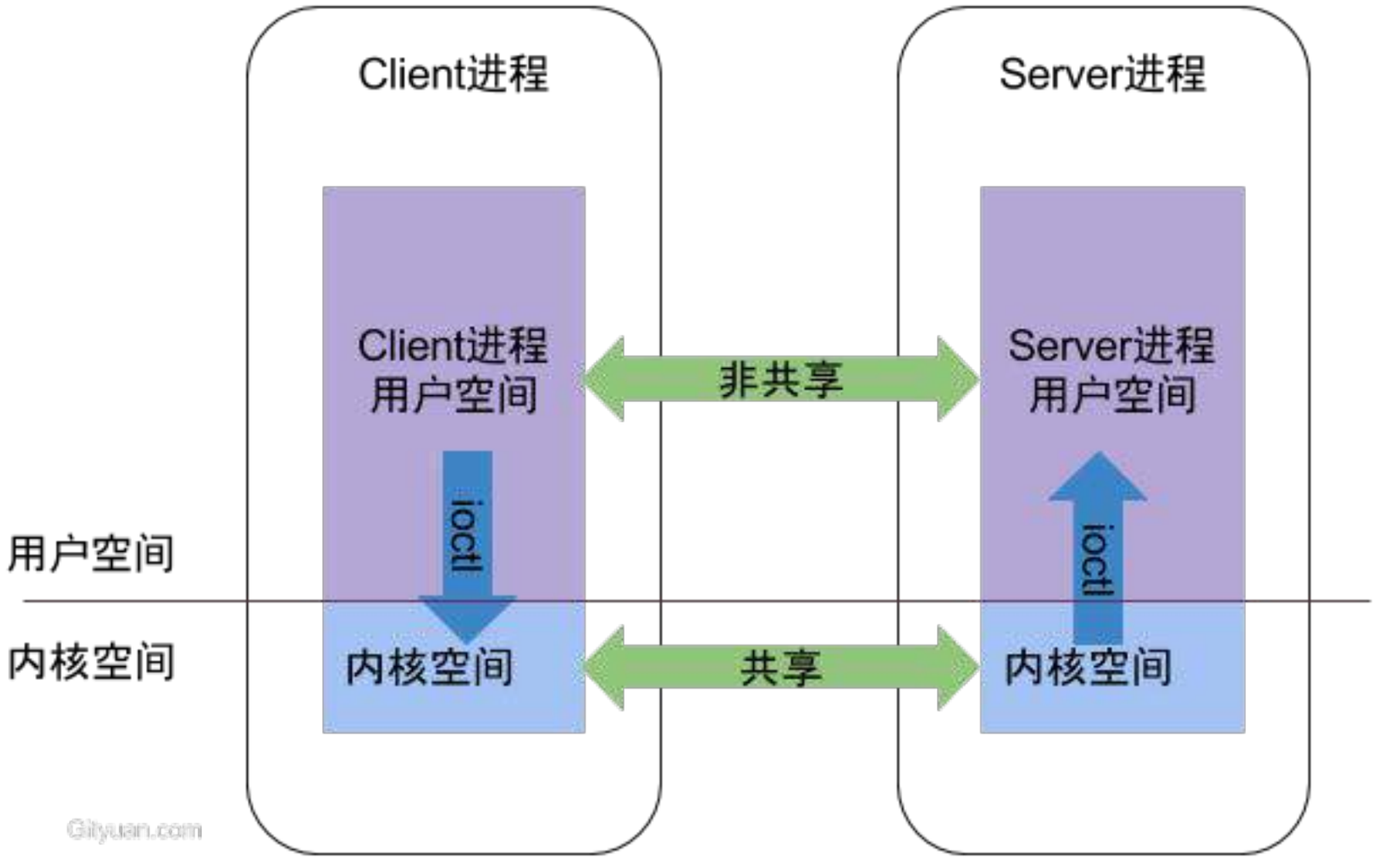
图: TCP/IP五层模型结构

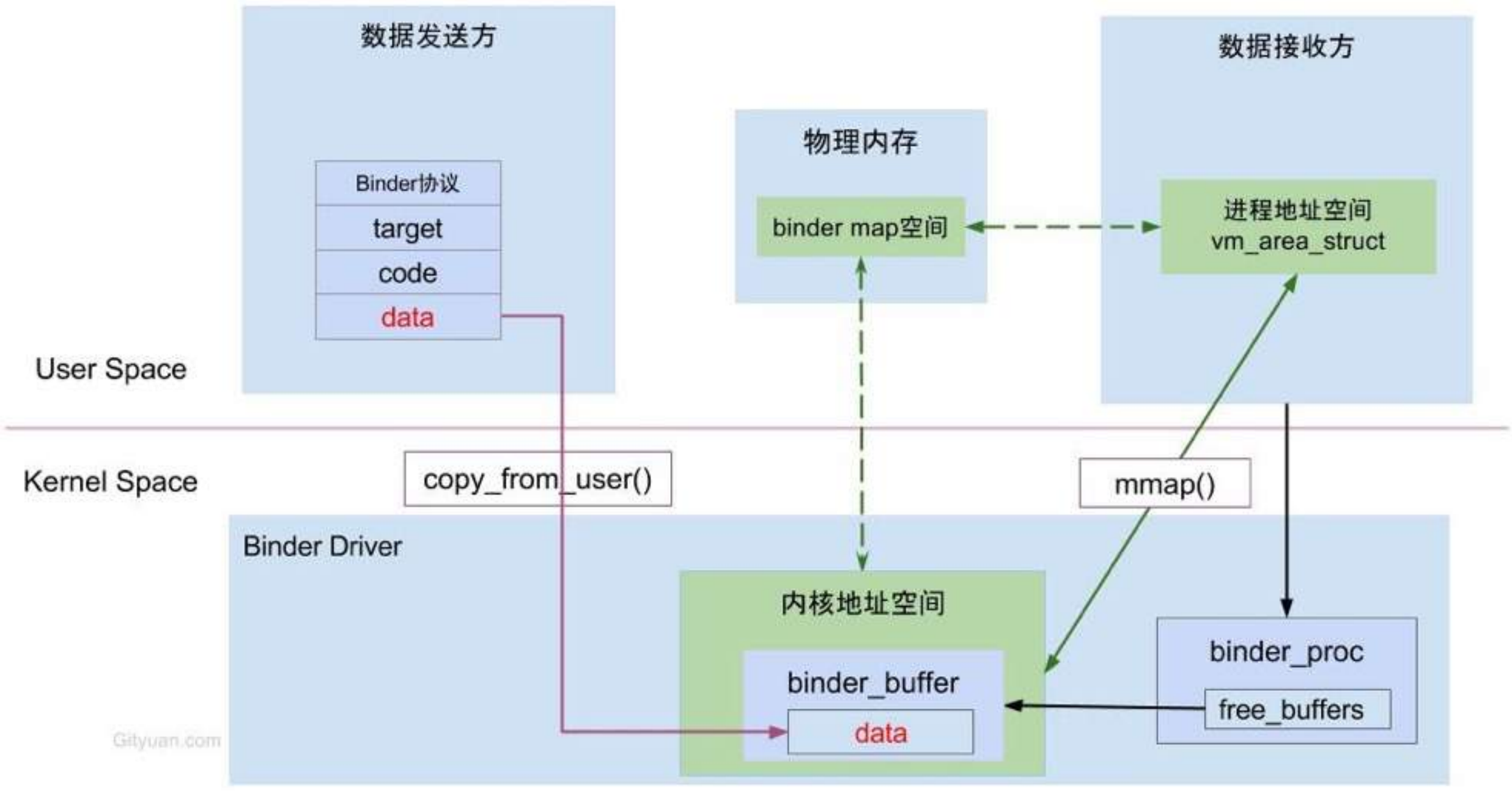

AMP.startService
Java 应用层
AMS.startService
BinderProxy
Java IPC层
Binder
BpBinder
Native IPC层
BBinder
Binder Driver
Linux 内核层
Gityuan.com

内存

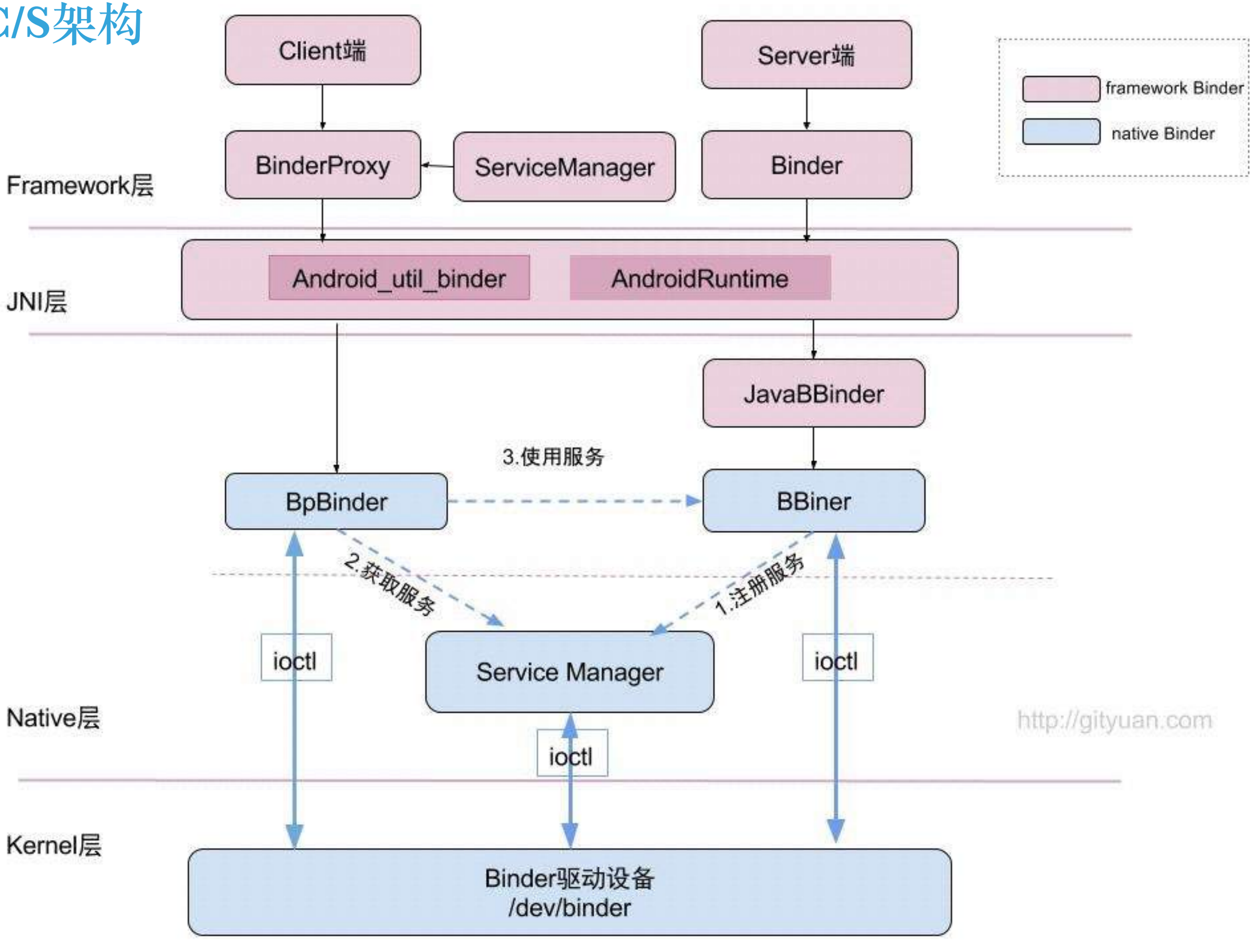
IPC方式	数据拷贝次数
管道、消息队列、Socket	2
Binder	1
共享内存	0

内存


Client进程
Server进程
Client进程
用户空间
非共享
Server进程
用户空间
ioctl
ioctl
用户空间
内核空间
内核空间
共享
内核空间
Gityuan.com


数据发送方
Binder协议
target
code
data
User Space
物理内存
binder map空间
数据接收方
进程地址空间
vm_area_struct
Kernel Space
copy_from_user()
mmap()
Binder Driver
内核地址空间
binder_buffer
data
binder_proc
free_buffers
Gityuan.com

C/S架构


Client端
Server端
framework Binder
native Binder
BinderProxy
ServiceManager
Binder
Framework层
Android_util_binder
AndroidRuntime
JNI层
JavaBBinder
3.使用服务
BpBinder
BBiner
2.获取服务
1.注册服务
ioctl
Service Manager
ioctl
ioctl
Native层
http://gityuan.com
Kernel层
Binder驱动设备
/dev/binder

BINDER 安全性

- 传统IPC的接收方无法获得对方进程可靠的UID/PID，从而无法鉴别对方身份；

- Android在应用程序安装过程便已分配指定的UID，进程的UID是鉴别进程身份的重要标志；使用传统IPC只能由用户在数据包里填入UID/PID；

- Binder通信的身份信息PID/UID，是由Linux内核在进程通信过程填充，安全可靠。

设计角度

- 面向对象的设计体验
- 系统平台级的全方位支持，使用简单
- Binder模糊了进程边界，淡化了进程间通信过程，整个系统仿佛运行于同一个面向对象的程序之中

App

Browser | Launcher | Phone | ... | Email

System Server

ActivityManagerService | WindowManagerService | PowerManagerService | ...

Java Framework

Zygote

Android Runtime

JNI

C++ Framework (Media)

Media Server

AudioFlinger | MediaPlayerService | ...

Native

User Daemons

Init(pid=1)

bootanim | Service Manager | ...

HAL

SysCall

Kernel

Kernel Daemons

Camera Driver | Binder Driver

Display Driver | ...

kthreadd(pid=2)

swapper(pid=0)

Loader

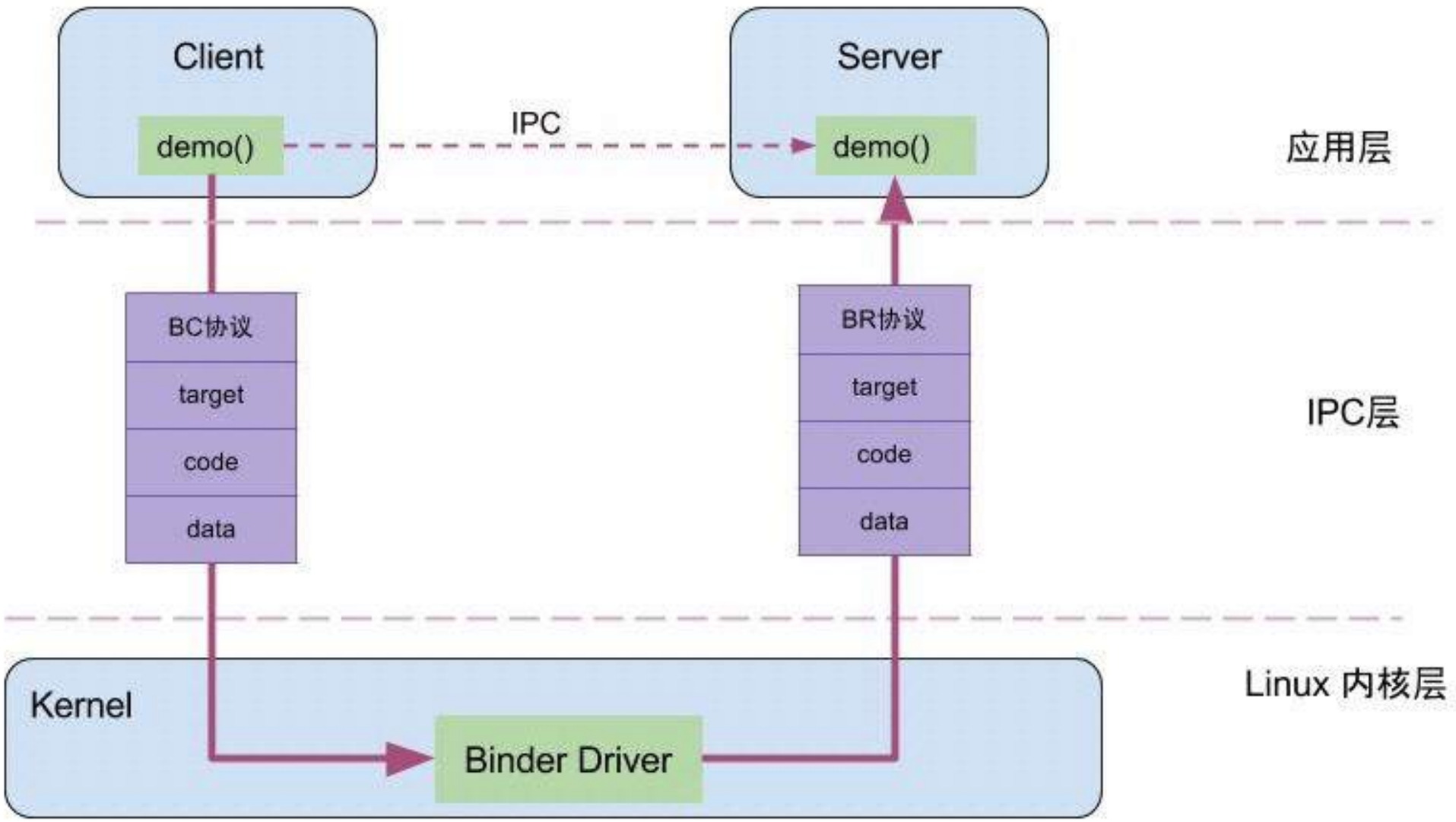
Boot ROM

Boot Loader

提纲

- Binder概述
- Binder设计思想
- Binder通信协议
- Binder死亡通知
- Binder线程池管理

BINDER 通信协议


Client
demo()
IPC
Server
demo()
应用层
BC协议
target
code
data
BR协议
target
code
data
IPC层
Kernel
Binder Driver
Linux 内核层

实例

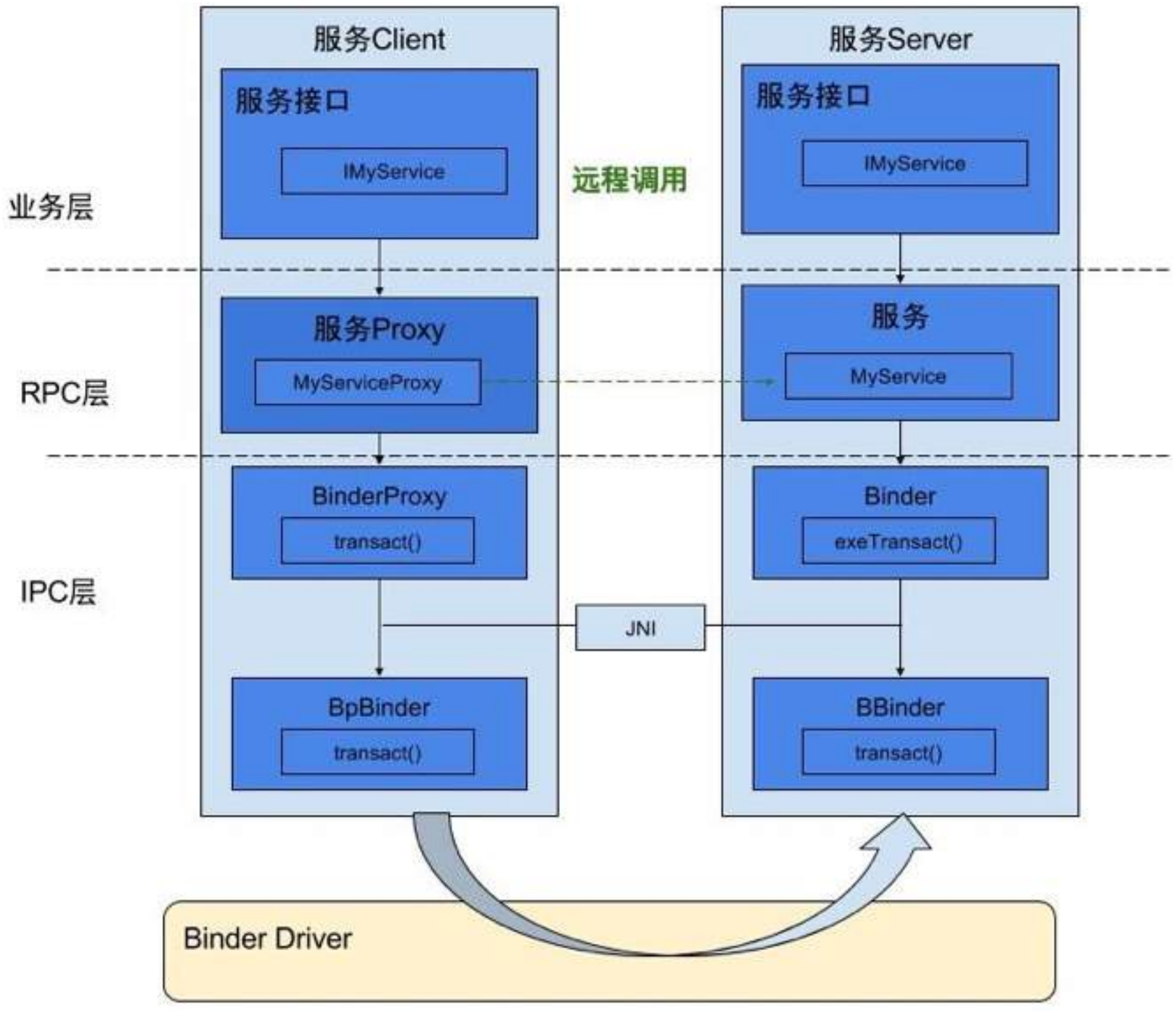
```
#include "IMyService.h"
int main() {
    //获取service manager引用
    sp < IServiceManager > sm = defaultServiceManager();
    //注册名为"service.myservice"的服务到service manager
    sm->addService(String16("service.myservice"), new BnMyService());
    ProcessState::self()->startThreadPool(); //启动线程池
    IPCThreadState::self()->joinThreadPool(); //把主线程加入线程池
    return 0;
}
```

注册服务

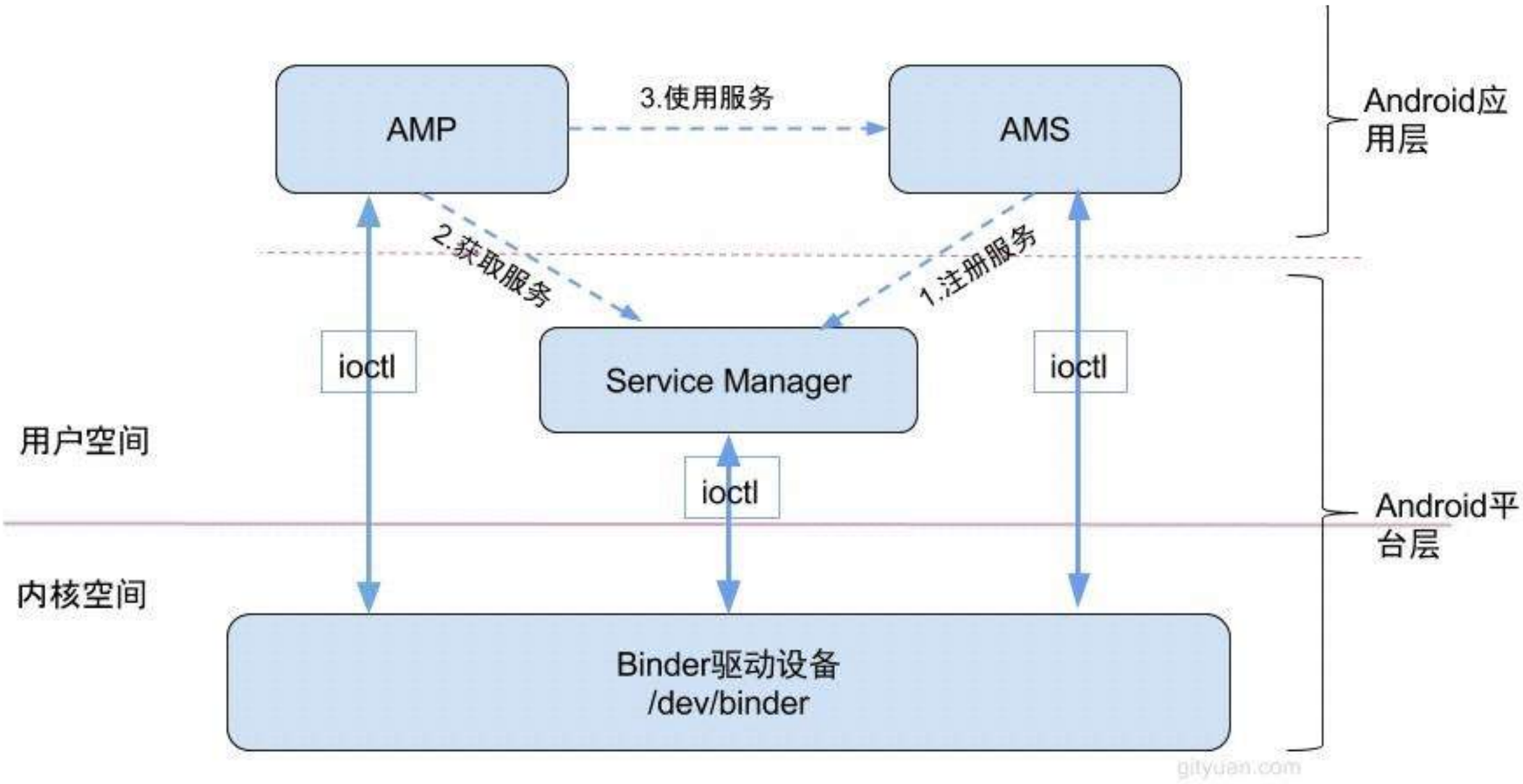
```
#include "IMyService.h"
int main() {
    //获取service manager引用
    sp < IServiceManager > sm = defaultServiceManager();
    //获取名为"service.myservice"的binder接口
    sp < IBinder > binder = sm->getService(String16("service.myservice"));
    //将Biher对象转换为强引用类型的IMyService
    sp<IMyService> cs = interface_cast < IMyService > (binder);
    //利用binder引用调用远程sayHello()方法
    cs->sayHello();
    return 0;
}
```

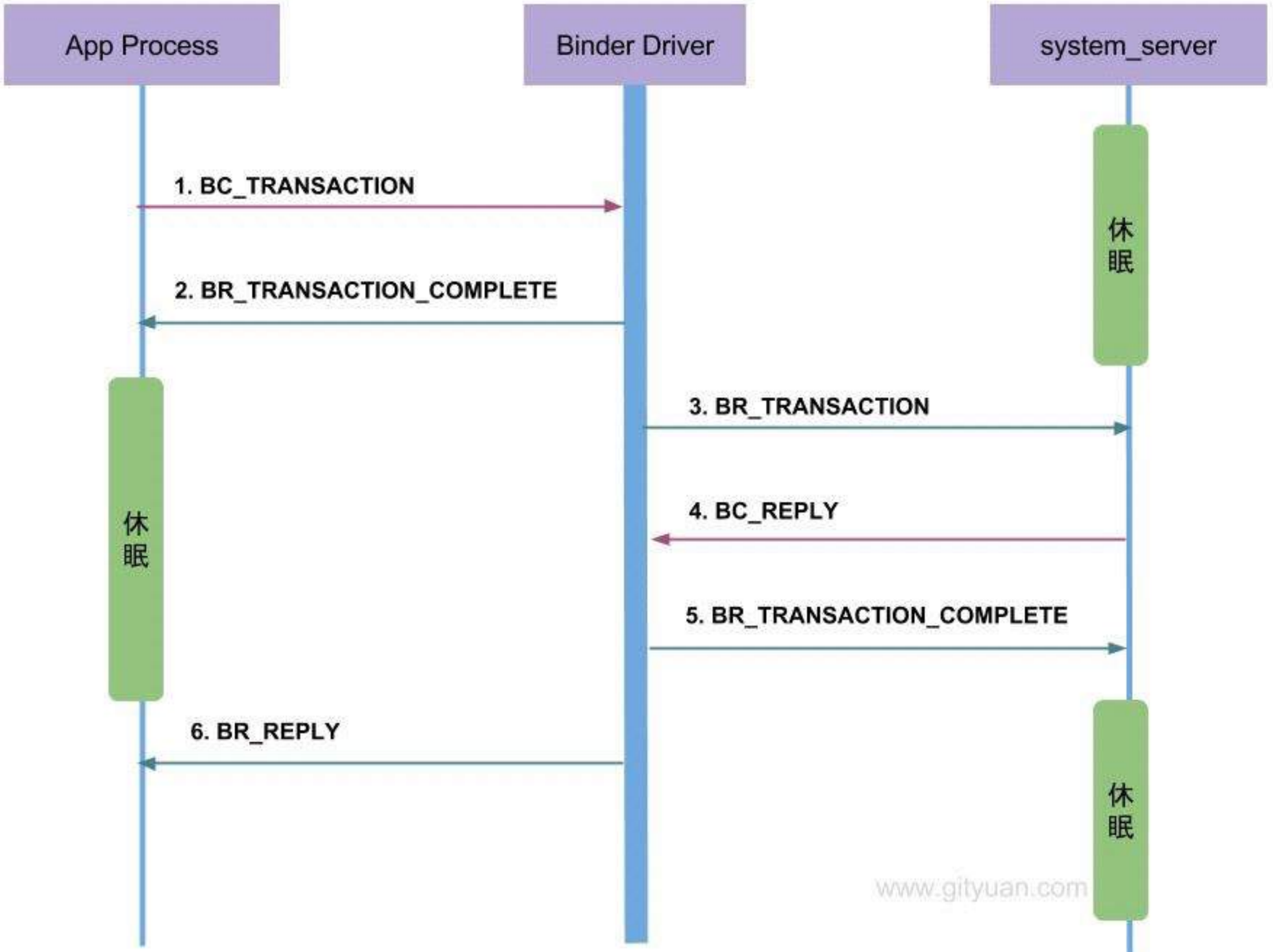
请求服务

实例

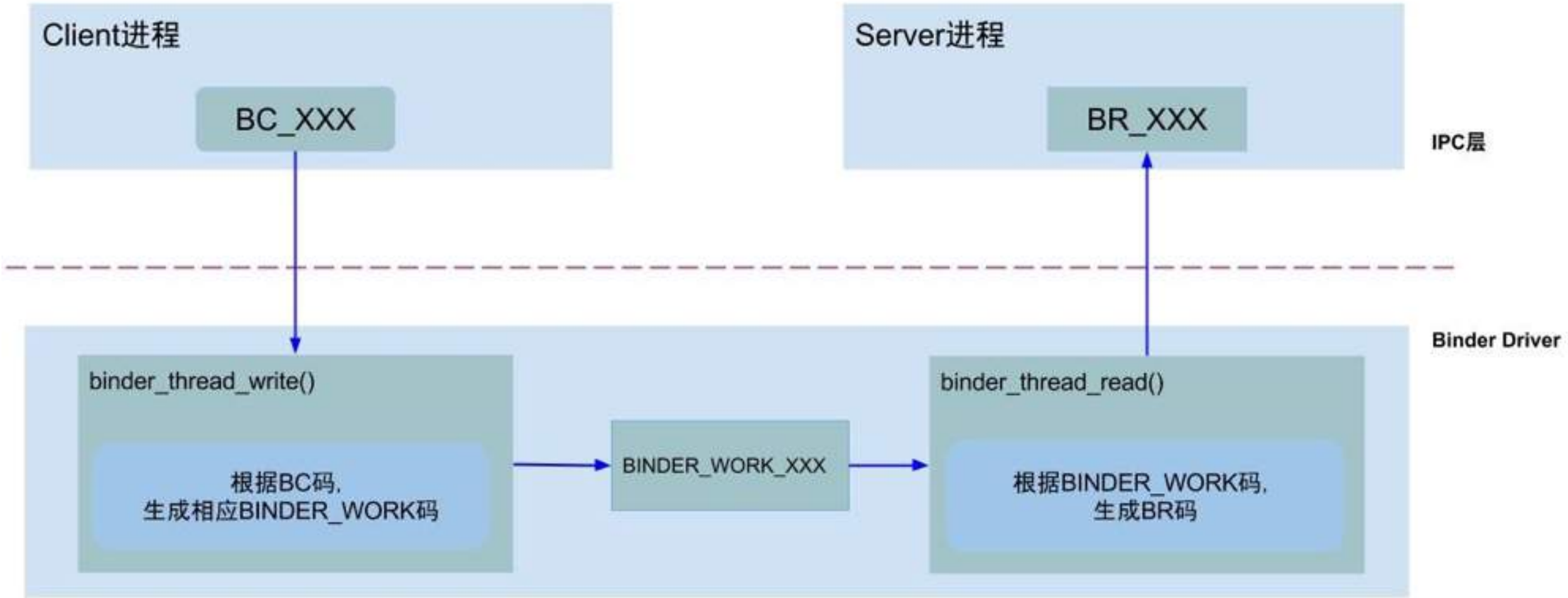

服务Client
服务接口
IMyService
远程调用
服务Server
服务接口
IMyService
业务层
服务Proxy
MyServiceProxy
服务
MyService
RPC层
BinderProxy
transact()
Binder
exeTransact()
IPC层
JNI
BpBinder
transact()
BBinder
transact()
Binder Driver

BINDER 通信协议


AMP
3.使用服务
AMS
Android应用层
2.获取服务
1.注册服务
ioctl
Service Manager
ioctl
用户空间
ioctl
Android平台层
内核空间
Binder驱动设备
/dev/binder
gityuan.com


App Process
Binder Driver
system_server
1. BC_TRANSACTION
2. BR_TRANSACTION_COMPLETE
休眠
3. BR_TRANSACTION
4. BC_REPLY
5. BR_TRANSACTION_COMPLETE
6. BR_REPLY
休眠
休眠
www.gityuan.com

BINDER 通信协议


Client进程
BC_XXX
Server进程
BR_XXX
IPC层
Binder Driver
binder_thread_write()
根据BC码,
生成相应BINDER_WORK码
BINDER_WORK_XXX
binder_thread_read()
根据BINDER_WORK码,
生成BR码

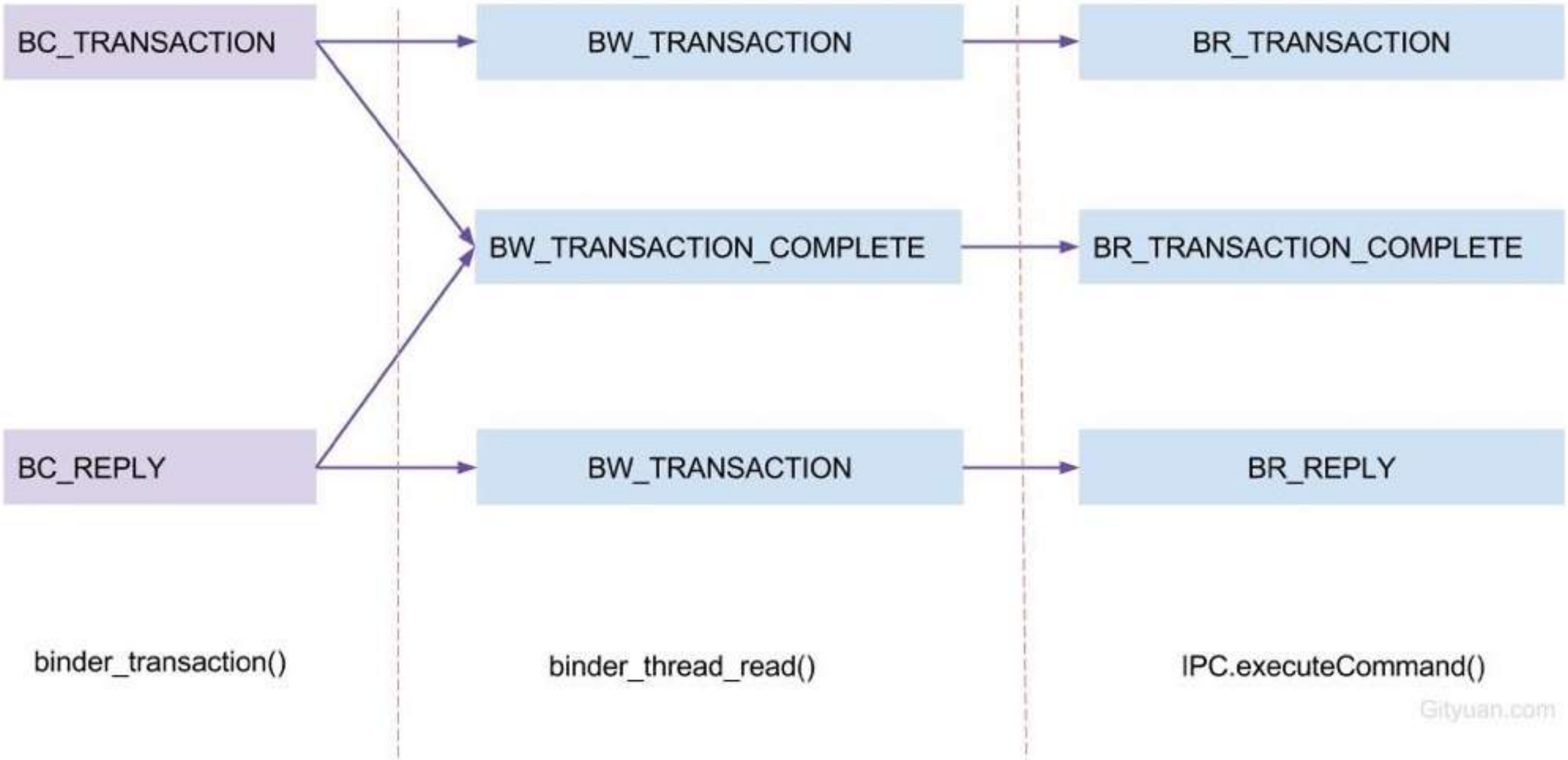
BC协议

BC协议	调用方法
BC_TRANSACTION	IPC.transact()
BC_REPLY	IPC.sendReply()
BC_FREE_BUFFER	IPC.freeBuffer()
BC_REQUEST_DEATH_NOTIFICATION	IPC.requestDeathNotification()
BC_CLEAR_DEATH_NOTIFICATION	IPC.clearDeathNotification()
BC_DEAD_BINDER_DONE	IPC.execute()

BR协议

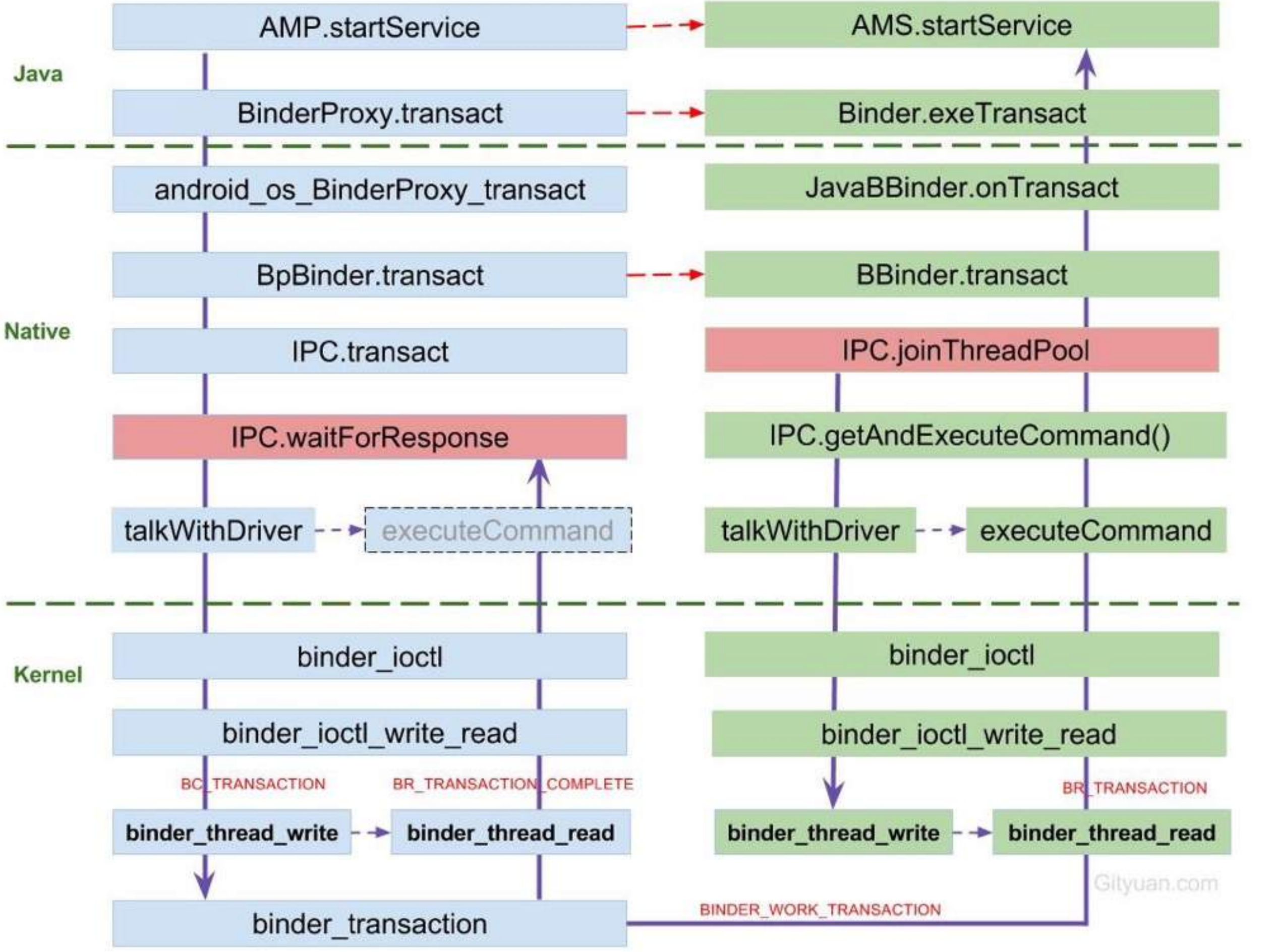
BR协议	触发时机
BR_TRANSACTION	收到BINDER_WORK_TRANSACTION
BR_REPLY	收到BINDER_WORK_TRANSACTION
BR_TRANSACTION_COMPLETE	收到BINDER_WORK_TRANSACTION_COMPLETE
BR_DEAD_BINDER	收到BINDER_WORK_DEAD_BINDER或 BINDER_WORK_DEAD_BINDER_AND_CLEAR
BR_CLEAR_DEATH_NOTIFICATION_DONE	收到BINDER_WORK_CLEAR_DEATH_NOTIFICATION

协议转换

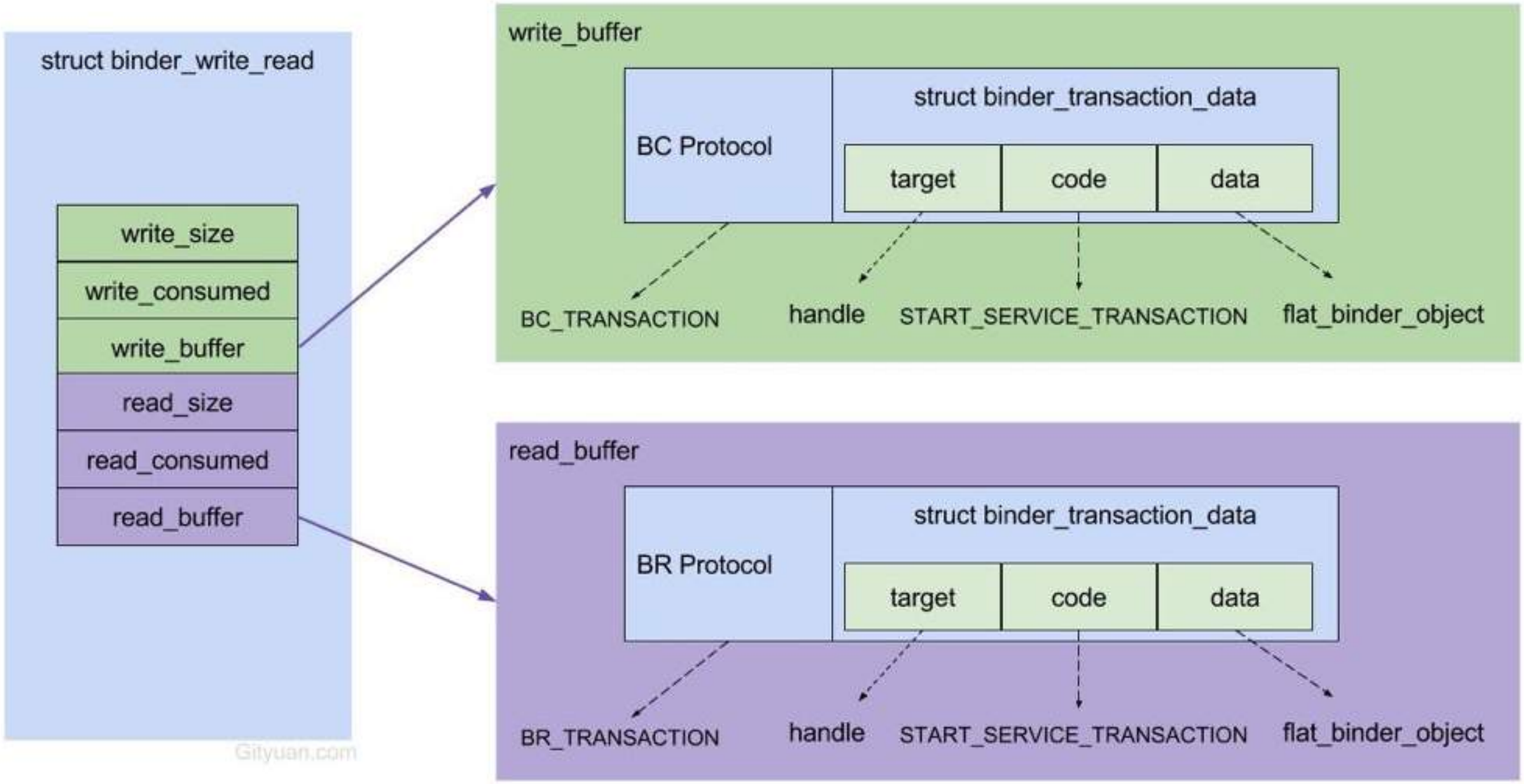

BC_TRANSACTION
BW_TRANSACTION
BR_TRANSACTION
BW_TRANSACTION_COMPLETE
BR_TRANSACTION_COMPLETE
BC_REPLY
BW_TRANSACTION
BR_REPLY
binder_transaction()
binder_thread_read()
IPC.executeCommand()
Gityuan.com

BINDER调用栈

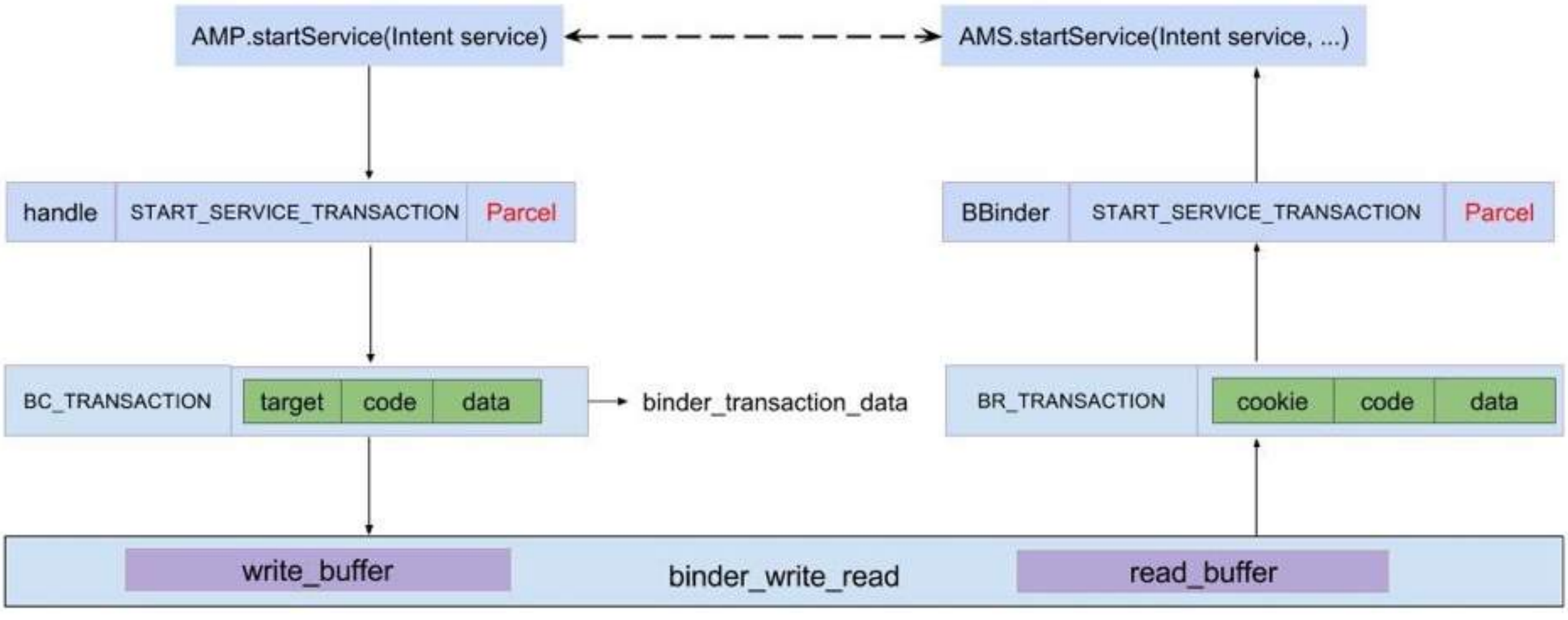
```
"Binder:1557_16" prio=5 tid=117 Blocked
  | group="main" sCount=1 dsCount=0 obj=0x153120d0 self=0x7f89b9ee00
  | sysTid=10642 nice=0 cgrp=default sched=0/0 handle=0x7f7352c450
  | state=S schedstat=( 0 0 0 ) utm=2748 stm=1377 core=1 HZ=100
  | stack=0x7f73432000-0x7f73434000 stackSize=1005KB
  | held mutexes=
  at com.android.server.am.ActivityManagerService.checkContentProviderAccess(ActivityManagerService.java:10786)
  - waiting to lock <0x0ef02a62> (a com.android.server.am.ActivityManagerService) held by thread 101
  at com.android.server.am.ActivityManagerService$LocalService.checkContentProviderAccess(ActivityManagerService.java:22502)
  at com.android.server.content.ContentService.registerContentObserver(ContentService.java:302)
  at android.content.IContentService$Stub.onTransact IContentService.java:73)
  at com.android.server.content.ContentService.onTransact(ContentService.java:235)
  at android.os.Binder.execTransact(Binder.java:565)

"Binder:1557_17" prio=5 tid=119 Native
  | group="main" sCount=1 dsCount=0 obj=0x15312a60 self=0x7f8a542600
  | sysTid=10643 nice=0 cgrp=default sched=0/0 handle=0x7f7342f450
  | state=S schedstat=( 0 0 0 ) utm=2781 stm=1306 core=1 HZ=100
  | stack=0x7f73335000-0x7f73337000 stackSize=1005KB
  | held mutexes=
  kernel: (couldn't read /proc/self/task/10643/stack)
  native: #00 pc 00000000007831c  /system/lib64/libc.so (__ioctl+4)
  native: #01 pc 0000000000020020  /system/lib64/libc.so (ioctl+140)
  native: #02 pc 0000000000055b14  /system/lib64/libbinder.so (_ZN7android1 IPCThreadState14talkWithDriver b+256)
  native: #03 pc 0000000000055c7c  /system/lib64/libbinder.so (_ZN7android14IPCThreadState20getAndExecuteCommandEv+24)
  native: #04 pc 00000000000563f8  /system/lib64/libbinder.so (_ZN7android14IPCThreadState14joinThreadPoolEb+128)
  native: #05 pc 00000000000747e4  /system/lib64/libbinder.so (???)
  native: #06 pc 0000000000012938  /system/lib64/libutils.so (_ZN7android6Thread11_threadLoopEPv+336)
  native: #07 pc 00000000000a0c88  /system/lib64/libandroid_runtime.so (_ZN7android14AndroidRuntime15javaThreadShellEPv+116)
  native: #08 pc 0000000000075c54  /system/lib64/libc.so (_ZL15__pthread_startPv+204)
  native: #09 pc 00000000001e0fc  /system/lib64/libc.so (__start_thread+16)
```


AMP.startService
AMS.startService
Java
BinderProxy.transact
Binder.exeTransact
android_os_BinderProxy_transact
JavaBBinder.onTransact
BpBinder.transact
BBinder.transact
Native
IPC.transact
IPC.joinThreadPool
IPC.waitForResponse
IPC.getAndExecuteCommand()
talkWithDriver
executeCommand
talkWithDriver
executeCommand
Kernel
binder_ioctl
binder_ioctl
binder_ioctl_write_read
binder_ioctl_write_read
BC_TRANSACTION
BR_TRANSACTION_COMPLETE
BR_TRANSACTION
binder_thread_write
binder_thread_read
binder_thread_write
binder_thread_read
binder_transaction
BINDER_WORK_TRANSACTION
Gityuan.com

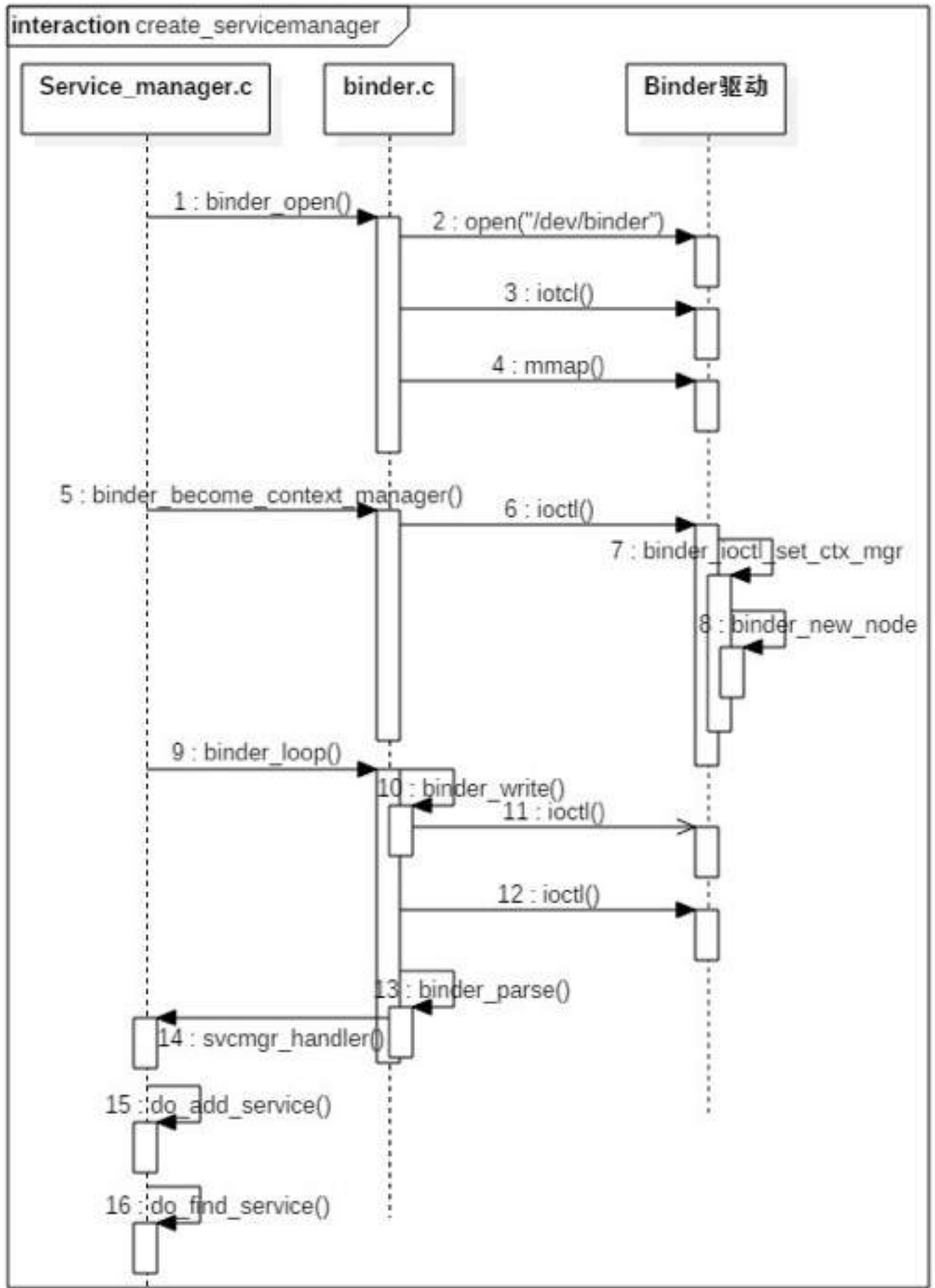
数据加工


struct binder_write_read
write_size
write_consumed
write_buffer
read_size
read_consumed
read_buffer
write_buffer
BC Protocol
struct binder_transaction_data
target
code
data
BC_TRANSACTION
handle
START_SERVICE_TRANSACTION
flat_binder_object
read_buffer
BR Protocol
struct binder_transaction_data
target
code
data
BR_TRANSACTION
handle
START_SERVICE_TRANSACTION
flat_binder_object
Gityuan.com

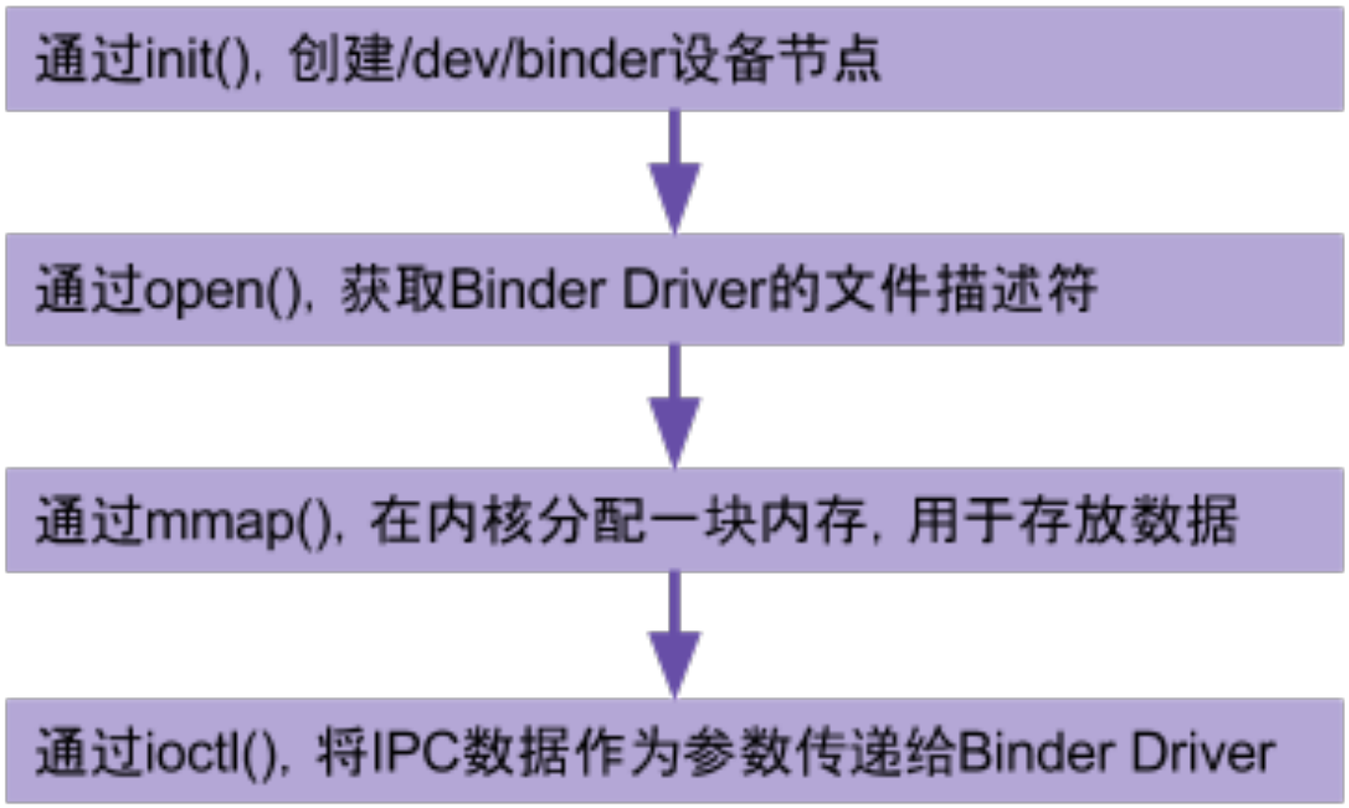
数据加工


AMP.startService(Intent service)
AMS.startService(Intent service, ...)
handle
START_SERVICE_TRANSACTION
Parcel
BBinder
START_SERVICE_TRANSACTION
Parcel
BC_TRANSACTION
target
code
data
binder_transaction_data
BR_TRANSACTION
cookie
code
data
write_buffer
binder_write_read
read_buffer

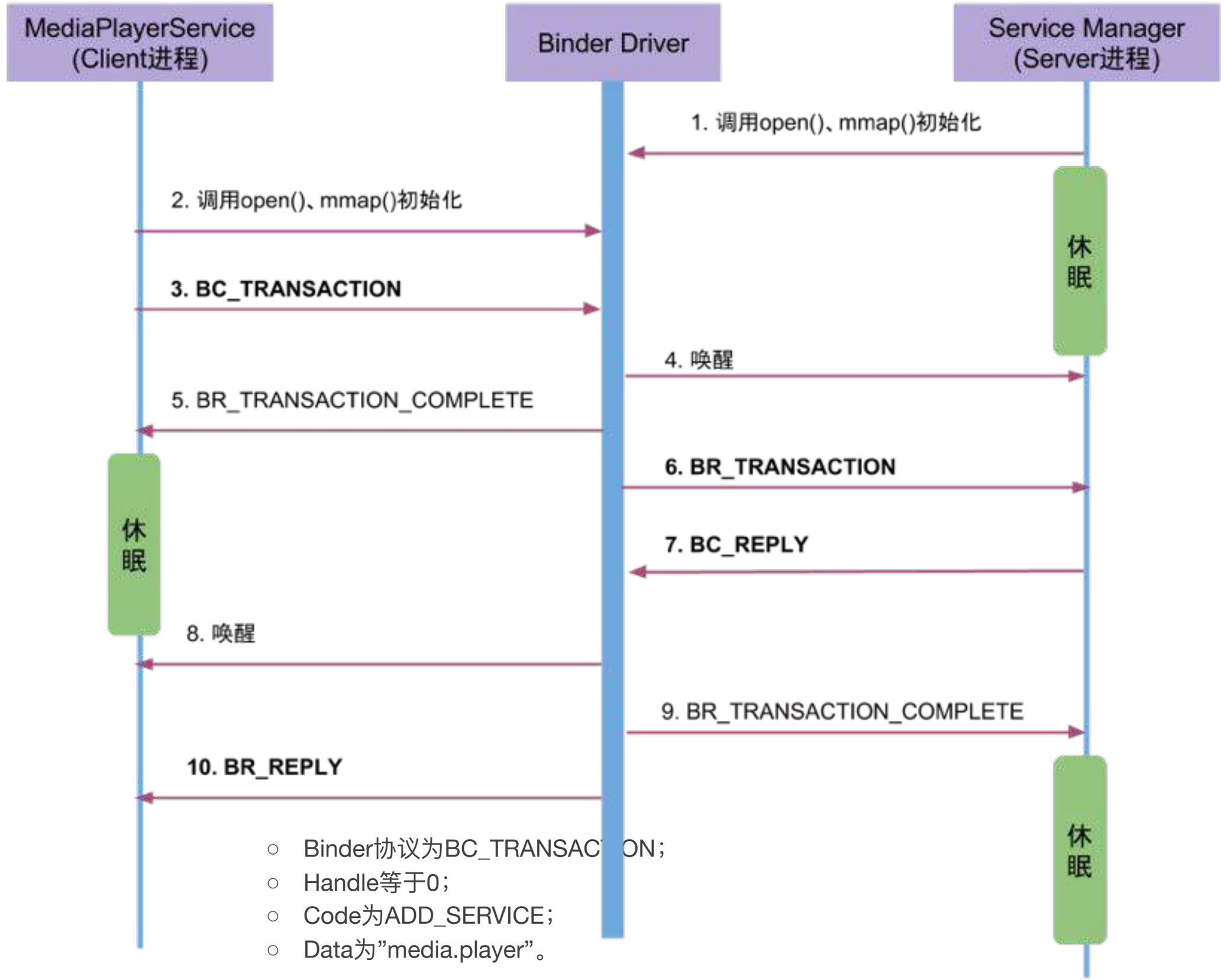
启动大管家


interaction create_servicemanager
Service_manager.c
binder.c
Binder驱动
1 : binder_open()
2 : open("/dev/binder")
3 : iotcl()
4 : mmap()
5 : binder_become_context_manager()
6 : ioctl()
7 : binder_ioctl_set_ctx_mgr
8 : binder_new_node
9 : binder_loop()
10 : binder_write()
11 : ioctl()
12 : ioctl()
13 : binder_parse()
14 : svcmgr_handler()
15 : do_add_service()
16 : do_find_service()

驱动加载


通过init()，创建/dev/binder设备节点
通过open()，获取Binder Driver的文件描述符
通过mmap()，在内核分配一块内存，用于存放数据
通过ioctl()，将IPC数据作为参数传递给Binder Driver

注册服务



- Binder协议为BC_TRANSACTION；
- Handle等于0；
- Code为ADD_SERVICE；
- Data为”media.player”。

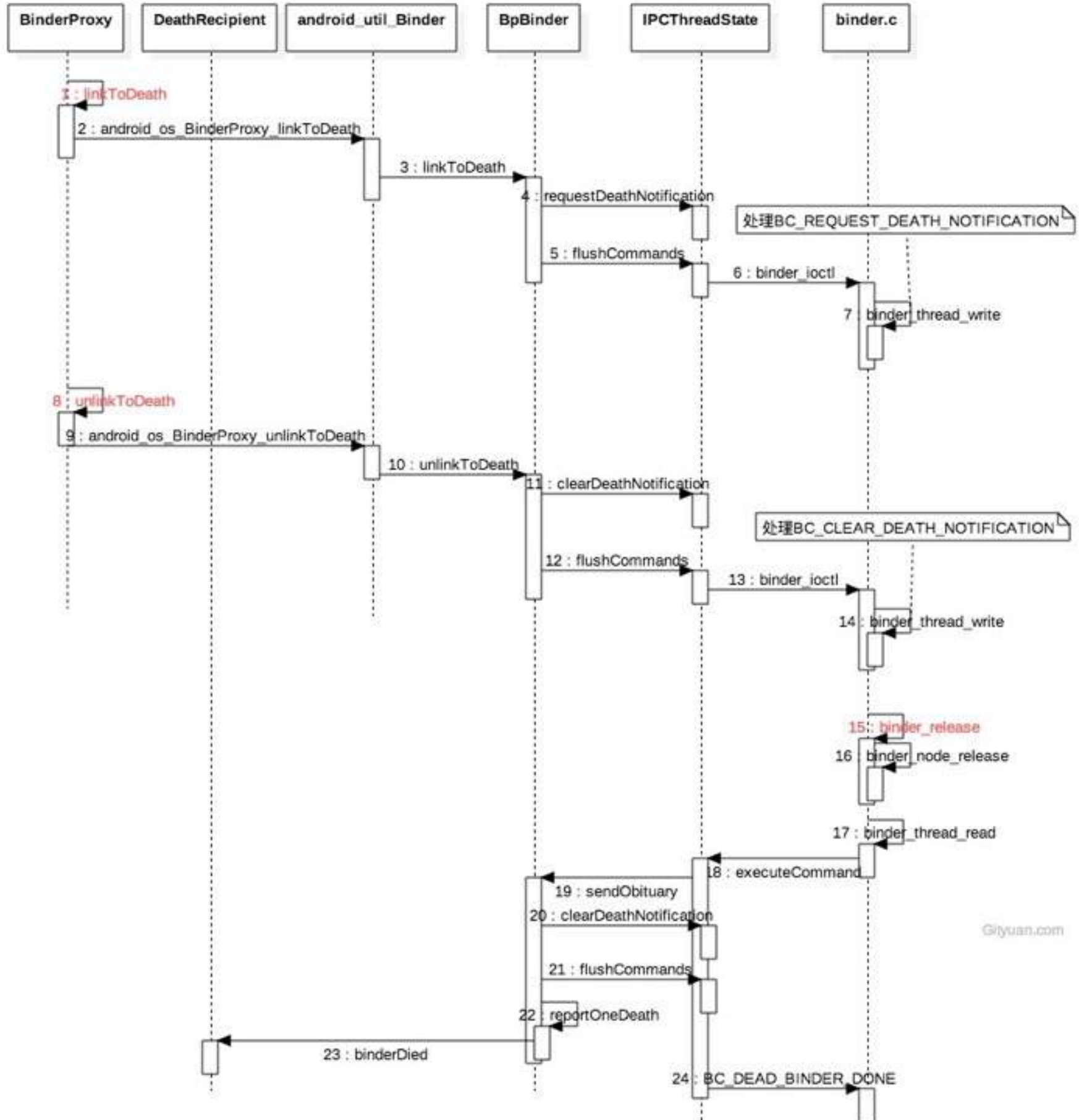
请求服务

```
case BINDER_TYPE_HANDLE:
case BINDER_TYPE_WEAK_HANDLE: {
        struct binder_ref *ref = binder_get_ref(proc, fp->handle);

        if (ref == NULL) {
                binder_user_error("%d:%d got transaction with invalid handle, %d\n",
                                  proc->pid,
                                  thread->pid, fp->handle);
                return_error = BR_FAILED_REPLY;
                goto err_binder_get_ref_failed;
        }
        if (security_binder_transfer_binder(proc->tsk, target_proc->tsk)) {
                return_error = BR_FAILED_REPLY;
                goto err_binder_get_ref_failed;
        }
        if (ref->node->proc == target_proc) {
                if (fp->type == BINDER_TYPE_HANDLE)
                        fp->type = BINDER_TYPE_BINDER;
                else
                        fp->type = BINDER_TYPE_WEAK_BINDER;
                fp->binder = ref->node->ptr;
                fp->cookie = ref->node->cookie;
                binder_inc_node(ref->node, fp->type == BINDER_TYPE_BINDER, 0, NULL);
                trace_binder_transaction_ref_to_node(t, ref);
                binder_debug(BINDER_DEBUG_TRANSACTION,
                             "        ref %d desc %d -> node %d u%016llx\n",
                             ref->debug_id, ref->desc, ref->node->debug_id,
                             (u64)ref->node->ptr);
        } else {
                struct binder_ref *new_ref;

                new_ref = binder_get_ref_for_node(target_proc, ref->node);
                if (new_ref == NULL) {
                        return_error = BR_FAILED_REPLY;
                        goto err_binder_get_ref_for_node_failed;
                }
                fp->handle = new_ref->desc;
                binder_inc_ref(new_ref, fp->type == BINDER_TYPE_HANDLE, NULL);
                trace_binder_transaction_ref_to_ref(t, ref,
                                                    new_ref);
```

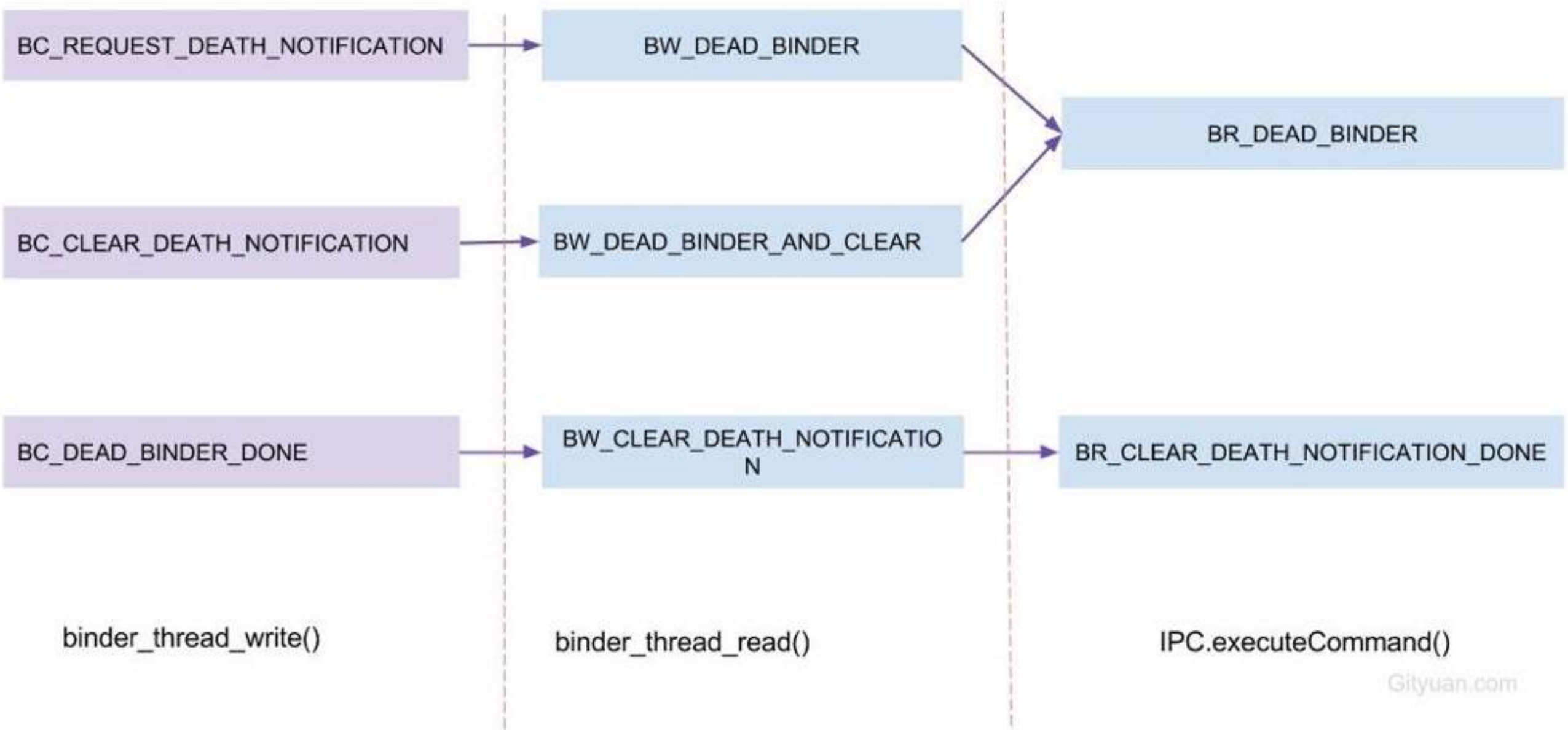
提纲

- Binder概述
- Binder设计思想
- Binder通信协议
- Binder死亡通知
- Binder线程池管理

死亡通知

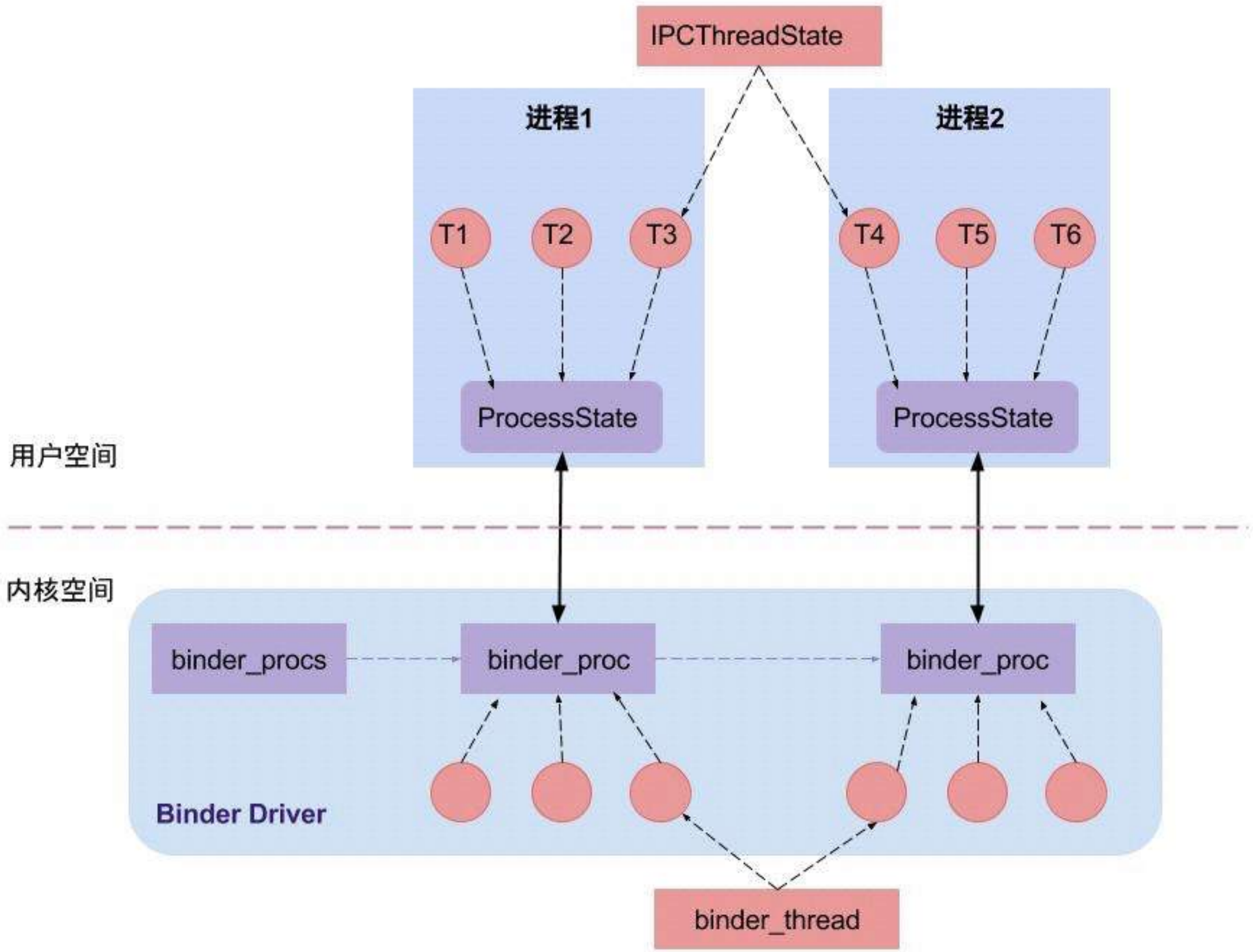

BinderProxy
DeathRecipient
android_util_Binder
BpBinder
IPCThreadState
binder.c
1 : linkToDeath
2 : android_os_BinderProxy_linkToDeath
3 : linkToDeath
4 : requestDeathNotification
处理BC_REQUEST_DEATH_NOTIFICATION
5 : flushCommands
6 : binder_ioctl
7 : binder_thread_write
8 : unlinkToDeath
9 : android_os_BinderProxy_unlinkToDeath
10 : unlinkToDeath
11 : clearDeathNotification
处理BC_CLEAR_DEATH_NOTIFICATION
12 : flushCommands
13 : binder_ioctl
14 : binder_thread_write
15 : binder_release
16 : binder_node_release
17 : binder_thread_read
18 : executeCommand
19 : sendObituary
20 : clearDeathNotification
21 : flushCommands
22 : reportOneDeath
23 : binderDied
24 : BC_DEAD_BINDER_DONE
Gityuan.com

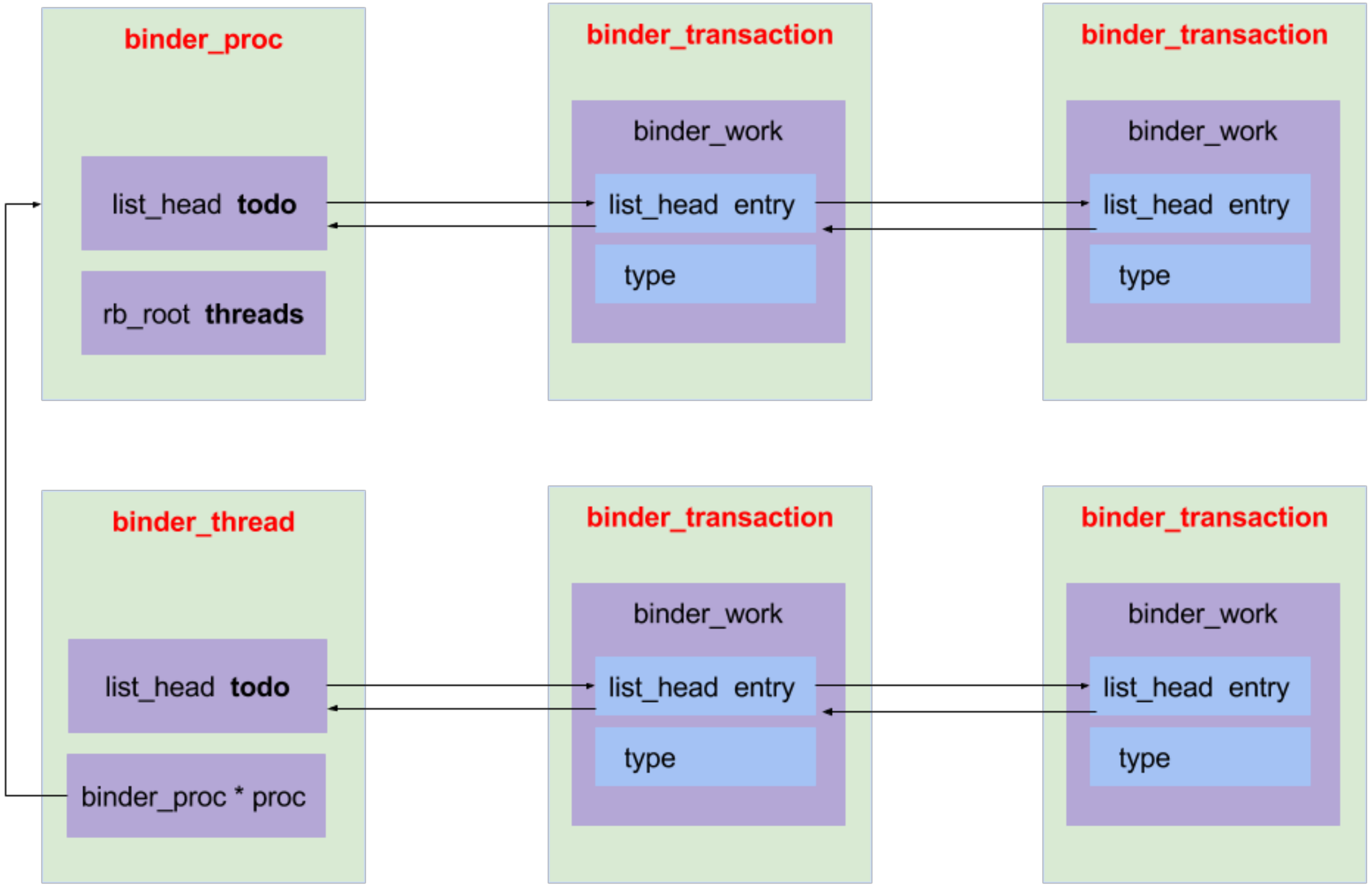
死亡通知

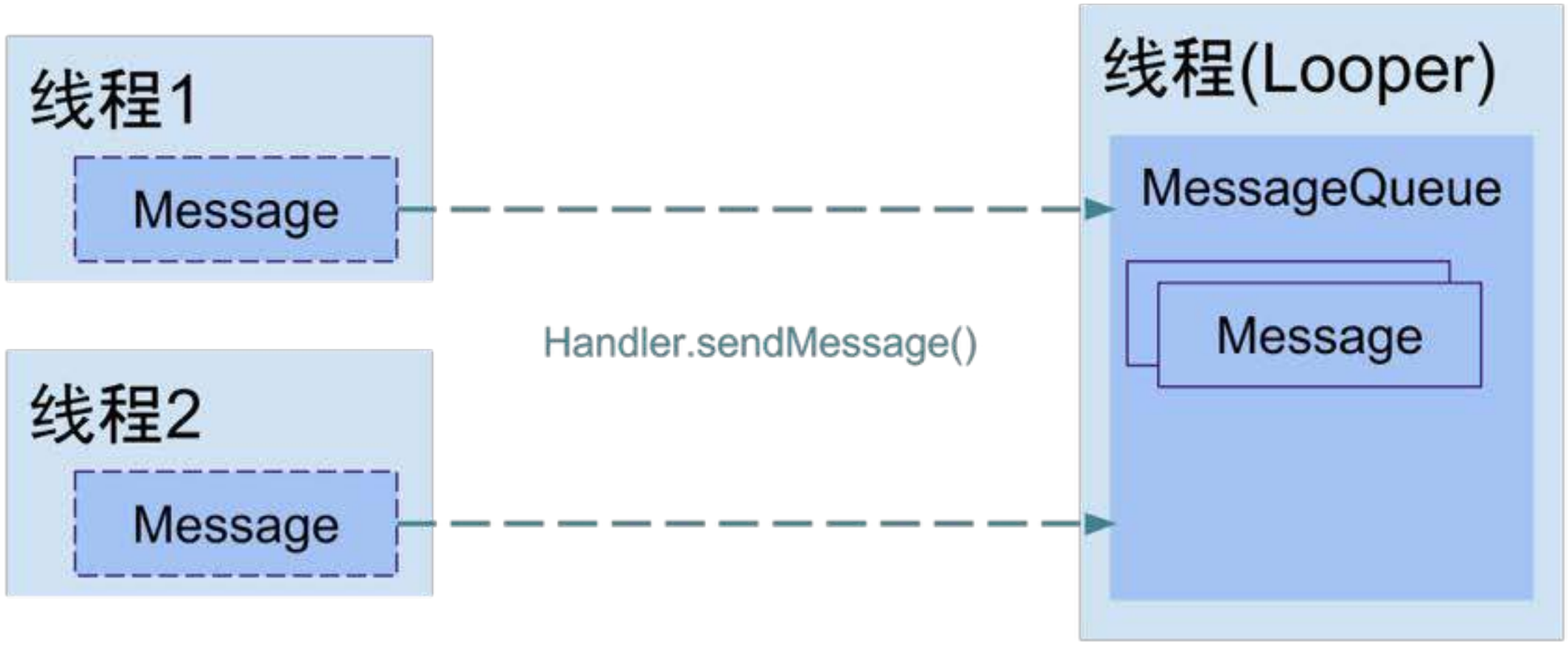

BC_REQUEST_DEATH_NOTIFICATION
BW_DEAD_BINDER
BR_DEAD_BINDER
BC_CLEAR_DEATH_NOTIFICATION
BW_DEAD_BINDER_AND_CLEAR
BC_DEAD_BINDER_DONE
BW_CLEAR_DEATH_NOTIFICATIO N
BR_CLEAR_DEATH_NOTIFICATION_DONE
binder_thread_write()
binder_thread_read()
IPC.executeCommand()
Gityuan.com

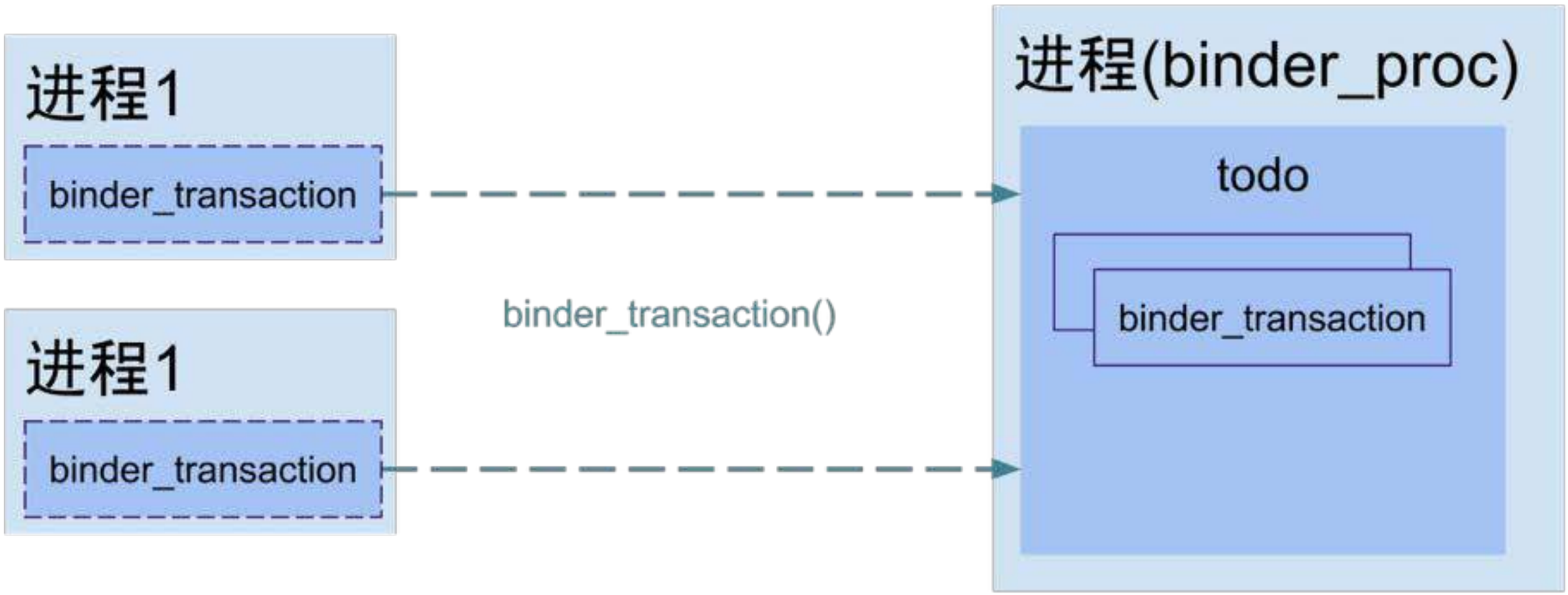
提纲

- Binder概述
- Binder设计思想
- Binder通信协议
- Binder死亡通知
- Binder线程池管理


IPCThreadState
进程1
进程2
T1
T2
T3
T4
T5
T6
ProcessState
ProcessState
用户空间
内核空间
binder_procs
binder_proc
binder_proc
Binder Driver
binder_thread


binder_proc
list_head todo
rb_root threads
binder_transaction
binder_work
list_head entry
type
binder_transaction
binder_work
list_head entry
type
binder_thread
list_head todo
binder_proc * proc
binder_transaction
binder_work
list_head entry
type
binder_transaction
binder_work
list_head entry
type


线程1
Message
线程2
Message
Handler.sendMessage()
线程(Looper)
MessageQueue
Message


进程1
binder_transaction
进程(binder_proc)
todo
binder_transaction
binder_transaction()
进程1
binder_transaction

Thank You