voucher_swap

bazad

TyphoonCon 2019

whoami

Brandon Azad - @_bazad

Google Project Zero

macOS / iOS

Timeline

CVE-2019-6225

Reported on December 7, 2018

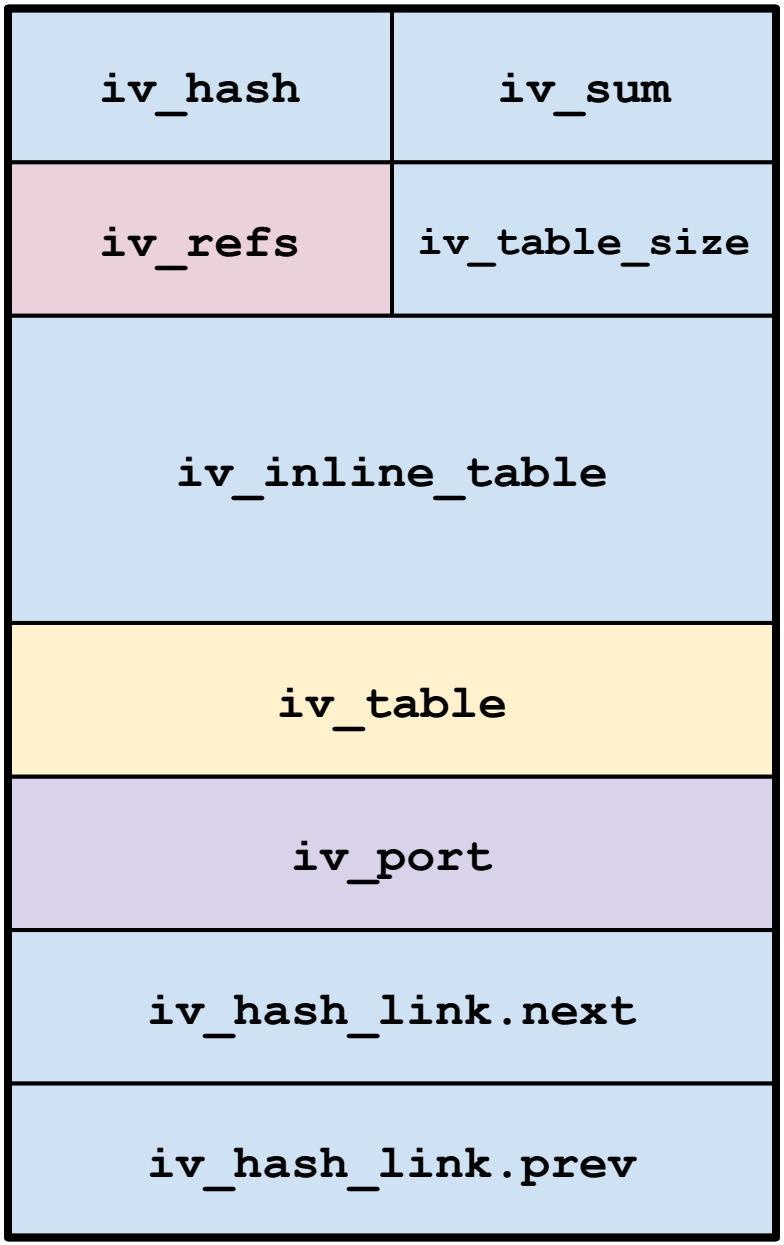
@S0rryMybad exploited for remote jailbreak in Tianfu Cup

Fixed in iOS 12.1.3

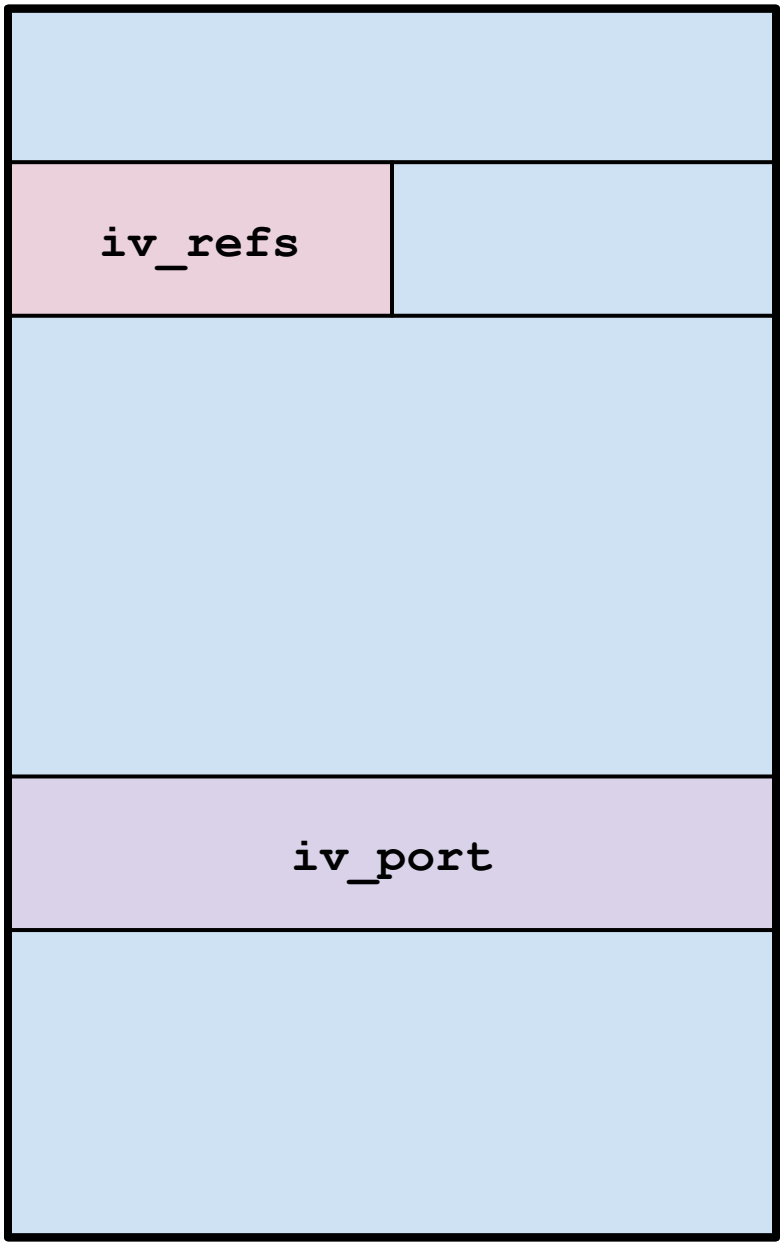
The vulnerability

```
routine task_swap_mach_voucher(
                task            : task_t;
                new_voucher     : ipc_voucher_t;
    inout       old_voucher     : ipc_voucher_t);
```

IPC voucher


iv_hash
iv_sum
iv_refs
iv_table_size
iv_inline_table
iv_table
iv_port
iv_hash_link.next
iv_hash_link.prev

IPC voucher



```
mig_internal novalue _Xtask_swap_mach_voucher
        (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
...
    task = convert_port_to_task(In0P->Head.msgh_request_port);
    new_voucher = convert_port_to_voucher(In0P->new_voucher.name);
    old_voucher = convert_port_to_voucher(In0P->old_voucher.name);

    RetCode = task_swap_mach_voucher(task, new_voucher, &old_voucher);

    ipc_voucher_release(new_voucher);

    task_deallocate(task);
...
    OutP->old_voucher.name = convert_voucher_to_port(old_voucher);
...
}
```

```
mig_internal novalue _Xtask_swap_mach_voucher
        (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
...
    task = convert_port_to_task(In0P->Head.msgh_request_port);
    new_voucher = convert_port_to_voucher(In0P->new_voucher.name);
    old_voucher = convert_port_to_voucher(In0P->old_voucher.name);

    RetCode = task_swap_mach_voucher(task, new_voucher, &old_voucher);

    ipc_voucher_release(new_voucher);

    task_deallocate(task);
...
    OutP->old_voucher.name = convert_voucher_to_port(old_voucher);
...
}
```

```
mig_internal novalue _Xtask_swap_mach_voucher
        (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
...
    task = convert_port_to_task(In0P->Head.msgh_request_port);
    new_voucher = convert_port_to_voucher(In0P->new_voucher.name);
    old_voucher = convert_port_to_voucher(In0P->old_voucher.name);

    RetCode = task_swap_mach_voucher(task, new_voucher, &old_voucher);

    ipc_voucher_release(new_voucher);

    task_deallocate(task);
...
    OutP->old_voucher.name = convert_voucher_to_port(old_voucher);
...
}
```

```
mig_internal novalue _Xtask_swap_mach_voucher
        (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
...
    task = convert_port_to_task(In0P->Head.msgh_request_port);
    new_voucher = convert_port_to_voucher(In0P->new_voucher.name);
    old_voucher = convert_port_to_voucher(In0P->old_voucher.name);

    RetCode = task_swap_mach_voucher(task, new_voucher, &old_voucher);

    ipc_voucher_release(new_voucher);

    task_deallocate(task);
...
    OutP->old_voucher.name = convert_voucher_to_port(old_voucher);
...
}
```

```
kern_return_t
task_swap_mach_voucher(
        task_t          task,
        ipc_voucher_t   new_voucher,
        ipc_voucher_t   *in_out_old_voucher)
{
    if (TASK_NULL == task)
        return KERN_INVALID_TASK;

    *in_out_old_voucher = new_voucher;
    return KERN_SUCCESS;
}
```

```
kern_return_t
task_swap_mach_voucher(
        task_t          task,
        ipc_voucher_t   new_voucher,
        ipc_voucher_t   *in_out_old_voucher)
{
    if (TASK_NULL == task)
        return KERN_INVALID_TASK;

    *in_out_old_voucher = new_voucher;
    return KERN_SUCCESS;
}
```

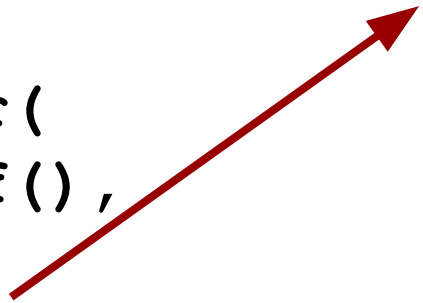
over-released

leaks a ref

```
kern_return_t
task_swap_mach_voucher(
        task_t          task,
        ipc_voucher_t   new_voucher,
        ipc_voucher_t   *in_out_old_voucher)
{
    if (TASK_NULL == task)
        return KERN_INVALID_TASK;

    *in_out_old_voucher = new_voucher;
    return KERN_SUCCESS;
}
```

Manipulating `iv_refs` from userspace

```
void
voucher_inc_iv_refs(mach_port_t voucher)
{
    mach_port_t in_out_voucher = voucher;

    task_swap_mach_voucher(
            mach_task_self(),
            0,
            &in_out_voucher);

    mach_port_deallocate(mach_task_self(),
            in_out_voucher);
}
```

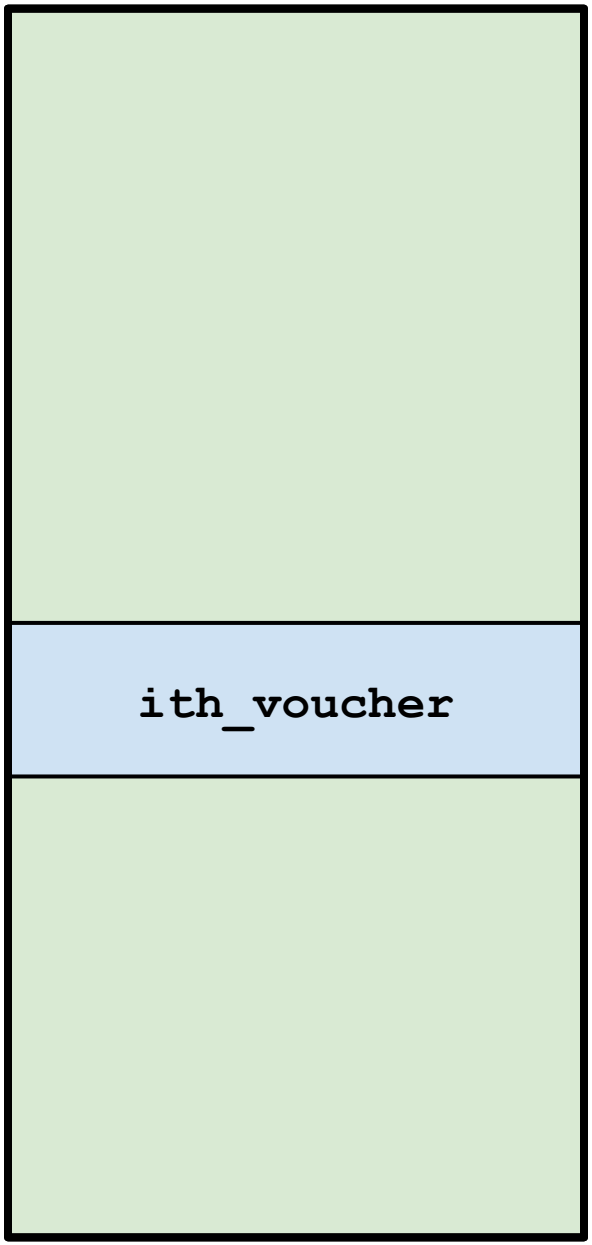


leaks a ref

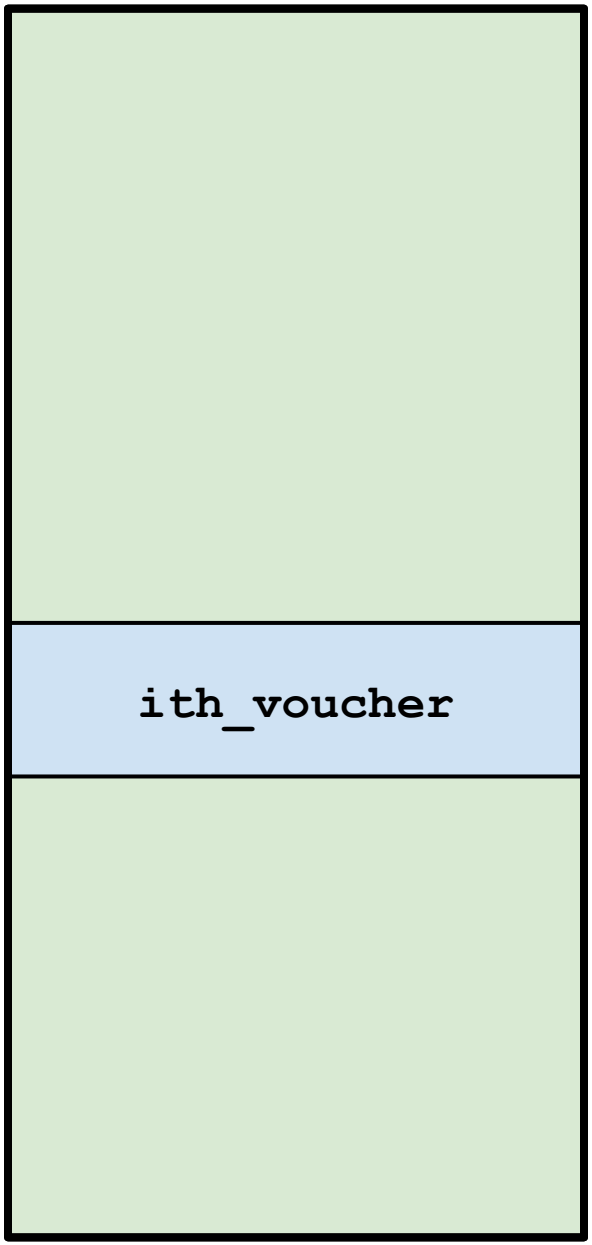
```
void
voucher_dec_iv_refs(mach_port_t voucher)
{
    mach_port_t in_out_voucher = 0;

    task_swap_mach_voucher(
            mach_task_self(),
            voucher,
            &in_out_voucher);

    mach_port_deallocate(mach_task_self(),
            in_out_voucher);
}
```

over-released

After we free a voucher, how do we get it reused?



struct thread



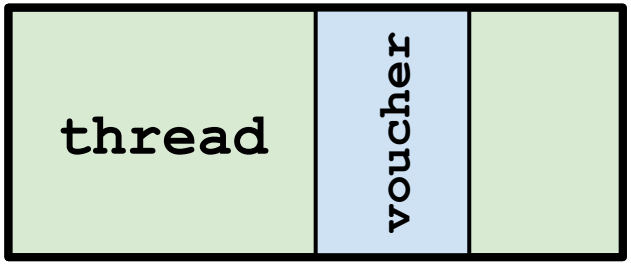
struct thread

thread_set_mach_voucher()

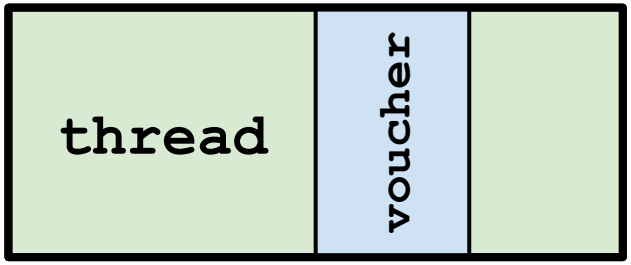
thread_get_mach_voucher()

Initial ideas

32-bit UAF increment

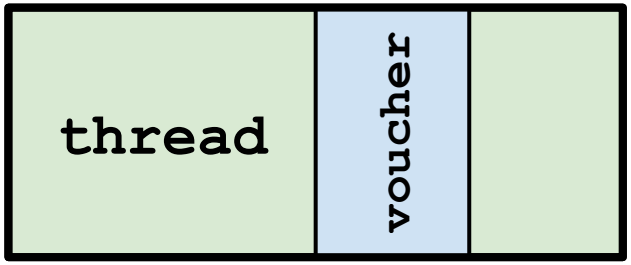


ipc.vouchers



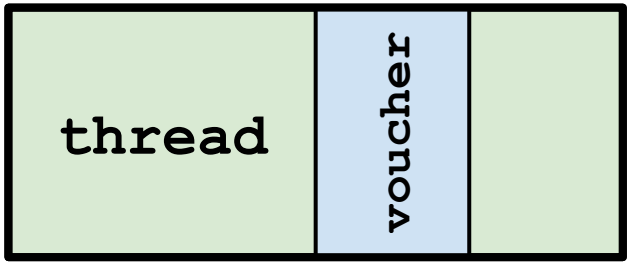
ipc.vouchers

host_create_mach_voucher()



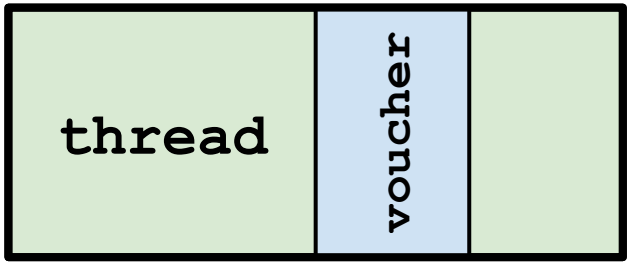
ipc.vouchers

host_create_mach_voucher()



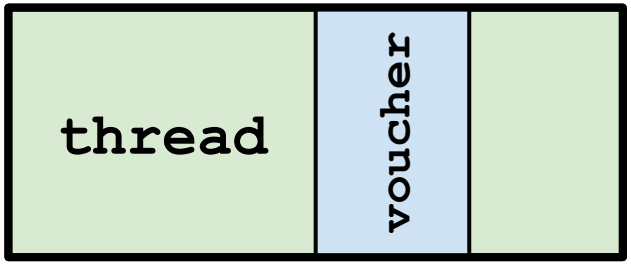
ipc.vouchers

host_create_mach_voucher()



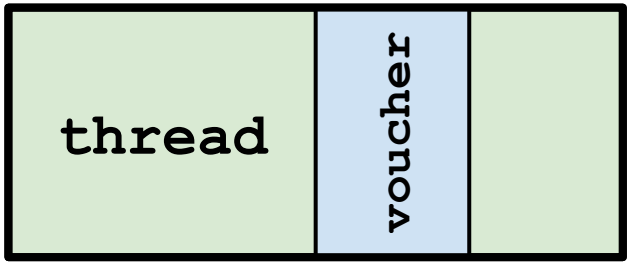
ipc.vouchers

host_create_mach_voucher()



ipc.vouchers

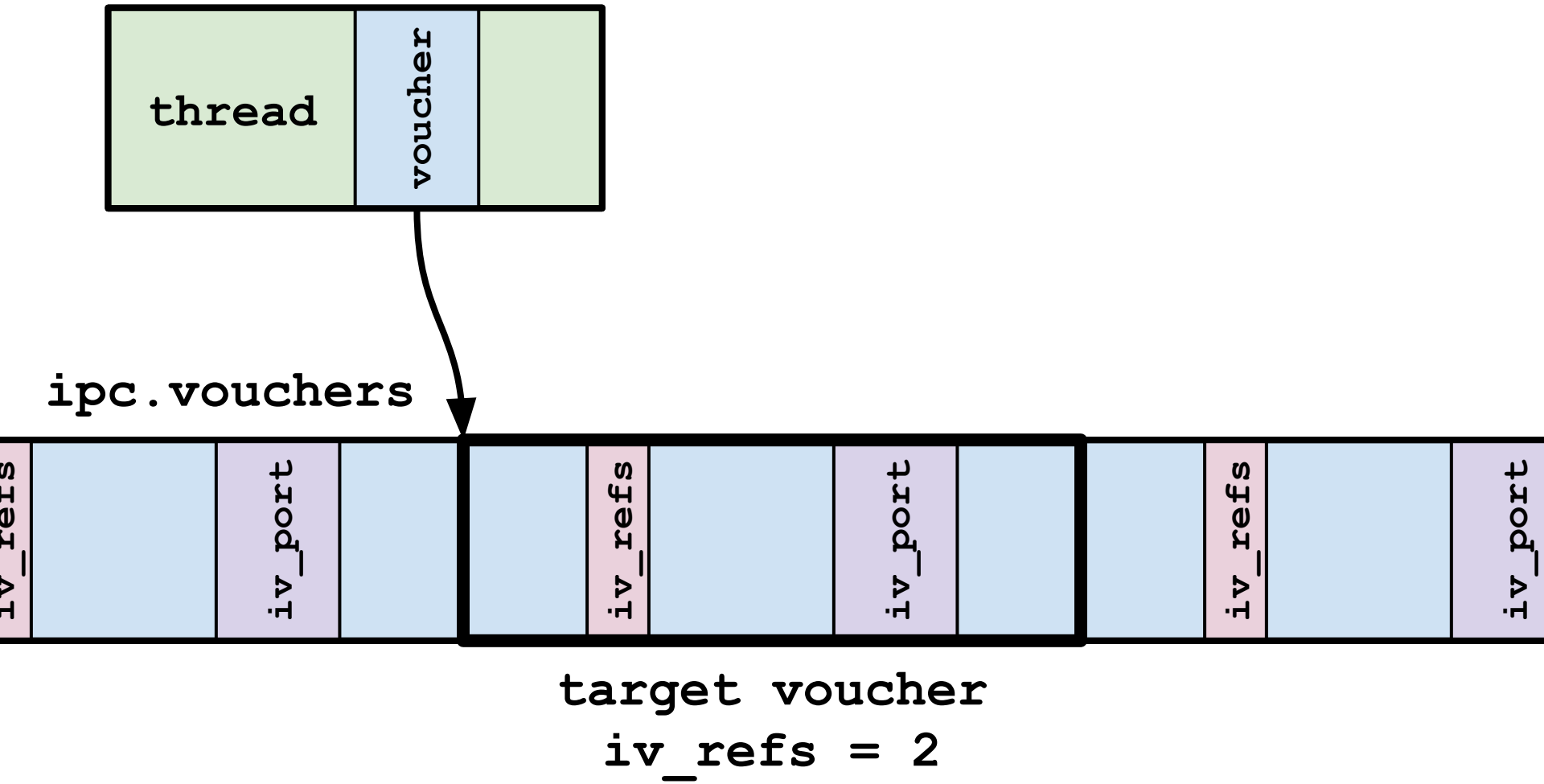
target voucher
iv_refs = 1

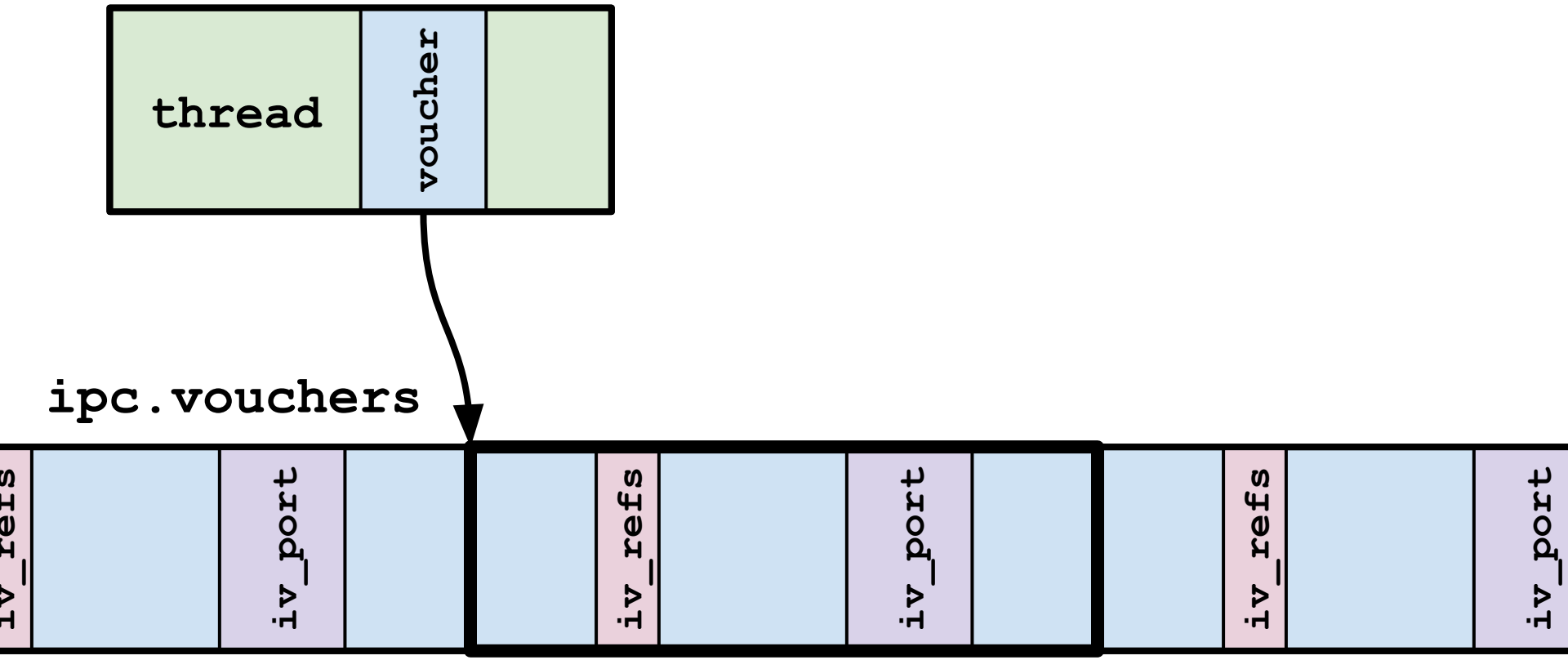


ipc.vouchers

target voucher
iv_refs = 1

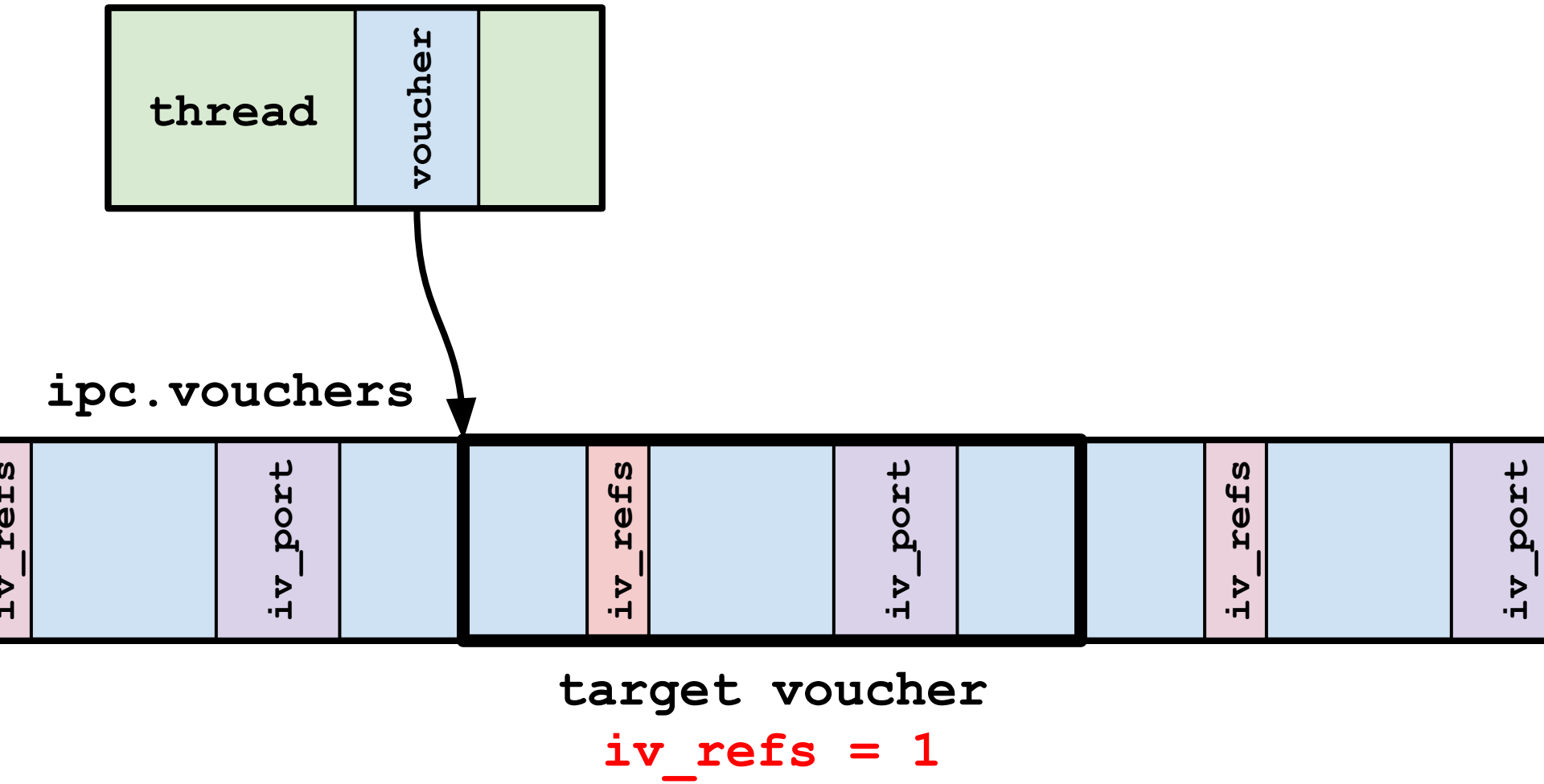
thread_set_mach_voucher()

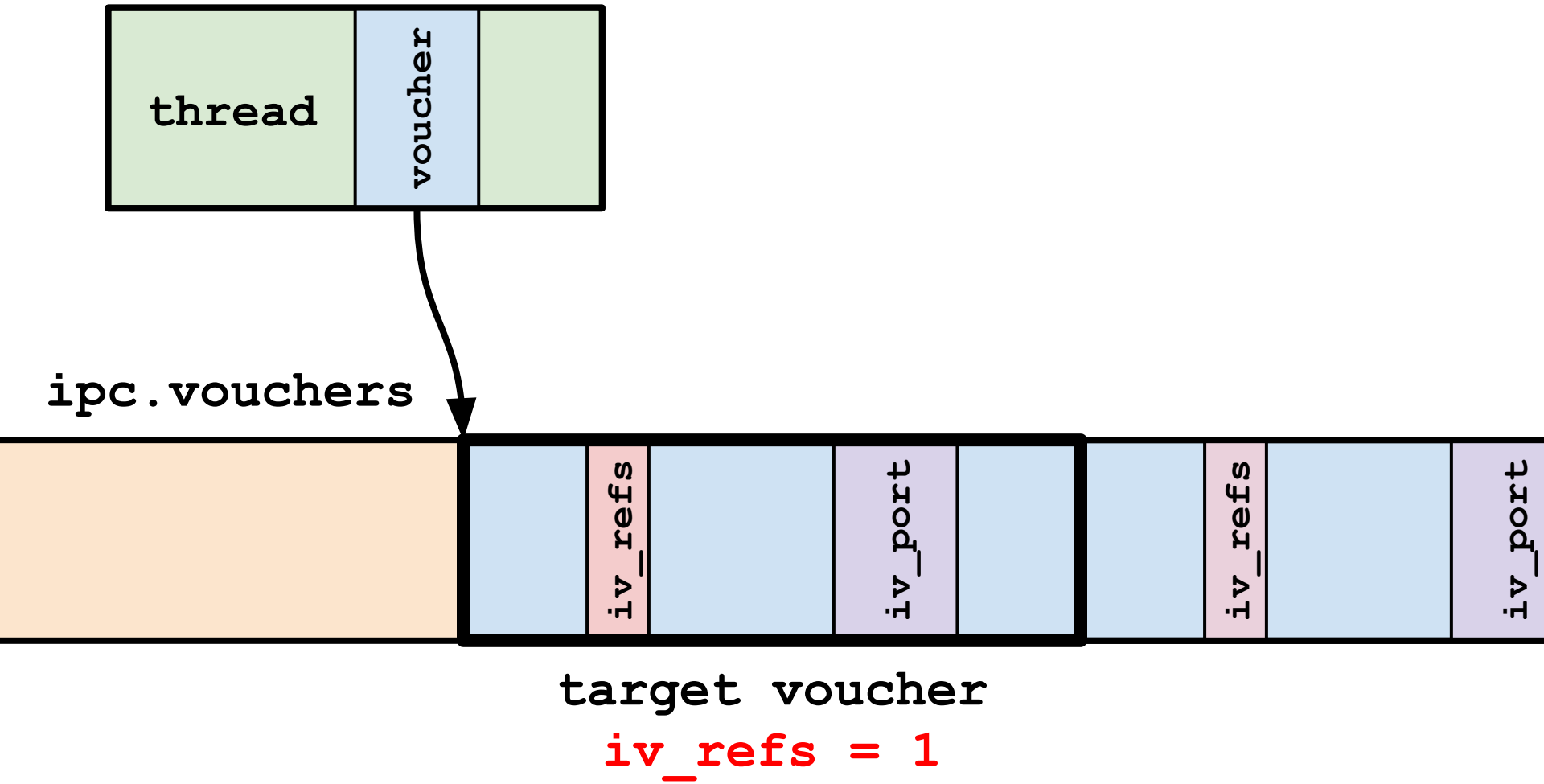

thread
voucher
ipc.vouchers
iv_port
iv_refs
iv_port
iv_refs
iv_port
target voucher
iv_refs = 2

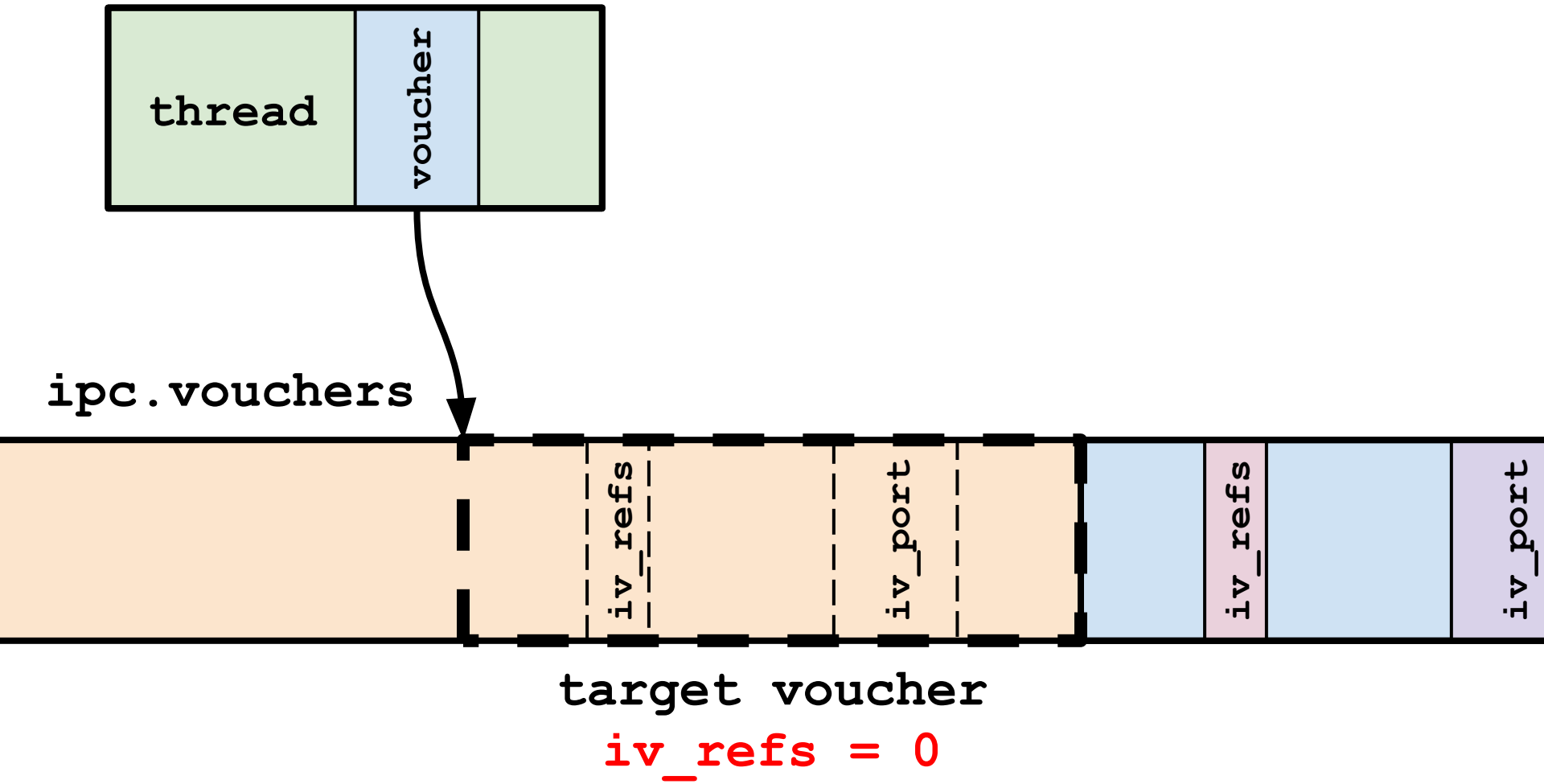


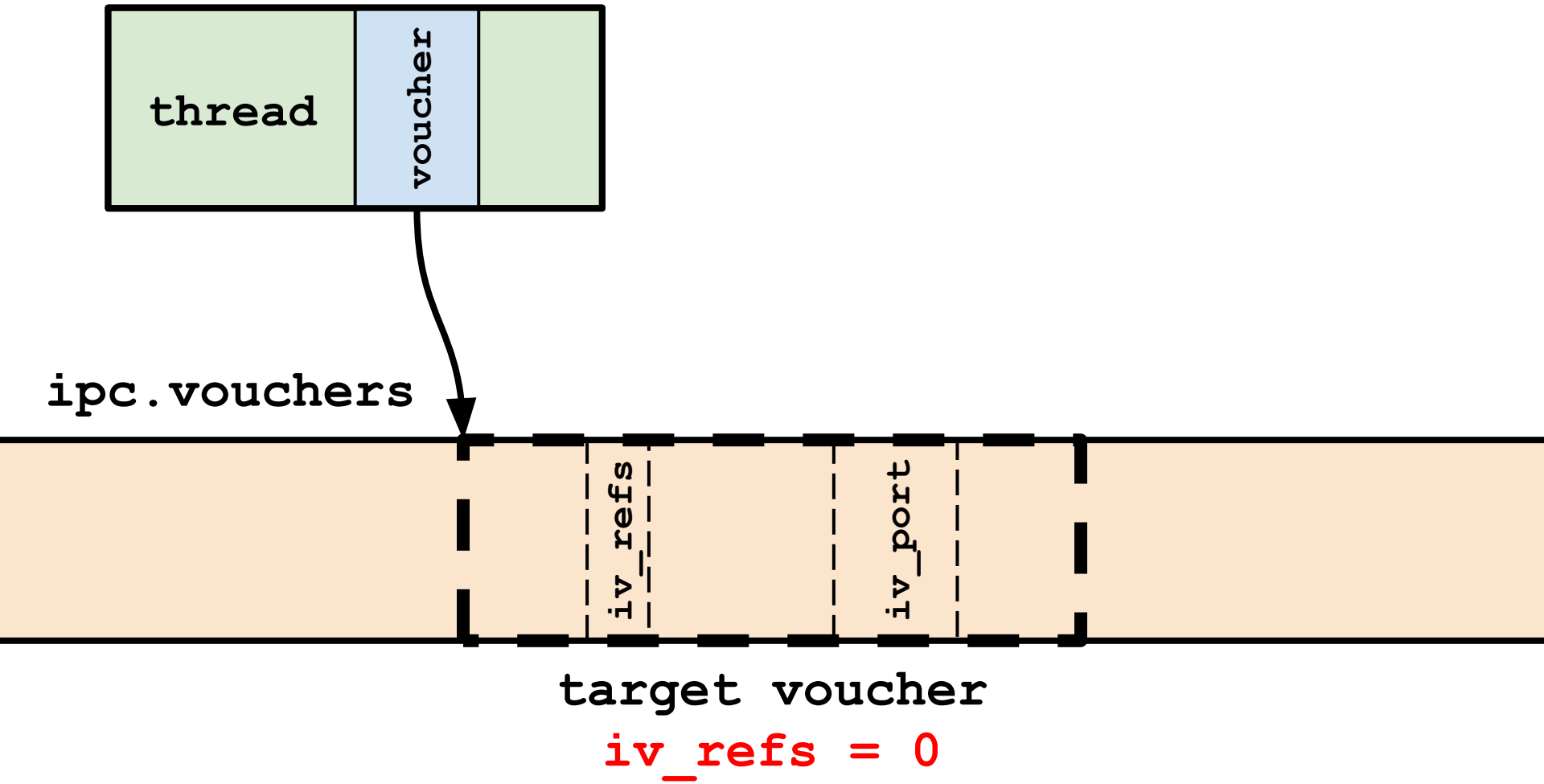
target voucher
iv_refs = 2

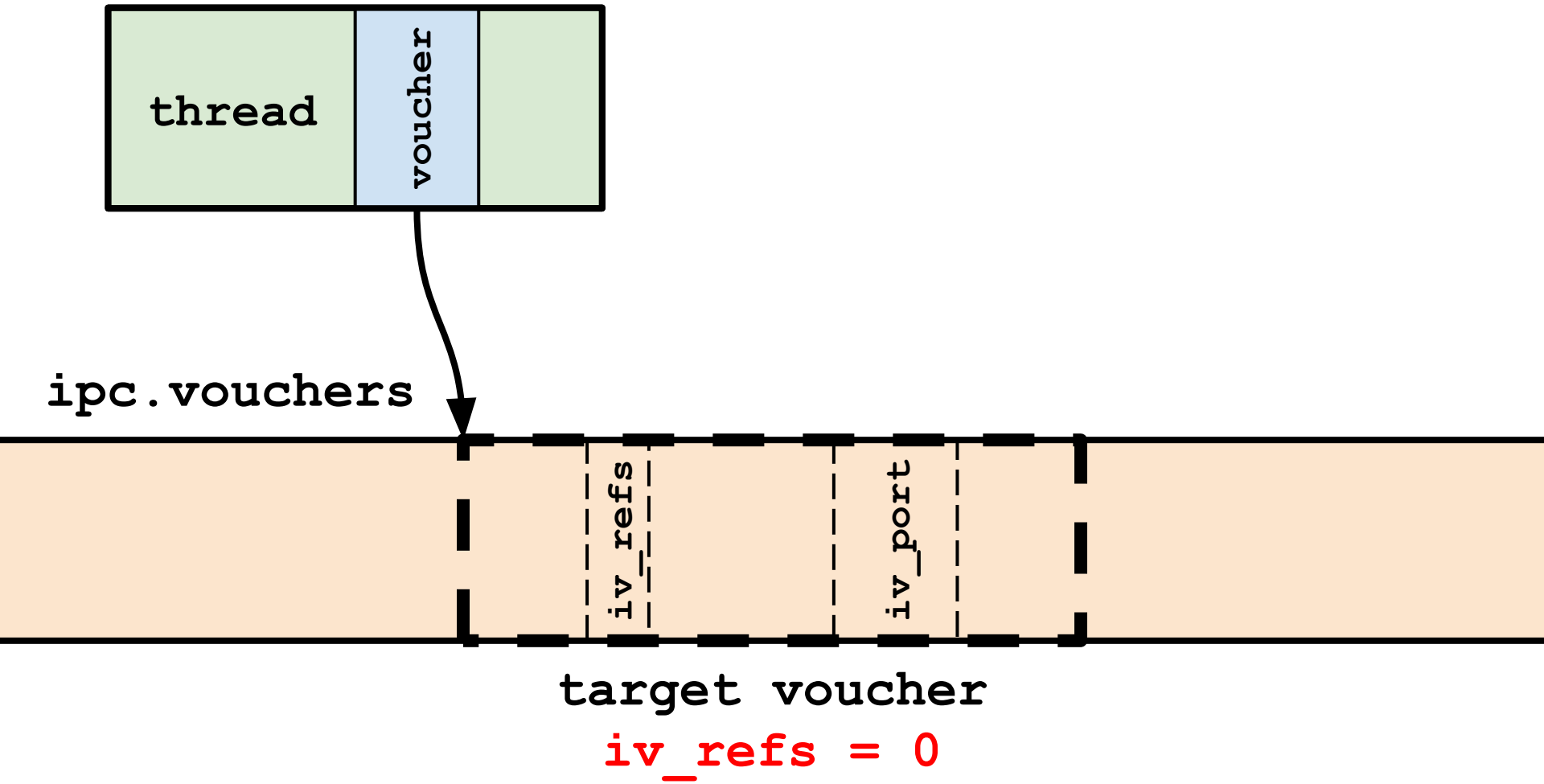
task_swap_mach_voucher()


thread
voucher
ipc.vouchers
iv_port
iv_refs
iv_port
iv_refs
iv_port
target voucher
iv_refs = 1

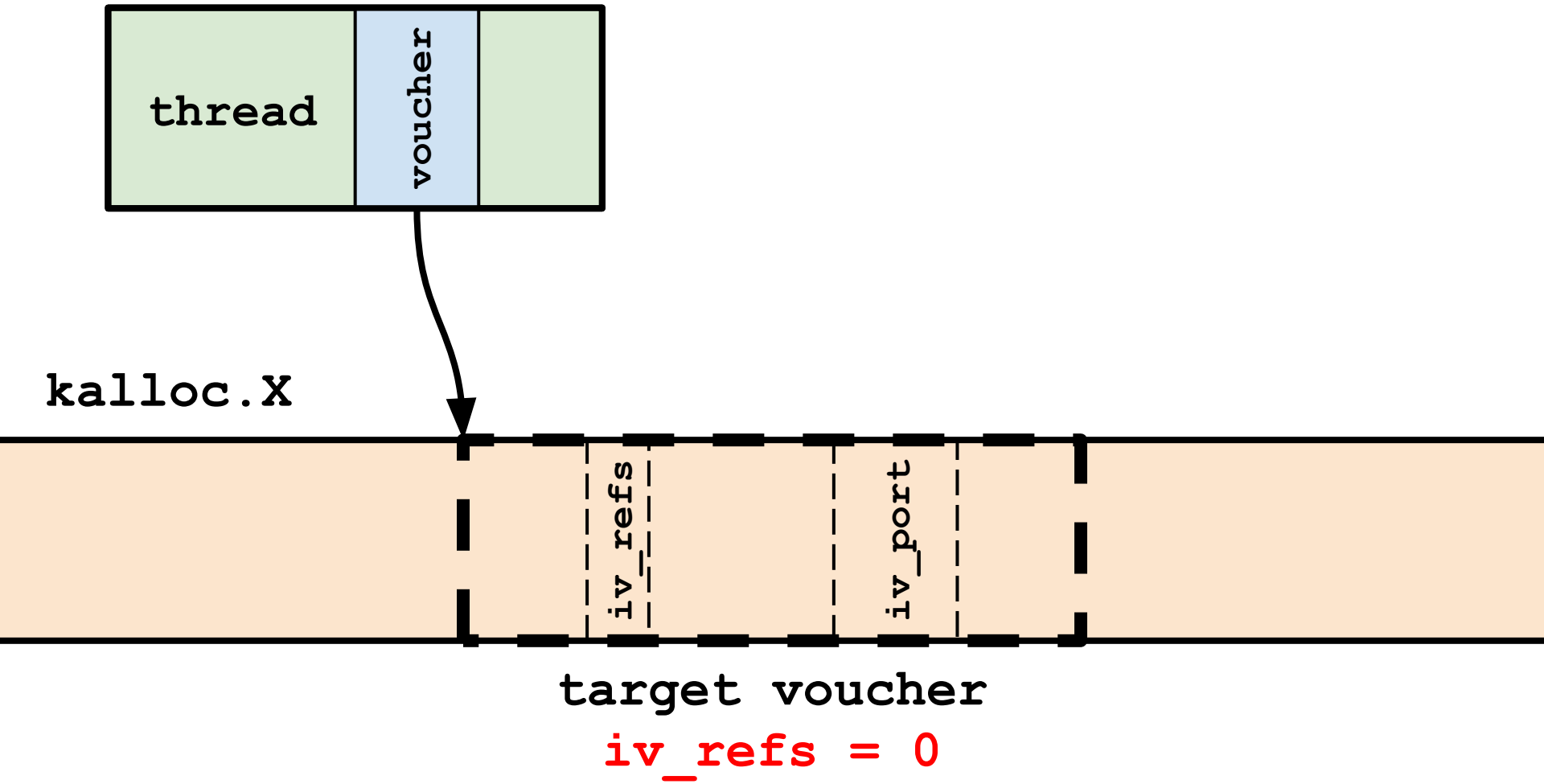

thread
voucher
ipc.vouchers
iv_refs
iv_port
iv_refs
iv_port
target voucher
iv_refs = 1

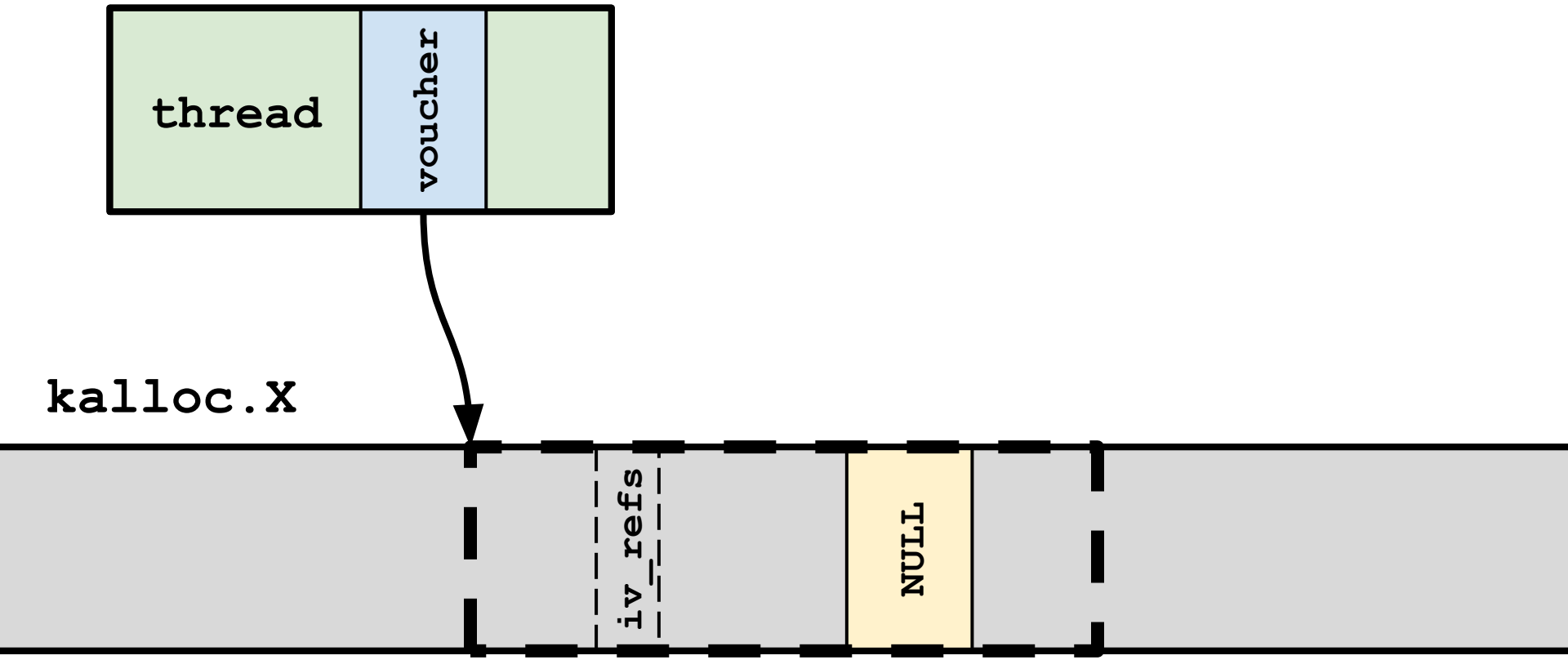

thread
voucher
ipc.vouchers
iv_refs
iv_port
iv_refs
iv_port
target voucher
iv_refs = 0

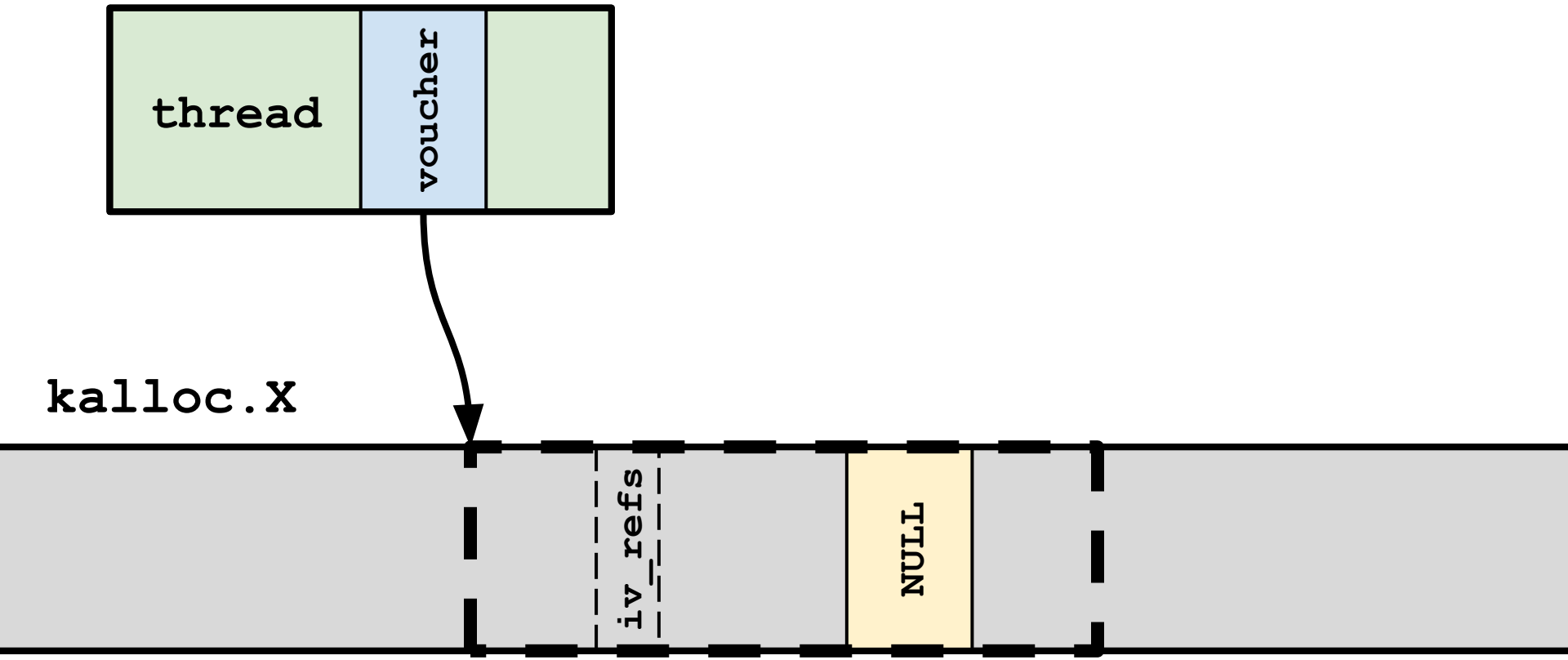

thread
voucher
ipc.vouchers
iv_refs
iv_port
target voucher
iv_refs = 0



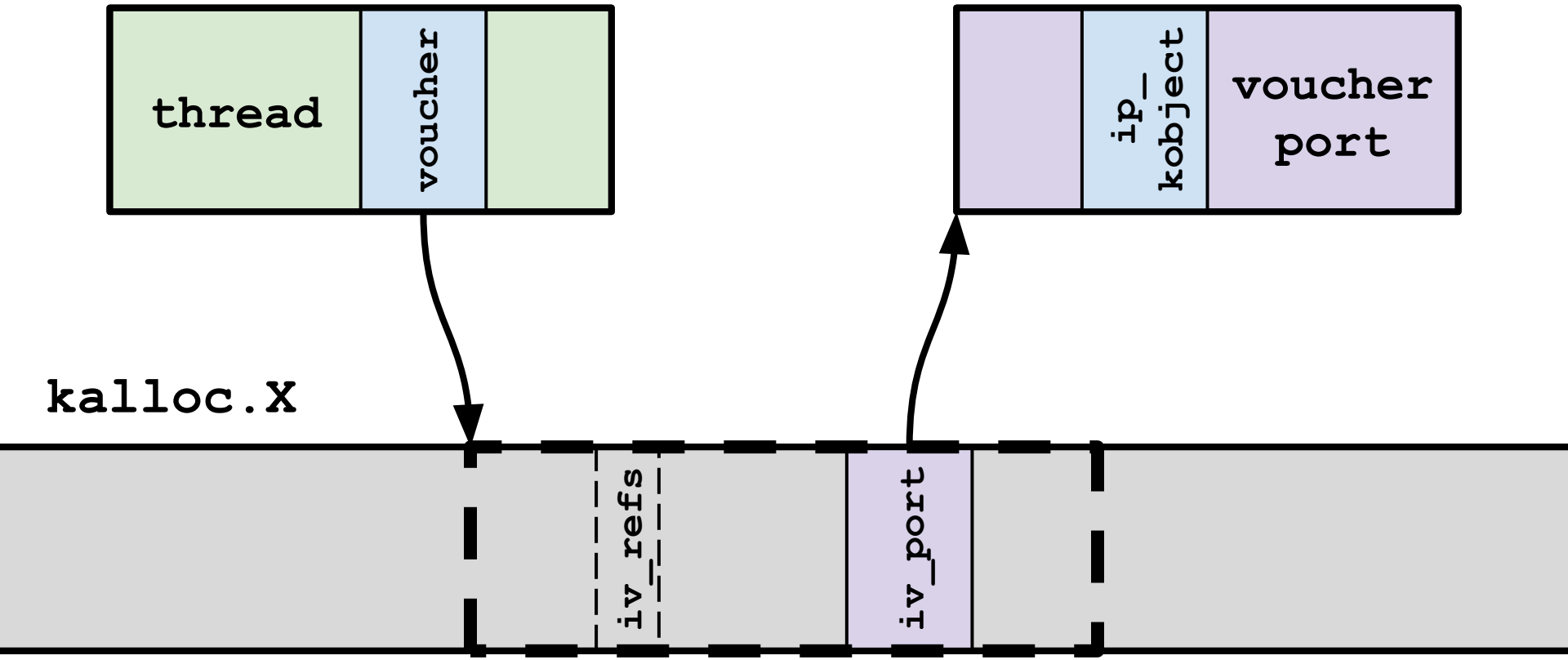
Trigger zone gc

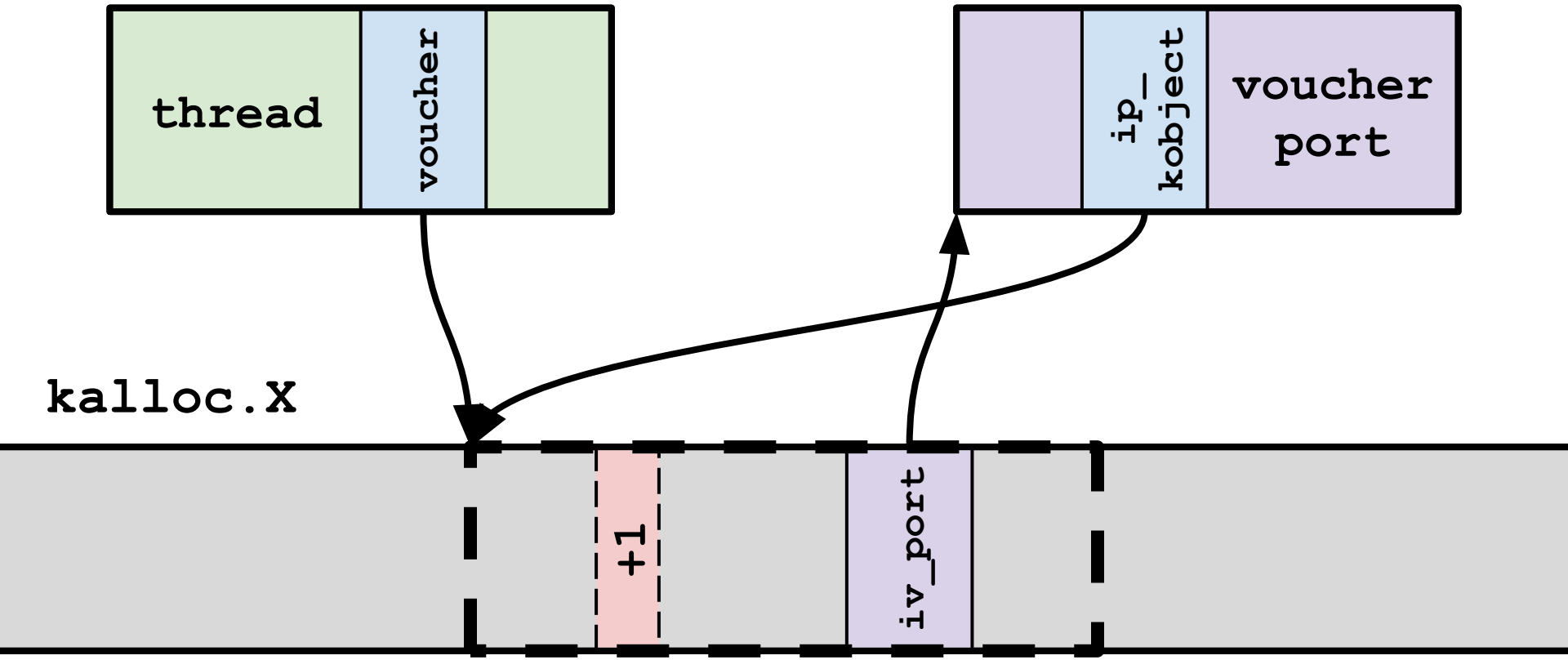

thread
voucher
kalloc.X
iv_refs
iv_port
target voucher
iv_refs = 0

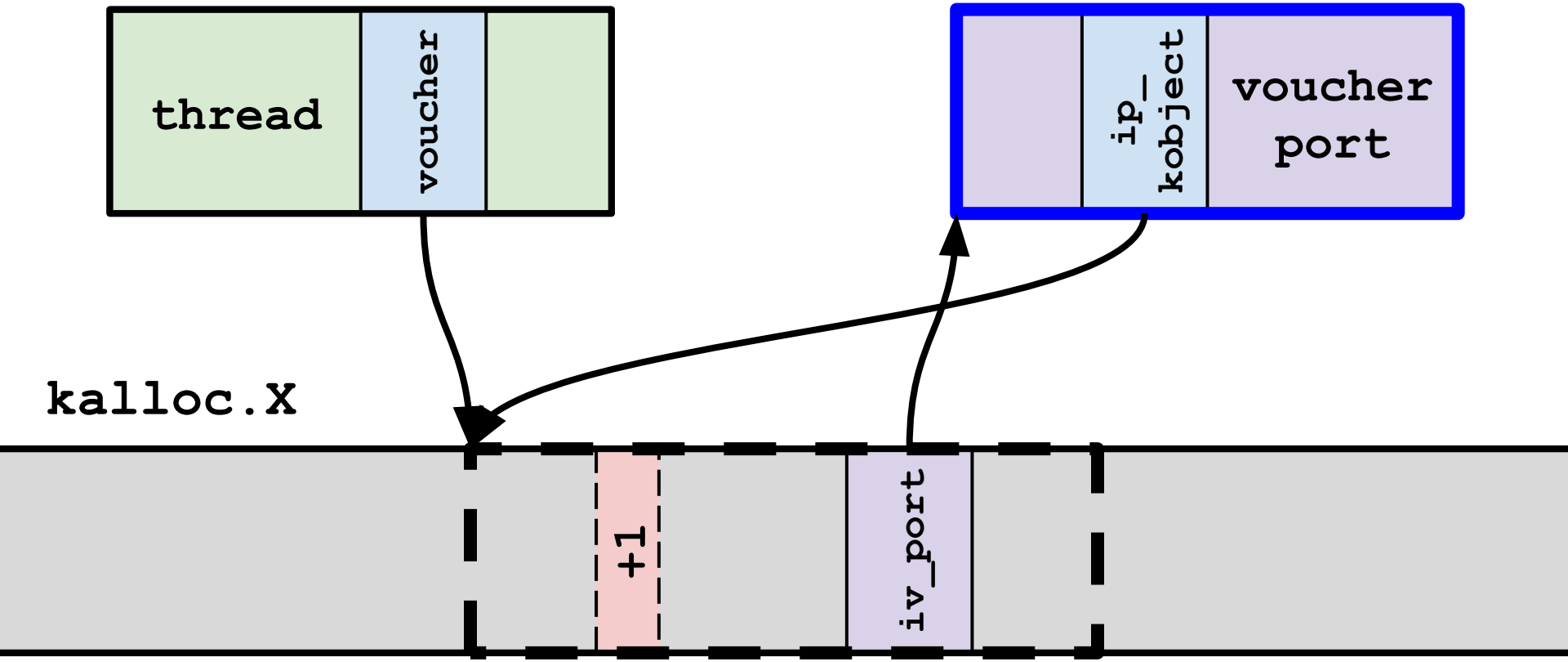

thread
voucher
kalloc.X
iv_refs
NULL

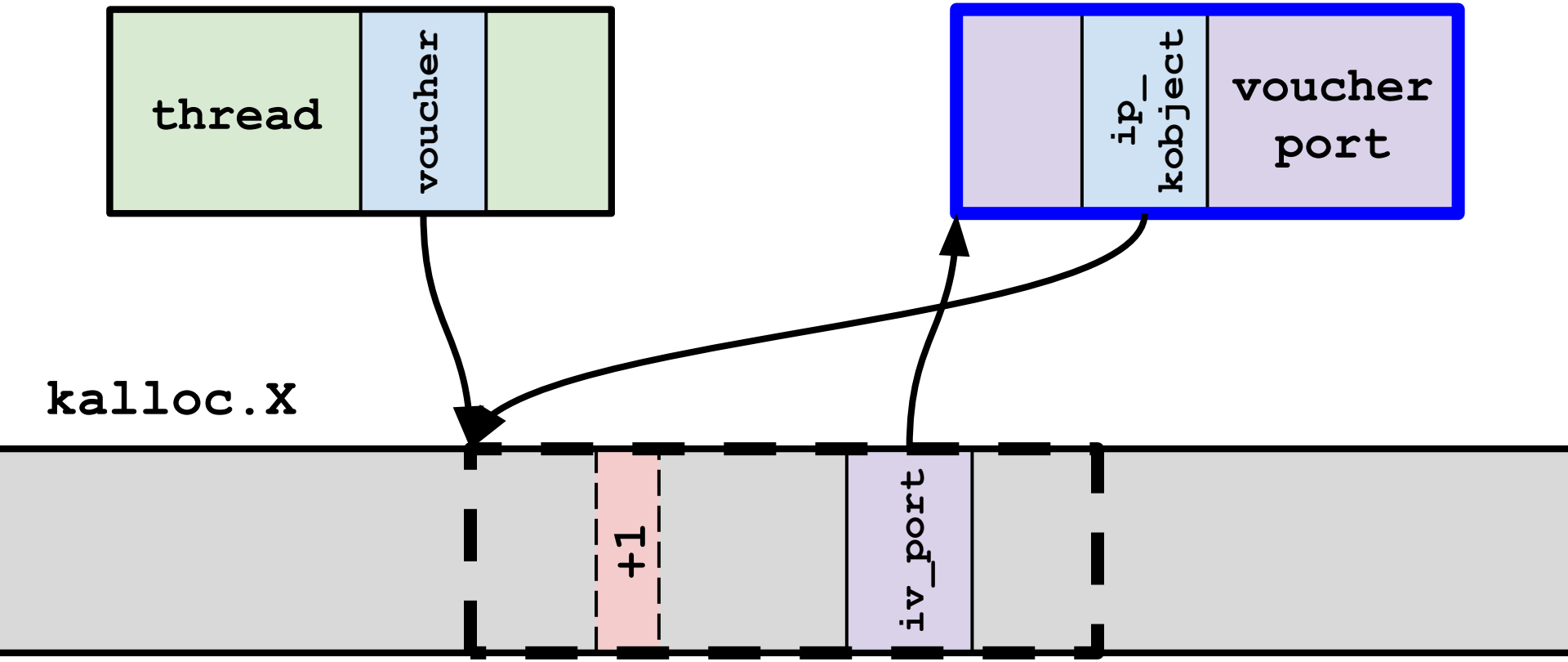


thread_get_mach_voucher()

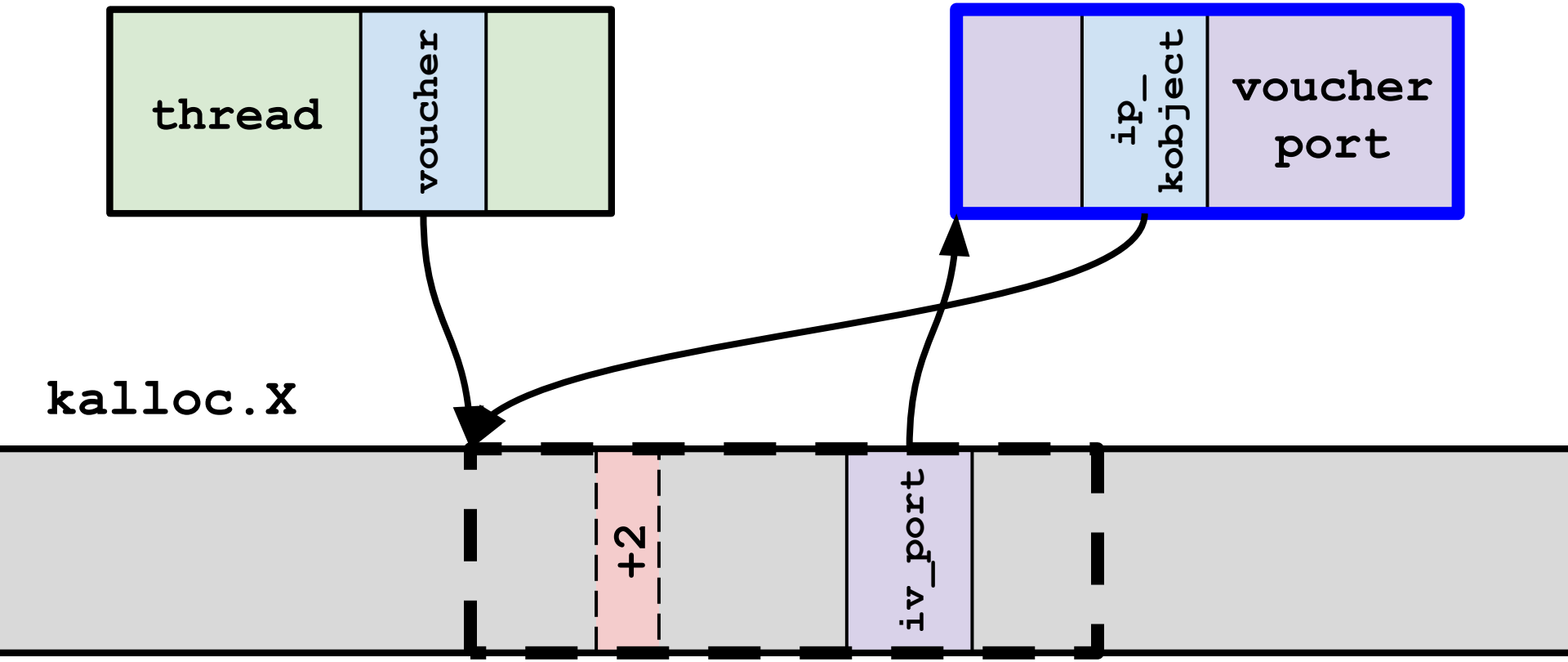

thread
voucher
ip_
kobject
voucher
port
kalloc.X
iv_refs
iv_port

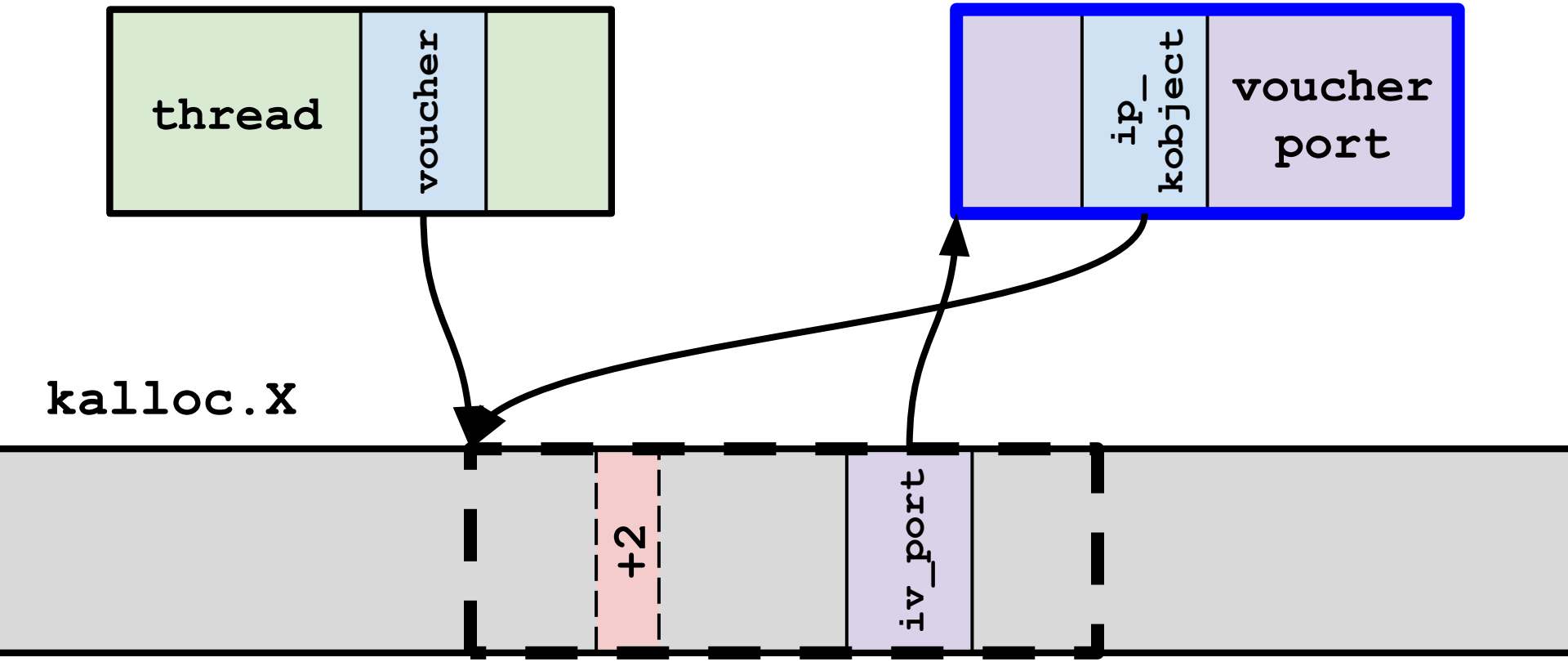

thread
voucher
ip_
kobject
voucher
port
kalloc.X
+1
iv_port


thread
voucher
ip_
kobject
voucher
port
kalloc.X
+1
iv_port

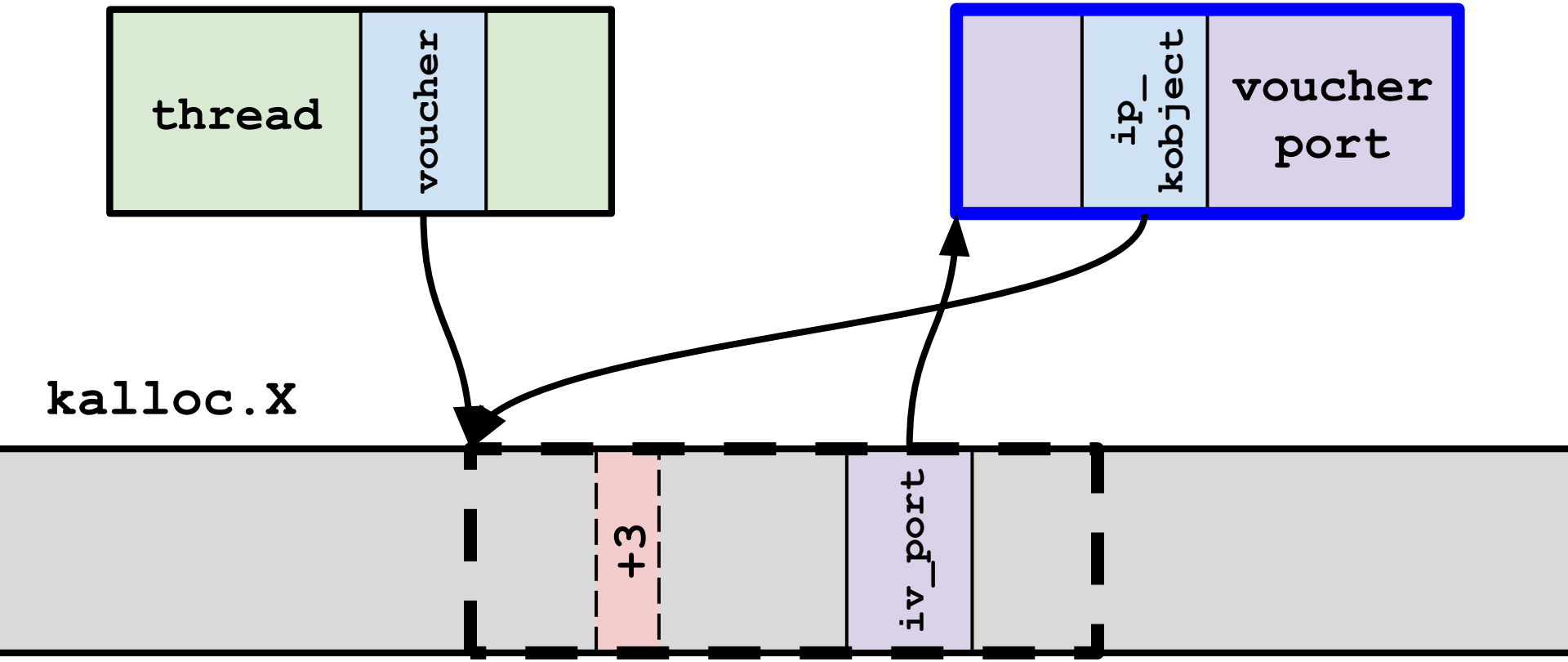


task_swap_mach_voucher()

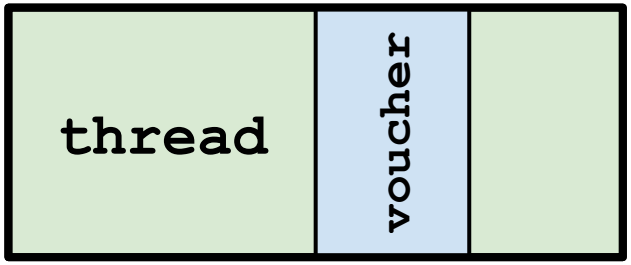

thread
voucher
ip_
kobject
voucher
port
kalloc.X
+2
iv_port



task_swap_mach_voucher()

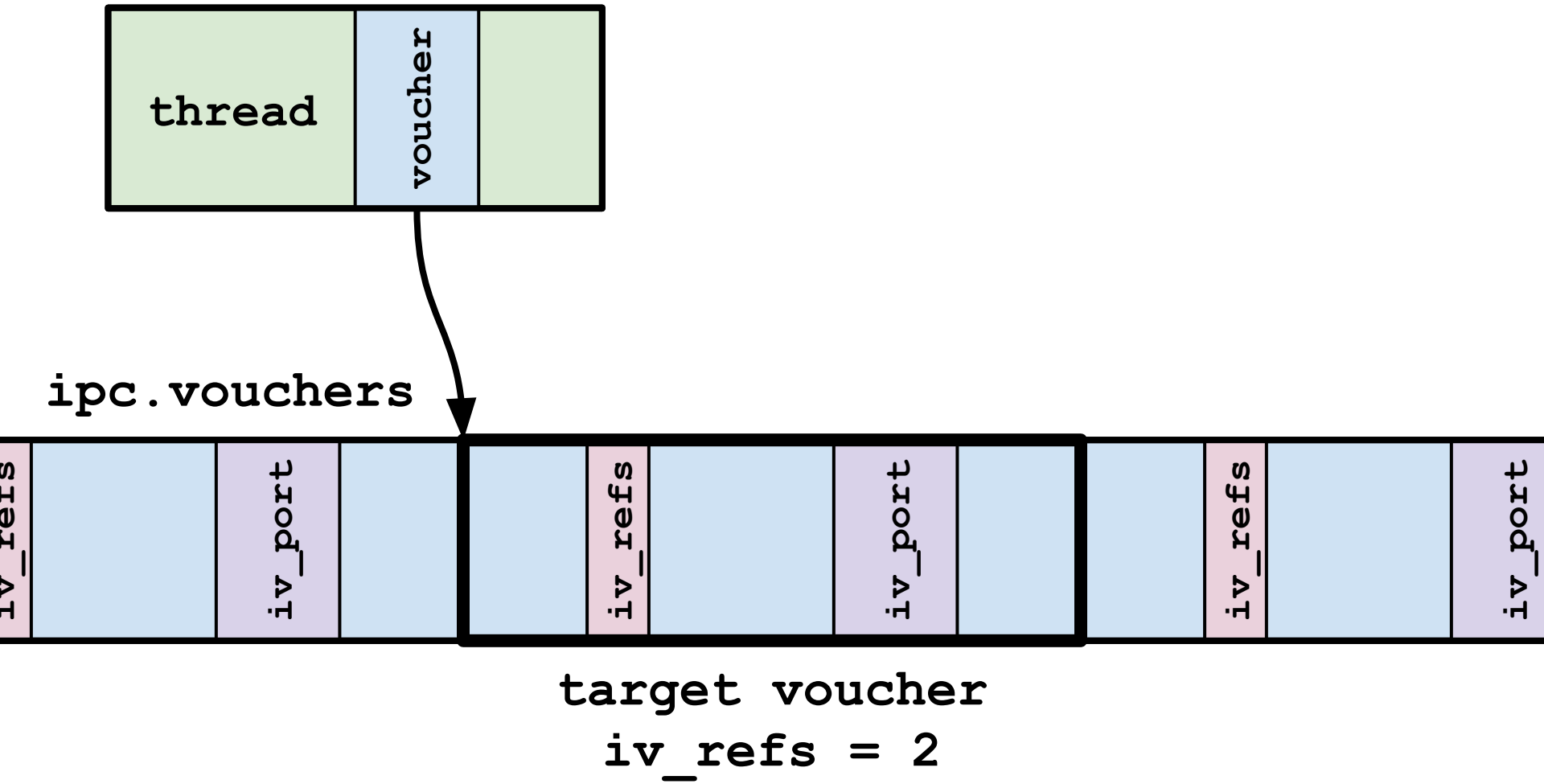

thread
voucher
ip_
kobject
voucher
port
kalloc.X
+3
iv_port

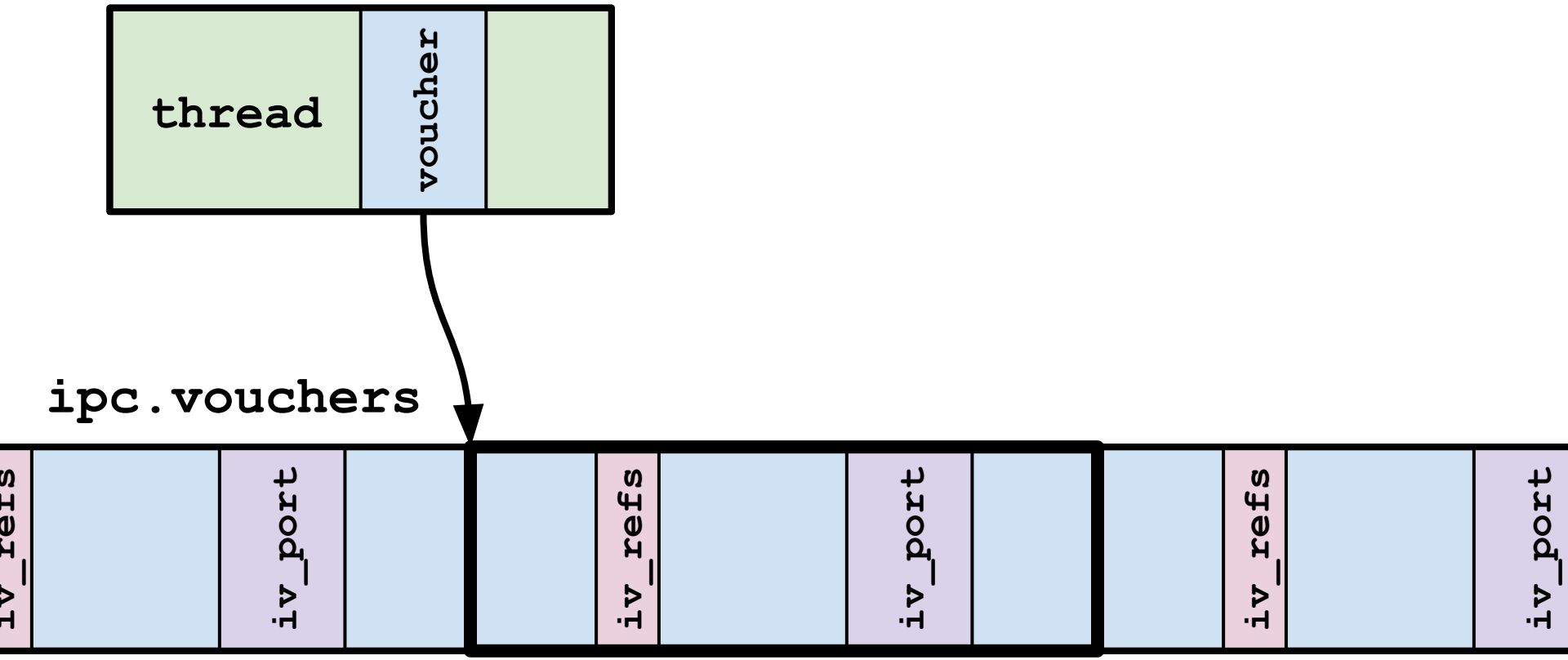
Get a send right to arbitrary memory as a Mach port



ipc.vouchers

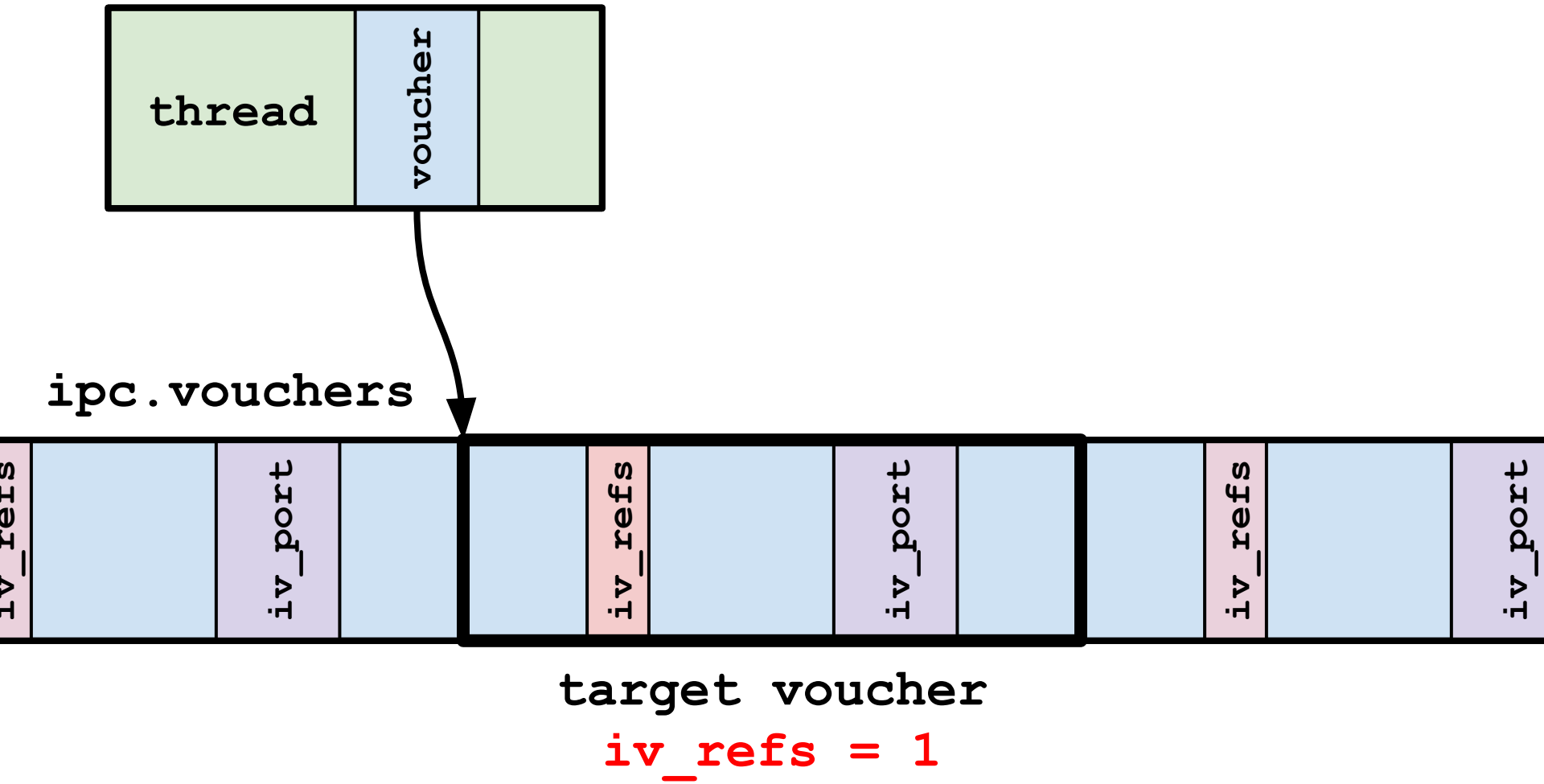
target voucher
iv_refs = 1

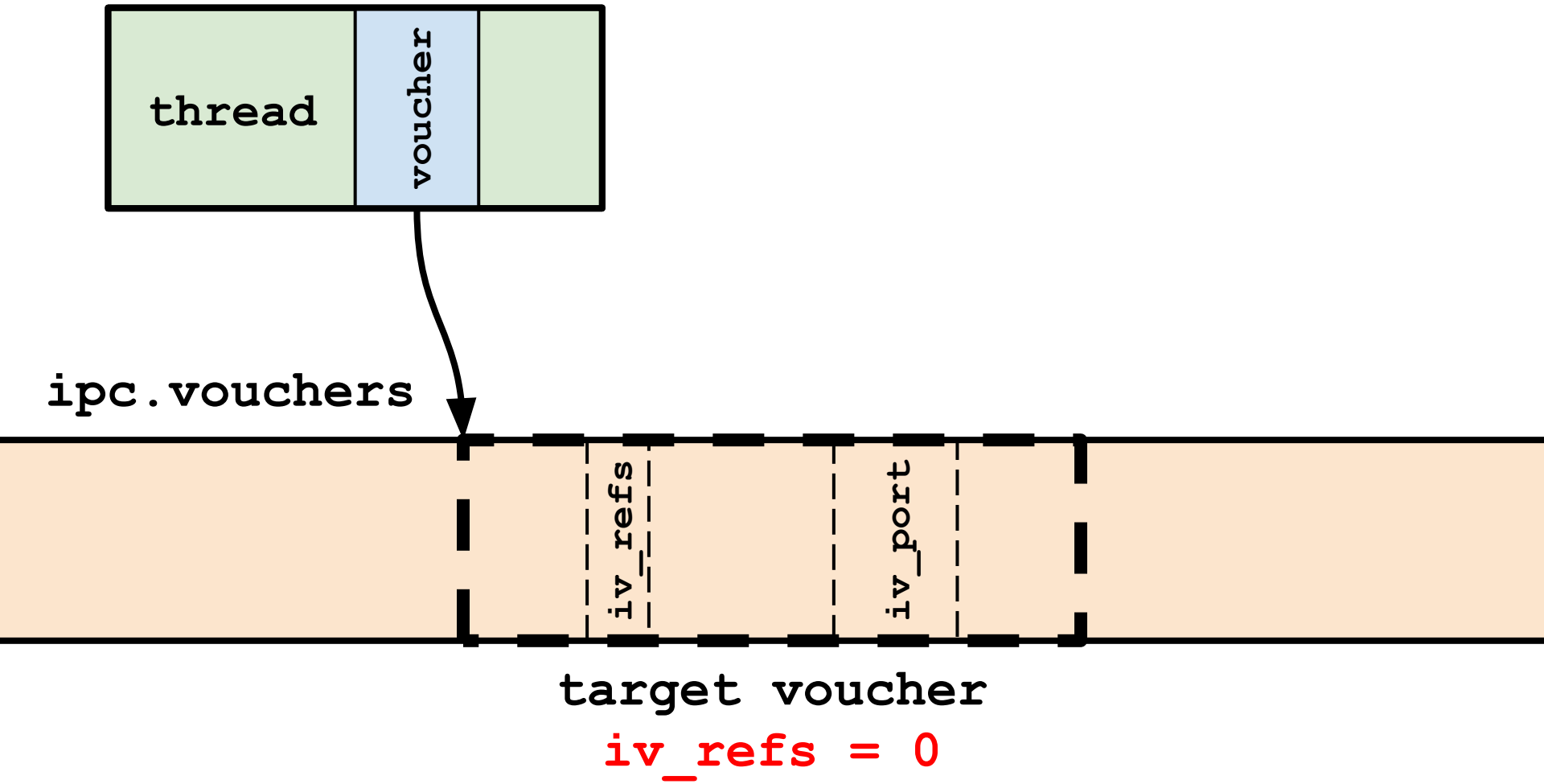

thread
voucher
ipc.vouchers
iv_port
iv_refs
iv_port
iv_refs
iv_port
target voucher
iv_refs = 2

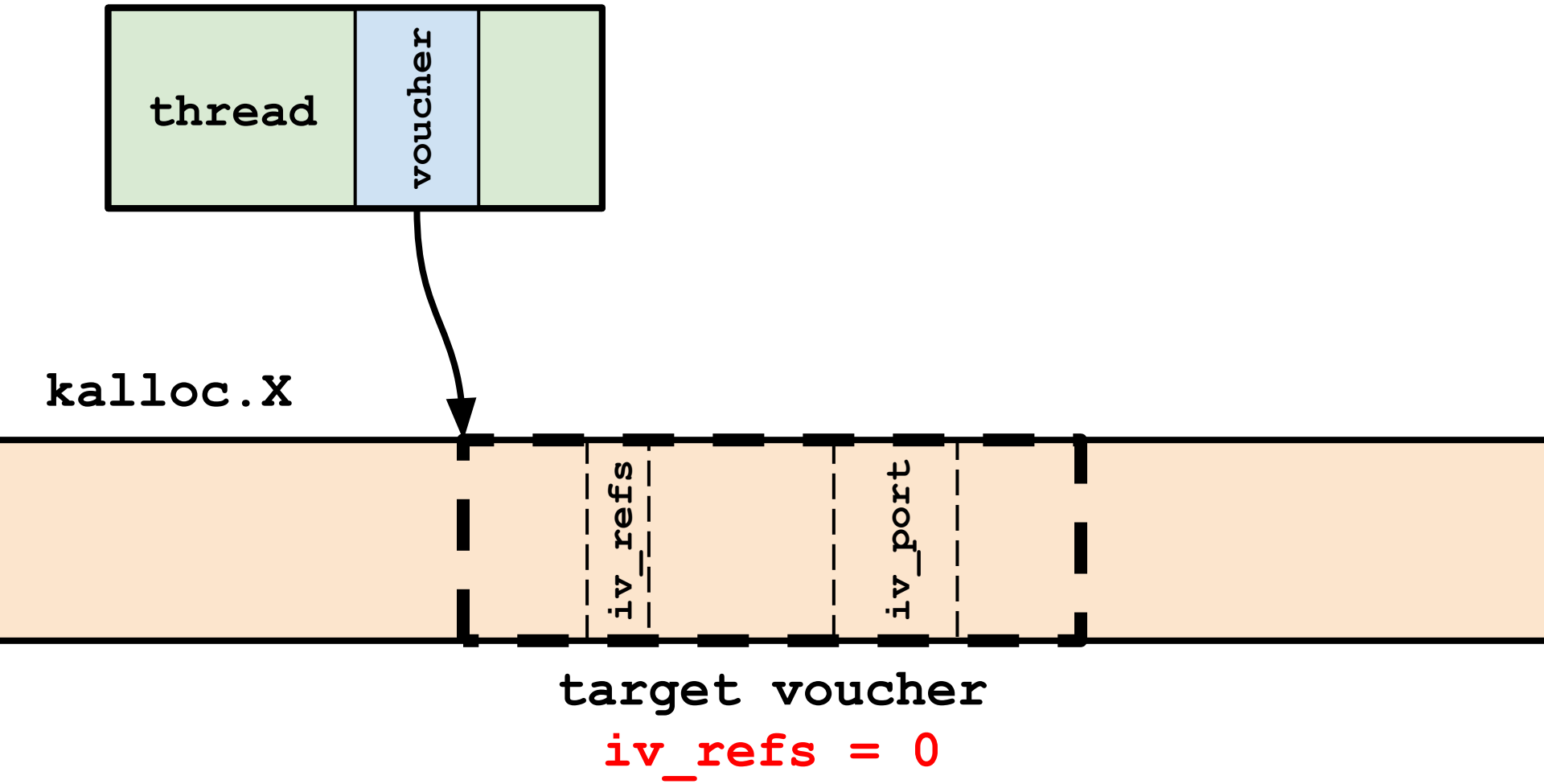


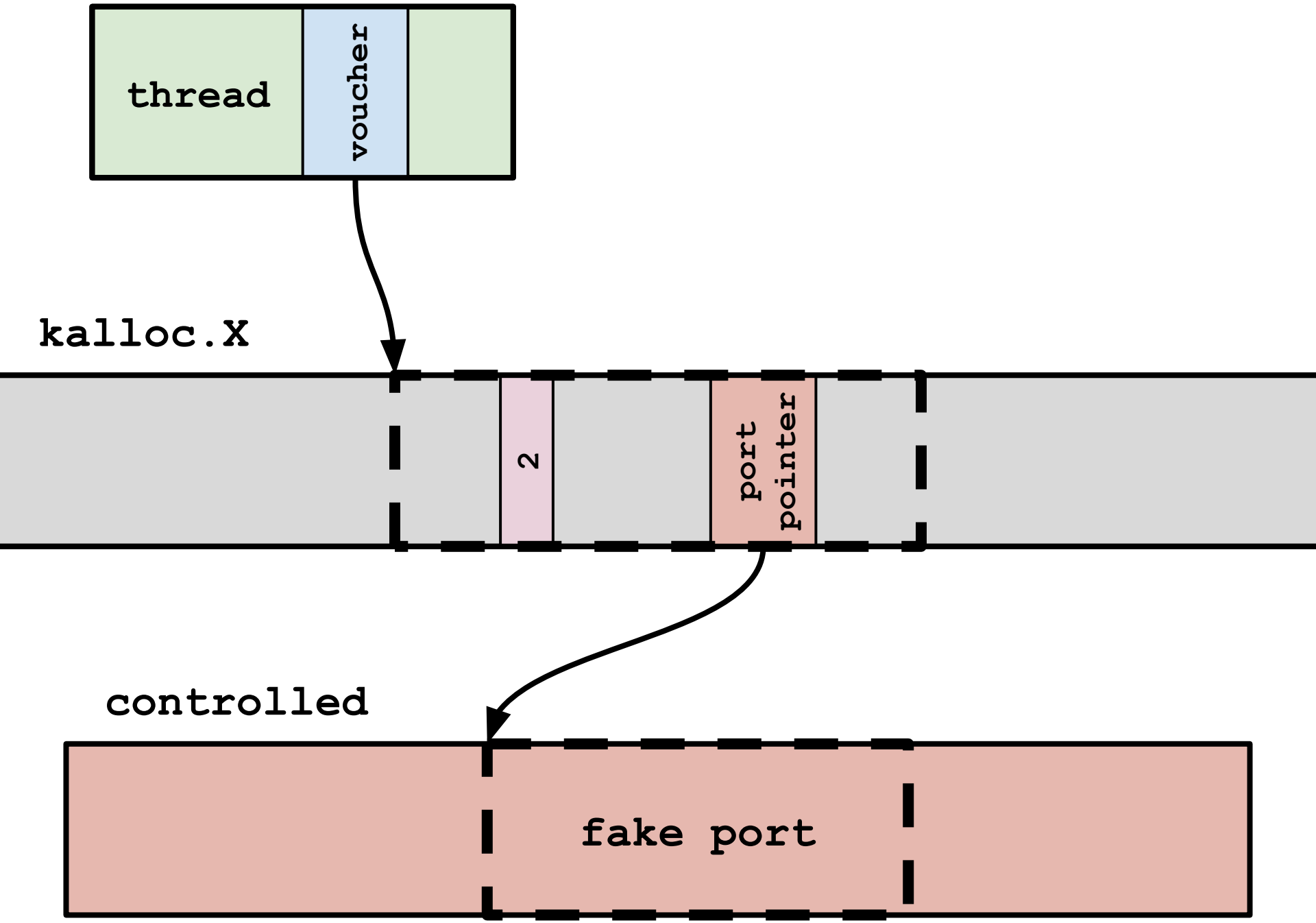
target voucher
iv_refs = 2

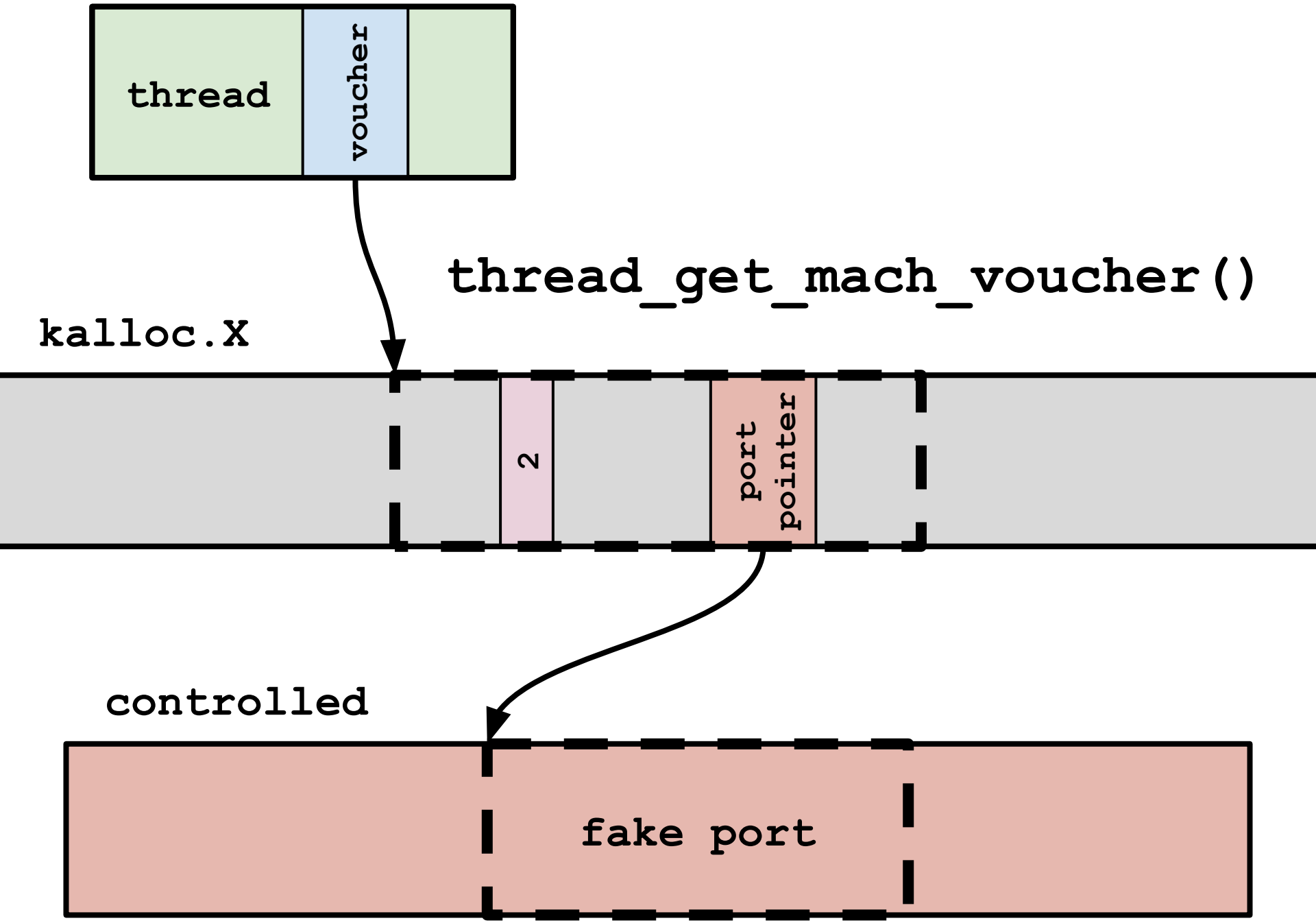
task_swap_mach_voucher()

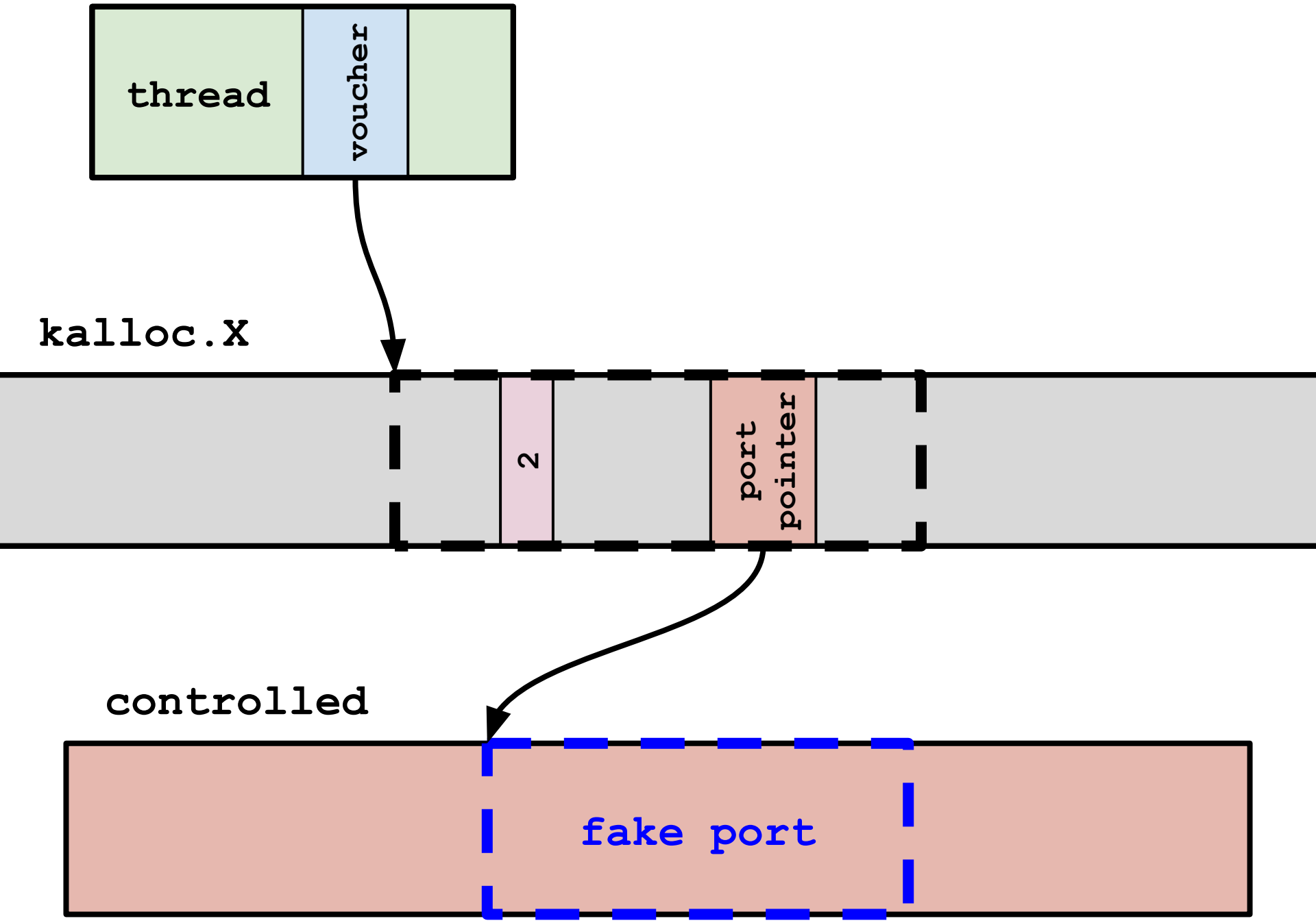

thread
voucher
ipc.vouchers
iv_port
iv_refs
iv_port
iv_refs
iv_port
target voucher
iv_refs = 1


thread
voucher
ipc.vouchers
iv_refs
iv_port
target voucher
iv_refs = 0


thread
voucher
kalloc.X
iv_refs
iv_port
target voucher
iv_refs = 0


thread
voucher
kalloc.X
2
port
pointer
controlled
fake port

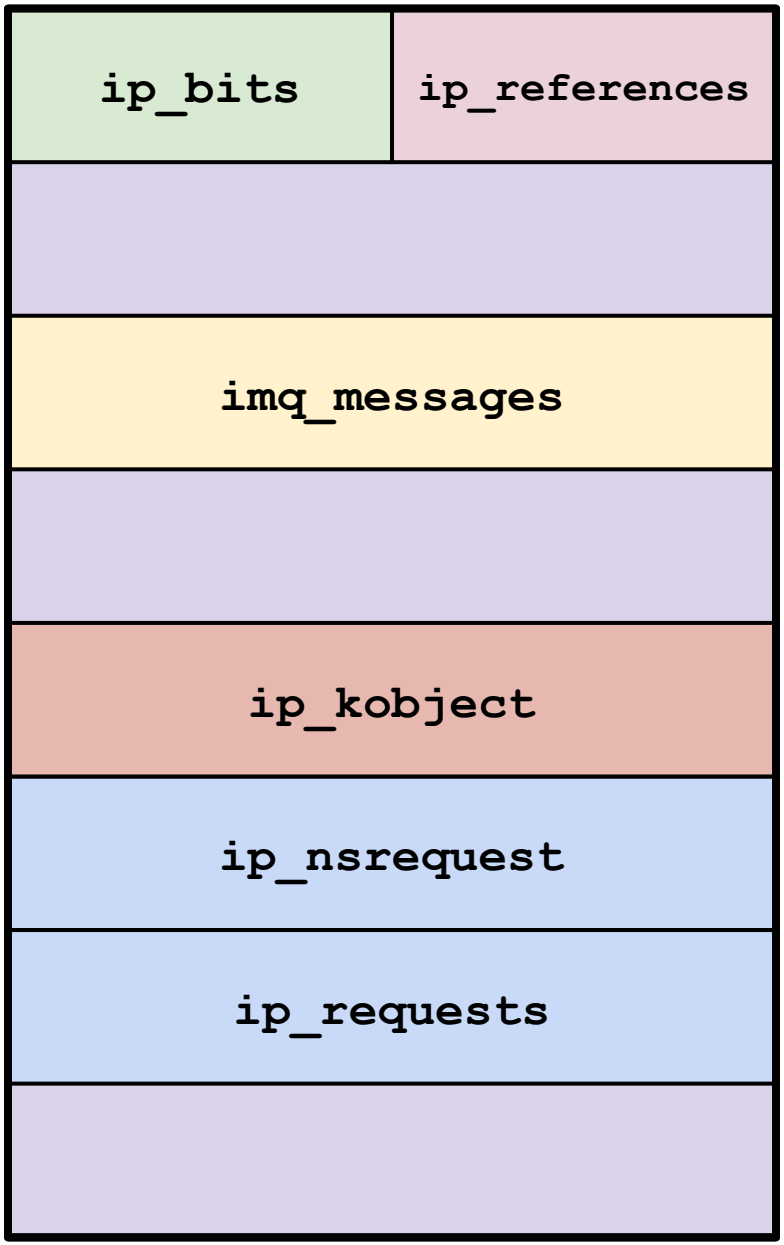

thread
voucher
thread_get_mach_voucher()
kalloc.X
2
port
pointer
controlled
fake port


thread
voucher
kalloc.X
2
port
pointer
controlled
fake port

We do not know any kernel addresses

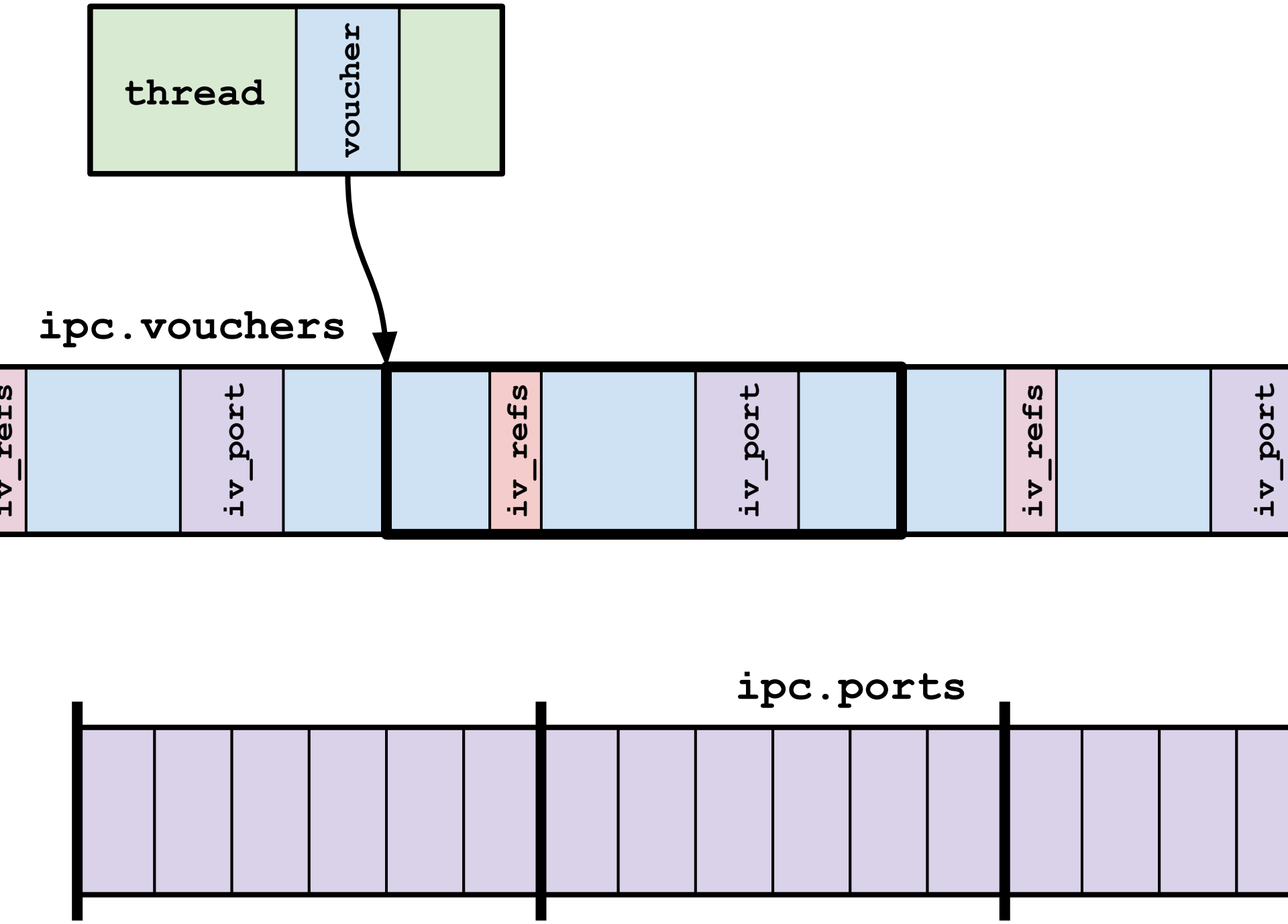
A first exploit

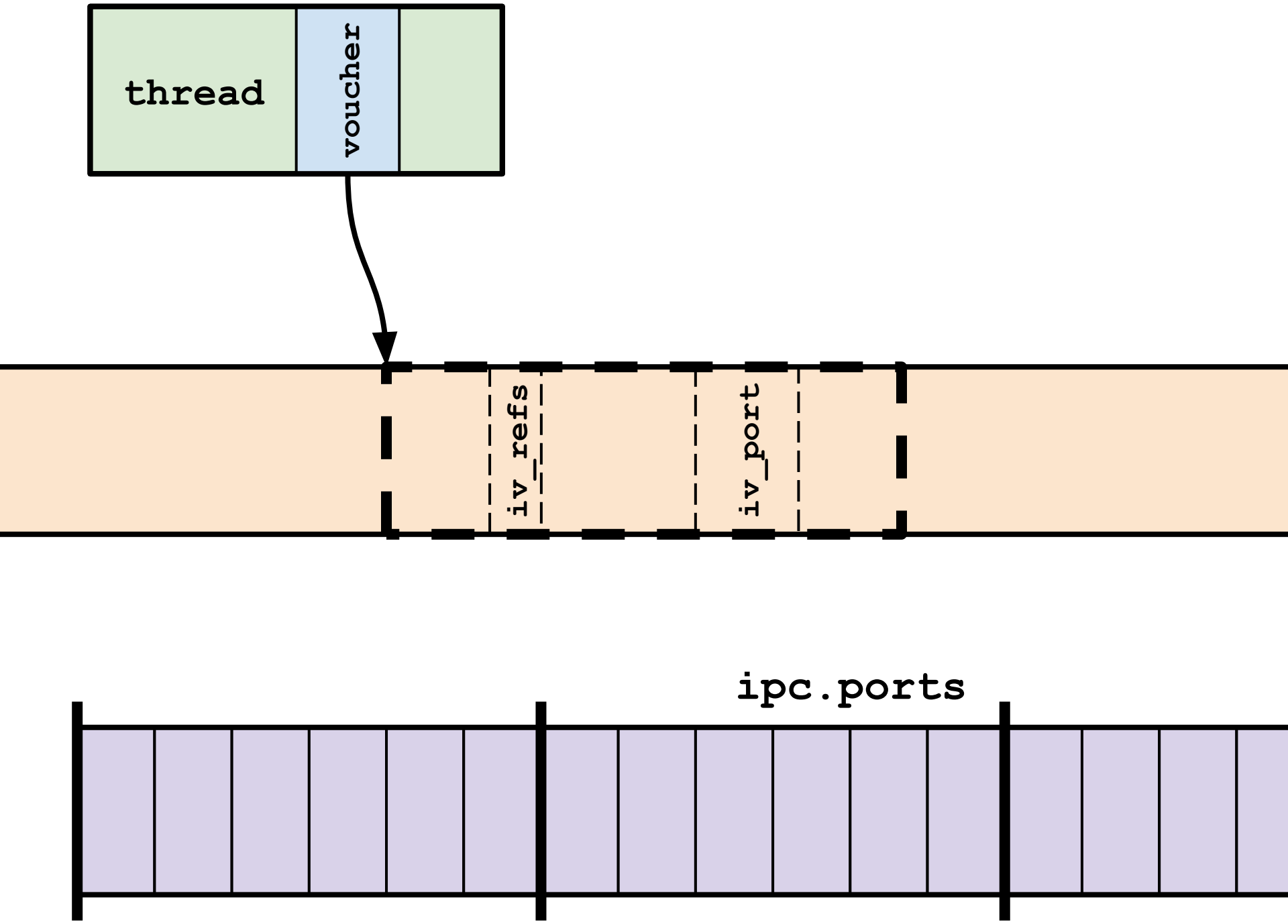
Convert the UAF on vouchers into a UAF on ports

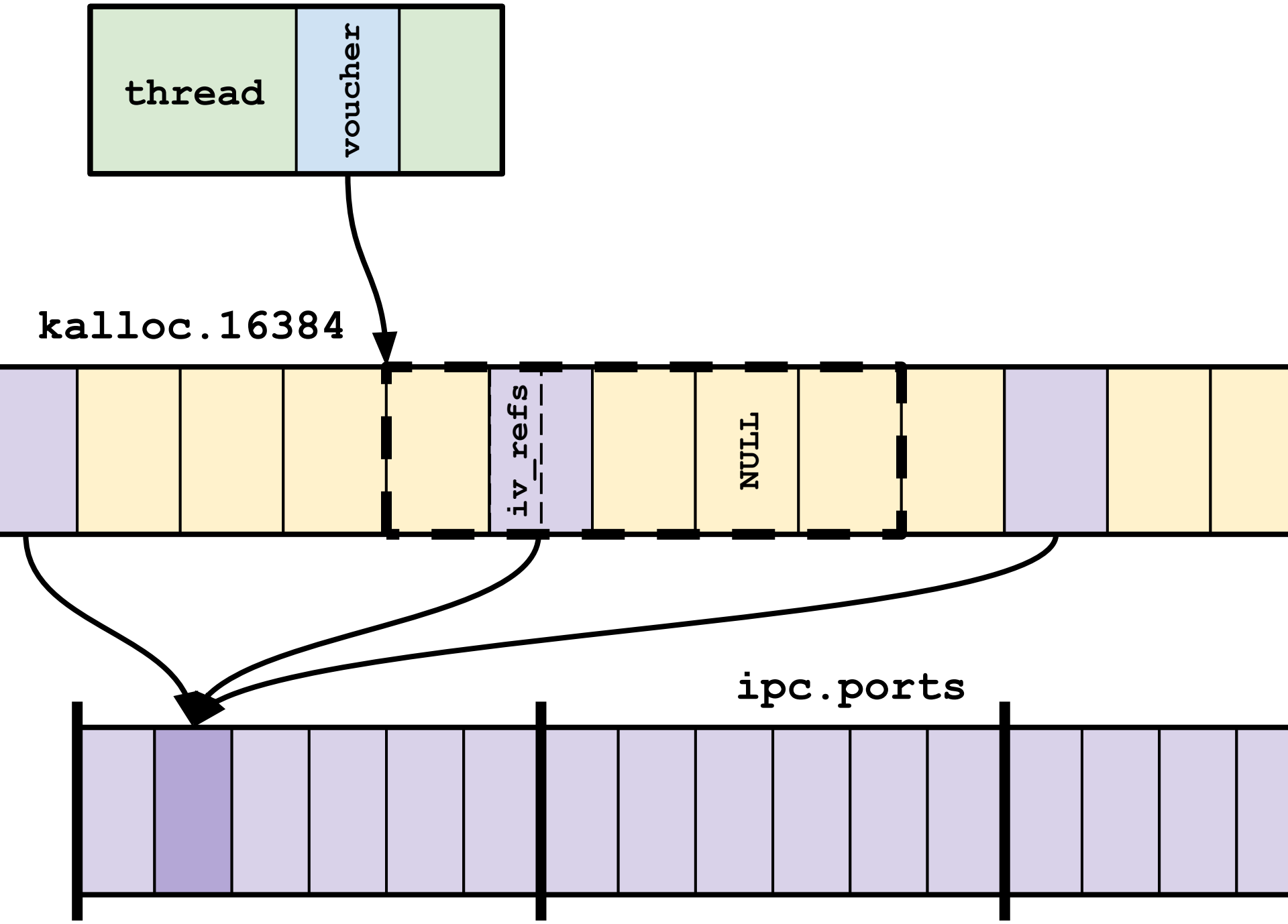


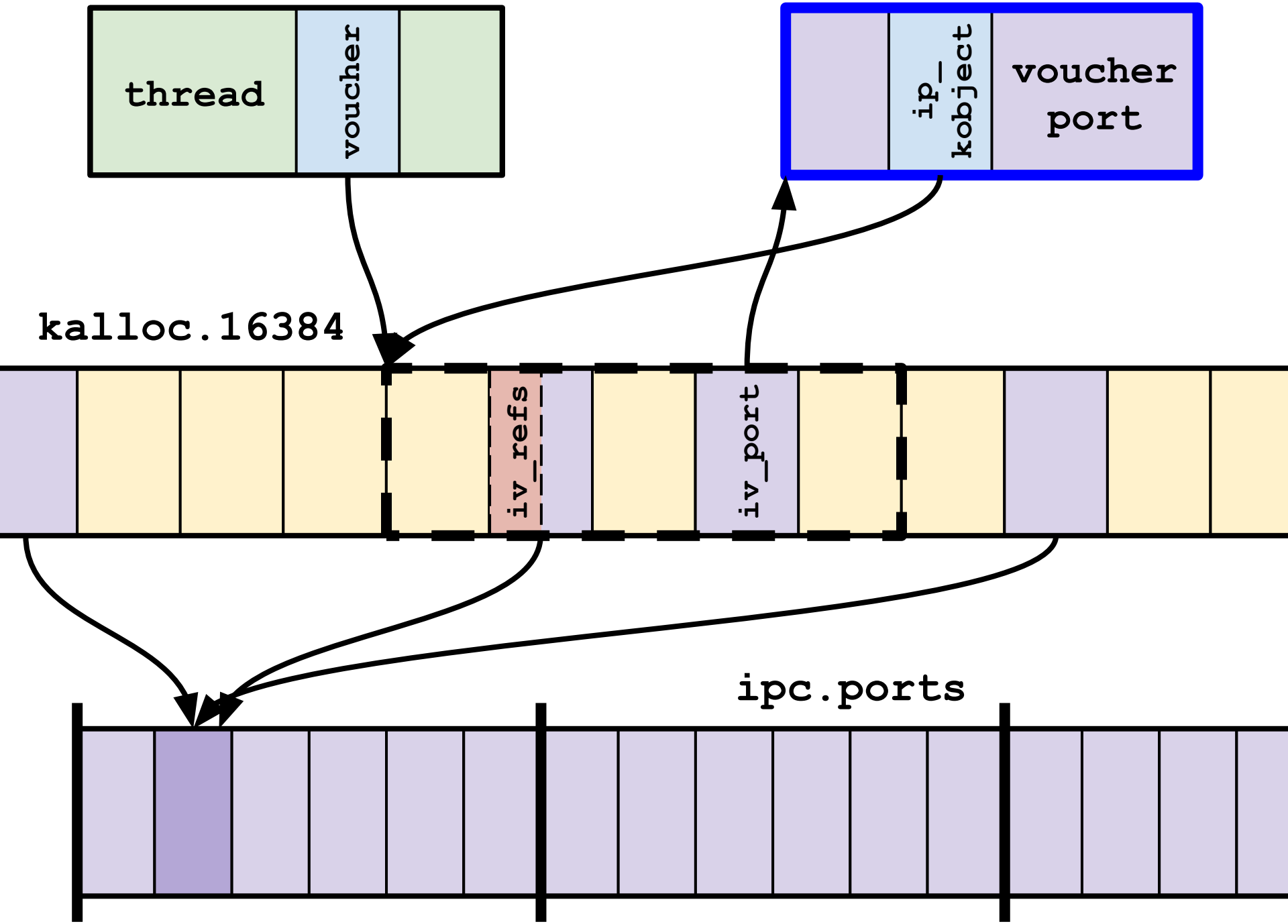
Mach port
(struct ipc_port)

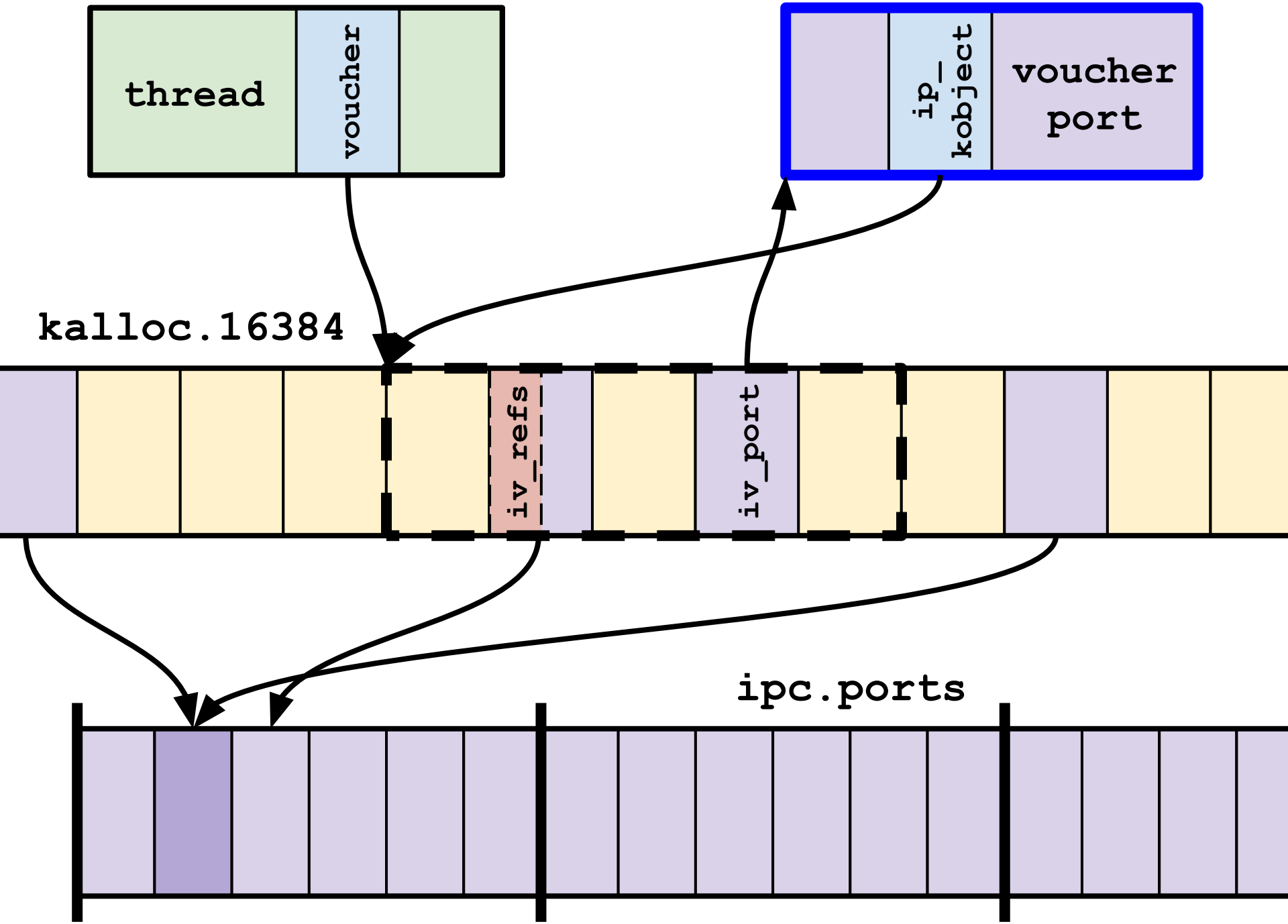
Drop a reference on a port while we still hold the send right

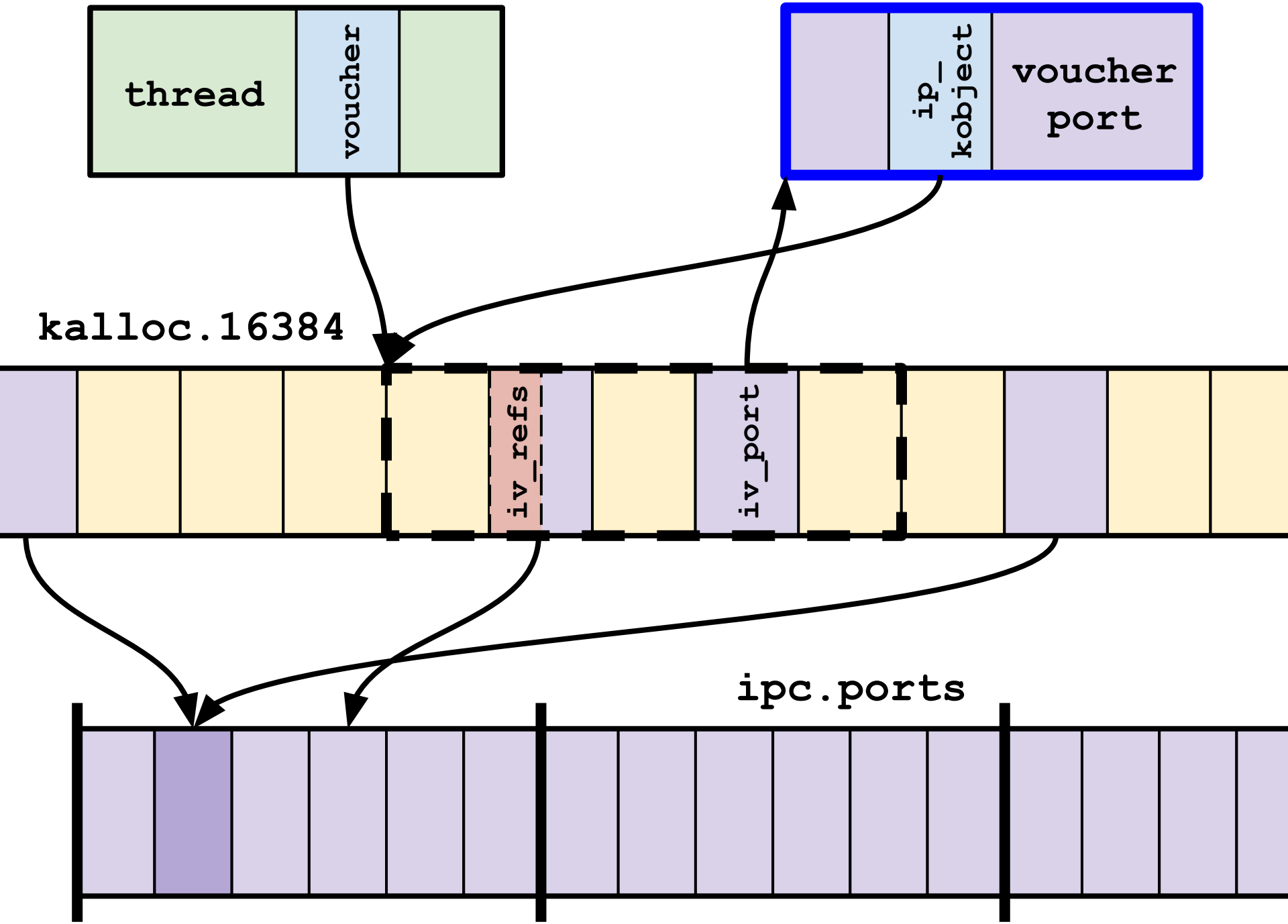

thread
voucher
ipc.vouchers
iv_port
iv_refs
iv_port
iv_refs
iv_port
ipc.ports

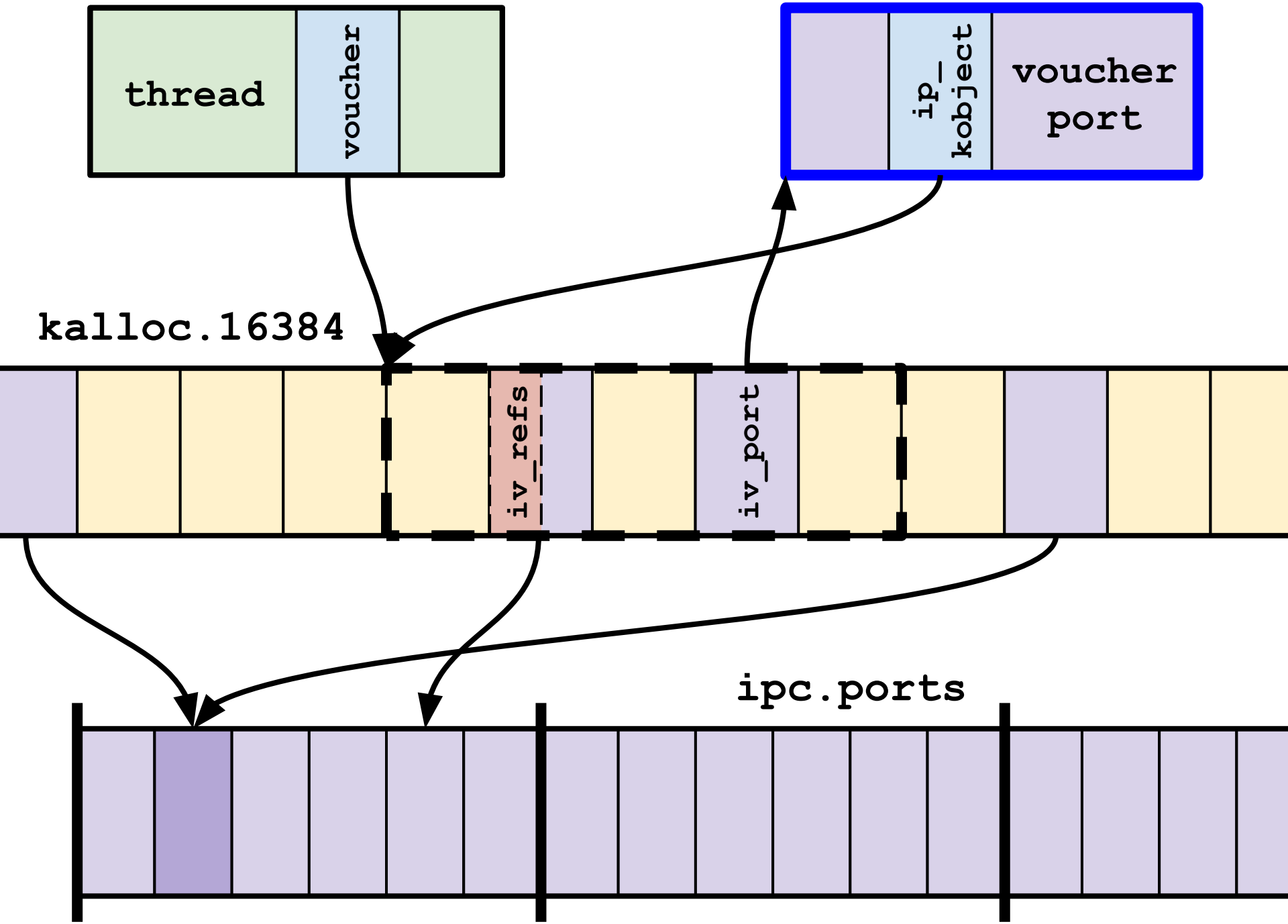

thread
voucher
iv_refs
iv_port
ipc.ports

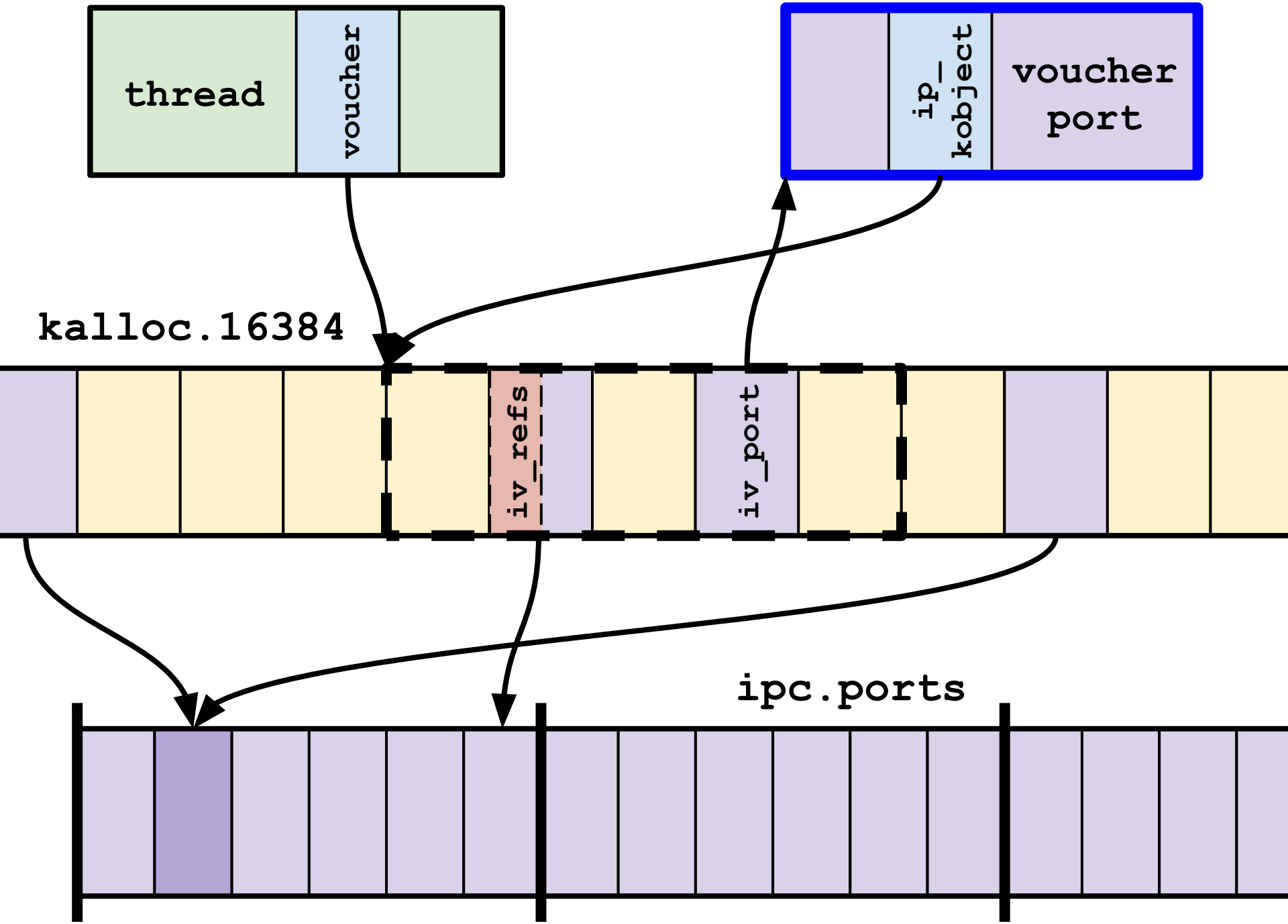

thread
voucher
kalloc.16384
iv_refs
NULL
ipc.ports

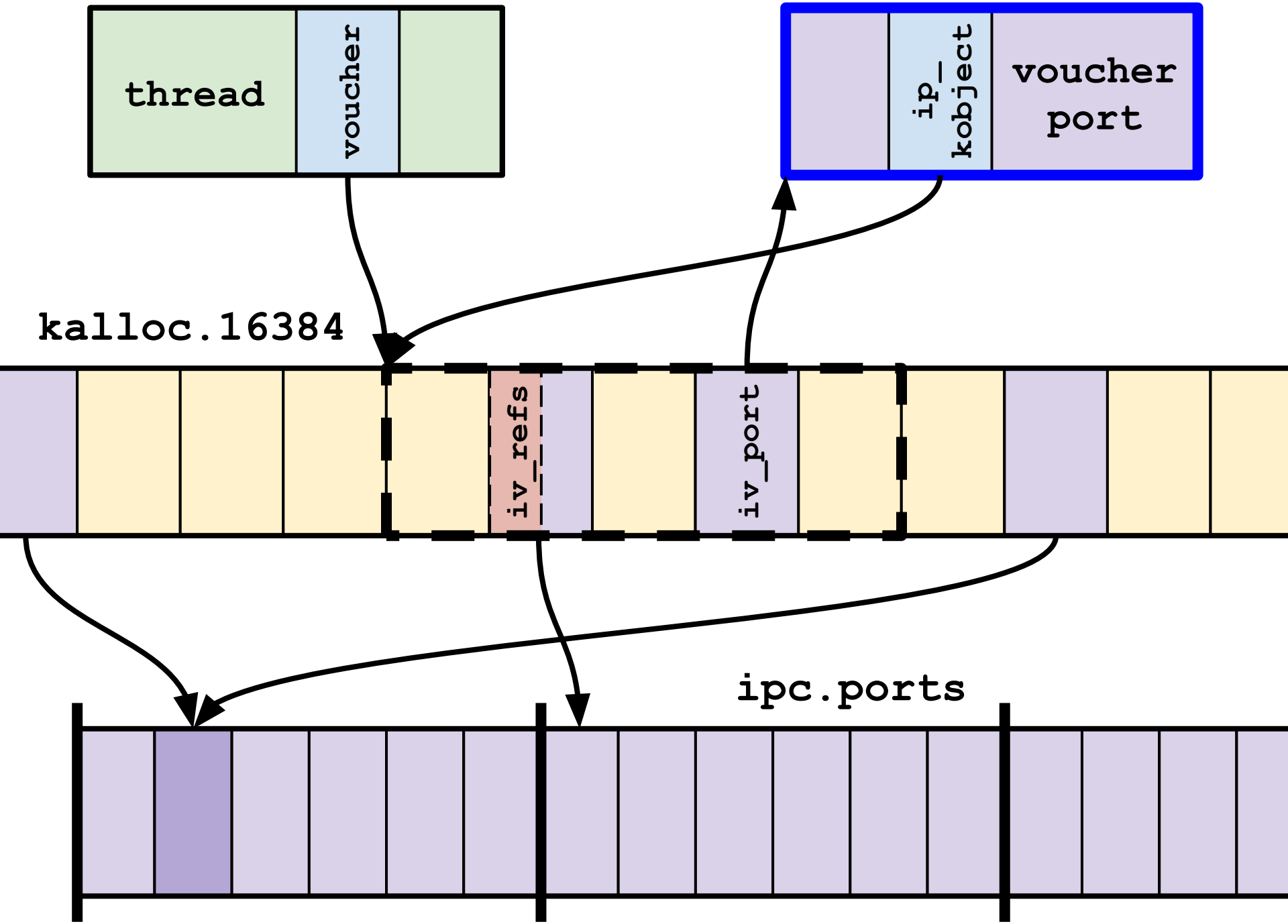

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

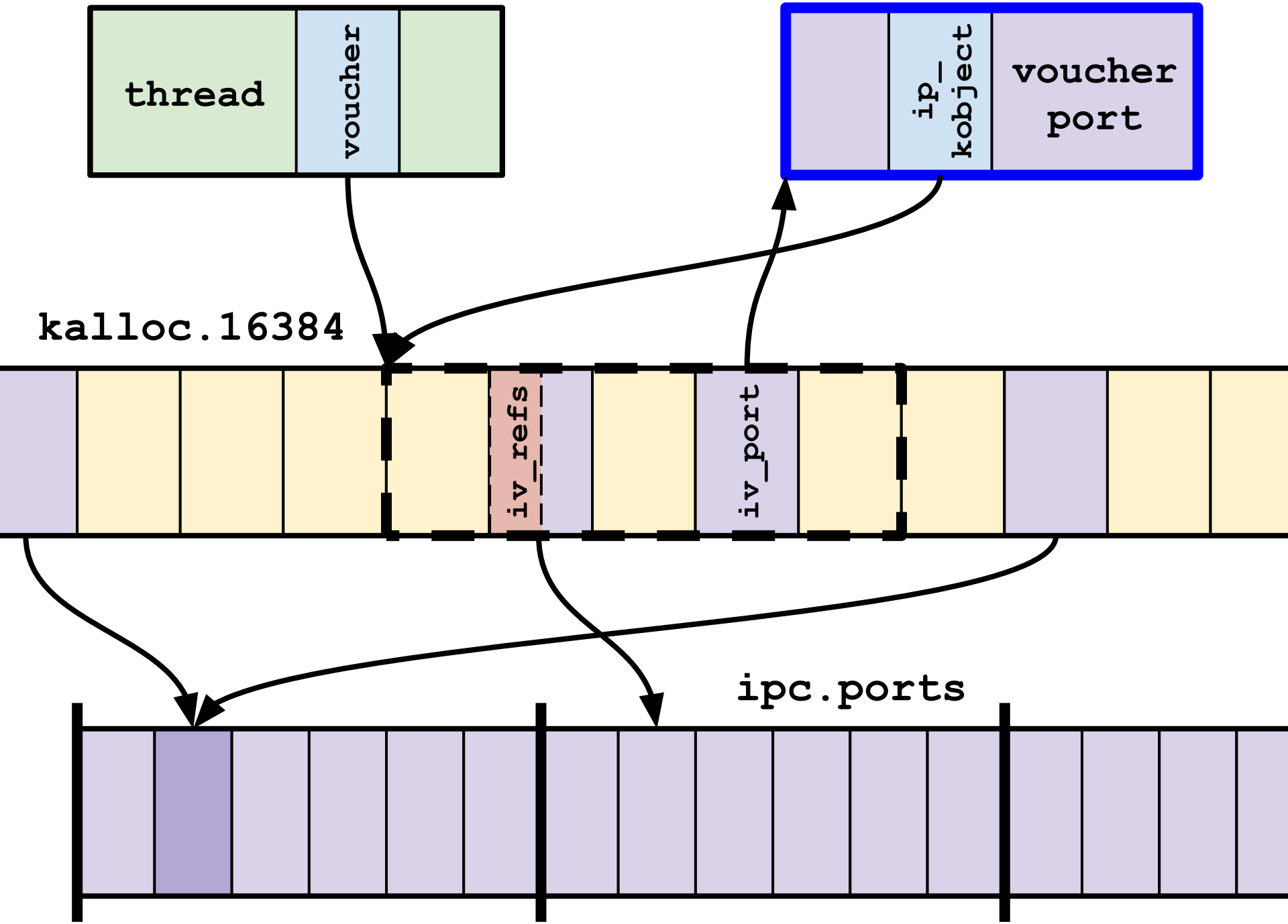

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

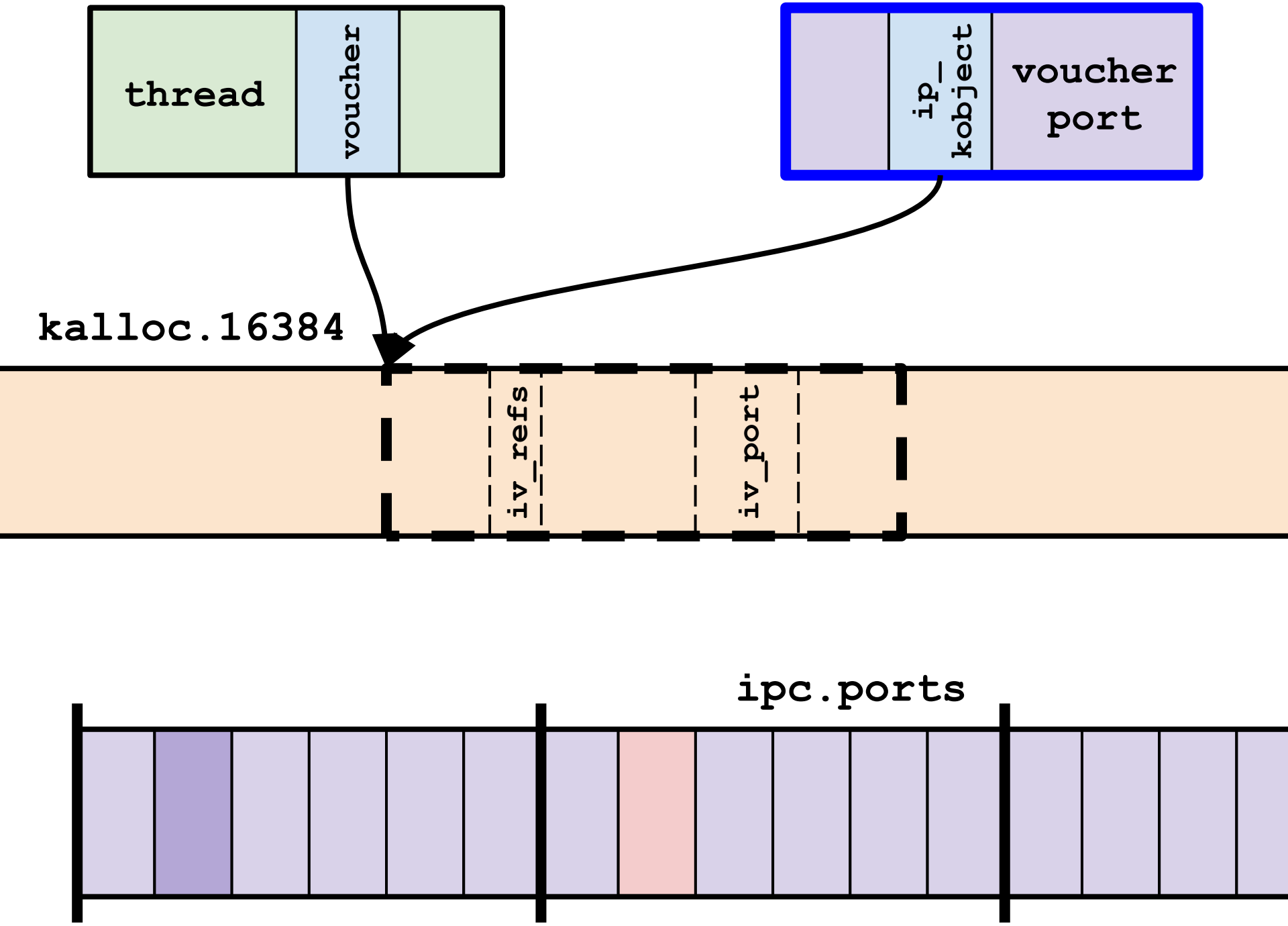

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports


thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

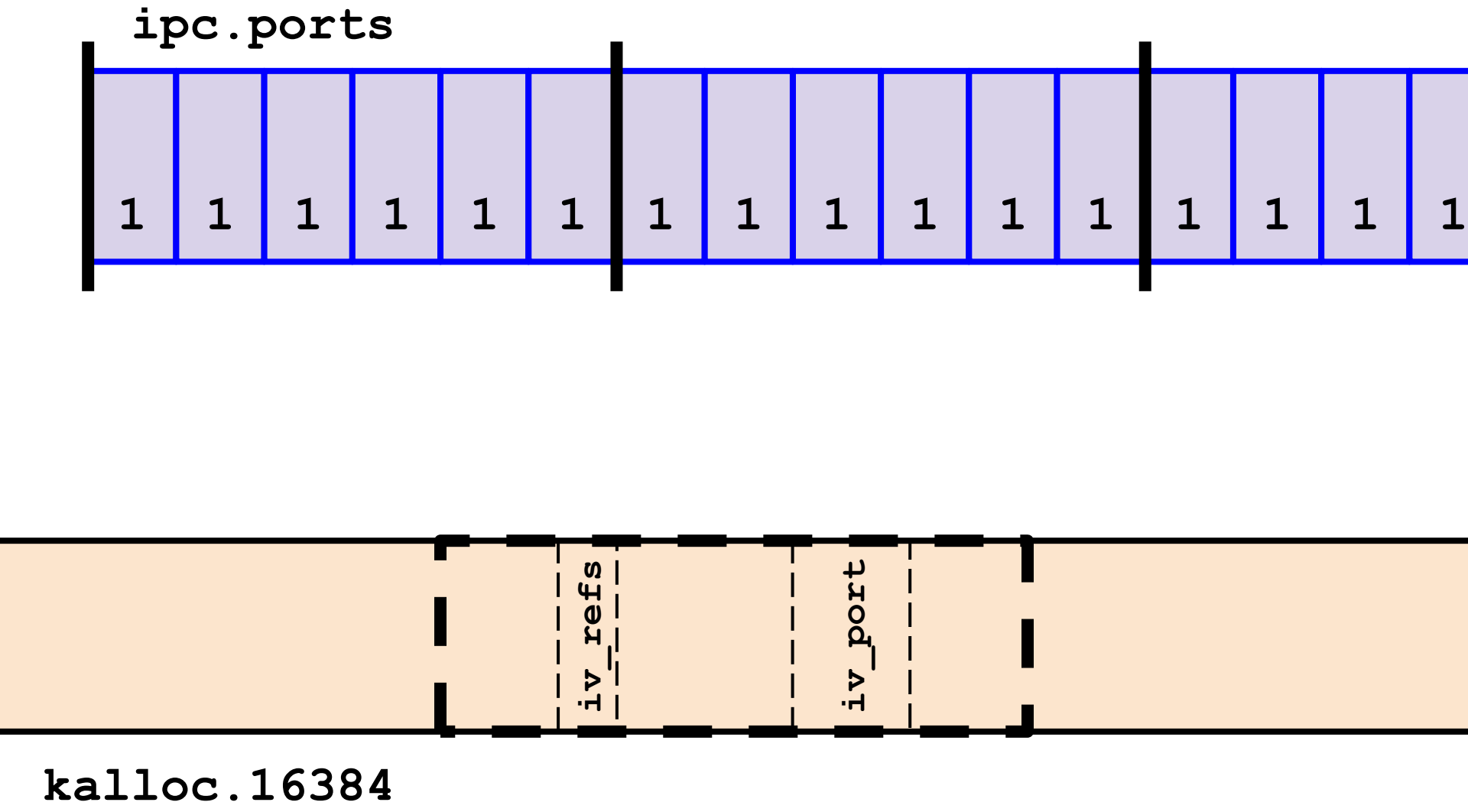

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

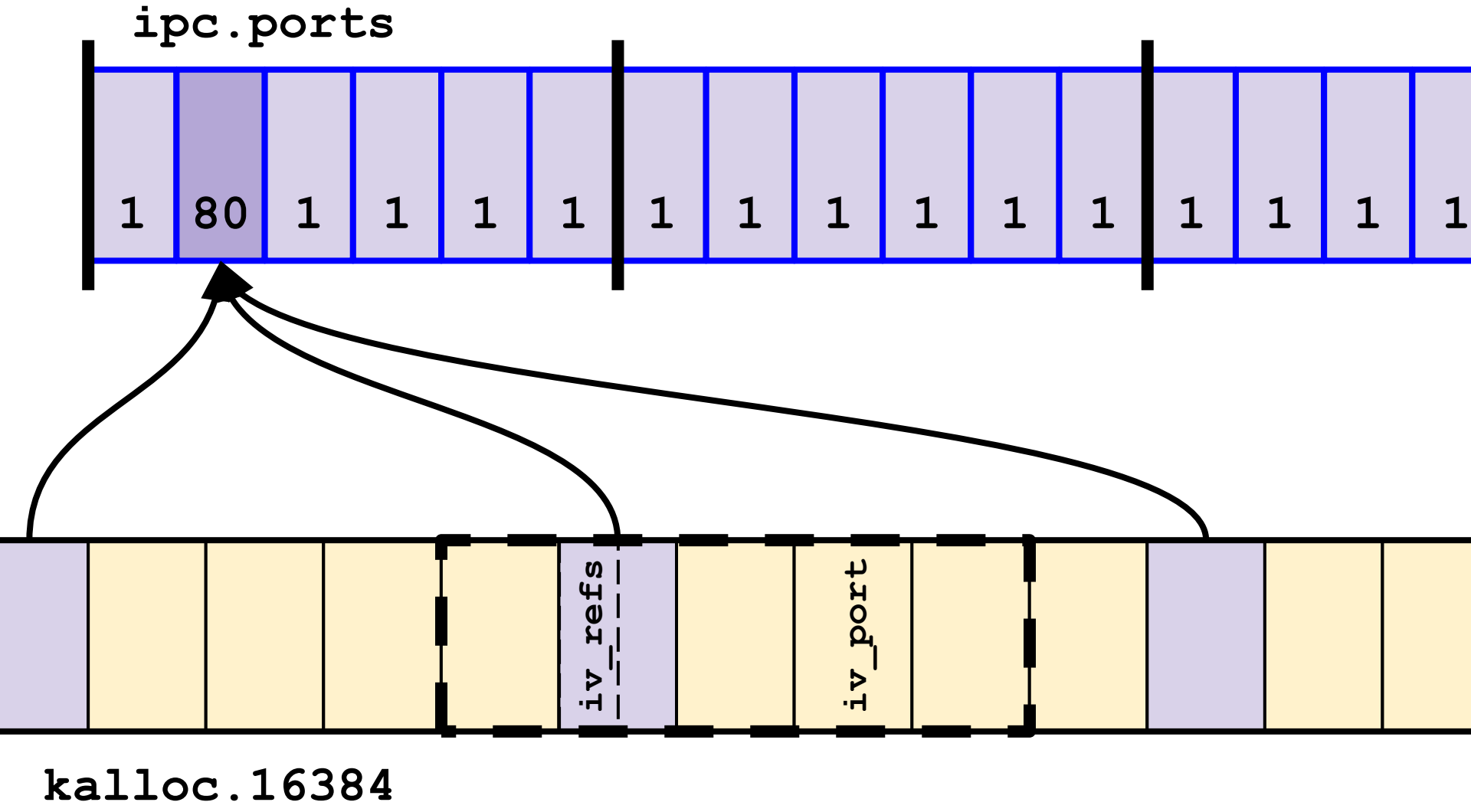

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

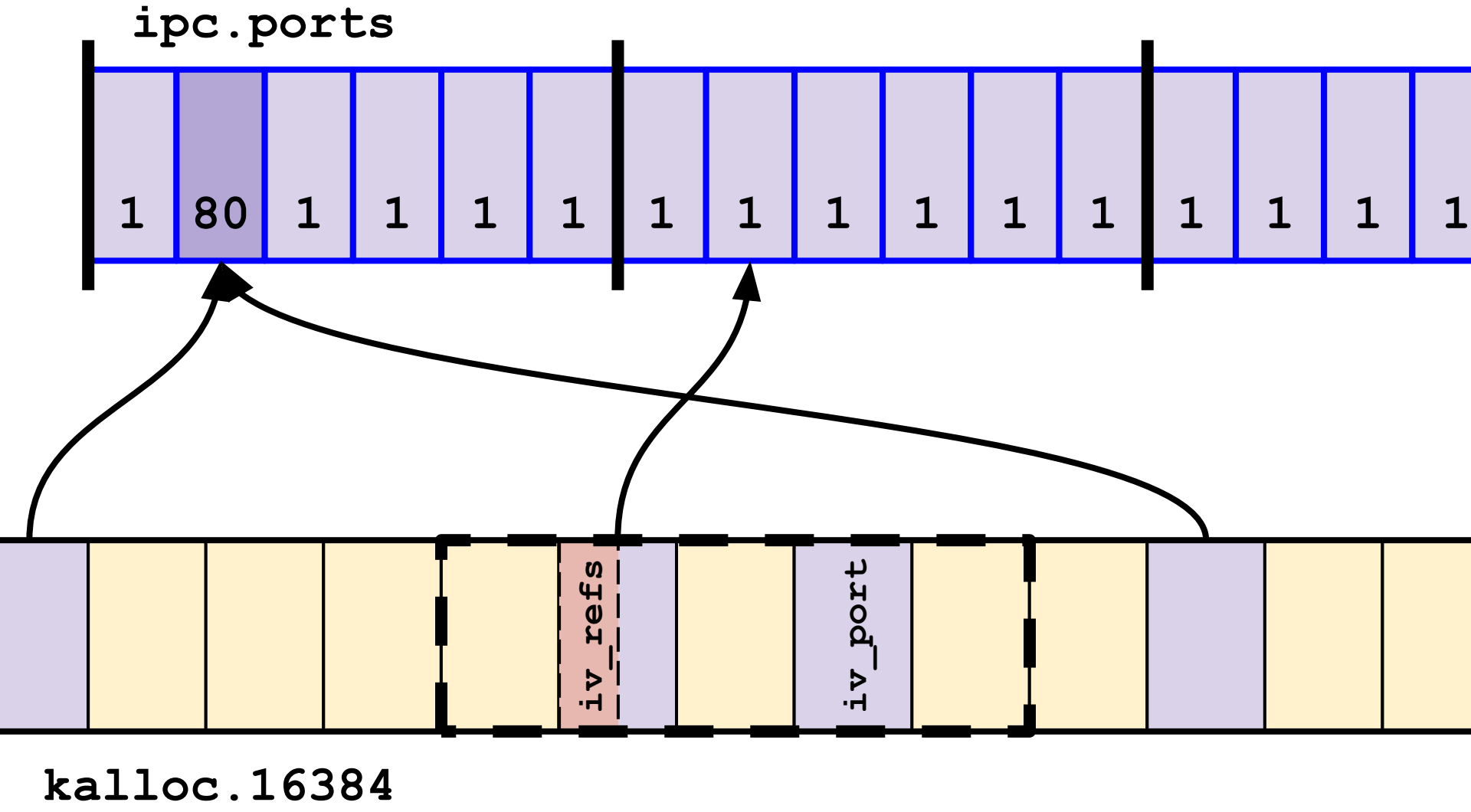

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

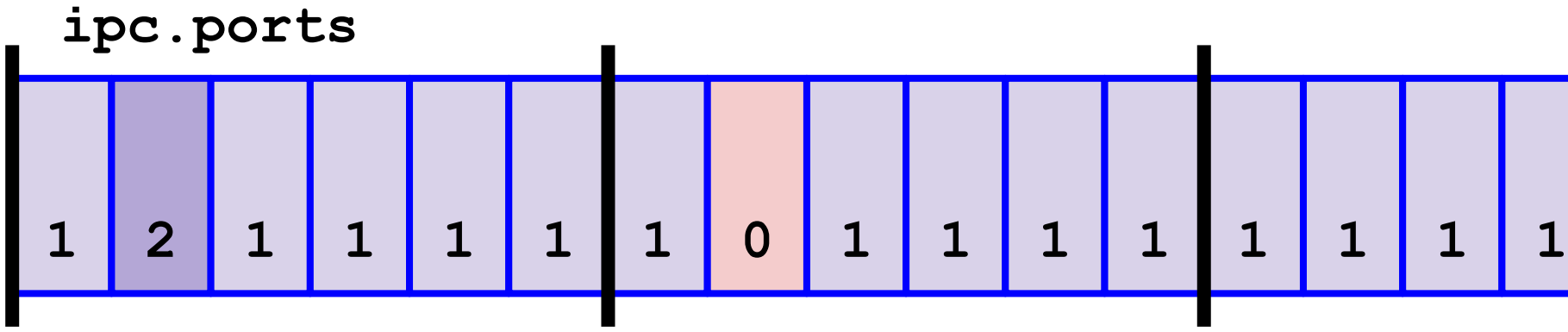

thread
voucher
ip_
kobject
voucher
port
kalloc.16384
iv_refs
iv_port
ipc.ports

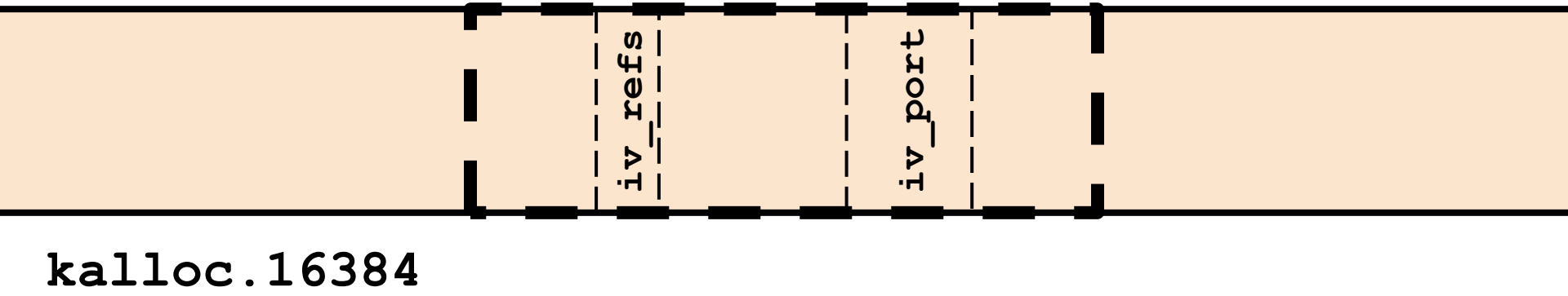
Transferring a send right from one port to another

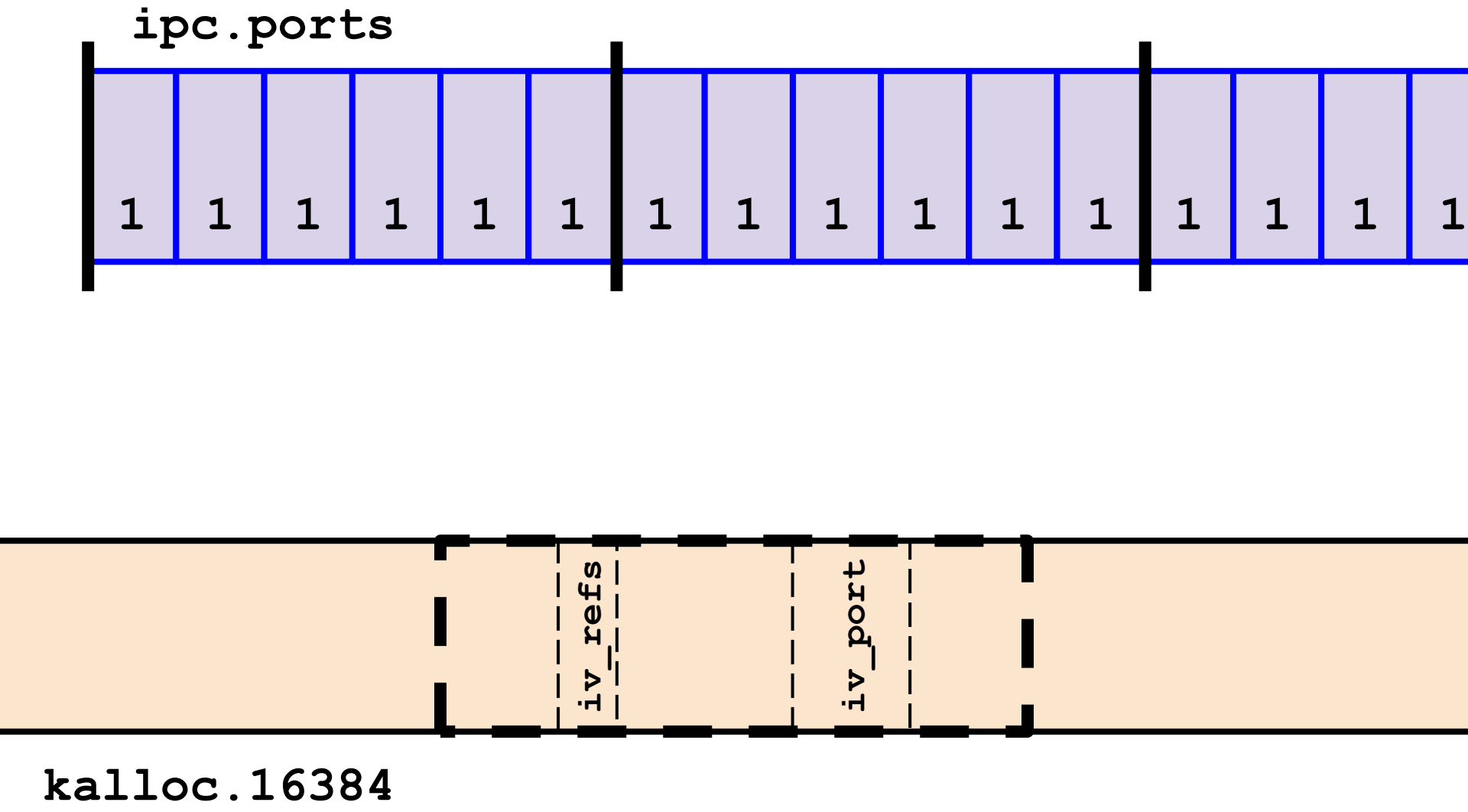

ipc.ports
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
iv_refs
iv_port
kalloc.16384

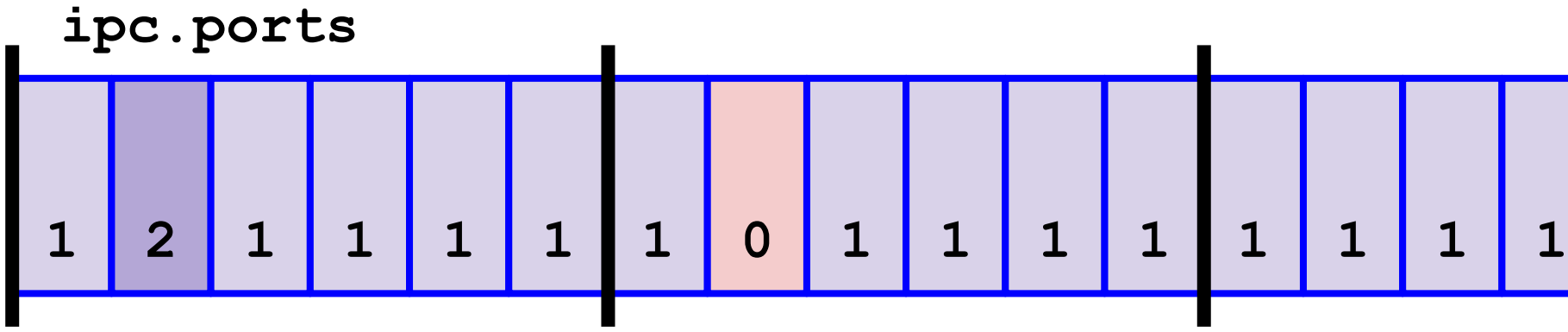

ipc.ports
1
80
1
1
1
1
1
1
1
1
1
1
1
1
1
1
iv_refs
iv_port
kalloc.16384

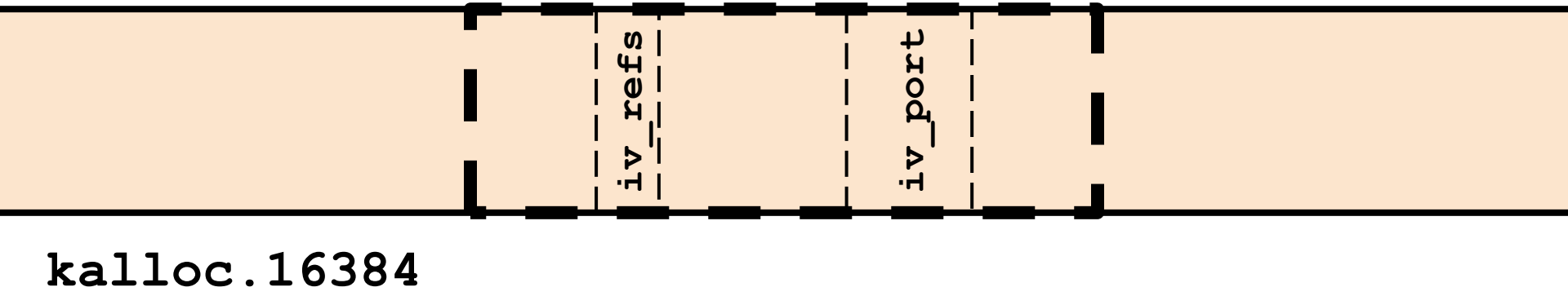

ipc.ports
1
80
1
1
1
1
1
1
1
1
1
1
1
1
1
1
iv_refs
iv_port
kalloc.16384


ipc.ports
1
2
1
1
1
1
1
0
1
1
1
1
1
1
1
1

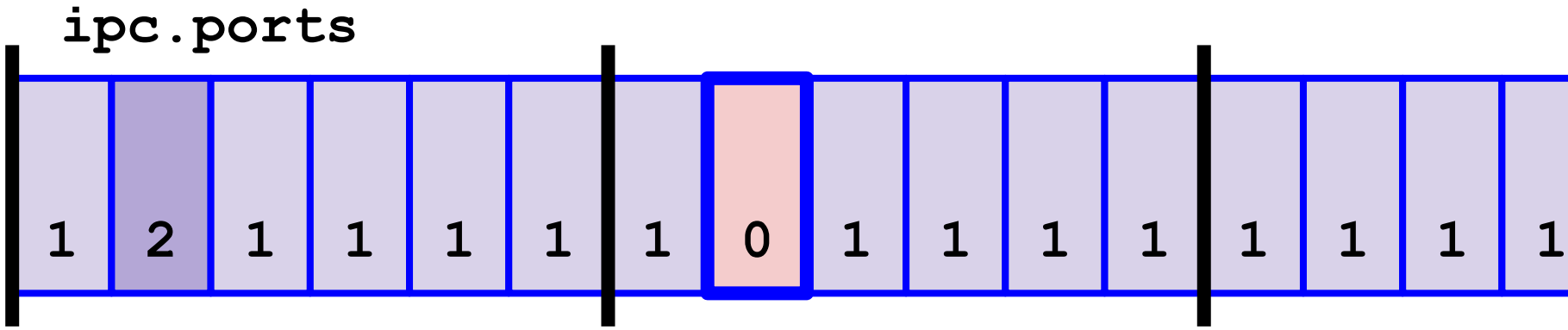

iv_refs
iv_port
kalloc.16384

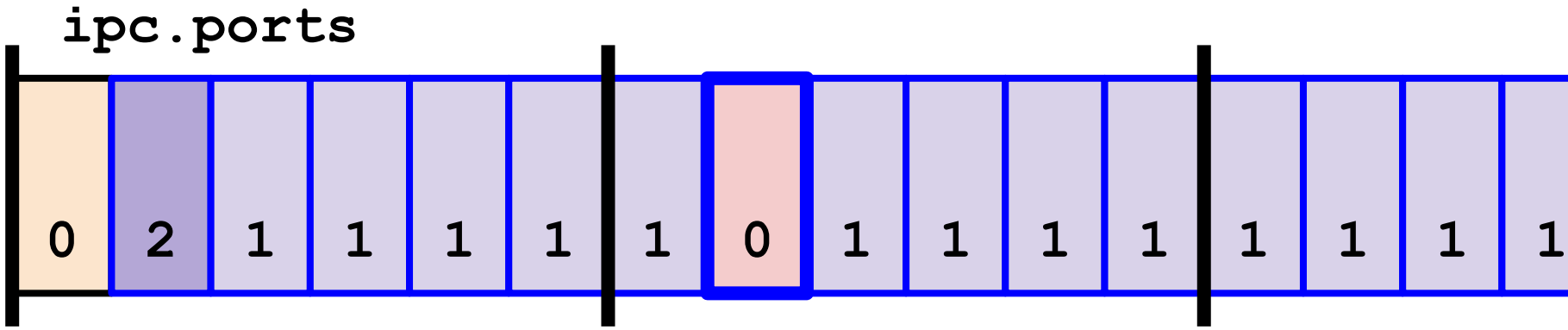

ipc.ports
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
iv_refs
iv_port
kalloc.16384

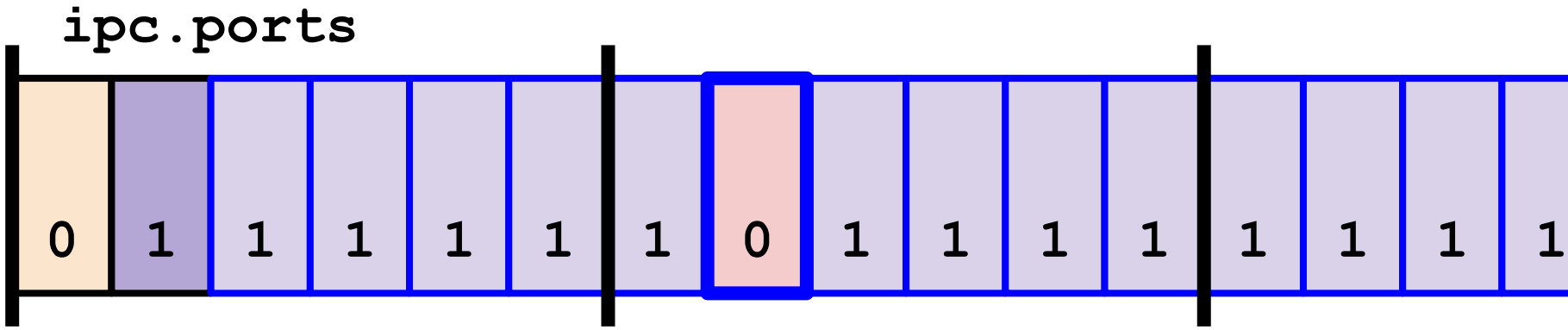

ipc.ports
1
2
1
1
1
1
1
0
1
1
1
1
1
1
1
1

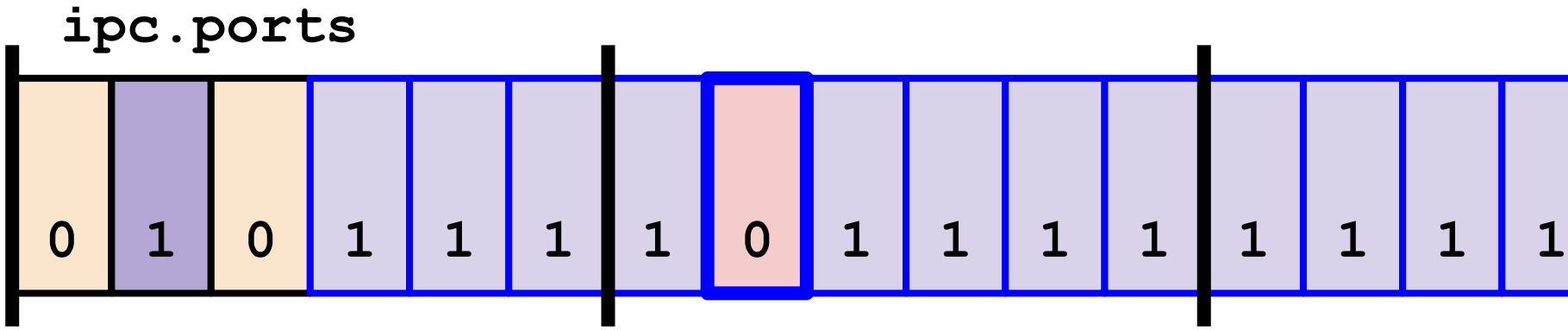

iv_refs
iv_port
kalloc.16384

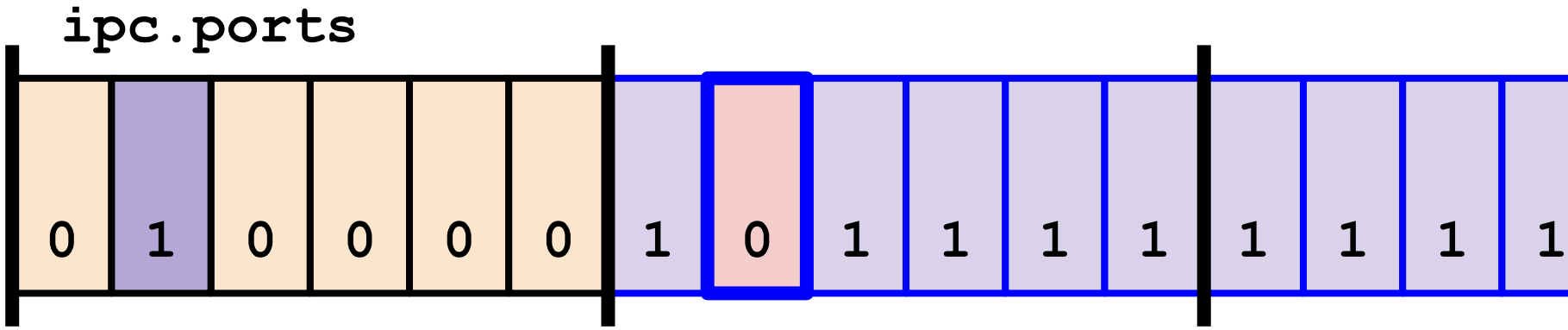
Exploitation will require a 2nd zone garbage collection

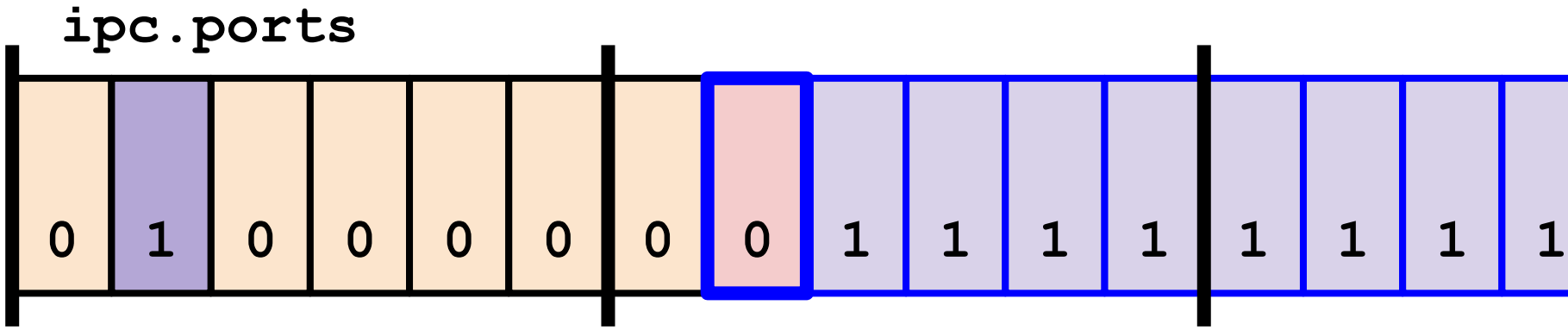

ipc.ports
1
2
1
1
1
1
1
0
1
1
1
1
1
1
1
1

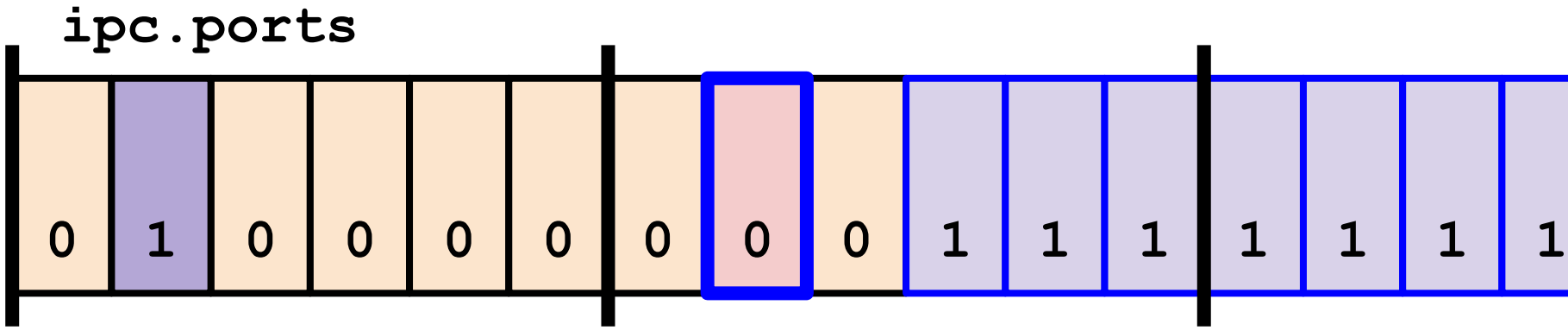

ipc.ports
0
2
1
1
1
1
1
0
1
1
1
1
1
1
1
1

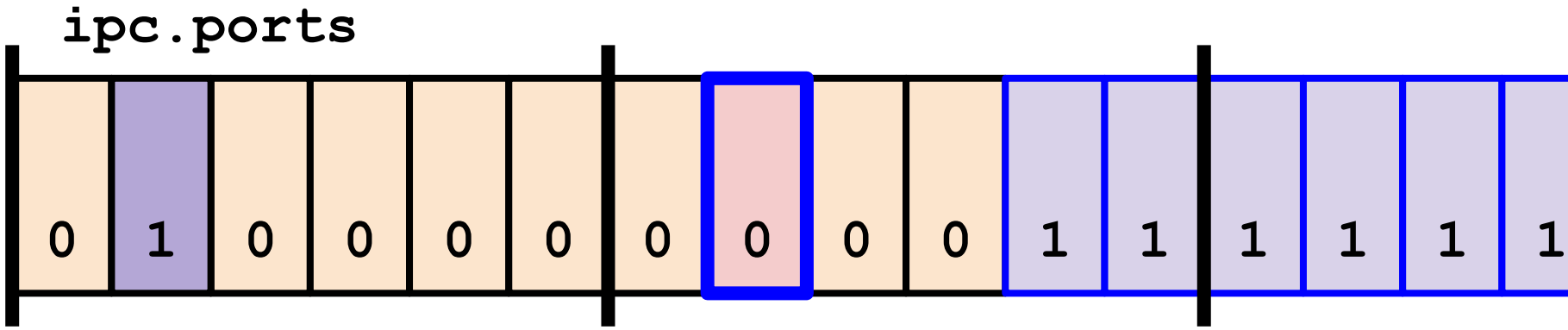

ipc.ports
0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1

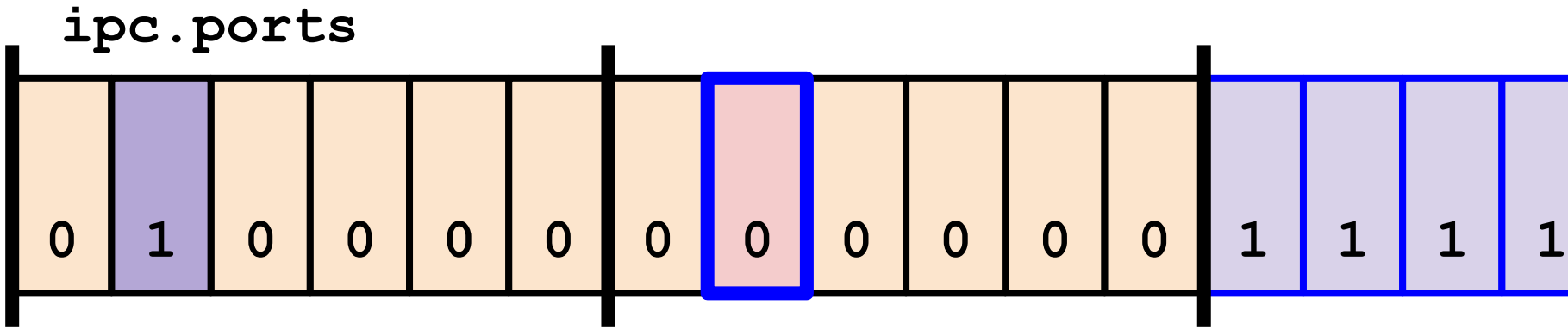

ipc.ports
0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1

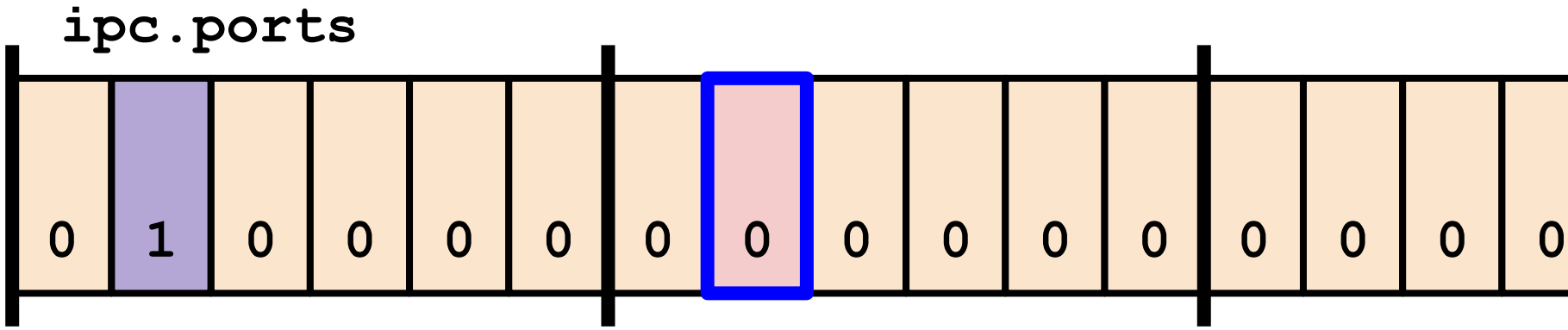

ipc.ports
0 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1

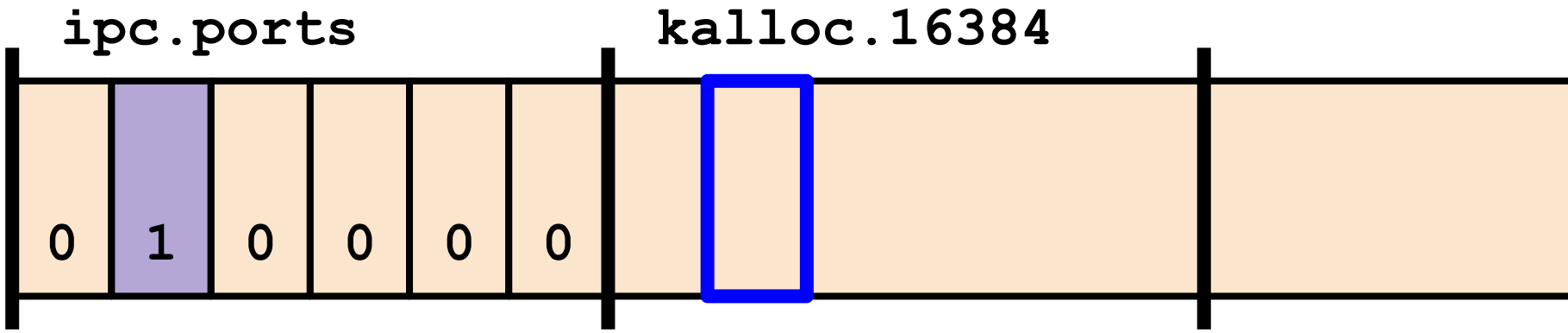

ipc.ports
0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1

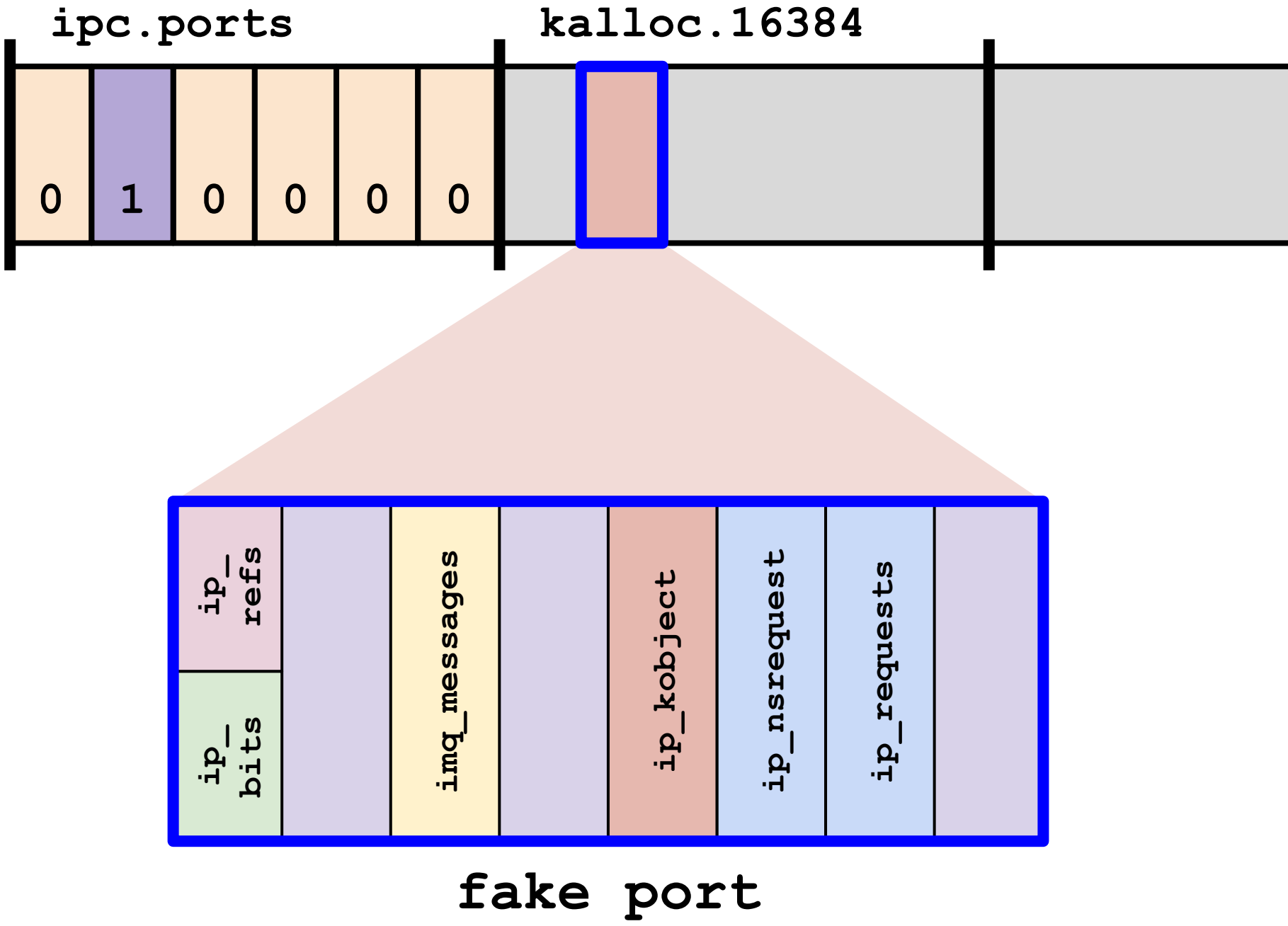

ipc.ports
0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1


ipc.ports
0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1


ipc.ports
0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1


ipc.ports
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0


ipc.ports
kalloc.16384
0
1
0
0
0
0

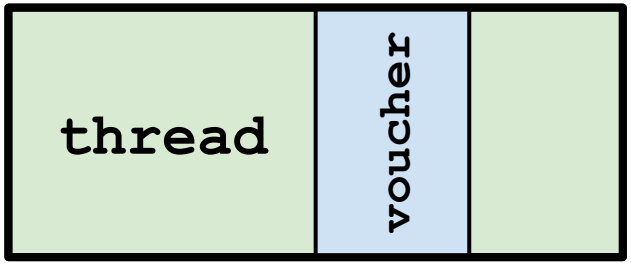

ipc.ports
kalloc.16384
0
1
0
0
0
0
ip_ refs
ip_ bits
imq_messages
ip_kobject
ip_nsrequest
ip_requests
fake port

Can we make an exploit with only *one* zone garbage collection?

The pipes technique

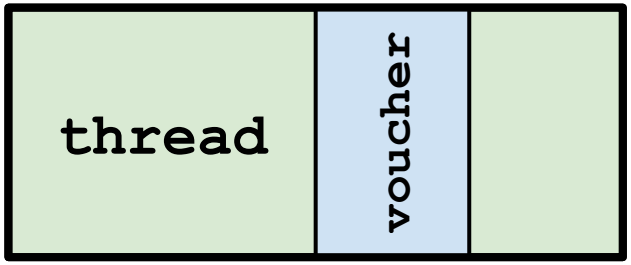
Exploiting the reference count bug is 100% reliable

Use the arbitrary 32-bit increment to get a fake port directly



ipc.vouchers

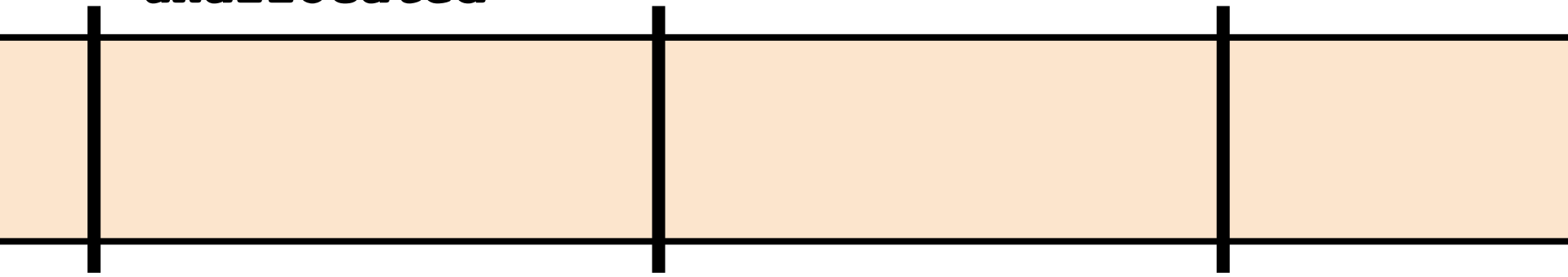
unallocated

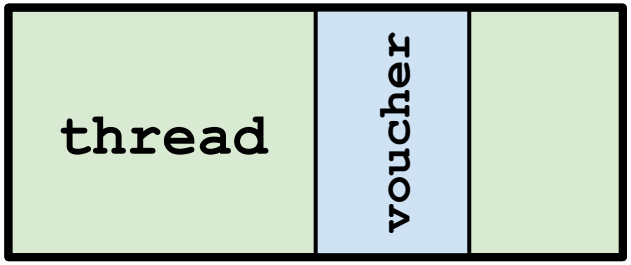


ipc.vouchers

iv_refs = 1

unallocated



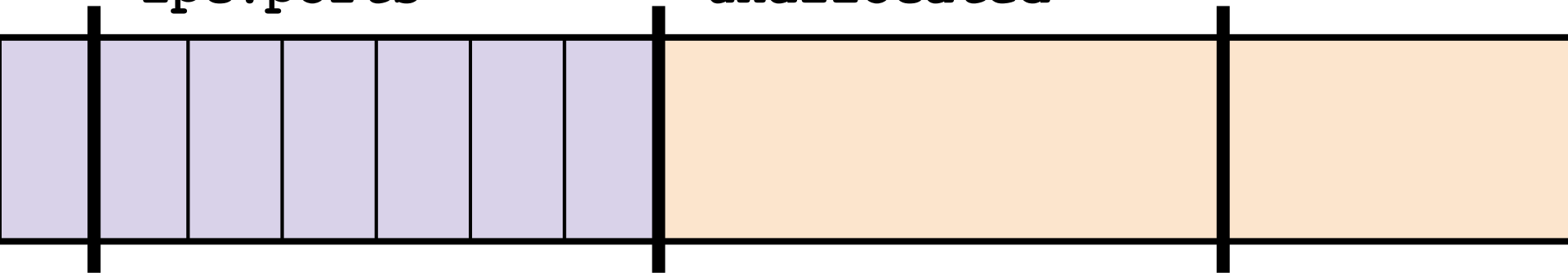


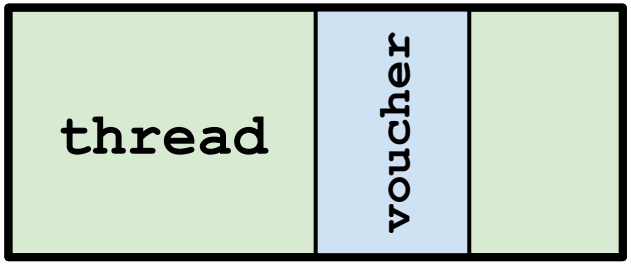
ipc.vouchers

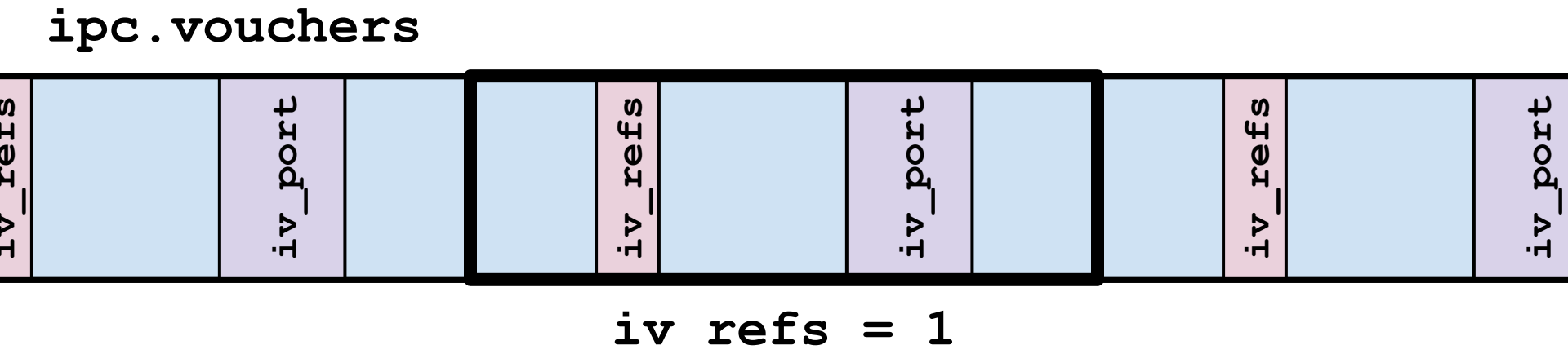
iv_refs = 1

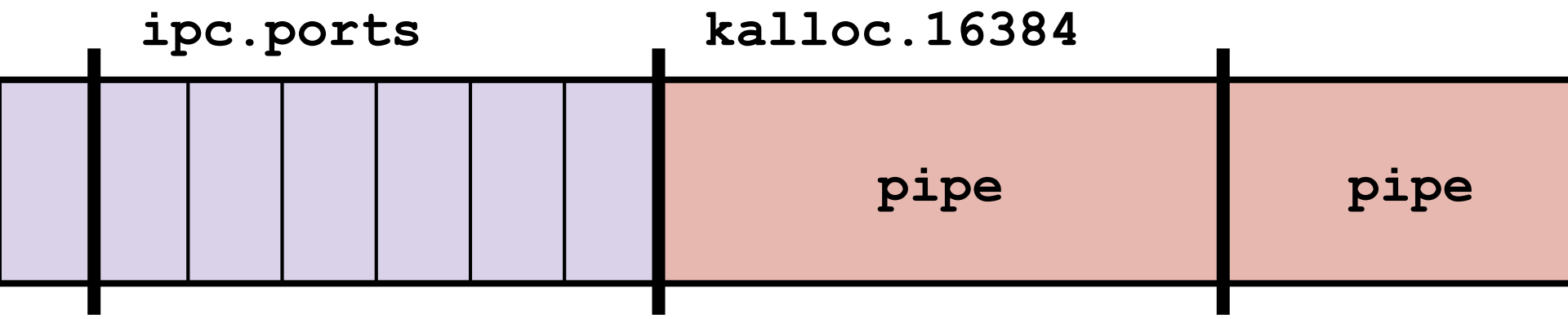
ipc.ports

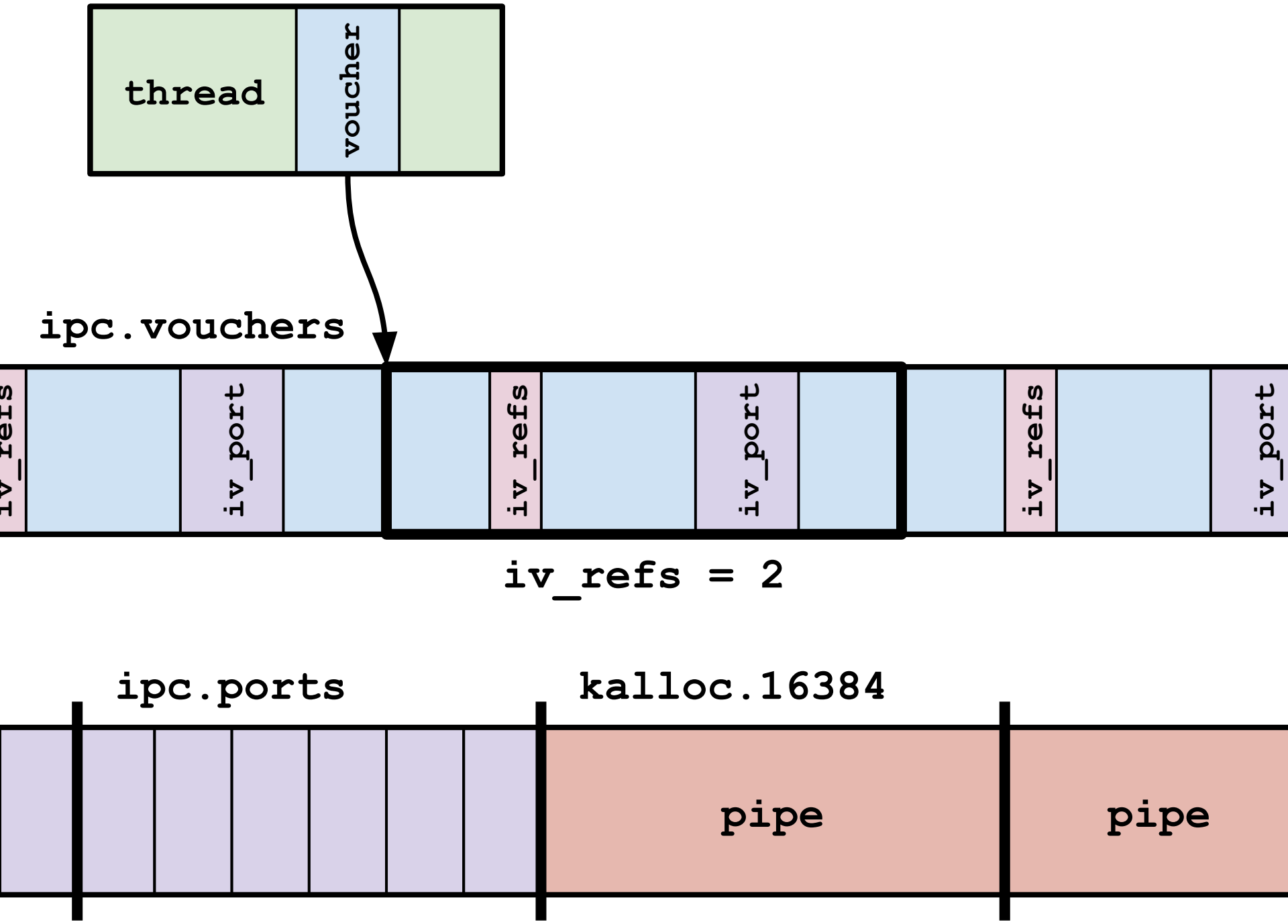
unallocated

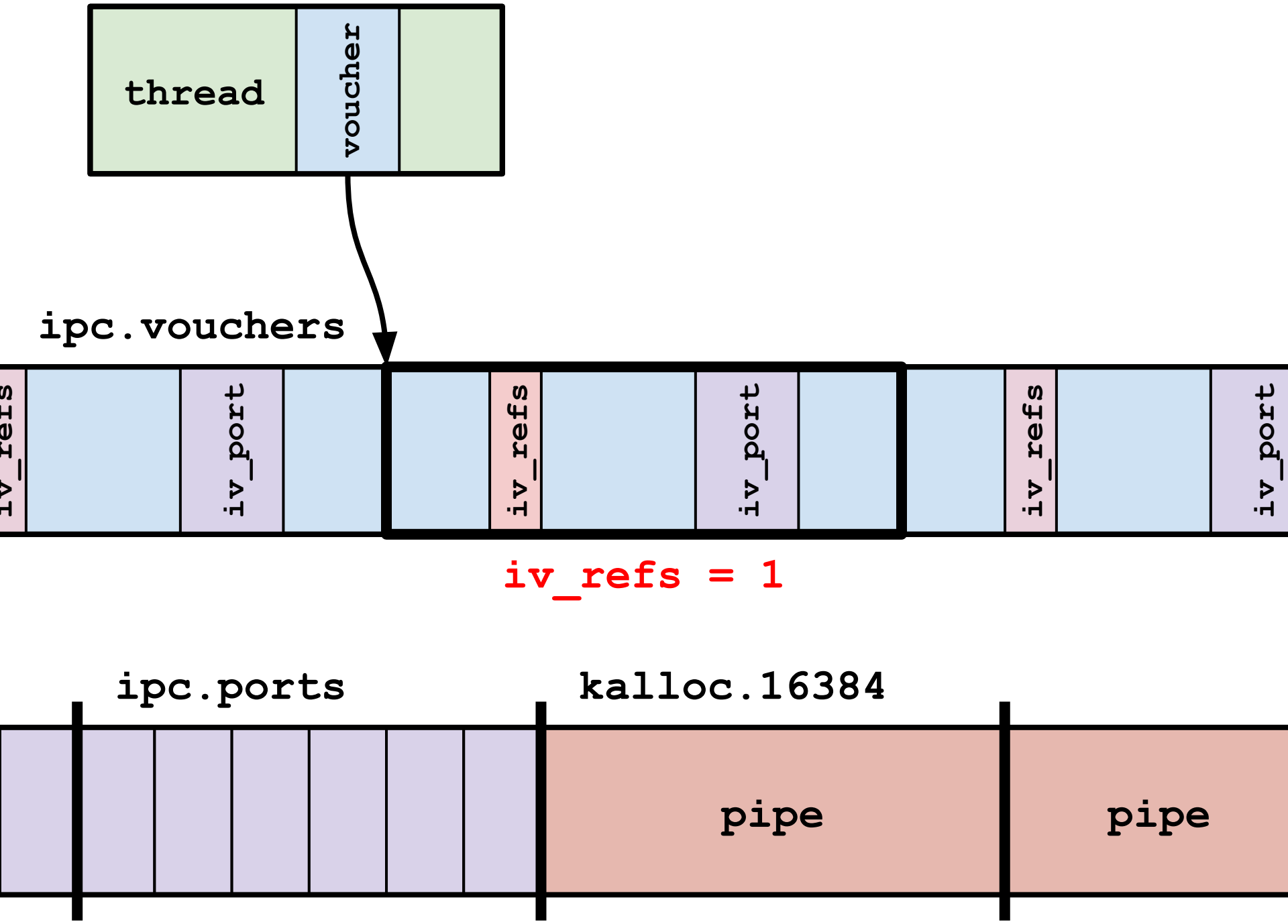


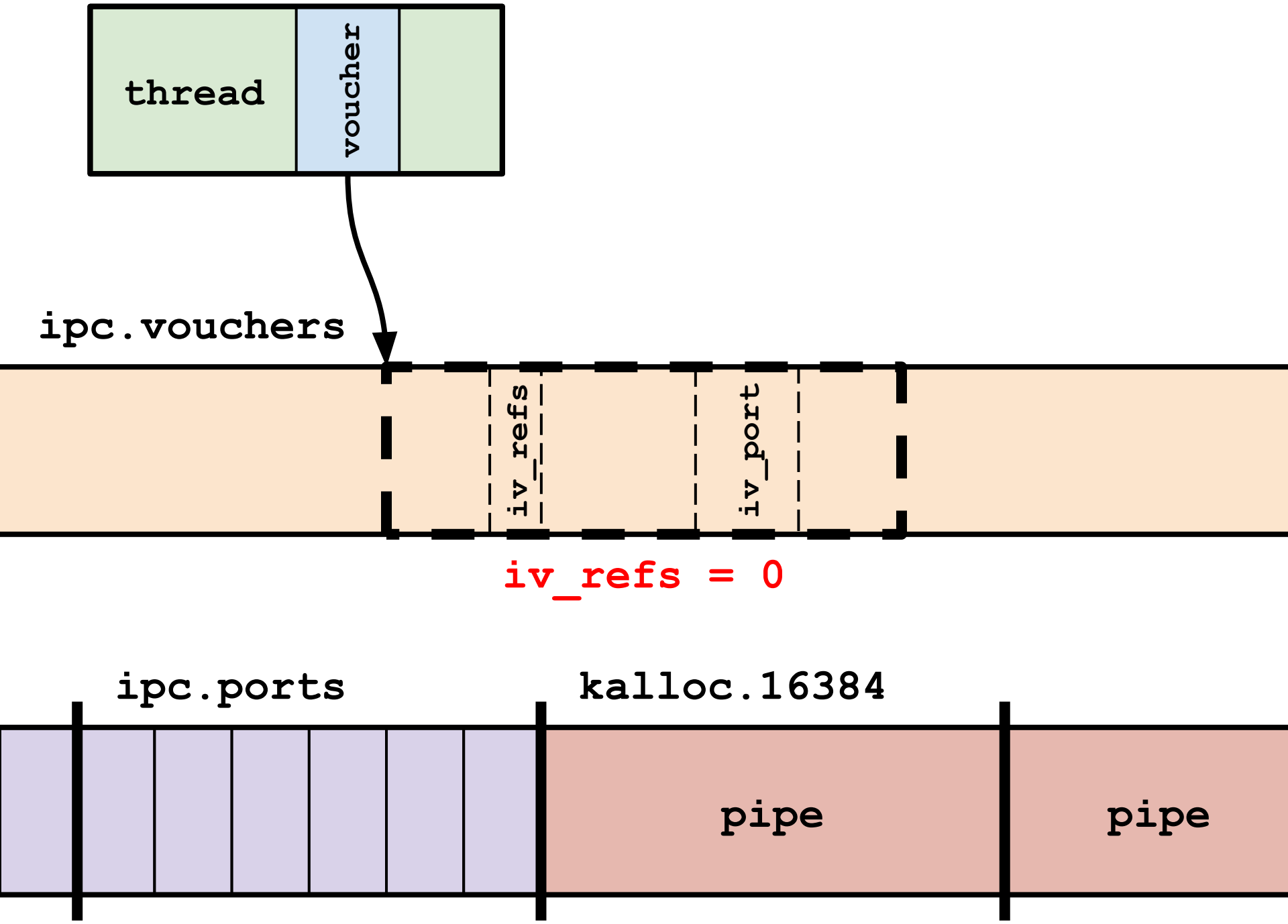

thread
voucher

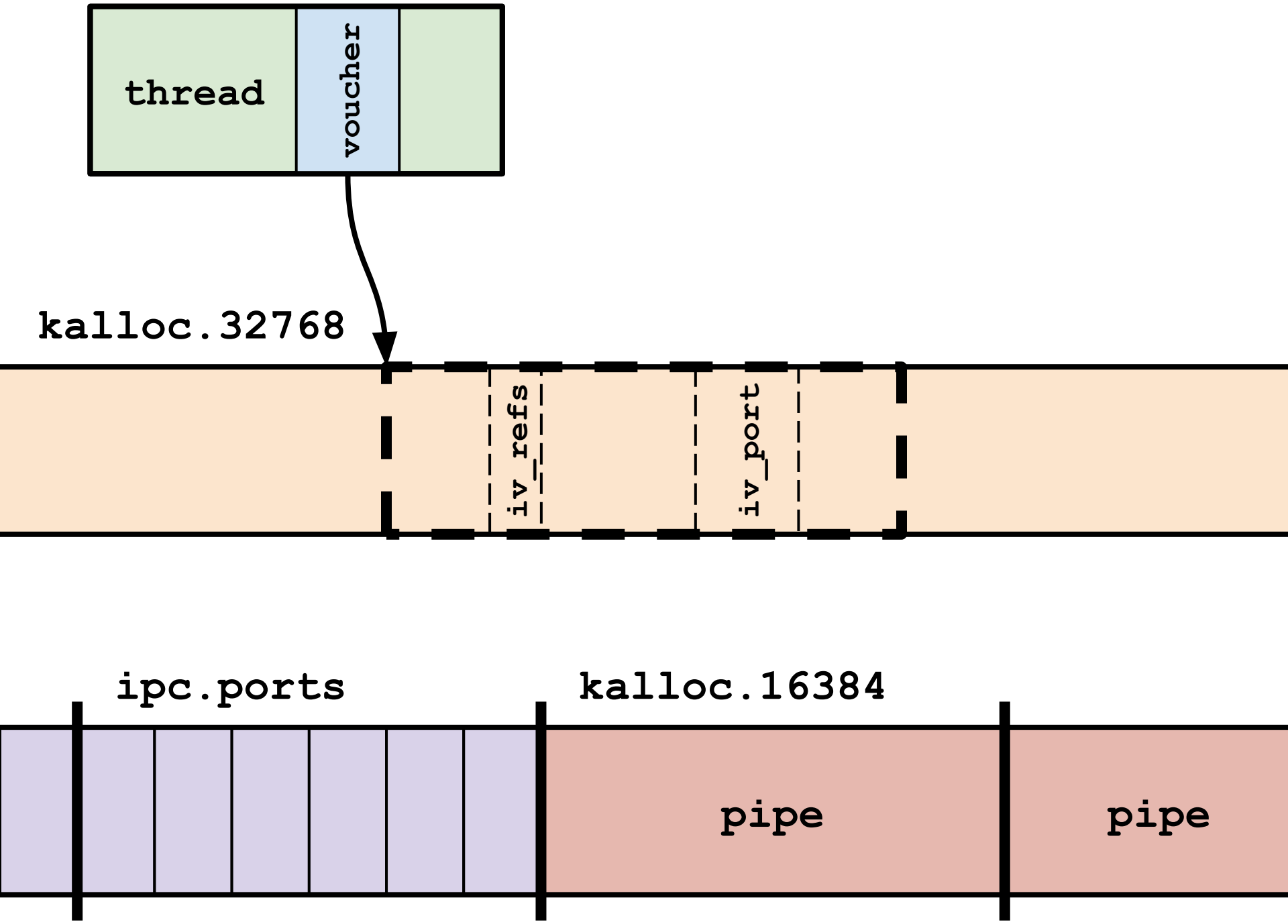

ipc.vouchers
iv_refs
iv_port
iv_refs
iv_port
iv_refs
iv_port
iv_refs = 1

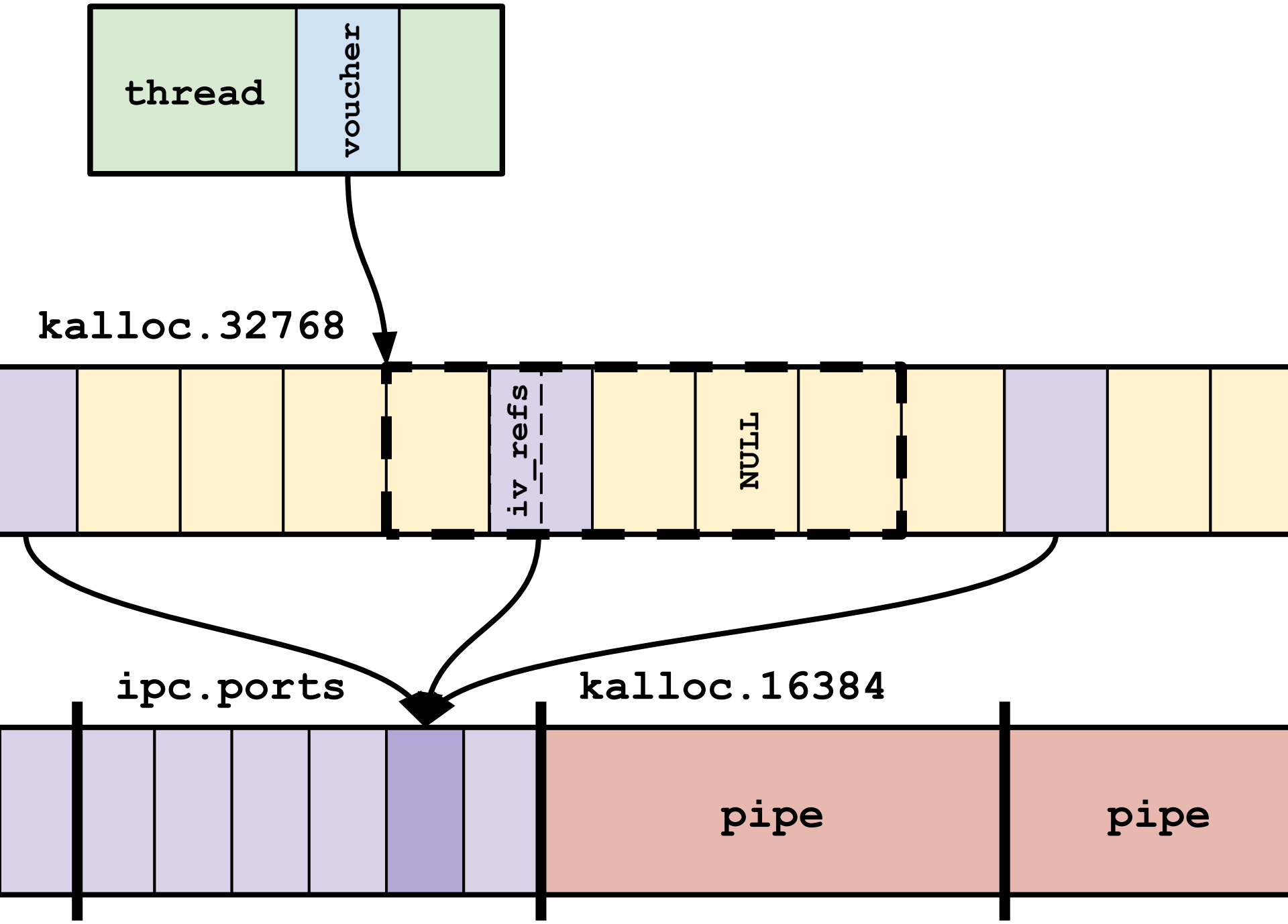

ipc.ports
kalloc.16384
pipe
pipe

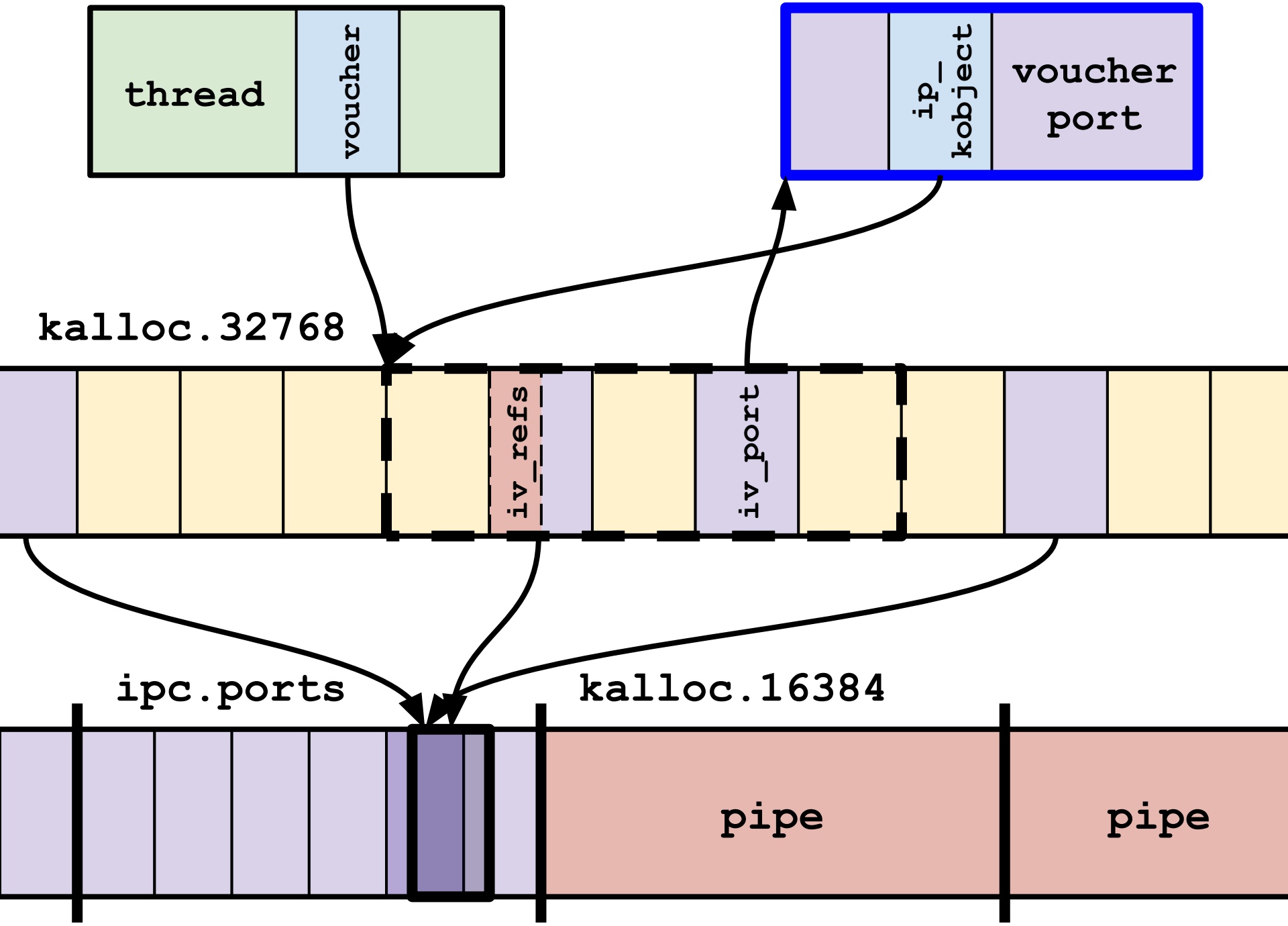

thread
voucher
ipc.vouchers
iv_refs
iv_port
iv_refs
iv_port
iv_refs
iv_port
iv_refs = 2
ipc.ports
kalloc.16384
pipe
pipe

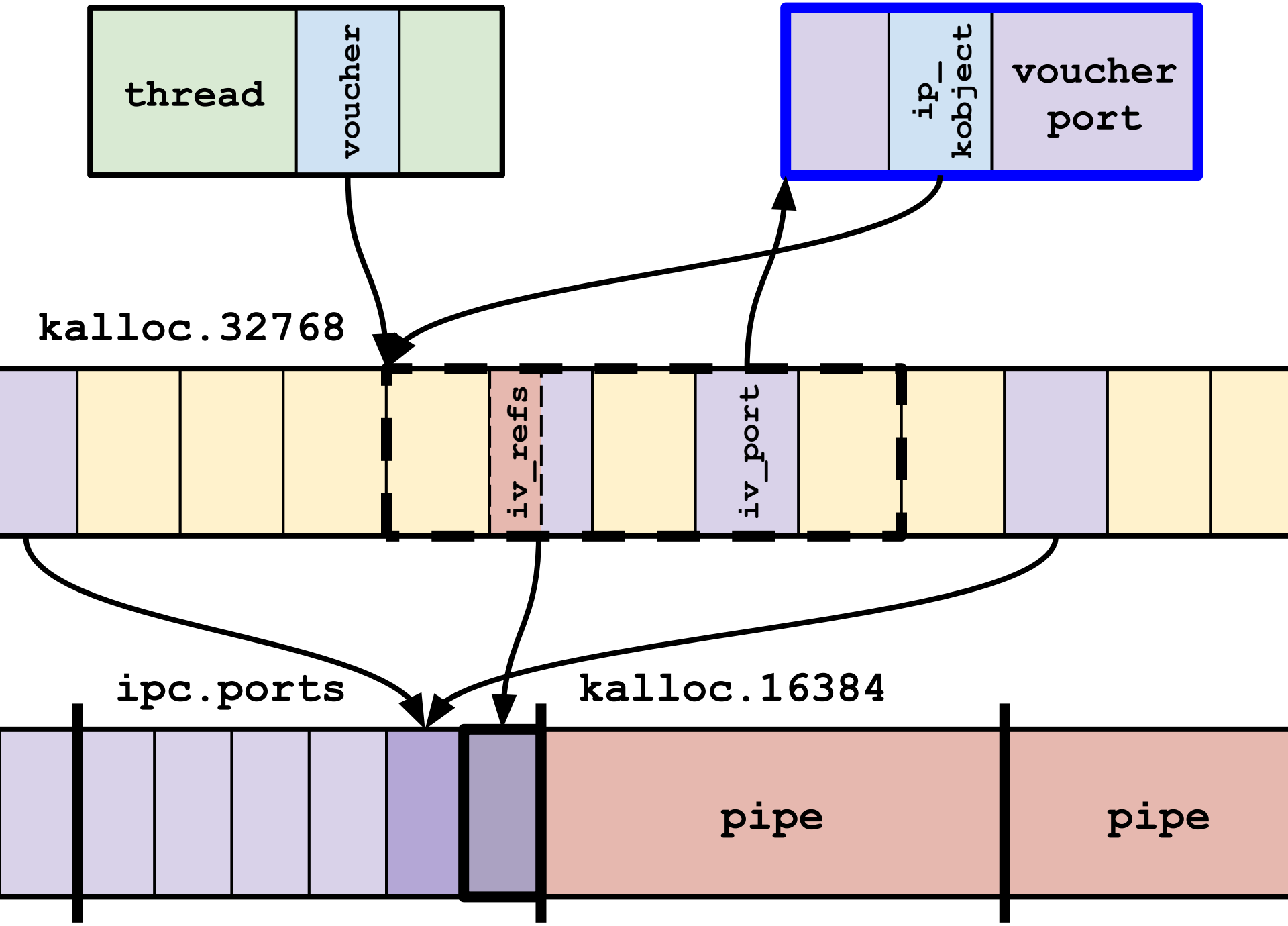

thread
voucher
ipc.vouchers
iv_refs
iv_port
iv_refs
iv_port
iv_refs
iv_port
iv_refs = 1
ipc.ports
kalloc.16384
pipe
pipe

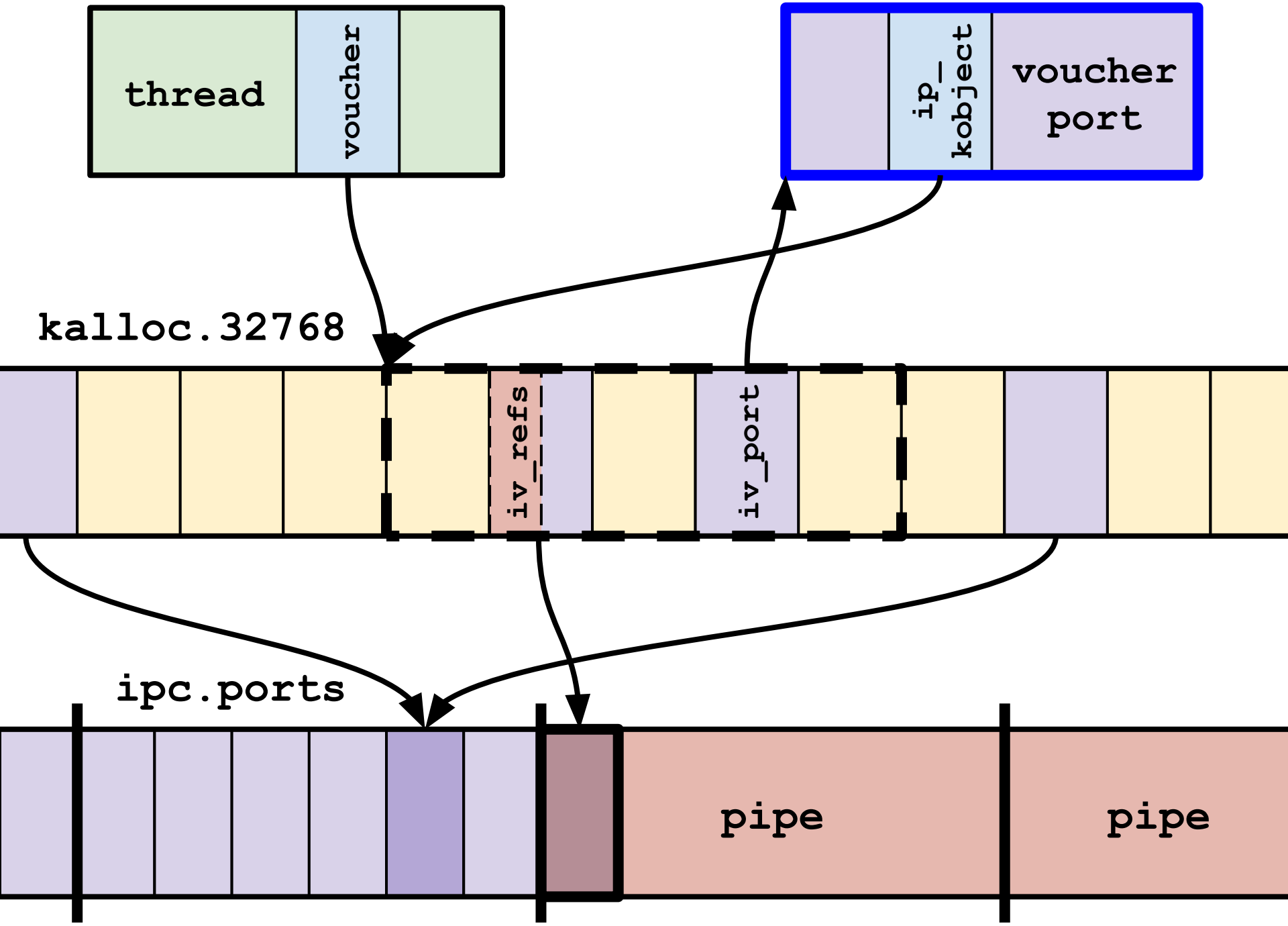

thread
voucher
ipc.vouchers
iv_refs
iv_port
iv_refs = 0
ipc.ports
kalloc.16384
pipe
pipe

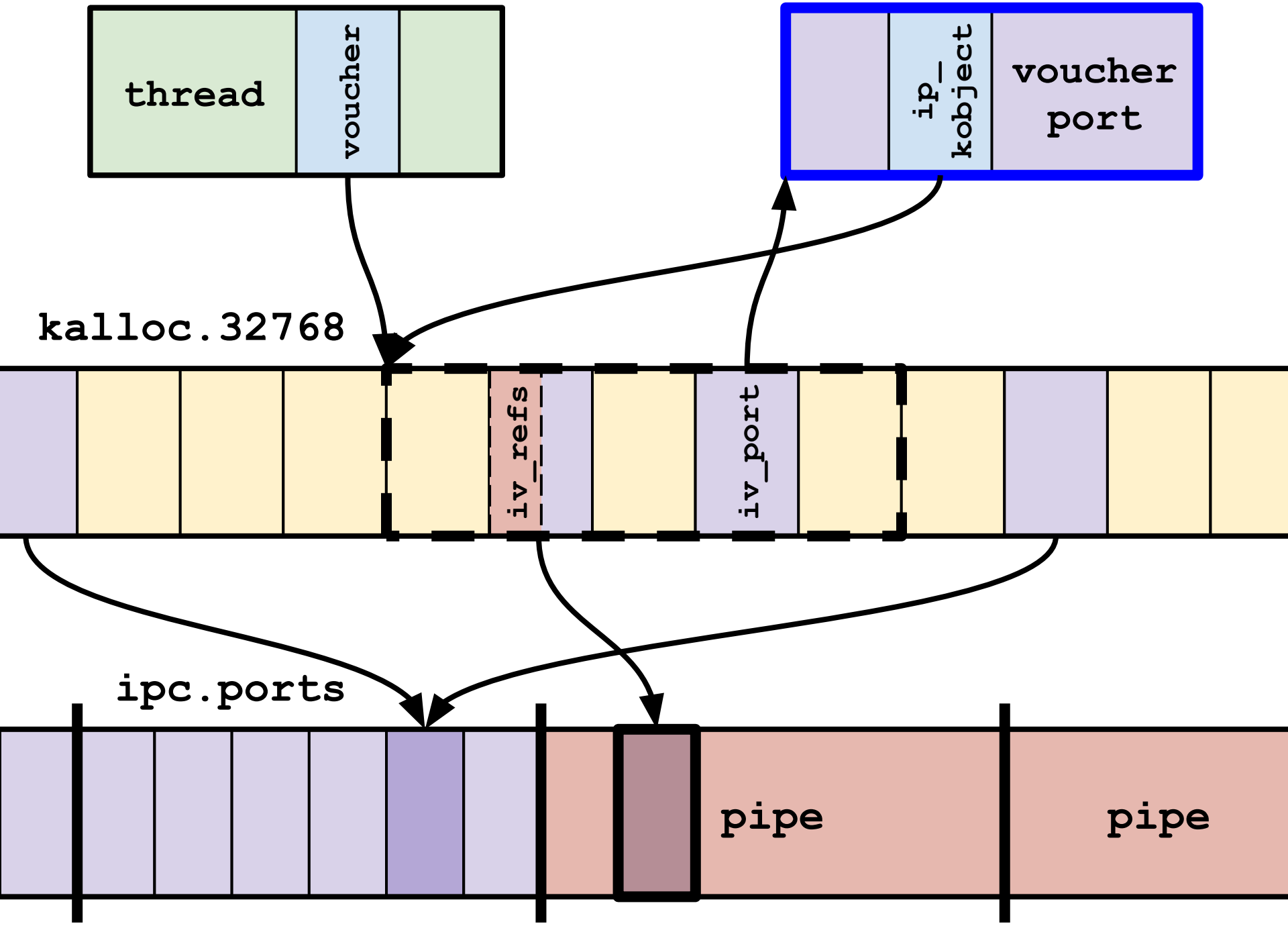

thread
voucher
kalloc.32768
iv_refs
iv_port
ipc.ports
kalloc.16384
pipe
pipe

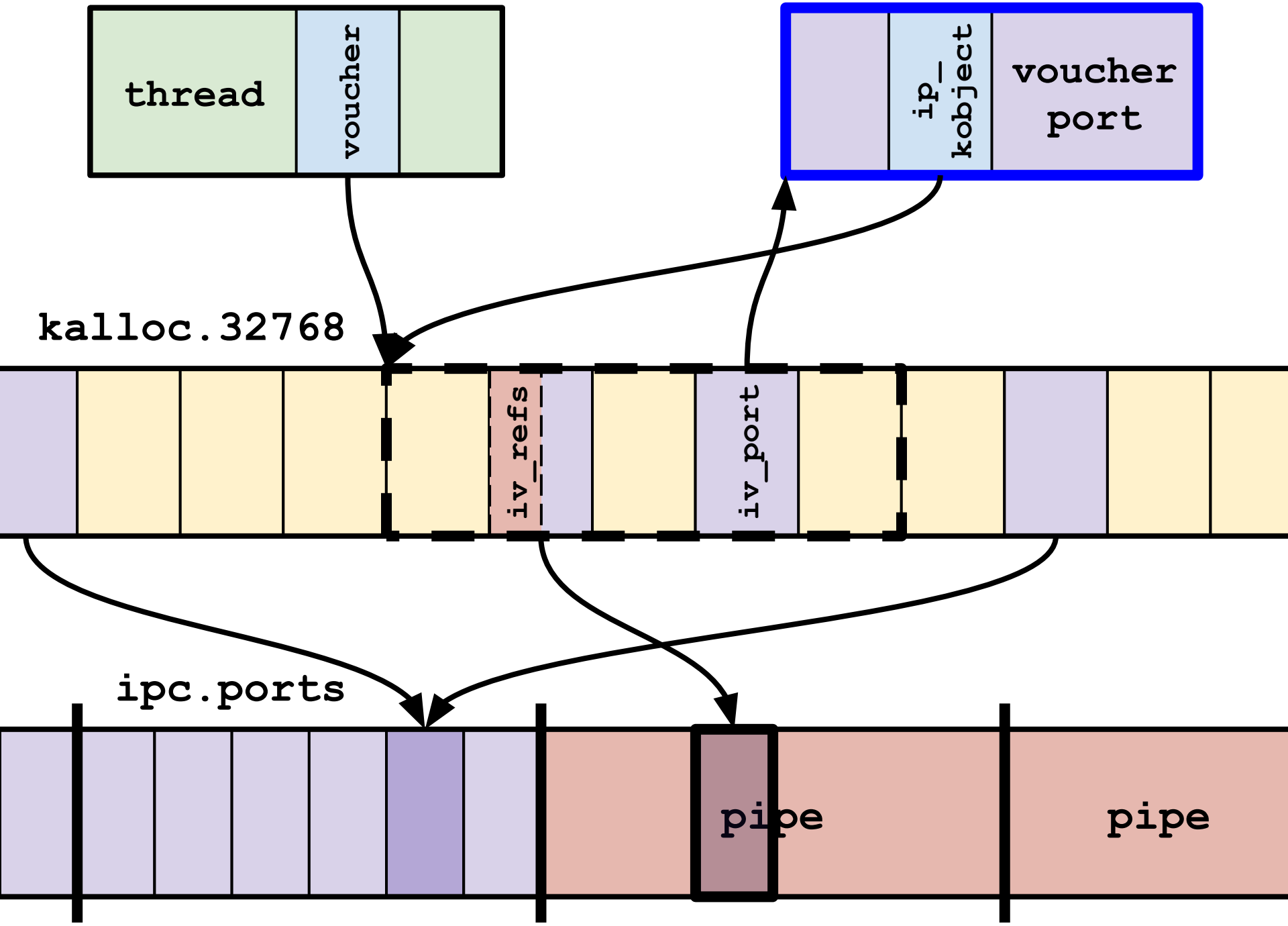

thread
voucher
kalloc.32768
iv_refs
NULL
ipc.ports
kalloc.16384
pipe
pipe

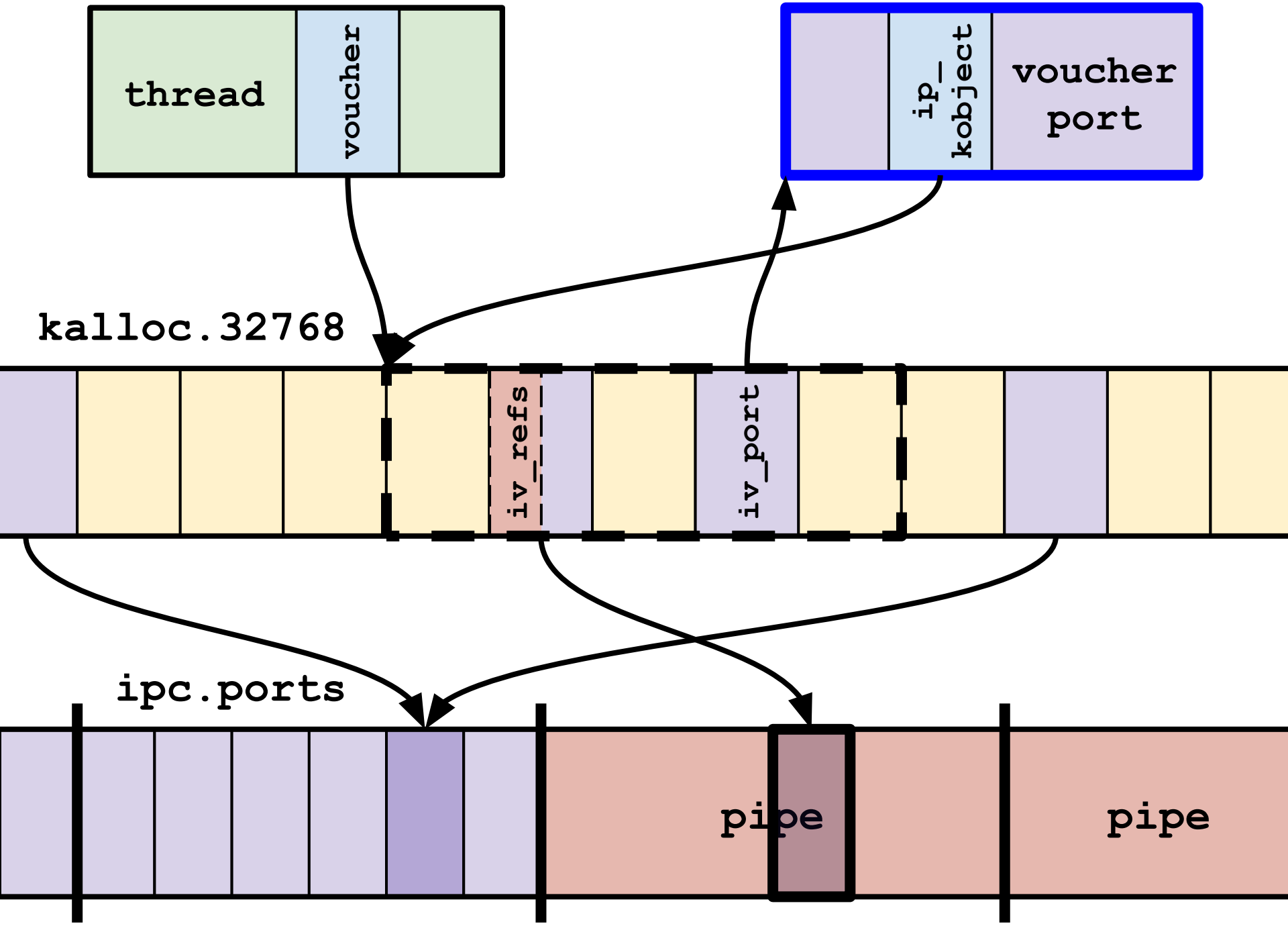

thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
kalloc.16384
pipe
pipe

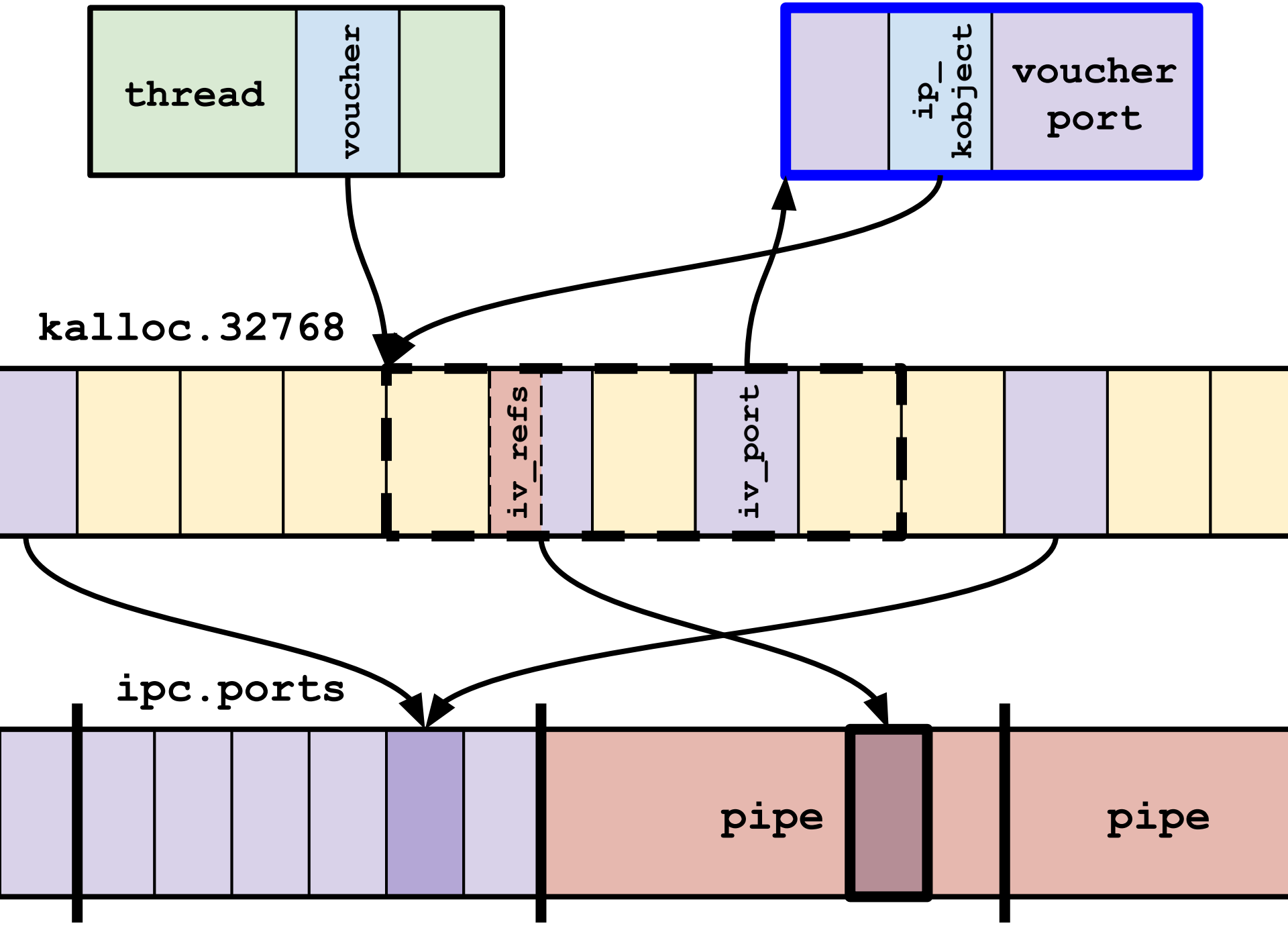

thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
kalloc.16384
pipe
pipe

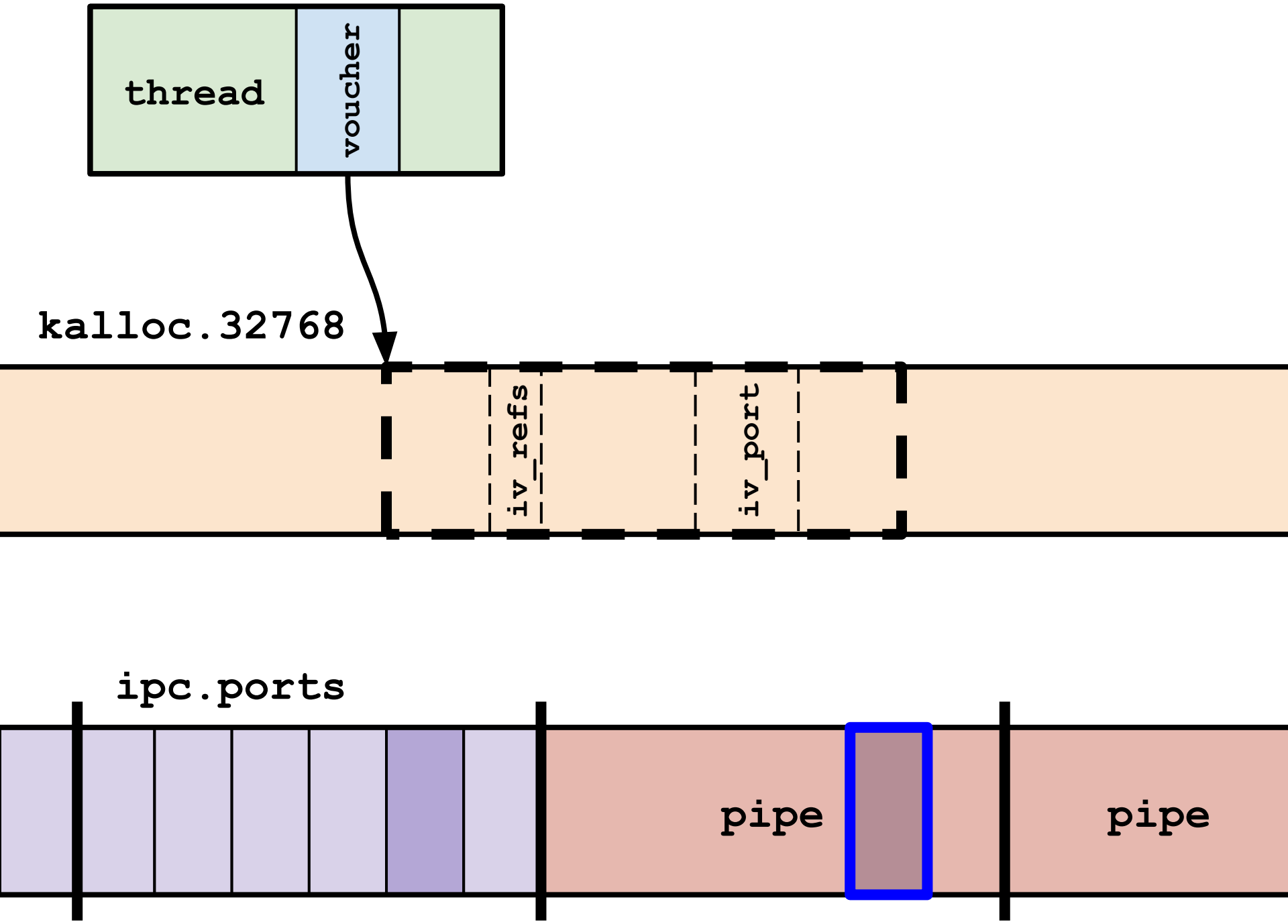

thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
pipe
pipe

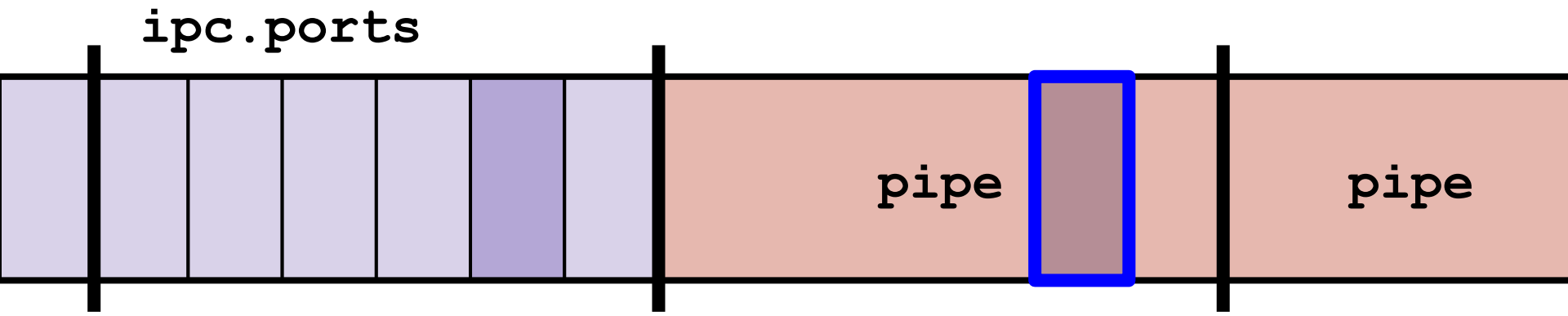

thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
pipe
pipe

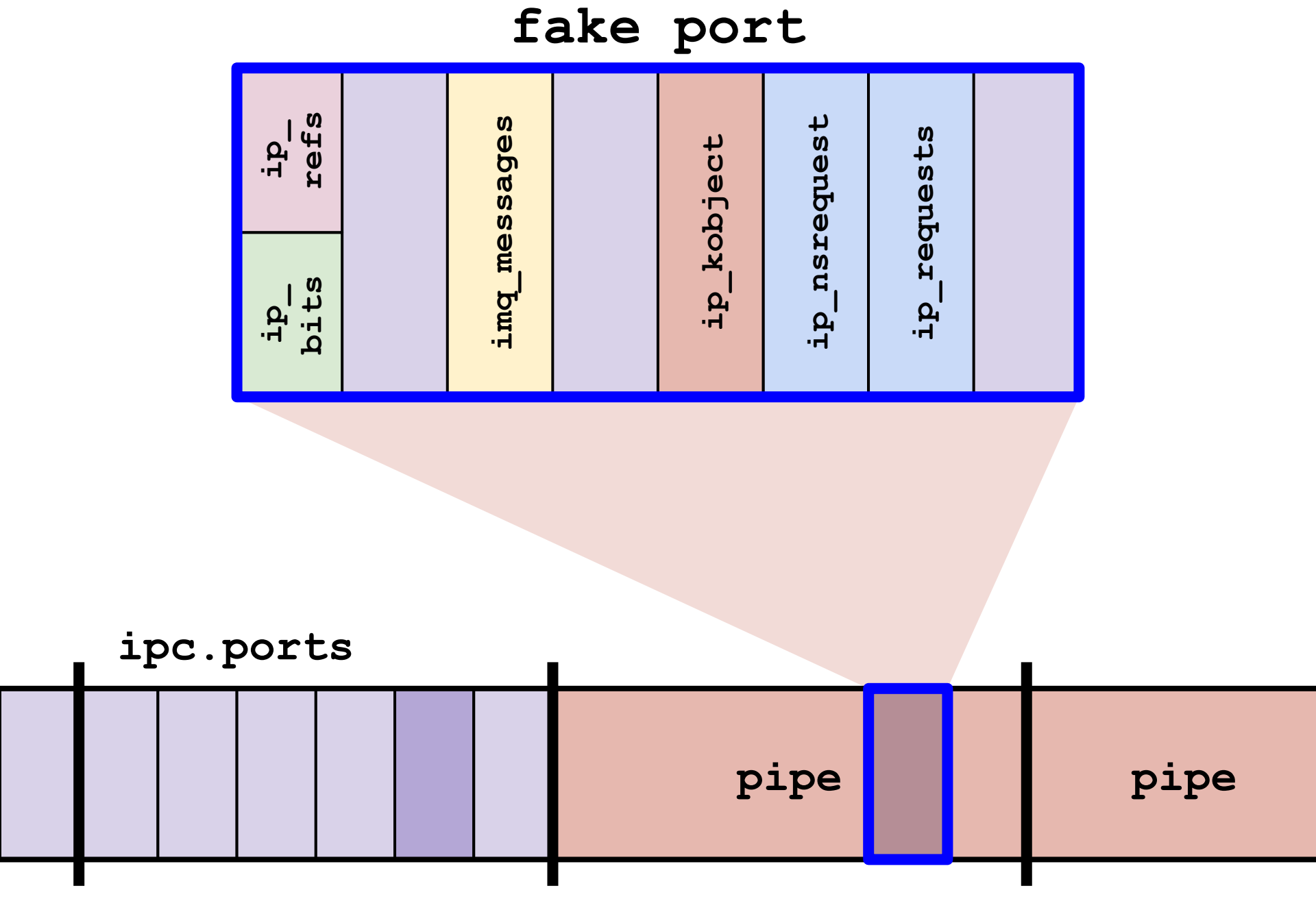

thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
pipe
pipe


thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
pipe
pipe


thread
voucher
ip_
kobject
voucher
port
kalloc.32768
iv_refs
iv_port
ipc.ports
pipe
pipe

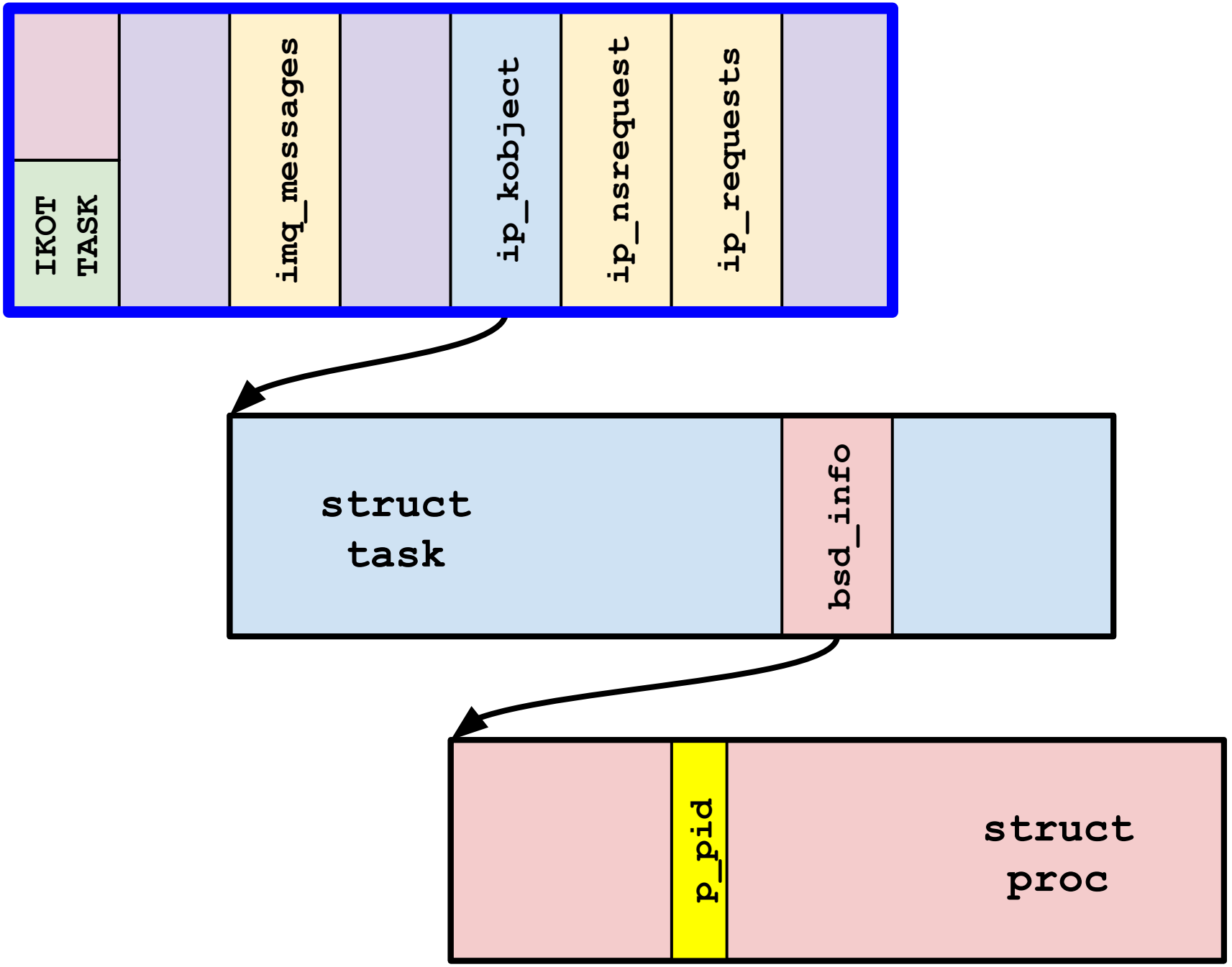

thread
voucher
kalloc.32768
iv_refs
iv_port
ipc.ports
pipe
pipe

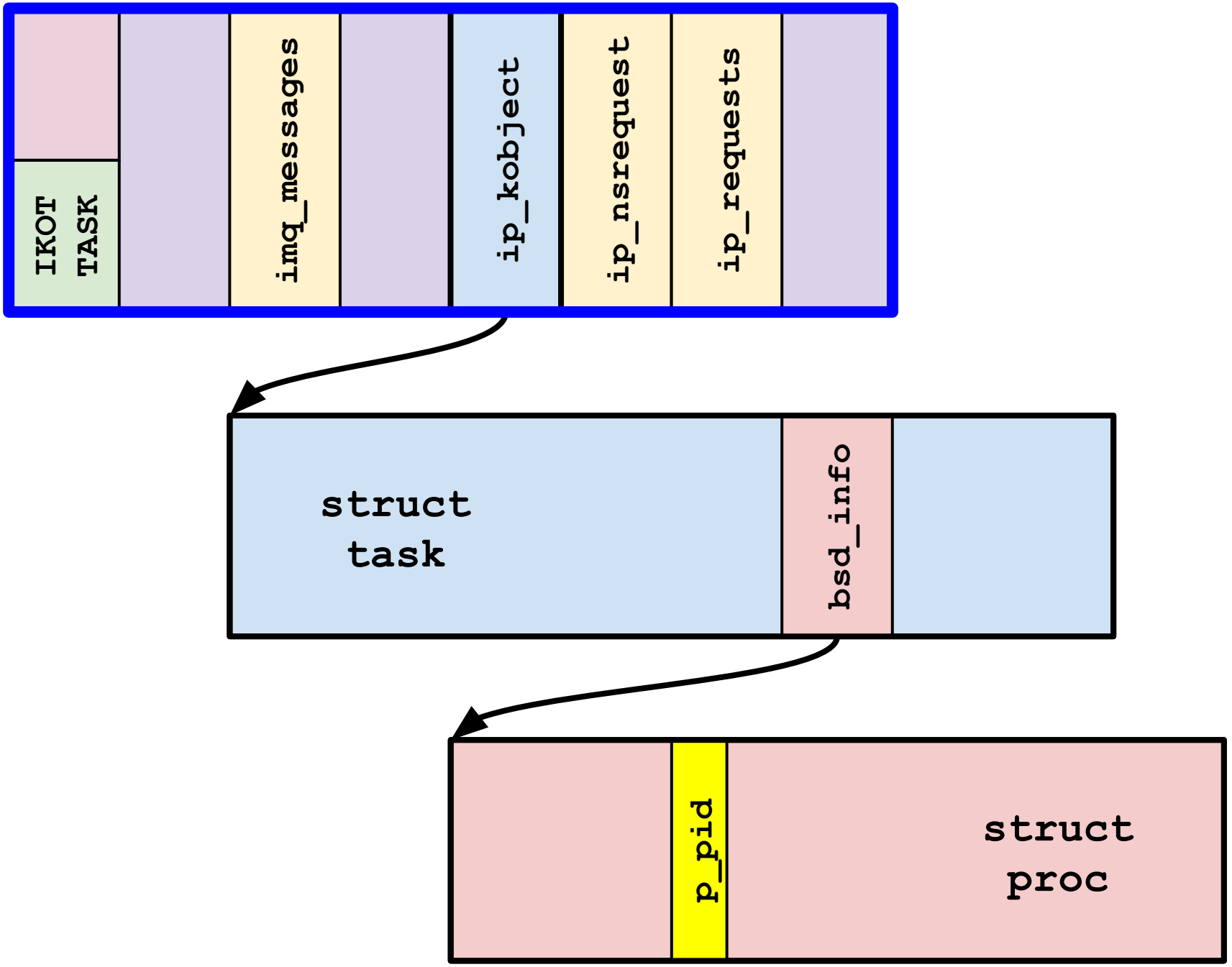

ipc.ports
pipe
pipe

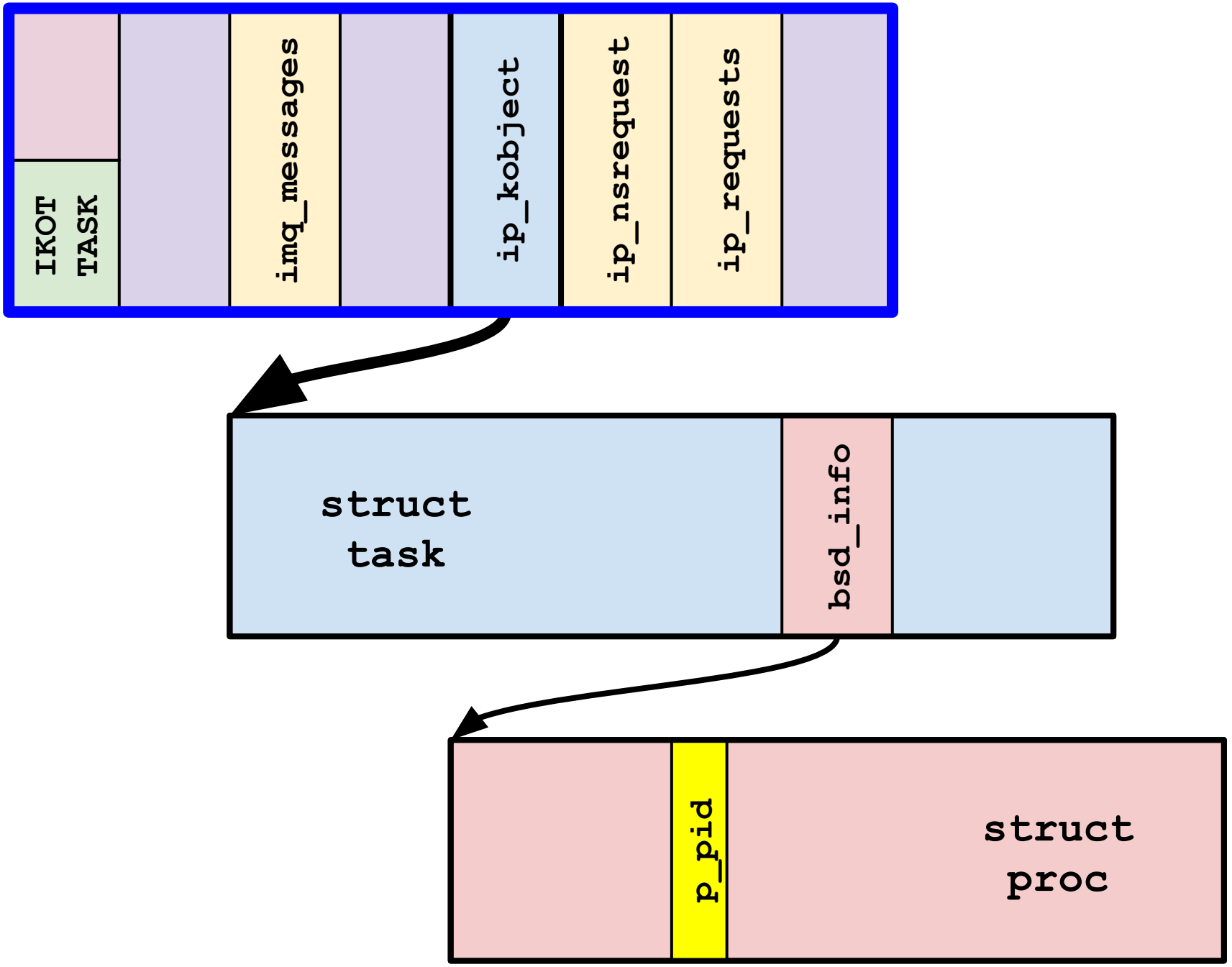

fake port
ip_ refs
ip_ bits
imq_messages
ip_kobject
ip_nsrequest
ip_requests
ipc.ports
pipe
pipe

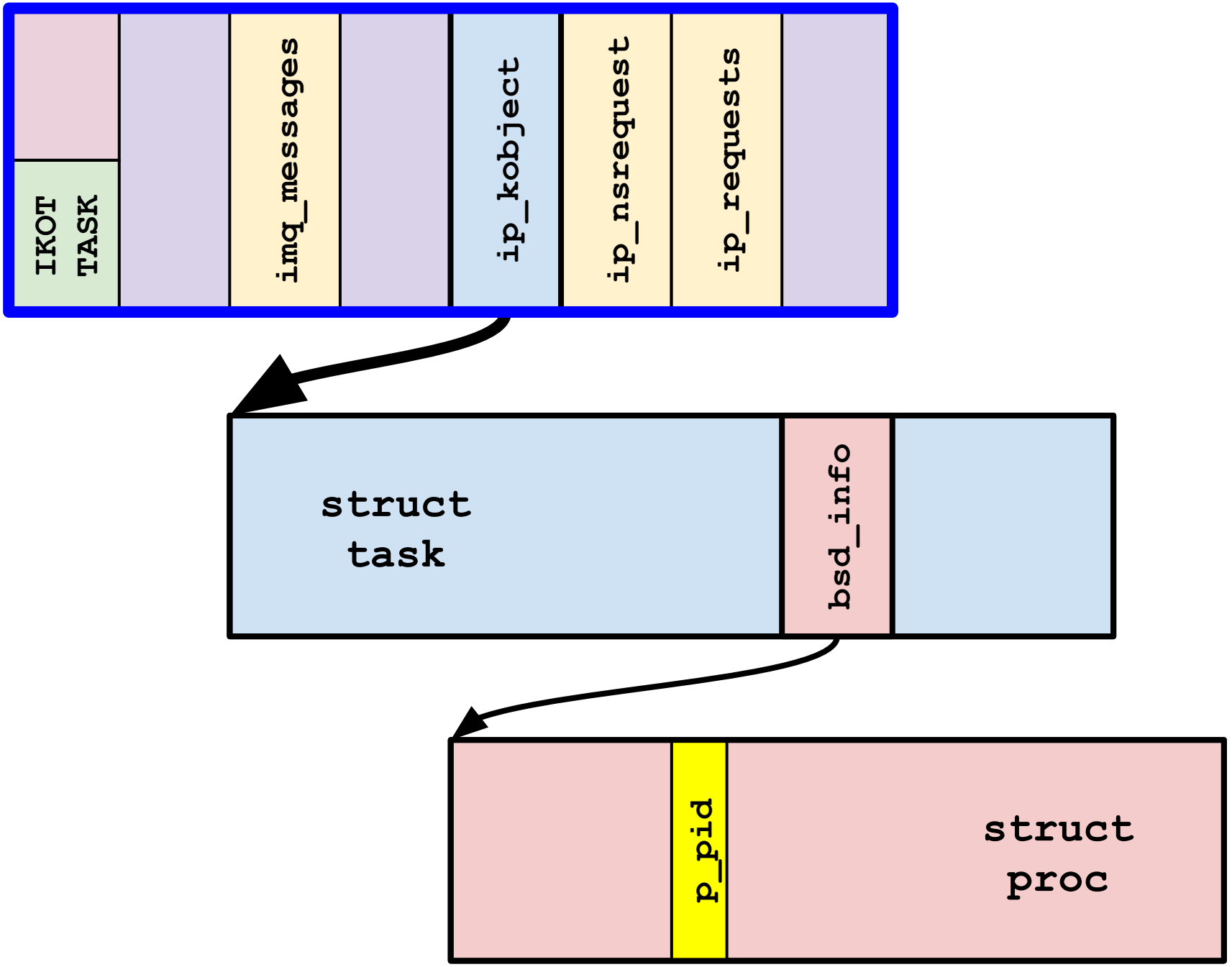
From controlled port to kernel read/write

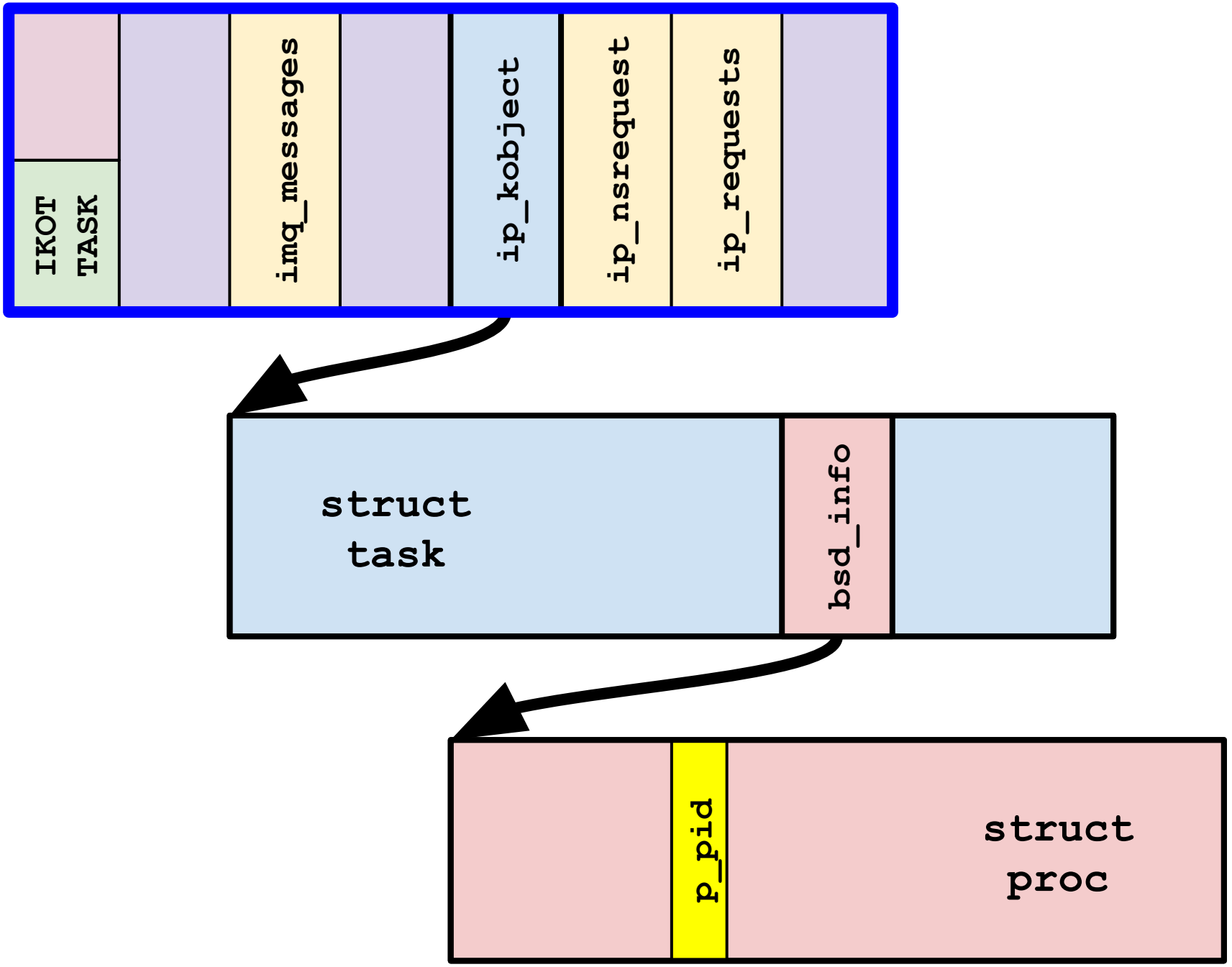
Use
`pid_for_task()`
as a basic 32-bit
read primitive

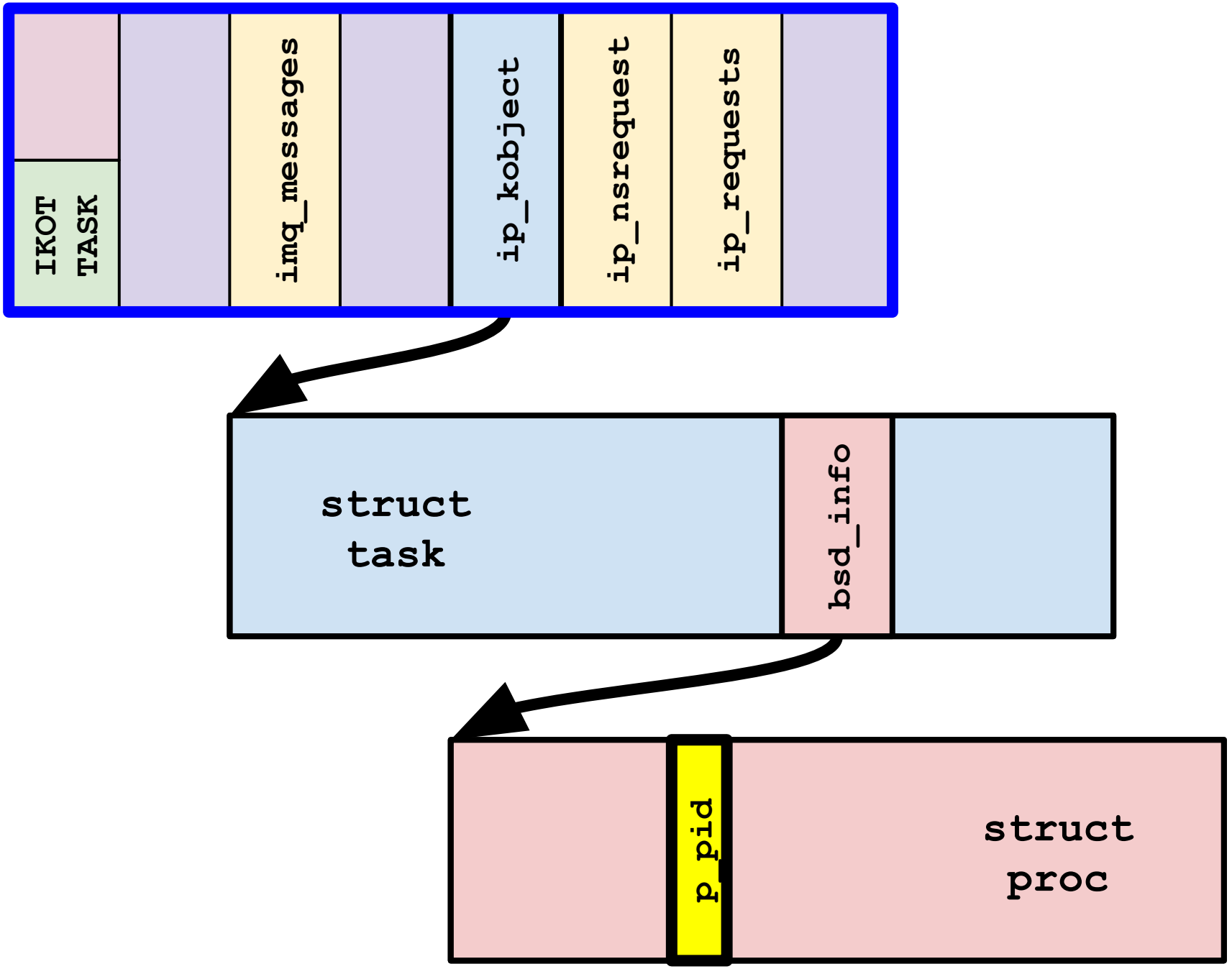

IKOT
TASK
imq_messages
ip_kobject
ip_nsrequest
ip_requests
struct
task
bsd_info
p_pid
struct
proc


IKOT
TASK
imq_messages
ip_kobject
ip_nsrequest
ip_requests
struct
task
bsd_info
p_pid
struct
proc


IKOT TASK
imq_messages
ip_kobject
ip_nsrequest
ip_requests
struct task
bsd_info
p_pid
struct proc

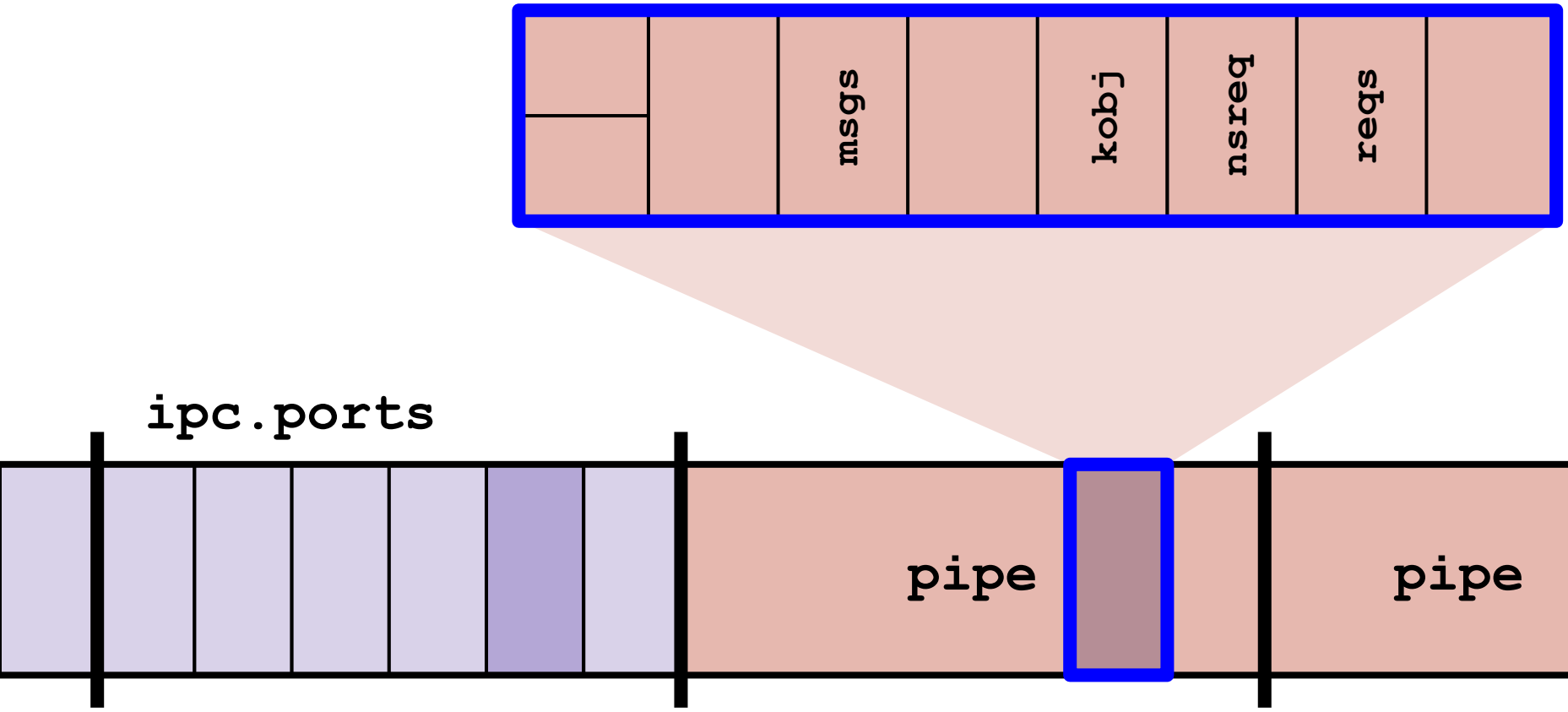

IKOT
TASK
imq_messages
ip_kobject
ip_nsrequest
ip_requests
struct
task
bsd_info
p_pid
struct
proc

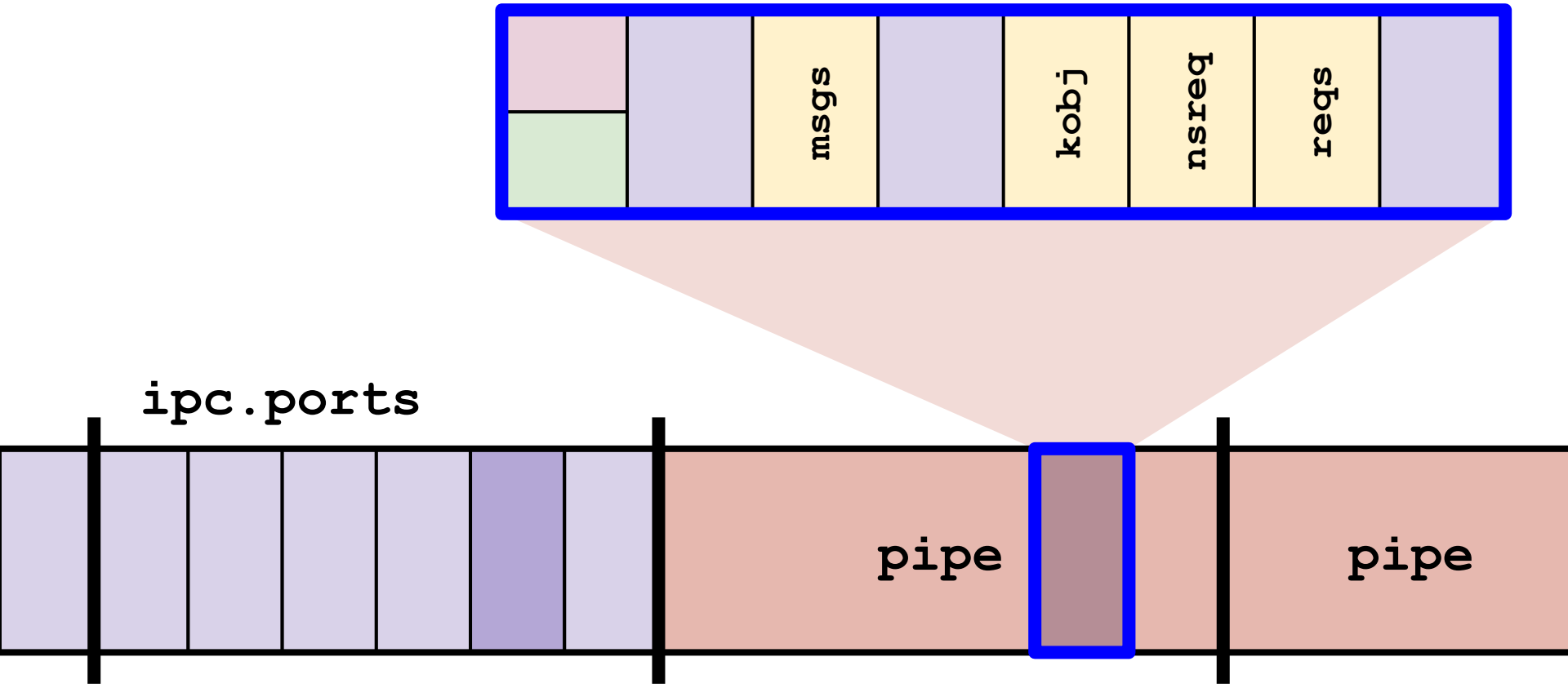

IKOT
TASK
imq_messages
ip_kobject
ip_nsrequest
ip_requests
struct
task
bsd_info
p_pid
struct
proc

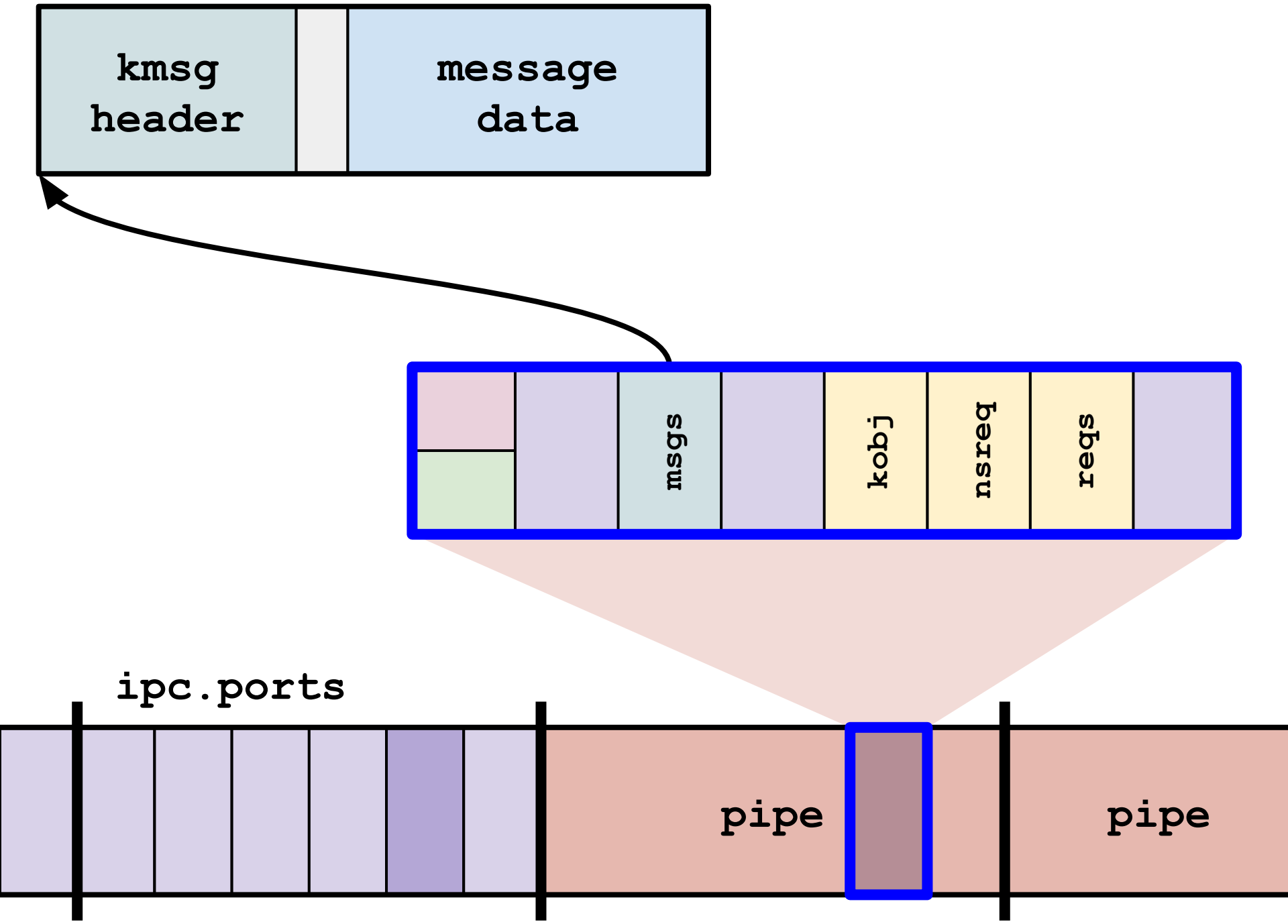

IKOT
TASK
imq_messages
ip_kobject
ip_nsrequest
ip_requests
struct
task
bsd_info
p_pid
struct
proc

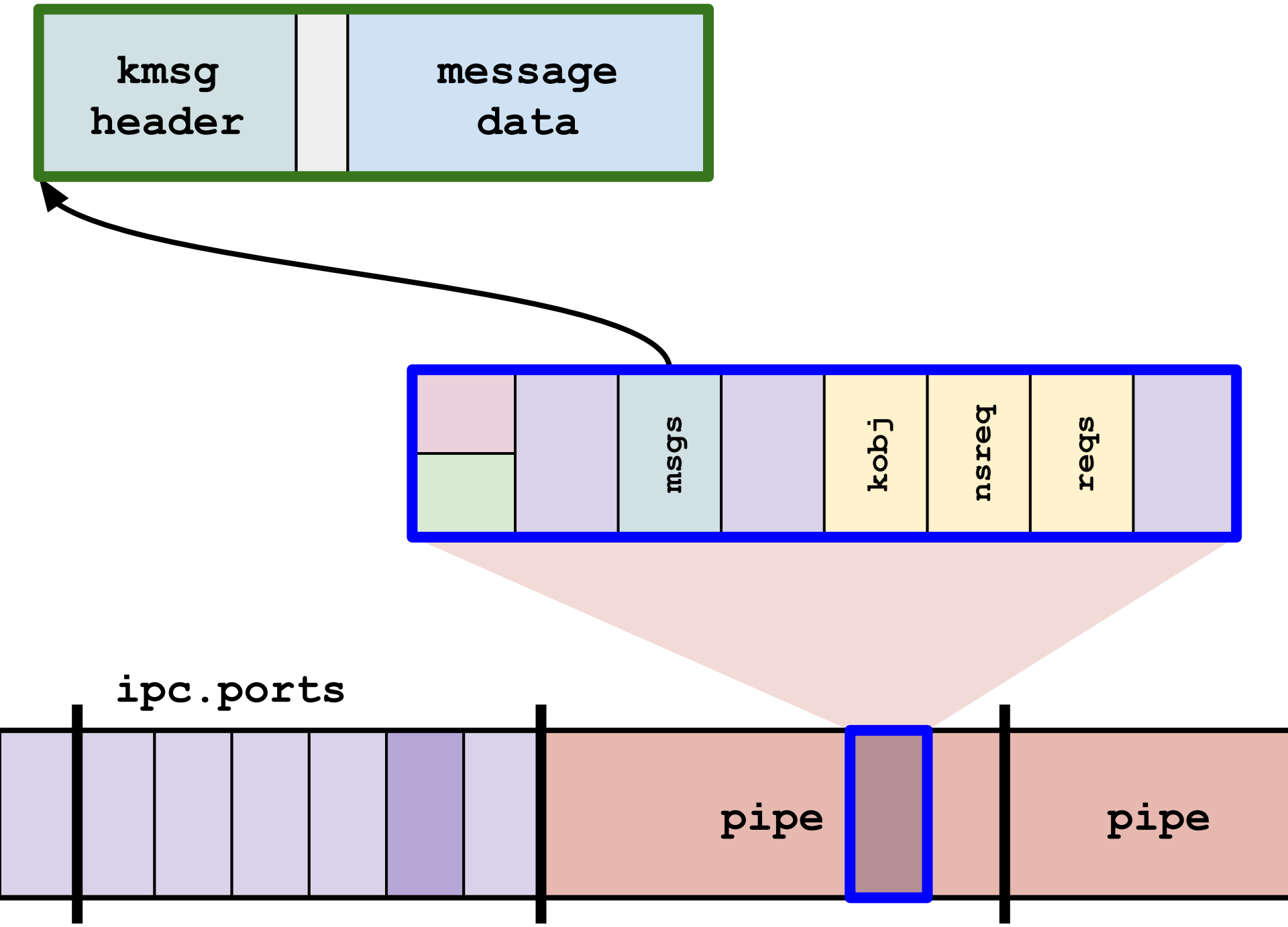
Requires a fake task
at a known address

Send a message to
our fake port

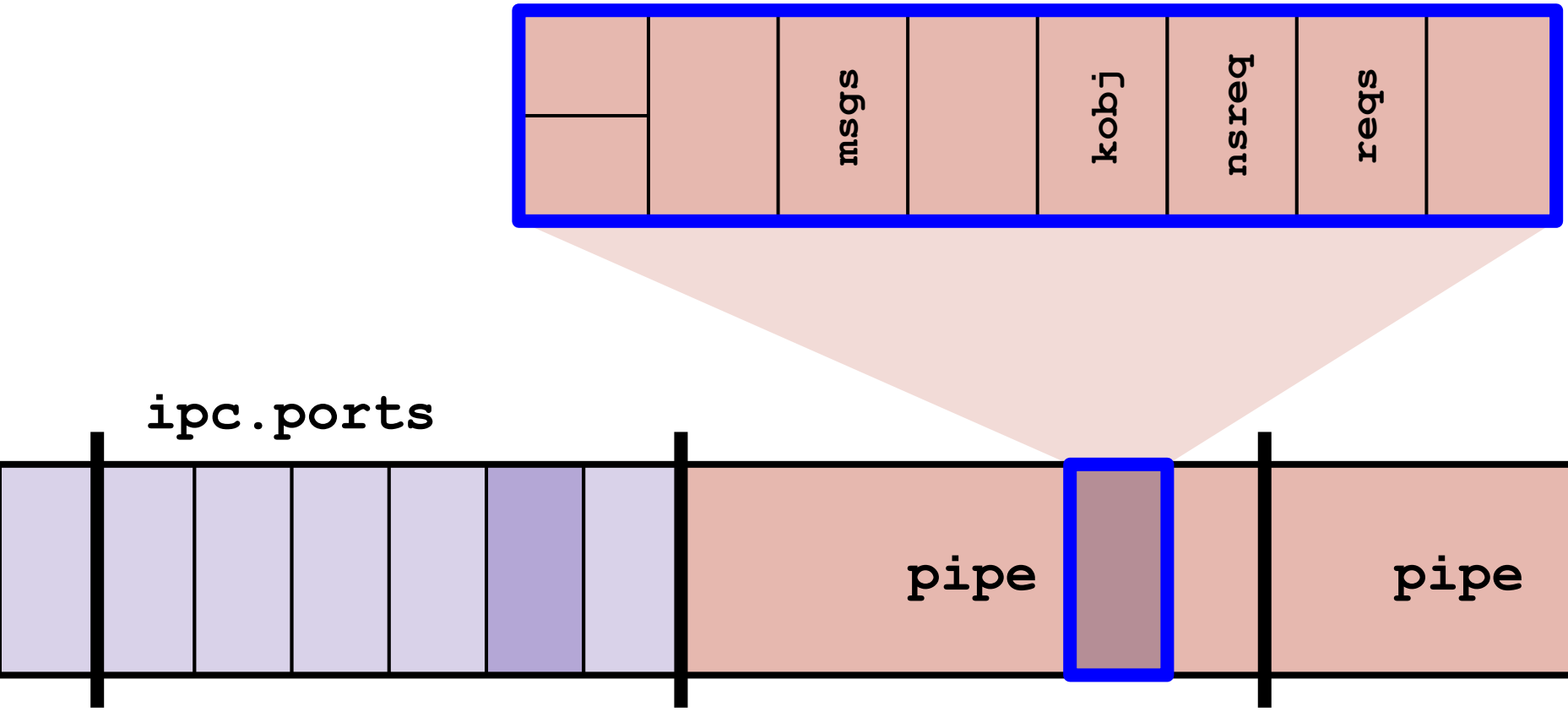

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

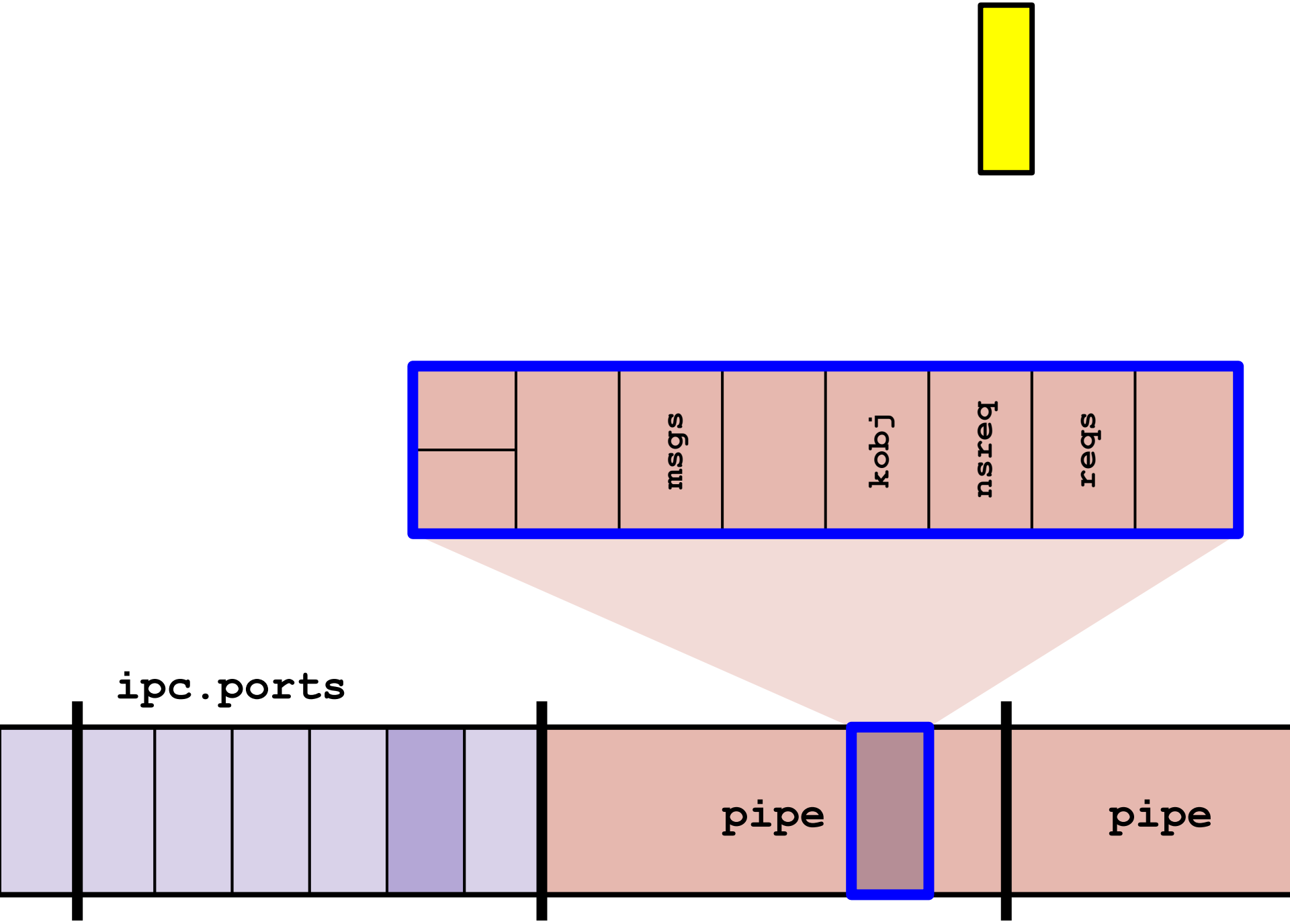

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

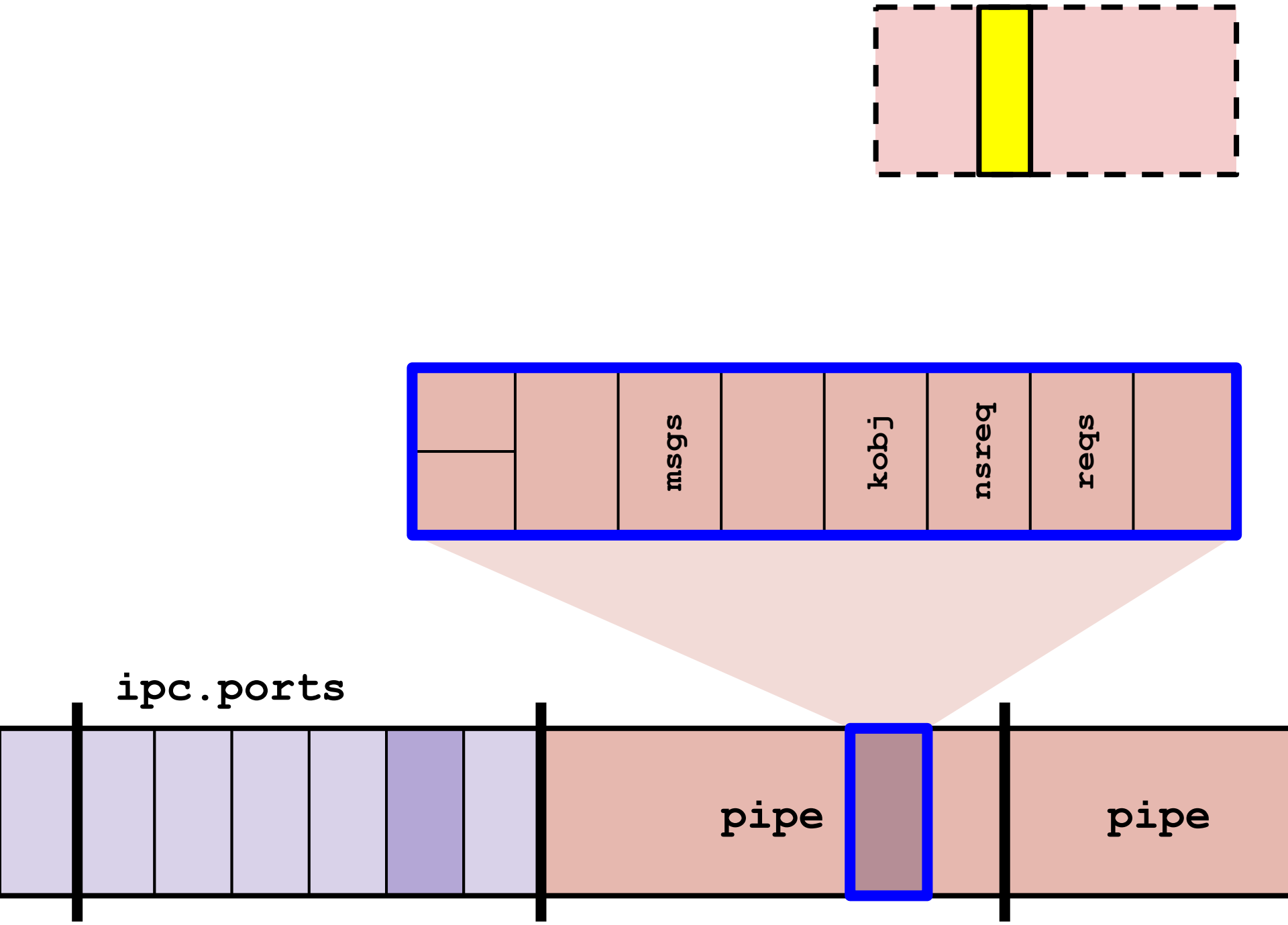

kmsg
header
message
data
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

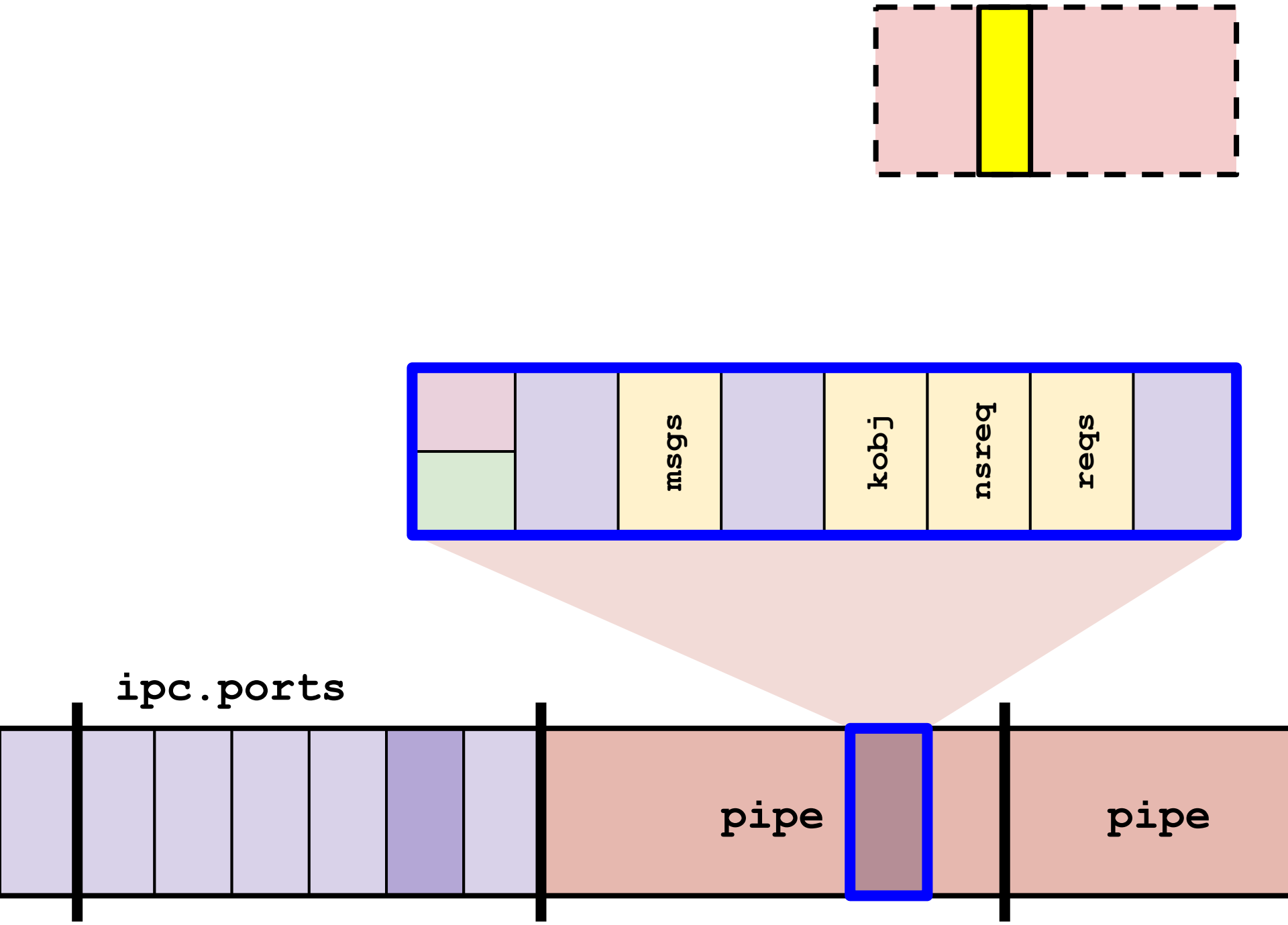

kmsg
header
message
data
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

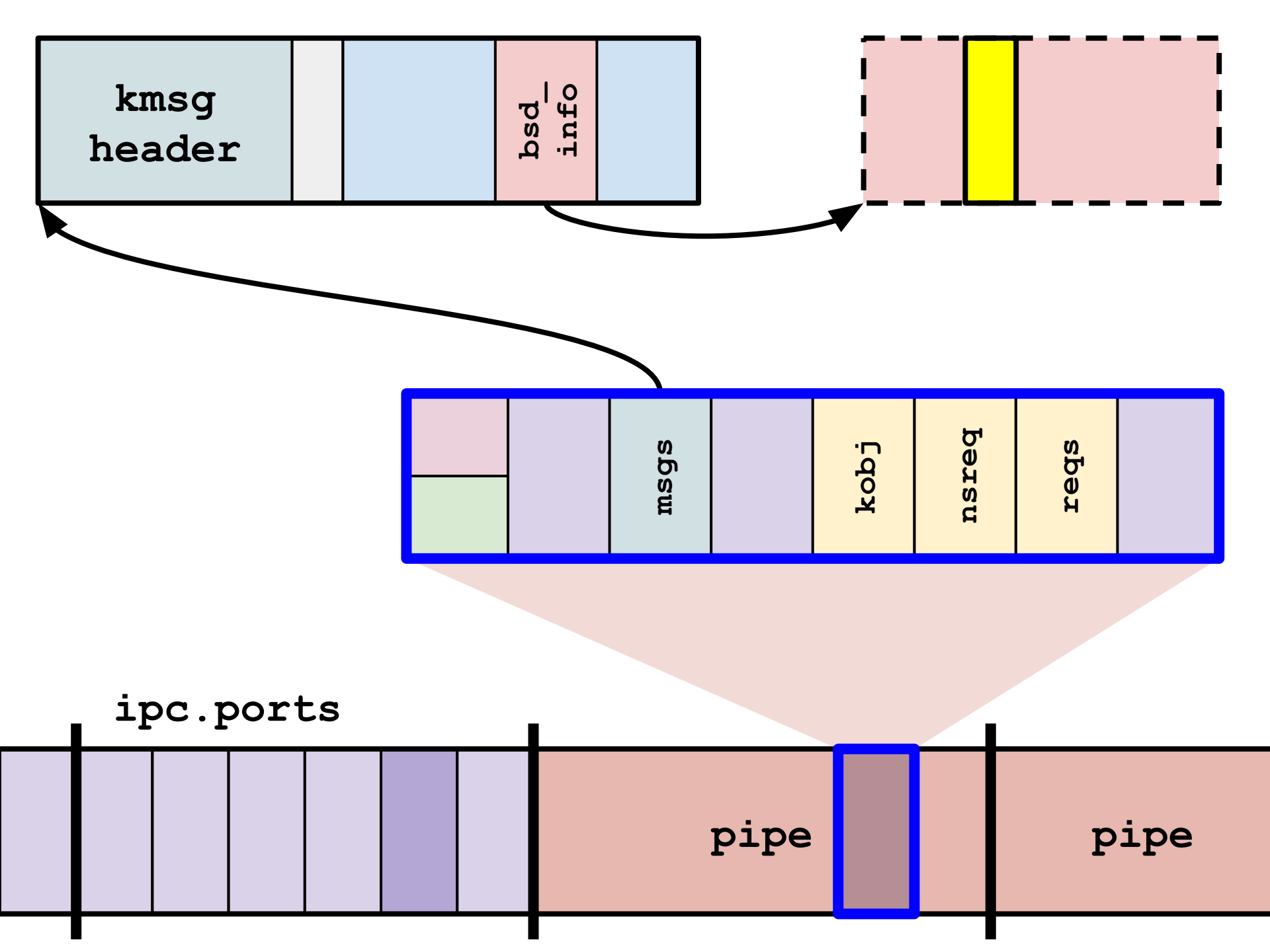
A basic 32-bit read primitive

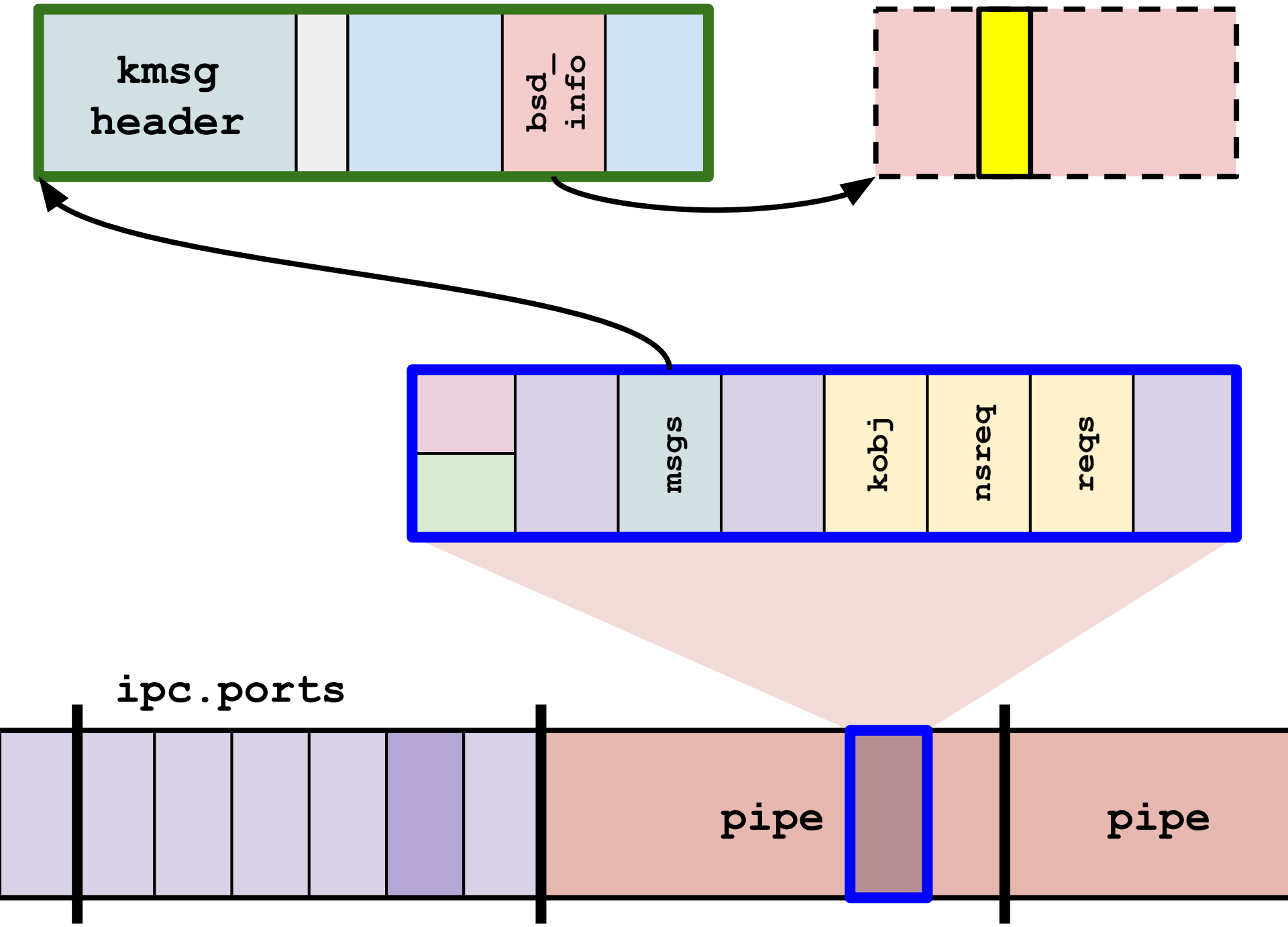

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

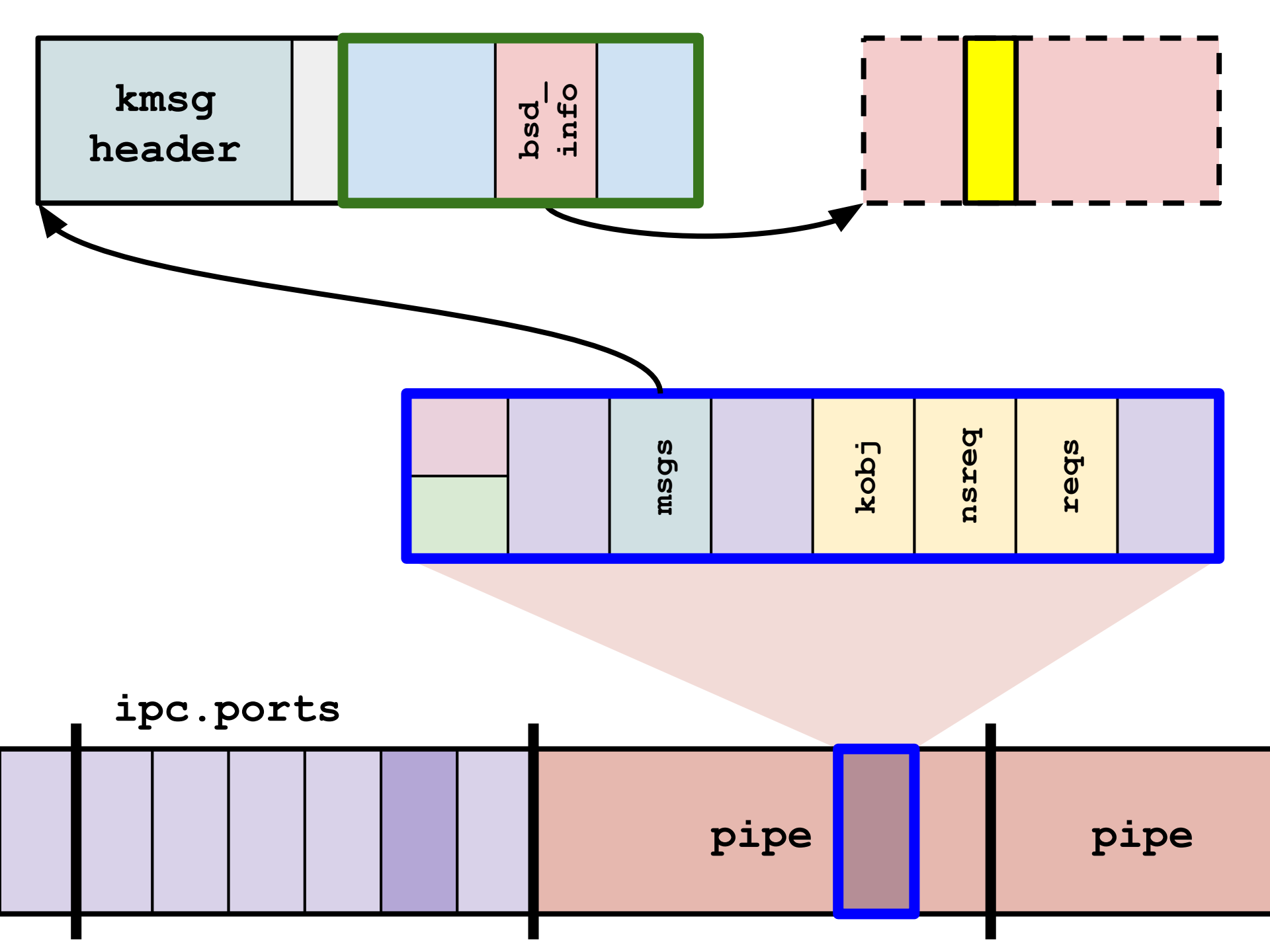

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

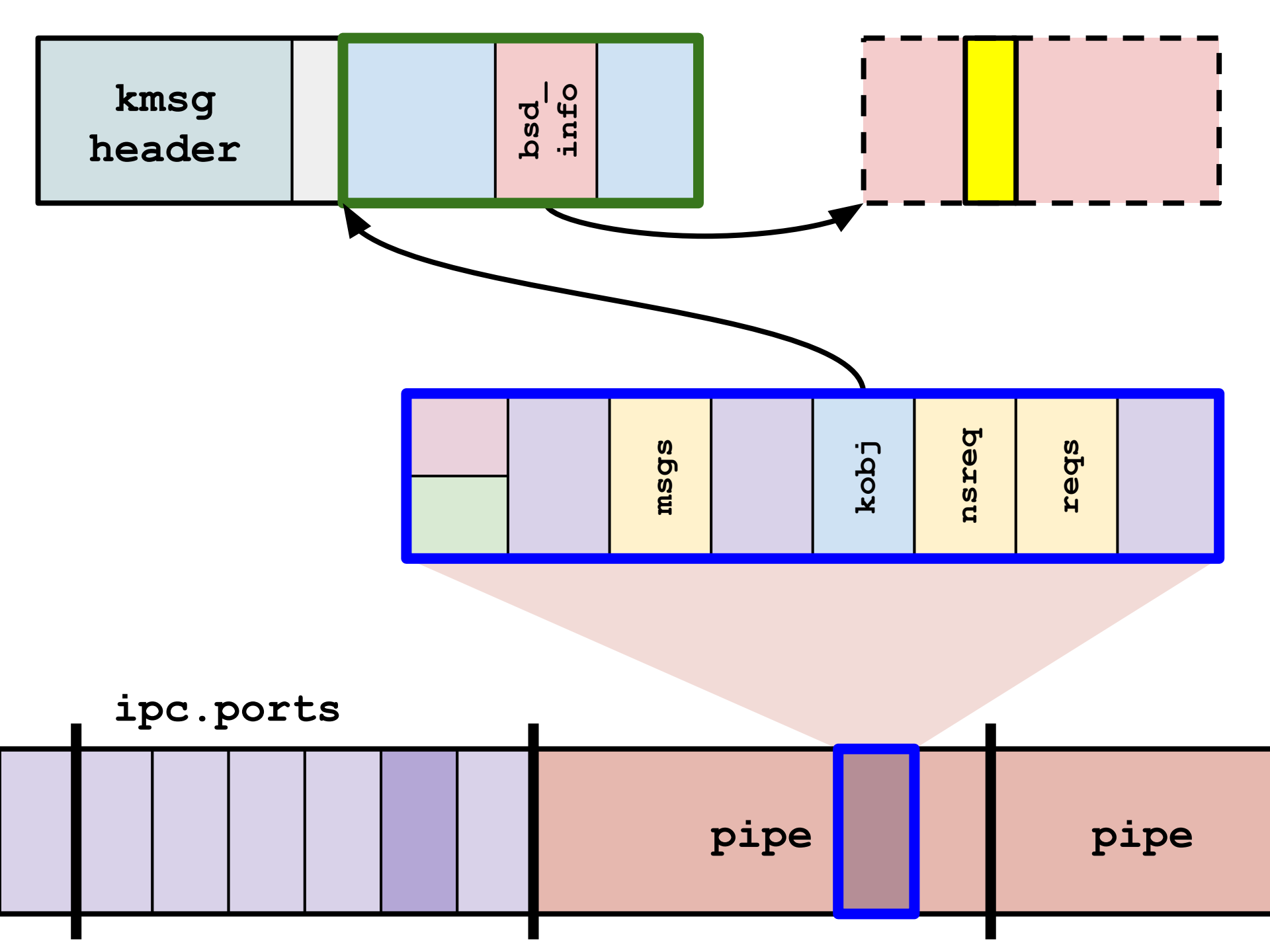

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

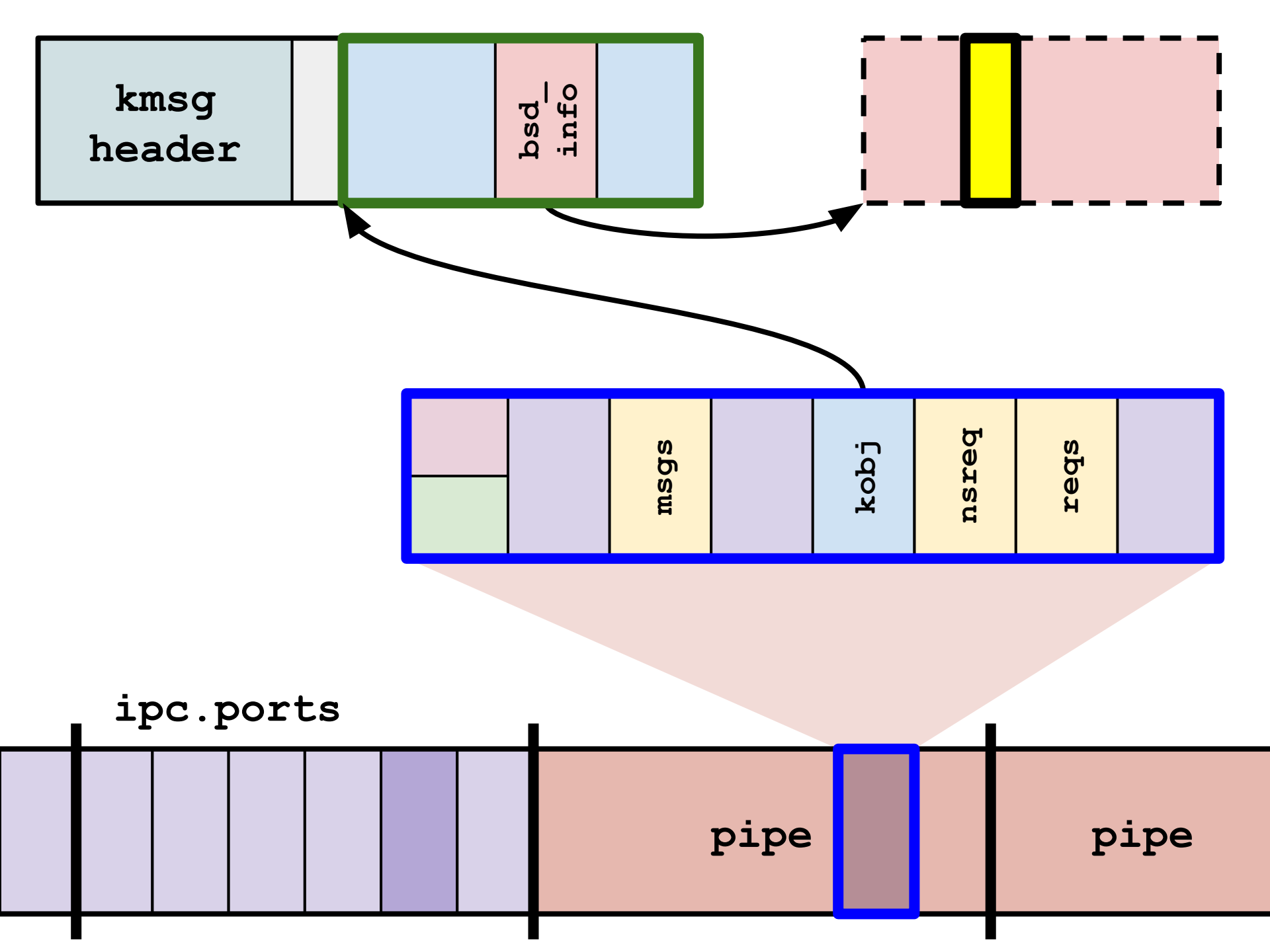

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe


kmsg
header
bsd_
info
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe


kmsg
header
bsd_
info
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

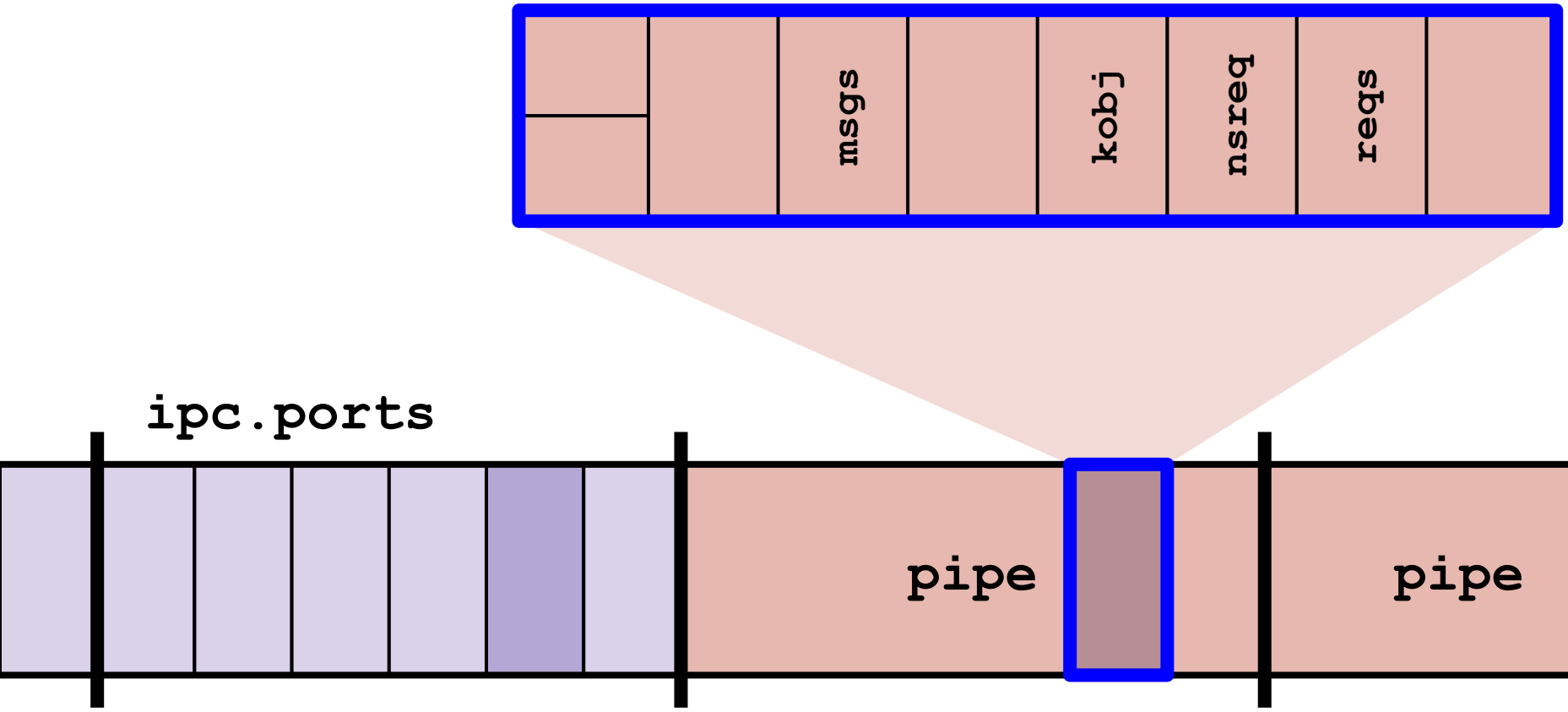

kmsg
header
bsd_
info
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

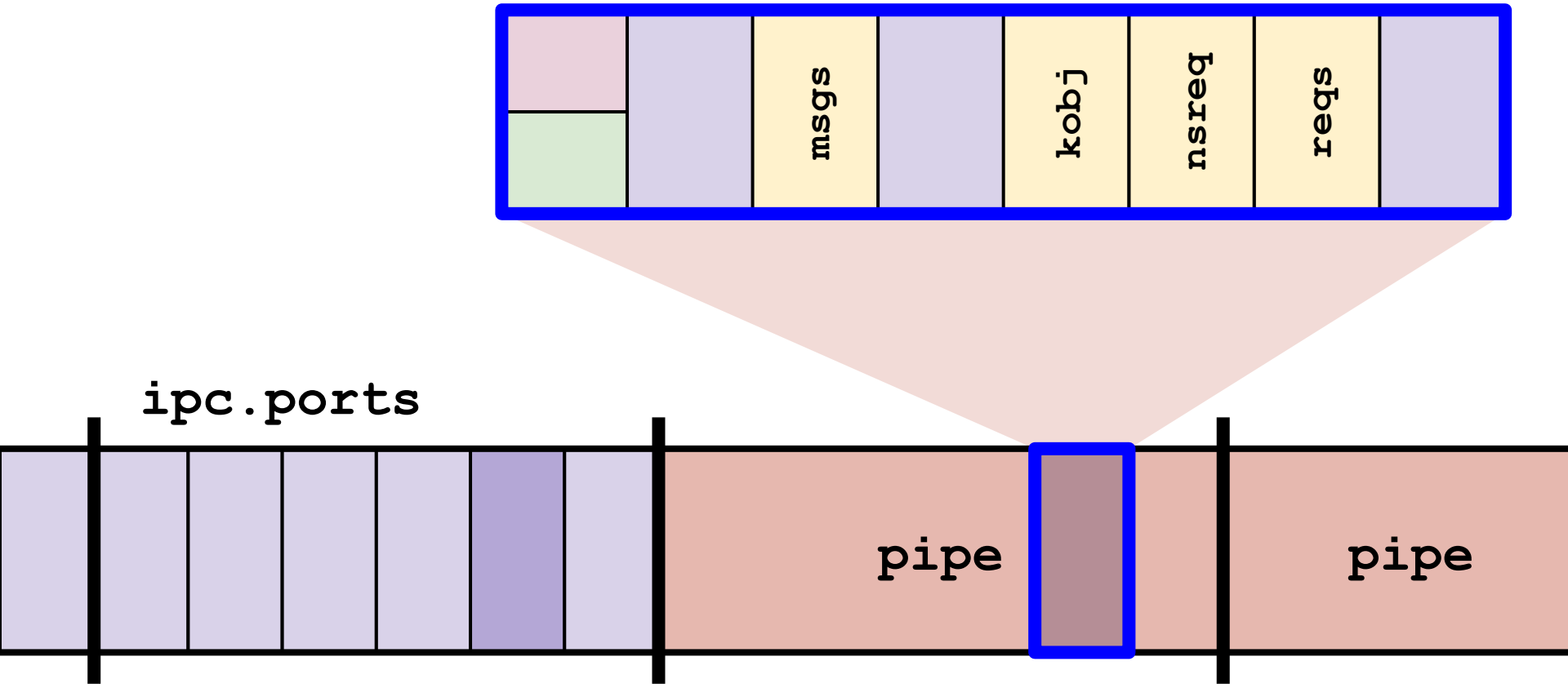

kmsg
header
bsd_
info
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

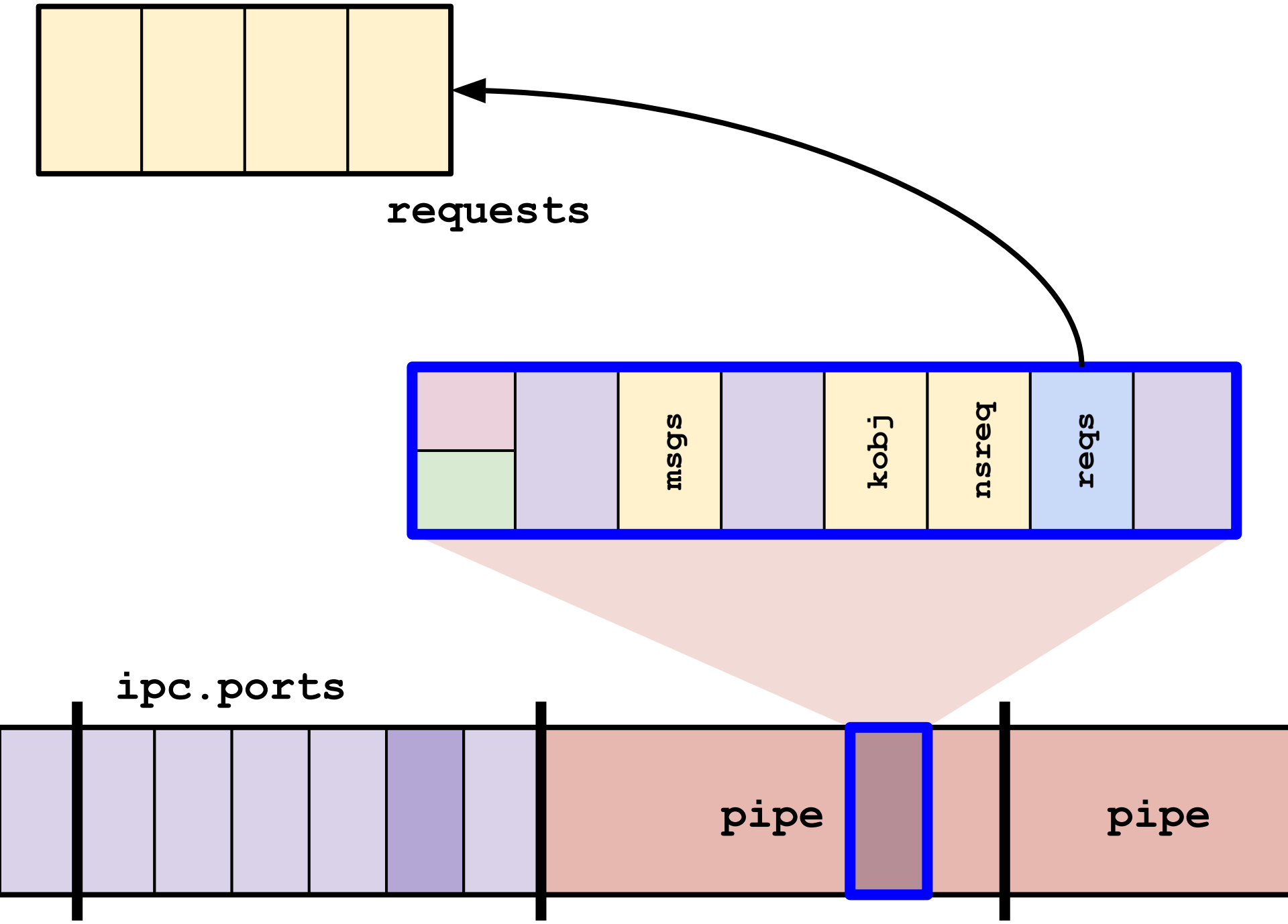

kmsg
header
bsd_
info
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

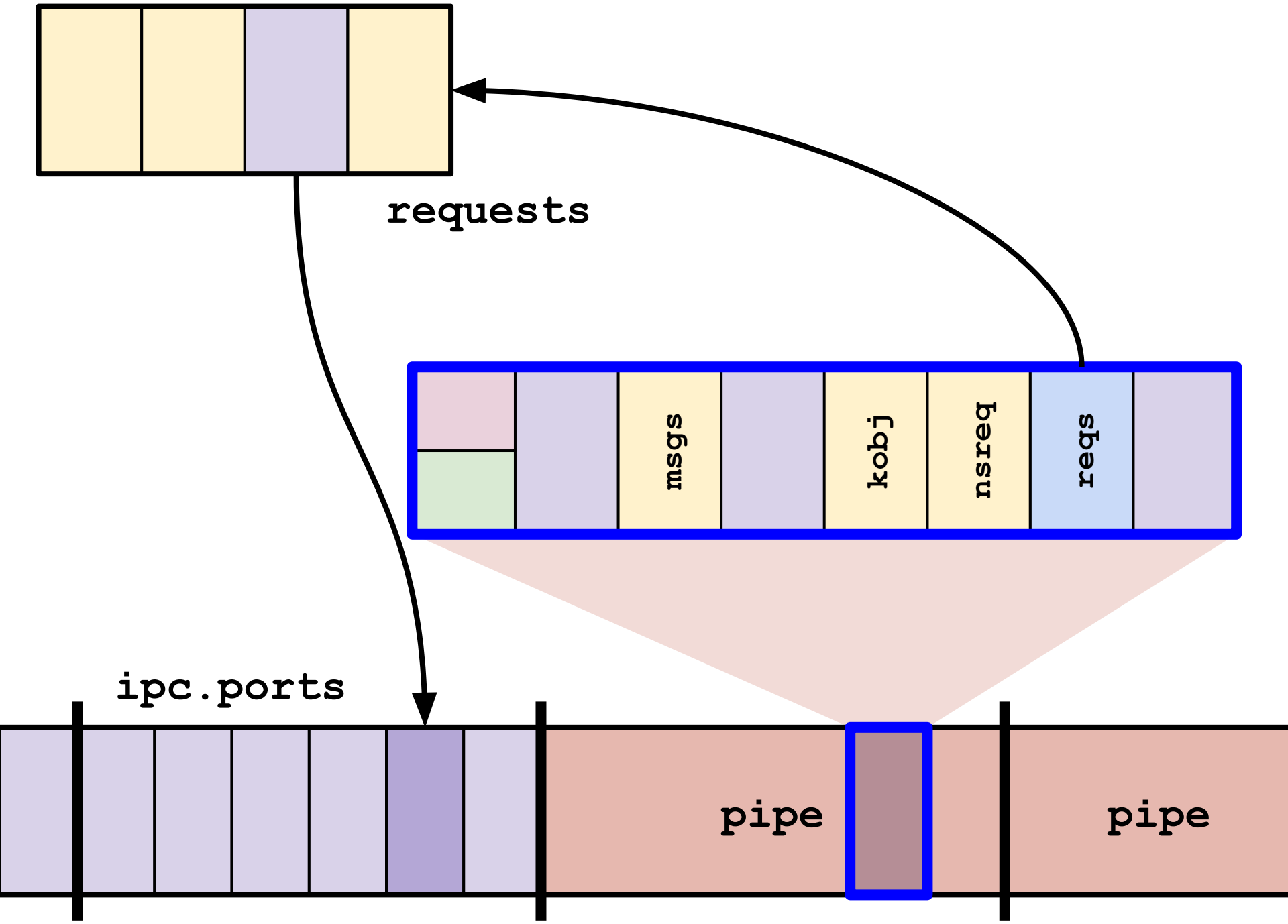
We want to build a read/write primitive using kernel_task

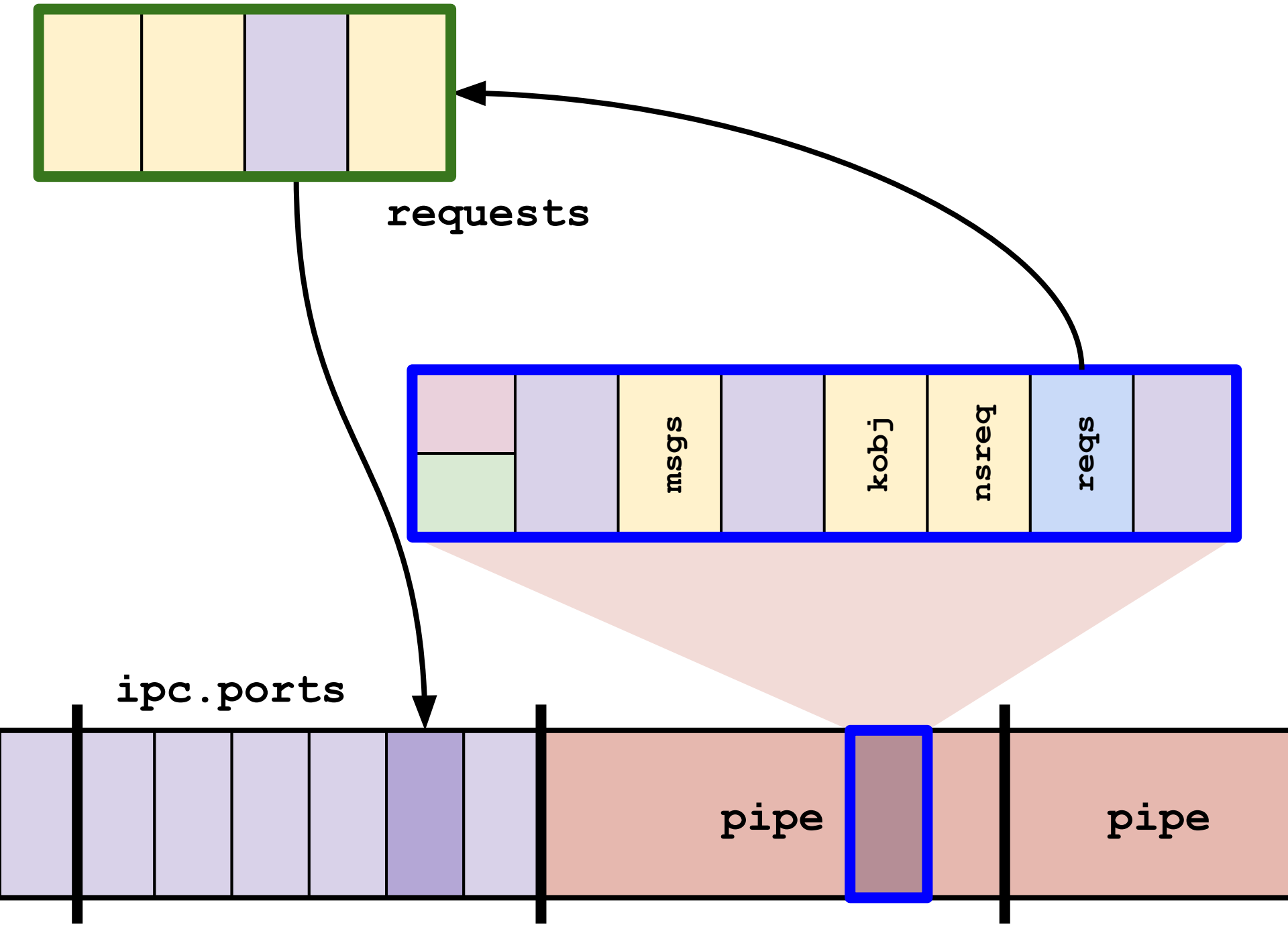
Need the address of our fake port

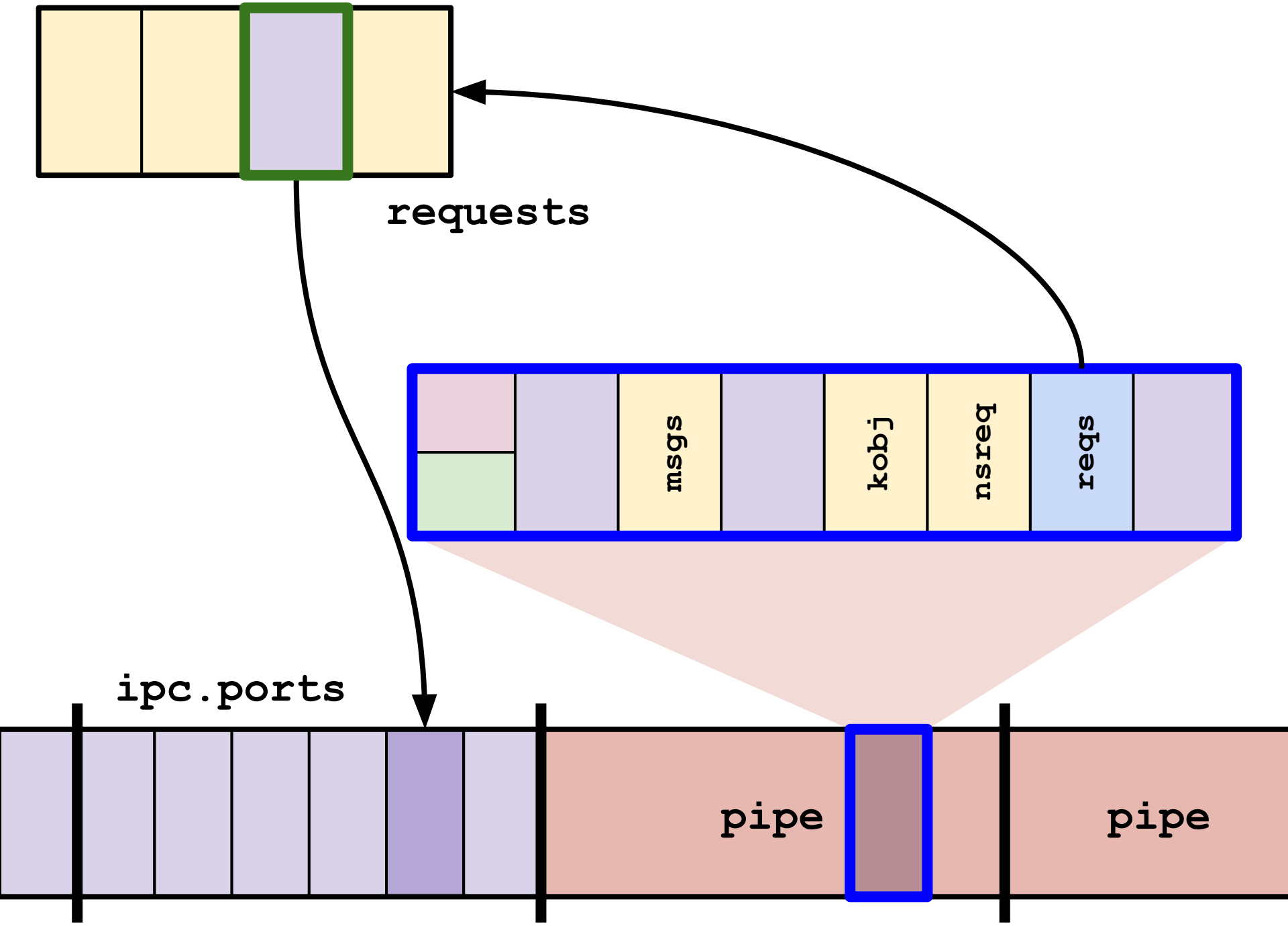

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

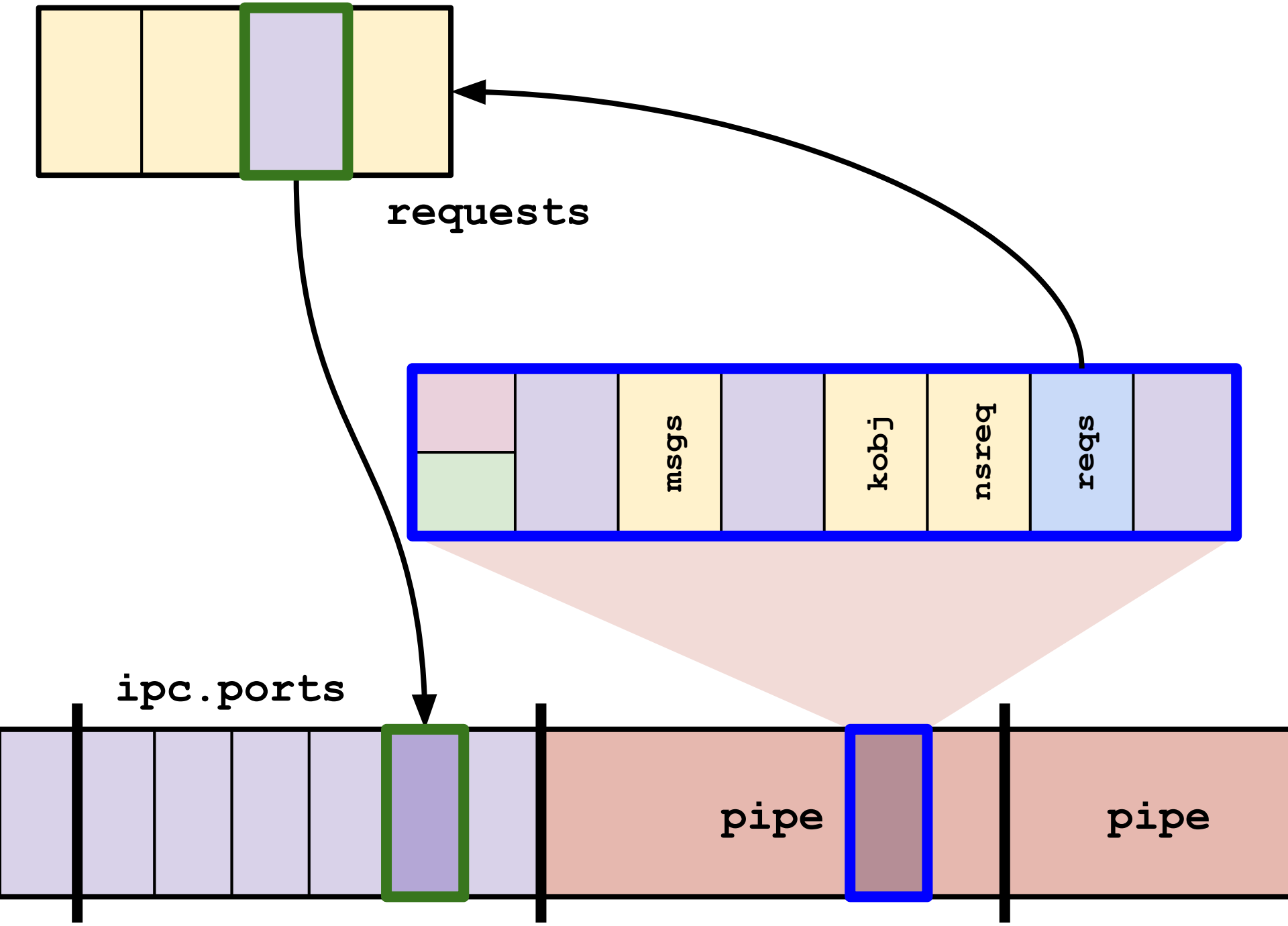

msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

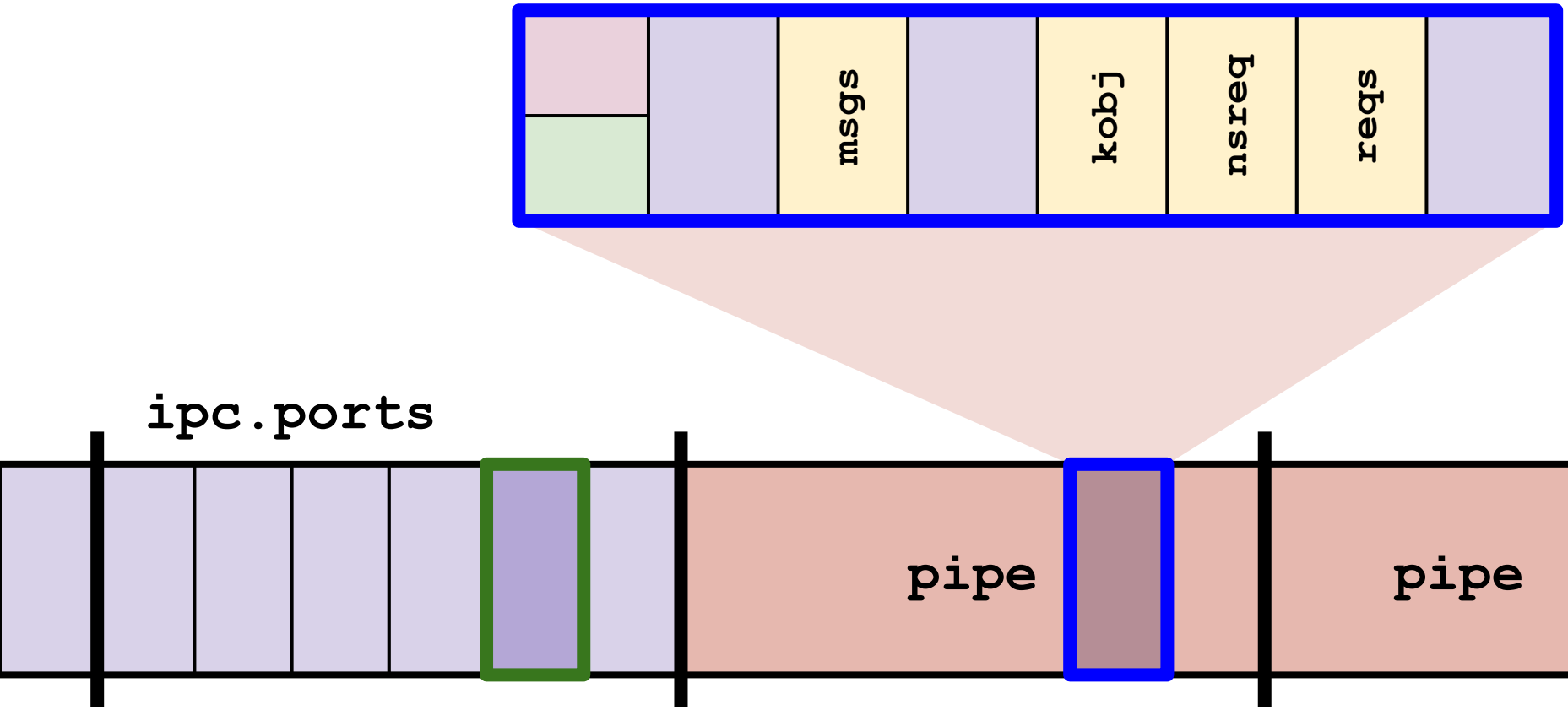

requests
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

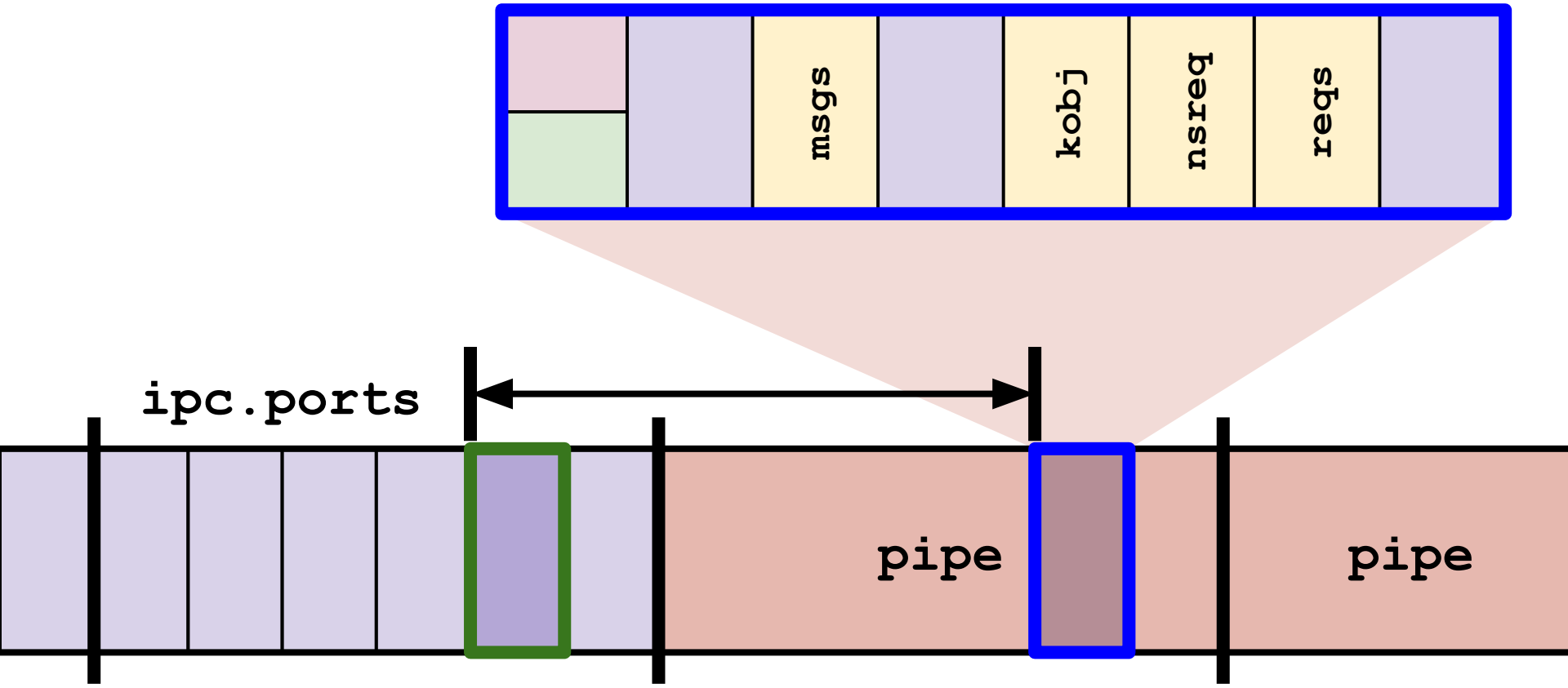

requests
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe


requests
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

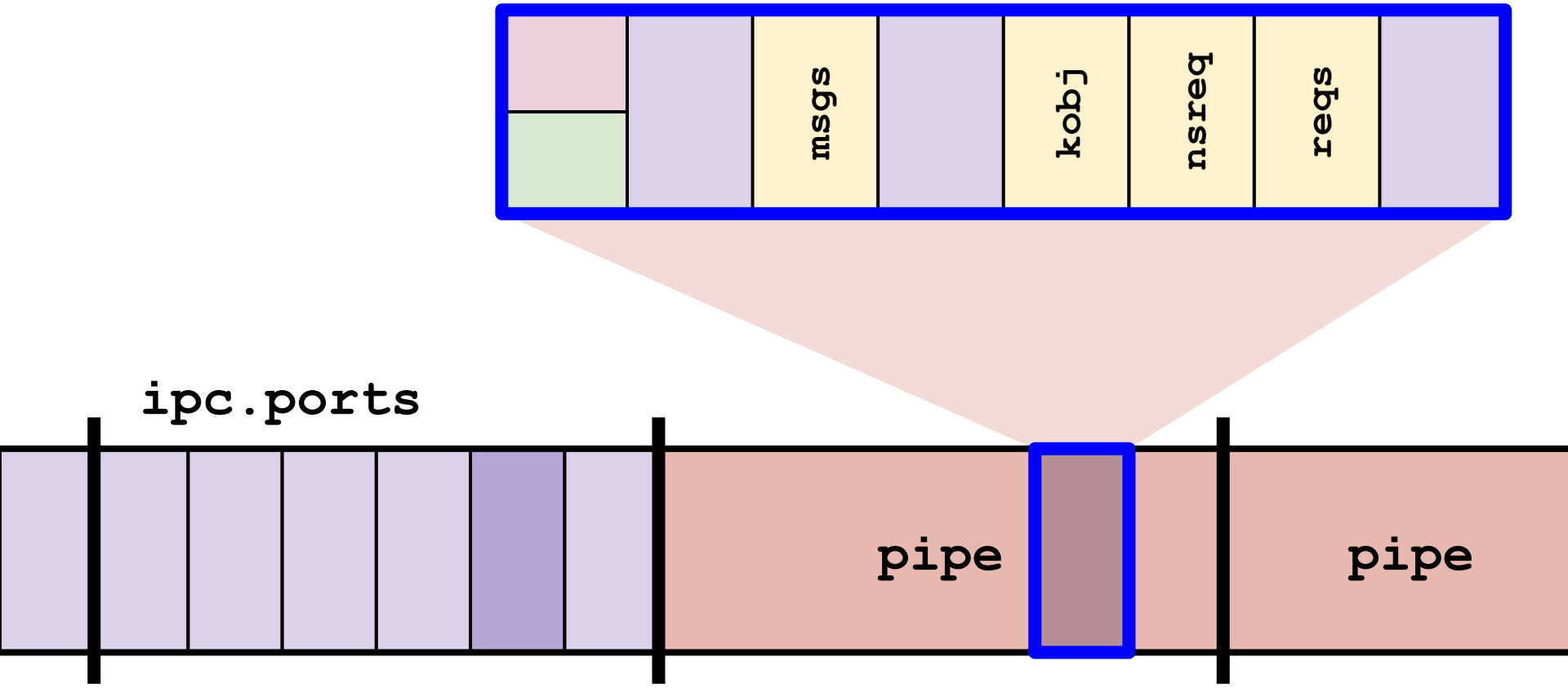

requests
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

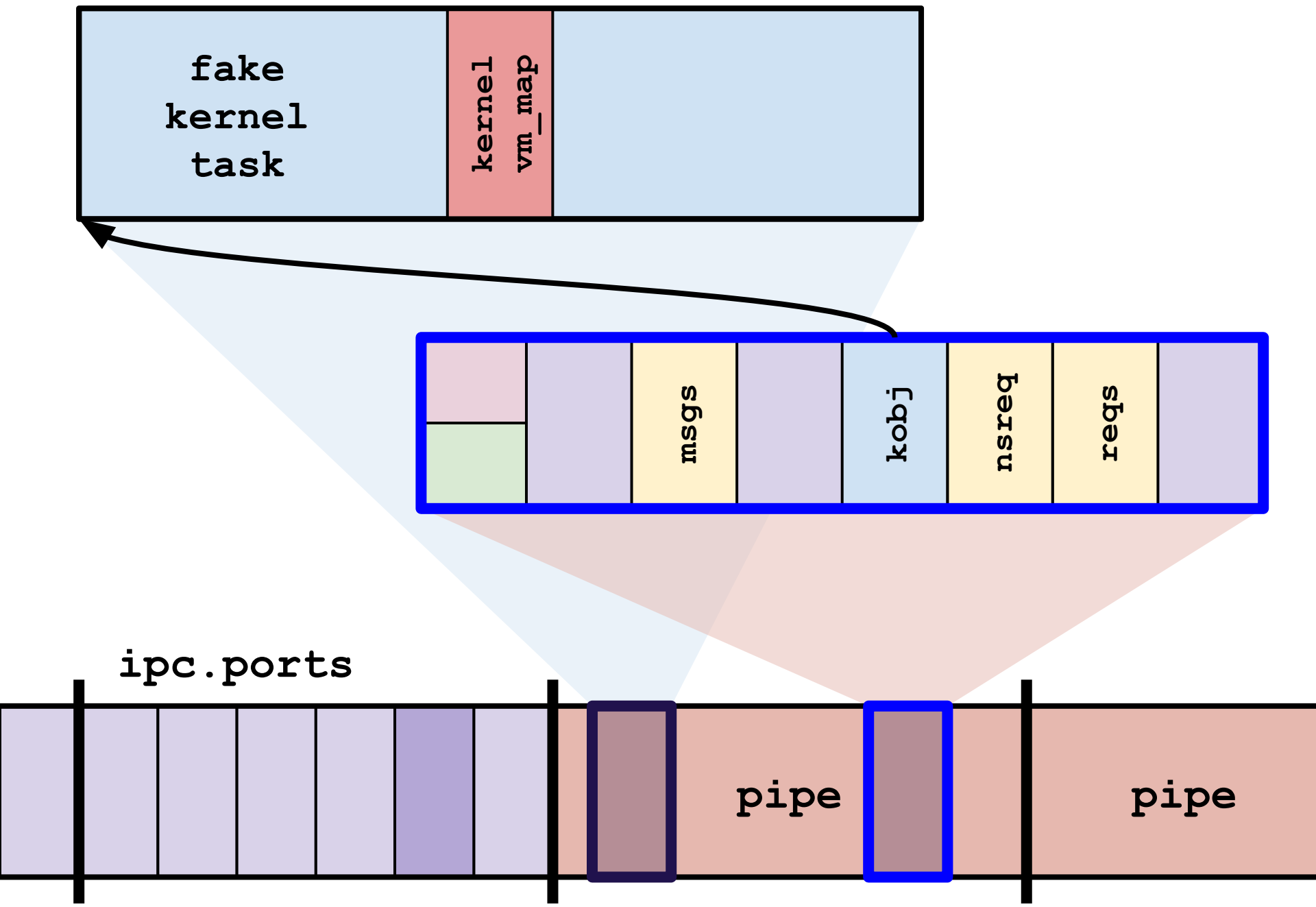

requests
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe


msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe


msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

Construct a fake kernel_task for arbitrary read/write


msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe


fake
kernel
task
kernel
vm_map
msgs
kobj
nsreq
reqs
ipc.ports
pipe
pipe

DEMO

Conclusion

MIG bugs still exist

Highly reliable, direct-to-kernel exploits are still possible

Mitigations make things harder, but old techniques still (mostly) work

https://bugs.chromium.org/p/project-zero/issues/detail?id=1731