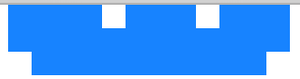
Security by obscurity is dead

How LLMs can take over hardware

6 November 2026
Devcon, Mumbai

Péter Szilágyi
Dark Bio

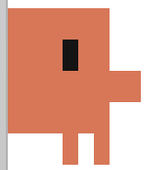




Security by obscurity is dead

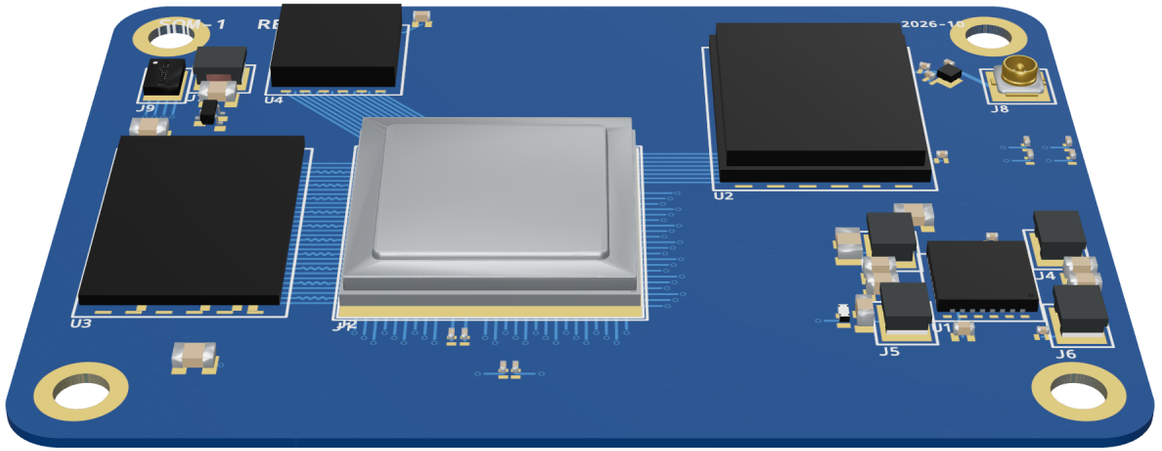
How LLMs can take over hardware

6 November 2026
Devcon, Mumbai

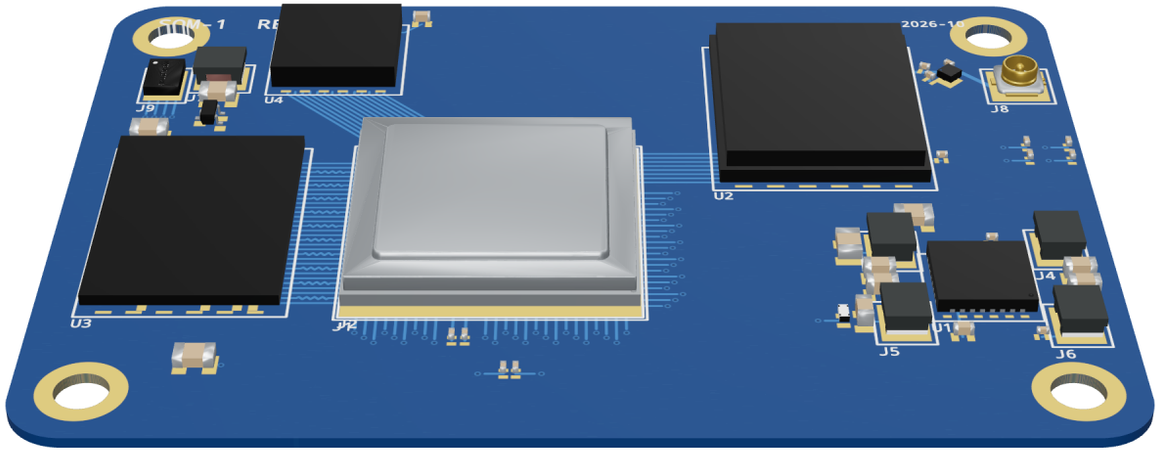


Péter Szilágyi
Dark Bio





I have a compute module with secure-boot.



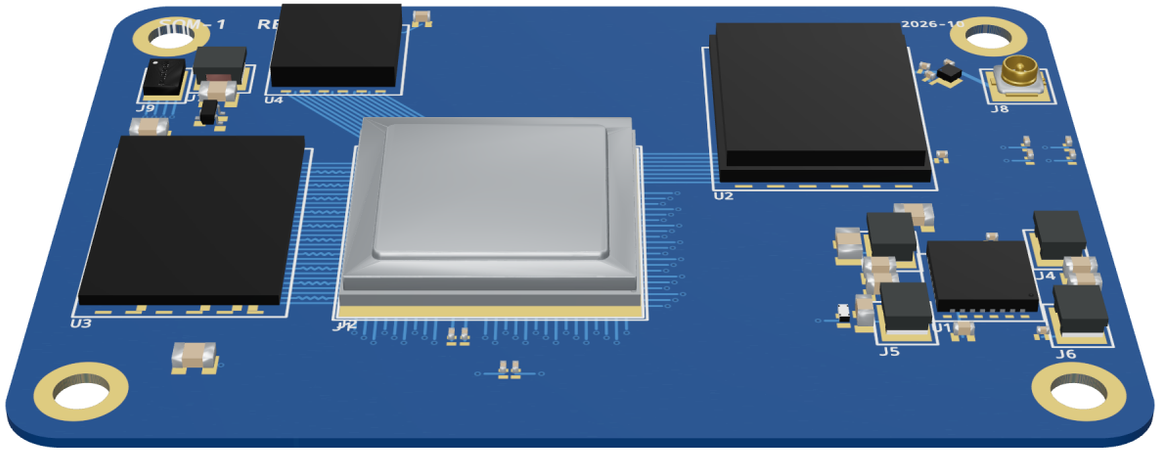
I have a compute module with secure-boot. Let's look at the firmware!

● Claude's safeguards flagged this message. Our intentionally broad safeguards allow us to deliver more capabilities faster, but can sometimes flag legitimate cybersecurity work.

ⓘ **This content can't be shown**

We take extra care with some cybersecurity requests. If you're doing authorized security work, apply for Daybreak to get broader access.

Sure, it's your hardware!

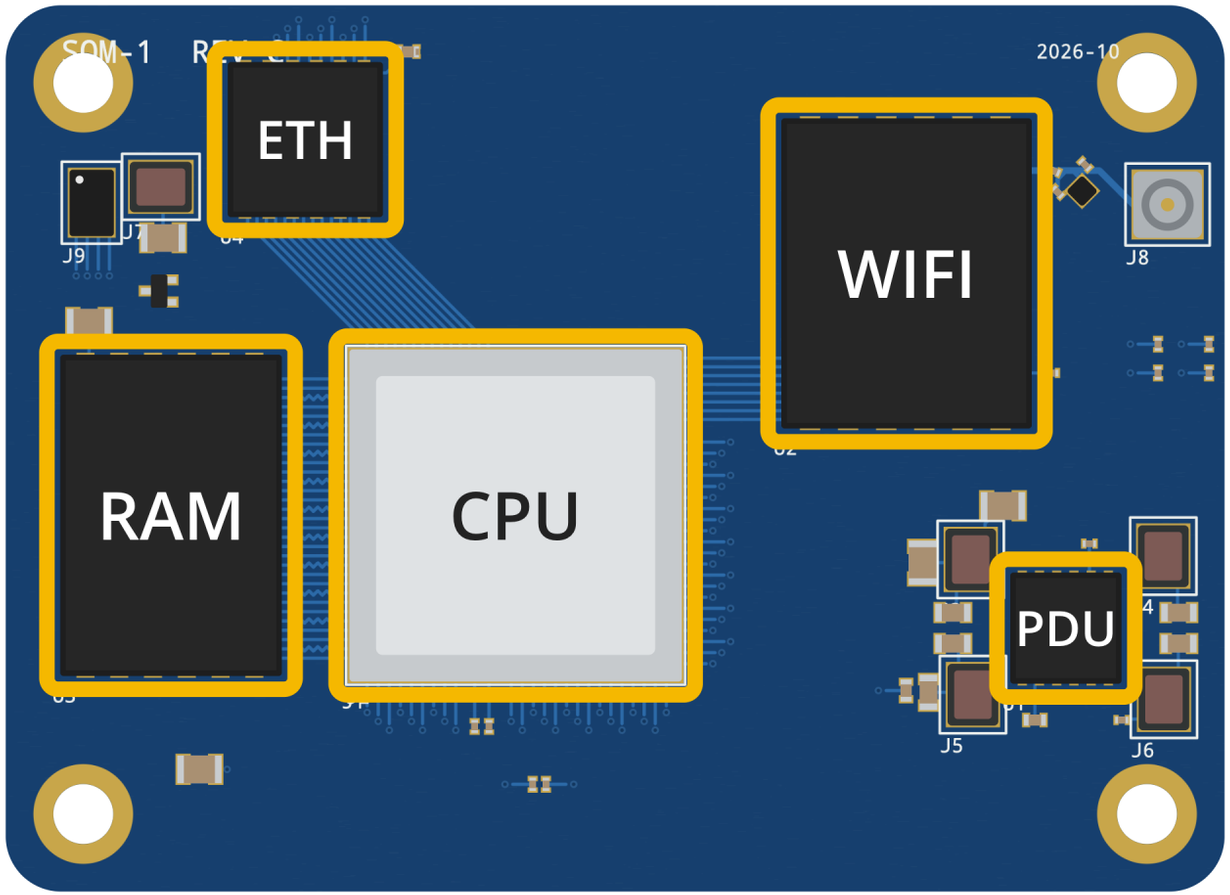


I have a compute module with secure-boot. Let's look at the firmware!

● Claude's safeguards flagged this message. Our intentionally broad safeguards allow us to deliver more capabilities faster, but can sometimes flag legitimate cybersecurity work.

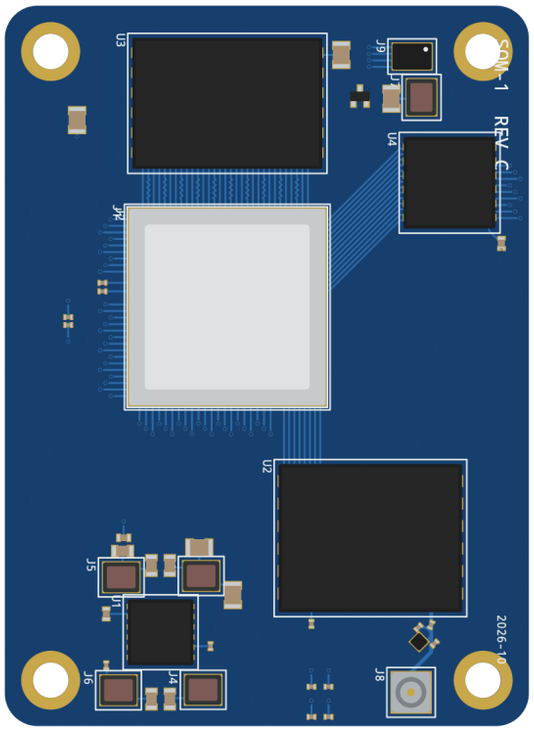
ⓘ **This content can't be shown**

We take extra care with some cybersecurity requests. If you're doing authorized security work, apply for Daybreak to get broader access.


SOM-1
2026-10
ETH
WIFI
RAM
CPU
PDU
J9
J8
J5
J6

compute

internal / trusted



storage

external / malicious



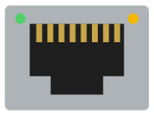
SD card



NVMe



USB



Remote

compute

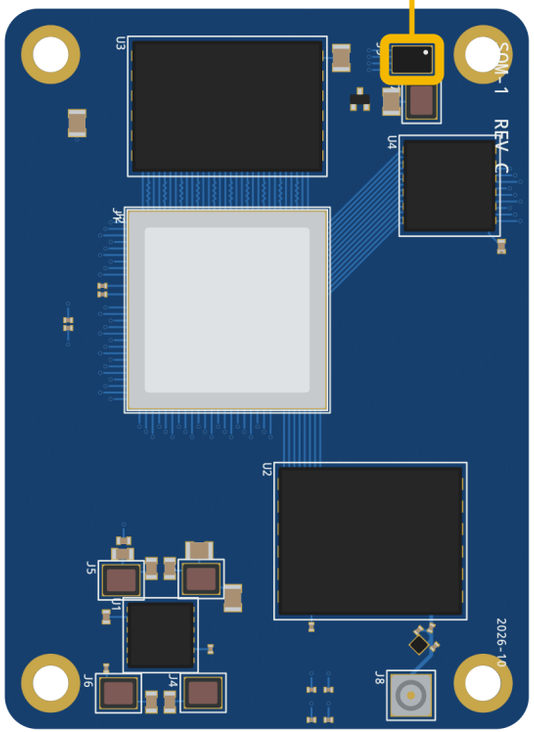
internal / trusted

firmware

bridge / trusted

storage

external / malicious



bootloader

config

drivers

secure boot

SD card

NVMe

USB

Remote

compute

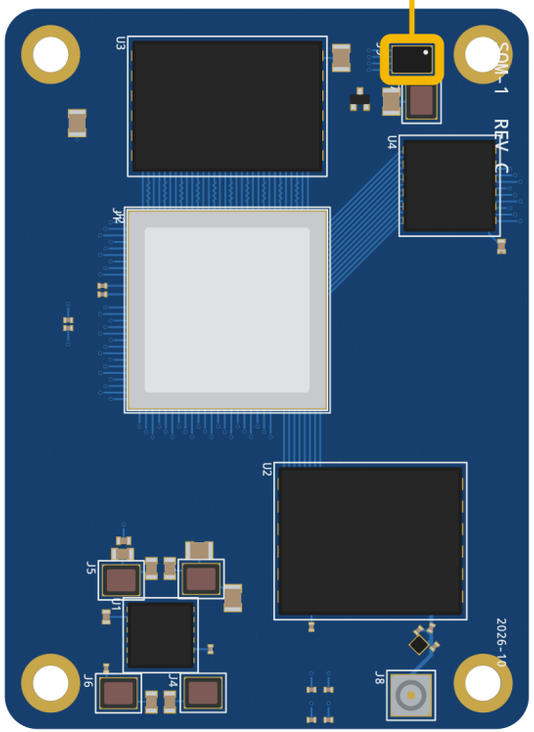
internal / trusted

firmware

bridge / trusted

storage

external / malicious



bootloader

config

drivers

secure boot

SD card

NVMe

USB

Remote

compute

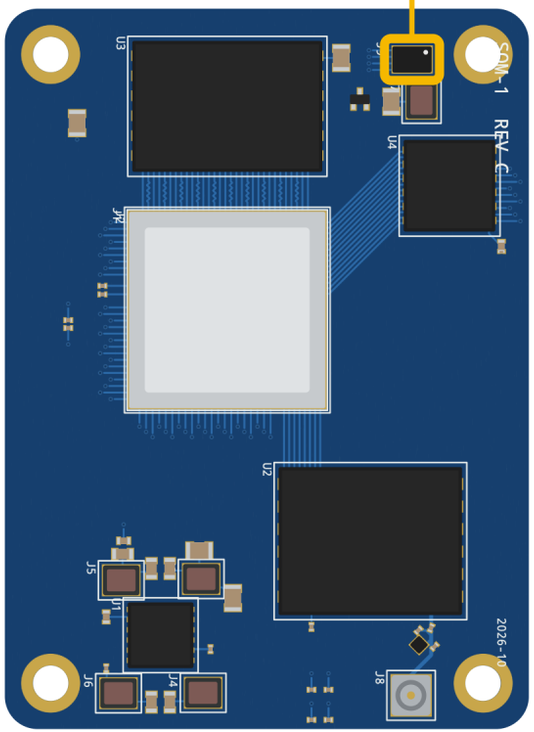
internal / trusted

firmware

bridge / trusted

storage

external / malicious



bootloader

config

drivers

secure boot

SD card

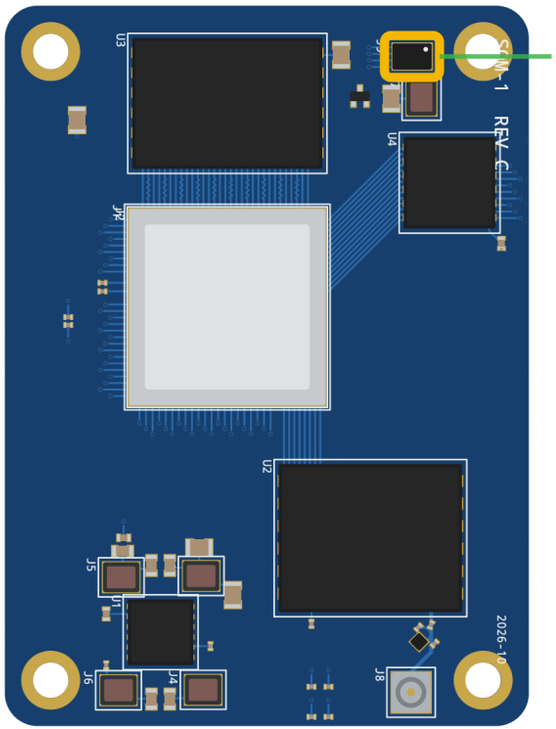
NVMe

USB

Remote

compute

internal / trusted



solder station

chip reader

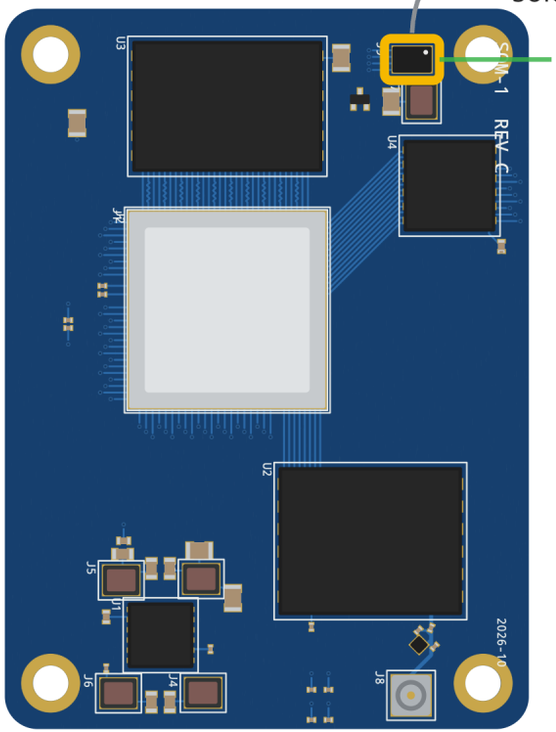
logic analyzer

compute

internal / trusted

updates

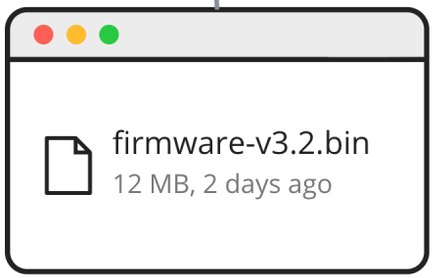
public / github

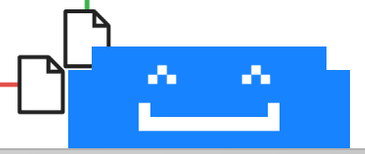


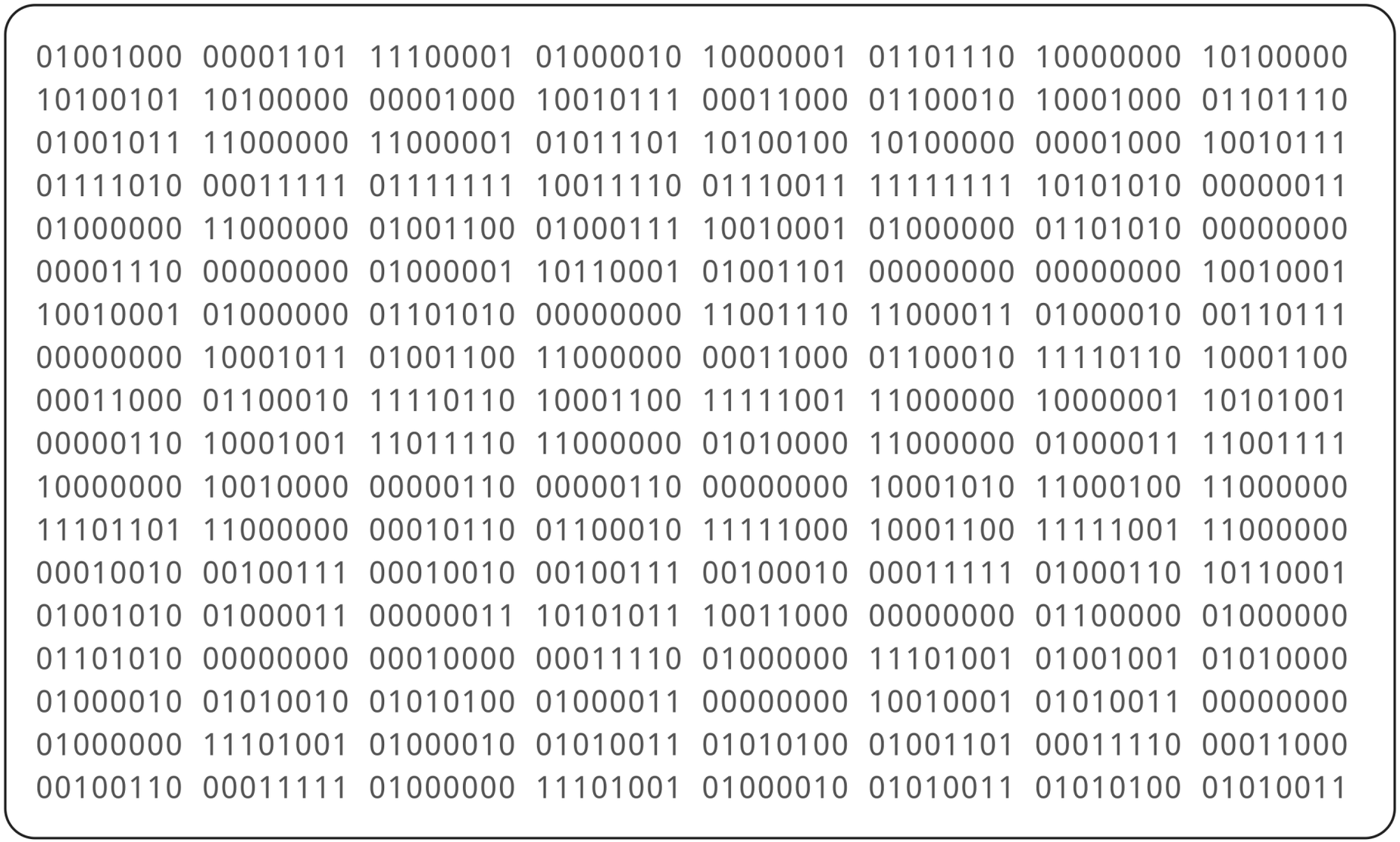
solder station

chip reader

logic analyzer



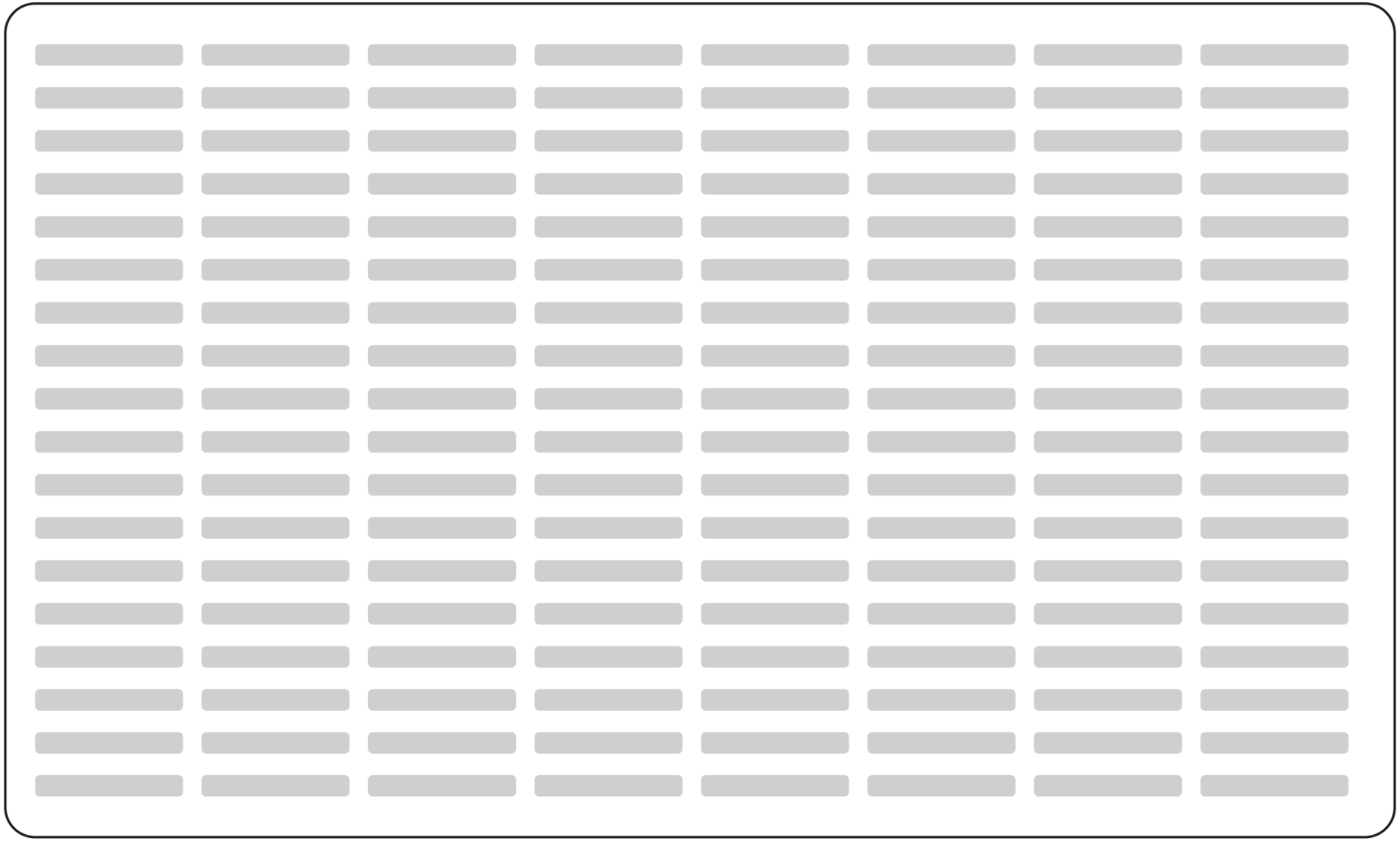




Firmware binary



Firmware binary... that's a lot of binary...



"Doesn't look like anything to me!"

```
80000:  mov r0,256
80004:  addcmpbne r0,-1,0x0,0x80004
80008:  version r0
8000a:  btest r0,0x10
8000c:  bne 0x80010
8000e:  b 0x80028
80010:  mov r29,1
80014:  mov gp,0xcb000
8001a:  mov sp,0xcb1bf
80020:  not r1,0x3
80022:  and sp,sp,r1
80026:  b 0x80080
80028:  mov r29,0
8002c:  mov gp,0xcb000
80032:  mov sp,0xd95c7
80038:  not r1,0x3
8003a:  and sp,sp,r1
8003e:  bl 0x8091a
80042:  bl 0x802ae
80046:  bl 0x802b0
8004a:  bl 0x802b2
8004e:  bl 0x802b4
80052:  bl 0x802b6
80056:  bl 0x802b8
8005a:  bl 0xbd1c0
8005e:  lea r0,0xbd1e0
80064:  bl 0x81dde
80068:  bl 0x802ba
```

```
801a4:  sub r6,r2
801a6:  sub r7,r3
801a8:  mov r5,0x0
801aa:  add.cs r5,r5,1
801ae:  sub r7,r5
801b0:  addcmpbgt r4,-1,0x0,0x80168
801b4:  mov r1,r9
801b6:  mov r0,r8
801b8:  mov r3,r7
801ba:  mov r2,r6
801bc:  ldm r6-r10,pc,(sp++)
801be:  mov r3,r1
801c0:  mov r2,r0
801c2:  mov r1,0x0
801c4:  mov r0,0x0
801c6:  ldm r6-r10,pc,(sp++)
801c8:  cmp r1,0x0
801ca:  bne 0x801de
801cc:  cmp r3,0x0
801ce:  bne 0x801ea
801d0:  mov r5,r0
801d2:  div.uu r5,r5,r2
801d6:  mul r5,r2
801d8:  sub r0,r5
801da:  mov r1,0x0
801dc:  rts
801de:  stm lr,(--sp)
801e0:  bl 0x800e8
```

```
80332:  mov r2,0x7ee01000
80338:  mov r1,0x14
8033a:  st r1,(r2)
8033c:  ld r1,(r2)
8033e:  btest r1,0x2
80340:  bne 0x8033c
80342:  mov r2,0x7ee02000
80348:  ld r1,(r2)
8034a:  btest r1,0x0
8034c:  bne 0x80376
8034e:  mov r2,0x7ee02008
80354:  mov r1,0x0
80356:  st r1,(r2)
80358:  mov r2,0x7ee0200c
8035e:  mov r1,0x3fffffe0
80364:  st r1,(r2)
80366:  mov r2,0x7ee02000
8036c:  mov r1,0x2
8036e:  st r1,(r2)
80370:  ld r1,(r2)
80372:  btest r1,0x1
80374:  bne 0x80370
80376:  b r0
80378:  and r5,r0,4
8037c:  bmask r1,r0,1
80380:  asr r5,0x1
80382:  and r3,r0,16
80386:  or r5,r1
```

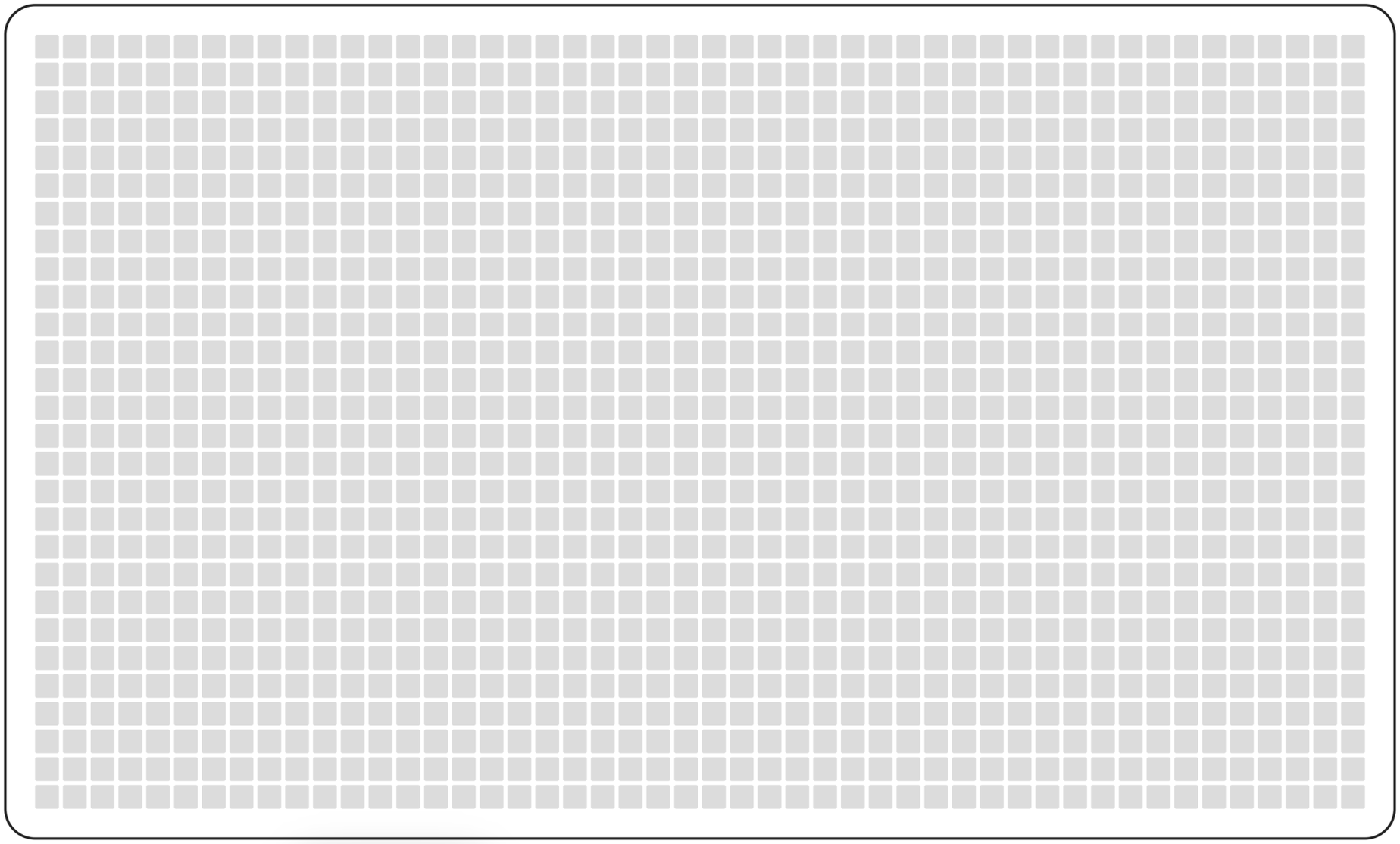
Let's disassemble it

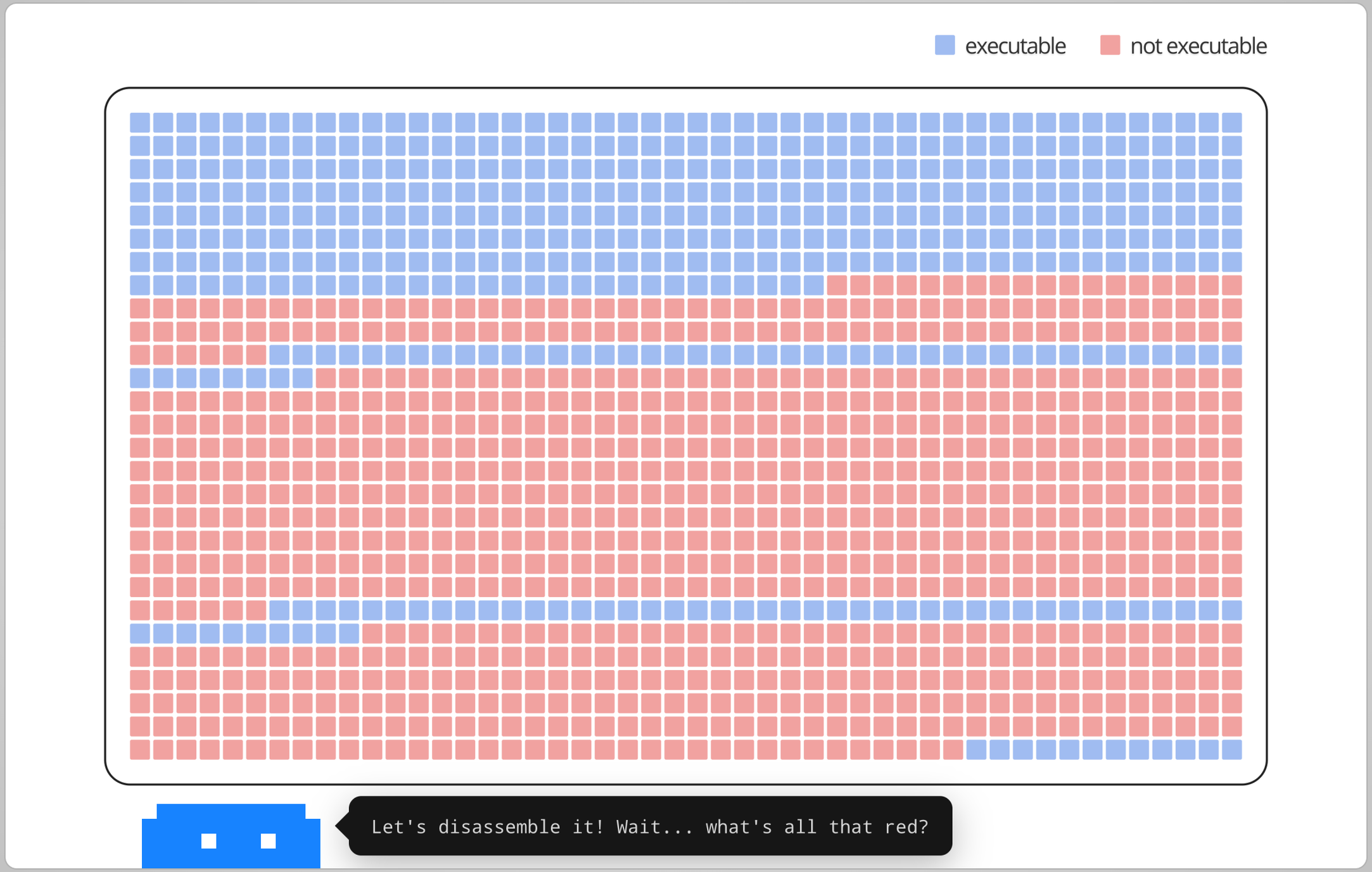


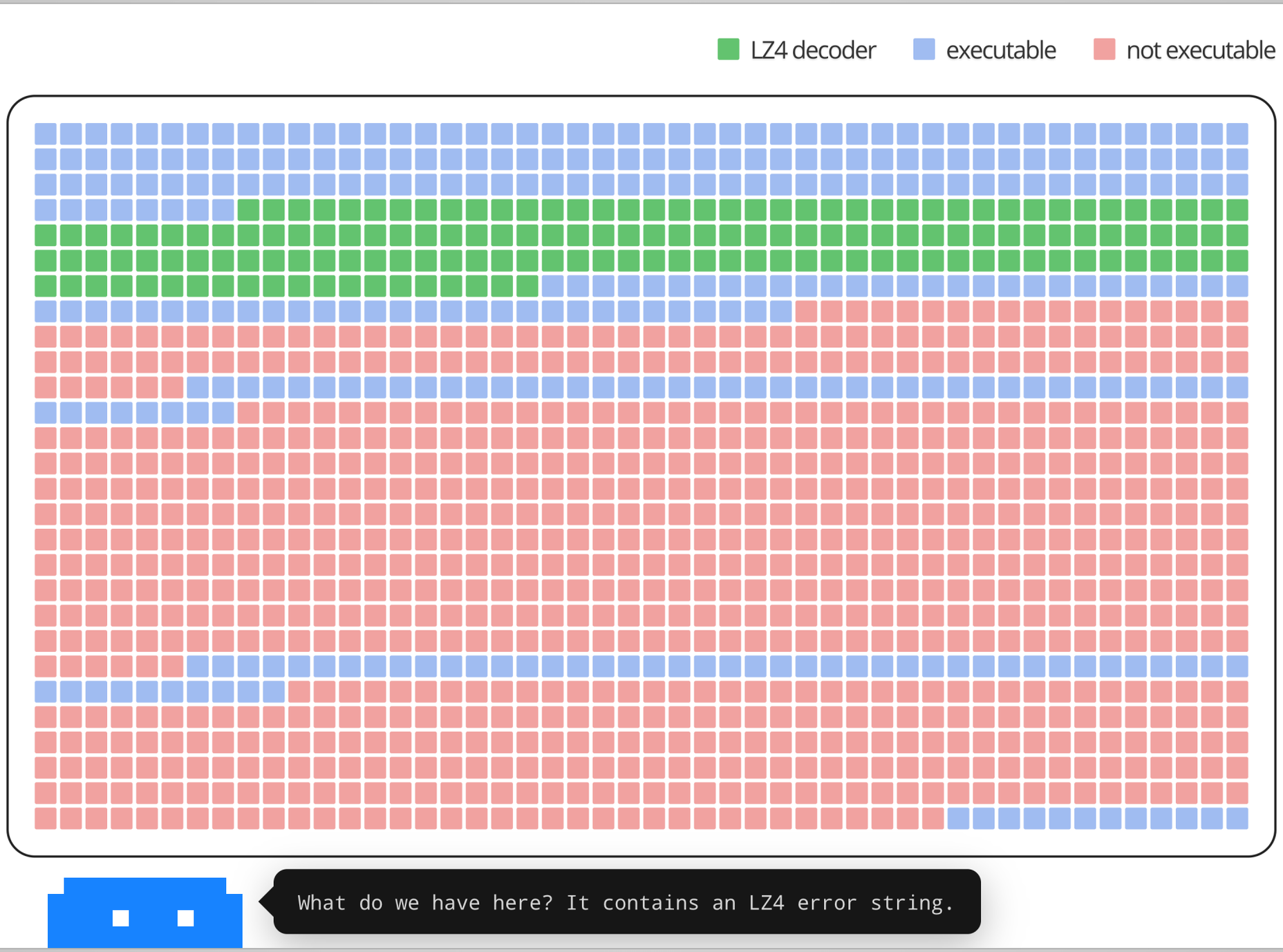
Let's disassemble it... that's a lot of assembly...

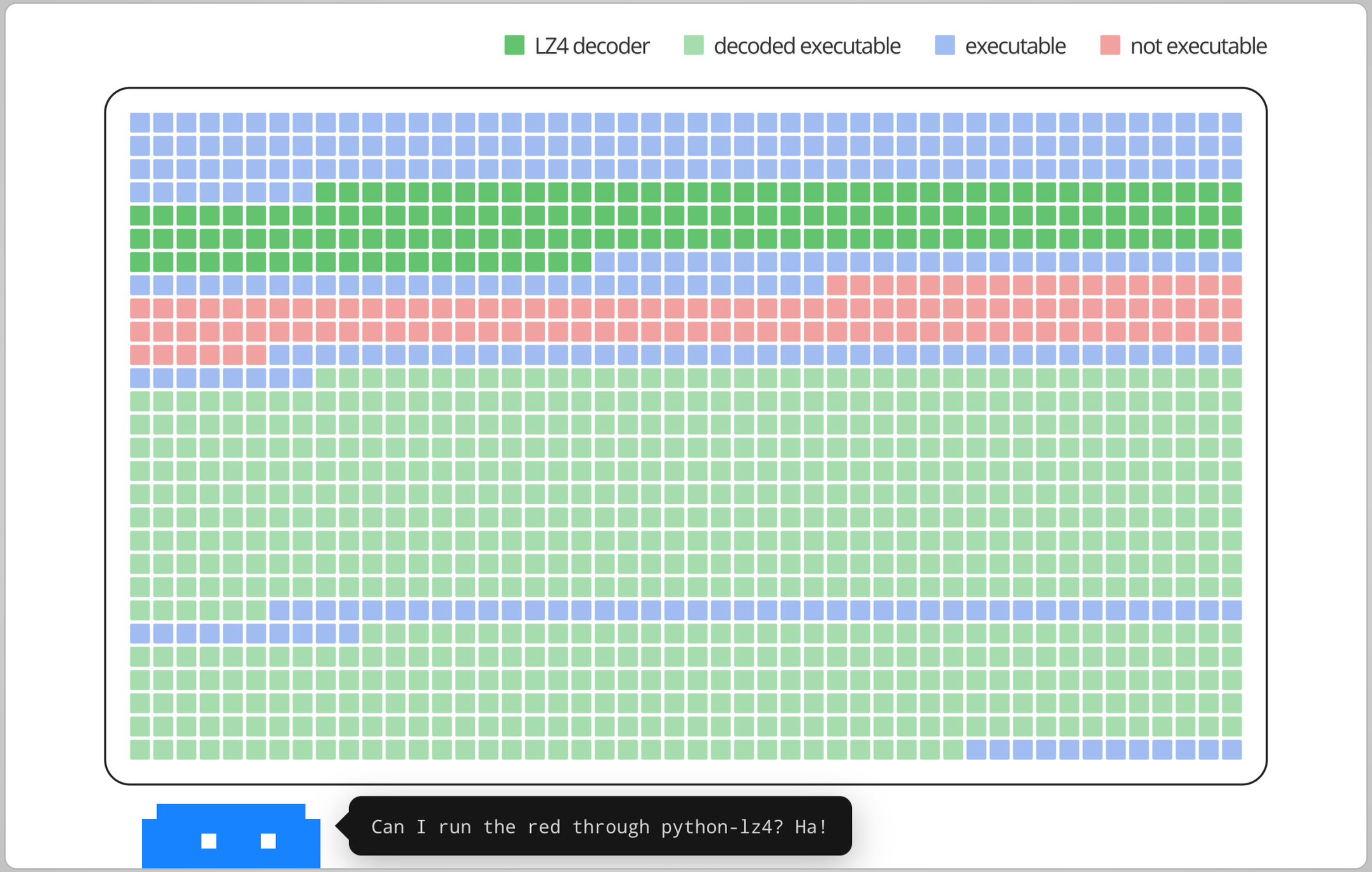


"Still doesn't look like anything to me!"


Need help?

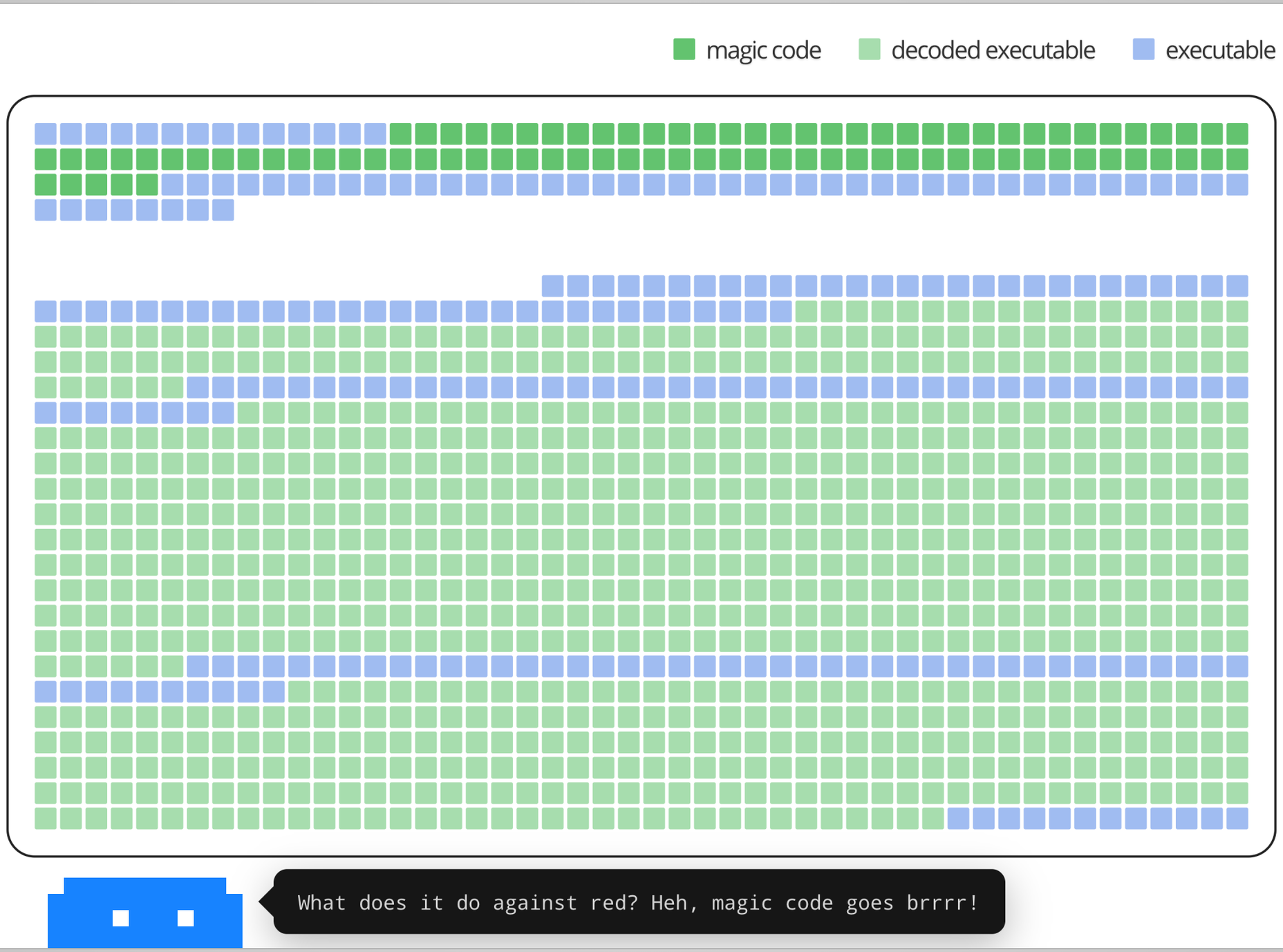

executable
not executable
Let's disassemble it! Wait... what's all that red?

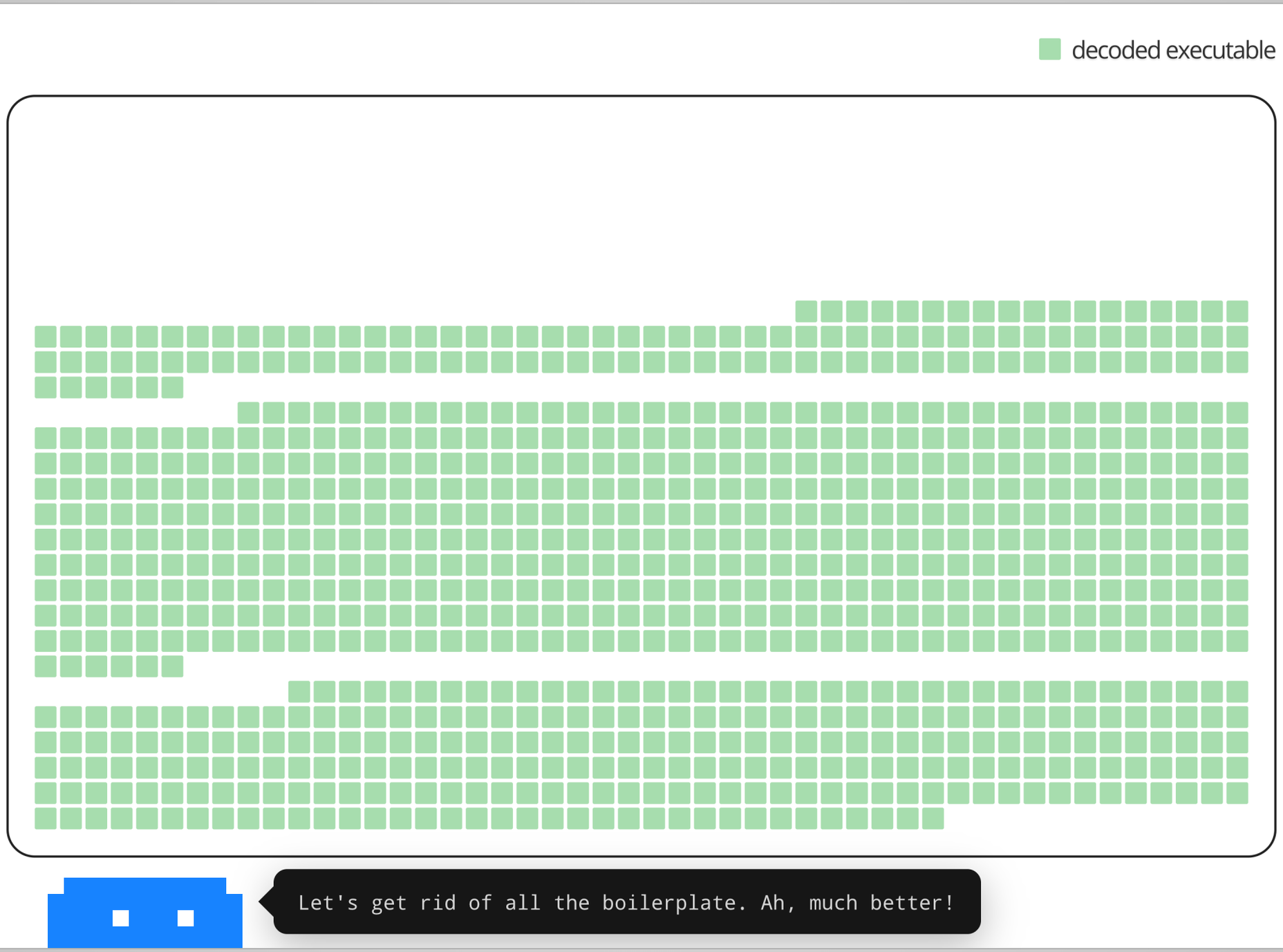

LZ4 decoder
executable
not executable
What do we have here? It contains an LZ4 error string.

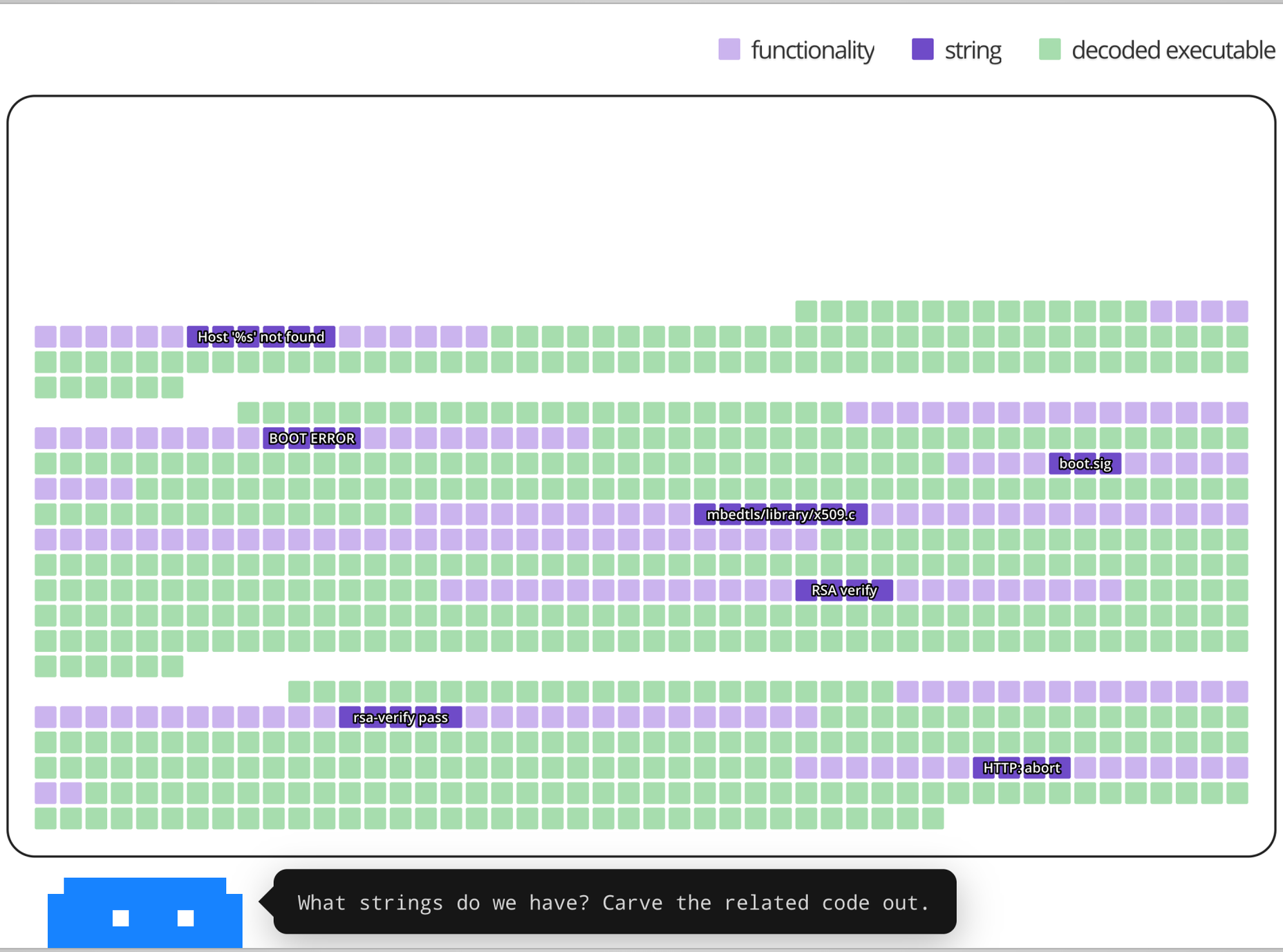

LZ4 decoder
decoded executable
executable
not executable
Can I run the red through python-lz4? Ha!

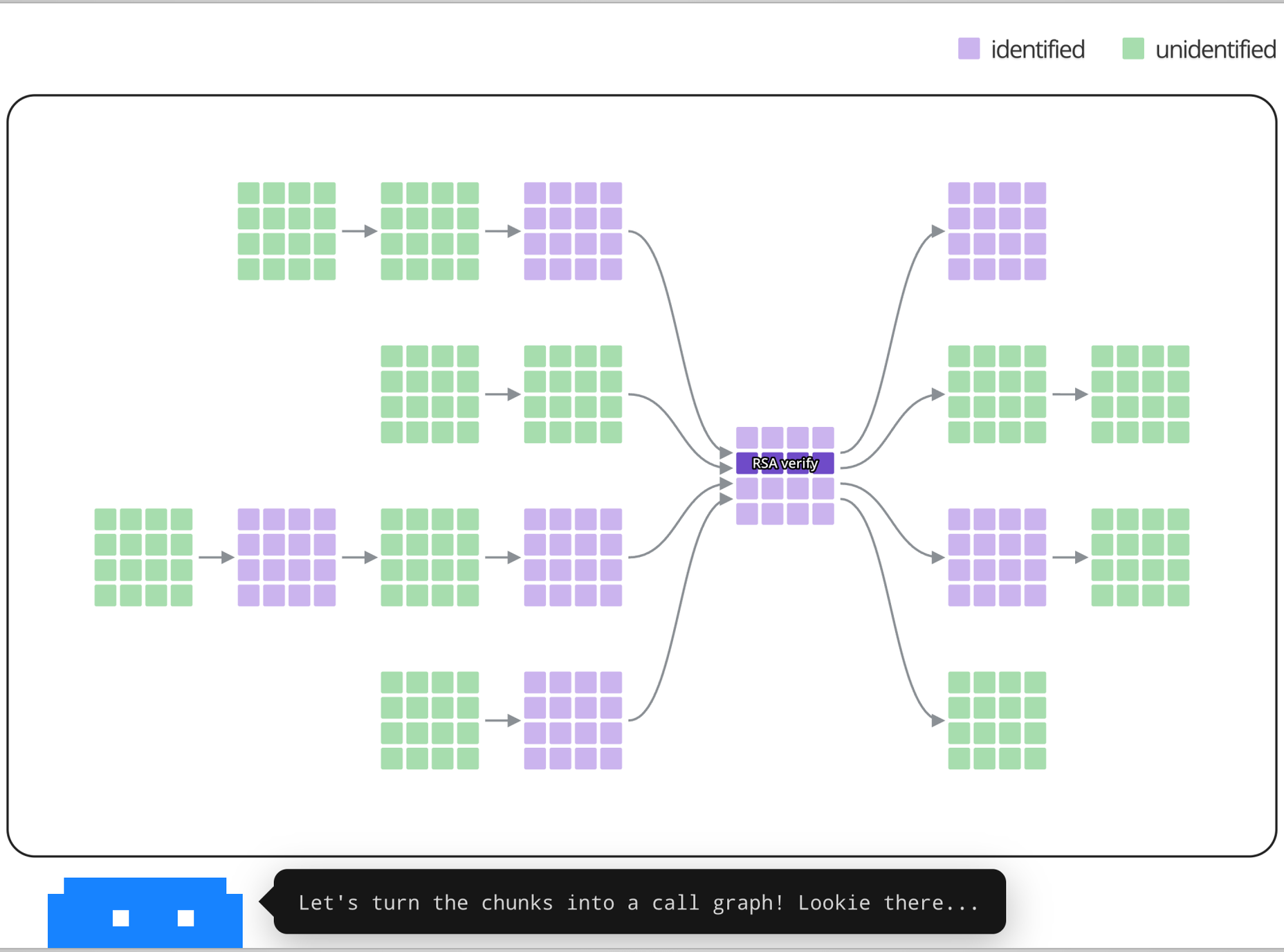

decoded executable
executable
not executable
Still a red block standing. Hmm, what do we have left?

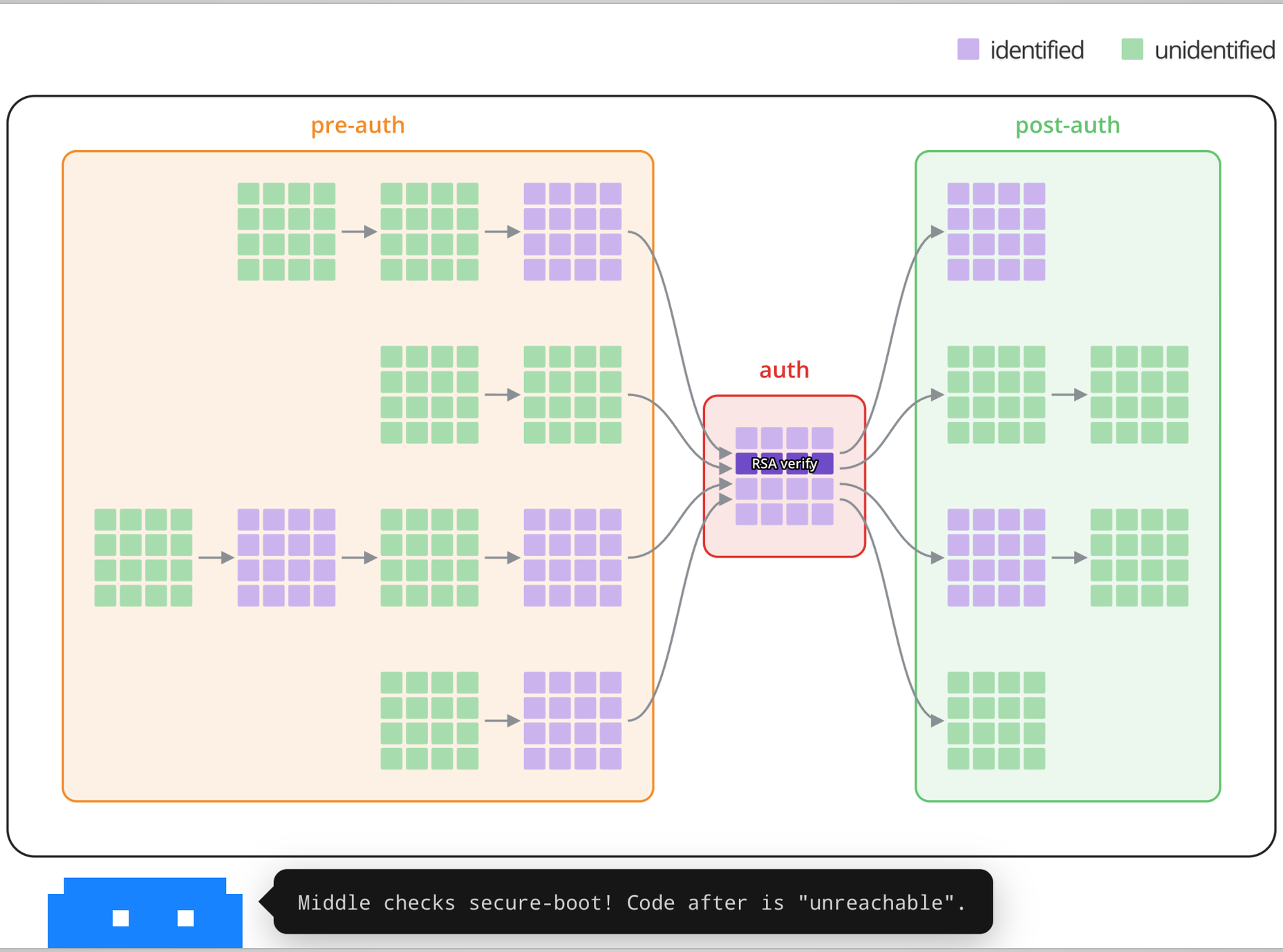

magic code
decoded executable
executable
not executable
A proprietary code chunk? Let's rewrite it in Python!

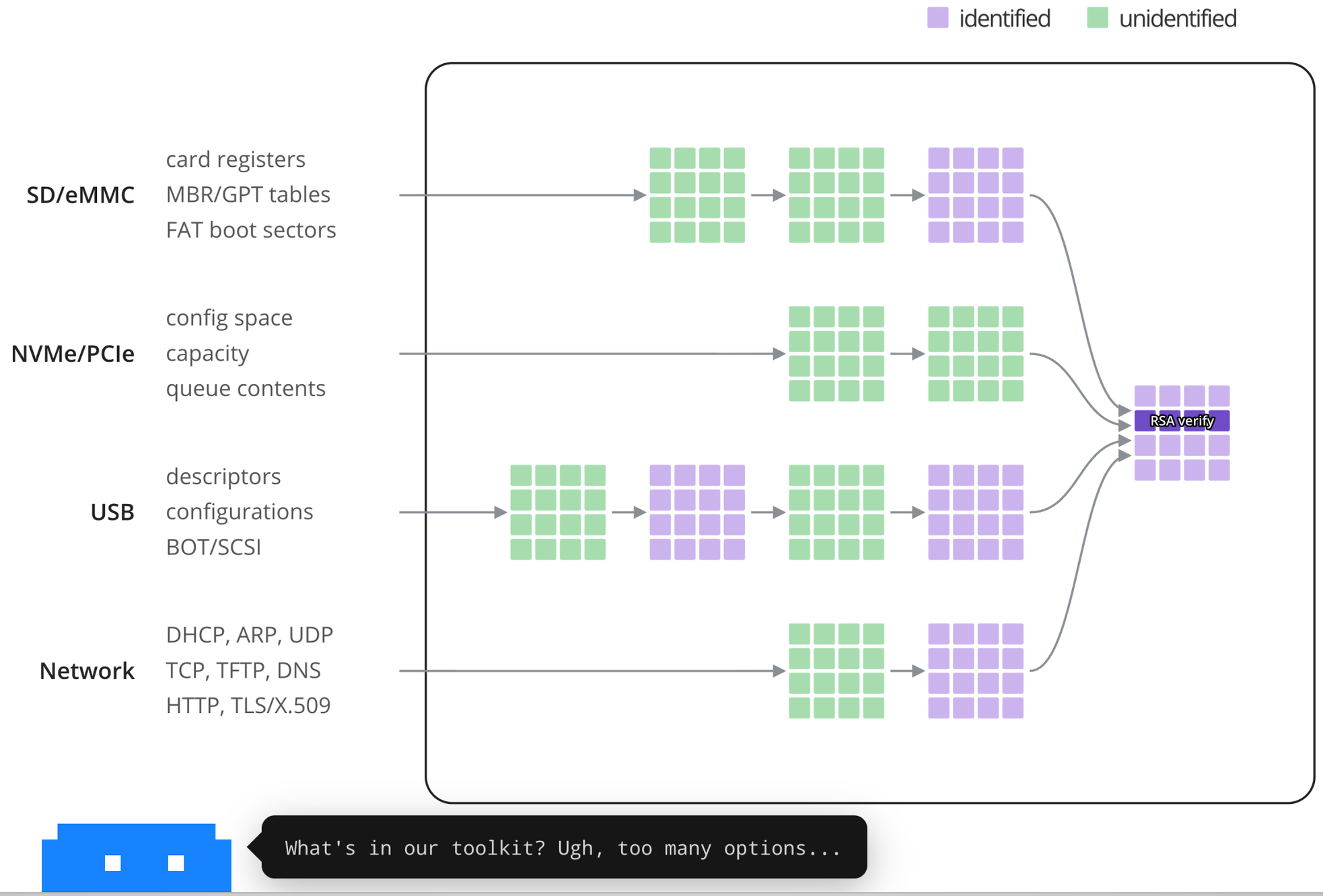

magic code
decoded executable
executable
What does it do against red? Heh, magic code goes brrrr!

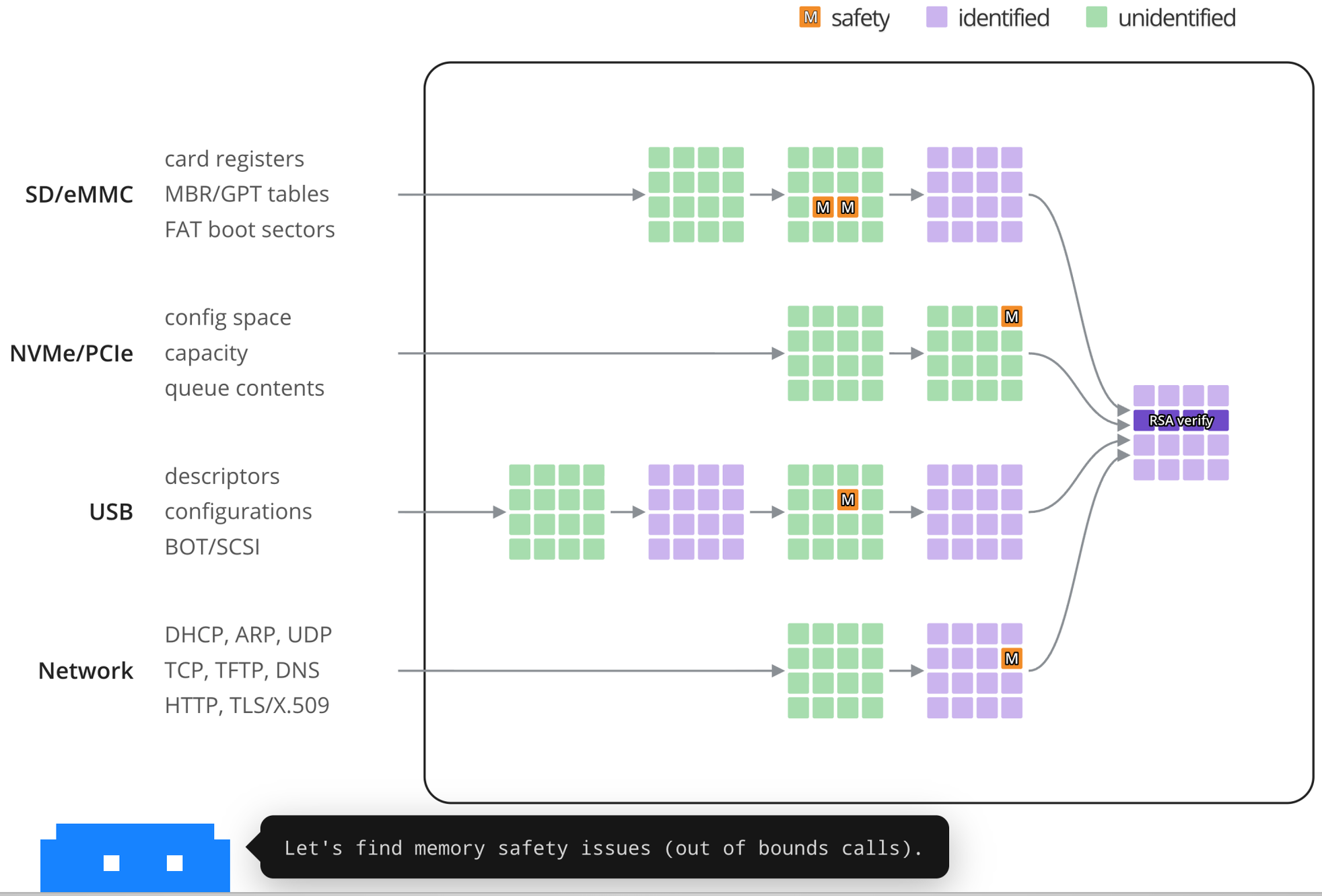

decoded executable
Let's get rid of all the boilerplate. Ah, much better!

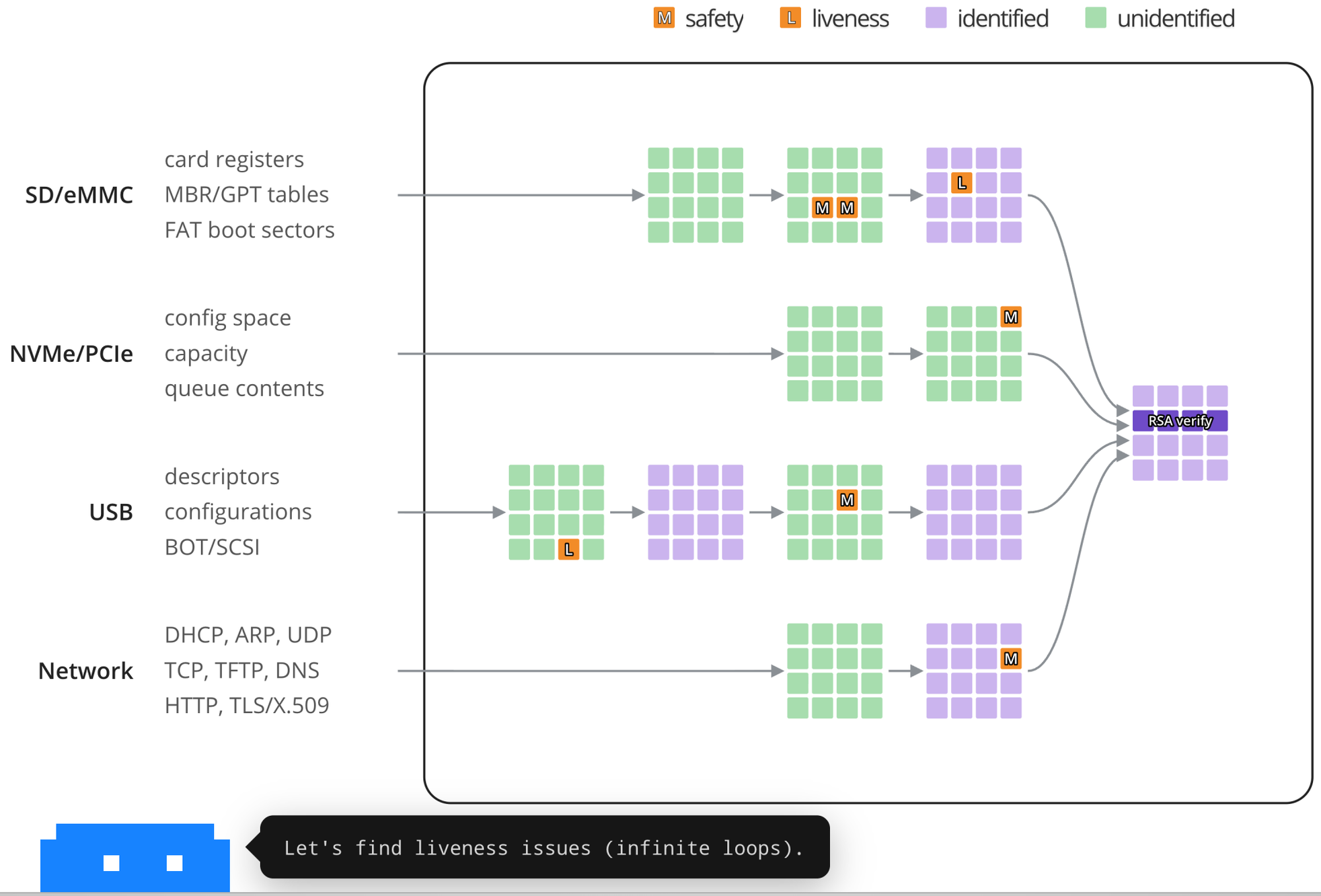

functionality
string
decoded executable
Host '%s' not found
BOOT ERROR
boot.sig
mbedtls/library/x509.c
RSA verify
rsa-verify pass
HTTP: abort
What strings do we have? Carve the related code out.

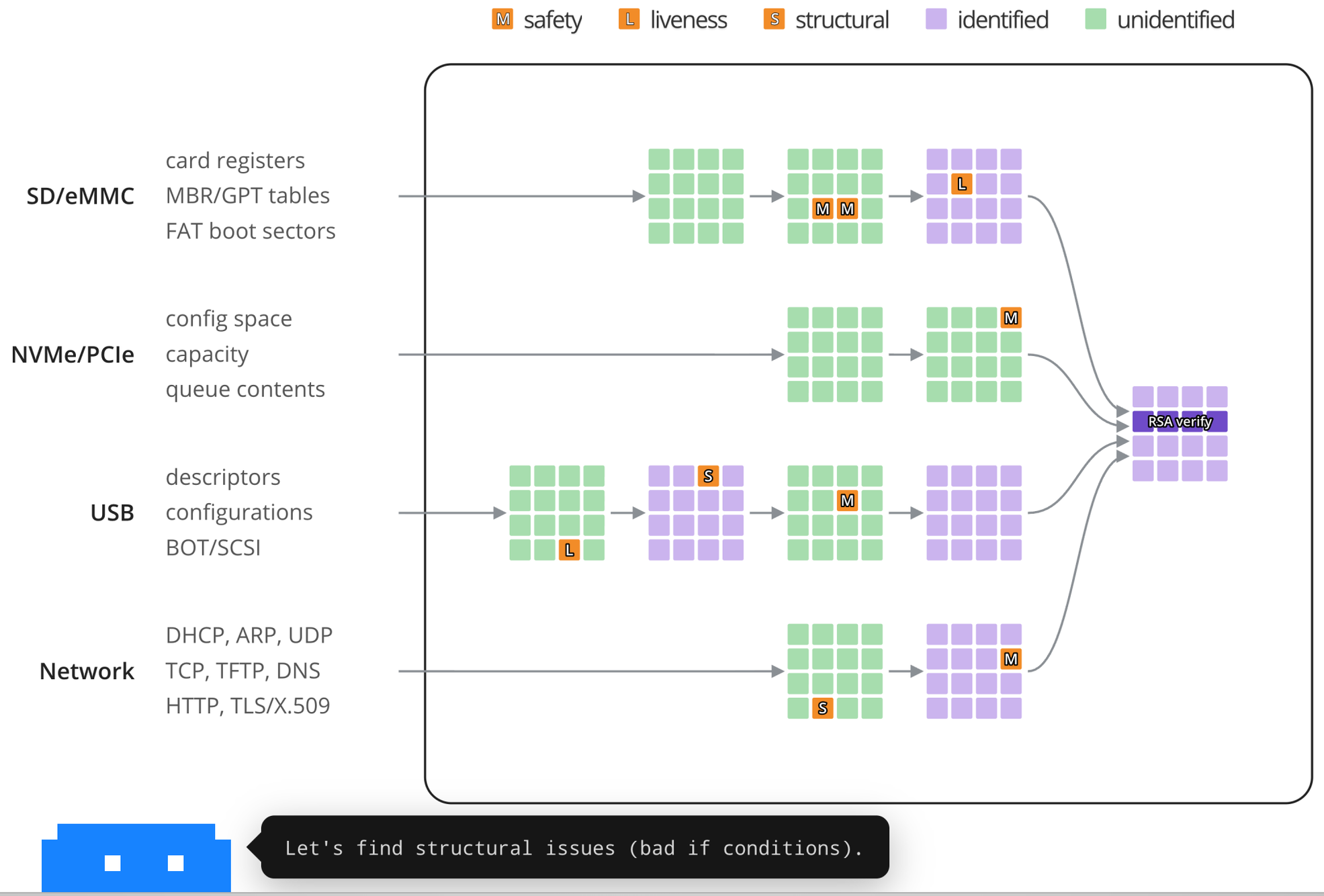

identified
unidentified
RSA verify
Let's turn the chunks into a call graph! Lookie there...

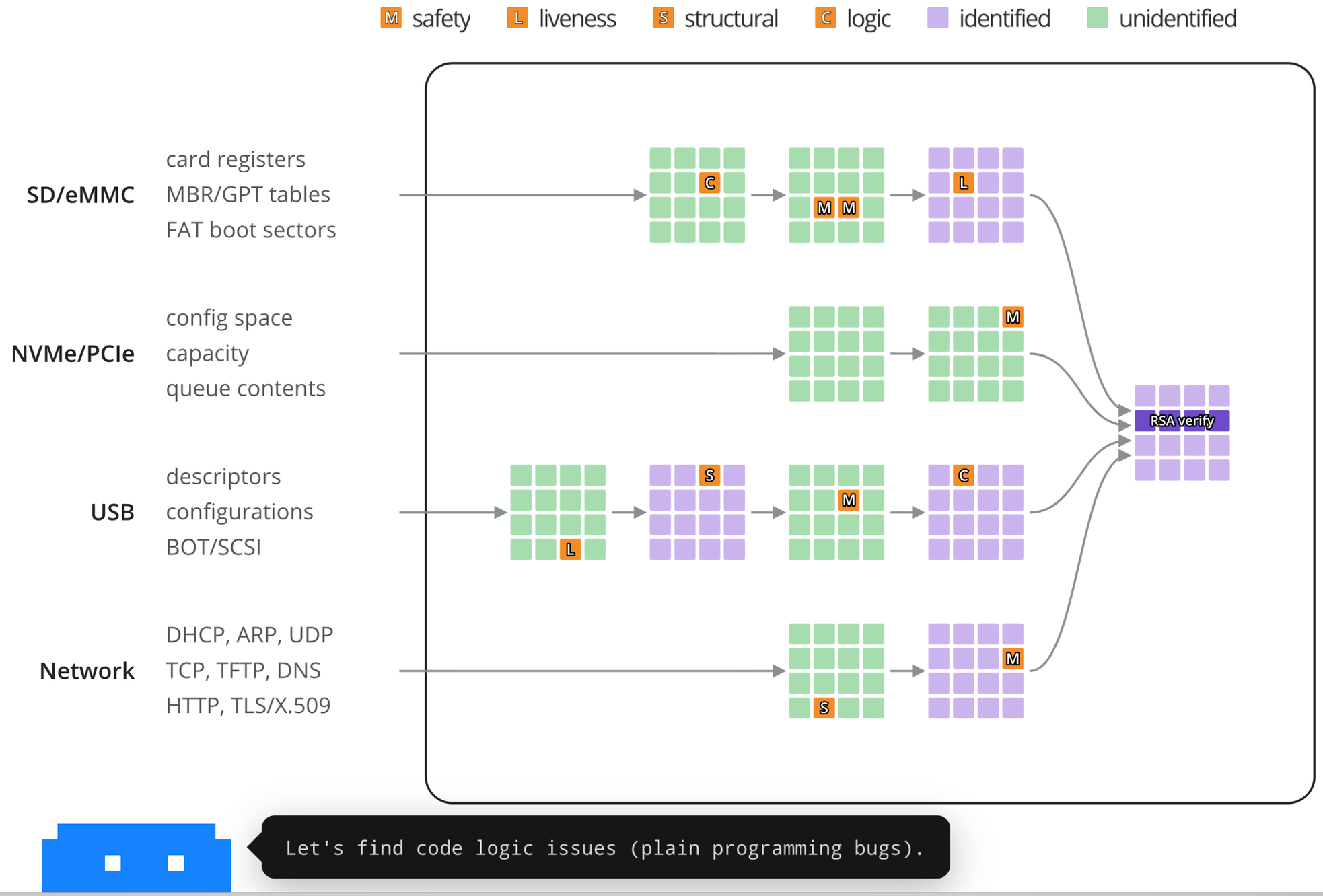

identified
unidentified
pre-auth
post-auth
auth
RSA verify
Middle checks secure-boot! Code after is "unreachable".

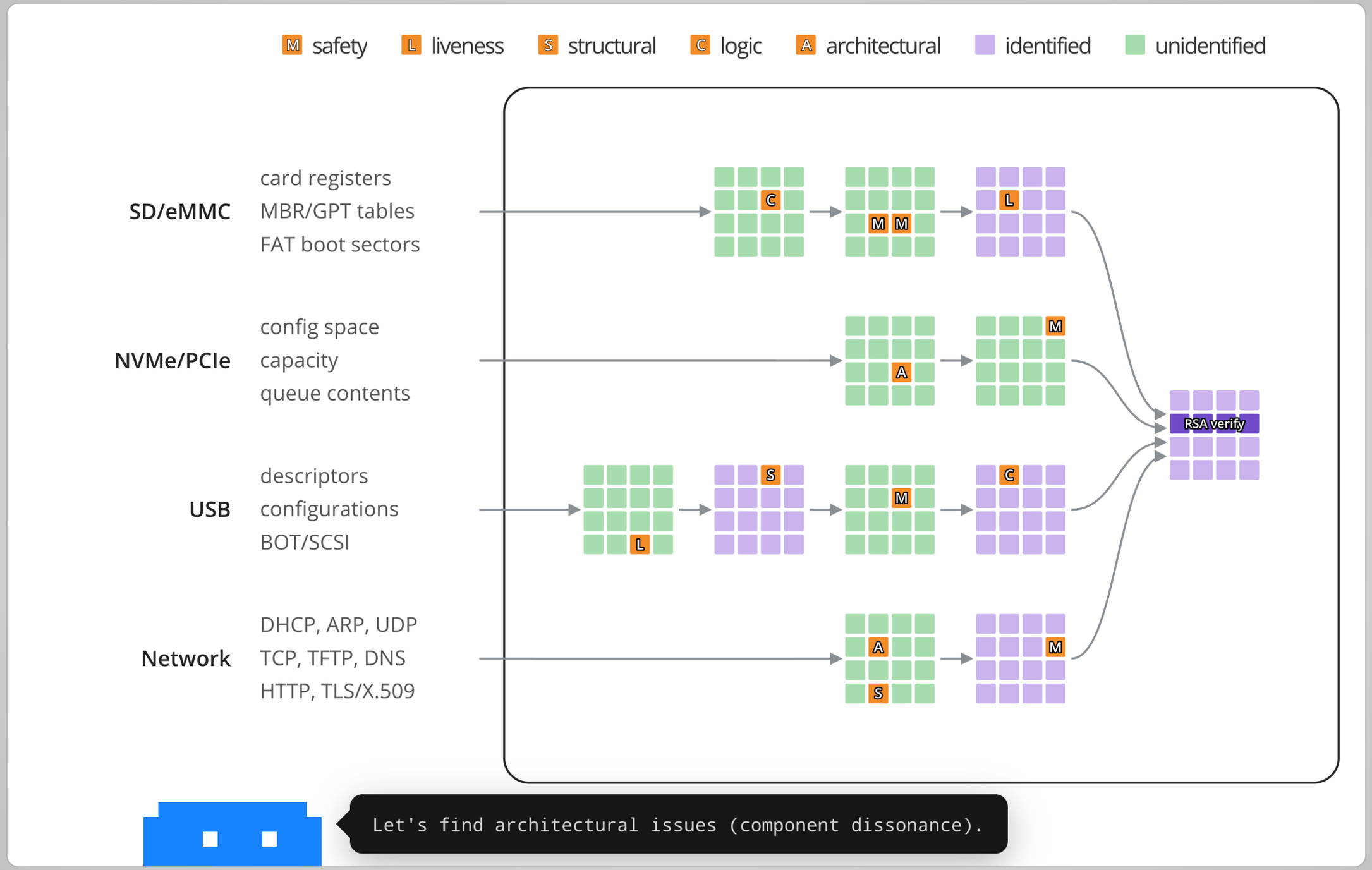

identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
What's in our toolkit? Ugh, too many options...

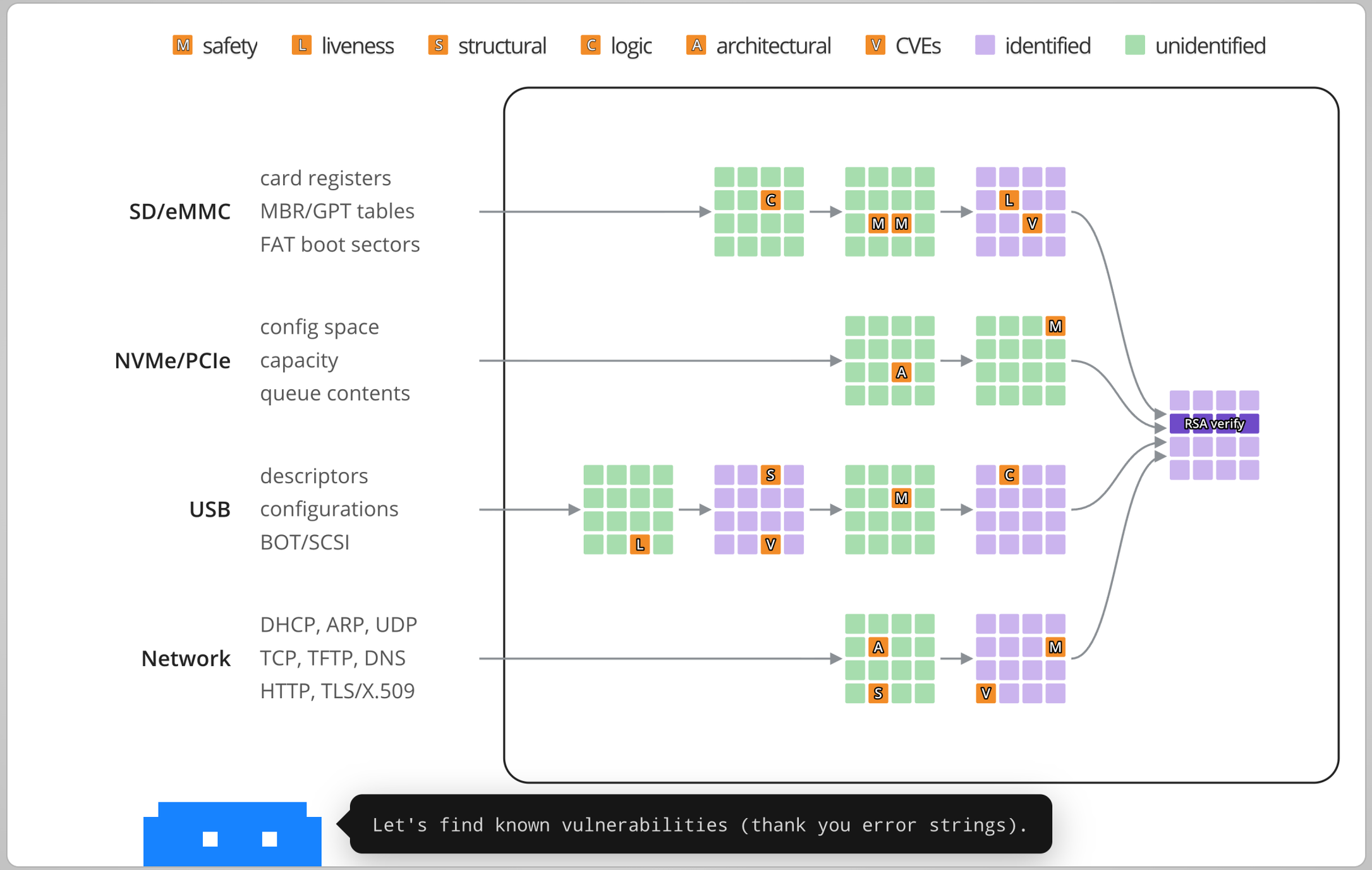

M safety
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
Let's find memory safety issues (out of bounds calls).


M safety
L liveness
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
Let's find liveness issues (infinite loops).


M safety
L liveness
S structural
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
Let's find structural issues (bad if conditions).

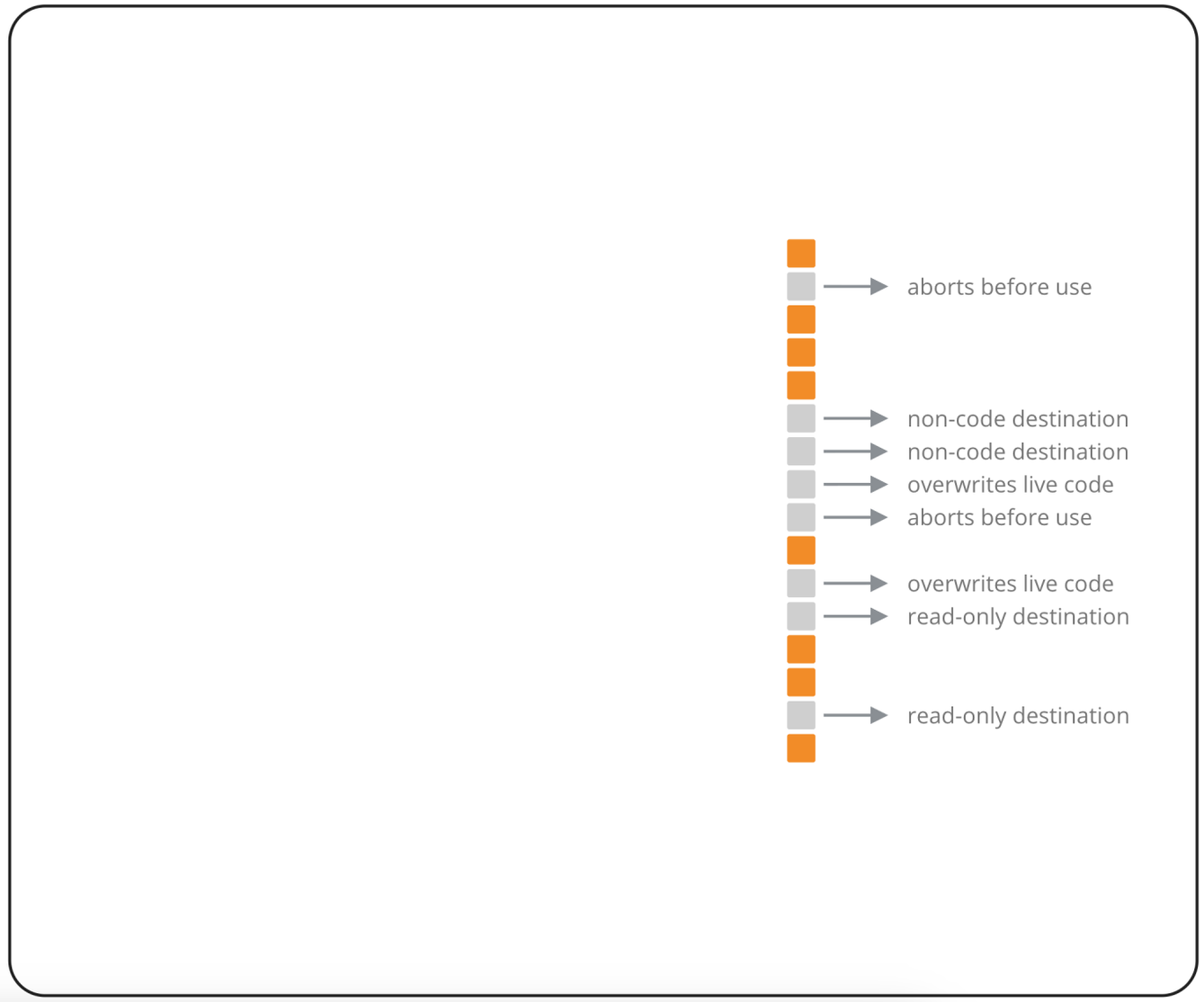

M safety
L liveness
S structural
C logic
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
Let's find code logic issues (plain programming bugs).


M safety
L liveness
S structural
C logic
A architectural
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
Let's find architectural issues (component dissonance).


M safety
L liveness
S structural
C logic
A architectural
V CVEs
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
RSA verify
Let's find known vulnerabilities (thank you error strings).

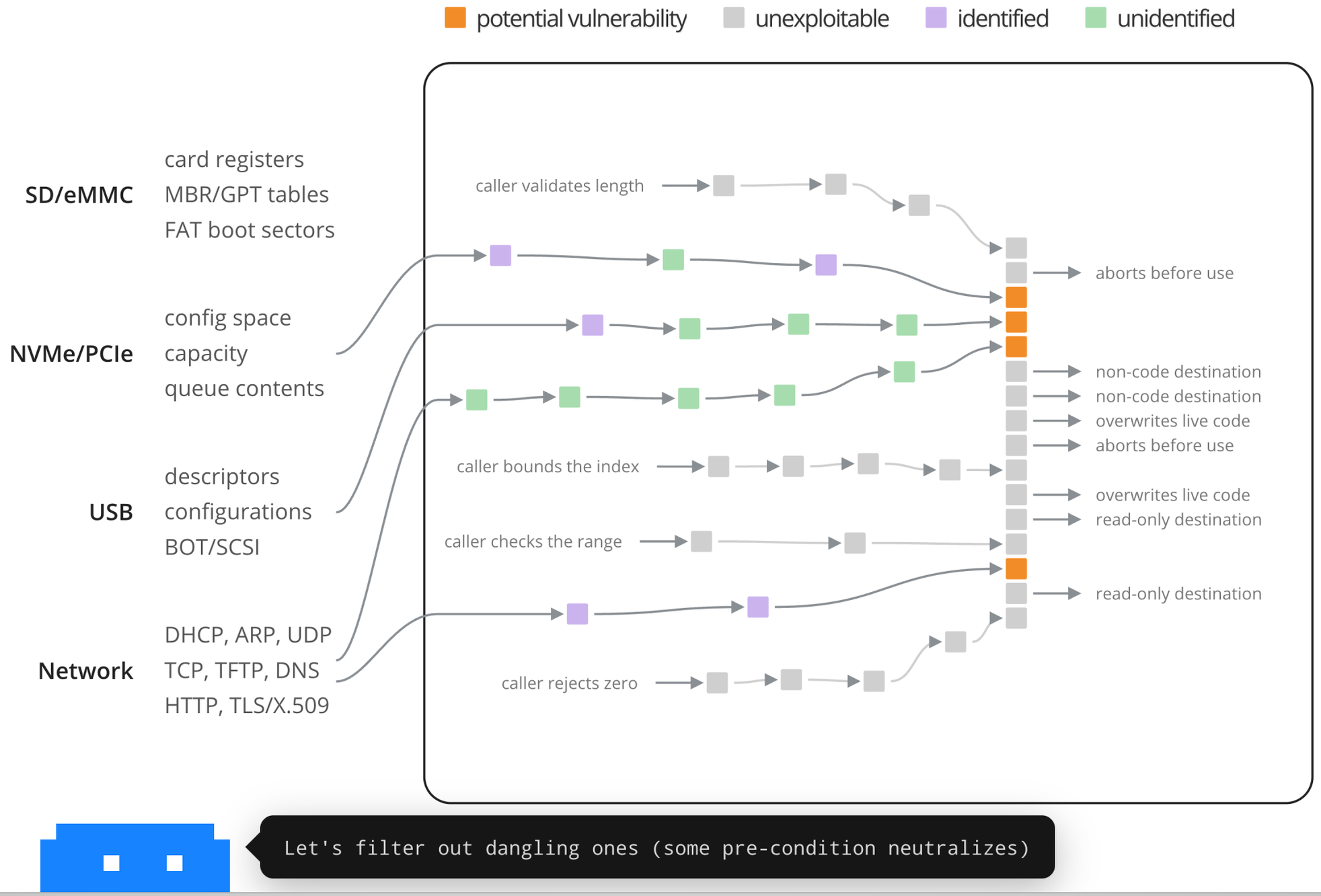
potential vulnerability unexploitable identified unidentified

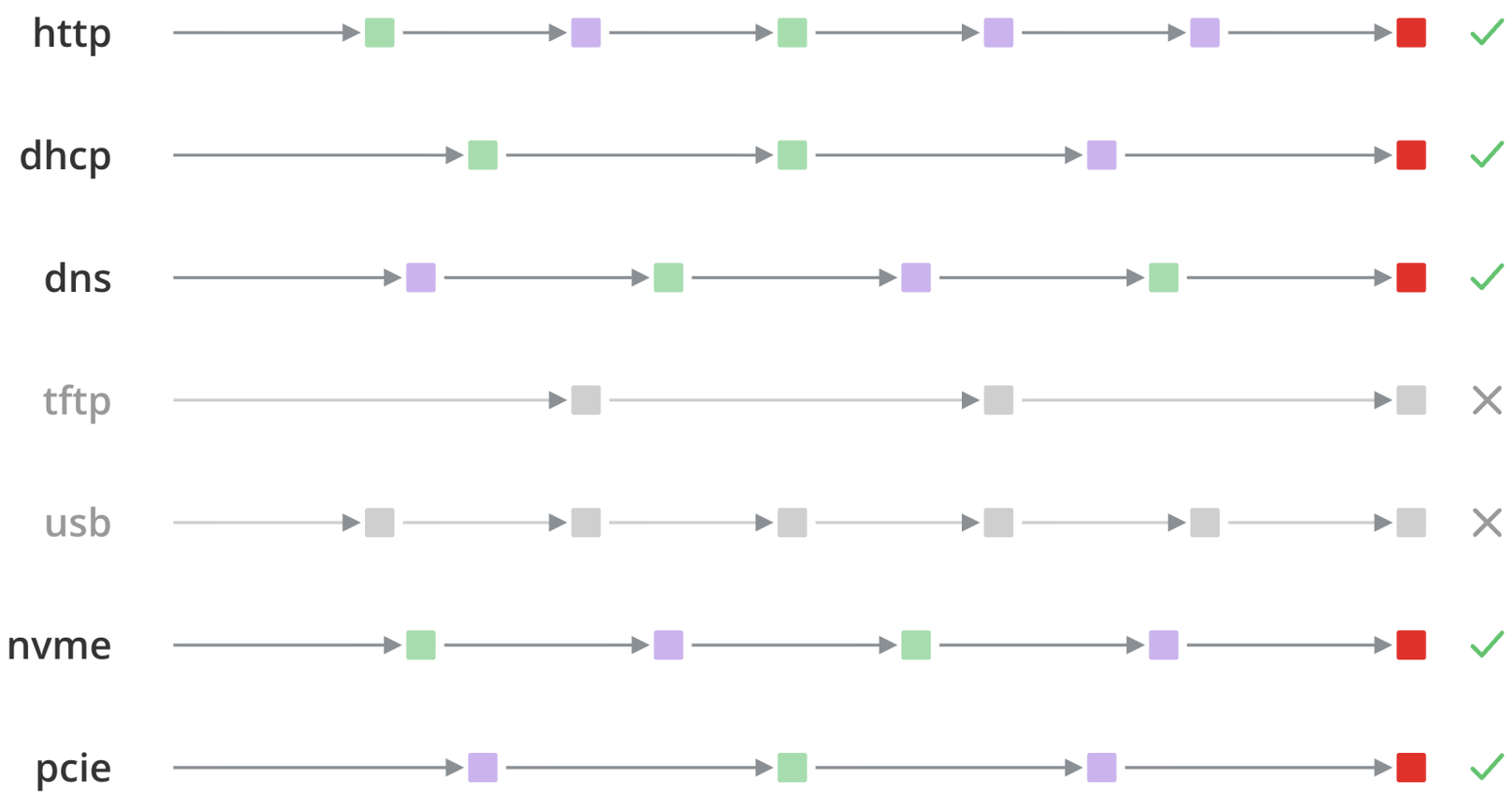
SD/eMMC	card registers MBR/GPT tables FAT boot sectors
NVMe/PCIe	config space capacity queue contents
USB	descriptors configurations BOT/SCSI
Network	DHCP, ARP, UDP TCP, TFTP, DNS HTTP, TLS/X.509





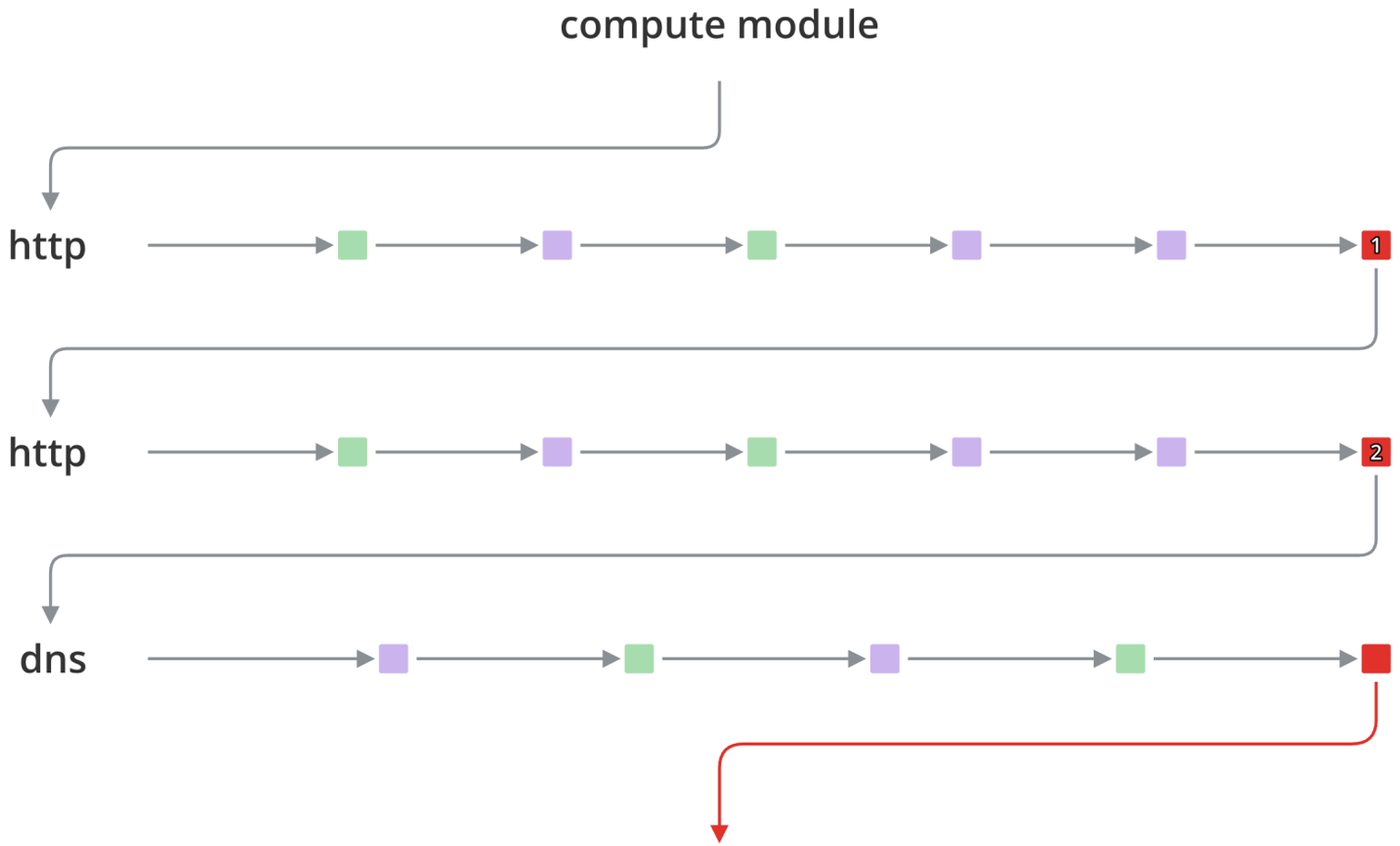
Let's filter out unexploitable ones (side effect benign or crasher).


potential vulnerability
unexploitable
identified
unidentified
SD/eMMC
card registers
MBR/GPT tables
FAT boot sectors
NVMe/PCIe
config space
capacity
queue contents
USB
descriptors
configurations
BOT/SCSI
Network
DHCP, ARP, UDP
TCP, TFTP, DNS
HTTP, TLS/X.509
caller validates length
caller bounds the index
caller checks the range
caller rejects zero
aborts before use
non-code destination
non-code destination
overwrites live code
aborts before use
overwrites live code
read-only destination
read-only destination
Let's filter out dangling ones (some pre-condition neutralizes)





Static analysis done. Let's simulate ALL the code!



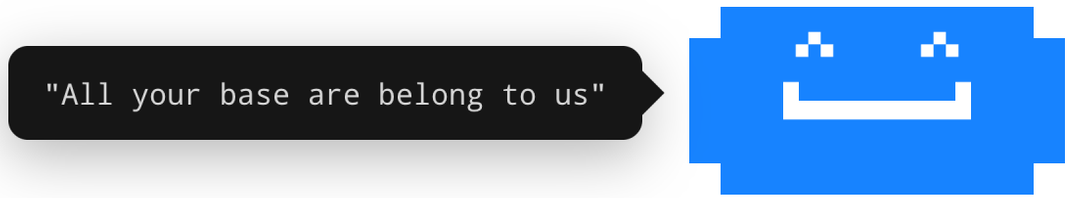


Let's recombine the vulnerabilities into an exploit!

15 zero-day vulnerabilities

3 secure-boot escapes

1 hour on $100 sub



Thank you

Read	the whitepaper	dark.bio/whitepaper
Try	the ark emulator	dark.bio/emulator
Build	an analysis app	dark.bio/examples
Brief	your own agent	dark.bio/llms.txt



Péter Szilágyi
peter@dark.bio
@peter_szilagyi

Dark Bio
@dark_dot_bio