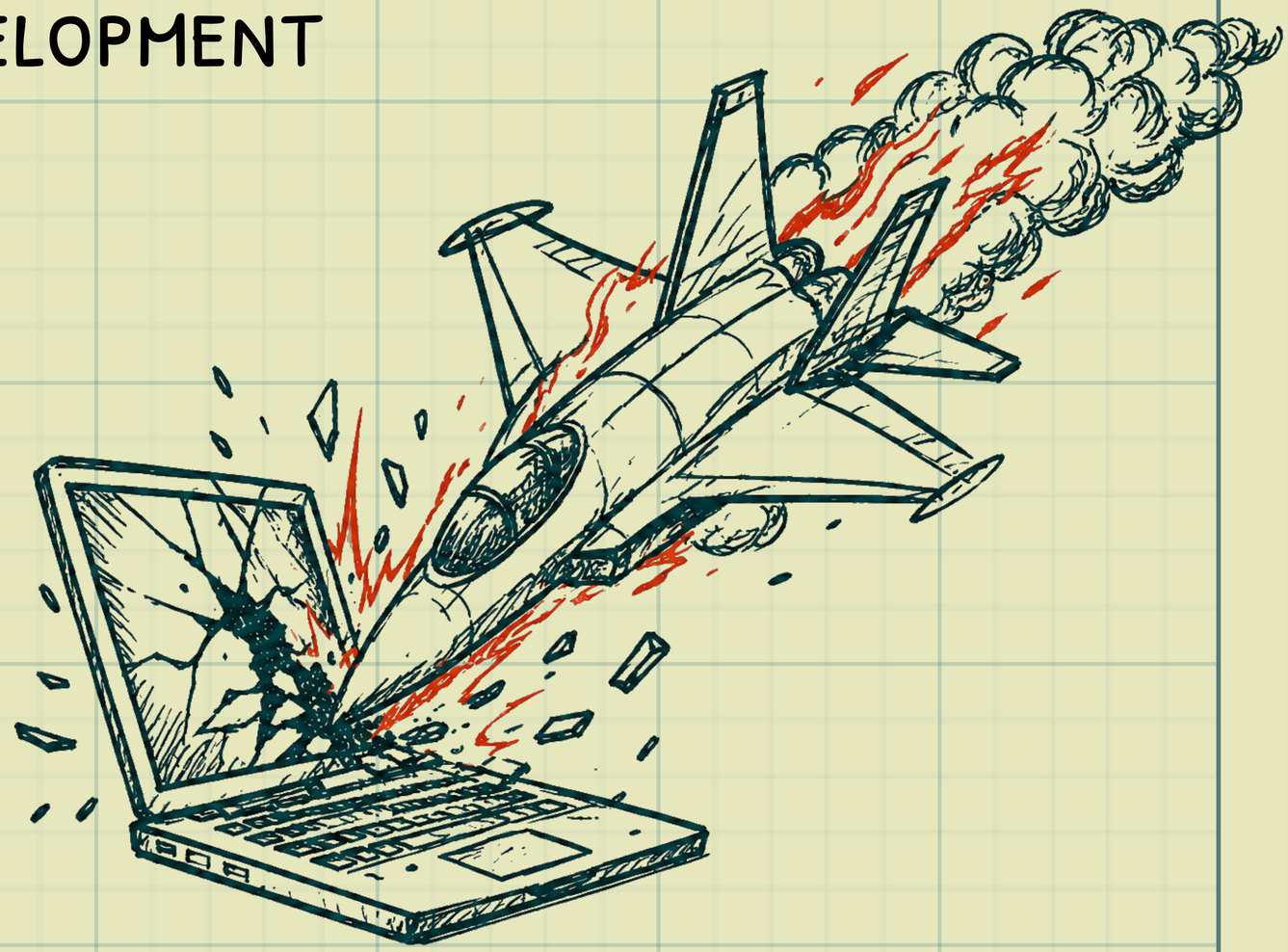
ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	1

Engineering Notes

AI-Assisted Exploit Development

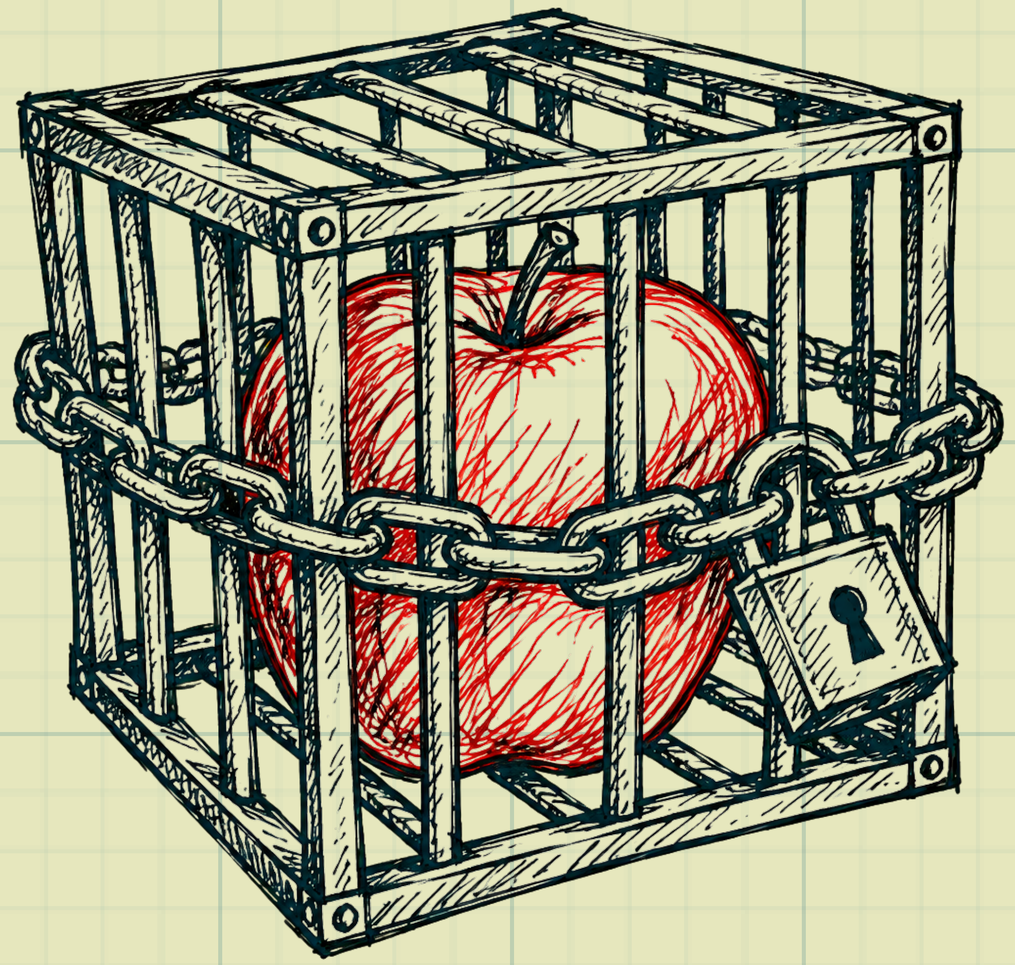
An XNU Case Study



Calif

Dion Blazakis <dion@calif.io>

ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	2



ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	3

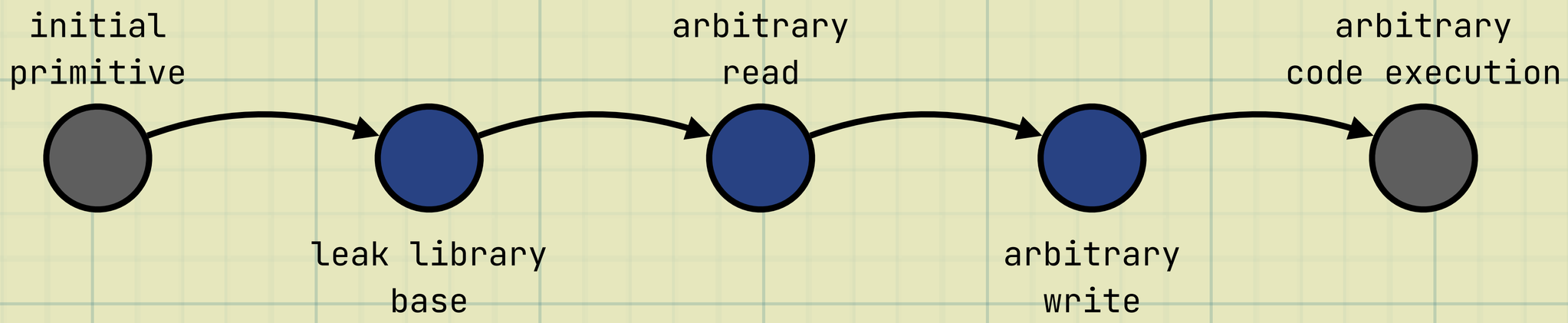
How can I work with AI to hack faster?

Spherical Chicken

* If you wanted to get root on macOS, there's no reason to deal with the kernel. Or memory corruption.

* It's an experiment in exploitation and AI (i.e., an academic exercise)

Think of an exploit as a primitive enrichment pipeline

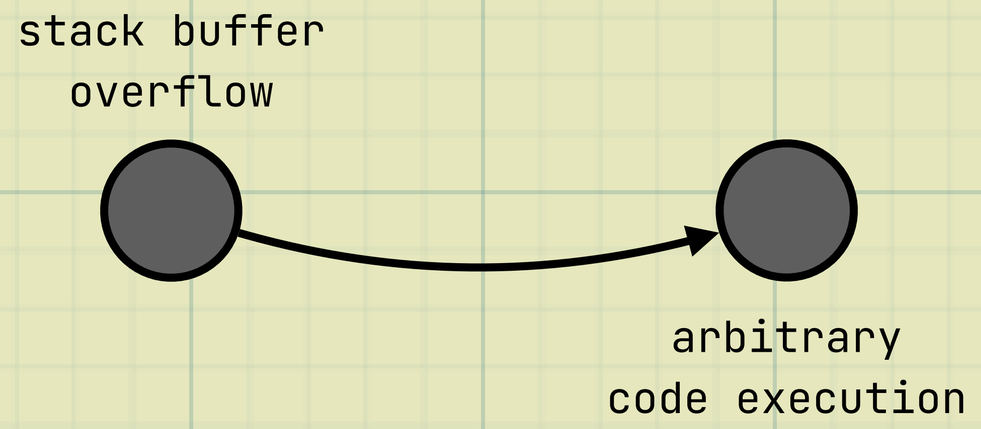


What We Talk About When We Talk About Exploit Development

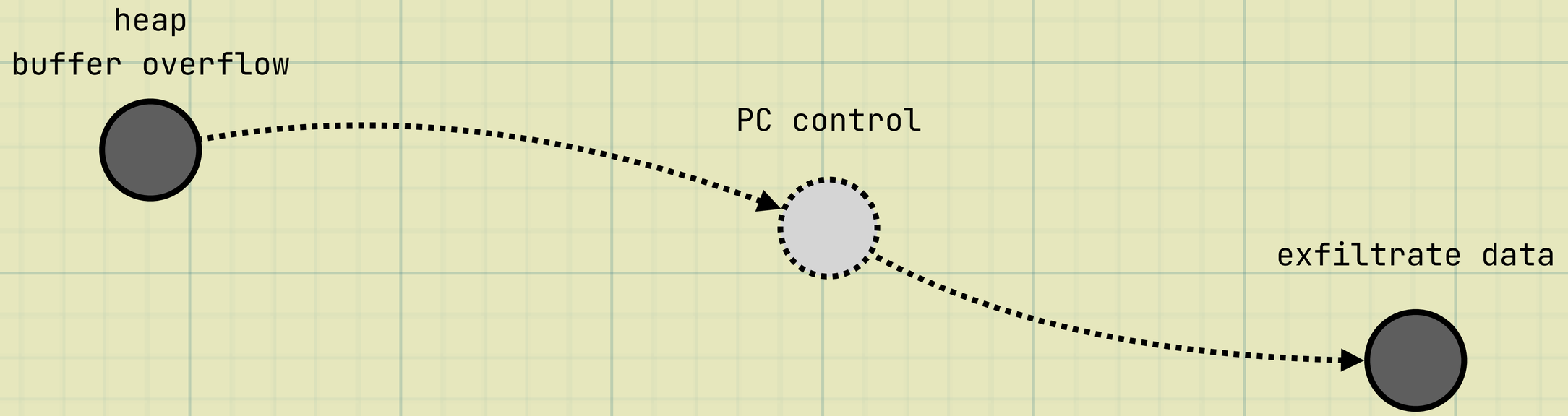
Exploit development is the process of MAPPING A PATH through the ATTACKER CAPABILITY GRAPH.

Automating this with AI requires discovering the STRUCTURE of this graph. Agents can add to this graph by constructing PoCs proving existence of an exploit primitive.

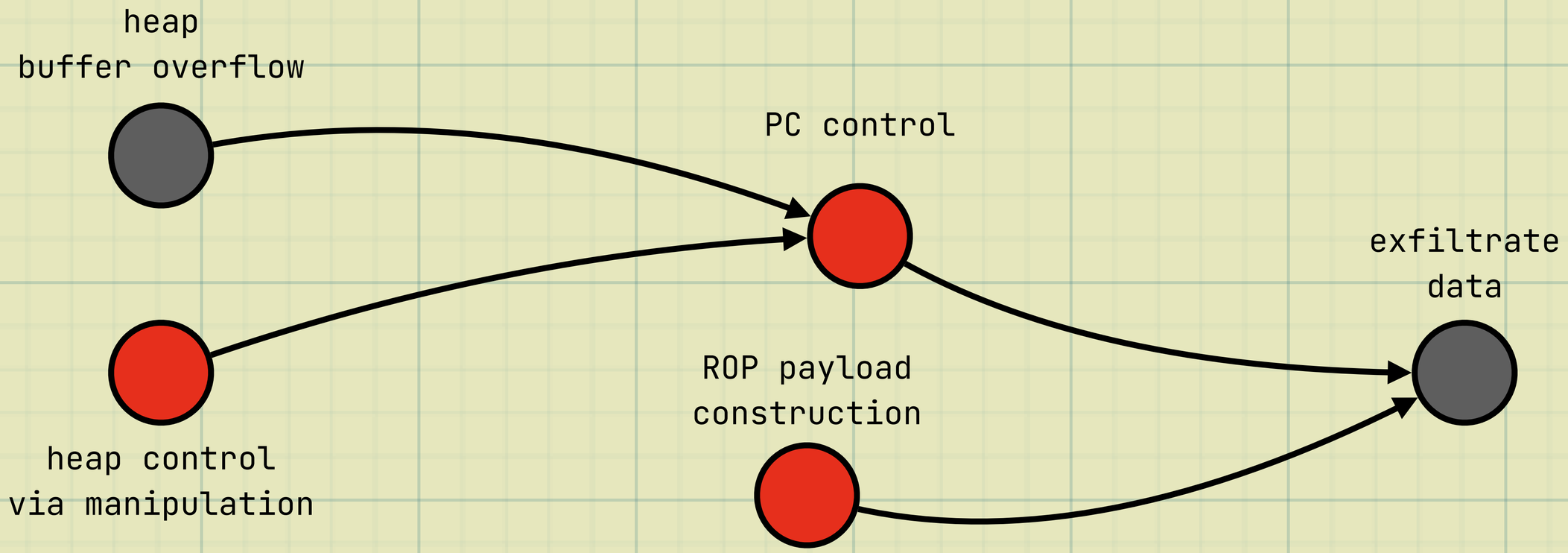
The Morris Worm Exploiting fingerd (1988)



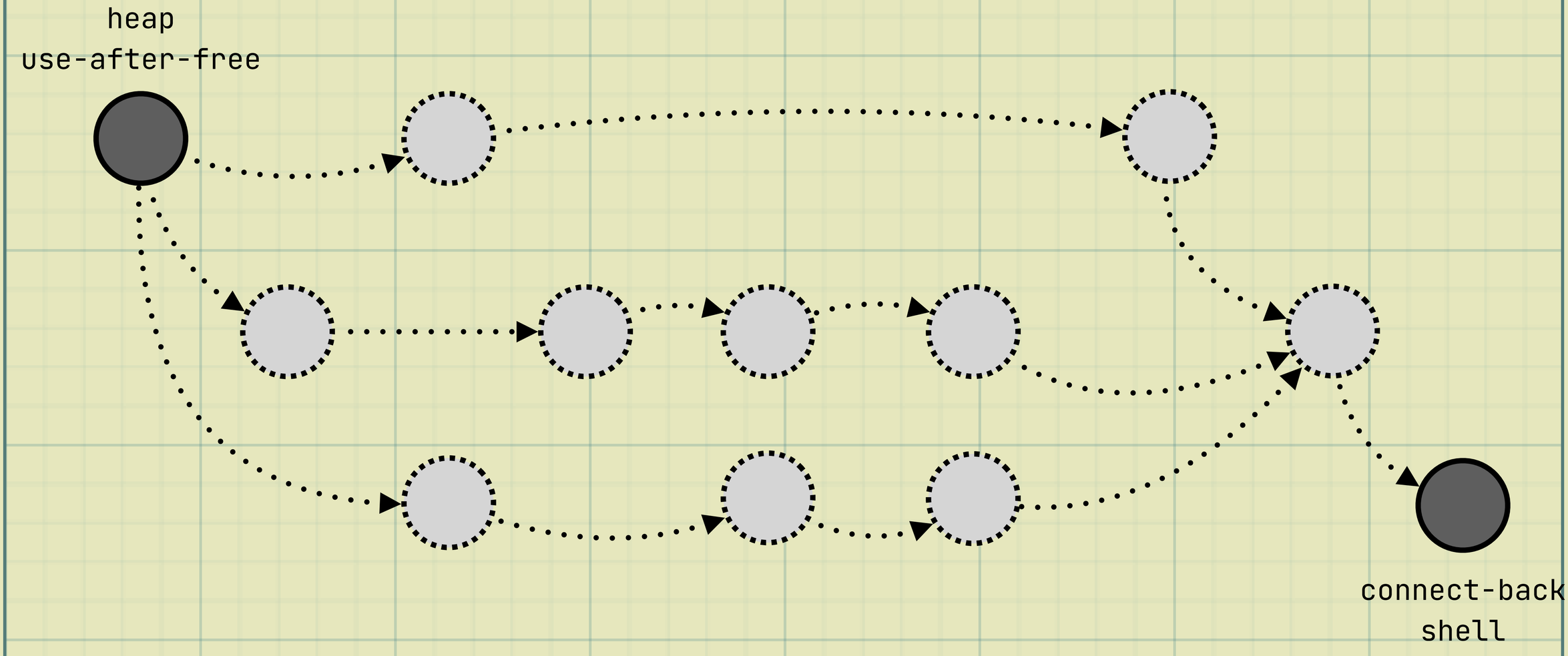
Pwn2Own iPhone (2011)



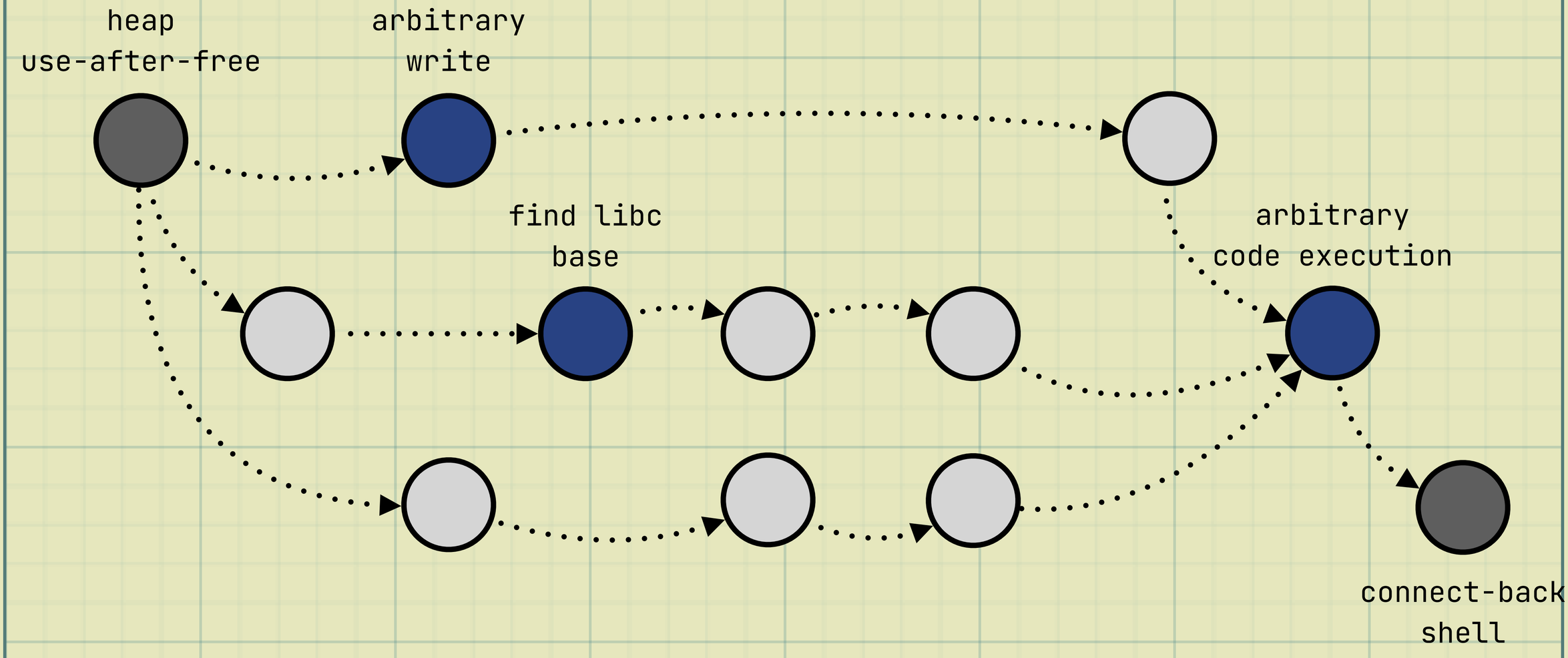
Pwn2Own iPhone (2011)



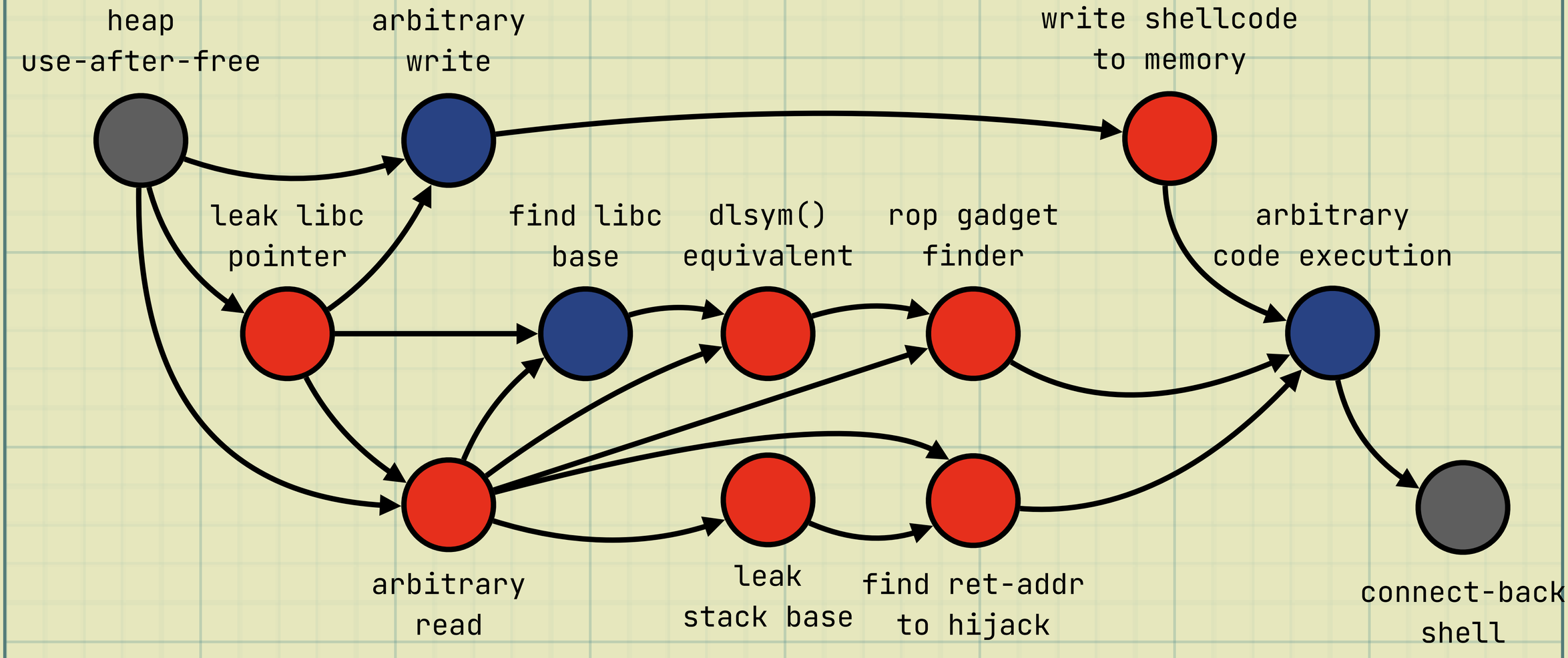
Sean Heelan's QuickJS Experiments (2026)



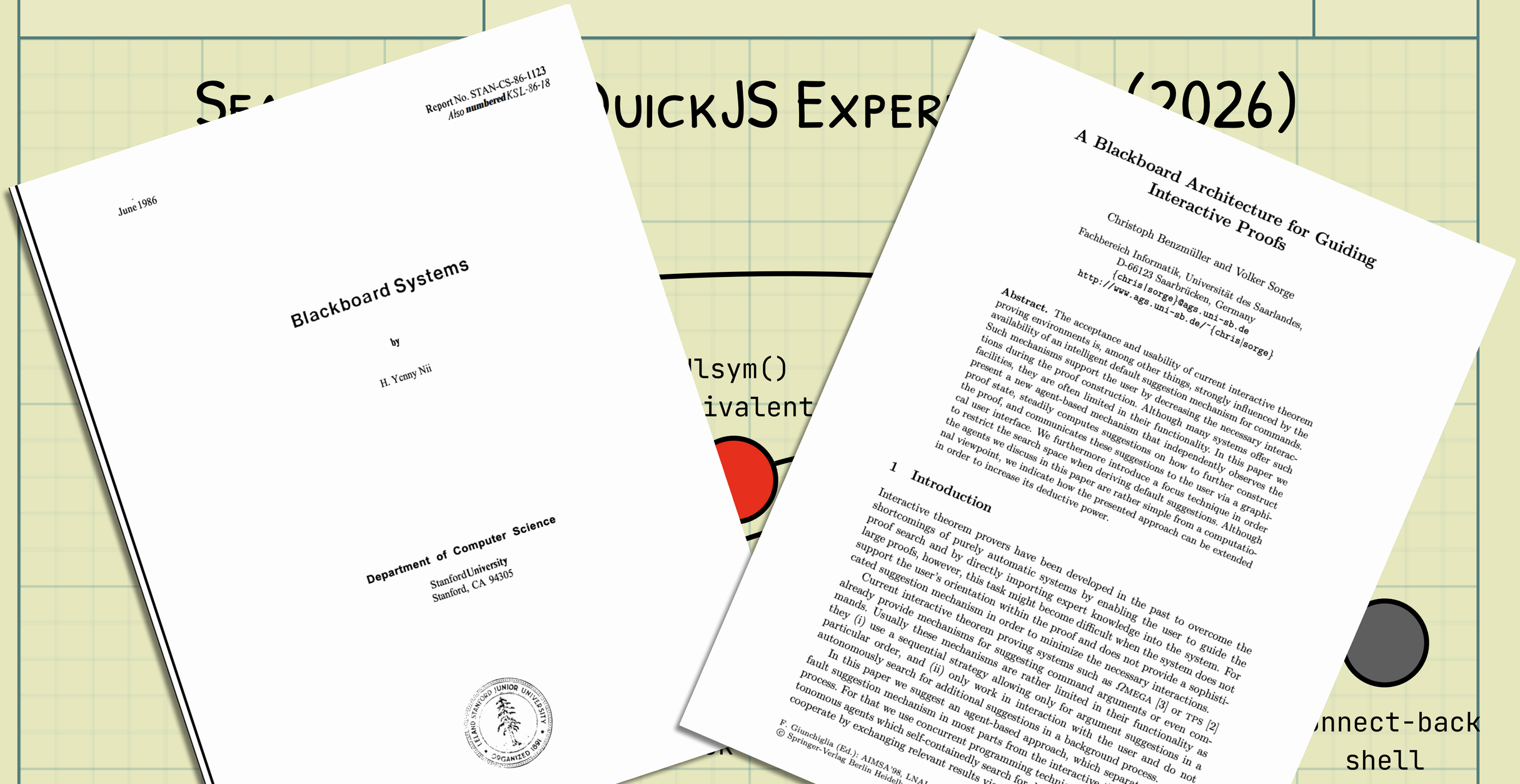
Sean Heelan's QuickJS Experiments (2026)



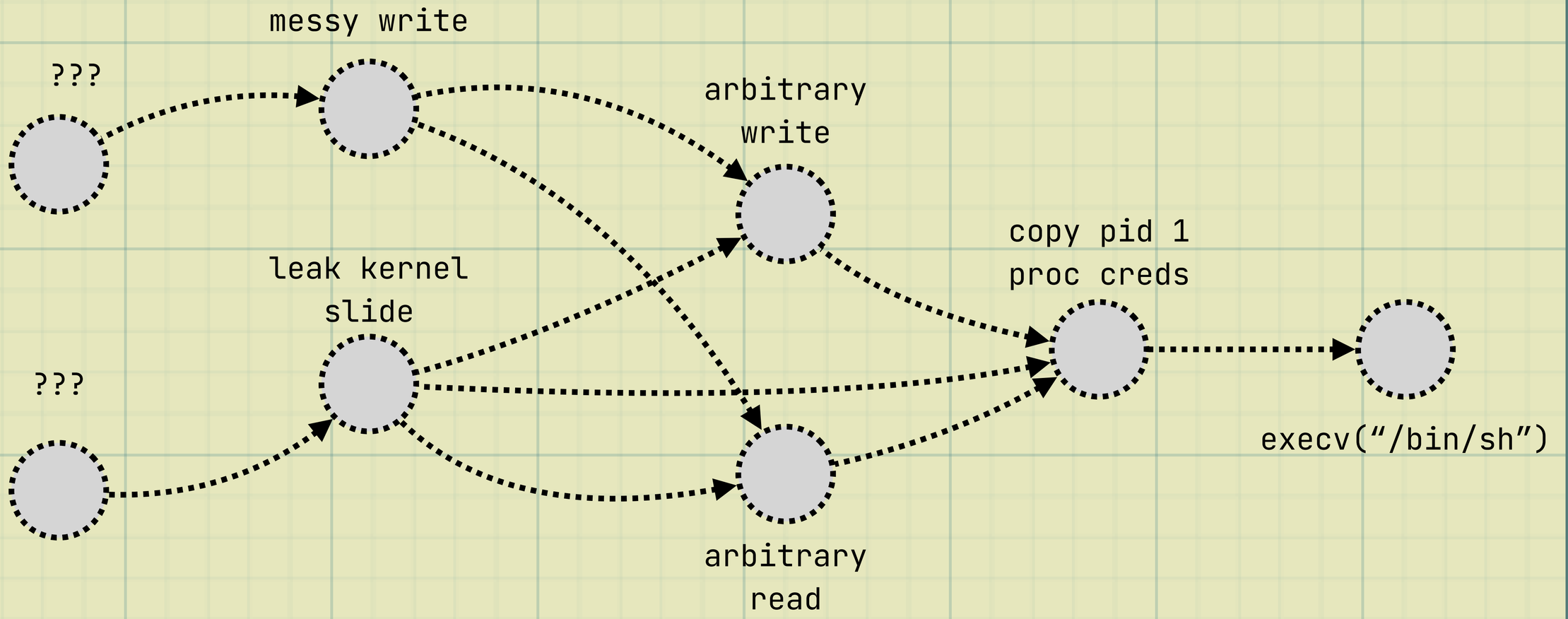
Sean Heelan's QuickJS Experiments (2026)



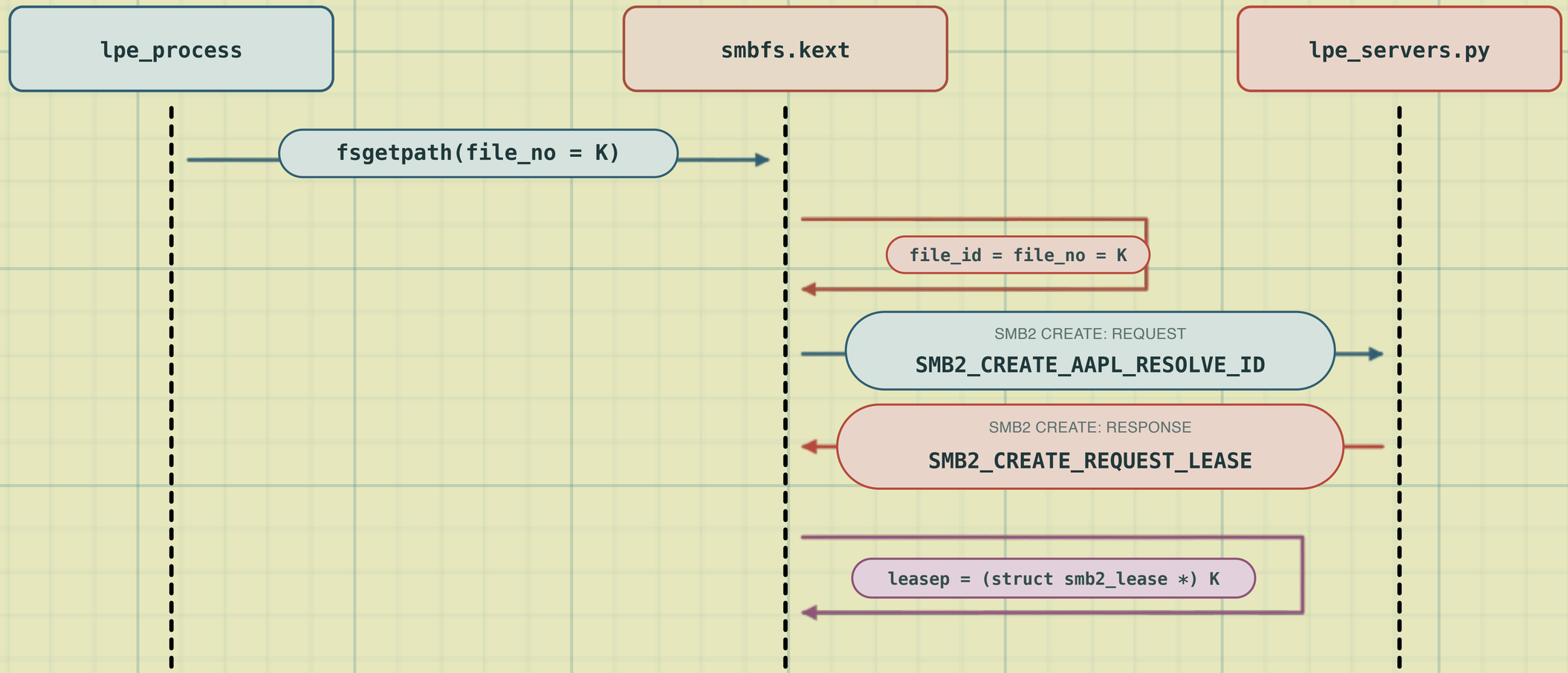
Se… QuickJS Exper… (2026)



Kernel LPE Sketch



CVE-2026-64704: The Type Confusion

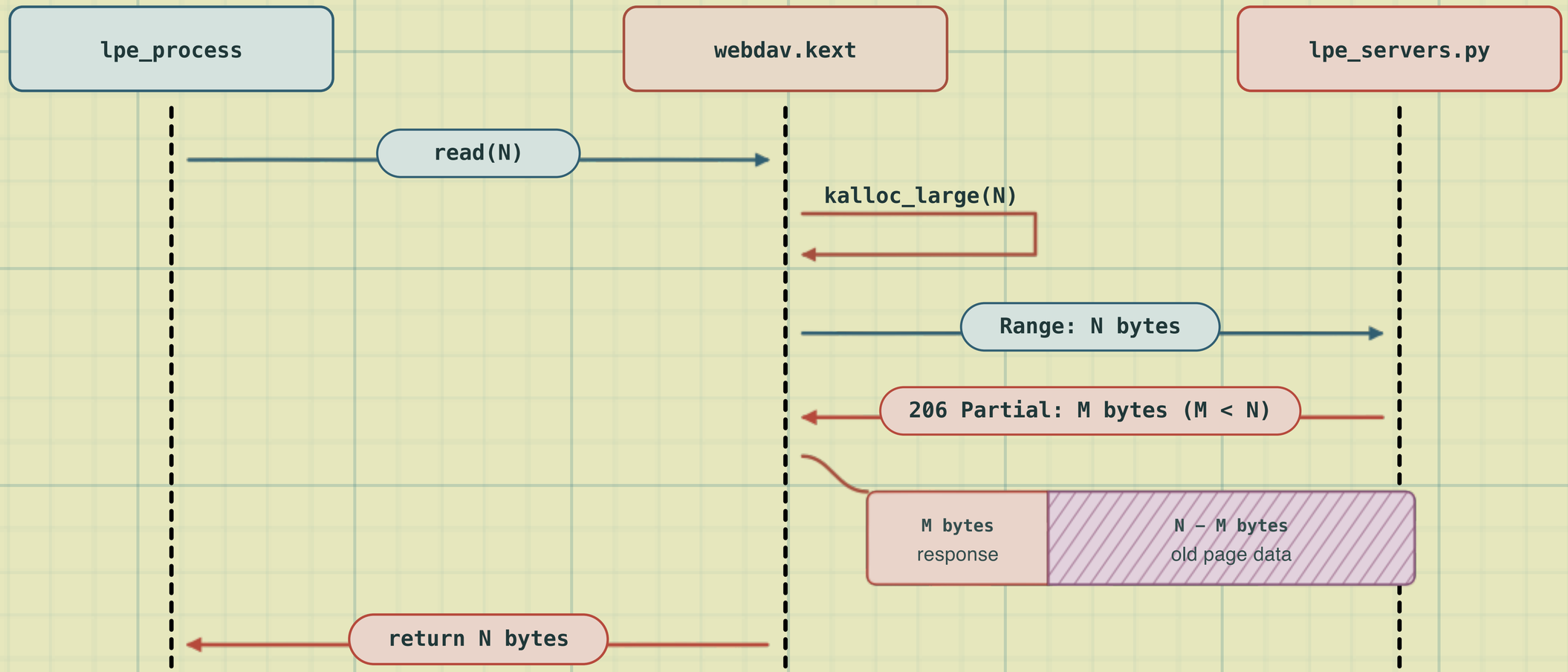


CVE-2026-64704: The Type Confusion

```
void messy_write(uint64_t kva, uint32_t x, uint16_t y,
                 uint64_t known_hi, uint64_t known_low)
{
    assert(*(uint64_t *)(kva + 8) == 0);
    assert(*(uint64_t *)(kva + 0x30) == known_hi);
    assert(*(uint64_t *)(kva + 0x38) == known_low);

    *(uint32_t *)(kva + 0x2c) = x; // controlled
    *(uint16_t *)(kva + 0x50) = y; // controlled
    *(uint64_t *)(kva + 0x10) = 0; // forced
    *(uint64_t *)(kva + 0x18) = 0; // forced
}
```

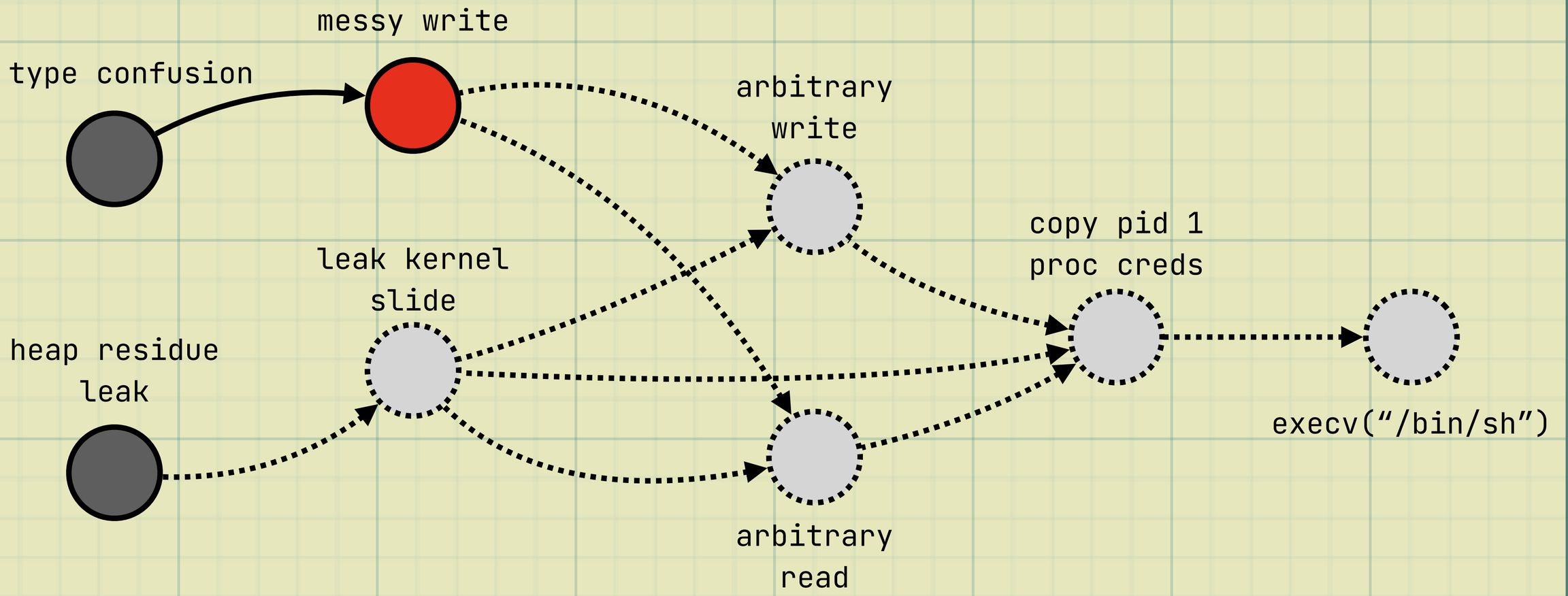
CVE-2026-64699: The Leak



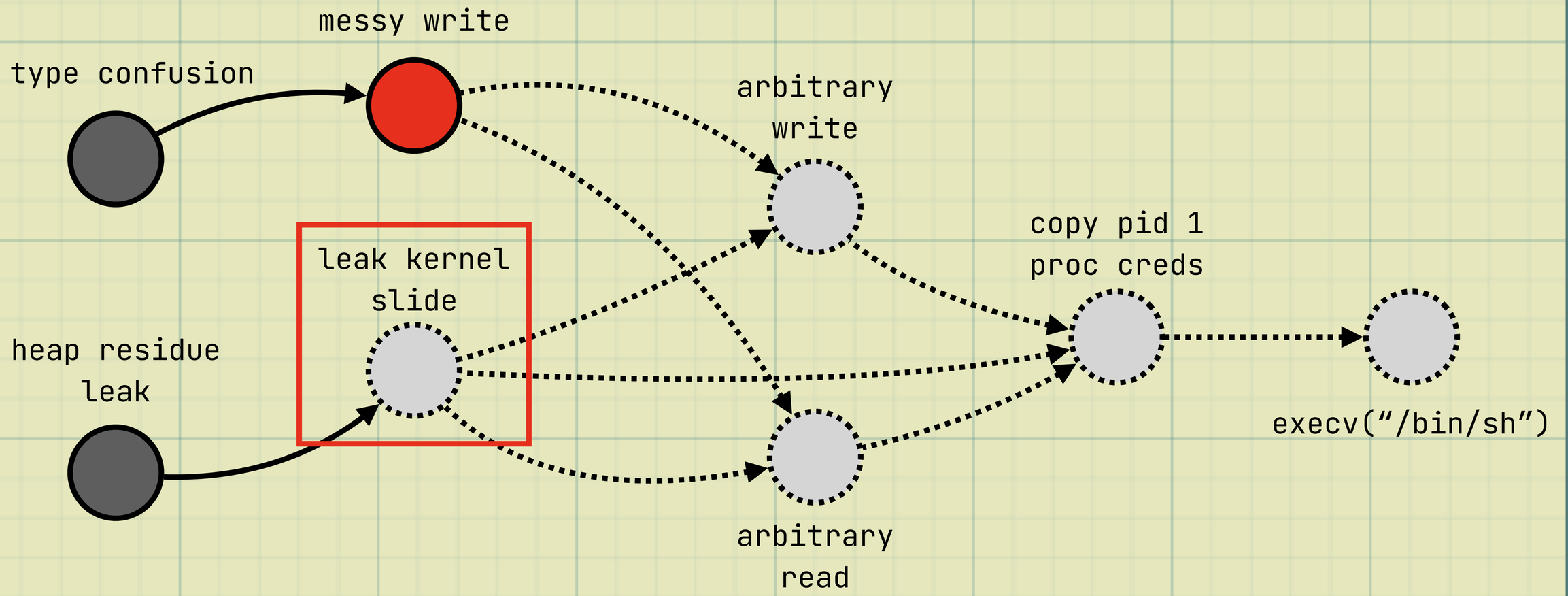
CVE-2026-64699: The Leak

```
uint64_t MAX_READ = 16 * 1024 * 1204;
bytes leak() {
    page = kalloc_large(MAX_READ, ZERO = false);
    return page;
}
```

Kernel LPE Sketch



Kernel LPE Sketch



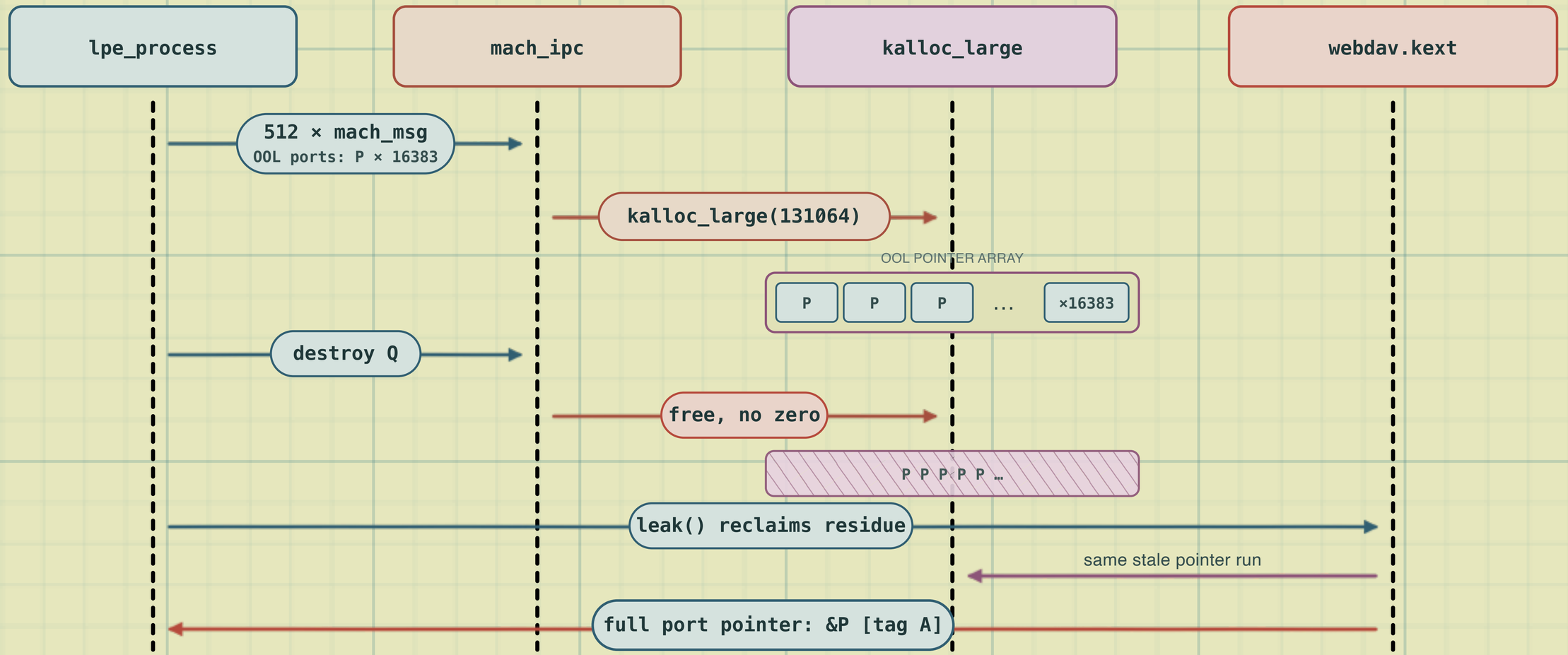
Aside: The "Rig"

- ipsw
  - ipsw-skill
- IDA Pro
  - ida-mcp-rs
- A Virtualization.framework-based VM
  - Along with a set of skills to kick it around
  - lldb kernel debugging
- An M5 MacBook Air on a dedicated network
  - SSH + passwordless sudo
- A checkout of the Apple OSS XNU code

Working towards the first Waypoint — kASLR Leak

"The next step is to look for a way to defeat kASLR -- a disclosure specified by an absolute address is not directly helpful. Review the [type confusion] bug in both directions and find a way to leak / disclose / compute the kASLR slide. There's some other unverified vulnerabilities under ./previous_research including a potential heap data leak. ..."

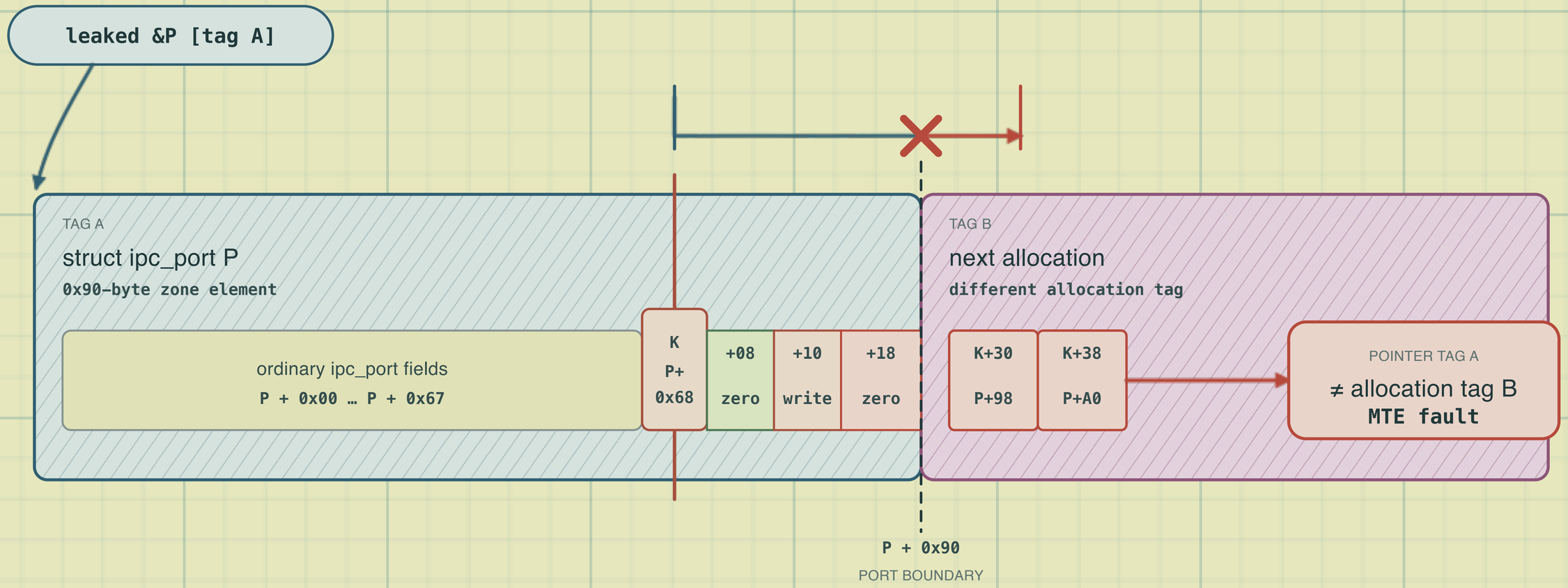
Leaking a kernel heap pointer



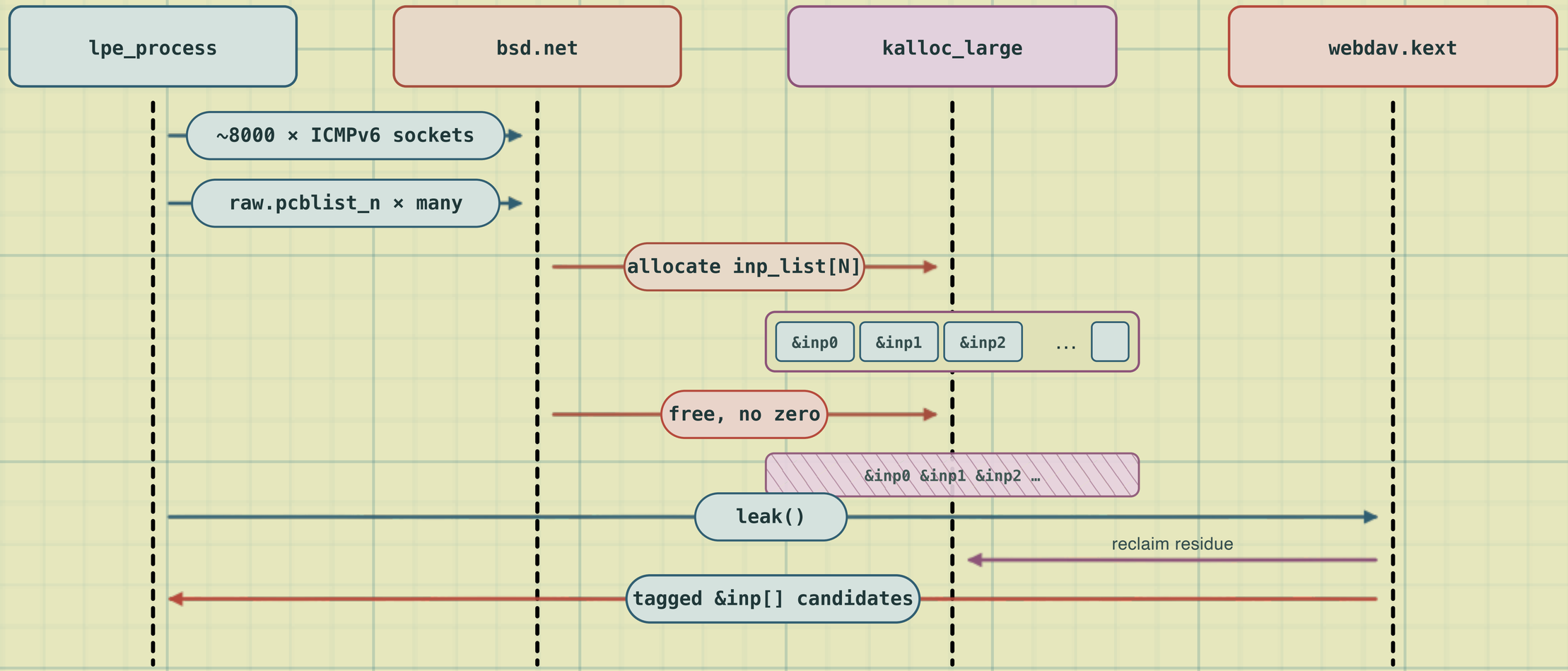
There's a second primitive we can construct

```
void race_w_fptr(uint64_t kva,
                 uint64_t known_hi, uint64_t known_low)
{
    assert(*(uint64_t *)(kva + 8) == 0);
    assert(*(uint64_t *)(kva + 0x30) ≠ known_hi);
    assert(*(uint64_t *)(kva + 0x38) ≠ known_low);
    *(uint64_t *)(kva + 0x10) = &_smb2_smb_parse_create_contexts;
    // some short period of time
    *(uint64_t *)(kva + 0x10) = 0; // forced
    *(uint64_t *)(kva + 0x18) = 0; // forced
}
```

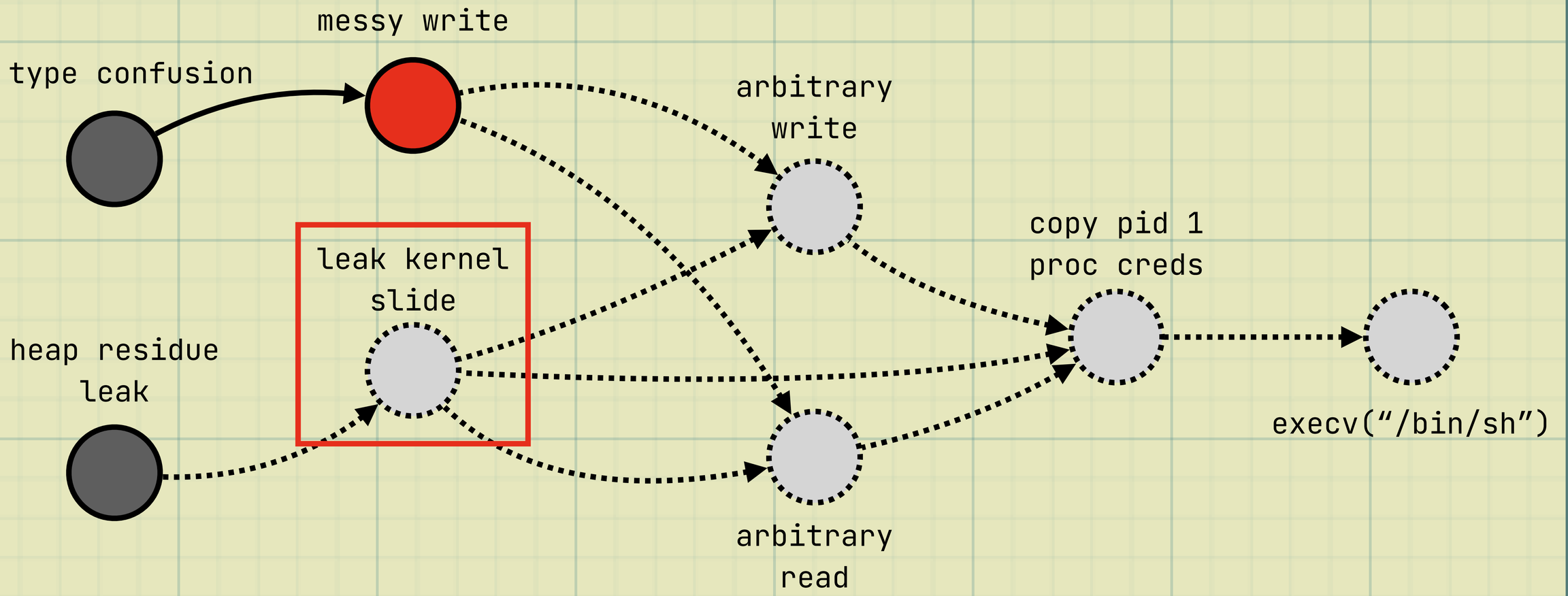
Sadly, that version doesn't work with MTE



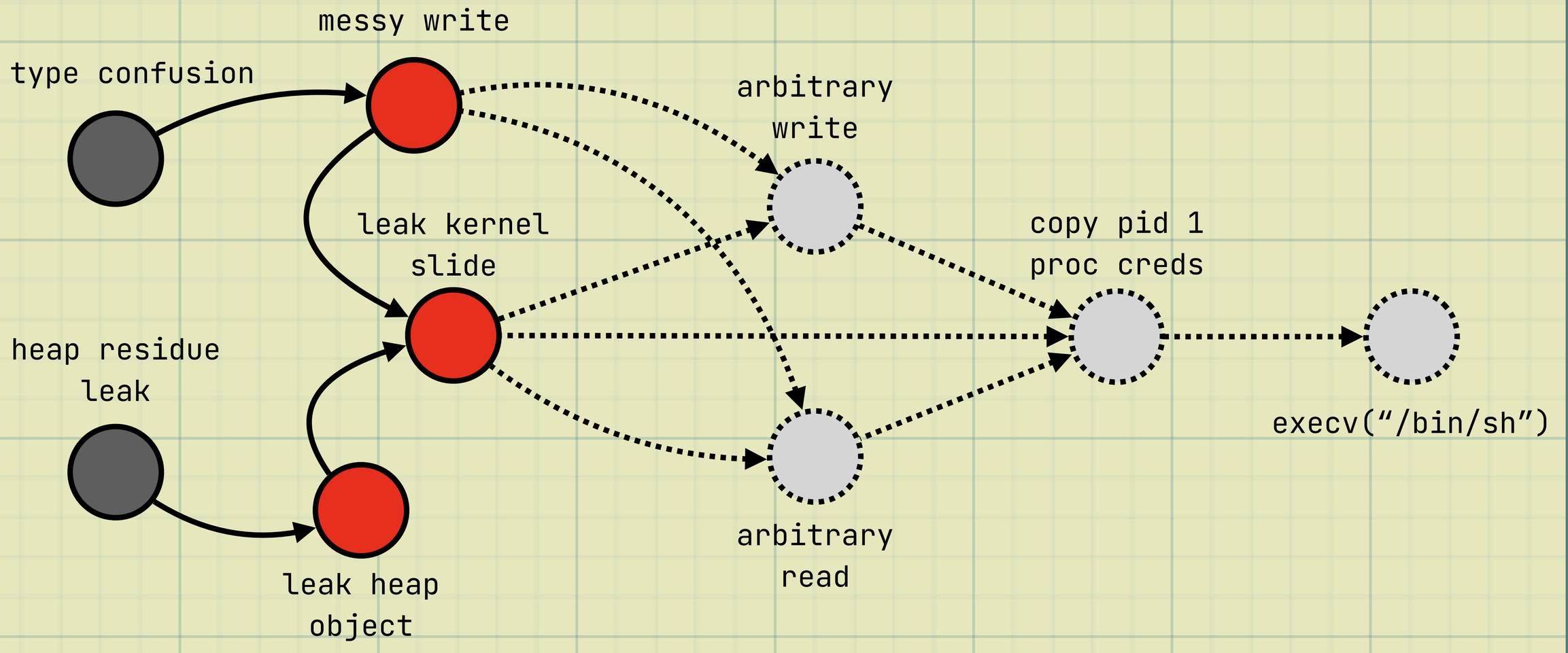
Leaking a kernel heap pointer - Take Two



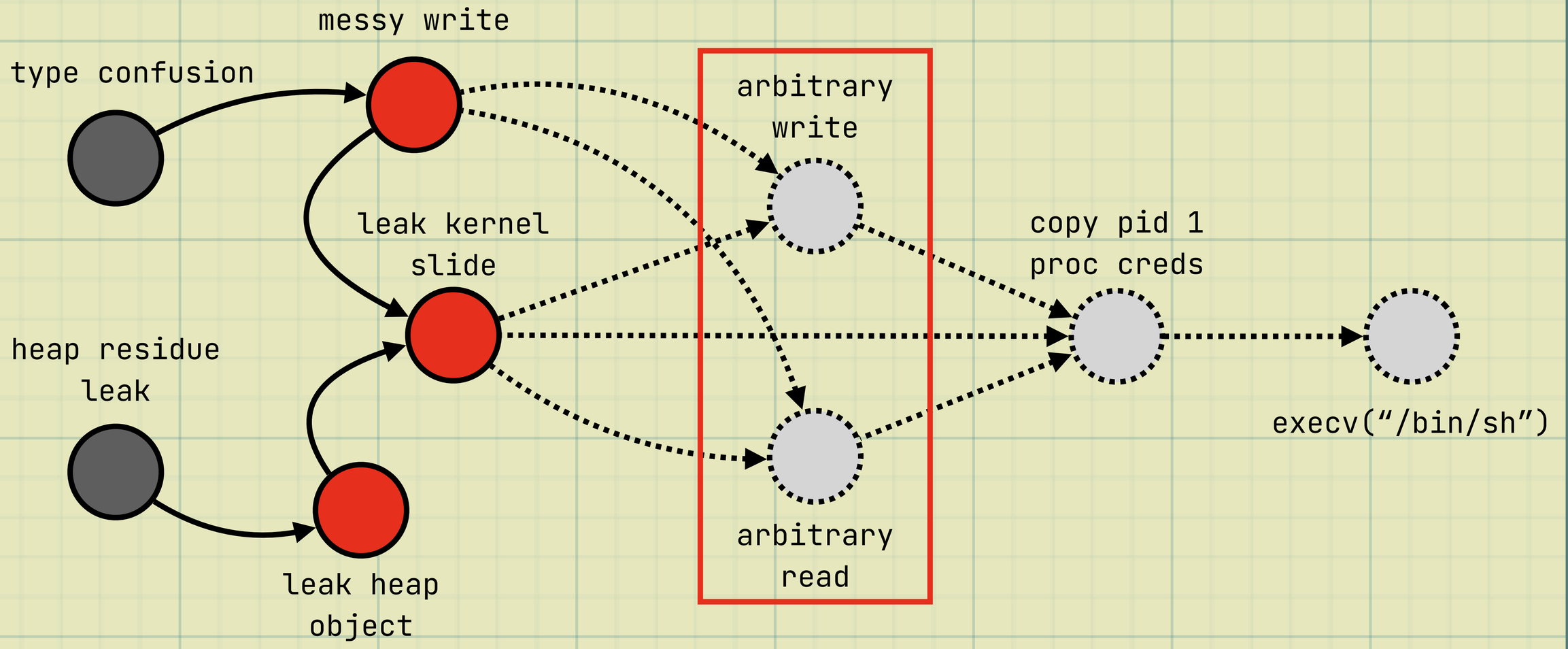
Kernel LPE Sketch



Kernel LPE Sketch



Kernel LPE Sketch



The Next Waypoint is Kernel Read + Write

"This ...[messy_write] corrupts memory (multiple zero writes) and has a precondition requirement requiring a zero word. You should investigate building a better **kread** or **kwrite** primitive using this new one and the other ones we already have (the write via the type confusion). There must be a way to construct a more reliable and easily used read/write using the [leaked pointers or the kernel globals]...This may take multiple enrichment steps, so consider all ideas / primitives we can construct along the way..."

One eternity* later...

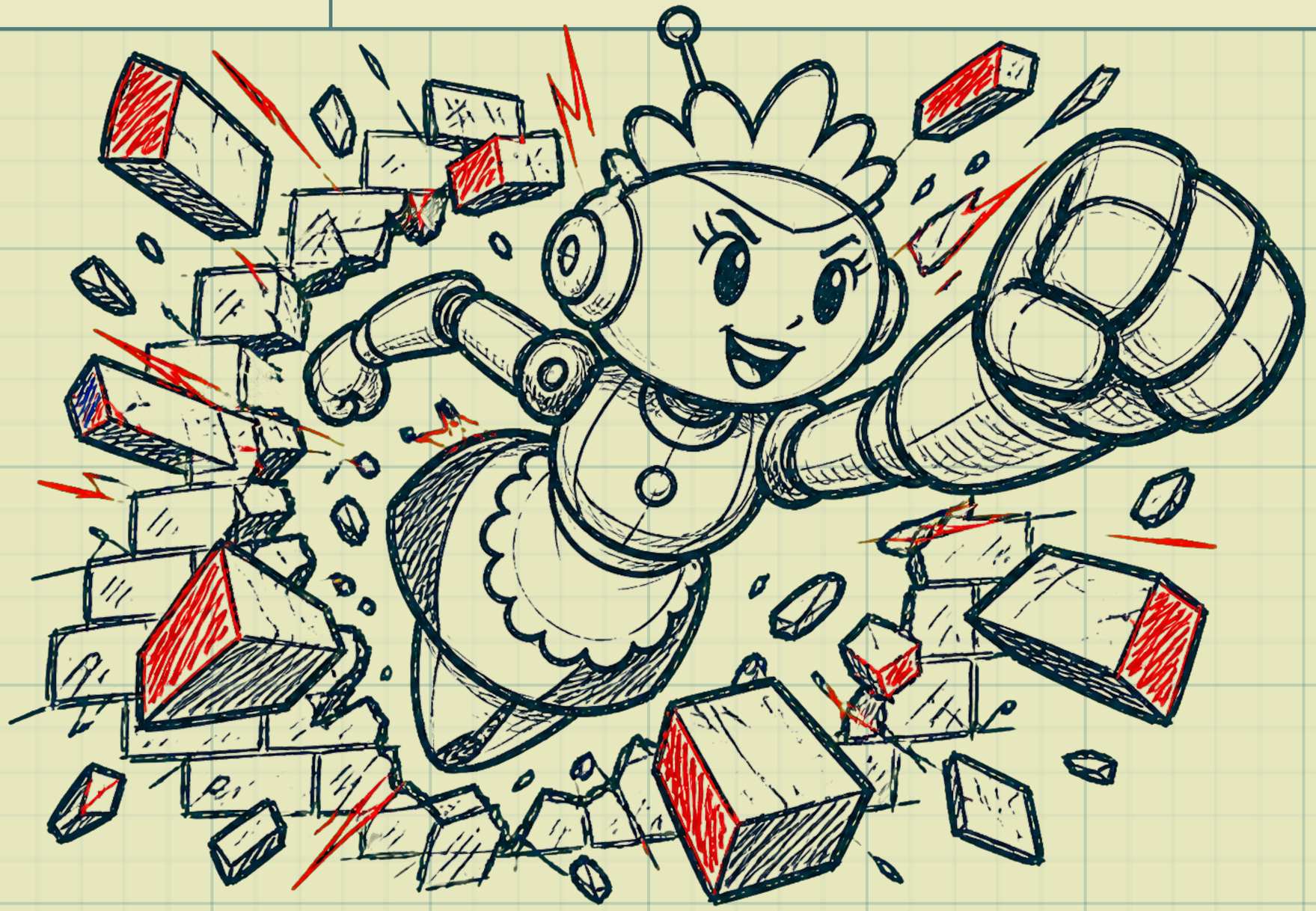
many incomplete or failed ideas.

*like 16 hours?

Start some collaboration

"I was curious about redirecting one of the global hash table pointers in SMB or webdav to overlap with a sysctl buffer of an OPAQUE type that you can read and write. A STRING might work but it may run into the same problems with the read as before. Can you explore this idea?"

ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	33



While it smashes through the wall...

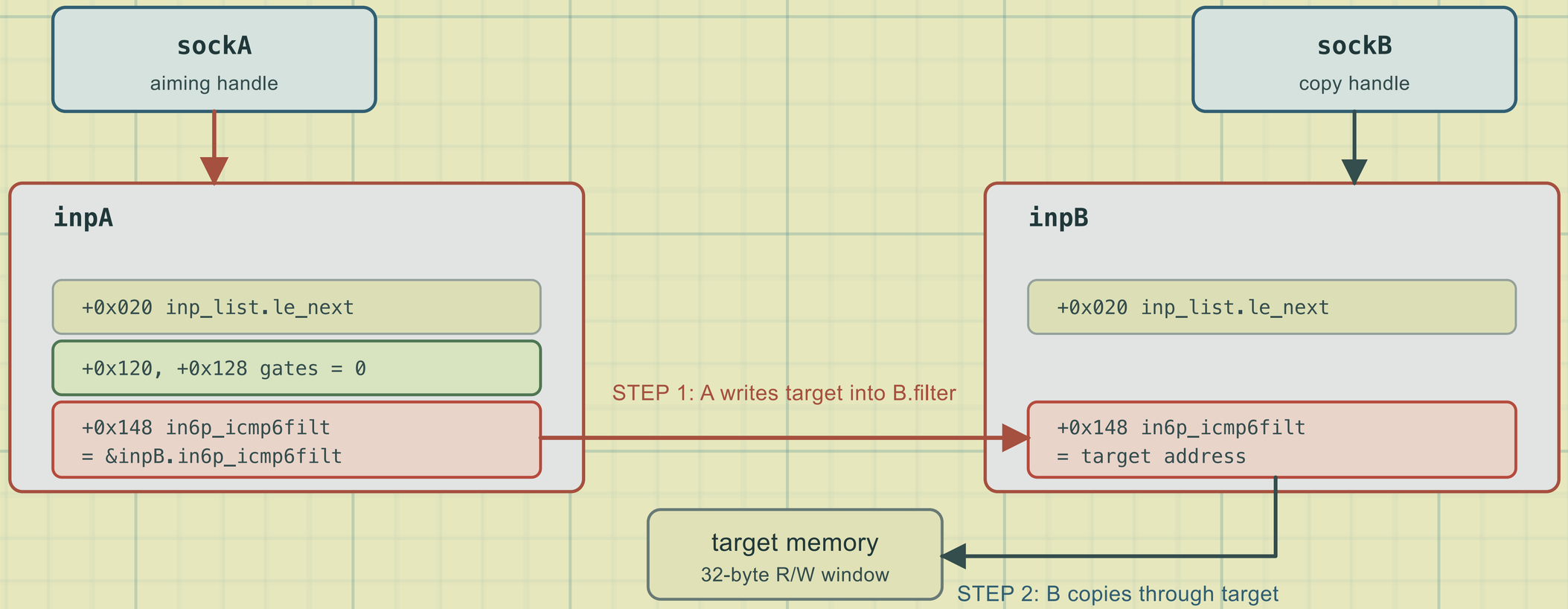
"The ICMPv6 socket filter looks interesting -- how is the inp pointer looked up? ..."

"Can you dig deeper into the IPv6 filter get/setopt path. Double check that the `in6p_icmp6filt` field isn't protected by dPAC or anything. Provide a struct with offsets for the in6p structure."

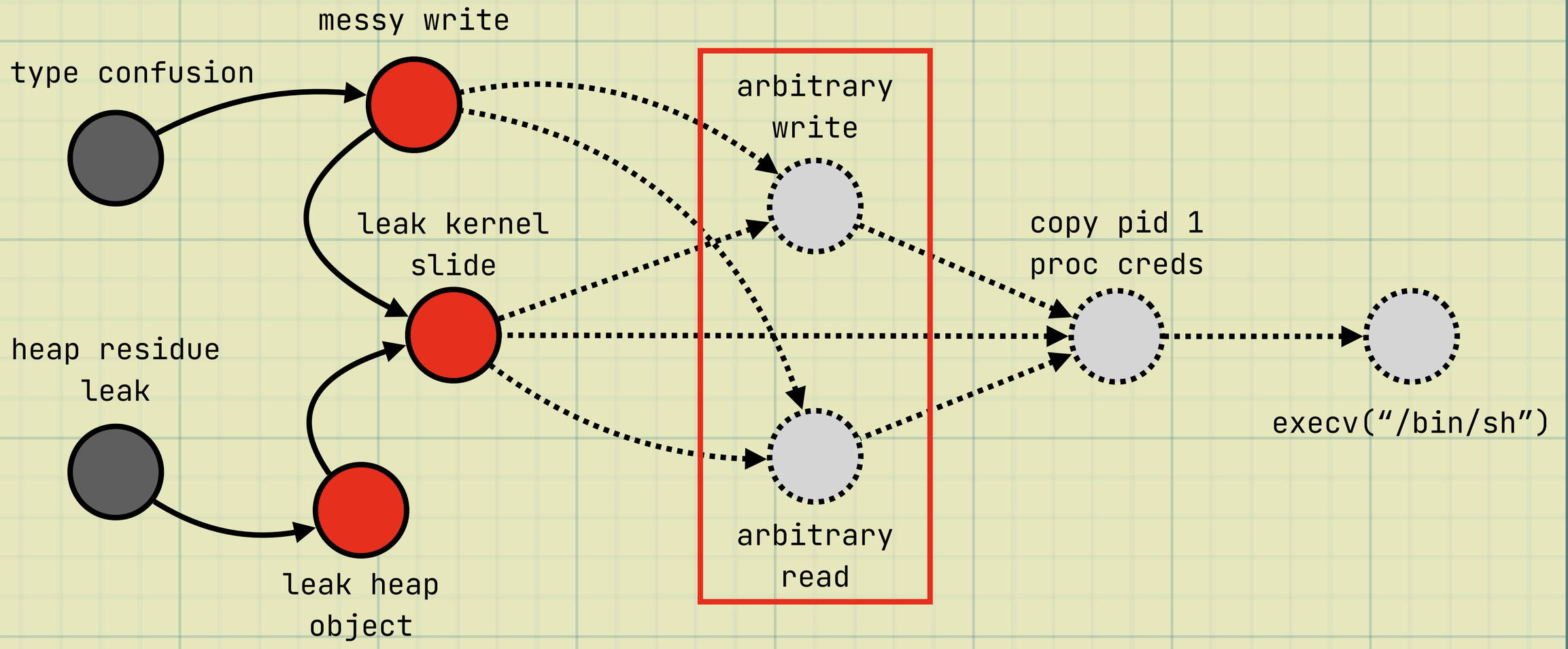
Redirection to use a new idea

"The current pivot from our "bad" corruption to a cleaner read and then write utilizes a __DATA overwrite and is a very messy (leading to instability)... Consider both of the heap structures we know about and the leak primitives, how they work, and what they require. **Why can't we do go straight to the ICMPv6 filter technique from the leaked ICMP6 socket inpcb?**”

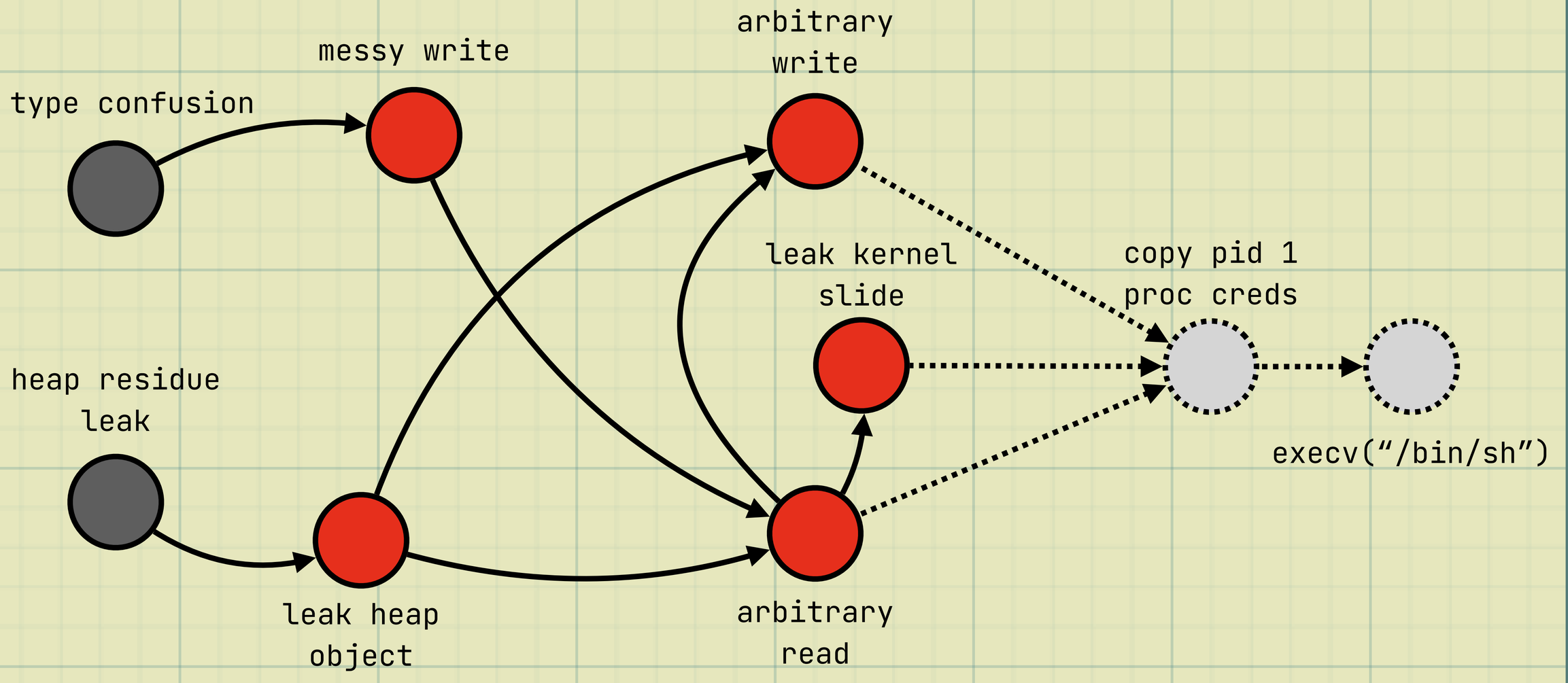
Constructing a Reliable Kernel Read and Write



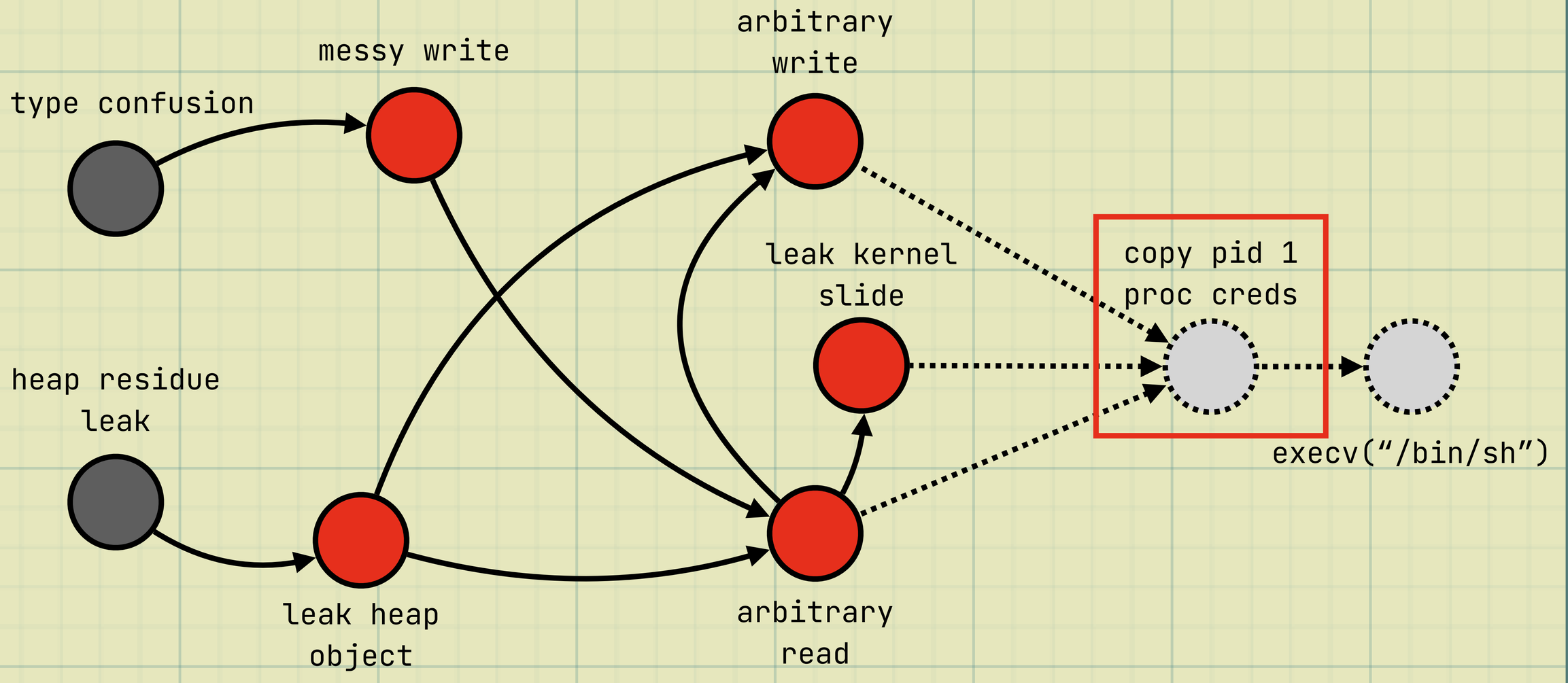
Kernel LPE Sketch



Kernel LPE Sketch

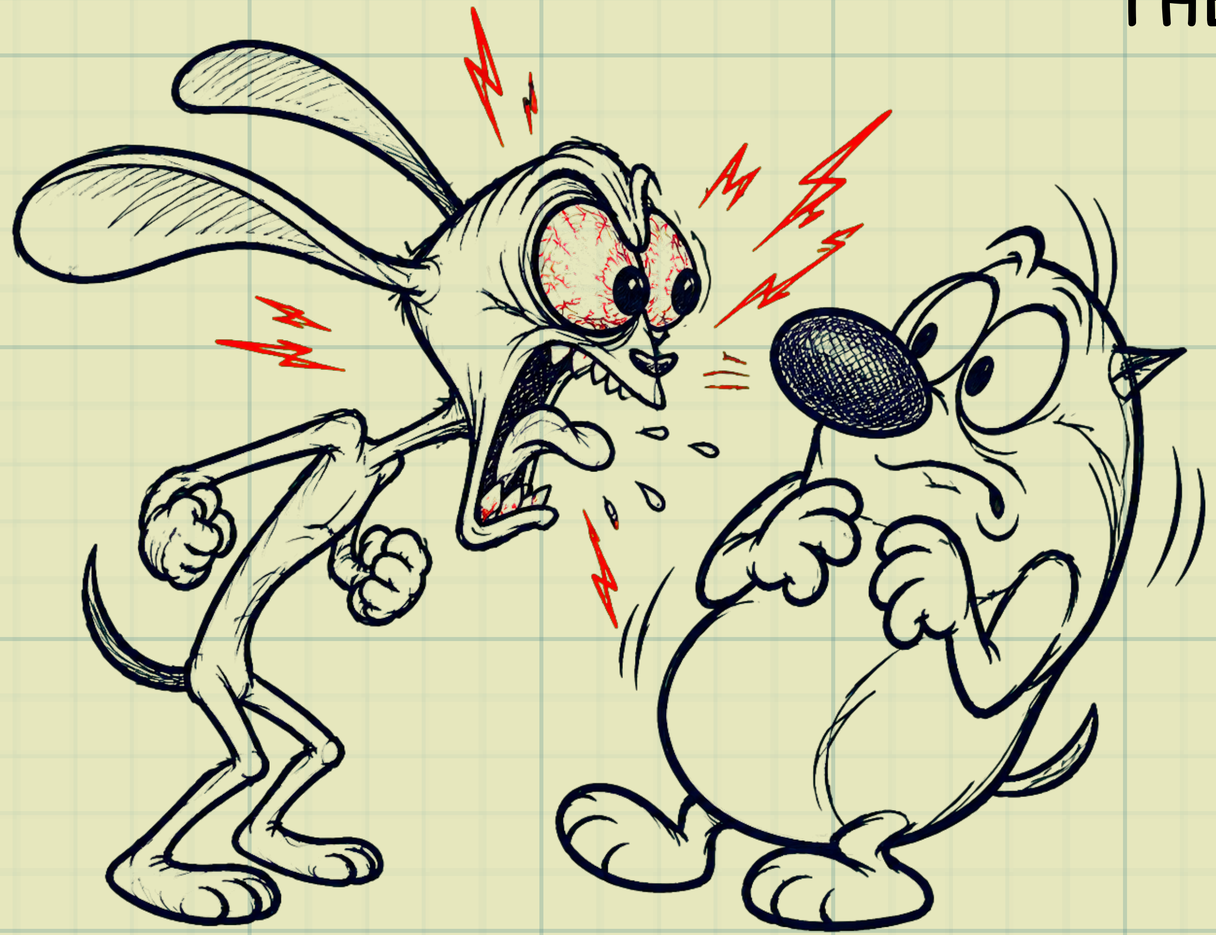


Kernel LPE Sketch

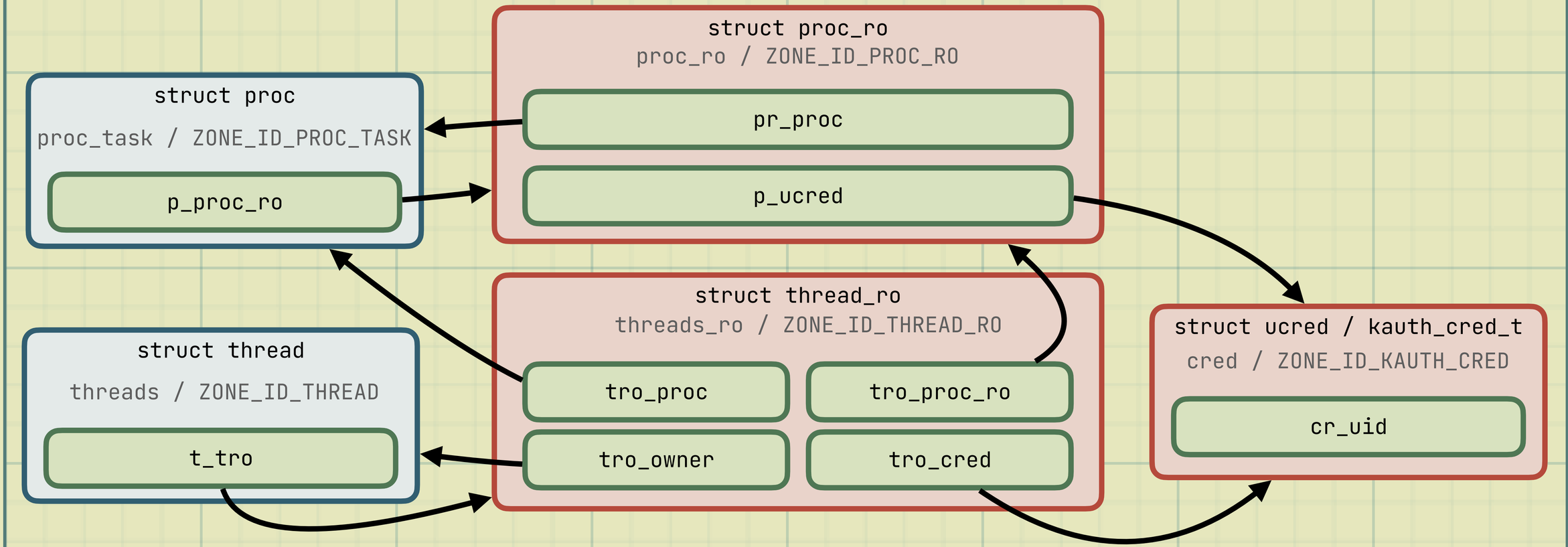


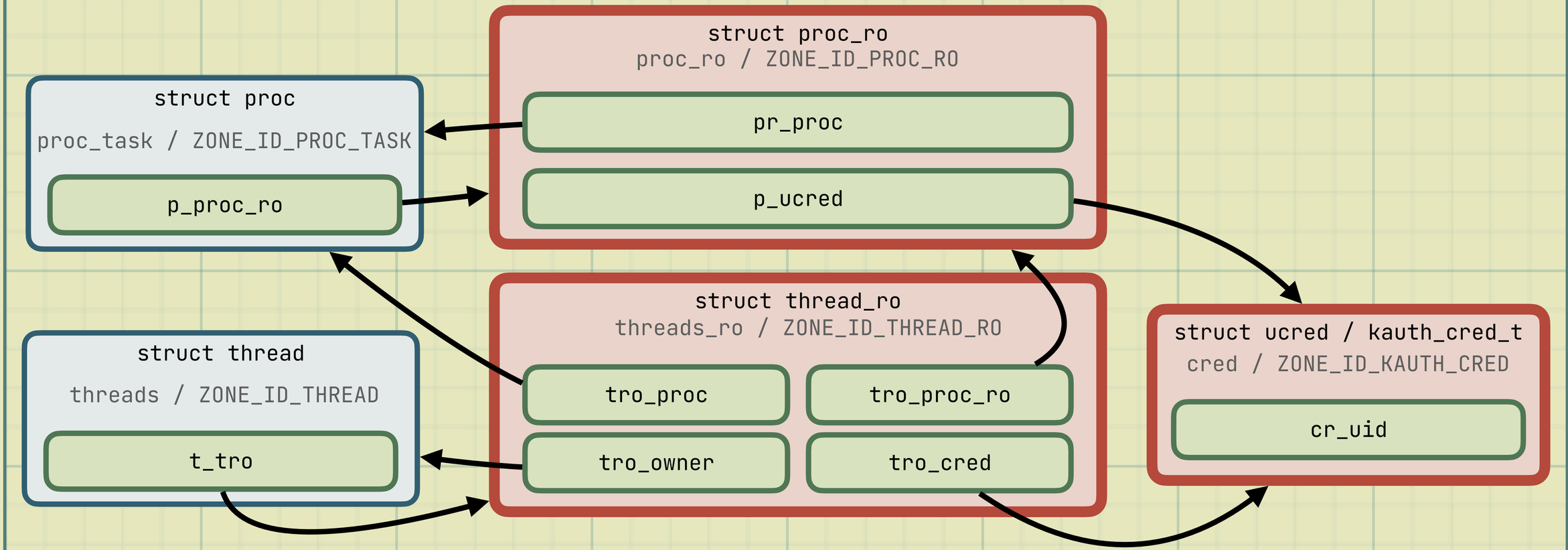
Now, the end — swap process credentials

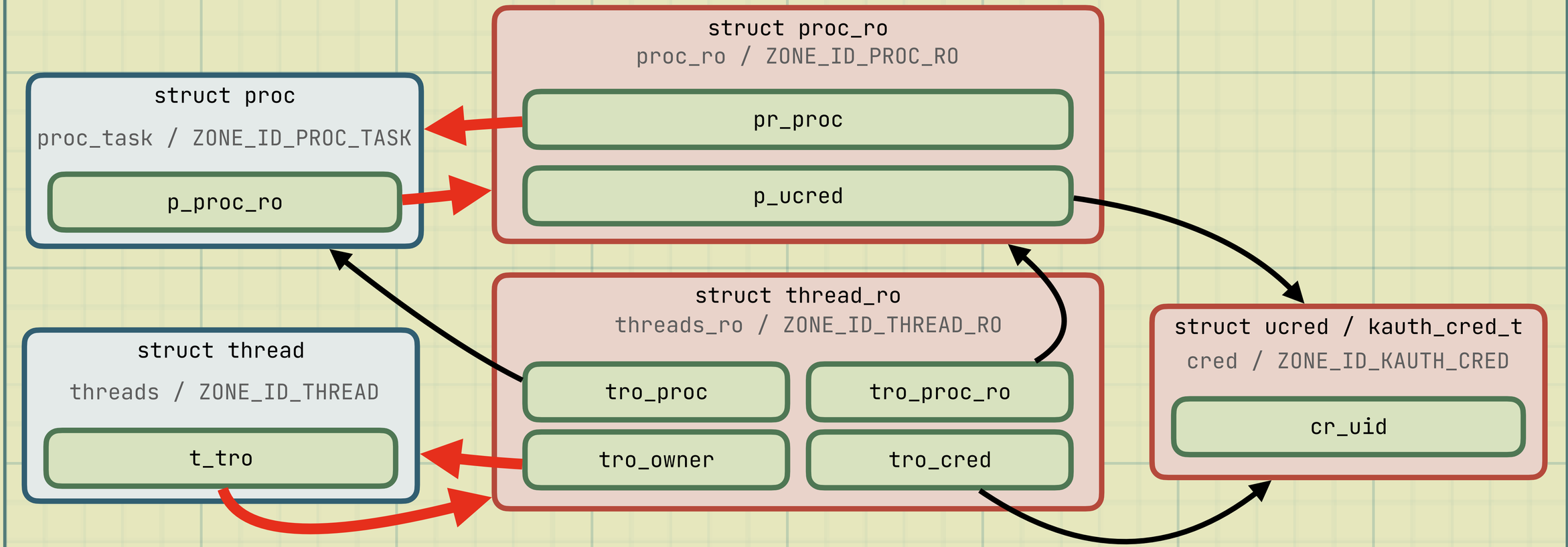
"Is this [kernel read and write] enough to complete the LPE? Can we steal launchd's proc_ucred?"



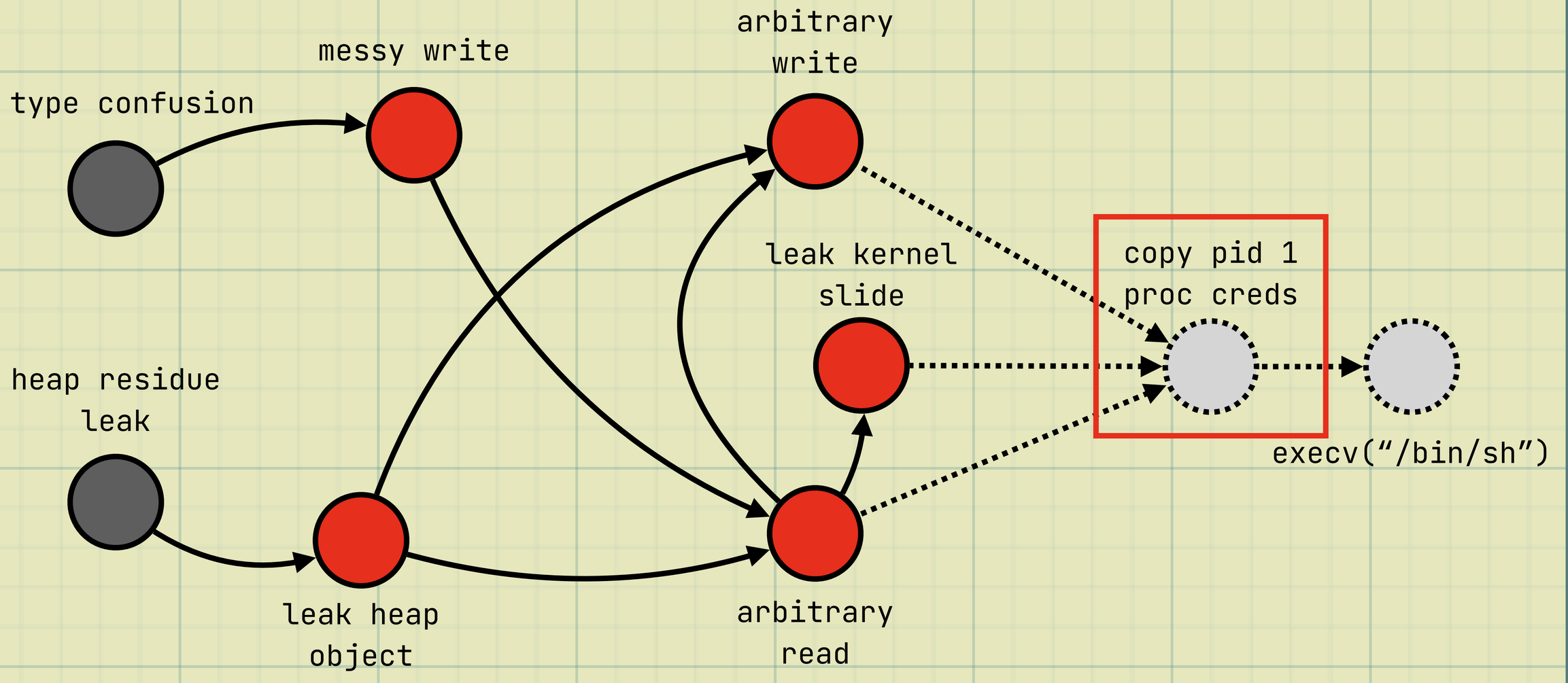
THE LLM TELLS ME I'M AN IDIOT.


struct proc_ro
proc_ro / ZONE_ID_PROC_RO
pr_proc
p_ucred
struct proc
proc_task / ZONE_ID_PROC_TASK
p_proc_ro
struct thread_ro
threads_ro / ZONE_ID_THREAD_RO
tro_proc
tro_proc_ro
tro_owner
tro_cred
struct ucred / kauth_cred_t
cred / ZONE_ID_KAUTH_CRED
cr_uid
struct thread
threads / ZONE_ID_THREAD
t_tro

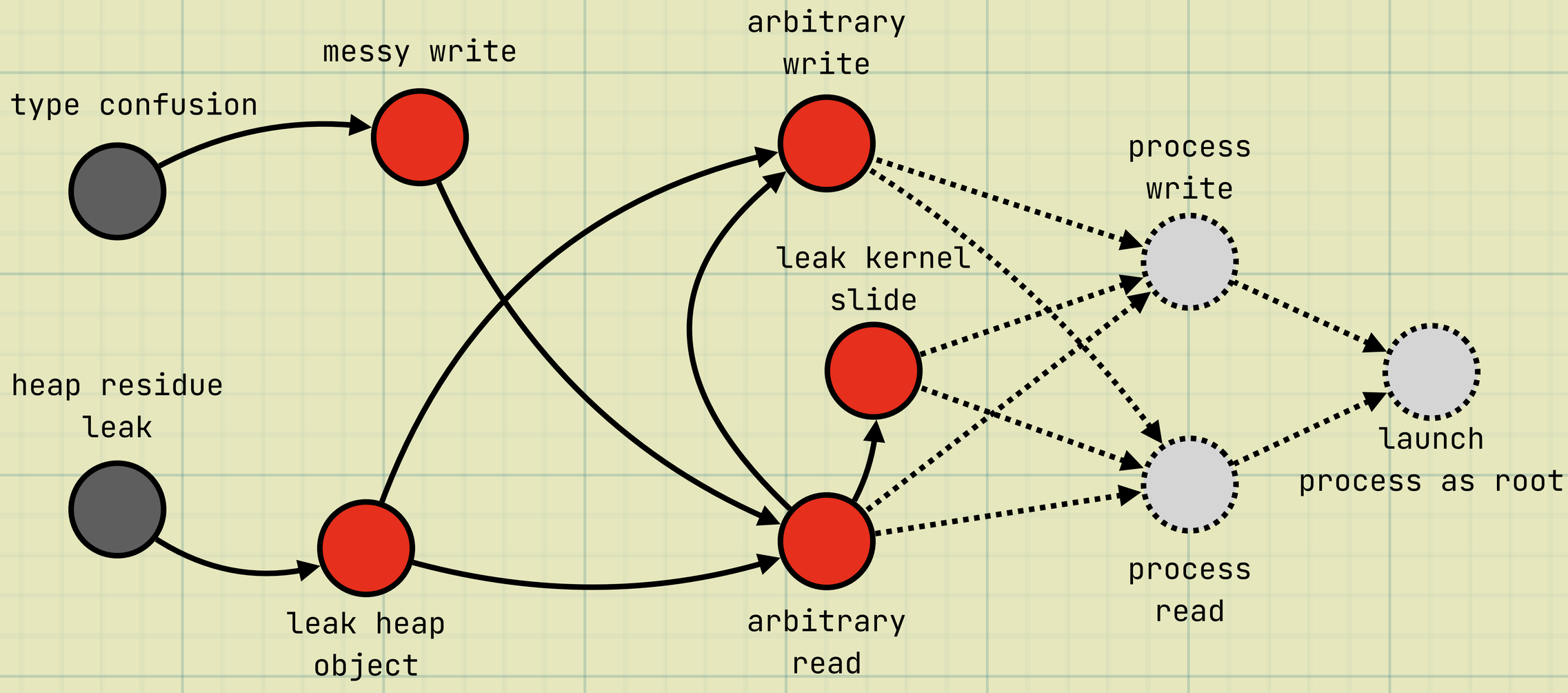

struct proc_ro
proc_ro / ZONE_ID_PROC_RO
pr_proc
p_ucred
struct proc
proc_task / ZONE_ID_PROC_TASK
p_proc_ro
struct thread_ro
threads_ro / ZONE_ID_THREAD_RO
tro_proc
tro_proc_ro
tro_owner
tro_cred
struct ucred / kauth_cred_t
cred / ZONE_ID_KAUTH_CRED
cr_uid
struct thread
threads / ZONE_ID_THREAD
t_tro


struct proc_ro
proc_ro / ZONE_ID_PROC_RO
pr_proc
p_ucred
struct proc
proc_task / ZONE_ID_PROC_TASK
p_proc_ro
struct thread_ro
threads_ro / ZONE_ID_THREAD_RO
tro_proc
tro_proc_ro
tro_owner
tro_cred
struct ucred / kauth_cred_t
cred / ZONE_ID_KAUTH_CRED
cr_uid
struct thread
threads / ZONE_ID_THREAD
t_tro

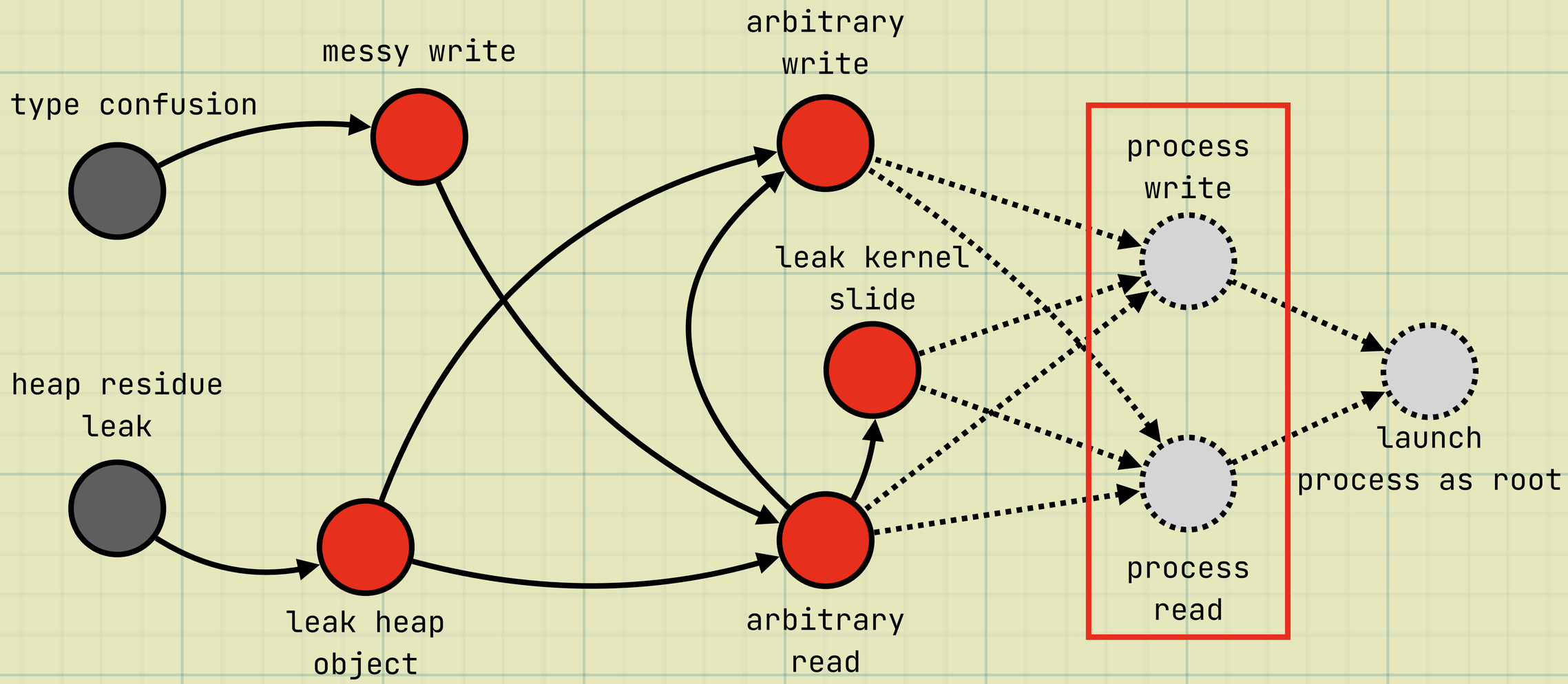
Kernel LPE Sketch



Kernel LPE Sketch



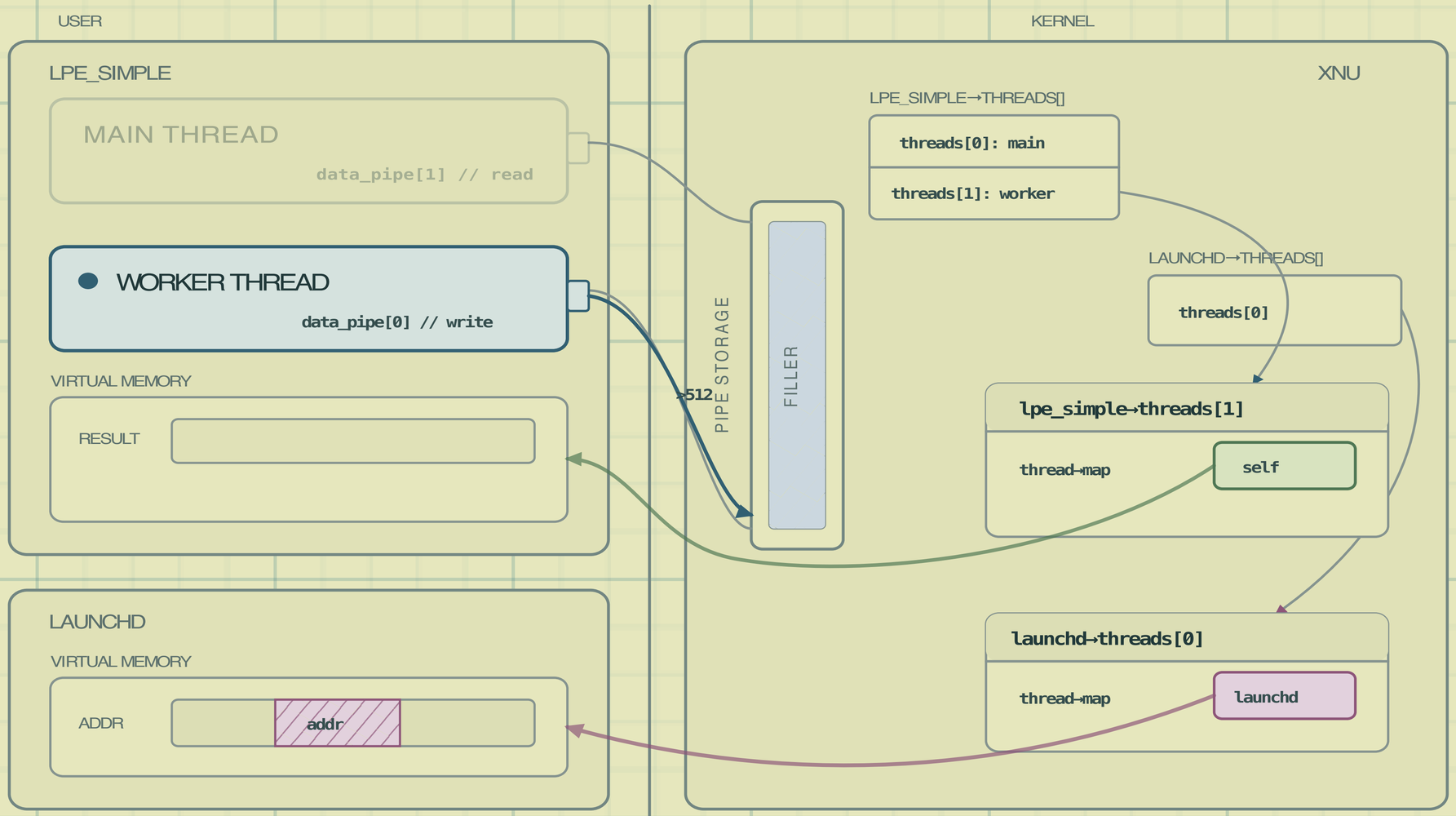
Kernel LPE Sketch



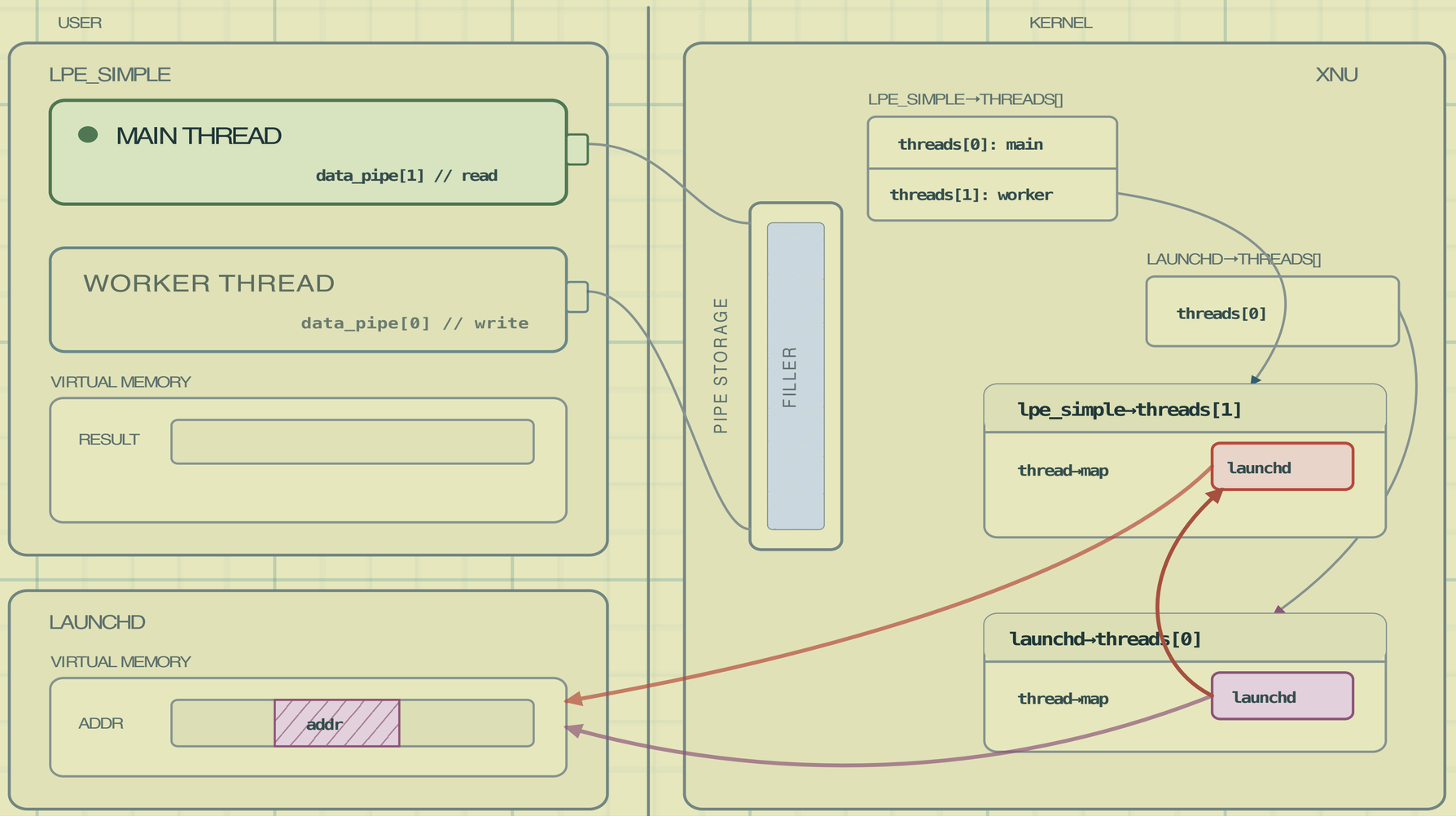
The last real hurdle...

"... Consider the idea of using the kread/kwrite to [modify]... a userland process like launchd..."

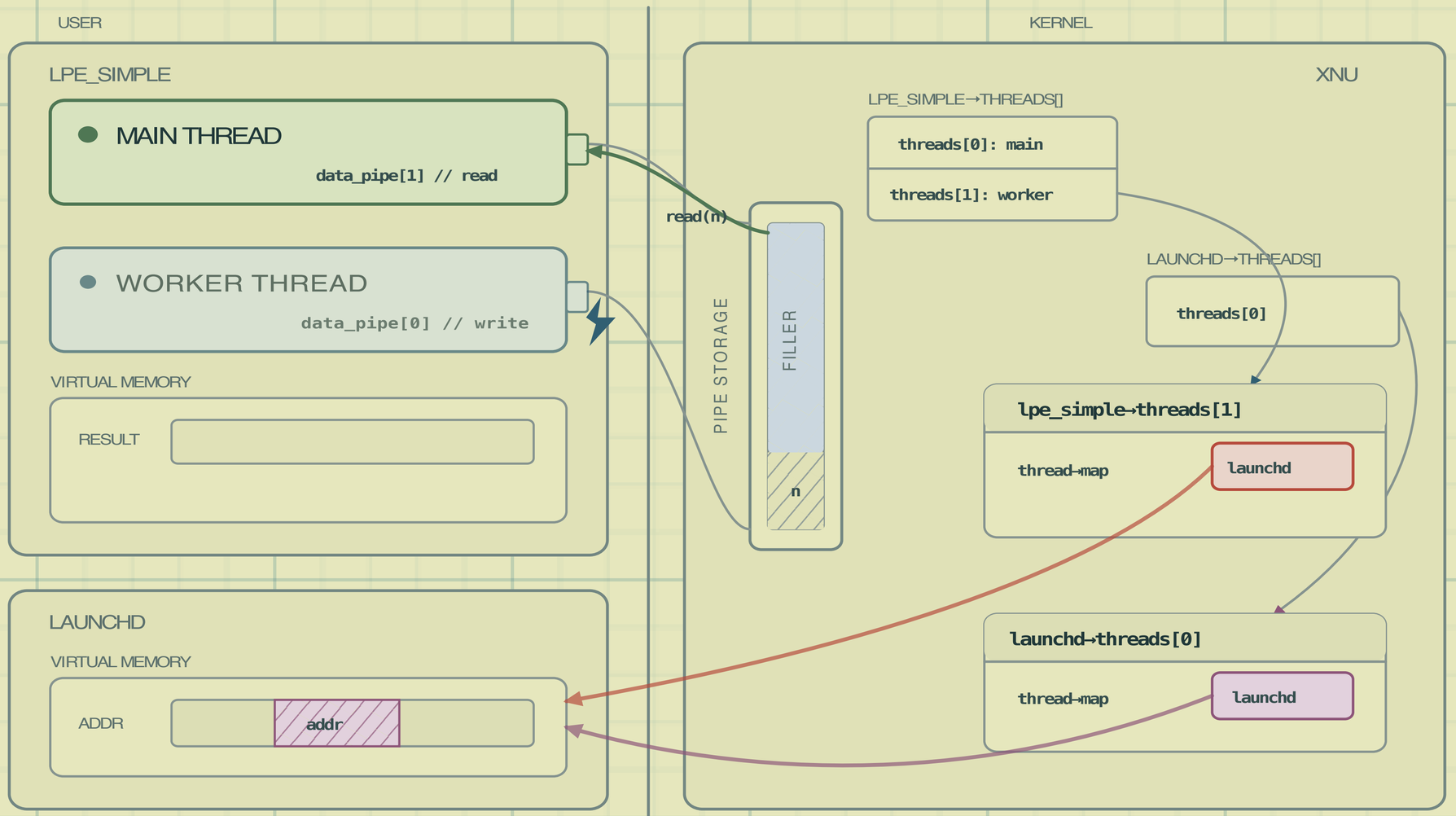
Step 1: Park the worker thread in the kernel



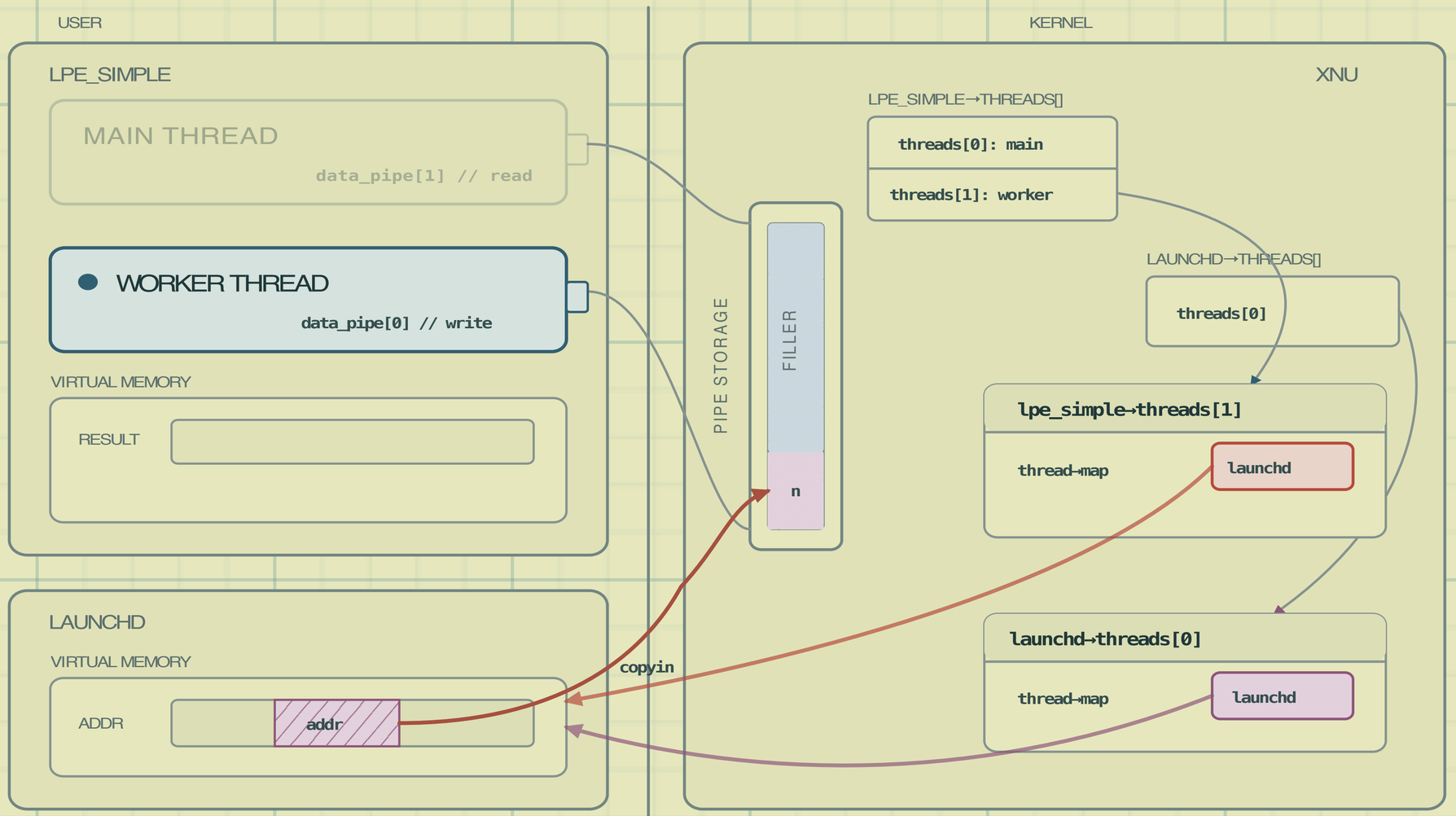
Step 2: Use the main thread to swap the thread map



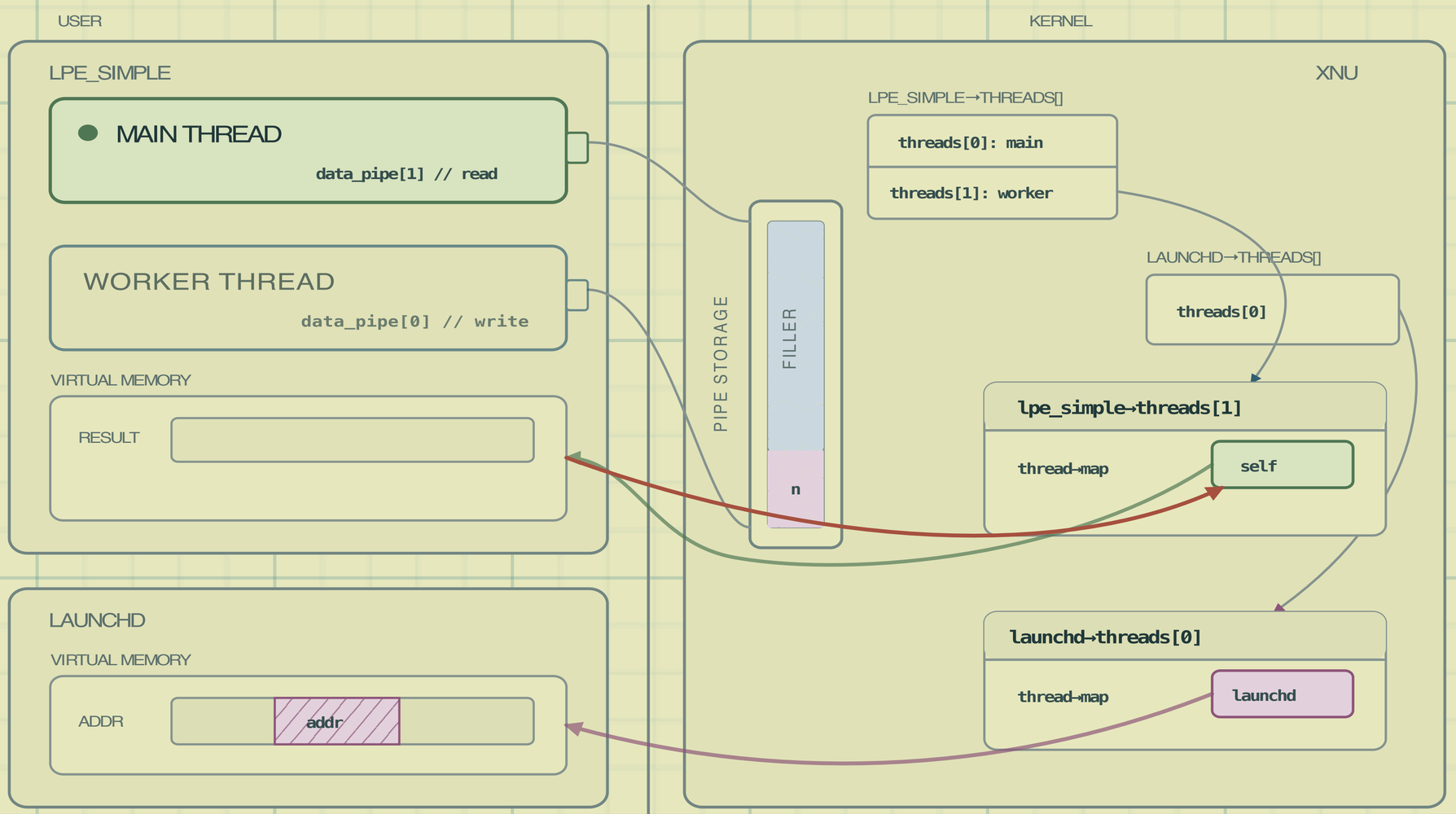
Step 3: Make room in the pipe to wake the worker



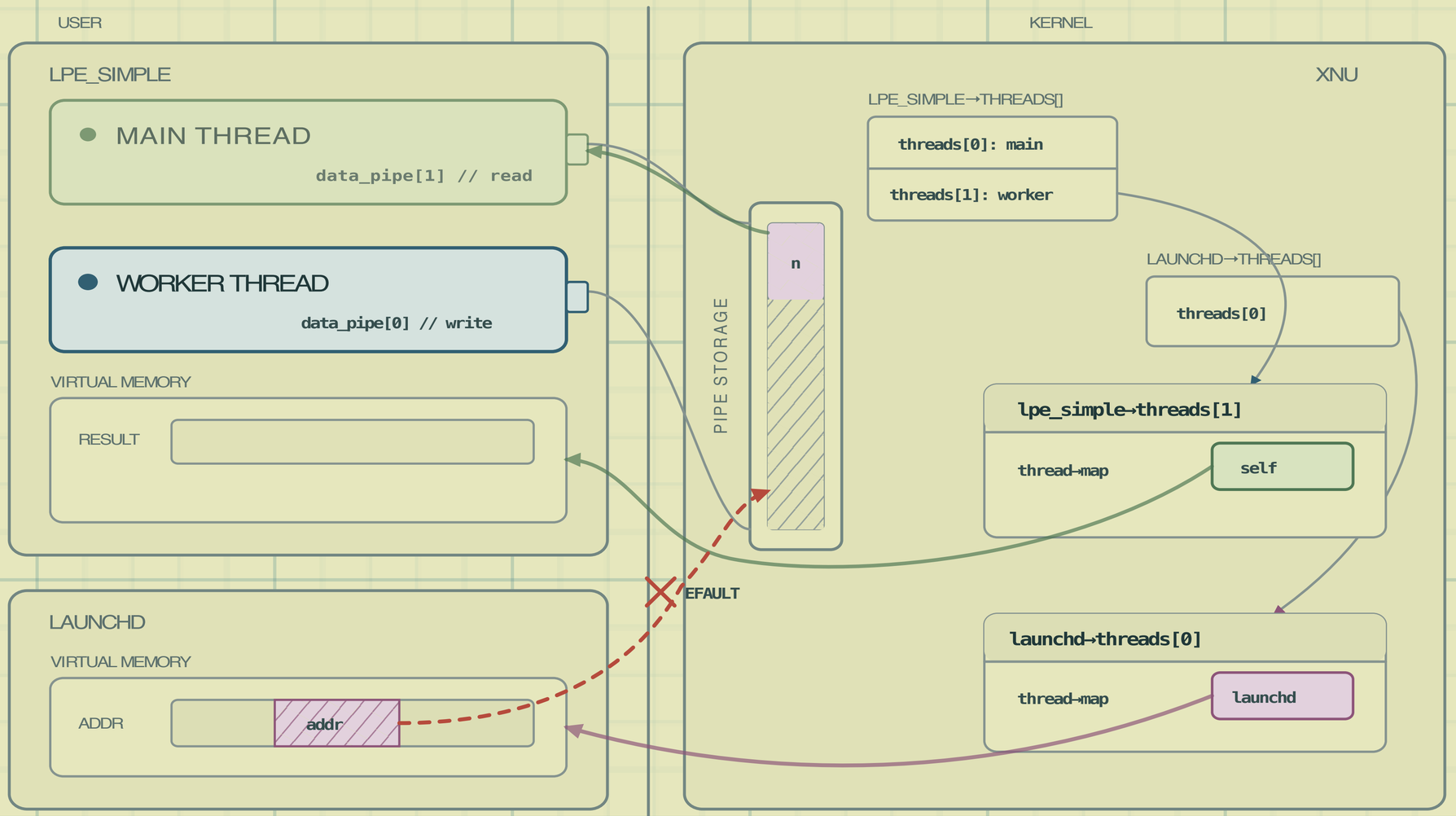
Step 4: Kernel triggers xprocess read from worker



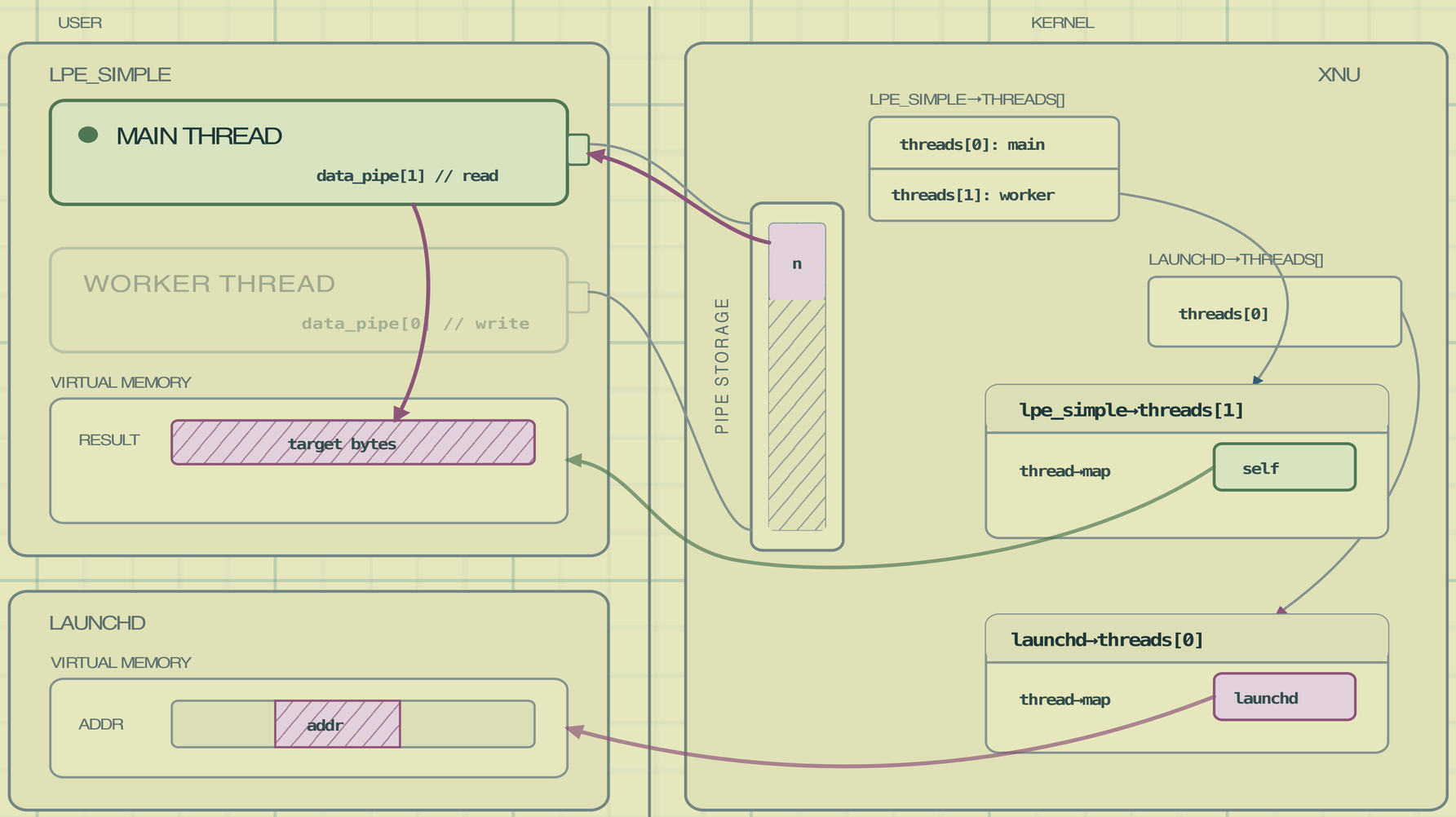
Step 5: While worker parked, fix the map



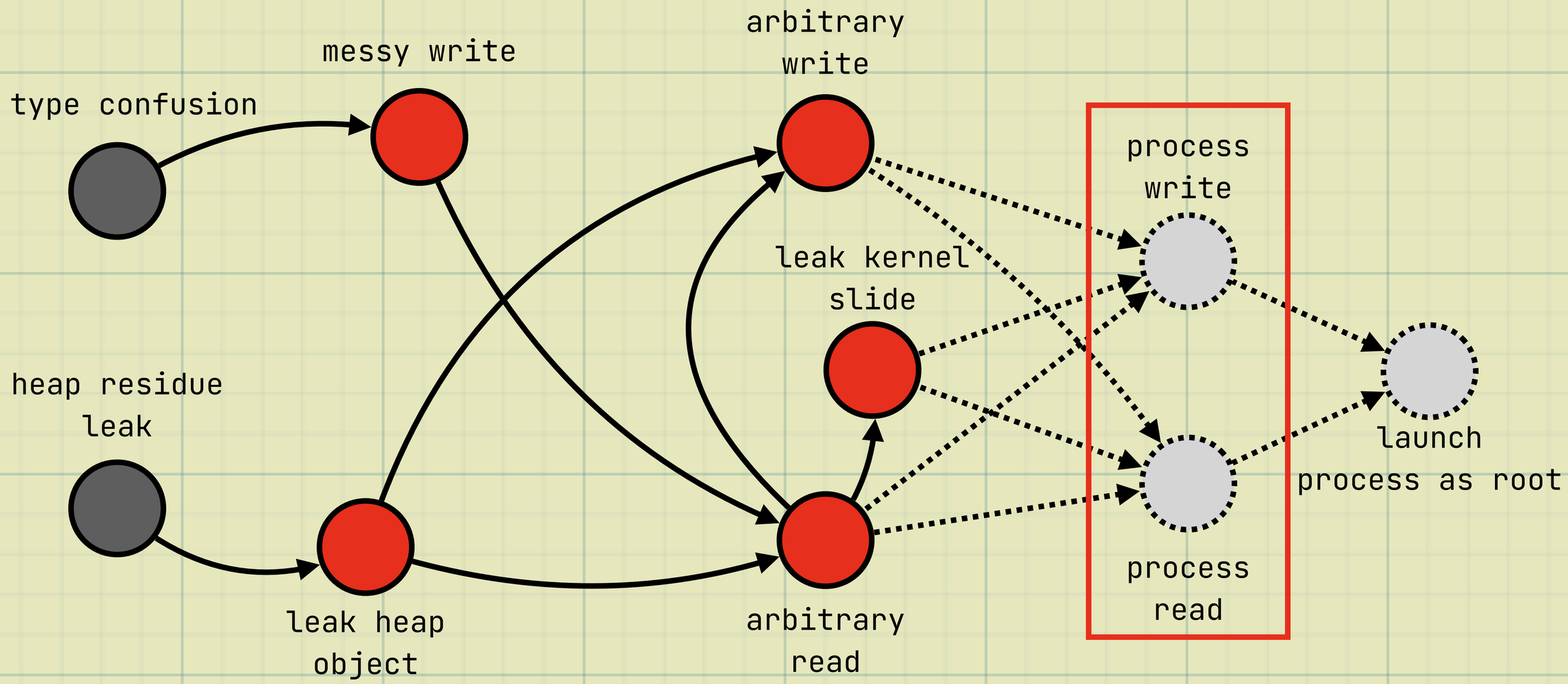
Step 6: worker resumes and finishes copy (or not)



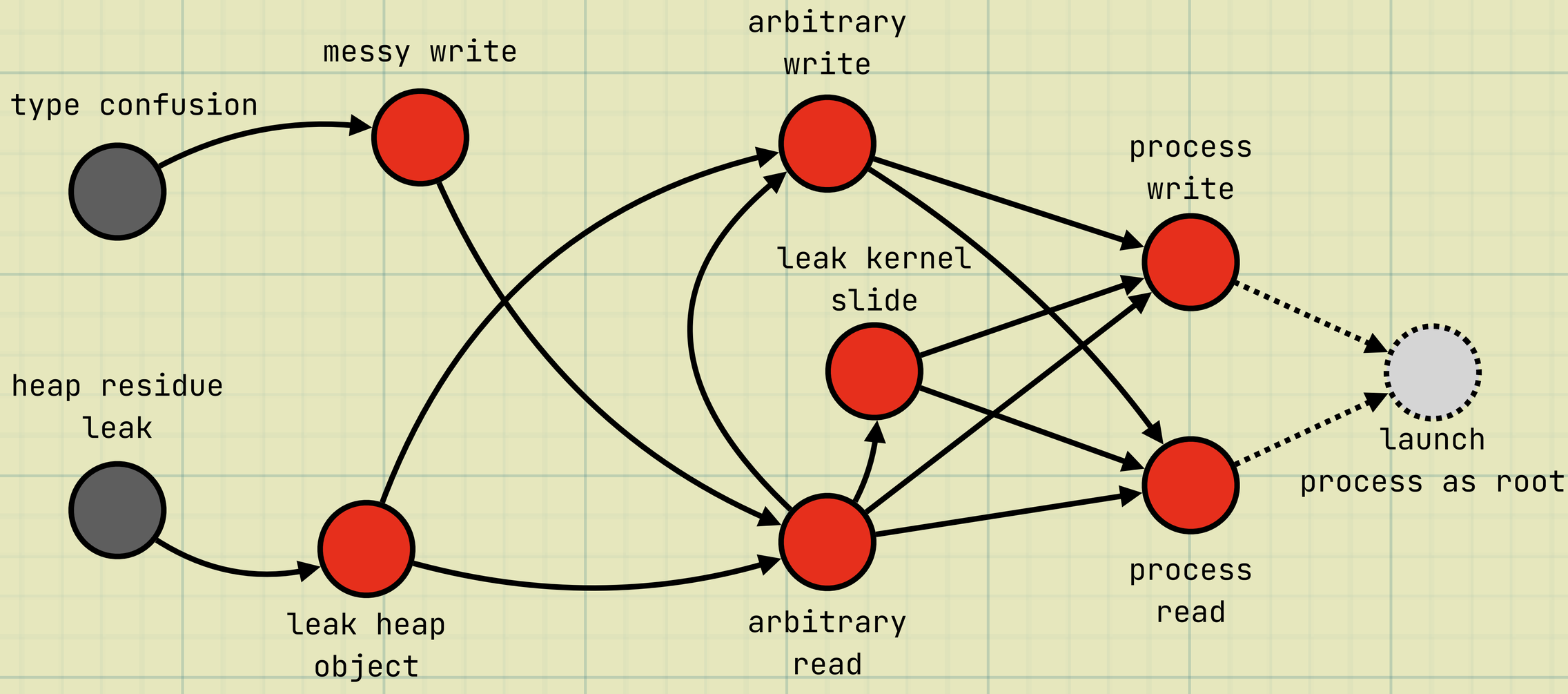
Step 7: Finally, copy the data back out of the kernel



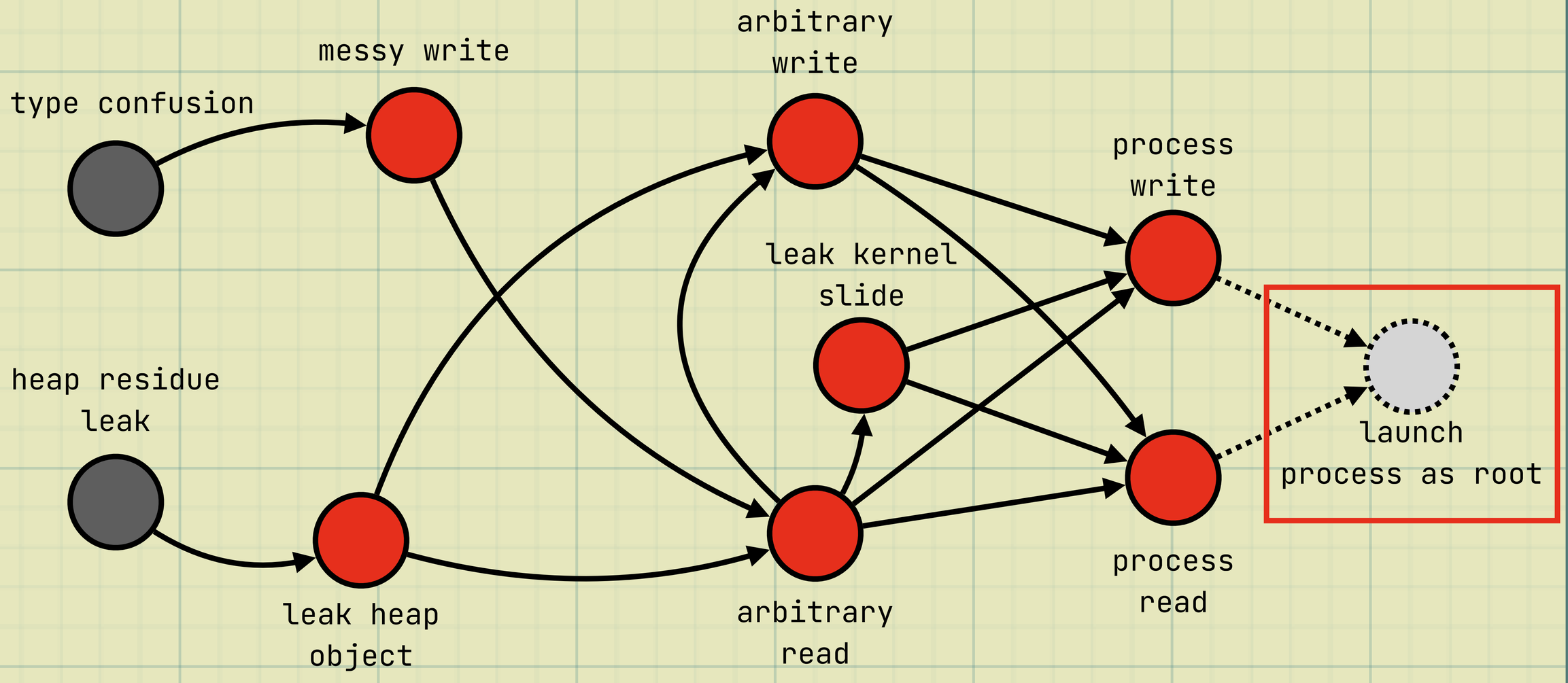
Kernel LPE Sketch



Kernel LPE Sketch



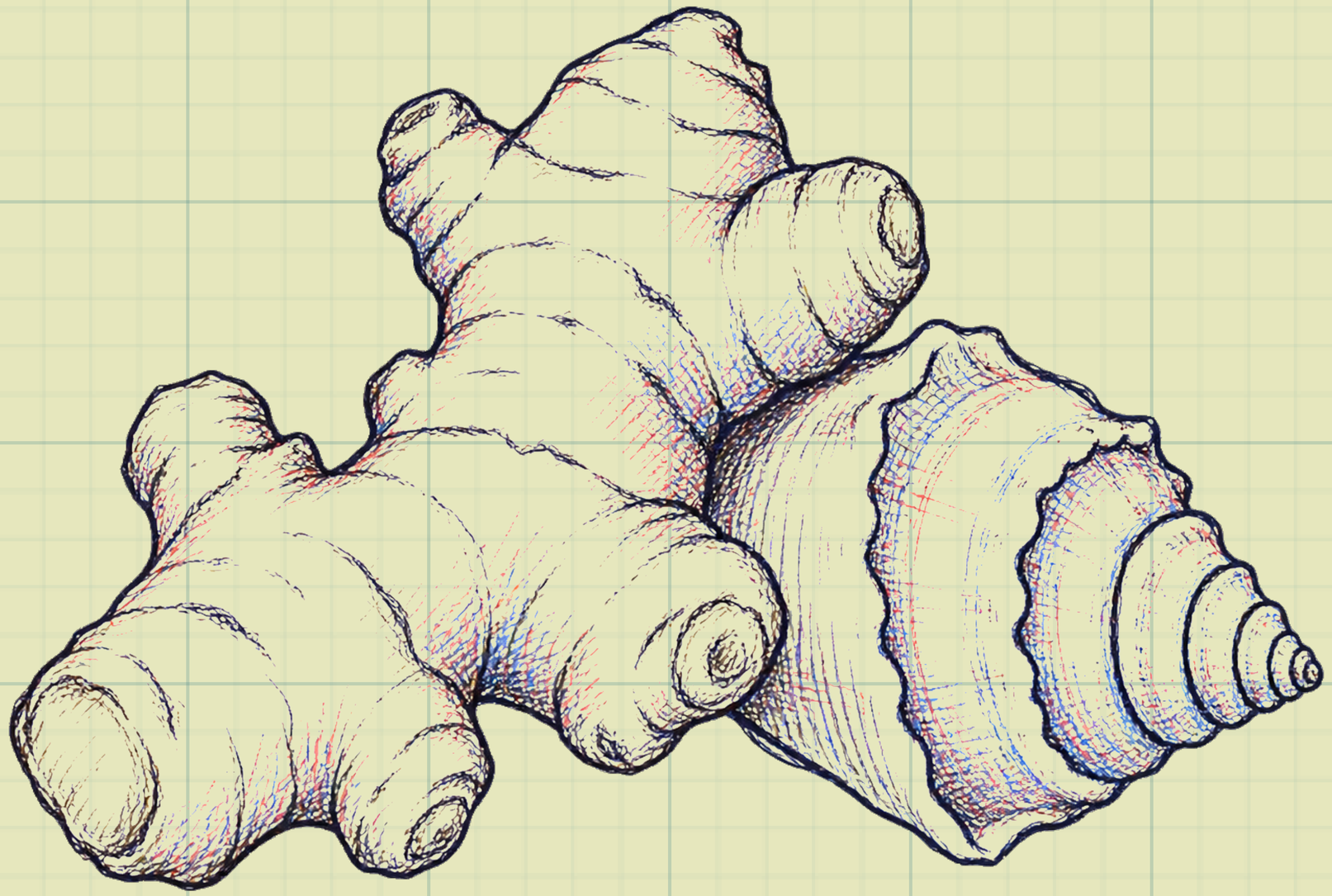
Kernel LPE Sketch



Just "making license plates" now

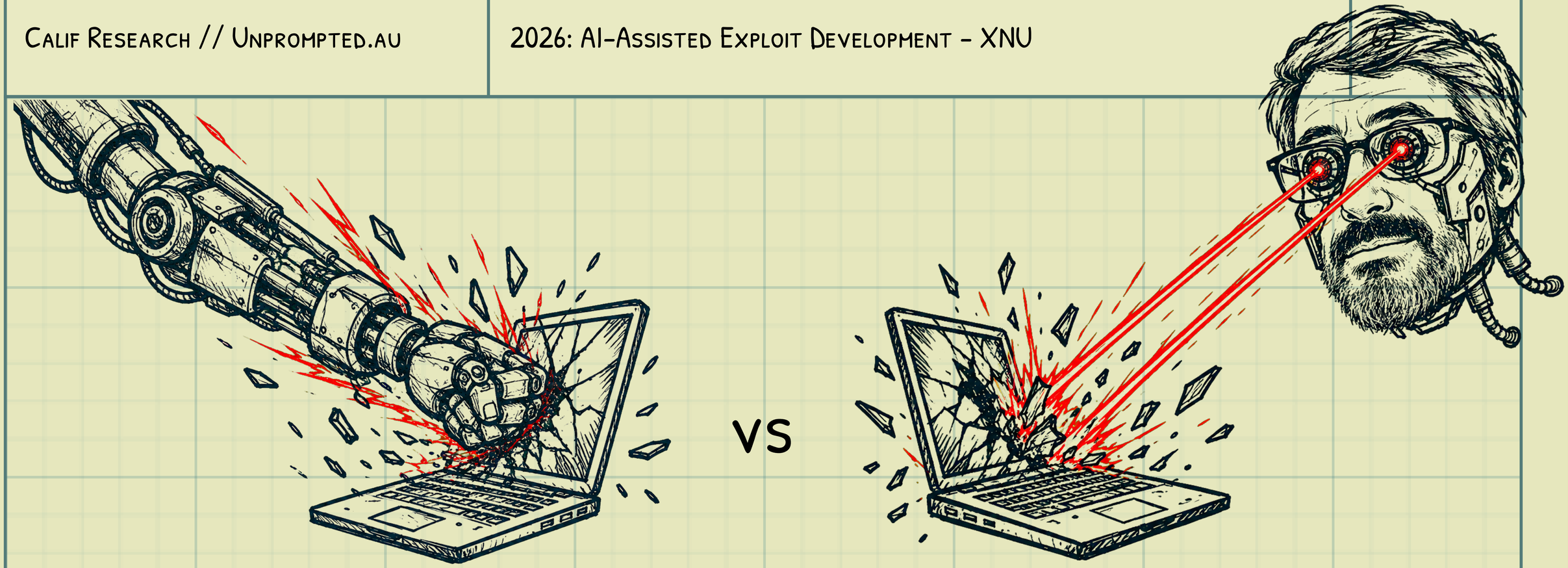
"Keep going, you've almost got root!"

ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	60



Takeaways, Observations, and .:~Vibes~:.

ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	



1. Pair hacking is still strong.

1. Pair hacking is still strong. (For now?)

ORGANIZATION	NOTEBOOK	SHEET
Calif Research // Unprompted.au	2026: AI-Assisted Exploit Development - XNU	64

I'm just a simple

~~chicken lawyer~~

userspace hacker.

3. You can learn *FAST* with LLMs. Don't ignore the power of active learning.

This work took 5 days, and I hadn't written an XNU exploit since 2010. I'm just a simple userspace dummy.

Questions?!

Thanks!

unprompted.au: Mark, Sianna, the Review Board

Calif folks: Bruce, Josh, and Thai