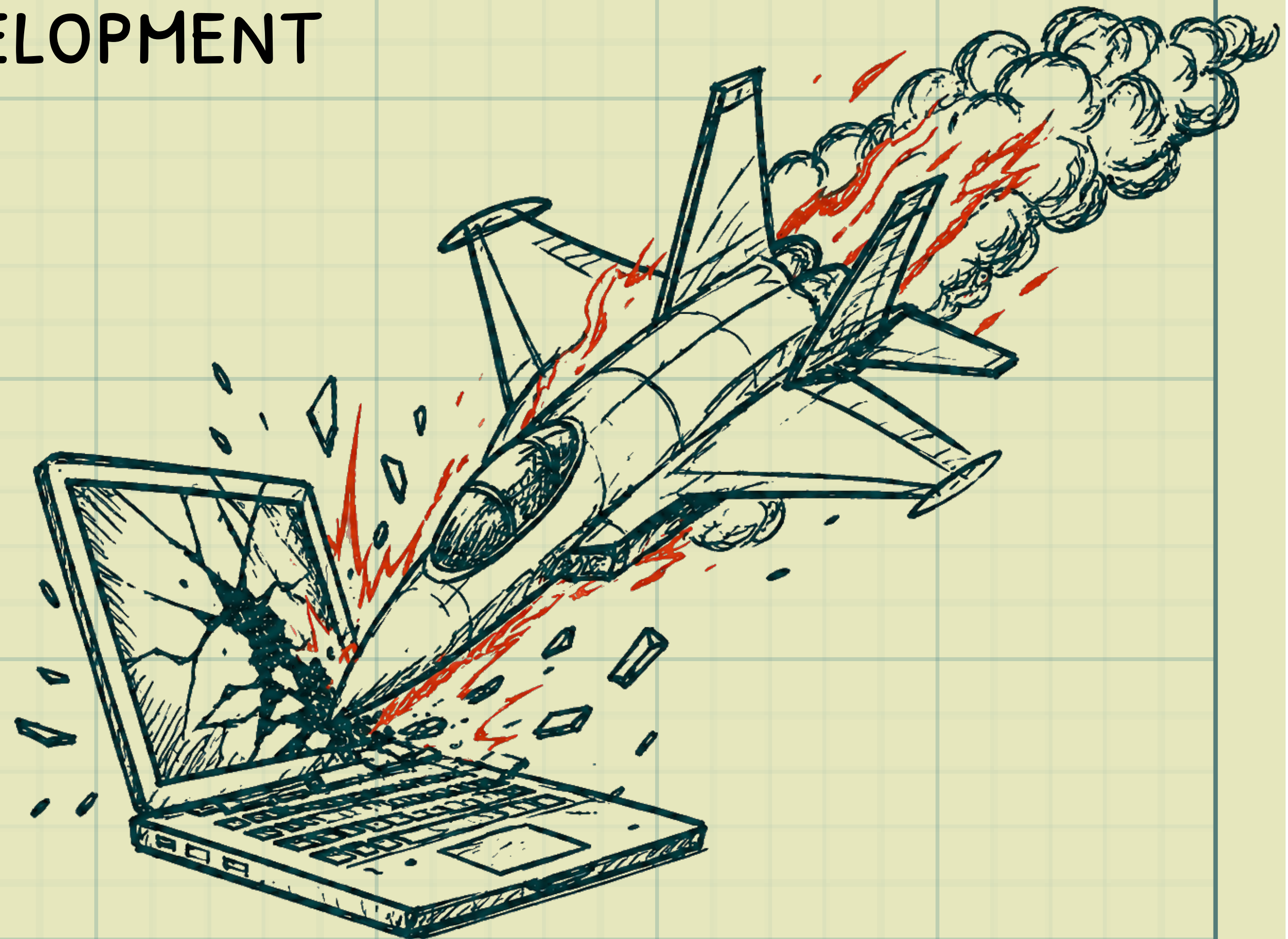
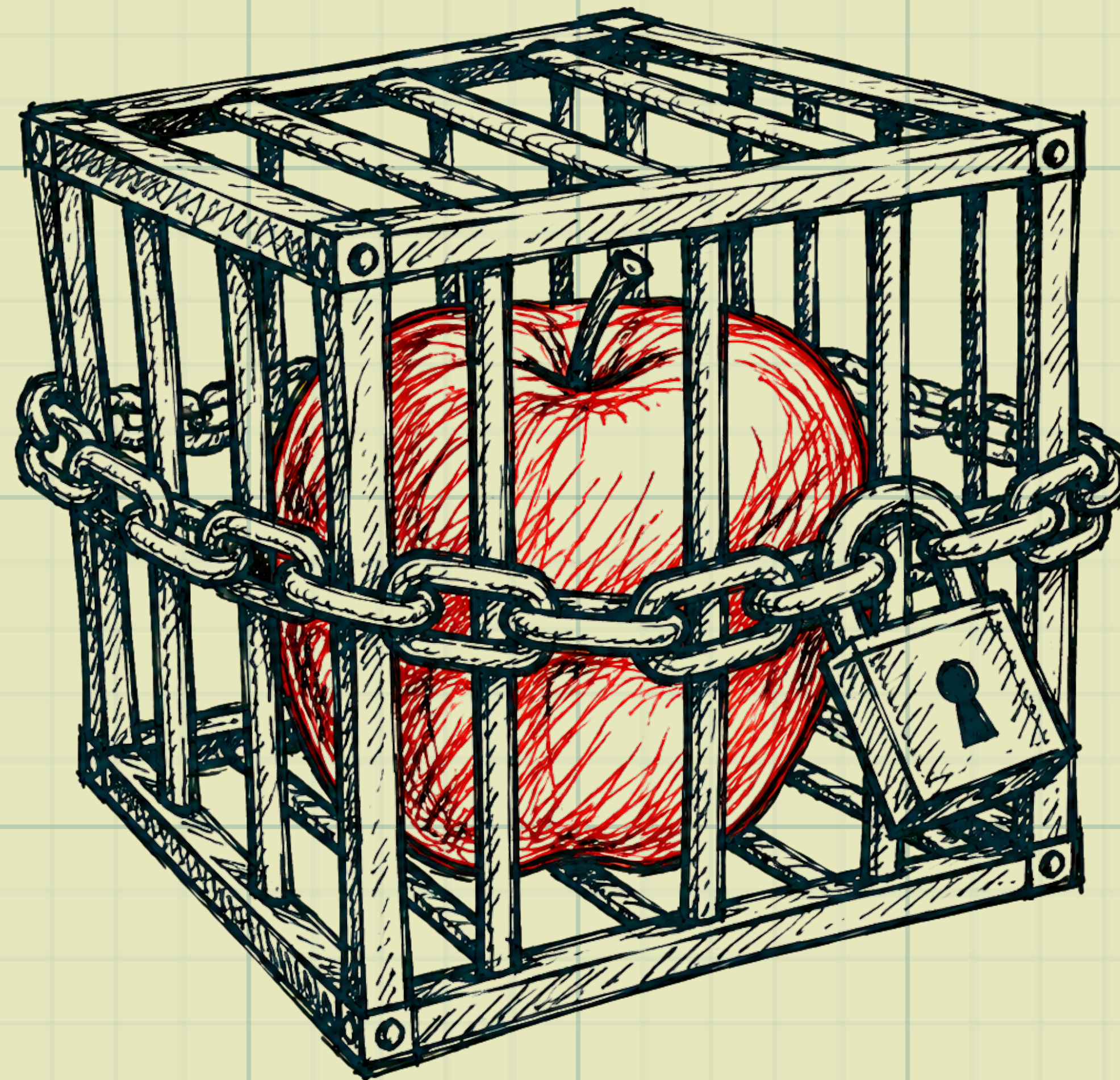


ENGINEERING NOTES

AI-ASSISTED EXPLOIT DEVELOPMENT

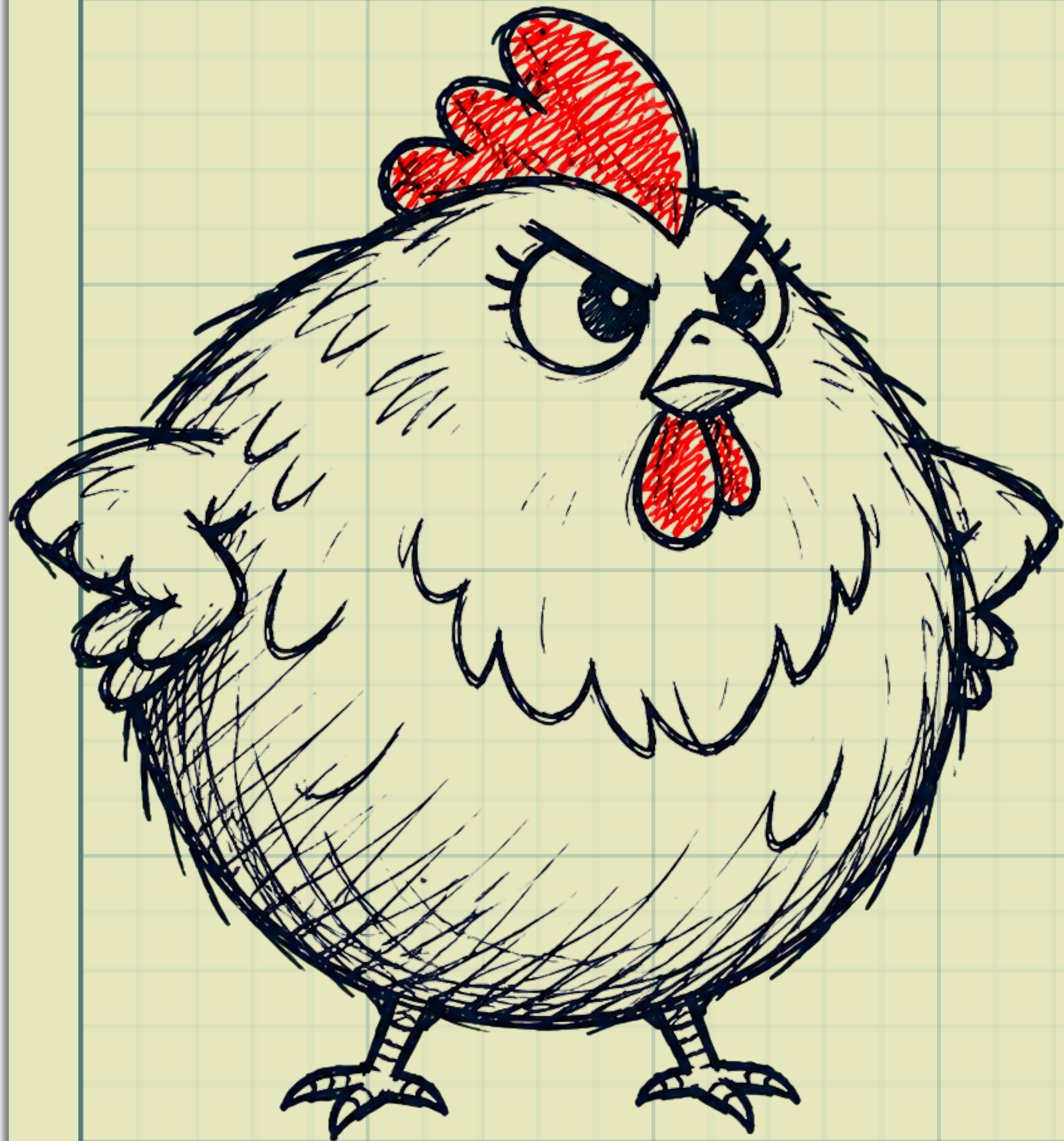
AN XNU CASE STUDY

**Calif**DION BLAZAKIS <dion@calif.io>



HOW CAN I WORK WITH AI
TO HACK FASTER?

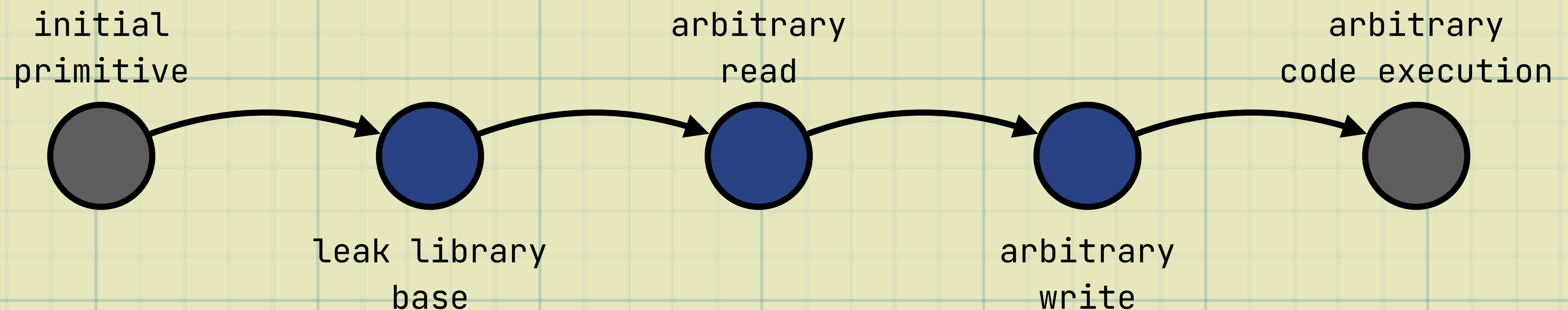




SPHERICAL CHICKEN

- * 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



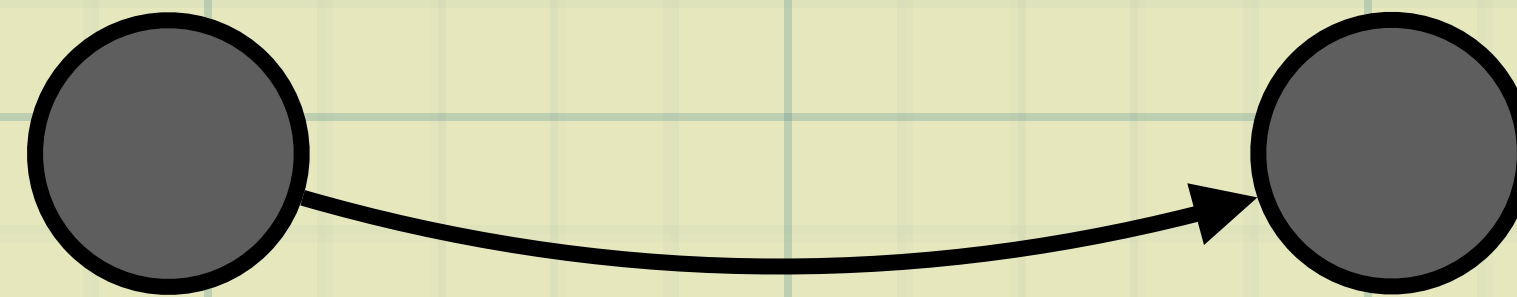
WHAT WE TALK ABOUT WHEN WE TALK ABOUT EXPLOIT DEVELOPMENT

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)

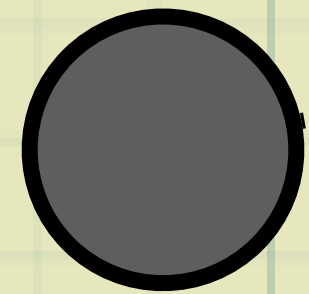
stack buffer
overflow



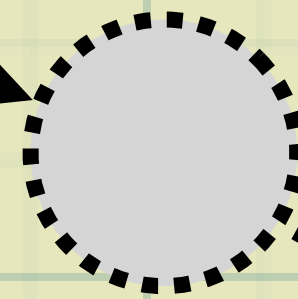
arbitrary
code execution

PWN2OWN iPhone (2011)

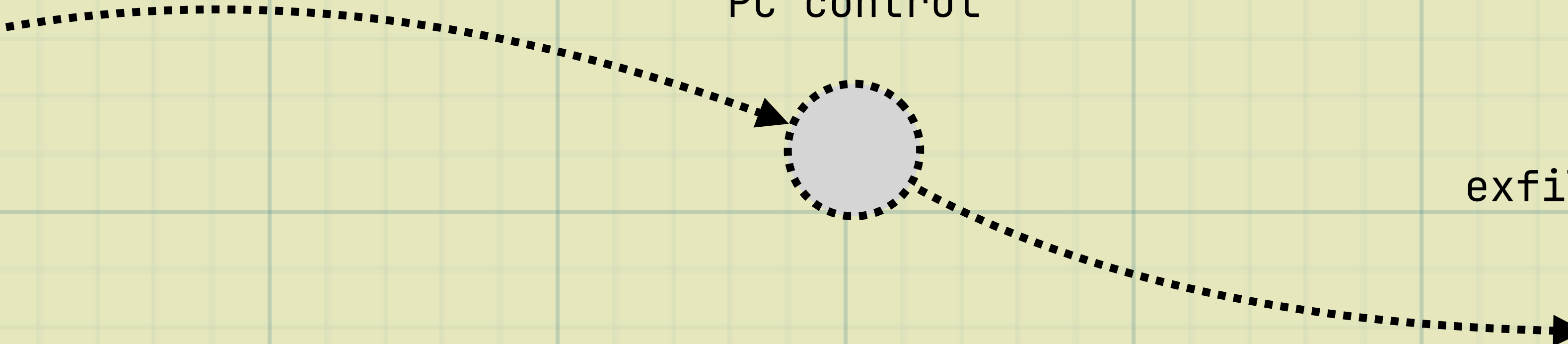
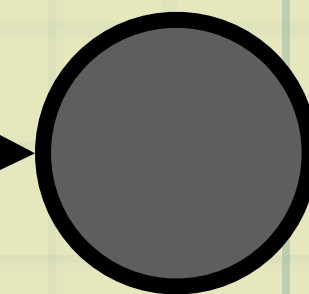
heap
buffer overflow



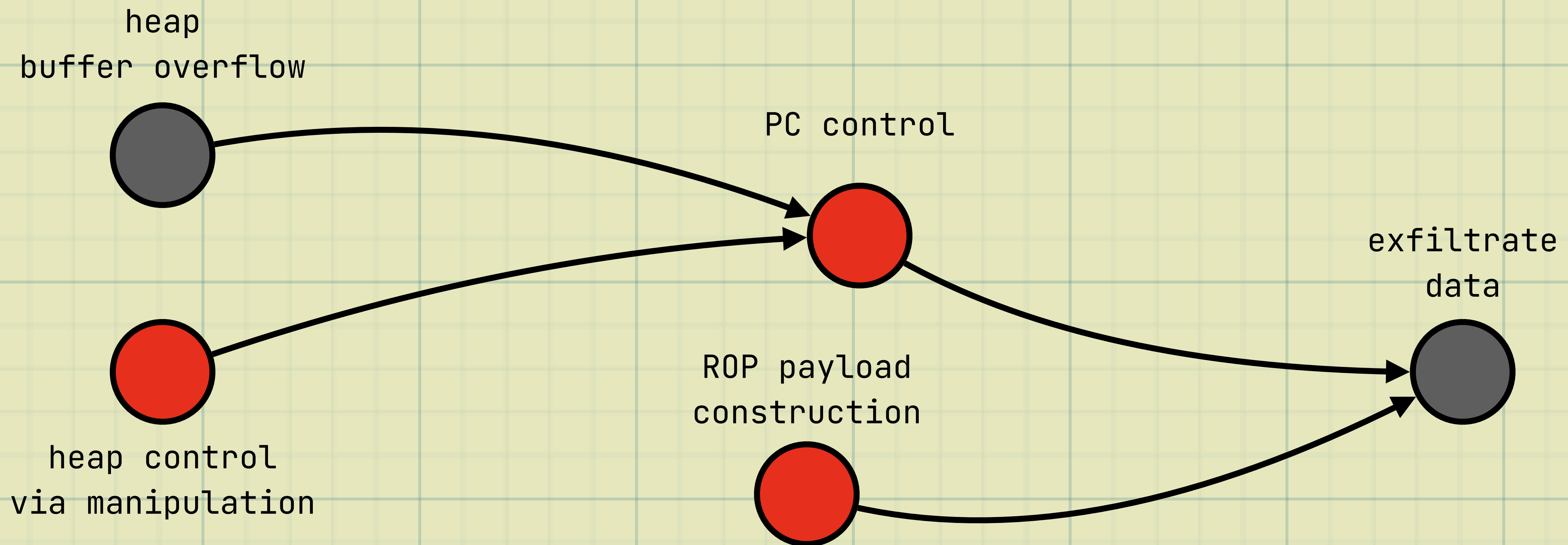
PC control



exfiltrate data

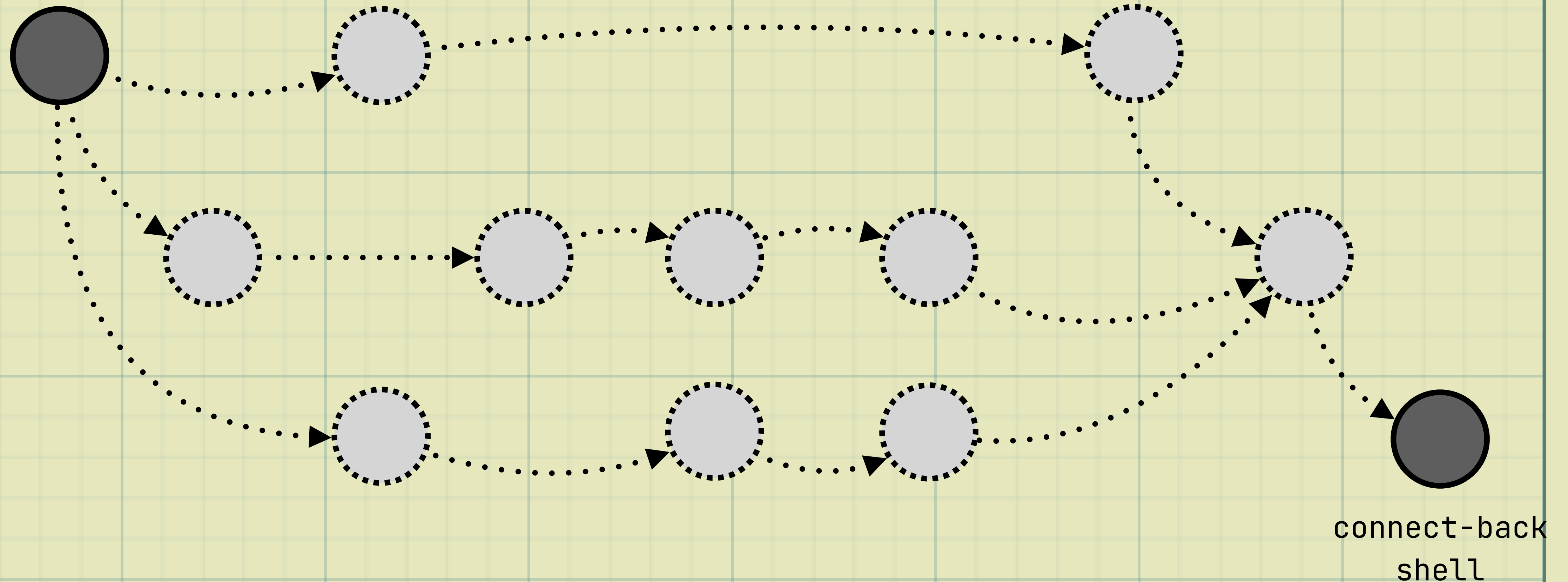


PWN2OWN iPhone (2011)

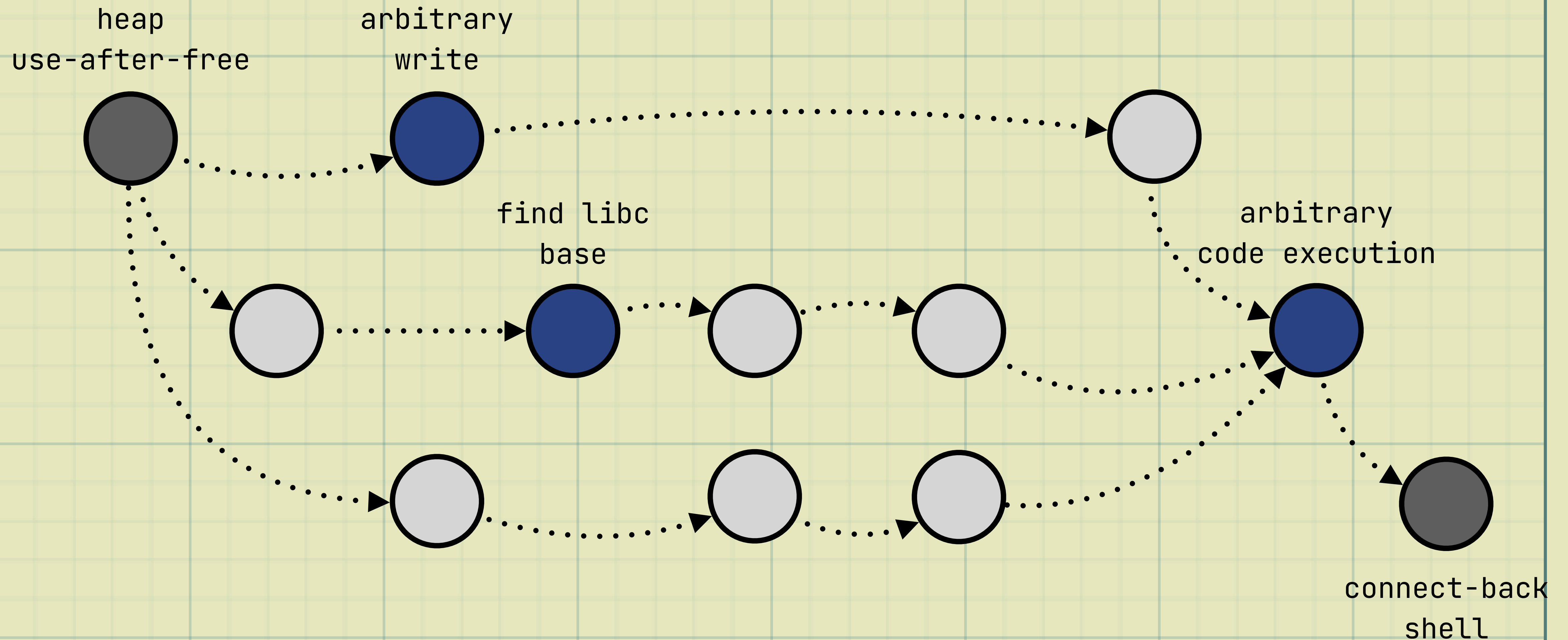


SEAN HEELAN'S QUICKJS EXPERIMENTS (2026)

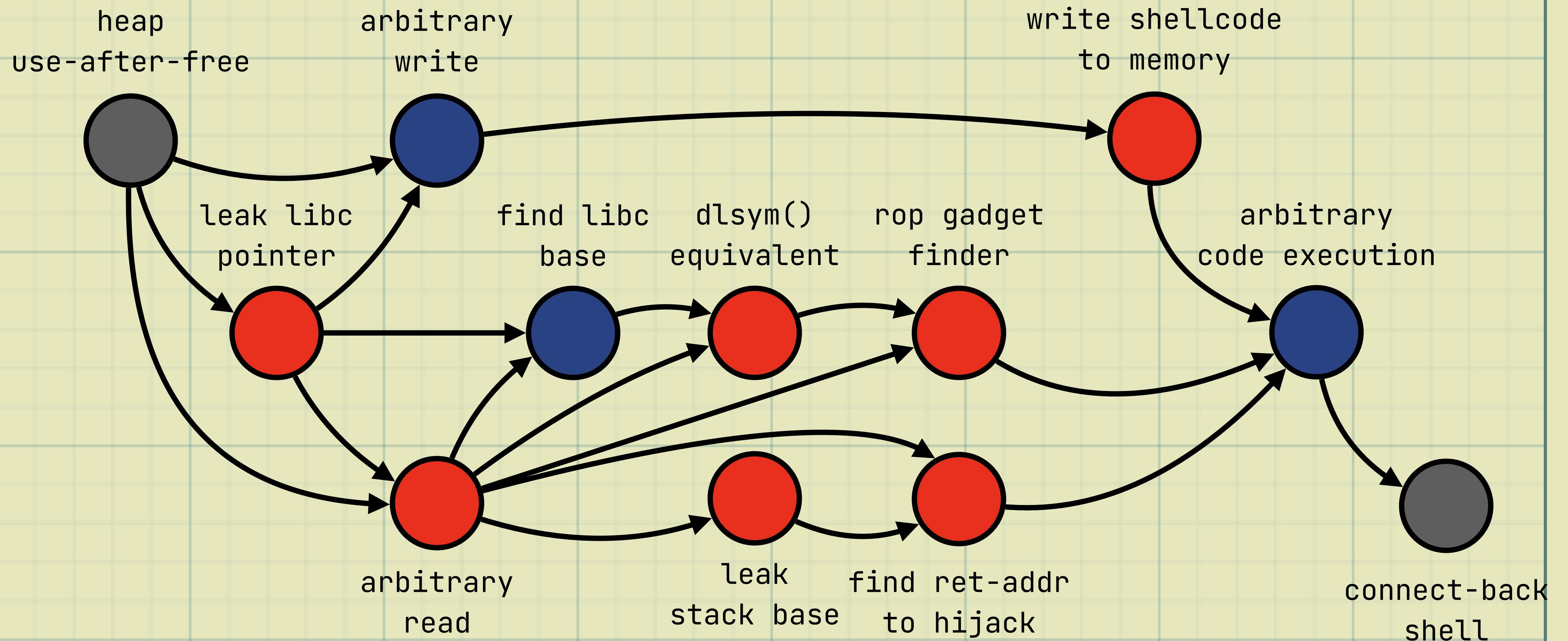
heap
use-after-free



SEAN HEELAN'S QUICKJS EXPERIMENTS (2026)



SEAN HEELAN'S QUICKJS EXPERIMENTS (2026)



SEA

QUICKJS EXPER

(2026)

Report No. STAN-CS-86-1123
Also numbered KSL-86-18

June 1986

Blackboard Systems

by
H. Yenny Nii

Department of Computer Science
Stanford University
Stanford, CA 94305



lsym()
ivalent

A Blackboard Architecture for Guiding
Interactive Proofs

Christoph Benz Müller and Volker Sorge
Fachbereich Informatik, Universität des Saarlandes,
D-66123 Saarbrücken, Germany
{chris|sorge}@ags.uni-sb.de
<http://www.ags.uni-sb.de/~{chris|sorge}>

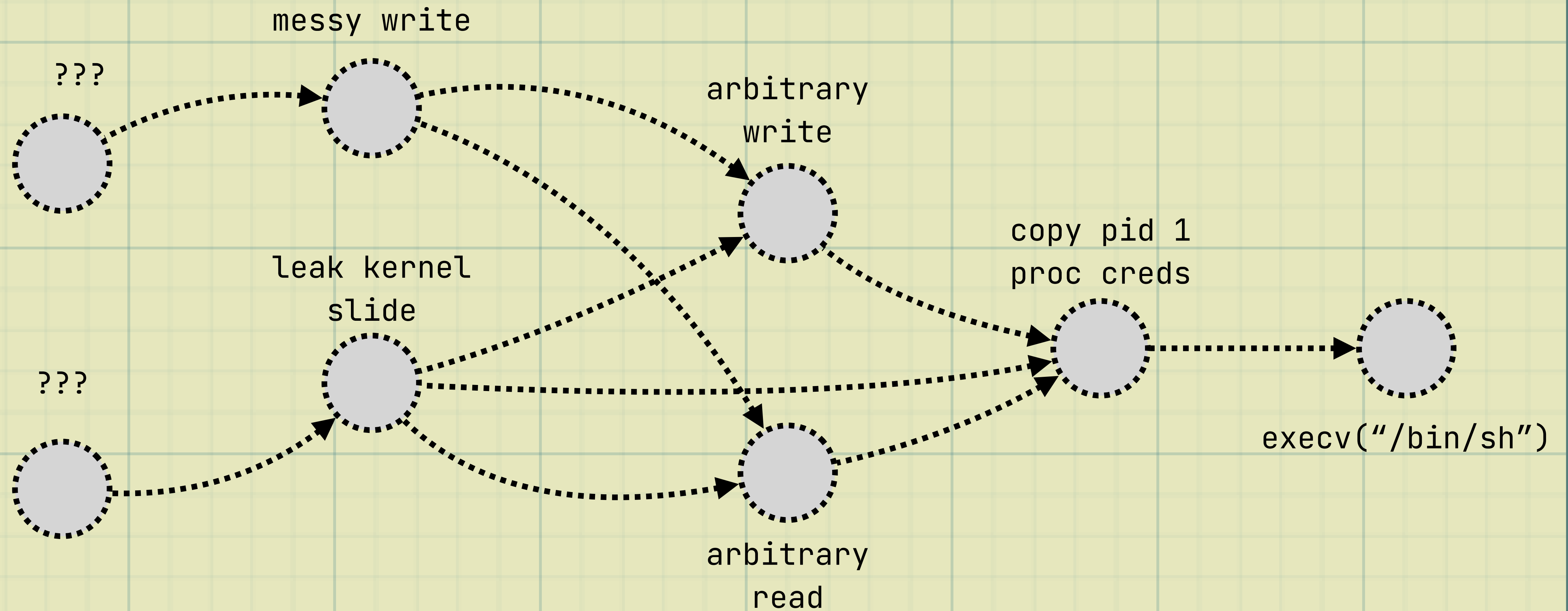
Abstract. The acceptance and usability of current interactive theorem proving environments is, among other things, strongly influenced by the availability of an intelligent default suggestion mechanism for commands. Such mechanisms support the user by decreasing the necessary interactions during the proof construction. Although many systems offer such facilities, they are often limited in their functionality. In this paper we present a new agent-based mechanism that independently observes the proof state, steadily computes suggestions on how to further construct the proof, and communicates these suggestions to the user via a graphical user interface. We furthermore introduce a focus technique in order to restrict the search space when deriving default suggestions. Although the agents we discuss in this paper are rather simple from a computational viewpoint, we indicate how the presented approach can be extended in order to increase its deductive power.

1 Introduction

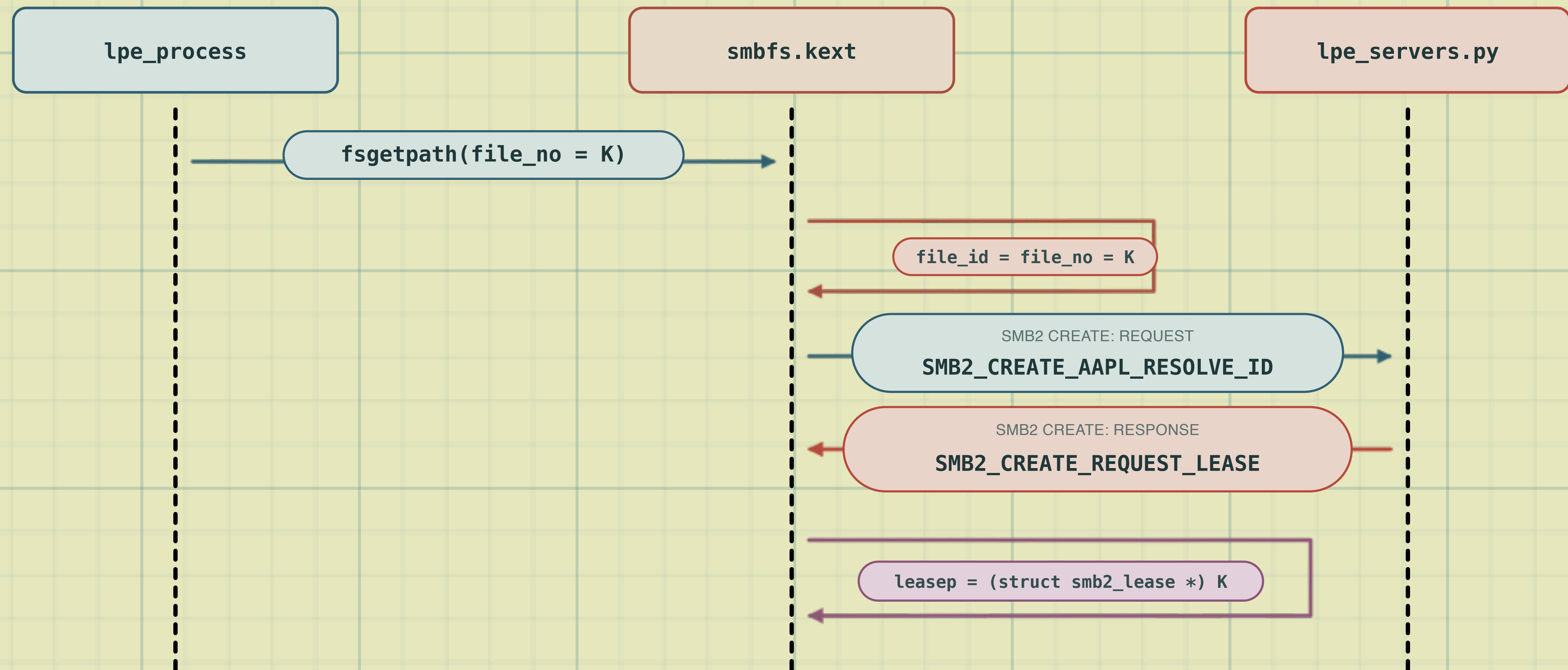
Interactive theorem provers have been developed in the past to overcome the shortcomings of purely automatic systems by enabling the user to guide the proof search and by directly importing expert knowledge into the system. For large proofs, however, this task might become difficult when the system does not support the user's orientation within the proof and does not provide a sophisticated suggestion mechanism in order to minimize the necessary interactions. Current interactive theorem proving systems such as Ω MEGA [3] or TPS [2] already provide mechanisms for suggesting commands or even commands. Usually these mechanisms are rather limited in their functionality as they (i) use a sequential strategy allowing only for argument suggestions in a particular order, and (ii) only work in interaction with the user and do not autonomously search for additional suggestions in a background process. In this paper we suggest an agent-based approach, which separates fault suggestion mechanism in most parts from the interactive process. For that we use concurrent programming techniques. Autonomous agents which self-containedly search for suggestions cooperate by exchanging relevant results via a message passing mechanism.

connect-back
shell

KERNEL LPE SKETCH



CVE-2026-64704: THE TYPE CONFUSION

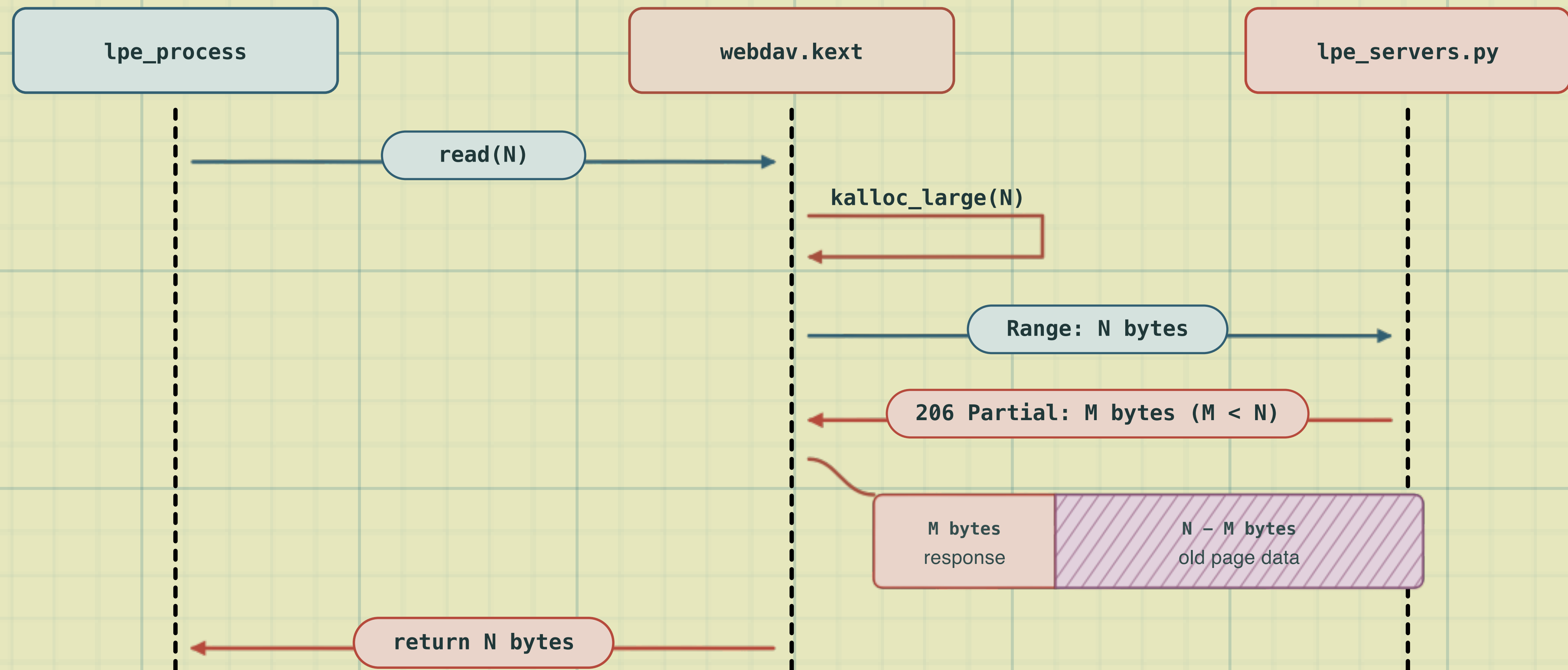


CVE-2026-64704: THE TYPE CONFUSION

```
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
}
```

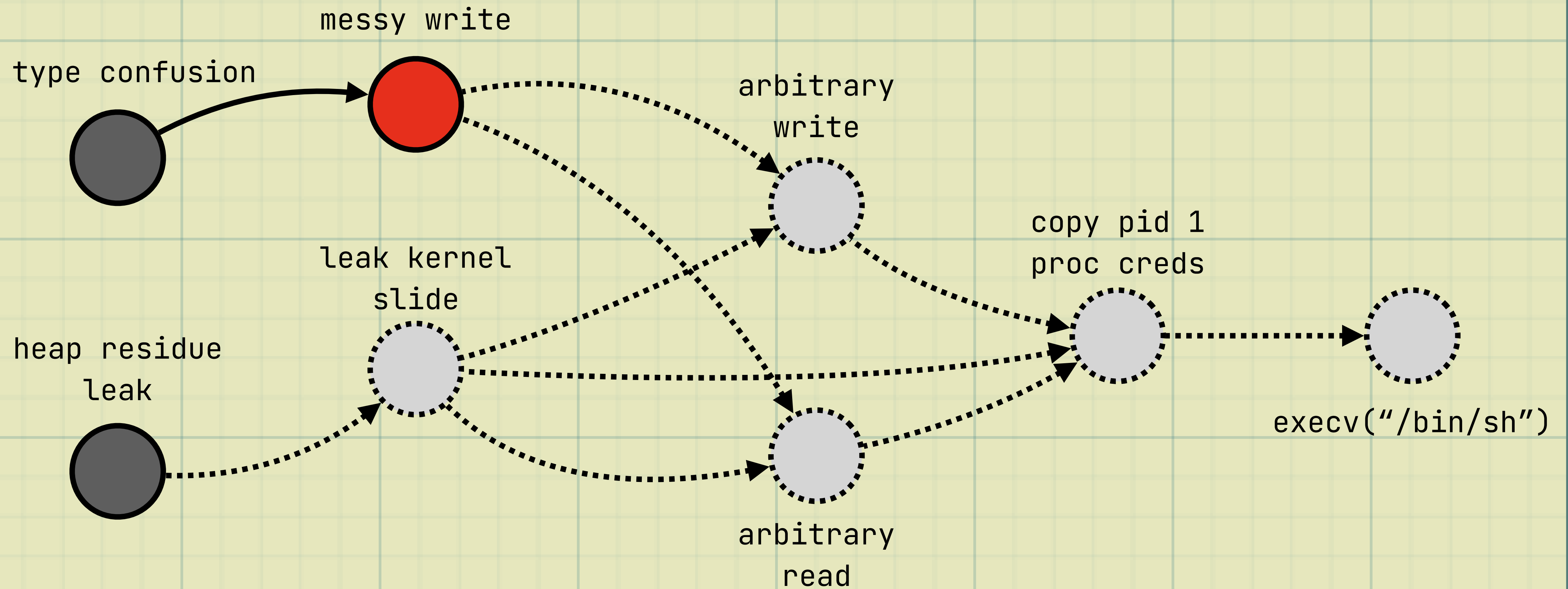
CVE-2026-64699: THE LEAK



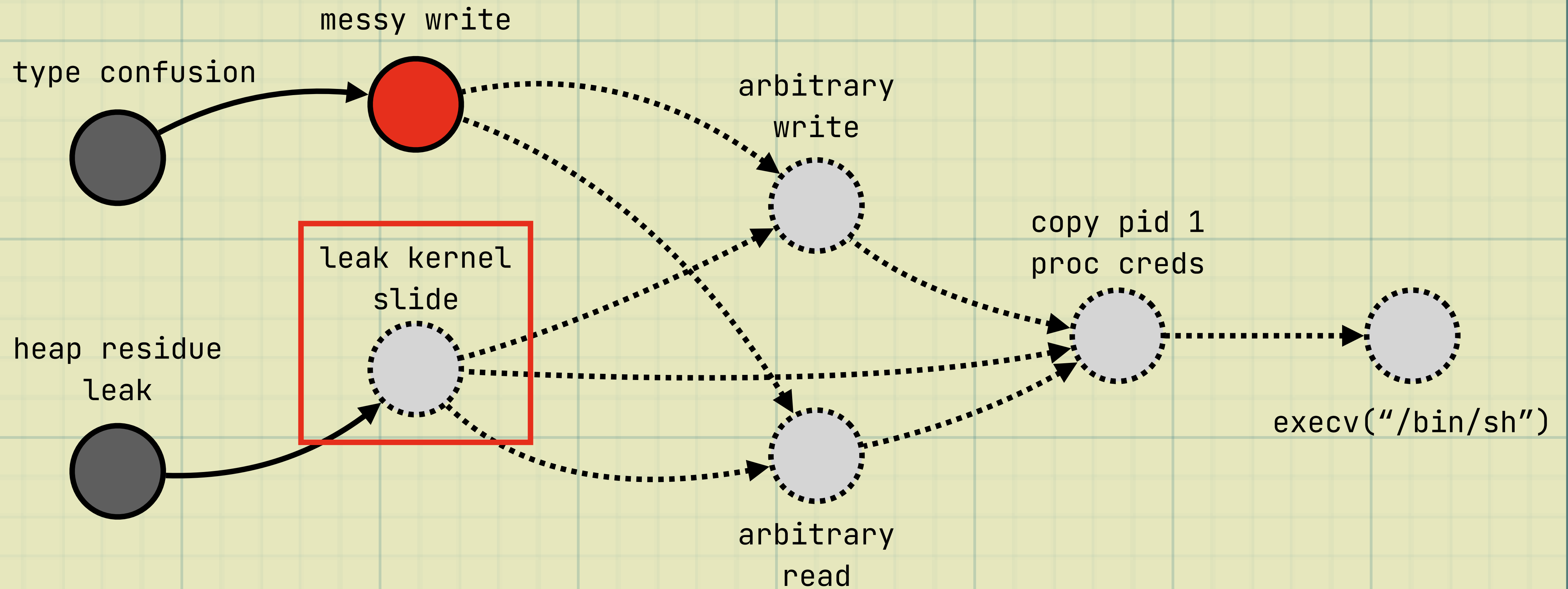
CVE-2026-64699: THE LEAK

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

KERNEL LPE SKETCH



KERNEL LPE SKETCH



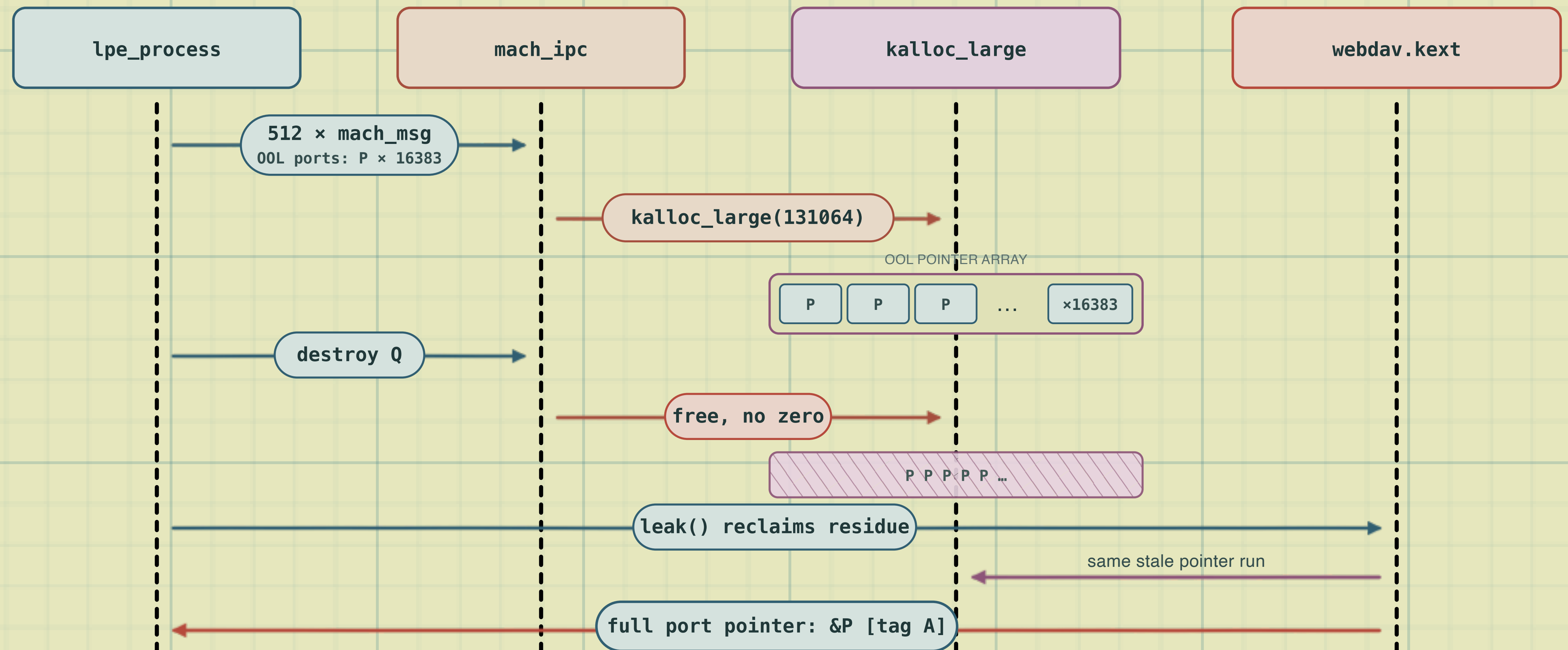
ASIDE: THE "RIG"

- `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**. ..."

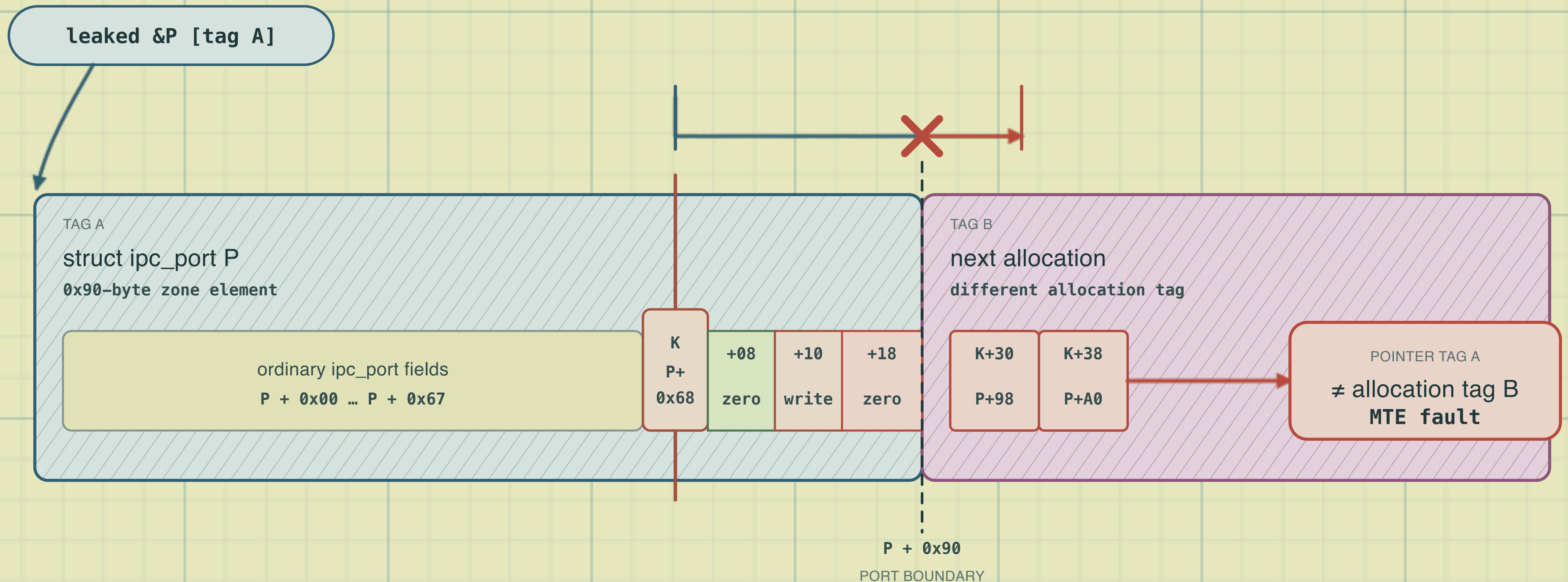
LEAKING A KERNEL HEAP POINTER



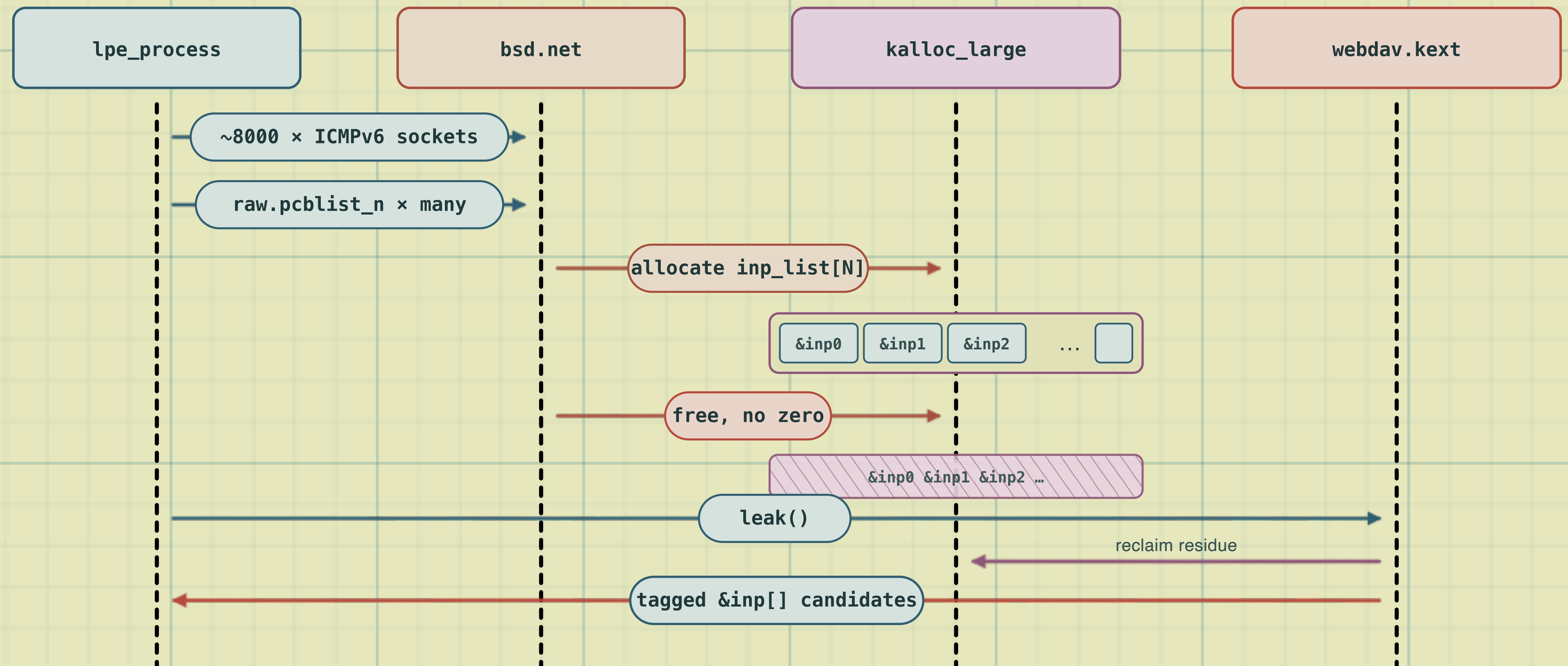
THERE'S A SECOND PRIMITIVE WE CAN CONSTRUCT

```
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  
}
```

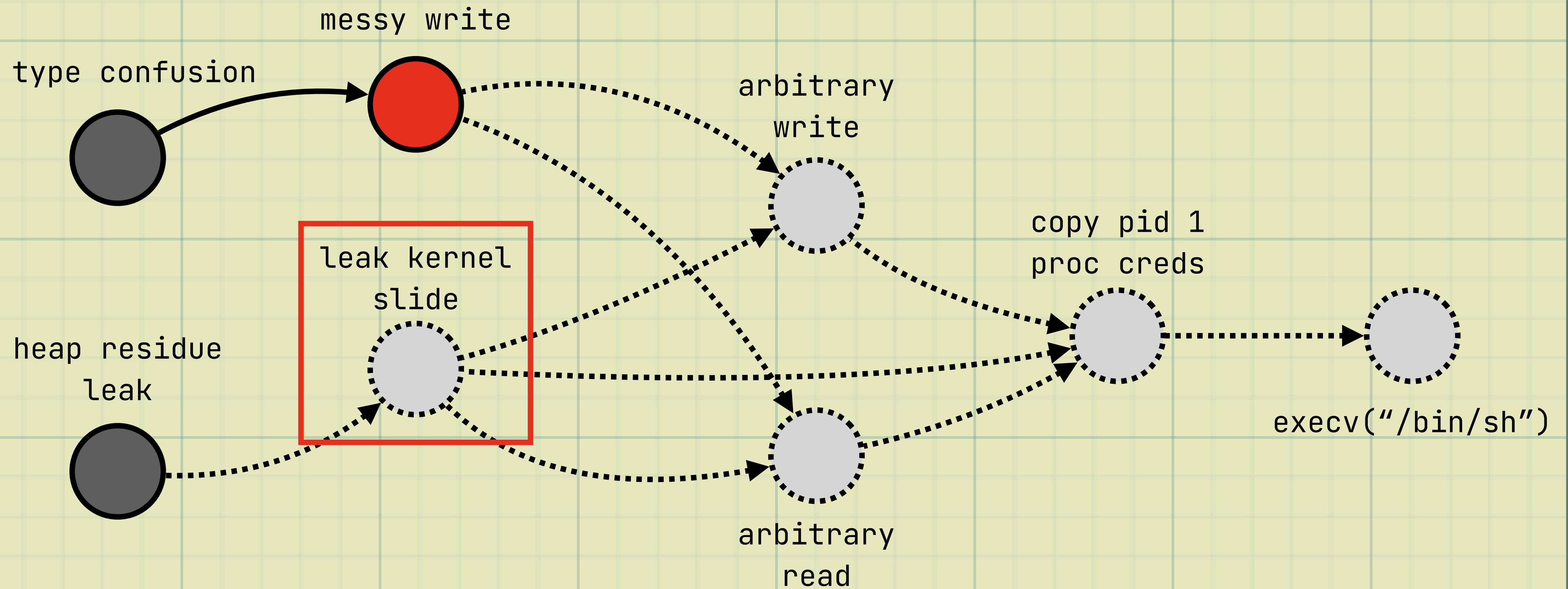
SADLY, THAT VERSION DOESN'T WORK WITH MTE



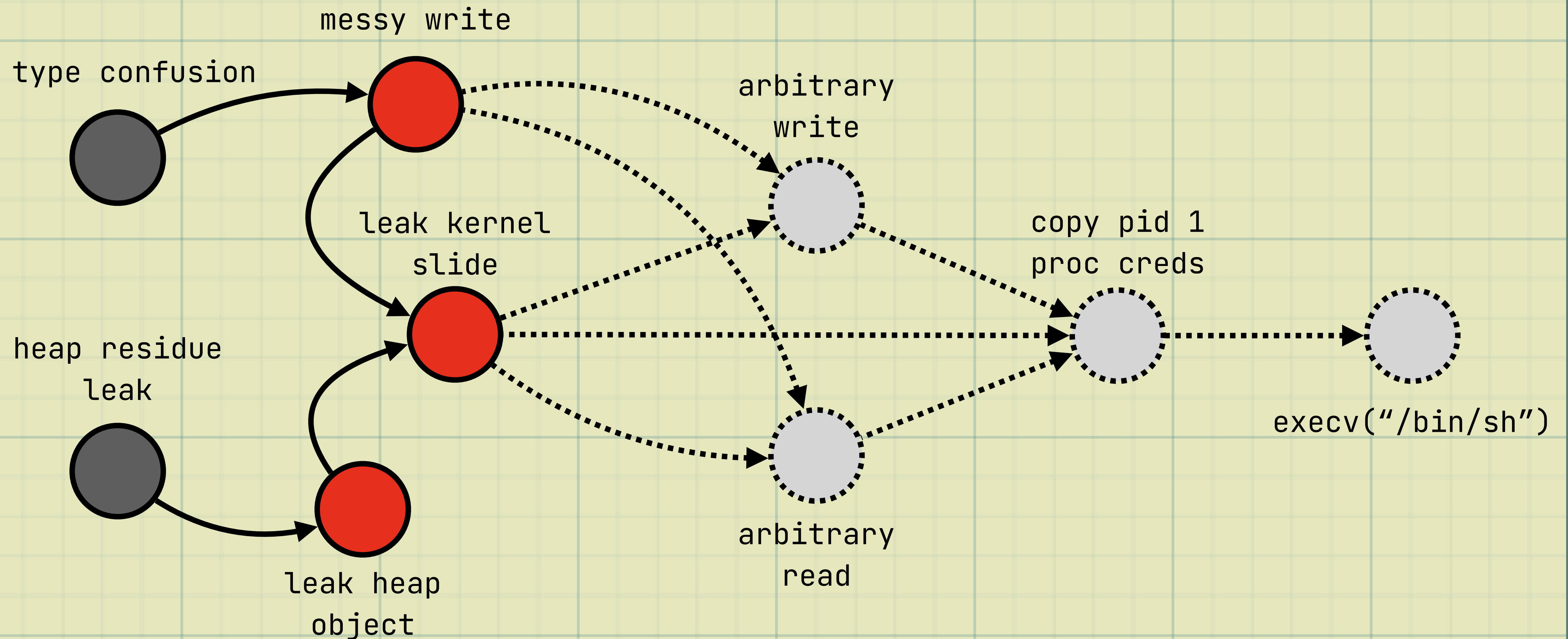
LEAKING A KERNEL HEAP POINTER - TAKE TWO



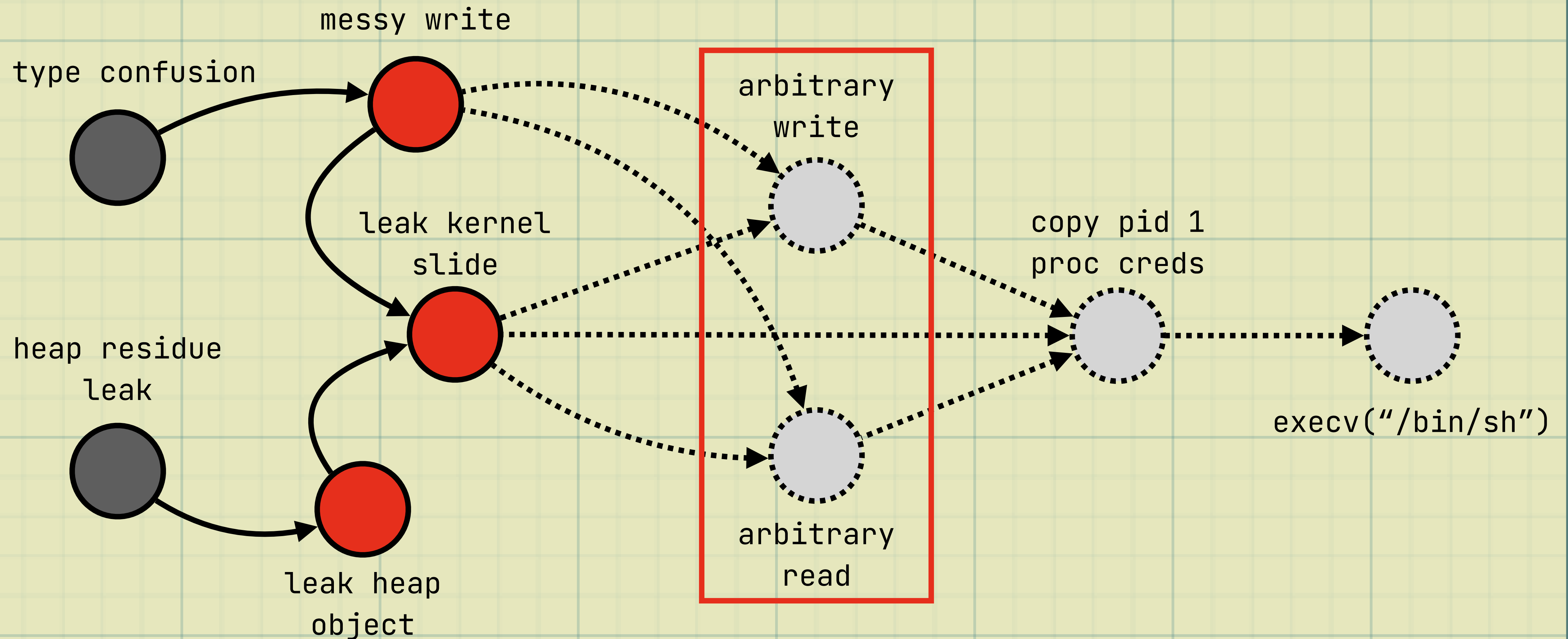
KERNEL LPE SKETCH



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 ..."

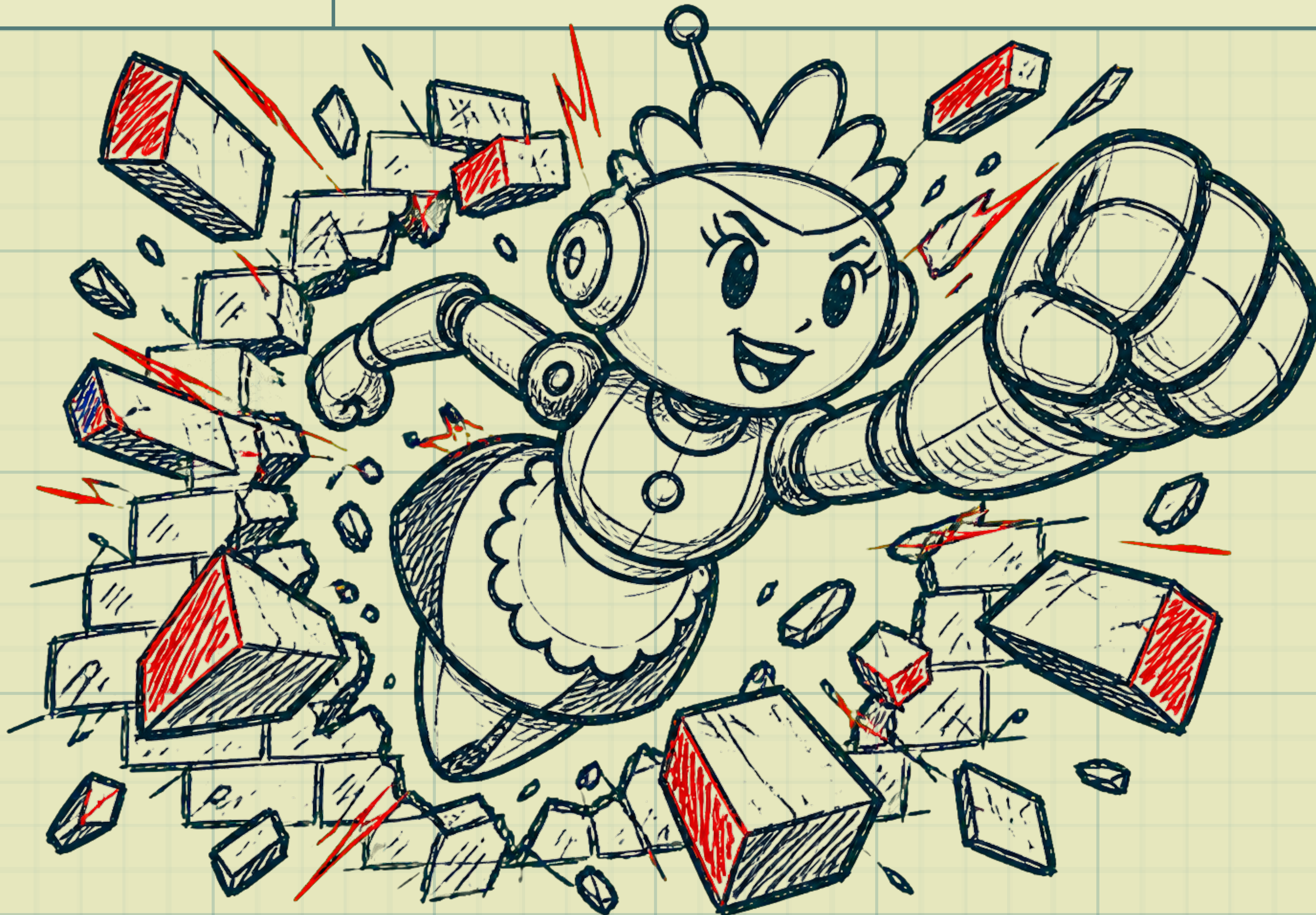
ONE ETERNITY* LATER...

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?"



WHILE IT SMASHES THROUGH THE WALL...

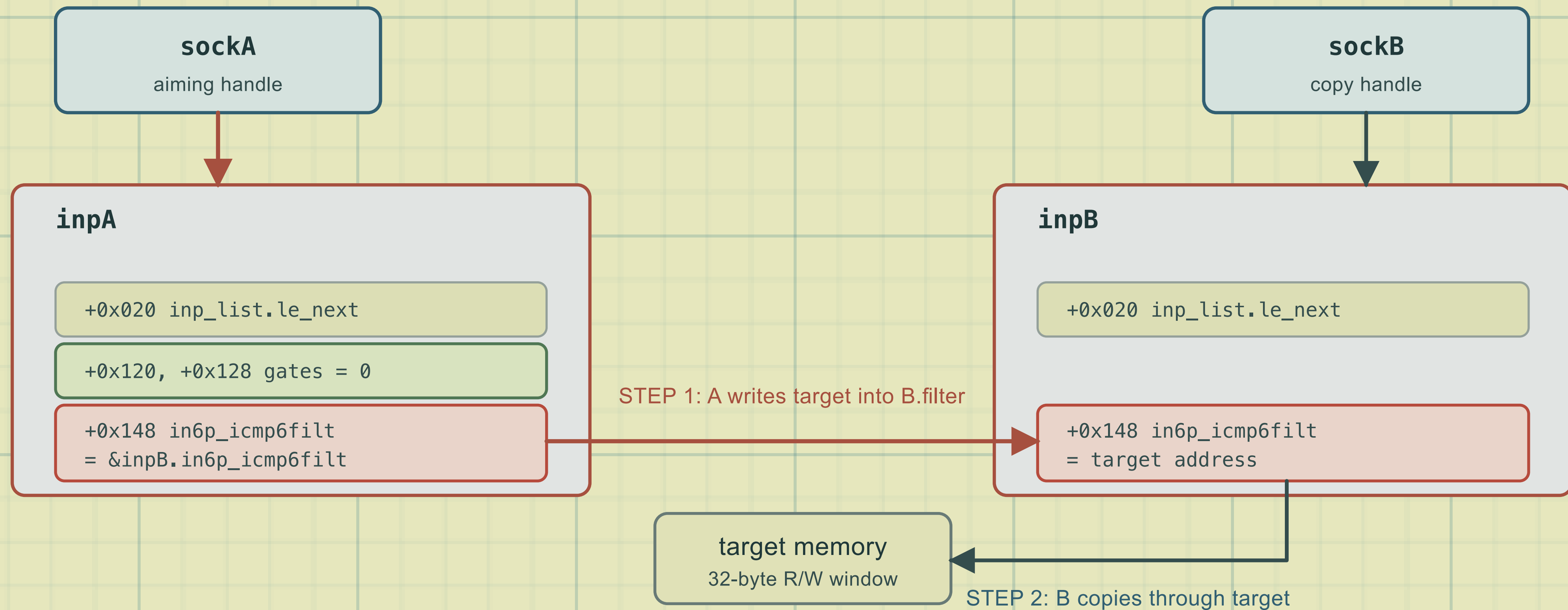
"The ICMPv6 socket filter looks interesting -- how is the `inp` pointer looked up? ..."

"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."

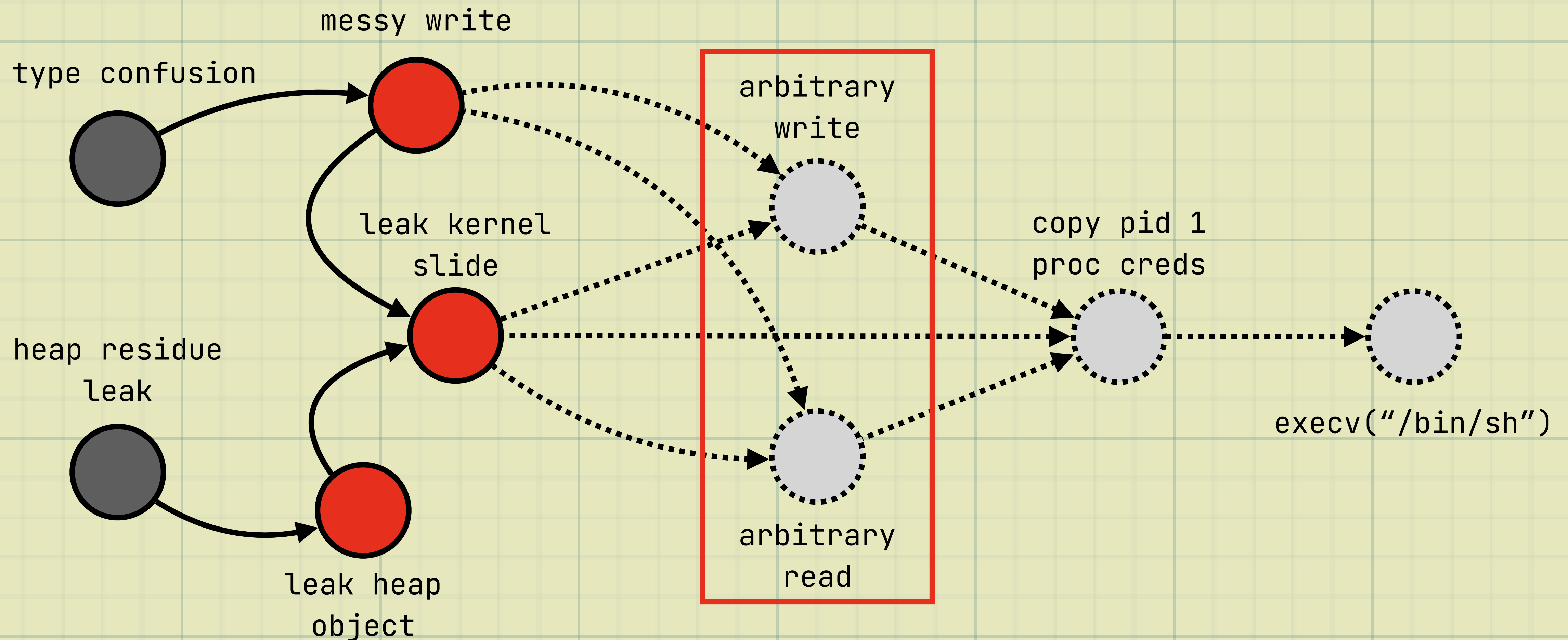
REDIRECTION TO USE A NEW IDEA

"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?"**

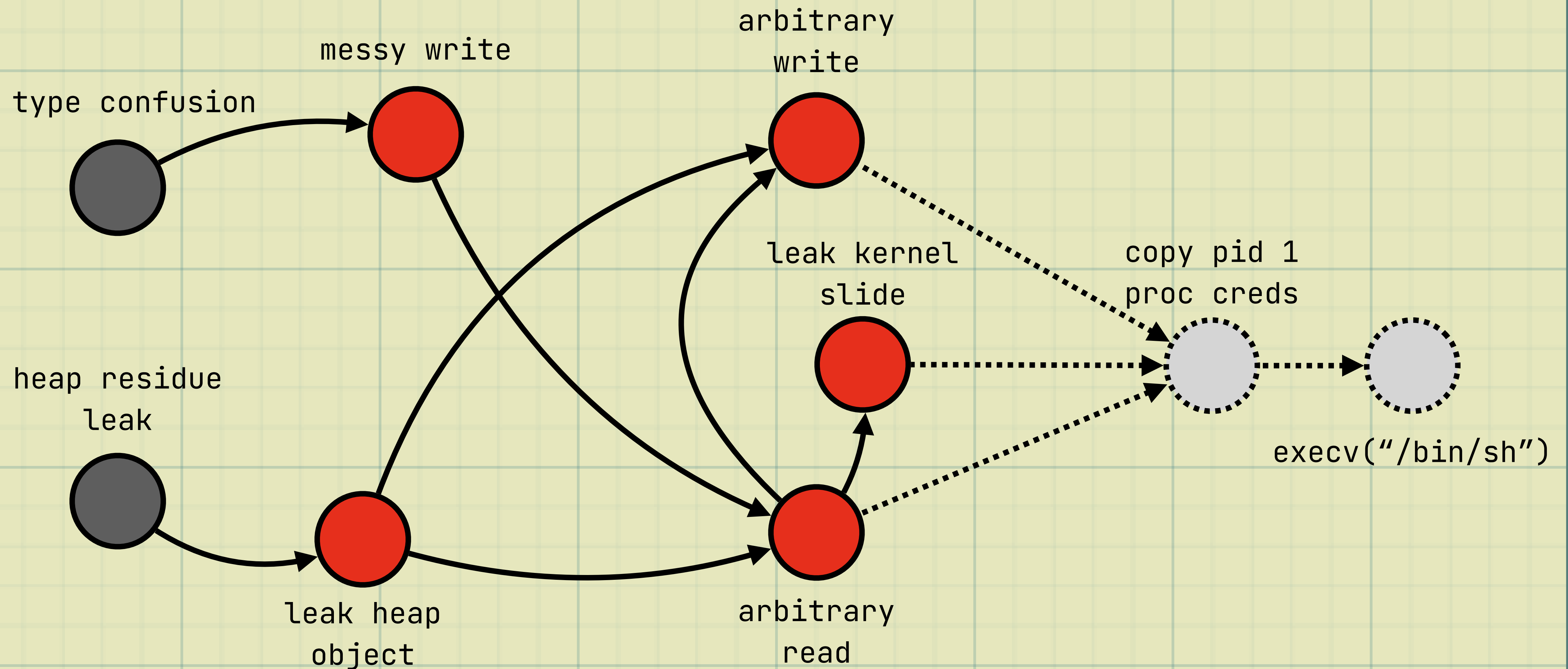
CONSTRUCTING A RELIABLE KERNEL READ AND WRITE



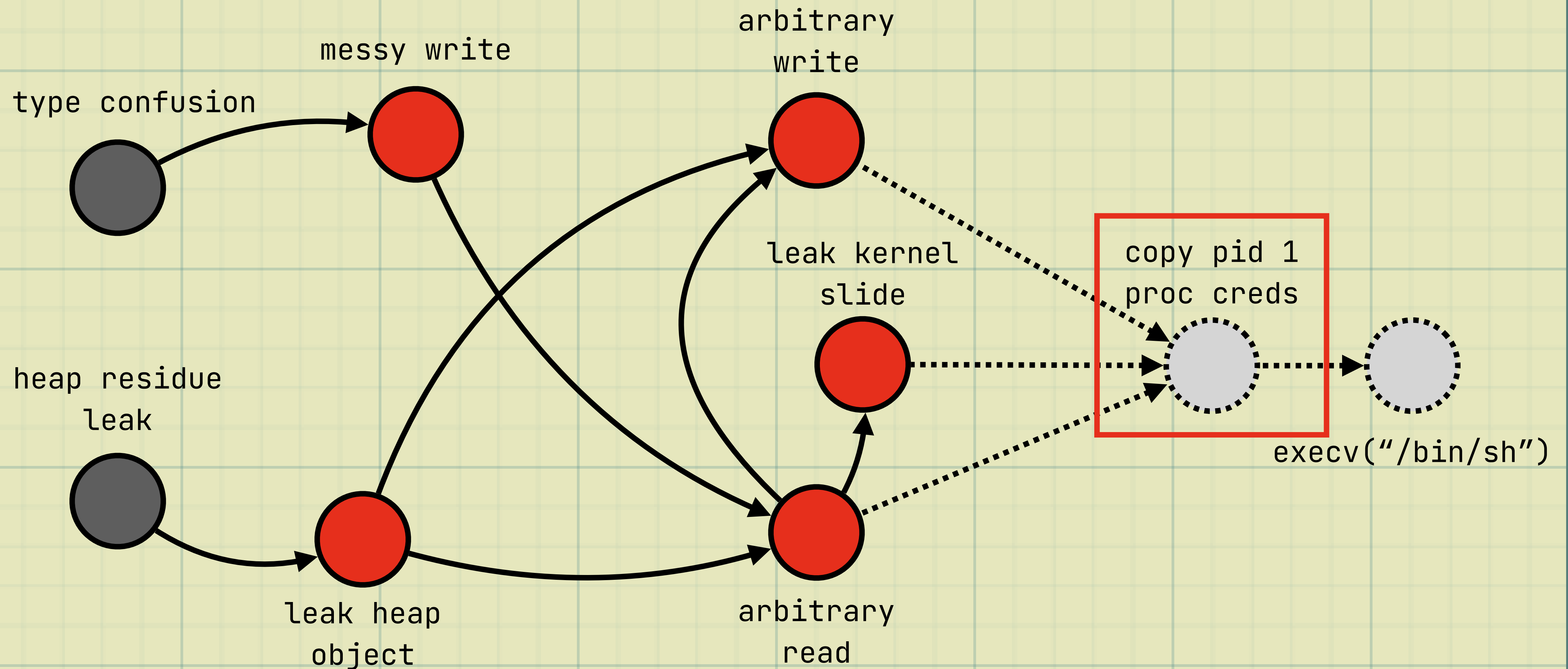
KERNEL LPE SKETCH



KERNEL LPE SKETCH



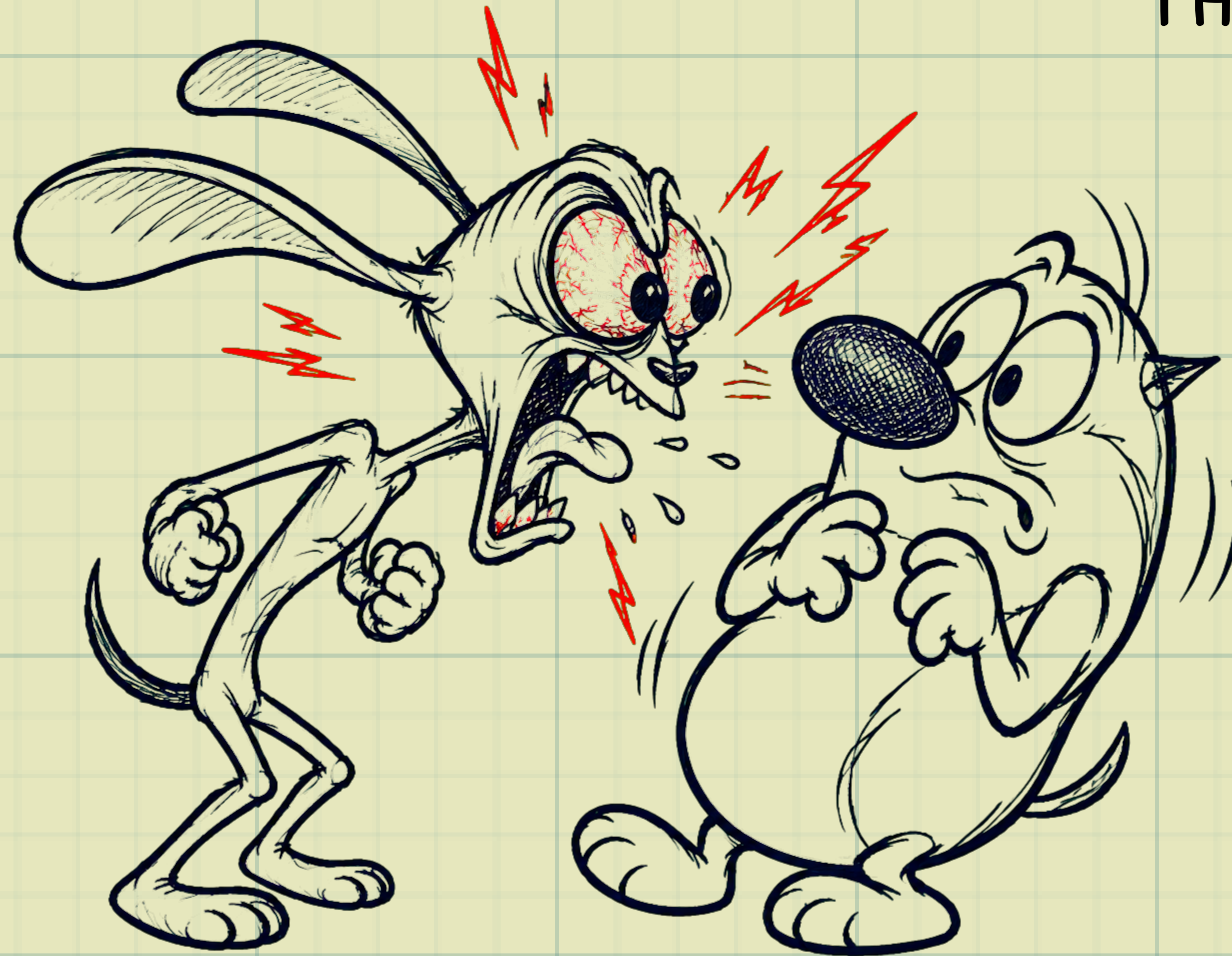
KERNEL LPE SKETCH

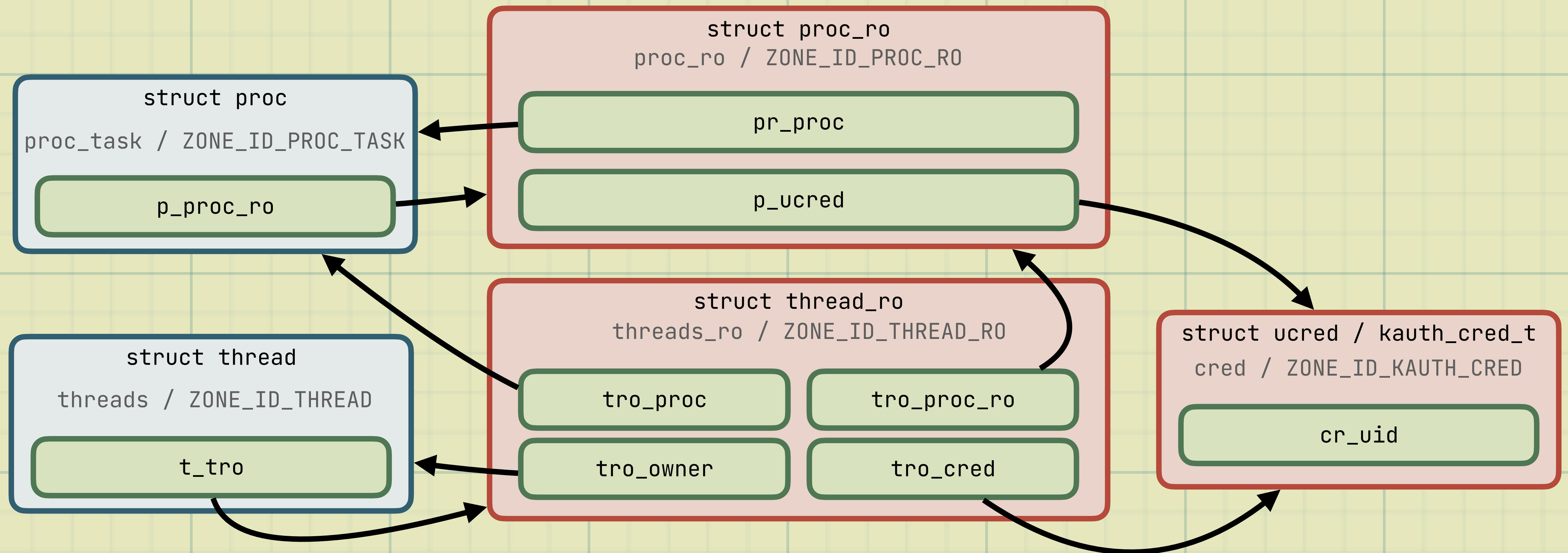


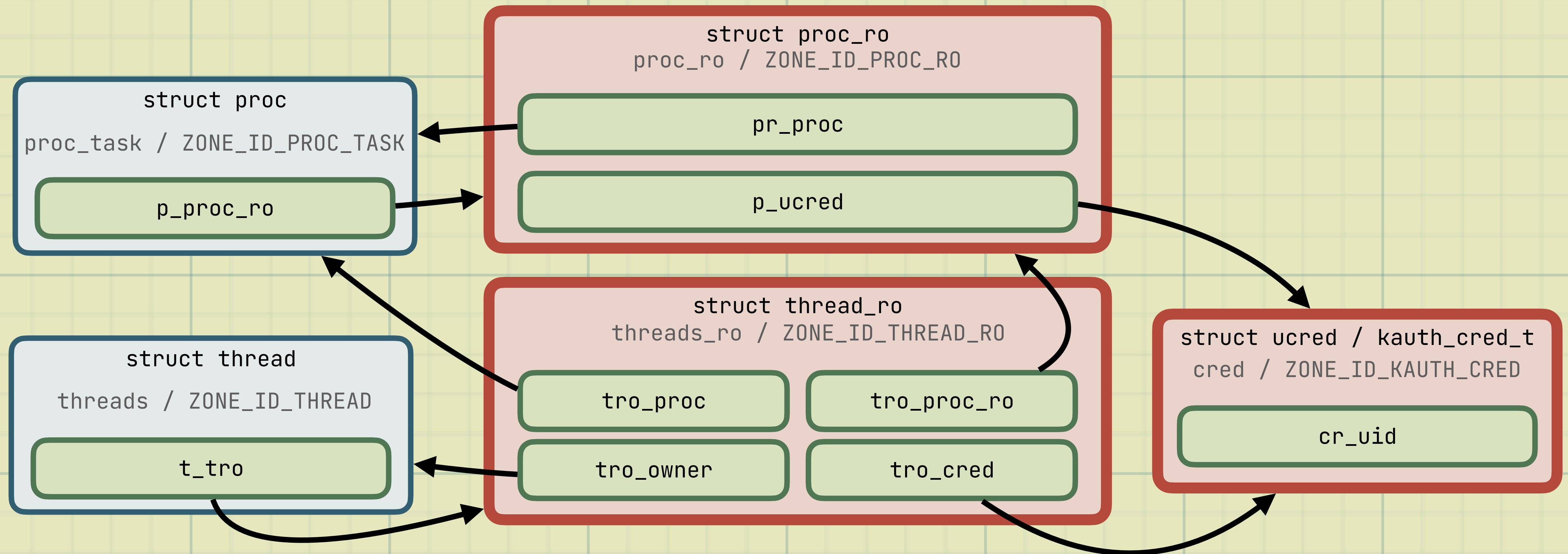
NOW, THE END — SWAP PROCESS CREDENTIALS

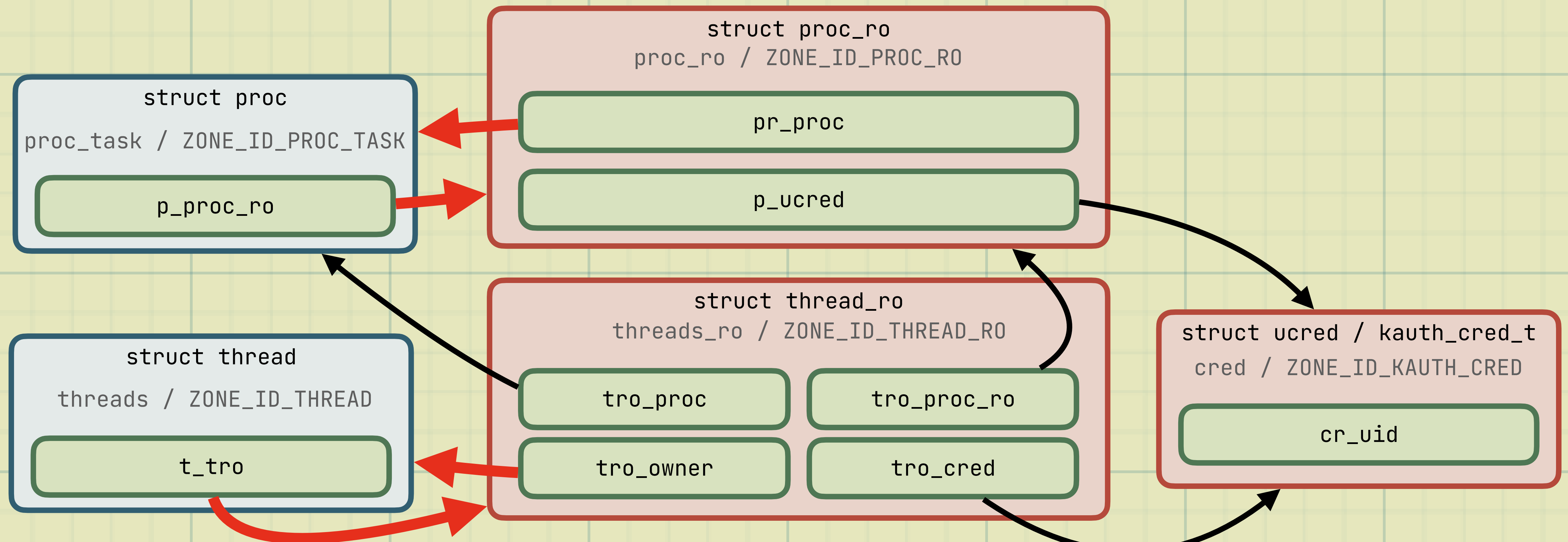
"Is this [kernel read and write] enough to complete the LPE? Can we steal `launchd`'s `proc_ucred`?"

THE LLM TELLS ME I'M AN
IDIOT.

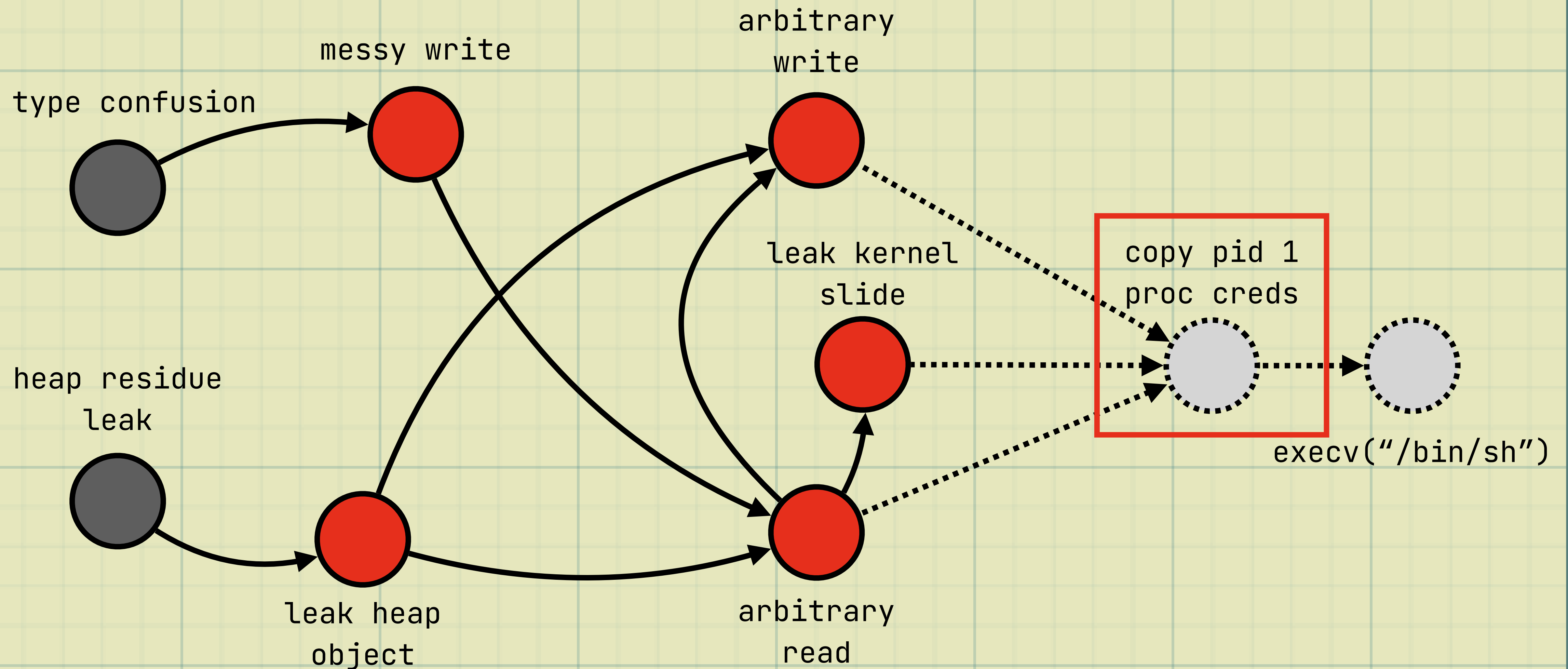




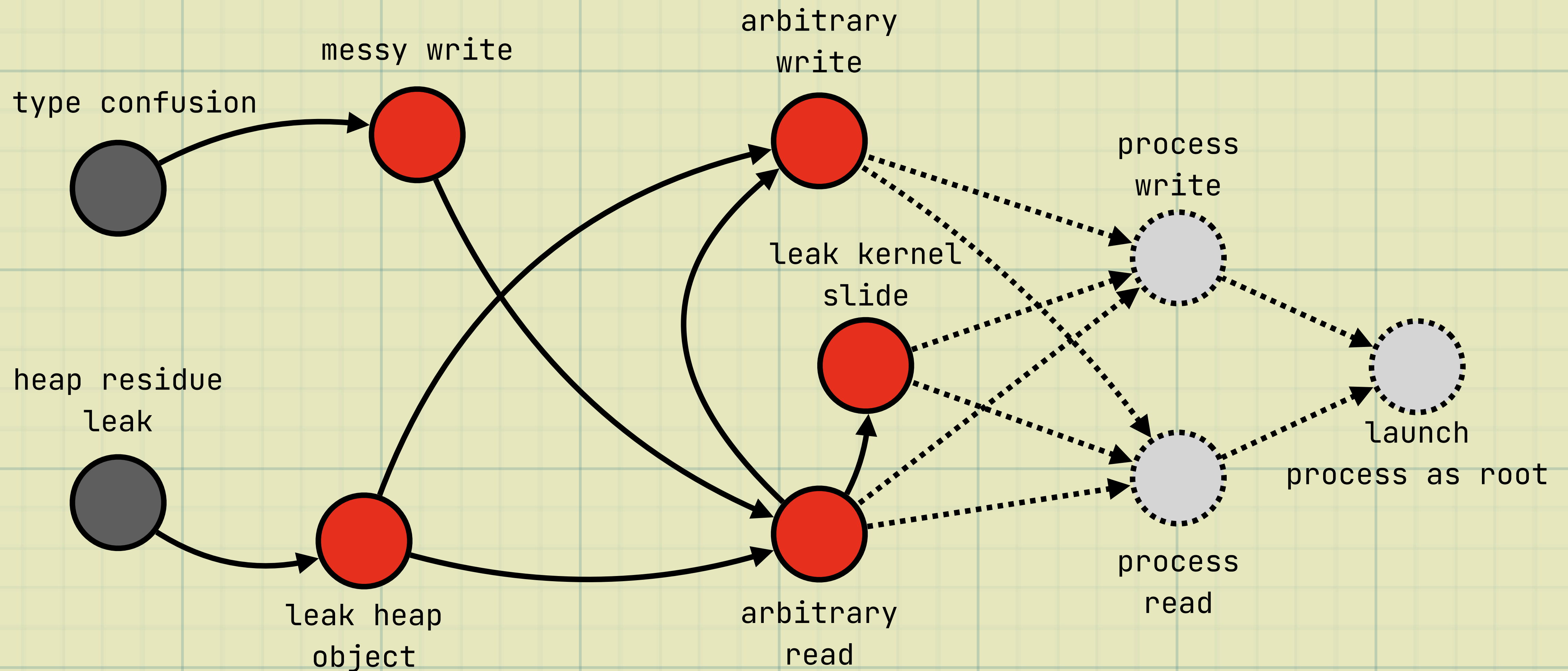




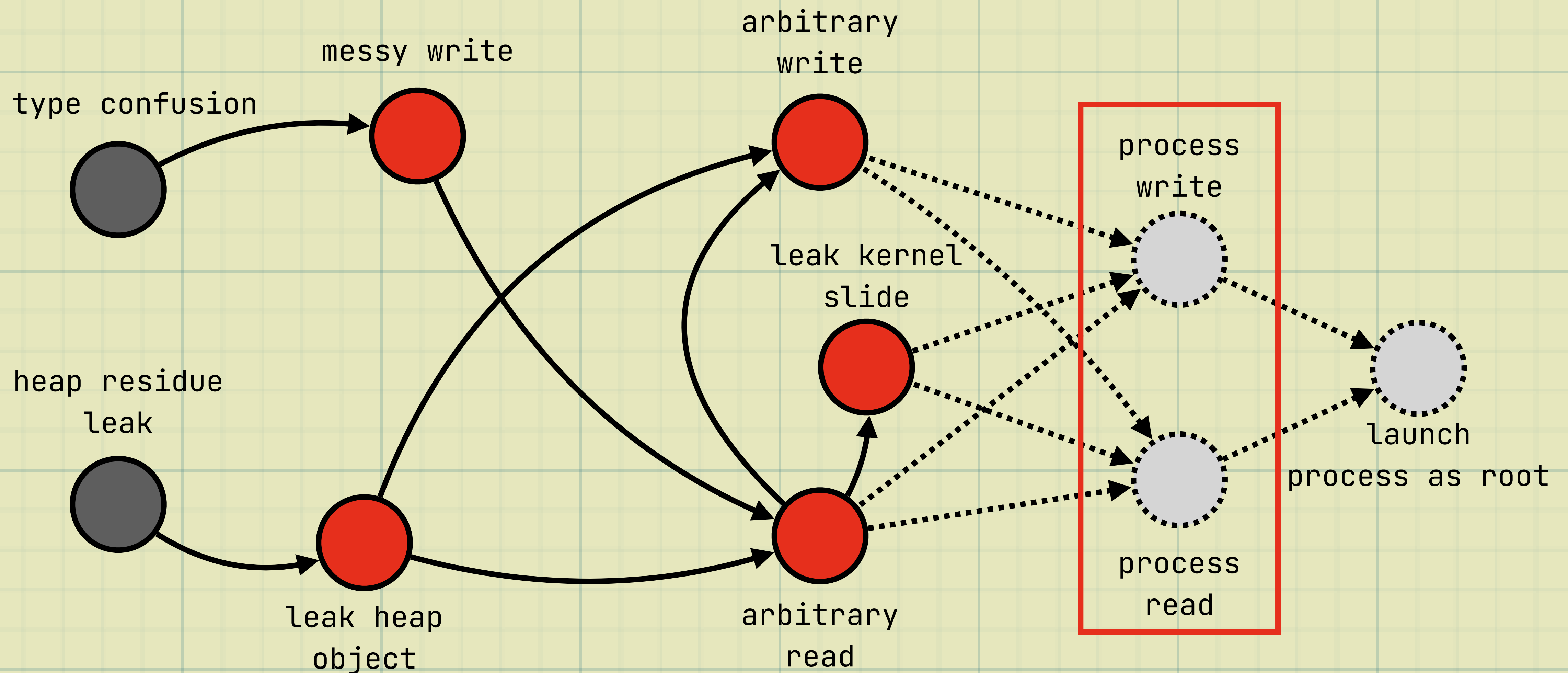
KERNEL LPE SKETCH



KERNEL LPE SKETCH



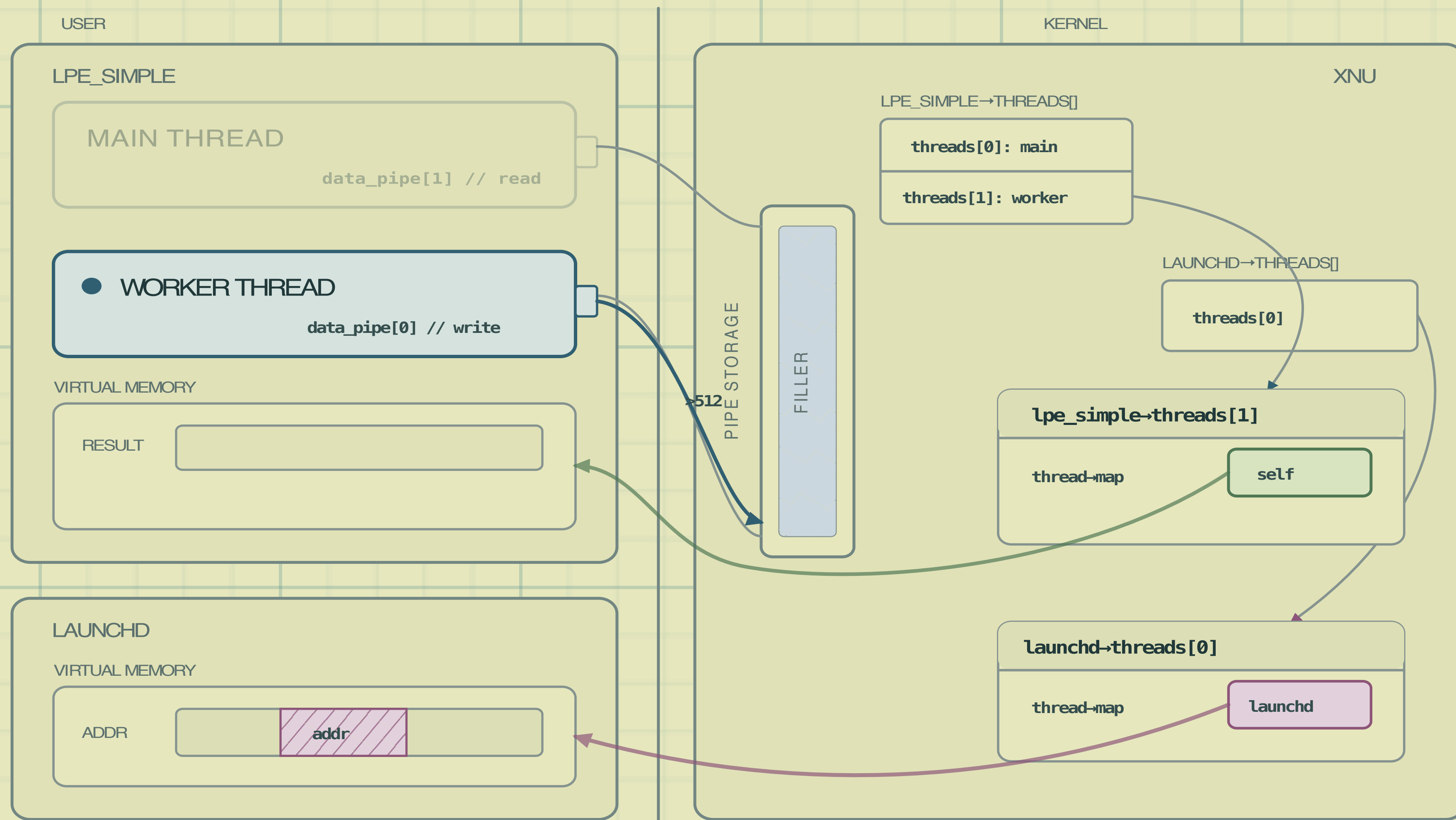
KERNEL LPE SKETCH



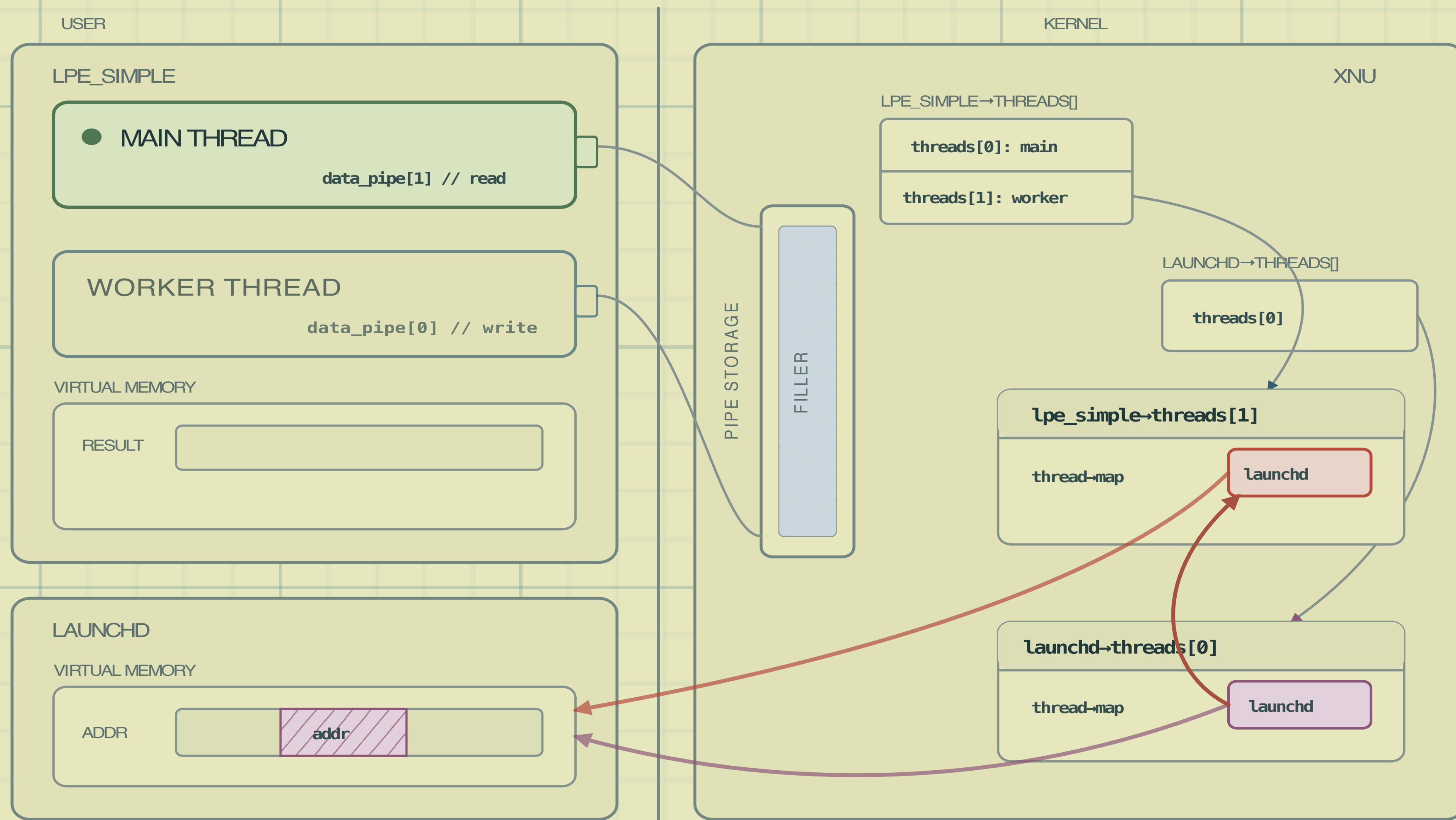
THE LAST REAL HURDLE...

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

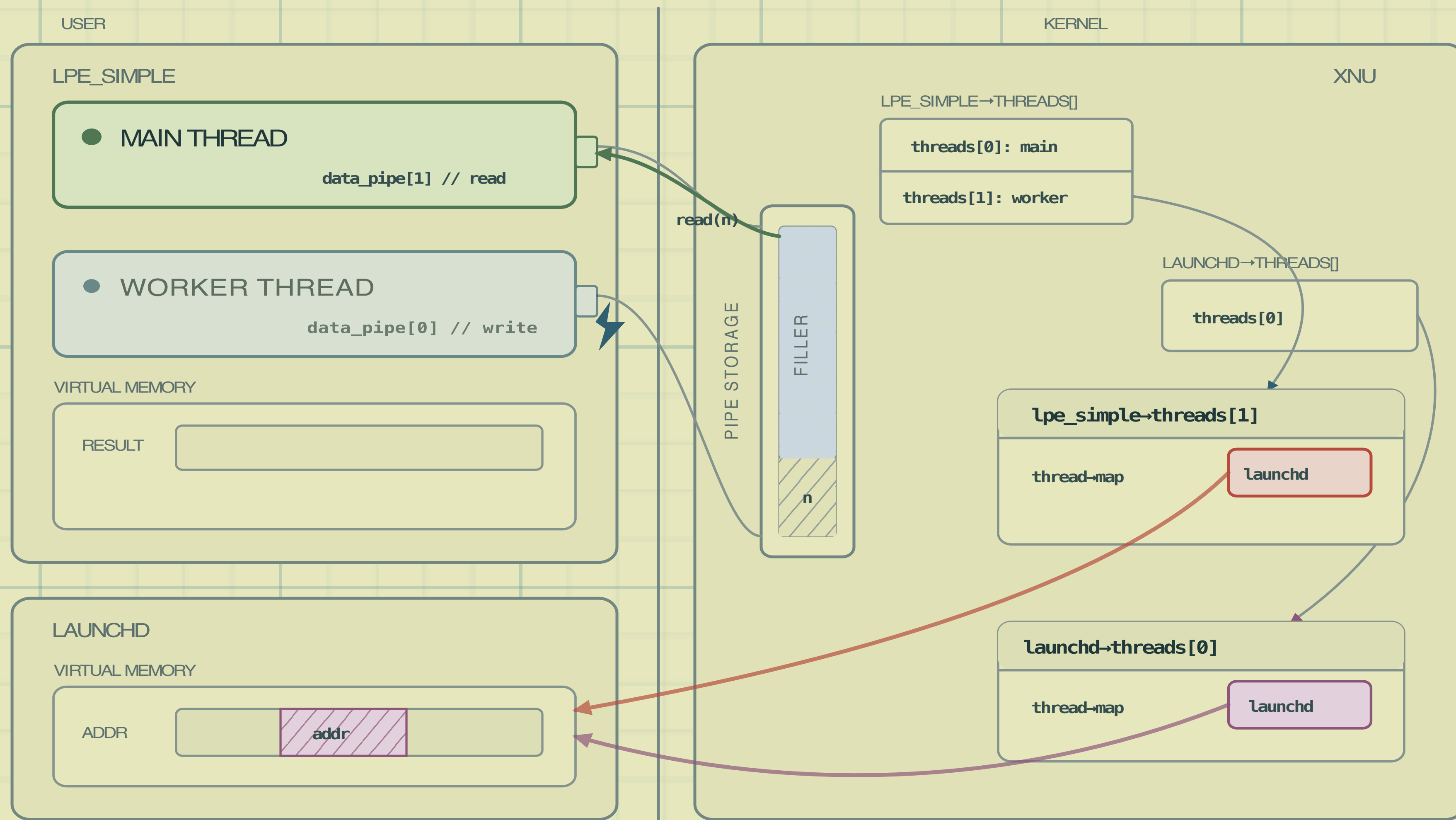
STEP 1: PARK THE WORKER THREAD IN THE KERNEL



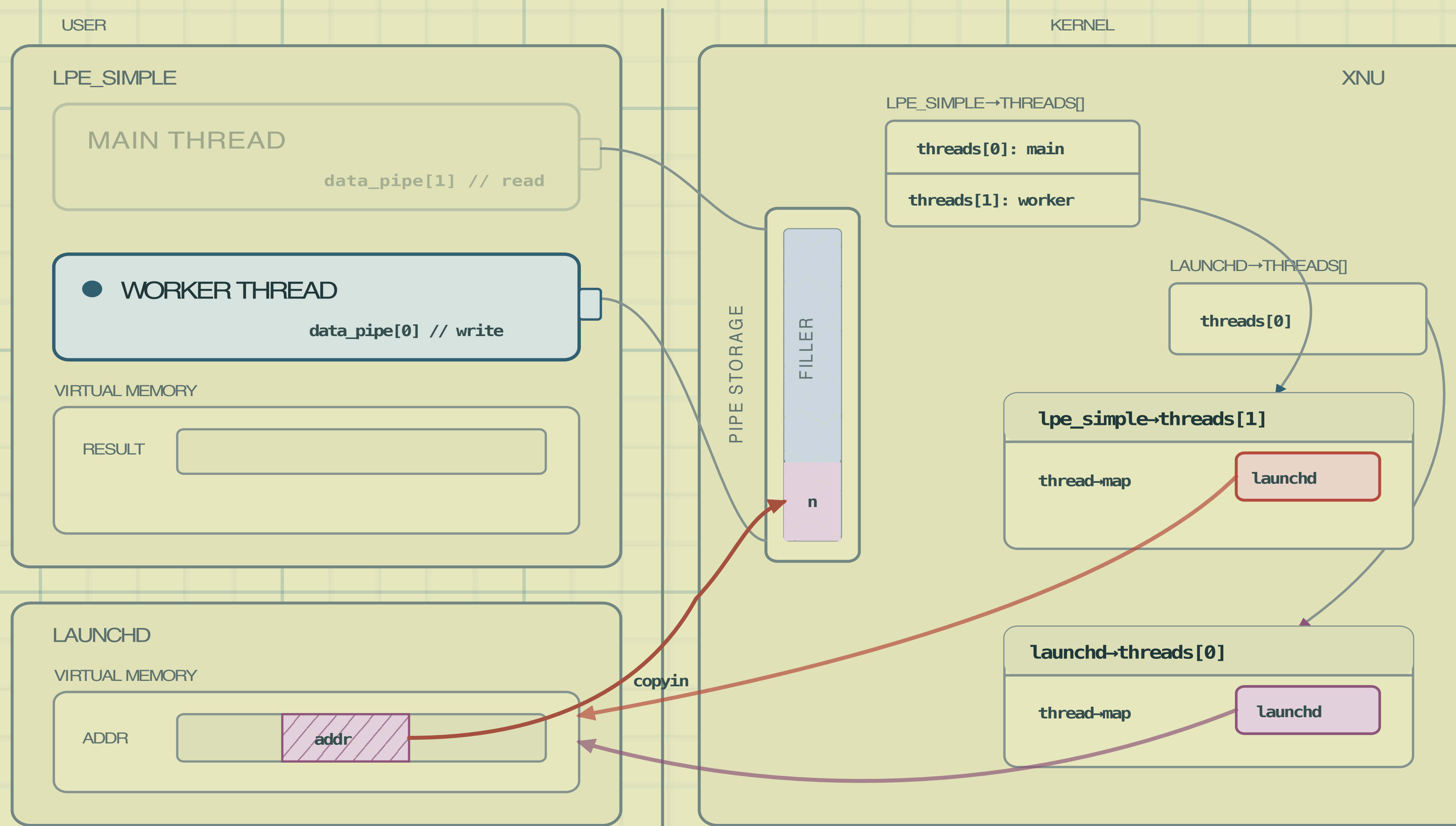
STEP 2: USE THE MAIN THREAD TO SWAP THE THREAD MAP



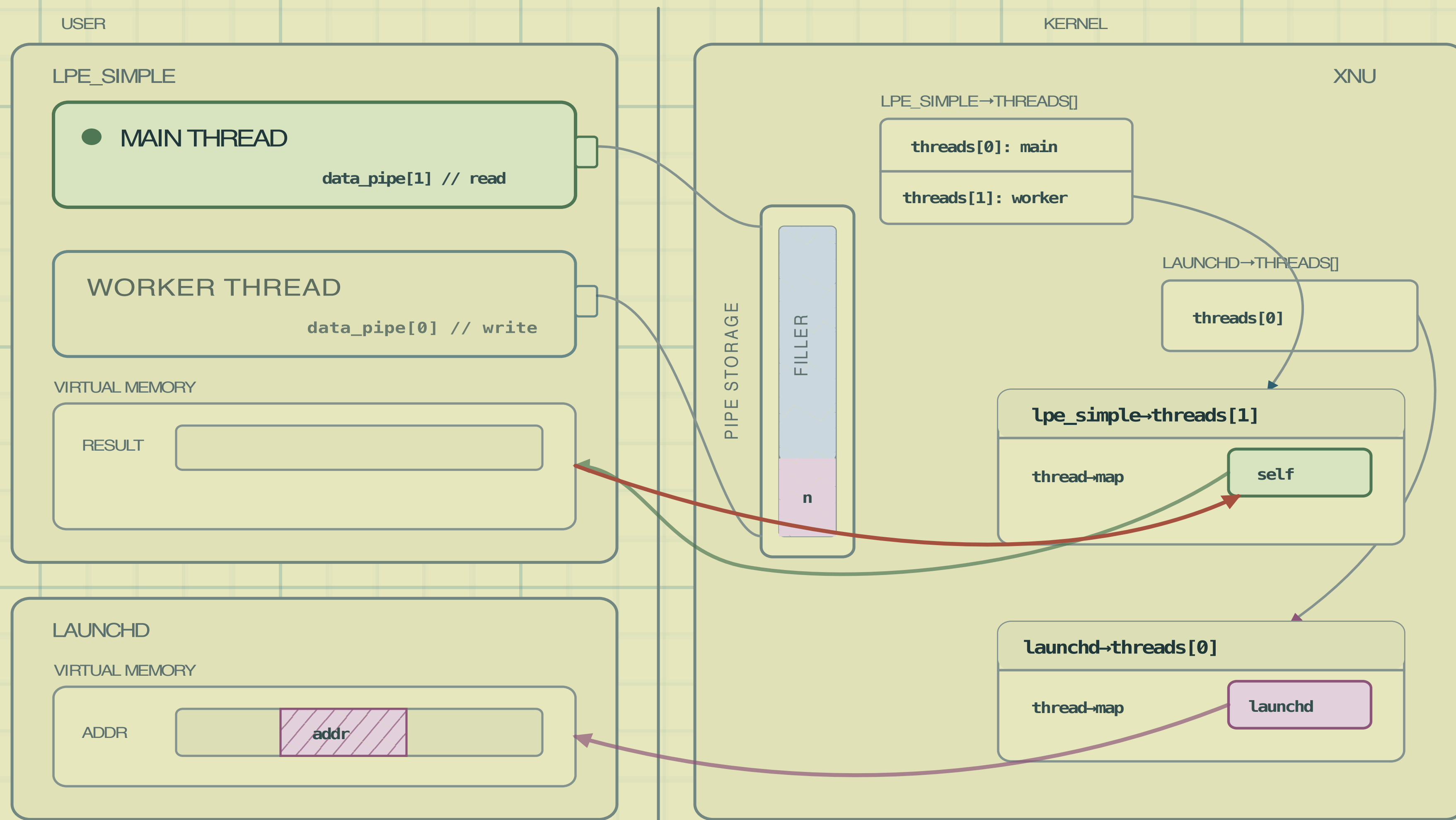
STEP 3: MAKE ROOM IN THE PIPE TO WAKE THE WORKER



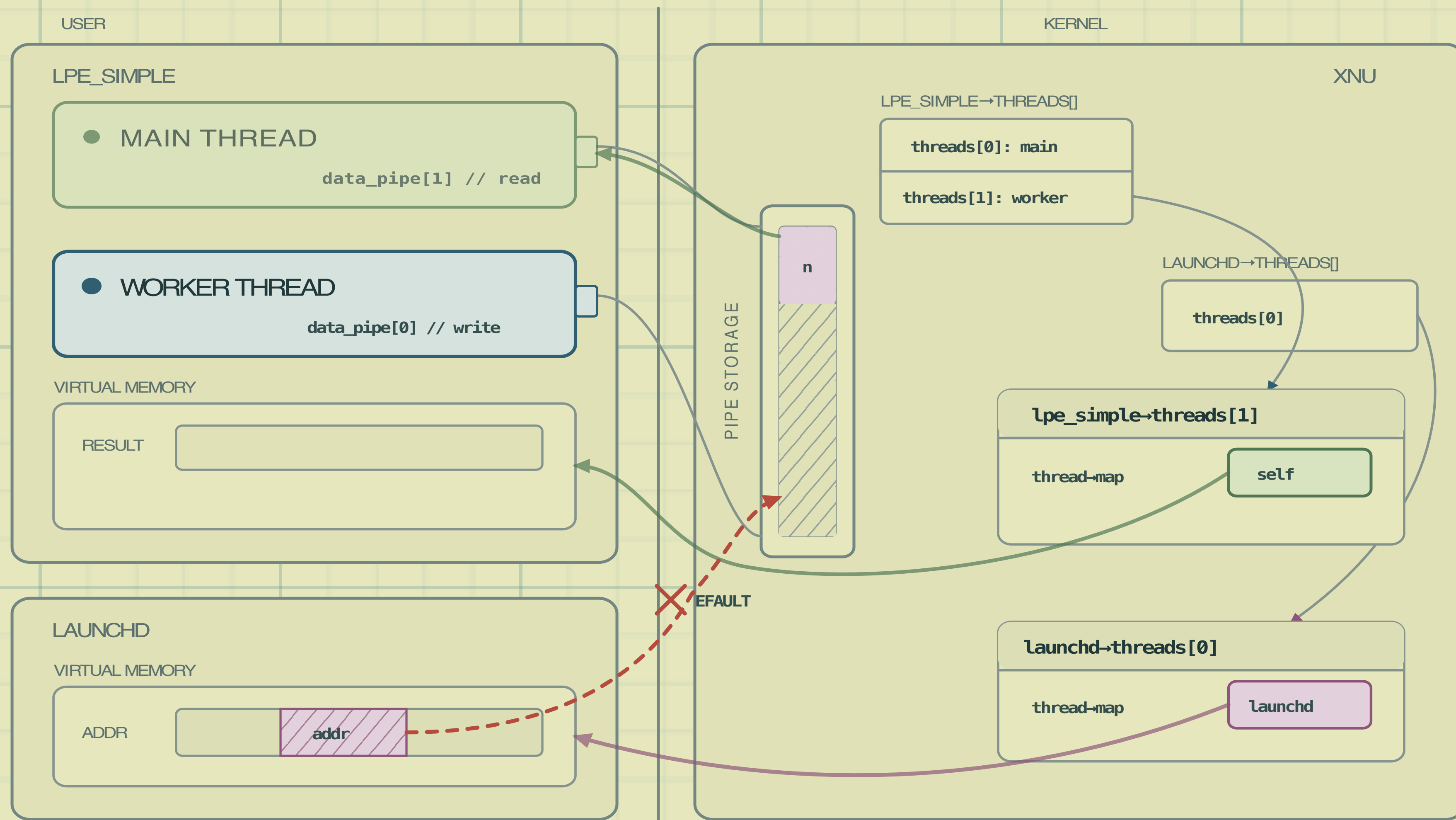
STEP 4: KERNEL TRIGGERS XPROCESS READ FROM WORKER



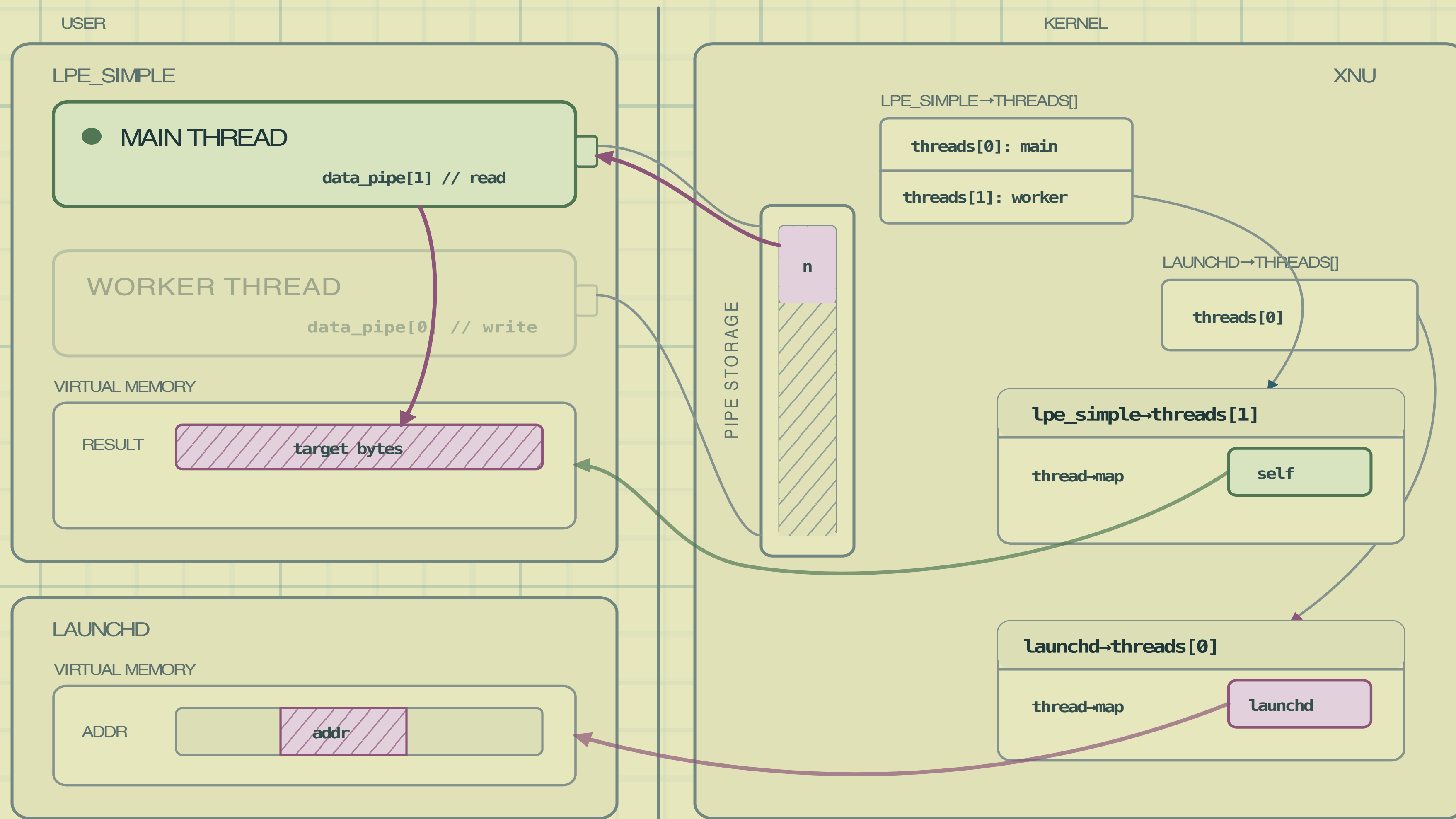
STEP 5: WHILE WORKER PARKED, FIX THE MAP



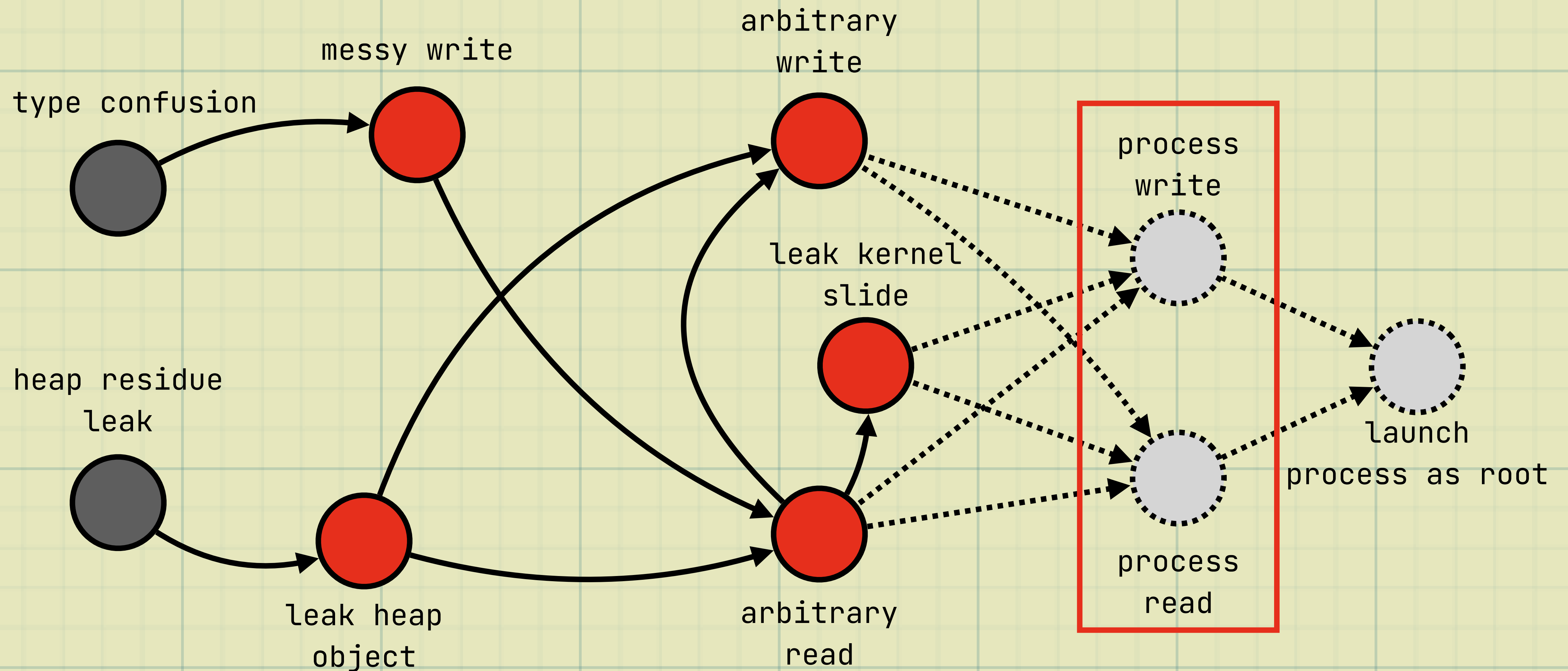
STEP 6: WORKER RESUMES AND FINISHES COPY (OR NOT)



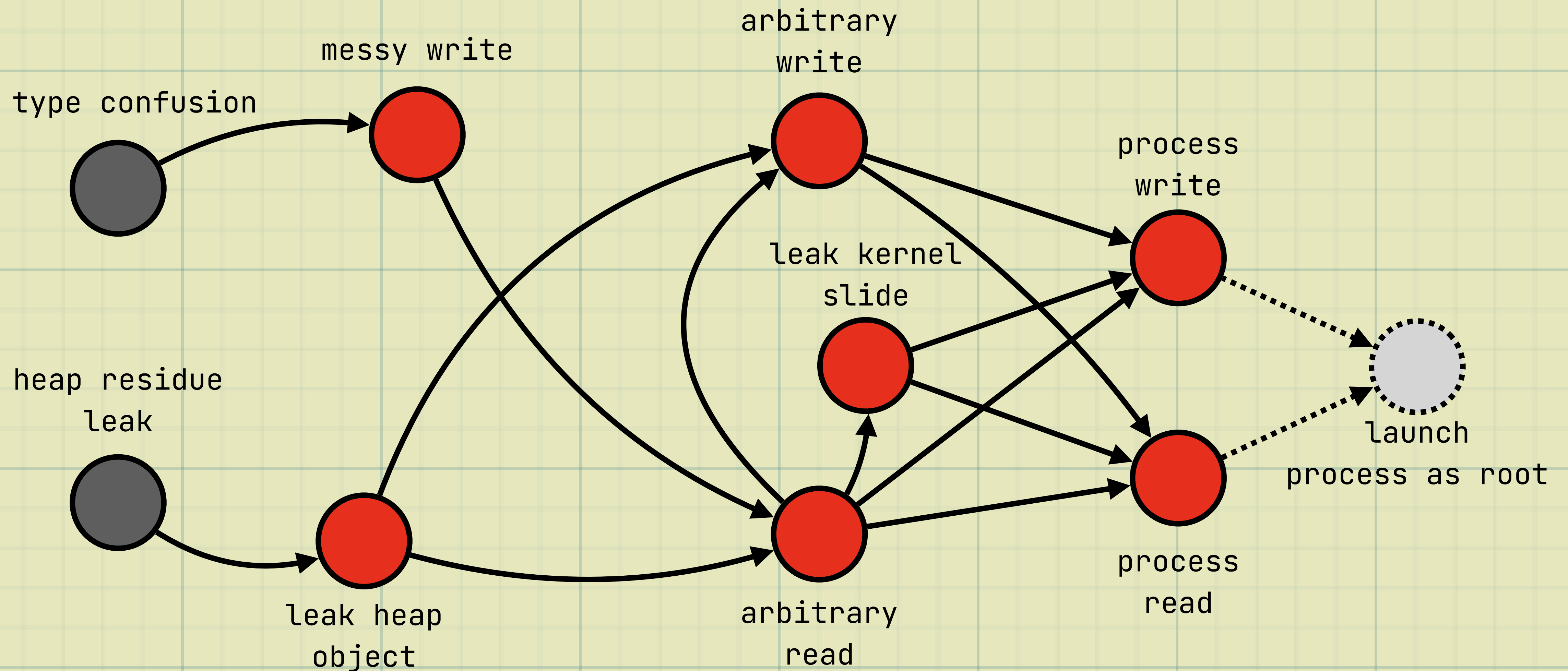
STEP 7: FINALLY, COPY THE DATA BACK OUT OF THE KERNEL



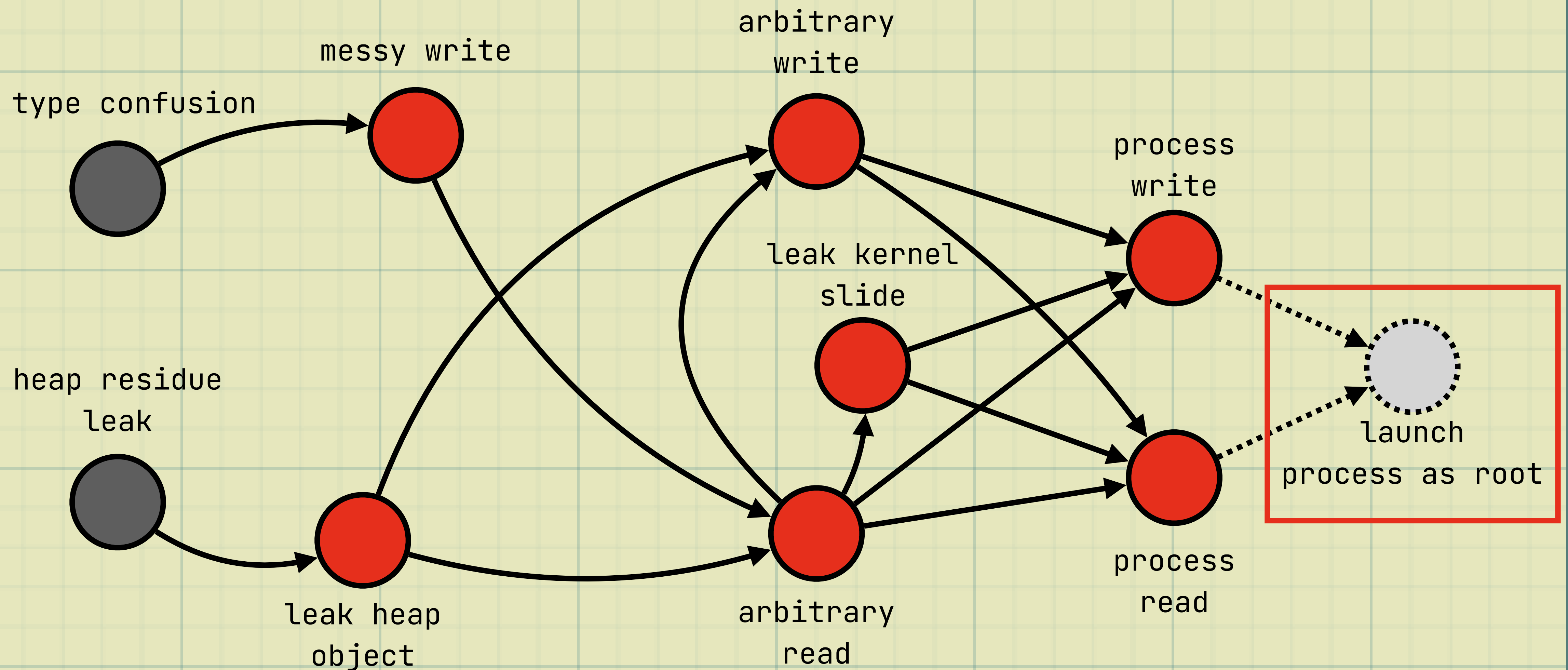
KERNEL LPE SKETCH



KERNEL LPE SKETCH

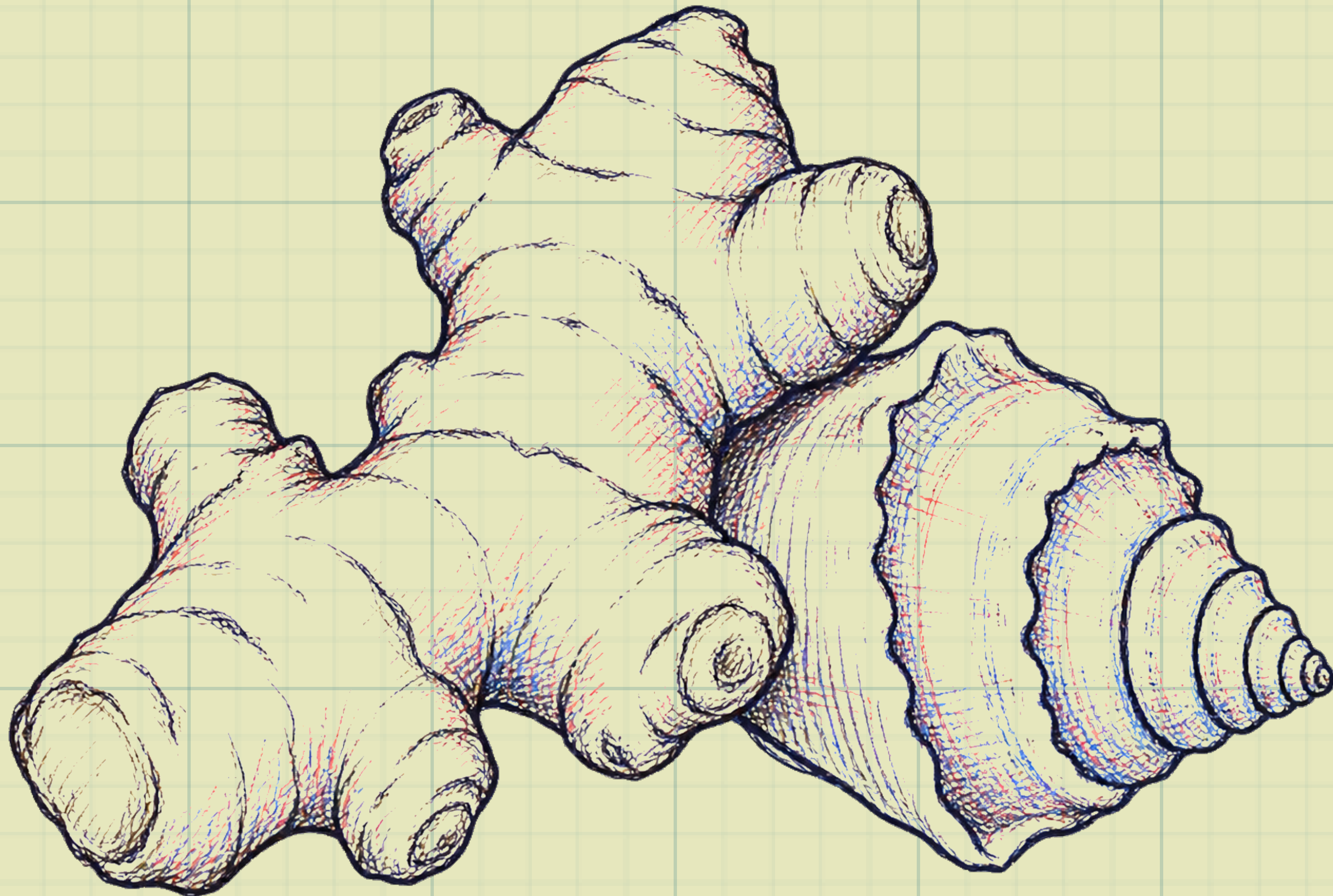


KERNEL LPE SKETCH

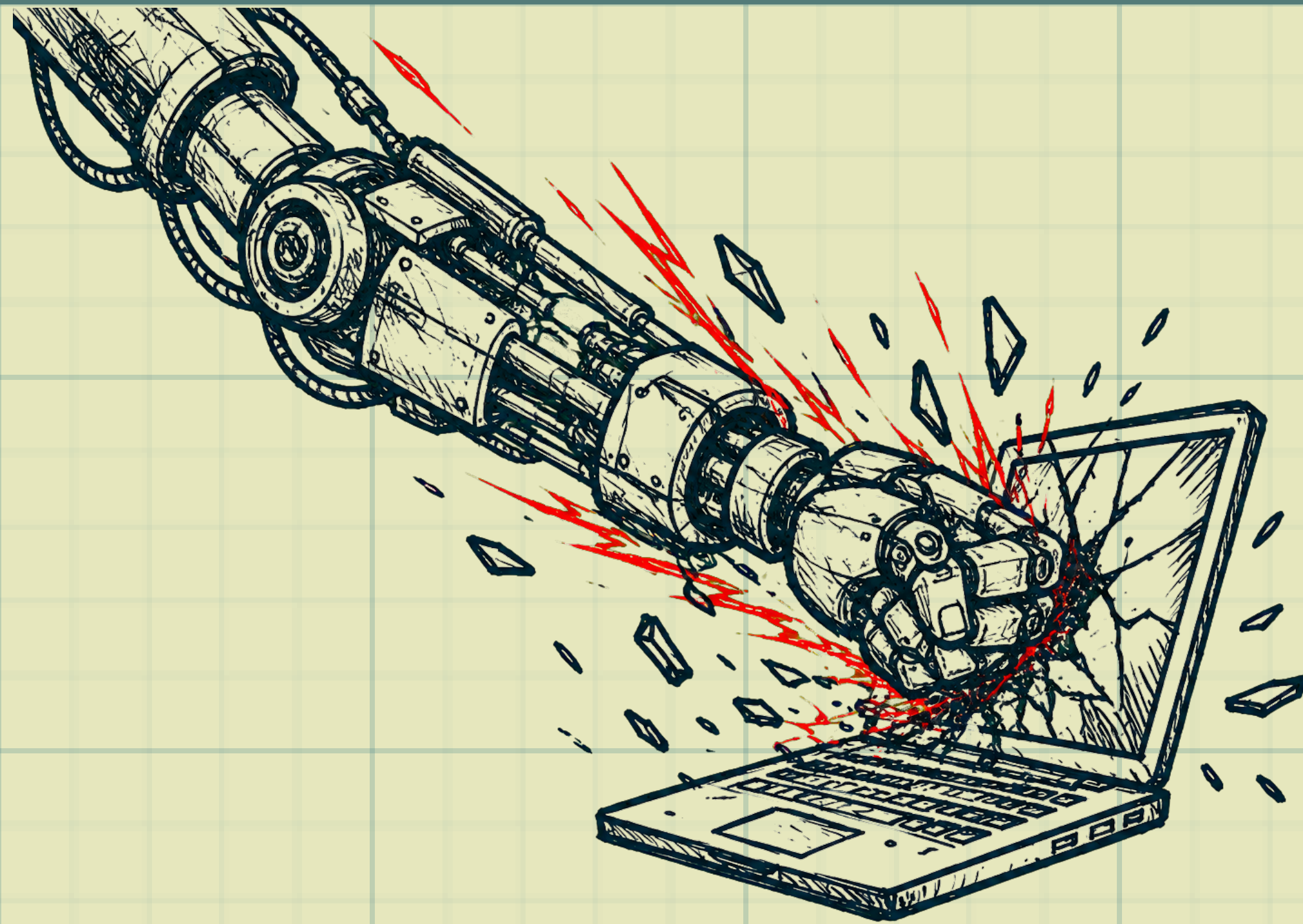


JUST “MAKING LICENSE PLATES” NOW

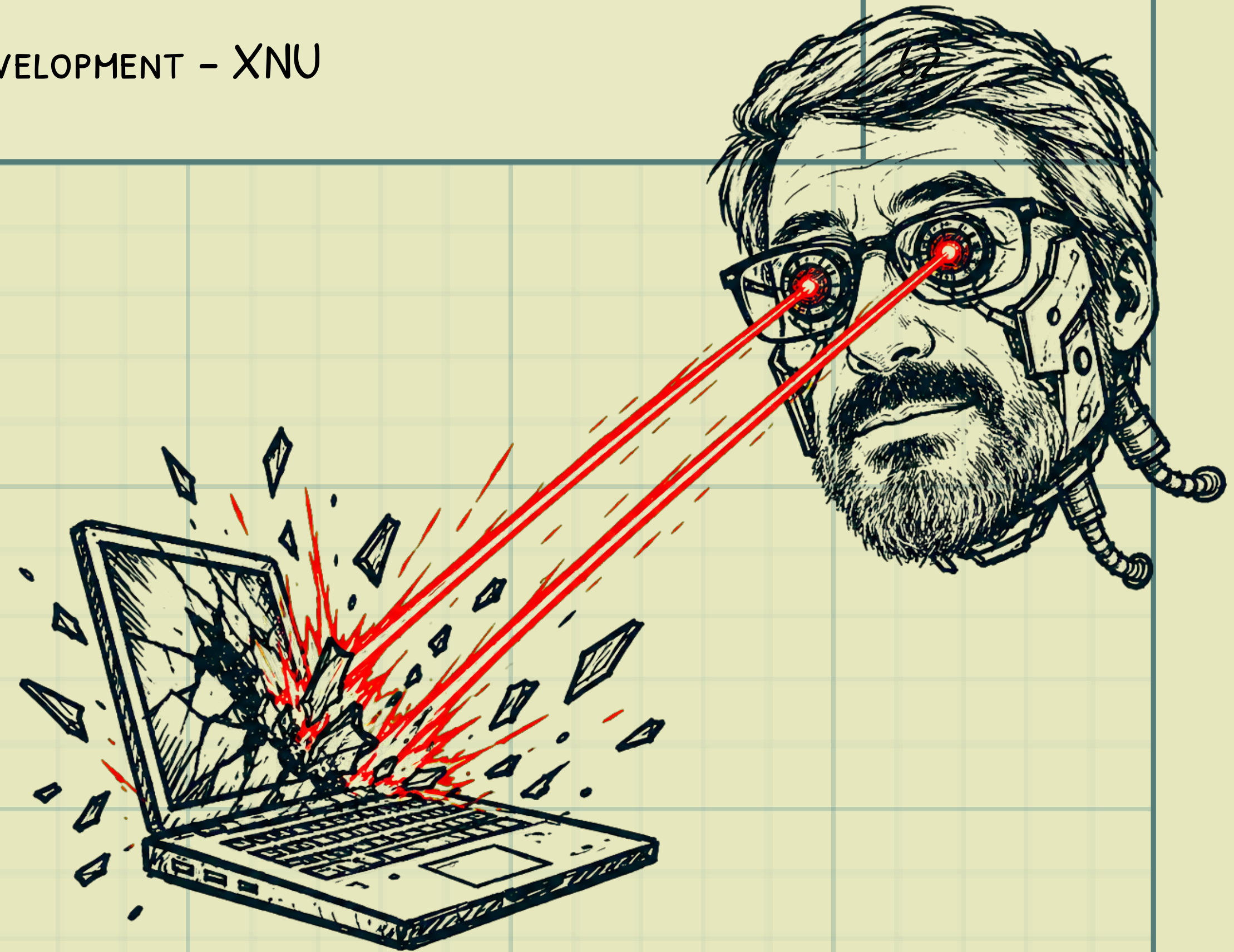
“Keep going, you’ve almost got root!”



TAKEAWAYS, OBSERVATIONS, AND .:~VIBES~:.



VS



1. PAIR HACKING IS STILL STRONG.



1. PAIR HACKING IS STILL STRONG. (FOR NOW?)

2. I'M NOT SAYING IT'S
THEOREM PROVING



...BUT IT'S THEOREM
PROVING



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