# CS61 Lecture 15

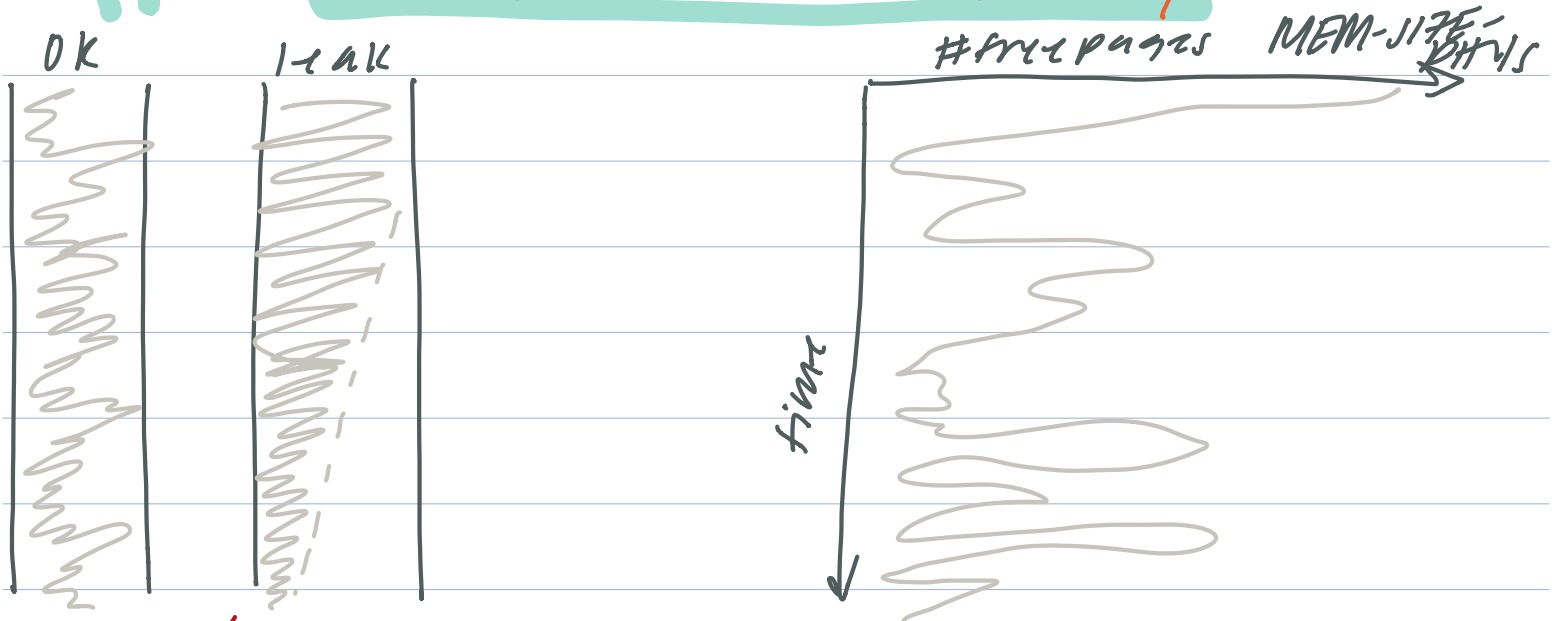
Oct 22, 2019

## Problem Set

### Looking for memory leaks

- enumerate places w/ allocated memory
  - process_setup: kalloc page table
    - free in exit by kfree
    - * nullptr = ran out of memory

IMPORTANT




- ✓ process_setup: code, data, and stack
  - freed in exit by vmiter loop
- fork: kalloc page table
  - if it succeeds, freed in exit w/ kfree
  - if it doesn't end, we can exit if we design it properly (can exit a half-initialized process)
- ✓ fork: code/data/stack
  - freed in exit/cleanup w/ vmiter loop
- ✓ sys_page_alloc (heap)
  - freed on exit w/ vmiter loop

- vmiter::map
  vmiter::tmp_map } page table tables

freed in exit w/ ptiter loop

```
void *ptr = kalloc(PAGESIZE);   // destination physical page
if(!ptr) { exit(); ... }
int r = vmiter(current, va).tmpmap(ptr, PTE_P|W|V);   // if tmpmap fails, this page goes out of scope and cannot be referenced again
if(r(0)) { exit(), ... }
```

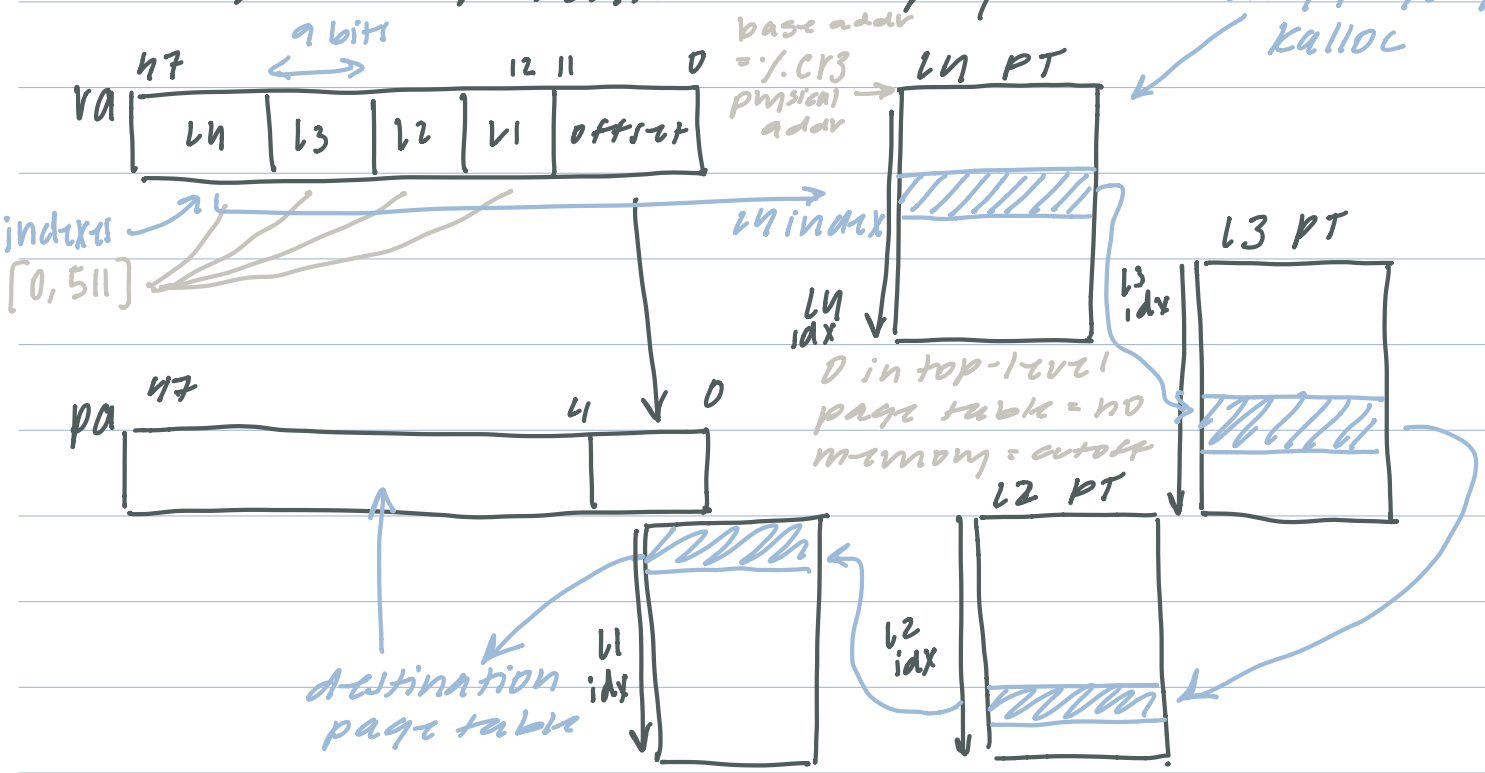
*** use tmpmap for steps 3-7 ***

IMPORTANT

# last kernel lecture

## x86-64 page tables

1. chunking: page sized units ($2^{12}$ B)
2. high-level cutoff: large empty addr space regions representable cheaply



* based on page size, addr / page, physical addr

↓ $2^{12}$ bytes

↳ $2^{12}/2^3 = 2^9$

↓ x86-64 = 8 bytes = $2^3$ bytes

virtual address → 48 meaningful bits: 12 offset + 36 indexes

$$\frac{\text{total index bits}}{\text{\# bits per level}} = \frac{36}{9} = 4 \text{ page levels}$$