



LINUX
SECURITY
SUMMIT

v1

Exploiting race conditions on [ancient] Linux

Jann Horn, Google Project Zero

(if this text is too small for you to read, maybe open the slides on your laptop)

slides at: <https://sched.co/TynD>

Introduction

- bugs described here have been fixed for a long time
- all exploits against kernel 4.4
- focus on exploitation techniques, not impact of individual bugs

Agenda

- physical-page use-after-free via stale TLB [bug 1]
 - [kernel bug, PoC for Google Pixel 2]
 - buddy allocator
 - preemption and scheduler control
- refcount decrement on struct file [bug 2]
 - [kernel bug, PoC for Ubuntu 16.04]
 - userfaultfd() and FUSE
 - kcmp()
- a poor substitute for FUSE/userfaultfd [bug 3]
 - [userspace bug, PoC for Google Pixel 2]
 - i_mutex on kernel 4.4
 - priority inversion
 - repeated file mapping faults



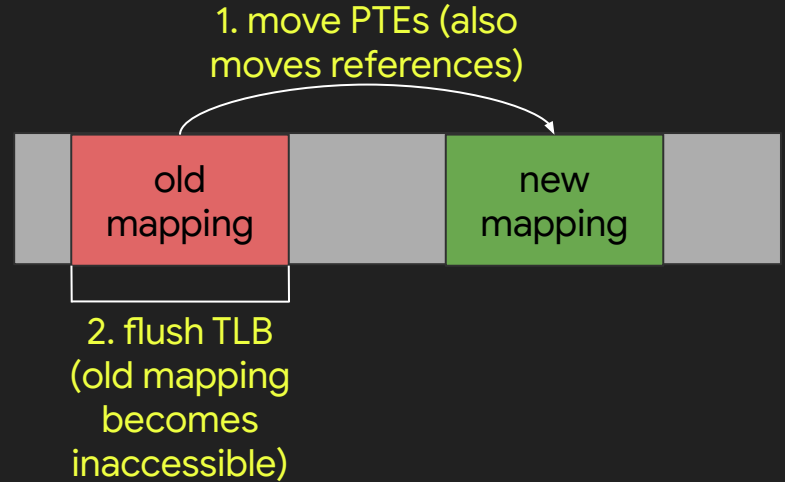
Bug 1: `mremap()`+`fallocate()` race

Translation Lookaside Buffer (TLB)

- in-CPU cache for page table entries (PTEs)
- PTEs are essentially refcounted page pointers
- TLB borrows references from PTEs
- kernel can invalidate TLB for virtual address ranges
 - x86: IPI for remote CPUs
 - arm64: magic system-wide TLBI instruction

mremap(): moving a memory mapping

- moves associated page table entries (PTEs)
- has to flush the TLB for the old address range



`fallocate()`: (de)allocate space for a file

- interesting case: punch a hole in a file
- file pages in the hole are:
 - yanked out of all mappings (in all processes)
 - released once all references are gone

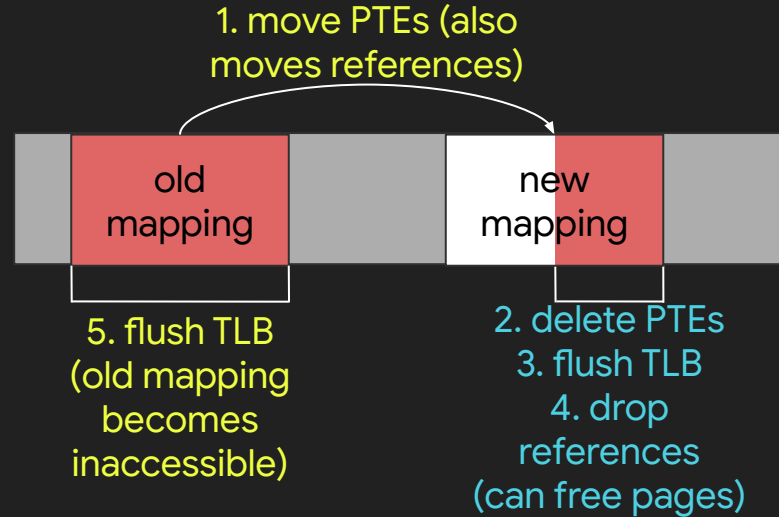


1. delete PTEs
2. flush TLB
3. drop references

Bug 1: `mremap()`+`fallocate()` race

crbug.com/project-zero/1695

- `mremap()` holds no relevant lock between **moving PTEs** and **TLB flush**, `fallocate()` possible in between
- `fallocate()` drops page references after its TLB flush
- stale TLB entry for old mapping permits **physical-page use-after-free** between **dropping page reference** and **flushing TLB for old mapping**



Exploit plan: Basics

- biggest impact on Linux <4.9; exploiting for *write* access is much harder on newer kernels
- goal: Pixel 2 (Linux 4.4) exploit
- exploit idea: reallocate freed page with kernel data



TLB flush pending
pages freed
=> physical UAF!

Buddy allocator

(highly simplified, not entirely correct)

percpu freelist [with UAF page]
cpu X
MIGRATE_MOVABLE

Page freelist
order 0
MIGRATE_MOVABLE

Page freelist
order 1
MIGRATE_MOVABLE

Page freelist
order 2
MIGRATE_MOVABLE

Page freelist
order 0
MIGRATE_UNMOVABLE

Page freelist
order 1
MIGRATE_UNMOVABLE

Page freelist
order 2
MIGRATE_UNMOVABLE

percpu freelist
cpu X
MIGRATE_UNMOVABLE

SLAB
e.g. kmalloc-256

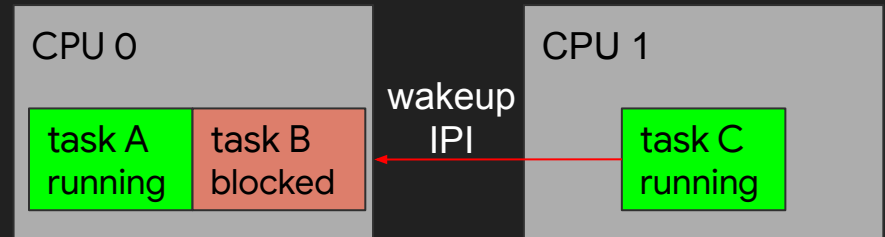
Exploit plan

- biggest impact on Linux <4.9; exploiting for *write* access is much harder on newer kernels
- goal: Pixel 2 (Linux 4.4) exploit
- ~~exploit idea: reallocate freed page with kernel data - ✘~~, looked too messy
- exploit idea: reallocate freed page as page cache for privileged code ✓
 - requires disk I/O within the race window
 - **need to make the mremap() race window wide enough for disk I/O**
- race window detectable through procfs



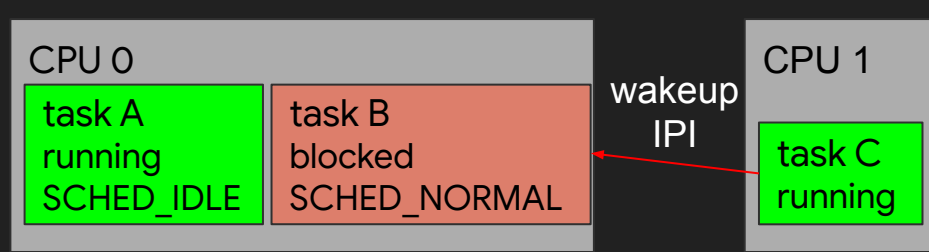
TLB flush pending
pages freed
=> physical UAF!

Preemption



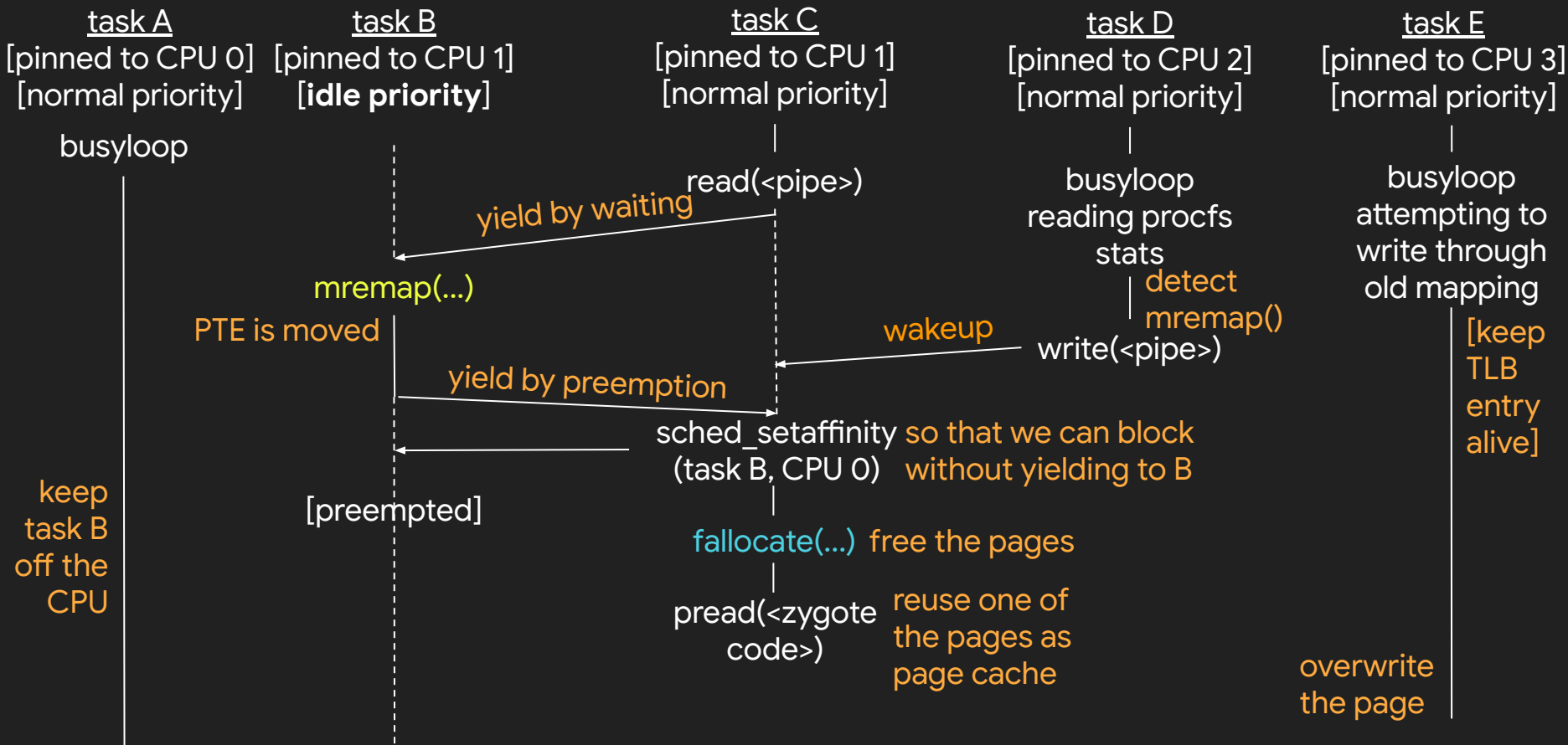
- waking up a task can cause a scheduler Inter-Processor Interrupt (IPI)
 - depending on policy, priority and past CPU usage [see `check_preempt_wakeup()`]
- Linux supports three kernel preemption models:
 - "voluntary" preemption can yield the CPU at `cond_resched()` [called in ~1000 places]
 - used by many Linux distributions by default
 - full preemption
 - enabled on Android
 - syscall context interruptible directly via inter-processor interrupt (IPI) on task wakeup
 - no preemption in some code regions (holding a spinlock [/ preemption explicitly disabled / interrupts disabled])
 - [preemption requests in critical region are delivered on critical section exit]
 - mutexes don't block preemption!

Scheduler control

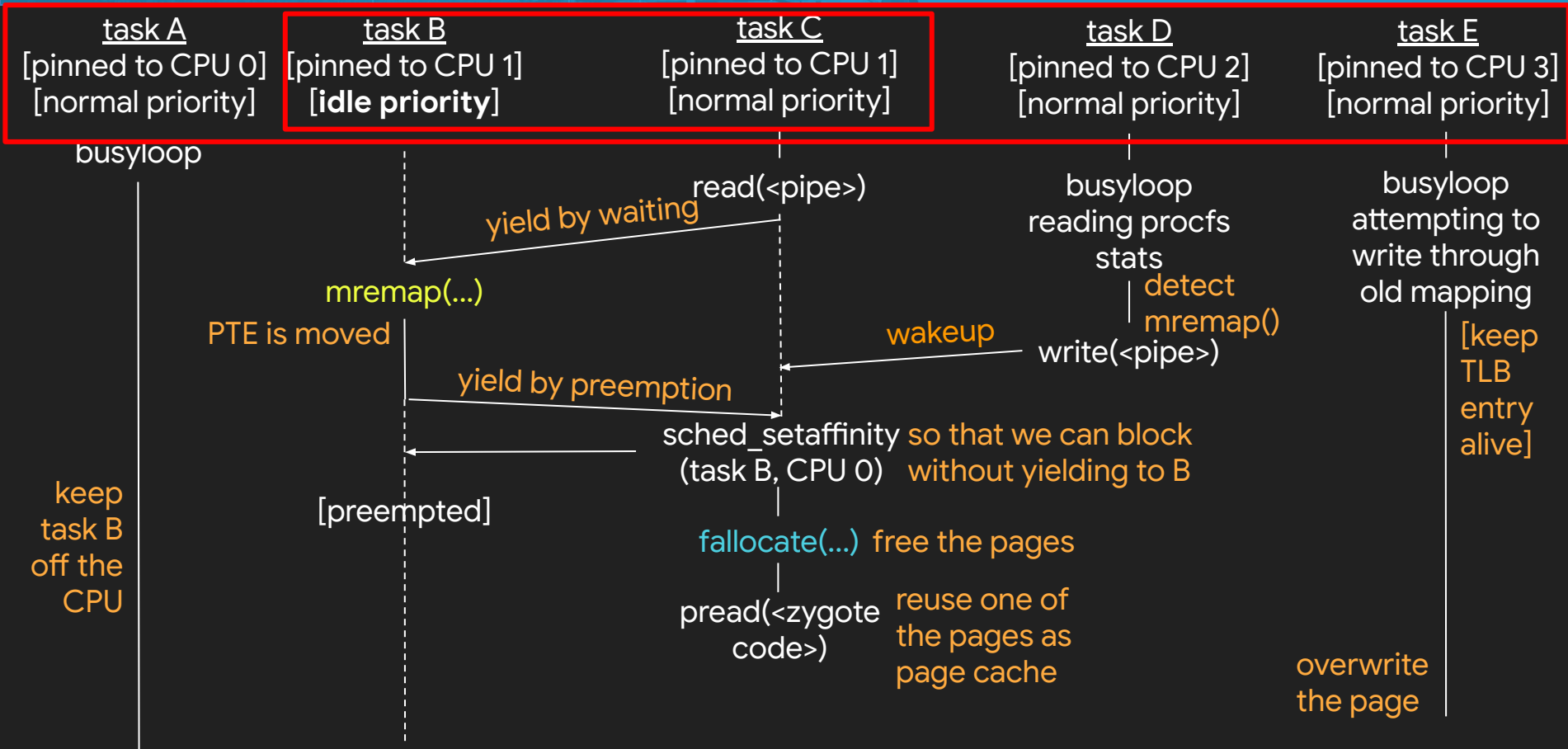


- `sched_setscheduler()`: set `SCHED_NORMAL` / `SCHED_IDLE`
 - [realtime policies require `CAP_SYS_NICE` or `RealtimeKit`]
 - on busy CPU, `SCHED_IDLE` has infrequent wakeups
 - `SCHED_IDLE` never preempts
 - `sched_setaffinity()`: pin task to CPU bitmask
 - also affects execution in kernel mode!
- pin two own tasks to a single CPU
- set different scheduling classes
- interrupt kernel code execution

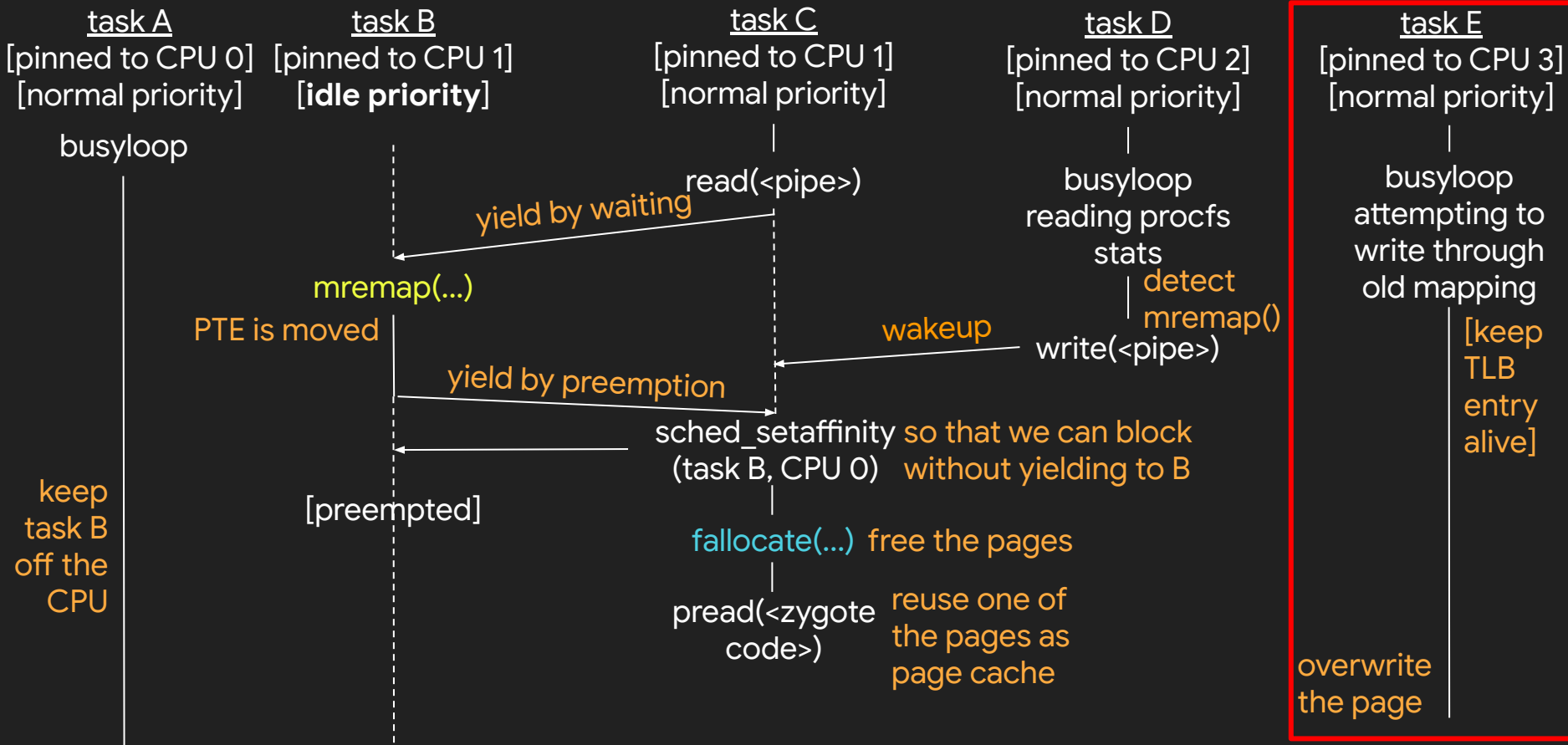
Android kernel exploit (app -> zygote)



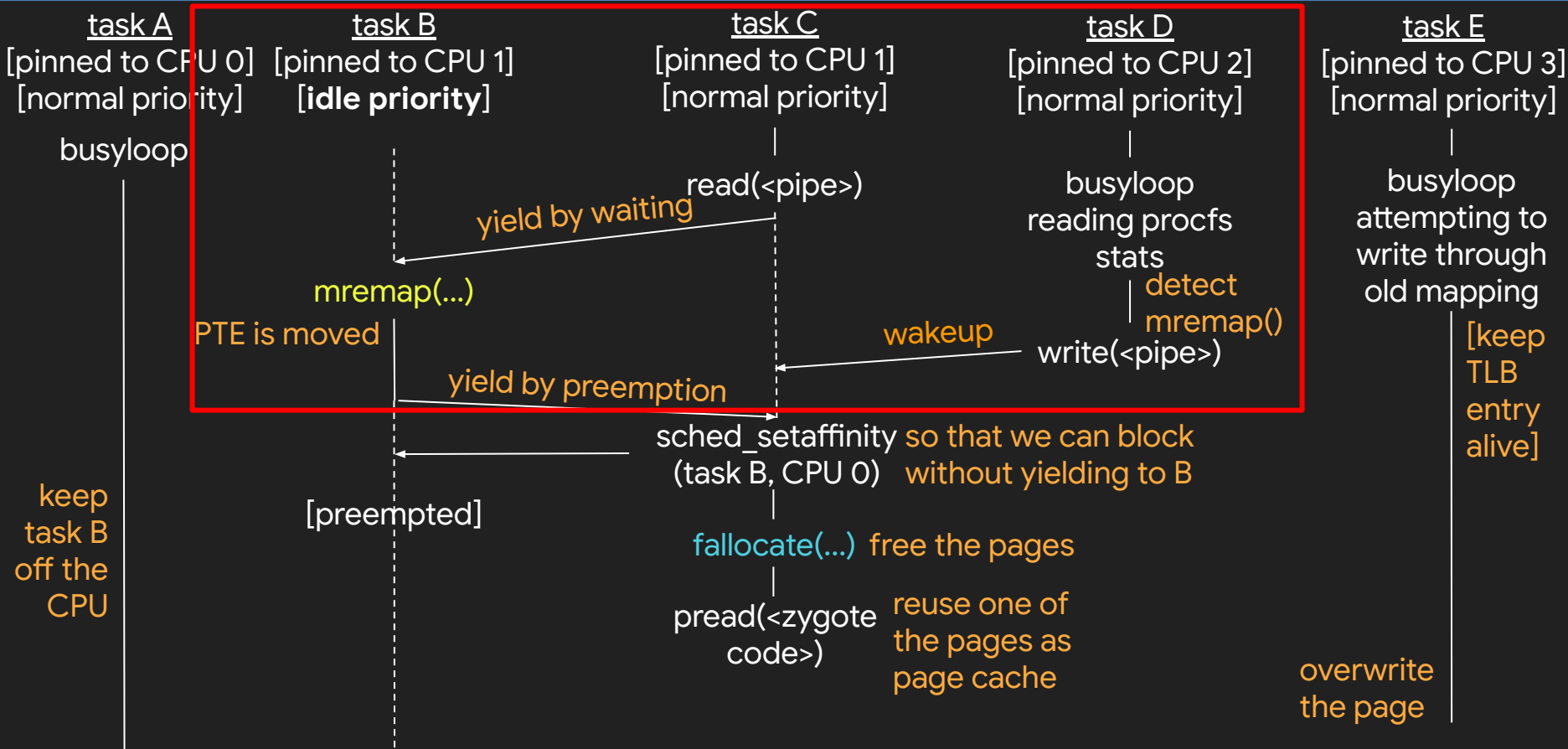
Android kernel exploit (app -> zygote)



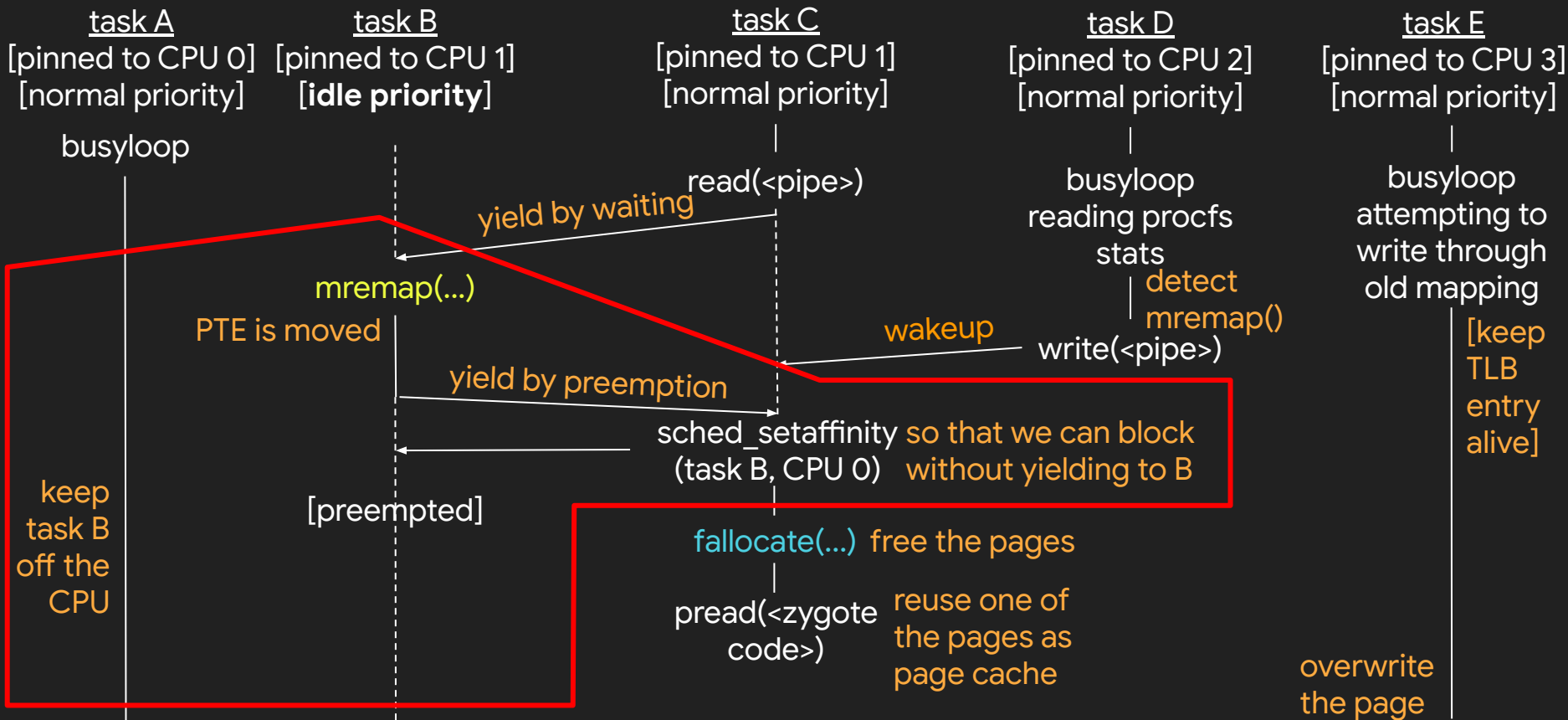
Android kernel exploit (app -> zygote)



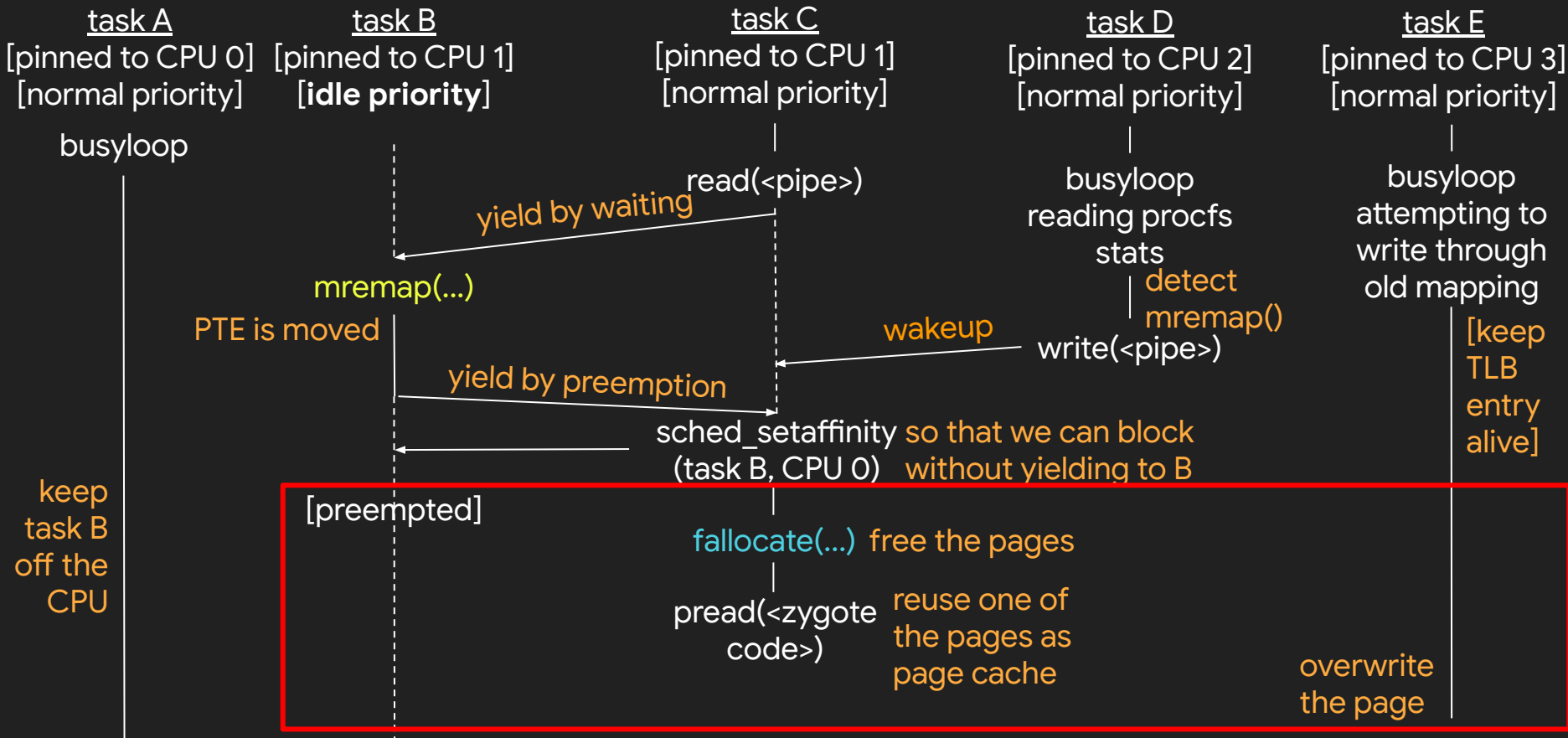
Android kernel exploit (app -> zygote)



Android kernel exploit (app -> zygote)



Android kernel exploit (app -> zygote)





Bug 2: recount decrement on struct file

(yes, the bug doesn't involve a race condition, but the exploit kinda does)

userfaultfd and FUSE

- userfaultfd() and FUSE allow userspace to synchronously handle page faults
- => userspace can block arbitrarily at copy_from_user()/copy_to_user()
- userfaultfd() and FUSE are not exposed to unprivileged Android code

=> not applicable on Android, but relevant on desktop Linux

FUSE for exploiting struct file refcount overdecrement in Linux 4.4

bug from 2016 to illustrate FUSE-based use-after-free exploitation

- file reference acquired with `fdget()`
- error path accidentally called `fdput()` twice
- struct file freed prematurely
- use-after-free
- exploited on Ubuntu 16.04

```
f = fdget(insn->imm);
map = __bpf_map_get(f);
if (IS_ERR(map)) {
    verbose("fd %d is not pointing to valid
bpf_map\n", insn->imm);
    fdput(f);
    return PTR_ERR(map);
}
struct bpf_map * __bpf_map_get(struct fd f)
{
    if (!f.file)
        return ERR_PTR(-EBADF);
    if (f.file->f_op != &bpf_map_fops) {
        fdput(f);
        return ERR_PTR(-EINVAL);
    }
    return f.file->private_data;
}
```

kcmp() for reliable UAF

- CONFIG_CHECKPOINT_RESTORE
- **smaller/equal/greater** comparison between **permuted** kernel pointers
- intended for grouping same-object references in $O(n \log(n))$
- works on:
 - **struct file**
 - struct mm_struct
 - struct files_struct
 - struct fs_struct
 - struct sighand_struct
 - struct io_context
 - struct sem_undo_list
- tag reuse oracle for Memory Tagging unless tag bits are ignored

```
static long kptr_obfuscate(long v, int type)
{
    return (v ^ cookies[type][0]) *
    cookies[type][1];
}
```

```
static int kcmp_ptr(void *v1, void *v2, enum
kcmp_type type)
{
    long t1, t2;

    t1 = kptr_obfuscate((long)v1, type);
    t2 = kptr_obfuscate((long)v2, type);


    return (t1 < t2) | ((t1 > t2) << 1);
}
```

FUSE for exploiting struct file refcount overdecrement in Linux 4.4

- create FUSE mapping
- open writable file (/dev/null)
- start `writev()` with iov in FUSE mapping
- `write mode check` passes
- `import_iovec()` stalls on page fault
- trigger bug to free the file
- open `/etc/crontab` as read-only
- verify that struct file was allocated at the same address with `kcmp()` (else re-open `/etc/crontab`)
- resolve FUSE page fault
- `writev()` **writes into `/etc/crontab`**

```
ssize_t vfs_writev(struct file *file, const
struct iovec user *vec, [...]) {
    if (!(file->f_mode & FMODE_WRITE))
        return -EBADF;
    [...]
    return do_readv_writev(WRITE, file, vec,
vlen, pos);
}

static ssize_t do_readv_writev(int type,
struct file *file, const struct iovec __user
* uvector, unsigned long nr_segs, loff_t
*pos) {
    [...]
    ret = import_iovec(type, uvector, nr_segs,
        ARRAY_SIZE(iovstack), &iov, &iter);
    [...]
    if (iter fn)
        ret = do_iter_readv_writev(file, &iter,
pos, iter_fn);
    [...]
}
```





Bug 3: use of `getpidcon()`

int getpidcon(pid_t pid, char **context)

- userspace daemons need to check peer SELinux contexts
- unix domain sockets: SO_PEERSEC
- Android binder: until recently no context name, only sender PID

```
fd = open("/proc/$pid/attr/current", O_RDONLY)
read(fd, buf, len)
```

Bug 3: race condition in hw servicemanager

crbug.com/project-zero/1741

- receive binder IPC call (with caller PID)
 - `getpidcon(pid, &context)`
 - ACL check for `context`
- exit and make privileged thread reuse the PID
- race window can be widened to ~15s

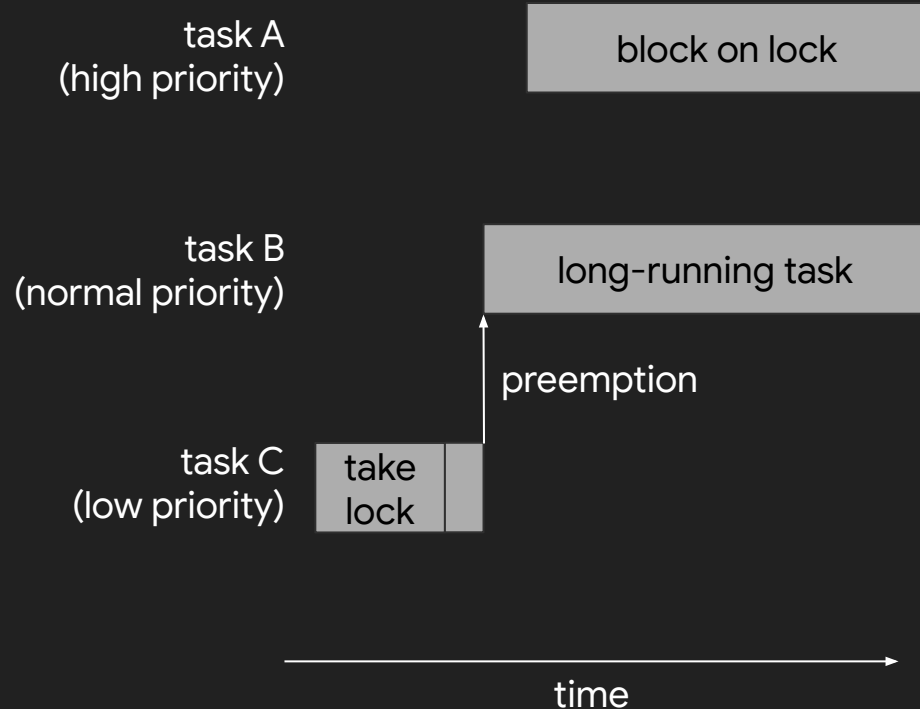
i_mutex on kernel 4.4

- `sys_getdents()` (for `readdir()`) iterates directory entries and copies to userspace under `inode->i_mutex`
 - potentially a large amount of data if the directory has many entries
- `lookup_slow()` (for looking up uncached directory entries) takes `parent->d_inode->i_mutex`
- => blocking userspace access in the middle of `sys_getdents()` blocks concurrent path traversal (e.g. `open()`) on the same inode

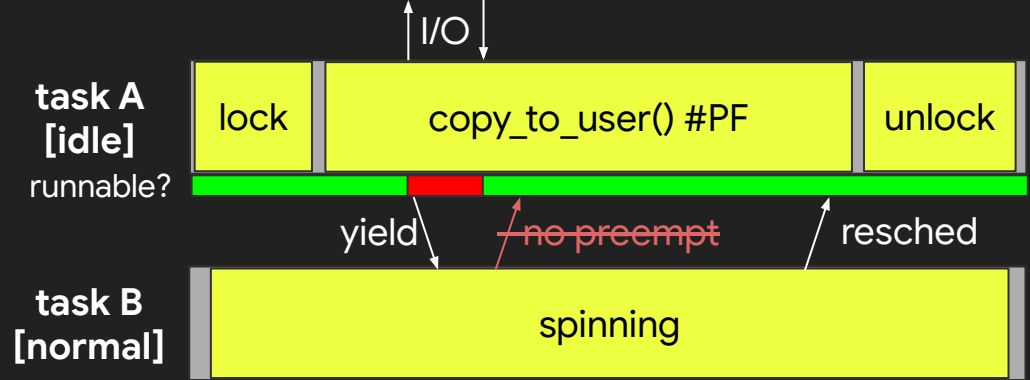
(Linux >=4.7 uses a semaphore `i_rwsem` in read mode instead of `i_mutex`)

Priority Inversion

- high-priority task blocks on mutex held by low-priority task
- low-priority task is preempted by medium-priority task (same CPU)
- also applies for violating fairness between two normal-priority tasks
- kernel mutexes are vulnerable to priority inversion!
 - (unless you're on PREEMPT_RT)
- => we can artificially create a priority inversion problem
- mitigated by infrequent idle-priority scheduling



Major faults



Instead of userfaultfd():

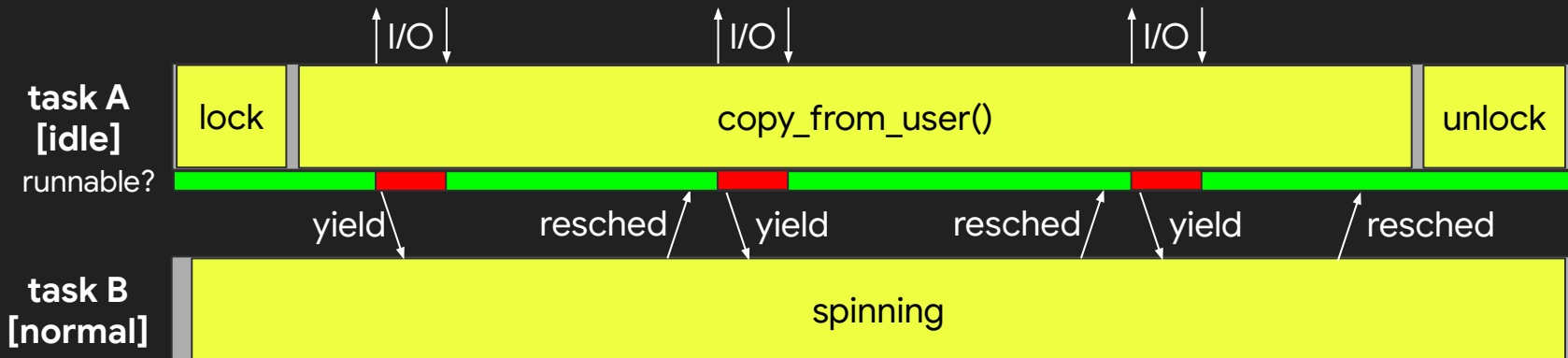
- create an uncached writable file mapping
 - [by filling up RAM with other data to force page cache eviction]
- let A trigger `copy_to_user()` on the file mapping while holding a lock
- let B spinloop at the same time

Consequences:

- `copy_to_user()` enters disk I/O path
- I/O path sleeps until disk responds, yielding the CPU
- scheduler won't preempt B when A is runnable again

Repeated file mapping faults

- [map pages such that readahead logic can't fire]
- 83560 bytes output from `sys_getdents()` = 21 pages [rounded up]
- >1s delay per disk read because of scheduler policy
- => >21s total delay






LINUX SECURITY SUMMIT

Click to edit title

- Click to edit text
 - Second level
 - Third level
 - Fourth level
 - Fifth level



**Click to place
text here**

Timing diagram

task A
[pinned to CPU 0]
[normal priority]

busyloop

task B
[pinned to CPU 0]
[**IDLE** priority]

keep
task B
off the
CPU

task C
[normal priority]

open binder

task D (simplified)
[normal priority]