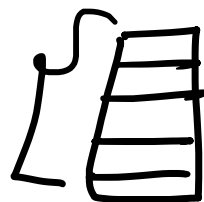


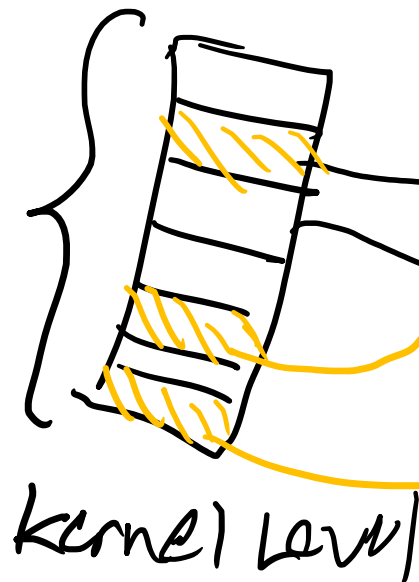
UG6



PCB

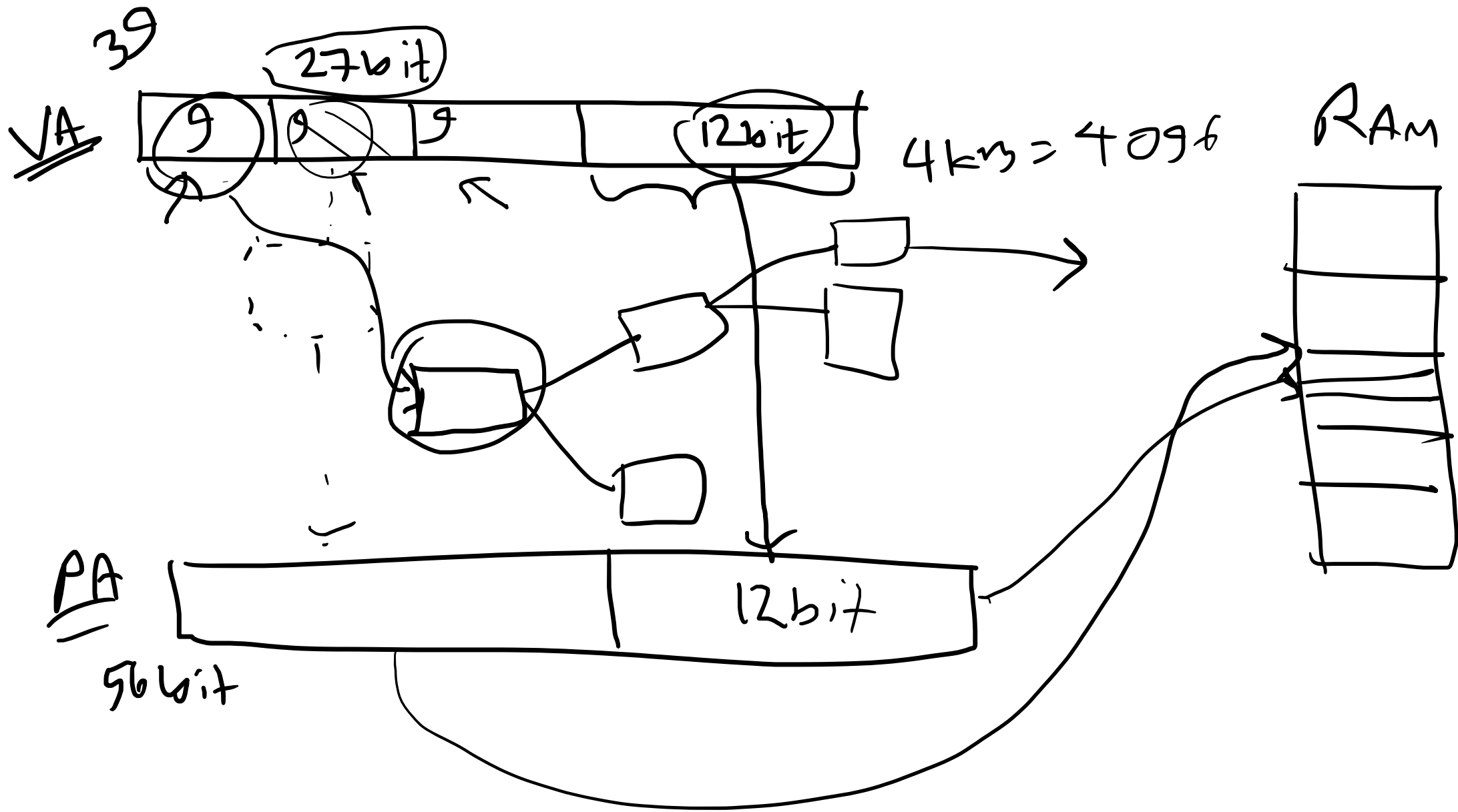
64

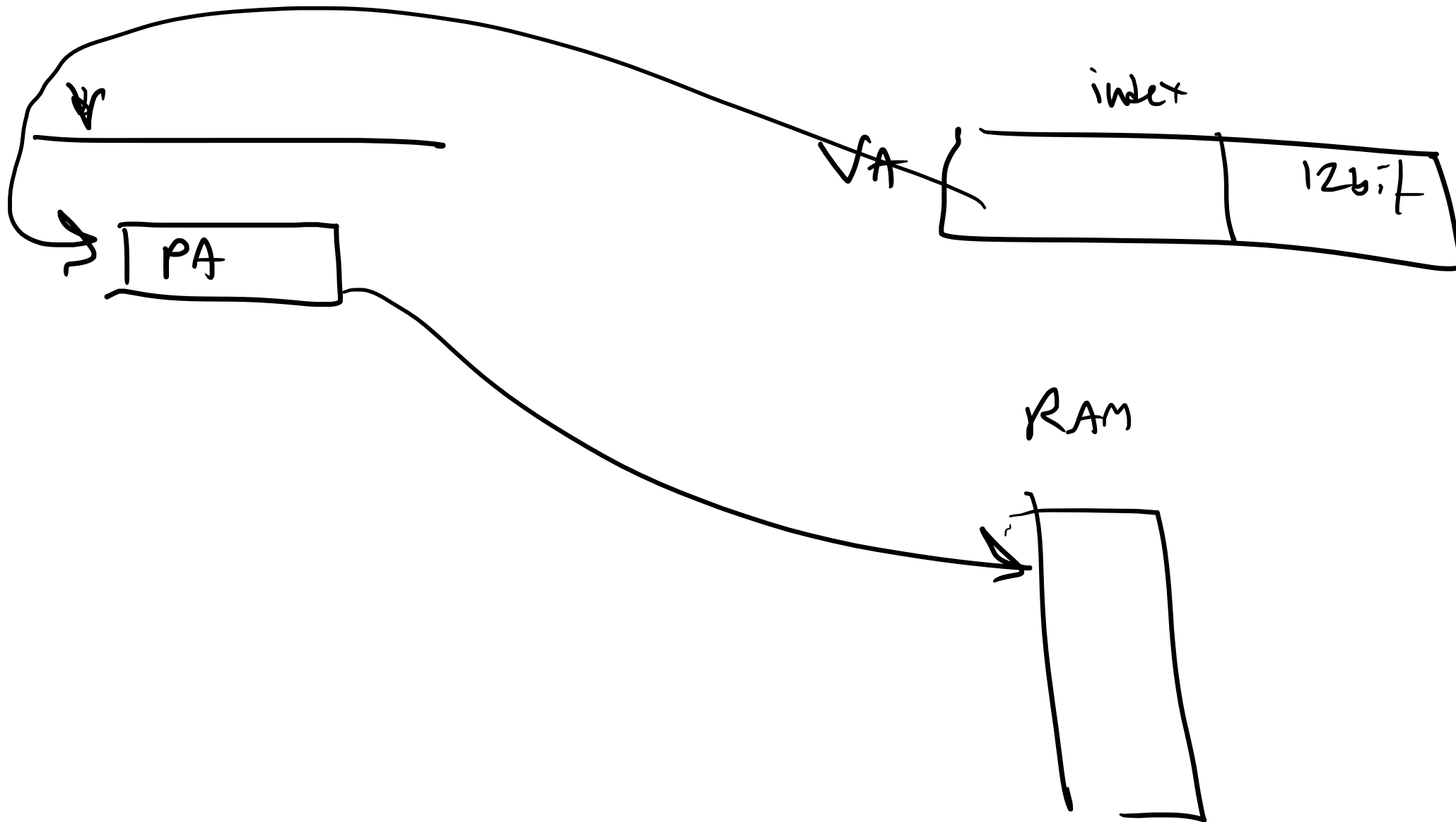
$X \sqrt{6}$
 \downarrow
 64
 $proc[N, proc]$

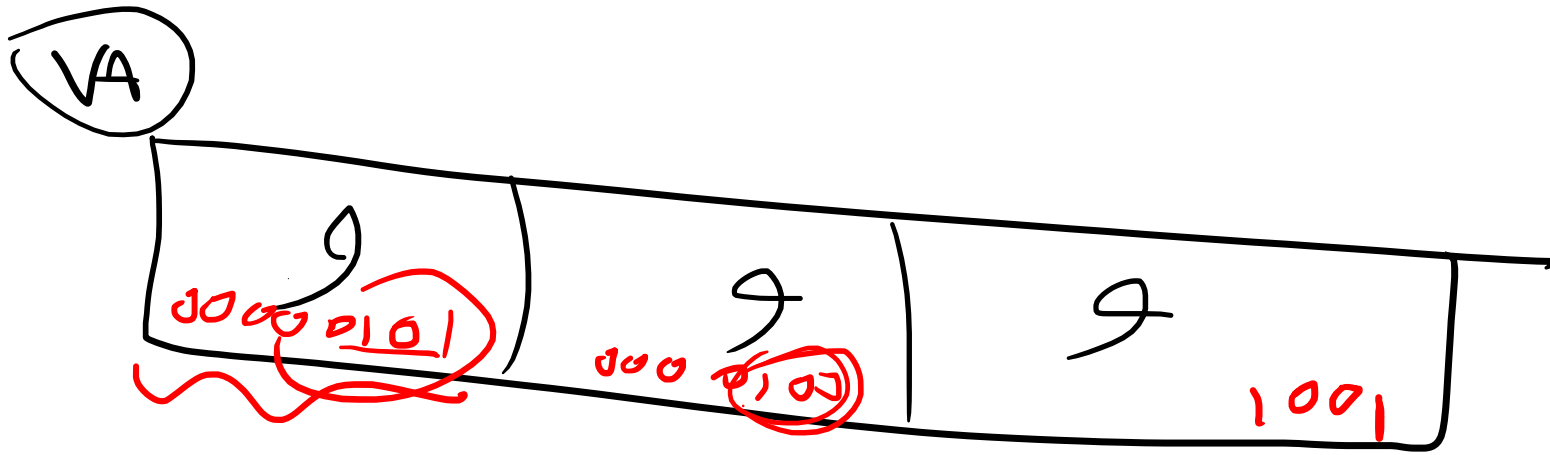


RAM

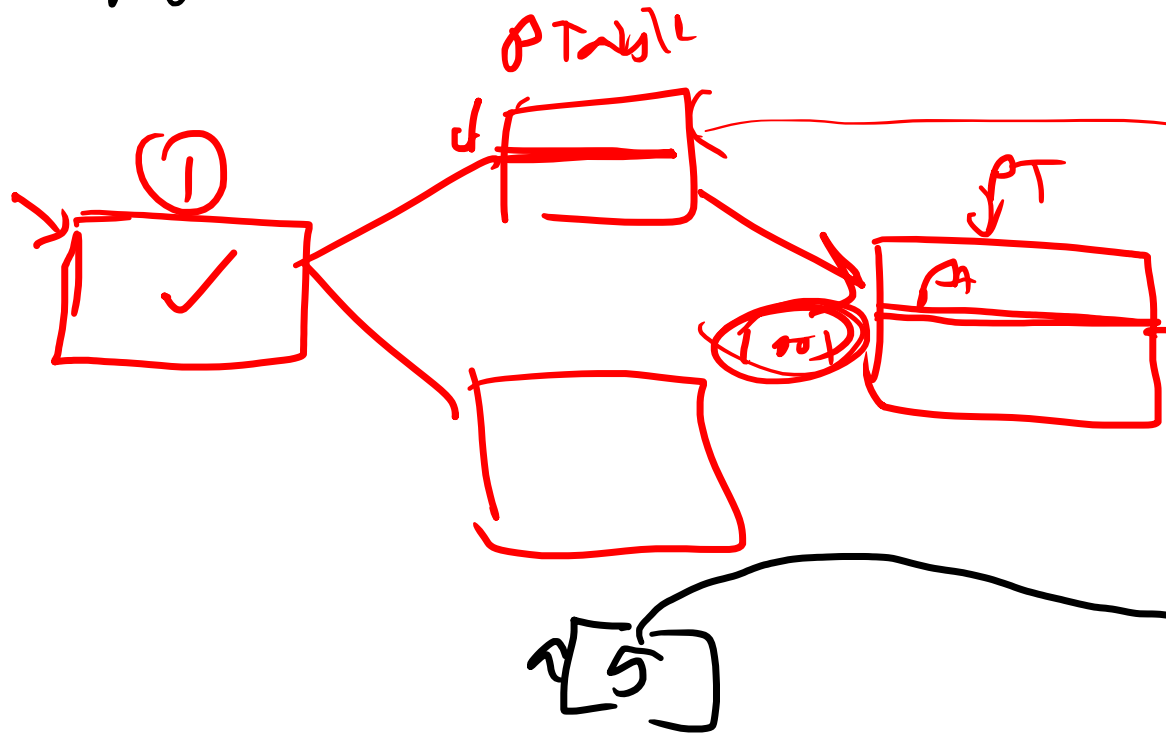




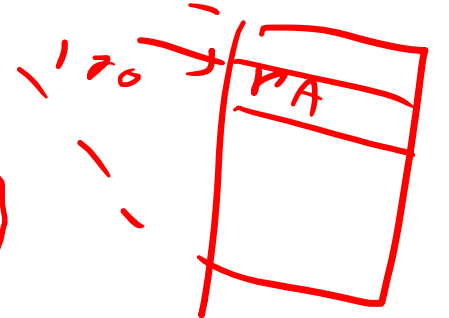
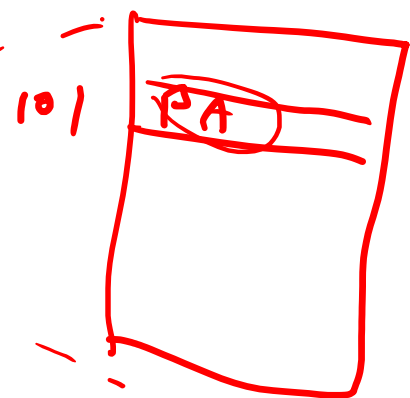




P → pagetask

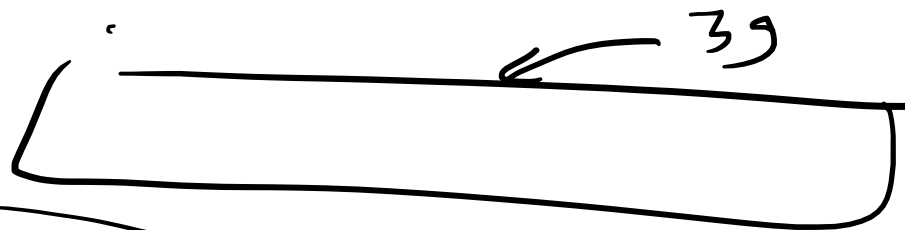


RAM

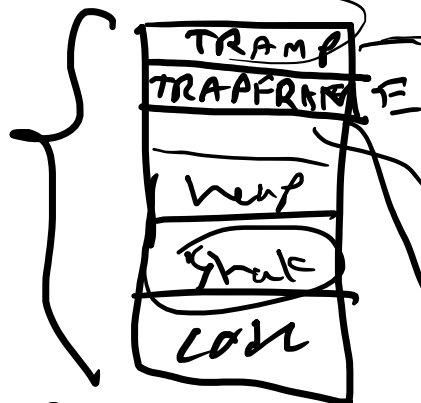


Fork

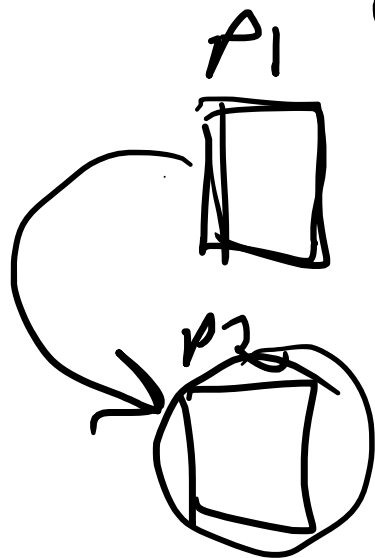
$$\text{MAXVA} = 2^{39} - 1$$



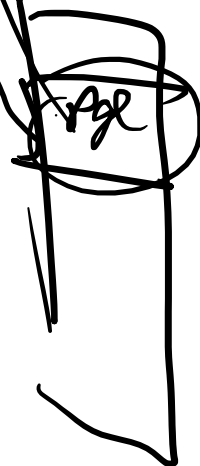
VA



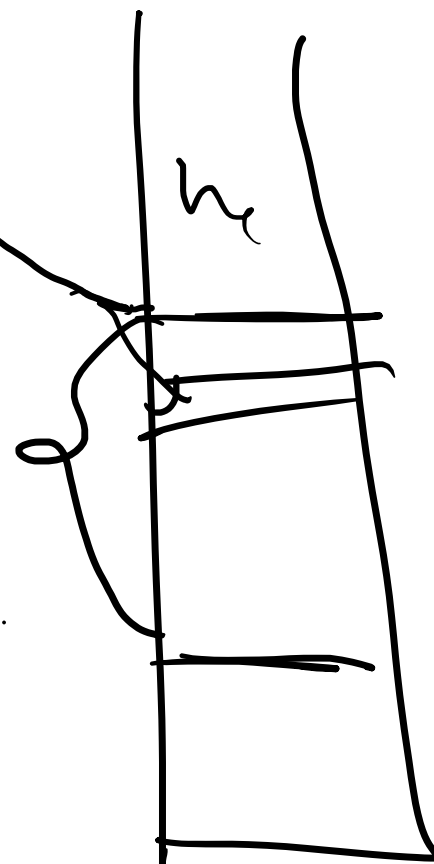
system

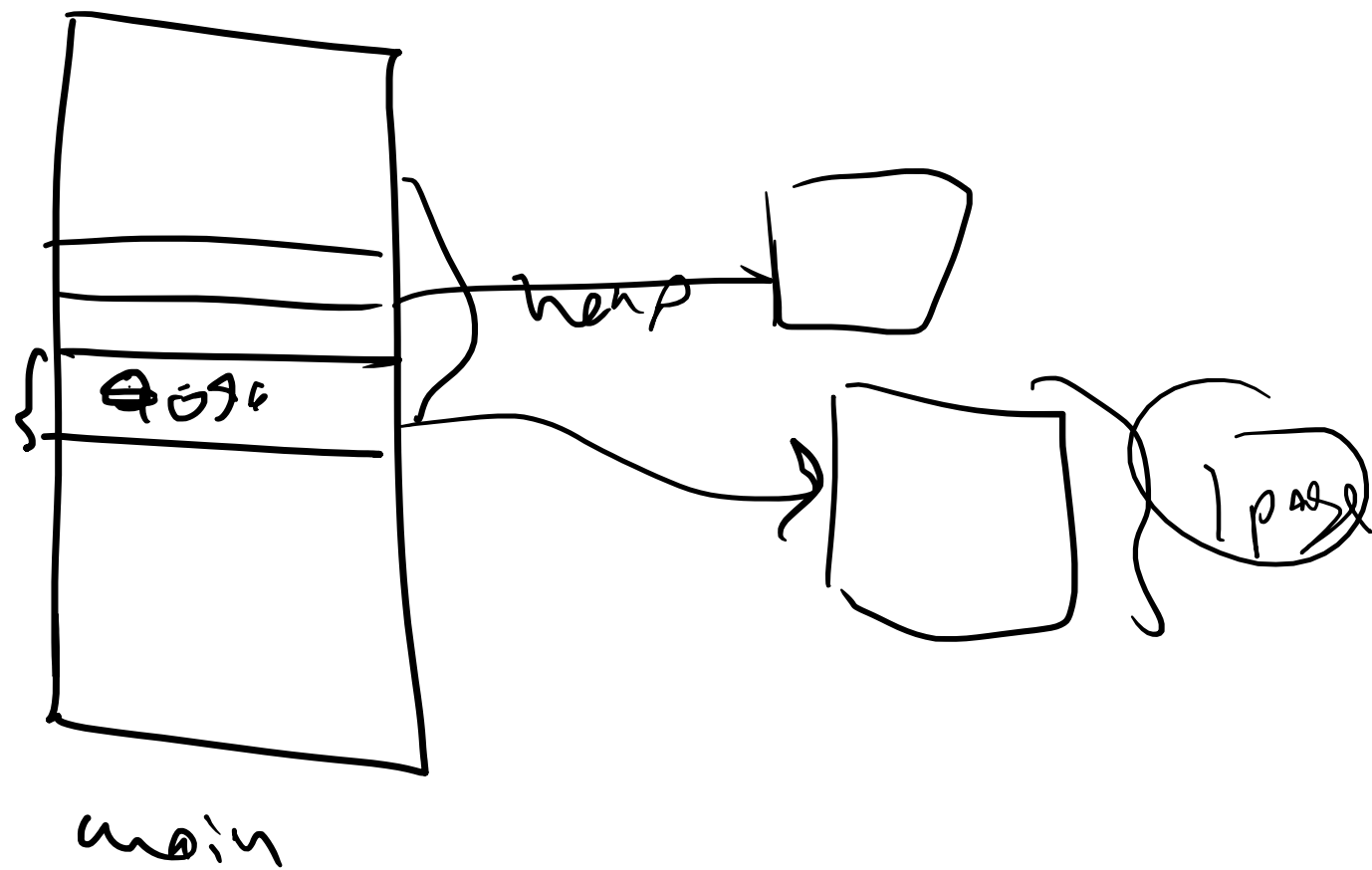


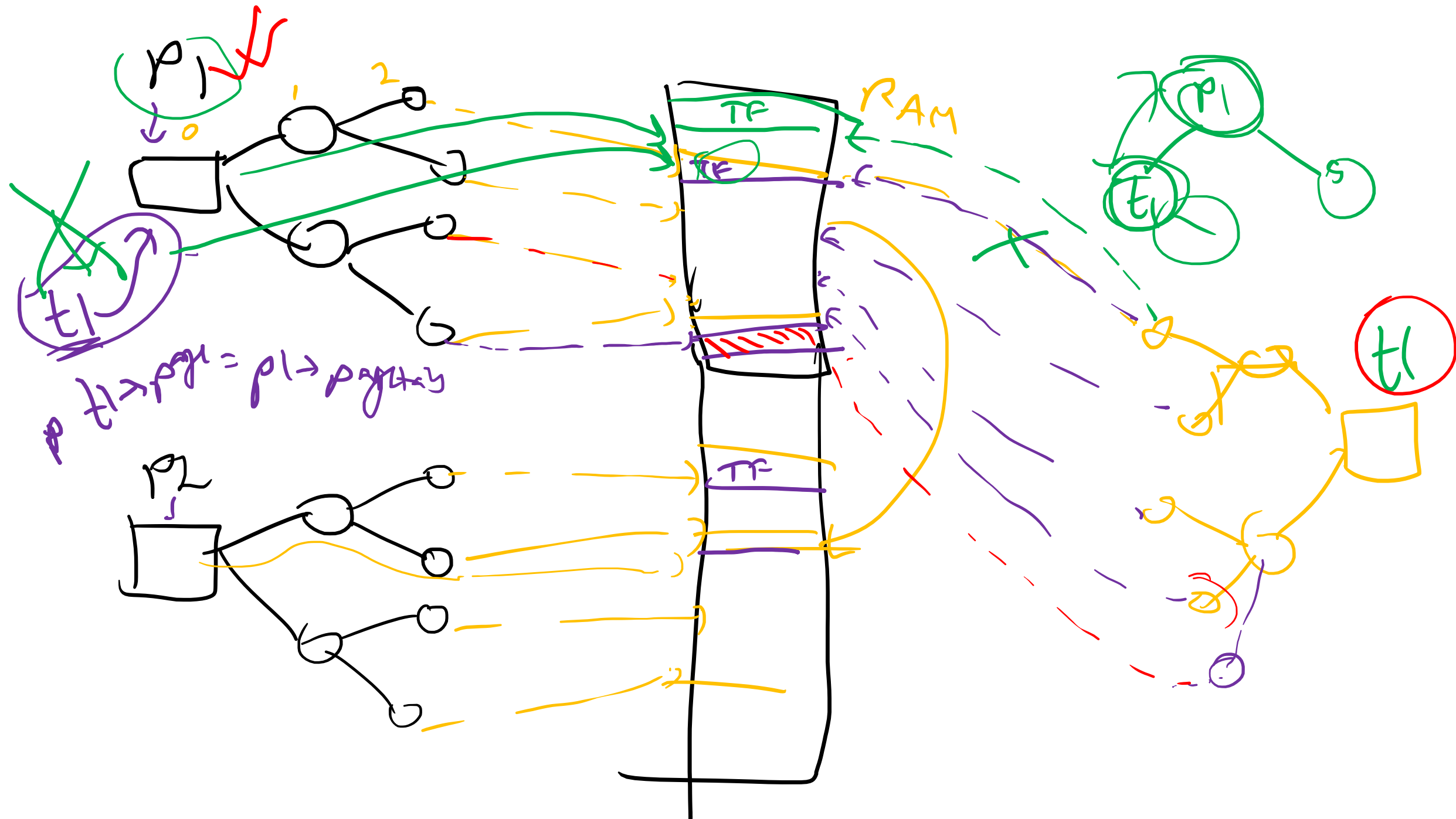
0x880000



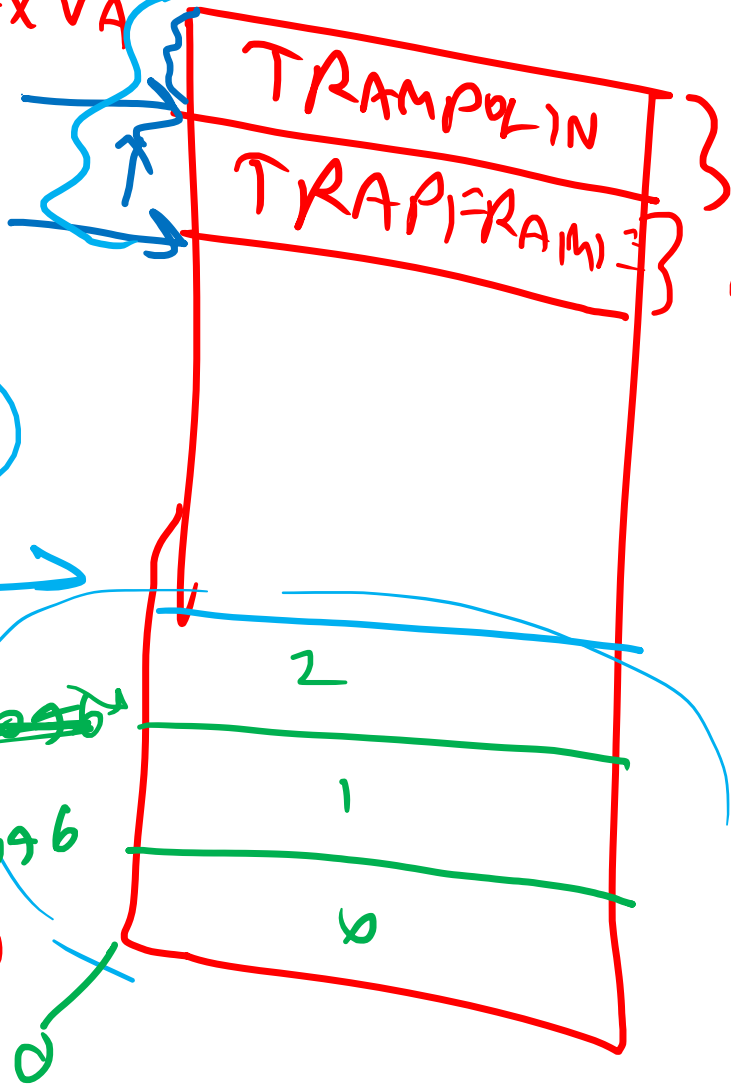
0x005000





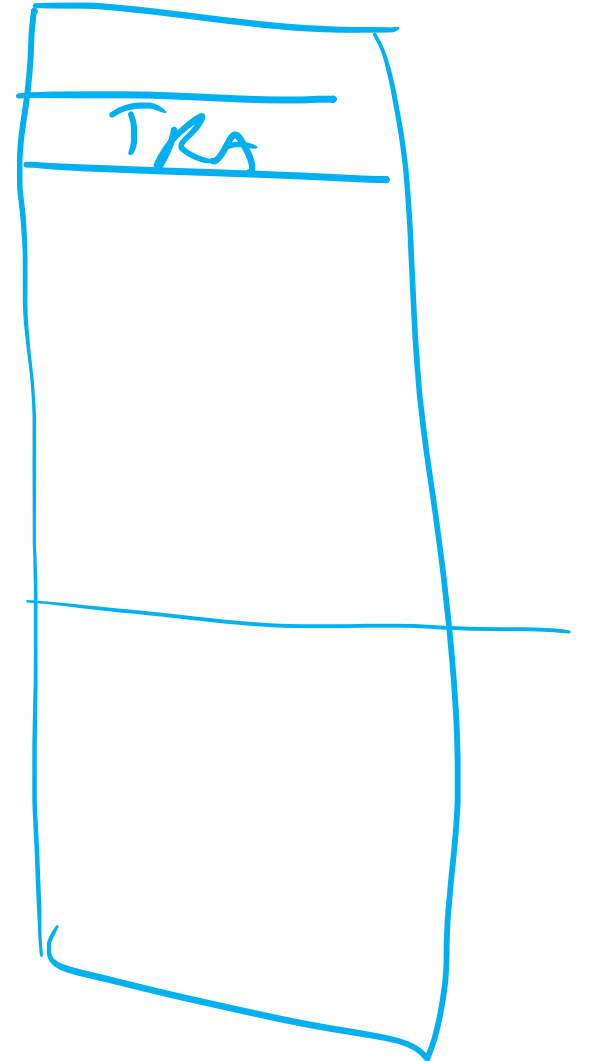


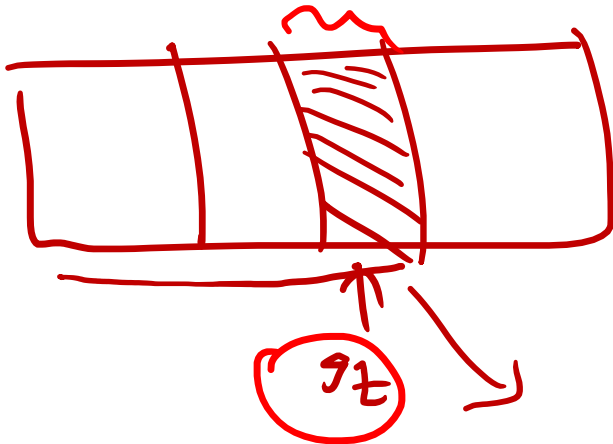
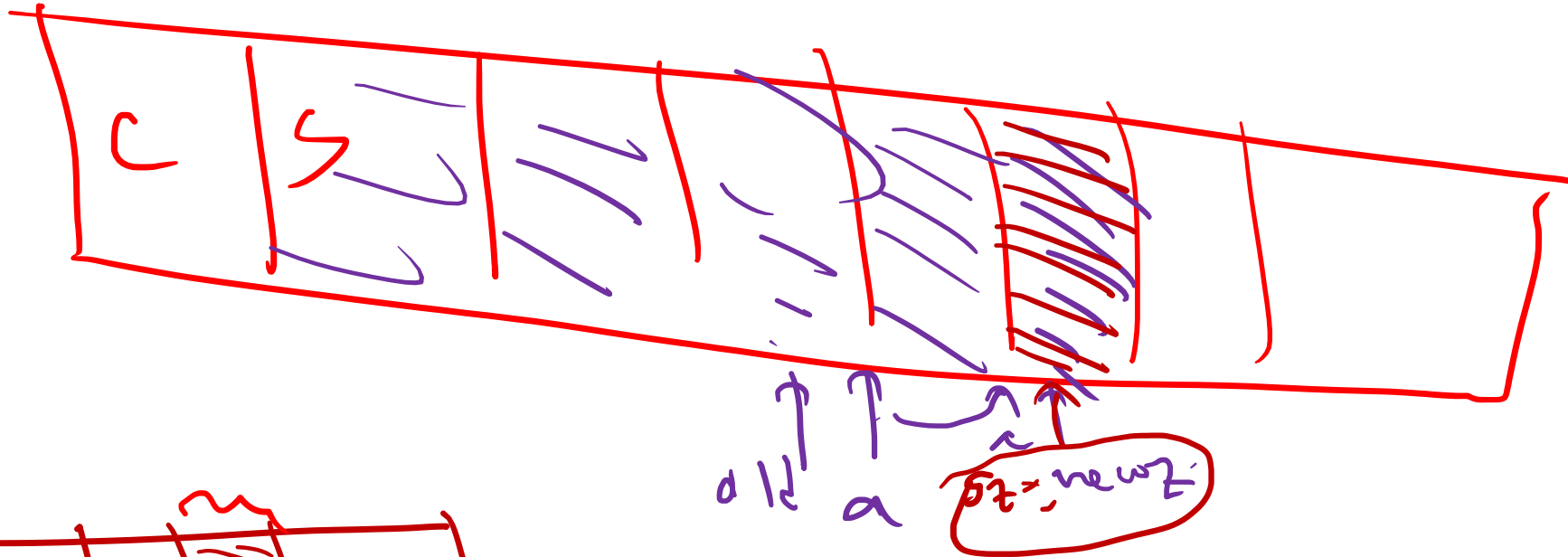
MAXVA



} PAGE SIZE = 4096

} PAGE SIZE = 4096



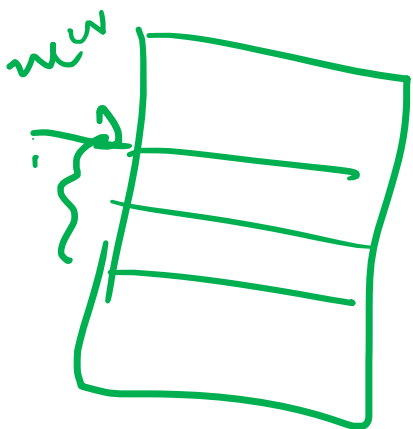
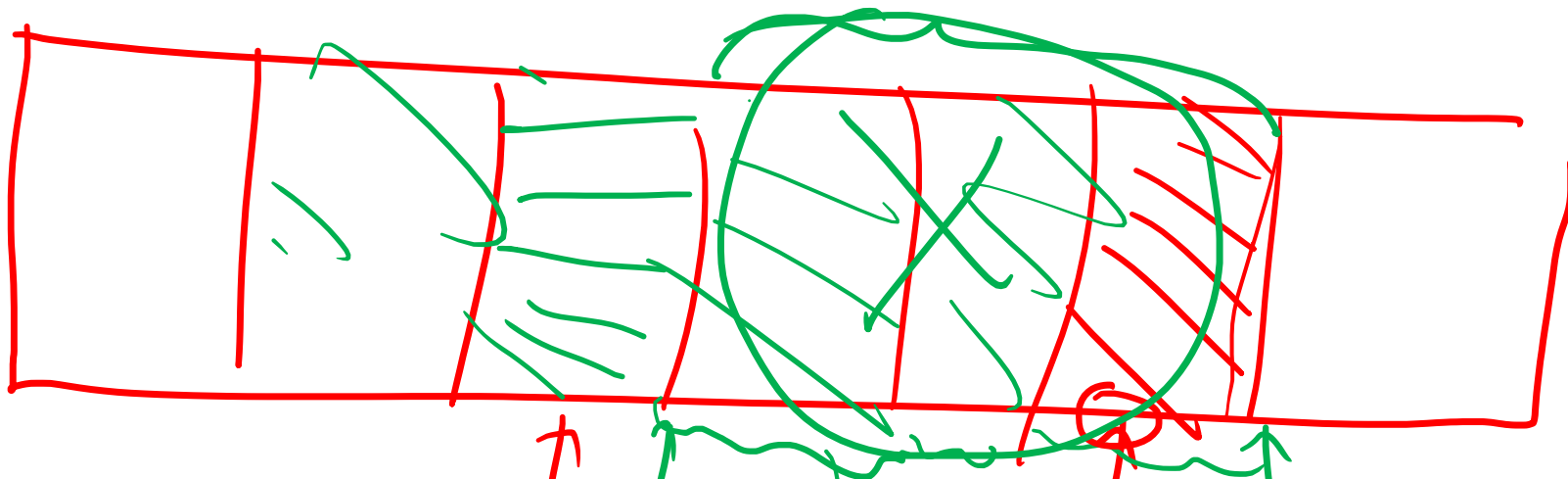


```

return oldsz;

oldsz = PGROUNDUP(oldsz);
for(a = oldsz; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
        uvmdealloc(pagetable, a, oldsz);
        return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pagetable, a, PGSIZE, (uint64)mem, PTE_R|PTE_U|xperm) != 0){
        kfree(mem);
        uvmdealloc(pagetable, a, oldsz);
        return 0;
    }
}

```



↑ newsz
↑ oldsz
↑ oldsz

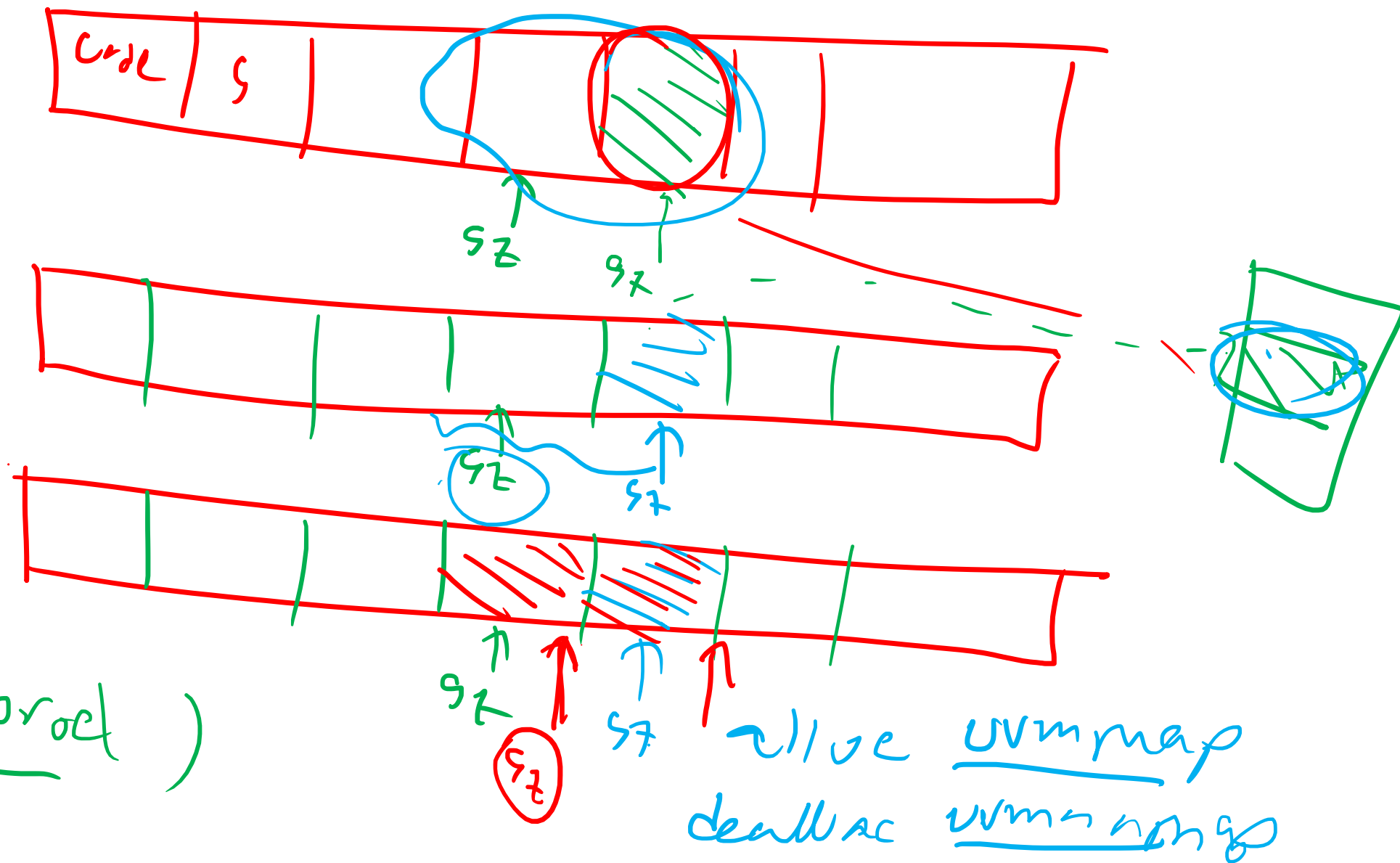
3

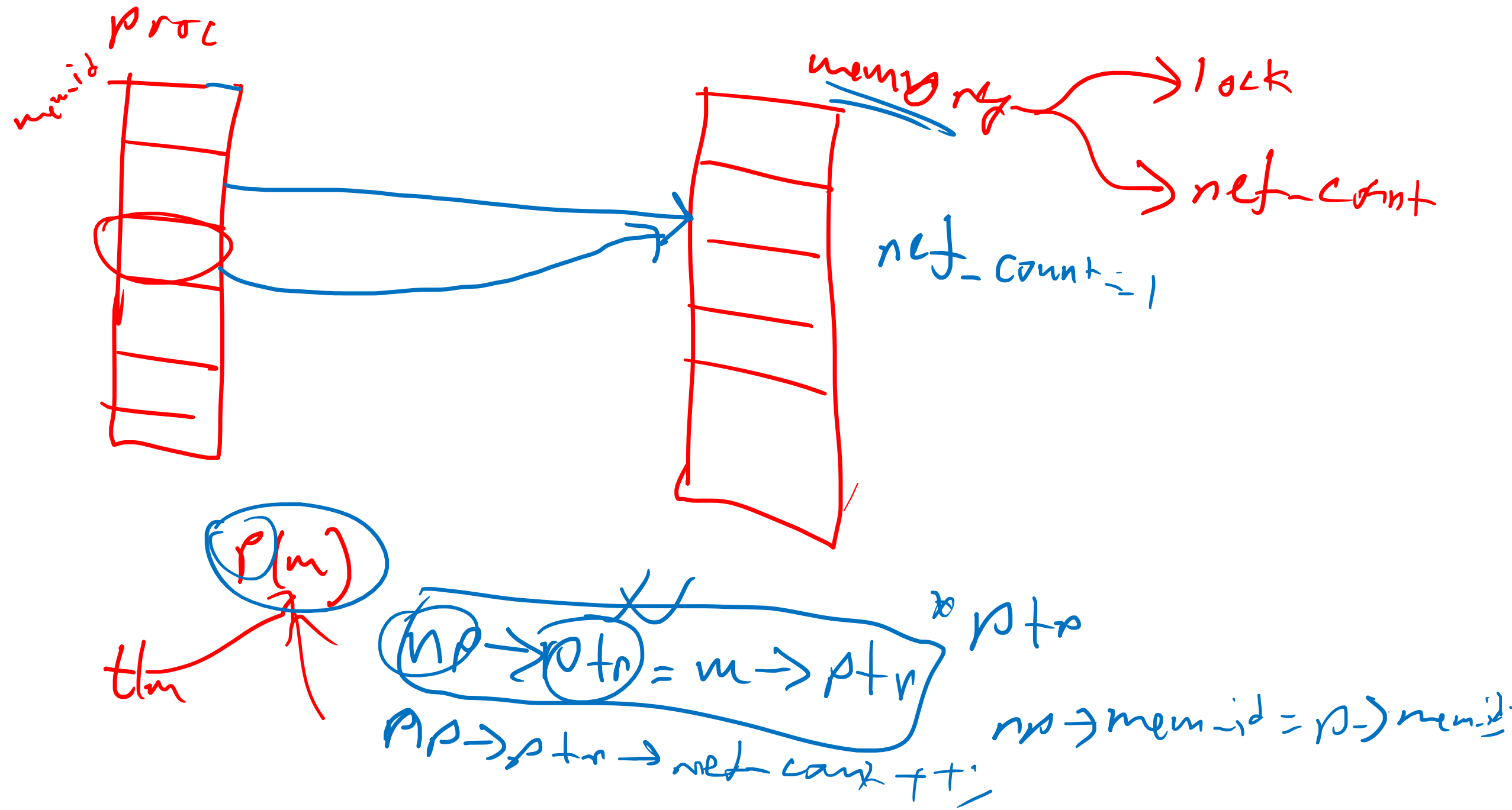
```
uint64
vmdealloc(pagetable_t pagetable, uint64 oldsz, uint64 newsz)
{
    if(newsz >= oldsz)
        return oldsz;
    if(PGROUNDUP(newsz) < PGROUNDUP(oldsz)){
        int npages = (PGROUNDUP(oldsz) - PGROUNDUP(newsz)) / PGSIZE;
        vmunmap(pagetable, PGROUNDUP(newsz), npages, 1);
    }
    return newsz;
}
```

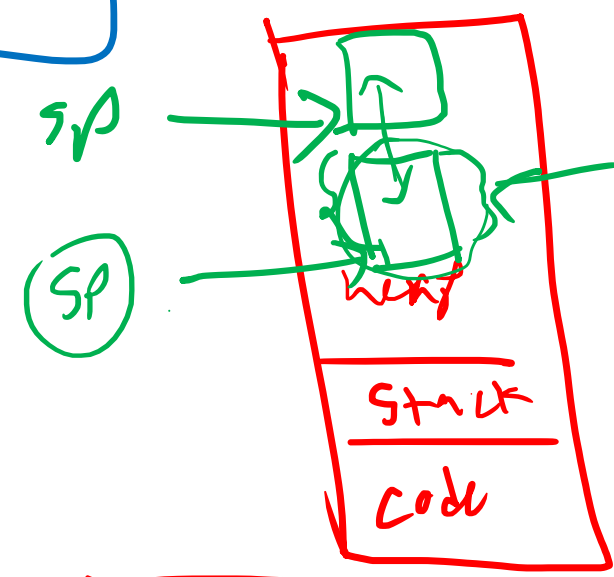
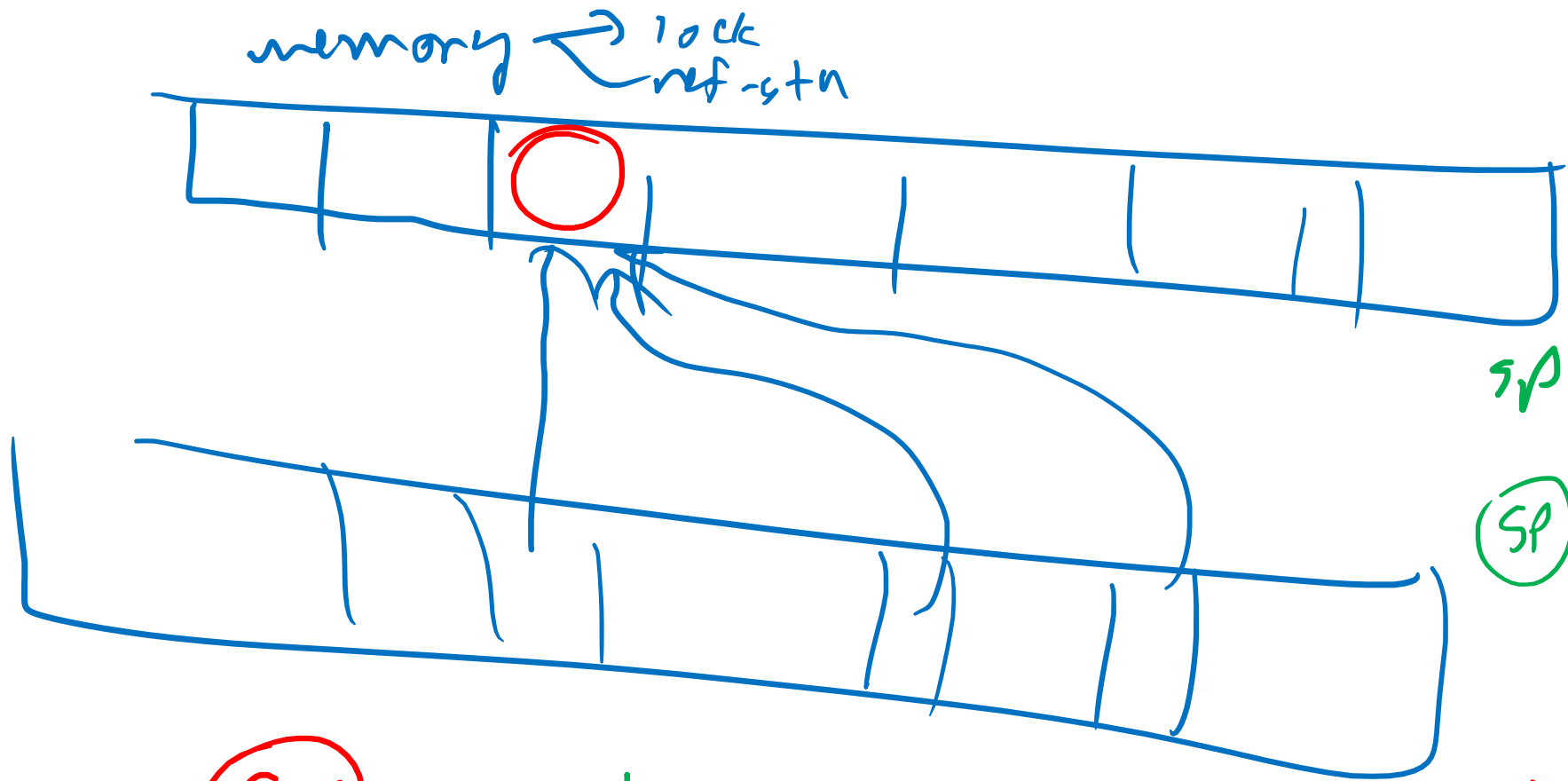
u alloc

t₁
t₂

group proc)

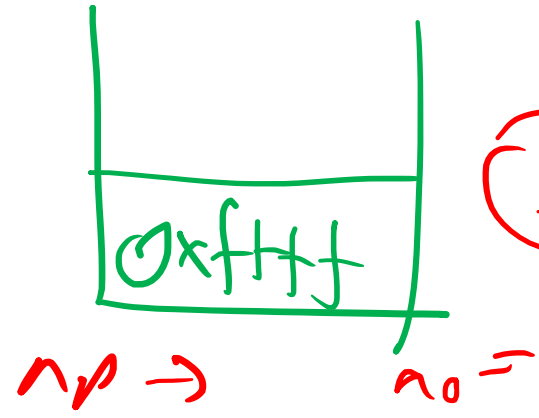






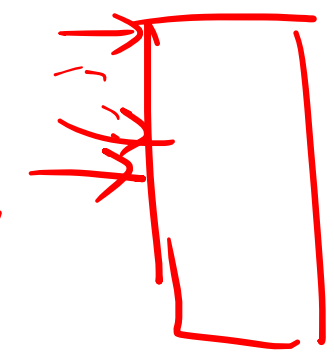
$SP = \text{pin } 6$

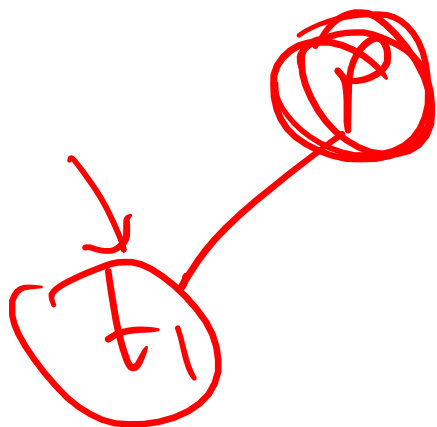
$SP = SP - 16$



$SP = SP - 16$

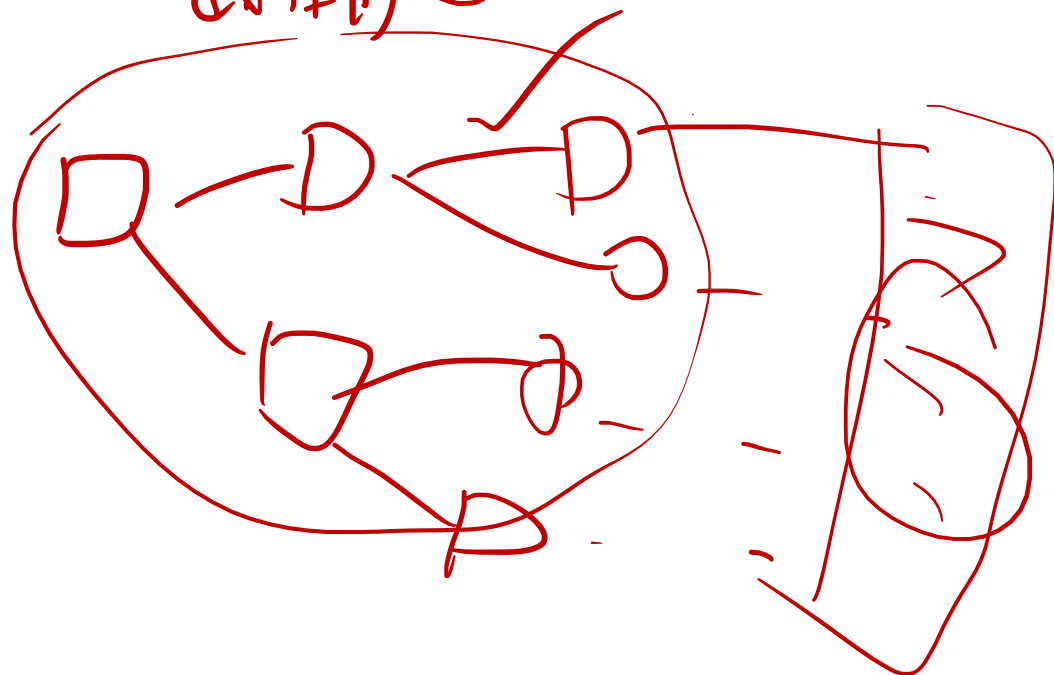
0000





join()
exit()

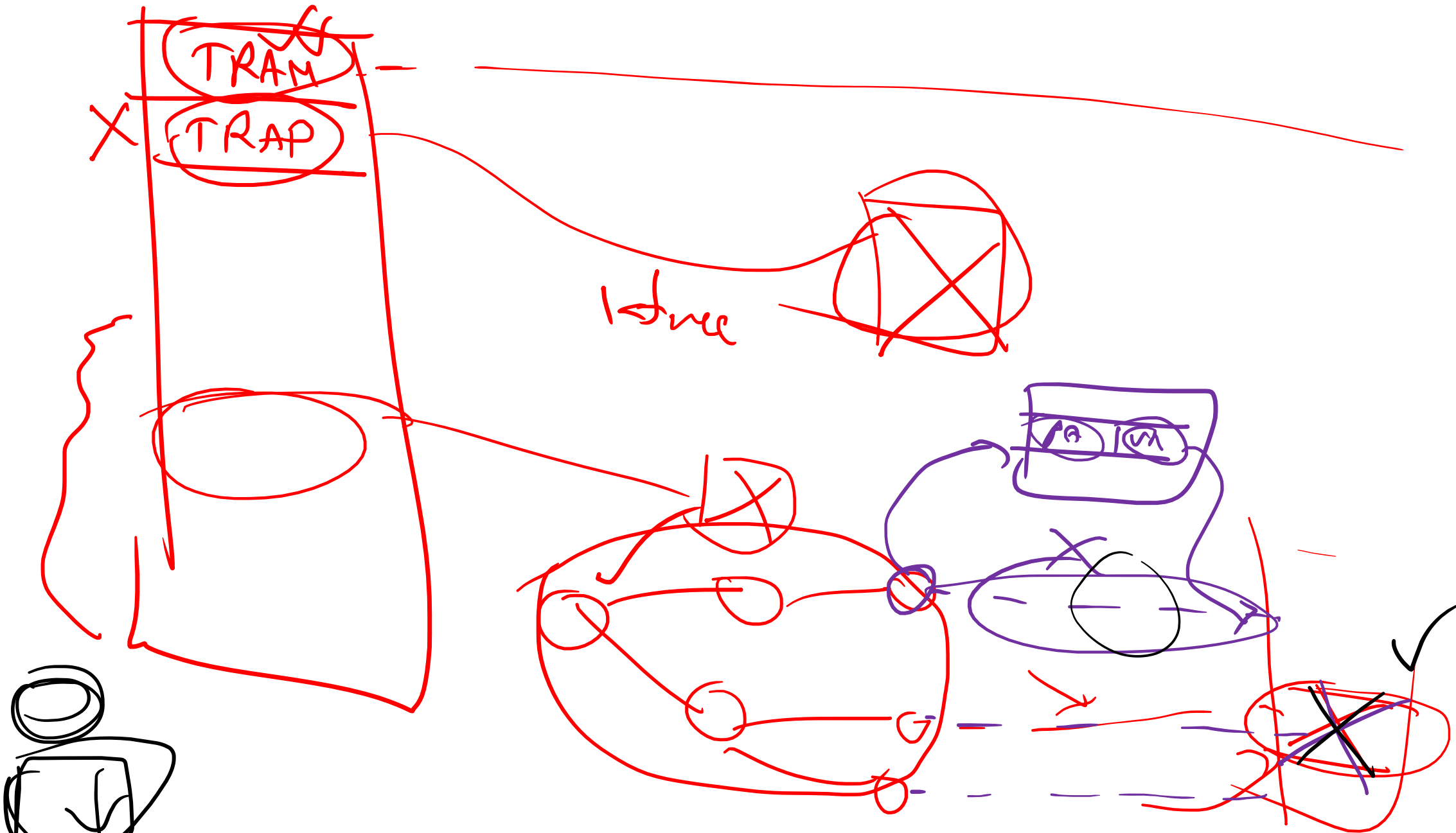
exit() → ZOMBIE



```

393 int
394 wait(uint64_t *addr)
395 {
396     struct proc *pp;
397     int havekids, pid;
398     struct proc *p = myproc();
399
400     acquire(&wait_lock);
401
402     for(;;){
403         // Scan through table looking for exited children.
404         havekids = 0;
405         for(pp = proc; pp < &proc[NPROC]; pp++){
406             if(pp->parent == p){
407                 // make sure the child isn't still in exit() or swtch().
408                 acquire(&pp->lock);
409
410                 havekids = 1;
411                 if(pp->state == ZOMBIE){
412                     // Found one.
413                     pid = pp->pid;
414                     if(addr != 0 && copyout(p->pagetable, addr, (char *)pp->xstate,
415                                             sizeof(pp->xstate)) < 0) {
416                         release(&pp->lock);
417                         release(&wait_lock);
418                         return -1;
419                     }
420                     freeproc(pp);
421                     release(&pp->lock);
422                     release(&wait_lock);
423                     return pid;
424                 }
425                 release(&pp->lock);
426             }
427         }
428
429         // No point waiting if we don't have any children.
430         if(!havekids || killed(p)){
431             release(&wait_lock);
432             return -1;
433         }
434
435         // wait for a child to exit.
436         sleep(p, &wait_lock); //DOC: wait-sleep
437     }
438 }
439

```

```

539 void
540 sleep(void *chan, struct spinlock *lk)
541 {
542     struct proc *p = myproc();
543     // Must acquire p->lock in order to
544     // change p->state and then call sched.
545     // Once we hold p->lock, we can be
546     // guaranteed that we won't miss any wakeup
547     // (wakeup locks p->lock),
548     // so it's okay to release lk.
549 }

```

```

550 acquire(&p->lock); //DOC: sleeplock1
551 release(lk);

```

```

552 // Go to sleep.
553 p->chan = chan;
554 p->state = SLEEPING;

```

```

555 sched();

```

```

556 // Tidy up.
557 p->chan = 0;

```

```

558 // Reacquire original lock.

```

```

559 release(&p->lock);

```

```

560 acquire(lk);

```

```

561 void
562 wakeup(void *chan)

```

```

563 {
564     struct proc *p;

```

```

565 for(p = proc; p < &proc[NPROC]; p++) {

```

```

566     if(p != myproc())

```

```

567         acquire(&p->lock);

```

```

568         if(p->state == SLEEPING && p->chan == chan) {

```

```

569             p->state = RUNNABLE;

```

```

570         }

```

```

571         release(&p->lock);

```

```

572     }

```

```

573 }

```

```

574 }

```

```

575 }

```

```

576 }

```

```

577 }

```

```

578 }

```

```

579 }

```

```

580 }

```

```

581 }

```

```

582 }

```

```

583 }

```

```

584 }

```

```

350 void
351 exit(int status)
352 {
353     struct proc *p = myproc();
354     if(p == initproc)
355         panic("init exiting");
356     // Close all open files.
357     for(int fd = 0; fd < NOFILE; fd++){
358         if(p->ofile[fd]){
359             struct file *f = p->ofile[fd];
360             fileclose(f);
361             p->ofile[fd] = 0;
362         }
363     }
364     begin_op();
365     input(p->cwd);
366     end_op();
367     p->cwd = 0;
368     acquire(&wait_lock);
369     reparent(p);
370     // Parent might be sleeping in wait().
371     wakeup(p->parent);
372     acquire(&p->lock);
373     p->xstate = status;
374     p->state = ZOMBIE;
375     release(&wait_lock);
376     // Jump into the scheduler, never to return.
377     sched();
378     panic("zombie exit");
379 }

```

```

380 int
381 wait(uint64 addr)
382 {
383     struct proc *pp;
384     int havekids, pid;
385     struct proc *p = myproc();
386     acquire(&wait_lock);
387     for(;;){
388         // Scan through table looking for exited children.
389         havekids = 0;
390         for(pp = proc; pp < &proc[NPROC]; pp++){
391             if(pp->parent == p){
392                 // make sure the child isn't still in exit() or swtch().
393                 acquire(&pp->lock);
394                 havekids = 1;
395                 if(pp->state == ZOMBIE){
396                     // round one
397                     pid = pp->pid;
398                     if(addr != 0 && copyout(p->pagetable, addr, (char *)&pp->xstate,
399                                     sizeof(pp->xstate)) < 0) {
400                         release(&pp->lock);
401                         release(&wait_lock);
402                         return -1;
403                     }
404                     freeproc(pp);
405                     release(&pp->lock);
406                     release(&wait_lock);
407                     return pid;
408                 }
409                 release(&pp->lock);
410             }
411         }
412         // No point waiting if we don't have any children.
413         if(!havekids || killed(p)){
414             release(&wait_lock);
415             return -1;
416         }
417         // Wait for a child to exit.
418         sleep(p, &wait_lock); //DOC: wait-sleep
419     }

```

```

393 int
394 wait(uint64 addr)
395 {
396     struct proc *pp;
397     int havekids, pid;
398     struct proc *p = myproc();
399     acquire(&wait_lock);
400     for(;;){
401         // Scan through table looking for exited children.
402         havekids = 0;
403         for(pp = proc; pp < &proc[NPROC]; pp++){
404             if(pp->parent == p){
405                 // make sure the child isn't still in exit() or swtch().
406                 acquire(&pp->lock);
407                 havekids = 1;
408                 if(pp->state == ZOMBIE){
409                     // round one
410                     pid = pp->pid;
411                     if(addr != 0 && copyout(p->pagetable, addr, (char *)&pp->xstate,
412                                     sizeof(pp->xstate)) < 0) {
413                         release(&pp->lock);
414                         release(&wait_lock);
415                         return -1;
416                     }
417                     freeproc(pp);
418                     release(&pp->lock);
419                     release(&wait_lock);
420                     return pid;
421                 }
422                 release(&pp->lock);
423             }
424         }
425         // No point waiting if we don't have any children.
426         if(!havekids || killed(p)){
427             release(&wait_lock);
428             return -1;
429         }
430         // Wait for a child to exit.
431         sleep(p, &wait_lock); //DOC: wait-sleep
432     }

```

$p_0 \rightarrow \text{chan} = (p_0)$

