xv6©-rev10

(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)

exec syscall II

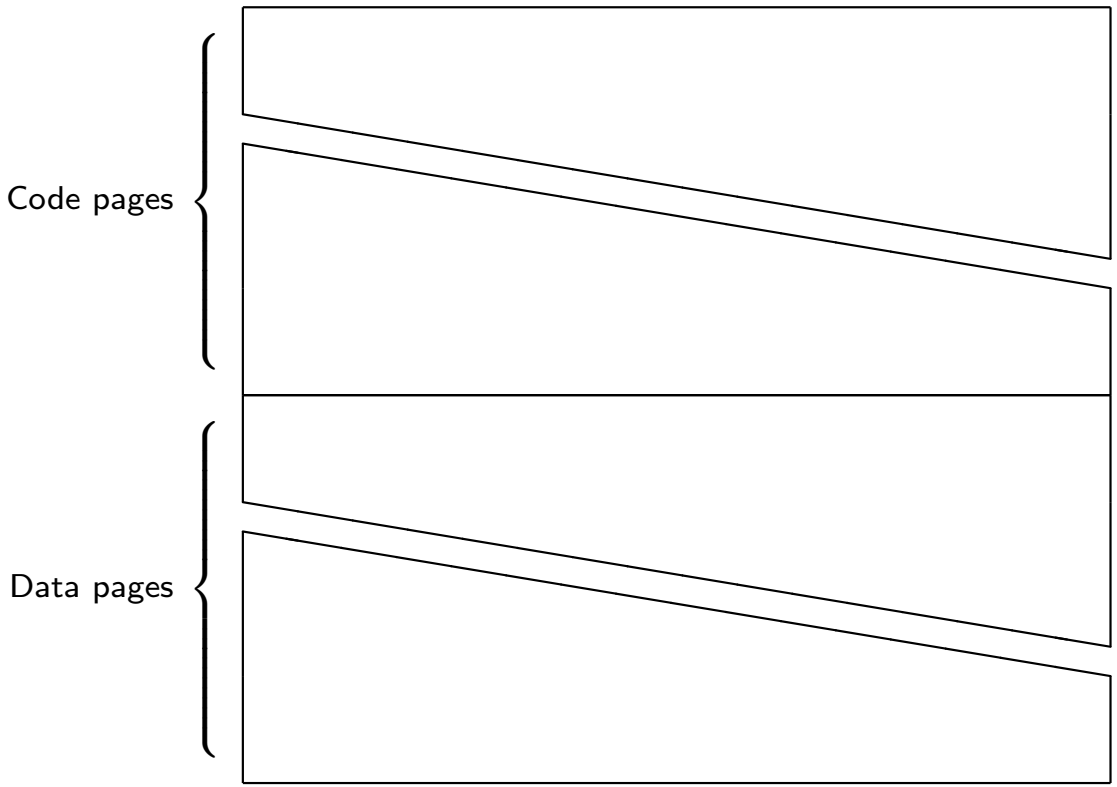
Carmi Merimovich

Tel-Aviv Academic College

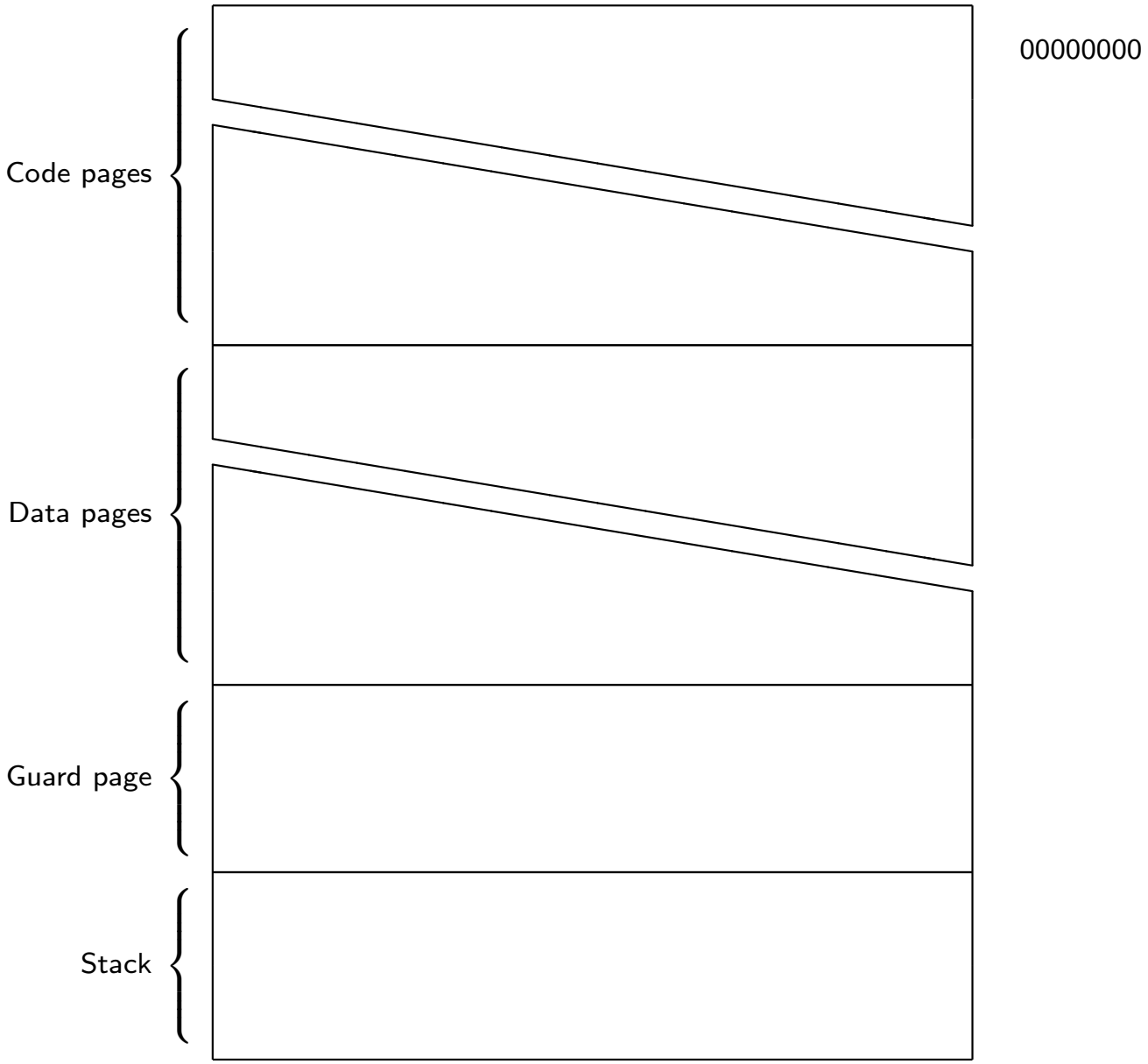
January 10, 2017

exec: step 2.

Address space after loading elf



Adding guard page and stack page



exec: Allocate user stack and guard page

```
6664    sz = PGROUNDUP(sz);
        if ((sz = allocuvm(pgdir, sz, sz + 2*PGSIZE)) == 0)
         goto bad;
        clearpteu(pgdir, (char*)(sz - 2*PGSIZE));
        sp = sz;
```

- Note the sp variable for later use.

exec: step 3

- The invoked executable expects to start with the following signature:

```
main(int cnt, char **argv);
```

- Hence the following stack should be prepared:

	argv
	cnt
esp →	??

- Hence somewhere a vector of pointers should be allocated.
- Hence a copy of the strings supplied to the `exec` syscall should be done.
- All these allocation are done on the new stack.
- Note, **I believe** all this should be done in user mode!

Copying between address spaces: Common AS

- First tool: Convert full virtual address to kernel address.
- Note 0 is not a legal kernel address.

```
char *my_uva2ka(pde_t *pgdir, char *uva) {
  pte_t *pte = walkpgdir(pgdir, uva, 0);
  if (pte == 0 || (*pte & PTE_P) == 0)
    return 0;
  if ((*pte & PTE_U) == 0)
    return 0;
  return (char*)P2V(PTE_ADDR(*pte) |
                    (uva & (PGSIZE-1)));
}
```

Copying between address spaces: Generalized addresses

```
int gcopy(pde_t *tpgdir, int taddr,
        pde_t *fpgdir, int faddr, int len) {
  for (int i = 0; i < len; i++) {
    char *fka = my_uva2ka(fpgdir, faddr++);
    char *tka = my_uva2ka(tpgdir, taddr++);
    if (fka == 0 || tka == 0) return 0;
    *tka = *fka;
  }
}
```

- The above code is horribly inefficient.
- However, it is quite clear.

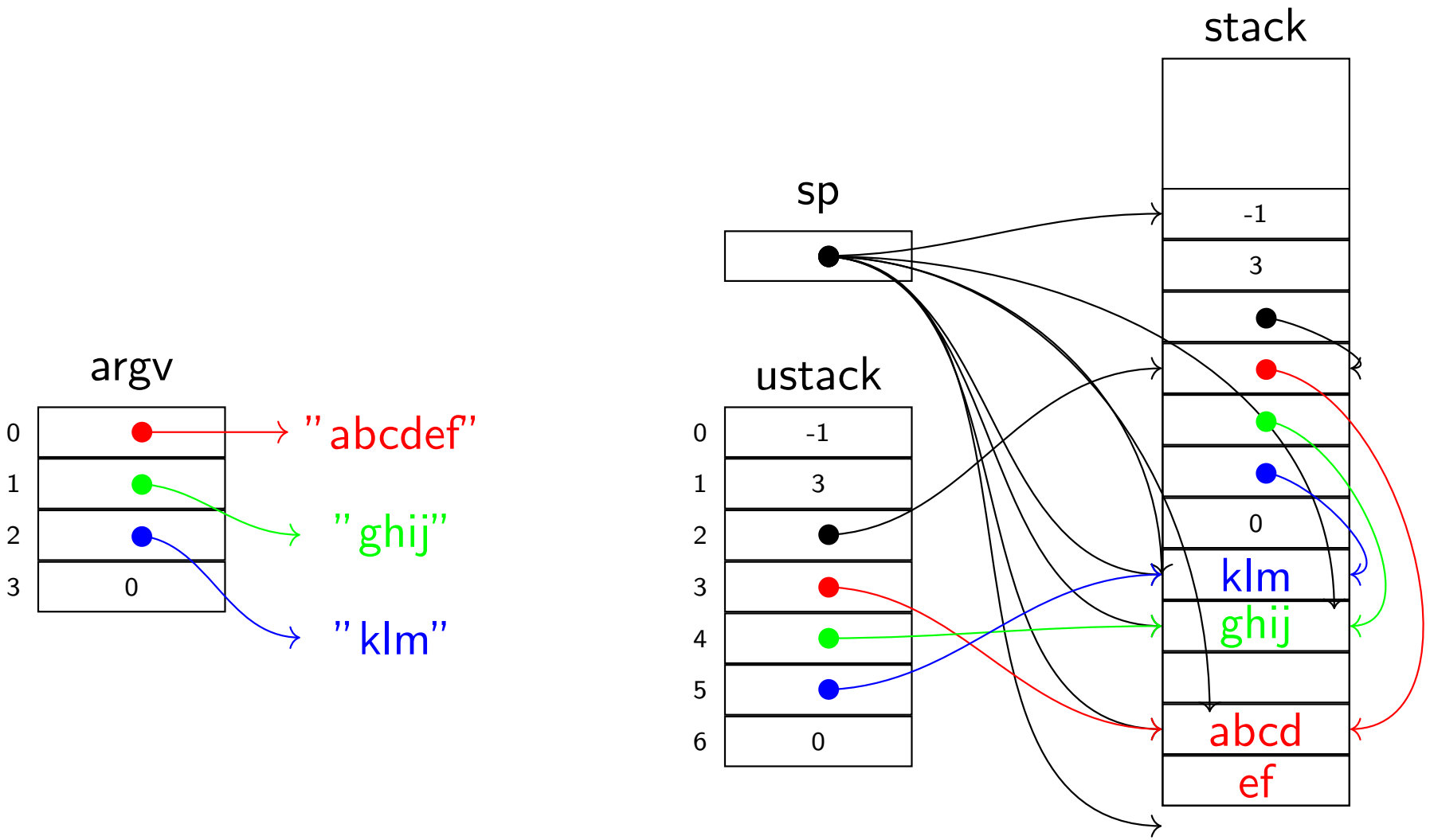
xv6 copying between address spaces: Common AS

- Convert full page address to kernel address.
- Note 0 is not a legal kernel address.

```
2102 char *uva2ka(pde_t *pgdir, char *uva) {
       pte_t *pte = walkpgdir(pgdir, uva, 0);
       if ((*pte & PTE_P) == 0)
        return 0;
       if ((*pte & PTE_U) == 0)
        return 0;
       return (char*)P2V(PTE_ADDR(*pte));
     }
```

xv6 copy to a generalized address

```
int copyout(pde_t *pgdir, uint va,
                          void *p, uint len) {
  char *buf = (char*)p;
  while (len > 0) {
    uint va0 = (uint)PGROUNDDOWN(va);
    char *pa0 = uva2ka(pgdir, (char*)va0);
    if (pa0 == 0) return -1;
    uint n = PGSIZE - (va - va0);
    if (n > len) n = len;
    memmove(pa0 + (va - va0), buf, n);
    len -= n;
    buf += n;
    va = va0 + PGSIZE;
  }
  return 0;
}
```


argv
0
1
2
3
0
"abcdef"
"ghij"
"klm"
sp
ustack
0
1
2
3
4
5
6
-1
3
0
stack
-1
3
0
klm
ghij
abcd
ef

exec: Copying argv[]

```
6671    uint ustack[3+MAXARG+1];
        for (argc = 0; argv[argc]; argc++) {
         if (argc >= MAXARG) goto bad;
         sp = (sp - (strlen(argv[argc]) + 1)) & ~3;
         if (copyout(pgdir, sp, argv[argc],
                 strlen(argv[argc]) + 1) < 0)  goto bad;
         ustack[3+argc] = sp;
        }
        ustack[3+argc] = 0;

        ustack[0] = 0xffffffff; // fake return PC
        ustack[1] = argc;
        ustack[2] = sp - (argc+1)*4; // argv pointer

        sp -= (3+argc+1) * 4;
        if (copyout(pgdir, sp, ustack, (3+argc+1)*4) < 0)
         goto bad;
```

exec: step 4

exec: Switching address space, fixing the **trapframe**

```
6690    for (last=s=path; *s; s++) if (*s == '/')
            last = s+1;
        safestrcpy(myproc()->name, last, sizeof(myproc()->n

        oldpgdir = myproc()->pgdir;
        myproc()->pgdir = pgdir;
        myproc()->sz = sz;
        myproc()->tf->eip = elf.entry; // main
        myproc()->tf->esp = sp;
        switchuvm(myproc());
        freevm(oldpgdir);
        return 0;
      bad:
        if (pgdir) freevm(pgdir);
        if (ip) {iunlockput(ip);end_op()};
        return -1;
      }
```