# lecture 15

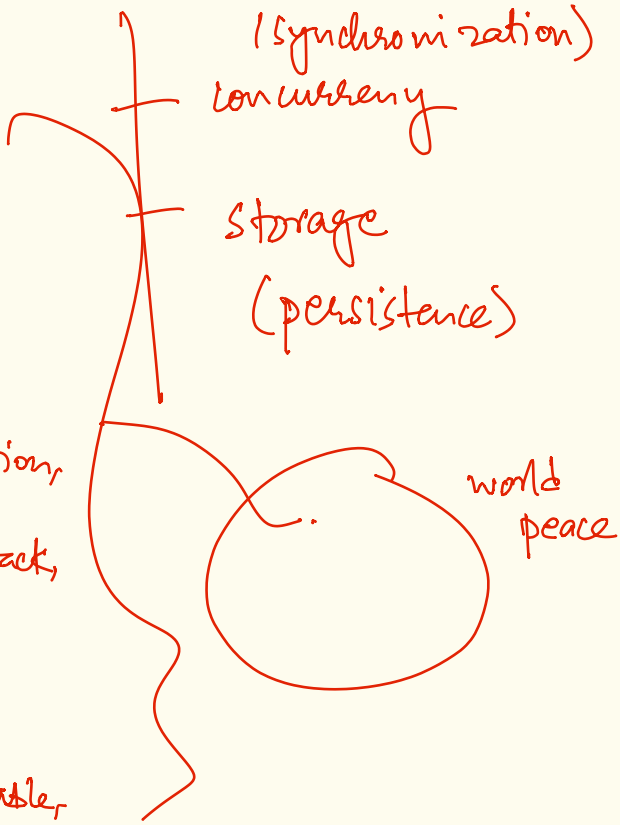
process redux

process creation & scheduling (xv6 details)

recap

Keywords:

abstractions, OS, LDE, user mode, Kernel mode, ISA, system calls, interrupts, interrupt handler, timer, PCB, signals, scheduler, context, address space, address trs, segmentation, paging, mmu, CR3, CR2, Kernel stack, user stack, heap, TLB, malloc, page replacement (policy), demand paging/swapping, page tables, pte

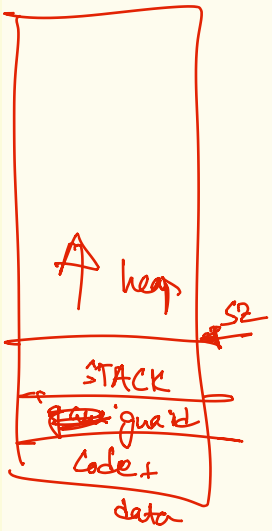



- relook at the PCB
- process creation
  - first
  - child processes
- scheduler (mechanism)
- scheduling policy

```
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct proc {
39   uint sz;                     // Size of process memory (bytes)
40   pde_t* pgdir;                // Page table
41   char *kstack;                // Bottom of kernel stack for this process
42   enum procstate state;        // Process state
43   int pid;                     // Process ID
44   struct proc *parent;         // Parent process
45   struct trapframe *tf;        // Trap frame for current syscall
46   struct context *context;     // swtch() here to run process
47   void *chan;                  // If non-zero, sleeping on chan
48   int killed;                  // If non-zero, have been killed
49   struct file *ofile[NOFILE];  // Open files
50   struct inode *cwd;           // Current directory
51   char name[16];               // Process name (debugging)
52 };
53
54 // Process memory is laid out contiguously, low addresses first:
55 //   text
56 //   original data and bss
57 //   fixed-size stack
58 //   expandable heap
```

kVA for page dir

per process kstack




- first process — userinit — setup of process — allocproc, setupkvm, inituvm
  - mpmain — invoke scheduler
    - schedule the process.

- child process creation.
  - fork
    - allocproc — find a free pcb, add details
    - copyuvm — setup VA space & mapping for new process (identical to parent)
    - alloc & build kstack. — np->kstack = kalloc();
    - prepare for return.

np->stack = kalloc();

sp = np->stack + KSTACKSIZE;

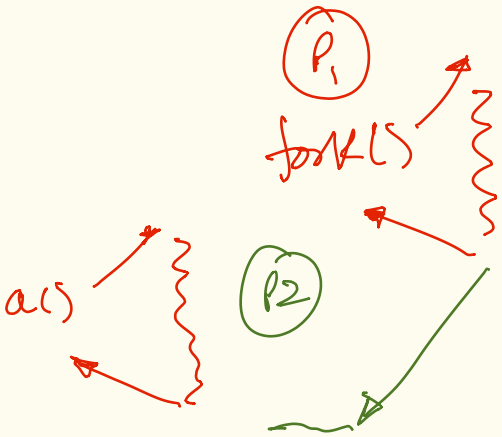
np->tf = sp - sizeof(*tf);

→ copy (p->tf, np->tf);

sp = np->tf;

sp = sp - 4;

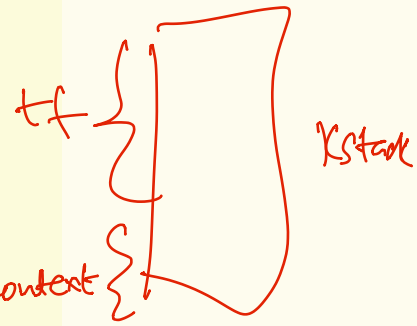
*sp = trapret; — ?

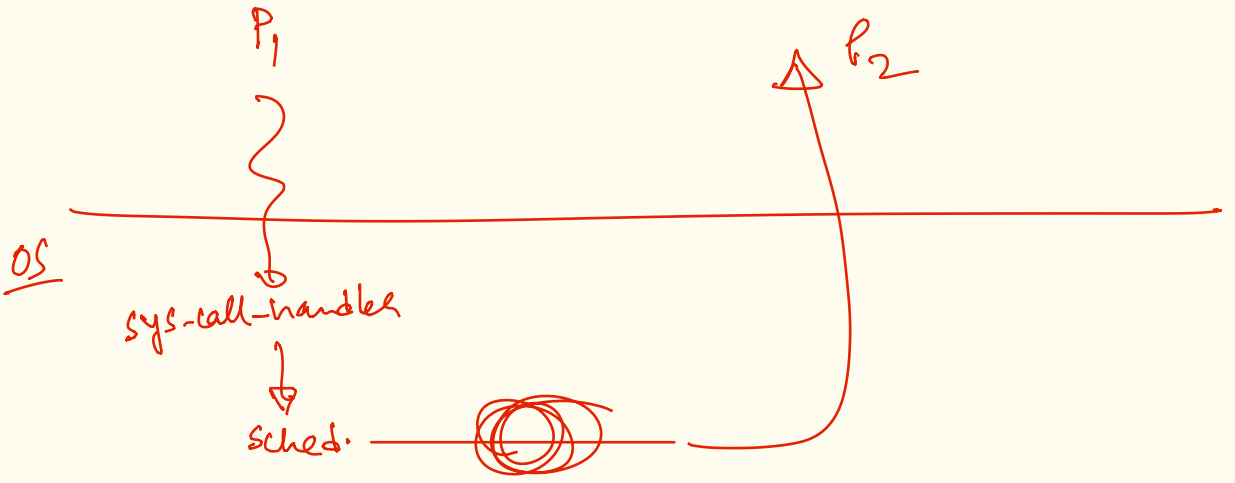



np->tf->eax = 0;

```
// Per-CPU state
struct cpu {
  uchar apicid;                // Local APIC ID
  struct context *scheduler;   // swtch() here to enter scheduler
  struct taskstate ts;         // Used by x86 to find stack for interrupt
  struct segdesc gdt[NSEGS];   // x86 global descriptor table
  volatile uint started;       // Has the CPU started?
  int ncli;                    // Depth of pushcli nesting.
  int intena;                  // Were interrupts enabled before pushcli?
  struct proc *proc;           // The process running on this cpu or null
};

extern struct cpu cpus[NCPU];
extern int ncpu;

//PAGEBREAK: 17
// Saved registers for kernel context switches.
// Don't need to save all the segment registers (%cs, etc),
// because they are constant across kernel contexts.
// Don't need to save %eax, %ecx, %edx, because the
// x86 convention is that the caller has saved them.
// Contexts are stored at the bottom of the stack they
// describe; the stack pointer is the address of the context.
// The layout of the context matches the layout of the stack in swtch.S
// at the "Switch stacks" comment. Switch doesn't save eip explicitly,
// but it is on the stack and allocproc() manipulates it.
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

??




mpmain — last call of the kernel setup.

↑ scheduler()

while (1) {

pick next process, P2;

switch to (P2);

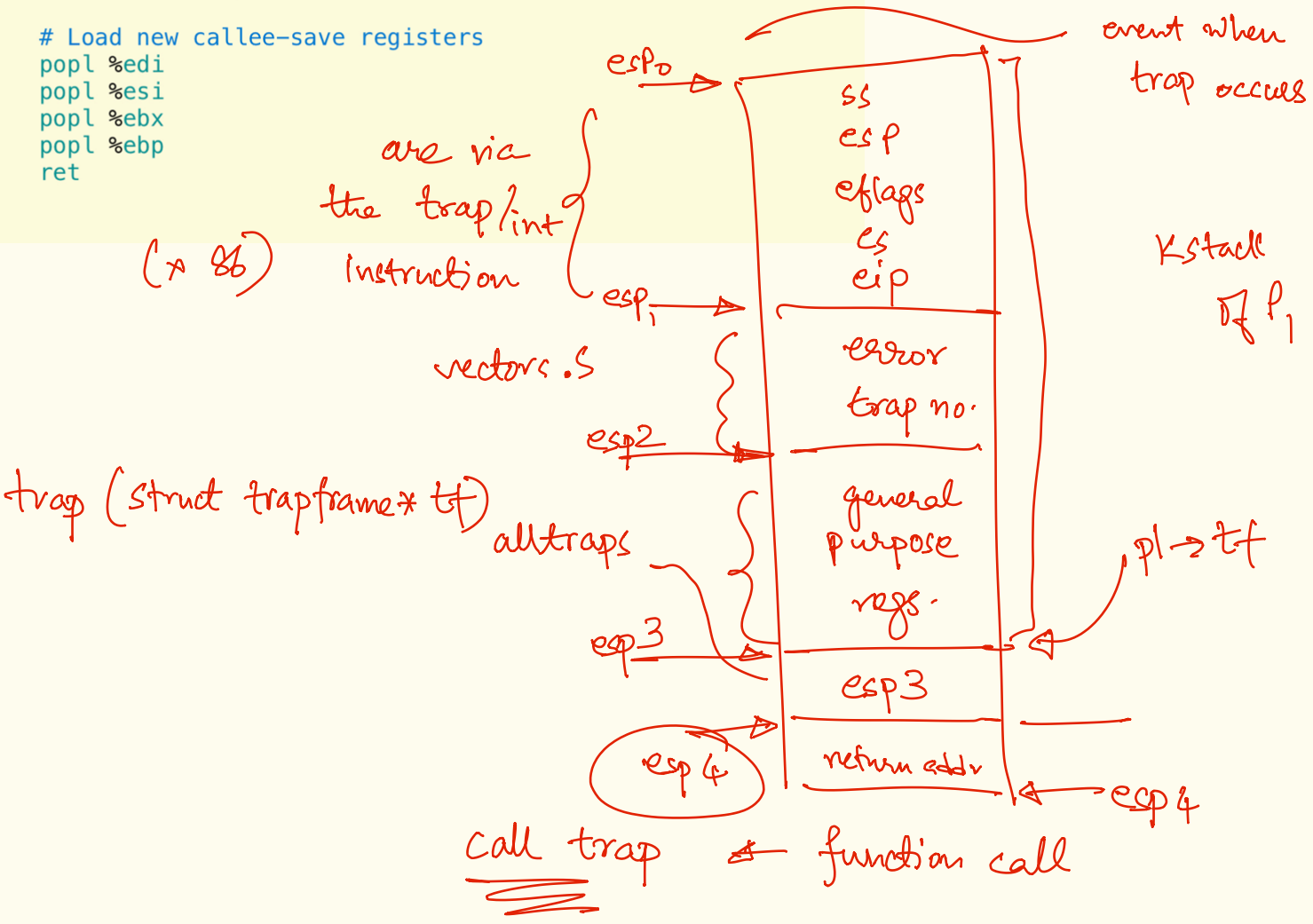
}

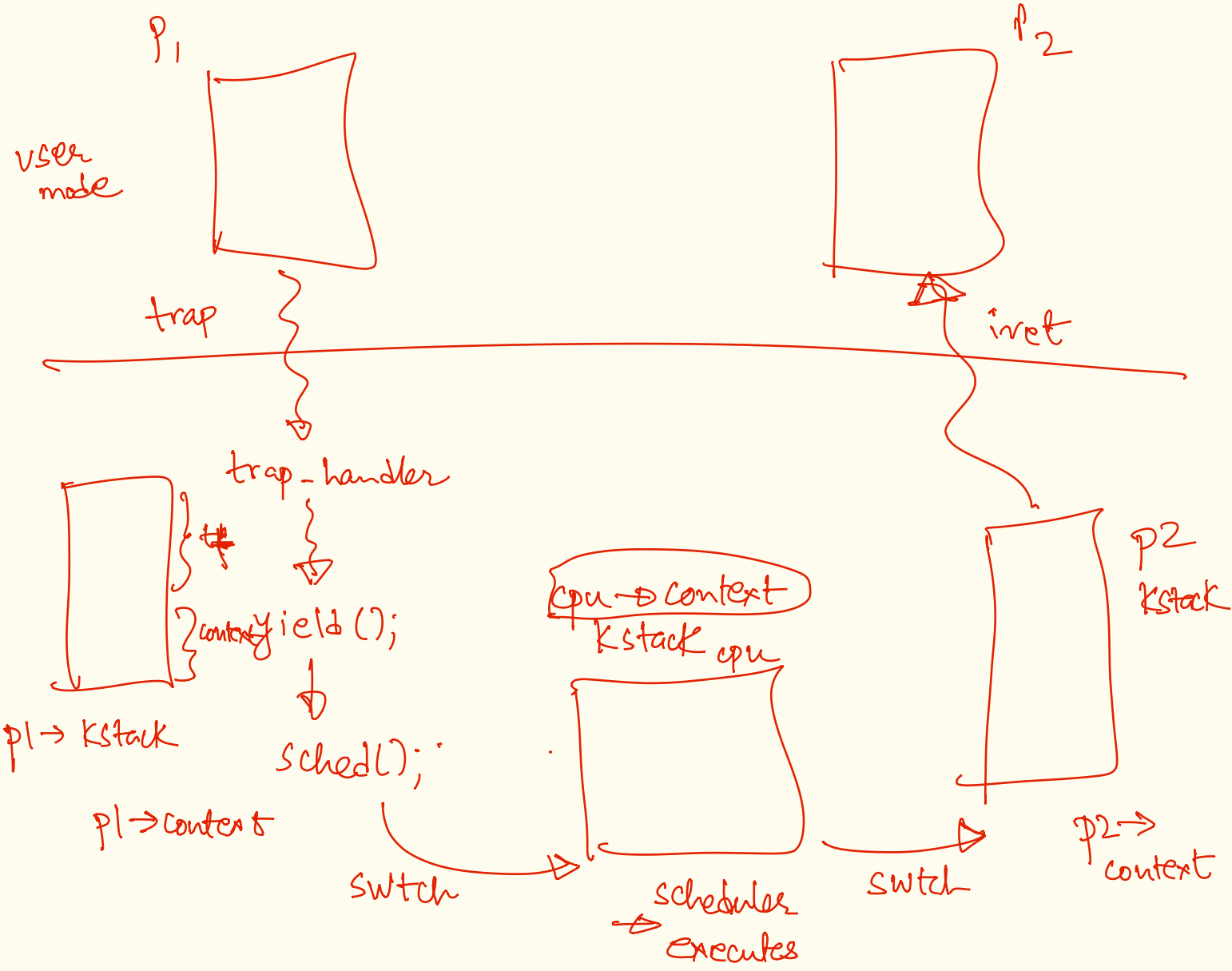
scheduling policy

switching mechanism.

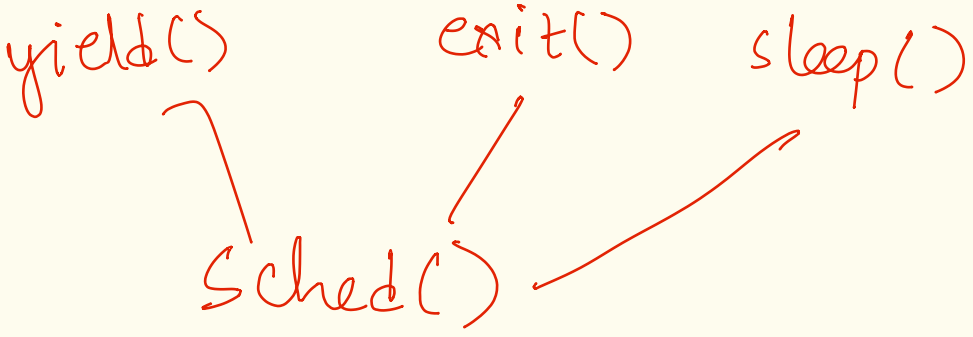
On a cpu for the kernel on per cpu kernel stack

```
# Context switch
#
#   void swtch(struct context **old, struct context *new);
#
# Save the current registers on the stack, creating
# a struct context, and save its address in *old.
# Switch stacks to new and pop previously-saved registers.

.globl swtch
swtch:
  movl 4(%esp), %eax
  movl 8(%esp), %edx

  # Save old callee-save registers
  pushl %ebp
  pushl %ebx
  pushl %esi
  pushl %edi

  # Switch stacks
  movl %esp, (%eax)
  movl %edx, %esp

  # Load new callee-save registers
  popl %edi
  popl %esi
  popl %ebx
  popl %ebp
  ret
```






swtch ( **context , context*) ;




HW lookup implementation of swtch.

file → swtch.S