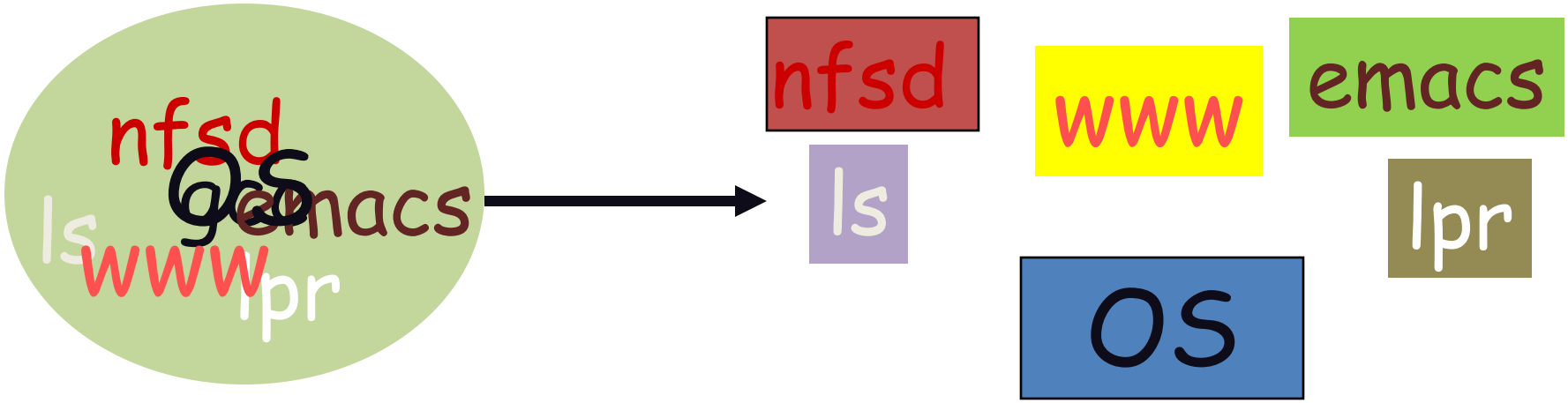
Why processes? Simplicity



Processes give isolated *Address Spaces*

nfsd

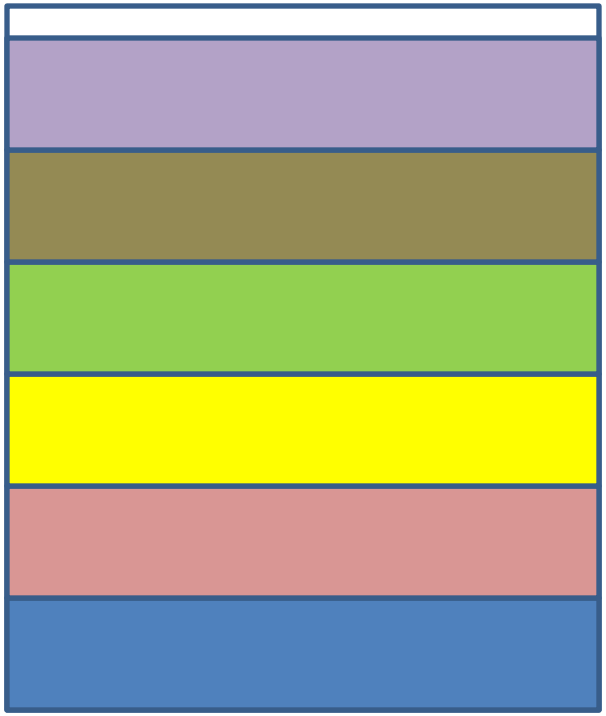
www

emacs

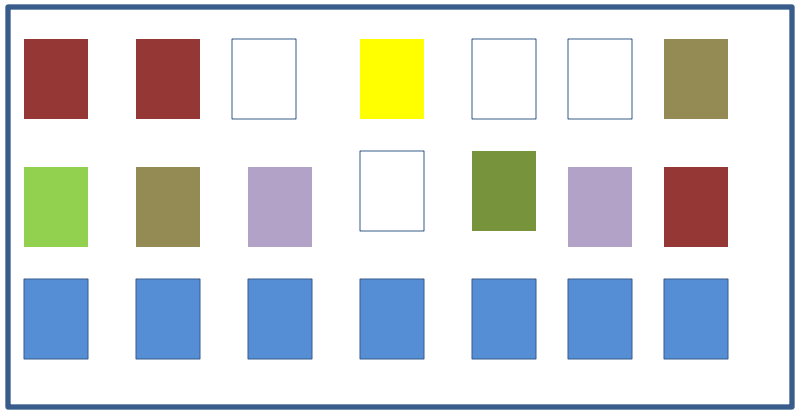
lpr

OS

ls



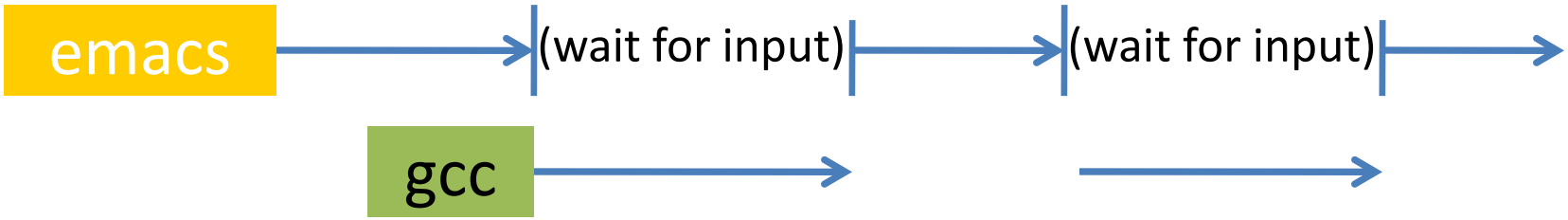
Physical Memory (Segmentation)



Physical Memory (Paging)

Why processes? Speed

- I/O parallelism:

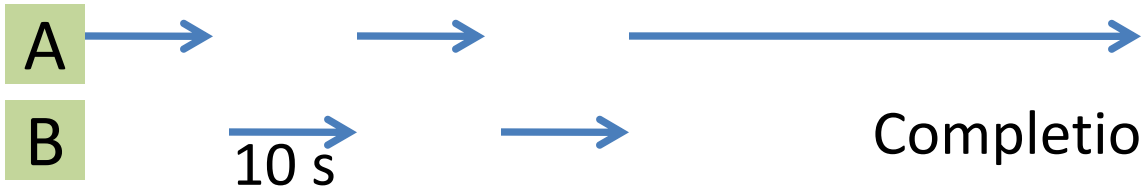


overlap execution: make 1 CPU into many

(Real parallelism: > 1 CPU (multiprocessing))

- Completion time:

B's completion time = 100s (A + B). So overlap



Completion time for B? A?

Process != Program

- Program: code + data

  passive

```
int a;
int main() {
    printf("hello");
}
```

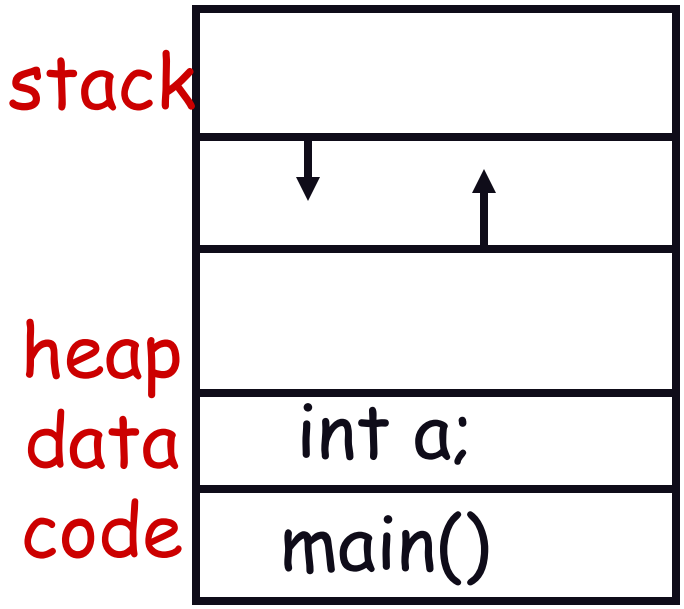
- Process: running program

  state: registers, stack, heap…

  position: program counter

- We both run netscape:

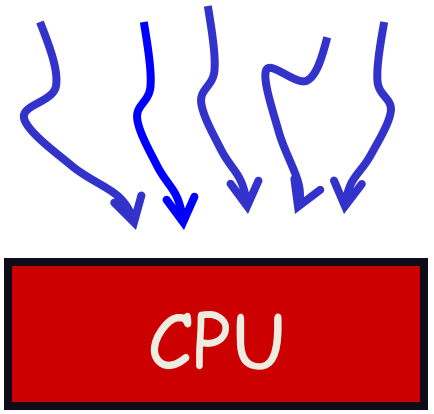
  Same program, different process



The multithreading illusion

- Each thread has its illusion of own CPU
  - yet on a uni-processor, all threads share the same physical CPU!

How does this work?



- Two key pieces:
  - thread control block: one per thread, holds execution state
  - dispatching loop:

```
while(1)
    interrupt thread
    save state
    get next thread
    load state, jump to it
```

Remote Procedure Call (RPC) Comparison

P1:
calls send(args) [2]
calls recv(), blocks [1]

P2:
calls recv(args) [2]
… does work …
calls send(results) [2]

P1:
recv() returns [1]

Monolithic Kernel: around [8] total user/kernel crossings

P1:
sets up args in regs
calls yield(P2) [1]

P2:
resumes [1]
… does work …
sets up results in regs
calls yield(P1) [1]

P1:
resumes [1]

ExoKernel: around [4] total user/kernel crossings