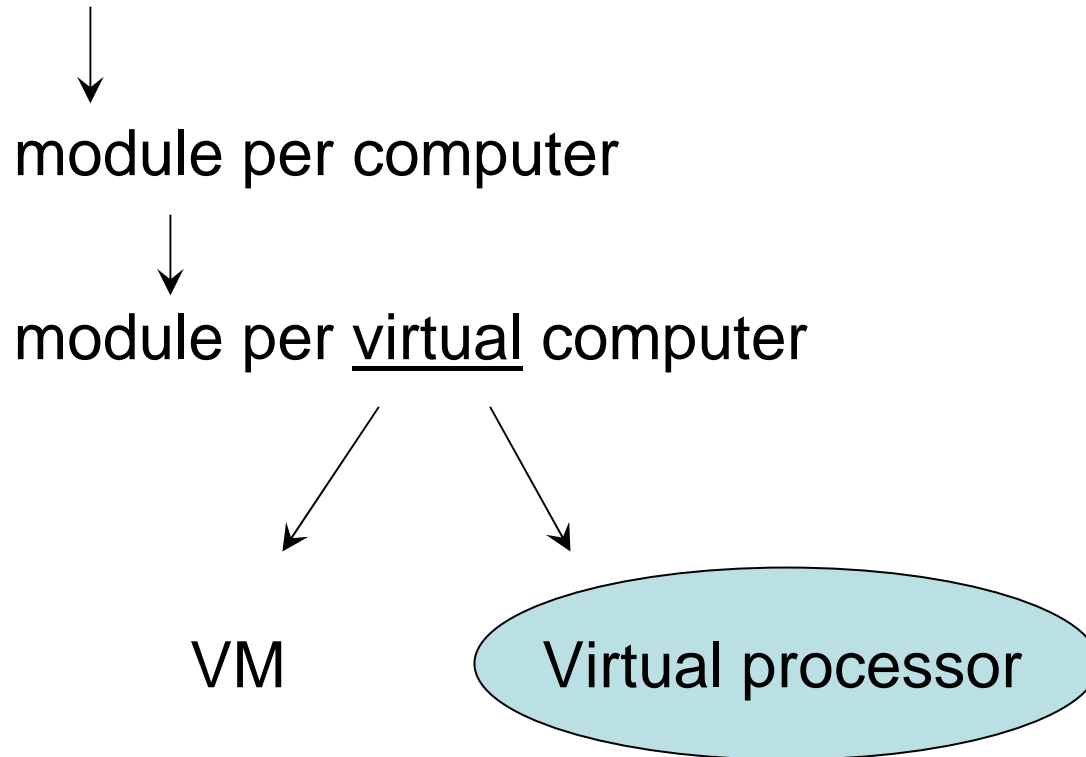


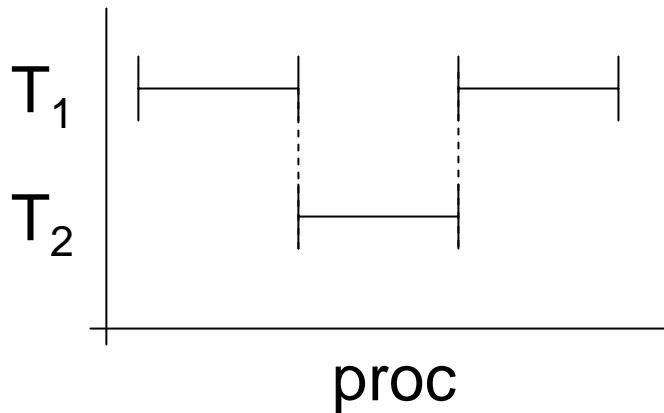
# Enforcing Modularity



# Virtual Processor

Each program – “Thread” of execution

instructions,  
registers, PC, SP  
stack



Cooperative

vs

Preemptive

Same AS

vs

Diff

# Yield()

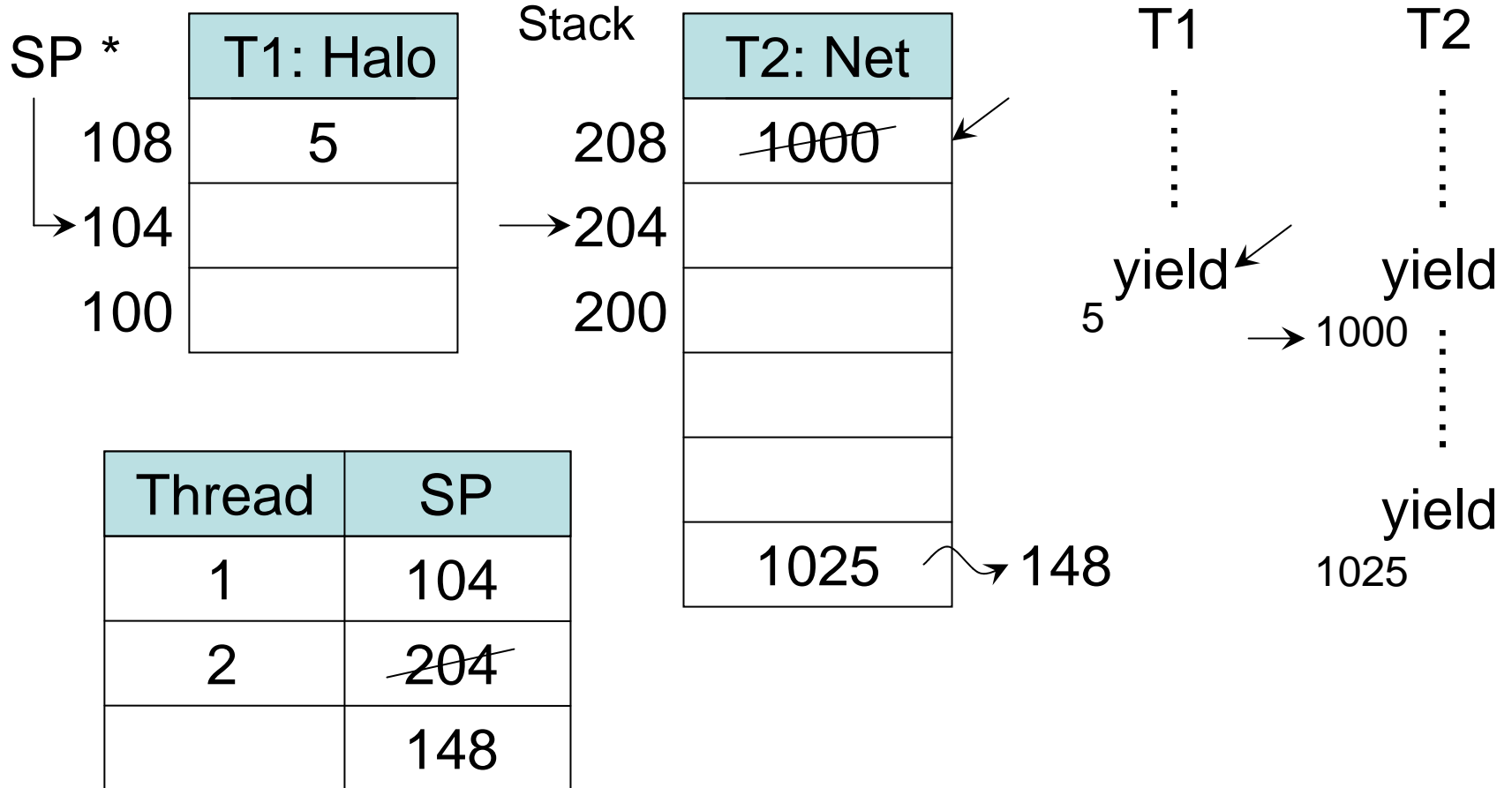
```
yield {  
    Save state  
    Schedule next thread  
    dispatch next thread  
}
```

```
int table[NUM_THREADS]
```

```
int next
```

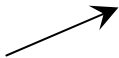
```
int me ← local to thread
```

# Stack Example



# Preemptive scheduling

(No explicit yield)

Timer interrupt  Processor line  
checked by  $\mu$ Proc before each instr  
If high, calls “gate”

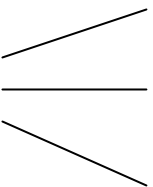
Kernel calls yield() on current thread

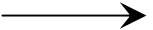
Save state

schedule + run next thread

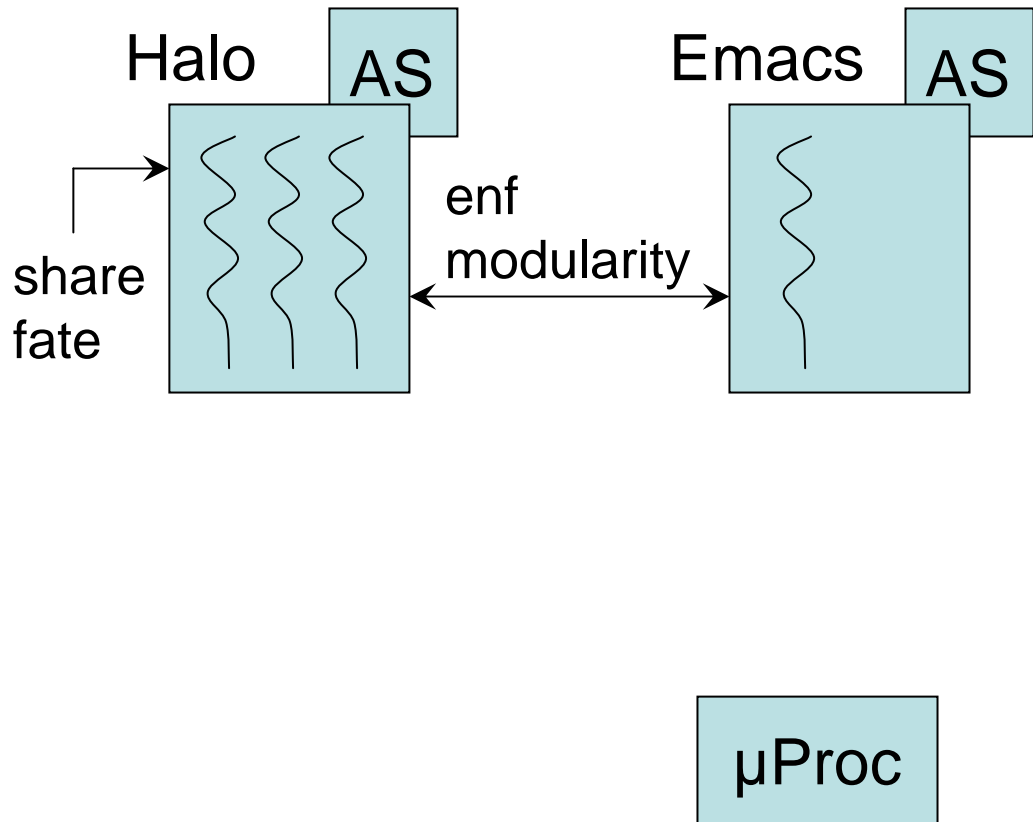
# Processes – AS + thread(s)

## Kernel support

create  alloc AS, phys. mem, page map  
load code into mem, map into AS  
create thread, add to thread table

destroy  remove AS  
remove thread from table

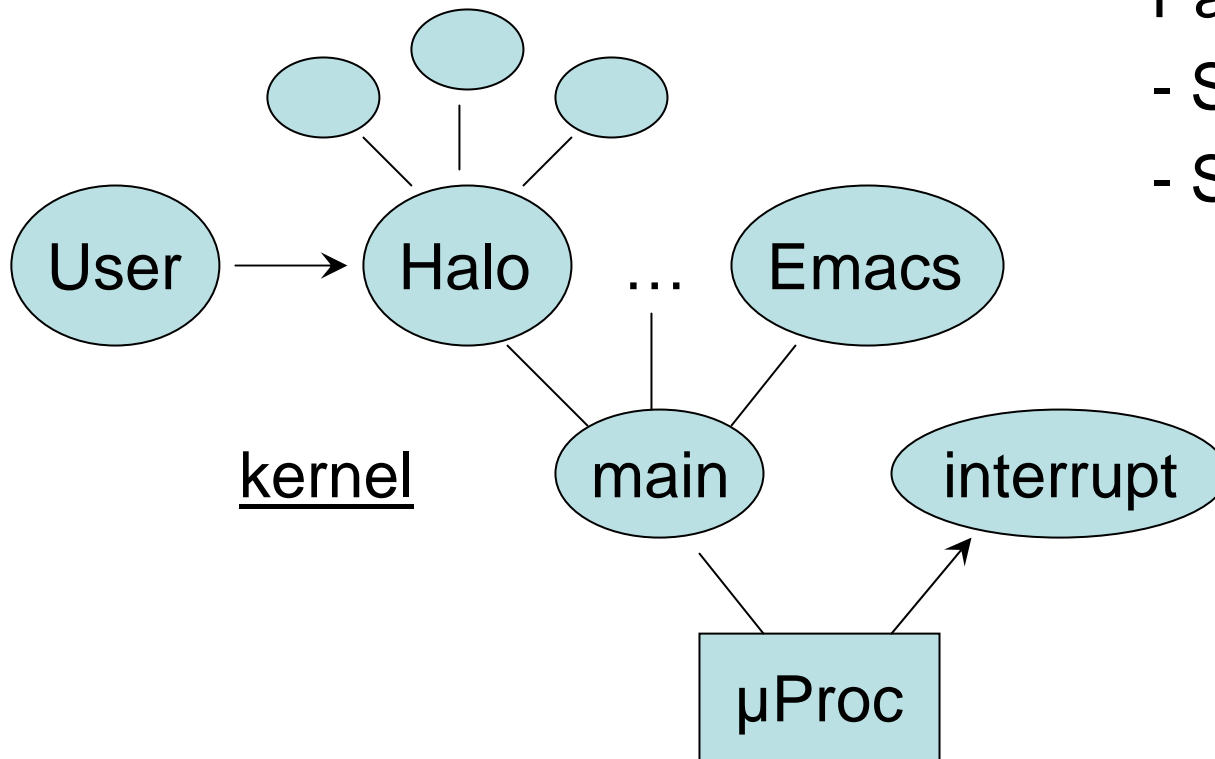
# Layering of Threads



Parent threads

- Scheduling policy
- Switching mech.

# Layering of Threads



Parent threads

- Scheduling policy
- Switching mech.



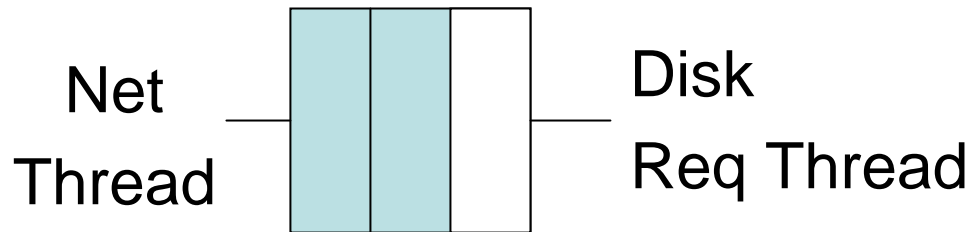
# Coordinating Access

Sequence Coord

`wait(v, cond)`

`signal(v)`

Web Server



```
while(true)
    while(empty());
    m = dequeue()
    process(m)
```

```
while(true)
    m = next_blk()
    while(full()){};
    enqueue(m)
```