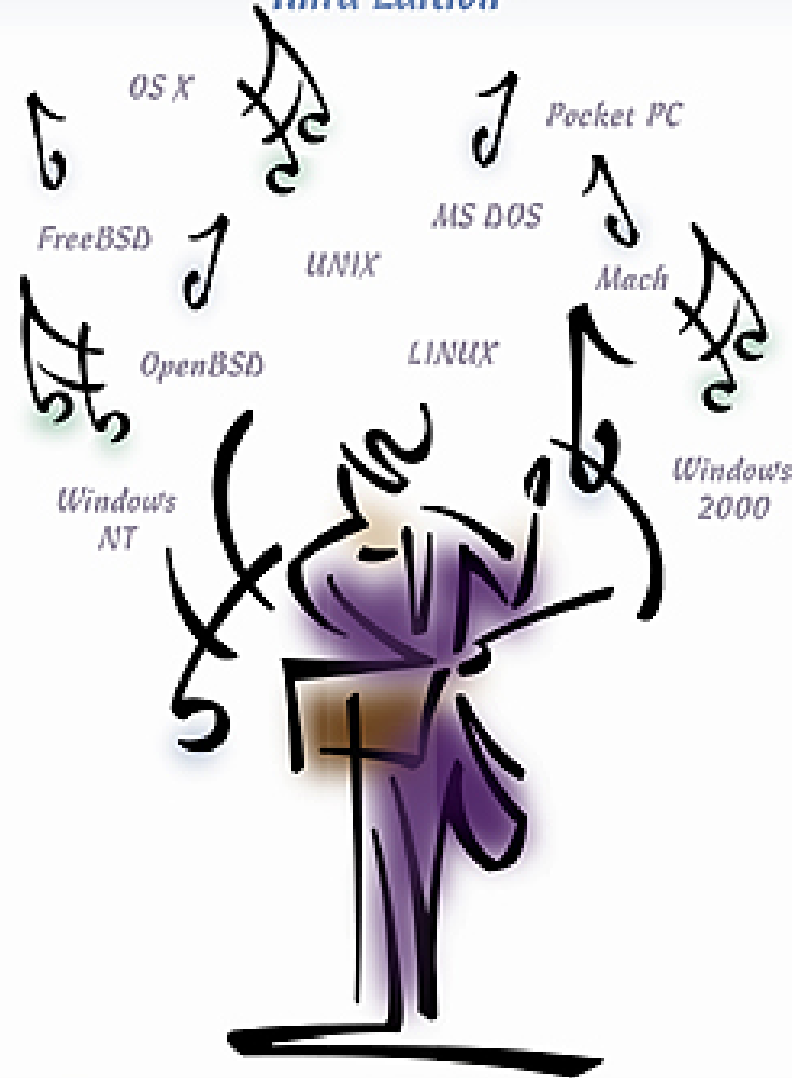


OPERATING SYSTEMS

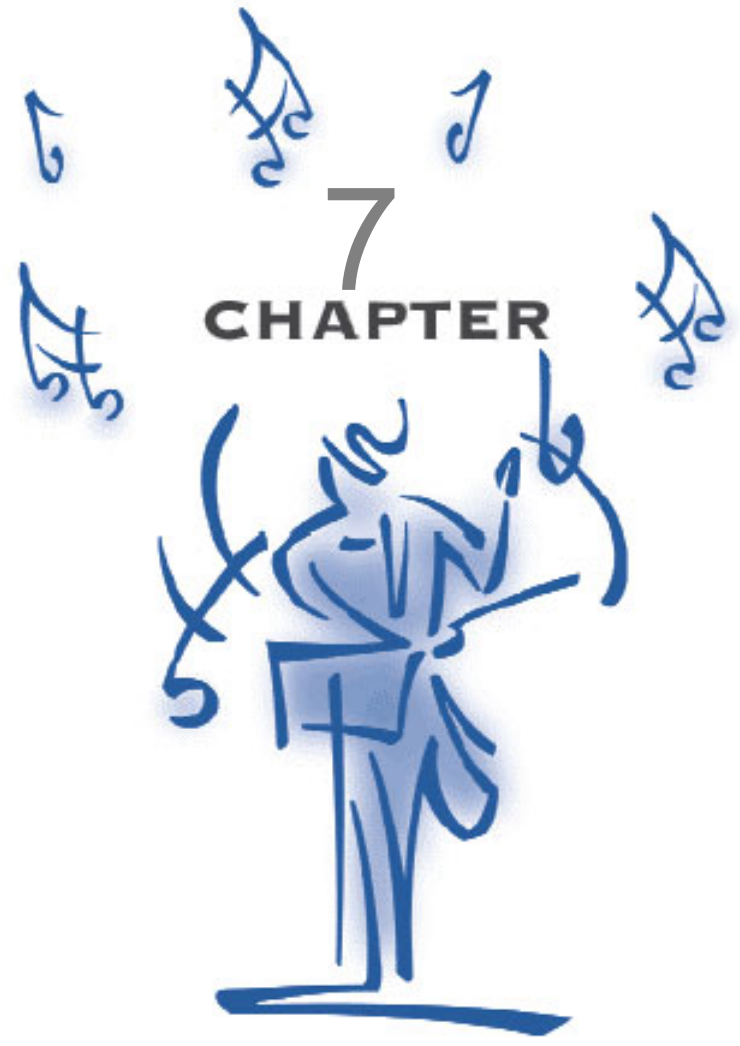
Third Edition



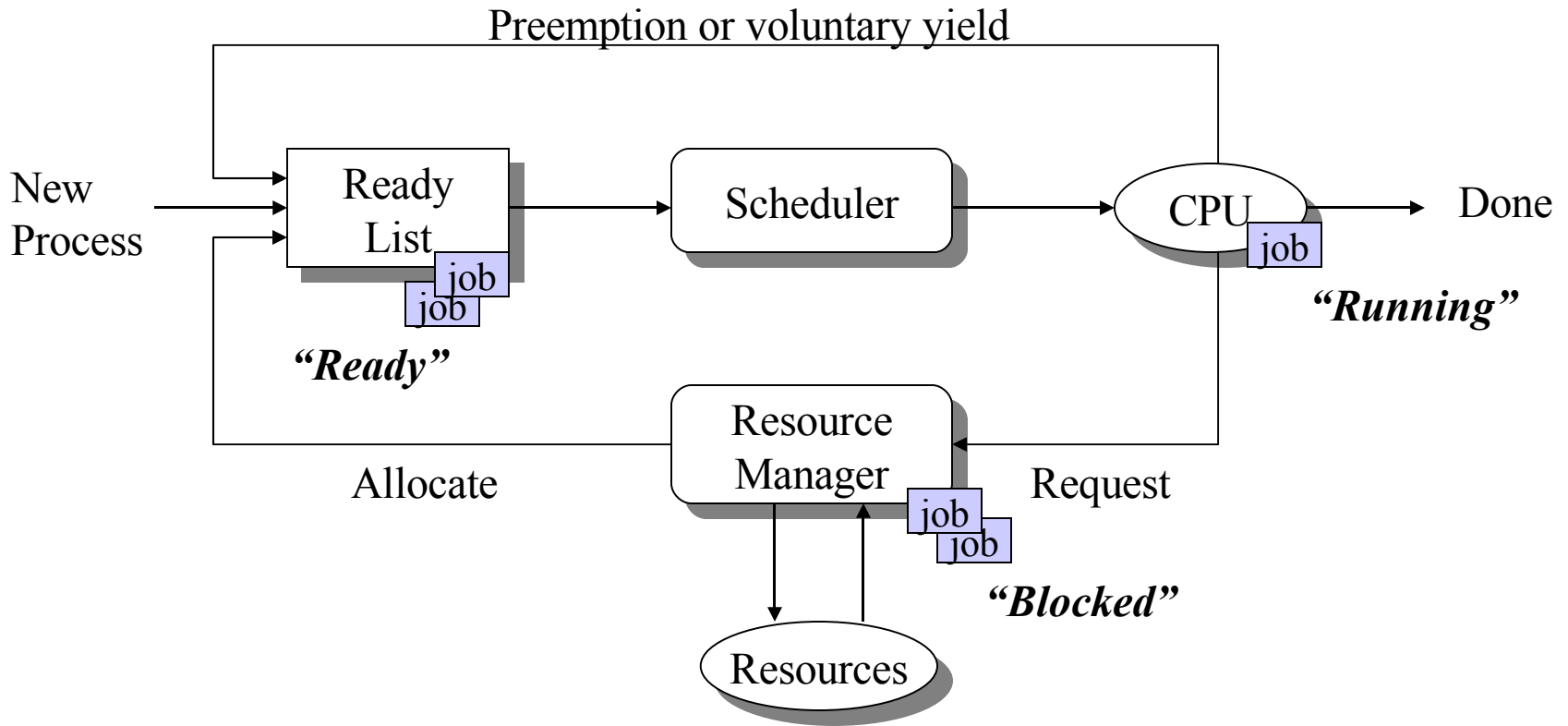
GARY NUTT



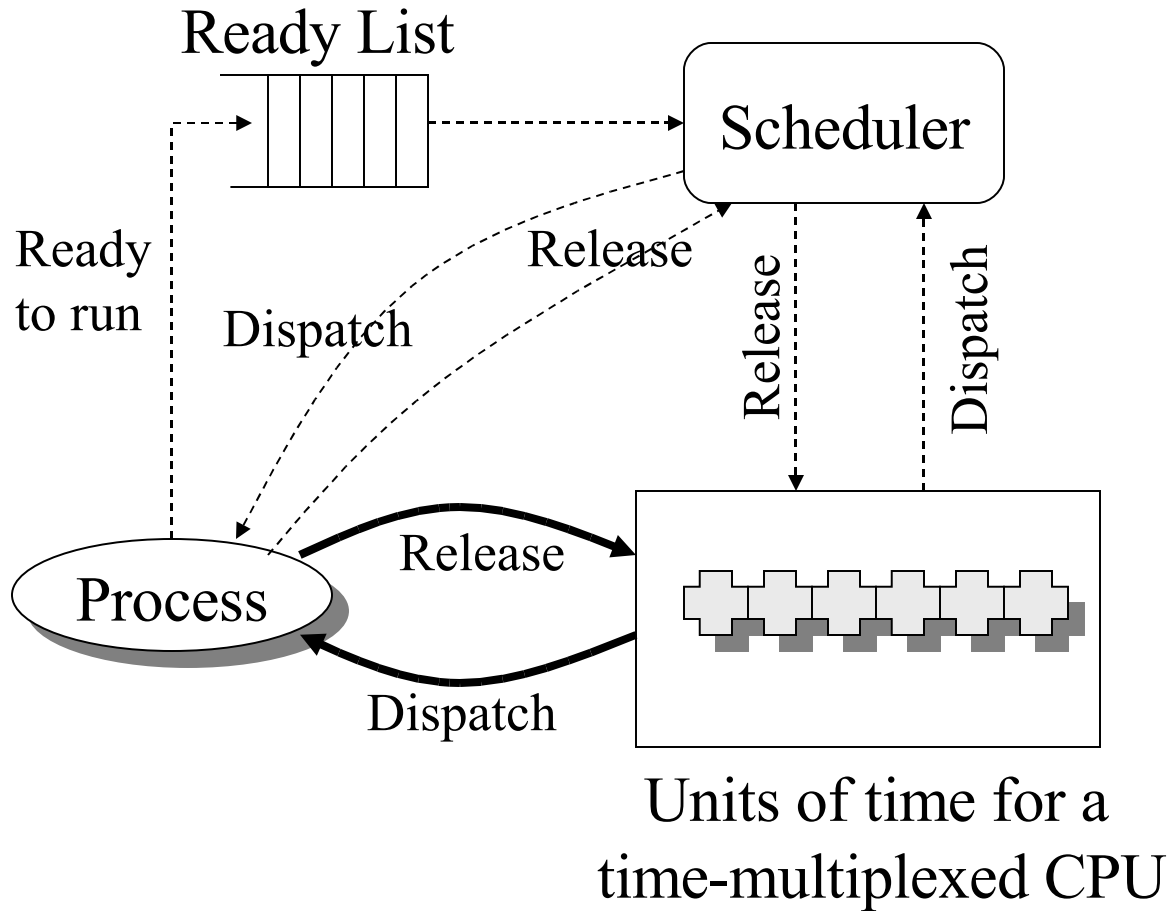
Scheduling



Model of Process Execution



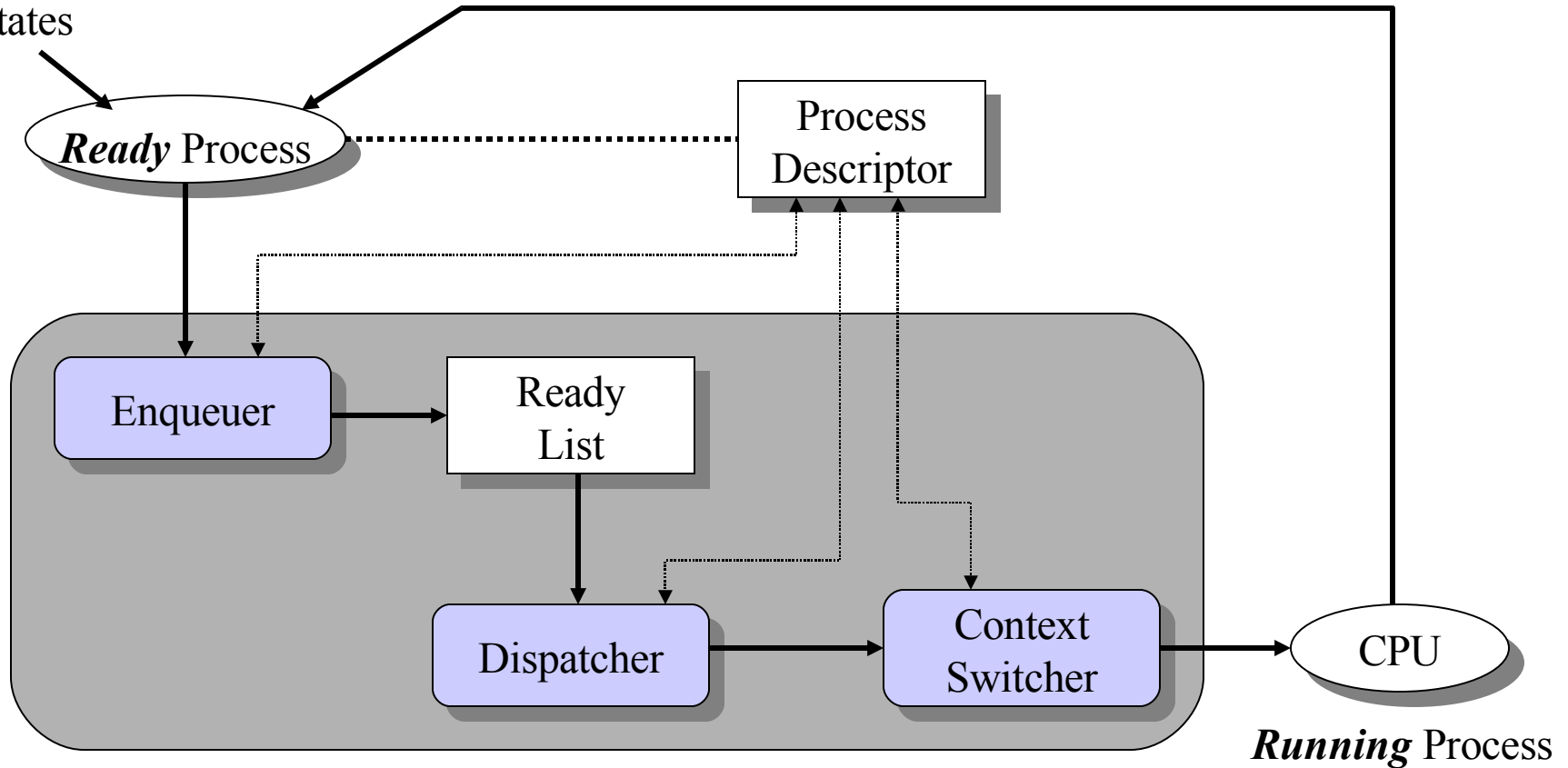
Scheduler as CPU Resource Manager



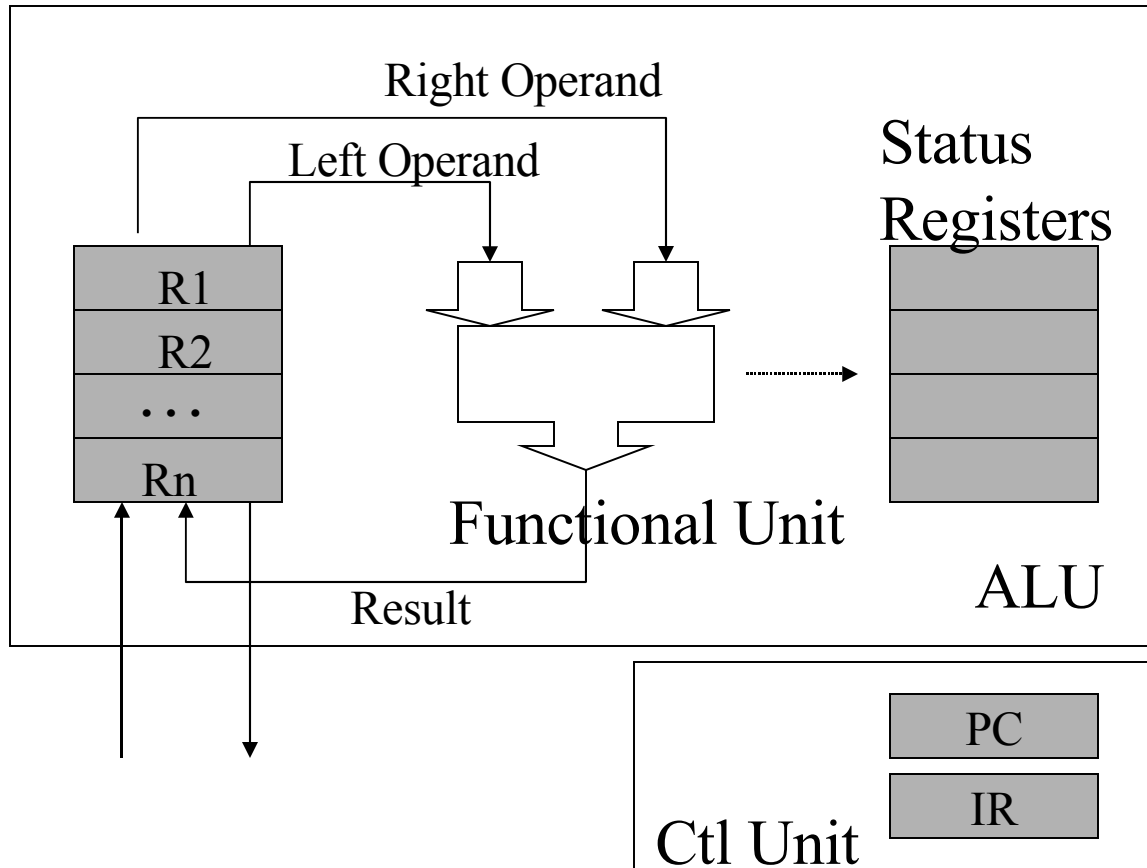
The Scheduler



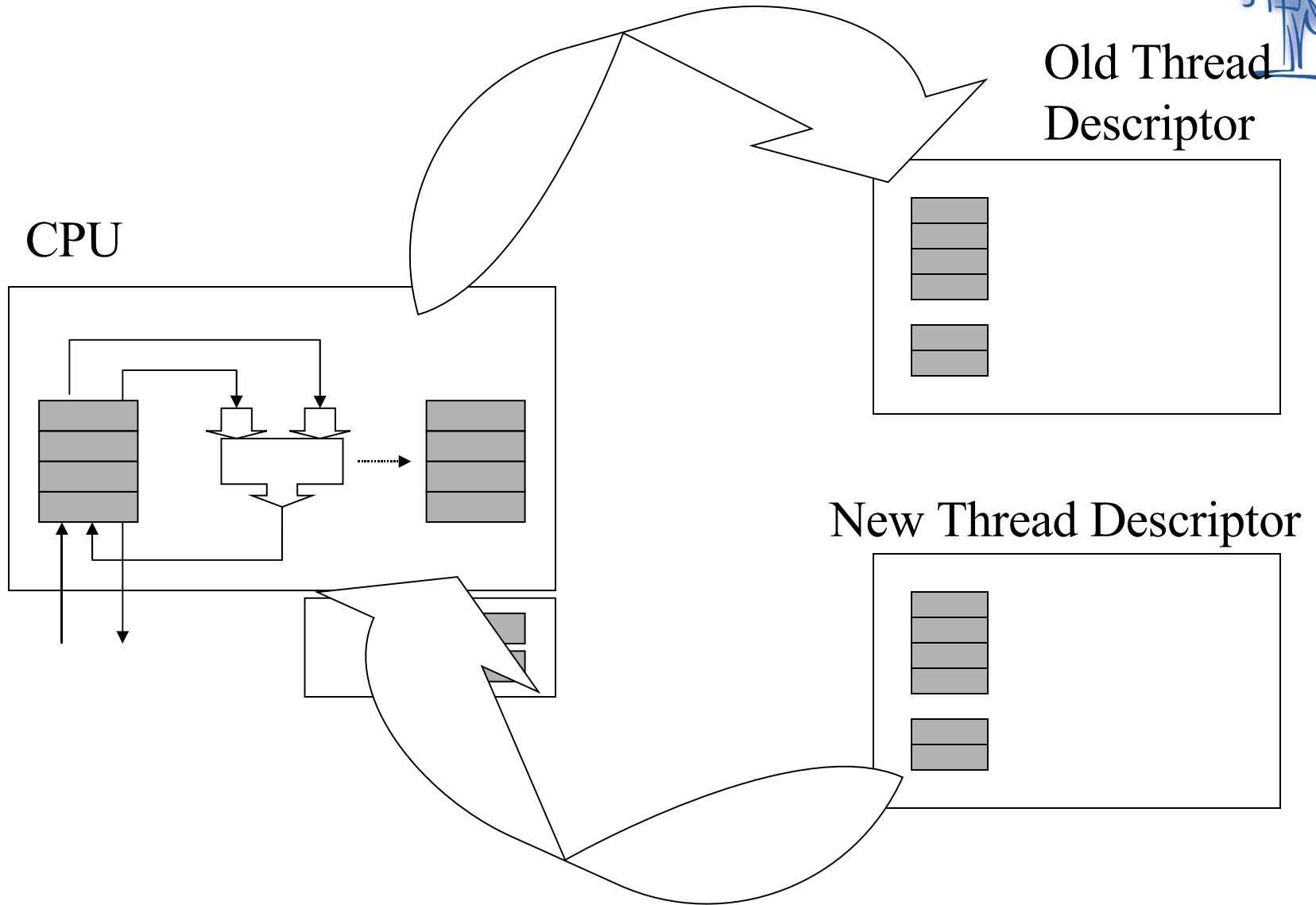
From
Other
States



Process/Thread Context



Context Switching



Invoking the Scheduler



- Need a mechanism to call the scheduler
- Voluntary call
 - Process blocks itself
 - Calls the scheduler
- Involuntary call
 - External force (interrupt) blocks the process
 - Calls the scheduler

Voluntary CPU Sharing



```
yield(pi.pc, pj.pc) {  
    memory[pi.pc] = PC;  
    PC = memory[pj.pc];  
}
```

- p_i can be “automatically” determined from the processor status registers

```
yield(*, pj.pc) {  
    memory[pi.pc] = PC;  
    PC = memory[pj.pc];  
}
```

More on Yield



- p_i and p_j can resume one another's execution

```
yield(*, p_j.pc);
```

```
• • •
```

```
yield(*, p_i.pc);
```

```
• • •
```

```
yield(*, p_j.pc);
```

- Suppose p_j is the scheduler:

```
// p_i yields to scheduler
```

```
yield(*, p_j.pc);
```

```
// scheduler chooses  $p_k$ 
```

```
yield(*, p_k.pc);
```

```
//  $p_k$  yields to scheduler
```

```
yield(*, p_j.pc);
```

```
// scheduler chooses
```

Voluntary Sharing



- Every process periodically yields to the scheduler
- Relies on correct process behavior
 - Malicious
 - Accidental
- Need a mechanism to override running process

Involuntary CPU Sharing



- Interval timer
 - Device to produce a periodic interrupt
 - Programmable period

```
IntervalTimer() {
    InterruptCount--;
    if(InterruptCount <= 0) {
        InterruptRequest = TRUE;
        InterruptCount = K;
    }
}

SetInterval(programmableValue) {
    K = programmableValue;
    InterruptCount = K;
}
}
```

Involuntary CPU Sharing (cont)



- Interval timer device handler
 - Keeps an in-memory clock up-to-date (see Chap 4 lab exercise)
 - Invokes the scheduler

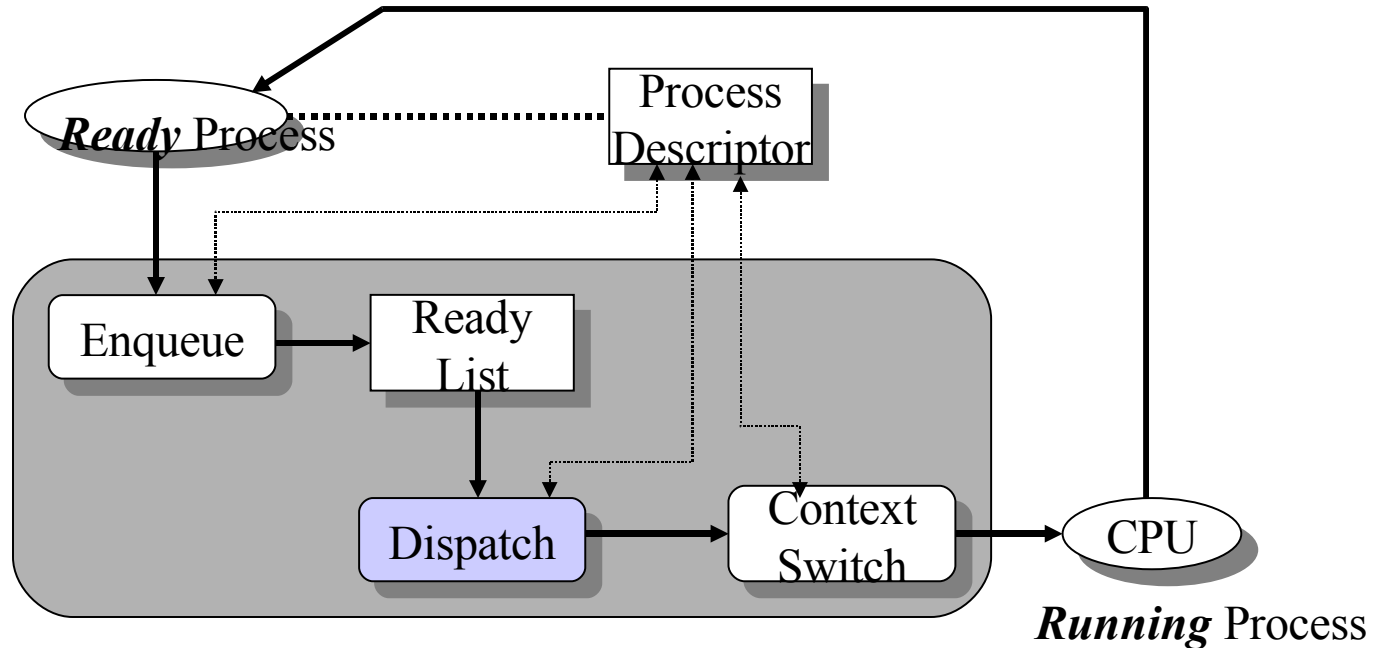
```
IntervalTimerHandler() {
    Time++; // update the clock
    TimeToSchedule--;
    if(TimeToSchedule <= 0) {
        <invoke scheduler>;
        TimeToSchedule = TimeSlice;
    }
}
```

Contemporary Scheduling



- Involuntary CPU sharing – timer interrupts
 - Time quantum determined by interval timer – usually fixed size for every process using the system
 - Sometimes called the time slice length

Choosing a Process to Run



- Mechanism never changes
- Strategy = policy the dispatcher uses to select a process from the ready list
- Different policies for different requirements

Policy Considerations



- Policy can control/influence:
 - CPU utilization
 - Average time a process waits for service
 - Average amount of time to complete a job
- Could strive for any of:
 - Equitability
 - Favor very short or long jobs
 - Meet priority requirements
 - Meet deadlines



Optimal Scheduling

- Suppose the scheduler knows each process p_i 's service time, p_i -- or it can estimate each p_i :
- Policy can optimize on any criteria, e.g.,
 - CPU utilization
 - Waiting time
 - Deadline
- To find an optimal schedule:
 - Have a finite, fixed # of p_i
 - Know p_i for each p_i
 - Enumerate all schedules, then choose the best

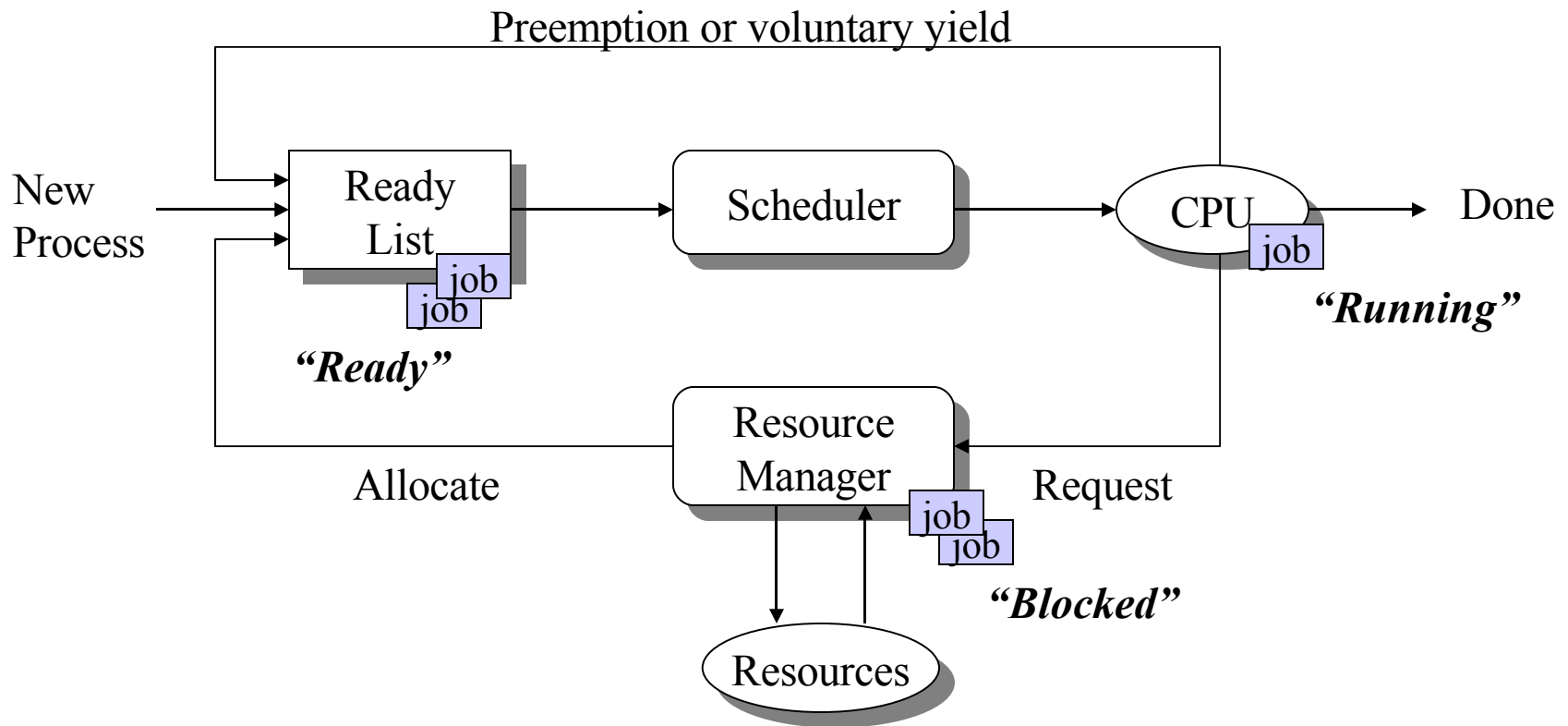


However ...

- The (p_i) are almost certainly just estimates
- General algorithm to choose optimal schedule is $O(n^2)$
- Other processes may arrive while these processes are being serviced
- Usually, optimal schedule is only a theoretical benchmark – scheduling policies try to approximate an optimal schedule



Model of Process Execution



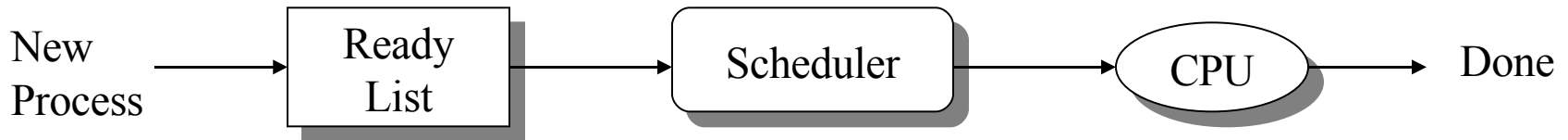


Talking About Scheduling ...

- Let $P = \{p_i \mid 0 \leq i < n\}$ = set of processes
- Let $S(p_i) \in \{\text{running, ready, blocked}\}$
- Let $t_{\text{run}}(p_i)$ = Time process needs to be in running state (the service time)
- Let $W(p_i)$ = Time p_i is in ready state before first transition to running (wait time)
- Let $T_{\text{TRnd}}(p_i)$ = Time from p_i first enter ready to last exit ready (turnaround time)
- Batch Throughput rate = inverse of avg T_{TRnd}

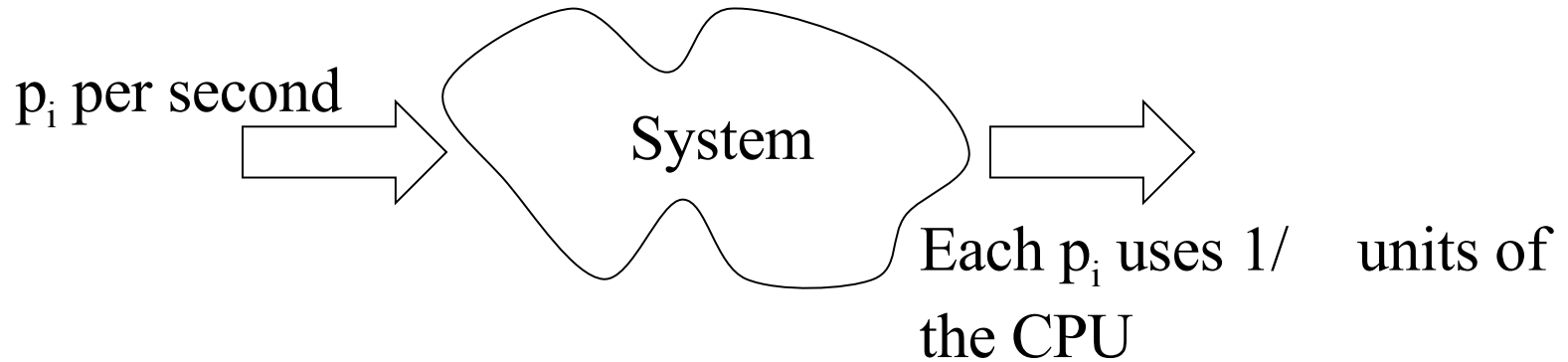
• Timesharing response time = $W(p_i)$

Estimating CPU Utilization



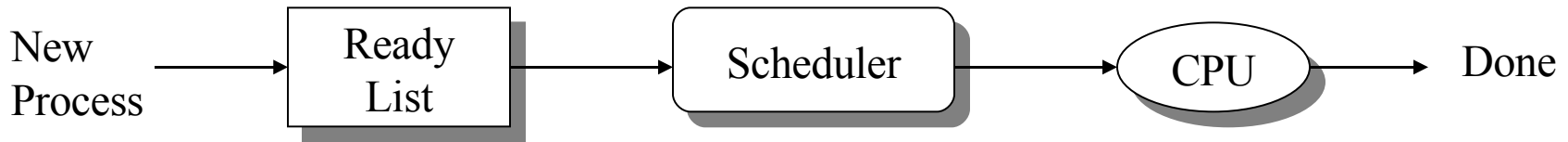
Let λ = the average rate at which processes are placed in the Ready List, arrival rate

Let μ = the average service rate
 $1/\mu$ = the average (p_i)





Estimating CPU Utilization



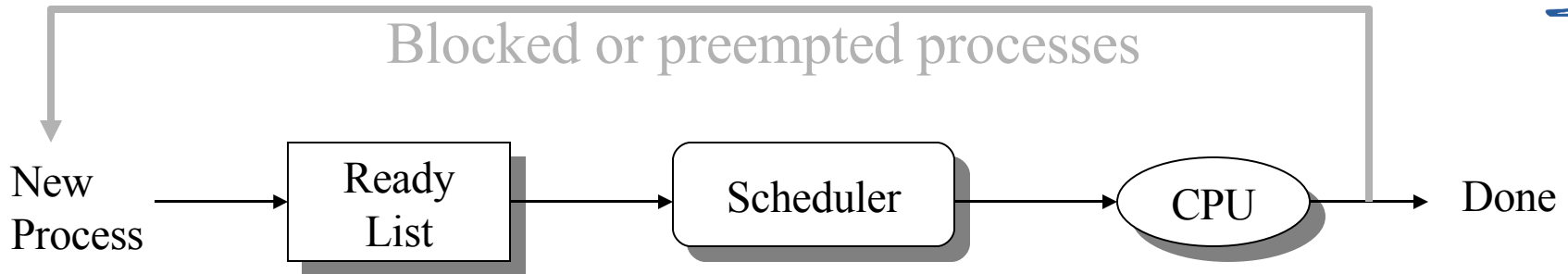
Let λ = the average rate at which processes are placed in the Ready List, arrival rate

Let μ = the average service rate
 $1/\mu$ = the average (p_i)

Let ρ = the fraction of the time that the CPU is expected to be busy
 $= \# p_i \text{ that arrive per unit time} * \text{avg time each spends on CPU}$
 $= \lambda * 1/\mu = \lambda / \mu$

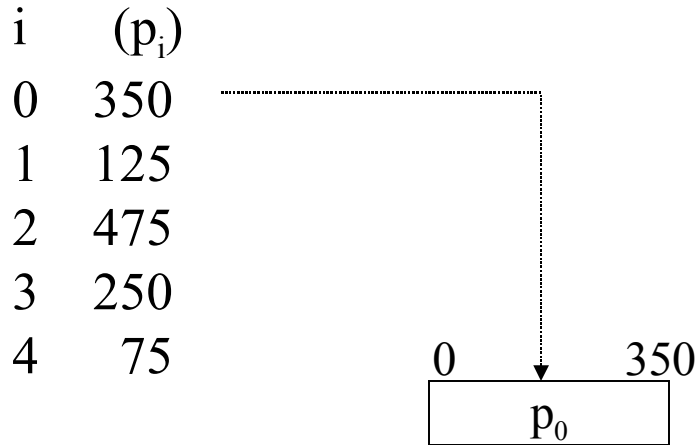
- Notice must have $\lambda < \mu$ (i.e., $\rho < 1$)
- What if ρ approaches 1?

Nonpreemptive Schedulers



- Try to use the simplified scheduling model
- Only consider running and ready states
- Ignores time in blocked state:
 - “New process created when it enters ready state”
 - “Process is destroyed when it enters blocked state”
 - Really just looking at “small phases” of a process

First-Come-First-Served



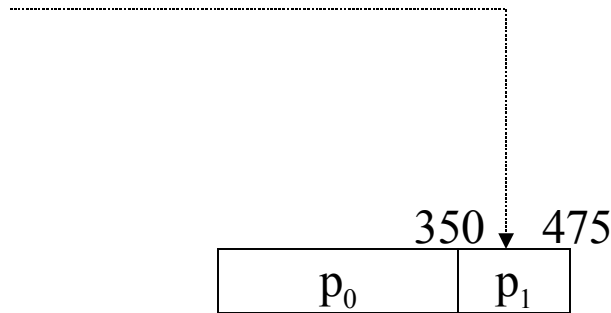
$$T_{\text{TRnd}}(p_0) = (p_0) = 350$$

$$W(p_0) = 0$$



First-Come-First-Served

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = (p_0) = 350$$

$$W(p_0) = 0$$

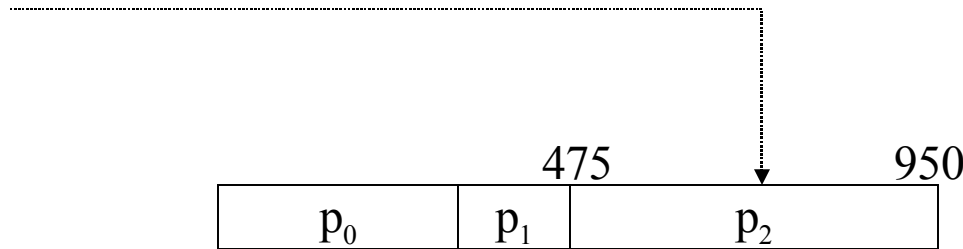
$$T_{\text{TRnd}}(p_1) = (p_1) + T_{\text{TRnd}}(p_0) = 125 + 350 = 475$$

$$W(p_1) = T_{\text{TRnd}}(p_0) = 350$$



First-Come-First-Served

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = (p_0) = 350$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) = (p_1) + T_{\text{TRnd}}(p_0) = 125 + 350 = 475$$

$$W(p_1) = T_{\text{TRnd}}(p_0) = 350$$

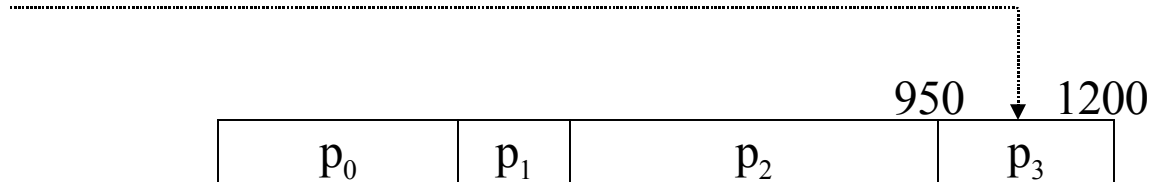
$$T_{\text{TRnd}}(p_2) = (p_2) + T_{\text{TRnd}}(p_1) = 475 + 475 = 950$$

$$W(p_2) = T_{\text{TRnd}}(p_1) = 475$$



First-Come-First-Served

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = (p_0) = 350$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) = (p_1) + T_{\text{TRnd}}(p_0) = 125 + 350 = 475$$

$$W(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$T_{\text{TRnd}}(p_2) = (p_2) + T_{\text{TRnd}}(p_1) = 475 + 475 = 950$$

$$W(p_2) = T_{\text{TRnd}}(p_1) = 475$$

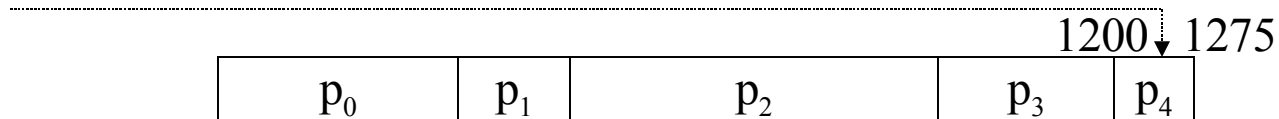
$$T_{\text{TRnd}}(p_3) = (p_3) + T_{\text{TRnd}}(p_2) = 250 + 950 = 1200$$

$$W(p_3) = T_{\text{TRnd}}(p_2) = 950$$



First-Come-First-Served

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = (p_0) = 350$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) = (p_1) + T_{\text{TRnd}}(p_0) = 125 + 350 = 475$$

$$W(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$T_{\text{TRnd}}(p_2) = (p_2) + T_{\text{TRnd}}(p_1) = 475 + 475 = 950$$

$$W(p_2) = T_{\text{TRnd}}(p_1) = 475$$

$$T_{\text{TRnd}}(p_3) = (p_3) + T_{\text{TRnd}}(p_2) = 250 + 950 = 1200$$

$$W(p_3) = T_{\text{TRnd}}(p_2) = 950$$

$$T_{\text{TRnd}}(p_4) = (p_4) + T_{\text{TRnd}}(p_3) = 75 + 1200 = 1275$$

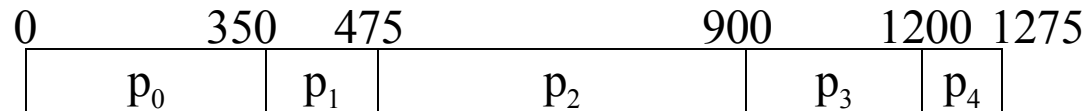
$$W(p_4) = T_{\text{TRnd}}(p_3) = 1200$$

FCFS Average Wait Time



i	(p _i)
0	350
1	125
2	475
3	250
4	75

- Easy to implement
- Ignores service time, etc
- Not a great performer



$$T_{\text{TRnd}}(p_0) = (p_0) = 350$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) = (p_1) + T_{\text{TRnd}}(p_0) = 125 + 350 = 475$$

$$W(p_1) = T_{\text{TRnd}}(p_0) = 350$$

$$T_{\text{TRnd}}(p_2) = (p_2) + T_{\text{TRnd}}(p_1) = 475 + 475 = 950$$

$$W(p_2) = T_{\text{TRnd}}(p_1) = 475$$

$$T_{\text{TRnd}}(p_3) = (p_3) + T_{\text{TRnd}}(p_2) = 250 + 950 = 1200$$

$$W(p_3) = T_{\text{TRnd}}(p_2) = 950$$

$$T_{\text{TRnd}}(p_4) = (p_4) + T_{\text{TRnd}}(p_3) = 75 + 1200 = 1275$$

$$W(p_4) = T_{\text{TRnd}}(p_3) = 1200$$

$$W_{\text{avg}} = (0 + 350 + 475 + 950 + 1200) / 5 = 2974 / 5 = 595$$



Predicting Wait Time in FCFS

- In FCFS, when a process arrives, all in ready list will be processed before this job
- Let μ be the service rate
- Let L be the ready list length
- $W_{avg}(p) = L * 1/\mu$
- Compare predicted wait with actual in earlier examples



Shortest Job Next

i	(p_i)
0	350
1	125
2	475
3	250
4	75



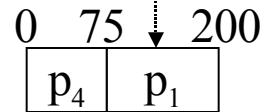
$$T_{\text{TRnd}}(p_4) = (p_4) = 75$$

$$W(p_4) = 0$$



Shortest Job Next

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_1) = (p_1) + (p_4) = 125 + 75 = 200$$

$$W(p_1) = 75$$

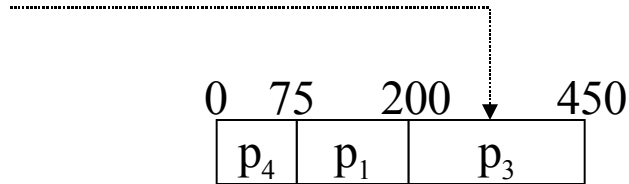
$$T_{\text{TRnd}}(p_4) = (p_4) = 75$$

$$W(p_4) = 0$$



Shortest Job Next

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_1) = (p_1) + (p_4) = 125 + 75 = 200$$

$$W(p_1) = 75$$

$$T_{\text{TRnd}}(p_3) = (p_3) + (p_1) + (p_4) = 250 + 125 + 75 = 450$$

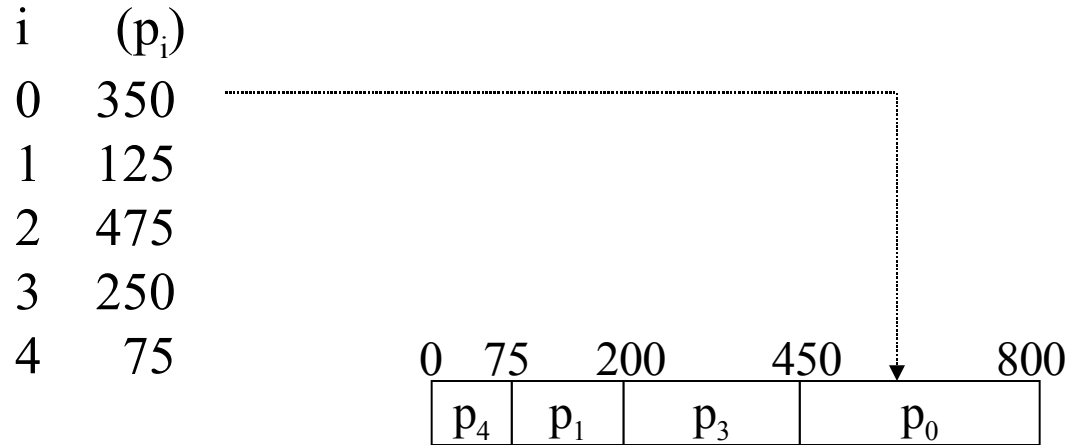
$$W(p_3) = 200$$

$$T_{\text{TRnd}}(p_4) = (p_4) = 75$$

$$W(p_4) = 0$$



Shortest Job Next



$$T_{\text{TRnd}}(p_0) = (p_0) + (p_3) + (p_1) + (p_4) = 350 + 250 + 125 + 75 = 800$$

$$W(p_0) = 450$$

$$T_{\text{TRnd}}(p_1) = (p_1) + (p_4) = 125 + 75 = 200$$

$$W(p_1) = 75$$

$$T_{\text{TRnd}}(p_3) = (p_3) + (p_1) + (p_4) = 250 + 125 + 75 = 450$$

$$W(p_3) = 200$$

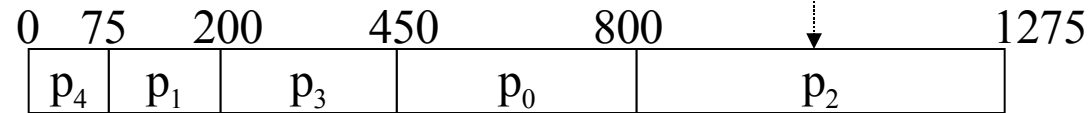
$$T_{\text{TRnd}}(p_4) = (p_4) = 75$$

$$W(p_4) = 0$$



Shortest Job Next

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) = (p_0) + (p_3) + (p_1) + (p_4) = 350 + 250 + 125 + 75 = 800$$

$$W(p_0) = 450$$

$$T_{\text{TRnd}}(p_1) = (p_1) + (p_4) = 125 + 75 = 200$$

$$W(p_1) = 75$$

$$T_{\text{TRnd}}(p_2) = (p_2) + (p_0) + (p_3) + (p_1) + (p_4) = 475 + 350 + 250 + 125 + 75 = 1275$$

$$W(p_2) = 800$$

$$T_{\text{TRnd}}(p_3) = (p_3) + (p_1) + (p_4) = 250 + 125 + 75 = 450$$

$$W(p_3) = 200$$

$$T_{\text{TRnd}}(p_4) = (p_4) = 75$$

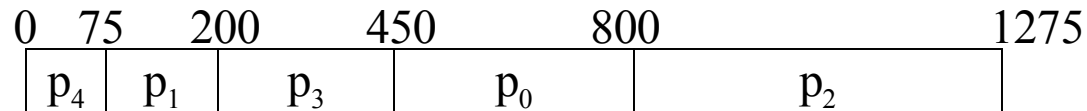
$$W(p_4) = 0$$



Shortest Job Next

i	(p _i)
0	350
1	125
2	475
3	250
4	75

- Minimizes wait time
- May starve large jobs
- Must know service times



$$T_{\text{TRnd}}(p_0) = (p_0) + (p_3) + (p_1) + (p_4) = 350 + 250 + 125 + 75 = 800$$

$$W(p_0) = 450$$

$$T_{\text{TRnd}}(p_1) = (p_1) + (p_4) = 125 + 75 = 200$$

$$W(p_1) = 75$$

$$T_{\text{TRnd}}(p_2) = (p_2) + (p_0) + (p_3) + (p_1) + (p_4) = 475 + 350 + 250 + 125 + 75 = 1275$$

$$W(p_2) = 800$$

$$T_{\text{TRnd}}(p_3) = (p_3) + (p_1) + (p_4) = 250 + 125 + 75 = 450$$

$$W(p_3) = 200$$

$$T_{\text{TRnd}}(p_4) = (p_4) = 75$$

$$W(p_4) = 0$$

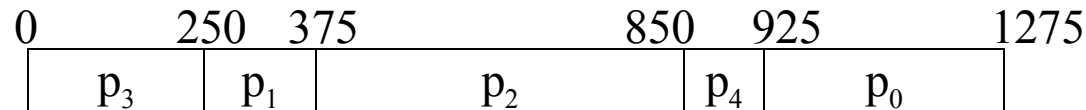
$$W_{\text{avg}} = (450 + 75 + 800 + 200 + 0) / 5 = 1525 / 5 = 305$$



Priority Scheduling

i	(p _i)	Pri
0	350	5
1	125	2
2	475	3
3	250	1
4	75	4

- Reflects importance of external use
- May cause starvation
- Can address starvation with aging



$$T_{\text{TRnd}}(p_0) = (p_0) + (p_4) + (p_2) + (p_1) + (p_3) = 350 + 75 + 475 + 125 + 250 = 1275 \quad W(p_0) = 925$$

$$T_{\text{TRnd}}(p_1) = (p_1) + (p_3) = 125 + 250 = 375 \quad W(p_1) = 250$$

$$T_{\text{TRnd}}(p_2) = (p_2) + (p_1) + (p_3) = 475 + 125 + 250 = 850 \quad W(p_2) = 375$$

$$T_{\text{TRnd}}(p_3) = (p_3) = 250 \quad W(p_3) = 0$$

$$T_{\text{TRnd}}(p_4) = (p_4) + (p_2) + (p_1) + (p_3) = 75 + 475 + 125 + 250 = 925 \quad W(p_4) = 850$$

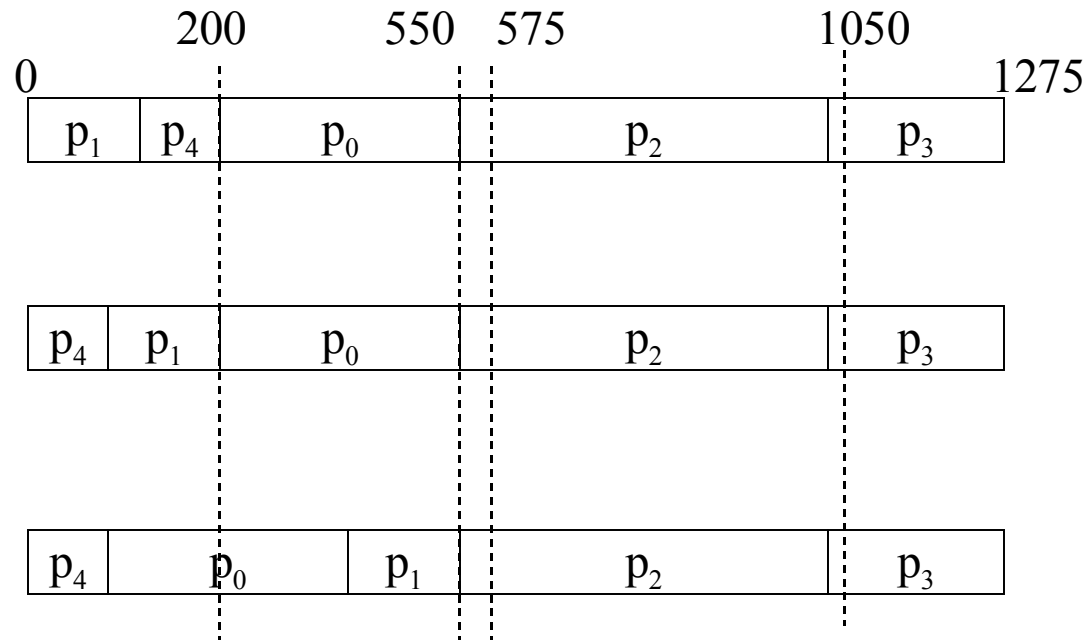
$$W_{\text{avg}} = (925 + 250 + 375 + 0 + 850) / 5 = 2400 / 5 = 480$$



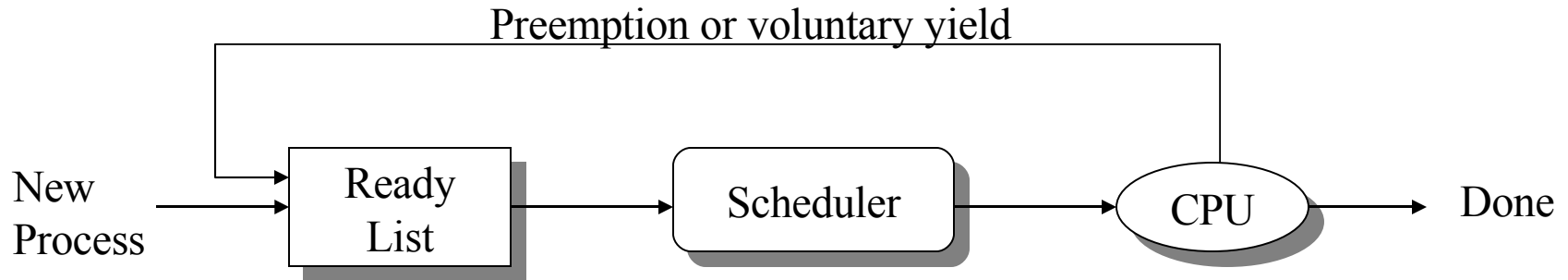
Deadline Scheduling

i	(p _i)	Deadline
0	350	575
1	125	550
2	475	1050
3	250	(none)
4	75	200

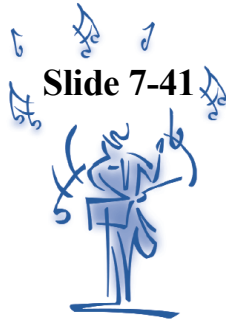
- Allocates service by deadline
- May not be feasible



Preemptive Schedulers

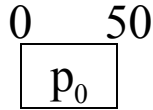


- Highest priority process is guaranteed to be running at all times
 - Or at least at the beginning of a time slice
- Dominant form of contemporary scheduling
- But complex to build & analyze



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75

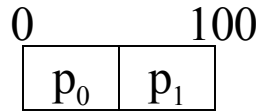


$$W(p_0) = 0$$



Round Robin (TQ=50)

i	(p_i)
0	350
1	125
2	475
3	250
4	75



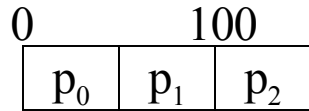
$$W(p_0) = 0$$

$$W(p_1) = 50$$



Round Robin (TQ=50)

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$W(p_0) = 0$$

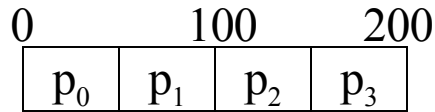
$$W(p_1) = 50$$

$$W(p_2) = 100$$



Round Robin (TQ=50)

i	(p_i)
0	350
1	125
2	475
3	250
4	75



$$W(p_0) = 0$$

$$W(p_1) = 50$$

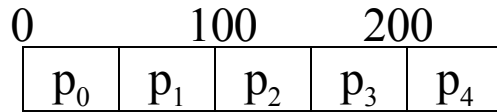
$$W(p_2) = 100$$

$$W(p_3) = 150$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$W(p_0) = 0$$

$$W(p_1) = 50$$

$$W(p_2) = 100$$

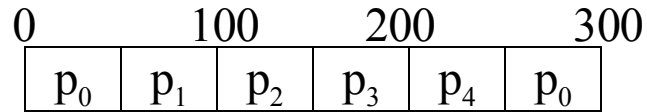
$$W(p_3) = 150$$

$$W(p_4) = 200$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$W(p_0) = 0$$

$$W(p_1) = 50$$

$$W(p_2) = 100$$

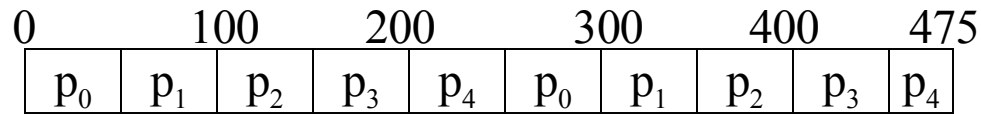
$$W(p_3) = 150$$

$$W(p_4) = 200$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_4) =$$

$$W(p_0) = 0$$

$$W(p_1) = 50$$

$$W(p_2) = 100$$

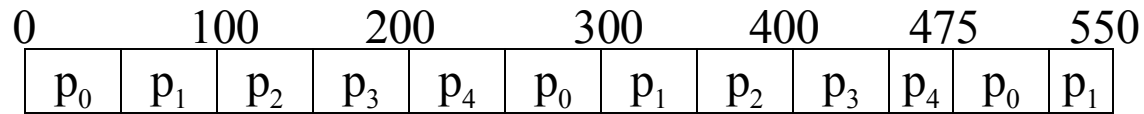
$$W(p_3) = 150$$

$$W(p_4) = 200$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_1) =$$

$$T_{\text{TRnd}}(p_4) =$$

$$W(p_0) = 0$$

$$W(p_1) = 50$$

$$W(p_2) = 100$$

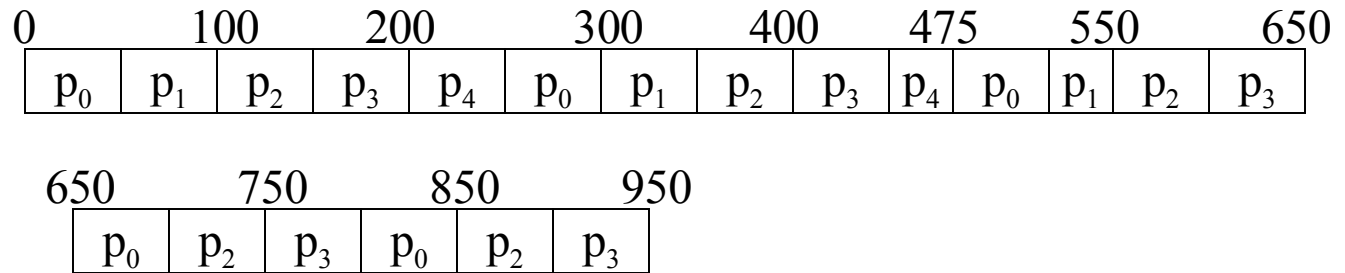
$$W(p_3) = 150$$

$$W(p_4) = 200$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75

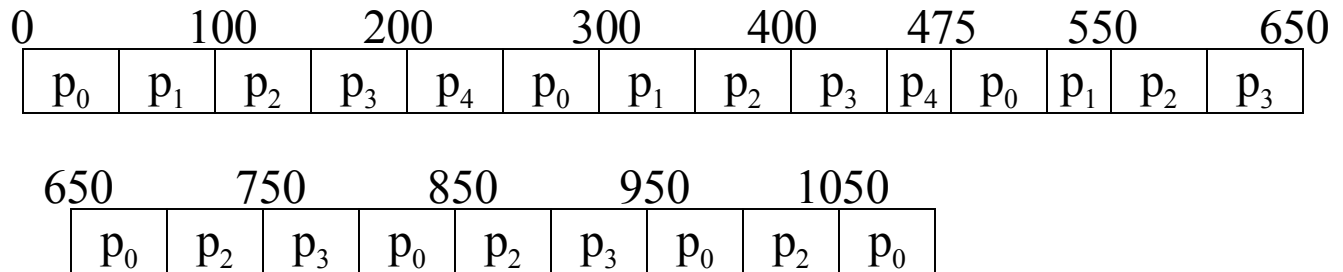


$T_{\text{TRnd}}(p_1) =$	$W(p_0) = 0$
$T_{\text{TRnd}}(p_3) =$	$W(p_1) = 50$
$T_{\text{TRnd}}(p_4) =$	$W(p_2) = 100$
	$W(p_3) = 150$
	$W(p_4) = 200$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) =$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) =$$

$$W(p_1) = 50$$

$$W(p_2) = 100$$

$$T_{\text{TRnd}}(p_3) =$$

$$W(p_3) = 150$$

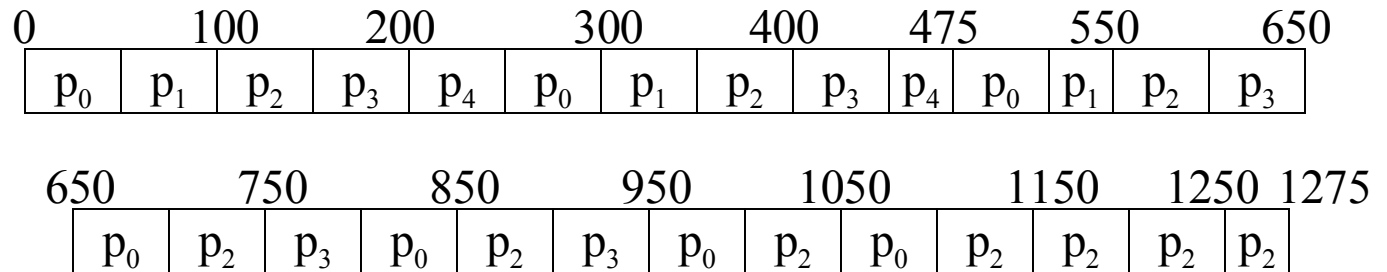
$$T_{\text{TRnd}}(p_4) =$$

$$W(p_4) = 200$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{\text{TRnd}}(p_0) =$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) =$$

$$W(p_1) = 50$$

$$T_{\text{TRnd}}(p_2) =$$

$$W(p_2) = 100$$

$$T_{\text{TRnd}}(p_3) =$$

$$W(p_3) = 150$$

$$T_{\text{TRnd}}(p_4) =$$

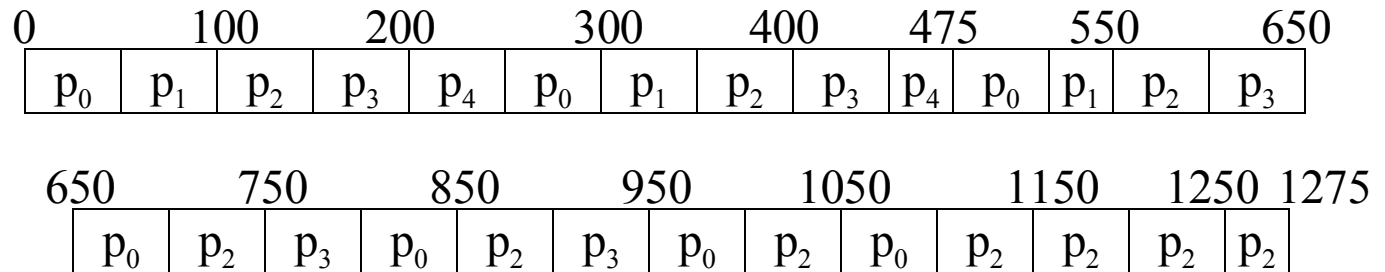
$$W(p_4) = 200$$



Round Robin (TQ=50)

i	(p _i)
0	350
1	125
2	475
3	250
4	75

- Equitable
- Most widely-used
- Fits naturally with interval timer



$$T_{\text{TRnd}}(p_0) =$$

$$W(p_0) = 0$$

$$T_{\text{TRnd}}(p_1) =$$

$$W(p_1) = 50$$

$$T_{\text{TRnd}}(p_2) =$$

$$W(p_2) = 100$$

$$T_{\text{TRnd}}(p_3) =$$

$$W(p_3) = 150$$

$$T_{\text{TRnd}}(p_4) =$$

$$W(p_4) = 200$$

$$T_{\text{TRnd-avg}} = (1100 + 550 + 1275 + 950 + 475) / 5 = 4350 / 5 = 870$$

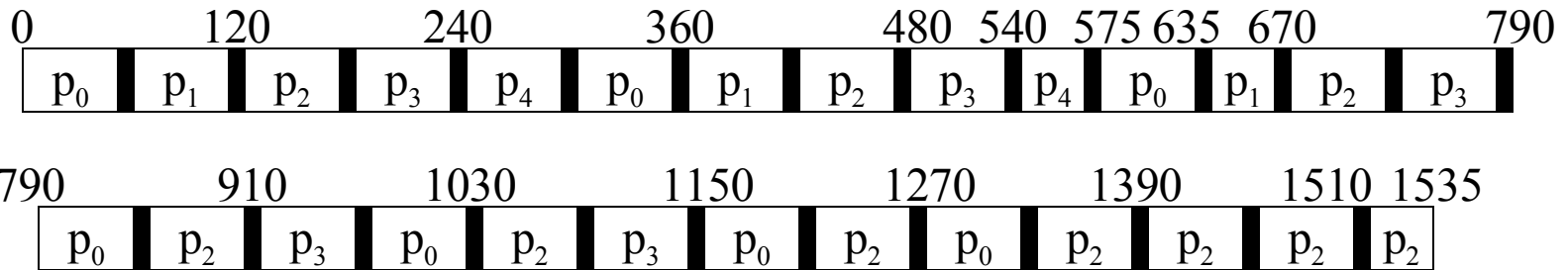
$$W_{\text{avg}} = (0 + 50 + 100 + 150 + 200) / 5 = 500 / 5 = 100$$



RR with Overhead=10 (TQ=50)

•Overhead must be considered

i	(p _i)
0	350
1	125
2	475
3	250
4	75



$$T_{TRnd}(p_0) =$$

$$W(p_0) = 0$$

$$T_{TRnd}(p_1) =$$

$$W(p_1) = 60$$

$$T_{TRnd}(p_2) =$$

$$W(p_2) = 120$$

$$T_{TRnd}(p_3) =$$

$$W(p_3) = 180$$

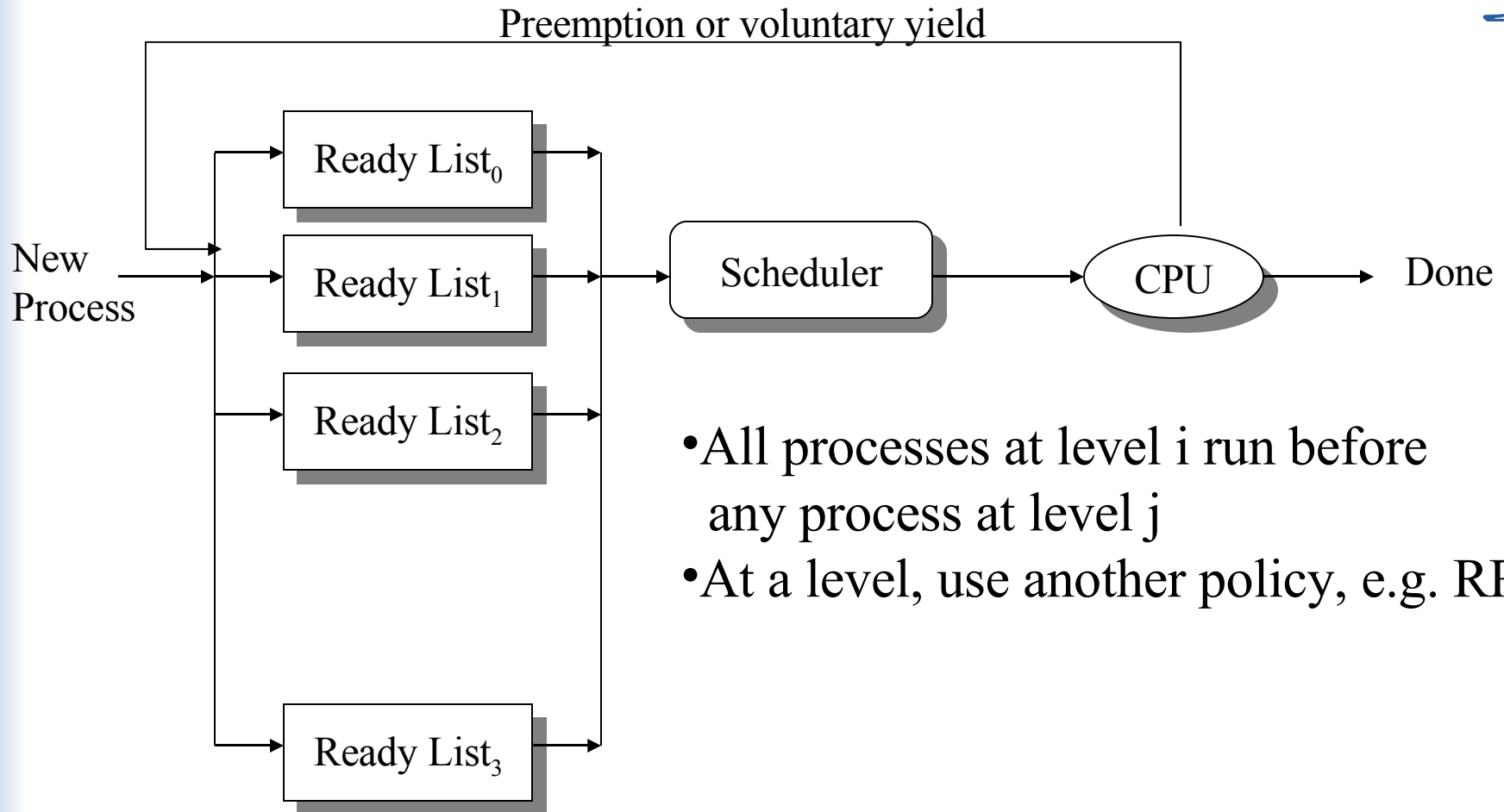
$$T_{TRnd}(p_4) =$$

$$W(p_4) = 240$$

$$T_{TRnd-avg} = (1320+660+1535+1140+565)/5 = 5220/5 = 1044$$

$$W_{avg} = (0+60+120+180+240)/5 = 600/5 = 120$$

Multi-Level Queues





Contemporary Scheduling

- Involuntary CPU sharing -- timer interrupts
 - Time quantum determined by interval timer -- usually fixed for every process using the system
 - Sometimes called the time slice length
- Priority-based process (job) selection
 - Select the highest priority process
 - Priority reflects policy
- With preemption
- Usually a variant of Multi-Level Queues

BSD 4.4 Scheduling



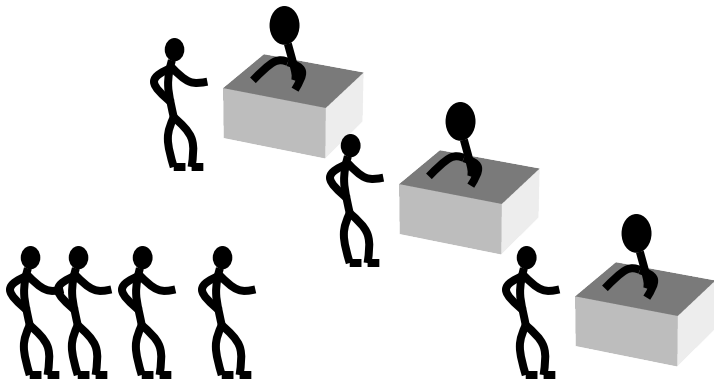
- Involuntary CPU Sharing
- Preemptive algorithms
- 32 Multi-Level Queues
 - Queues 0-7 are reserved for system functions
 - Queues 8-31 are for user space functions
 - `nice` influences (but does not dictate) queue level

Windows NT/2K Scheduling

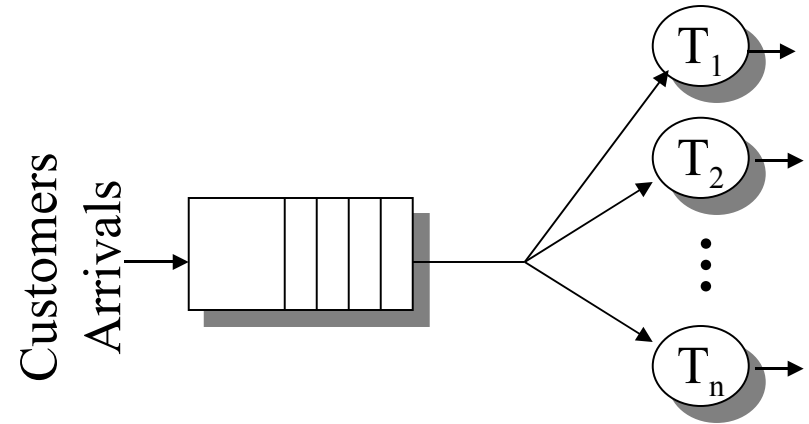


- Involuntary CPU Sharing across threads
- Preemptive algorithms
- 32 Multi-Level Queues
 - Highest 16 levels are “real-time”
 - Next lower 15 are for system/user threads
 - Range determined by process base priority
 - Lowest level is for the idle thread

Bank Teller Simulation



Tellers at the Bank



Model of Tellers at the Bank

Simulation Kernel Loop



```
simulated_time = 0;
while (true) {
    event = select_next_event();
    if (event->time > simulated_time)
        simulated_time = event->time;
    evaluate(event->function, ...);
}
```



Simulation Kernel Loop(2)

```
void runKernel(int quitTime)
{
    Event *thisEvent;

    // Stop by running to elapsed time, or by causing quit execute
    if(quitTime <= 0) quitTime = 9999999;
    simTime = 0;
    while(simTime < quitTime) {
        // Get the next event
        if(eventList == NIL) {
            // No more events to process
            break;
        }
        thisEvent = eventList;
        eventList = thisEvent->next;
        simTime = thisEvent->getTime(); // Set the time
        // Execute this event
        thisEvent->fire();
        delete(thisEvent);
    };
}
```