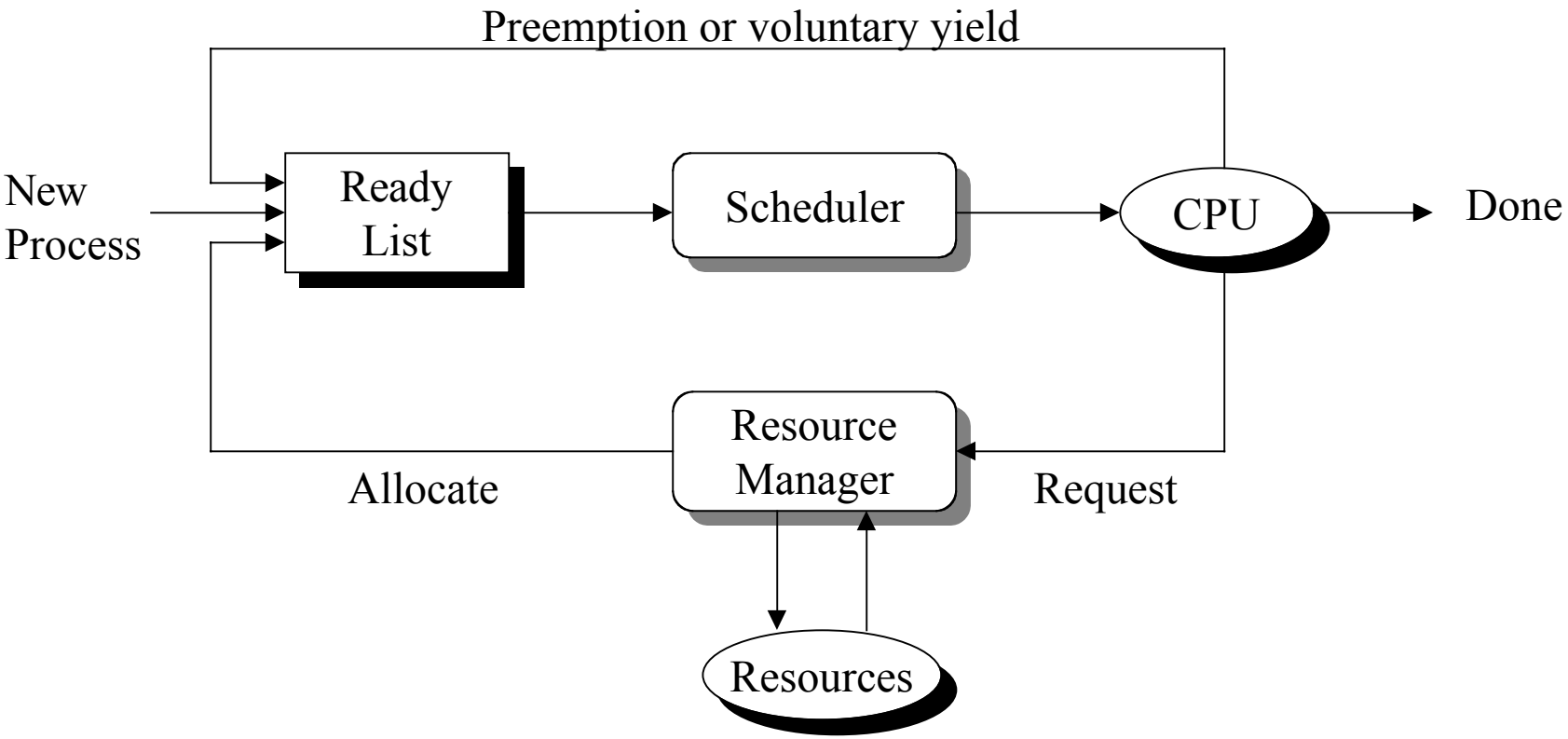
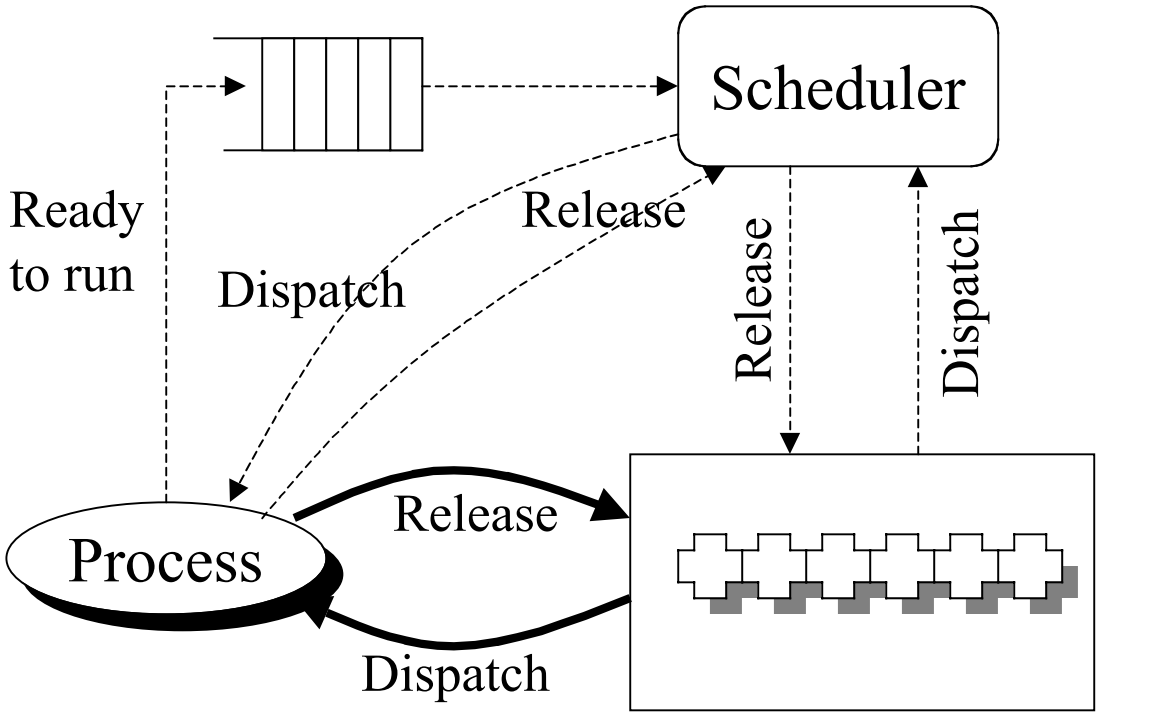
Scheduling

Process Execution

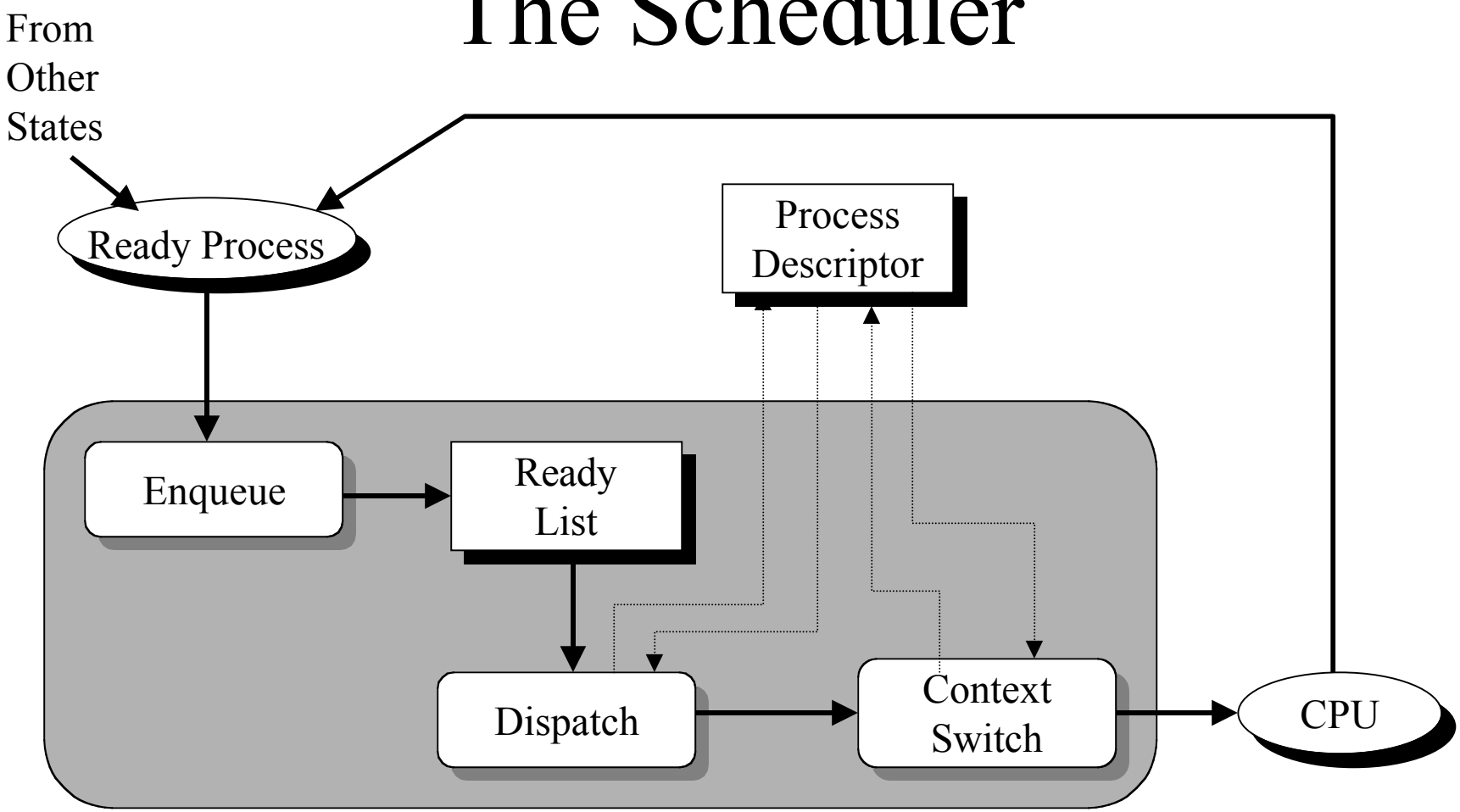


Scheduler as Resource Manager



Units of time for a
time-multiplexed CPU

The Scheduler



Voluntary CPU Sharing

```
yield(p_i.pc, p_j.pc) {
    memory[p_i.pc] = PC;
    PC = memory[p_j.pc];
}
```

- p_i can be "automatically" determined from the processor status registers

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

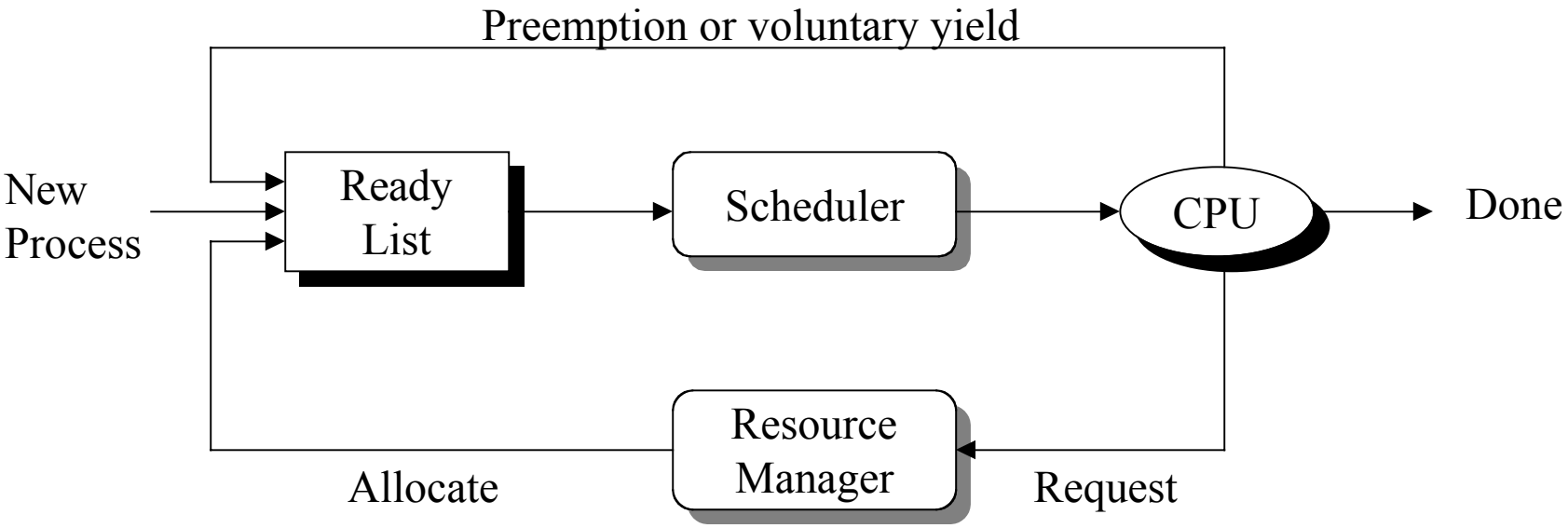
```
SetInterval(programmableValue) {
    K = programmableValue:
    InterruptCount = K;
    }
}
```

Involuntary CPU Sharing (cont)

- Interval timer device handler
  - Can keep an in-memory clock up-to-date
  - Can invoke the scheduler

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

Scheduling & Performance



- Scheduler can control:
  - CPU utilization
  - Average time a process waits for service
  - Average amount of time to complete a job

Scheduler Strategy

- Strategy = *policy* the scheduler mechanism uses to choose from the ready list
- Strive for any of:
  - Equitability
  - Favor very short or long jobs
  - Meet priority requirements
- Different policies for different requirements
- *Mechanism* never changes

Contemporary Scheduling Mechanism

- 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
- May or may not have *preemption*

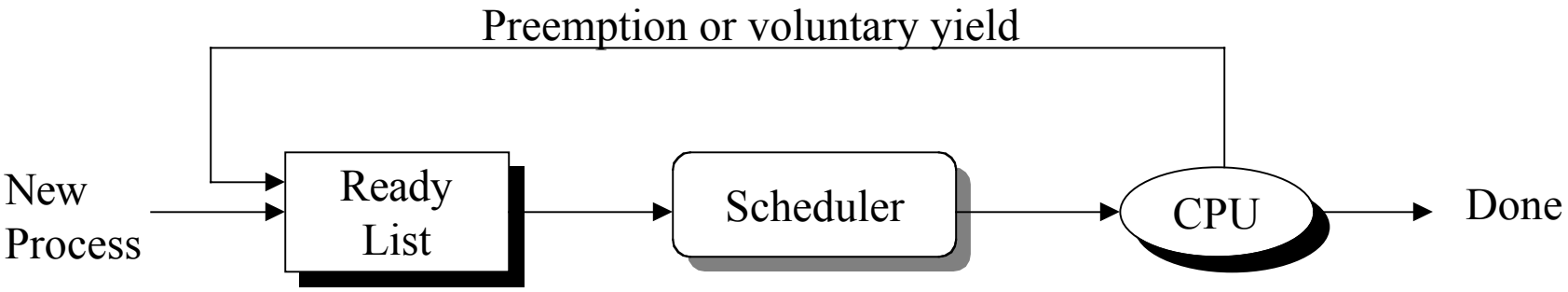
Optimal Scheduling

- Suppose the scheduler knows each process p_i's *service time*, $\tau(p_i)$.
- Could look at every p_i and $\tau(p_i)$ in the ready list and choose the scheduler that optimized on any desired criteria
- But
  - Other processes may arrive while these processes are being serviced
  - The $\tau(p_i)$ are almost certainly just estimates
  - Algorithm to choose optimal schedule is $O(n^2)$

Talking About Scheduling ...

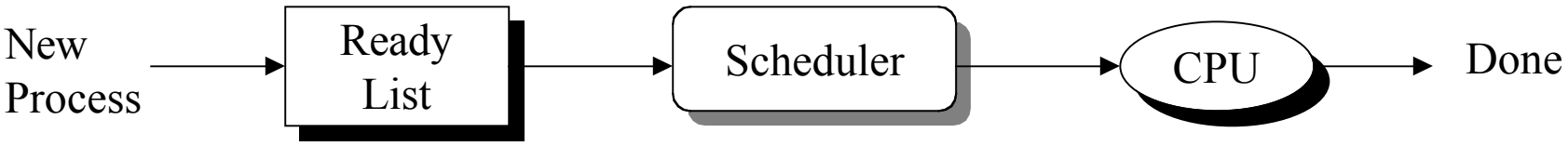
- Let $P = \{p_i \mid 0 \leq i < n\}$ = set of processes
- Let $S(p_i) \in$ {running, ready, blocked}
- Let $\tau(p_i)$ = Time process needs to be in running state (the *service time*)
- Let $W(p_i)$ = Time p_i is in ready state before first transition to running (*wait time*)
- Let $T_{TRnd}(p_i)$ =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

Simple Scheduling Model



- Ignore the resource manager
- Only consider running and ready states
- Ignores time in blocked state
  - "Process created when it enters ready"
  - "Process is destroyed when it enters blocked"
  - Really just looking at "small phases" of a process

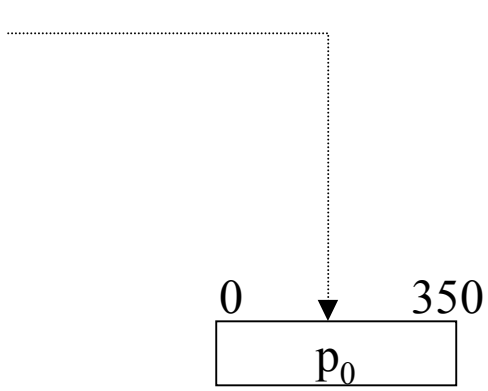
Nonpreemptive Schedulers



- Easy to build and analyze
  - Easy to analyze performance
  - No issue of voluntary/involuntary sharing

First-Come-First-Served

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

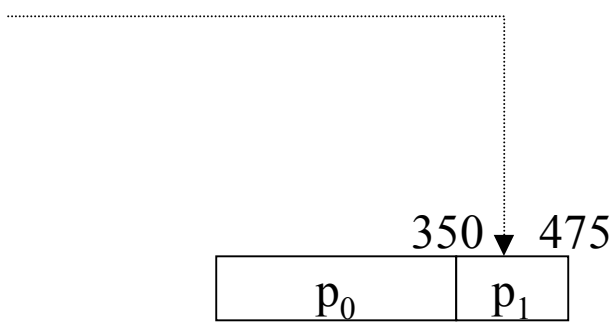


$T_{TRnd}(p_0) = \tau(p_0) = 350$

$W(p_0) = 0$

First-Come-First-Served

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



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

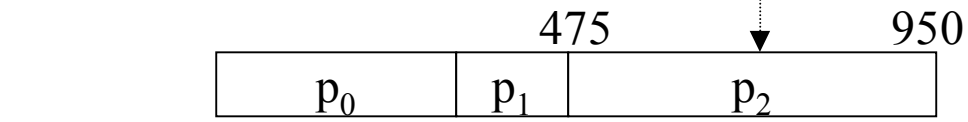
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$

$W(p_0) = 0$

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

First-Come-First-Served

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

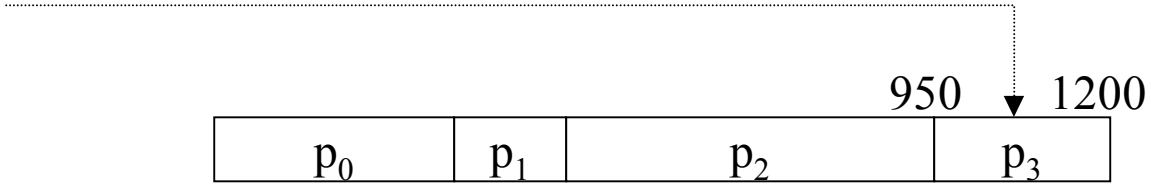


$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$

First-Come-First-Served

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

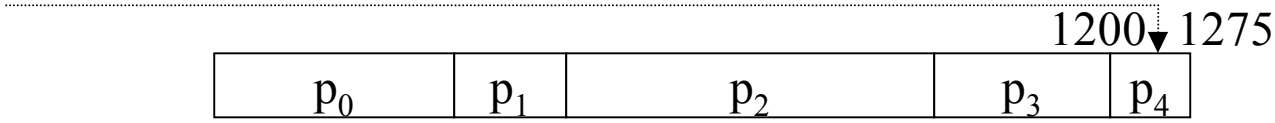


$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$
$W(p_3) = T_{TRnd}(p_2) = 950$

First-Come-First-Served

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



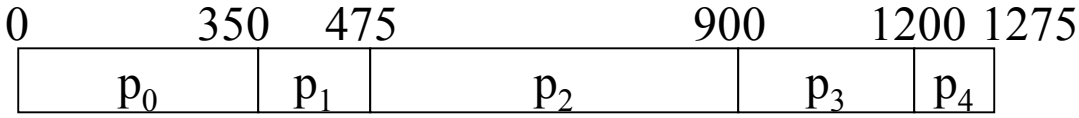
$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$
$T_{TRnd}(p_4) = (\tau(p_4) + T_{TRnd}(p_3)) = 75+1200 = 1275$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$
$W(p_3) = T_{TRnd}(p_2) = 950$
$W(p_4) = T_{TRnd}(p_3) = 1200$

FCFS Average Wait Time

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

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



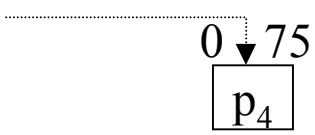
$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$
$T_{TRnd}(p_4) = (\tau(p_4) + T_{TRnd}(p_3)) = 75+1200 = 1275$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$
$W(p_3) = T_{TRnd}(p_2) = 950$
$W(p_4) = T_{TRnd}(p_3) = 1200$

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

Shortest Job Next

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

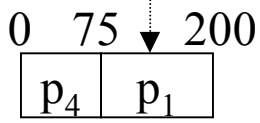


$T_{TRnd}(p_4) = \tau(p_4) = 75$

$W(p_4) = 0$

Shortest Job Next

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



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

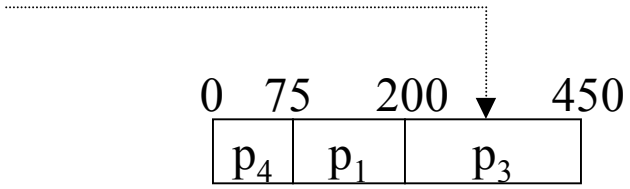
$W(p_1) = 75$

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

$W(p_4) = 0$

Shortest Job Next

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



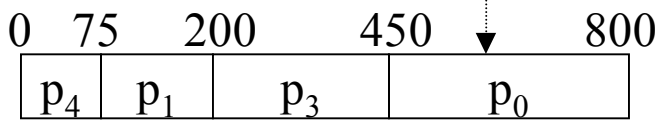
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $W(p_1) = 75$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Shortest Job Next

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



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $W(p_0) = 450$

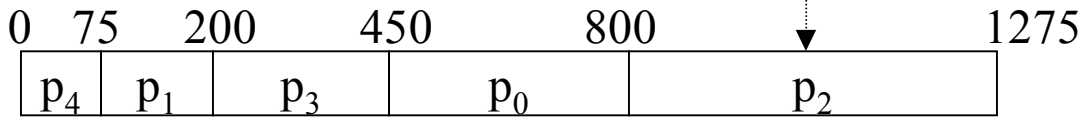
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $W(p_1) = 75$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Shortest Job Next

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



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $W(p_0) = 450$

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

$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 475+350+250+125+75 = 1275$ $W(p_2) = 800$

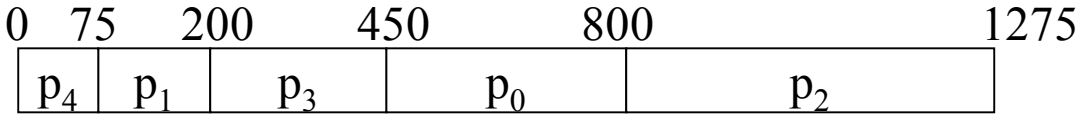
$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Shortest Job Next

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

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



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $W(p_0) = 450$

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

$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 475+350+250+125+75 = 1275$ $W(p_2) = 800$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

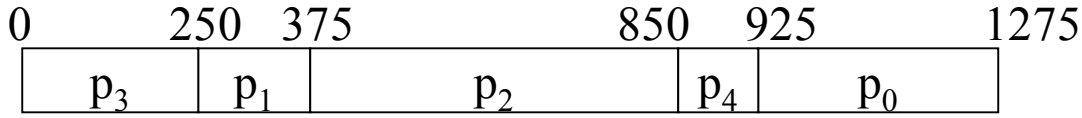
$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

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

Priority Scheduling

i	$\tau(p_i)$	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



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_4)+\tau(p_2)+\tau(p_1))+\tau(p_3) = 350+75+475+125+250$
$= 1275$
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_3) = 125+250 = 375$
$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_1)+\tau(p_3) = 475+125+250 = 850$
$T_{TRnd}(p_3) = \tau(p_3) = 250$
$T_{TRnd}(p_4) = \tau(p_4)+ \tau(p_2)+ \tau(p_1)+\tau(p_3) = 75+475+125+250 = 925$

$W(p_0) = 925$
$W(p_1) = 250$
$W(p_2) = 375$

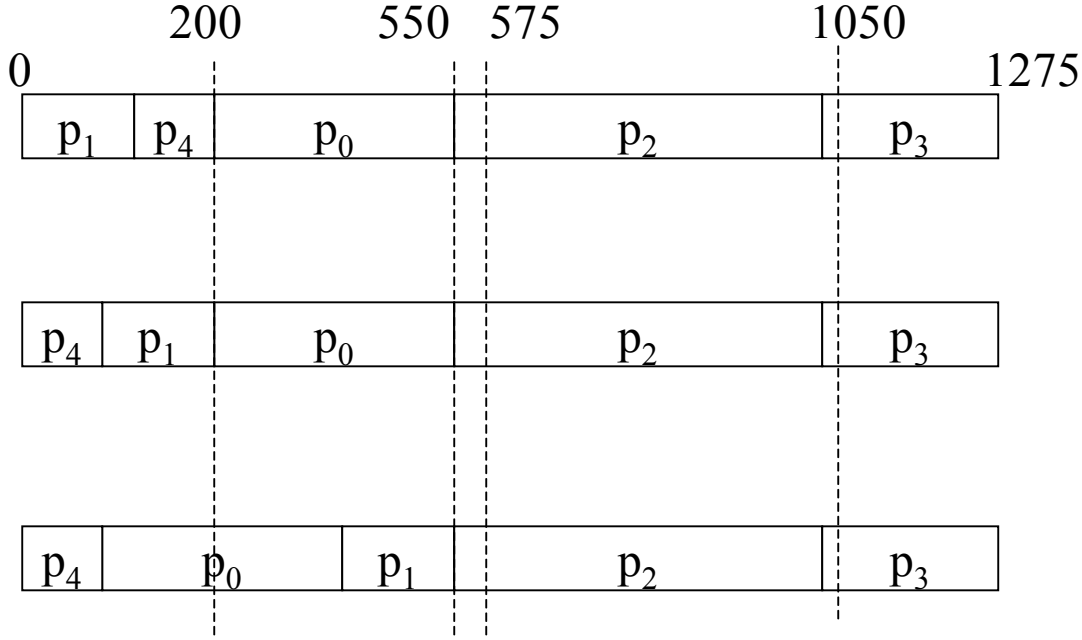
$W(p_3) = 0$
$W(p_4) = 850$

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

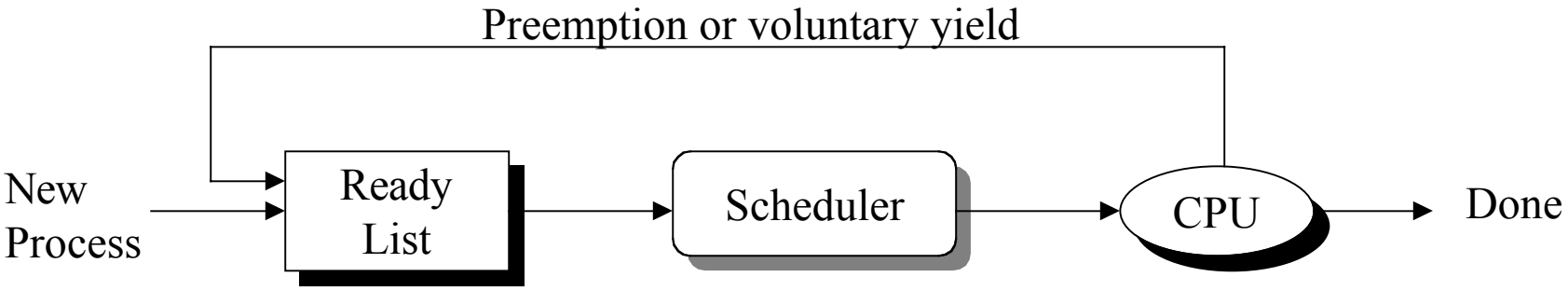
Deadline Scheduling

i	$\tau(p_i)$	Deadline
0	350	575
1	125	550
2	475	1050
3	250	(none)
4	75	200

- *Must* receive service by deadline
- May not be feasible



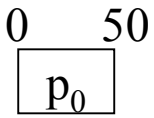
Preemptive Schedulers



- 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)

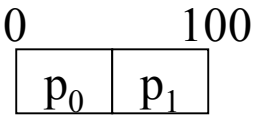
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$

Round Robin (TQ=50)

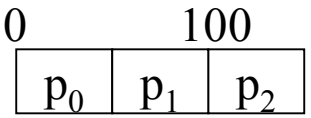
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$
$W(p_1) = 50$

Round Robin (TQ=50)

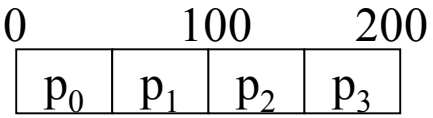
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$

Round Robin (TQ=50)

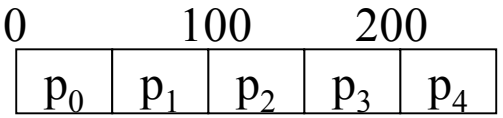
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$

Round Robin (TQ=50)

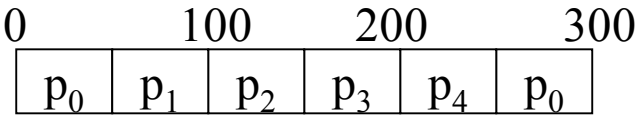
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

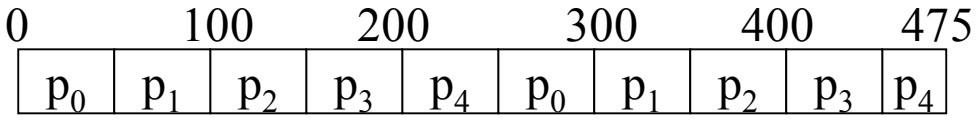
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

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

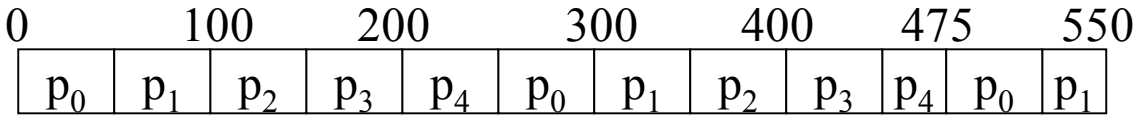


$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

$T_{TRnd}(p_4) = 475$

Round Robin (TQ=50)

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



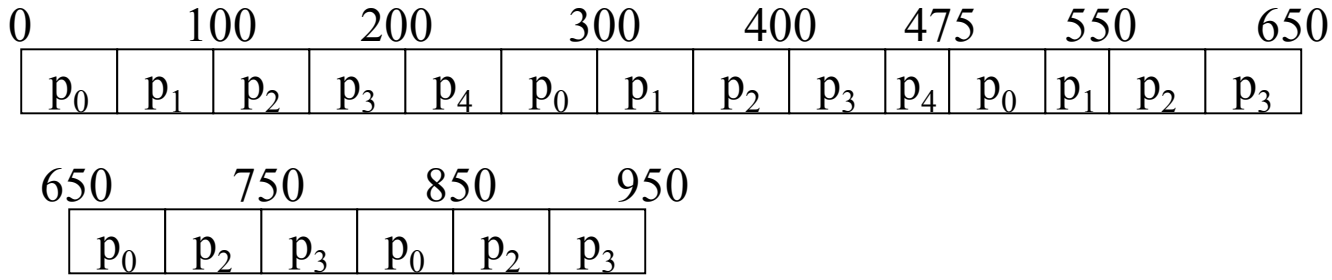
$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

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



$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_3) = 950$

$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$

$W(p_1) = 50$

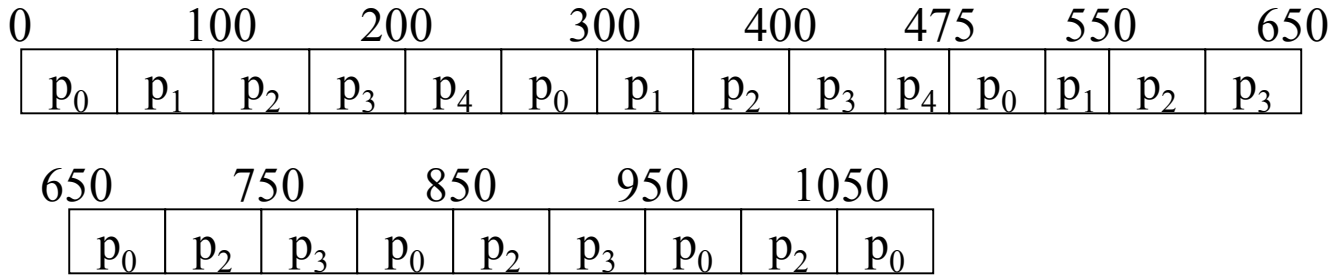
$W(p_2) = 100$

$W(p_3) = 150$

$W(p_4) = 200$

Round Robin (TQ=50)

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



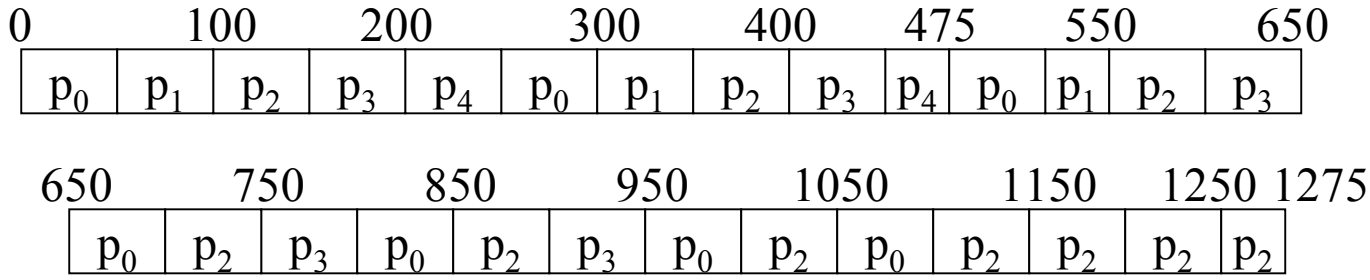
$T_{TRnd}(p_0) = 1100$
$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_3) = 950$
$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

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



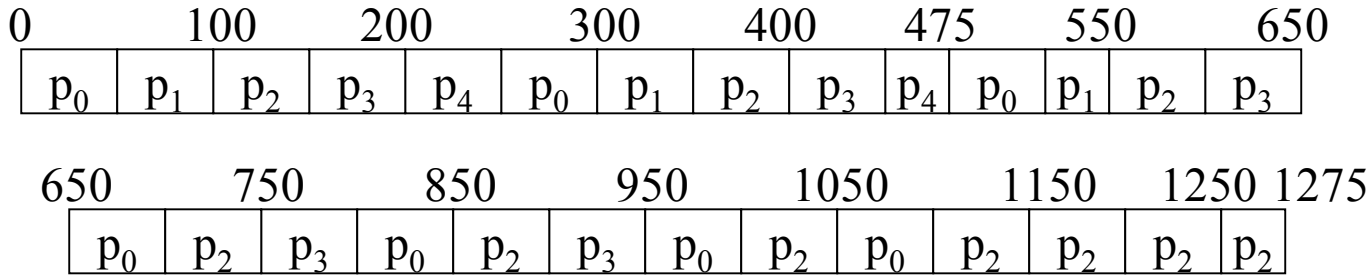
$T_{TRnd}(p_0) = 1100$
$T_{TRnd}(p_1) = 550$
$T_{TRnd}(p_2) = 1275$
$T_{TRnd}(p_3) = 950$
$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Round Robin (TQ=50)

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

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



$T_{TRnd}(p_0) = 1100$
$T_{TRnd}(p_1) = 550$
$T_{TRnd}(p_2) = 1275$
$T_{TRnd}(p_3) = 950$
$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

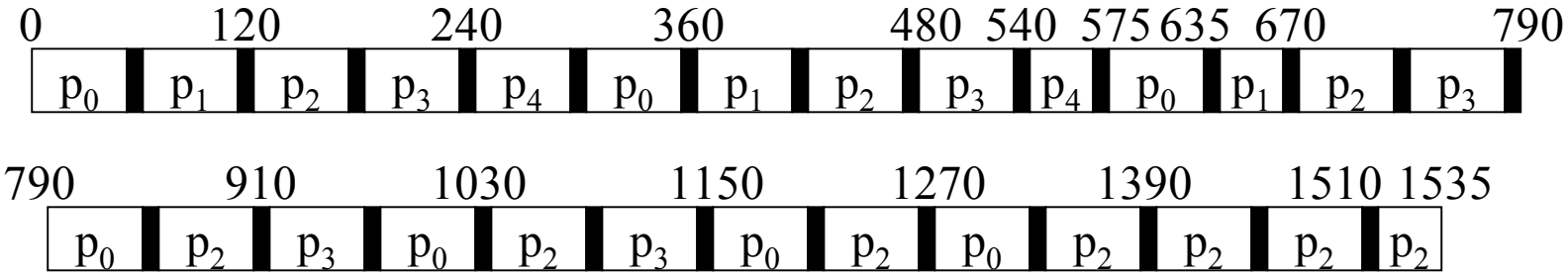
$T_{TRnd\_avg} = (1100+550+1275+950+475)/5 = 4350/5 = 870$

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

RR with Overhead=10 (TQ=50)

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

- Overhead must be considered



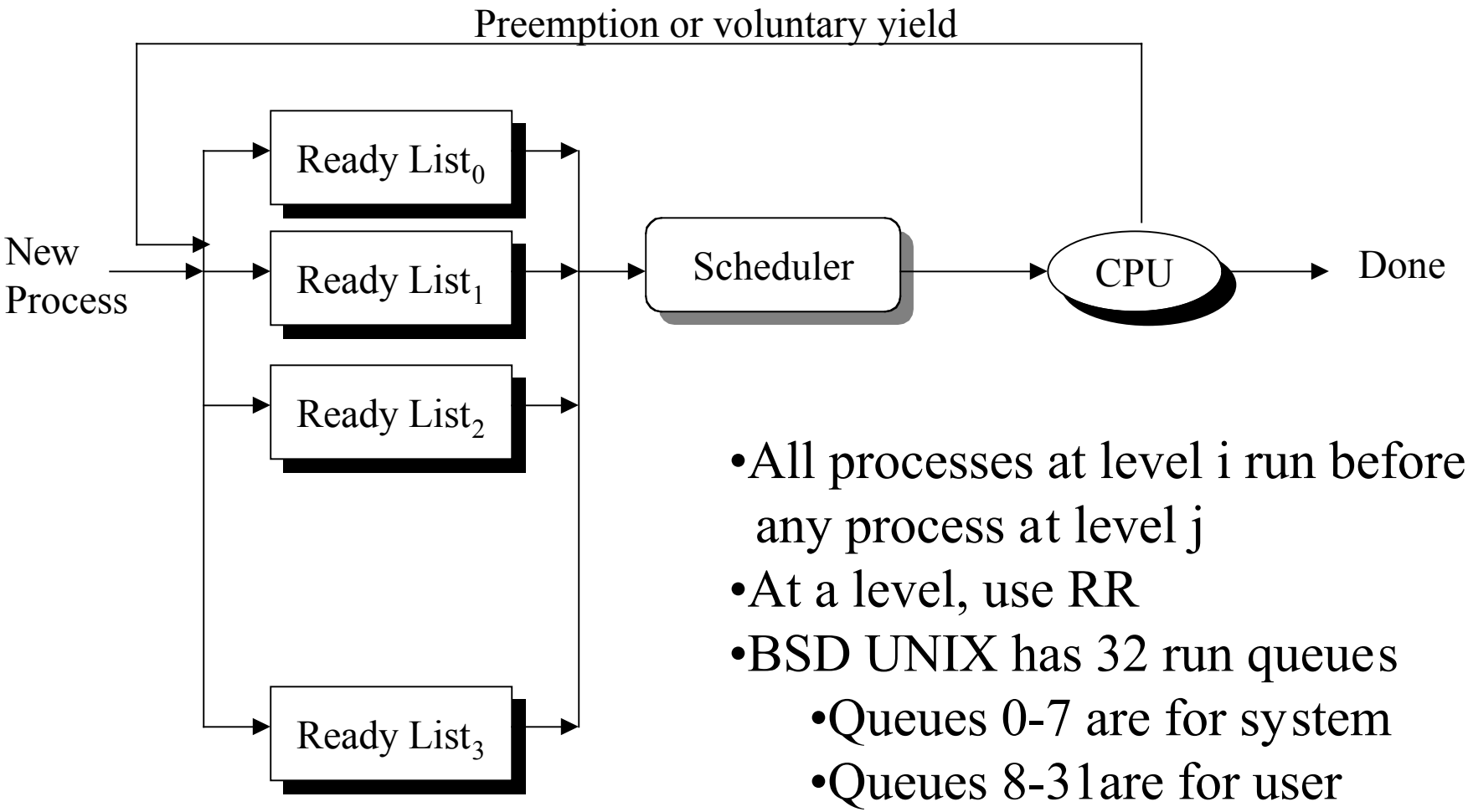
$T_{TRnd}(p_0) = 1320$
$T_{TRnd}(p_1) = 660$
$T_{TRnd}(p_2) = 1535$
$T_{TRnd}(p_3) = 1140$
$T_{TRnd}(p_4) = 565$

$W(p_0) = 0$
$W(p_1) = 60$
$W(p_2) = 120$
$W(p_3) = 180$
$W(p_4) = 240$

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

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

Multi-Level Queues



- All processes at level i run before any process at level j
- At a level, use RR
- BSD UNIX has 32 run queues
  - Queues 0-7 are for system
  - Queues 8-31are for user
  - `nice` influences run queue

Approximating Load

- Let λ = mean arrival rate
- So $1/\lambda$ = mean time between arrivals
- And μ = mean service rate
- So $1/\mu$ = mean service time (avg τ(pi))
- CPU busy = $\rho = \lambda * 1/\mu = \lambda/\mu$
- Notice must have $\lambda < \mu$ (i.e., $\rho < 1$)
- What if ρ approaches 1?

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
- $W_{avg}(p) = L*1/\mu + 0.5* 1/\mu = L/\mu+1/(2\mu)$
- Compare predicted wait with actual in earlier examples