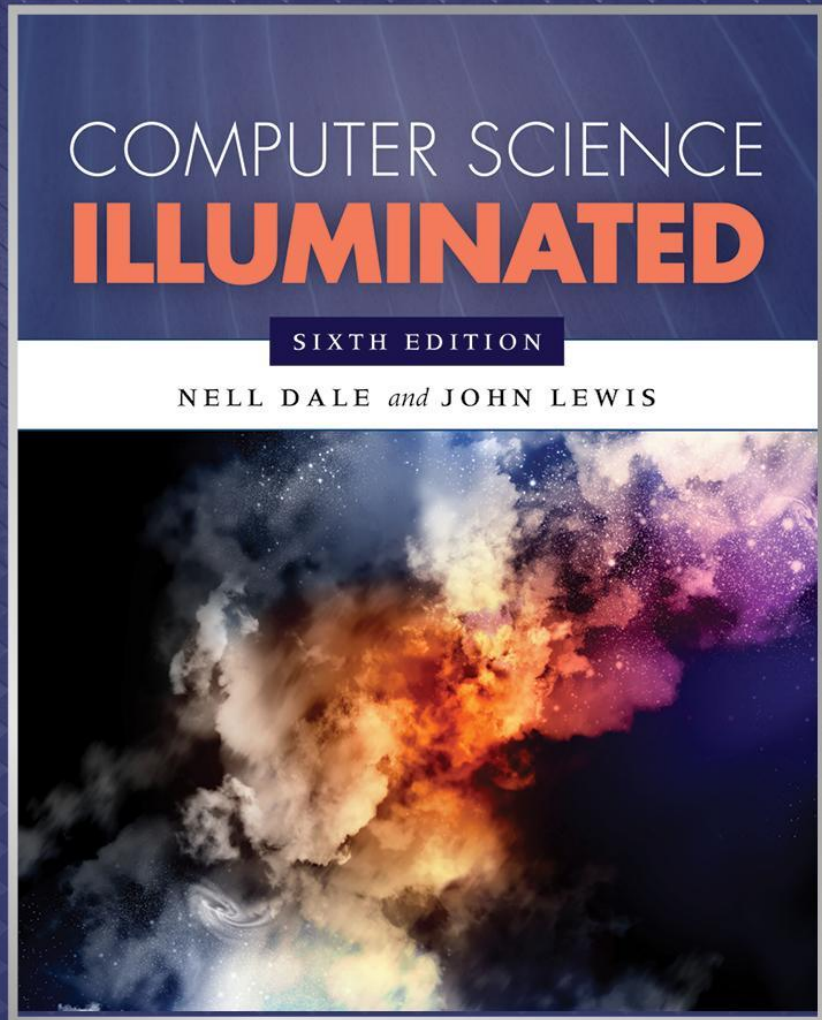


Operating Systems



Chapter Goals

- Describe the two main **responsibilities** of an operating system
- Define **memory** and **process** management
- Explain how **timesharing** creates the virtual machine illusion
- Explain the relationship between **logical** and **physical** addresses
- Compare and contrast **memory management techniques**

Chapter Goals

- Distinguish between **fixed** and **dynamic partitions**
- Define and apply partition **selection** algorithms
- Explain how **demand paging** creates the virtual memory illusion
- Explain the stages and transitions of the **process life cycle**
- Explain the processing of various CPU **scheduling** algorithms

Software Categories

Application software

Software written to address specific needs—to solve problems in the real world

System software

Software that manages a computer system at a fundamental level

Can you name examples of each?

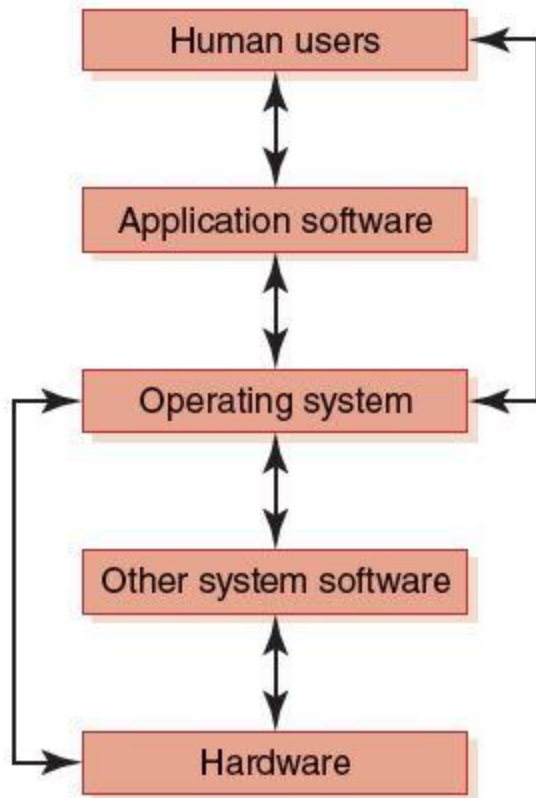
Roles of an Operating System

Operating system

System software that

- **manages** computer resources, such as memory and input/output devices
- **provides** an interface through which a human can interact with the computer
- **allows** an application program to interact with these other system resources

Roles of an Operating System



What operating systems have you used?

FIGURE 10.1 An operating system interacts with many aspects of a computer system.

Roles of an Operating System

The various roles of an operating system generally revolve around the idea of “sharing nicely”

An operating system manages resources, and these resources are often shared in one way or another among programs that want to use them

Services Provided by the Operating System I

- Program development
 - Editors and debuggers.
- Program execution
 - OS handles scheduling of numerous tasks required to execute a program
- System access
 - Controls access to the system and its resources

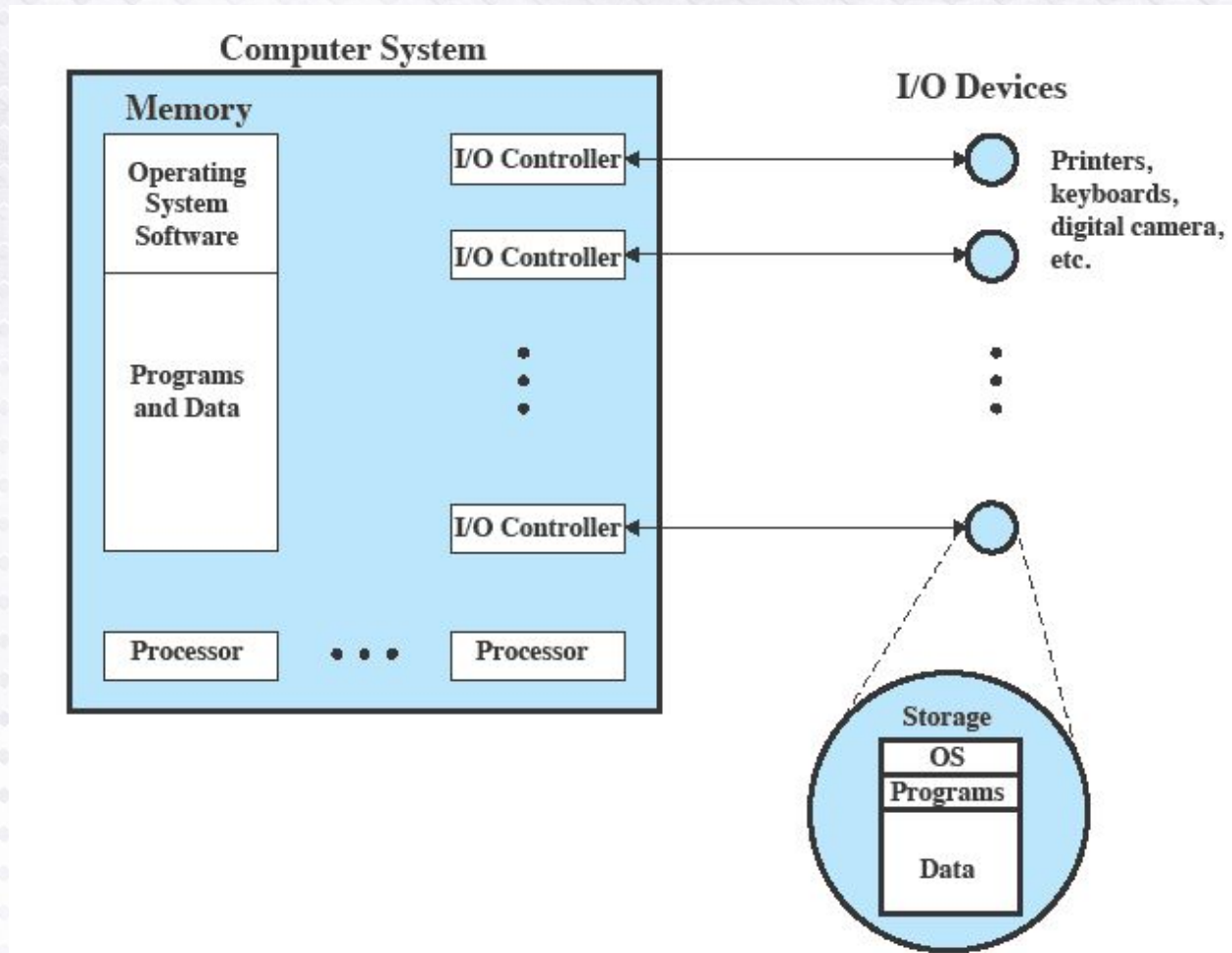
Services Provided by the Operating System II

- Controlled access to files
 - Accessing different media but presenting a common interface to users
 - Provides protection in multi-access systems
- Access I/O devices
 - Each device will have unique interface
 - OS presents standard interface to users
 - Device drivers – A small program that “knows” the way a particular device expects to receive and deliver information

Services Provided by the Operating System III

- Error detection and response
 - Internal and external hardware errors
 - Software errors
 - Operating system cannot grant request of application
- Accounting
 - Collect usage statistics
 - Monitor performance

OS as a Resource Manager



Resource Management

Multiprogramming

The technique of keeping multiple programs that compete for access to the CPU in main memory at the same time so that they can execute

Memory management

The process of keeping track of what programs are in memory and where in memory they reside

Resource Management

Process

A program in execution

Process management

The act of carefully tracking the progress of a process and all of its intermediate states

CPU scheduling

Determining which process in memory is executed by the CPU at any given point

Evolution of Operating Systems

Serial Processing

Simple Batch Systems

Multiprogrammed Batch Systems

Time Sharing Systems

Evolution of Operating Systems

Serial Processing

- No operating system
- Machines run from a console with display lights, toggle switches, input device, and printer
- Problems include:
 - Scheduling
 - Setup time

Batch Processing

The first operating system was a **human operator**, who organized various jobs from multiple users into *batches* of jobs that needed the same resources

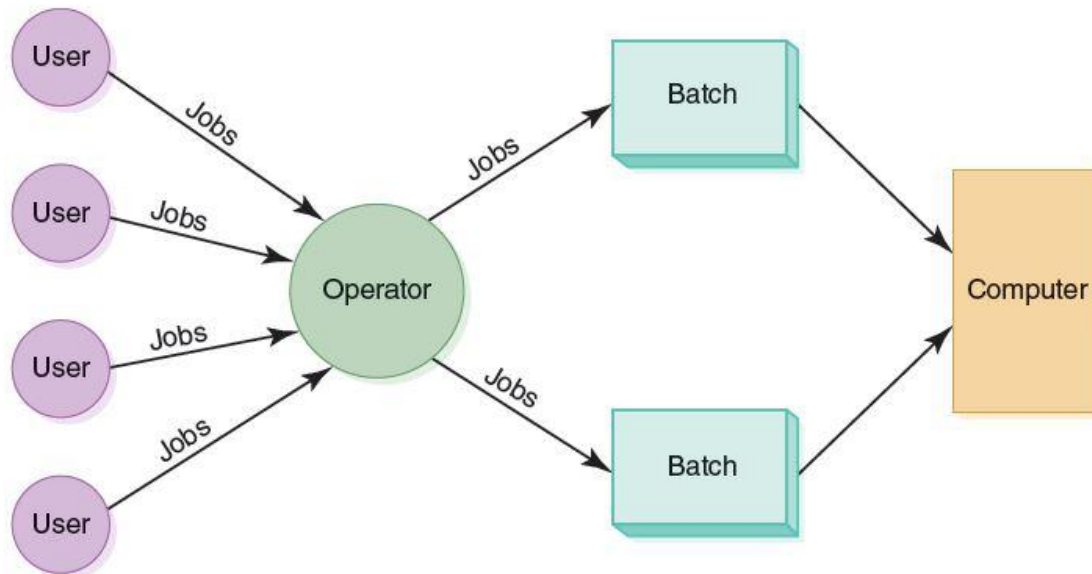
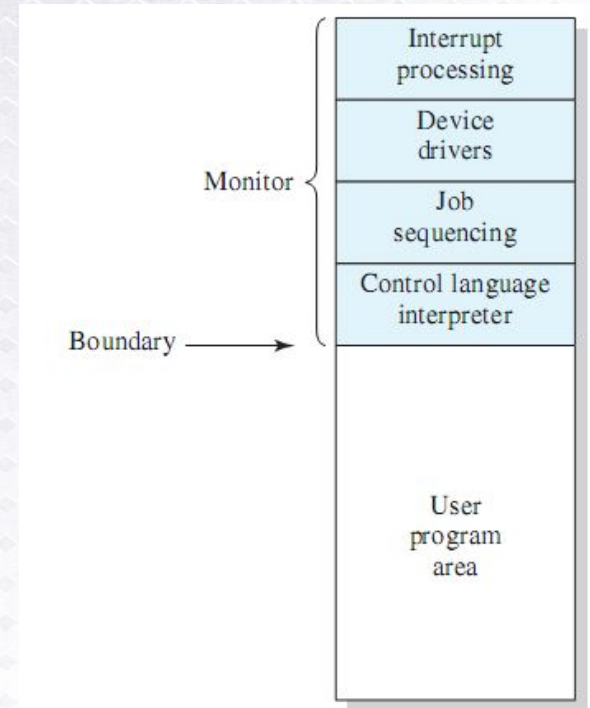


FIGURE 10.3 In early systems, human operators would organize jobs into batches

Simple Batch OS

Monitor

- Software that controls the sequence of events
- Batches jobs together
- Resident Monitor is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor



Job Control Language

Special type of programming language to control jobs

Provides instructions to the monitor

- What compiler to use
- What data to use

Desirable Hardware Features

Memory Protection for monitor

- Jobs cannot overwrite or alter monitor

Timer

- Prevents a job monopolizing the system

Privileged Instructions

- Only executed by Monitor

Interrupts

Modes of Operation

User Mode

- User program executes in user mode
- Certain area of memory protected from user access
- Certain instructions may not be executed

Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed, all memory is accessible.

Batch Processing

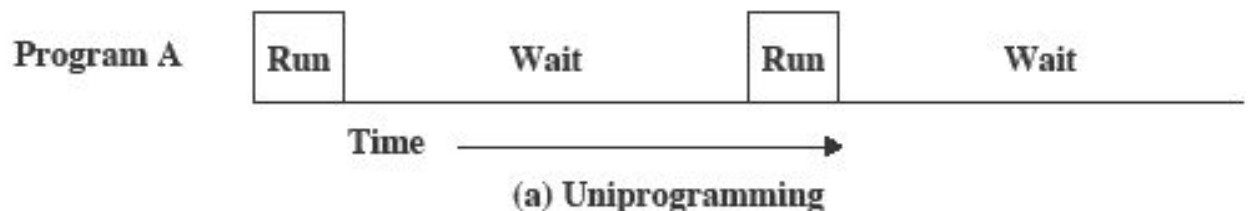
Even in Batch processing, CPU is often idle.

- I/O devices are slow in comparison

Read one record from file	15 μ s
Execute 100 instructions	1 μ s
Write one record to file	15 μ s
TOTAL	31 μ s

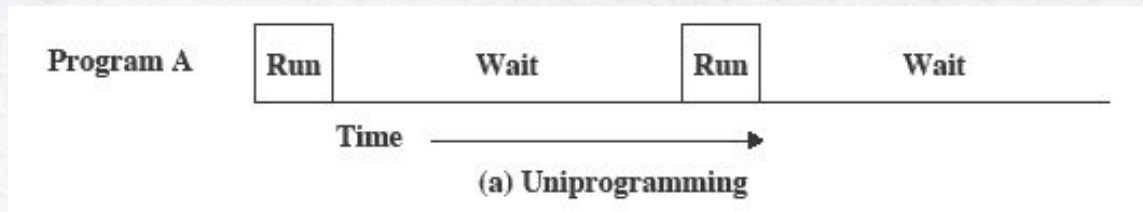
$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Uniprogramming - Processor must wait for I/O instruction to complete before proceeding

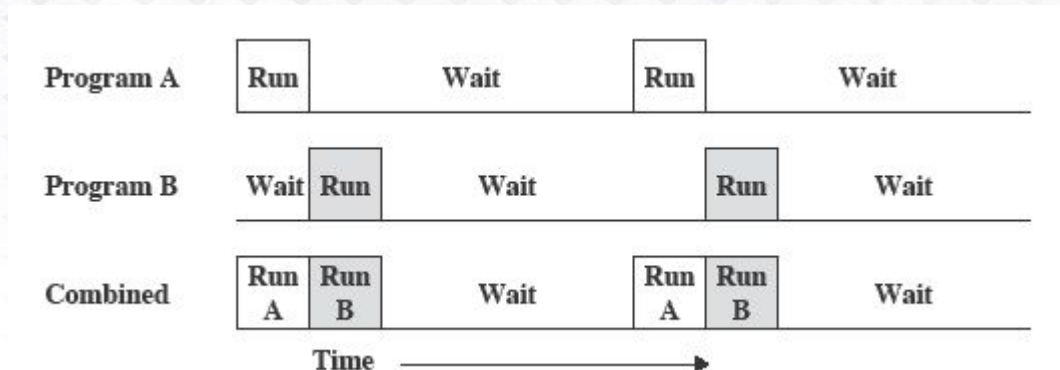


Batch Processing

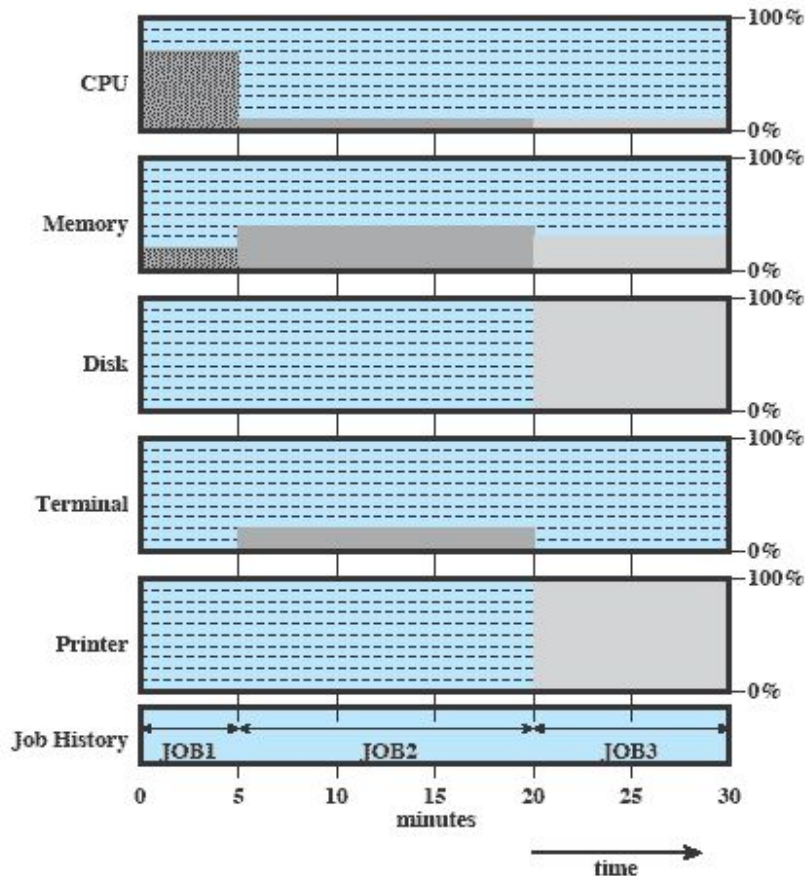
Uniprogramming - Processor must wait for I/O instruction to complete before proceeding



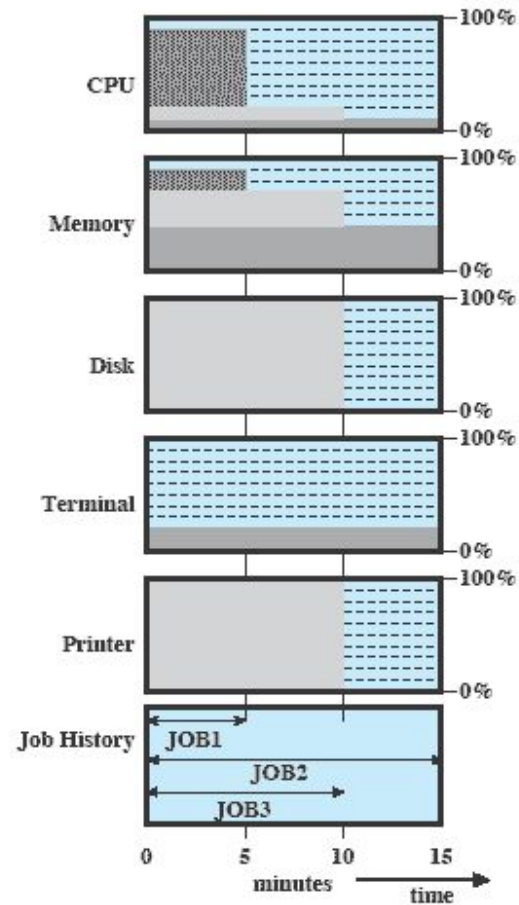
Multiprogramming – When one job needs to wait for I/O, the processor can switch to another job



Utilization Histograms



(a) Uniprogramming



(b) Multiprogramming

Batch Multiprogramming

Maximises Processor use

Commands via Job Control Language

Allows multiple jobs to execute in one time period

Batch Multiprogramming

Allows multiple users to interact with computer at same time

- Minimise response time
- Time delay between receiving **stimulus** and producing a **response**
- Commands entered at terminal

Timesharing

Timesharing system

A system that allows multiple users to interact with a computer at the same time

Virtual machine

The illusion created by a time-sharing system that each user has his/her own machine

As computer speed increased, the human operator became the bottleneck

Other Factors

Real-time System

A system in which response time is crucial given the nature of the application

Response time

The time delay between receiving a stimulus and producing a response

Device driver

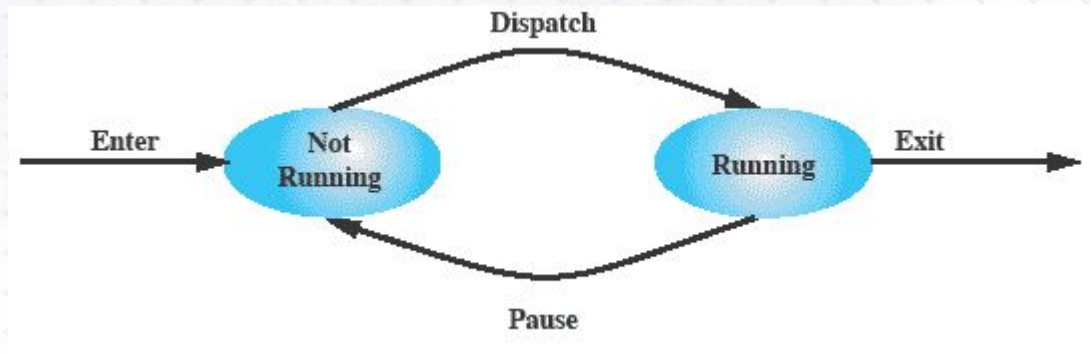
A small program that “knows” the way a particular device expects to receive and deliver information

Process Management

The act of managing the use of the CPU by individual processes

In the simplest model, a process may be in one of two states:

- Running
- Not Running



Process Management

The Process States

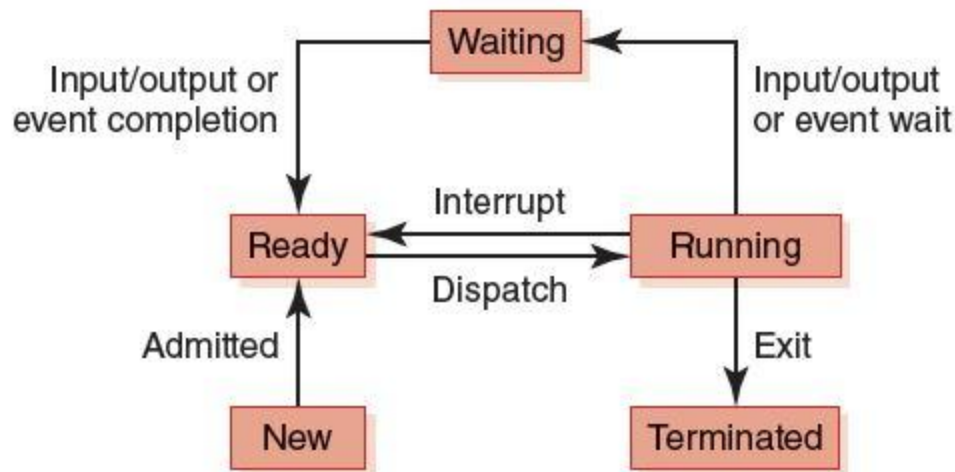


FIGURE 10.9 The process life cycle

What can cause a process to move to the Waiting state?

Process Creation

Operating System builds a data structure to manage the process

Traditionally, the OS created all processes:

- New batch job
- Interactive Login
- Created to provide a service

But it can be useful to let a running process create another..

Process Spawning

The creation of processes.

Parent Process - The original, creating, process

Child Process - The newly created process

Process Termination

Various reasons to terminate a process:

- New batch job
 - HALT instruction generates interrupt alert to OS
- Interactive Login
 - User logs off
- Created to provide a service
 - Service / Application completed
- Fault / Error
- Parent Process terminating can terminate children
- Operating System or User interrupt

Process Suspension

Simple batch processes may run to completion, but to improve CPU utilisation and interactive response times we may need to suspend processes.

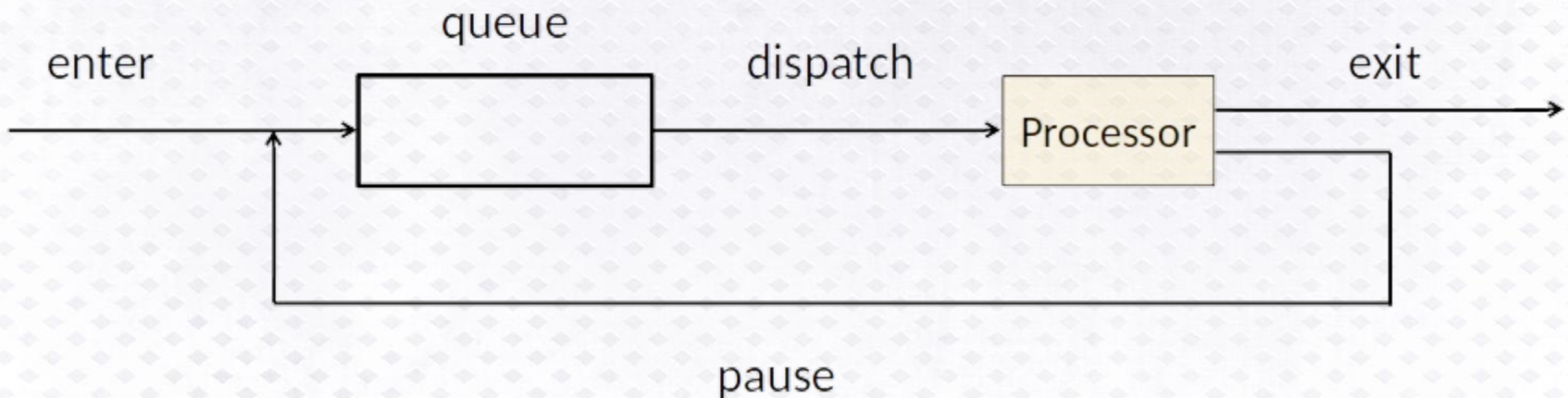
Process Suspension

Processes may be suspended because:

- Process needs an unavailable resource
- OS suspects the process of causing a problem
- Interactive user wishes to pause process (debugging etc)
- Process may only need to run periodically (eg mail system)
- Parent process wishes to suspend it
- OS needs resources for another process

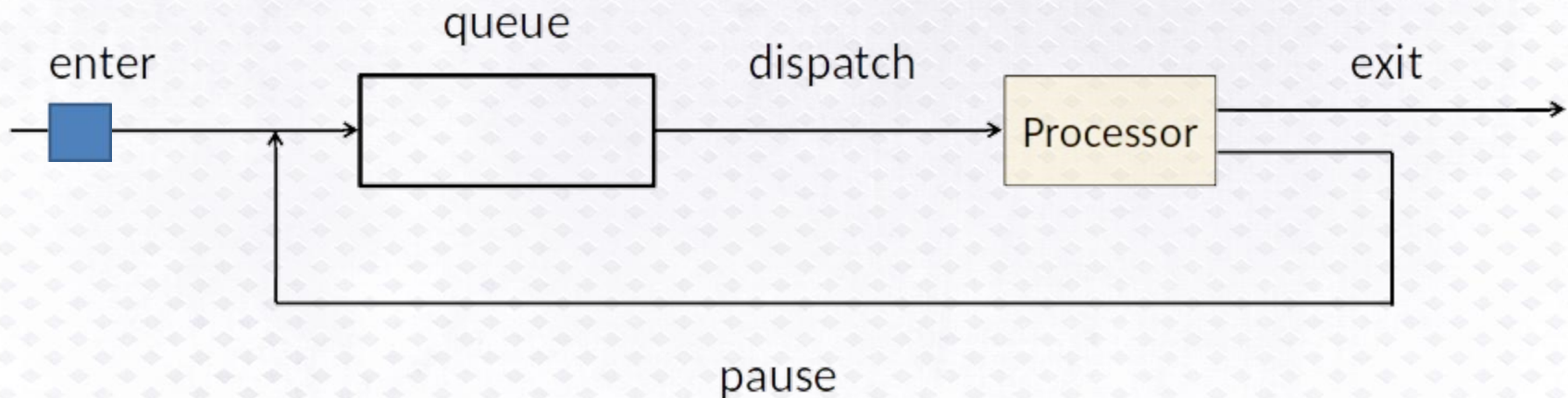
Process Management

To manage the process, we could use a simple queue:



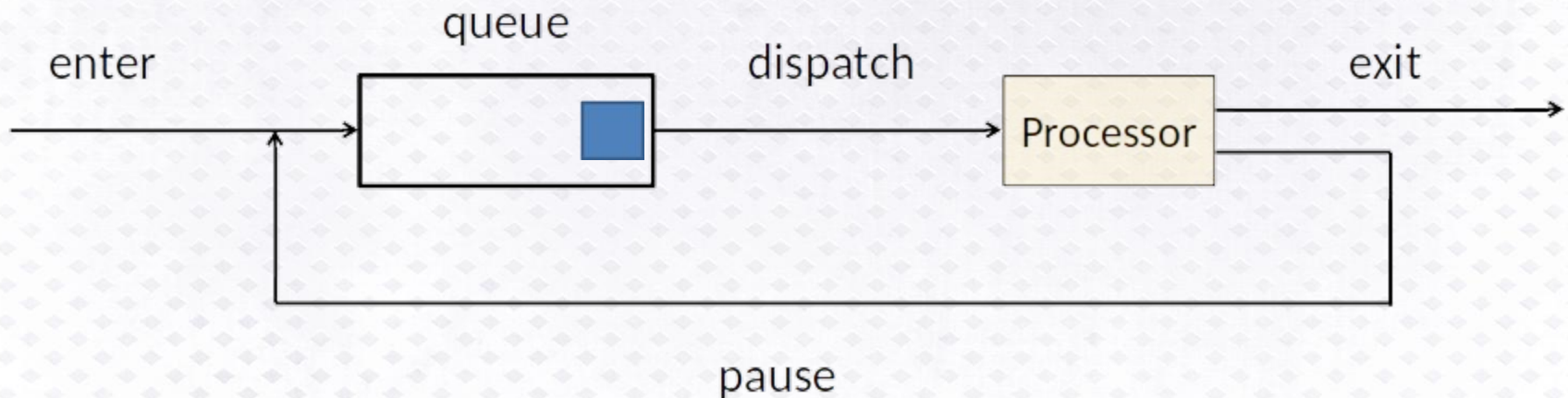
Process Management

To manage the process, we could use a simple queue:



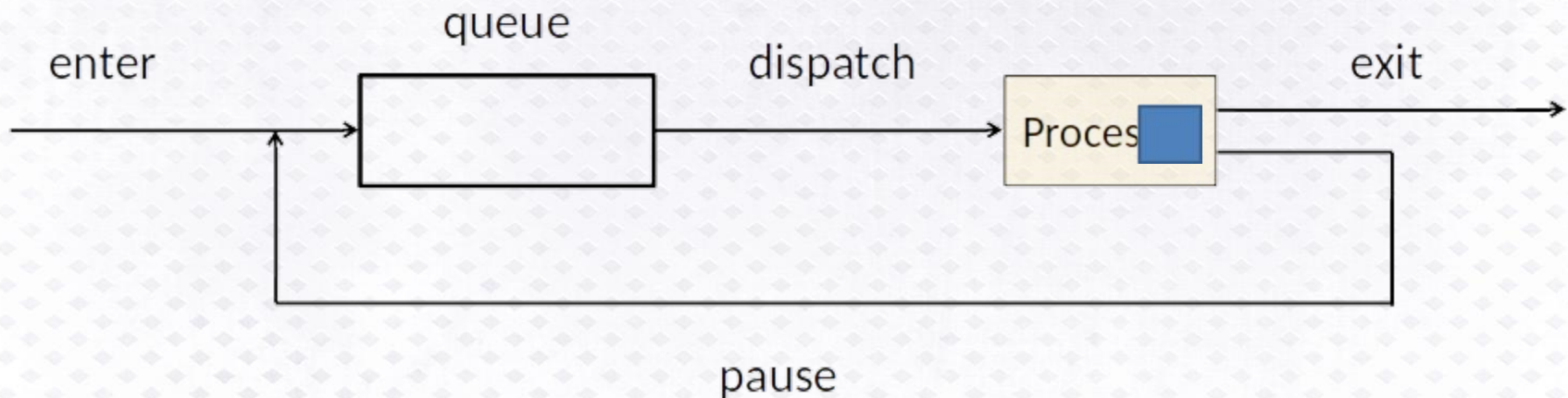
Process Management

To manage the process, we could use a simple queue:



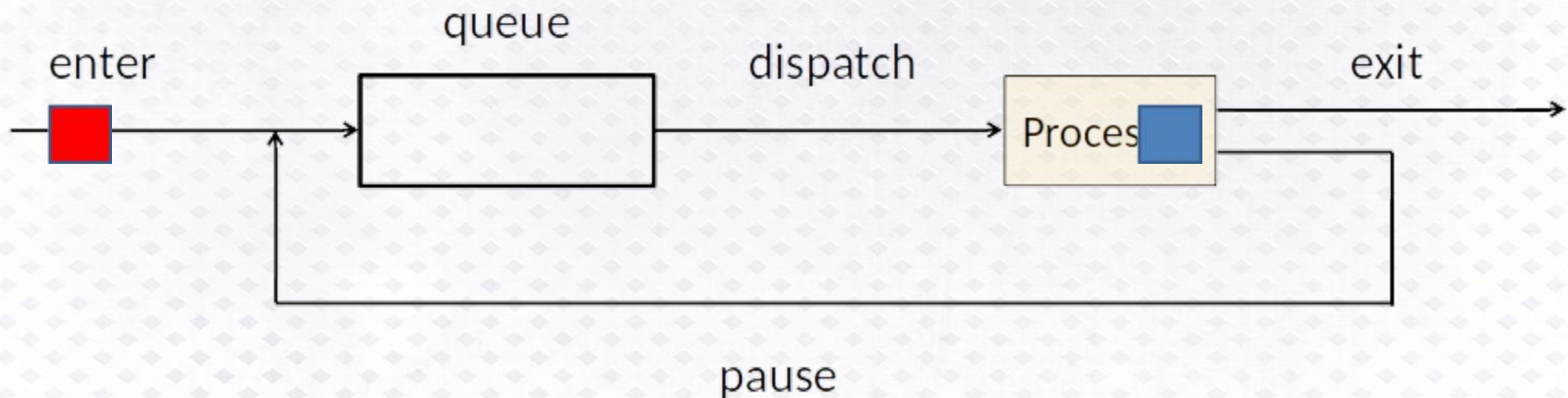
Process Management

To manage the process, we could use a simple queue:



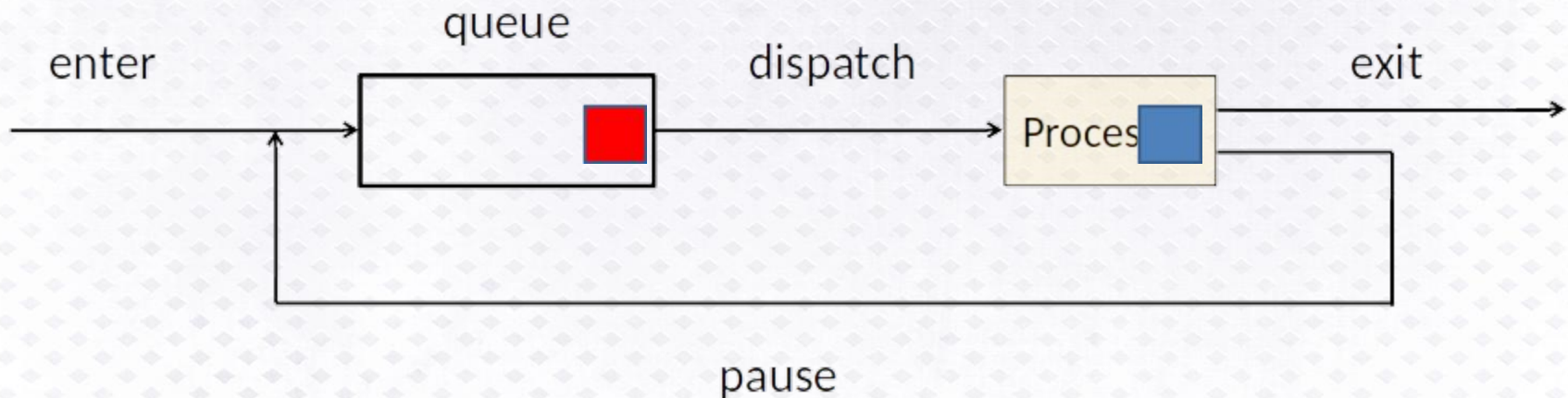
Process Management

To manage the process, we could use a simple queue:



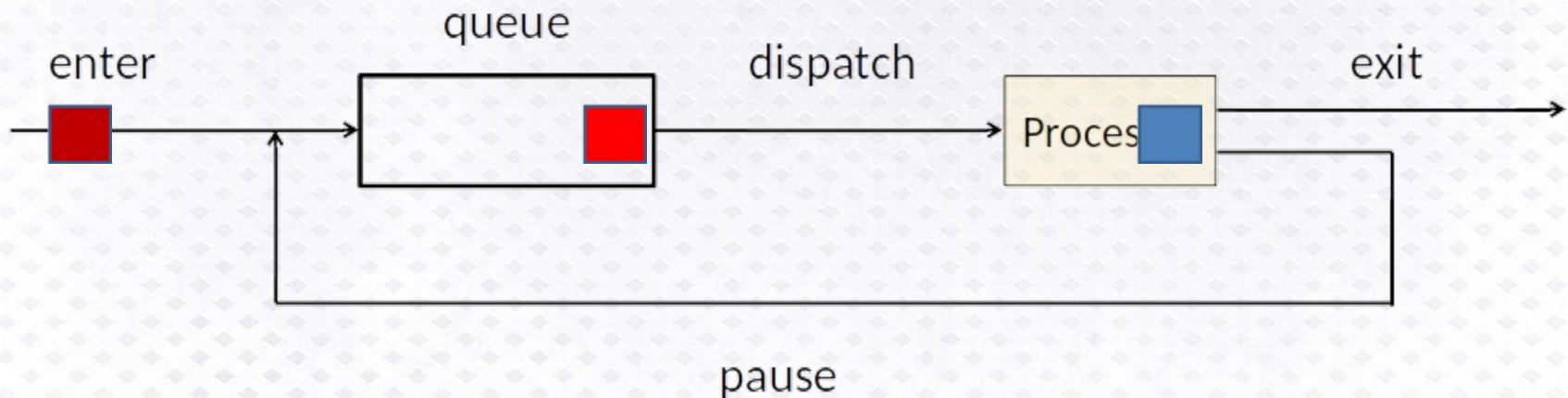
Process Management

To manage the process, we could use a simple queue:



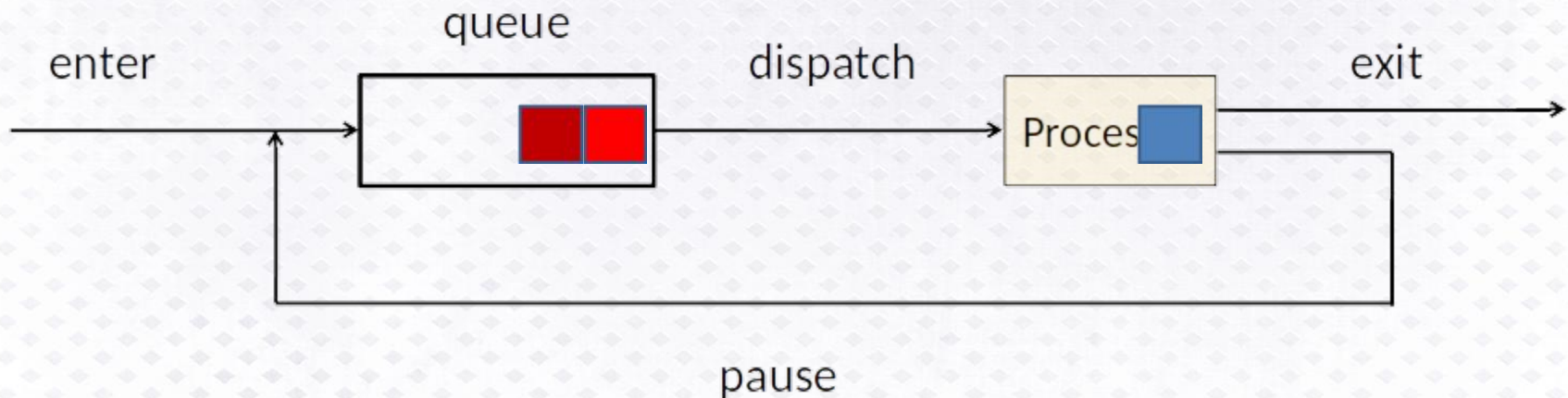
Process Management

To manage the process, we could use a simple queue:



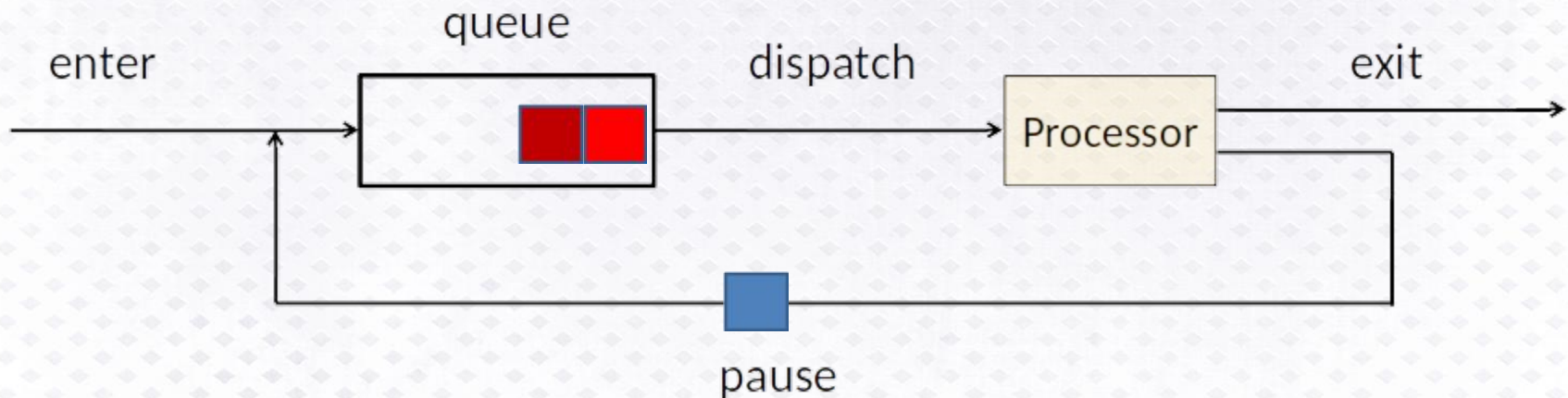
Process Management

To manage the process, we could use a simple queue:



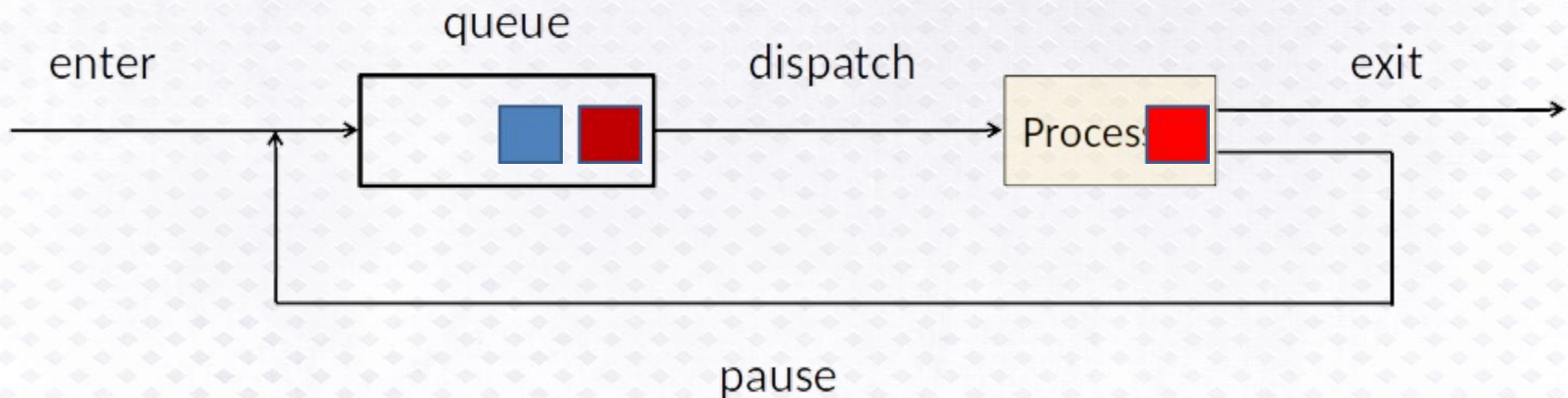
Process Management

To manage the process, we could use a simple queue:



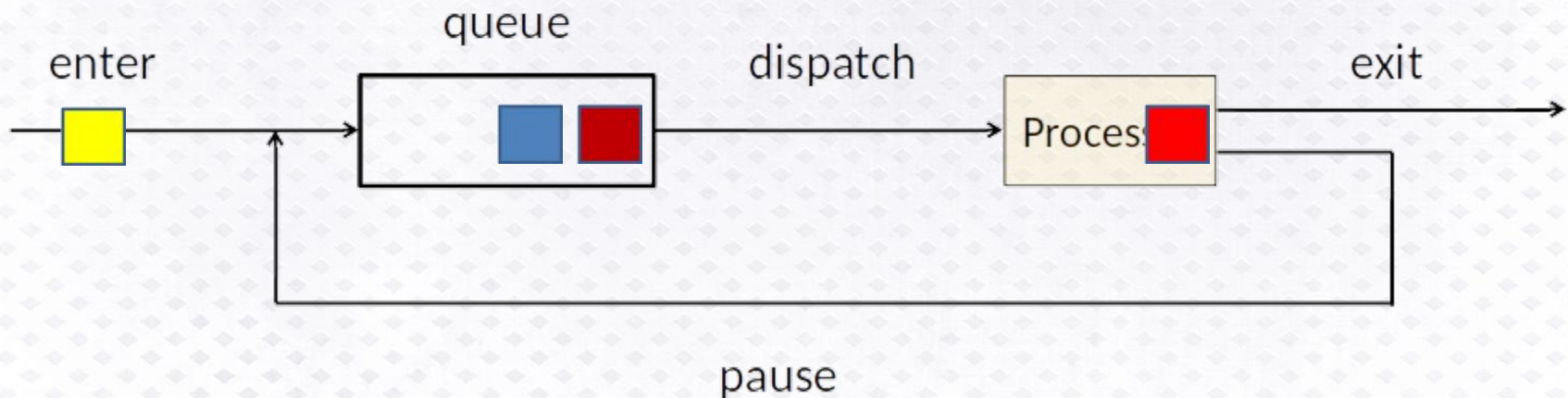
Process Management

To manage the process, we could use a simple queue:



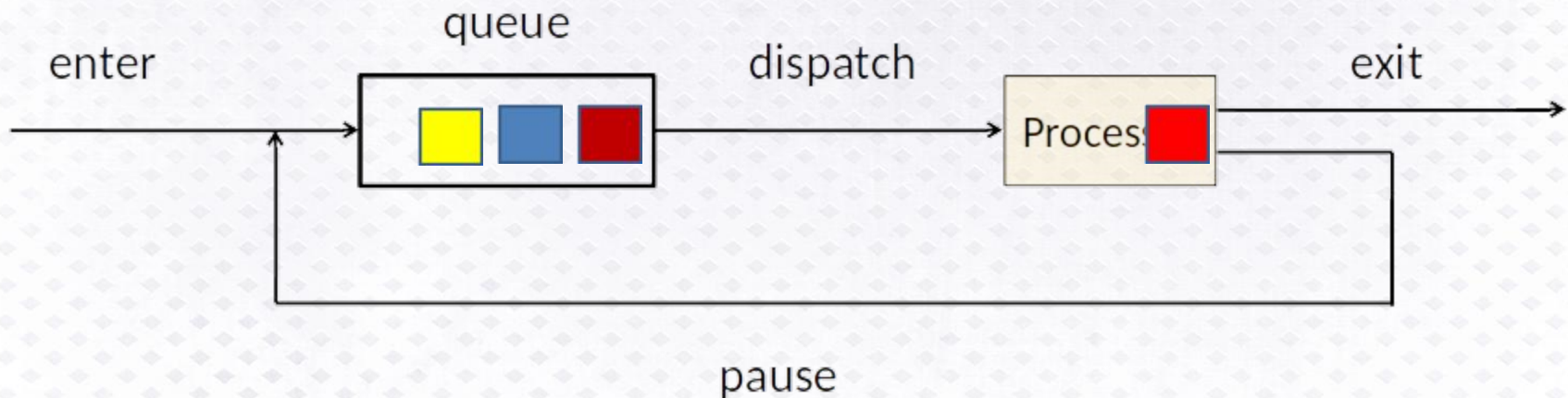
Process Management

To manage the process, we could use a simple queue:



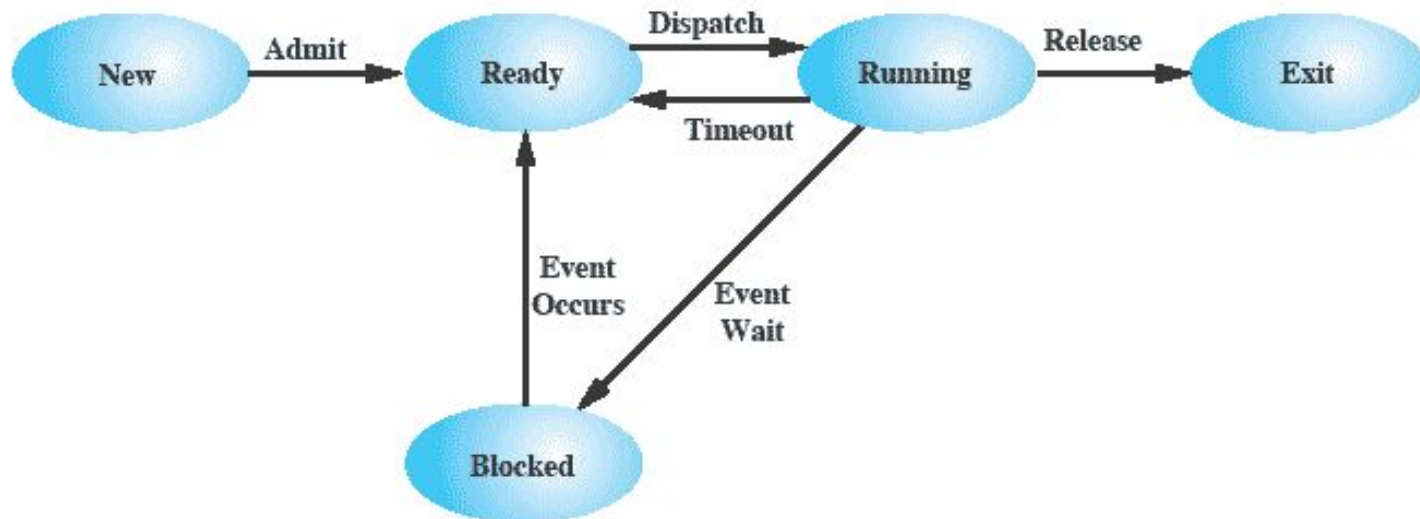
Process Management

To manage the process, we could use a simple queue:



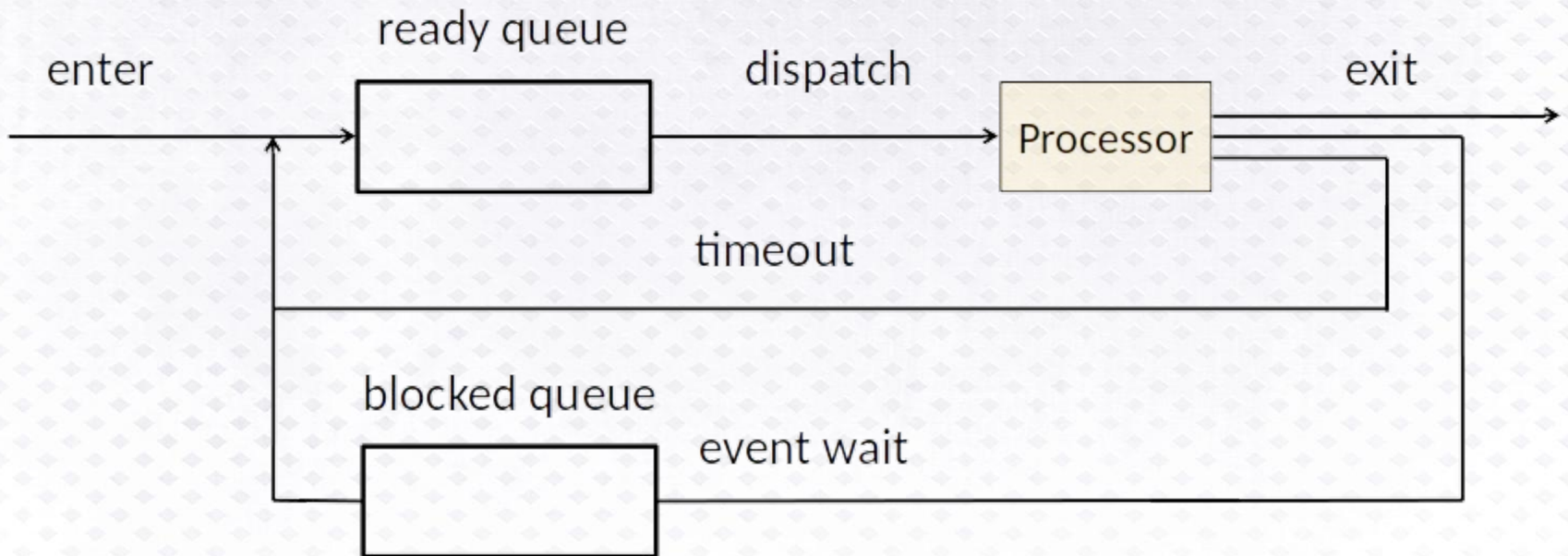
Process Management

To reflect the different states of non-running processes, we can extend our simple state model:



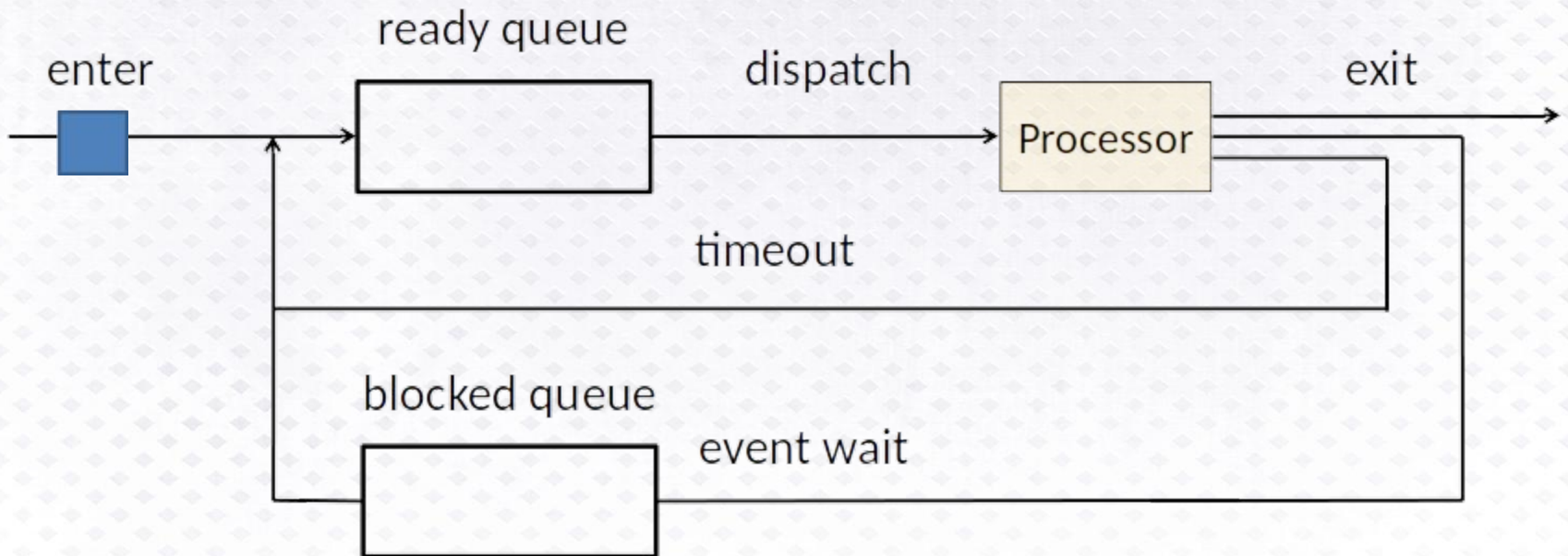
Process Management

To reflect the two pause states, use two queues



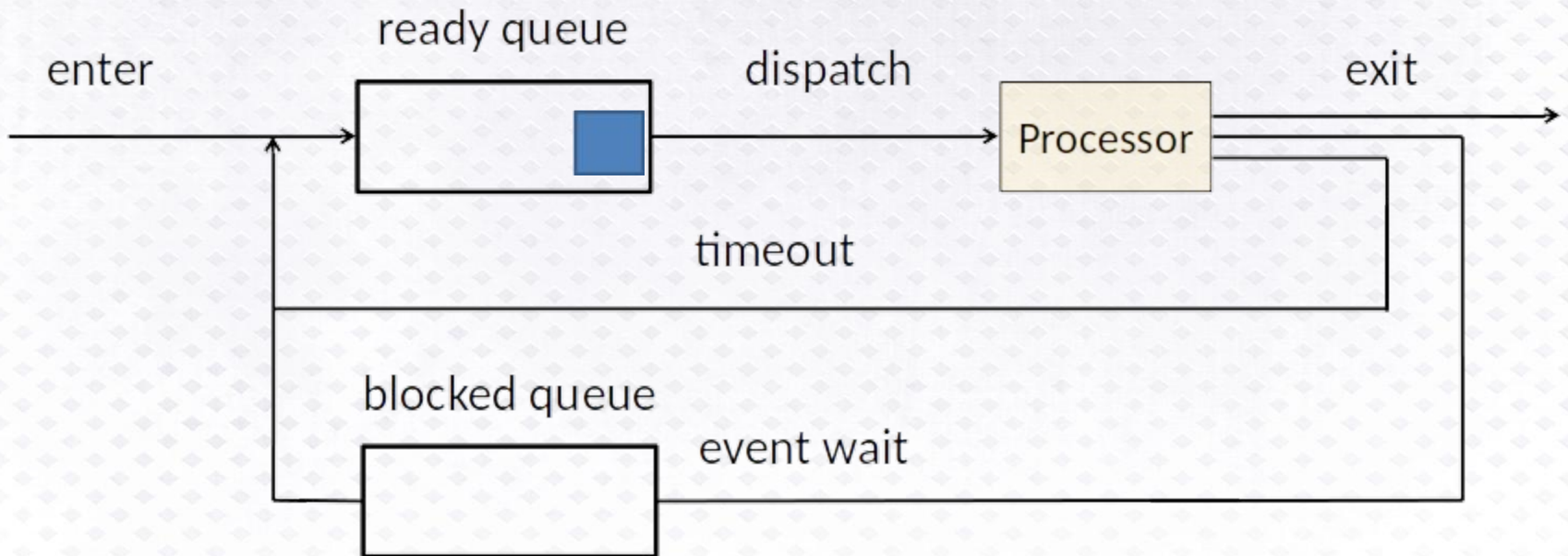
Process Management

To reflect the two pause states, use two queues



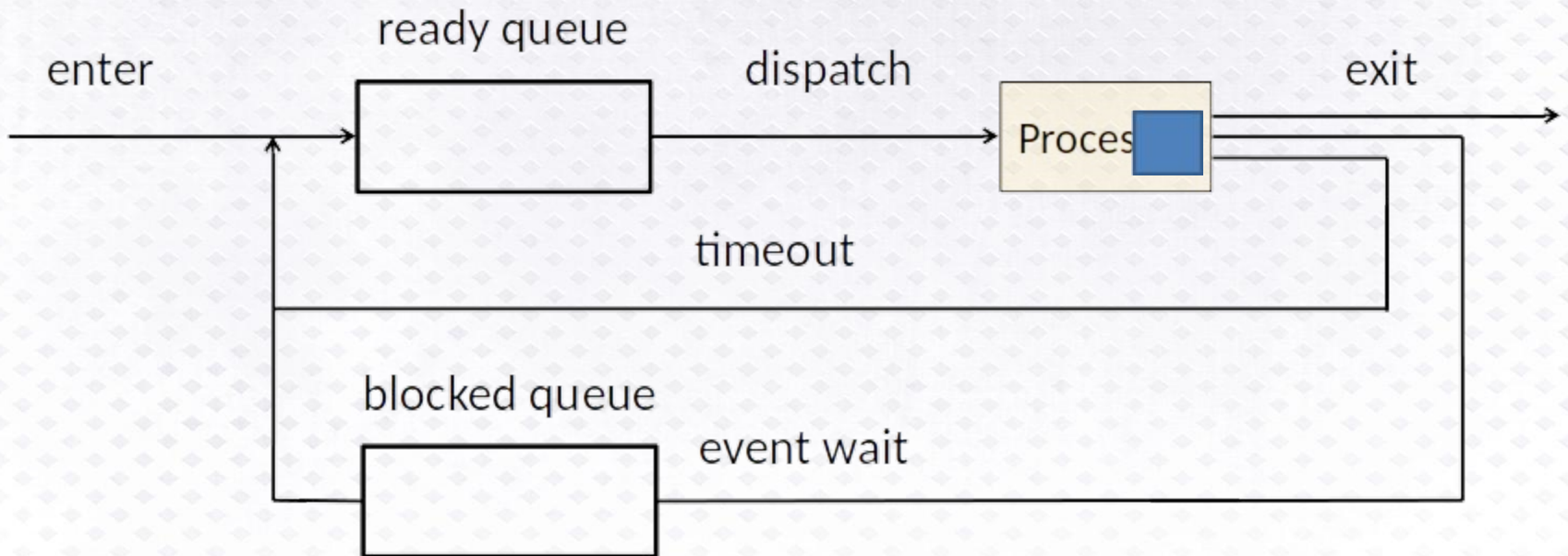
Process Management

To reflect the two pause states, use two queues



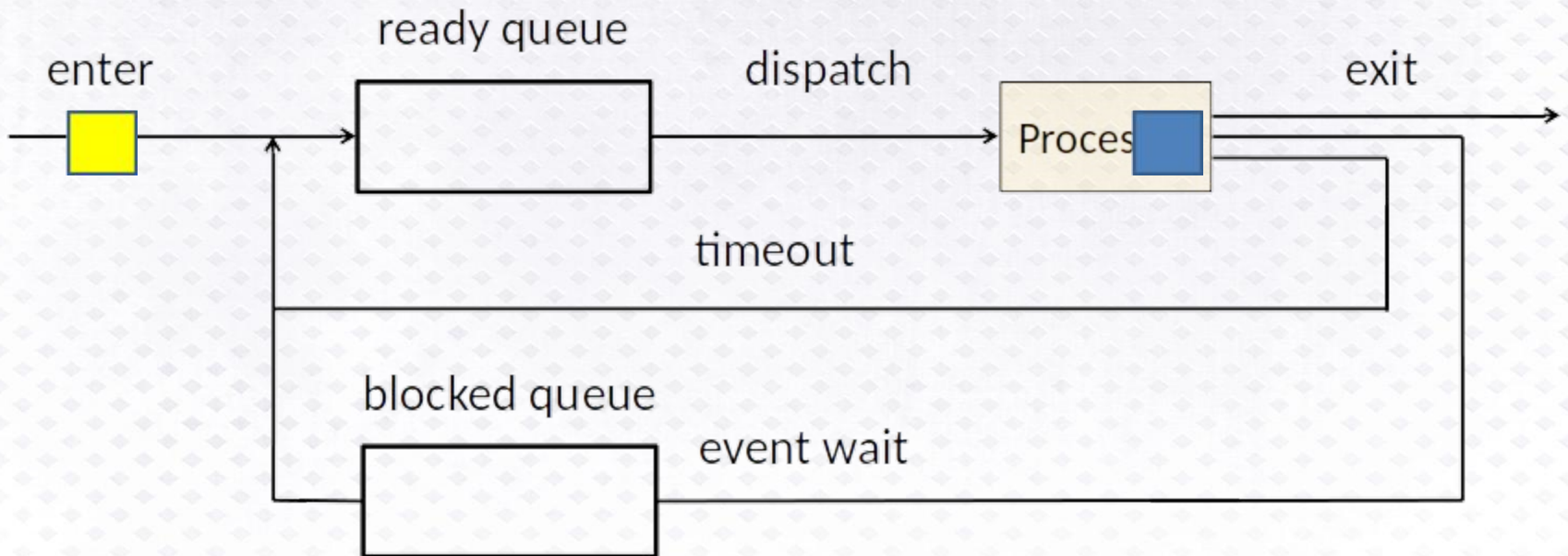
Process Management

To reflect the two pause states, use two queues



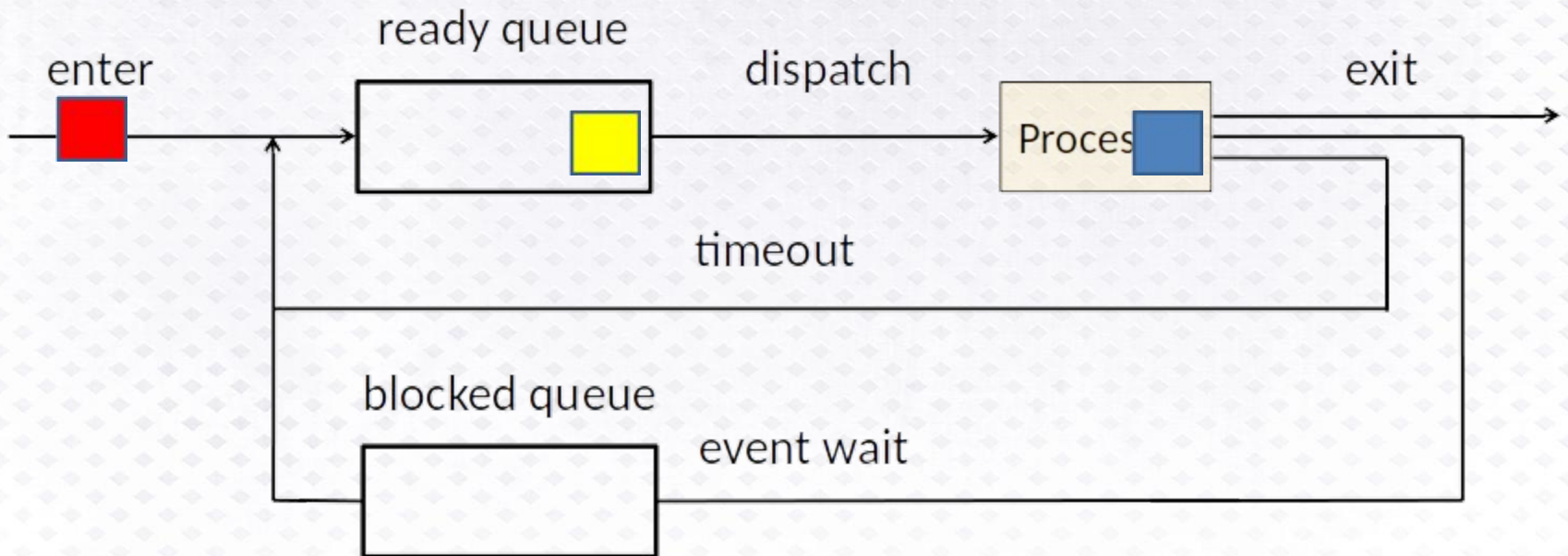
Process Management

To reflect the two pause states, use two queues



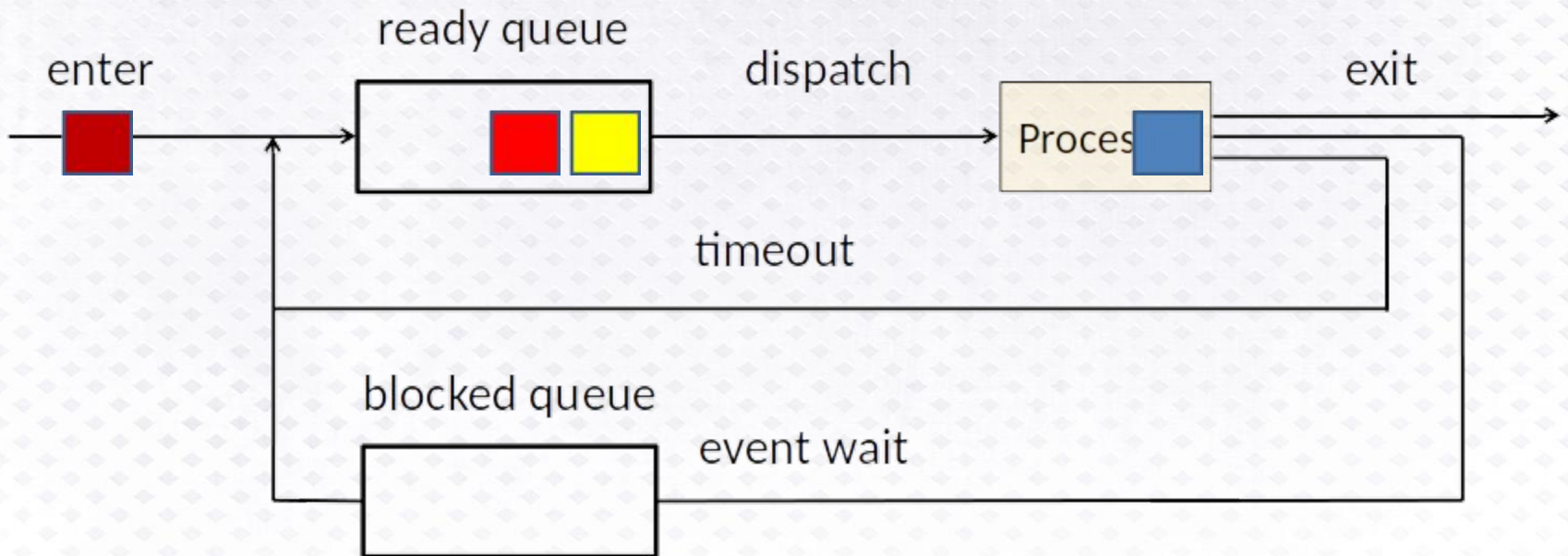
Process Management

To reflect the two pause states, use two queues



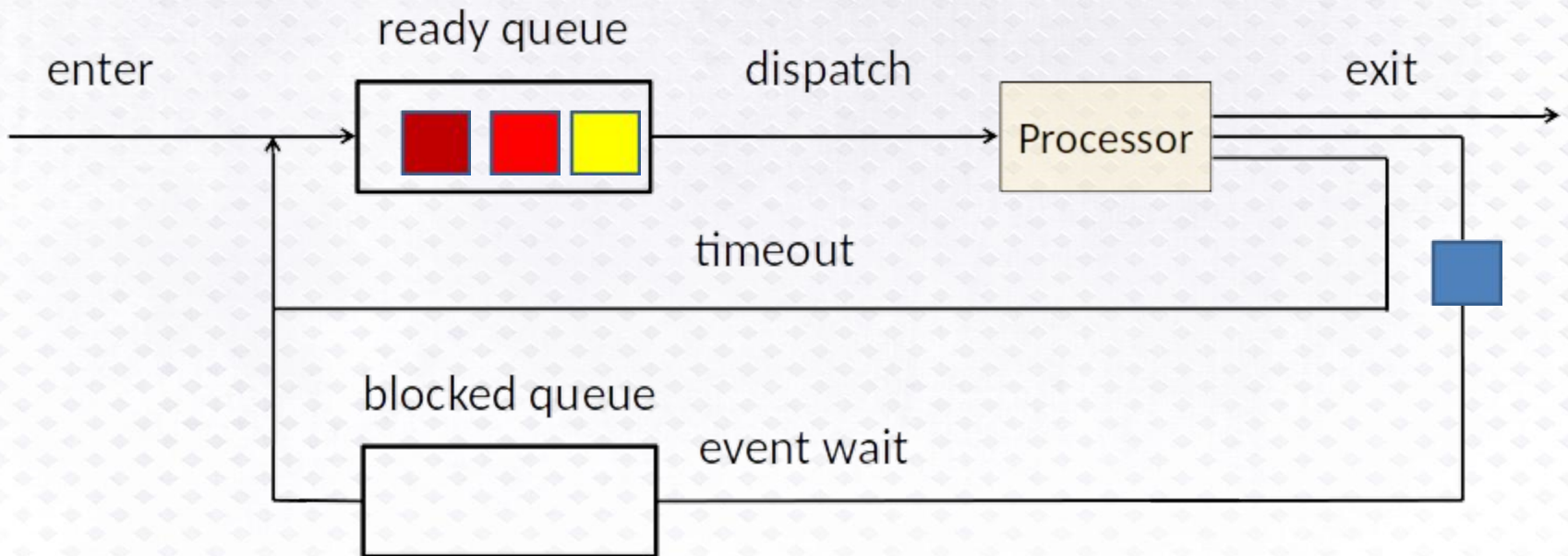
Process Management

To reflect the two pause states, use two queues



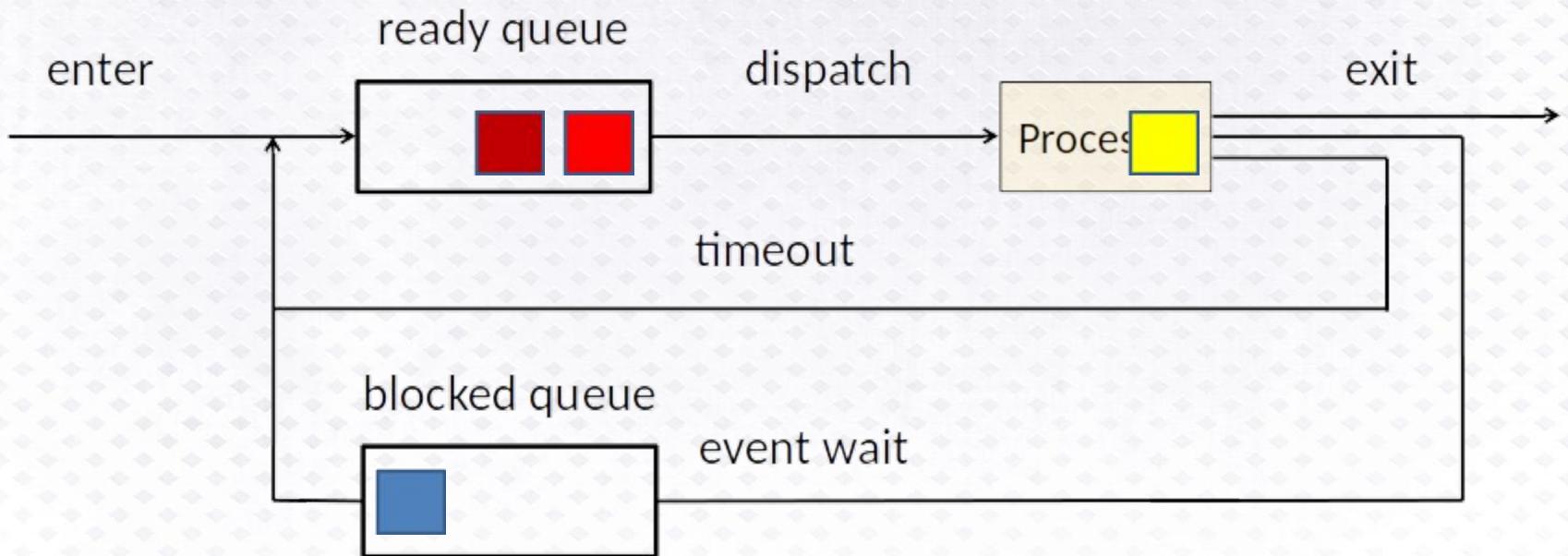
Process Management

To reflect the two pause states, use two queues



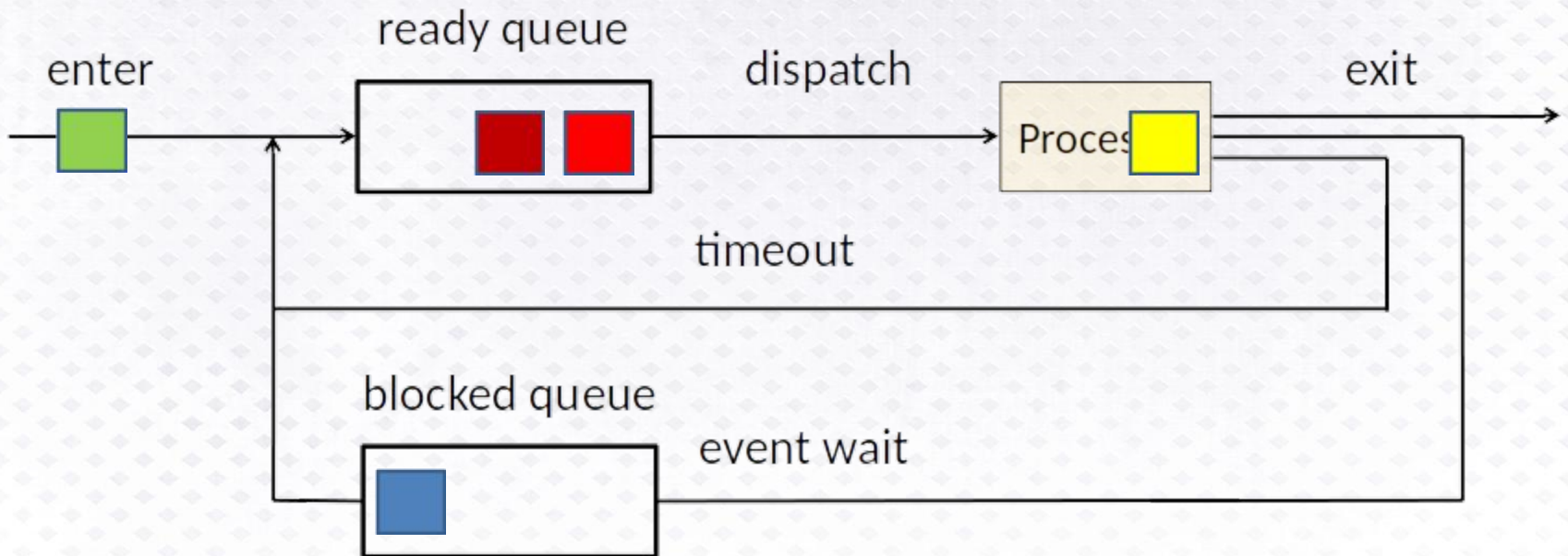
Process Management

To reflect the two pause states, use two queues



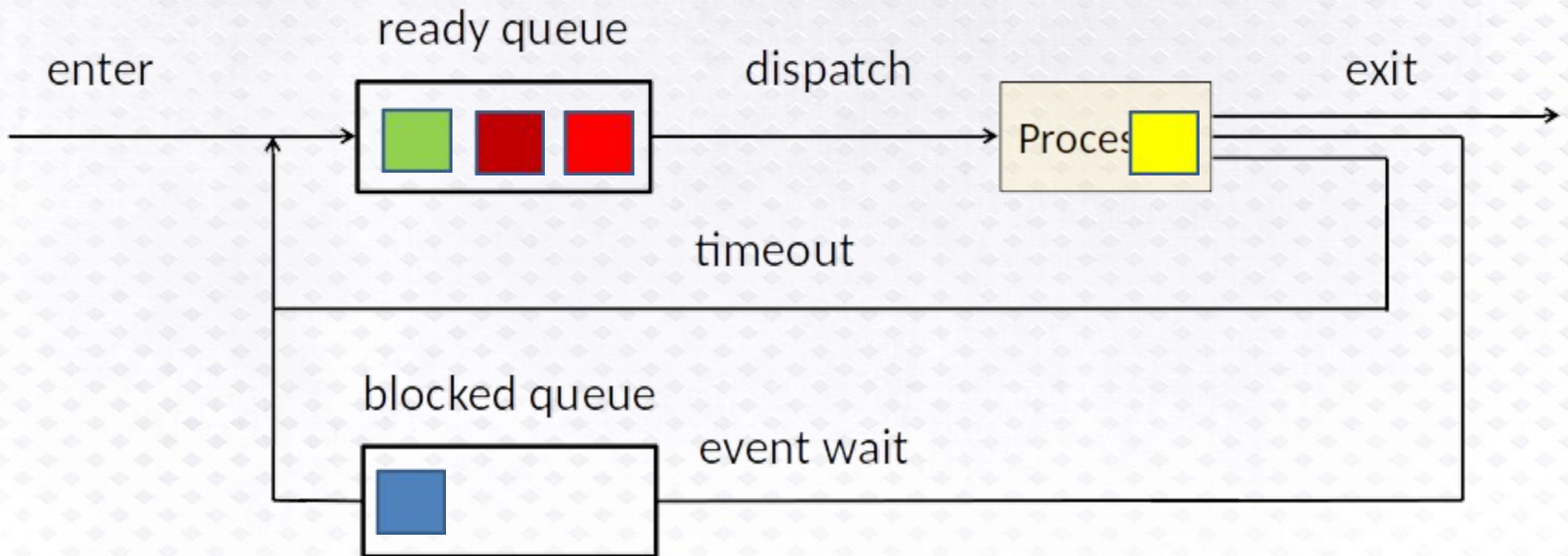
Process Management

To reflect the two pause states, use two queues



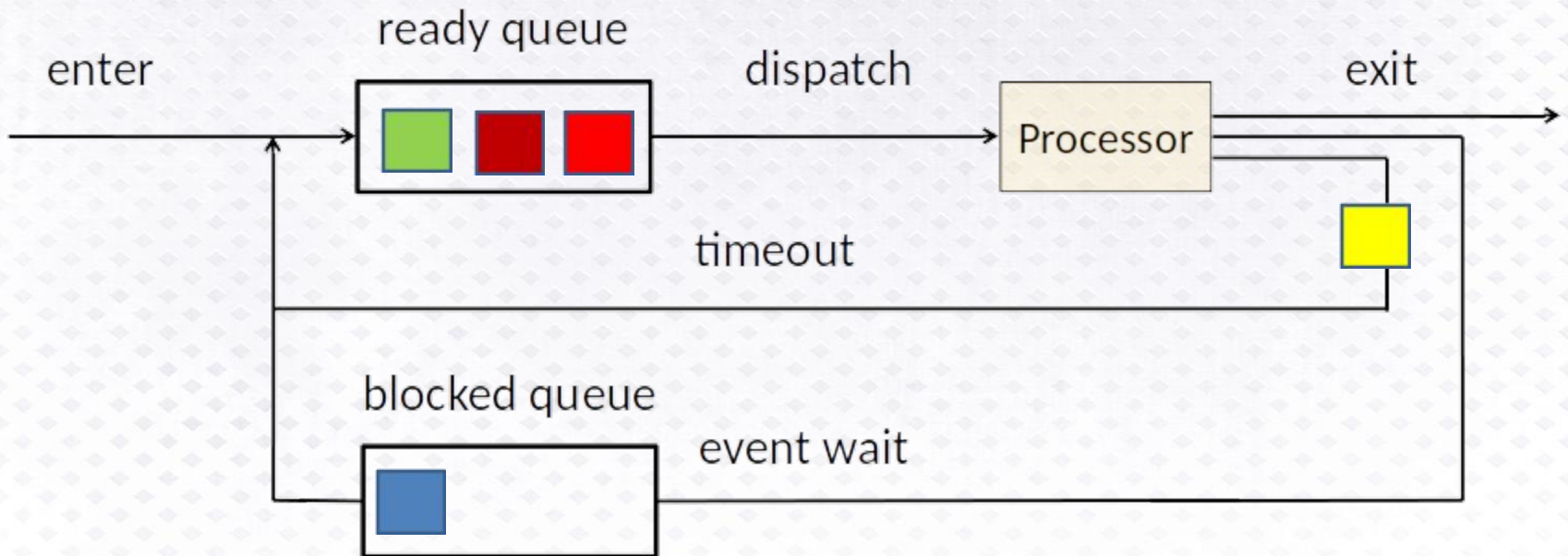
Process Management

To reflect the two pause states, use two queues



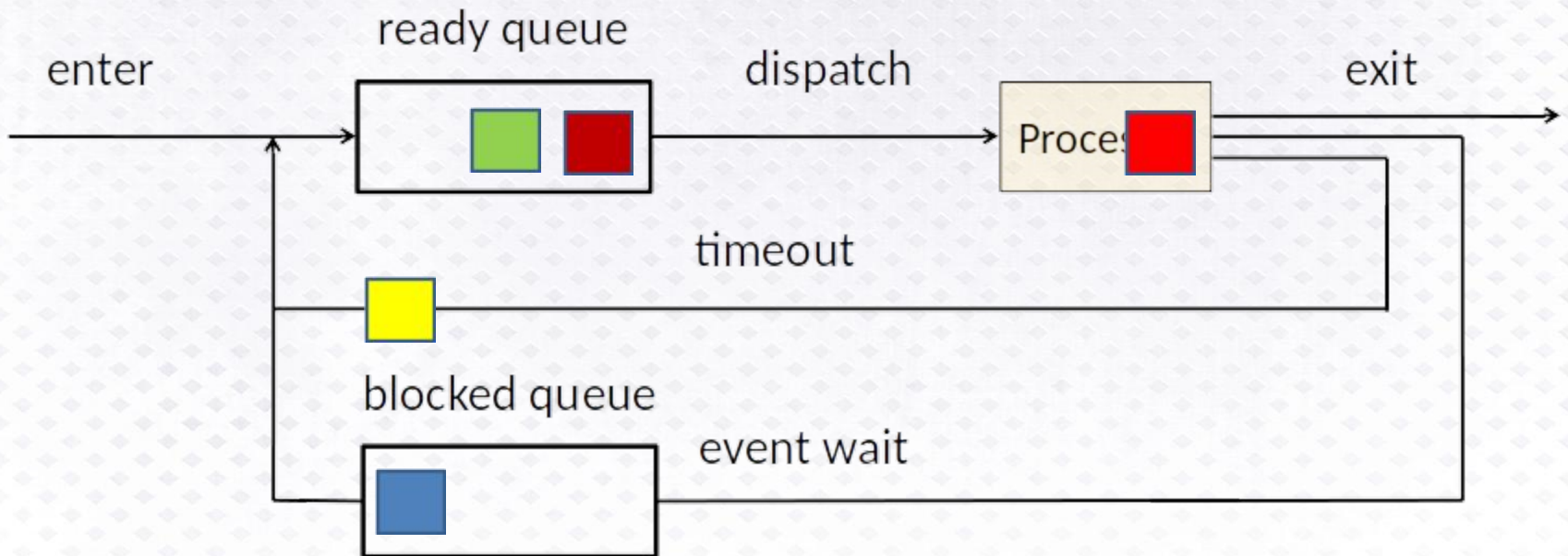
Process Management

To reflect the two pause states, use two queues



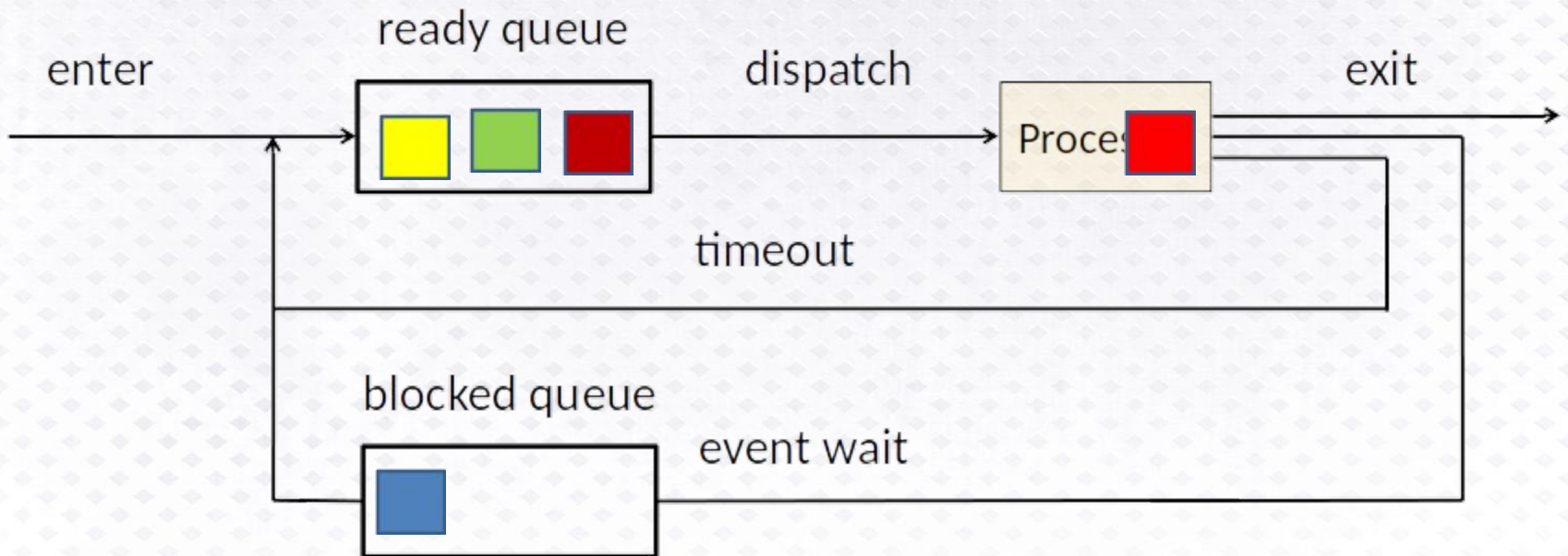
Process Management

To reflect the two pause states, use two queues



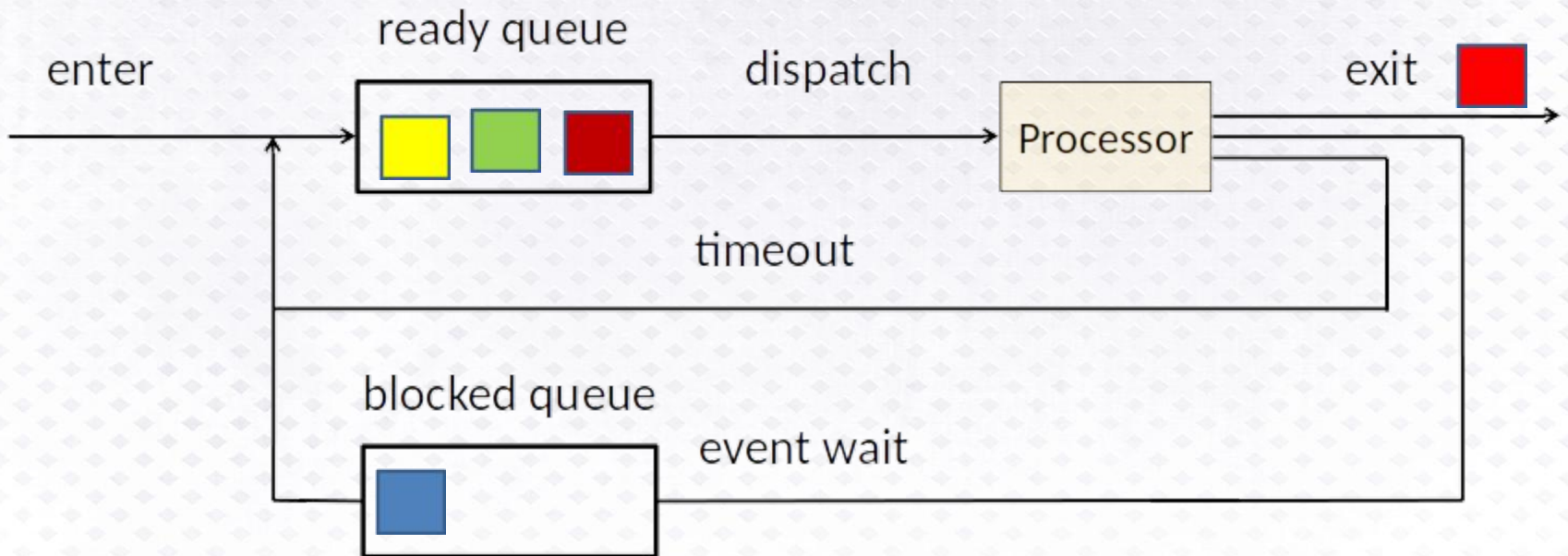
Process Management

To reflect the two pause states, use two queues



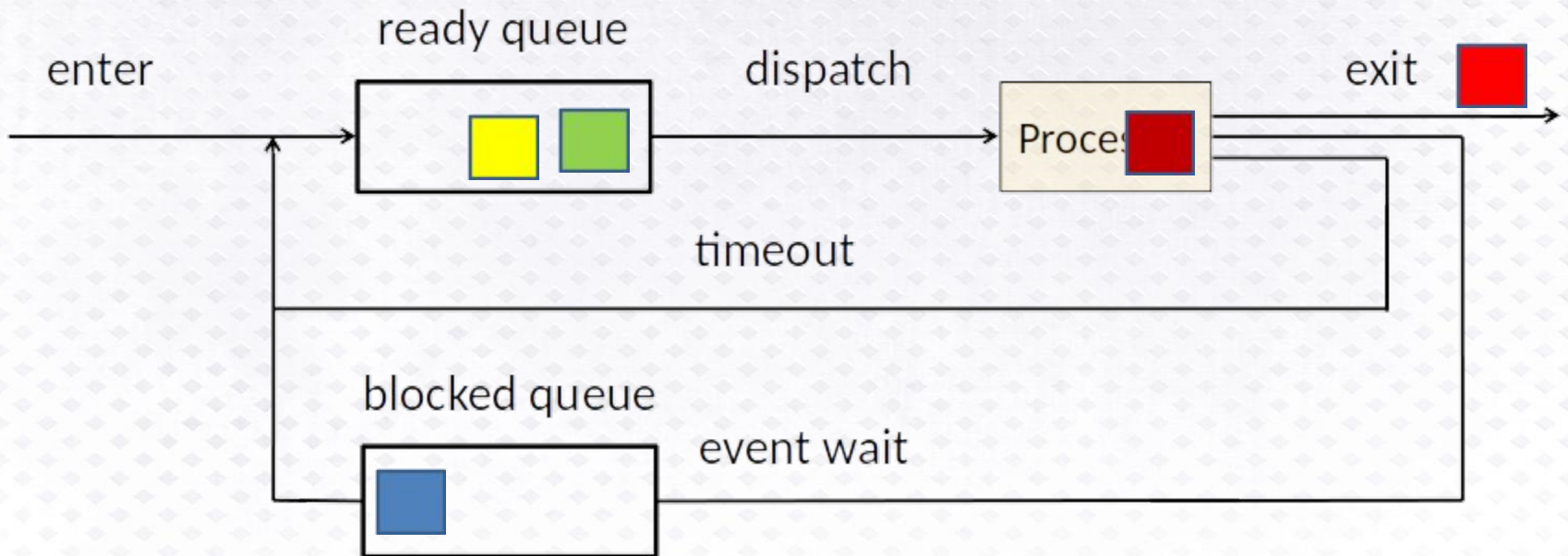
Process Management

To reflect the two pause states, use two queues



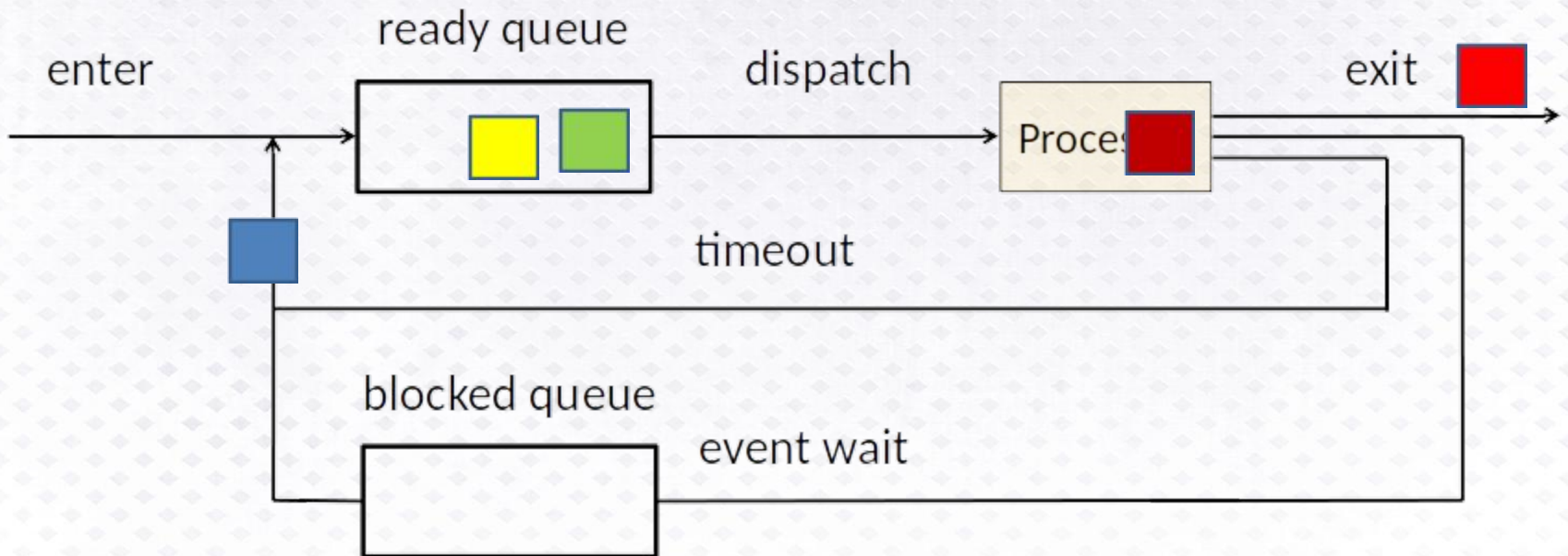
Process Management

To reflect the two pause states, use two queues



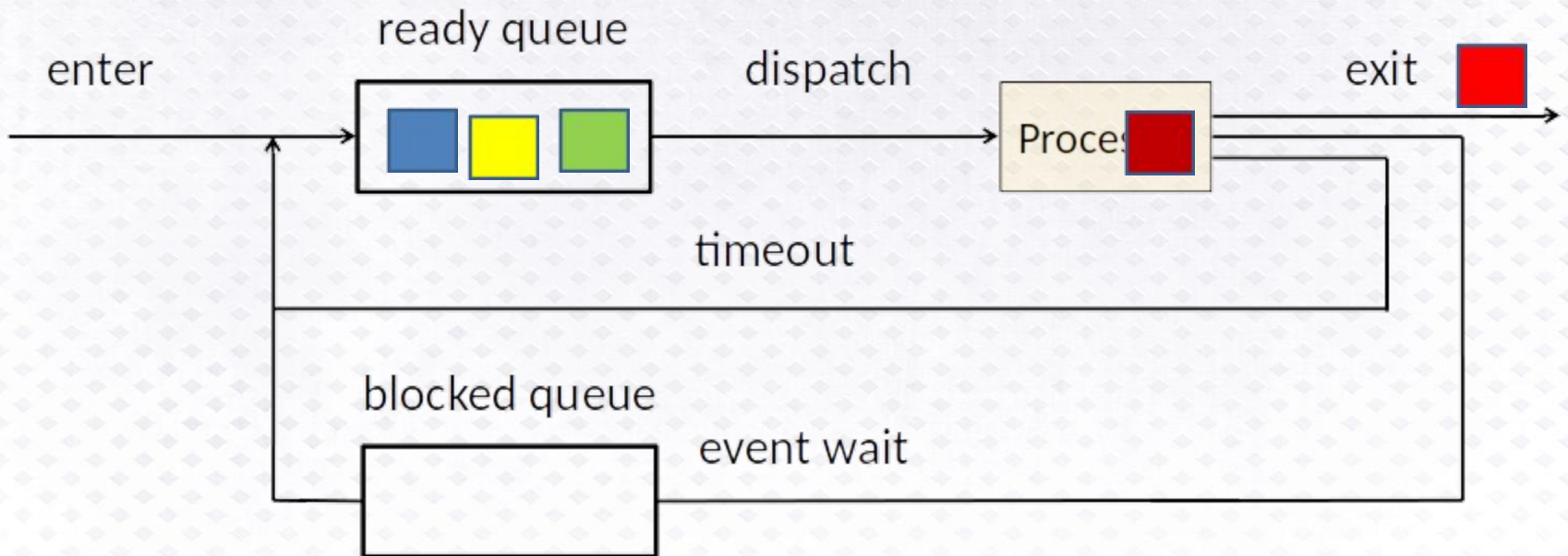
Process Management

To reflect the two pause states, use two queues



Process Management

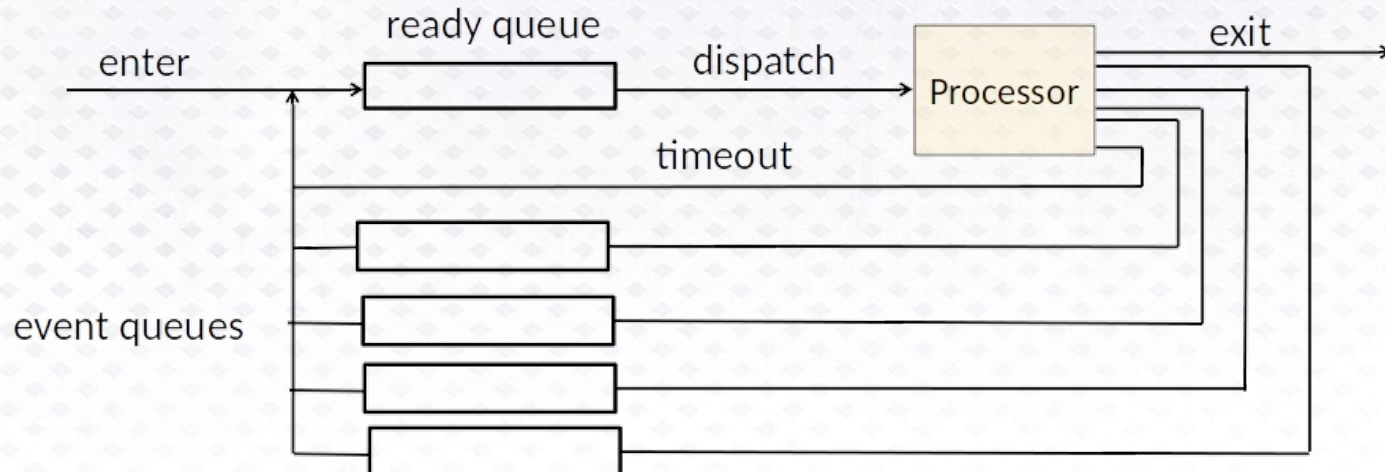
To reflect the two pause states, use two queues



Process Management

With single blocked queue when an event occurs need to cycle through the entire queue to see which process is waiting for the event

Solution is to have multiple queues



Suspended Processes

CPU is faster than I/O, so all processes could be waiting for I/O

Solution - Swap these processes to disk to free up memory and use processors effectively

Suspended Processes

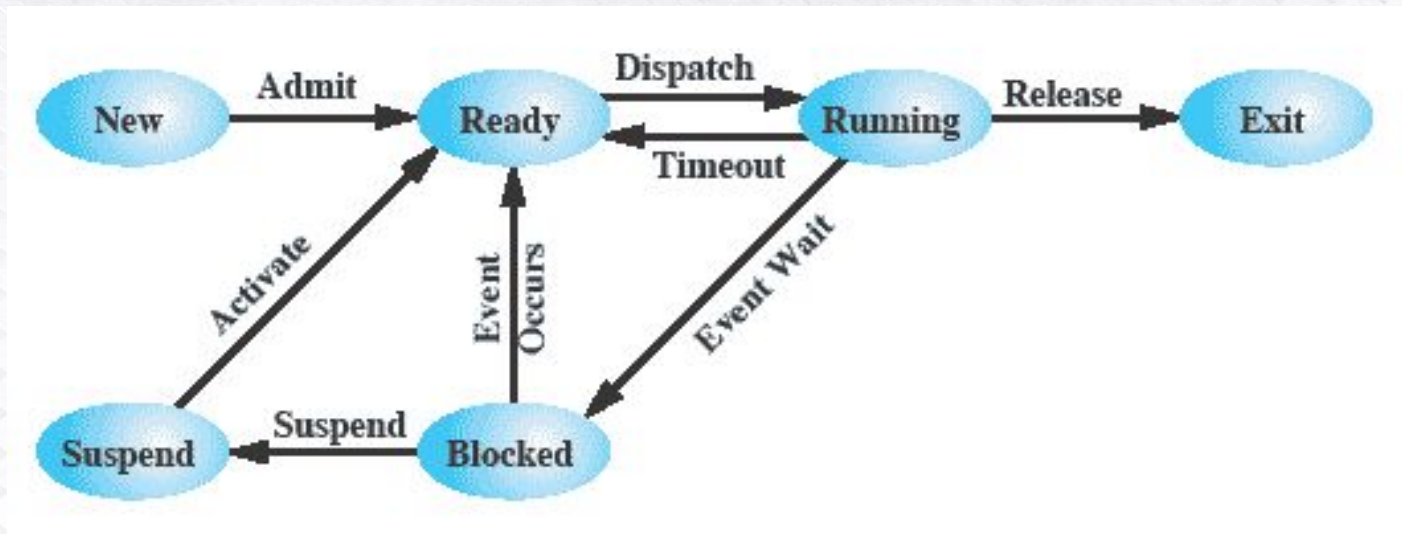
Blocked/Waiting state becomes **Suspend** state when swapped to disk

Two new states:

- Blocked / Suspend
 - Waiting process swapped out
- Ready / Suspend
 - Process could be executed, but swapped out

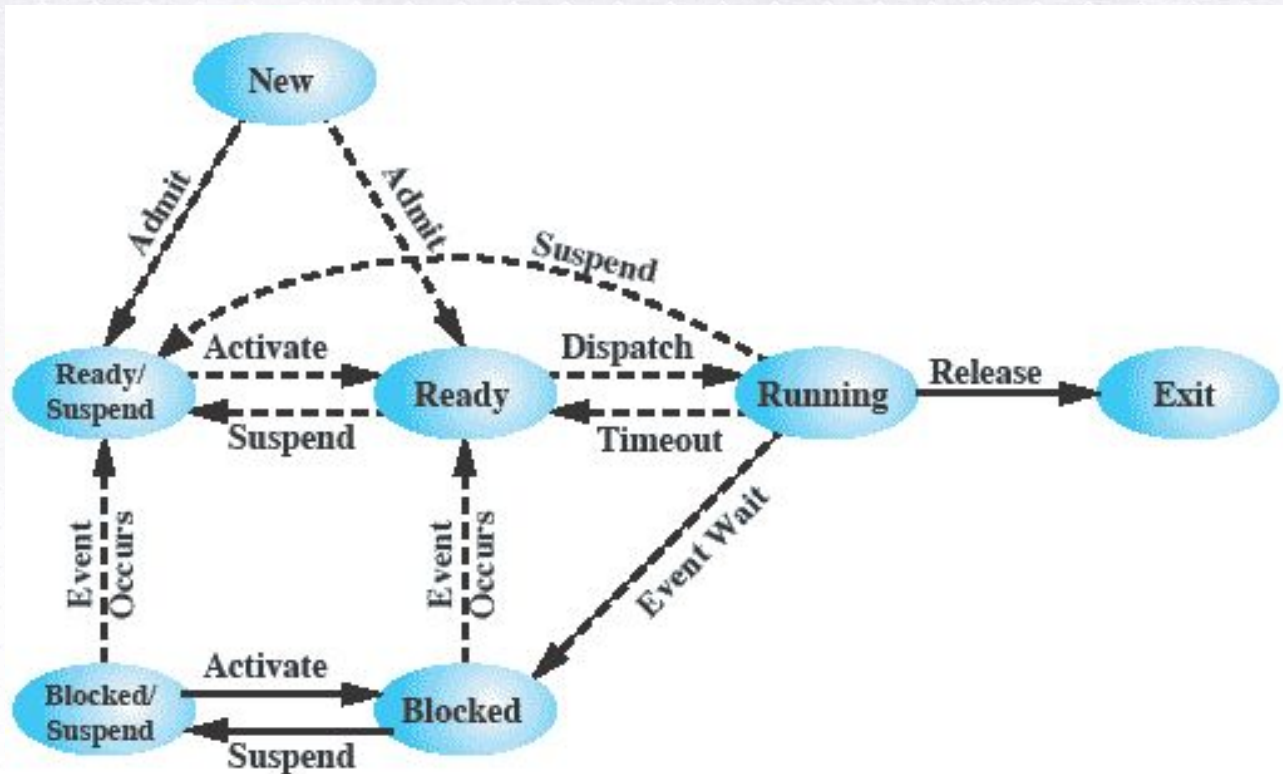
Suspended Processes

One Suspend state:



Suspended Processes

Two Suspend states:



Process Management

Process control block (PCB)

A *data structure* used by the OS to manage information about a process, including

- current value of the program counter
- values of all CPU registers for the process
- base and bound register values (or page tables)
- accounting information

Each *state* is represented by a list of PCBs, one for each process in that state

Process Control Block (PCB)

Single CPU implies a set of CPU registers containing values for currently executing process

Each time a process is moved to the running state:

- Register values for the currently running process are stored into its PCB
- Its PCB is moved to the list of the state into which it goes
- Register values of the new process moving into the running state are loaded into the CPU

The exchange of register information is called a *context switch*

Load Control

Determines the number of processes that will be resident in main memory

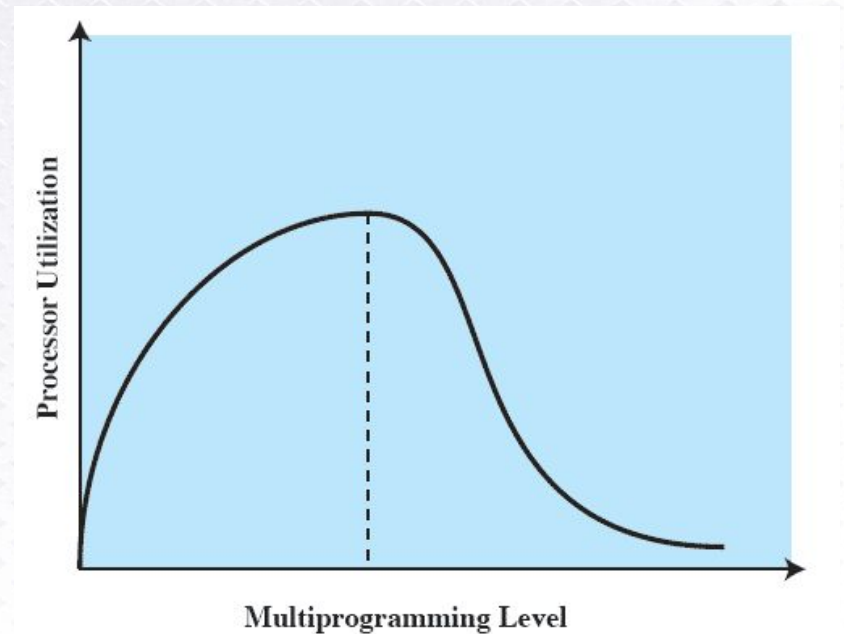
- The Multiprogramming level

Too few processes:

- often all processes blocked
- time will be spent in swapping

Too many processes:

- Thrashing



Process Suspension

To reduce the degree of multiprogramming, one or more of the currently resident processes must be suspended (swapped out):

- **Lowest priority process**
- **Faulting process** - Will be blocked anyway
- **Last process activated** - least likely to have its working set resident
- **Process with smallest resident set** - requires least effort to load
- **Largest process** - obtains the most free frames
- **Process with largest remaining execution window** - approximates the shortest-job-first scheduling approach (see later)

Unix Process States

System V revision 4

A process in UNIX is a set of data structures that provide the OS with all of the information necessary to manage and dispatch processes

Process creation is by means of the kernel system call, `fork()`. This causes the OS, in Kernel Mode, to:

- Allocate a slot in the process table for the new process.
- Assign a unique process ID to the child process.
- Copy process image of the parent, with the exception of any shared memory.
- Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
- Assign the child process to the Ready to Run state.
- Return the ID number of the child to the parent process, and a 0 value to the child process.

Unix Process States

System V revision 4

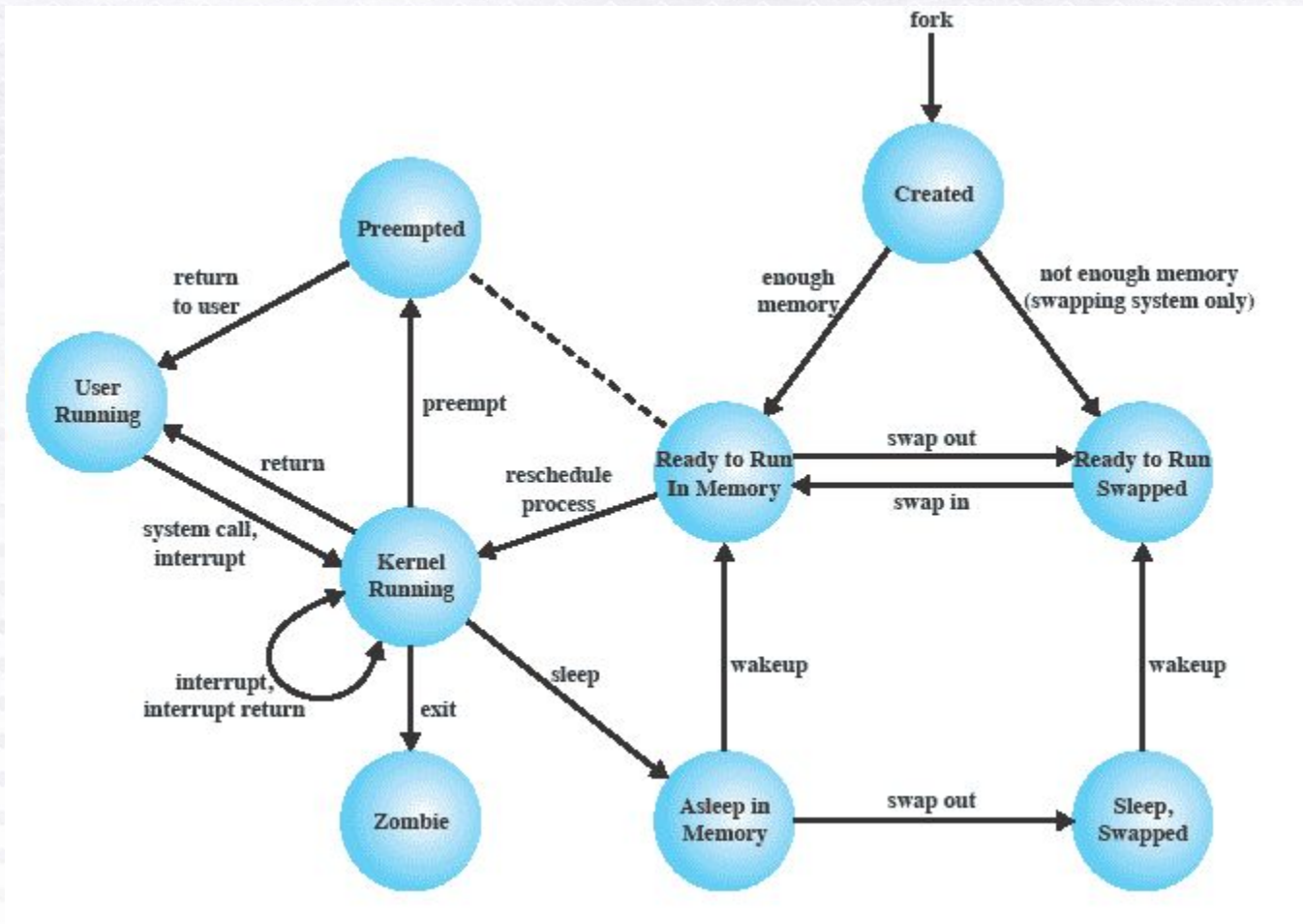
After creating the process the Kernel can do one of the following, as part of the dispatcher routine:

- Stay in the parent process.
- Transfer control to the child process
- Transfer control to another process.

Possible Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Possible Process States



CPU Scheduling

CPU Scheduling

The act of determining which process in the *ready* state should be moved to the *running* state

- Many processes may be in the ready state
- Only one process can be in the running state, making progress at any one time

Which one gets to move from ready to running?

CPU Scheduling

Nonpreemptive scheduling

The currently executing process gives up the CPU voluntarily

Preemptive scheduling

The operating system decides to favor another process, preempting the currently executing process

CPU Scheduling

Turnaround time

The amount of time between when a process arrives in the ready state the first time and when it exits the running state for the last time

% Delay

Calculated by:

$$(\text{turnaround} - \text{service time}) / \text{service time} * 100$$

Processor Utilization

Percentage of time CPU doing useful work

CPU Scheduling Algorithms

First-Come, First-Served

Processes are moved to the CPU in the order in which they arrive in the running state

Shortest Job Next

Process with shortest estimated running time in the ready state is moved into the running state first

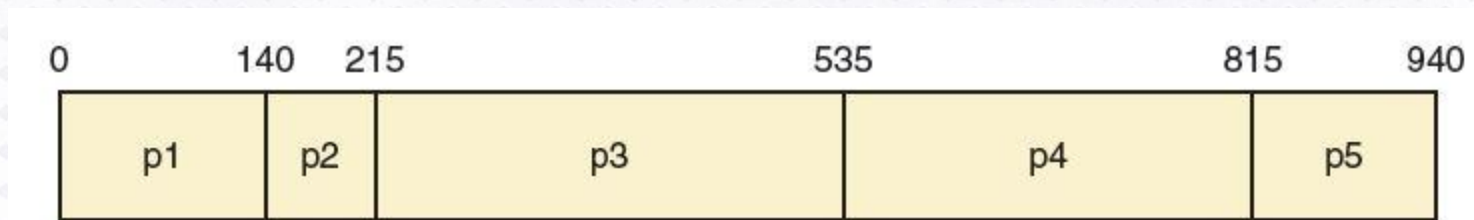
Round Robin

Each process runs for a specified time slice and moves from the running state to the ready state to await its next turn if not finished

First-Come, First-Served

Process	Service Time
p1	140
p2	75
p3	320
p4	280
p5	125

What is the average turn-around time?



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0

First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 140

P1

First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320			
P4	60	280			
P5	80	125			

0 140 215

P1

P2

First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320	535		
P4	60	280			
P5	80	125			

0 140 215 535

P1

P2

P3

First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320	535		
P4	60	280	815		
P5	80	125			



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320	535		
P4	60	280	815		
P5	80	125	940		



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215		
P3	40	320	535		
P4	60	280	815		
P5	80	125	940		



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	535		
P4	60	280	815		
P5	80	125	940		



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	535	495	
P4	60	280	815		
P5	80	125	940		



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	535	495	
P4	60	280	815	755	
P5	80	125	940		



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	535	495	
P4	60	280	815	755	
P5	80	125	940	860	



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	
P3	40	320	535	495	
P4	60	280	815	755	
P5	80	125	940	860	



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	535	495	
P4	60	280	815	755	
P5	80	125	940	860	



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	535	495	55
P4	60	280	815	755	
P5	80	125	940	860	



First-Come, First-Served Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	535	495	55
P4	60	280	815	755	170
P5	80	125	940	860	



First-Come, First-Served Worked Example

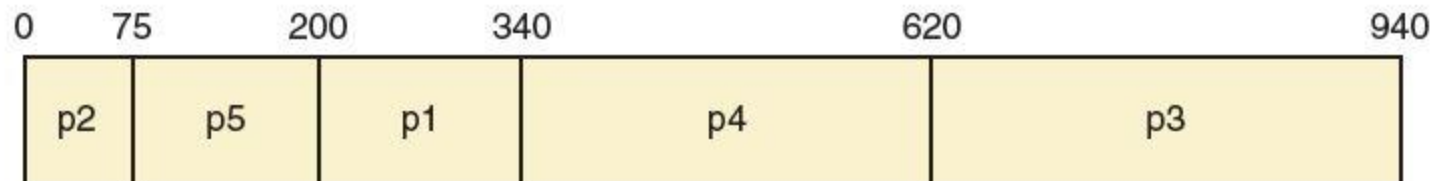
Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	535	495	55
P4	60	280	815	755	170
P5	80	125	940	860	588



Shortest Job Next

Process	Service Time
p1	140
p2	75
p3	320
p4	280
p5	125

What is the average turn-around time?



Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0

Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 140

P1

Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320			
P4	60	280			
P5	80	125			

0 140 215

P1

P2

Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320			
P4	60	280			
P5	80	125	340		

0 140 215 340

P1

P2

P5

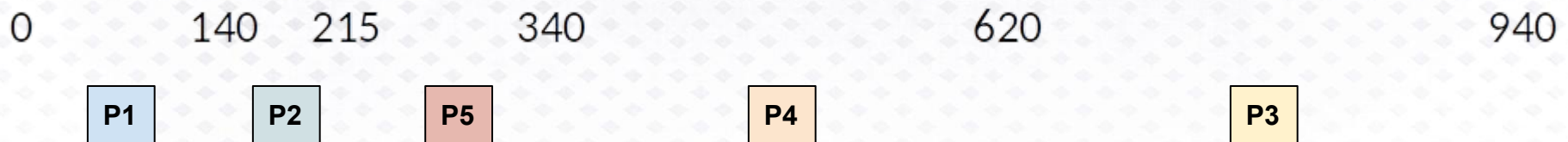
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320			
P4	60	280	620		
P5	80	125	340		



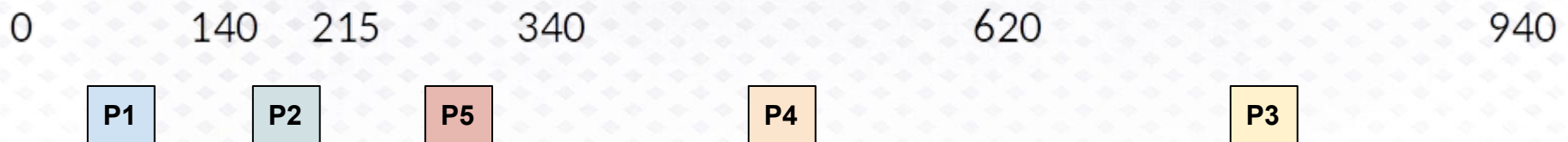
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140		
P2	20	75	215		
P3	40	320	940		
P4	60	280	620		
P5	80	125	340		



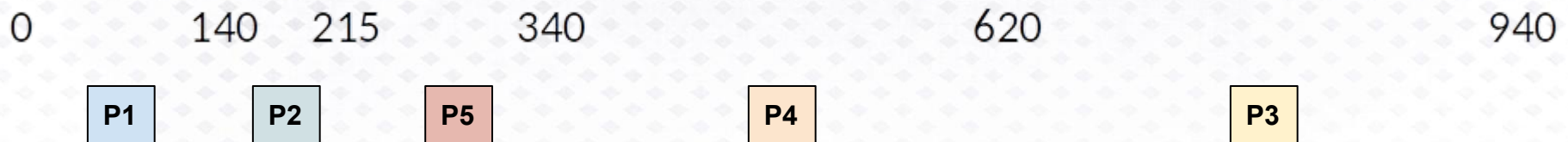
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215		
P3	40	320	940		
P4	60	280	620		
P5	80	125	340		



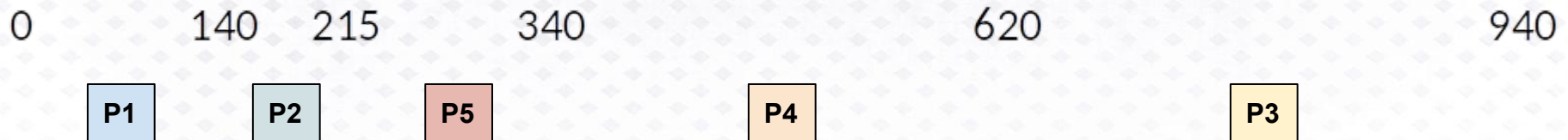
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	940		
P4	60	280	620		
P5	80	125	340		



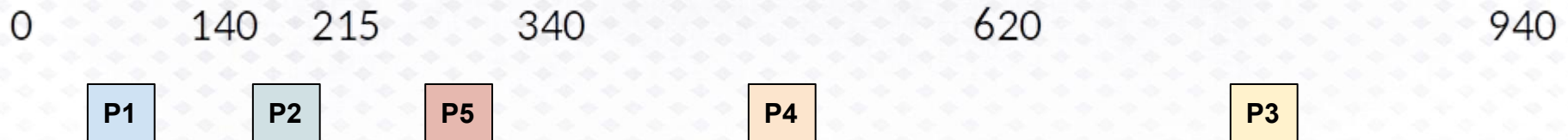
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	940	900	
P4	60	280	620		
P5	80	125	340		



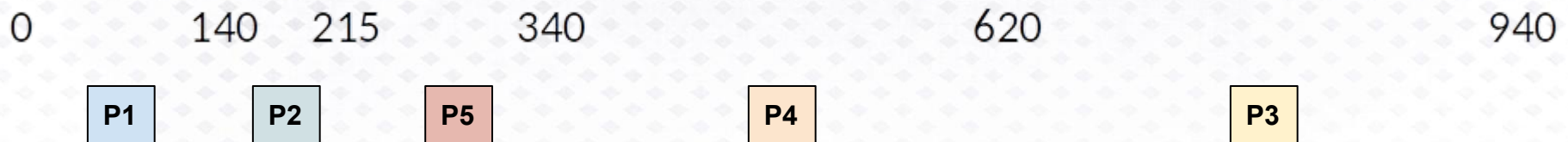
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	940	900	
P4	60	280	620	560	
P5	80	125	340		



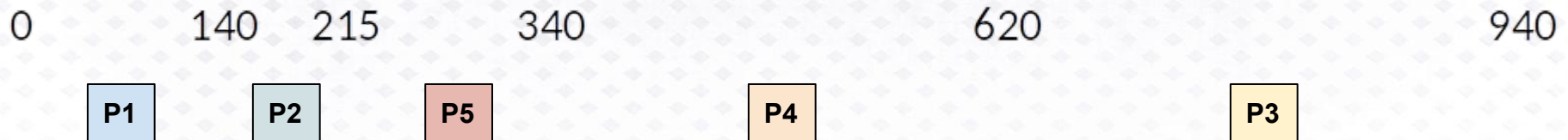
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	
P2	20	75	215	195	
P3	40	320	940	900	
P4	60	280	620	560	
P5	80	125	340	260	



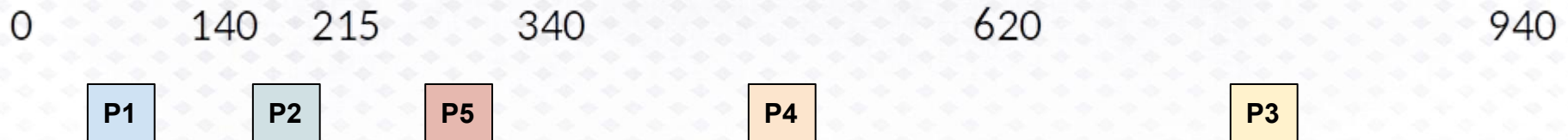
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	
P3	40	320	940	900	
P4	60	280	620	560	
P5	80	125	340	260	



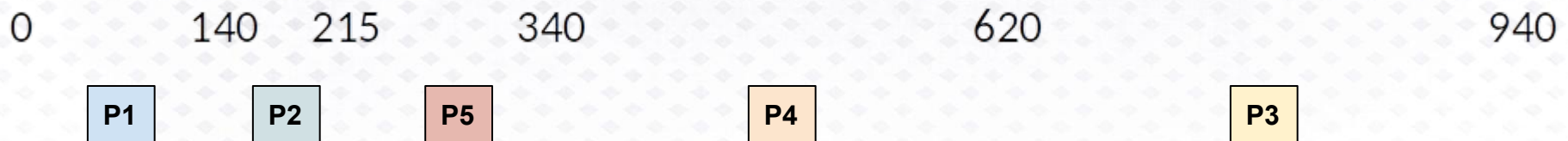
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	940	900	
P4	60	280	620	560	
P5	80	125	340	260	



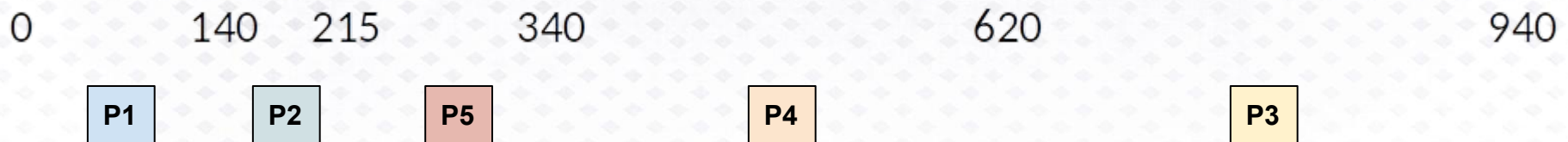
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	940	900	611
P4	60	280	620	560	
P5	80	125	340	260	



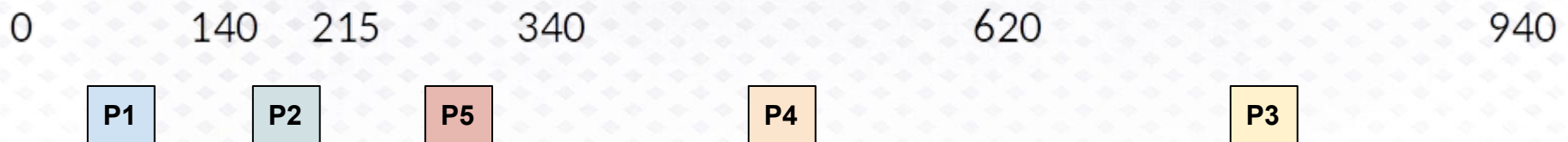
Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	940	900	611
P4	60	280	620	560	100
P5	80	125	340	260	



Shortest Job Next Worked Example

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	140	140	0
P2	20	75	215	195	105
P3	40	320	940	900	611
P4	60	280	620	560	100
P5	80	125	340	260	108



Round Robin

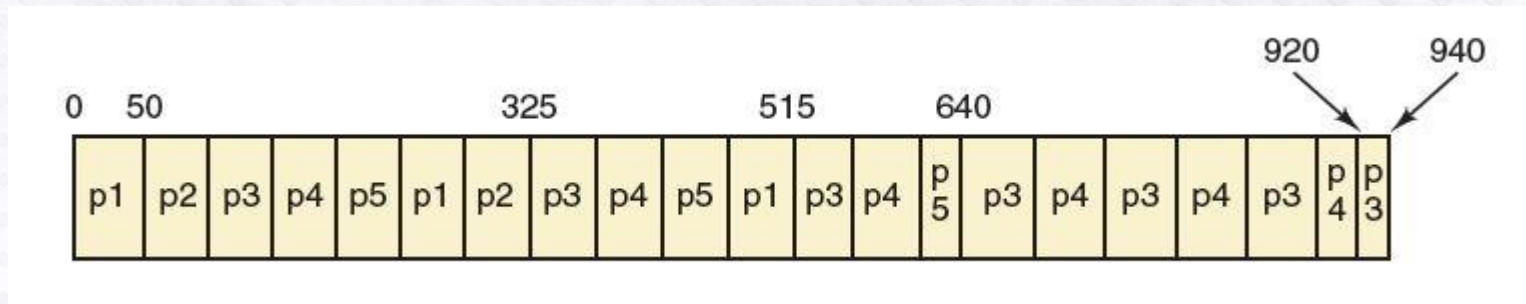
Every process is treated the same!

Time slice (quantum)

The amount of time each process receives before being preempted and returned to the ready state to allow another process its turn

Round Robin

Suppose the time slice is 50



What is the average turnaround time?

Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0

Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 50

P1

Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 50

P1	P2
----	----

Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 50

P1	P2	P3
----	----	----

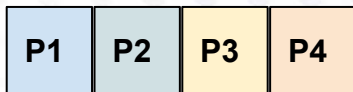
Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 50



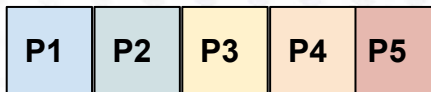
Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 50



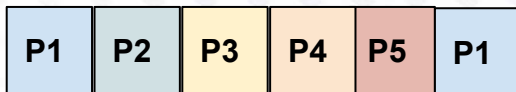
Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75			
P3	40	320			
P4	60	280			
P5	80	125			

0 50

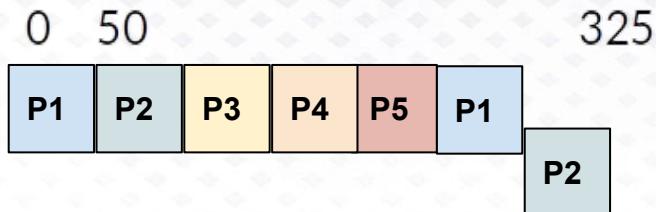


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

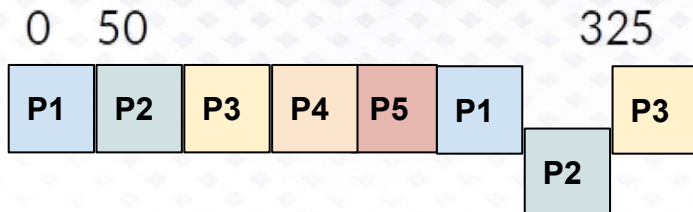


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

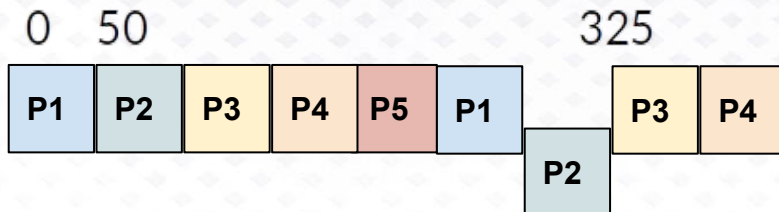


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

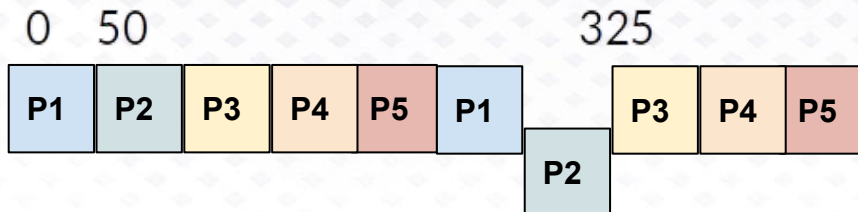


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140			
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

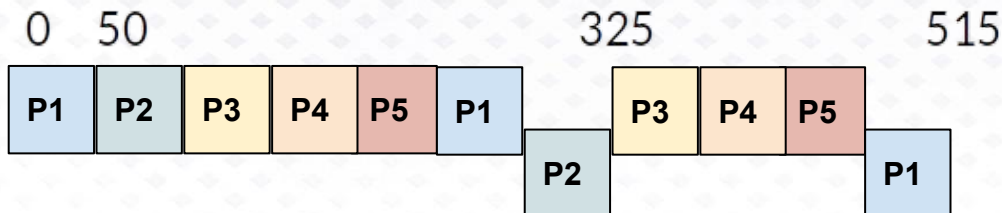


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

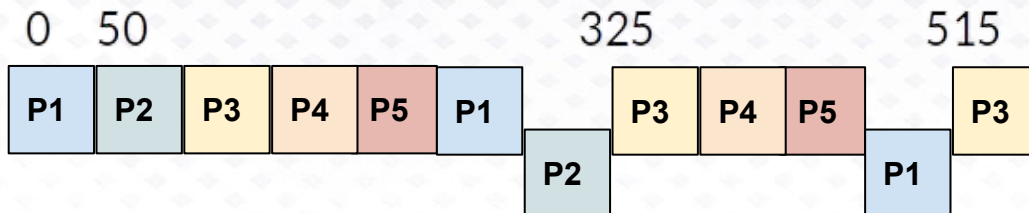


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

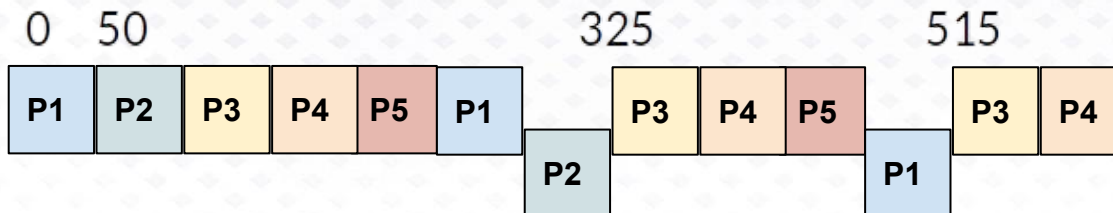


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125			

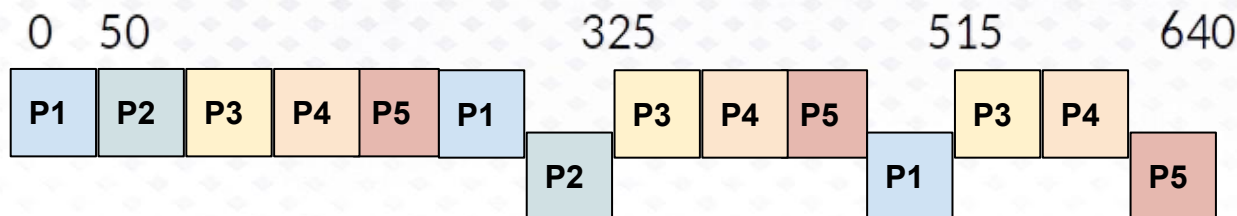


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125	640		

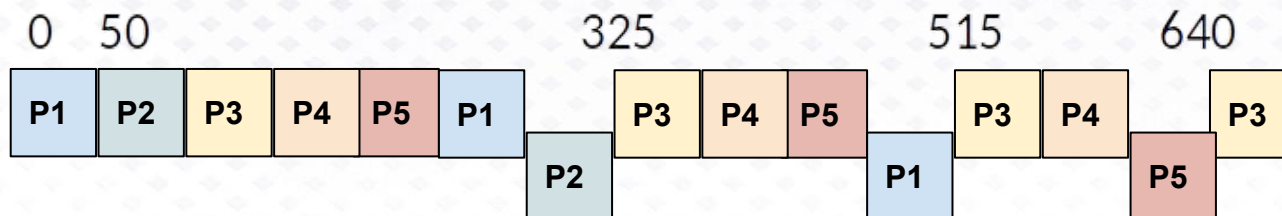


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125	640		

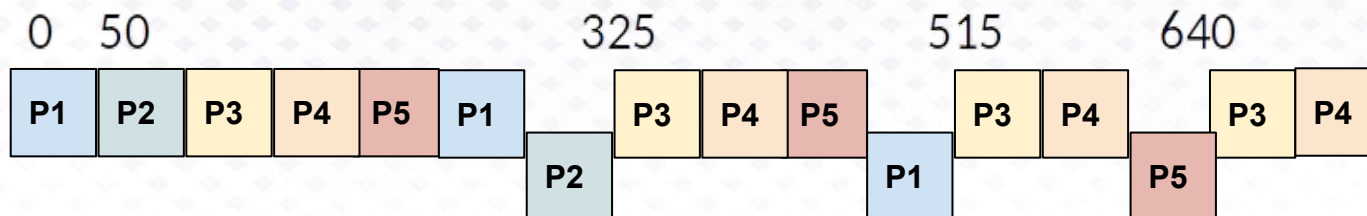


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125	640		

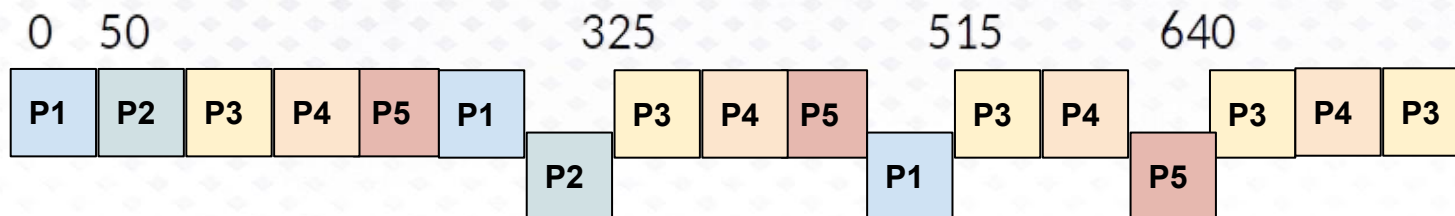


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125	640		

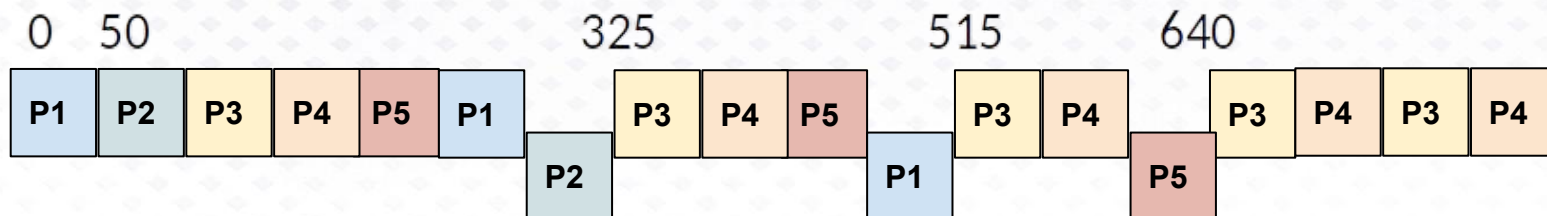


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125	640		

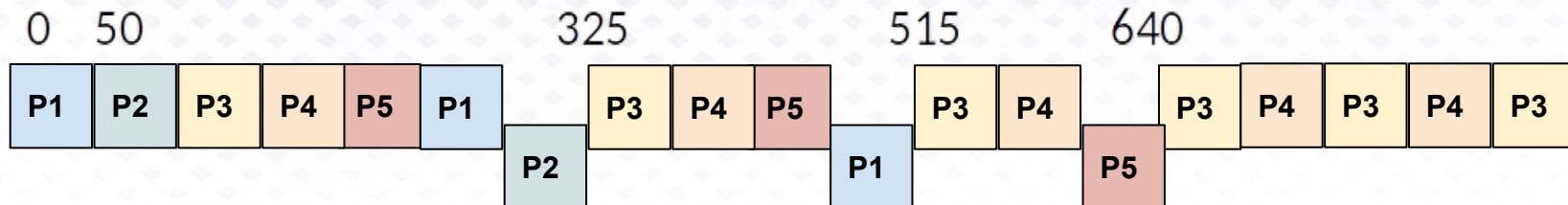


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280			
P5	80	125	640		

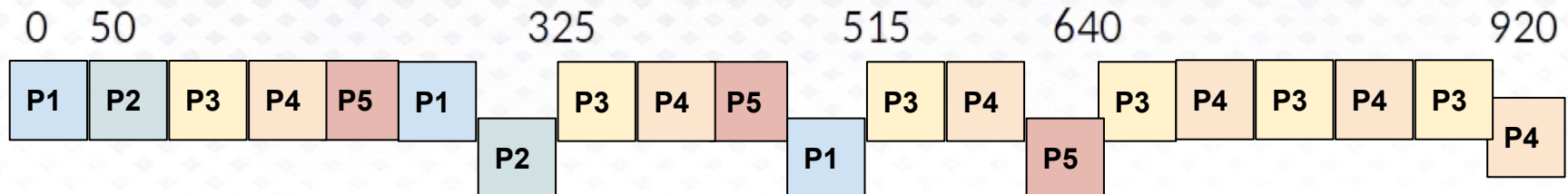


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320			
P4	60	280	920		
P5	80	125	640		

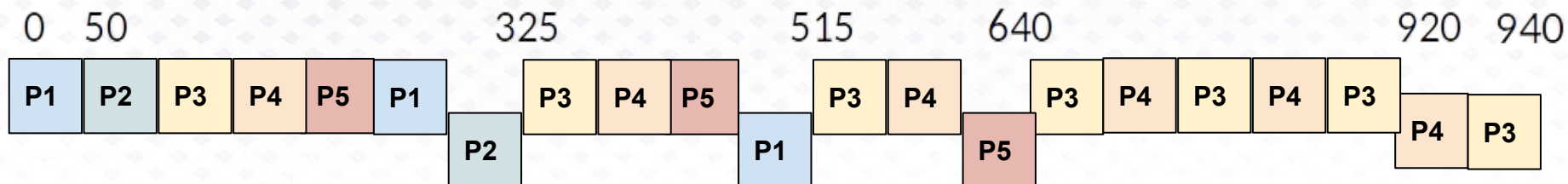


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515		
P2	20	75	325		
P3	40	320	940		
P4	60	280	920		
P5	80	125	640		

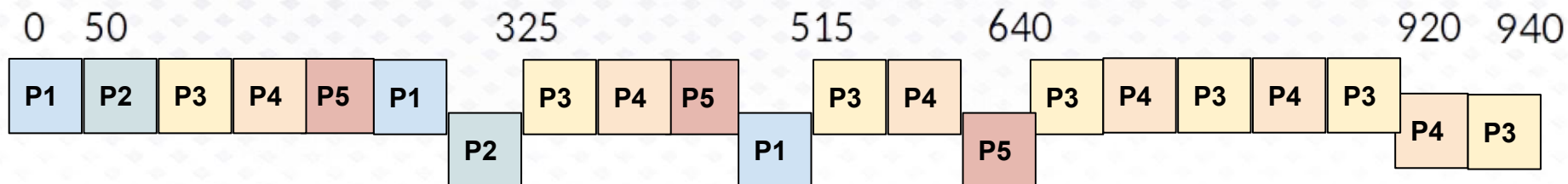


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	
P2	20	75	325		
P3	40	320	940		
P4	60	280	920		
P5	80	125	640		

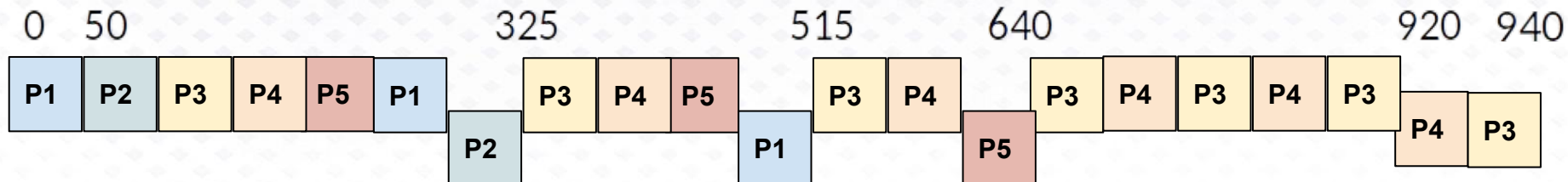


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	
P2	20	75	325	305	
P3	40	320	940		
P4	60	280	920		
P5	80	125	640		

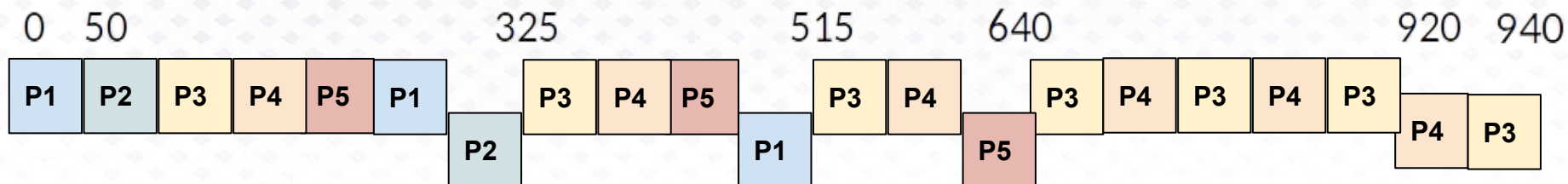


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	
P2	20	75	325	305	
P3	40	320	940	900	
P4	60	280	920		
P5	80	125	640		

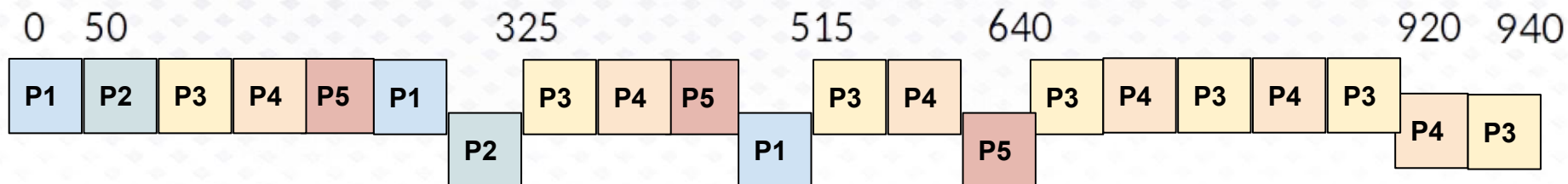


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	
P2	20	75	325	305	
P3	40	320	940	900	
P4	60	280	920	860	
P5	80	125	640		

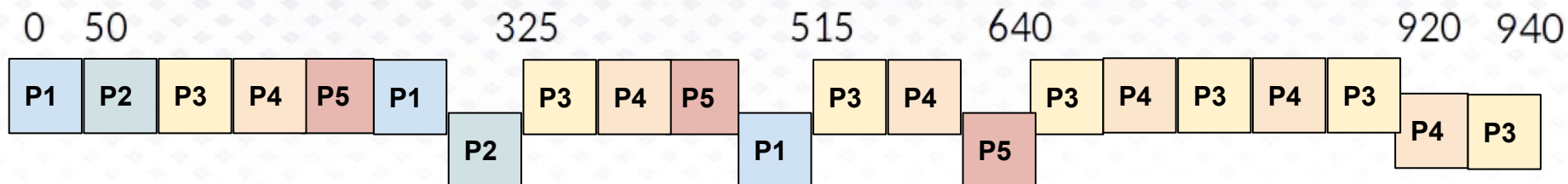


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	
P2	20	75	325	305	
P3	40	320	940	900	
P4	60	280	920	860	
P5	80	125	640	560	

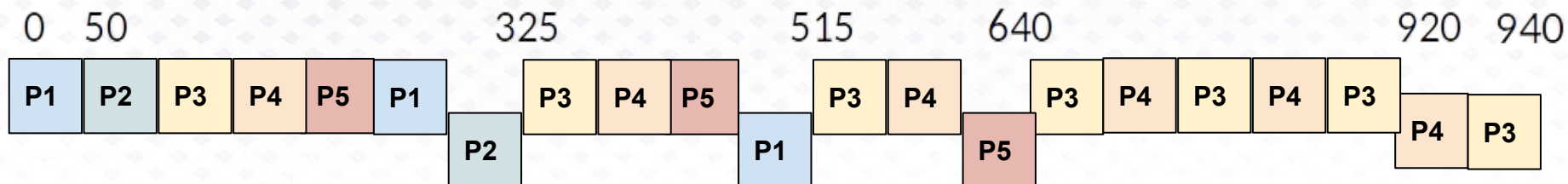


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	268
P2	20	75	325	305	
P3	40	320	940	900	
P4	60	280	920	860	
P5	80	125	640	560	

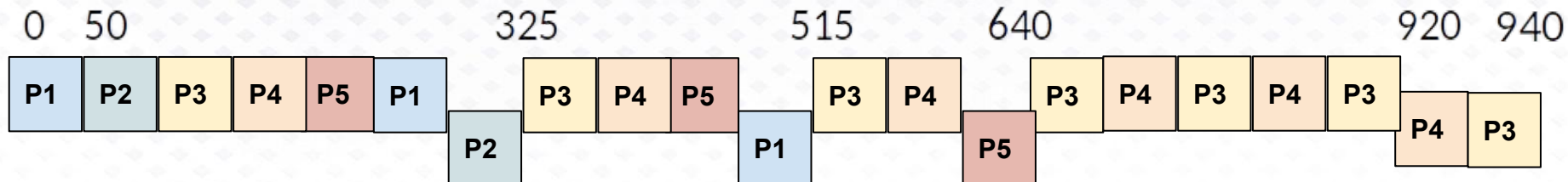


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	268
P2	20	75	325	305	307
P3	40	320	940	900	
P4	60	280	920	860	
P5	80	125	640	560	

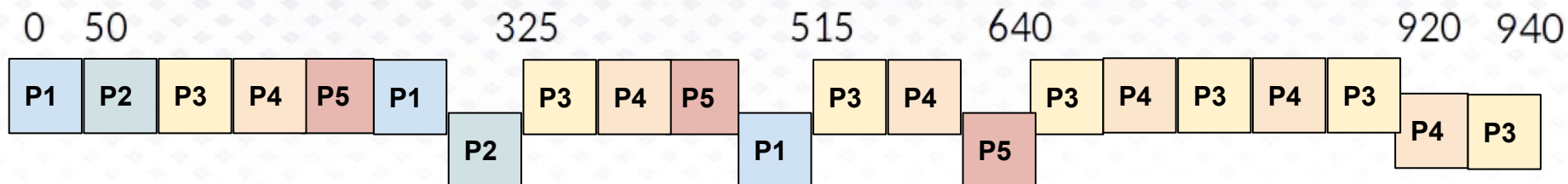


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	268
P2	20	75	325	305	307
P3	40	320	940	900	181
P4	60	280	920	860	
P5	80	125	640	560	

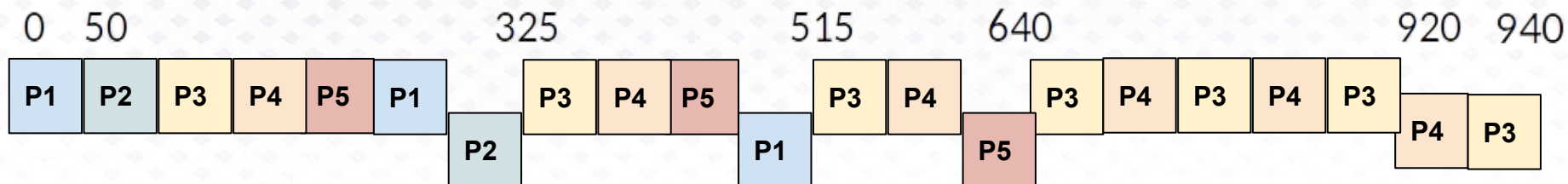


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	268
P2	20	75	325	305	307
P3	40	320	940	900	181
P4	60	280	920	860	207
P5	80	125	640	560	

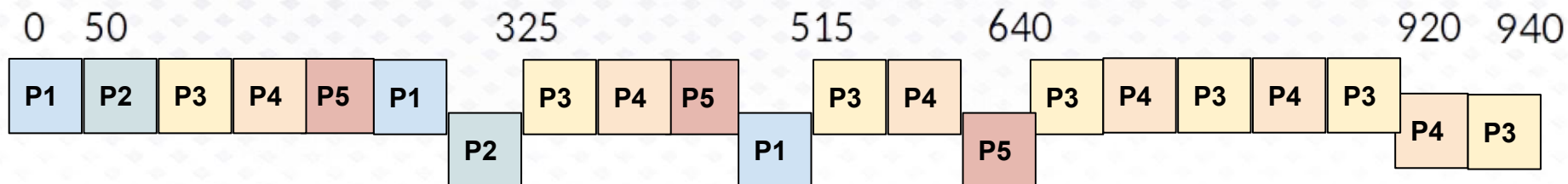


Round Robin

Worked Example

(Time slice: 50)

Process Name	Arrival Time	Service Time	Finish Time	Turnaround Time	% Delay
P1	0	140	515	515	268
P2	20	75	325	305	307
P3	40	320	940	900	181
P4	60	280	920	860	207
P5	80	125	640	560	348



CPU Scheduling Algorithms

Are these scheduling algorithms preemptive or non-preemptive? Explain

First-Come, First-Served?

Shortest Job Next?

Round Robin?

Memory Management

Operating systems must employ techniques to

- Track where and how a program resides in memory
- Convert **logical addresses** into actual **addresses**

Logical address

Reference to a stored value relative to the program making the reference

Physical address

Actual address in main memory

Memory Management

Responsibilities:

- Process isolation
- Automatic allocation and management
- Support of modular programming
- Protection and access control
- Long-term storage

Memory Management

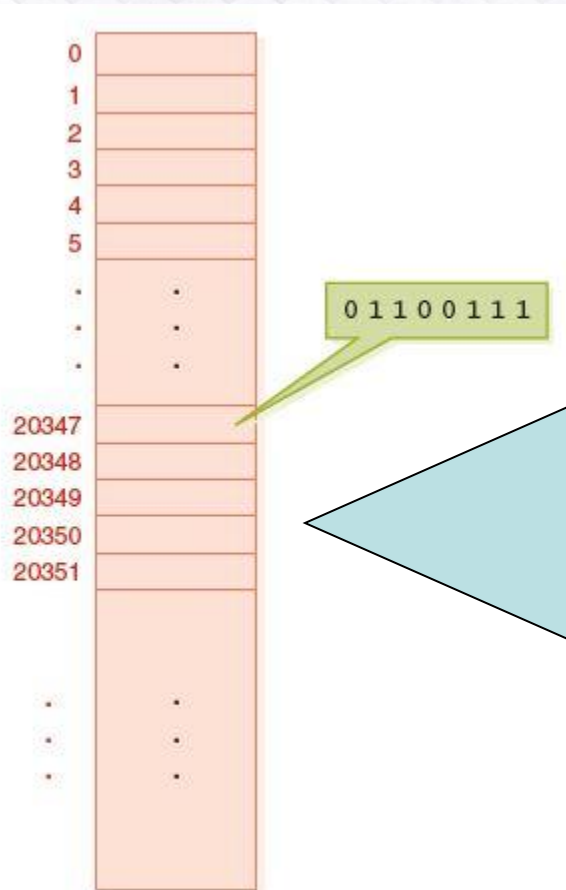


FIGURE 10.4 Memory is a continuous set of bits referenced by specific addresses

Program 1:
sum is assigned memory
location 23, a location
relative to Program 1

OS must map sum (relative location 23)
to a specific physical address

Logical address for sum (23) is bound to a
physical address in memory before the
program runs

Single Contiguous MM



FIGURE 10.5 Main memory divided into two sections

There are only two programs in memory

The operating system

The application program

This approach is called **single contiguous memory management**

Single Contiguous MM

In concrete terms:

A **logical address** is simply an **integer** value relative to the **starting point** of the program

A **physical address** is a logical address added to the starting location of the program in main memory

Single Contiguous MM

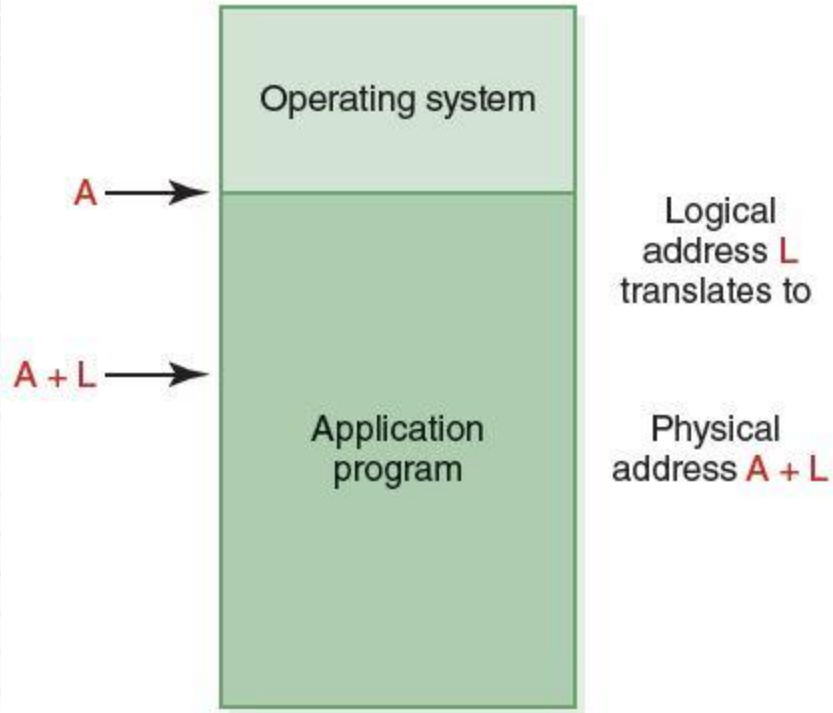


FIGURE 10.6 Binding a logical address to a physical address

If A is location 100, and the application program is Program 1, then sum is stored at location 123.

Partition Memory Management

Single contiguous MM has only the OS and one other program in memory at one time

Partition MM has the OS and any number of other programs in memory at one time

There are two schemes for dividing up memory for programs:

- **Fixed partitions** Main memory is divided into a fixed number of partitions into which programs can be loaded
- **Dynamic partitions** Partitions are created as needed to fit the programs waiting to be loaded

Partition Memory Management

Memory is divided into a set of partitions, some empty and some allocated to programs

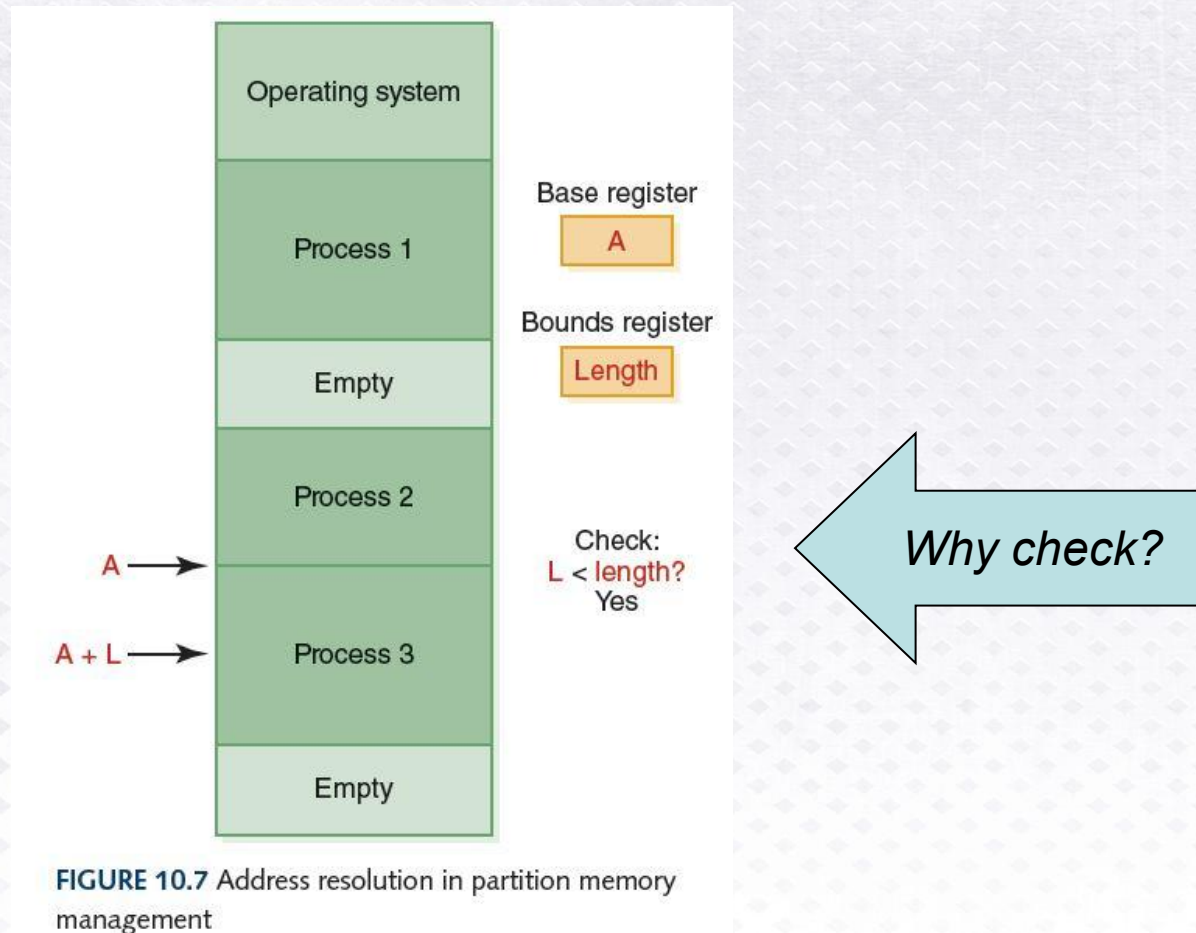
Base register

A register that holds the beginning address of the current partition (the one that is running)

Bounds register

A register that holds the length of the current partition

Partition Memory Management



Partition Selection Algorithms

Which partition should we allocate to a new program?

- **First fit** Allocate program to the first partition big enough to hold it
- **Best fit** Allocated program to the smallest partition big enough to hold it
- **Worst fit** Allocate program to the largest partition big enough to hold it
- **Next fit** Allocate program to next partition big enough to hold it

Worked Examples of Partition Management Algorithms in a Fixed Partition System

Fixed Partition Problem

A: 1000
B: 700
C: 750
D: 1500
E: 300
F: 350

Requests come in for blocks of the following sizes:

1000, 25, 780, 1600, and 325

and need to be allocated to the **fixed memory partitions** on the left

What block will be assigned to each request if the

- *first-fit algorithm is used?*
- *best-fit algorithm is used?*
- *next-fit algorithm is used?*

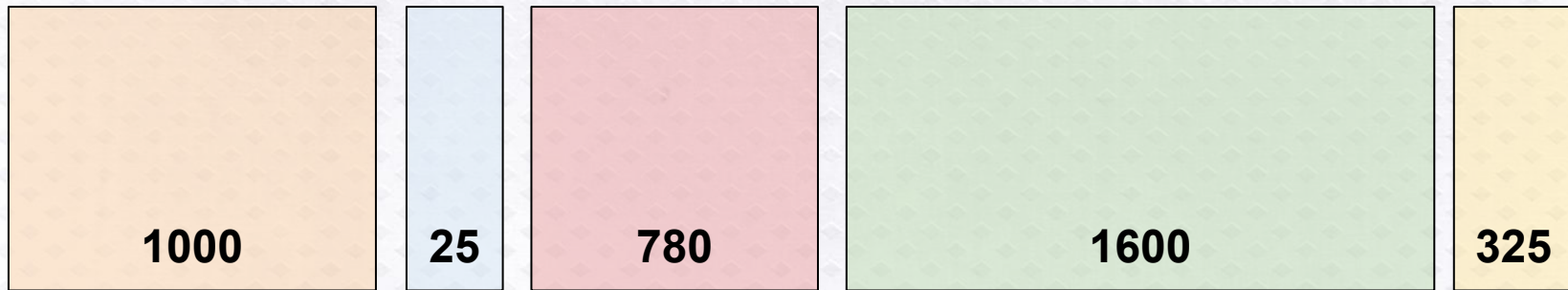
(Treat each request as an independent event)

Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



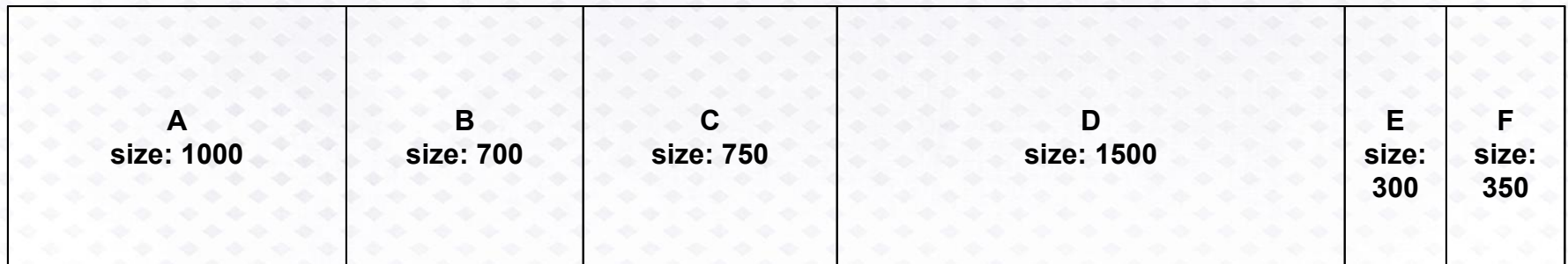
A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

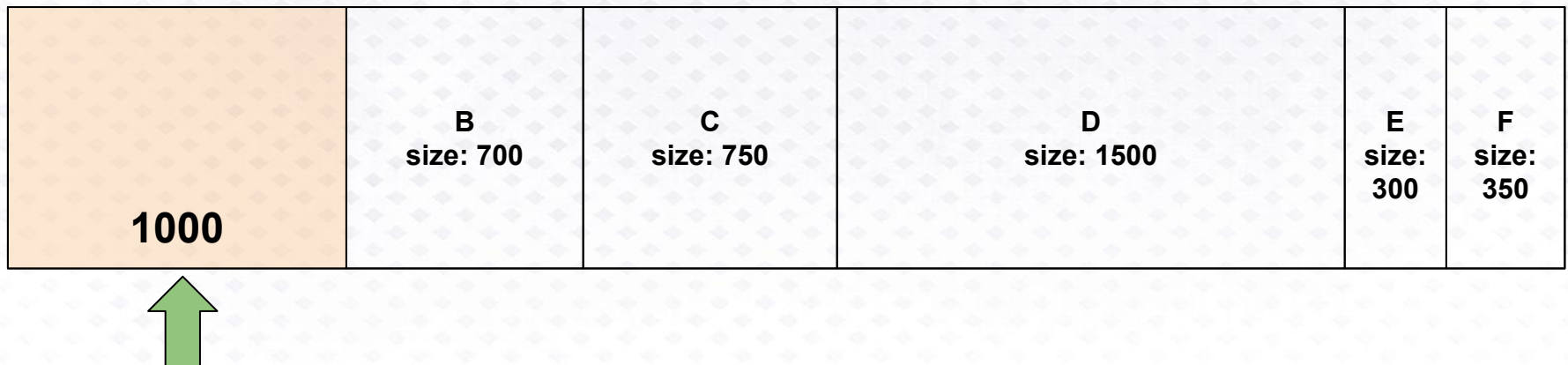


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

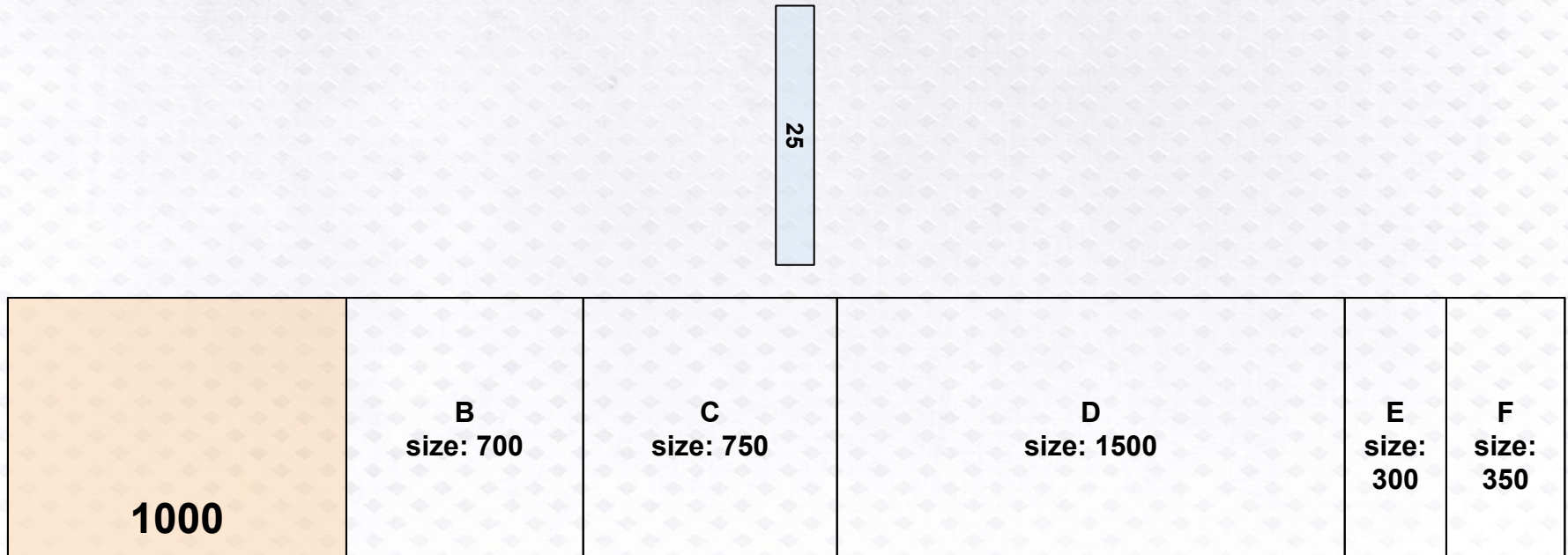


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

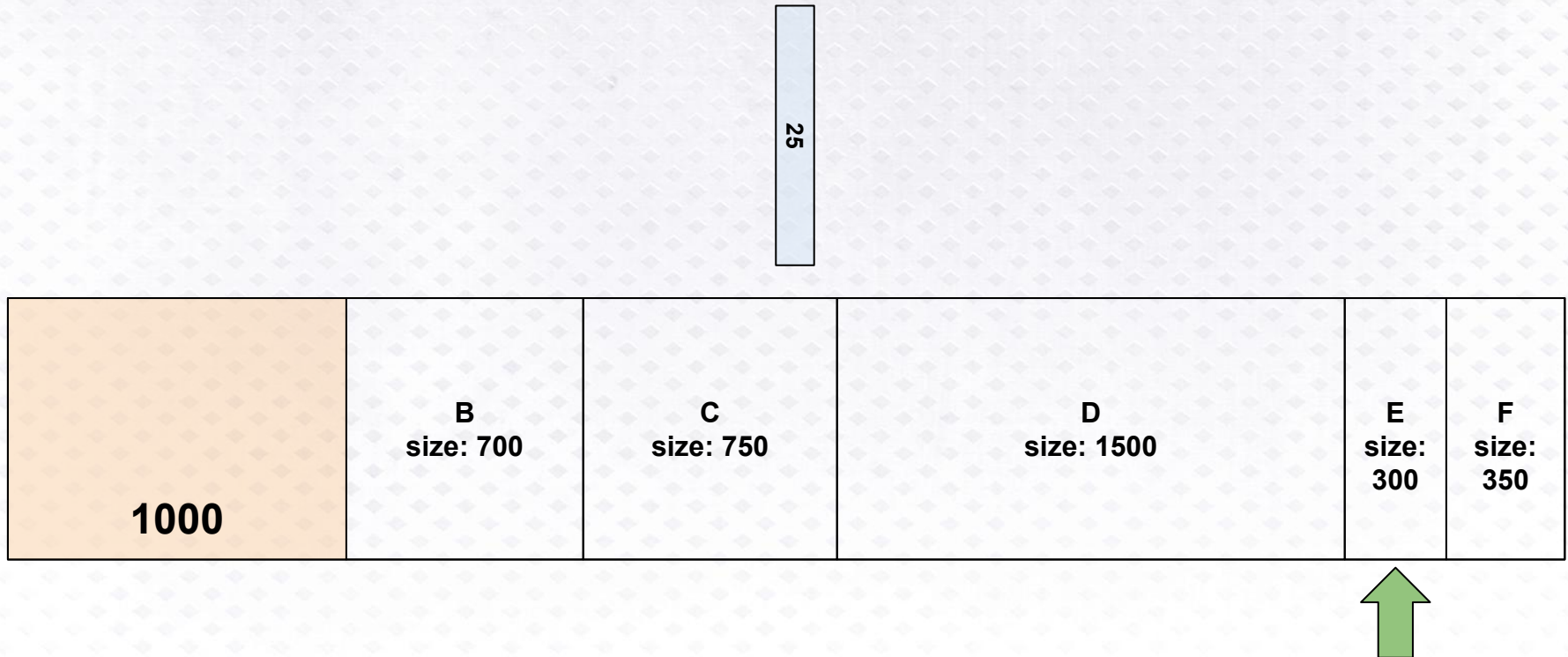


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

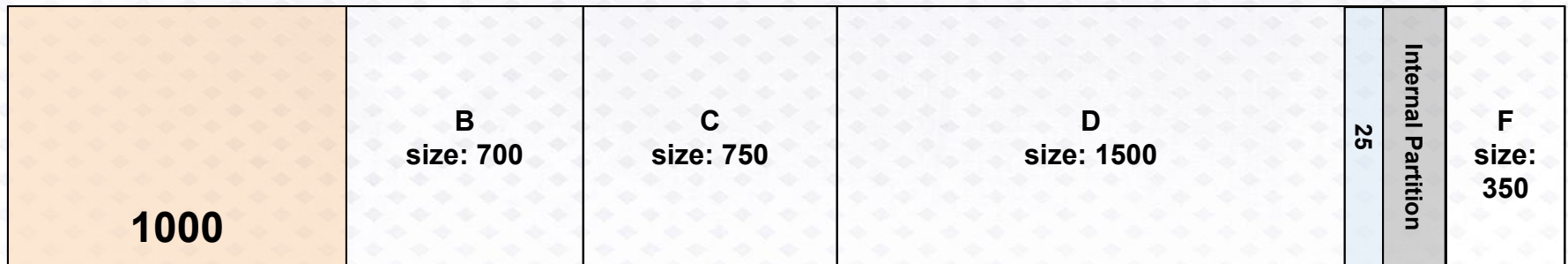


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

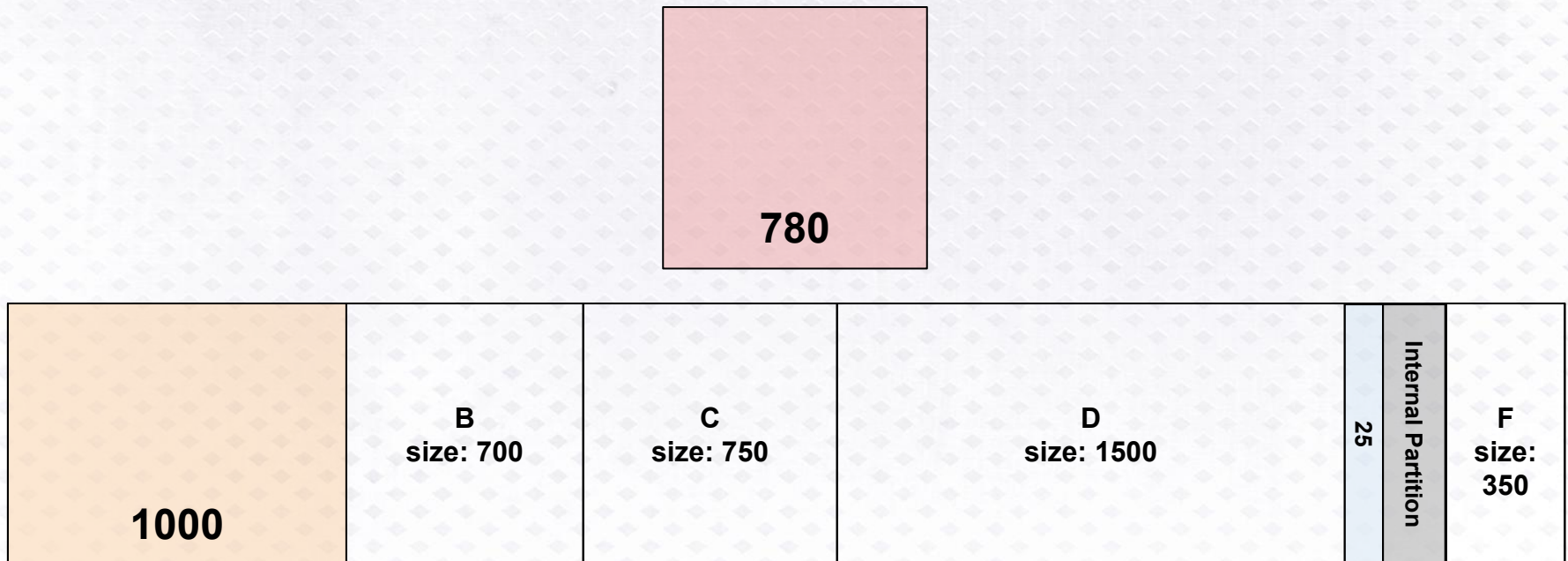


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

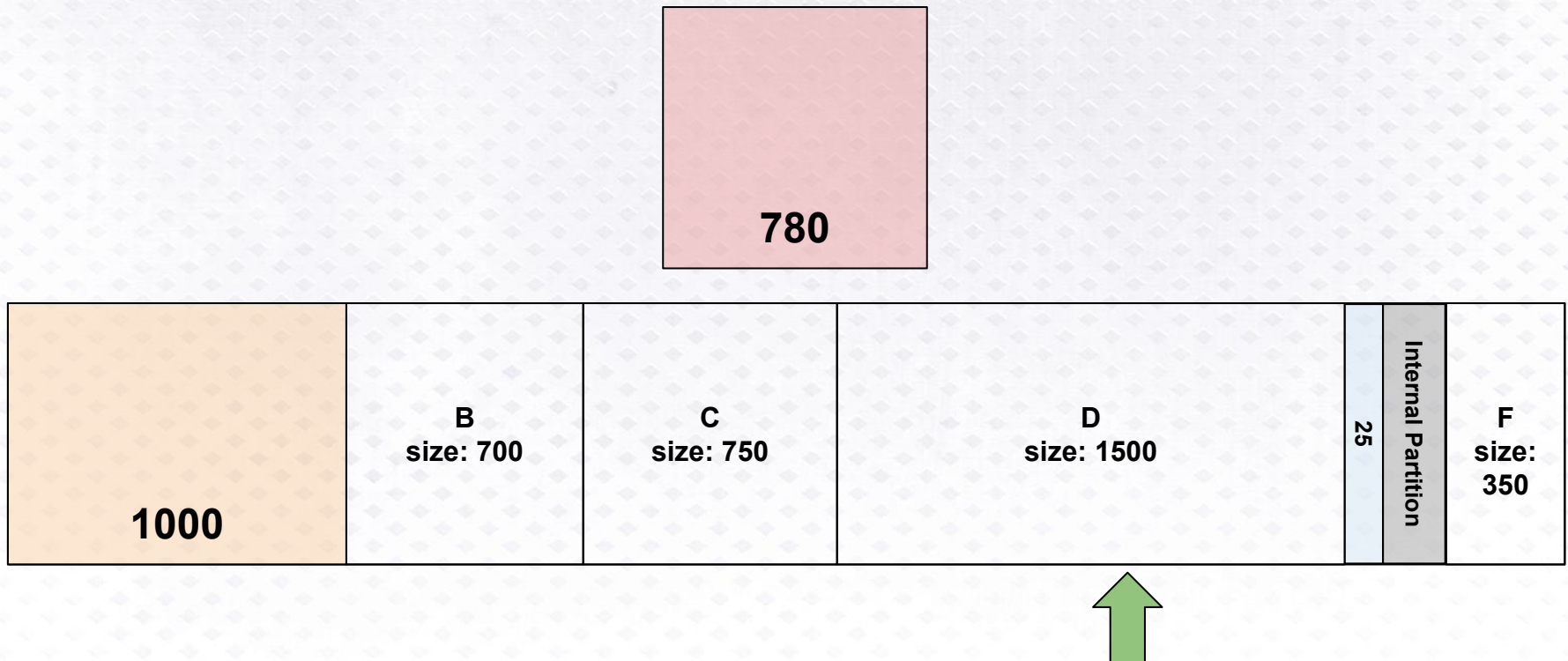


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

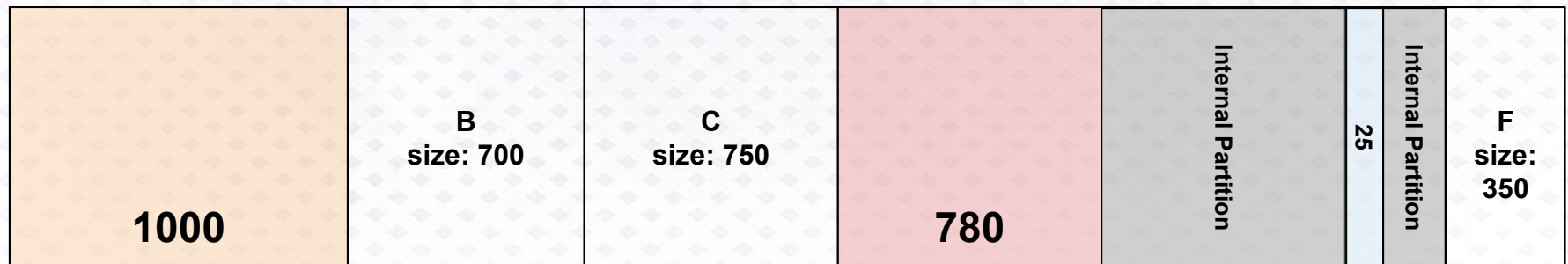


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

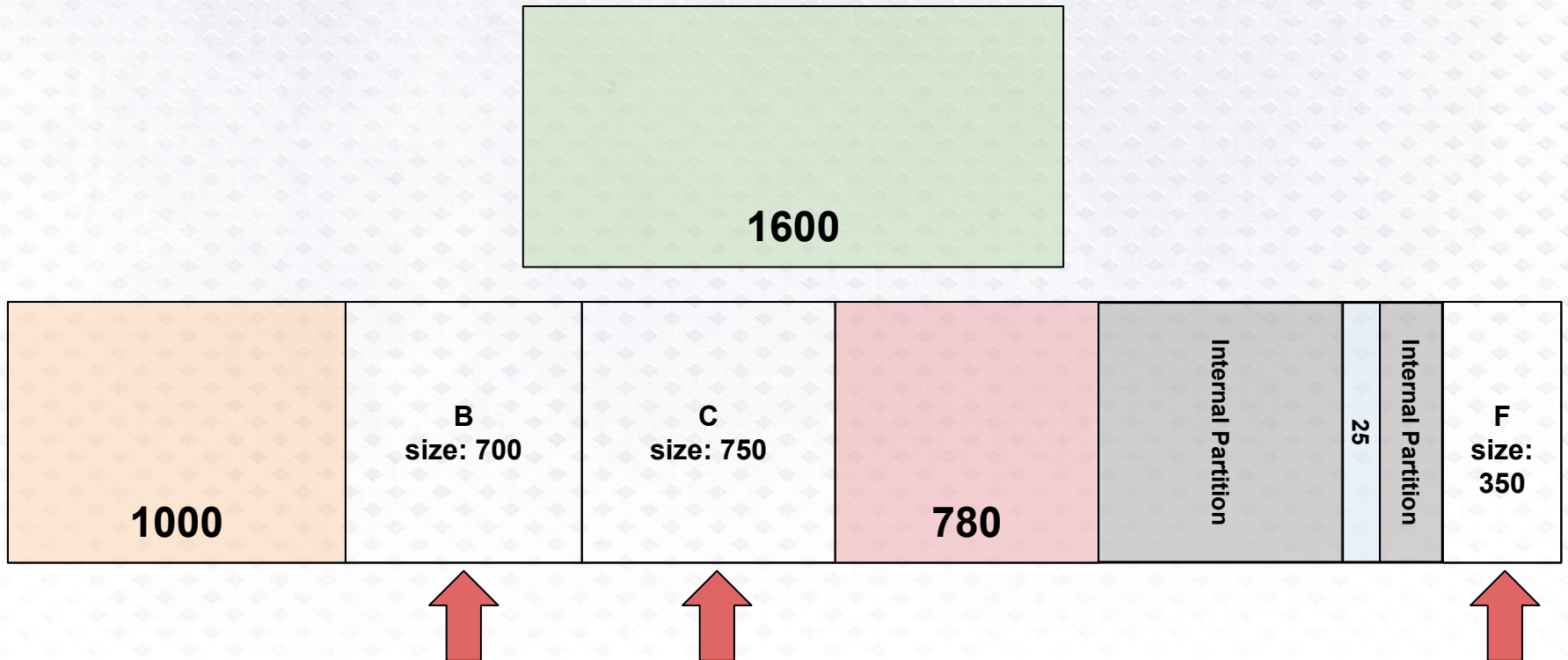


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

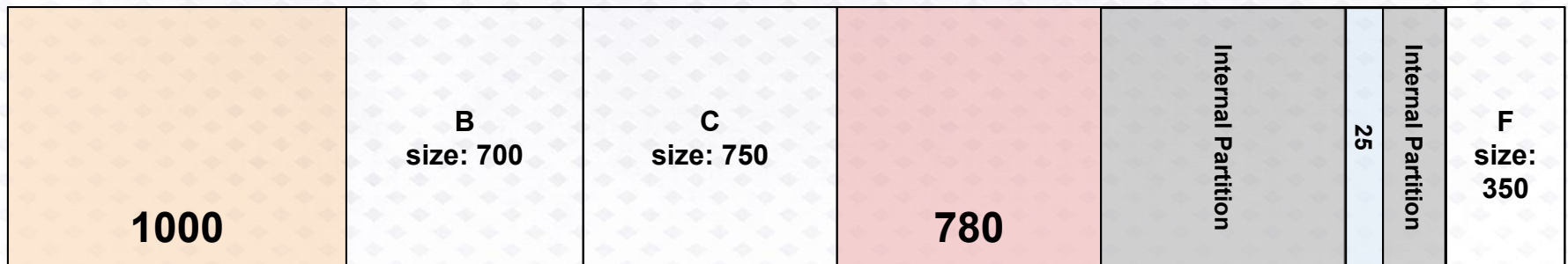
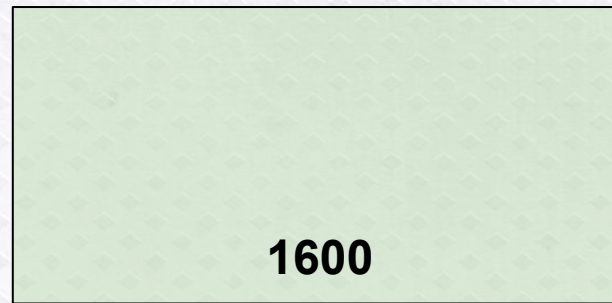


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

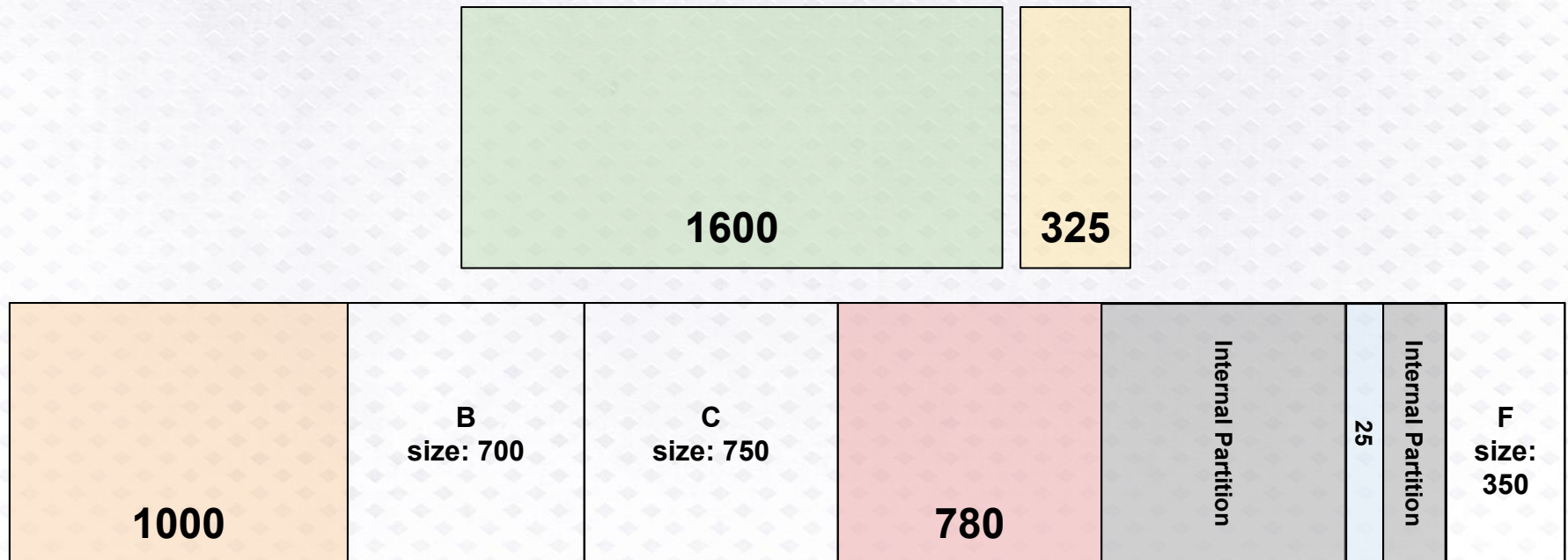


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

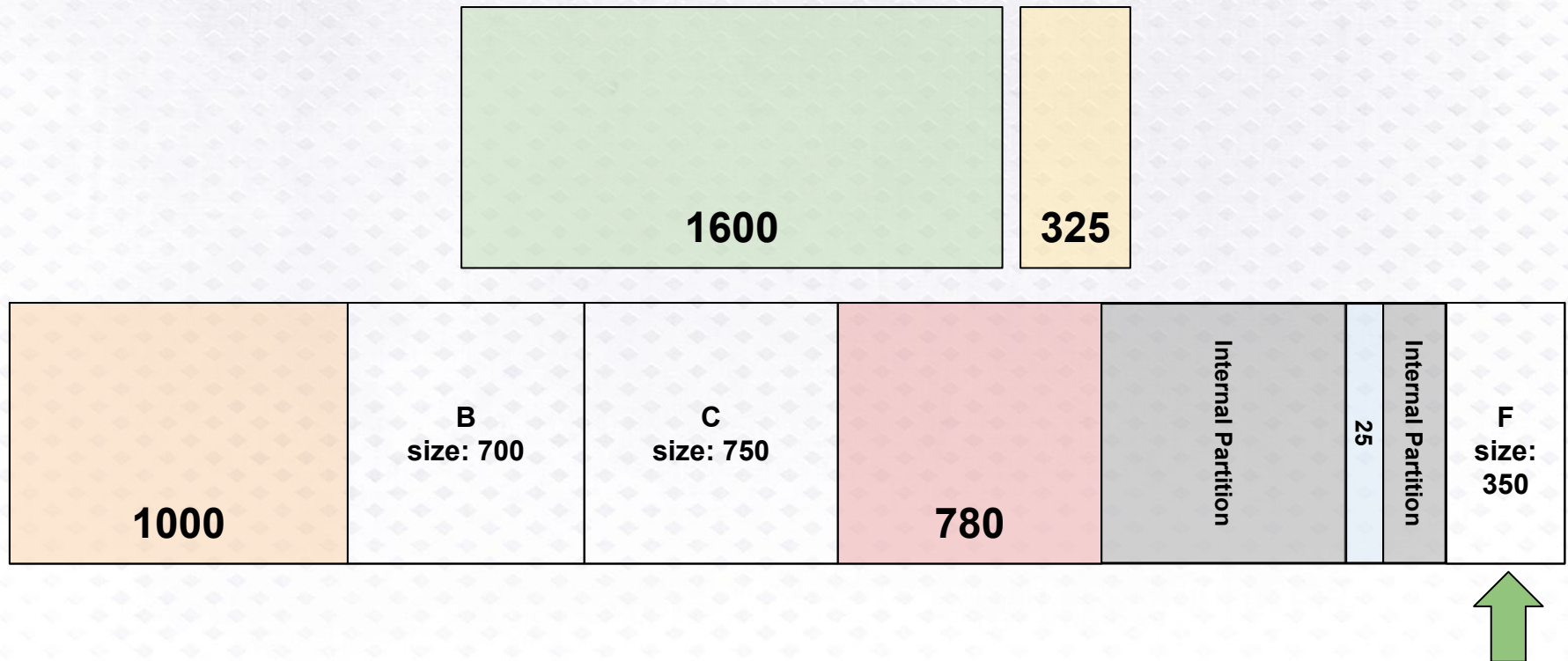
Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



Fixed Partition: Best Fit Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

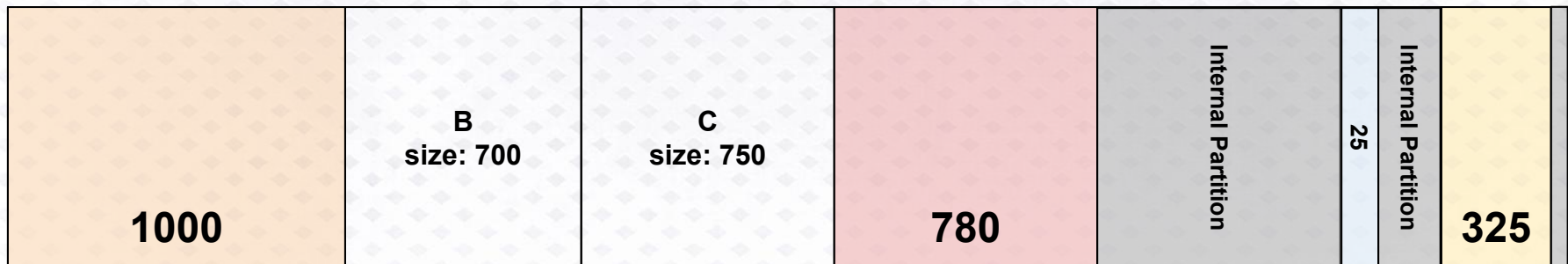
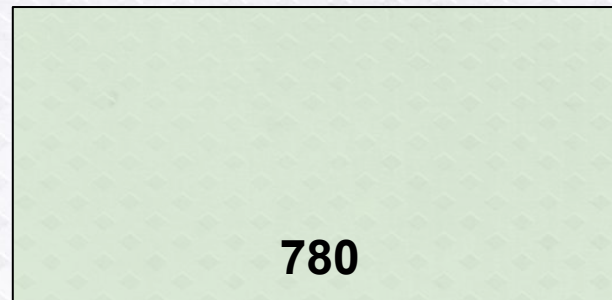


Fixed Partition: Best Fit

Worked Example

Search whole free list to find free space which is closest to that required

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

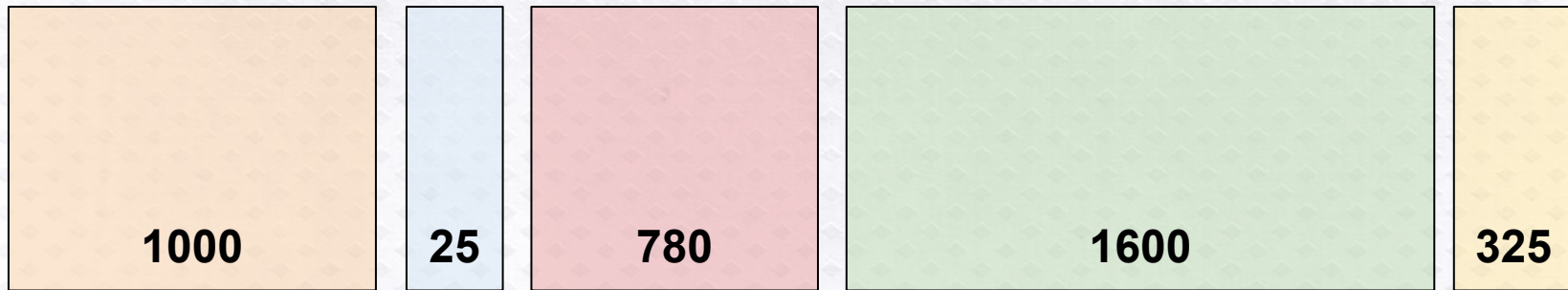


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



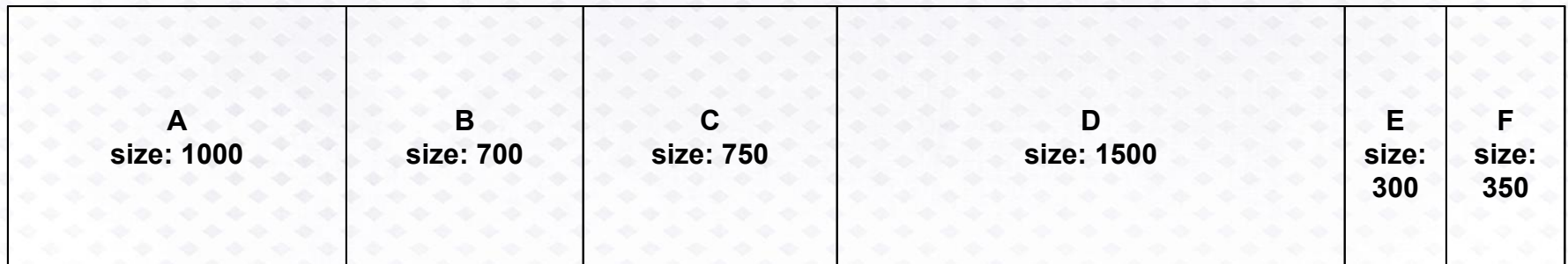
A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

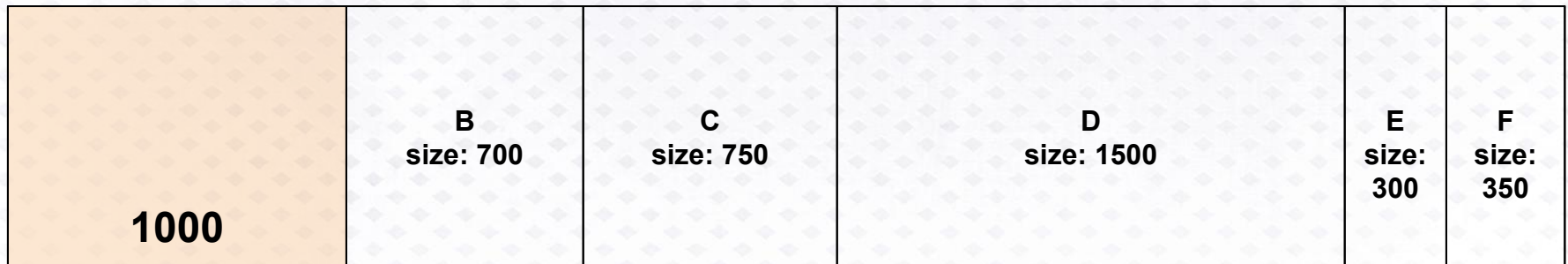
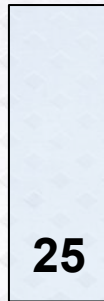
1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

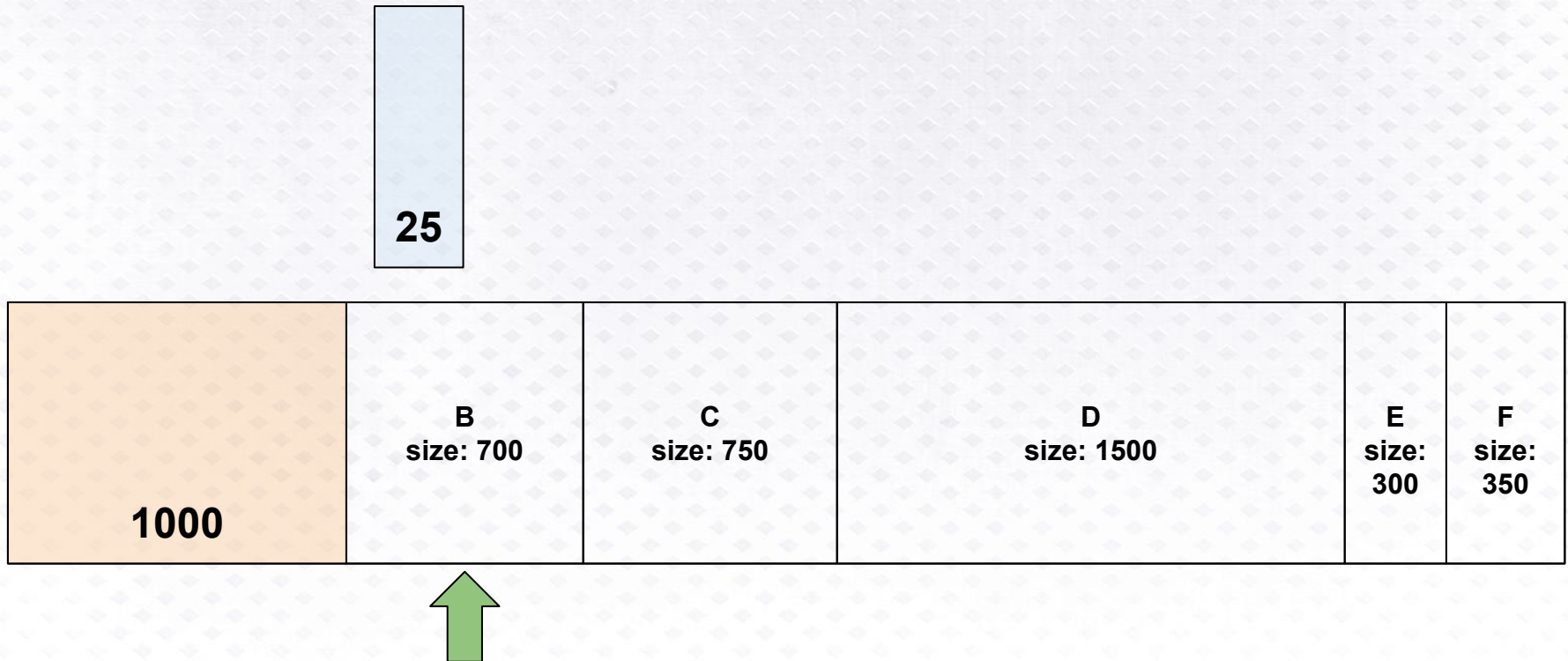


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

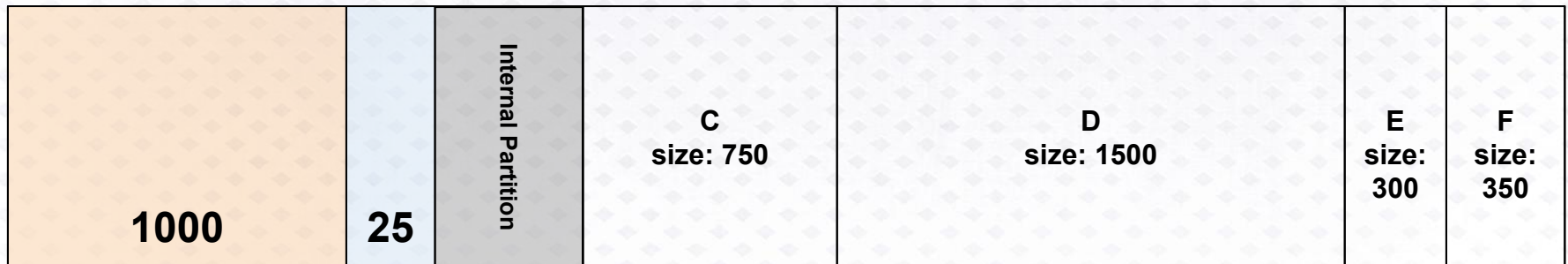


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

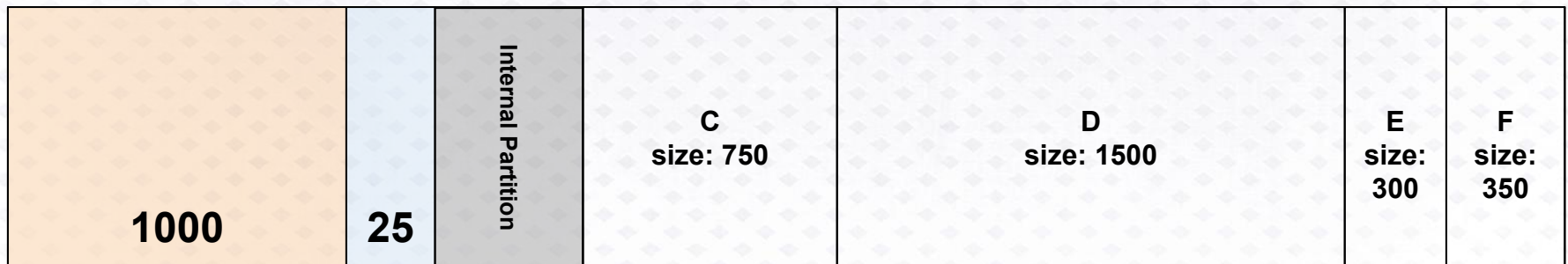
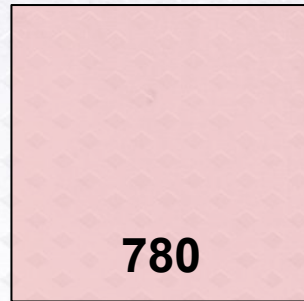


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

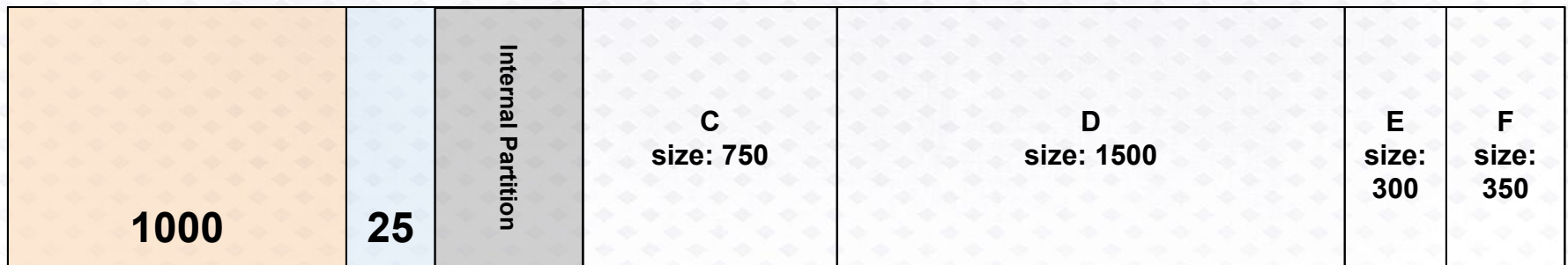
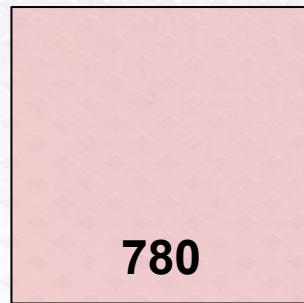


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

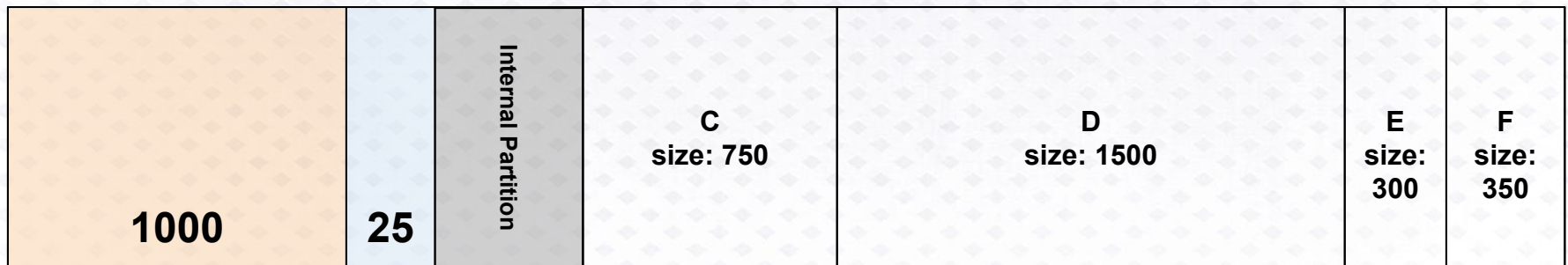
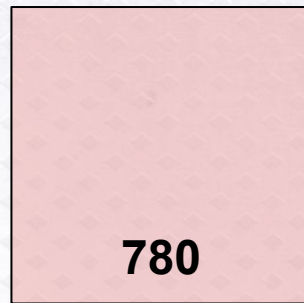


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

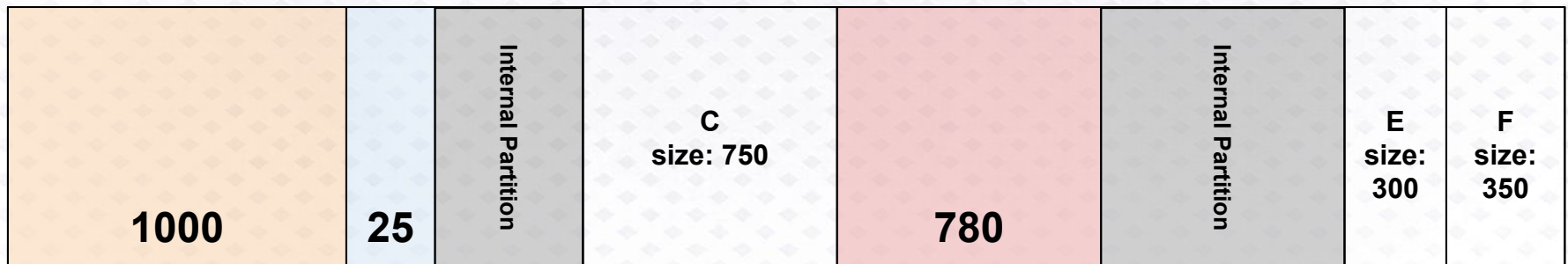


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

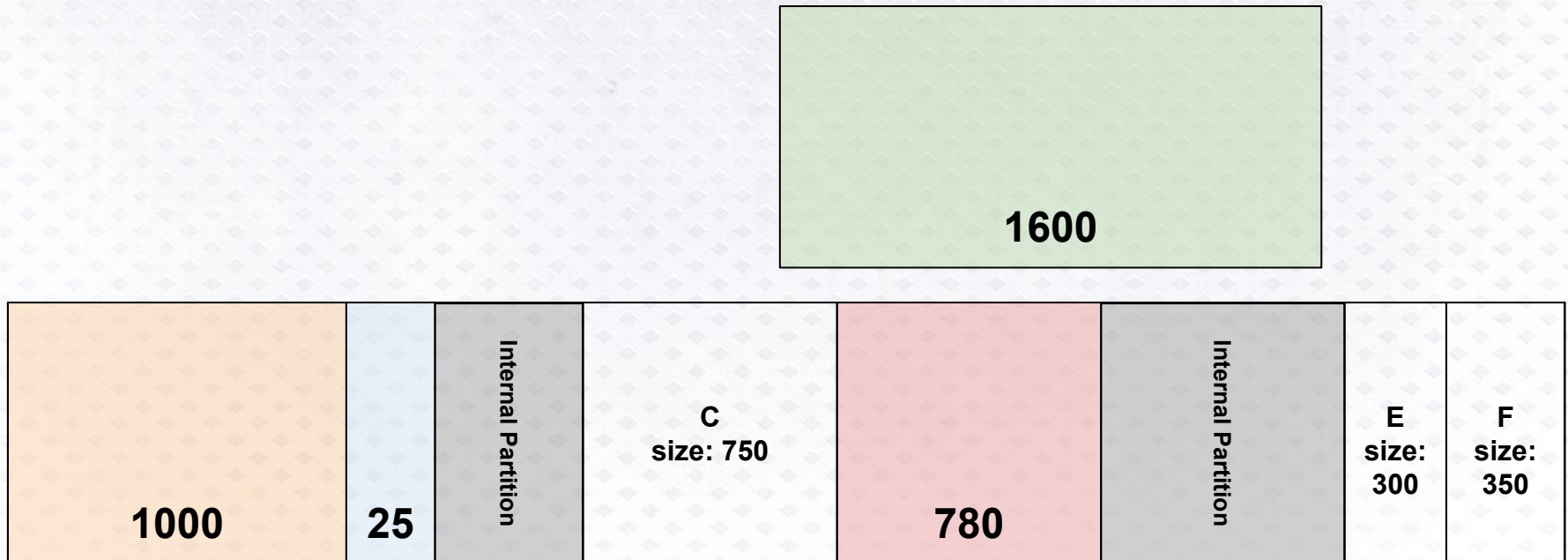


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

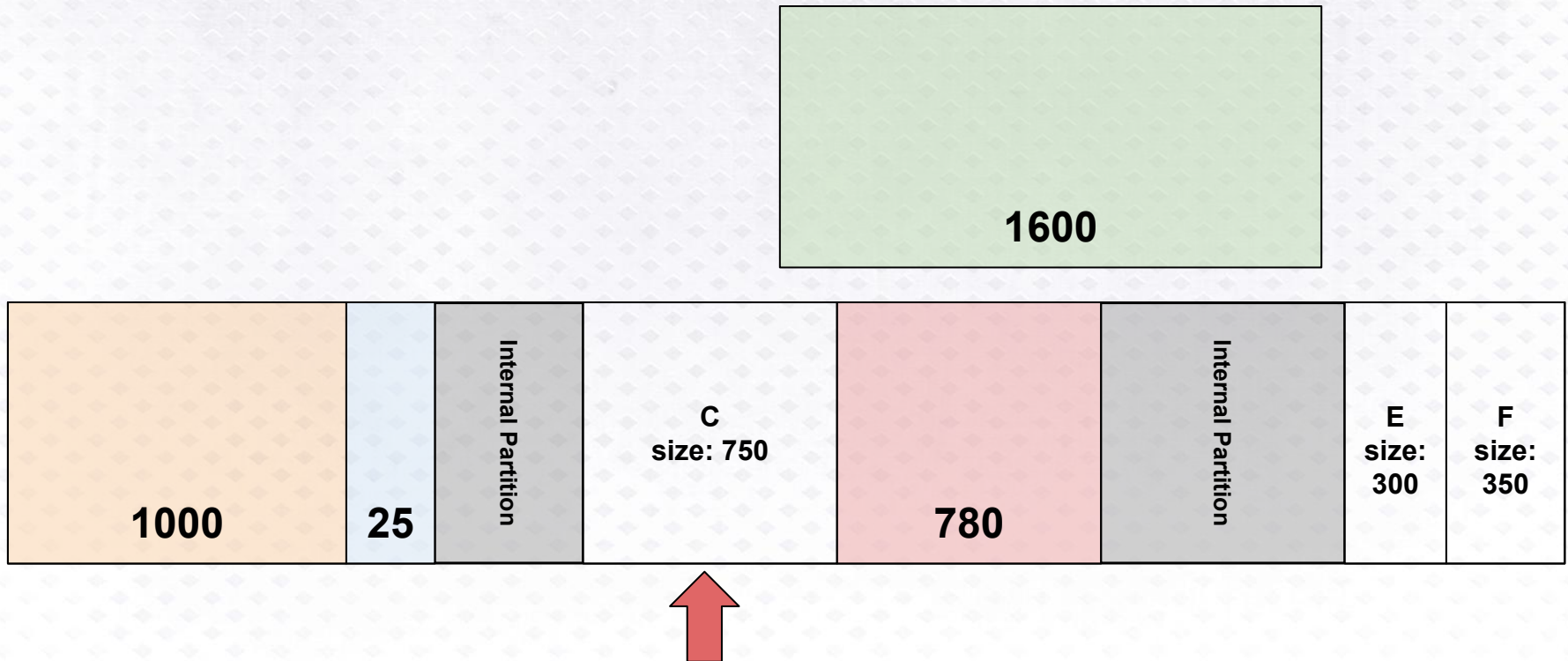


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

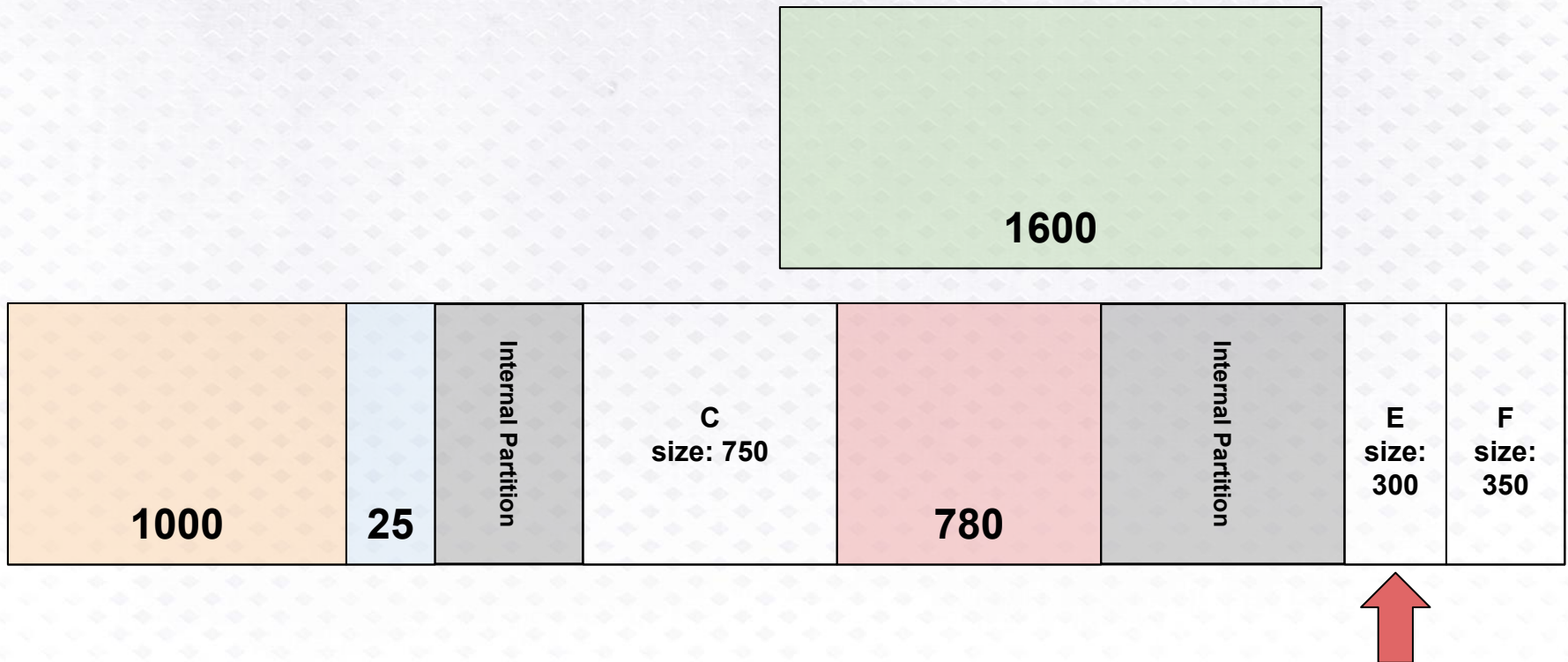


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

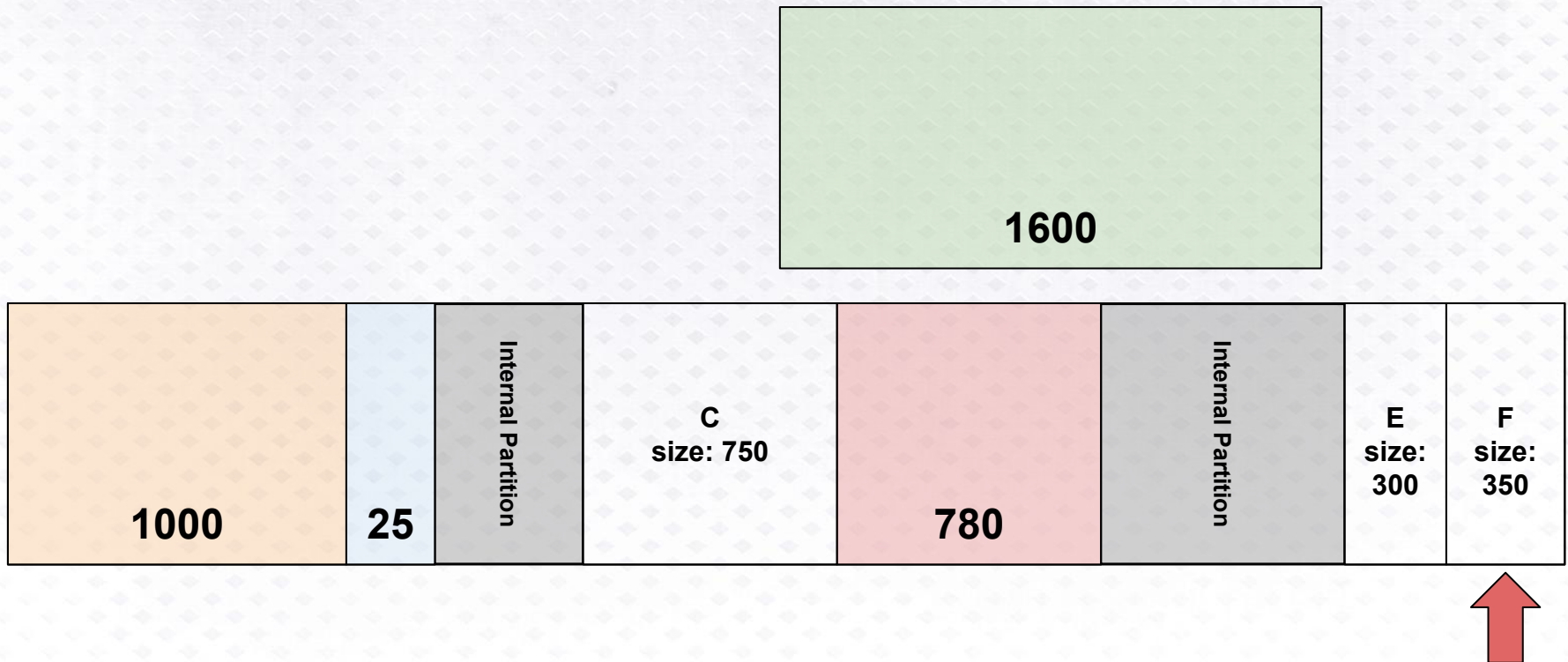


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

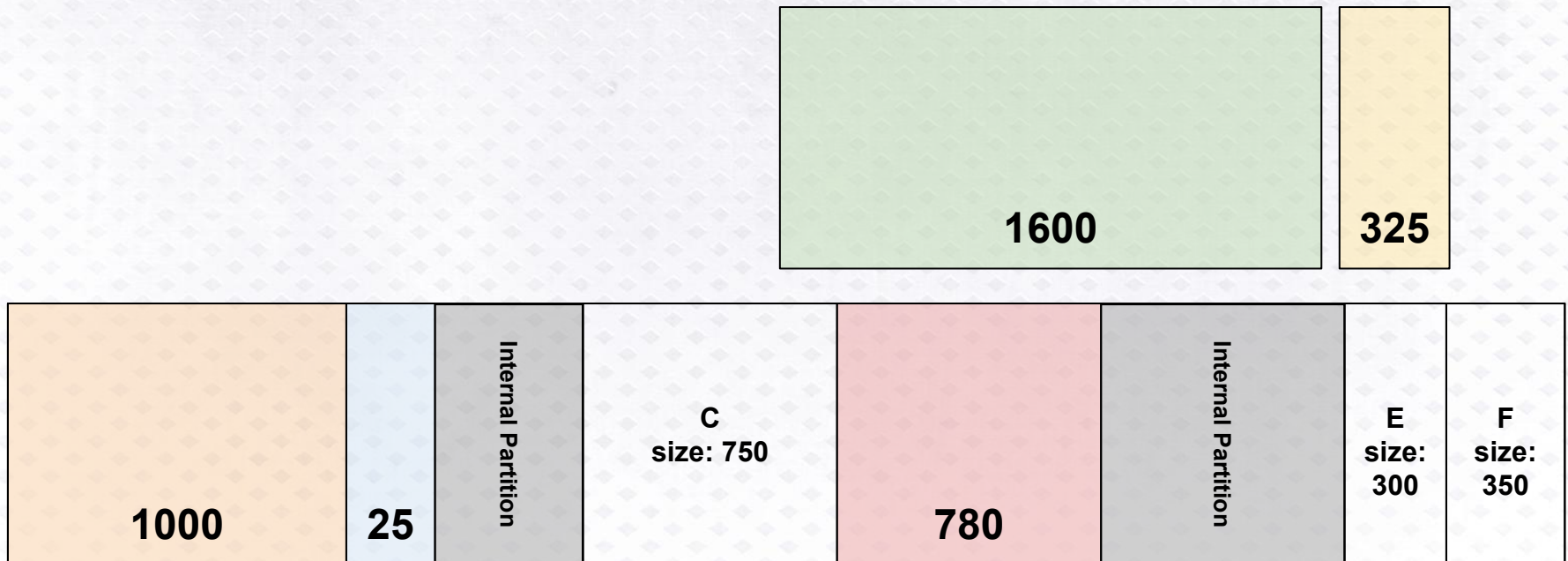


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

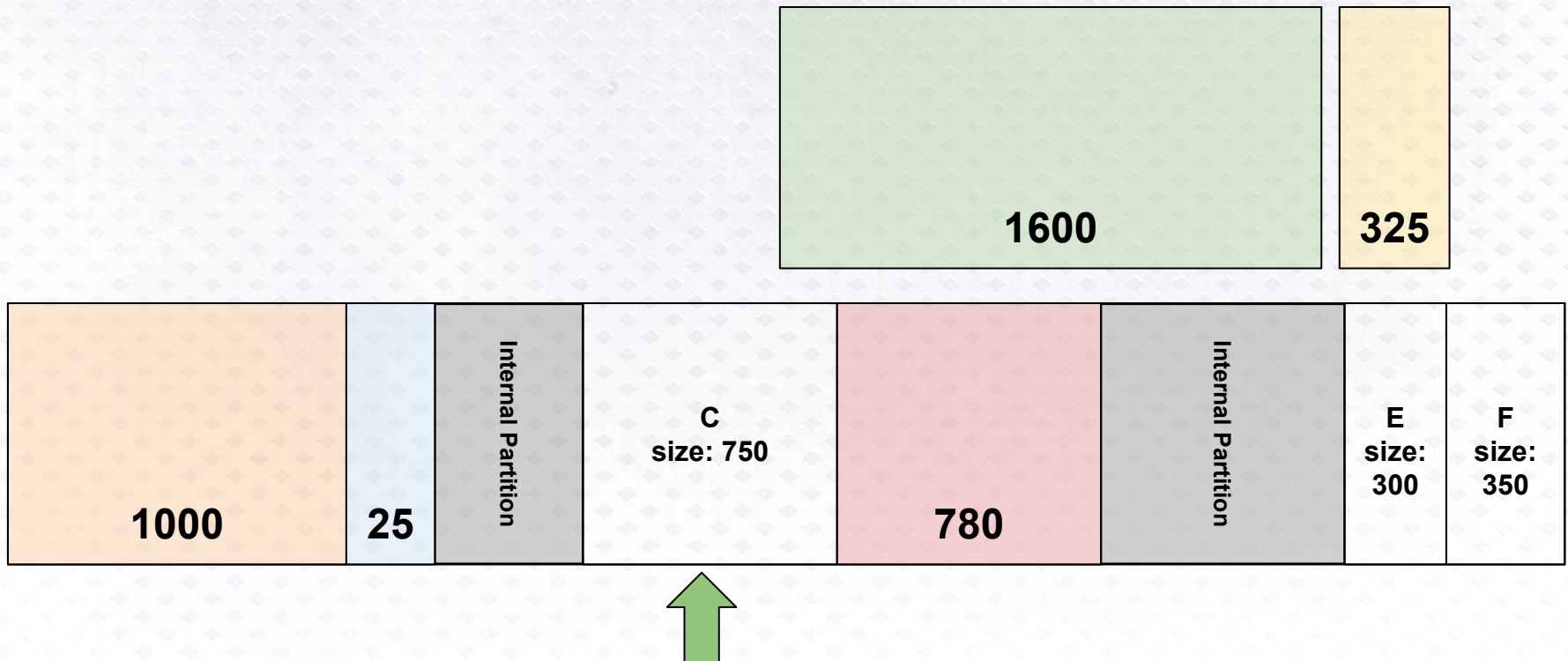


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

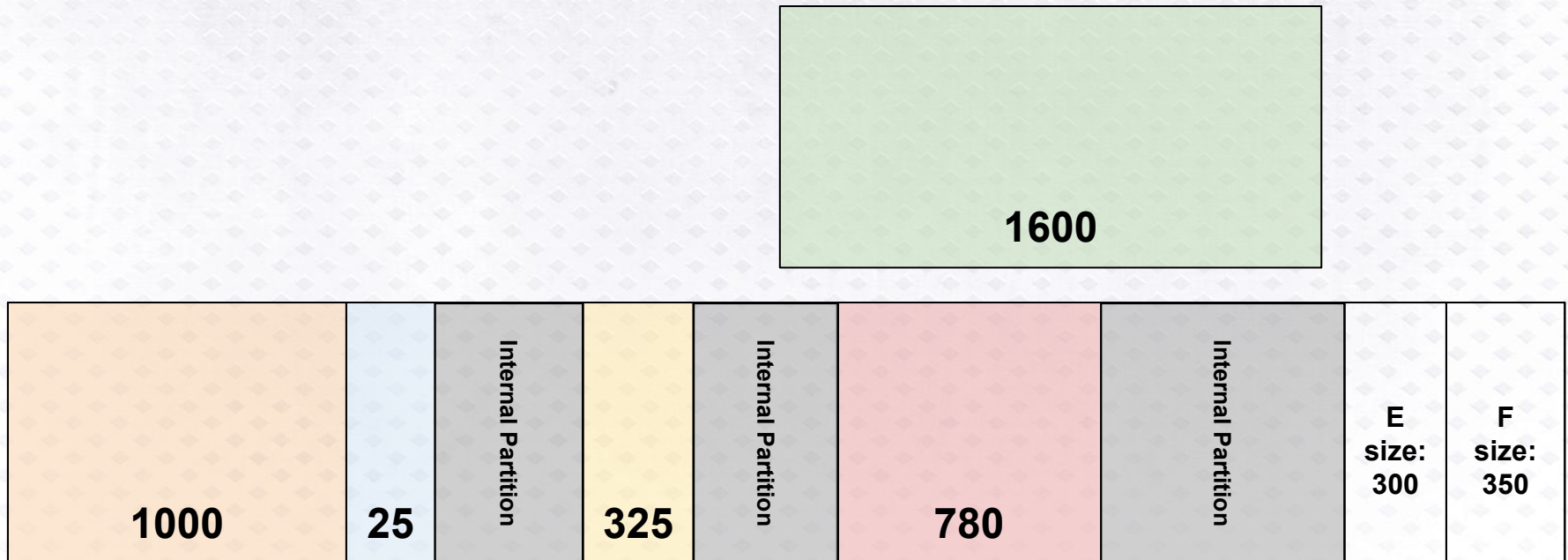


Fixed Partition: First Fit

Worked Example

Search whole free list to find first free space which is large enough

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

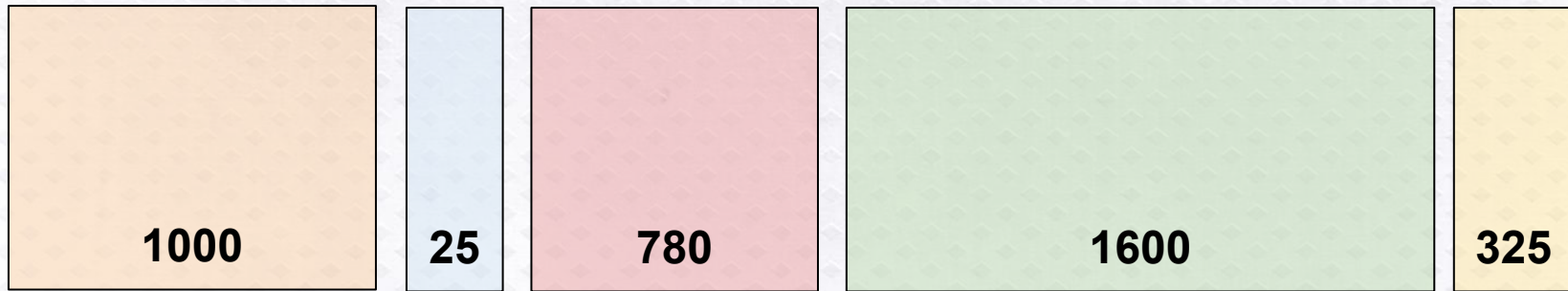


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

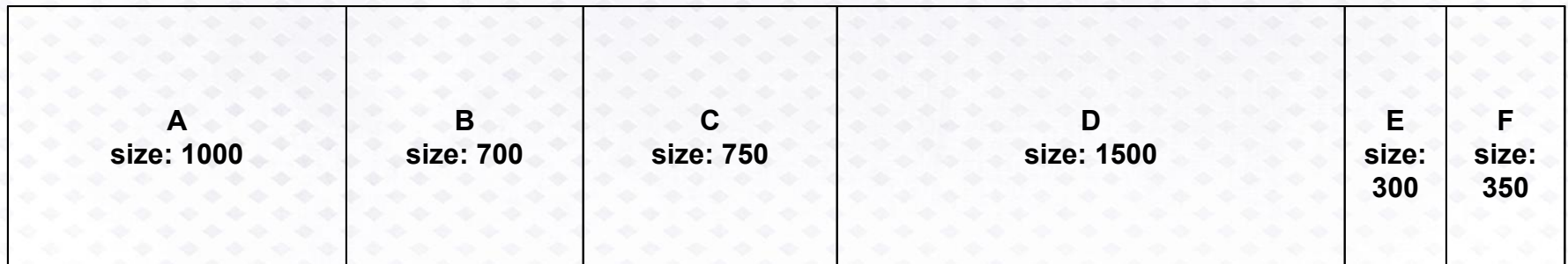
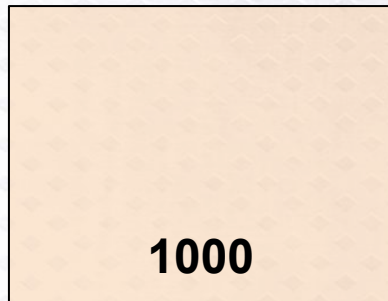
A size: 1000	B size: 700	C size: 750	D size: 1500	E size: 300	F size: 350
------------------------	-----------------------	-----------------------	------------------------	-----------------------	-----------------------

Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

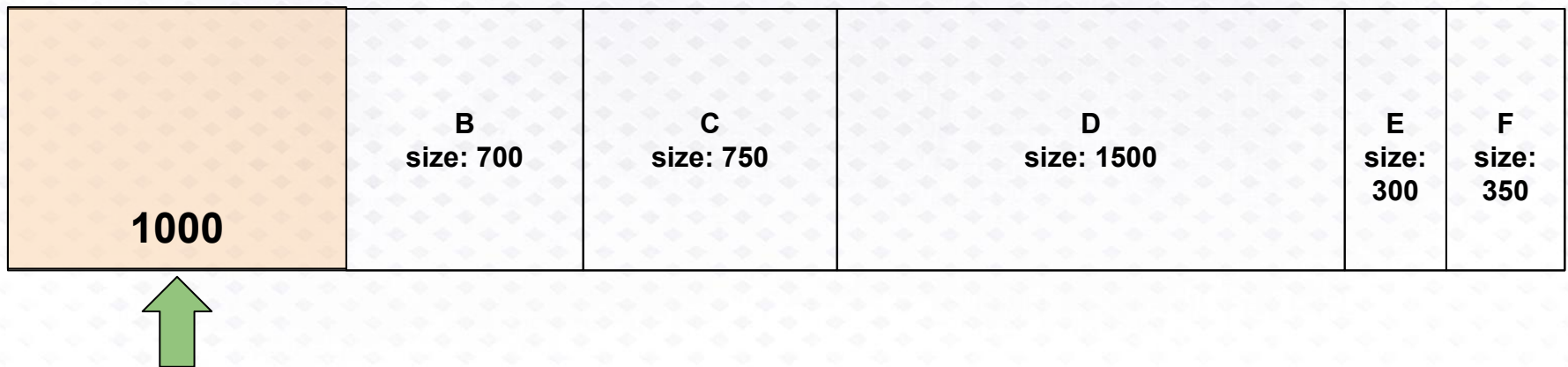


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

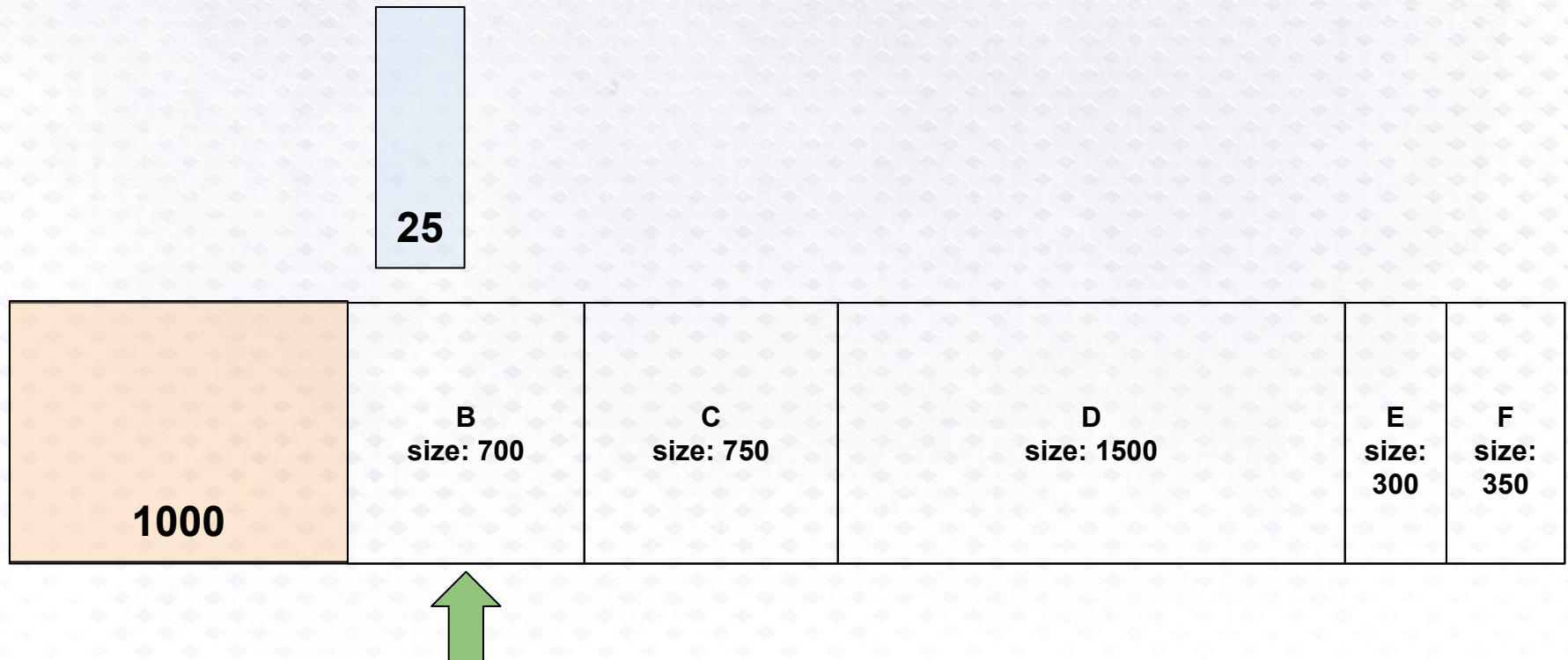


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

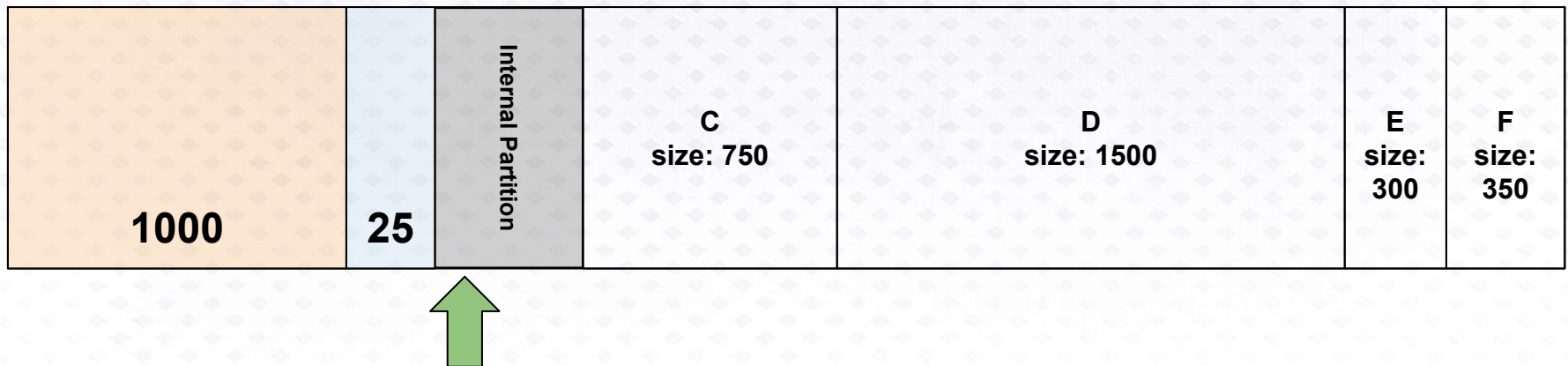


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

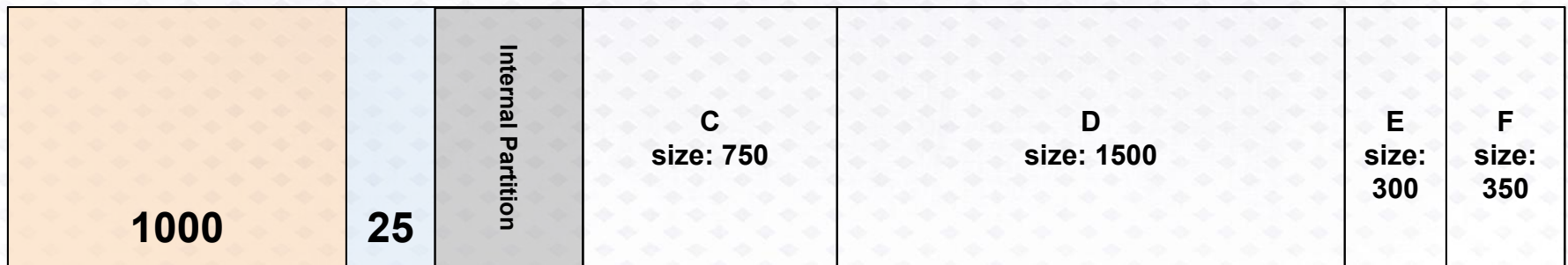
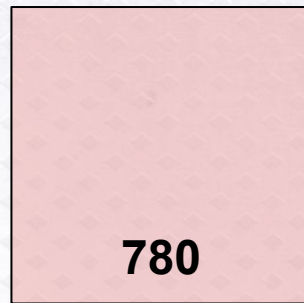


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

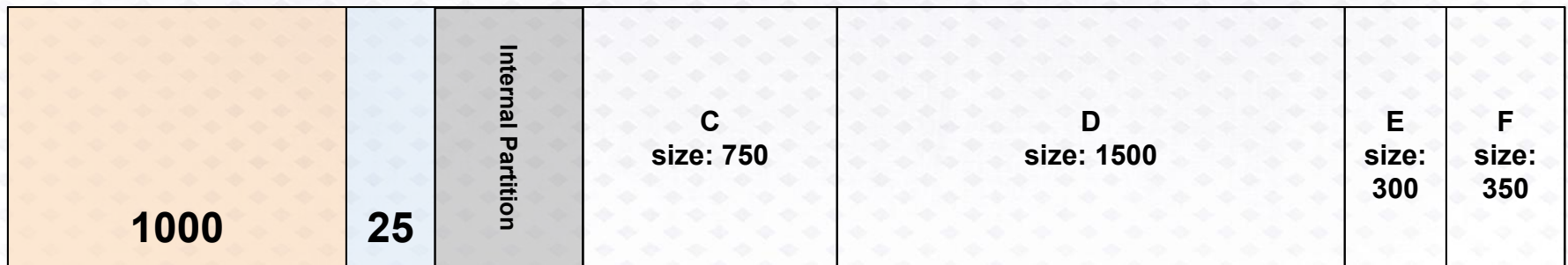
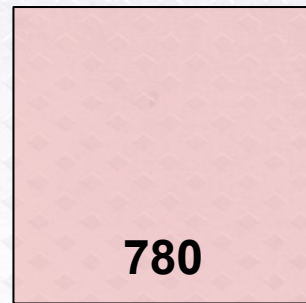


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

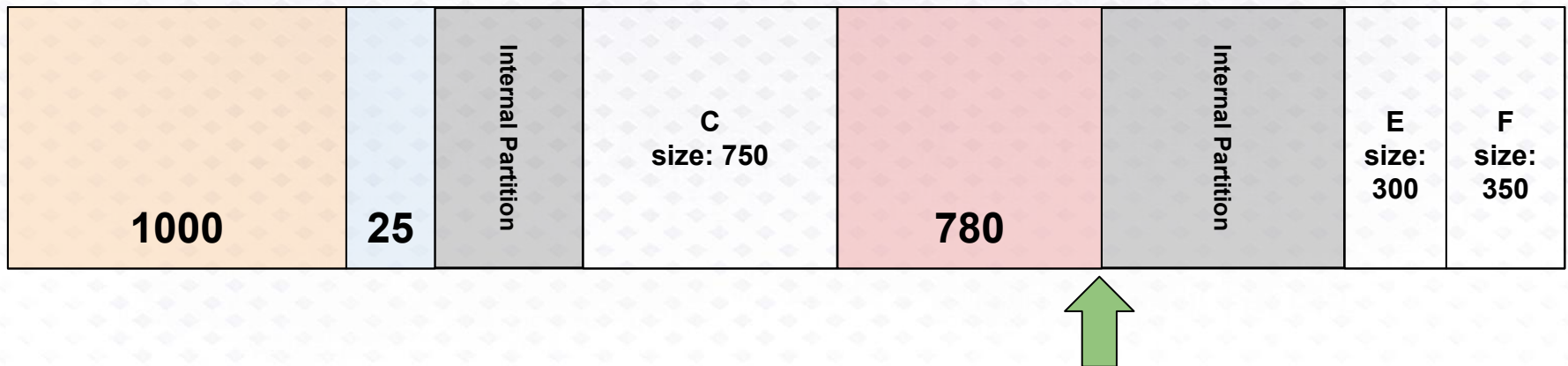


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

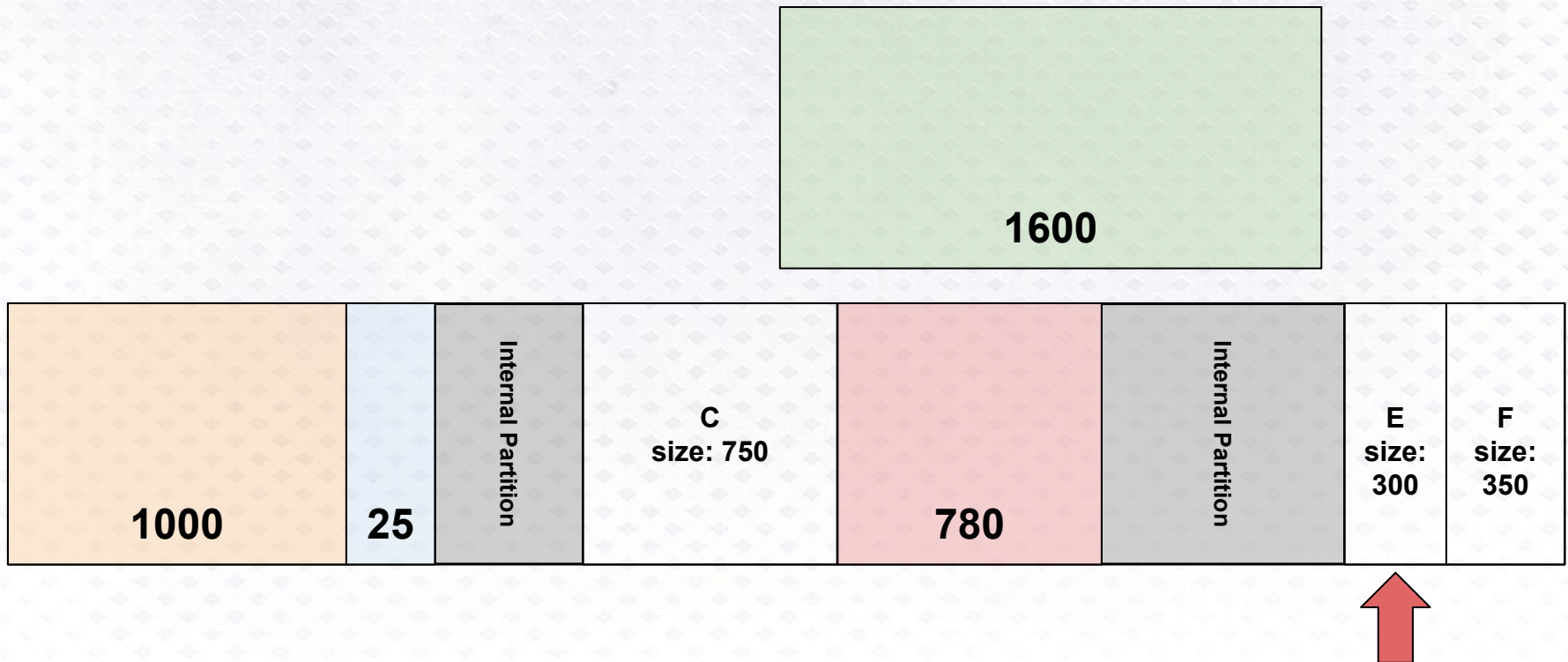


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

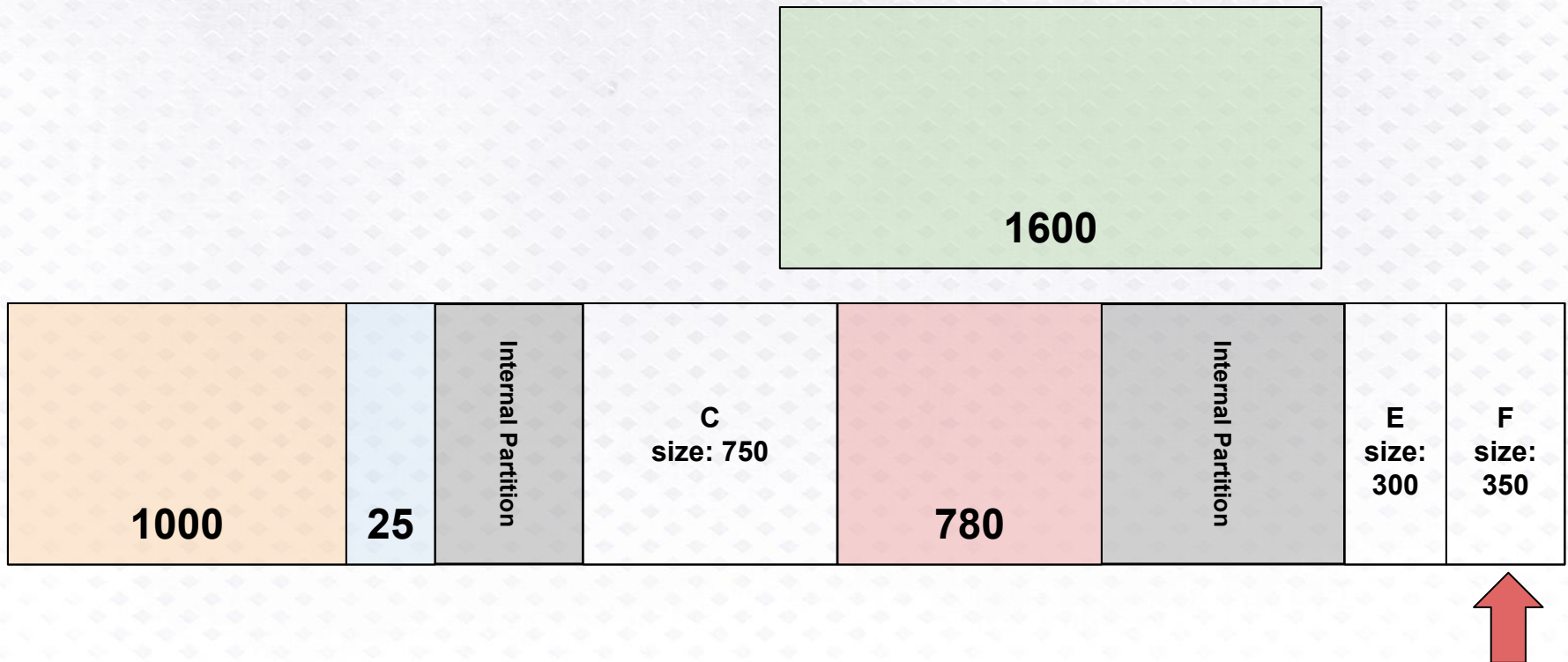


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

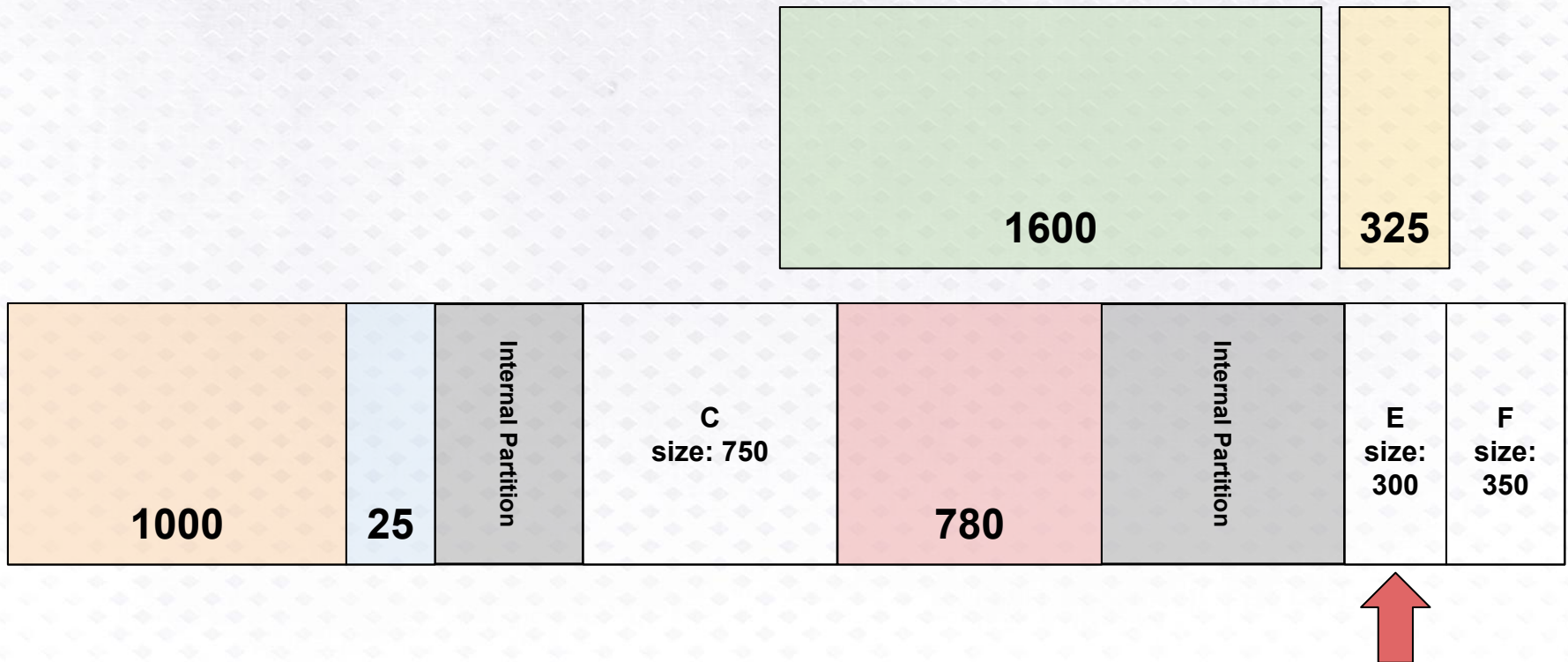


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

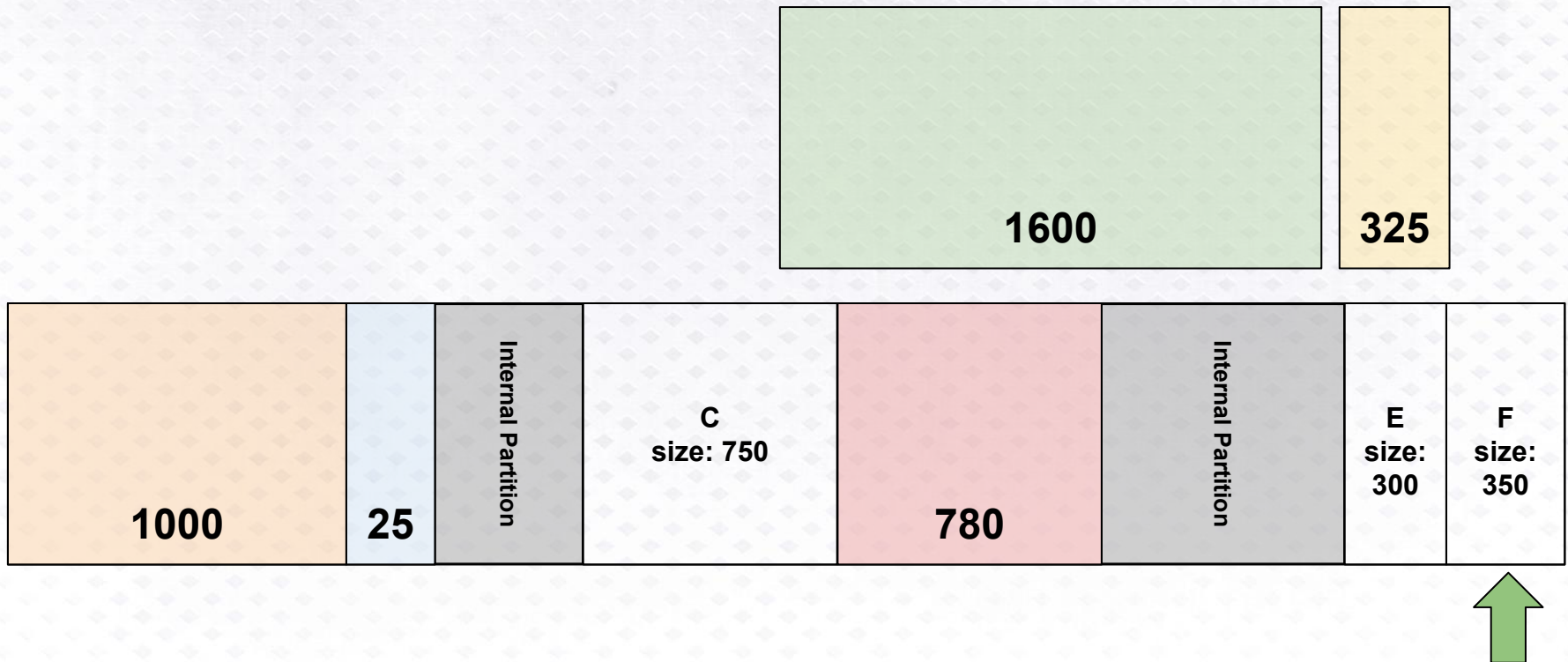


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325

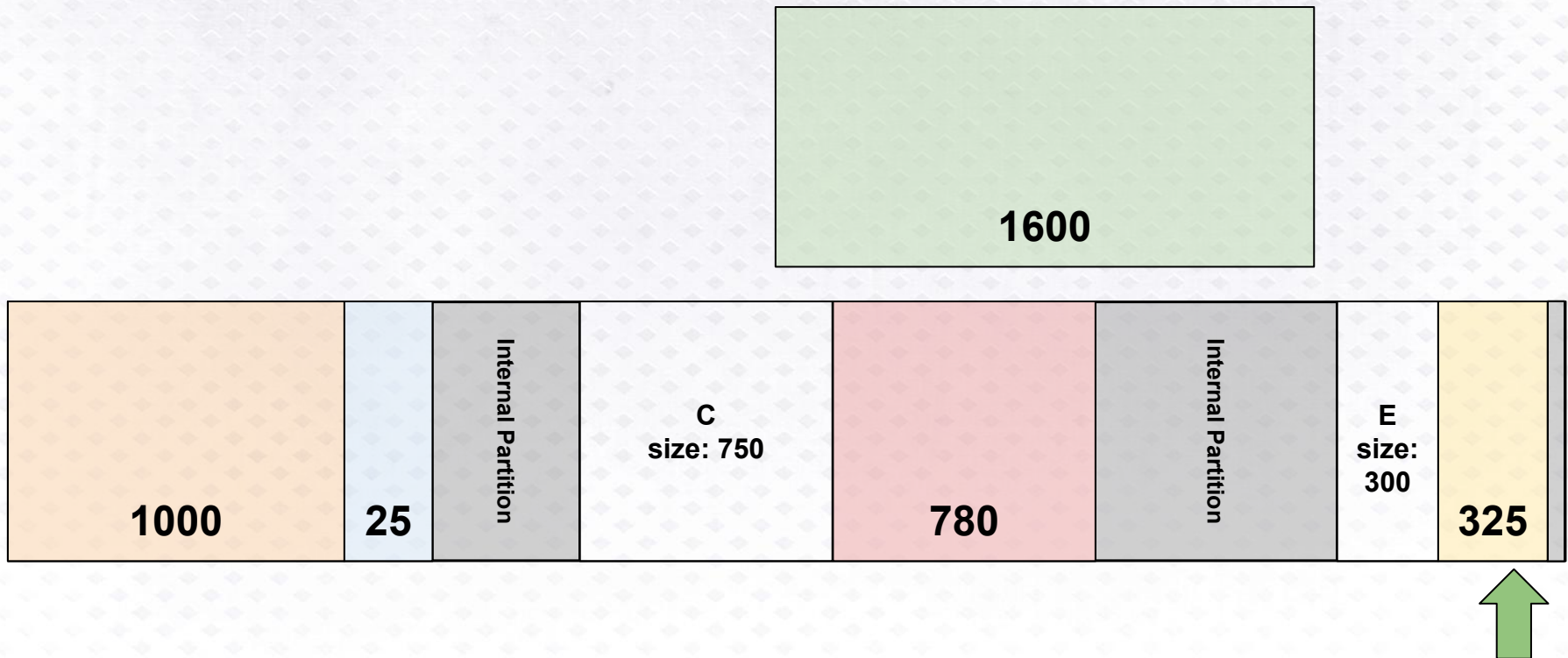


Fixed Partition: Next Fit

Worked Example

Search free list from point where last allocation was made

Requests come in for blocks of the following sizes:
1000, 25, 780, 1600, and 325



Problems of Partition Selection Algorithms

Best Fit

- Fragmentation occurs and long search times are needed

First Fit

- Fragmentation due to smaller partitions at head of the list
- Search time creeps up

Next Fit

- Fragmentation does occur, but worst effects are delayed as more likely distributed throughout memory

Worst Fit

- Long search times
- Reduction of potential large partitions, some programs may be impossible to load

Paged Memory Management

Paged memory technique

A technique in which processes are divided into fixed-size **pages** and stored in memory **frames** when loaded

Frame

A fixed-size portion of *main memory* that holds a process page

Page

A fixed-size portion of a *process* that is stored into a memory frame

We assume that a frame and a page are the same size

Paged Memory Management

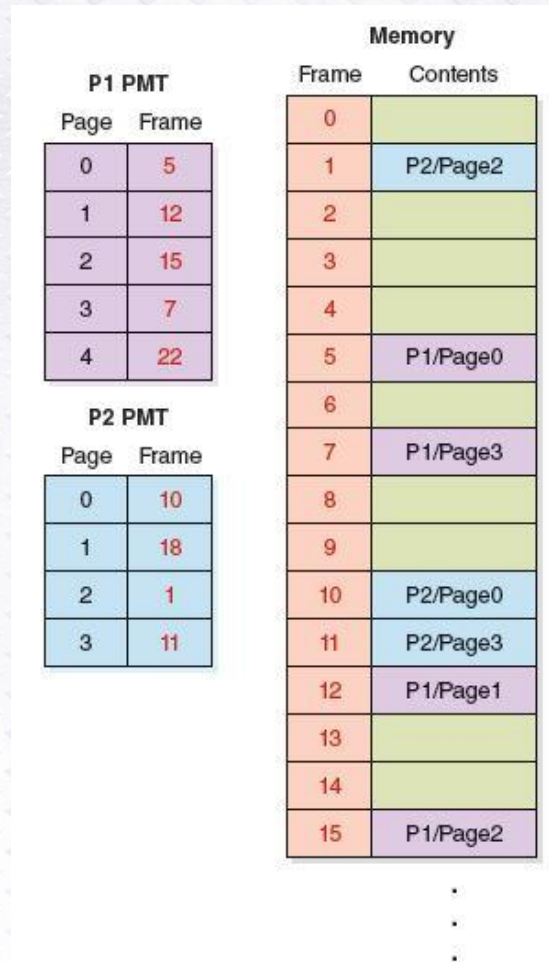


FIGURE 10.8 A paged memory management approach

*Prog. 2,
Page 2*

*Prog. 1,
Page 3*

If Prog. 1 is running and needs logical address 2566, how is the actual address calculated?

Paged Memory Management

Integer logical address is mapped into a
<page number, offset> logical address

Page number

Address divided by the page size (say 1024)

Offset

The remainder of the address divided by the page size

$$2566 \text{ DIV } 1024 = 2$$

$$2566 \text{ MOD } 1024 = 518 \implies \langle 2, 518 \rangle$$

And???

Paged Memory Management

P1 PMT		Memory	
Page	Frame	Frame	Contents
0	5	0	
1	12	1	P2/Page2
2	15	2	
3	7	3	
4	22	4	
		5	P1/Page0
		6	
		7	P1/Page3
		8	
		9	
		10	P2/Page0
		11	P2/Page3
		12	P1/Page1
		13	
		14	
		15	P1/Page2

FIGURE 10.8 A paged memory management approach

This new logical address is mapped to a physical address with the help of a page-map table (PMT)

Every program has a PMT that shows into which frame each page of the program is stored

What is the physical address of <2, 518>?

Paged Memory Management

Demand paging

An extension of paged memory management in which pages are brought into memory on demand

Page swap

The act of bringing in a page from secondary memory, which often causes another page to be written back to secondary memory

Paged Memory Management

Virtual memory

The illusion that there are no restrictions on the size of a program because an entire process doesn't have to be in memory at the same time

Thrashing

Inefficient processing caused by constant page swaps

Relate the expression "all computing is a tradeoff" to this process

Page Replacement Algorithms

- **Optimal**
- **Least Recently Used (LRU)**
- **First-in-First-Out**
- **Clock**

Page Replacement Algorithms

Optimal

- Select replacement page for which time to next reference is the longest
- Impossible as requires perfect knowledge of future events
- Useful as reference point for comparison

Page Replacement Algorithms

Least Recently Used (LRU)

- Replaces page that has not been referenced for the longest time
- By principle the locality should be page least likely to be referenced in the near future
- Difficult to implement
- One approach is to tag each page with the time of last reference, but this has large overhead

Page Replacement Algorithms

First-In-First-Out

- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
- Page that has been in memory the longest is replaced
 - Simplest replacement policy to implement
 - But pages may be needed again very soon

Page Replacement Algorithms

Clock

- Uses an additional bit called a “use bit”
- When page is first loaded or referenced, use bit is set to 1
- When time to replace a page, the OS scans the set flipping all 1's to 0
- First frame encountered with use bit already set to 0 is replaced

Optimal Allocation Worked Example

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2



Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2

2

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3
2	

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3
2	2
	3

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2
2	2	
	3	

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2
2	2	2
	3	3

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1
2	2	2	
	3	3	

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1
2	2	2	2
	3	3	3
			1

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	
	3	3	3	
			1	

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	2
	3	3	3	3
			1	5

Fault count: 1

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2
2	2	2	2	2	
	3	3	3	3	
			1	5	

Fault count: 1

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2
2	2	2	2	2	2
	3	3	3	3	3
			1	5	5

Fault count: 1

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4
2	2	2	2	2	2	
	3	3	3	3	3	
			1	5	5	

Fault count: 1

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4
2	2	2	2	2	2	4
	3	3	3	3	3	3
			1	5	5	5

Fault count: 2

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5
2	2	2	2	2	2	4	
	3	3	3	3	3	3	
			1	5	5	5	

Fault count: 2

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5
2	2	2	2	2	2	4	4
	3	3	3	3	3	3	3
			1	5	5	5	5

Fault count: 2

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3
2	2	2	2	2	2	4	4	
	3	3	3	3	3	3	3	
			1	5	5	5	5	

Fault count: 2

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3
2	2	2	2	2	2	4	4	4
	3	3	3	3	3	3	3	3
			1	5	5	5	5	5

Fault count: 2

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2
2	2	2	2	2	2	4	4	4	
	3	3	3	3	3	3	3	3	
			1	5	5	5	5	5	

Fault count: 2

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2
2	2	2	2	2	2	4	4	4	2
	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5

Fault count: 3

F

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	2	2	4	4	4	2	
	3	3	3	3	3	3	3	3	3	
			1	5	5	5	5	5	5	

Fault count: 3

F

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	2	2	4	4	4	2	2
	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5

Fault count: 3

F

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	
	3	3	3	3	3	3	3	3	3	3	
			1	5	5	5	5	5	5	5	

Fault count: 3

F

F

F

Optimal Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	4	4	4	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
			1	5	5	5	5	5	5	5	5

Fault count: 3

F

F

F

Last Recently Used Allocation Worked Example

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2



Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2

2

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3
2	

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3
2	2
	3

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2
2	2	
	3	

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2
2	2	2
	3	3

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1
2	2	2	
	3	3	

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1
2	2	2	2
	3	3	3
			1

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	
	3	3	3	
			1	

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	2
	3	3	3	5
			1	1

Fault count: 1

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2
2	2	2	2	2	
	3	3	3	5	
			1	1	

Fault count: 1

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2
2	2	2	2	2	2
	3	3	3	5	5
			1	1	1

Fault count: 1

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4
2	2	2	2	2	2	
	3	3	3	5	5	
			1	1	1	

Fault count: 1

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4
2	2	2	2	2	2	2
	3	3	3	5	5	5
			1	1	1	4

Fault count: 2

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5
2	2	2	2	2	2	2	
	3	3	3	5	5	5	
			1	1	1	4	

Fault count: 2

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5
2	2	2	2	2	2	2	2
	3	3	3	5	5	5	5
			1	1	1	4	4

Fault count: 2

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3
2	2	2	2	2	2	2	2	
	3	3	3	5	5	5	5	
			1	1	1	4	4	

Fault count: 2

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3
2	2	2	2	2	2	2	2	3
	3	3	3	5	5	5	5	5
			1	1	1	4	4	4

Fault count: 3

F

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2
2	2	2	2	2	2	2	2	3	
	3	3	3	5	5	5	5	5	
			1	1	1	4	4	4	

Fault count: 3

F

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2
2	2	2	2	2	2	2	2	3	3
	3	3	3	5	5	5	5	5	5
			1	1	1	4	4	4	2
Fault count: 4				F		F		F	F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	2	2	2	2	3	3	
	3	3	3	5	5	5	5	5	5	
			1	1	1	4	4	4	2	
Fault count: 4				F		F		F	F	

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	2	2	2	2	3	3	3
	3	3	3	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2

Fault count: 4

F

F

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	
	3	3	3	5	5	5	5	5	5	5	
			1	1	1	4	4	4	2	2	

Fault count: 4

F

F

F

F

Least Recently Used Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2
				F		F		F	F		

Fault count: 4

First-In-First-Out Allocation Worked Example

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2



First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2

2

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3
2	

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3
2	2
	3

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2
2	2	
	3	

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2
2	2	2
	3	3

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1
2	2	2	
	3	3	

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1
2	2	2	2
	3	3	3
			1

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	
	3	3	3	
			1	

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	5
	3	3	3	3
			1	1

Fault count: 1

F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2
2	2	2	2	5	
	3	3	3	3	
			1	1	

Fault count: 1

F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2
2	2	2	2	5	5
	3	3	3	3	2
			1	1	1

Fault count: 2

F **F**

First-In-First-Out Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4
2	2	2	2	5	5	
	3	3	3	3	2	
			1	1	1	

Fault count: 2

F **F**

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4
2	2	2	2	5	5	5
	3	3	3	3	2	2
			1	1	1	4

Fault count: 3

F F F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5
2	2	2	2	5	5	5	
	3	3	3	3	2	2	
			1	1	1	4	

Fault count: 3

F F F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5
2	2	2	2	5	5	5	5
	3	3	3	3	2	2	2
			1	1	1	4	4

Fault count: 3

F F F

First-In-First-Out Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3
2	2	2	2	5	5	5	5	
	3	3	3	3	2	2	2	
			1	1	1	4	4	

Fault count: 3

F F F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3
2	2	2	2	5	5	5	5	3
	3	3	3	3	2	2	2	2
			1	1	1	4	4	4
Fault count: 4				F	F	F		F

First-In-First-Out Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2
2	2	2	2	5	5	5	5	3	
	3	3	3	3	2	2	2	2	
			1	1	1	4	4	4	

Fault count: 4

F F F F

First-In-First-Out Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2
2	2	2	2	5	5	5	5	3	3
	3	3	3	3	2	2	2	2	2
			1	1	1	4	4	4	4

Fault count: 4

F F F F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	5	5	5	5	3	3	
	3	3	3	3	2	2	2	2	2	
			1	1	1	4	4	4	4	

Fault count: 4

F

F

F

F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	5	5	5	5	3	3	3
	3	3	3	3	2	2	2	2	2	5
			1	1	1	4	4	4	4	4
Fault count: 5				F	F	F		F		F

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	
	3	3	3	3	2	2	2	2	2	5	
			1	1	1	4	4	4	4	4	
Fault count: 5				F	F	F		F		F	

First-In-First-Out Allocation

Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2
Fault count: 6				F	F	F		F		F	F

Clock Allocation Worked Example

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2



Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2

2

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2

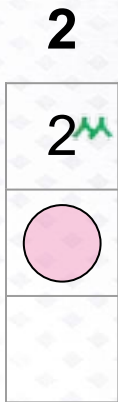


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

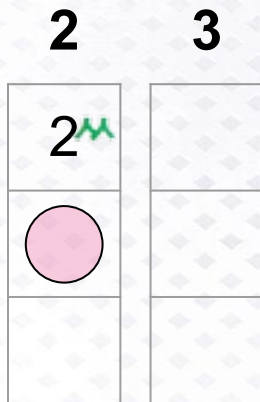


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

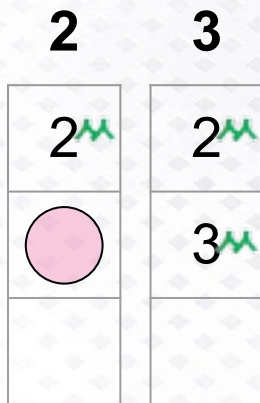


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

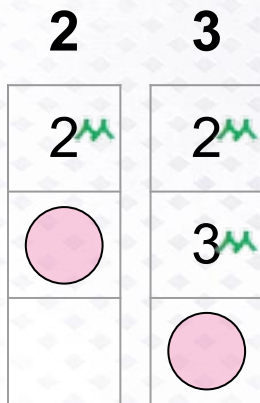


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

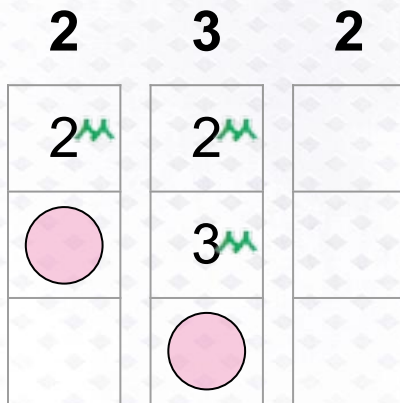


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

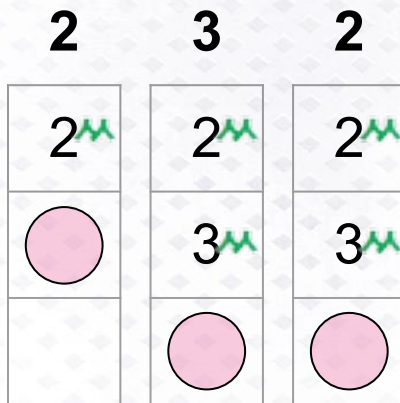


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

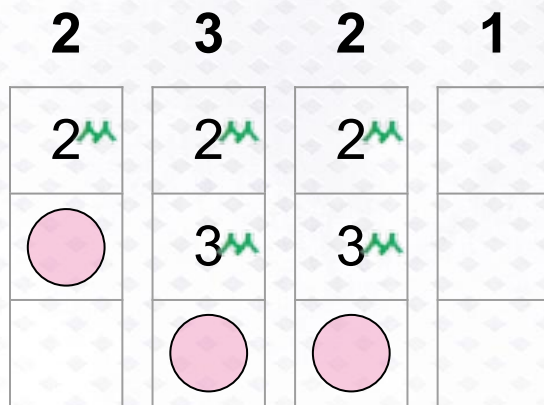


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

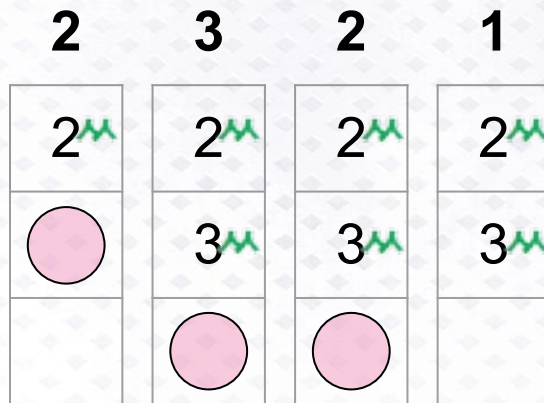


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

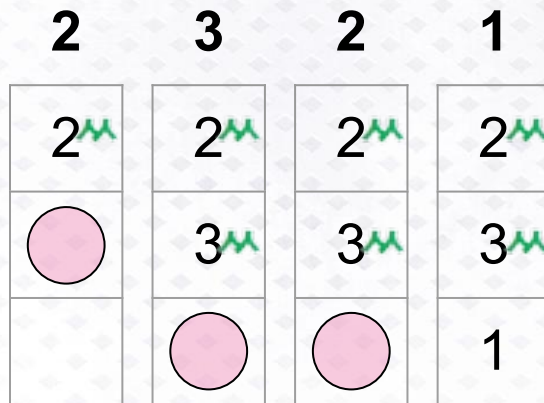


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

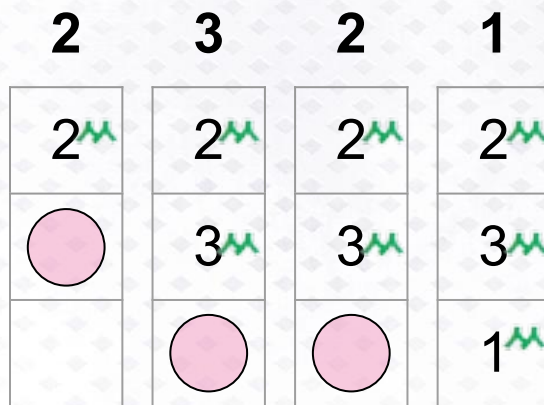


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

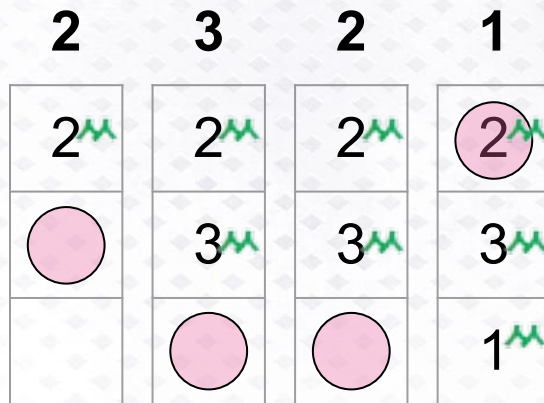


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

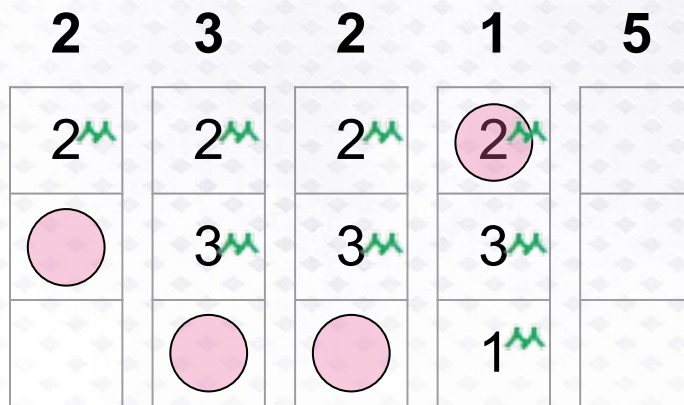


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

2	3	2	1	5
2	2	2	2	5
	3	3	3	3
			1	1

Fault count: 1

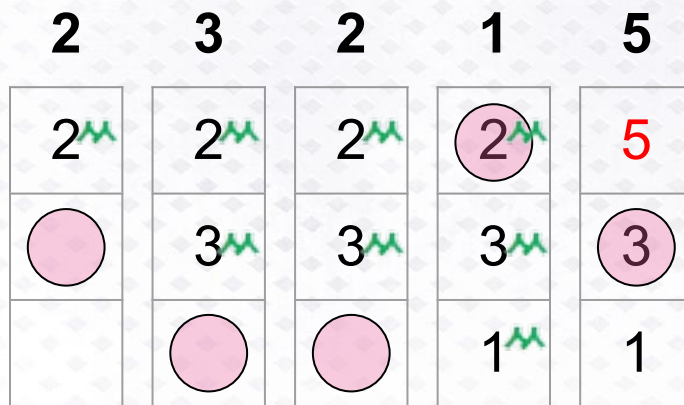
F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 1

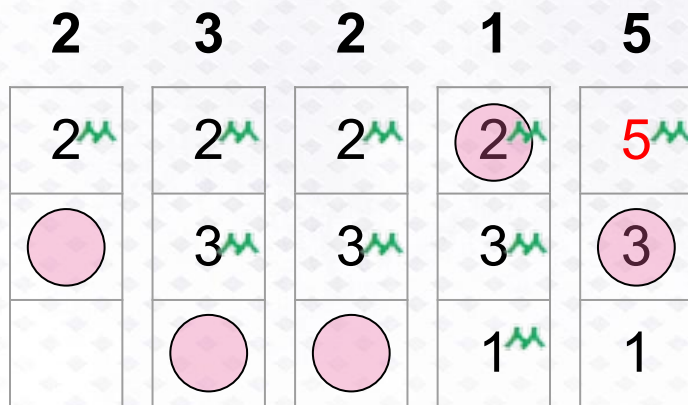
F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 1

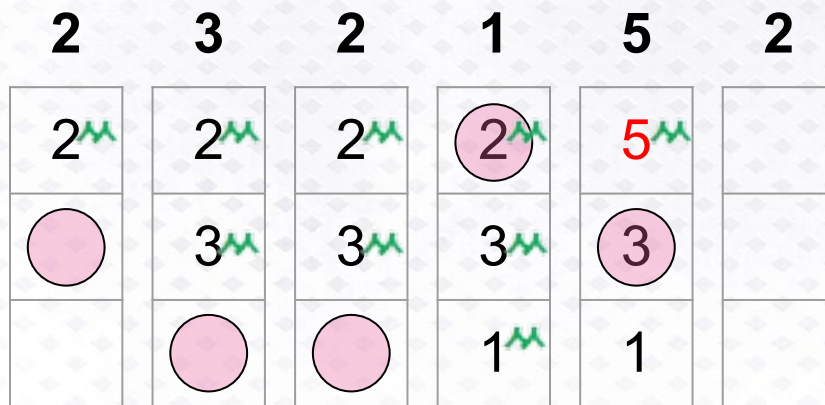
F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 1

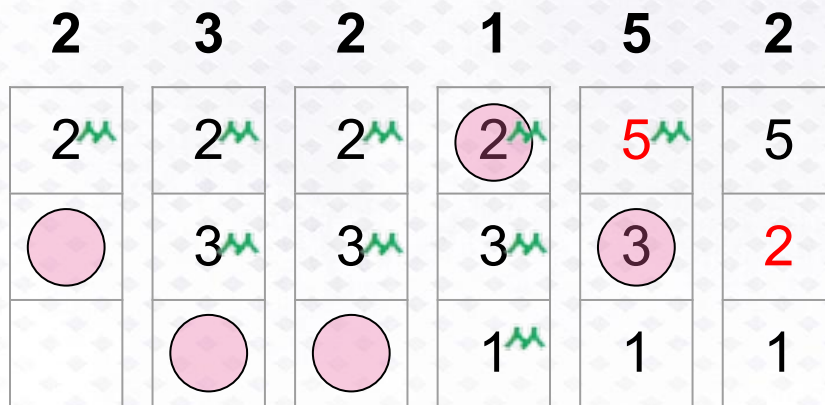
F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 2

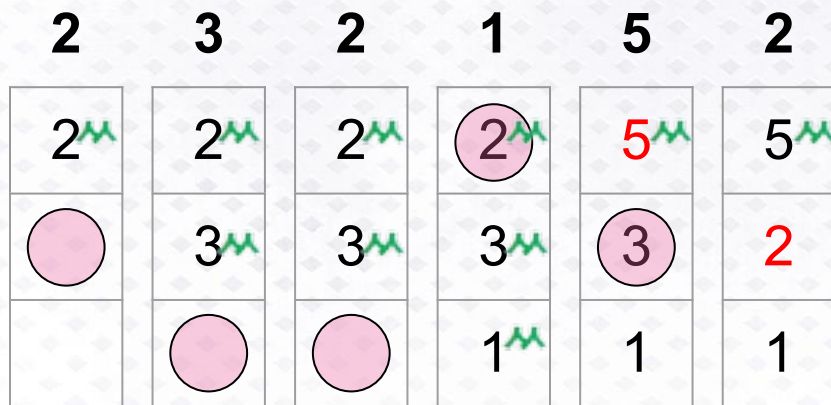
F **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 2

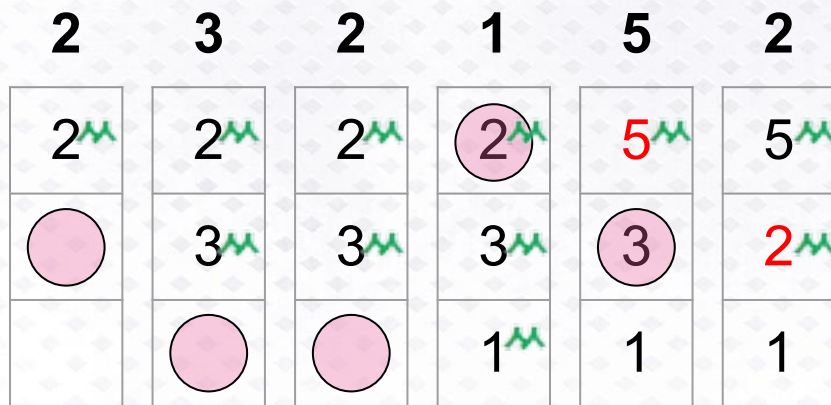
F **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 2

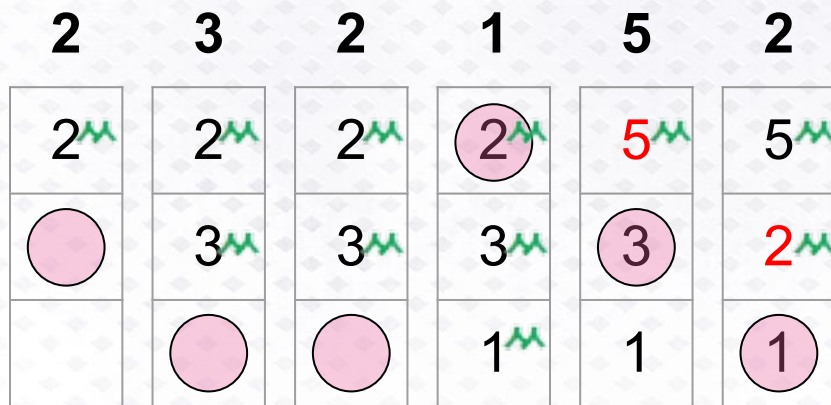
F **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 2

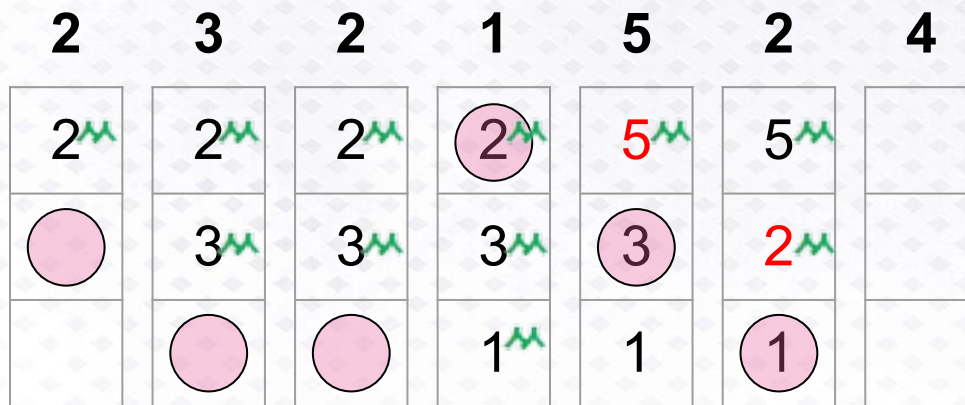
F **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 2

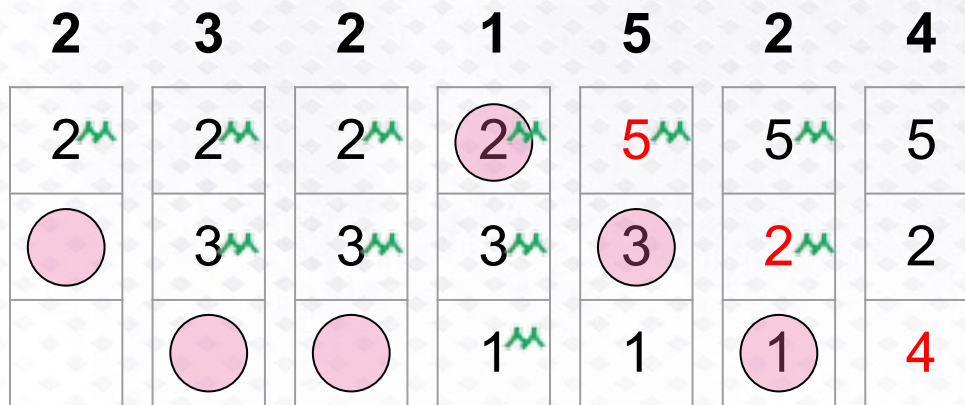
F **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 3

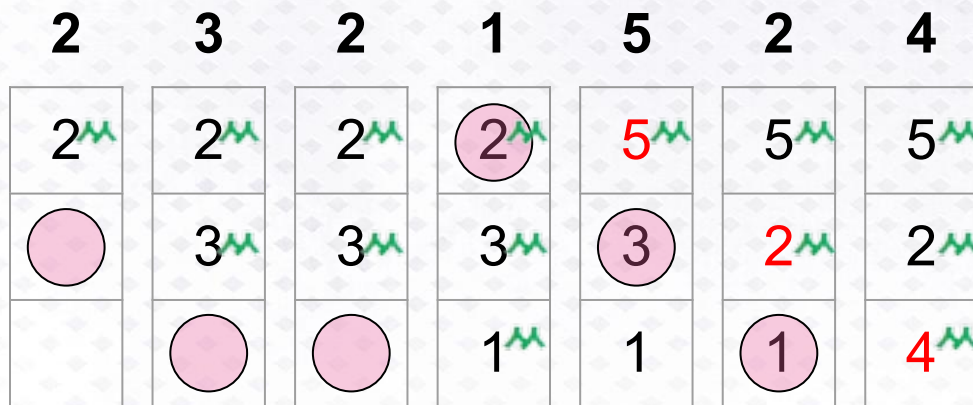
F F F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 3

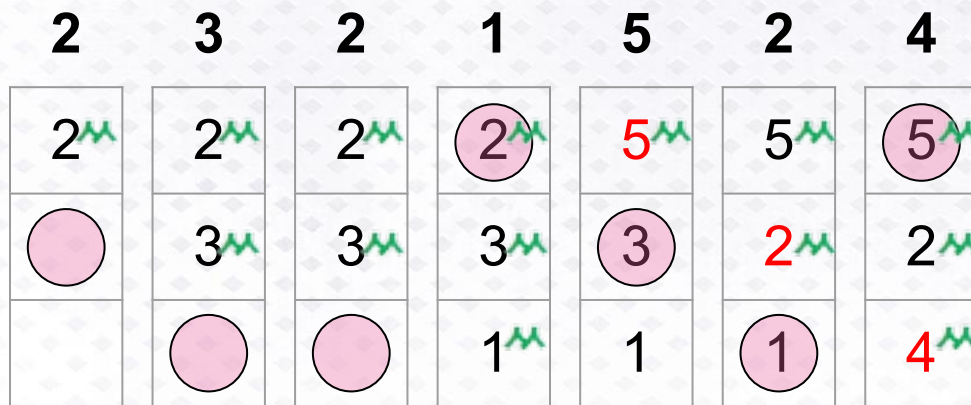
F **F** **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 3

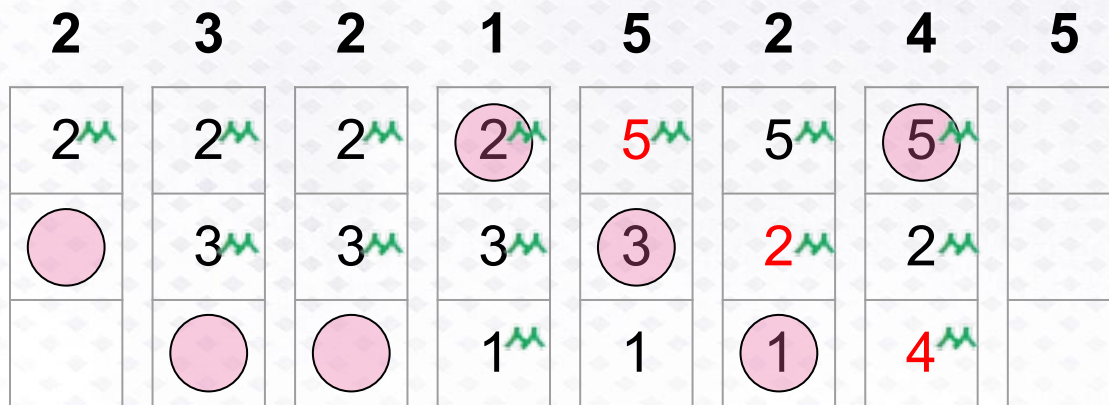
F **F** **F**

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 3

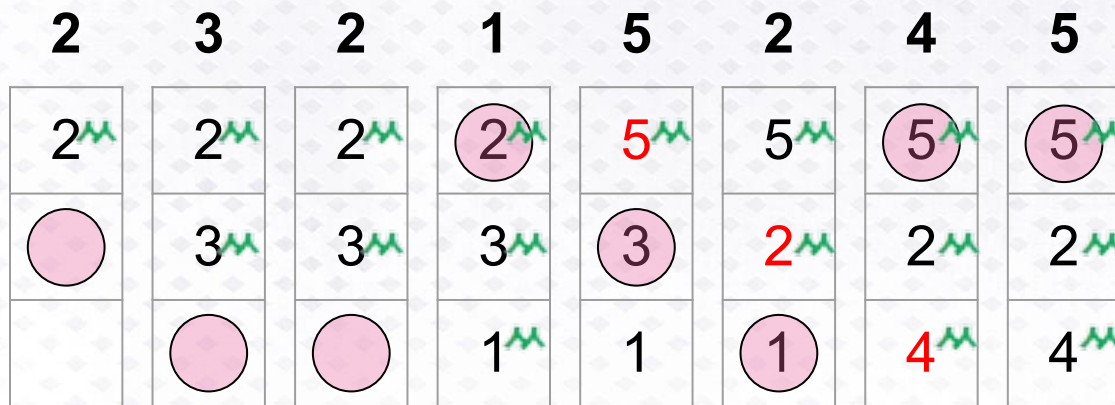
F F F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 3

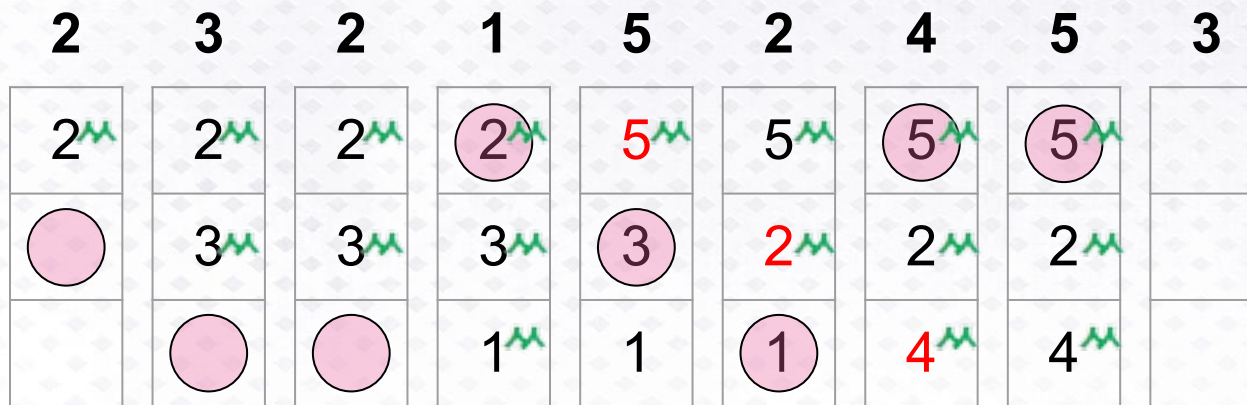
F F F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 3

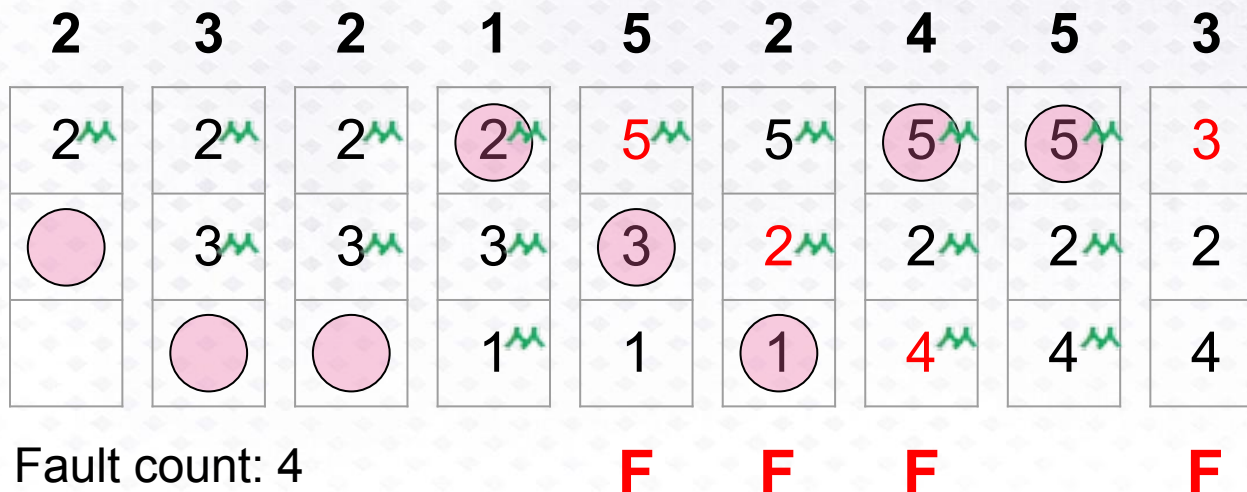
F F F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



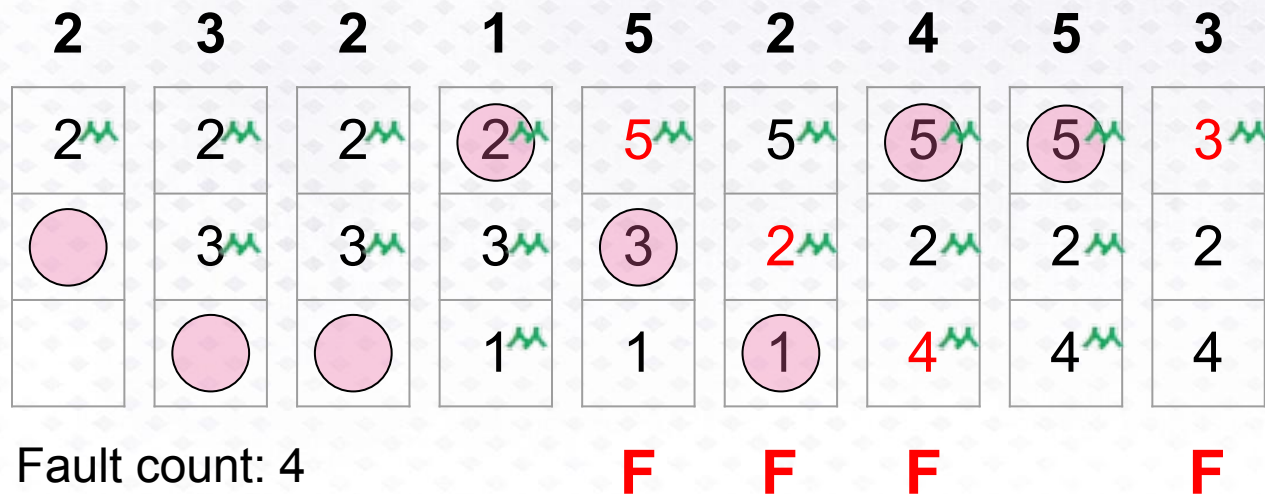
Fault count: 4

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

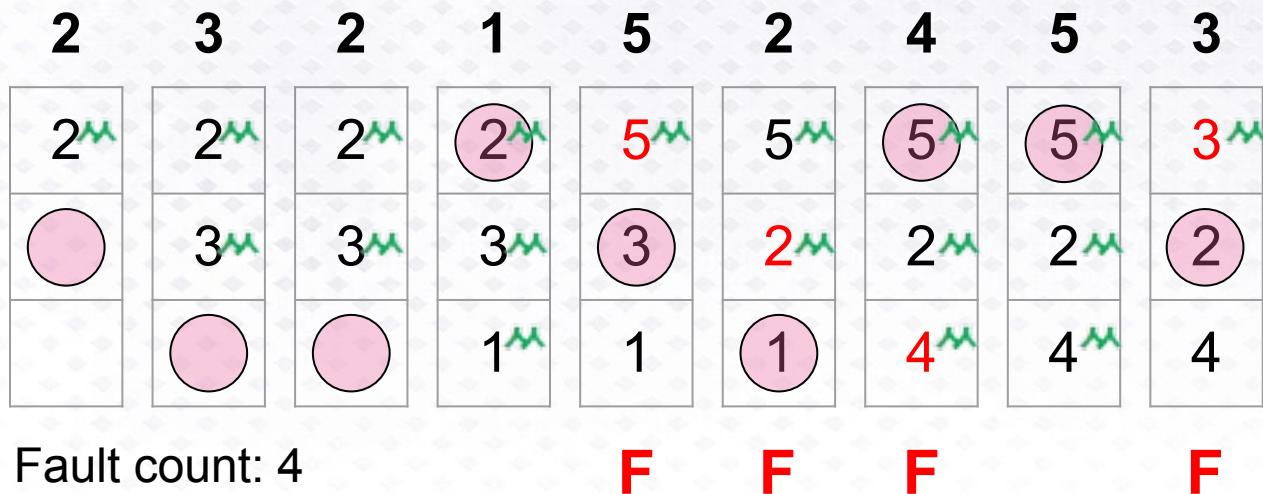


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

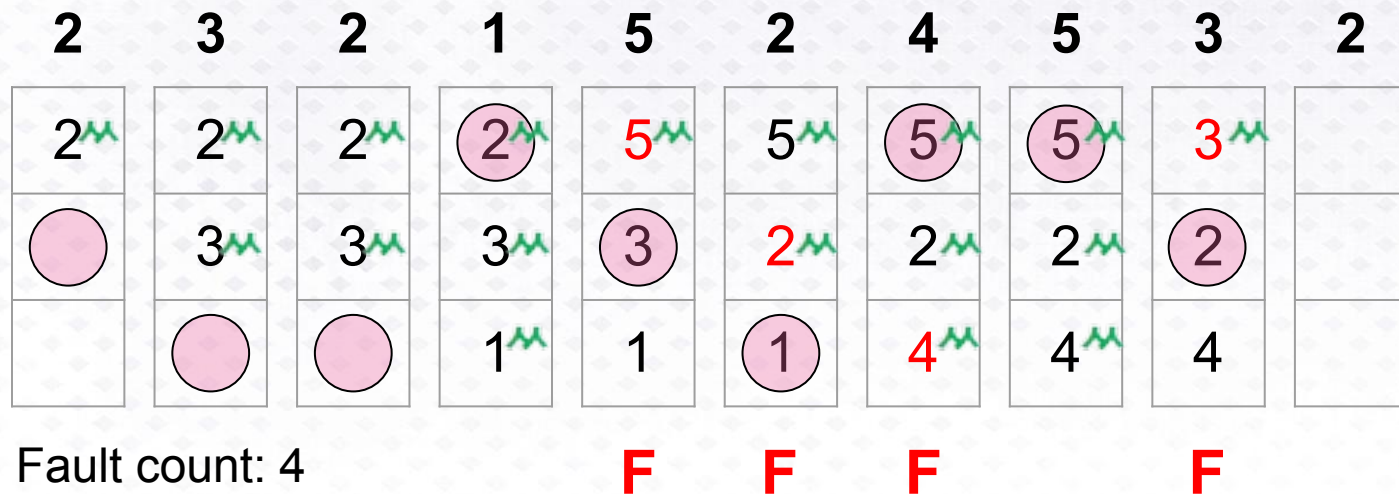


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2

Assume three frames, so room for three pages:

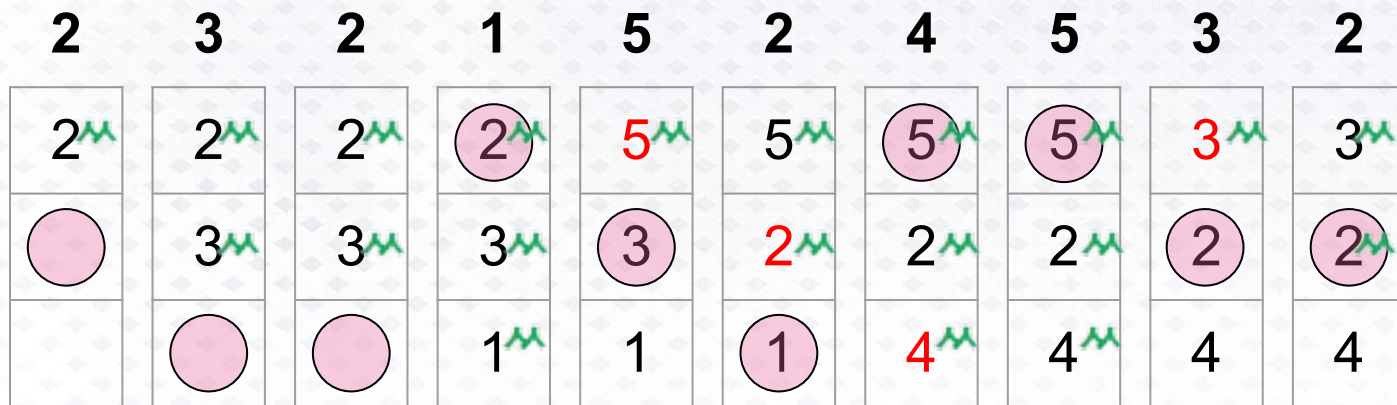


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2

Assume three frames, so room for three pages:



Fault count: 4

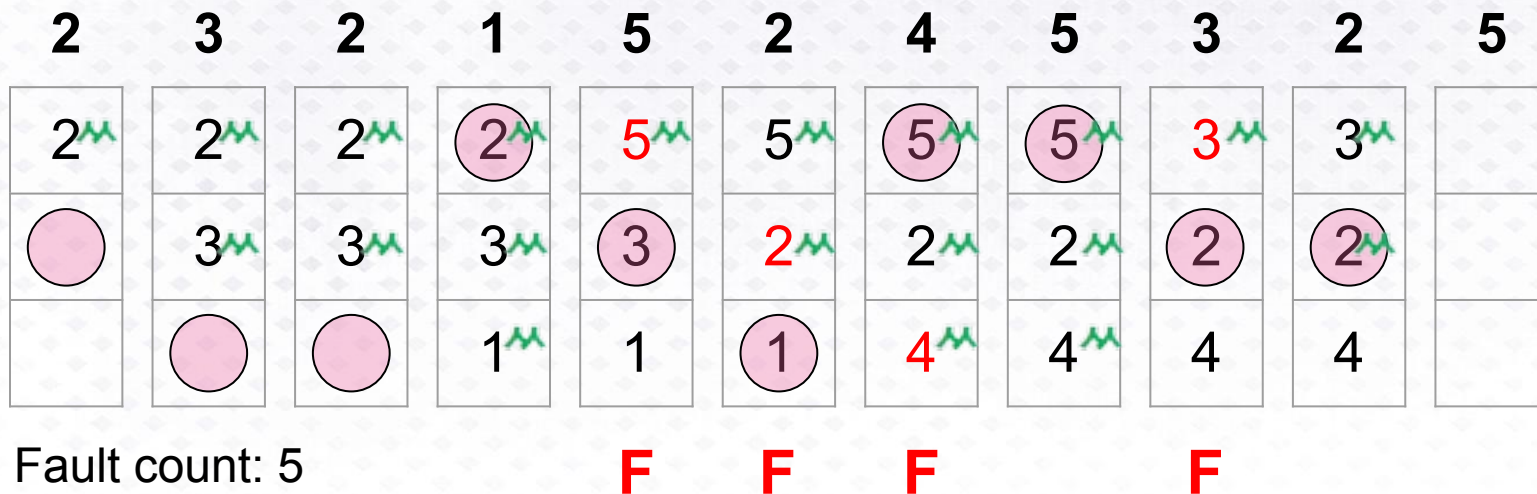
F F F F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

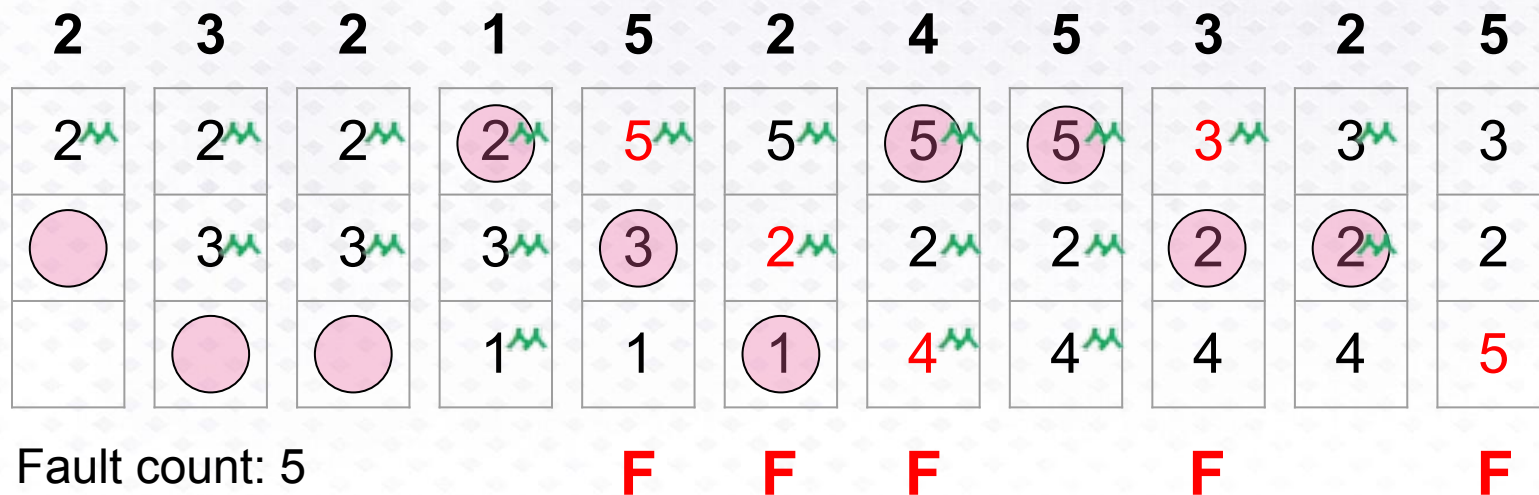


Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Fault count: 5

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

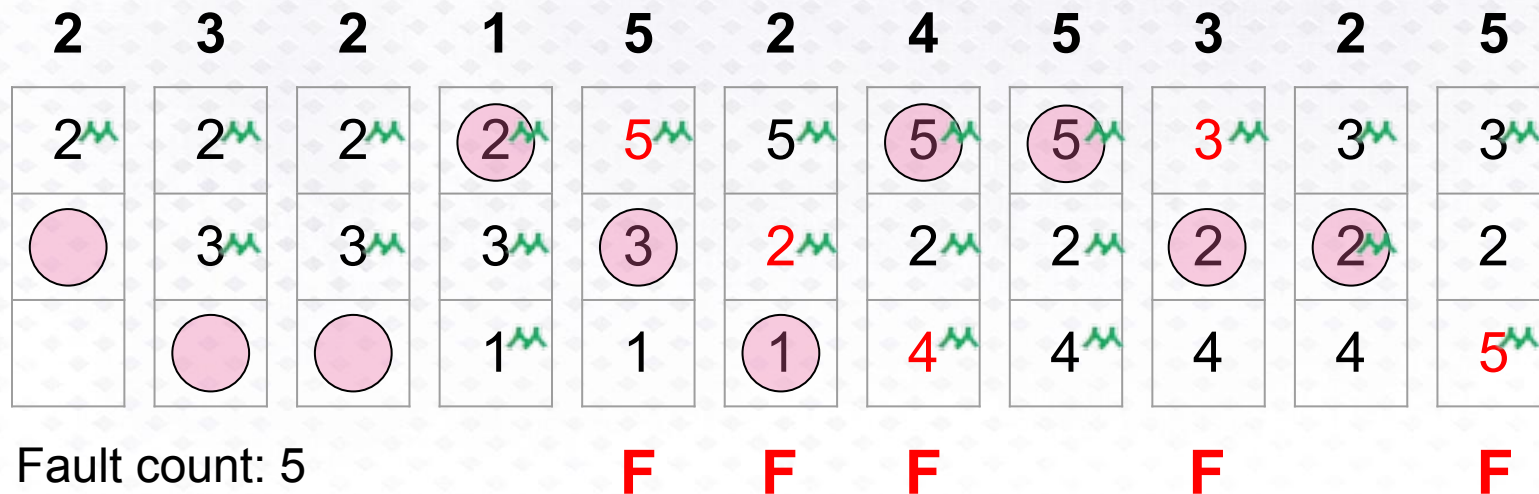
2	3	2	1	5	2	4	5	3	2	5
2	2	2	2	5	5	5	5	3	3	3
	3	3	3	3	2	2	2	2	2	2
			1	1	1	4	4	4	4	5
Fault count: 5				F	F	F		F		F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

Fault count: 5

F F F F F

Clock Allocation Worked Example

Page address stream formed by executing the program:

2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:

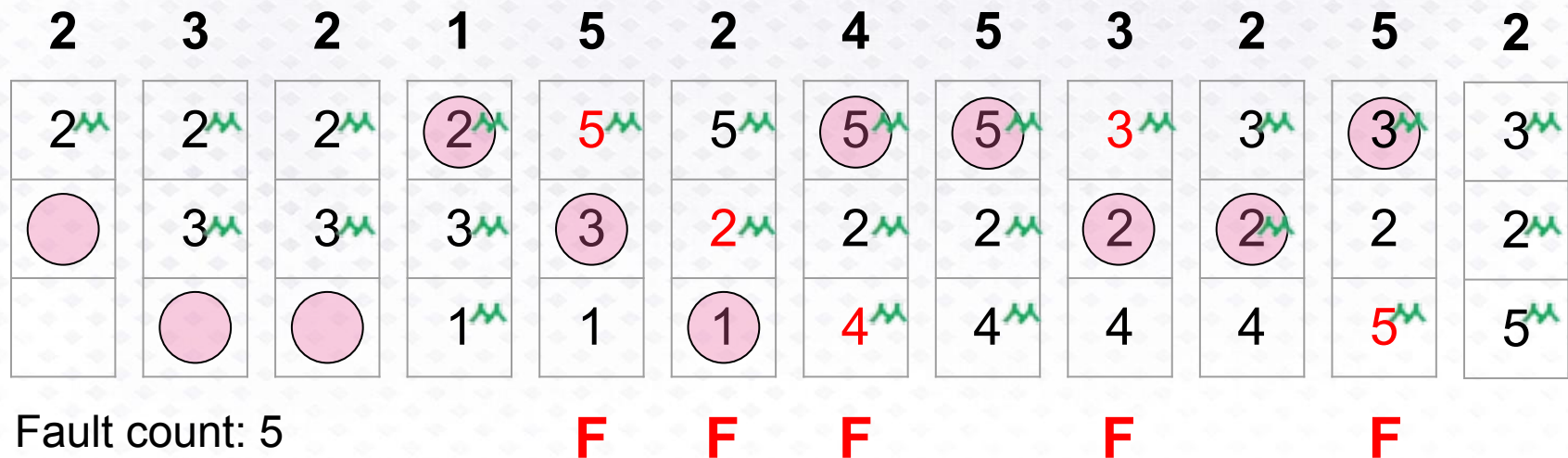
2	3	2	1	5	2	4	5	3	2	5	2
2	2	2	2	5	5	5	5	3	3	3	
	3	3	3	3	2	2	2	2	2	2	
			1	1	1	4	4	4	4	5	
Fault count: 5				F	F	F		F		F	

Clock Allocation Worked Example

Page address stream formed by executing the program:

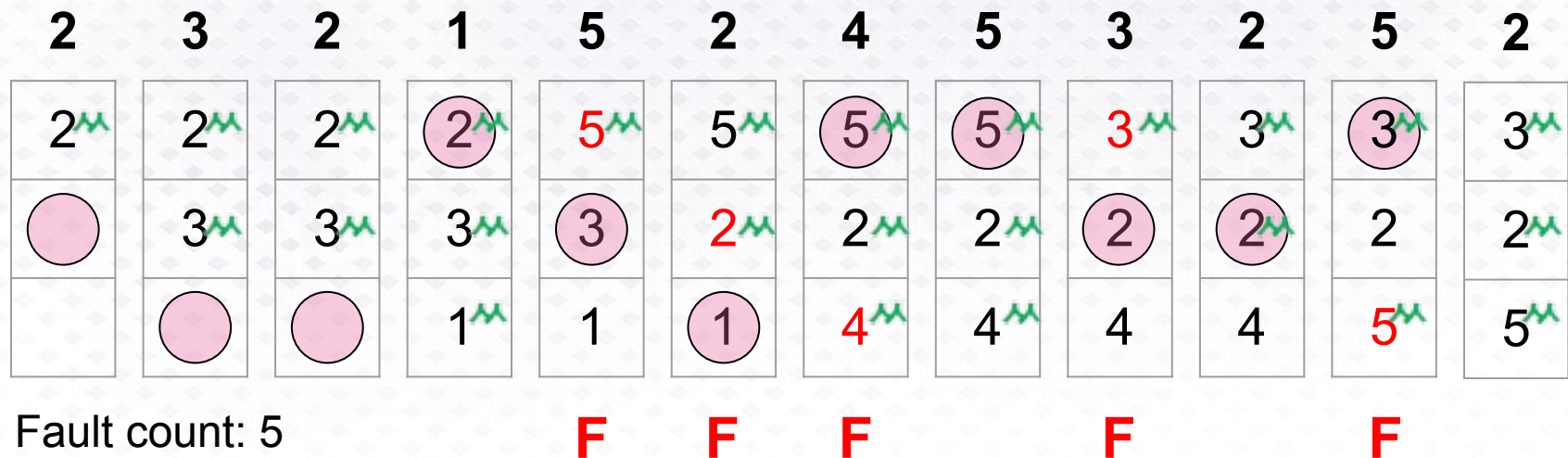
2 3 2 1 5 2 4 5 3 2 5 2

Assume three frames, so room for three pages:



Clock Allocation Worked Example

Note that the clock policy was adept at protecting frames 2 and 5 from replacement.



Clock Allocation Worked Example

An alternative view of the process:

Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	1	1	0	0	1	1	0	...	0	1

Clock Allocation Worked Example

An alternative view of the process:


Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	1	1	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME




Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	1	1	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	0	1	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	0	1	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	0	0	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	43	18	56	13	67	...	22	33
Use	1	1	0	0	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	74	18	56	13	67	...	22	33
Use	1	1	0	0	0	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	74	18	56	13	67	...	22	33
Use	1	1	0	0	1	0	1	1	0	...	0	1



Clock Allocation Worked Example

An alternative view of the process:

NEXT FRAME



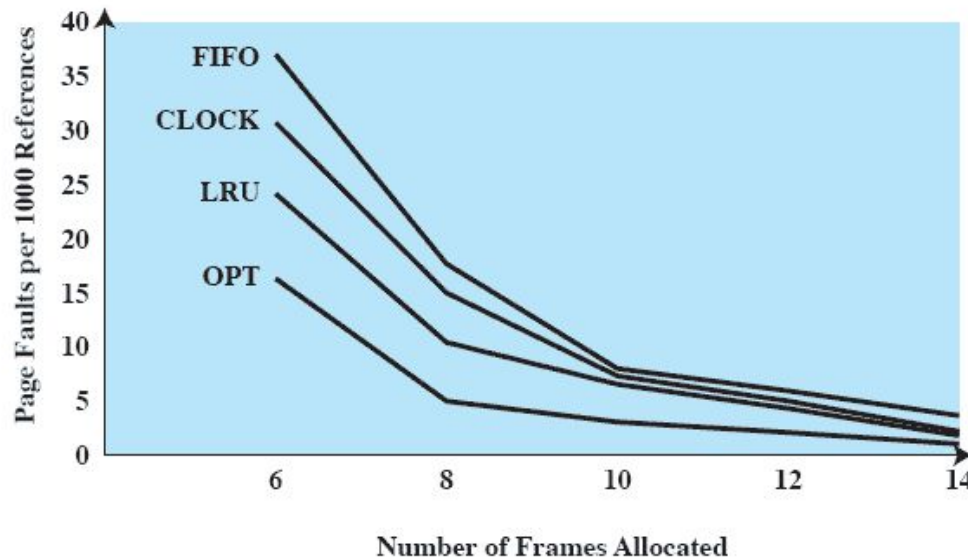
Frame	0	1	2	3	4	5	6	7	8	...	n-1	n
Page	9	19	1	45	74	18	56	13	67	...	22	33
Use	1	1	0	0	1	0	1	1	0	...	0	1



A comparison

(J Baer, Computer Systems Architecture, Computer Science Press, 1980)

- 2.5 x 10⁵ references in FORTRAN program
- Page size 256 words
- Frame allocations of 6, 8, 10, 12, and 14 frames



“Two Handed” clock replacement

(system V release 4)

Uses reference bit in page table entry

- Set bit to 0 when brought in
- Set bit to 1 when referenced
- Front hand sweeps list of eligible pages, setting bit to 0 on all pages
- (later) back hand sweeps and checks reference bit:
 - If bit = 1, page has been referenced, ignore
 - If bit = 0, page not referenced, place on “page out” list

“Two Handed” clock replacement

(system V release 4)

Parameters:

- Scan Rate
 - How fast hand sweeps (in pages per second)
- Handspread
 - Gap between front and back hands

Defaults are set at boot time, based on physical memory

Ethical Issues

HIPAA: Health Insurance Portability and Accountability Act

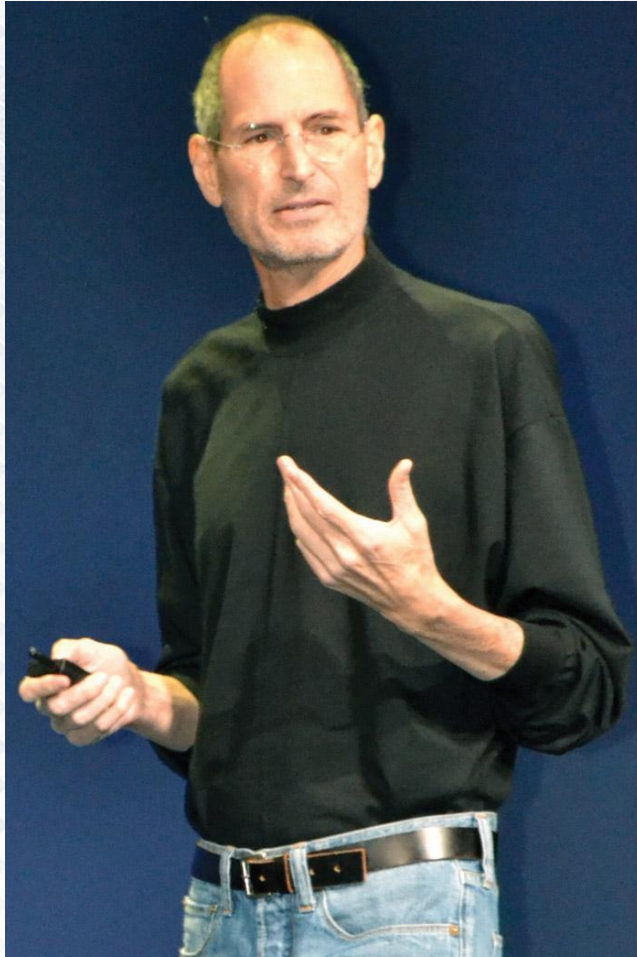
What was the goal of this act?

Have you ever had to sign one of HIPAA forms at the doctor's office?

What is “de-identified” information?

What are the problems with and benefits of this law?

Who am I?



© Christophe Ena/AP Photos

I will always be associated with a garage and a fruit. Do you know which fruit?

I have been in the news since 1976.

Why was I in the news in 2011?

Do you know?



Who is Blake Ross? What was his net worth when he retired at age 29?

What was the most influential computing job in the early '60s? Why?

Is Bitcoin a currency? Why or why not?

Do your sneakers talk to your iPod?

Explain the relationship between Nike and iTunes.