Nanyang Polytechnic

Medsync

IT2164/IT2561 Operating Systems and Administration

Chapter 5

Process Management

Process Management

At the end of this chapter, you should be able to

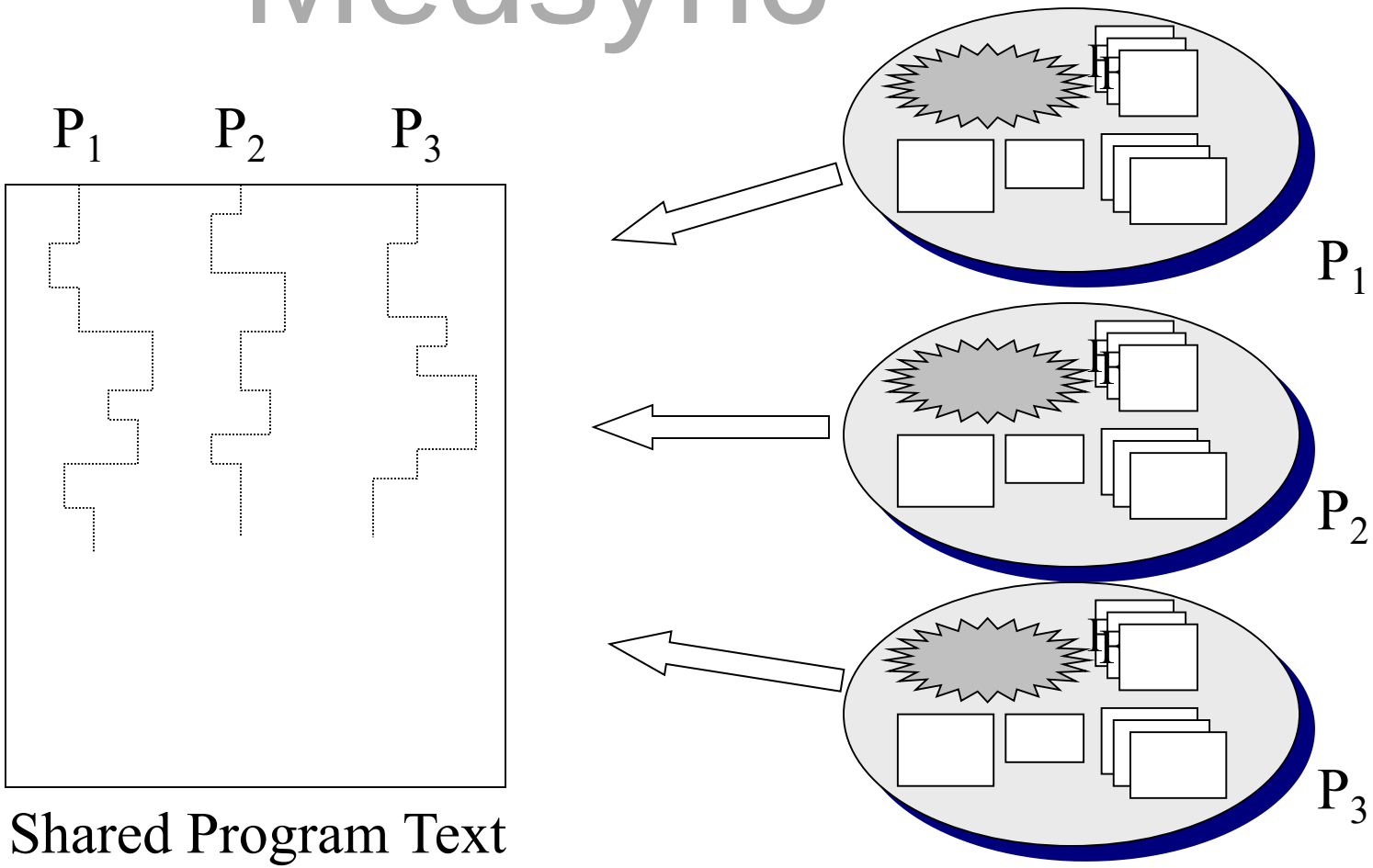
- Understand processes, threads and associated data structures
- Understand single and multithreading
- Understand and explain context switching
- Understand process states and process transition diagram

Processes and Threads

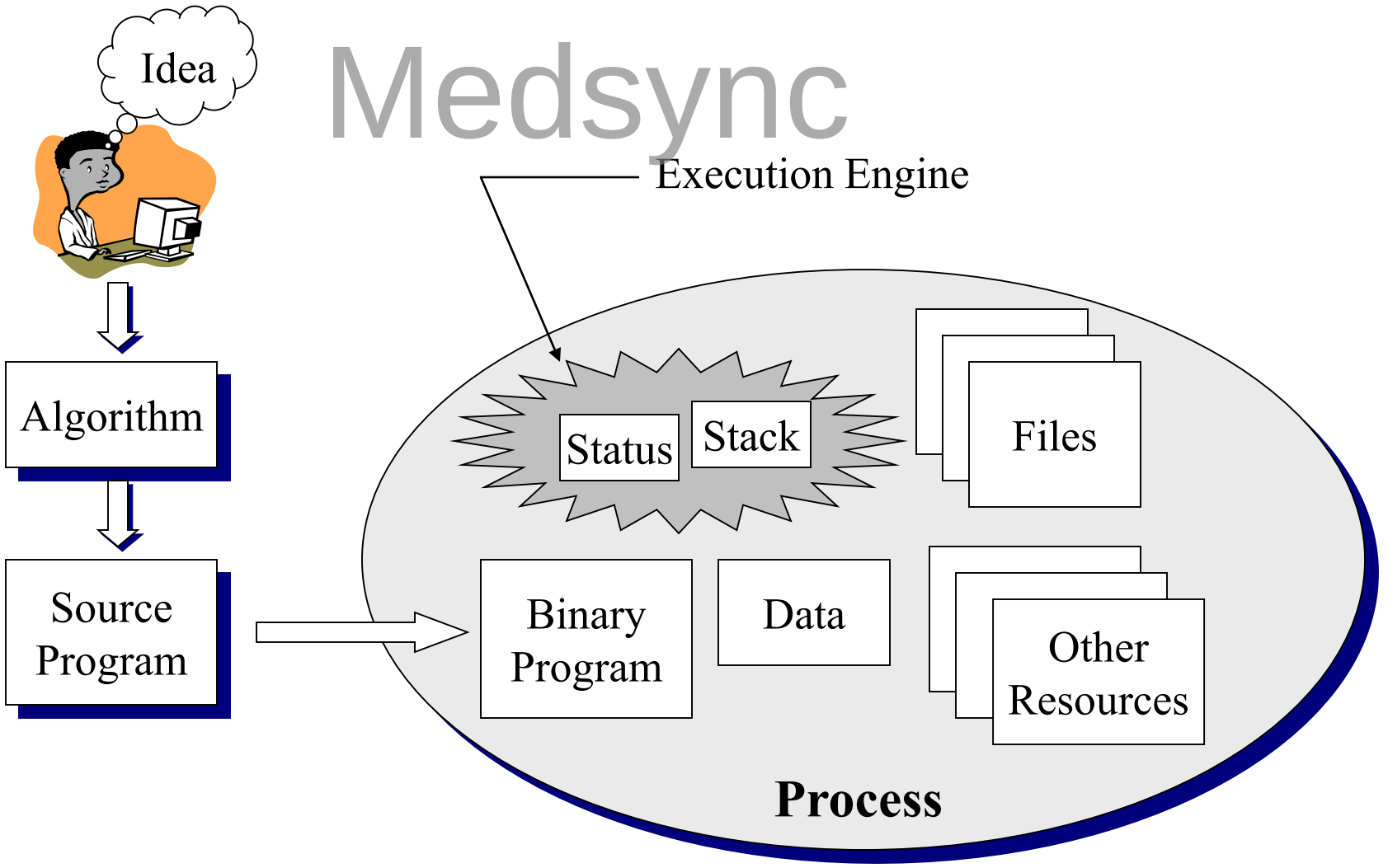
- A process is a binary program in execution.
- A process is made up of:
  - A binary **program**
  - **Data** on which the program will execute
  - **Resources** required for execution, including files, devices which contains or provides the data required.

Processes Sharing a Program

Medsync



Algorithms, Programs, and Processes



Processes and Threads

- In a classic process design, there is only one execution engine for each process.
- That is, there is only one thread of execution in one process.
- In modern computers, the modern process can consist of multiple execution engines.
- In such a design, a process can contain multiple threads.

Threads

- A thread is a single execution engine capable of performing a series of instructions in a computer program.
- In a multiple threaded process, each thread needs to maintain its own set of data in order to perform its own series of instructions.

Threads

- Examples
  - A web browser might have one thread display images or text while another thread retrieves data from network
  - A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from user and a third for performing spelling and grammar checking in the background

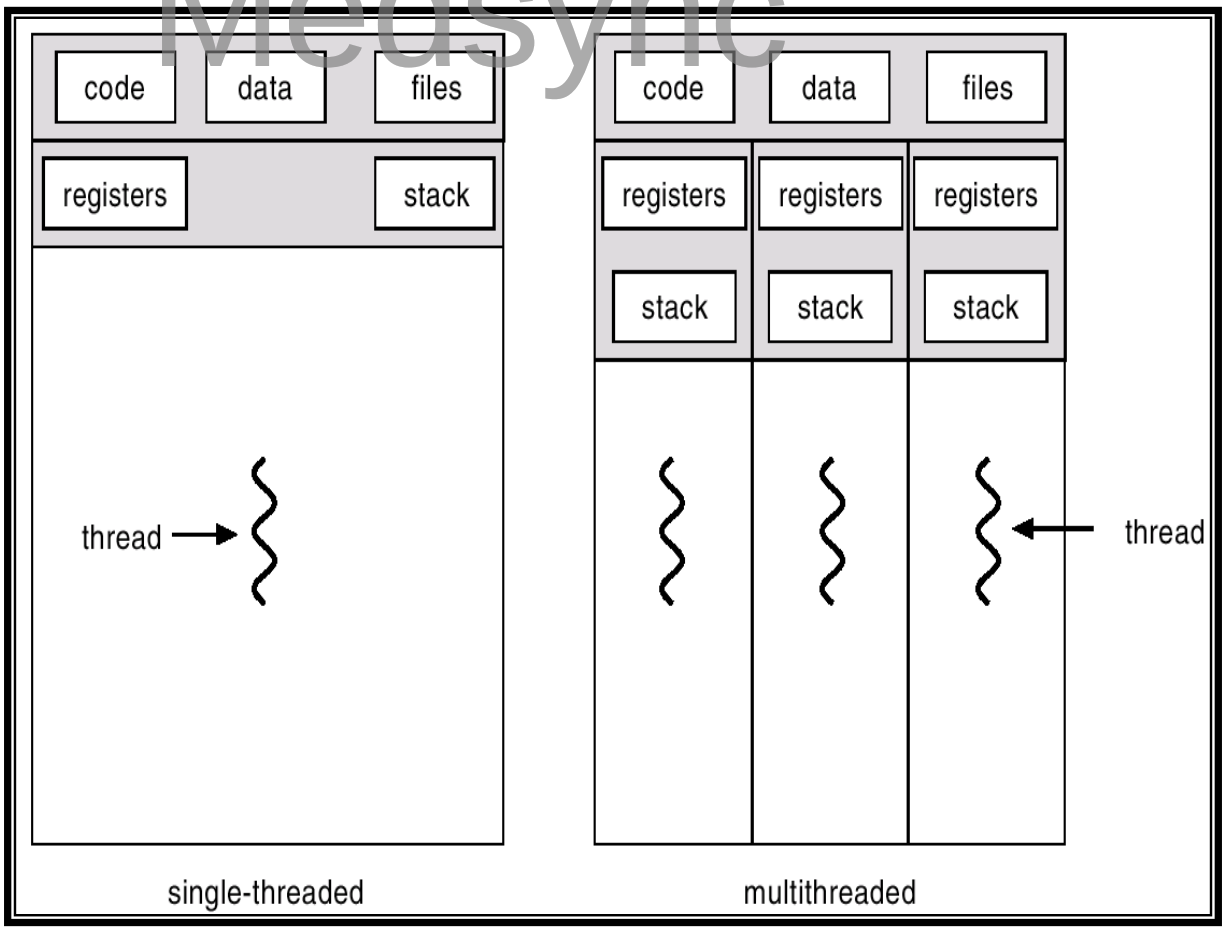
Threads

- Thread specific data is private to the thread. This data is usually stored in a stack.
- Thread specific data includes :
  - Program counter
  - Status of the thread
  - Processor registers
  - Stack space

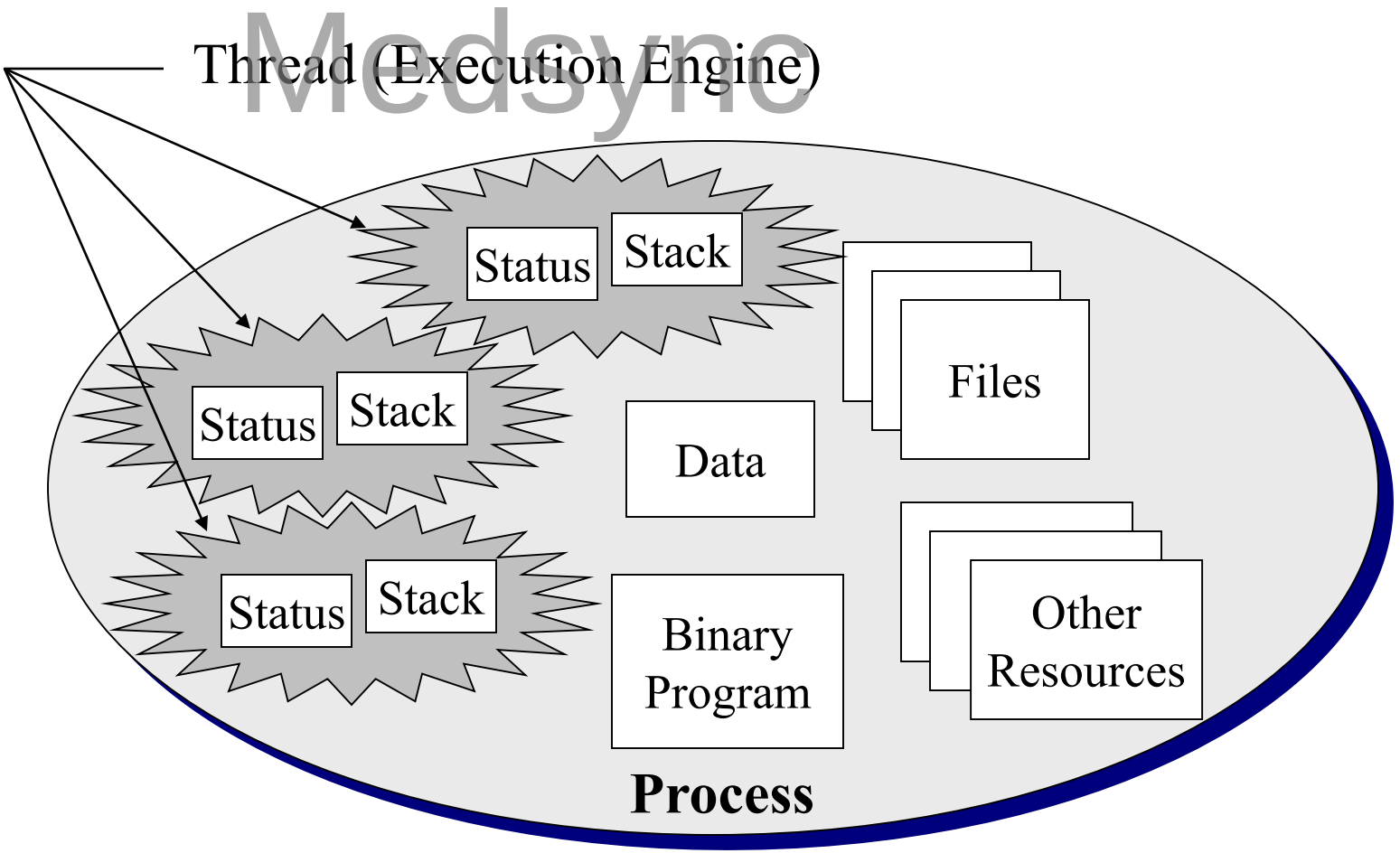
Threads

- Threads within the same process shares the same :
  - Program code
  - Data
  - Resources
- Sometimes threads are also called *lightweight processes.*

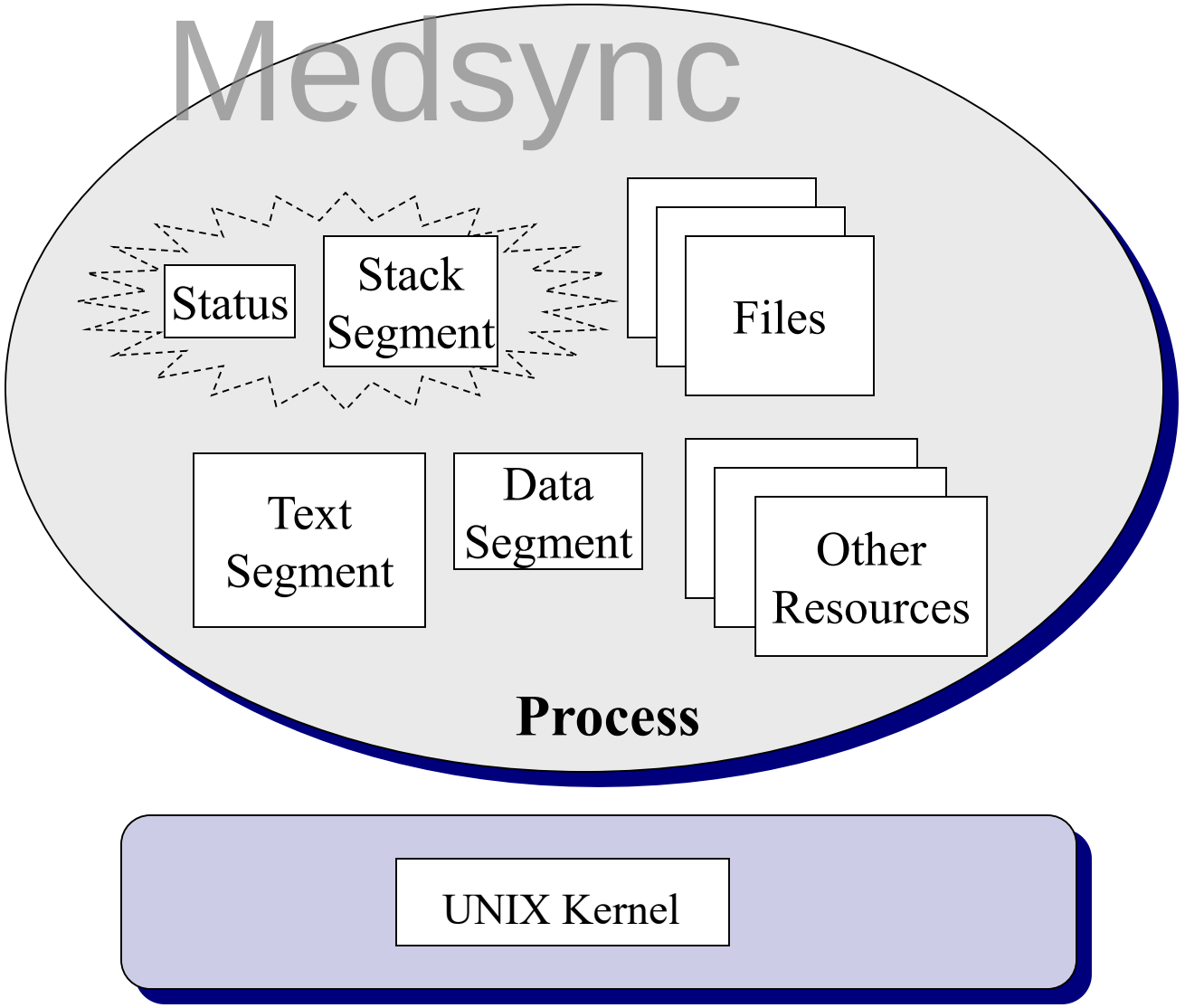
Single-threaded and Multi-threaded processes



A Process with Multiple Threads



UNIX Processes



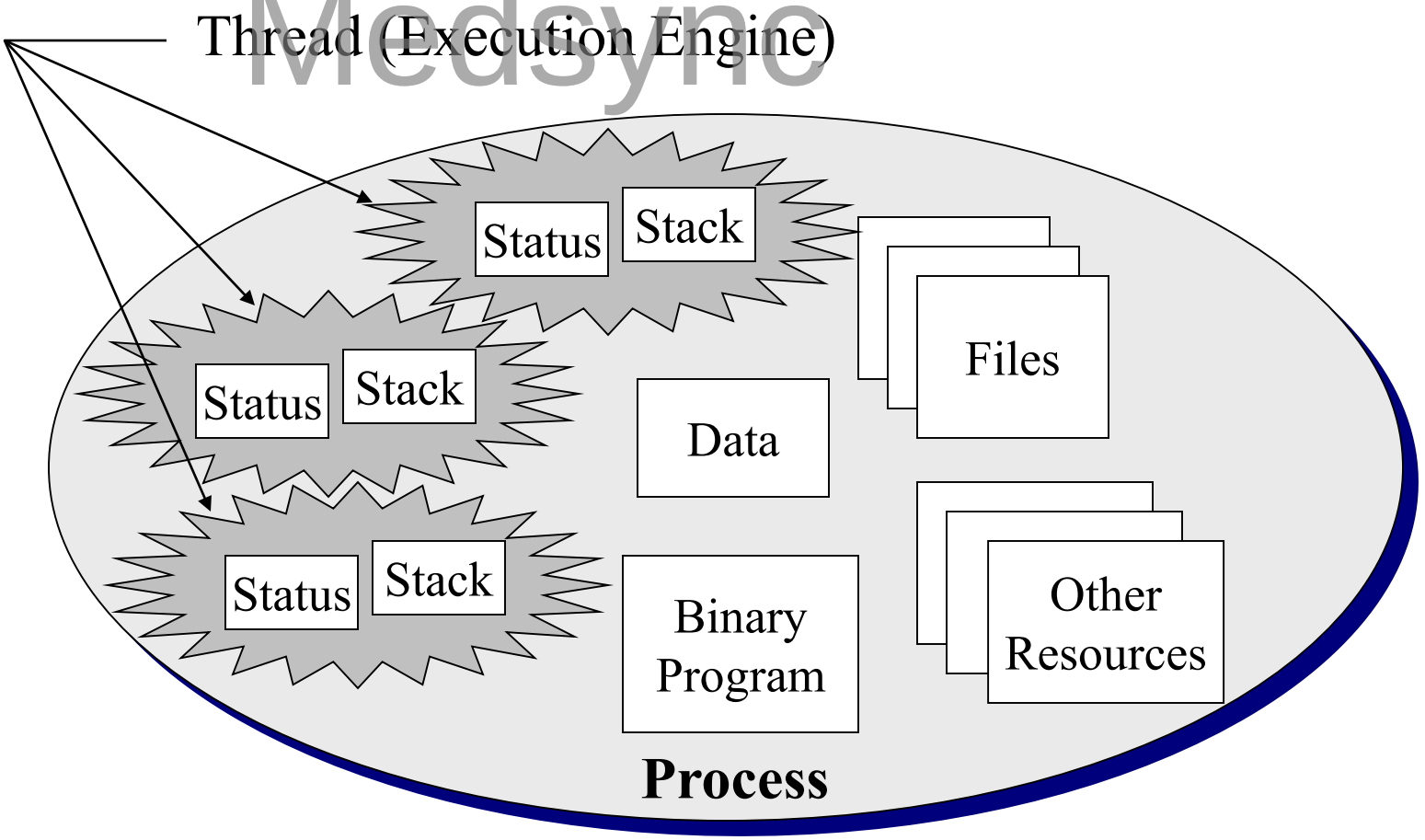
UNIX Processes

- Each process has its own address space
  - Subdivided into text, data, & stack segment
  - Program file describes the address space
- OS kernel creates a process descriptor to manage process
- Process identifier (PID): User handle for the process (descriptor)
- Try "`ps`" and "`ps -aux`"
- Unix classic processes have not explicit notion of a thread.

top command

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1697	root	-51	0	0	0	0	S	0.7	0.0	4:49.04	irq/46-nvidia
2093	root	20	0	1287452	400908	330456	S	0.7	1.2	26:57.87	Xorg
1	root	20	0	225840	9712	6800	S	0.3	0.0	0:12.00	systemd
286	root	20	0	0	0	0	S	0.3	0.0	6:33.85	nvidia-modeset
2	root	20	0	0	0	0	S	0.0	0.0	0:00.04	kthreadd
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H
7	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
8	root	20	0	0	0	0	S	0.0	0.0	0:00.08	ksoftirqd/0
9	root	20	0	0	0	0	I	0.0	0.0	0:04.43	rcu_sched
10	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_bh
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.03	migration/0
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.13	watchdog/0
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
15	root	rt	0	0	0	0	S	0.0	0.0	0:00.13	watchdog/1

Threads -- The NT Model



Threads -- The NT Model

- Windows Win32 API allows processes with multiple threads to be created through its CreateProcess() function.
- Options provided include :
  - Creating a new child process with a single thread.
  - Creating new additional threads in the current process.

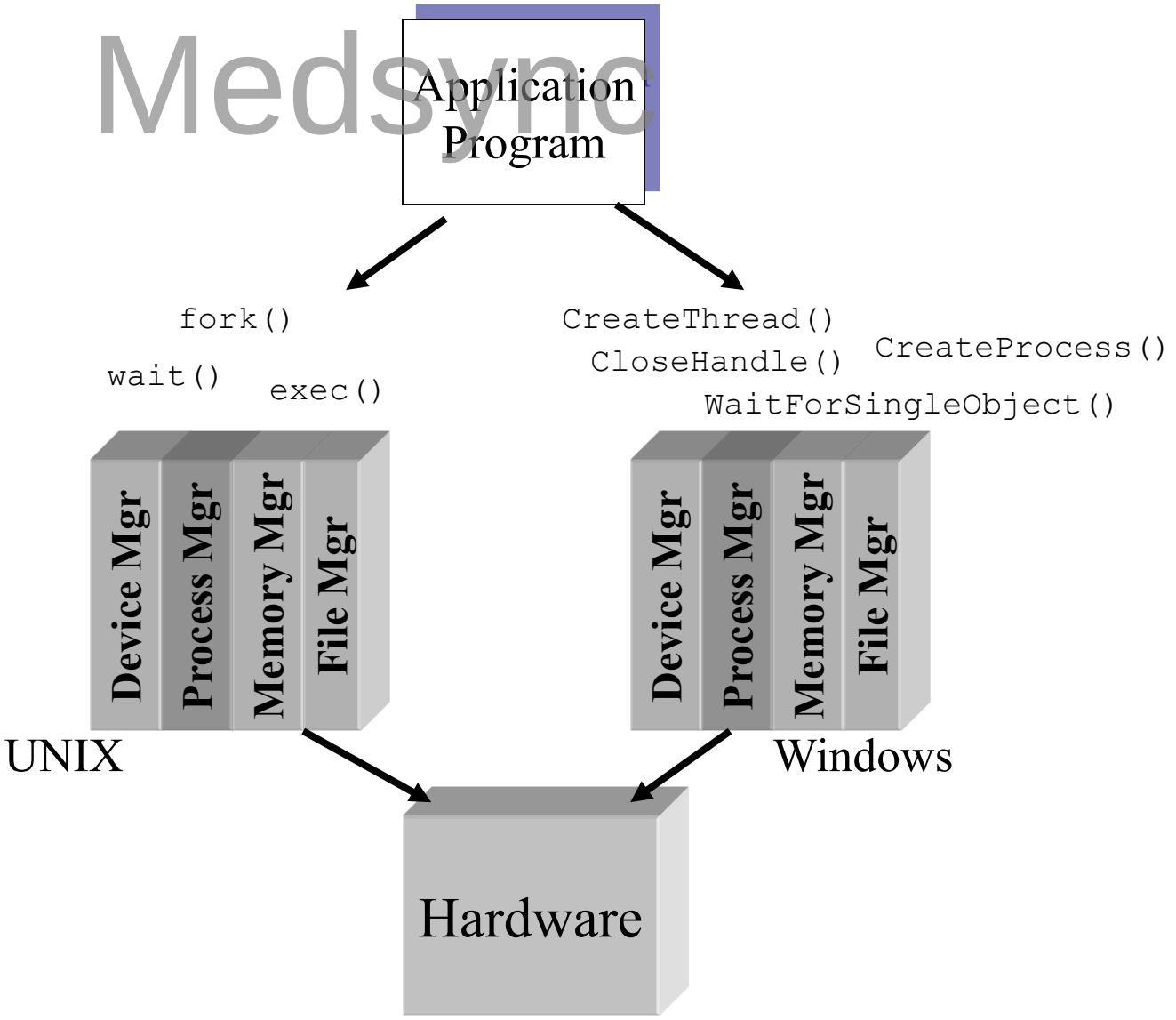
Benefits

- Some benefits of multithreaded programming
  - Responsiveness
  - Resource sharing
  - Ease of memory and resource allocation
  - Utilization of multiprocessor architectures

Process Manager

- To manage multiple processes, modern OS implement the process manager to manage the processes.
- The process manager implements :
  - Calls like fork() in UNIX and CreateProcess() in windows to create processes.
  - Calls like pthread_create() in Linux and CreateThread() in Windows to support threading.
  - Calls like close() in Unix and CloseHandle() in Windows to close processes/threads to release resources.

External View of the Process Manager



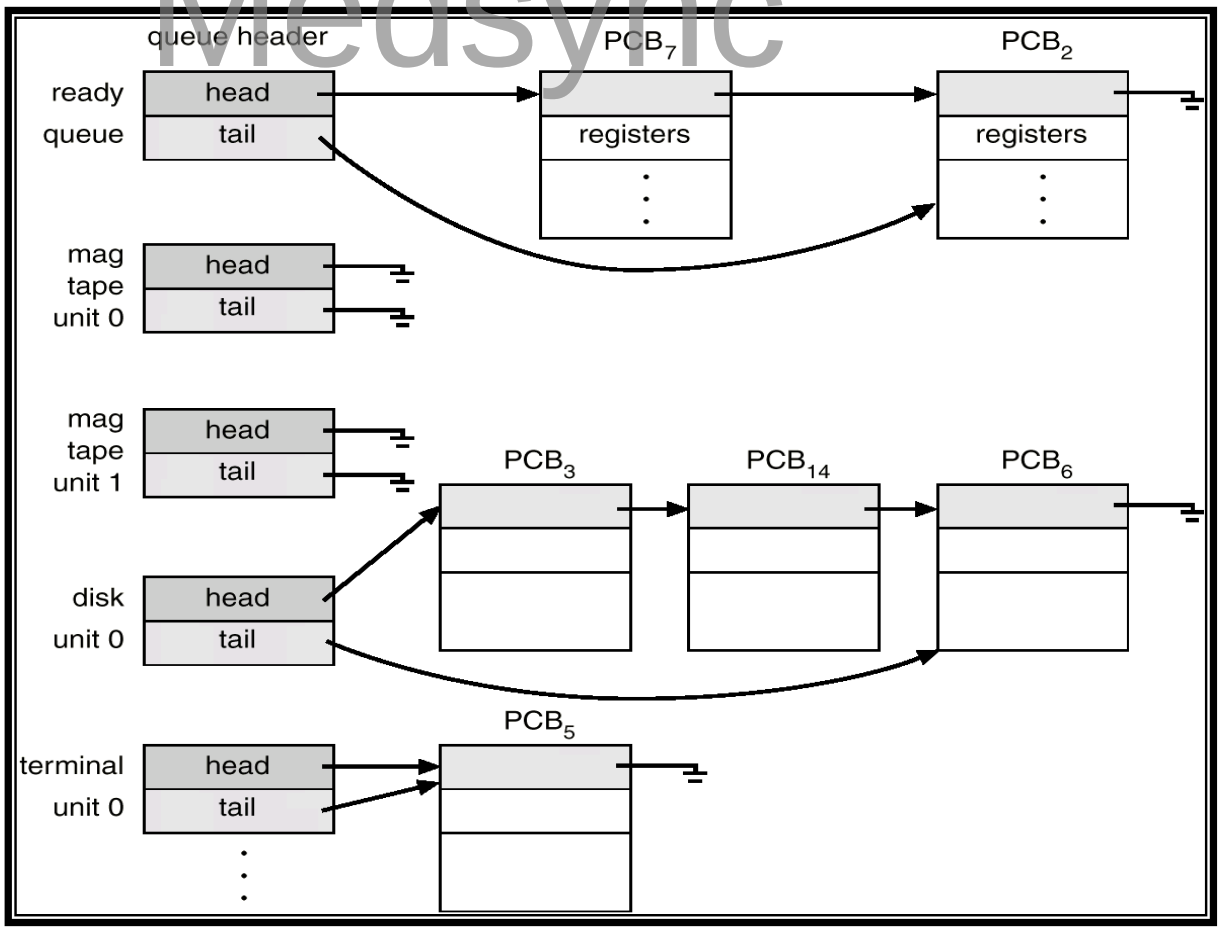
Process Manager Responsibilities

- Define & implement the essential characteristics of a process and thread
  - Algorithms to define the behavior
  - Data structures to preserve the state of the execution
- Define what “things” threads in the process can reference – the *address space* (most of the “things” are memory locations)
- Manage the resources used by the processes/threads
- Tools to create/destroy/manipulate processes & threads
- Tools to schedule the processes on the CPU.
- Tools to allow threads to synchronization the operation with one another.
- Mechanisms to handle deadlock.
- Mechanisms to handle protection.

Process Descriptors

- OS creates/manages process abstraction
- Descriptor is data structure for each process
  - Process ID
  - Program counter
  - Register values
  - Process state
  - Type & location of resources it holds
  - List of resources it needs
  - Security keys
- Also known as Process Control Block (PCB)

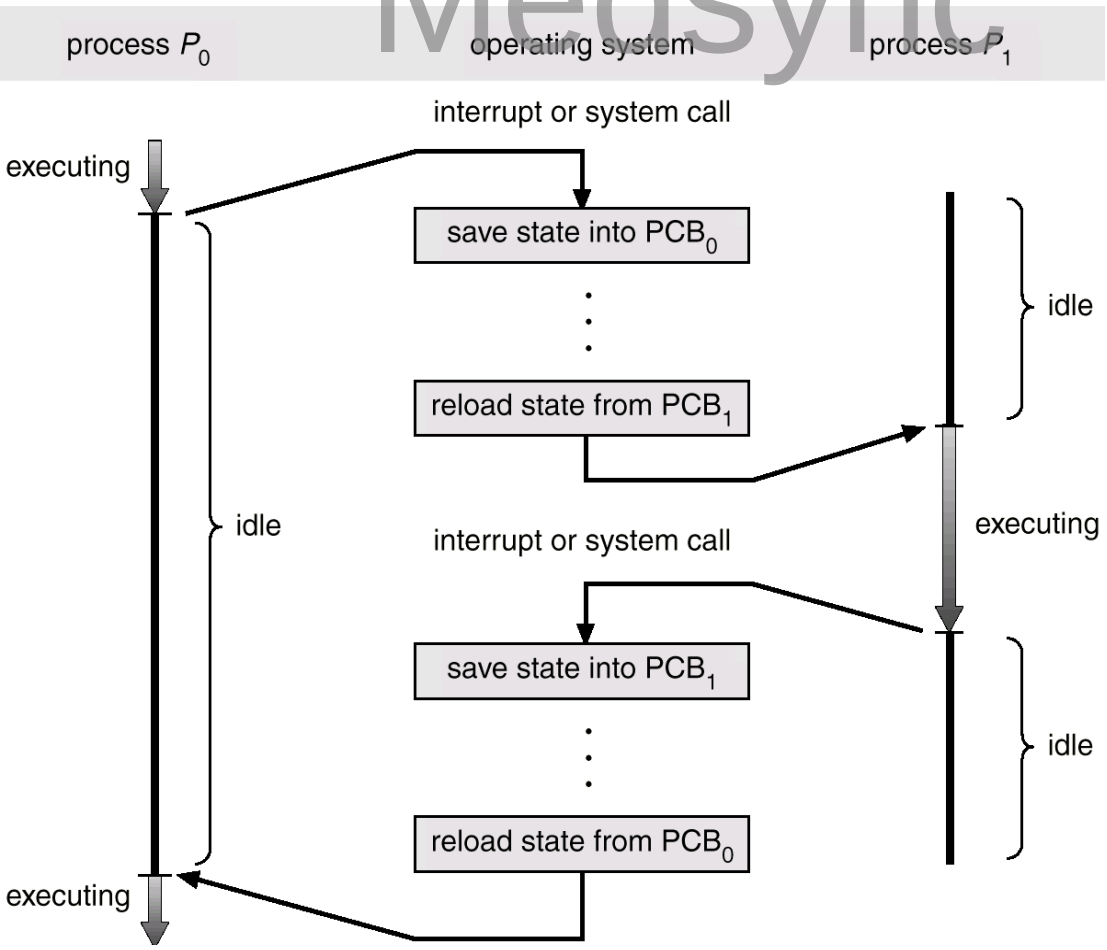
Queues of PCBs



Context Switching

- In a multiple process environment, each thread of execution is a context.
- When the CPU switches between two processes/threads, it is called a **context switch**.
- A context switch can only occur when the OS gets control of the CPU through traps or interrupts.

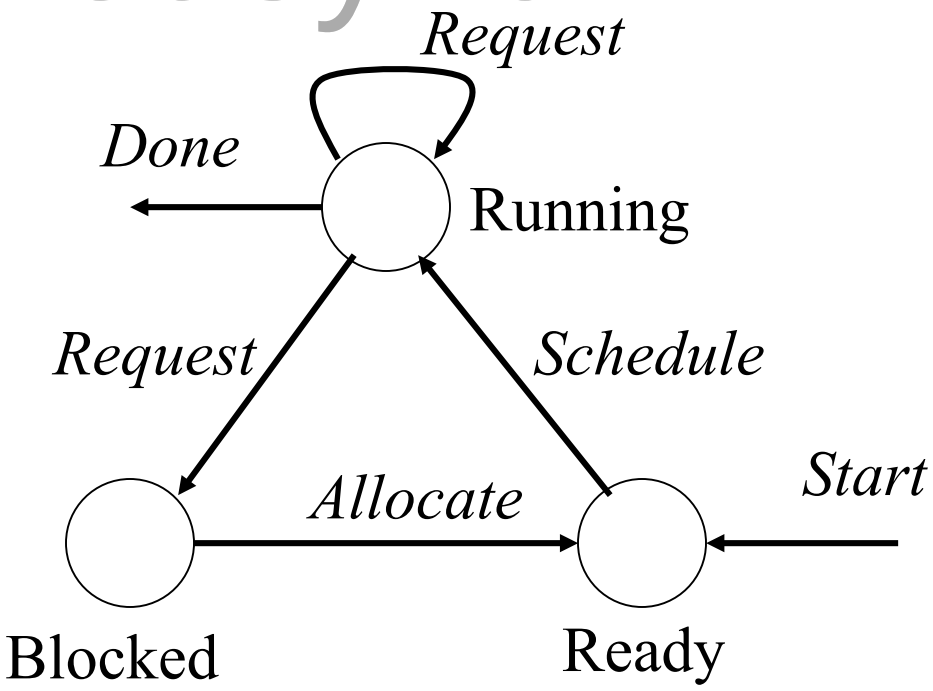
Context Switch



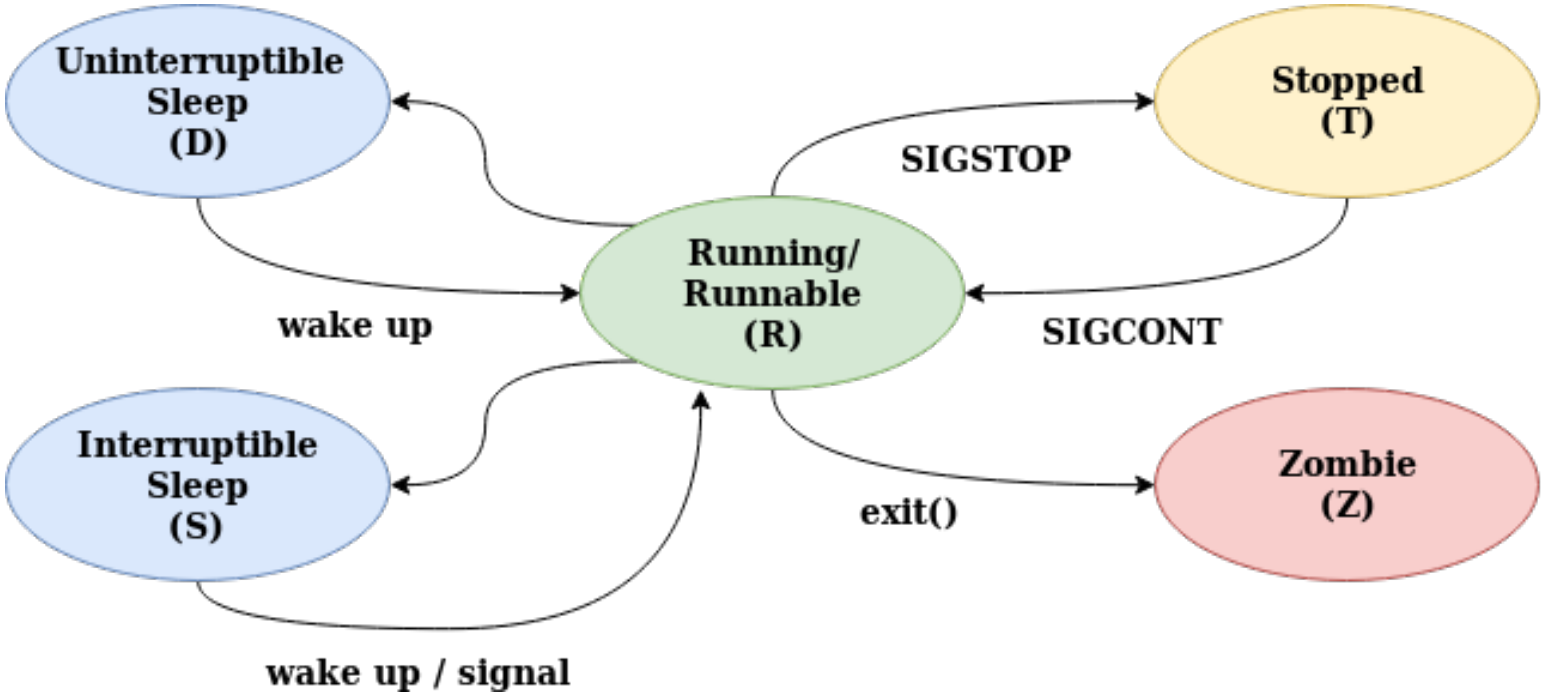
Process States

- As a process executes, it changes state:
  - Running: Instructions are being executed
  - Blocked: The process is waiting for some event to occur (eg, I/O completion)
  - Ready: The process is waiting to be assigned to a processor.
  - Done: The process has finished execution
- Modern OS implement additional states as required to support more complex features.

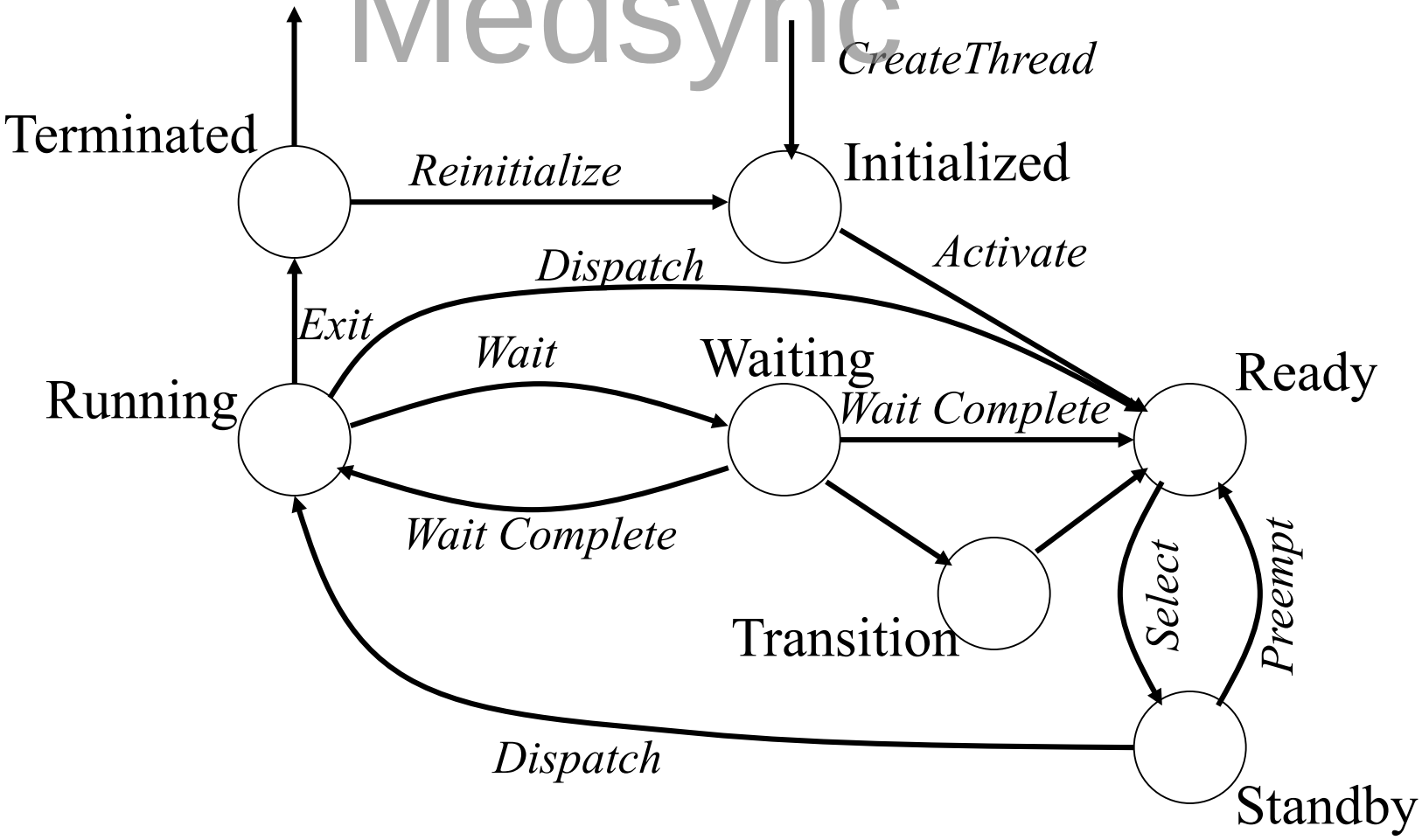
Process States



Linux State Transition Diagram



Windows NT Thread States



Conclusion

- Processes and threads form the basic form of execution in an operating system.
- Through abstraction, modern OS can support multiple processes and threads, leading to more efficient use of resources.
- Modern OS implement complex process/thread abstractions through the process manager to support complex features.
- Process manager is an integral part of modern OSs today.