**Important new transitions are the following:**

• **Blocked** → **Blocked/Suspend:** If there are no ready processes, then at least one blocked process is swapped out to make room for another process that is not blocked. This transition can be made even if there are ready processes available. In particular, if the OS determines that the currently running process, or a ready process that it would like to dispatch, requires more main memory to maintain adequate performance, a blocked process will be suspended.

• **Blocked/Suspend** → **Ready/Suspend:** A process in the Blocked/Suspend state is moved to the Ready/Suspend state when the event for which it has been waiting occurs. Note this requires that the state information concerning suspended processes must be accessible to the OS.

• **Ready/Suspend** → **Ready:** When there are no ready processes in main memory, the OS will need to bring one in to continue execution. In addition, it might be the case that a process in the Ready/Suspend state has higher priority than any of the processes in the Ready state. In that case, the OS designer may dictate that it is more important to get at the higher-priority process than to minimize swapping.

• **Ready** → **Ready/Suspend:** Normally, the OS would prefer to suspend a blocked process rather than a ready one, because the ready process can now be executed, whereas the blocked process is taking up main memory space and cannot be executed. However, it may be necessary to suspend a ready process if that is the only way to free up a sufficiently large block of main memory. Also, the OS may choose to suspend a lower–priority ready process rather than a higher–priority blocked process if it believes that the blocked process will be

ready soon.

**Several other transitions that are worth considering are the following:**

• **New** → **Ready/Suspend and New** → **Ready:** When a new process is created, it can either be added to the Ready queue or the Ready/Suspend queue. In either case, the OS must create a process control block and allocate an address space to the process. It might be preferable for the OS to perform these housekeeping duties at an early time, so it can maintain a large pool of processes that are not blocked. With this strategy, there would often be insufficient room in main memory for a new process; hence the use of the (New → Ready/Suspend) transition. On the other hand, we could argue that a just-in-time philosophy of creating processes as late as possible reduces OS overhead, and allows that

OS to perform the process creation duties at a time when the system is clogged with blocked processes anyway.

**Blocked/Suspend** → **Blocked:** Inclusion of this transition may seem to be poor design. After all, if a process is not ready to execute and is not already in main memory, what is the point of bringing it in? But consider the following scenario: A process terminates, freeing up some main memory. There is a process in the (Blocked/Suspend) queue with a higher priority than any of the processes in the (Ready/Suspend) queue and the OS has reason to believe that the blocking event for that process will occur soon. Under these circumstances, it would seem reasonable to bring a blocked process into main memory in preference to a ready process.

• **Running** → **Ready/Suspend:** Normally, a running process is moved to the Ready state when its time allocation expires. If, however, the OS is preempting the process because a higher-priority process on the Blocked/Suspend queue has just become unblocked, the OS could move the running process directly to the (Ready/Suspend) queue and free some main memory.

• **Any State** → **Exit:** Typically, a process terminates while it is running, either because it has completed or because of some fatal fault condition. However, in some operating systems, a process may be terminated by the process that created it or when the parent process is itself terminated. If this is allowed, then a process in any state can be moved to the Exit state.