Real-time computing is when a process *must* finish by a particular time. In some real-time systems, this might mean also that a process finishes no earlier than a certain time.
Real-time computing is when a process *must* finish by a particular time. In some real-time systems, this might mean also that a process finishes no earlier than a certain time.

There are *hard* real-time systems where there must be a *guarantee* on the finishing time.
Real-time computing is when a process must finish by a particular time. In some real-time systems, this might mean also that a process finishes no earlier than a certain time.

There are hard real-time systems where there must be a guarantee on the finishing time.

Soft real-time computing gives higher priority to processes with this requirement over other processes, but there need not be a guaranteed finishing time.
Examples are manufacturing control robots, autopilot flight systems, autocontrolled cars, safety cutoffs, weapon control systems.
Examples are manufacturing control robots, autopilot flight systems, autocontrolled cars, safety cutoffs, weapon control systems.

In a general computing system, these processes come with a specification of the times they need to complete or perform I/O. The scheduler can either admit the process with a guarantee on timely completion or reject the request as impossible. Such a rejection is termed resource reservation.
Examples are manufacturing control robots, autopilot flight systems, autocontrolled cars, safety cutoffs, weapon control systems.

In a general computing system, these processes come with a specification of the times they need to complete or perform I/O. The scheduler can either admit the process with a guarantee on timely completion or reject the request as impossible. Such a rejection is termed resource reservation.

Such a guarantee is impossible in computing environments with secondary memory, virtual memory, and general purpose system processes.
Examples are manufacturing control robots, autopilot flight systems, autocontrolled cars, safety cutoffs, weapon control systems.

In a general computing system, these processes come with a specification of the times they need to complete or perform I/O. The scheduler can either admit the process with a guarantee on timely completion or reject the request as impossible. Such a rejection is termed *resource reservation*.

Such a guarantee is impossible in computing environments with secondary memory, virtual memory, and general purpose system processes.

Thus, hard real-time systems are build on dedicated hardware systems and special-purpose software. These systems lack the full functionality of modern computer systems and operating systems.
These are less stringent than hard real-time systems and can be supported on general purpose computers.
These are less stringent than hard real-time systems and can be supported on general purpose computers.

The requirement is that these processes are given a higher priority than all other types of processes and that this priority does not degrade over time, unlike for other processes.
Soft Real-time Systems

- These are less stringent than hard real-time systems and can be supported on general purpose computers.
- The requirement is that these processes are given a higher priority than all other types of processes and that this priority does not degrade over time, unlike for other processes.
- Soft real-time functionality in time-sharing systems could lead to long delays, or even starvation for some processes.
Soft Real-time Systems

- These are less stringent than hard real-time systems and can be supported on general purpose computers.
- The requirement is that these processes are given a higher priority than all other types of processes and that this priority does not degrade over time, unlike for other processes.
- Soft real-time functionality in time-sharing systems could lead to long delays, or even starvation for some processes.
- The provision for soft real-time computing enables the support of multimedia, high-speed interactive graphics in general purpose systems.
Priority scheduling with highest priority for soft real-time processes is to be provided.
Scheduler Design for Soft Real-time Systems

- Priority scheduling with highest priority for soft real-time processes is to be provided.
- Priority degradation should not be allowed for soft real-time processes.
Priority scheduling with highest priority for soft real-time processes is to be provided.

Priority degradation should not be allowed for soft real-time processes.

Dispatch latency must be small. This reduces the delay between the request and beginning of execution of a real-time process.
• Priority scheduling with highest priority for soft real-time processes is to be provided.
• Priority degradation should not be allowed for soft real-time processes.
• Dispatch latency must be small. This reduces the delay between the request and beginning of execution of a real-time process.
• The first requirement is relatively straightforward to ensure. The low dispatch latency is much harder.
Preemption must normally wait for a system call to complete, or an I/O block to be copied before doing a context switch. This causes high dispatch latency because some system calls are complex and some I/O devices are slow.
Reducing Dispatch Latency

- Preemption must normally wait for a system call to complete, or an I/O block to be copied before doing a context switch. This causes high dispatch latency because some system calls are complex and some I/O devices are slow.

- The solution is to allow system calls to be preemptible. This is achieved by inserting preemption points in long-duration system calls. At these points, a check is made to see if a high priority process needs to run. After the termination of the higher priority process, the original process continues with the system call.
Preemption must normally wait for a system call to complete, or an I/O block to be copied before doing a context switch. This causes high dispatch latency because some system calls are complex and some I/O devices are slow.

The solution is to allow system calls to be preemptible. This is achieved by inserting preemption points in long-duration system calls. At these points, a check is made to see if a high priority process needs to run. After the termination of the higher priority process, the original process continues with the system call.

Preemption can only be placed in safe locations in the kernel. That is, points where no kernel datastructures are being modified.
Another possibility is to make the entire kernel preemptible. Here, any ordinary process modifying a data structure used by higher priority processes are temporarily assigned a higher priority till they finish that modification. This is a solution using synchronisation.
Another possibility is to make the entire kernel preemptible. Here, any ordinary process modifying a data structure used by higher priority processes are temporarily assigned a higher priority till they finish that modification. This is a solution using synchronisation.

Higher priority processes waiting for lower priority processes to finish, in these circumstances is called *priority inversion*. 
Another possibility is to make the entire kernel preemptible. Here, any ordinary process modifying a data structure used by higher priority processes are temporarily assigned a higher priority till they finish that modification. This is a solution using synchronisation.

Higher priority processes waiting for lower priority processes to finish, in these circumstances is called priority inversion.

A chain of such processes can exist with increasing priorities. The solution of this problem as above is called the priority-inheritance protocol. Low priority processes modifying data structures used by higher priority processes, temporarily inherit that higher priority.
Another possibility is to make the entire kernel preemptible. Here, any ordinary process modifying a data structure used by higher priority processes are temporarily assigned a higher priority till they finish that modification. This is a solution using synchronisation.

Higher priority processes waiting for lower priority processes to finish, in these circumstances is called priority inversion.

A chain of such processes can exist with increasing priorities. The solution of this problem as above is called the priority-inheritance protocol. Low priority processes modifying data structures used by higher priority processes, temporarily inherit that higher priority.

This is called the conflict phase of the dispatch latency.
• Preemption of any process running in the kernel.
Conflict Phase Components

- Preemption of any process running in the kernel.
- Low priority processes releasing resources needed by the higher priority processes.
Conflict Phase Components

- Preemption of any process running in the kernel.
- Low priority processes releasing resources needed by the higher priority processes.
- Context switching from current process to a higher priority process.
Deterministic modelling is an explicit system review for a specific workload which has occurred. This is a fast and simple method but applies only to a specific case. Deterministic modelling is a special instance of a class of evaluation methods called analytic evaluation.
Deterministic modelling is an explicit system review for a specific workload which has occurred. This is a fast and simple method but applies only to a specific case. Deterministic modelling is a special instance of a class of evaluation methods called analytic evaluation.

Queueing models analyse the dynamics of the size of the various scheduling queues and draw conclusions on the system performance. This area is called queueing-network analysis.
Simulations program a model of the computer system with various assumptions. Detailed simulation with greater compliance with the real situation gives better results. A simulation however, requires a lot of time.
Simulations program a model of the computer system with various assumptions. Detailed simulation with greater compliance with the real situation gives better results. A simulation however, requires a lot of time.

Implementation is a review method where the evaluation takes place in a system in functioning. This is clearly the most accurate, but acting on the conclusions drawn, require reworking the system which is expensive. Frequent changes are also normally not acceptable to the users.