**Types of Recovery**

Error recovery is part of the larger software fault tolerance process involving error (or fault) detection, diagnosis, isolation or containment, and recovery. The fault tolerance process is that set of activities whose goal is to remove errors and their effects from the computational state before a failure occurs. The process consists of:

• **Error detection**: in which an erroneous state is identified;

• **Error diagnosis**: in which the damage caused by the error is assessed and the cause of the error is determined;

• **Error containment/isolation**: in which further damages are prevented (or the error is prevented from propagating);

• **Error recovery**: in which the erroneous state is substituted with an error-free state. Error recovery is performed using backward recovery or forward recovery.

**Backward Recovery**

When an error occurs in a program, the program generally enters a contaminated or erroneous state. Recovery techniques attempt to return the system to a correct or error-free state. Backward recovery attempts to do this by restoring or rolling back the system to a previously saved state. It is usually assumed that the previously saved state occurred before the fault manifested itself, that is, that the prior state is error-free. If it is not error-free, the same error may cause problems in the recovery attempt. System states are saved at predetermined recovery points.

Recording or saving this previous state is called **check pointing**. The state should be check pointed on stable storage that will not be affected by failure. In addition to, and sometimes instead of, check pointing, incremental check pointing, audit trail, or logs may be used. Upon error detection, the system state is restored to the last saved state, and operations continue or restart from that state. If the failure occurred after the checkpoint was established, the check pointed state will be error-free and after the rollback, the system state will also be error-free. Figure 1.3 illustrates backward recovery.

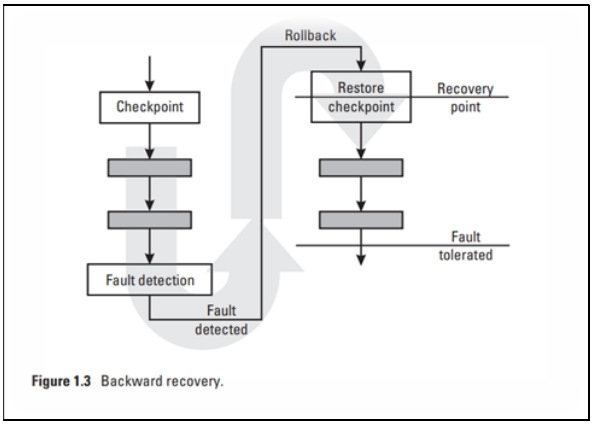


**Figure 1.3** Backward recovery.

**There are several advantages to backward recovery over other recovery techniques:**

• Backward recovery can handle unpredictable errors caused by residual design faults if the errors do not affect the recovery mechanism.

• Backward recovery can be used regardless of the damage sustained by the state.

• Backward recovery provides a general recovery scheme it has a uniform pattern of error detection and recovery, is application independent, and does not change from program to program.

• Backward recovery requires no knowledge of the errors in the system state.

• The only knowledge required by backward recovery is that the relevant prior state is error-free.

• Backward recovery can handle transparent or permanent arbitrary faults.

• Backward recovery is particularly suited to recovery of transient faults. After recovery, the error may have gone away, and restarting with the check pointed system state should not produce the same fault.

**There are, however, a few disadvantages to the backward recovery approach:**

- Backward recovery requires significant resources (i.e., time, computation, and stable storage) to perform check pointing and recovery.

• The implementation of backward recovery often requires that the system be halted temporarily.

• If communication and coordination of interacting processes using backward recovery (e.g., nested recovery blocks) are not synchronized, a domino effect may occur. This happens when one process rolls back to its previous

checkpoint, which in turn causes another process to roll further back (to its checkpoint), which in turn causes the first process to roll back further, and so on.

The backward recovery approach is the most generally applicable recovery technique for software fault tolerance. It is used frequently, despite its overhead. The RcB technique and most distributed systems incorporating software fault tolerance employ backward recovery.

**Forward Recovery**

As stated earlier, after an error occurs in a program, recovery techniques attempt to return the system to a correct or error-free state. Forward recovery attempts to do this by finding a new state from which the system can continue operation. This state may be a degraded mode of the previous error-free state. As an alternative to this state transitioning for recovery, forward recovery can utilize error compensation. Error compensation is based on an algorithm that uses redundancy (typically built into the system) to select or derive the correct answer or an acceptable answer. Figure 1.4 illustrates the forward-recovery concept. Note that redundant (diverse) software processes are executed in parallel. The redundancy provides a set of potential results or answers from which a fault detection and handling unit performs error compensation and selects or derives an answer deemed correct or acceptable. The NVP technique is based on the forward-recovery concept and uses a fault detection and handling unit typically called an adjudicator. Error compensation may be performed in several ways. If used with self-checking components, then state transformation can be induced by switching from a failed component to a non failed one executing the same task. Error compensation may be applied all the time, whether or not an error occurred. This is called **fault masking** and may be implemented using one of many available voting schemes.

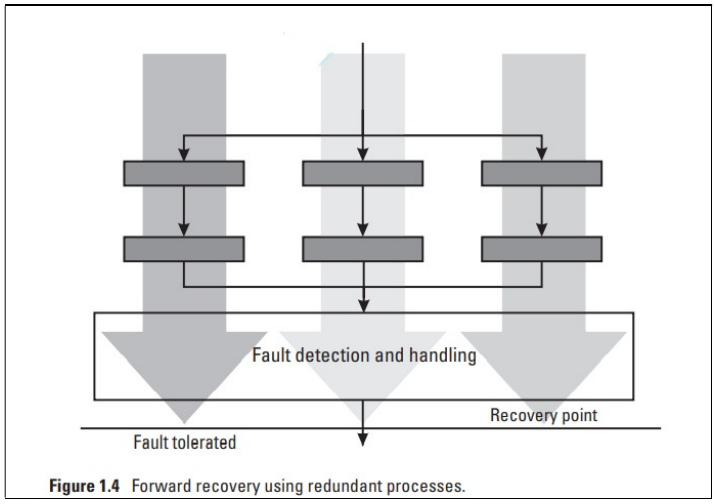


**Figure 1.4** Forward recovery using redundant processes.

**The advantages to forward recovery include:**

• Forward recovery is fairly efficient in terms of the overhead (time and memory) it requires. This can be crucial in real-time applications where the time overhead of backward recovery can exceed stringent time constraints.

• If the fault is an anticipated one, such as the potential loss of data, then redundancy and forward recovery can be a useful and timely approach.

• Faults involving missed deadlines may be better recovered from using forward recovery than by introducing additional delay in rolling back and recovering.

• When the characteristics of a fault are well understood, forward recovery can provide the more efficient solution.

**The disadvantages of forward recovery include:**

• Forward recovery is application-specific, that is, it must be tailored to each situation or program.

• Forward recovery can only remove predictable errors from the system state.

• Forward recovery requires knowledge of the error.

• Forward recovery cannot aid in recovery if the state is damaged beyond "specification wise recoverability.

• Forward recovery depends on the ability to accurately detect the occurrence of a fault (thus initiating the recovery actions), predict potential damage from a fault, and assess the actual damage. Forward recovery is primarily used when there is no time for backward recovery. Techniques using forward recovery include NVP, NCP, and the distributed recovery block (DRB) technique (which has the effect of forward recovery).