waits until it knows that the notification was received. As Figure 7-8 shows, this mode of operation leads to poor performance, especially when only a single processor is available. Xen instead uses *event channels* to send asynchronous notifications. Event channels implement virtual interrupts, but a virtual interrupt is serviced only when the target domain is next scheduled. Therefore, the requestor can generate multiple requests, raising the event channel each time, before the target domain is scheduled to act upon them. Then, when the target domain is scheduled, it can process several requests and send responses, again asynchronously.

Look at Figure 7-8. With synchronous notification, the frontend has to wait for the backend to complete its work before it can make the next request. This means waiting for the backend domain to be scheduled, and then for the frontend domain to be rescheduled. By contrast, with asynchronous notification the frontend can send as many requests as possible while it is scheduled, and the backend can send as many responses as possible. This leads to much-improved throughput.

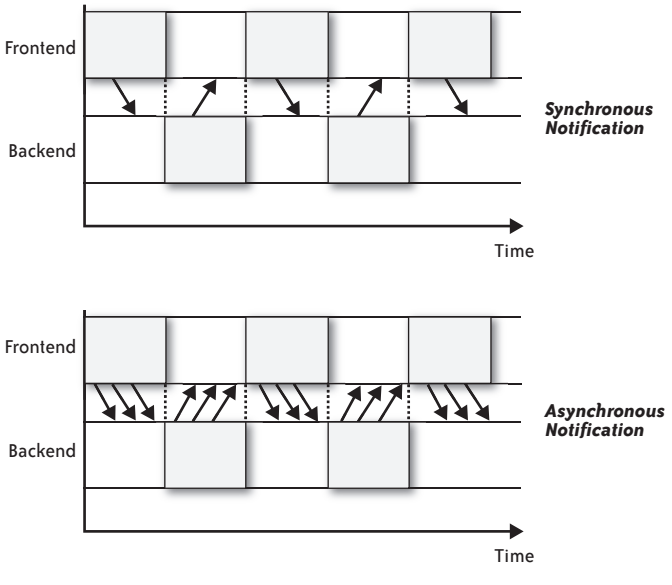

FIGURE 7-8. The advantages of asynchronous notification

Of course, if you move too much functionality into domain zero, it becomes a single point of failure. This is especially true of device failures, which can bring down the whole operating system (and with it, the entire virtualized system). Hence Xen allows for *driver domains*, to which domain zero can delegate the control of one or more devices. These are simply implemented by placing the backend driver in the driver domain and granting some I/O privileges to the domain. Then, if a driver should fail, the failure would be isolated within the driver domain, which can be restarted without harming the system or the client domain.