added in domain zero rather than in the hypervisor, where additional complexity is generally undesirable.

Earlier we noted how Xen benefited from the availability of an open source operating system, which provided a testbed for paravirtualization. A second benefit of using Linux is its vast range of support for different hardware devices. Xen is able to support almost any device for which a Linux driver exists, as it reuses the Linux driver code. Xen has always reused Linux drivers in order to support a variety of hardware. However, between versions 1.0 and 2.0, the nature of this reuse changed significantly.

In Xen 1.0, all virtual machines (including domain zero) accessed hardware through the virtual devices, as described in the previous section. The hypervisor was responsible for multiplexing these accesses onto real hardware, and therefore it contained ported versions of the Linux hardware drivers and the virtual driver backends. Although this simplified the virtual machines, it placed a lot of complexity in the hypervisor and put the burden of supporting new drivers on the Xen development team.

Figure 7-6 shows changes to the device architecture between Xen 1.0 and 2.0. In version 1.0, the virtual backends were implemented inside the hypervisor: all domains, including domain zero, accessed the hardware through these devices. In version 2.0, the hypervisor was slimmed down, and domain zero was given access to the hardware with native drivers. Therefore, the backend drivers moved to domain zero.

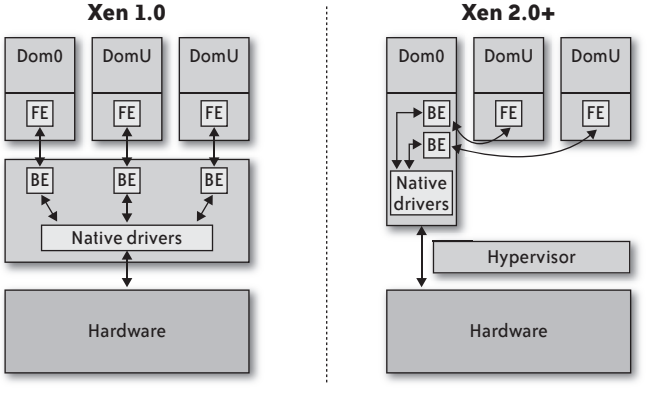

FIGURE 7-6. Changes to the device architecture between Xen 1.0 and 2.0

In the development of Xen 2.0, the device architecture was completely redesigned: the native device drivers and virtual backends were moved out of the hypervisor and into domain zero.[‖] The frontend and backend drivers now communicate using *device channels*, which

[‖] In fact, the architecture allows any authorized virtual machine to access the hardware, and therefore act as a *driver domain*.