## KILL THE OPERATING SYSTEM!

You use Windows, I use a Mac, and we both know people who use GNU/Linux. But for all the differences between these three families of computer operating systems, they implement the same fundamental design; all are equally powerful, and equally limiting.

Virtually every operating system in use today is based on a single computer system architecture developed in the 1960s and '70s. This architecture divides code running on computers into a "kernel," responsible for controlling the computer's hardware, and so-called application programs, which are loaded into the computer's memory to perform individual tasks. Applications, in turn, operate on named files arranged in a tree of folders. True, there are a few niche operating systems that don't adhere to this tripartite structure, but they are but bit players on the digital stage. Even PalmOS has a kernel, apps, and files (which PalmOS mistakenly calls "databases"). It's almost inconceivable that this approach won't be the dominant paradigm for many years to come. And that's a deep problem for the future of computing.

Hollywood, though, has a better idea. When computers show up in good science fiction movies, they rarely have interfaces with windows, icons, applications, and files. Instead, Hollywood's systems let people rapidly navigate through a sea of information and quickly address their needs. Some technical folks scoff at this representation as unrealistic. But why is that so?

Computing's standard model owes its success to the economics of the computer industry. The first computer programs were monolithic systems that talked to the hardware, communicated with users, and got the job done. But soon it became clear that organizations were spending far more money on software, custom software development, and training then they would ever spend on hardware alone. These businesses wanted guarantees that the programs they were creating would run on next year's computer. The only way to assure this was to take all of the hardware-specific code and put it into some kind of "supervisor" program—what we now call the kernel. The supervisor evolved into a kind of traffic cop that could allow multiple programs to run on the same computer at the same time without interfering with one another. That was vital back in the day when a single computer might have dozens of simultaneous users. It's equally important today for people who run dozens of programs simultaneously on their desktop systems.

But you could imagine building computers differently. Movie directors have pointed the way, showing interfaces that appear to make all of the computer's data and power always instantly available. Achieving such flexibility, however, would require us to rethink operating-system dogma. For example, instead of isolating applications from each other—where transferring data between them requires cutting, pasting, and usually reformatting—a hypothetical computer might run all programs at the same time and in the same workspace. Programs might not display information in their own distinct windows, the way they do now; instead, they would work behind the scenes, contributing as needed to a common display.

Most people can't imagine how such a system would work. The idea of editing an Adobe Illustrator document with Microsoft Word seems nonsensical: one program is designed for drawings, the other for words—and besides, they're made by different companies! Yet many Illustrator documents contain blocks of text: why not use Word's superior text-editing capabilities? In our imagined new computer, the boundaries between applications would melt away.

Computer scientists periodically experiment with systems that do away with the software barriers on which today's computers are based, but these systems are rarely successful in the marketplace. Both the Lisp Machine and the Canon Cat encouraged developers to create programs that ran in the same workspace, rather than dividing the computer up into different applications. The Apple Newton stored information not in files but in "soups"—little object-oriented databases that could be accessed by many different programs, even at the same time. The commercial failure of these systems does not vindicate today's way of computing but rather is testimony to just how dangerous the dominant-paradigm trap actually is.

**For all their apparent differences, the three great families of computer operating systems—Windows, MacOS, and GNU/Linux—are all equally limiting. But Hollywood has a better idea.**

Consider files and directories. The hierarchical directory system used by Windows, MacOS, and Unix made sense to computer pioneers who grew up using paper and filing cabinets. But why limit today's computers with 40-year-old metaphors? Computers have fantastic search capabilities. Some documents logically belong in multiple places; why not eliminate the folders and store all of the computer's information in one massive data warehouse? That's the way computers in the movies seem to work.

It's not such a far-fetched notion. It wouldn't take much to enable today's computers to store every version of every document they have ever been used to modify: most people perform fewer than a million keystrokes and mouse clicks each day; a paltry four gigabytes could hold a decade's worth of typing and revisions if we stored those keystrokes directly, rather than using the inefficient Microsoft Word document format. Alas, the convenient abstractions of directories and files make it difficult for designers to create something different.With a little thought, though, we could do far better. Hollywood has dreamed it; now Silicon Valley needs to make it real. TR