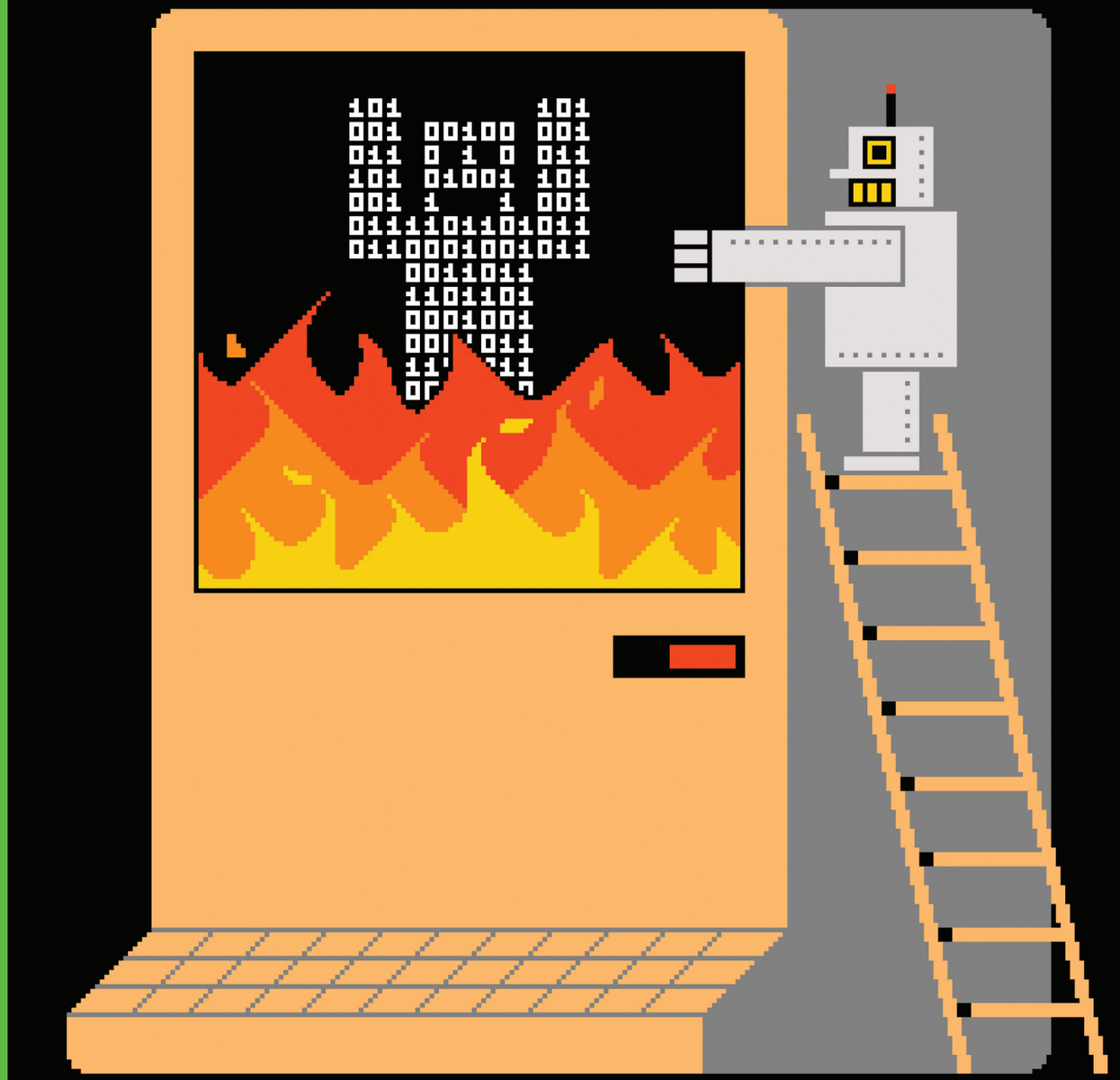
SAVING SOFTWARE FROM OBLIVION

A PROTOTYPE ARCHIVING SYSTEM LETS VINTAGE CODE RUN ON TODAY'S COMPUTERS

In early 2010, Harvard economists Carmen Reinhart and Kenneth Rogoff published an analysis of economic data from many countries and concluded that when debt levels exceed 90 percent of gross national product, a nation's economic growth is threatened. With debt that high, expect growth to become negative, they argued. ■ This analysis was done shortly after the 2008 recession, so it had enormous relevance to policymakers, many of whom were promoting high levels of debt spending in the interest of stimulating their nations' economies. At the same time, conservative politicians, such as Olli Rehn, then an EU commissioner, and U.S. congressman Paul Ryan, used Reinhart and Rogoff's findings to argue for fiscal austerity. ■ Three years later, Thomas Herndon, a graduate student at the University of Massachusetts, discovered an error in the Excel spreadsheet that Reinhart and Rogoff had used to make their calculations. The significance of the blunder was enormous: When the analysis was done properly, Herndon showed, debt levels in excess of 90 percent were associated with average growth of positive 2.2 percent, not the negative 0.1 percent that Reinhart and Rogoff had found. ■ Herndon could easily test the Harvard economists' conclusions because the software that they had used to calculate their results—Microsoft Excel—was readily available. But what about much older findings for which the software originally used is hard to come by? ■ You might think that the solution—preserving the relevant software for future researchers to use—should be no big deal. After all, software is nothing more than a bunch of files, and those files are easy enough to store on a hard drive or on tape in digital format. For some software at least, the all-important source code could even be duplicated on paper, avoiding the possibility that whatever digital medium it's written to could become obsolete. ■ Saving old programs in this way is done routinely, even for decades-old software. You can find online, for example, a full program listing for the Apollo Guidance Computer—code that took astronauts to the moon during the 1960s. It was transcribed from a paper copy and uploaded to GitHub in 2016.

By Mahadev Satyanarayanan
Illustrations by Nicholas Little



While perusing such vintage source code might delight hard-core programmers, most people aren't interested in such things. What they want to do is *use* the software. But keeping software in ready-to-run form over long periods of time is enormously difficult, because to be able to run most old code, you need both an old computer and an old operating system.

You might have faced this challenge yourself, perhaps while trying to play a computer game from your youth. But being unable to run an old program can have much more serious repercussions, particularly for scientific and technical research.

Along with economists, many other researchers, including physicists, chemists, biologists, and engineers, routinely use software to slice and dice their data and visualize the results of their analyses. They simulate phenomena with computer models that are written in a variety of programming languages and that use a wide range of supporting software libraries and reference data sets. Such investigations and the software on which they are based are central to the discovery and reporting of new research results.

Imagine that you're an investigator and want to check calculations done by another researcher 25 years ago. Would the relevant software still be around? The company that made it may have disappeared. Even if a contemporary version of the software exists, will it still accept the format of the original data? Will the calculations be identical in every respect—for example, in the handling of rounding errors—to those obtained using a computer of a generation ago? Probably not.

Researchers' growing dependence on computers and the difficulty they encounter when attempting to run old software are hampering their ability to check published results. The problem of obsolescent software is thus eroding the very premise of reproducibility—which is, after all, the bedrock of science.

The issue also affects matters that could be subject to litigation. Suppose, for example, that an engineer's calculations show that a building design is robust, but the roof of that building nevertheless collapses. Did the engineer make a mistake, or was the software used for the calculations faulty? It would be hard to know years later if the software could no longer be run.

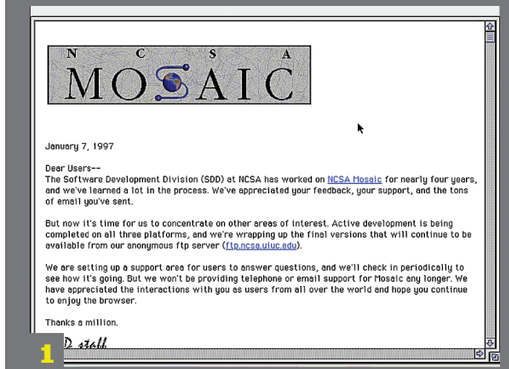
That's why my colleagues and I at Carnegie Mellon University, in Pittsburgh, have been developing ways to archive programs in forms that can be run easily today and into the future. My fellow computer scientists Benjamin Gilbert and Jan Harkes did most of the required coding. But the collaboration has also involved software archivist Daniel Ryan and librarians Gloriana St. Clair, Erika Linke, and Keith Webster, who naturally have a keen interest in properly preserving this slice of modern culture.

Because this project is more one of archival preservation than mainstream computer science, we garnered

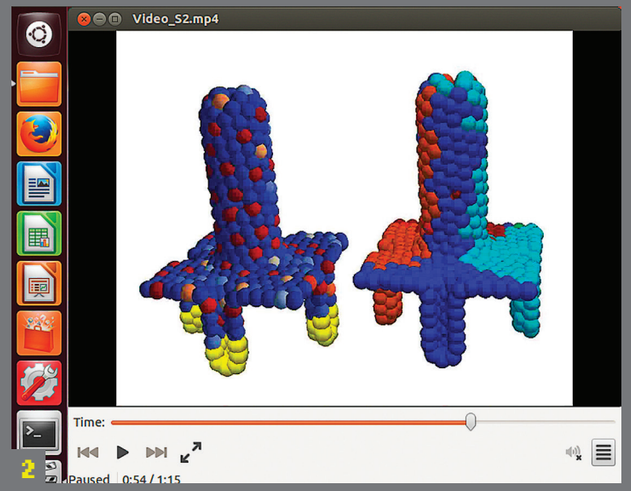
BRINGING BACK YESTERDAY'S SOFTWARE

The Olive system has been used to create 17 different virtual machines that run a variety of old software, some serious, some just for fun. Here are several views from those archived applications.

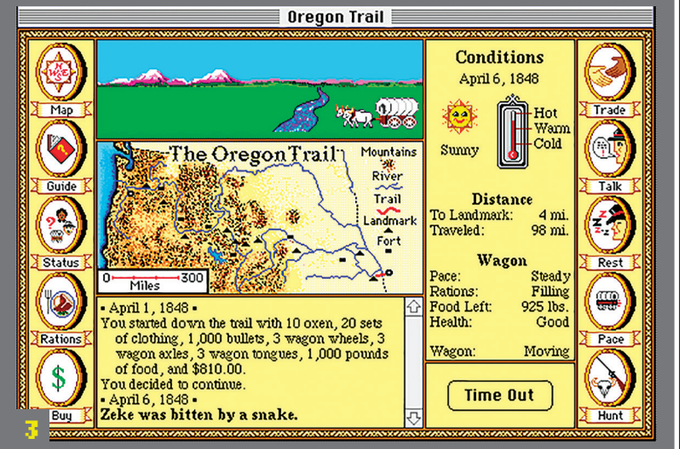
1. NCSA Mosaic 1.0, a pioneering Web browser for the Macintosh from 1993 2. Chaste (Cancer, Heart and Soft Tissue Environment) 3.1 for Linux from 2013
3. *The Oregon Trail* 1.1, a game for the Macintosh from 1990 4. *Wanderer*, a game for MS-DOS from 1988
5. *Mystery House*, a game for the Apple II from 1982
6. The Great American History Machine, an educational interactive atlas for Windows 3.1 from 1991 7. Microsoft Office 4.3 for Windows 3.1 from 1994 8. ChemCollective, educational chemistry software for Linux from 2013



1



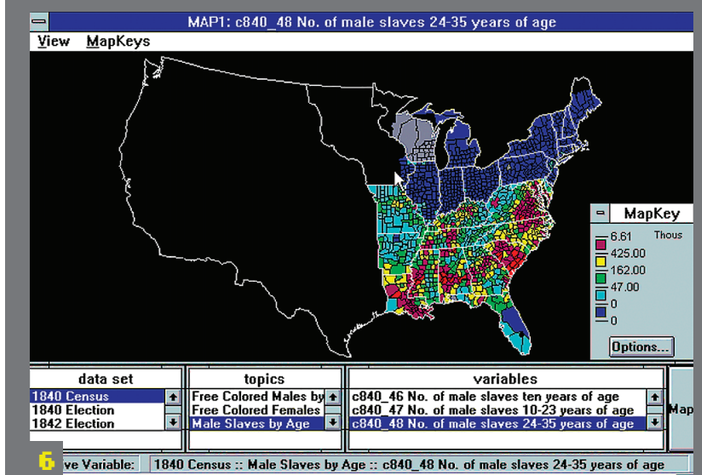
2



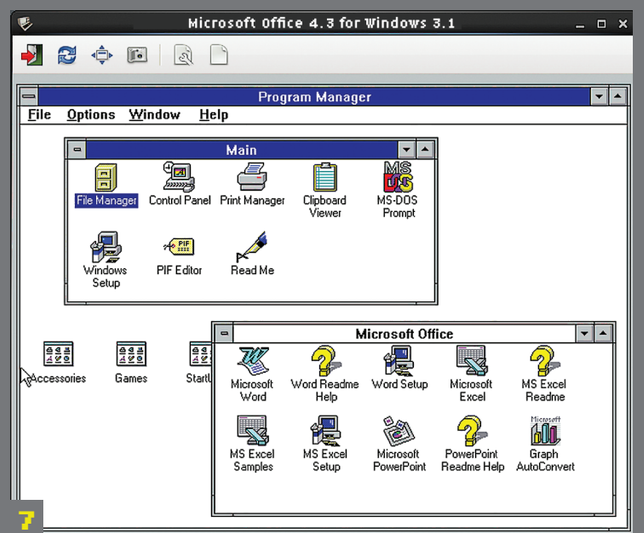
3

4

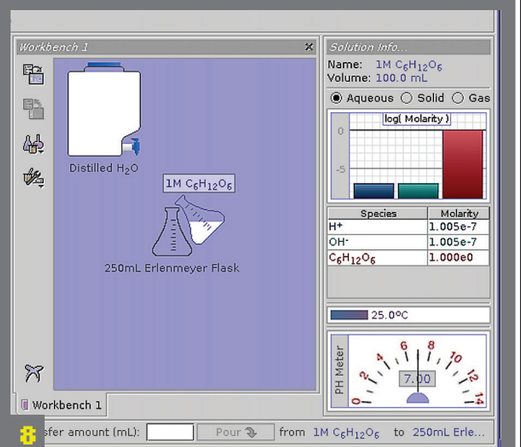
5



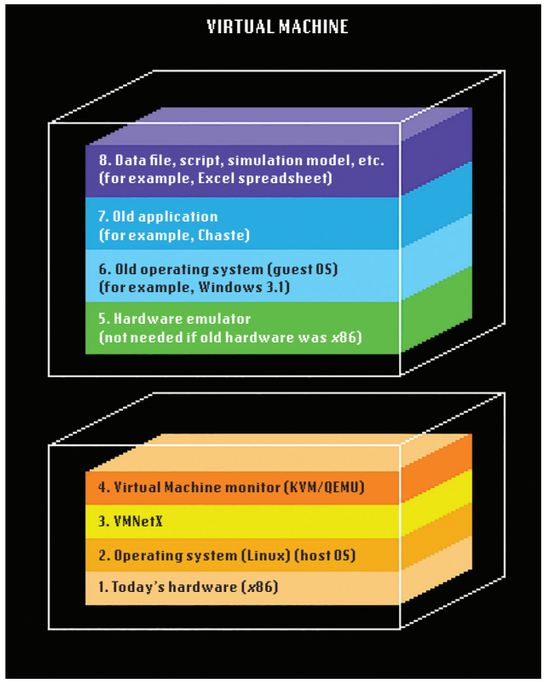
6



7



8



LAYERS OF ABSTRACTION: Olive requires many layers of software abstraction to create a suitable virtual machine. That virtual machine then runs the old operating system and application.

financial support for it not from the usual government funding agencies for computer science but from the Alfred P. Sloan Foundation and the Institute for Museum and Library Services. With that support, we showed how to reconstitute long-gone computing environments and make them available online so that any computer user can, in essence, go back in time with just a click of the mouse.

We created a system called Olive—an acronym for Open Library of Images for Virtualized Execution. Olive delivers over the Internet an experience that in every way matches what you would have obtained by running an application, operating system, and computer from the past. So once you install Olive, you can interact with some very old software as if it were brand new. Think of it as a Wayback Machine for executable content.

TO UNDERSTAND HOW OLIVE can bring old computing environments back to life, you have to dig through quite a few layers of software abstraction. At the very bottom is the common base of much of today's computer technology: a standard desktop or laptop endowed with one or more *x*86 microprocessors. On that computer, we run the Linux operating system, which forms the second layer in Olive's stack of technology.

Sitting immediately above the operating system is software written in my lab called VMNetX, for Virtual Machine Network Execution. A virtual machine is a computing environment that mimics one kind of computer using software running on a different kind of computer. VMNetX is special in that it allows virtual machines to be stored on a central server and then executed on demand by a remote system. The advantage of this arrangement is that your computer doesn't need to download the virtual machine's entire disk and memory state from the server before running that virtual machine. Instead, the information stored on disk and in memory is retrieved in chunks as needed by the next layer up: the virtual-machine monitor (also called a hypervisor), which can keep several virtual machines going at once.

Each one of those virtual machines runs a hardware emulator, which is the next layer in the Olive stack. That emulator presents the illusion of being a now-obsolete computer—for example, an old Macintosh Quadra with its 1990s-era Motorola 68040 CPU. (The emulation layer can be omitted if the archived software you want to explore runs on an *x*86-based computer.)

The next layer up is the old operating system needed for the archived software to work. That operating system has access to a virtual disk, which mimics actual disk storage, providing what looks like the usual file system to still-higher components in this great layer cake of software abstraction.

Above the old operating system is the archived program itself. This may represent the very top of the heap, or there could be an additional layer, consisting of data that must be fed to the archived application to get it to do what you want.

The upper layers of Olive are specific to particular archived applications and are stored on a central server. The lower layers are installed on the user's own computer in the form of the Olive client software package. When you launch an archived application, the Olive client fetches parts of the relevant upper layers as needed from the central server.

THAT'S WHAT YOU'LL FIND under the hood. But what can Olive do? Today, Olive consists of 17 different virtual machines that can run a variety of operating systems and applications. The choice of what to include in that set was driven by a mix of curiosity, availability, and personal interests. For example, one member of our team fondly remembered playing *The Oregon Trail* when he was in school in the early 1990s. That led us to acquire an old Mac version of the game and to get it running again through Olive. Once word of that accomplishment got out, many people started approaching us to see if we could resurrect their favorite software from the past.

The oldest application we've revived is *Mystery House*, a graphics-enabled game from the early 1980s for the Apple II computer. Another program is NCSA Mosaic, which people of a certain age might remember as the browser that introduced them to the wonders of the World Wide Web.

Olive provides a version of Mosaic that was written in 1993 for Apple's Macintosh System 7.5 operating system. That operating system runs on an emulation of the Motorola 68040 CPU, which in turn is created by software running on an actual *x*86-based computer that runs Linux. In spite of all this virtualization, performance is pretty good, because modern computers are so much faster than the original Apple hardware.

Pointing Olive's reconstituted Mosaic browser at today's Web is instructive: Because Mosaic predates Web technologies such as JavaScript, HTTP 1.1, Cascading Style Sheets, and HTML 5, it is unable to render most sites. But you can have some fun tracking down websites composed so long ago that they still look just fine.

What else can Olive do? Maybe you're wondering what tools businesses were using shortly after Intel introduced the Pentium processor. Olive can help with that, too. Just fire up Microsoft Office 4.3 from 1994 (which thankfully predates the annoying automated office assistant "Clippy").

Perhaps you just want to spend a nostalgic evening playing *Doom* for DOS—or trying to understand what made such

first-person shooter games so popular in the early 1990s. Or maybe you need to redo your 1997 taxes and can't find the disk for that year's version of TurboTax in your attic. Have no fear: Olive has you covered.

On the more serious side, Olive includes Chaste 3.1. The name of this software is short for Cancer, Heart and Soft Tissue Environment. It's a simulation package developed at the University of Oxford for computationally demanding problems in biology and physiology. Version 3.1 of Chaste was tied to a research paper published in March 2013. Within two years of publication, though, the source code for Chaste 3.1 no longer compiled on new Linux releases. That's emblematic of the challenge to scientific reproducibility Olive was designed to address.

To keep Chaste 3.1 working, Olive provides a Linux environment that's frozen in time. Olive's re-creation of Chaste also contains the example data that was published with the 2013 paper. Running the data through Chaste produces visualizations of certain muscle functions. Future physiology researchers who wish to explore those visualizations or make modifications to the published software will be able to use Olive to edit the code on the virtual machine and then run it.

For now, though, Olive is available only to a limited group of users. Because of software-licensing restrictions, Olive's collection of vintage software is currently accessible only to people who have been collaborating on the project. The relevant companies will need to give permissions to present Olive's re-creations to broader audiences.

We are not alone in our quest to keep old software alive. For example, the Internet Archive is preserving thousands of old programs using an emulation of MS-DOS that runs in the user's browser. And a project being mounted at Yale, called EaaSI (Emulation as a Service Infrastructure), hopes to make available thousands of emulated software environments from the past. The scholars and librarians involved with the Software Preservation Network have been coordinating this and similar efforts. They are also working to address the copyright issues that arise when old software is kept running in this way.

OLIVE HAS COME A LONG WAY, but it is still far from being a fully developed system. In addition to the problem of restrictive software licensing, various technical roadblocks remain.

One challenge is how to import new data to be processed by an old application. Right now, such data has to be entered manually, which is both laborious and error prone. Doing so also limits the amount of data that can be analyzed. Even if we were to add a mechanism to import data, the amount that could be saved would be limited to the size of the virtual machine's virtual disk. That may not seem like a problem, but you have to remember that the file systems on older computers sometimes had what now seem like quaint limits on the amount of data they could store.

Another hurdle is how to emulate graphics processing units (GPUs). For a long while now, the scientific community has been leveraging the parallel-processing power of GPUs to speed up many sorts of calculations. To archive executable versions of software that takes advantage of GPUs, Olive would need to re-create virtual versions of those chips, a thorny task. That's because GPU interfaces—what gets input to them and what they output—are not standardized.

Clearly there's quite a bit of work to do before we can declare that we have solved the problem of archiving executable content. But Olive represents a good start at creating the kinds of systems that will be required to ensure that software from the past can live on to be explored, tested, and used long into the future. ■

↗ POST YOUR COMMENTS at https://spectrum.ieee.org/olive1018