Tools and techniques for computational reproducibility

Stephen R. Piccolo[1*], Adam B. Lee[1], Michael B. Frampton[2]

Department of Biology, Brigham Young University, Provo, UT, USA

Department of Computer Science, Brigham Young University, Provo, UT, USA

* Corresponding author: 4102 LSB, Brigham Young University, Provo, UT, 84602; 801-422-7116; Stephen_Piccolo@byu.edu

Abstract

When reporting research findings, scientists document the steps they followed so that others can verify and build upon the research. When those steps have been described in sufficient detail that others can retrace the steps and obtain similar results, the research is said to be reproducible. Computers play a vital role in many research disciplines and present both opportunities and challenges for reproducibility. Computers can be programmed to execute analysis tasks, and those programs can be repeated and shared with others. Due to the deterministic nature of most computer programs, the same analysis tasks, applied to the same data, will often produce the same outputs. However, in practice, computational findings often cannot be reproduced, due to complexities in how software is packaged, installed, and executed—and due to limitations in how scientists document analysis steps. Many tools and techniques are available to help overcome these challenges. Here we describe six such strategies. With a broad scientific audience in mind, we describe strengths and limitations of each approach, as well as circumstances under which each might be applied. No single strategy is sufficient for every scenario; thus we emphasize that it is often useful to combine approaches.

Introduction

When reporting a research study, scientists document the steps they followed to obtain their results. If the description is comprehensive enough that they and others can repeat the procedures and obtain semantically consistent results, the findings are considered to be "reproducible"[1–6]. Reproducible research forms the basic building blocks of science, insofar as it allows researchers to verify and build on each other's work with confidence.

Computers play an increasingly important role in many scientific disciplines[7]. For example, in the United Kingdom, 92% of academic scientists use some type of software in their research, and 69% of scientists say their research is feasible only with software tools[8]. Thus efforts to increase scientific reproducibility should consider the ubiquity of computers in research.

Computers present both opportunities and challenges for scientific reproducibility. On one hand, due to the deterministic nature of most computer programs, computational analyses can be performed such that others can obtain exactly identical results when applied to the same input data[9]; accordingly, computational research can be held to a higher reproducibility standard than other types of research. On the other hand, in practice, scientists often cannot reproduce computational findings due to complexities in how software is packaged, installed, and executed—and due to limitations in how scientists document these steps[10]. This problem is acute in many disciplines, including genomics, signal processing, and ecological modeling[11–13], where data sets are large and computational tools are

evolving rapidly. However, the same problem can affect any scientific discipline that requires computers for research, irrespective of data type or size. Seemingly minor differences in computational approaches can have major influences on analytical outputs[9,14–19]. The effects of these differences may meet or exceed those that result from experimental factors[20].

Journal editors, funding agencies, governmental institutions, and individual scientists have increasingly made calls for the scientific community to embrace practices that support computational reproducibility[21–28]. This movement has been motivated, in part, by scientists' failed efforts to reproduce previously published analyses. For example, Ioannidis, et al. evaluated 18 published research studies that used computational methods to evaluate gene-expression data but were able to reproduce only 2 of those studies[29]. In many cases, failure to share the study's data was the culprit; however, incomplete descriptions of software-based analyses were also common. Nekrutenko and Taylor examined 50 papers that analyzed next-generation sequencing data and observed that fewer than half provided any details about software versions or parameters[30]. Recreating analyses that lack such details can require hundreds of hours of effort[31,32] and may be impossible, even after consulting the original authors. Worse, failure to reproduce research can lead to retractions[33,34].

Noting such concerns, some journals have emphasized the value of placing computer source code in open-access repositories, such as *GitHub* (http://www.github.com) or *BitBucket* (http://www.bitbucket.org). In addition,

journals have extended requirements for "Methods" sections, now asking researchers to provide detailed descriptions of 1) how to install software and its dependencies and 2) what parameters and data-preprocessing steps were used in analyses[7,21]. A recent Institute of Medicine report emphasized that, in addition to computer code and research data, "fully specified computational procedures" should be made available to the scientific community[22]. They elaborated that such procedures should include "all of the steps of computational analysis" and that "all aspects of the analysis need to be transparently reported"[22]. Such policies represent important progress in the quest for better policies. However, it is ultimately the responsibility of individual scientists to ensure that others can verify and build upon their analyses.

Describing a computational analysis sufficiently—such that others can reexecute it, validate it, and refine it—requires more than simply stating what software was used, what commands were executed, and where to find the source code[10,24,35–37]. Software is executed within the context of an operating system (for example, *Windows*, *Mac OS*, or *Linux*), which enables the software to interface with computer hardware (Figure 1). In addition, most software relies on a hierarchy of software dependencies, which perform complementary functions and must be installed alongside the main software tool. One version of a given software tool or dependency may behave differently or have a different interface than another version of the same software. In addition, most analytical software offers a range of parameters (or settings) that the user can specify. If any of these variables differs from what the original experimenter used, the software may not execute properly or

analytical outputs may differ considerably from what the original experimenter observed.

Scientists can use various tools and techniques to overcome these challenges and to increase the likelihood that their computational analyses will be reproducible. These techniques range in complexity from simple (e.g., providing written documentation) to advanced (e.g., providing a "virtual" environment that includes an operating system and all software necessary to execute the analysis). This review describes six strategies across this spectrum. We describe strengths and limitations of each approach, as well as circumstances under which each might be applied. No single strategy will be sufficient for every scenario; therefore, in many cases, it will be most practical to combine multiple approaches. This review focuses primarily on the computational aspects of reproducibility. The related topics of empirical reproducibility, statistical reproducibility, and data sharing have been described elsewhere[38–44]. We believe that with greater awareness and understanding of computational-reproducibility techniques, scientists—including those with limited computational experience—will be more apt to perform computational research in a reproducible manner.

Narrative descriptions are a simple but valuable way to support computational reproducibility

The most fundamental strategy for enabling others to reproduce a computational analysis is to provide a detailed, written description of the process. When reporting

116 computational results in a research article, authors customarily provide a narrative
117 that describes the software they used and the analytical steps they followed. Such
118 narratives can be invaluable in enabling others not only to evaluate the scientific
119 approach but also to reproduce the findings. In many situations—for example, when
120 software execution requires interaction from the user or when proprietary software
121 is used—narratives are the only feasible option for documenting such steps.
122 However, even when a computational analysis uses open-source software and can
123 be fully automated, narratives help others understand how to execute the analysis.

124 Although most research articles that use computational methods provide some type
125 of narrative, these descriptions often lack sufficient detail to enable others to retrace
126 those steps[29,30]. To ensure that others can reproduce a computational analysis,
127 narrative descriptions should indicate the operating system(s), software
128 dependencies, and analytical software that were used and where to obtain them. In
129 addition, narratives should indicate the exact software versions used, the order in
130 which they were executed, and all non-default parameters that were specified. Such
131 descriptions should account for the fact that computer configurations differ vastly,
132 even for computers that use the same operating system. A limitation of narratives is
133 that it can be difficult to remember such details post hoc; thus the documentation
134 process will be most efficient when scientists record these steps throughout the
135 research process, rather than at the time of manuscript preparation.

136 The following sections describe techniques for automating software execution and
137 thus characterizing analytical steps in computer-readable formats. These techniques

can diminish the need for scientists to write narratives. However, because it is often not practical to automate all computational steps, we expect that, for the foreseeable future, narratives will play a vital role in enabling computational reproducibility.

Computer code and scripts can automate the analysis process

Scientific software can often be executed in an automated manner via text-based commands. In these cases, no amount of narrative can substitute for providing the actual commands that were used to automate the analysis. Using a command-line interface, a scientist can indicate which software program(s) to execute and which parameter(s) to use. When multiple commands must be executed, they can be compiled into scripts, which specify the order in which the commands should be executed and whether they can be executed in parallel. In many cases, scripts also include commands for installing and configuring software. Such scripts serve as valuable documentation not only for individuals who wish to reexecute the analysis but also for the researcher who performed the original analysis.

When writing command-line scripts, it is essential to document any software dependencies and input data required for each step in the analysis. The *Make* utility[45] provides one way to specify such requirements[35]. Before any command is executed, *Make* verifies that each documented dependency is available. Accordingly, researchers can use *Make* to specify the full hierarchy of operating-system components and dependent software that must be present. In addition, *Make* can be configured to automatically identify any commands that can be executed in parallel,

159 potentially reducing the amount of time required to execute the analysis. Although
160 *Make* was designed originally for UNIX-based operating systems (such as *Mac OS* or
161 *Linux*), similar utilities have since been developed for *Windows* operating systems[46].
162 Box 1 lists various other utilities that can be used to automate software execution.

163 In many scientific analyses, authors write custom computer code. Such code may
164 perform relatively simple tasks, such as reformatting data files or invoking third-
165 party software libraries. In other cases, code may constitute a manuscript's key
166 intellectual contribution. In either situation, when authors provide code alongside a
167 manuscript, readers can evaluate the authors' computational approach in full
168 detail[47]. A common way to manage code is to place it in a source-control repository
169 such as *Git* (http://git-scm.com) or *Mercurial* (http://mercurial.selenic.com). Such
170 repositories then can be shared via openly accessible services like *GitHub*
171 (https://github.com) or *Bitbucket* (https://bitbucket.org/), along with a full history
172 of changes that have been made to the code. Other researchers may then access
173 previous versions of the code, extend the code, and contribute revisions[48].

174 Although it is common for code to be published as a standalone software package,
175 much can be gained from incorporating code into a preexisting software framework.
176 *Bioconductor*[49], written in the *R* statistical programming language[50], is a popular
177 framework that contains hundreds of software packages for analyzing genomic
178 data[51]. The *Bioconductor* framework facilitates the processes of versioning,
179 documenting, and distributing code. Once computer code has been incorporated
180 into a *Bioconductor* software package, other researchers can find, download, install,

181 and configure it on most operating systems with relative ease. In addition,
182 *Bioconductor* installs software dependencies automatically. These features ease the
183 process of performing a scientific analysis and enabling other scientists to
184 reproduce the work. Various software frameworks exist for other scientific
185 disciplines[52–57].

186

Box 1: Utilities that can be used to automate software execution.

- *GNU Make*[45] and Make for Windows[46]: Tools for building software from source files and for ensuring that the software's dependencies are met.
- *Snakemake*[58] = An extension of Make that provides a more flexible syntax and makes it easier to execute tasks in parallel.
- *BPipe*[59] = A tool that provides a flexible syntax for users to specify commands to be executed; it maintains an audit trail of all commands that have been executed.
- *GNU Parallel*[60] = A tool for executing commands in parallel across one or more computers.
- *Makeflow*[61] = A tool that can execute commands simultaneously on various types of computer architectures, including computer clusters and cloud environments.

187

188 Literate programming combines narratives directly with code

189 Although computer code and narratives support reproducibility individually,
190 additional value can be gained from combining these entities. Even though
191 computer code may be provided alongside a research article, other scientists may
192 have difficulty interpreting how the code accomplishes its scientific objectives. A
193 longstanding way to address this problem is via code comments, which are human-

readable annotations interspersed throughout computer code. Going a step further, scientists can use a technique called *literate programming*[62]. With this approach, the scientist writes a narrative of the scientific analysis and intermingles code directly within the narrative. As the code is executed, a document is created that includes the code, narratives, and any outputs that the code produces. Accordingly, literate programming helps ensure that readers understand exactly how a particular result was obtained. In addition, this approach motivates the scientist to keep the target audience in mind when performing a computational analysis, rather than simply to write code that a computer can parse[62]. Consequently, by reducing barriers of understanding among scientists, literate programming can help to engender greater trust in computational findings.

One popular literate-programming tool is *IPython*[63]. Using its Web interface, scientists can create interactive "notebooks" that combine code, data, mathematical equations, plots, and rich media[64]. As its name implies, *IPython* was designed originally for the *Python* programming language; however, a recent iteration of the tool, *Jupyter* (https://jupyter.org), makes it possible to execute code in a variety of programming languages. Such functionality may be important to scientists who prefer to combine the strengths of different programming languages.

knitr[65] has also gained considerable popularity as a literate-programming tool. It is written in the *R* programming language and thus can be integrated seamlessly with the array of statistical and plotting tools available in that environment. However, like *IPython*, *knitr* can execute code written in multiple programming languages.

Commonly, *knitr* is applied to documents that have been authored using *RStudio* (http://www.rstudio.com), an open-source tool with advanced editing and package-management features.

IPython notebooks and *knitr* reports can be saved in various output formats, including HTML and PDF. Increasingly, scientists include such documents with journal manuscripts as supplementary material, enabling others to repeat analysis steps and recreate manuscript figures[66–69].

Literate-programming tools are suitable for applied research because they enable scientists to apply computational methods in specific scenarios. However, scientists often desire to generalize code so it can be applied in additional contexts. Current literate-programming tools may not be well suited to extensive software development and testing. However, they can aid in the process of illustrating scenarios in which the software might be applied.

Workflow-management systems enable reproducible software execution via a graphical user interface

Writing computer code and scripts may seem daunting to many researchers. Although various courses and tutorials are helping to make this task less formidable[70–73], many scientists use "workflow management systems" to facilitate the process of executing scientific software[74]. Typically managed via a Web interface, a workflow management system enables scientists to upload data and process it using existing tools. For multistep analyses, the output from one tool can

237 be used as input to another tool, resulting in a series of commands known as a
238 workflow.

239 *Galaxy*[75,76] has gained considerable popularity within the bioinformatics
240 community—especially for performing next-generation sequencing analysis. As
241 users construct workflows, *Galaxy* provides descriptions of how software
242 parameters should be used, examples of how input files should be formatted, and
243 links to relevant discussion forums. To help with processing large data sets and
244 computationally complex algorithms, *Galaxy* also provides an option to execute
245 workflows on cloud-computing services[77]. In addition, researchers can share
246 workflows with each other (see https://usegalaxy.org/workflow/list_published);
247 this feature has enabled the *Galaxy* team to build a community that helps to
248 encourage reproducibility, define best practices, and reduce the time required for
249 novices to get started.

250 Various other workflow systems are freely available to the research community (see
251 Box 2). For example, *VisTrails* is used by researchers from many disciplines,
252 including climate science, microbial ecology, and quantum mechanics[78]. It enables
253 scientists to design workflows visually, connecting data inputs with analytical
254 modules and the resulting outputs. In addition, *VisTrails* tracks a full history of how
255 each workflow was created. This capability, referred to as "retrospective
256 provenance", makes it possible for others not only to reproduce the final version of
257 an analysis but also to examine previous incarnations of the workflow and examine
258 how each change influenced analytical outputs[79].

Box 2: Workflow management tools freely available to the research community.

- Galaxy[75,76] - https://usegalaxy.org
- VisTrails[78] - http://www.vistrails.org
- Kepler[80] - https://kepler-project.org
- iPlant Collaborative[81] - http://www.iplantcollaborative.org
- GenePattern[82,83] - http://www.broadinstitute.org/cancer/software/genepattern
- Taverna[84] - http://www.taverna.org.uk
- LONI Pipeline[85] - http://pipeline.bmap.ucla.edu

Although workflow-management systems offer many advantages, users must accept tradeoffs. For example, although the teams that develop these tools often provide public servers where users can execute workflows, many scientists share these limited resources, so the public servers may not have adequate computational power or storage space to execute large-scale analyses in a timely manner. As an alternative, many scientists install these systems on their own computers; however, configuring and supporting them requires time and expertise. In addition, if a workflow tool does not yet provide a module to support a given analysis, the scientist must create a new module to support it. This task constitutes additional overhead; however, utilities such as the *Galaxy Tool Shed* (https://toolshed.g2.bx.psu.ed) are helping to facilitate this process.

Virtual machines encapsulate an operating system and software dependencies

Whether an analysis is executed at the command line, within a literate-programming notebook, or via a workflow-management system, an operating system and software dependencies must be installed before the analysis can be performed. The process of identifying, installing, and configuring such dependencies consumes a considerable amount of scientists' time. Different operating systems (and versions thereof) may require different installation and configuration steps. Furthermore, earlier versions of software dependencies, which may currently be installed on a given computer, may be incompatible with—or produce different outputs than—newer versions.

One solution is to use virtual machines, which can encapsulate an entire operating system and all software, scripts, and code necessary to execute a computational analysis[86,87] (Figure 2). Using virtualization software—such as *VirtualBox* or *VMWare* (see Box 3)—a virtual machine can be executed on practically any desktop, laptop, or server, irrespective of the main ("host") operating system on the computer. For example, even though a scientist's computer may be running a *Windows* operating system, the scientist may perform an analysis on a *Linux* operating system that is running concurrently—within a virtual machine—on the same computer. The scientist has full control over the virtual ("guest") operating system and thus can install software and modify configuration settings as necessary. In addition, a virtual machine can be constrained to use limited computational

292 resources (e.g., computer memory, processing power); thus multiple virtual
293 machines can be executed simultaneously on the same computer without impacting
294 each other's performance. After executing an analysis, the scientist can export the
295 entire virtual machine to a single, binary file. Other scientists can then use this file to
296 reconstitute the same computational environment that was used for the original
297 analysis. With a few exceptions (see Discussion), these scientists will obtain exactly
298 the same results that the original scientist obtained. This process provides the
299 added benefits that 1) the scientist must only document the installation and
300 configuration steps for a single operating system, 2) other scientists need only
301 install the virtualization software and not individual software components, and 3)
302 analyses can be reexecuted indefinitely, so long as the virtualization software
303 remains compatible with current computer systems[88]. Also useful, a team of
304 scientists can employ virtual machines to ensure that each team member has the
305 same computational environment, even though the team members may have
306 different configurations on their host operating systems.

307 One criticism of using virtual machines to support computational reproducibility is
308 that virtual-machine files are large (typically multiple gigabytes); this imposes a
309 barrier for researchers to share such files with the research community. One option
310 is to use cloud-computing services (see Box 4). Scientists can execute an analysis in
311 the cloud, take a "snapshot" of their virtual machine, and share it with others in that
312 environment[86,89]. Cloud-based services typically provide repositories where virtual-
313 machine files can be stored and shared easily among users. Despite these
314 advantages, some researchers may prefer that their data reside on local computers,

315 rather than in the cloud—at least while the research is being performed. In addition,
316 cloud-based services may use proprietary software, so virtual machines may only be
317 executable within each provider's infrastructure. Furthermore, to use a cloud-
318 service provider, scientists may need to activate a fee-based account.

319 Another criticism of using virtual machines to support computational
320 reproducibility is that the software and scripts used in the analysis will be less easily
321 accessible to other scientists—details of the analysis are effectively concealed
322 behind a "black box"[90]. Although other researchers may be able to reexecute the
323 analysis within the virtual machine, it may be more difficult for them to understand
324 and extend the analysis[90]. This problem can be ameliorated when all narratives,
325 scripts, and code are stored in public repositories—separate from the virtual
326 machine—and then imported when the analysis is executed[91]. Another solution is to
327 use a prepackaged virtual machine, such as *Cloud BioLinux*, that contains a variety of
328 software tools commonly used within a given research community[92].

329 Scientists can automate the process of building and configuring virtual machines
330 using tools such as *Vagrant* or *Vortex* (see Box 3). For either tool, users can write
331 text-based configuration files that provide instructions for building virtual machines
332 and allocating computational resources to them. In addition, these configuration
333 files can be used to specify analysis steps[91]. Because these files are text based and
334 relatively small (usually a few kilobytes), scientists can share them easily and track
335 different versions of the files via source-control repositories. This approach also
336 mitigates problems that might arise during the analysis stage. For example, even

when a computer's *host* operating system must be reinstalled due to a computer hardware failure, the virtual machine can be recreated with relative ease.

Box 3: Virtual-machine software.

Virtualization hypervisors:

- VirtualBox (open source) - https://www.virtualbox.org
- Xen (open source) - http://www.xenproject.org
- VMWare (partially open source) - http://www.vmware.com

Virtual-machine management tools:

- Vagrant (open source) - https://www.vagrantup.com
- Vortex (open source) - https://github.com/websecurify/node-vortex

Box 4: Commercial cloud-service providers.

- Amazon Web Services - http://aws.amazon.com
- Rackspace Cloud - http://www.rackspace.com/cloud
- Google Cloud Platform - https://cloud.google.com/compute
- Windows Azure - https://azure.microsoft.com

Software containers ease the process of installing and configuring dependencies

Software containers are a lighter-weight alternative to virtual machines. Like virtual machines, containers encapsulate operating-system components and software into a single package that can be shared with others. Thus, as with virtual machines, analyses executed within a software container should produce identical outputs,

355 irrespective of the underlying operating system or whatever software may be
356 installed outside the container (see Discussion for caveats). As is true for virtual
357 machines, multiple containers can be executed simultaneously on a single computer,
358 and each container may contain different software versions and configurations.
359 However, whereas virtual machines include an entire operating system, software
360 containers interface directly with the computer's main operating system and extend
361 it as needed (Figure 3). This design provides less flexibility than virtual machines
362 because containers are specific to a given type of operating system; however,
363 containers require considerably less computational overhead than virtual machines
364 and can be initialized much more quickly[93].

365 The open-source *Docker* utility (https://www.docker.com)—which has gained
366 popularity among informaticians since its release in 2013—provides the ability to
367 build, execute, and share software containers for Linux-based operating systems.
368 Users specify a *Docker* container's contents using text-based commands. These
369 instructions can be placed in a "Dockerfile," which other scientists can use to rebuild
370 the container. As with virtual-machine configuration files, Dockerfiles are text
371 based, so they can be shared easily and can be tracked and versioned in source-
372 control repositories. Once a *Docker* container has been built, its contents can be
373 exported to a binary file; these files are generally much smaller than virtual-machine
374 files, so they can be shared more easily—for example, via *DockerHub*
375 (https://hub.docker.com).

A key feature of *Docker* containers is that their contents can be stacked in distinct layers (or "images"). Each image includes software component(s) that address a particular need (see Figure 4 for an example). Within a given research lab, scientists might create general-purpose images that support functionality for multiple projects, and they might create specialized images that address the needs of specific projects. *Docker's* modular design provides the advantage that when images within a container are updated, *Docker* only needs to track the specific components that have changed; users who wish to update to a newer version must download a relatively small update. In contrast, even a minor change to a virtual machine would require users to rexport and reshare the entire virtual machine.

Scientists have begun to share *Docker* images with others who are working in the same subdiscipline. For example, *nucleotid.es* is a catalog of genome-assembly tools that have been encapsulated in *Docker* images (http://nucleotid.es). Genome-assembly tools differ considerably in the dependencies that they require and in the parameters that they support. This project provides a means to standardize these assemblers, to circumvent the need to install dependencies for each tool, and to perform benchmarks across the tools. Such projects may help to reduce the reproducibility burden on individual scientists.

The use of *Docker* containers for reproducible research comes with caveats. Individual containers are stored and executed in isolation from other containers on the same computer; however, because all containers on a given machine share the same operating system, this isolation is not as complete as it is with virtual

machines. This means, for example, that a given container is not guaranteed to have access to a specific amount of computer memory or processing power—multiple containers may have to compete for these resources[93]. In addition, containers may be more vulnerable to security breaches[93]. Another caveat is that *Docker* containers can only be executed on Linux-based operating systems. For other operating systems, *Docker* containers must be executed within a virtual machine (for example, see http://boot2docker.io). Although this configuration offsets some benefits of using containers, combining virtual machines with containers may provide a happy medium for many scientists, allowing them to use a non-Linux *host* operating system, while receiving the benefits of containers within the *guest* operating system.

Efforts are ongoing to develop and refine software-container technologies. Box 5 lists various tools that are currently available. In coming years, these technologies promise to play an influential role within the scientific community.

Box 5: Open-source containerization software.

- Docker - https://www.docker.com
- Linux Containers - https://linuxcontainers.org
- lmctfy - https://github.com/google/lmctfy
- OpenVZ - http://openvz.org
- Warden - http://docs.cloudfoundry.org/concepts/architecture/warden.html

412 **Discussion**

413 Scientific advancement requires trust. This review provides a comprehensive,
414 though inexhaustive, list of techniques that can help to engender such trust.
415 Principally, scientists must perform research in such ways that they can trust their
416 own findings[3,94]. Science philosopher Karl Popper contended that "[w]e do not take
417 even our own observations quite seriously, or accept them as scientific
418 observations, until we have repeated and tested them"[2]. Indeed, in many cases, the
419 individuals who benefit most from computational reproducibility are those who
420 performed the original analysis. But reproducibility practices can also help
421 scientists garner each other's trust[94,95]. When other scientists can reproduce an
422 analysis and determine exactly how its conclusions were drawn, they may be more
423 apt to cite the work and build upon it. In contrast, when others fail to reproduce
424 research findings, it can lead to embarrassment, accusations, and retractions.

425 We have described six tools and techniques for computational reproducibility. None
426 of these approaches is sufficient for every scenario in isolation. Rather scientists will
427 often find value from combining approaches. For example, a researcher who uses a
428 literate-programming notebook (which by its nature combines narratives with
429 code) might incorporate the notebook into a software container so that others can
430 execute it without needing to install specific software dependencies. The container
431 might also include a workflow-management system to ease the process of
432 integrating multiple tools and incorporating best practices for the analysis (see
433 Figure 4). This container could be packaged within a virtual machine to ensure that

434 it can be executed on many operating systems. In determining a reproducibility
435 strategy, scientists must evaluate the tradeoff between robustness and practicality.

436 The call for computational reproducibility relies on the premise that reproducible
437 science will bolster the efficiency of the overall scientific enterprise[96]. Although
438 reproducibility practices may require additional time and effort, these practices
439 provides ancillary benefits that help offset those expenditures[94]. Primarily, the
440 scientists who perform a study may experience increased efficiency. For example,
441 before and after a manuscript is submitted for publication, it faces scrutiny from co-
442 authors and peer reviewers who may suggest alterations to the analysis. Having a
443 complete record of all analysis steps and being able to retrace those steps precisely,
444 makes it faster and easier to implement the requested alterations[94,97].
445 Reproducibility practices can also improve the efficiency of team science because
446 colleagues can more easily communicate their research protocols and inspect each
447 other's work; one type of relationship where this is important is that between
448 academic advisors and mentees[97]. Finally, when research protocols are shared
449 transparently with the broader community, scientific advancement increases
450 because scientists can learn more easily from each other's work and duplicate each
451 other's efforts less frequently[97].

452 Reproducibility practices do not necessarily ensure that others can obtain results
453 that are perfectly identical to what the original scientists obtained. Indeed, this
454 objective may be infeasible for various types of computational analysis, including
455 those that use randomization procedures, floating-point operations, or specialized

computer hardware[87]. In such cases, the goal may shift to ensuring that others can obtain results that are semantically consistent with the original findings[5,6]. In addition, in studies where vast computational resources are needed to perform an analysis or where data sets are distributed geographically[98–100], full reproducibility may be infeasible; in these cases, researchers can provide relatively simple examples that demonstrate the methodology. When legal restrictions prevent researchers from sharing software or data publicly, or when software is available only via a Web interface, researchers should document the analysis steps as well as possible and describe why such components cannot be shared[22].

Computational reproducibility does not guarantee against analytical biases or ensure that software produces scientifically valid results[101]. As with any research, a poor study design, confounding effects, or improper use of analytical software may plague even the most reproducible analyses[101,102]. On one hand, increased transparency puts scientists at a greater risk that such problems will be exposed. On the other hand, scientists who are fully transparent about their scientific approach may be more likely to avoid such pitfalls, knowing that they will be more vulnerable to such criticisms. Either way, the scientific community benefits.

Lastly, we emphasize that some reproducibility is better than none. As Voltaire said, the perfect should not be the enemy of the good[103]. However, the practices described in this review are accessible to all scientists and can be implemented with a modest extra effort. As scientists act in good faith to perform these practices, where feasible, the pace of scientific progress will surely increase.

Figure Legends

Figure 1: Basic computer architecture. Computer hardware consists of hardware devices, including central processing units, hard drives, random access memory, keyboard, mouse, etc. Operating systems enable software to interface with hardware; popular operating-system families are *Windows*, *Mac OS*, and *Linux*. Users interact with computers via software interfaces. In scientific computing, software enables users to execute algorithms, analyze data, generate graphics, etc. To execute properly, most software tools depend on specific versions of software dependencies, which must be installed on the same operating system.

Figure 2: Architecture of virtual machines. Virtual machines encapsulate analytical software and dependencies within a "guest" operating system, which may be different than the main ("host") operating system. A virtual machine executes in the context of virtualization software, which executes alongside whatever other software is installed on the computer.

Figure 3: Architecture of software containers. Software containers encapsulate analytical software and dependencies. In contrast to virtual machines, containers execute within the context of the computer's main operating system.

499 **Figure 4: Example of a *Docker* container that could be used for genomics**
500 **research.** This container would enable researchers to preprocess various types of
501 molecular data, using tools from *Bioconductor* and *Galaxy*, and to analyze the
502 resulting data within an *IPython* notebook. Each box within the container represents
503 a distinct *Docker* image. These images are layered such that some images depend on
504 others (for example, the *Bioconductor* image depends on *R*). At its base, the
505 container includes operating-system libraries, which may not be present (or may be
506 configured differently) on the computer's main operating system.

507

508 **References**

509 1.Fisher, R. A. *The Design of Experiments*. (Hafner Press, 1935).

510 2.Popper, K. R. *The Logic of Scientific Discovery*. 23 (Routledge, 2002).

511 3.Peng, R. D. Reproducible research in computational science. *Science (New York,*
512 *N.Y.)* **334,** 1226–7 (2011).

513 4.Russell, J. F. If a job is worth doing, it is worth doing twice. *Nature* **496,** 7 (2013).

514 5.Feynman, R. P., Leighton, R. B. & Sands, M. *Six Easy Pieces: Essentials of Physics*
515 *Explained by Its Most Brilliant Teacher*. 34–35 (Perseus Books, 1994).

516 6.Murray-Rust, P. & Murray-Rust, D. in *Implementing reproducible research* (eds.
517 Stodden, V. C., Leisch, F. & Peng, R. D.) 113 (CRC Press, 2014).

518 7.Software with impact. *Nature Methods* **11,** 211 (2014).

519 8.Chue Hong, N. We are the 92%. in *Second workshop on sustainable software for*
520 *science: Practice and experiences* (2014).
521 doi:http://dx.doi.org/10.6084/m9.figshare.1243288

522 9.Sacks, J., Welch, W. J., Mitchell, T. J. & Wynn, H. P. Design and Analysis of Computer
523 Experiments. *Statistical Science* **4,** 409–423 (1989).

524 10.Garijo, D. *et al.* Quantifying reproducibility in computational biology: The case of
525 the tuberculosis drugome. *PLoS ONE* **8,** (2013).

526 11.Error prone. *Nature* **487,** 406 (2012).

527 12.Vandewalle, P., Barrenetxea, G., Jovanovic, I., Ridolfi, A. & Vetterli, M. Experiences
528 with Reproducible Research in Various Facets of Signal Processing Research. in
529 *2007 iEEE international conference on acoustics, speech and signal processing -*
530 *iCASSP '07* **4,** IV–1253–IV–1256 (IEEE, 2007).

531 13.Cassey, P., Cassey, P., Blackburn, T. & Blackburn, T. Reproducibility and
532 Repeatability in Ecology. *BioScience* **56,** 958–9 (2006).

533 14.Murphy, J. M. *et al.* Quantification of modelling uncertainties in a large ensemble
534 of climate change simulations. *Nature* **430,** 768–772 (2004).

535 15.McCarthy, D. J. *et al.* Choice of transcripts and software has a large effect on
536 variant annotation. *Genome Medicine* **6,** 26 (2014).

537 16.Neuman, J. A., Isakov, O. & Shomron, N. Analysis of insertion-deletion from deep-
538 sequencing data: Software evaluation for optimal detection. *Briefings in*
539 *Bioinformatics* **14,** 46–55 (2013).

540 17.Fonseca, N. A., Marioni, J. A. & Brazma, A. *RNA-seq gene profiling - a systematic*
541 *empirical comparison*. *BioRxiv* 005207 (2014). doi:10.1101/005207

542 18.Bradnam, K. R. *et al.* Assemblathon 2: evaluating de novo methods of genome
543 assembly in three vertebrate species. *GigaScience* **2,** 10 (2013).

544 19.Bilal, E. *et al.* Improving Breast Cancer Survival Analysis through Competition-
545 Based Multidimensional Modeling. *PLoS Computational Biology* **9,** e1003047 (2013).

546 20.Moskvin, O., McIlwain, S. & Ong, I. *Making sense of RNA-Seq data: from low-level*
547 *processing to functional analysis*. *BioRxiv* 010488 (Cold Spring Harbor Labs Journals,
548 2014). doi:10.1101/010488

549 21.Reducing our irreproducibility. *Nature* **496,** 398–398 (2013).

550 22.*Evolution of Translational Omics: Lessons Learned and the Path Forward*. (The
551 National Academies Press, 2012).

552 23.Collins, F. S. & Tabak, L. a. Policy: NIH plans to enhance reproducibility. *Nature*
553 **505,** 612–3 (2014).

554 24.Chambers, J. M. S as a Programming Environment for Data Analysis and Graphics.
555 in *Problem solving environments for scientific computing, proc. 17th symp. on the*
556 *interface of stat. and comp.* 211–214 (1985).

557 25.LeVeque, R. J., Mitchell, I. M. & Stodden, V. Reproducible research for scientific
558 computing: Tools and strategies for changing the culture. *Computing in Science and*
559 *Engineering* **14,** 13 (2012).

560 26.Stodden, V., Guo, P. & Ma, Z. Toward Reproducible Computational Research: An
561 Empirical Analysis of Data and Code Policy Adoption by Journals. *PLoS ONE* **8,** 2–9
562 (2013).

563 27.Morin, A. *et al.* Shining Light into Black Boxes. *Science* **336,** 159–160 (2012).

564 28.Rebooting review. *Nature Biotechnology* **33,** 2015 (2015).

565 29.Ioannidis, J. P. a *et al.* Repeatability of published microarray gene expression
566 analyses. *Nature genetics* **41,** 149–55 (2009).

567 30.Nekrutenko, A. & Taylor, J. Next-generation sequencing data interpretation:
568 enhancing reproducibility and accessibility. *Nature reviews. Genetics* **13,** 667–72
569 (2012).

570 31.Baggerly, K. a. & Coombes, K. R. Deriving chemosensitivity from cell lines:
571 Forensic bioinformatics and reproducible research in high-throughput biology.
572 *Annals of Applied Statistics* **3,** 1309–1334 (2009).

573 32.Sainani, K. It's easy to make mistakes in computational models ... and hard to
574 catch them . *Biomedical Computation Review* **Fall,** 12–19 (2011).

575 33.Decullier, E., Huot, L., Samson, G. & Maisonneuve, H. Visibility of retractions: a
576 cross-sectional one-year study. *BMC Research Notes* **6,** 238 (2013).

577 34.Fang, F. C. & Casadevall, A. Retracted science and the retraction index. *Infection*
578 *and Immunity* **79,** 3855–3859 (2011).

579 35.Claerbout, J. F. & Karrenbach, M. Electronic Documents Give Reproducible
580 Research a New Meaning. in *Meeting of the society of exploration geophysics* (1992).

581 36.Stodden, V. & Miguez, S. Best Practices for Computational Science: Software
582 Infrastructure and Environments for Reproducible and Extensible Research. *Journal*
583 *of Open Research Software* **2,** 21 (2014).

584 37.Ravel, J. & Wommack, K. E. All hail reproducibility in microbiome research.
585 *Microbiome* **2,** 8 (2014).

586 38.Stodden, V. 2014: What scientific idea is ready for retirement? (2014). at
587 <http://edge.org/response-detail/25340>

588 39.Birney, E. *et al.* Prepublication data sharing. *Nature* **461,** 168–170 (2009).

589 40.Hothorn, T. & Leisch, F. Case studies in reproducibility. *Briefings in Bioinformatics*
590 **12,** 288–300 (2011).

591 41.Schofield, P. N. *et al.* Post-publication sharing of data and tools. *Nature* **461,** 171–
592 173 (2009).

593 42.Piwowar, H. a., Day, R. S. & Fridsma, D. B. Sharing detailed research data is
594 associated with increased citation rate. *PLoS ONE* **2,** (2007).

595 43.Johnson, V. E. Revised standards for statistical evidence. *Proceedings of the*
596 *National Academy of Sciences of the United States of America* **110,** 19313–7 (2013).

597 44.Halsey, L. G., Curran-everett, D., Vowler, S. L. & Drummond, G. B. The fickle P
598 value generates irreproducible results. *Nature Methods* **12,** 179–185 (2015).

599 45.Foundation, F. S. GNU Make. at <https://www.gnu.org/software/make>

600 46.Make for Windows. at <http://gnuwin32.sourceforge.net/packages/make.htm>

601 47.Code share. *Nature* **514,** 536 (2014).

48.Loeliger, J. & McCullough, M. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. 456 ('O'Reilly Media, Inc.', 2012).

49.Gentleman, R. C. *et al.* Bioconductor: open software development for computational biology and bioinformatics. *Genome biology* **5,** R80 (2004).

50.R Core Team. *R: A Language and Environment for Statistical Computing*. (R Foundation for Statistical Computing, 2014). at <http://www.r-project.org/>

51.Huber, W. *et al.* Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods* **12,** 115–121 (2015).

52.Tóth, G. *et al.* Space Weather Modeling Framework: A new tool for the space science community. *Journal of Geophysical Research* **110,** A12226 (2005).

53.Tan, E., Choi, E., Thoutireddy, P., Gurnis, M. & Aivazis, M. GeoFramework: Coupling multiple models of mantle convection within a computational framework. *Geochemistry, Geophysics, Geosystems* **7,** n/a–n/a (2006).

54.Heisen, B. *et al.* Karabo: An Integrated Software Framework Combining Control, Data Management, and Scientific Computing Tasks. in *14th international conference on accelerator & large experimental physics control systems, iCALEPCS2013* (2013).

55.Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nature Methods* **9,** 671–675 (2012).

621 56.Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis.
622 *Nature methods* **9,** 676–82 (2012).

623 57.Biasini, M. *et al.* OpenStructure: an integrated software framework for
624 computational structural biology. *Acta crystallographica. Section D, Biological*
625 *crystallography* **69,** 701–9 (2013).

626 58.Köster, J. & Rahmann, S. Snakemake–a scalable bioinformatics workflow engine.
627 *Bioinformatics (Oxford, England)* **28,** 2520–2 (2012).

628 59.Sadedin, S. P., Pope, B. & Oshlack, A. Bpipe : A Tool for Running and Managing
629 Bioinformatics Pipelines. *Bioinformatics (Oxford, England)* **28,** 1525–1526 (2012).

630 60.Tange, O. GNU Parallel - The Command-Line Power Tool.*;login: The USENIX*
631 *Magazine* **36,** 42–47 (2011).

632 61.Albrecht, M., Donnelly, P., Bui, P. & Thain, D. Makeflow: A portable abstraction for
633 data intensive computing on clusters, clouds, and grids. in *Proceedings of the 1st aCM*
634 *sIGMOD workshop on scalable workflow execution engines and technologies* (2012).

635 62.Knuth, D. E. Literate Programming. *The Computer Journal* **27,** 97–111 (1984).

636 63.Pérez, F. & Granger, B. E. IPython: a System for Interactive Scientific Computing.
637 *Computing in Science and Engineering* **9,** 21–29 (2007).

638 64.Shen, H. Interactive notebooks: Sharing the code. *Nature* **515,** 151–152 (2014).

639 65.Xie, Y. *Dynamic Documents with R and knitr*. 216 (CRC Press, 2013).

640 66.Gross, A. M. *et al.* Multi-tiered genomic analysis of head and neck cancer ties
641 TP53 mutation to 3p loss. *Nature Genetics* **46,** 1–7 (2014).

642 67.Ding, T. & Schloss, P. D. Dynamics and associations of microbial community types
643 across the human body. *Nature* **509,** 357–60 (2014).

644 68.Ram, Y. & Hadany, L. The probability of improvement in Fisher's geometric
645 model: A probabilistic approach. *Theoretical population biology* **99,** 1–6 (2015).

646 69.Meadow, J. F. *et al.* Bacterial communities on classroom surfaces vary with
647 human contact. *Microbiome* **2,** 7 (2014).

648 70.White, E. Programming for Biologists. at
649 <http://www.programmingforbiologists.org>

650 71.Software Carpentry. at <https://software-carpentry.org>

651 72.Peng, R. D. Coursera course: Computing for Data Analysis. at
652 <https://www.coursera.org/course/compdata>

653 73.Bioconductor Courses & Conferences. at
654 <http://master.bioconductor.org/help/course-materials>

655 74.Gil, Y. *et al.* Examining the challenges of scientific workflows. *Computer* **40,** 24–32
656 (2007).

657 75.Giardine, B. *et al.* Galaxy: a platform for interactive large-scale genome analysis.
658 *Genome research* **15,** 1451–5 (2005).

659 76.Goecks, J., Nekrutenko, A. & Taylor, J. Galaxy: a comprehensive approach for
660 supporting accessible, reproducible, and transparent computational research in the
661 life sciences. *Genome biology* **11,** R86 (2010).

662 77.Afgan, E. *et al.* Harnessing cloud computing with Galaxy Cloud. *Nature*
663 *Biotechnology* **29,** 972–974 (2011).

664 78.Callahan, S. P. *et al.* VisTrails: Visualization Meets Data Management. in
665 *Proceedings of the 2006 aCM sIGMOD international conference on management of*
666 *data* 745–747 (ACM, 2006). doi:10.1145/1142473.1142574

667 79.Davidson, S. B. & Freire, J. Provenance and scientific workflows. in *Proceedings of*
668 *the 2008 aCM sIGMOD international conference on management of data - sIGMOD '08*
669 1345 (2008). doi:10.1145/1376616.1376772

670 80.Altintas, I. *et al.* Kepler: an extensible system for design and execution of
671 scientific workflows. in *Proceedings. 16th international conference on scientific and*
672 *statistical database management, 2004.* 423–424 (IEEE, 2004).
673 doi:10.1109/SSDM.2004.1311241

674 81.Goff, S. A. *et al.* The iPlant Collaborative: Cyberinfrastructure for Plant Biology.
675 *Frontiers in plant science* **2,** 34 (2011).

676 82.Reich, M. *et al.* GenePattern 2.0. *Nature genetics* **38,** 500–1 (2006).

677 83.Reich, M. *et al.* GenomeSpace: An environment for frictionless bioinformatics.
678 *Cancer Research* **72,** 3966–3966 (2012).

84.Wolstencroft, K. *et al.* The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research* **41,** 557–561 (2013).

85.Rex, D. E., Ma, J. Q. & Toga, A. W. The LONI Pipeline Processing Environment. *NeuroImage* **19,** 1033–1048 (2003).

86.Dudley, J. T. & Butte, A. J. In silico research in the era of cloud computing. *Nature biotechnology* **28,** 1181–1185 (2010).

87.Hurley, D. G., Budden, D. M. & Crampin, E. J. Virtual Reference Environments: a simple way to make research reproducible. *Briefings in Bioinformatics* 1–3 (2014). doi:10.1093/bib/bbu043

88.Gent, I. P. The Recomputation Manifesto. *arXiv* (2013). at <http://arxiv.org/abs/1304.3674>

89.Howe, B. Virtual Appliances, Cloud Computing, and Reproducible Research. *Computing in Science & Engineering* **14,** 36–41 (2012).

90.Brown, C. T. Virtual machines considered harmful for reproducibility. *Living in an ivory basement: Stochastic thoughts on science, testing, and programming.* (2012). at <http://ivory.idyll.org/blog/vms-considered-harmful.html>

91.Piccolo, S. R. Building portable analytical environments to improve sustainability of computational-analysis pipelines in the sciences. *Figshare* (2014). at <http://dx.doi.org/10.6084/m9.figshare.1112571>

699 92.Krampis, K. *et al.* Cloud BioLinux: pre-configured and on-demand bioinformatics
700 computing for the genomics community. *BMC Bioinformatics* **13,** 42 (2012).

701 93.Felter, W., Ferreira, A., Rajamony, R. & Rubio, J. An Updated Performance
702 Comparison of Virtual Machines and Linux Containers. (2014). at
703 <http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD81
704 9C85257D2300681E7B/\$File/rc25482.pdf>

705 94.Sandve, G. K., Nekrutenko, A., Taylor, J. & Hovig, E. Ten Simple Rules for
706 Reproducible Computational Research. *PLoS Computational Biology* **9,** 1–4 (2013).

707 95.Hones, M. J. . Reproducibility as a Methodological Imperative in Experimental
708 Research. in *PSA: Proceedings of the biennial meeting of the philosophy of science*
709 *association* **One,** 585–599 (Philosophy of Science Association, 1990).

710 96.Crick, T., Hall, B. A., Ishtiaq, S. & Takeda, K. 'Share and Enjoy': Publishing Useful
711 and Usable Scientific Models. in *1st international workshop on recomputability* 1–5
712 (2014). at <http://arxiv.org/abs/1409.0367>

713 97.Donoho, D. L. An invitation to reproducible computational research. *Biostatistics*
714 *(Oxford, England)* **11,** 385–8 (2010).

715 98.Shirts, M. & Pande, V. S. COMPUTING: Screen Savers of the World Unite! *Science*
716 *(New York, N.Y.)* **290,** 1903–1904 (2000).

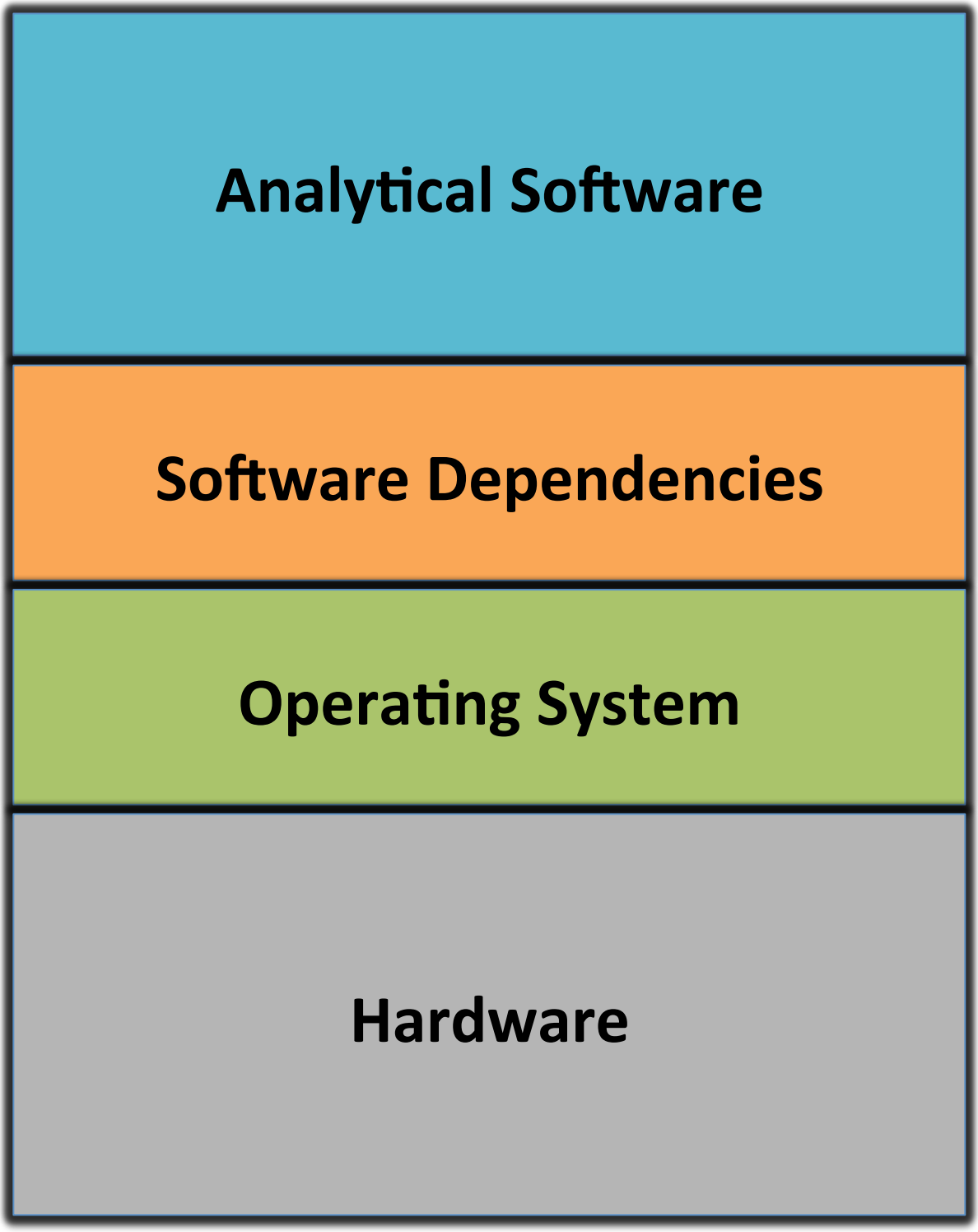
717 99.Bird, I. Computing for the Large Hadron Collider. *Annual Review of Nuclear and*
718 *Particle Science* **61,** 99–118 (2011).

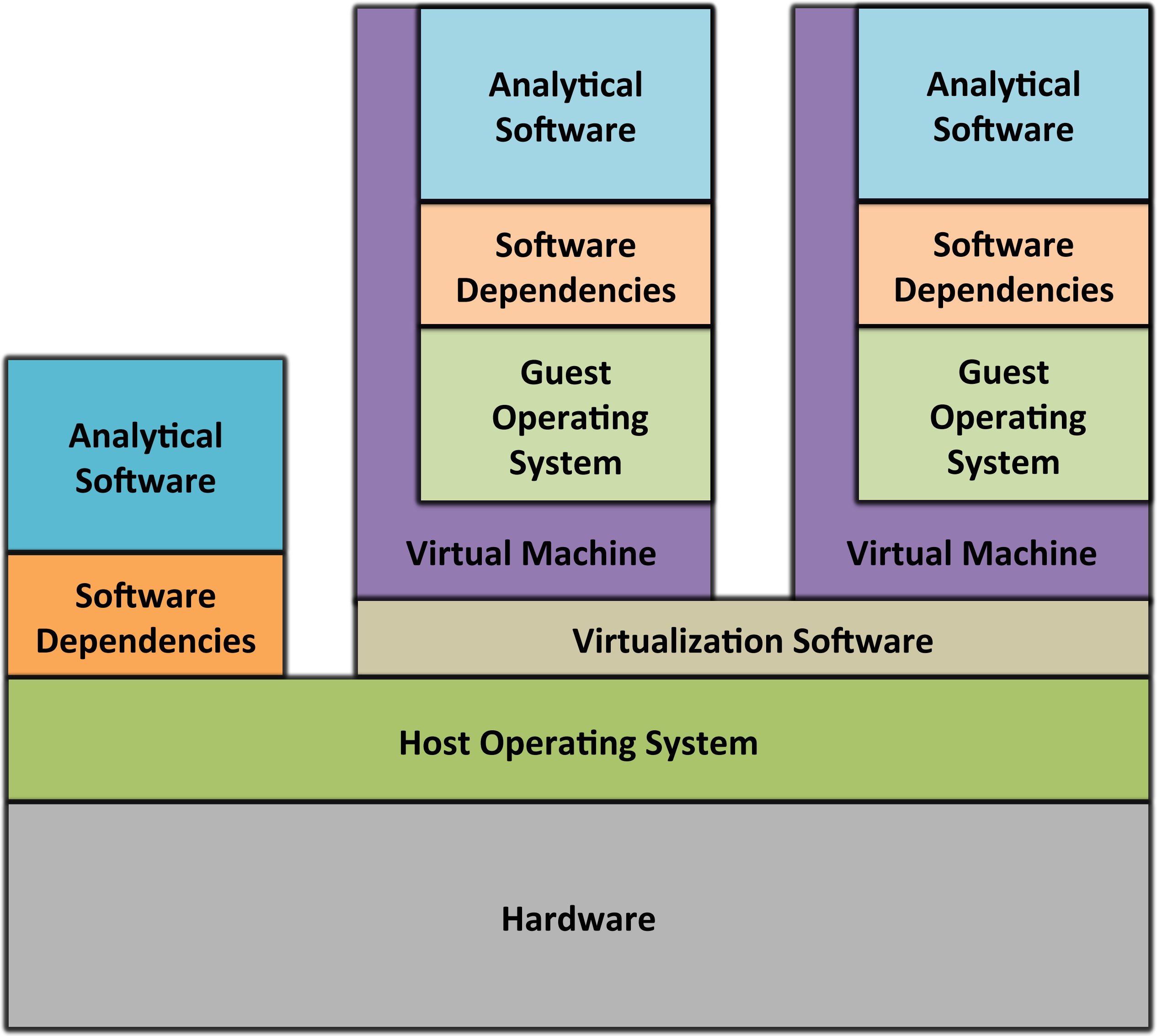
100.Anderson, D. P. BOINC: A System for Public Resource Computing and Storage. in *Proceedings of the fifth iEEE/ACM international workshop on grid computing (gRID'04)* (2004).

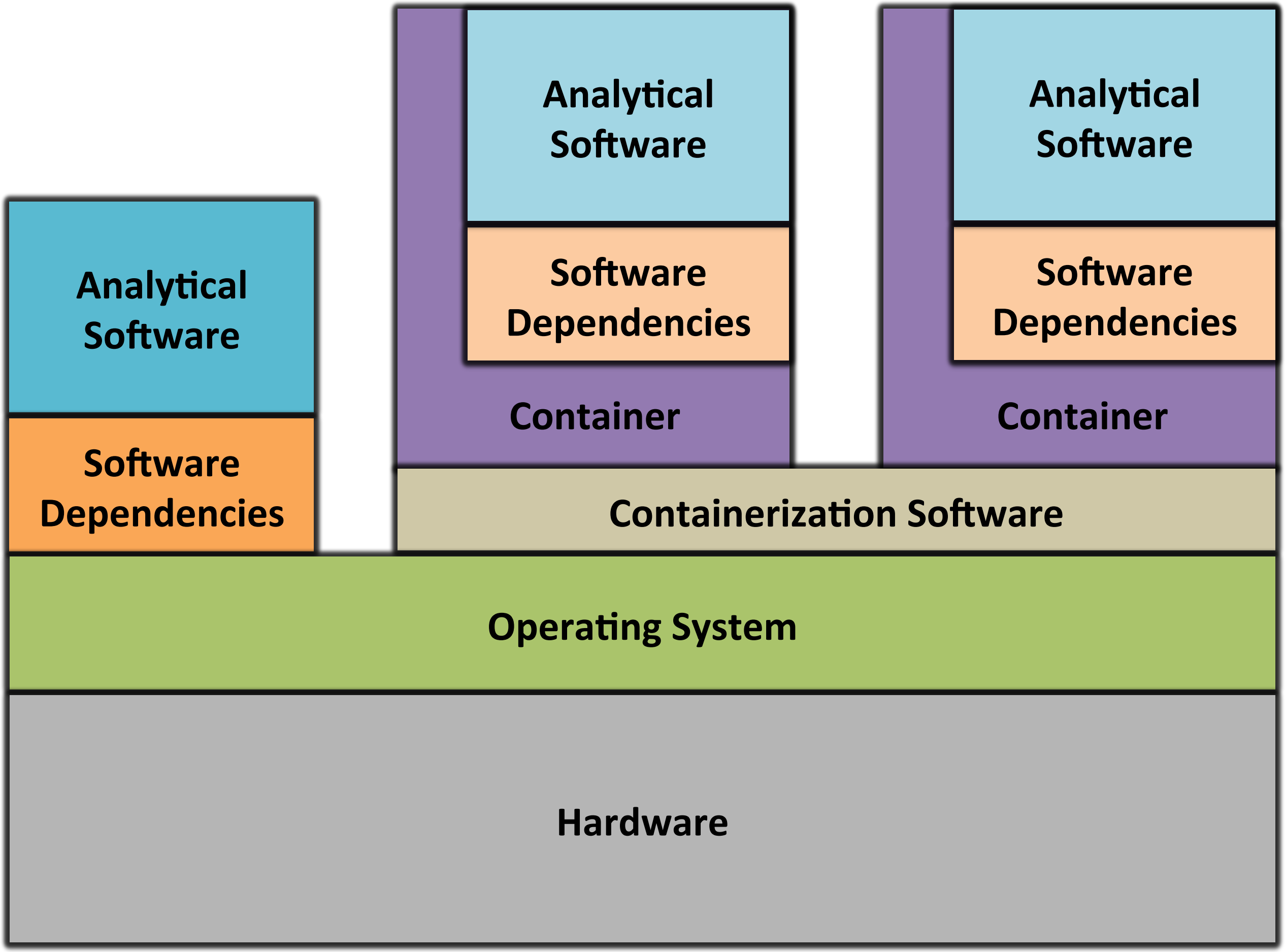
101.Ransohoff, D. F. Bias as a threat to the validity of cancer molecular-marker research. *Nature reviews. Cancer* **5,** 142–9 (2005).

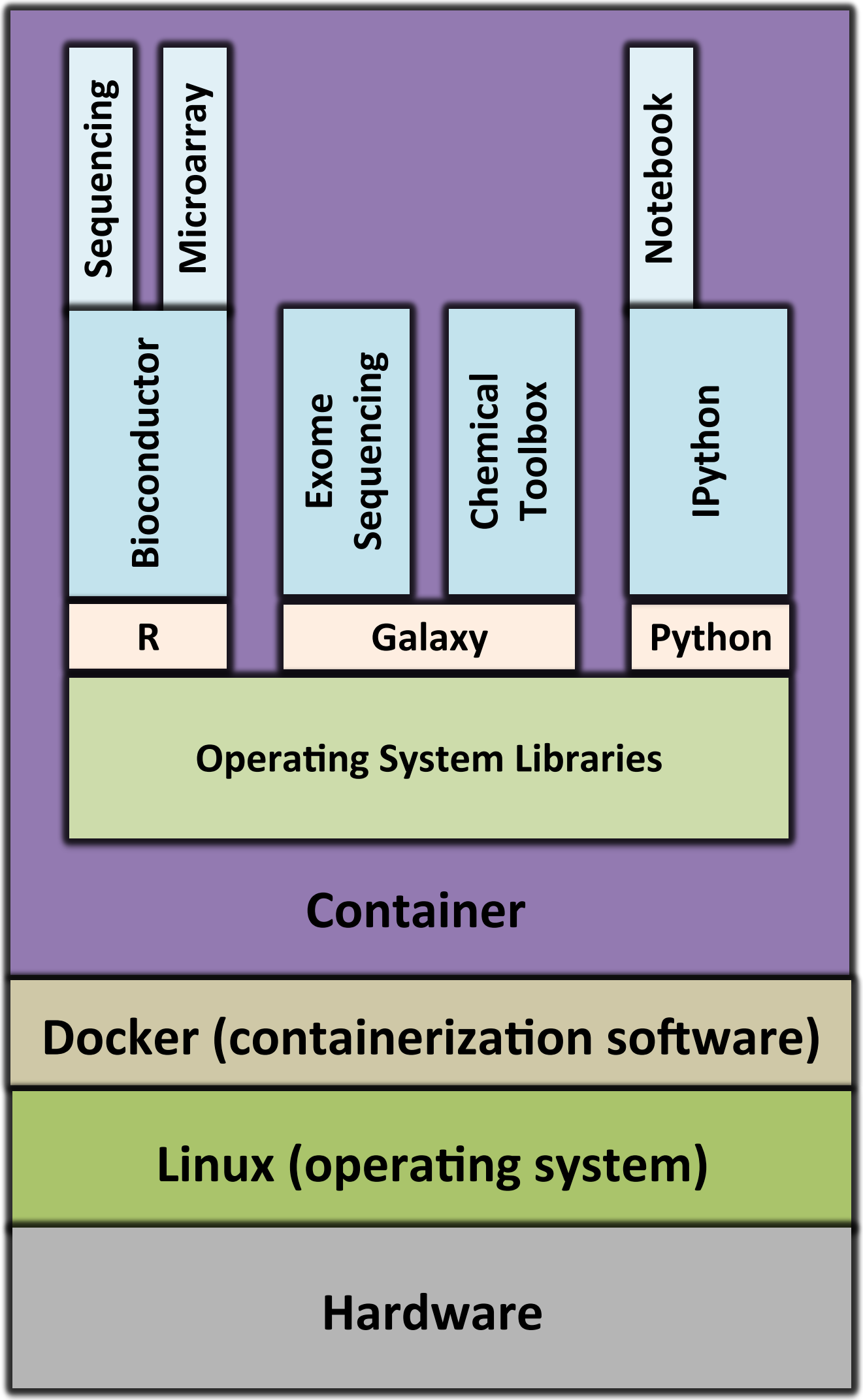
102.Bild, A. H., Chang, J. T., Johnson, W. E. & Piccolo, S. R. A field guide to genomics research. *PLoS biology* **12,** e1001744 (2014).

103.Ratcliffe, S. *Concise Oxford Dictionary of Quotations*. 389 (Oxford University Press, 2011).


Analytical Software
Software Dependencies
Operating System
Hardware


Analytical Software
Software Dependencies
Analytical Software
Software Dependencies
Guest Operating System
Virtual Machine
Analytical Software
Software Dependencies
Guest Operating System
Virtual Machine
Virtualization Software
Host Operating System
Hardware


Analytical Software
Software Dependencies
Analytical Software
Software Dependencies
Container
Analytical Software
Software Dependencies
Container
Containerization Software
Operating System
Hardware


Sequencing
Microarray
Notebook
Bioconductor
Exome Sequencing
Chemical Toolbox
IPython
R
Galaxy
Python
Operating System Libraries
Container
Docker (containerization software)
Linux (operating system)
Hardware