

SOFTWARE PROVENANCE: TRACK THE REALITY NOT THE VIRTUAL MACHINE



David Wilkinson, Luís Oliveira, Daniel Mossé, Bruce Childers
Computer Science Department – University of Pittsburgh
{dwilk, loliveira}@cs.pitt.edu

Motivation

Preservation crisis

- Substantial amount of new data created everyday
- Both scientific and creative data/work

How to best preserve future **interactivity**?

- What can we do to allow data to not only be *read* but understood or manipulated later?
- Particularly if we discover defects very late!

Possible Solutions Abound

Preserve older hardware to run older software

- but how practical is this? How expensive/accessible?

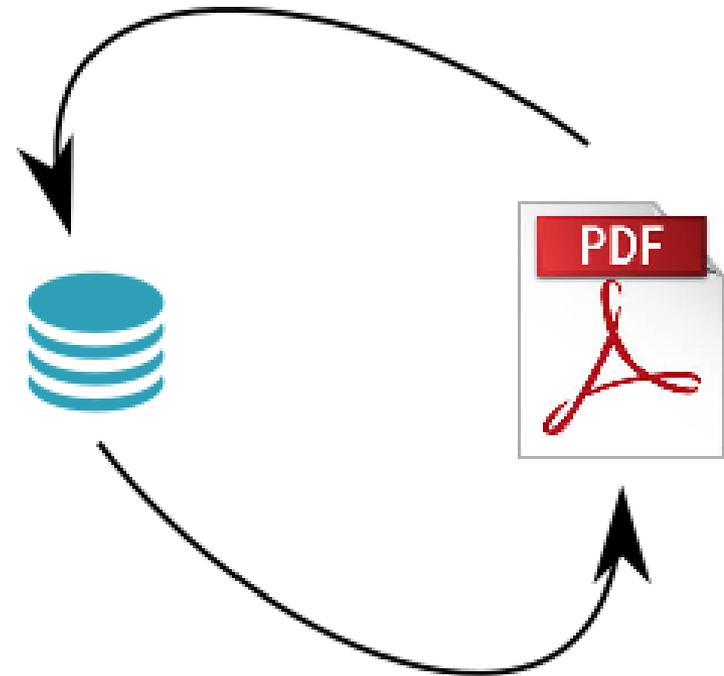
The Byte Cellar | <http://www.bytecellar.com/>



Possible Solutions Abound

Transform files into more stable, more modern formats.

- Old Word -> XML, PDF
- Old Images -> PNG, JPG



Transformation Fatigue

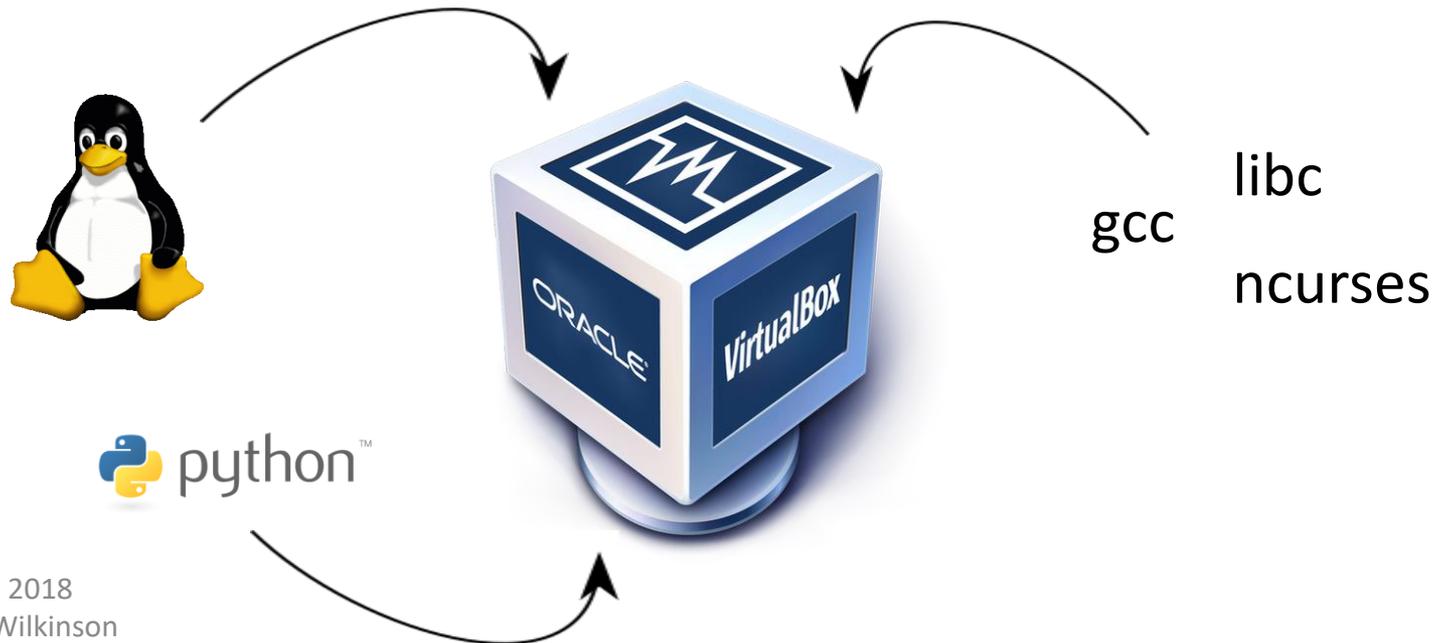
The difficulty is within the presumption that the resulting file is actually both better-preserved and complete.

- PDF is not necessarily a stable standard.
- (Do we commit to a subset of PDF? Who decides this?)
- Nothing prevents PDF, PNG, etc from itself becoming obsolete
- Do we convert again in the future?

The Scientific Method

Datasets, plots, papers are all possible artifacts that may also have these preservation concerns.

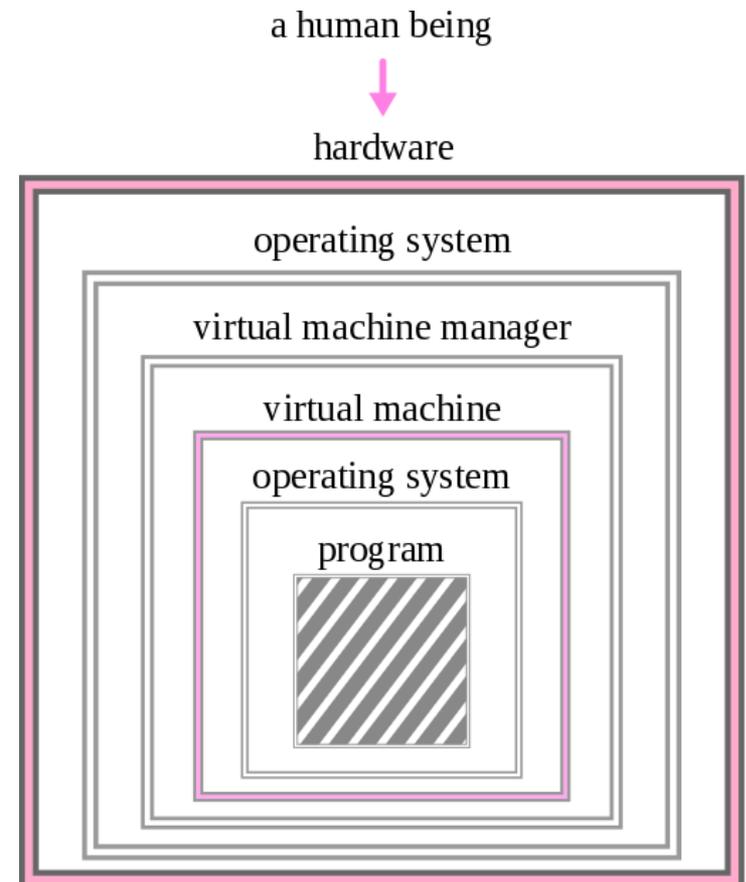
Current acceptable approach: Distribute a VM image.



The Power/Weaknesses of Virtual Machines

Virtual Machines are useful because they are powerful **abstractions**.

Yet, that also means much is hidden or taken as faith.



Generating Virtual Machines

Occam takes the following approach:

Instead of distributing virtual machine images (OVA/OVF), distribute descriptions of virtual machines.

Maintain modular/composable archives of software, much like package managers.

Example: DOS Games

We want to run some old DOS game: Doom

We would describe Doom as an object that needs a “dos” environment.

The system would prepare a virtual machine for the host machine that can execute a dos environment.

Example: DOS Games

```
{  
  "name": "Doom", "type": "game",  
  "install": [ {  
    "type": "resource", "subtype": "application/zip",  
    "source": "ftp://ftp.idsoftware.com/doom/doom19s.zip",  
    "actions": { "unpack": "." }  
  } ],  
  "environment": "dos", "architecture": "x86",  
  "run": { "command": "DOOM.EXE" }  
}
```

Example: DOS Games

To build the VM, we need to map a “dos” environment to a native “linux” environment.

In Occam, another artifact can describe such a relationship. Such as DOSBox, a DOS emulator for Linux.



Example: DOS Games

```
{  
  "name": "DOSBox", "type": "emulator",  
  "provides": [ {  
    "environment": "dos", "architecture": "x86"  
  } ],  
  "dependencies": [ { "name": "x11"}, ... ],  
  "environment": "linux", "architecture": "x86-64",  
  "build": { ... },  
  "run": { "command": "/usr/bin/dosbox" }  
}
```

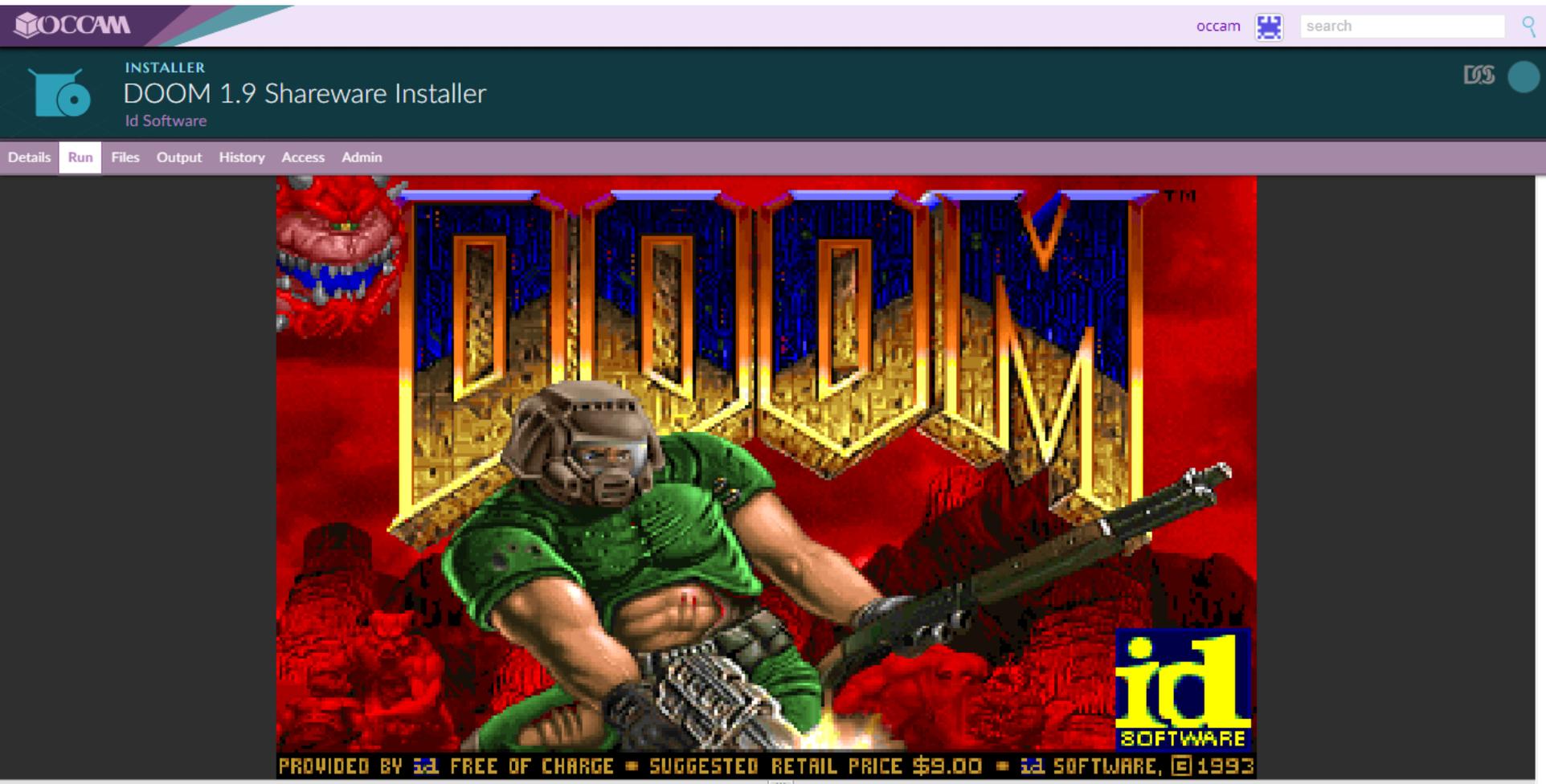
Example: DOS Games

Occam then can create a virtual machine or execute a docker container with the emulator (DOSBox), any libraries it has as dependencies, and the DOS game.

This is the **manifest**, which is the main distributable (kilobytes), along with individual software artifacts (megabytes) instead of a complete VM image (gigabytes).

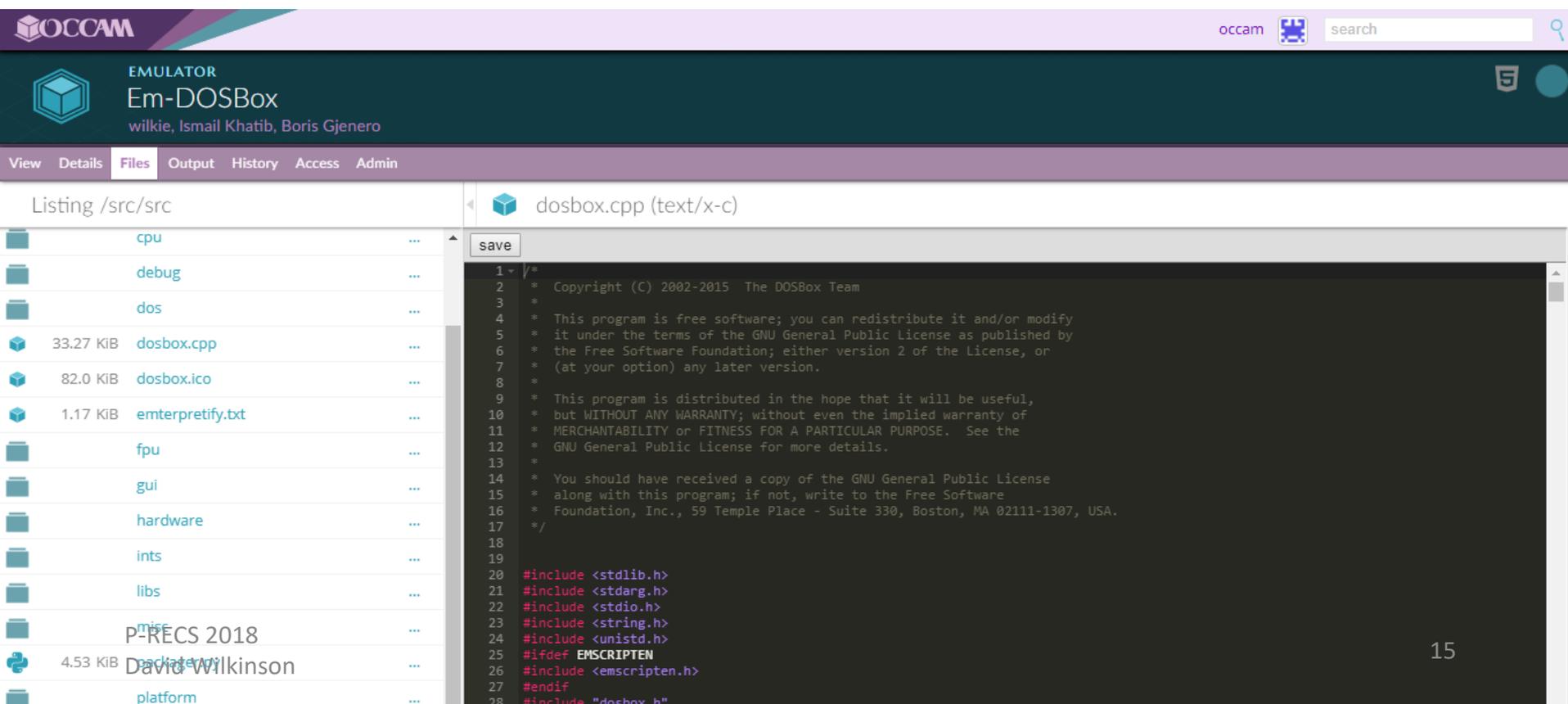
It then executes the virtual machine by running DOSBox, which is then responsible for running the game or application.

Example: DOS Games



Software Provenance for Emulation

Through this, we can emulate obsolete software while retaining easy access to how the emulator was built.

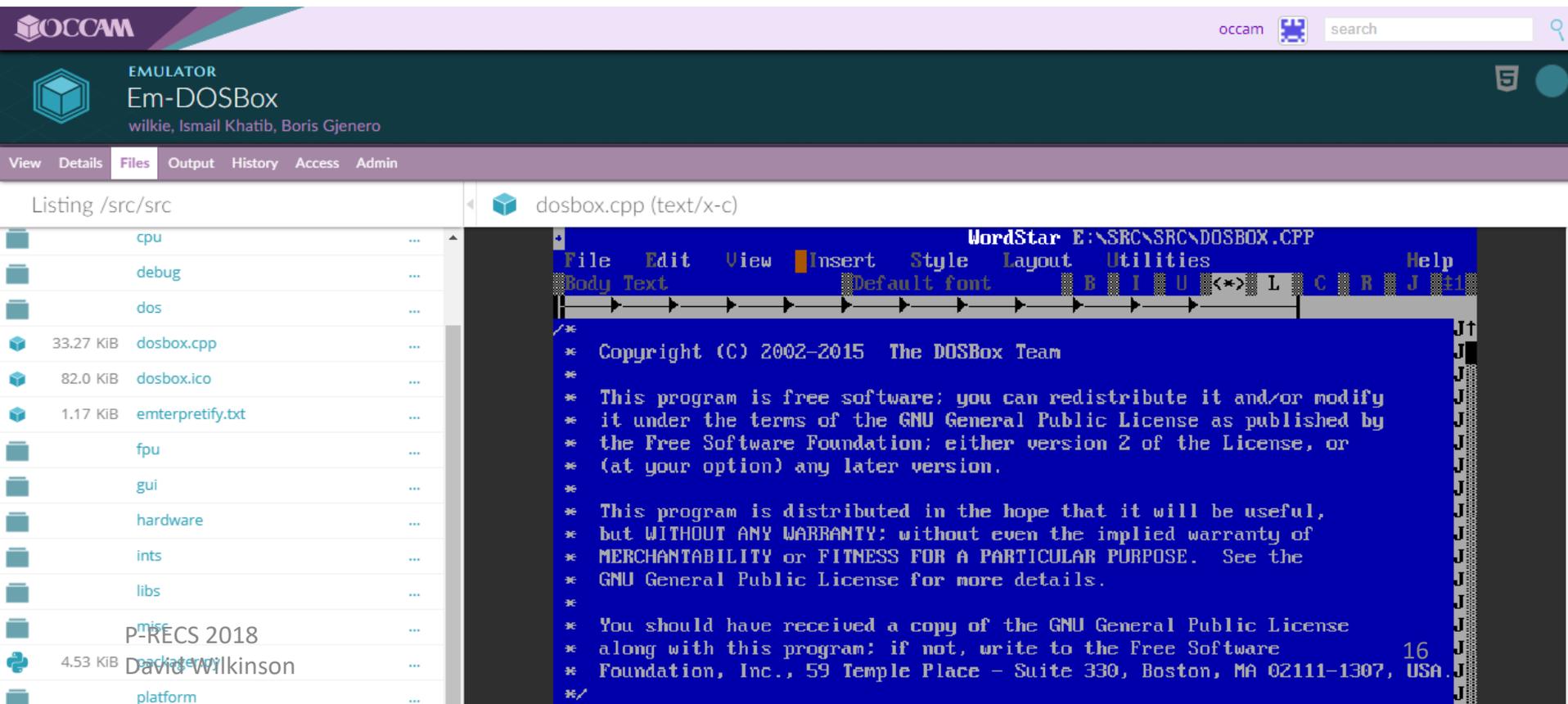


The screenshot shows the Em-DOSBox web interface. The top navigation bar includes the OCCAM logo, the text "EMULATOR Em-DOSBox", and the authors "wilkie, Ismail Khatib, Boris Gjenaro". A search bar is also present. Below the navigation bar, there are tabs for "View", "Details", "Files", "Output", "History", "Access", and "Admin". The "Files" tab is active, showing a file listing for the "/src/src" directory. The listing includes folders like "cpu", "debug", "dos", "fpu", "gui", "hardware", "ints", "libs", "platform", and "misc", as well as files "dosbox.cpp" (33.27 KiB), "dosbox.ico" (82.0 KiB), and "emterpretify.txt" (1.17 KiB). The "dosbox.cpp" file is selected, and its source code is displayed in a dark-themed editor. The code includes a copyright notice for 2002-2015 by the DOSBox Team, followed by the GNU General Public License text. The code also includes standard C++ headers and defines the EMSCRIPTEN environment.

```
1 /*
2  * Copyright (C) 2002-2015 The DOSBox Team
3  *
4  * This program is free software; you can redistribute it and/or modify
5  * it under the terms of the GNU General Public License as published by
6  * the Free Software Foundation; either version 2 of the License, or
7  * (at your option) any later version.
8  *
9  * This program is distributed in the hope that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 * GNU General Public License for more details.
13 *
14 * You should have received a copy of the GNU General Public License
15 * along with this program; if not, write to the Free Software
16 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
17 */
18
19
20 #include <stdlib.h>
21 #include <stdarg.h>
22 #include <stdio.h>
23 #include <string.h>
24 #include <unistd.h>
25 #ifdef EMSCRIPTEN
26 #include <emscripten.h>
27 #endif
28 #include "dosbox.h"
```

Example: DOS Applications

If we can handle a game, we can handle a word processing application!



The screenshot displays the Em-DOSBox emulator interface. The top bar includes the OCCAM logo, the text "EMULATOR Em-DOSBox", and the names "wilkie, Ismail Khatib, Boris Gjenero". A search bar is visible on the right. Below the top bar is a navigation menu with "View", "Details", "Files", "Output", "History", "Access", and "Admin".

The main window is split into two panes. The left pane shows a file listing for the directory "/src/src":

- cpu
- debug
- dos
- 33.27 KIB dosbox.cpp
- 82.0 KIB dosbox.ico
- 1.17 KIB emterpretify.txt
- fpu
- gui
- hardware
- ints
- libs
- misc
- P-RECS 2018
- 4.53 KIB David Wilkinson
- platform

The right pane shows a window titled "WordStar E:\SRC\SRC\DOSBOX.CPP". The window has a menu bar with "File", "Edit", "View", "Insert", "Style", "Layout", "Utilities", and "Help". Below the menu bar is a toolbar with icons for "Body Text", "Default font", "B", "I", "U", "L", "C", "R", "J", and "E1". The main text area contains the following text:

```
/*
 * Copyright (C) 2002-2015 The DOSBox Team
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
 */
```

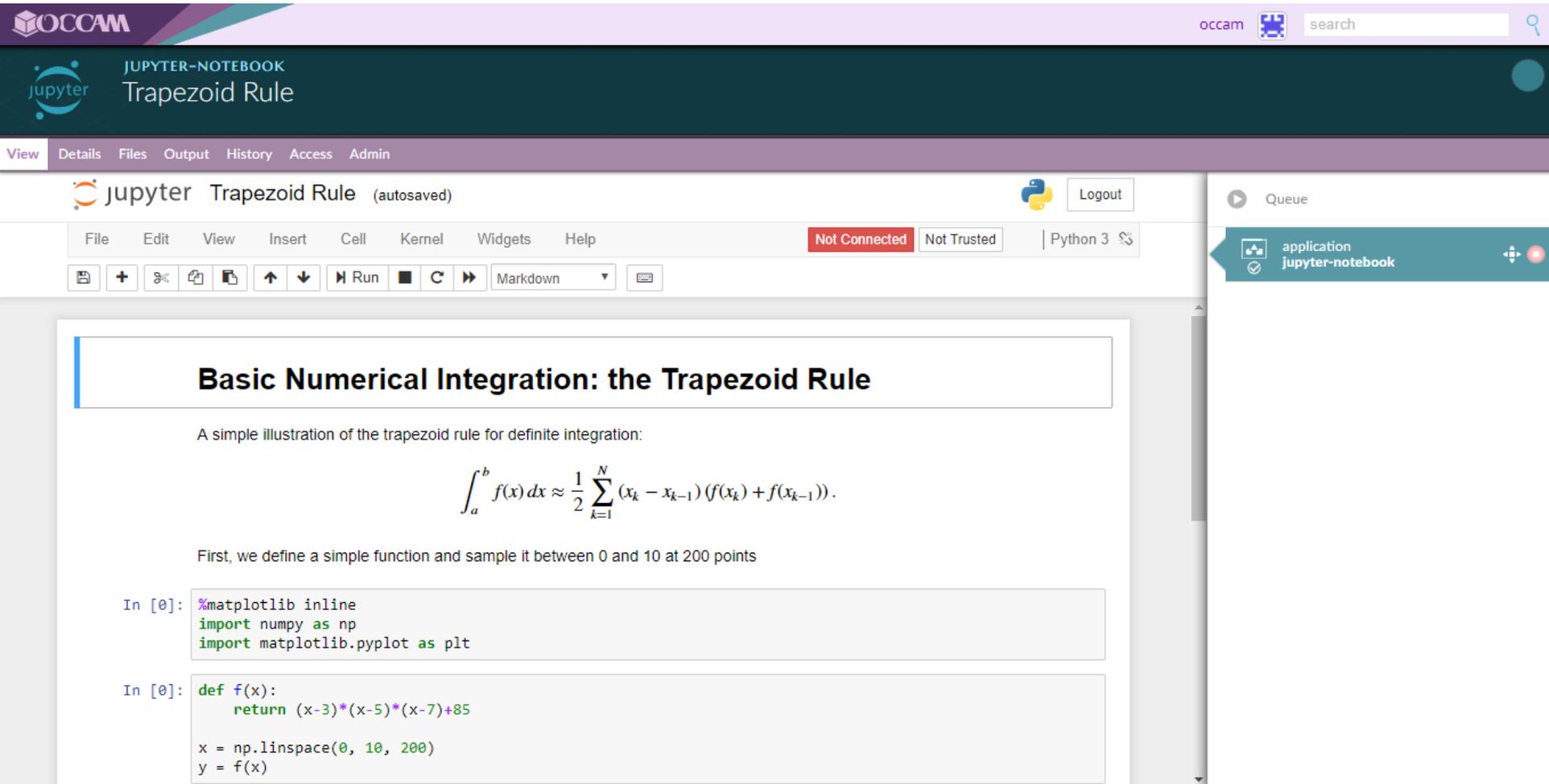
The page number "16" is visible in the bottom right corner of the WordStar window.

Example: Jupyter Notebook

Preserving Jupyter notebooks has been a challenge in the community solved mostly by well-defined common distributions (Anaconda)

Yet, can we provide a more general solution that preserves run-time/build-time provenance?

Example: Jupyter Notebook



The screenshot shows a Jupyter Notebook interface. At the top, there's a purple header with the OCCAM logo and a search bar. Below that, a dark green header contains the Jupyter logo and the notebook title 'Trapezoid Rule'. A navigation bar includes 'View', 'Details', 'Files', 'Output', 'History', 'Access', and 'Admin'. The main interface has a light purple header with 'jupyter Trapezoid Rule (autosaved)' and a 'Logout' button. A menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu bar, there are status indicators: 'Not Connected', 'Not Trusted', and 'Python 3'. A toolbar contains icons for file operations, a 'Run' button, and a 'Markdown' dropdown. The notebook content area has a title 'Basic Numerical Integration: the Trapezoid Rule' and a paragraph: 'A simple illustration of the trapezoid rule for definite integration:'. Below this is a mathematical formula:
$$\int_a^b f(x) dx \approx \frac{1}{2} \sum_{k=1}^N (x_k - x_{k-1}) (f(x_k) + f(x_{k-1})).$$
 This is followed by another paragraph: 'First, we define a simple function and sample it between 0 and 10 at 200 points'. There are two code input cells. The first cell contains:

```
In [0]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

 The second cell contains:

```
In [0]: def f(x):
    return (x-3)*(x-5)*(x-7)+85

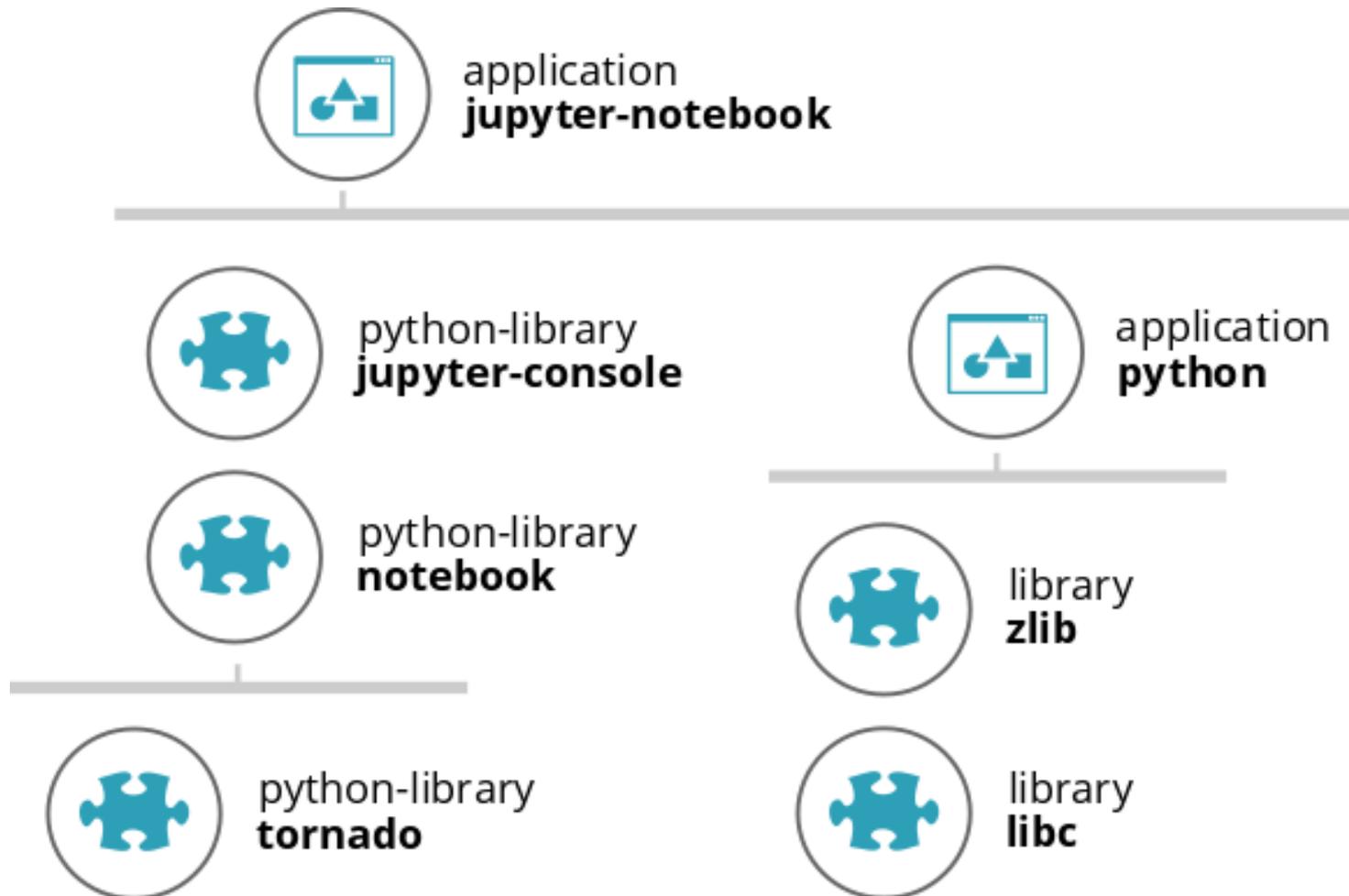
x = np.linspace(0, 10, 200)
y = f(x)
```

 On the right side, there is a 'Queue' panel showing 'application jupyter-notebook' with a play button and a red status indicator.

Example: Jupyter Notebook

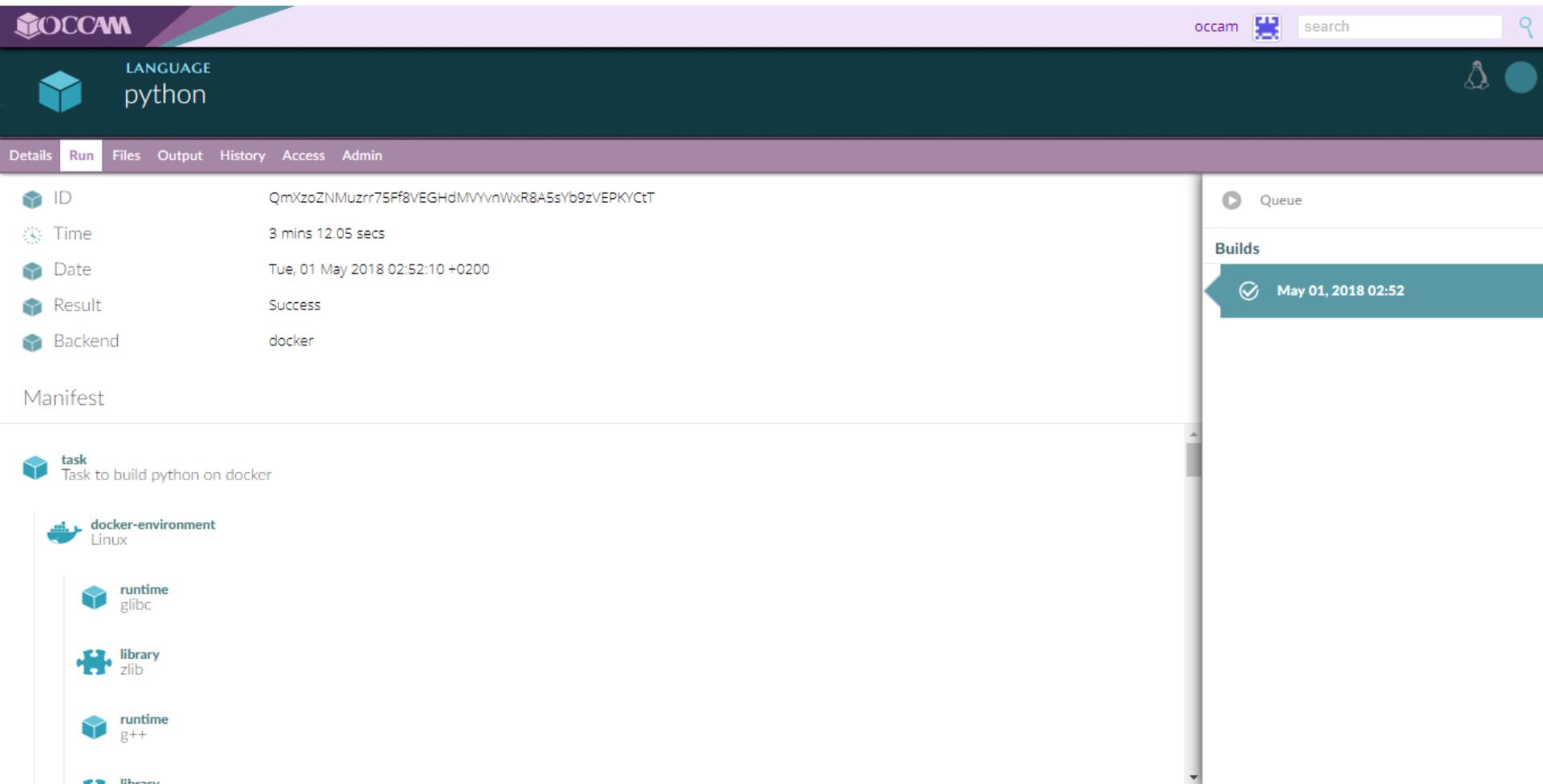
```
{  
  "name": "jupyter-notebook", "type": "application",  
  "views": [ {  
    "type": "file", "subtype": "extension/ipynb"  
  } ],  
  "dependencies": [ { "name": "python"}, ... ],  
  "environment": "linux", "architecture": "x86-64",  
  "build": { ... },  
  "network": { "bind": "8888" },  
  "run": { "command": "run.sh" }  
}
```

Example: Jupyter Notebook



Example: Jupyter Notebook

The Build Provenance of Python 3



The screenshot displays the OCCAM web interface. At the top, there is a navigation bar with the OCCAM logo, a search bar, and a user profile icon. Below this is a dark blue header with the text 'LANGUAGE python'. A secondary navigation bar contains tabs for 'Details', 'Run', 'Files', 'Output', 'History', 'Access', and 'Admin'. The main content area is divided into two sections. The left section, titled 'Manifest', shows a table of build details:

ID	QmXzoZNMuzrr75Ff8VEGHdMVVvnWxR8A5sYb9zVEPKYCT
Time	3 mins 12.05 secs
Date	Tue, 01 May 2018 02:52:10 +0200
Result	Success
Backend	docker

The right section, titled 'Builds', shows a 'Queue' and a single build entry for 'May 01, 2018 02:52' with a checkmark icon. Below the 'Manifest' section, a 'task' is listed: 'Task to build python on docker'. This task is composed of several components:

- docker-environment Linux
 - runtime glibc
 - library zlib
 - runtime g++
 - library

Conclusions

In order to provide **interactivity** in the future, we must preserve software (the mechanism of interactivity)

Virtual Machines are great tools, but they are not interested in detailed provenance.

The greatest preservation of both provenance and interactivity is through the generation of virtual machines from manifests.

Final thoughts

Need feedback from the community:

{dwilk,loliveira}@cs.pitt.edu

Current Occam implementation:

- Preserve as much as possible
 - Prevent silent loss of fidelity
 - Improve the longevity of software
- Preserving source code is vital
 - And the ability to build/run it.
- Dependencies are important
 - They may be the source of errors