Software Provenance: Track the Reality not the Virtual Machine

HTTPS://OCCAM.CS.PITT.EDU

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

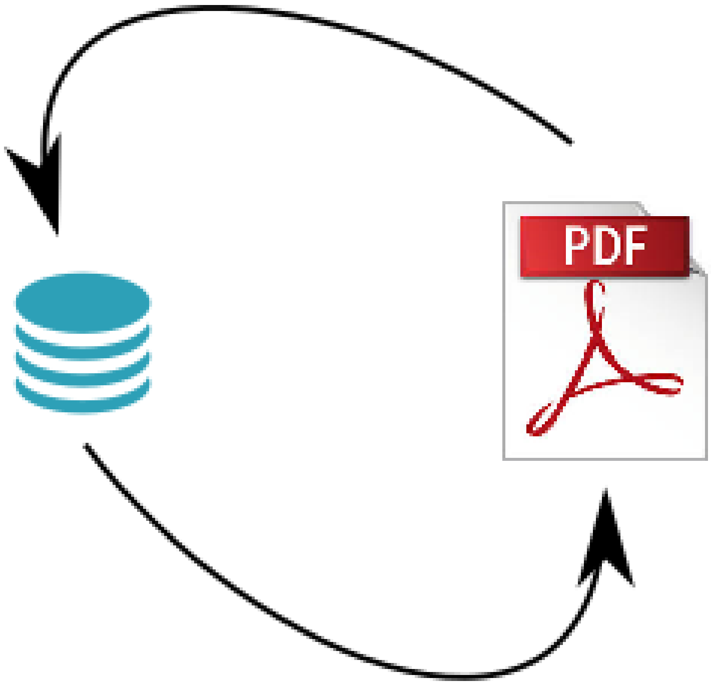
- but how practical is this? How expensive/accessible?

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

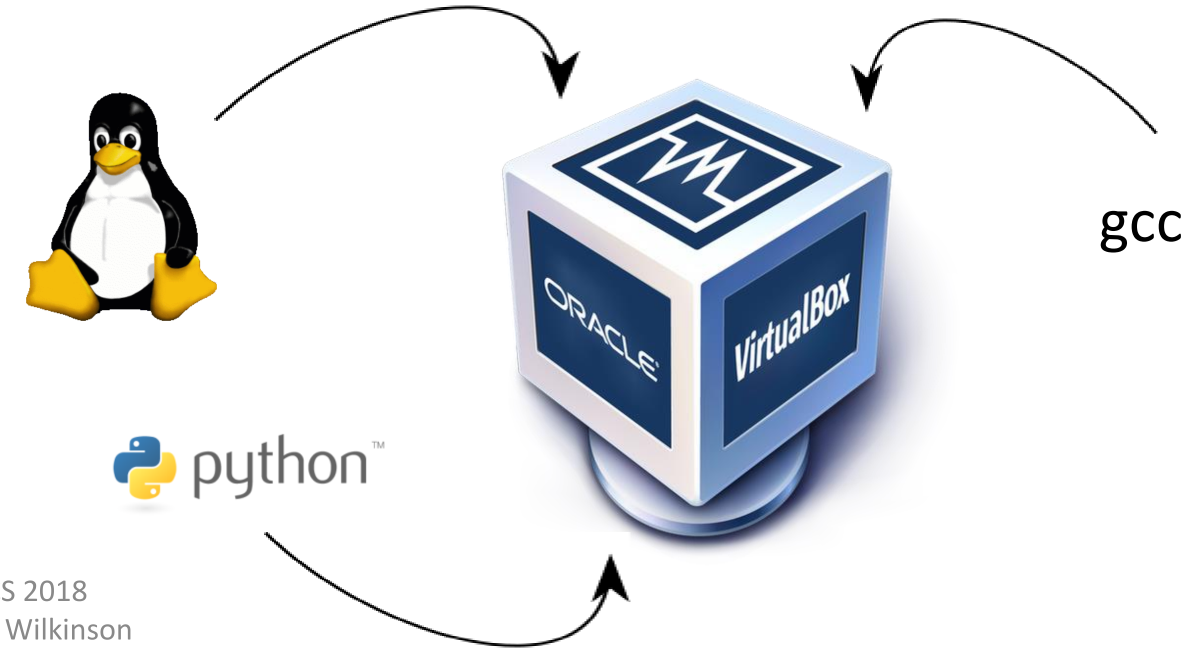
The difficulty is within the presumption that the resulting file is actually both better-preserved and complete.

- 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.



gcc

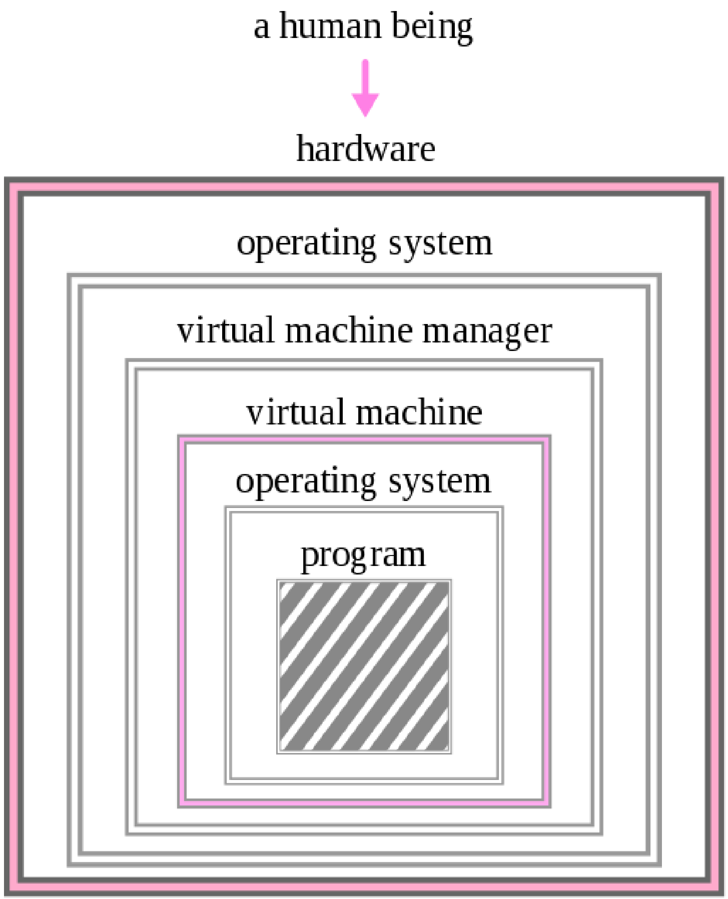
libc

ncurses

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.

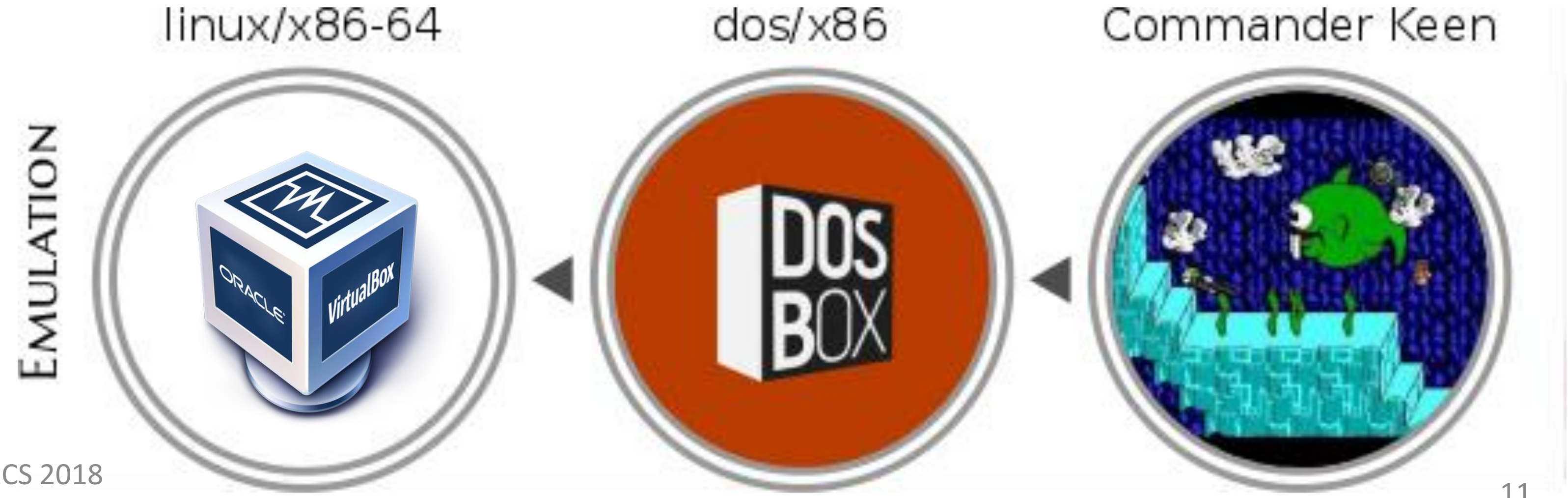
Example: DOS Games

```
{
  “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” }
}
```

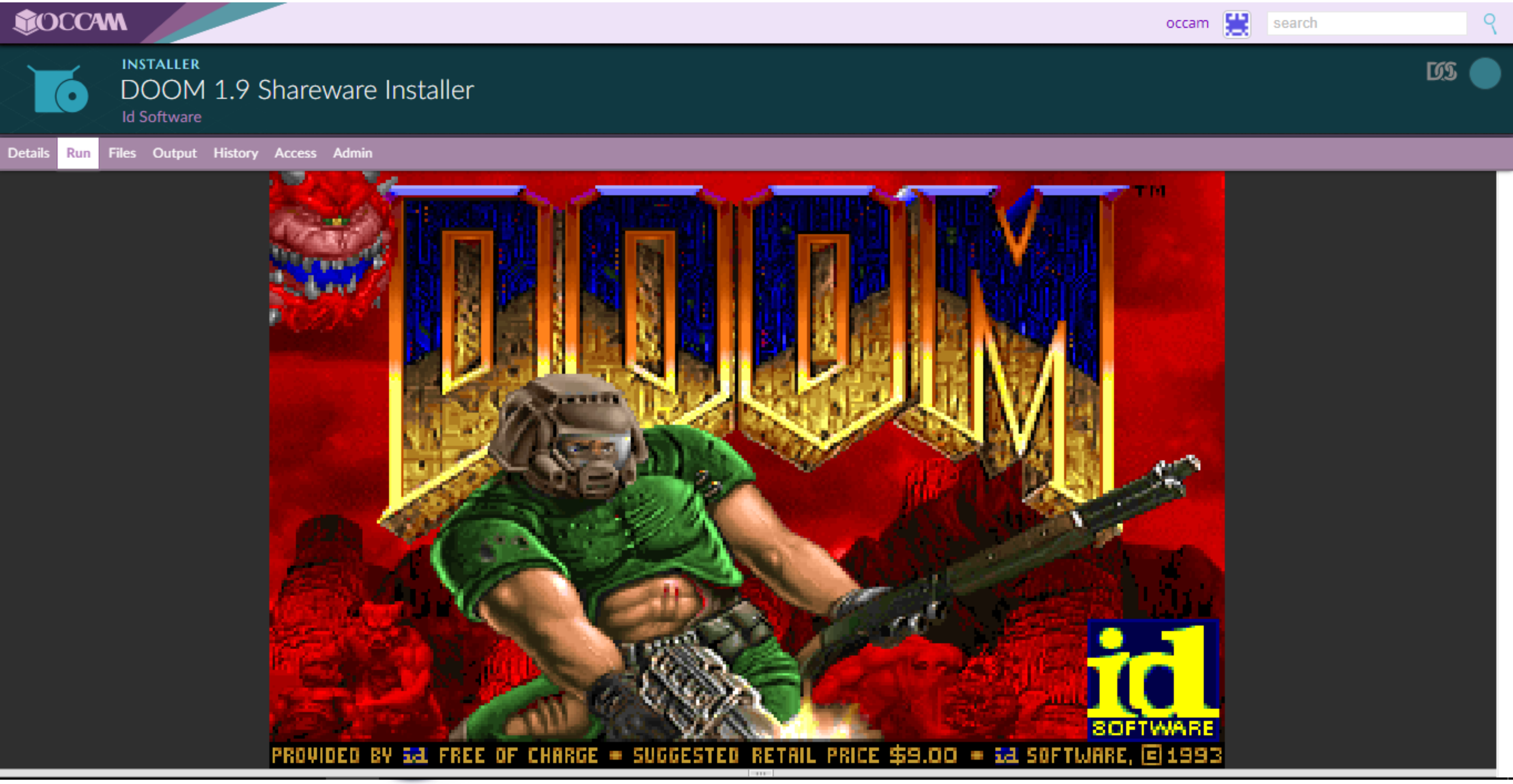
Example: DOS Games

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



occam

search

INSTALLER

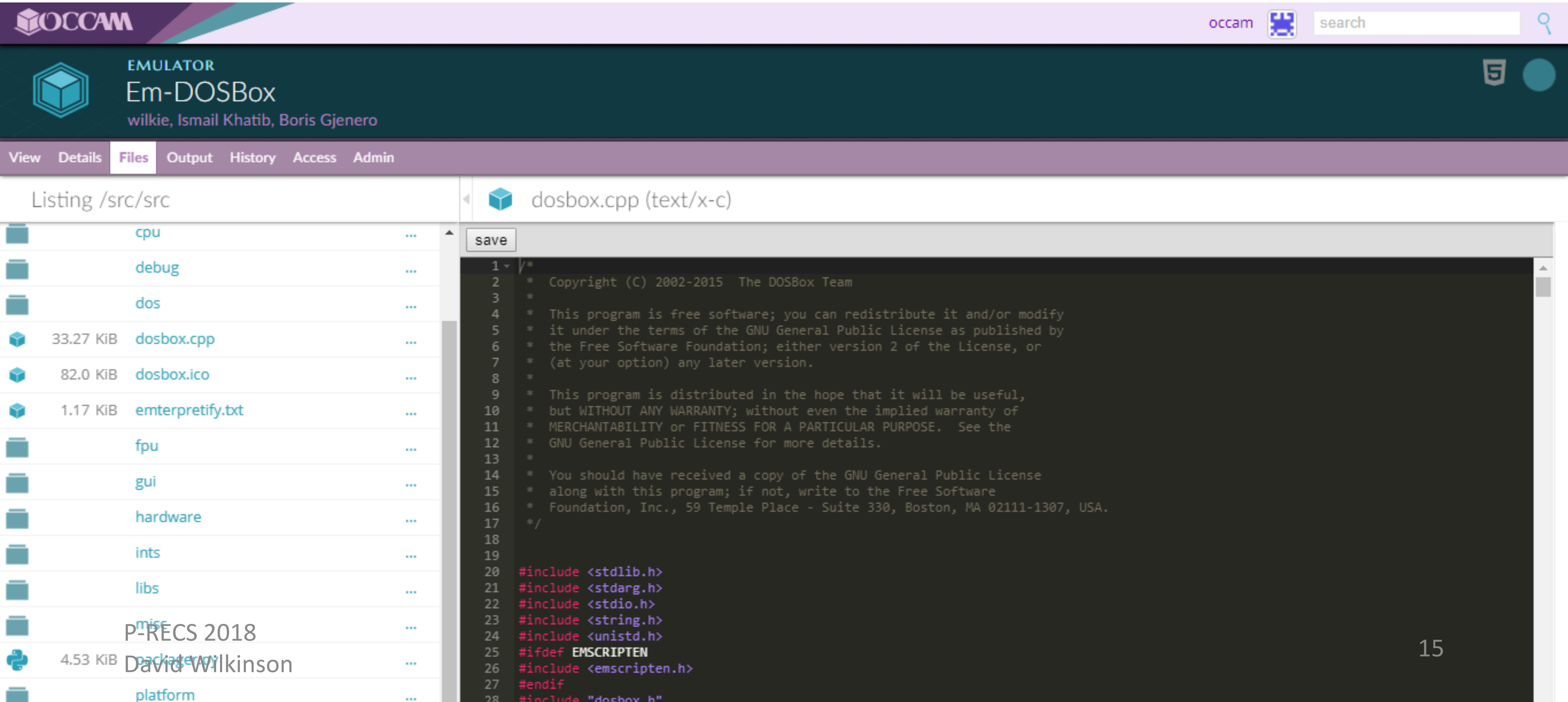
DOOM 1.9 Shareware Installer

Id Software

Details
Run
Files
Output
History
Access
Admin
DOOM
TM
id SOFTWARE
PROVIDED BY id FREE OF CHARGE • SUGGESTED RETAIL PRICE $9.00 • id SOFTWARE, © 1993

Software Provenance for Emulation

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



Listing /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	...
4.53 KiB	package.py	...
	platform	...

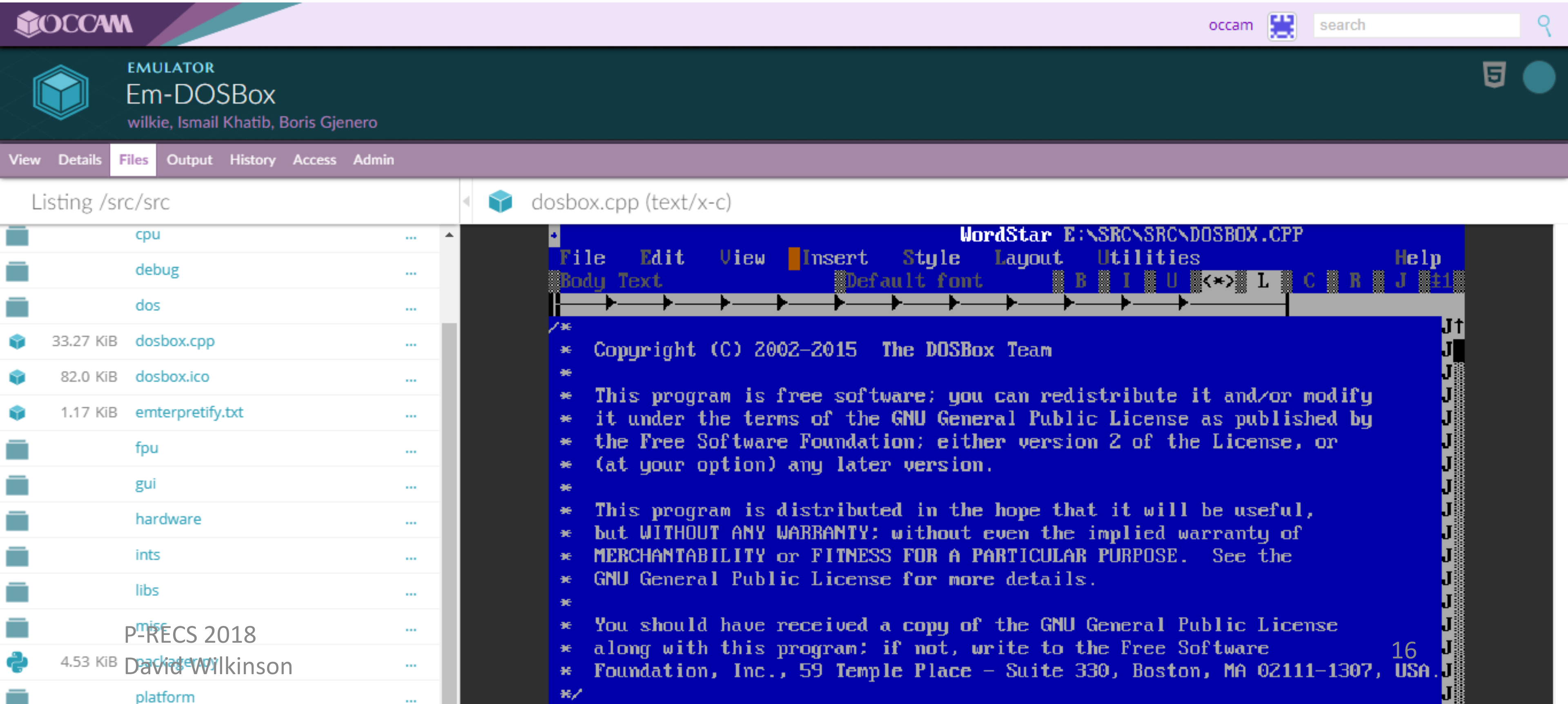
dosbox.cpp (text/x-c)

```
/*
 *  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.
 */


#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#ifdef EMSCRIPTEN
#include <emscripten.h>
#endif
#include "dosbox.h"
```

Example: DOS Applications

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



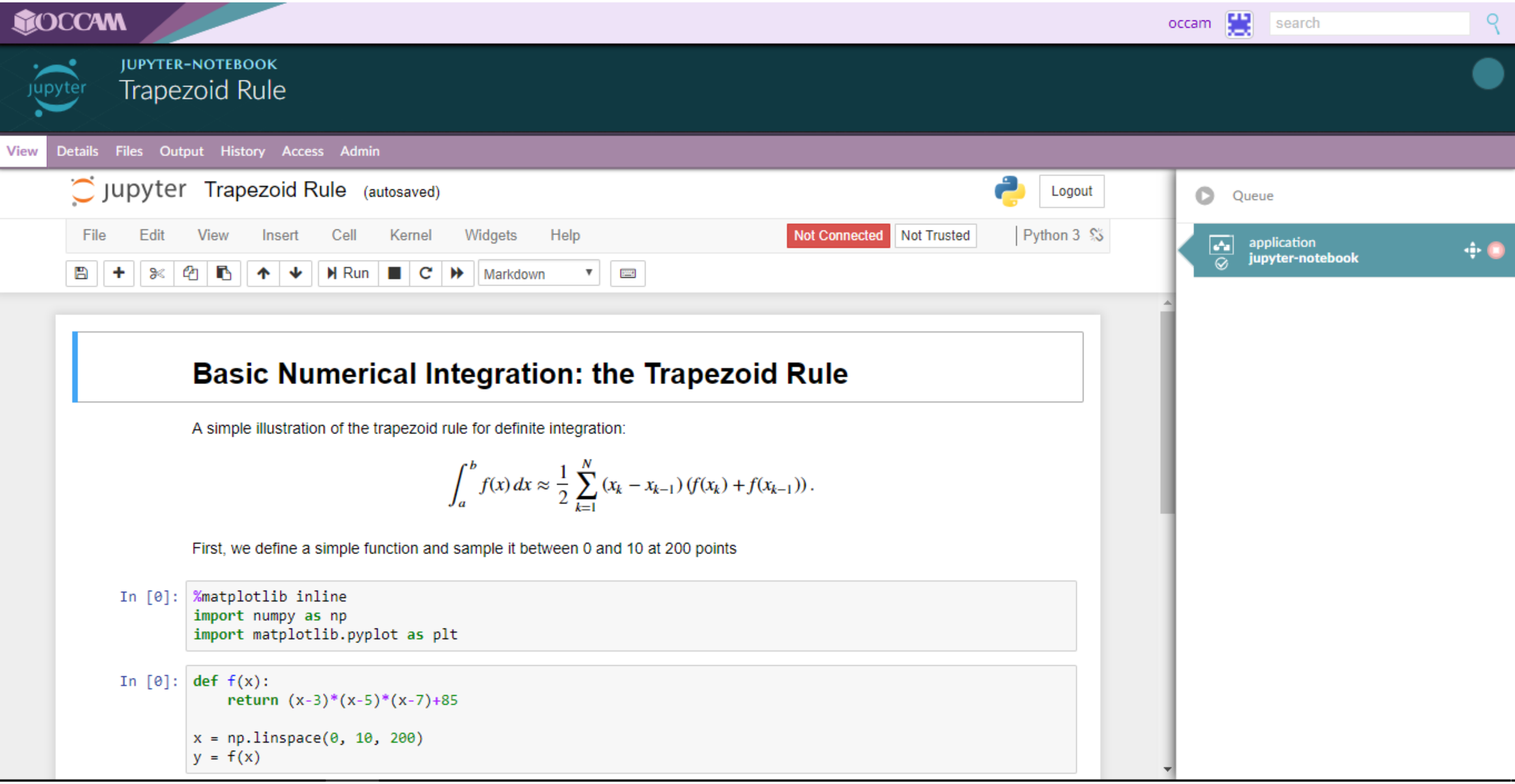
```
/*
 *  Copyright (C) 2002-2015  The DOSBox Team
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later 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.
 */
```

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



Basic Numerical Integration: the Trapezoid Rule

A simple illustration of the trapezoid rule for definite integration:

$$\int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{k=1}^{N}(x_k - x_{k-1})(f(x_k)+f(x_{k-1})).$$

First, we define a simple function and sample it between 0 and 10 at 200 points

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

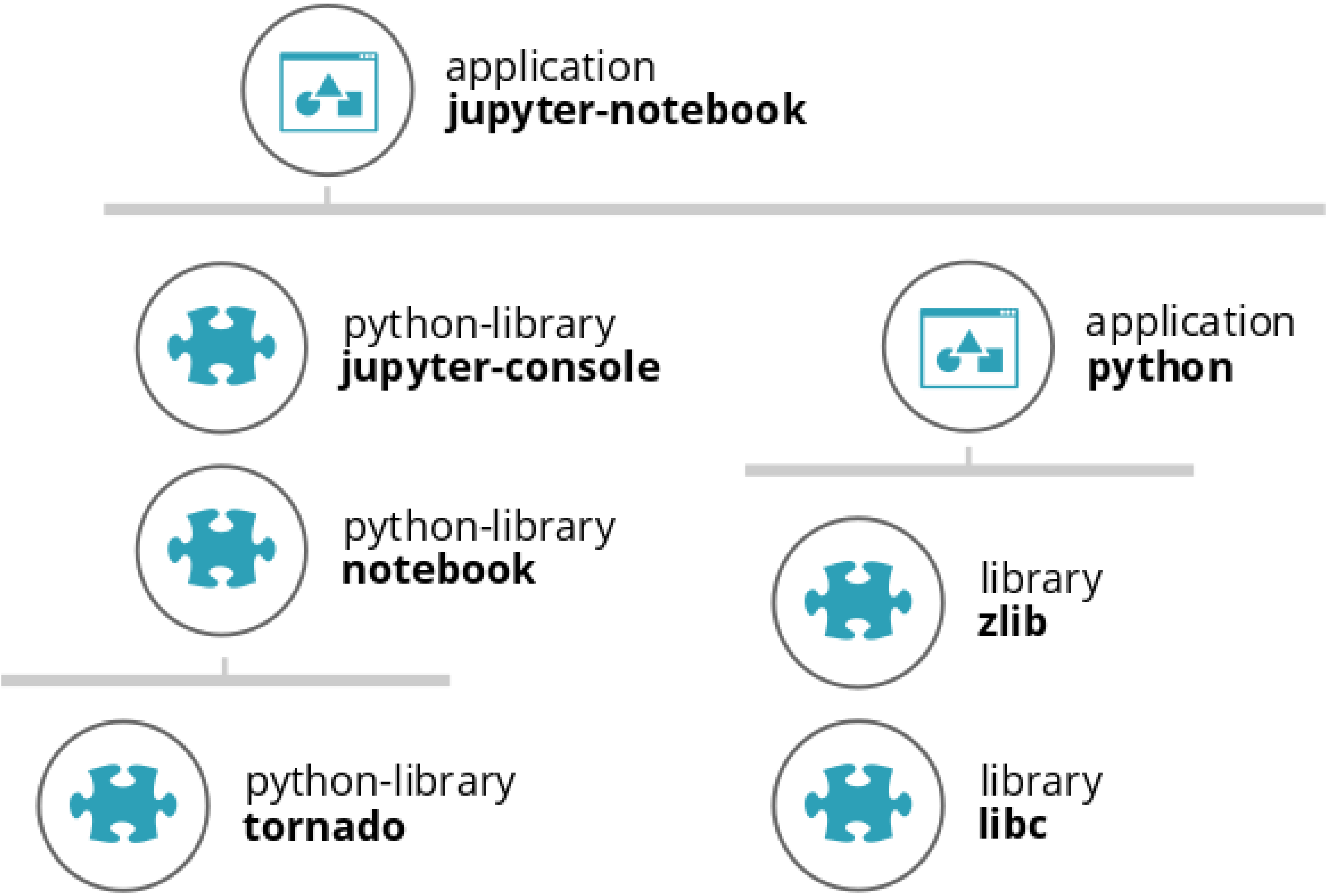
```
def f(x):
    return (x-3)*(x-5)*(x-7)+85

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

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



application
jupyter-notebook

python-library
jupyter-console

python-library
notebook

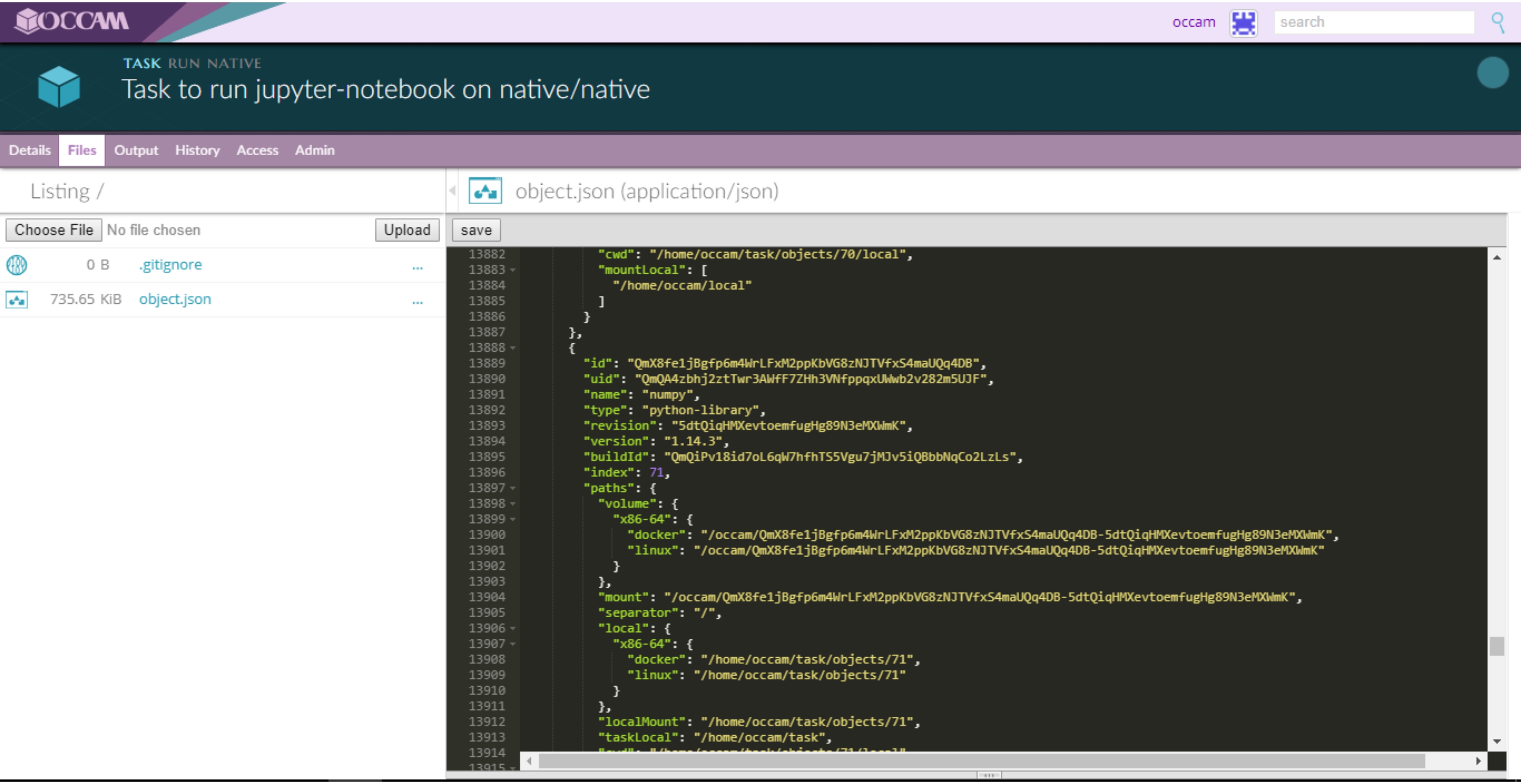
python-library
tornado

application
python

library
zlib

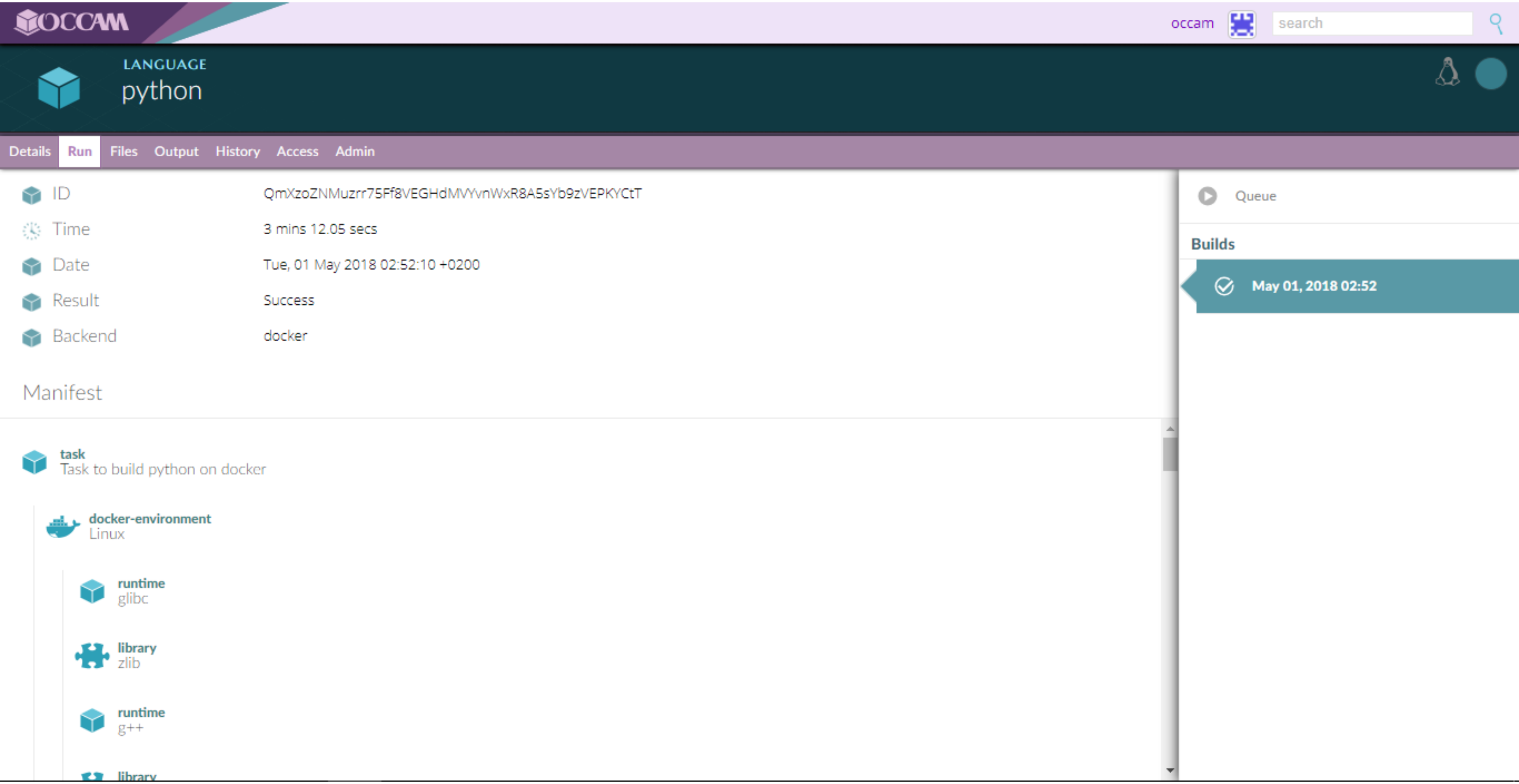
library
libc

Example: Jupyter Notebook Runtime Provenance via Manifests:



```
        "cwd": "/home/occam/task/objects/70/local",
        "mountLocal": [
          "/home/occam/local"
        ]
      }
    },
    {
      "id": "QmX8fe1jBgfp6m4WrLFxM2ppKbVG8zNJTVfxS4maUQq4DB",
      "uid": "QmQA4zbhj2ztTwr3AWfF7ZHh3VNfppqxUWwb2v282m5UJF",
      "name": "numpy",
      "type": "python-library",
      "revision": "5dtQiqHMXevtoemfugHg89N3eMXWmK",
      "version": "1.14.3",
      "buildId": "QmQiPv18id7oL6qW7hfhTS5Vgu7jMJv5iQBbbNqCo2LzLs",
      "index": 71,
      "paths": {
        "volume": {
          "x86-64": {
            "docker": "/occam/QmX8fe1jBgfp6m4WrLFxM2ppKbVG8zNJTVfxS4maUQq4DB-5dtQiqHMXevtoemfugHg89N3eMXWmK",
            "linux": "/occam/QmX8fe1jBgfp6m4WrLFxM2ppKbVG8zNJTVfxS4maUQq4DB-5dtQiqHMXevtoemfugHg89N3eMXWmK"
          }
        },
        "mount": "/occam/QmX8fe1jBgfp6m4WrLFxM2ppKbVG8zNJTVfxS4maUQq4DB-5dtQiqHMXevtoemfugHg89N3eMXWmK",
        "separator": "/",
        "local": {
          "x86-64": {
            "docker": "/home/occam/task/objects/71",
            "linux": "/home/occam/task/objects/71"
          }
        },
        "localMount": "/home/occam/task/objects/71",
        "taskLocal": "/home/occam/task",
        "cwd": "/home/occam/task/objects/71/local"
```

Example: Jupyter Notebook
The Build Provenance of Python 3



ID	QmXzoZNMuzrr75Ff8VEGHdMVYvnWxR8A5sYb9zVEPKYCtT
Time	3 mins 12.05 secs
Date	Tue, 01 May 2018 02:52:10 +0200
Result	Success
Backend	docker

Manifest

- **task** Task to build python on docker
  - **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