Web Application Deployment: Render

Robert M. Dondero, Ph.D
Princeton University

Objectives

- The lecture will cover:
  - Web app deployment options
  - How to deploy a database and web app to Render

Agenda

- **Deployment options**
- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Deployment Options

- **Question**:
  - Where/how might you run Penny app so it's constantly available to the world…
    - Using a production-quality HTTP server?
    - Using a production-quality DBMS?

Deployment Options

- **Answer 1**: Your computer

Deployment Options

- **Answer 2**: Princeton CS Dept HTTP servers
  - See https://csguide.cs.princeton.edu/publishing/webpages
  - Provides MySQL databases
  - Pros and cons:
    - (pro) XXX.cs.princeton.edu address
    - (pro) Free
    - (con) Requires CS Dept approval
    - (con) No access to server logs

Deployment Options

- **Answer 3**: Princeton OIT servers
  - Pros and cons:
    - (pro) Free
    - (con) Requires OIT approval
    - (con) Requires sponsorship of some Princeton department
    - (con) Difficult; requires intense cooperation with OIT

Deployment Options

- **Answer 4**: Commercial cloud service
  - Render, Heroku, DigitalOcean, …
  - Pros and cons:
    - (pro) The service does sys admin for you
    - (con) You may need to pay for it!
  - The best option for most COS 333 projects

Deployment Options

- But *which* cloud service?

- Must choose a service for:
  - Web application
  - Database

Deployment Options

A little COS 333 history:

Historical Phase	Application Cloud Service	DB Cloud Service
1	Heroku (free forever)	Heroku (free forever)
Heroku eliminated its free tier		
2	Render (free forever)	Render (free for 3 mths)
2	Render (free forever)	ElephantSQL (free forever)
Heroku teamed with GitHub: GitHub Student Developer Pack		
3	Render (free forever)	Render (free for 3 mths)
3	Render (free forever)	ElephantSQL (free forever)
3	Heroku with GitHub Student Developer Pack (free for 1 yr)	Heroku with GitHub Student Developer Pack (free for 1 yr)

Deployment Options

A little COS 333 history (cont.):

Historical Phase	Application Cloud Service	DB Cloud Service
Render limited its free tier DBs to 1 month ElephantSQL went out of the DB business		
4	Render (free forever)	Render (free for 1 mth)
4	Render (free forever)	Neon (free forever)
4	Heroku with GitHub Student Developer Pack (free for 1 yr)	Heroku with GitHub Student Developer Pack (free for 1 yr)
Heroku extended GitHub Student Developer Pack to 2 yrs		
5	Render (free forever)	Render (free for 1 mth)
5	Render (free forever)	Neon (free forever)
5	Heroku with GitHub Student Developer Pack (free for 2 yrs)	Heroku with GitHub Student Developer Pack (free for 2 yrs)

Deployment Options

Web app cloud services:

Web App Cloud Service	Cost
Render.com Free	Free for 750 hours of running time per month *
Render.com Starter	$7/month
Heroku Eco	$5 for 1000 dyno hours per month **
Heroku Basic	~$0.01 per hour (max of $7 per month)

* Slow deployment; slow at runtime?

** *GitHub Student Developer Pack* provides $13/month for 2 years; cannot renew

Deployment Options

Database cloud services (cont.):

DB Cloud Service	Time Period	Size	Concurrent Connections	Cost
Render.com Free	1 month *	1 GB	97	0
Render.com Starter	Unlimited	1 GB	97	$7/month
Neon Free Plan	Unlimited	0.5GB	10000	0
Neon Launch	Unlimited	10GB	10000	$19/month
Heroku Mini	Unlimited	1 GB	20	$5/month **
Heroku Basic	Unlimited	10 GB	20	$10/month

* Can create another

** *GitHub Student Developer Pack* provides $13/month for 2 years; cannot renew

Deployment Options

- All of those DB options (can) use ***PostgreSQL***

Deployment Options

Michael Stonebreaker

Deployment Options

- PostgreSQL assessment (vs. SQLite)
  - (con) Setup on local computer (not shown) is **much** more complex
  - (con) Setup on cloud service is more complex
  - (pro) Production-quality
    - Distinct process
    - Authentication
    - Row-level (vs. database-level) locking
    - Reasonable for apps deployed to the cloud
    - …

Agenda

- Deployment options
- **Render**
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Render

Anurag Goel
Founder and CEO

Render

- Render
  - Can create **app** or **DB** first
- Heroku
  - Must create **app** first
  - Then create **DB** specifically for the app

Agenda

- Deployment options
- Render
- **Creating the DB (demo)**
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Creating the DB

- Create a Render account
  - Browse to https://render.com
    - Clicked *Get Started for Free*
    - Follow the instructions
      - Need not provide a credit card number

Creating the DB

- Log into Render
  - Browse to https://dashboard.render.com/
    - For *Email address* enter `youremailaddress`
    - For *Password* enter `yourpassword`

Creating the DB

- From your Render dashboard
  - Click + *Add New*
  - Select *Postgres*

Creating the DB

- In the resulting page
  - For *Name* enter `pennydb`
  - For *Region* choose `Ohio (US East)`
  - For *Instance Type* choose `Free`
  - Click *Create Database*
  - Wait (~1 minute) for the database to be created

Creating the DB

- In the resulting page (cont.)
  - Note the *Internal Database URL*
    - **Save it someplace**
    - Let's call it ***yourinternaldburl***
  - Note the *External Database URL*
    - **Save it someplace**
    - Let's call it ***yourexternaldburl***

Agenda

- Deployment options
- Render
- Creating the DB (demo)
- **Using the DB (demo)**
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Using the DB

- From a command-line
  - Install `psql`
    - Without installing PostgreSQL server
    - See https://www.risingwave.dev/docs/current/install-psql-without-postgresql/

Using the DB

- From a command-line (cont.)
  - Install `psql` (Mac)
    - First install homebrew; then:

```
$ brew update
$ brew install libpq
$ brew link --force libpq
$
```

Using the DB

- From a command-line (cont.)
  - Install `psql` (MS Windows)
  - Download the installer from https://www.postgresql.org/download/windows/
  - Run the installer
    - Select *Command Line Tools* and uncheck other options during installation

Using the DB

- From a command-line (cont.)
  - Install `psql` (Linux)

```
$ sudo apt update
$ sudo apt install postgresql-client
$
```

Using the DB

Some `psql` statements

sqlite3 Statement	psql Statement
`.help`	`\h`
`.quit`	`\q`
`.tables`	`\d`
`.schema` *table*	`\d` *table*
`.read` *file*	`\i` *file*

Using the DB

```
$ cat penny.sql
DROP TABLE IF EXISTS books;

CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);

INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');
$
```

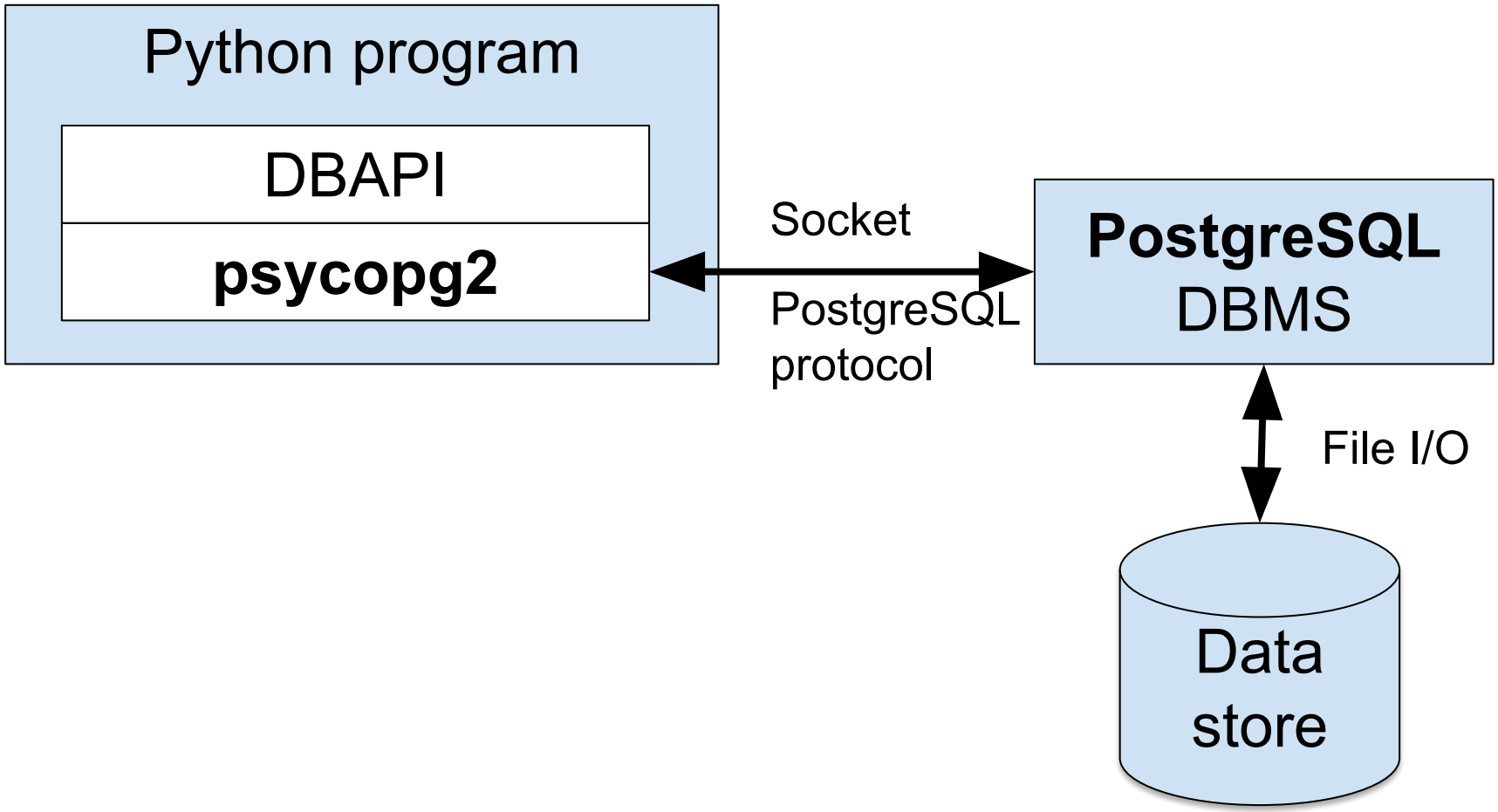
Using the DB

```
$ psql yourexternaldburl
yourdbname=> \i penny.sql
DROP TABLE
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
yourdbname=> SELECT * FROM books;
 isbn |   author    |          title
------+-------------+--------------------------
 123  | Kernighan | The Practice of Programming
 234  | Kernighan | The C Programming Language
 345  | Sedgewick | Algorithms in C
(3 rows)

yourdbname=> \q
$
```

Agenda

- Deployment options
- Render
- Creating the DB (demo)
- Using the DB (demo)
- **Using the DB: Python**
- Defining the app
- Deploying the app (demo)
- Using the app (demo)

Using the DB: Python



Using the DB: Python

- Installing the `psycopg2` driver:

```
$ activate333
$ python -m pip install psycopg2
$
```

or, if that fails:

```
$ activate333
$ python -m pip install psycopg2-binary
$
```

Using the DB: Python

- See **pennyrender/database1.py**
  - Baseline version
  - Uses SQLite

```
$ python database1.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

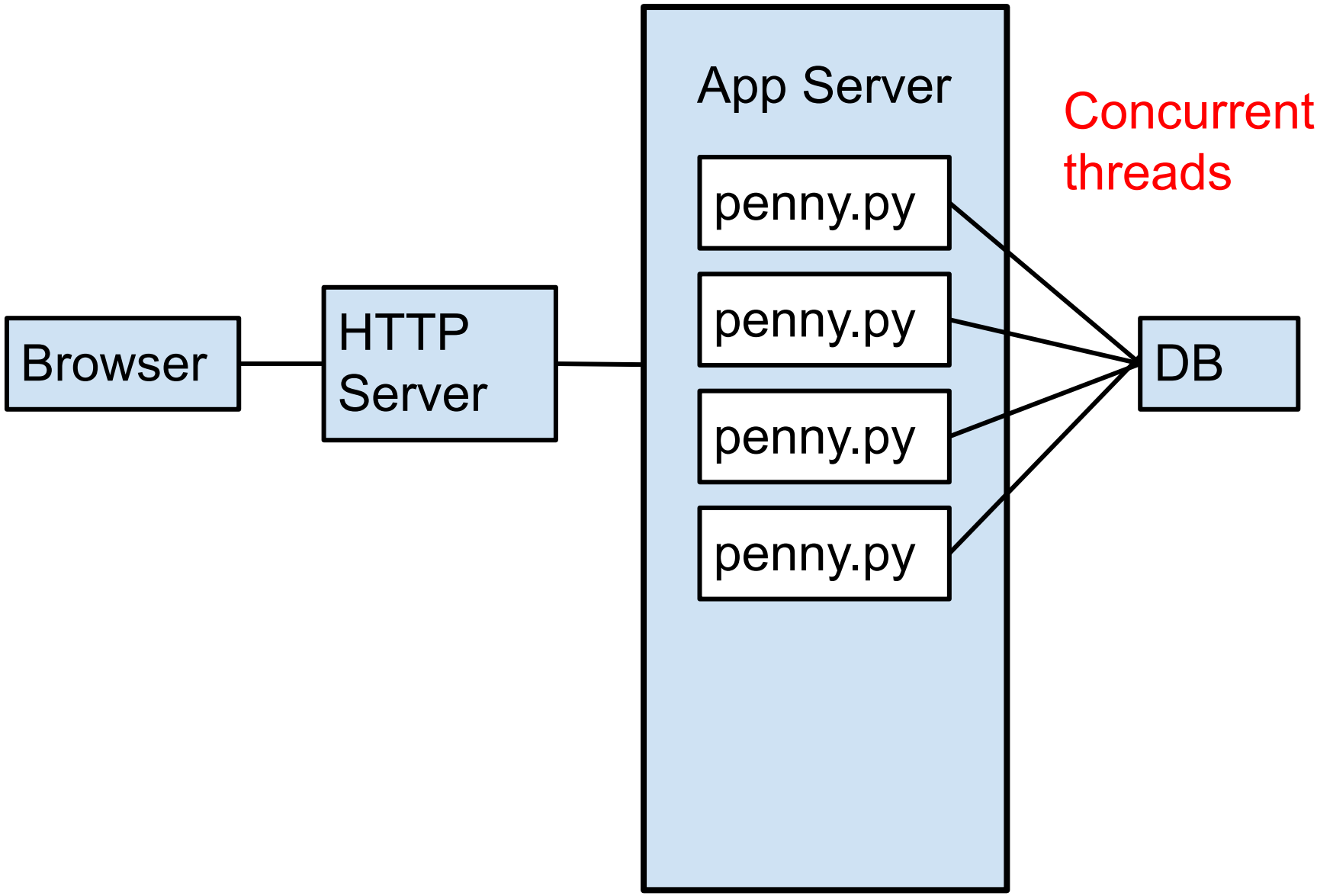
- See **pennyrender/database2.py**
  - Uses PostgreSQL
  - Uses an environment variable

```
$ export DATABASE_URL=yourexternaldburl
$ python database2.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

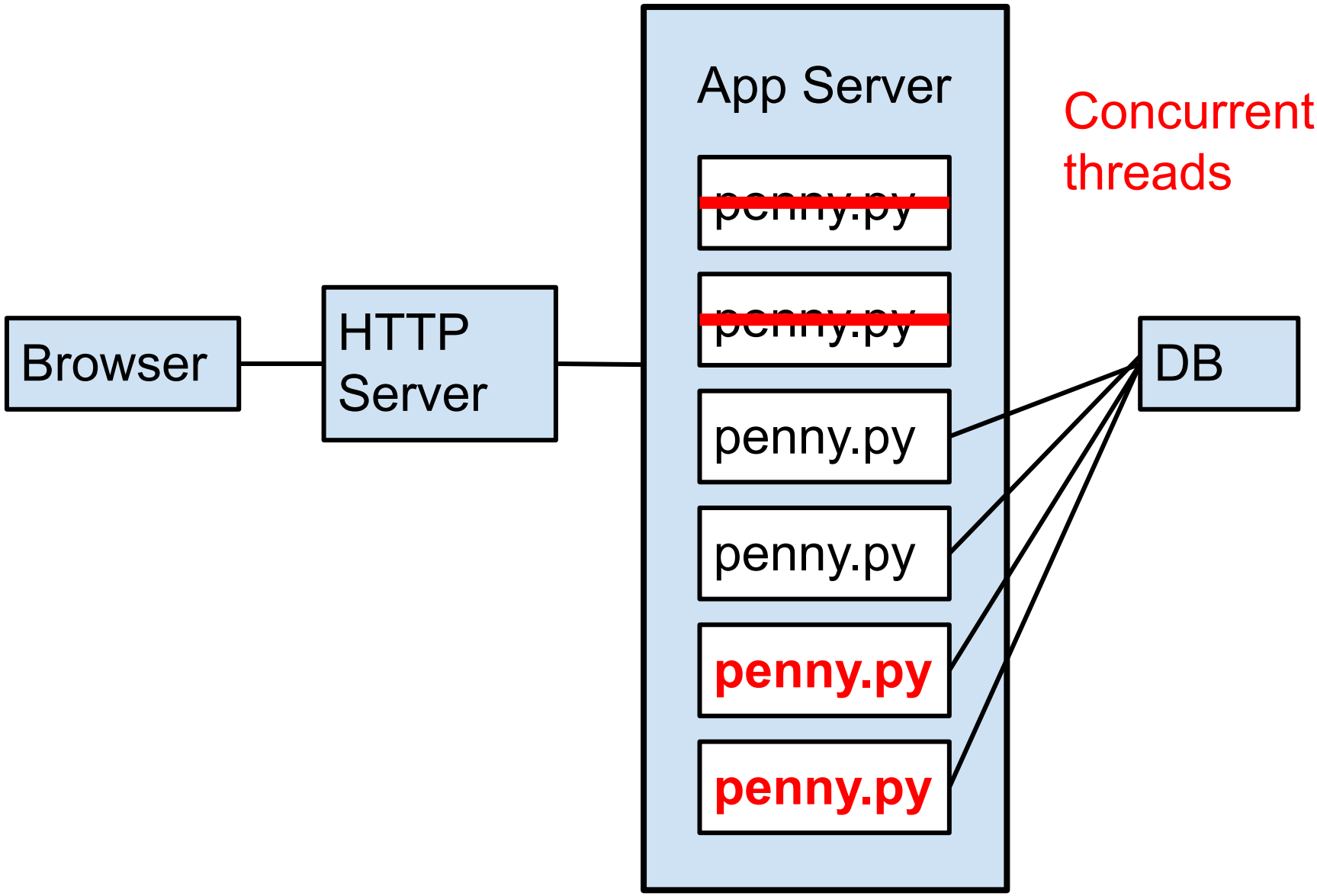
Using the DB: Python

- **Problem**:
  - In the context of a Flask app…
  - database2.py is too slow

Using the DB: Python



Using the DB: Python



Using the DB: Python

- **Solution**:
  - ***DB connection pooling***
    - Maintain a “pool” (queue) of open DB connections that threads can use

Using the DB: Python

- See **pennyrender/database3.py**
  - Uses DB connection pooling

```
$ export DATABASE_URL=yourexternaldburl
$ python database3.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- **Problems???**
  - database3.py requires extra logic for connection pooling
  - database3.py requires maintenance pgmmers to know SQL
  - database3.py is (somewhat) specific to PostgreSQL

Using the DB: Python

- **Solution**: ***SQLAlchemy***
  - An ***Object Relational Mapper (ORM)*** for Python
    - Maps each table to a class
    - Maps each row to an object
  - See optional lecture for more thorough description

Using the DB: Python

- Installing SQLAlchemy

```
$ activate333
$ python -m pip install SQLAlchemy
$
```

Using the DB: Python

- See **pennyrender/database.py**
  - Uses ***SQLAlchemy***

```
$ export DATABASE_URL=yourexternaldburl
$ python database.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

Using the DB: Python

- See **pennyrender/database.py** (cont.)

```
$ export DATABASE_URL=sqlite:///penny.sqlite
$ python database.py
123
Kernighan
The Practice of Programming

234
Kernighan
The C Programming Language

$
```

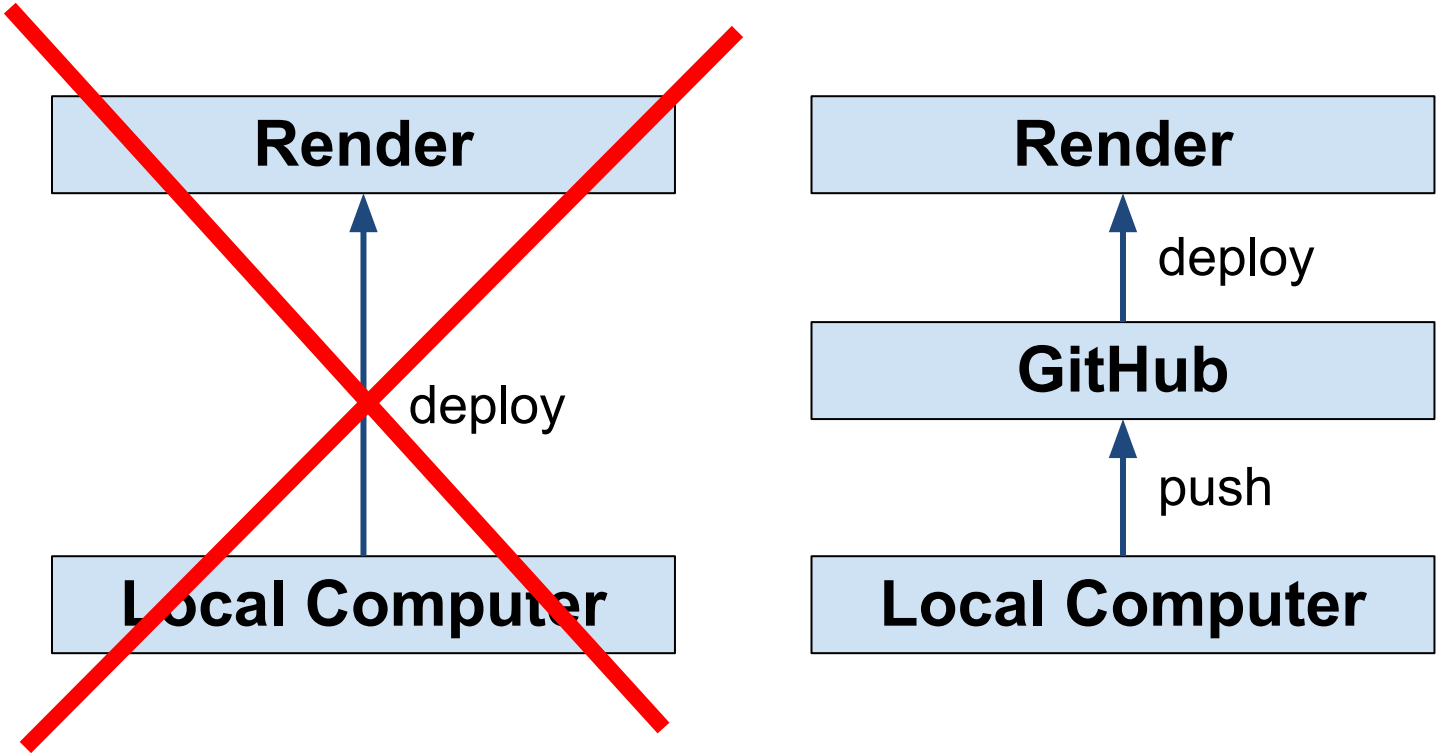
Also works with SQLite on local computer!!!

Agenda

- Deployment options
- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- **Defining the app**
- Deploying the app (demo)
- Using the app (demo)

Defining the App

- Deployment



Defining the App

- You must change names as appropriate…

Defining the App

- After performing setup steps in *Git and GitHub Primer* document...

- Create a GitHub repo
  - Browse to https://github.com
    - Click the *New* button
      - For *Repository name* enter `pennyrender`
      - Click the *Private* radio button
      - Check the *Add a README file* check button
      - Click the *Create Repository* button

Defining the App

- Clone the GitHub repo to your local computer

```
$ git clone https://github.com/rdondero/pennyrender.git
```

  - Creates `pennyrender` dir on local computer

Defining the App

- Create these files in pennyrender dir:
  - From the PennyFlaskJinja app
    - runserver.py (optionally)
    - penny.sql
    - penny.sqlite (optionally)
    - header.html, footer.html, index.html, searchform.html, searchresults.html
    - penny.py
  - From this lecture
    - database.py

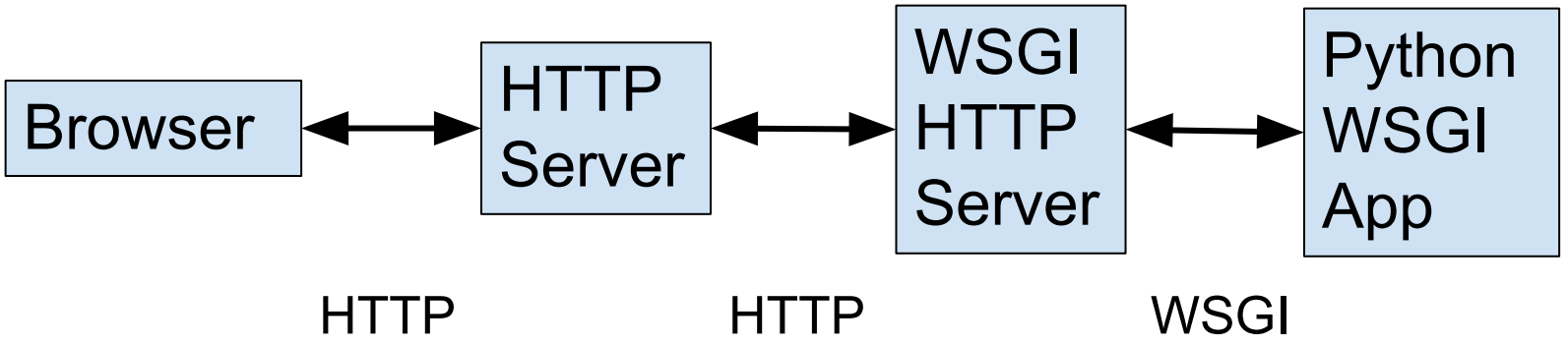
Defining the App

- Also create in pennyrender dir:
  - **requirements.txt**

```
Flask
psycopg2
SQLAlchemy
python-dotenv
gunicorn
```

  - Tells Render what additional modules the app uses
  - Create manually, or issue command:
    `python -m pip freeze > requirements.txt`

Aside: Gunicorn



Chrome	Nginx	***Gunicorn***	Flask
Firefox	Apache	Phusion Passenger	Django

Defining the App

- Stage the app to the local git repo, commit the app to the local git repo, and push it to the GitHub repo

```
$ cd pennyrender
$ git add .
$ git commit -m "initial load"
$ git push origin main
```

Agenda

- Deployment options
- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- **Deploying the app (demo)**
- Using the app (demo)

Deploying the App

- “Install Render” on GitHub
  - Inform GitHub that Render is permitted to access the repo
  - Browse to https://github.com/apps/render/installations/new
    - Click *rdondero*

Deploying the App

- “Install Render” on GitHub (cont.)
  - In the resulting Render page
    - Click *Only select repositories*
    - Click *Select repositories*
    - Choose *rdondero/pennyrender*
    - Click *Save*

Deploying the App

- (If necessary) Browse to Render dashboard
  - Browse to https://dashboard.render.com
    - Click *Dashboard*

Deploying the App

- Create a new Render app
  - In the *Dashboard* page
    - Click + *Add New*
    - Select *Web Service*
    - Select *Git Provider*
    - Select rdondero/pennyrender

Deploying the App

- Create a new app (cont.)
  - In the resulting page
    - For *Name* enter **`pennyrender`**
    - For *Region* choose **`Ohio (US East)`**
    - For *Start Command* enter `gunicorn` **`penny:`**`app`
    - For *Plans* choose `Free`
    - In *Environment Variables*
      - For *NAME_OF_VARIABLE* enter `DATABASE_URL`
      - For *value* enter *`yourinternaldburl`*
    - In *Advanced*
      - Set *Auto-Deploy* to *`No`*
    - Click *Deploy Web Service*

Deploying the App

- Create a new app (cont.)
  - The app deployment begins
    - Wait up to 5 minutes for deployment to finish 😖

Deploying the App

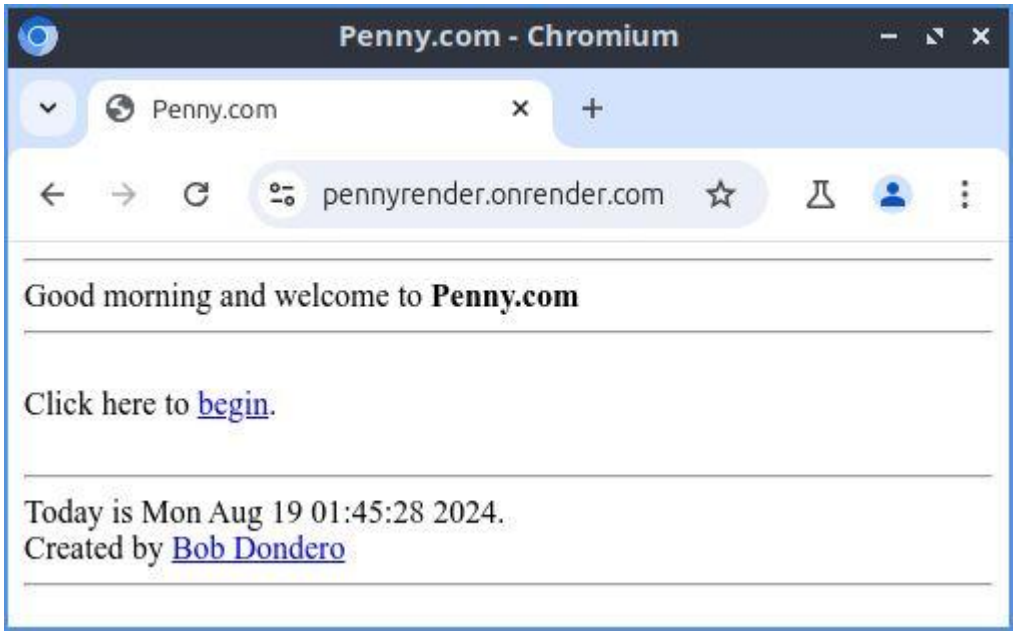
- Thereafter:
  - Push new code to pennyrender GitHub repo
  - In Render, click *Manual Deploy*
  - Repeat as necessary

Agenda

- Deployment options
- Render
- Creating the DB (demo)
- Using the DB (demo)
- Using the DB: Python
- Defining the app
- Deploying the app (demo)
- **Using the app (demo)**

Using the App

- Browse to the app!
  - https://pennyrender.onrender.com



Summary

- The lecture has covered:
  - Web app deployment options
  - How to deploy a database and web app to Render
- See also:
  - **Appendix 1**: DB Backup and Restore
  - **Appendix 2**: Render vs. Heroku
  - **Appendix 3**: Cloud Service Types

Appendix 1:
PostgreSQL Backup and Restore

PostgreSQL Backup and Restore

- **Step 1**
  - (If necessary) install `pg_dump`
    - It probably was installed when you installed `psql`

PostgreSQL Backup and Restore

- **Step 2**
  - Export the Render DB to a text file

```
$ pg_dump yourexternaldburl > penny.postgresql
$
```

PostgreSQL Backup and Restore

- **Step 3**
  - Delete the old Render DB
    - Perform through the Render website
- **Step 4**
  - Create a new Render DB
    - Use the instructions provided previously in this lecture

PostgreSQL Backup and Restore

- **Step 5**
  - Import from the text file to the new Render DB

```
$ psql yournewexternaldburl
==> \i penny.postgresql
==> \q
$
```

PostgreSQL Backup and Restore

- **Step 6**
  - Change the value of the `DATABASE_URL` env var of your deployed Render app to *yournewinternaldburl*
- **Step 7**
  - Re-deploy your app to Render

Appendix 2:
Render vs. Heroku

Render vs. Heroku

- Render
  - (pro) Easier to create DB
  - (pro) Easier to create app
  - (pro) Stable DB URL
  - (pro) Free
- Heroku
  - (pro) Faster deploys
  - (pro) Faster initial app launch
  - (pro) Faster app???
  - (pro) DB does not expire

Render vs. Heroku

- Render
  - (con) Slower deploys
  - (con) Slower initial app launches
  - (con) Slower app???
  - (con) DB expires after 1 month
    - But can move data to new DB
- Heroku
  - (con) Harder to create app
  - (con) Harder to create DB
  - (con) Unstable DB URL
  - (con) Costs money
    - But non-renewable *GitHub Student Developer Pack* covers 2 years

Appendix 3:
Cloud Service Types

Cloud Service Types

- ***Software as a Service (SaaS)***
  - "The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure."
    -- https://en.wikipedia.org/wiki/Cloud_computing
  - Examples: Google Docs, Microsoft Word Online

Cloud Service Types

- ***Platform as a Service (PaaS)***
  - "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider."
    -- https://en.wikipedia.org/wiki/Cloud_computing
  - Examples: Render, Heroku, Google App Engine

Cloud Service Types

- ***Infrastructure as a Service (IaaS)***
  - "The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications."
    -- https://en.wikipedia.org/wiki/Cloud_computing
  - Examples: Amazon Web Services (AWS), Google Cloud Platform, Microsoft Azure

Cloud Service Types

- Which **cloud service type** for Penny?
  - SaaS: impossible; too narrow
  - IaaS: possible, but too broad
  - **PaaS**: just right!