Week 2 – Python Bootcamp

02613 Python and High-Performance Computing

Last week in 02613…

Corrected/updated exam date

- Exam on 1 June 2026
  - DTU’s websites (student.dtu.dk) authoritative
  - DTU’s calendar for Spring semester 2026 updated last week

Week 1 lecture recording not available

VPN issues

- Upgrade to the latest client version https://net.ait.dtu.dk/vpn

- If that does not help, contact IT itservice@dtu.dk.

- Alternative: SSH keys https://www.hpc.dtu.dk/?page_id=4317
  - Setup once on a DTU network or on VPN
  - From there, no VPN required

- “Emergency” alternative: https://remote.dtu.dk
  - Windows desktop ”within DTUs walls”
  - No VPN required (but maybe need 2FA)

Login nodes

Four login nodes available

```
ssh <username>@login.hpc.dtu.dk
ssh <username>@login2.hpc.dtu.dk
ssh <username>@login.gbar.dtu.dk
ssh <username>@login2.gbar.dtu.dk
```

Login nodes

Remember:

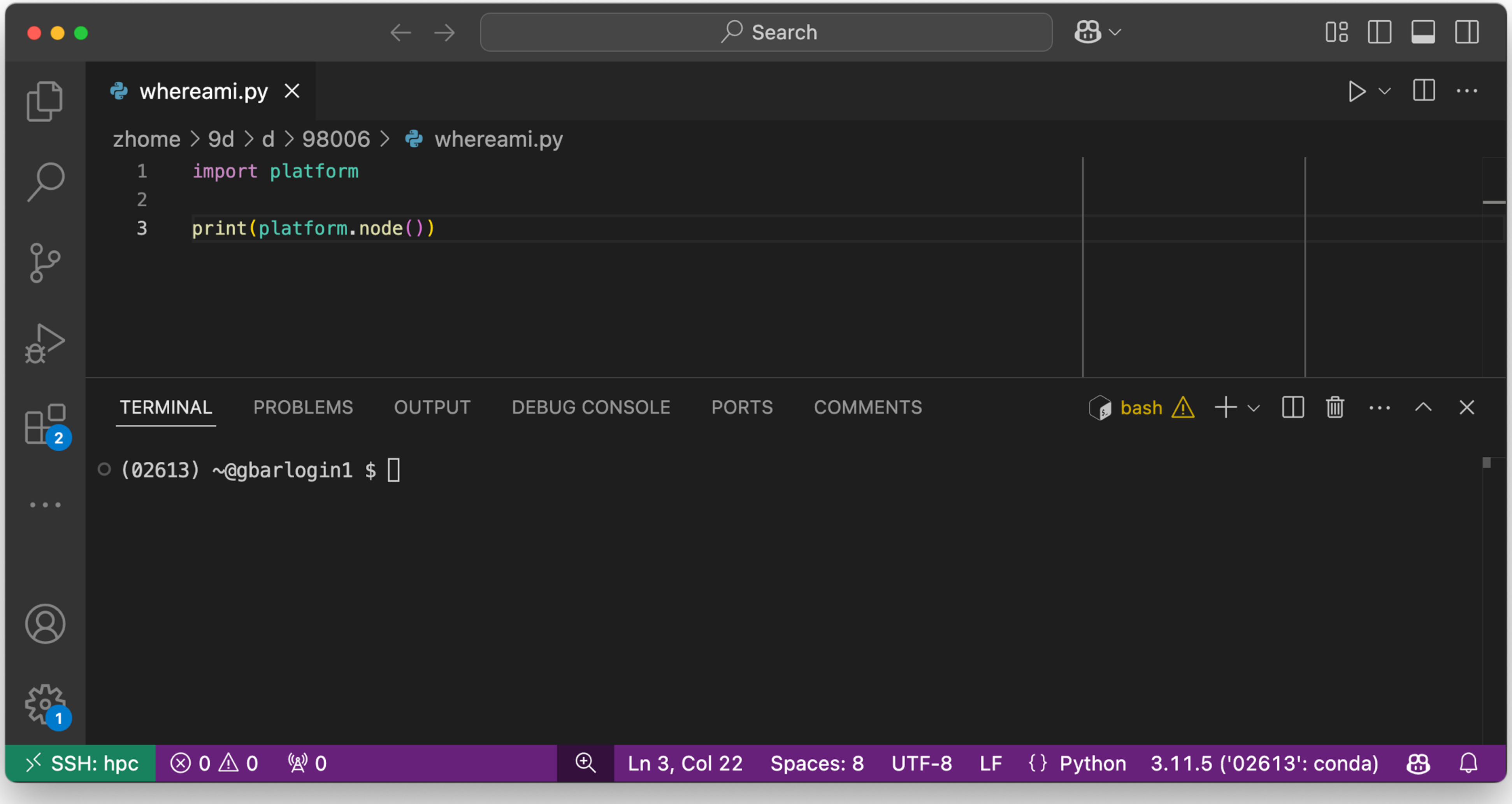
Login nodes are *NOT* for computations

Please keep them free to handle logins

Use **linuxsh** for an interactive session (test, debug)

Use **bsub** for “real” computations

Login nodes and VS Code

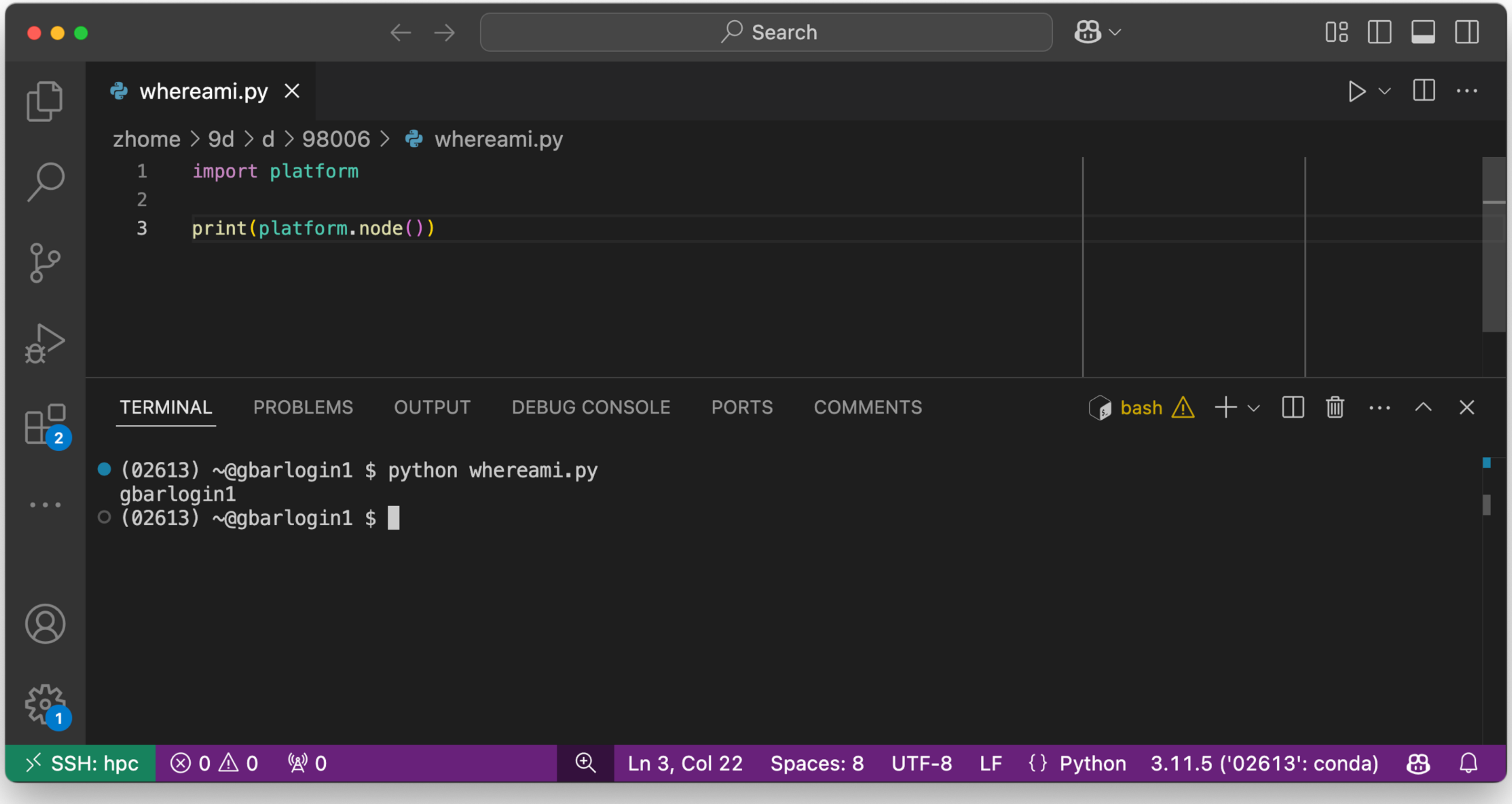


```python
import platform

print(platform.node())
```

```
(02613) ~@gbarlogin1 $
```

Login nodes and VS Code

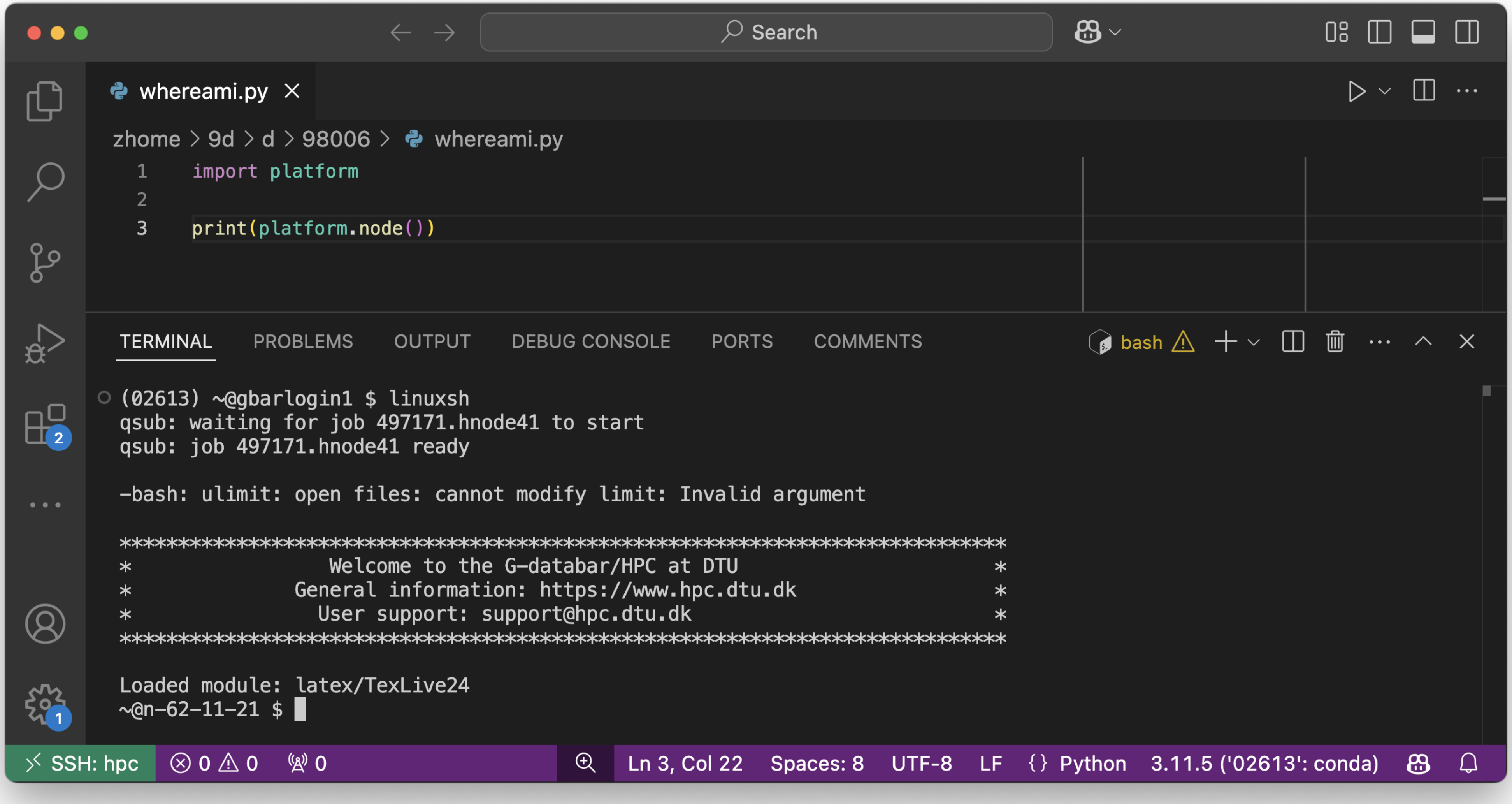


```python
import platform

print(platform.node())
```

```
(02613) ~@gbarlogin1 $ python whereami.py
gbarlogin1
(02613) ~@gbarlogin1 $ 
```

Login nodes and VS Code



```python
import platform

print(platform.node())
```

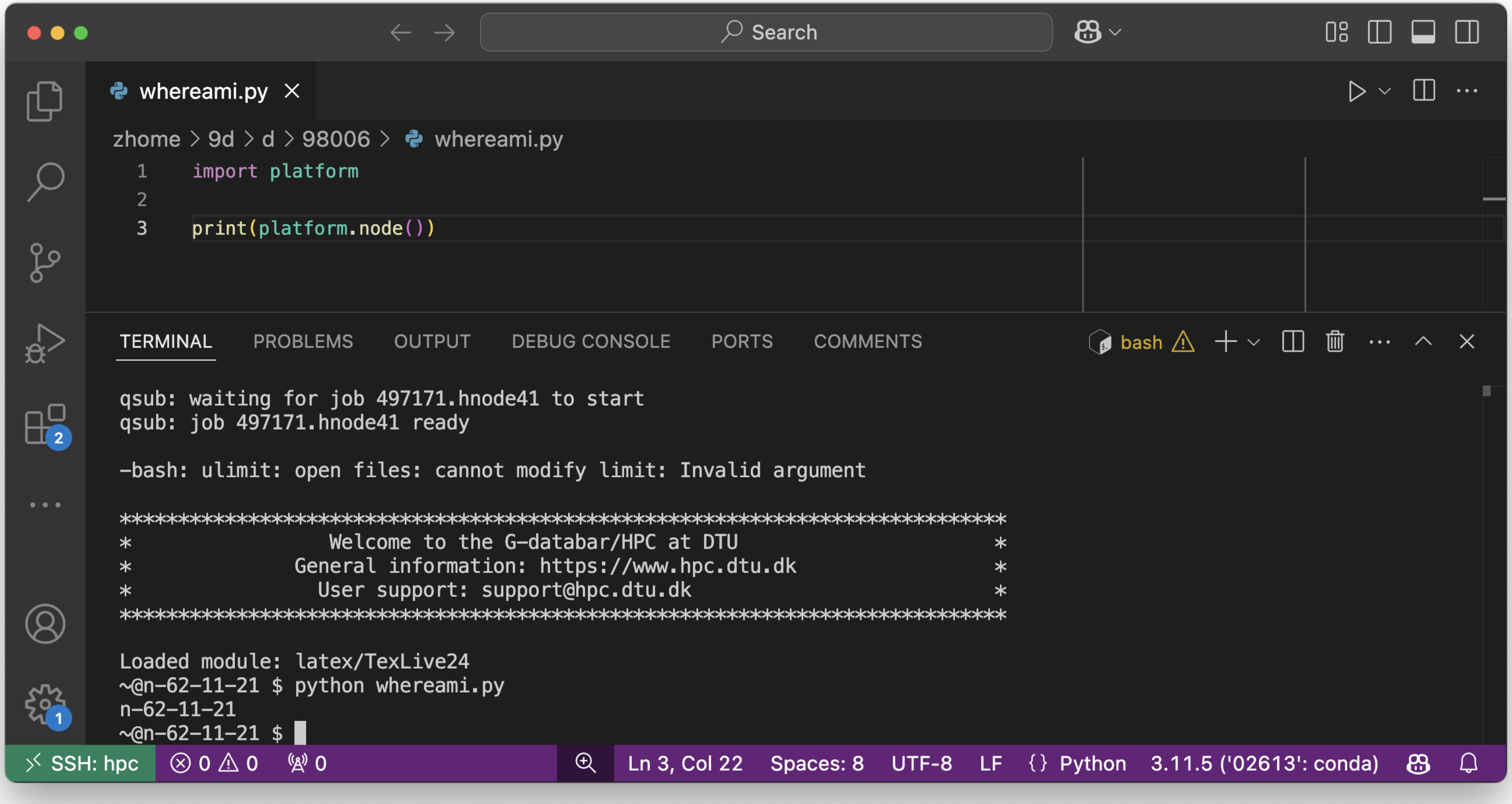
```
(02613) ~@gbarlogin1 $ linuxsh
qsub: waiting for job 497171.hnode41 to start
qsub: job 497171.hnode41 ready

-bash: ulimit: open files: cannot modify limit: Invalid argument

****************************************************************************
*                  Welcome to the G-databar/HPC at DTU                     *
*               General information: https://www.hpc.dtu.dk                *
*                 User support: support@hpc.dtu.dk                         *
****************************************************************************

Loaded module: latex/TexLive24
~@n-62-11-21 $
```

Login nodes and VS Code



```python
import platform

print(platform.node())
```

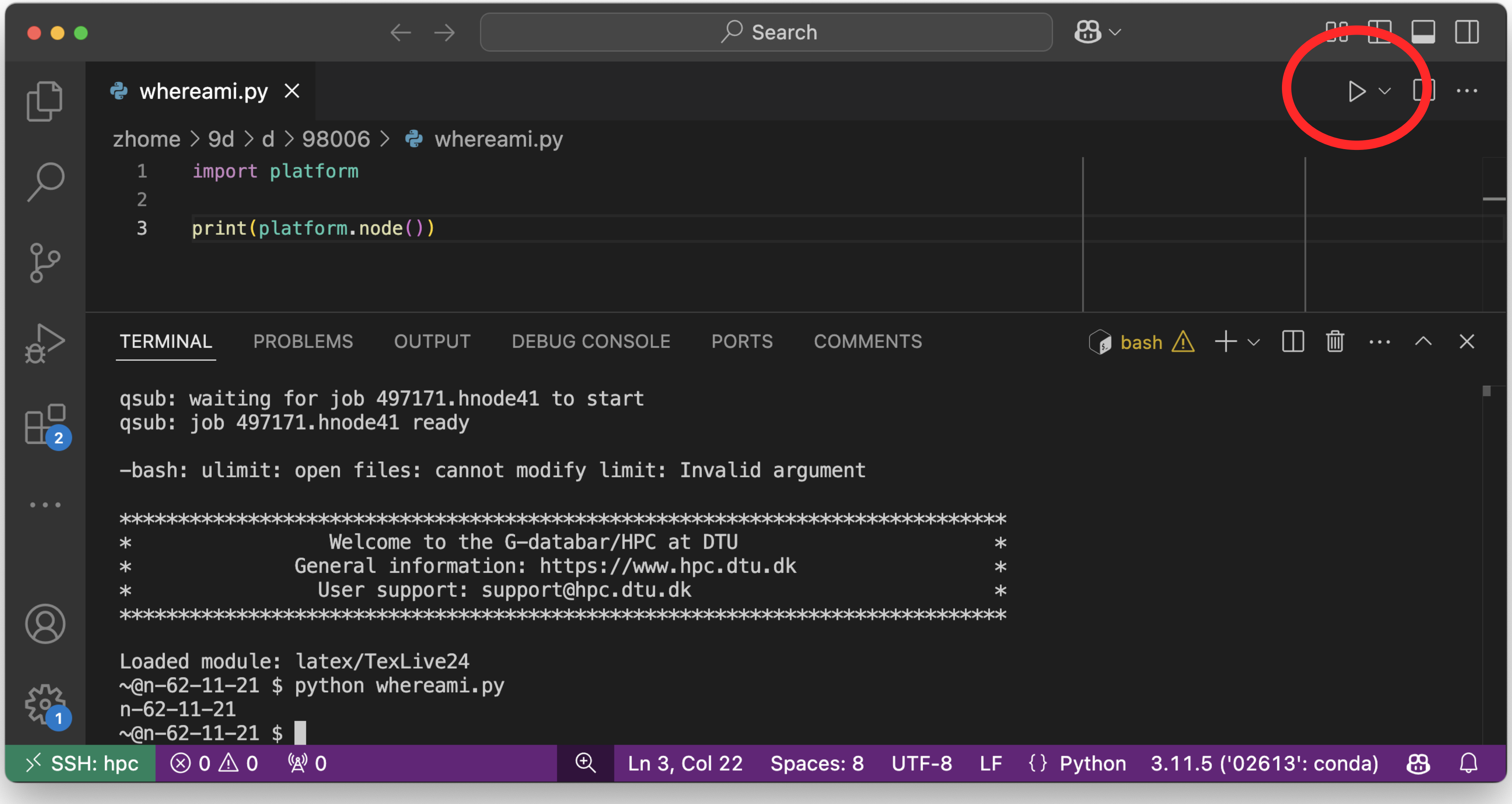
```
qsub: waiting for job 497171.hnode41 to start
qsub: job 497171.hnode41 ready

-bash: ulimit: open files: cannot modify limit: Invalid argument

******************************************************************************
*                 Welcome to the G-databar/HPC at DTU                        *
*              General information: https://www.hpc.dtu.dk                   *
*                User support: support@hpc.dtu.dk                            *
******************************************************************************

Loaded module: latex/TexLive24
~@n-62-11-21 $ python whereami.py
n-62-11-21
~@n-62-11-21 $
```

Login nodes and VS Code



```python
import platform

print(platform.node())
```

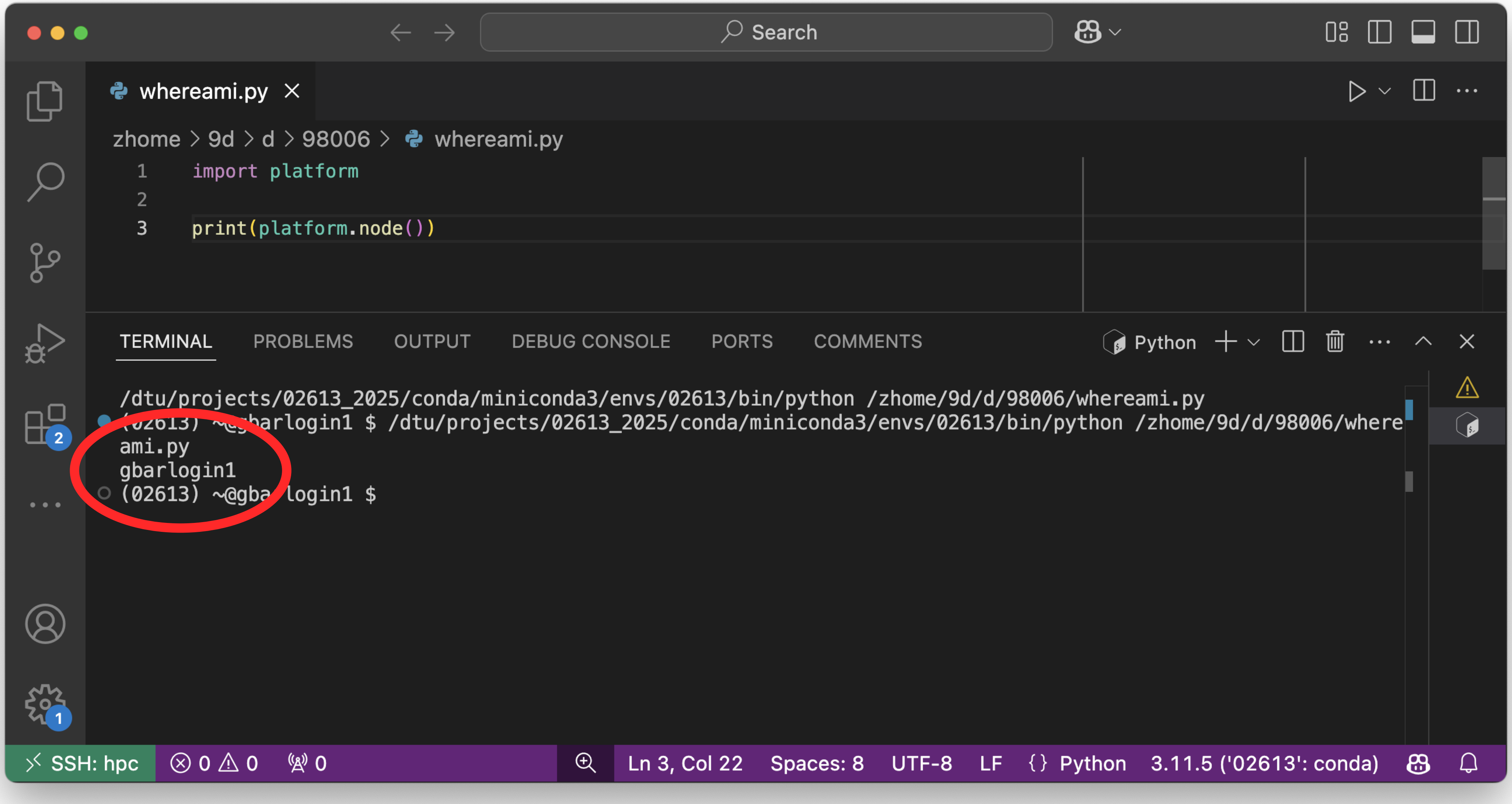
```
qsub: waiting for job 497171.hnode41 to start
qsub: job 497171.hnode41 ready

-bash: ulimit: open files: cannot modify limit: Invalid argument

****************************************************************************
*                 Welcome to the G-databar/HPC at DTU                      *
*              General information: https://www.hpc.dtu.dk                 *
*                User support: support@hpc.dtu.dk                          *
****************************************************************************

Loaded module: latex/TexLive24
~@n-62-11-21 $ python whereami.py
n-62-11-21
~@n-62-11-21 $
```

Login nodes and VS Code

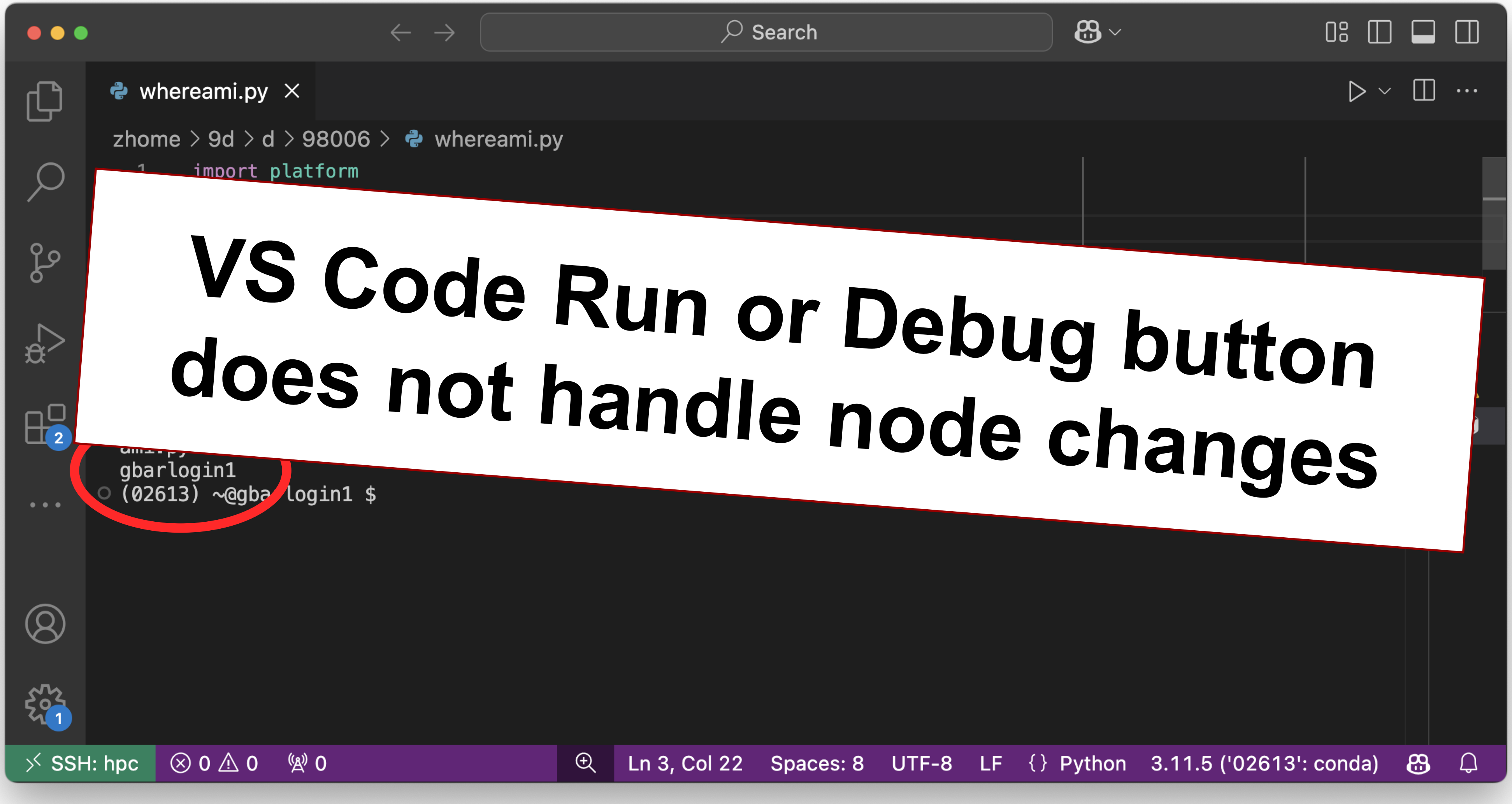


```python
import platform

print(platform.node())
```

```
/dtu/projects/02613_2025/conda/miniconda3/envs/02613/bin/python /zhome/9d/d/98006/whereami.py
(02613) ~@gbarlogin1 $ /dtu/projects/02613_2025/conda/miniconda3/envs/02613/bin/python /zhome/9d/d/98006/whereami.py
gbarlogin1
(02613) ~@gbarlogin1 $
```

Login nodes and VS Code

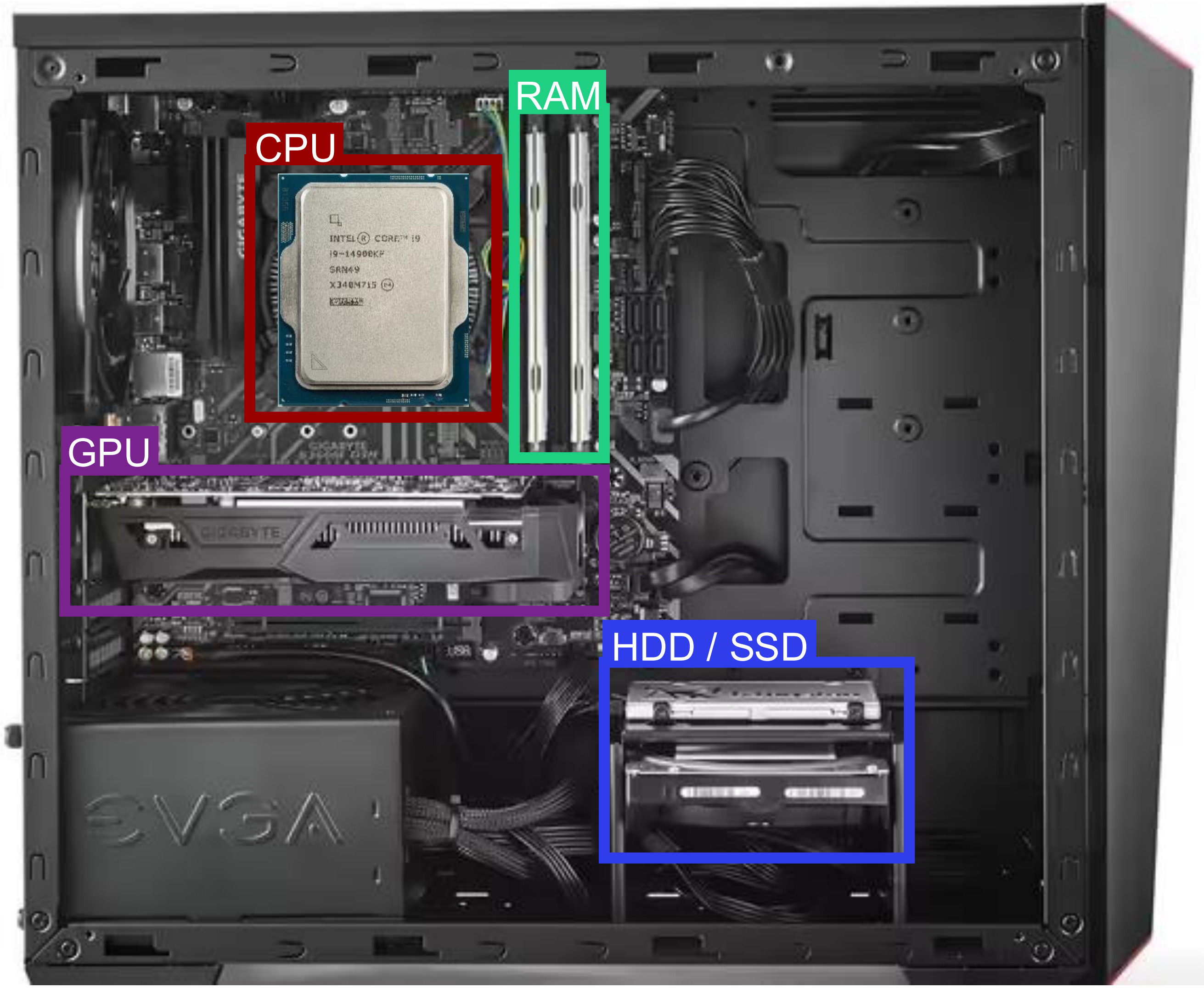


```
import platform
```

```
gbarlogin1
(02613) ~@gbarlogin1 $
```

VS Code Run or Debug button does not handle node changes

Computer Terminology



CPU

Performs computations

HDD / SSD

Permanent file storage

RAM

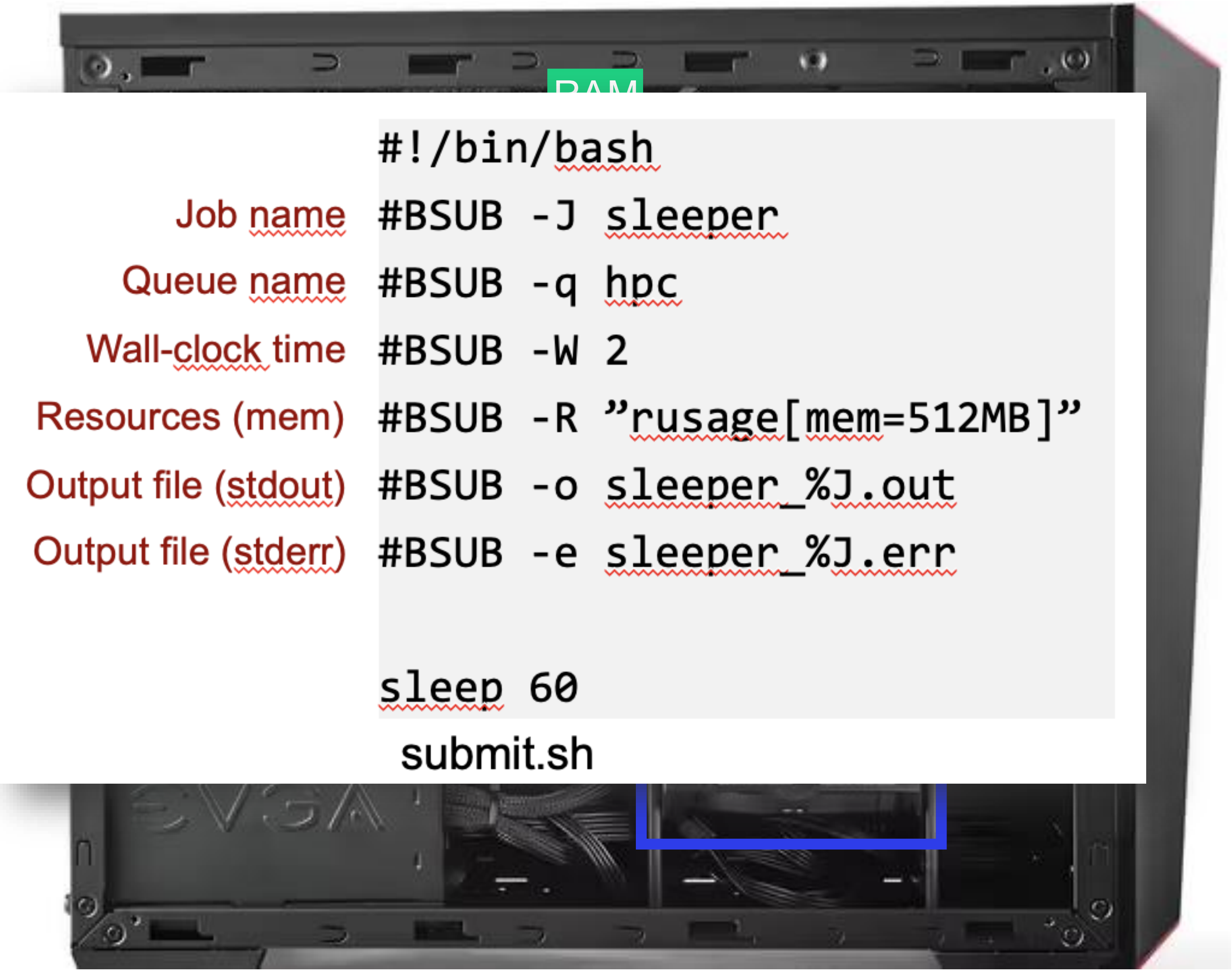
Temporary working memory

Where the CPU “works from”

GPU

Stay tuned for week 9 ;)

Computer Terminology



```
#!/bin/bash
#BSUB -J sleeper
#BSUB -q hpc
#BSUB -W 2
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o sleeper_%J.out
#BSUB -e sleeper_%J.err


sleep 60
```

submit.sh

CPU

Performs computations

HDD / SSD

Permanent file storage

RAM

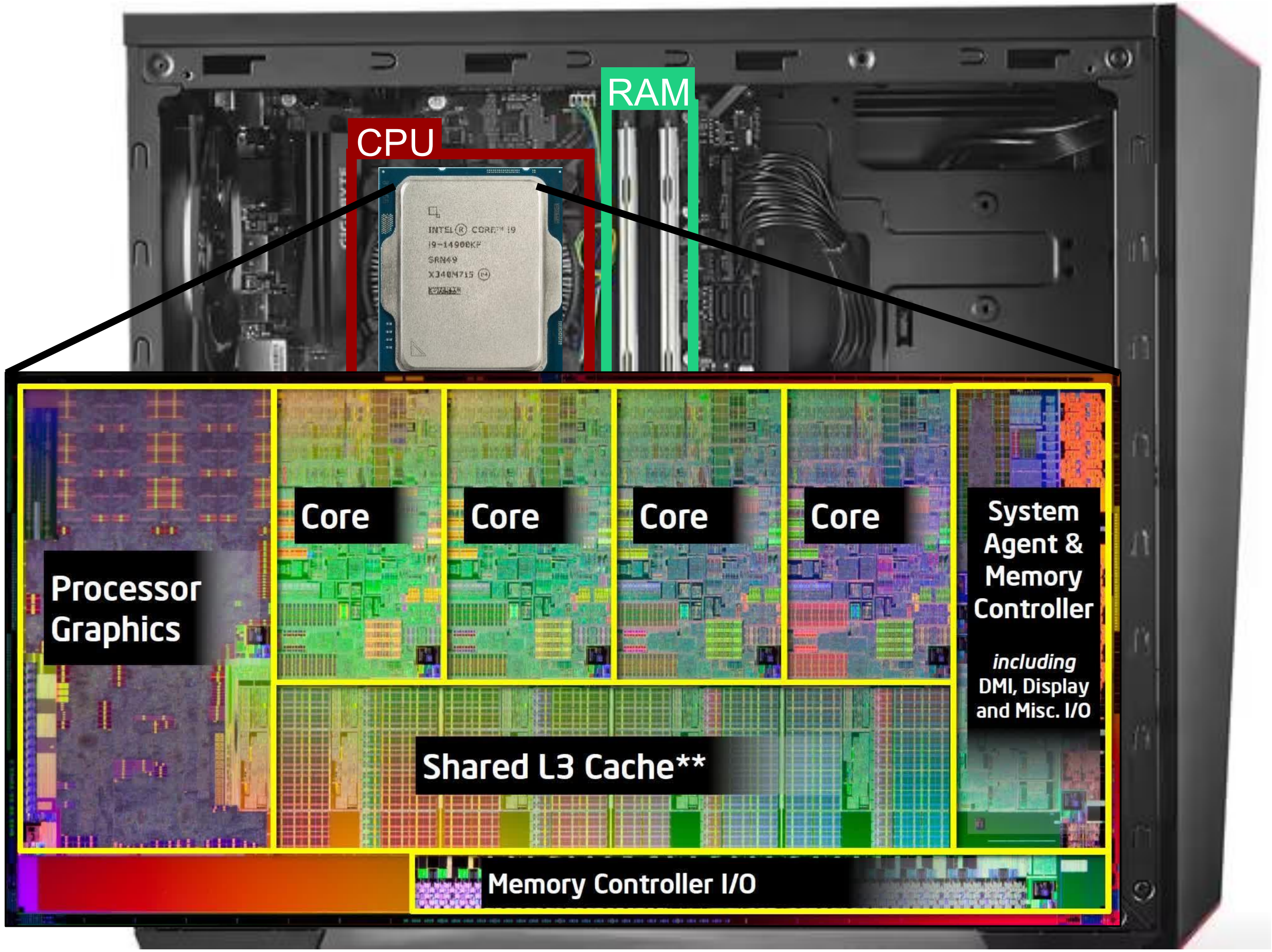
Temporary working memory

Where the CPU “works from”

GPU

Stay tuned for week 9 ;)

Computer Terminology



CPU

Performs computations

Contains multiple **cores**

Core can work independently

HDD / SSD

Permanent file storage

RAM

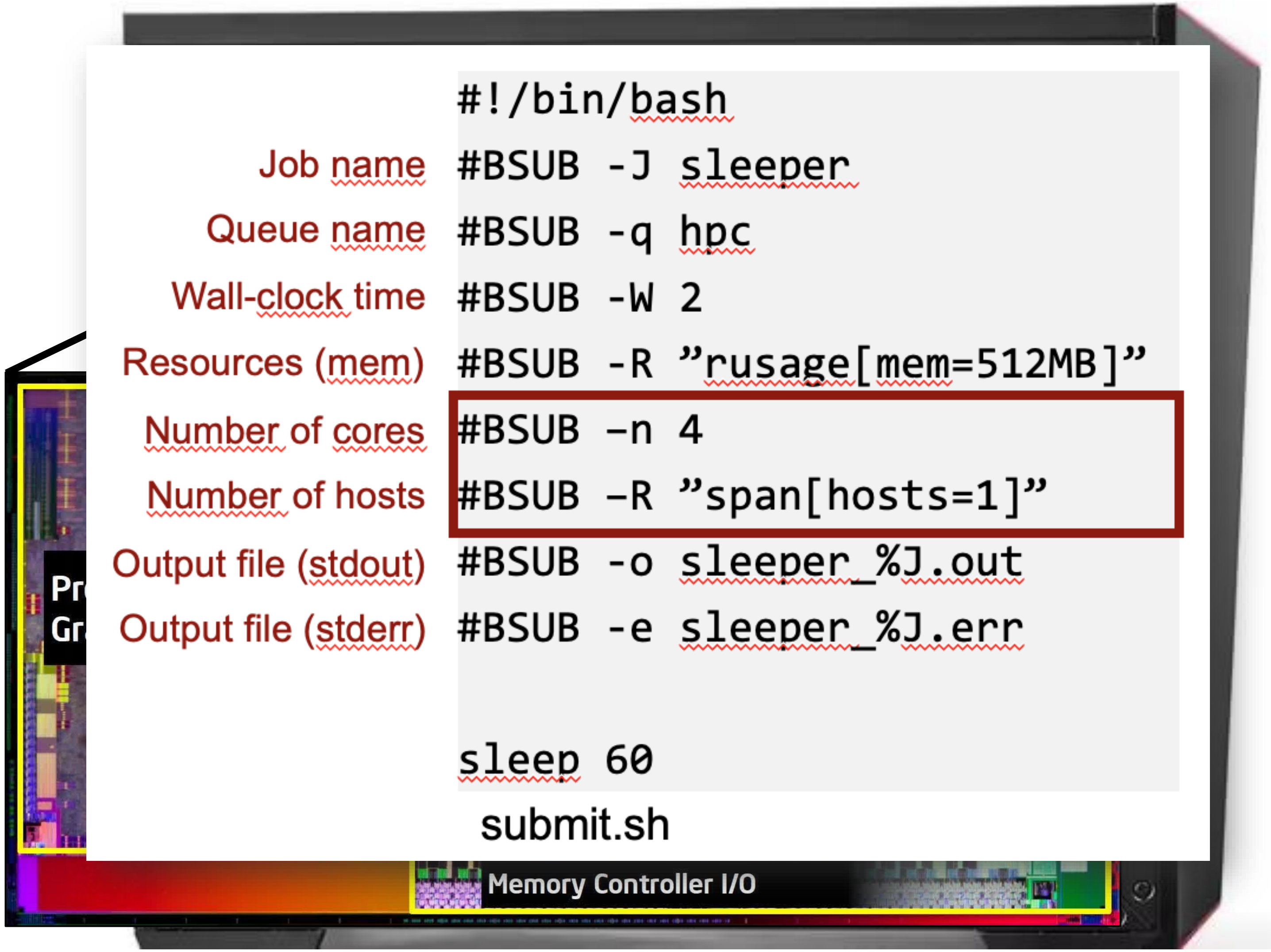
Temporary working memory

Where the CPU "works from"

GPU

Stay tuned for week 9 ;)

Computer Terminology



	`#!/bin/bash`
Job name	`#BSUB -J sleeper`
Queue name	`#BSUB -q hpc`
Wall-clock time	`#BSUB -W 2`
Resources (mem)	`#BSUB -R ”rusage[mem=512MB]”`
Number of cores	`#BSUB -n 4`
Number of hosts	`#BSUB -R ”span[hosts=1]”`
Output file (stdout)	`#BSUB -o sleeper_%J.out`
Output file (stderr)	`#BSUB -e sleeper_%J.err`
	`sleep 60`

submit.sh

CPU

Performs computations

Contains multiple **cores**

Core can work independently

HDD / SSD

Permanent file storage

RAM

Temporary working memory

Where the CPU “works from”

GPU

Stay tuned for week 9 ;)

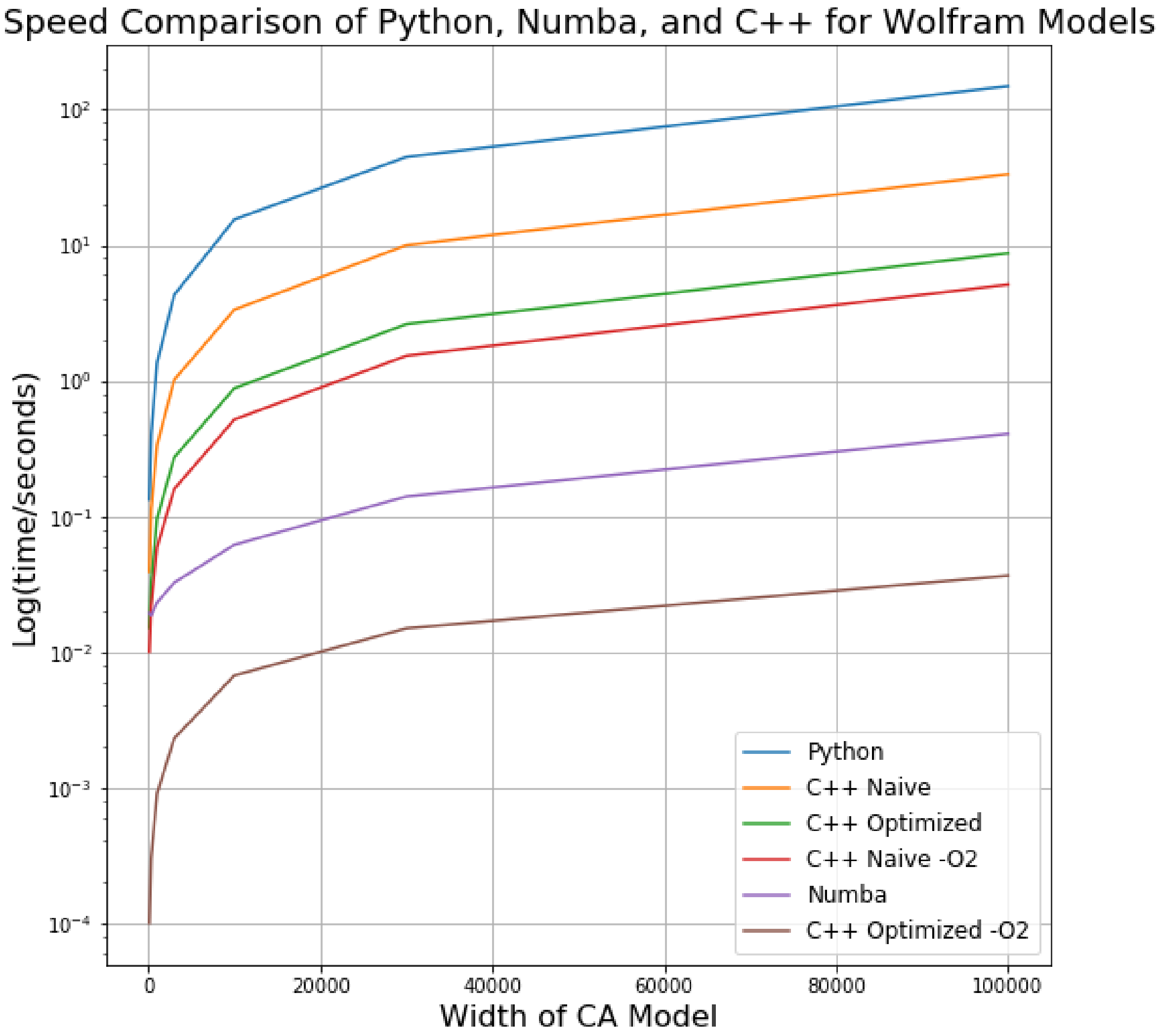
Today

1. Why Python?

2. Python programs & command line args

3. Python as batch jobs

Indeed, why Python?



Indeed, why Python?

		fannkuch-redux	n-body	spectral-norm	mandelbrot	pidigits	regex-redux	fasta
C	Time	2.10	2.08	0.40	1.30	0.81	0.80	0.78
	Memory	11,236	11,392	11,392	32,992	11,392	152,148	11,284
Python	Time	285.20	383.12	78.36	155.28	1.25	1.38	49.99
	Memory	14,264	11,096	15,704	14,244	13,564	168,144	11,088

https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gcc.html

Indeed, why Python?

GIL

Global **I**nterpreter **L**ock

GIL: Global Interpreter Lock

```
<< Main >>
a = 2

<< Thread 1 >>
a = a + 2

<< Thread 2 >>
a = a + 3

<< Main >>
wait_for_threads()
print(a)
```

What does this print?

Example from: https://python.land/python-concurrency/the-python-gil

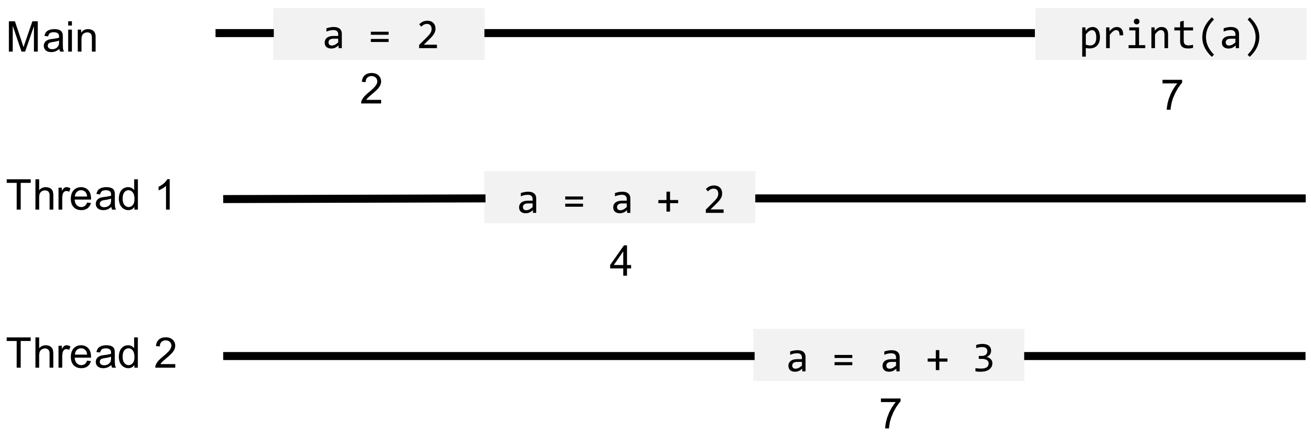
GIL: Global Interpreter Lock

```
<< Main >>
a = 2

<< Thread 1 >>
a = a + 2

<< Thread 2 >>
a = a + 3

<< Main >>
wait_for_threads()
print(a)
```



What does this print?

Example from: https://python.land/python-concurrency/the-python-gil

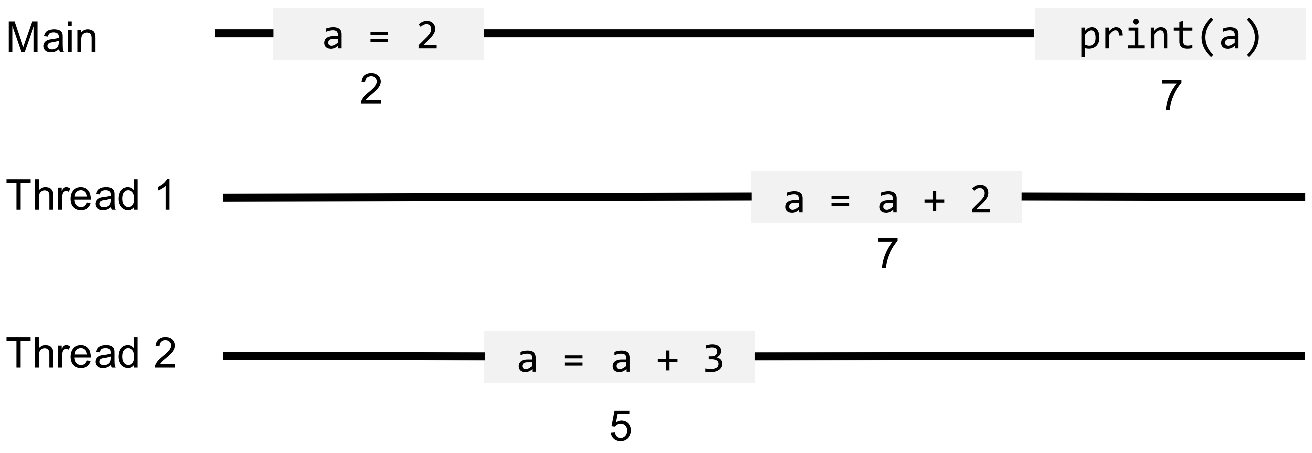
GIL: Global Interpreter Lock

```
<< Main >>
a = 2

<< Thread 1 >>
a = a + 2

<< Thread 2 >>
a = a + 3

<< Main >>
wait_for_threads()
print(a)
```



What does this print?

Example from: https://python.land/python-concurrency/the-python-gil

GIL: Global Interpreter Lock

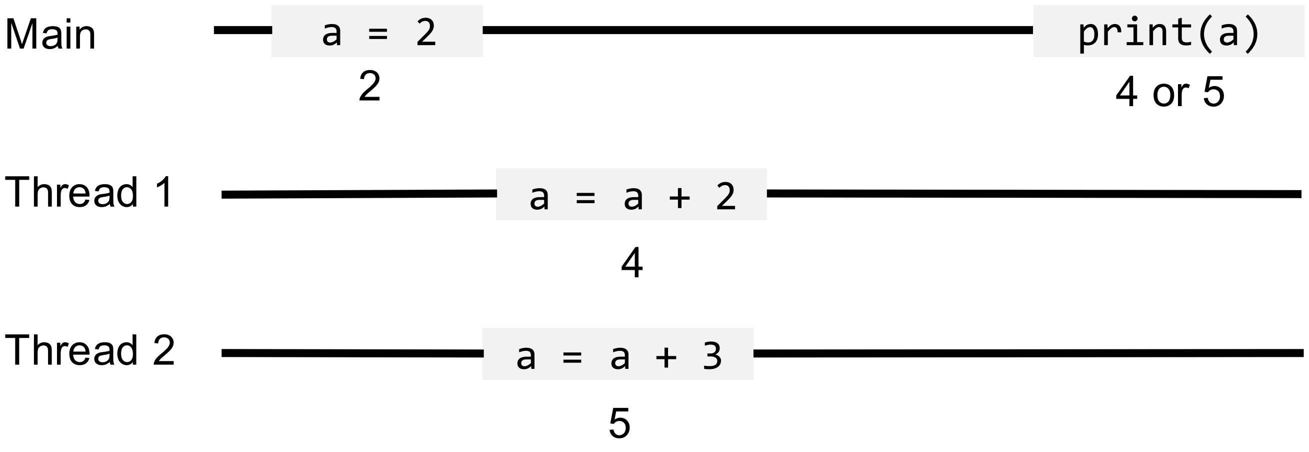
```
<< Main >>
a = 2

<< Thread 1 >>
a = a + 2

<< Thread 2 >>
a = a + 3

<< Main >>
wait_for_threads()
print(a)
```

What does this print?



Race condition!

Example from: https://python.land/python-concurrency/the-python-gil

GIL: Global Interpreter Lock

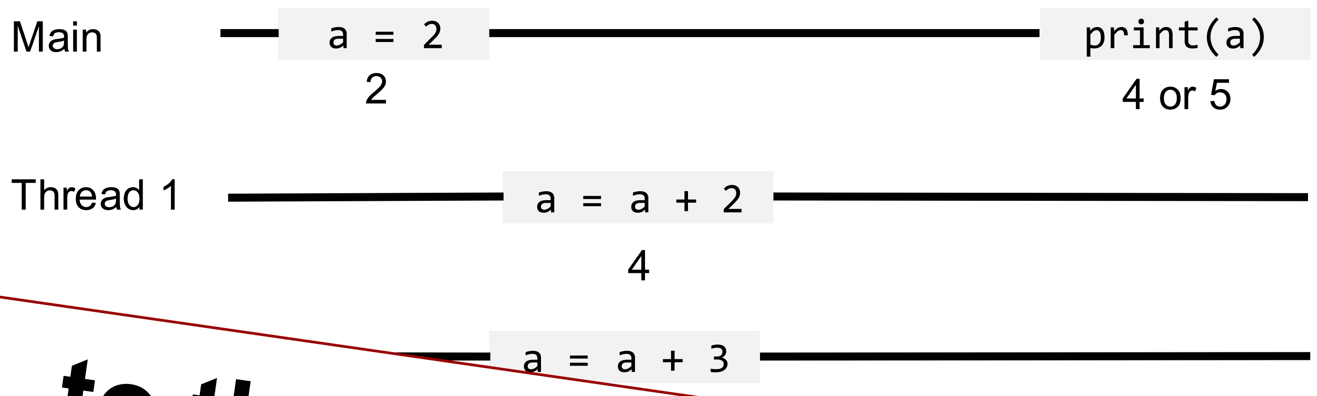
```
<< Main >>
a = 2

<< Thread 1 >>
a = a + 2

<< Thread 2 >>
a = a + 3

<< Main >>
wait_for_threads()
print(a)
```

What does this print?



Race conditi

GIL to the rescue!

Example from: https://python.land/python-concurrency/the-python-gil

GIL: Global Interpreter Lock

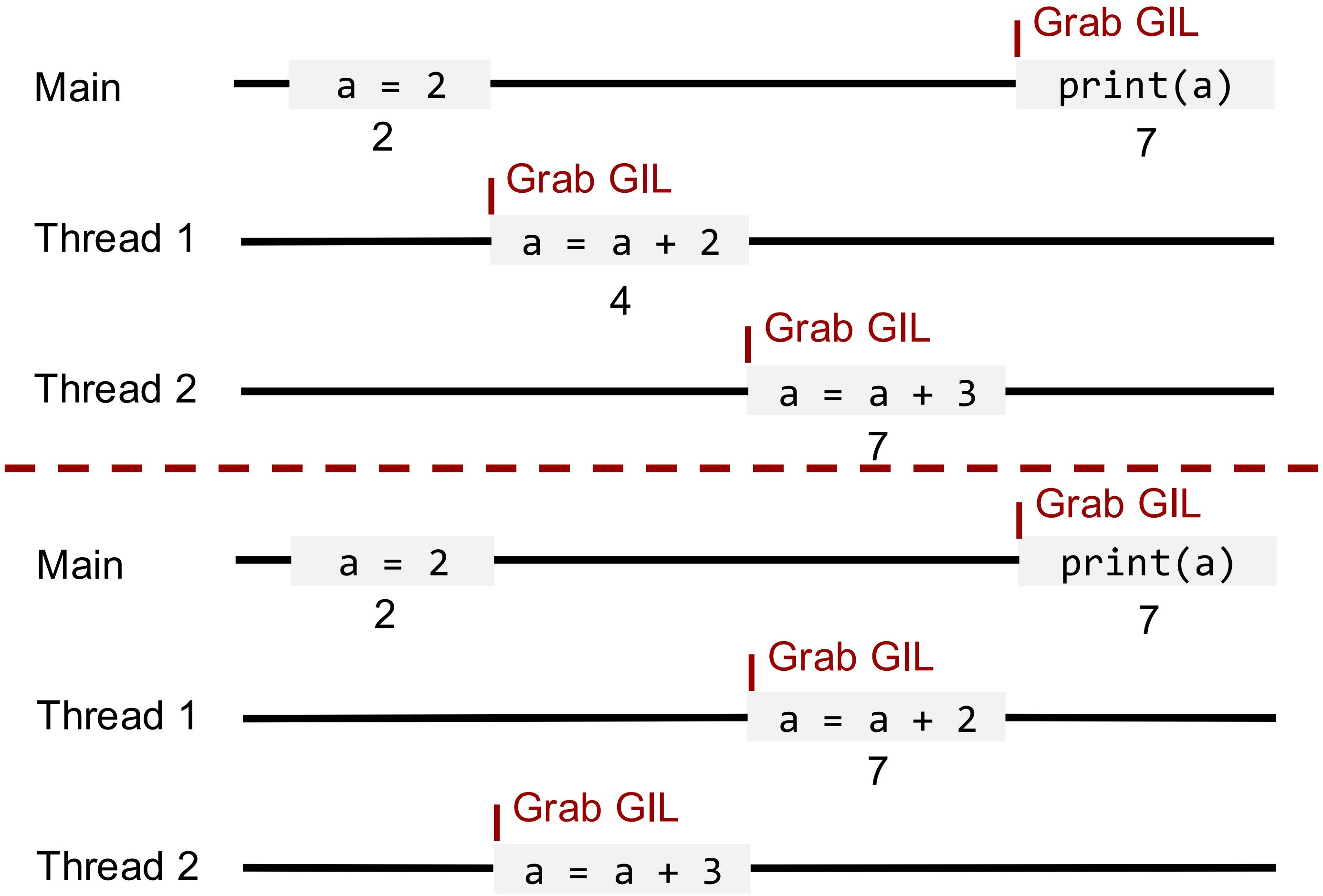
```
<< Main >>
a = 2

<< Thread 1 >>
a = a + 2

<< Thread 2 >>
a = a + 3

<< Main >>
wait_for_threads()
print(a)
```

What does this print?



Main: a = 2 → 2; Grab GIL print(a) → 7

Thread 1: Grab GIL a = a + 2 → 4

Thread 2: Grab GIL a = a + 3 → 7

Main: a = 2 → 2; Grab GIL print(a) → 7

Thread 1: Grab GIL a = a + 2 → 7

Thread 2: Grab GIL a = a + 3

Example from: https://python.land/python-concurrency/the-python-gil

GIL: Global Interpreter Lock

Pro:

- Ensures no race conditions – convenient!
- Some ops release GIL (e.g., I/O, NumPy)

Con:

- Compute heavy multi-threaded = impossible
- Finicky to code around

Instead:

- Use multi-processing
  - more heavy duty
  - harder to share data
- Escape to other languages

GIL: Global Interpreter Lock

Pro:

- Ensures no race conditions – convenient!
- So GIL (e.g., I/O, NumPy)

Con:

- Compute heavy multi-threaded = impossible
- Finicky to code around

We’ll get back to this in weeks 6 + 7

Instead:

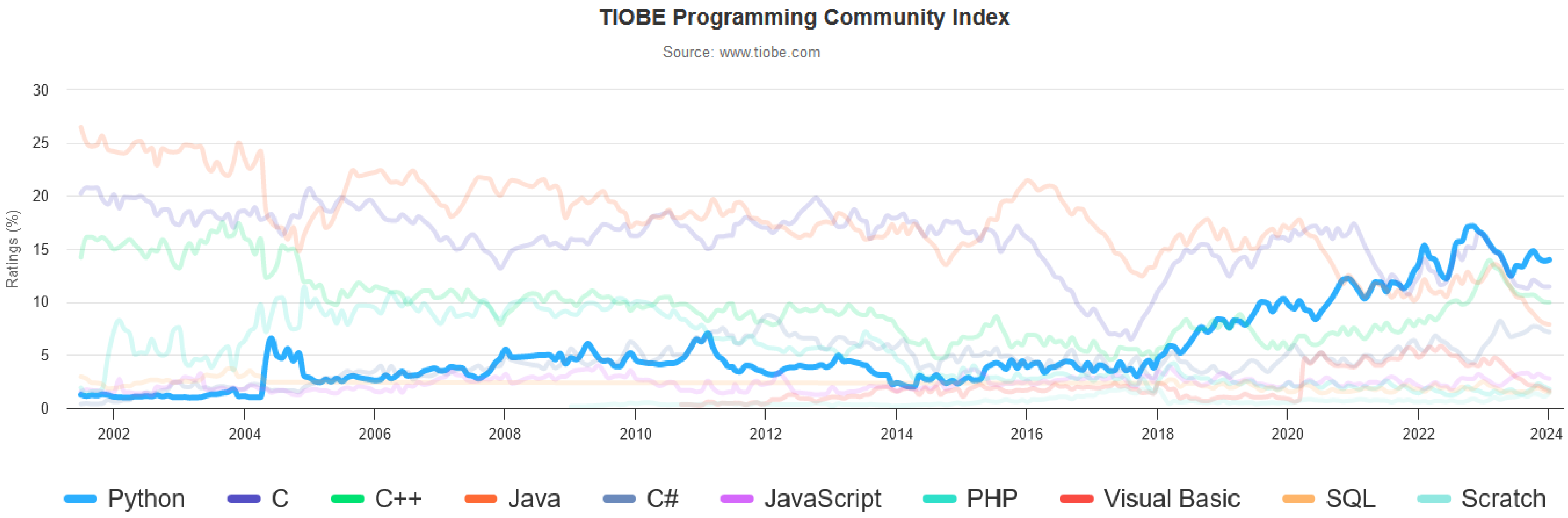
- Use multi-processing
  - more heavy duty
  - harder to share data
- Escape to other languages

Indeed, why Python?

So, Python is

1. Slow
2. Not great for parallel

Python is popular!



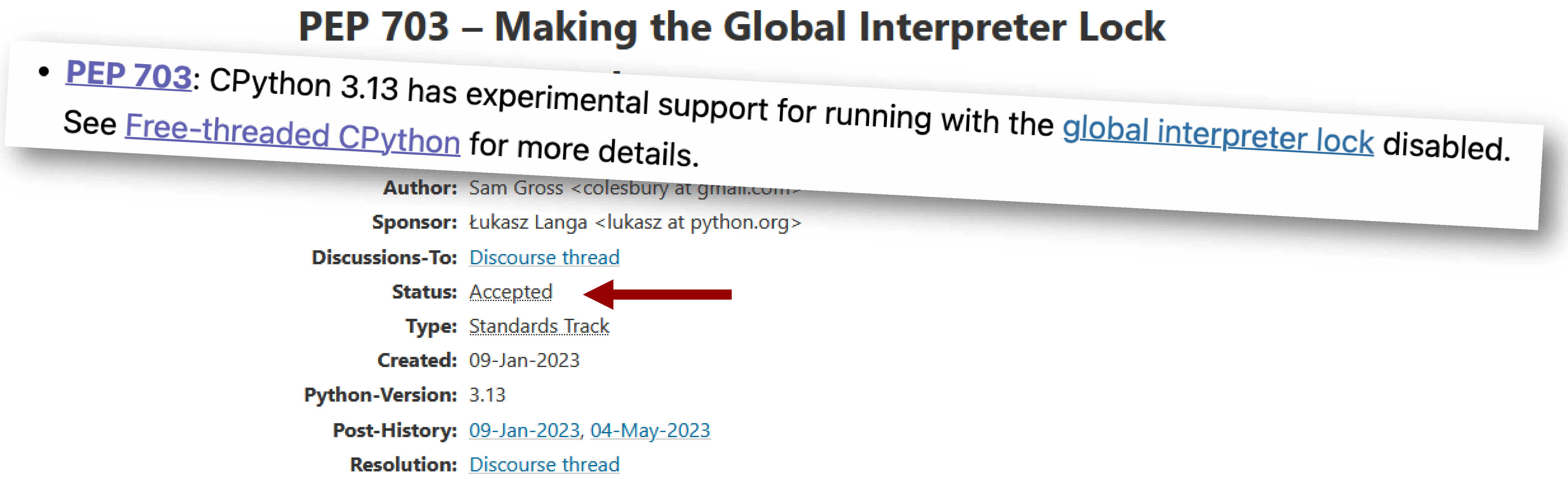
With a great ecosystem

And easy to use

speed vs agility

…and the GIL is going away too

peps.python.org/pep-0703/



- PEP 703: CPython 3.13 has experimental support for running with the global interpreter lock disabled. See Free-threaded CPython for more details.

PEP 703 – Making the Global Interpreter Lock

Author:	Sam Gross <colesbury at gmail.com>
Sponsor:	Łukasz Langa <lukasz at python.org>
Discussions-To:	Discourse thread
Status:	Accepted
Type:	Standards Track
Created:	09-Jan-2023
Python-Version:	3.13
Post-History:	09-Jan-2023, 04-May-2023
Resolution:	Discourse thread

Join the Vevox session

Go to **vevox.app**

Enter the session ID: **136-979-360**

Or scan the QR code

Python is slow because...

it is compiled to machine code

##.##%

it is interpreted

##.##%

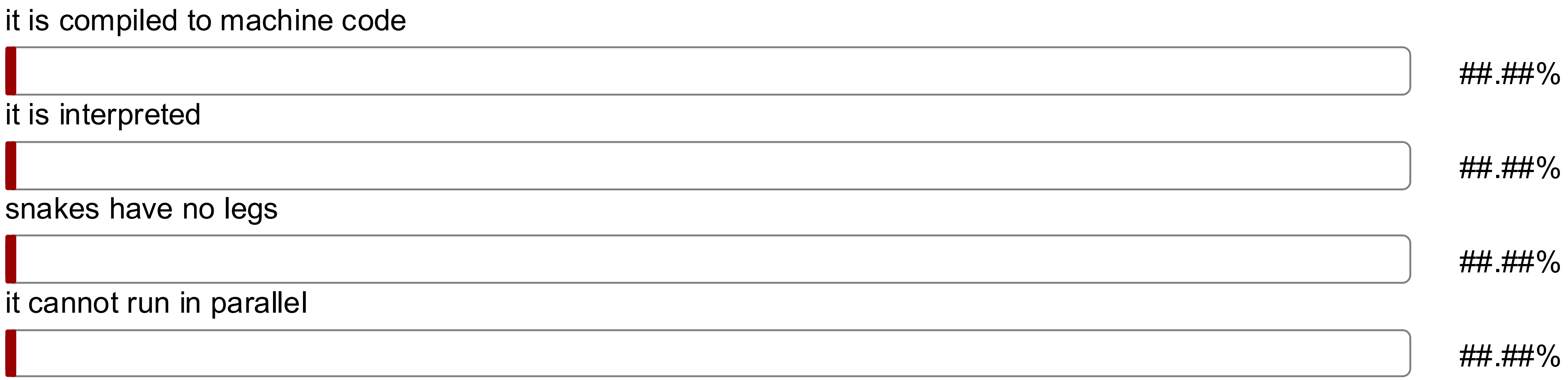
snakes have no legs

##.##%

it cannot run in parallel

##.##%

Python is slow because...



##/##

Join at: **vevox.app**

ID: **XXX-XXX-XXX**

Question slide

What does GIL stand for?

Global Interpreter Lock

##.##%

Good Interpreted Language

##.##%

Global International Language

##.##%

Grand Interpreter Lie

##.##%

What does GIL stand for?

Global Interpreter Lock ##.##%

Good Interpreted Language ##.##%

Global International Language ##.##%

Grand Interpreter Lie ##.##%

##/##

Join at: **vevox.app**

ID: **XXX-XXX-XXX**

Question slide

What is the problem with the GIL?

It's hard to pronounce

##.##%

It makes all code slower

##.##%

It must be manually turned on/off

##.##%

It prevents efficient multi-threading

##.##%

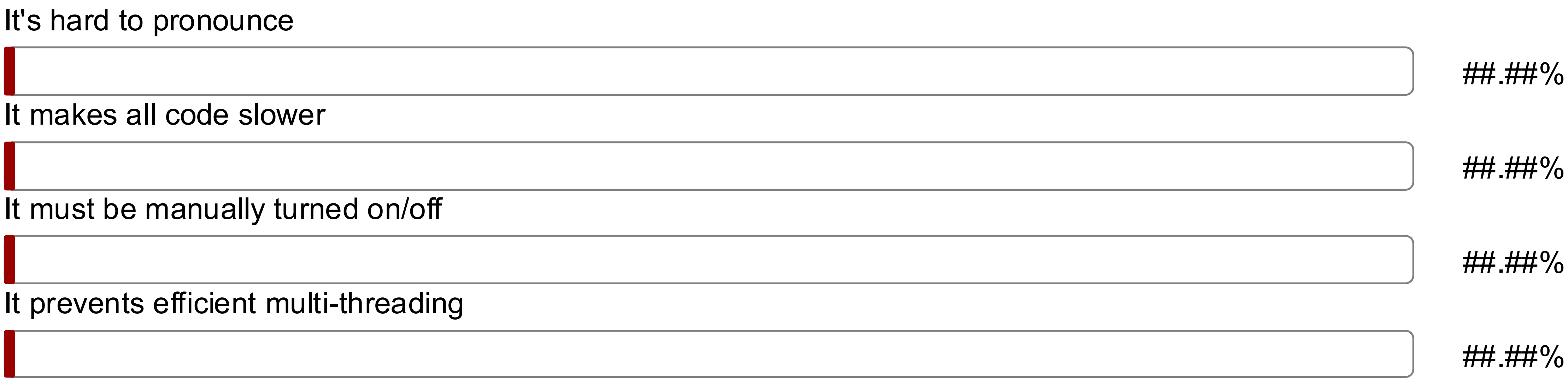
##/##

Join at: **vevox.app**

ID: **XXX-XXX-XXX**

Results slide

What is the problem with the GIL?



It's hard to pronounce

##.##%

It makes all code slower

##.##%

It must be manually turned on/off

##.##%

It prevents efficient multi-threading

##.##%

##/##

Join at: **vevox.app**

ID: **XXX-XXX-XXX**

Question slide

Why might we still use Python despite being slow?

Snakes are cool! ##.##%

Ease of use increases productivity ##.##%

Nothing else can run on the HPC system ##.##%

Slow code means more breaks for me ##.##%

Why might we still use Python despite being slow?

Snakes are cool!

##.##%

Ease of use increases productivity

##.##%

Nothing else can run on the HPC system

##.##%

Slow code means more breaks for me

##.##%

Python programs a.k.a. scripts

Python programs

```
print(”Hello World”)
```

hello.py

```
$
```

Python programs

```
print(”Hello World”)
```

hello.py

```
$ python hello.py
Hello World
$
```

Python programs

```
print(”Hello World”)
```

hello.py

```
$ source /dtu/projects/02613_2025/...
$ conda activate 02613
(02613) $ python hello.py
Hello World
(02613) $
```

Python programs

```
print(”Hello World”)
```

hello.py

```
$ python hello.py
Hello World
$
```

Python programs

```
name = ”02613”
print(f”Hello {name}”)
```

hello.py

```
$ python hello.py
Hello 02613
$
```

Python programs

```
name = ??
print(f”Hello {name}”)
```

hello.py

```
$ python hello.py 02613
Hello 02613
$
```

Python programs

```python
def compute_from_data(file_path):
    data = load_data(file_path)
    # Compute...
    ...
    return result



result = compute_from_data(”path/to/data.txt”)
print(result)
```

program.py

```
$
```

Python programs

```python
def compute_from_data(file_path):
    data = load_data(file_path)
    # Compute...
    ...
    return result



result = compute_from_data(”path/to/data.txt”)
print(result)
```

program.py

```
$ python program.py
42
$
```

Python programs

```python
def compute_from_data(file_path):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = ???
result = compute_from_data(file_path)
print(result)
```

program.py

```
$ python program.py path/to/data.txt
42
$
```

Python programs

```python
import sys

print(sys.argv)
```

List of command line arguments:

0: Script file name

1: First argument

2: Second argument

…

```
$
```

program.py

Python programs

```python
import sys

print(sys.argv)
```

List of command line arguments:

0: Script file name

1: First argument

2: Second argument

…

```
$ python program.py path/to/data.txt
['program.py', 'path/to/data.txt']
$
```

program.py

Python programs

```python
import sys

print(sys.argv)
```

List of command line arguments:

0: Script file name

1: First argument

2: Second argument

…

program.py

```
$ python program.py path/to/data.txt
['program.py', 'path/to/data.txt']
$ python program.py 1st 2nd 3rd
['program.py', '1st', '2nd', '3rd']
$
```

Python programs

```python
import sys

print(sys.argv)
```

List of command line arguments:
0: Script file name
1: First argument
2: Second argument
…

program.py

```
$ python program.py path/to/data.txt
['program.py', 'path/to/data.txt']
$ python program.py 1st 2nd 3rd
['program.py', '1st', '2nd', '3rd']
$ python program.py
```

Python programs

```python
import sys

print(sys.argv)
```

List of command line arguments:

0: Script file name

1: First argument

2: Second argument

…

program.py

```
$ python program.py path/to/data.txt
['program.py', 'path/to/data.txt']
$ python program.py 1st 2nd 3rd
['program.py', '1st', '2nd', '3rd']
$ python program.py
['program.py']
$
```

Python programs

```python
def compute_from_data(file_path):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = ???
result = compute_from_data(file_path)
print(result)
```

program.py

```
$ python program.py path/to/data.txt
42
$
```

Python programs

```python
import sys

def compute_from_data(file_path):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = sys.argv[1]
result = compute_from_data(file_path)
print(result)
```

program.py

```
$ python program.py path/to/data.txt
42
$
```

Python programs

```python
import sys

def compute_from_data(file_path, num_iters):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = sys.argv[1]
num_iters = sys.argv[2]
result = compute_from_data(file_path, num_iters)
print(result)
```

program.py

```
$ python program.py path/to/data.txt 100
```

Python programs

```python
import sys

def compute_from_data(file_path, num_iters):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = sys.argv[1]
num_iters = sys.argv[2]
result = compute_from_data(file_path, num_iters)
print(result)
```

program.py

```
$ python program.py path/to/data.txt 100
Traceback (most recent call last):
…
TypeError: 'str' object cannot be
interpreted as an integer
$
```

Python programs

```python
import sys

def compute_from_data(file_path, num_iters, a):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = sys.argv[1]
num_iters = int(sys.argv[2])  # Cast to int
result = compute_from_data(file_path, num_iters, a)
print(result)
```

program.py

```
$ python program.py path/to/data.txt 100
42
$
```

Python programs

```python
import sys

def compute_from_data(file_path, num_iters, a):
    data = load_data(file_path)
    # Compute...
    ...
    return result


file_path = sys.argv[1]
num_iters = int(sys.argv[2])  # Cast to int
a = float(sys.argv[3])        # Cast to float
result = compute_from_data(file_path, num_iters, a)
print(result)
```

program.py

```
$ python program.py path/to/data.txt 100 0.2
42
$
```

Join the Vevox session

Go to **vevox.app**

Enter the session ID: **136-979-360**

Or scan the QR code

##/## Join at: **vevox.app** ID: **XXX-XXX-XXX** Question slide

How do you access command line arguments?

A

```
import sys
sys.args
```

B

```
import cli
cli.args
```

C

```
import sys
sys.argv
```

D

```
argv = input()
```

##/## Join at: **vevox.app** ID: **XXX-XXX-XXX** Results slide

How do you access command line arguments?

A

```
import sys
sys.args
```

B

```
import cli
cli.args
```

C

```
import sys
sys.argv
```

D

```
argv = input()
```

What will this print?

A

##.##%

B

##.##%

C

##.##%

prog.py:

```
import sys
print(sys.args)
```

```
$ python prog.py hello
```

A

```
[‘python’, ‘prog.py’,
‘hello’]
```

B

```
[‘prog.py’, ‘hello’]
```

C

```
[‘hello’]
```

What will this print?

A

##.##%

B

##.##%

C

##.##%

prog.py:

```
import sys
print(sys.args)
```

```
$ python prog.py hello
```

A

```
[‘python’, ‘prog.py’,
‘hello’]
```

B

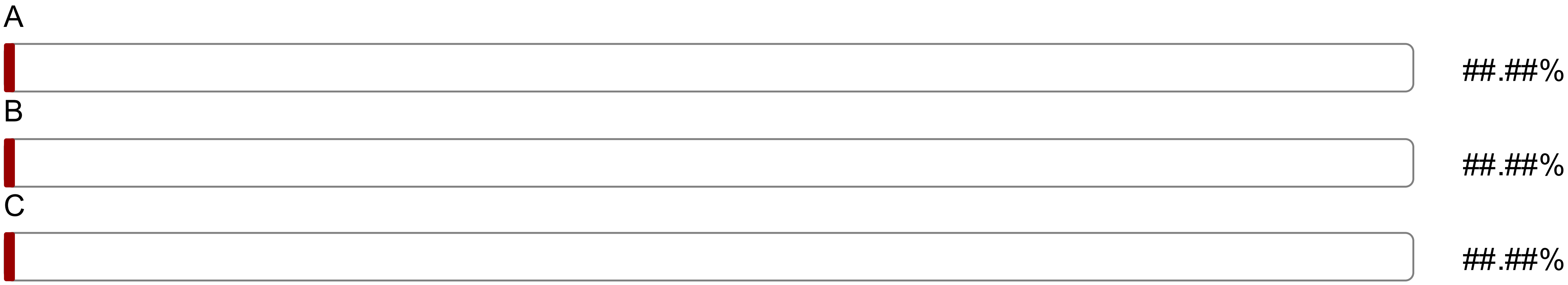
```
[‘prog.py’, ‘hello’]
```

C

```
[‘hello’]
```

##/## Join at: **vevox.app** ID: **XXX-XXX-XXX** Question slide

What will this print?



prog.py:

```
import sys
print(sys.args)
```

```
$ python prog.py data/my data.d
```

A

```
[‘prog.py’, ‘data/my data.d’]
```

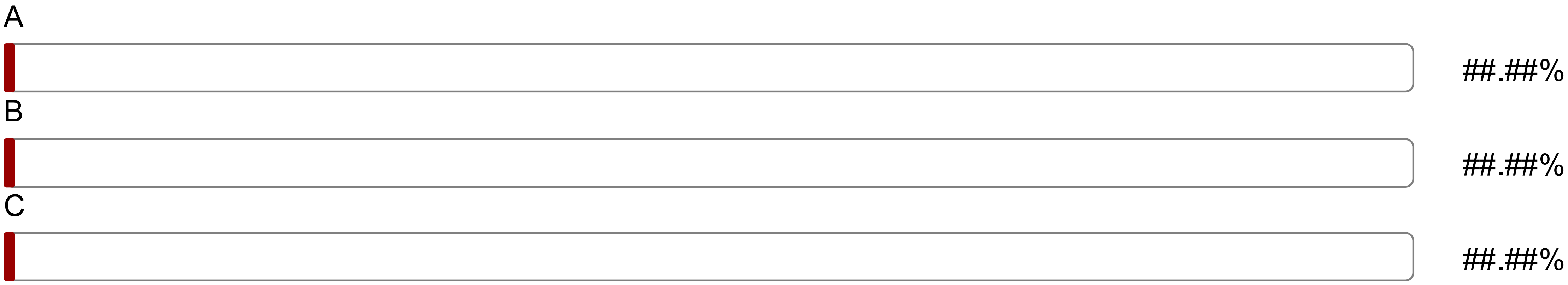
B

```
[‘prog.py’, ‘data/my’, ‘data.d’]
```

C

```
[‘prog.py’, ‘data’, ‘my data.d’]
```

What will this print?



prog.py:

```
import sys
print(sys.args)
```

```
$ python prog.py data/my data.d
```

A

```
[‘prog.py’, ‘data/my data.d’]
```

B

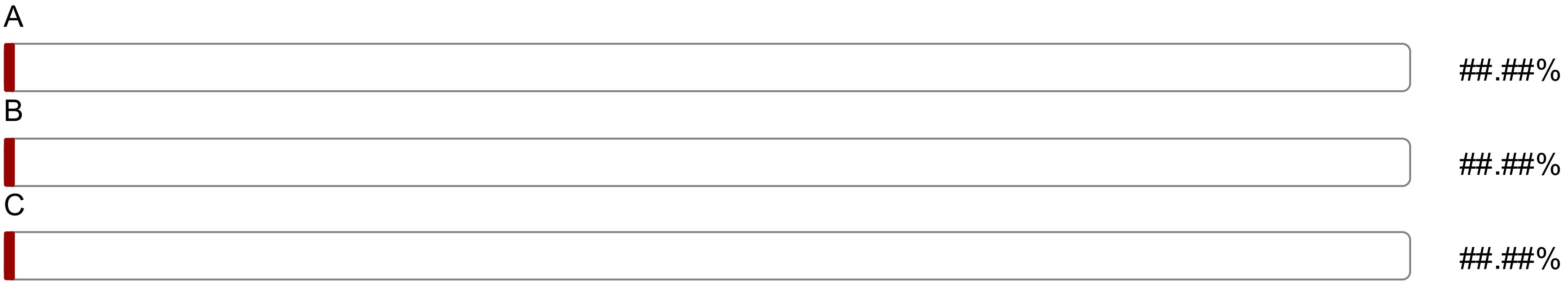
```
[‘prog.py’, ‘data/my’, ‘data.d’]
```

C

```
[‘prog.py’, ‘data’, ‘my data.d’]
```

##/## Join at: **vevox.app** ID: **XXX-XXX-XXX** Question slide

What will this print?



prog.py:

```
import sys
print(sys.args)
```

```
$ python prog.py x 1 2.0
```

A

```
[‘prog.py’, ‘x’, 1, 2.0]
```

B

```
[‘prog.py’, ‘x’, ‘12.0’]
```

C

```
[‘prog.py’, ‘x’, ‘1’, ‘2.0’]
```

##/##

Join at: **vevox.app**

ID: **XXX-XXX-XXX**

Results slide

What will this print?

A

##.##%

B

##.##%

C

##.##%

prog.py:

```
import sys
print(sys.args)
```

```
$ python prog.py x 1 2.0
```

A

```
[‘prog.py’, ‘x’, 1, 2.0]
```

B

```
[‘prog.py’, ‘x’, ‘12.0’]
```

C

```
[‘prog.py’, ‘x’, ‘1’, ‘2.0’]
```

Python in batch jobs

Python in batch jobs

Resources

```
#!/bin/bash
#BSUB -J sleeper
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o sleeper_%J.out
#BSUB -e sleeper_%J.err
```

Command

```
sleep 60
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs

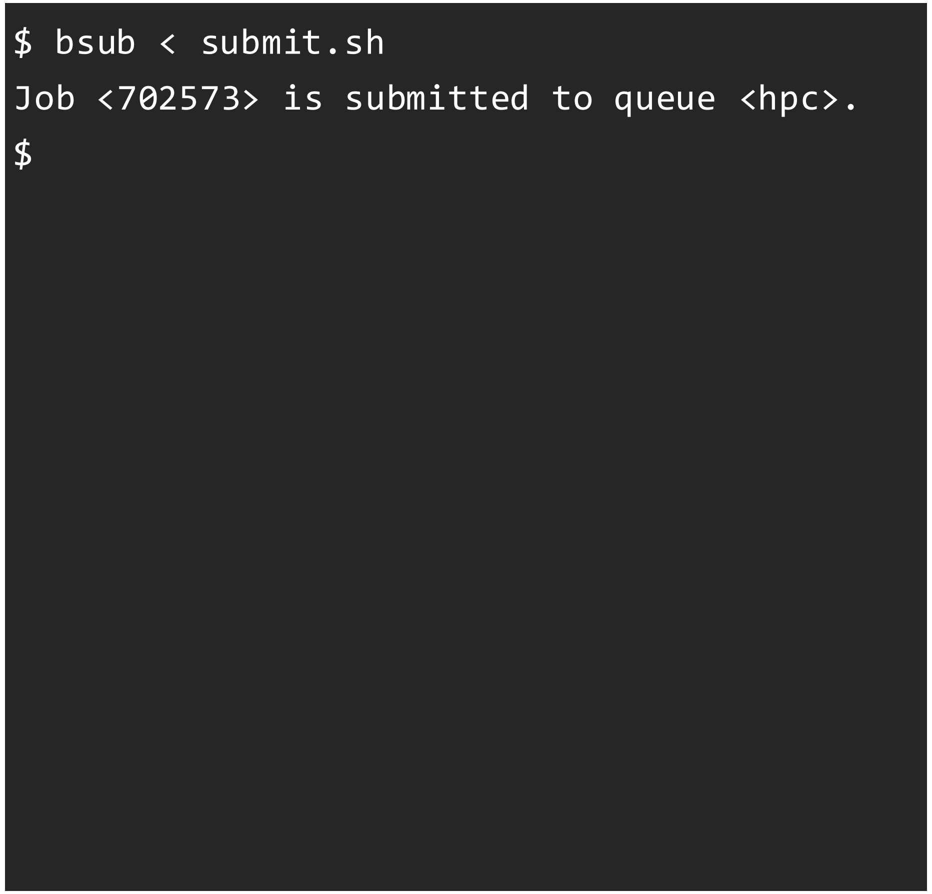
Resources

Initialize

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err

# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

submit.sh



```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs

```
#!/bin/bash
```

Resources

```
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err

# Initialize Python environment
```

Initialize

```
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613

# Run Python script
```

Run

```
python program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
```

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Your job script is read at <u>submission</u>, but…
Your Python script is read when the job <u>starts</u>!

Run

```
# Run Python script
python program.py
```

submit.sh

Python in batch jobs

Resources

Initialize

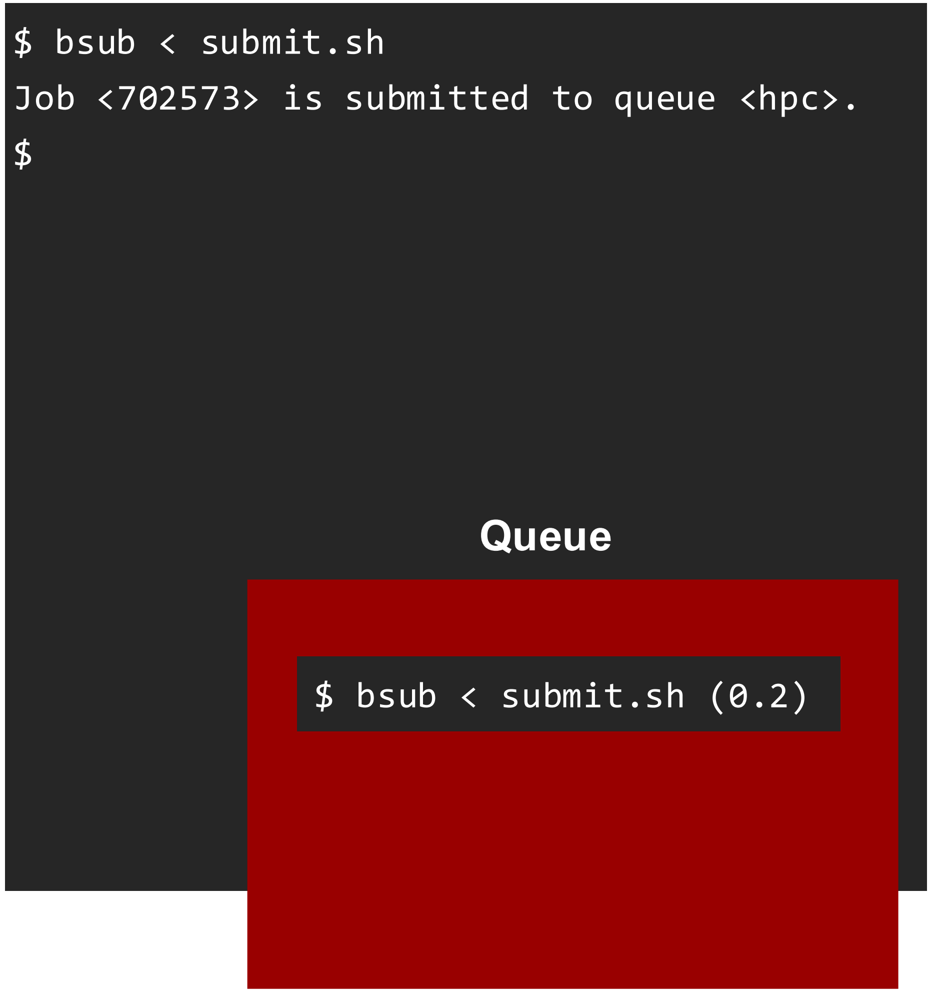
Run

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err

# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613

# Run Python script
python program.py path/to/data.txt 0.2
```

submit.sh



```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

```
$ bsub < submit.sh (0.2)
```

Python in batch jobs

```
#!/bin/bash
```

Resources

```
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

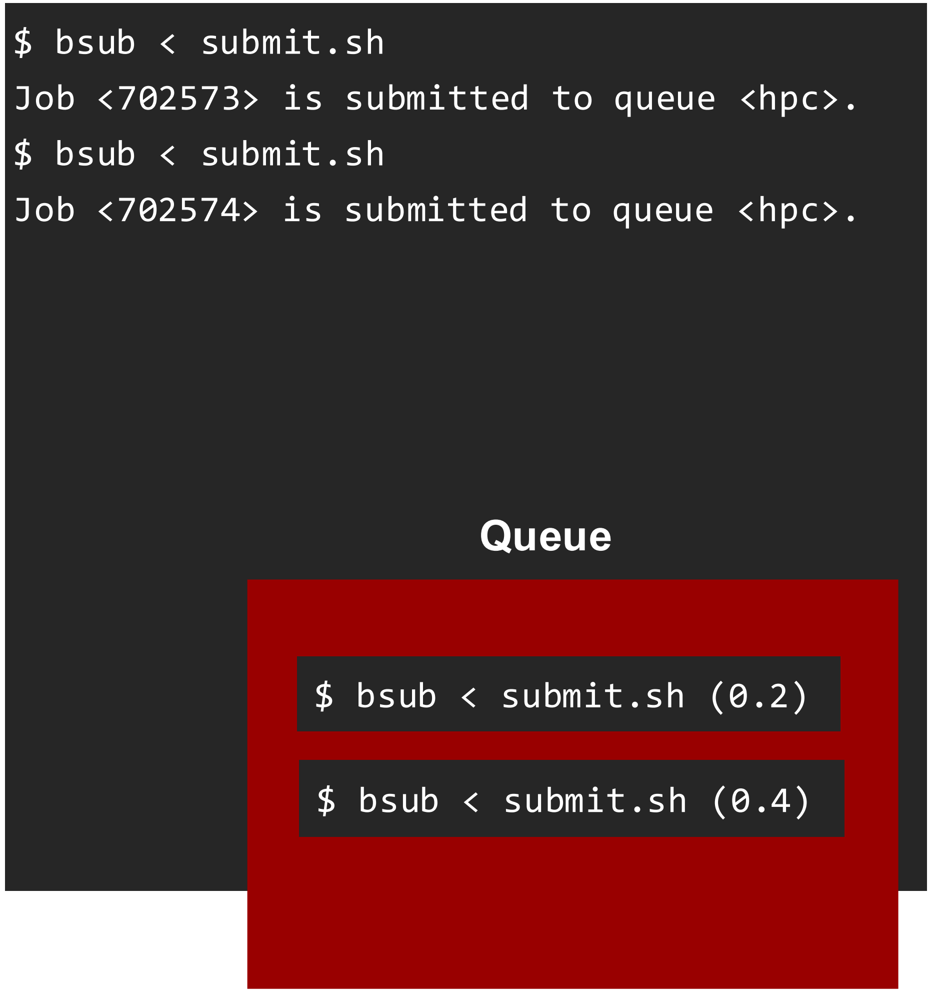
Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py path/to/data.txt 0.4
```

submit.sh



```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ bsub < submit.sh
Job <702574> is submitted to queue <hpc>.
```

Queue

```
$ bsub < submit.sh (0.2)
$ bsub < submit.sh (0.4)
```

Python in batch jobs

```
#!/bin/bash
```

Resources

```
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

Initialize

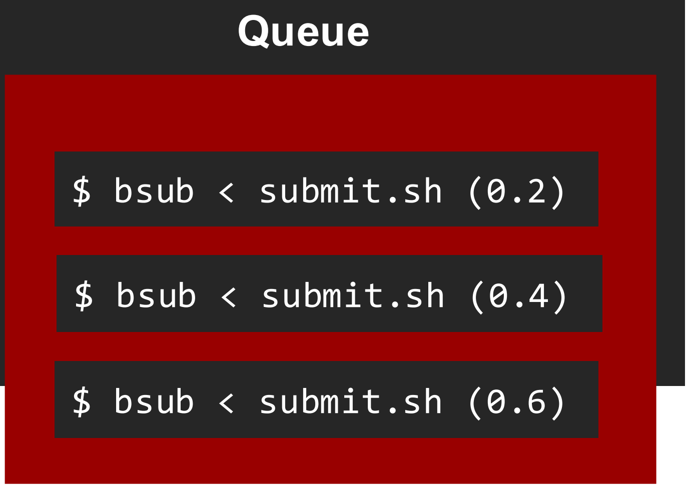
```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py path/to/data.txt 0.6
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ bsub < submit.sh
Job <702574> is submitted to queue <hpc>.
$ bsub < submit.sh
Job <702575> is submitted to queue <hpc>.
```



Queue

```
$ bsub < submit.sh (0.2)
$ bsub < submit.sh (0.4)
$ bsub < submit.sh (0.6)
```

Python in batch jobs: some tips

Python in batch jobs: unbuffered output

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

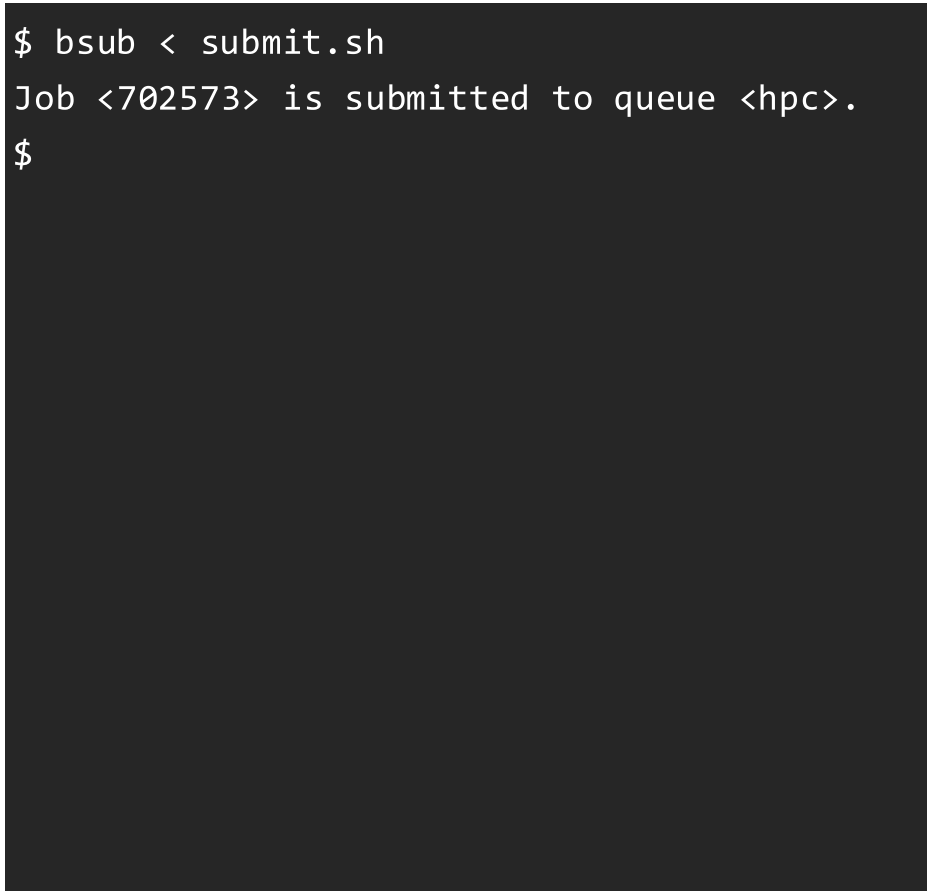
Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py
```

submit.sh



```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs: unbuffered output

```
#!/bin/bash
```

Resources

```
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

```
# Initialize Python environment
```

Initialize

```
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

```
# Run Python script
```

Run

```
python program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

```
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: unbuffered output

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ bstat
JOBID  ... STAT START_TIME    ELAPSED
702573 ... RUN  Dec 14 13:48  0:04:32
$
```

```
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: unbuffered output

```
#!/bin/bash
```

Resources

```
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ bstat
JOBID   ... STAT START_TIME    ELAPSED
702573 ... RUN  Dec 14 13:48  0:04:32
$ bpeek
<< output from stdout >>

<< output from stderr >>
$
```

No ”Hello World”!

```python
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: unbuffered output

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ bstat
JOBID   ... STAT START_TIME    ELAPSED
702573 ... RUN  Dec 14 13:48  0:04:32
$ bpeek
<< output from stdout >>

<< output from stderr >>
$ cat python_702573.out
$
```

No ”Hello World”!

```
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: unbuffered output

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ bstat
No unfinished job found
$ cat python_702573.out
Hello World

------------------------------------------
Sender: LSF System <lsfadmin@hpc.dtu.dk>
…
```

```
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: unbuffered output

Resources

Initialize

Run

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err

# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613

# Run Python script
python -u program.py
```

submit.sh

Unbuffered output

```
$ bsub < submit.sh
Job <702574> is submitted to queue <hpc>.
$ bstat
JOBID  ... STAT START_TIME    ELAPSED
702574 ... RUN  Dec 14 13:48  0:04:32
$ bpeek
<< output from stdout >>
Hello World

<< output from stderr >>
$
```

```python
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: unbuffered output

```
#!/bin/bash
```

Resources

```
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BS
```

Initialize

```
# I
sou
con
```

Run

```
# Run Python script
python -u program.py
```

Unbuffered output

submit.sh

Has performance implications!
Not good if you print a lot

```
$ bsub < submit.sh
Job <702574> is submitted to queue <hpc>.
$ bstat
JOBID   ... STAT START_TIME     ELAPSED
702574 ... RUN  Dec 14 13:48  0:04:32
$ bpeek
```

```python
import time
print(”Hello World”)
time.sleep(600)  # Sleep 10 minutes
```

program.py

Python in batch jobs: organizing your output

Resources

Initialize

Run

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err

# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613

# Run Python script
python -u program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs: organizing your output

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o python_%J.out
#BSUB -e python_%J.err
```

Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python -u program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ ls
program.py
python_702531.err
python_702531.out
python_702572.err
python_702571.err
python_702571.out
python_702572.err
python_702572.out
python_702573.err
python_702573.out
…
```

Python in batch jobs: organizing your output

Resources

Initialize

Run

Directory for output files

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o batch_output/python_%J.out
#BSUB -e batch_output/python_%J.err

# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613

# Run Python script
python -u program.py
```

submit.sh

```
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs: organizing your output

Resources

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o batch_output/python_%J.out
#BSUB -e batch_output/python_%J.err
```

Directory for output files

Initialize

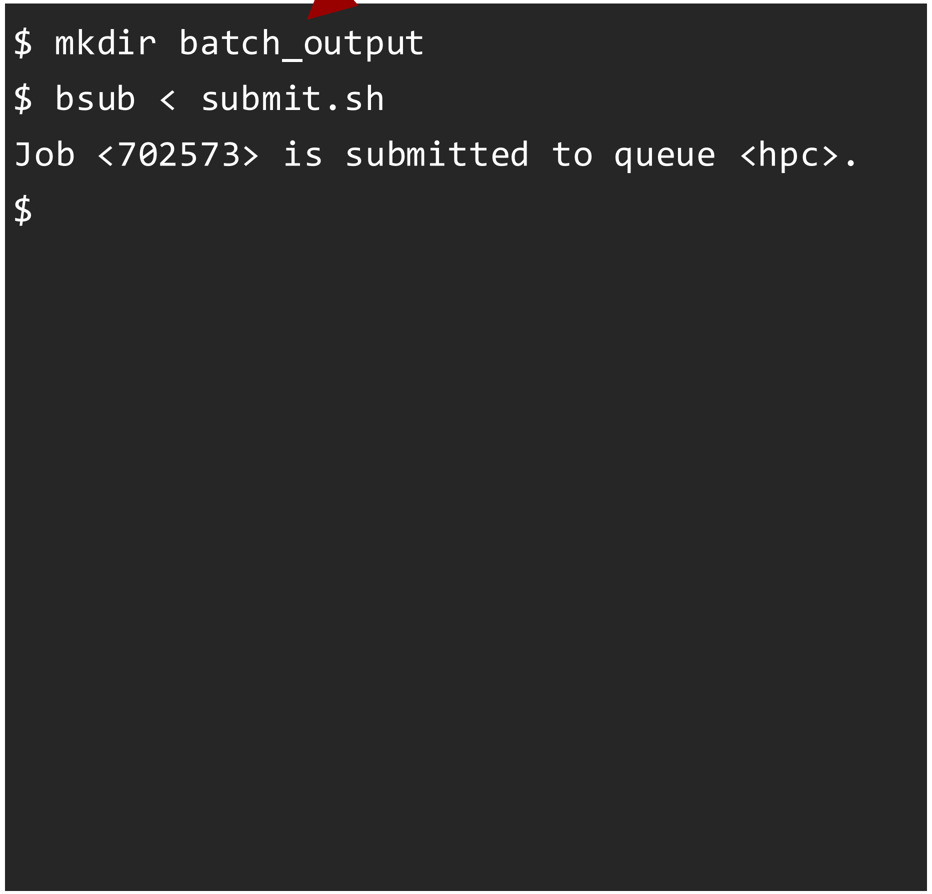
```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python -u program.py
```

submit.sh

Make directory before bsub!



```
$ mkdir batch_output
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$
```

Python in batch jobs: organizing your output

Make directory before bsub!

Resources

Directory for output files

```
#!/bin/bash
#BSUB -J python
#BSUB -q hpc
#BSUB -W 15
#BSUB -R ”rusage[mem=512MB]”
#BSUB -o batch_output/python_%J.out
#BSUB -e batch_output/python_%J.err
```

Initialize

```
# Initialize Python environment
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Run

```
# Run Python script
python -u program.py
```

submit.sh

```
$ mkdir batch_output
$ bsub < submit.sh
Job <702573> is submitted to queue <hpc>.
$ ls
batch_output
program.py
submit.sh
```

Today’s exercise

Get comfortable with Python on HPC

- Run Python interactively for testing

- Touch on relevant Python concepts

  command line args, functions, lambdas, classes,
  list comprehensions, basic NumPy, basic I/O

- Run Python in batch jobs

- **Remember:** Autolab exercises today do not count towards mandatory 20. But must still complete 1 (one from every week).

Useful commands

Log on to HPC

```
ssh <username>@login.hpc.dtu.dk
ssh <username>@login2.hpc.dtu.dk
ssh <username>@login.gbar.dtu.dk
ssh <username>@login2.gbar.dtu.dk
```

Init. Anaconda environment (also for job scripts)

```
source /dtu/projects/02613_2025/conda/conda_init.sh
conda activate 02613
```

Python with unbuffered output

```
python -u program.py
```

Python command line arguments

```
import sys
sys.argv      # List of arguments
```

Change to work node

```
linuxsh
```

Submit job script

```
bsub < submit.sh
```

Job status

```
bstat
bjobs
bjobs –p    # pending reason
```

Check job output

```
bpeek
bpeek <JOBID>
```

Kill job

```
bkill <JOBID>
```