Rensselaer

Submitty
Rensselaer Center for Open Source

Autograding Distributed Algorithms in Networked Containers

Evan Maicus,
Matthew Peveler,
Stacy Patterson,
Barbara Cutler

SCAN ME

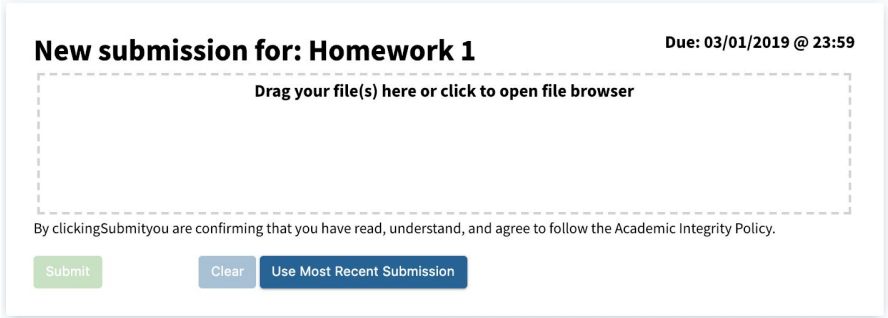
RCOS
Rensselaer Center for Open Source

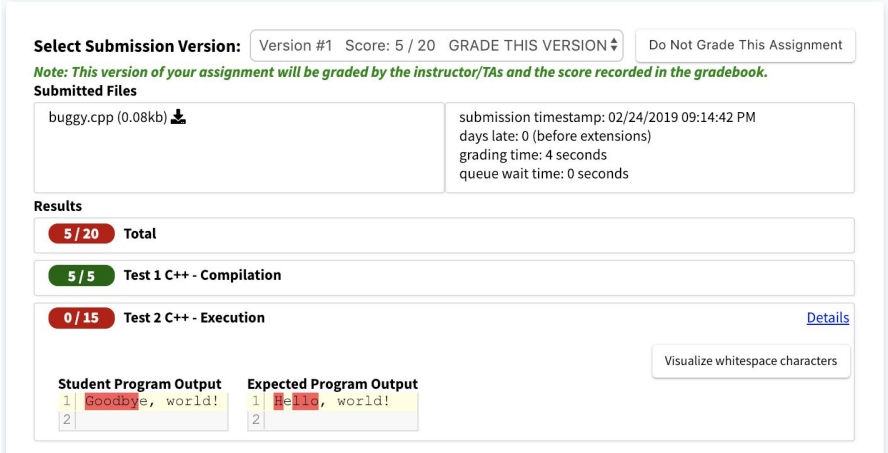
https://submitty.org/

Submitty

Rensselaer Center for Open Source

- A free, open source autograding platform
  - ~2500 users
  - 12-15 courses supported per term at RPI
  - In operation since 2014
- Support for:
  - Assignment Submission
  - Autograding
  - Exam Grading/Scanned PDF upload
  - Course Communications (Email/Forum)
  - Course Material Hosting
  - Plagiarism Detection

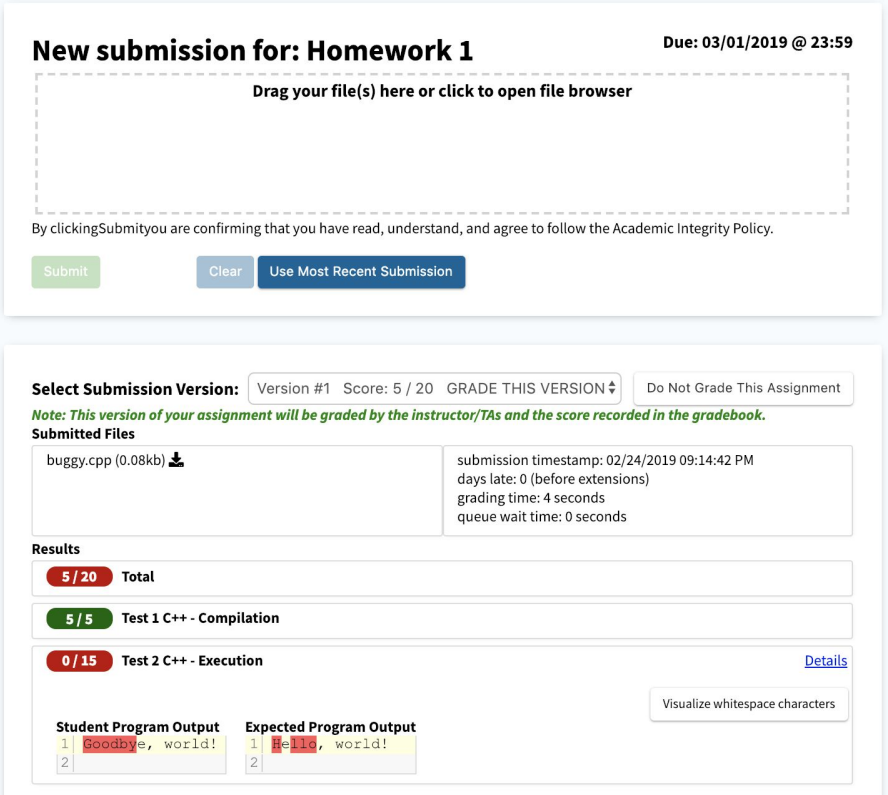




Autograding Configuration

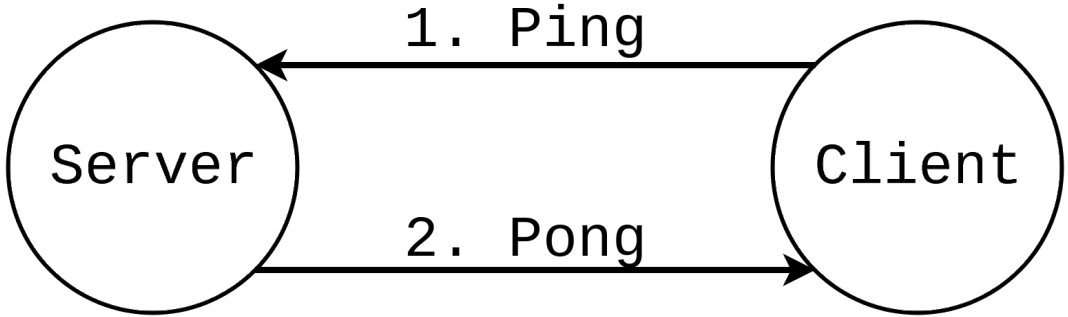
Submitty
Rensselaer Center for Open Source

```
{
  "testcases" : [
    {
      "type" : "Compilation",
      "title" : "C++ - Compilation",
      "command" : "clang++ -Wall -o a.out -- *.cpp",
      "executable_name" : "a.out",
      "points" : 5
    },
    {
      "title" : "C++ - Execution",
      "command" : "./a.out input_file.txt",
      "points" : 15,
      "validation" : [
        {
          "method" : "diff",
          "actual_file" : "STDOUT.txt",
          "description" : "Program Output",
          "expected_file" : "test1_output.txt"
        }
      ]
    }
  ]
}
```



Autograding Networked Assignments

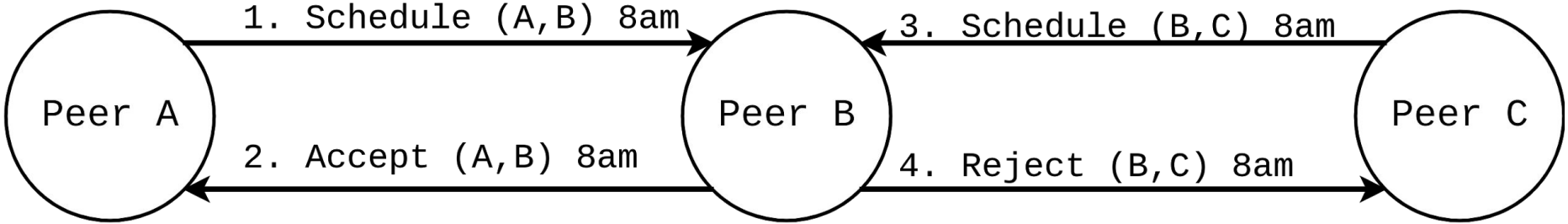
- Course sizes are swelling, autograding can help [Wilcox, 2016]
- The number of students in advanced topics courses is swelling
- We aim to autograde networked assignments
- **Networked Assignment:** Two or more **instances** of software which communicate over a network.
- **Distributed Algorithm:** An algorithm carried out over connected **instances**.



Complex Distributed Assignment

Peer-To-Peer Itinerary

- A node can **schedule** an event
- **Accepted** if the time is available
- **Rejected** if the time is not available



Challenges of Grading Distributed Assignments

Logistics:

1. Prepare multiple instances
2. Network each test
3. Coordinate invocation and input
4. Isolate student nodes and networks

Nondeterminism:

1. Issues of Concurrency
2. Network Stress
3. Issues of Distributed State [Beschastnikh et. al, 2015]

Related Work:

1. Validate abstract postconditions [Marroquin et al. 2015]
2. Unit test system components [Torens et. al, 2010]
3. Inject network stress [Lubke et al. 2013, Alnawasreh et. al, 2017, PUMBA]

Research Question

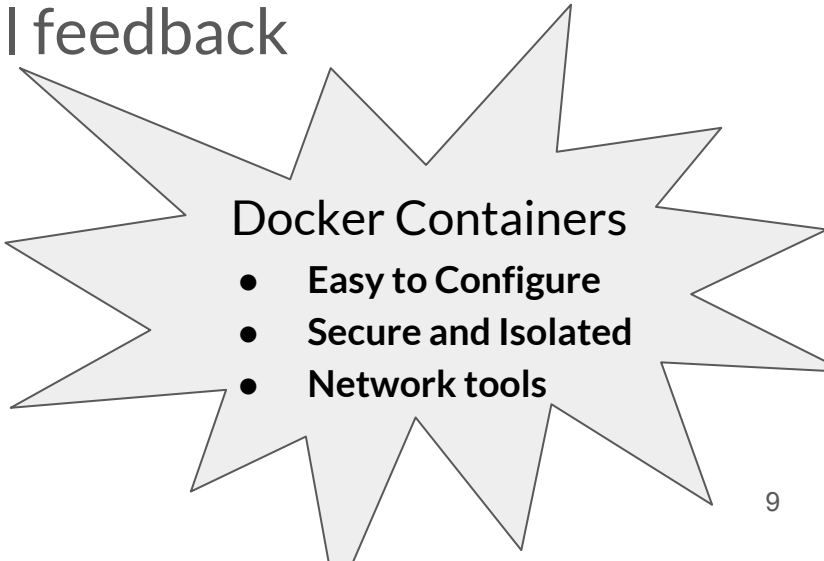
Can we build a lightweight tool to help instructors assess student mastery of distributed assignments? Might this tool additionally provide benefits to student learning?

Outline: Requirements of Our System

- Provide powerful, simple network specification tools
- Facilitate scripted interaction
- Control network conditions
- Provide students with meaningful feedback

Outline: Requirements of Our System

- Provide powerful, simple network specification tools
- Facilitate scripted interaction
- Control network conditions
- Provide students with meaningful feedback

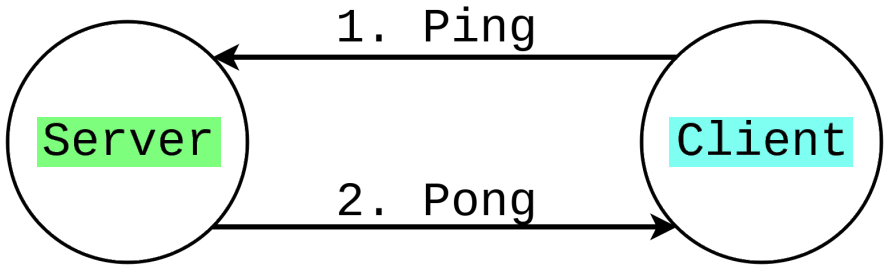


Generating Reasonably Sized Student Networks

For each instance:

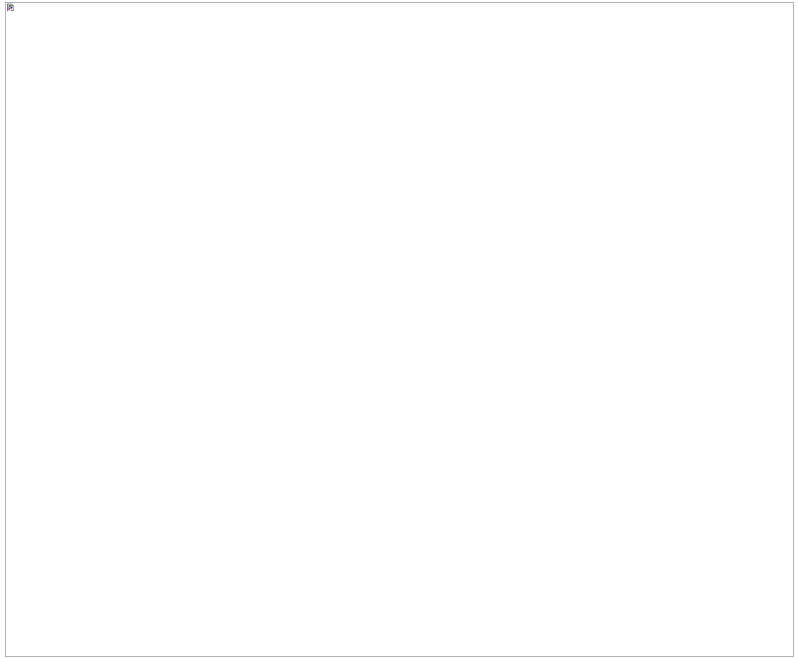
- **Host:** Where it's run
- **Connectivity:** Who it can talk to
- **Commands:** What it does
- **Environment:** What it has

```
"containers" : [
  {
    "container_name" : "server",
    "container_image" : "python:3.6",
    "outgoing_connections" : ["client"],
    "commands" : ["python3 server.py"]
  },
  {
    "container_name" : "client",
    "container_image" : "python:3.6",
    "outgoing_connections" : ["server"],
    "commands" : ["python3 client.py"]
  }
]
```



Itinerary Specification

```
"containers" : [
  {
    "container_name" : "Peer A",
    "container_image" : "python:3.6",
    "commands" : ["python3 itinerary.py"]
  },
  {
    "container_name" : "Peer B",
    "container_image" : "python:3.6",
    "commands" : ["python3 itinerary.py"]
  },
  {
    "container_name" : "Peer C",
    "container_image" : "python:3.6",
    "commands" : ["python3 itinerary.py"]
  }
]
```



Delivering Input to Student Applications

Communication

- Standard Input

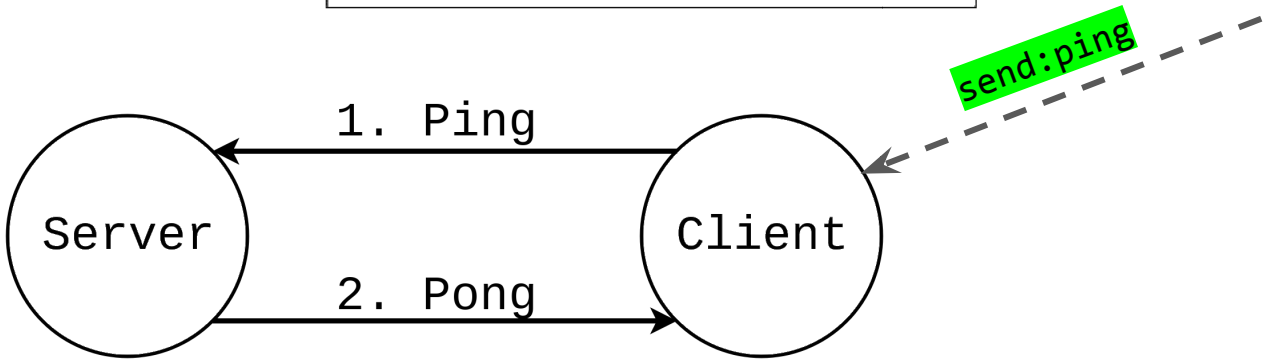
Fault Tolerance

- Stop
- Start
- Kill

Sequencing

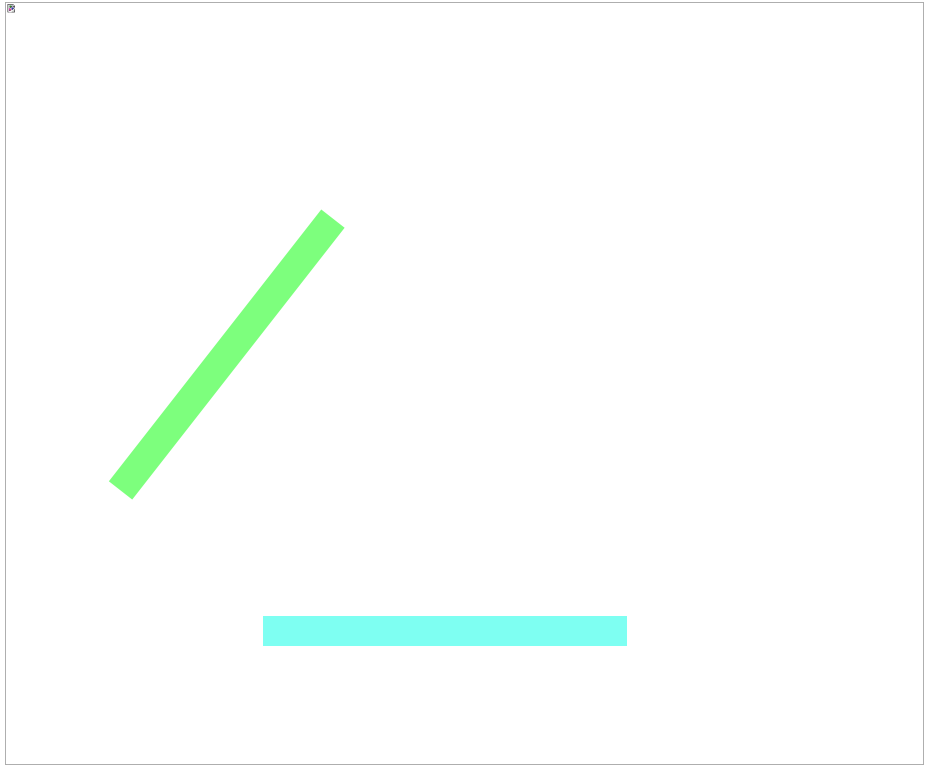
- Delay

```
"input_actions" : [
  {
    "containers" : ["Client"],
    "action" : "stdin",
    "string" : "send:ping\n"
  }
]
```



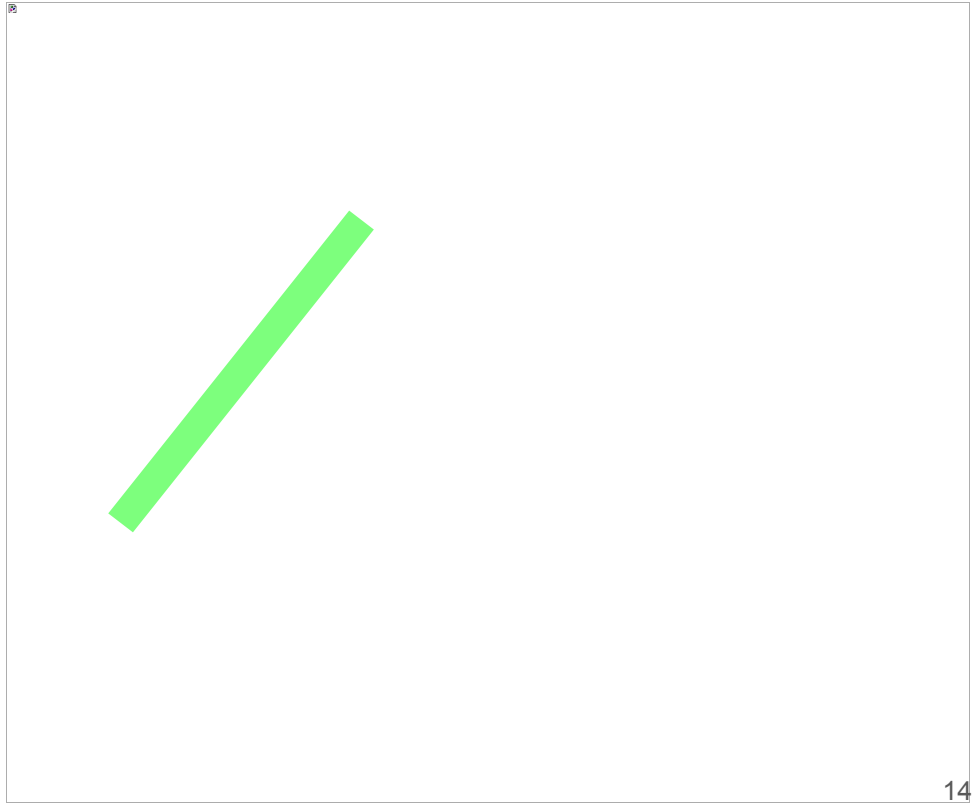
Itinerary Input

```
"input_actions" : [
  {
    "containers" : ["Peer A"],
    "action" : "stdin",
    "string" : "schedule B 8:00\n"
  },
  {
    "action" : "delay",
    "seconds" : 1
  },
  {
    "containers" : ["Peer C"],
    "action" : "stdin",
    "string" : "schedule B 8:00\n"
  },
  {
    "action" : "delay",
    "seconds" : 1
  },
  {
    "containers" : ["Peer A", "Peer B", "Peer C"],
    "action" : "stdin",
    "string" : "print itinerary\n"
  }
]
```



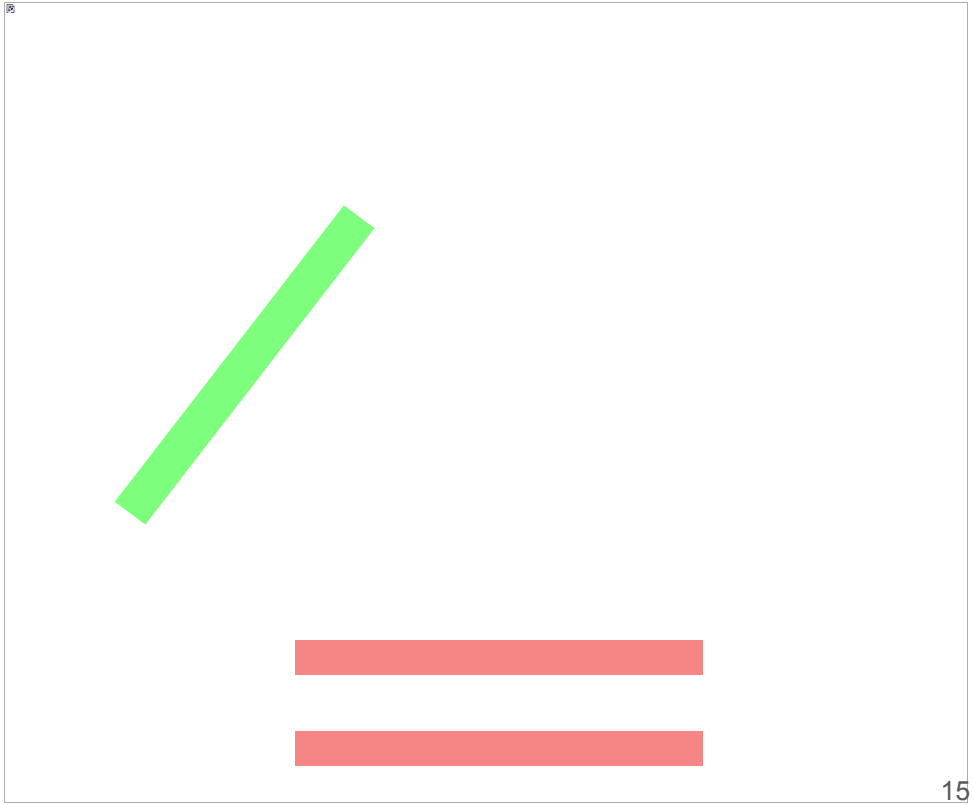
Testing Fault Tolerance

```
"input_actions" : [
  {
    "containers" : ["Peer A"],
    "action" : "stdin",
    "string" : "schedule B 8:00\n"
  },
  {
    "action" : "delay",
    "seconds" : 1
  },
  {
    "containers" : ["Peer B"],
    "action" : "stop"
  },
  {
    "containers" : ["Peer C"],
    "action" : "stdin",
    "string" : "schedule B 8:00\n"
  },
  {
    "action" : "delay",
    "seconds" : 1
  },
  {
    "containers" : ["Peer A", "Peer B", "Peer C"],
    "action" : "stdin",
    "string" : "print itinerary\n"
  }
]
```



Delivering Input to Student Applications

```
"input_actions" : [
  {
    "containers" : ["Peer A"],
    "action" : "stdin",
    "string" : "schedule B 8:00\n"
  },
  {
    "action" : "delay",
    "seconds" : 1
  },
  {
    "containers" : ["Peer B"],
    "action" : "stop"
  },
  {
    "containers" : ["Peer C"],
    "action" : "stdin",
    "string" : "schedule B 8:00\n"
  },
  {
    "action" : "delay",
    "seconds" : 1
  },
  {
    "containers" : ["Peer A", "Peer B", "Peer C"],
    "action" : "stdin",
    "string" : "print itinerary\n"
  }
]
```

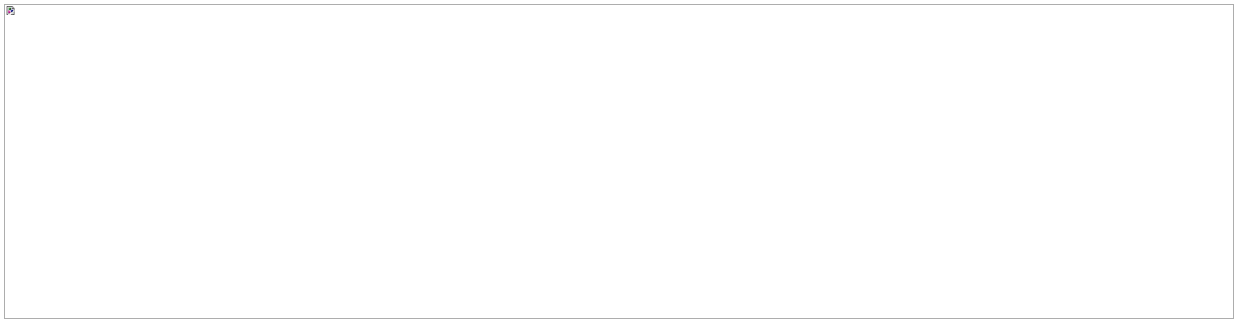


Providing Students with Meaningful Feedback

- It is difficult to determine a “happens before” relationship when debugging distributed algorithms [Beschastnikh et. al, 2015].

The Logging Node

- Transparently injected into student applications.
- Intercepts all messages to provide a totally ordered message log
- **Network Aliasing**



System Output

C1 Log:

```
1  Initialized
2  Requested an appointment with C2 at 8:00
3  Appointment Scheduled
```

C2 Log:

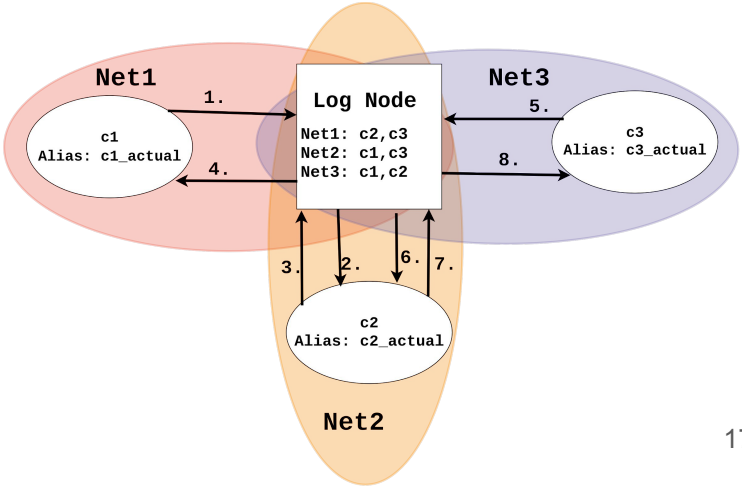
```
1  Initialized
2  C1 requested an appointment at 8:00
3  Accepting
4  C3 requested an appointment at 8:00
5  Rejecting
```

C3 Log:

```
1  Initialized
2  Requested an appointment with C2 at 8:00
3  Appointment Conflict
```

Logger Output:

```
1  Message passed from C1 to C2: "appt-8:00"
2  Message passed from C2 to C1: "accept"
3  Message passed from C3 to C2: "appt-8:00"
4  Message passed from C2 to C3: "conflict"
```



Transparent Logging Node

Advantages to the Student

- Provide an additional debugging resource

Advantages to the Professor

- Impose network stress
- Enforces protocol use (TCP vs UDP)
- Restrict container communication
- Gauge program efficiency

Limitations

- False Positive Message Receipt
- Sender Confusion
- Does not support Swarm Networks

Case Study: Distributed Systems and Algorithms 2018

Two Peer-To-Peer Assignments

- Team assignments
- Distributed Itinerary using log replication algorithm by Wuu and Bernstein, then Paxos

Live Demonstrations

- 20 minute face to face sessions with the instructor
- 7 cumulative testcases which increase in difficulty
- Adaptive expected output

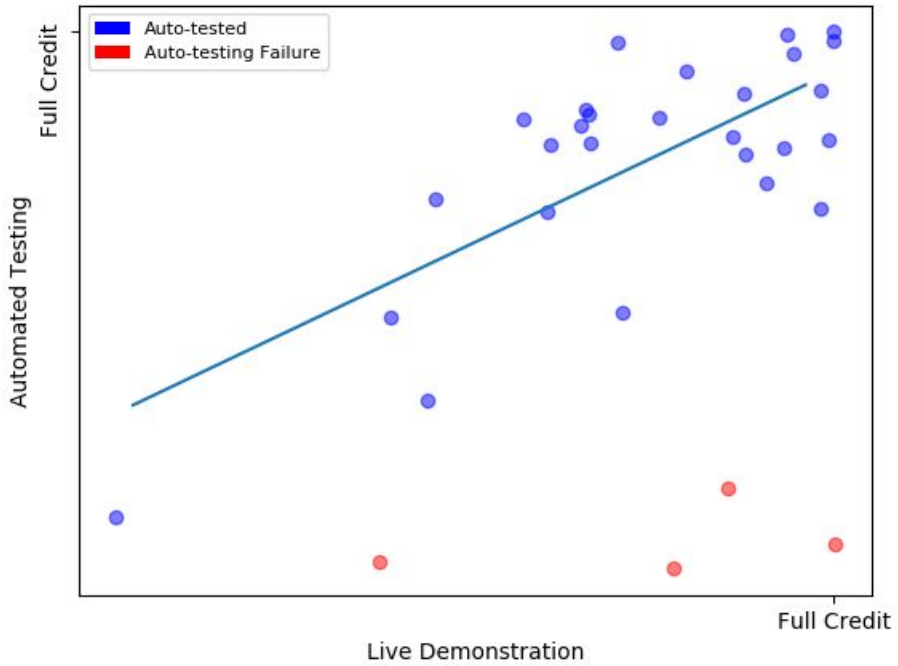
Our System:

- Used as a prerequisite for participation
- Post-semester study

Case Study: Distributed Systems and Algorithms 2018

Post-Semester Study

- 31 student pairs, 27 successfully graded
- Offline test with no feedback or resubmission
- 7 adapted testcases
- Pearson Correlation of .78
- Live demonstrations allow for more granular partial credit



Autograding Complemented by Live Demonstration

Advantages of Autograding:

- Multiple Submissions/early testing
- More testcase coverage
- Students can focus on the algorithm, rather than testing process
- Output is gathered to assist students in debugging

Advantages of Live Demonstration:

- Better partial credit
- Adaptive corner case testing
- Dialog with the student

Contributions

- **An Interface to Easily Create Student Networks**
  - Create networks and deploy student code
  - Automate timely interaction
  - Test program fault tolerance
- **Control Network Conditions with the Logging Node**
  - Simulate network stress
  - Enforce protocol use
- **Provide Useful Information to Students**
  - The logging node provides a totally ordered message log
- **Network generation is available as a standalone tool**

Future Work

- **Additional Utilities for Controlling Network Conditions**
  - An interface to inject message delay
  - Chaos testing
- **Additional Output for Students**
  - Interleaved Output
- **Autograding Additional Advanced Topics Course**
  - Database Systems
  - Computer Graphics

Rensselaer

Submitty

Rensselaer Center for Open Source

SCAN ME

https://submitty.org/

For More Submitty:

Next Paper:
Comparing Jailed Sandboxes vs Containers Within an Autograding System

Tomorrow @3pm, Poster:
Lichen: Customizable, Open Source Plagiarism Detection in Submitty

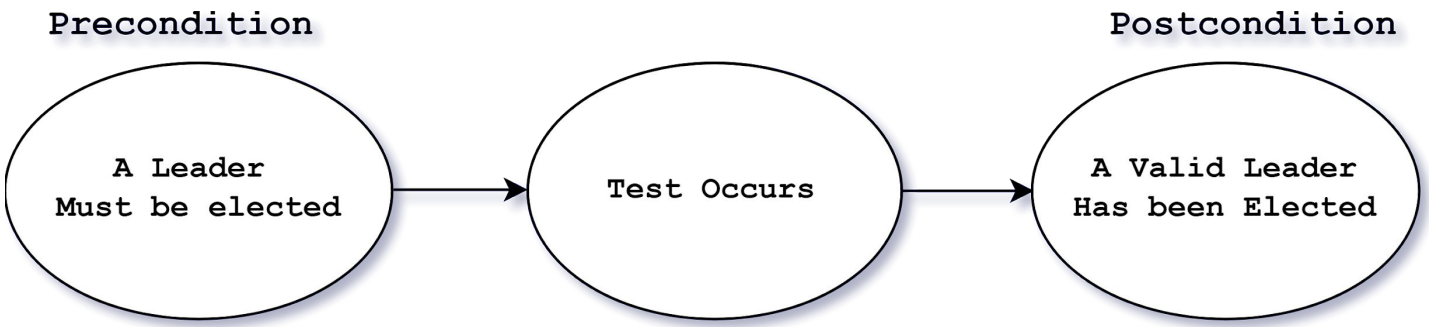
Tomorrow @3pm, Poster:
Facilitating Discussion-Based Grading And Private Channels via an Integrated Forum

References

1. C. Wilcox, “Testing strategies for the automated grading of student programs,” in Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE ’16, (New York, NY, USA), pp. 437–442, ACM, 2016.
2. A. Marroquin, D. Gonzalez, and S. Maag, “Testing distributed systems with test cases dependencies architecture,” in 2015 7th IEEE Latin-American Conference on Communications (LATINCOM), pp. 1–6, Nov 2015.
3. I. Beschastnikh, P. Wang, Y. Brun, and M. D. Ernst, “Debugging distributed systems,” Queue, vol. 14, pp. 50:91–50:110, Mar. 2016.
4. R. Lu¨bke, D. Schuster, and A. Schill, “Reproducing network conditions for tests of large-scale distributed systems,” in 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, pp. 74–77, May 2013.
5. C. Torens and L. Ebrecht. 2010. RemoteTest: A Framework for Testing Distributed Systems.
6. K. Alnawasreh, P. Pelliccione, Z. Hao, M. RÃěnge, and A. Bertolino. 2017. Online Robustness Testing of Distributed Embedded Systems: An Industrial Approach. In 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). 133–142.
7. TERRA NULLIS. 2018. Pumba - Chaos Testing for Docker. https://alexei-led. github.io/post/pumba_docker_chaos_testing/

Validating Distributed Algorithms

- **Instructor Client, Student Server.**
  - **Instrumentation**
- **Input Actions:** Test peer to peer. Test fault tolerance.
- **Validate Abstract Postconditions** [Marroquin et al. 2015]



Capturing Output

Peer A Log:

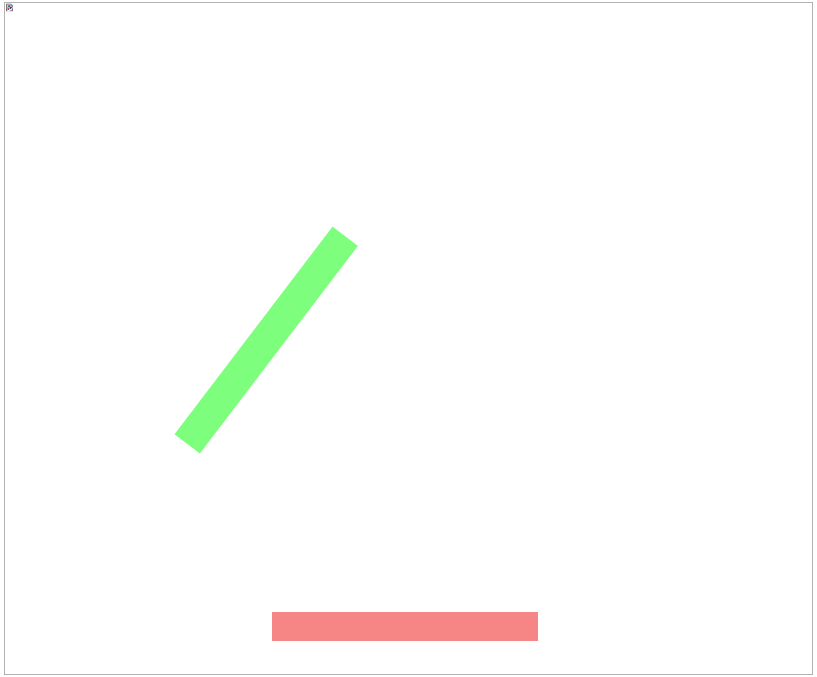
```
1  Initialized
2  Requested an appointment with Peer B at 8:00
3  Appointment Scheduled
```

Peer B Log:

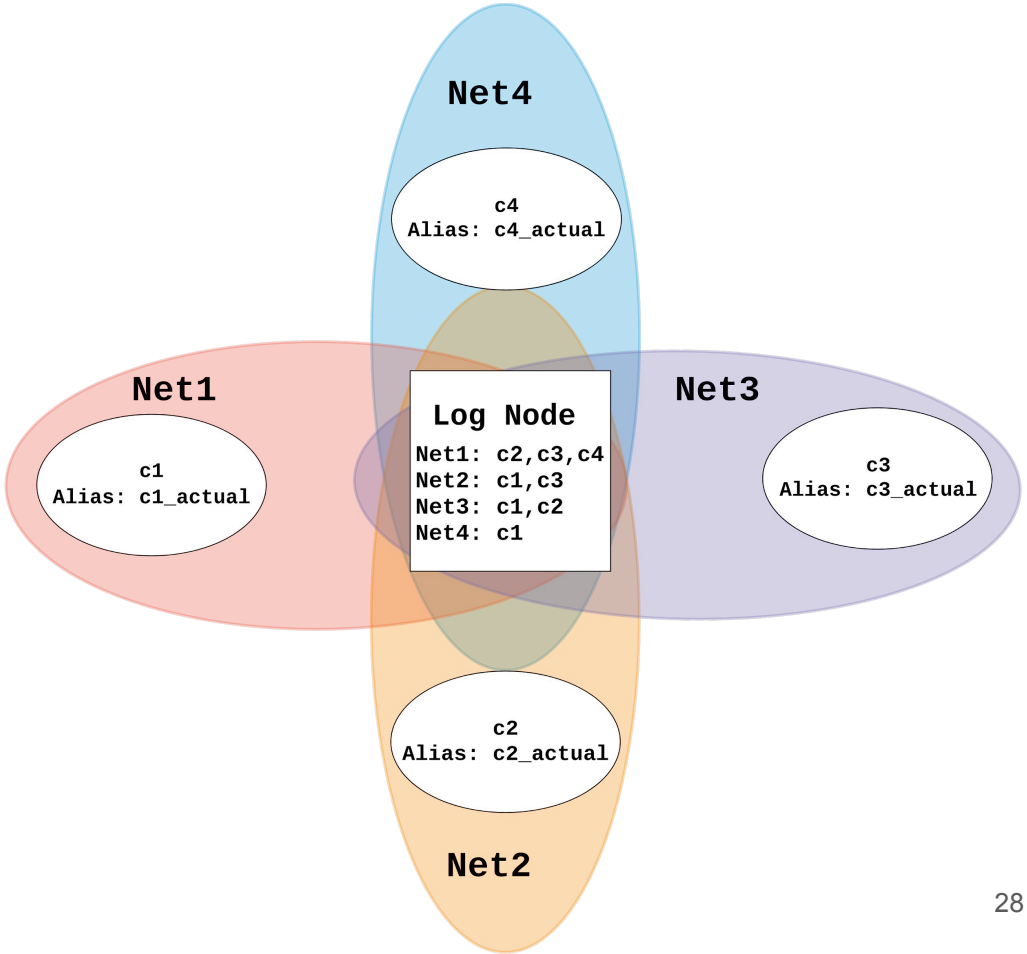
```
1  Initialized
2  Peer A requested an appointment at 8:00
3  Accepting
4  Peer C requested an appointment at 8:00
5  Rejecting
```

Peer C Log:

```
1  Initialized
2  Requested an appointment with Peer B at 8:00
3  Appointment Conflict
```



Using Docker Networks to Monitor and Restrict Communication



Methods of Validation

- **Output Comparison Testing**

Student Program Output

```
#icebucketchallenge vs #alsicebucketchallenge, percentage change
150.0 vs 200.0
300.0 vs 400.0
500.0 vs 766.666666667
100.0 vs 92.3076923077
170.833333333 vs 320.0
7.69230769231 vs -19.0476190476

```

Expected Program Output

```
#icebucketchallenge vs #alsicebucketchallenge, percentage change
150 vs 200
300 vs 400
500 vs 766
100 vs 92
170 vs 320
7 vs -19

```

- **Script Based Testing:** Output is preprocessed using a Unix command

```
carrot
apple
banana
```

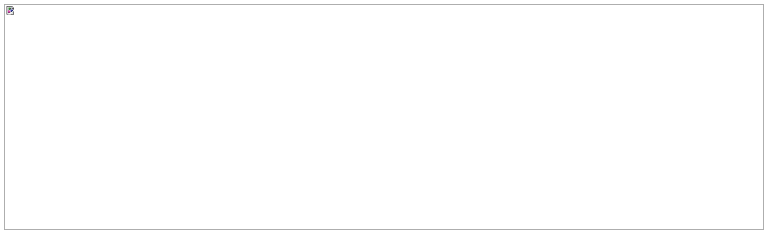
```
apple
banana
carrot
```

- **Output Analysis:** A custom algorithm is run on student output

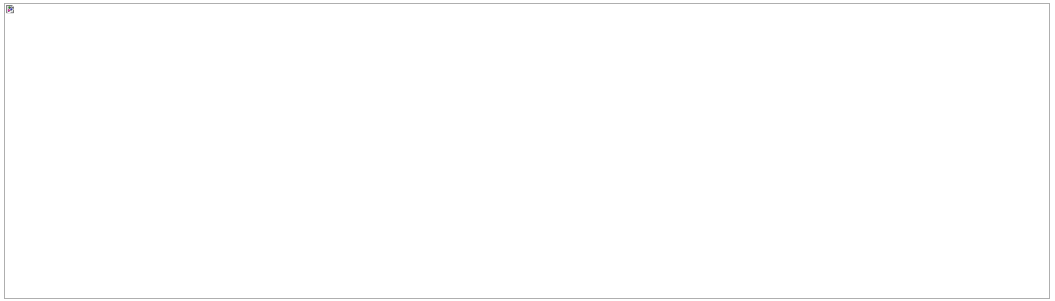
```
1 + 2 = 3
3 - 4 = -1
5 + 4 = 9
```

Limitation: False Positive Message Receipt

Ideal:



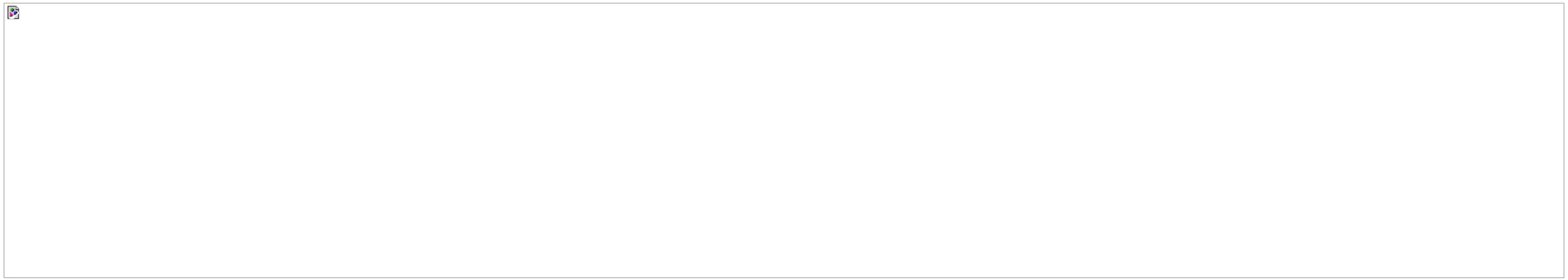
Our System:



Complex Distributed Assignment

Peer-To-Peer Itinerary

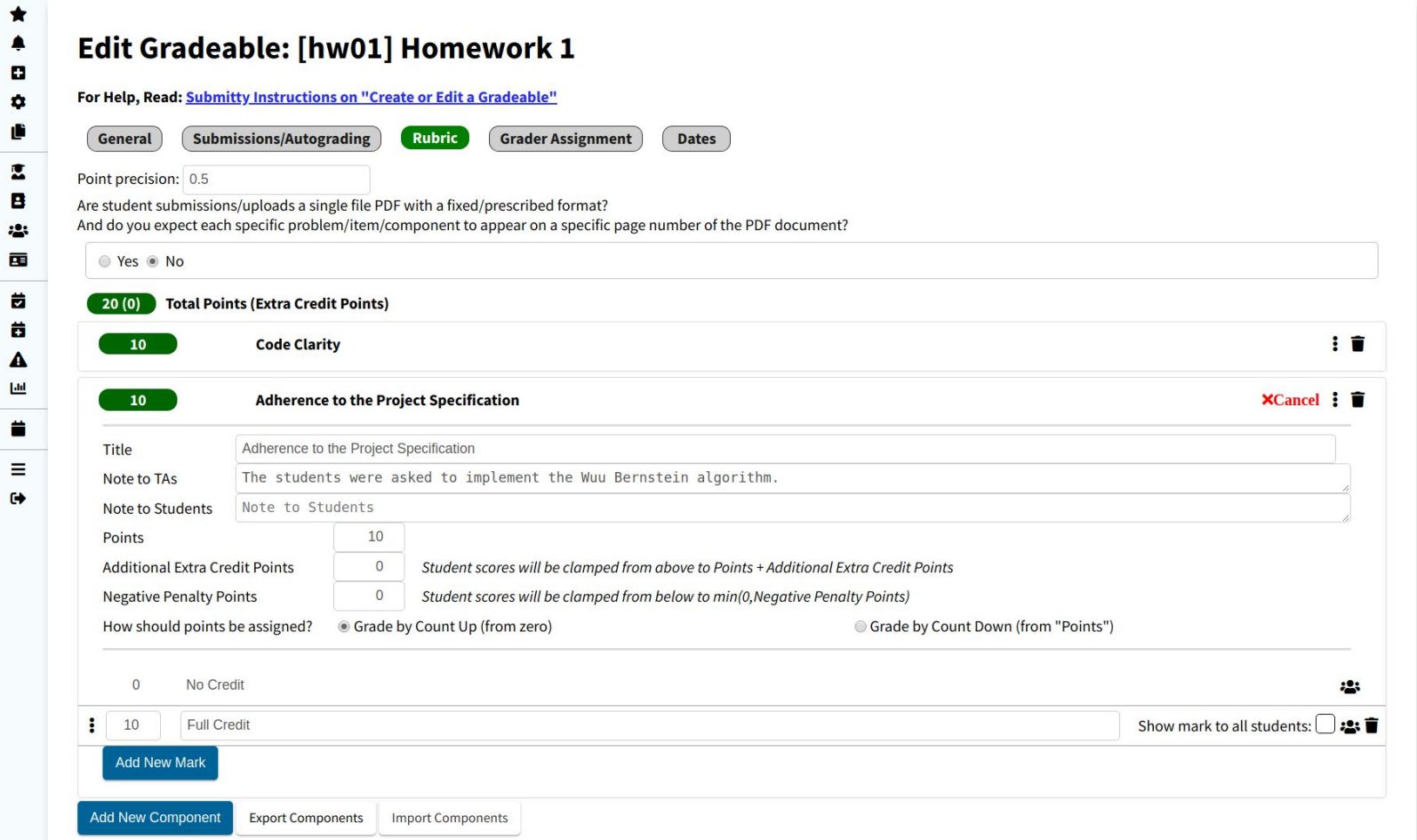
- A node can **schedule** an event
- **Accepted** if the time is available
- **Rejected** if the time is not available



Docker Containers

Lightweight, secure environments virtualized at the operating system level and used to execute applications.

- Performant
- Easy to Configure
- Secure and Isolated
- Network tools


Edit Gradeable: [hw01] Homework 1
For Help, Read: Submitty Instructions on "Create or Edit a Gradeable"
General
Submissions/Autograding
Rubric
Grader Assignment
Dates
Point precision: 0.5
Are student submissions/uploads a single file PDF with a fixed/prescribed format?
And do you expect each specific problem/item/component to appear on a specific page number of the PDF document?
Yes
No
20 (0)
Total Points (Extra Credit Points)
10
Code Clarity
10
Adherence to the Project Specification
Cancel
Title
Adherence to the Project Specification
Note to TAs
The students were asked to implement the Wuu Bernstein algorithm.
Note to Students
Note to Students
Points
10
Additional Extra Credit Points
0
Student scores will be clamped from above to Points + Additional Extra Credit Points
Negative Penalty Points
0
Student scores will be clamped from below to min(0,Negative Penalty Points)
How should points be assigned?
Grade by Count Up (from zero)
Grade by Count Down (from "Points")
0
No Credit
10
Full Credit
Show mark to all students:
Add New Mark
Add New Component
Export Components
Import Components

Student Submission/Autograding

Submitty > **sample** > **Open Homework**

Submitty
Rensselaer Center for Open Source

- Gradeables
- Notifications
- New Gradeable
- Course Settings
- Course Materials
- Discussion Forum
- Manage Students
- Manage Graders
- Manage Sections
- Student Photos
- Late Days Allowed
- Excused Absence Extensions
- Plagiarism Detection
- Grade Reports
- My Late Days/Extensions
- Collapse Sidebar
- Logout Quinn

New submission for: Open Homework

Due: 12/31/9996 @ 23:59

Normal Submission | Make Submission for a Student | Bulk Upload

Drag your file(s) here or click to open file browser

By clickingSubmityou are confirming that you have read, understand, and agree to follow the Academic Integrity Policy.

Submit | Clear | Use Most Recent Submission

Select Submission Version: Version #3 GRADE THIS VERSION

Do Not Grade This Assignment

Note: This version of your assignment will be graded by the instructor/TAs and the score recorded in the gradebook.

Submitted Files

part1_buggy2.py (0.48kb)

submission timestamp: 02/08/2019 05:30:09 PM
days late: 0 (before extensions)
grading time: 2 seconds
queue wait time: 0 seconds

Results

Test 1 Lab 1 Checkpoint 1 `python *.py` — Details

Visualize whitespace characters

Student Program Output

```
1 #icebucketchallenge vs #alsicebucketchallenge, percentage change
2 150.0 vs 200.0
3 300.0 vs 400.0
4 500.0 vs 766.666666667
5 100.0 vs 92.3076923077
6 170.833333333 vs 320.0
7 7.69230769231 vs -19.0476190476
8
```

Expected Program Output

```
1 #icebucketchallenge vs #alsicebucketchallenge, percentage change
2 150 vs 200
3 300 vs 400
4 500 vs 766
5 100 vs 92
6 170 vs 320
7 7 vs -19
8
```