CEE LAB

The Computational Experimental Economics Laboratory

"Microeconomic Systems Experiments Using mTree"

Kevin A. McCabe
Computational Experimental Economics Lab
Interdisciplinary Center for Economic Science
George Mason University
kmccabe@gmu.edu

Your Experiment

mTree Subject Interface - - Subject ID: 8d781 - Total Earnings:

Messages

Experiment has concluded

Information

ID:
Period: 5
Total Earnings: 1049

Previous Period Results

Results for previous period: 5
Returns on group investment: 196.66666666666666 (5 x total)/num participants
Returns on savings: 55
Period Earnings: 251.66666666666666

Decision

Beginning Cash: 100 ← Starting amount each period
Rate of Return on Group Investment: 5
Rate of Return on Savings: 1 } rate of returns
Number of Participants: 6 ← Only if you submit investment
Time left: 0 ← 60 second countdown timer
Total invested so far: 236 ← Total group investment so far
Savings: 55 ← 100 – your group investment
Enter Your Group Investment:

45

Submit Investment

Enter investment from 0 to 100

Increment/Decrement Investment

Must hit submit to make investment
Savings will update after submit
Submit as often as you like until the period time is up. Last submission counts.

History

Period	Investment	Total	# Investors	Earnings
1	25	47	6	114.16666666666666
2	0	88	6	173.33333333333331
3	50	110	5	160
4	1	301	6	349.83333333333337
5	45	236	6	251.66666666666666

mTree as an implementation of actors for microeconomic systems design

Stephen Kunath Dissertation (forthcoming)

Goal: Integrate Microeconomic Systems with Action Situations to run agent-based simulations and human-subject experiments.

Source Code: https://github.com/gmucsn/mTree

: mTree is available as a Docker file explained in the documentation.

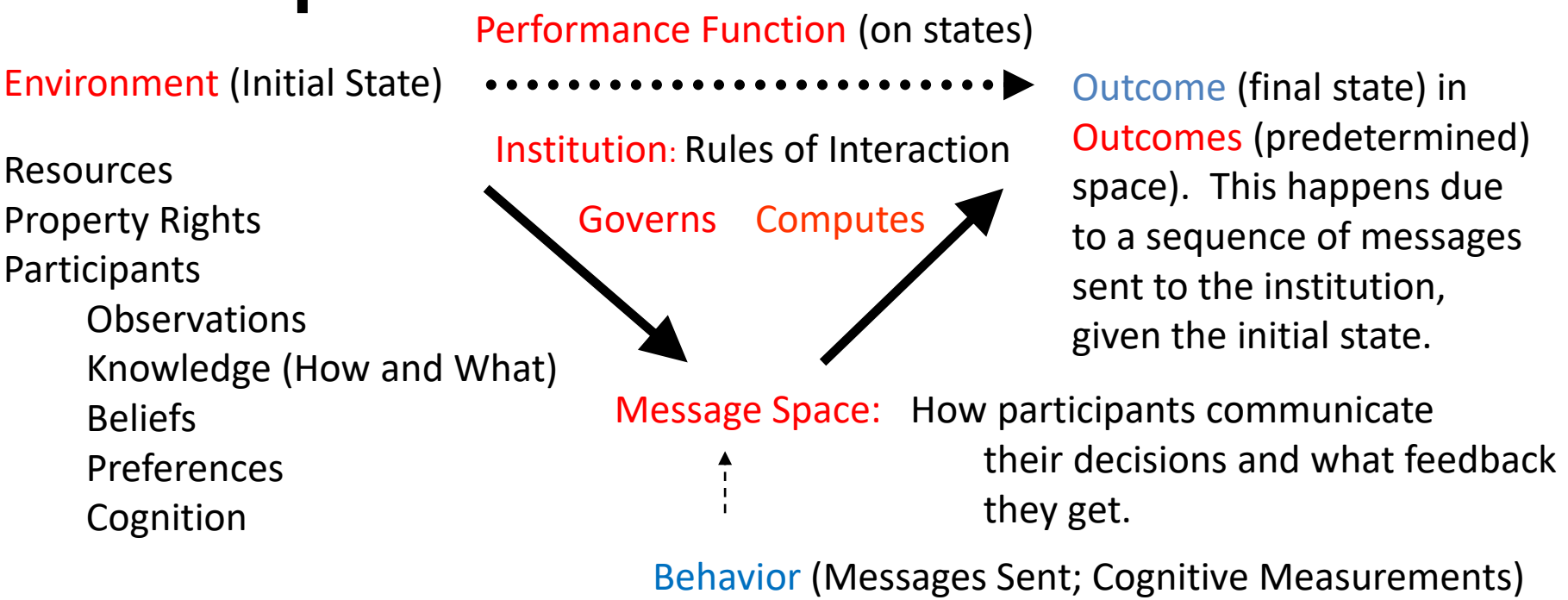
Documentation:

Overview — mTree v1 documentation (mtree-doc-develop.readthedocs.io)

Frameworks that Influence mTree Design

- Economic Frameworks
  - Microeconomic Systems (Institutions)
  - Action Situations (Organizations)
- Computer Science Frameworks
  - Thespian Actor Systems (Concurrent Computations)
  - Flask Micro Web Framework (Network Interactions)

Microeconomic Systems

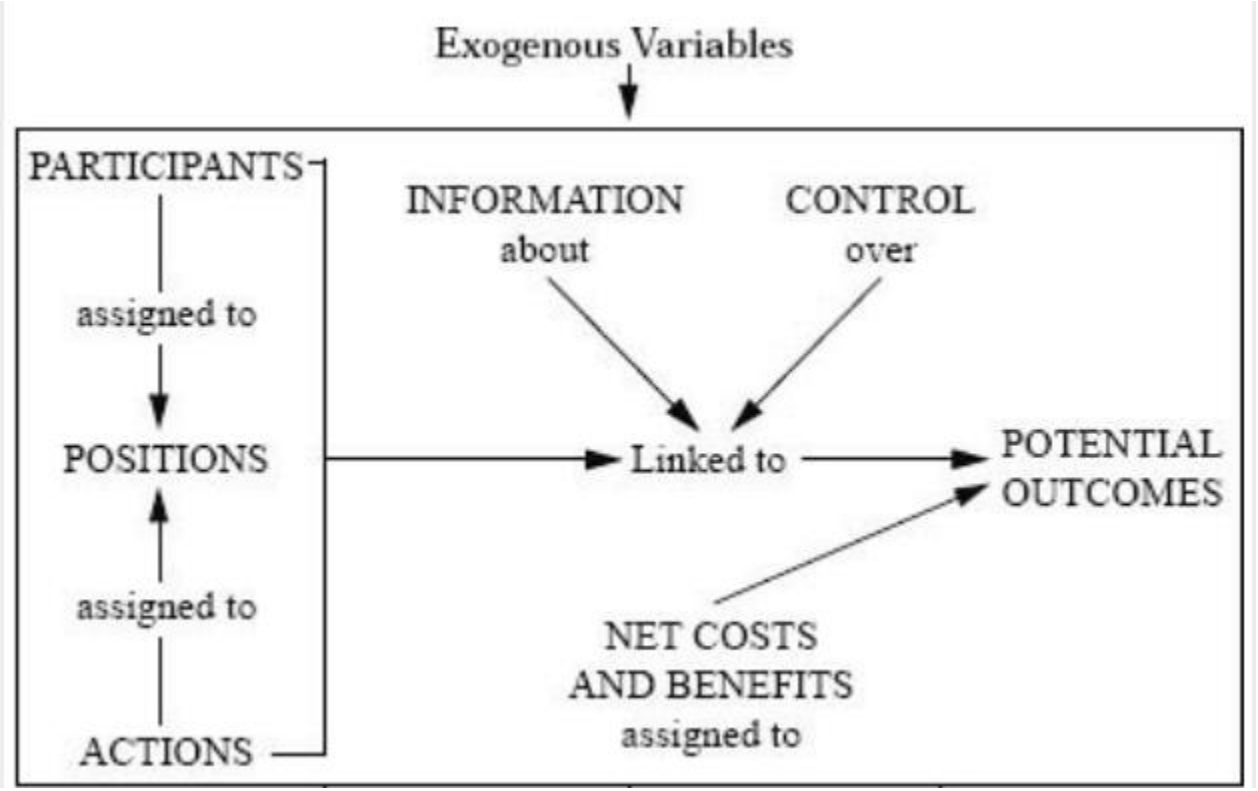


Hurwicz, Leonid. "Optimality and informational efficiency in resource allocation processes," in Mathematical models in the social sciences, 1959.

Smith, Vernon L. "Microeconomic Systems as an Experimental Science." *The American Economic Review* 72, no. 5 (1982): 923–55.

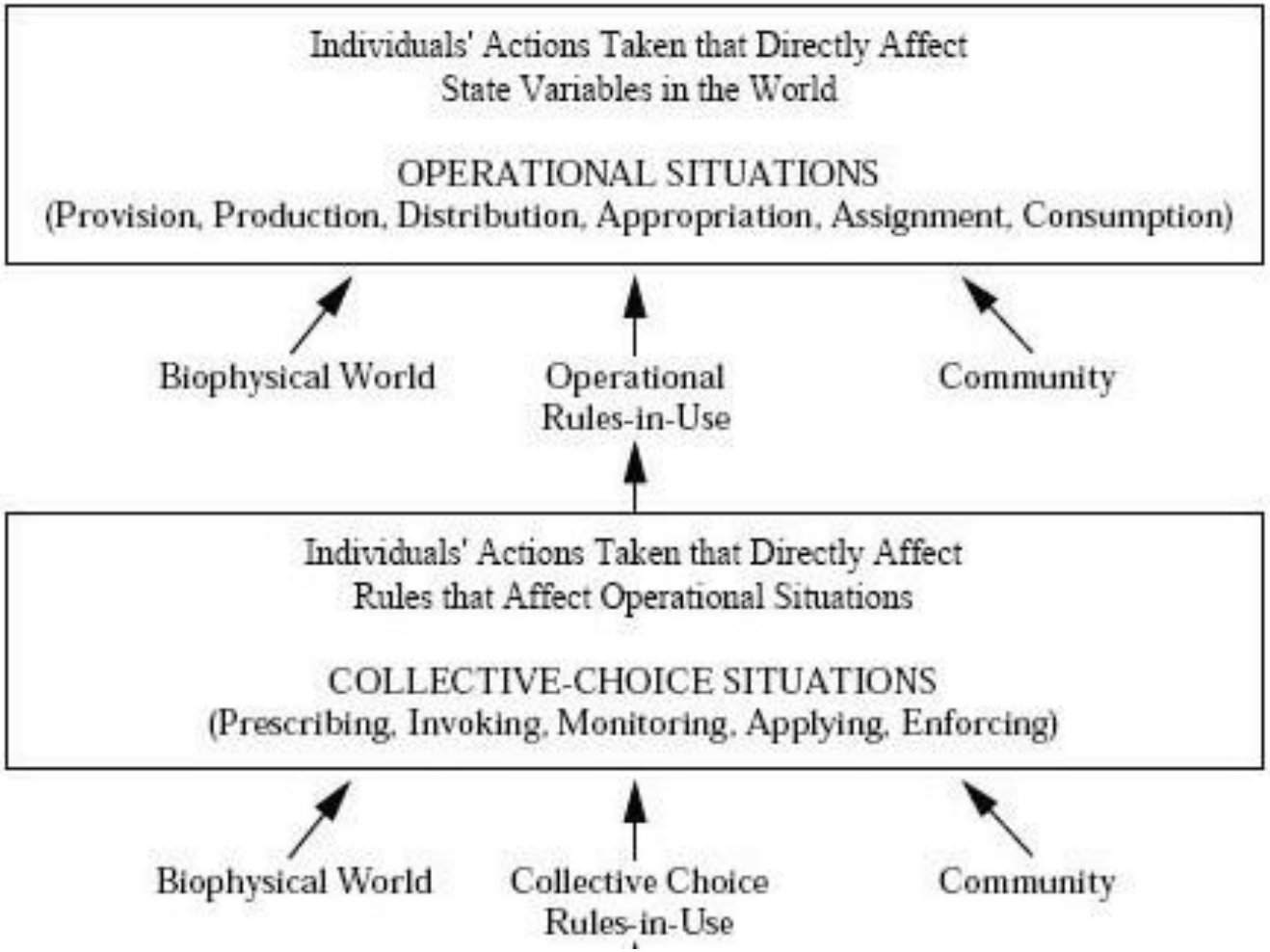
Mount, Kenneth R., and Stanley Reiter. *Computation and Complexity in Economic Behavior and Organization*. Cambridge University Press, 2002.

Action Situations

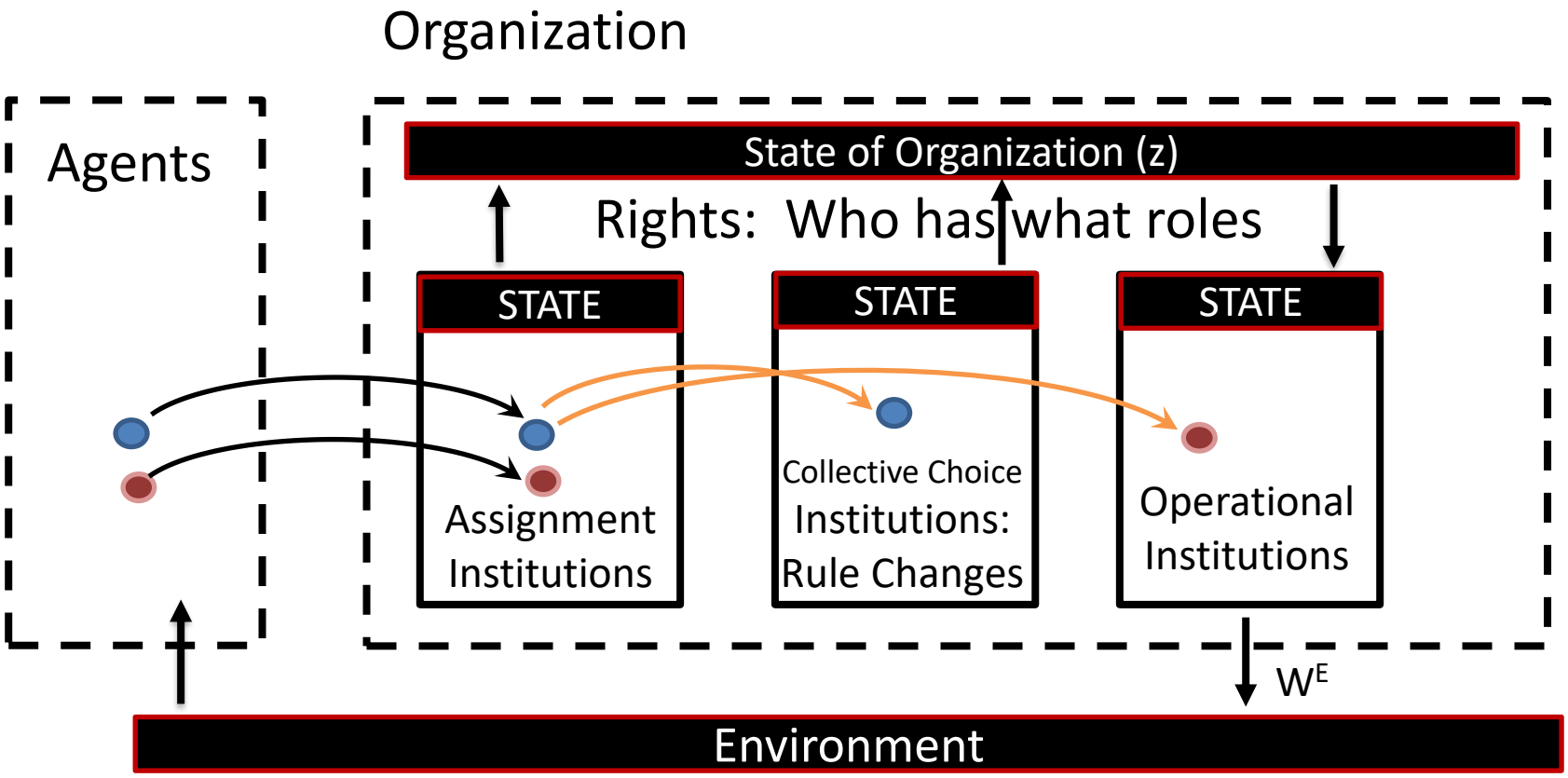


Ostrom, Elinor. “Understanding Institutional Diversity", Princeton University Press, 2005.

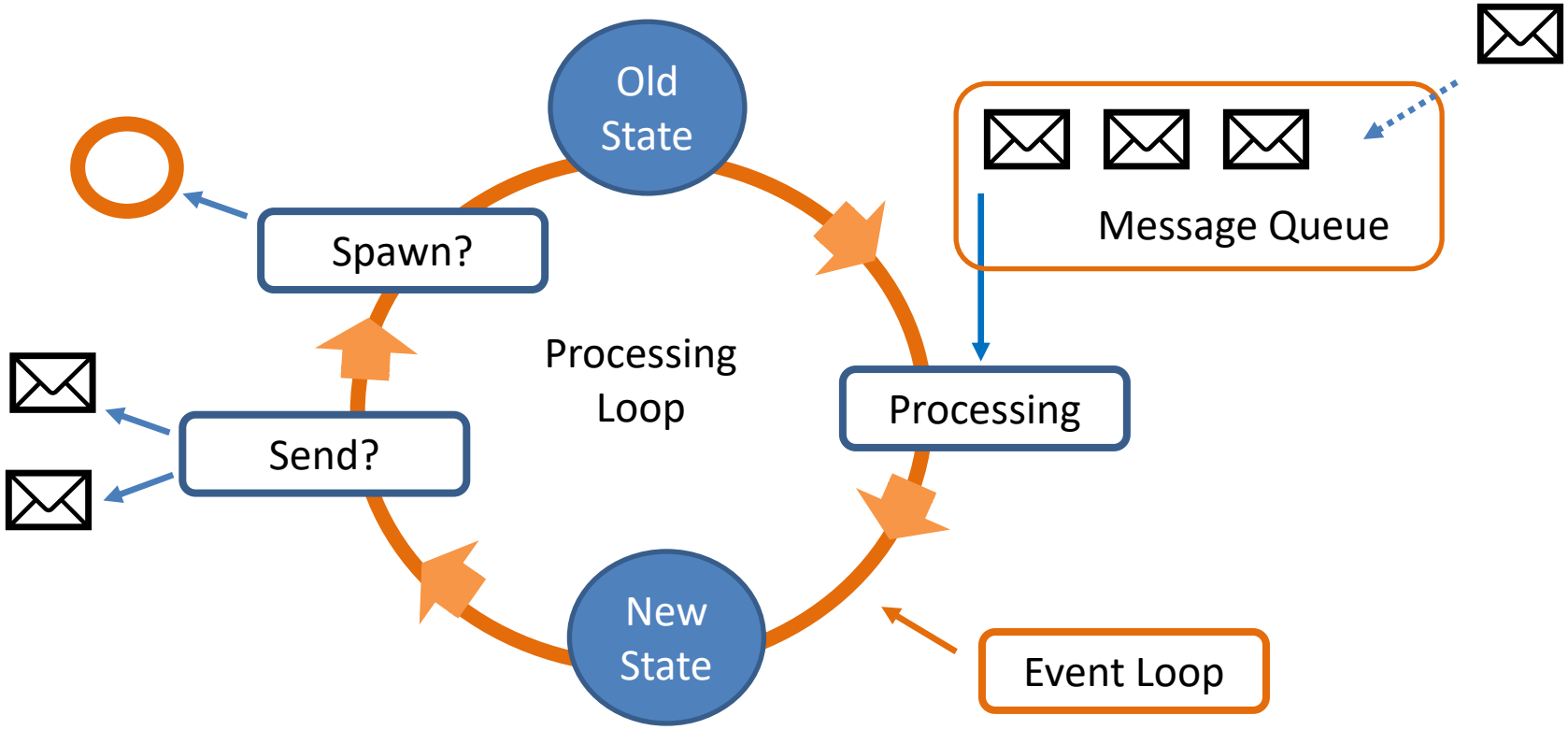
Organizations as Collective Choice



Organizations Spawn/Own/Manage Institutions



Actor Systems as Models of Decentralized Computations



✉ = (directive, sender_addr, receiver_addr, payload)

Agha, Gul. “Actors: A model of concurrent computation in distributed systems", MIT Press, 1985.

What you need to know to program a mTree experiment (computer related)*

- Basic Python programming skills.
- Basic Operating system skills.
- Basic web skills: HTML, CSS skills, and JavaScript.
- Visual Studio Code and Jupyter Notebook skills.
- Basic Git and GitHub skills.

* These can all be learned in a semester and are generally useful in your career. As you improve these skills over time your ability to maintain research programs will be greatly improved.

Some General Thoughts

- The state of our system is observed as information defined by a set of measurements.
- All experiments are computational, i.e., are an algorithmic defined process of information processing on the system state. Some of the algorithms are run by the institution(s) (this is what we control), while others are run by computer agents (we still control) or subjects (we don't control).
- The move from the natural world to the theoretical model to the computational model to the experimental model to the statistical model introduces auxiliary hypotheses jointly being tested.

Why mTree?

- To study the impact of these auxiliary hypotheses, we need to study agent-based simulations and human-subject experiments in a consistent framework.
- mTree allows us to design a microeconomic system that we can use for simulations and with small changes to the code to run experiments.
- This framework also allows us to think more clearly about research programs* studying systematic changes in Environments, Organizations, Institutions, and Agents/Humans.

* Research programs will greatly speed up the research cycle explained next.

Research with mTree (one)

- Build your **economic narrative*.** Find a good question in the context of a microeconomic system. Understand the microeconomic and/or social choice systems in which you want to study your question. (Note, all experiments are economic systems whether you have thought it through or not.)
- Build your **theory narrative**. Formalize the microeconomic systems regarding economic states, performance measures, institutions, and organizations. This includes determining the logical consequences of your model stated as theoretical hypotheses.

* Note, all narratives should be written and dated in Jupyter notebooks maintained on GitHub. This allows you to follow an open-science model and maintain the integrity of your work.

Research with mTree (two)

- Start your **experimental narrative.** Make a **preliminary design** of your experimental task, control variables, treatments, and agent UI.
- Build your **computational narrative**. Build the microeconomic system as a computational system in mTree. Build the environment (state) and institution(s) (rules) for changing state based on participant messages.
- **Test** your agent-based model. Build your first cognitive agents who send messages to the system to change state. (This closes the computational model).
  - **test agents** validate your microeconomic system.
  - **theory agents** validate the theoretical predictions.

Research with mTree (three)

- Continue your **experimental narrative**. With a randomized design of your experiment.
  - Build **noisy theory agents** to generate simulation data.
  - Run **Monte-Carlo** simulations to choose control variables in your experiment that will give you the most **statistical power**.
  - Finalize your statistical hypotheses and statistical tests.
  - Code the **subject User Interface** to allow human subjects to take over the decisions made by the agent actors. The **UI** is coded in HTML and minimal JavaScript.
  - Write and test **instructions and procedures** for your experiment. Get human subject permission and preregister your experiment. Hopefully, you have acquired funding, reserved rooms, or set up online subject recruiting.

Research with mTree (four)

- Run experiments in the lab or online and collect and analyze your data. Build a **data narrative,** starting from raw data to statistical results.
- Improve your **data narrative** by running additional agent-based experiments to explain your data further.
  - Build behavioral agents (this may have happened in your theory agent building, but now you can further explore the cognitive strategies of these agents).
- **Write up** your paper using your different narratives. **Submit** your paper to working paper archives. **Present** your paper and adjust your narrative. **Publish** your paper.
- **Generate new hypotheses** for your research program by running simulations on an extended microeconomic system.

Bargaining Example Code Walkthrough

Bargaining Example

- A buyer and seller bargain over a price for one unit of a good.
- Buyer has a Value for the unit, and Seller has a Cost
- Buyer bids go to the bargaining institution
  - Bids between min offer and max offer
- Seller asks also go to the bargaining institution
  - Asks between min offer and max offer
- The standing bid and standing ask are the most current bid and ask
- When bid ≥ ask, a contract is formed
  - If ask caused the contract, price = standing bid
  - If bid caused the contract, price = standing ask

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagram
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

mTree Folder Structure

```
<DIR> economic_system_name
      <DIR> config
            config_name.json
      <DIR> logs
           <DIR> name_date_time_run
      <DIR> mes
            name_agent.py
            name_environment.py
           name_institution.py
      <DIR> notebooks
            name_design.ipynb
            name_config.ipynb
            name_analysis.ipynb
      <DIR> resources
            name_sequence.txt
            name_diagram.svg
      <DIR> ui
            subject.html
            connector.js
      README.md
            .gitignore
```

Control Variables Configurations

Data Folders

Microeconomic System Files

Notebook Docs and Utilities

Sequence diagrams

User Interface – Uses Flask

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagram
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

mes Folder

```
<DIR> mes
      barg_environment.py
      barg_institution.py
      barg_agent.py
```

mTree Imports

```
from mTree.microeconomic_system.environment import Environment
from mTree.microeconomic_system.institution import Institution
from mTree.microeconomic_system.agent import Agent
from mTree.microeconomic_system.directive_decorators import *
from mTree.microeconomic_system.message import Message
```

Classes in .py files

```
@directive_enabled_class
class BargEnvironment(Environment):

@directive_enabled_class
class BargInstitution(Institution):

@directive_enabled_class
class BargAgent(Agent):
```

Add Actor Capabilities

Inherit Environment

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagram
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

Config file: config_name.json

```
{"mtree_type": "mes_simulation_description",
  "name":"bargain",
  "id": "1",
  "environment": "barg_environment.BargEnvironment",
  "institutions": [{"institution": "barg_institution.BargInstitution"}],
  "number_of_runs": 1,
  "agents": [{"agent_name": "barg_agent.BargAgent", "number": 2}],
  "properties": {"endowment": 1000,
                 "value": 800,
                 "cost": 300,
                 "starting_bid": 0,
                 "starting_ask": 1000,
                 "period_length":15,
                 "offer_wait_time": 3,
                 "number_of_rounds": 1,
                 "number_of_agents": 2,
                 “agent_types”: [“Seller”, “Buyer”],
                 "escrow": 0},
  "data_logging": false
}
```

Config file: config_name.json

```
{
    "mtree_type": "mes_simulation_description",
    "name": "bargain",
    "id": "1",
    "description": null,
    "environment": "barg_environment.BargEnvironment",
    "institutions": [
        {
            "institution_class": "barg_institution.BargInstitution",
                "short_name": "barg_institution"
        }
    ],
    "number_of_runs": 1,
    "agents": [
        {
            "agent_name": "barg_agent.BargAgent",
            "number": 2
        }
    ],
```

Short Name

Overridden Short Name

Number of agents

Short Names =
barg_agent.BargAgent 1
barg_agent.BargAgent 2

Config file: config_name.json (Continued)

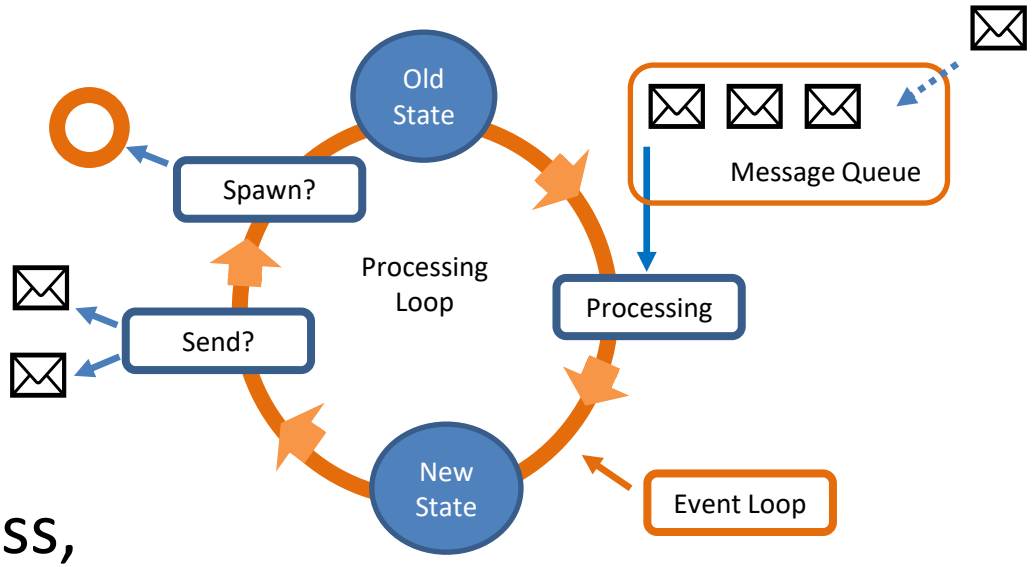
Control variables that can be set

```
 ],
 "properties": {
     "endowment": 1000,
     "value": 800,
     "cost": 300,
     "starting_bid": 0,
     "starting_ask": 1000,
     "period_length": 15,
     "offer_wait_time": 3,
     "number_of_rounds": 1,
     "number_of_agents": 2,
     "escrow": 0
 },
 "data_logging": false
}
```

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagram
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

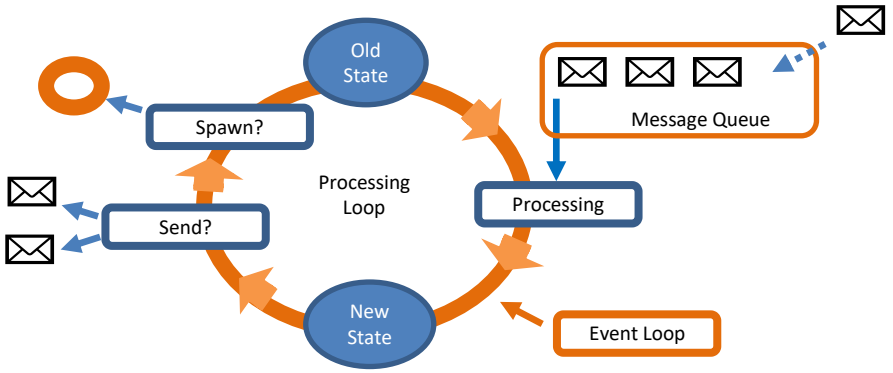
Message



✉ = (directive,
sender_address,
receiver_address,
payload)

Messages are handled by message handlers in the actor code. An example comes later. What is important to notice is that the next message in the queue is processed only after the message handler returns, i.e., finishes processing.

Message Space



✉ = (directive,
sender_address,
receiver_address,
payload)

In Python we can think of a message space as a dictionary of dictionaries that look like this:

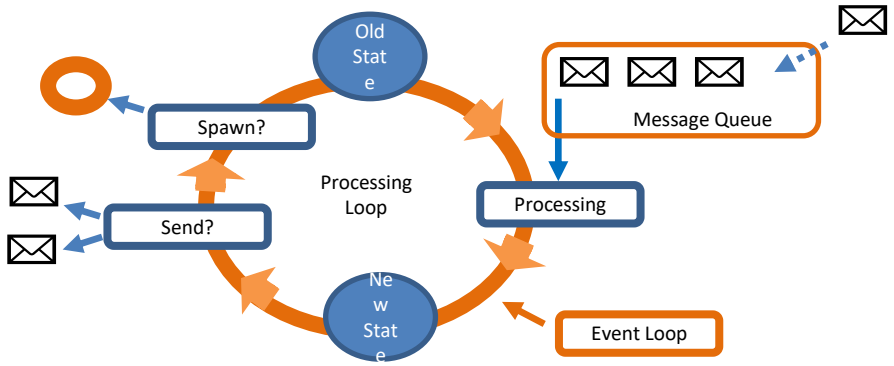
payload_2

{“1st directive”: payload_1,
“2nd directive”: payload_2,
…
…
…
“last directive”: payload_L}

→

{“key_1”: value_type,
“key_2”: value_type,
…
…
…
“Key_m”: value_type}

Message Space



{“1st directive”: payload_1,
“2nd directive”: payload_2,
…
…
…
“last directive”: payload_L}

payload_2

{“key_1”: value_type,
“key_2”: value_type,
…
…
…
“Key_m”: value_type}

Examples of key, value_type

“agent”: short_name

“bid”: integer

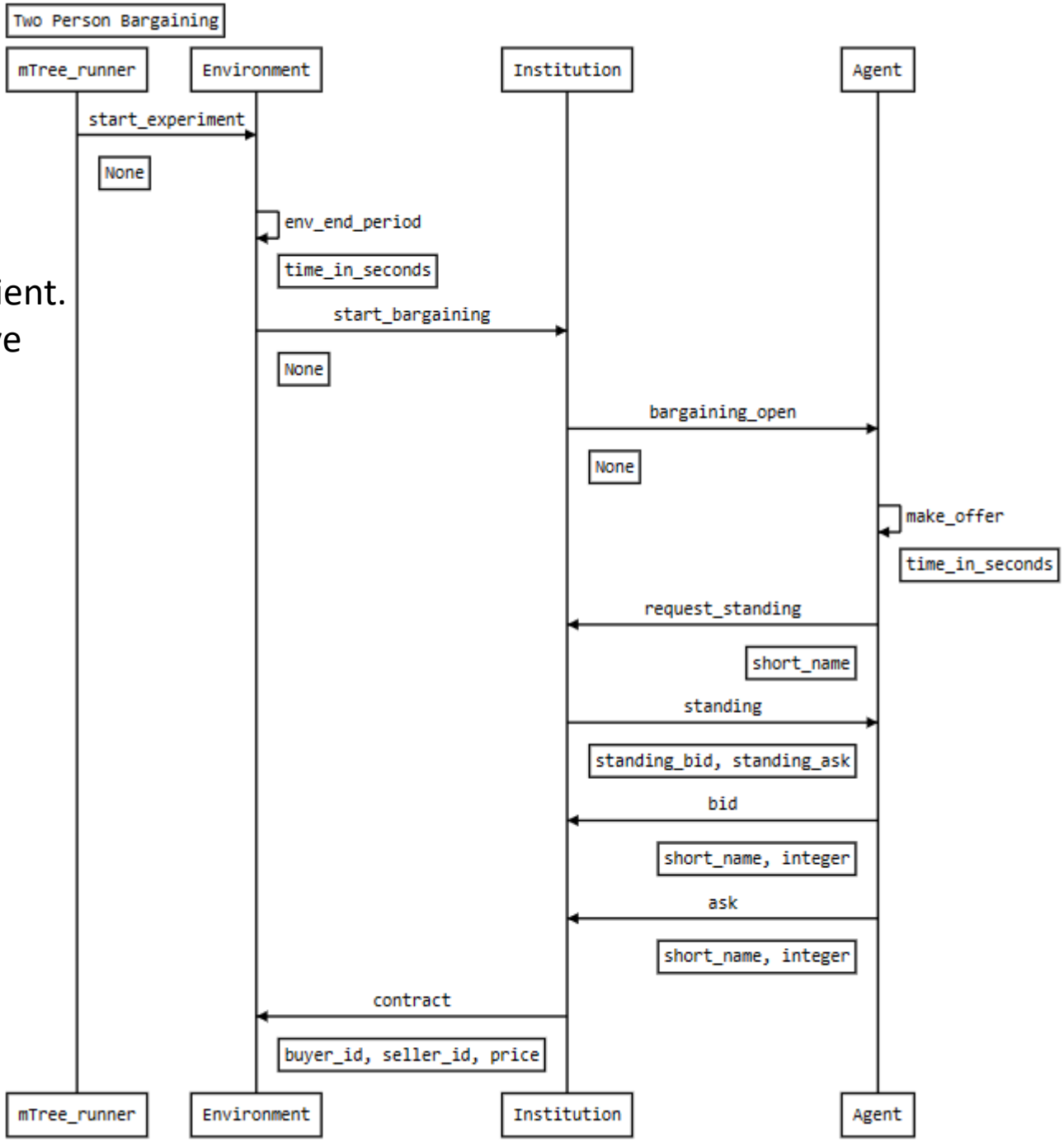
Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagrams
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

Sequence Diagram

[js-sequence-diagrams]

Arrows show sender and recipient.
Above the arrow is the directive
Below the arrow is payload.



Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagrams
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

File Structure for Agent (typical of all)

```
Imports here
@directive_enabled_class
class BargAgent(Agent):
    def prepare(self):
    @directive_decorator("bargaining_open")
    def bargaining_open(self, message: Message):
    @directive_decorator("standing")
    def standing(self, message: Message):
    @directive_decorator("contracts")
    def contracts(self, message: Message):
    @directive_decorator("end_round")
    def end_round(self, message: Message):
    @directive_decorator("make_offer")
    def make_offer(self, message: Message):
    def wait_offer(self):
    def make_bid(self):
    def make_ask(self):
```

Setup Method

Message Handler

Timing Loop for Decisions

Decision Cognition

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagrams
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

Agent prepare()

```
CONFIG -"properties": {"endowment": 1000,
                       "value": 800,
                       "cost": 300,
                       "starting_bid": 0,
                       "starting_ask": 1000,
                       "period_length":15,
                       "offer_wait_time": 3,
                       "number_of_rounds": 1,
                       "number_of_agents": 2,
                       "escrow": 0},
```

```
def prepare(self):
    """
    setup agent as buyer or seller
    """
     self.my_id = int(self.short_name[-1])
     if self.my_id == 1:
         self.role = 'Buyer'
         self.value = int(self.get_property("value"))
     else:
         self.role = 'Seller'
         self.cost = int(self.get_property("cost"))

     self.offer_wait_time = int(self.get_property("offer_wait_time")
     self.barg_open = False
```

self.short name is short_name of actor

Get property from config file properties: section

Agent
bargaining_open()

Message directive from institution

Message from institution

```
@directive_decorator("bargaining_open")
def bargaining_open(self, message: Message):
    """
    Informs agents that bargaining has started.
    """
    self.barg_open = True
    self.send_message("request_standing", "barg_institution",
                      self.short_name)
```

Request standing bid and ask from institution

```
self.send_message("request_standing",
                  "barg_institution",
                  self.short_name)
```

Directive

Recipient short Name

Payload

Agent standing() and make_bid()

```
@directive_decorator("standing")
def standing(self, message: Message):
    self.standing_bid, self.standing_ask = message.get_payload()
    if self.role == "Buyer":
        self.make_bid()
```

```
def make_bid(self):
    bid = random.randint(10, self.value)
    payload = {'bid': bid, 'short_name': self.short_name}
    self.send_message("bid", "barg_institution", payload)
    self.wait_offer()
```

Institution: bid()

message object from Message class

get sender address from message

get payload from message

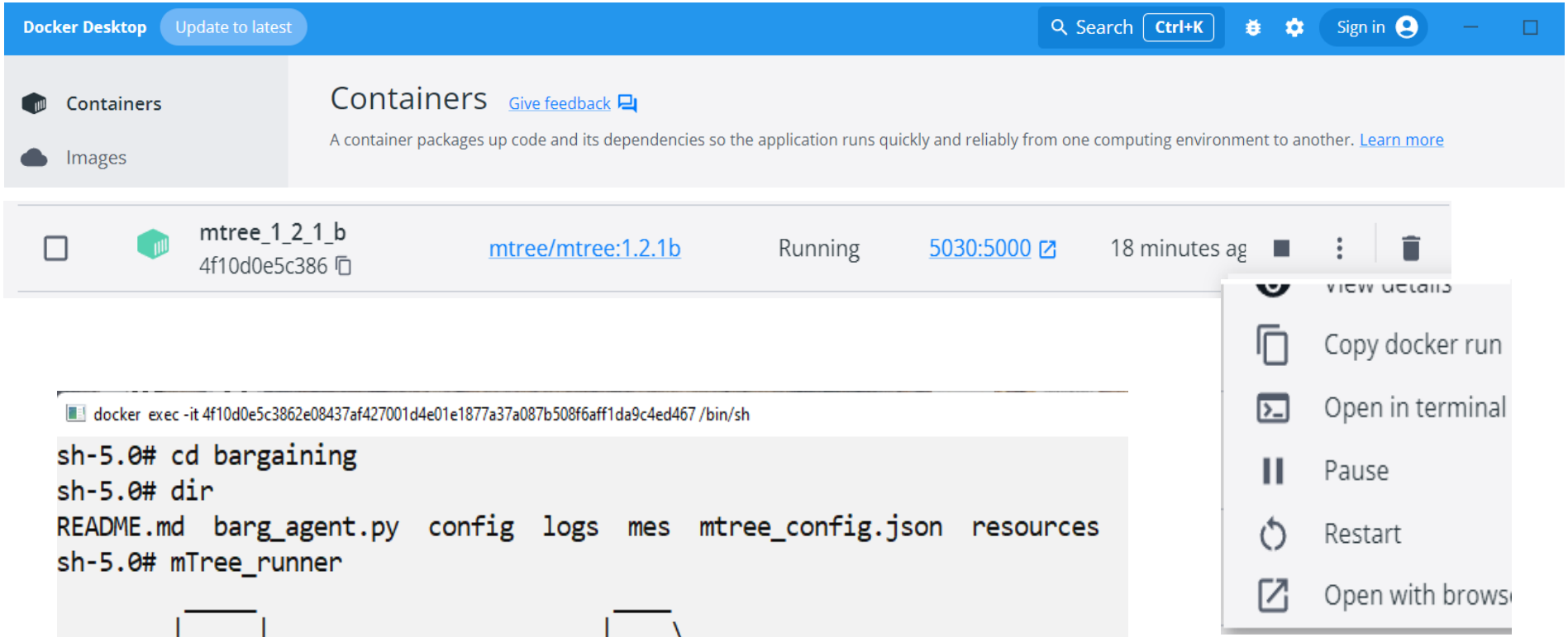
send data to data log

```
@directive_decorator("bid")
def bid(self, message: Message):
    if self.barg_open == False: return
    self.agent_address = message.get_sender()
    payload = message.get_payload()
    self.agent_bid = payload['bid']
    self.agent_id = payload['short_name']
    self.log_data(f"id = {self.agent_id}, bid = {self.agent_bid}")
    if self.agent_bid >= self.standing_ask:
        self.barg_open = False
        contract = (self.agent_id, self.standing_ask_id,
                    self.standing_ask)
        self.log_data(f"contract = {contract}")
        self.send_message("contract",
                          "barg_environment.BargEnvironment",
                           contract)
    elif self.agent_bid < self.standing_ask:
        self.standing_bid = self.agent_bid
        self.standing_bid_id = self.agent_id
```

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagrams
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

Running mTree



docker exec -it 4f10d0e5c3862e08437af427001d4e01e1877a37a087b508f6aff1da9c4ed467 /bin/sh

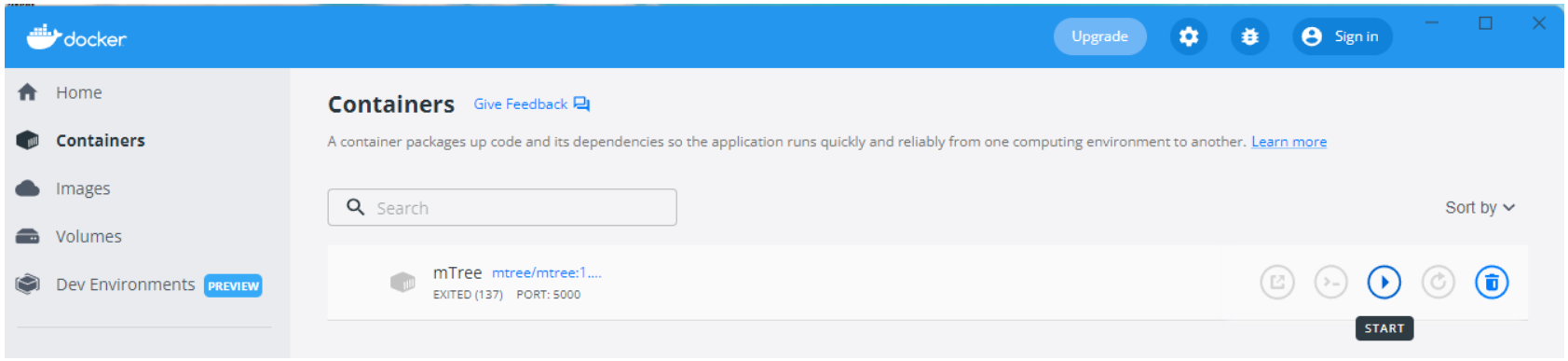
```
sh-5.0# cd bargaining
sh-5.0# dir
README.md  barg_agent.py  config  logs  mes  mtree_config.json  resources
sh-5.0# mTree_runner      Start mTree
           _____                      ____
 _ __ ___ |_   _| __ ___  ___        |  _ \ _   _ _ __  _ __   ___ _ __
| '_ ` _ \  | || '__/ _ \/ _ \ _____ | |_) | | | | '_ \| '_ \ / _ \ '__|
| | | | | | | || | |  __/  __/|_____||  _ <| |_| | | | | | | |  __/ |
|_| |_| |_| |_||_|  \___|\___|       |_| \_\\__,_|_| |_|_| |_|\___|_|


Starting prompt...
mTree> ?      help

Documented commands (type help <topic>):
========================================
check_status  force_shutdown  hello  help  kill_run  quit  run_simulation

mTree> run_simulation      Run simulation
===> Starting to run:  test_5_config.json
mTree> check_status      Check status of running simulations
STATUS REPORTING
[['a957a2', 'bargain', 1, 'Running', '0:00:14.005650']]
+----------+---------------+------------+---------+----------------+
| Run Code | Configuration | Run Number | Status  | Total Time     |
+----------+---------------+------------+---------+----------------+
| a957a2   | bargain       | 1          | Running | 0:00:14.005650 |
+----------+---------------+------------+---------+----------------+
```

Running mTree




```
docker exec -it bcfc94789e41864d05a7e1004dabd8aefddc6535c7a792b7575ef9faa6494afe /bin/sh
sh-5.0# dir
README.md  config  logs  mes  notebooks  resources  system.log  thespian.log
sh-5.0# mTree_runner
```

```
 _____                                  ____
 _ __ __|_    _| __ ___  ___           |  _ \ _   _ _ __  _ __   ___ _ __
| '_ ` _ \| || '__/ _ \/ _ \  _____   | |_) | | | | '_ \| '_ \ / _ \ '__|
| | | | | | || | |  __/  __/ |_____|  |  _ <| |_| | | | | | | |  __/ |
|_| |_| |_|_||_|  \___|\___|          |_| \_\\__,_|_| |_|_| |_|\___|_|
```

```
Starting prompt...
mTree> ?

Documented commands (type help <topic>):
========================================
check_status  force_shutdown  hello  help  kill_run  quit  run_simulation

mTree>
```

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagrams
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

Data file .dat

time-stamp

```
1673306001.841825       "id = barg_agent.BargAgent 2, ask = 661"
1673306001.8550704      "id = barg_agent.BargAgent 1, bid = 253"
1673306005.8638859      "id = barg_agent.BargAgent 2, ask = 812"
1673306005.868271       "id = barg_agent.BargAgent 1, bid = 331"
1673306009.8864255      "id = barg_agent.BargAgent 2, ask = 317"
1673306009.8865702      "contract = ('barg_agent.BargAgent 1',
                                    'barg_agent.BargAgent 2', 331)"
```

Designing a mTree project

- mTree Folder Structure
- MES Folder
- Config Folder and files
- Messages and Message Space
- Sequence Diagrams
- File Structure
- mTree methods
- Running mTree
- Data Files
- Log Files

log file .log

time-stamp

mTree logging

```
1673306001.8054824   Institution (barg_institution) :
                     sending to  directive: bargaining_open
1673306001.8059318   Institution (barg_institution) :
                     sending to  directive: bargaining_open
1673306001.812985    ****<A> myid = 2
1673306001.8145661   ****<A> myid = 1
1673306001.8252928   ***<A>*** Seller: Initialized
1673306001.8264716   Agent (barg_agent.BargAgent 2) :
                     About to enter directive: bargaining_open
1673306001.832151    Agent (barg_agent.BargAgent 2:
                     Exited directive: bargaining_open
1673306001.8324857   ***<A>*** Buyer: Initialized
1673306001.832565    Institution (barg_institution) :
                     About to enter directive: request_standing
```

mes program logging*

*in code `self.log_message(f'****<A> myid = {self.my_id}')`

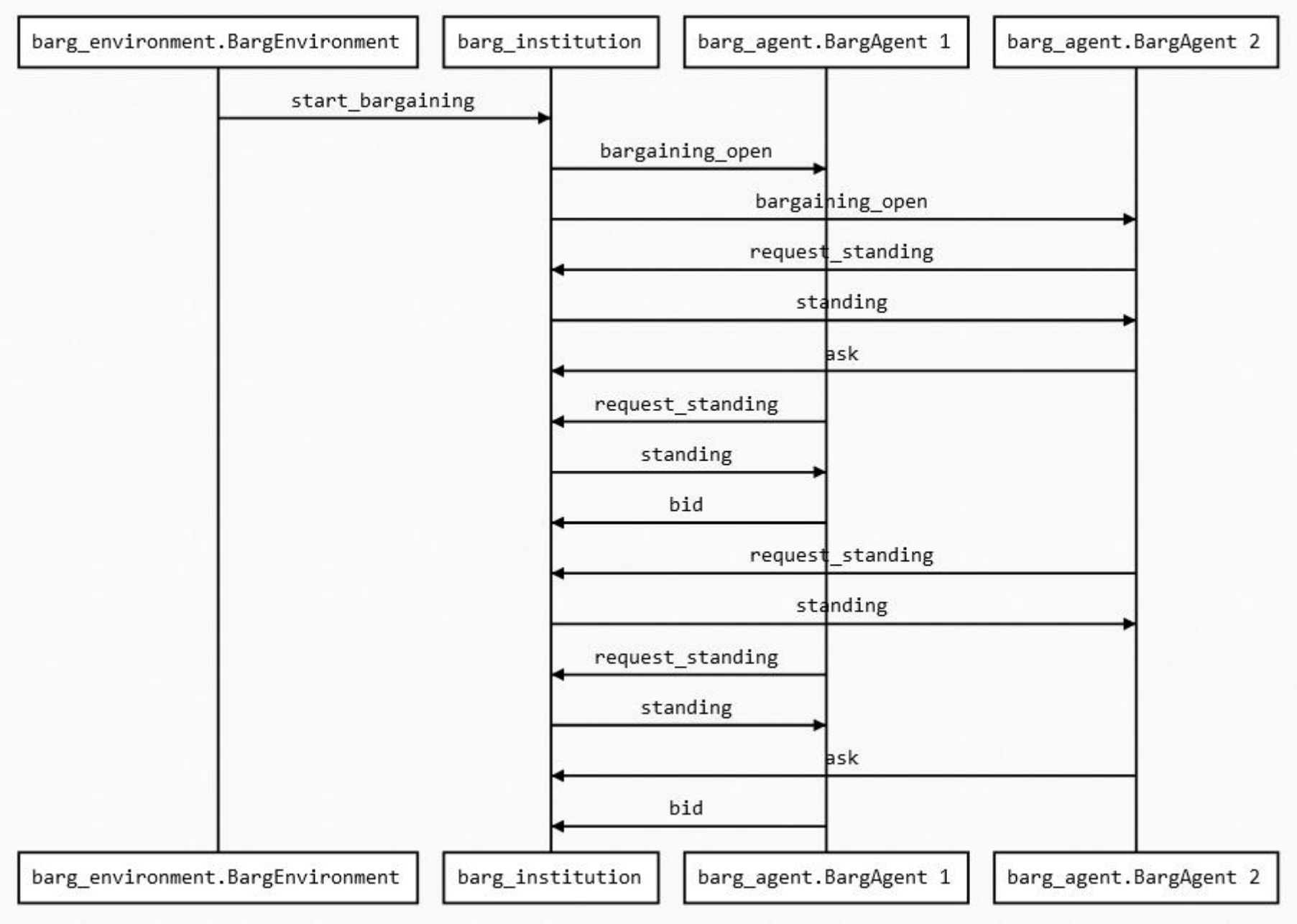
Sequence log

```
barg_environment.BargEnvironment->barg_institution: start_bargaining
barg_institution->barg_agent.BargAgent 1: bargaining_open
barg_institution->barg_agent.BargAgent 2: bargaining_open
barg_agent.BargAgent 2->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 2: standing
barg_agent.BargAgent 2->barg_institution: ask
barg_agent.BargAgent 1->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 1: standing
barg_agent.BargAgent 1->barg_institution: bid
barg_agent.BargAgent 2->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 2: standing
barg_agent.BargAgent 1->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 1: standing
barg_agent.BargAgent 2->barg_institution: ask
barg_agent.BargAgent 1->barg_institution: bid
```

Sequence log in js-diagram-maker

```
barg_environment.BargEnvironment->barg_institution: start_bargaining
barg_institution->barg_agent.BargAgent 1: bargaining_open
barg_institution->barg_agent.BargAgent 2: bargaining_open
barg_agent.BargAgent 2->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 2: standing
barg_agent.BargAgent 2->barg_institution: ask
barg_agent.BargAgent 1->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 1: standing
barg_agent.BargAgent 1->barg_institution: bid
barg_agent.BargAgent 2->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 2: standing
barg_agent.BargAgent 1->barg_institution: request_standing
barg_institution->barg_agent.BargAgent 1: standing
barg_agent.BargAgent 2->barg_institution: ask
barg_agent.BargAgent 1->barg_institution: bid
```

Sequence log in js-diagram-maker



Thank You


Wavy Walsh
Cid MacMoragh
Freddy Fugazi
Touch me for Vecon Lab
Kevin McCabe
HeatherLeigh Jiminy
ProfessorSL String

Actor Decision Making

$v(d): M_d \times S \rightarrow \{Accept, Reject\}$

Actor attention to incoming messages is determined by the validity rules, v.

$w(d): M_d \times S \rightarrow S$

Actor knowledge is embodied in the state variable, s_t, and is updated using the processing rules, w, on incoming messages.

$x(d): M_d \times S \rightarrow M$

$y(d): M_d \times S \rightarrow A$

Actor action is both outgoing messages, determined by the message rules, x, and the agent spawned rules determined by y.

Spawned actors act independently, but in general, we think of them as surrogates of the agent who spawned them. Principal-agent problems are eliminated by giving the surrogate identical preferences as the principal.

Models of Agent Cognition

Rational: Agent Choices are Constrained Optimal
Nash Equilibrium and Refinements
Optimal Learning
e.g., Repeated Game Cooperation

Cognitive: Agent Choices are Goal Directed
Limited Perception and Attention
Memory and Foresight
Secondary Goal Formation
Motor Responses
Costly Cognitive Operations
Evolved Cognitive Algorithms
Learned Cognitive Algorithms
e.g., Reinforcement Learning

Minimal Rationality: Agent Choices are random (e.g., ZI)
with Some updating of the choice domain (e.g., BC-ZI)