
Programming for Social Scientists

Introduction

Johan A. Dornschneider-Elkink



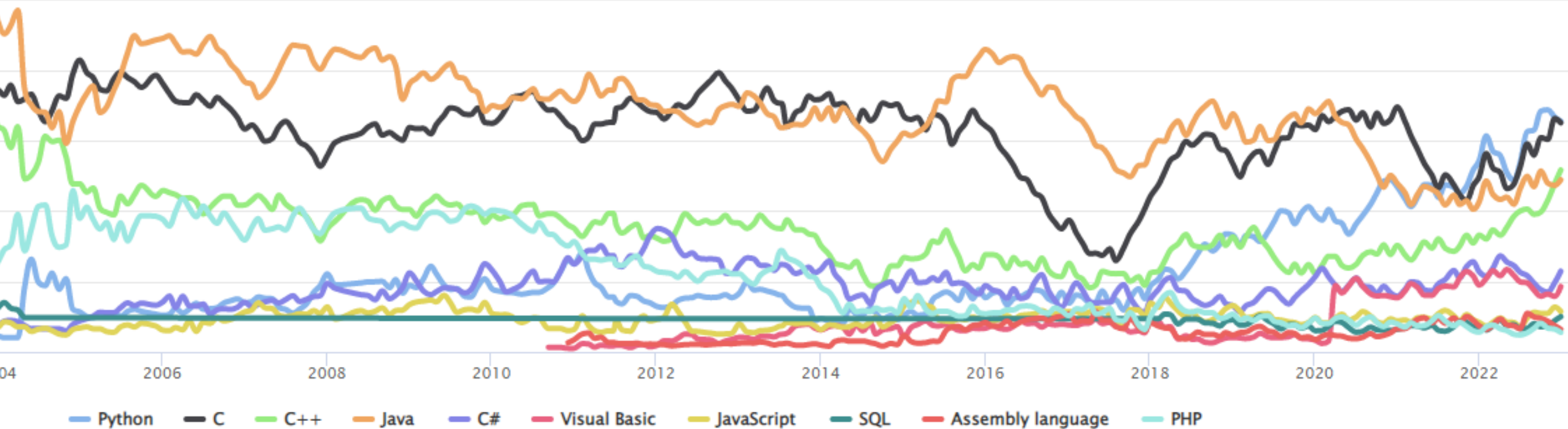
python

```
7 from watson.framework import events
8 from watson.http.messages import Response, Request
9 from watson.common.imports import get_qualified_name
10 from watson.common.contextmanagers import suppress
11
12
13 ACCEPTABLE_RETURN_TYPES = (str, int, float, bool)
14
15 class Base(ContinueAware, metaclass=abc.ABCMeta):
16     """The base class for all controllers.
17
18     Attributes:
19         __action__ (string): The last action that was called
20
21     """
22     def execute(self, **kwargs):
23         method = self.get_execute_method(**kwargs)
24         self.__action__ = method
25         return method(**kwargs) or {}
26
27 @abc.abstractmethod
28 def execute_method(self, **kwargs):
29     """Execute the method"""
```

TIOBE Programming Community Index

Source: www.tiobe.com

(R is nr. 13 as of January 2023)



The TIOBE Programming Community index is an indicator of the **popularity** of programming languages. The index is updated once a month. The ratings are based on the **number of skilled engineers** world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is **not about the best programming language** or the language in which *most lines of code* have been written.

Python

Scripting vs. programming

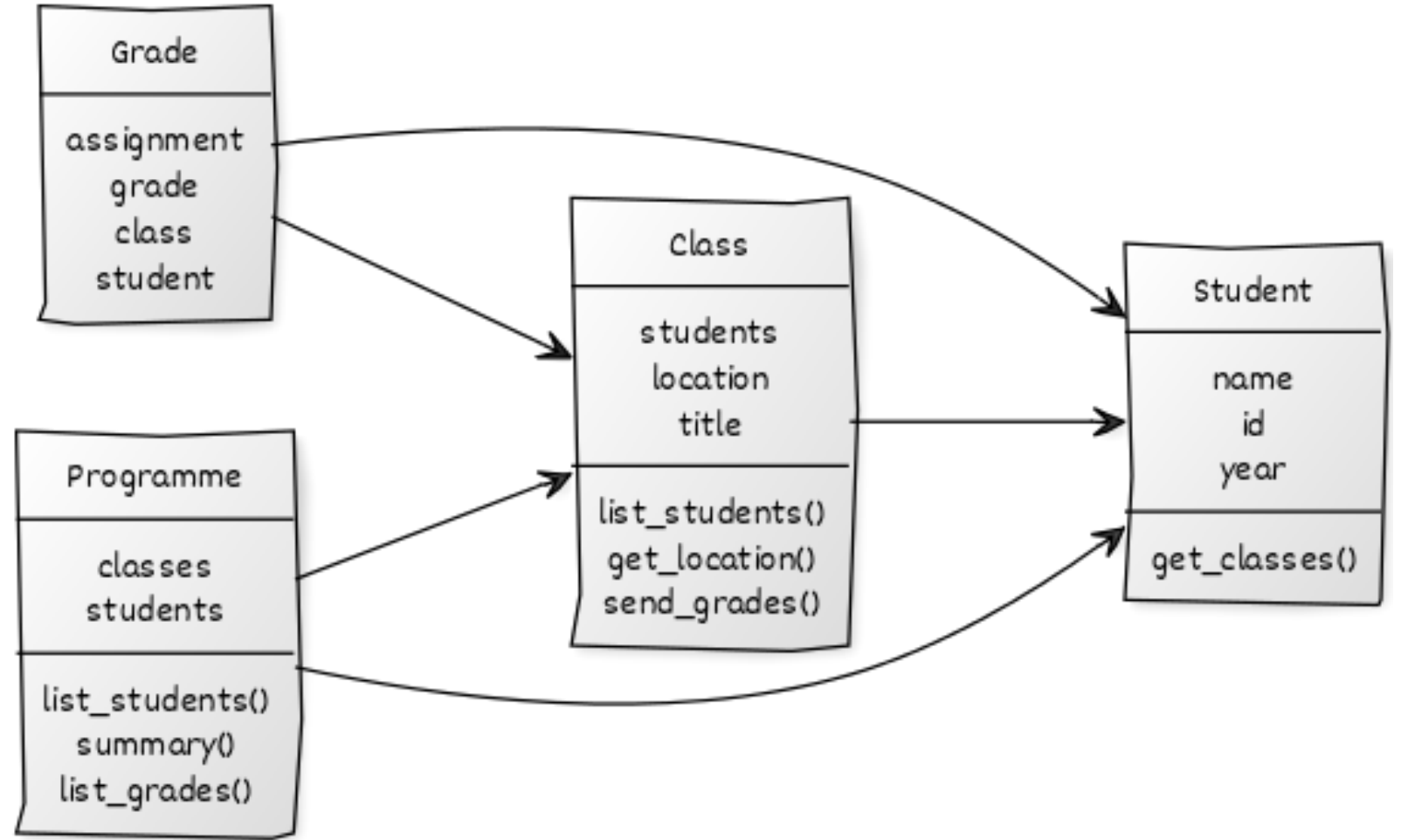
R vs. python

Applications

Object-oriented design



Object-oriented design



Tools


Slack

Github / git

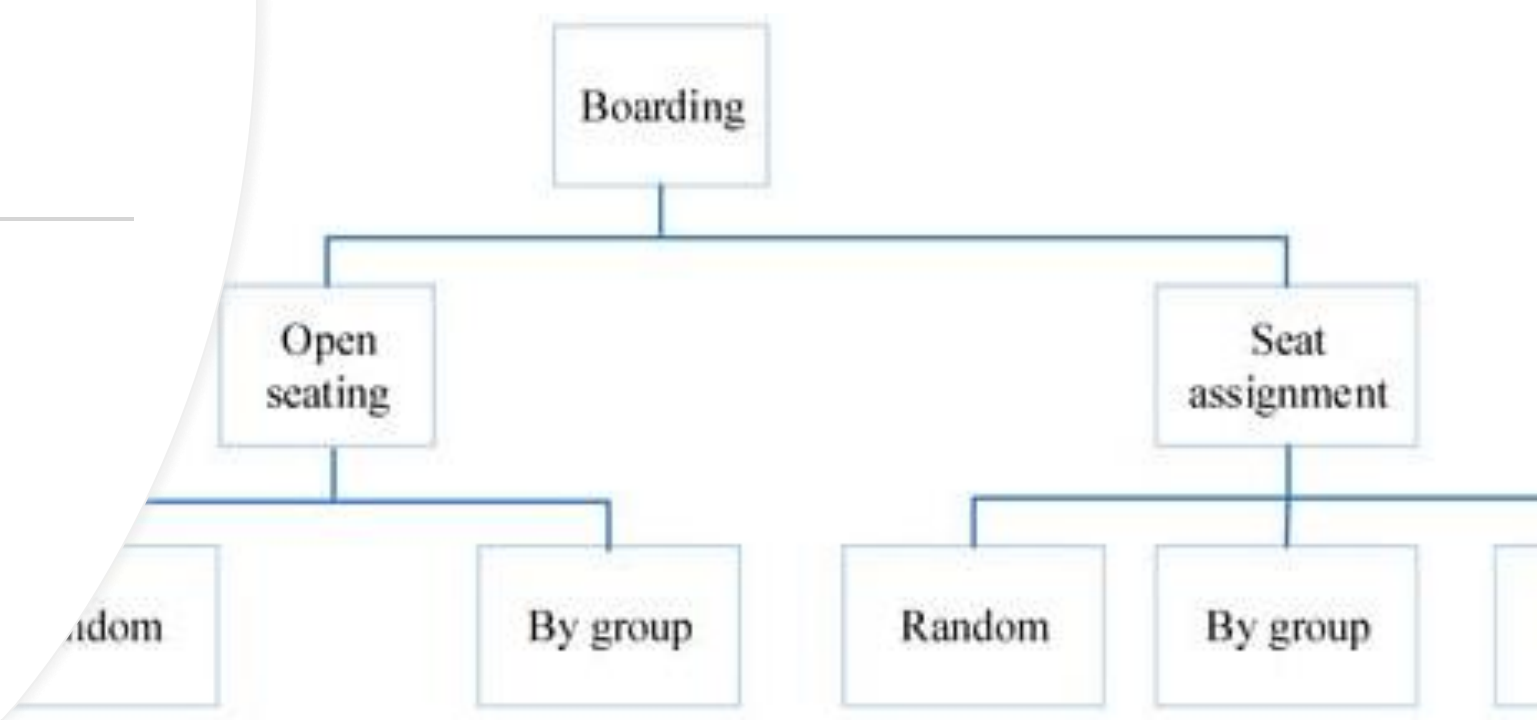
Repl.it

Local installation ?





Agent-based simulation

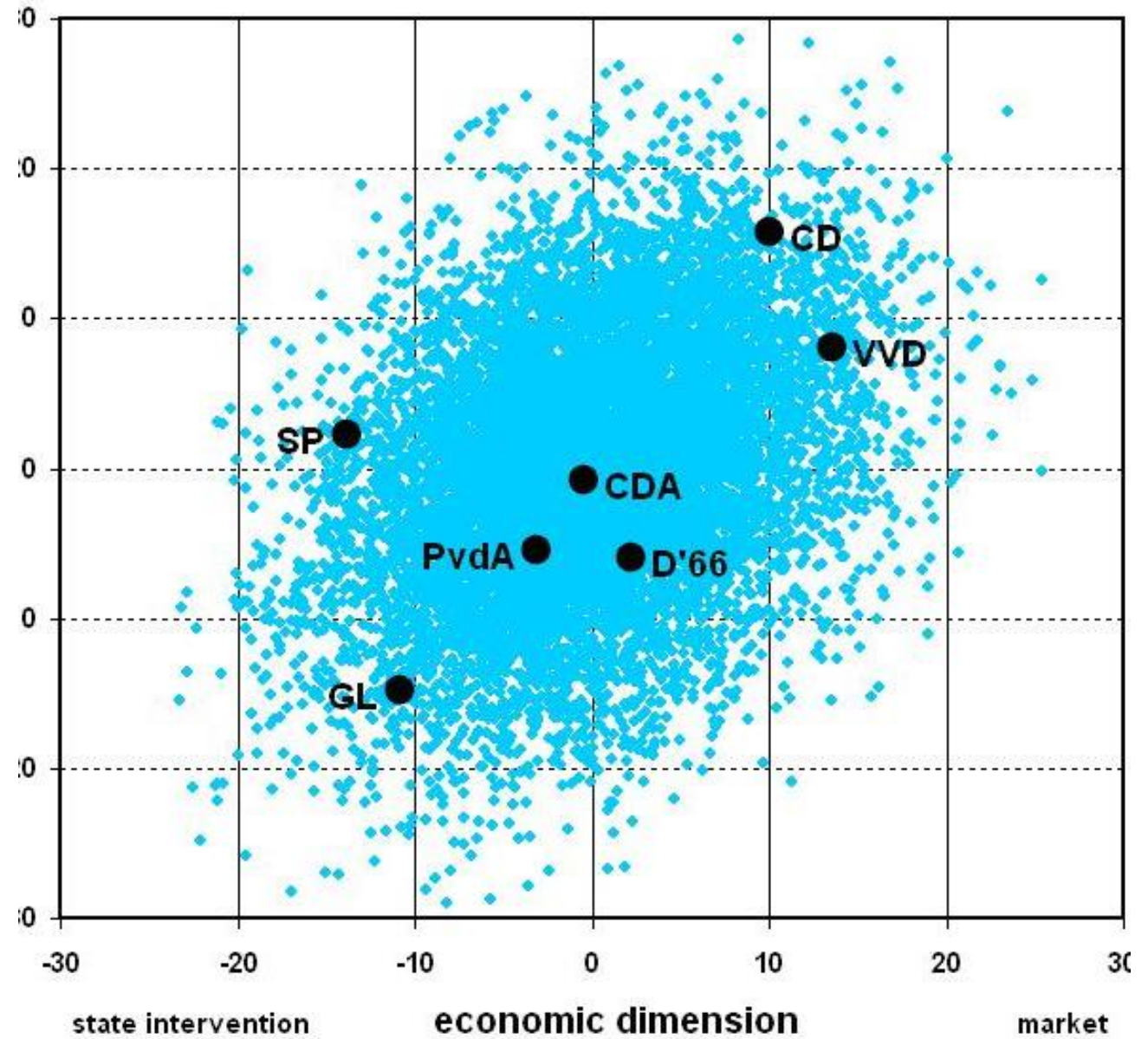


Policy and the Dynamics of Political Competition

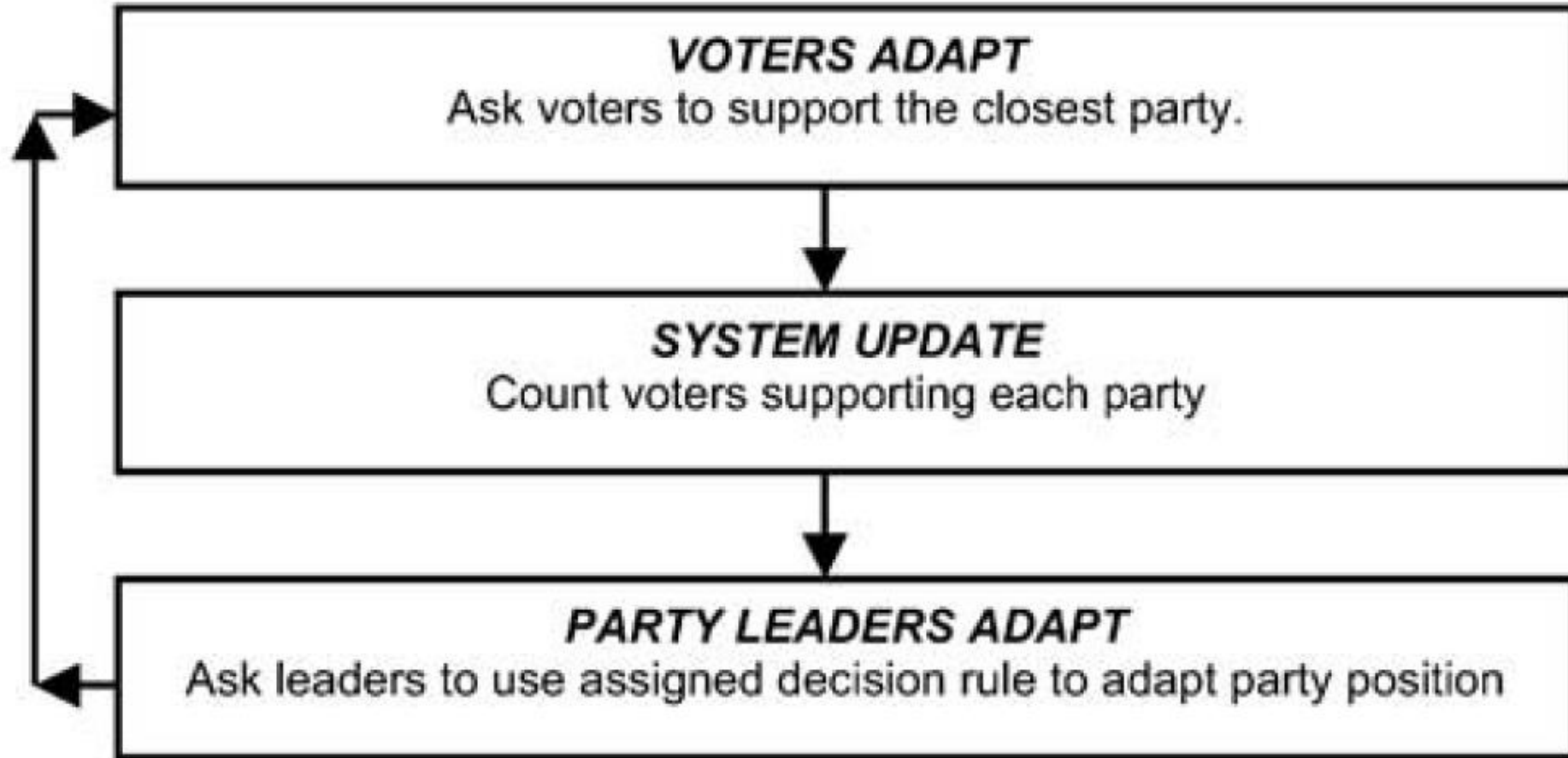
MICHAEL LAVER *New York University*

This paper proposes a model that takes the dynamic agent-based analysis of policy-driven party competition into a multiparty environment. In this, voters continually review party support and switch parties to increase their expectations; parties continually readapt policy positions to the shifting affiliations of voters. Different algorithms for party adaptation are explored, including “Aggregator” (adapt party policy to the ideal policy positions of party supporters), Hunter (repeat policy moves that were rewarded; otherwise make random moves), Predator (move party policy toward the policy position of the largest party), and “Sticker” (never change party policy). Strong trends in the behavior of parties using different methods of adaptation are explored. The model is then applied in a series of experiments to the dynamics of a real party system, described in a published opinion poll time series. This paper reports first steps toward endogenizing key features of the process, including the birth and death of parties, internal party decision rules, and voter ideal points.

Spatial
model of
voting



Sequence



Party strategies

ADAPTIVE DECISION RULES

AGGREGATOR

Go to mean position of current party supporters on each dimension.

HUNTER

Was previous move followed by increased party support? If yes, repeat move. If no, turn 180° from direction of last move, make unit move in direction randomly selected from arc 90° either side of direction now faced.

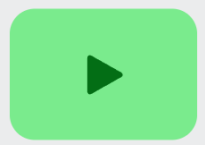
PREDATOR

Observe party sizes. If you are the largest party, stand still. If not the not largest party, set heading towards largest party, make unit move.

STICKER

Never change policy position.

MCQ test 1	23 Feb	20%
MCQ test 2	5 Apr	20%
MCQ test 3	26 Apr	20%
Class diagram	26 Mar, 1 pm	10%
Lab report	7 May, 5 pm	30%



Q Search

Files

- main.py
- .gitignore
- LICENSE

Tools

CPU RAM Storage

? Help

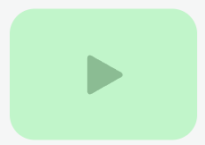
main.py

```
main.py
1
2 def print_insult():
3     print("You're an idiot!")
4
5     print_insult()
```

Line 3 : Col 25 History

_ Console Shell

```
You're an idiot!
> print("You too!")
You too!
> []
```



Search

Files

- main.py
- .gitignore
- LICENSE

Tools

CPU RAM Storage

Help

main.py

```

1
2 def print_insult():
3     print("You're an idiot!")
4
5     print_insult()

```

Programme

Line 3 : Col 25 History

Console Shell

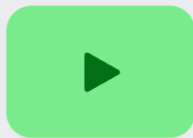
```

You're an idiot!
> print("You too!")
You too!
>

```

REPL

Read
Eval
Print
Loop



Search

Files [Add] [Folder] [More]

Tools



Docs Chat Threads



Packages Git Debugger



Shell Console Secrets

CPU RAM Storage [Progress Bars]

Help

main.py x + [Menu] [More]

main.py

```
1
2 def print_insult():
3     print("You're an idiot!")
4
5     print_insult()
```

Line 3 : Col 25

History [Refresh]

Git x Shell x + [More]

Version control

jelkink/ucd-prog-2023 [Trash]

up to date with main

main [Dropdown] +

What did you change?

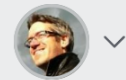
Commit All & Push → [Dropdown]

3 staged files or folders

Send feedback



Invite



Search

Files



Tools



Docs



Chat



Threads



Packages



Git



Debugger



Shell



Console



Secrets

CPU

RAM

Storage

Help

main.py

main.py

```
1
2 def print_insult():
3     print("You're an idiot!")
4
5 print_insult()
```

Line 3 : Col 25

History

Git Shell

Version control

jelkink/ucd-prog-2023

up to date with main

main

What did you change?

Commit All & Push

3 staged files or folders

Send feedback

Getting used to Python

```
print("Hello, World!")
```

Hello, World!

Using Python as calculator

```
5 + 30 * 3
```

95

```
2 ** 3
```

8

```
5 / 2
```

2.5

```
5.0 / 2
```

2.5

```
5 // 2
```

2

```
5.0 // 2
```

2.0

Work on Lab 1

Working with your
neighbours is a good
idea!