# Program Analysis

# Introduction of Course Project

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2023/2024

# Goal

**Design and implement dynamic slicing**

- Input:
  - Executable program with all inputs
  - Slicing criterion
- Output:
  - Reduced program that yields same behavior w.r.t. slicing criterion (for same input)

# Example

```
def slice_me(n):
  x = n + 1;
  if x == 5:
    print("hey")
  else:
    print("ho")

  print("brrr")

slice_me(5)
```

# Example

```
def slice_me(n):
  x = n + 1;
  if x == 5:
    print("hey")
  else:
    print("ho")

  print("brrr")

slice_me(5)
```

Slicing criterion

# Example

```
def slice_me(n):
  x = n + 1;
  if x == 5:
    print("hey")
  else:
    print("ho")

  print("brrr")

slice_me(5)
```

Slicing criterion

```
def sliceMe(n):
  x = n + 1
  if x == 5:
    pass
  else:
    print("ho")

slice_me(5)
```

# Example

```
def slice_me(n):
  x = n + 1;
  if x == 5:
    print("hey")
  else:
    print("ho")

  print("brrr")

slice_me(5)
```

Slicing criterion

# Example

```
def slice_me(n):
  x = n + 1;
  if x == 5:
    print("hey")
  else:
    print("ho")

  print("brrr")

slice_me(5)
```

Slicing criterion

```
def slice_me(n):
  print("brrr")

slice_me(5)
```

# Slicing Algorithms

**Different algorithms differ in**

- Precision: How small does the slice get?
- Efficiency: How long does the slicing take?
- Conceptual complexity

**Objective: Smallest possible slice (i.e., as precise as possible), but still sound**

- Soundness: All statements included to preserve behavior w.r.t. slicing criterion

# Assumptions

## Kind of programs to consider

- Single function
- Single file: Defines the function and then calls it
- Slice should always keep all arguments to the sliced function (even if unused)
- No definitions of classes or other functions in the sliced function

# Assumptions (2)

## Subset of Python to consider

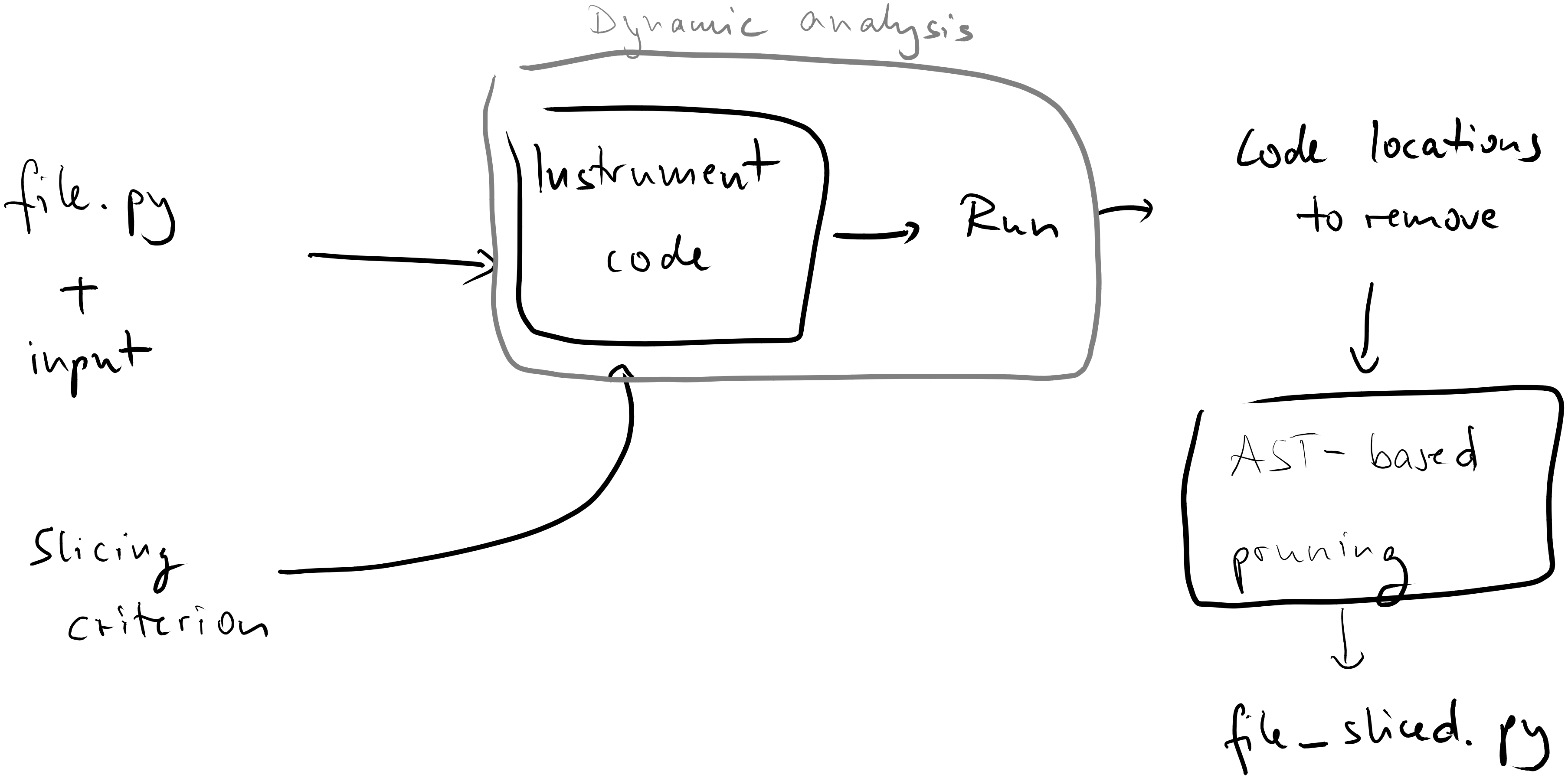
- Language features until Python 3.10
- No calls to `eval` or `exec`
- No `with` statements
- Left-hand side of assignments: Single variable, attribute, or index access

# Assumptions (3)

## Intra-procedural analysis

- Analysis considers only one function
- Calls to other functions are possible:
  - Callee code not analyzed
  - Assume data flows:
    - From arguments to return value
    - From base object to return value
    - From arguments to base object

# Overview of Analysis



# Dynamic Analysis

- **Based on DynaPyt framework**
- **Hooks/callbacks for different kinds of runtime events, e.g.,**
  - variable reads/writes
  - binary expressions
  - conditionals
- **Based on source-to-source instrumentation**

# DynaPyt Demo

**[simple Python code, single-hook analysis, instrumented code, output of running the analysis on the code]**

# Tips on DynaPyt

- Rich framework that provides more than what you need
- Work through the tutorial to understand the basics
- Check out example analyses under *src/dynapyt/analyses*

# Implementing Slicing

- **Track data-flow and control-flow dependencies at runtime**
  - □ Data flow: Whenever a new value gets computed, track dependency from inputs
  - □ Control flow: Whenever a control flow decision is made, track what it depends on

# Location Information

- Every runtime event happens at some code location
- IID = unique identifier of location in original program (i.e., before instrumentation)
- Use it to determine which code is needed in the slice

# Example: IIDs

[demo of IIDs; how to obtain, how to resolve, what they contain (line, column)]

# AST-based Pruning of Code

- **Once locations to keep are known:**
  - Prune away remaining code
- **Implement it via AST transformation**
  - Parse
  - Manipulate
  - Pretty-print

# Demo

**[show code in syntax_tree_manipulation, run if trom Python console on a simple example]**

**[show printed ast]**

# Project Milestones

- **Milestone 1**
  - □ Simple DynaPyt analysis
  - □ AST manipulation
- **Milestone 2**
  - □ Slicing w.r.t. data-flow only
- **Milestone 3**
  - □ Slicing w.r.t. control-flow and data-flow

# Milestone 1: Simple DynaPyt Analysis

- **Goal: Prints values of variable writes**
  - □ Actual goal: Get familiar with DynaPyt
- **Example:**

```
y = 0
x = 23
if x > 5:
  y = x - 3

print(y)
```

# Milestone 1: Simple DynaPyt Analysis

- **Goal: Prints values of variable writes**
  - □ Actual goal: Get familiar with DynaPyt
- **Example:**

```python
y = 0
x = 23
if x > 5:
  y = x - 3

print(y)
```

```
0
23
20
```

# Milestone 1: AST Manipulation

- Input: Code, line numbers
- Output: Subset of code
- Example:

```
# lines to keep: 2, 3, 5
print("hello")
y = 0
x = 23
if x > 5:
  y = x - 3

print(y)
```

# Milestone 1: AST Manipulation

- Input: Code, line numbers
- Output: Subset of code
- Example:

```
# lines to keep: 2, 3, 5
print("hello")
y = 0
x = 23
if x > 5:
  y = x - 3

print(y)
```

→

```
y = 0
x = 23
y = x - 3
```

# Milestone 2

- Slicing based on data flow only
- Assume: Straightline code without control flow
- Example:

```
x = 0
y = 0
x = 23
z = 5
y = x - 3
z = x  + 1
z = y * 3
```

# Milestone 2

- Slicing based on data flow only
- Assume: Straightline code without control flow
- Example:

```
x = 0
y = 0
x = 23
z = 5
y = x - 3
z = x   + 1
z = y * 3
```

Slicing criterion

# Milestone 2

- Slicing based on **data flow only**
- Assume: **Straightline code** without control flow
- Example:

```
x = 0
y = 0
x = 23
z = 5
y = x - 3
z = x  + 1
z = y * 3
```

Slicing criterion

→

```
y = 0
x = 23
y = x - 3
```

# Milestone 3

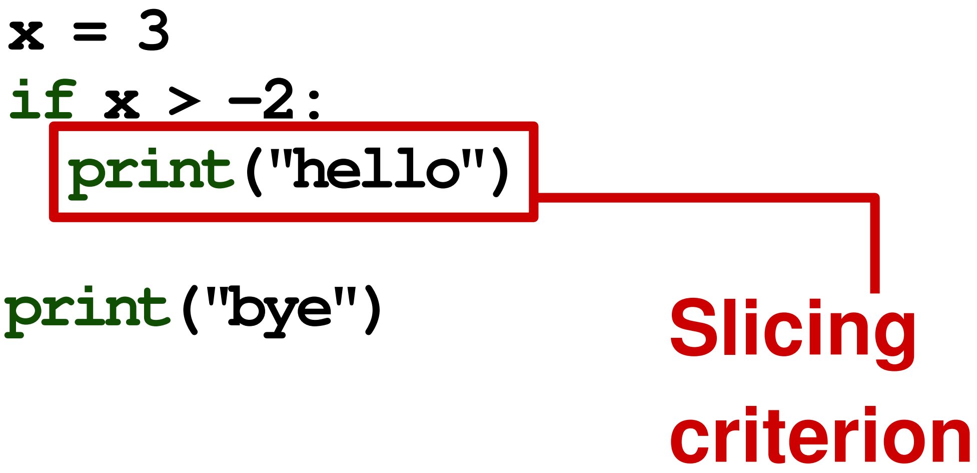
- Slicing based on both data flow and control flow
- Now, code may have branches, loops, etc.
- Example:

```
x = 3
if x > -2:
  print("hello")

print("bye")
```

# Milestone 3

- Slicing based on **both data flow and control flow**
- Now, code may have branches, loops, etc.
- Example:



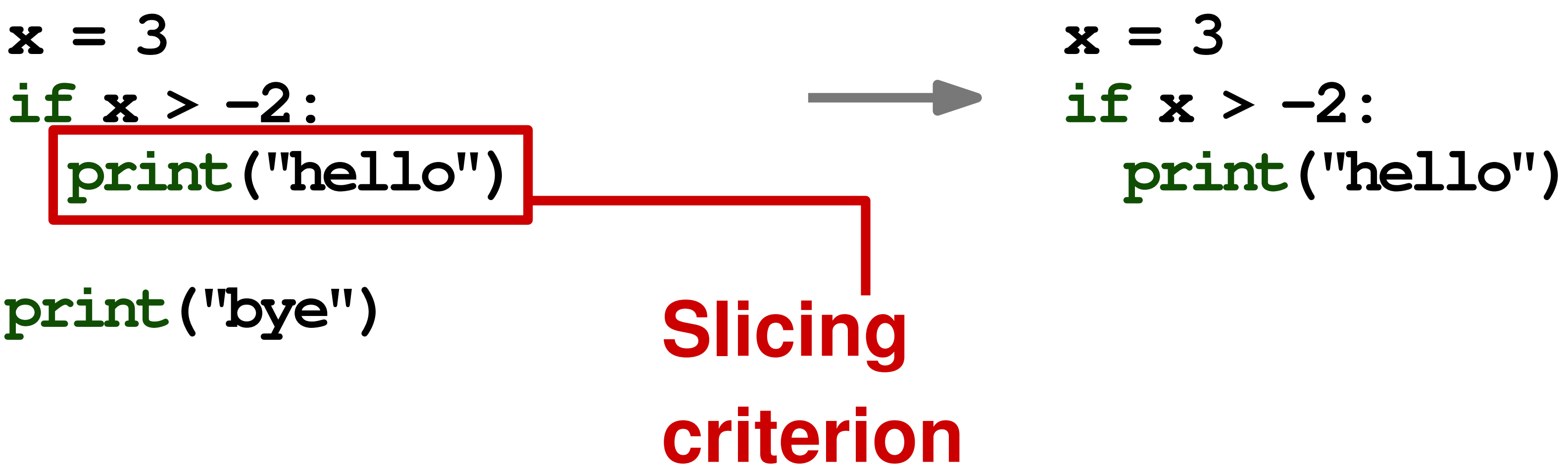
```
x = 3
if x > -2:
 print("hello")

print("bye")
```

Slicing criterion

# Milestone 3

- Slicing based on **both data flow and control flow**
- Now, code may have branches, loops, etc.
- Example:



```
x = 3
if x > -2:
  print("hello")

print("bye")
```

```
x = 3
if x > -2:
  print("hello")
```

Slicing criterion

# Scripts and Tests

**Provided by us:**

- **To-be-implemented scripts, e.g., slice.py**
- **Test suite of programs to slice**
  - □ Run with *pytest*

**Expected from you:**

- **Don’t rename any files**
- **Add more tests**

# Mentoring

- Each student gets a mentor
- Meet at least three times (once per milestone)
- Mentor assignment and meeting dates: Message in Ilias

# Timeline

- **Milestone 1: Due in week of Nov 20–24**
- **Milestone 2: Due in week of Dec 11-15**
- **Milestone 3: Due in week of Jan 15–19**
- **Full project due: Feb 1**
  - □ Project report (up to 4 pages)
  - □ Your implementation
- **Oral presentation: Week of Feb 5–9**

# Timeline

- **Milestone 1: Due in week of Nov 20–24**
- **Milestone 2: Due in week of Dec 11-15**
- **Milestone 3: Due in week of Jan 15–19**
- **Full project due: Feb 1**
  - □ Project report (up to 4 pages)
  - □ Your implementation
- **Oral presentation: Week of Feb 5–9**

**Soft deadlines**

# Timeline

- Milestone 1: Due in week of Nov 20–24
- Milestone 2: Due in week of Dec 11-15
- Milestone 3: Due in week of Jan 15–19
- Full project due: Feb 1
  - □ Project report (up to 4 pages)
  - □ Your implementation
- Oral presentation: Week of Feb 5–9

**Hard deadlines**