2024. 04. 11. Project Proposal Talk

Visualizing Fuzzing Status on Def-Use Graph and its impact

Geon Park

KAIST Programming Systems Laboratory

KAIST

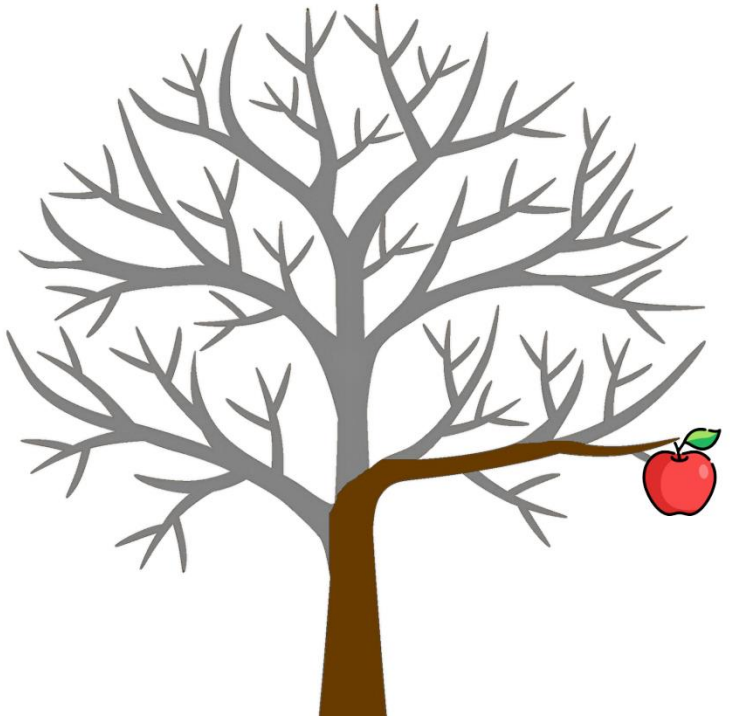
Programming Systems Laboratory

Directed Fuzzing

- **Fuzzing** Tests programs through randomly generated inputs.
  - e.g., Google's OSS Fuzz project, AFL

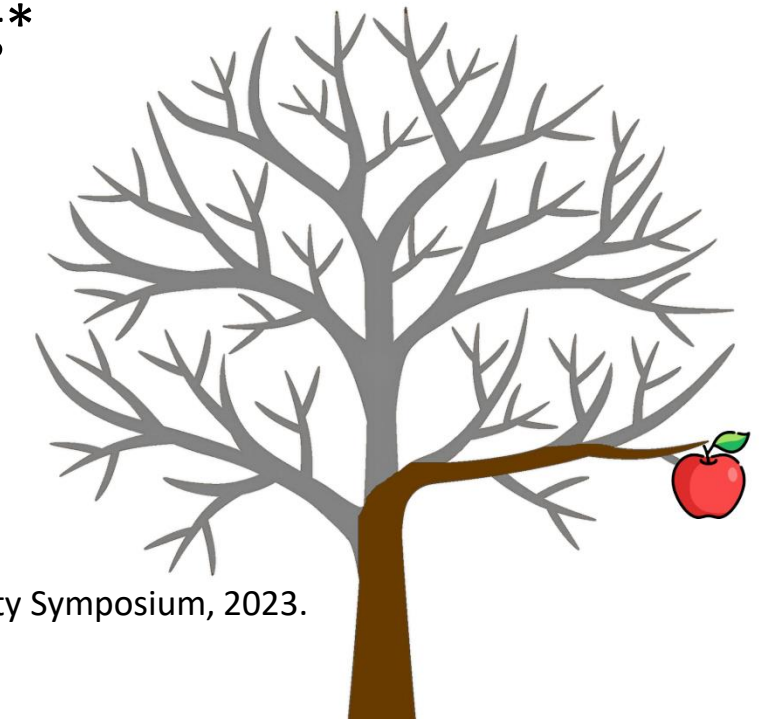
Directed Fuzzing

- **Fuzzing** Tests programs through randomly generated inputs.
  - e.g., Google's OSS Fuzz project, AFL
- **Directed fuzzing** aims to reach target location(s) of code
  - e.g., Examine recently changed code area, generate crashing input from bug report



Directed Fuzzing

- **Fuzzing** Tests programs through randomly generated inputs.
  - e.g., Google's OSS Fuzz project, AFL
- **Directed fuzzing** aims to reach target location(s) of code
  - e.g., Examine recently changed code area, generate crashing input from bug report
- Reproduces bug 1.93 times faster than undirected fuzzing*
  - Undirected fuzzing: AFL, directed fuzzing: DAFL



* Tae Eun Kim et al., DAFL: Directed Grey-box Fuzzing guided by Data Dependency. USENIX Security Symposium, 2023.

Developing Fuzzing

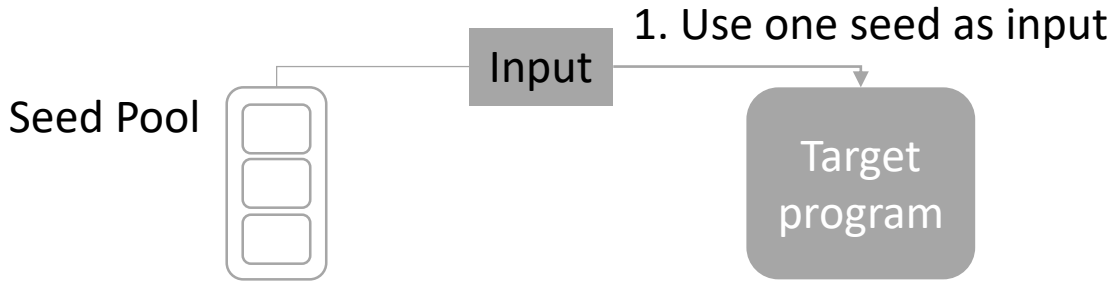
- Performance varies based on how fuzzing guidance is given and used.

Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.

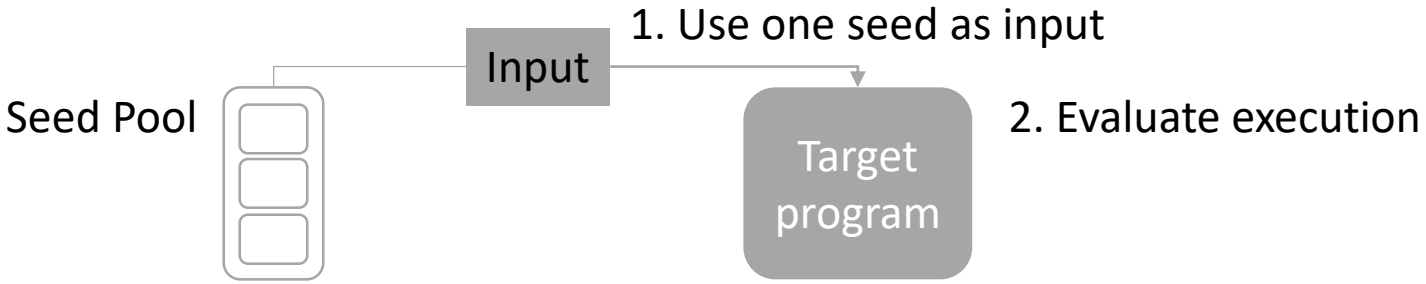
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



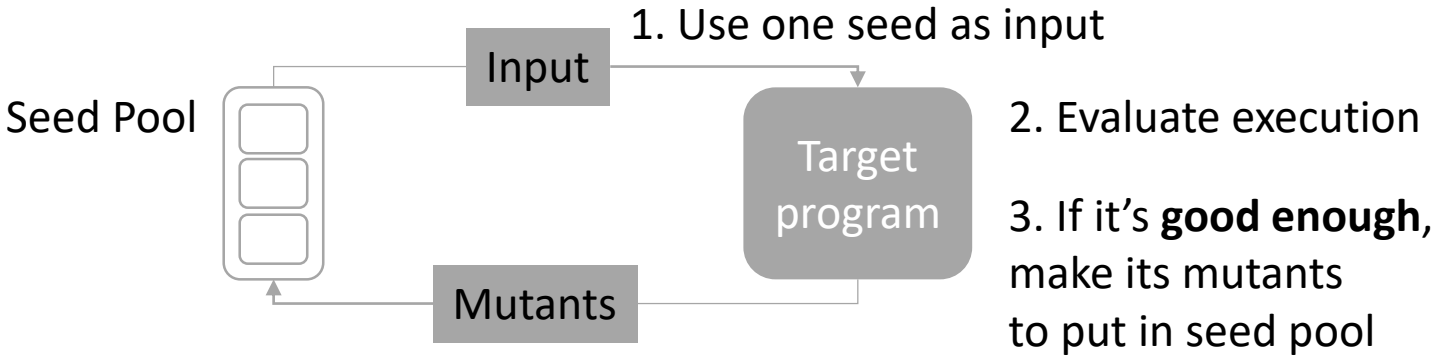
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



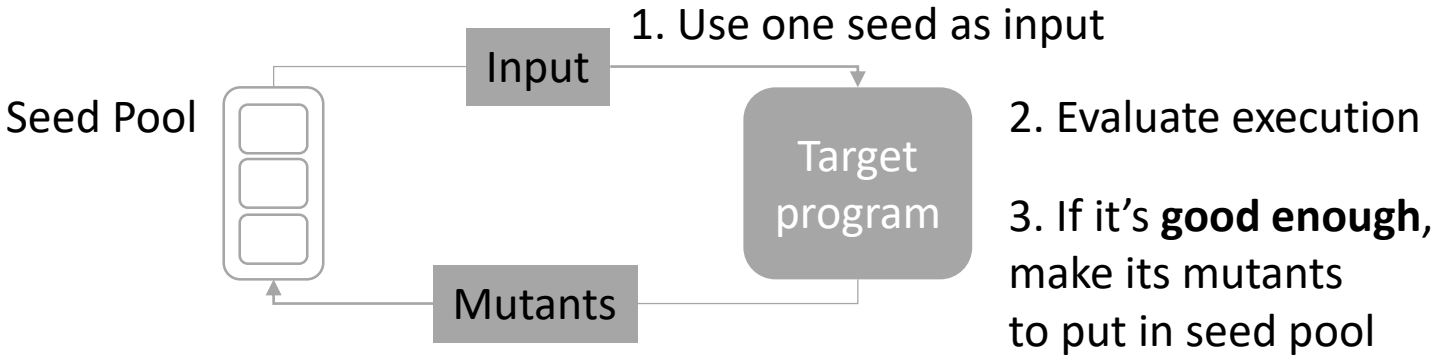
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



- What is the criteria for the execution to be **good enough**?

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

```
1:  def doSomething():
2:  if (flag == 0) then
3:     print('0')
4:  end if
5:  if (flag == 1) then
6:     print('1')
7:  end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important
  - Data-flow analysis is for what part of code effects data used in target line

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important
  - Data-flow analysis is for what part of code effects data used in target line

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important
  - Data-flow analysis is for what part of code effects data used in target line
  - In example, doSomething() does not effect data for line 5

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important
  - Data-flow analysis is for what part of code effects data used in target line
  - In example, doSomething() does not effect data for line 5
  - increasing code coverage of doSomething() does not affect performance

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

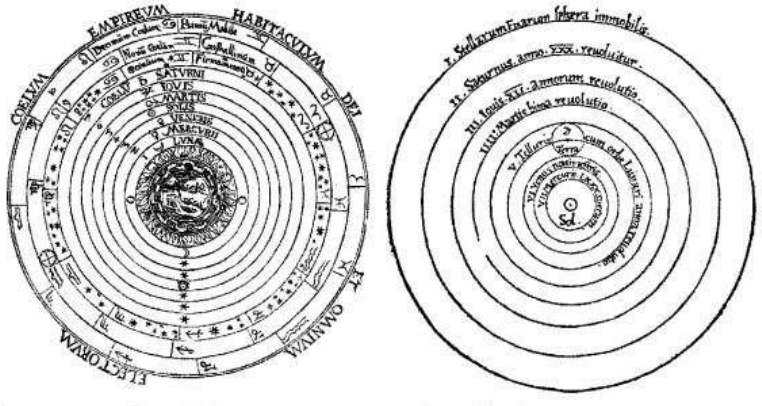
width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

Observating

- Observation is the foundation of scientific inquiry

Observating

- Observation is the foundation of scientific inquiry



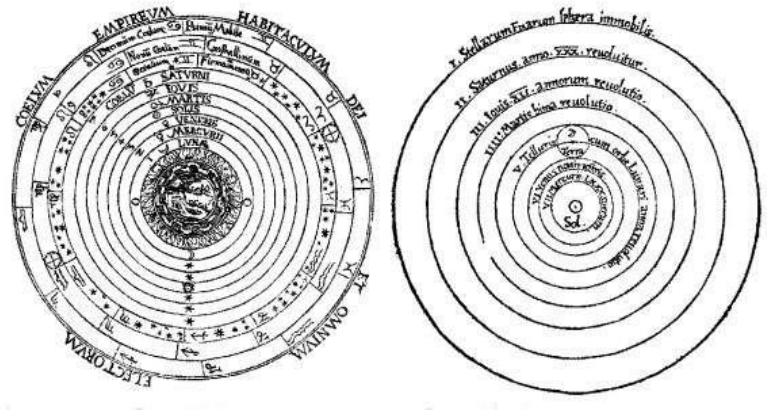
Heliocentrism, for example

was made by observing motions of celestials

describe the structure of the solar system

Observating

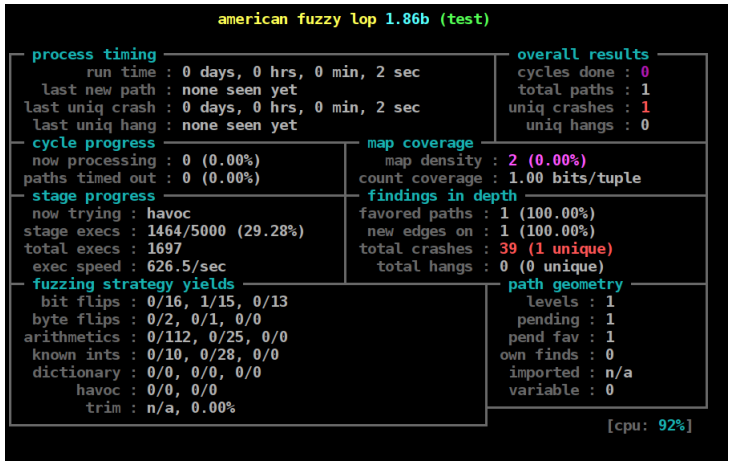
- Observation is the foundation of scientific inquiry



Heliocentrism, for example

was made by observing motions of celestials

describe the structure of the solar system



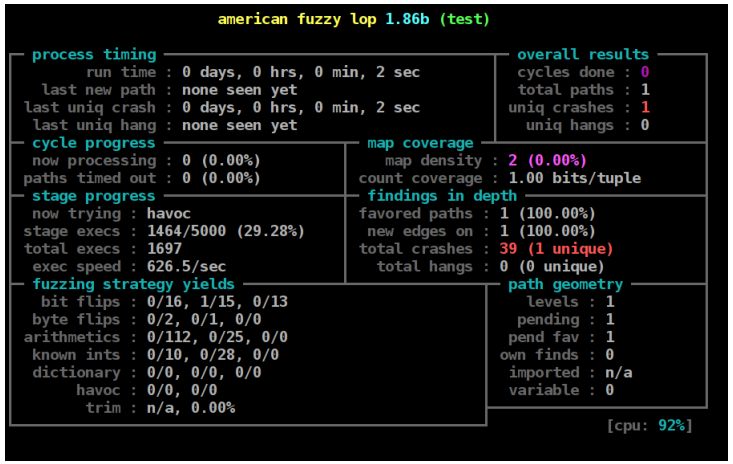
Fuzzing has multiple output data

lots of text/non-text data to observe

such can describe the progress of system

Observating

- Observation is the foundation of scientific inquiry



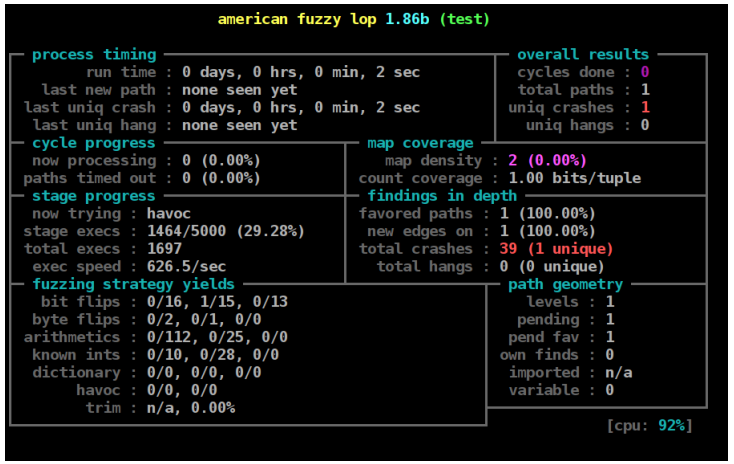
Fuzzing has multiple output data

lots of text/non-text data to observe

such can describe the progress of system

Observating

- Observation is the foundation of scientific inquiry
- Fuzzing dev. wants to know overall performance



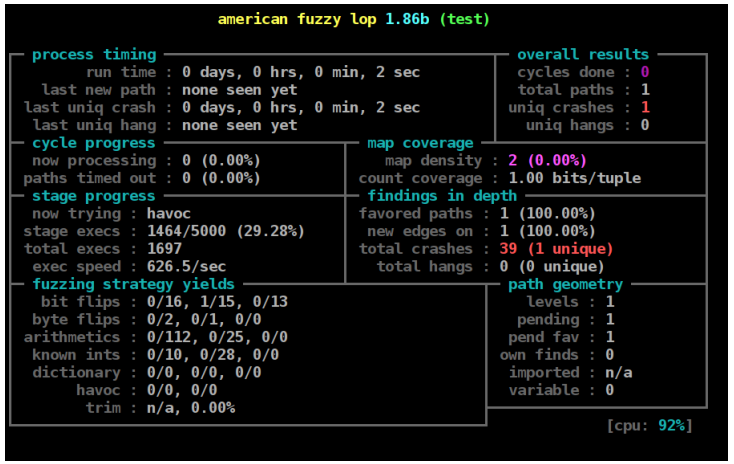
Fuzzing has multiple output data

lots of text/non-text data to observe

such can describe the progress of system

Observating

- Observation is the foundation of scientific inquiry
- Fuzzing dev. wants to know overall performance
- Yet, fuzzing output is too big
  - 24-hour fuzzing saves 2,874 seeds on average*



Fuzzing has multiple output data

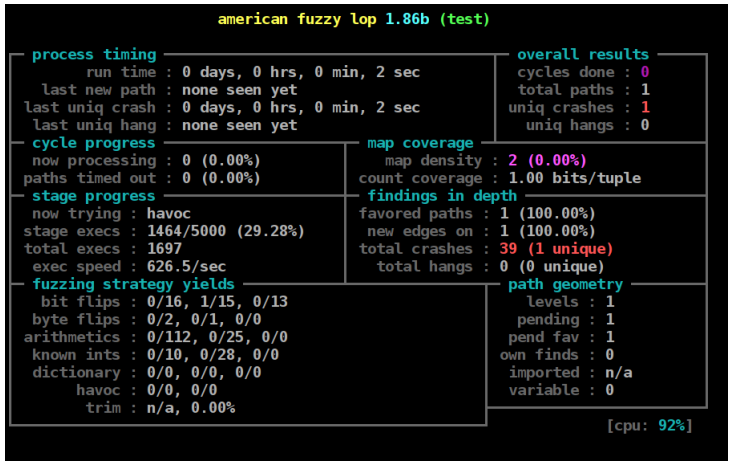
lots of text/non-text data to observe

such can describe the progress of system

*Done 40 iterations, on program swftophp (v0.4.8)

Observating

- Observation is the foundation of scientific inquiry
- Fuzzing dev. wants to know overall performance
- Yet, fuzzing output is too big
  - 24-hour fuzzing saves 2,874 seeds on average*
- so dev. can't grasp on one's performance



Fuzzing has multiple output data

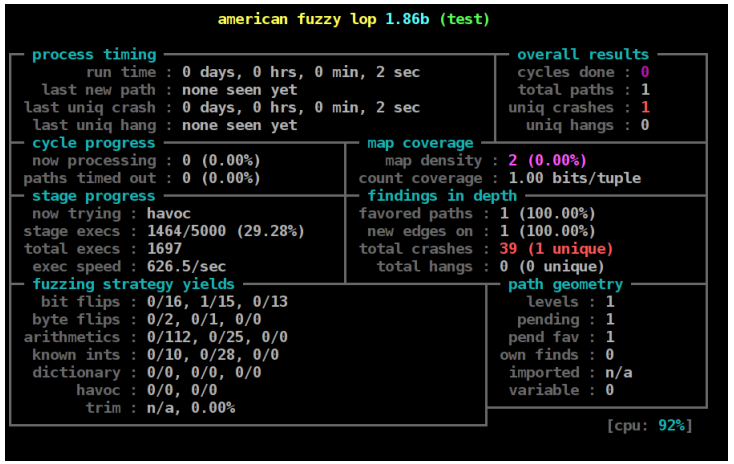
lots of text/non-text data to observe

such can describe the progress of system

*Done 40 iterations, on program swftophp (v0.4.8)

Observating

- Observation is the foundation of scientific inquiry
- Fuzzing dev. wants to know overall performance
- Yet, fuzzing output is too big
  - 24-hour fuzzing saves 2,874 seeds on average*
- so dev. can't grasp on one's performance
- **None** visualizing tool for directed fuzzing **yet!**



Fuzzing has multiple output data

lots of text/non-text data to observe

such can describe the progress of system

*Done 40 iterations, on program swftophp (v0.4.8)

Observating

- Questions for directed fuzzing visualization

Observating

- Questions for directed fuzzing visualization

1. How to observe the **performance** of directed fuzzing in one eyesight?

Observating

- Questions for directed fuzzing visualization

1. How to observe the **performance** of directed fuzzing in one eyesight?
2. How to **visualize** such information?

Observating

- Questions for directed fuzzing visualization

1. How to observe the **performance** of directed fuzzing in one eyesight?
2. How to **visualize** such information?
3. How to see the **hourly status** of performance?

Observating data-flow

- How to observe the performance of directed fuzzing in one eyesight?

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

Observating data-flow

- How to observe the performance of directed fuzzing in one eyesight?
- One key aspect is **data-flow coverage**

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow

Observating data-flow

- How to observe the performance of directed fuzzing in one eyesight?
- One key aspect is **data-flow coverage**
- We can focus on data-flow that affects the target line

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow

Visualizing data-flow

- How to visualize such information?

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow

Visualizing data-flow

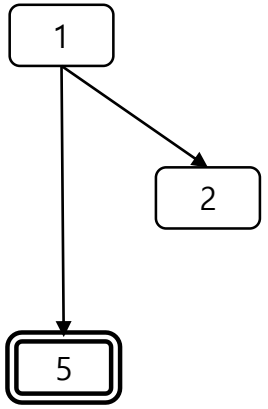
- How to visualize such information?
- Transform code to **def-use graph**.

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



def-use graph

Visualizing data-flow

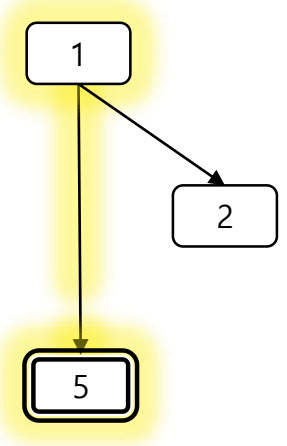
- How to visualize such information?
- Transform code to **def-use graph**.

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



def-use graph

Visualizing data-flow

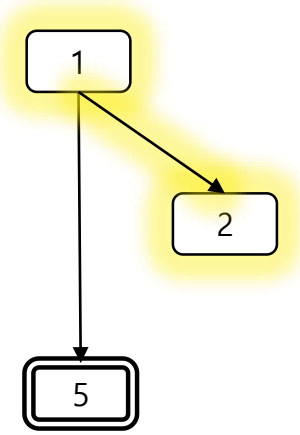
- How to visualize such information?
- Transform code to **def-use graph**.

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



def-use graph

Visualizing data-flow

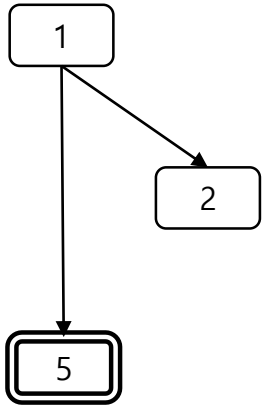
- How to visualize such information?
- Transform code to **def-use graph**.
- It is **yet first** to observe data-flow in fuzzing.

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

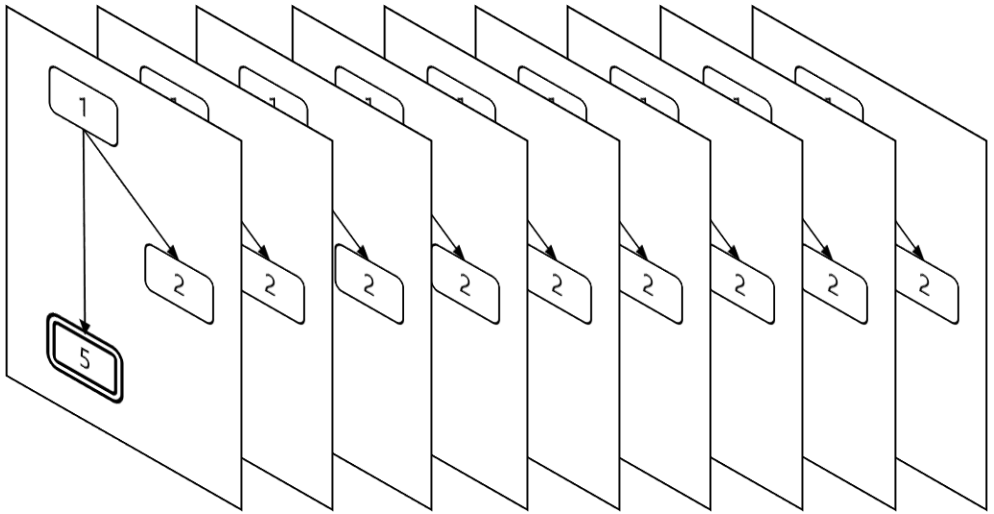
data-flow



def-use graph

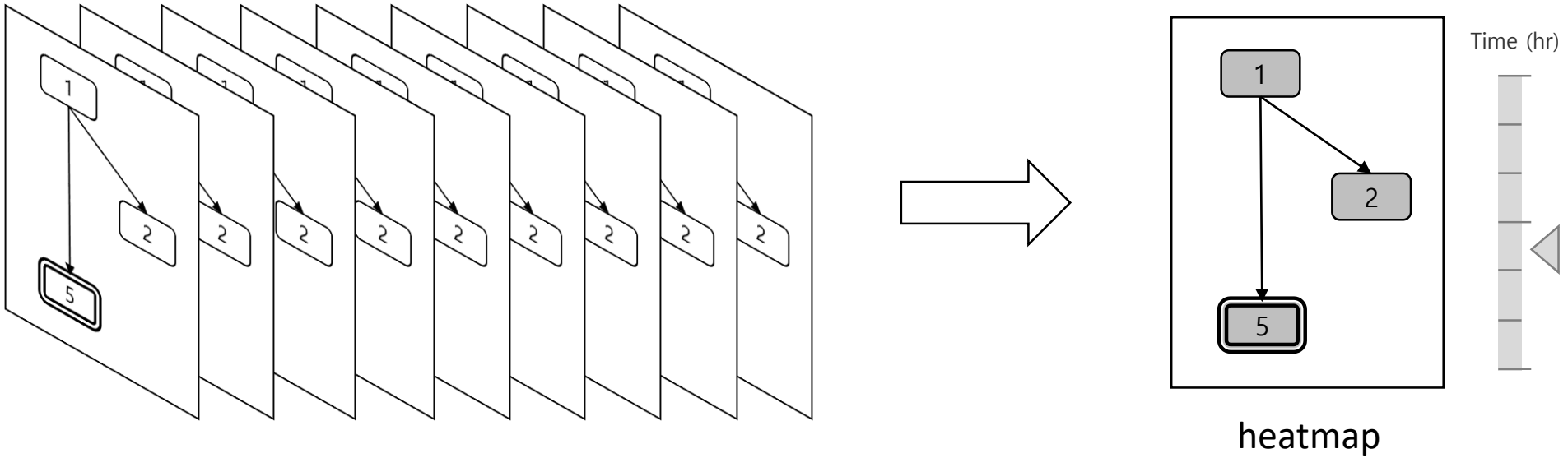
Visualizing data-flow over time

- How to see the hourly status of performance?



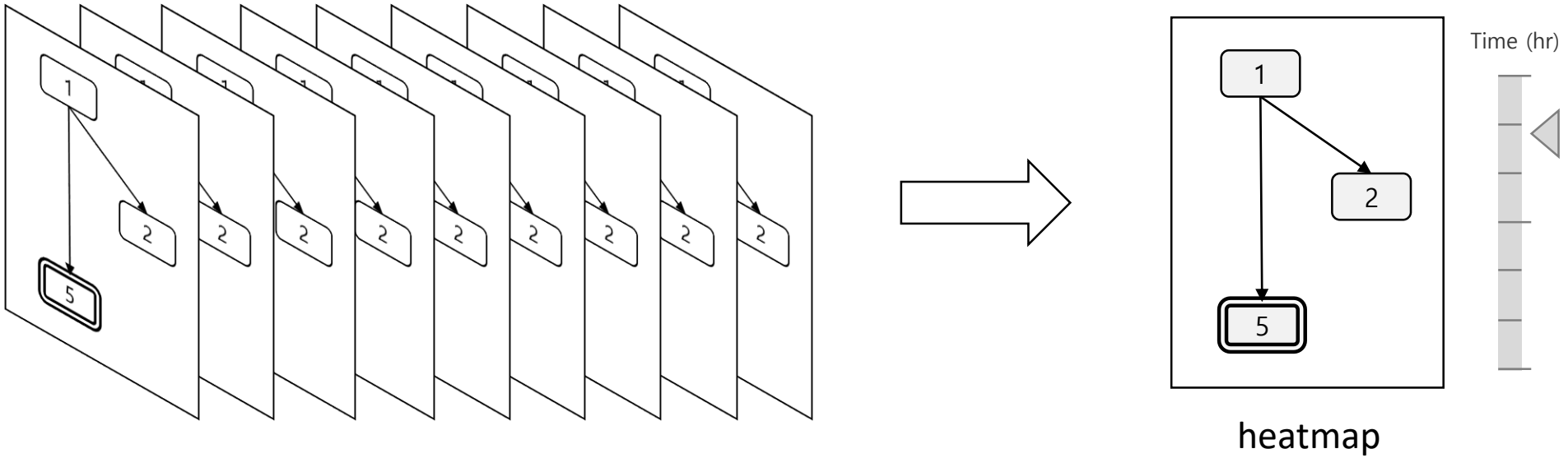
Visualizing data-flow over time

- How to see the hourly status of performance?
- Create a **heatmap** (how often node was hit) with adjustable time frames
- Good for evaluation: More often target line is hit, better chance to trigger bug



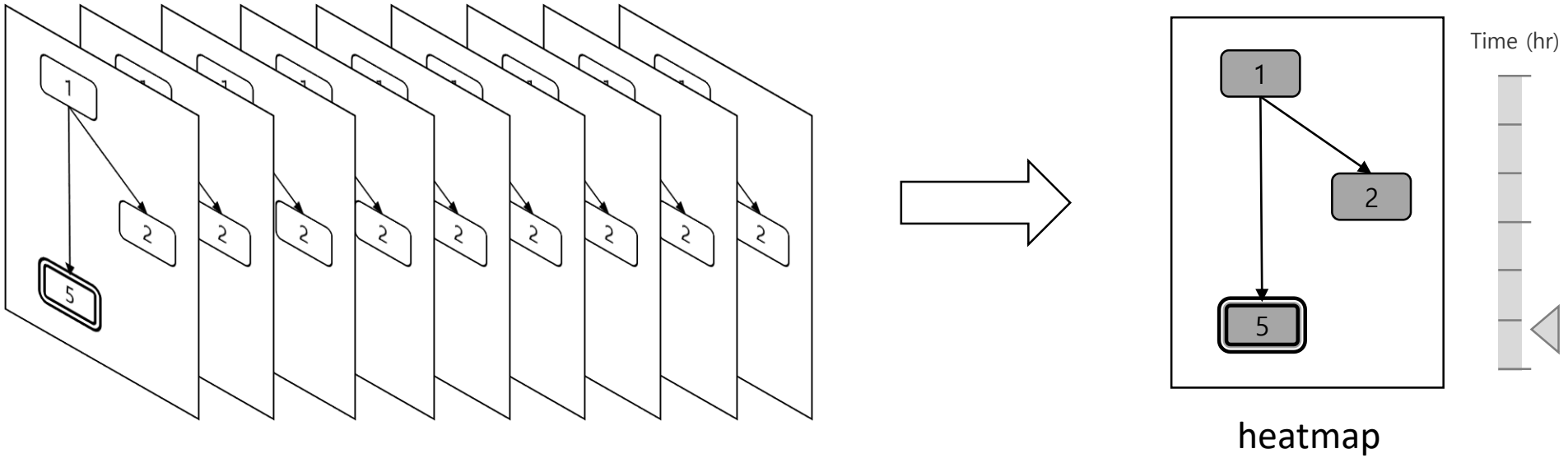
Visualizing data-flow over time

- How to see the hourly status of performance?
- Create a **heatmap** (how often node was hit) with adjustable time frames
- Good for evaluation: More often target line is hit, better chance to trigger bug



Visualizing data-flow over time

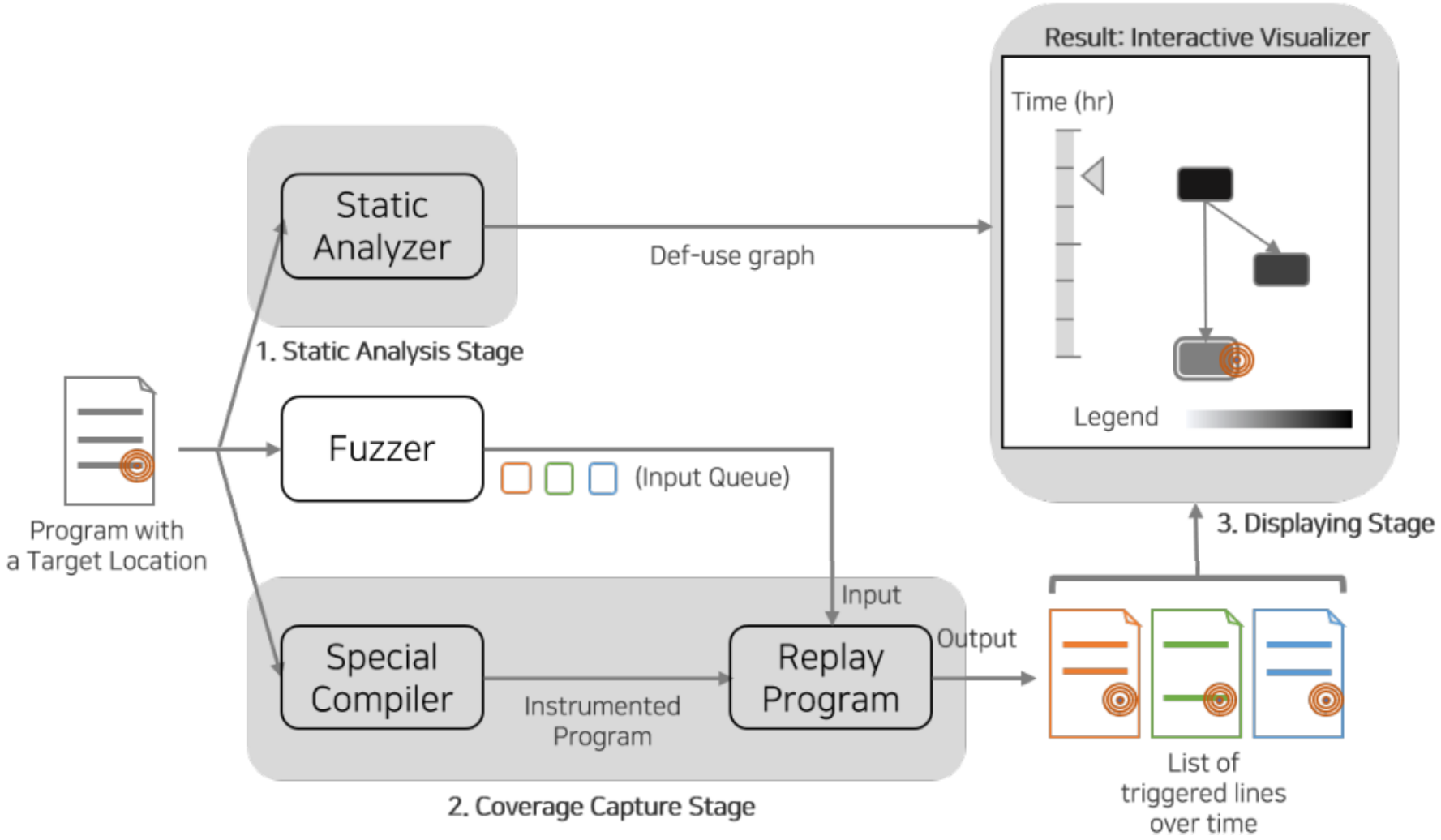
- How to see the hourly status of performance?
- Create a **heatmap** (how often node was hit) with adjustable time frames
- Good for evaluation: More often target line is hit, better chance to trigger bug



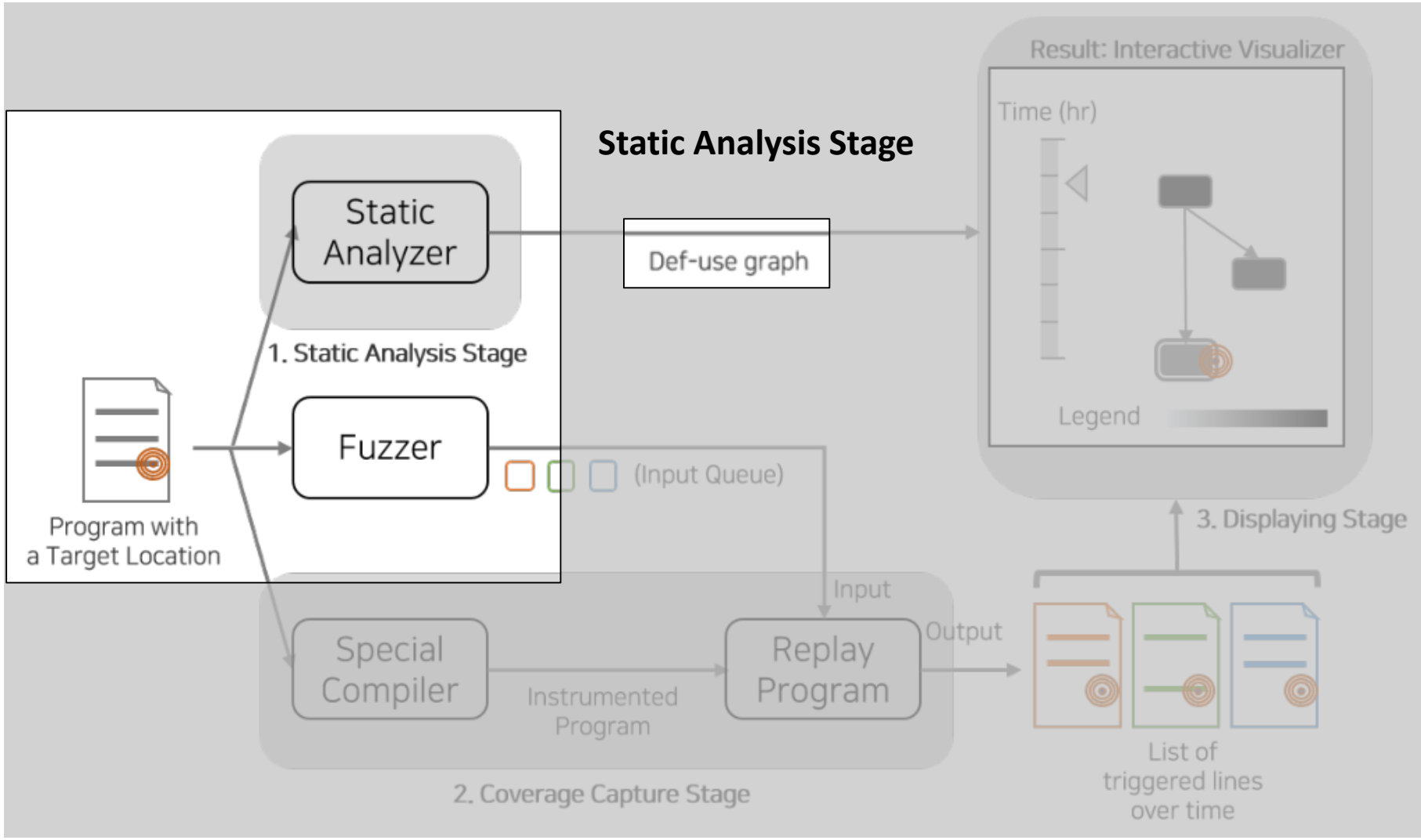
Visualizer Architecture

Visualizer Architecture

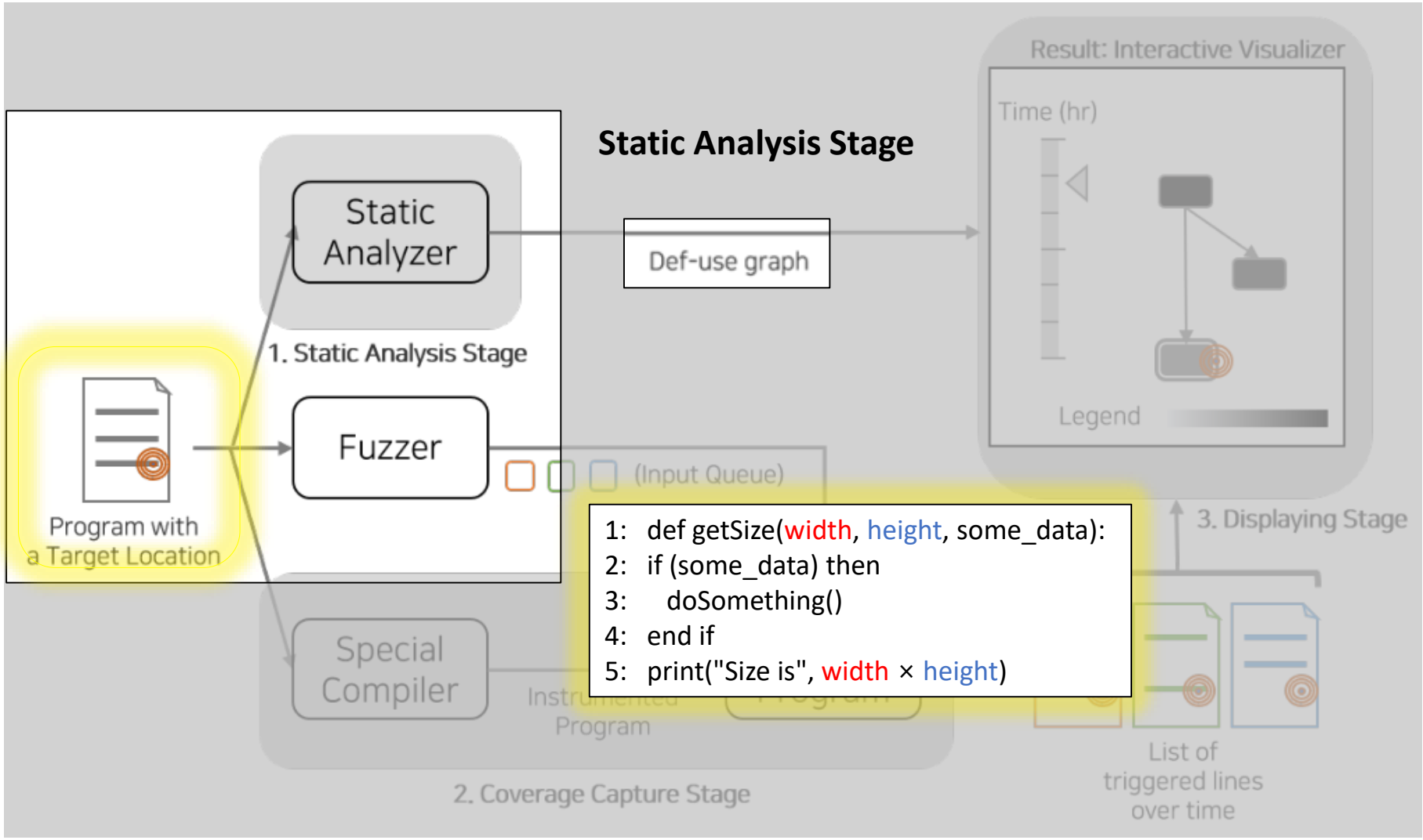
Visualizer Architecture



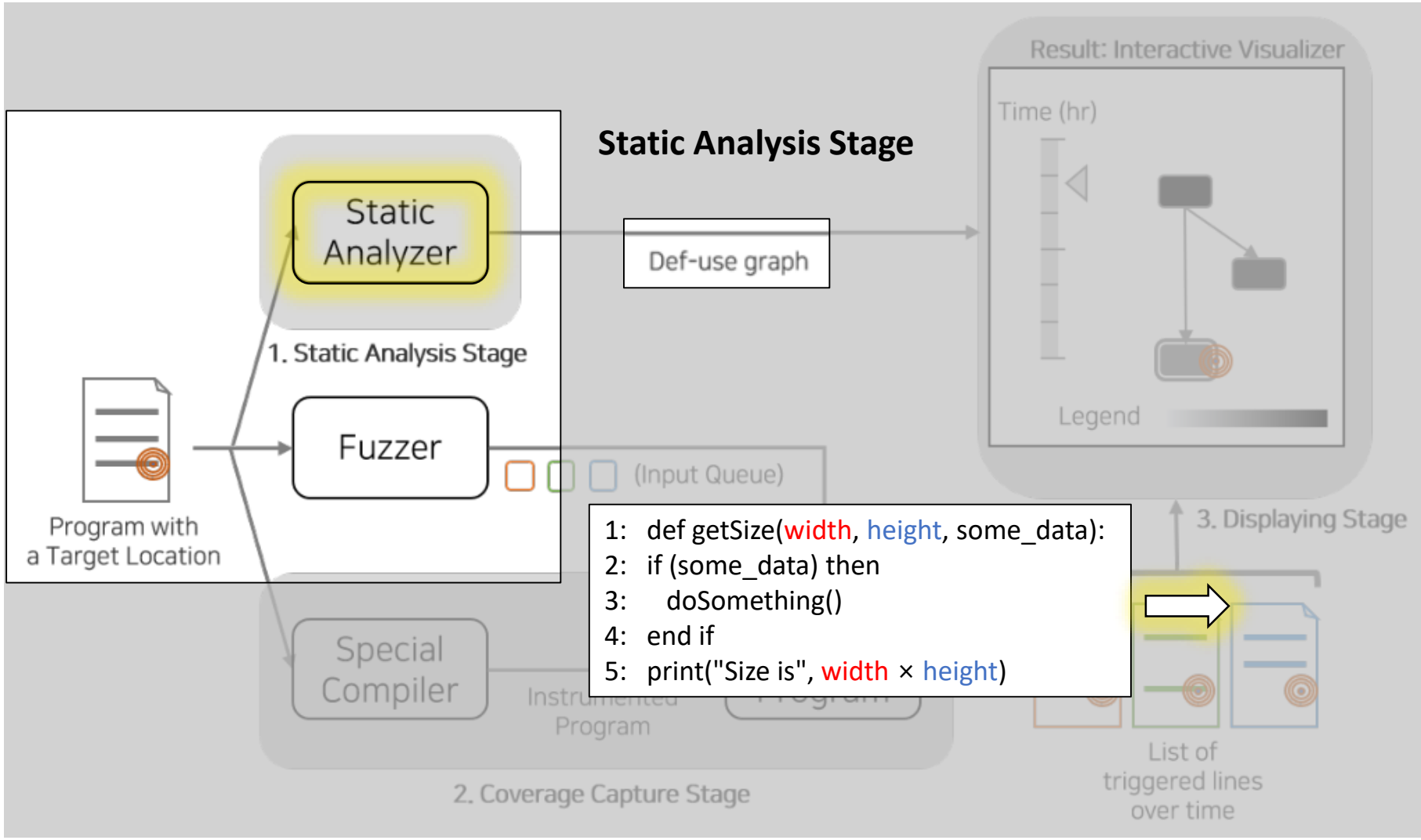
Visualizer Architecture



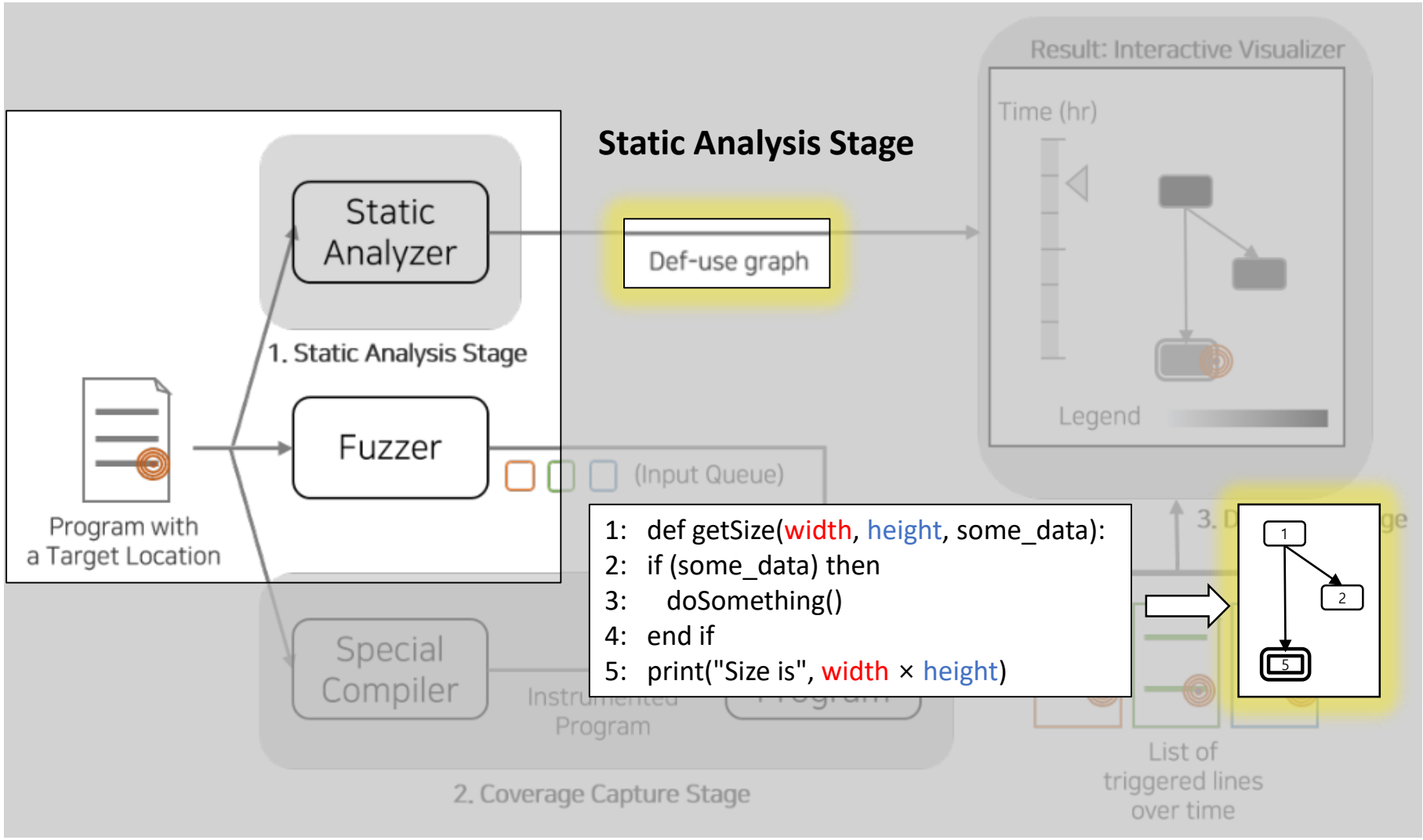
Visualizer Architecture



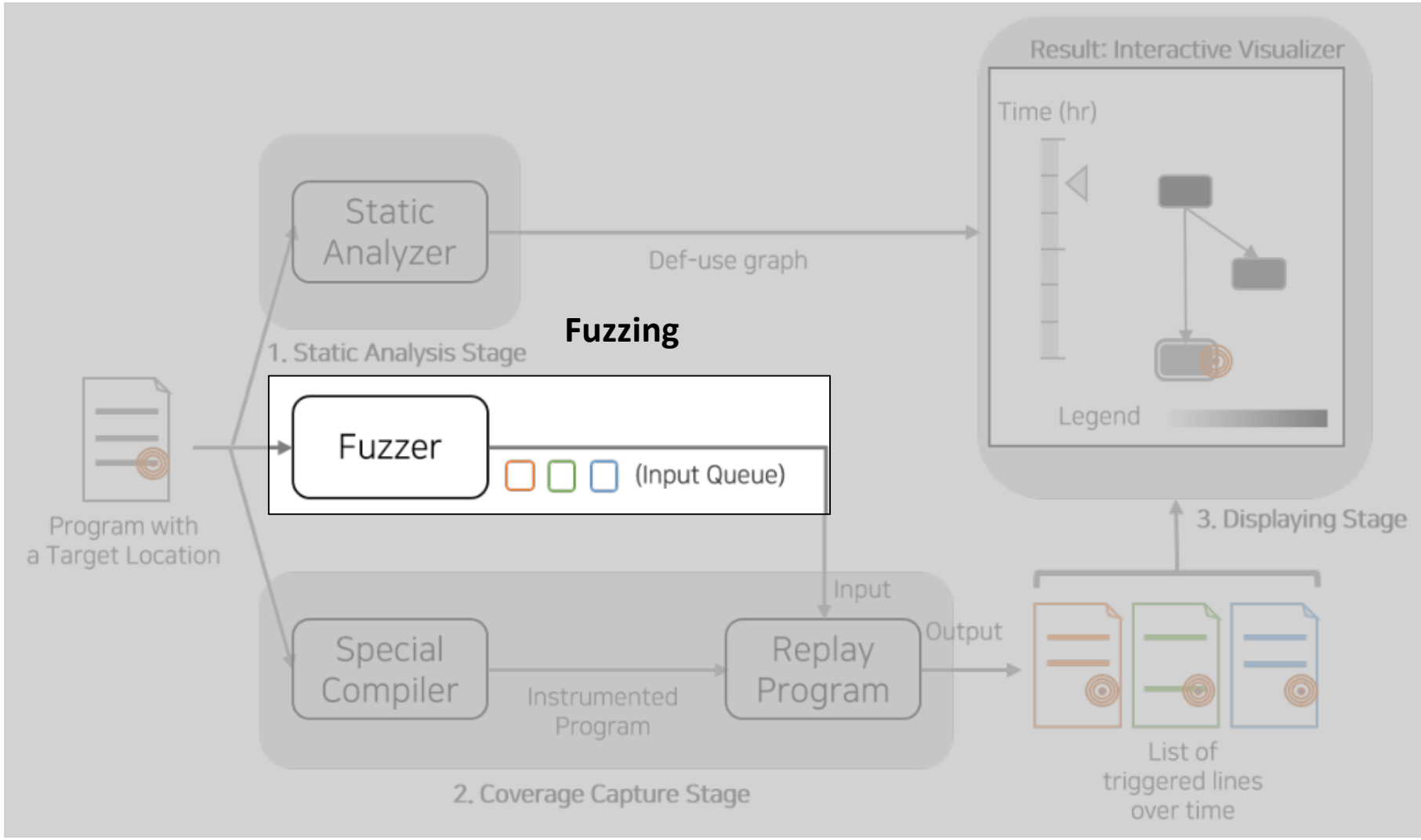
Visualizer Architecture



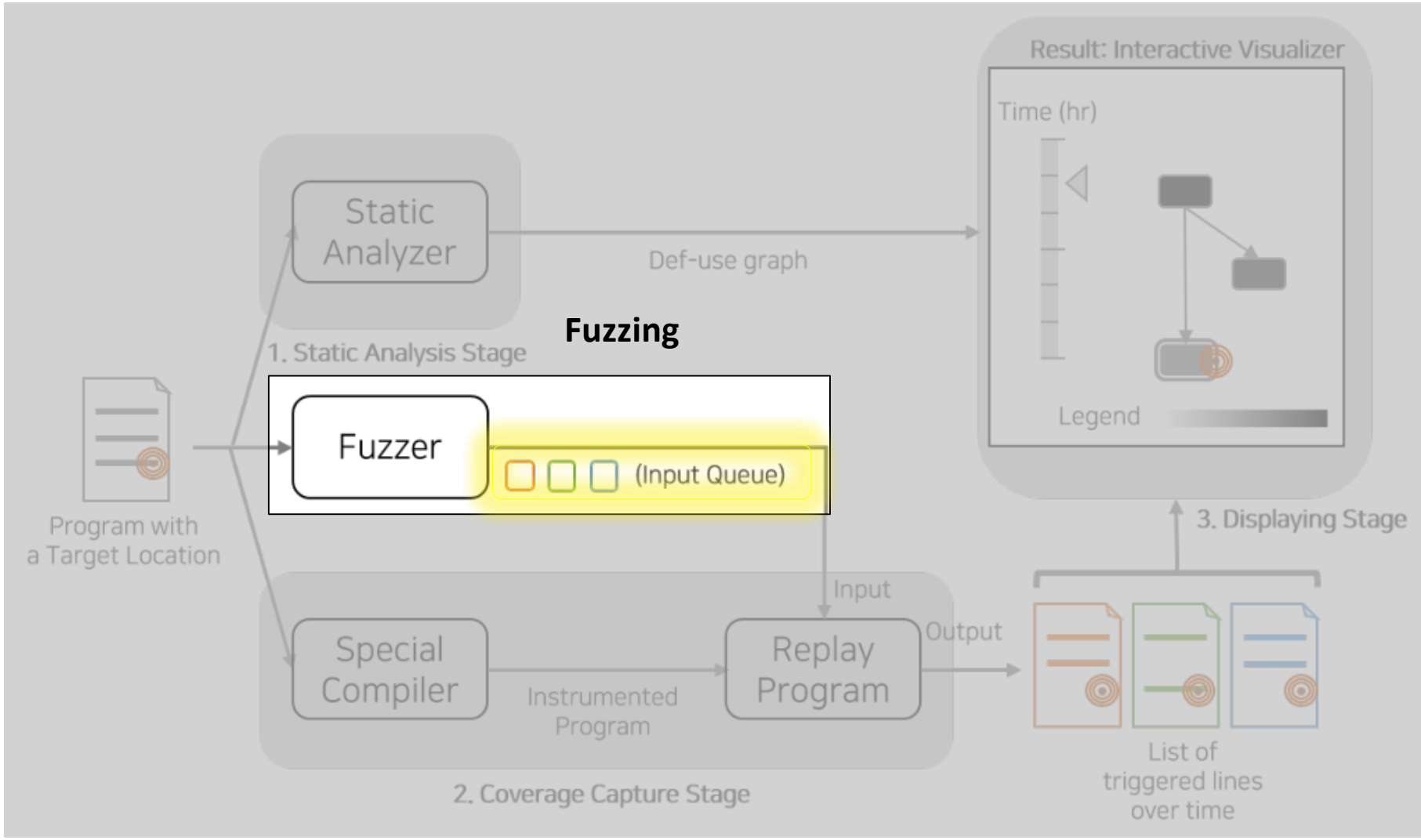
Visualizer Architecture



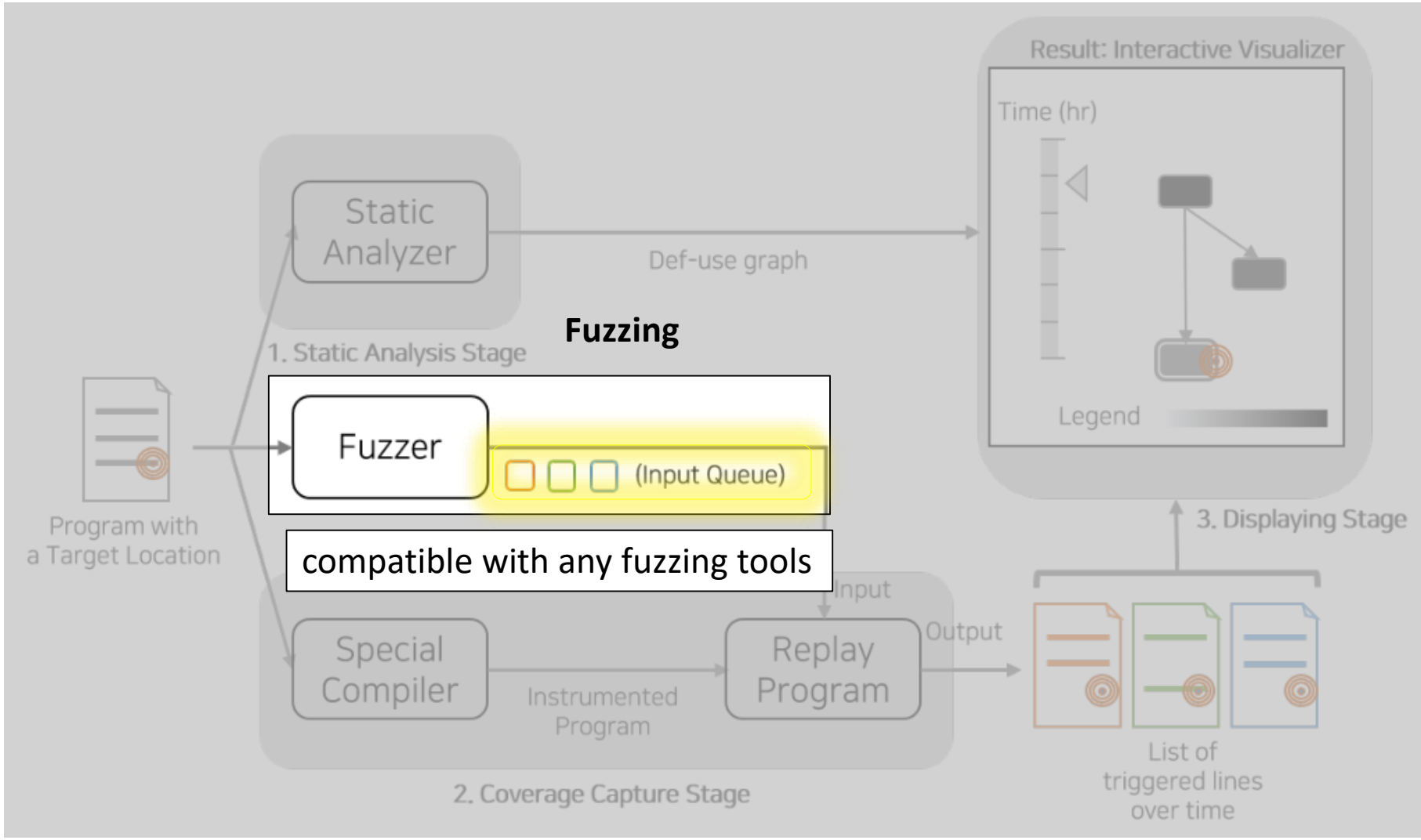
Visualizer Architecture



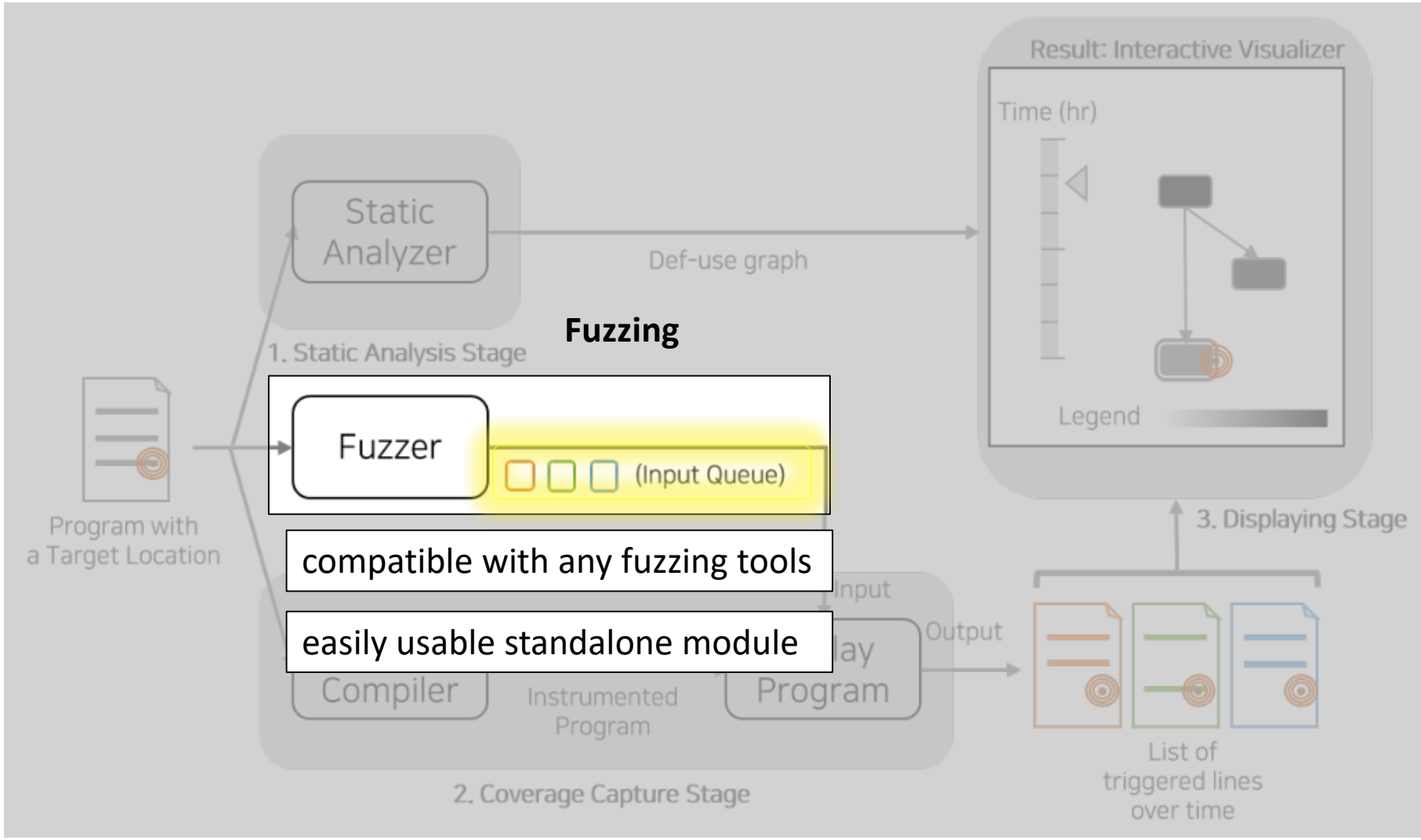
Visualizer Architecture



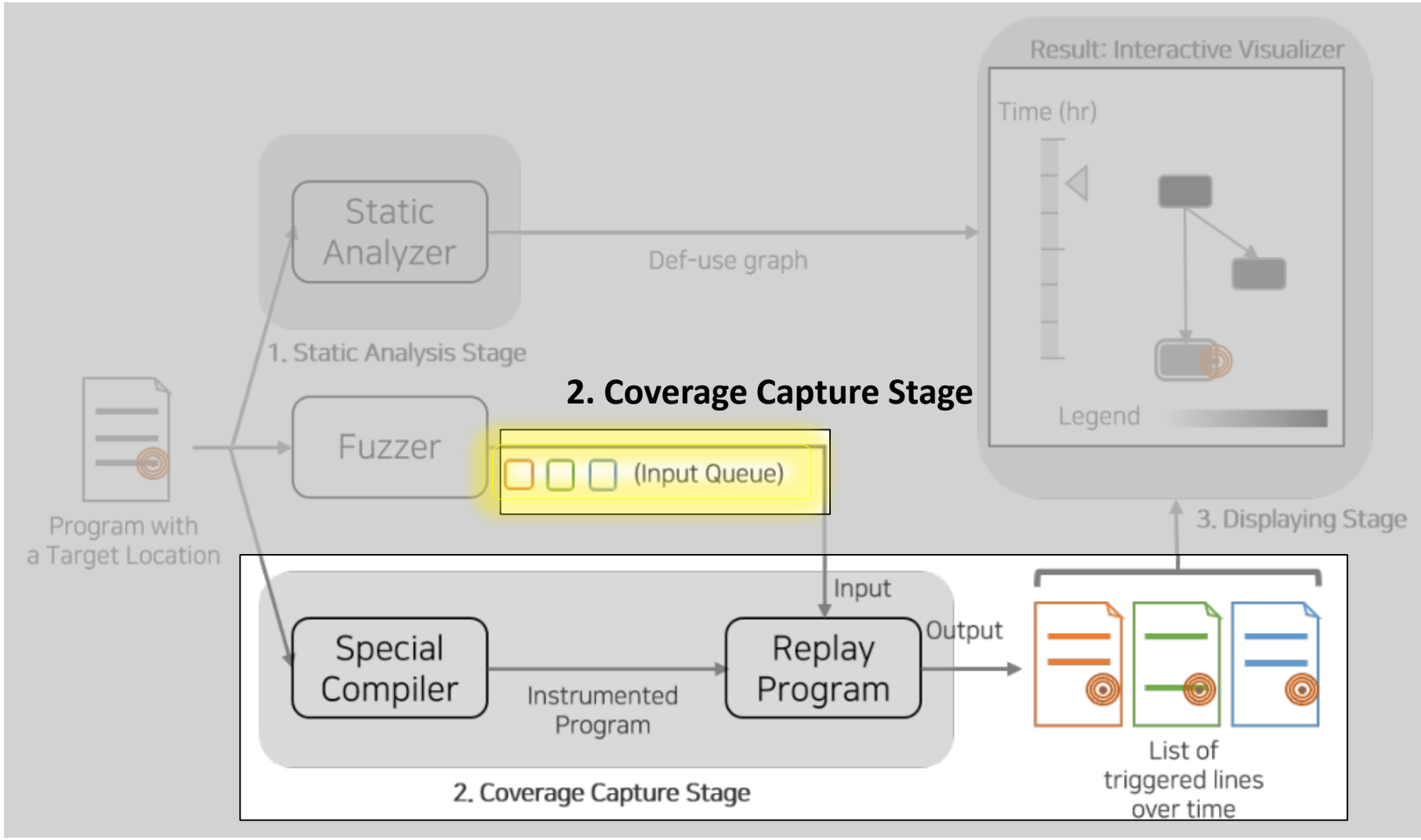
Visualizer Architecture



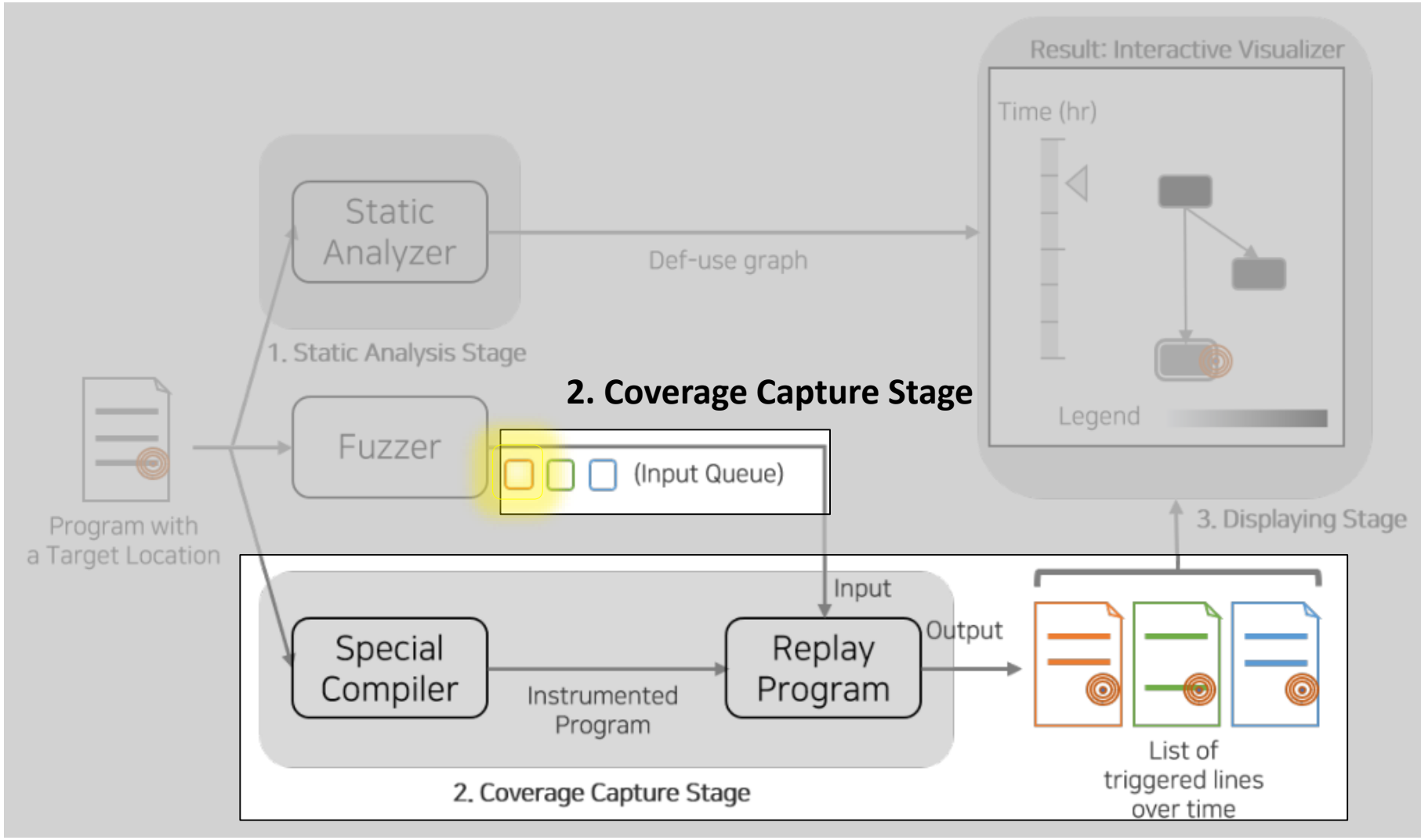
Visualizer Architecture



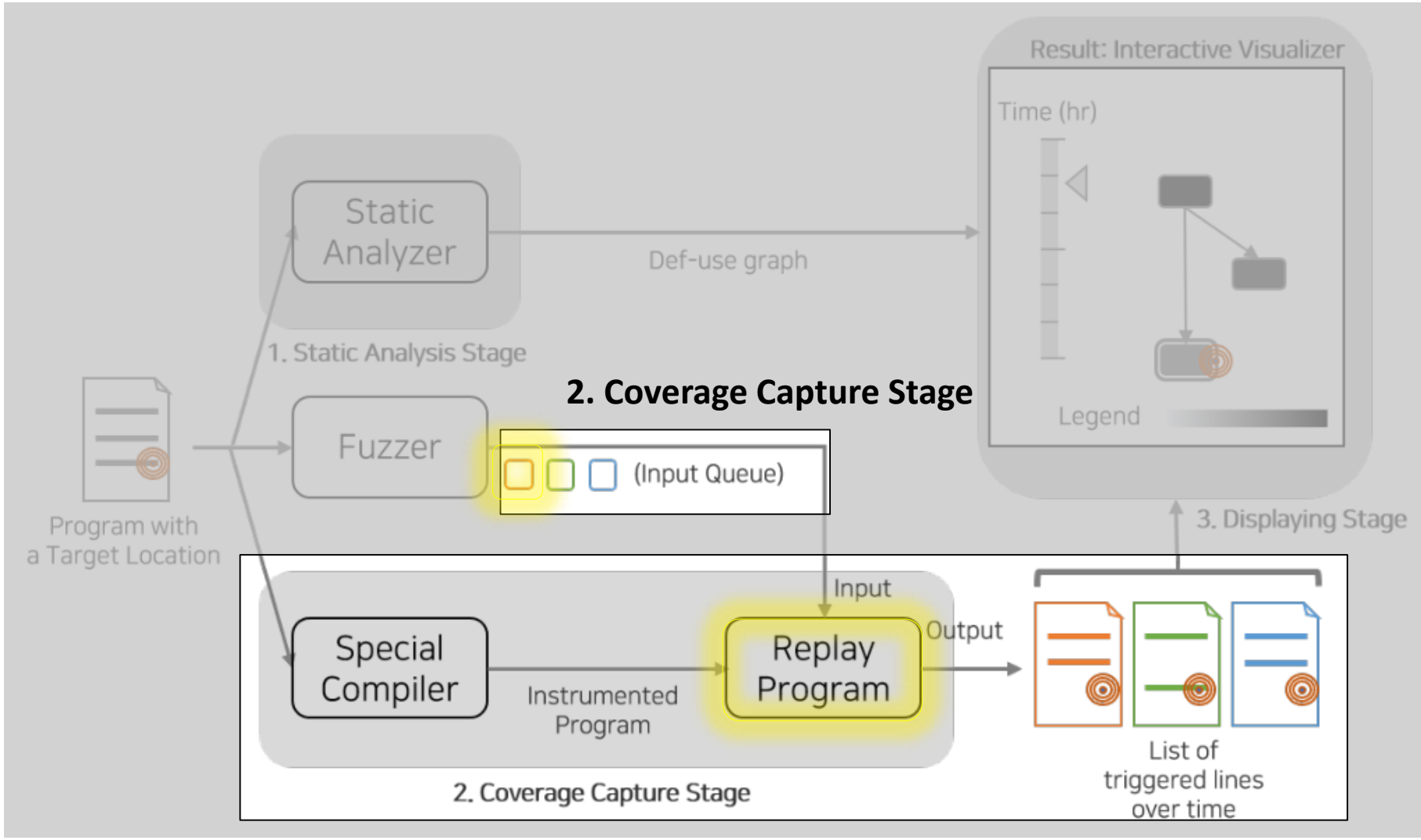
Visualizer Architecture



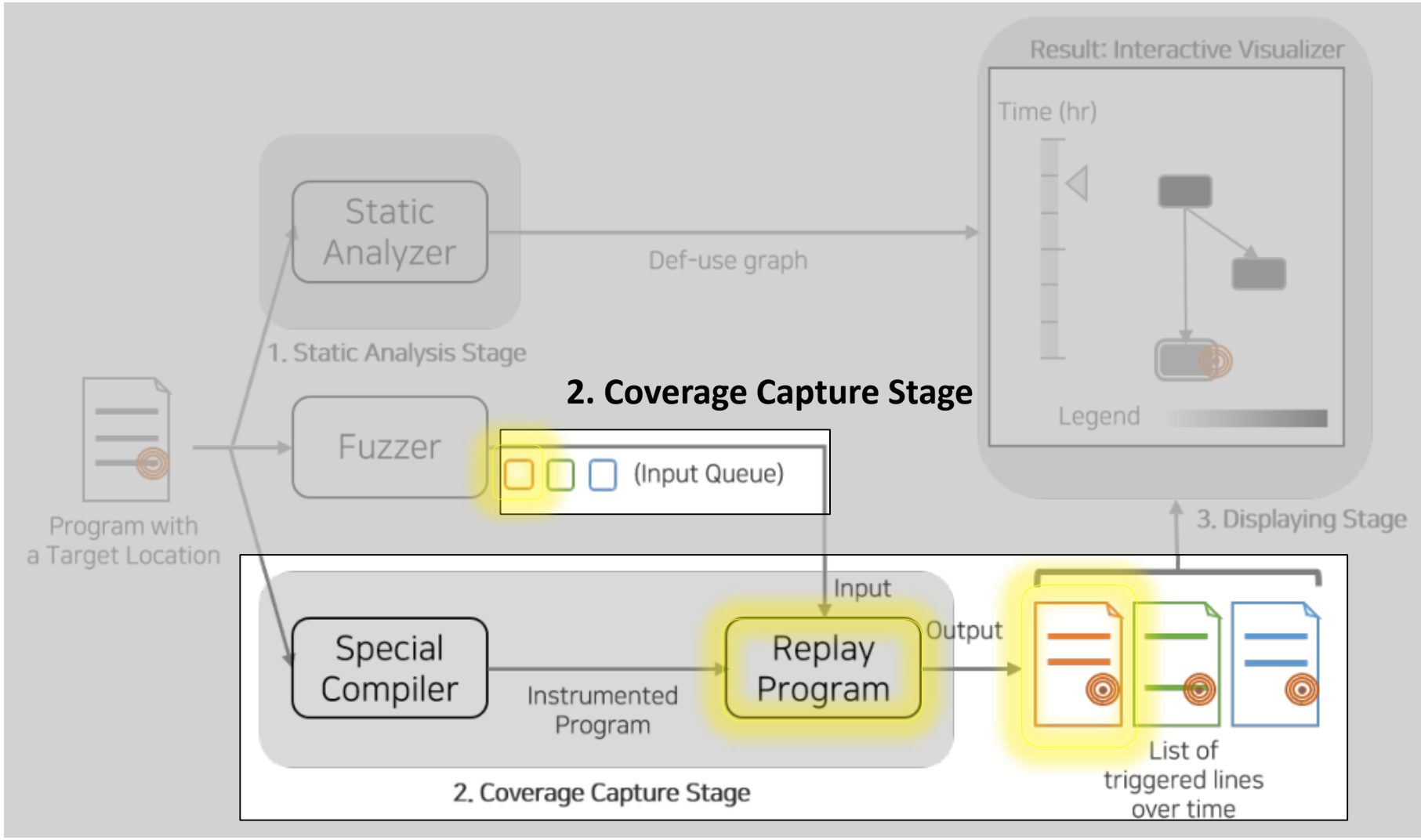
Visualizer Architecture



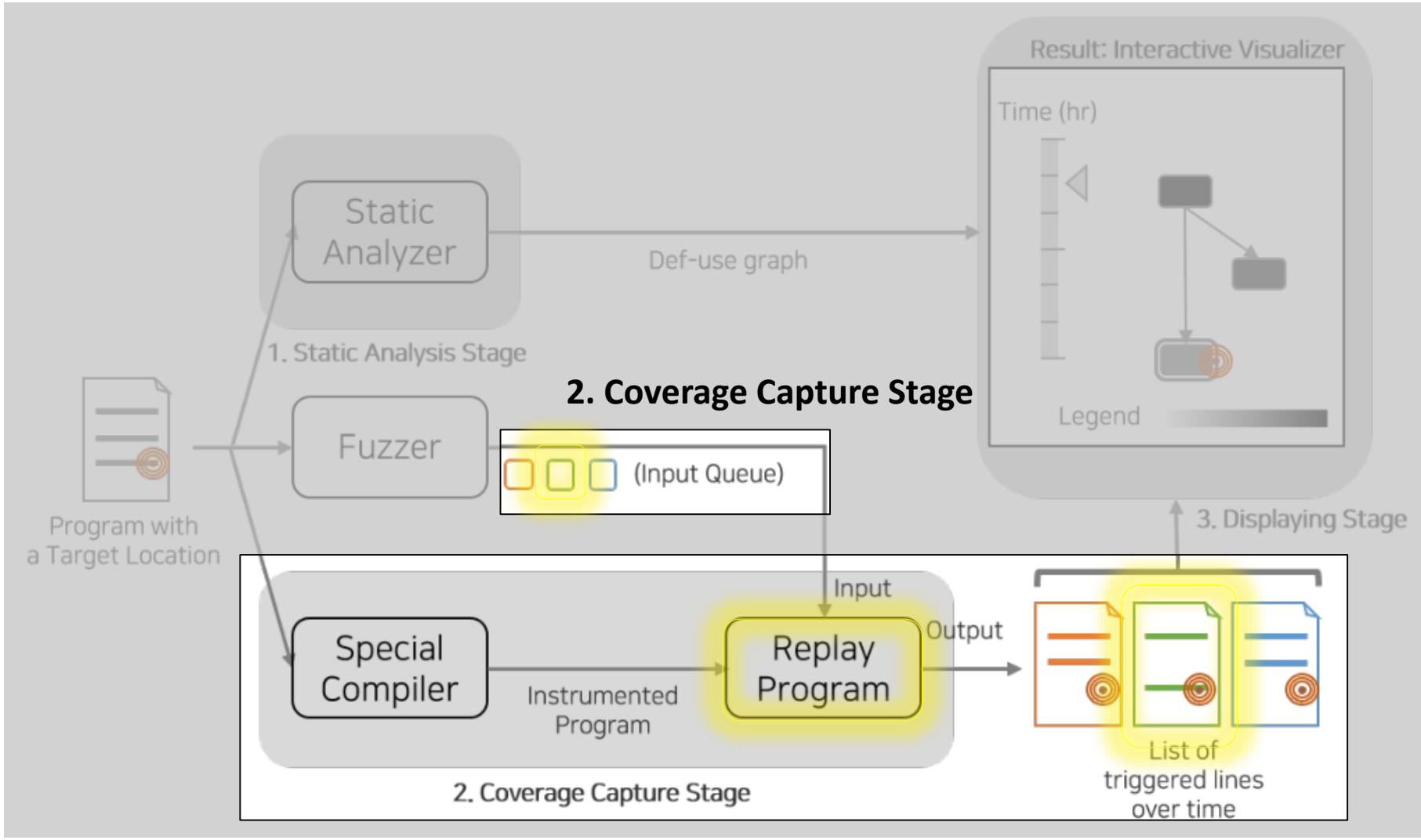
Visualizer Architecture



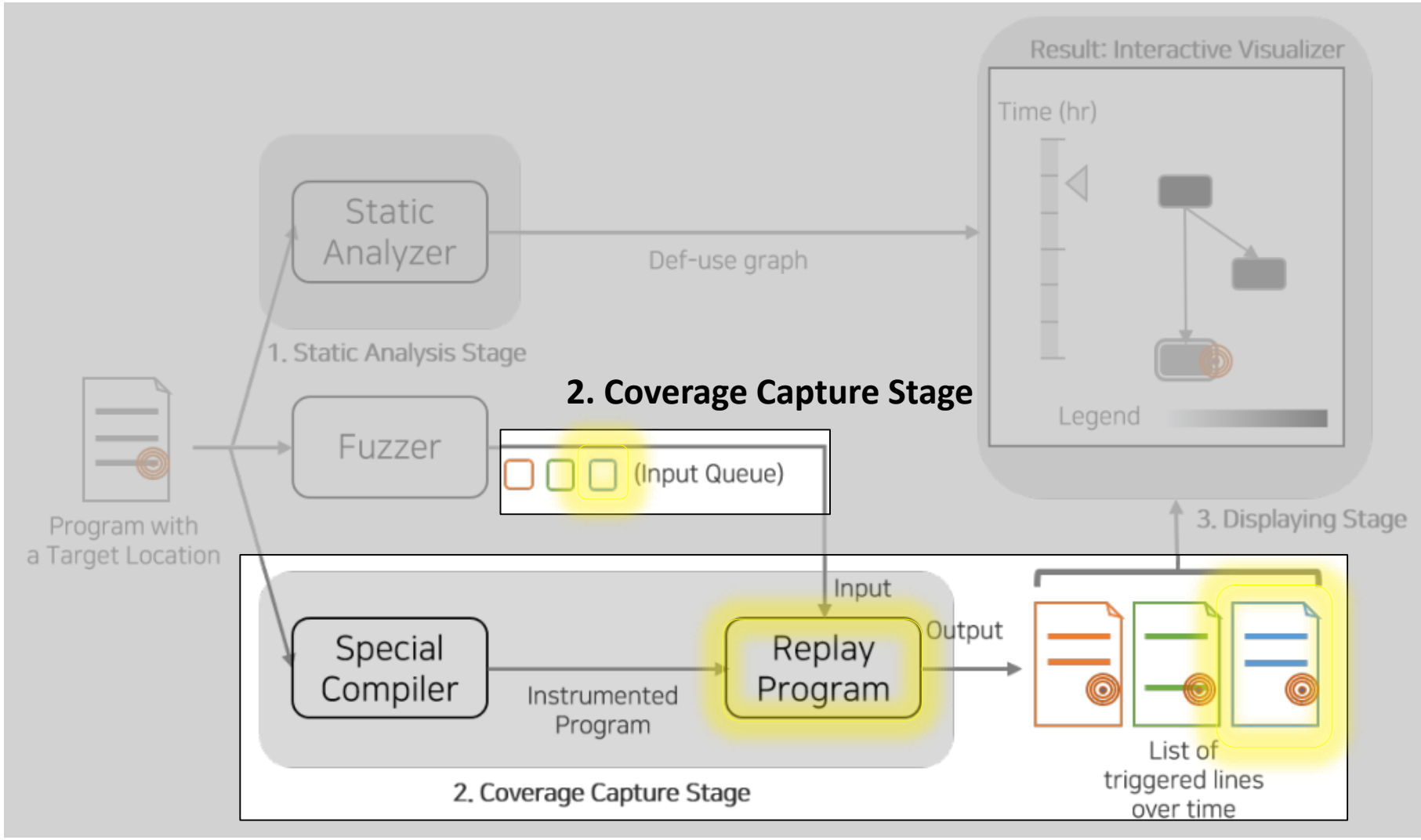
Visualizer Architecture



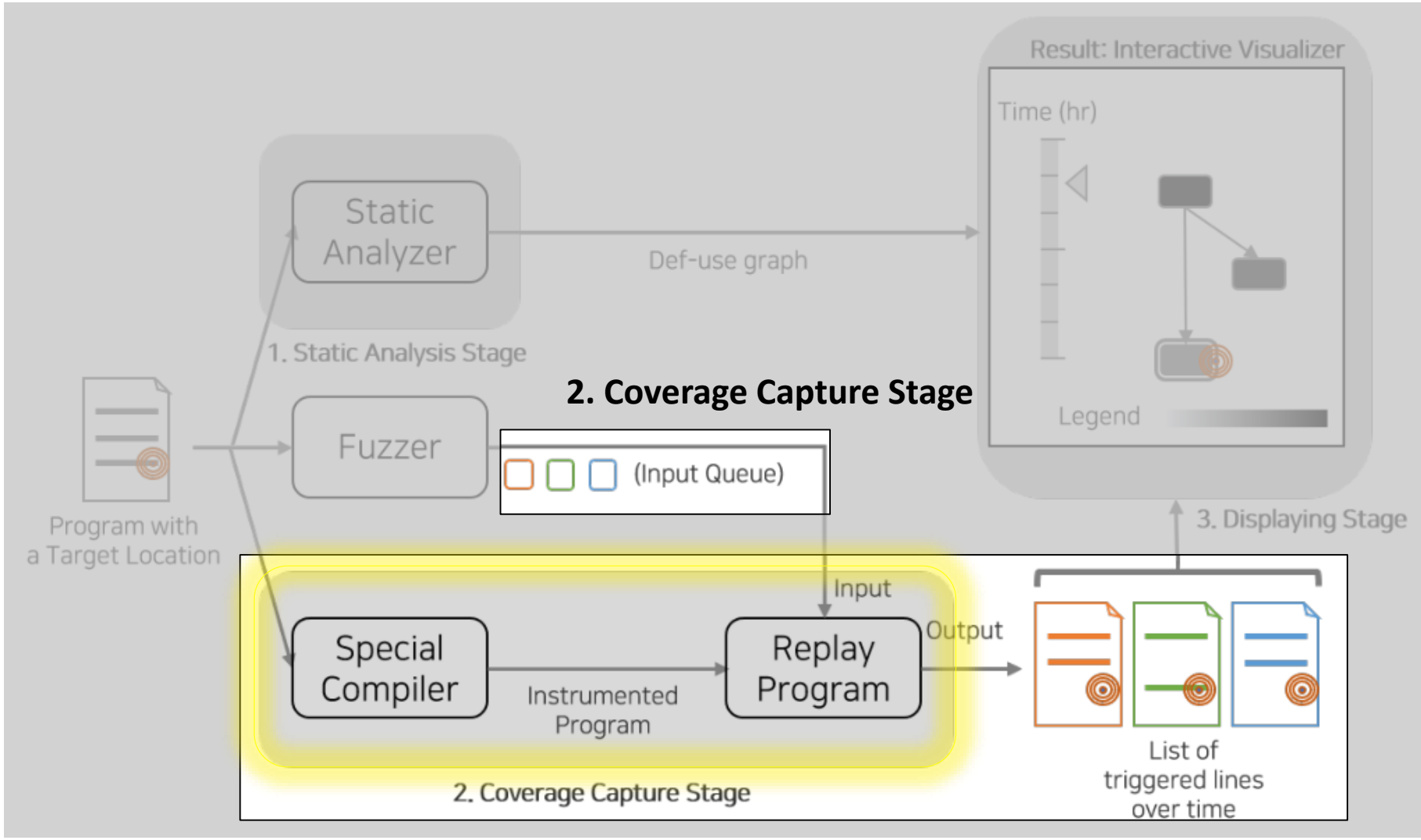
Visualizer Architecture



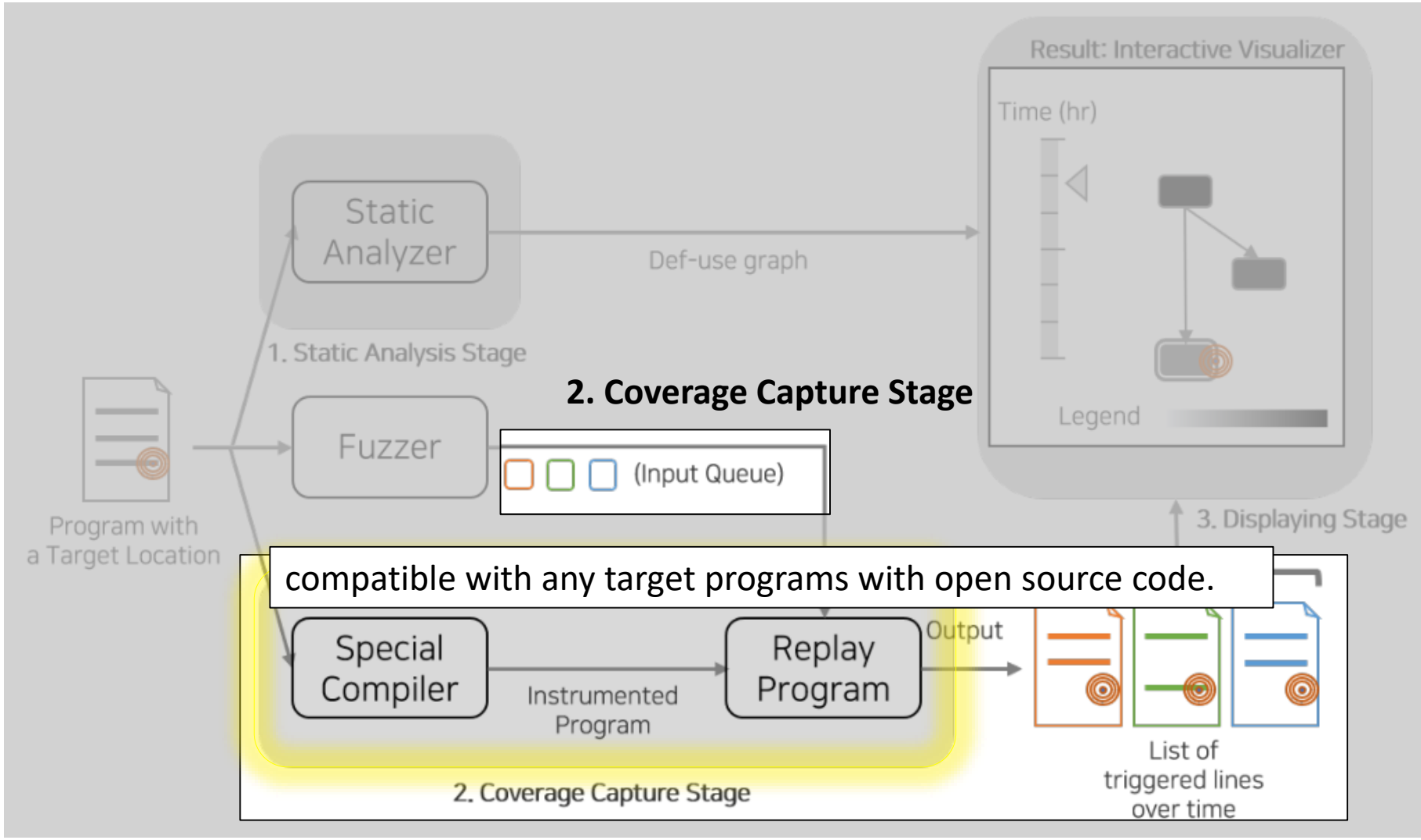
Visualizer Architecture



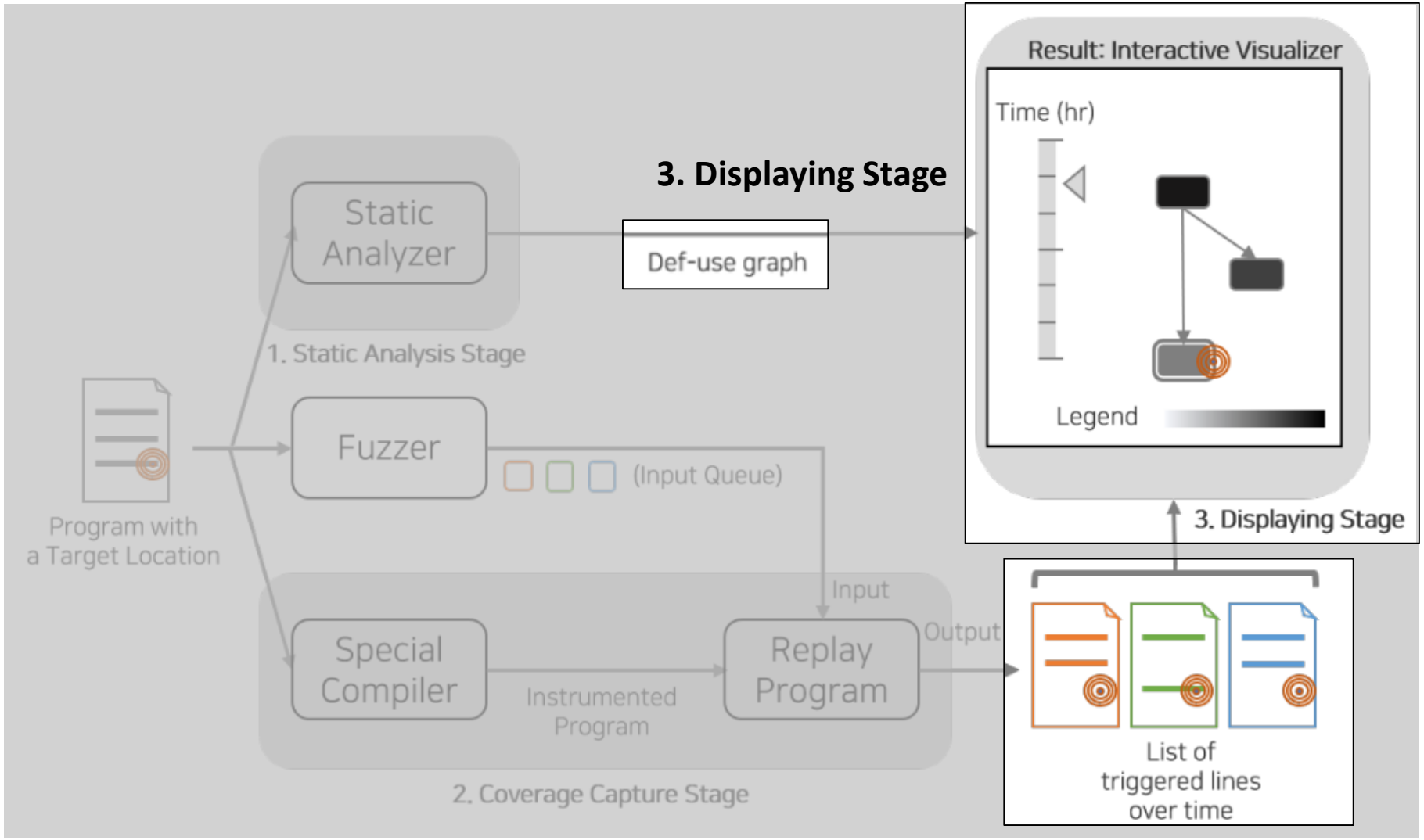
Visualizer Architecture



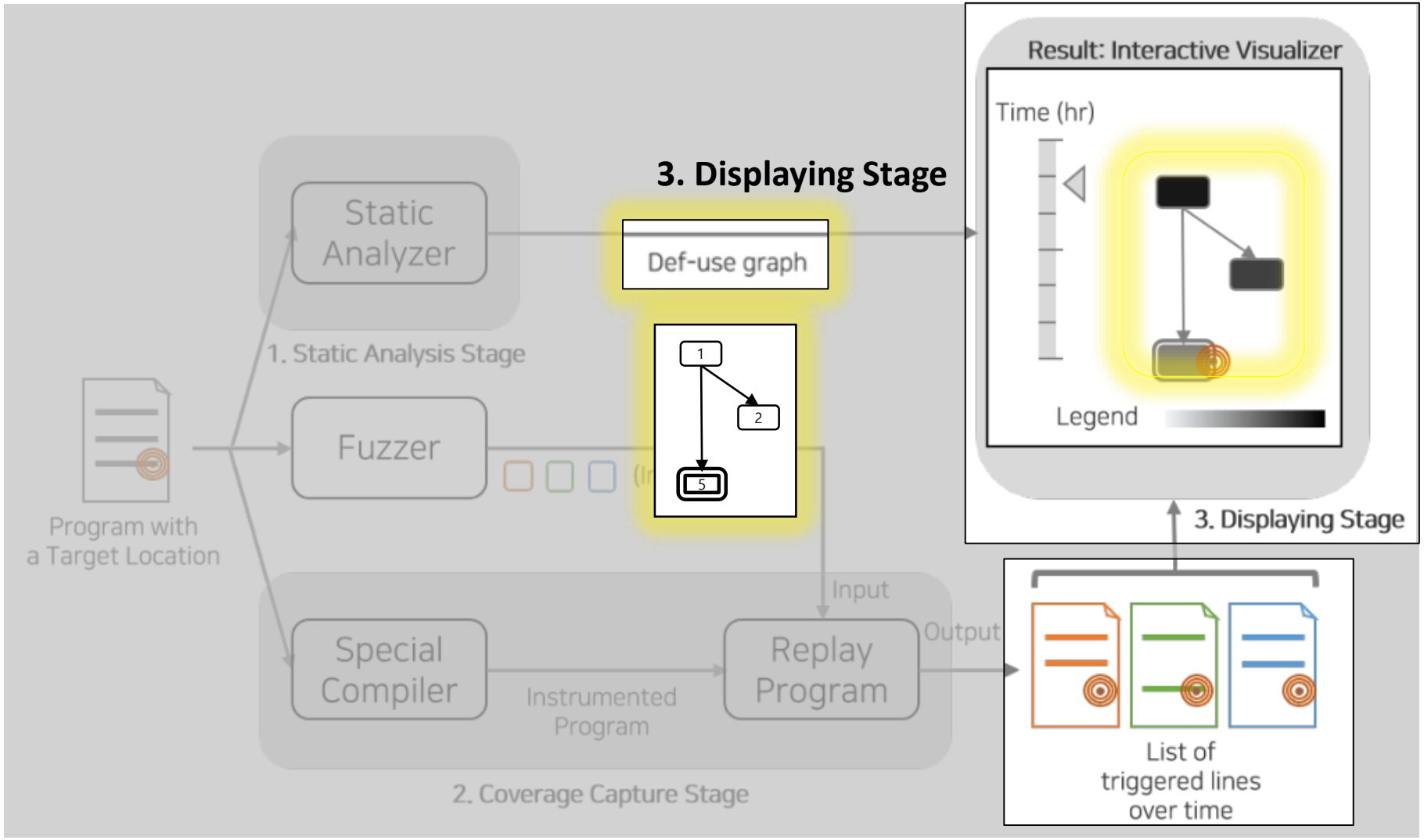
Visualizer Architecture



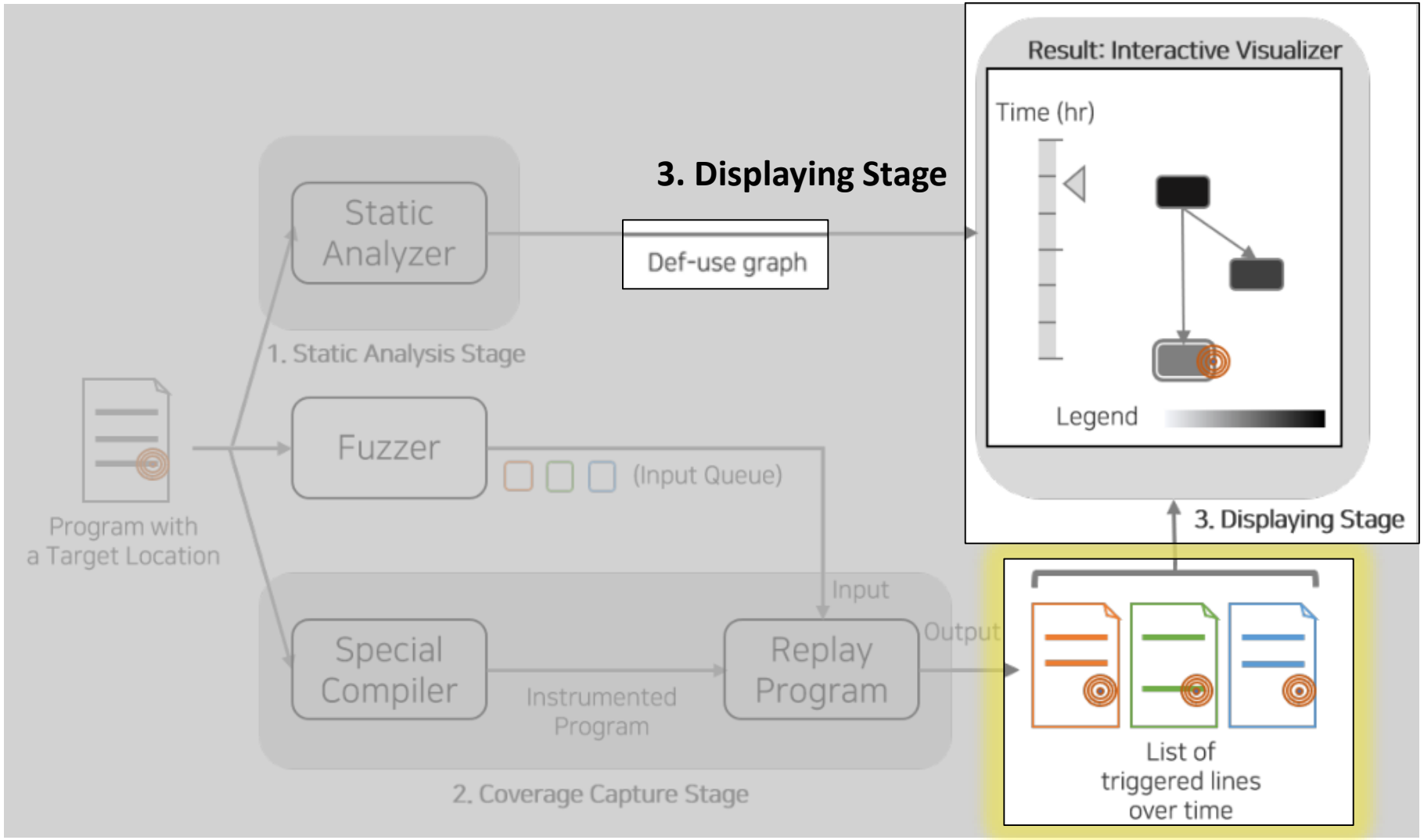
Visualizer Architecture



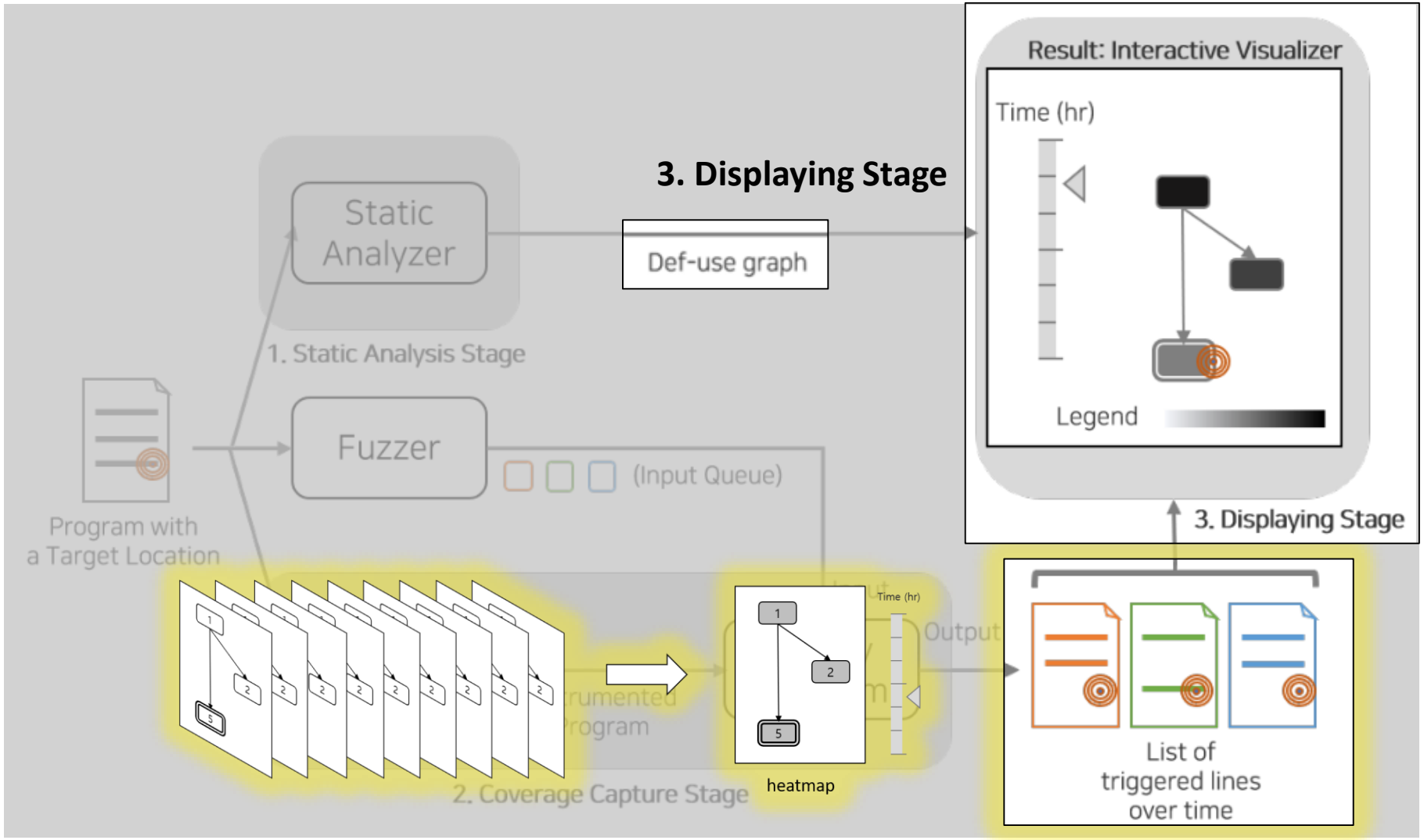
Visualizer Architecture



Visualizer Architecture

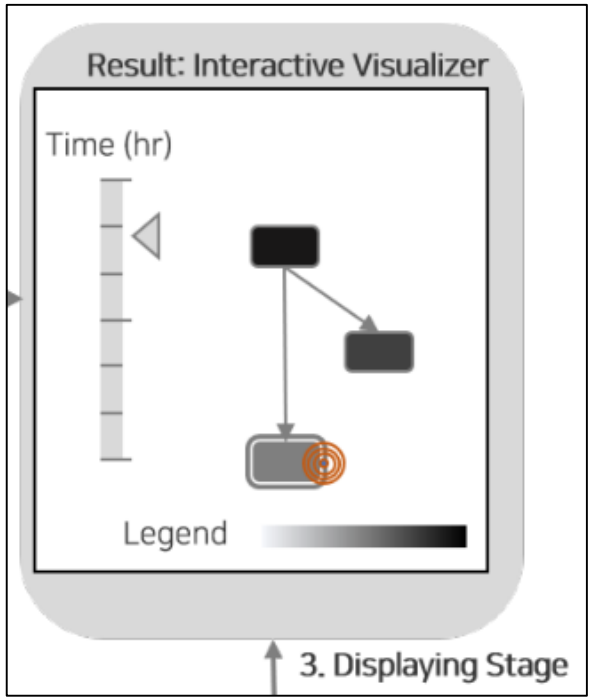


Visualizer Architecture

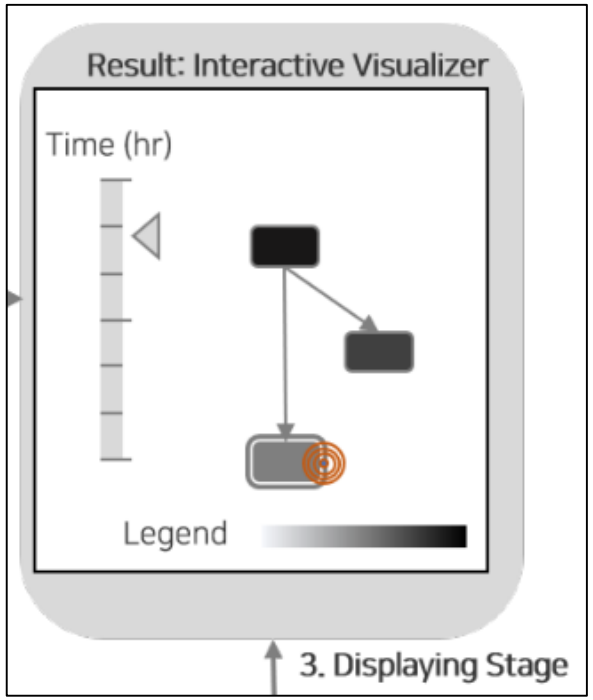


Visualizer Architecture

3. Displaying Stage

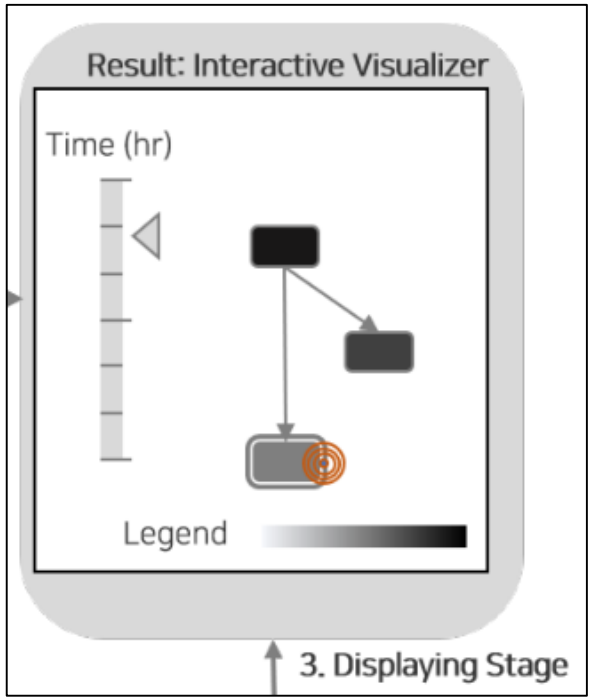


Visualizing more data



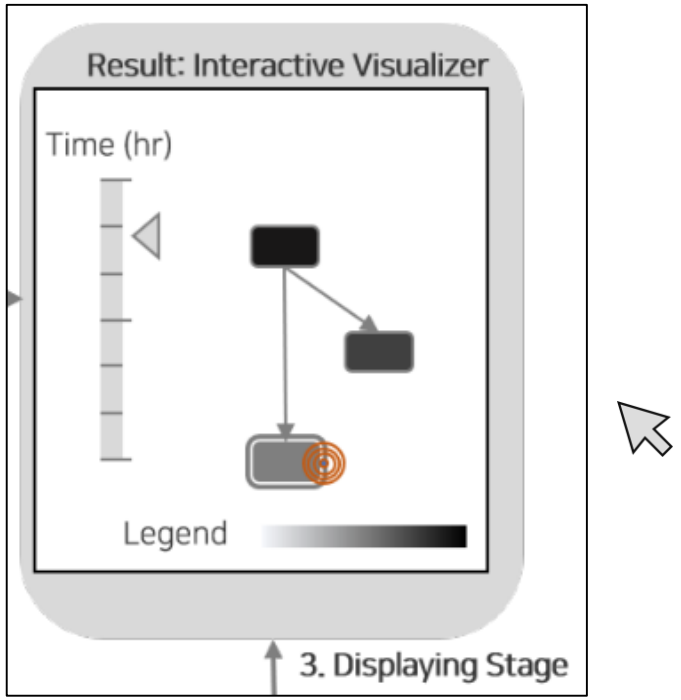
Visualizing more data

- More info becomes visible on click.



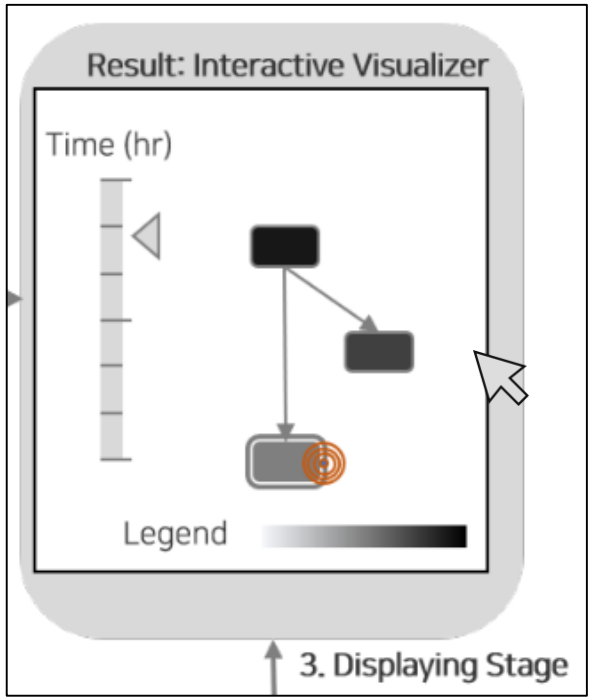
Visualizing more data

- More info becomes visible on click.



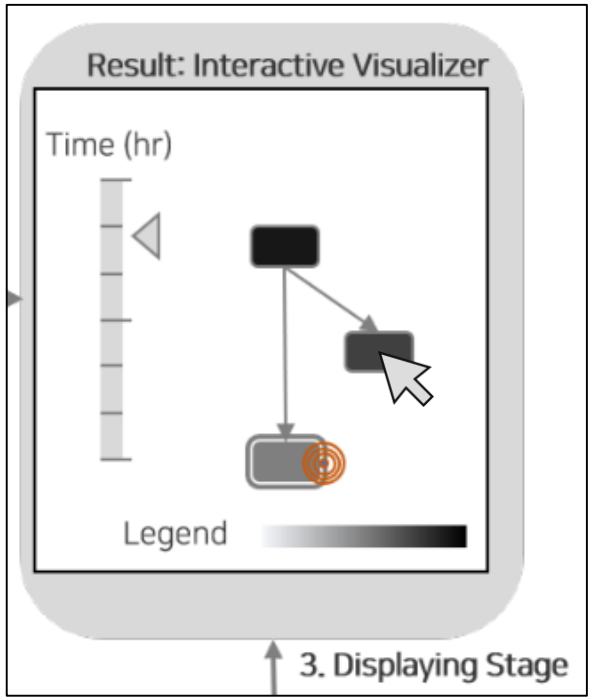
Visualizing more data

- More info becomes visible on click.



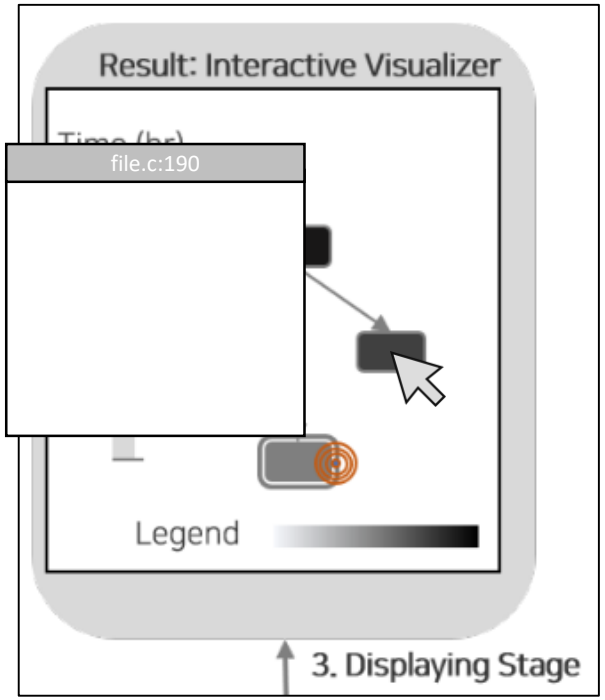
Visualizing more data

- More info becomes visible on click.



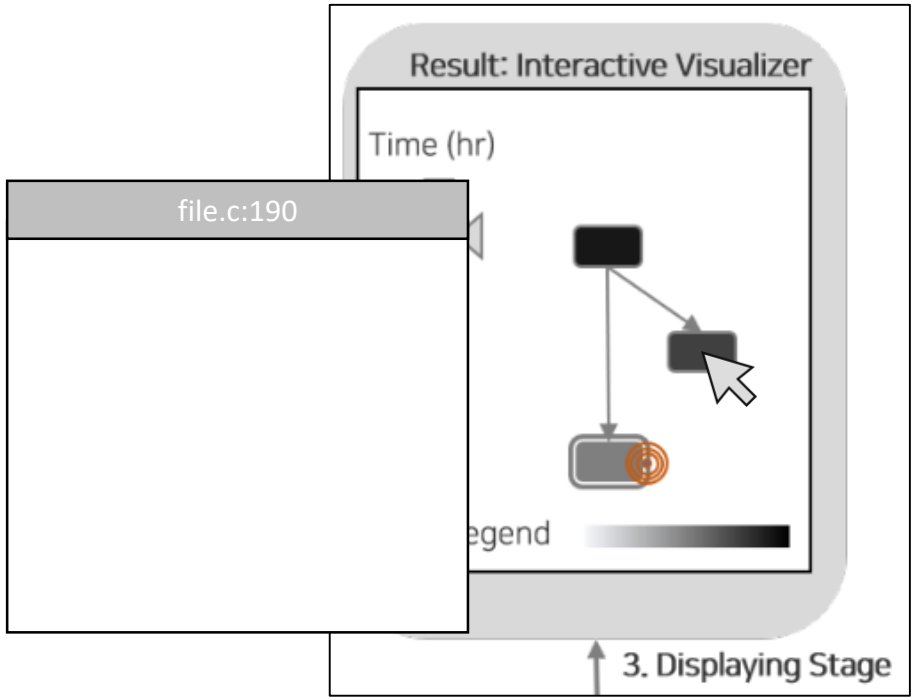
Visualizing more data

- More info becomes visible on click.



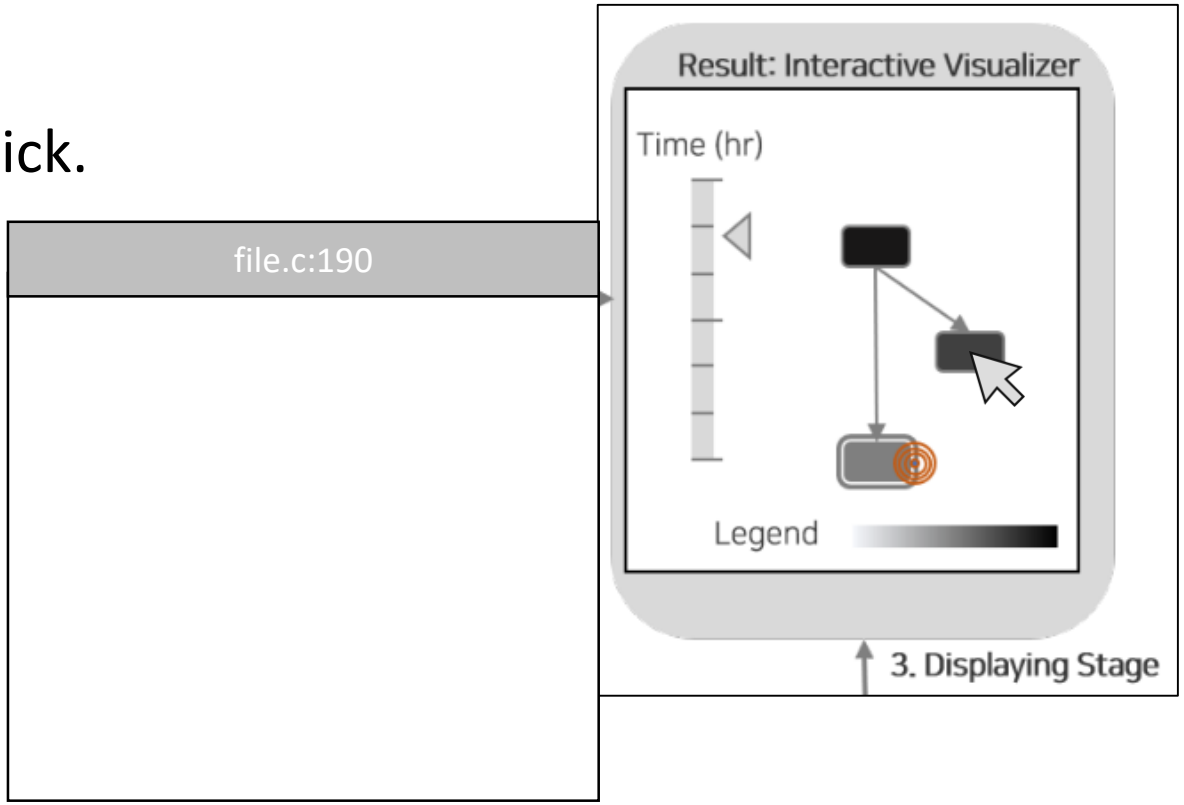
Visualizing more data

- More info becomes visible on click.



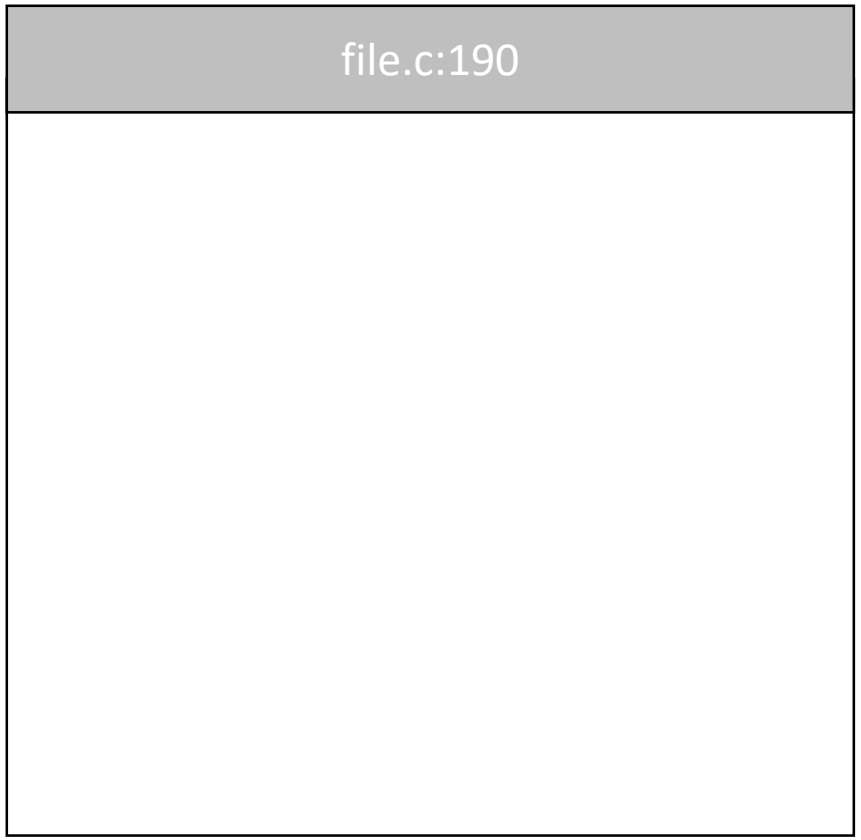
Visualizing more data

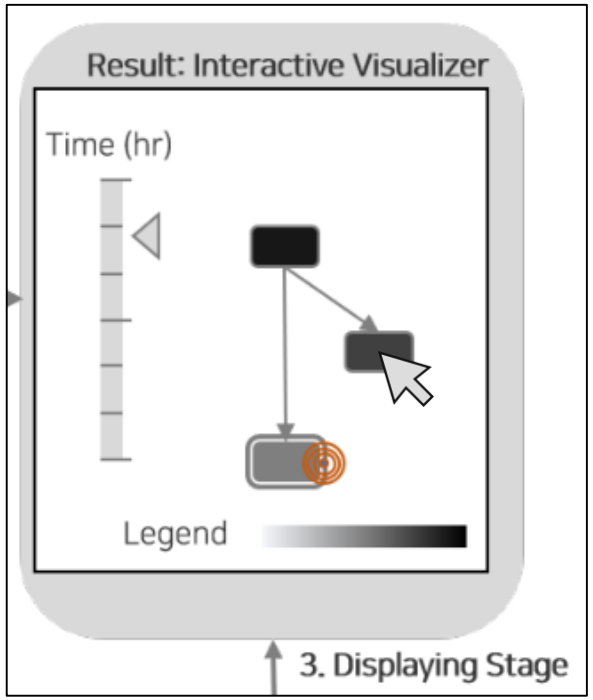
- More info becomes visible on click.



Visualizing more data

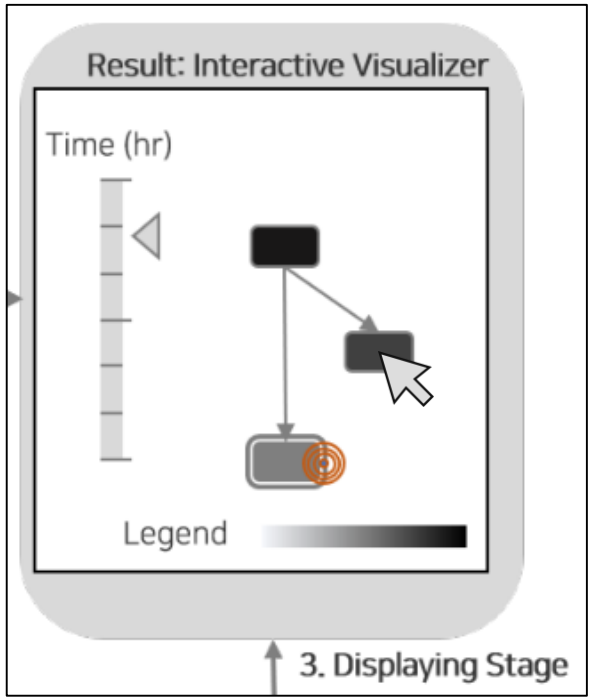
- More info becomes visible on click.





Visualizing more data

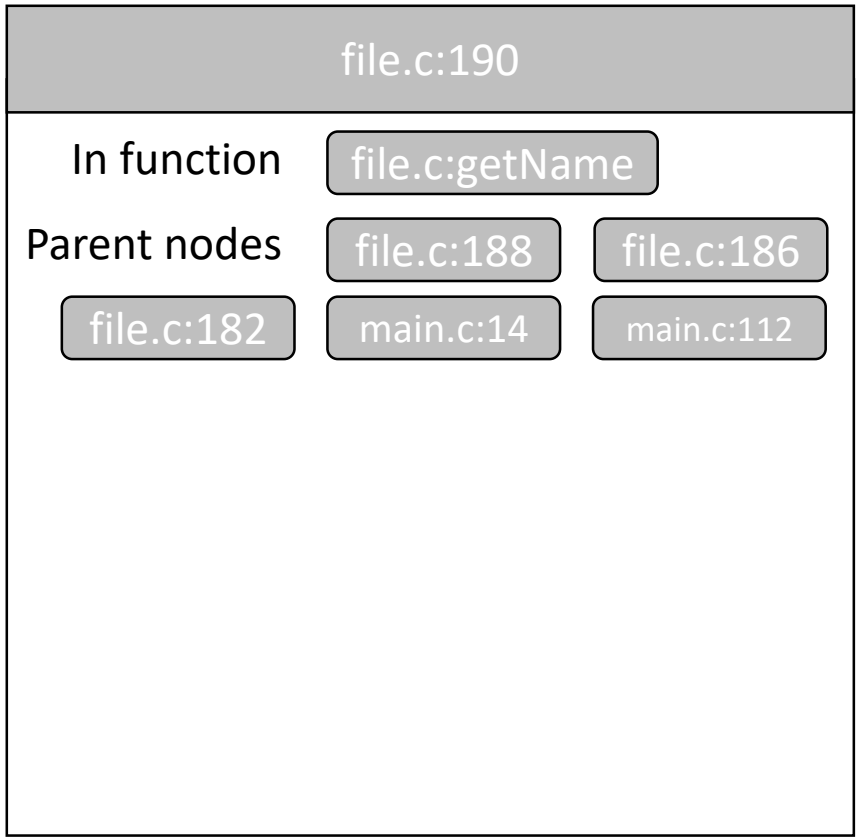
- More info becomes visible on click.
- Function name

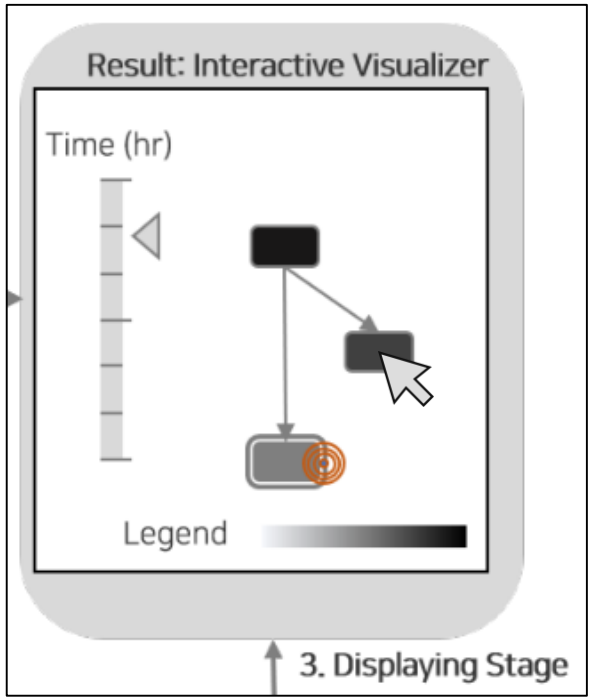


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes

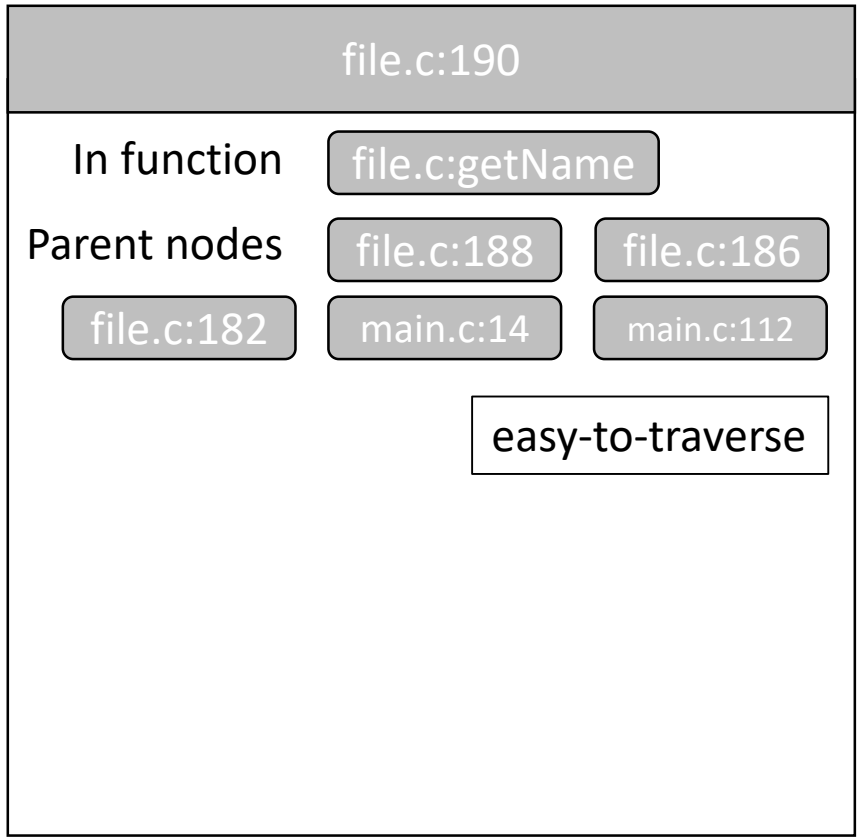


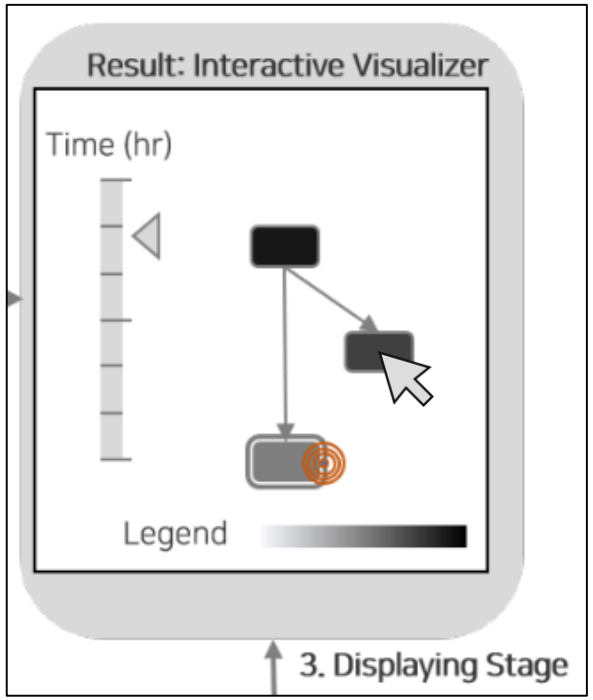


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes

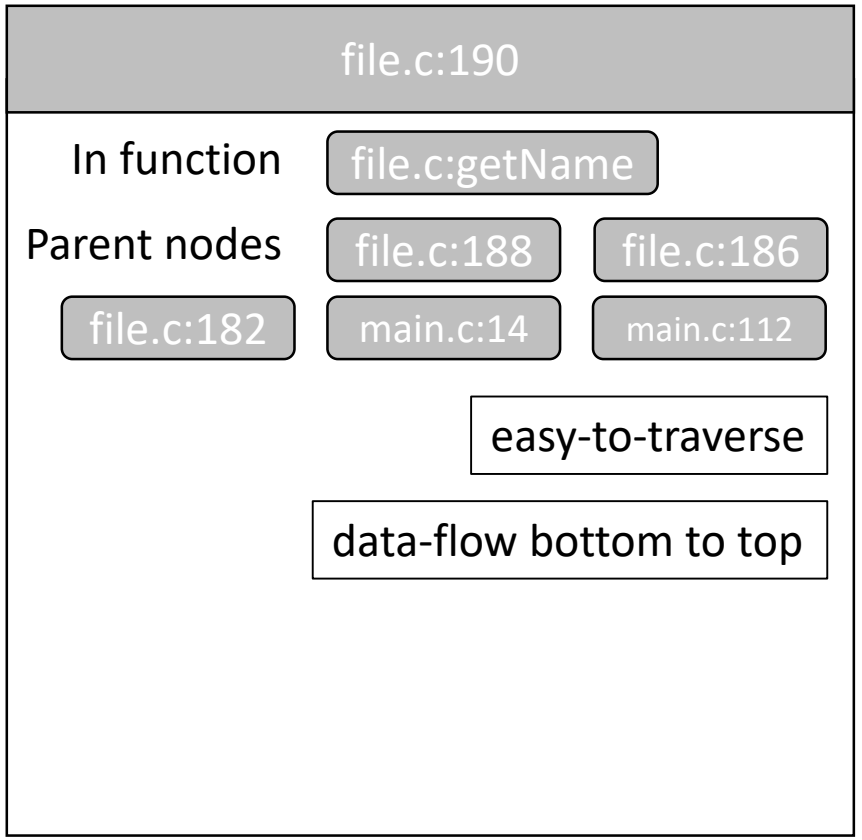


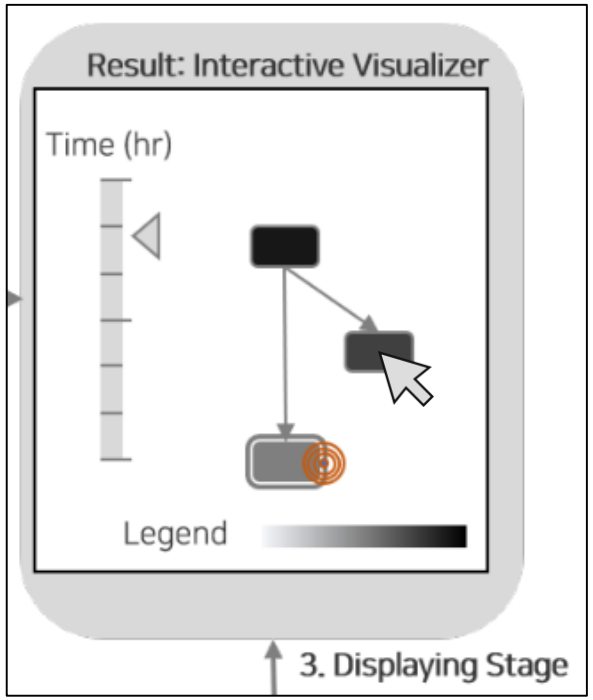


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes

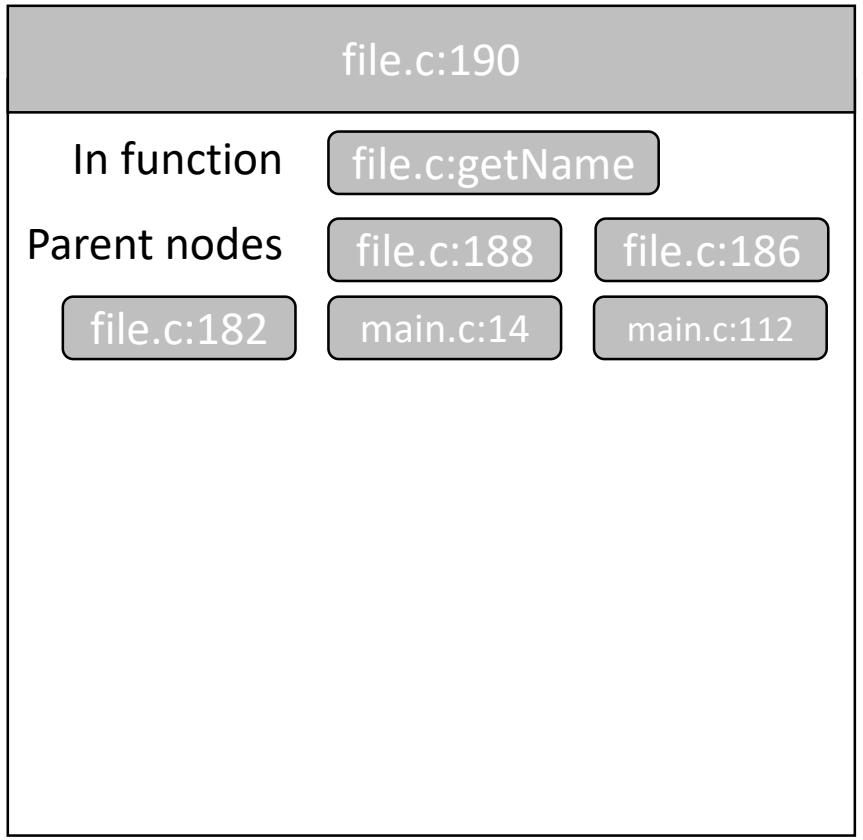


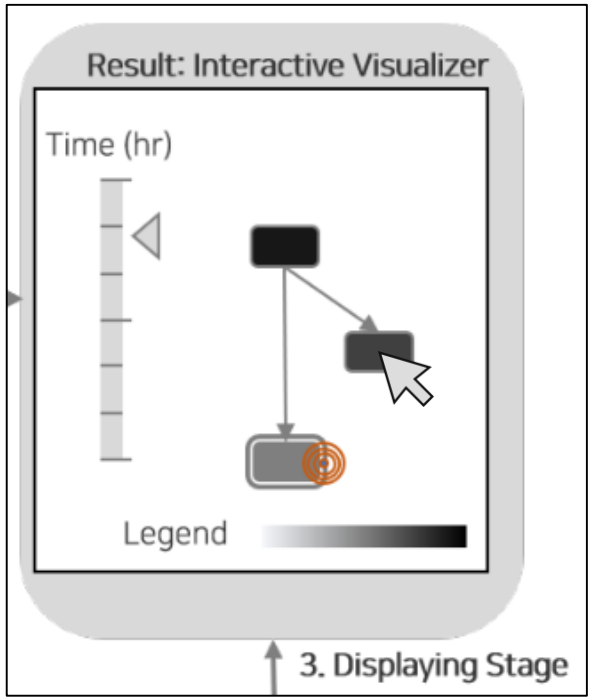


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes

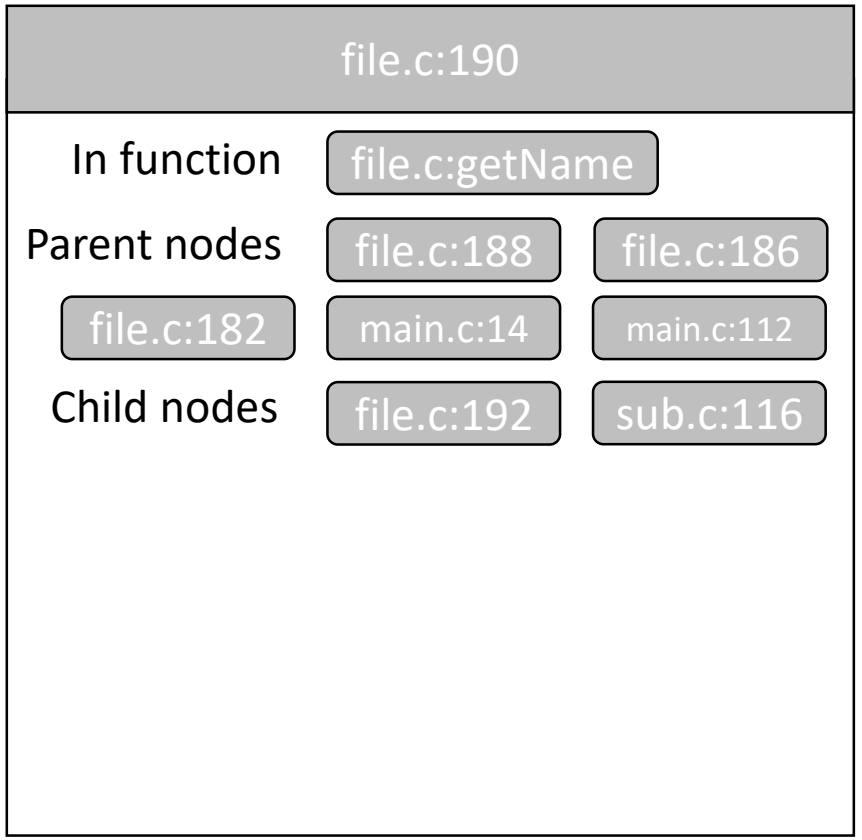


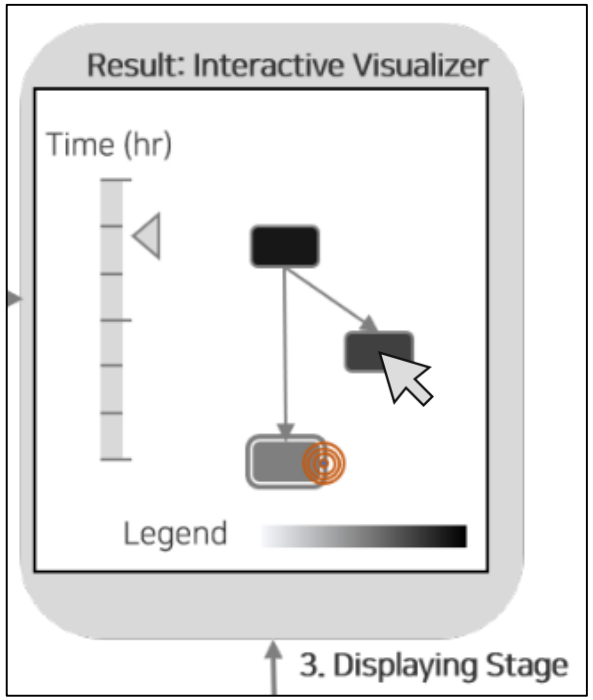


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes
- Child nodes

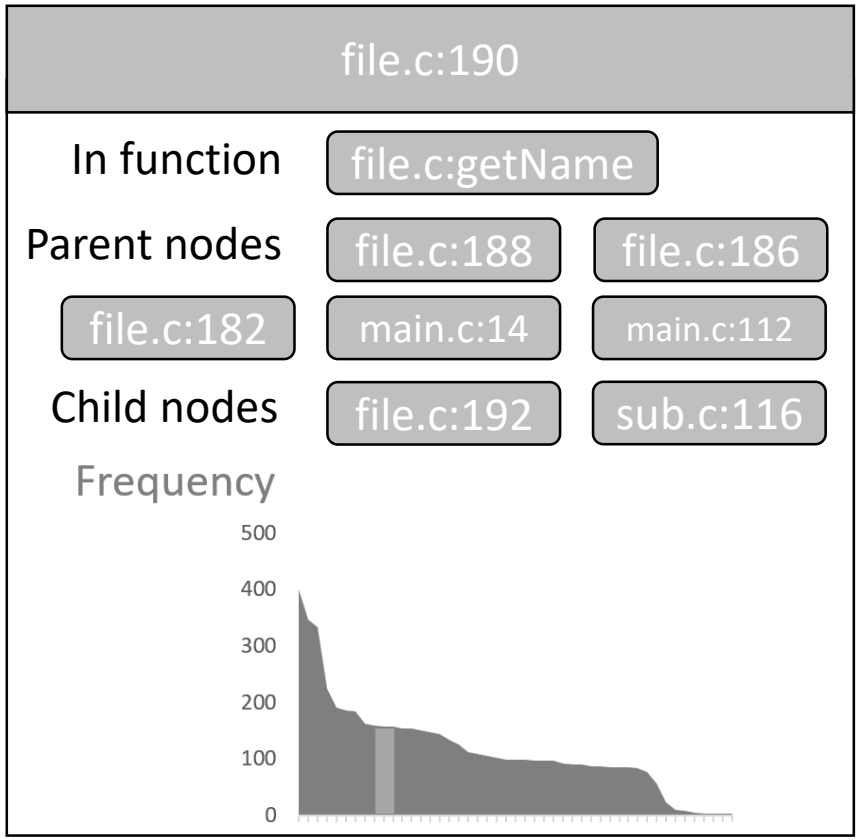


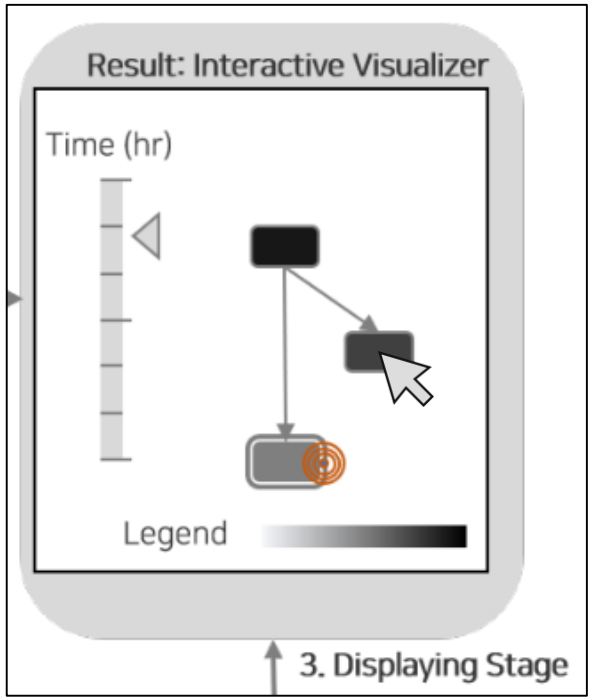


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes
- Child nodes
- Visit frequency

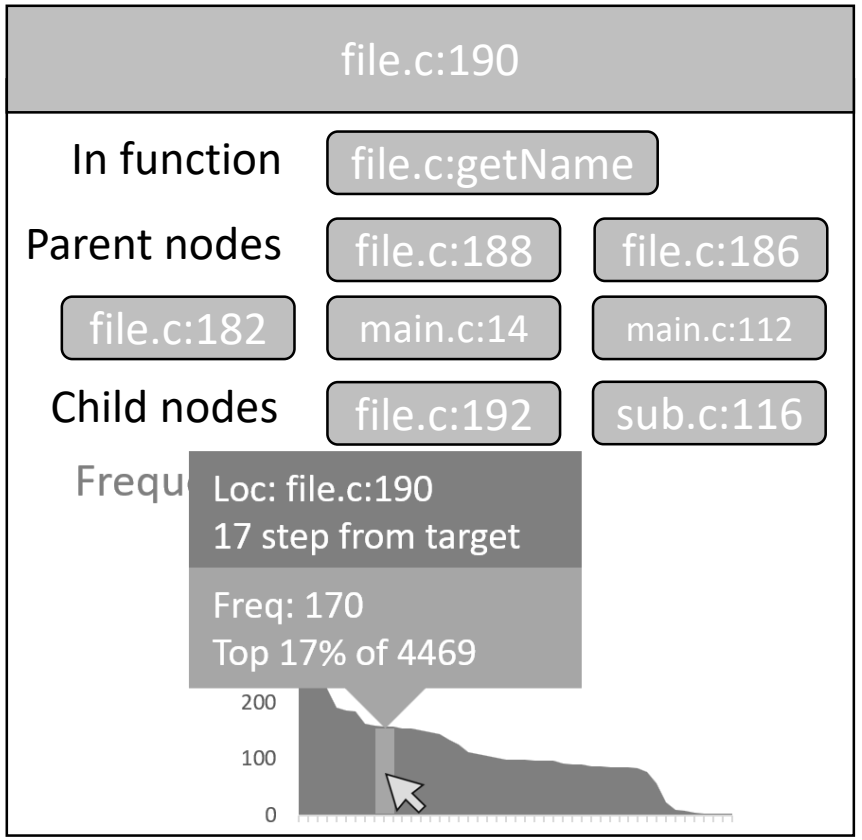


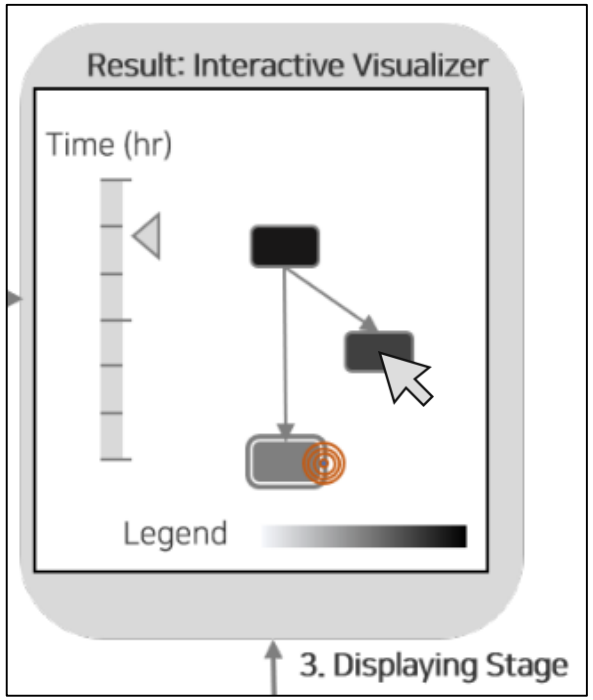


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes
- Child nodes
- Visit frequency

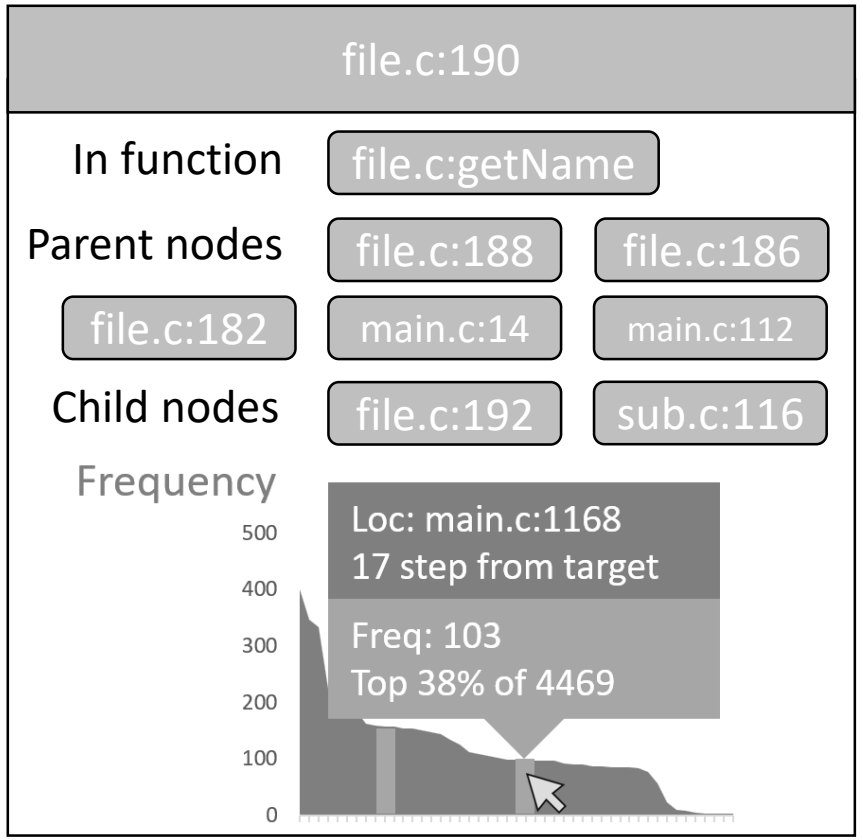


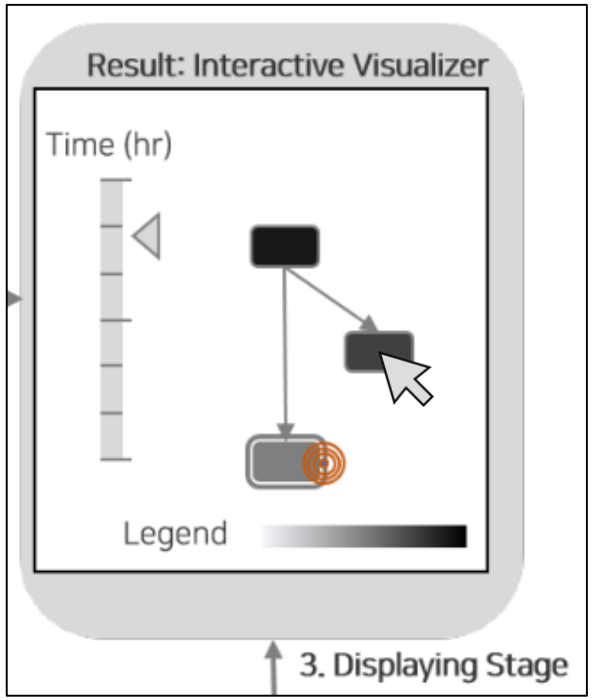


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes
- Child nodes
- Visit frequency

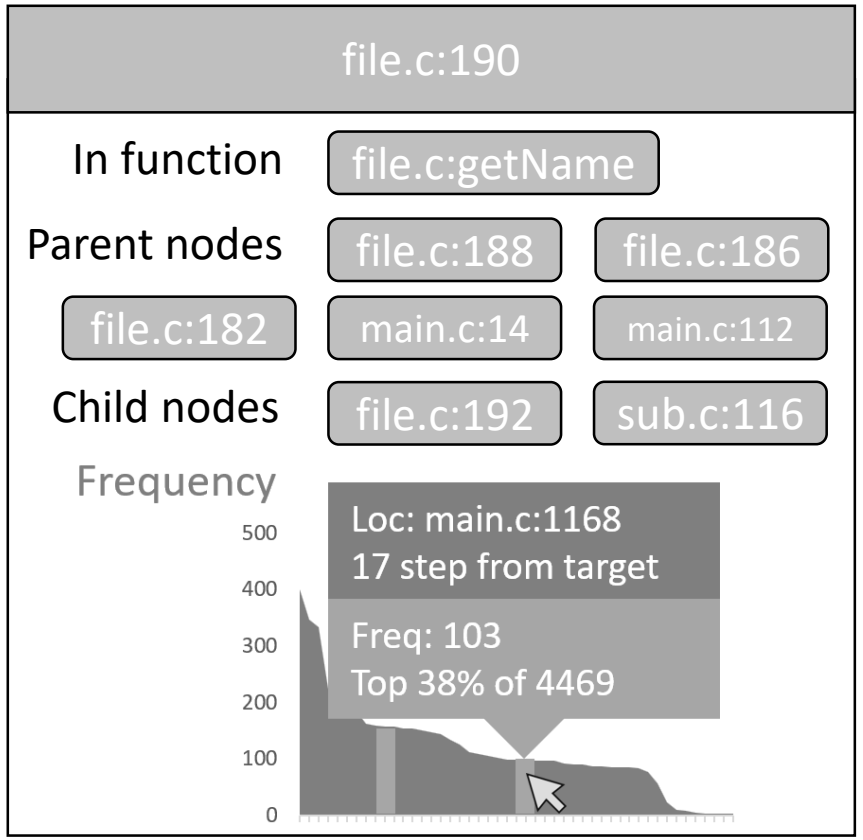


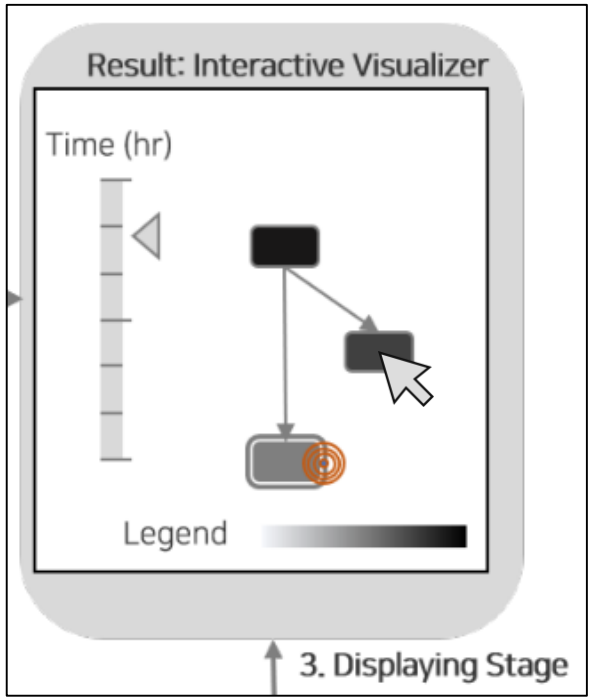


Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes
- Child nodes
- Visit frequency
- Time-to-reach



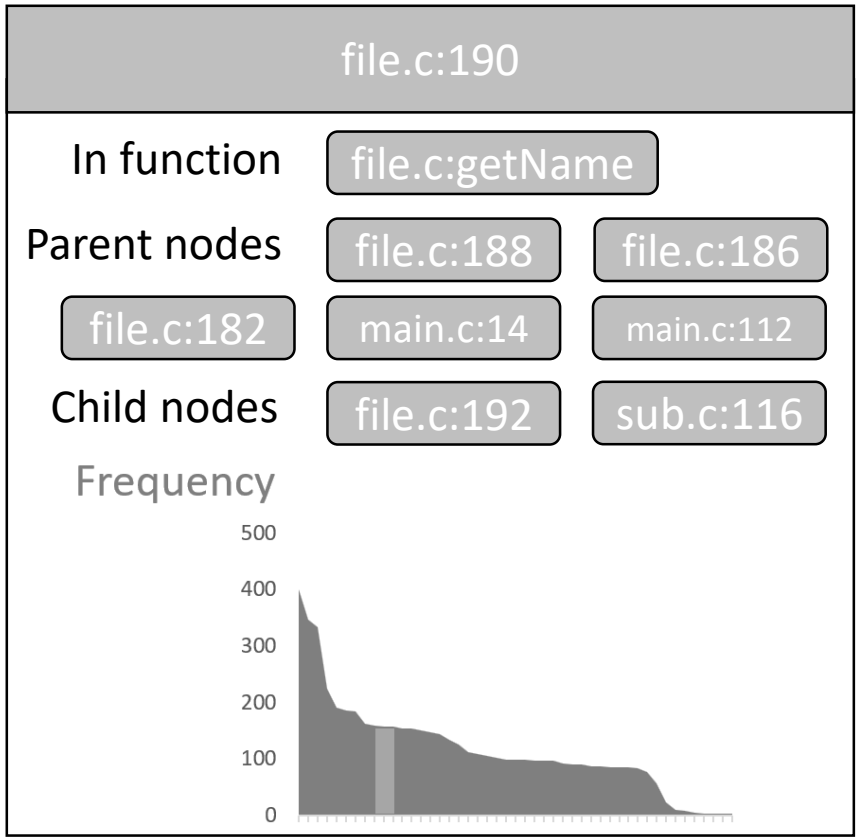


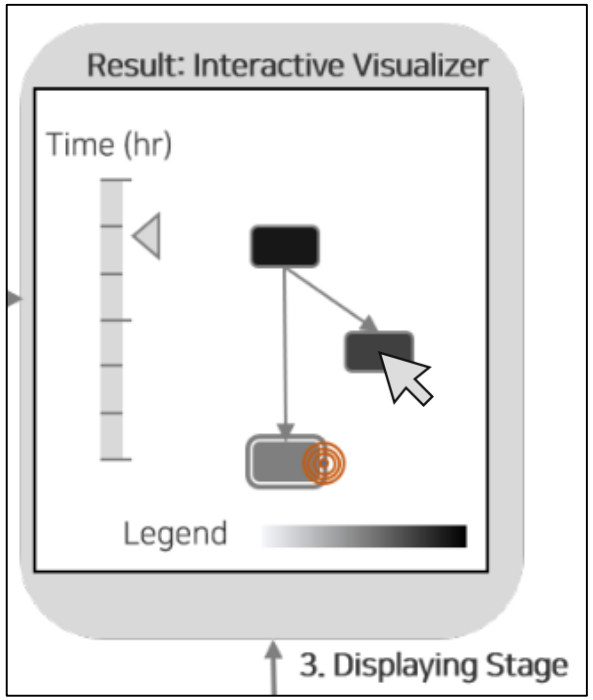
Visualizing more data

- More info becomes visible on click.

- Function name
- Parent nodes
- Child nodes
- Visit frequency
- Time-to-reach

…





Impact

Impact

- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.

Impact

- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.

Impact

- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.
- TopViz is **first** to aid directed fuzzing research.

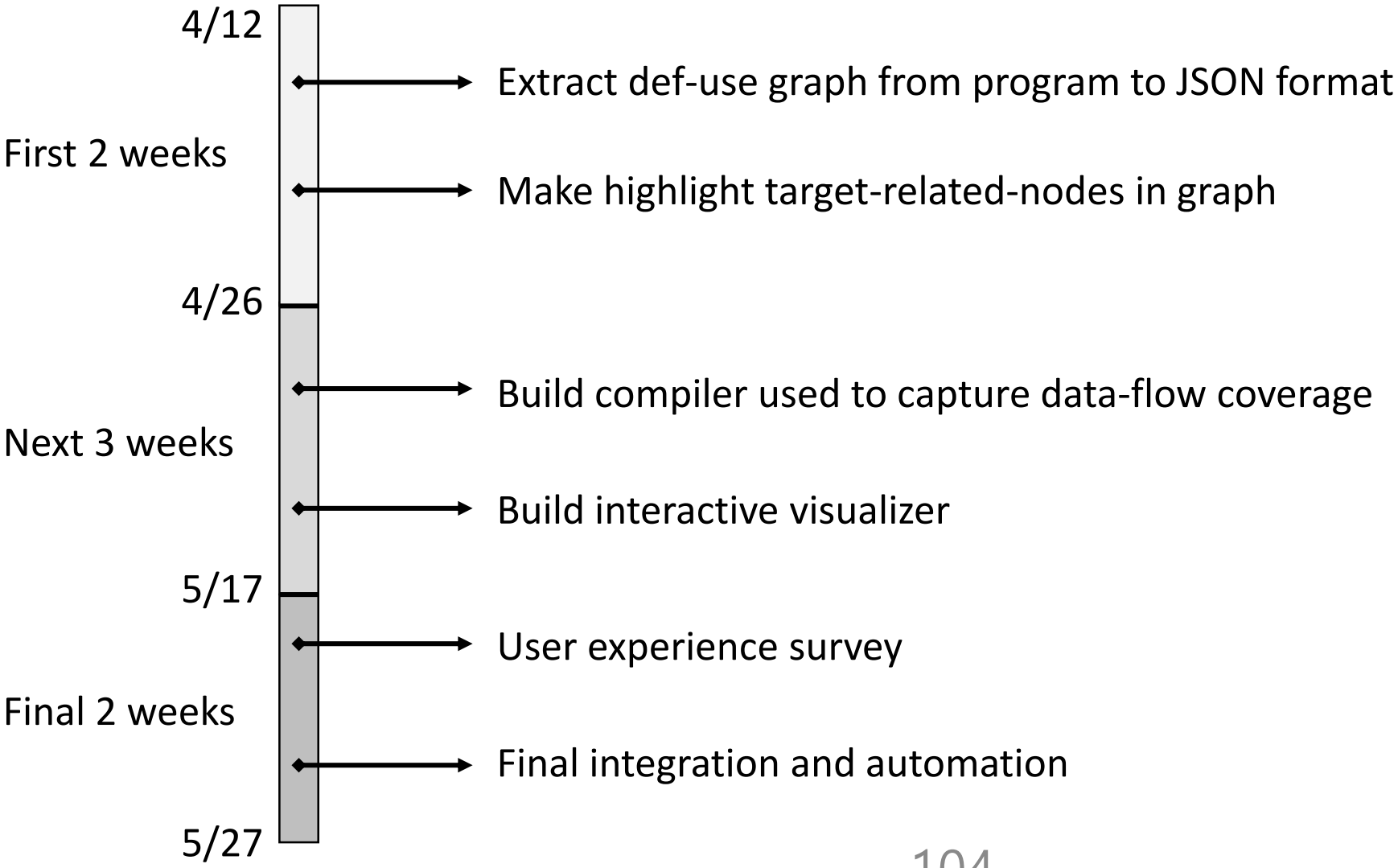
Impact

- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.
- TopViz is **first** to aid directed fuzzing research.
- TopViz is compatible with..
  - **any** fuzzing tools.

Impact

- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.
- TopViz is **first** to aid directed fuzzing research.
- TopViz is compatible with..
  - **any** fuzzing tools.
  - **any** target programs with open source code.

Time table and Milestones



Time table and Milestones

4/12

Extract def-use graph from program to JSON format

First 2 weeks

Make highlight target-related-nodes in graph

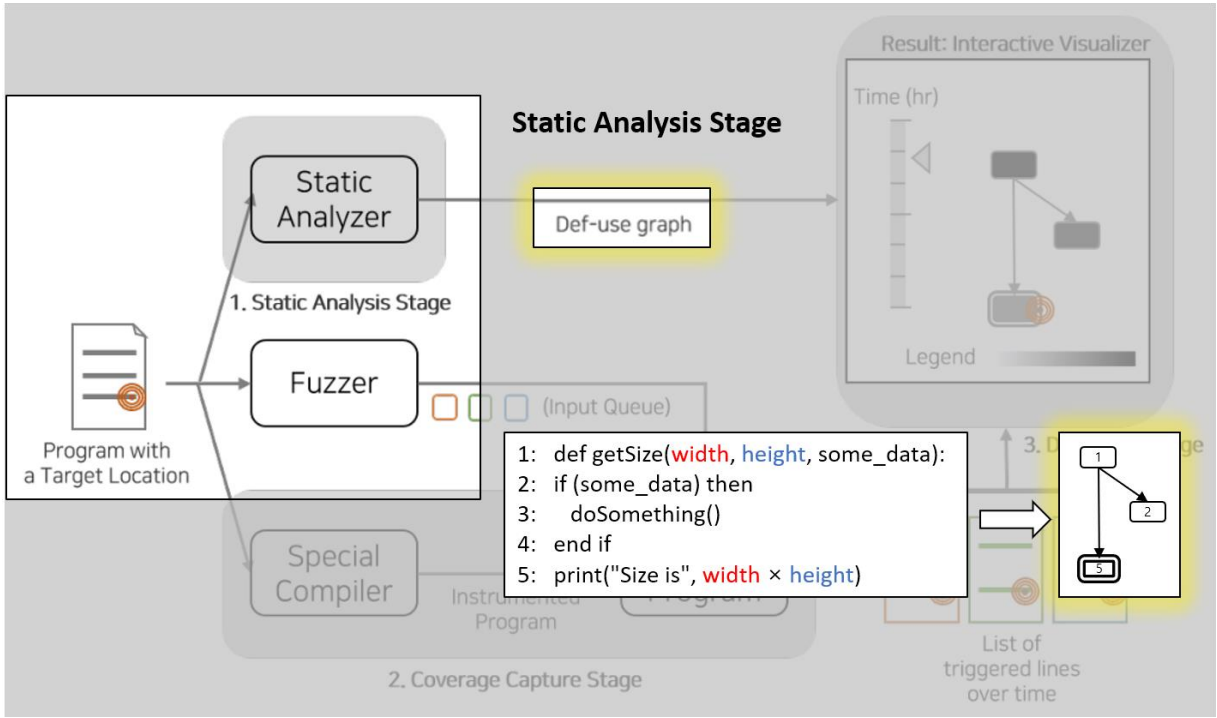
4/26

Next 3 weeks

5/17

Final 2 weeks

5/27



Time table and Milestones

4/26

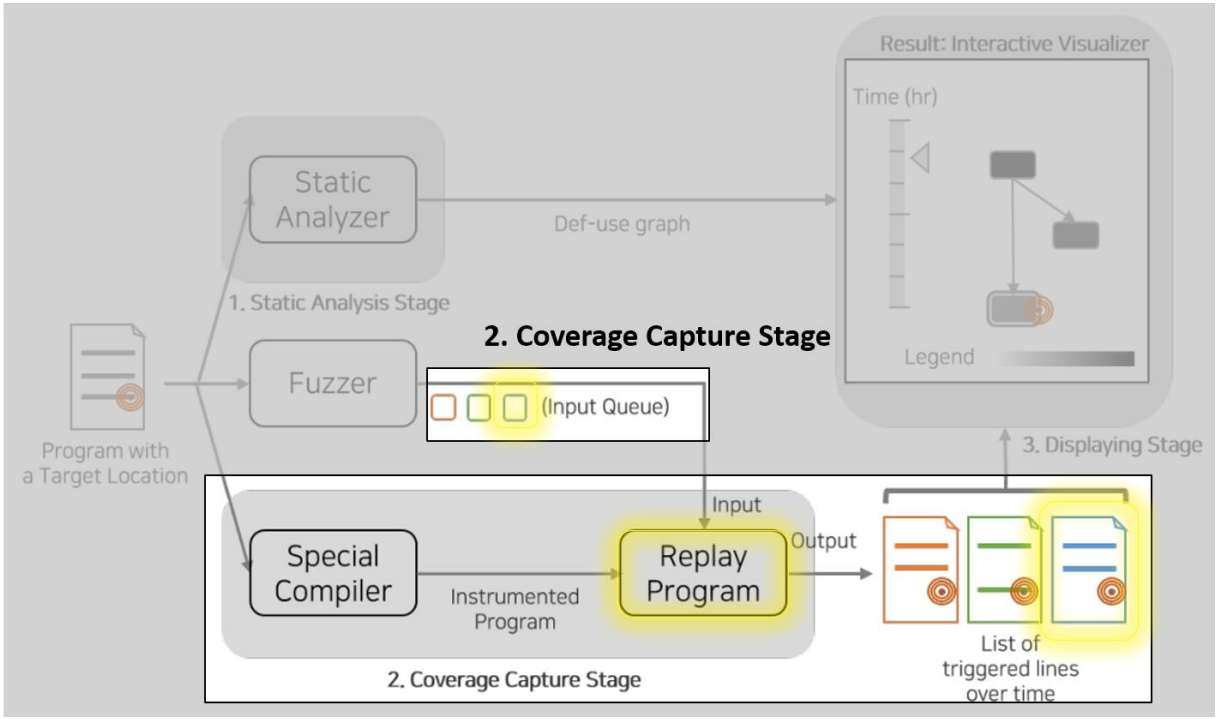
Build compiler used to capture data-flow coverage

Next 3 weeks

5/17

Final 2 weeks

5/27



Time table and Milestones

4/26

Build compiler used to capture data-flow coverage

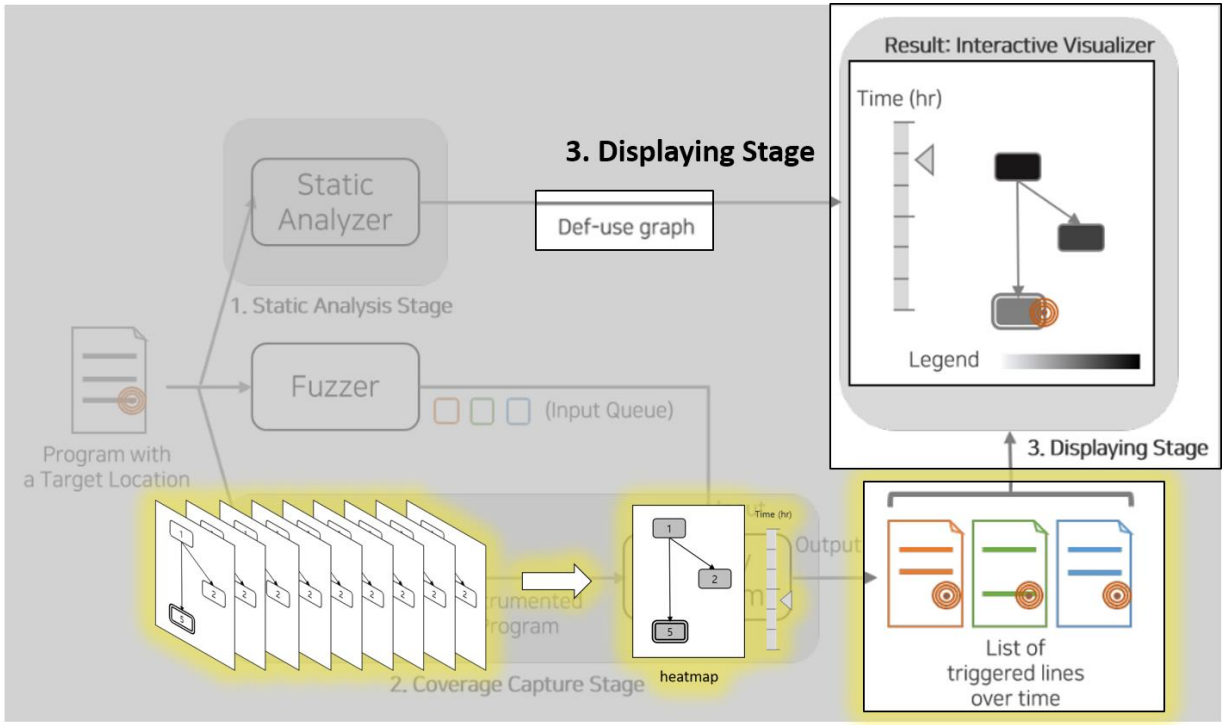
Next 3 weeks

Build interactive visualizer

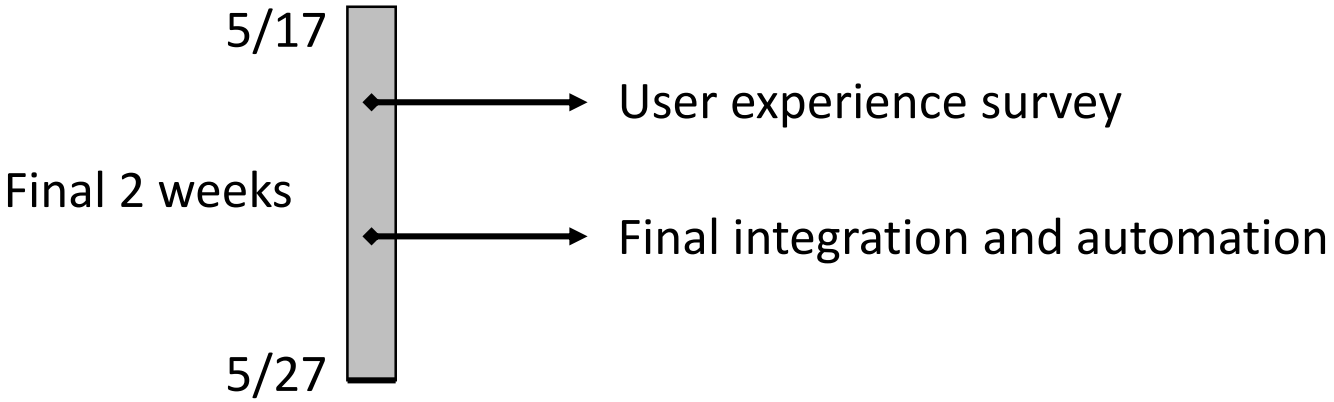
5/17

Final 2 weeks

5/27



Time table and Milestones



Conclusion

- This research builds a brand-new visualization tool, **TopViz**.
- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.
- TopViz is **first** to aid directed fuzzing research.
- TopViz is compatible with..
  - **any** fuzzing tools.
  - **any** target programs with open source code.
- TopViz's development will spend 7 weeks, with **user experience survey**.