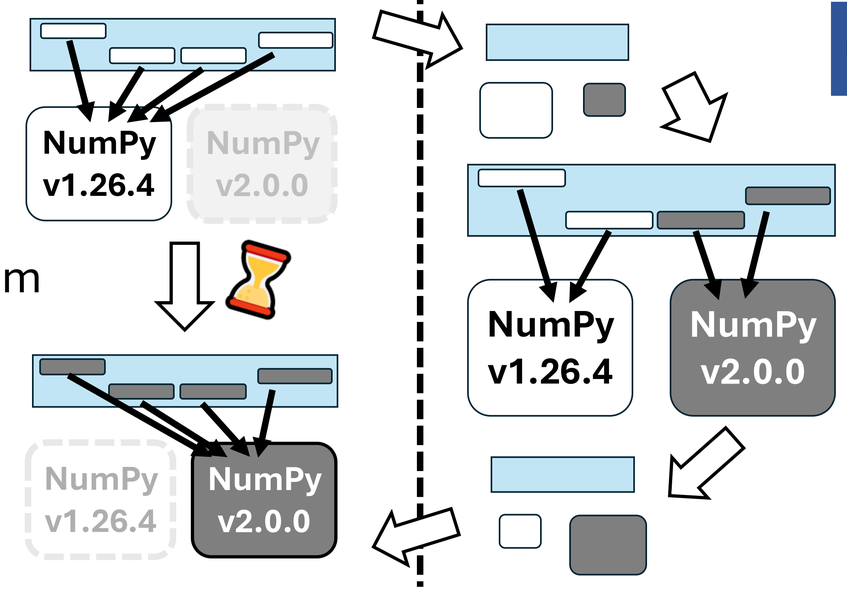
Vython: a Language with Dynamic Version Checking for Gradual Updating

Satsuki Kasuya
supervised by Yudai Tanabe and Hidehiko Masuhara
Institute of SCIENCE TOKYO
formerly Tokyo Institute of Technology

Motivation

Package-wise updates are *NOT flexible*
for fixing broken behaviors in downstream

- **All call sites** of the package must be targets for refactoring
- **Untraceable causes** of broken behaviors due to upstream incomp.



Goal

Gradual Updating
[Tanabe+'21,'23, Lubis+'22]

1. **Selective update** of call sites
2. **Confirm behavior by running**

Comparison with Related Work

On **statically-**typed languages	On **dynamically-**typed languages
• VL[Tanabe+'21,'23] • BatakJava[Lubis+'22]	*This research*

Challenges

RQ 1: How to ***support multiple versions*** **in one program**?

RQ 2: How to **deal with *value incompatibilities***?
as exemplified by case study Artifact 2

Proposal: Vython

Feature 1: Can **specify a class version** at instantiation

Feature 2: ***Dynamically tracing versions*** to detect conflicts & suggest refactoring hints

Limitation: require versions for ***all*** class declarations & instantiations.

Upstream

```
1 class NumPy!1.26.4():
2   def solve(self, A, B):
3     return res
```

```
1 class NumPy!2.0.0():
2   def solve(self, A, B):
3     return res
```

Annotate

```
1 class SciPy!1.12.0():
2   def place_poles(self, A, B, poles):
3     return NumPy!1.26.4().solve(a, b)
```

Downstream

```
1 def my_place_poles(A, B, poles):
2   return NumPy!2.0.0().solve(a, b)
3 array_equal(
4   my_place_poles(A, B, poles),
5   SciPy!1.12.0().place_poles(A, B, poles) )
```

Conflict!

Upstream developers specify

[Incompatibility] Removed ambiguity when broadcasting in np.solve (gh-25914)
The broadcasting rules for `np.solve(a, b)` were ambiguous when `b` had 1 fewer dimensions than `a`. This has been resolved in a backward-incompatible way …

[Refactoring Hints]
The old behavior can be reconstructed by using `np.solve(a, b[…, None])[…, 0]`.
https://numpy.org/devdocs/release/2.0.0-notes.html

Downstream developers are notified

Found the cause of incompatibility! **Refactoring & PR**

```
> Warning: Incompatible values is used together.
 4 | my_place_poles(A, B, poles)
   | ^ --> [[2. 3. ] [2.2 2.6]]
   |       ^ from solve in NumPy-v2.0.0/numpy.py:2:3
 5 | SciPy!1.12.0().place_poles(A, B, poles)
   | ^ --> [[[2.2 1.2] [2.4 4.4]] [[4. 2.8] [1. 2.4]]]
   |       ^ from place_poles in SciPy-v1.12.0/scipy.py:3:11
   |       ^ from solve in NumPy-v1.26.4/numpy.py:2:3
```

… and ***Incompatibilities & refactoring hints*** from

Implementation Artifact 1

```
class NumPy_v_2.0.0:
  @version_initializer
  def __init__(self, ~): …
  @version_checker
  @version_propagator
  def solve(self, a, b):
    incompatible(res)

array_equal(
  my_place_poles(..),
  .. )
```

- **Object tracks its source versions.**

```
Array {
  'value': ..,
  'ver-info': 1100
}
```

- Feature 2 is compiled into **bitwise operations** in helper functions.

Current Results

- [x] POC Implementation
- [x] Preliminary Evaluation
- [ ] Compatibility Manager
- [ ] Automatic Feedback Generation
- [ ] Designing Surface Language
- [ ] Case Study

Artifact 1
Vython transpiler

Artifact 2
Case study on NumPy value incompatibilities

Performance Evaluation (preliminary)

Acceptable perf. for debugging despite a prototype
cf. ≒ **~16x** overhead in DynaPyt[Egnbail+'22]