Weak scalars proposal

NEP 50

What is the problem?

- NumPy has “value-based” promotion:

```
np.array([1], dtype=np.float32) + 4.5 -> float32
np.array([1], dtype=np.float32) + 4e200 -> float64
np.array(1., dtype=np.float32) + 4.5 -> float64
```

- This depends on it being “scalar” (zero dimensional):
  - All scalars (even 0-D arrays) often act as if they have no type attached!

```
np.array(1., dtype=np.float32) + 4.5 -> float64
```

- Even more confusing for integers!

Why is it like this?

```
np.array([1], dtype=np.float32) + 4.5 -> float32
```

- This is arguably convenient when working with specific dtypes!

```
np.array([1], dtype=np.float32) + np.float32(4.5) -> float32  🙁
```

Why is it like this?

```
np.array([1], dtype=np.float32) + 4.5 -> float32
```

- Convenient when working with specific dtypes (float32, int32)!

```
np.array([1], dtype=np.float32) + np.float32(4.5) -> float32
```
🙁

- Also: We have a mess around 0-D arrays and scalars
  (but it is tedious to distinguish!)

Numeric literals should be special! But what else and to what degree?

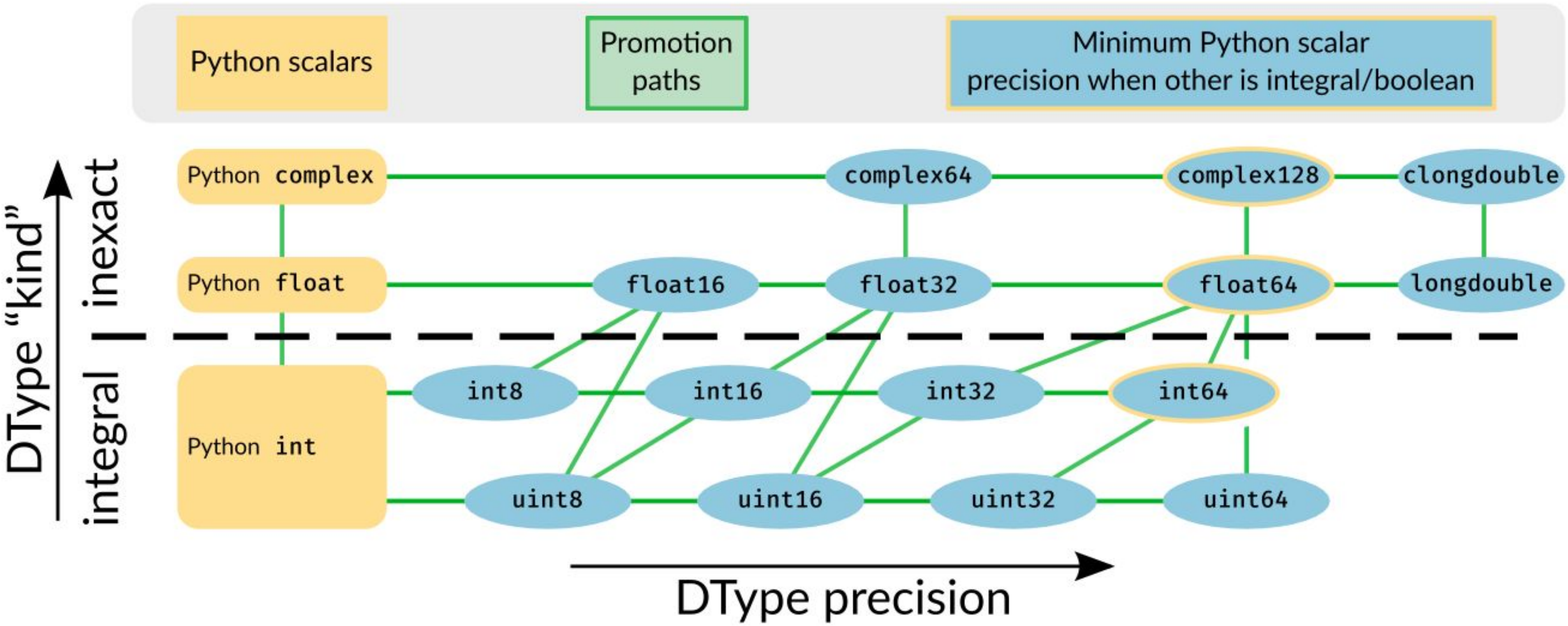
Current NEP 50 Proposal – Weak Python scalars

Expression	Old result	New result
`uint8(1) + 2`	`int64(3)`	`uint8(3)`
`array([1], uint8) + int64(1)` or `array([1], uint8) + array(1, int64)`	`array([2], unit8)`	`array([2], int64)`
`array([1.], float32) + float64(1.)` or `array([1.], float32) + array(1.,float64)`	`array([2.], float32)`	`array([2.], float64)`
`array([1], uint8) + 1`	`array([2], uint8)`	*unchanged*
`array([1], uint8) + 200`	`array([201], np.uint8)`	*unchanged*
`array([100], uint8) + 200`	`array([ 44], uint8)`	*unchanged*
`array([1], uint8) + 300`	`array([301], uint16)`	*Exception*

Current NEP 50 Proposal – Weak Python scalars

Expression	Old result	New result
`uint8(1) + 300`	`int64(301)`	*Exception*
`uint8(100) + 200`	`int64(301)`	`uint8(44)` *and* `RuntimeWarning`
`float32(1) + 3e100`	`float64(3e100)`	`float32(Inf)` *and* `RuntimeWarning`
`array([0.1], float32) == 0.1`	`array([False])`	*unchanged*
`array([0.1], float32) == float64(0.1)`	`array([ True])`	`array([False])`
`array([1.], float32) + 3`	`array([4.], float32)`	*unchanged*
`array([1.], float32) + int64(3)`	`array([4.], float32)`	`array([4.], float64)`

Current NEP 50 Proposal – Weak Python scalars



- This is not simple, please do have a look at the NEP: https://numpy.org/neps/nep-0050-scalar-promotion.html
- There is a table highlighting some changes!

Most of this is implemented and usable

- I have done most of the impossible things!

```
export NPY_PROMOTION_STATE=weak
np._set_promotion_state(“weak”)
```

- No, not thread safe (and I have no intention)

- Yes, there are still some bugs (but most likely you won’t notice)

Design space and open questions 1

```
(float32(3) + 4.0).dtype == float32
```

```
np.add(float32(3), 4.0).dtype == ?
```

```
py_function(float32(3), 4.0).dtype == ?
```

```
(float32(3) + np.asarray(4.0)).dtype == float64
```

- JAX actually uses the weak logic for the last one too!
- There will *always* be inconsistencies
- Do we need `asarray_or_literal()`?!
  - Maybe `result_type` is enough (other alternative?)

Design space and open questions 2

1. Scalars *could* behave different from arrays
   - (generally, I suspect this would be too confusing)

2. “Cast safety” is a hard and unsolved concept!
   - `np.add(uint8(3), 3, casting=”equiv”) ?`
   - `np.add(uint8(3), 300, casting=”unsafe”) ?`

3. Working with large Python integer 2**1000 will be hard;
   - `np.int64(3) + 10**300  # Error!`

Things to figure out! (hopefully not more)

- Cast safety for scalars!
    - There is probably no fully consistent solution.

- Test failures in NumPy:
    - Have been chipping away at it.
    - `np.arange(uint8(100))` of all is tricky (hopefully the solution is not)

- If and how to deal with Python level pattern (also in downstream):
    - `def function(a, b): a, b = np.asarray(a), np.asarray(b)`

- If we need to do things (and if what) to make transition easier.

User Impact and Alternatives

Changes affect mainly **end-users** and **not** libraries! Mostly, I expect changes to not matter, but they will break scripts in hard to track down ways!

- Affects *many* users but also many will not be affected:
  - Array math effectively "weak" scalars!
  - Array math results will be more exact precision usually
  - Floating point comparisons may change results!
  - Code using default DTypes (integer/floats) is not be affected.

- Scalar math is likely the main problem:
  - *Most* paths will at least give a warning (float math) or error (integer math)
  - ```
    arr = np.arange(100, dtype=np.uint8)  # storage array with low precision
    value = arr[10]
    # use "value" without considering where it came from
    value * 100  # Overflow warning (luckily)
    ```

- Large integer math will become harder.
    ```

Alternatives

- JAX style `np.asarray(3.0)` is still “weak”
    - I doubt this is feasible but…
- Strong scalars:
    - Python float is float64, Python int is “default integer”.
    - Only 100% consistent solution, but gives up on all the nice things
- Make scalars special (but how?)
- Give up :)