Integrating Dynamic Region-Based Ownership Model into CPython and NumPy

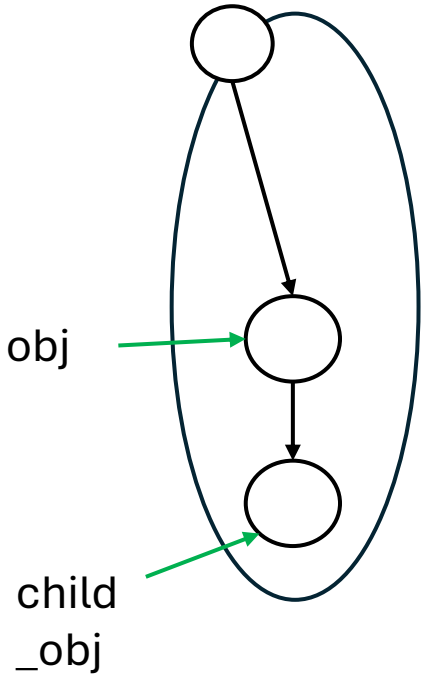
Degree Projects in Computer Science at Masters Level

Sivakorn (Mil) Lerttripinyo

Research Questions

- Analyzing recurring code patterns
- Finding any deviation from recurring code patterns
- Verifying the conformance to the model
- Finding challenges from applying the model to NumPy

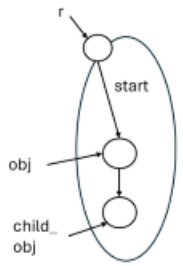
Local Reference Count (LRC)



LRC
+1+1

How to enforce these constraints?

Region and Moving Objects into Region



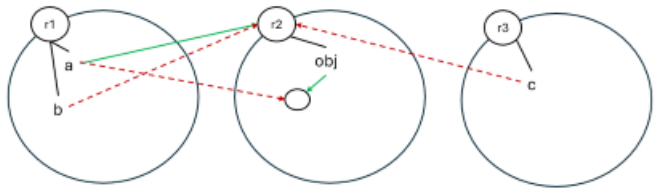

```
r = Region()
child_obj = {"child_key": "child_value"}
obj = {"key": "value", "child": child_obj}
r.start = obj
```

Keyword:
- **Ephemeral Property**

9

External Uniqueness

```
r1 = Region()
r2 = Region()
r3 = Region()
r2.obj = {"key": "value"}
r1.a = r2
# r1.b = r2
# r3.c = r2
```

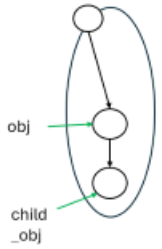


: This reference is allowed.

: This reference is forbidden.

12

Local Reference Count (LRC)

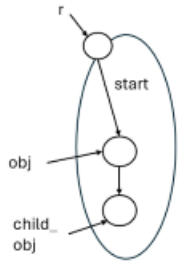


LRC
+1+1

10

Write-Barriers enforce these constraints.

Region and Moving Objects into Region



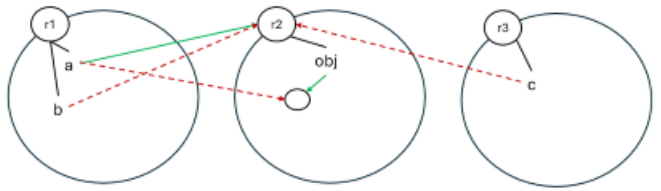

```
r = Region()
child_obj = {"child_key": "child_value"}
obj = {"key": "value", "child": child_obj}
r.start = obj
```

Keyword:
- **Ephemeral Property**

9

External Uniqueness

```
r1 = Region()
r2 = Region()
r3 = Region()
r2.obj = {"key": "value"}
r1.a = r2
# r1.b = r2
# r3.c = r2
```

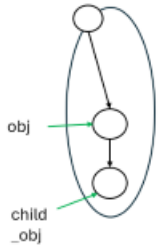


: This reference is allowed.

: This reference is forbidden.

12

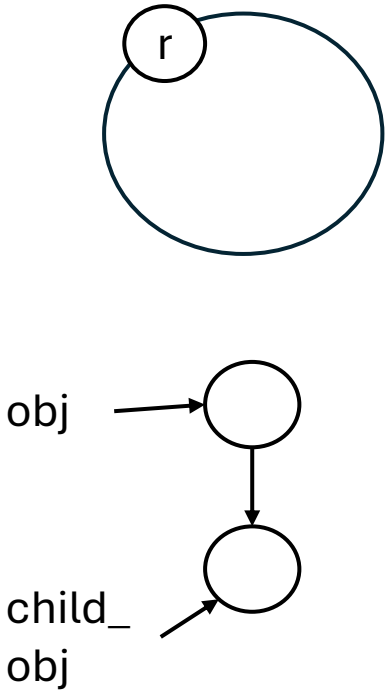
Local Reference Count (LRC)



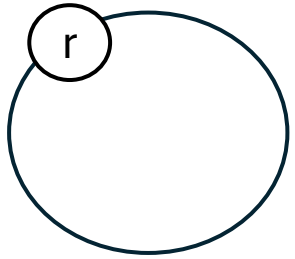
LRC
+1+1

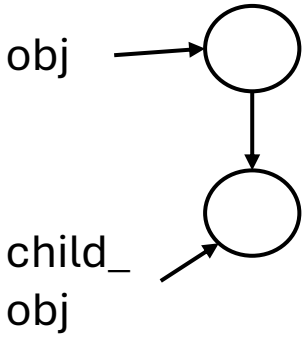
10

Write-Barriers enforce these constraints.



Write-Barriers enforce these constraints.

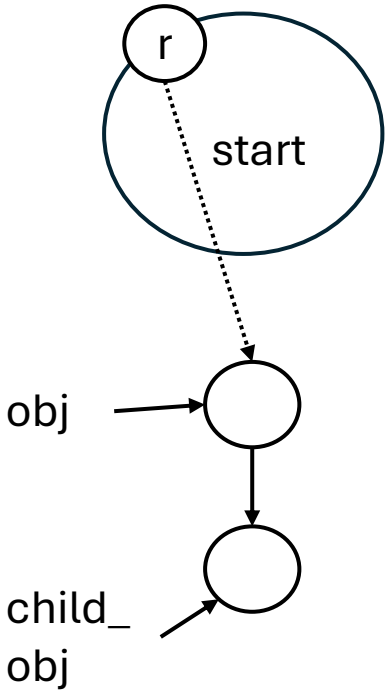




Python Code:

```
r.start = obj
```

Write-Barriers enforce these constraints.

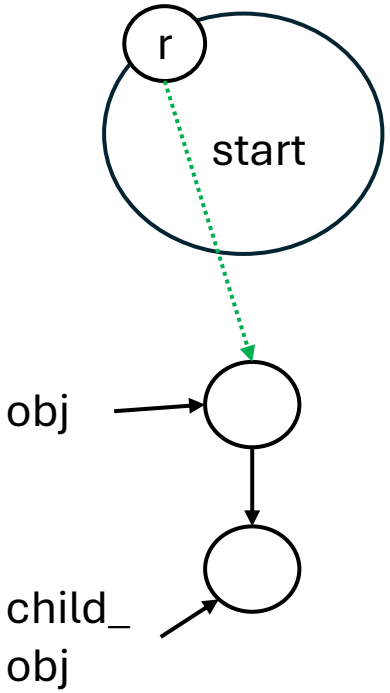


C code in Cpython:

```
writeBarrier_AddConstraint(r, obj)
```

Write-Barriers enforce these constraints.

Ephemeral Property

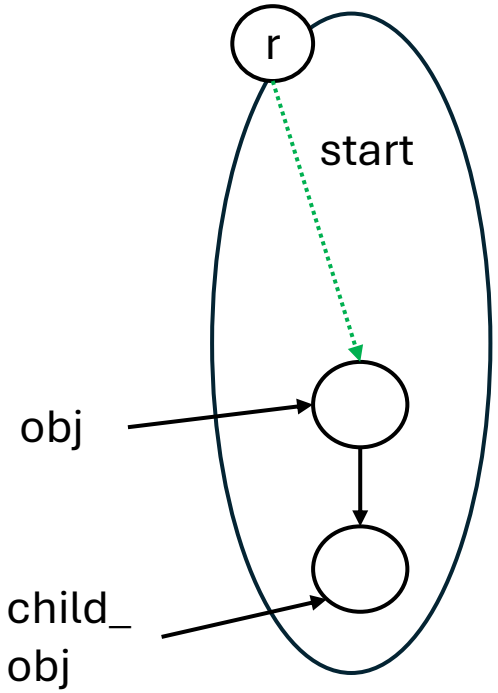


C code in Cpython:

```
writeBarrier_AddConstraint(r, obj)
```

Write-Barriers enforce these constraints.

Ephemeral Property

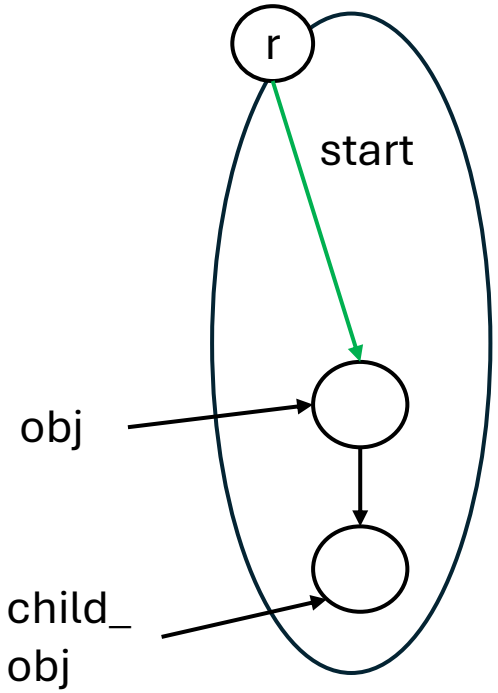


C code in Cpython:

```
writeBarrier_AddConstraint(r, obj)
```

Write-Barriers enforce these constraints.

Ephemeral Property

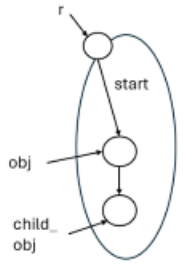


C code in Cpython:

```
writeBarrier_AddConstraint(r, obj)
r.start = Py_NewRef(obj)
```

Write-Barriers enforce these constraints.

Region and Moving Objects into Region



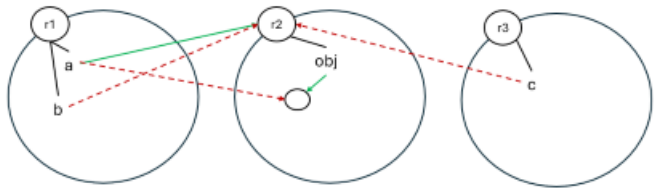

```
r = Region()
child_obj = {"child_key": "child_value"}
obj = {"key": "value", "child": child_obj}
r.start = obj
```

Keyword:
- **Ephemeral Property**

9

External Uniqueness

```
r1 = Region()
r2 = Region()
r3 = Region()
r2.obj = {"key": "value"}
r1.a = r2
# r1.b = r2
# r3.c = r2
```

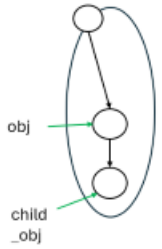


: This reference is allowed.

: This reference is forbidden.

12

Local Reference Count (LRC)



LRC
+1+1

10

Write-Barriers enforce these constraints.

$S \rightarrow T$	T				
S	local	immutable	bridge	contained	cown
local	✓	✓	✓[B]	✓[B]	✓
immutable	✗	✓	✗	✗	✓
bridge	→	✓	✓[E∨U]	✓[E]	✓
contained	→	✓	✓[E∨U]	✓[E]	✓
cown	✗	✓	✓[U]	✗	✓

✓ : Permitted

✗ : Forbidden

→ : Ownership Transfer

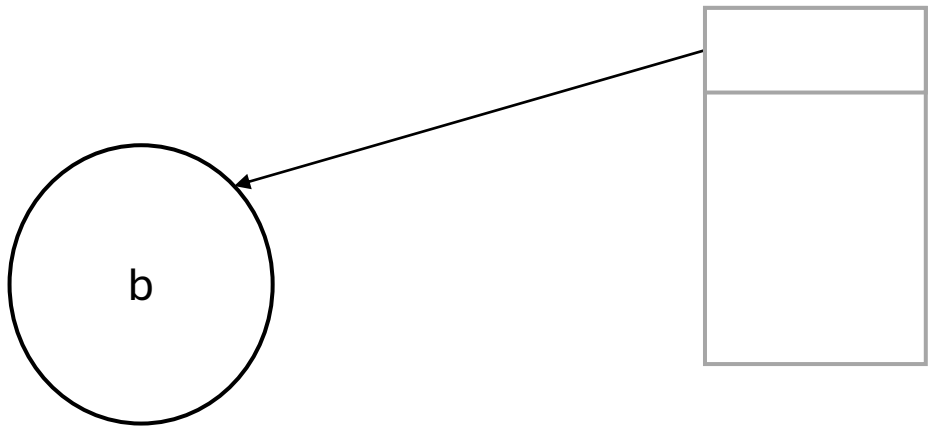
E : If S and T belong to the same region

B : The → reference is borrowed

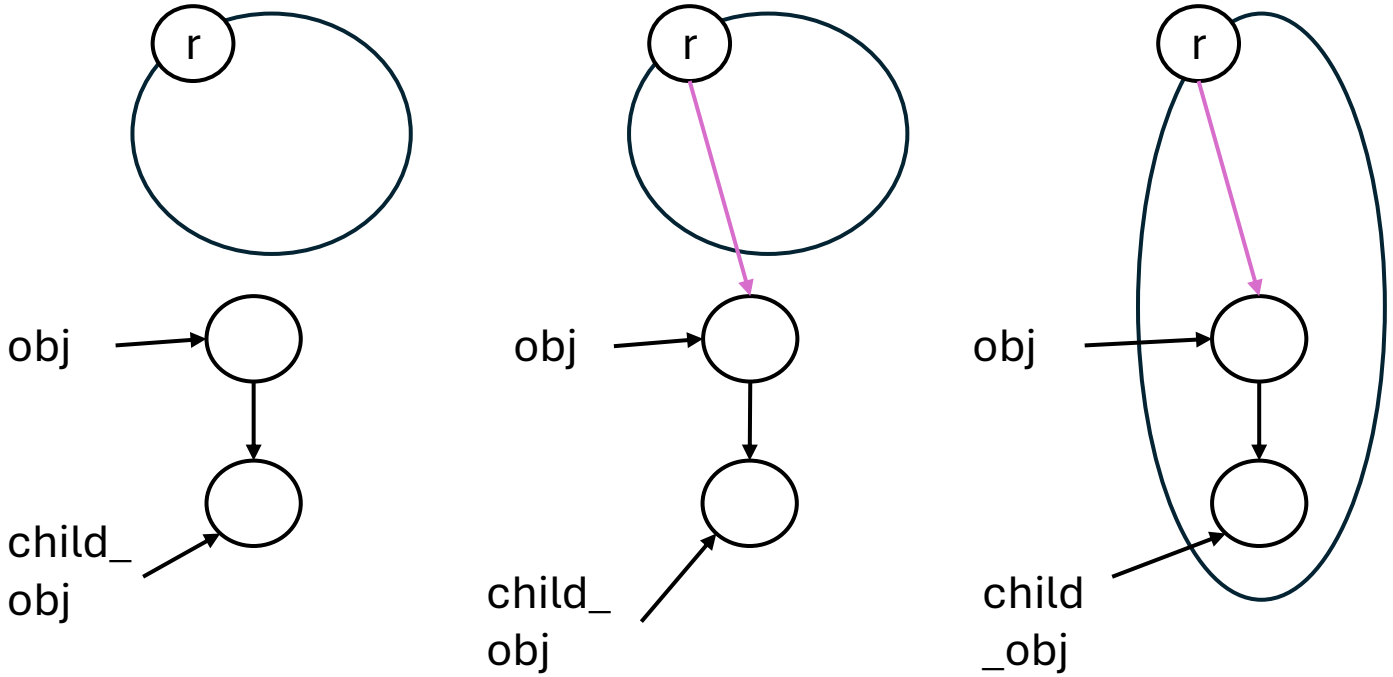
U : If S's reference to T is the only reference to T originating from outside of T

https://doi.org/10.1145/3729313

Reverting `AddLocalRef` with `RemoveLocalRef`

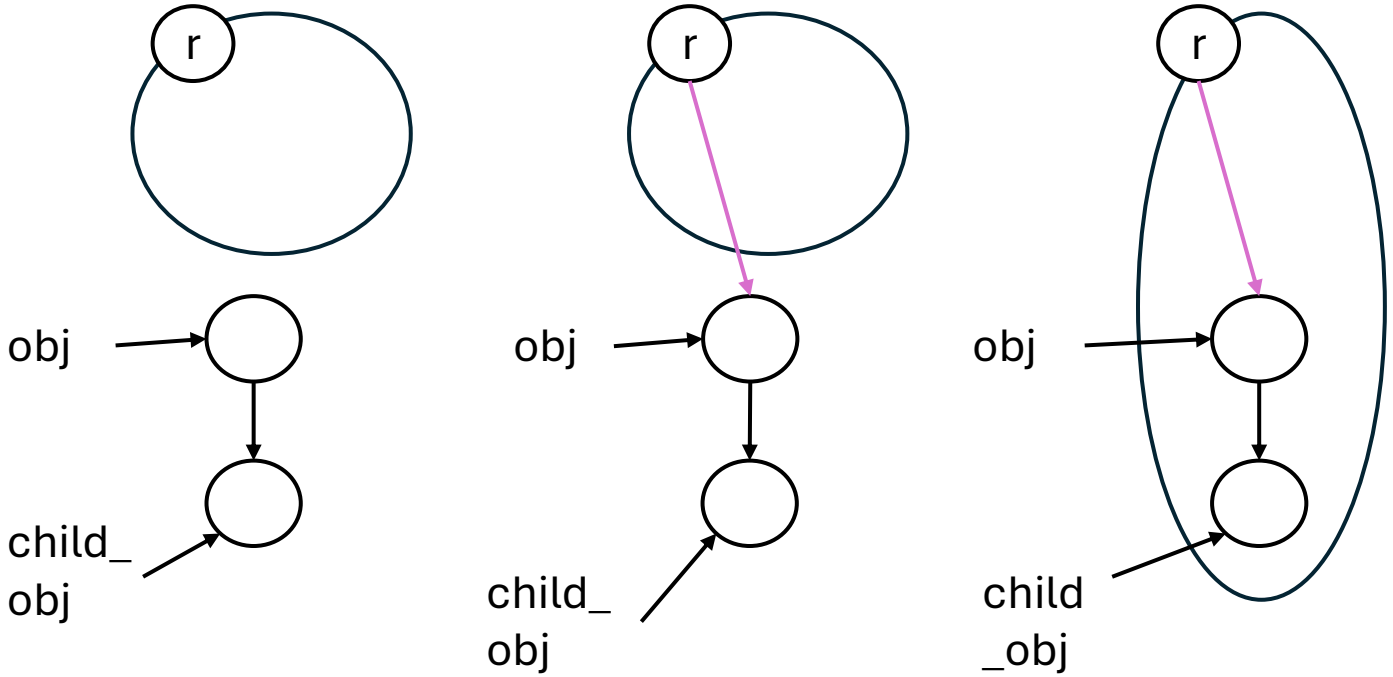


Reverting `AddRef` with `RemoveRef`



AddRef(r, obj)

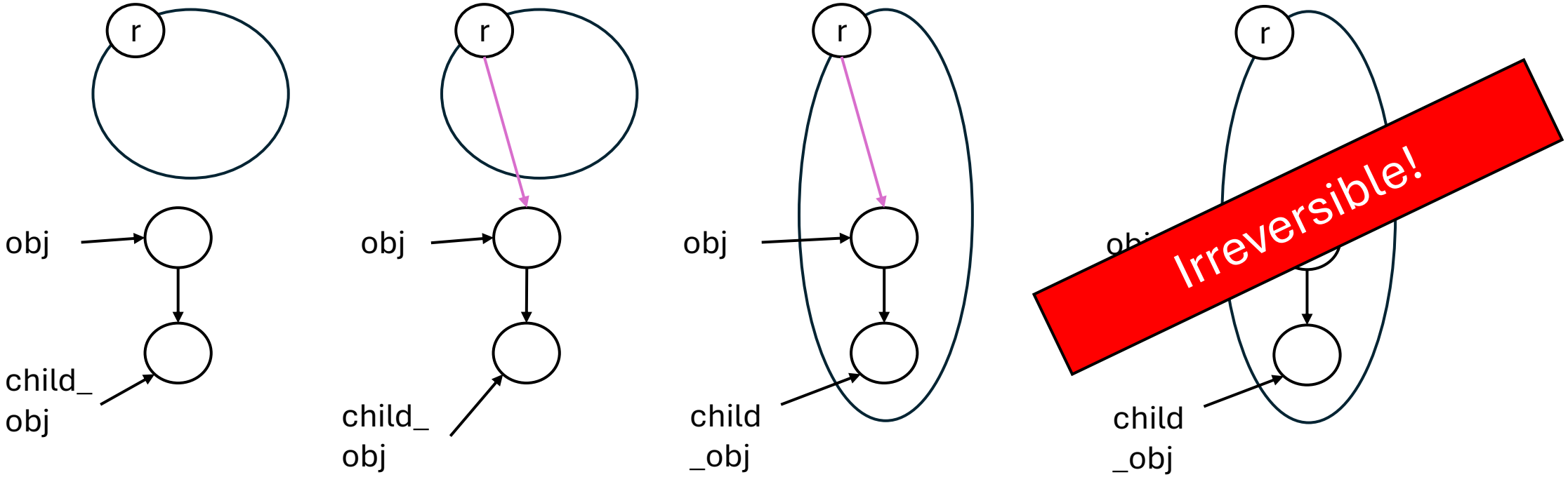
Reverting `AddRef` with `RemoveRef`



AddRef(r, obj)

RemoveRef(r, obj)

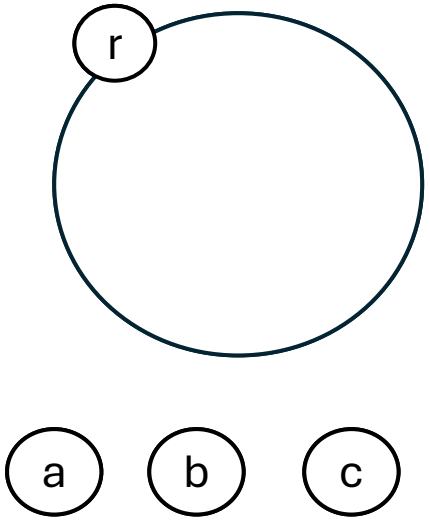
Reverting `AddRef` with `RemoveRef`



AddRef(r, obj)

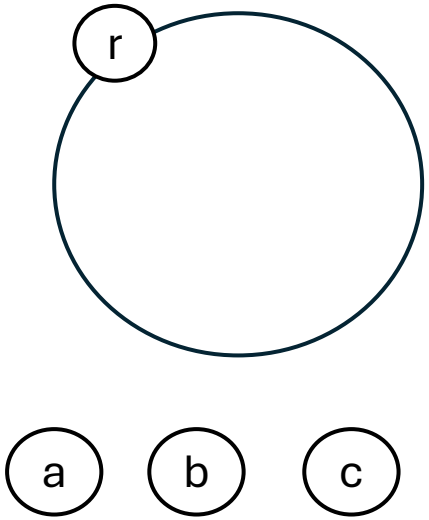
RemoveRef(r, obj)

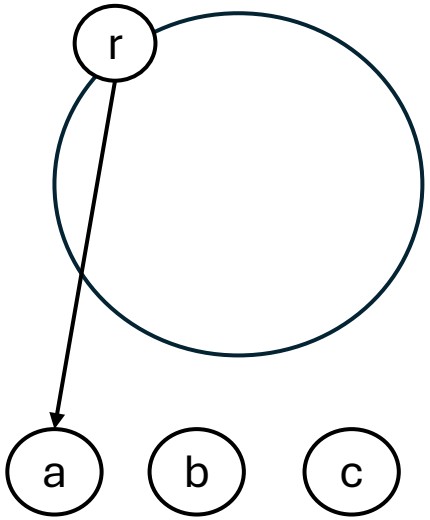
Add Several Objects



PyRegion_AddRef(r, a)
PyRegion_AddRef(r, b)
PyRegion_AddRef(r, c)

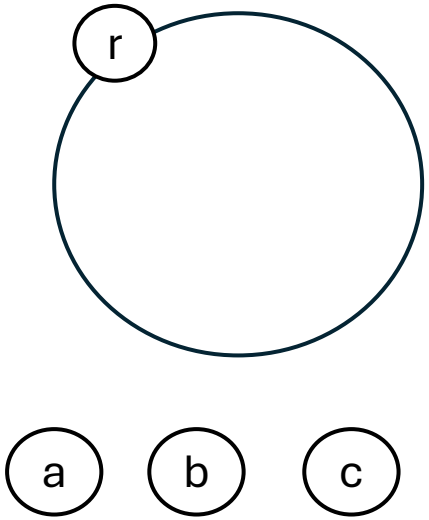
Add Several Objects

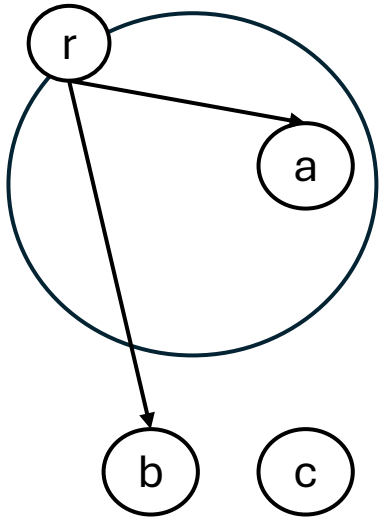




PyRegion_AddRef(r, a)
PyRegion_AddRef(r, b)
PyRegion_AddRef(r, c)

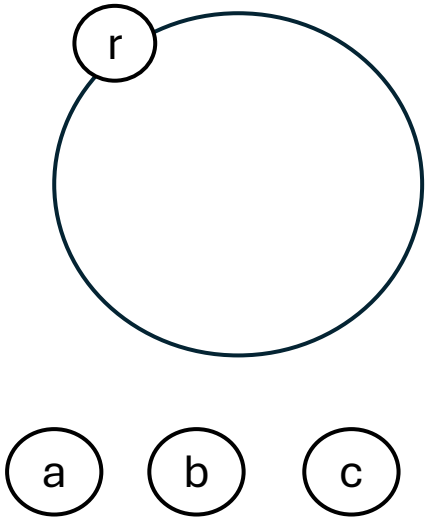
Add Several Objects

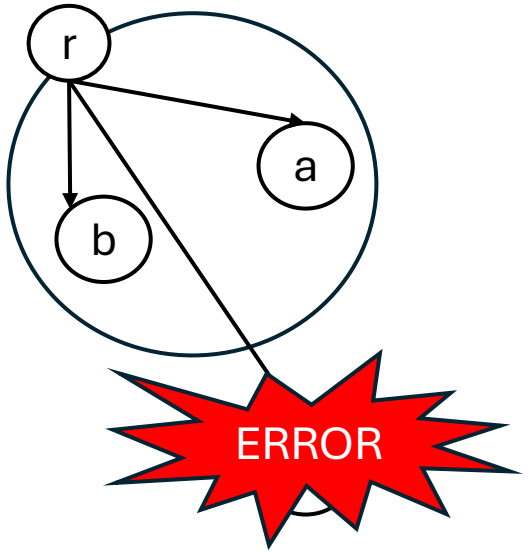




PyRegion_AddRef(r, a)
PyRegion_AddRef(r, b)
PyRegion_AddRef(r, c)

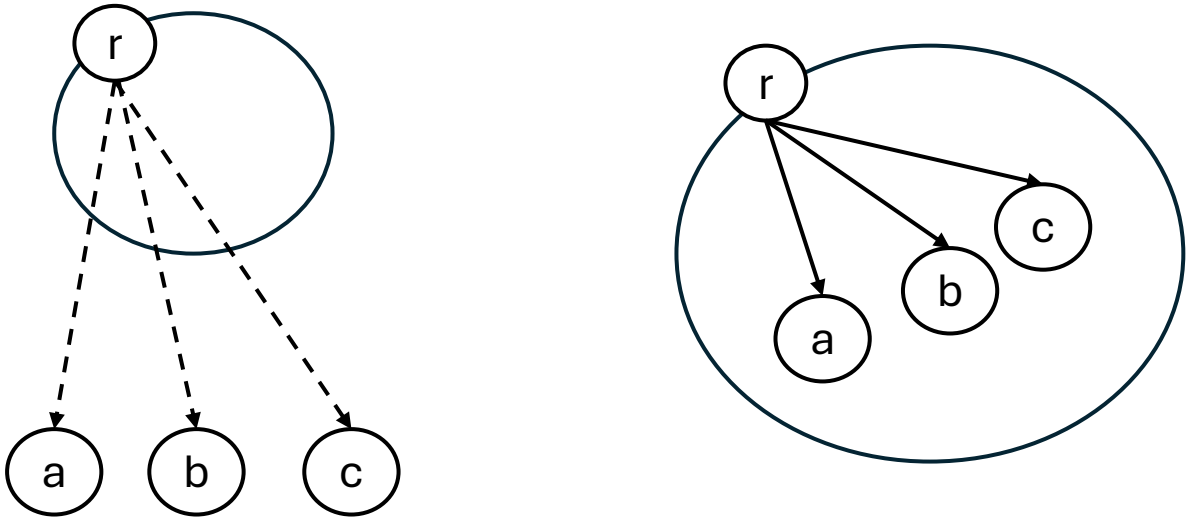
Add Several Objects





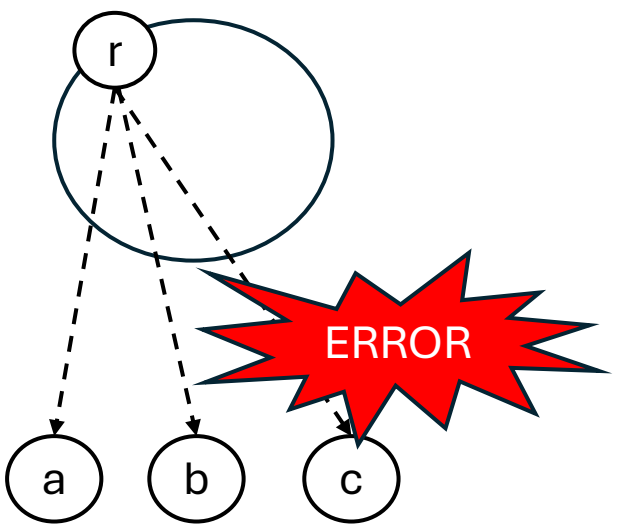
PyRegion_AddRef(r, a)
PyRegion_AddRef(r, b)
PyRegion_AddRef(r, c)

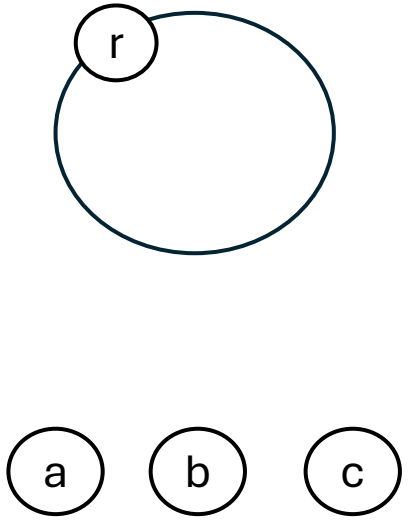
Add Several Objects



PyRegion_AddRefs(r, a, b, c)

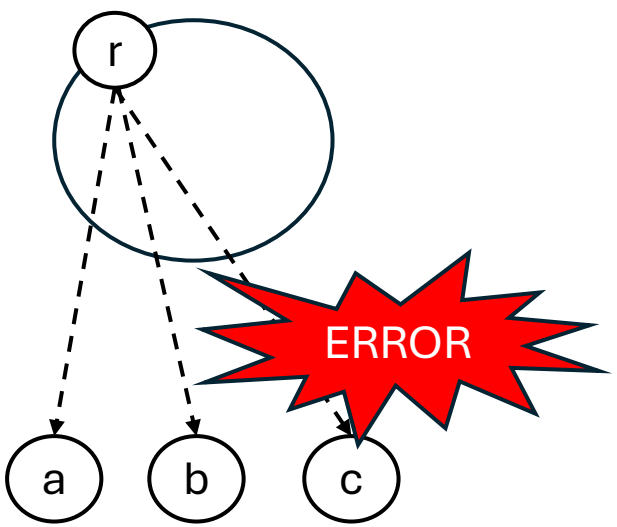
Add Several Objects

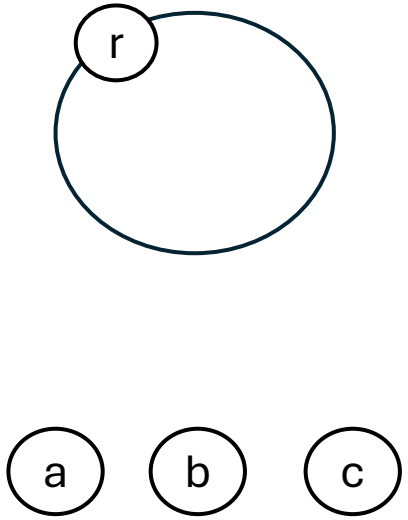




PyRegion_AddRefs(r, a, b, c)

Add Several Objects Atomically

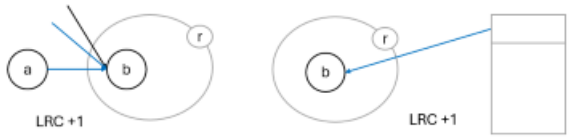




PyRegion_AddRefs(r, a, b, c)
PyRegion_TakeRefs(r, a, b, c)

General Patterns (More detail in Report)

AddRef or AddLocalRef



if(PyRegion_AddRef(a, b) {...}

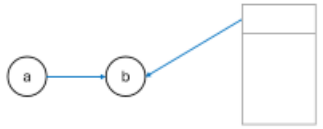
a = Py_NewRef(b)

if(PyRegion_AddLocalRef(b) {...}

Py_INCREF(b)

36

TakeRef



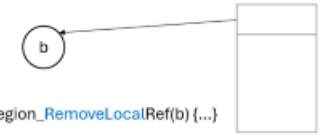

```
int func1 (…) {
    if(PyRegion_AddLocalRef(b) {…}
    Py_INCREF(b)
    func2(…, b)
}

int func2 (…) {
     if(PyRegion_TakeRef(a,b) {…}
     a = b
}
```

39

RemoveRef or RemoveLocalRef

if(PyRegion_RemoveRef(a, b) {...}

Py_DECREF(b)



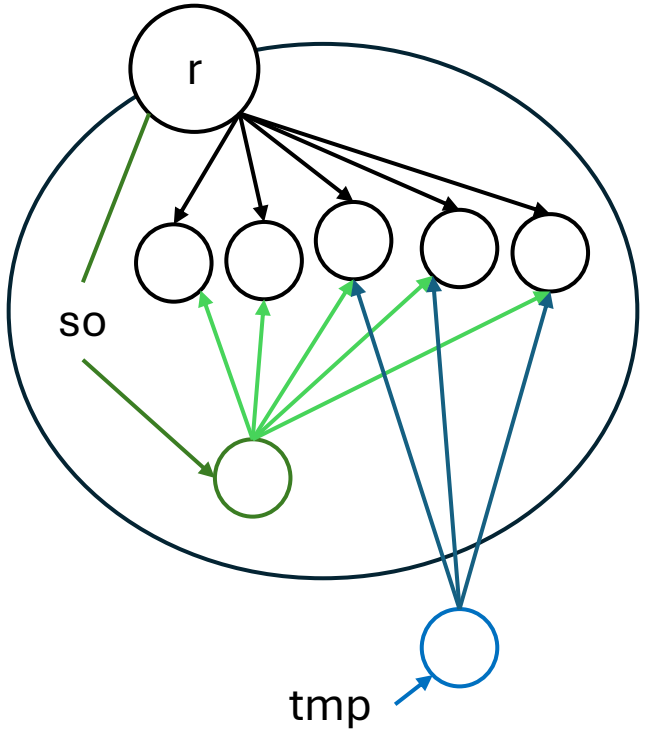
if(PyRegion_RemoveLocalRef(b) {...}

Py_DECREF(b)

41

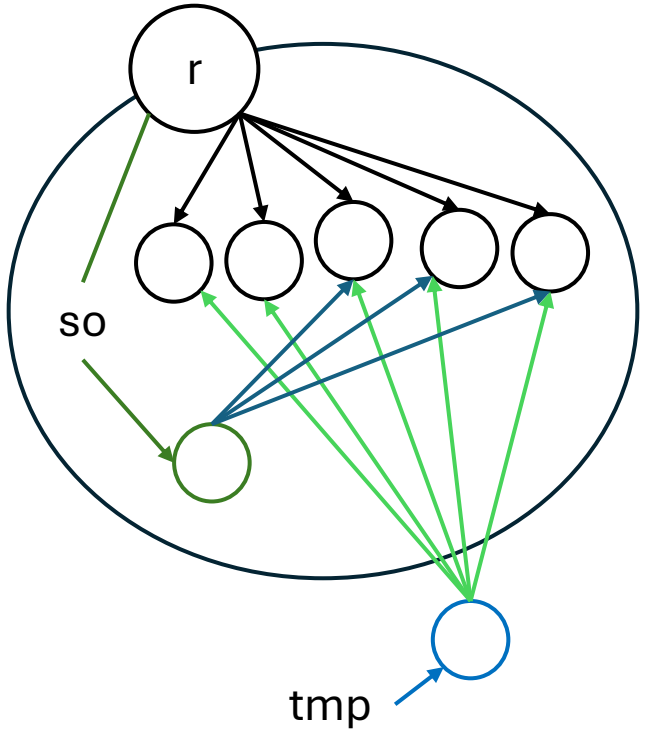
Non-General Patterns

set_swap_bodies(so, tmp) without Migration



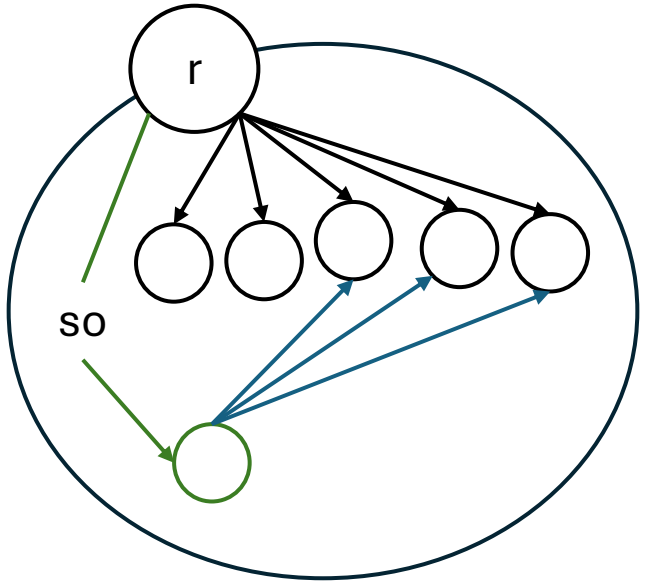
LRC +3

set_swap_bodies(so, tmp) without Migration



LRC +3

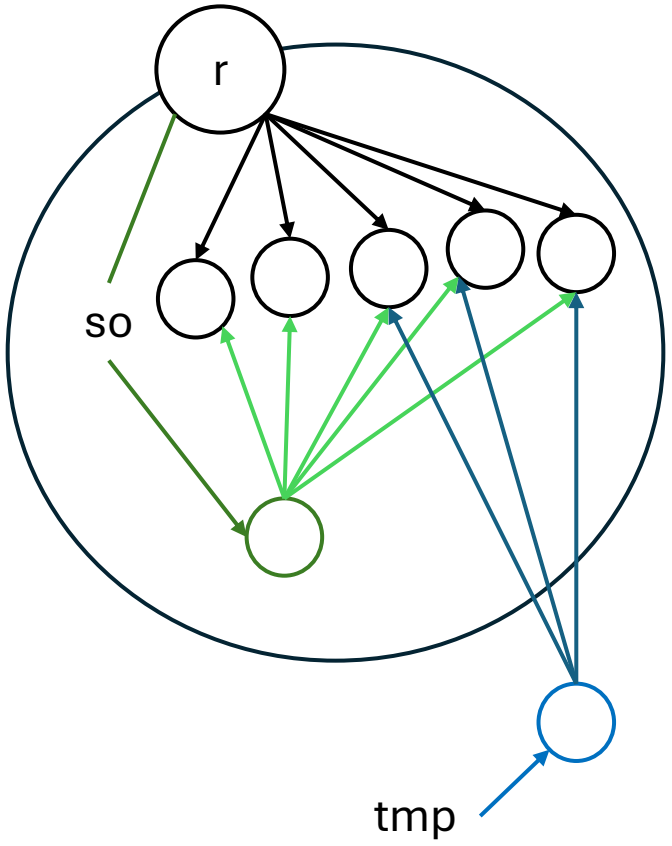
set_swap_bodies(so, tmp) without Migration



LRC +3 -5

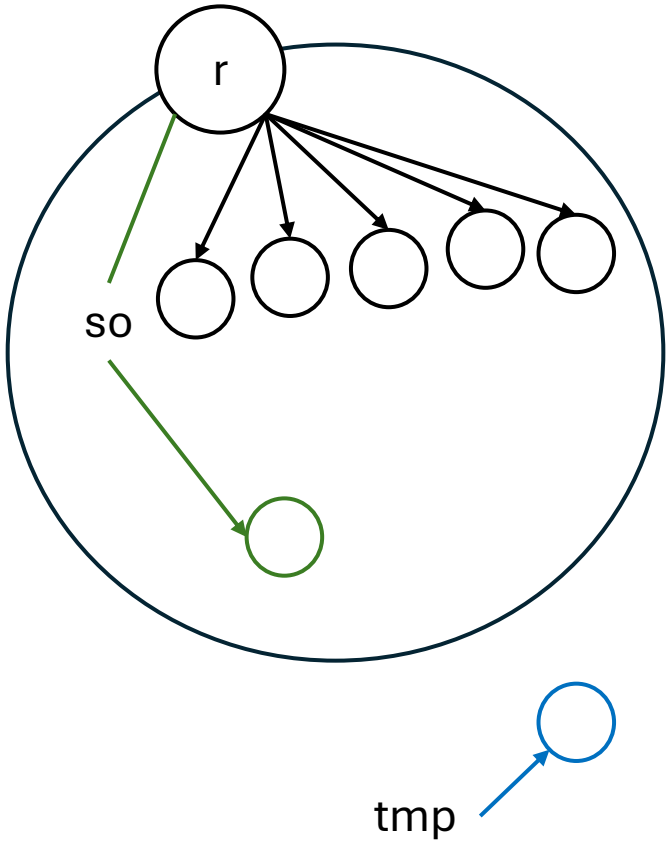
Does not make sense!

set_swap_bodies(so, tmp) with Migration - Direct Method



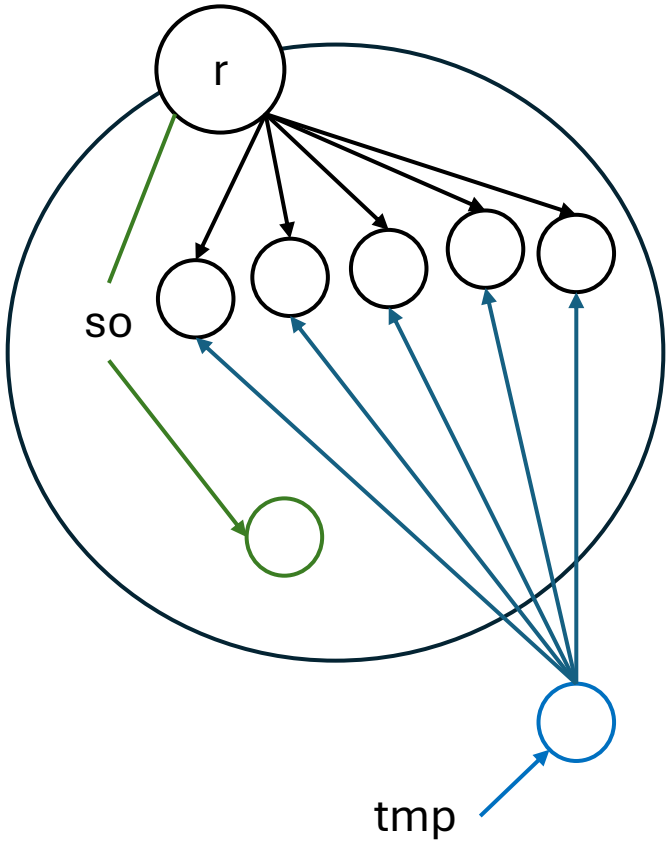
LRC +3

set_swap_bodies(so, tmp) with Migration - Direct Method



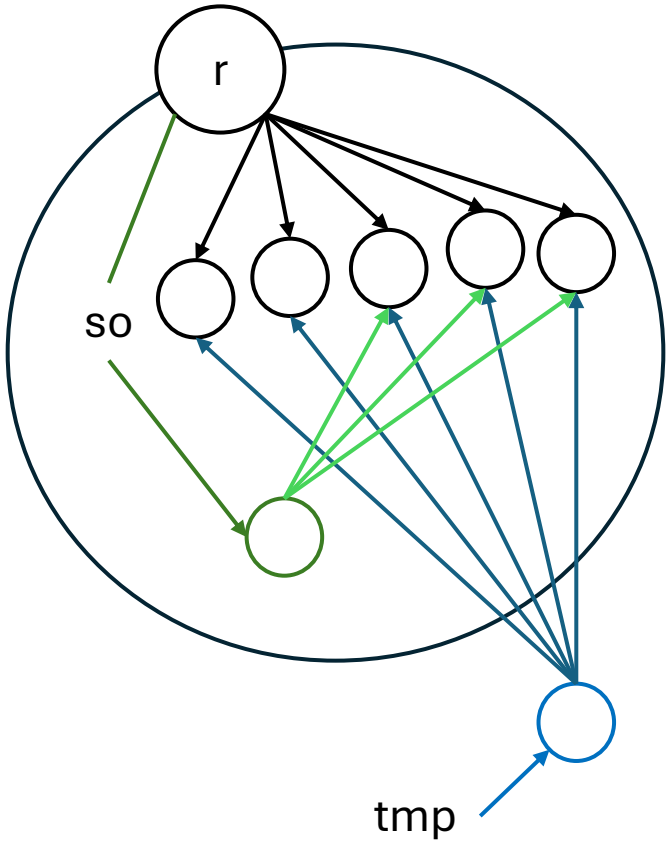
LRC +3 -3

set_swap_bodies(so, tmp) with Migration - Direct Method



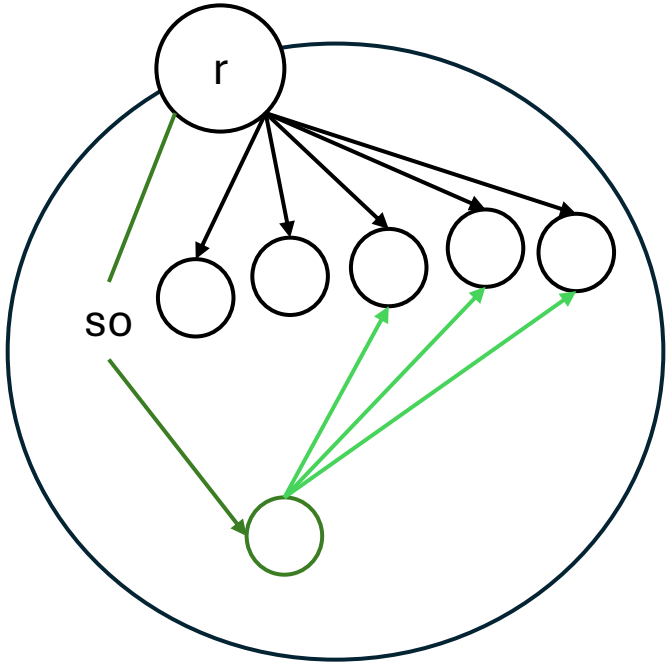
LRC +3 -3 +5

set_swap_bodies(so, tmp) with Migration - Direct Method



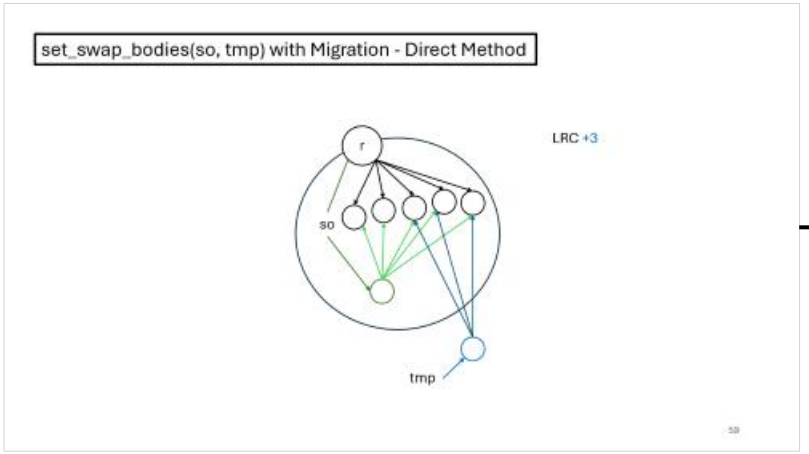
LRC +3 -3 +5

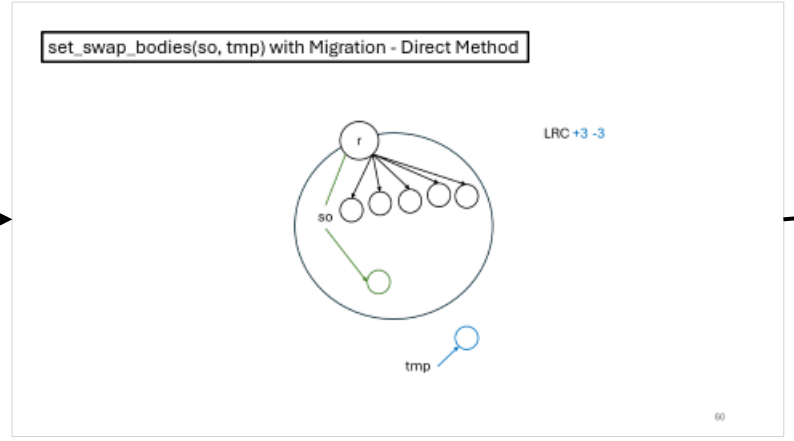
set_swap_bodies(so, tmp) with Migration - Direct Method

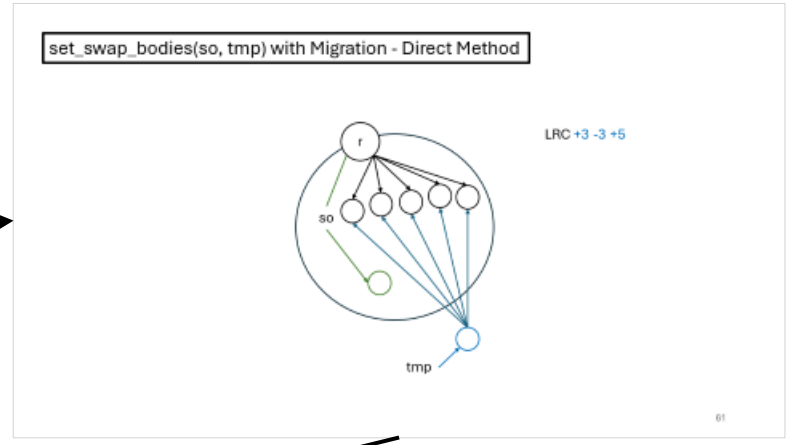


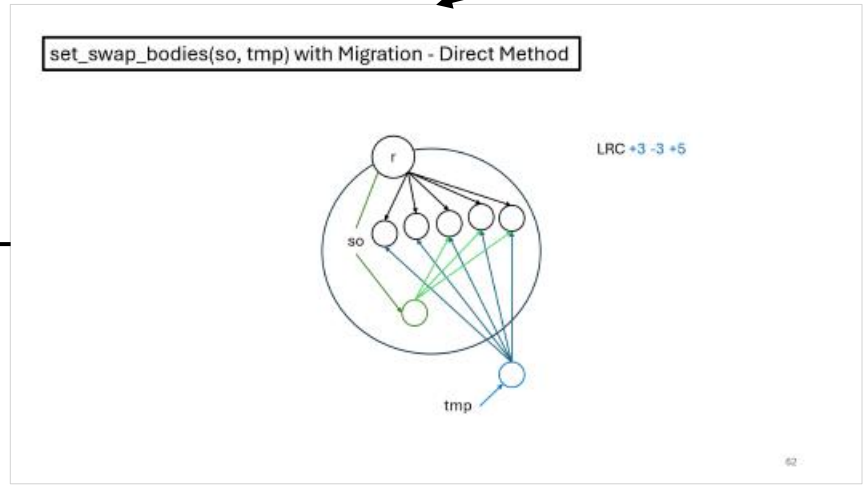
LRC +3 -3 +5 -5

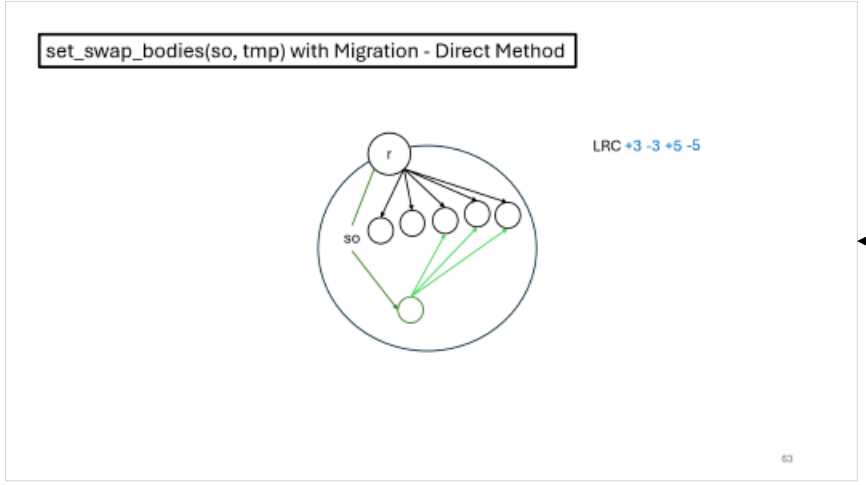
set_swap_bodies(so, tmp) with Migration - Direct Method



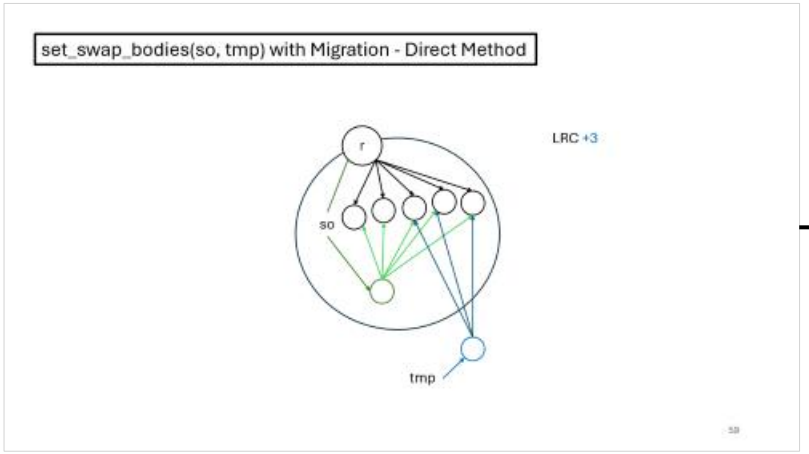


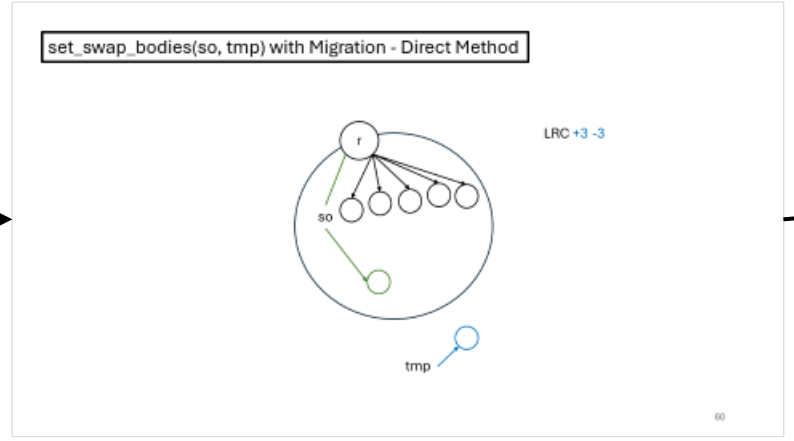


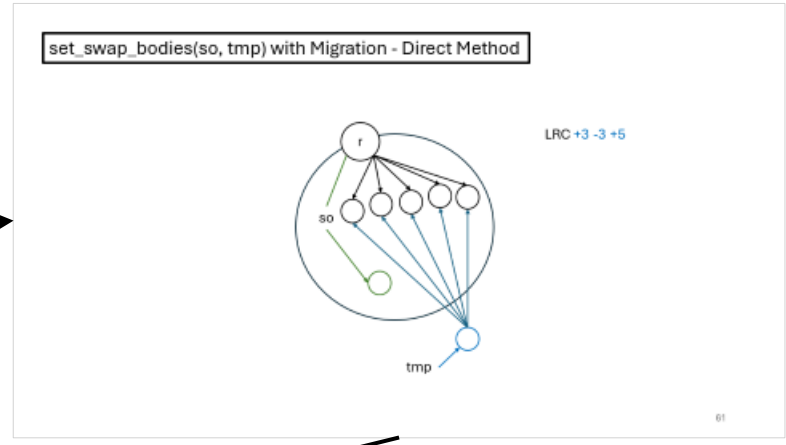


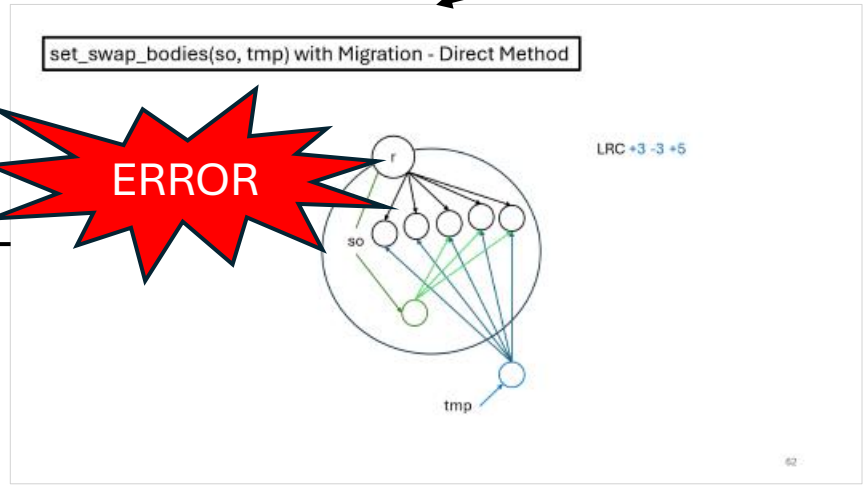


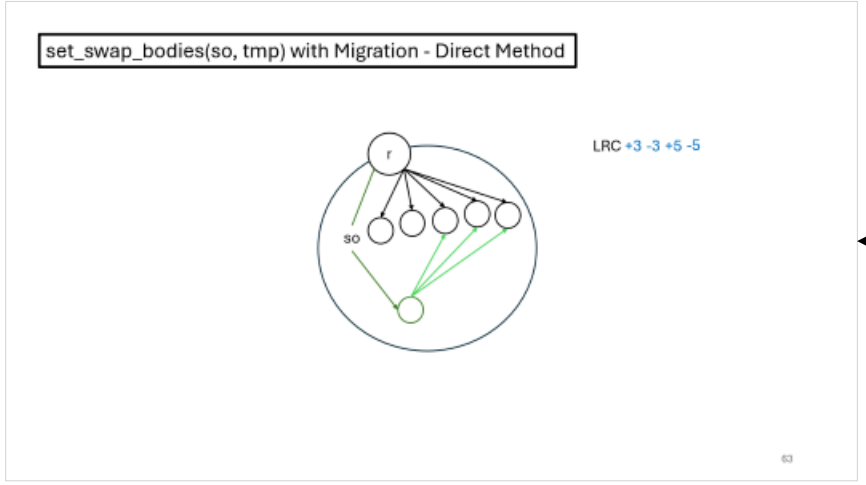
set_swap_bodies(so, tmp) with Migration - Direct Method



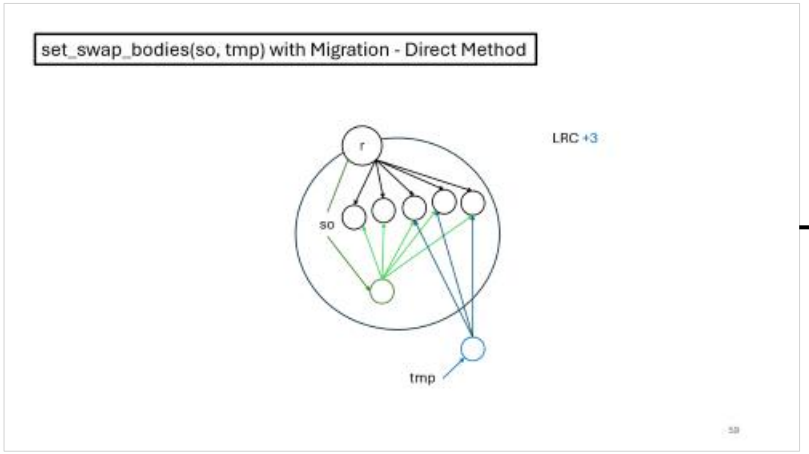


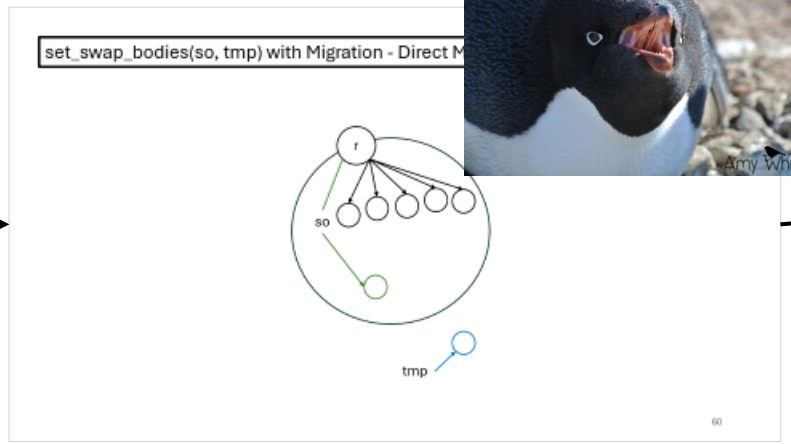


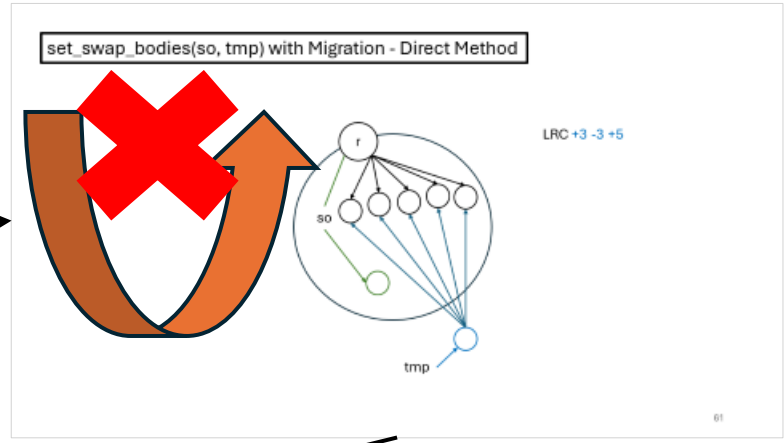


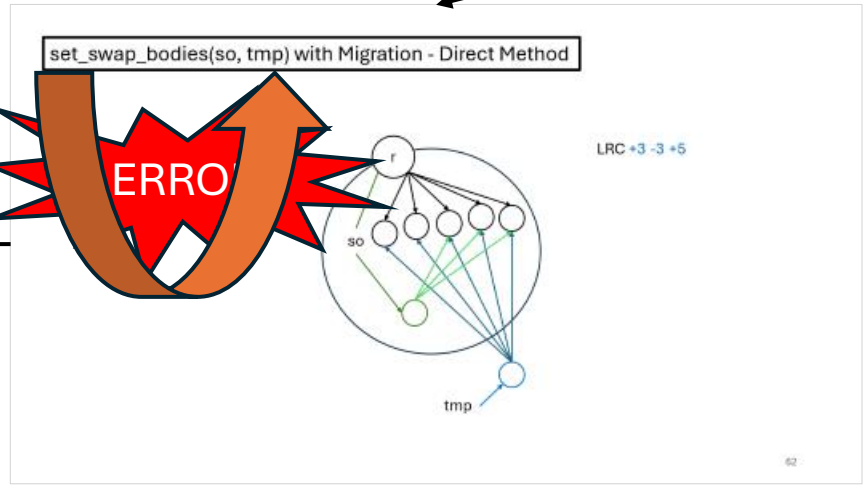


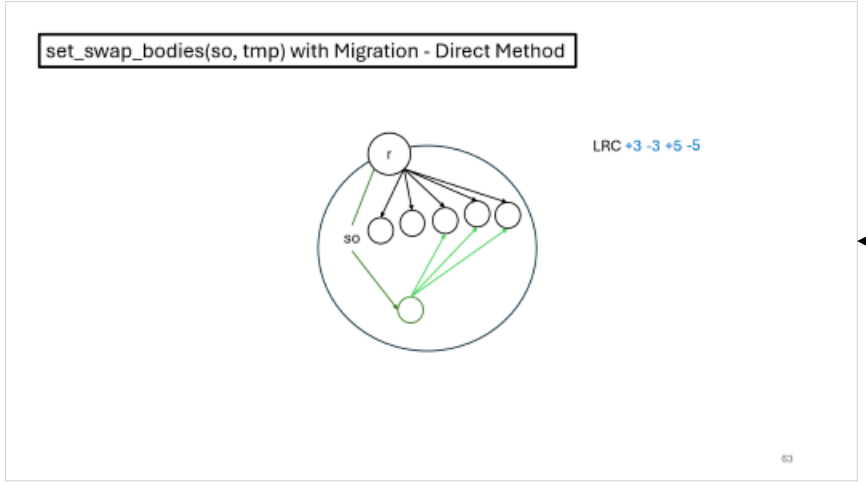
set_swap_bodies(so, tmp) with Migration - Direct Method









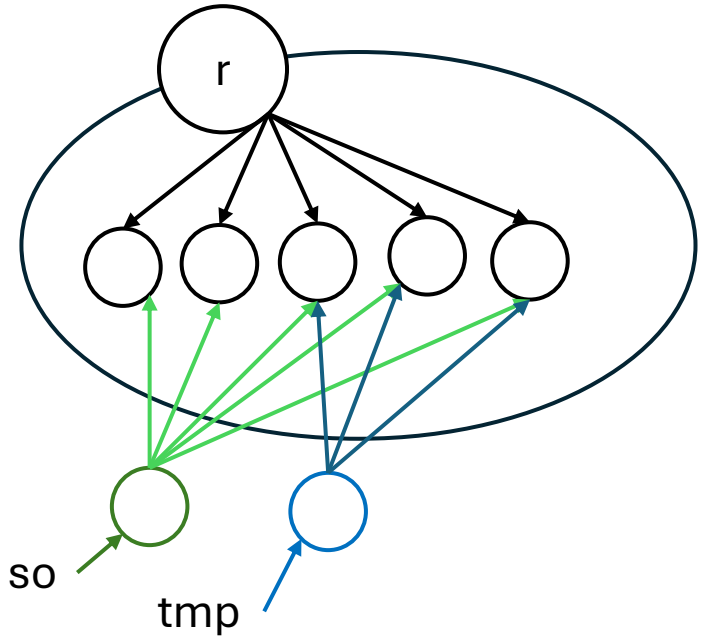


set_swap_bodies(so, tmp) with Migration – Using Assumption

Observe how this method is actually used.

set_swap_bodies(so, tmp) with Migration – Using Assumption

First Assumption: If both “so” and “tmp” are in local, no barrier is required.

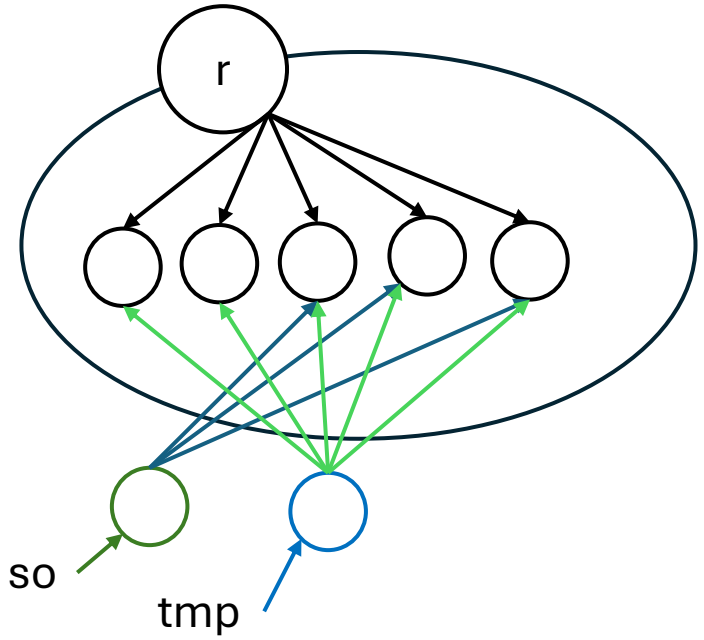


(a) Before executing
set_swap_bodies(...)

LRC +5+3

set_swap_bodies(so, tmp) with Migration – Using Assumption

First Assumption: If both “so” and “tmp” are in local, no barrier is required.

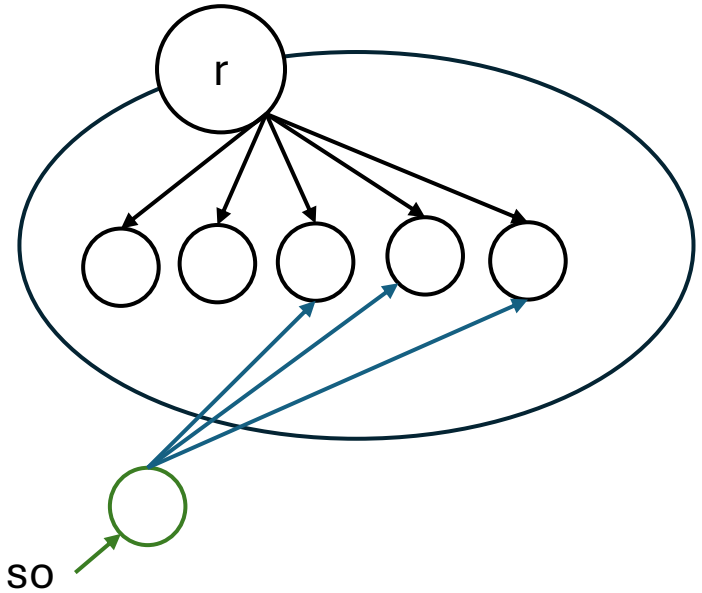


(b) After executing
set_swap_bodies(...)

LRC +5+3

set_swap_bodies(so, tmp) with Migration – Using Assumption

First Assumption: If both “so” and “tmp” are in local, no barrier is required.

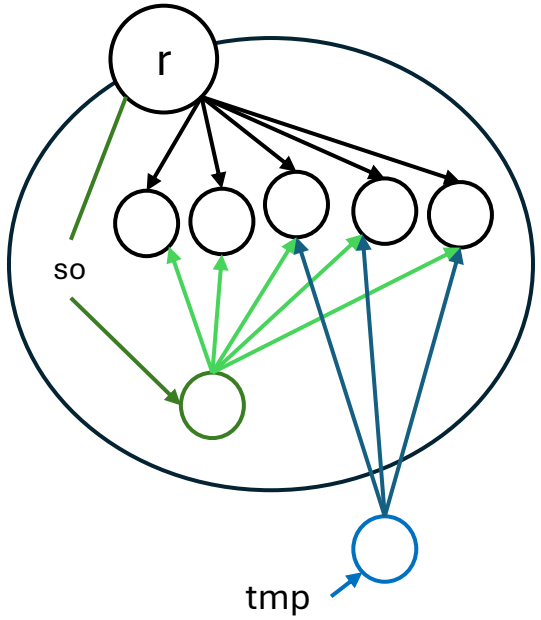


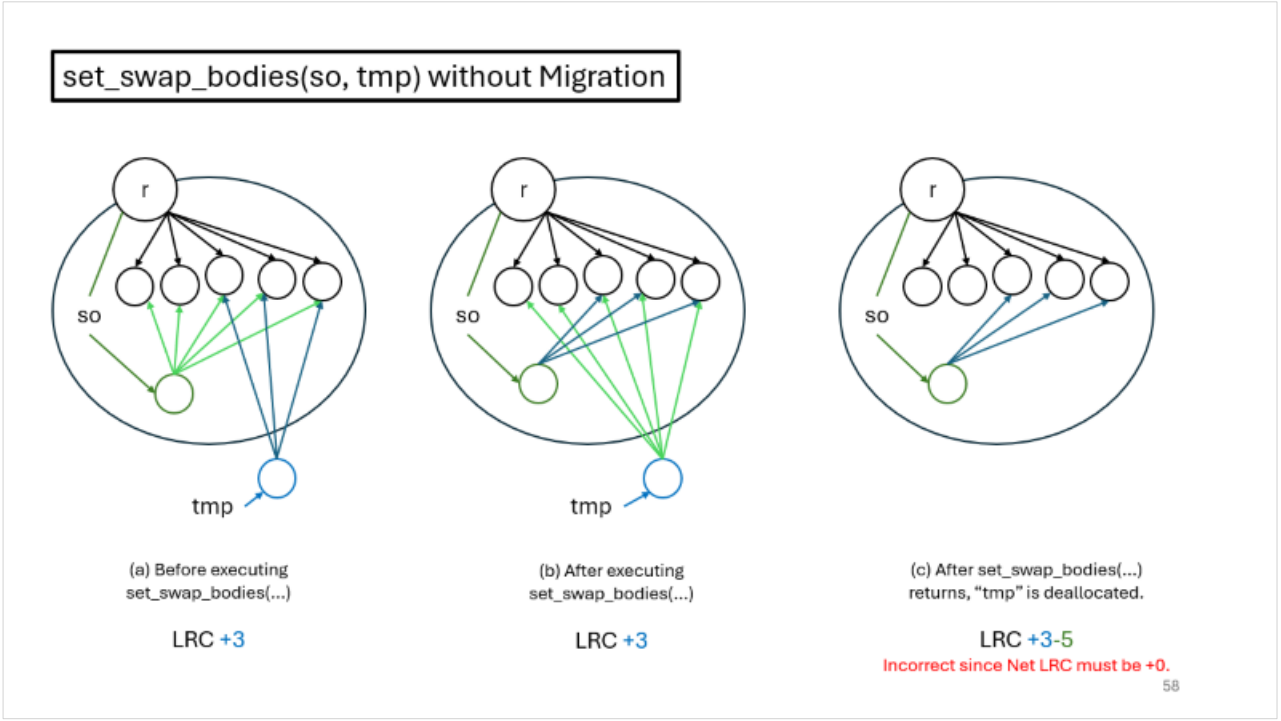
(c) After set_swap_bodies(...) returns, “tmp” is deallocated.

LRC +3

set_swap_bodies(so, tmp) with Migration – Using Assumption

Second Assumption: The variable “tmp” is always in local due to this method usage.

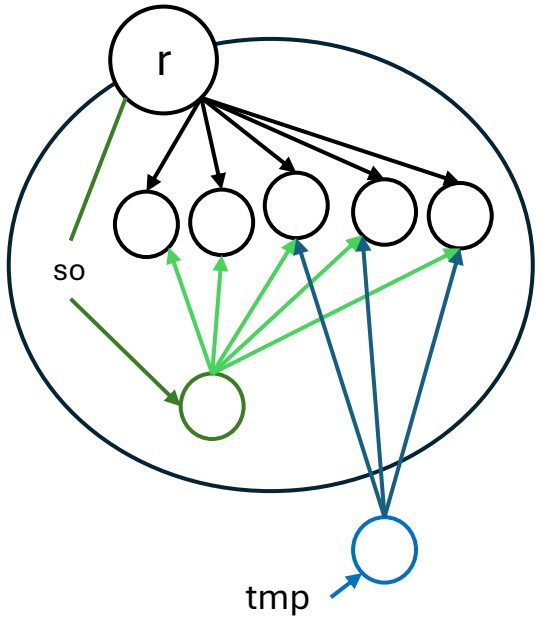




set_swap_bodies(so, tmp) with Migration – Using Assumption

Second Assumption: The variable “tmp” is always in local due to this method usage.

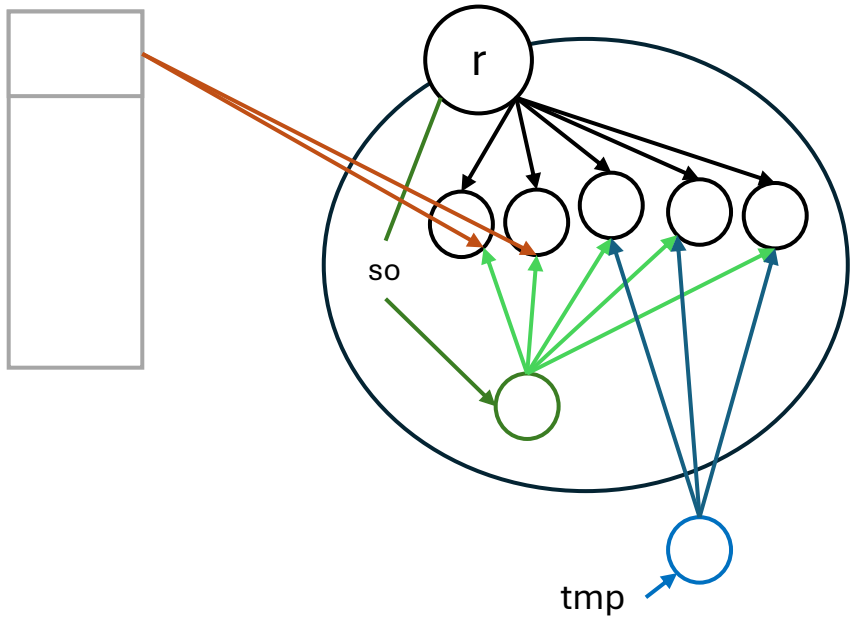
LRC +3



set_swap_bodies(so, tmp) with Migration – Using Assumption

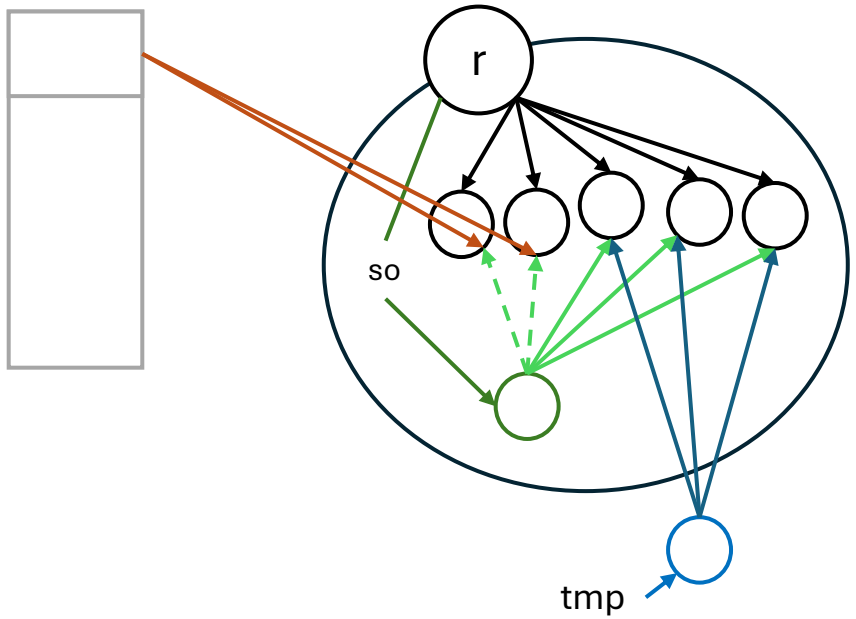
Second Assumption: The variable “tmp” is always in local due to this method usage.

LRC +3 +2



set_swap_bodies(so, tmp) with Migration – Using Assumption

Second Assumption: The variable “tmp” is always in local due to this method usage.

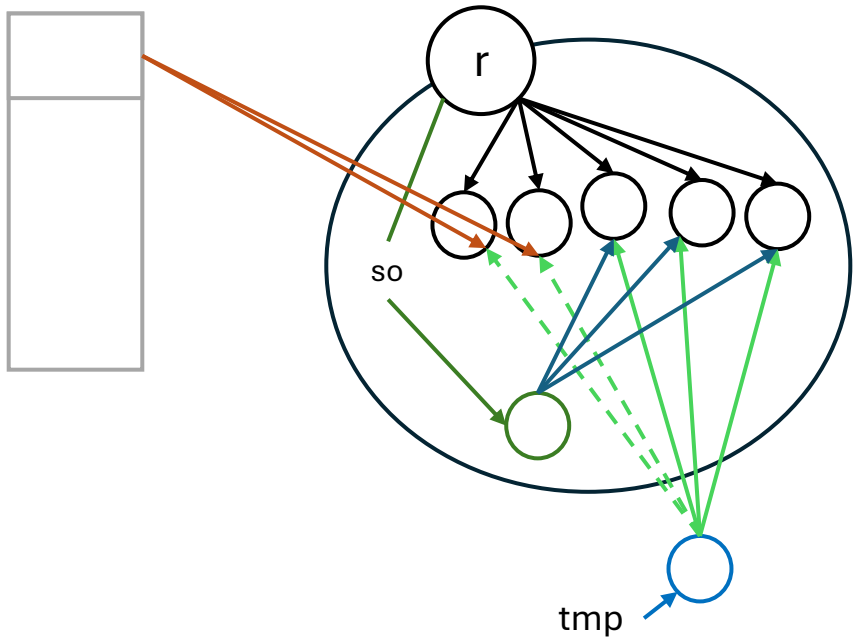


LRC +3 +2

: The constraint is removed,
but the set still holds this key.

set_swap_bodies(so, tmp) with Migration – Using Assumption

Second Assumption: The variable “tmp” is always in local due to this method usage.



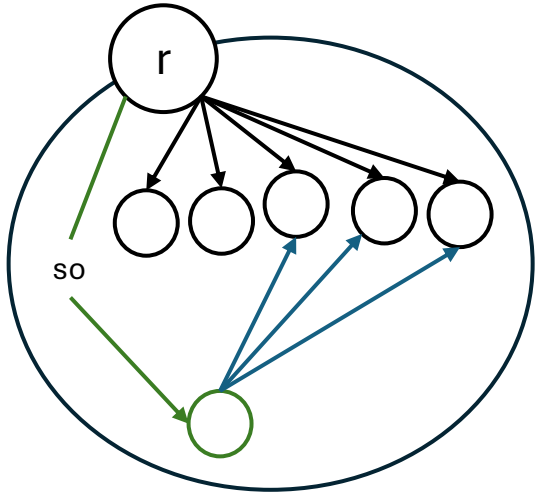
LRC +3 +2

: The constraint is removed,
but the set still holds this key.

set_swap_bodies(so, tmp) with Migration – Using Assumption

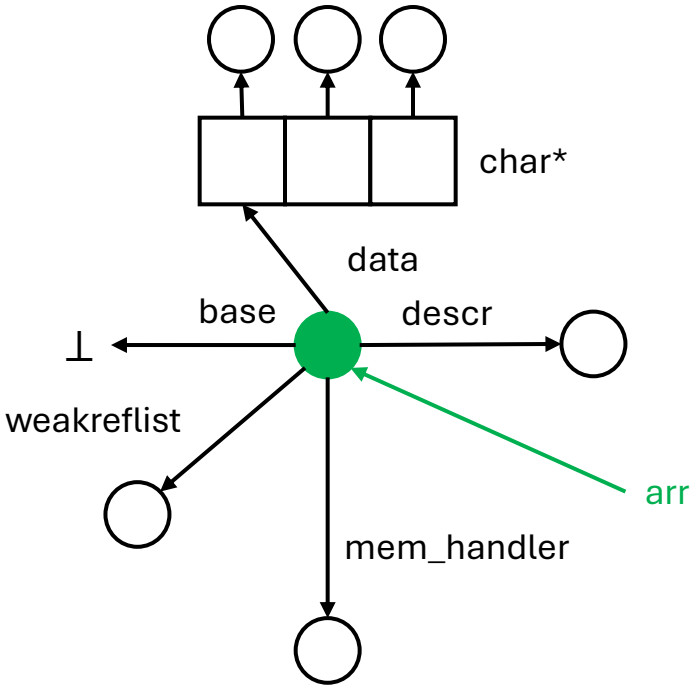
Second Assumption: The variable “tmp” is always in local due to this method usage.

LRC +3 +2 -5

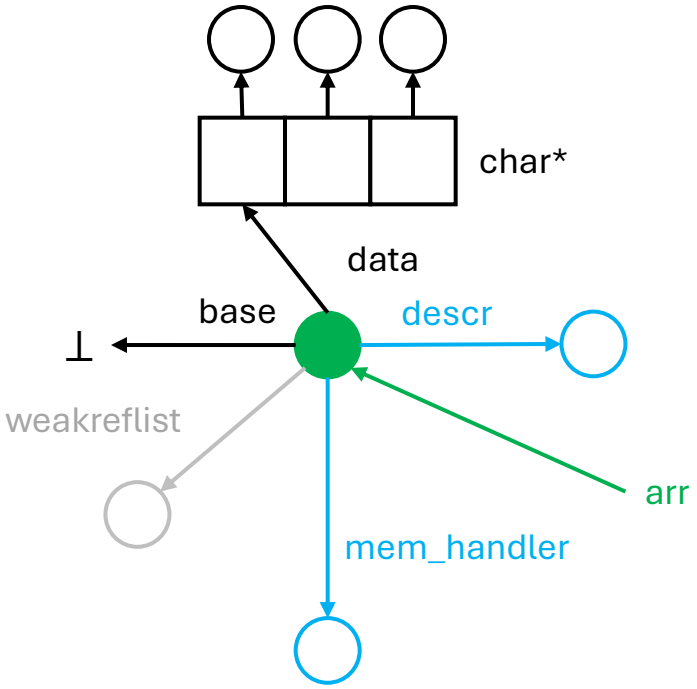


NumPy Migration

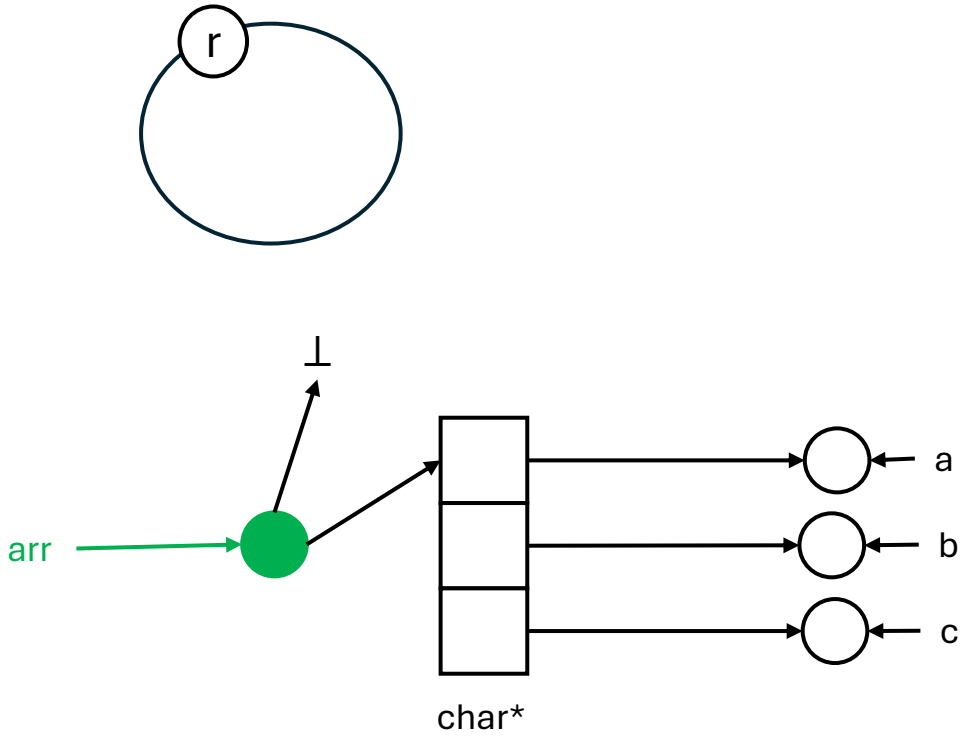
NumPy Array Structure



NumPy Array Structure

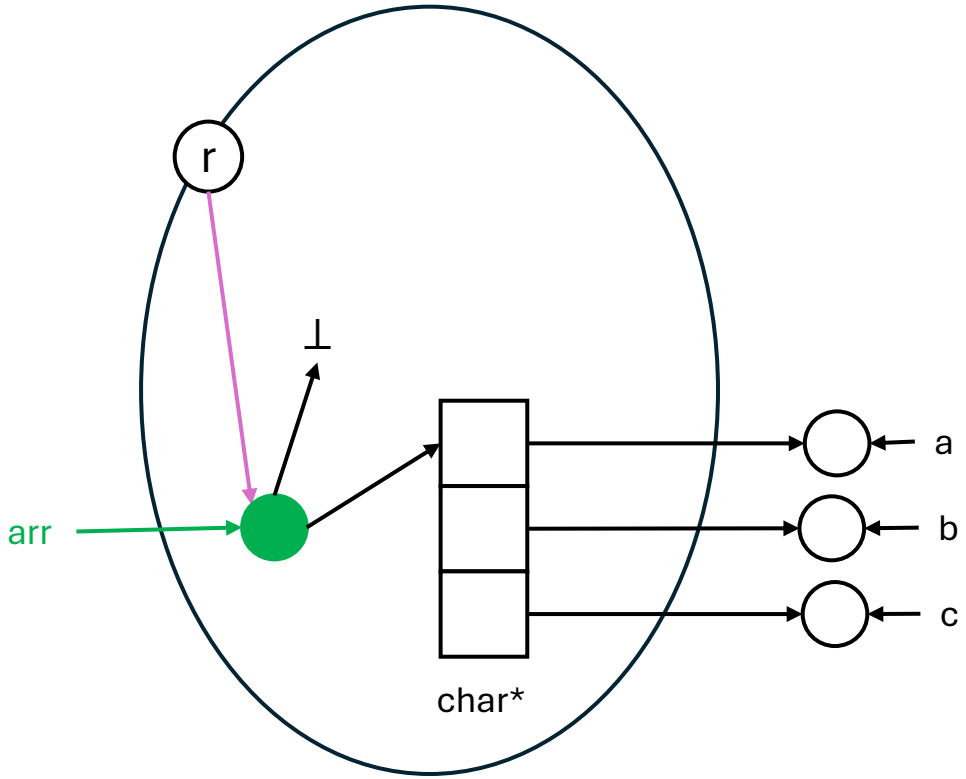


```
arr = np.array([a, b, c], dtype=object)
r.arr = arr
```



LRC +0

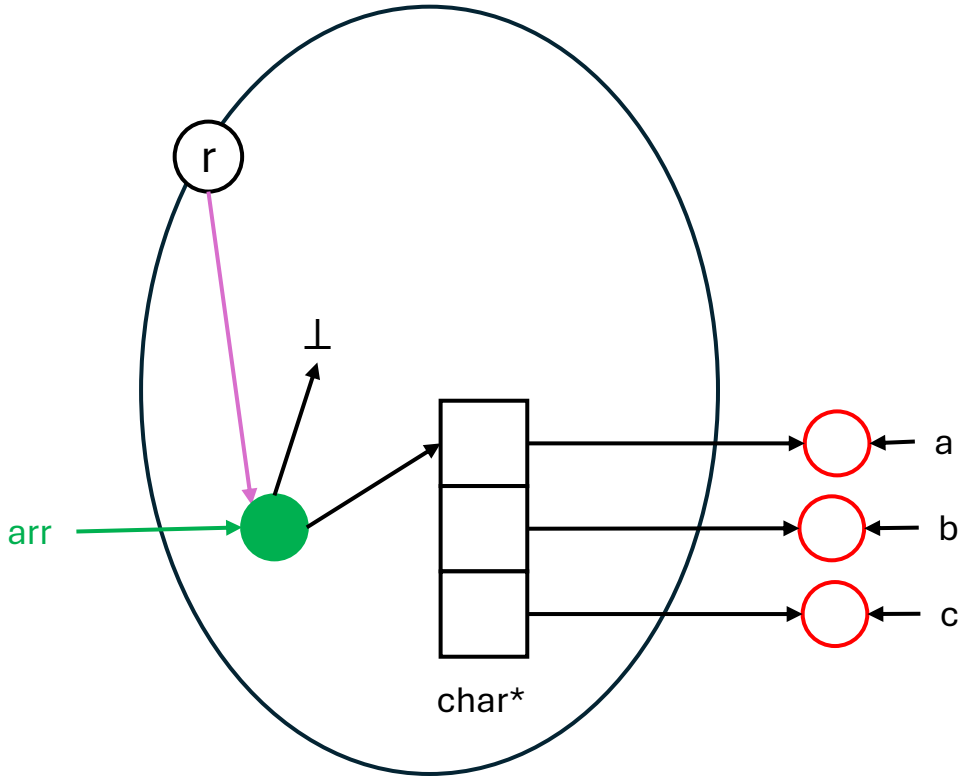
```
arr = np.array([a, b, c], dtype=object)
r.arr = arr
```



LRC +1

`tp_traverse` is Required

```
arr = np.array([a, b, c], dtype=object)
r.arr = arr
```

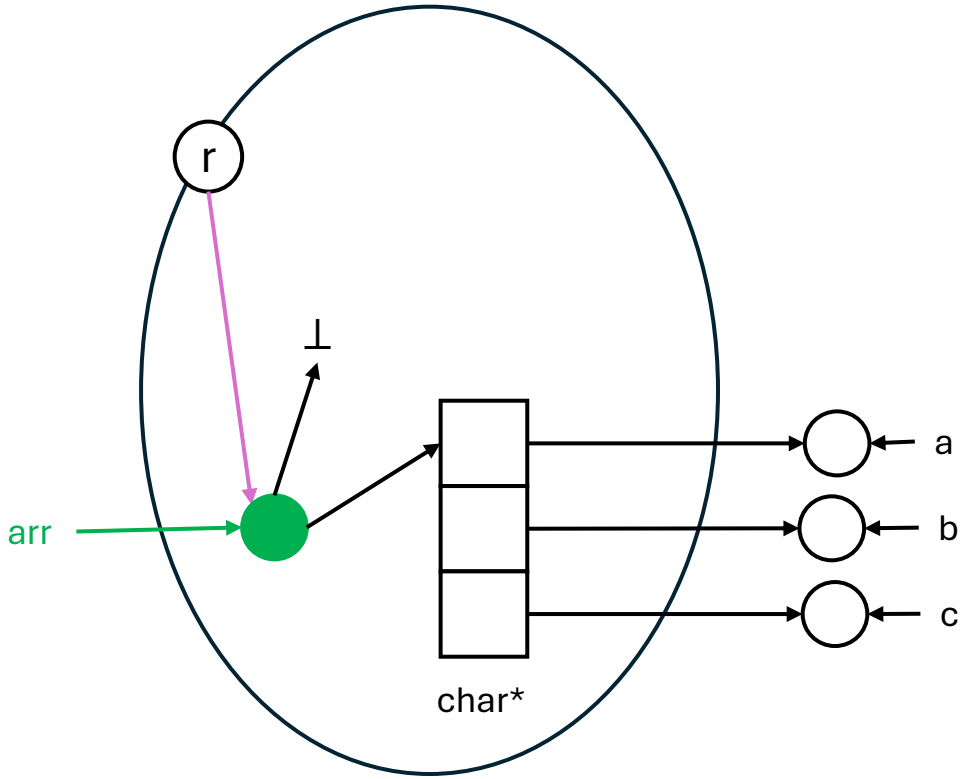


Incorrect !
- Ephemeral Property

LRC +1

`tp_traverse` is Required

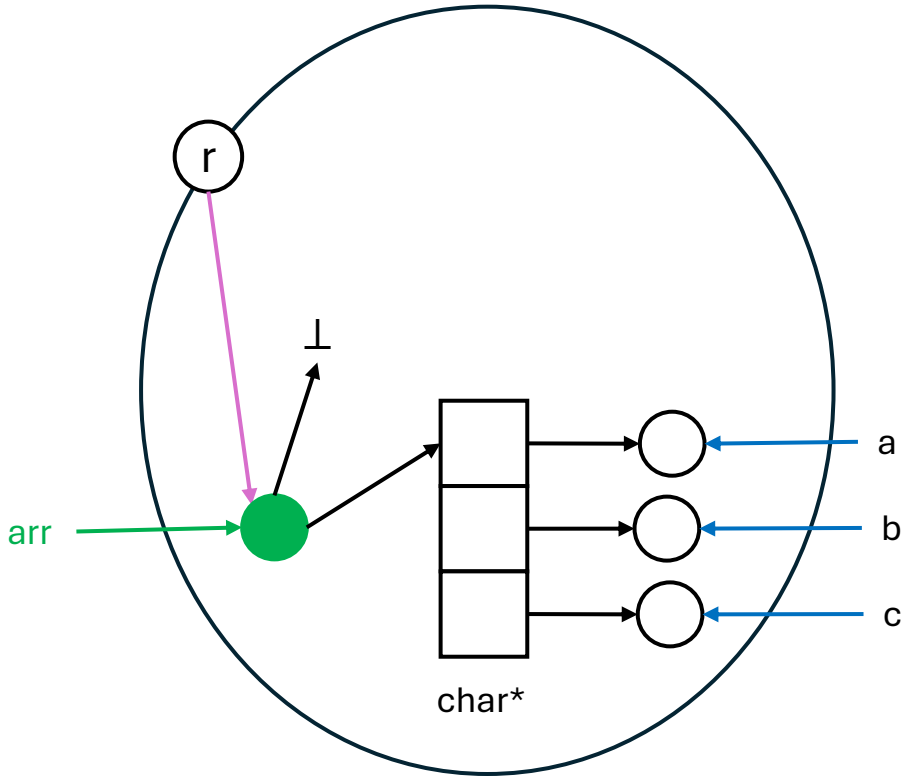
```
arr = np.array([a, b, c], dtype=object)
r.arr = arr
```



LRC +1

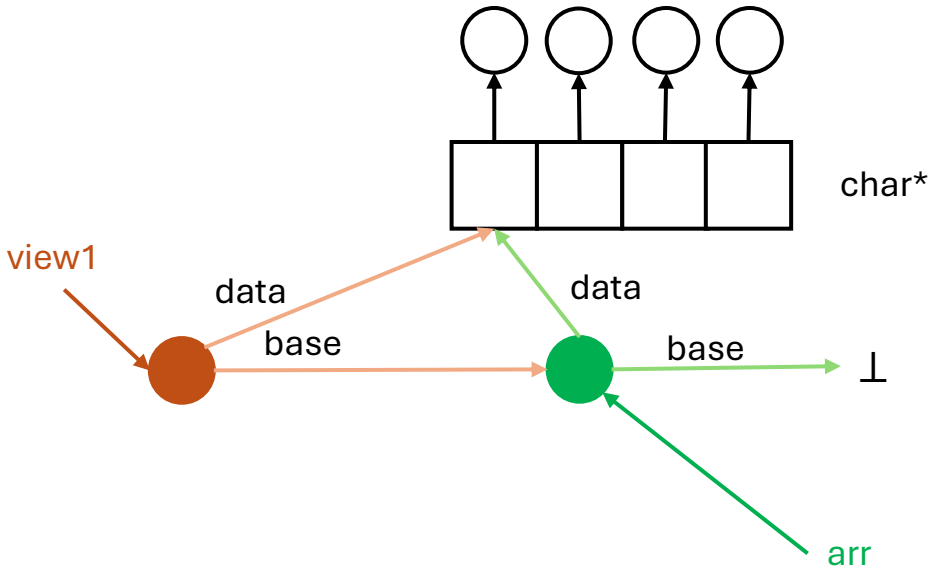
`tp_traverse` is Required

```
arr = np.array([a, b, c], dtype=object)
r.arr = arr
```



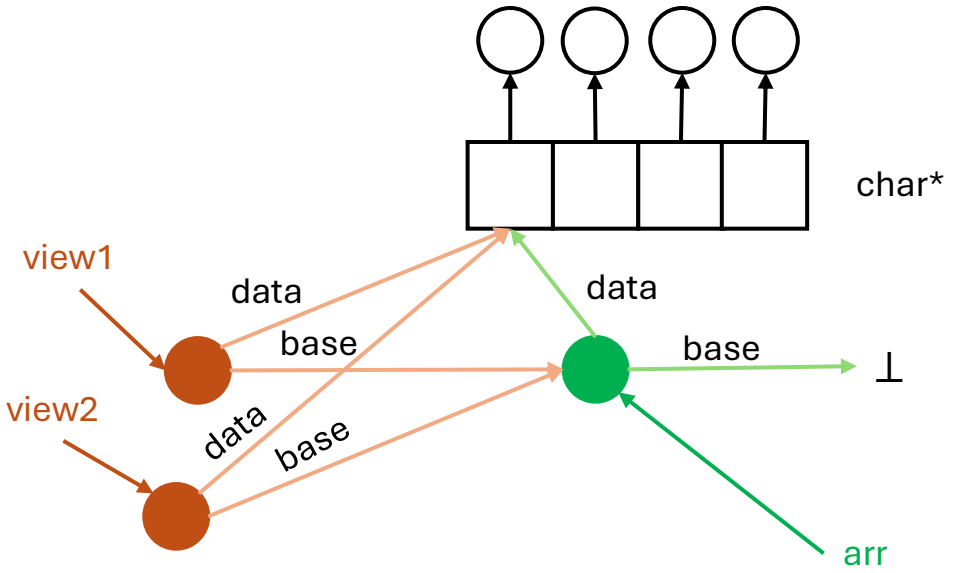
LRC +1+3

NumPy Structure - View

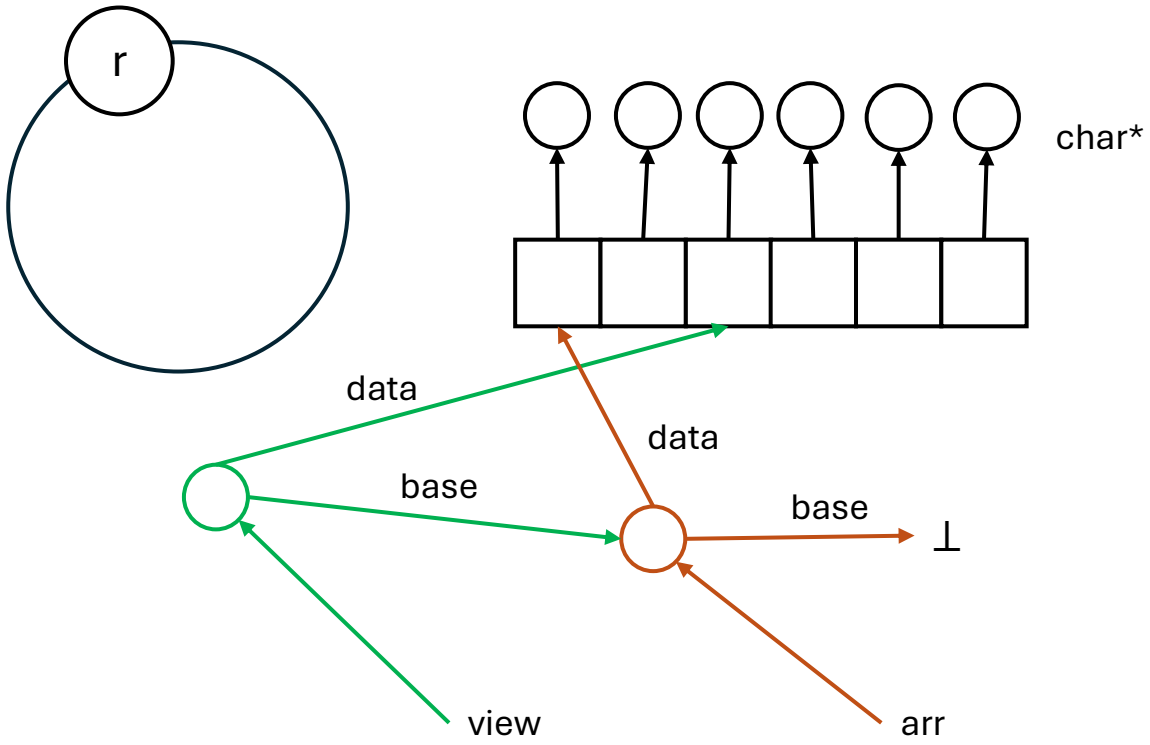



```
arr = np.array([A(), A(), A(), A()], dtype=object)
view1 = arr[0:3]
```

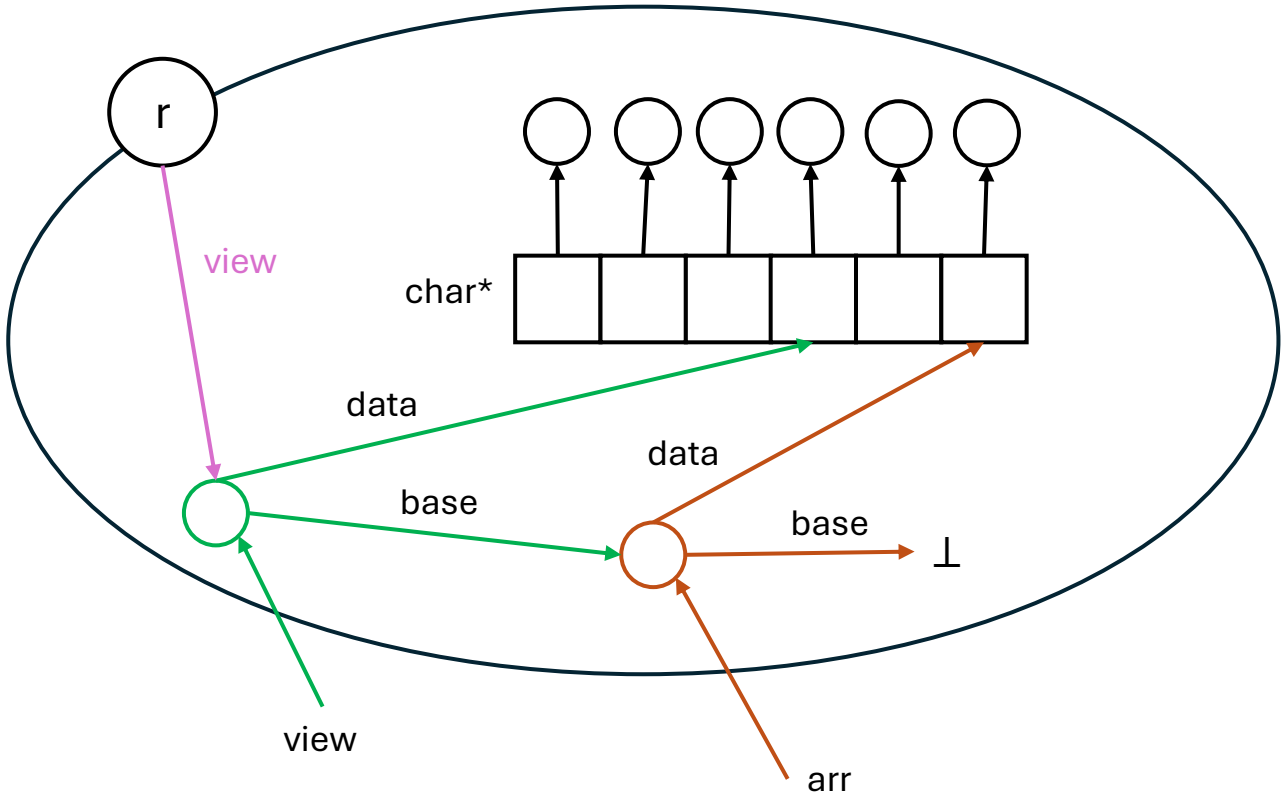
NumPy Structure - View



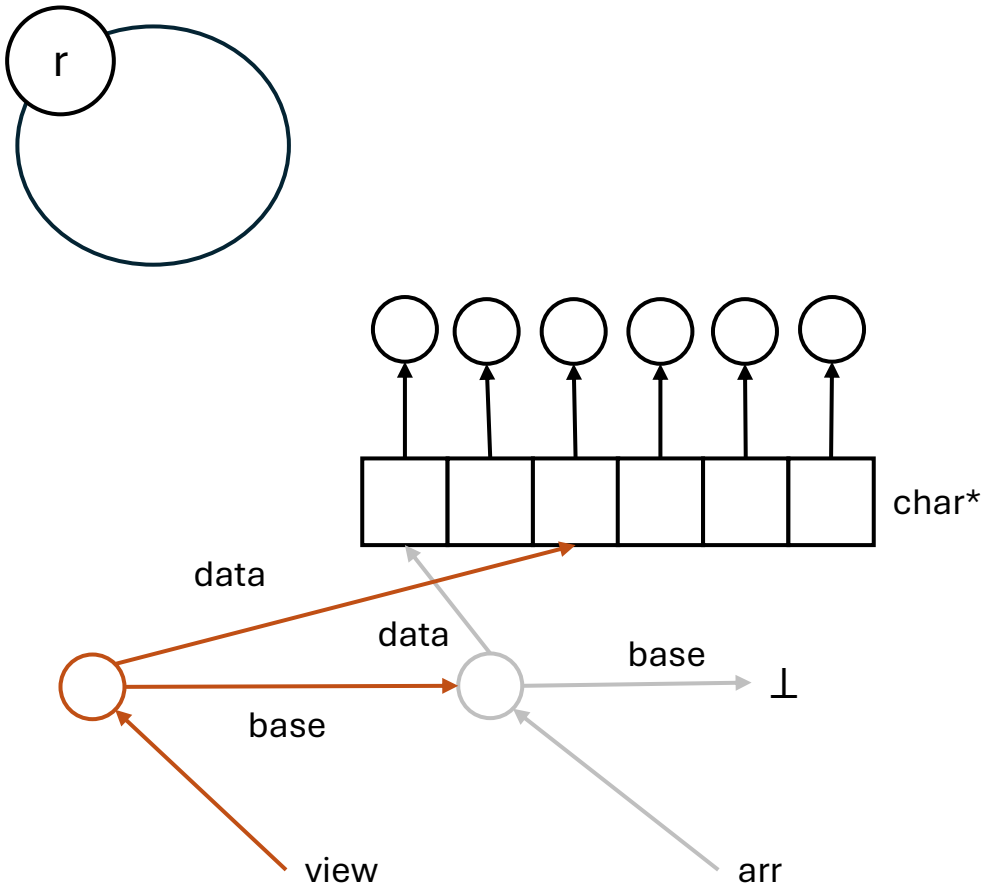

```
arr = np.array([A(), A(), A(), A()], dtype=object)
view1 = arr[0:3]
view2 = view1[0:2]
```



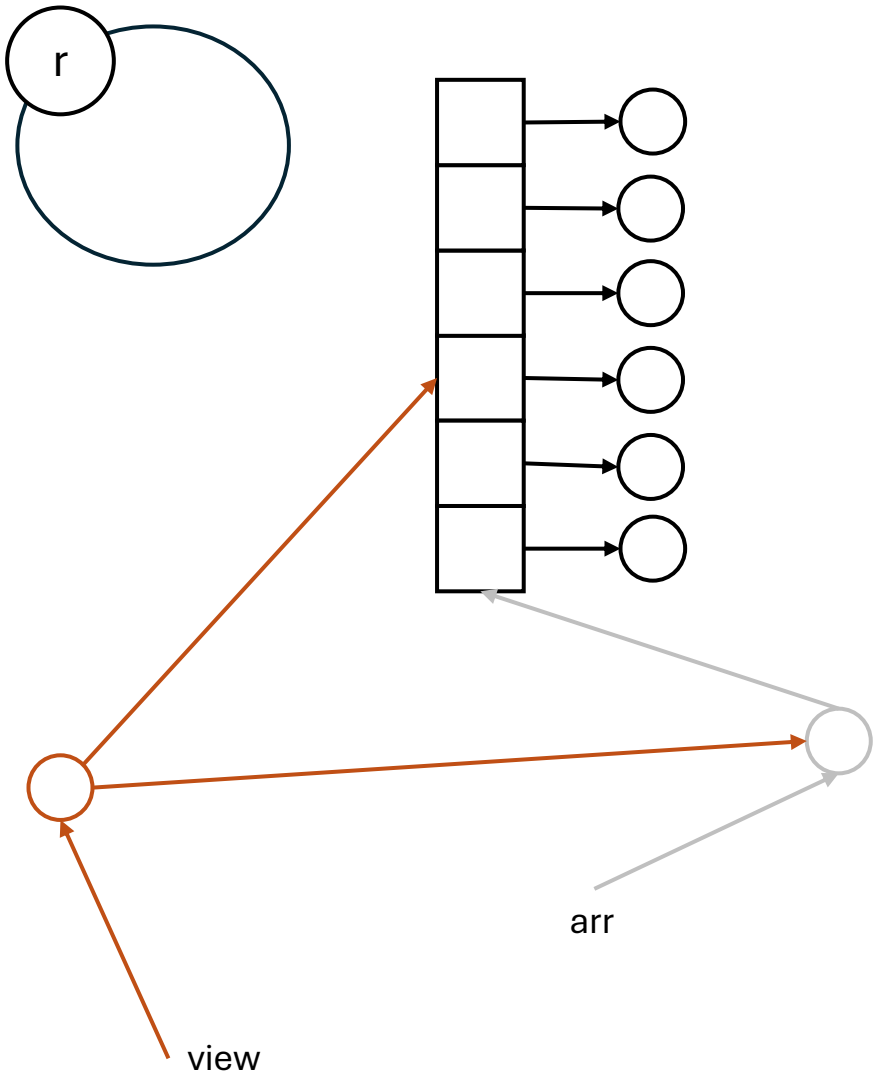

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
```



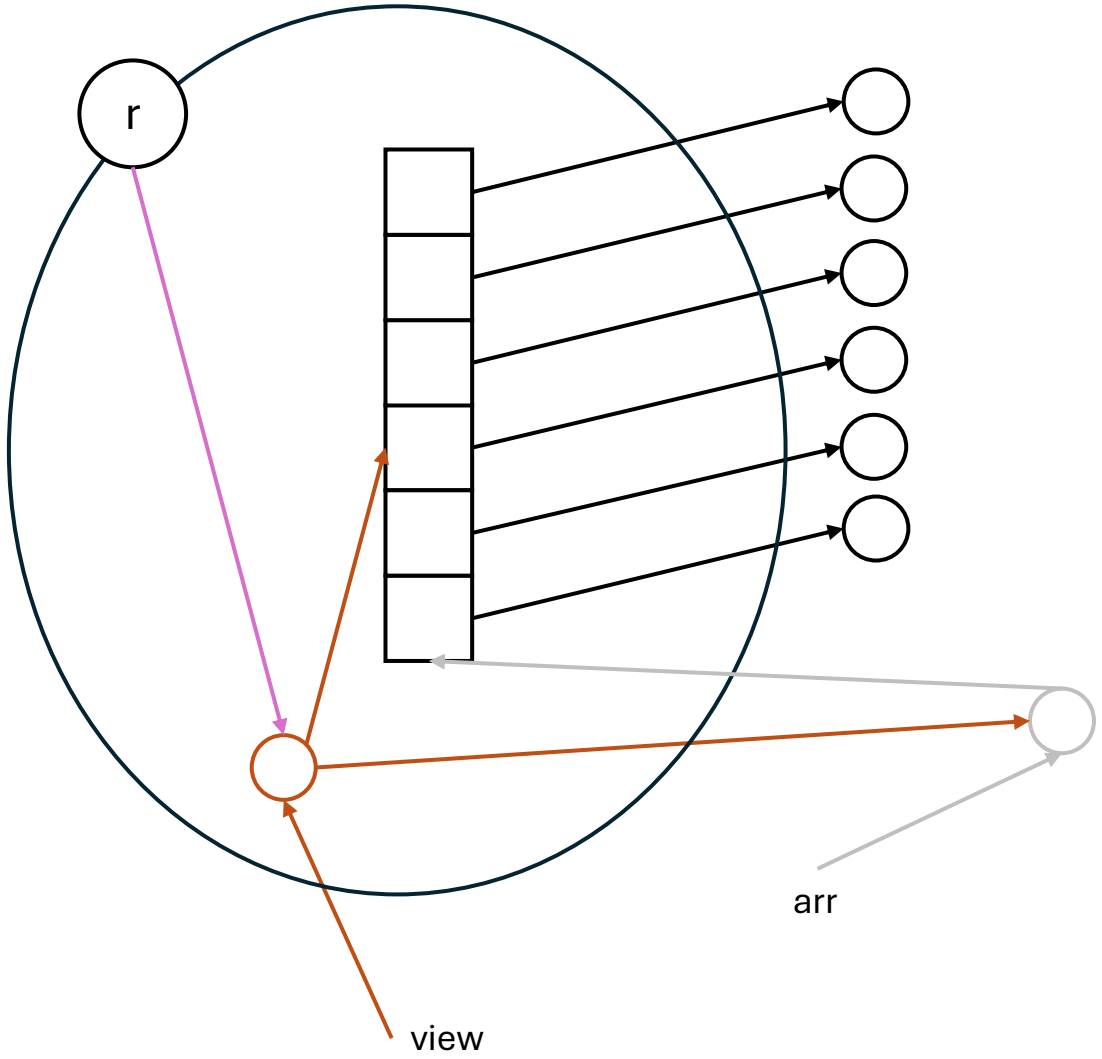

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



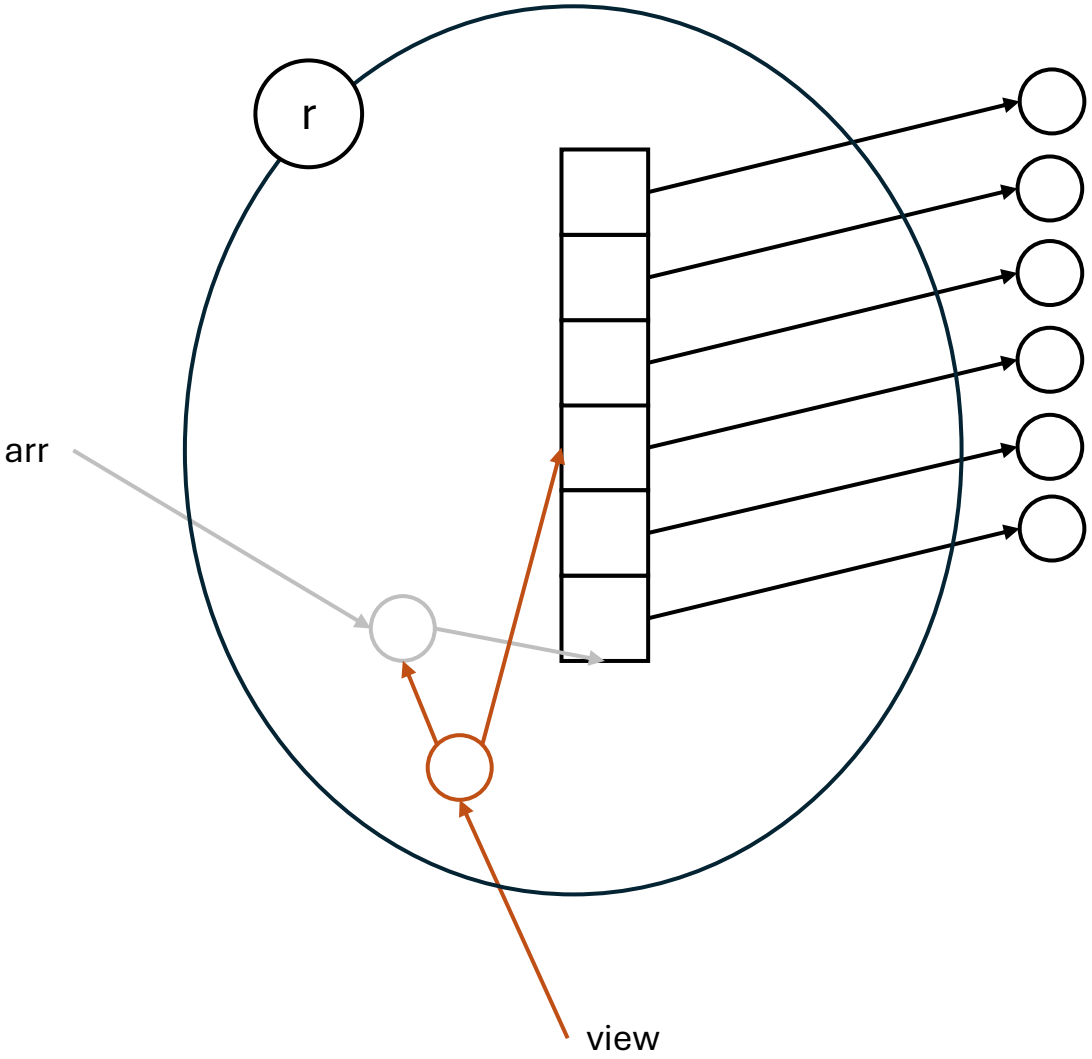

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



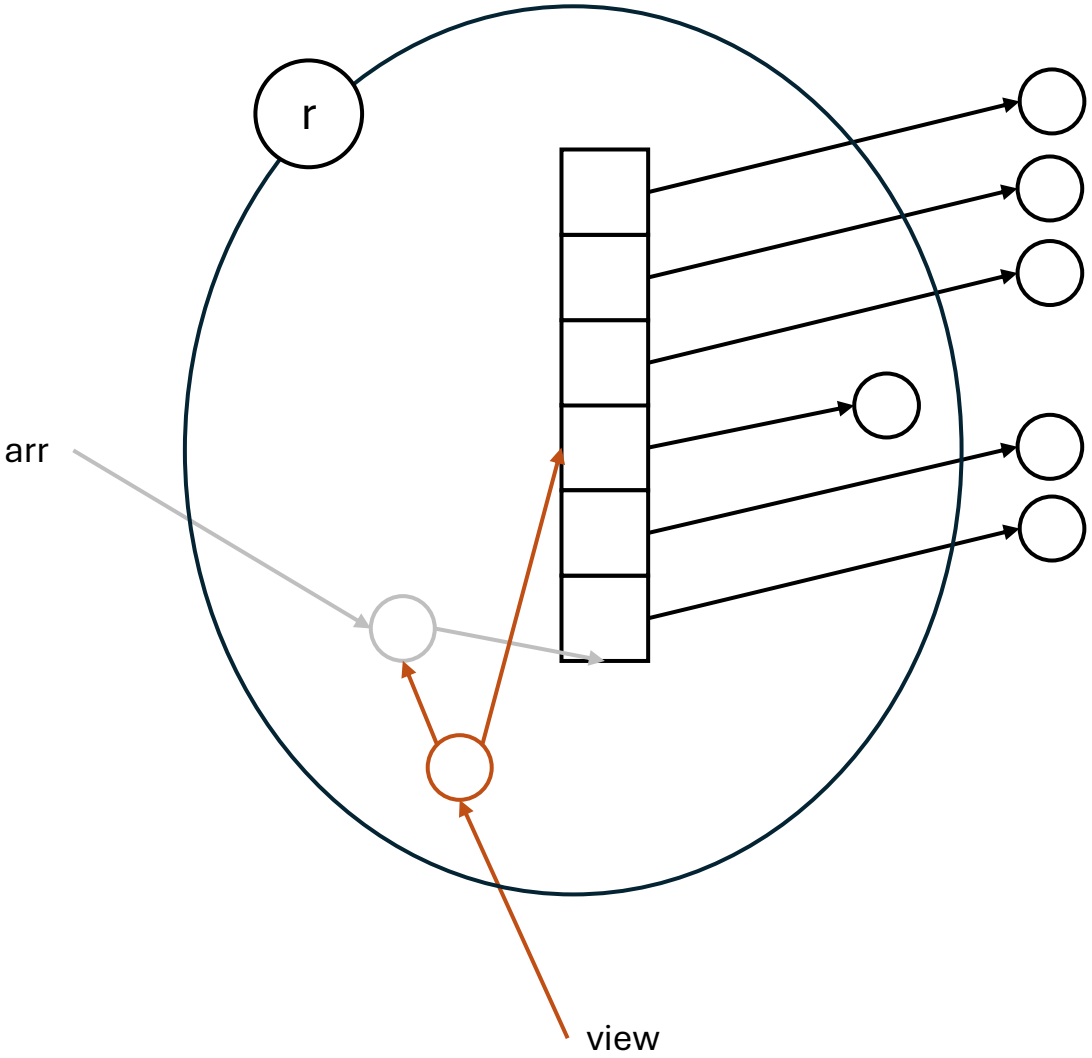

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



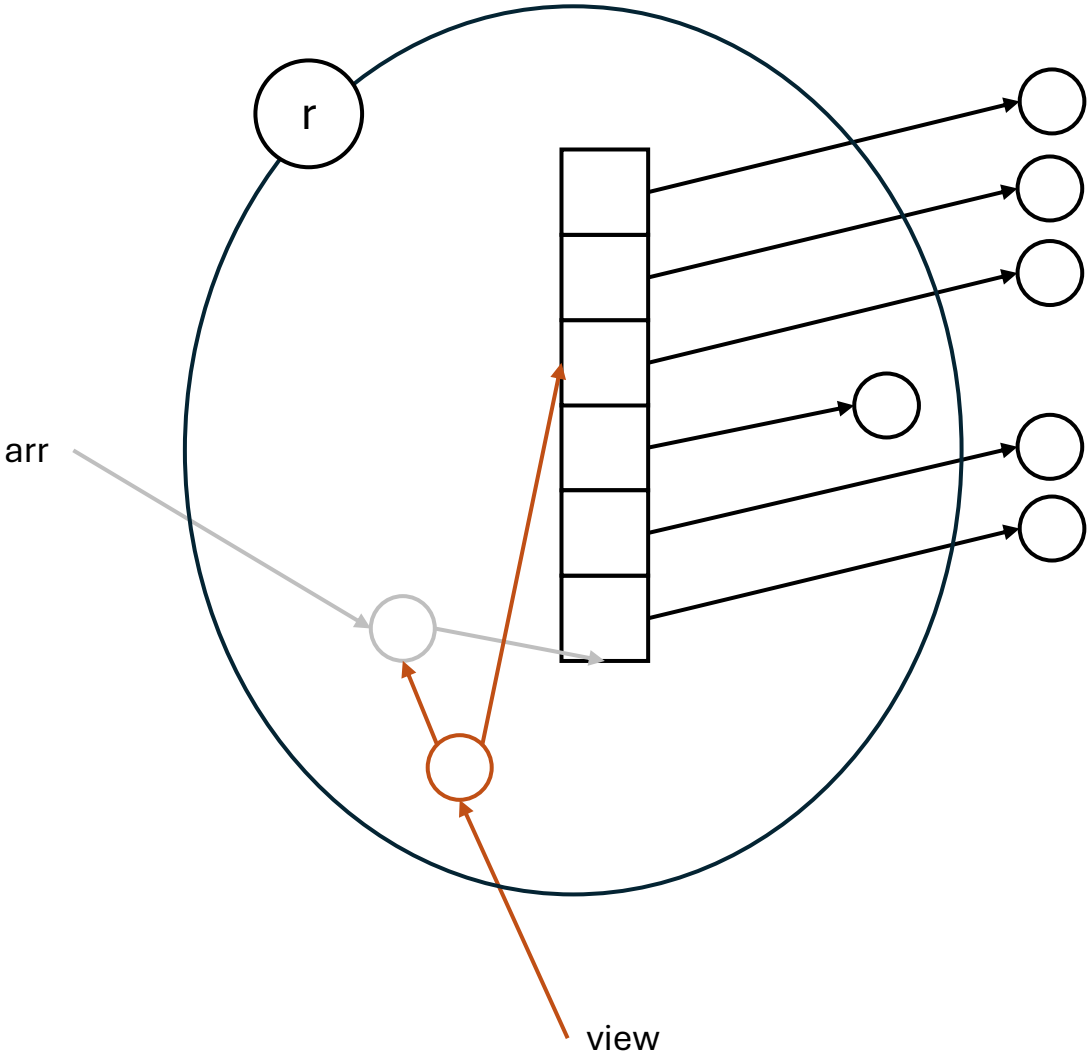

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



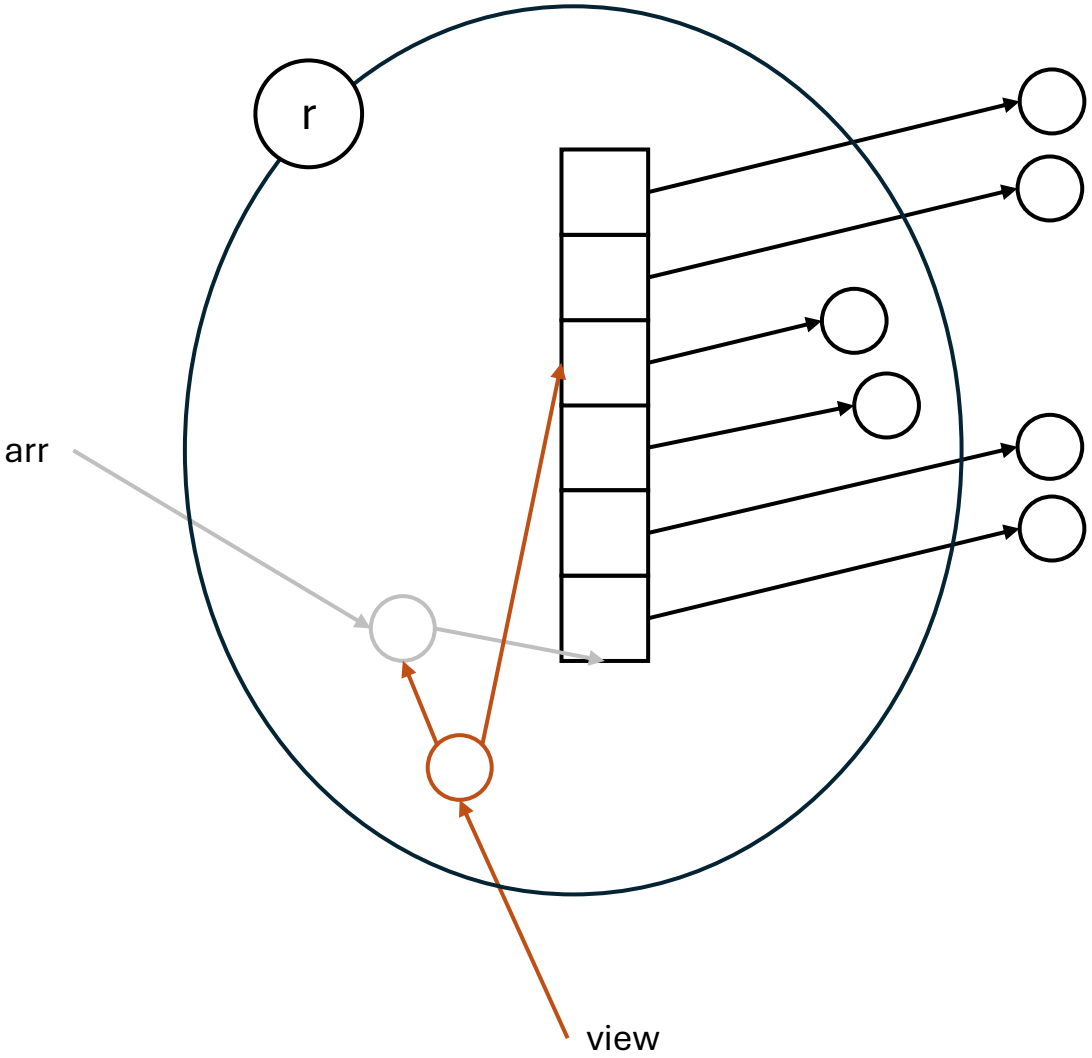

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



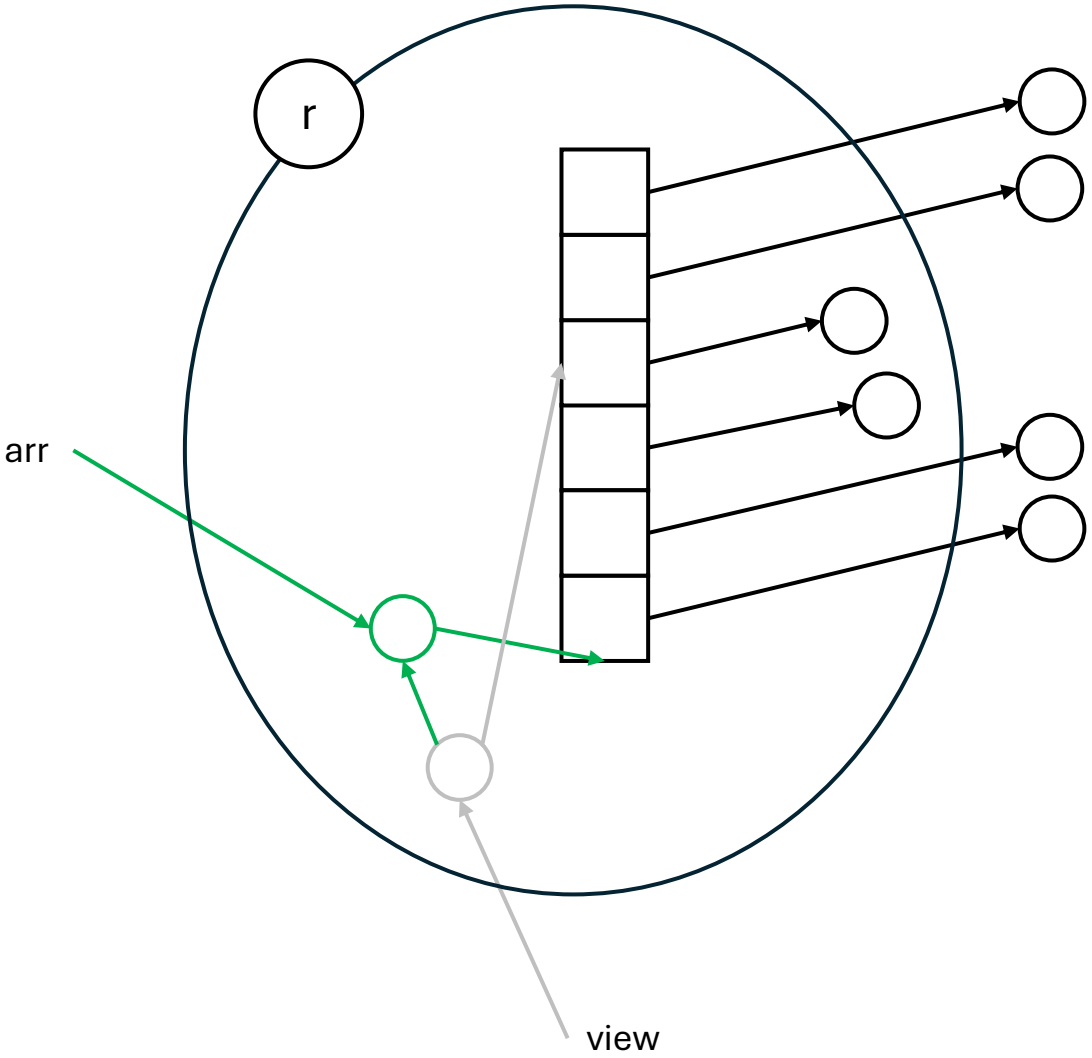

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



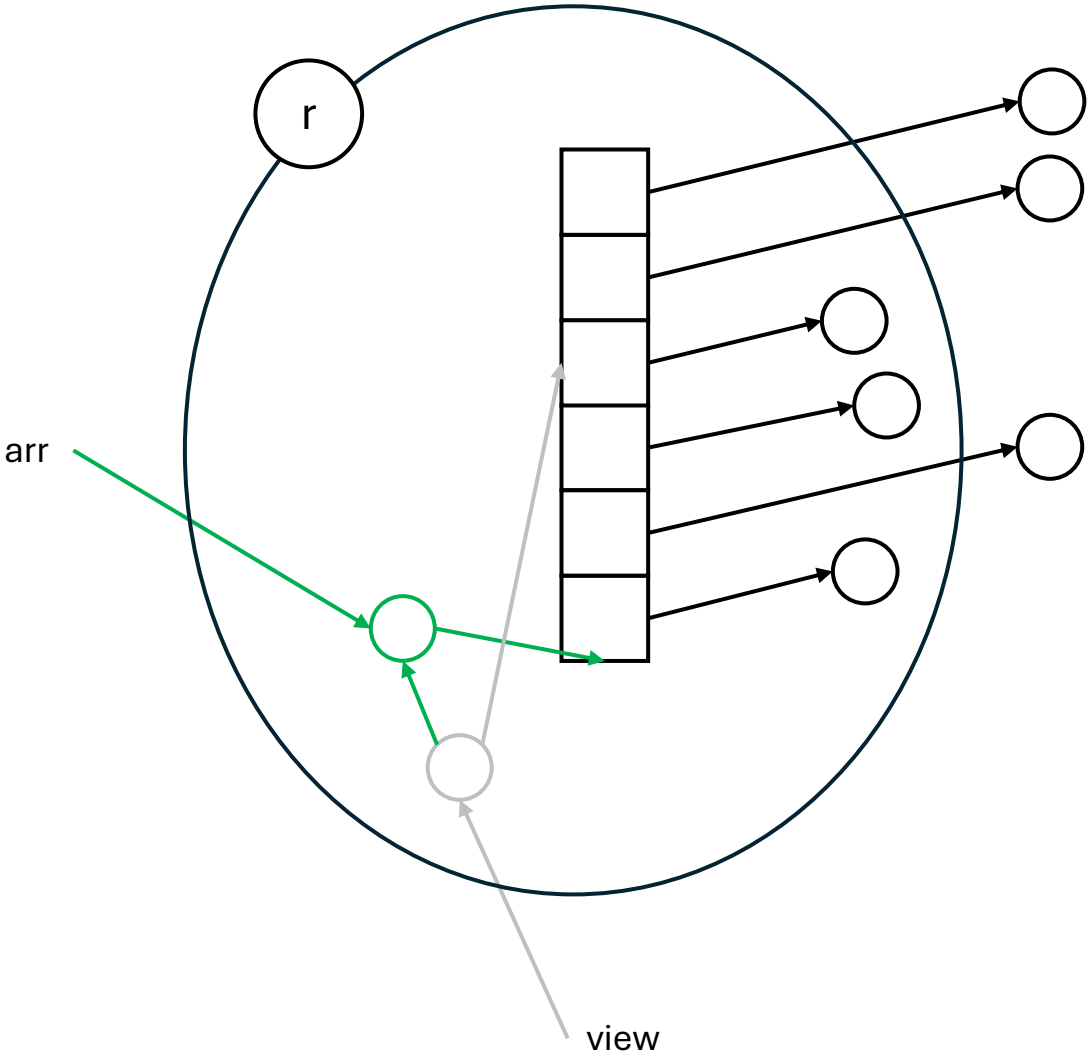

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



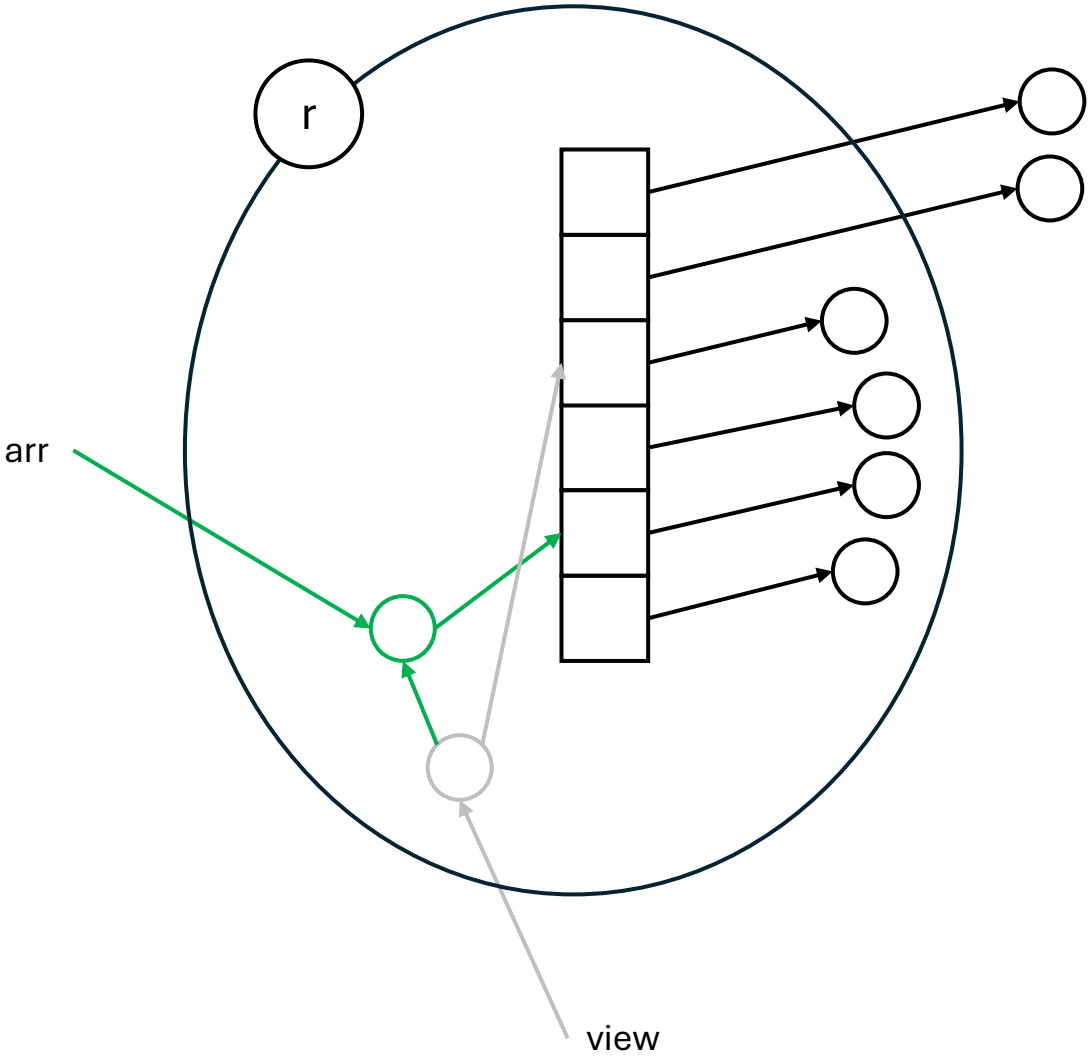

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



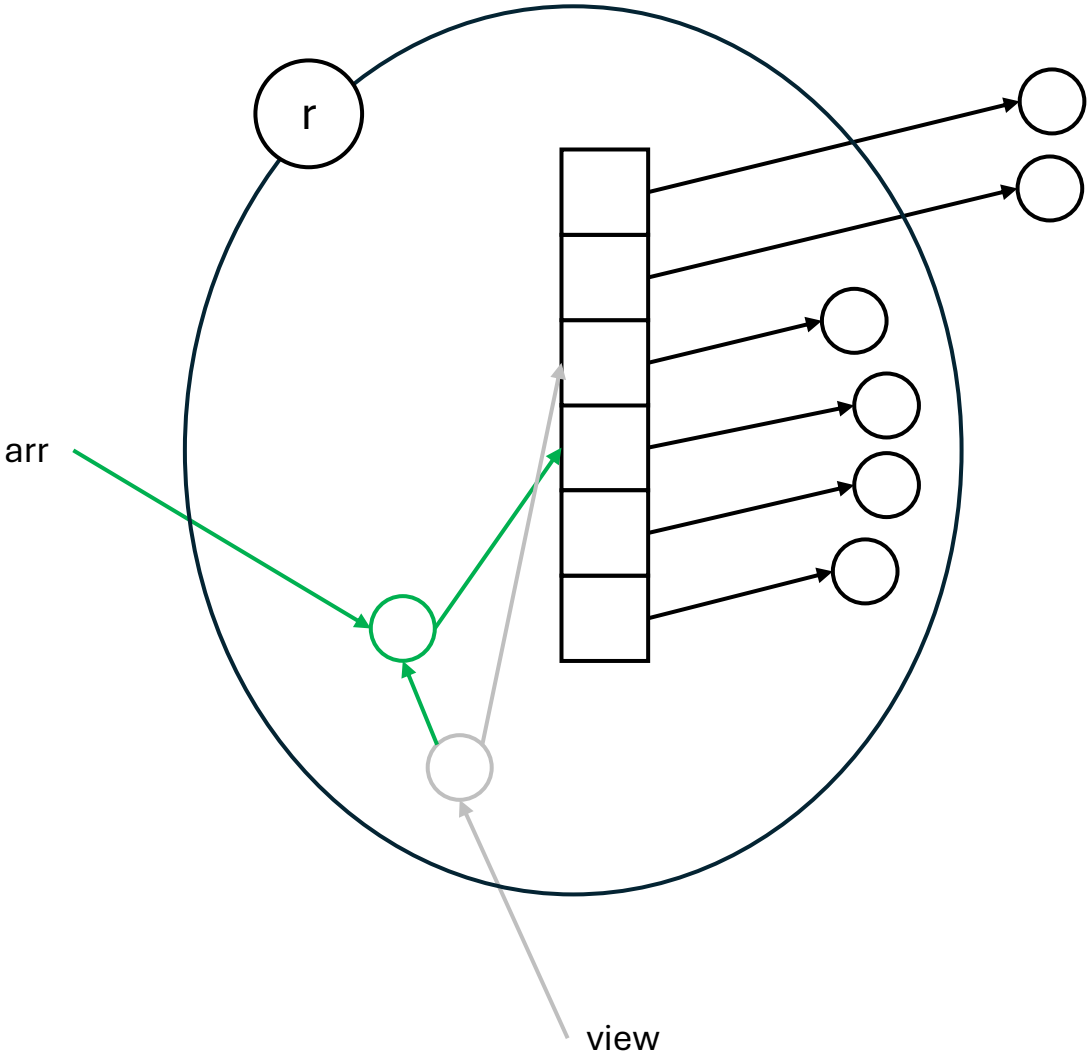

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```




```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

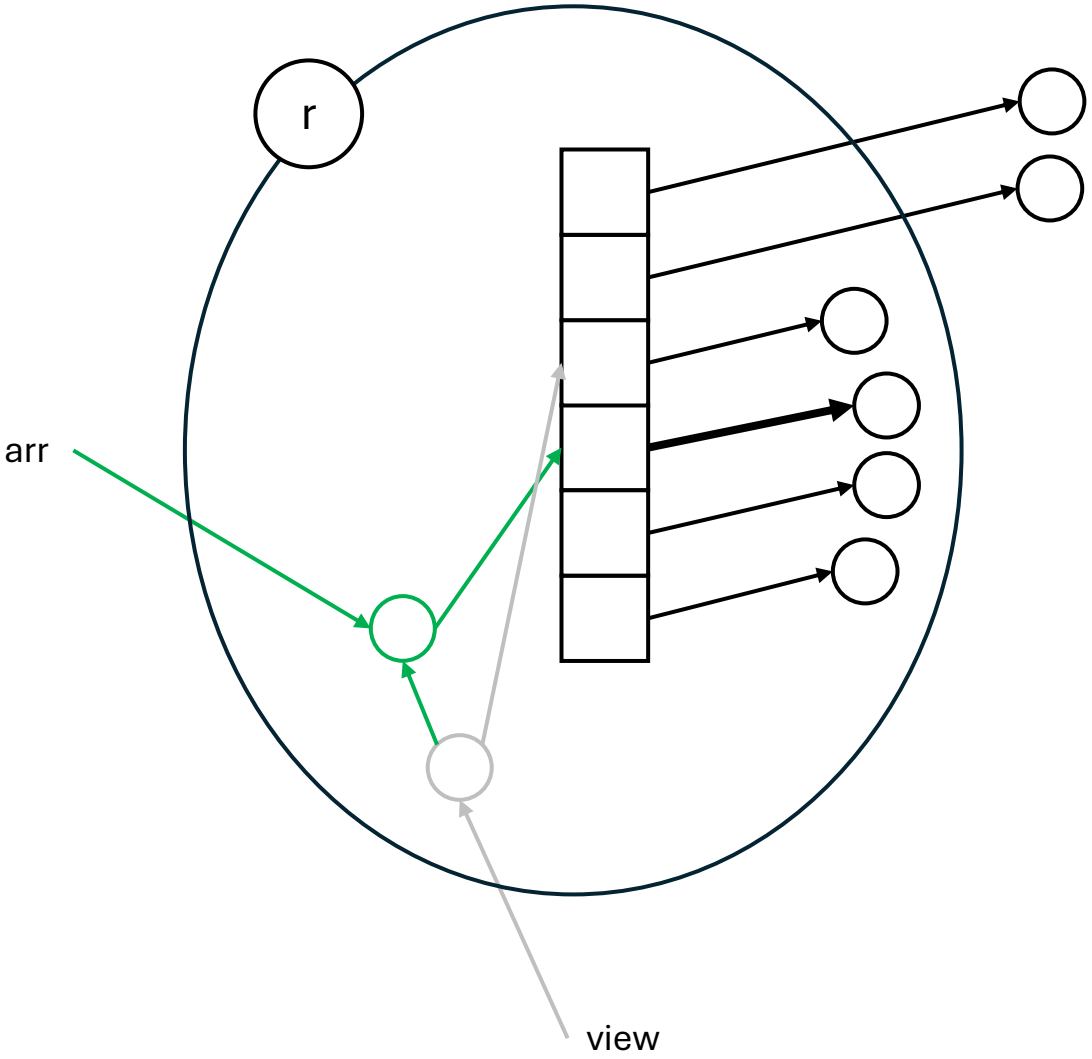



```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```



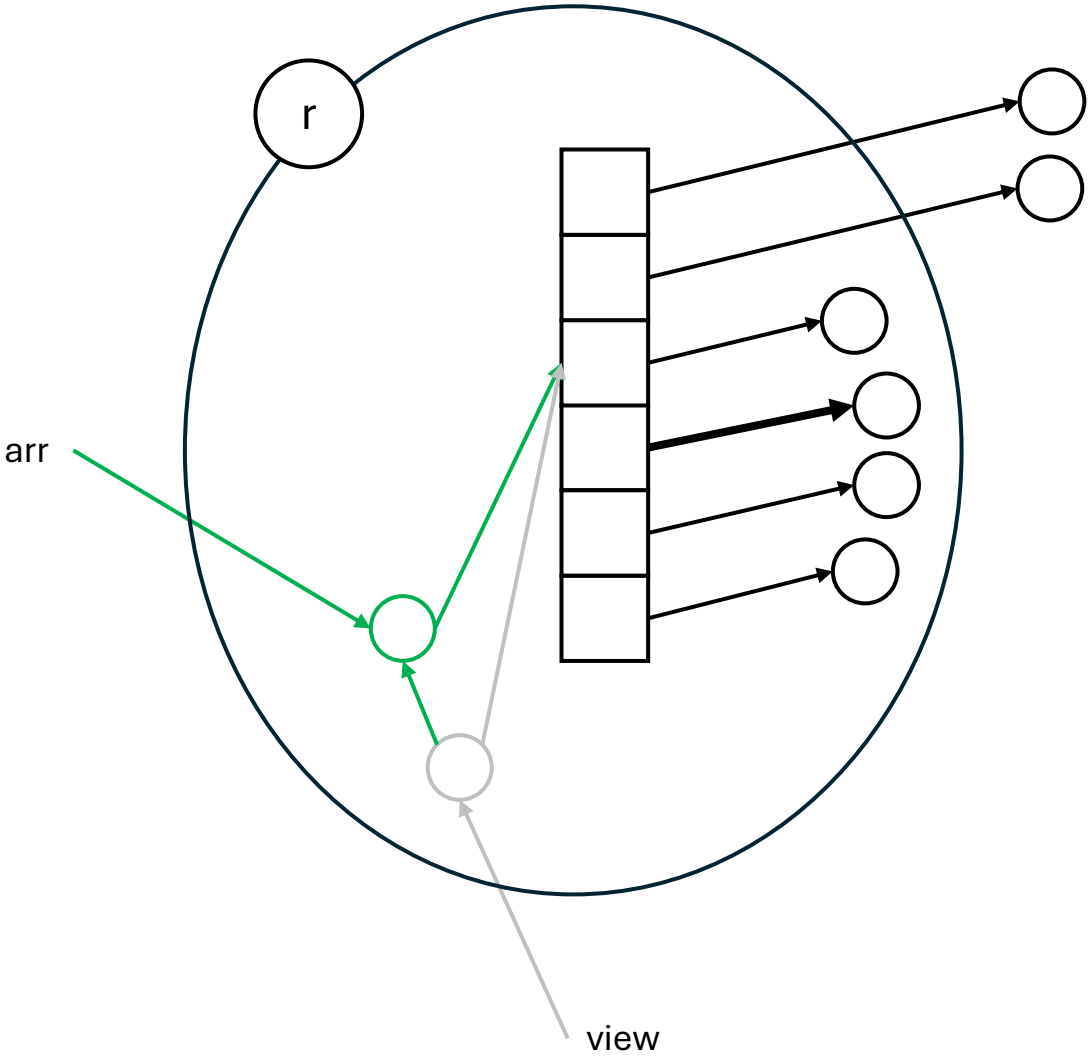

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal



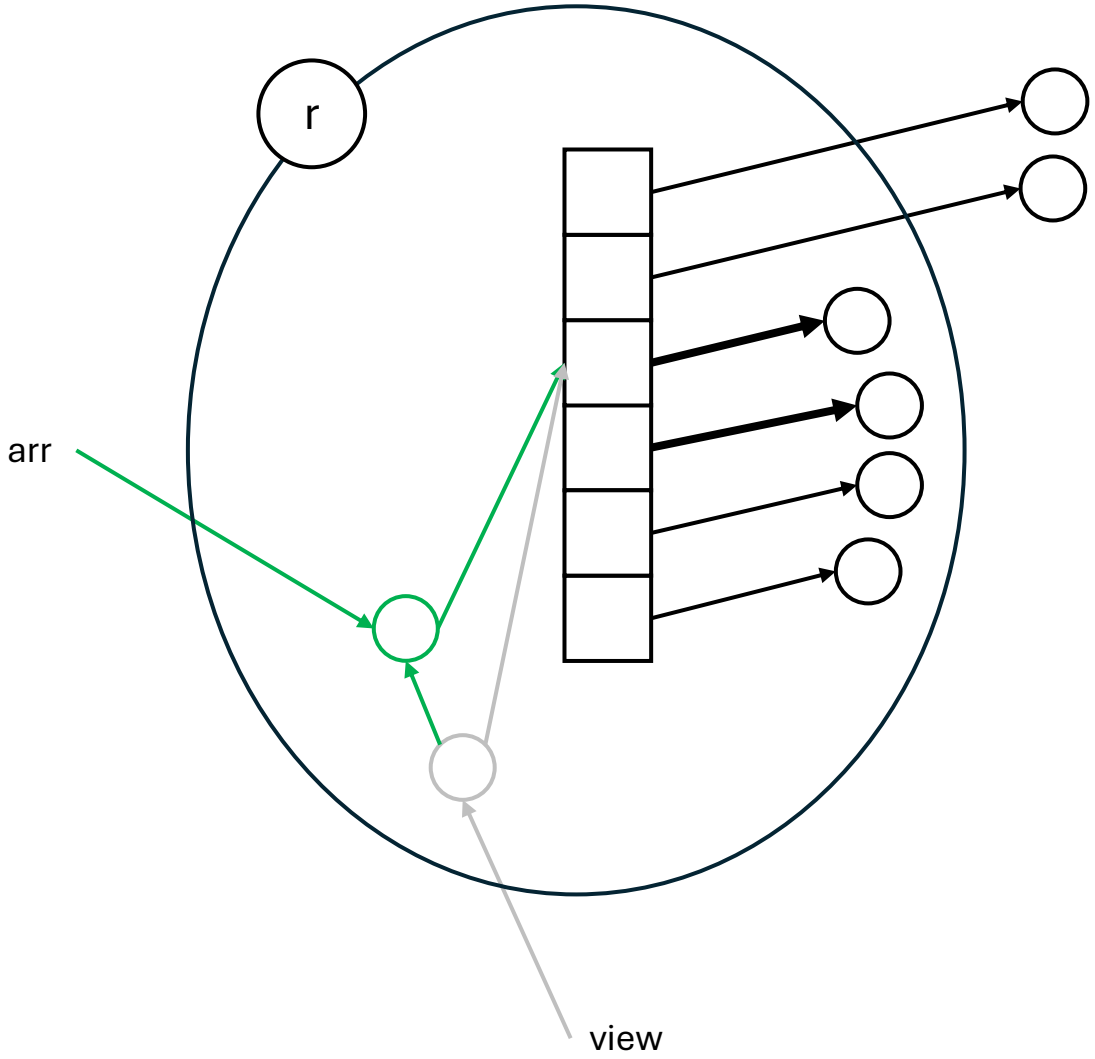

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal



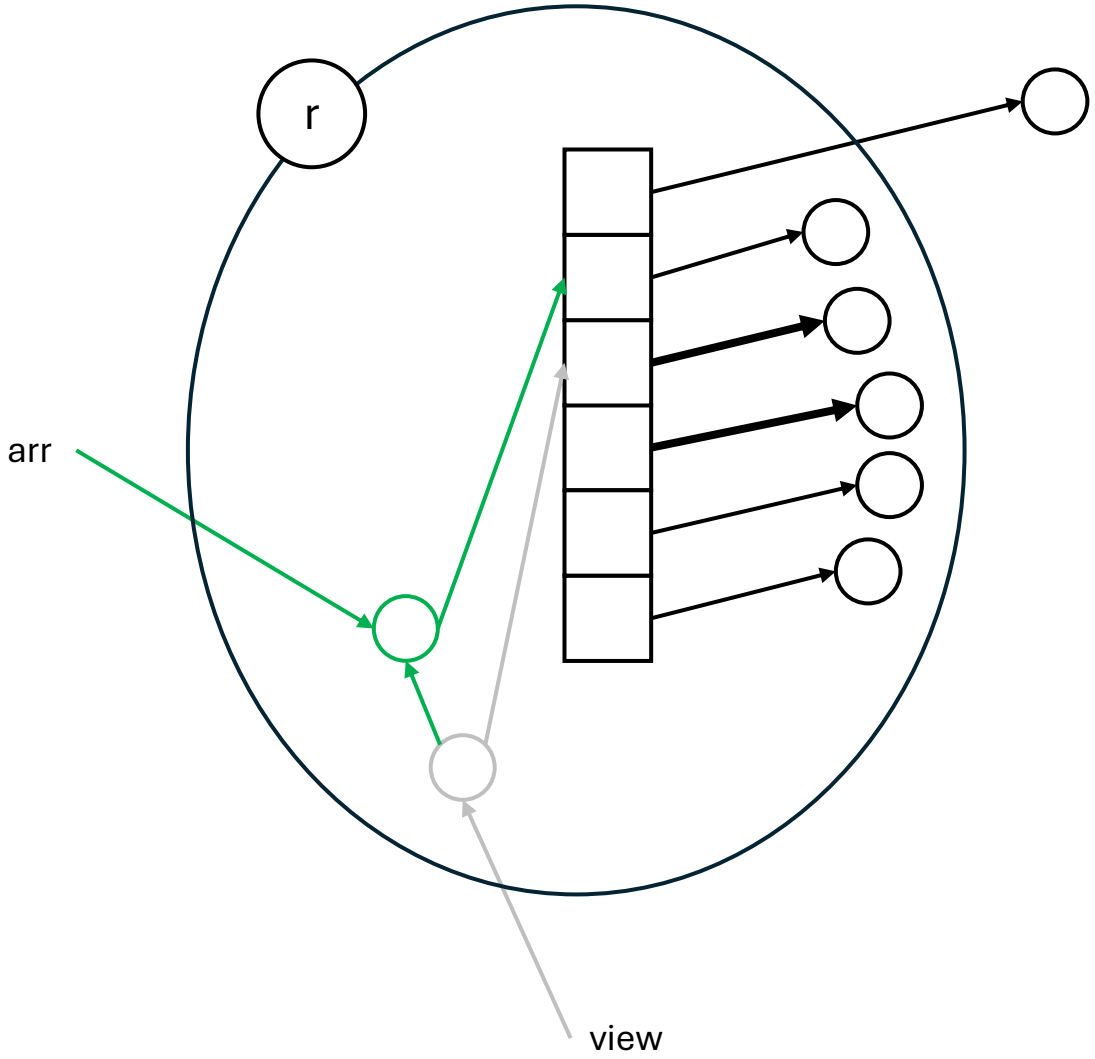

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal



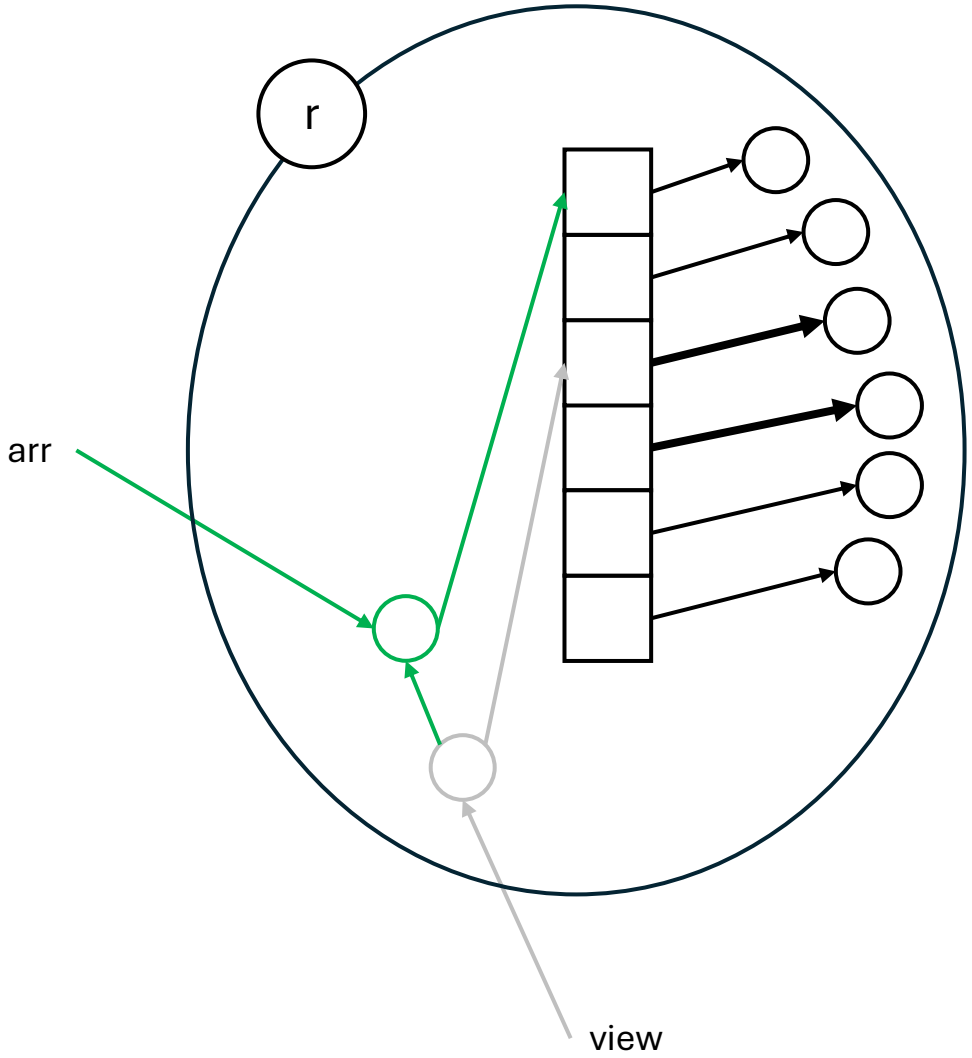

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal



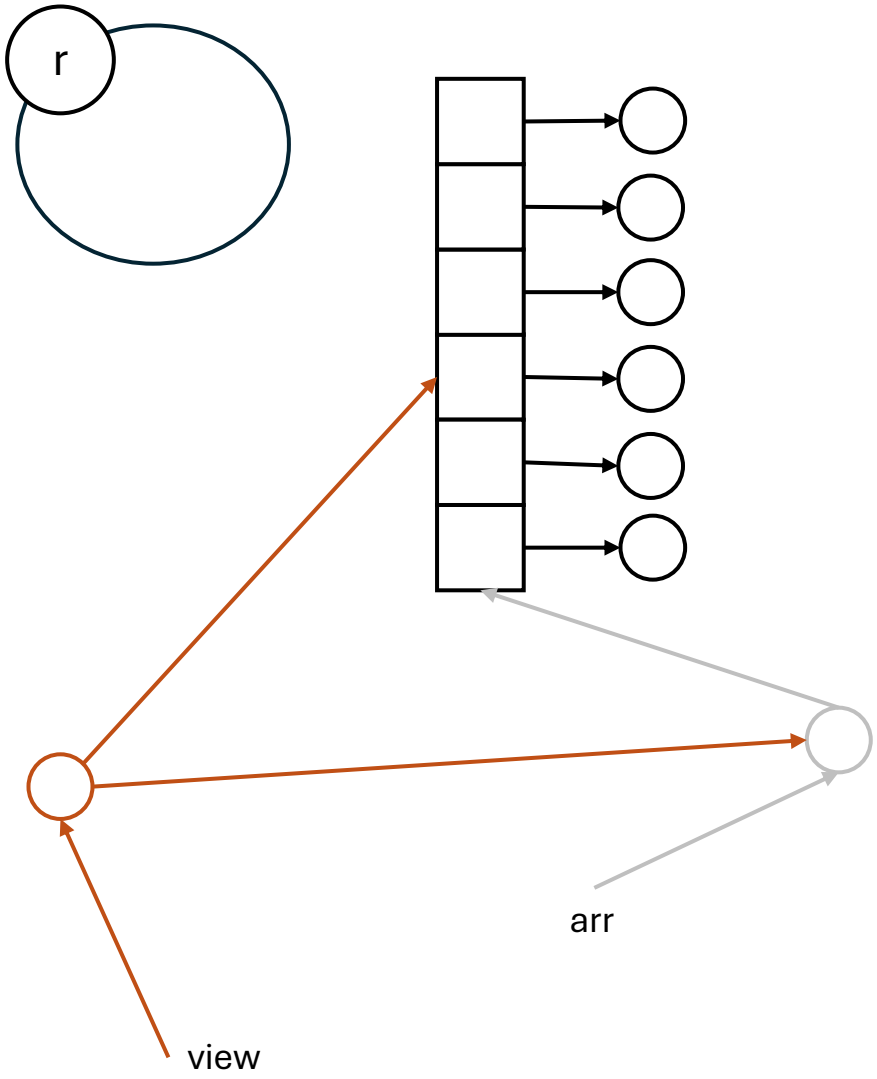

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal



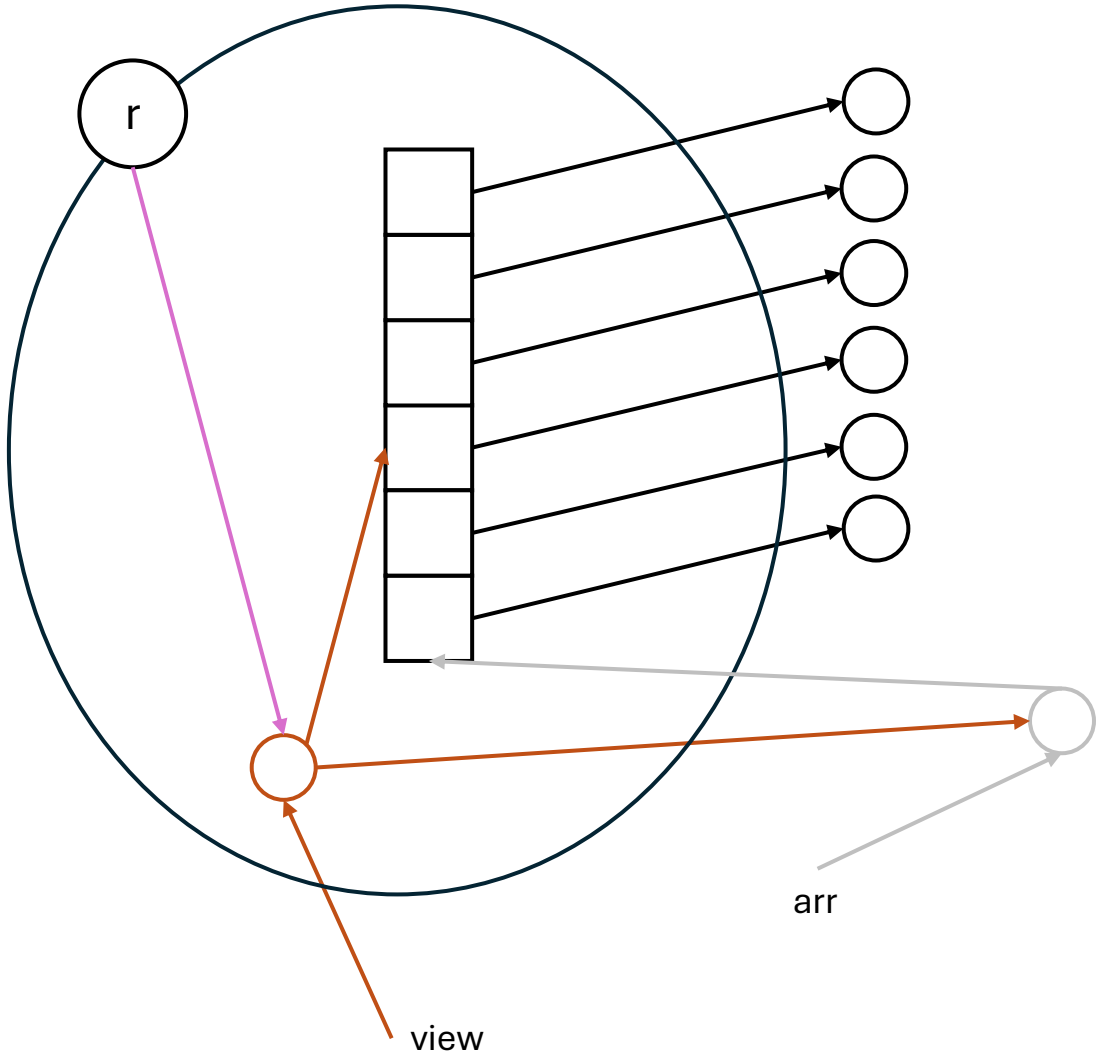

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal - Solution



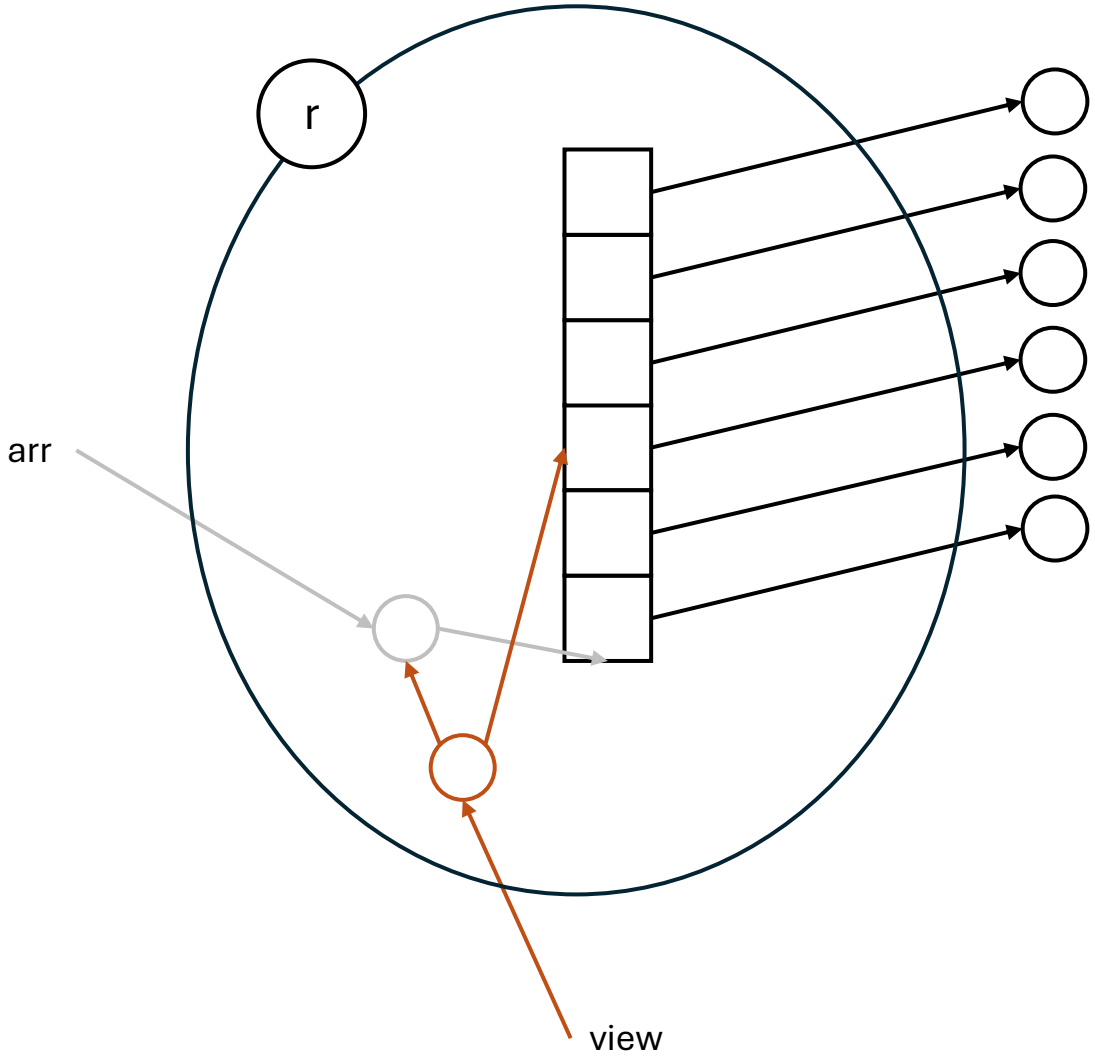

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal - Solution



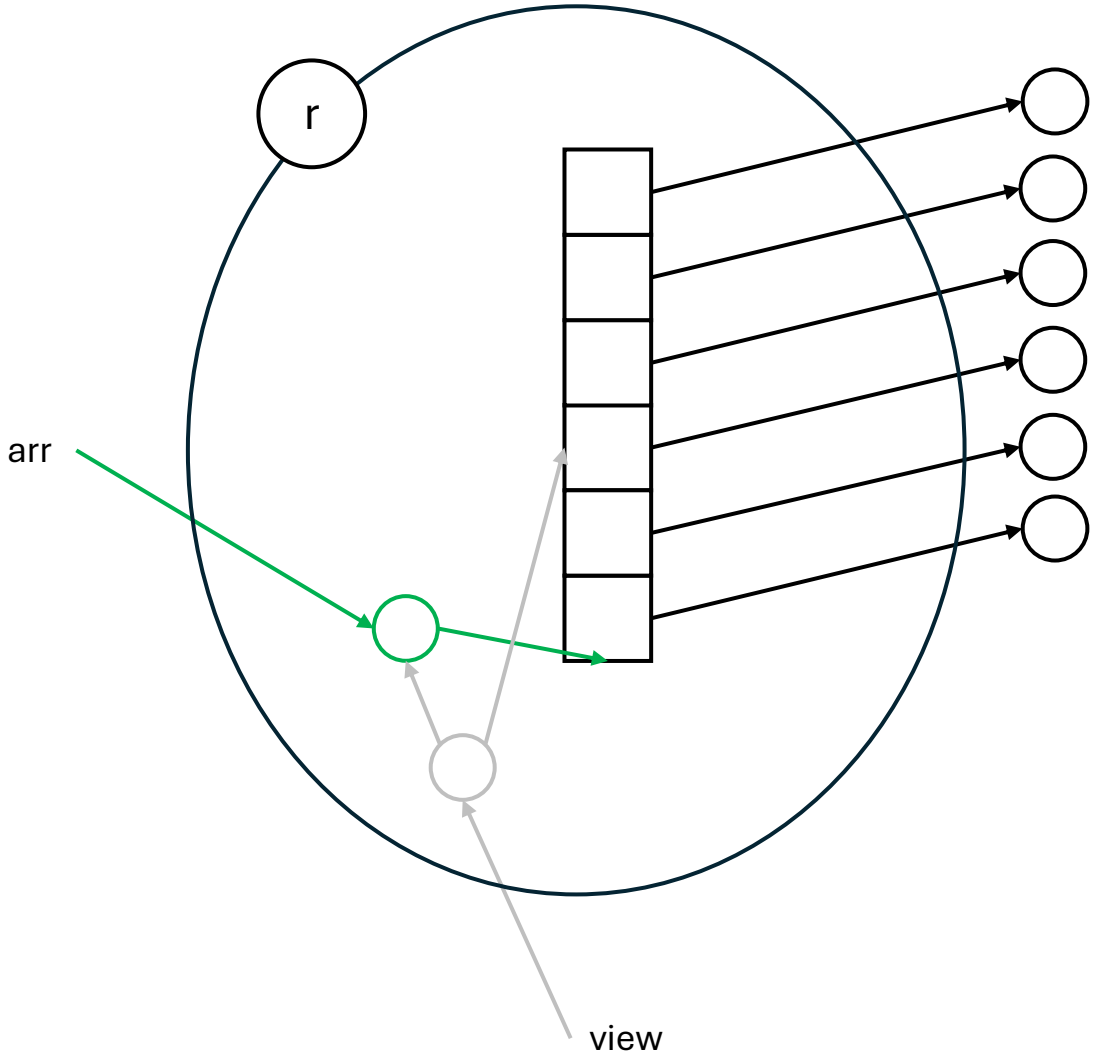

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal - Solution



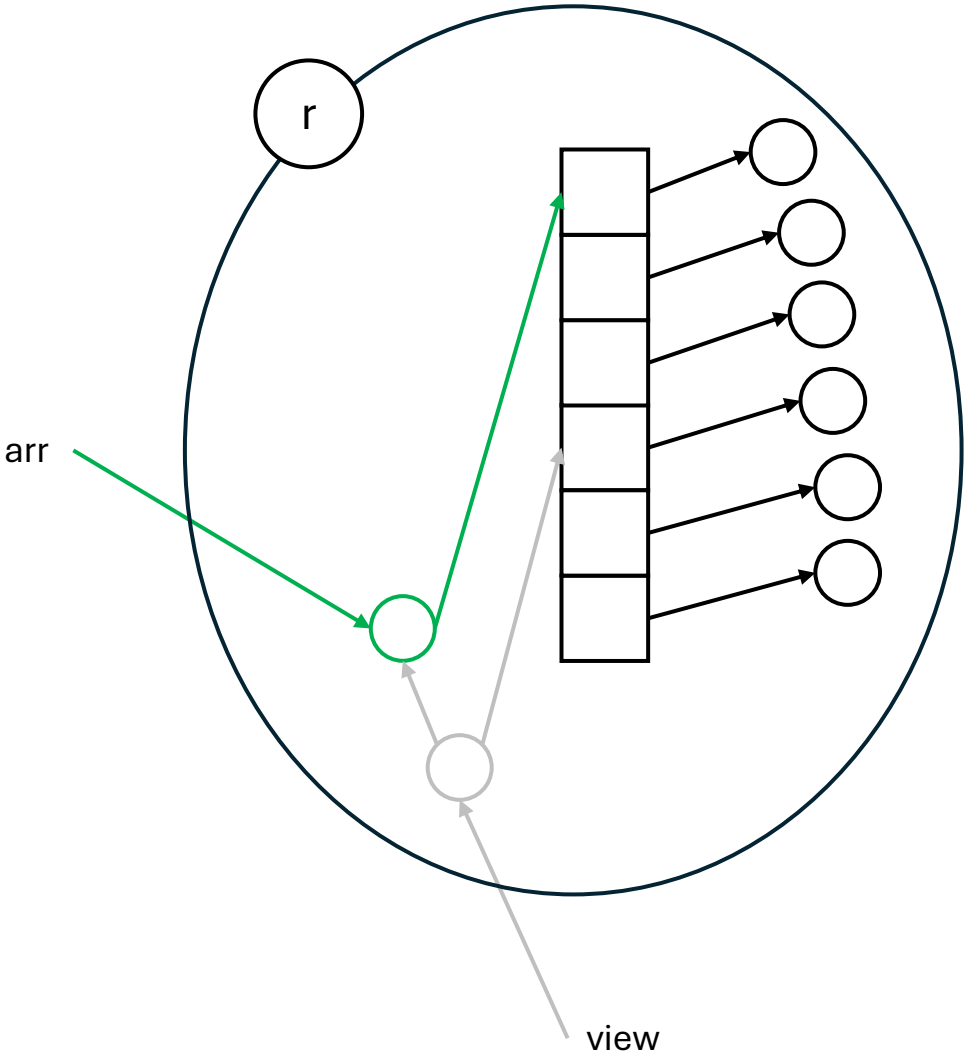

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal - Solution




```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Redundant Traversal - Solution



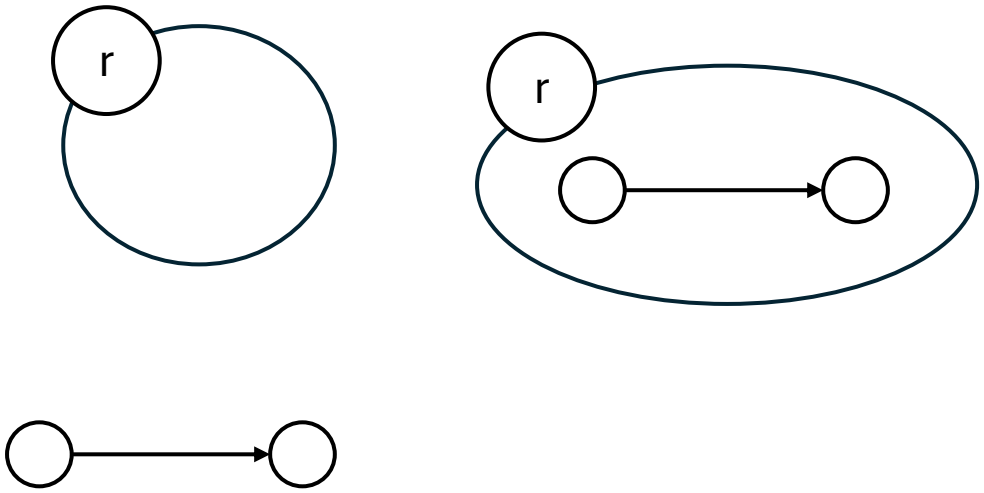

```
arr = np.array([A(), …], dtype=object)
view = arr[2:4]
r.view = view
```

Evaluation

Evaluation

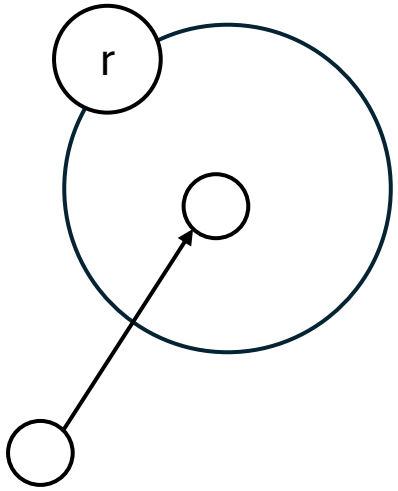
- Original Functionality
- LRC
- Objects Location

Scenario 1: Both are in the same location.



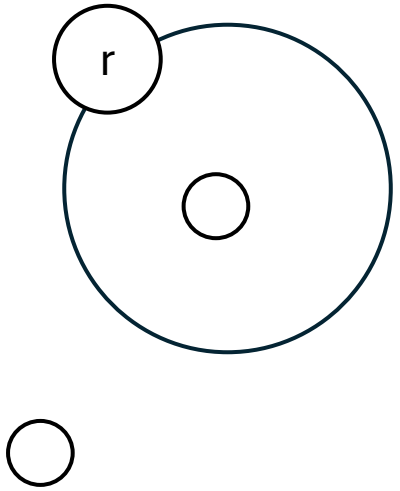
- LRC remains the same.
- Objects are still in local.

Scenario 2: Local Source, Regional Target (Adding Reference)



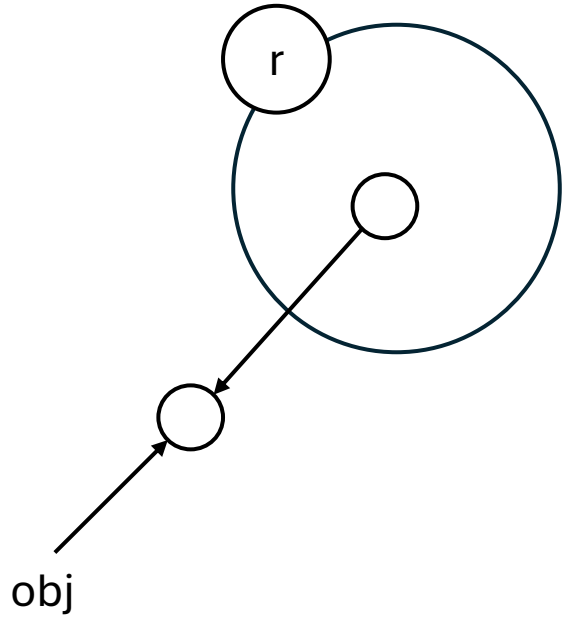
- LRC is increased.
- Objects are still in their location.

Scenario 3: Local Source, Regional Target (Removing Reference)



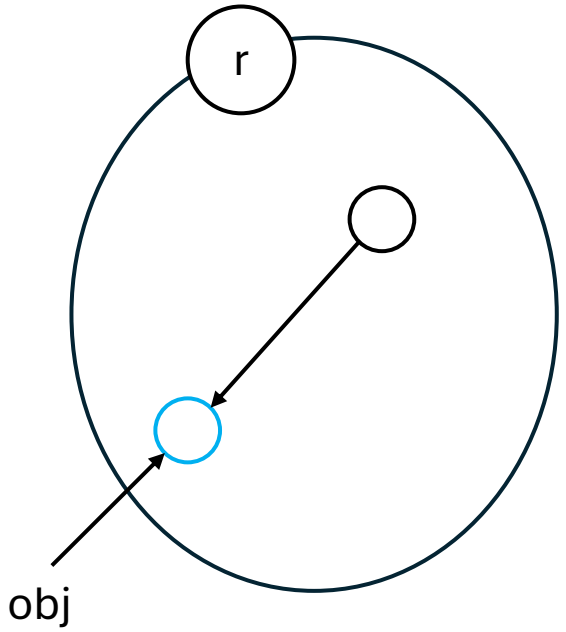
- LRC is decreased.
- Objects are still in their location.

Scenario 4: Regional Source, Local Target (Ephemeral Move)



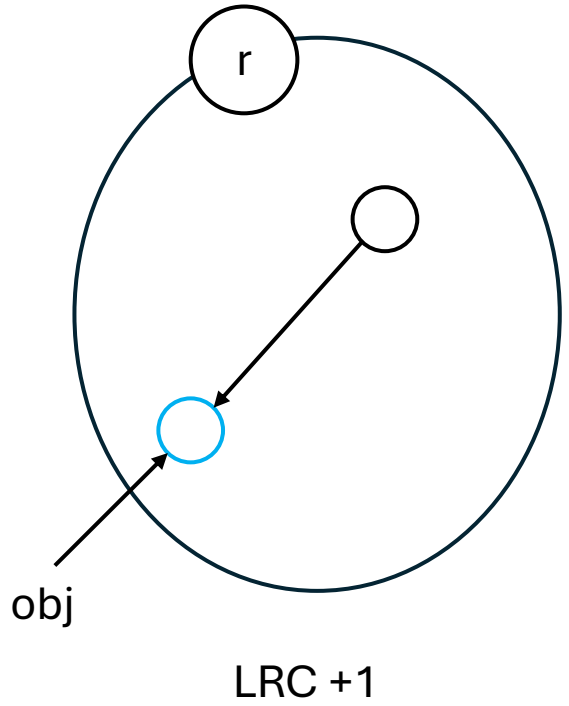
- Local Target is moved into the region.
- LRC has to reflect this change.

Scenario 4: Regional Source, Local Target (Ephemeral Move)



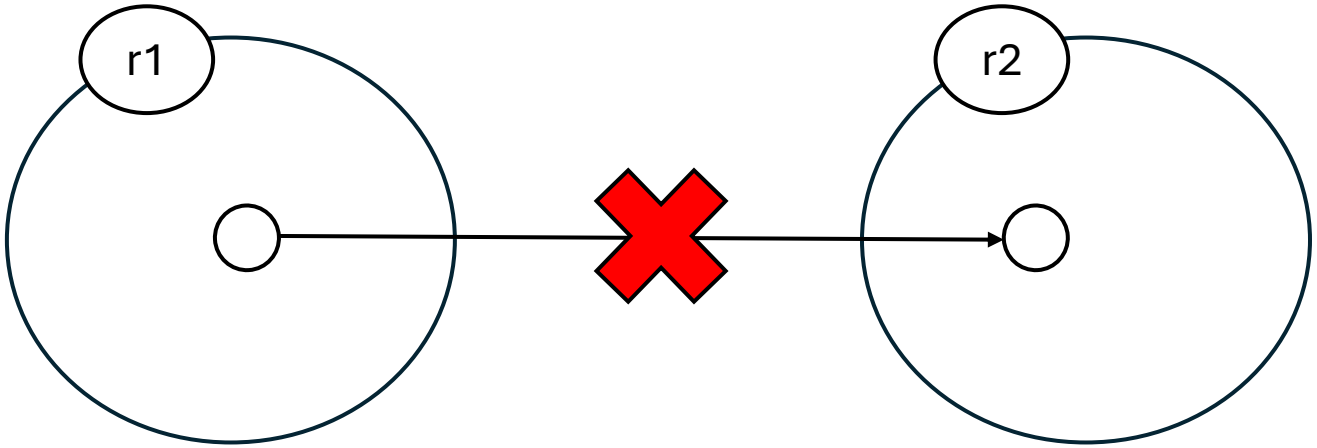
- Local Target is moved into the region.
- LRC has to reflect this change.

Scenario 4: Regional Source, Local Target (Ephemeral Move)



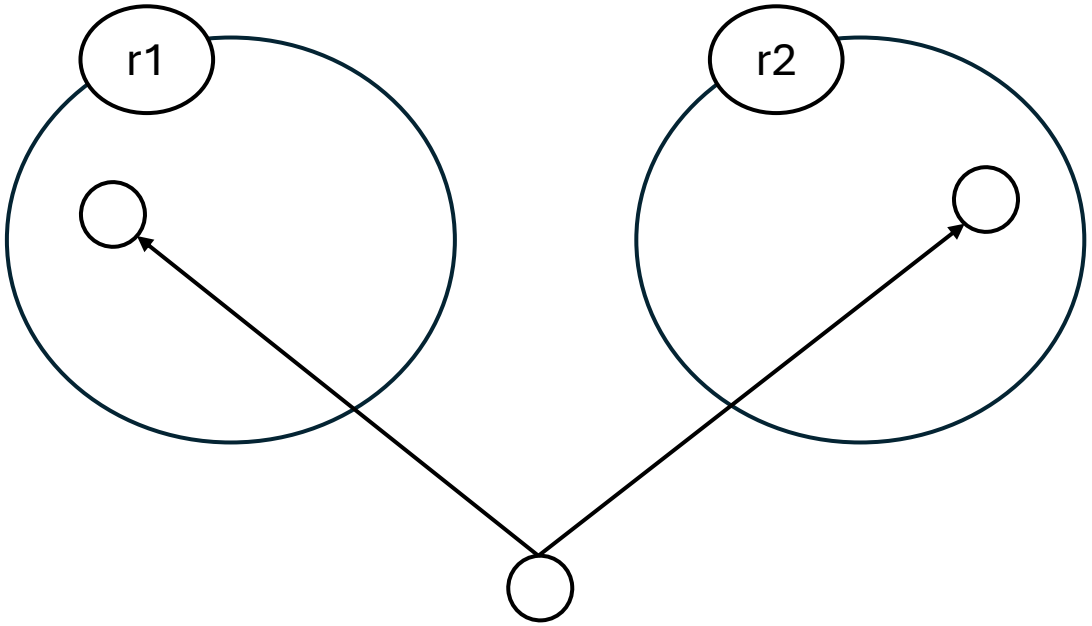
- Local Target is moved into the region.
- LRC has to reflect this change.

Scenario 5: Cross-Region References

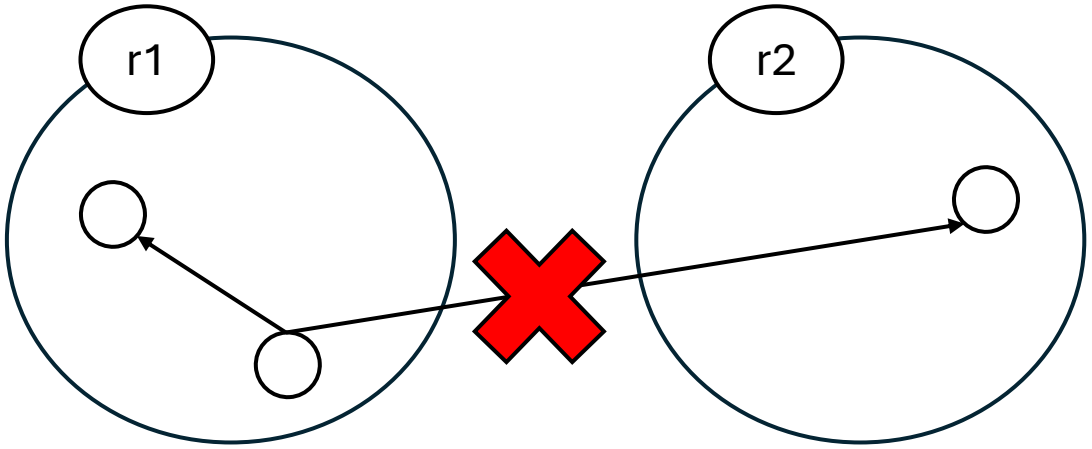


- LRC of both regions are not changed.
- Both objects are still in their same region.

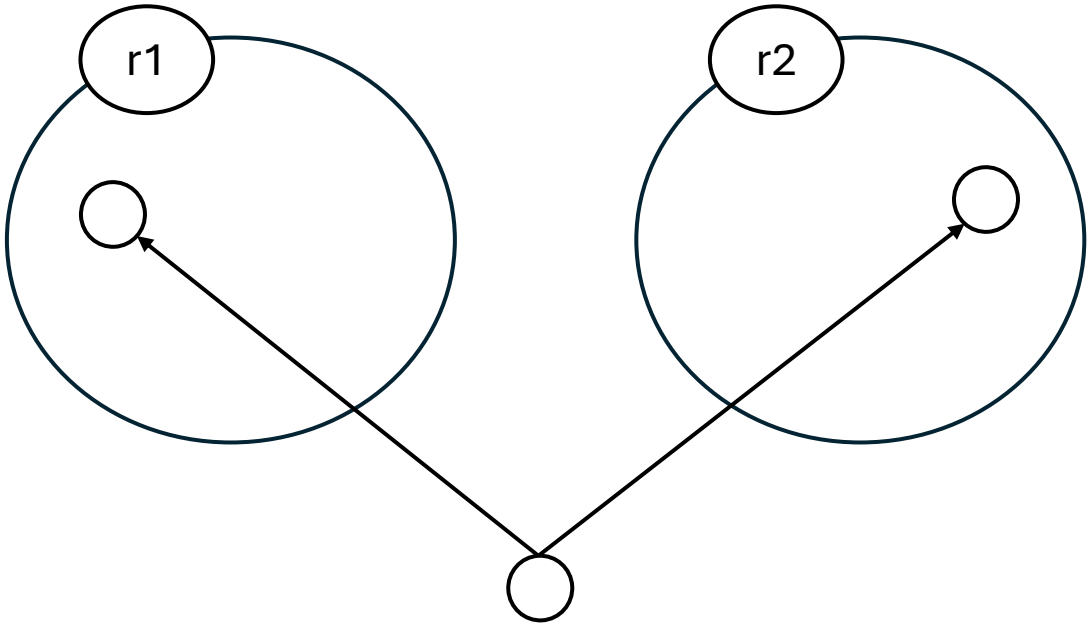
Scenario 5: Cross-Region References



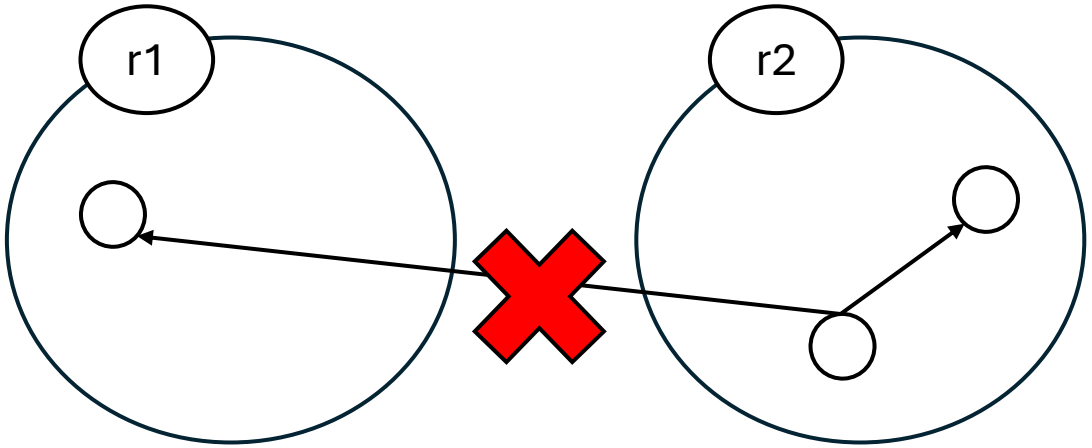
Scenario 5: Cross-Region References



Scenario 5: Cross-Region References

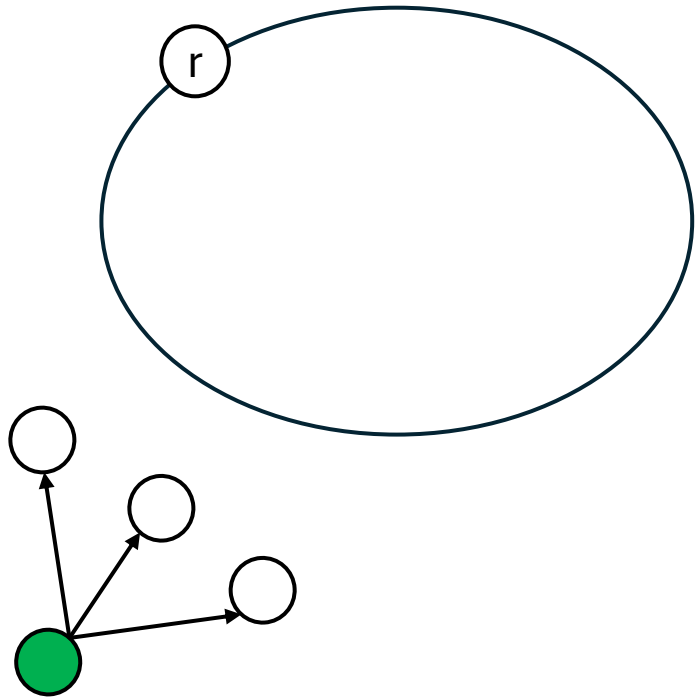


Scenario 5: Cross-Region References

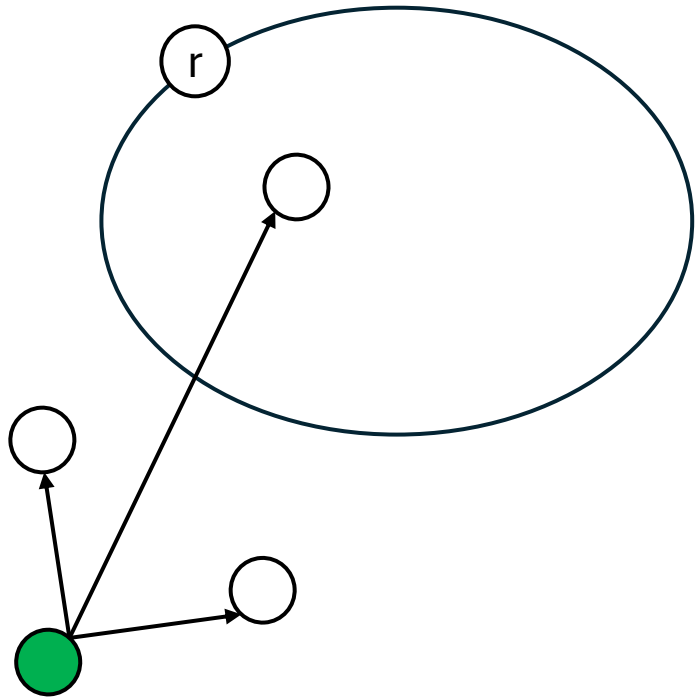


Exploding Cases - Set

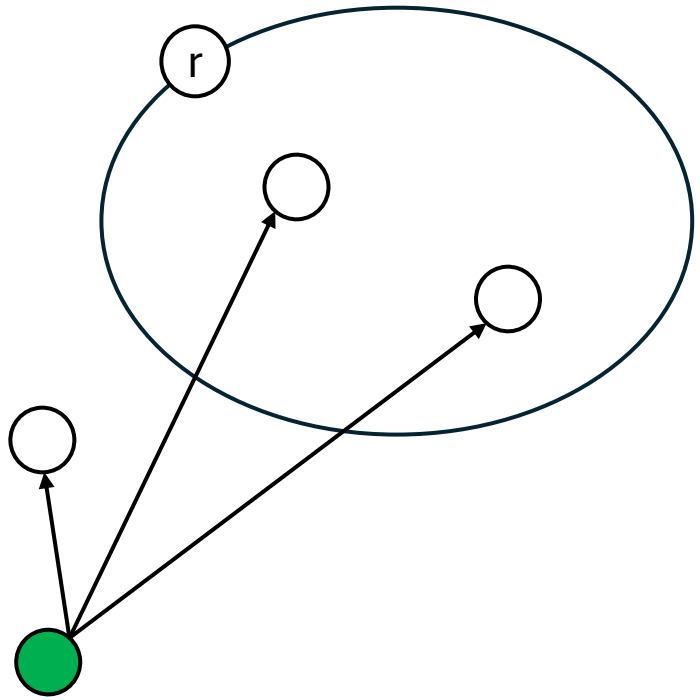
Testcases Exploding from only 5 Scenarios.



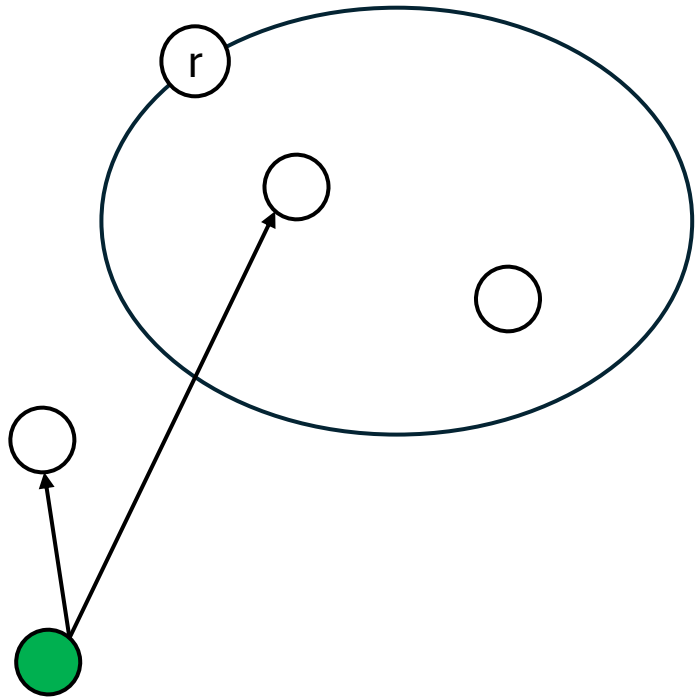
Testcases Exploding from only 5 Scenarios.



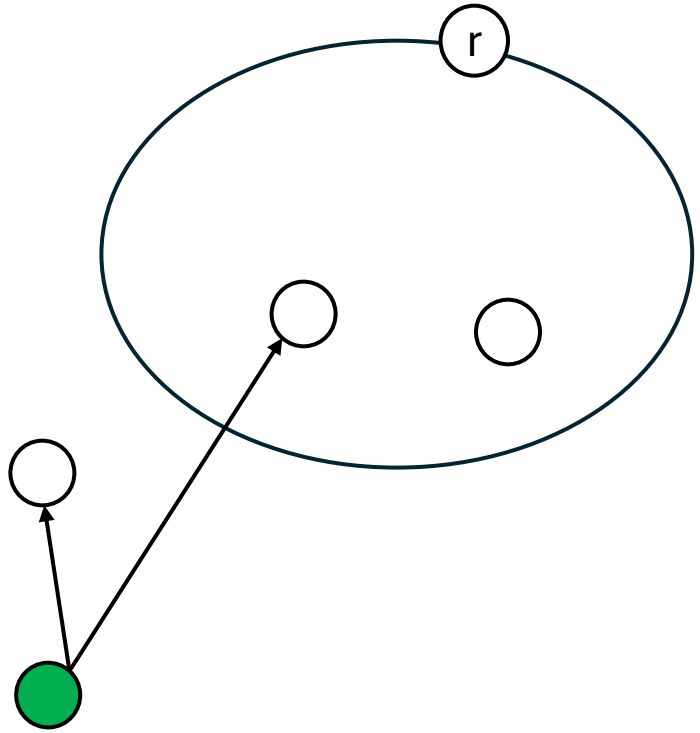
Testcases Exploding from only 5 Scenarios.



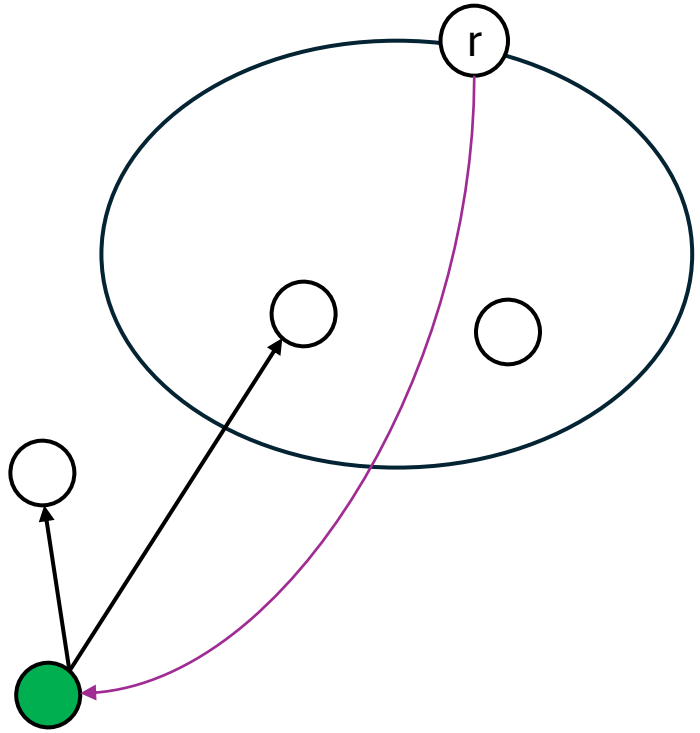
Testcases Exploding from only 5 Scenarios.



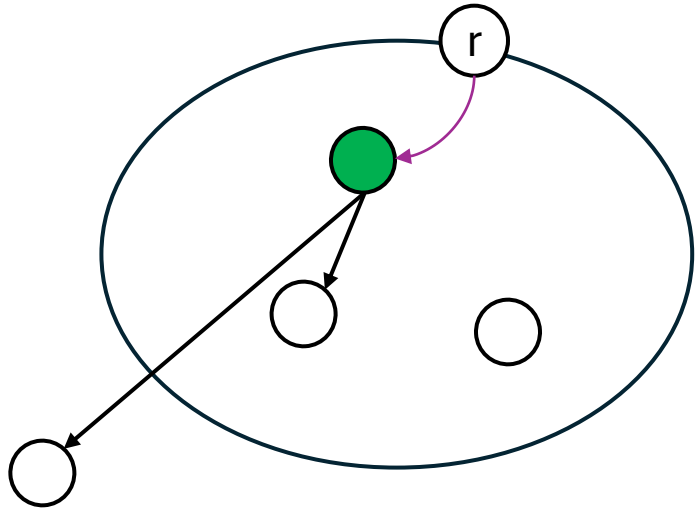
Testcases Exploding from only 5 Scenarios.



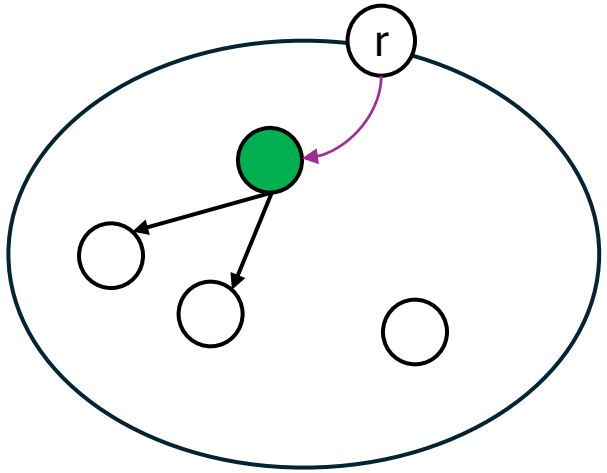
Testcases Exploding from only 5 Scenarios.



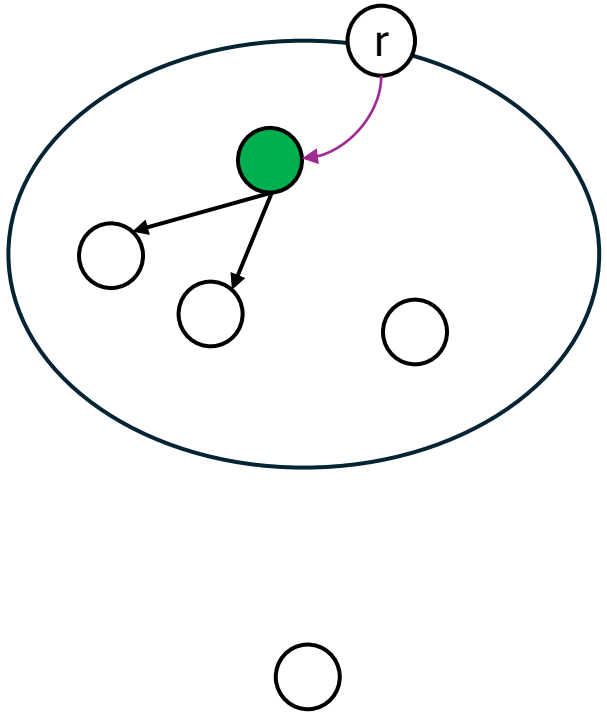
Testcases Exploding from only 5 Scenarios.



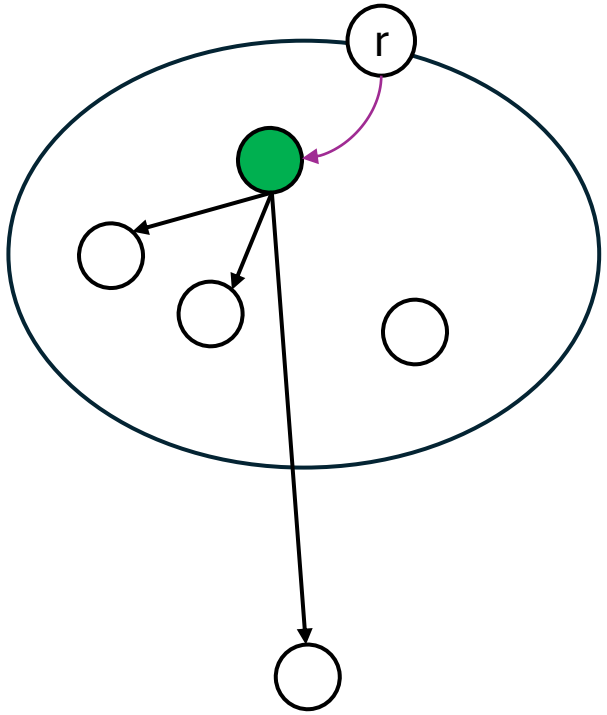
Testcases Exploding from only 5 Scenarios.



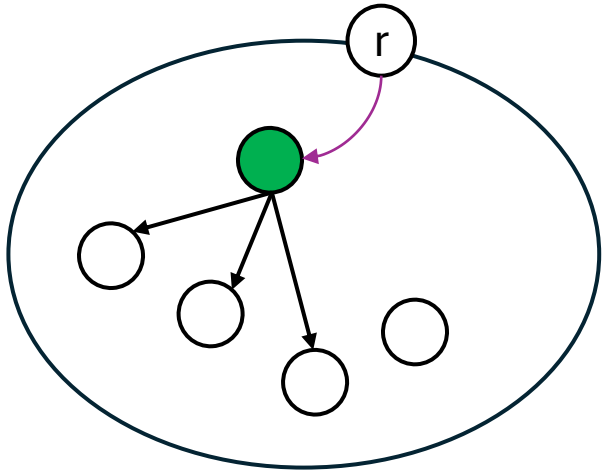
Testcases Exploding from only 5 Scenarios.



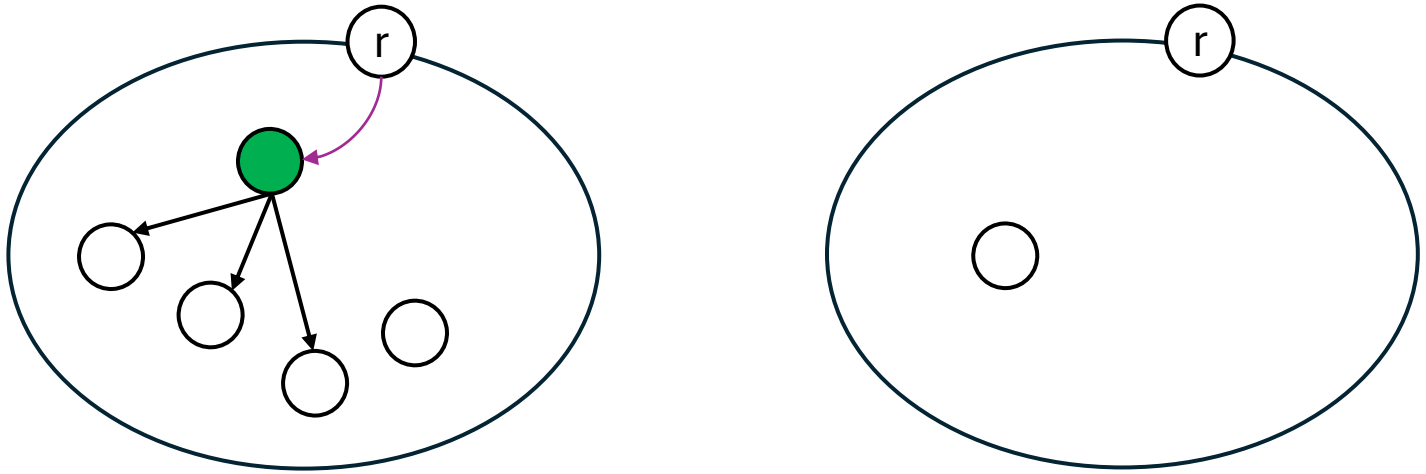
Testcases Exploding from only 5 Scenarios.



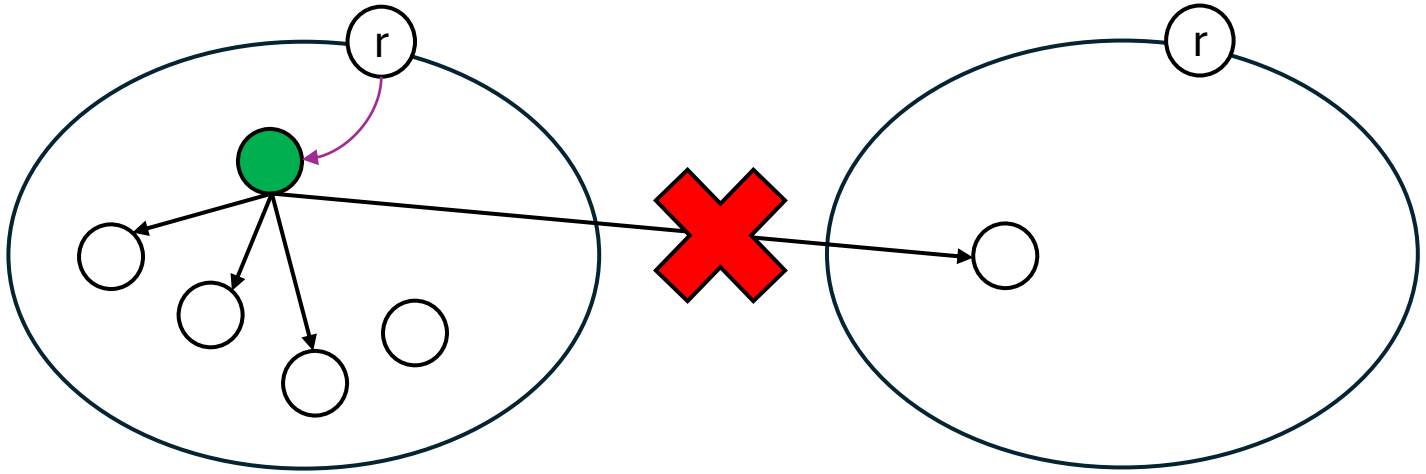
Testcases Exploding from only 5 Scenarios.



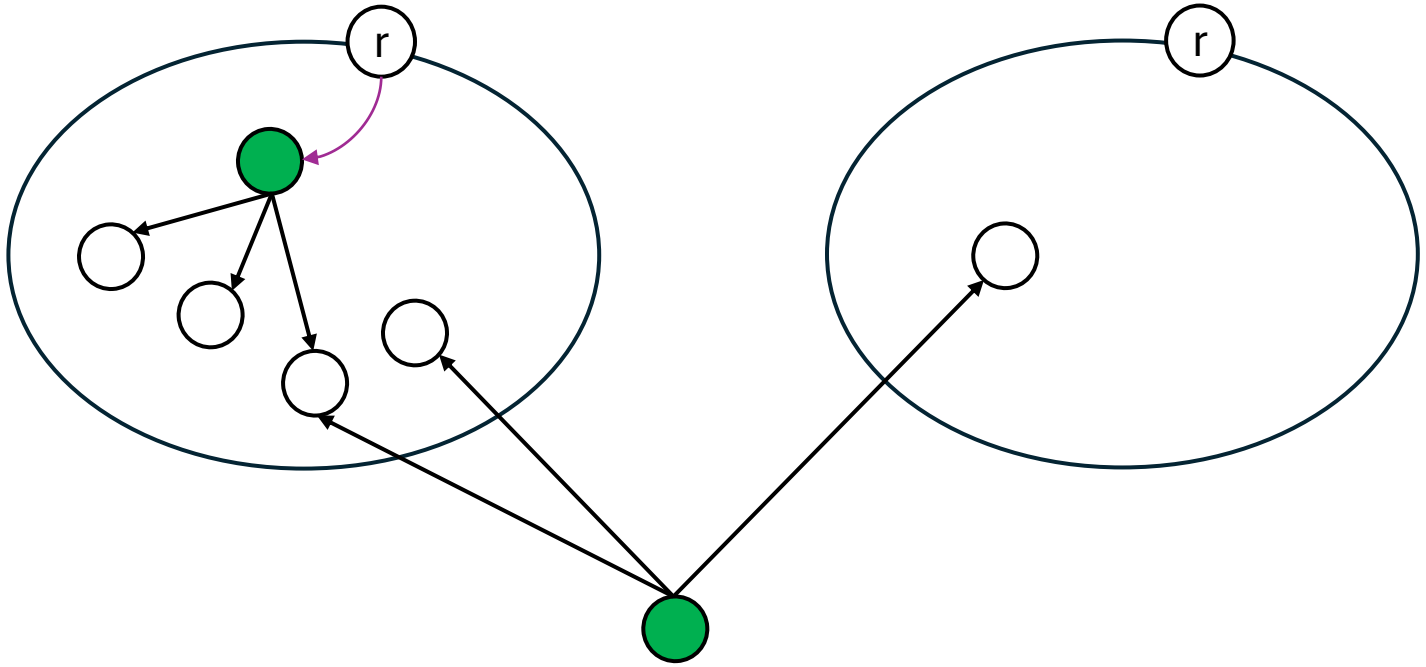
Testcases Exploding from only 5 Scenarios.



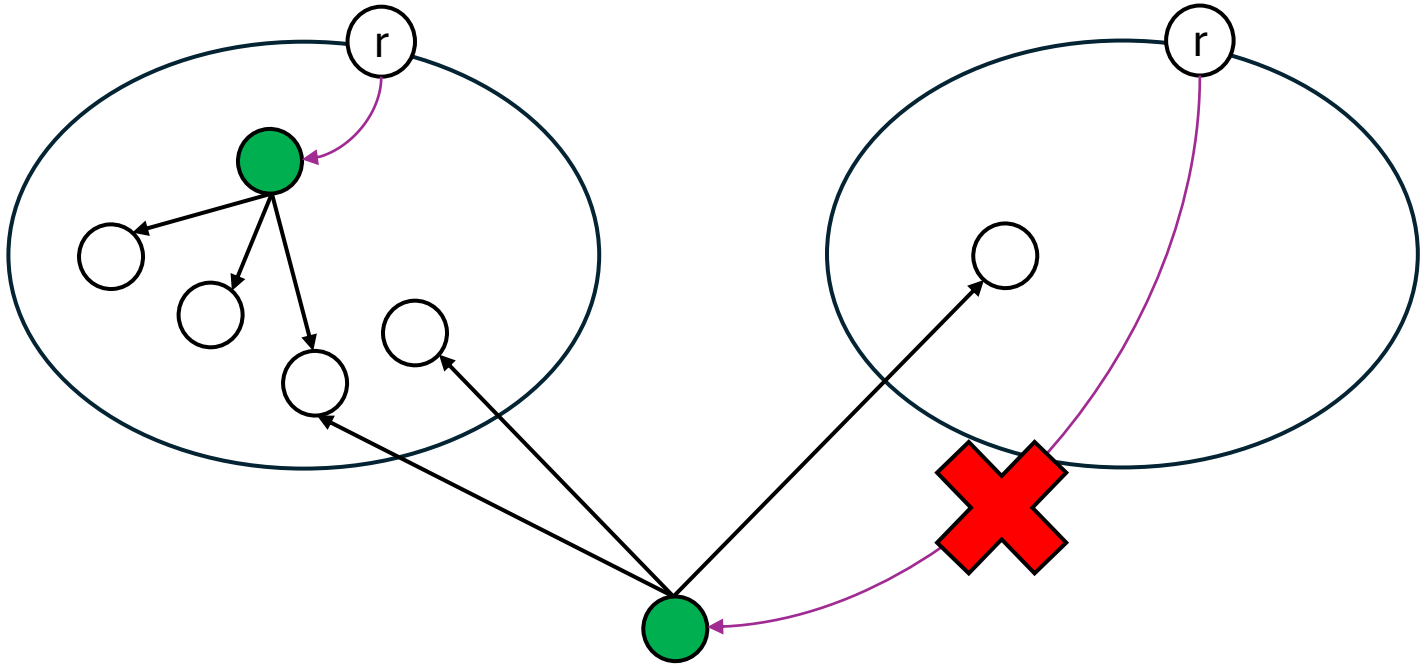
Testcases Exploding from only 5 Scenarios.



Testcases Exploding from only 5 Scenarios.



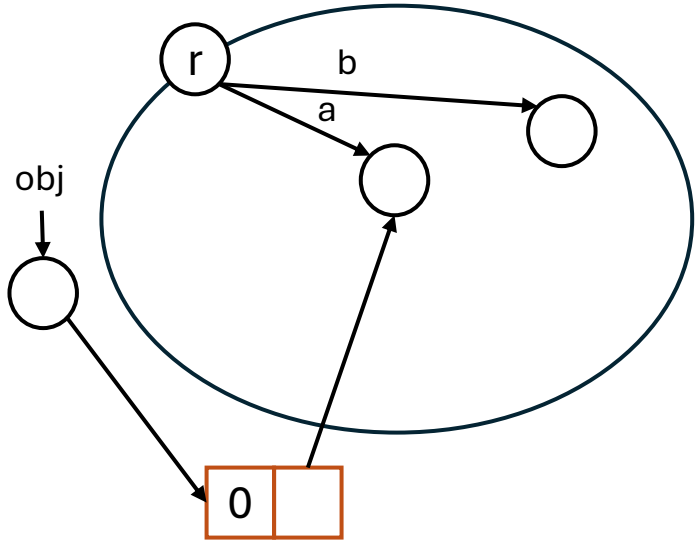
Testcases Exploding from only 5 Scenarios.



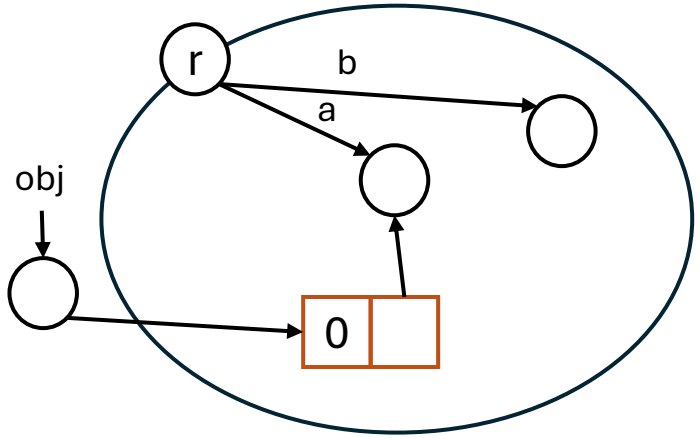
Let’s stop here.
You get the idea!

Exploding - Enumerate

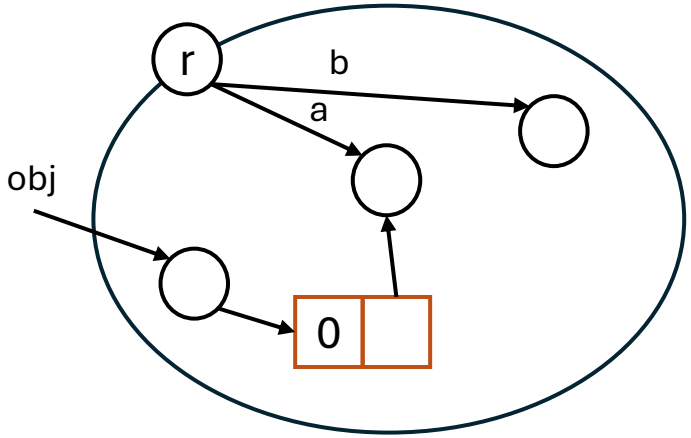
Testcases Exploding from only 5 Scenarios.



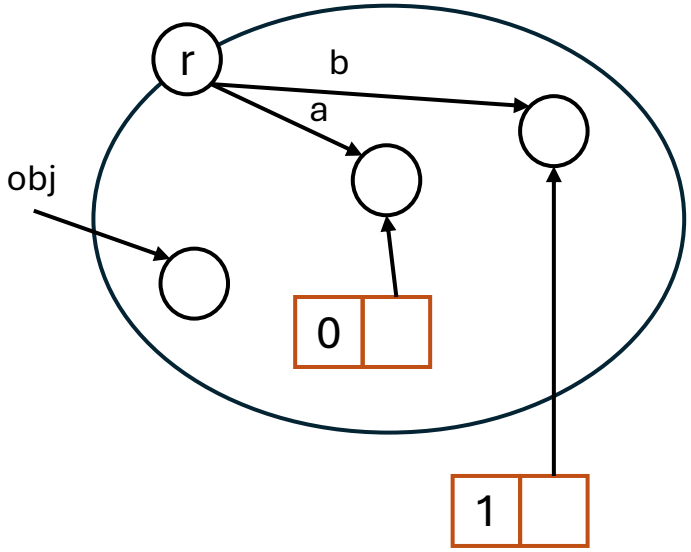
Testcases Exploding from only 5 Scenarios.



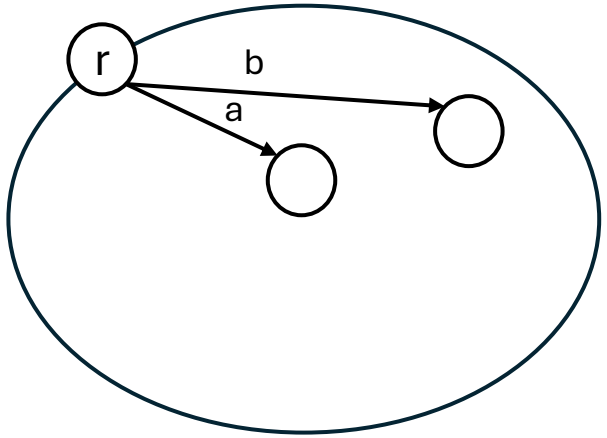
Testcases Exploding from only 5 Scenarios.



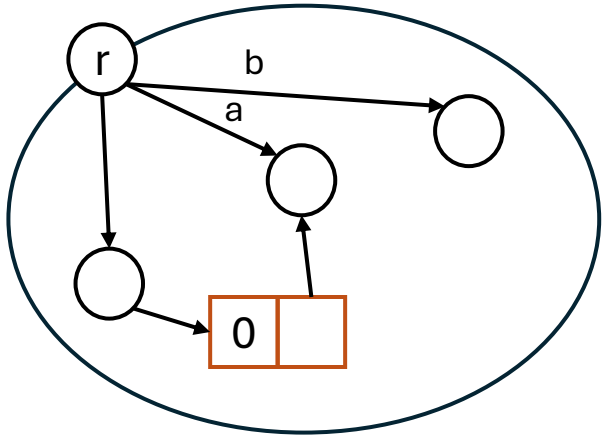
Testcases Exploding from only 5 Scenarios.



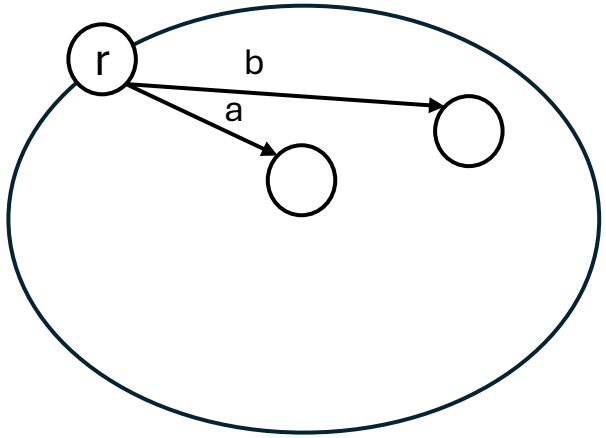
Testcases Exploding from only 5 Scenarios.

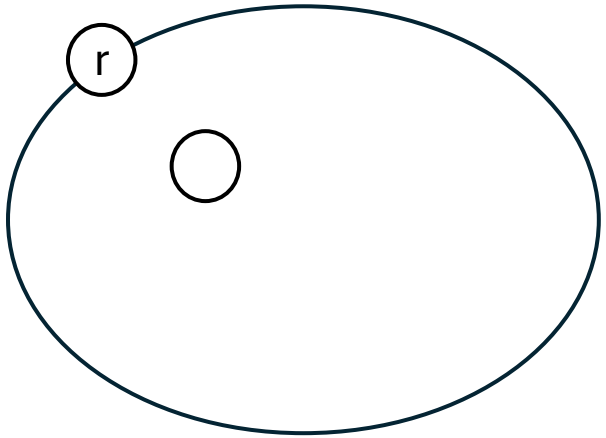


Testcases Exploding from only 5 Scenarios.

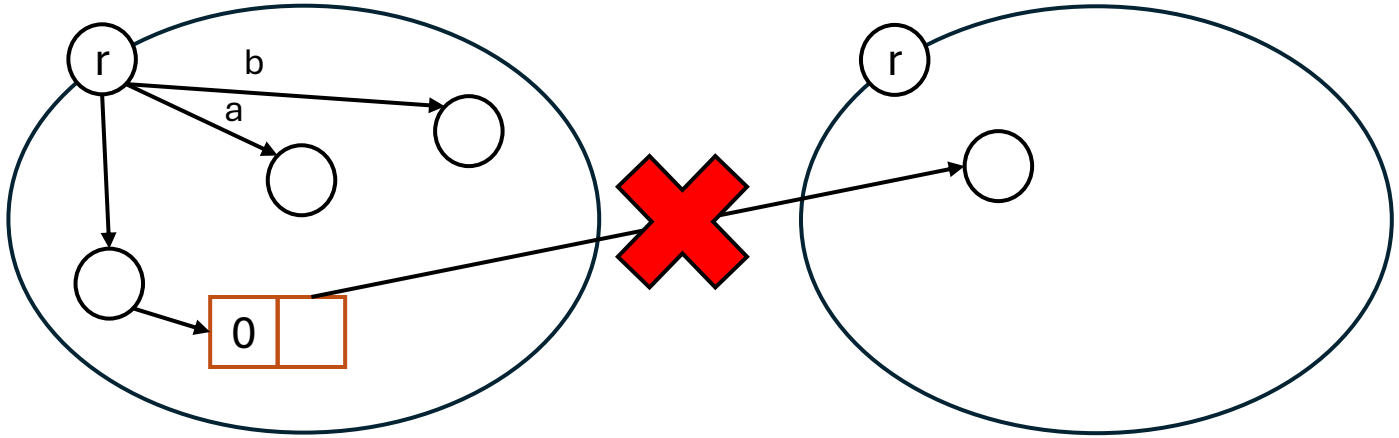


Testcases Exploding from only 5 Scenarios.

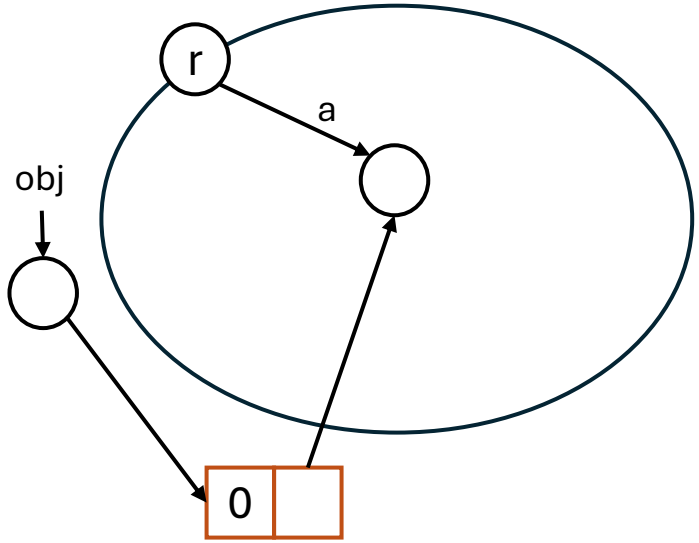


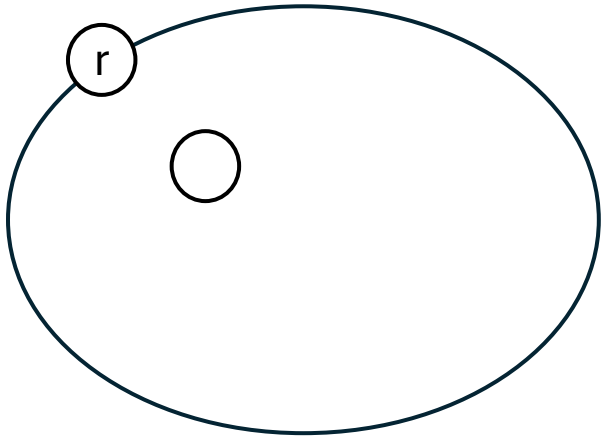


Testcases Exploding from only 5 Scenarios.

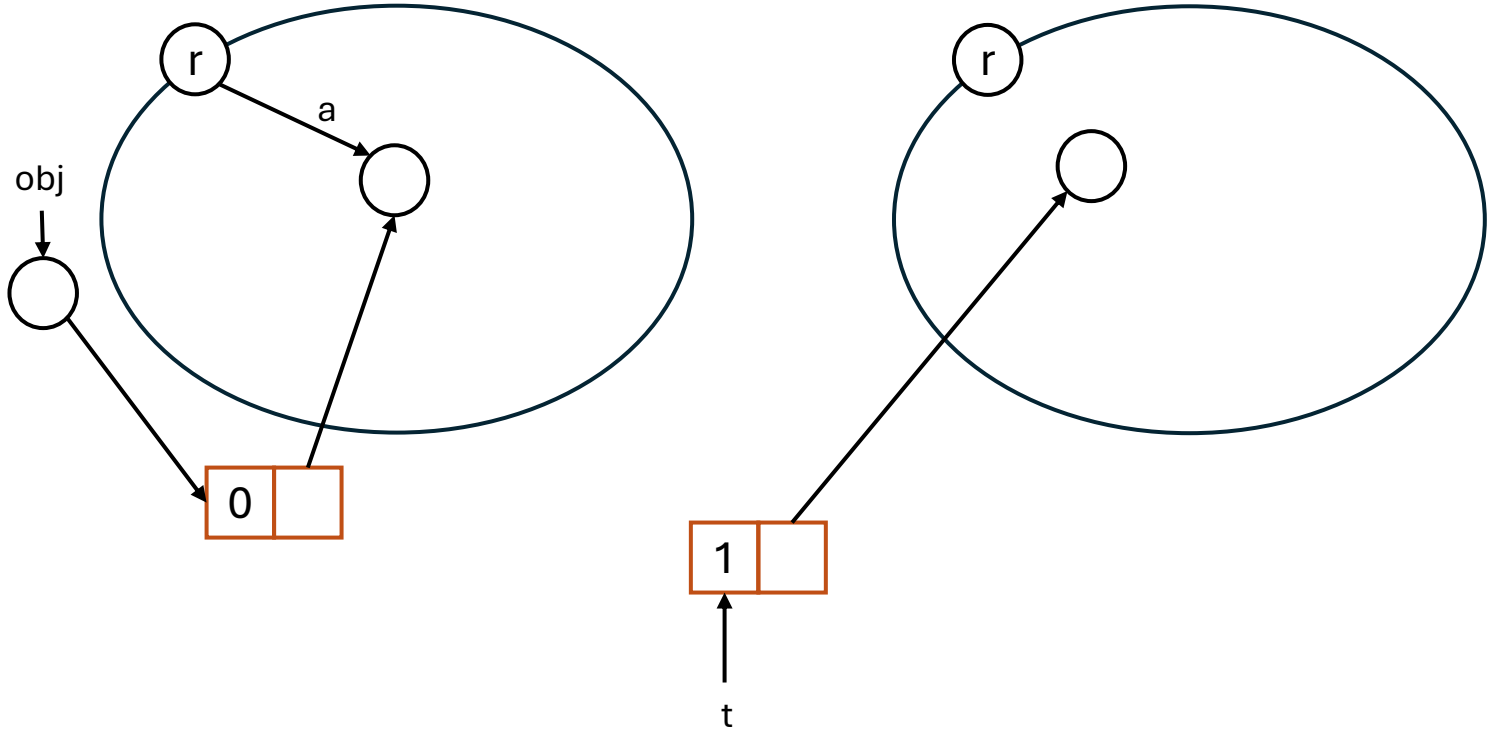


Testcases Exploding from only 5 Scenarios.

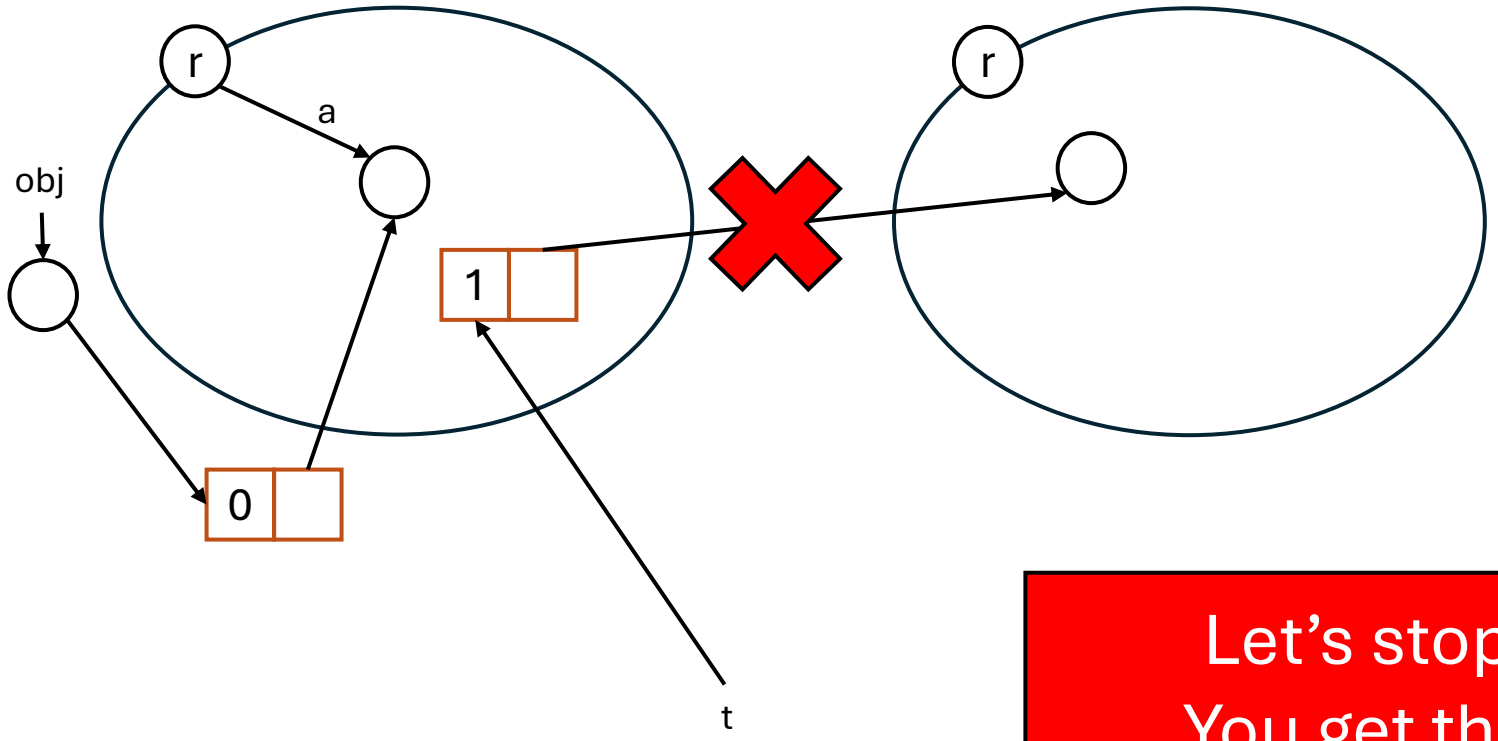




Testcases Exploding from only 5 Scenarios.



Testcases Exploding from only 5 Scenarios.



Let’s stop here.
You get the idea!

Limitations

- ABA problem for LRC
- Demonstrate the presence of bugs
  - But not the absence!
    - Another formal method is required
  - Unit-Test is sufficient for now!
- Only verify under sequential execution

Conclusion

- Patterns that can apply write barriers directly.
- Non-General Pattern in CPython Codebase.
- NumPy Migration
- Evaluation

Outlook

- Migrate more Python Data Structure
- Migrate more NumPy operations
  - From 1 to N-dimensional Array
  - More Slicing Techniques
    - Fancy Indexing (a[np.array([0, 2, 4])])
    - HAS_NEWAXIS (a[np.newaxis, :])
    - HAS_SCALAR_ARRAY (a[np.array(1)])
    - …
  - Change `mem_handler` and `descr` assumption
- Formal Verification for Testing (Maybe a good Master Thesis Topic)

Done. Thank You!

Do I?