The SciPy Stack

A Python ecosystem for Scientific Computation

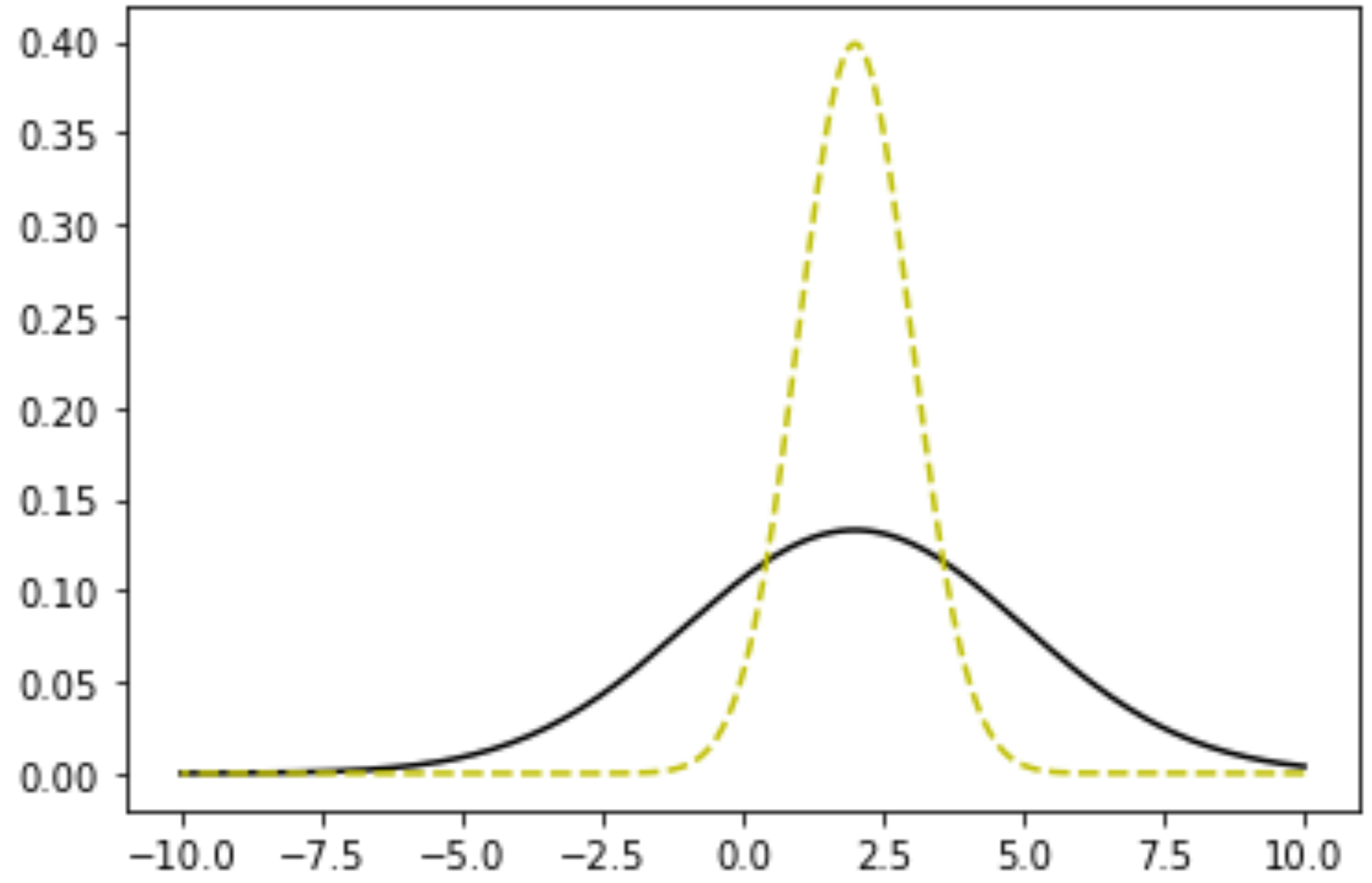
Michael Colaresi

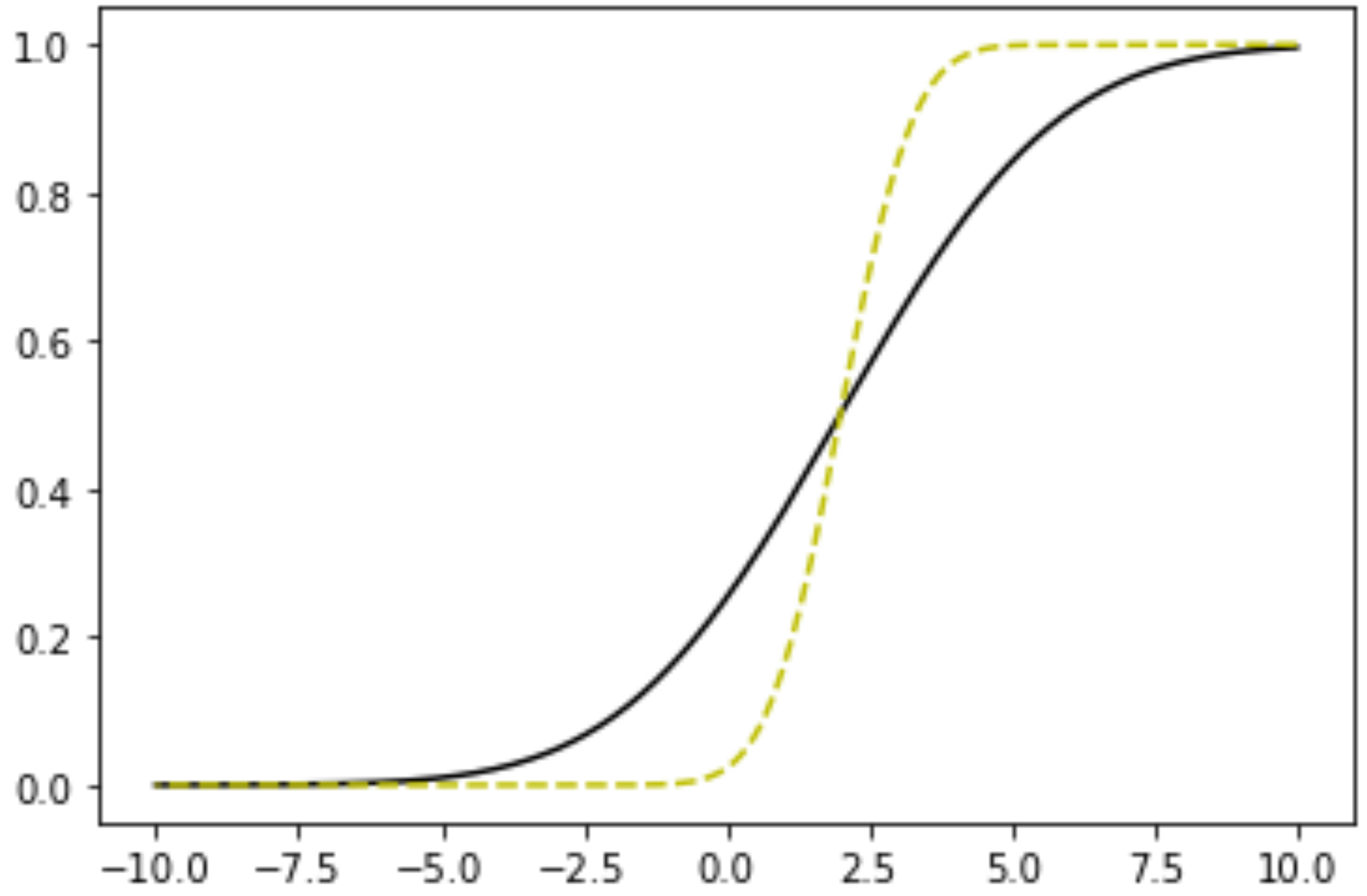
SciPy

- Is both a **library** containing numerical computing tools and distribution
- And a **collection** of related tools that build on each other
    - NumPy
    - matplotlib
    - pandas
    - sklearn

There are others, like SymPy for symbolic math (it can do your algebra homework for you)

SciPy
Distributions

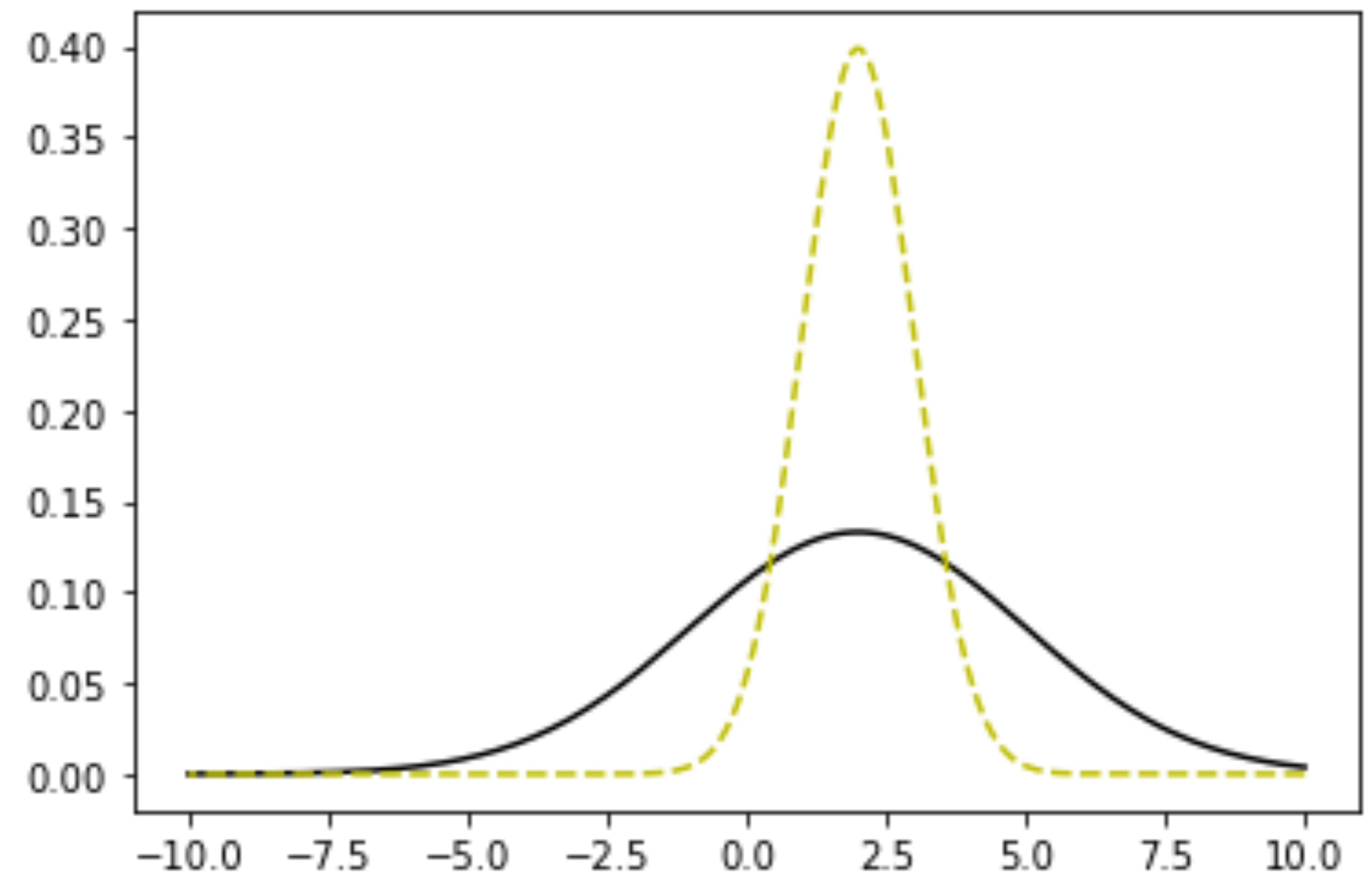


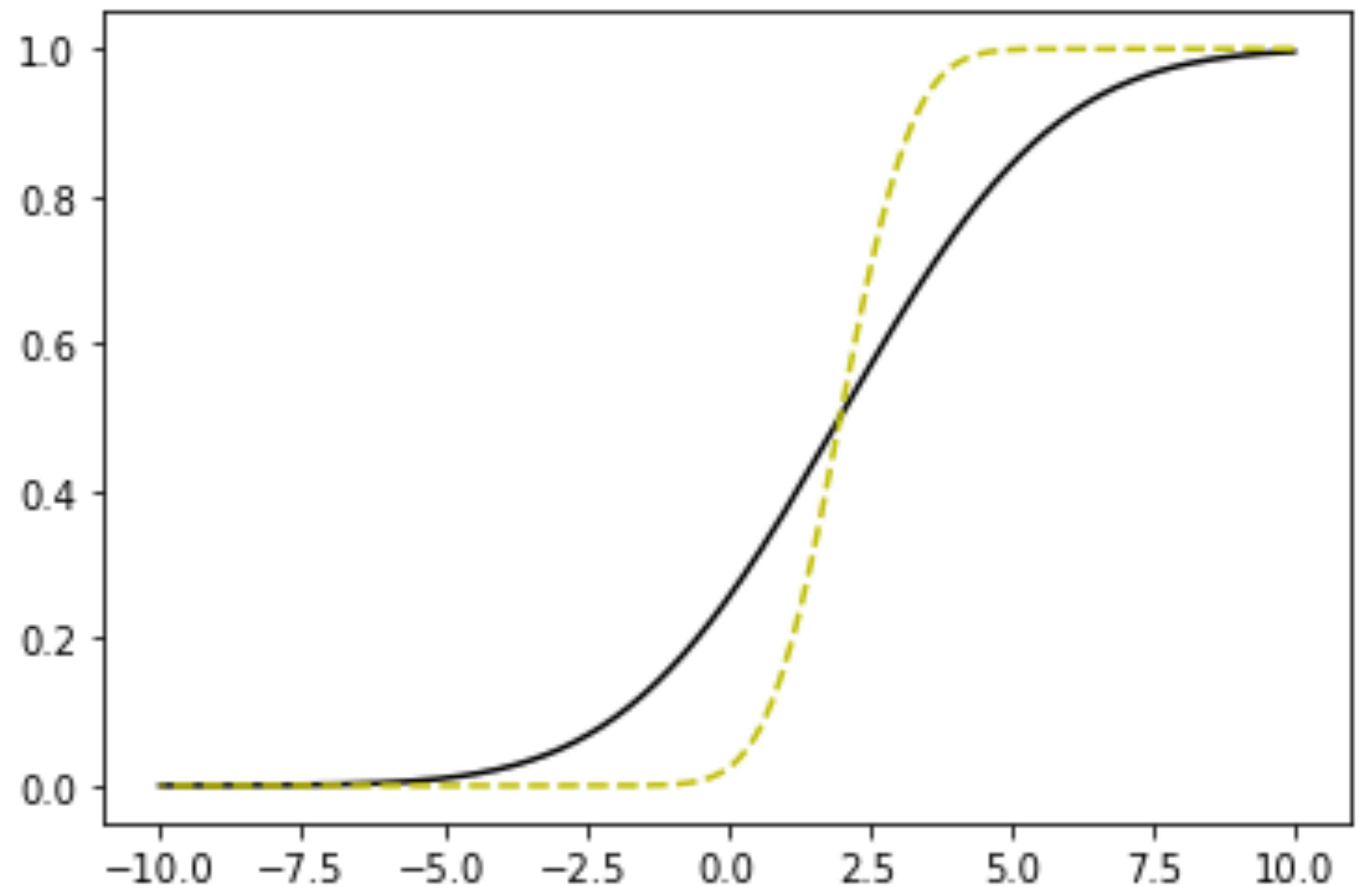


- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

SciPy

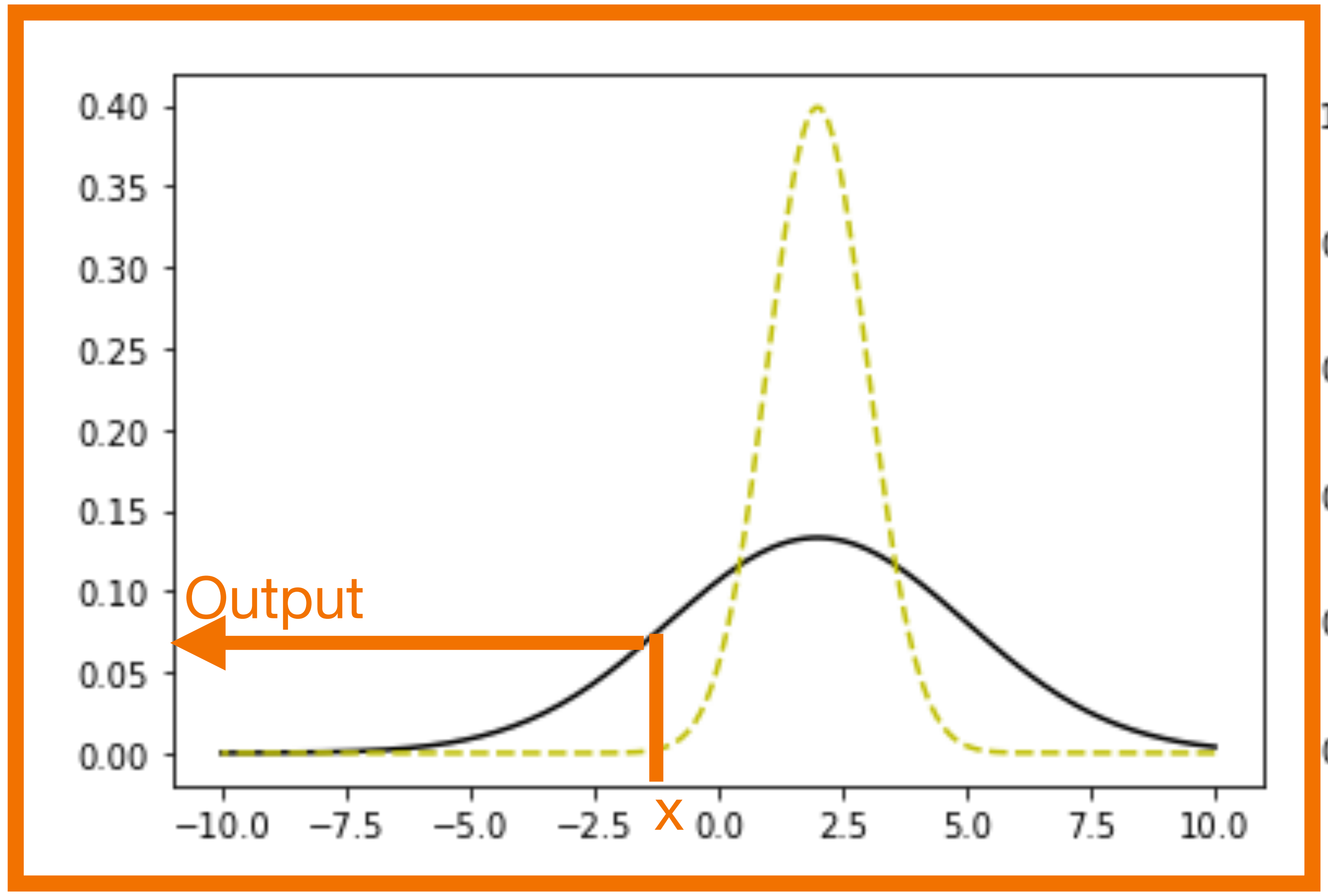
Distributions

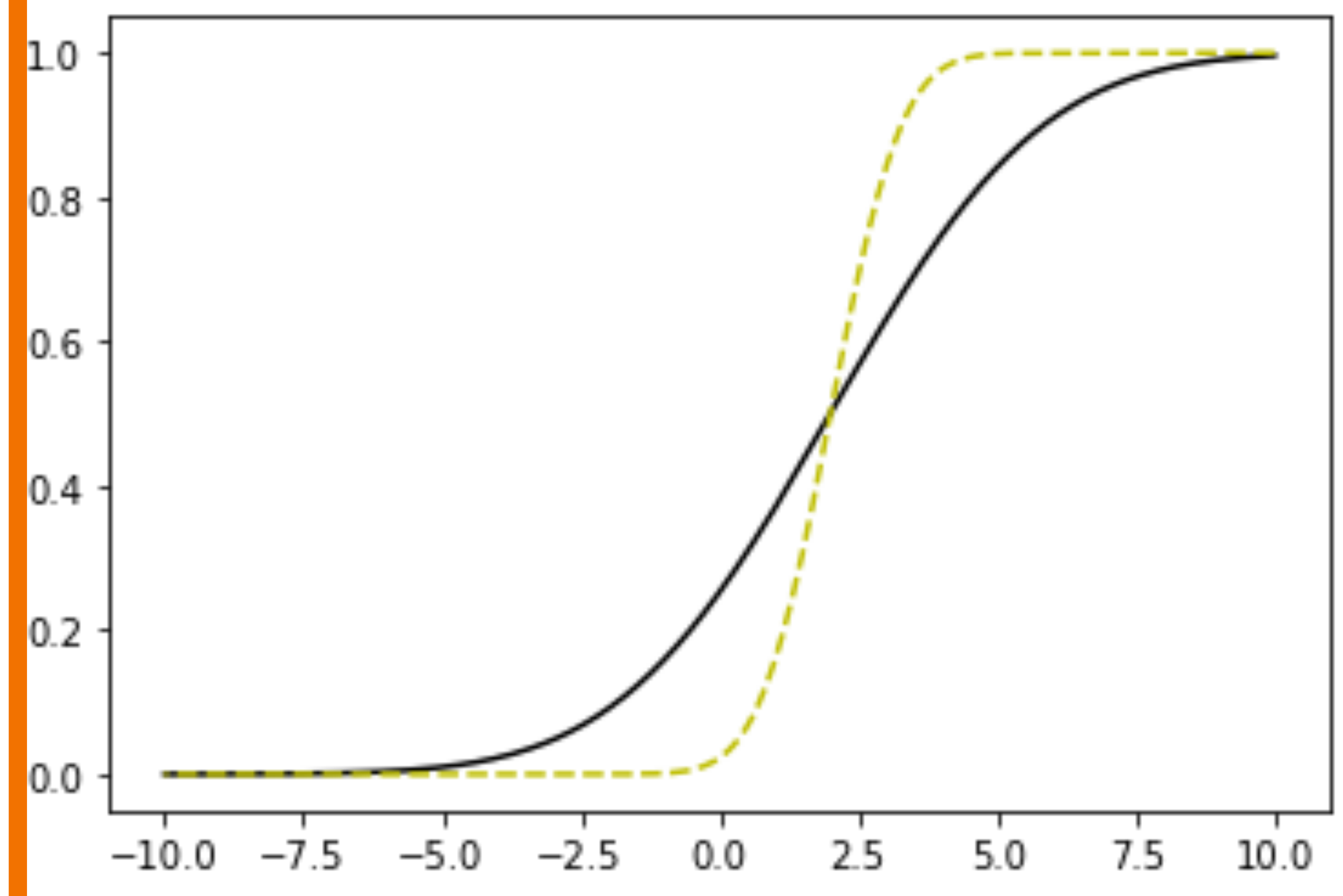




- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

SciPy
Distributions

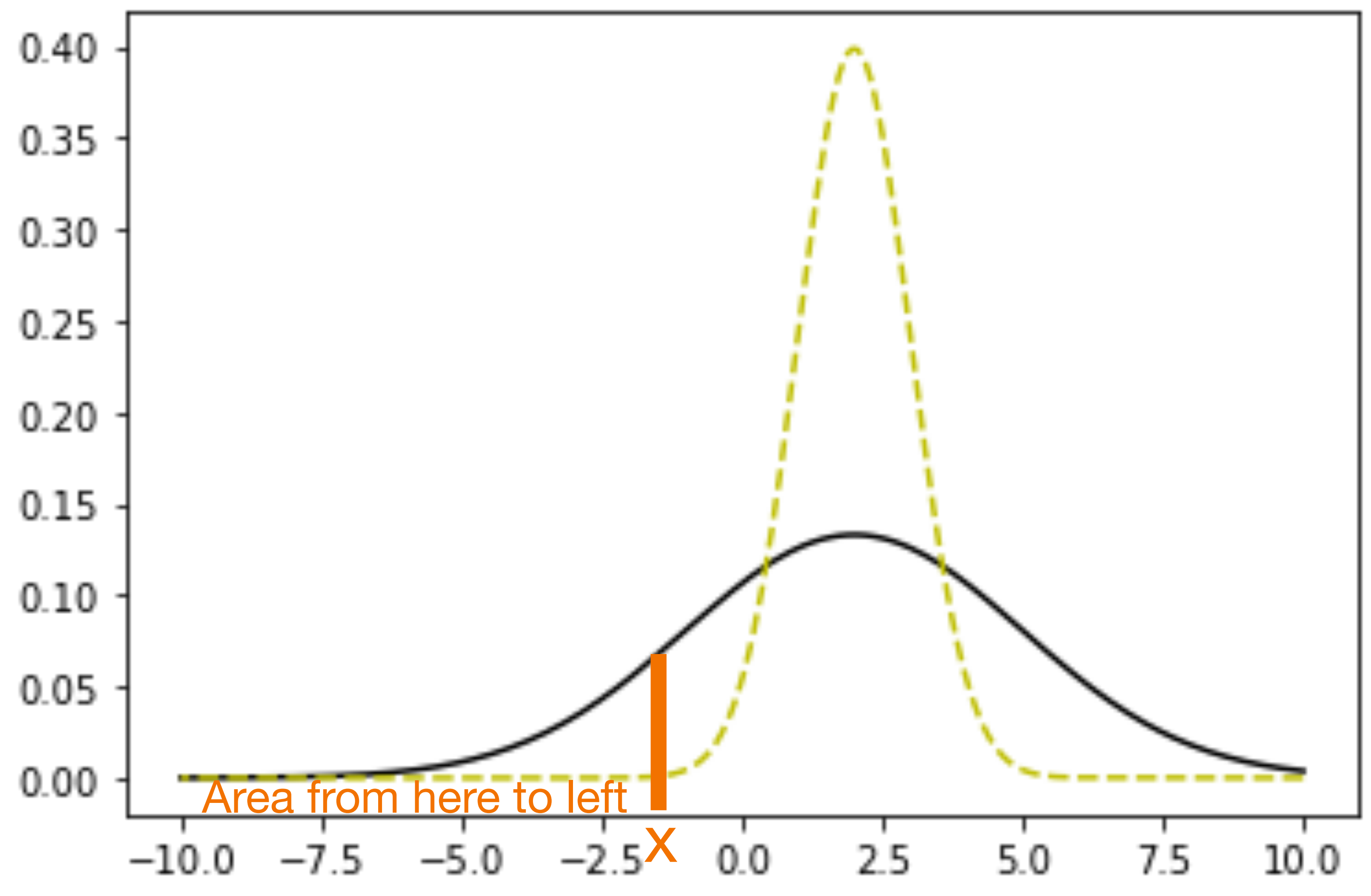


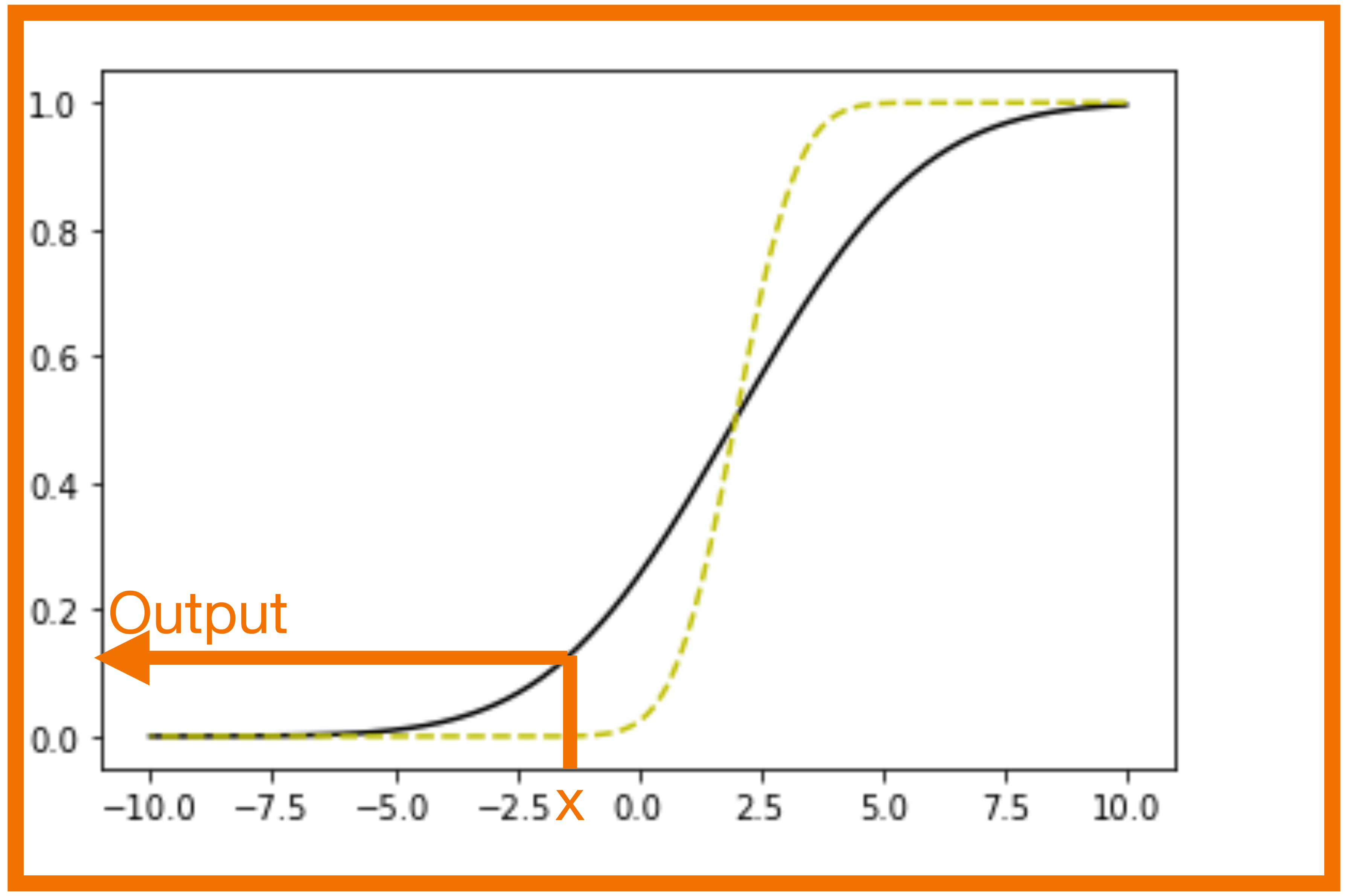


- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

SciPy

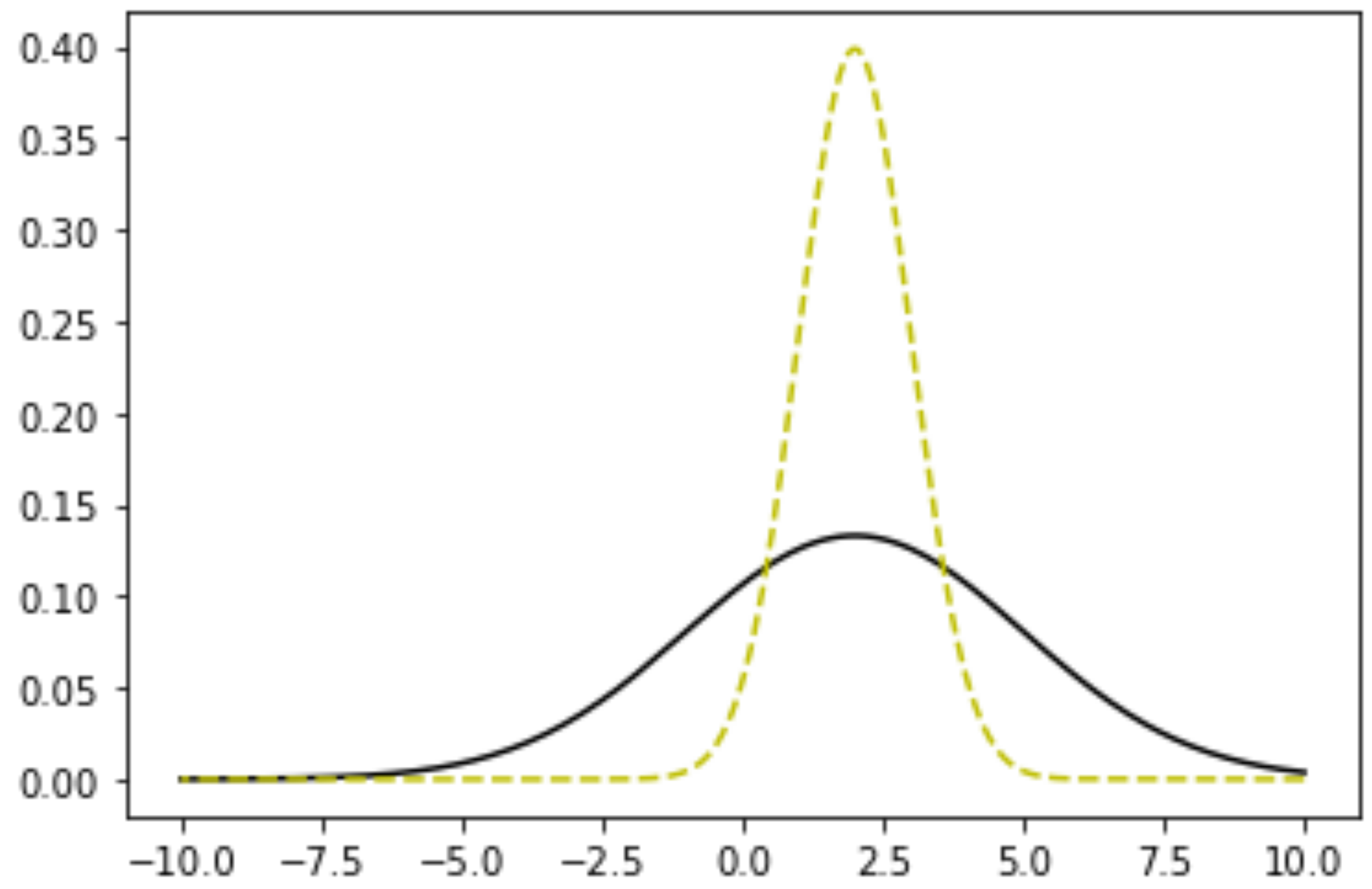
Distributions

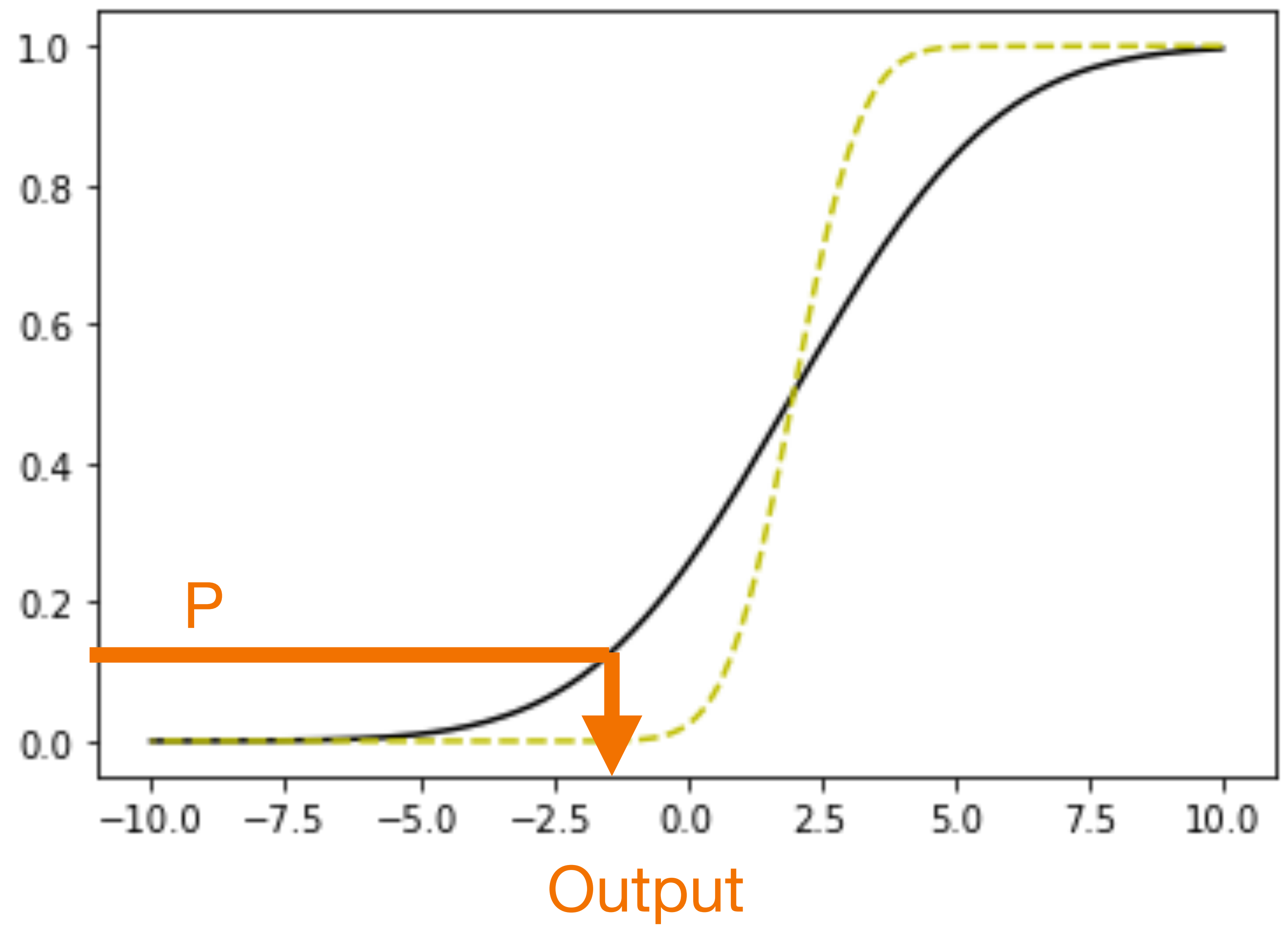




- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

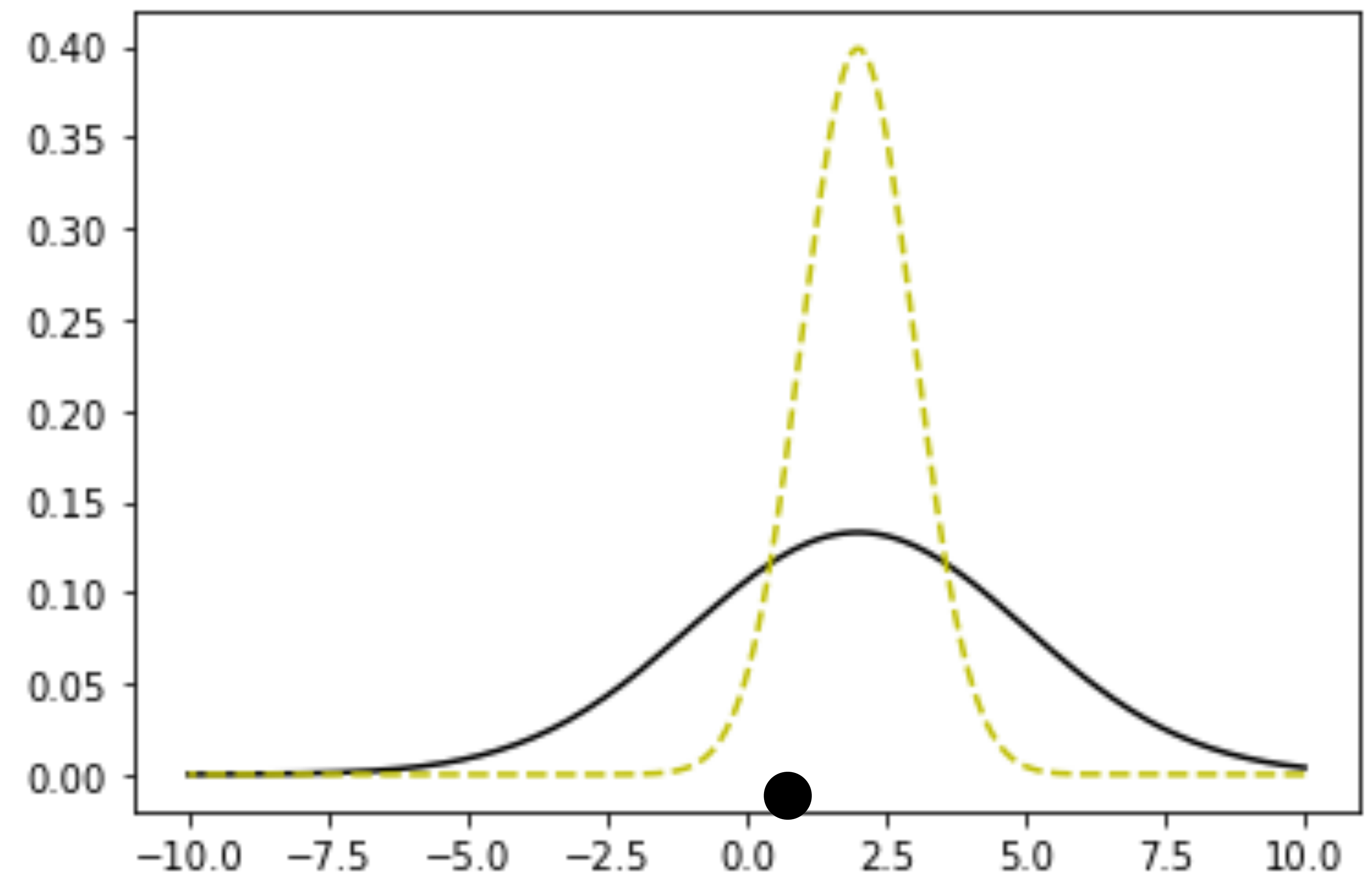
SciPy
Distributions

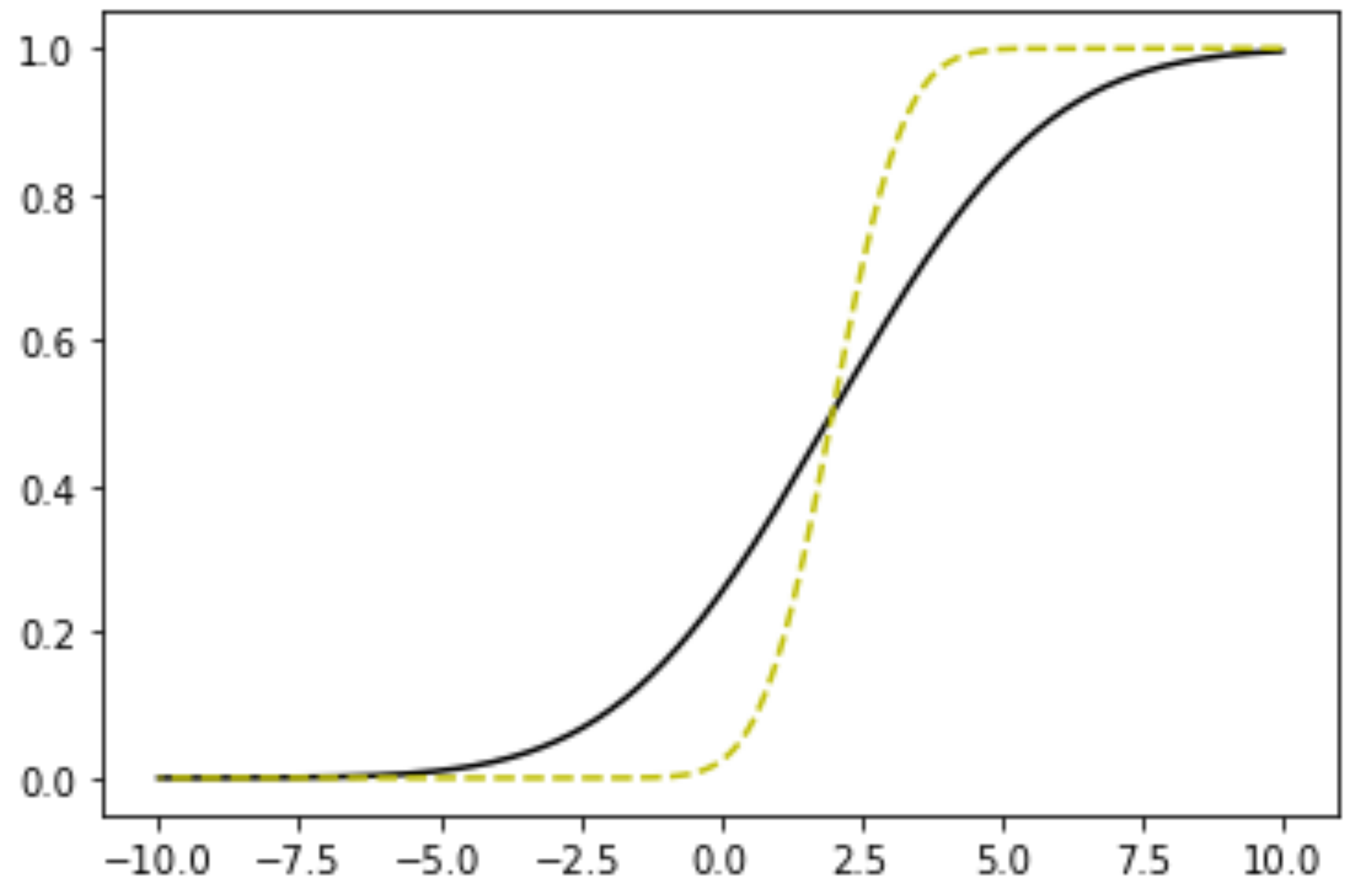




- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

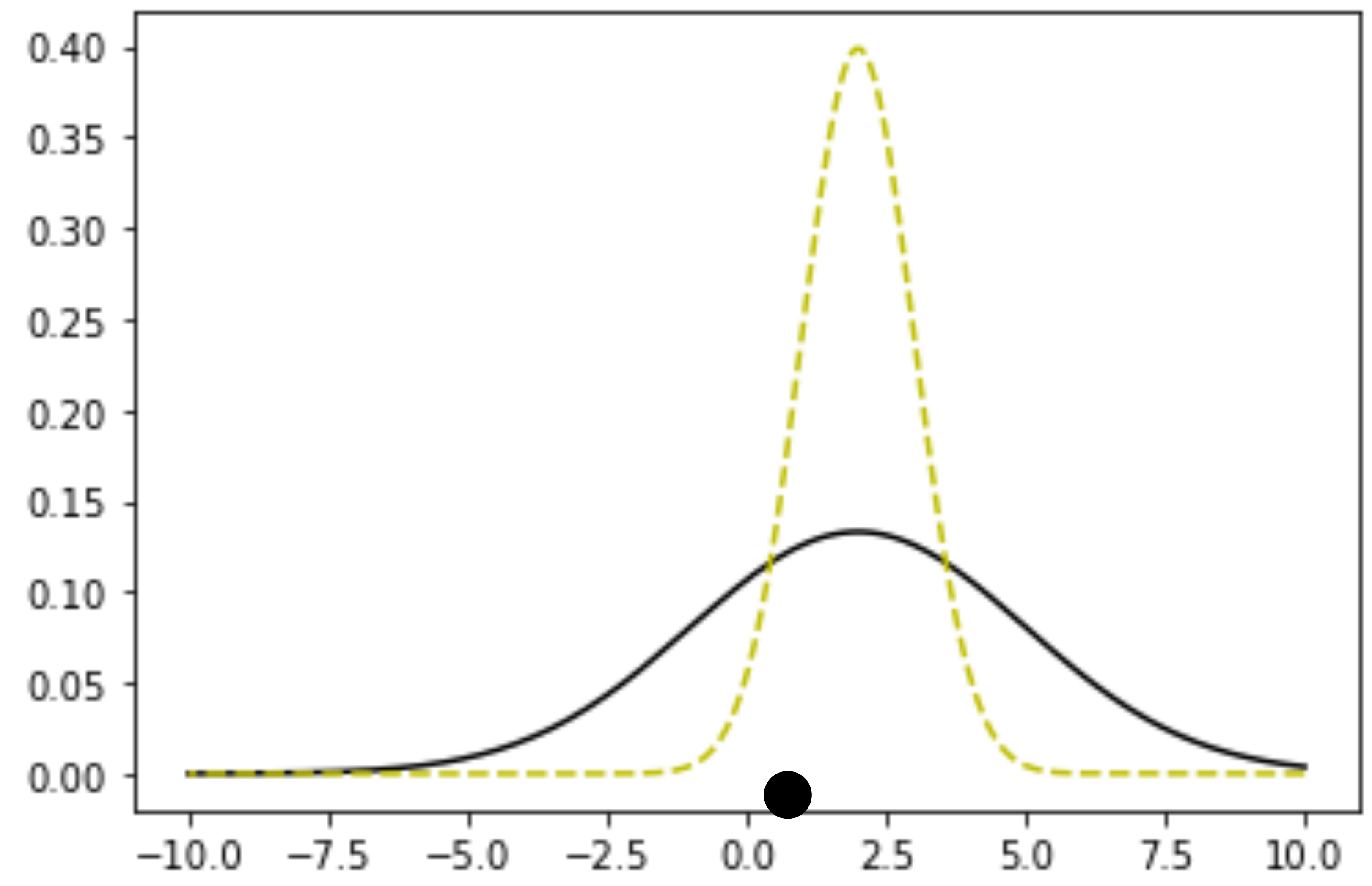
SciPy
Distributions

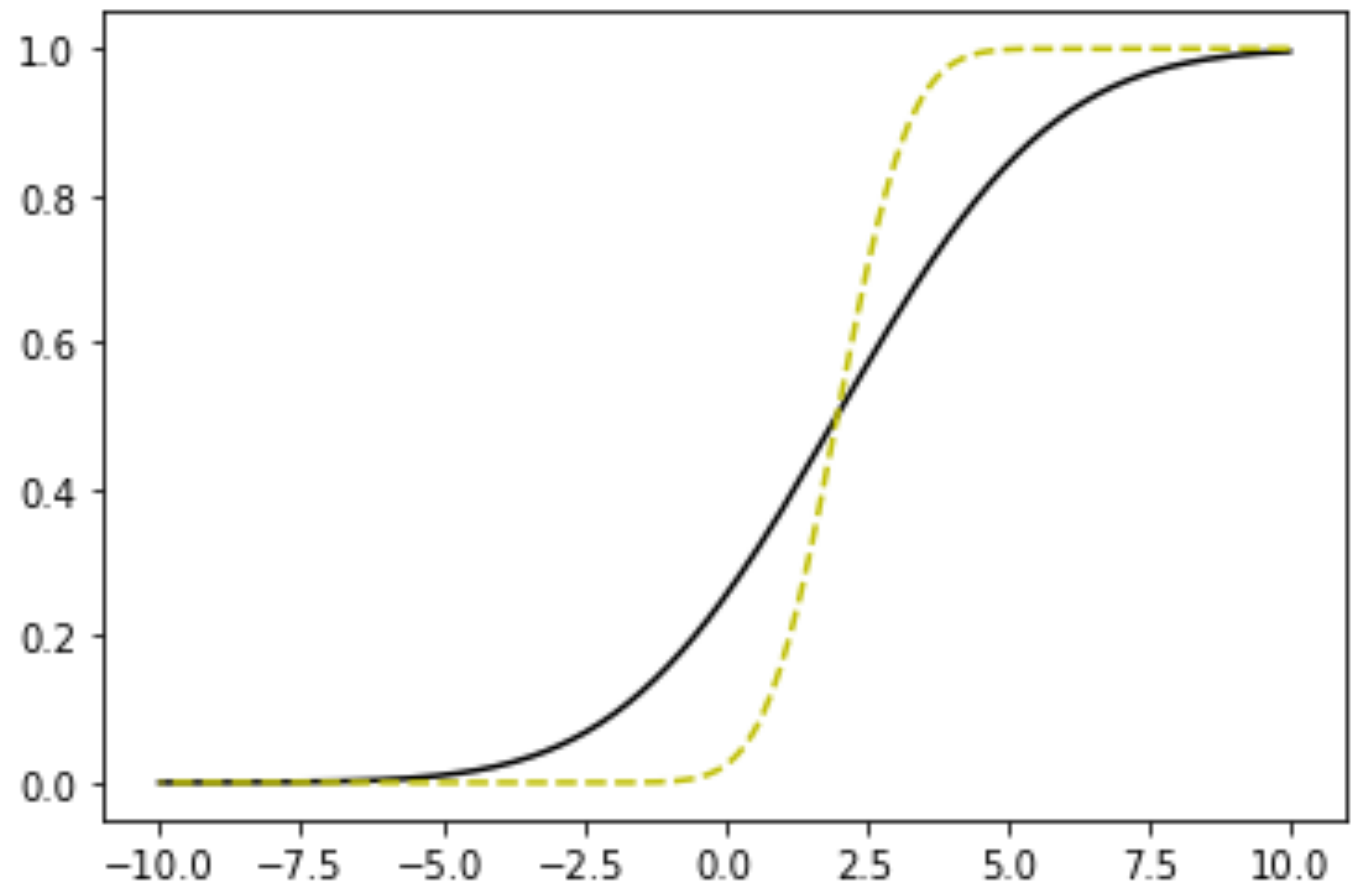




- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

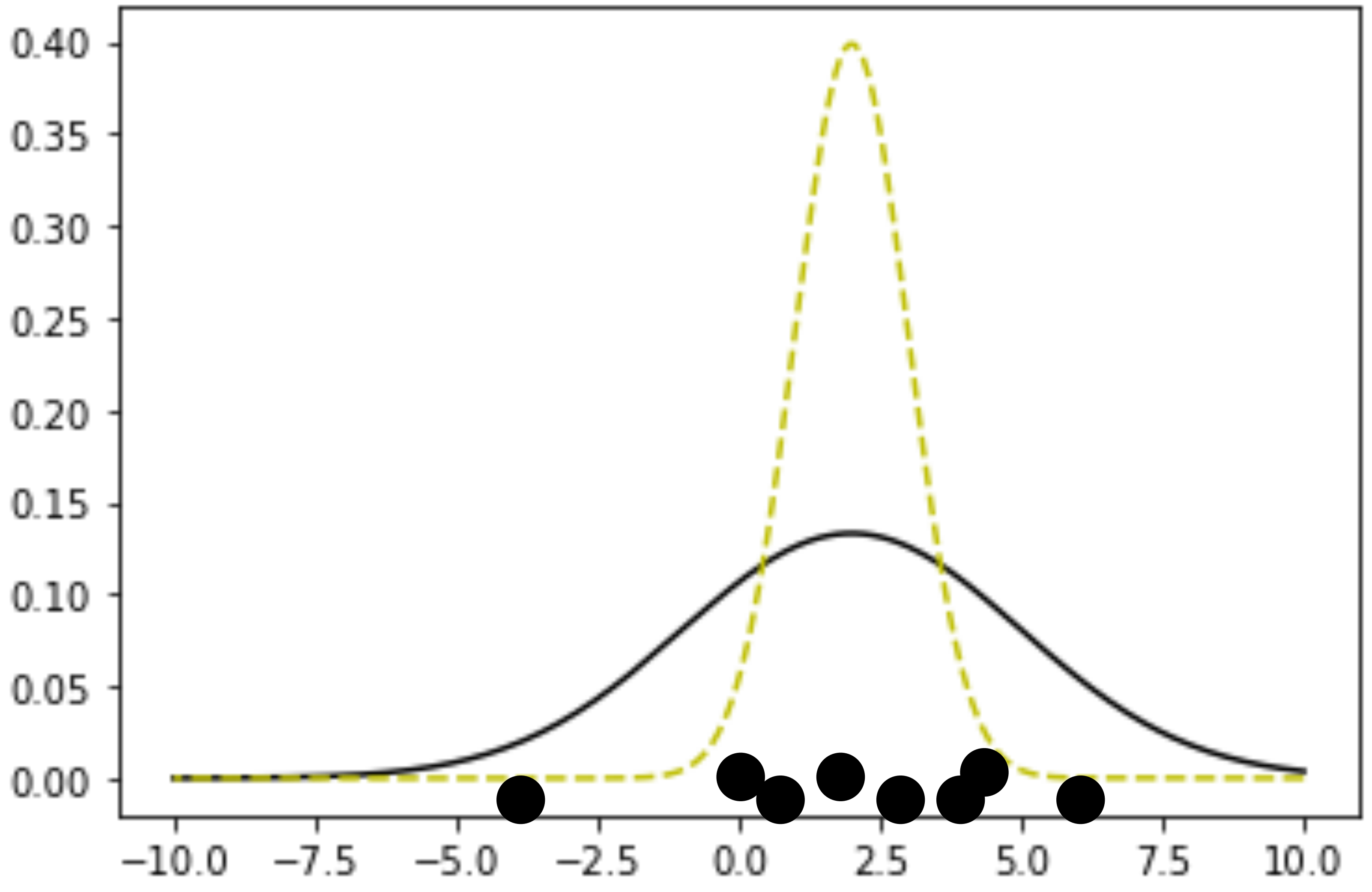
SciPy
Distributions

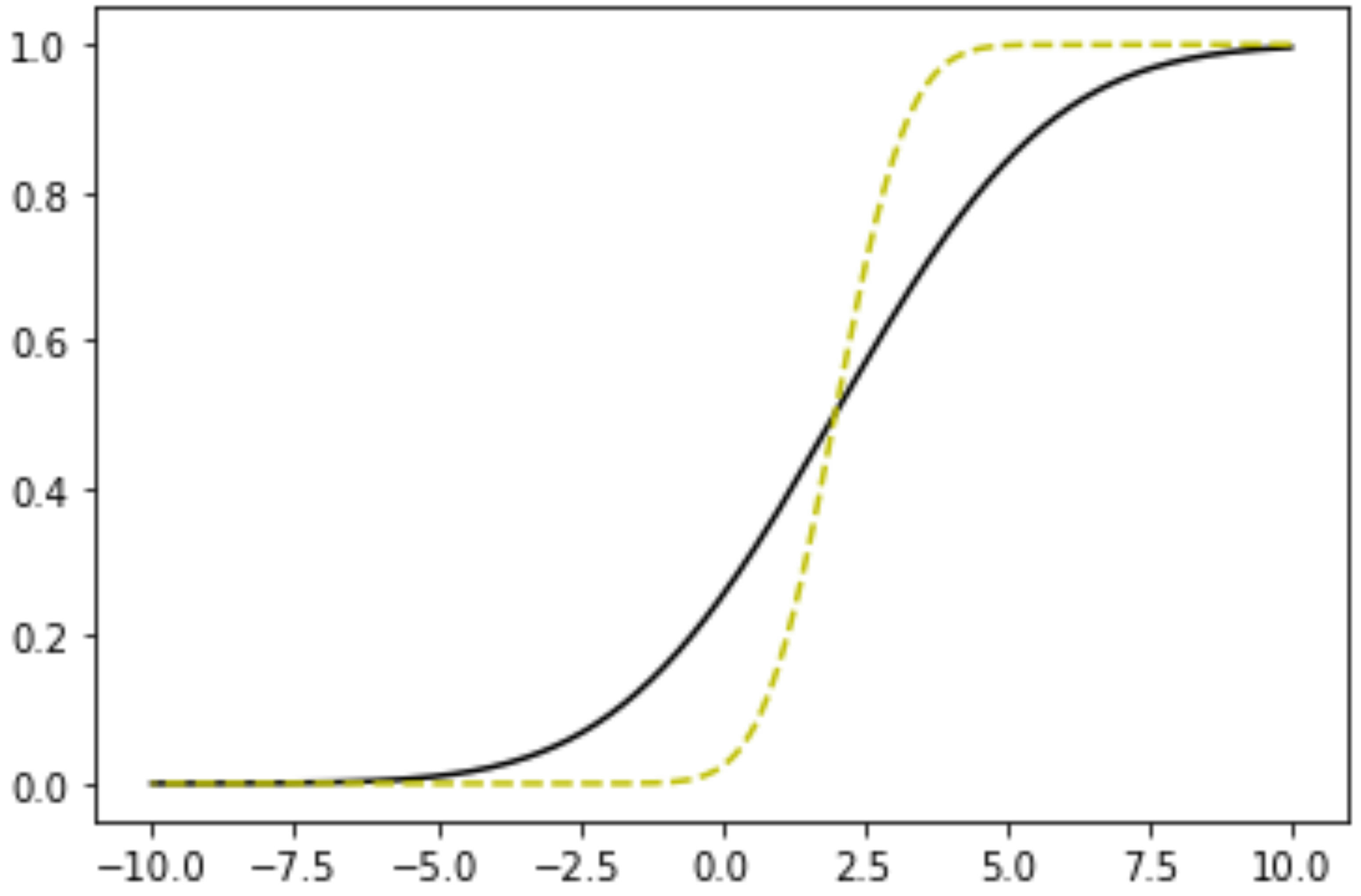




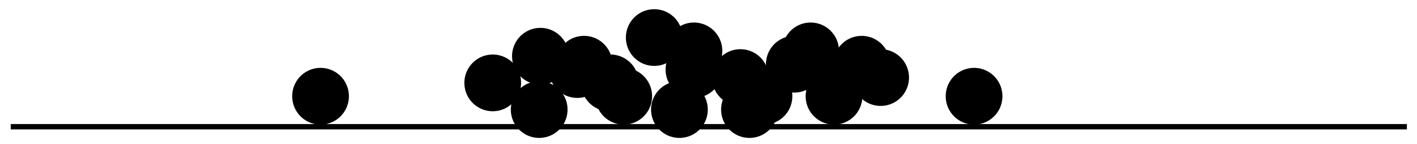
- scipy.stats.DISTRIBUTION(params)
  - methods:
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

SciPy
Distributions



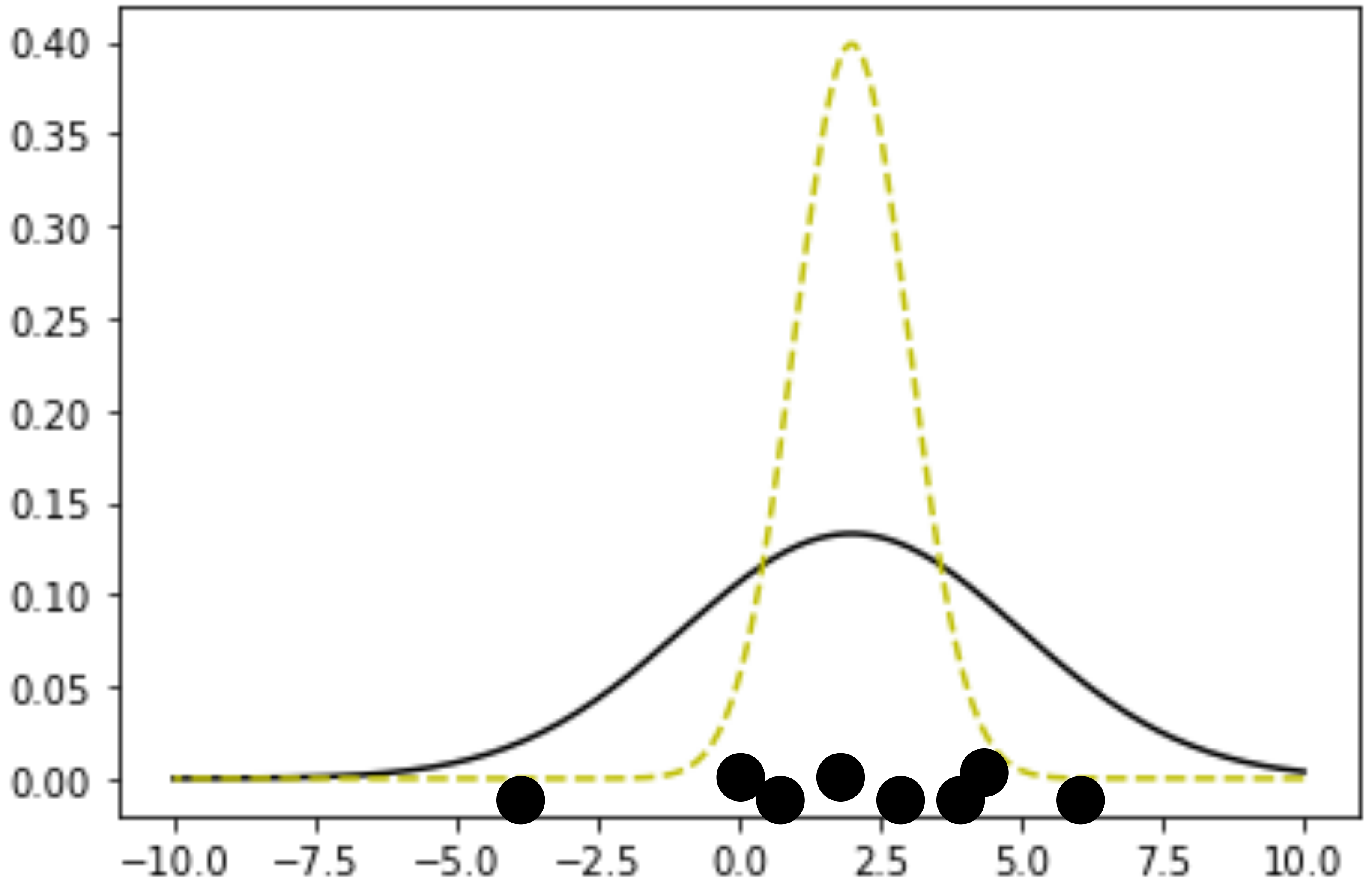


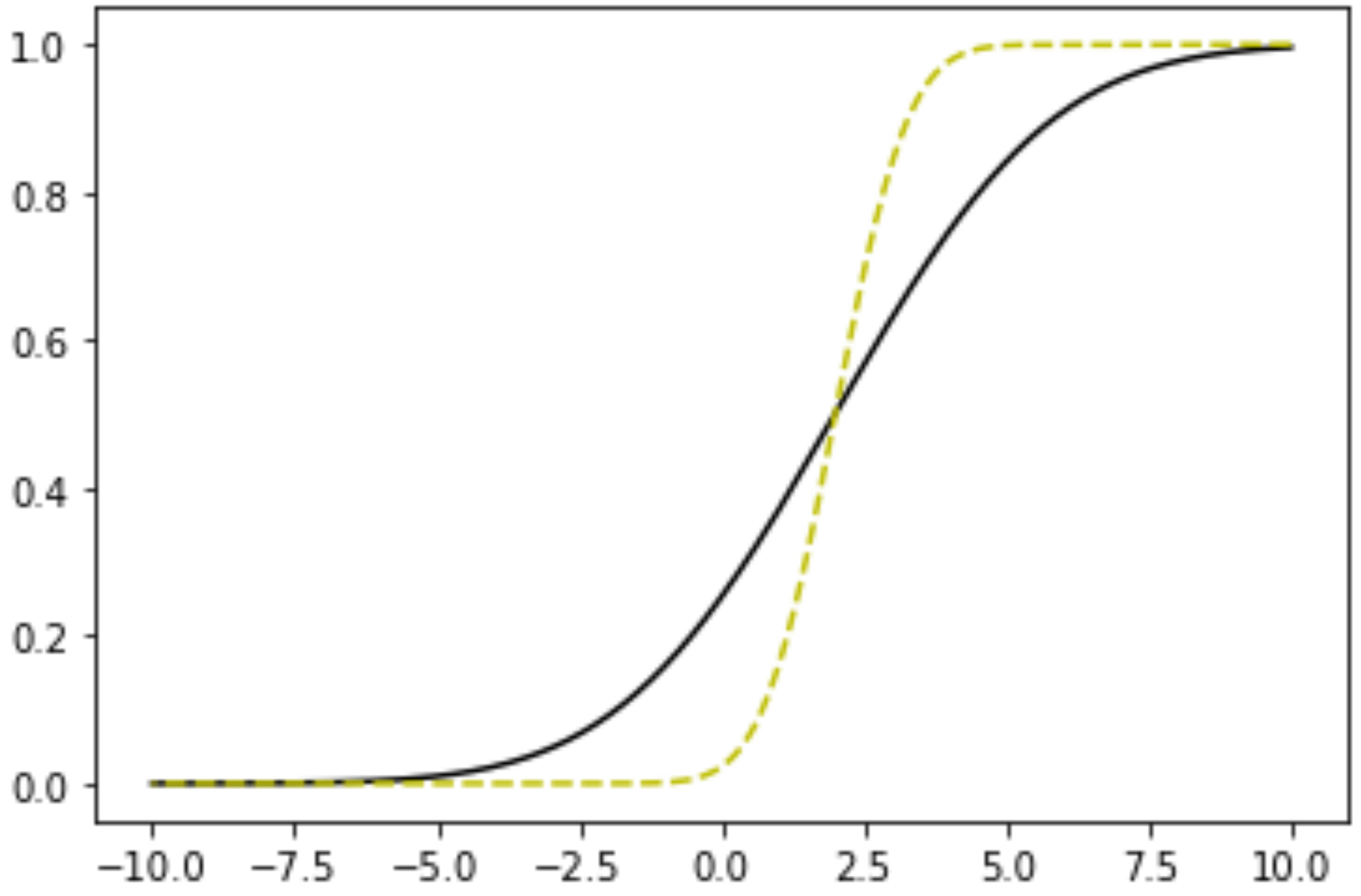
- scipy.stats.DISTRIBUTION(params)
  - methods:



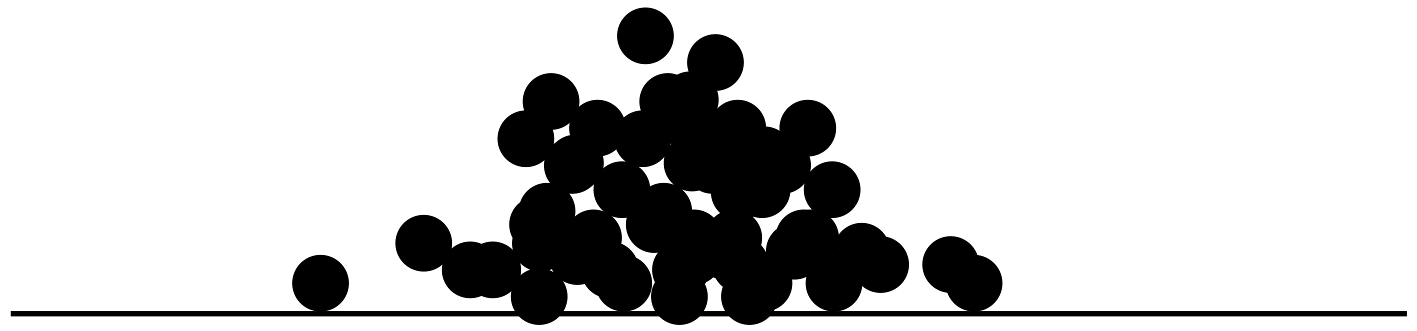
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

SciPy
Distributions





- scipy.stats.DISTRIBUTION(params)
  - methods:



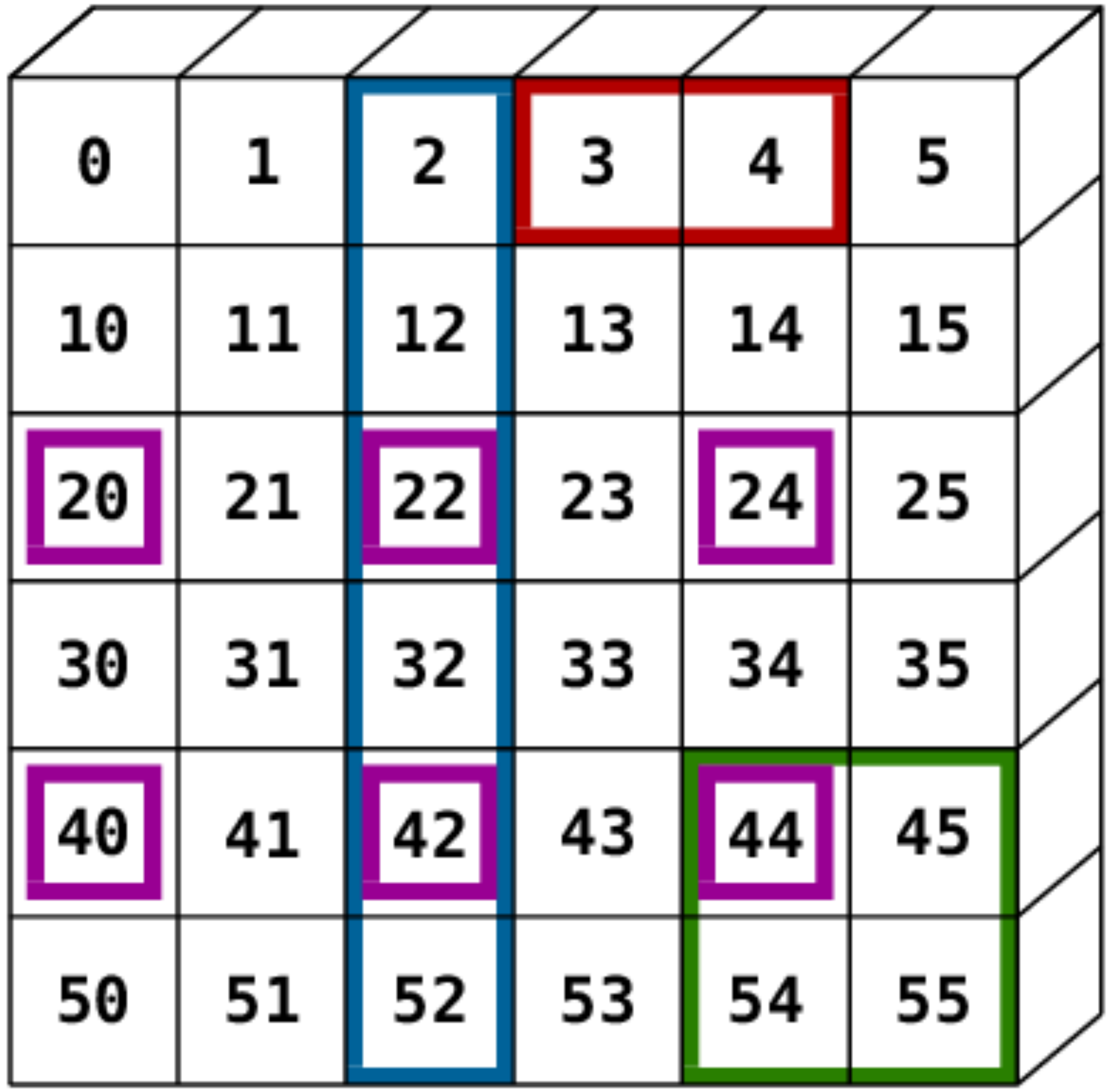
    - pdf(x) — probability density function (or pmf for discrete distribution) at x
    - cdf(x) — cumulative density function at x (pr(y<=x))
    - ppf(P) — percentile function, returns the x that makes pr(x<=x)=P true
    - rvs(n) — generate random variates from distribution

NumPy indexing

Basic

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

np.ndarray



.shape= (6, 6)

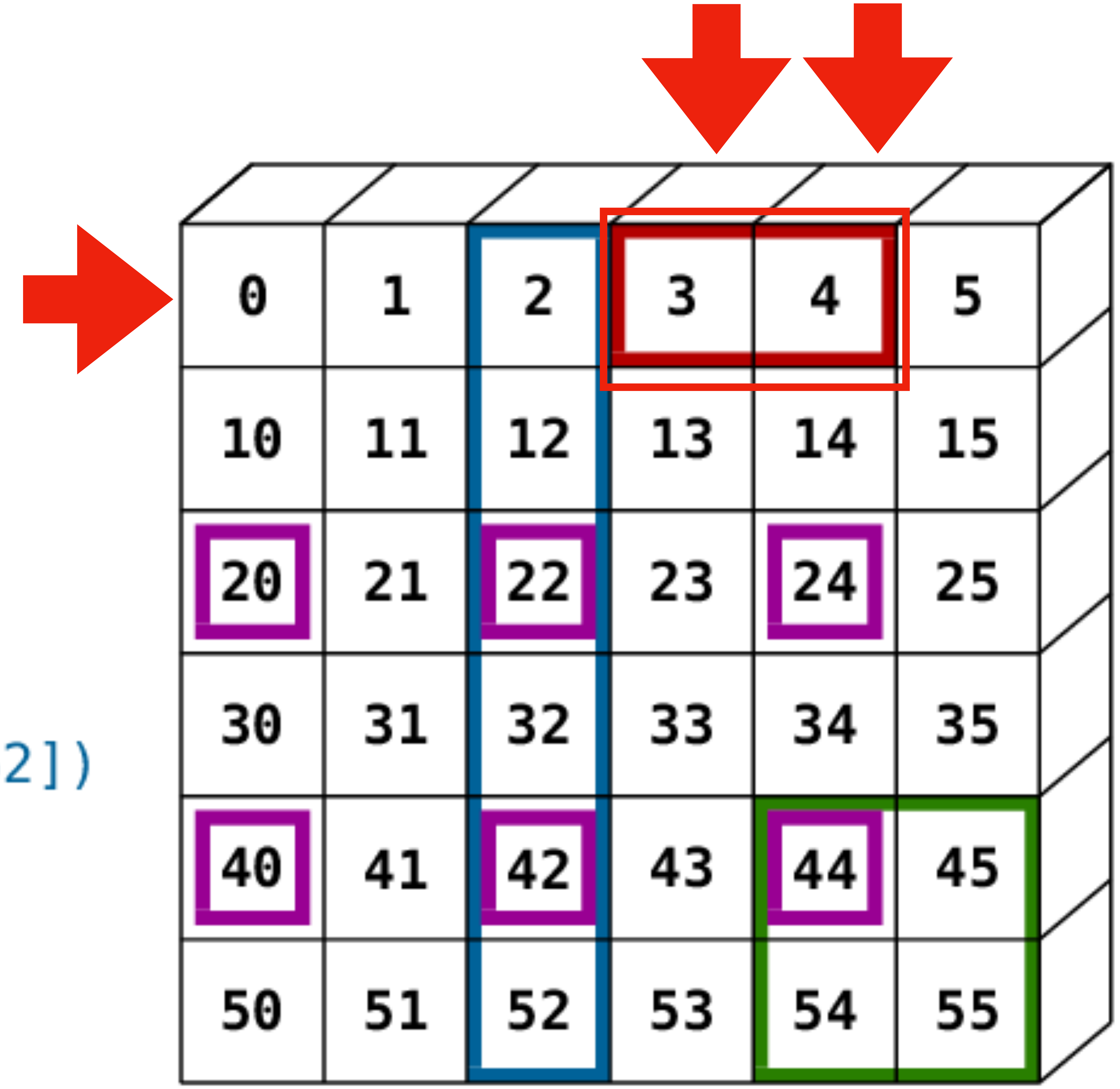
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy indexing

Basic

Row Cols

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

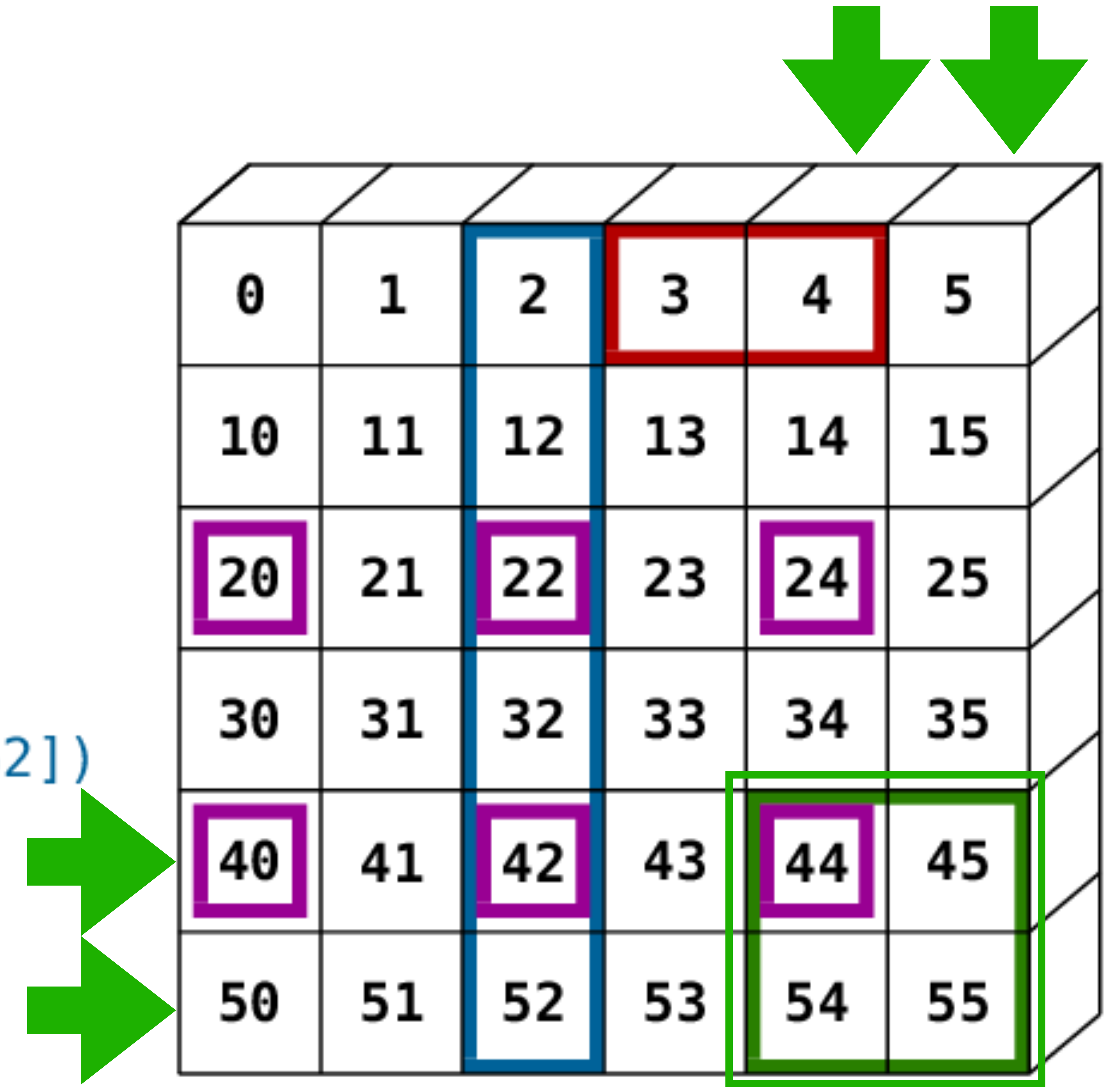


Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy indexing

Basic

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

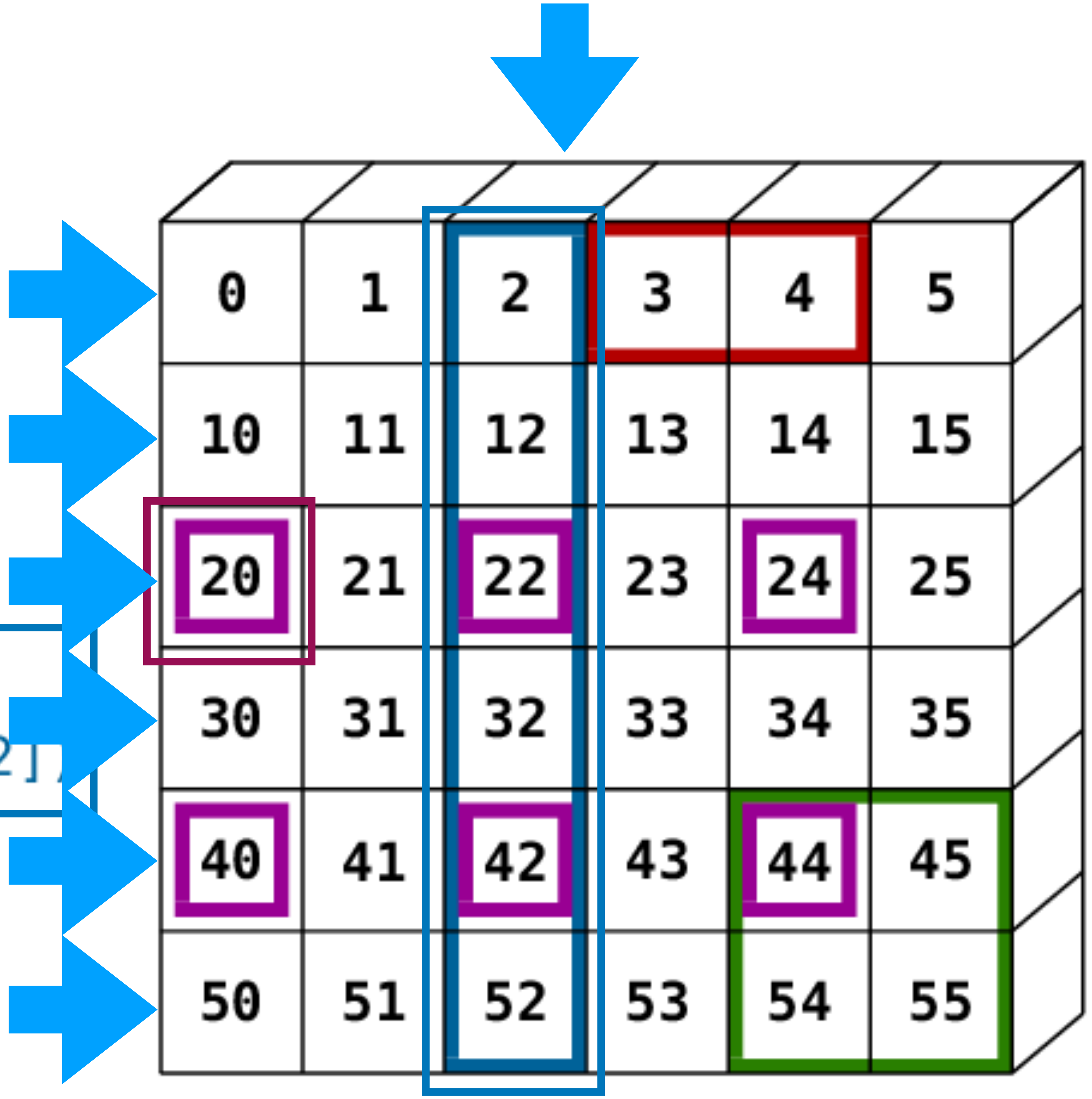


Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy indexing

Basic

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

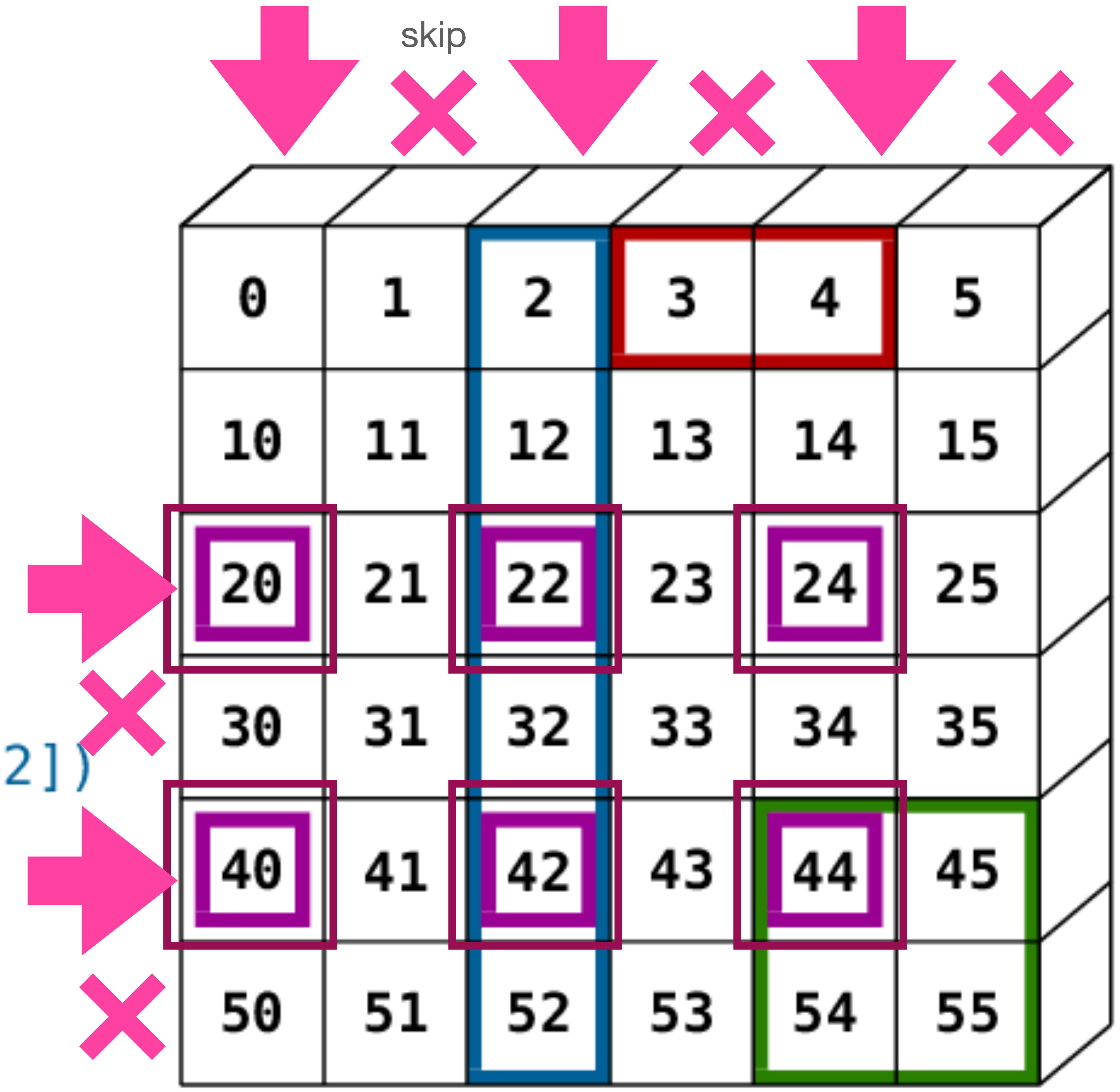
NumPy indexing

Basic

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])
```

Rows Cols

```
>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



start:stop before:step

Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy indexing

Basic

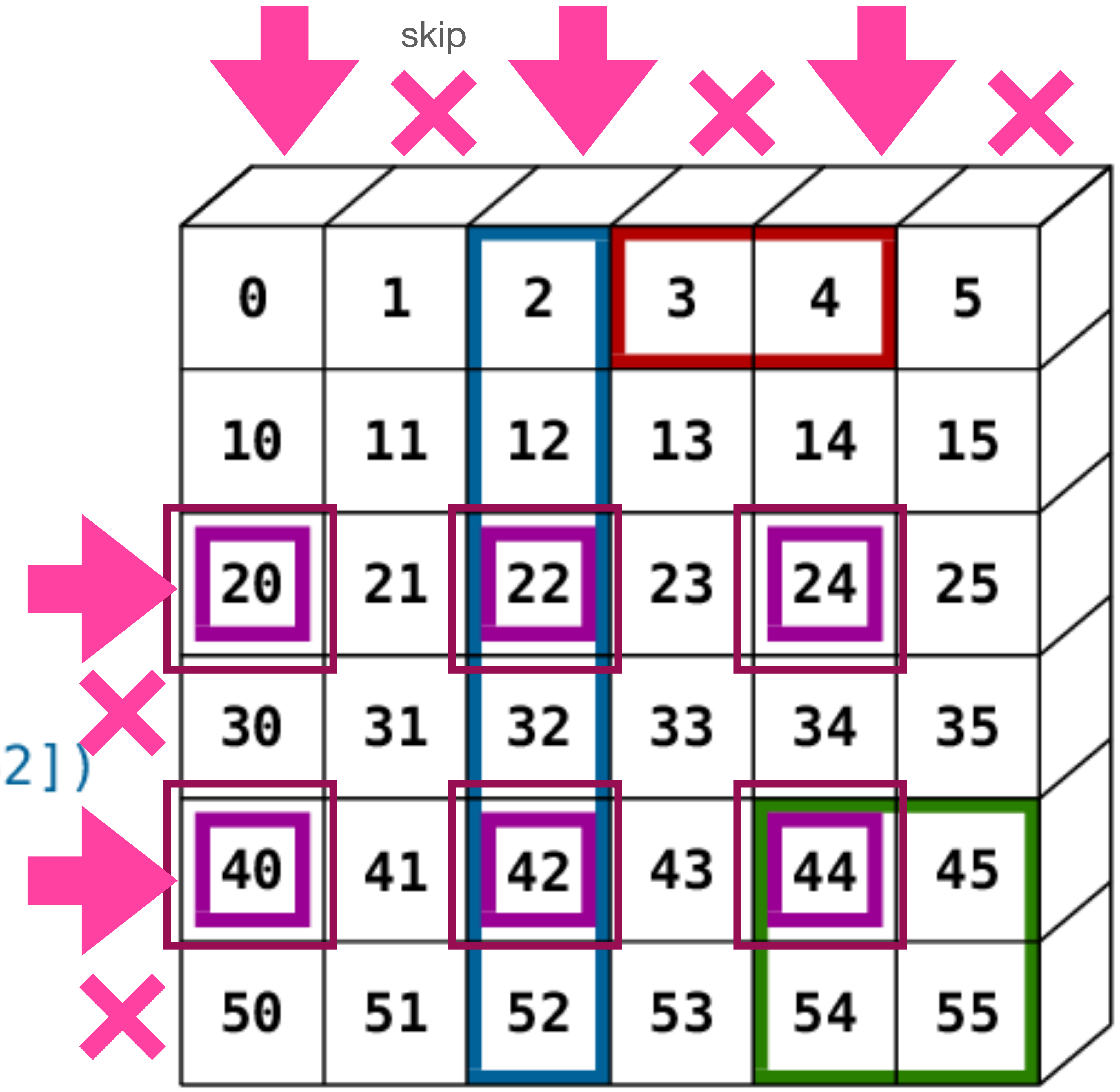
NOTE: These are VIEWS on the original data, not copies
Remember discussion of lists of lists and pointers (and deep copy)

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])
```

Rows Cols

```
>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

start:stop before:step



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

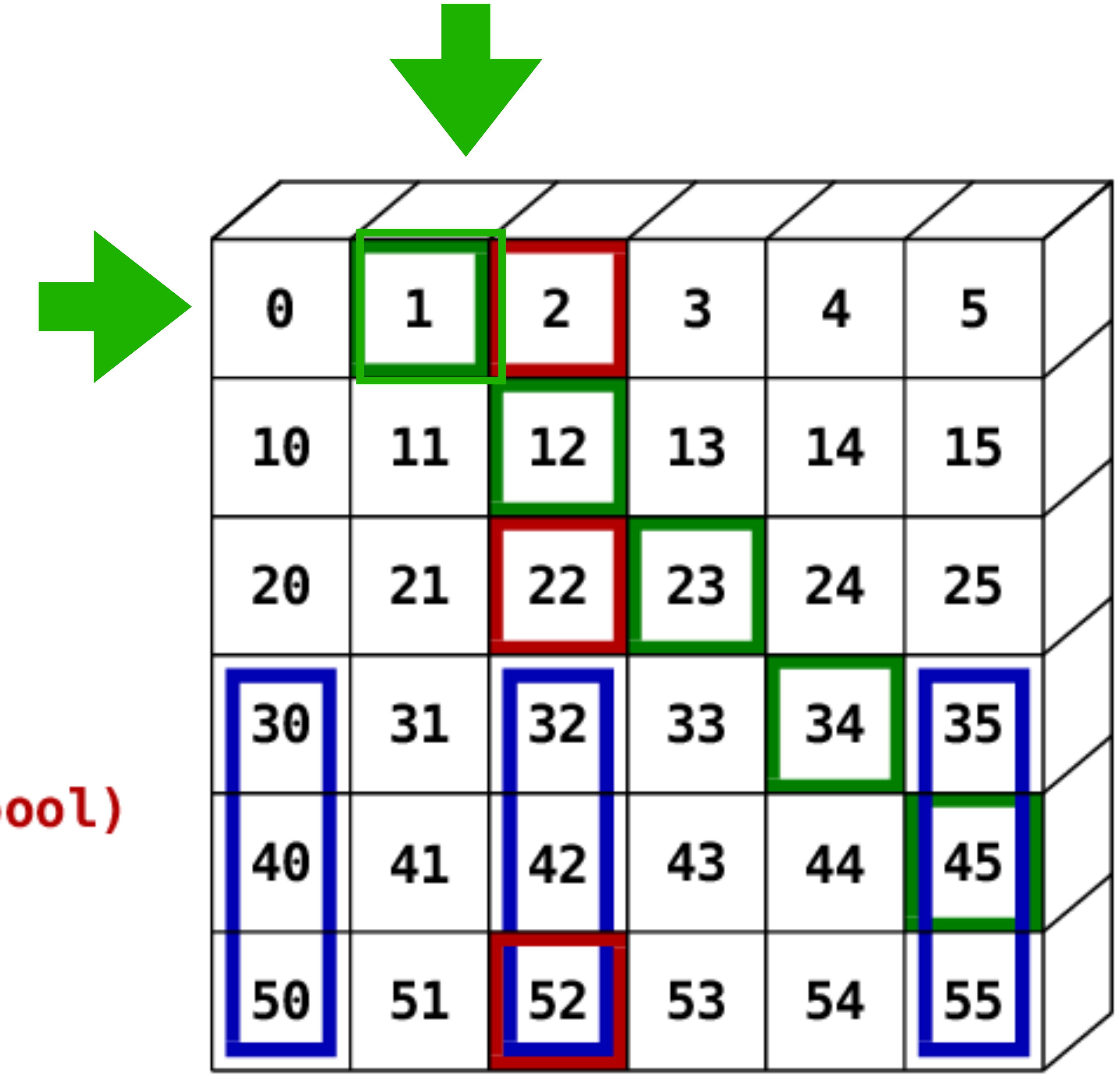
NumPy *fancy* indexing

Basic

NOTE: These are COPIES, not VIEWS

Rows Cols

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



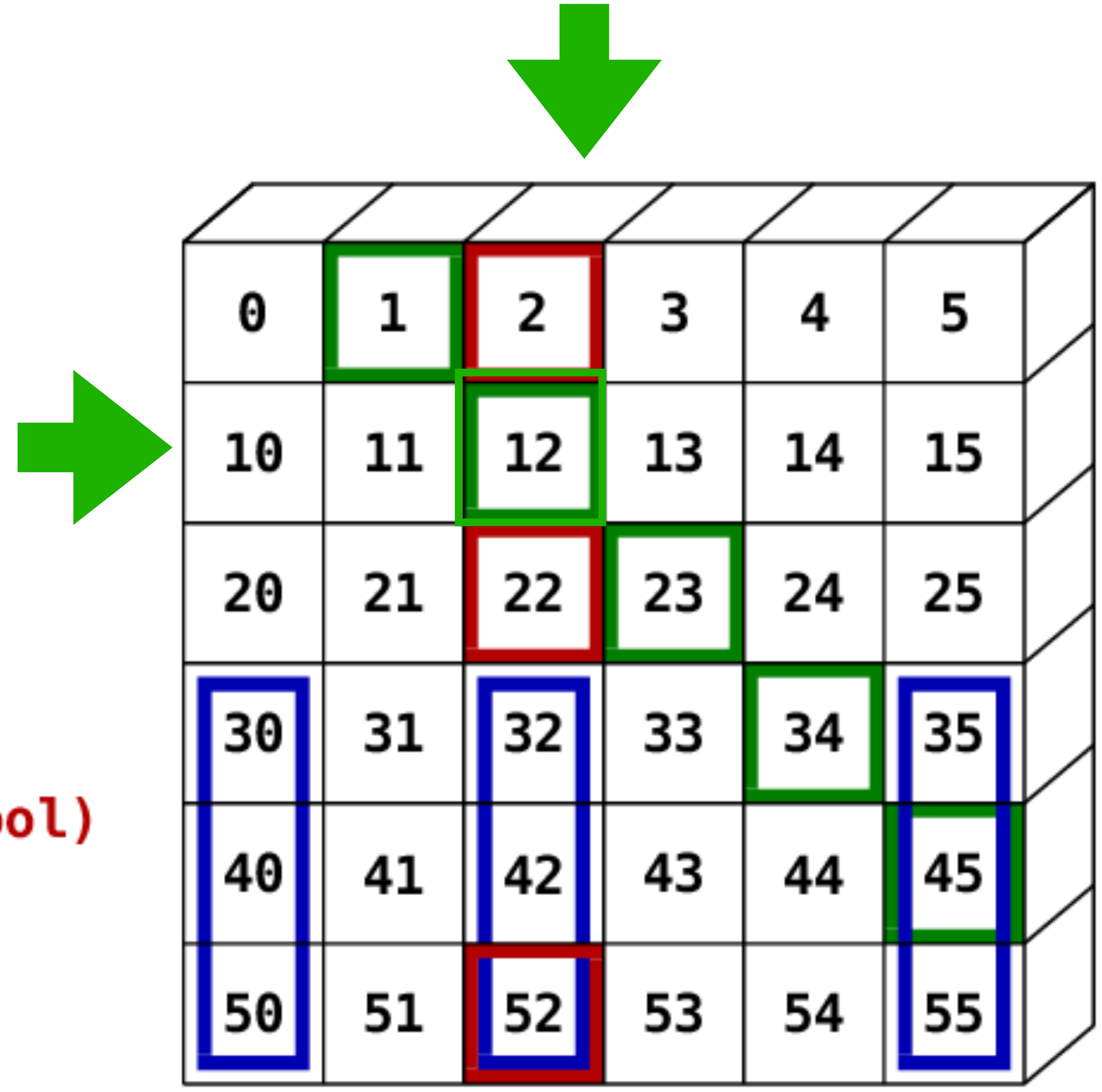
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

Basic

Rows Cols

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



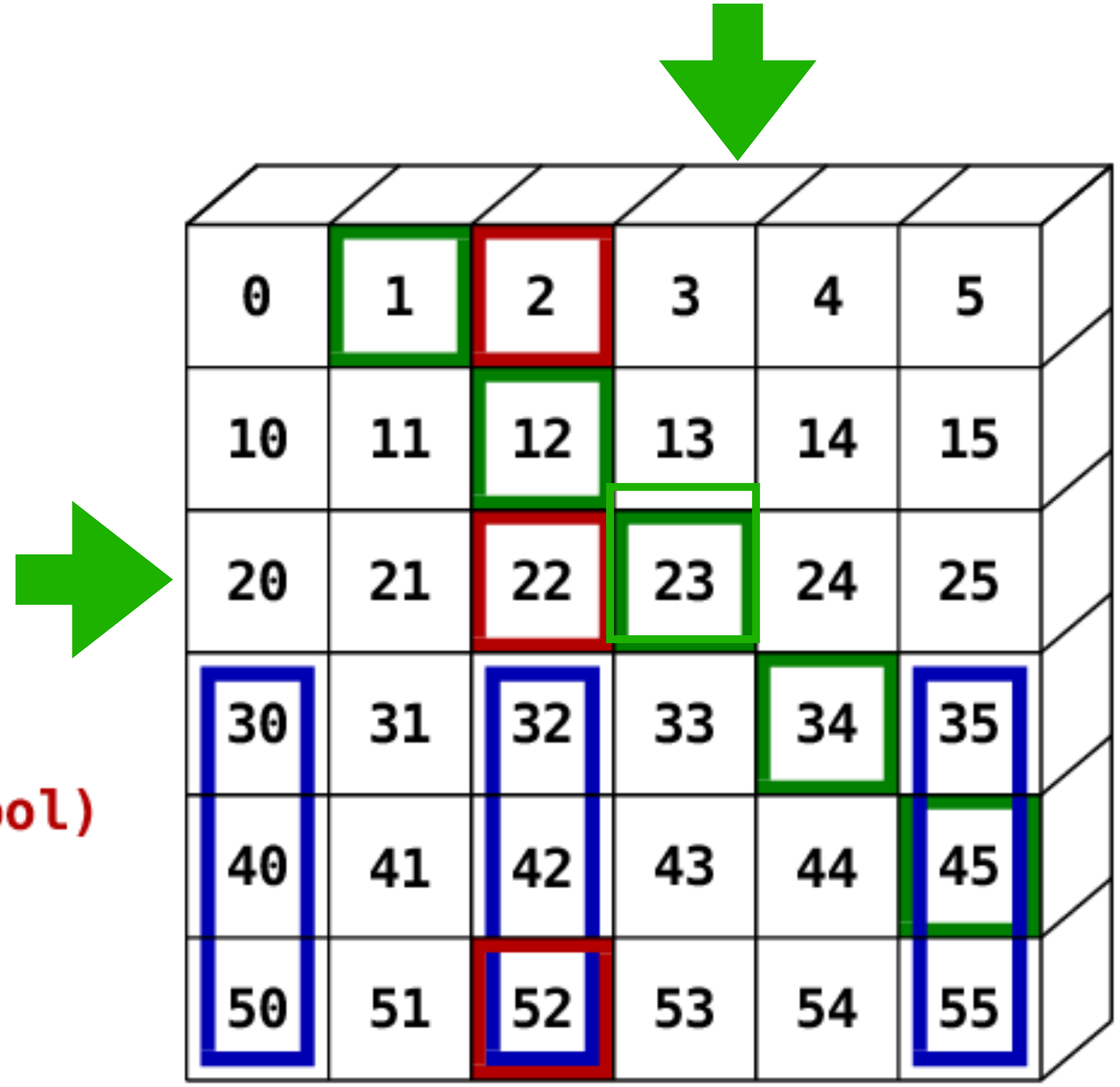
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

Basic

Rows Cols

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



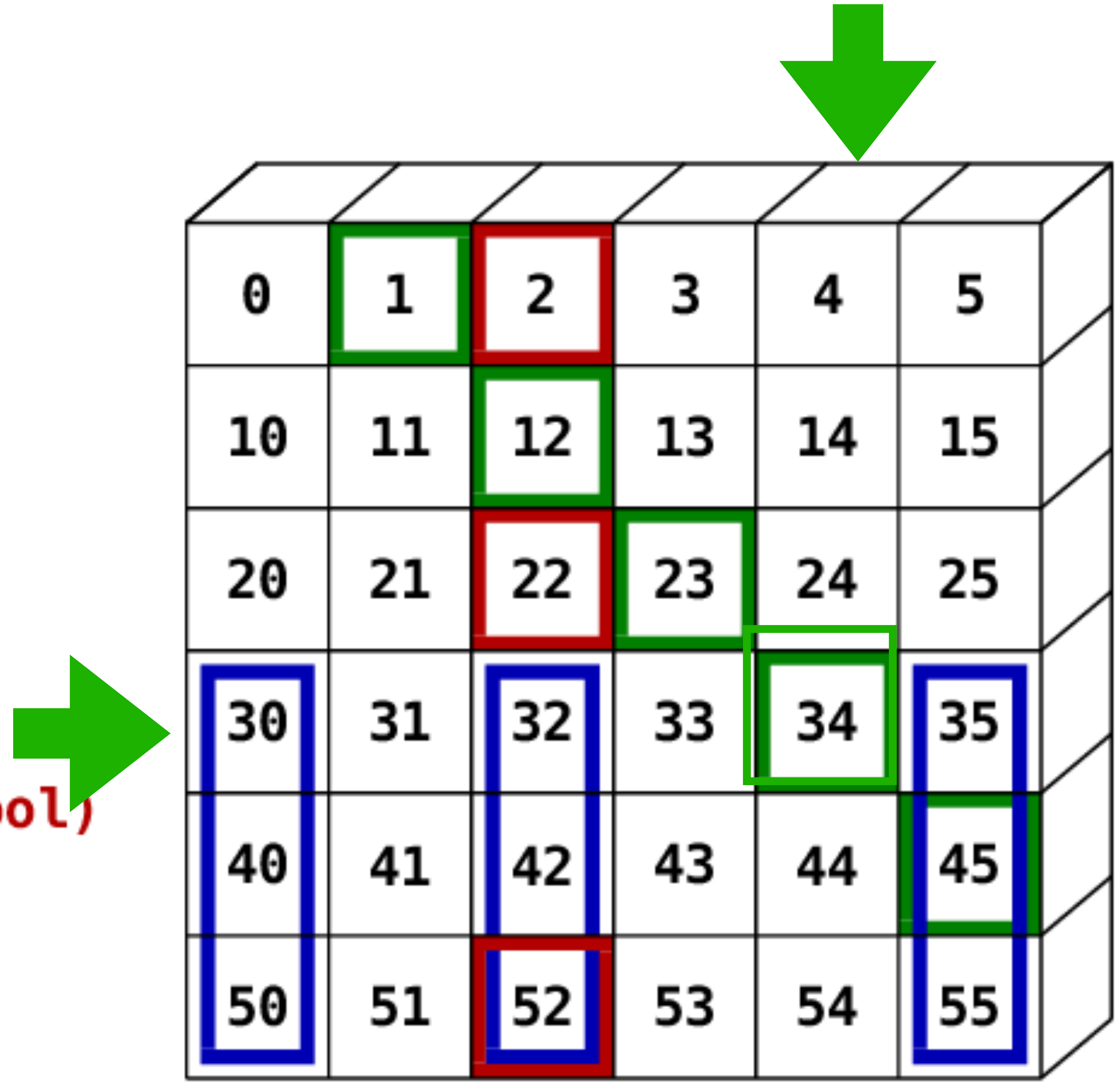
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

Basic

Rows Cols

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



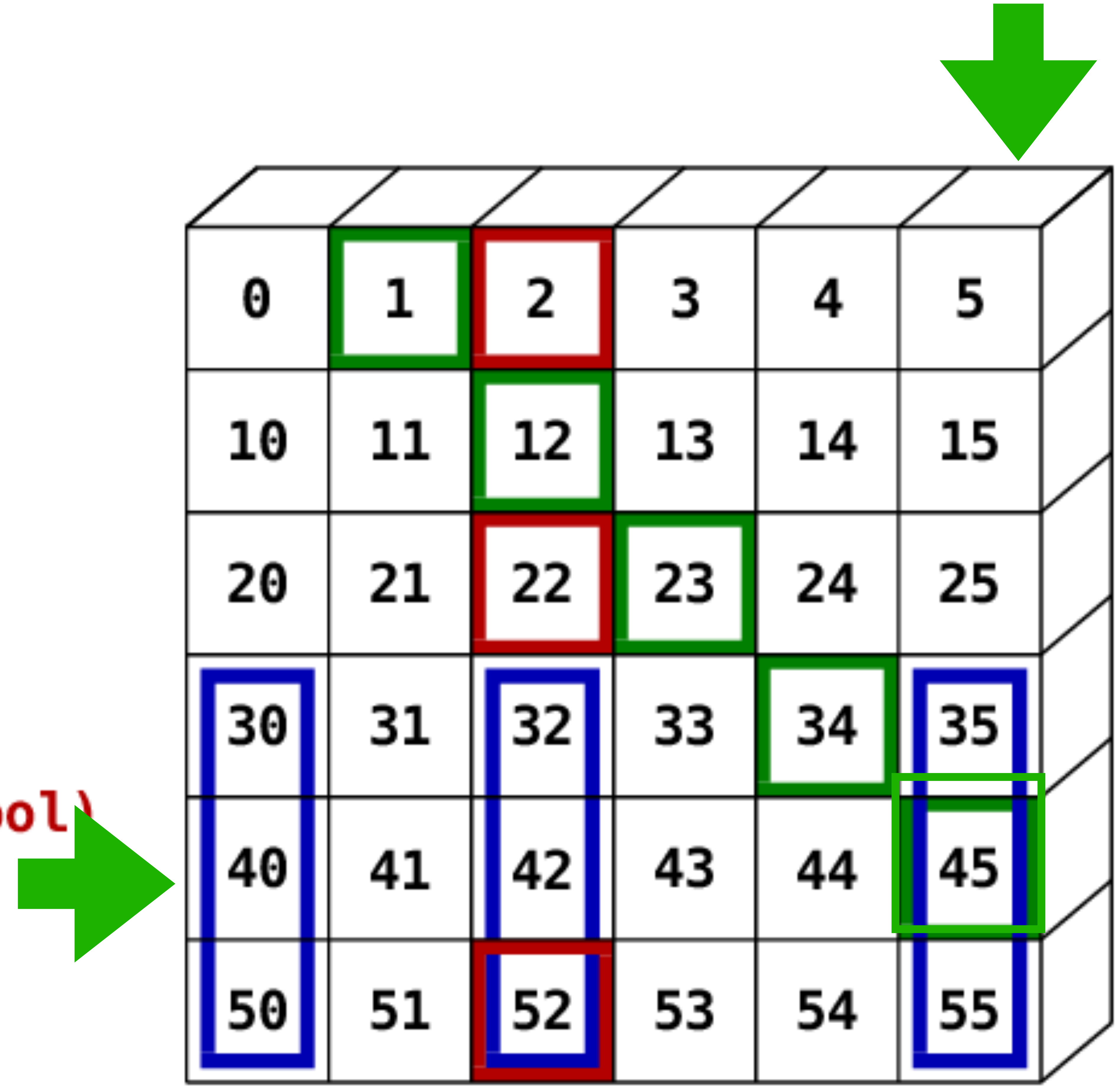
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

Basic

Rows Cols

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

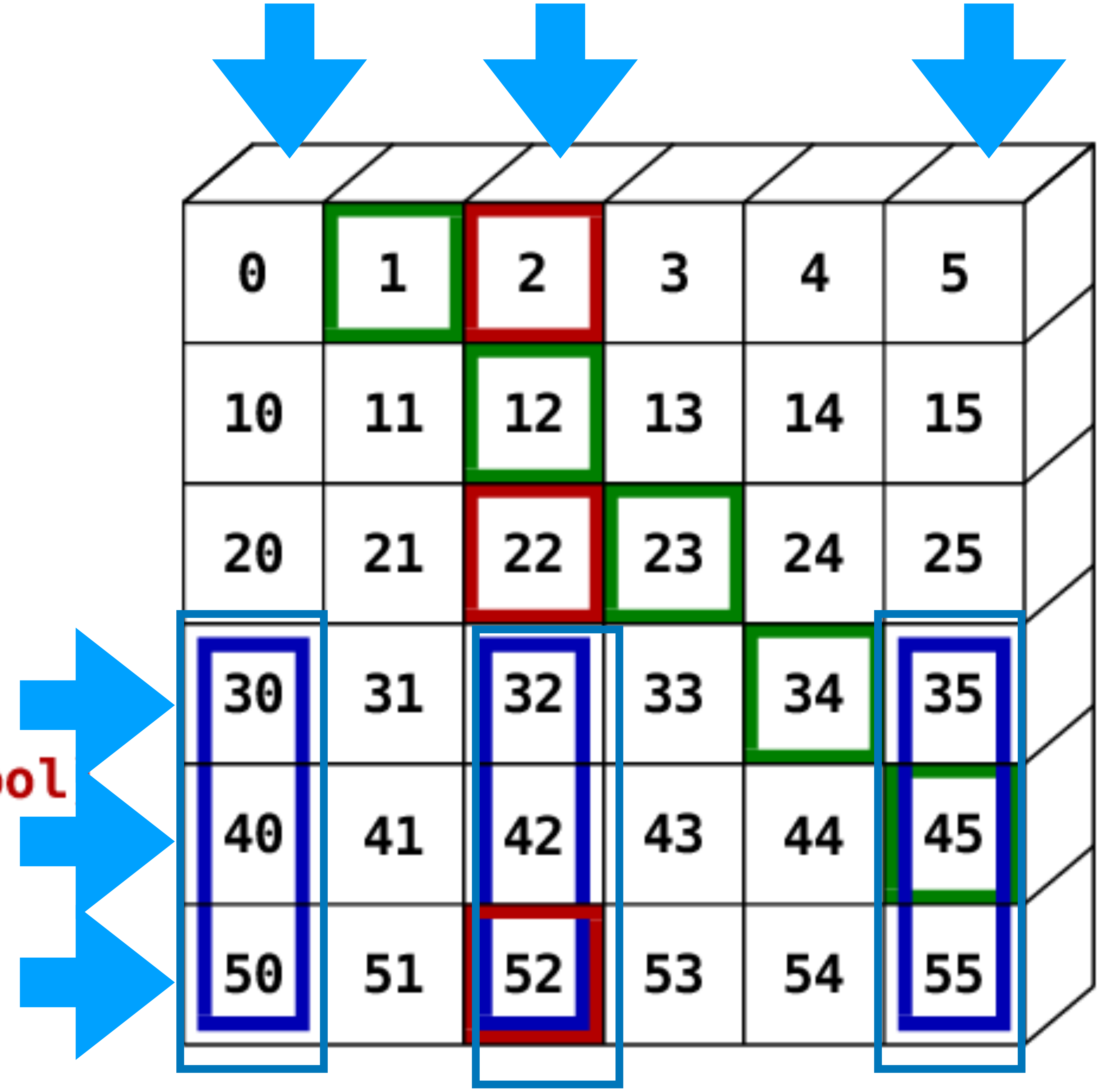
Basic

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
```

Rows Cols

```
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool
>>> a[mask, 2]
array([2, 22, 52])
```



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

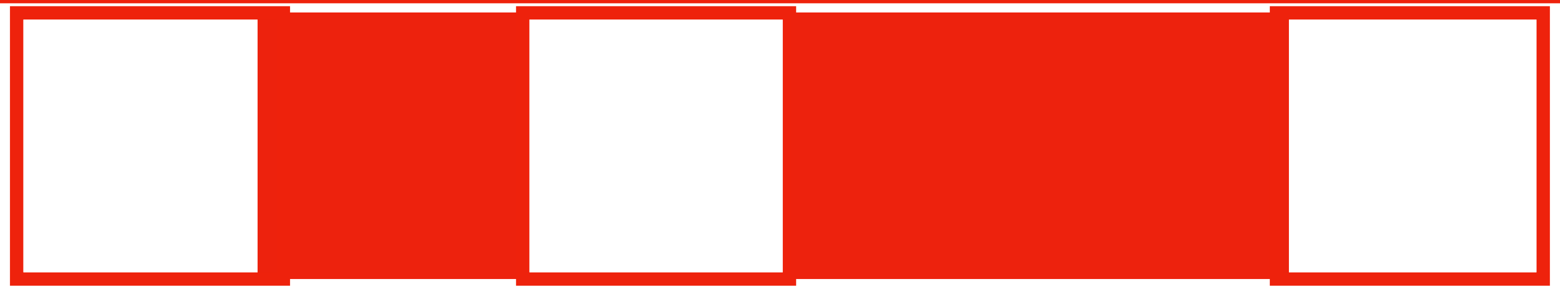
Basic

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
```

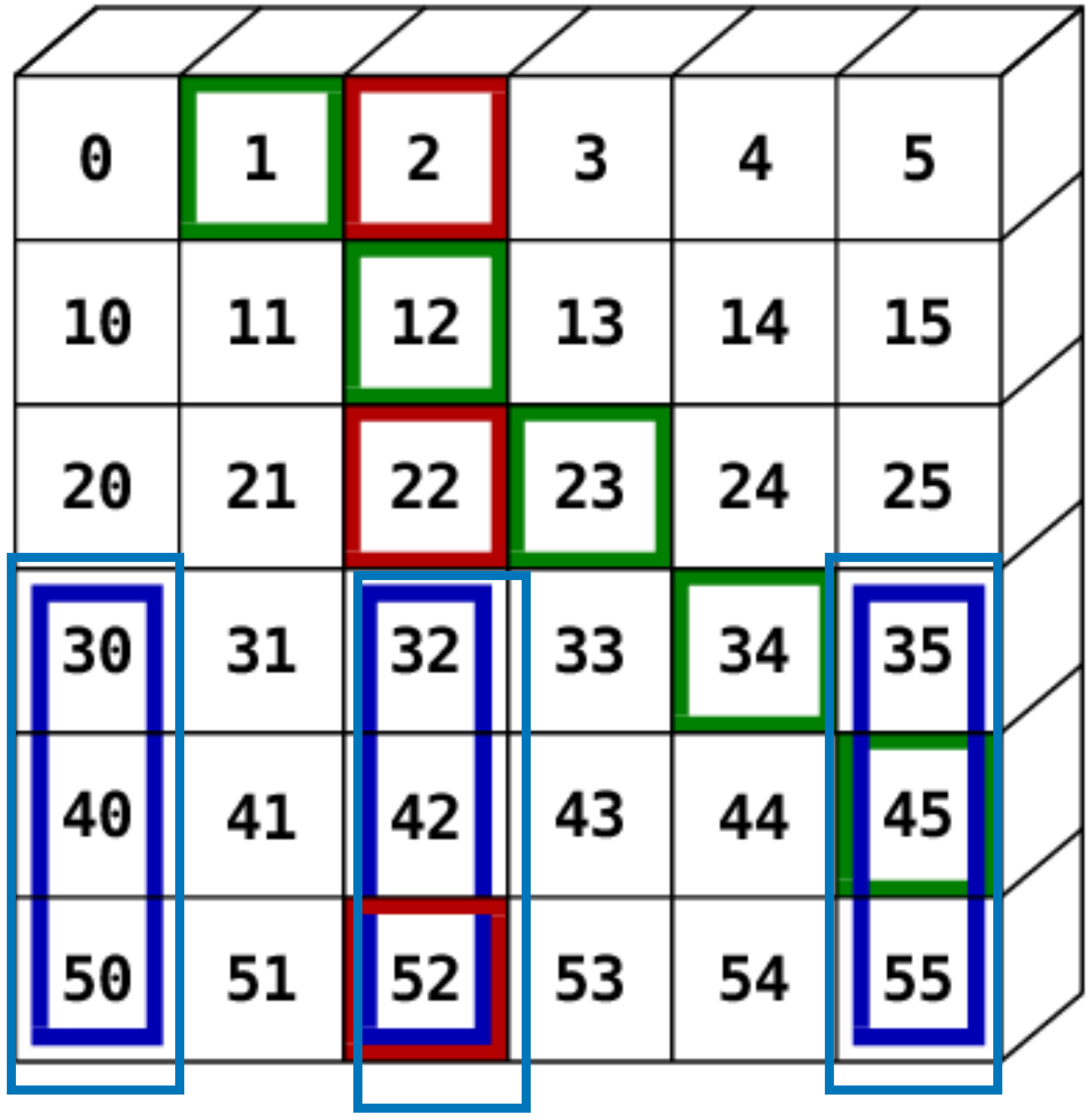
Rows Cols

```
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



1 0 1 0 0 1



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

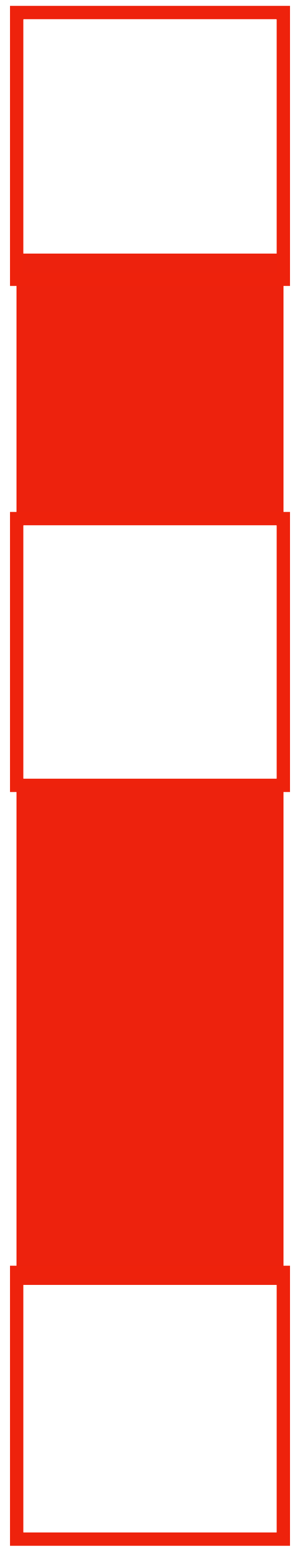
Basic

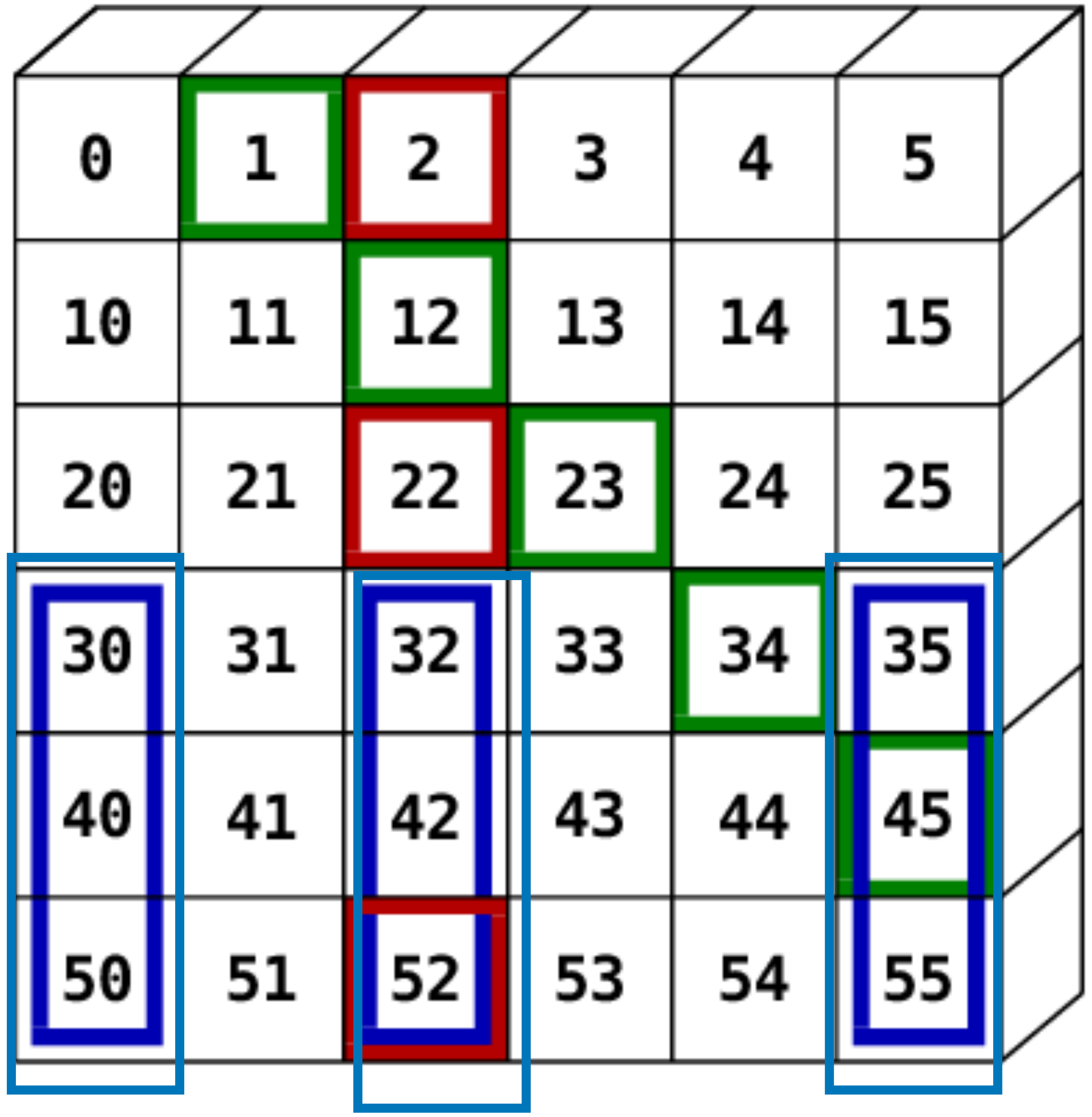
```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
```

Rows Cols

```
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```





Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

Basic

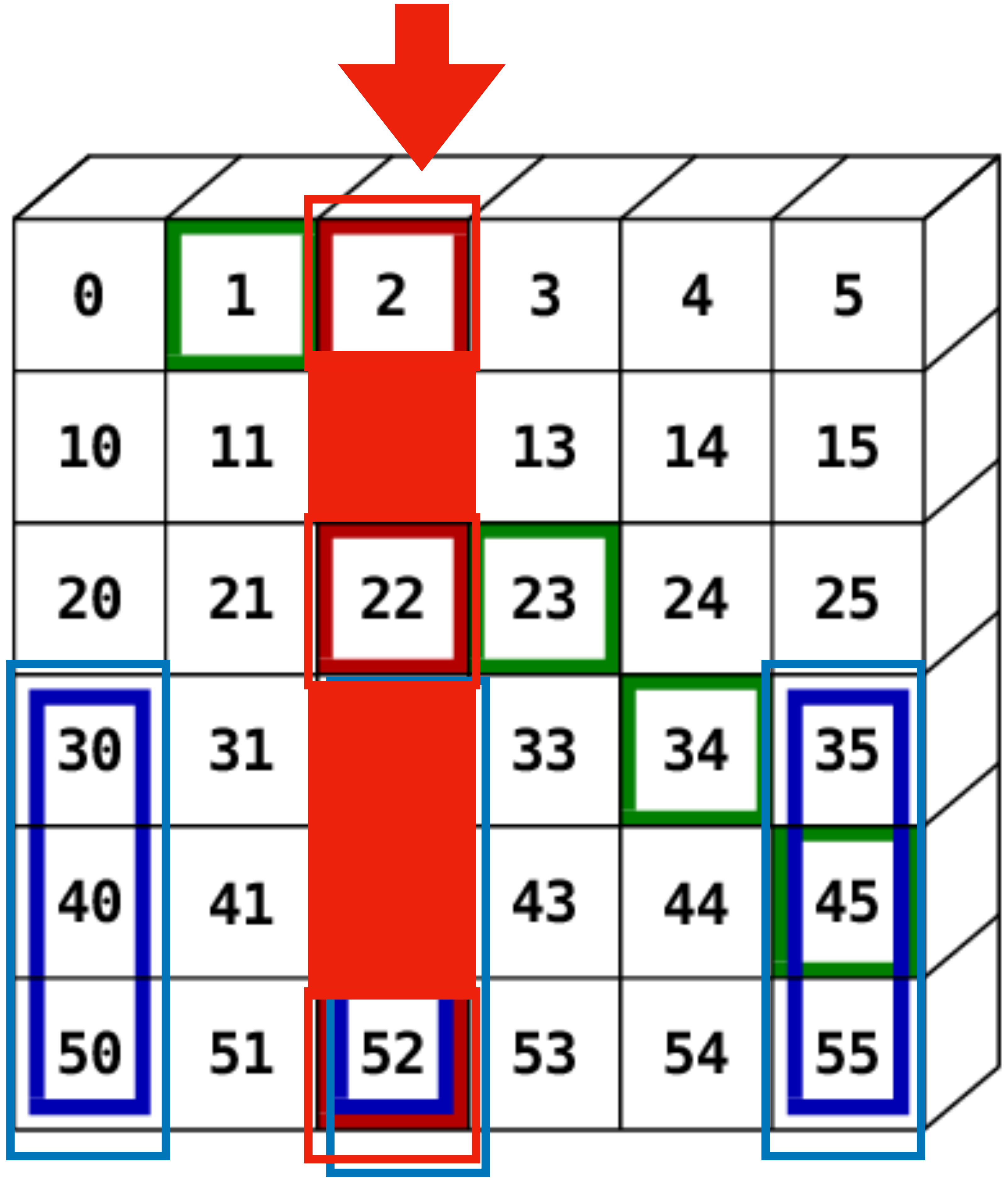
```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
```

Rows Cols

```
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```

1 0 1 0 0 1



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy *fancy* indexing

Basic

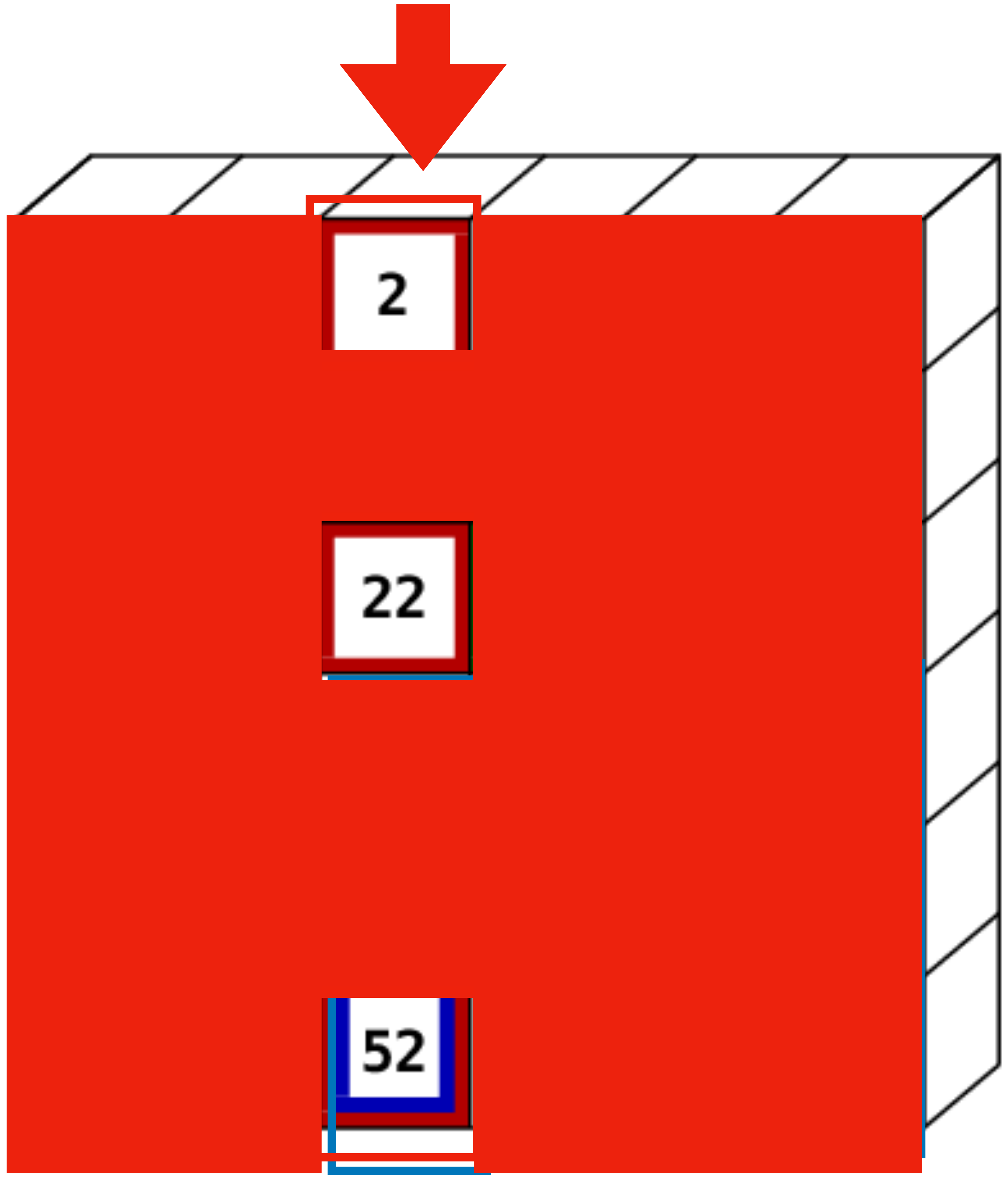
So fancy indexing copies!
And is done by either a sequence of slices (notice the lists/tuples of indices)
or a boolean mask

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])
```

Rows Cols

```
>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

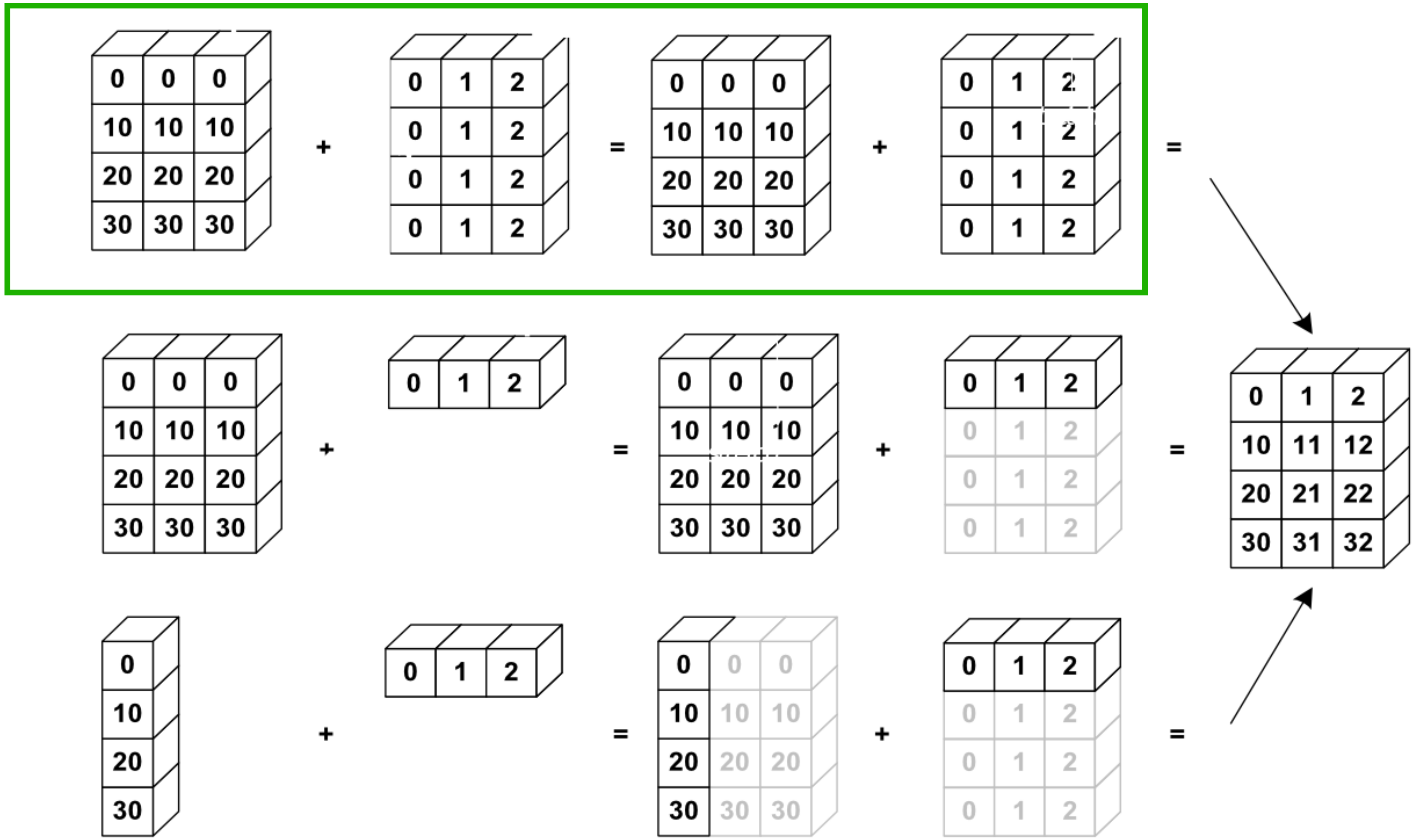
```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

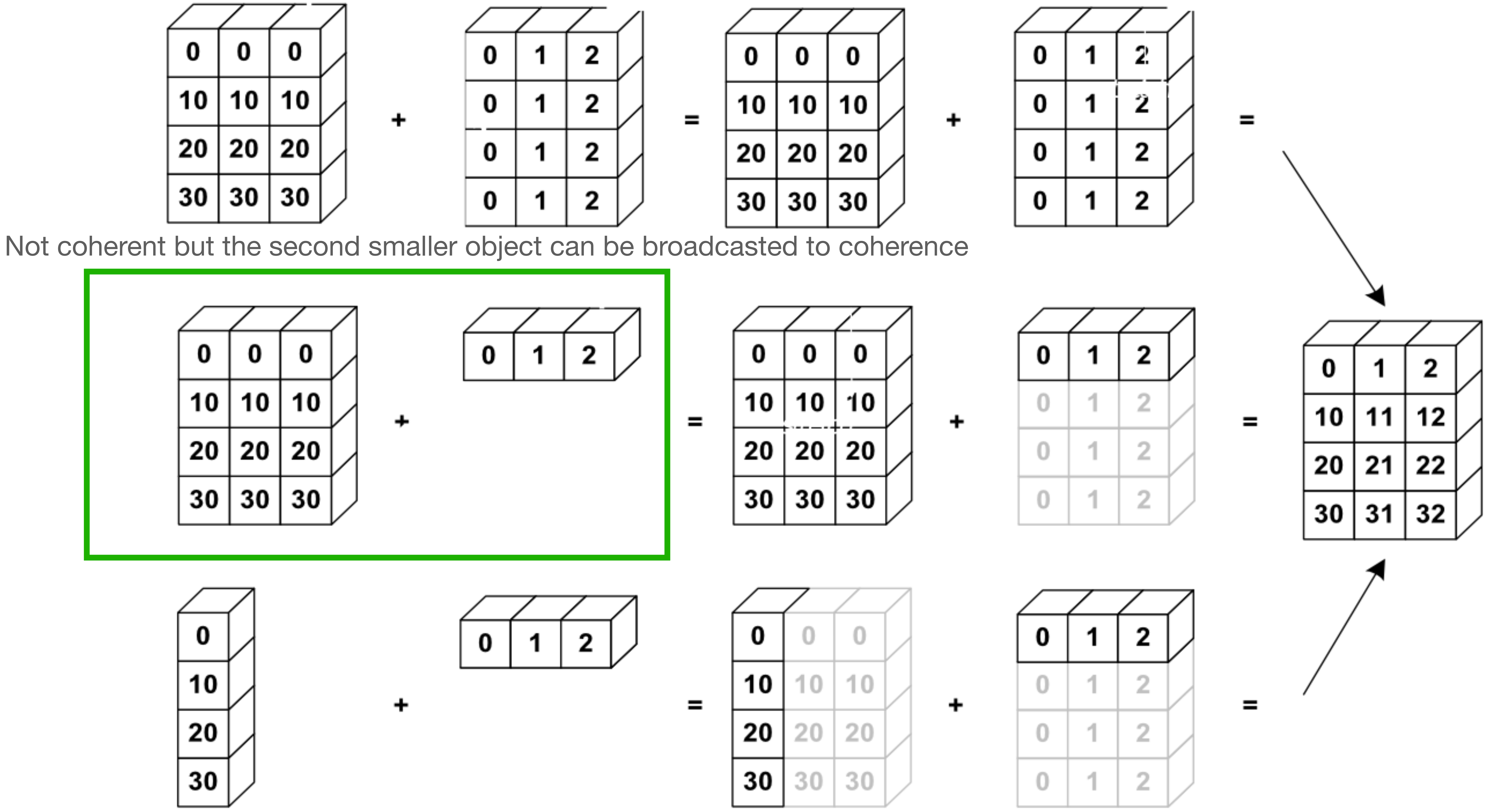
NumPy Broadcasting Example

coherent shapes for action (addition here)



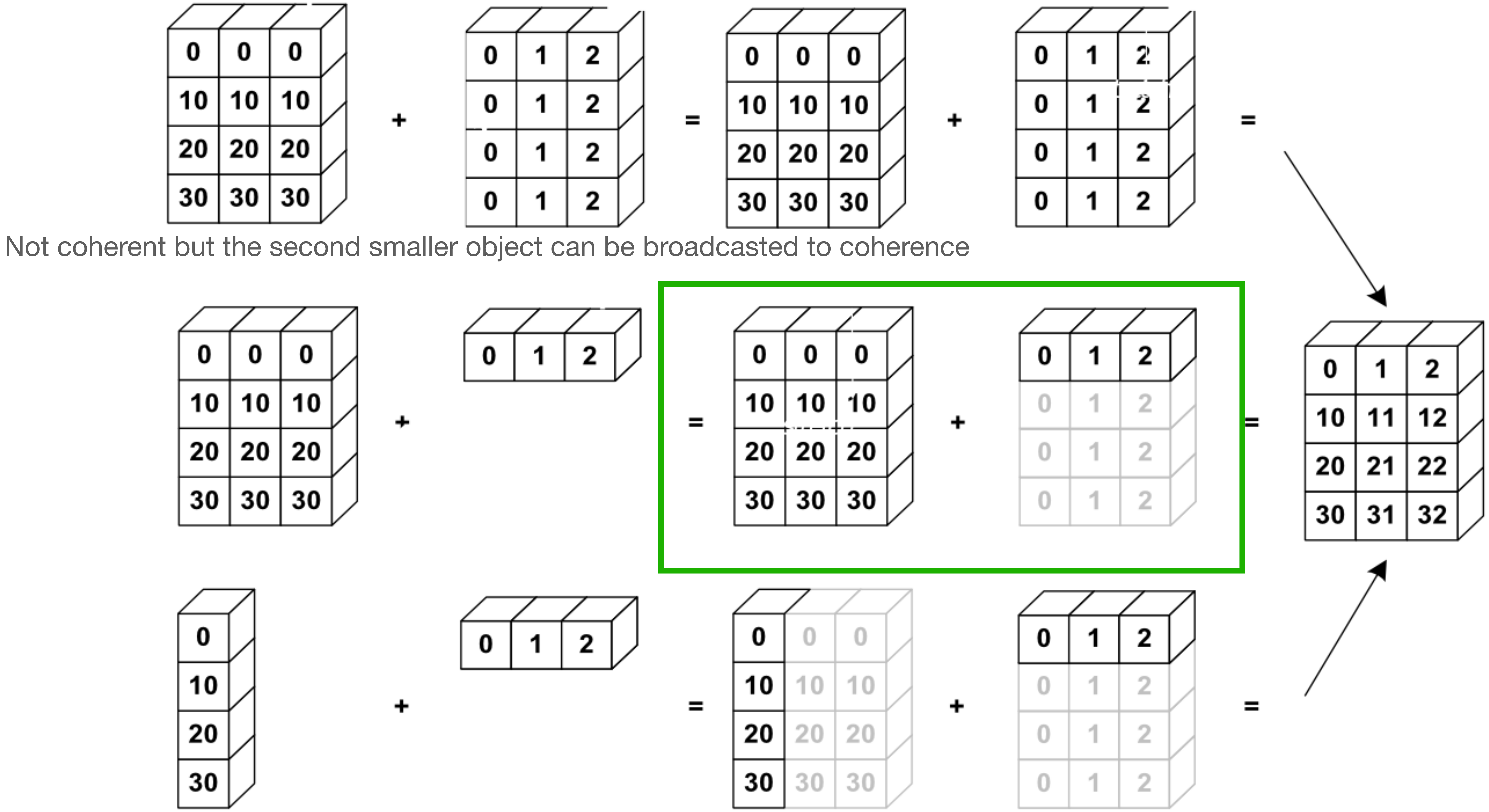
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy Broadcasting Example



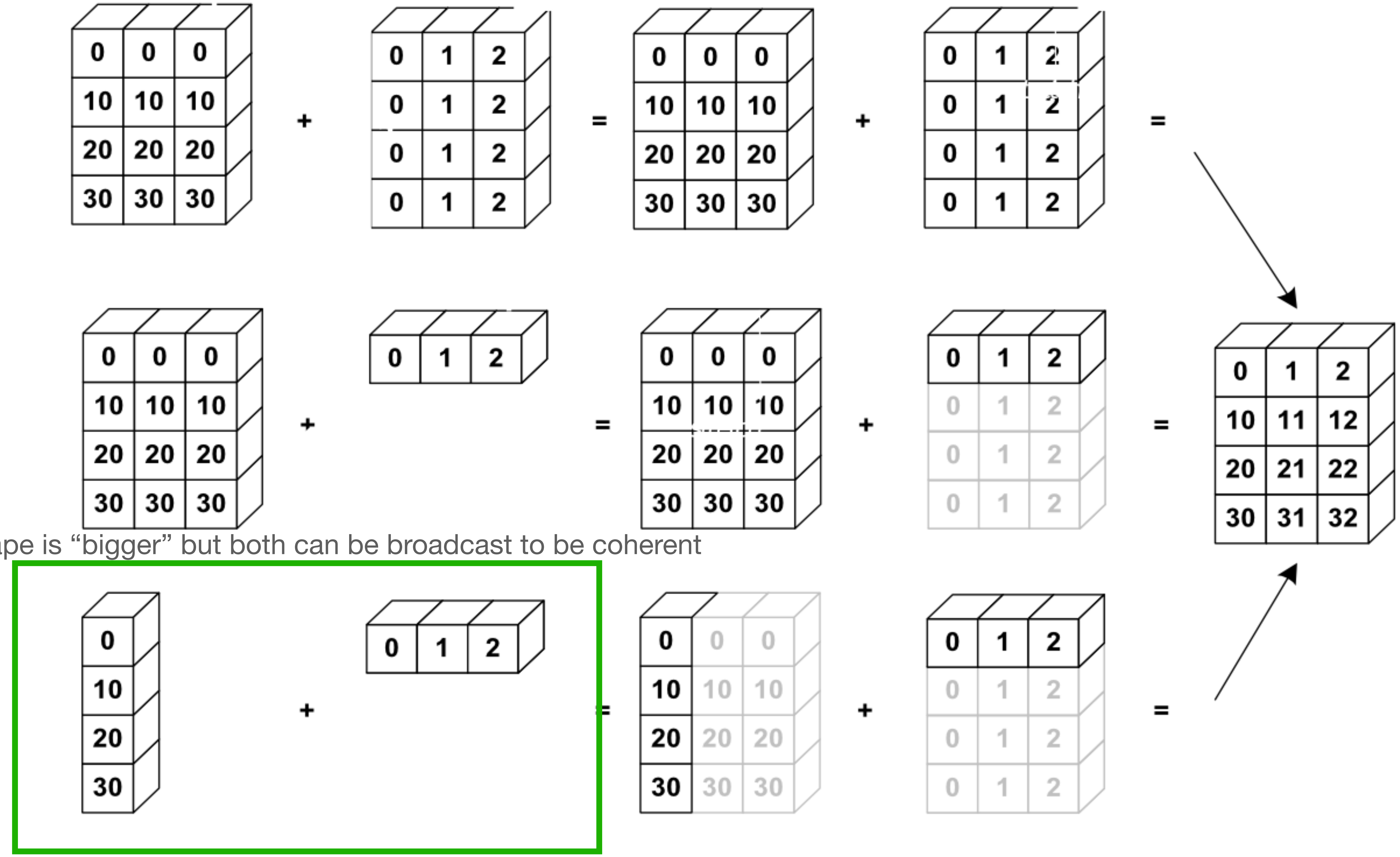
Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy Broadcasting Example



Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

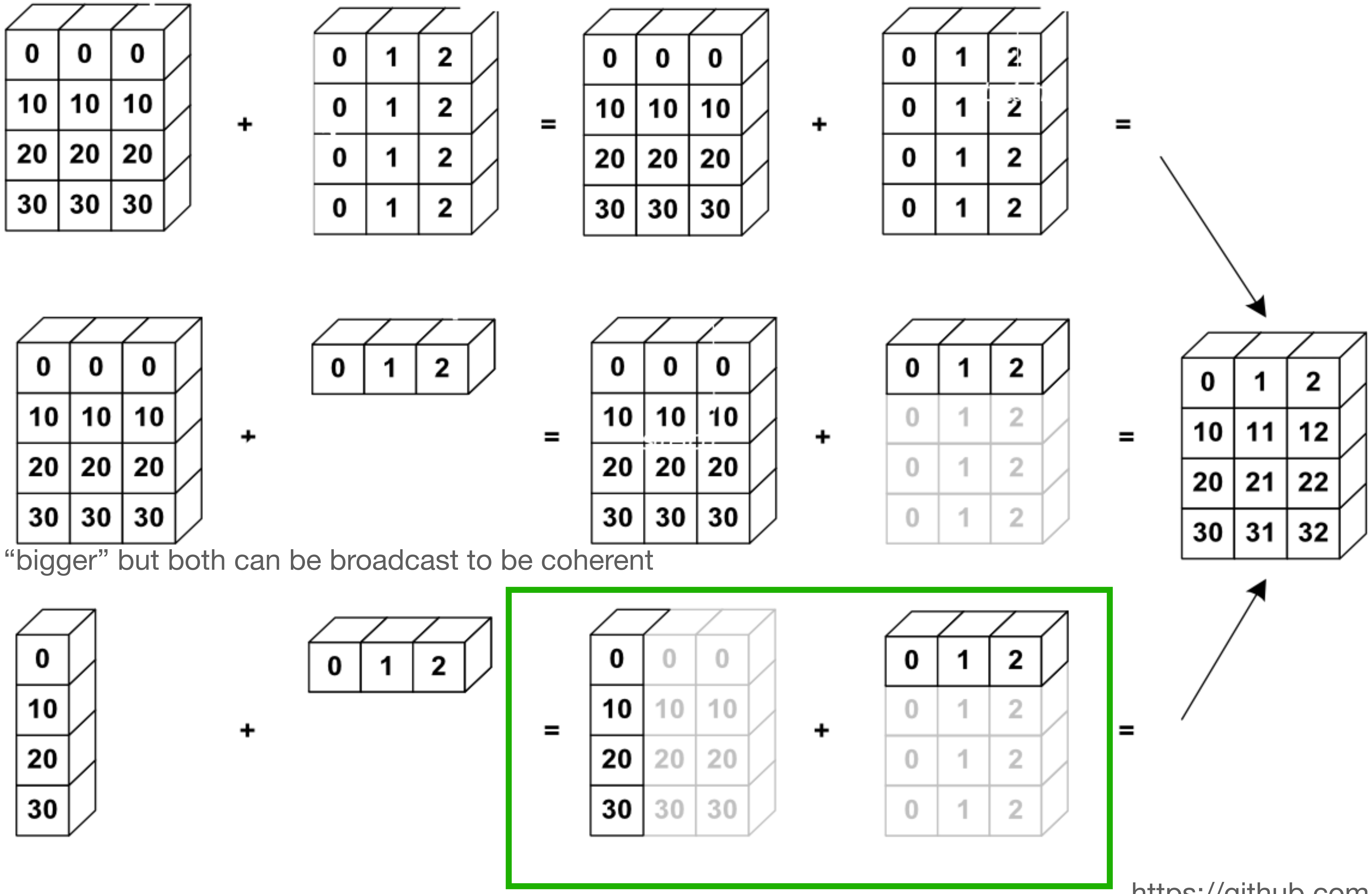
NumPy Broadcasting Example



Here neither shape is “bigger” but both can be broadcast to be coherent

Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

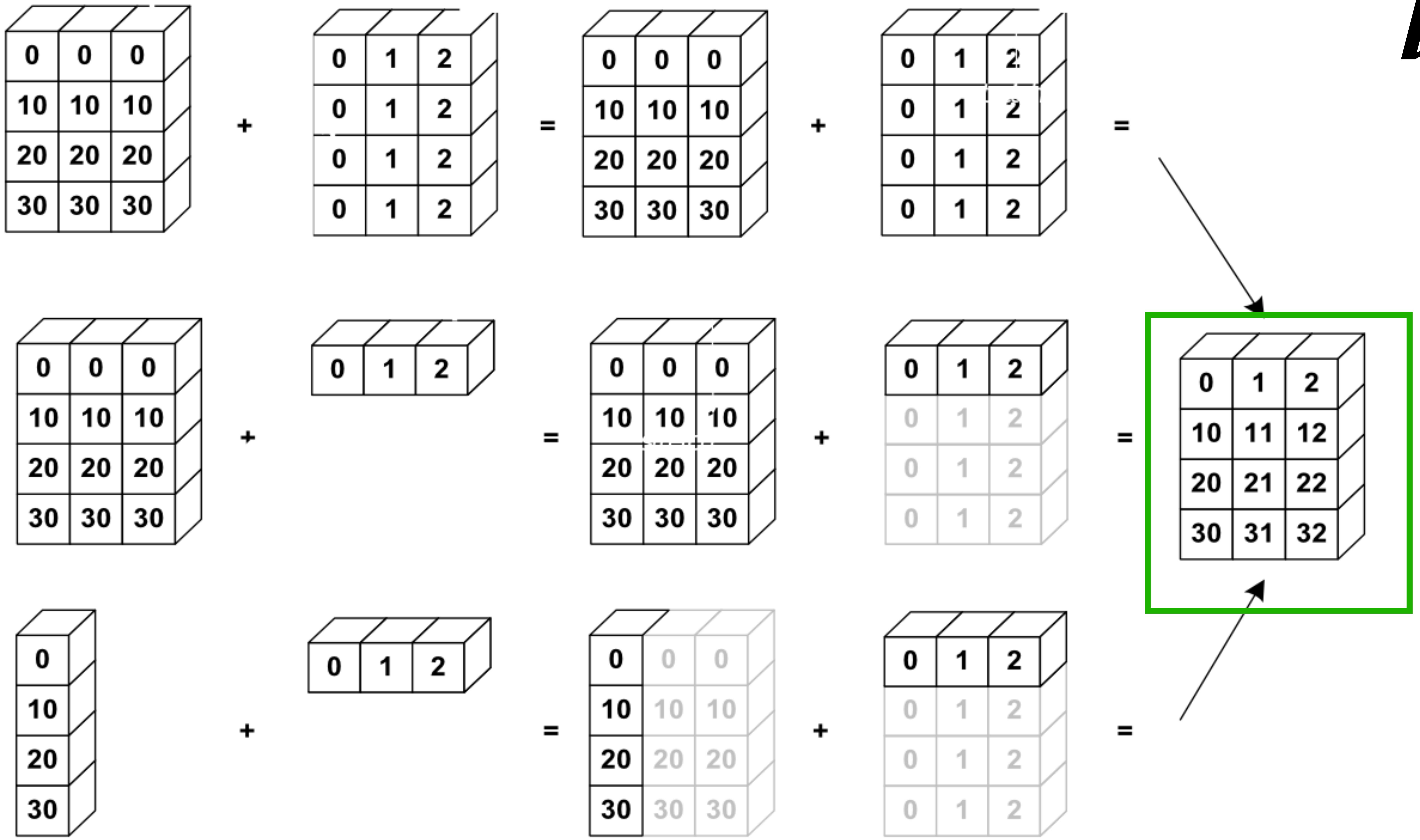
NumPy Broadcasting Example



Here neither shape is “bigger” but both can be broadcast to be coherent

Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy Broadcasting Example



All 3 give the same answer

Graphic from
https://github.com/scipy-lectures/scipy-lecture-notes

NumPy type checking

See https://numpy.org/devdocs/reference/typing.html

- Numpy has a number of helper functions for type checking
  - import np.typing as npt
    - npt.ArrayLike — Union of array-like objects
    - npt.dtype — can be used as output to change data type

NumPy *shape shifting*

ravel and reshape

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> np.ravel(x)
array([1, 2, 3, 4, 5, 6])
```

Un-ravel the (2, 3) array into a (6,) vector

Row-major order by default, like C

Can also use .reshape and scrap axis

```
>>> x.reshape(-1)
array([1, 2, 3, 4, 5, 6])
```

```
>>> np.ravel(x, order='F')
array([1, 4, 2, 5, 3, 6])
```

order='F' uses Fortran order, column-major

Other useful NumPy array operations

- See:
    - http://scipy-lectures.org/intro/numpy/operations.html
    - Code here: https://github.com/scipy-lectures/scipy-lecture-notes/blob/master/intro/numpy/operations.rst

Pandas

Built on top of and extended from numpy

- pd.Series — like a np.array
  - Can be used for vectors/columns of one type of feature across instances (usually one type or a general type)
    - eg. “count: [1, 12, 15, 0, 3]
    - dtype is an object type (np.float64, np.str, np.object)
  - Can also be used for vectors that represent instances/rows across multiple features (then usually mutiple types across these features/columns)
  - This about a pd.Series like a item in a dictionary “name: Vals”

Pandas

Built on top of and extended from numpy

	color	animal
row0	red	panda
row1	brown	fox
row2	green	Turtle

- pd.DataFrame — like a collection of pd.Series objects
  - Multiple columns are named series, for features
    - {“color”: [“red”, “brown”, “green”], “animal”: [“panda”, “fox”, “turtle”]}
  - Can also think about this as multiple rows for instances
    - [[“red”, “panda”], [“brown”, “fox”], [“green”, “turtle]]
  - In either case we get a rectangular DataFrame

Pandas

Built on top of and extended from numpy

	color	animal
row0	red	panda
row1	brown	fox
row2	green	Turtle

- pd.DataFrame — like a collection of pd.Series objects
  - Multiple columns are named series, for features
    - {“color”: [“red”, “brown”, “green”], “animal”: [“panda”, “fox”, “turtle”]}
  - Can also think about this as multiple rows for instances
    - [[“red”, “panda”], [“brown”, “fox”], [“green”, “turtle]]
  - In either case we get a rectangular DataFrame

Pandas

Built on top of and extended from numpy

	color	animal
row0	red	panda
row1	brown	fox
row2	green	Turtle

- pd.DataFrame — like a collection of pd.Series objects
  - Multiple columns are named series, for features
    - {“color”: [“red”, “brown”, “green”], “animal”: [“panda”, “fox”, “turtle”]}
  - Can also think about this as multiple rows for instances
    - [[“red”, “panda”], [“brown”, “fox”], [“green”, “turtle]]
  - In either case we get a rectangular DataFrame

Pandas

Built on top of and extended from numpy

	color	animal
row0	red	panda
row1	brown	fox
row2	green	Turtle

- pd.DataFrame — like a collection of pd.Series objects
    - Multiple columns are named series, for features
        - {“color”: [“red”, “brown”, “green”], “animal”: [“panda”, “fox”, “turtle”]}
    - Can also think about this as multiple rows for instances
        - [[“red”, “panda”], [“brown”, “fox”], [“green”, “turtle]]
    - In either case we get a rectangular DataFrame

Pandas

Subsetting columns, many flavors

TRY PASSING IN NAMES WHEN YOU CAN
INCREASES READABILITY (first 3)

- my_DataFrame[“col_name”]
  - my_DataFrame[[“col_name1”, “col_name2”]] (notice list of names as index)
- my_DataFrame.col_name
- my_DataFrame.loc[:, [“col_name1”, “col_name2”]]
- my_DataFrame.iloc[:, [1, 7, -1]] COLUMN NUMBERS (ints)
- Old documents will say you can pass in integers for column positions
  - Like: my_DataFrame[[1, 7, -1]]
  - This behavior was deprecated circa pandas 0.20

Pandas

Subsetting rows, many flavors

TRY PASSING IN NAMES WHEN YOU CAN
INCREASES READABILITY (loc)

- my_DataFrame.loc[0]
    - my_Data.loc[[0, 2, 8]]
    - my_Data.loc[[0, 2, 8], :]

ROW NAMES! Ints by default, but can change

- my_DataFrame.iloc[0]
    - my_DataFrame.iloc[[0, 2, -1]]
    - my_DataFrame.iloc[[0, 2, -1], :]

ROW NUMBERS (ints)

- Back in the day, you could use .ix to do either, this was confusing and has been deprecated circa pandas .20

More Pandas

See munging with pandas for more...

Matplotlib

See IntroSciPy.ipynb and SC:P3H4MJ book

Imperative plotting, Seaborn wrapper

Also useful: http://scipy-lectures.org/intro/matplotlib/index.html
with code here: https://github.com/scipy-lectures/scipy-lecture-notes/blob/master/intro/matplotlib/index.rst