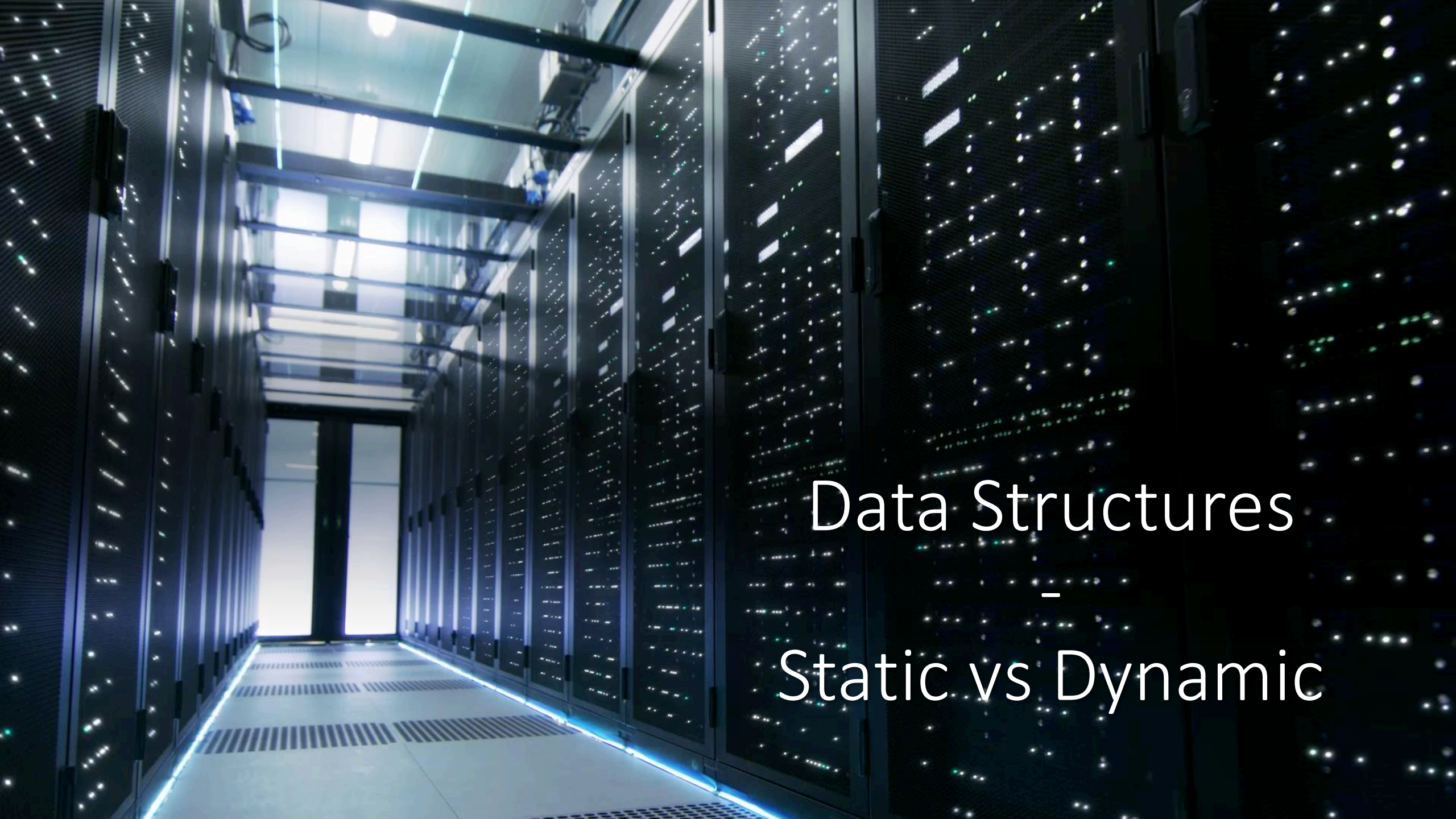


Data Structures

-

Static vs Dynamic

What is a “data structure”?

“…As the term is used more precisely in computer science, a *data structure* is:

- a collection of data values,
- the relationships among them, and
- the functions or operations that can be applied to the data.

If any one of these three characteristics is missing or not stated precisely, the structure being examined does not qualify as a data structure.”

“*Encyclopedia of Computer Science:*” https://dl.acm.org/doi/10.5555/1074100.1074312

Some Python data structures
(we have *already encountered)*

- Lists
  - "Mutable"
  - Size/content can be updated *after* initialization
    - `a = [1, 'two', 3.0, 4, 'five']`
    - `a[0]; a[0:3]`
    - `a.append('6'); a.pop()`

```
1
[1, 'two', 3.0]

[1, 'two', 3.0, 4, 'five', '6']
'6'
```

- Arrays
  - Mutable
  - Its objects are of the same, *fixed* type
    - `a = array.array('i', range(1,6))`
    - `a[0]; a[0:3]`
    - `a.append(6); a.pop()`

```
array('i', [1, 2, 3, 4, 5])

1
array('i', [1, 2, 3])

array('i', [1, 2, 3, 4, 5, 6])
6
```

- Tuples
  - *Immutable*
  - Allow different types
    - `a = (1, 'two', 3.0, 4, 'five')`
    - `a[0], a[0:3]`

```
1
(1, 'two', 3.0)
```

Static vs dynamic data structures

- *Fixed* vs *variable* size

- Pros and cons:

Dynamic **(Memory allocated when required, at run-time)**	**Static** **(Memory allocated when program is written, and it thus has a fixed size)**
Efficient use of memory (as much as "really" needed)	
Easier to insert/remove data	
We don't need to know size in advance	
	Memory usage can be wasteful
	Inserting/deleting elements is generally inefficient (part of array needs to be rewritten)
	Array size must be determined *before* use

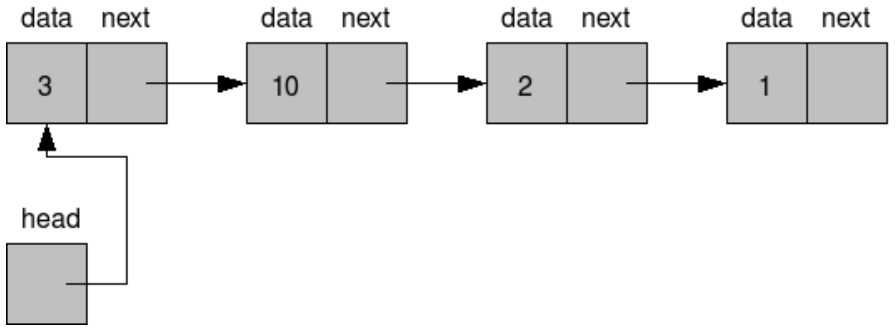
Static vs dynamic data structures

- *Fixed* vs *variable* size

- Pros and cons:

Dynamic **(Memory allocated when required, at run-time)**	**Static** **(Memory allocated when program is written, and it thus has a fixed size)**
Efficient use of memory (as much as "really" needed)	Memory allocated in advance, no risk of "running out"
Easier to insert/remove data	No need to worry about keeping track of size at run-time
We don't need to know size in advance	Random access is easier
Run the risk of "overflow" at run-time	Memory usage can be wasteful
Harder to implement (keep track of various data structure attributes, pointers, etc)	Inserting/deleting elements is generally inefficient (part of array needs to be rewritten)
Data can be "orphaned", filling up memory with "garbage"	Array size must be determined *before* use

Dynamic data structures: this course



- Linked Lists
- Trees