Sequence Represented As Two Stacks
Client View

• A situation as seen by a client of the kernel class `Sequence3`, based on its interface:

<4, 5, 6, 7>
Implementer View

• The same situation as seen by the kernel class implementer, where the data representation is two Stacks called left and right:
Implementer View

The same situation as seen by the kernel class implementer, where the data representation is two Stacks called `left` and `right`:

But this shows *just one* possible data representation for the same `Sequence` value!
Implementer View

• The same situation as seen by the kernel class implementer, where the data representation is two Stacks called `left` and `right`:
Implementer View

• The same situation as seen by the kernel class implementer, where the data representation is two Stacks called left and right:
Implementer View

The same situation as seen by the kernel class implementer, where the data representation is two stacks called left and right:

What? Yes, left has 5 on the top of the Stack, not 4. That is, these entries of the Sequence are in reverse order from those in the Stack called left!
Implementer View

- The same situation as seen by the kernel class implementer, where the data representation is two Stacks called **left** and **right**: 
Implementer View

- The same situation as seen by the kernel class implementer, where the data representation is two Stacks called `left` and `right`:

```
<7, 6, 5, 4>
```

```
<>
```
Picture: **Sequence**\textsubscript{3} on **Stack**

What the client sees: a Sequence whose value is \langle 4, 5, 6, 7 \rangle.
What the implementer might see: two Stacks whose values are <6, 5, 4> and <7>.
A Better Picture

(\langle4\rangle, \langle5, 6, 7\rangle)

(\langle4, 5, 6, 7\rangle)

(\langle7, 6, 5, 4\rangle, \langle\rangle)

(\langle7, 6, 5, 4\rangle, \langle\rangle)

(\langle6, 5, 4\rangle, \langle7\rangle)

(\langle5, 4\rangle, \langle6, 7\rangle)
A Better Picture

Client view: a Sequence.
A Better Picture

Implementer view: two Stacks (several possibilities).
A Better Picture

Notation: each data representation is an ordered pair of strings.
The Interpretation

- To get the Sequence that is represented by the two Stacks, do the following:
  - Start with the reverse of left, e.g.:
    
    \[
    \text{left} = <5, 4> \\
    \text{rev}(\text{left}) = <4, 5>
    \]
The Interpretation

- To get the `Sequence` that is represented by the two `Stack`s, do the following:
  - Start with the reverse of `left`, e.g.:
    
    \[
    \text{left} = <5, 4> \\
    \text{rev}(\text{left}) = <4, 5>
    \]

Remember, a `Sequence` is modeled as a `string`. 
The Interpretation

• To get the Sequence that is represented by the two Stacks, do the following:
  – Start with the reverse of left, e.g.:

\[
\text{left} = \langle 5, 4 \rangle \\
\text{rev}(\text{left}) = \langle 4, 5 \rangle
\]

Remember, a Stack is modeled as a string.
The Interpretation

• To get the Sequence that is represented by the two Stacks, do the following:
  – Start with the reverse of left, e.g.:
    \[
    \text{left} = <5, 4>
    \]
    \[
    \text{rev(left)} = <4, 5>
    \]
  – Concatenate right to it (on the right), e.g.:
    \[
    \text{rev(left)} \ast \text{right} = <4, 5> \ast <6, 7>
    = <4, 5, 6, 7>
    \]
The Interpretation

• To get the Sequence that is represented by the two Stacks, do the following:
  – Start with the reverse of left, e.g.:
    \[
    \text{left} = <5, 4> \\
    \text{rev(left)} = <4, 5>
    \]
  – Concatenate right to it:
    \[
    \text{rev(left)} \ast \text{right} = <4, 5> \ast <6, 7> \\
    = <4, 5, 6, 7>
    \]

This is the Sequence value represented by the two Stacks left and right.
Why Reverse Anything?

• To simplify the code for Sequence3, and for performance reasons, it is helpful to have the tops of the two Stacks “facing each other” in the middle of the Sequence.

• Since it is *up to us as implementers* how to *interpret* the two Stacks as a Sequence, this is how we choose to do it!
Why Reverse Anything?

- To simplify the code for `Sequence3`, and for performance reasons, it is helpful to have the tops of the stacks "facing each other" in the middle of the `Sequence`.
- Since it is up to us as implementers how to interpret the two `Stacks` as a `Sequence`, this is how we choose to do it!

A critical point to understand: there is no code in `Sequence3` that reverses a `Stack`!
Why Reverse Anything?

• To simplify the code for `Sequence3`, and for performance reasons, it is helpful to have the tops of the two Stacks "facing each other" in the middle of the Sequence.

• Since it is up to us as implementers how to interpret the two Stacks as a Sequence, this is how we choose to do it!

The reversal is simply part of the **interpretation** of the data representation—a **mental computation**!