

Hornpipe: State Management for Asynchronous Applications

Sam Sartor

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

<https://gitlab.com/samsartor/hornpipe>

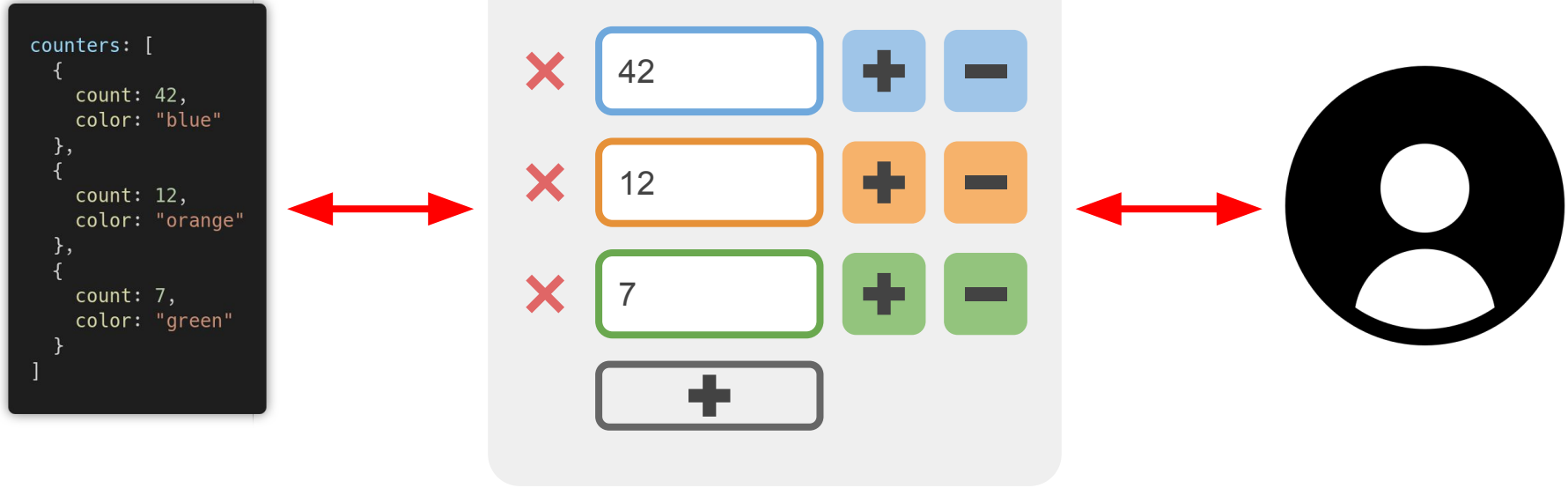
Stateless Applications

- Compilers
- Command-line Utilities
- Batch Analysis
- School Assignments

Stateful Applications

- Web Servers
- Industrial Control Systems
- Operating Systems
- Embedded Devices
- Video Games
- Streaming Data Analysis
- *User Interfaces*

User Interface

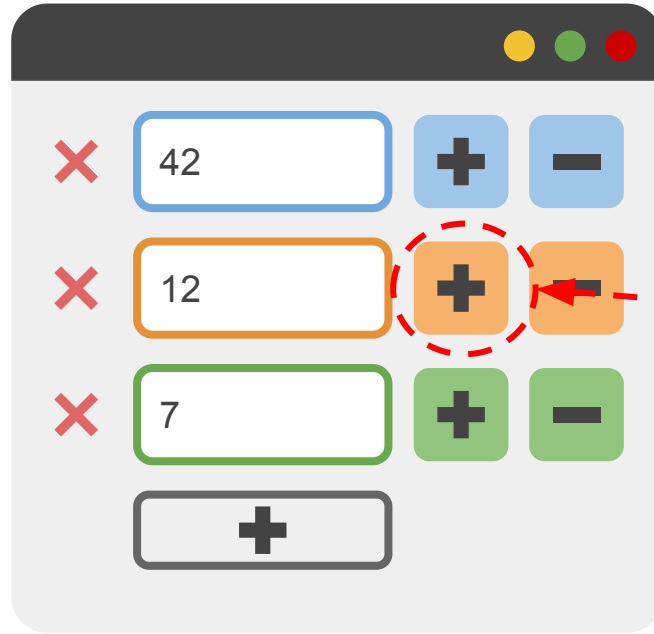


Problems With State

1. Mutation
2. Deletion
3. Dependency

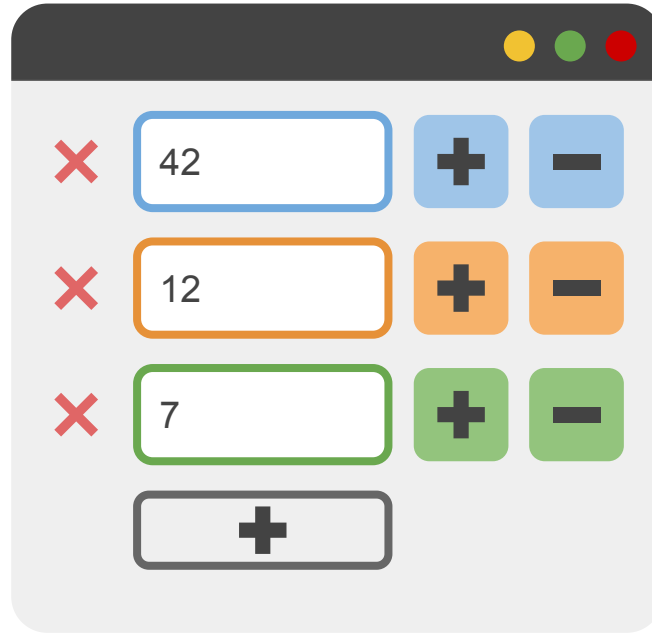
#1 - Mutation

```
counters: [  
  {  
    count: 42,  
    color: "blue"  
  },  
  {  
    count: 12,  
    color: "orange"  
  },  
  {  
    count: 7,  
    color: "green"  
  }  
]
```



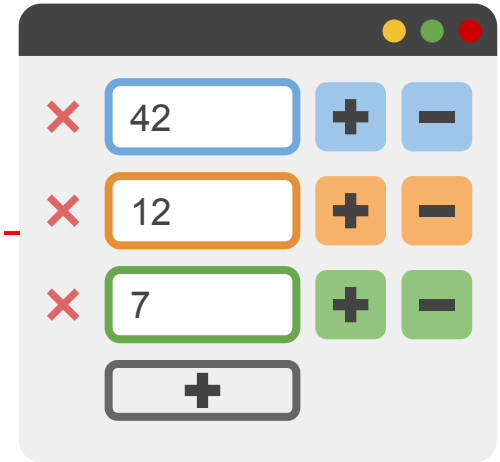
#1 - Mutation

```
counters: [  
  {  
    count: 42,  
    color: "blue"  
  },  
  {  
    count: 13,  
    color: "orange"  
  },  
  {  
    count: 7,  
    color: "green"  
  }  
]
```



Mutation - C

```
void on_click_add(counter* obj) {  
    int current_count = obj->count;  
    int new_count = current_count + 1;  
    obj->count = new_count;  
}
```



Mutation

- Mutexes
 - C++
 - Java
- Channels
 - Rust
 - Go
- Single-threaded Execution
 - Python
 - JS

Mutation – Databases

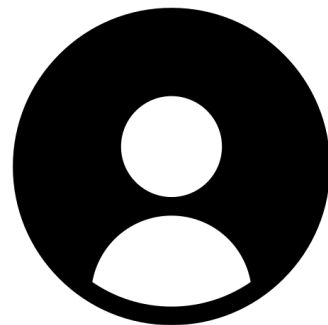
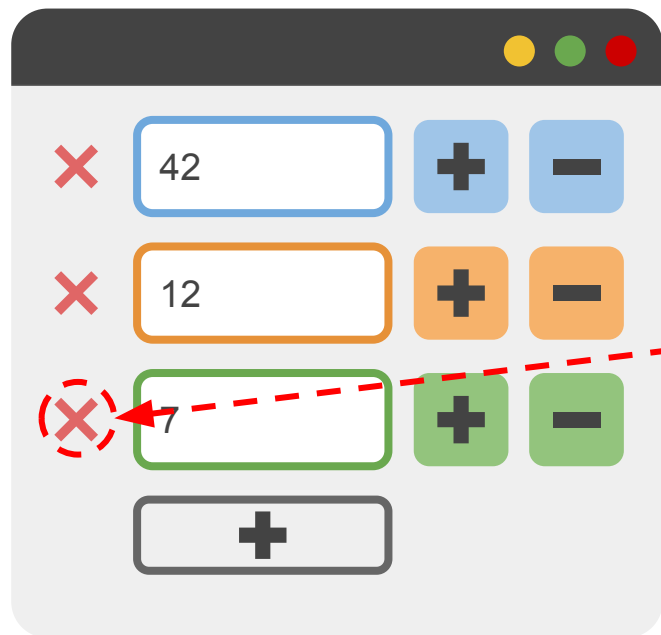
- Serializable Transactions
 - *MVCC!*

```
BEGIN TRANSACTION;  
UPDATE counters  
    SET @count = @count + 1;  
COMMIT;
```

```
BEGIN TRANSACTION;  
UPDATE counters  
    SET @count = @count + 1;  
COMMIT;
```

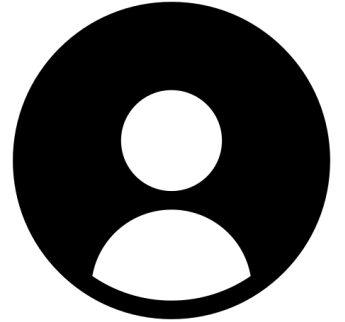
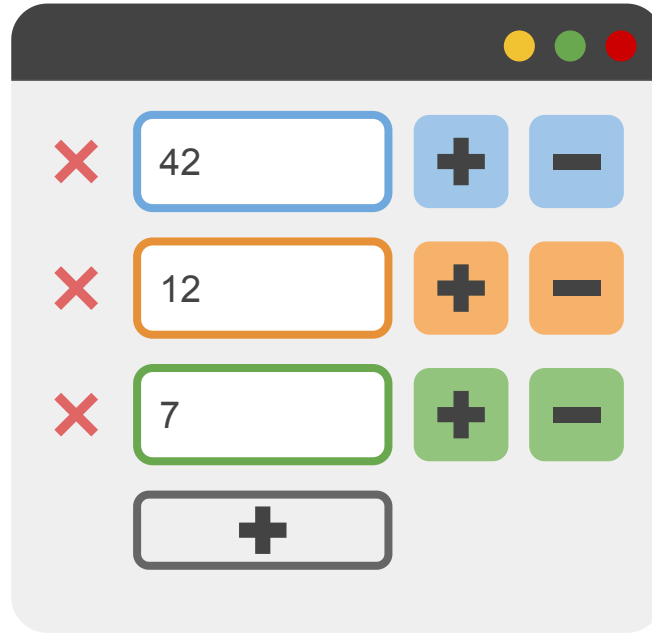
#2 - Deletion

```
counters: [  
  {  
    count: 42,  
    color: "blue"  
  },  
  {  
    count: 12,  
    color: "orange"  
  },  
  {  
    count: 7,  
    color: "green"  
  }  
]
```



#2 - Deletion

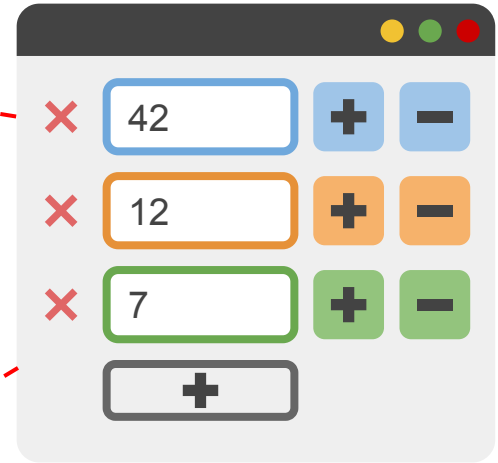
```
counters: [  
  {  
    count: 42,  
    color: "blue"  
  },  
  {  
    count: 12,  
    color: "orange"  
  },  
]
```



Deletion - C

```
void on_click_x(counter* obj) {  
    free(obj);  
}
```

```
void on_render_frame(counter* obj) {  
    obj->render_function(obj);  
}
```

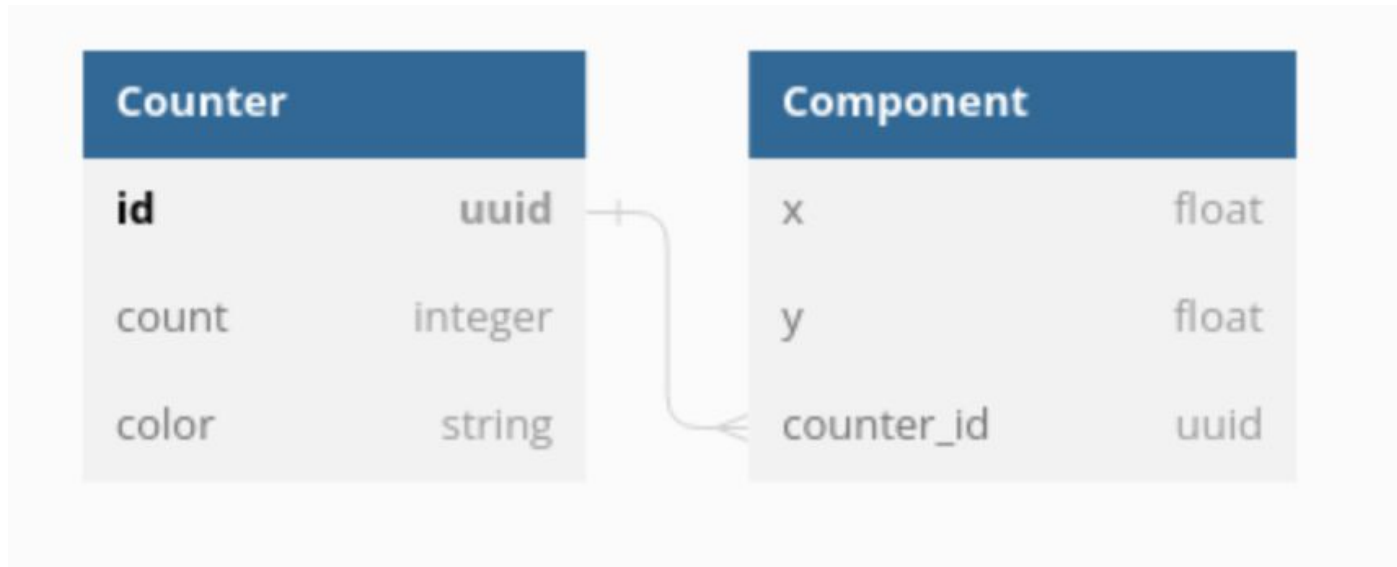


Deletion

- Reference Counting
 - Rust
 - C++
 - Python
- Garbage Collection
 - Java
 - JS
 - Go

Deletion – Databases

- *Weak References!*

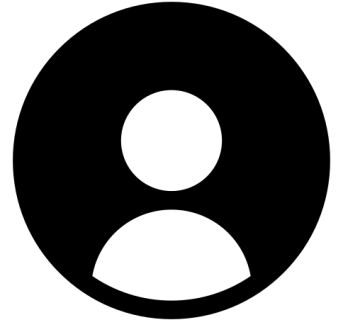
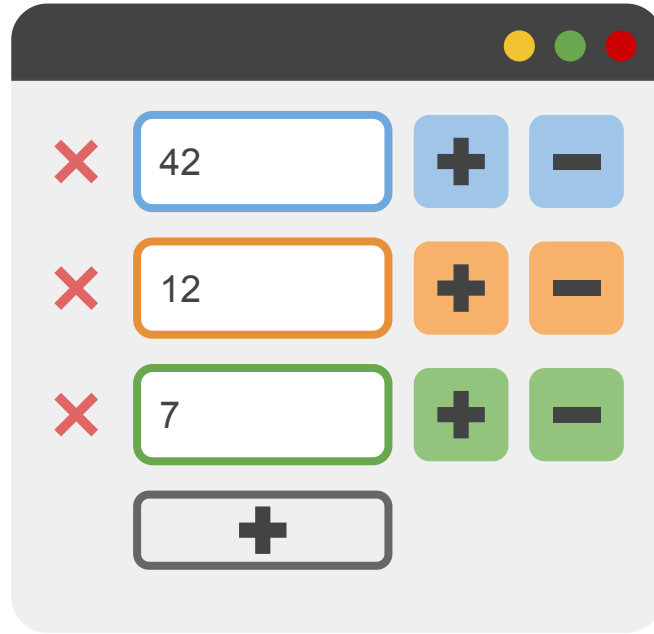


Deletion & Mutation

```
struct List {  
    int* pointer;  
    int length;  
};  
  
void append(List* my_list, int value) {  
    int length = my_list->length;  
  
    int* new_pointer = malloc(  
        (length+1) * sizeof(int)  
    );  
  
    memcpy(  
        new_pointer,  
        my_list->pointer,  
        length * sizeof(int)  
    )  
  
    free(my_list->pointer);  
    my_list->pointer = new_pointer;  
}
```

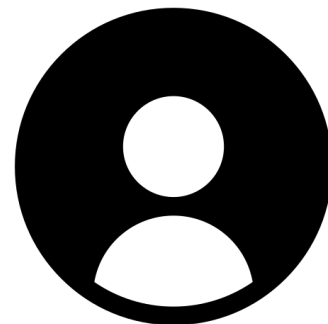
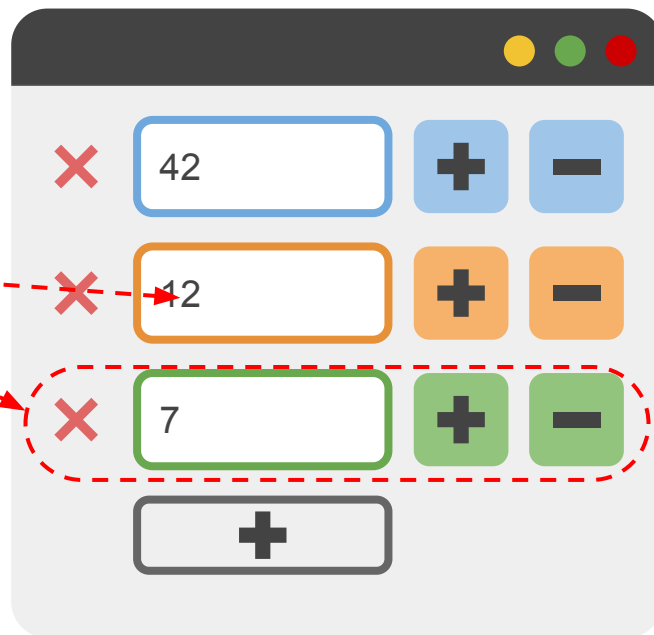
#3 - Dependency

```
counters: [  
  {  
    count: 42,  
    color: "blue"  
  },  
  {  
    count: 12,  
    color: "orange"  
  },  
  {  
    count: 7,  
    color: "green"  
  }  
]
```



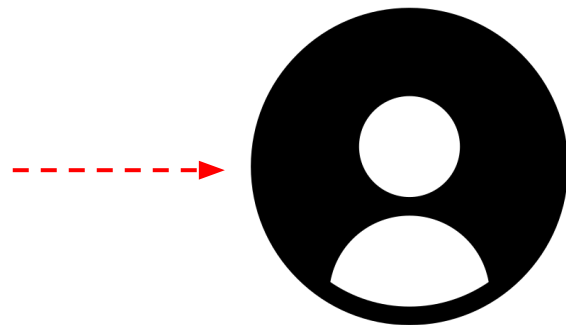
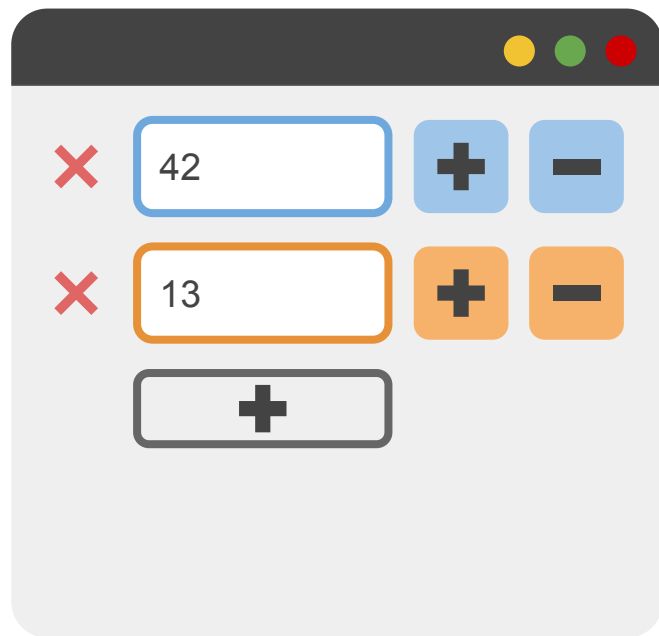
#3 - Dependency

```
counters: [
  {
    count: 42,
    color: "blue",
  },
  {
    count: 13,
    color: "orange",
  },
]
```



#3 - Dependency

```
counters: [  
  {  
    count: 42,  
    color: "blue",  
  },  
  {  
    count: 13,  
    color: "orange",  
  },  
]
```



Dependency

- Callback
 - C++
 - Java
 - Python
- Event Bus
 - Go
 - Rust
- *Reactivity!*
 - JS (React, RxJS, Vue.js)

Reactivity

```
<span id="count"></span>  
<button onClick="add()">  
  Increment  
</button>
```

```
let count = 0;  
$('#count').html(count);  
function add() {  
  count += 1;  
  $('#count').html(count);  
}
```

```
let count = 0;
```

```
<span>{ count }</span>  
<button onClick={ count += 1 }>  
  Increment  
</button>
```



Business Report ☆ 📁 ☁



Share



fx

	A	B	C	D	E	F	G	H
1	Business Report							
2								
3								
4								
5								
6								
7								
8								
9		Helen						
10								
11								
12								
13								
14								
15								
16								
17								



Sheet1

Sheet2

Sheet3



Dependency - Databases

- Event Stream
 - Kafka
- Pub/sub
 - Postgres
 - Redis
- Dataflow
 - Google/Microsoft Cloud

Hornpipe

Hornpipe is a state-management framework for the Rust programming language.

Hornpipe Provides

- Multiversion Concurrency Control
 - Deadlock-free
 - Applications can be multiplayer
 - First-class undo/redo
- Weak references
 - First-class deletion
 - Objects have ownership
- Reactivity
 - Modifications automatically propagate

Reactivity & Weak Ref & MVCC

- Reference becomes NULL
→ Notify dependents!
- Transaction aborts
→ Notify dependents!

```

#[hornpipe]
struct Figure {
    image: [u8],
    #[compute]
    mut title: Text,
    mut figure_number: u32,
    mut subfigures: [&Figure],
}

impl Figure {
    fn new(url: Url, n: u32) -> Figure {
        let bytes = url.download();
        Figure::Builder {
            image: bytes,
            figure_number: n,
            subfigures: List::new(),
        }.build()
    }

    fn title(self, tx: Tx) -> Text {
        let n = self.figure_number.get(tx);
        Text::new("Figure {n}")
    }
}

```

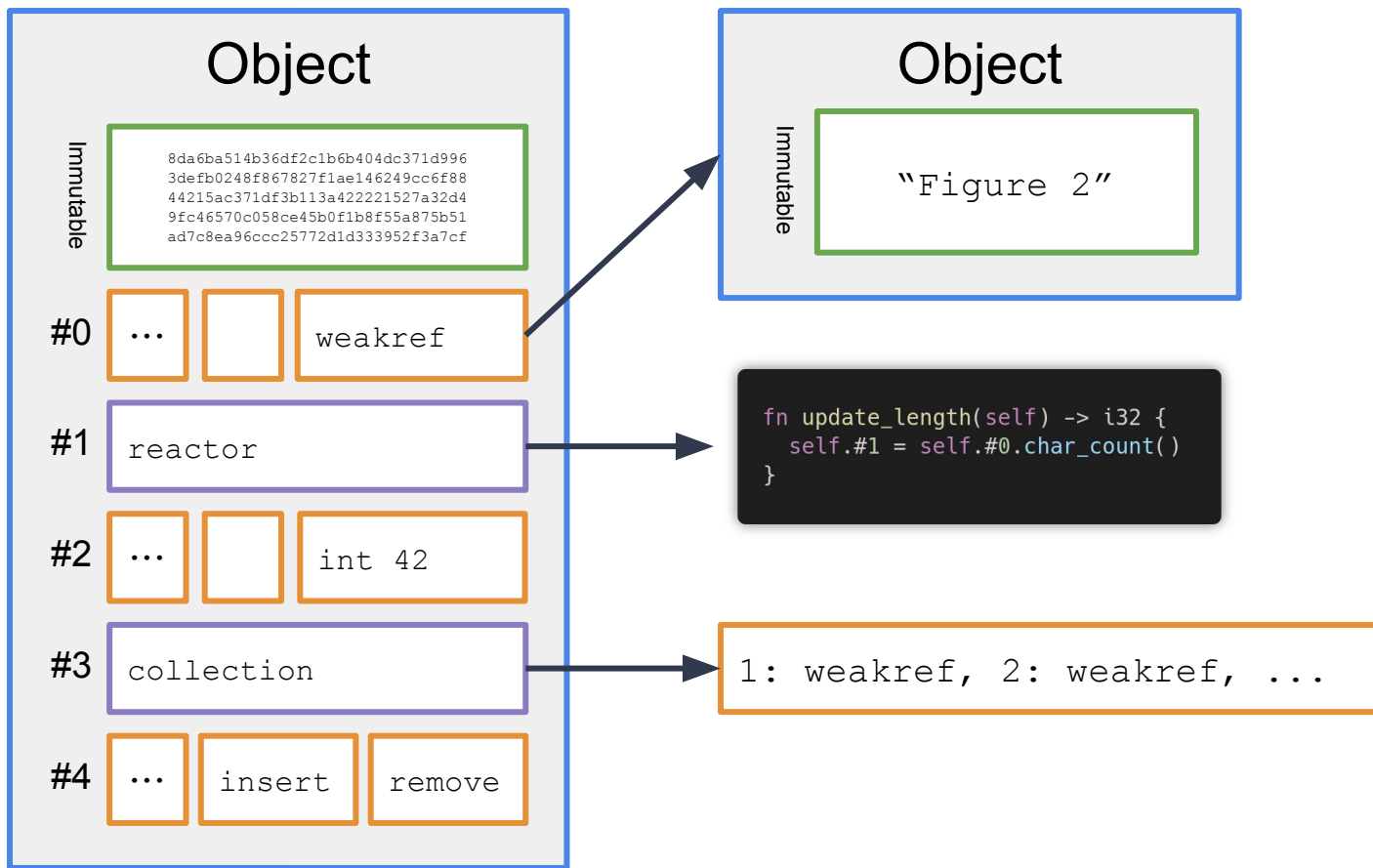
```

let tx: Tx = Tx::start();
let fig: Figure = ...;

render(rsx! {
    image {
        data: fig.image.get(tx),
    },
    text_input {
        width: fig.title_width.get(tx),
        value: fig.title.get(tx),
        keypress: |evt: Press<UpArrow>| {
            let n = fig.figure_number.get(tx);
            fig.figure_number.set(tx, n - 1);
        },
        keypress: |evt: Press<DownArrow>| {
            let n = fig.figure_number.get(tx);
            fig.figure_number.set(tx, n + 1);
        },
    },
});

tx.commit();

```



Demo

Write Slot

- Writing a variable x at time t
- Insert a message into x 's slot
 - Sort messages by transaction order
- DFS over all dependents of x
 - Clear “clean” flag
 - Save any callbacks to run on commit

Read Slot

- Accessing variable x at time t
- If x is not known to be “clean” after t :
 - DFS dirty dependencies of x
 - Mark each as clean after t
 - Place all dependencies in post-order
 - Run each associated reactor
 - Break out if reaction no longer needed
- Provide the latest value of x

Transaction Ordering

- Transactions given a total order on creation
 - Can be arbitrary
- Ignore *uncommitted messages* or messages from younger transactions
- If an ignored message turns out to be from an older transaction: conflict!

Future Wok

- Higher level interface!
 - Declare composite types
 - Autogenerated getters/setters
 - Simpler collection types (List, Map, Set)
- “Fixups” - resolve conflict rather than abort
- More complete testing
- Network connections & multiplayer
- More GUI functionality (e.g. redo/undo)

Questions?