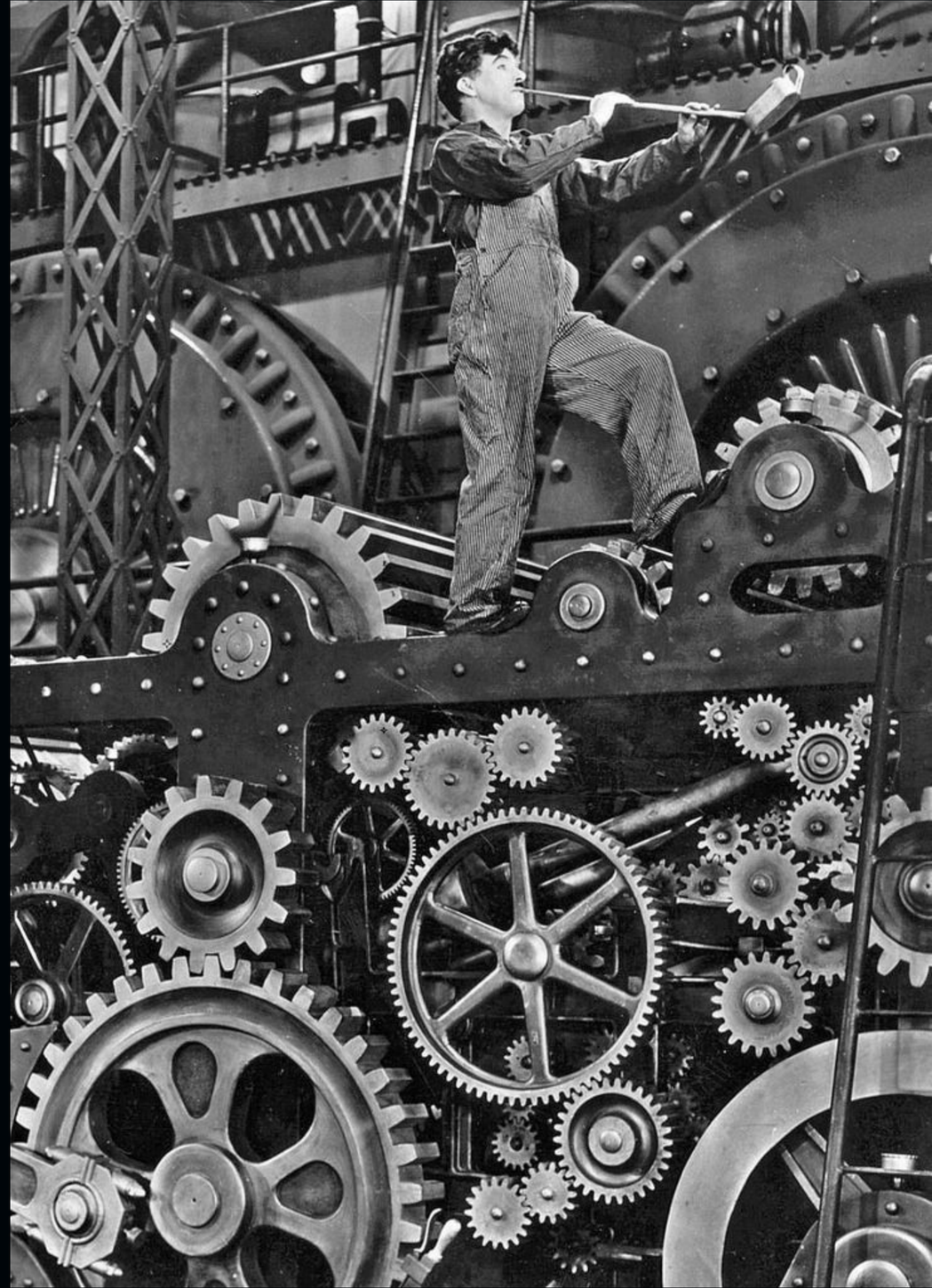
COMPOSABLE COMPONENTS

MARKUS SCHLEGEL
ACTIVE GROUP GMBH



COMPOSITION

REUSE

~~COMPOSITION~~

~~REUSE~~

COMPOSABILITY

REUSABILITY

MOVEMENT OF CODE

«[We have to regard] every developed form as in fluid movement, and therefore [take] into account its transient nature not less than its momentary existence.»

– KARL MARX

MOTIVATION
COMPOSABILITY

```
function inc (x) {return x + 1}
```

```
  function inc (x) {return x + 1}

+ function incTuple ([x, y]) {
+   return [inc(x), inc(y)]
+ }
```

```
  function inc (x) {return x + 1}

  function incTuple ([x, y]) {
    return [inc(x), inc(y)]
  }

+ function incTupleTuple ([xs, ys]) {
+   return [incTuple(xs), incTuple(ys)]
+ }
```

Composition of functions:
Combine small functions to build large functions

Composition of UI components:
Combine small components to build large components

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

Counter becomes a *Controlled Component*

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

Counter becomes a *Controlled Component*

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

Counter becomes a *Controlled Component*

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

TwoCounters becomes *Controlled Component*

```
  function Counter ({value, onChange}) {
    // [state, setState] = useState(0);
    return (<button
              onclick={onChange(value + 1)}>
              {value}
            </button>)}

#  function TwoCounters ({value, onChange}) {
#    // [val, setVal] = useState([0,0]);
#    return (
#      <div>
#        <Counter
#          value={value[0]}
#          onChange={xx => onChange([xx,val[1]])} />
#        <Counter
#          value={value[1]}
#          onChange={yy => onChange([val[0], yy])} />
#        Insgesamt: {sum(value)}
       </div>);}

+  function TwoTwoCounters () {
+    [val, setVal] = useState([[0,0],[0,0]]);
+    return (
+      <div>
+        <TwoCounters
+          value={val[0]}
+          onChange={xx => setVal([xx, val[1]])} />
+        <TwoCounters
+          value={val[1]}
+          onChange={yy => setVal([val[0], yy])} />
+        val.toString()
+      </div>);}
```

```
function Counter () {
  [state, setState] = useState(0);
  return (<button
            onclick={setState(state + 1)}>
            {state}
          </button>)}
```

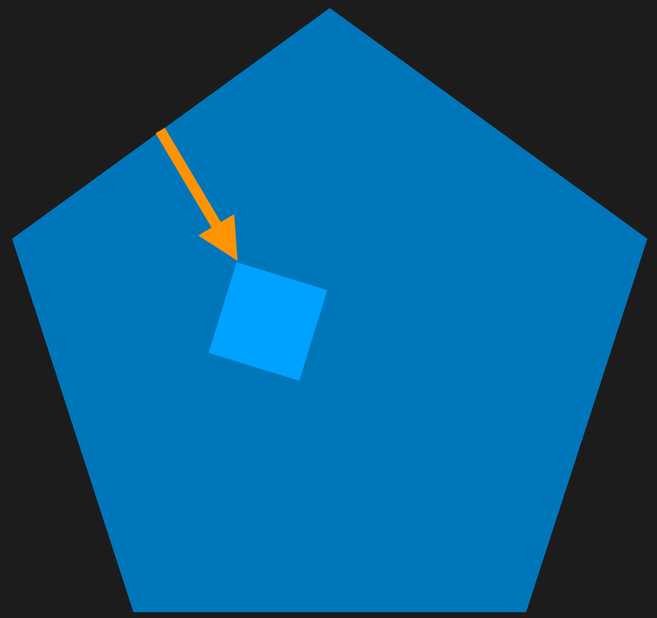
Counter becomes a *Controlled Component*

```
#  function Counter ({value, onChange}) {
#    // [state, setState] = useState(0);
#    return (<button
#              onclick={onChange(value + 1)}>
#              {value}
             </button>)}

+ function TwoCounters () {
+   [state, setState] = useState([0,0]);
+   return (
+     <div>
+       <Counter
+         value={state[0]}
+         onChange={xx => setState([xx,state[1]])} />
+       <Counter
+         value={state[1]}
+         onChange={yy => setState([state[0], yy])} />
+       Insgesamt: {sum(state)}
+     </div>);}
```

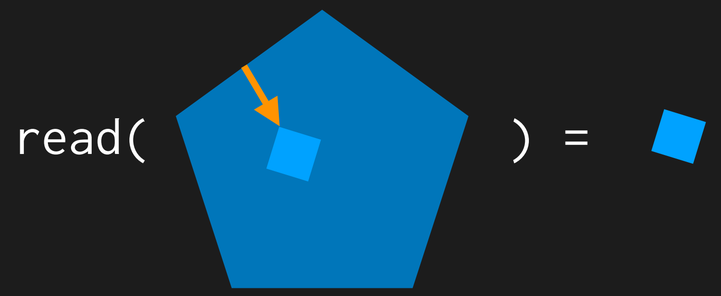
TwoCounters becomes *Controlled Component*

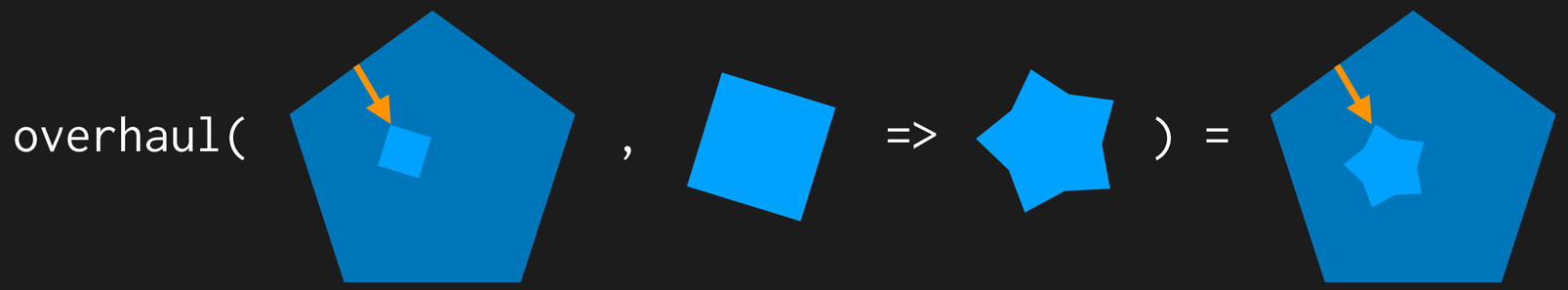
```
  function Counter ({value, onChange}) {
    // [state, setState] = useState(0);
    return (<button
              onclick={onChange(value + 1)}>
              {value}
            </button>)}

#  function TwoCounters ({value, onChange}) {
#    // [val, setVal] = useState([0,0]);
#    return (
#      <div>
#        <Counter
#          value={value[0]}
#          onChange={xx => onChange([xx,val[1]])} />
#        <Counter
#          value={value[1]}
#          onChange={yy => onChange([val[0], yy])} />
#        Insgesamt: {sum(value)}
       </div>);}

+  function TwoTwoCounters () {
+    [val, setVal] = useState([[0,0],[0,0]]);
+    return (
+      <div>
+        <TwoCounters
+          value={val[0]}
+          onChange={xx => setVal([xx, val[1]])} />
+        <TwoCounters
+          value={val[1]}
+          onChange={yy => setVal([val[0], yy])} />
+        val.toString()
+      </div>);}
```

LENSES & STORES
COMPOSABLE READING AND WRITING

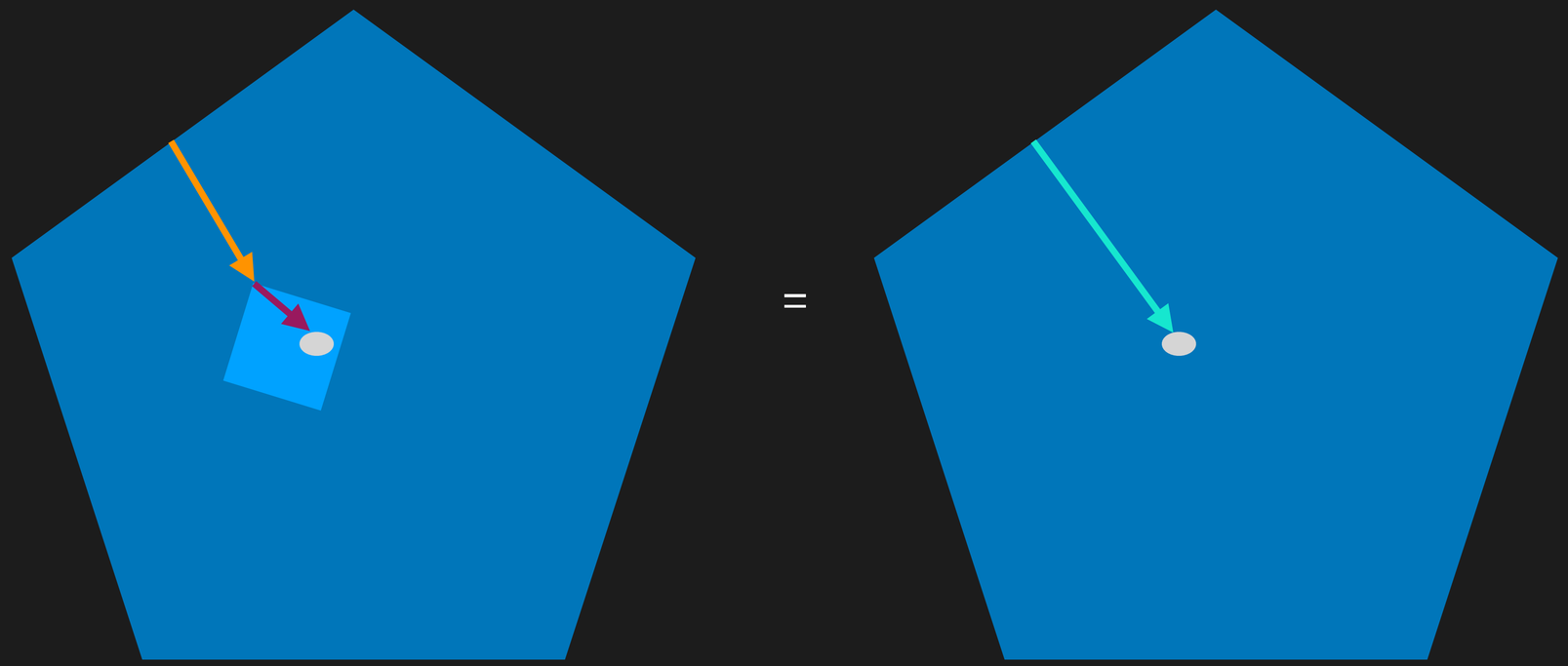


LENSES & STORES
COMPOSABLE READING AND WRITING





LENSES & STORES
COMPOSABLE READING AND WRITING



focus( , ) = 

```
function Counter ({store}) {
  return (<button
            onclick=
            {overhaul(store, x => x + 1)}>
            read(store)
          </button>);}
```

```
function Counter ({store}) {
  return (<button
            onclick=
            {overhaul(store, x => x + 1)}>
            read(store)
          </button>);}
```

read(store)

```
function Counter ({store}) {
  return (<button
            onclick=
            {overhaul(store, x => x + 1)}>
            read(store)
          </button>);}
```

read(store)

overhaul(store, oldValueToNewValue)

```
function Counter ({store}) {
  return (<button
            onclick=
            {overhaul(store, x => x + 1)}>
            read(store)
          </button>);}
```

```
  function Counter ({store}) {
    return (<button
              onclick=
              {overhaul(store, x => x + 1)}>
              read(store)
            </button>);}

+ function TwoCounters ({store}) {
+   return (
+     <div>
+       <Counter
+         store=focus(store, L.idx(0)) />
+       <Counter
+         store=focus(store, L.idx(1)) />
+       Insgesamt: {summe(read(store))}
+     </div>);}
```

read(store)

overhaul(store, oldValueToNewValue)

```
function Counter ({store}) {
  return (<button
            onclick=
            {overhaul(store, x => x + 1)}>
            read(store)
          </button>);}
```

```
  function Counter ({store}) {
    return (<button
              onclick=
              {overhaul(store, x => x + 1)}>
              read(store)
            </button>);}

+ function TwoCounters ({store}) {
+   return (
+     <div>
+       <Counter
+         store=focus(store, L.idx(0)) />
+       <Counter
+         store=focus(store, L.idx(1)) />
+       Insgesamt: {summe(read(store))}
+     </div>);}
```

read(store)

overhaul(store, oldValueToNewValue)

focus(store, lens) => newStore

```
function Counter ({store}) {
  return (<button
            onclick=
            {overhaul(store, x => x + 1)}>
            read(store)
          </button>);}
```

```
  function Counter ({store}) {
    return (<button
              onclick=
              {overhaul(store, x => x + 1)}>
              read(store)
            </button>);}

+ function TwoCounters ({store}) {
+   return (
+     <div>
+       <Counter
+         store=focus(store, L.idx(0)) />
+       <Counter
+         store=focus(store, L.idx(1)) />
+       Insgesamt: {summe(read(store))}
+     </div>);}
```

```
  function Counter ({store}) {
    return (<button
              onclick=
              {overhaul(store, x => x + 1)}>
              value
            </button>);}

  function TwoCounters ({store}) {
    return (
      <div>
        <Counter
          store=focusOnIndex(store, 0) />
        <Counter
          store=focusOnIndex(store, 1) />
        Insgesamt: {summe(read(store))}
      </div>);}

+ function TwoTwoCounters ({store}) {
+   return (
+     <div>
+       <TwoCounters
+         store=focus(store, L.idx(0)) />
+       <TwoCounters
+         store=focus(store, L.idx(1)) />
+       read(store).toString()
+     </div>);}
```

read(store)

overhaul(store, oldValueToNewValue)

focus(store, lens) => newStore

IN SUMMARY
COMPOSABLE COMPONENTS

1. Keep an eye on the movement of code
2. Composability allows for lossless reusability
3. Controlled Components are composable and reusable but cumbersome and veiled.
4. Lenses and stores allow for lossless composability without the veil

★ Next steps

- ☐ Components as tangible values
  https://www.youtube.com/watch?v=faJ8N0giqzw
- ☐ A lens library in JS (not an endorsement tho)
  https://github.com/calmm-js/partial.lenses
- ☐ reacl-c
  https://github.com/active-group/reacl-c ClojureScript