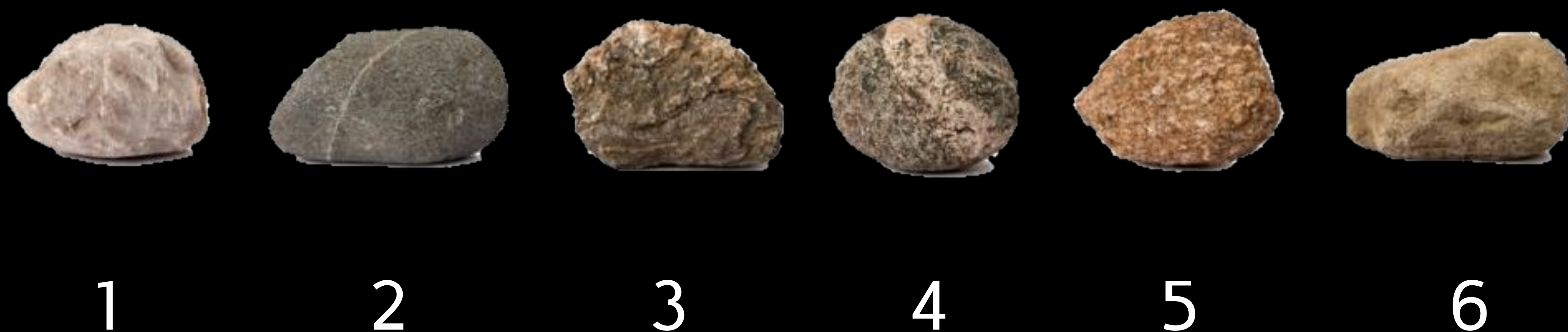
All I needed for FP I learned in High School Algebra

Eric Normand

PurelyFunctional.tv


1
2
3
4
5
6

LOG OF JENNY 2

From LORD HOWE **to** RAPA ITI **Date** TUES 16/4 **19**63

HOUR	LOG	DISTANCE MADE GOOD	COURSE	WIND DIRECTION AND FORCE	BAROMETER	LEEWAY	REMARKS
A.M. 2 1	299.3	5.0	T56	SW.F 5-6	29.9	NIL	HEAVIEY SEAS But havent shipped
3 2	305.	5.7	T56	S.W.F 5-6	29.9	NIL.	One yet as we are running before
3 3	308	3.0	T56	S.W. 5-6	29.9	NIL	Which corresponds with our Course
3 4	311	3.0	T56	S.W. 5-6	29.9	NIL	20 Knots of Breaking our First 100
3 5	319.5	8.5	T56	SW 5-6	29.9	NIL	Very uncomfutbell down below
3 6	330	8.5	T56	SW 8.	29.9	NIL.	decks.
3 7	337.4	7.4	T56	SW 7	29.95	NIL	SEAS HEAVY HUGE SWELL'S
3 8	344.8	7.4	T56	SW 7	29.95	NIL	
3 9	350.4	6.4	T56	SW 7	29.95	NIL	30FT SWELLS CREVIS TO PEAK
3 10	357.4	6.4	T56	SW 7	29.95	NIL	
3 11	364.4	7.0	T56	SW 7	29.95	NIL	
N. 3 12	371.2	6.8	T56	SW 7	29.95	NIL.	
P.M. 3 1	376.9	5.7	T56	SW 5 6	29.9	NIL	SEAS RECEDING
3 2	382.7	5.7	T56	S.W 4 5	29.9	NIL	" "
3	387.7	5.0	T56	S 5 6	29.9	NIL	
4	392.7	5.0	T56	SES 6	29.9	NIL	
3 5	397.7	5.0	T56	SES 6	29.95	NIL	
4 6	401-1	5.4	T56	SES 6	30.0	NIL.	VERY NEARLY REACHING
4 7	410	9.0	T56	SES 6	30.1	NIL	HER MAXIMUM HULL SPEED
4 8	419	9.0	T56	SES 6	30.1	NIL	THIS IS HER BEST POINT OF
4 9	428	9.0	T56	SES-6	30.1	NIL	SAILING WITH THE WIND JUST
4 10	437.9	9.9	T56	SE 6	30.1	NIL	ABAFT OF BEAM 9 NOTS FOR
4 11	445.2	7.3	T56	SE 6	30.1	NIL	THE HOUR
81 12	452.6	7.3	T56	SE 6	30.1	NIL	MODERATE SEAS

	LATITUDE	LONGITUDE
A.M.		
NOON	28°55'S	171°40'
12 MN. D.R. P.M.	28°35'S	169°50

D.R. POSITION

DAY'S RUN 140.2 Nautical Miles

CRUISE RUN 371.2 Nautical Miles

ENGINE USED .... Hrs. .... Mins.

FUEL—IN GALLONS

IN HAND	RECEIVED	CONSUMED	REMAINING
37	120	NIL	37

- Sum up all the rocks for the year
- Average # of rocks per day
- Biggest week
- Smallest month

For the video and transcript of this presentation, click here:

https://lispcast.com/all-i-needed-for-fp-i-learned-in-high-school-algebra/

What makes numbers, an abstract idea, so useful for modeling real piles of rocks?

Correspondence of Properties

Information System

Distributed and Concurrent

Parallelization/ Distributed work

In distributed/parallel work, work comes back out of order

Order doesn't matter

4

a

\+

2

b

6

a + b

2 4

b + a

6

b + a

$$a + b = b + a$$

```
(f a b)
```

```
(f a b)
(f b a)
```

```
(= (f a b)
   (f b a))
```

```
(f a)
```

```
(g (f a))
```

```
(g (f a))
   (g a)
```

```
(g (f a))
(f (g a))
```

```
(= (g (f a))
   (f (g a)))
```

Parallelization/ Distributed work

Need to break up task to give to workers

Need to combine groups of answers

Needs to be cheap to break up and recombine groups

Grouping doesn't matter

5

1

3

a + b + c

6

3

$(a + b) \quad + \quad c$

9

a + b + c

5 1 3

a + b + c

5

4

$a + (b + c)$

9

a + b + c

$$(a + b) + c = a + (b + c)$$

a	b	c
a	b	c

```
(f a    b)  c
   a    b   c
```

```
(f a    b)  c
   a (f b   c)
```

```
(f (f a   b) c)
   (f a (f b c))
```

```
(f (f a b) c)
(f a (f b c))
```

```
(= (f (f a b) c)
   (f a (f b c)))
```

Types

```
(= (f (f a b) c)
   (f a (f b c)))
```

return value of f and its two
arguments need to be the same type

Whole Values

Combining two piles makes a new pile

Concatenating two lists makes a new list

Self-contained

```clojure
(defn average [a b]
  (/ (+ a b) 2))
```

Order doesn't matter

```
(= (average a b) (average b a))
a = 10, b = 4

(average 10 4)  => 7
(average 4 10)  => 7
```

Does grouping matter?

```
(= (average (average a b) c)
   (average a (average b c)))
a = 10, b = 4, c = 6

(average 10 4)  => 7

(average 7  6)  => 6.5

(average 4  6)  => 5

(average 10 5)  => 7.5
```

```
function average(numbers) {
  var sum   = 0;
  var count = 0;
  for(i = 0; i < numbers.length; i++) {
    sum   += numbers[i];
    count += 1;
  }
  if(count === 0) {
    return null;
  }
  return sum / count;
}
```

```
function average(numbers) {
  var sum   = 0;
  var count = 0;
  for(i = 0; i < numbers.length; i++) {
    sum   += numbers[i];
    count += 1;
  }
  if(count === 0) {
    return null;
  }
  return sum / count;
}
```

```
(defn combine [[sum1 count1] [sum2 count2]]
  [(+ sum1 sum2) (+ count1 count2)])

(defn ->average [number]
  [number 1])

(defn average [numbers]
  (reduce combine ? (map ->average numbers)))
```

Where do you start a computation?

$$a + 0 = a$$

(f a i)

```
(= (f a i) a)
```

```
(defn combine [[sum1 count1] [sum2 count2]]
  [(+ sum1 sum2) (+ count1 count2)])

(defn ->average [number]
  [number 1])

(defn average [numbers]
  (reduce combine ? (map ->average numbers)))
```

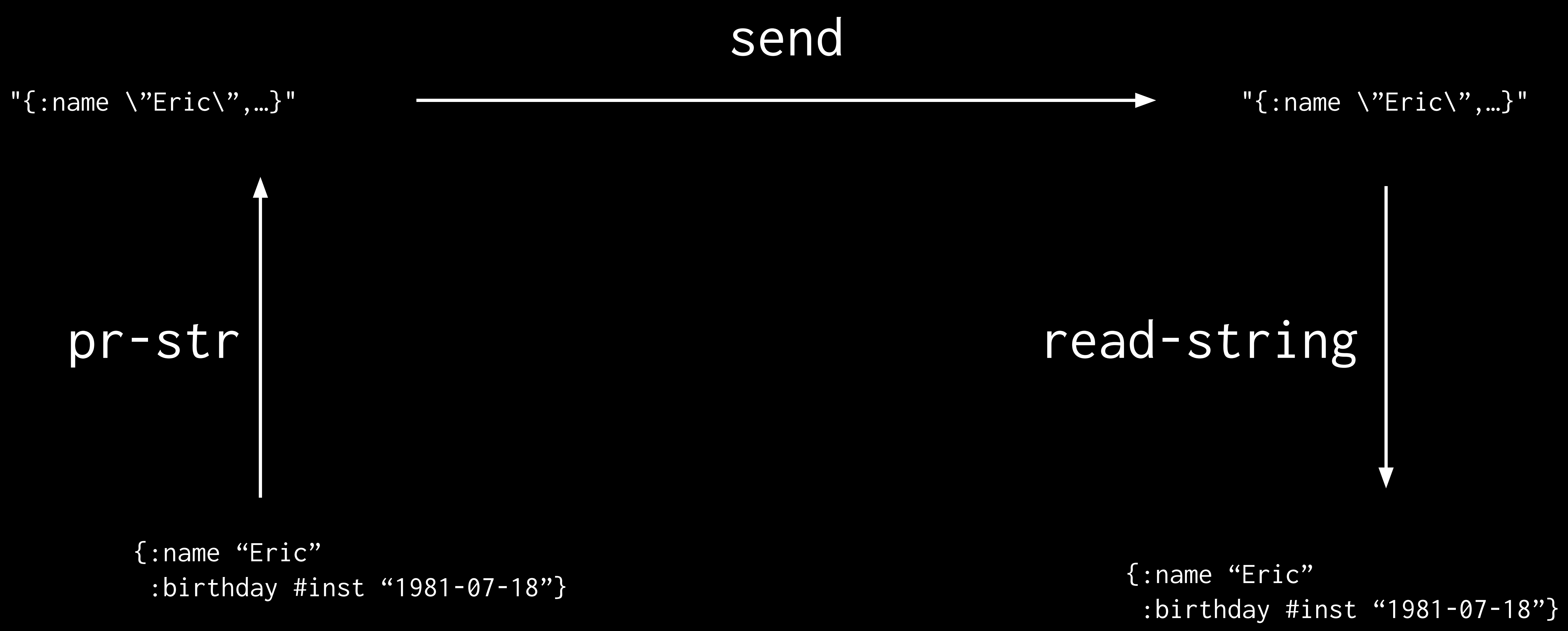
```
function average(numbers) {
  var sum   = 0;
  var count = 0;
  for(i = 0; i < numbers.length; i++) {
    sum   += numbers[i];
    count += 1;
  }
  if(count === 0) {
    return null;
  }
  return sum / count;
}
```

```
(defn combine [[sum1 count1] [sum2 count2]]
  [(+ sum1 sum2) (+ count1 count2)])

(defn ->average [number]
  [number 1])

(defn average [numbers]
  (reduce combine [0 0] (map ->average numbers)))
```


Entrannce
EXIT

Going back and forth matters

Great for moving into a new space, doing a calculation, then moving back


send
"{:name \”Eric\”,…}"
"{:name \”Eric\”,…}"
pr-str
read-string
{:name “Eric”
:birthday #inst “1981-07-18”}
{:name “Eric”
:birthday #inst “1981-07-18”}

(f a)

```
(g (f a))
```

```
(= (g (f a)) a)
```

```clojure
(defn combine [[sum1 count1] [sum2 count2]]
  [(+ sum1 sum2) (+ count1 count2)])

(defn ->average [number]
  [number 1])

(defn average [numbers]
  (reduce combine [0 0]
    (map ->average numbers)))
```

```clojure
(defn combine [[sum1 count1] [sum2 count2]]
  [(+ sum1 sum2) (+ count1 count2)])

(defn ->average [number]
  [number 1])

(defn average-> [[sum count]]
  (/ sum count))

(defn average [numbers]
  (->> numbers
    (map ->average)
    (reduce combine [0 0])
    average->))
```

Distributed

Messages arrive one or more times

Distributed

Independent workers have to coordinate to avoid duplicate work

Duplicates don't matter

2
U
2

```
(= (-> m
      (assoc :a “hello”)
      (assoc :a “hello”))
   (-> m
      (assoc :a “hello”))
```

```
(=    (f a)
      (f a))
```

```
(= (f (f a))
         (f a))
```

```
(def button-state (atom {}))

(defn press! [button-id]
  (swap! button-state assoc button-id true))

…

(press! :3rd-floor-north-up)

(press! :3rd-floor-north-up)

(press! :3rd-floor-north-up)
```

Nothing else matters

Know when to end

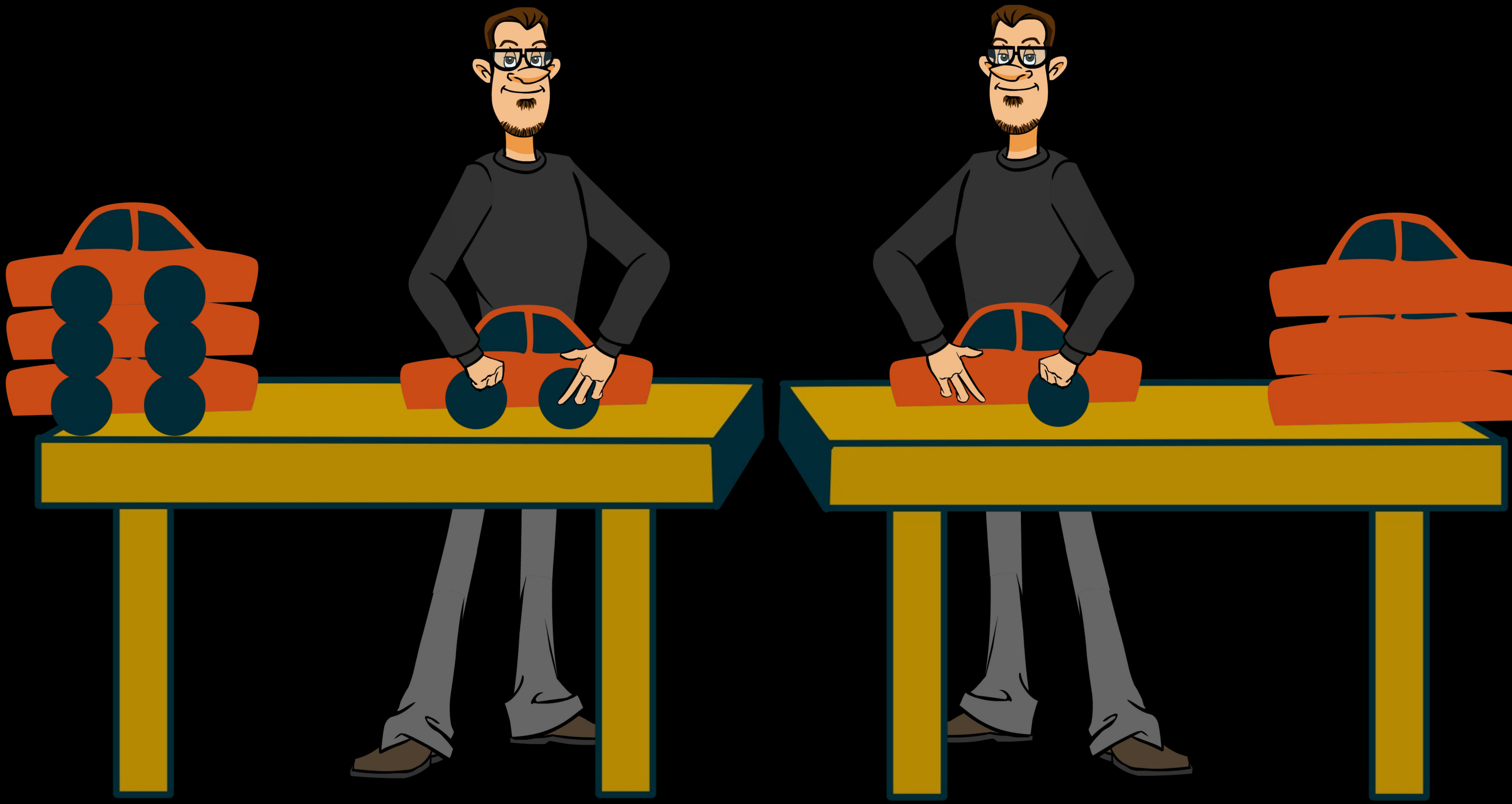
Circuit-breaking

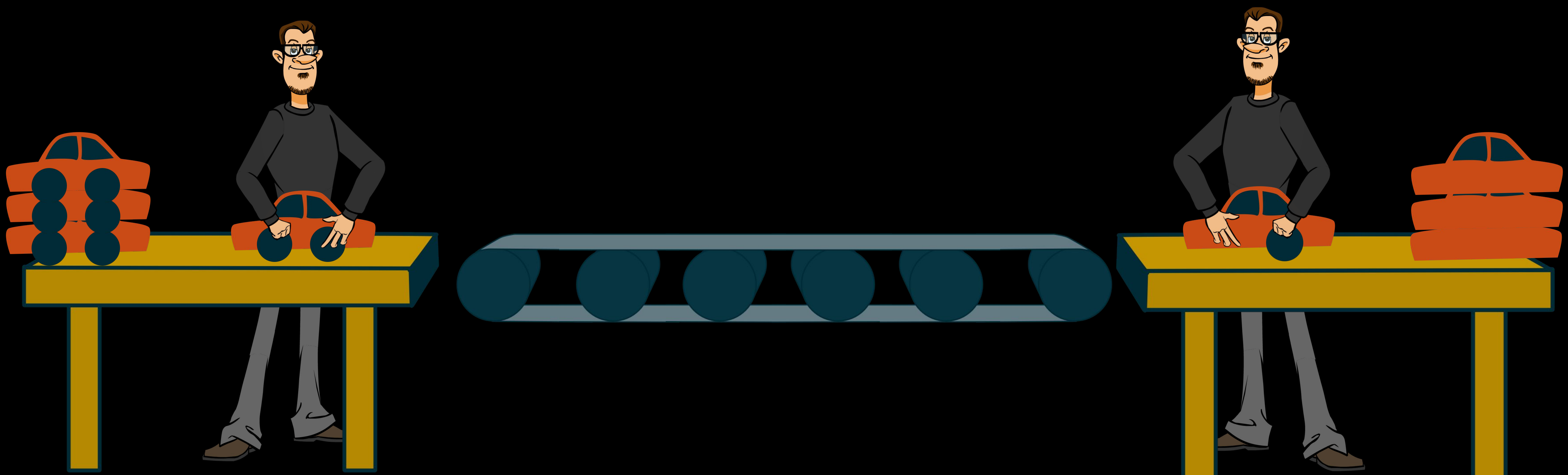
a * b * c * 0 * d * e * f

```
a * 0 = 0
```

```
(f a z)
```

```
(= (f a z) z)
```





Splitting up work and recombining it matters

Great for arranging and rearranging work in a pipeline

Composing transducers

```clojure
(= (->> cars
     (map add-back-wheel)
     (map add-front-wheel))
   (->> cars
     (map (comp
            add-front-wheel
            add-back-wheel))))
```

```
(= (map identity a) a)
```

```
(= (map identity a) a)

         (map g a)
```

```
(= (map identity a) a)

  (map f (map g a))
```

```
(= (map identity a) a)

   (map f (map g a))
        (comp f g)
```

```
(= (map identity a) a)

  (map f (map g a))
  (map (comp f g) a)
```

```
(= (map identity a) a)

(= (map f (map g a))
   (map (comp f g) a))
```

Conclusions

Commutative	Order doesn't matter	`(= (f a b) (f b a))`
Associative	Grouping doesn't matter	`(= (f (f a b) c)` `   (f a (f b c)))`
Identity value	Where to start	`(= (f a i) a)`
Zero value	When to stop	`(= (f a z) z)`
Idempotence	Duplicates don't matter	`(= (f (f a)) (f a))`
Reversibility	Going back and forth	`(= (g (f a)) a)`
Structure Preservation	Rearranging work	`(= (m identity a) a)` `(= (m (comp f g) a)   (m f (m g a)))`

These properties are what allow us to do our work

```
(= (f a b)
   (f b a)))
```

```
(prop/for-all [a S
               b S]
  (= (f a b)
     (f b a)))
```

```
(prop/for-all [a gen/int
               b gen/int]
  (= (* a b)
     (* b a)))
```

Algebraic properties make great test.check properties

Eric Normand

LispCast

Follow Eric on:

Eric Normand @EricNormand

lispcast.com eric@lispcast.com