Add ALL the things

avi@stripe.com

Steven H. Noble
@snoble

Following

If you are building a system for calculating aggregates and you don't know the relevance of things like abelian groups... STOP!!

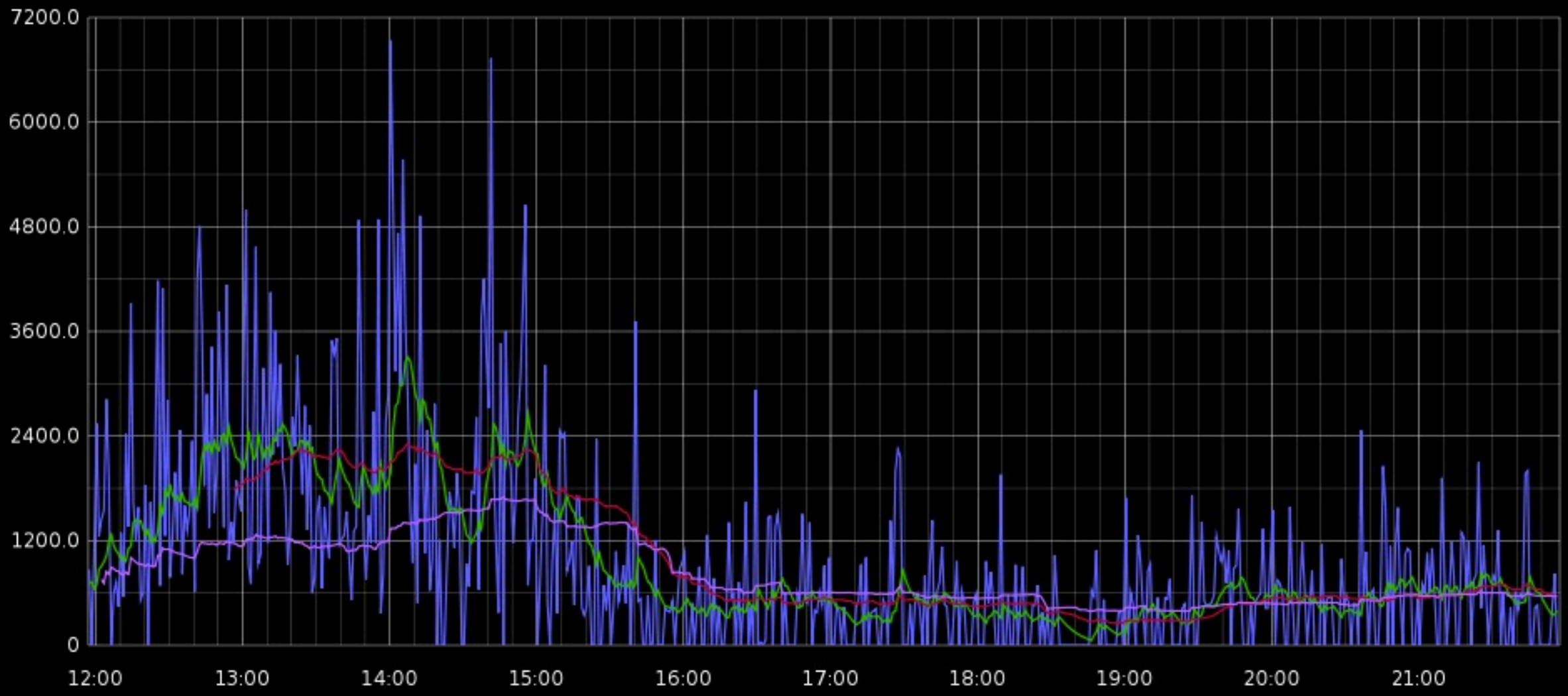
Reply Retweet Favorite More

lolwut

tl;dr

- Adding is awesome
- A **lot** of things that aren't adding are still "adding" (which is awesome)

Motivating example: StatsD-like



```ruby
module Stripe
  def process_payment(pmt)
    stats.increment("money", pmt.amount)
    transfer(pmt.from, pmt.to, pmt.amount)
  end
end
```

4

8

16

23

42

Addifier

93

4 + 8 + 15 + 16 + 23 + 42

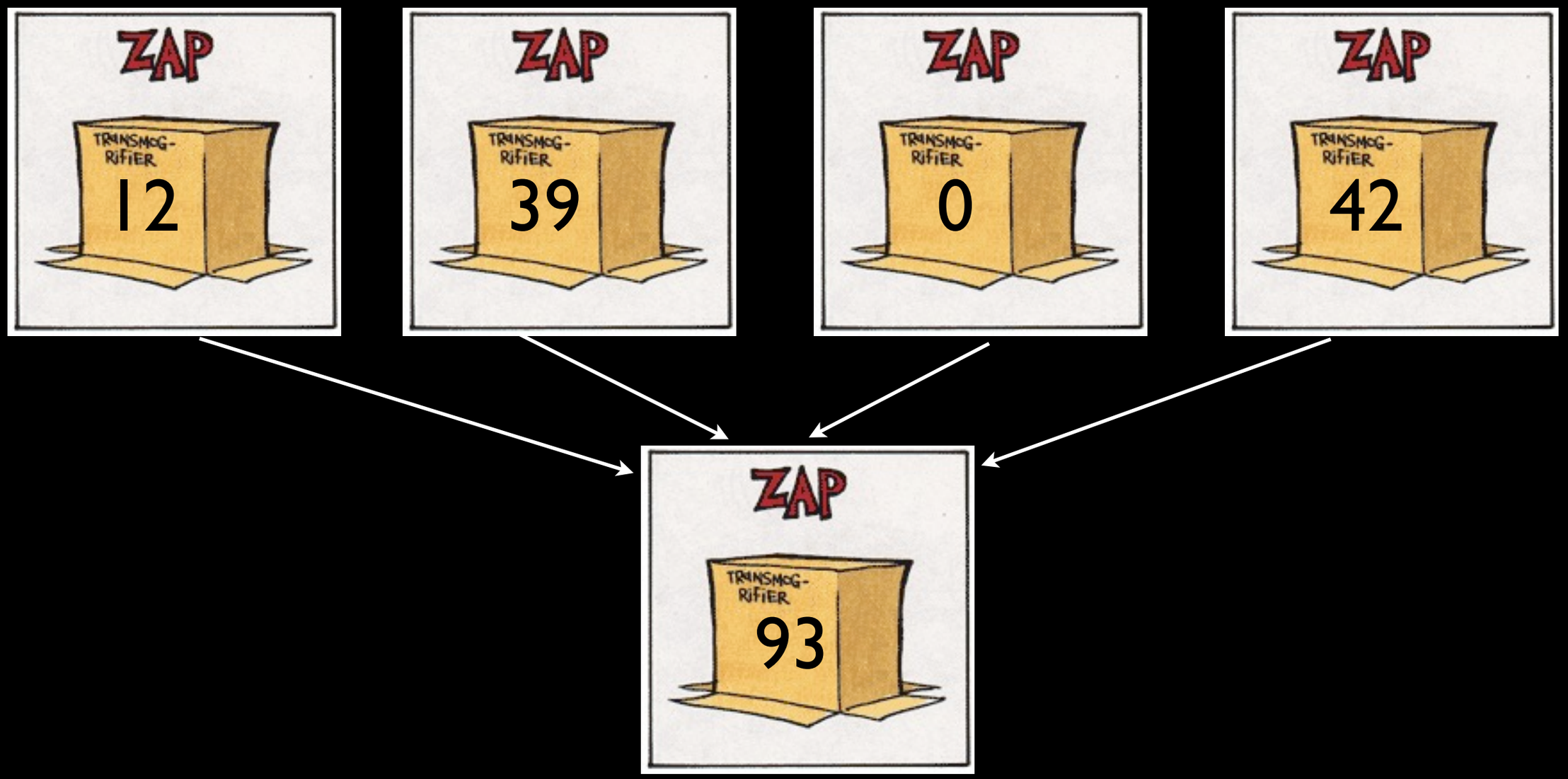
```
 8 class Addifier
 9   def initialize
10     @result = 0
11   end
12
13   def update(x)
14     @result = @result + x
15   end
16 end
```

4 8 16 23 42

(((((4+8)+15)+16)+23)+
42

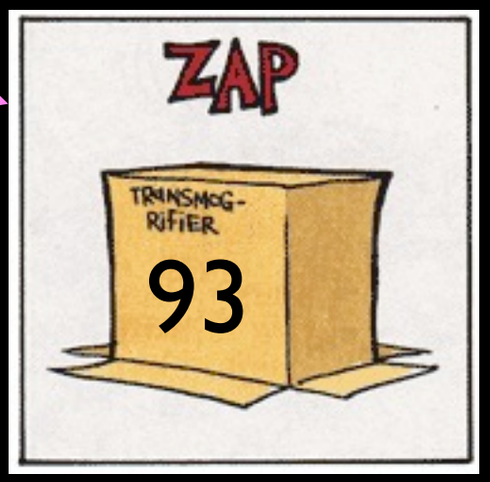
12:00 12:01 12:02 12:03

4 8 16 23 42



(4+8)+(16+23)+0+42

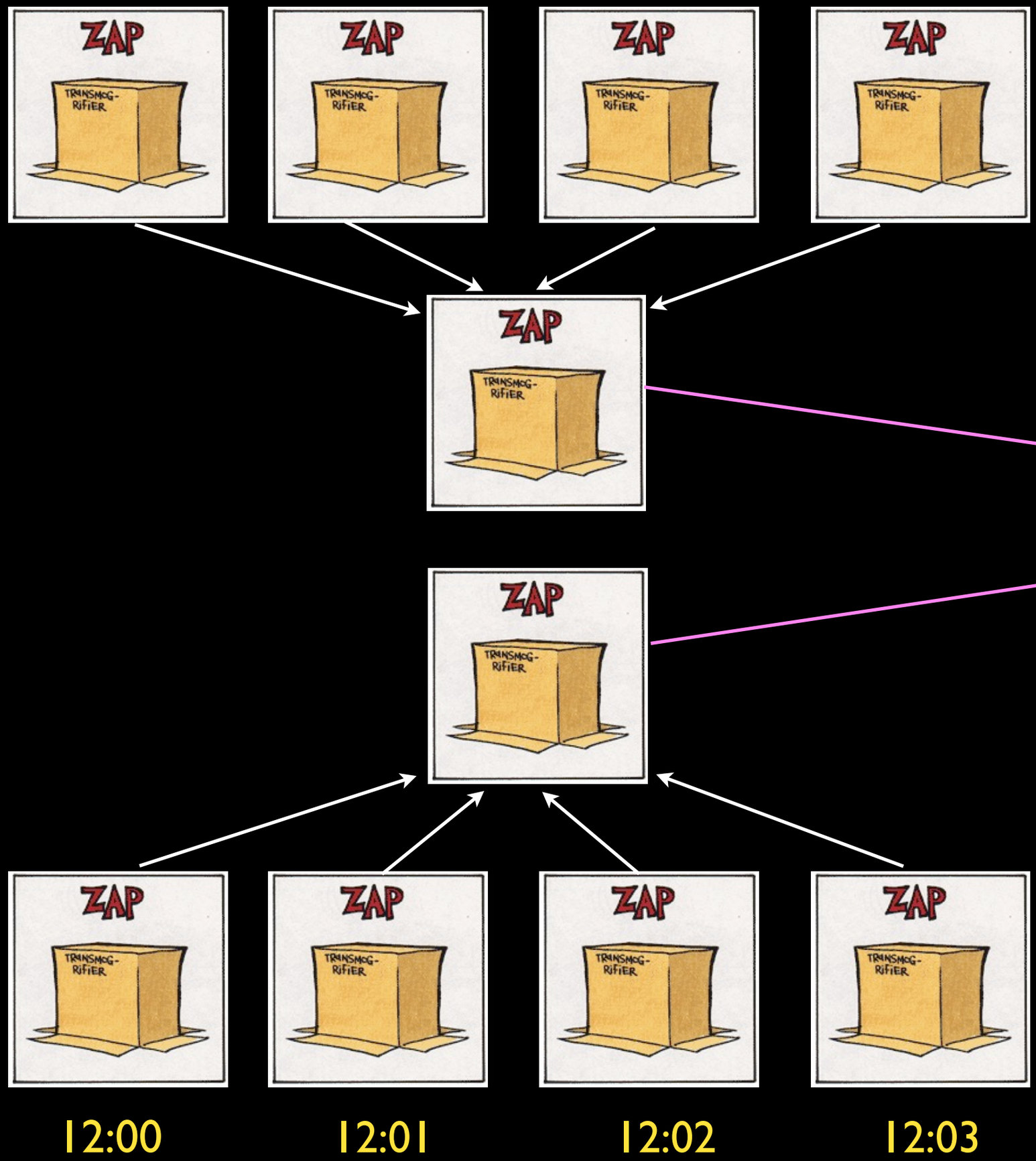
4 16 42



8 23

((4+16)+42)+(8+23)

12:00 12:01 12:02 12:03



12:00 12:01 12:02 12:03

```
 8 module Stripe
 9   def process_payment(pmt)
10     stats.time("transfer_ms") do
11       transfer(pmt.from, pmt.to, pmt.amount)
12     end
13   end
14 end
```

4

8

16

23

42

Maxifier

42

```ruby
class Maxifier
  def initialize
    @result = 0
  end

  def update(x)
    @result = [@result, x].max
  end
end

```

12:00

12:01

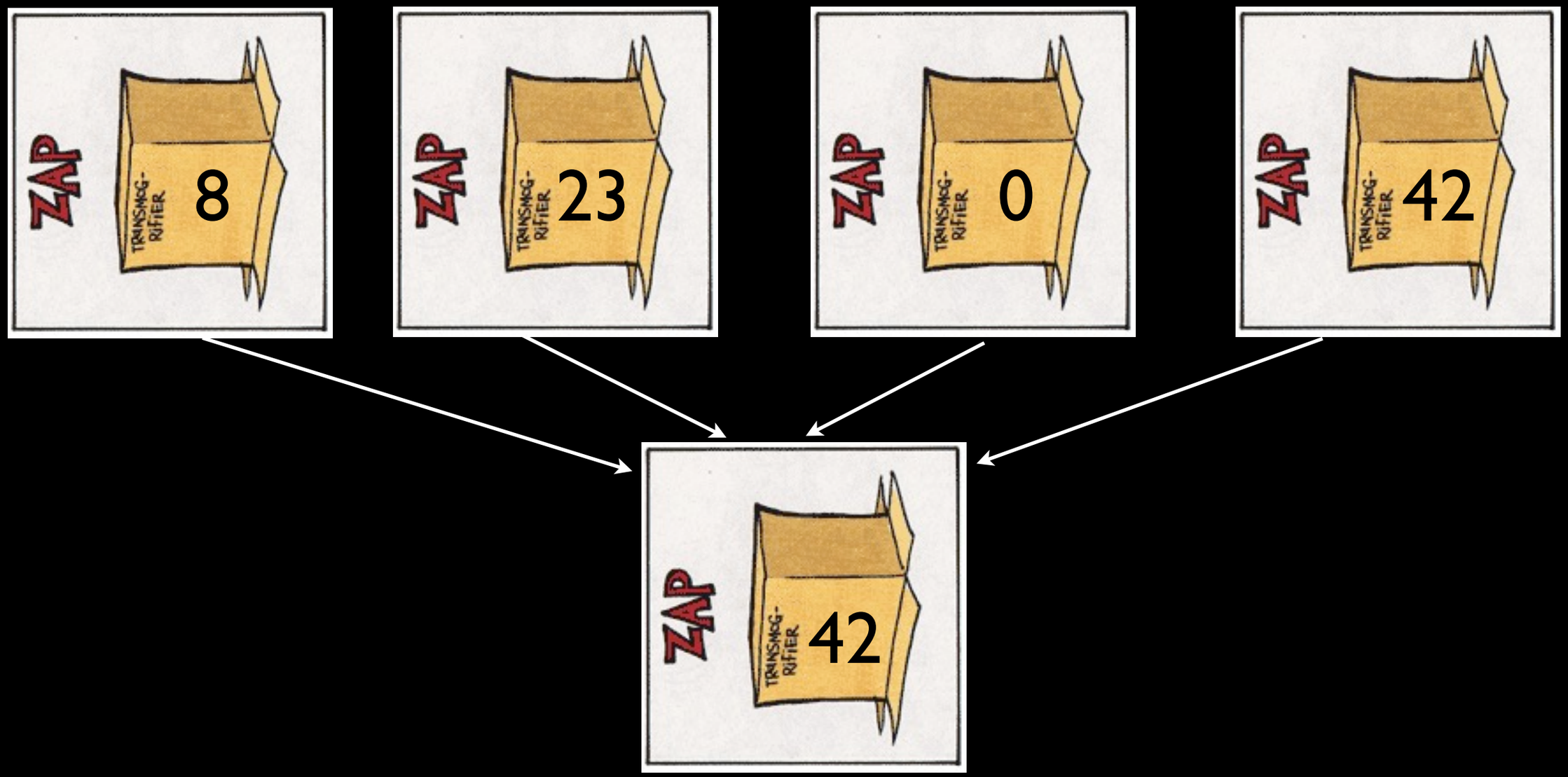
12:02

12:03

4 8

16 23

42



4 16 42

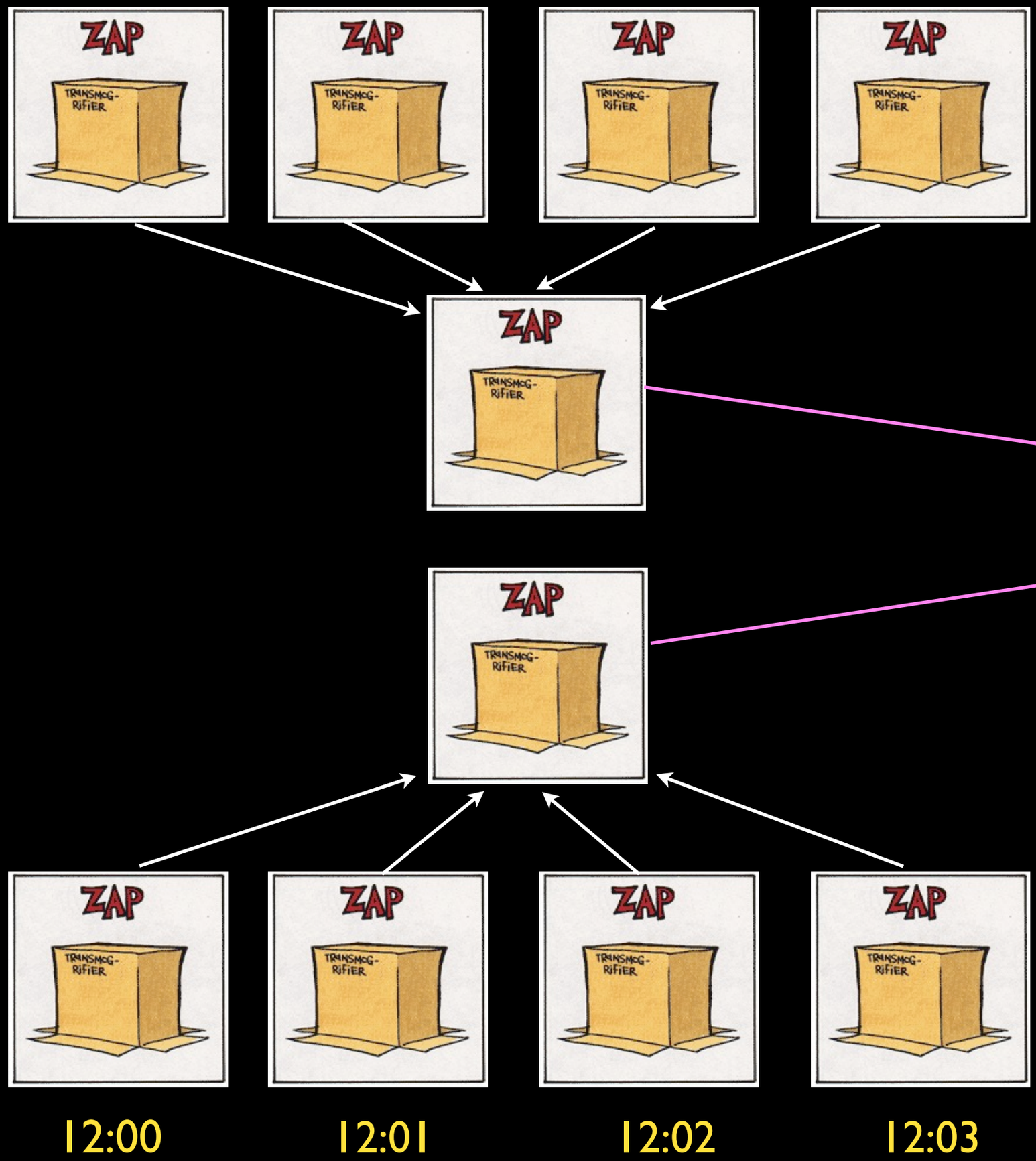

ZAP
TRANSMOG-RIFIER
42

8 23


ZAP
TRANSMOG-RIFIER
23


ZAP
TRANSMOG-RIFIER
42

12:00 12:01 12:02 12:03



12:00 12:01 12:02 12:03

Generalizing + and max

- 1. Takes two numbers and produces another number
- 2. Grouping doesn't matter (associative)
- 3. Ordering doesn't matter (commutative)
- 4. Zeros get ignored

Commutative monoid

A set S, with an operation that:

- 1. Takes two members of S and produces a member of S
- 2. Grouping doesn't matter (associative)
- 3. Ordering doesn't matter (commutative)
- 4. Ignores some "identity" element of S

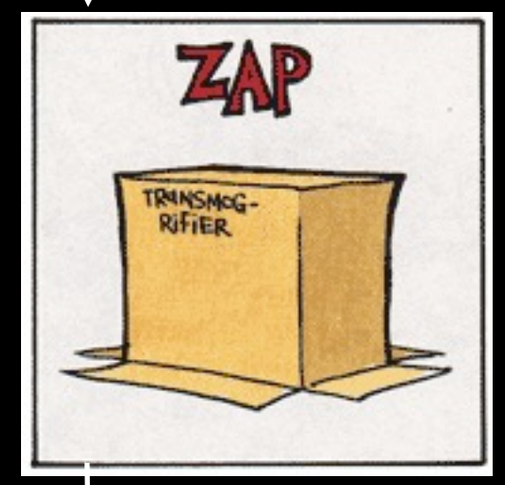
```ruby
class TopK
  def initialize(k)
    @k = k
    @result = {}
  end

  def update(hash)
    merged = @result.merge(hash)
    topk = merged.sort_by{|key,v| v}.reverse[0...@k]
    Hash[topk]
  end
end
```

{alice: 10}

{bob: 5}

{charlie: 7}

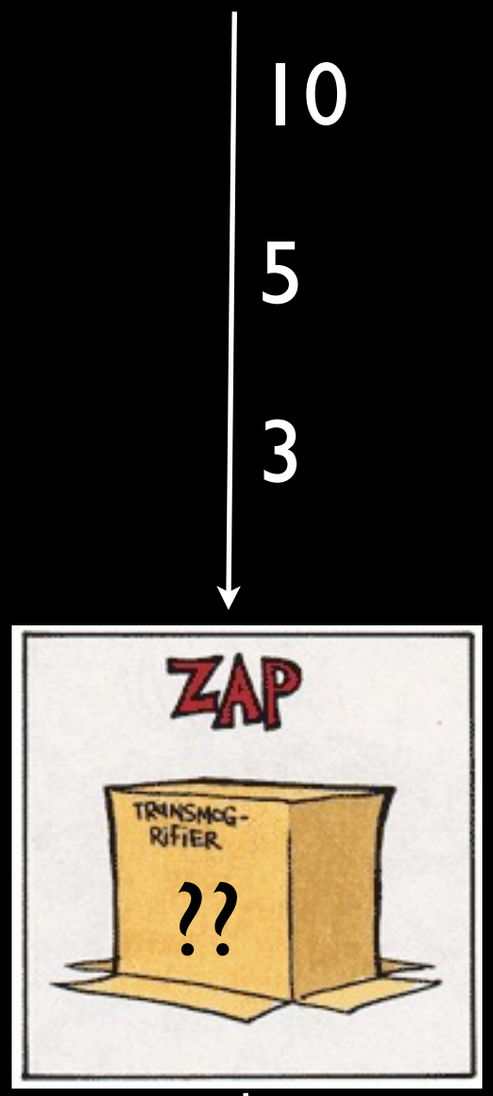
TopK Monoid



{alice: 10, charlie: 7}

(use a heap in real life, though)

Average Monoid

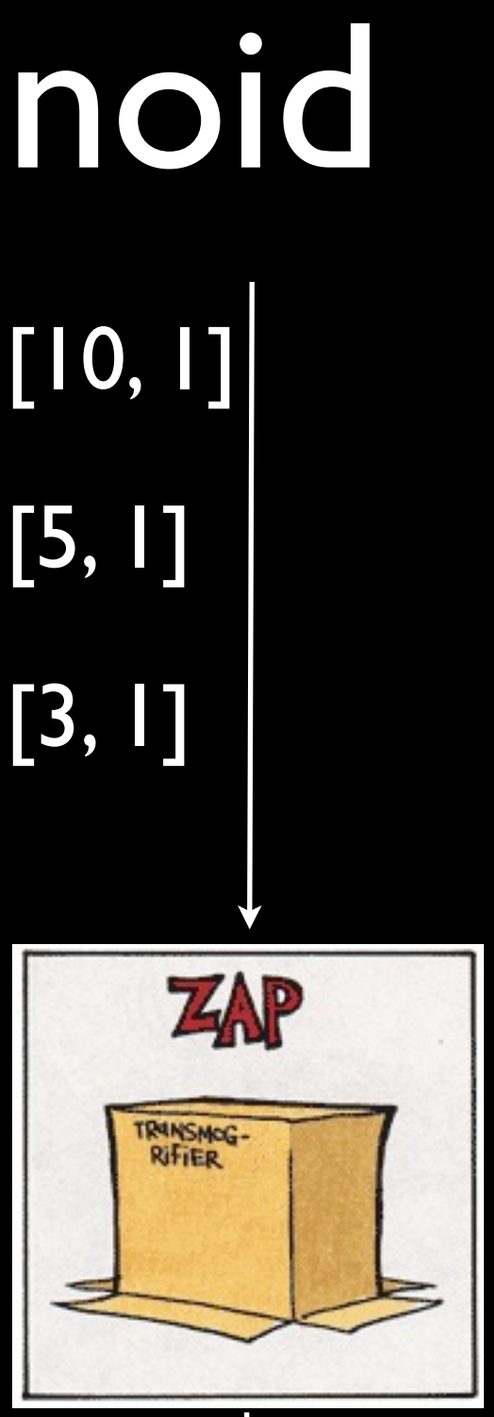


Average Monoid

10 → [10, 1]

5 → [5, 1]

3 → [3, 1]



[18, 3] → 6

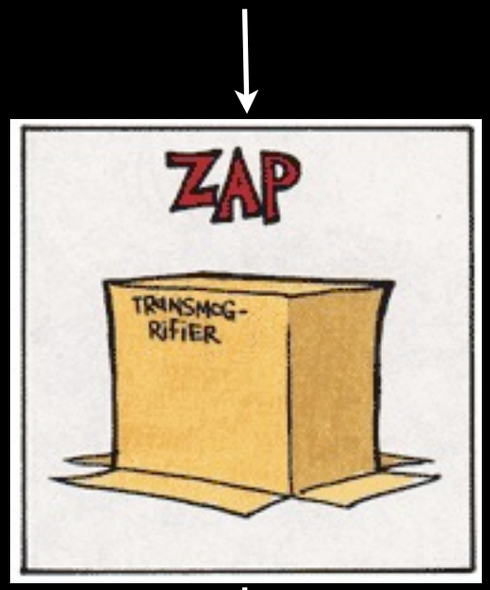
(use a numerically stable average in real life, though)

Histogram Monoid

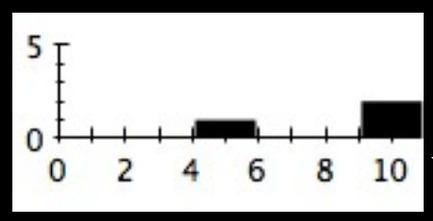
10 [0,0,0,0,0,0,0,0,0,1]

5 → [0,0,0,0,1,0,0,0,0,0]

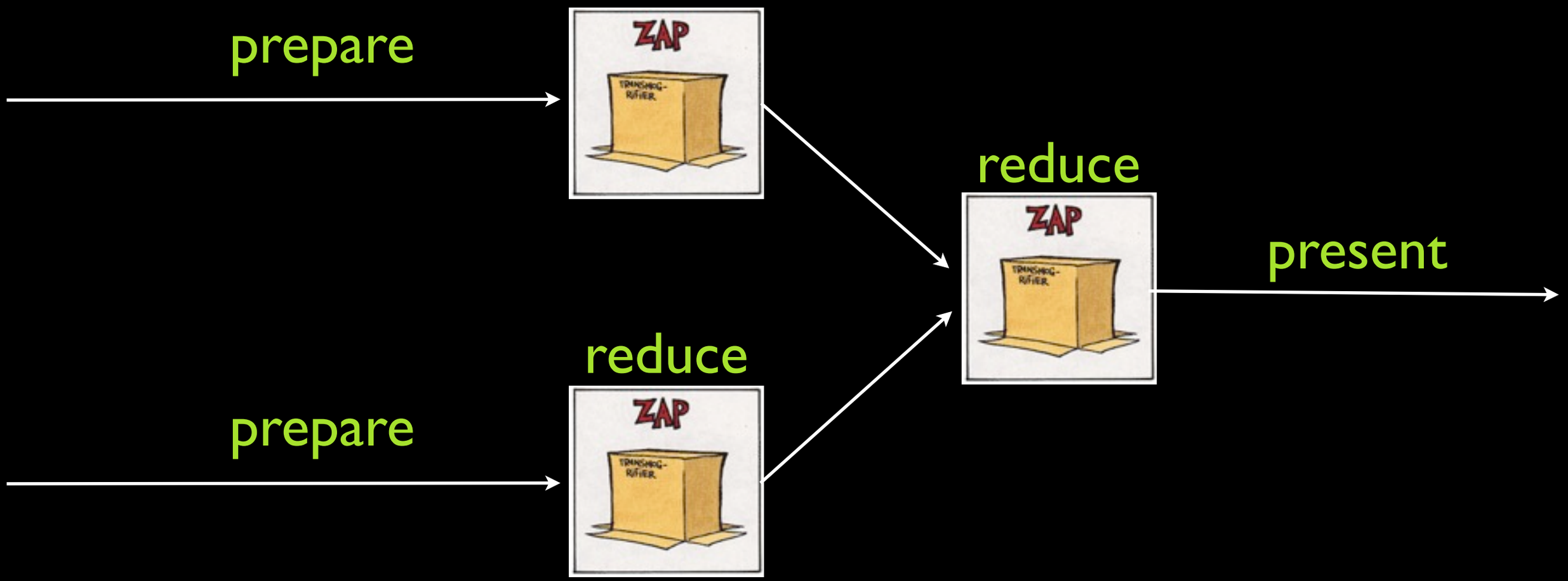
10 [0,0,0,0,0,0,0,0,0,1]



[0,0,0,0,1,0,0,0,0,2]



```java
public interface Aggregator<I,O,S> {
   S initial();
   S prepare(I input);
   S reduce(S left, S right);
   O present(S result);
}

```



Unique Values Monoid

alice

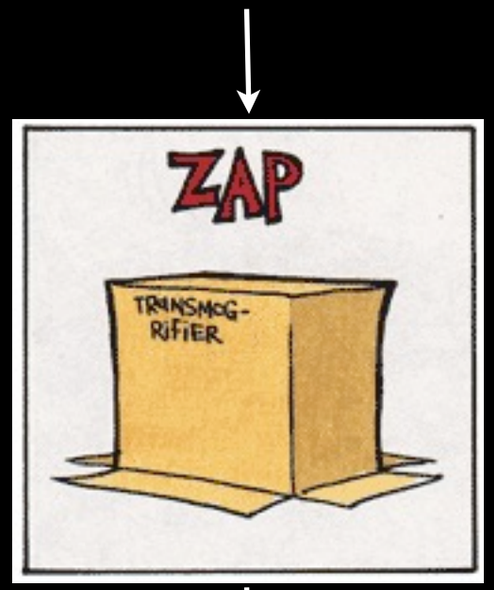
bob

alice

{alice}

{bob}

{alice}



2

{alice,bob}

Unique Values Monoid

alice

bob

alice

hash

0.789

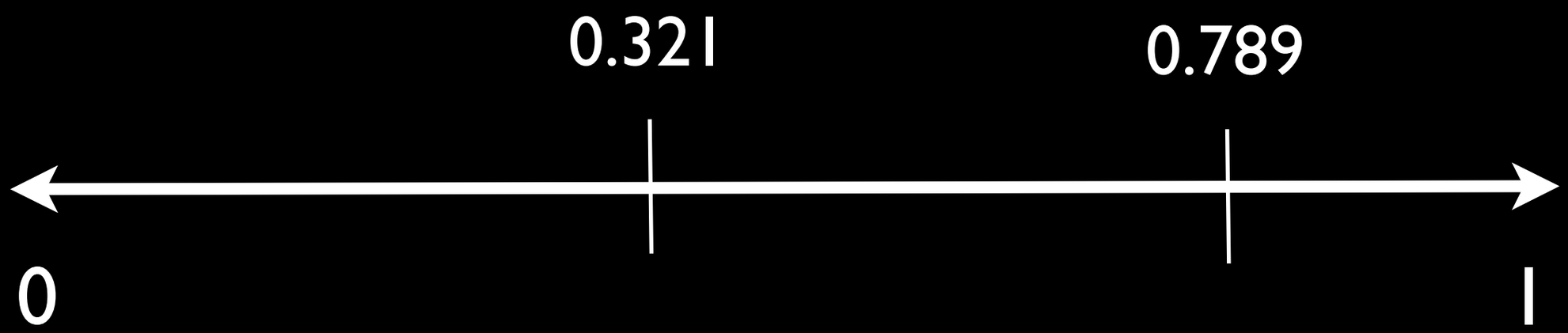
0.321

0.789

??

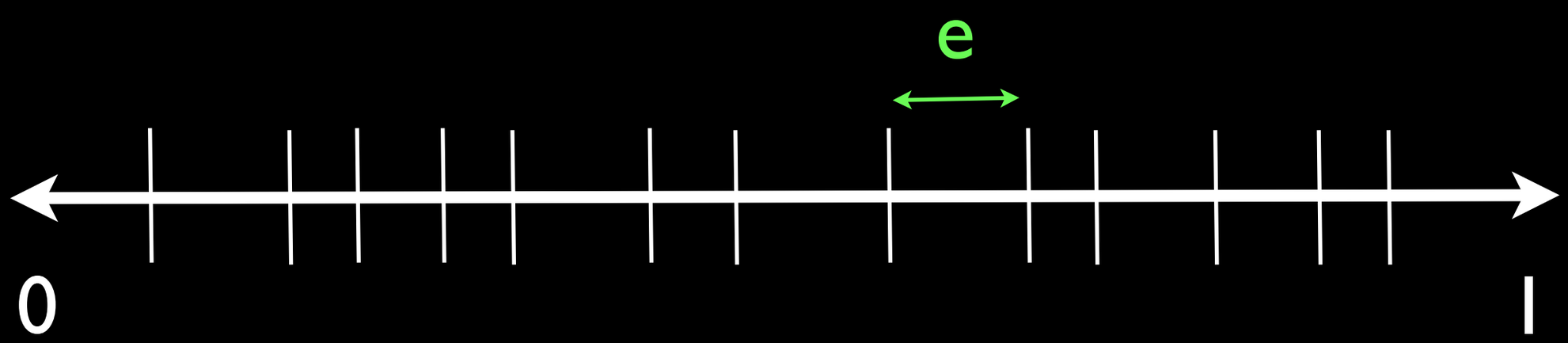
2

2 unique values



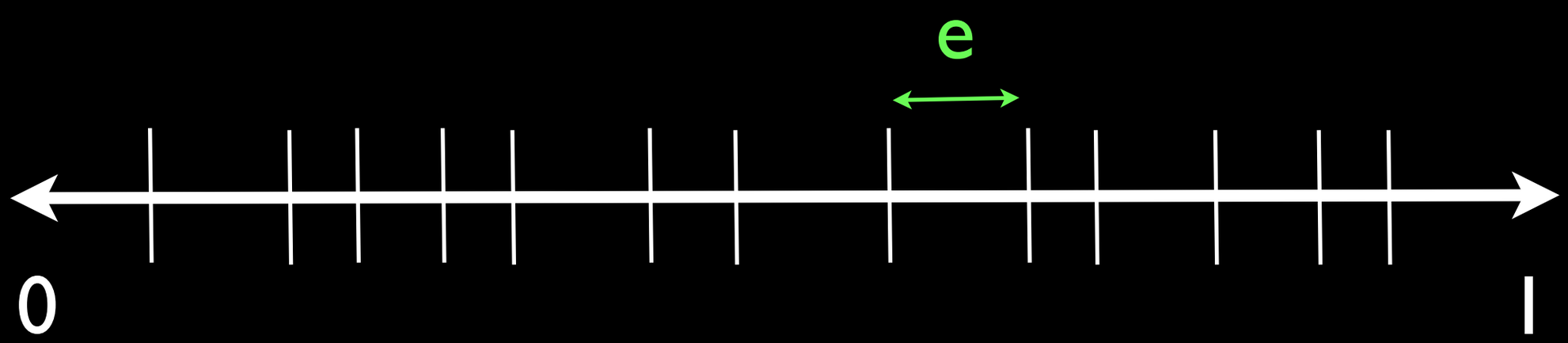
N unique values

E(e) = ??



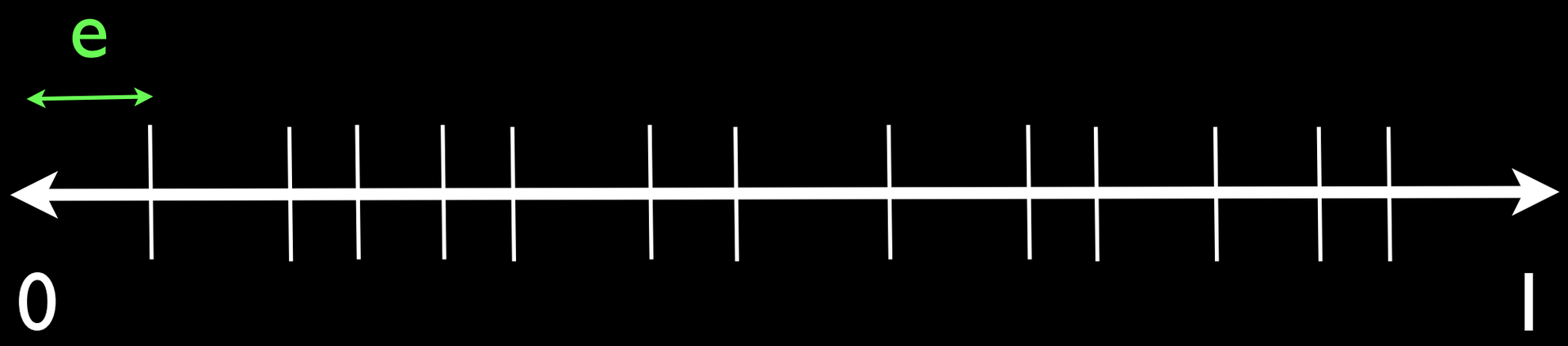
N unique values

$$E(e) = 1/(N+1)$$



N unique values

$$E(e) = 1/(N+1)$$



$$N = 1/e - 1$$

Unique Values Monoid

alice

bob

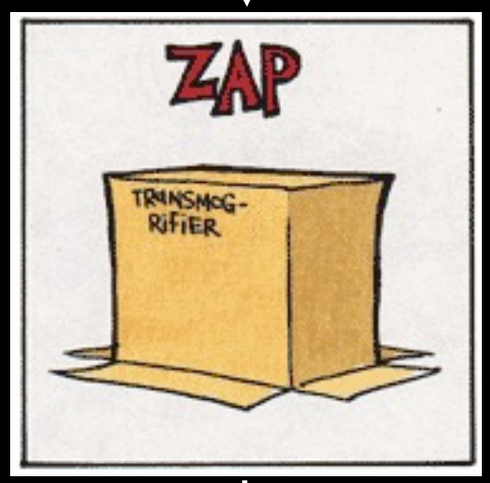
alice

hash

0.789

0.321

0.789



Min

0.321

1/e - 1

2.11

Unique Values Monoid

alice

bob

alice

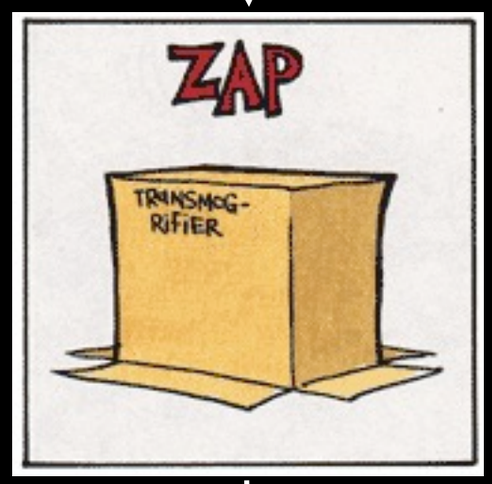
hash k times

[0.789, 0.456, 0.3]

[0.321, 0.666, 0.222]

[0.789, 0.456, 0.3]

Min



[0.321, 0.456, 0.222]

1/E(e) - 1

2.00

In real life

- HyperLogLog for unique values
- Min-hash for set similarity
- Bloom filters for set membership

Frequency Monoid

alice → {alice: 1}

bob → {bob: 1}

alice → {alice: 1}

{alice: 2, bob: 1}

Frequency Monoid

alice

bob

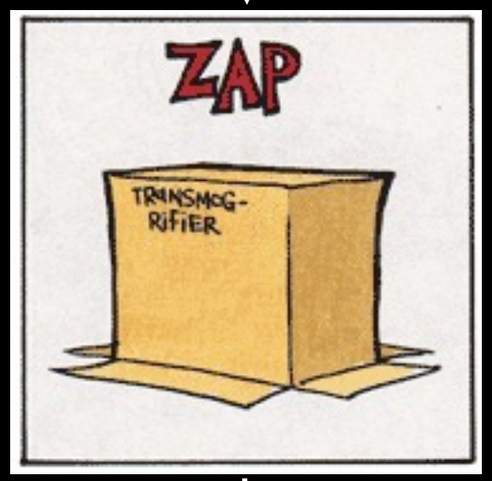
alice

hash % k

[0,0,1,0]

[0,1,0,0]

[0,0,1,0]



[0,1,2,0]

Frequency Monoid

alice

bob

alice

charlie

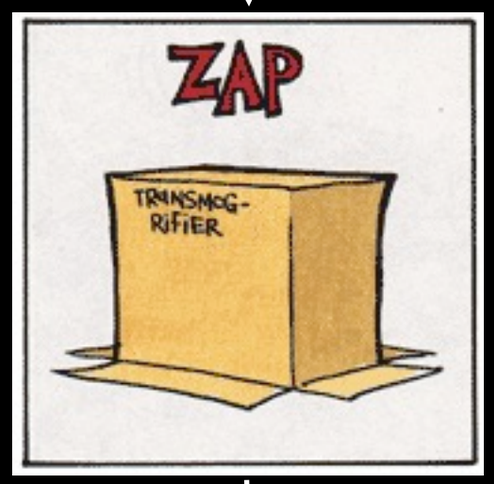
hash % k

[0,0,1,0]

[0,1,0,0]

[0,0,1,0]

[0,0,1,0]



[0,1,3,0]

Count-min Sketch

alice

		2	
	2		
			2

Count-min Sketch

charlie

		3	
	2	1	
	1		2

Count-min Sketch

bob

	1	3	
	3	1	
1	1		2

Count-min Sketch

alice?

	1	3	
	3	1	
1	1		2

- Semigroup: set and associative operation
- Monoid: semigroup with identity
- Group: monoid with inverse

Any of these can be (and usually are) commutative

Commutative Monoids:

- Max
- HyperLogLog
- Bloom Filter
- ...

Abelian Groups:

- Sum
- Average
- Count-min Sketch
- ...

Subtraction!

- http://github.com/twitter/algebird
- http://github.com/avibryant/simmer
- http://blog.aggregateknowledge.com

@avibryant
avi@stripe.com