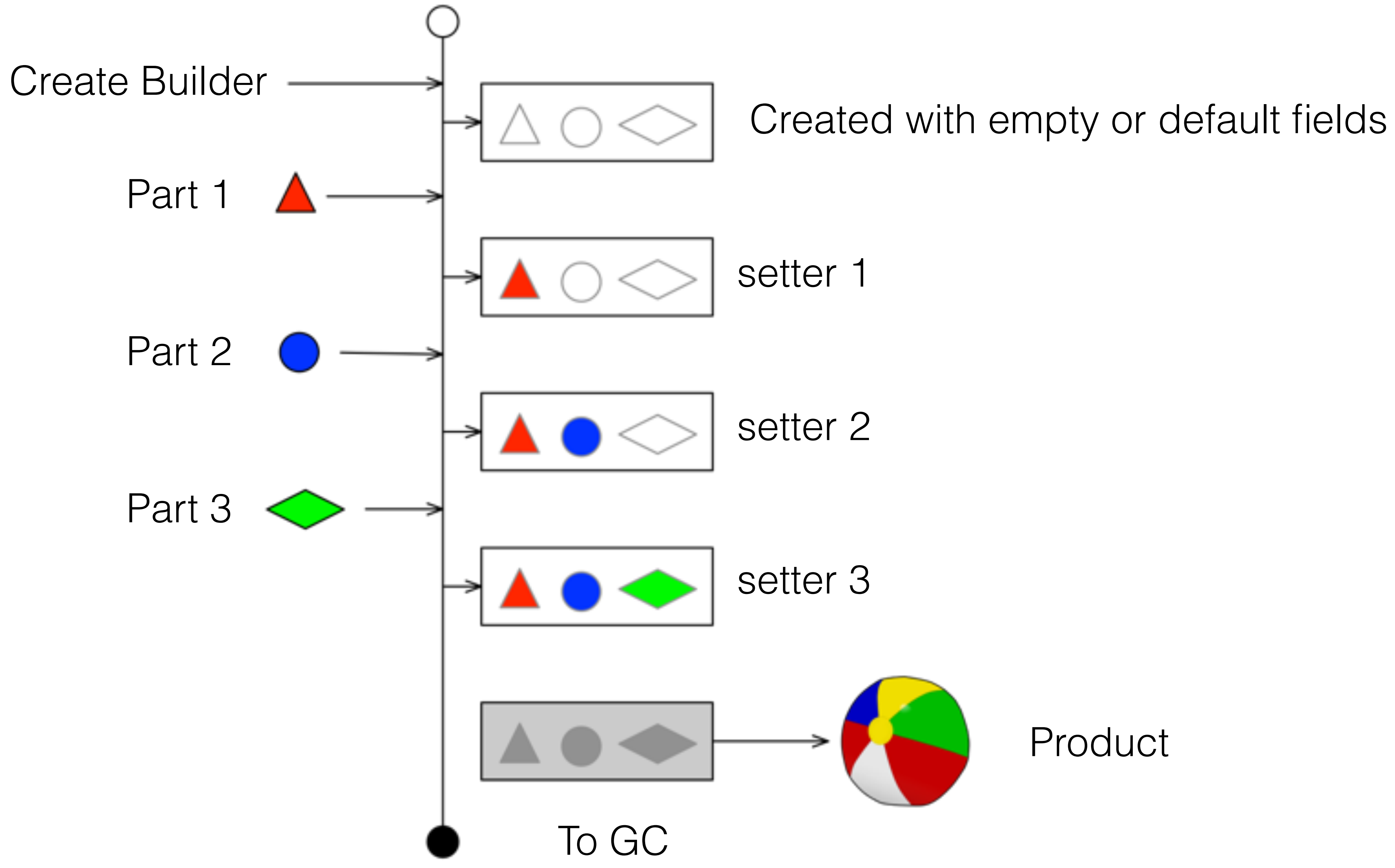
Reactive Builders

Builder Pattern

Why we use builders:

- Avoid “constructor hell”
- Building parts require complex validation
- Building parts require non trivial processing
- Avoid unnecessary dependencies in the Product
- Create immutable objects
- Object prototyping
- Flexible and configurable defaults
- Product Polymorphism


Create Builder
Created with empty or default fields
Part 1
setter 1
Part 2
setter 2
Part 3
setter 3
Product
To GC

Demo1

What can we expect from the Builder and the Product?

- Builder should be as reliable as constructor.
  It should never produce broken Products
- We know that parts will not change
  when we build the Product. Ideal parts should be immutable
- We know that if building parts change after
  we created the Product, the Product will not change
  Ideal Product is immutable

How we know we ready to create the Product?

- All mandatory Building Parts available
- All optional Parts have proper default values
- All parts were validated
- There is a valid consumer for the Product

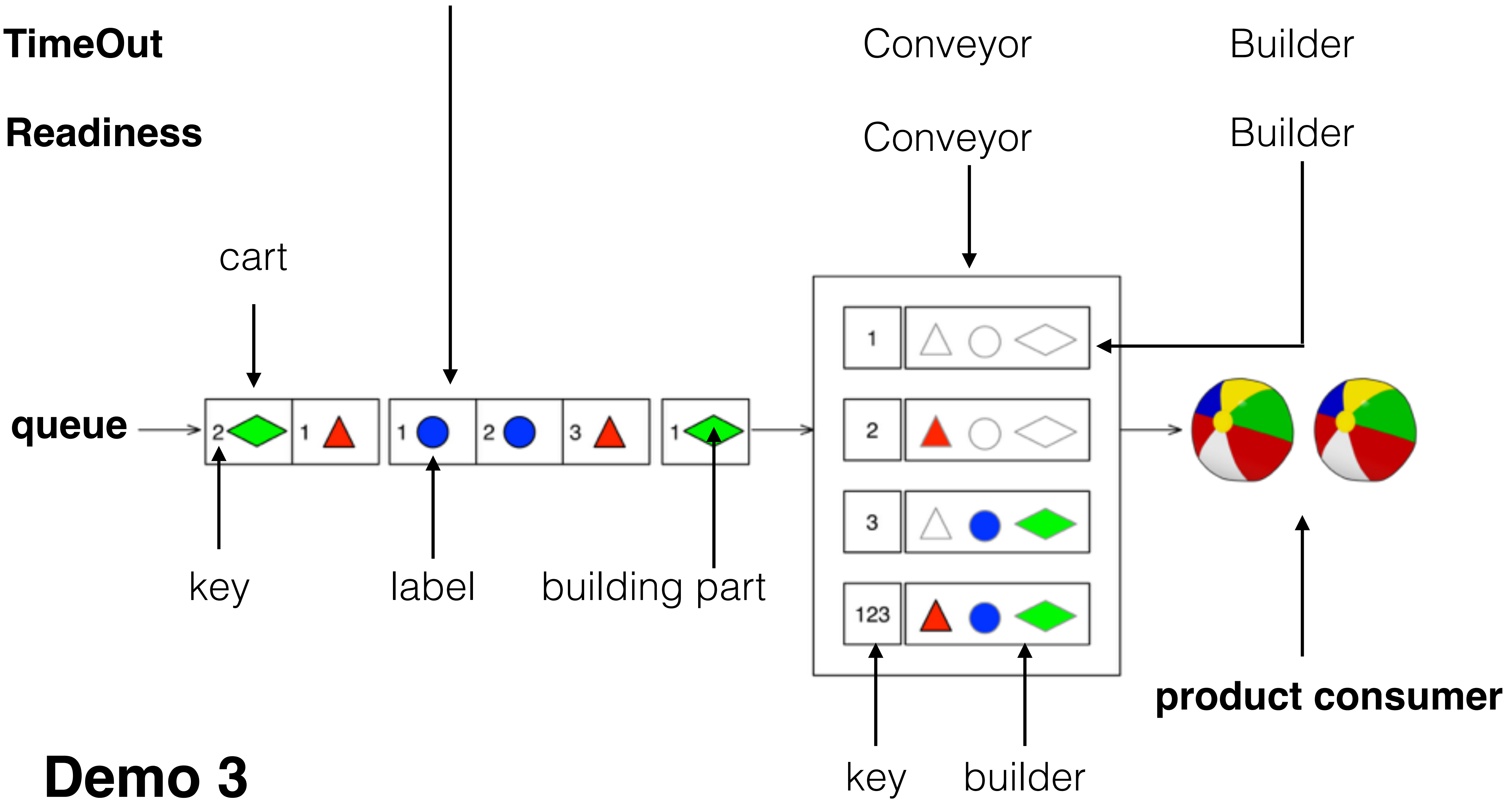
Reactive Builder concerns

- Building Parts arrive independently from each other
- Building Part carrier can change or use multiple carriers
- Order of Parts is not predictable
- TimeOfArrival is unknown
- What to do if some Parts have never arrived?
- What to do if Optional Parts have not arrived?
  Continue waiting or create Product without them?
- What to do if Product Consumer became unavailable?
- Does the Builder should take care about all this concerns?

Demo 2

Assembling Conveyor

Initialization	Cart	Conveyor	
Expiration	Cart	Conveyor	Builder
TimeOut		Conveyor	Builder
Readiness		Conveyor	Builder

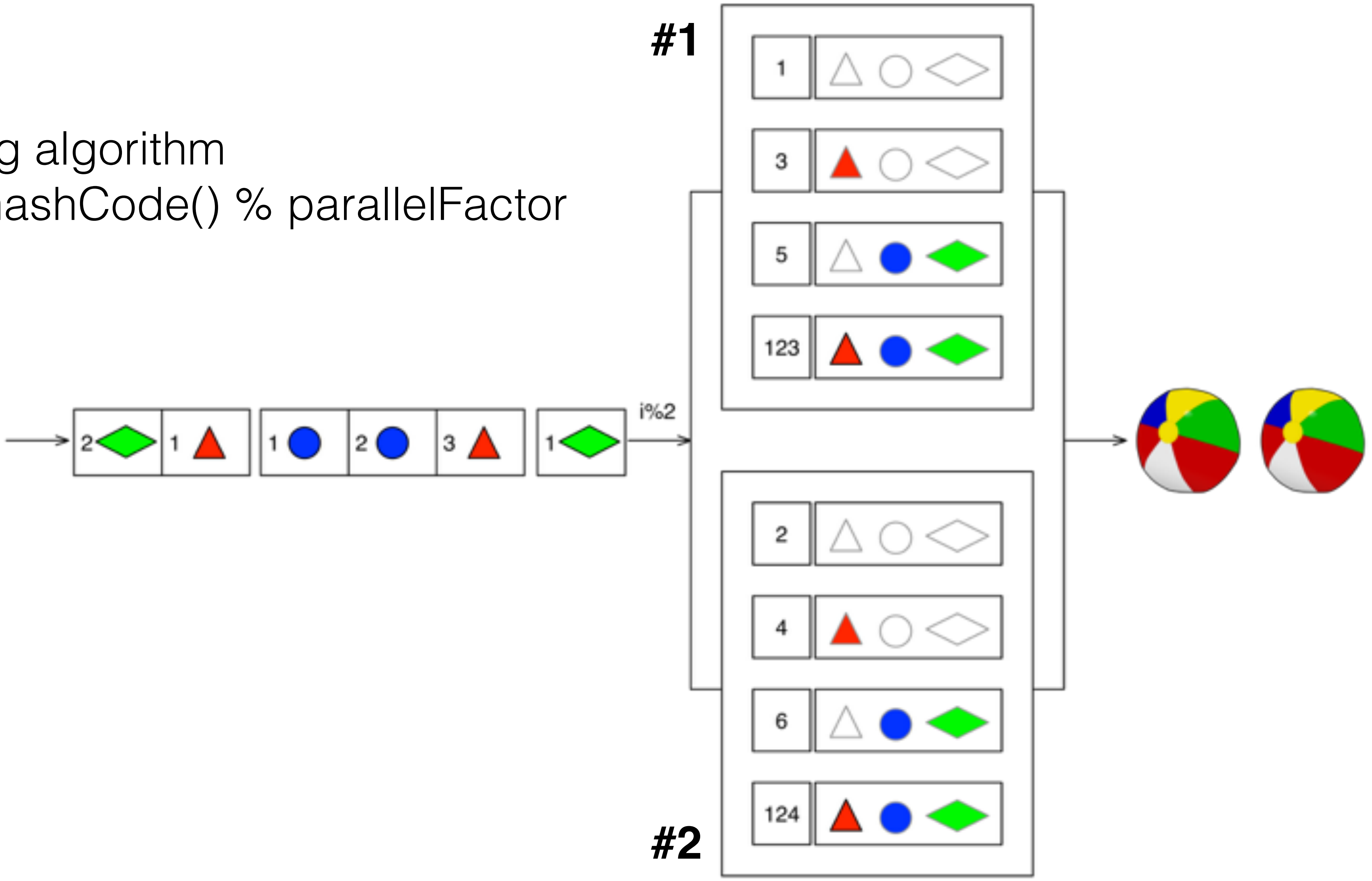


Demo 3

Special cases

Parallel Conveyor

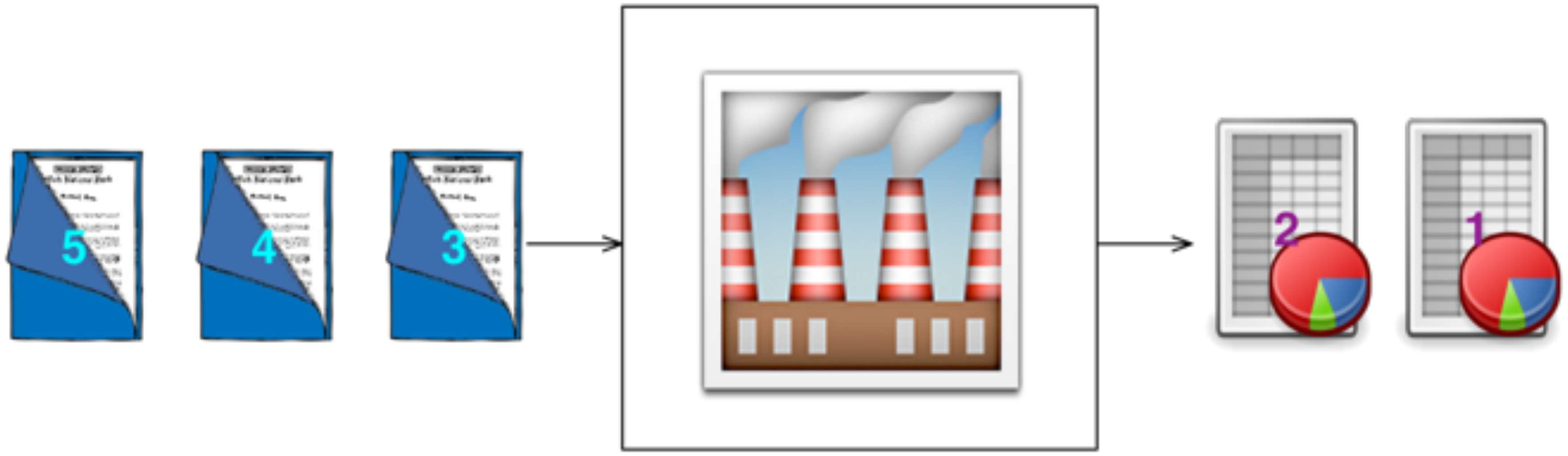
Balancing algorithm
# = key.hashCode() % parallelFactor



Special cases

readiness algorithm always return true:
(state,builder) -> true;

Features: can process only one cart or timeout
Use case: XML parser. One bulky piece of text to create a complex product

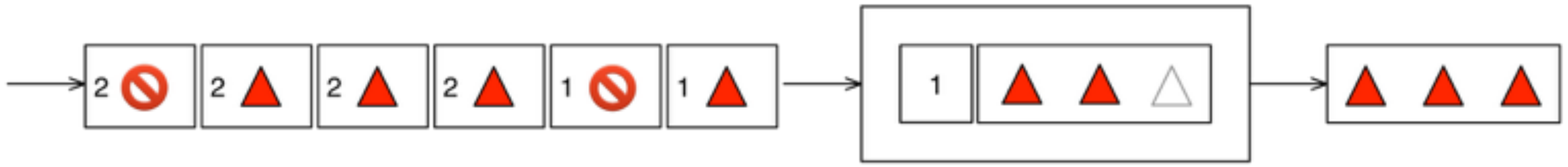


Demo 8

Special cases

Collections

- All building parts have the same type
- Special label for the end of collection with no value



Maps

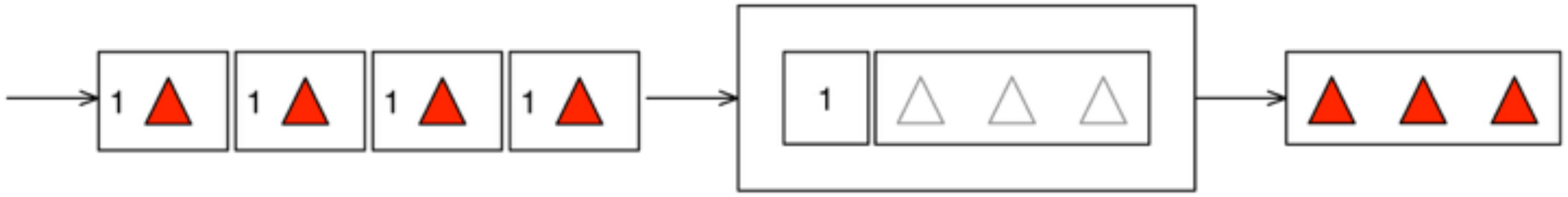
- Open list of labels which are used as keys for the map
- Special label marking end of the map with no value

Special cases

Batch

- All building parts have the same type
- very few distinct keys. 1 key per input channel
- readiness algorithm checking size of collection
- always true on timeout - save last portion.

Use cases: logger, database writer



Special cases

Caching

readiness algorithm always return false:
`(state,builder) -> false;`

Features:

- eviction only on timeout or never
- Layer of protection between Builder and Product user
  user only has access to product Supplier
- adjustable synchronization on a framework level

Use case: flexible cache

Demo 9

Thank you!

https://github.com/aegisql/conveyor