No time for math: How Clojure is making me a better Java programmer

```
{ :name "John Foo"
  :languages [:en :je :ja]
  :start-date #inst "2013-02-28" }
```

```
new HashMap<String, Object>() {{
     put("name", "John Foo");
     put("languages",
               new ArrayList<LanguageCode>(){{
                    add(LanguageCode.EN);
                    add(LanguageCode.JE);
                    add(LanguageCode.JA);
               }});
     put("start-date",
               DateFormat.
                    getDateInstance(
                         DateFormat.SHORT).
                              parse("02/28/13"));
}};
```

```
public interface Map<K, V>   {

    V get(java.lang.Object o);

}
```

```
public interface Map<K, V>   {

    int size();

    boolean isEmpty();

    boolean containsKey(java.lang.Object o);

    boolean containsValue(java.lang.Object o);

    V get(java.lang.Object o);

    java.util.Set<K> keySet();

    V put(K k, V v);

    V remove(java.lang.Object o);

    void putAll(java.util.Map<? extends K,? extends V> map);

    void clear();

    java.util.Set<java.util.Map.Entry<K,V>> entrySet();
}
```

```
public interface Map<K, V>   {

    int size();

    boolean isEmpty();

    boolean containsKey(java.lang.Object o);

    boolean containsValue(java.lang.Object o);

    V get(java.lang.Object o);

    java.util.Set<K> keySet();

    V put(K k, V v);

    V remove(java.lang.Object o);

    void putAll(java.util.Map<? extends K,? extends V> map);

    void clear();

    java.util.Set<java.util.Map.Entry<K,V>> entrySet();
}
```

```clojure
(let [f {:foo 2 :bar 3}]
  (+ (f :bar) 5))
```

What is a map?

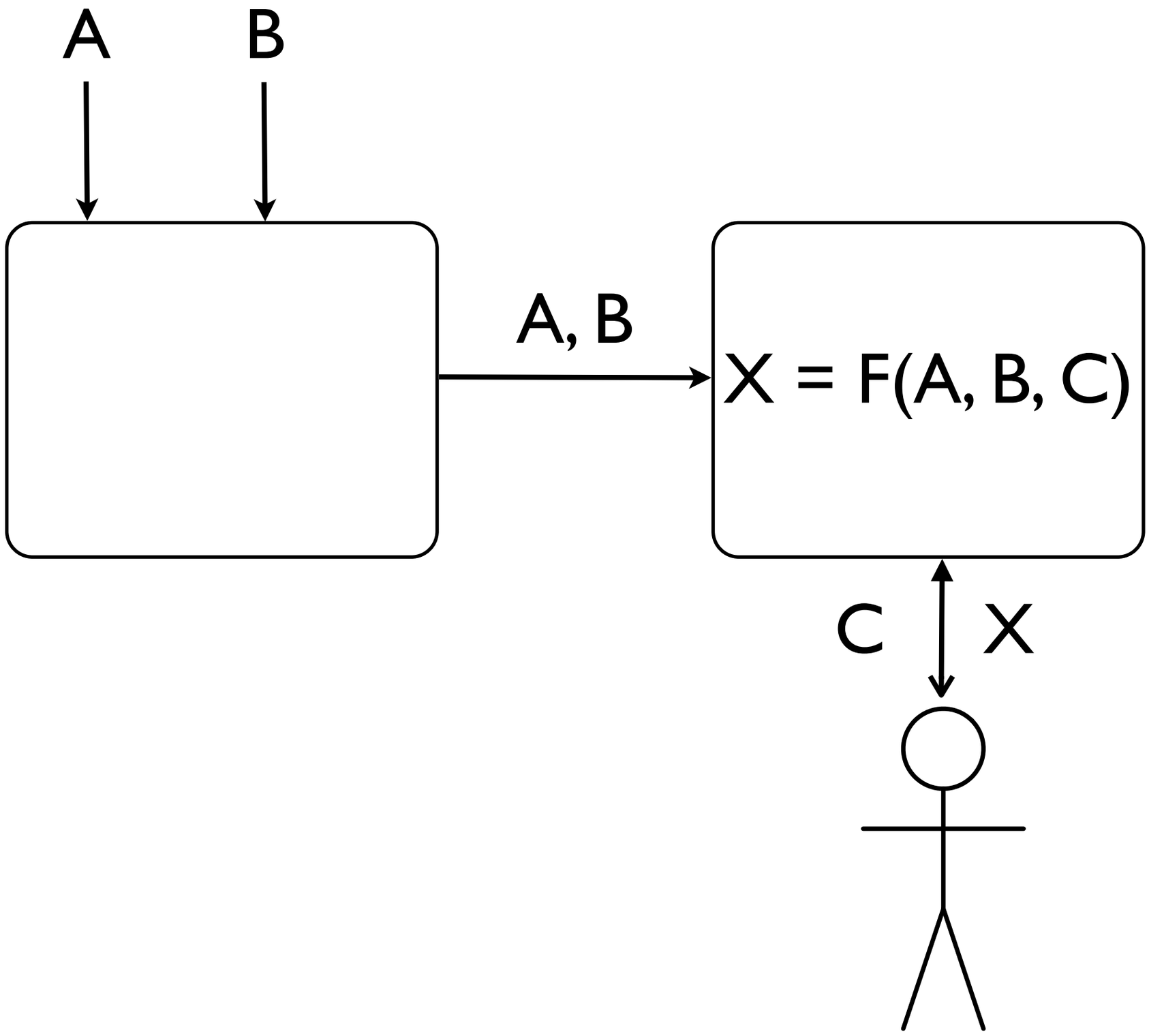
What is a map?

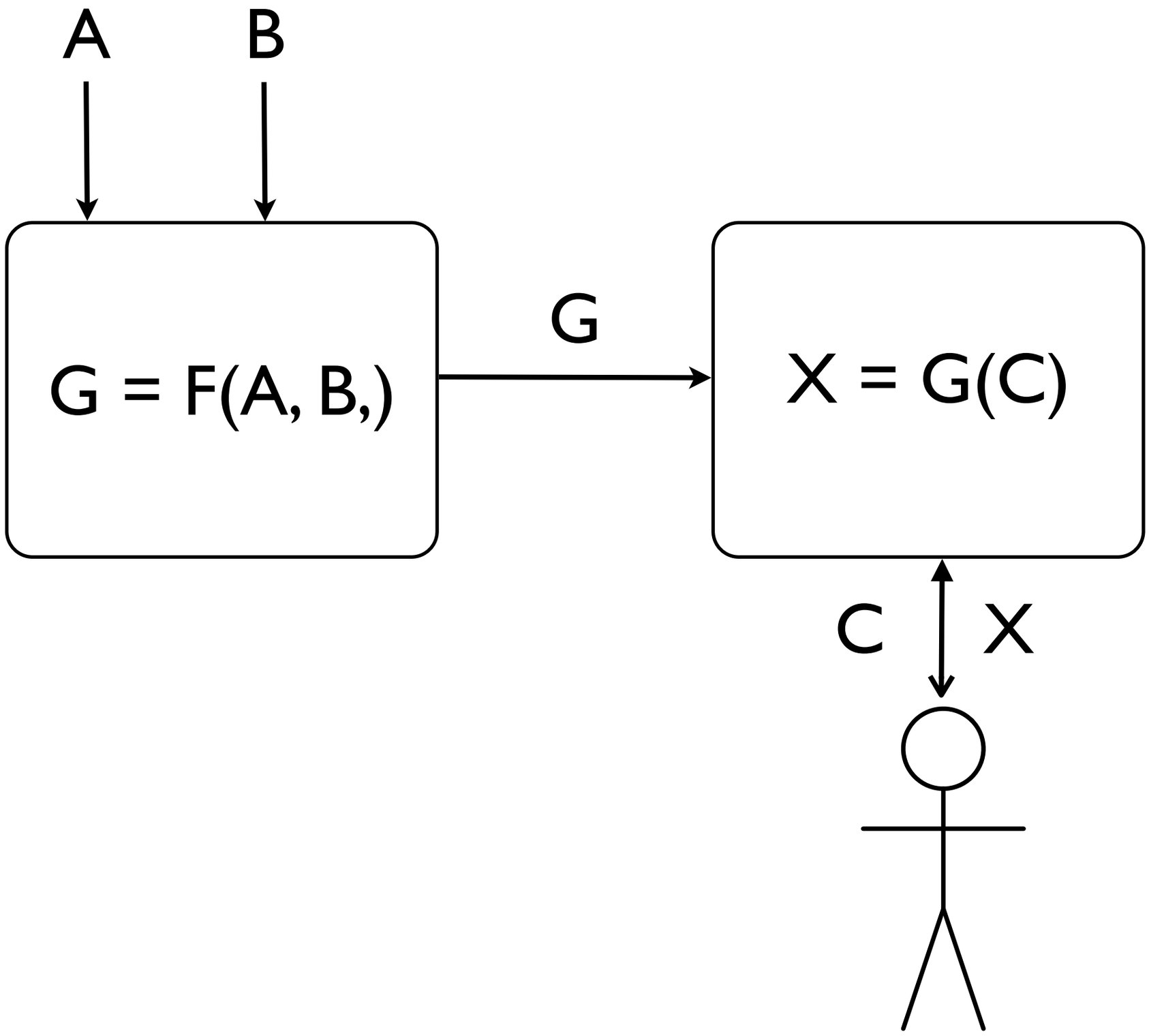
$$(f : \mathrm{K} \longrightarrow \mathrm{V}, \mathrm{S}), \mathrm{S} \subseteq \mathrm{K}$$

```
public interface Function<K, V> {
   V apply(K input);
}
```

```
public static <K, V> Function<K, V> mapToFunc(Map<K, V> map);

public static <K, V> Map<K, V> funcToMap(Function<K, V> func,
                                         Collection<K> keySet);
```

```
public static <K, V> Function<K, V> mapToFunc(Map<K, V> map);

public static <K, V> Map<K, V> funcToMap(Function<K, V> func,
                                         Collection<K> keySet);

public static <K,V> Collection<V> transform(Collection<K> coll,
                                            Function<K, V> func);

public static <V> Collection<V> filter(Collection<V> unfiltered,
                                       Function<V, Boolean> pred);

public static <K,V> Map<V, Collection<V>> groupBy(
                                         Collection<V> coll,
                                         Function<V, K> keyFunc);


public static <K,V> Function<K, V> memoize(Function<K, V> func);
```


A
B
A, B
X = F(A, B, C)
C
X


A
B
G = F(A, B,)
G
X = G(C)
C
X

Representing Functions

Representing Functions

- As code:

```
(fn [x] (+ 2 (* x 3)))
```

Representing Functions

- As code:

```
(fn [x] (+ 2 (* x 3)))
```

- As a set of ordered pairs:

```
#{[0 2], [1 5], [2 8], [3 11]}
```

Representing Functions

- As code:

```
(fn [x] (+ 2 (* x 3)))
```

- As a set of ordered pairs:

```
(into {} #{[0 2], [1 5], [2 8], [3 11]})
```

```clojure
(defn linear [lneighbor
              rneighbor
              exact]
  (f [x]
    (if-let [[_ y] (exact x)]
      y
      (let [[x1 y1] (lneighbor x)
            [x2 y2] (rneighbor x)
            m (/ (- y2 y1) (- x2 x1))]
        (+ (* m (- x x1)) y1)))))
```