Records

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

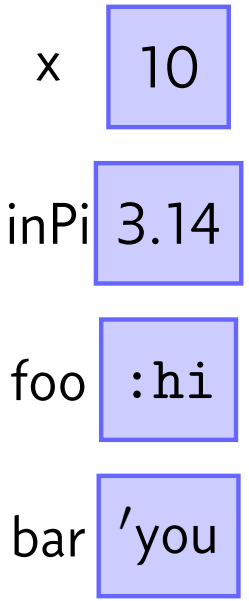
Objectives

- Explain the difference between a scalar and a reference.
  - Categorize CLOJURE expressions as either scalar or reference.
  - Draw memory diagrams for scalars and references.
  - Critique erroneous memory diagrams.
- Understand the syntax and semantics of CLOJURE's records.
  - Define record types.
  - Show how to create an instance of a record.
  - Draw memory diagrams corresponding to CLOJURE code.
  - Write CLOJURE code corresponding to a memory diagram.

Scalars

- A *scalar* is a value that can fit into a single machine word (currently 32 or 64 bits).
- Typical scalars: integers, floats
- Pretend scalars: keywords, symbols
- Memory diagram: put the value in a box.

```
(def x 10)
(def inPi 3.14)
(def foo :hi)
(def bar 'you)
```

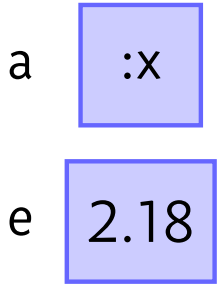


Your Turn

- Draw a memory diagram for the following CLOJURE code:

```
(def a 12.32)
(def b 'x)
```

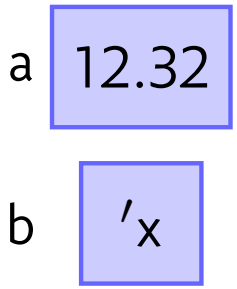
- Write CLOJURE code that will produce the following diagram.



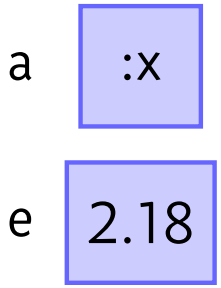
Your Turn

- Draw a memory diagram for the following CLOJURE code:

```
(def a 12.32)
(def b 'x)
```



- Write CLOJURE code that will produce the following diagram.

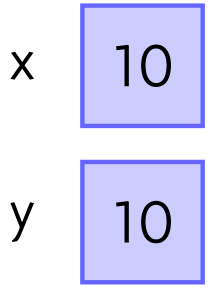



```
(def a :x)
(def e 2.18)
```

Copying

- When a scalar is copied, the second box has the same contents as the first. *The data itself is copied*.

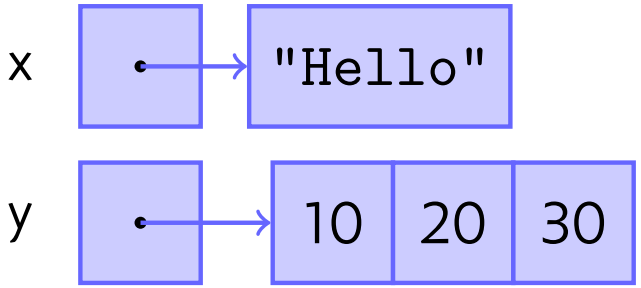
```
(def x 10)
(def y x)
```



References

- If it's too big to fit in a single word, then we use a *reference* (sometimes called a *pointer*) to handle it.
- A reference is drawn as an arrow.
- Two references are equal if their destinations are the same. The source doesn't count.
- Typical references: strings, vectors, hashmaps, records... any compound type.

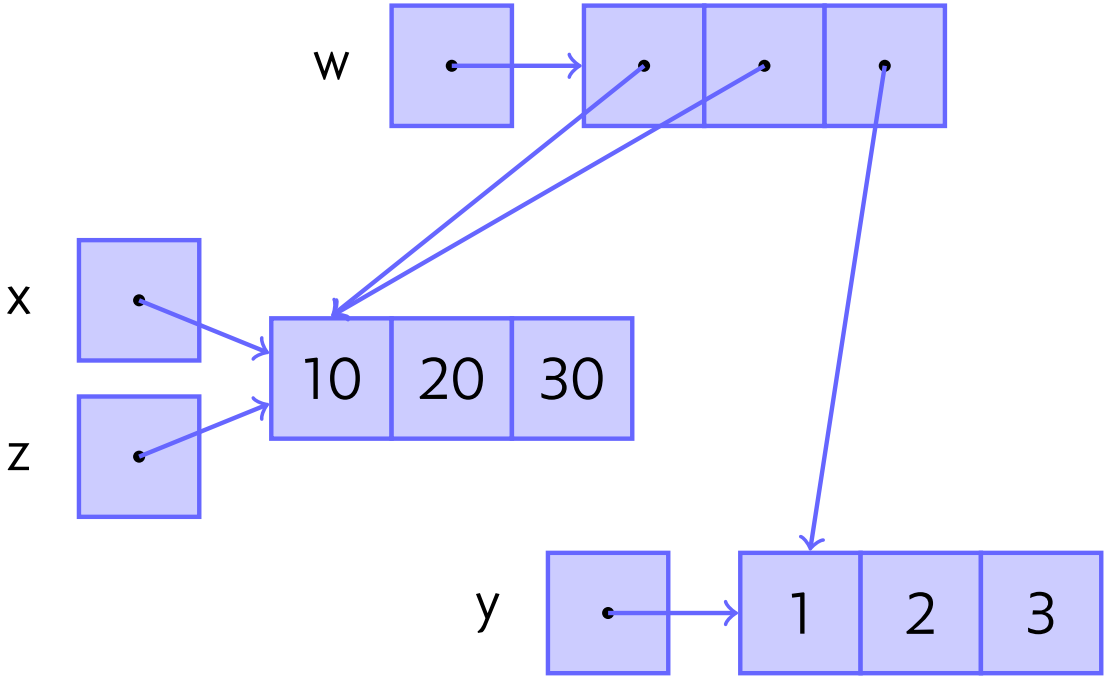
```
(def x "Hello")
(def y [10 20 30])
```



Examples

- When references are copied, we create an arrow with *the same destination*.
- An arrow can only point to a collection as a whole, never an individual element within the collection.

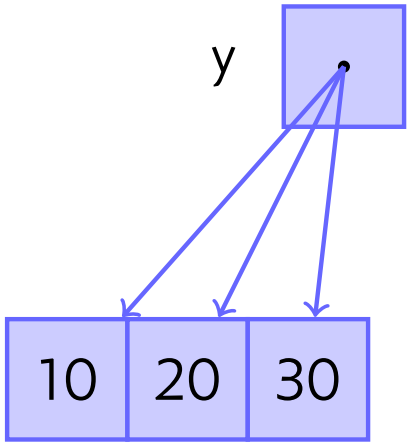
```
(def x [10 20 30])
(def y [1 2 3])
(def z x)
(def w [x z y])
```



Only one destination!

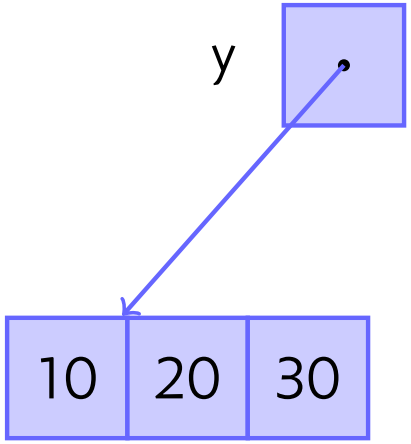
- A pointer can only point to one destination at a time!!!
- **WRONG**

```
(def y [10 20 30])
```



- **RIGHT**

```
(def y [10 20 30])
```



Your Turn 2a

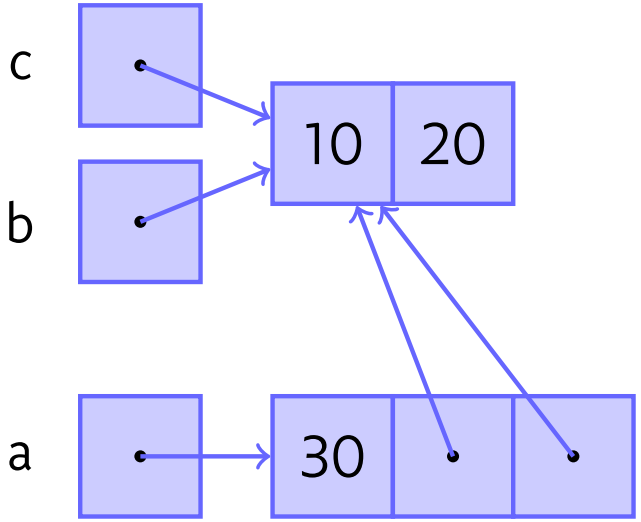
- Draw a memory diagram for the following CLOJURE code:

```
(def c [10 20])
(def b c)
(def a [30 b c])
```

Your Turn 2a

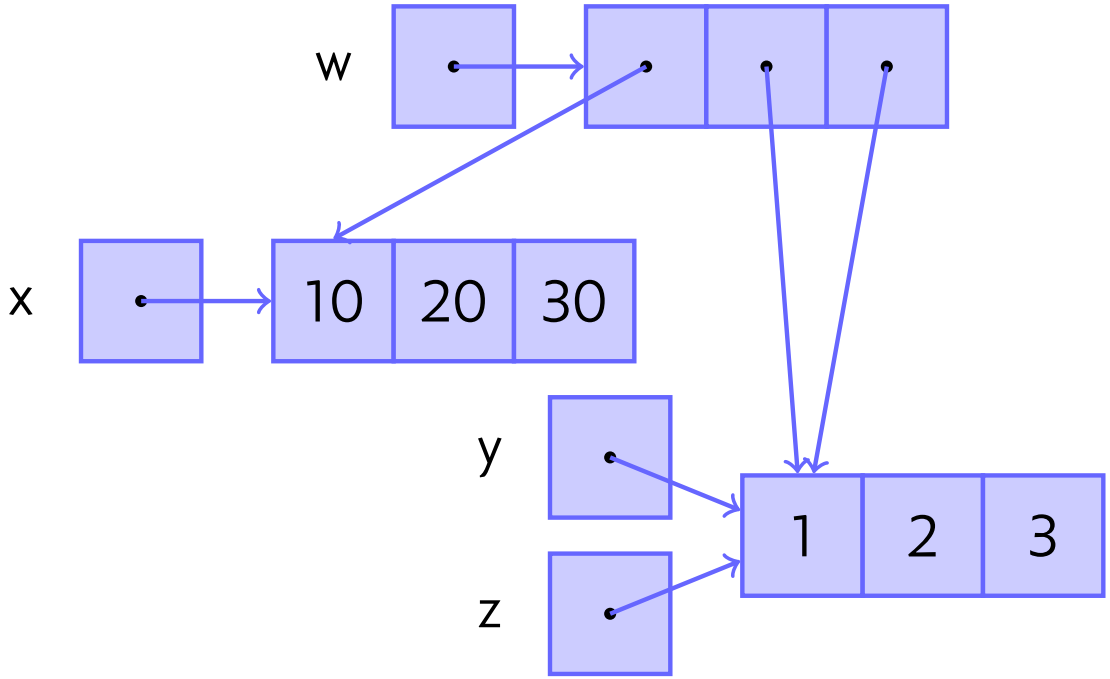
- Draw a memory diagram for the following CLOJURE code:

```
(def c [10 20])
(def b c)
(def a [30 b c])
```



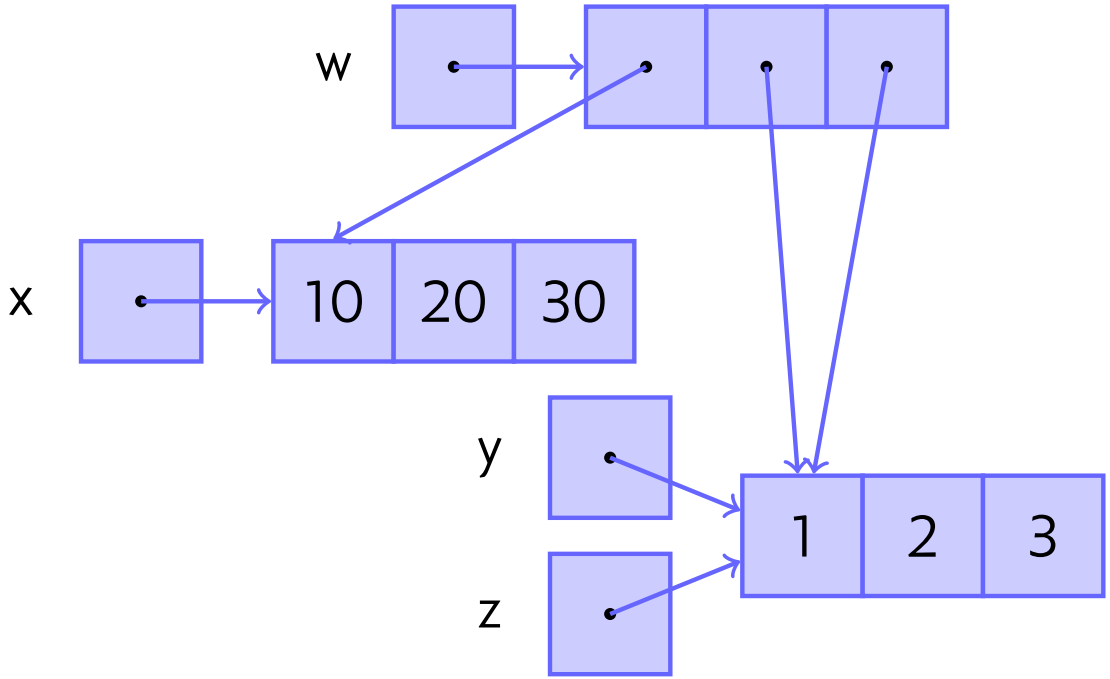
Your Turn 2b

- Write CLOJURE code that will produce the following diagram.



Your Turn 2b

- Write CLOJURE code that will produce the following diagram.




```clojure
(def x [10 20 30])
(def y [1 2 3])
(def z y)
(def w [x y y])
```

Defining a Record

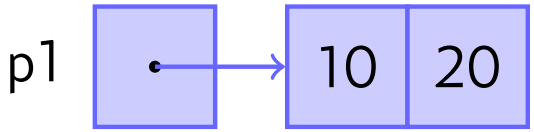
```
(defrecord name [fields*] specs*)
```

- The name can be any legal CLOJURE name. We usually capitalize the first letter.
- The fields are usually lower case, as other CLOJURE variables.
- We will discuss specs in a future lecture.
- This creates an actual Java class!

Example

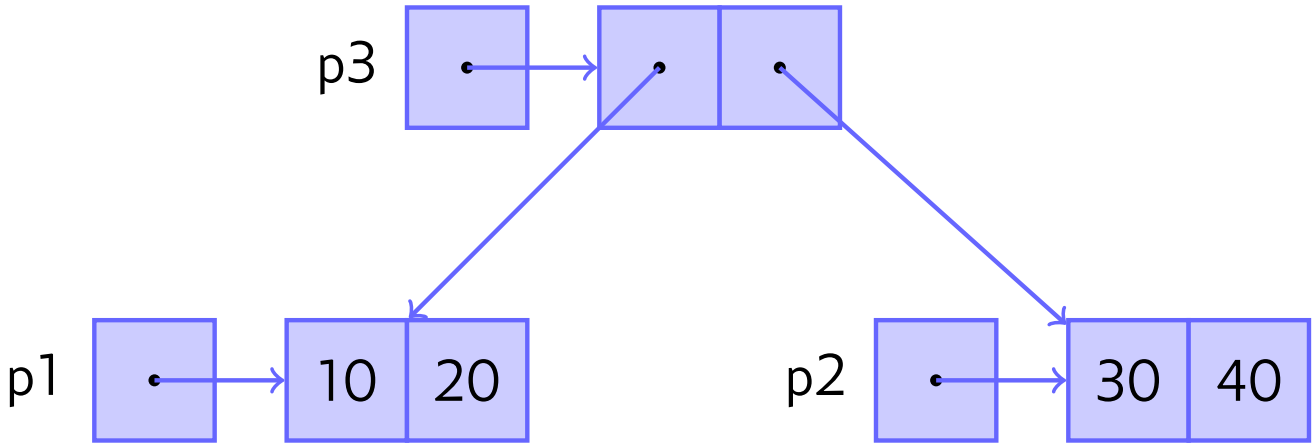
- Create an instance using the name followed by a dot.
- Keywords become field lookup functions, like in hash maps.

```
user> (defrecord Pair [x y])
user.Pair
user> (def p1 (Pair. 10 20))
#'user/p1
user> p1
#user.Pair{:x 10, :y 20}
user> (:x p1)
10
```



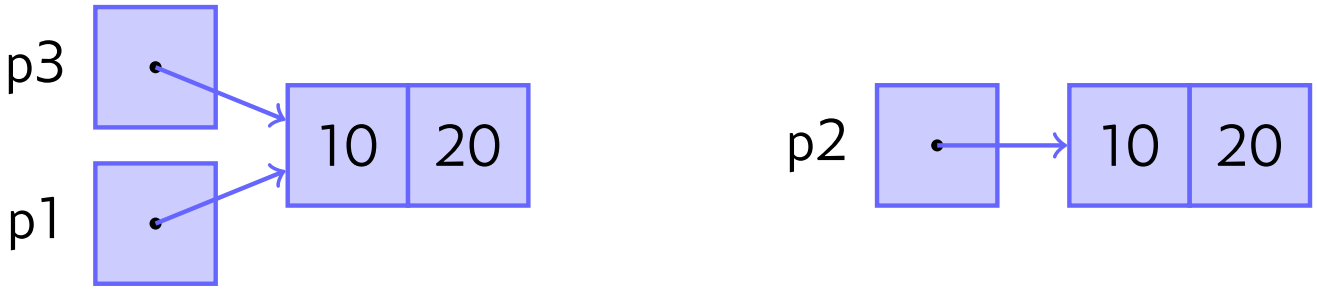
scalars vs references

```
user> (def p1 (Pair. 10 20))
user> (def p2 (Pair. 30 40))
user> (def p3 (Pair. p1 p2))
```



Equality

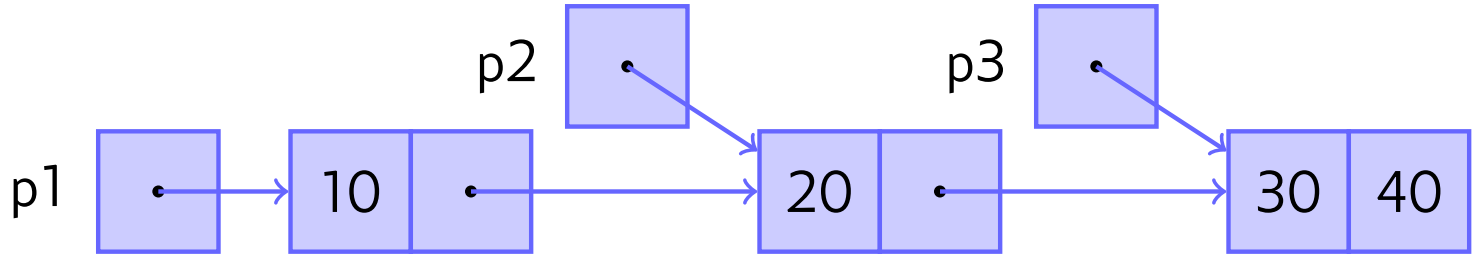
```
user> (def p1 (Pair. 10 20))
user> (def p2 (Pair. 10 20))
user> (def p3 p1)
```




```
user> (= p1 p2)
true
user> (= p1 p3)
true
user> (identical? p1 p2)
false
user> (identical? p1 p3)
true
```

the -> macro

```
user> (def p3 (Pair. 30 40))
user> (def p2 (Pair. 20 p3))
user> (def p1 (Pair. 10 p2))
```




```
user> (:x (:y (:y p1)))
30
user> (-> p1 :y :y :x)
30
```

Examples

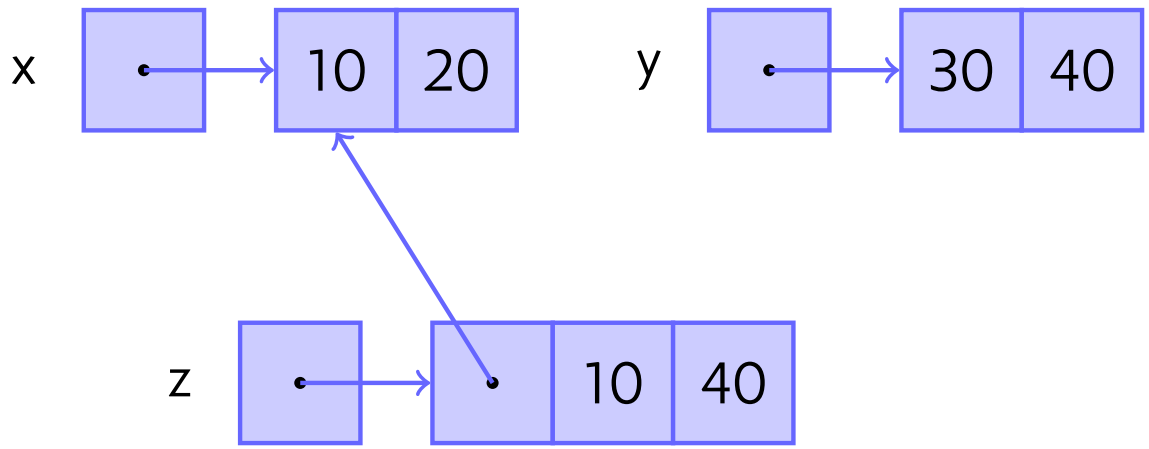
- What do you think this code will do?

```
(defrecord Triple [a b c])
(def x (Double. 10 20))
(def y (Double. 30 40))
(def z (Triple. x (:x x) (:y y)))
```

Examples

- What do you think this code will do?

```
(defrecord Triple [a b c])
(def x (Double. 10 20))
(def y (Double. 30 40))
(def z (Triple. x (:x x) (:y y)))
```



Examples

- What do you think this code will do?

```
(defrecord Triple [a b c])
(def x (Double. 10 20))
(def y (Double. x 40))
(def z (Triple. x (:x y) y))
```

Examples

- What do you think this code will do?

```
(defrecord Triple [a b c])
(def x (Double. 10 20))
(def y (Double. x 40))
(def z (Triple. x (:x y) y))
```