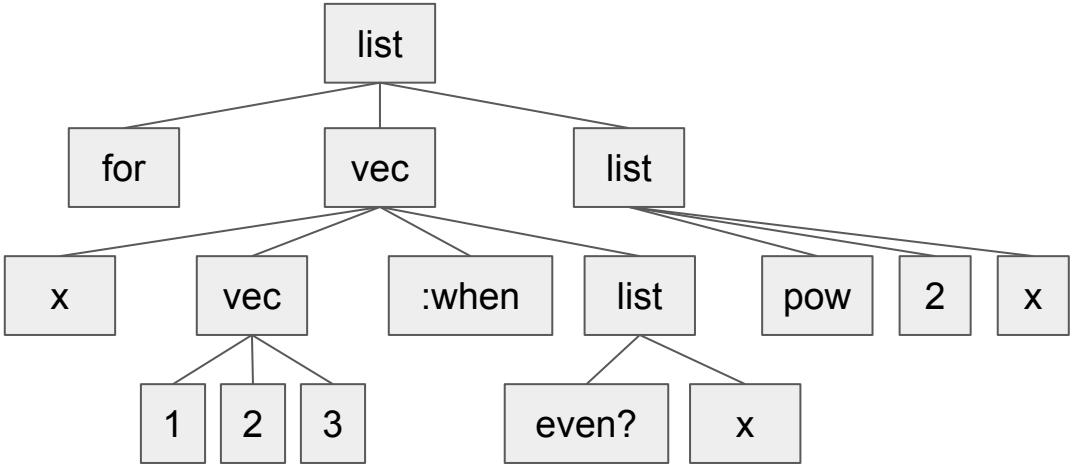
Introduction to Macros

Ken Q Pu, Faculty of Science, Ontario Tech University

Homoiconicity of Clojure

Source code of Clojure is entirely described as data structures supported by Clojure.

```
(for [x [1 2 3]
      :when (even? x)]
  (pow 2 x))
```

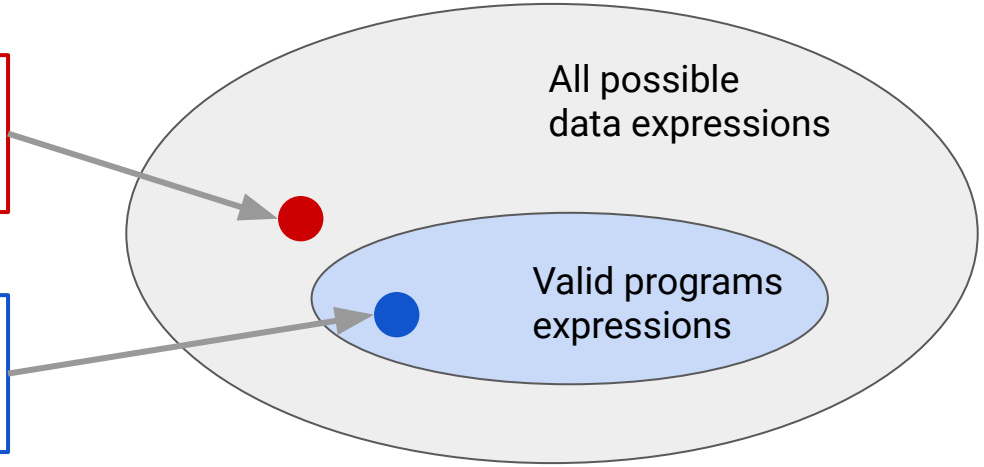


Valid Clojure programs

Not all data structures are valid programs.

```
(foreach x :from 0 :to 10]
  (return x/2 + 1))
```

```
(for [x (range 0 10)]
  (+ (/ x 2) 1))
```

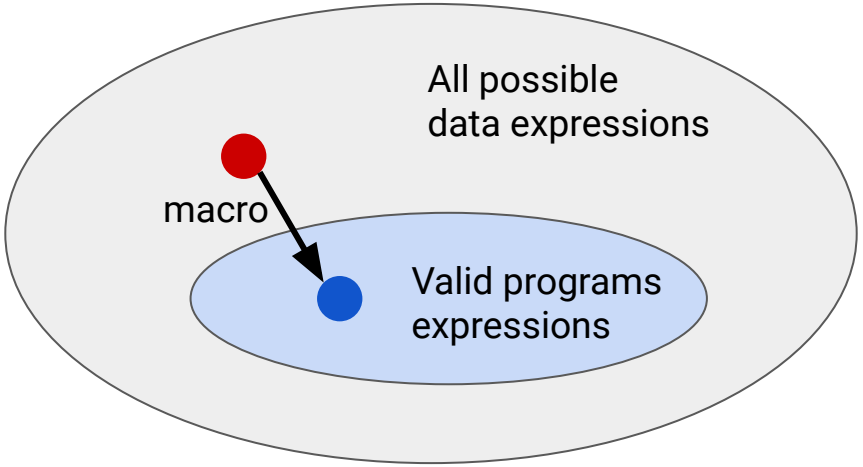


Code transformation and generation

Clojure's computing model is based on data transformation.

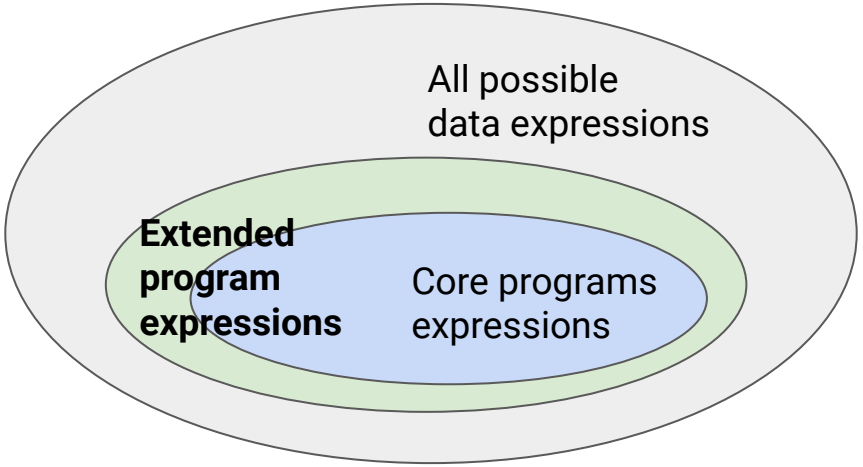
Since programs is expressed as data structures, we can build generators of ***valid*** Clojure programs.

These generators are known as ***macros***.



Why macros?

With macros, we give control to the programmers to decide what's a valid program expression in the sea of all possible data expressions.



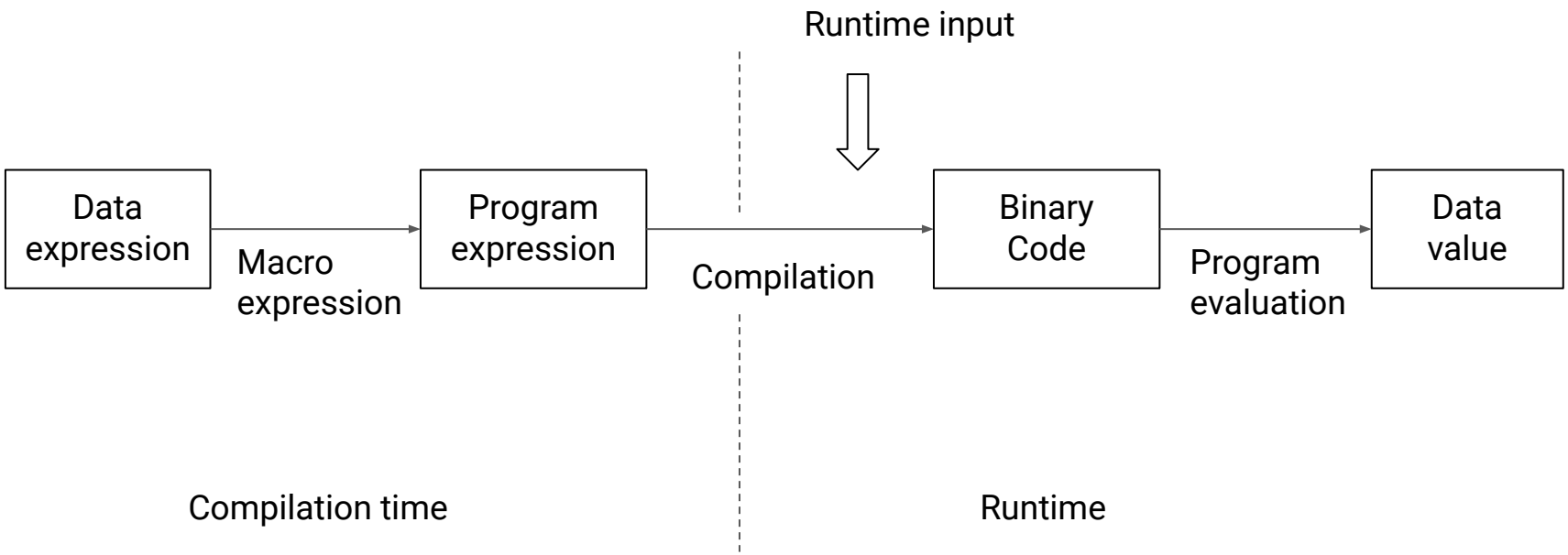
Why macros

```
(foreach x :from 0 :to 10]
  (return x/2 + 1))
```

✔

```
(for [x (range 0 10)]
  (+ (/ x 2) 1))
```

Macro Expansion

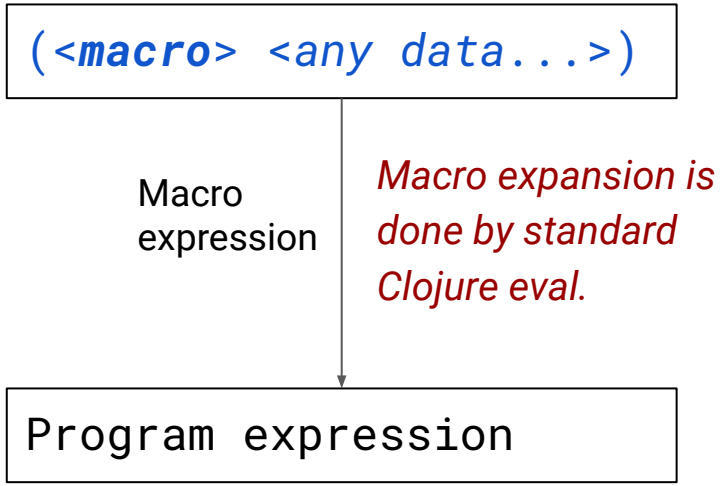


Macros

A macro is a *special function* that is executed during macro expansion.

```
(defmacro <macro> [parameters...]
 <body>)
```

A macro must be a valid Clojure function that returns Clojure program data structures.



Macro magic

Macro expansion is not program evaluation.

During macro expansion, arguments are *not evaluated*.

Macro expansion:

```
(str (+ 1 2 3))

⇒ "(+ 1 2 3")
```

Program evaluation:

```
(str (+ 1 2 3))

⇒ (str 6)
⇒ "6"
```

Macro magic

Since the arguments of macro functions are not evaluated, they can be arbitrary data expressions.

It's up to the macro function to convert them to valid Clojure program expressions.

```
(what-is (1 + 2))
```



```
(+ 1 2)
```

```
(defmacro what-is [form]
 (let [[x op y] form])
  (list op x y)))
```

Macro programming

- Macros are functions
  - input are arbitrary data expression that are **not** evaluated.
  - output is a valid Clojure program expression.
- Macros do not concern itself with computation on the actual data, but the **program** that will perform computation during run-time.
- Macros are executed **before** compilation. So, the generated binaries will not contain any macro code.

Example

```
(defmacro with-log
 [form]
 (let [msg (str form)]
  (list 'do
        (list 'println msg)
        form))))
```

Clojure provides many syntactic features to help with writing readable and maintainable macros.

- *Templates*
- *Symbol generation*

```
(let [x (with-log (+ 1 2))]
 (println "2^x =" (pow 2 x)))
```

```
(+ 1 2)
2^x = 8
```