Matters of Complexity

- Determining the worst-case asymptotic run-time for linear-recursive procedures is generally straightforward
  - It's pretty much always O(f(*length of the list*)), for some function f
  - Most frequently, $f(n)=n$, $f(n)=n^2$
- But for star-recursive procedures, things aren't so obvious

Matters of Complexity (cont.)

- A major issue lies in trying to characterize just how much work it is to go through all the levels of all the elements of a list
- Obviously the length of the list and the depth of the list both matter, but it’s more complicated than that

Examples

- `(a b c d e f g h)`
  - length 8, depth 1
  - So length matters
- `((((((((a))))))))`
  - length 1, depth 8
  - So depth matters
- `((a b c d e f g h))`
  - length 1, depth 2
  - So length and depth of the list alone can’t be enough

Examples (cont.)

- `((a b c d e f g h))`
  - length 1, depth 2
  - So length and depth of the list alone can't be enough
- `(((() ()) () () (() (() (())))))`
  - So number of atoms alone can't be enough

The Big Thing

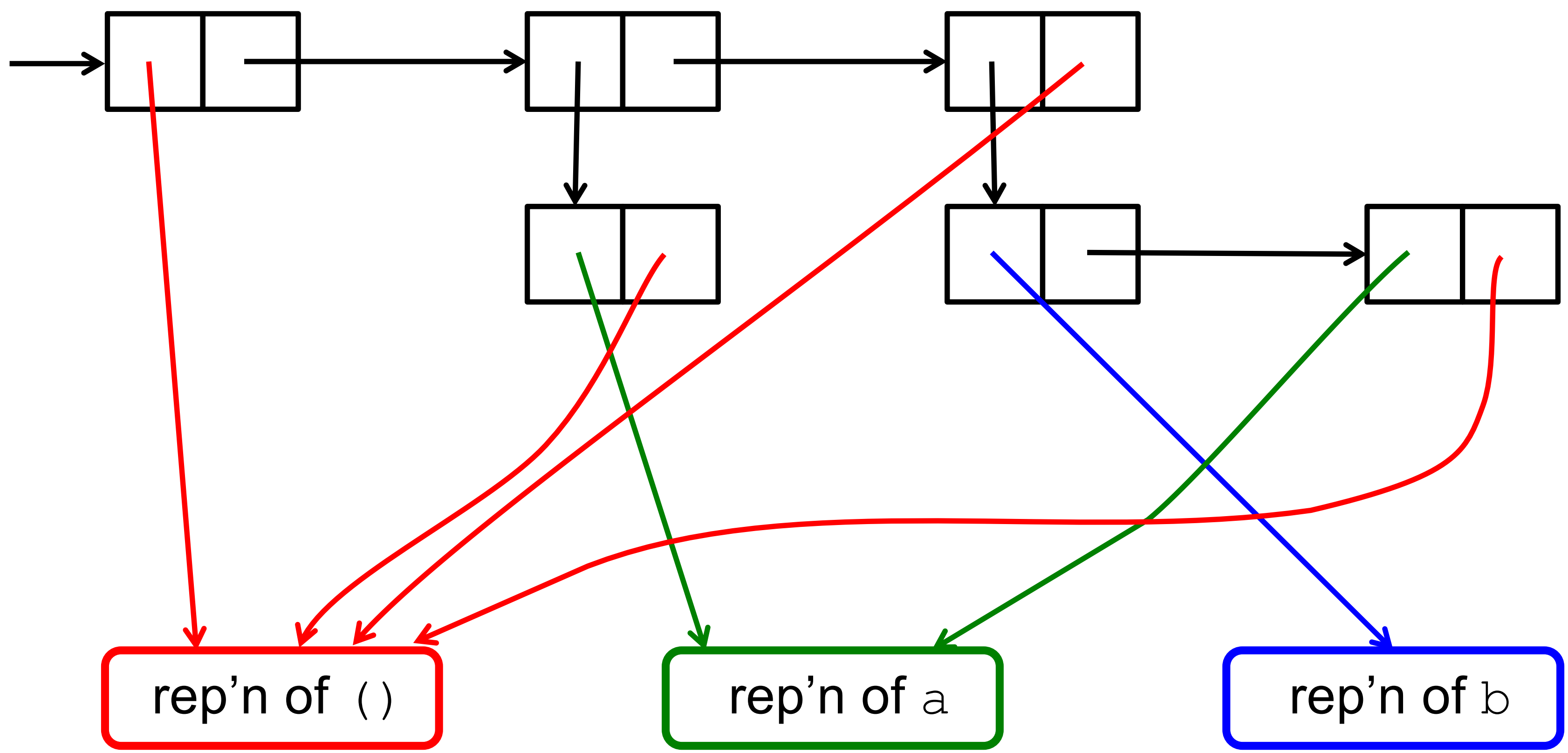
- A critical first step is to get a handle on just how much “stuff” it takes to represent a list internally in Scheme

Scheme List Structure

- Lists are represented internally *pairs* or *cons cells*
- We can sketch an approximation of pairs and other select Scheme values as Java classes
  - *See the related Java source files*

Scheme List Structure (cont.)

- So, for example, the list `(() (a) (b a))` would be represented with six pairs as



Complexity Again

- So, determining the worst-case asymptotic run-time for star-recursive procedures then generally becomes something like
  - O(*f*(*number of pairs in the rep'n of the list*)), for some function *f*