Strict vs ~~Lazy~~Non-strict

Whacha doin'? Plain' Shroedinger's cat in the box again??
Either
No, this is a thunk, not box! And I'm either dead or alive inside it. To check you need to evaluate the thunk!
Either
Noooo... I'm too lazy !
Eith

Cartesian Closed Comic: Schrdinger's cat

Evaluation strategies

Strict - Eager evaluation

Non-strict - Lazy evaluation

Evaluation strategies
- Define how expressions are evaluated in a prg lang

Strict

expression is evaluated as soon as it gets bound to a variable

e.g. most imperative prg langs, SML (functional)

Order is enforced

Easier to reason about program as order of evaluation is known (unlike demand-driven)

Lazy

only evaluate an expression when its results are needed

e.g. Haskell

Impl: uses thunks (pointers to code which are replaced with a value the first time they are executed)

- Laziness has its cost - extra bookkeeping required to delay evaluation (introduce a thunk for every expr)

- Hard to predict space behaviour of lazy programs

Soln: add strict features e.g. seq

+ avoid unnecessary calculations / error conditions

+ infinite data structures

Debate!

Nothing like all-lazy or all-strict language

Strict lang e.g. Occaml - lazy keyword to defer evaluation. strict use some form of non-strict evaluation for boolean expressions and if-statements

Lazy lang e.g. Haskell - strictness annotation to enforce certain evaluation order and degree (seq, $!, BangPatterns ext (!))

The Questions:

Which is the right default?

And how does this help in parallel prg?