

How to *Avoid* Bad Behavior

Model Integration, LLC

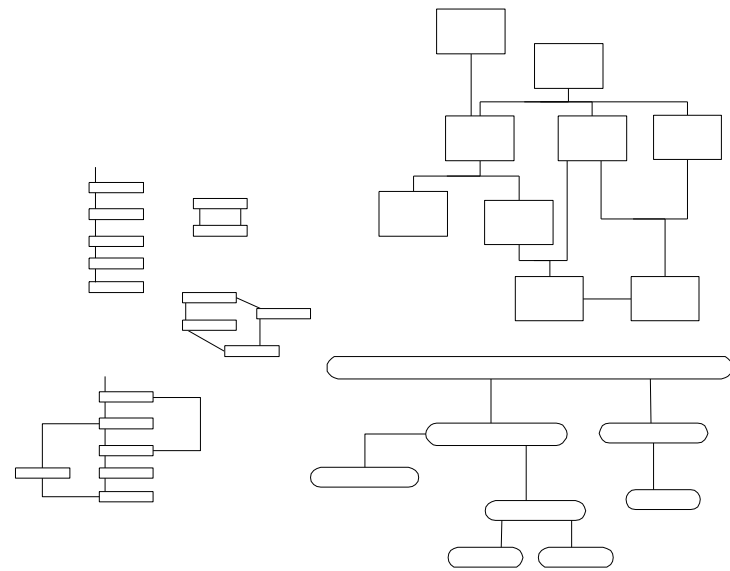
www.modelint.com



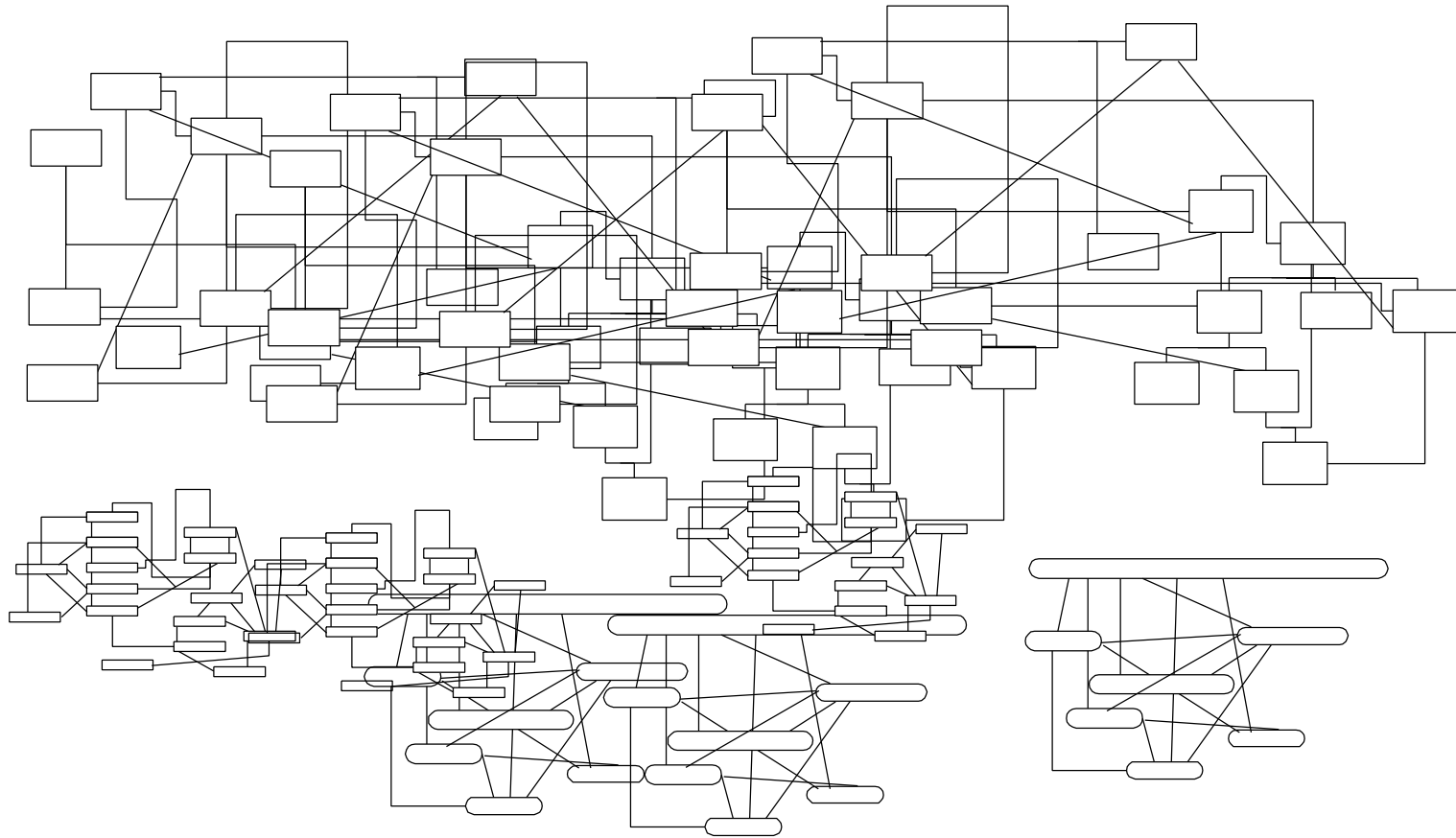
M O D E L I N T E G R A T I O N

Models in the classroom

- ◆ Well formed objects
- ◆ Simple control layering
- ◆ Easy to follow control threads
- ◆ Lifecycles in each state model
- ◆ Intuitive, for the most part



Models on a real project



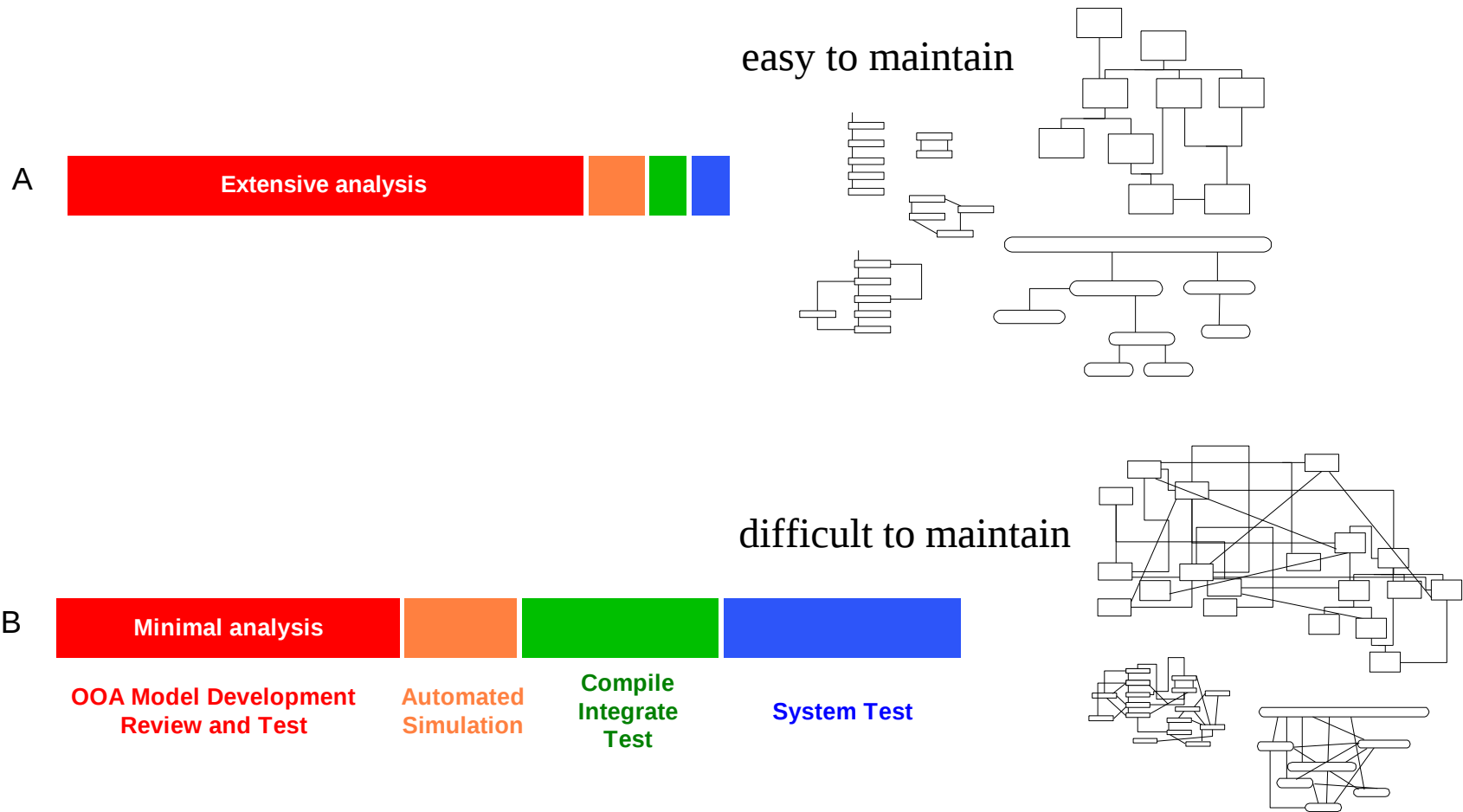
What's going wrong?

- ◆ Greater complexity?
- ◆ Limits of the modeling technology?
- ◆ Lack of analyst experience?
- ◆ Schedule pressure?
- ◆ Dementia?

Negative consequences

- ◆ Frequent recompilation of models
- ◆ Difficult to understand
- ◆ Constant debugging and simulation
- ◆ Hard to extend and maintain
- ◆ Difficult to integrate
- ◆ Impossible to schedule or control
- ◆ Inefficient code

Build the system better and faster

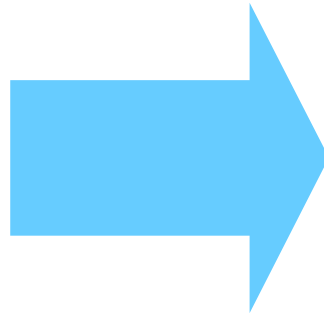
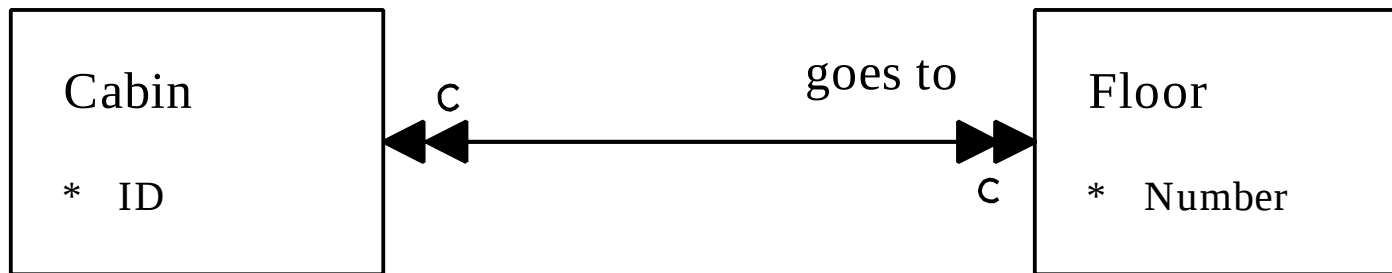


Bad patterns lead to disaster

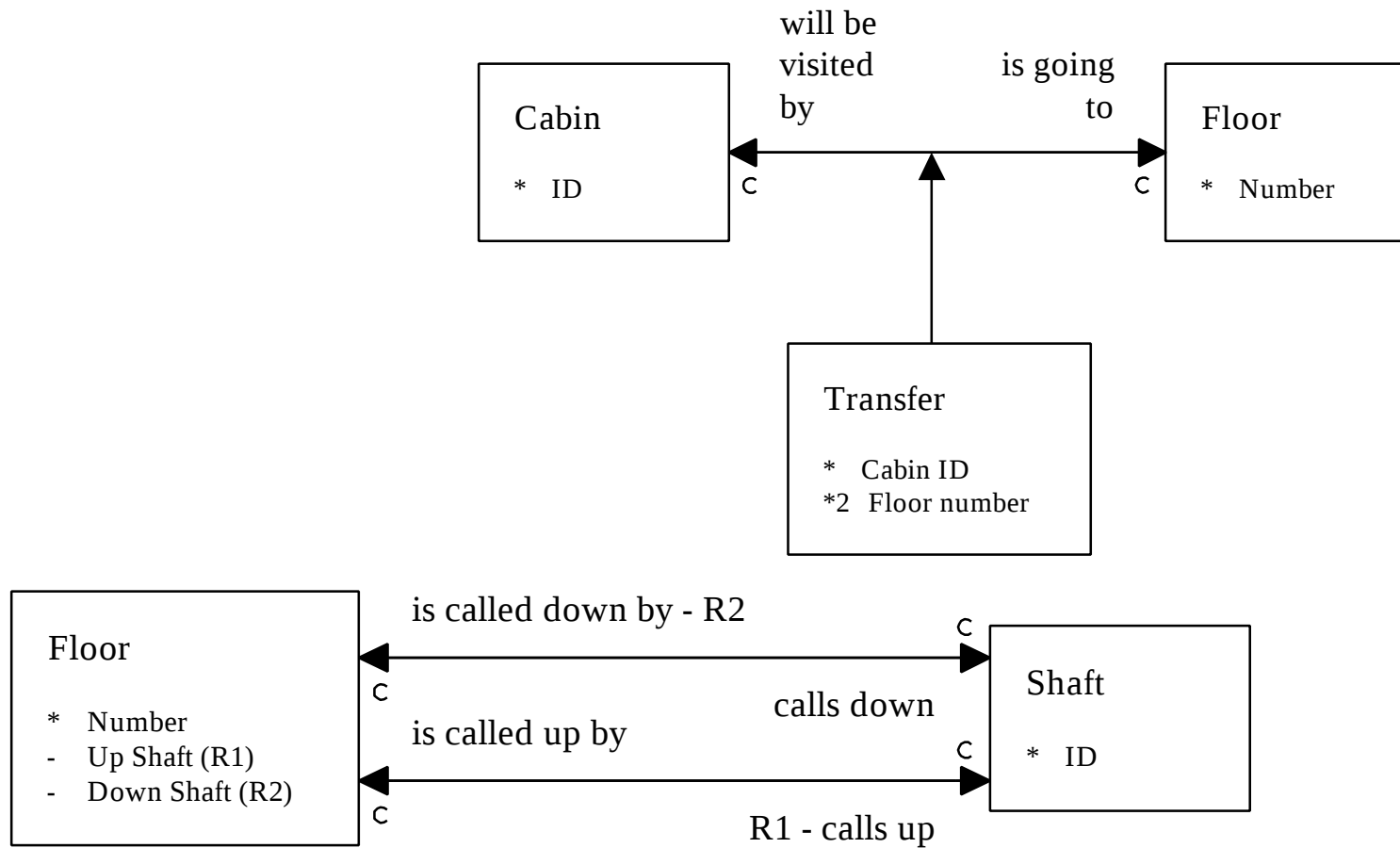
- ◆ Bad object model patterns lay the foundation for complex behavior models
- ◆ Bad state model patterns lead to excessive object communication and out-of-control threads
- ◆ After which there's no hope for the action language!

Bad Object Patterns

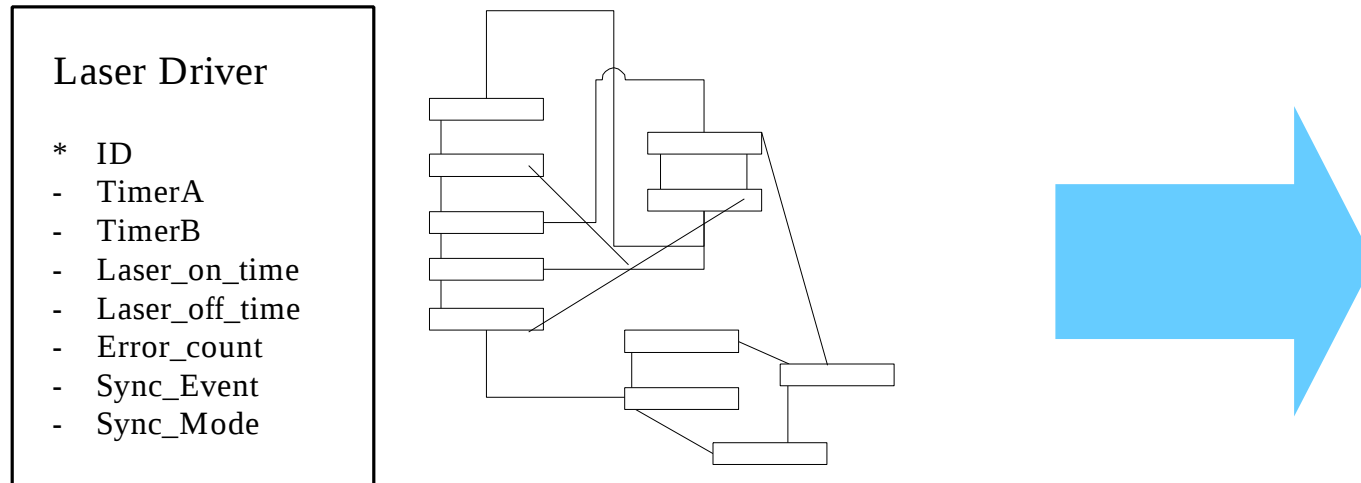
Lack of precision vs.



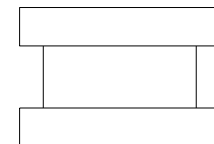
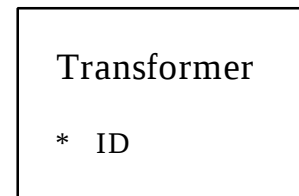
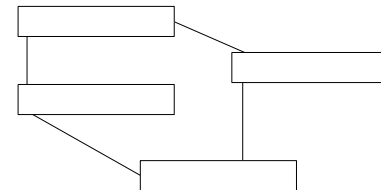
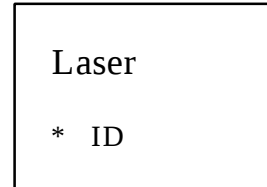
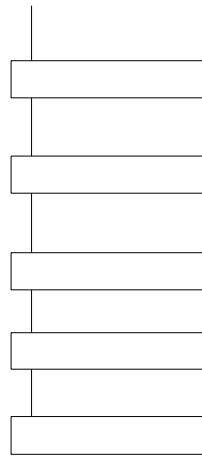
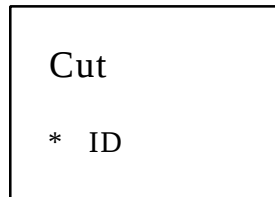
Precision!



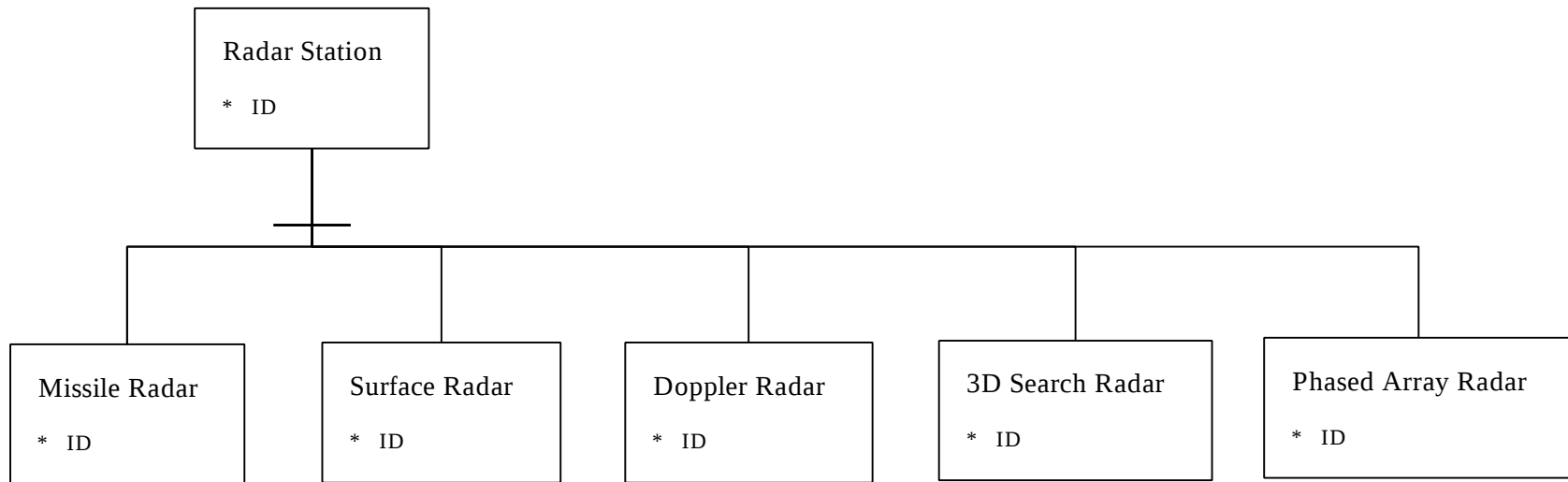
Functions disguised as objects vs.



... the real objects!

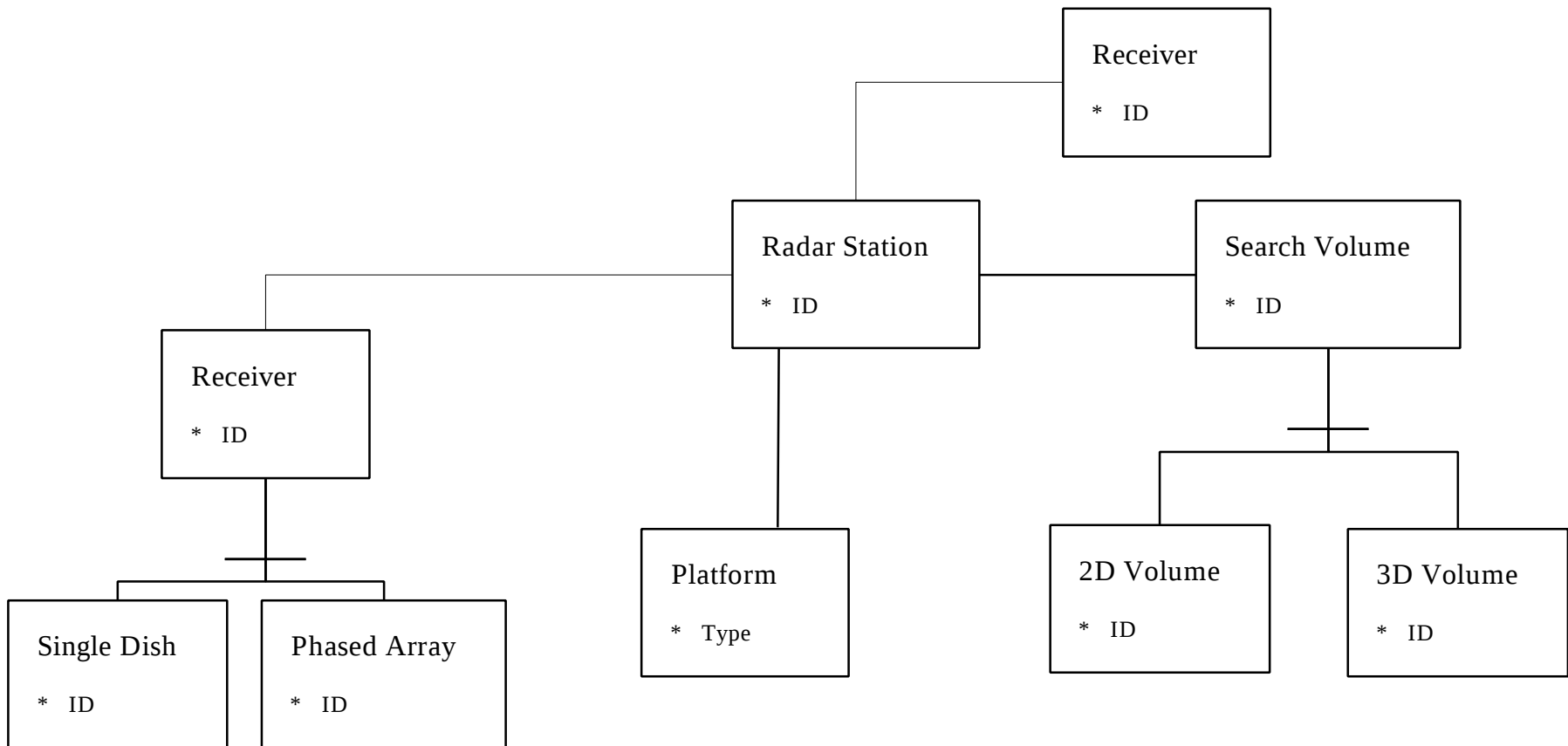


Sloppy subtyping vs.



→
ad nauseum

Stable Subtypes!

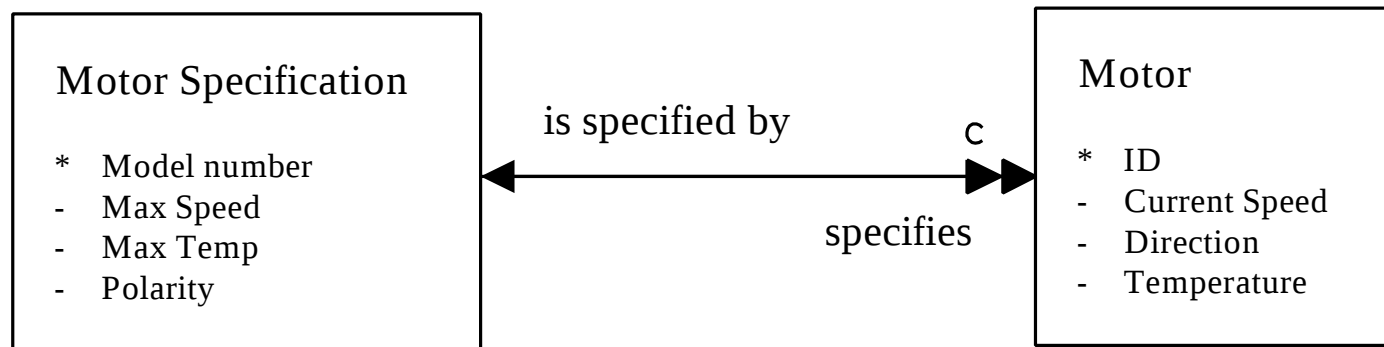


Mixed or missing specifications

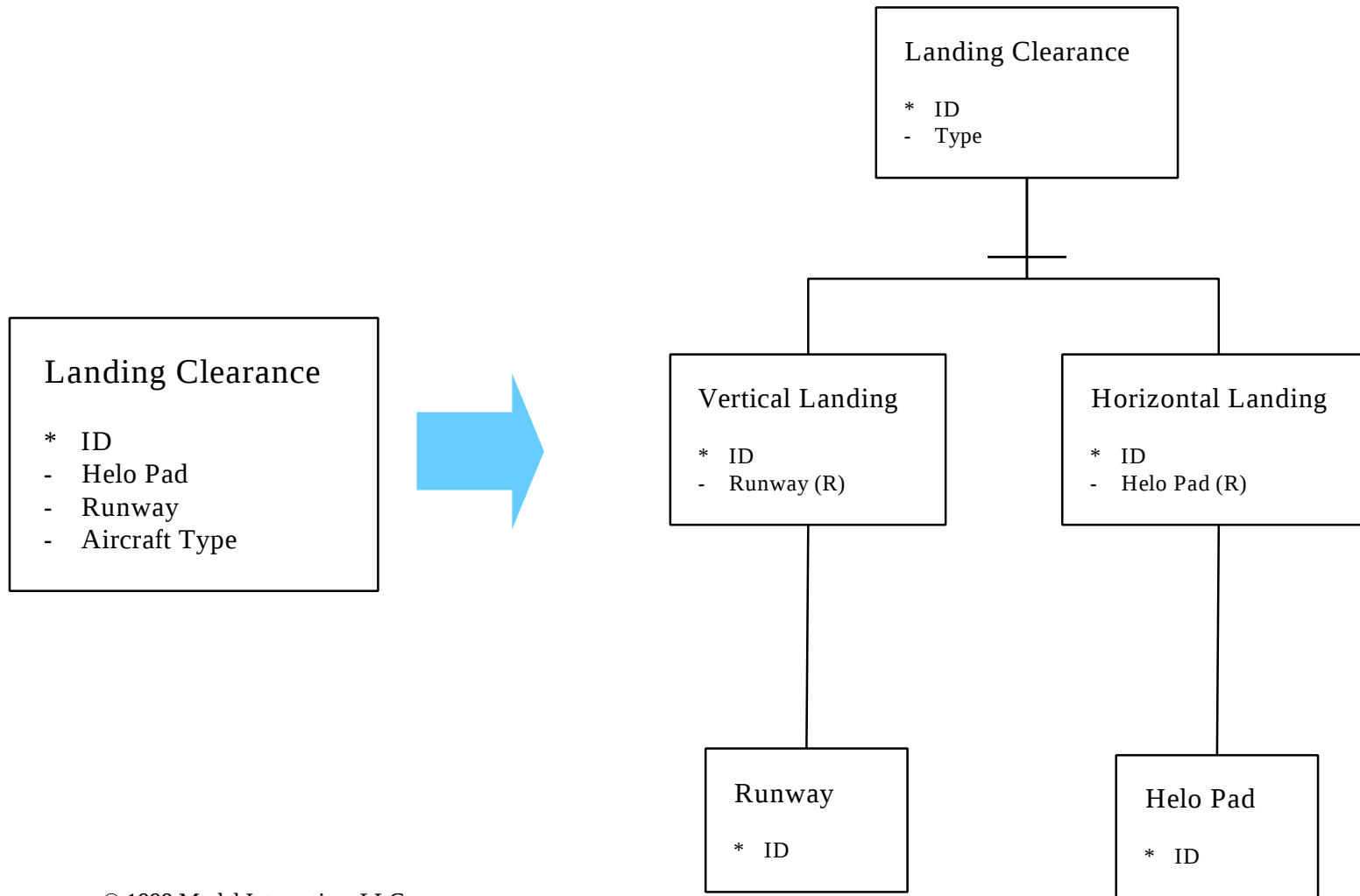
Motor

- * ID
- Current Speed
- Direction
- Temperature
- Max Speed
- Max Temp
- Polarity

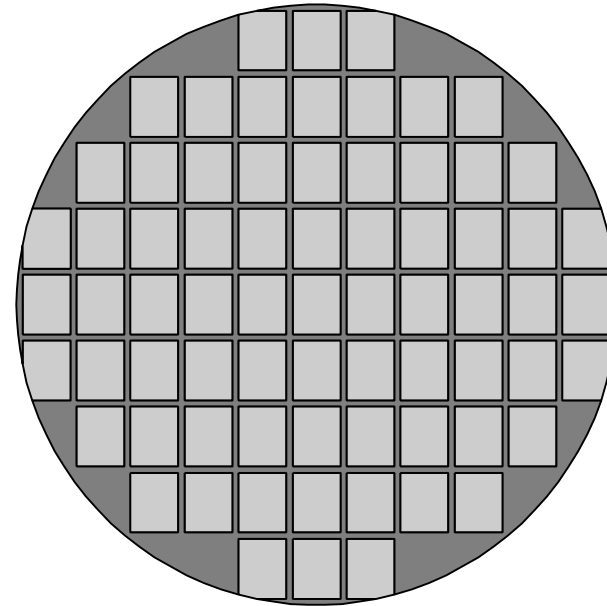
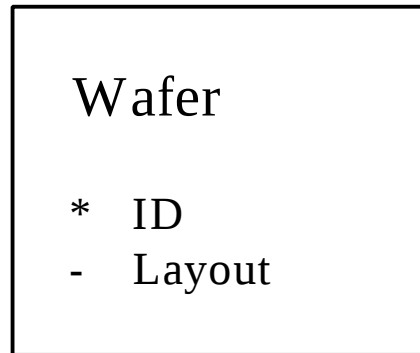
Found!



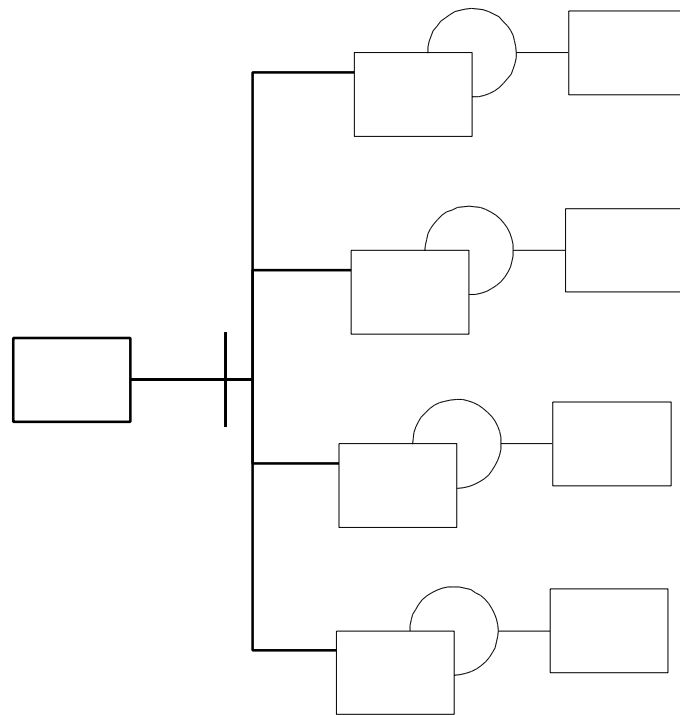
Interdependent attributes



Hidden data structure

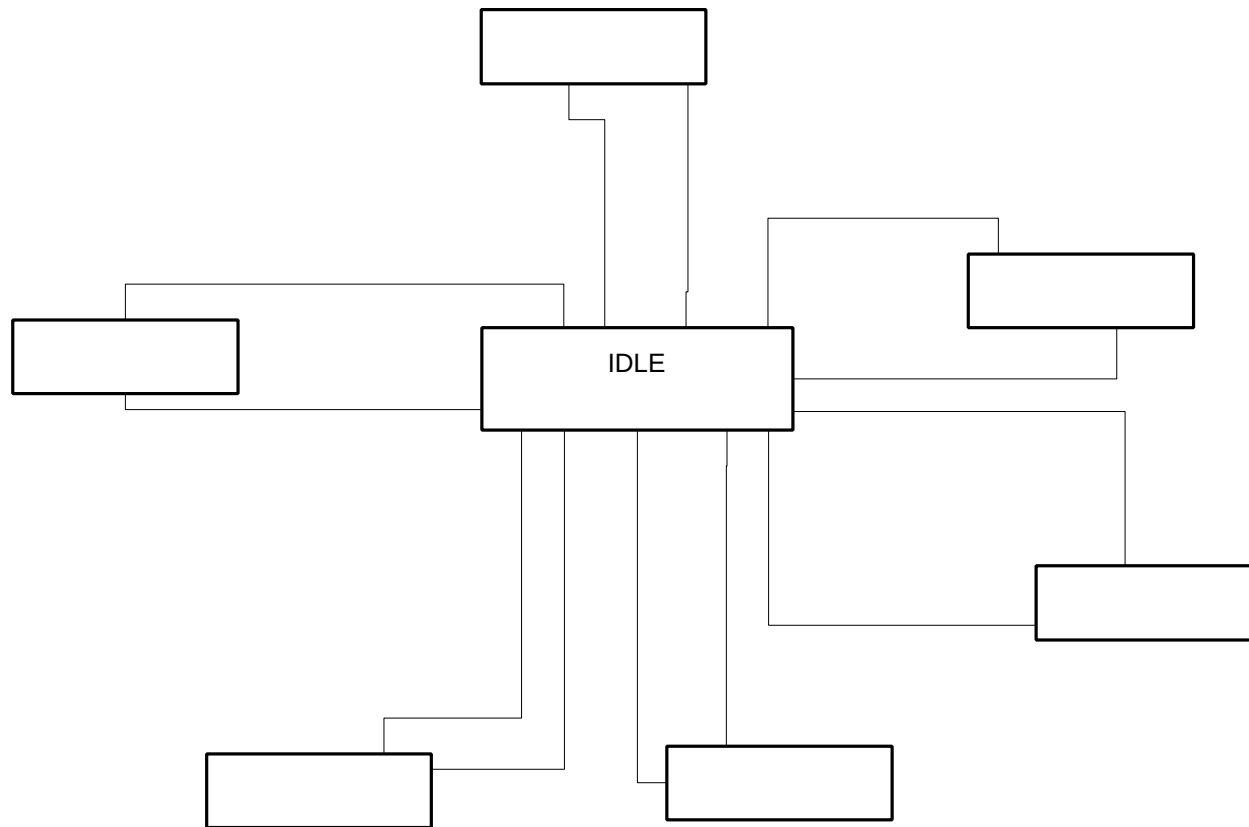


Cut and paste relationships

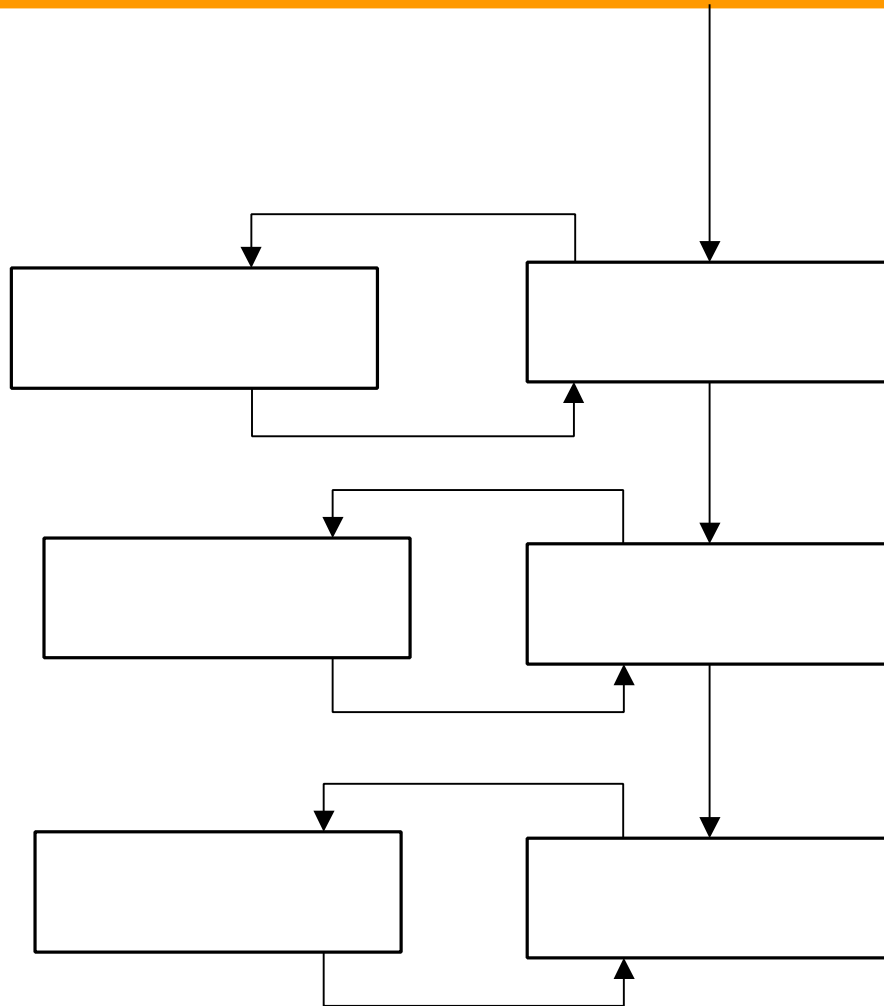


Bad State Patterns

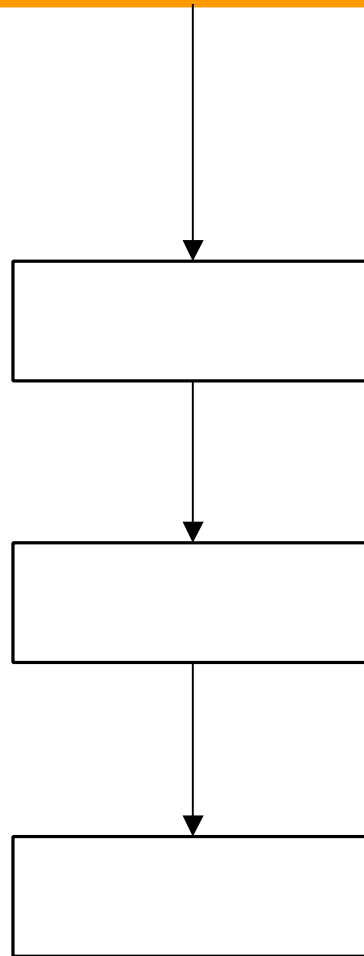
Spider state models



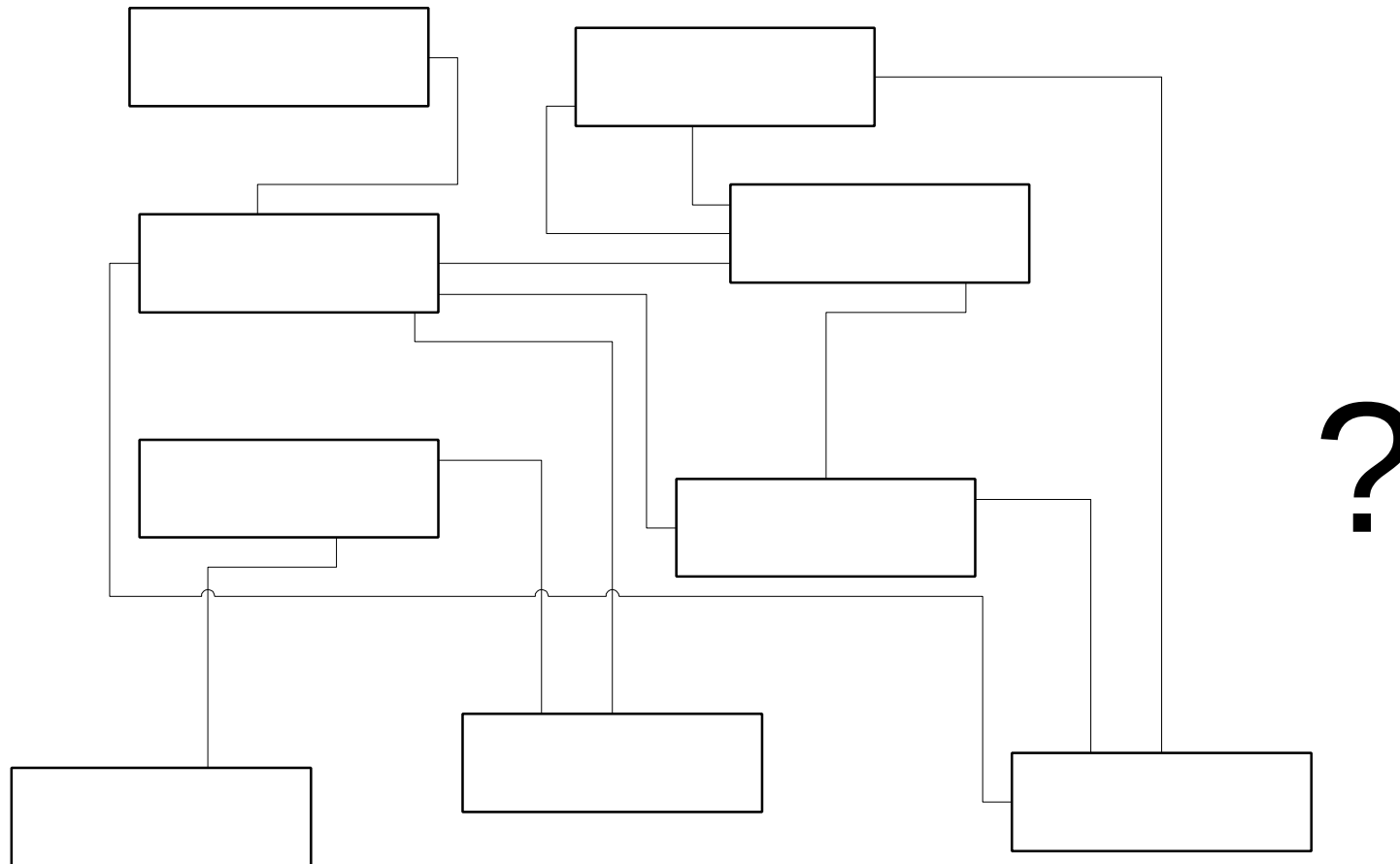
Millipede state models



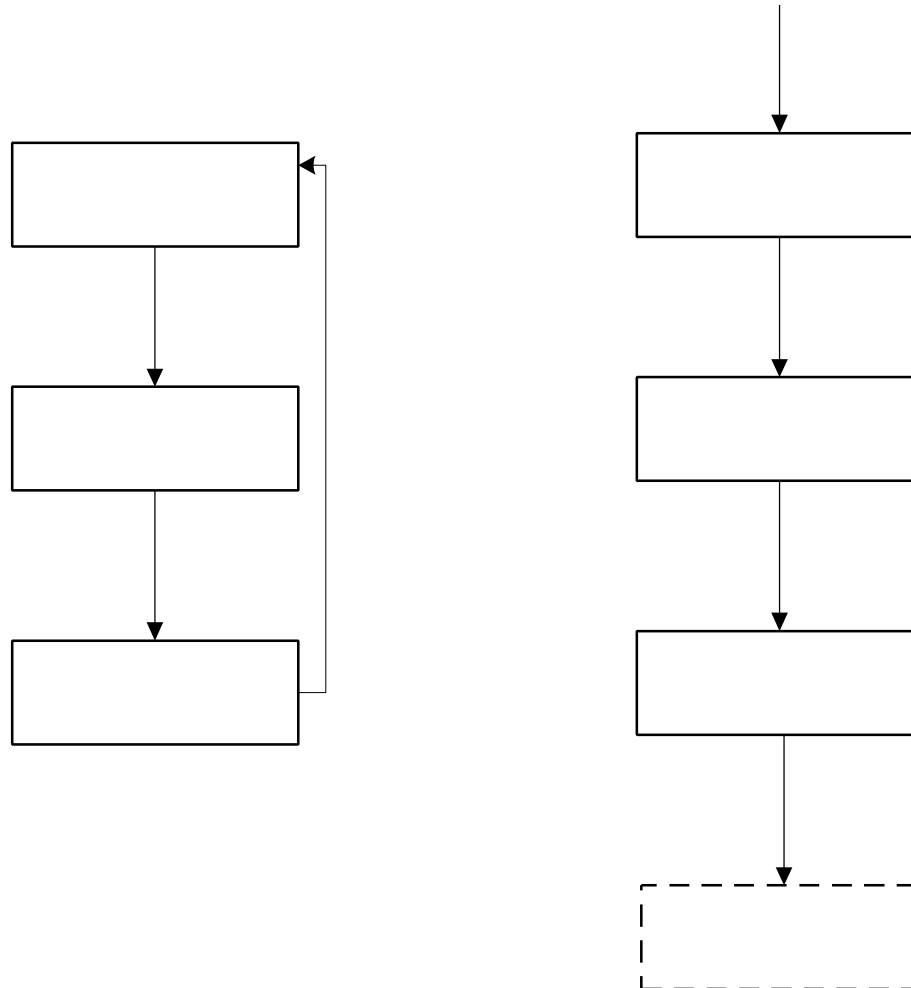
Squish them!



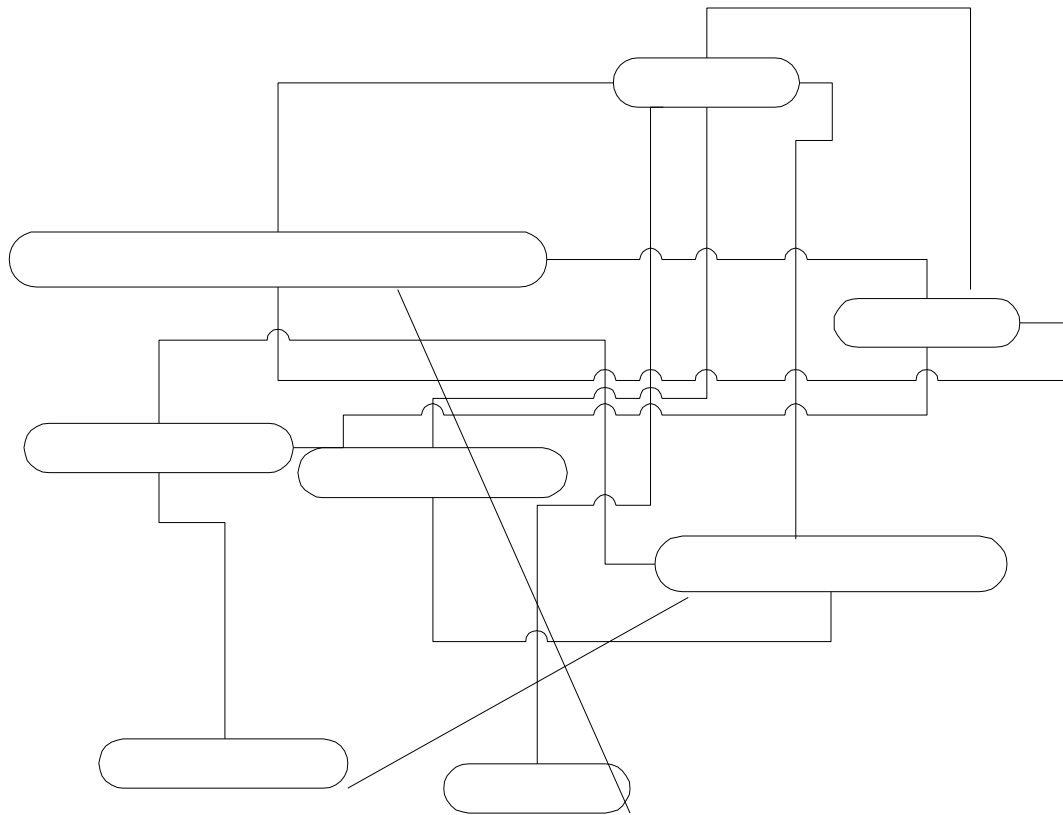
Where's the lifecycle?



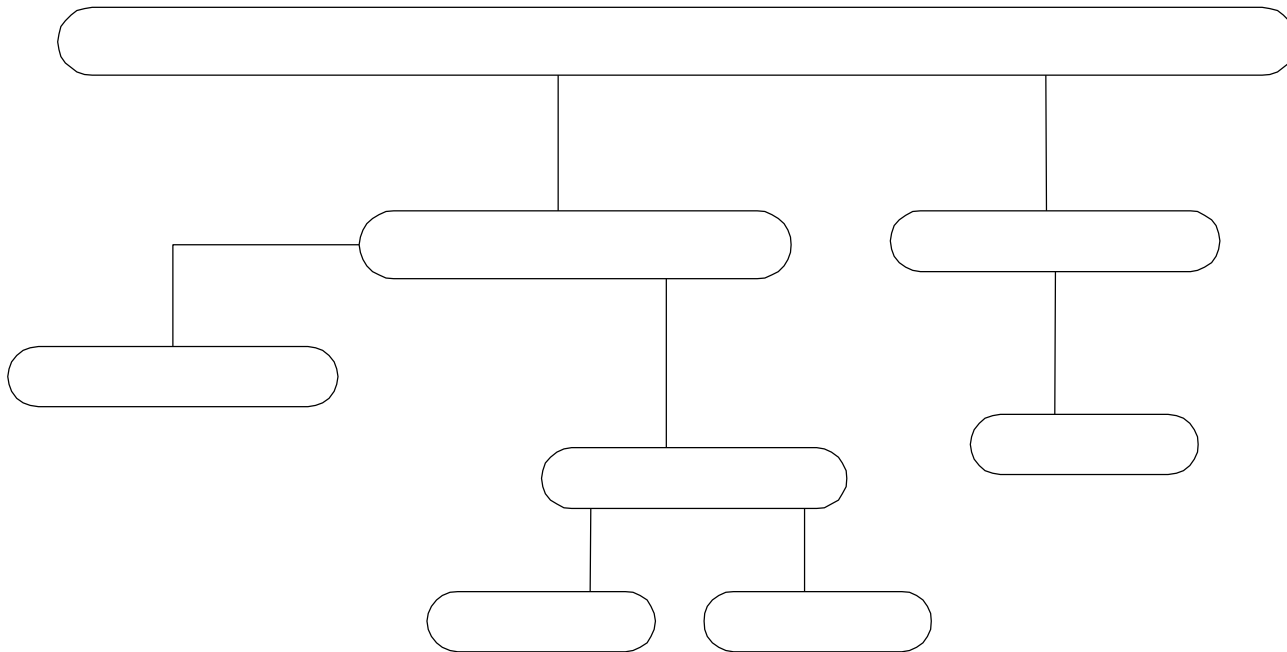
Cyclic or born-and-die are OK



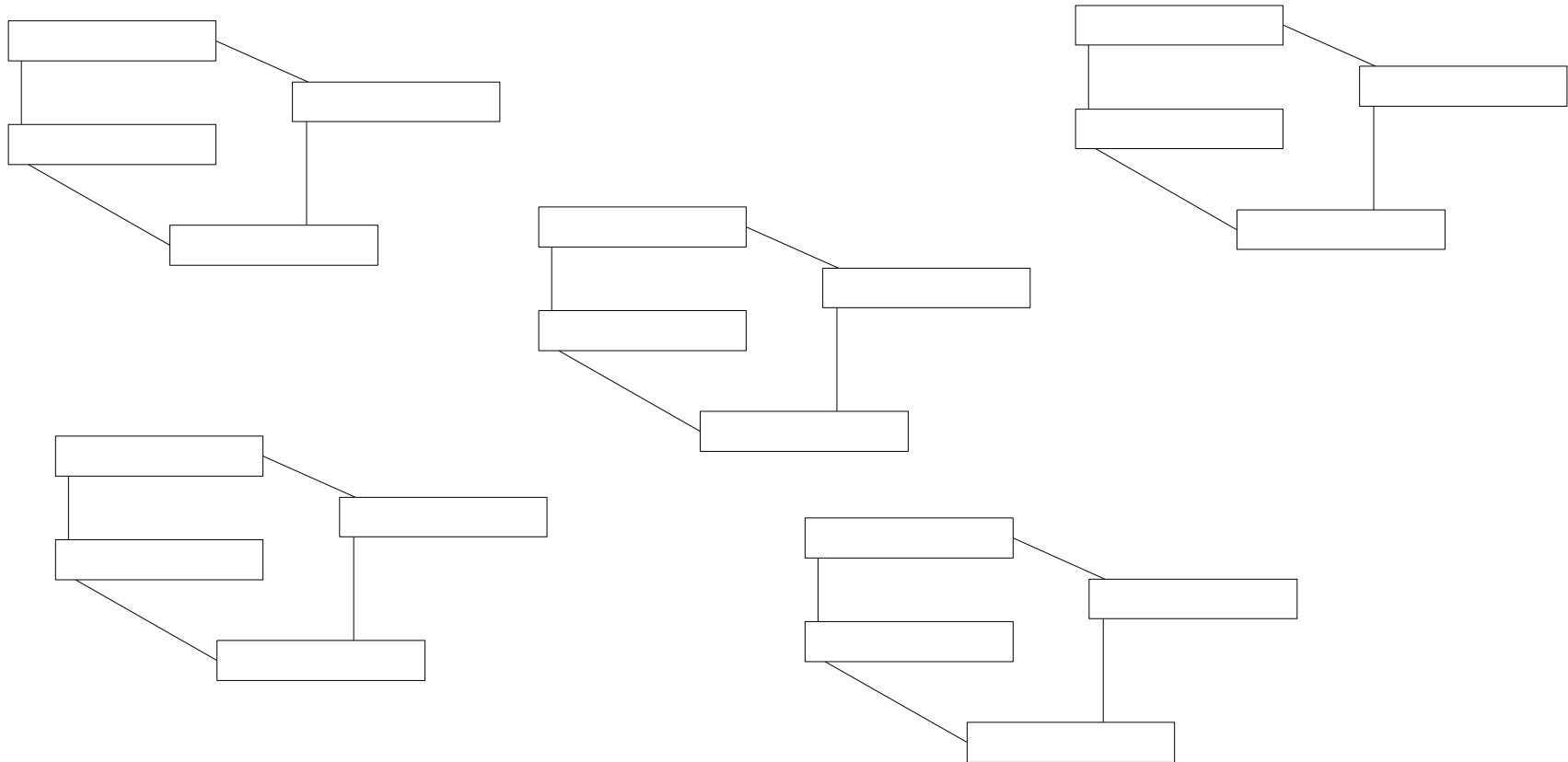
Excessive object communication vs.



Layered control!



Cut and paste states



Trouble in the action language

- ◆ Excessive if-then logic
- ◆ Use of literals

Summary of principles

- ◆ Put the if-then logic in the object models!
- ◆ Model objects - not functions
- ◆ Minimize control threads & object communication
- ◆ Minimize need for complicated action language
- ◆ Model lifecycles - not random functions

Observed results

- ◆ Most projects are able to get something working - that's progress!
- ◆ But on the first projects or two, they usually only scratch the surface of what's possible with OOA
- ◆ A 25-50% reduction in objects and states is usually possible