

Modeling Complex Behavior Simply or How Much is Too Much?

Stephen J. Mellor

Project Technology, Inc.

+1 (520) 544-2881

<http://www.projtech.com>



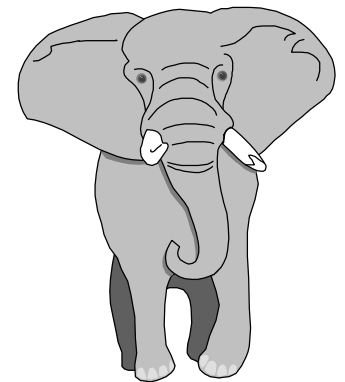
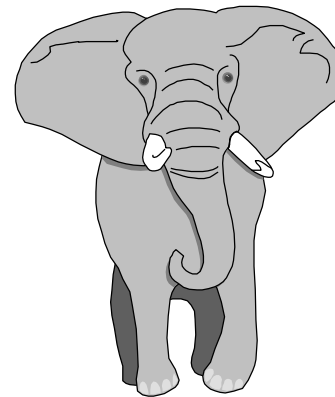
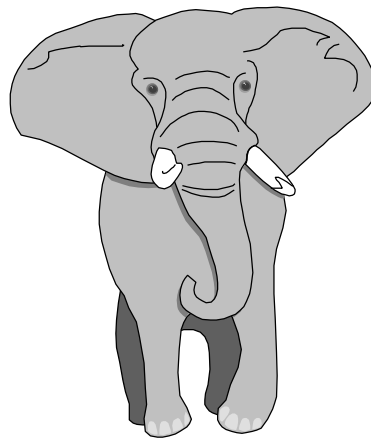
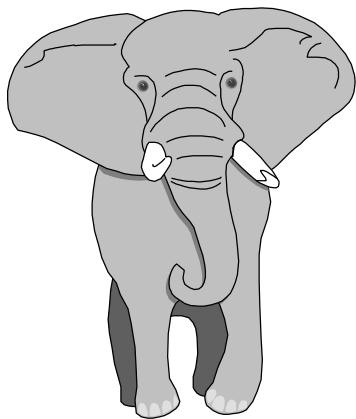
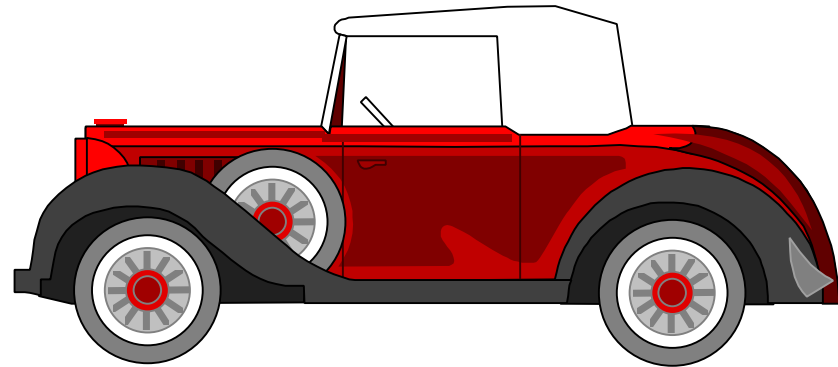
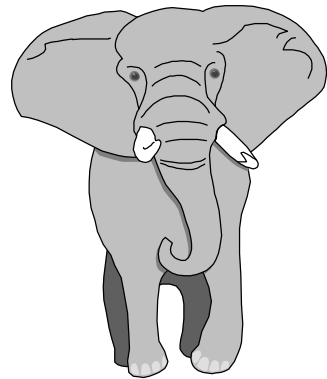
Time-To-Market

To meet time-to-market for the project, we need to:

- know that we're making steady verifiable progress
- avoid duplication and contradiction
- express results with an accessible organization
- use a simple but expressive notation

| Schedule | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | | | | | | |
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | |

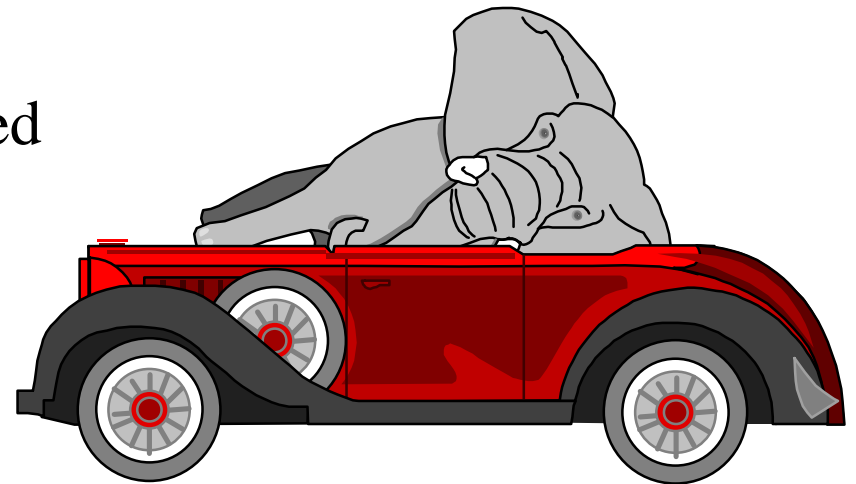
Verifiable Progress



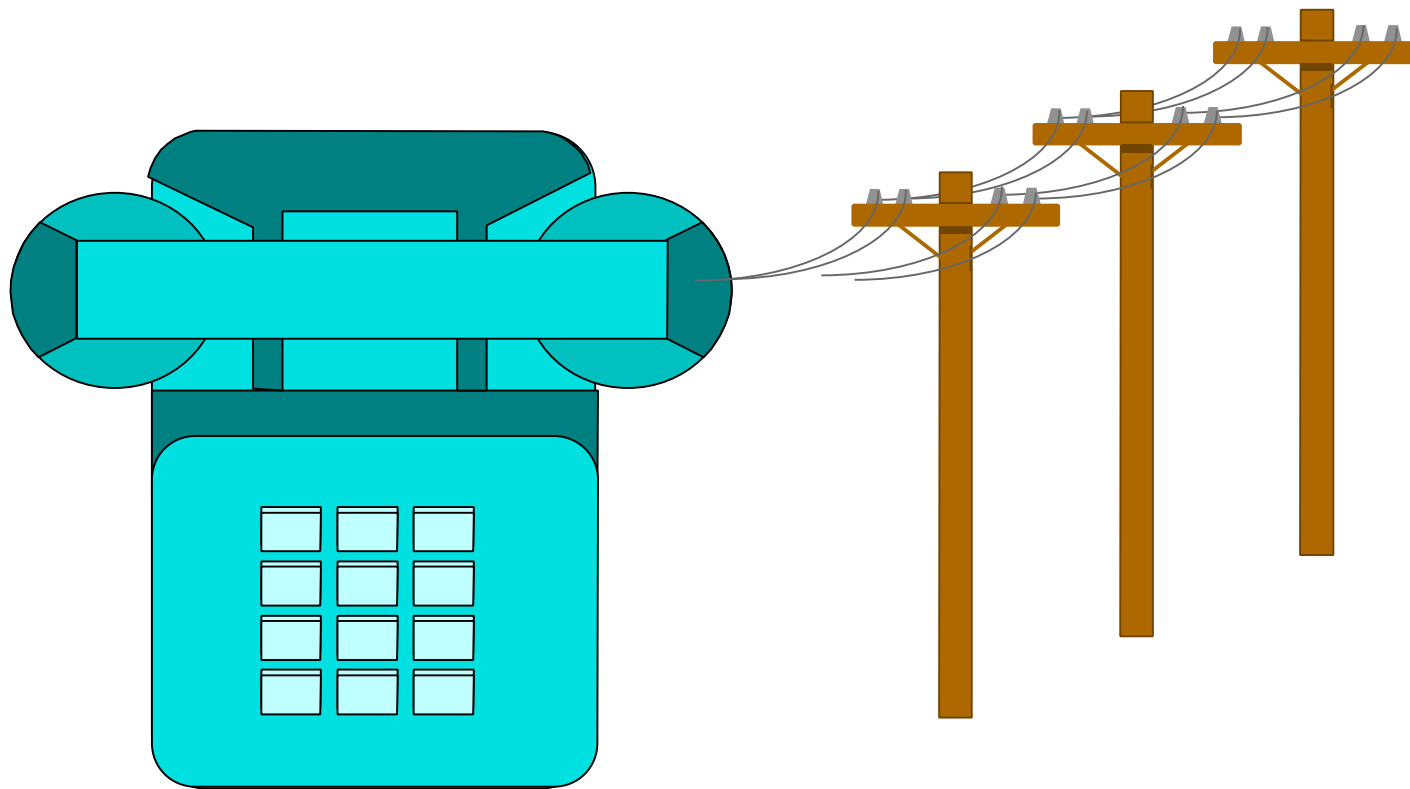
Verifiable Progress

To make verifiable progress:

- proceed bottom up
- accumulate 'facts':
behavioral requirements
on the system
- verify facts as you proceed



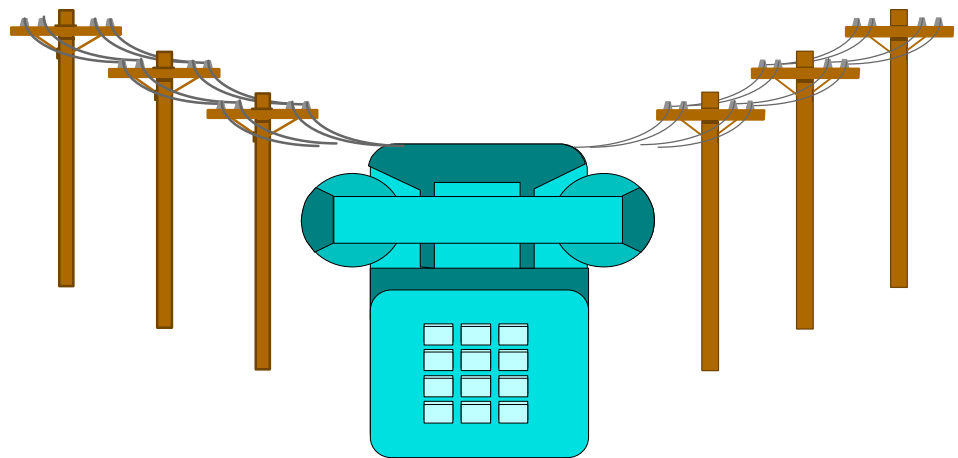
Avoid Duplication



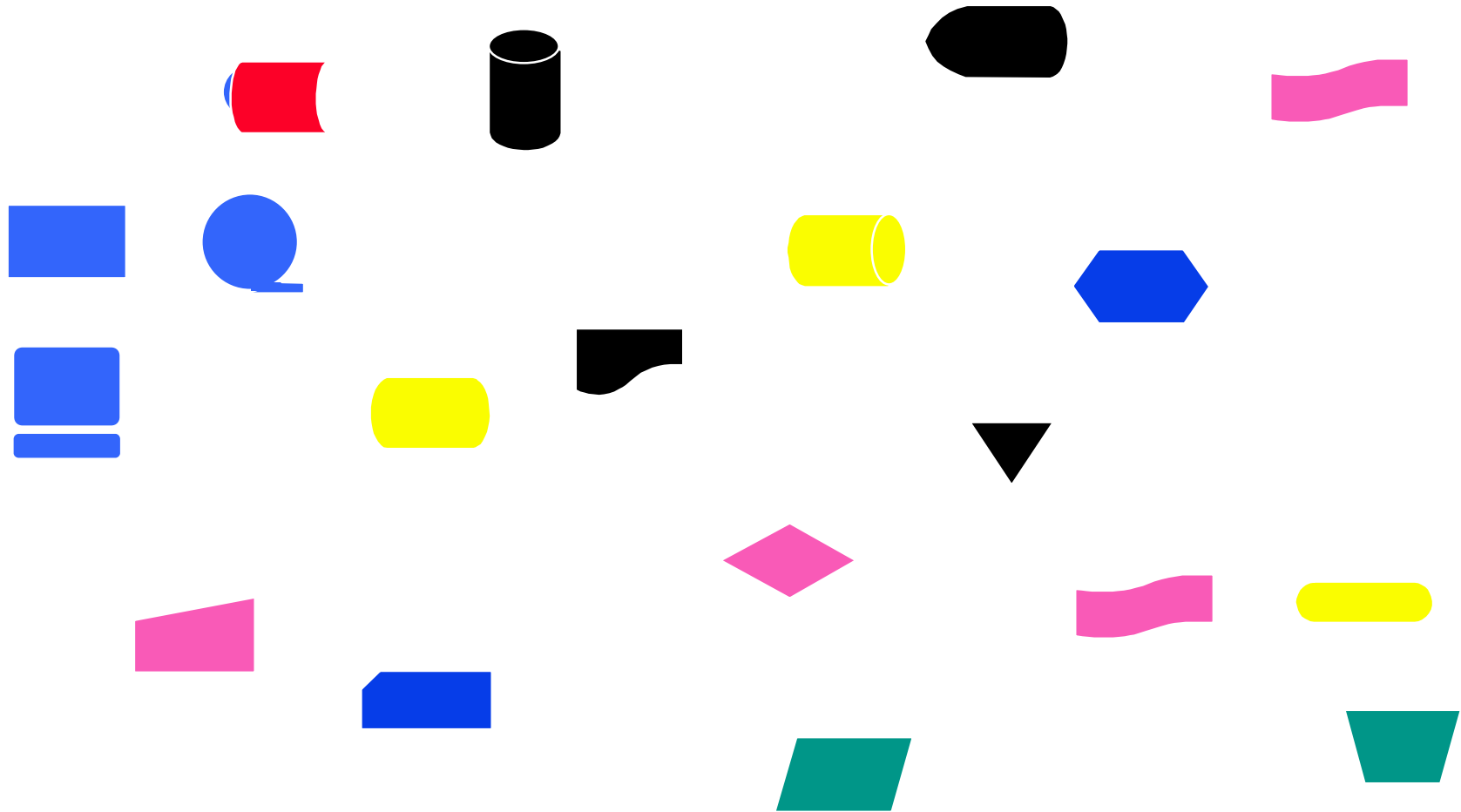
Avoid Duplication

State each fact *once* to obtain a minimal expression of a problem. Because:

- each fact is separately verifiable
- it's faster to write something once
- it avoids contradictions
- maintenance is easier



Readable Organization

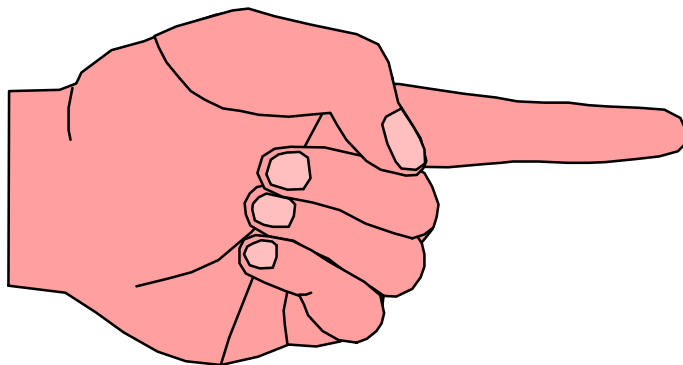


Readable Organization

Organize the models for:

- maximum exposure of information
- minimum organizational overhead

A reorganization of the model should not imply content changes.



The index should have *no* additional semantic content

Use of Notation

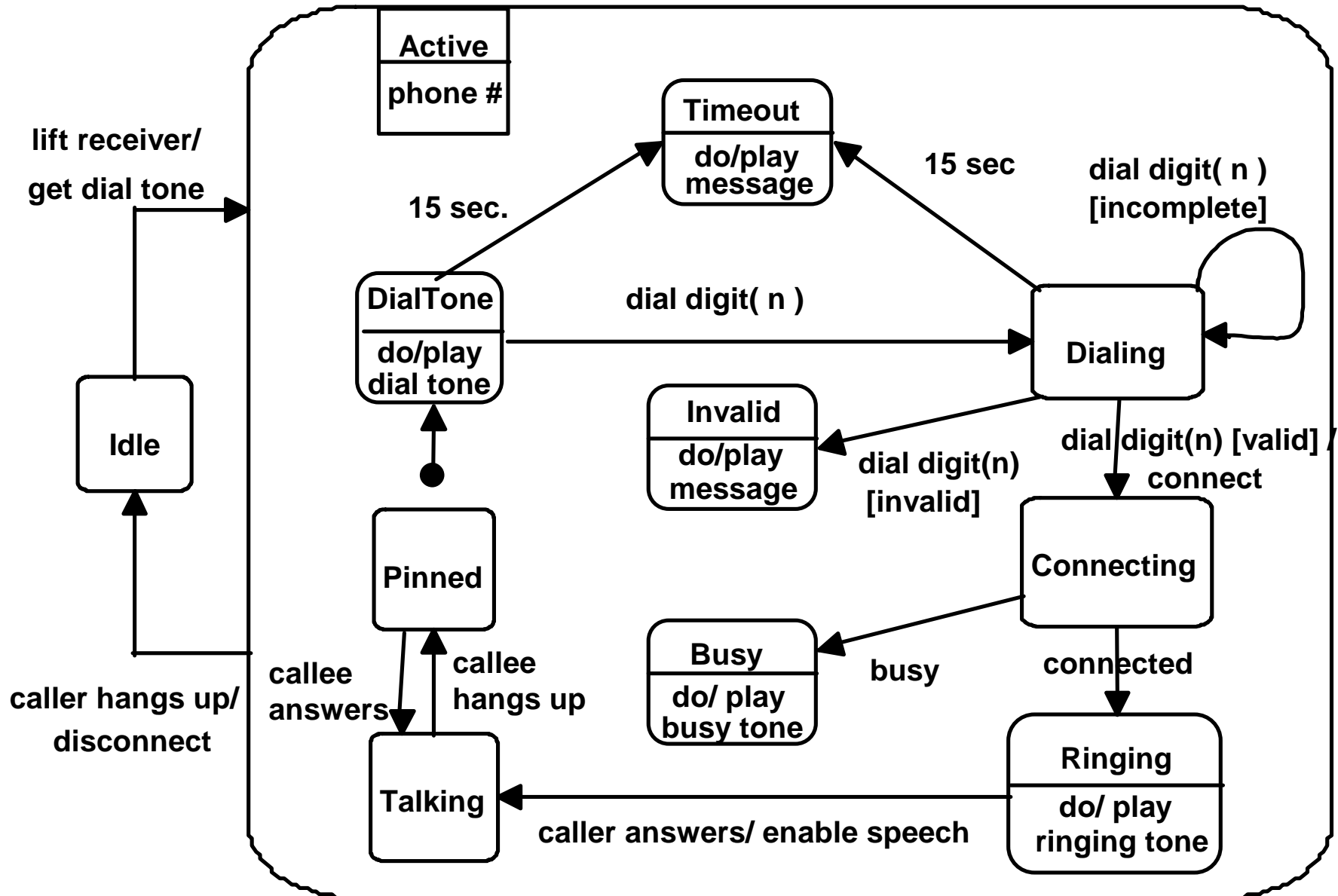
The UML has a rich notation for modeling behavior.

The StateChart may be used to model:

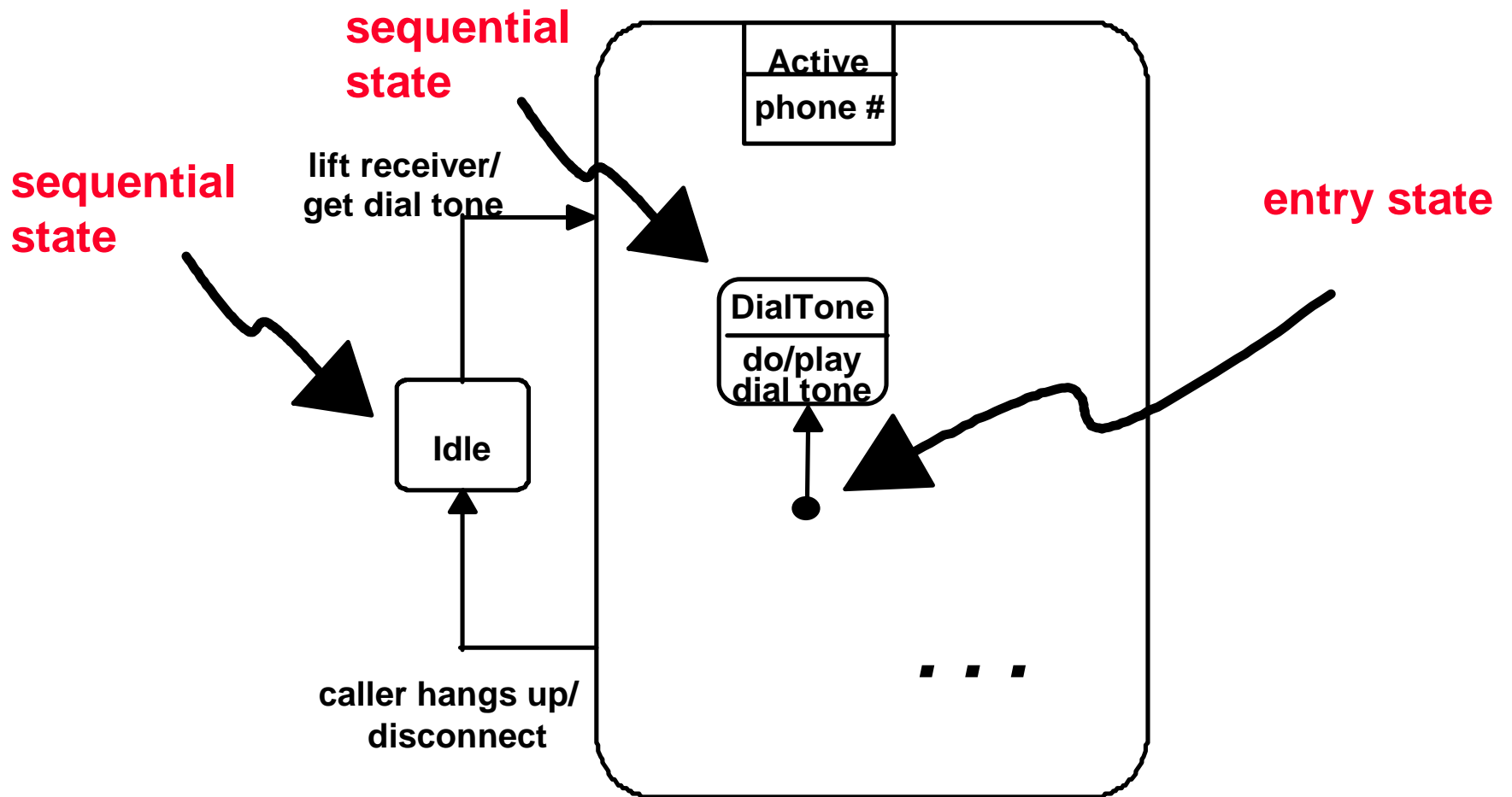
- Use Cases
- Classes
-

We'll use the StateChart to model the behavior of
Classes.

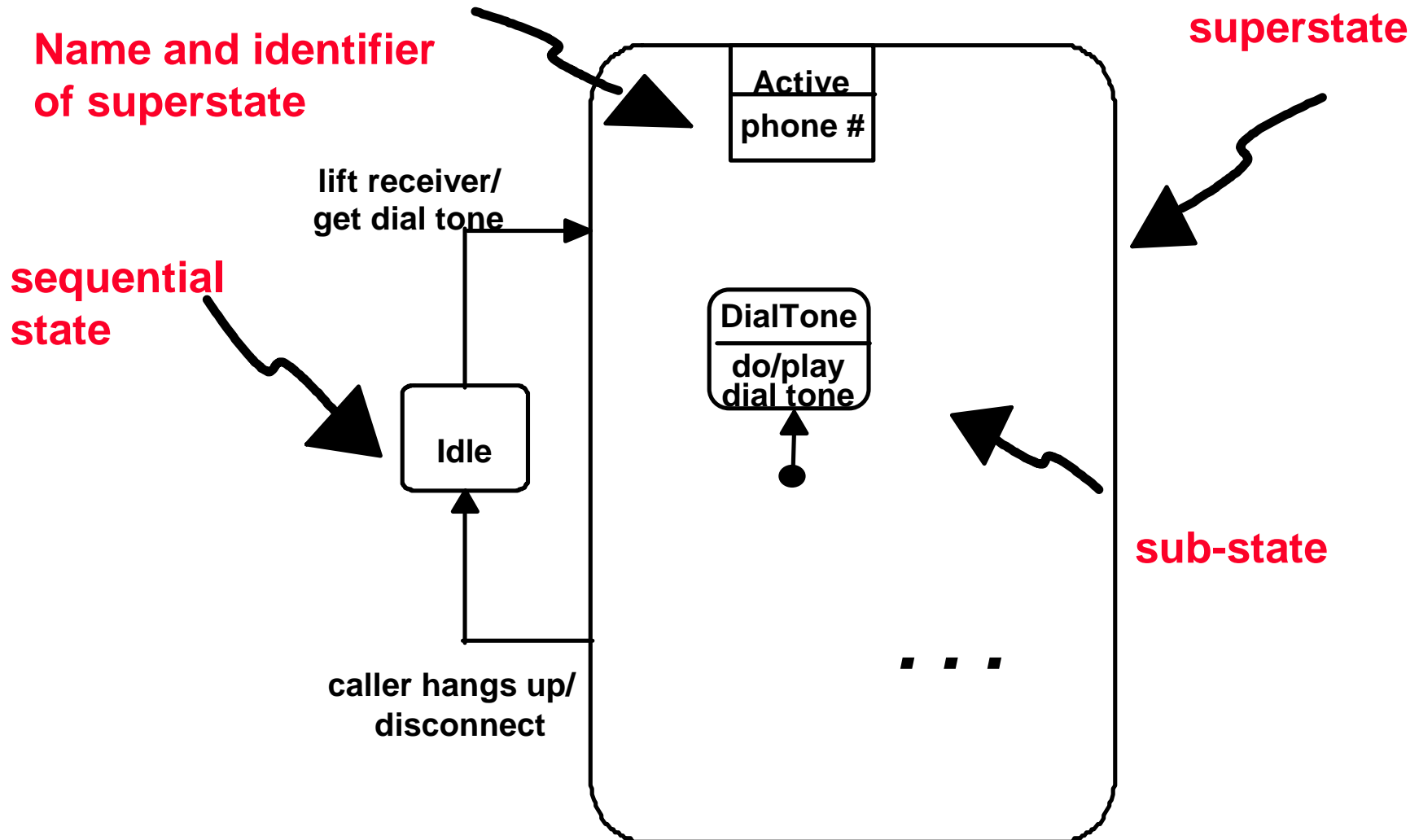
UML Notation



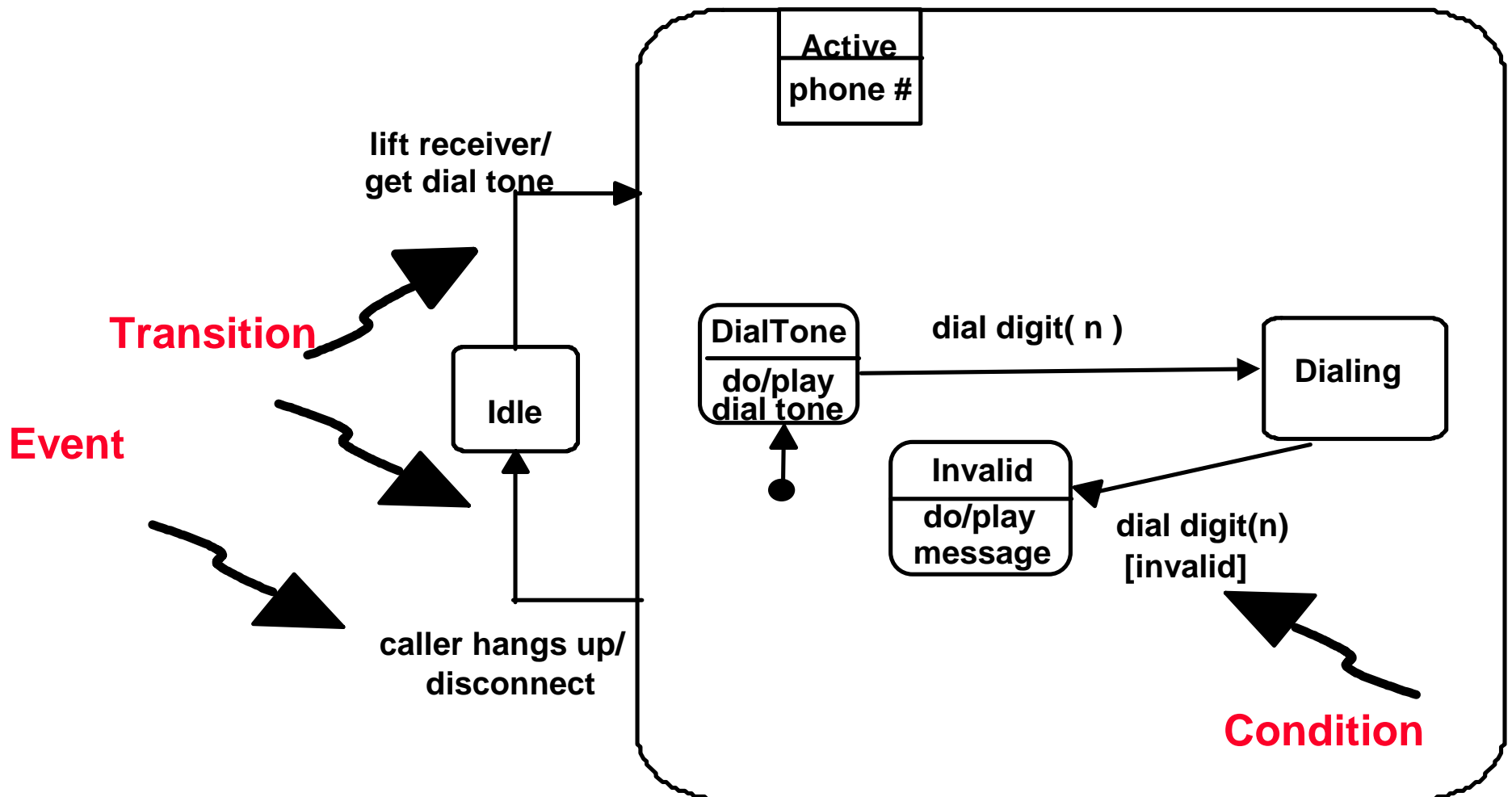
Entry States and Sequential States



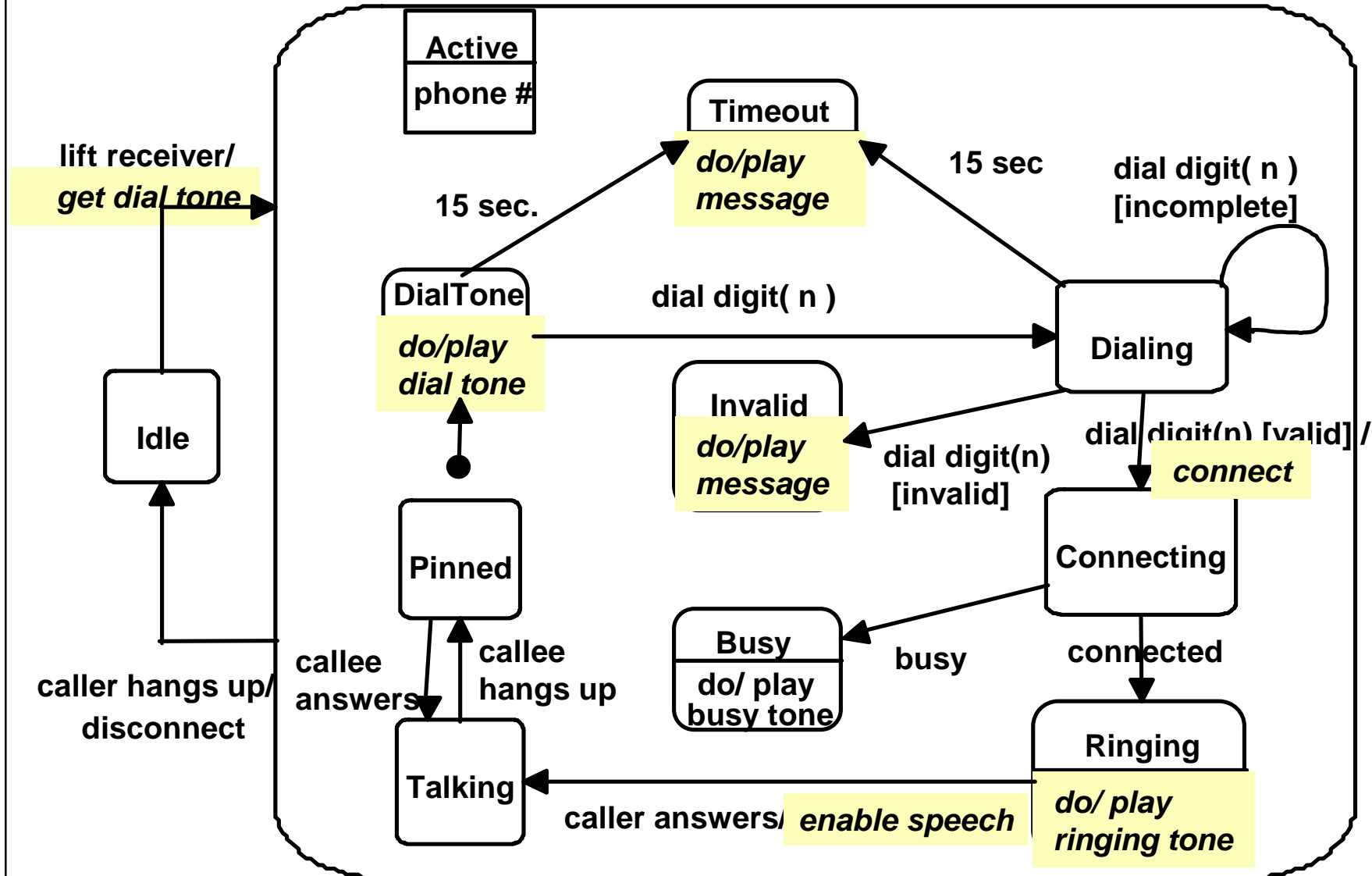
States and Substates



Events, Conditions, and Transitions



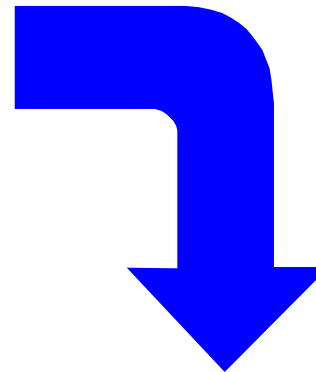
Actions



Work Bottom-Up

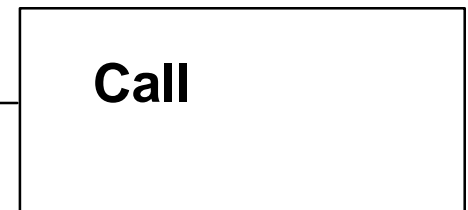
contains all the information about a telephone and how to make calls

(abbreviated class diagrams)

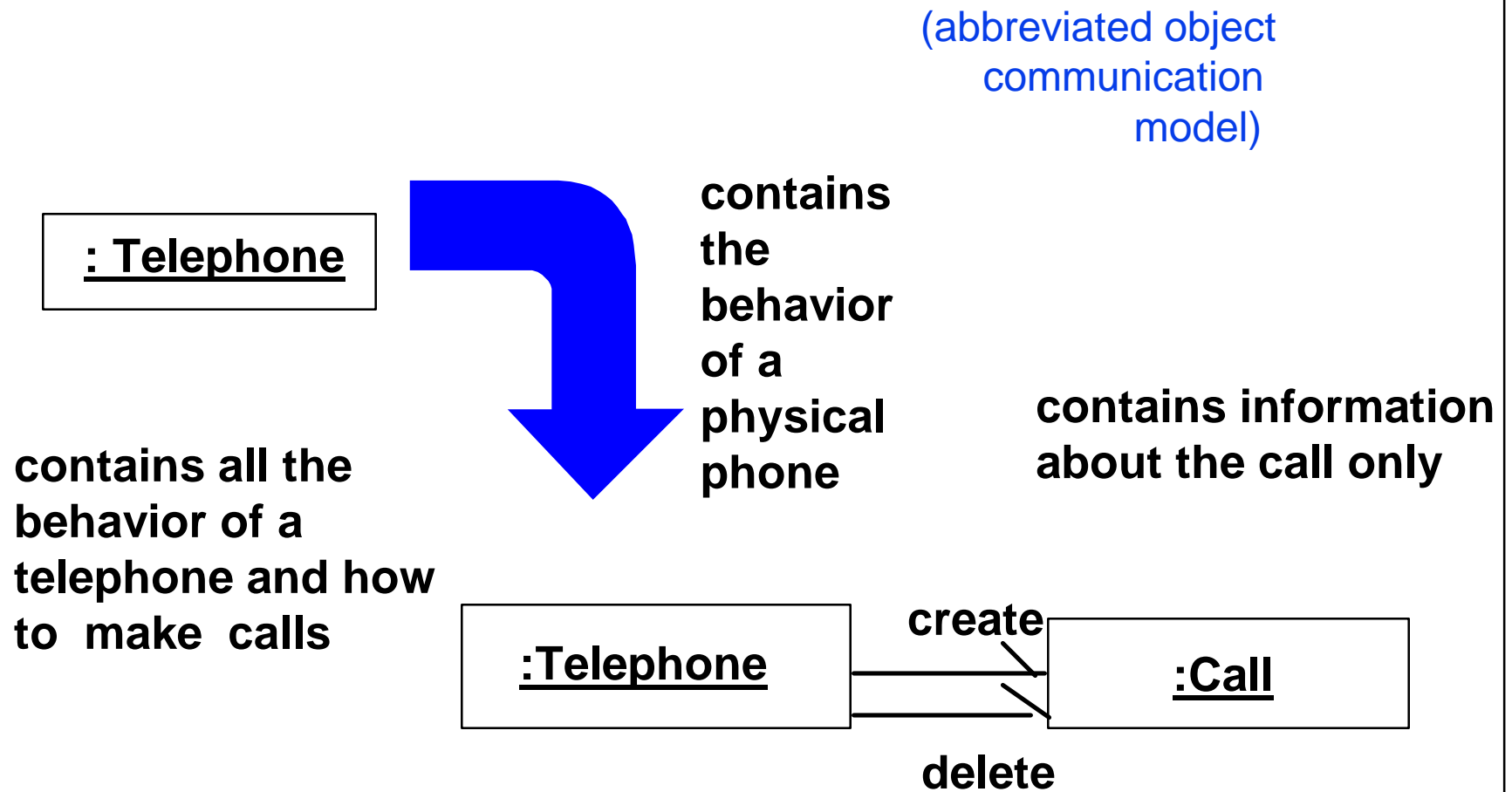


contains information about the call only

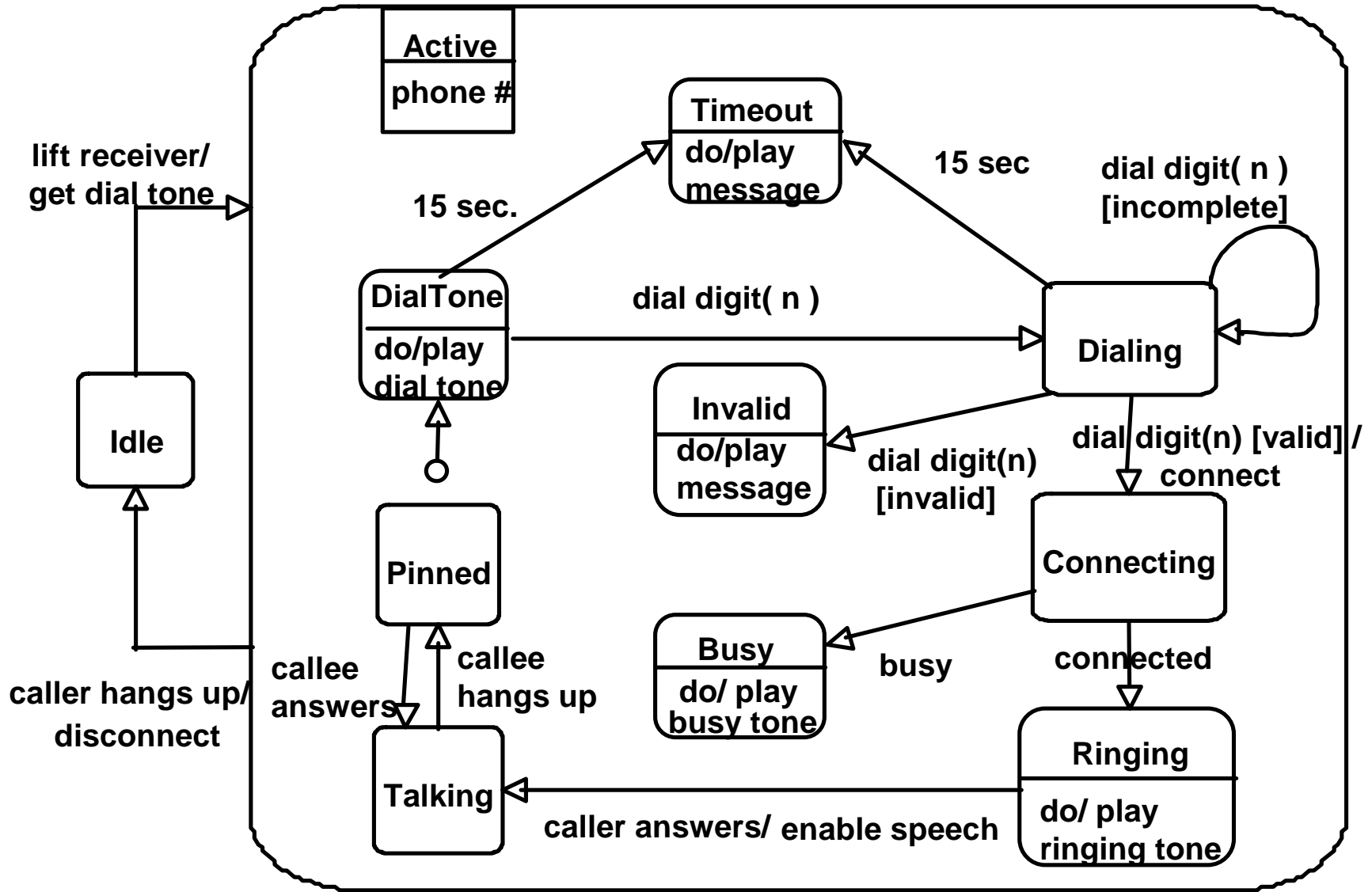
contains information about a physical phone

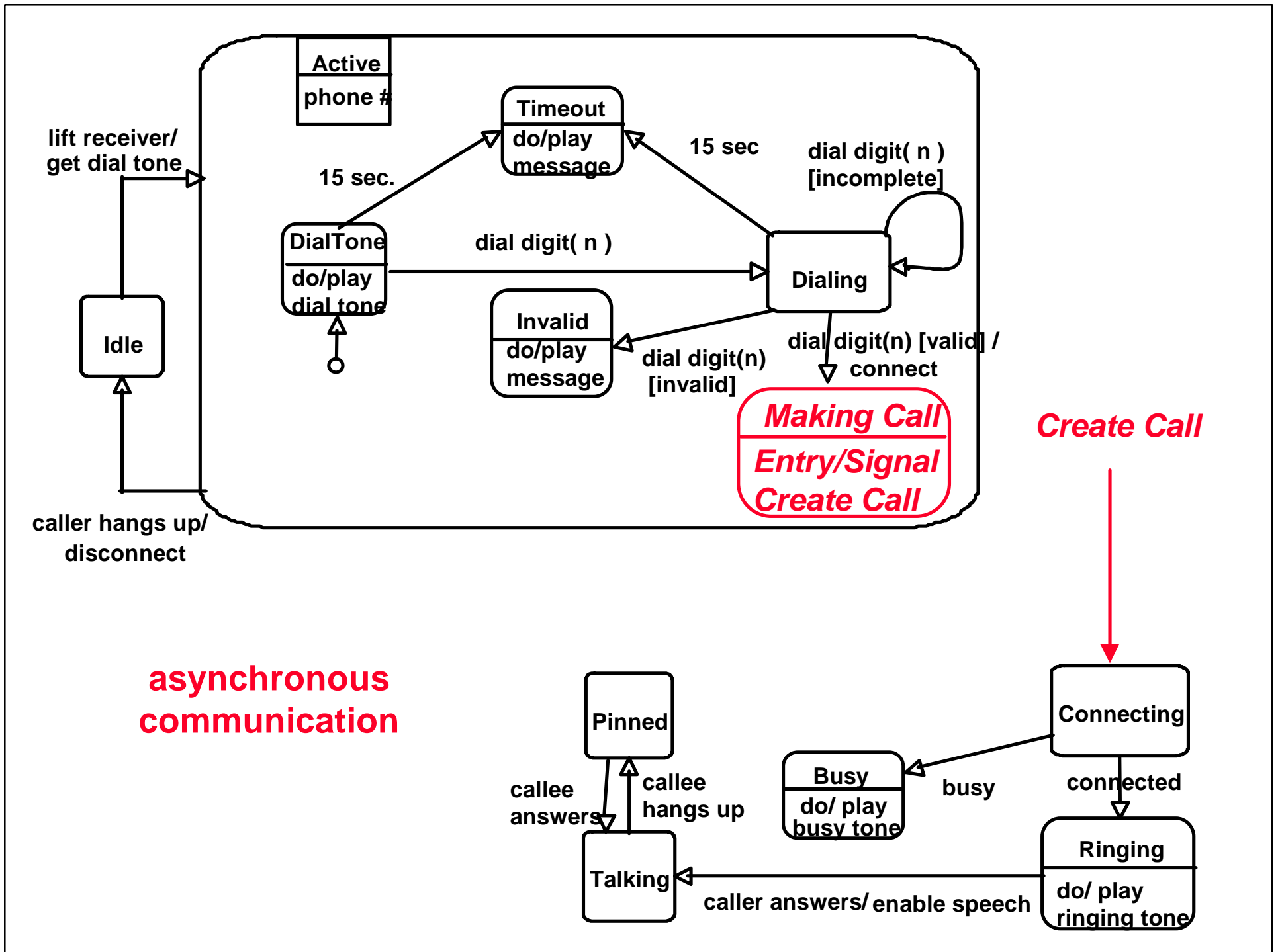


Object Oriented Partitioning

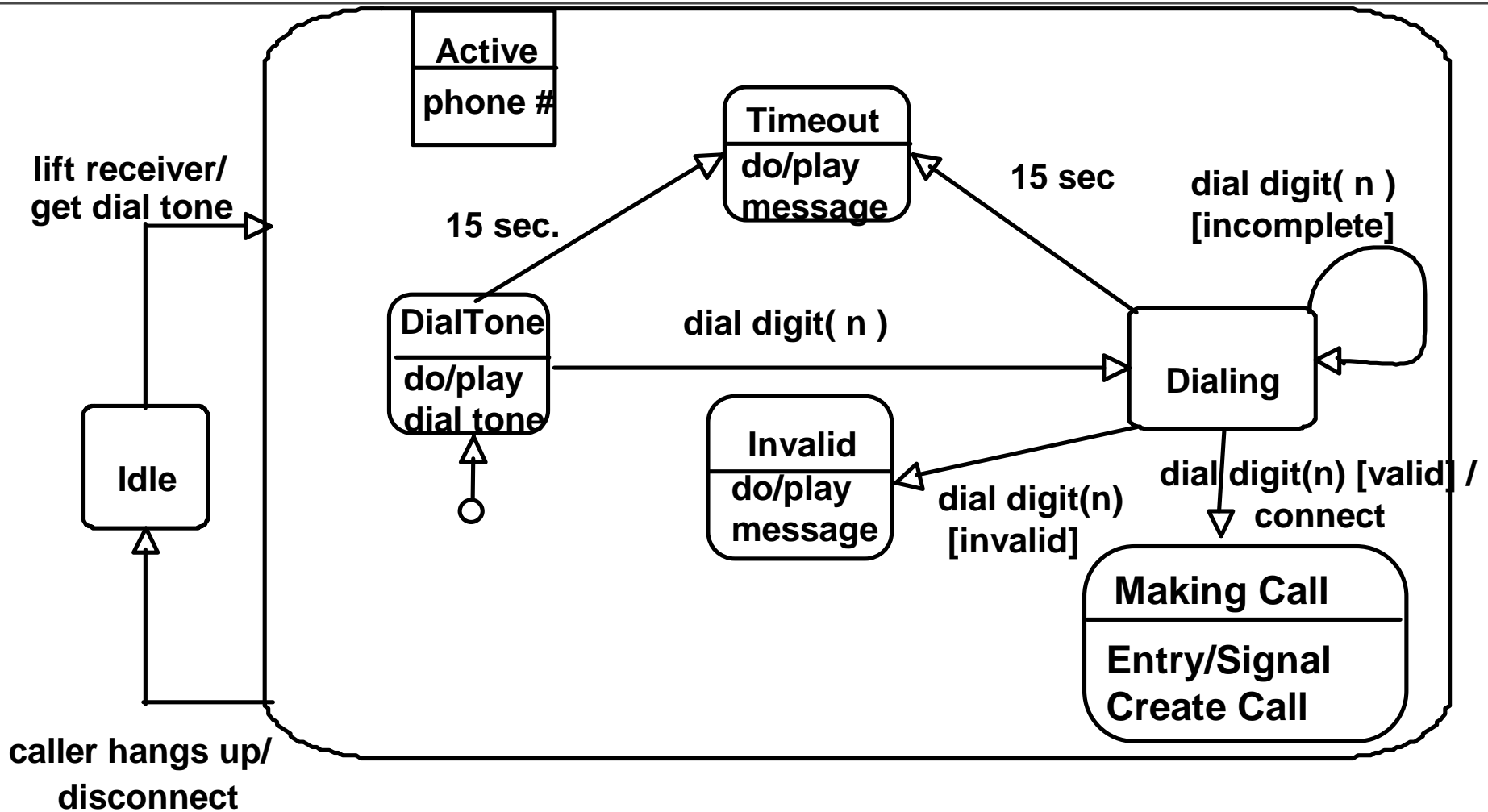


Original Model

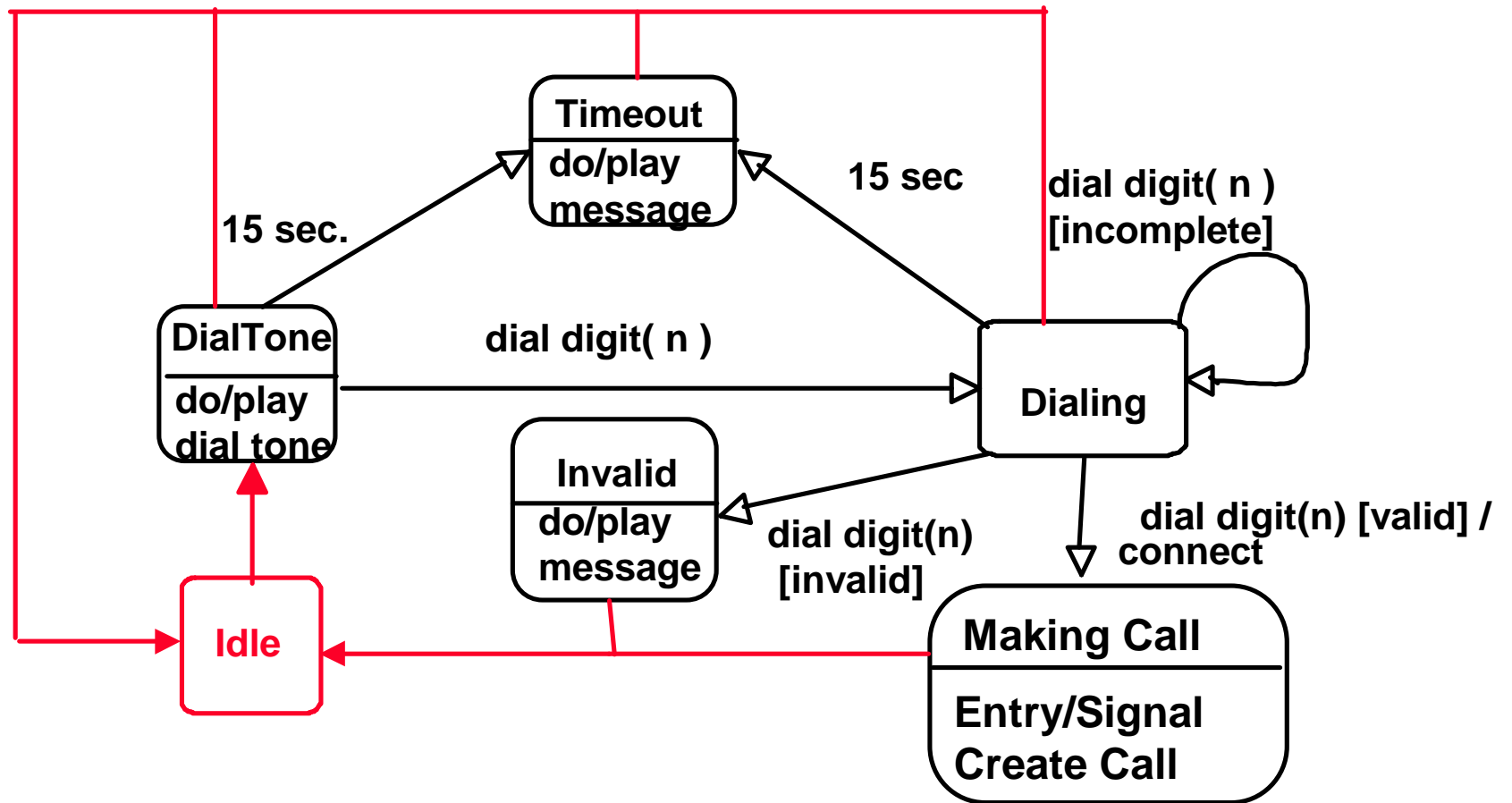




No Semantic Content in Index

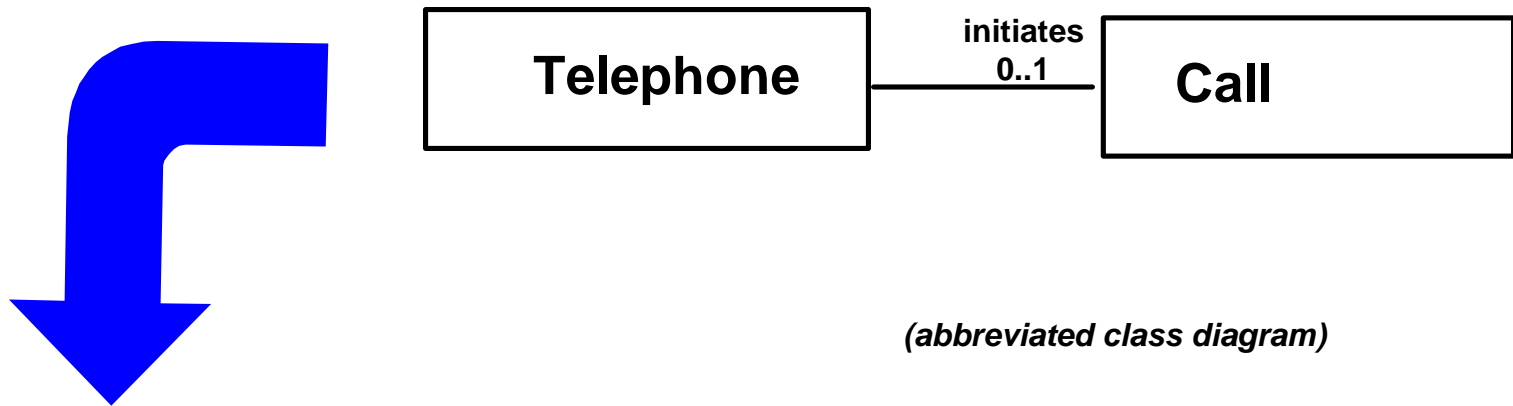


Avoid Hierarchical States

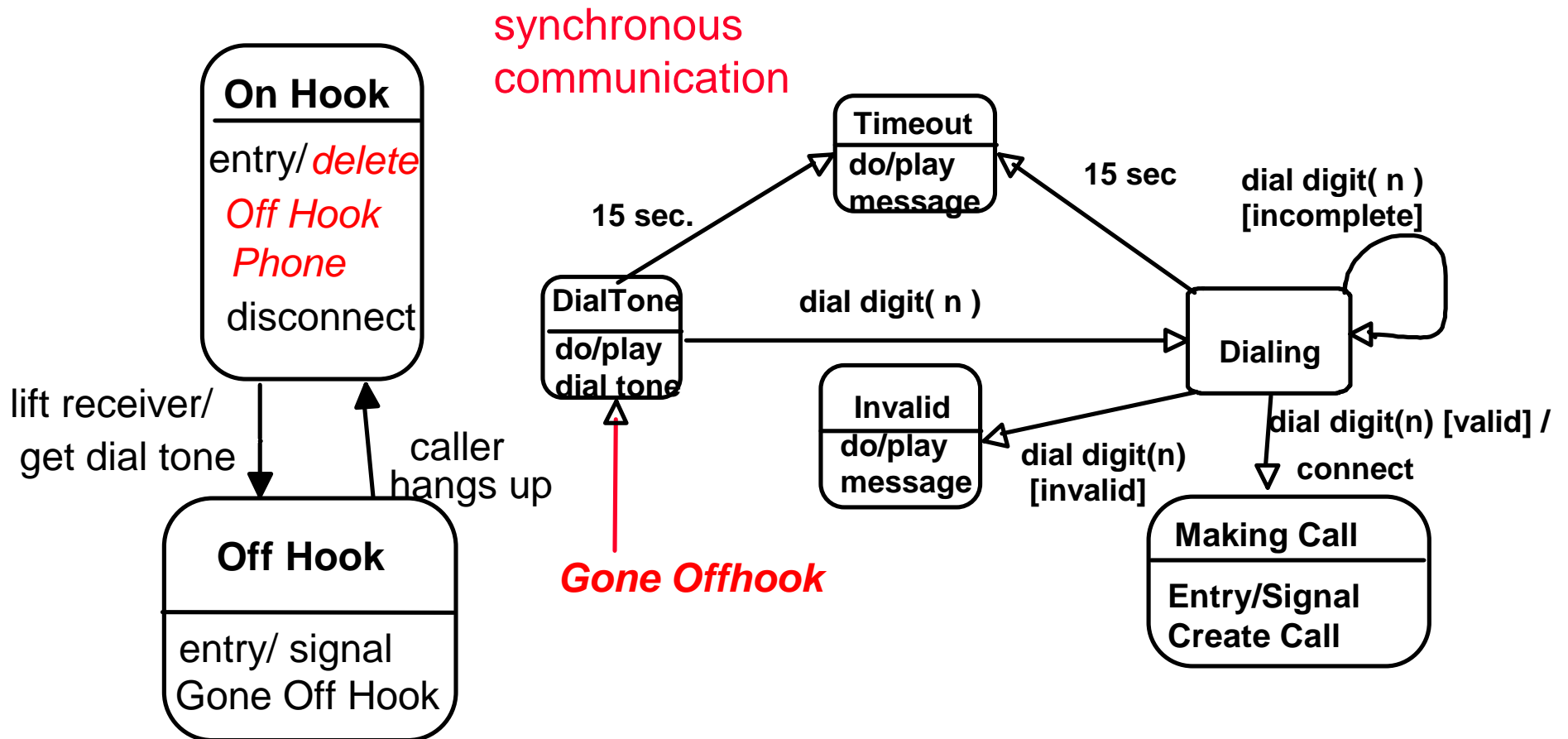


But WAIT! This is WORSE!

So.....build still *more* objects



Communicate Synchronously



Behavior Normalization

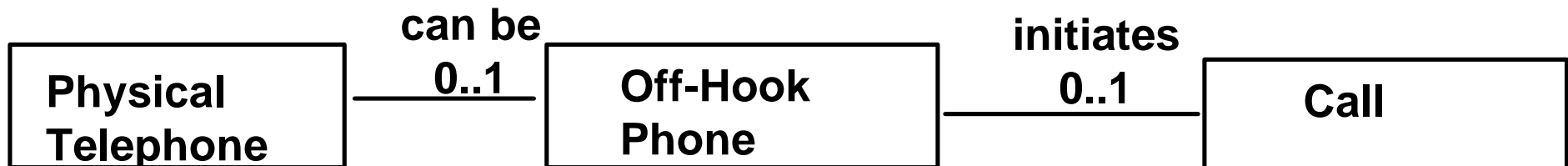
Partition so that each instance has the:

- same characteristics each instance has the same data

and

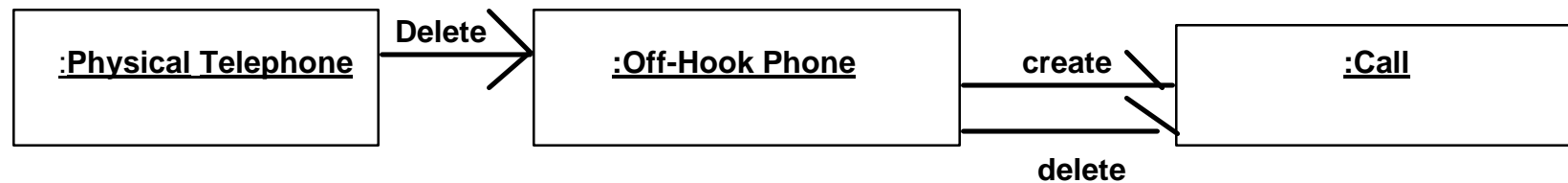
- conforms to the set of rules and policies

each instance has the same behavior



Partition Further When...

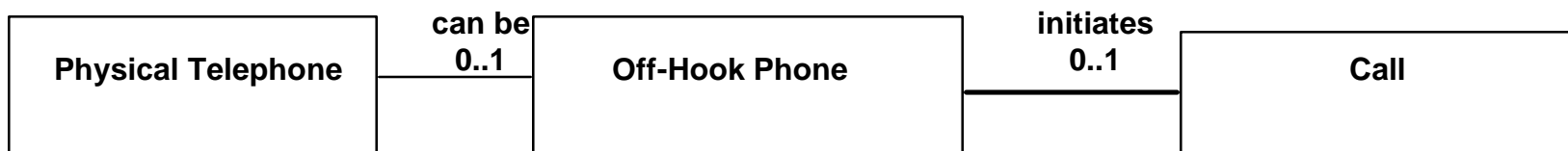
.....a set of related states all make the same transition.



Whatever is happening with an Off-Hook Phone, when you hang up, it goes away.

Partition Further When...

...a relationship exists only in certain states.

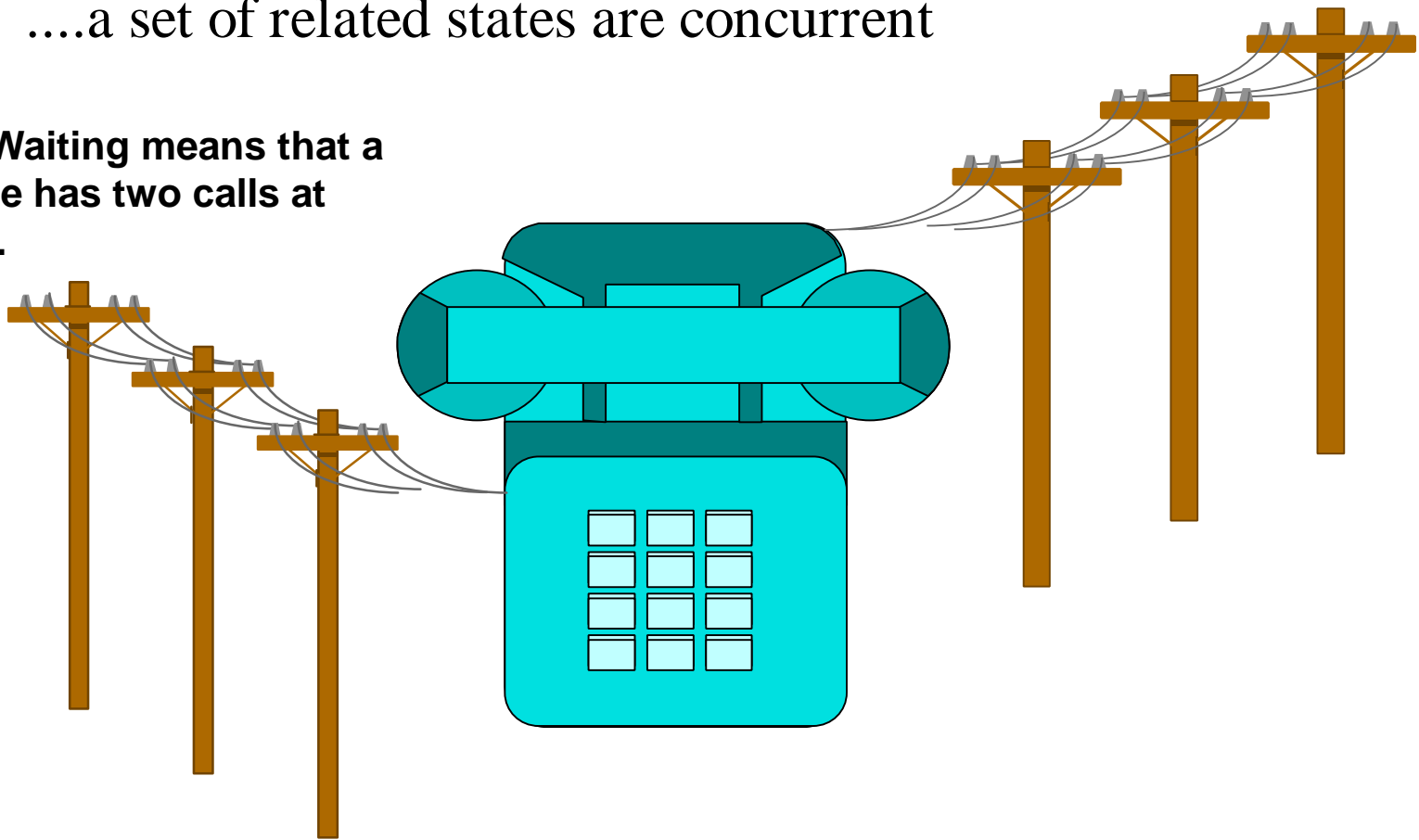


The call can only exist when the telephone is off-hook.

Partition Further When...

....a set of related states are concurrent

Call Waiting means that a phone has two calls at once.



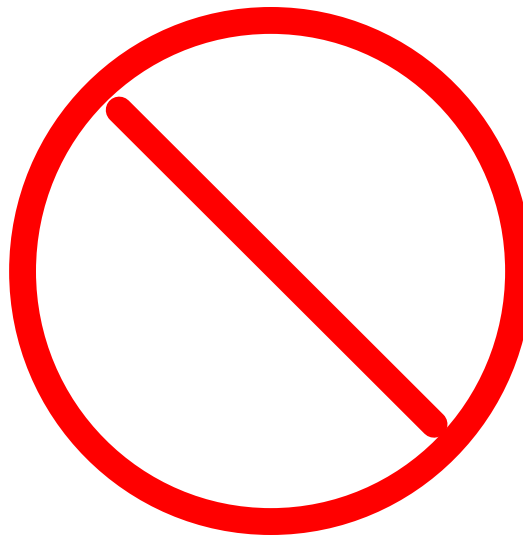
To Simplify Complex Behavior....

...use only a subset of the notation.

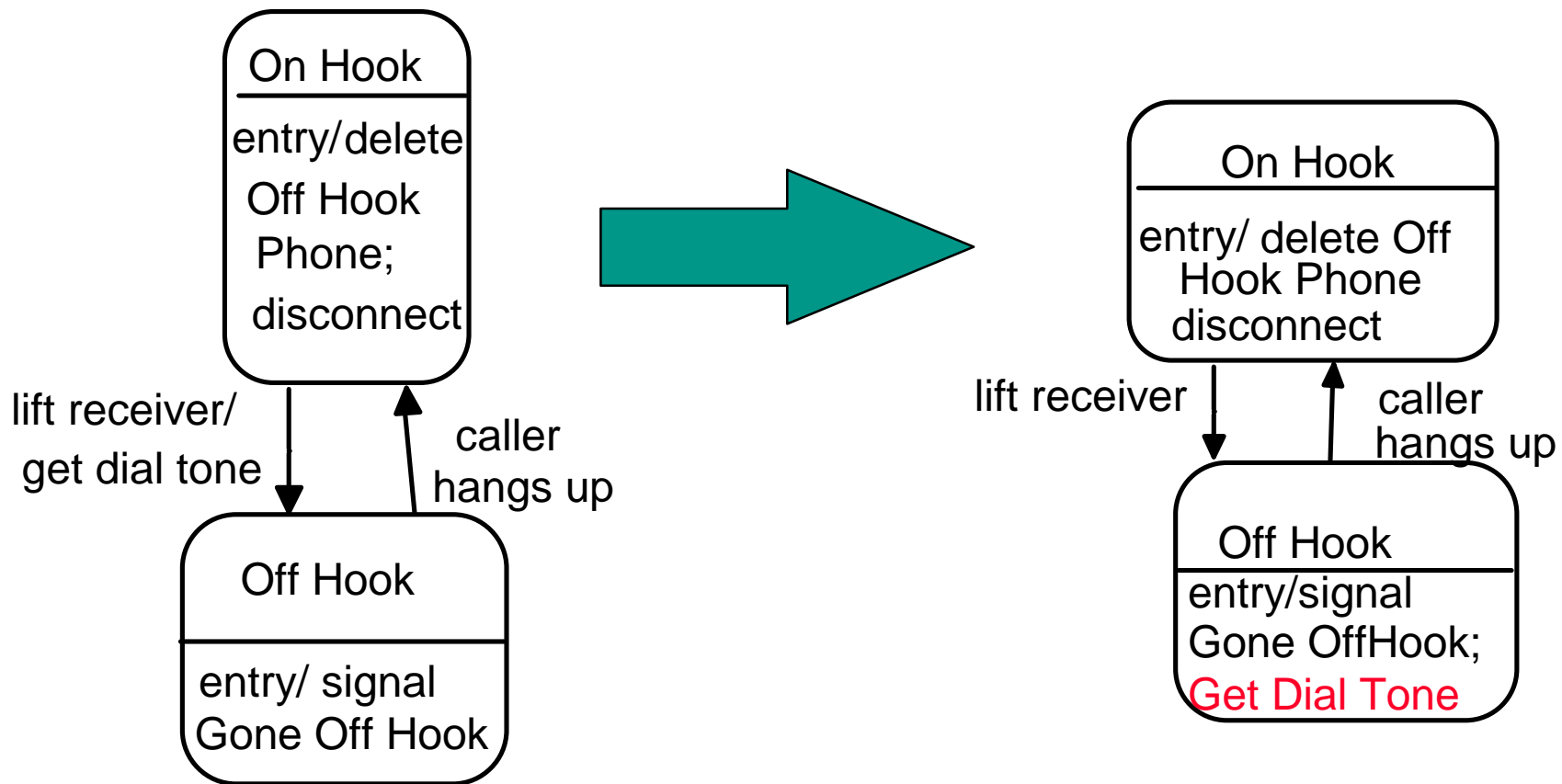


- entry actions

- transition actions
- exit actions



Entry Actions Only



Summary

To model complex behavior simply:

- work bottom-up from known facts
- express behavioral requirements once
- avoid semantic content in the index
- simplify the notation

This will speed time-to-market for your project.

