NO MORE GARBAGE IN:

VALIDATING FORMAL MODELS

25 May 2023

Pamela Zave

Princeton University

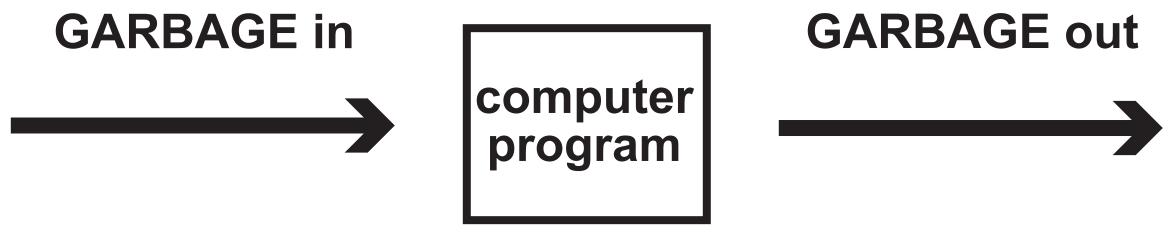
Princeton, New Jersey

Tim Nelson

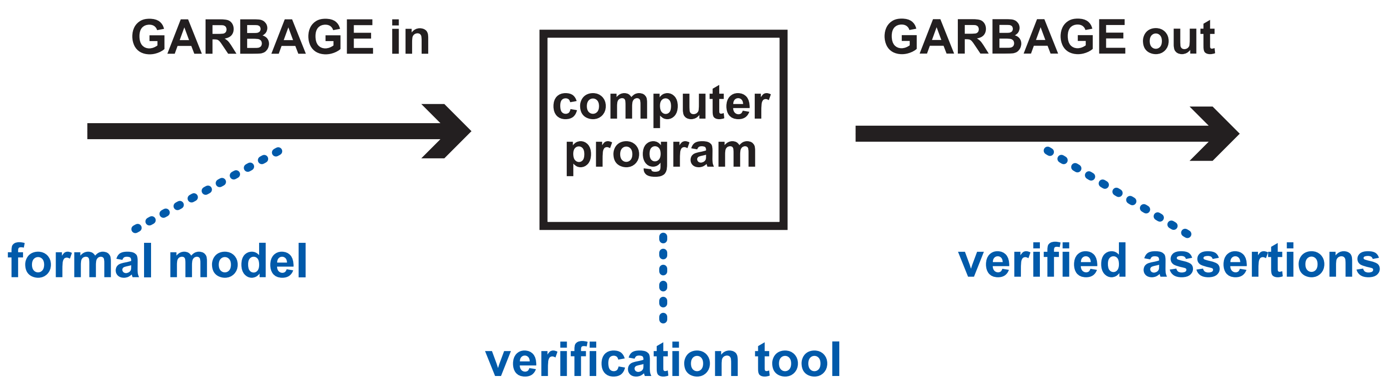
Brown University

Providence, Rhode Island

WISDOM THAT NEVER GROWS OLD . . .



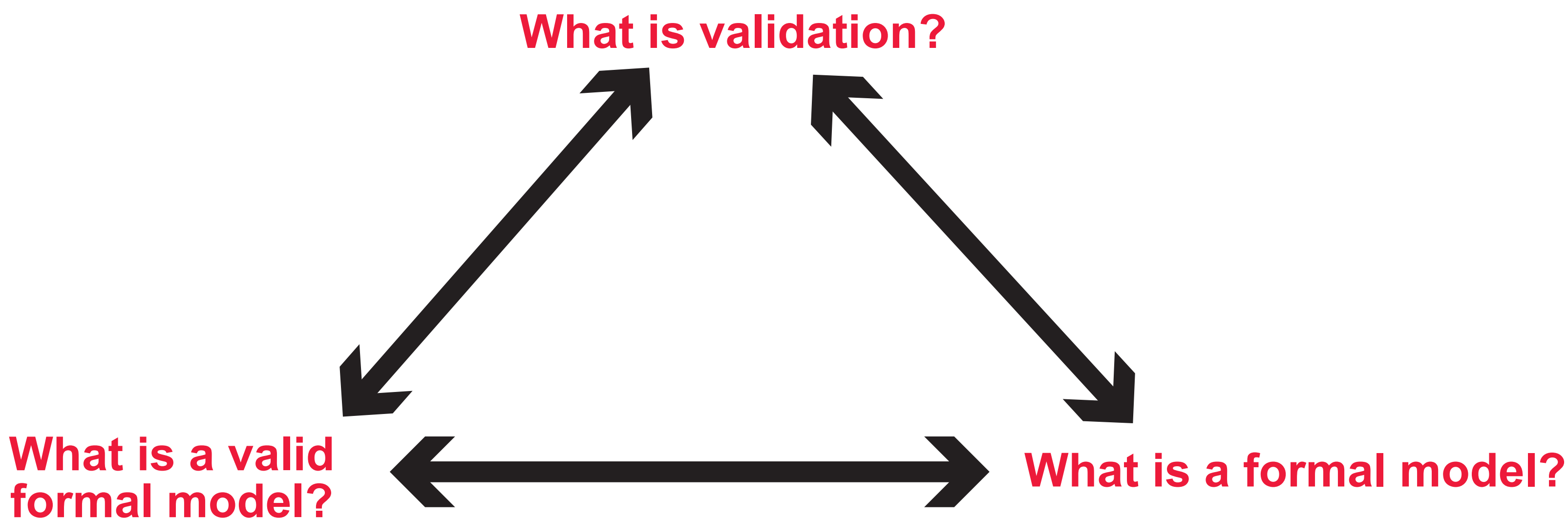
THE VERSION WE ARE INTERESTED IN:



VALIDATION OF A FORMAL MODEL . . .

IS THE ONLY WAY TO ENSURE THAT . . .

THE INPUT TO A VERIFICATION TOOL IS NOT GARBAGE



WHAT IS A FORMAL MODEL? TECHNOLOGY VERSION

DESCRIBES A STATE-TRANSITION SYSTEM ("operational" style)

- there is a state space, state transitions
- semantics is a set of possible traces, i.e., sequences of states

GIVES PROPERTIES OF THE TRACE SET ("declarative" style)

- in predicate logic and temporal logic
- facts and assertions (true of all traces), predicates (true of some traces)

MODELING LANGUAGES

- support both operational and declarative styles
- e.g., Alloy, TLA+, Event-B, Dafny, Promela plus temporal logic (for Spin model checker)

WHAT IS A FORMAL MODEL?

DESCRIBES A STATE-TRANSITION SYSTEM ("operational" style)

- there is a state space, state transitions
- semantics is a set of possible traces, i.e., sequences of states

GIVES PROPERTIES OF THE TRACE SET ("declarative" style)

- in predicate logic and temporal logic
- facts and assertions (true of all traces), predicates (true of some traces)

MODELING LANGUAGES

- support both operational and declarative styles
- e.g., Alloy, TLA+, Event-B, Dafny, Promela plus temporal logic (for Spin model checker)

EXAMPLE

inList: list *elem*

outList: list *elem*

order: elem -> elem

OPERATIONAL STYLE:

a sorting algorithm

DECLARATIVE STYLE:

- *outList* has the same length as *inList*
- *outList* has the same elements as *inList*
- *outList* is ordered according to *order*

WHAT IS A FORMAL MODEL?

DESCRIBES A STATE-TRANSITION SYSTEM (“operational” style)

- there is a state space, state transitions
- semantics is a set of possible traces, i.e., sequences of states

GIVES PROPERTIES OF THE TRACE SET (“declarative” style)

- in predicate logic and temporal logic
- facts and assertions (true of all traces), predicates (true of some traces)

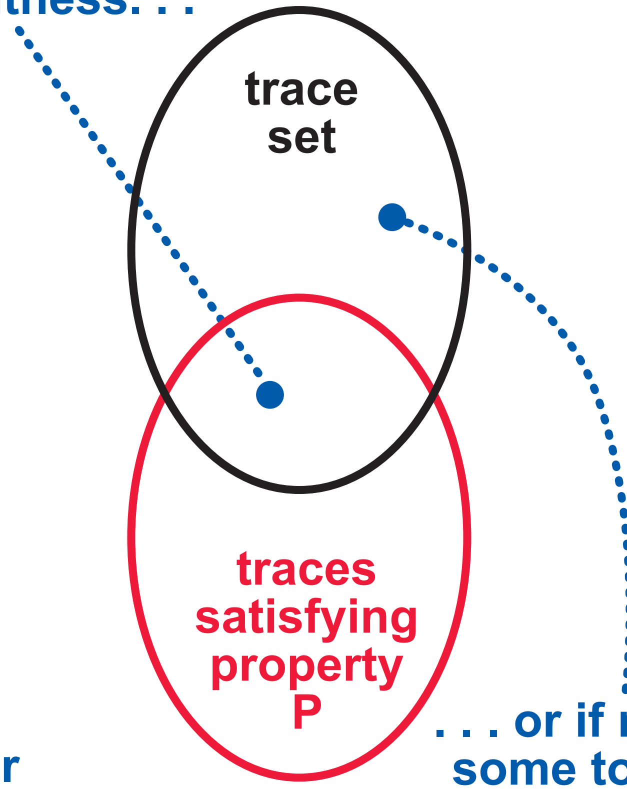
MODELING LANGUAGES

- support both operational and declarative styles
- e.g., Alloy, TLA+, Event-B, Dafny, Promela plus temporal logic (for Spin model checker)

HAS A VERIFICATION TOOL

if P is a predicate, the tool can give you an instance or witness. . .

if P is an assertion, the tool can confirm that P is verified . . .



. . . or tell you that the two are inconsistent

. . . or if not, some tools can give you a counter-example

VALIDATION OF A FORMAL MODEL . . .
IS THE ONLY WAY TO ENSURE THAT . . .
THE INPUT TO A VERIFICATION TOOL IS NOT GARBAGE

INITIAL DEFINITIONS

A *valid* formal model is an accurate, precise, and comprehensible formal description of real or hypothetically real phenomena.

***Validation* is the process of checking that a formal model is valid.**

Because the phenomena being modeled are informal, validation is *inherently informal,* although it can be assisted by formal analysis and verification.

VALIDATION OF A FORMAL MODEL . . .
IS THE ONLY WAY TO ENSURE THAT . . .
THE INPUT TO A VERIFICATION TOOL IS NOT GARBAGE

INITIAL DEFINITIONS

A *valid* formal model is an accurate, precise, and comprehensible formal description of real or hypothetically real phenomena.

Validation is the process of checking that a formal model is valid.

Because the phenomena being modeled are informal, validation is *inherently informal,* although it can be assisted by formal analysis and verification.

ACCURACY VS. PRECISION

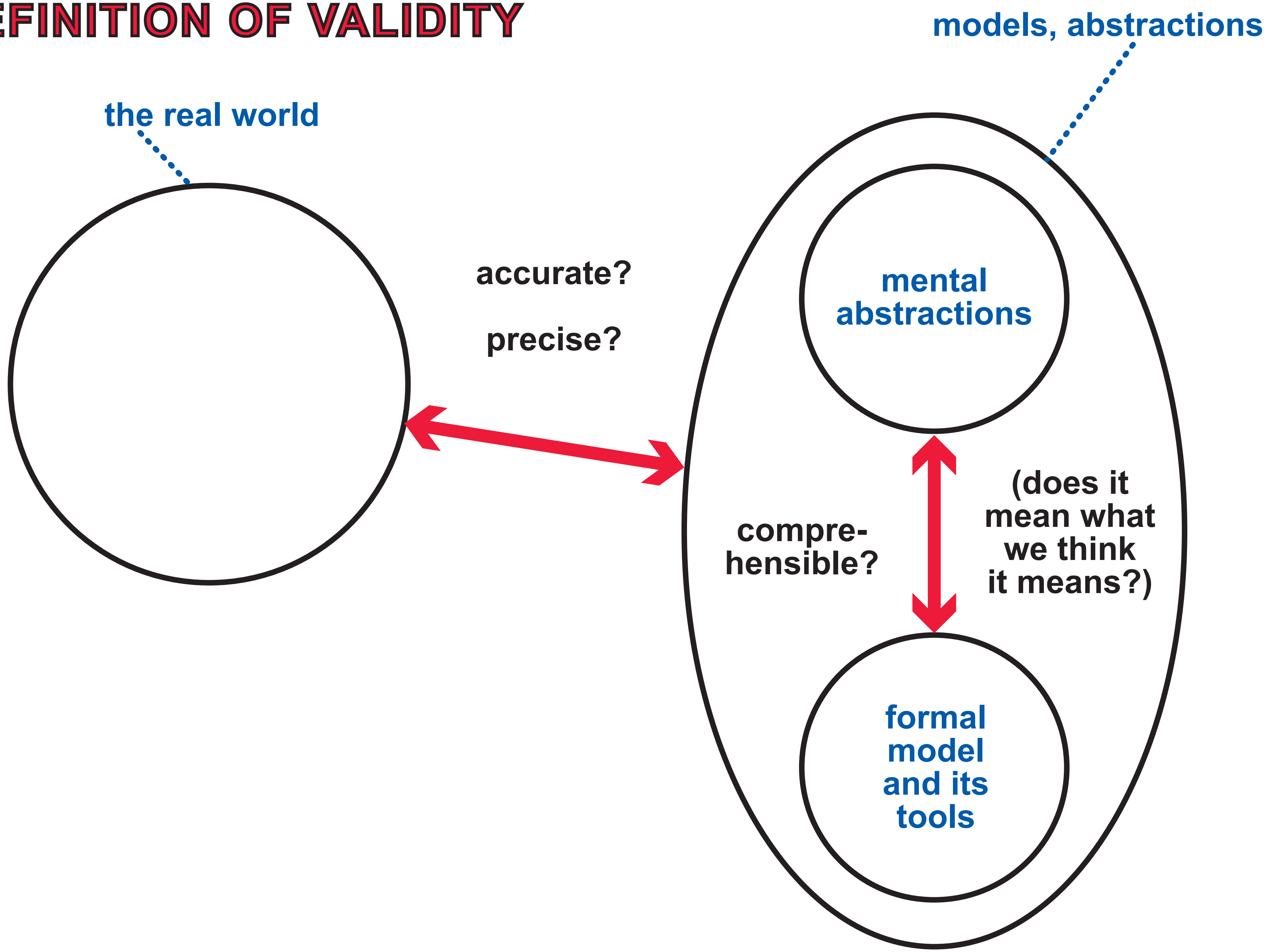
"The accuracy of a measurement is its closeness to that quantity's true value. The precision of a measurement, related to reproducibility and repeatability, is the degree to which repeated measurements under unchanged conditions show the same results."

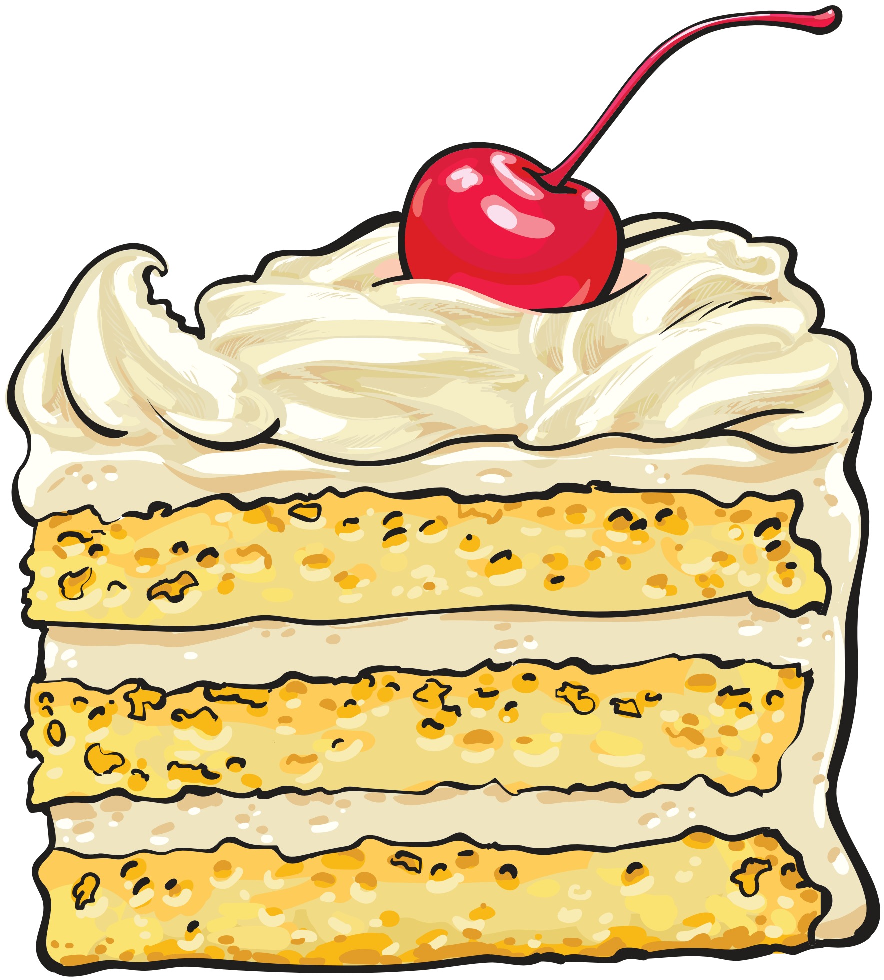
measurement:
describe a real phenomenon in terms of the formal model

repeated measurement:
ask different people to describe the same phenomenon, using the same abstractions

precision:
there is always a right answer, and a clear explanation of why a wrong answer is wrong

FORMAL MODELS: A MINIMAL DEFINITION OF VALIDITY





from *The Formalization of Baking*:

```
sig LayerCake extends Cake {
    layers: int                                  --  A layer of a layer cake is a distinct
                                                 --  horizontal stratum within the cake.
}   {   layers > 1   }
```

DESIGNATIONS

```
sig LayerCake extends Cake {
    layers: int
}    {   layers > 1   }
```

-- A *layer* of a *layer cake* is a distinct
-- horizontal stratum within the cake.

this is a *designation*—an informal description that relates a formal term to the real world

It is also a bad one! Which is why this formal model is so imprecise.

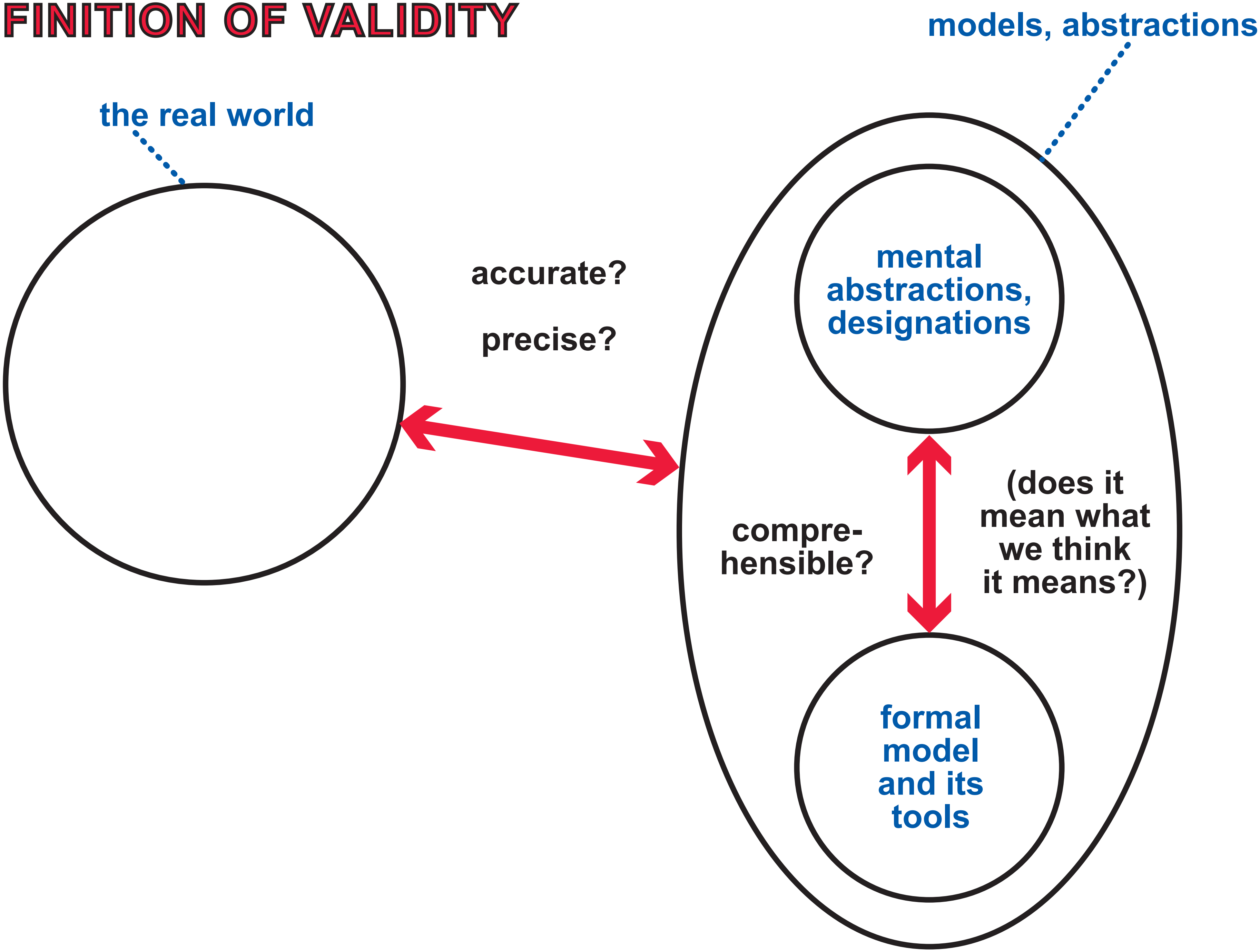
better

-- A *layer* of a *layer cake* is a distinct
-- horizontal stratum of cake within
-- the entire pastry.

Good designations are the first step toward a valid formal model.

"One can't proceed from the informal to the formal by formal means." —Alan Perlis

FORMAL MODELS: A MINIMAL DEFINITION OF VALIDITY



WHAT IS A FORMAL MODEL?

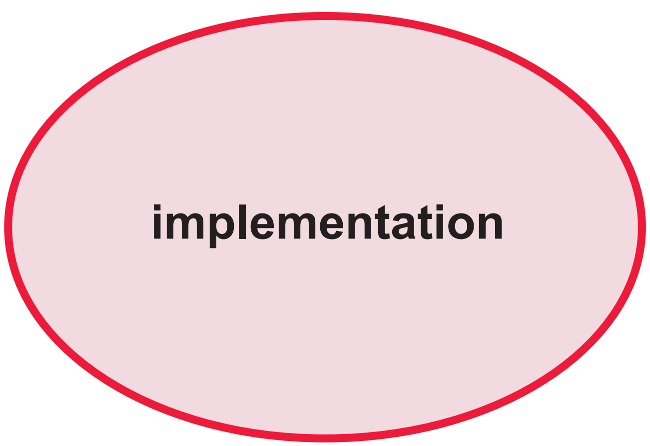
CONTENT VERSION

WHAT DO YOU CARE ABOUT?

building a computer system!

might also call it a program (software system), chip (hardware system), distributed system, etc.

there is, or eventually will be, an iMplementation *M*



WHY MAKE A FORMAL MODEL OF IT?

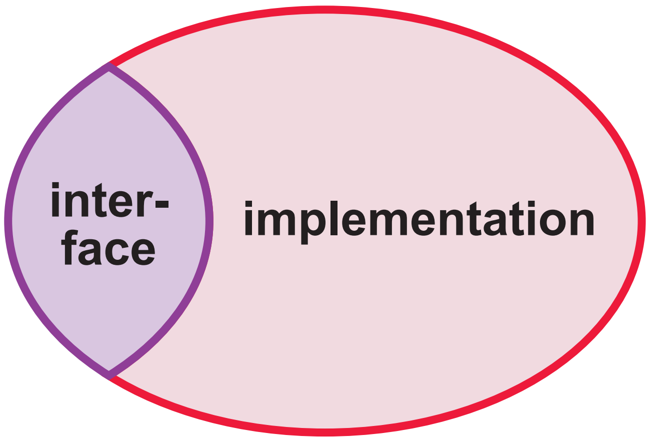
- **to verify that the implementation is correct**
- **to test the implementation thoroughly**
- **to make a contract with a customer for a system to be developed**

to do these things, you will need a Specification *S*

describes how the implementation behaves, but should be simpler and more comprehensible than the implementation

WHAT MAKES A SPECIFICATION SIMPLER?

It might be confined to what BEHAVIOR is OBSERVABLE at the system's INTERFACE.

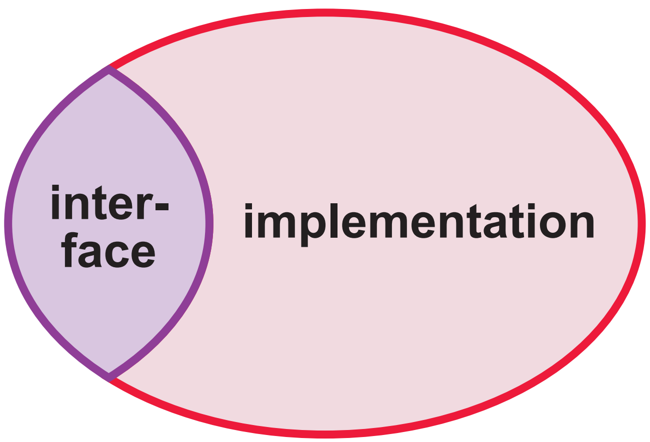


each phenomenon in the interface is either . . .

- domain-controlled (e.g., a sensor)
- system-controlled (e.g., an actuator)

WHAT MAKES A SPECIFICATION SIMPLER?

It might be confined to what BEHAVIOR is OBSERVABLE at the system's INTERFACE.



each phenomenon in the interface is either . . .

- domain-controlled (e.g., a sensor)
- system-controlled (e.g., an actuator)

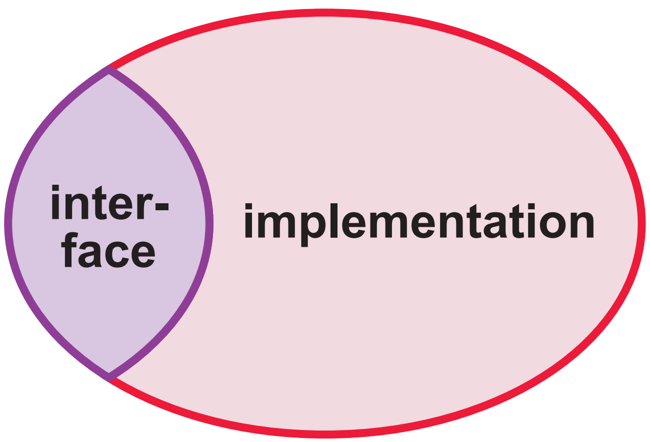
Or it might be DECLARATIVE, with the intention of deriving the implementation from it by REFINEMENT.

properties are easier to think about than whole systems—a property focuses on one narrow aspect of the system, ignores everything else

"*outList* has the same elements as *inList*"

WHAT MAKES A SPECIFICATION SIMPLER?

It might be confined to what BEHAVIOR is OBSERVABLE at the system's INTERFACE.



each phenomenon in the interface is either . . .

- domain-controlled (e.g., a sensor)
- system-controlled (e.g., an actuator)

Or it might be DECLARATIVE, with the intention of deriving the implementation from it by REFINEMENT.

properties are easier to think about than whole systems—a property focuses on one narrow aspect of the system, ignores everything else

"*outList* has the same elements as *inList*"

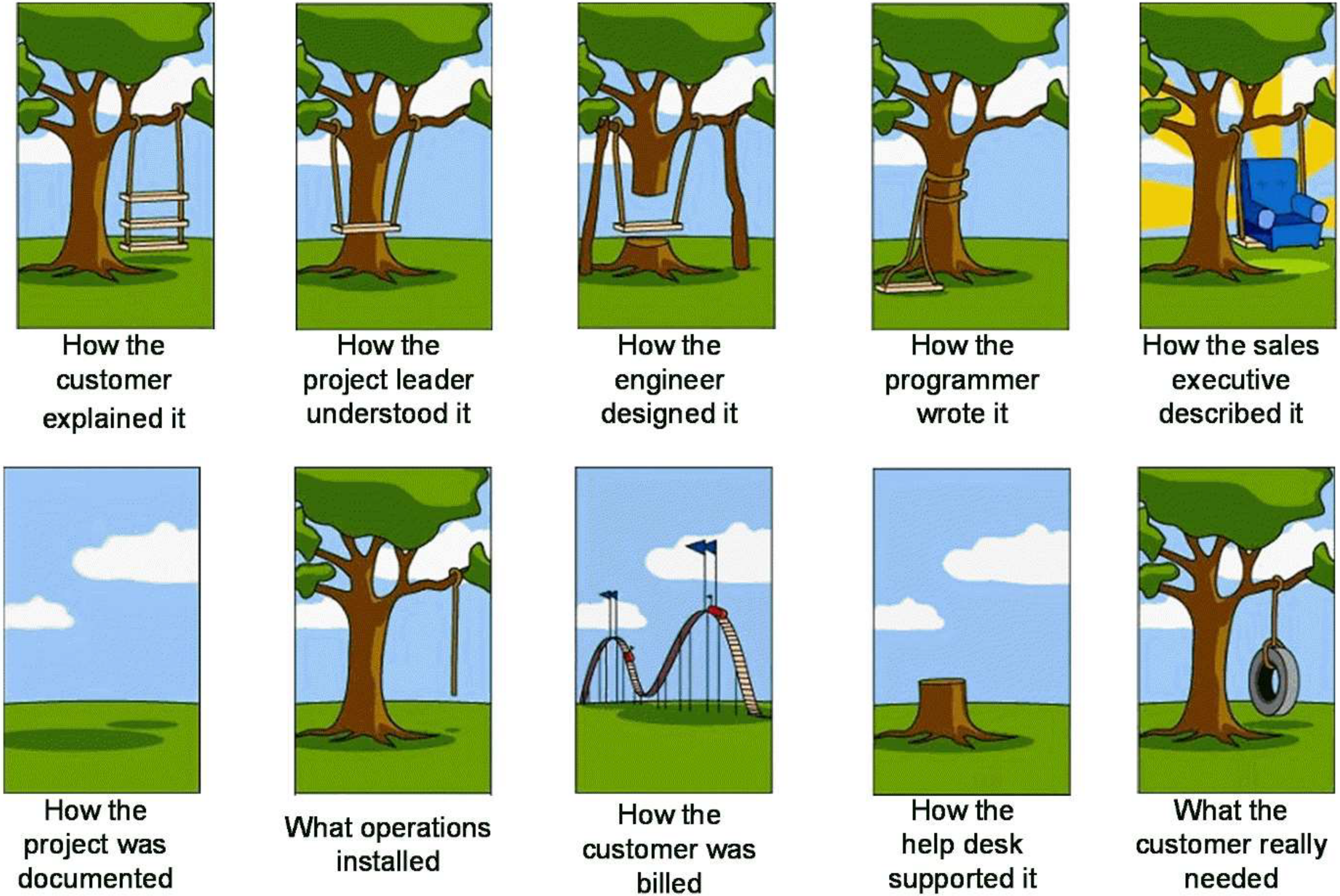
Or it might be INCOMPLETE in some WELL-DEFINED way.

It has always been difficult to

find a good-quality,

VALID specification,

and it still is.



WHY SHOULD YOU CARE ABOUT VALIDATION?

from a lecture by
Michael Hilton

institute for SOFTWARE RESEARCH

WHY SHOULD YOU CARE ABOUT VALIDATION?

IT IS THE ONLY WAY TO ENSURE THAT INPUT TO A VERIFICATION TOOL IS NOT GARBAGE

IT WILL MAKE YOU POPULAR WITH:

- **the customer, project leader, engineer, programmer, help desk**

it might make you popular with the sales executive, but don't count on it

- **teachers, students, readers of your papers**

A FORMAL MODEL COULD ALSO BE A DOMAIN MODEL

a DOMAIN MODEL . . .

attempts to formalize an entire application domain

- provides unambiguous terminology and useful abstractions for the domain
- supports a family of computer systems

is long-lasting and reusable

- provides a basis for domain-specific tools:

  specification languages
  verifiers
  visualizers
  code generators
  test generators

  without a domain model, a domain-specific view is not important or stable enough to be worth implementing

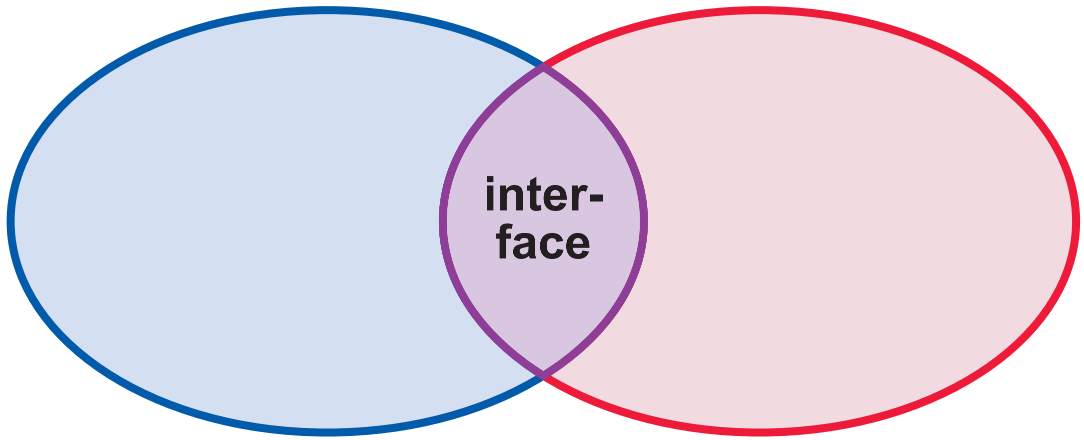
- includes artifacts of different kinds

THE CONTENT OF A DOMAIN MODEL . . . . IS MUCH BROADER THAN A SPECIFICATION

DOMAIN SYSTEM

Domain Knowledge *D*: how the domain behaves (by itself)

Implementation *M*: how the system behaves



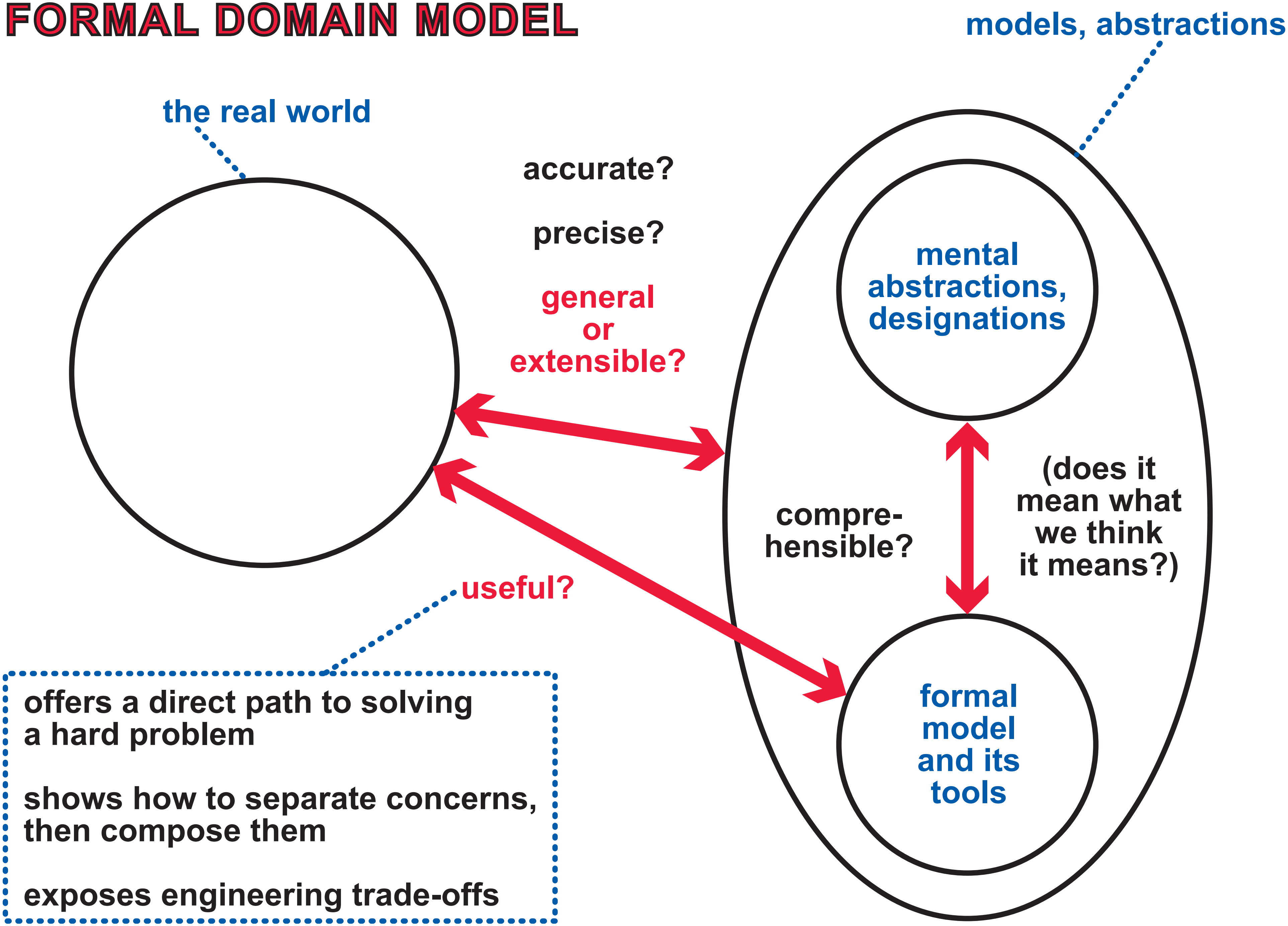
Requirements *R*: how the domain should behave (with the system in place)

Specification *S*: how the system behaves (as observed at its interface)

THE PRIMARY PROOF OBLIGATIONS

M implies *S*	verification, where *S* consists of assertions
D, *S* are consistent	validation, where *S* consists of facts
(*D* and *S*) implies *R*	

DEFINITION OF A VALID FORMAL DOMAIN MODEL

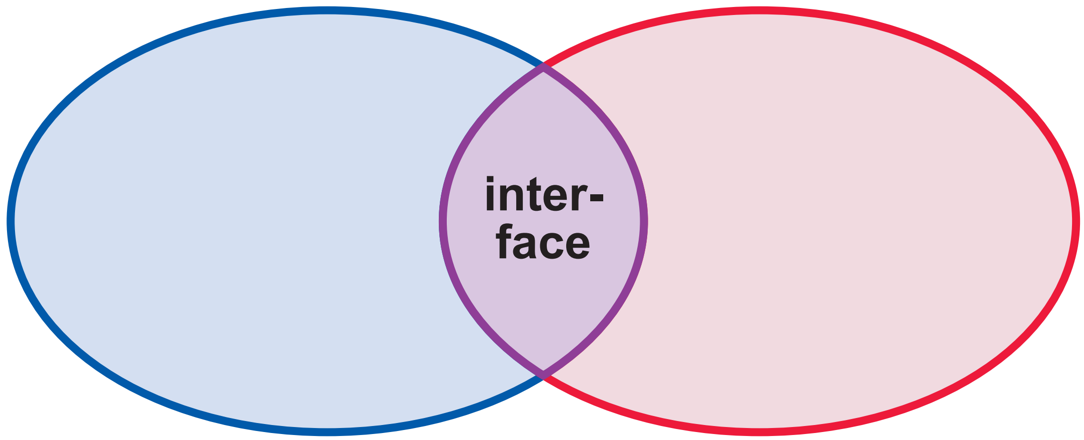


WHY SHOULD YOU CARE ABOUT VALIDATION OF DOMAIN MODELS?

DOMAIN **SYSTEM**

Domain Knowledge *D*:
how the domain
behaves (by itself)

Implementation *M*:
how the system
behaves



Requirements *R*:
how the domain
should behave (with
the system in place)

Specification *S*:
how the system
behaves (as observed
at its interface)

THE PRIMARY PROOF OBLIGATIONS

M* implies *S

D*, *S* are consistent** **(*D* and *S*) implies *R

BECAUSE THINKING ABOUT YOUR SPECIFICATION AS A DOMAIN MODEL,

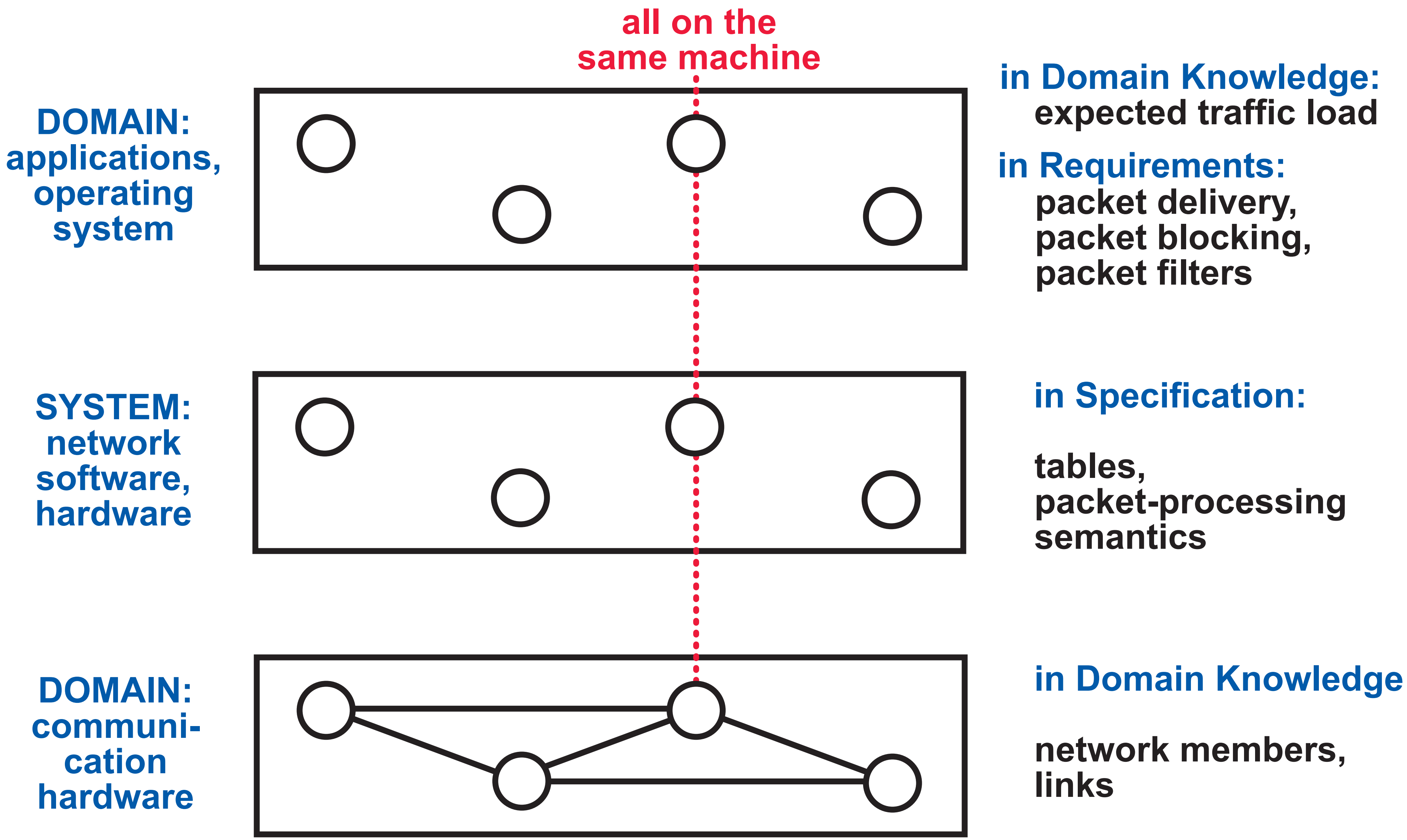
even a little,

WILL HELP YOU MAKE IT VALID!

A DOMAIN MODEL:

COMPOSITIONAL NETWORK ARCHITECTURE

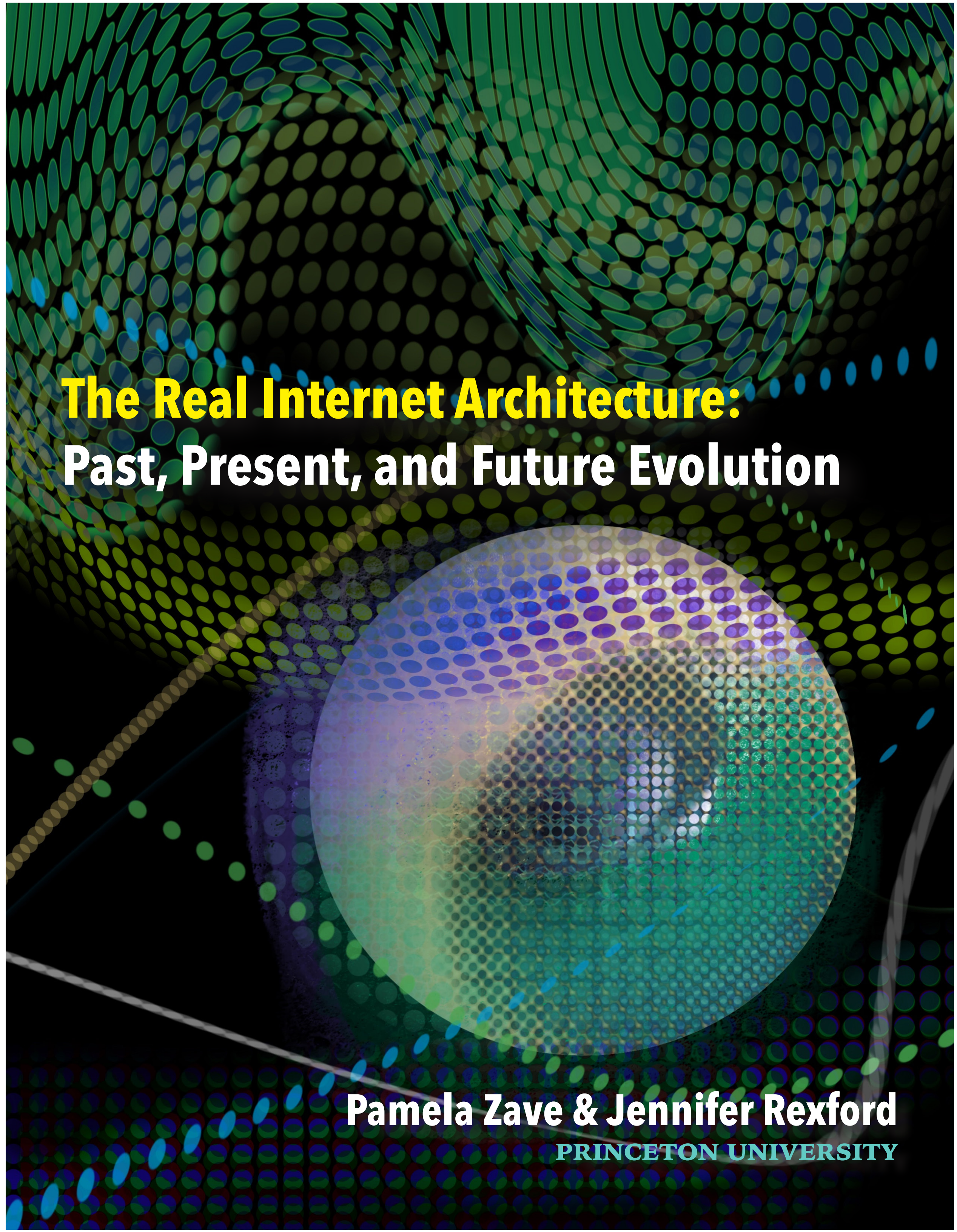
a network is a distributed system:



other domain models for networking cover:

- distributed routing algorithms
- cryptographic protocols
- performance optimization

this one provides a basis for relating communication services to network architecture



NETWORK STATE IN ALLOY

```
one sig NetworkState {
-- Network components.
   members: set Name,
   disj infras, users: set members,
   links: set Link,
   . . .
}
```

this is domain knowledge

a member has a unique name

trusted and untrusted members

Link object contains a sender and a receiver

there will be more state components, for requirements and specifications

it is a domain model

- many possible networks are instances of it

in one way, the domain model is very simple: it is static

- each instance is a trace
- each trace is the initial state of a network
- there are no state transitions, so each trace is one state long

NOW YOU HAVE A SPECIFICATION . . .

. . . HOW DO YOU VALIDATE IT?

YOU MAY HAVE:

- some requirements assertions *R*
- possibly some domain knowledge facts *D*

IF SO, GREAT!

- verify that *(D and S) implies R,*

  which often entails adding domain knowledge

BUT:

- facts and assertions can be hard to think of, you may not have many of them
- even if the verification succeeds, there is still no direct comparison of the model with the real world

ultimate goals

PRESCRIPTION: PREDICATES

WHY PREDICATES?

- a predicate is an optional property—it need only be true of one trace
- they are easy to think of
- they are easy to generalize to bigger sets of traces
- they are great for validation

in our example, one network

groups of networks with something in common

the more predicates, the better!

PRESCRIPTION: PREDICATES

WHY PREDICATES?

- a predicate is an optional property—it need only be true of one trace
- they are easy to think of
- they are easy to generalize to bigger sets of traces
- they are great for validation

the more predicates, the better!

HOW DO YOU THINK OF PREDICATES?

CATEGORIES!

- if you have a category, think of predicates in it
- if you have a predicate, think of what category it belongs in, then think of other predicates in that category

no need for a taxonomy— overlaps do no harm

THERE ARE DOMAIN-INDEPENDENT CATEGORIES . . .

- extreme examples *"inList* is empty"
- important practical examples
- categories from frameworks, e.g., domain knowledge, requirements, and specifications

. . . AND DOMAIN-SPECIFIC CATEGORIES

NETWORK STATE IN ALLOY

```
sig NetworkState {
-- Network components.
   members: set Name,
   disj infras, users: set members,
   links: set Link,
   . . .
}
```

domain knowledge

member has unique name

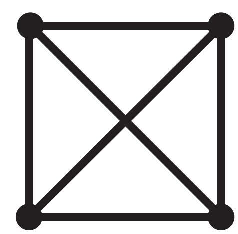
trusted and untrusted members

Link object contains a sender and a receiver

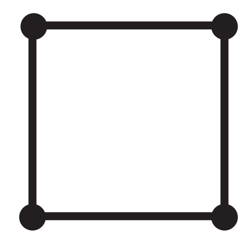
A DOMAIN-SPECIFIC CATEGORY:

TOPOLOGY PREDICATES (optional domain knowledge)

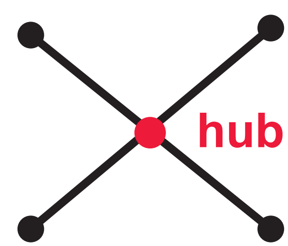
fully connected



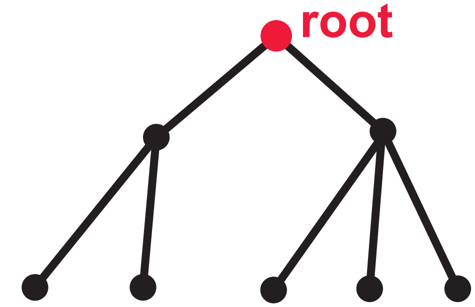
ring



hub and spoke



tree



the network components are viewed as nothing but a graph

NETWORK STATE IN ALLOY

```
sig NetworkState {

-- Network components.
   members: set Name,
   disj infras, users: set members,
   links: set Link,

-- Network traffic:

   sendTable: members -> NetHdr,

   receiveTable: members -> NetHdr,

   . . .

}
```

also domain knowledge

packets the members intend or are expected to send or receive, respectively

NetHdr object contains a source and a destination

this is a static summary of dynamic behavior

ANOTHER DOMAIN-SPECIFIC CATEGORY:

TRAFFIC PREDICATES (domain knowledge)

Users_fully_active [n: NetworkState]

there is communication between all user-member pairs

Sending_is_authentic [n: NetworkState]

a member only sends packets with its own name in the source field of the NetHdr

this is a security property—the beginning of a “threat model”

NETWORK STATE IN ALLOY

```
sig NetworkState {

-- Network components.
   members: set Name,
   disj infras, users: set members,
   links: set Link,

-- Network traffic.
   sendTable: members -> NetHdr,
   receiveTable: members -> NetHdr,

-- Network behavior.

   forwardTables:
      members -> lone ForwardTable,

   oneStep: NetHdr -> links -> links,

   reachable: NetHdr -> Name -> Name

}
```

specification

WORKING TOWARD A THIRD DOMAIN-SPECIFIC CATEGORY

Every row of a ForwardTable has this signature.

local identifier of an incoming link

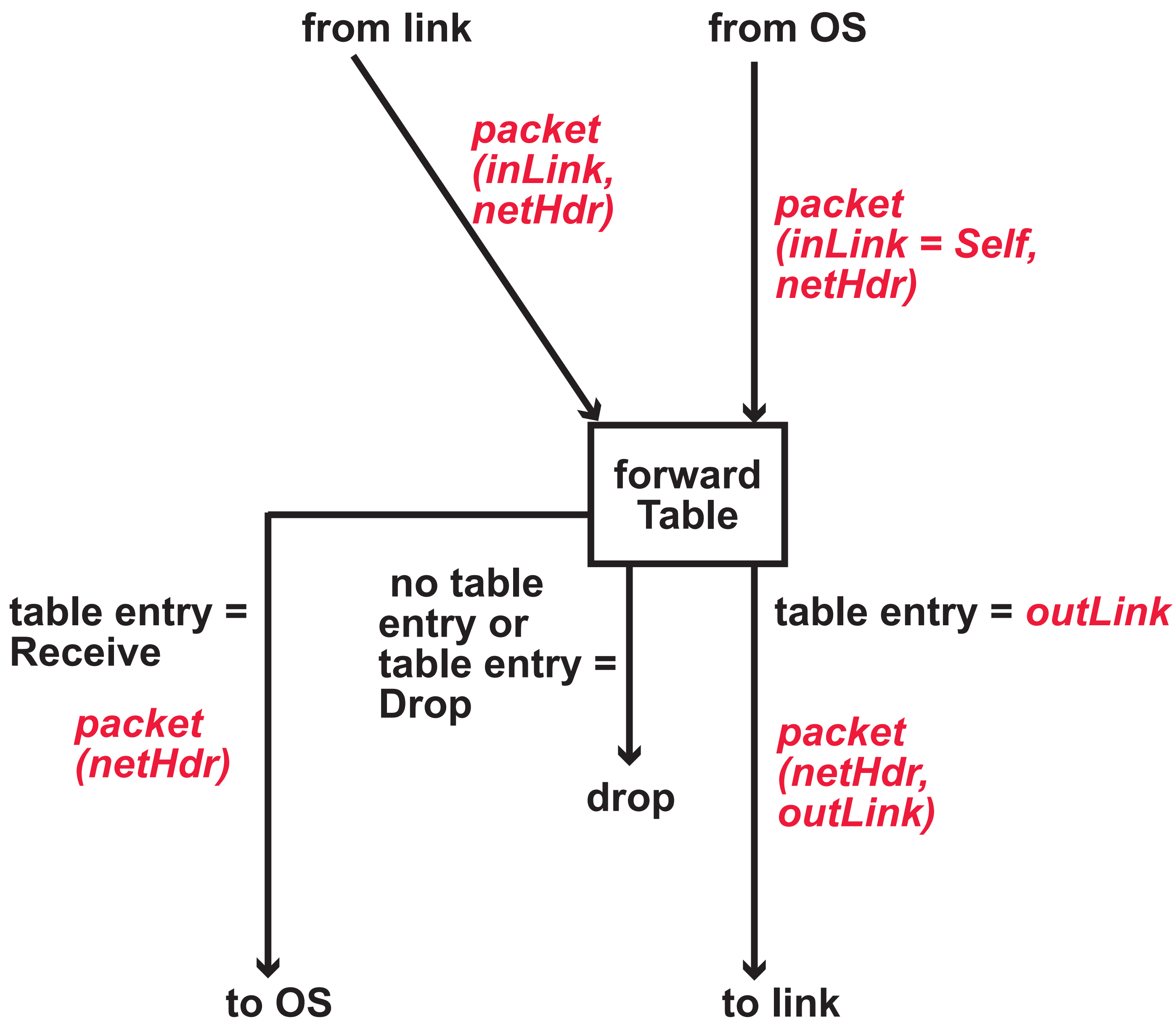
(LinkIdent + Self) -> NetHdr ->

(LinkIdent + Receive + Drop)

local identifier of an outgoing link

what the member can do instead of forwarding

PACKET PROCESSING IN A NETWORK MEMBER



NETWORK STATE IN ALLOY

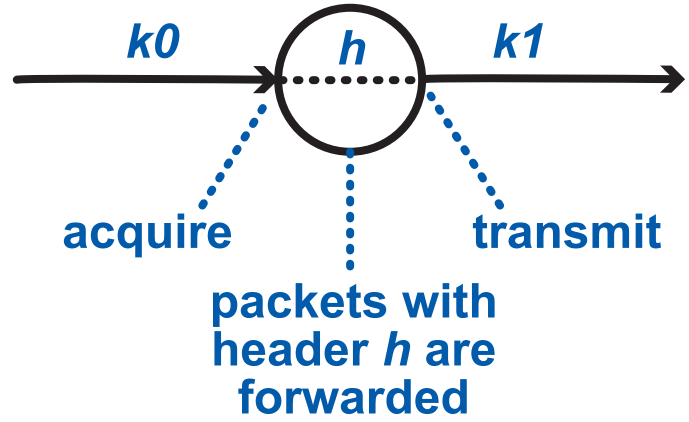
```
sig NetworkState {
   . . .
-- Network behavior.
   forwardTables:
      members -> lone ForwardTable,
   oneStep: NetHdr -> links -> links,
   reachable: NetHdr -> Name -> Name
}
```

specification (forwardTables)

derived from forwardTables and topology (oneStep, reachable)

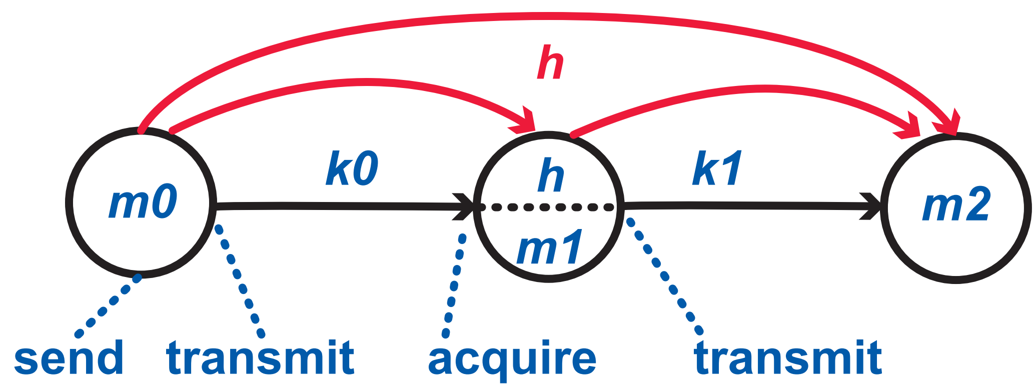
specification is . . .
. . . abstract: a static summary of dynamic behavior
. . . incomplete: failures are not modeled

(h -> k0 -> k1) in oneStep:



(h -> m0 -> m1) in reachable:

If a member *m0* transmits a packet with header *h* (on any outgoing link), then that packet will be acquired by *m1* (on some incoming link).



NETWORK STATE IN ALLOY

```
sig NetworkState {

  . . .

-- Network behavior.........specification

  forwardTables:
    members -> lone ForwardTable,

  oneStep: NetHdr -> links -> links,

  reachable: NetHdr -> Name -> Name

}
```

ANOTHER DOMAIN-SPECIFIC CATEGORY:

BEHAVIORAL PREDICATES
(optional specifications)

No_routing_loops [n: NetworkState]

packets cannot go around and around forever

Deterministic_forwarding [n: NetworkState]

for each member, incoming link, and header, there is only one entry in the forwarding table

ANOTHER DOMAIN-SPECIFIC CATEGORY:

POSSIBLE REQUIREMENTS

Fully_reachable [n: NetworkState]

every member can reach every other member

Network_satisfies_communication_demands [n: NetworkState]

for every matching pair in the send and receive tables, the sender can reach the receiver with that header

potential security requirements:

Only_authentic_traffic_delivered [n: NetworkState]

i.e., only packets with the sender's name as source

Delivery_is_blocked [n: NetworkState, disj bad, good: Name]

bad cannot reach good, with any header

Delivery_is_filtered [n: NetworkState, disj filter, good: Name]

all packets to good pass through filter

VALIDATION WITH PREDICATES

1 Instantiate all the predicates and look at the instances.

Why are these good rules?

When you are looking at tool output (instances), which is a lot of work, you will know . . .
. . . what you are looking *at*,
. . . what you are looking *for*,
. . . and *why* it is significant.

none of this is true for randomly-generated instances!

2 Also instantiate many Boolean combinations of them.

for optional *P, Q:*
P and Q, !P and Q,
P and !Q, !P and !Q

There will be *many* things to check with your tool.

There will be *many* bugs and other surprises, each of which you can learn from.

Why would I instantiate a predicate like ! No_routing_loops?

The possibility of loops is inherent in distributed routing and forwarding. If the structure of forwarding tables prohibits them, the structure is probably too restrictive to perform many useful functions.

VALIDATION WITH PREDICATES, CONTINUED

Why are these good rules?

3 Many of your predicates can be combined to make assertions about domain knowledge and the specification. Think of these and verify them, no matter how trivial they might seem. These are "sanity checks."

A sanity check is more powerful for validation than a predicate, because it must hold for all instances.

```
all n: NetworkState | Hub_and_spoke [n] => Spanning_tree [n]
```

although the definitions look very different

```
all n: NetworkState |
  NetHdr.(n.reachable) in (n.members -> n.members)
```

could have defined reachable to include external members, but I don't think I did

4 Sometimes a sanity check is quite important and valuable, because it gives new insight into some aspect of a domain model.

Assertions are the hardest properties to think of, and now you have some new ones!

SUMMARY

THINK OF YOUR MODEL AS A DOMAIN MODEL—RELEVANT TO A FAMILY OF SYSTEMS—NO MATTER HOW SPECIFIC YOUR GOALS REALLY ARE

in other words,
generalize whatever you can

think of the domain knowledge
and requirements
as well as the specification

PREDICATES ARE GREAT FOR VALIDATION

provided you have a tool
that will generate instances of them!

- with the help of categories, you can think of many predicates
- you can generate many instances
- instances are focused and meaningful
- instances can be compared to the real world that the model is supposed to describe
- predicates are a mental springboard for thinking of sanity checks, which can even become important assertions