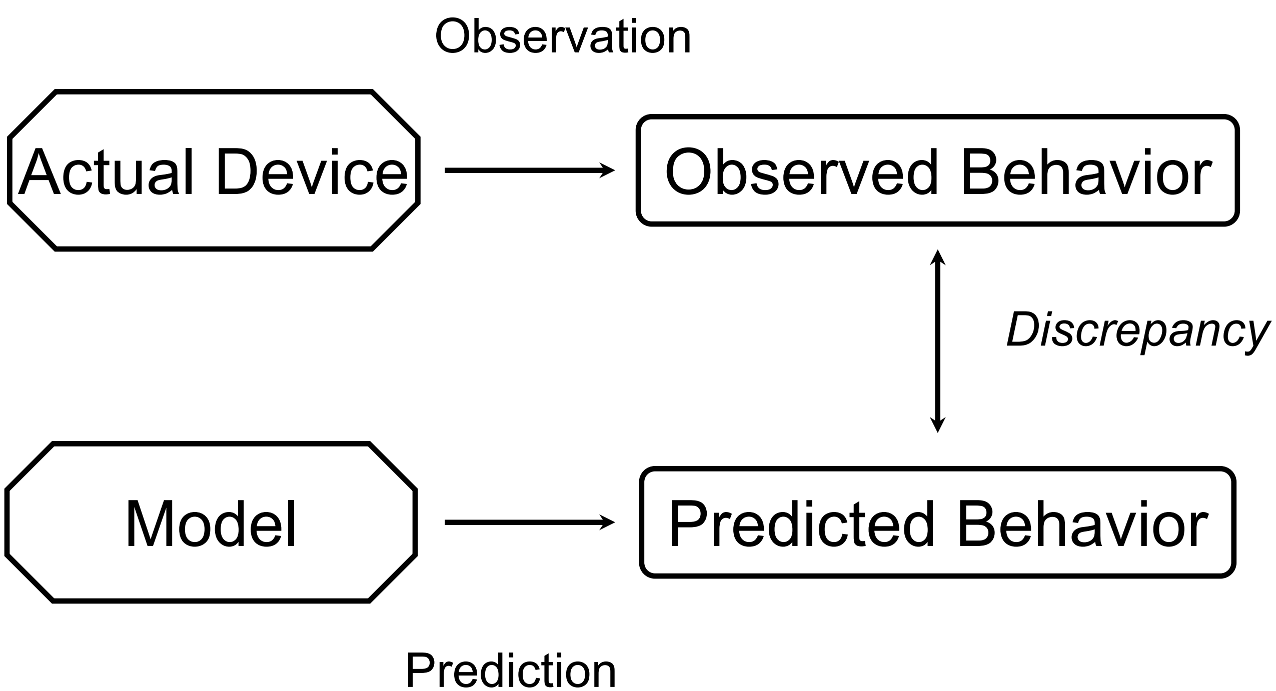
# Model Based Reasoning

6.871 - Lecture 15

# Outline

- Basics of the task
- The nature of models
- What we know how to do
- What we don't know how to do (so well)

# Interaction of Prediction and Observation



# Components of the Task

- Given
  - Observations of a device behavior (inputs, outputs)
  - a description of internal structure
  - a description of component behavior
- Determine
  - which components could have failed so as to product the observed misbehavior
  - the simplest set of component failures which can explain the misbehavior
- Buzzwords
  - Reasoning from design models
  - Reasoning from first principles
  - Deep reasoning

# Why Model Based Diagnosis

- Familiar task that people do well

- Compared to heuristic classification
  – Don't need new rule set needed for each device
  – Device independent
  – "Free" given a design description

- Compared to traditional diagnostics
  – Diagnosis is not verification or manufacturing testing
  – Symptom directed
  – Can cover a wider range of faults



- for determining what broke, not whether it's correctly manufactured
More focused and precise

# When not to use it

- Some things are too difficult to infer from the models
  - intermittent or flaky behavior
- The device and range of faults is small enough to permit exhaustive simulation
- The device and range of faults is small enough to generate an exhaustive fault dictionary

# Basic Theses

- Hypothesis generation, test and discrimination are fundamental problems of diagnosis
- Different amounts and types of knowledge can be brought to bear at each phase
- The set of possibilities explored spans a wide range of potential systems within this common PSP
- More complex devices require better abstractions.

# Useful Characteristics of Structure Representations

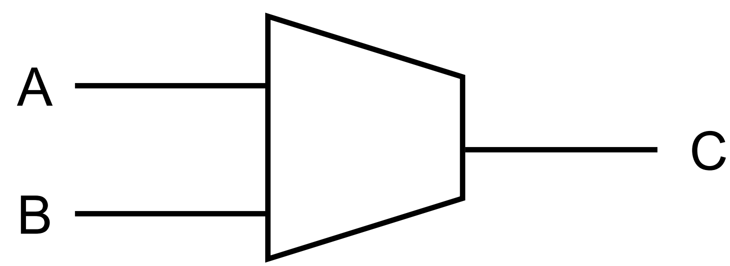
- Hierarchical
  - Possibly multiple: behavioral, physical
  - Possibly not strict: components with multiple functional roles

- Object-oriented, isomorphic to the device
  - Procedural objects
  - Interconnected in same topology

- Unified: Both runnable and examinable

# Behavior Representation

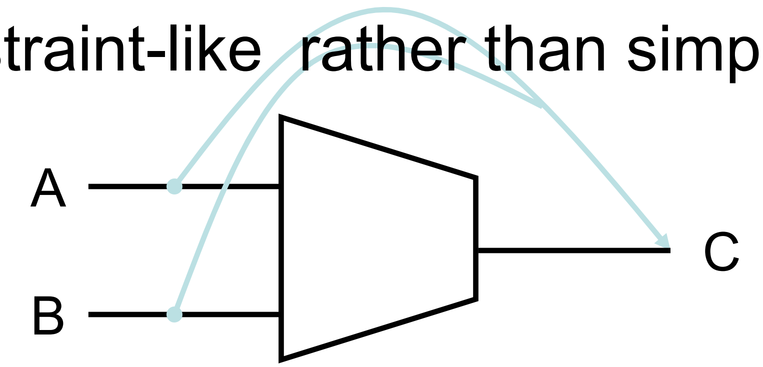
- Expressions capturing relationships between values at terminals
  - Multi-directional
  - Constraint-like  rather than simply procedural




    - To compute C:  Evaluate A + B

# Behavior Representation

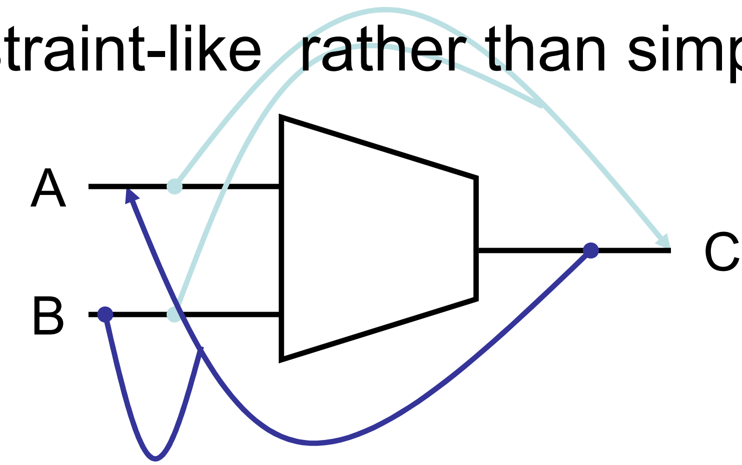
- Expressions capturing relationships between values at terminals
  - Multi-directional
  - Constraint-like  rather than simply procedural




- To compute C:  Evaluate A + B

# Behavior Representation

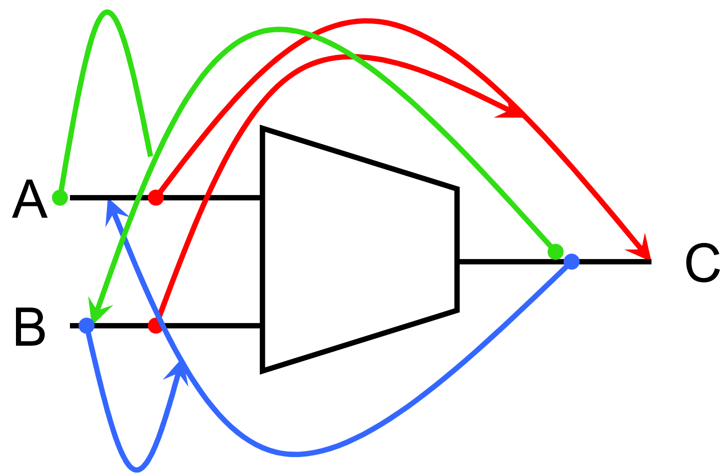
- Expressions capturing relationships between values at terminals
  - Multi-directional
  - Constraint-like  rather than simply procedural




  - To compute C:  Evaluate A + B
  - To compute A:  Evaluate C - B

# Behavior Representation

- Expressions capturing relationships between values at terminals
  - Multi-directional
  - Constraint-like  rather than simply procedural




- To compute C:  Evaluate A + B
- To compute A:  Evaluate C - B
- To compute B:  Evaluate C - A

# Three Fundamental Problems

- Hypothesis Generation
  - Given a symptom, which components could have produced it?
  - (Which are most likely to have produced it)

- Hypothesis Testing
  - Which components could have failed to account for all observations?

- Hypothesis Discrimination
  - What additional information should we acquire to distinguish among the remaining candidates?

# Generation

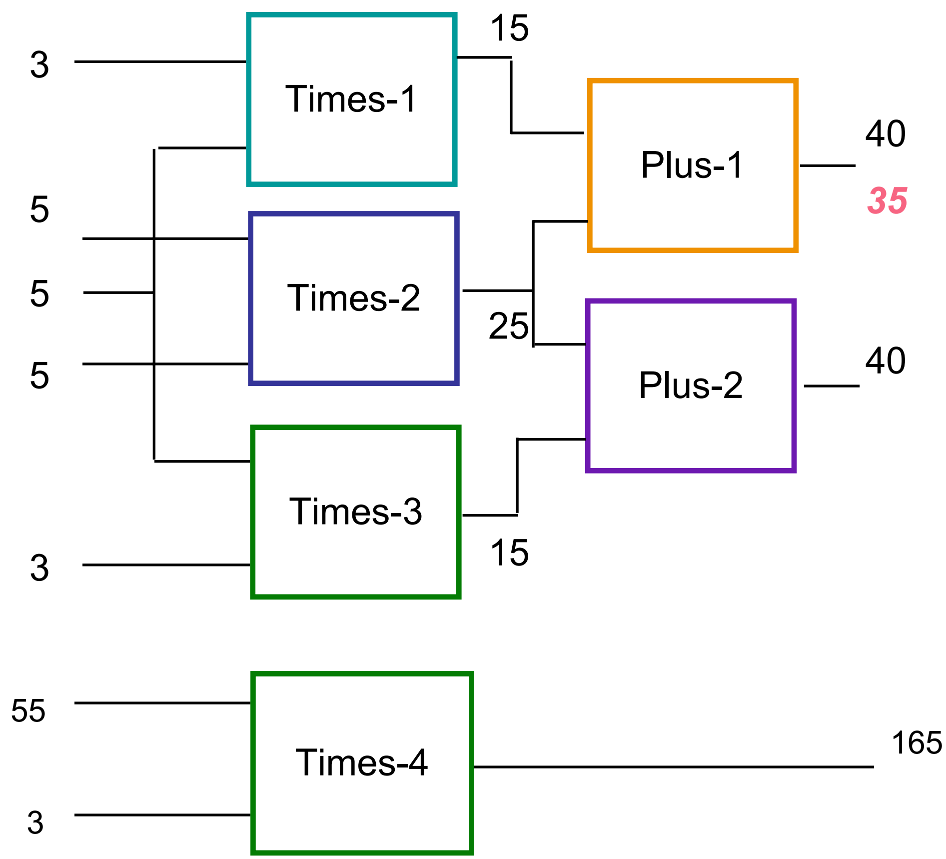
- Generator provides plausible hypotheses
  - Complete
  - Non-redundant
  - *Informed*

# Generation

## G1: Exhaustive enumeration of components

# Generation

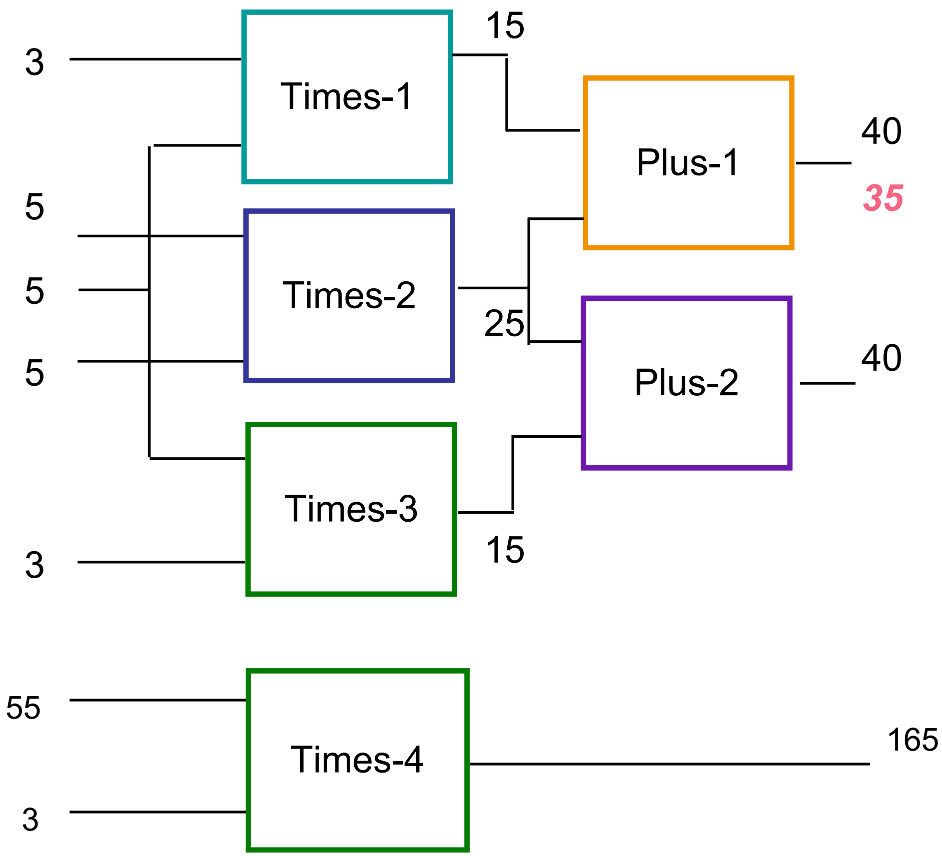
## G1: Exhaustive enumeration of components



# Generation

*But: to be a candidate, component must have contributed to the discrepancy*

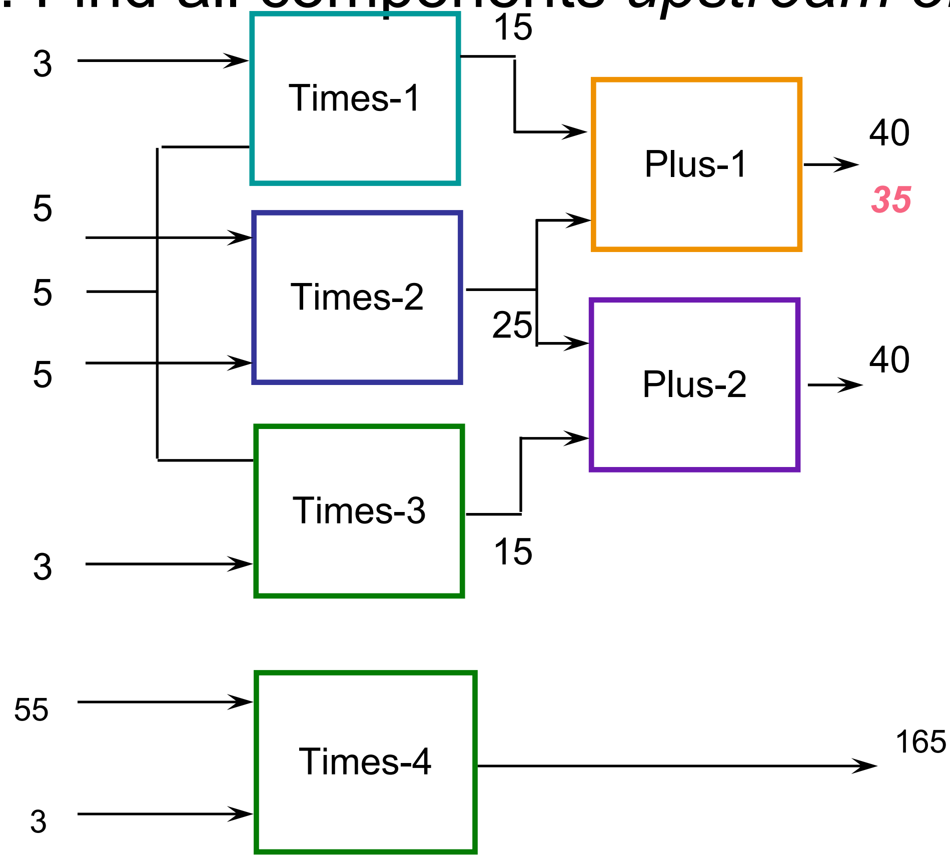
- G2: Find all components connected to the discrepancy



# Generation

*But: devices have distinguishable inputs and outputs*

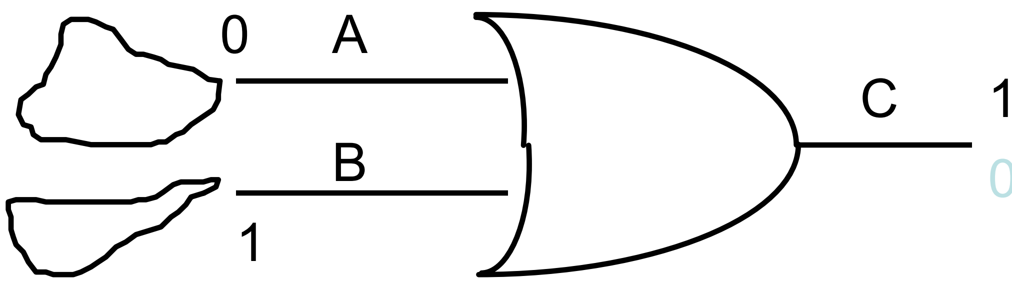
- G3: Find all components *upstream of* the discrepancy



# Generation

*But: Not every input influences the specified output*

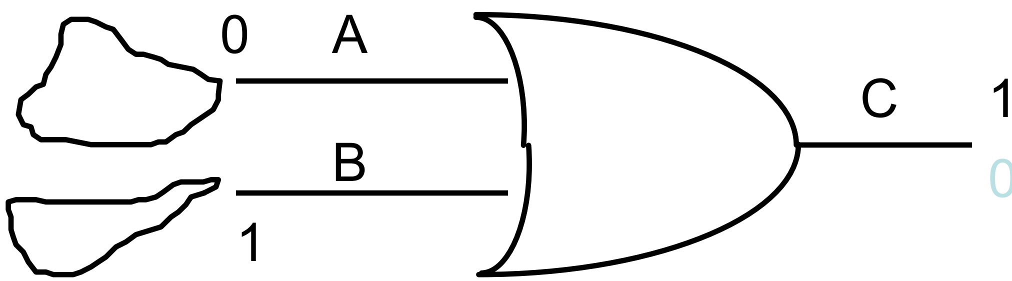
- G4: Use behavior model to determine relevant inputs
  - Have simulation keep dependency records
  - Trace back through these to determine candidates



# Generation

*But: Not every input influences the specified output*

- G4: Use behavior model to determine relevant inputs
  - Have simulation keep dependency records
  - Trace back through these to determine candidates

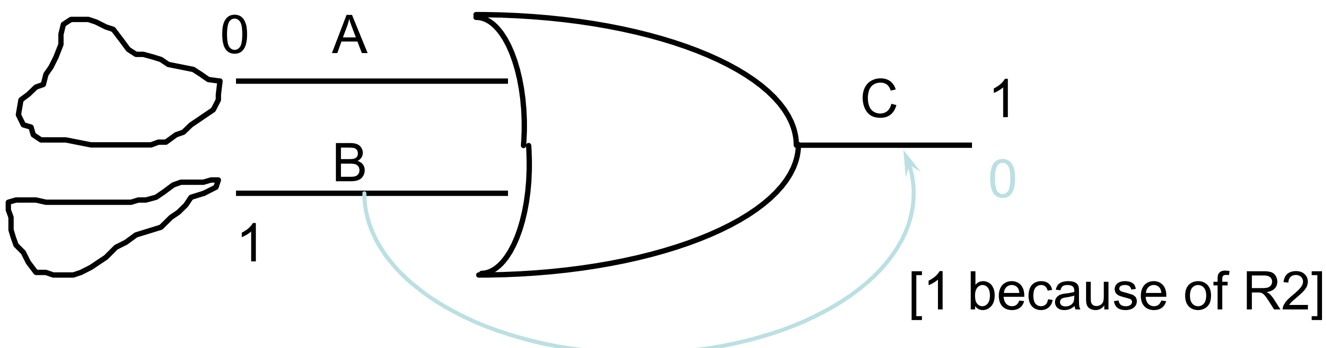



R1: IF A=1 then C=1
R2: IF B=1 then C=1
R3: IF A=0 and B=0 then C= 0

# Generation

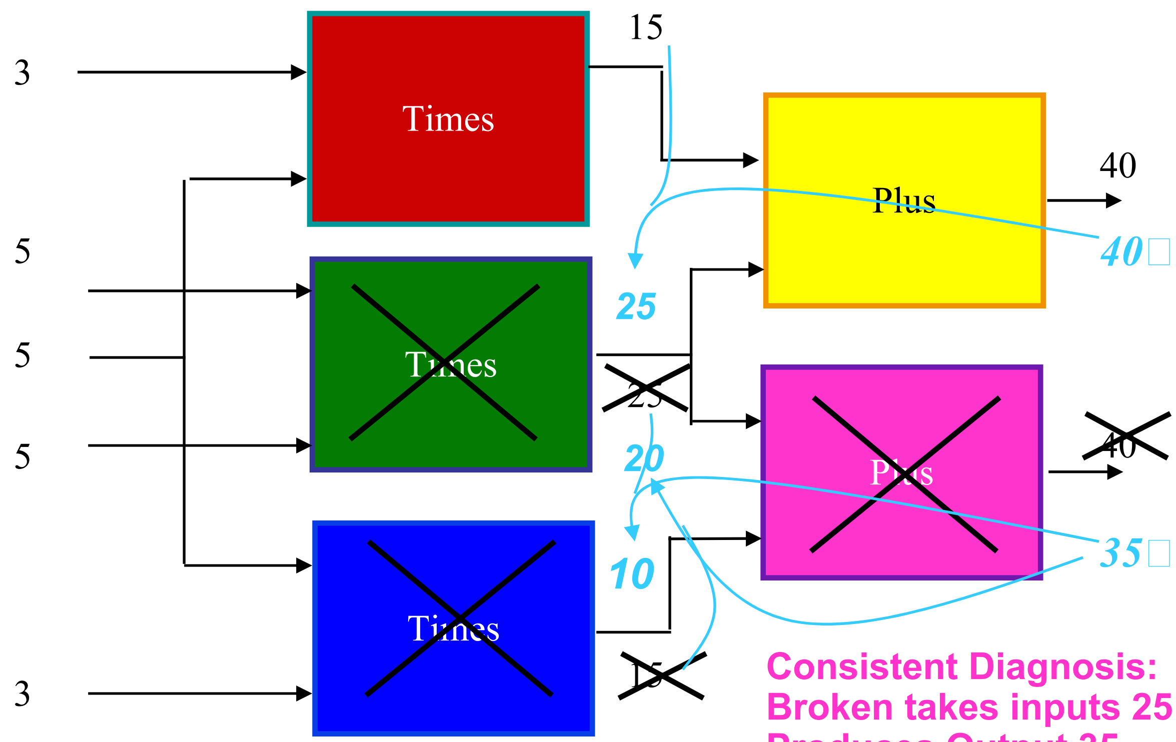
*But: Not every input influences the specified output*

- G4: Use behavior model to determine relevant inputs
  - Have simulation keep dependency records
  - Trace back through these to determine candidates




R1: IF A=1 then C=1
R2: IF B=1 then C=1
R3: IF A=0 and B=0 then C= 0

# Model Based Troubleshooting Constraint Suspension




Consistent Diagnosis:
Broken takes inputs 25 and 15
Produces Output 35

Consistent Diagnosis:
Broken takes inputs 5 and 3
Produces Output 10

No Consistent Diagnosis:
Conflict between 25 & 20

# Generation

Generators should be:

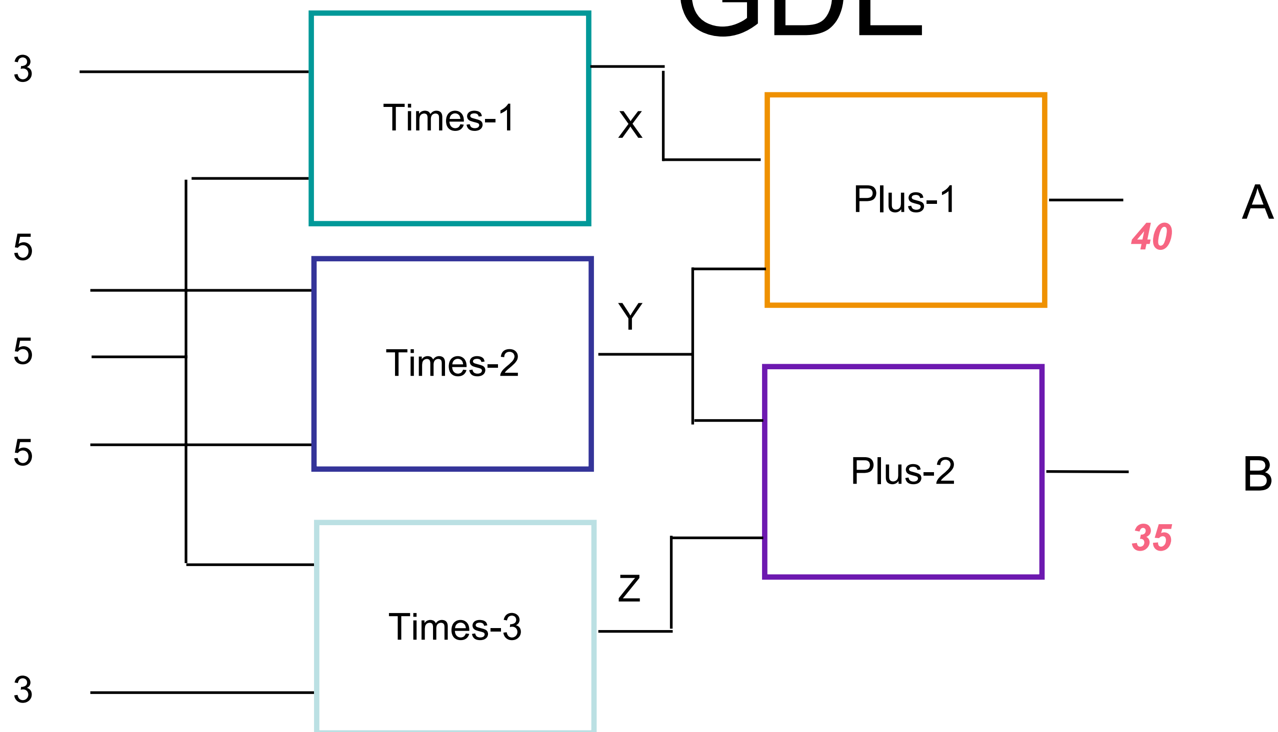
- Complete
- Non-redundant
- *Informed*

- G1: Exhaustive enumeration of components
- G2: Find all components connected to the discrepancy
- G3: Find all components *upstream of* the discrepancy
- G4: Use behavior model to determine relevant inputs

# Using Behavior Information: GDE

- GDE = General Diagnostic Engine
- Propagate not just values, but underlying assumptions as well
  - Assumptions are the proposition that a component is working according to design

# Model Based Troubleshooting GDE

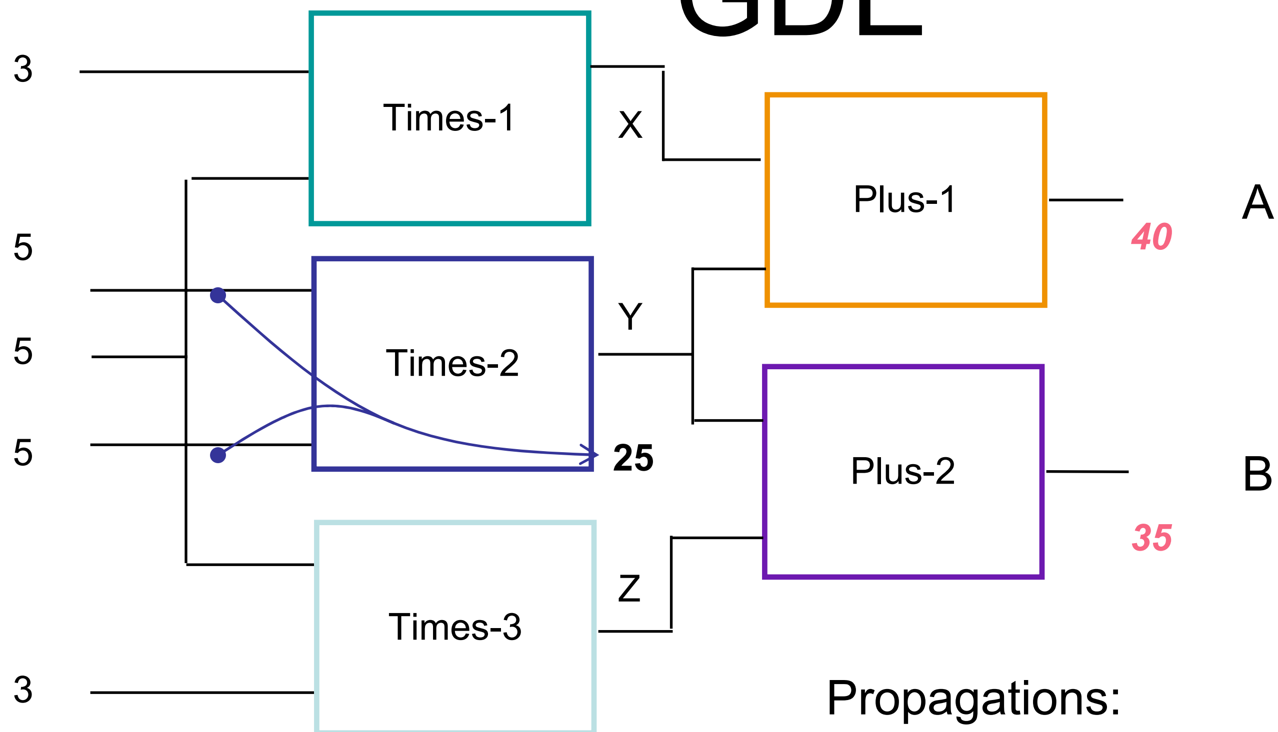



Assume P1 T1 working ==> Y=25 (P1 T1)
Assume P2 T3 working ==> Y=20 (P2 T3)
Assume T2 working ==> Y= 25 (T2)

Conflicts: (P1 T1 P2 T3)
(P2 T2 T3)

Diagnoses: (P2) (T3) (P1 T2) (T1 T2)

# Model Based Troubleshooting GDE



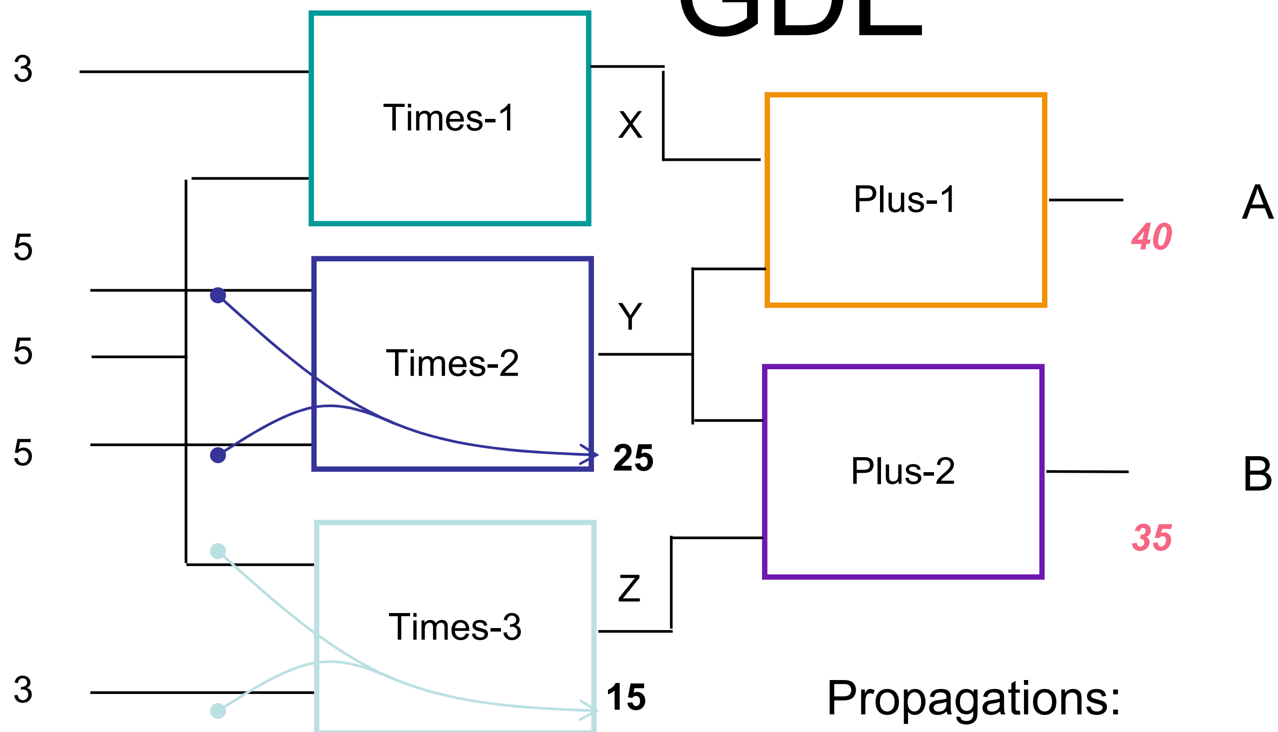

Propagations:
Y=25 (T2)

Assume P1 T1 working  ==> Y=25 (P1 T1)
Assume P2 T3 working  ==> Y=20 (P2 T3)
Assume T2 working  ==>  Y= 25 (T2)

Conflicts: (P1 T1 P2 T3)
(P2 T2 T3)

Diagnoses: (P2)  (T3)  (P1 T2)  (T1 T2)

# Model Based Troubleshooting
# GDE



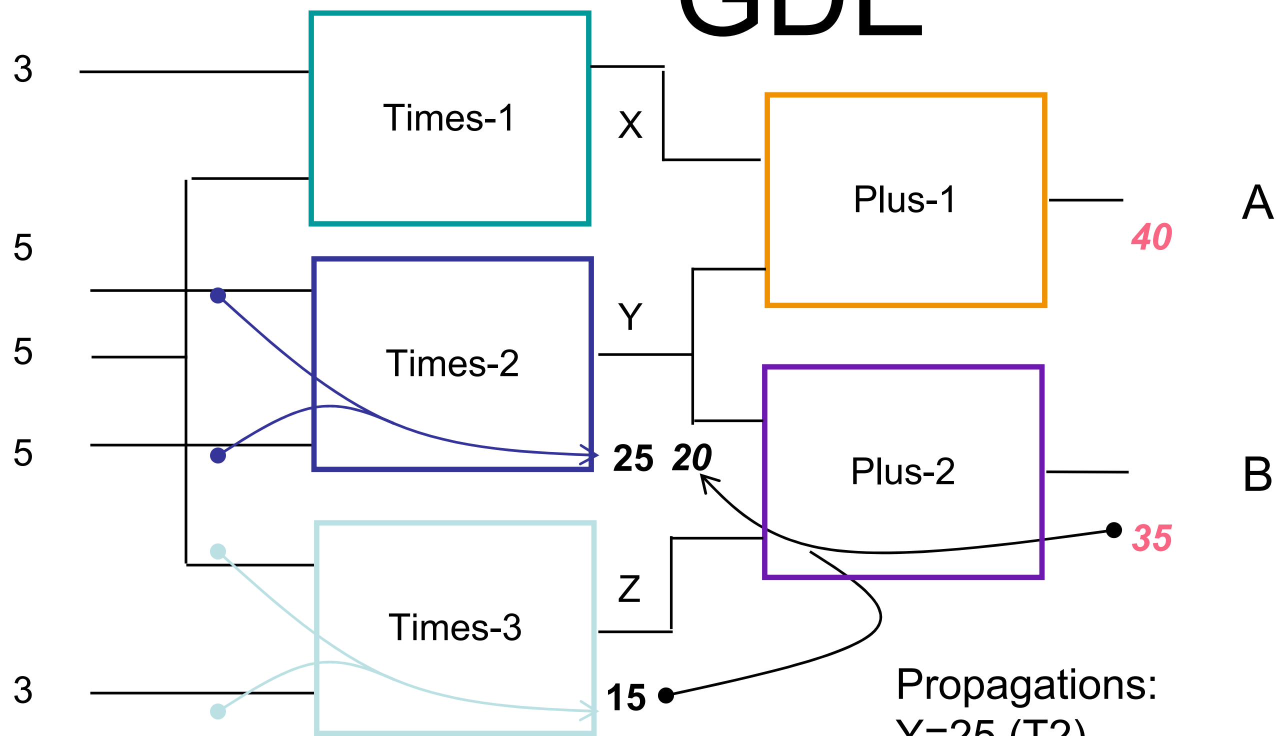

Propagations:
Y=25 (T2)
Z=15 (T3)

Assume P1 T1 working ==> Y=25 (P1 T1)
Assume P2 T3 working ==> Y=20 (P2 T3)
Assume T2 working ==> Y= 25 (T2)

Conflicts: (P1 T1 P2 T3)
(P2 T2 T3)

Diagnoses: (P2) (T3) (P1 T2) (T1 T2)

# Model Based Troubleshooting GDE



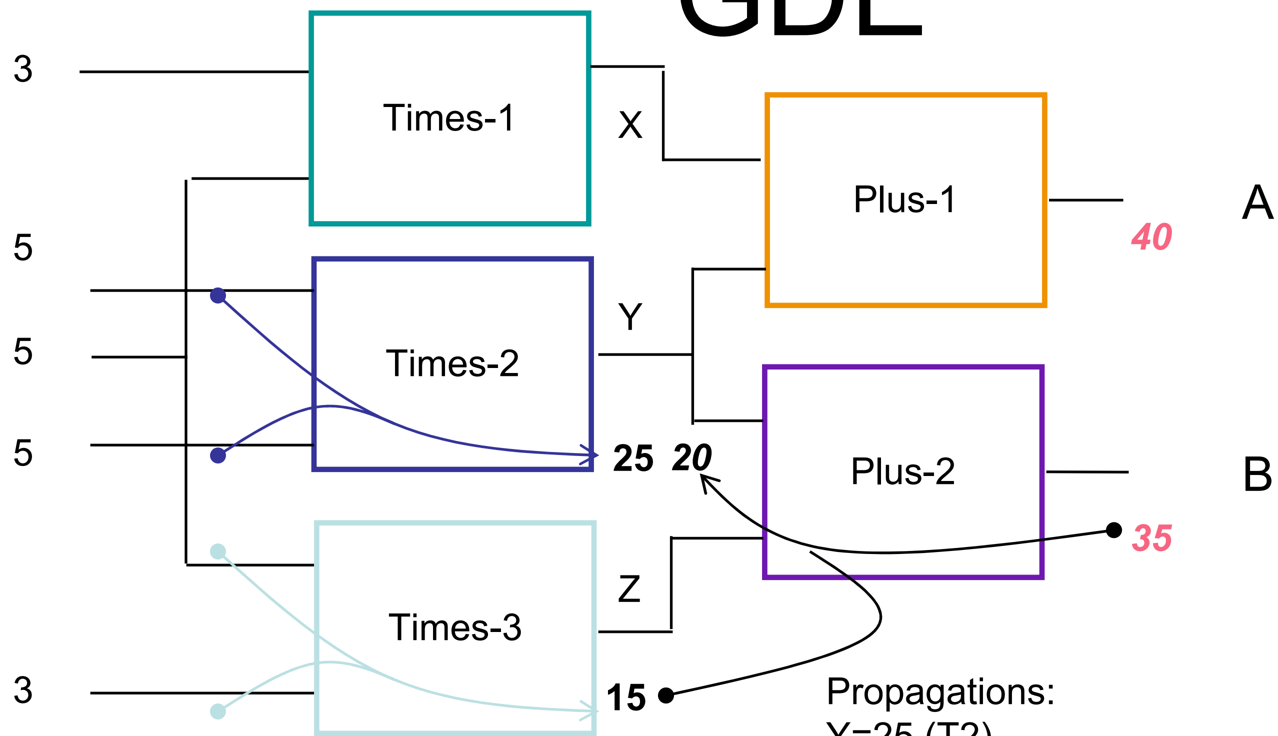

Propagations:
Y=25 (T2)
Z=15 (T3)
Y=20 (P2 T3)

Assume P1 T1 working ==> Y=25 (P1 T1)
Assume P2 T3 working ==> Y=20 (P2 T3)
Assume T2 working ==> Y= 25 (T2)

Conflicts: (P1 T1 P2 T3)
(P2 T2 T3)

Diagnoses: (P2) (T3) (P1 T2) (T1 T2)

# Model Based Troubleshooting GDE




Propagations:
Y=25 (T2)
Z=15 (T3)
Y=20 (P2 T3)

Conflict Sets:

(T2 T3 P2)

Assume P1 T1 working ==> Y=25 (P1 T1)
Assume P2 T3 working ==> Y=20 (P2 T3)
Assume T2 working ==> Y= 25 (T2)

Conflicts: (P1 T1 P2 T3)
(P2 T2 T3)
Diagnoses: (P2) (T3) (P1 T2) (T1 T2)

# Using Behavior Information: GDE Assumption Propagation and Set Covering

- GDE = General Diagnostic Engine
- Propagate not just values, but underlying assumptions as well
  - Assumptions are the proposition that a component is working according to design
- Construct conflict sets
  - Sets of assumptions, not all of which can be true at once
    eg: (T2 T3 P2)
    (T1 T3 P1 P2)
- "Explain" each conflict set

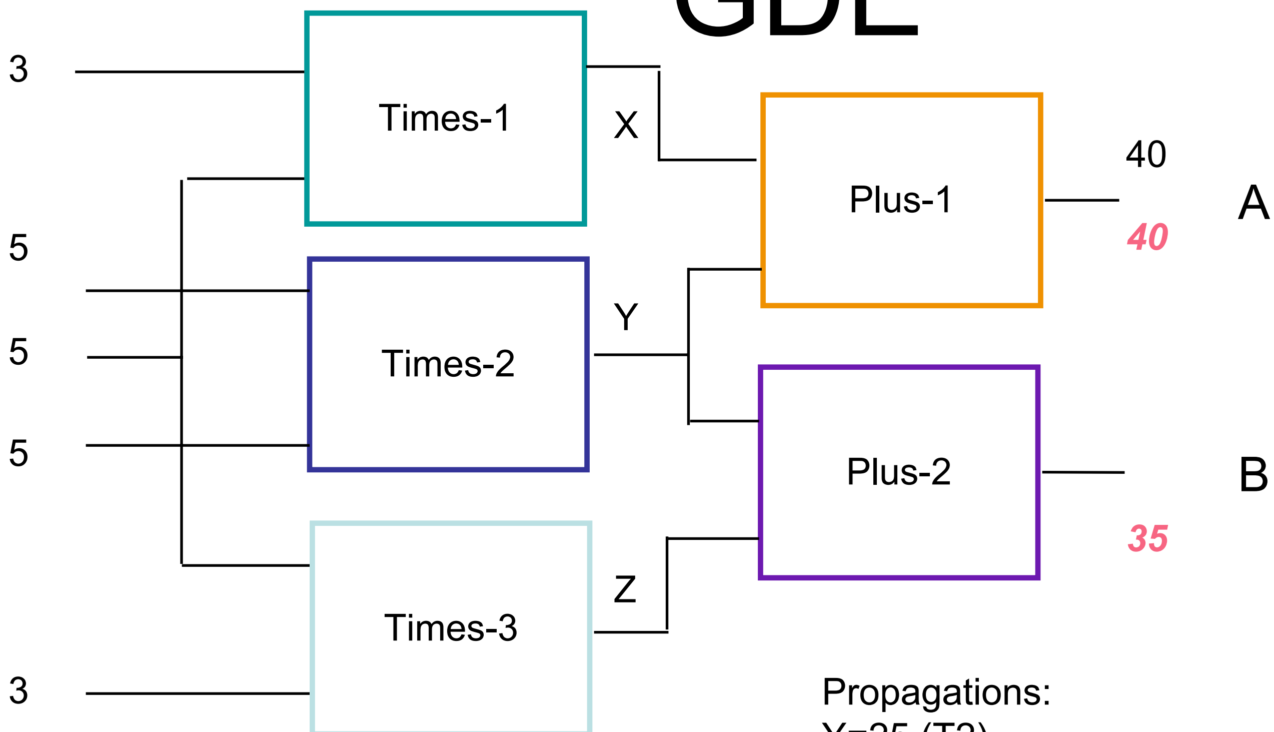
# Using Behavior Information: GDE Assumption Propagation and Set Covering

- GDE = General Diagnostic Engine
- Propagate not just values, but underlying assumptions as well
  - Assumptions are the proposition that a component is working according to design
- Construct conflict sets
  - Sets of assumptions, not all of which can be true at once
    eg: (T2 T3 P2)
    (T1 T3 P1 P2)
- "Explain" each conflict set
  - By a set covering
    eg: (P2) (T3 P2)

# Using Behavior Information: GDE Assumption Propagation and Set Covering

- GDE = General Diagnostic Engine
- Propagate not just values, but underlying assumptions as well
  - Assumptions are the proposition that a component is working according to design
- Construct conflict sets
  - Sets of assumptions, not all of which can be true at once
    eg: (T2 T3 P2)
    (T1 T3 P1 P2)
- "Explain" each conflict set
  - By a set covering
    eg: (P2) (T3 P2)
  - By a *minimal* set covering: eg: (T3)

# Model Based Troubleshooting GDE




Propagations:
Y=25 (T2)
Y=20 (P2 T3)
Y=25 (T1 P1)
Y=20 (T3 P2)

Conflict Sets:

(T2 P2 T3)

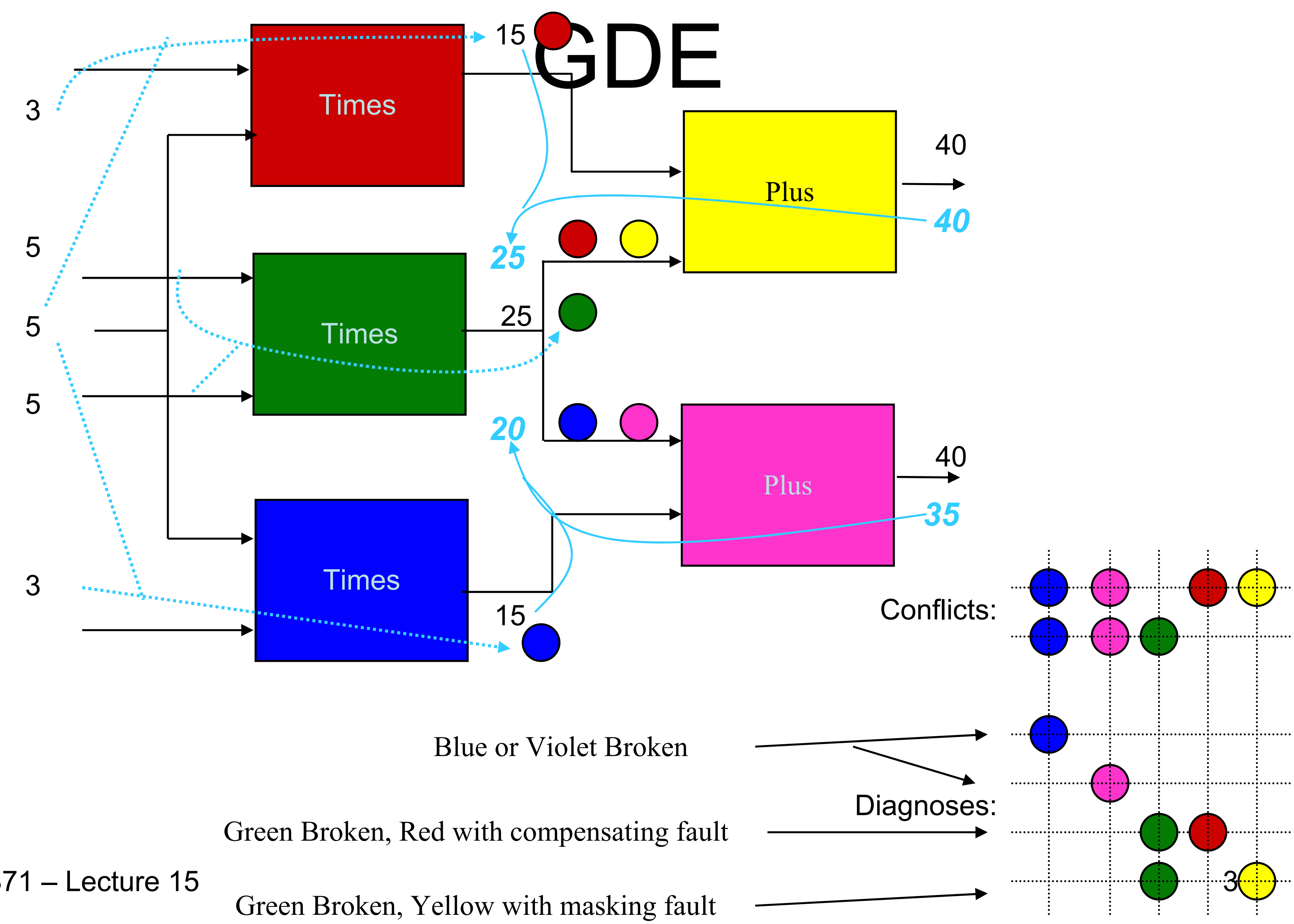
(T1 P1 T3 P2)

Assume P1 T1 working ==> Y=25 (P1 T1)
Assume P2 T3 working ==> Y=20 (P2 T3)
Assume T2 working ==> Y= 25 (T2)

Conflicts: (P1 T1 P2 T3)
(P2 T2 T3)
Diagnoses: (P2) (T3) (P1 T2) (T1 T2)

Diagnoses: (P2) (P1 T2) ...

# Model Based Troubleshooting GDE



# Good News/Bad News

- The good news
  - Generates all the logically possible candidates
  - Including multiple point of failure

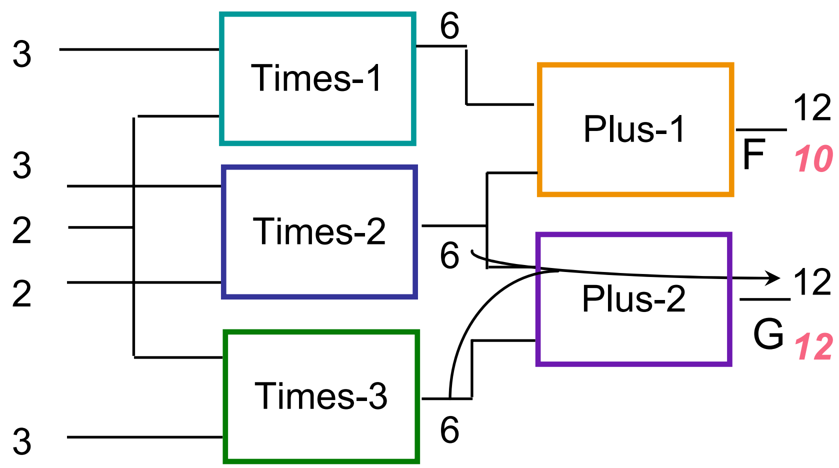
- The bad news
  - Set covering is well known to be exponential

- The (slightly less) bad news
  - The number of components at any level of detail is relatively small

# Corroboration Proves Nothing

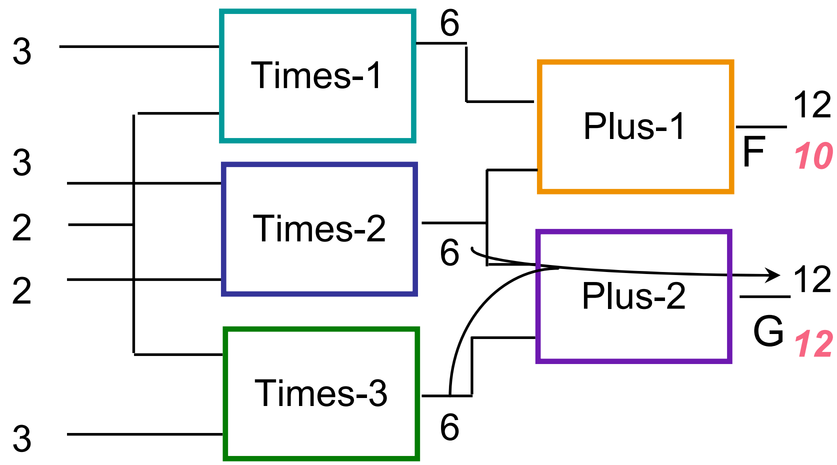
- The basic intuition
  - Involved in discrepancy means suspect
  - Therefore: Involved in corroboration means exonerated
- This is wrong
  - Involved in corroboration only means that you didn’t tickle this problem yet.
    - with these inputs
    - with the specific observations you chose to make so far

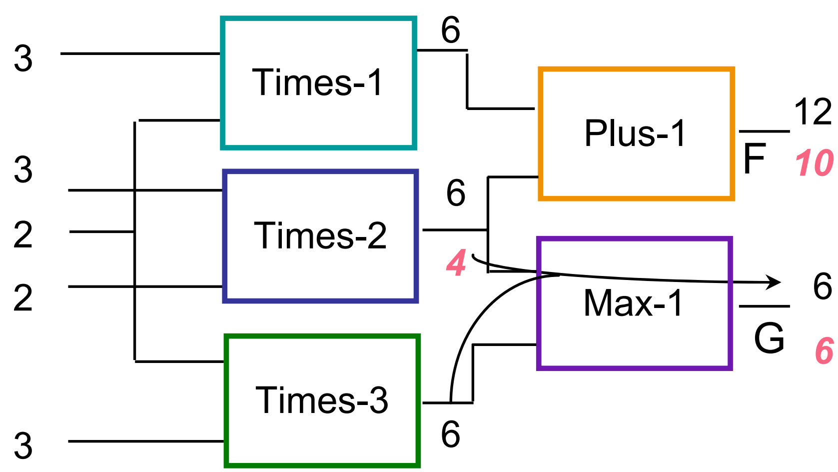
# Corroboration Example and Counter Example




Corroboration would exonerate Plus-2, Time-2, Time-3 since they are upstream from G which has the correct value.
In this case, this is correct.
Corroboration would exonerate Plus-2, Time-2, Time-3 since they are upstream from G which has the correct value.
In this case, this is not correct, since time-2 could be broken and produce 4 as its output.

# Corroboration Example and Counter Example

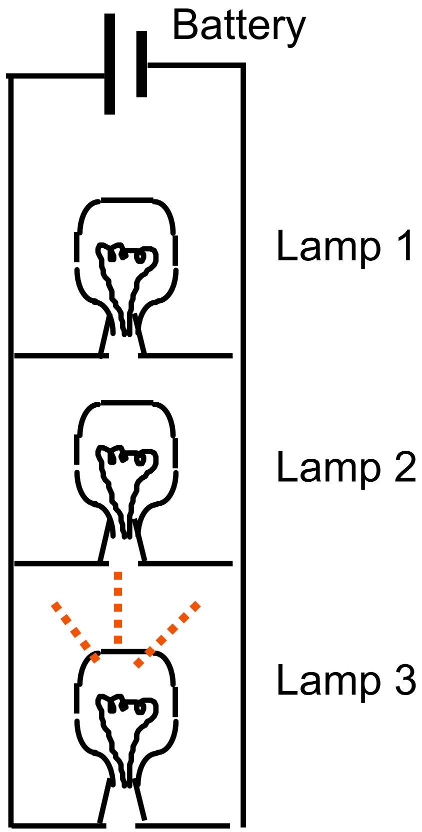



Corroboration would exonerate Plus-2, Time-2, Time-3 since they are upstream from G which has the correct value. In this case, this is correct.
Corroboration would exonerate Plus-2, Time-2, Time-3 since they are upstream from G which has the correct value. In this case, this is not correct, since time-2 could be broken and produce 4 as its output.

# Fault Models

- Good News: what we've seen so far doesn't need them
- Bad News: what we've seen so far can't use them




Conflicts:

B,L1
B,L2
L1,L3
L2,L3

Diagnoses:

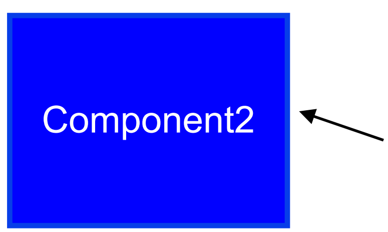
L1,L2

# Fault Models

- Extend the notion of fault model to include multiple behavioral modes:
  – Designed behavior (i.e., the *correct* behavior)
  – Known faulty behaviors
  – Residual behavior (i.e. everything *besides* designed and known faults)
  – Their probabilities
- Start with models of correct behavior
- When conflicts exist, substitute a fault model for some member of the conflict set
- Drive the choice of substitution by failure probabilities
  – best diagnosis is most likely set of behavior modes for the various candidates capable of removing all discrepancies
  – i.e., best first search for conflict free set of behavior modes

# Adding Failure Models

- In addition to modeling the normal behavior of each component, we can provide models of known abnormal behavior
- Each Model can have an associated probability
- A "leak Model" covering unknown failures/compromises covers residual probabilities.
- Diagnostic task becomes, finding most likely set(s) of models (one model for each component) consistent with the observations.
- Search process is best first search with joint probability as the metric

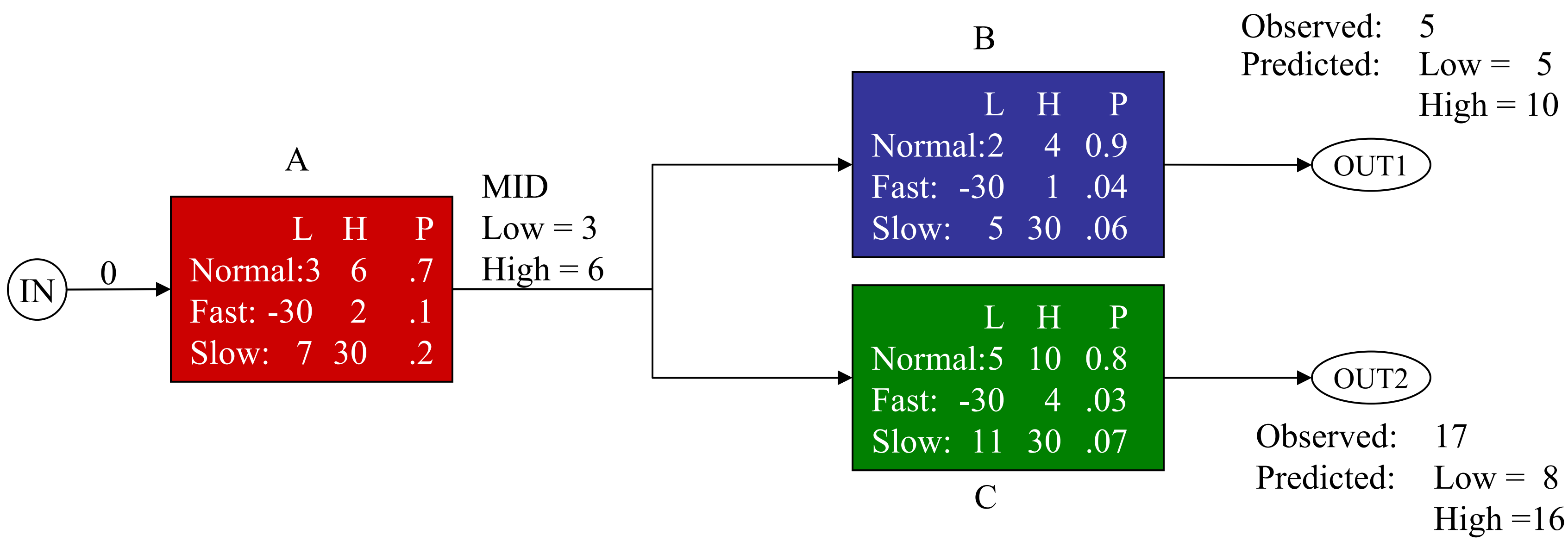



Normal: Delay: 2, 4 Probability 90%

Delayed: Delay 4, +inf Probability 9%

Accelerated: Delay -inf,4 Probability 1%

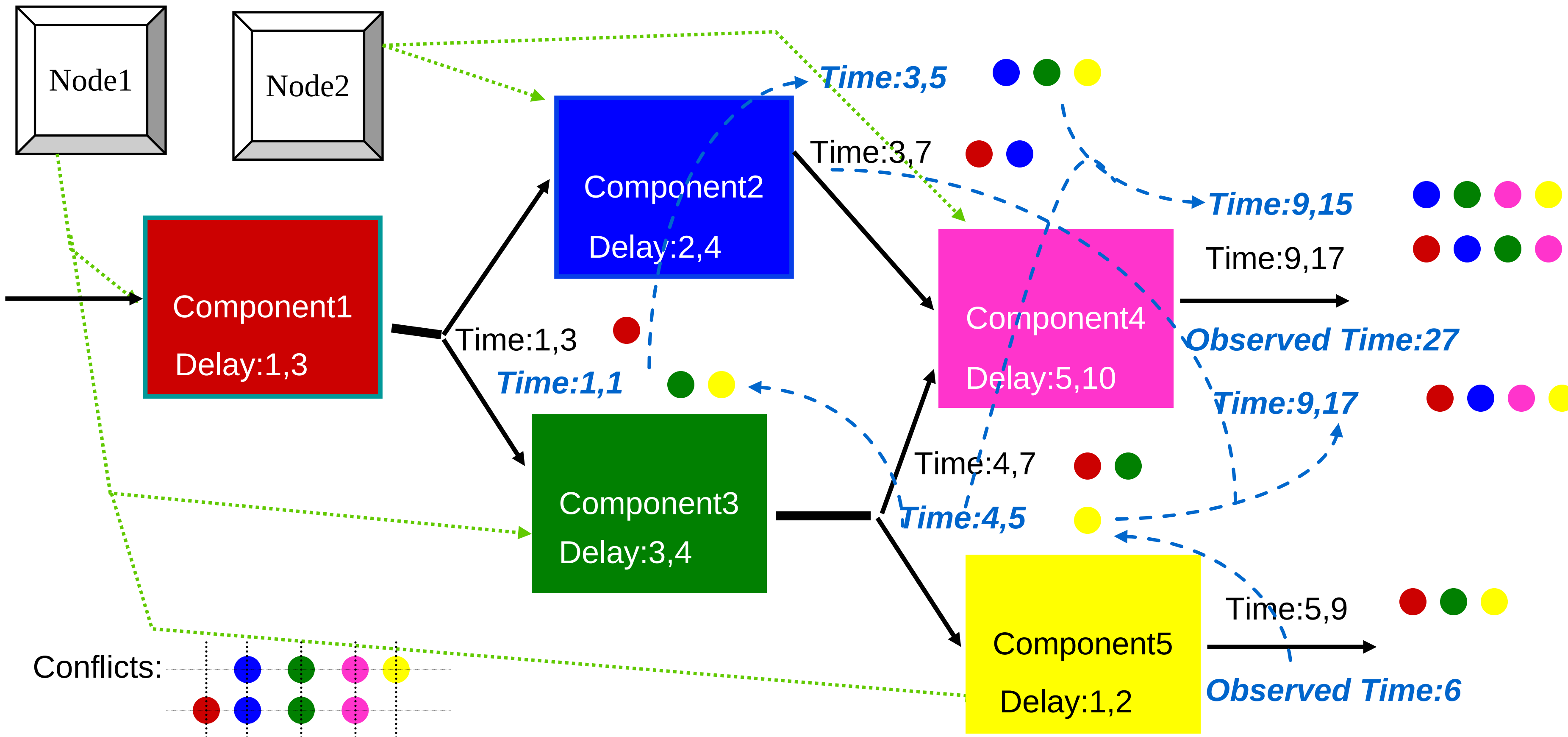
# Applying Failure Models




**Consistent Diagnoses**

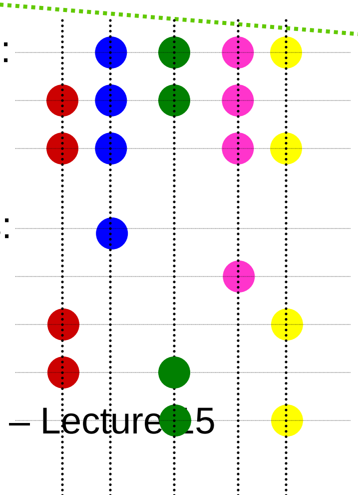
| A | B | C | MID Low | MID High | Prob | Explanation |
|---|---|---|---|---|---|---|
| Normal | Normal | Slow | 3 | 3 | .04410 | C is delayed |
| Slow | Fast | Normal | 7 | 12 | .00640 | A Slow, B Masks runs negative! |
| Fast | Normal | Slow | 1 | 2 | .00630 | A Fast, C Slower |
| Normal | Fast | Slow | 4 | 6 | .00196 | B not too fast, C slow |
| Fast | Slow | Slow | -30 | 0 | .00042 | A Fast, B Masks, C slow |
| Slow | Fast | Fast | 13 | 30 | .00024 | A Slow, B Masks, C not masking fast |

# Computational Models are Coupled through Resource Models




Conflicts:

Diagnoses:

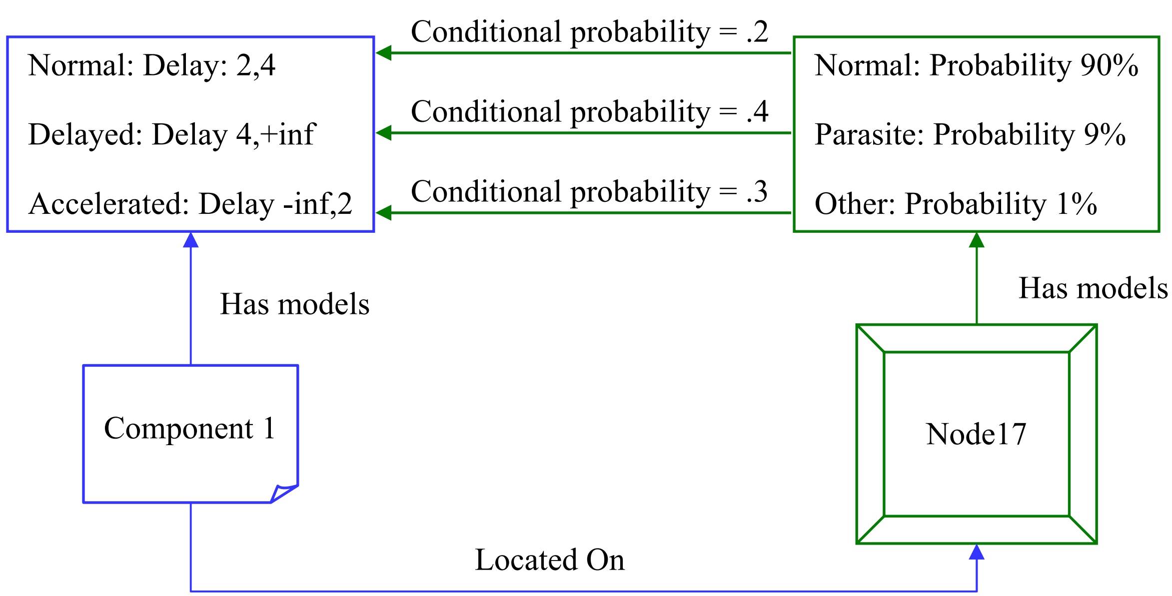


Blue delayed
Violet delayed
Red delayed, Yellow Negative Time
Red delayed, Green Negative Time
Green delayed, Yellow Negative Time

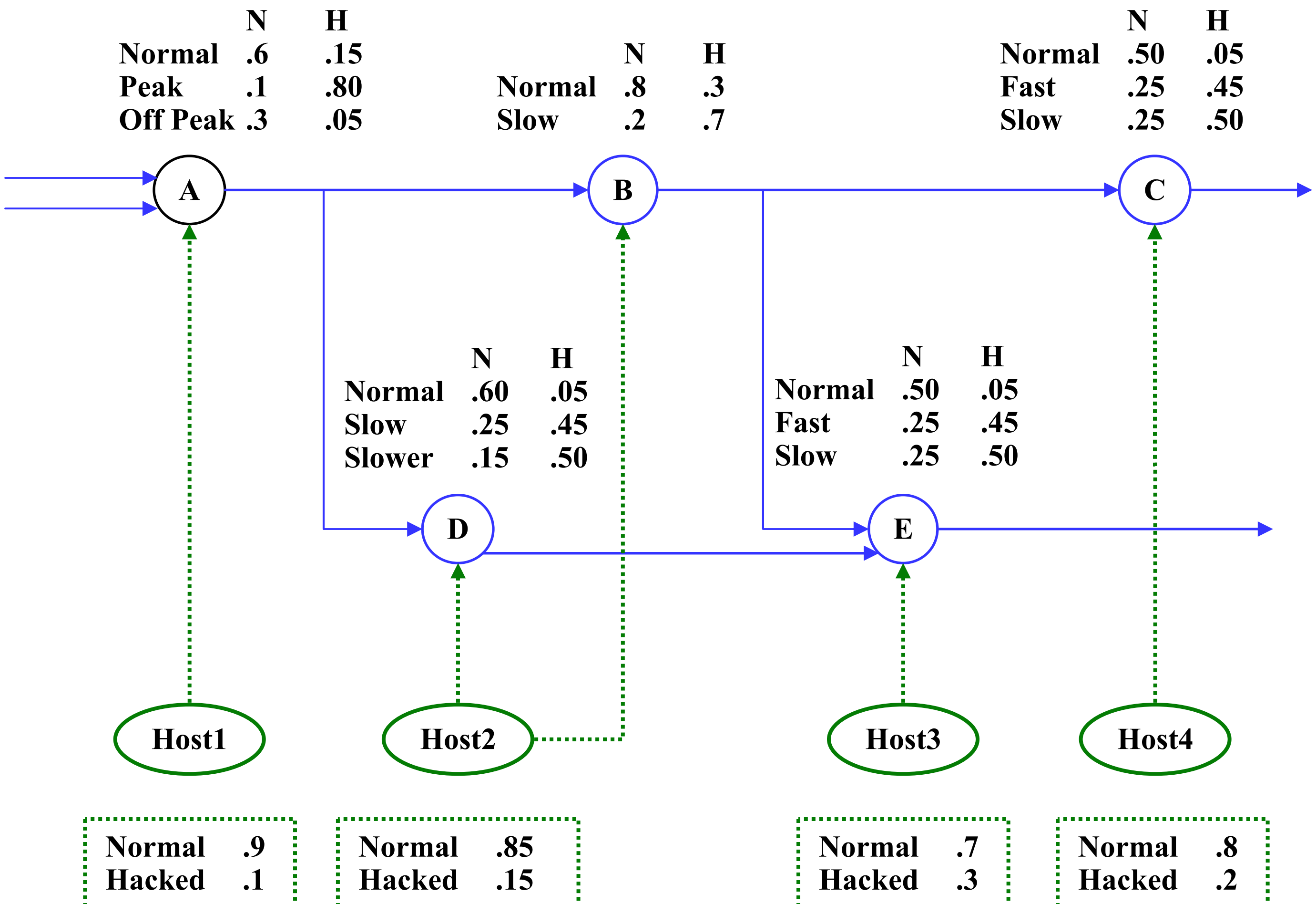
Precluded because physicality requires red green and yellow to all be delayed or all be accelerated

# A Multi-Tiered Bayesian Framework

- The model has levels of detail specifying computations, the underlying resources and the mapping of computations to resources
- Each resource has models of its state of compromise
- The modes of the resource models are linked to the modes of the computational models by conditional probabilities
- The Model forms a Bayesian Network

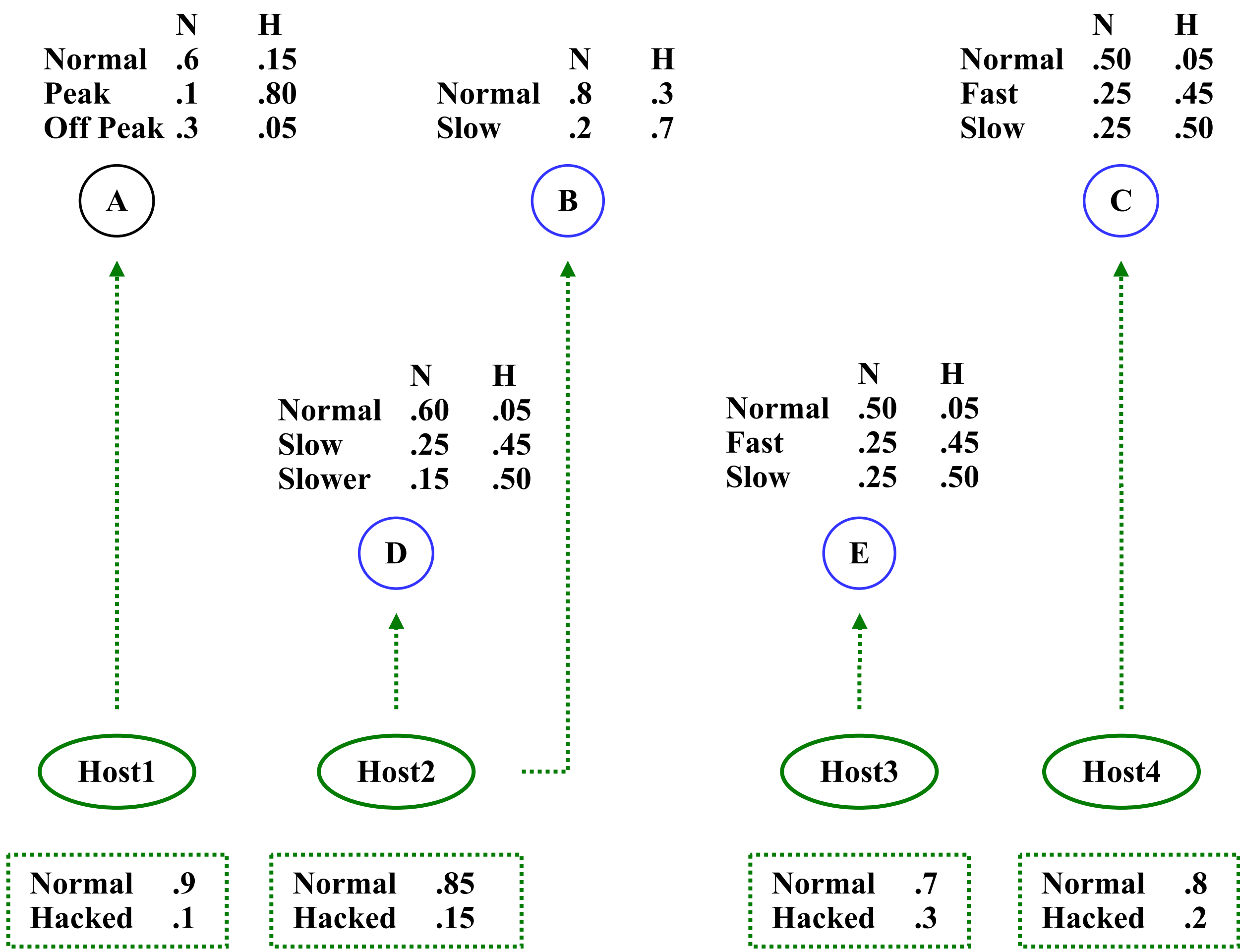


# An Example System Description



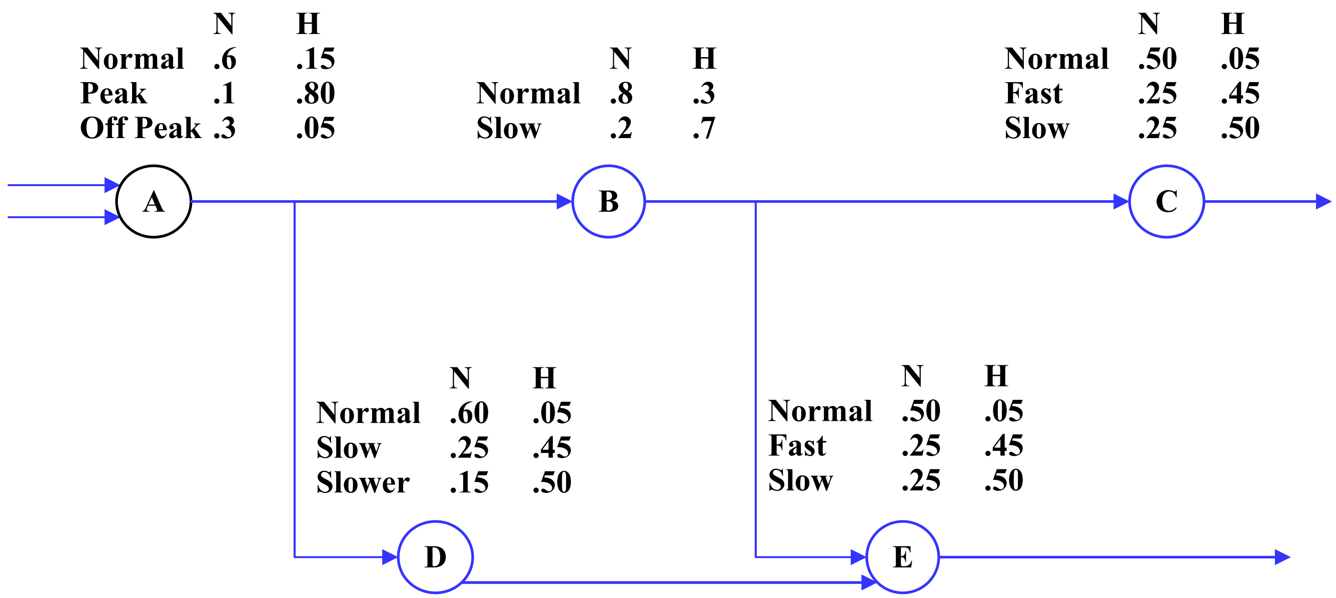
# System Description as a Bayesian Network

- The Model can be viewed as a Two-Tiered Bayesian Network
  - Resources with modes
  - Computations with modes
  - Conditional probabilities linking the modes



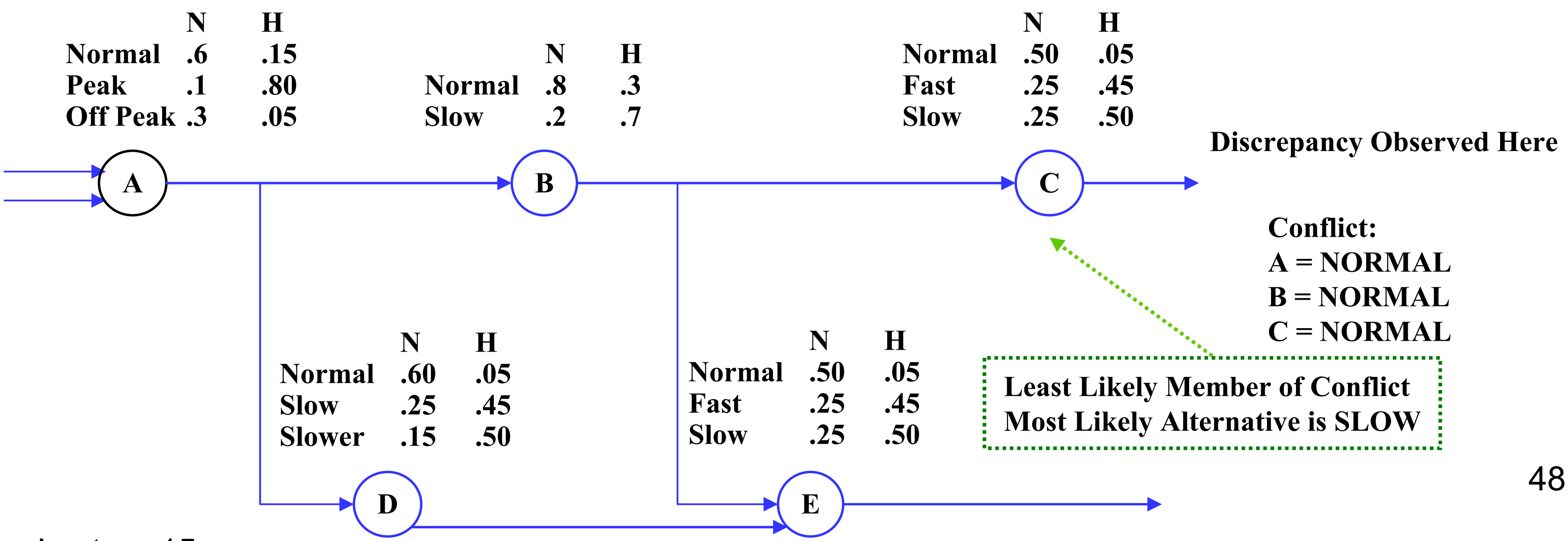
# System Description as a MBT Model

- The Model can also be viewed as a MBT model with multiple models per device
  - Each model has behavioral description
- Except the models have <u>conditional</u> probabilities



# Integrating MBT and Bayesian Reasoning

- Start with each behavioral model in the "normal" state
- Repeat: Check for Consistency of the current model
- If inconsistent,
  - Add a new node to the Bayesian network
    - This node represents the logical-and of the nodes in the conflict.
    - It's truth-value is pinned at FALSE.
  - Prune out all possible solutions which are a super-set of the conflict set.
  - Pick another set of models from the remaining solutions
- If consistent, Add to the set of possible diagnoses
- Continue until all inconsistent sets of models are found
- Solve the Bayesian network

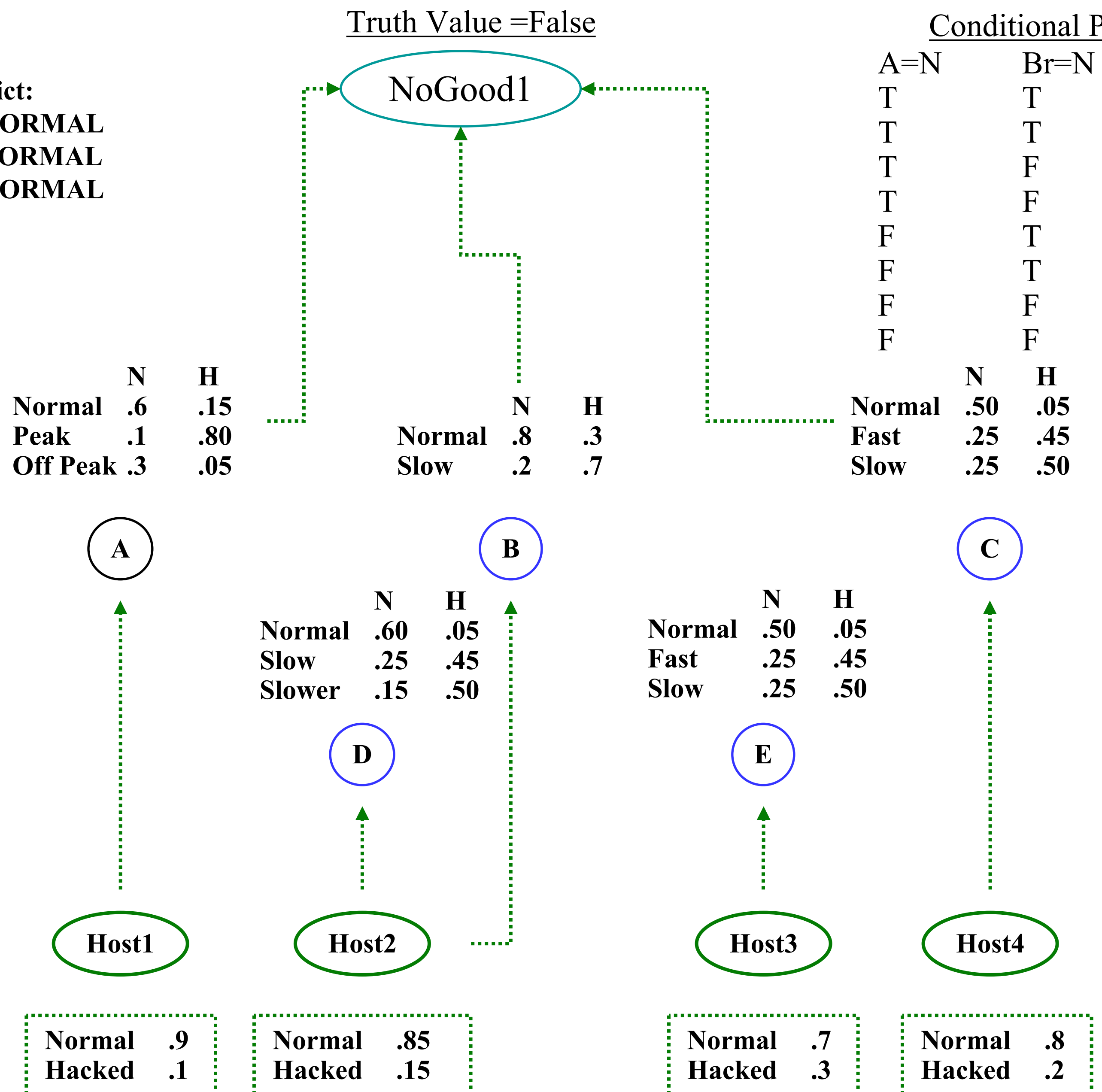


# Adding the Conflict to the Bayesian Network

**Conflict:**
**A = NORMAL**
**B = NORMAL**
**C = NORMAL**

Conditional Probability Table

| A=N | Br=N | C=N | T | F |
|---|---|---|---|---|
| T | T | T | 1 | 0 |
| T | T | F | 0 | 1 |
| T | F | T | 0 | 1 |
| T | F | F | 0 | 1 |
| F | T | T | 0 | 1 |
| F | T | F | 0 | 1 |
| F | F | T | 0 | 1 |
| F | F | F | 0 | 1 |

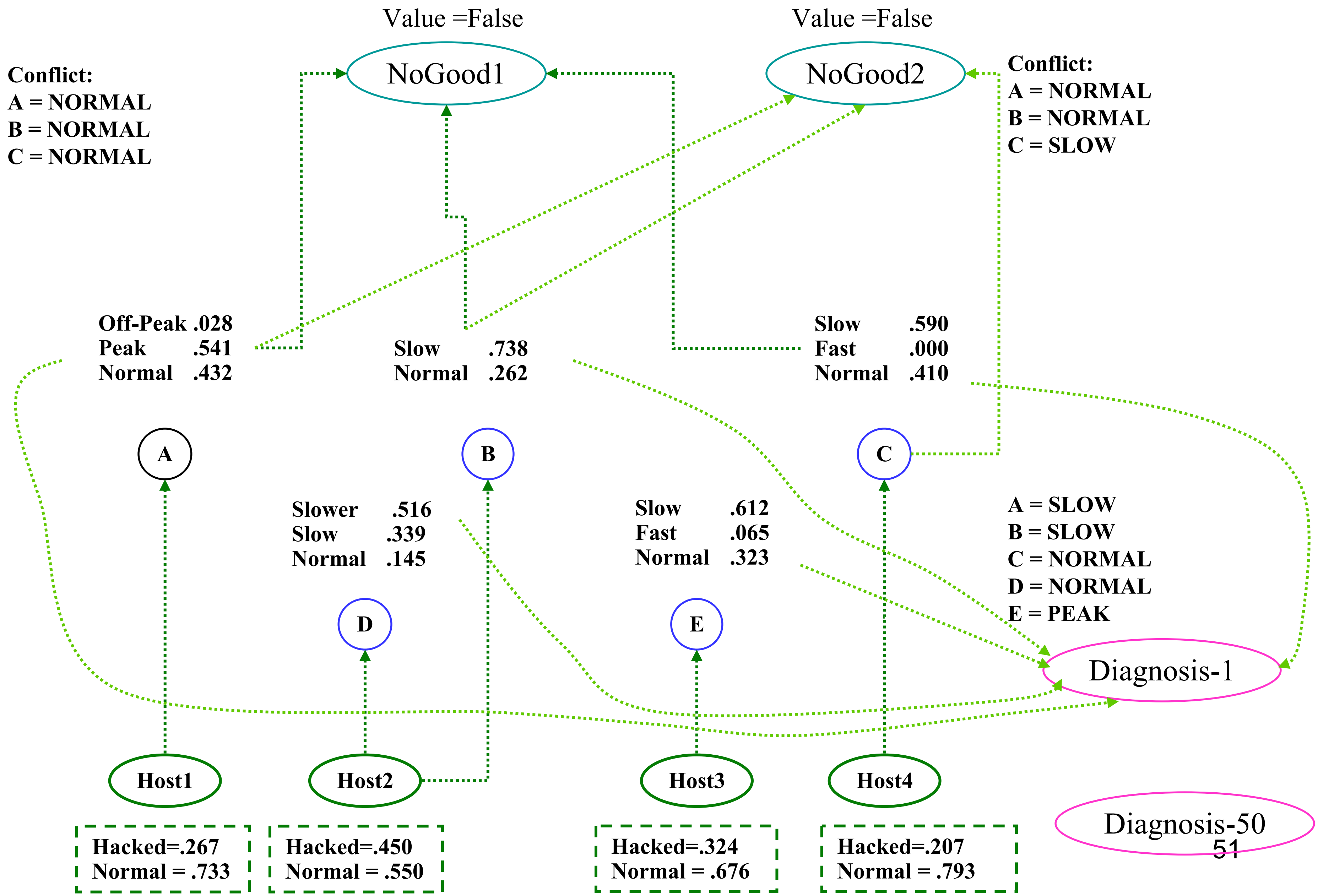


# Integrating MBT and Bayesian Reasoning (2)

- Repeat Finding all conflicts and adding them to the Bayesian Net.
- Solve the network again.
  - The posterior probabilities of the underlying resource models tell you how likely each model is.
  - These probabilities should inform the trust-model and lead to Updated Priors and guide resource selection.
  - The Posterior probabilities of the computational models tell you how likely each model is. This should guide recovery.

- All remaining non-conflicting combination of models are possible diagnoses
  - Create a conjunction node for each possible diagnosis and add the new node to the Bayesian Network (call this a diagnosis node)
- Finding most likely diagnoses:
  - Bias selection of next component model by current model probabilities

# The Final Bayesian Network

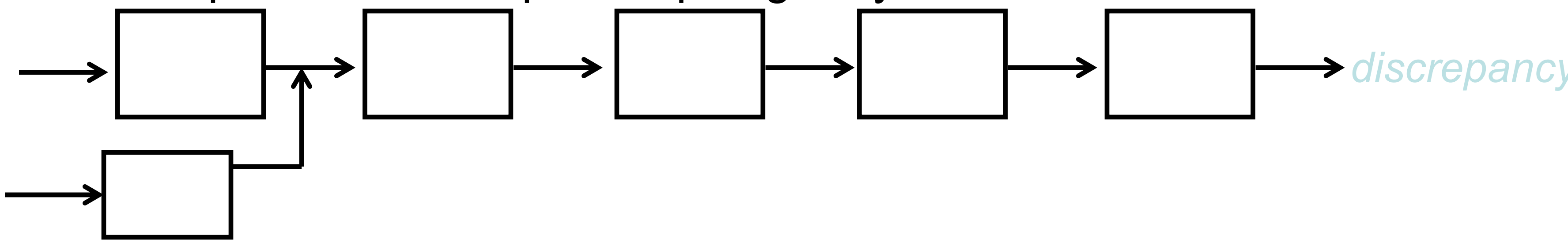


# Three Fundamental Problems

- Hypothesis Generation
  - Given a symptom, which components could have produced it?
- Hypothesis Testing
  - Which components could have failed to account for all observations?
- Hypothesis Discrimination
  - What additional information should we acquire to distinguish among the remaining candidates?

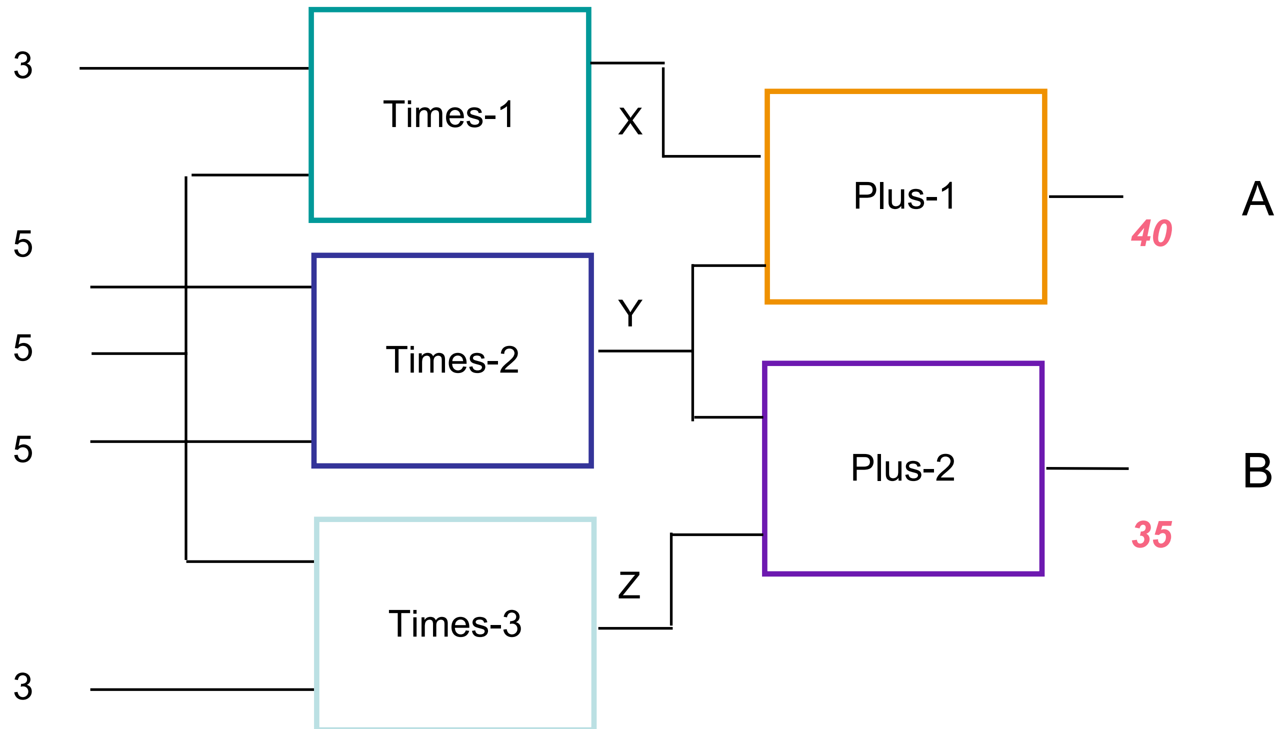
# Probing and Testing

- Purely structural
  - Follow discrepancies upstream (guided probe)
  - Split candidate space topologically




- Add behavioral information:
  - Split topologically: G&T on the sub-problem
  - Predict consequences of candidate malfunction; probe where it is most informative.

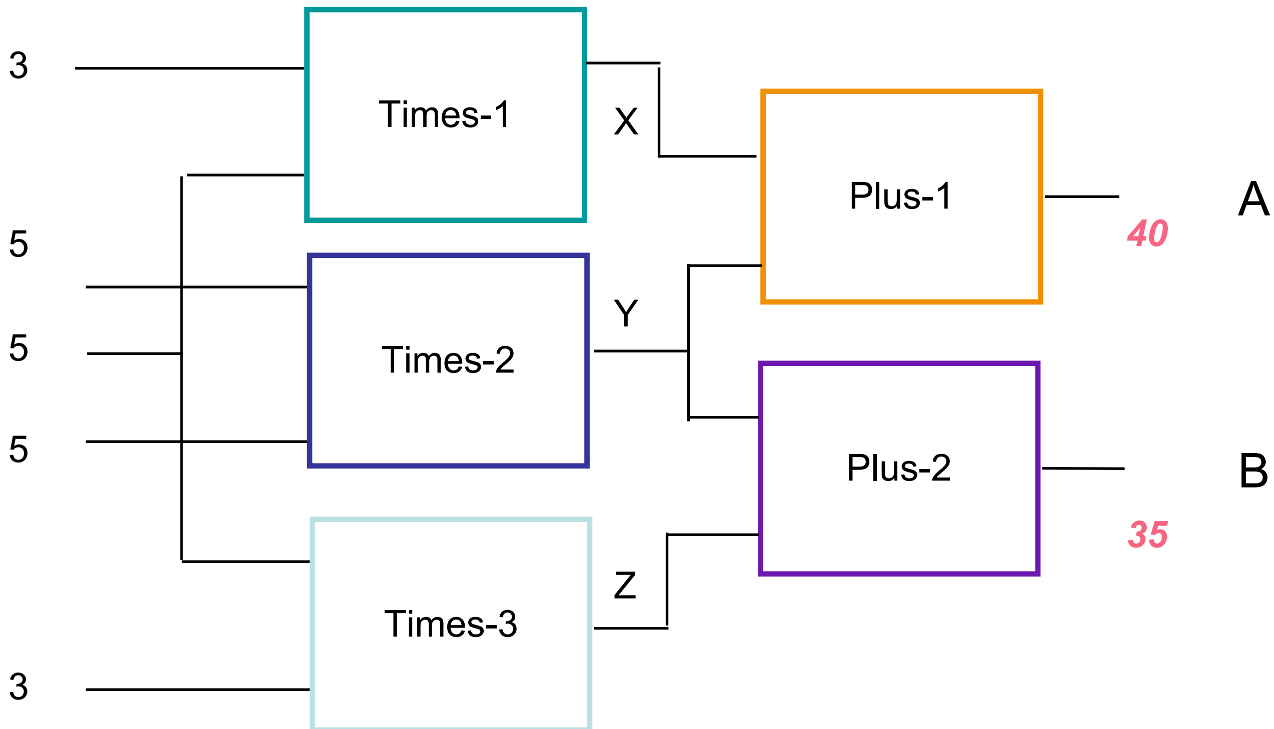
# Informative Probes




T3 Bad   P2 Bad

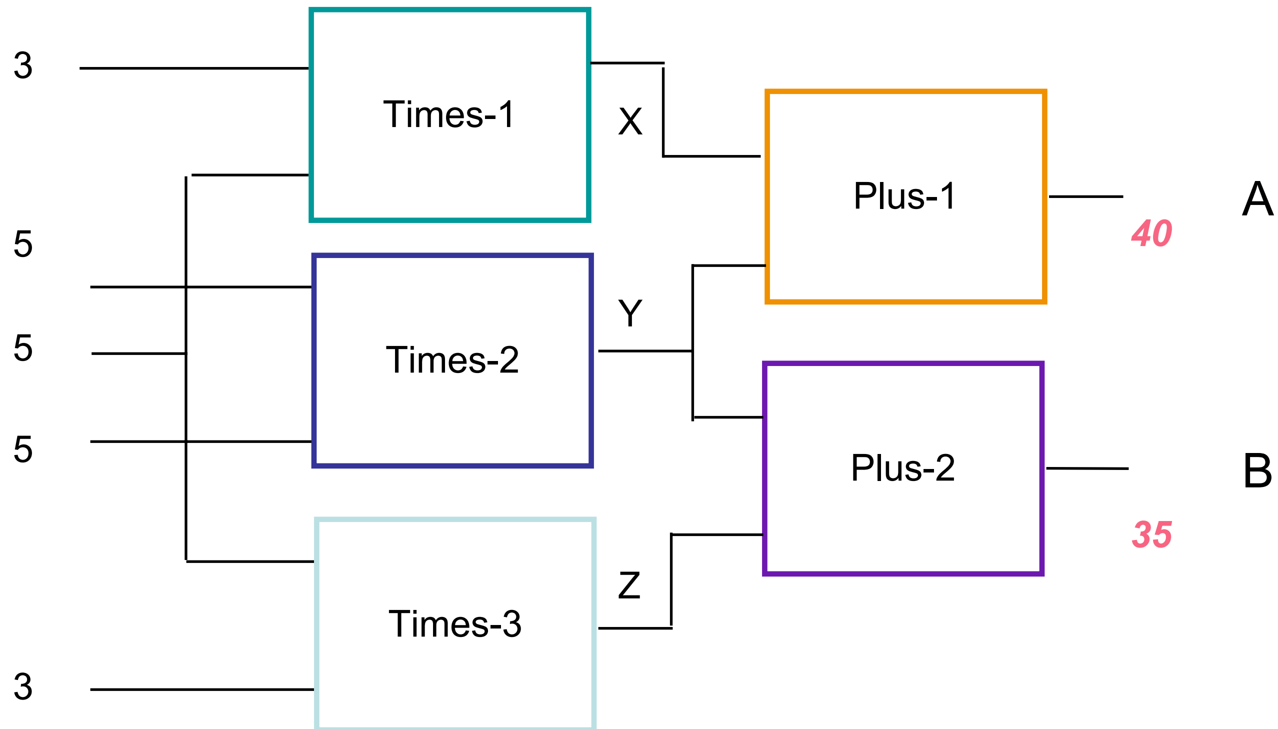
Probe at Y
Probe at Z

# Informative Probes




| | T3 Bad | P2 Bad |
|---|---|---|
| Probe at Y | 25 | 25 |
| Probe at Z | | |

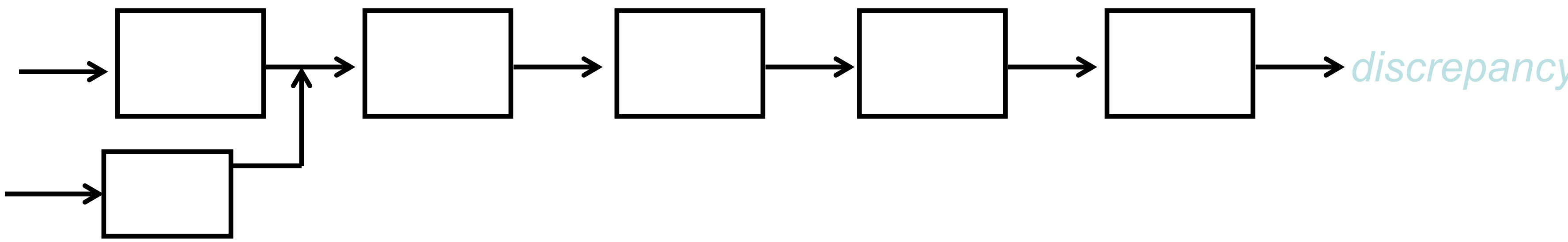
# Informative Probes




| | T3 Bad | P2 Bad |
|---|---|---|
| Probe at Y | 25 | 25 |
| Probe at Z | 10 | 15 |

# Probing and Testing

- Purely structural
  - Follow discrepancies upstream (guided probe)
  - Split candidate space topologically

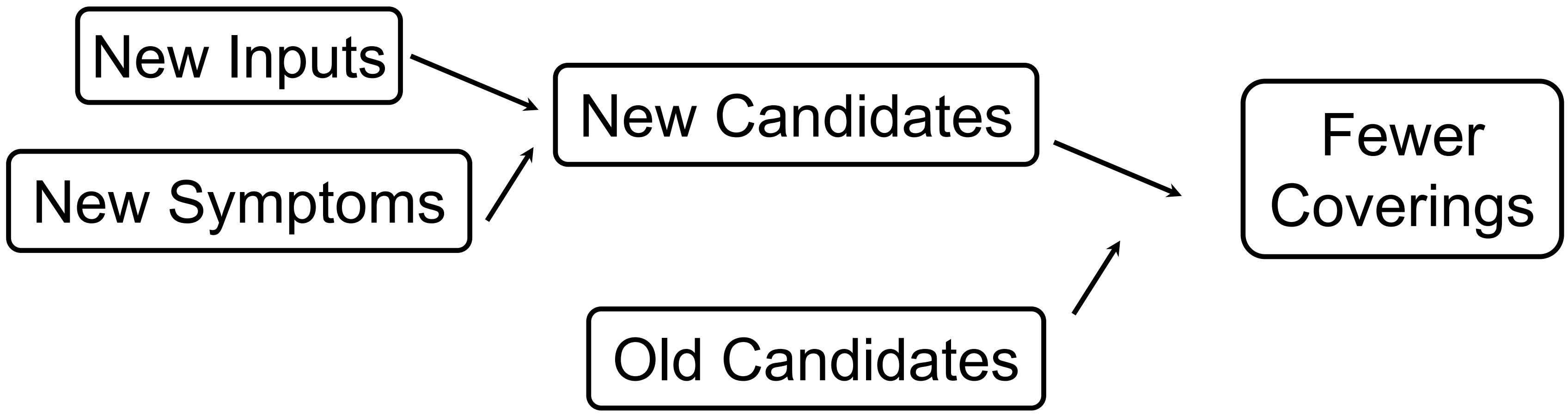



- Add behavioral information:
  - Split topologically: G&T on the sub-problem
  - Predict consequences of candidate malfunction; probe where it is most informative.

- Add failure probabilities
  - Cost-benefit calculation using maximum entropy methods

**Assumption: Computation is cheap compared to probing (think of chips)**

# Testing

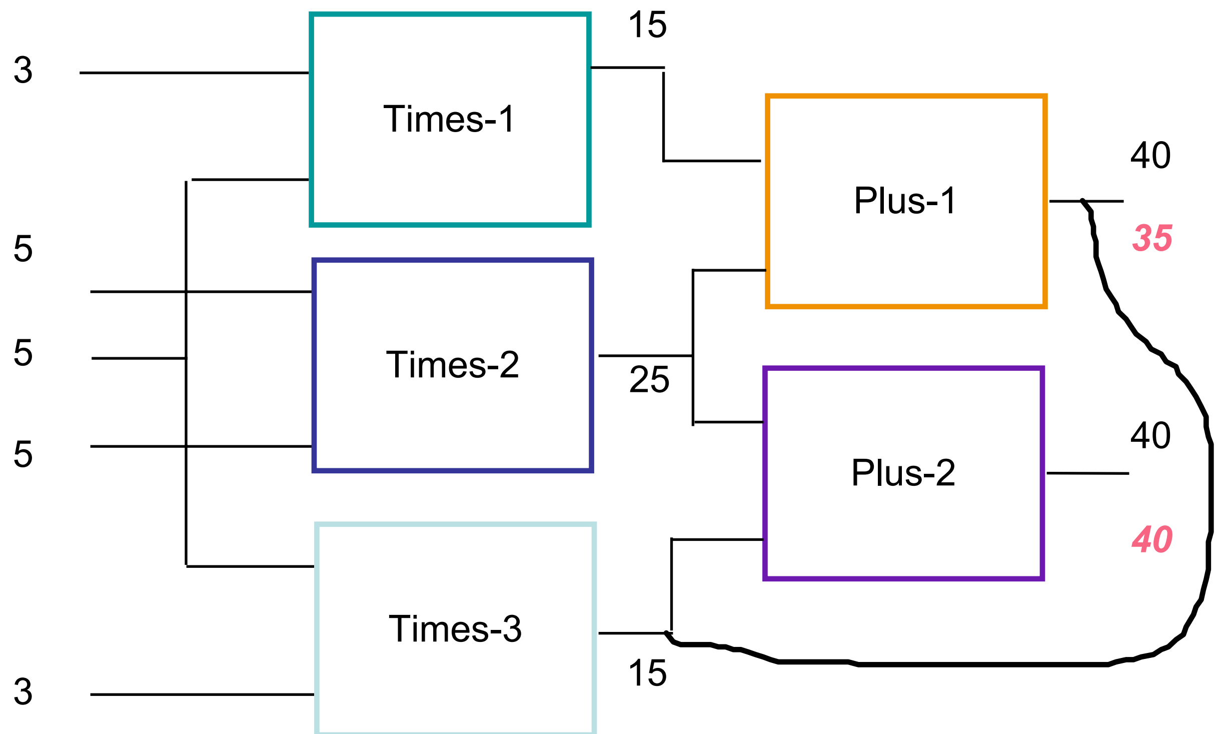



- General problem is very hard
- Basic insight: don't use members of candidate sets to route signals (i.e. use only parts believed to be good)

# Difficulties

- Model based reasoning is only as good as the model
- Tension between completeness of description and tractability of reasoning.
- Scaling: size alone isn't the issue (but it is an issue)
- Complex behavior is an issue
  - VCR, ALU, Pentium, PowerPC, Disk Controller
  - This requires new vocabulary, new abstractions
  - Temporally coarse descriptions are often important
    - Memory and state are hard to model
    - Temporally coarse representations can hide the state usefully

# The Model Isn't How It Is



# The Model Isn't How It Is

- Because it shouldn't be that way
  - bridge faults, assembly error

- Because of unexpected pathways of interaction
  - eg heat, radiation

- In practice, by our choices
  - deciding not to represent each individual wire segment

- In principle: it's impossible

# Complexity vs Completeness

- Any simplifying assumption risks incompleteness

- Make too few assumptions and
  – diagnosis becomes indiscriminate
  – drown in complexity, ambiguity

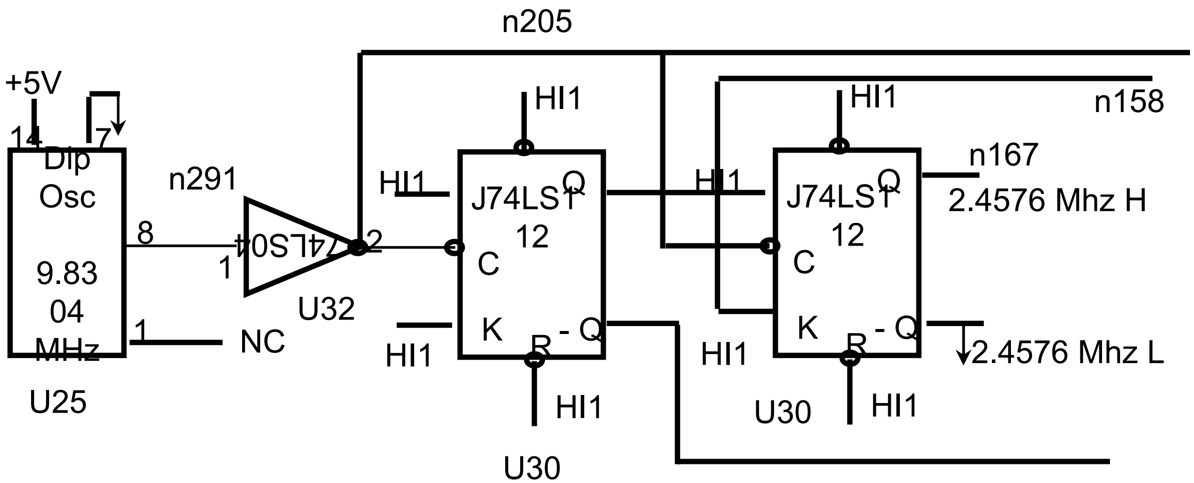
# Model Selection and Formulation Is a Key Problem

- There are no assumption-free representations
  - perhaps we can use more than one
- Completeness and complexity conflict
  - we’ll need to choose judiciously
- Basic question: *whence the model?*
  *How do we know how to think about the device?*

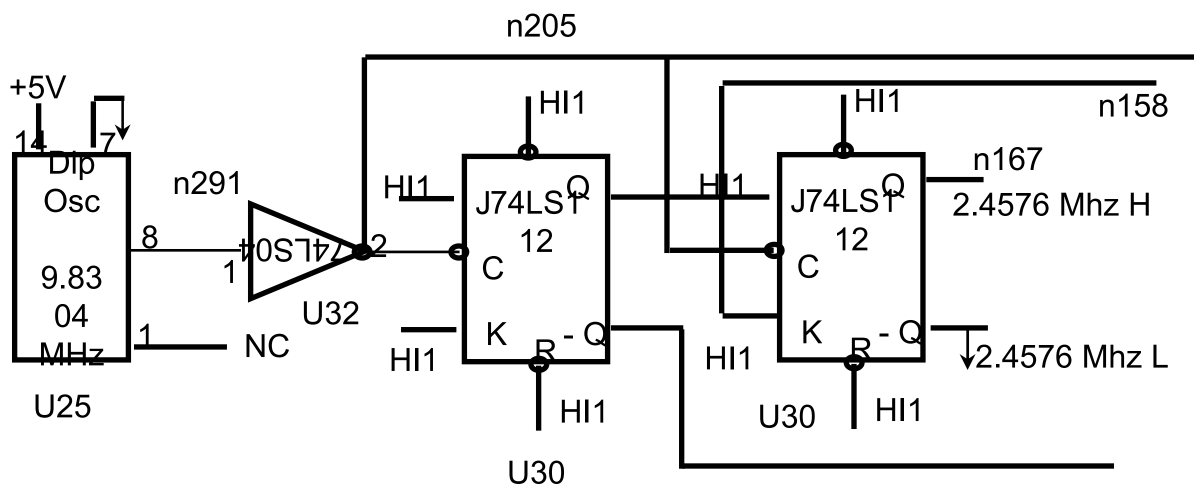
# Another Problem: Complex Behavior

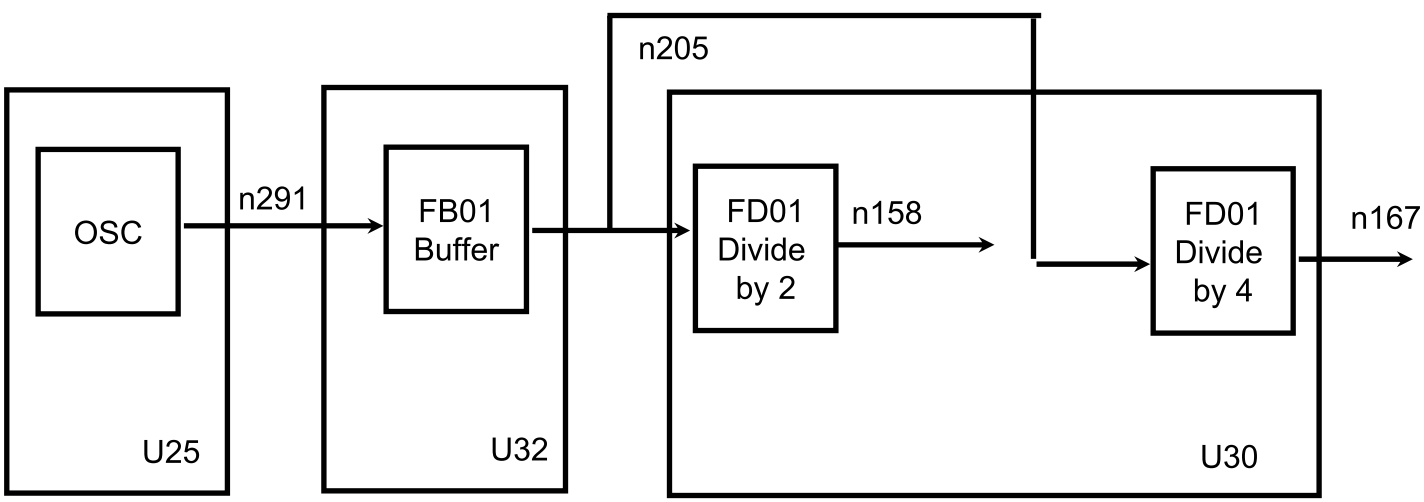
- An engineer plugs in a broken circuit board, makes a half dozen simple probes with an oscilloscope, and after ten minutes ends up swapping a chip, which fixes the problem.
- A model-based troubleshooting program spends a day simulating the expected behavior of the same misbehaving board, and requests that a logic analyzer be used to capture a certain subset of the signals. After some hours of computation, it concludes that any of the 40 chips or 400 wires on the board could be responsible for the misbehavior.
- Why?

# The Two Different Approaches to MBT



# The Two Different Approaches to MBT




If n167 is “flat” then U25, U32 and U30 form a conflict. But Oscillators tend to fail more frequently, so U25 is more likely to be broken. A probe of n291 is advised.

# More (detail) is Worse

- The naïve approach suggests a detailed, step by step simulation of the device as the first phase of the diagnosis.
- For a reasonable circuit with internal states, all interesting behavior exists over the time span of many thousands to millions of clock cycles.
- The naïve approach fails to capture the right functional abstractions
  - Devices: Central controller
  - Behavior: Frequency
    - Changing
    - Stable

# The Problems to be Faced

- Models are incomplete.
- Observations are costly.
- Observations are incomplete and imprecise.
- Prediction is costly.
- Prediction is incomplete.

# How to Address these Problems

- Choose the representation of primitive elements and connections so as to sacrifice completeness for efficiency.
  - Treat physically separate components with indistinguishable failure modes as one component.
  - Treat devices whose failure requires the same repair as one device.
  - Don't represent very unlikely failure modes
- Describe signals in a way which is easy to observe.
- Represent the likelihood of failure modes.
- Use temporally abstract description of signals.
- Use multiple levels of behavioral abstraction.

# Principles of Modeling

- Components in the *physical representation* should correspond to the possible repairs.
- Components in the *functional representation* should facilitate behavioral abstraction.

# Principles of Modeling

- Components' behavioral representation should employ features that are easy to observe.
- A temporally coarse description is better than no description.
- A sequential circuit should be encapsulated into a single component whose behavior can be described in a temporally coarse manner.
- Represent a failure mode if it has a high likelihood.
- Represent a failure mode if the misbehavior is drastically simpler than the normal behavior

# Conclusions

- General purpose paradigm (with variations)
- Largely domain independent
- Successfully employed in practice
- Major research issues are in modeling, not reasoning methods
  - complex behavior
  - model selection
  - model formulation