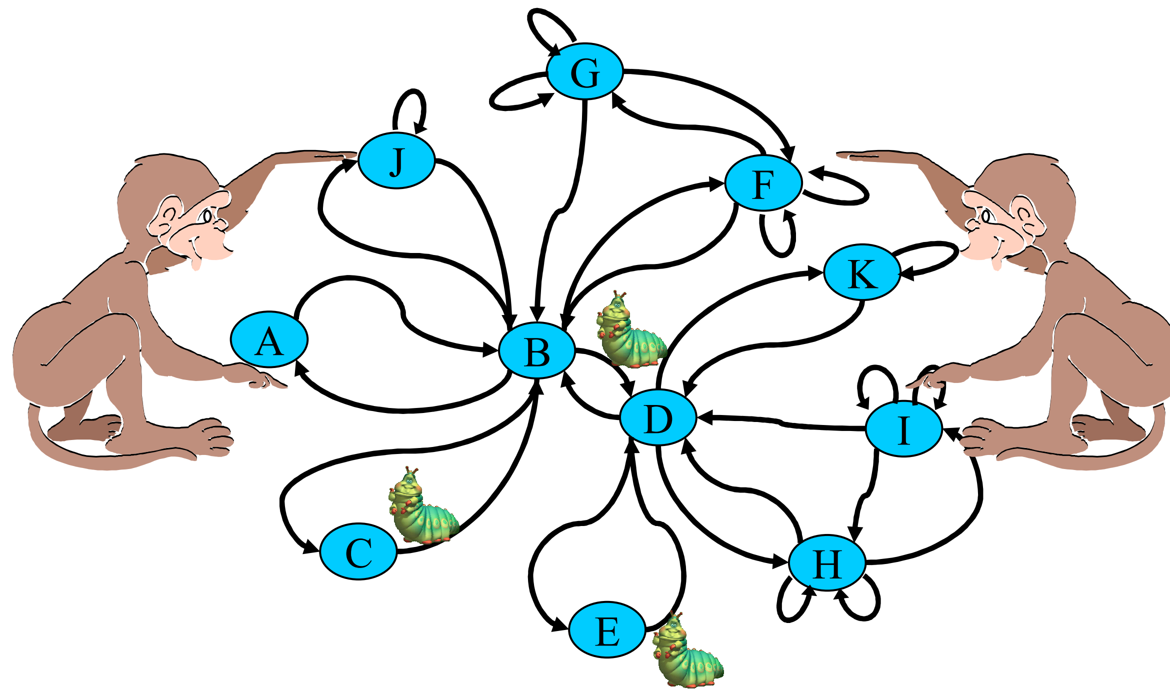
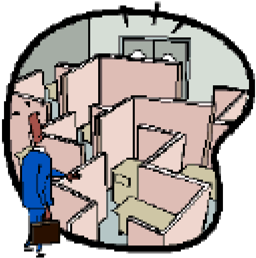
Model-Based Testing: Introduction

(Adapted from Harry Robinson's slides)



What are the Problems of Software Testing?

- Time is limited (time-to-market)



- Applications are complex

protocol: a special kind of software for communication systems
 - usually standardized (by, e.g., ISO, ITU, or other consortia of companies)
 - relatively easy to be formalized
 - relatively small state space (might even be finite)

- Requirements are fluid

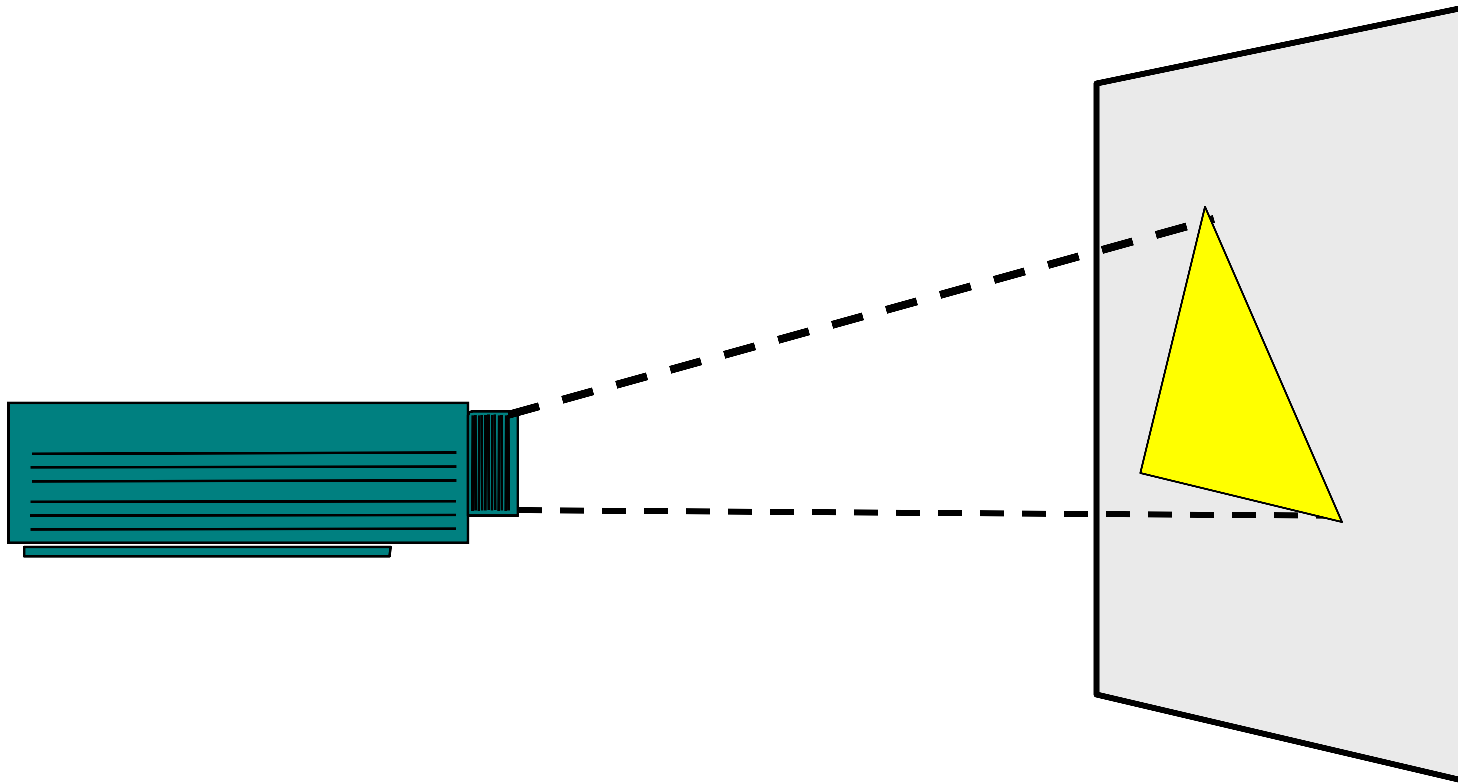
protocol testing:
 - relatively mature and well disciplined (more than 50 years of research)

Scripted Test Automation

```
WSetWndPosSiz(CurrentWindow, 7, 3, 292, 348)
WMenuSelect("&Settings\&Analog")
Sleep(2.193)
WMenuSelect("&Settings\&Digital")
Sleep(2.343)
Play "{DblClick 130, 188, Left}"
WResWnd(CurrentWindow)
Sleep(2.13)
Play "{Click 28, 36, Left}"
Play "{Click 142, 38, Left}"
Play "{DblClick 287, 16, Left}"
```

- Unchanging
- Chiseled in stone
- Usually undecipherable

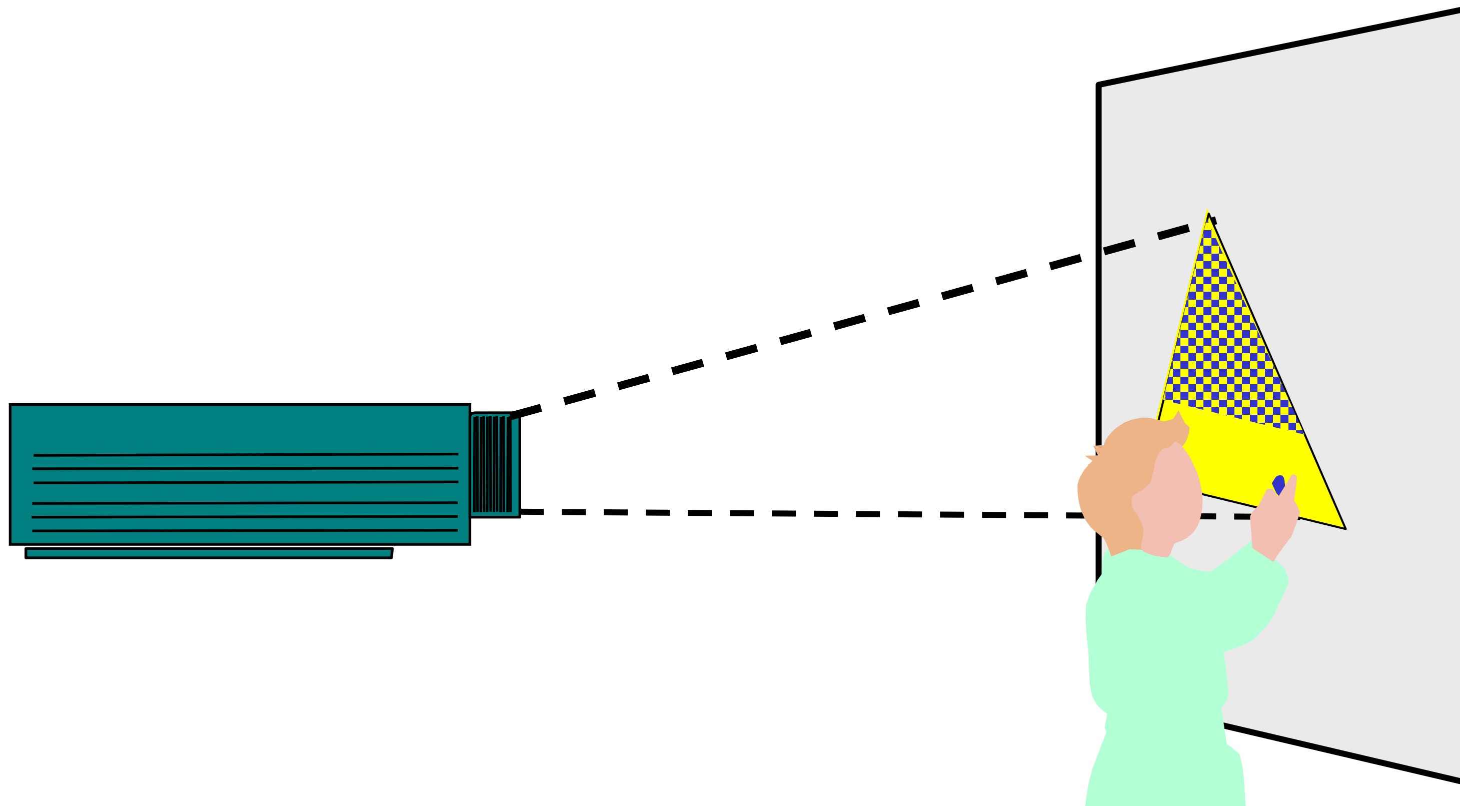
Traditional Software Development



Imagine this projector is the software under test,

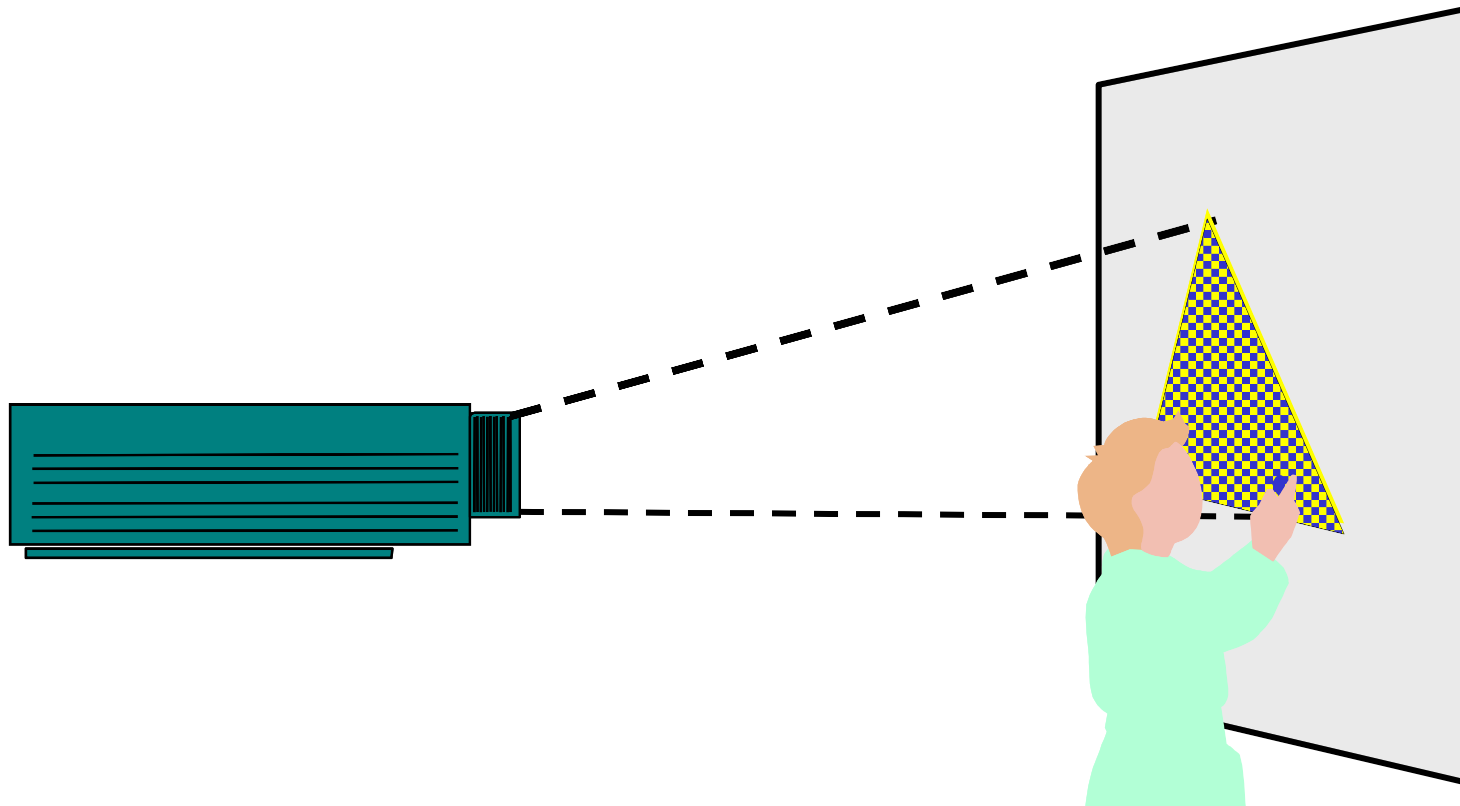
and the triangle is the behavior exposed to you

Traditional Automated Testing



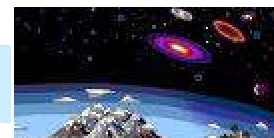
Typically, testers automate by creating static scripts.

Traditional Automated Testing

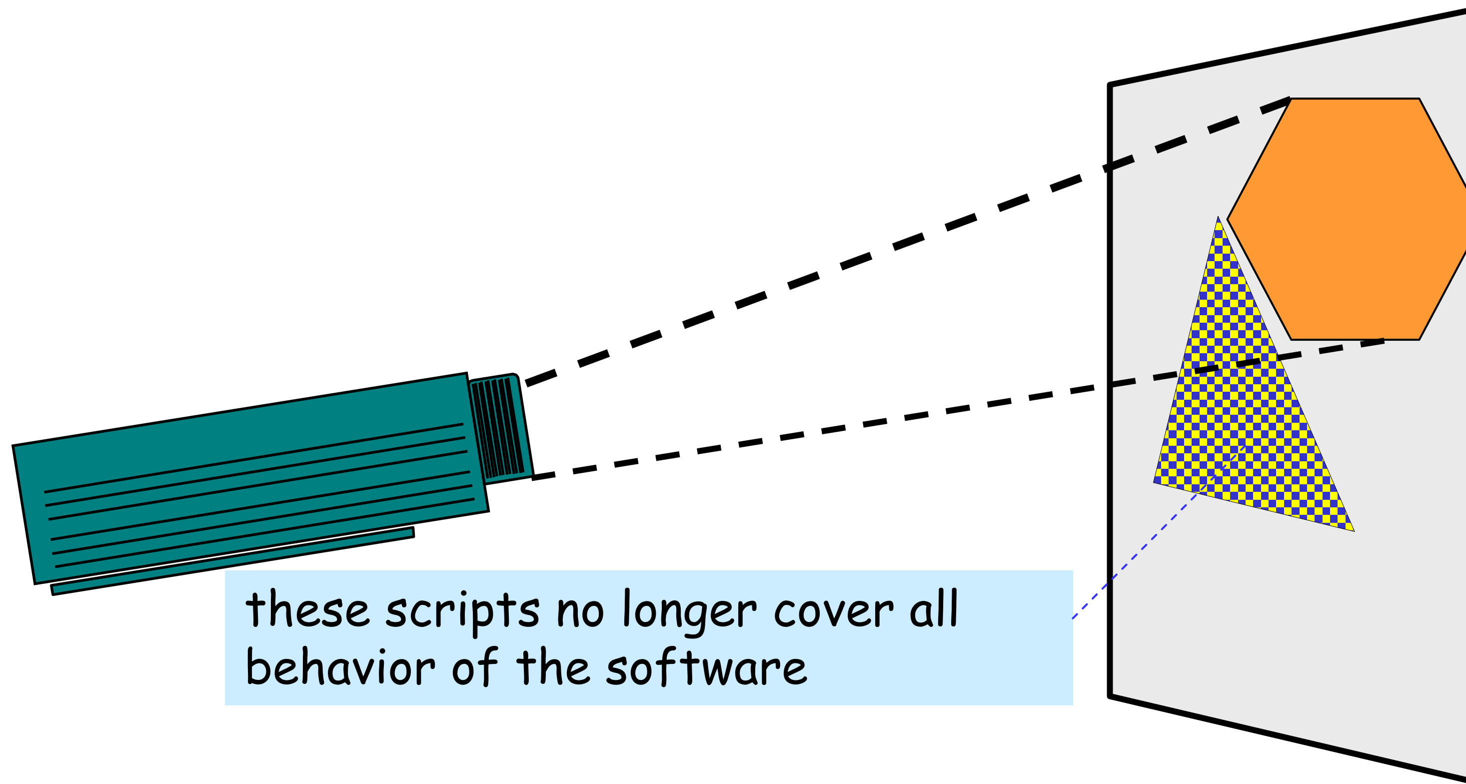


Given enough time, these scripts will cover the behavior.

may be up to thousands of years ...



Traditional Automated Testing

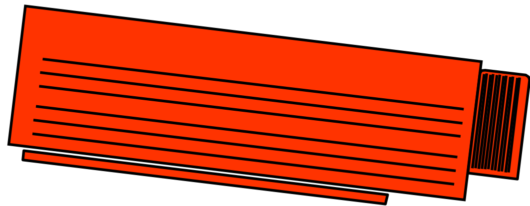


But what happens when the software's behavior changes?

(due to, e.g., requirement change or software maintenance)

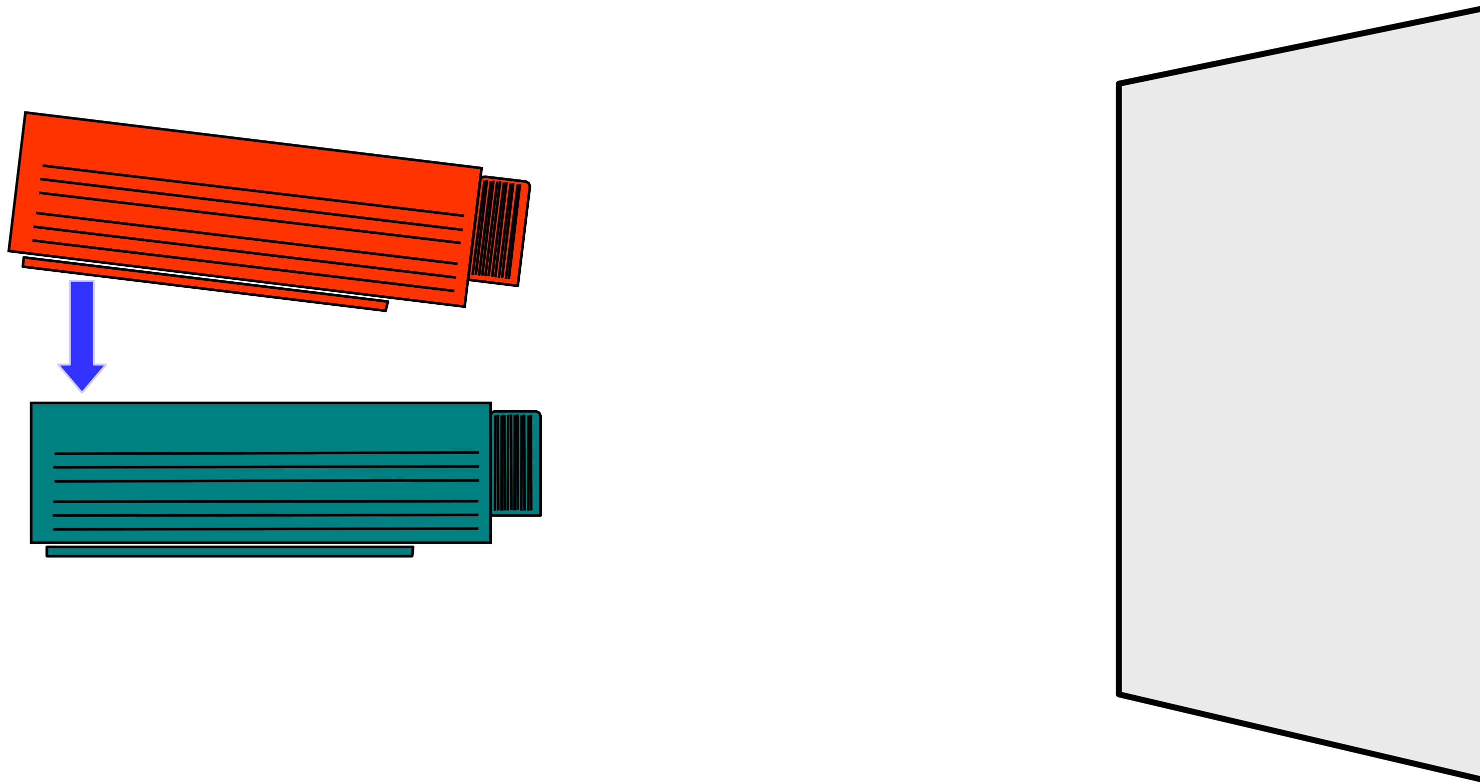
… a remedy…

Model-Based Development



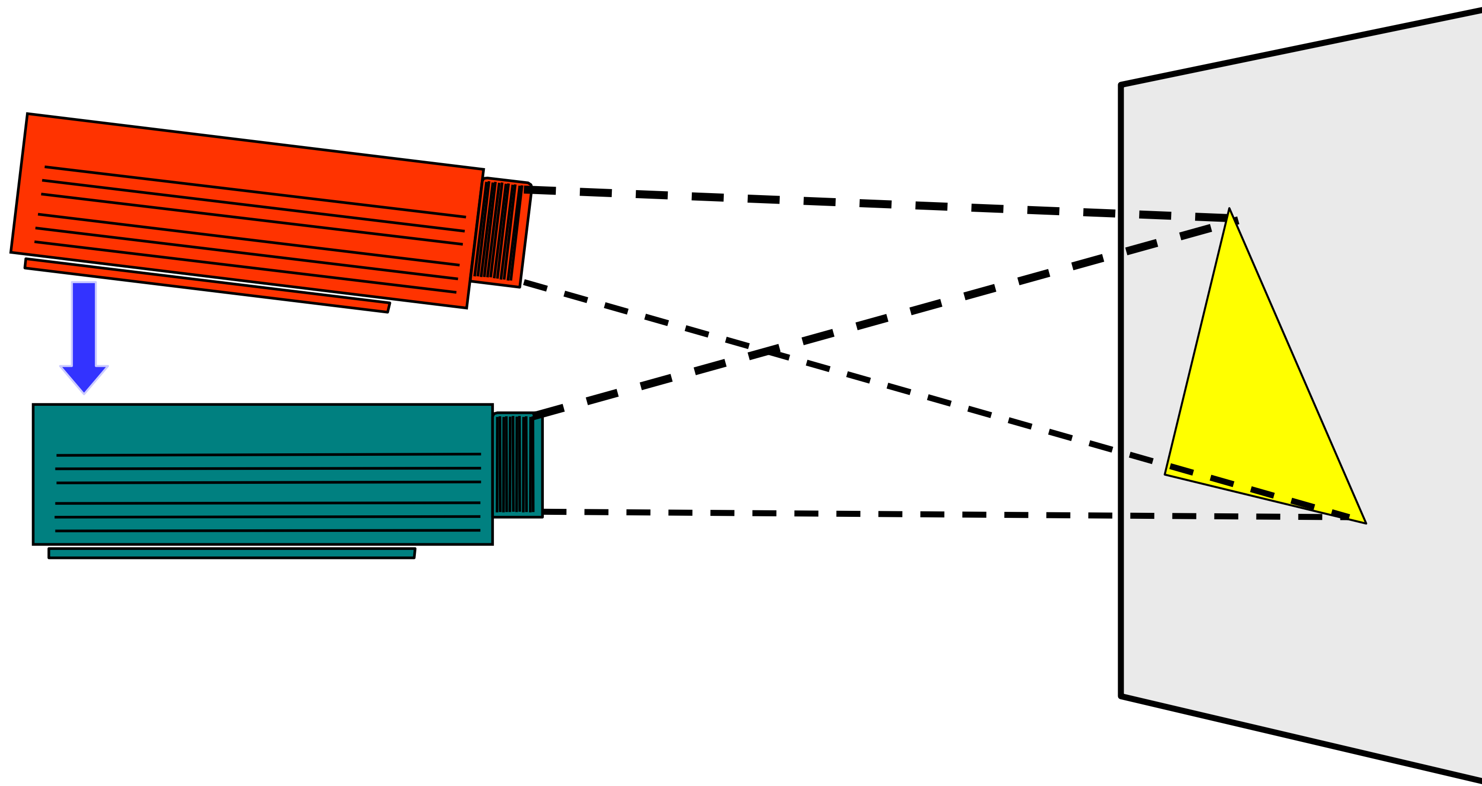
Now, imagine initially you build a model (the upper projecter)

Model-Based Development



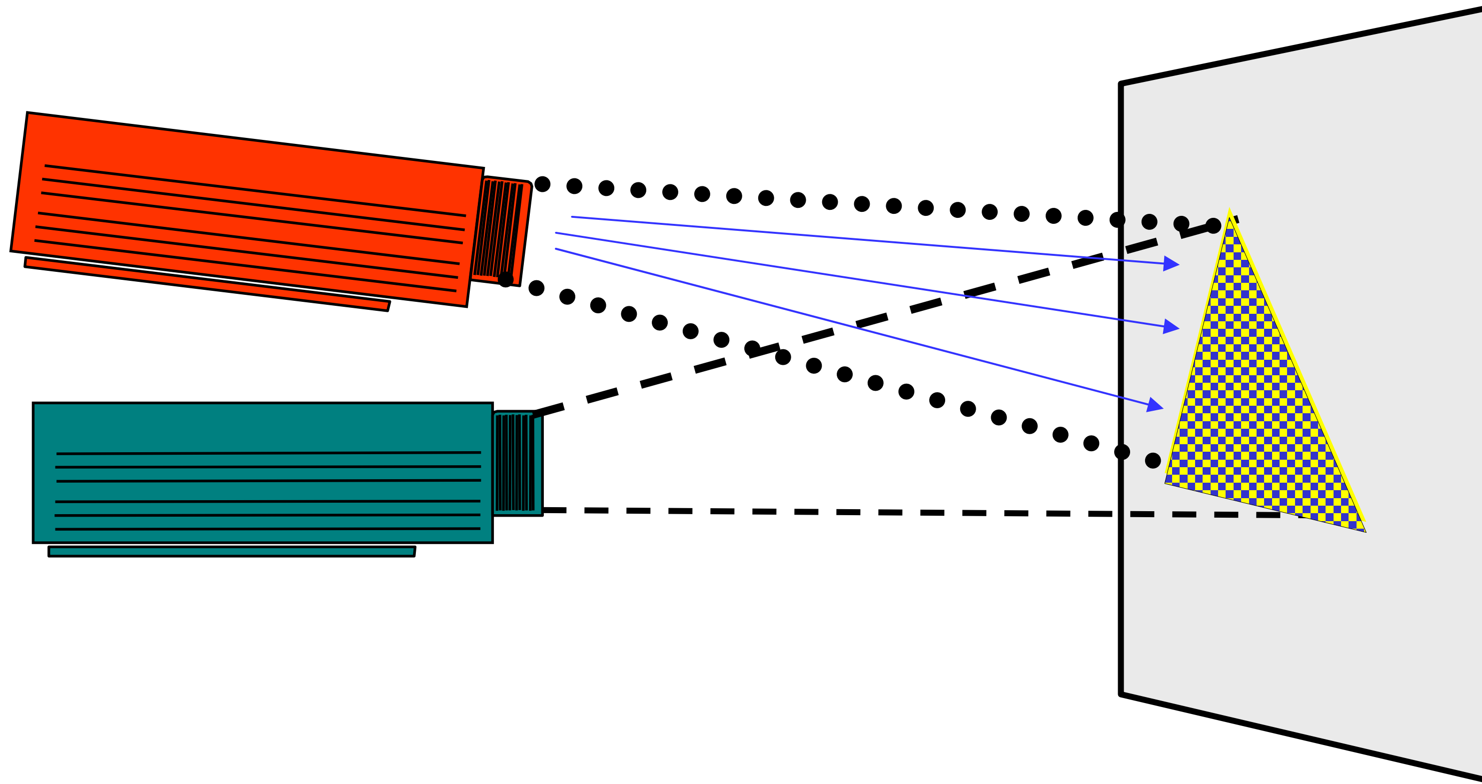
Now, imagine initially you build a model (the upper projecter),
based on which you "generate" your real software (the lower projecter)

Model-Based Development



Now, imagine initially you build a model (the upper projecter),
based on which you "generate" your real software (the lower projecter),
which implements the functionalities in your model

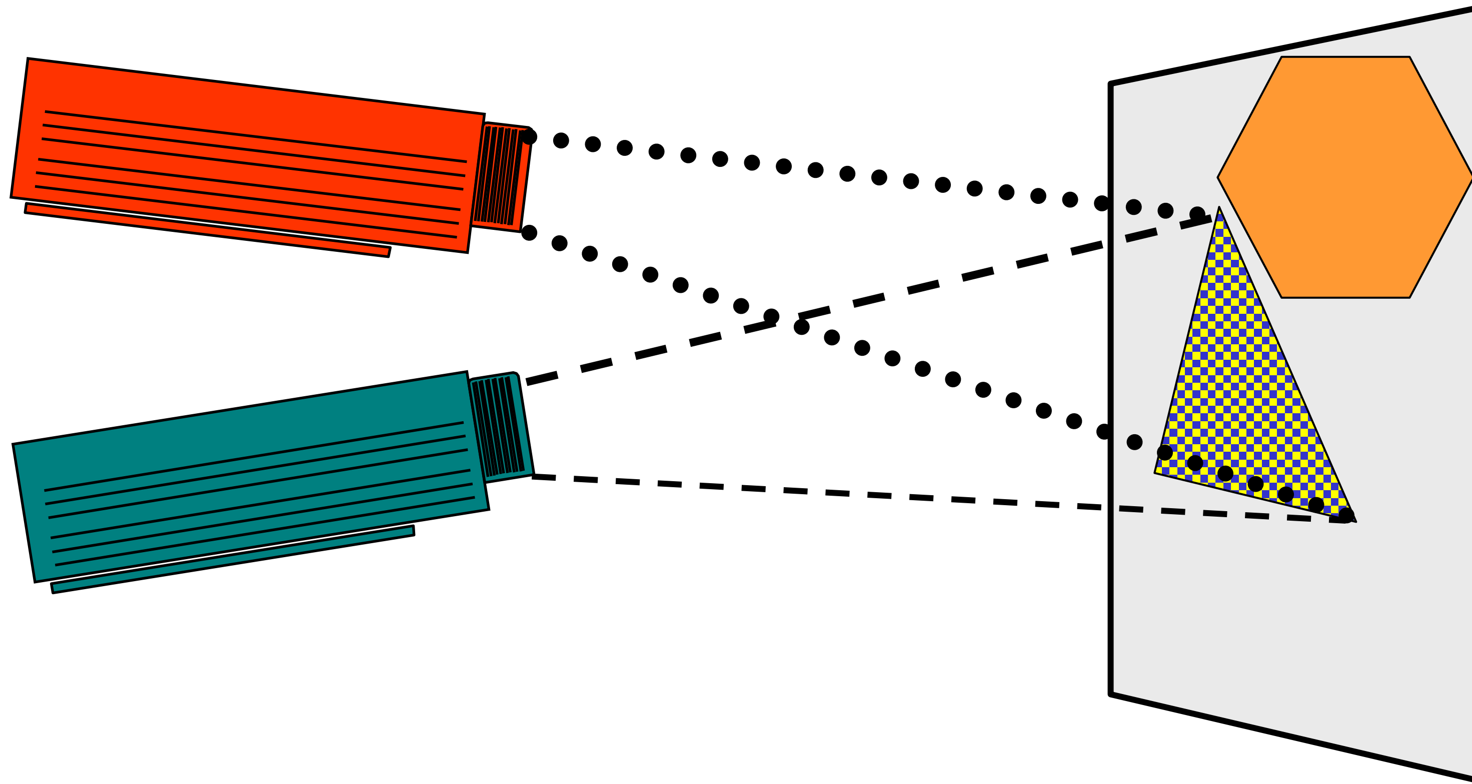
Model-Based Testing



From the model you generates tests to cover the behavior of the real software in a rigorous and systematic manner

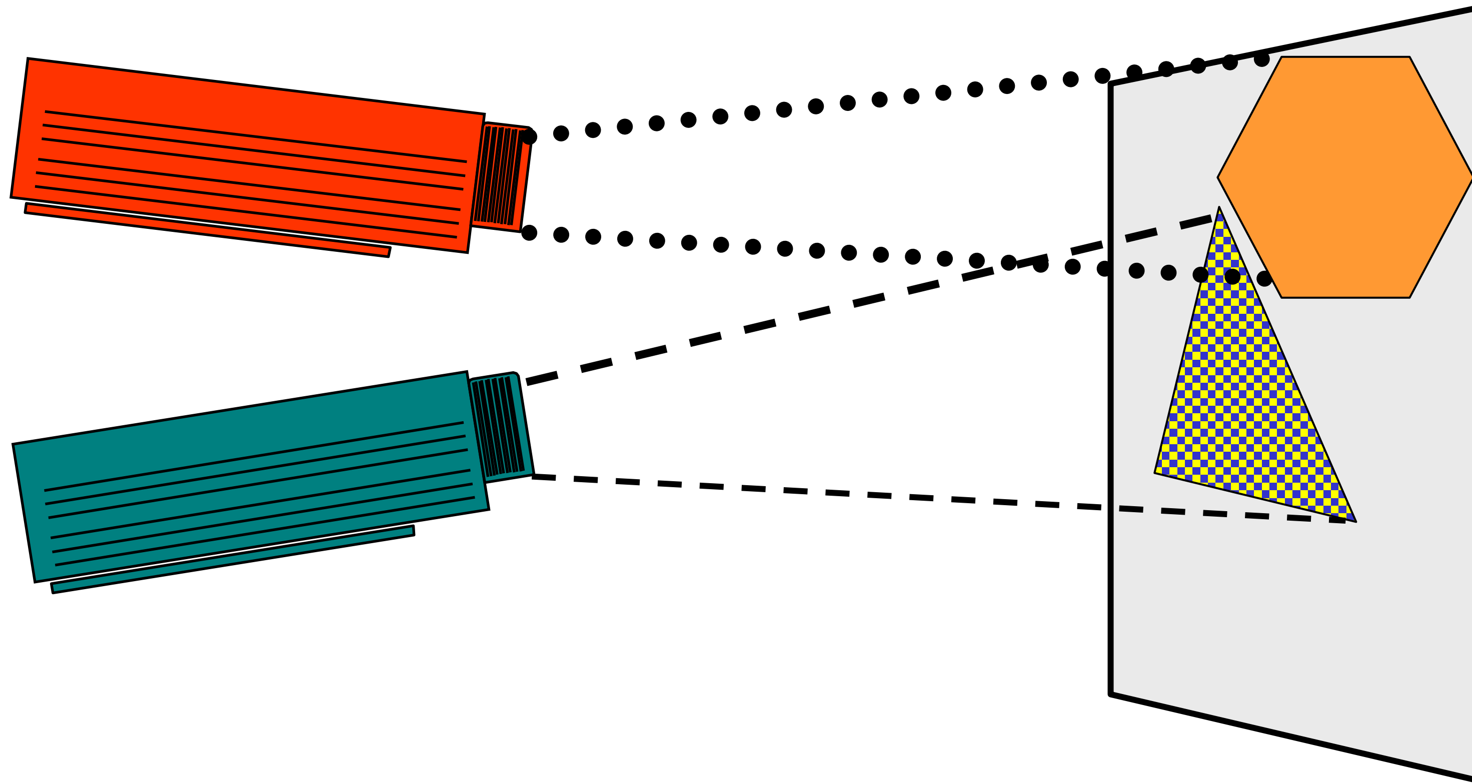
that's **much easier** than from the real software

Model-Based Testing



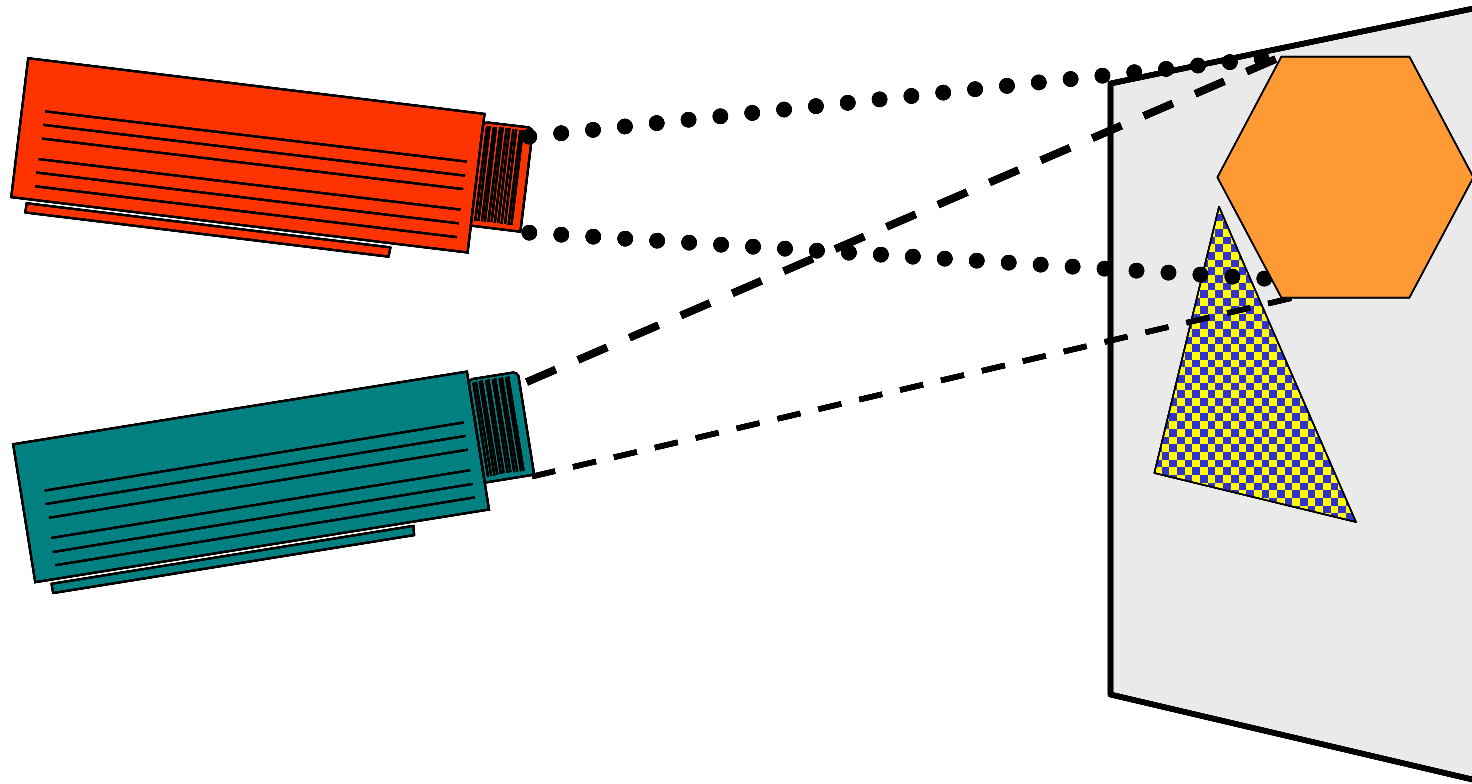
… and when there is a requirement change…

Model-Based Testing



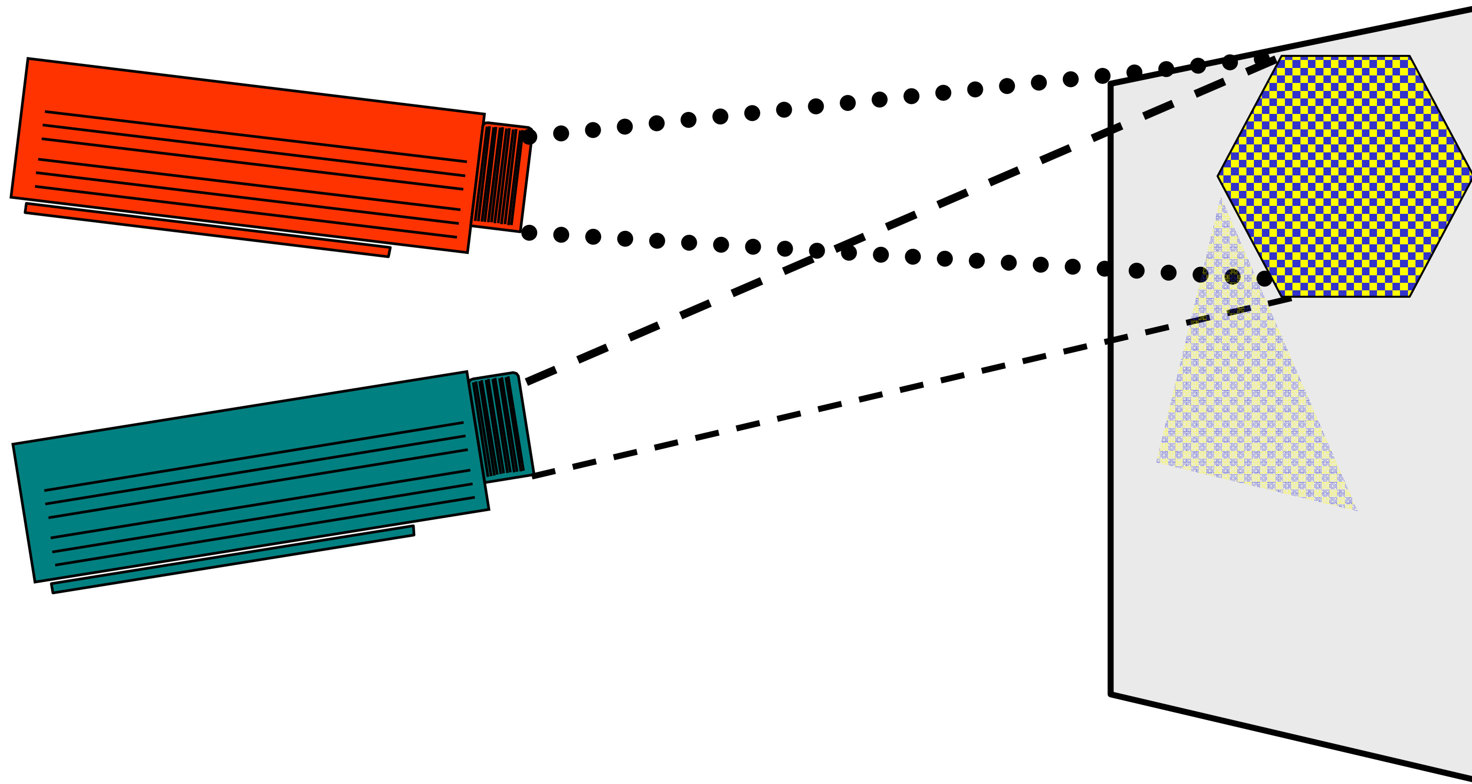
... you change the model...

Model-Based Testing



... and re-generate the real software from the new model...

Model-Based Testing



… and derive new tests from the new model….

So What's a Model?

- A model is a formal or semi-formal description of a system's behavior
- Models are (much) simpler than the systems they describe
  - They capture the key points rather than trivial details
- Models help us understand and predict the system's behavior

Formal models:
Finite State Machines (FSM) aka Automata, LTS, TA, CSP, CCS, Z, B, guarded command language, Message Sequence Charts, LSC, . . .

Semi-formal models:
UML, E-R Diagram, Data Flow Diagram, . . .

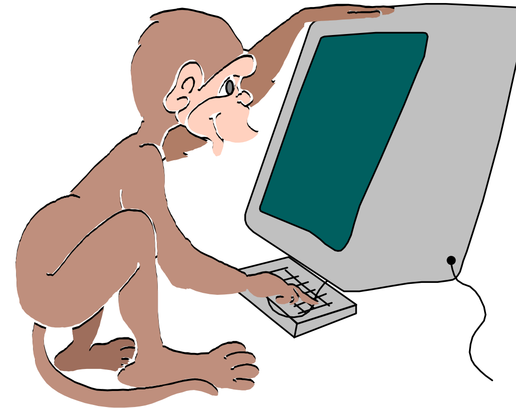
Approaches to Automated Testing

(static tests)

(generated tests)

Static Tests
(based on tester-written scripts)

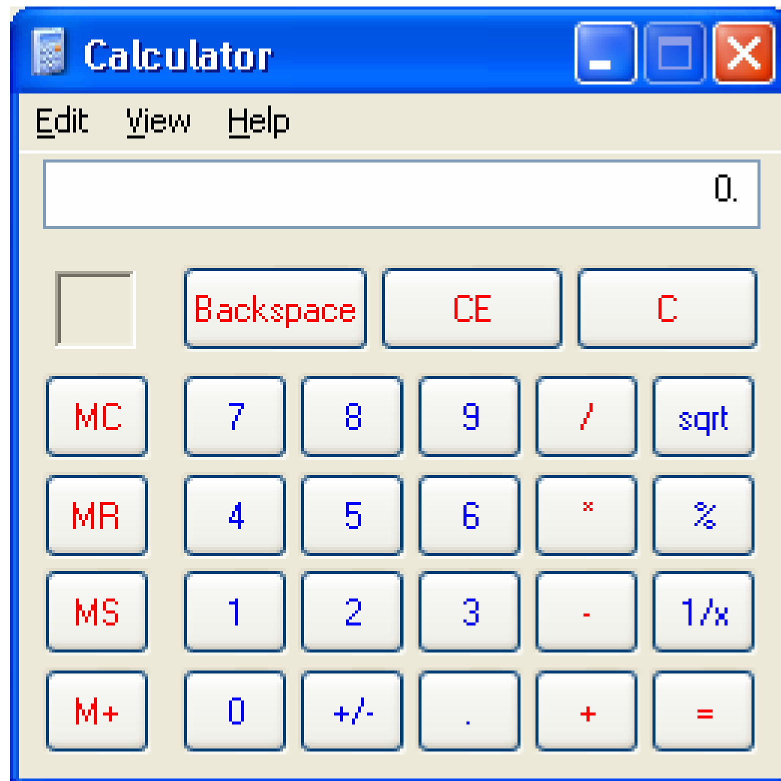
Model-Based Tests



Monkey Tests
(constant repetition of simple, isolated actions against the IUT)

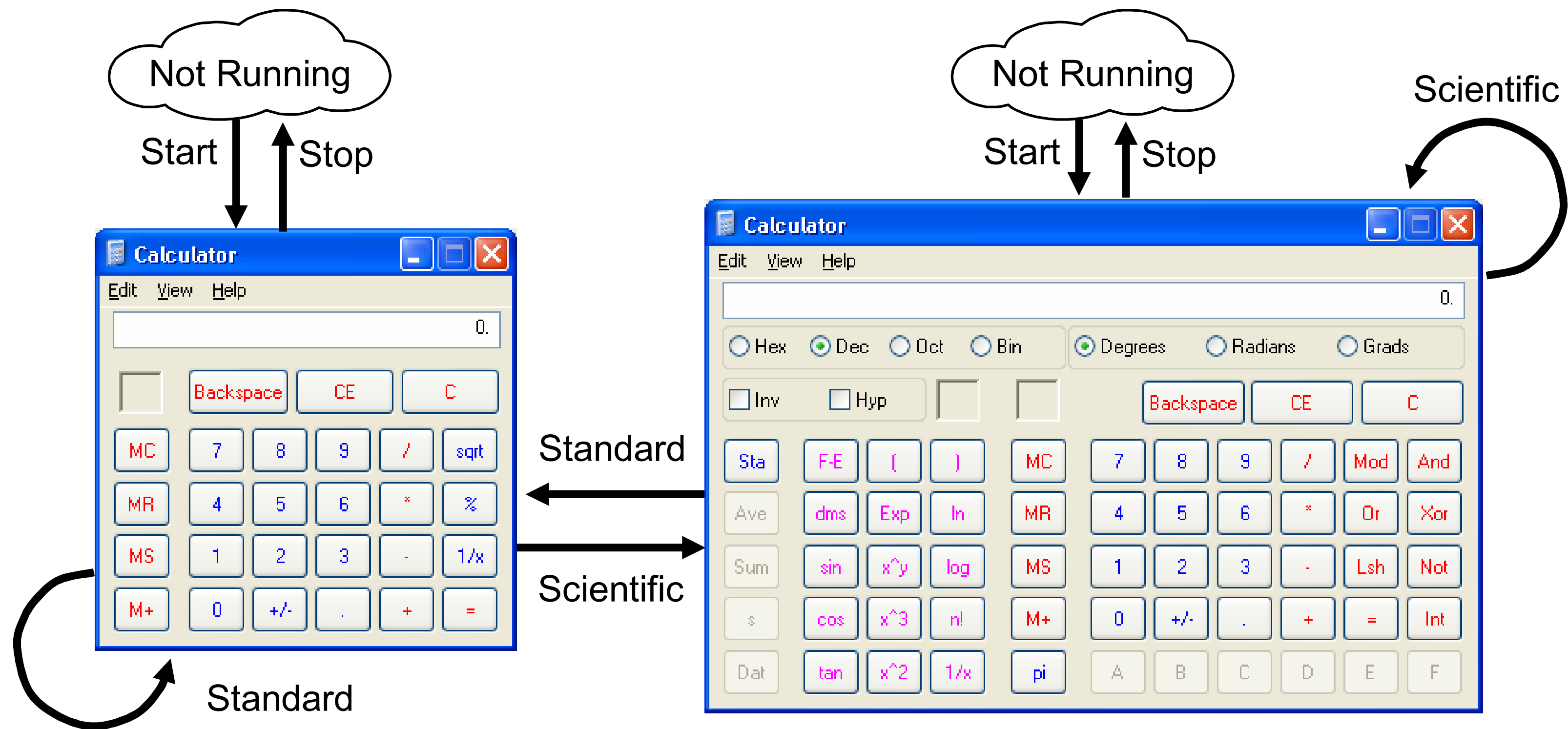
To go with a military analogy, **static tests** are like battlements: they are fairly cheap to build and maintain, and they can help keep you from losing ground you have already won. **Generated tests** are like tanks or ground troops: they need more thought in their design, and you need them if you want to win new territory, but they work best when they are on the move.

Calculator: A Fairly Typical GUI



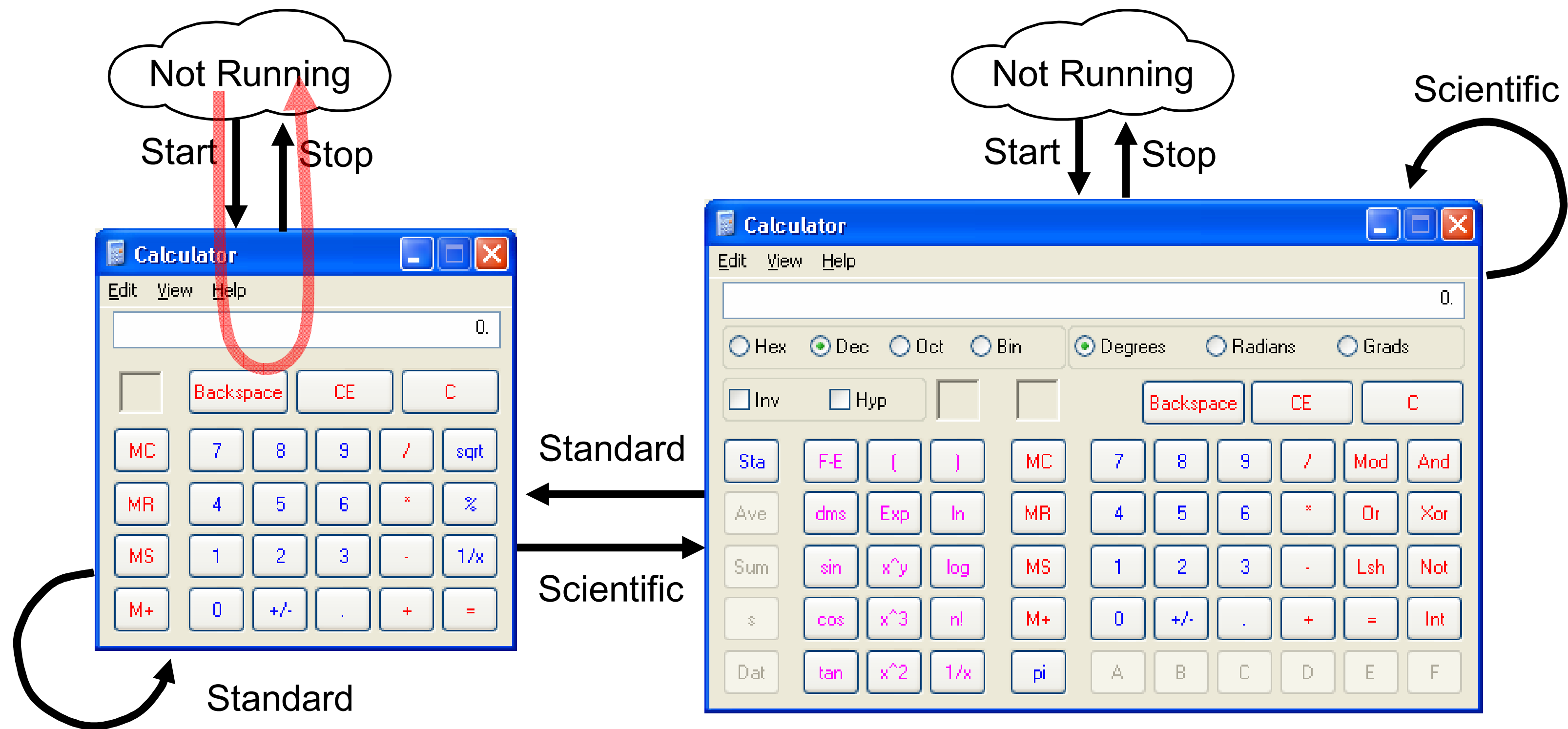
- Familiar enough
- Simple enough
- Complex enough
- Hard to test thoroughly

Calculator GUI Behavior



Static Tests vs. The Calculator

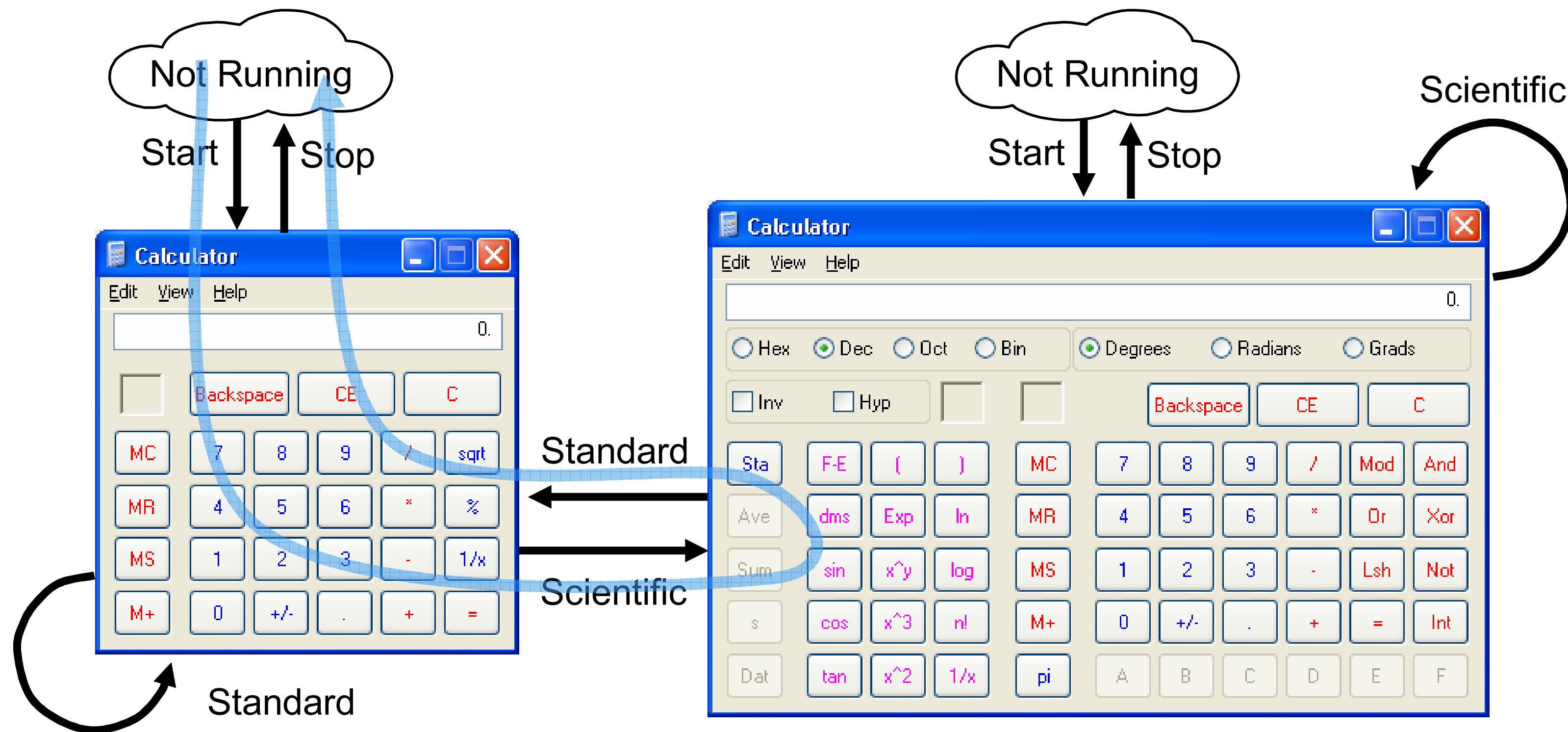
Test Case 1: Start Stop



Static Tests vs. The Calculator

Test Case 1: Start Stop

Test Case 2: Start Scientific Standard Stop

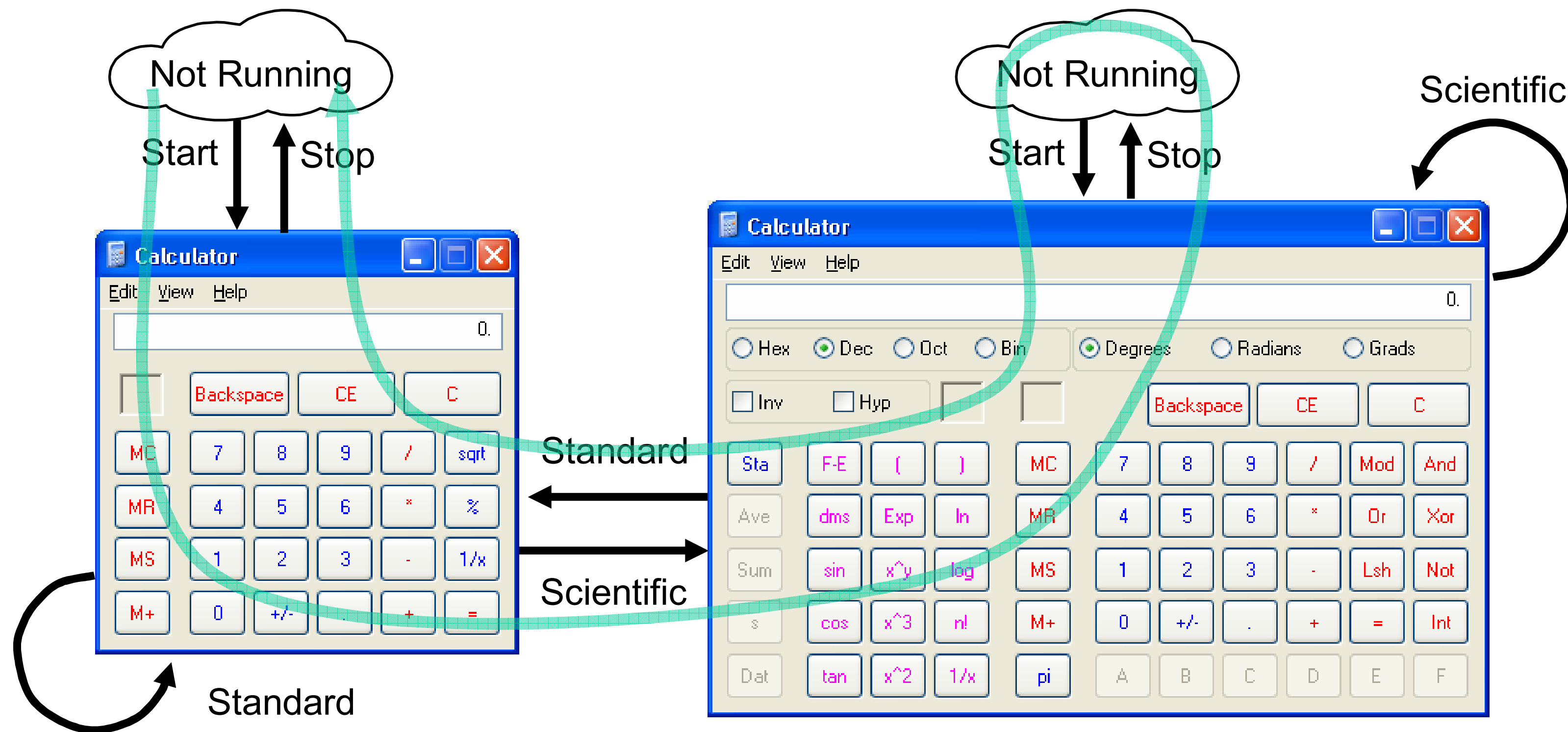


Static Tests vs. The Calculator

Test Case 1: Start Stop

Test Case 2: Start Scientific Standard Stop

Test Case 3: Start Scientific Stop Start Standard Stop



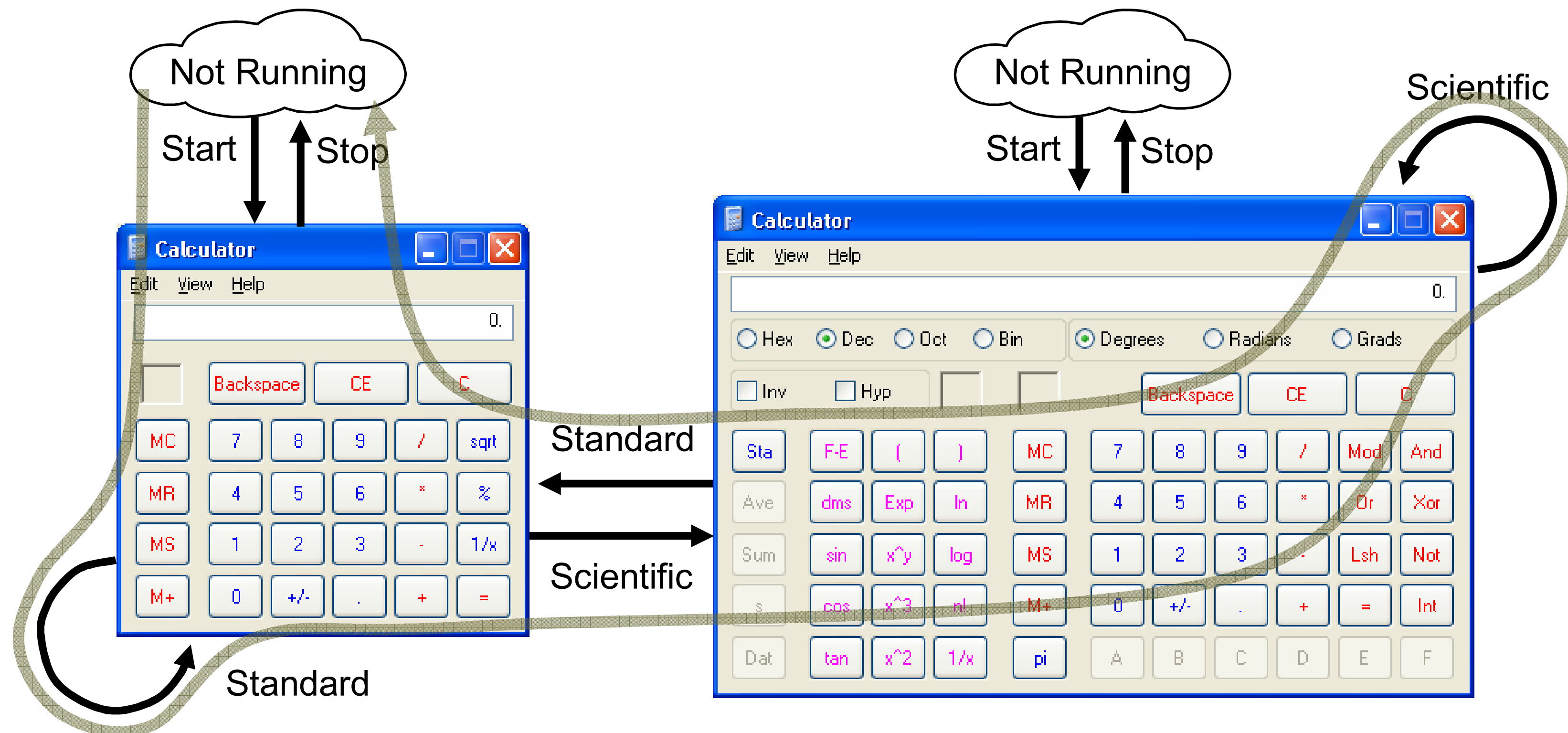
Static Tests vs. The Calculator

Test Case 1: Start Stop

Test Case 2: Start Scientific Standard Stop

Test Case 3: Start Scientific Stop Start Standard Stop

Test Case 4: Start Standard Scientific Scientific Standard Stop



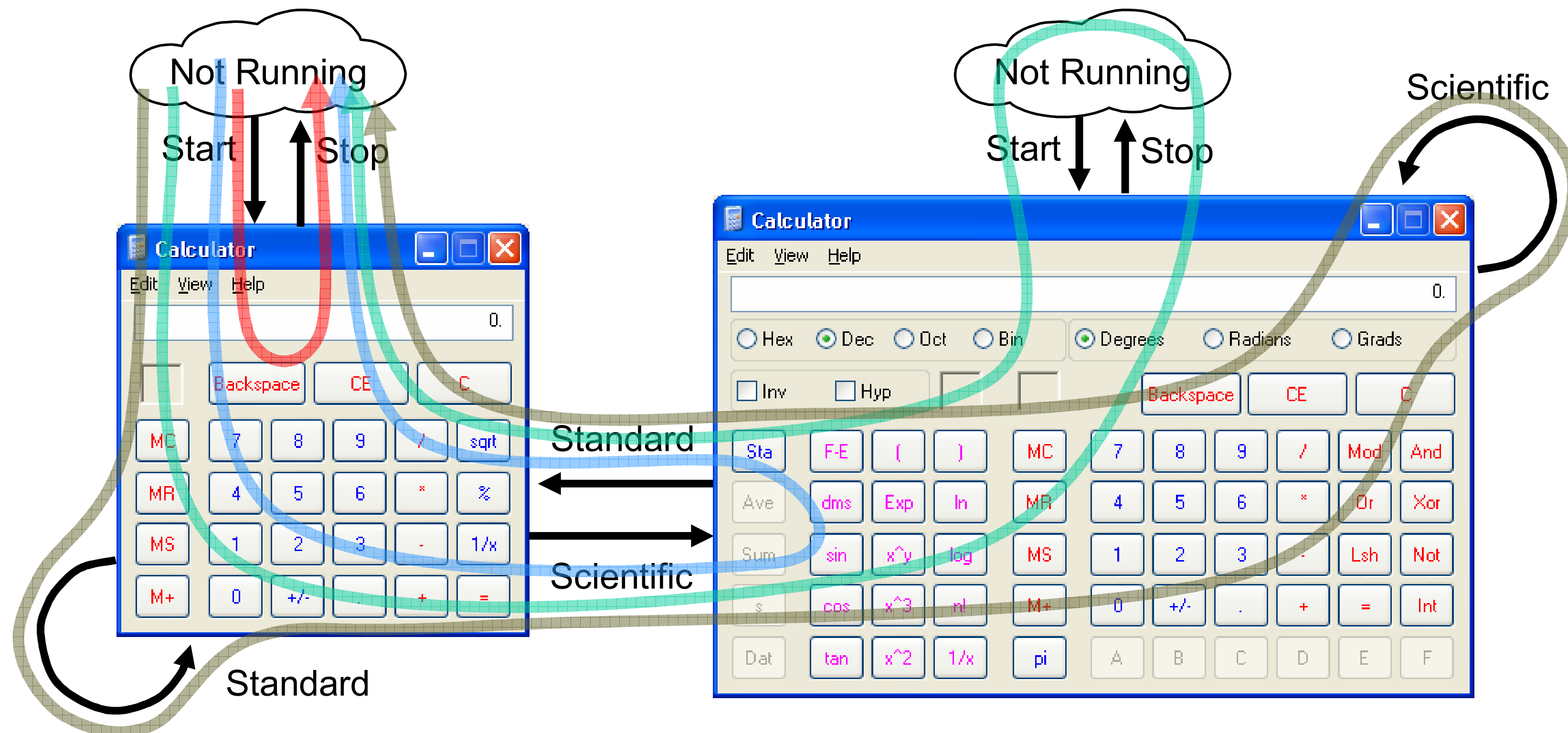
So, here's your test case library

Test Case 1: Start Stop

Test Case 2: Start Scientific Standard Stop

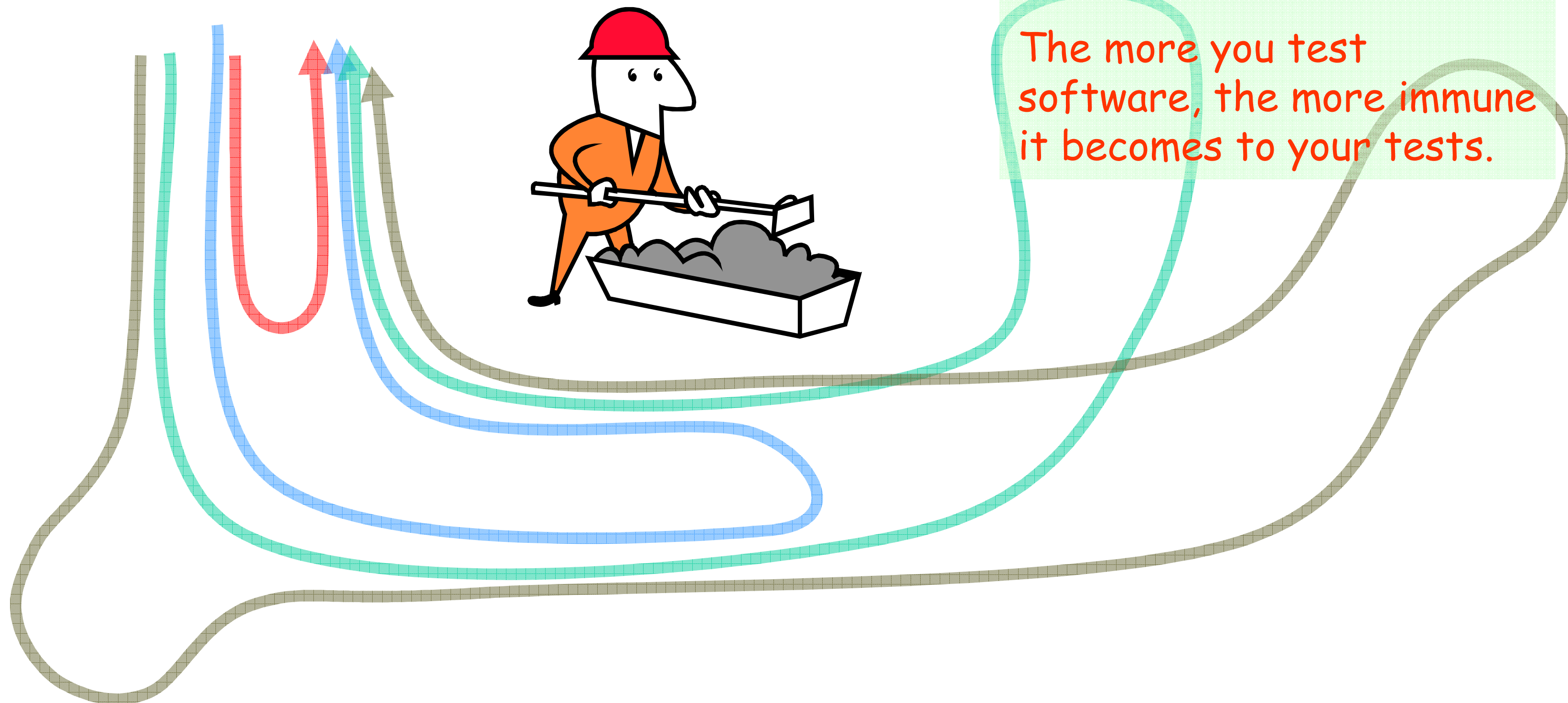
Test Case 3: Start Scientific Stop Start Standard Stop

Test Case 4: Start Standard Scientific Scientific Standard Stop



But, really, what are you left with?

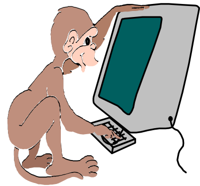
- Hard-coded test cases
- Tests that do only what you told them to
- Tests that wear out due to pesticide paradox



The more you test software, the more immune it becomes to your tests.

Monkey Tests vs. The Calculator

Test: Start-Standard-Standard-Standard-Scientific-Scientific-Scientific-Scientific ...



Not Running

Start

Stop

Calculator

Edit View Help

Standard

Standard

Scientific

Not Running

Start

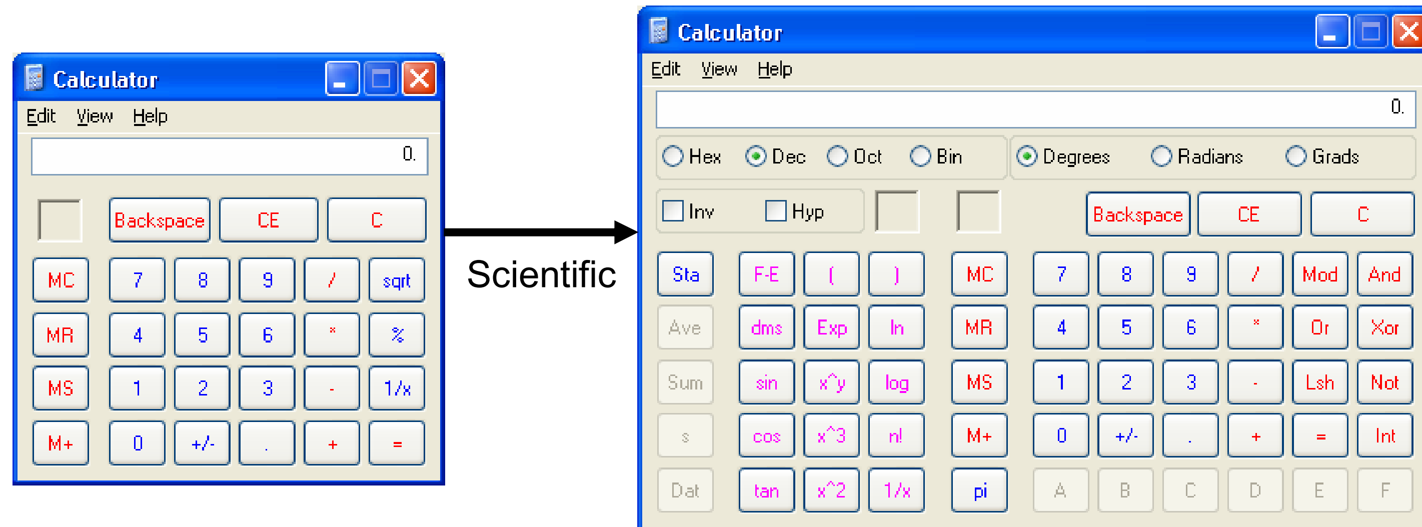
Stop

Calculator

Edit View Help

Scientific

MBT vs. The Calculator

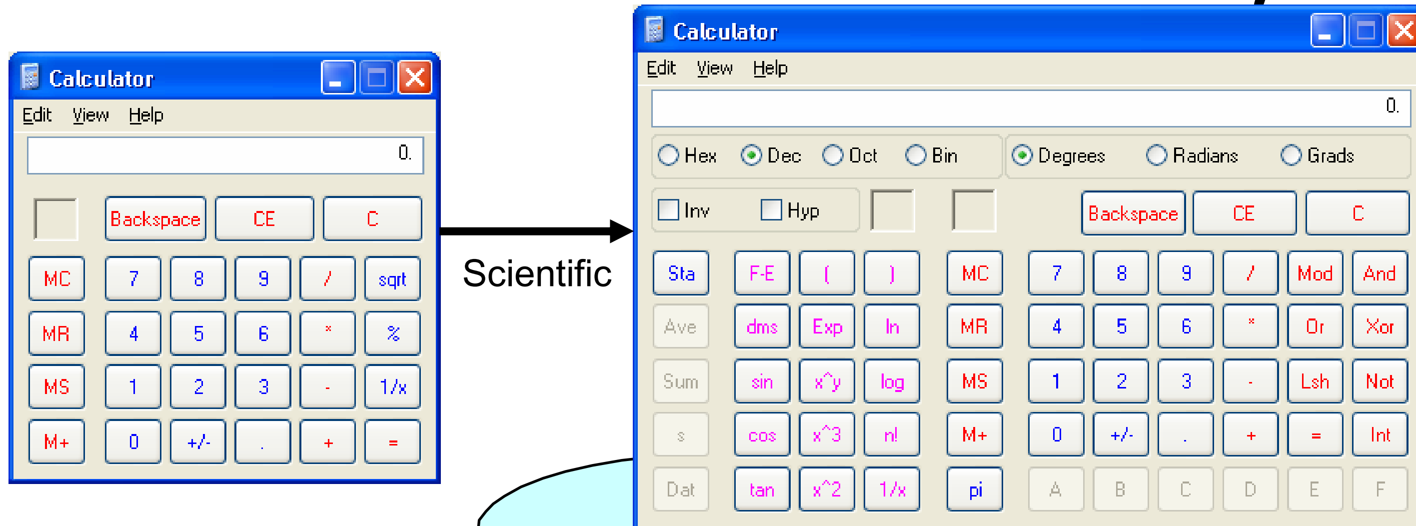


Setup: Calculator is running in "Standard" mode

Action: Select "Scientific" mode

Outcome: Did Calculator go correctly to "Scientific" mode?

We All Use Models Already



hmm …

if I am in **Standard** mode

and I select **Scientific** mode

I should end up in **Scientific** mode

Steps for Creating a Model

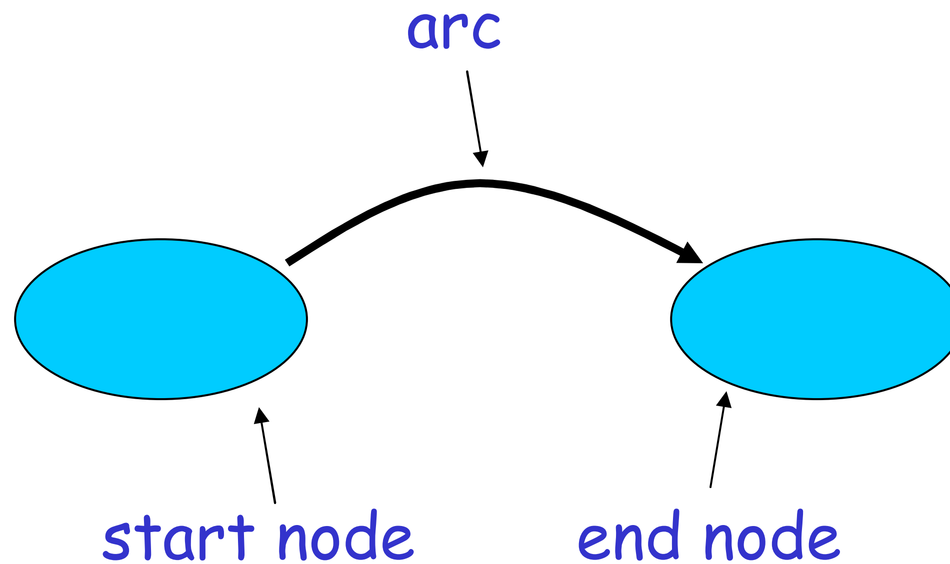
1. Walk through some scenarios
   a. What model do you have in your head?
   b. How do you know what you expect to see?

2. Figure out your scope:
   a. What are you testing?
   b. What are you ignoring?

3. Figure out a useful representation

A Graph is a Type of Model

A Few Quick Graph Theory Terms



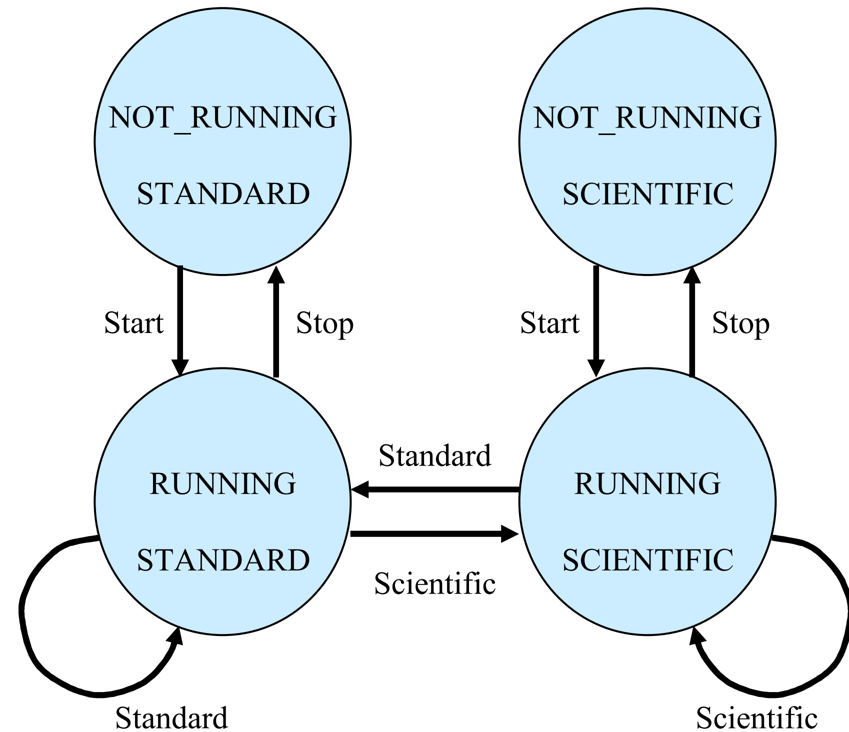
State Variables in the Calculator GUI

The System is either

- NOT_RUNNING, or
- RUNNING

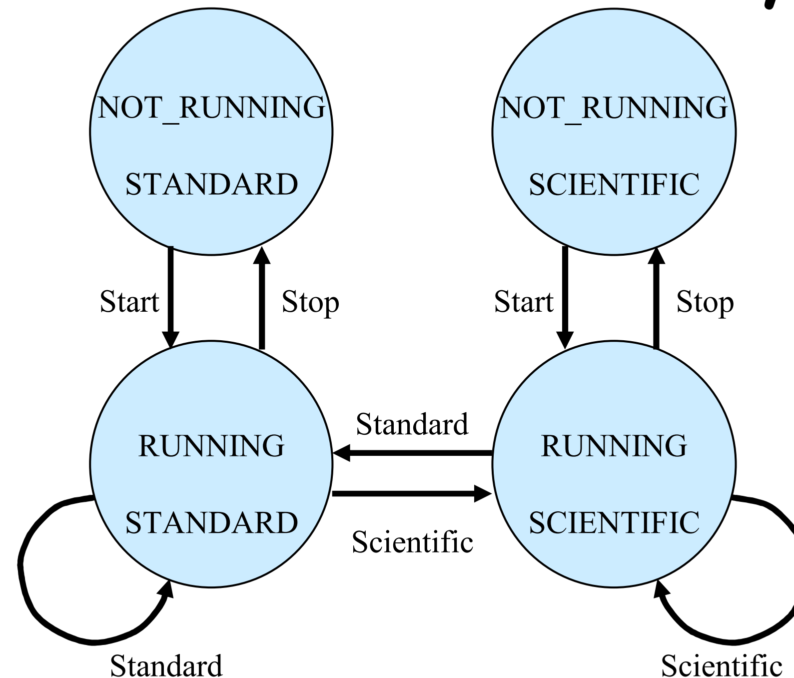
The Working Mode is either

- STANDARD, or
- SCIENTIFIC



Finite State Machine (FSM) model

All Actions Aren't Always Available



Rule: You can't execute the "Stop" action if the Calculator is not running

Finding the Rules

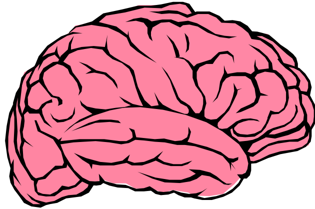
Stop

- When the System is NOT_RUNNING, the user <u>cannot</u> execute the **Stop** action.
- When the System is RUNNING, the user <u>can</u> execute the **Stop** action.
- After the **Stop** action executes, the System is NOT_RUNNING.

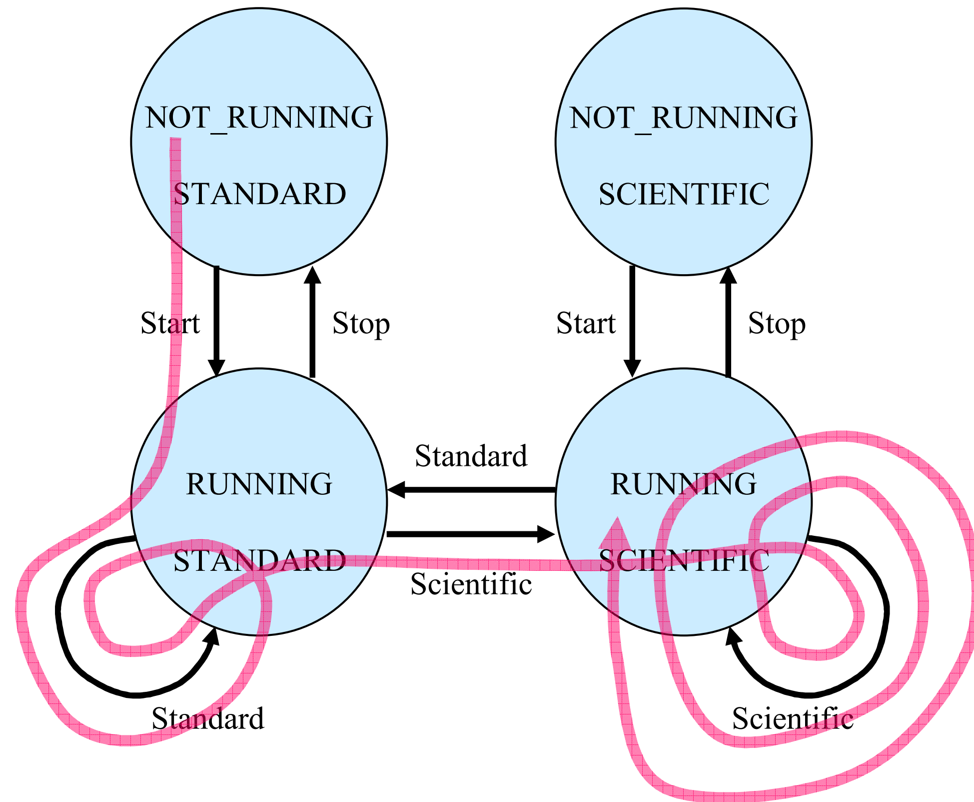
The Generated Finite State Table

Beginning State	Action	Ending State
NOT_RUNNING.STANDARD	Start	RUNNING.STANDARD
NOT_RUNNING.SCIENTIFIC	Start	RUNNING.SCIENTIFIC
RUNNING.STANDARD	Stop	NOT_RUNNING.STANDARD
RUNNING.SCIENTIFIC	Stop	NOT_RUNNING.SCIENTIFIC
RUNNING.STANDARD	Standard	RUNNING.STANDARD
RUNNING.STANDARD	Scientific	RUNNING.SCIENTIFIC
RUNNING.SCIENTIFIC	Standard	RUNNING.STANDARD
RUNNING.SCIENTIFIC	Scientific	RUNNING.SCIENTIFIC

A Random Walk

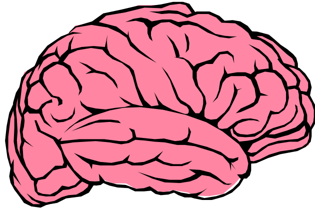


Start
Standard
Standard
Scientific
Scientific
Scientific
Scientific
…

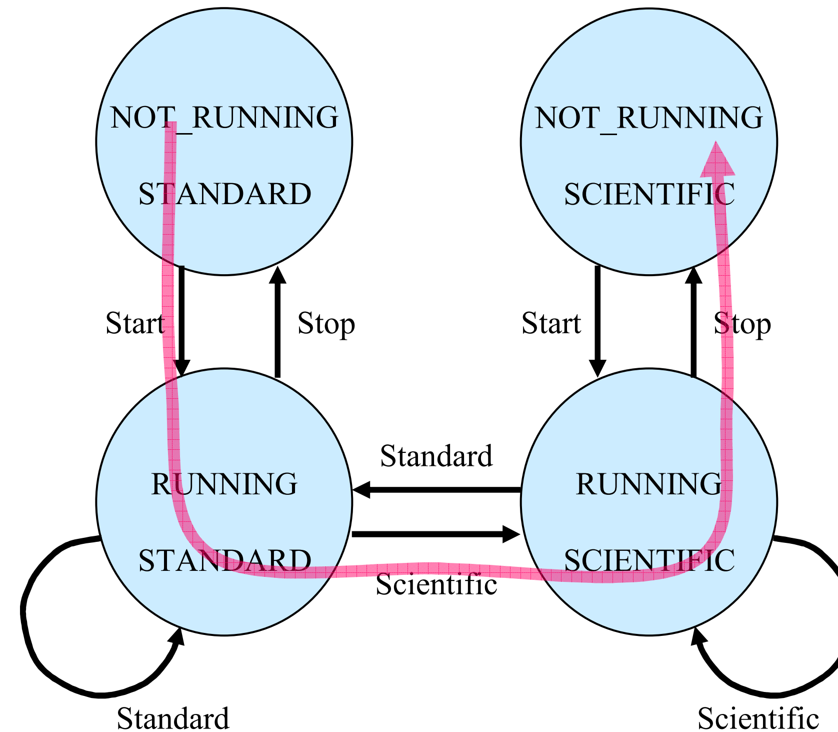


re-inventing the monkey

All-States ("salesman")

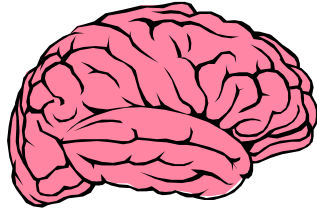


Start
Scientific
Stop
Start
Standard
Stop

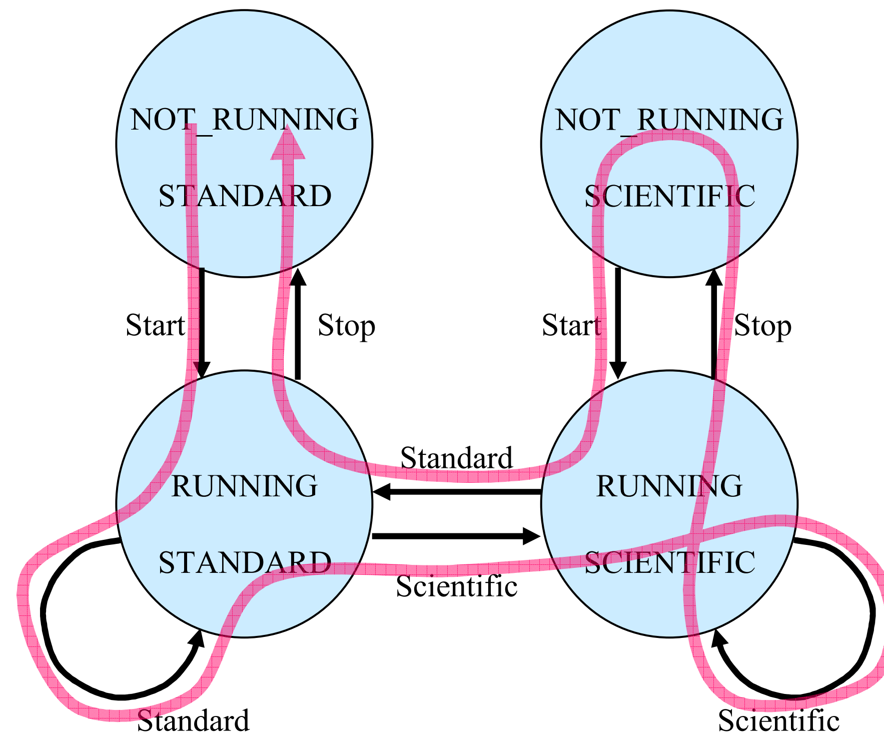


to reach every state in the model

All-Transitions ("postman")

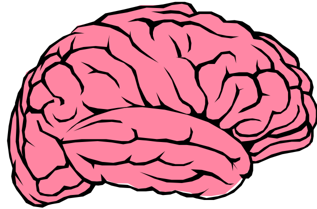


Start
Standard
Scientific
Scientific
Stop
Start
Standard
Stop

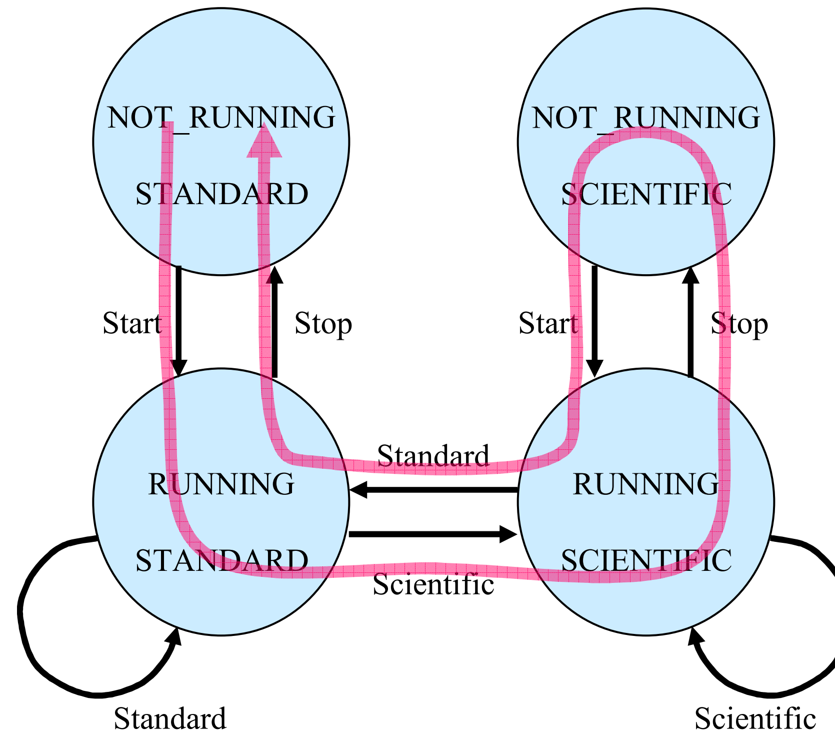


to execute every action

All State-Changing Transitions

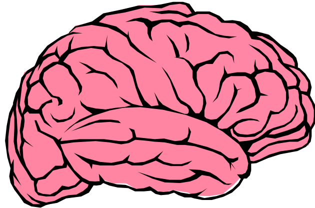


Start
Scientific
Stop
Start
Standard
Stop



to execute every state-changing action

Shortest Paths First

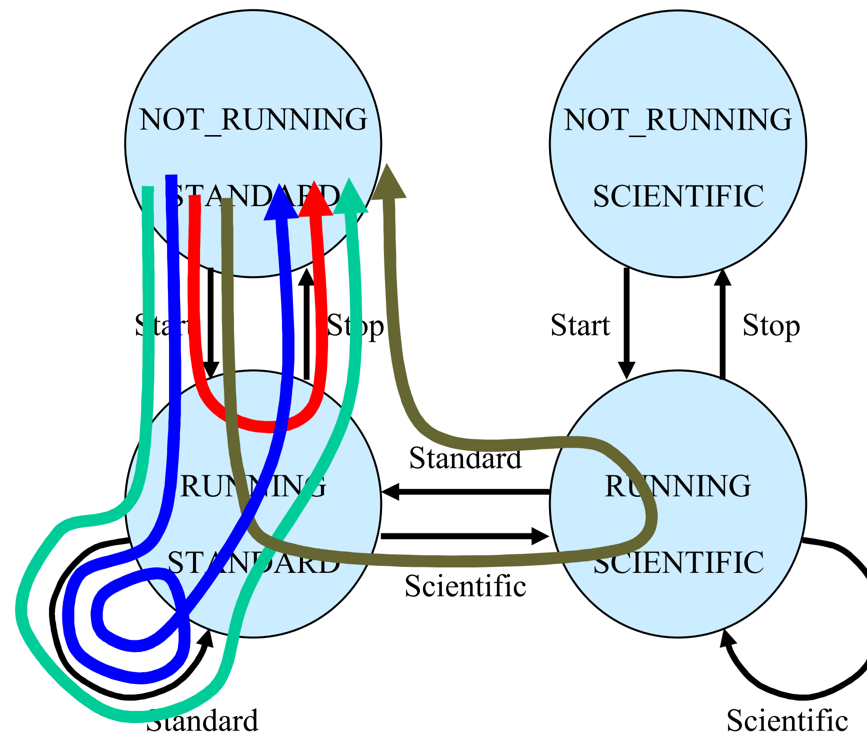


Length = 2
Start Stop

Length = 3
Start Standard Stop

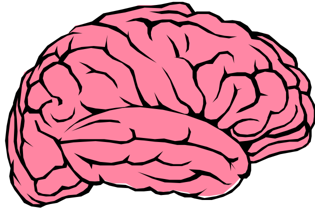
Length = 4
Start Standard Standard Stop
Start Scientific Standard Stop

and so on …



to execute every path (eventually!)

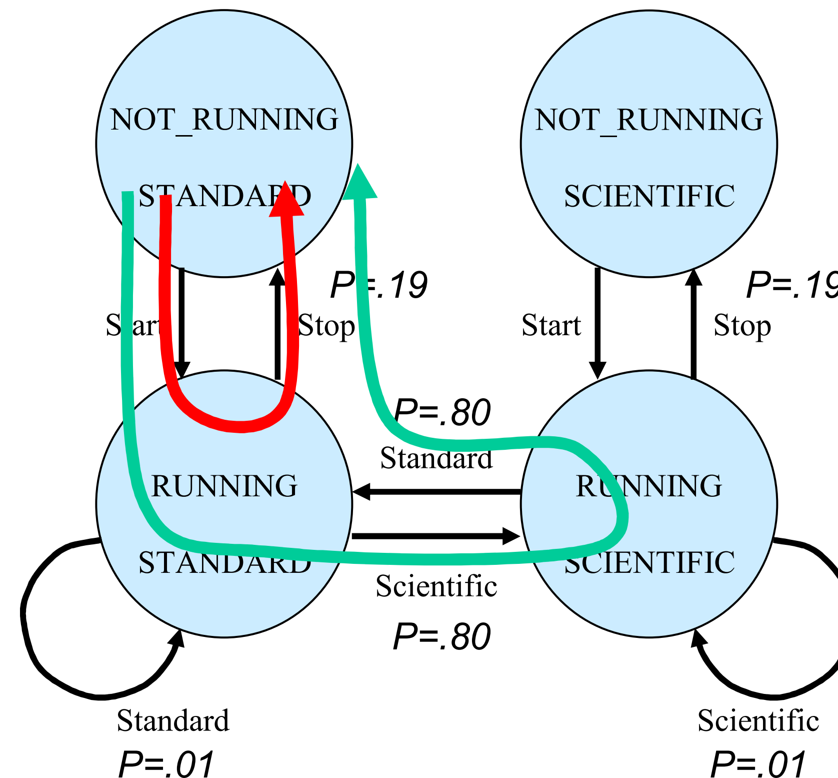
Most Likely Paths First



Probability = 0.19
Start Stop

Probability = 0.1216
Start Scientific Standard Stop

and so on …

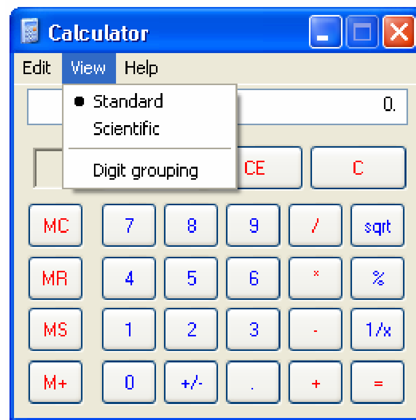


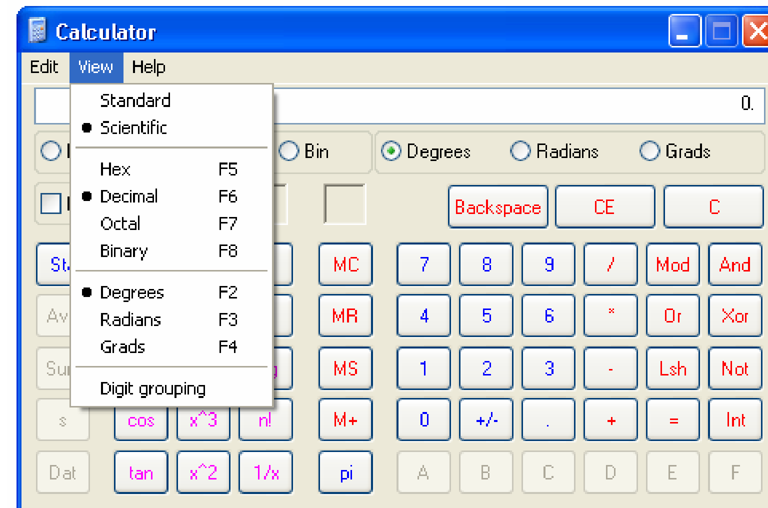
to execute favored paths in order

Executing the Test Actions

```
open "test_sequence.txt" for input as #infile      ‘get the list of test actions

while not (EOF(infile))

    line input #infile, action                     ‘read in a test action

    select case action

    case “Start“                                   ‘ Start the calculator
        run("C:\WINNT\System32\calc.exe”)          ‘ VT call to start calculator

    case “Standard“                                ‘ choose Standard mode
        WMenuSelect(“View\Standard")               ‘ VT call to select Standard

    case “Scientific“                              ‘ choose Scientific mode
        WMenuSelect(“View\Scientific")             ‘ VT call to select Scientific

    case “Stop“                                    ‘ Stop the calculator
        WSysMenu (0)                               ‘ VT call to bring up system menu
        WMenuSelect ("Close")                      ‘ VT call to select Close

    end select

wend
```

Use Rules as Heuristic Test Oracles

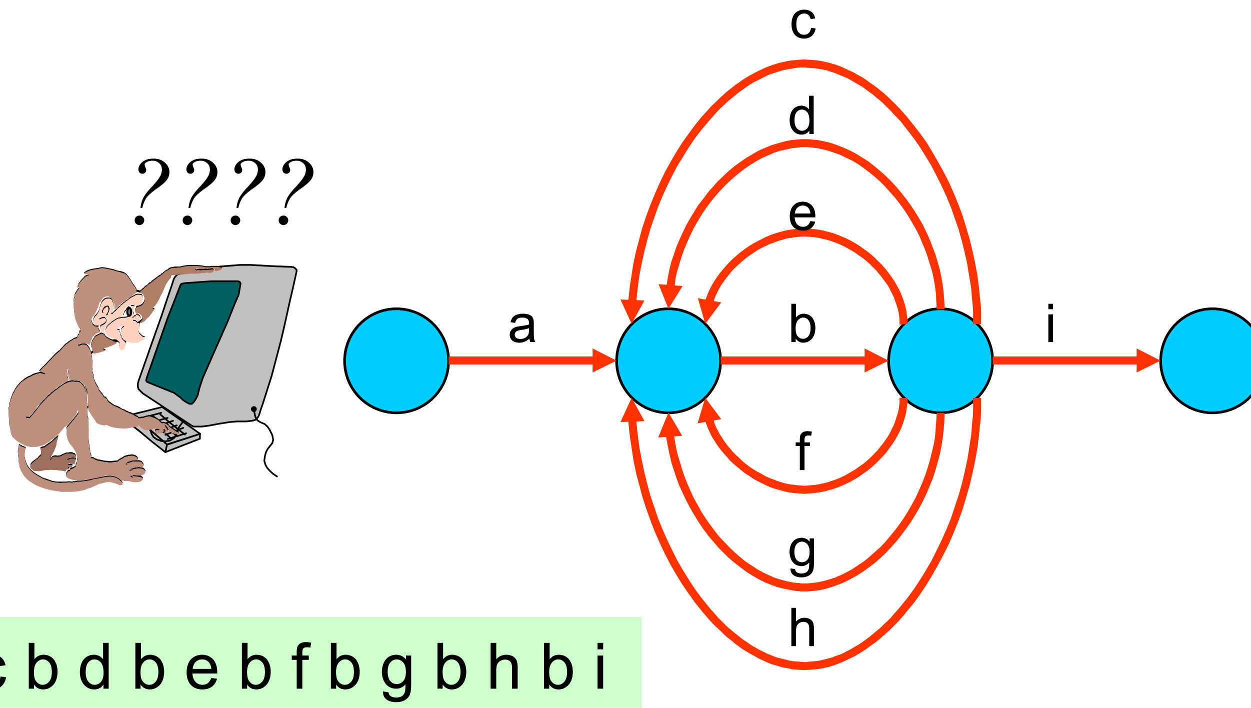





```
if ( (setting_mode = STANDARD) _

AND NOT WMenuChecked(“View\Standard") ) then

 print "Error: Calculator should be Standard mode“
 stop

endif
```

```
‘if we are in Standard mode

‘but Standard is not check-marked

 ‘alert the tester
```

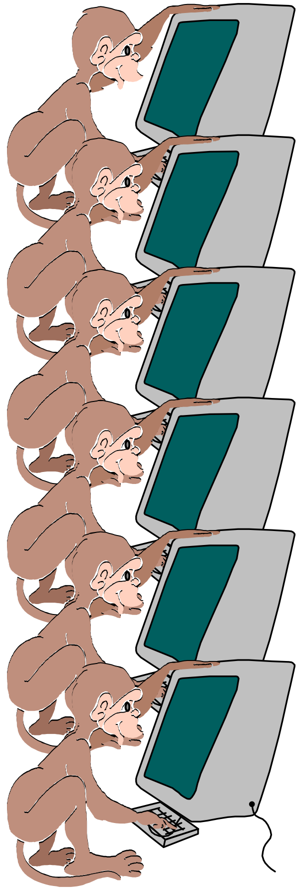
Executing tests quickly

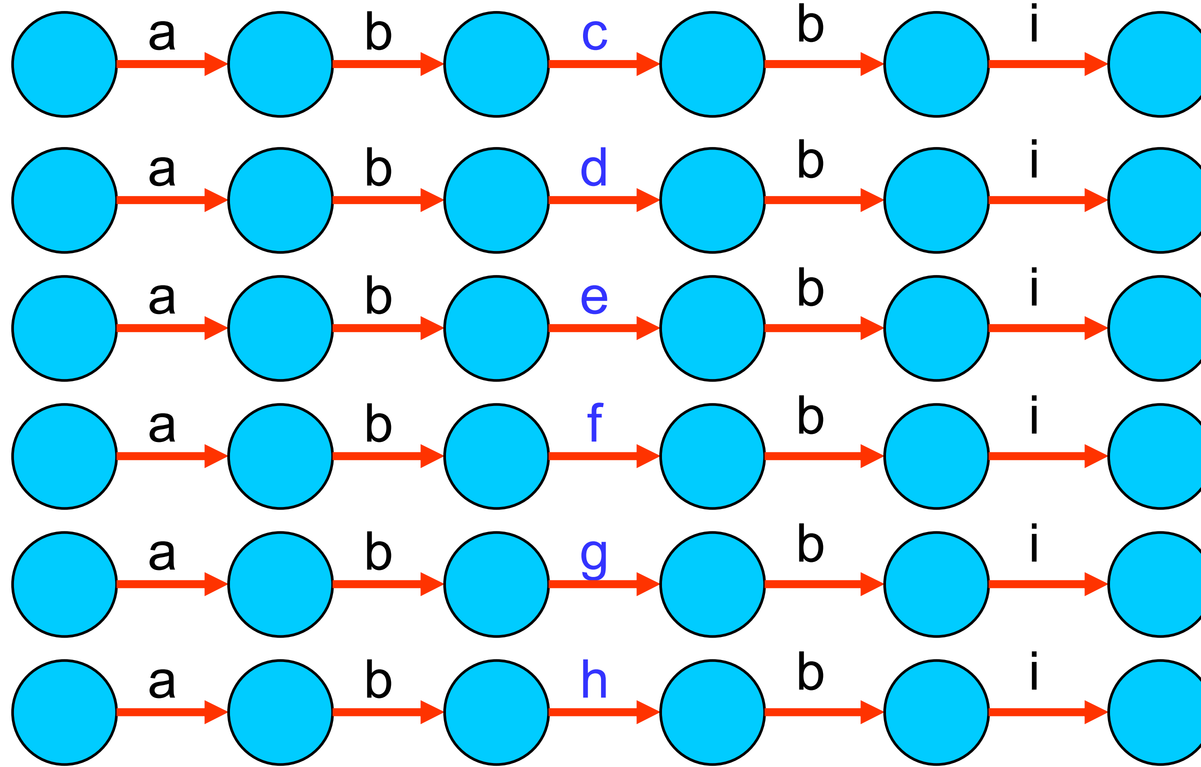


a b c b d b e b f b g b h b i

A single test machine approach takes 15 time units.

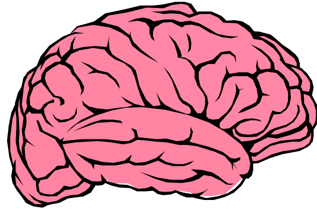
But distributing the work …





… gets the job done in 1/3 the time!

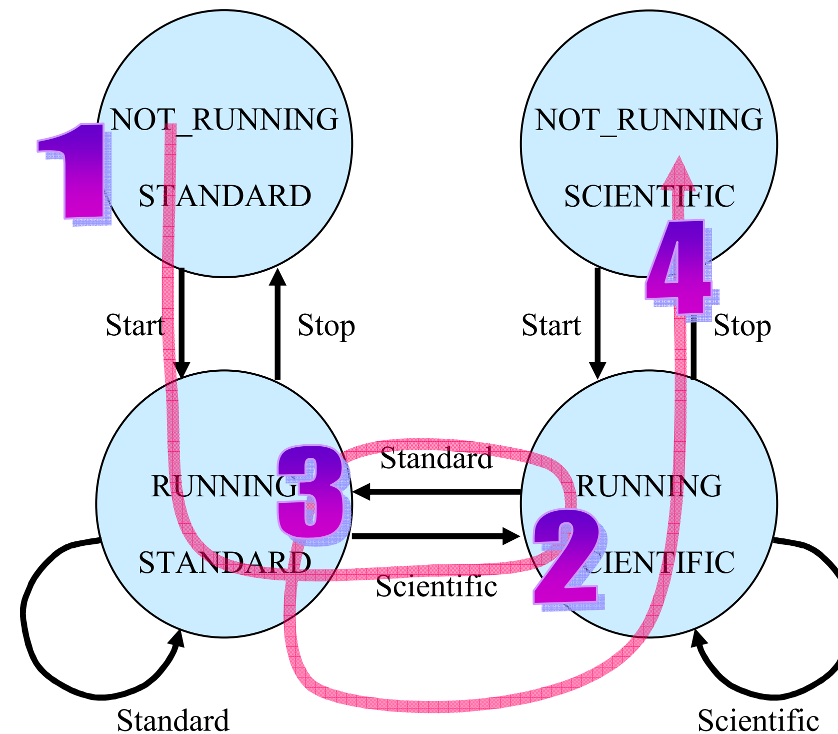
An Anti-Random Walk



Start
Scientific

Standard

Scientific
Stop

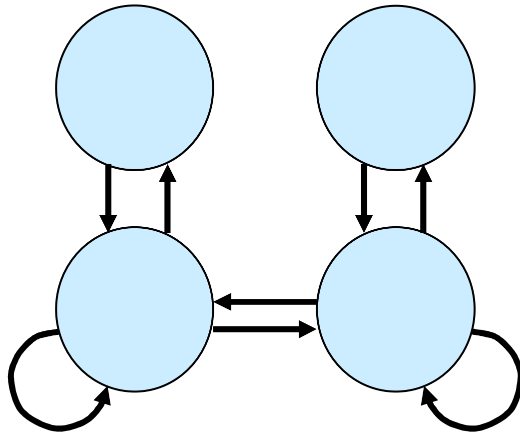


to visit states most different from where you've been

Here the most different state of (NOT_RUNNING, STANDARD) is (RUNNING, SCIENTIFIC).

Models + Traversals = Model-Based Testing

- State models are good at representing system behavior
- You can use models to generate tests
- Different algorithms can provide tests to suit your needs:



- Random walk
- All states
- All transitions
- State-changing transitions
- Shortest paths first
- Most likely paths first
- Anti-random walks

Q & A