Reliable Systems Engineering

Tom Henzinger

EPFL

1. What do we do?
2. Why do we do it?
3. How do we do it?

We catch bugs.

We catch bugs.

bug *n* an unexpected defect, fault, flaw, or imperfection ⟨the software is full of ~*s*⟩

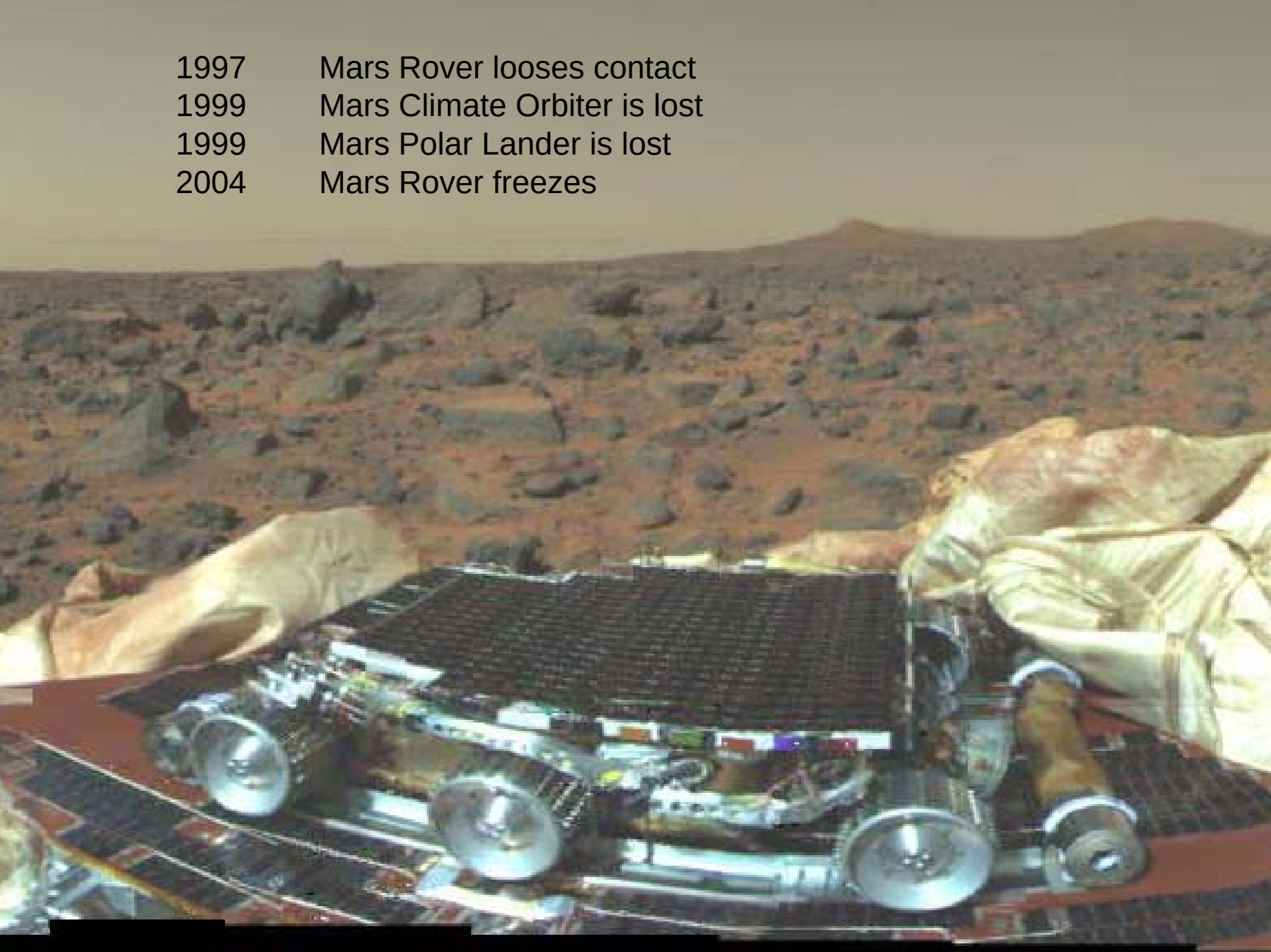
[Webster]


Windows
An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000. It may be possible to continue normally.
* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue

June 4, 1996

The European Ariane 5 rocket explodes 40 s into its maiden flight due to a software bug.

1997 Mars Rover looses contact
1999 Mars Climate Orbiter is lost
1999 Mars Polar Lander is lost
2004 Mars Rover freezes



August 14, 2003

A programming error has been identified as the cause of the Northeast power blackout. The failure occurred when multiple computer systems trying to access the same information at once got the equivalent of busy signals.

[Associated Press]

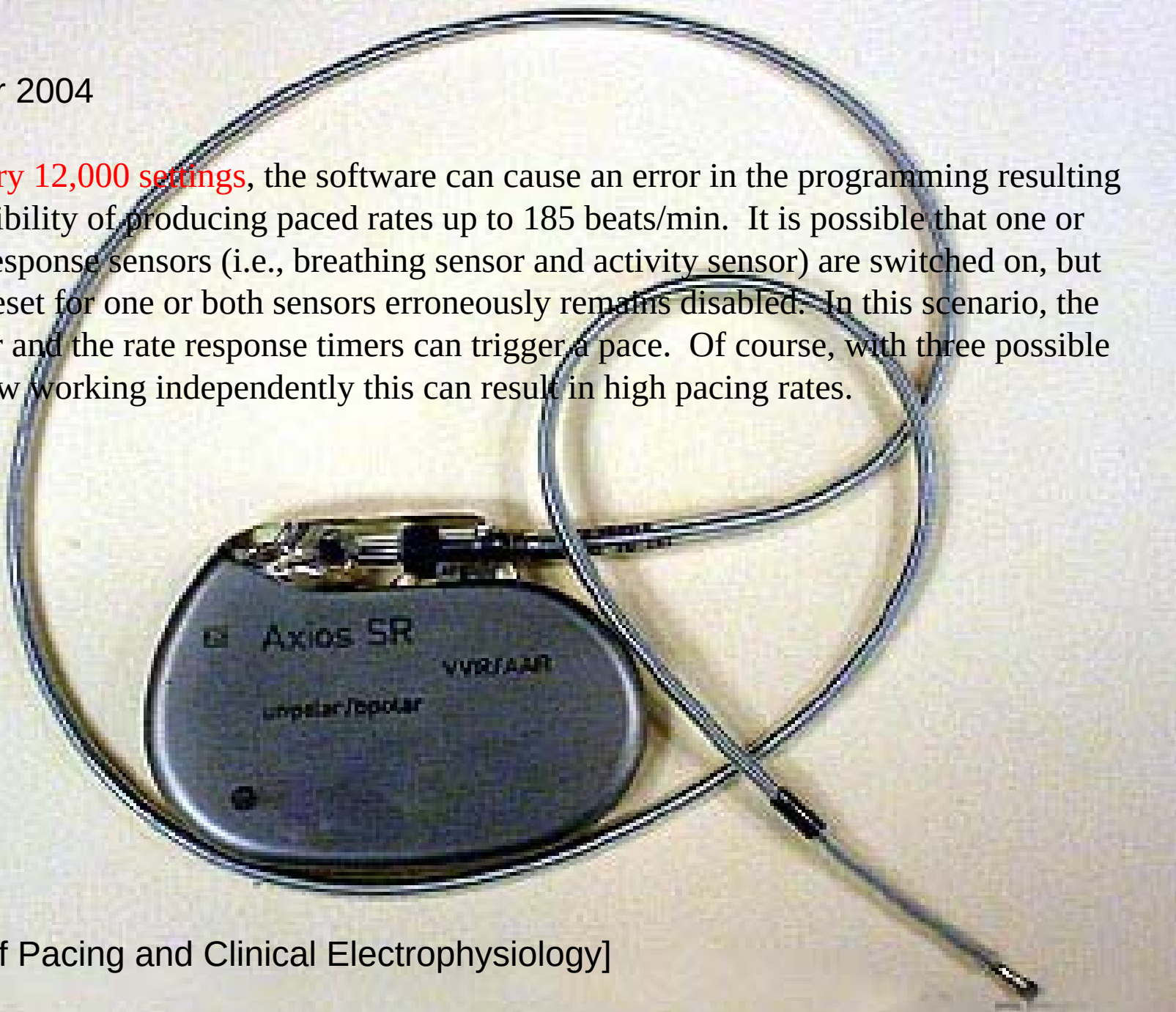
Price tag: $ 6-10 billion

2002 study by NIST:

Software bugs cost the US economy $ 60 billion annually (0.6 % of GDP).

December 2004

In 1 of every 12,000 settings, the software can cause an error in the programming resulting in the possibility of producing paced rates up to 185 beats/min. It is possible that one or both rate response sensors (i.e., breathing sensor and activity sensor) are switched on, but the timer reset for one or both sensors erroneously remains disabled. In this scenario, the clock timer and the rate response timers can trigger a pace. Of course, with three possible triggers now working independently this can result in high pacing rates.



[Journal of Pacing and Clinical Electrophysiology]

January 1-7, 2002

Ganz ohne Geheimzahl konnten Postbank-Kunden bis Montag mit ihrer Sparcard unbegrenzt Geld abheben. “Wir haben bereits nach wenigen Tagen den Fehler bei der neu installierten Software bemerkt und korrigiert. Wir wissen nur von einem einzigen Fall in Hamburg, wo der Softwarefehler zufaellig entdeckt wurde. Der Betroffene muss das abgehobene Geld zurueckgeben.”

[Sueddeutsche Zeitung]

Boeing could not assemble and integrate the fly-by-wire system until it solved problems with the databus and the flight management software. Solving these problems took more than a year longer than Boeing anticipated. In April, 1995, the FAA certified the 777 as safe.

Total development cost:	$ 3 billion
Software integration and validation cost:	one third of total

August 2005

As a Malaysia Airlines jetliner cruised from Perth, Australia, to Kuala Lumpur, Malaysia, one evening last August, it suddenly took on a mind of its own and zoomed 3,000 feet upward. The captain disconnected the autopilot and pointed the Boeing 777's nose down to avoid stalling, but was jerked into a steep dive. He throttled back sharply on both engines, trying to slow the plane. Instead, the jet raced into another climb. The crew eventually regained control and manually flew their 177 passengers safely back to Australia.

August 2005

As a Malaysia Airlines jetliner cruised from Perth, Australia, to Kuala Lumpur, Malaysia, one evening last August, it suddenly took on a mind of its own and zoomed 3,000 feet upward. The captain disconnected the autopilot and pointed the Boeing 777's nose down to avoid stalling, but was jerked into a steep dive. He throttled back sharply on both engines, trying to slow the plane. Instead, the jet raced into another climb. The crew eventually regained control and manually flew their 177 passengers safely back to Australia.

Investigators quickly discovered the reason for the plane's roller-coaster ride 38,000 feet above the Indian Ocean. A defective software program had provided incorrect data about the aircraft's speed and acceleration, confusing flight computers.

August 2005

With well over five million lines of code used on the latest jetliners, it's increasingly difficult to detect and fix software problems before they surprise pilots. Plane makers are accustomed to testing metals and plastics under almost every conceivable kind of extreme stress, but it's impossible to run a big computer program through every scenario to detect the bugs that invariably crop up.

August 2005

With well over five million lines of code used on the latest jetliners, it's increasingly difficult to detect and fix software problems before they surprise pilots. Plane makers are accustomed to testing metals and plastics under almost every conceivable kind of extreme stress, but it's impossible to run a big computer program through every scenario to detect the bugs that invariably crop up.

Specialists say the biggest problems in aviation software don't stem from bugs in the code of a single program but rather from the interaction between two different parts of a plane's computer system. In extreme cases, foul-ups can lead to sudden loss of control, sometimes not showing up until years after aircraft are introduced into service. Malaysia Airlines Flight 124 is a case in point. Boeing's 777 jets started service in 1995 and had never experienced a similar emergency before.

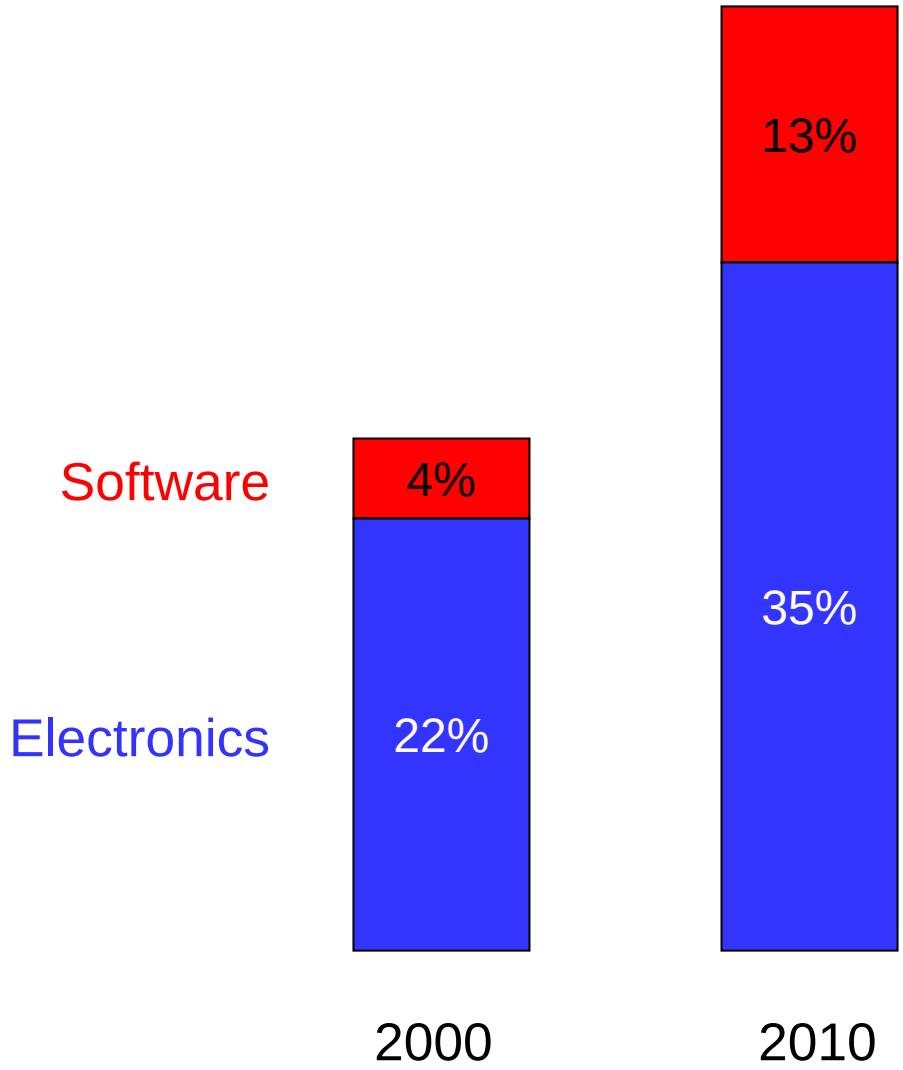
August 2005

Soon after the incident, Boeing issued a safety alert advising that, in such circumstances, pilots should immediately disconnect the autopilot and might need to exert an unusually strong force on the controls for as long as two minutes to regain normal flight.

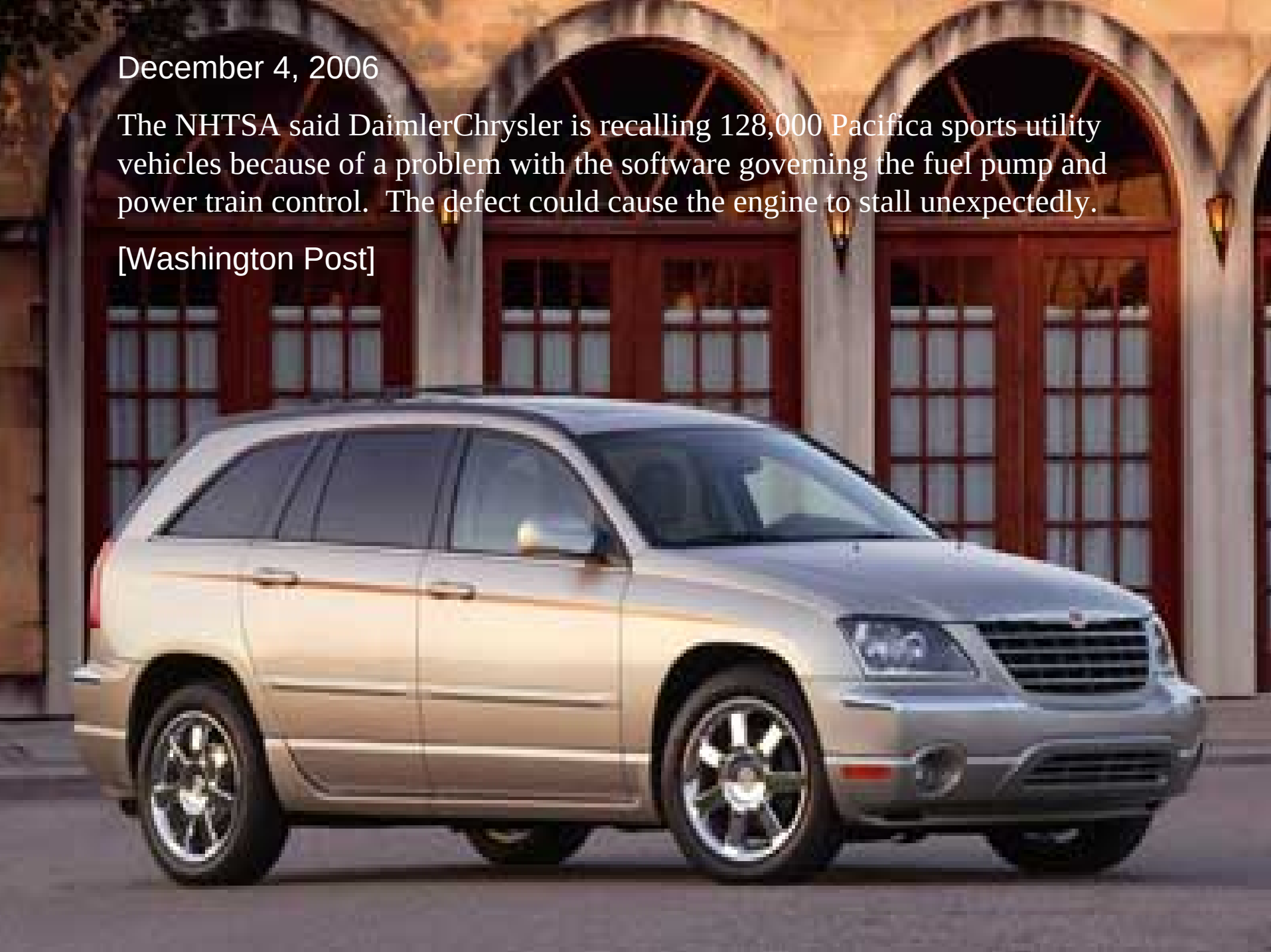
[Wall Street Journal; May 30, 2006]

500 horses
200 processors

Production Cost of Automobiles



[MIT Tech Review]



December 4, 2006

The NHTSA said DaimlerChrysler is recalling 128,000 Pacifica sports utility vehicles because of a problem with the software governing the fuel pump and power train control. The defect could cause the engine to stall unexpectedly.

[Washington Post]

It’s the Software, Stupid!

The value is in the software:

Microsoft is one of the three most valuable companies in the world.

It’s the Software, Stupid!

The value is in the software:

Microsoft is one of the three most valuable companies in the world.

The bugs are in the software:

What is more likely to crash: your modem or your browser?

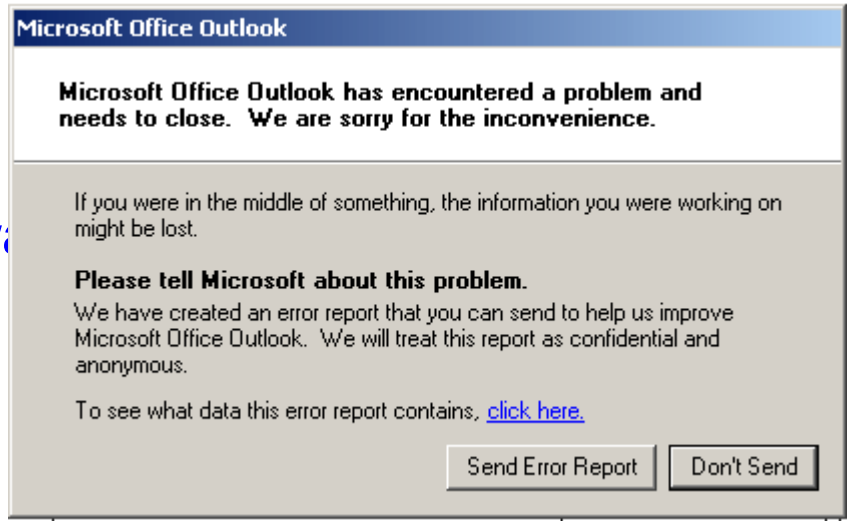
It’s the Software, Stupid!

The value is in the software:

Microsoft is one of the three

The bugs are in the softw

What is more likely to crash:



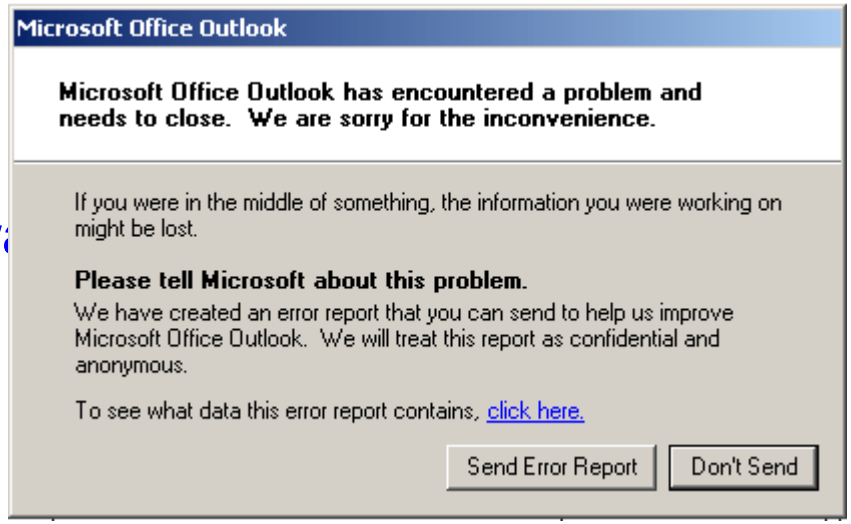
It’s the Software, Stupid!

The value is in the software:

Microsoft is one of the three

The bugs are in the softw

What is more likely to crash:



It’s the Software, Stupid!

The value is in the software:

Microsoft is one of the three most valuable companies in the world.

The bugs are in the software:

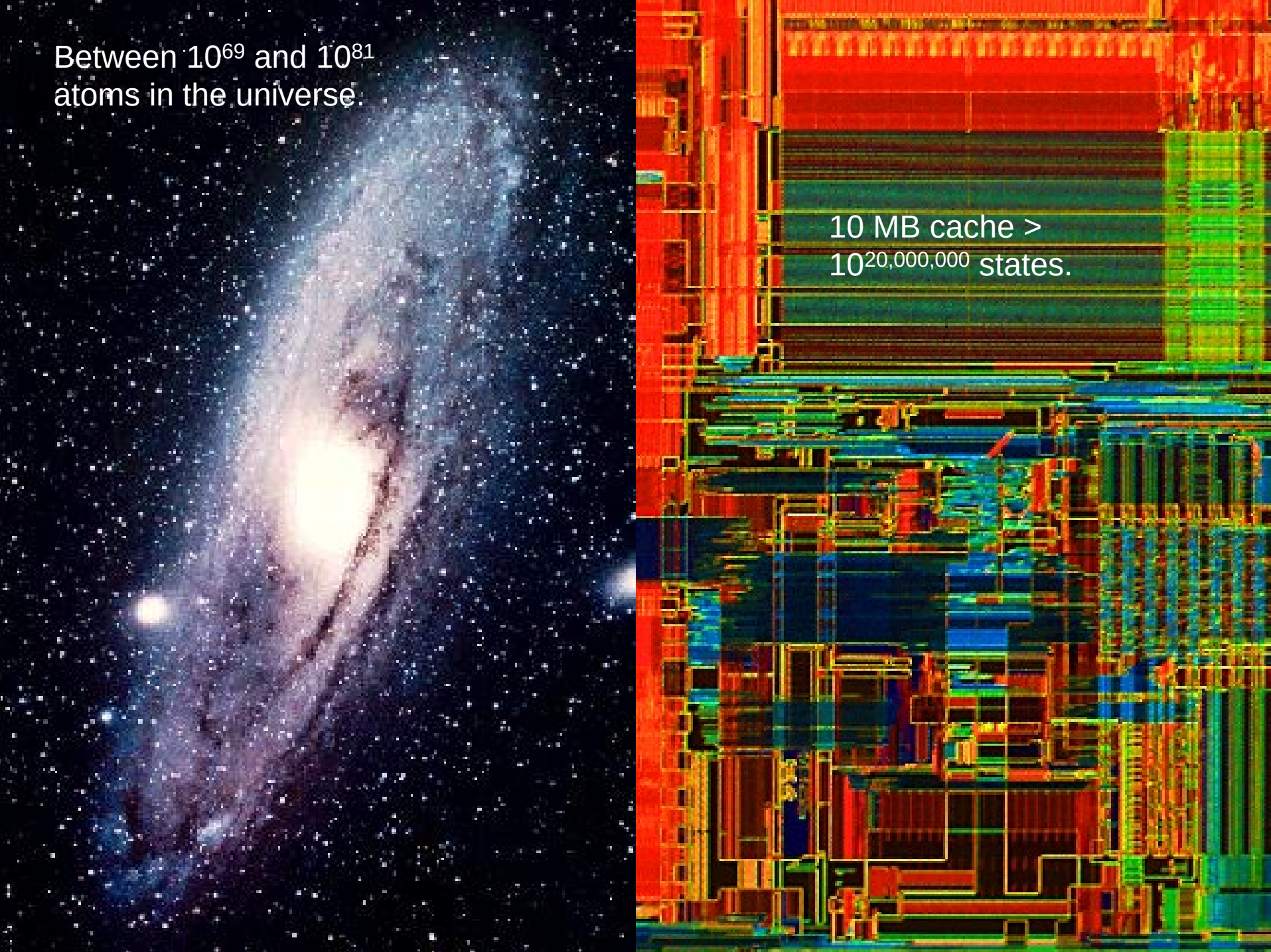
What is more likely to crash: your modem or your browser?

The challenges are in the software:

Is it that no smart people go into software engineering,
or is building software really *that* difficult?

Software truly is the most complex artifact we build routinely.
It’s not surprising we rarely get it right.


Between 10^{69} and 10^{81}
atoms in the universe.


Between 10^{69} and 10^{81} atoms in the universe.
10 MB cache > $10^{20,000,000}$ states.

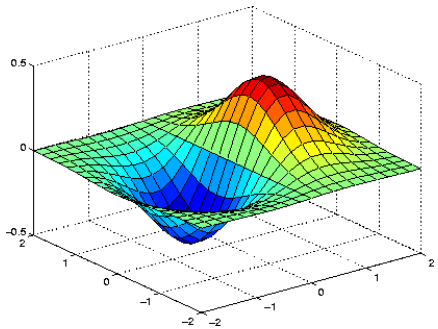
Complexity Management in Engineering

Build & test

System

Bridge
Aircraft
etc.

Complexity Management in Engineering



Calculate

Model

Abstract

Predict

System

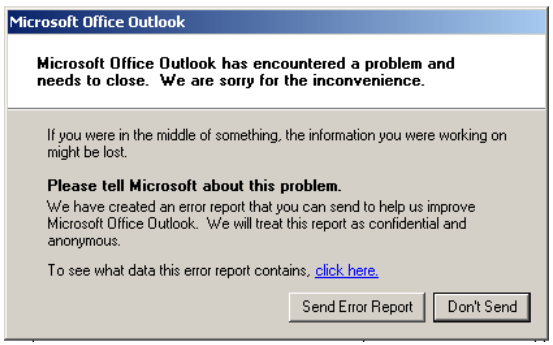
Applied Mathematics

Build & test

Bridge
Aircraft
etc.

Uptime: 123 years

Complexity Management in Engineering



Build & test

System

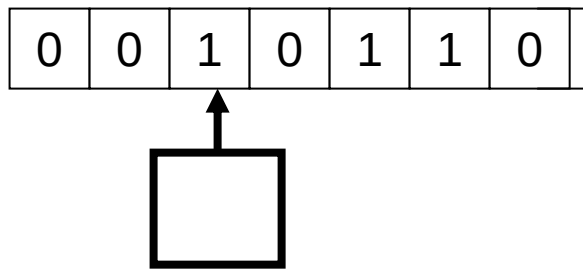
Bridge
Aircraft
Software

So, why don’t we have a mathematics for building software?

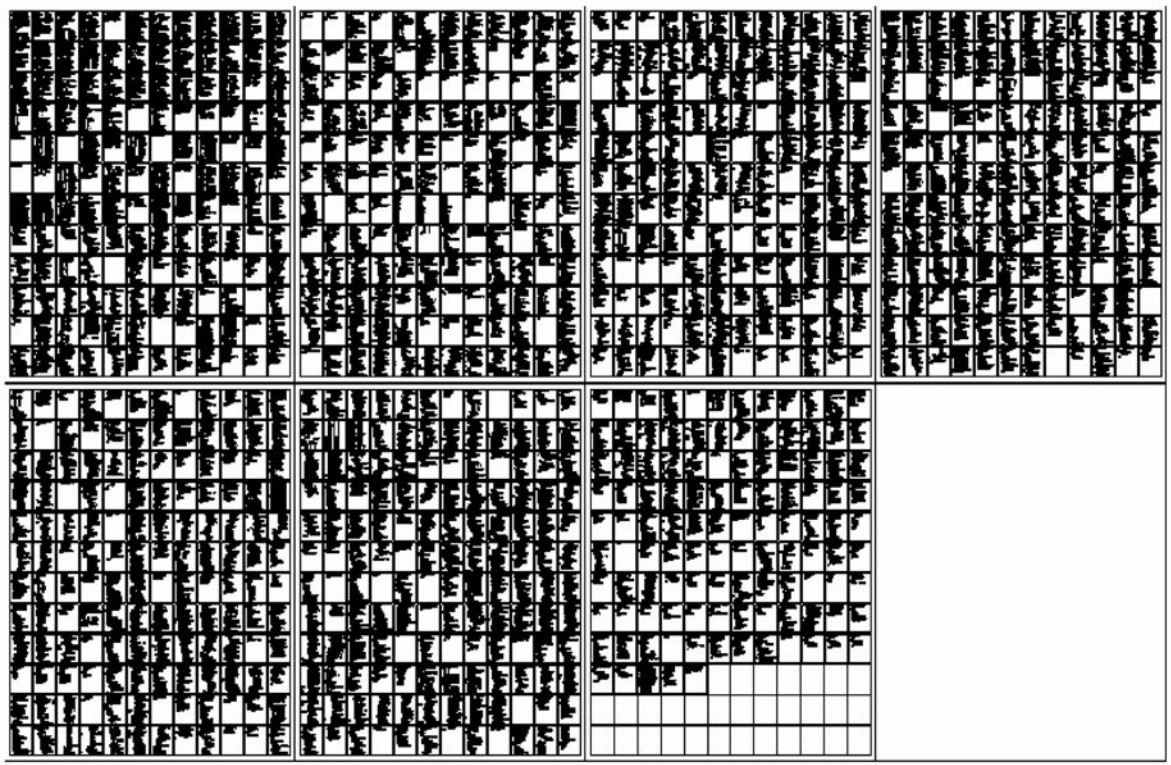
So, why don't we have a mathematics for building software?

We do, but it's not continuous.

-sensitive against perturbations
-difficult to overengineer
-difficult to abstract

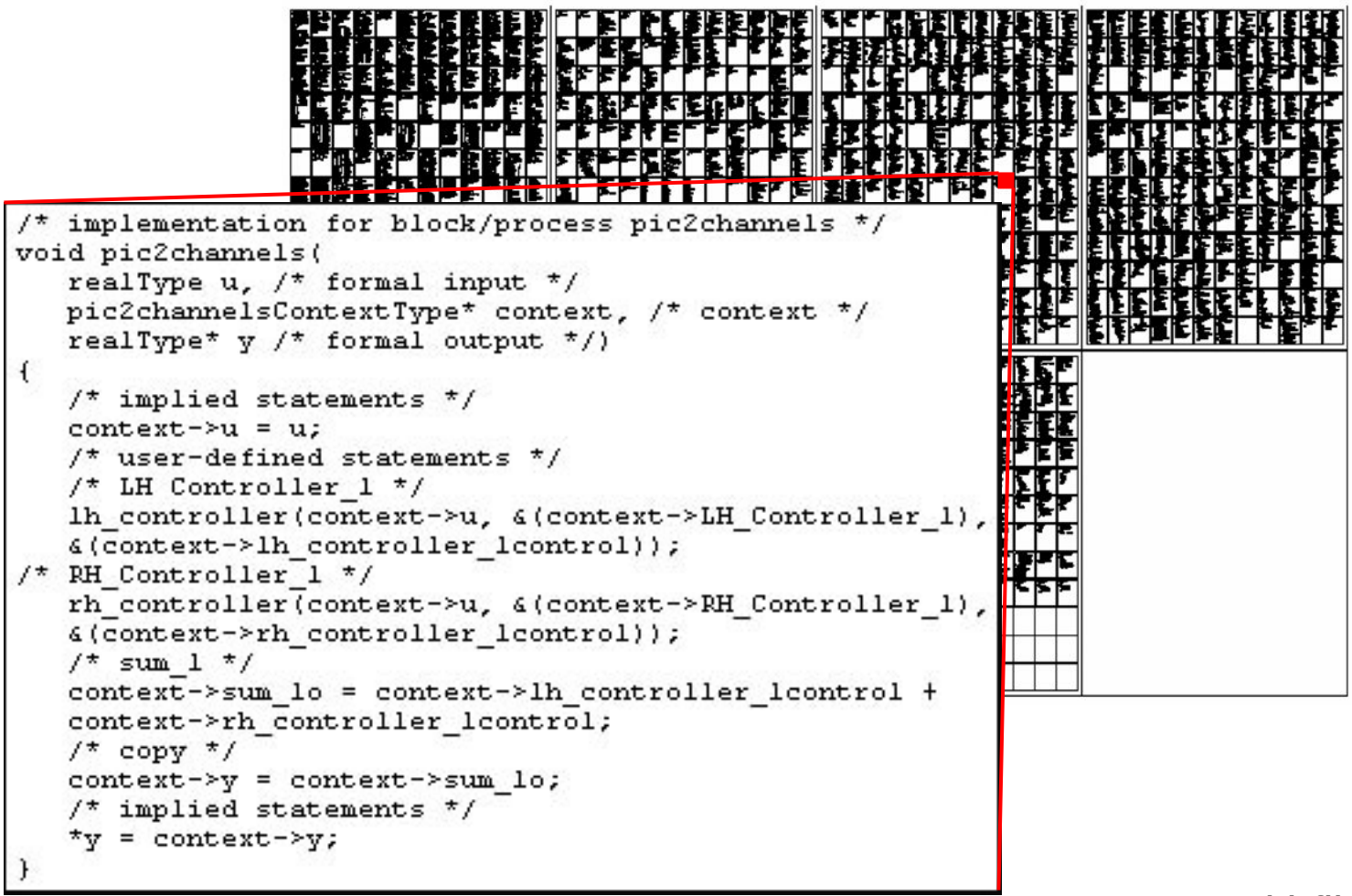


A Program



kbfilter.c
12,000 lines of code

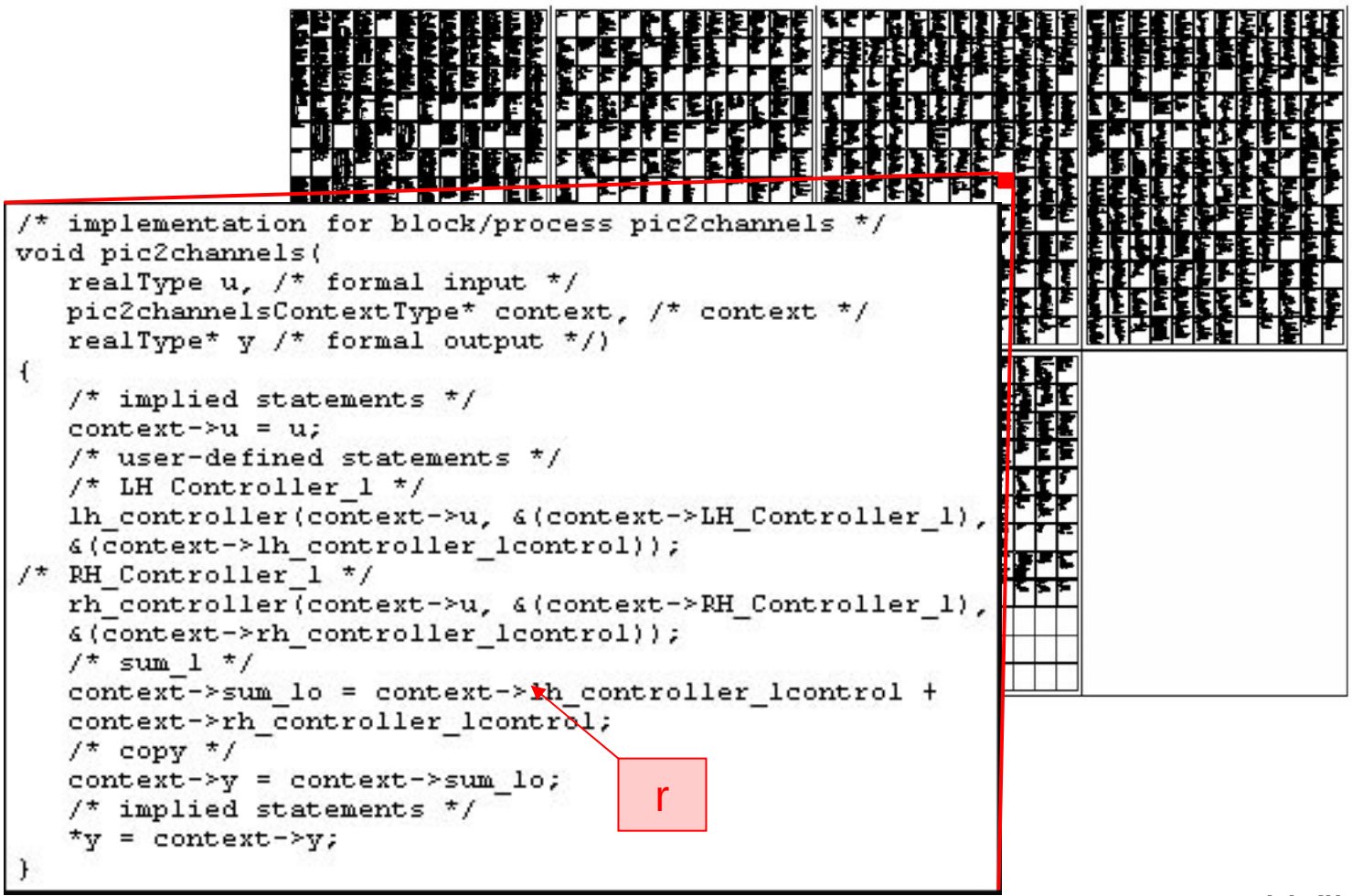
A Program



```
/* implementation for block/process pic2channels */
void pic2channels(
   realType u, /* formal input */
   pic2channelsContextType* context, /* context */
   realType* y /* formal output */)
{
   /* implied statements */
   context->u = u;
   /* user-defined statements */
   /* LH Controller_1 */
   lh_controller(context->u, &(context->LH_Controller_1),
   &(context->lh_controller_1control));
/* RH_Controller_1 */
   rh_controller(context->u, &(context->RH_Controller_1),
   &(context->rh_controller_1control));
   /* sum_1 */
   context->sum_1o = context->lh_controller_1control +
   context->rh_controller_1control;
   /* copy */
   context->y = context->sum_1o;
   /* implied statements */
   *y = context->y;
}
```

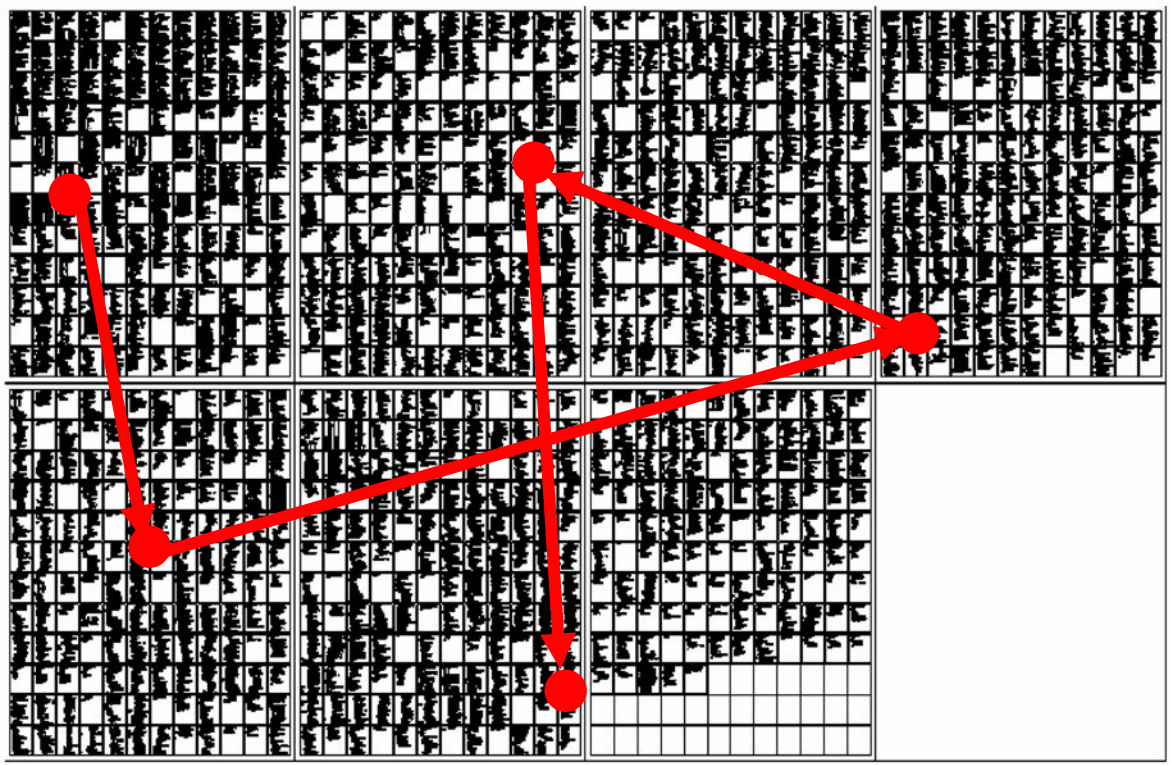
kbfilter.c
12,000 lines of code

A Program



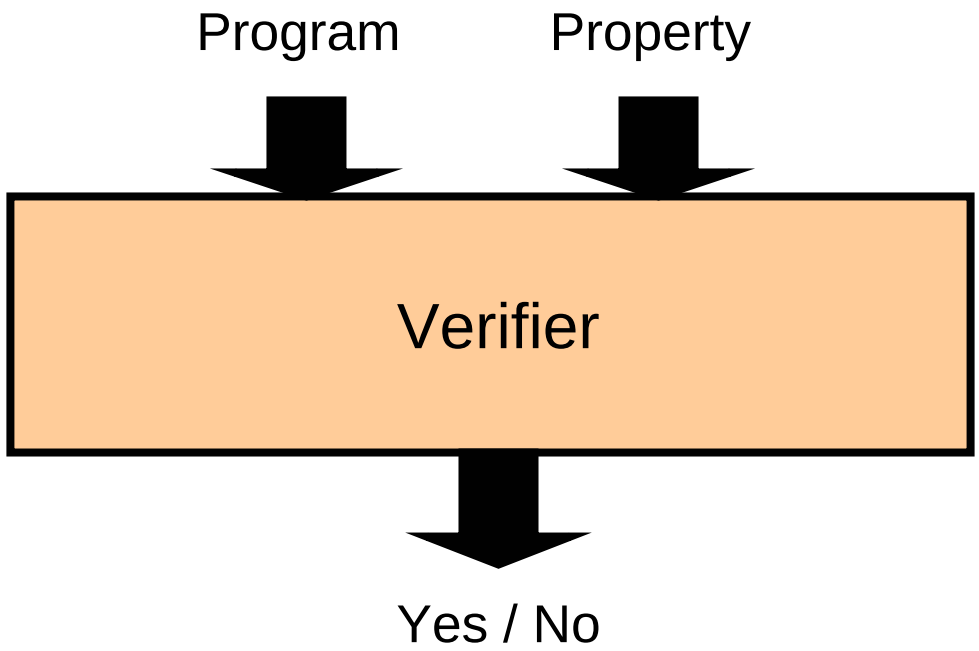
kbfilter.c
12,000 lines of code

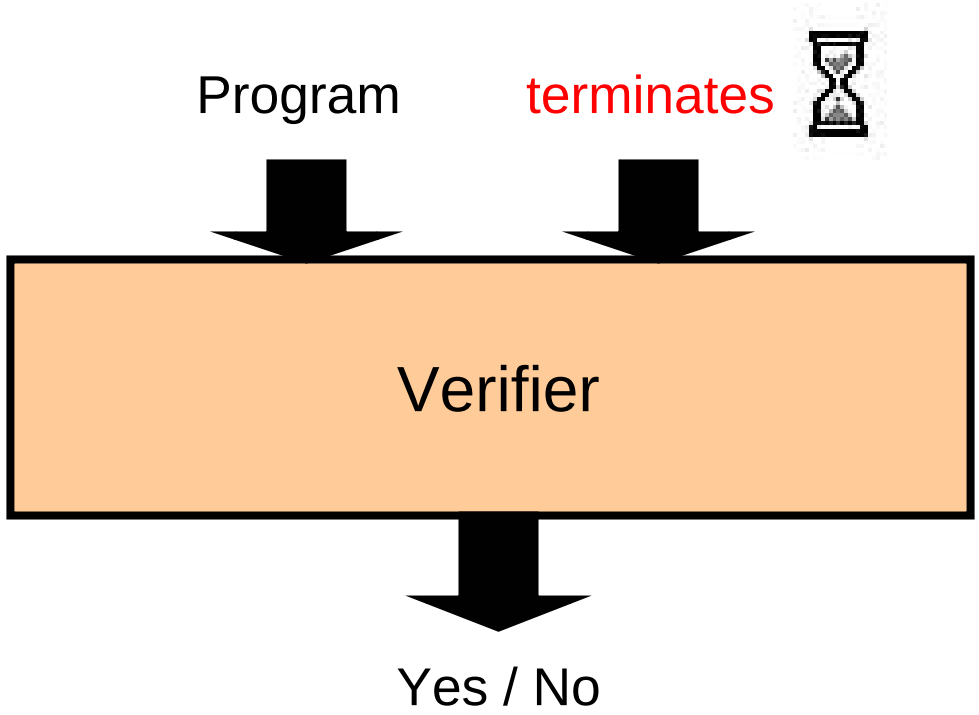
An Error Trajectory

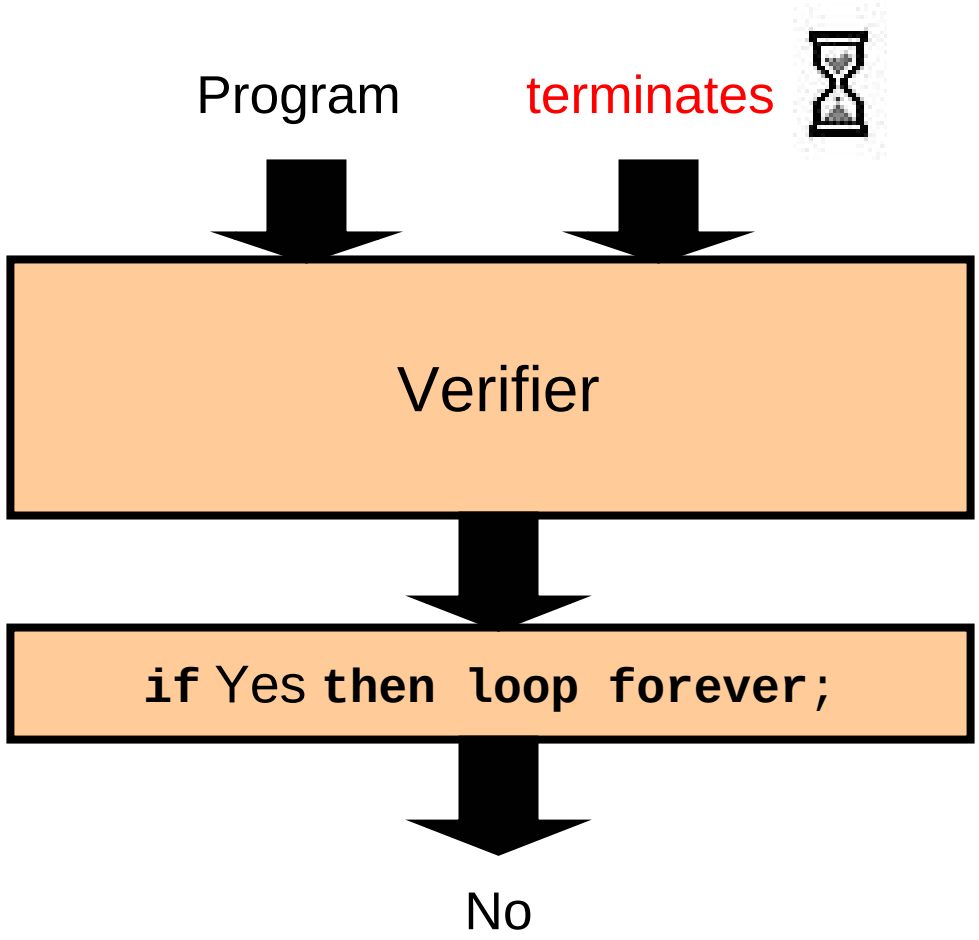


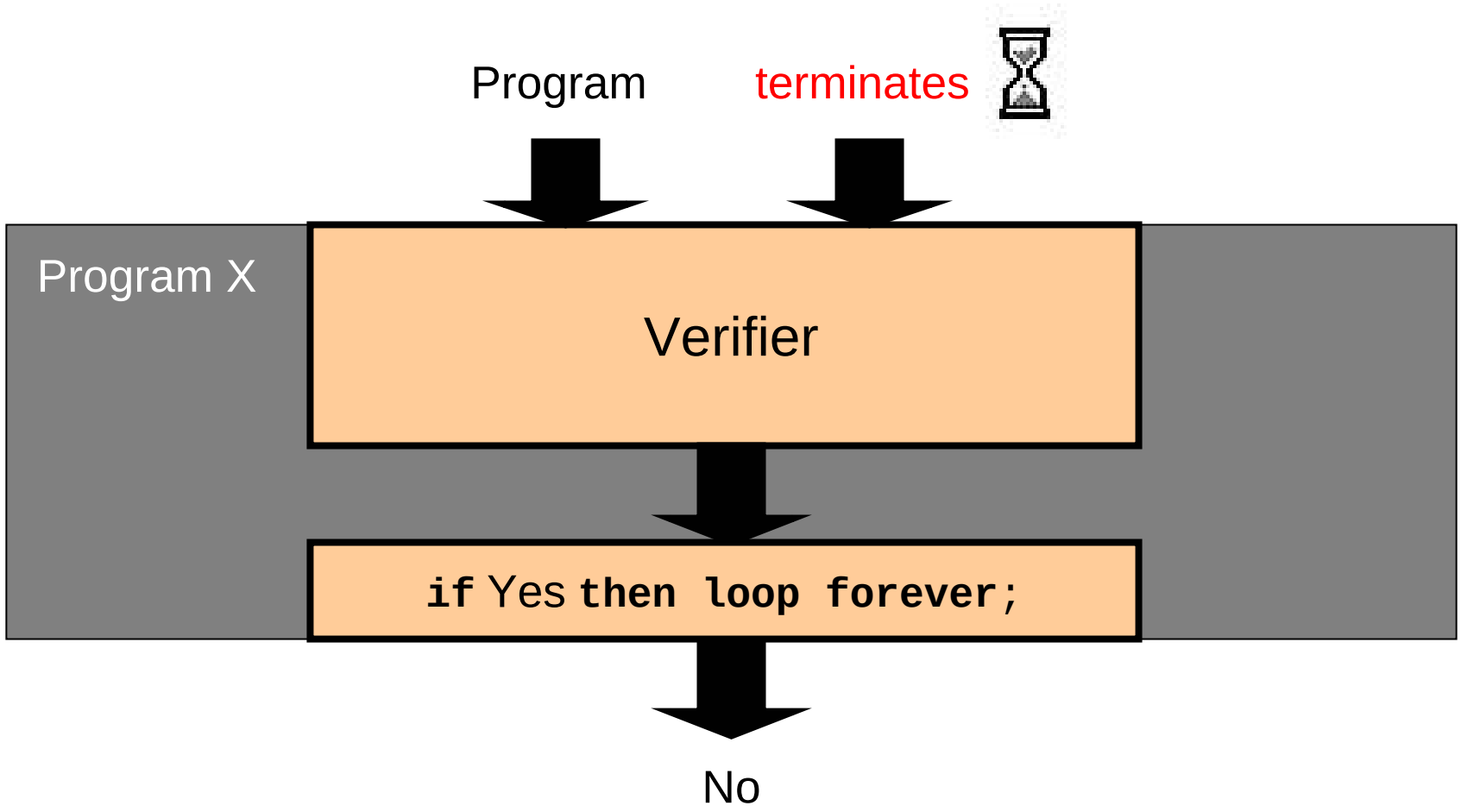
Programs are not continuous.

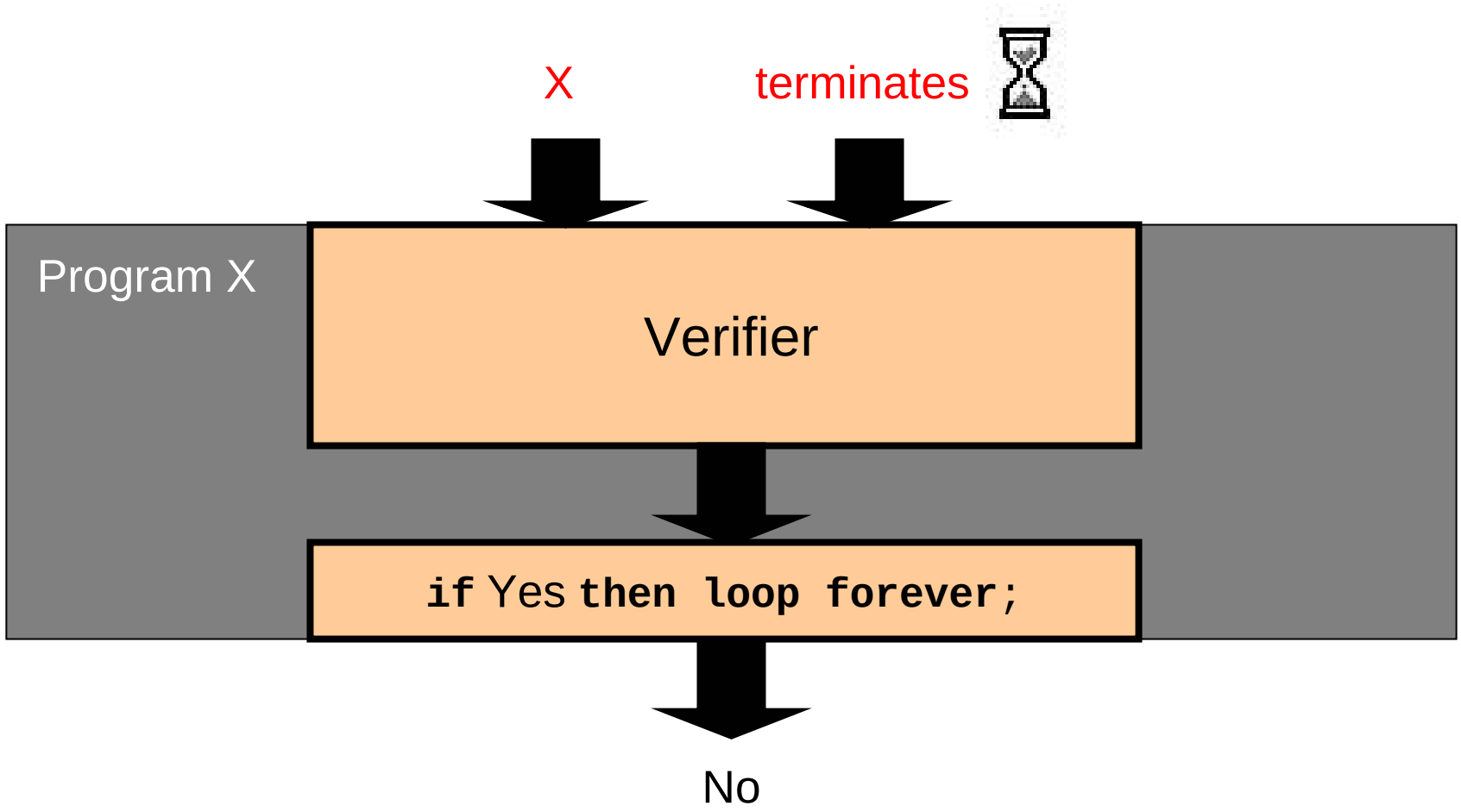
The Impossible Dream



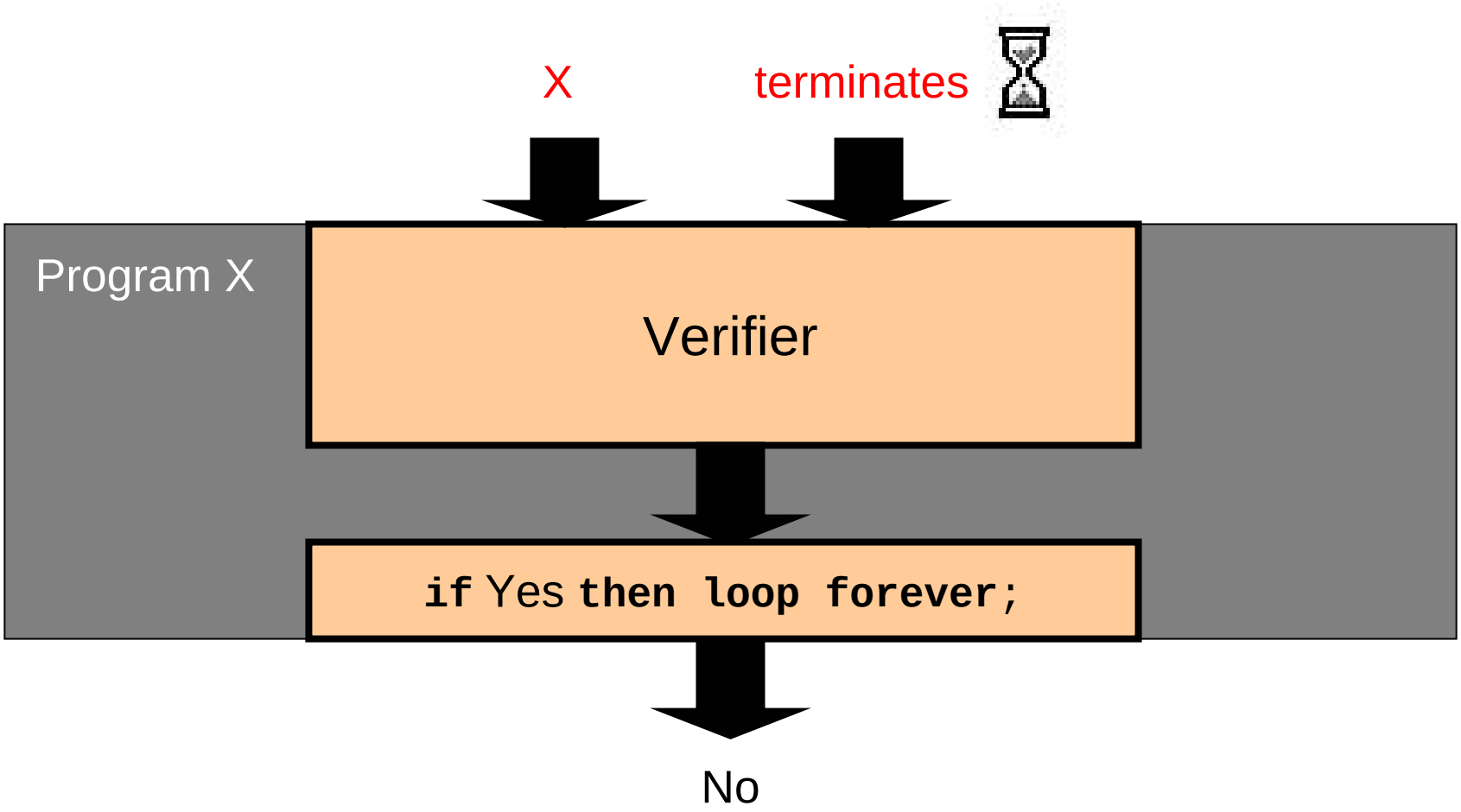

Program
terminates
Verifier
Yes / No


Program
terminates
Verifier
if Yes then loop forever;
No

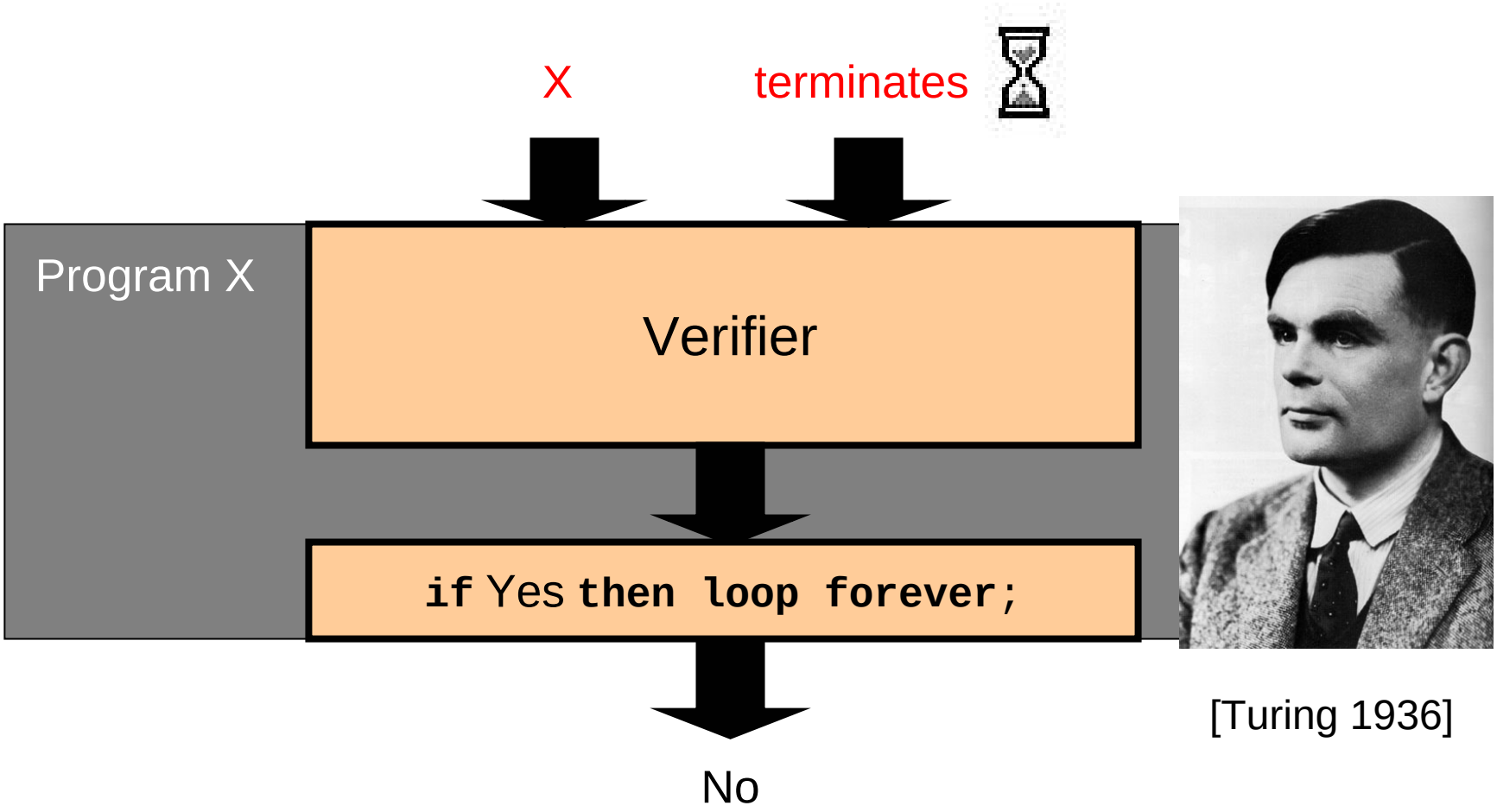

Program
terminates
Program X
Verifier
if Yes then loop forever;
No


X
terminates
Program X
Verifier
if Yes then loop forever;
No

Does X terminate?

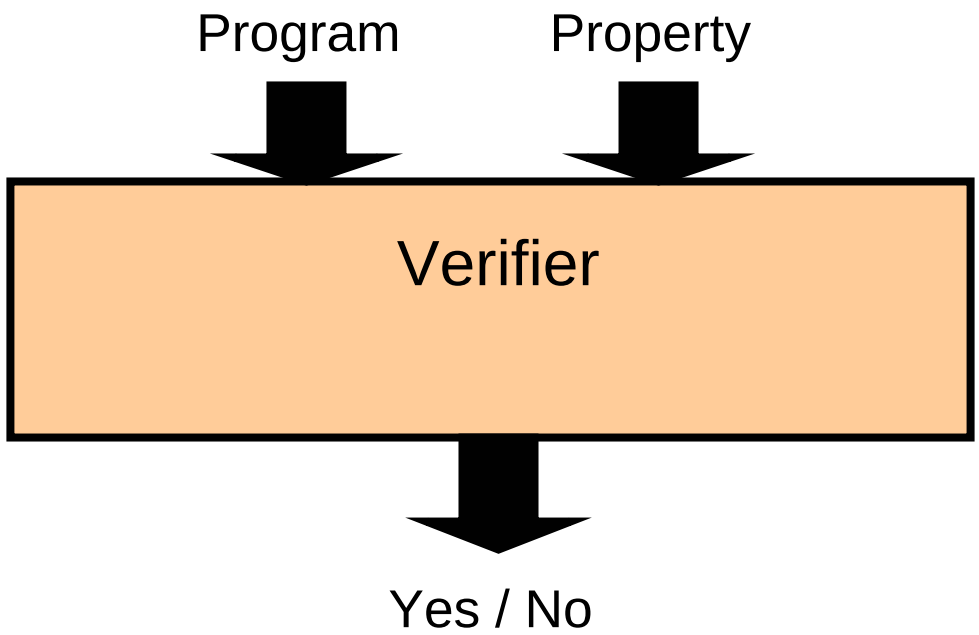


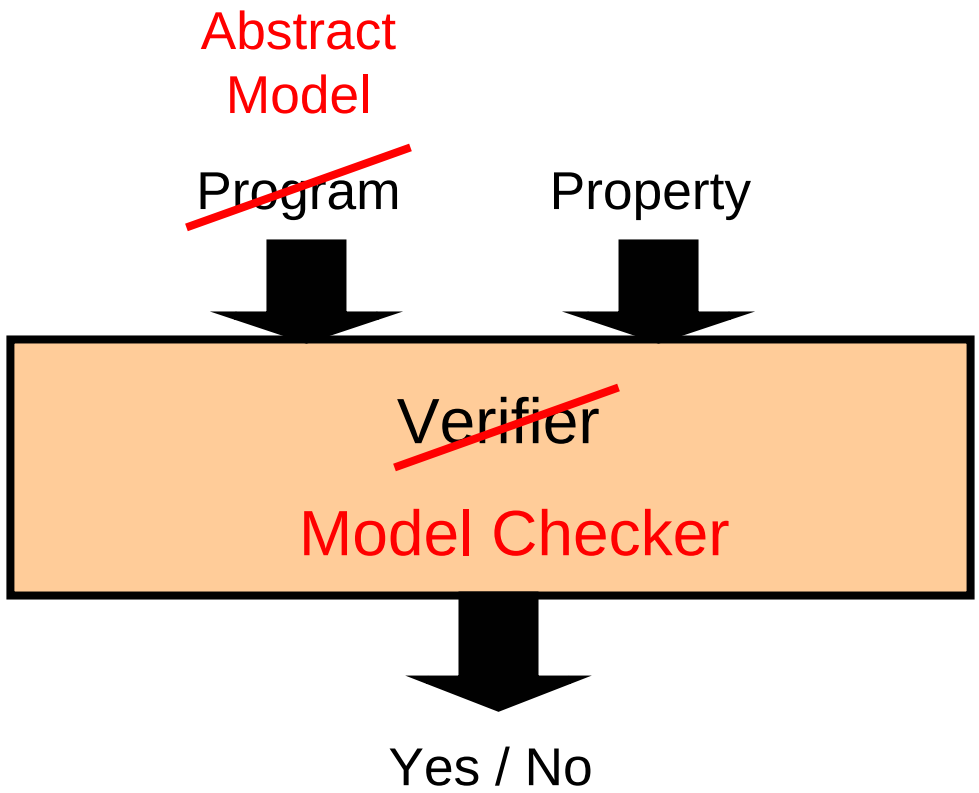
Conclusion: Verifier cannot exist!



[Turing 1936]

The Impossible Dream




Abstract
Model
Program
Property
Verifier
Model Checker
Yes / No

Mathematical Modeling:
A Tale of Two Cultures

Engineering	Computer Science
Differential Equations	Mathematical Logic
Linear Algebra	Discrete Structures
Probability Theory	Automata Theory

Our Methodology

1. Model Building:

   capture relevant aspects of the system formally (using logic and automata)

2. Model Checking:

   implement algorithms for model analysis [Clarke/Emerson; Queille/Sifakis 1981]

Two Examples

1 Concurrency Bugs

(due to interaction between two programs)

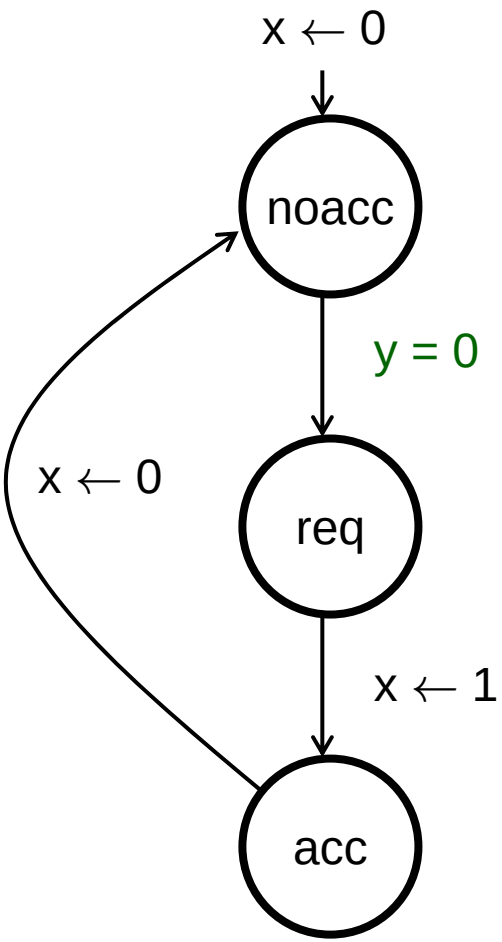
Model:	finite automata
Algorithm:	state graph exploration

2 Embedded Bugs

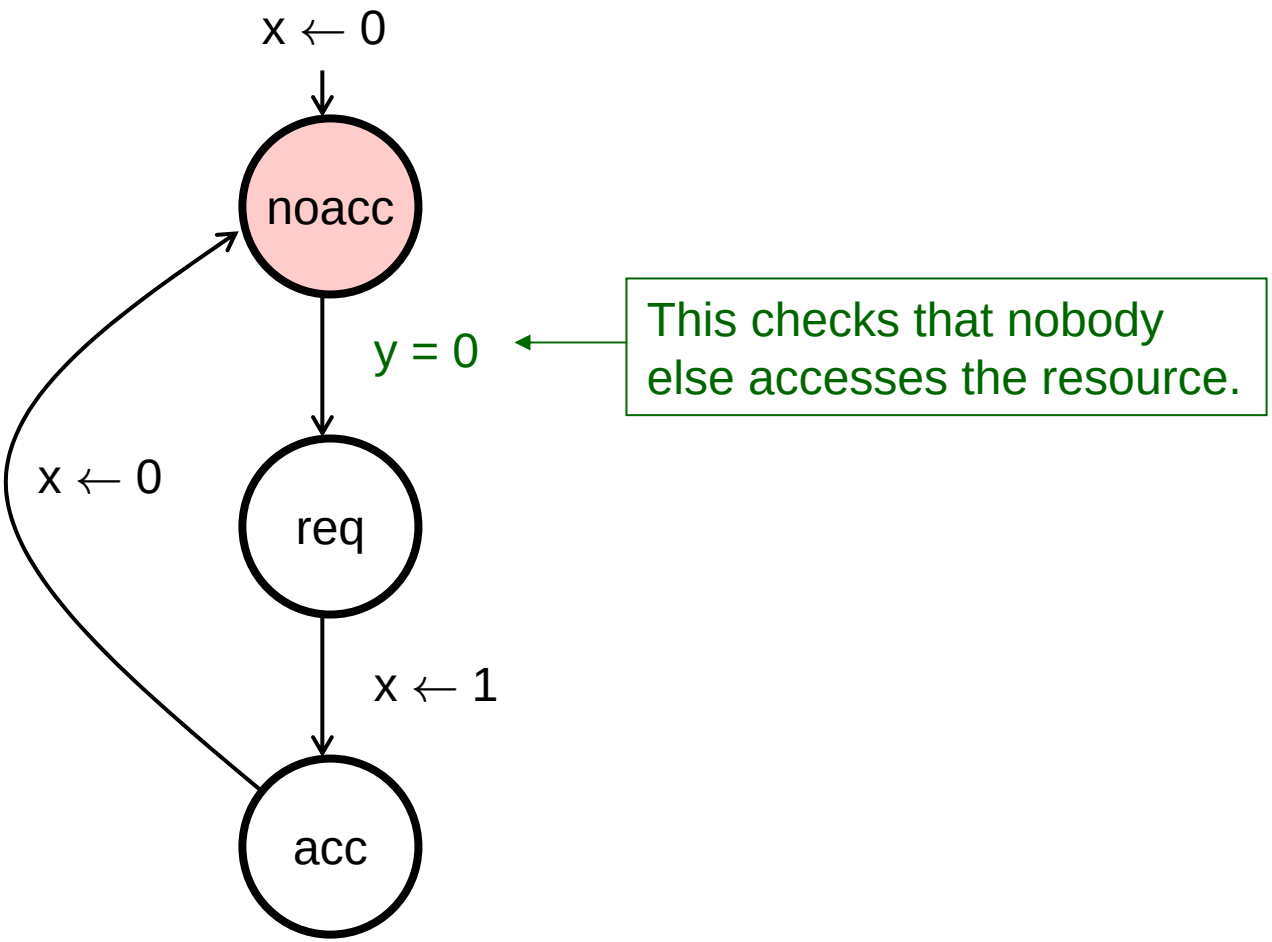
(due to interaction between a program and the physical world)

Model:	hybrid automata
Algorithm:	polyhedral state exploration

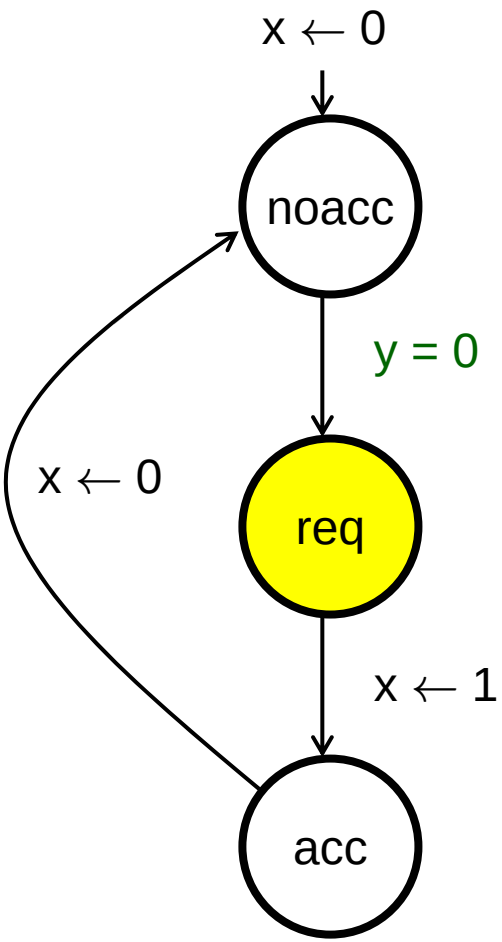
Access to a Shared Resource



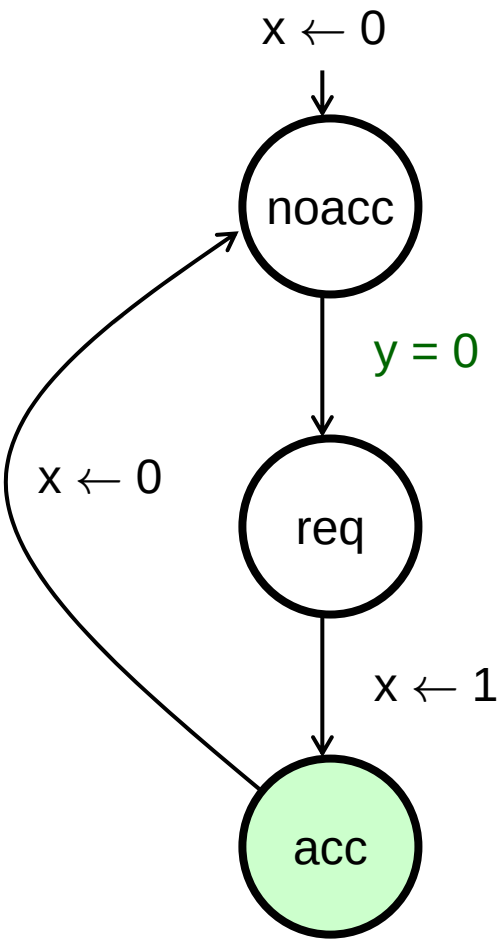
Access to a Shared Resource



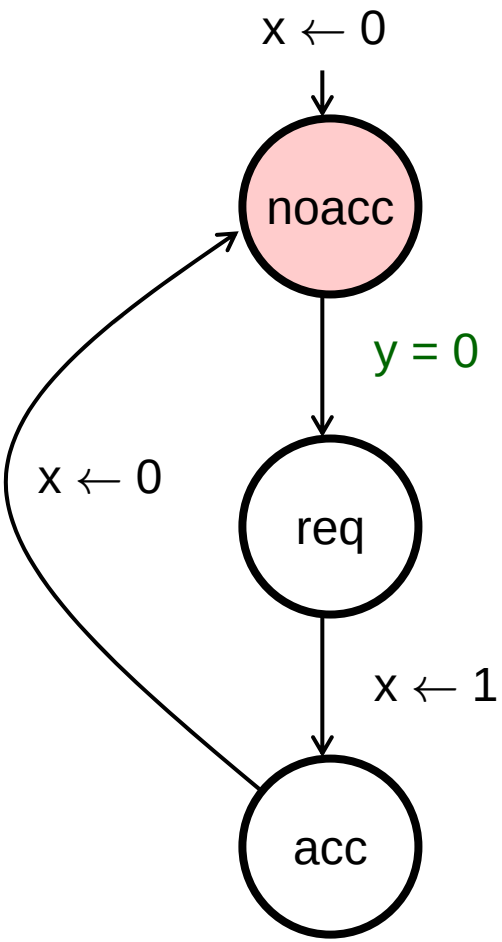
Access to a Shared Resource



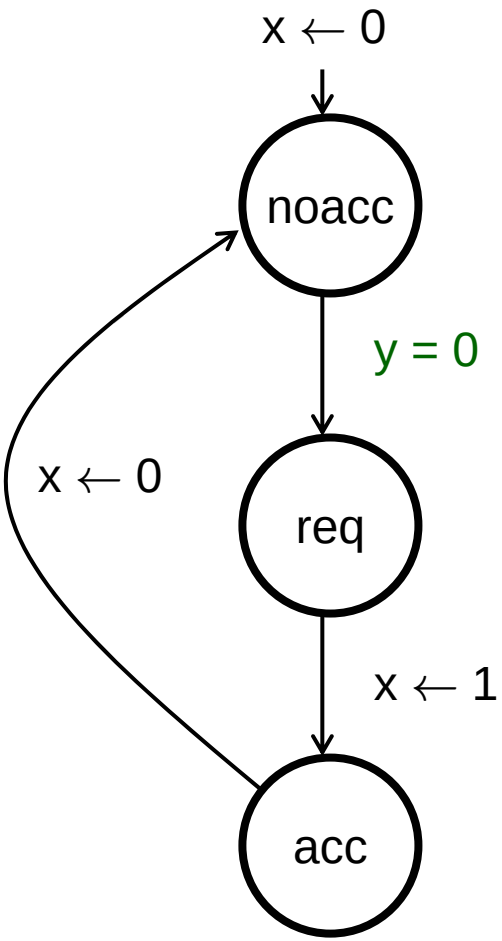
Access to a Shared Resource

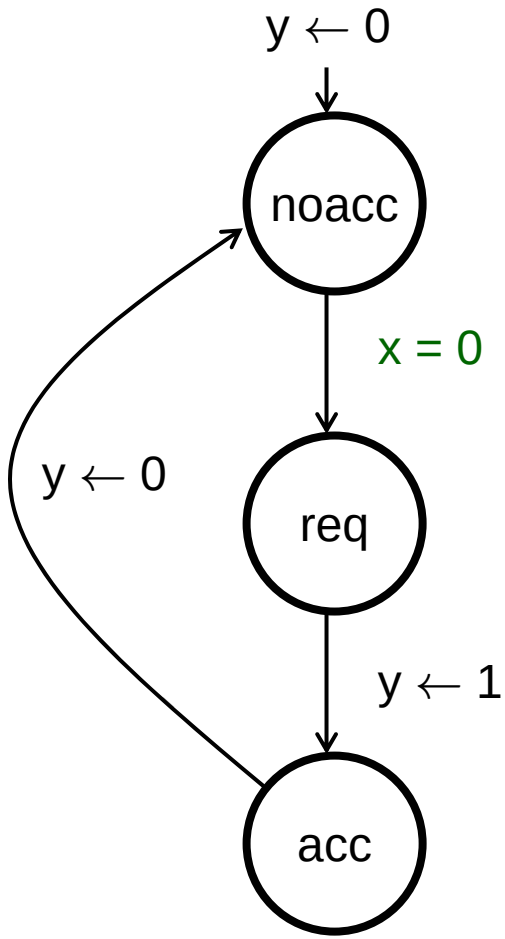


Access to a Shared Resource

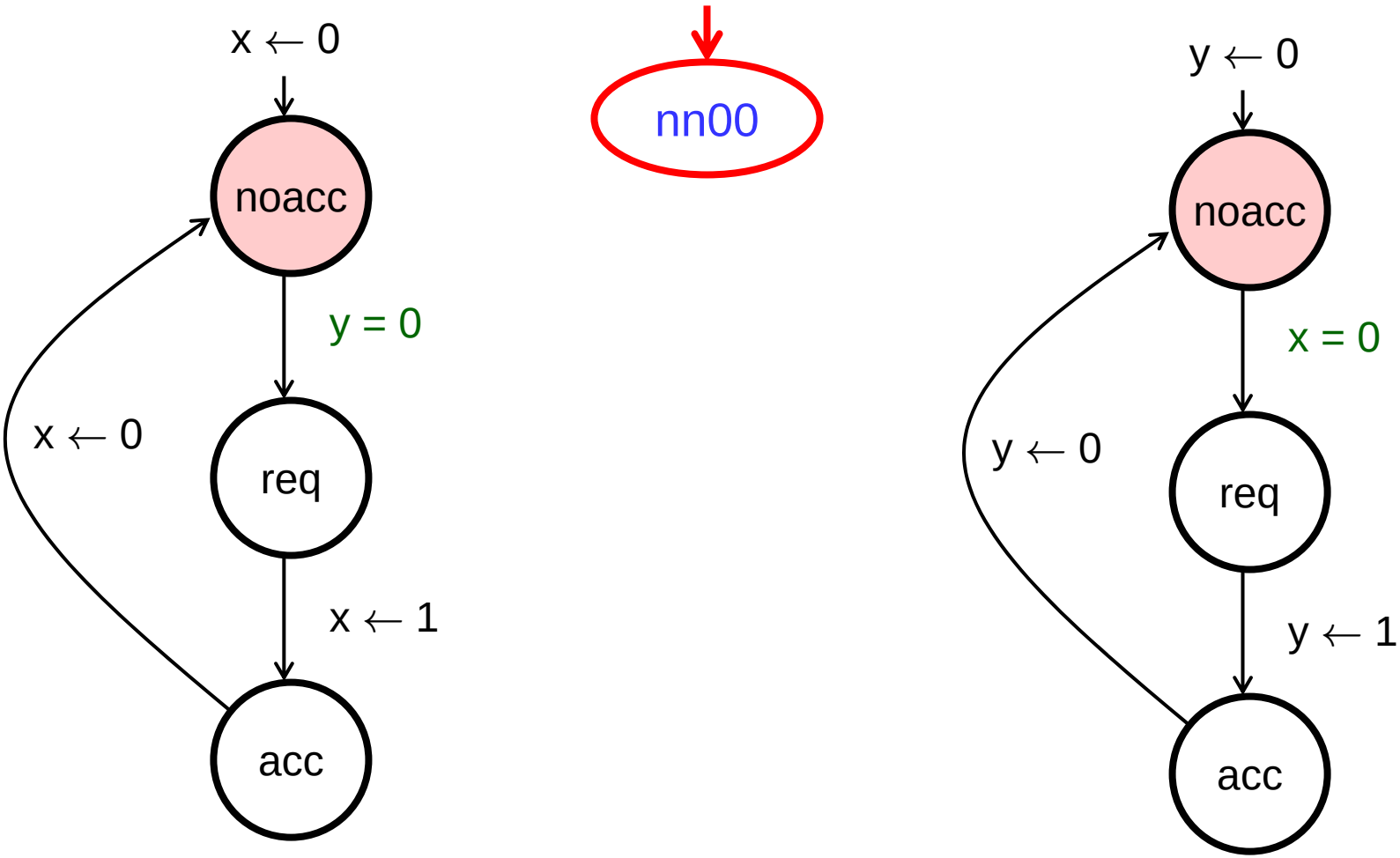


Access to a Shared Resource

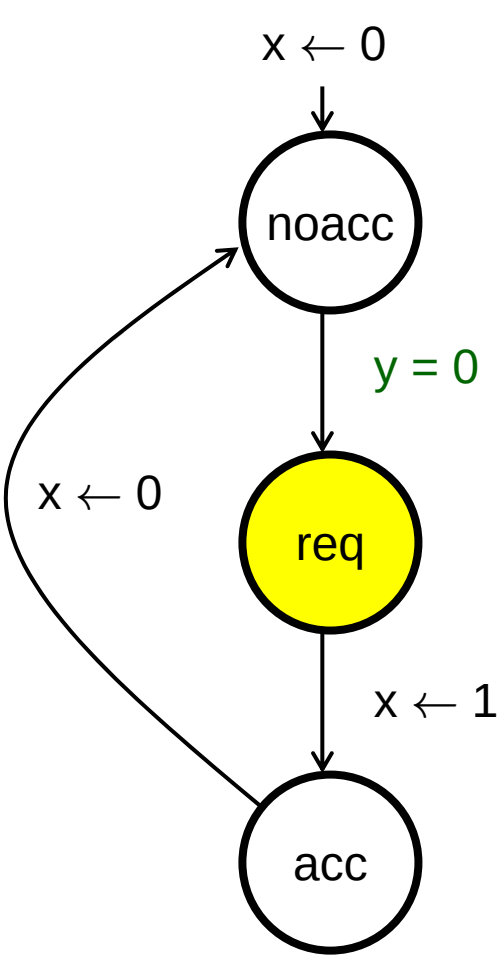


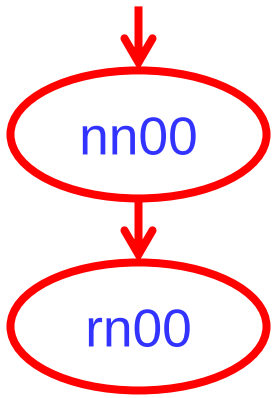


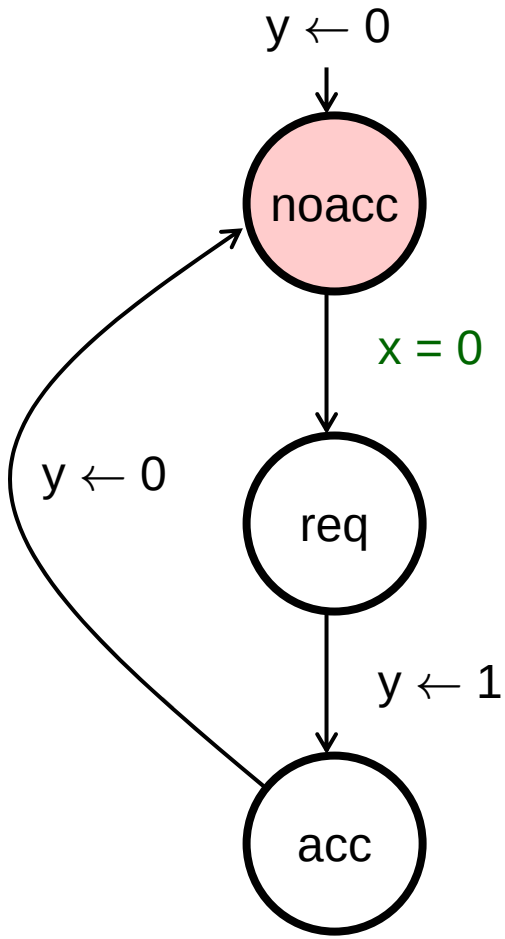
Access to a Shared Resource



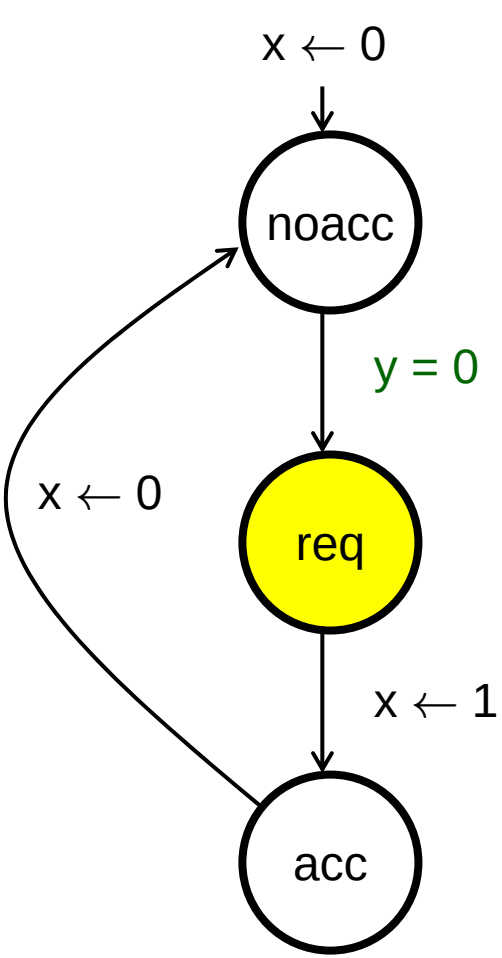
Access to a Shared Resource

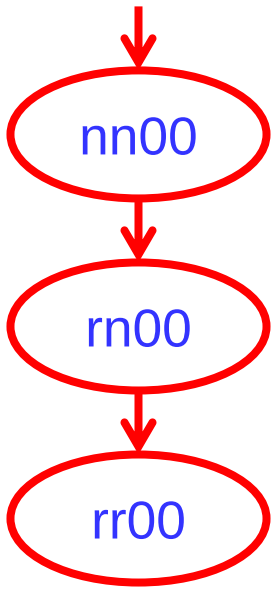


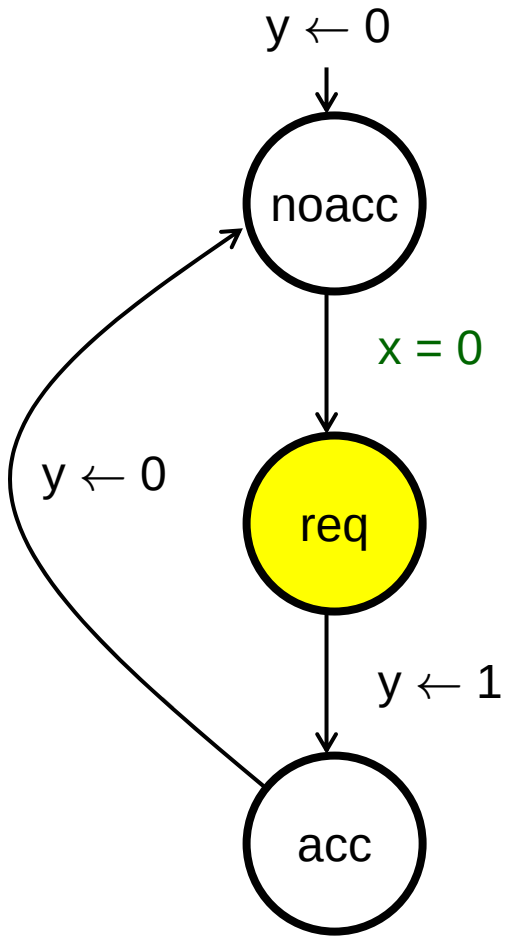




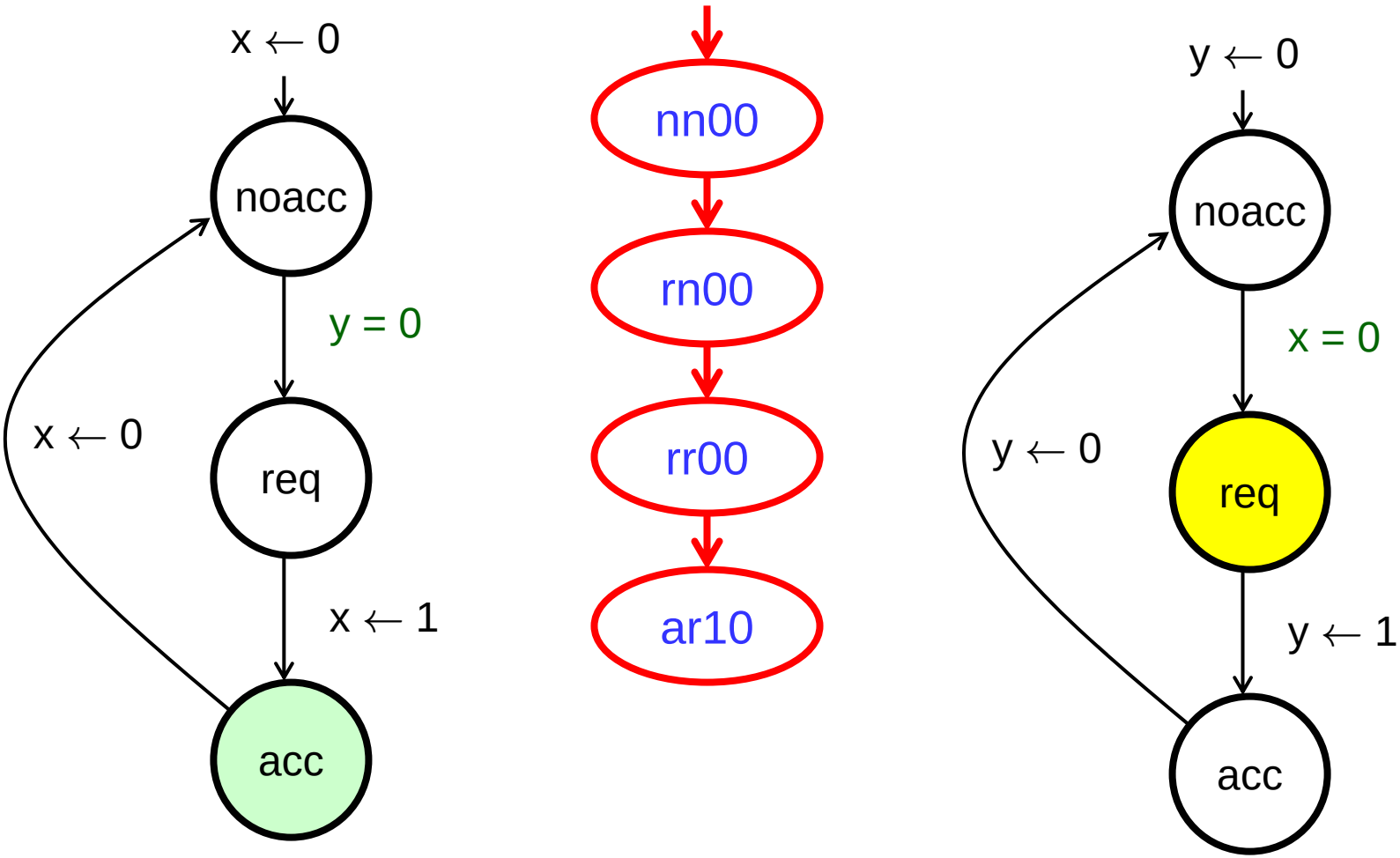
Access to a Shared Resource



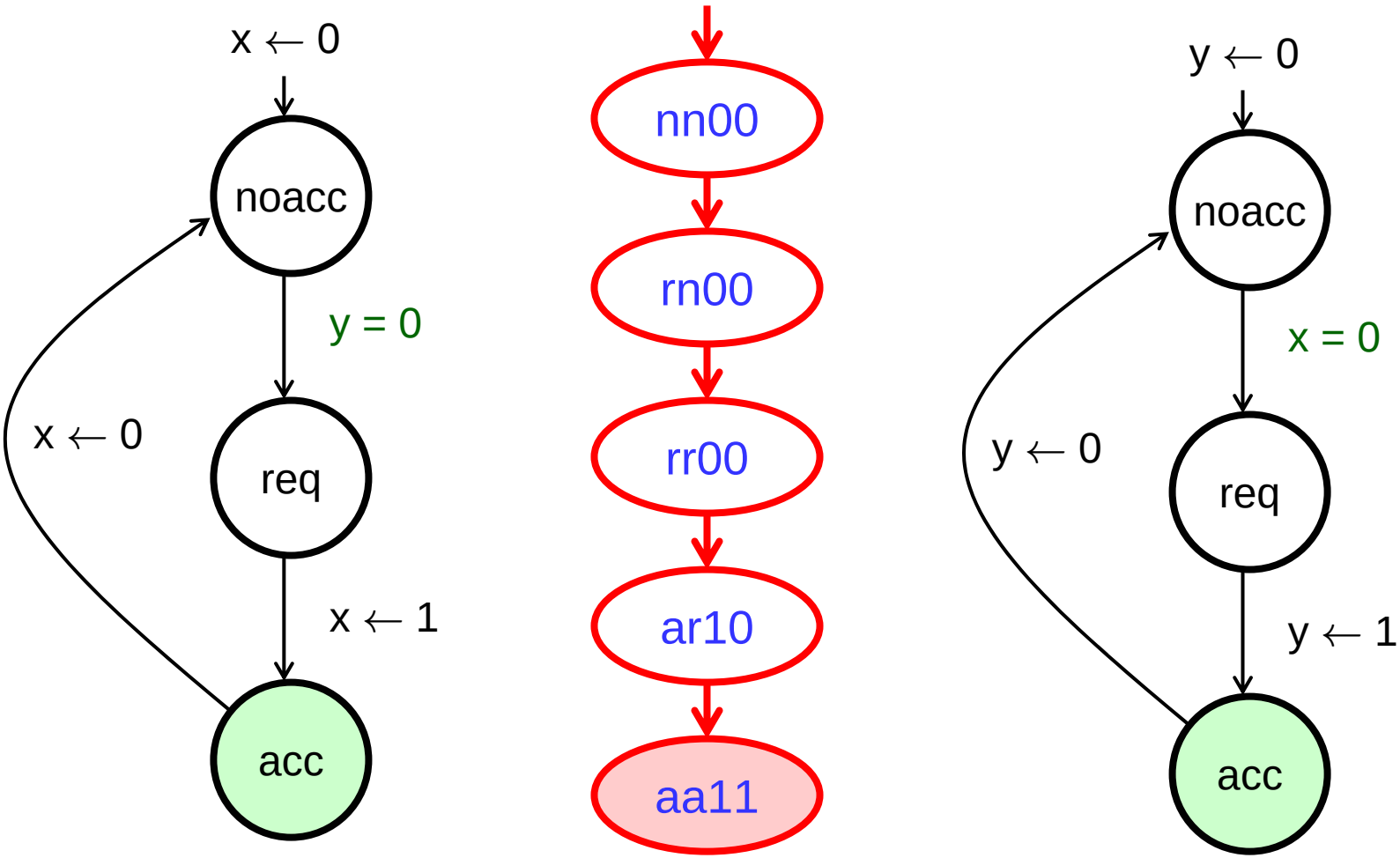




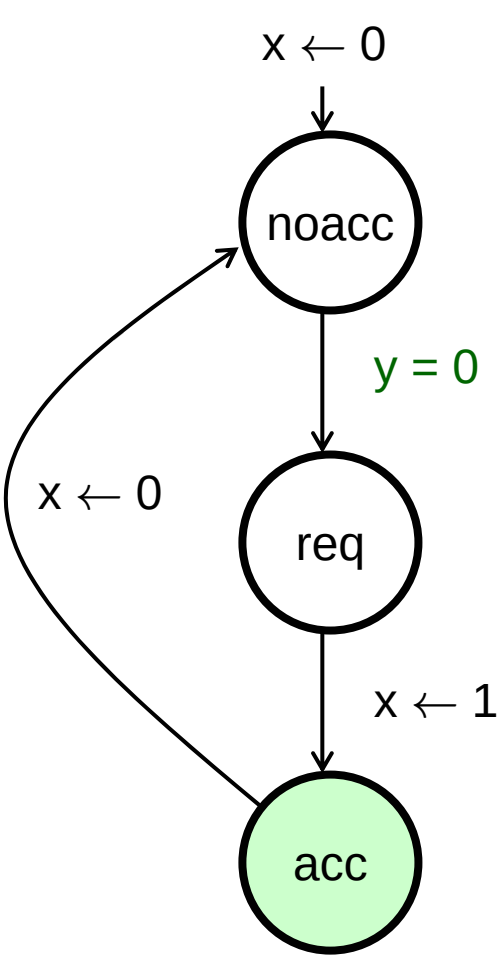
Access to a Shared Resource

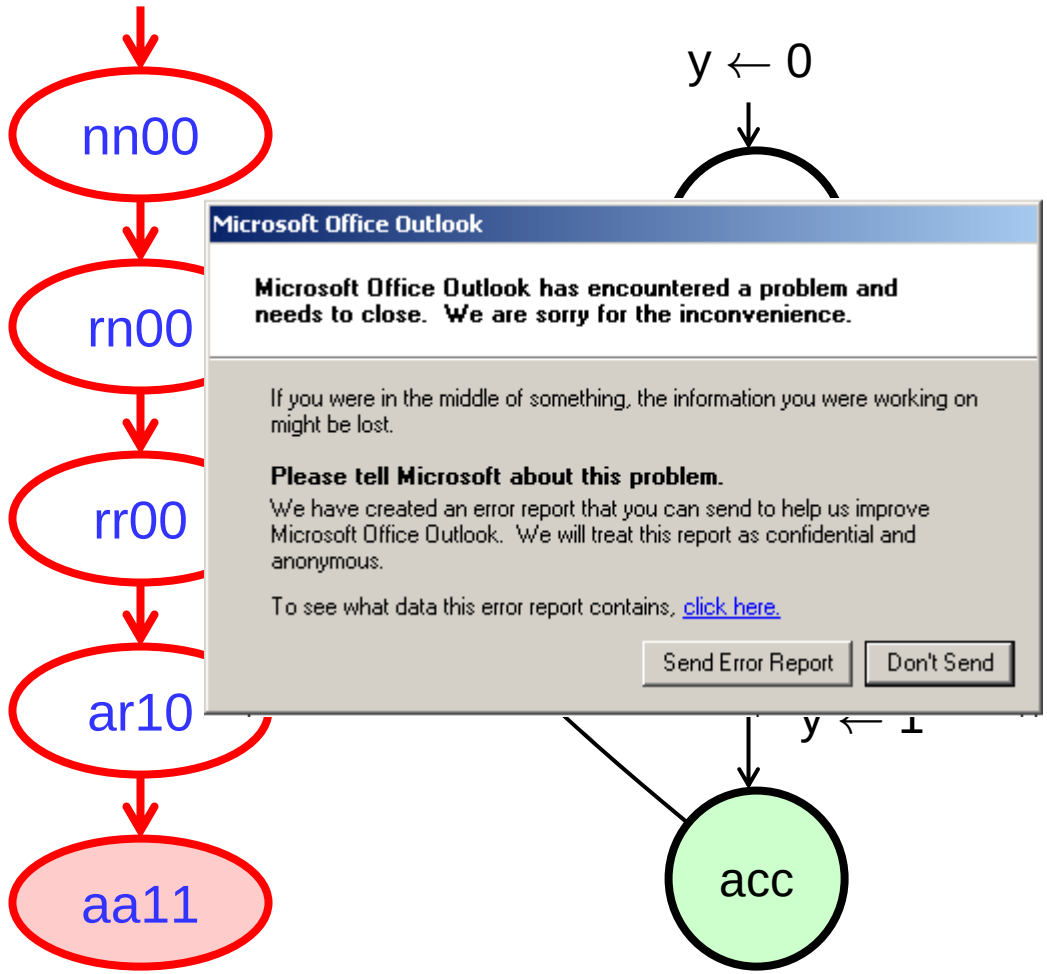


Access to a Shared Resource



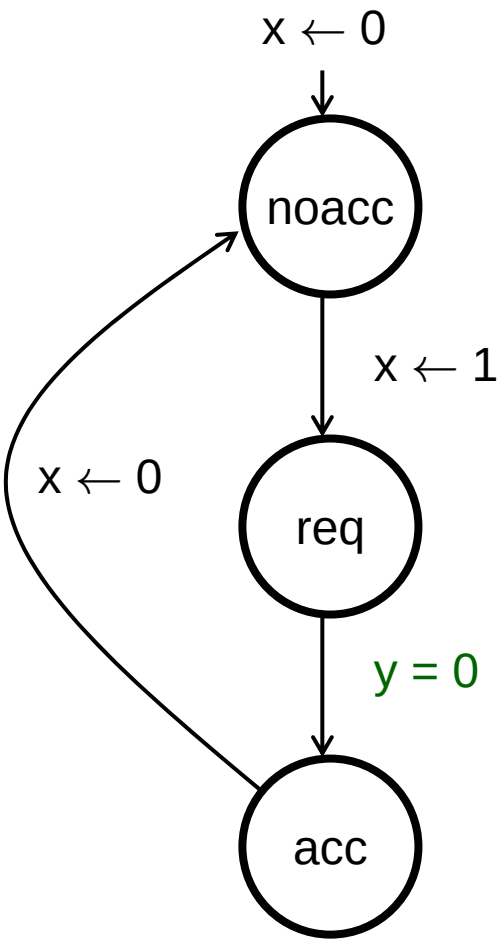
Access to a Shared Resource



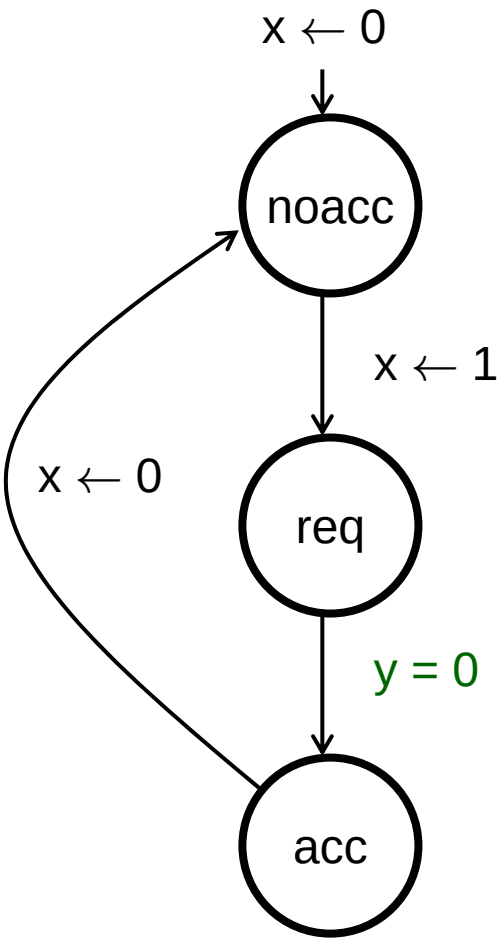


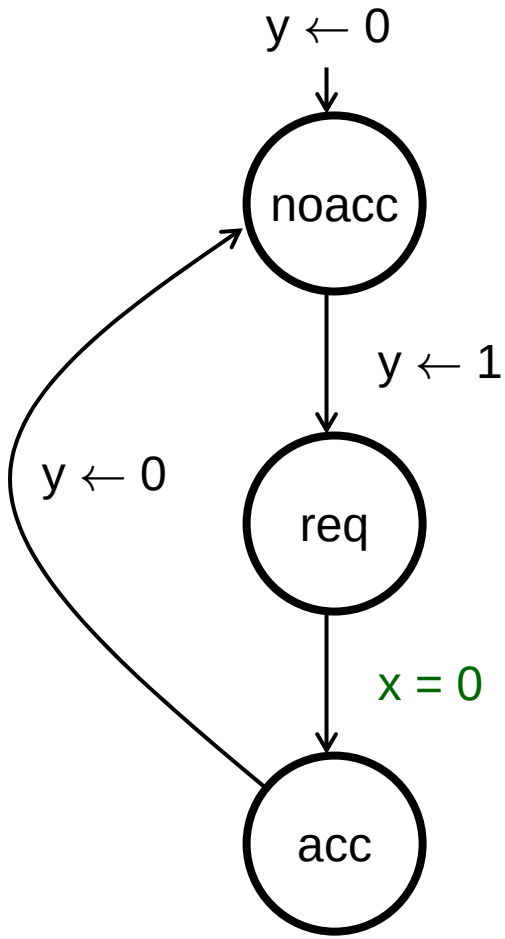
Violates safety.

Second Attempt

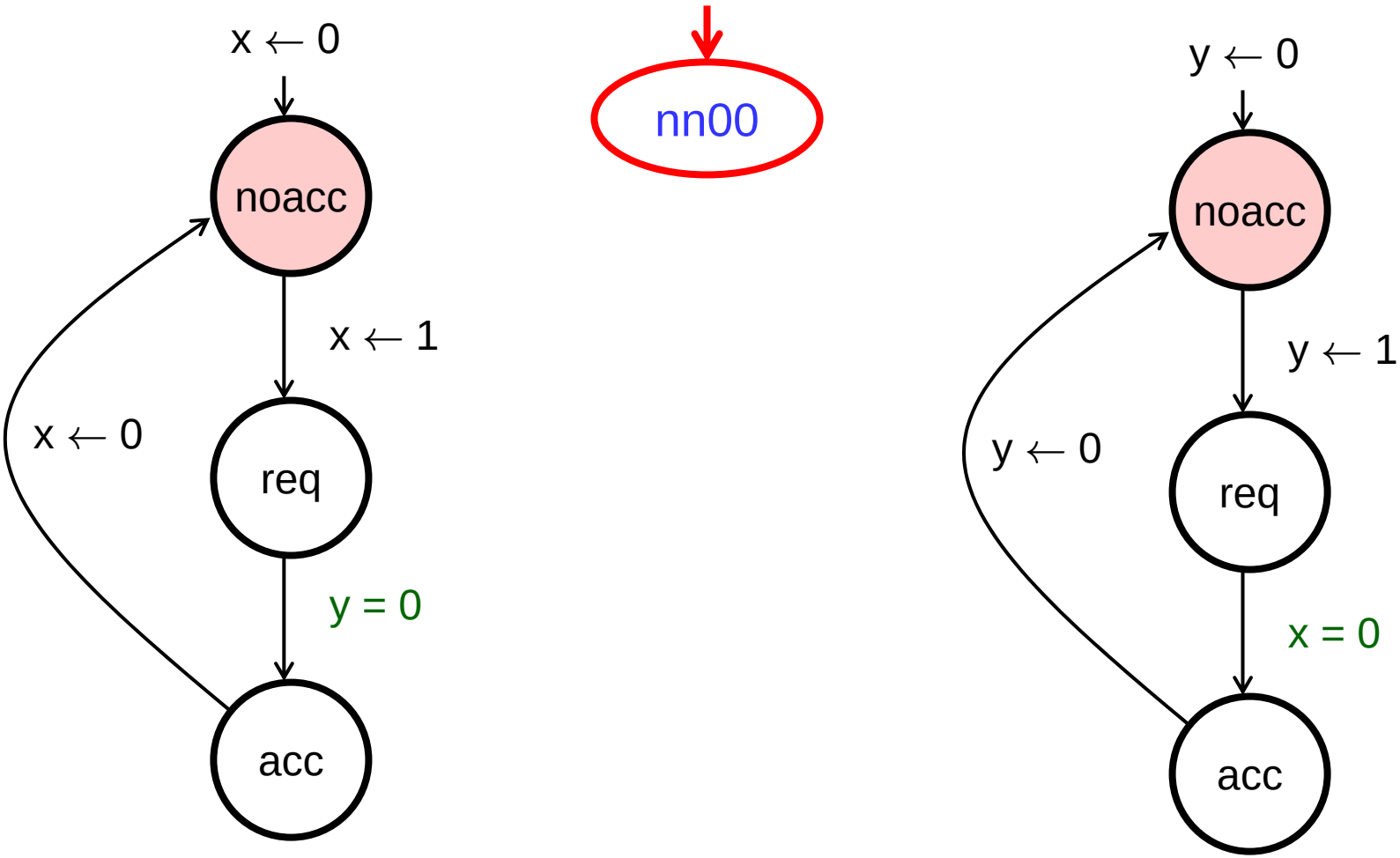


Second Attempt

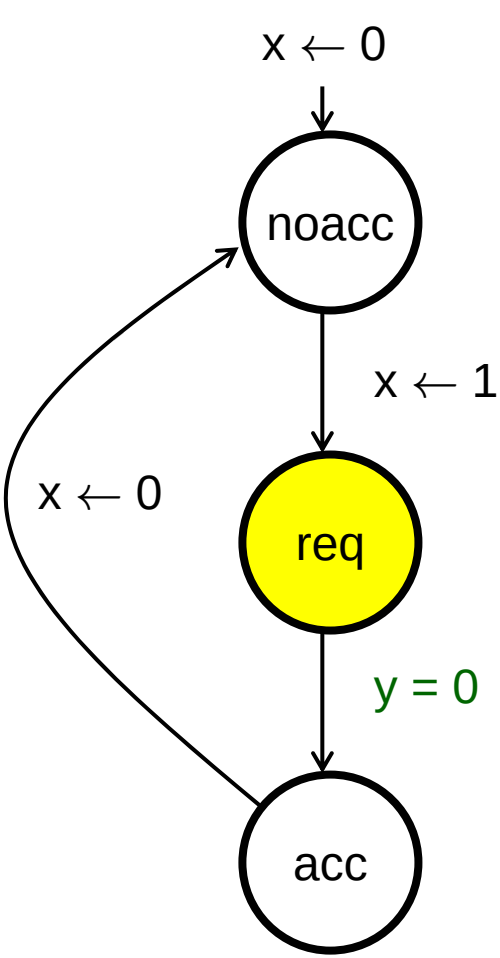


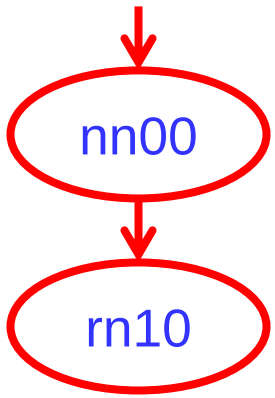


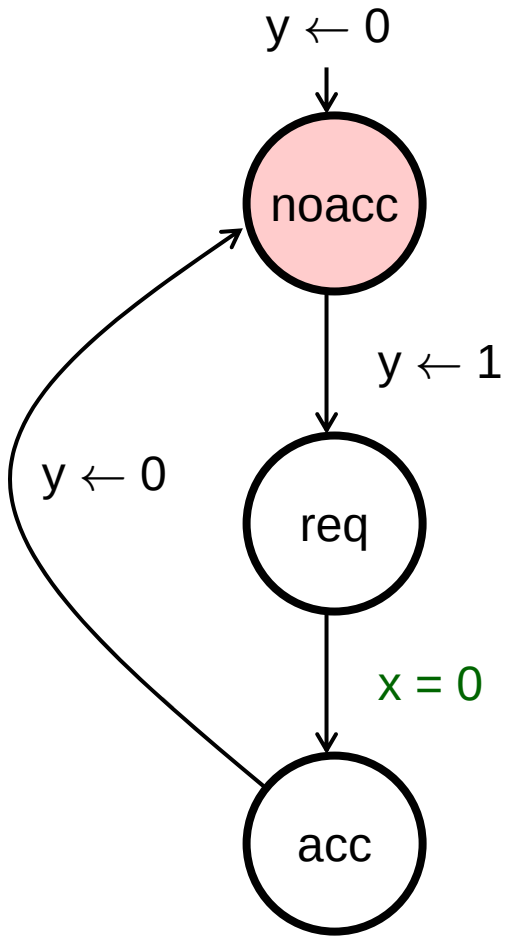
Second Attempt



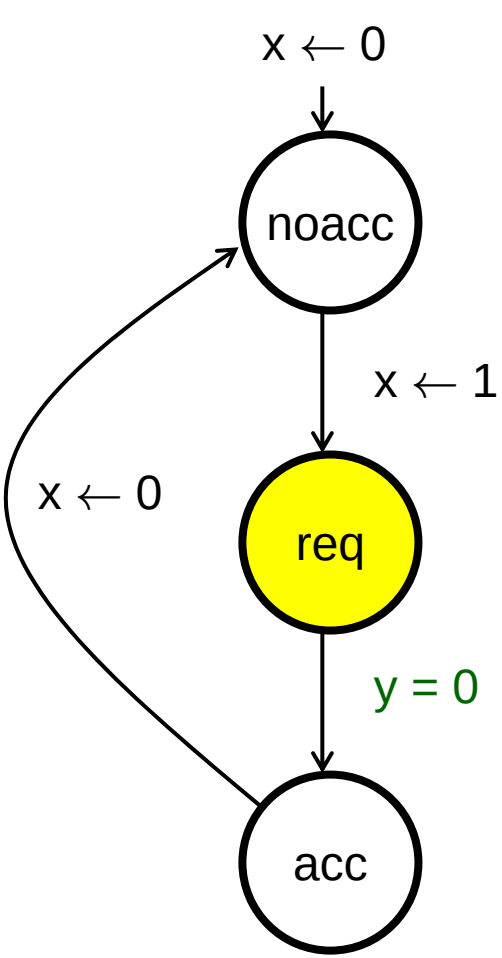
Second Attempt

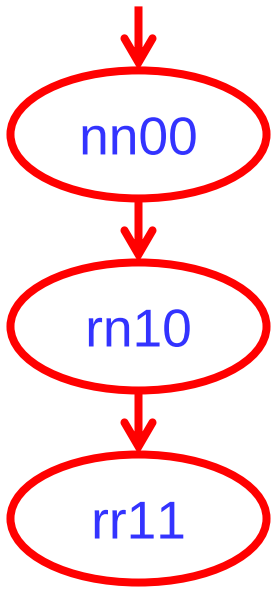


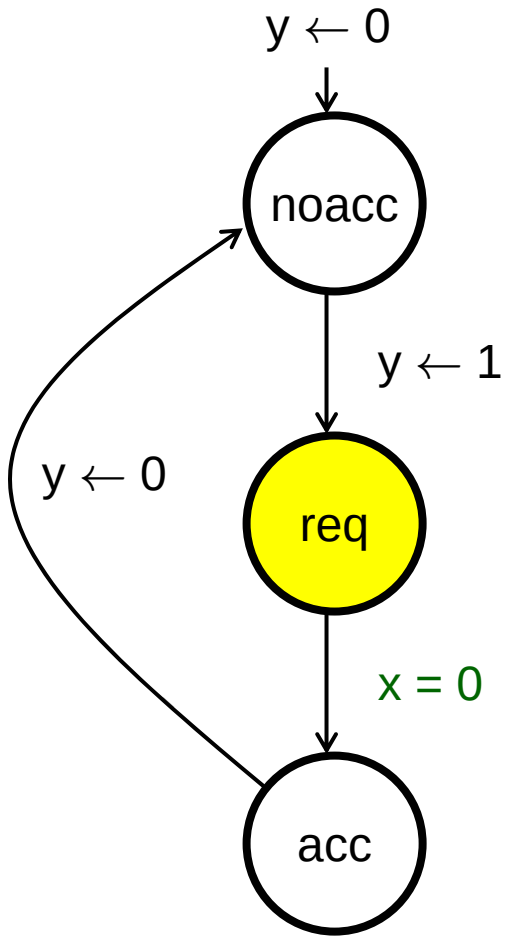




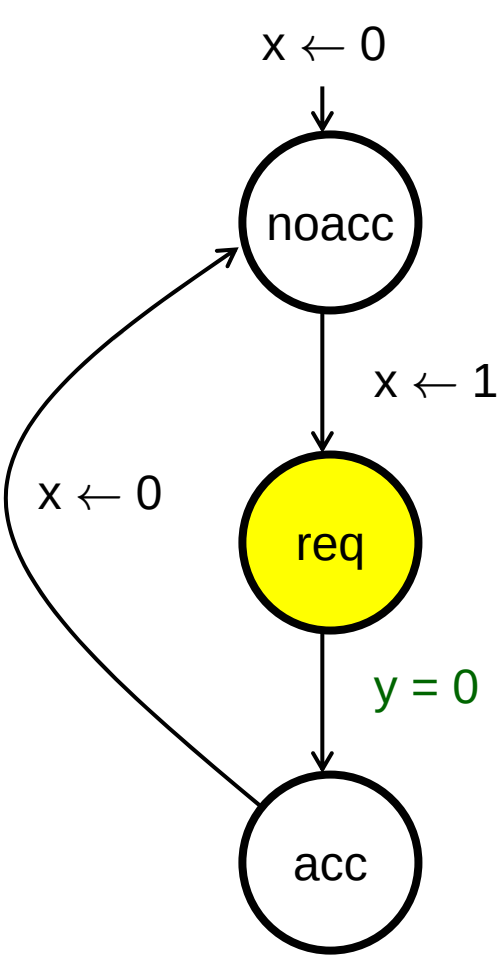
Second Attempt

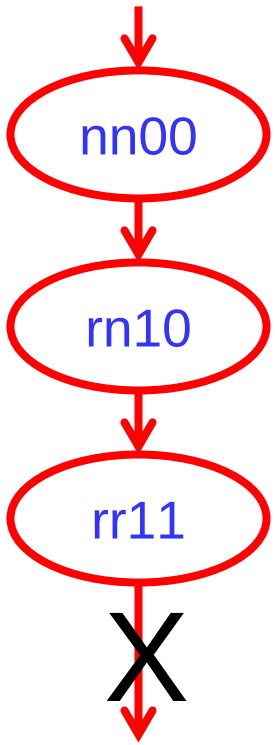


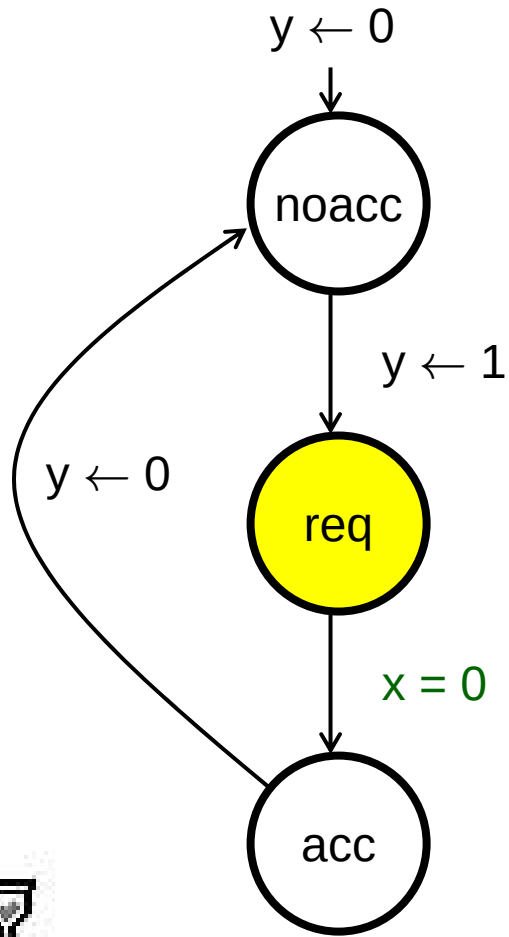




Second Attempt



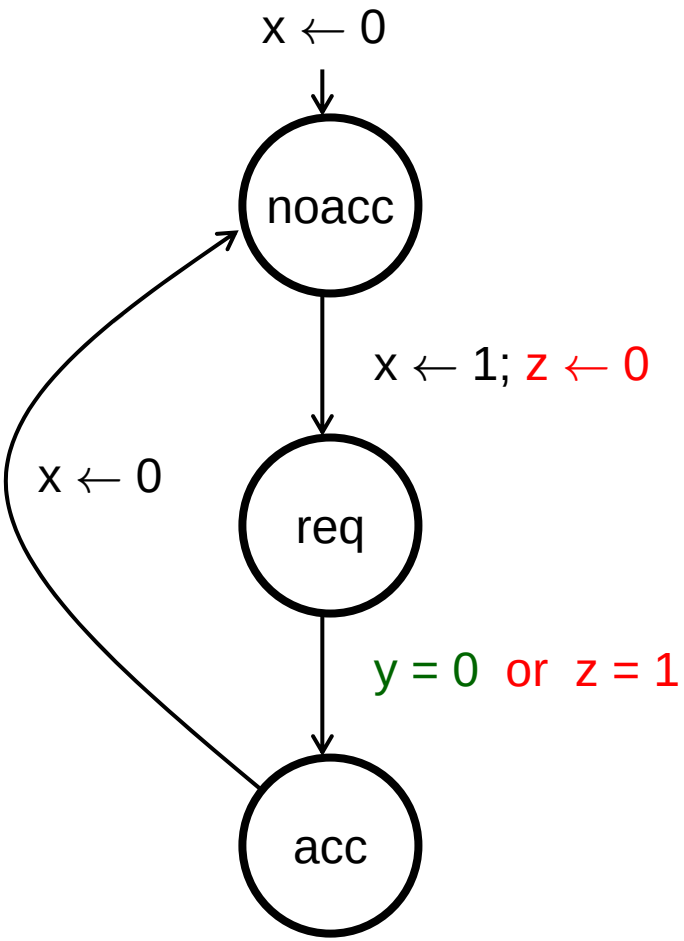


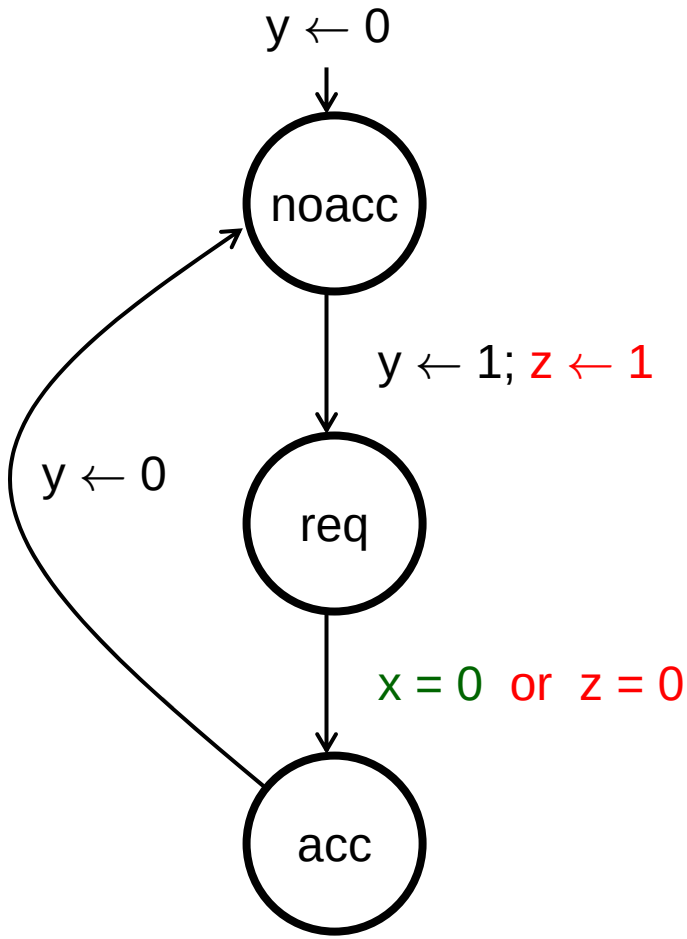


Deadlock.

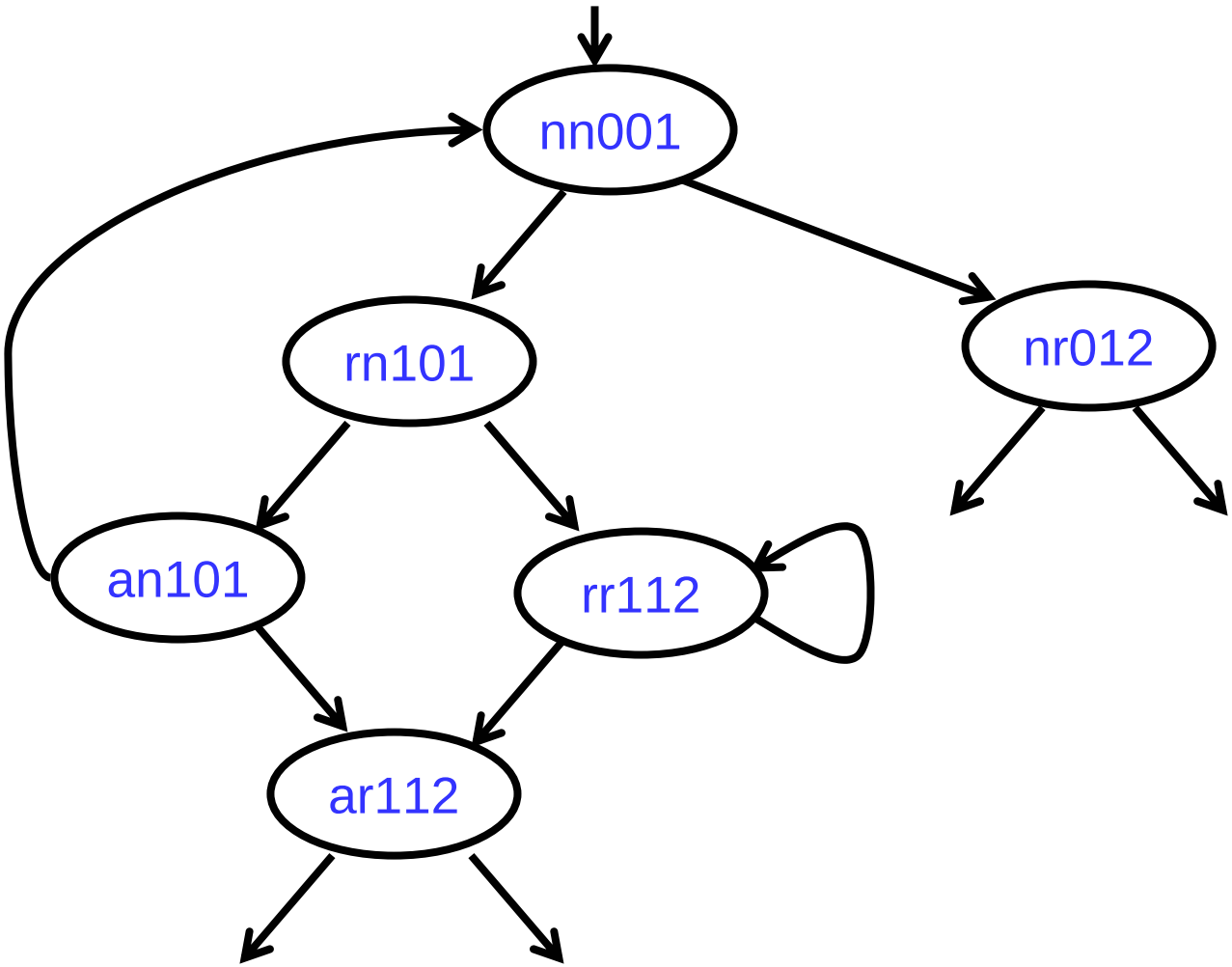
Violates liveness.

Third Attempt

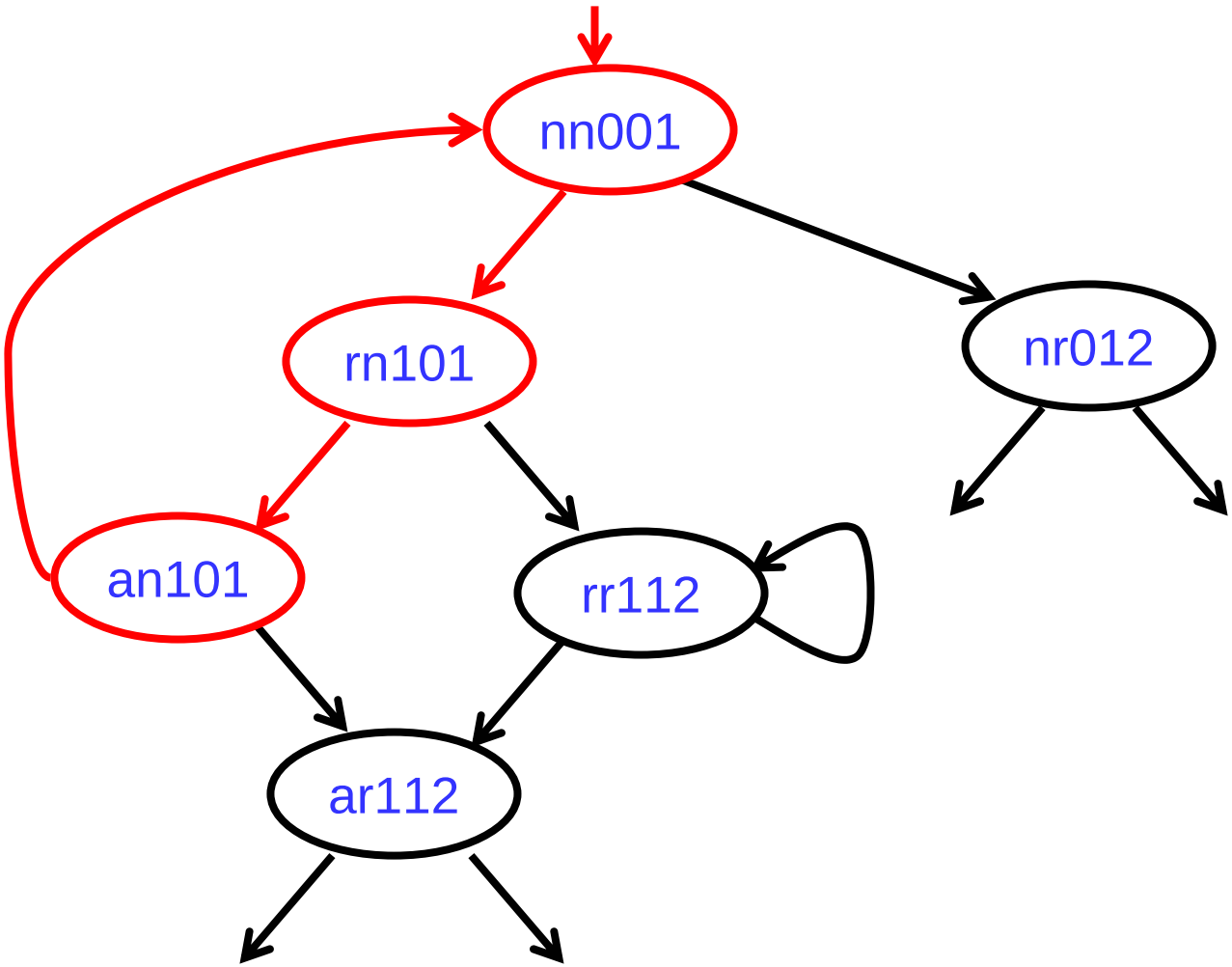




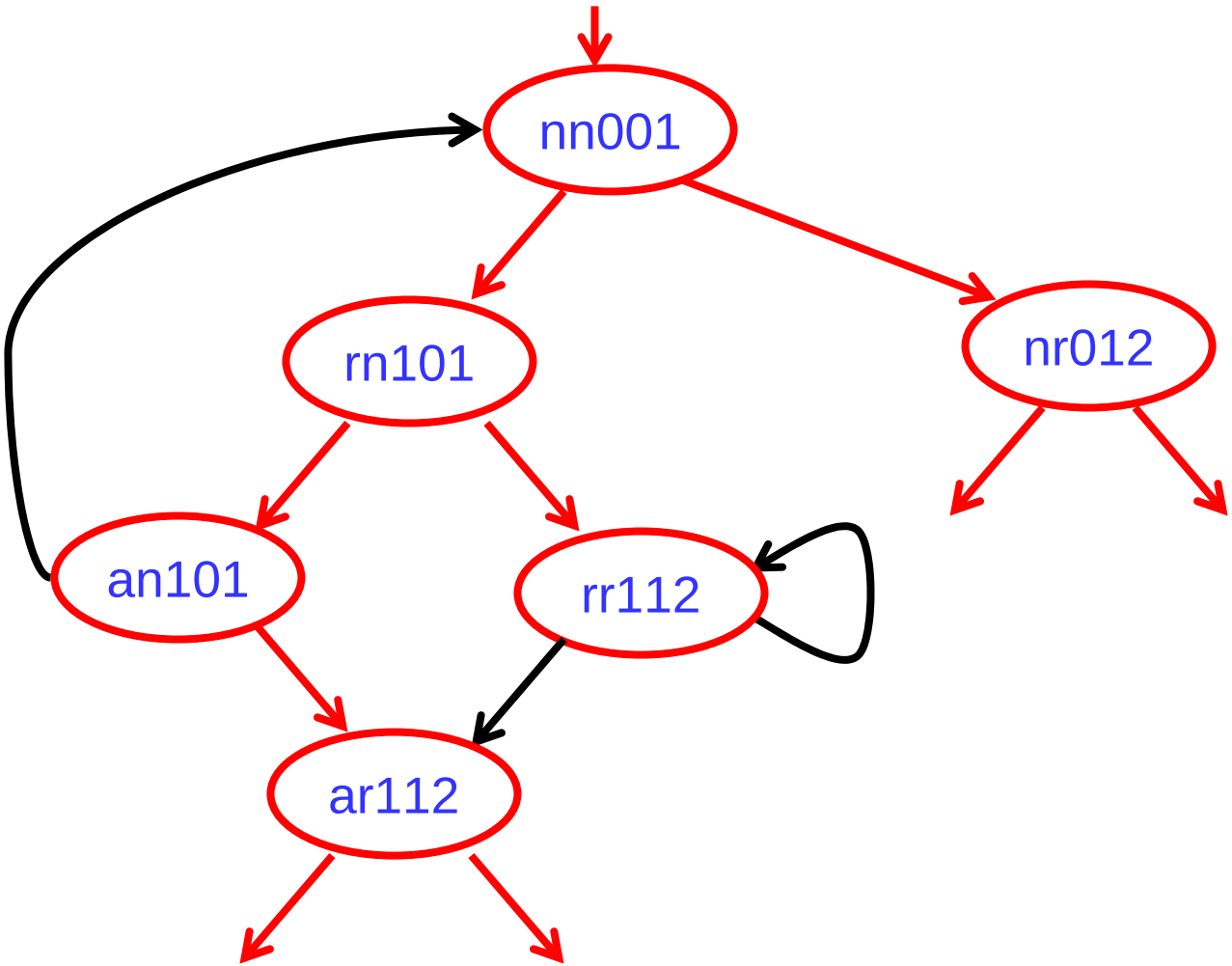
The State Graph



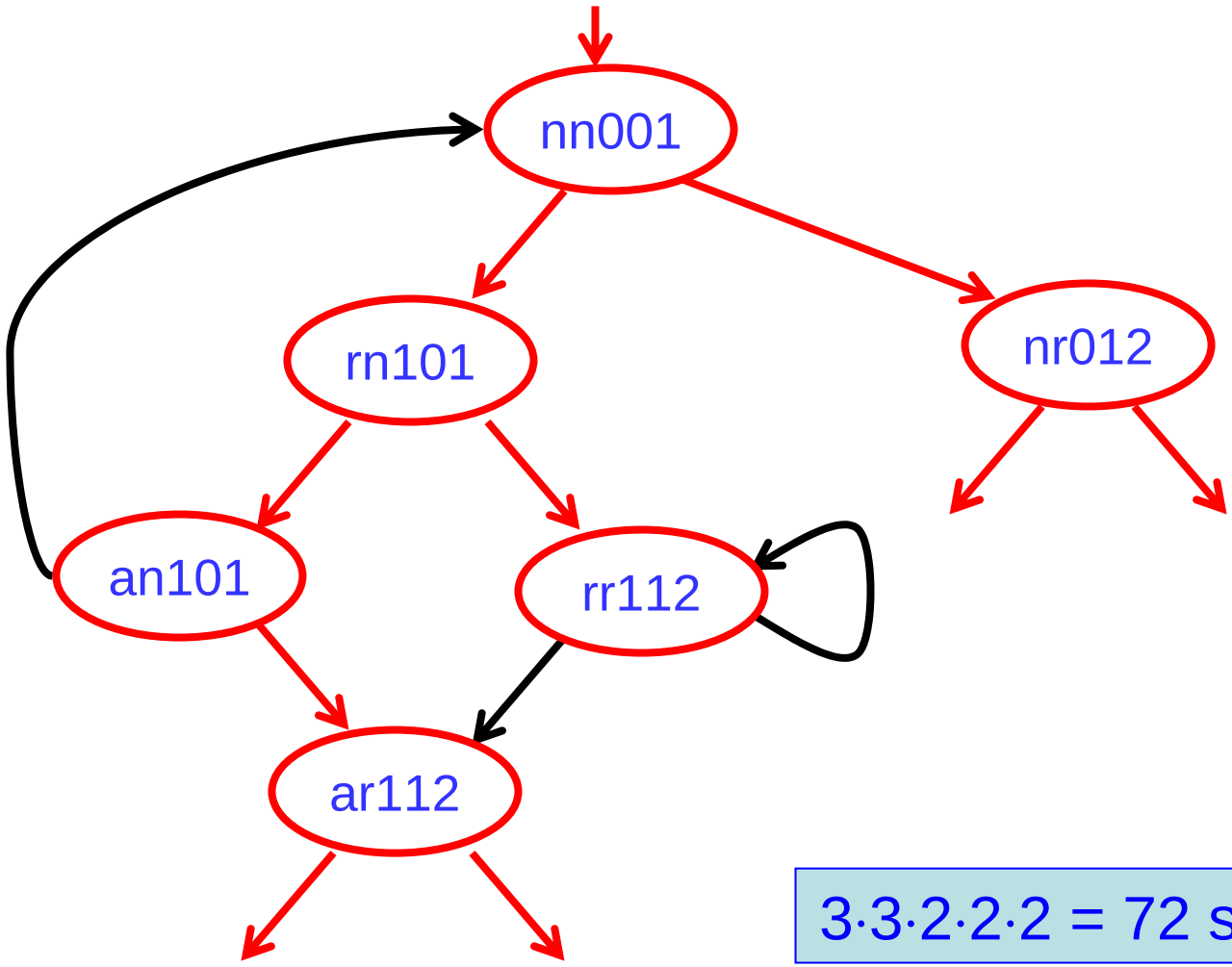
Testing / Simulation: Explore one path at a time.



Model Checking: Explore the whole graph.



Model Checking: Explore the whole graph.



3·3·2·2·2 = 72 states

Two Examples

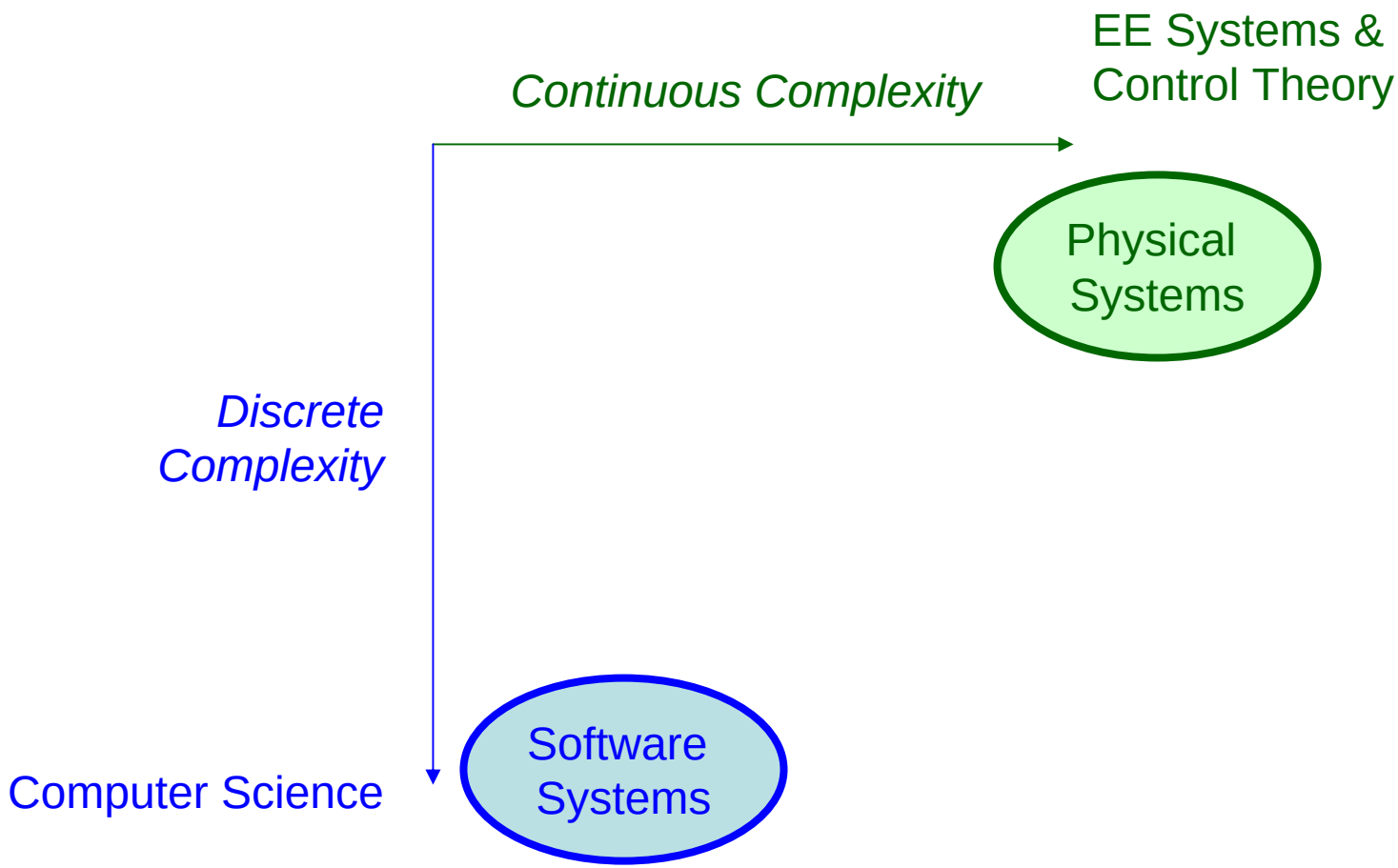
1 Concurrency Bugs
(due to interaction between two programs)

Model:	finite automata
Algorithm:	state graph exploration

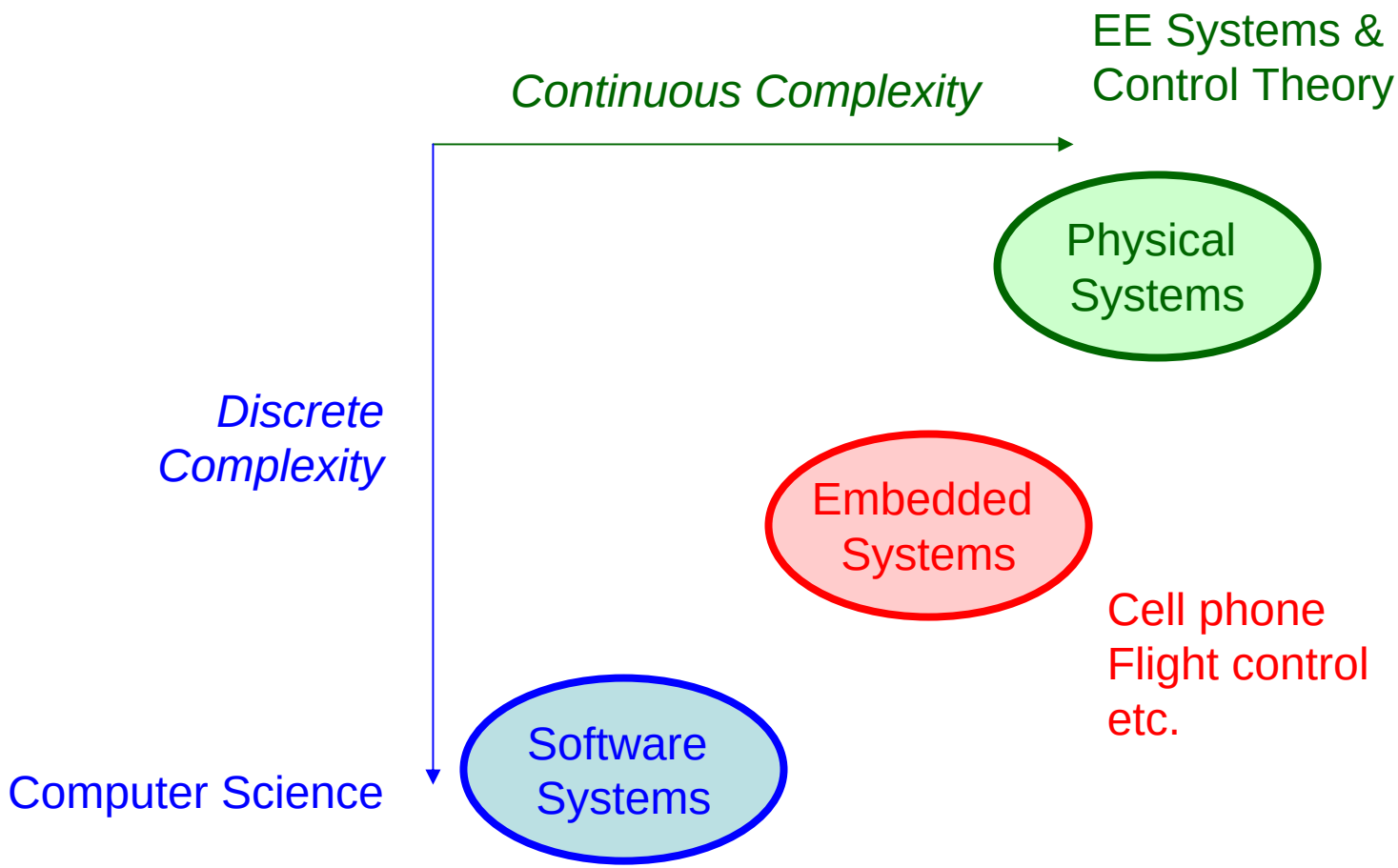
2 Embedded Bugs
(due to interaction between a program and the physical world)

Model:	hybrid automata
Algorithm:	polyhedral state exploration

The Two Cultures



The Two Cultures



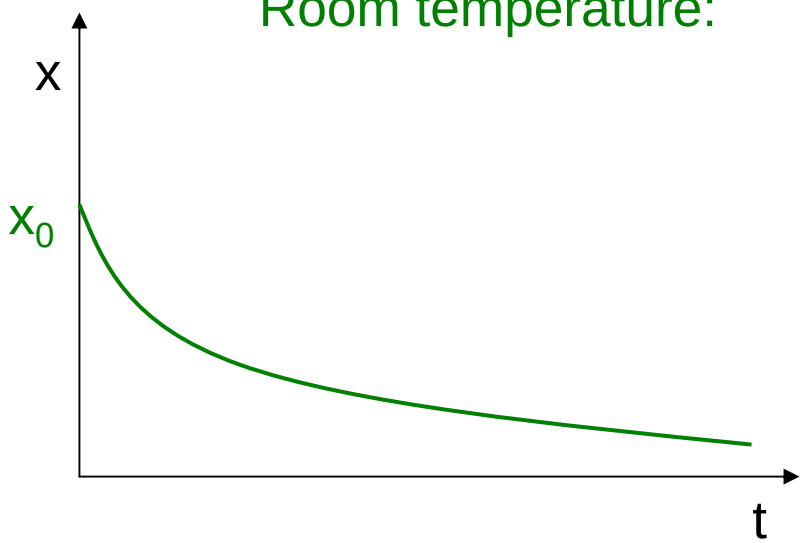
Continuous Dynamical System

State space: $\mathbb{R}^n$

Dynamics: initial condition + differential equations

Room temperature: $x(0) = x_0$

$x'(t) = -K \cdot x(t)$

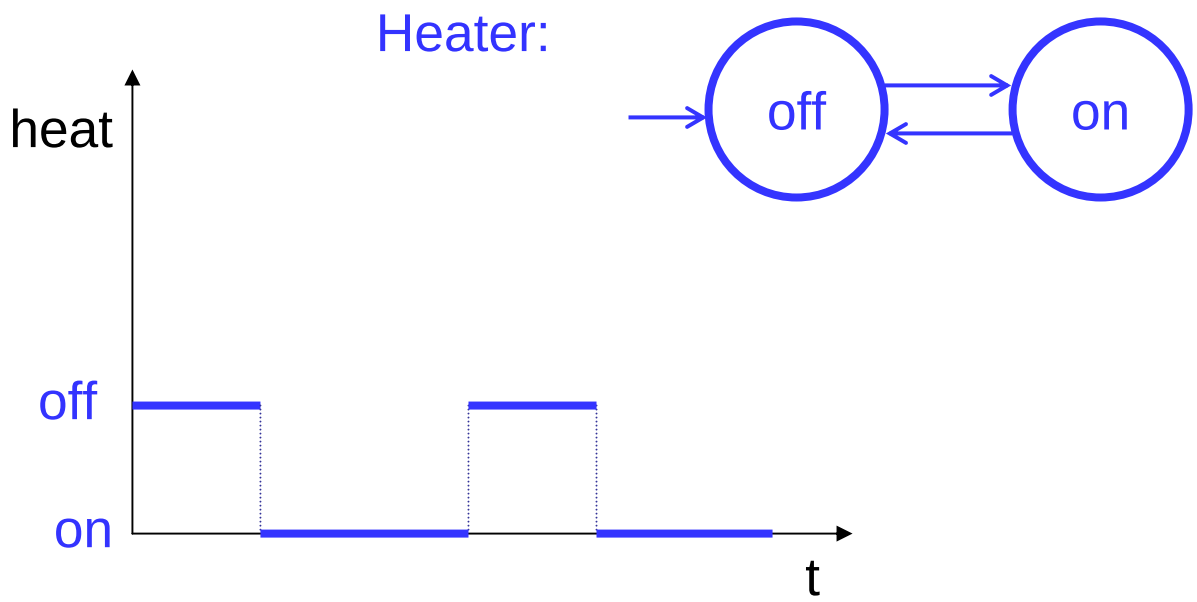


Analytic complexity.

Discrete Software System

State space: $\mathbb{B}^m$

Dynamics: initial condition + state transitions

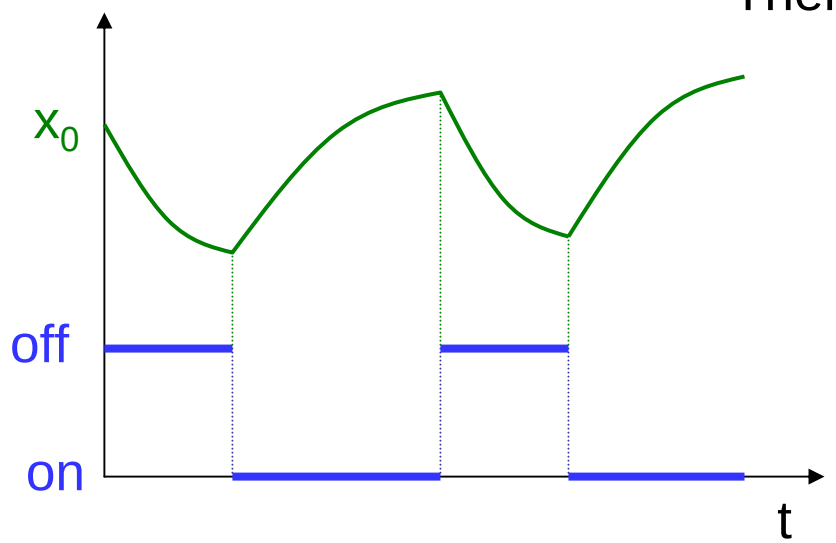


Combinatorial complexity.

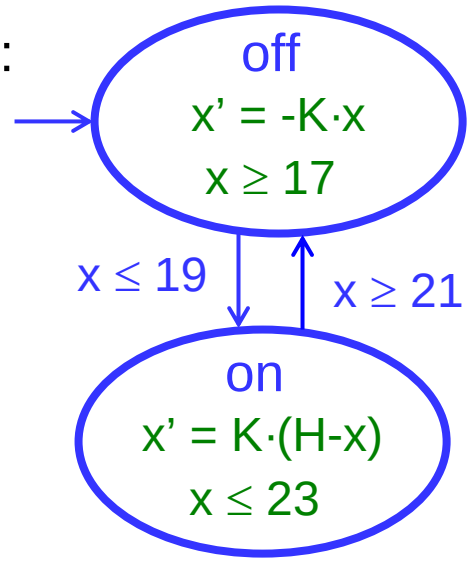
Hybrid Automaton

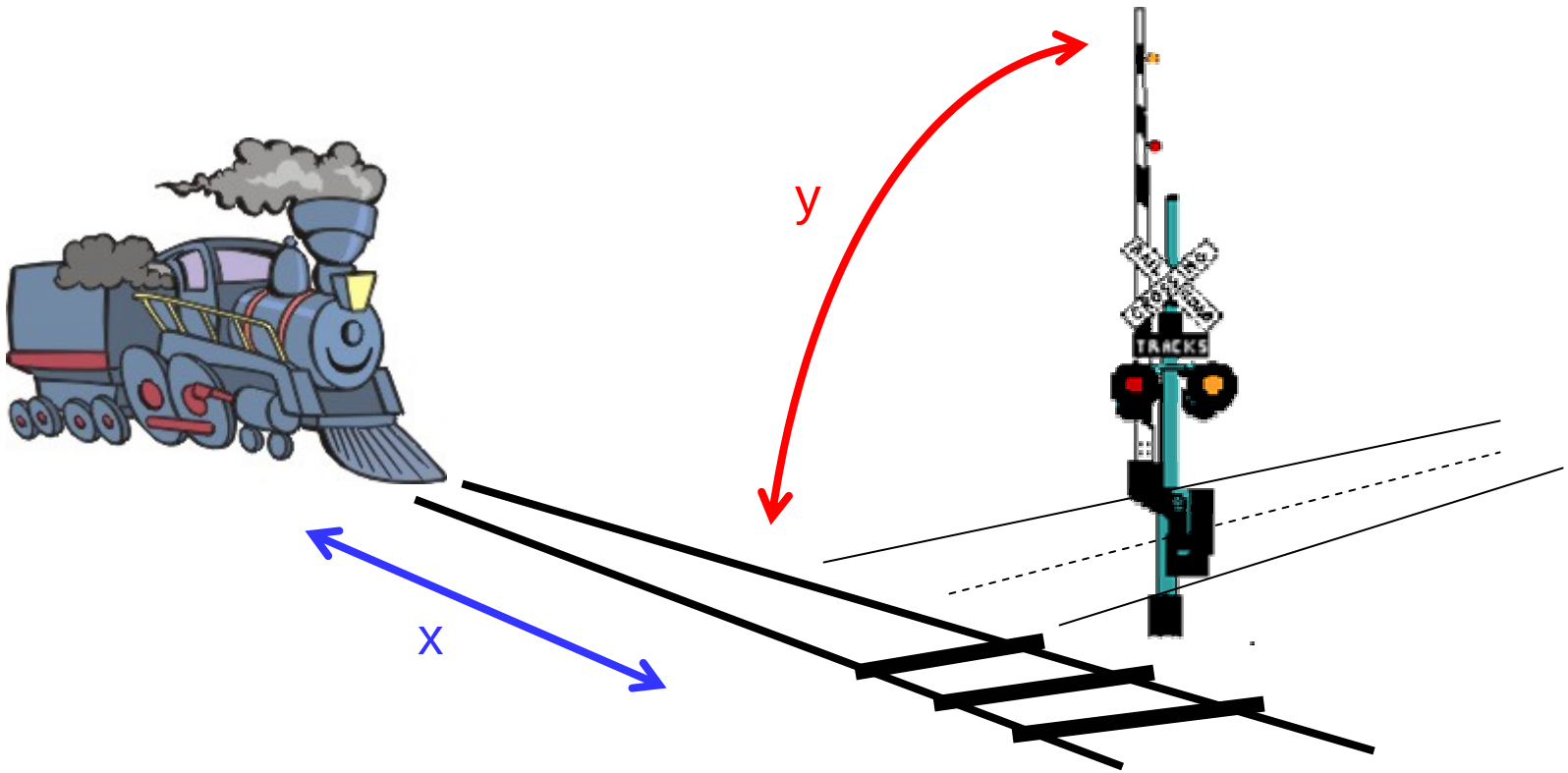
State space: $\mathbb{B}^m \times \mathbb{R}^n$

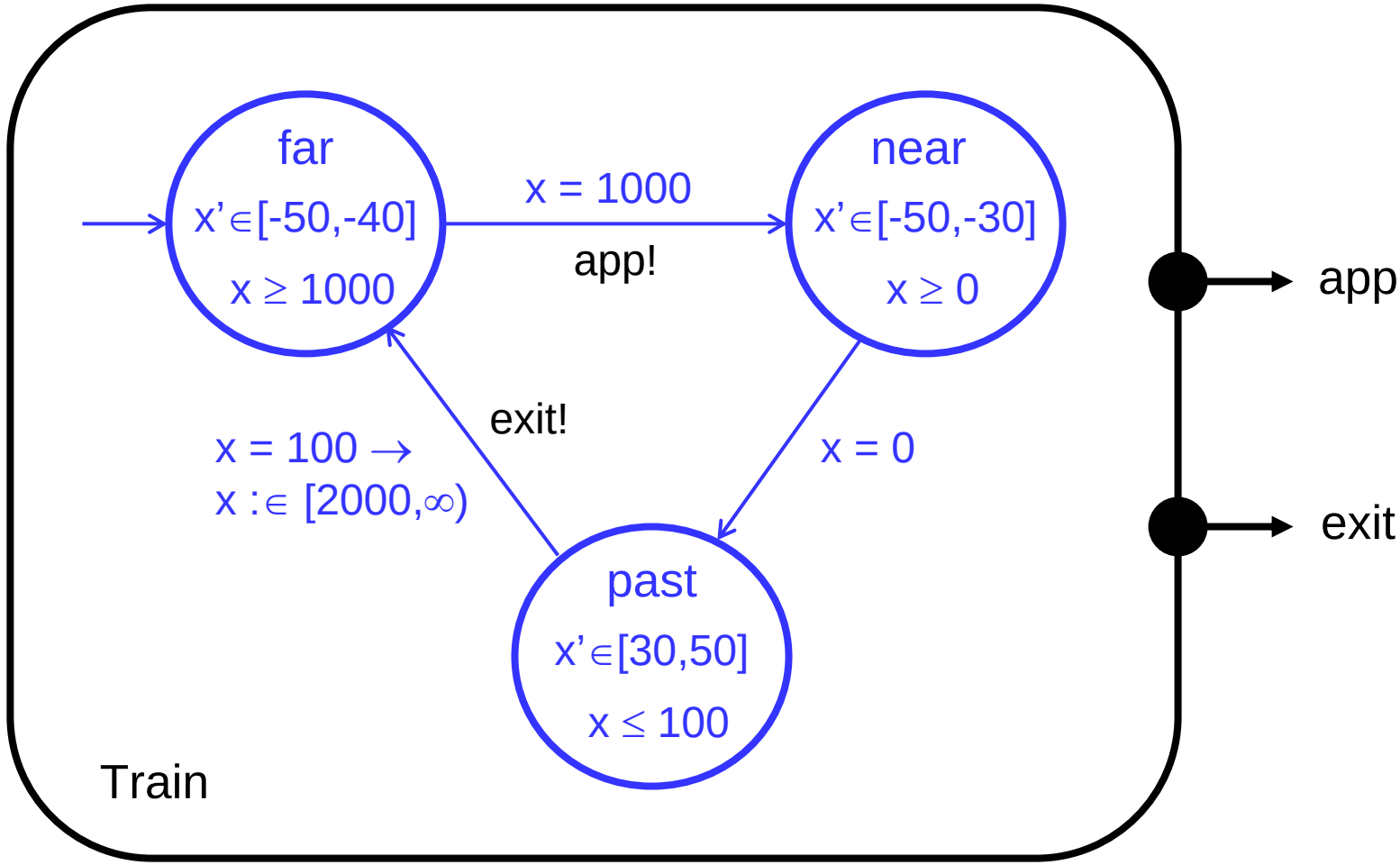
Dynamics: initial condition + state transitions
+ differential equations

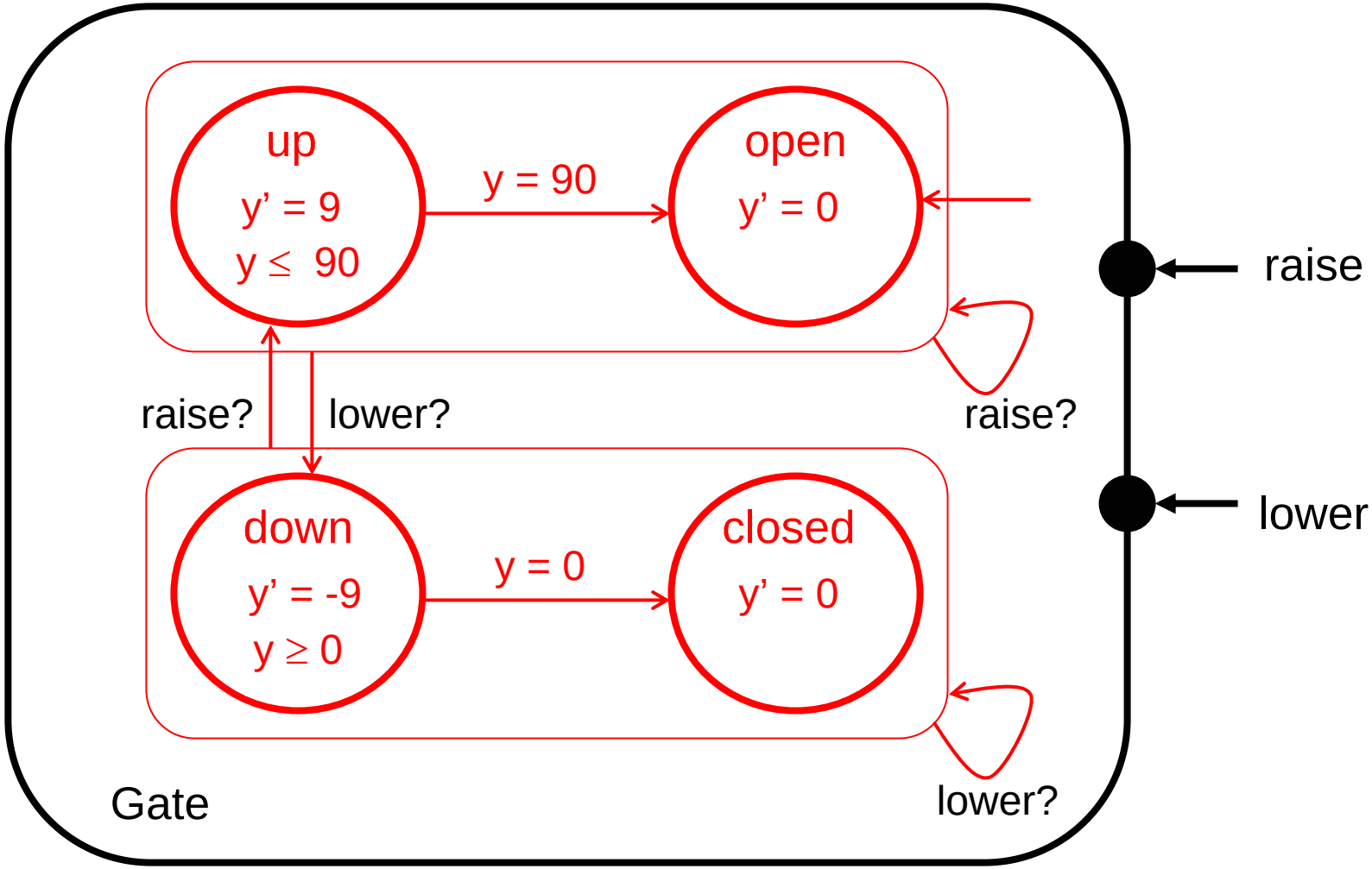


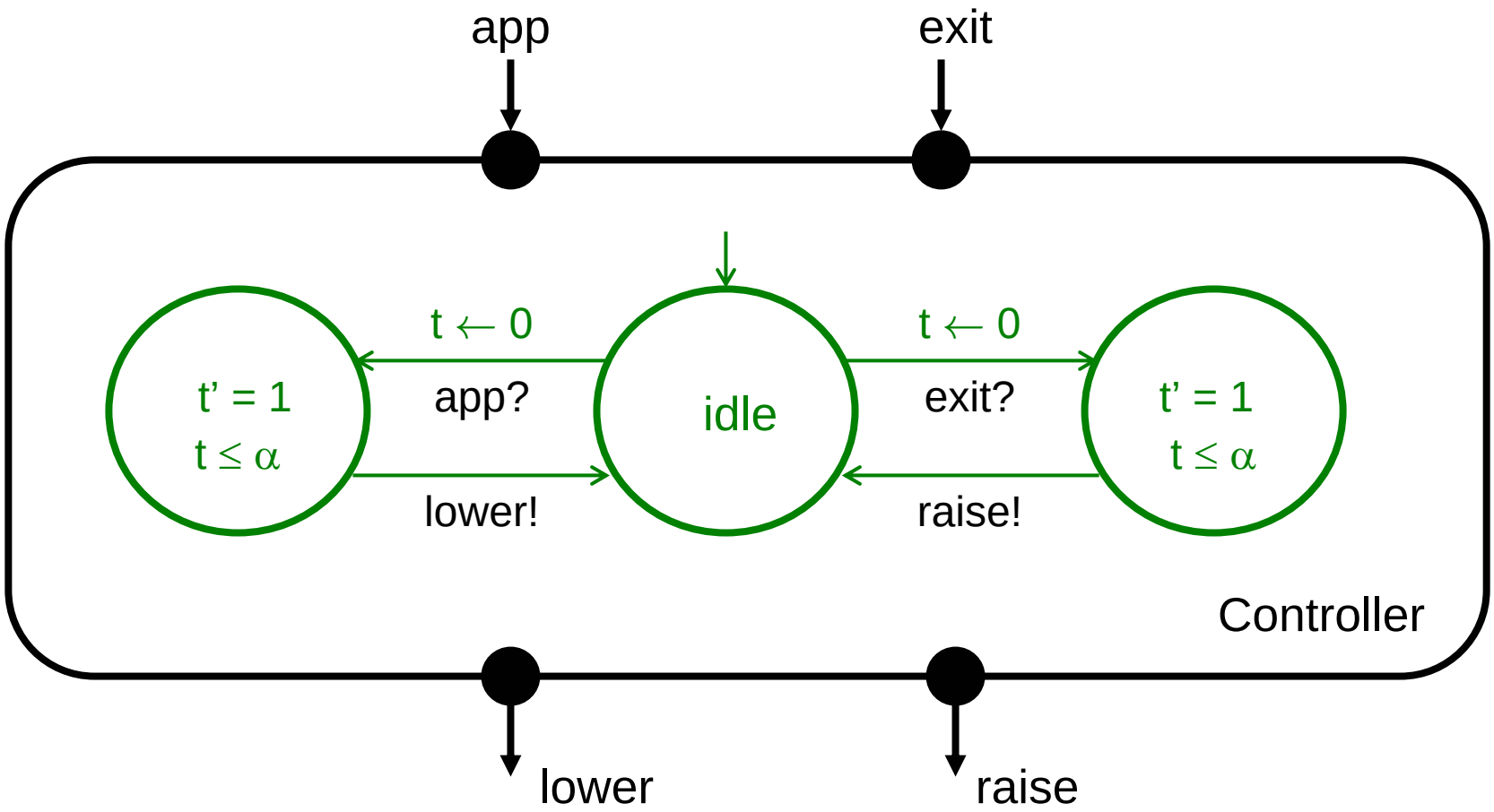
Thermostat:




y
x
RAILROAD CROSSING
TRACKS


far
x'∈[-50,-40]
x ≥ 1000
x = 1000
app!
near
x'∈[-50,-30]
x ≥ 0
exit!
x = 100 →
x :∈ [2000,∞)
x = 0
past
x'∈[30,50]
x ≤ 100
Train
app
exit


up
y' = 9
y ≤ 90
y = 90
open
y' = 0
raise
raise?
lower?
raise?
down
y' = -9
y ≥ 0
y = 0
closed
y' = 0
lower
Gate
lower?



α … response time of the controller

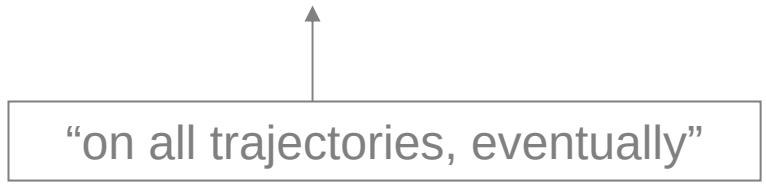
Temporal Logic

Safety: $\forall\Box\ (\ x \leq 10\ \Rightarrow\ \text{Gate = closed}\ )$

Temporal Logic

Safety: $\forall\Box\ (\ x \leq 10\ \Rightarrow\ \text{Gate = closed}\ )$

Liveness: $\forall\Box\ (\ \text{Gate = closed}\ \Rightarrow\ \forall\Diamond\ (\ \text{Gate = open}\ )\ )$

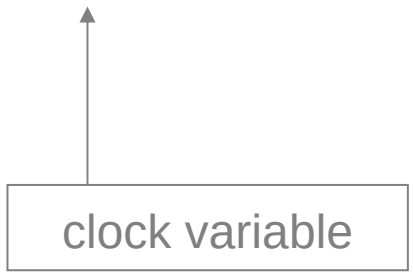


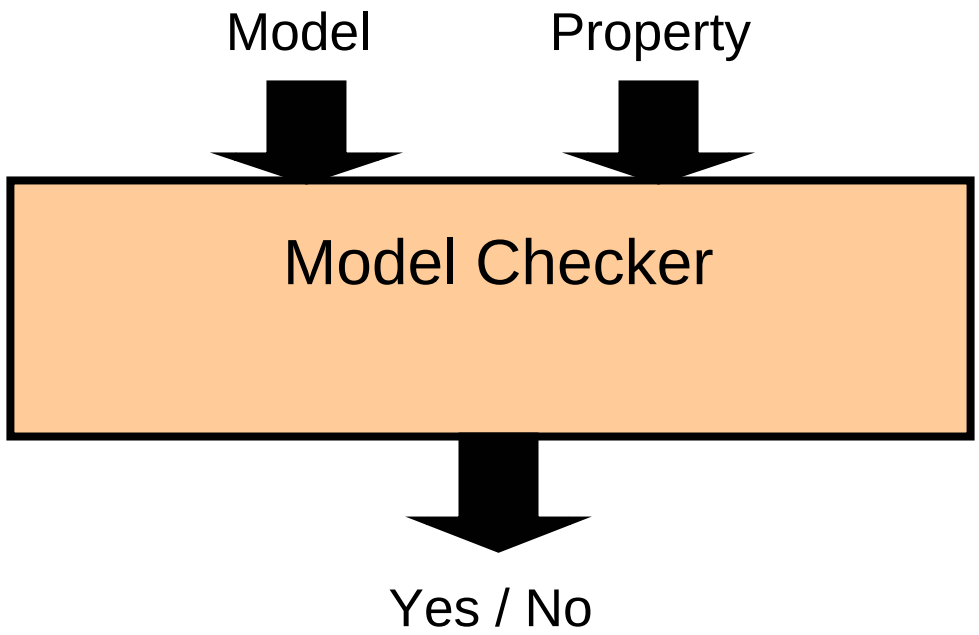
Temporal Logic

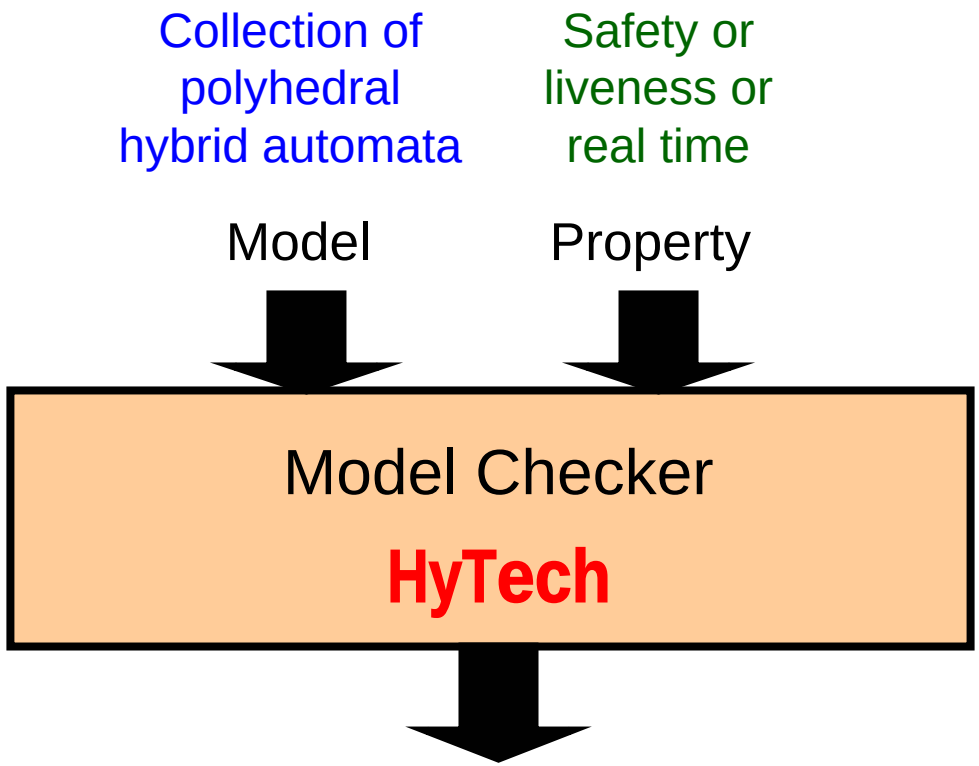
Safety: $\forall\Box\ (\ x \leq 10\ \Rightarrow\ \text{Gate = closed}\ )$

Liveness: $\forall\Box\ (\ \text{Gate = closed}\ \Rightarrow\ \forall\Diamond\ (\ \text{Gate = open}\ )\ )$

Real time: $\forall\Box\ z \leftarrow 0.\ (\ z' = 1\ \Rightarrow\ \forall\Diamond\ (\ \text{Gate = open}\ \wedge\ z \leq 60\ )\ )$

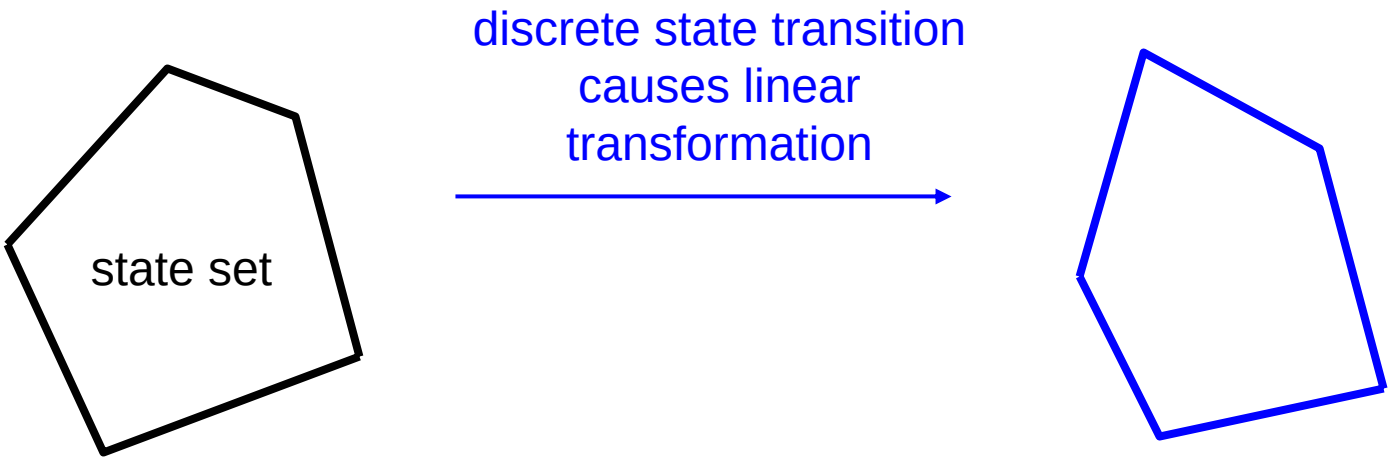



Model
Property
Model Checker
Yes / No

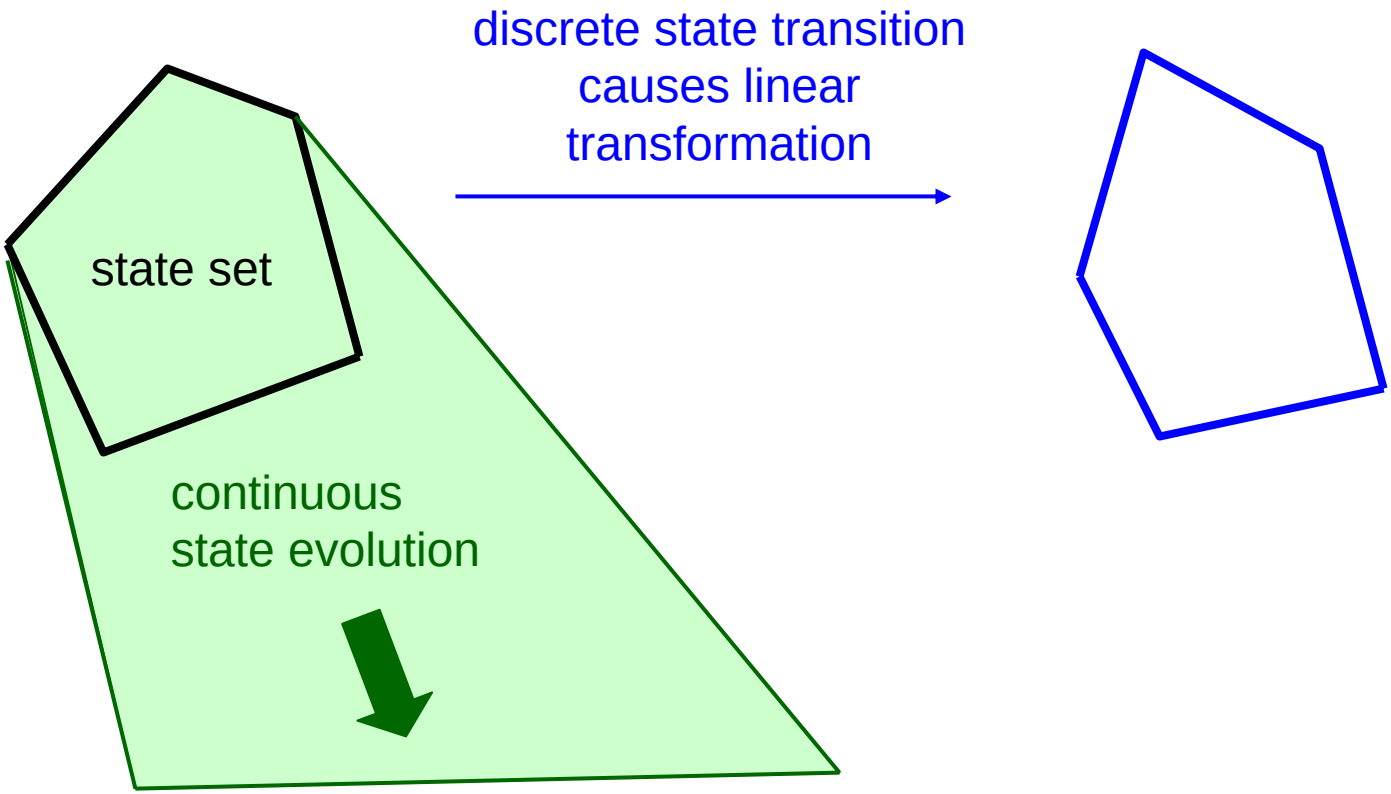


Example: “For which values of α is the controller safe?”

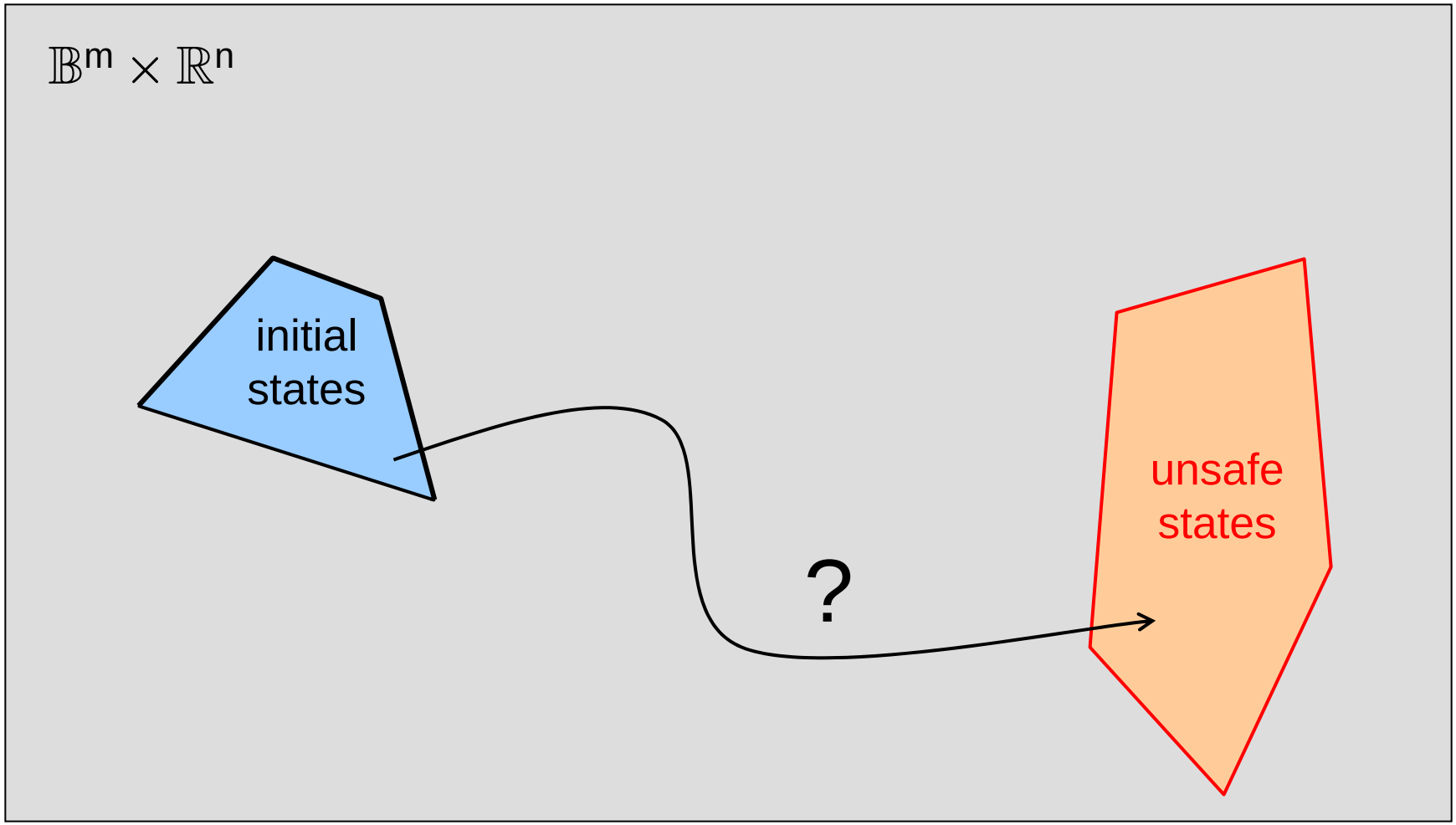
Each state change of a polyhedral hybrid automaton transforms a polyhedral set into a polyhedral set.



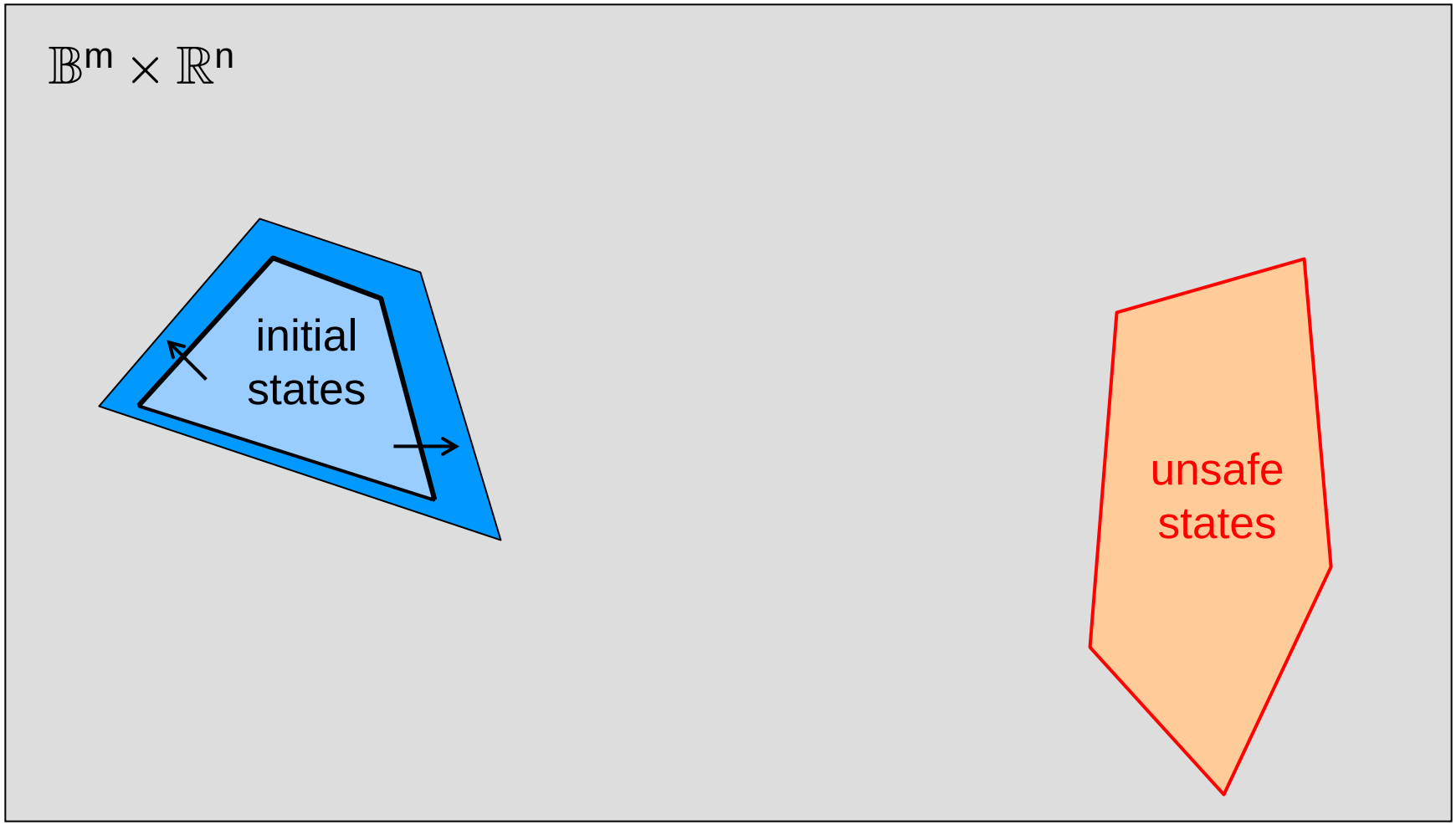
Each state change of a polyhedral hybrid automaton transforms a polyhedral set into a polyhedral set.



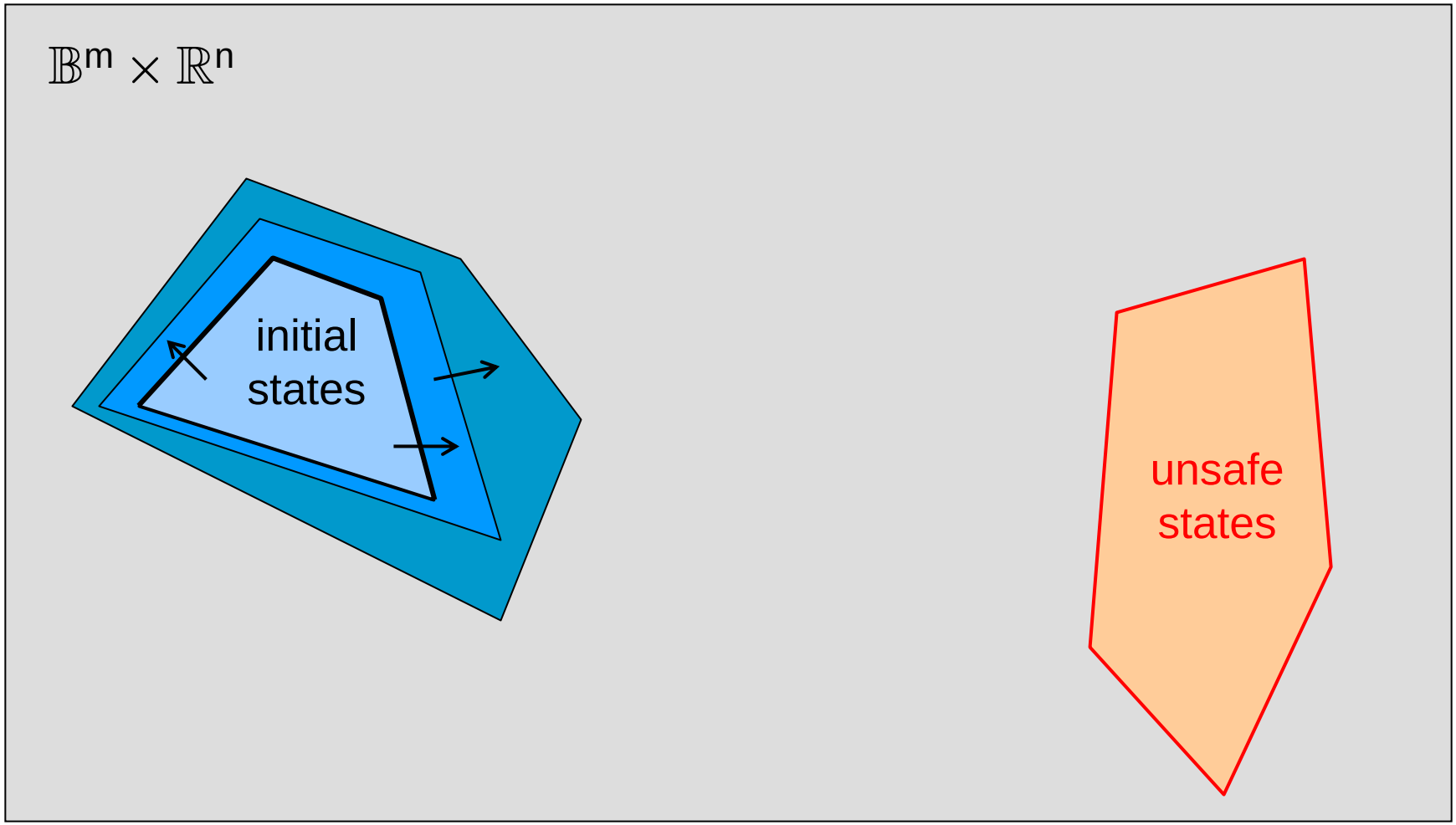
Model Checking for Safety



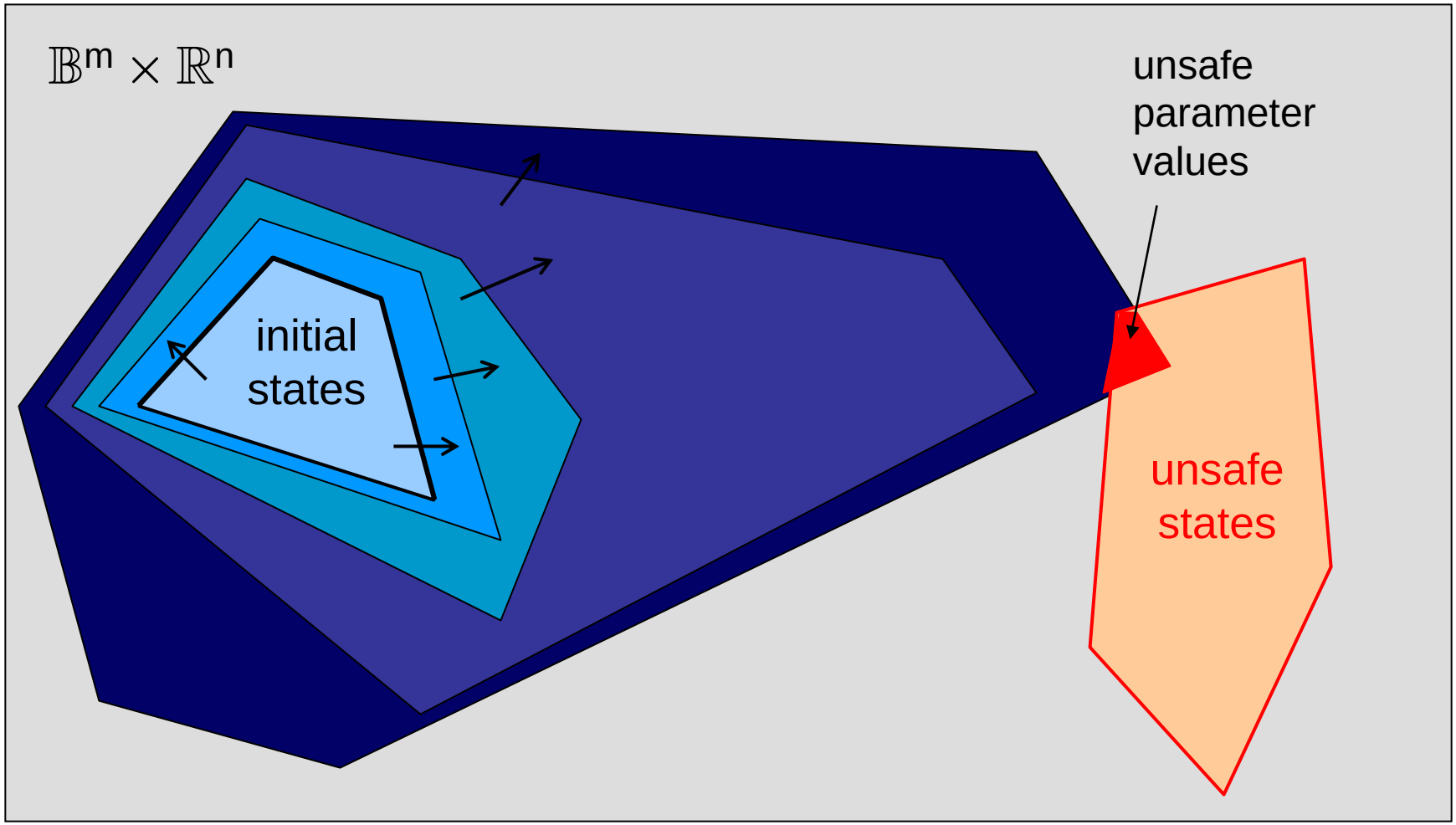
Model Checking for Safety



Model Checking for Safety



Model Checking for Safety



HyTech on Train + Gate + Controller

hytech-out.cgi.html

Applications of **HyTech**

-automotive engine control [Wong-Toi et al.]

-chemical plant control [Preussig et al.]

-flight control [Honeywell; Rockwell-Collins]

-air traffic control [Tomlin et al.]

-robot control [Corbett et al.]

Still a long way to go ...

Uptime: 123 years

Indeed, it will get worse before it gets better.

2004

2006

2007

[Intel]

Despite all the differences,
there are things we can learn from systems engineering:

Engineering	Computer Science
Theories of estimation Theories of robustness	Theories of correctness

Despite all the differences,
there are things we can learn from systems engineering:

Engineering	Computer Science
Theories of estimation Theories of robustness	Theories of correctness
Goal: build reliable systems.	*Temptation: programs are mathematical objects; hence we want to prove them correct.*

The MTC Mission Statement

Develop models and algorithms that let us quantify how the effort spent during design relates to the quality of the software product.

The MTC Mission Statement

Systems are not correct or incorrect, but there are many shades in between.

Develop models and algorithms that let us quantify how the effort spent during design relates to the quality of the software product.

The MTC Mission Statement

Systems are not correct or incorrect, but there are many shades in between.

Develop models and algorithms that let us quantify how the effort spent during design relates to the quality of the software product.

Not only programming, but especially system integration.

The MTC Mission Statement

Systems are not correct or incorrect, but there are many shades in between.

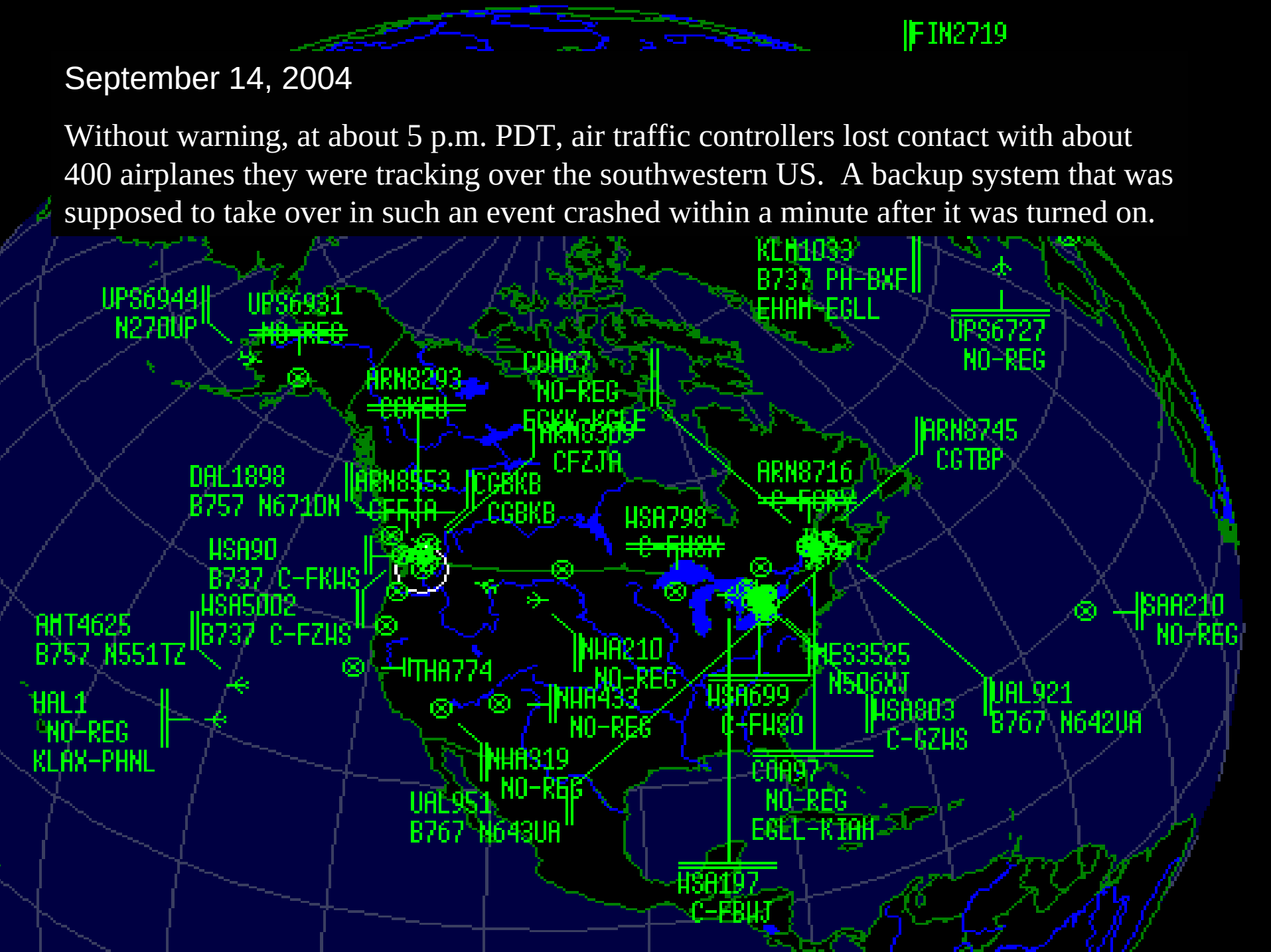
Develop models and algorithms that let us quantify how the effort spent during design relates to the quality of the software product.

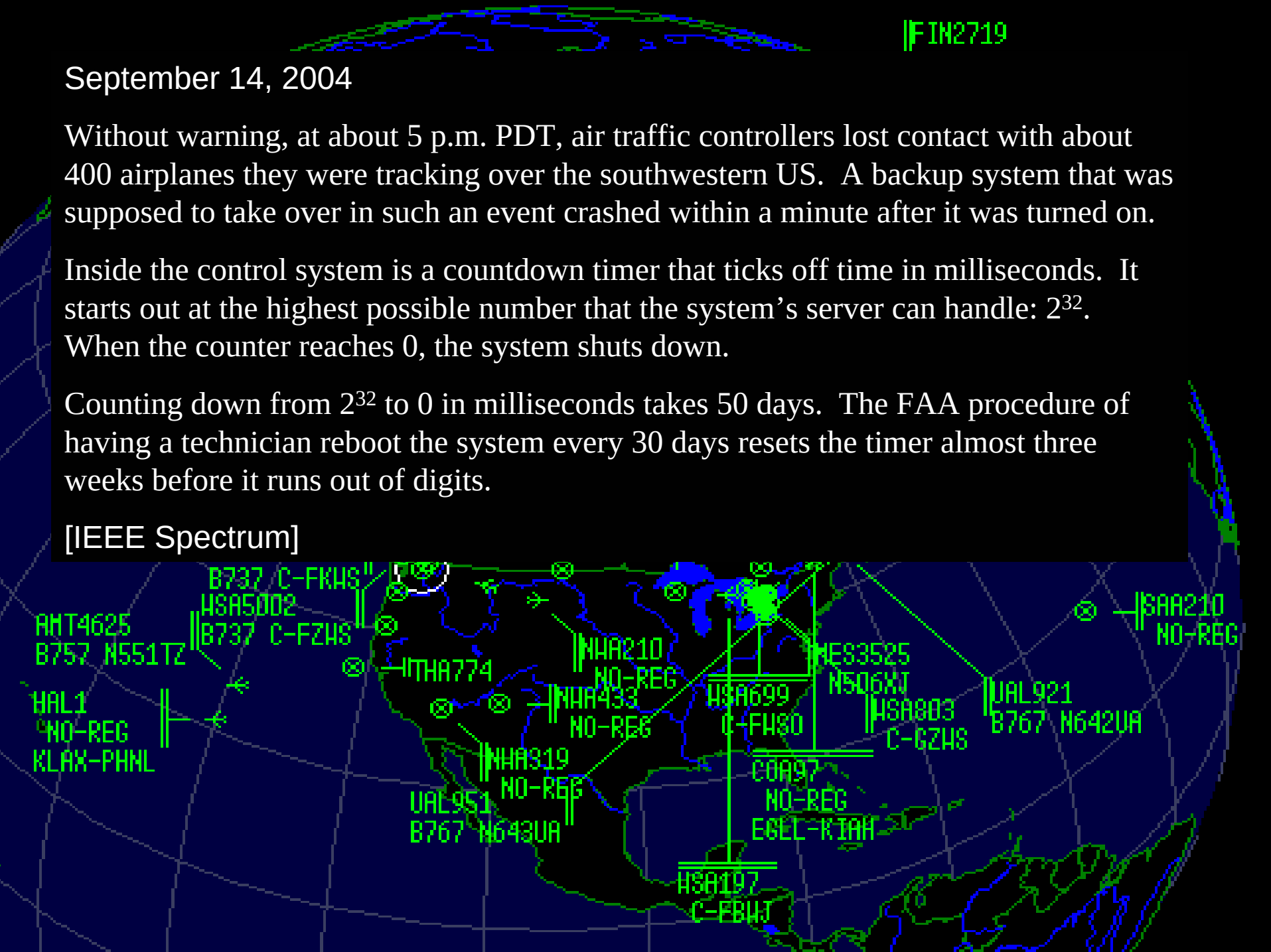
Not only programming, but especially system integration.

Not only functionality and performance, but also robustness (usability, fault tolerance, security).

September 14, 2004

Without warning, at about 5 p.m. PDT, air traffic controllers lost contact with about 400 airplanes they were tracking over the southwestern US. A backup system that was supposed to take over in such an event crashed within a minute after it was turned on.





September 14, 2004

Without warning, at about 5 p.m. PDT, air traffic controllers lost contact with about 400 airplanes they were tracking over the southwestern US. A backup system that was supposed to take over in such an event crashed within a minute after it was turned on.

Inside the control system is a countdown timer that ticks off time in milliseconds. It starts out at the highest possible number that the system's server can handle: 2^{32}. When the counter reaches 0, the system shuts down.

Counting down from 2^{32} to 0 in milliseconds takes 50 days. The FAA procedure of having a technician reboot the system every 30 days resets the timer almost three weeks before it runs out of digits.

[IEEE Spectrum]


Thank you for your attention.
BC 350