how to prevent disasters

Daniel Jackson, MIT

Siren//NL, Veldhoven · November 2, 2010

CSAIL

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

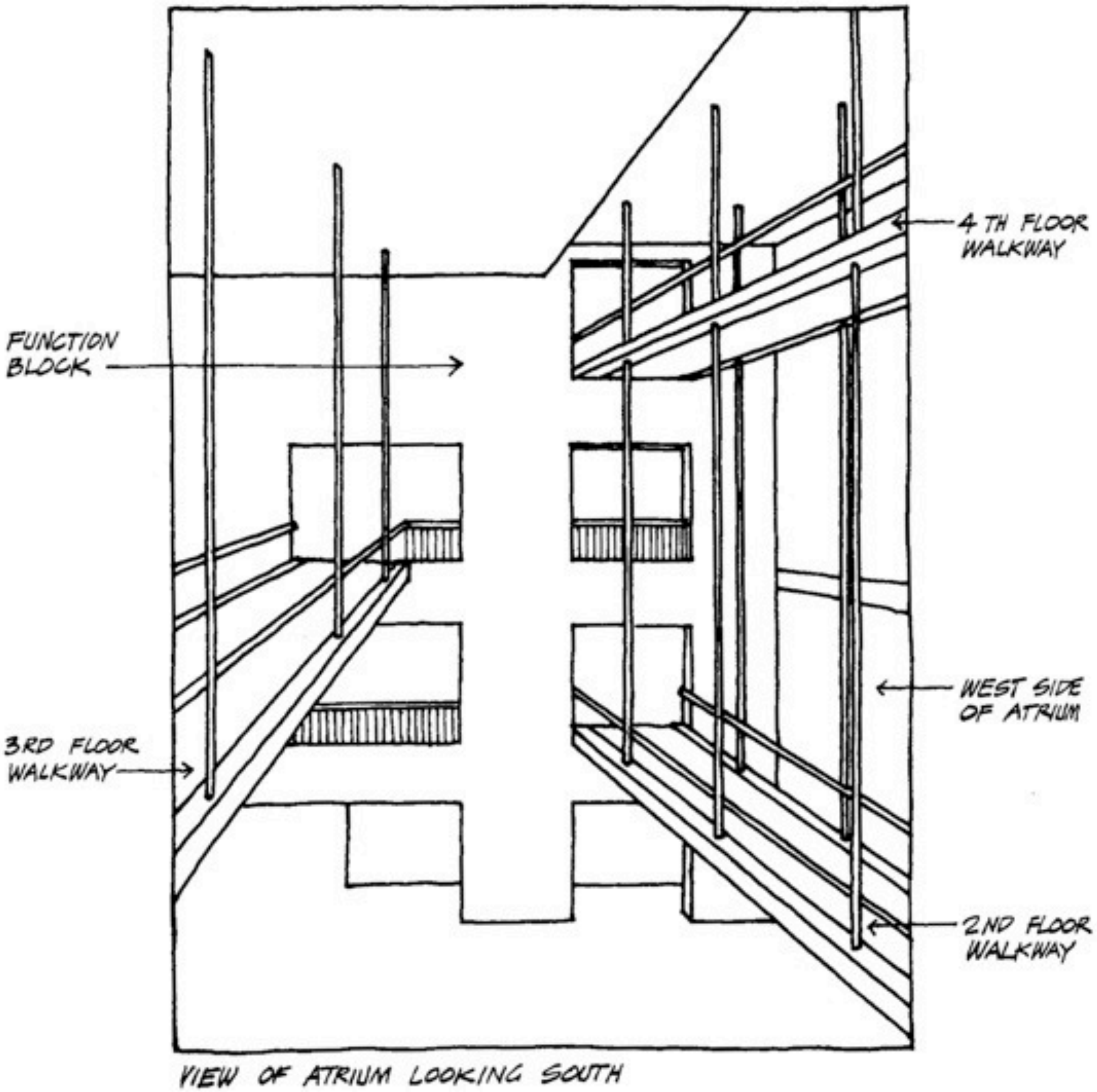
a civil engineering disaster

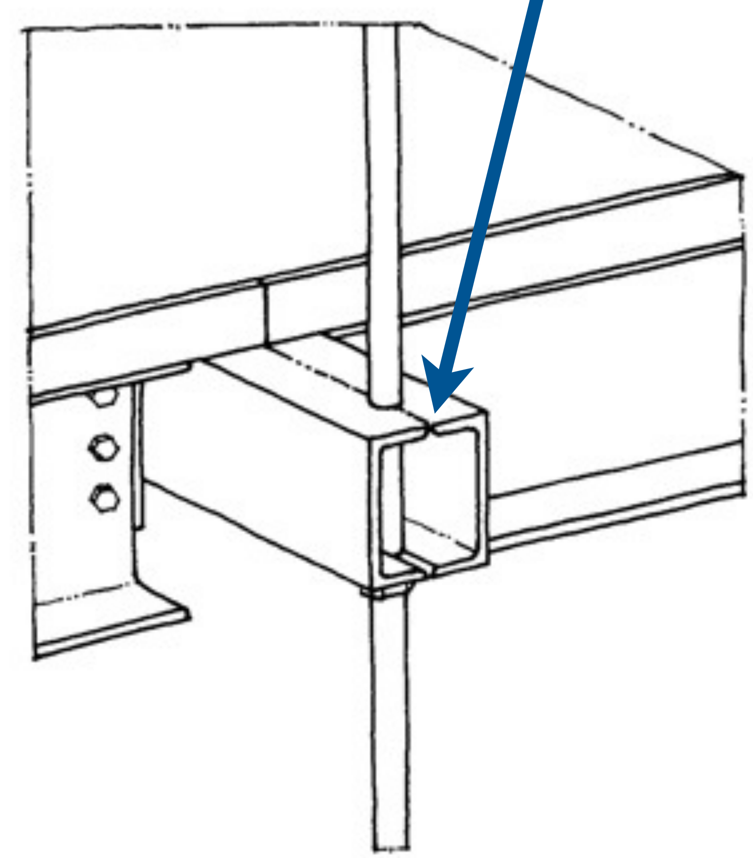
kansas city hyatt regency, 1981

New York Times

the design

beam supports one walkway

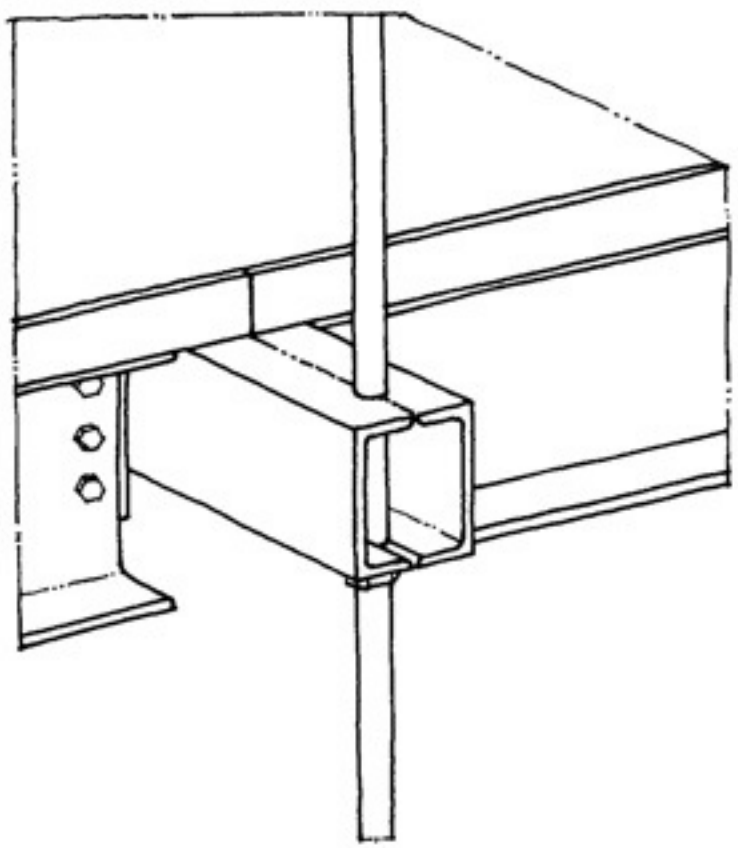




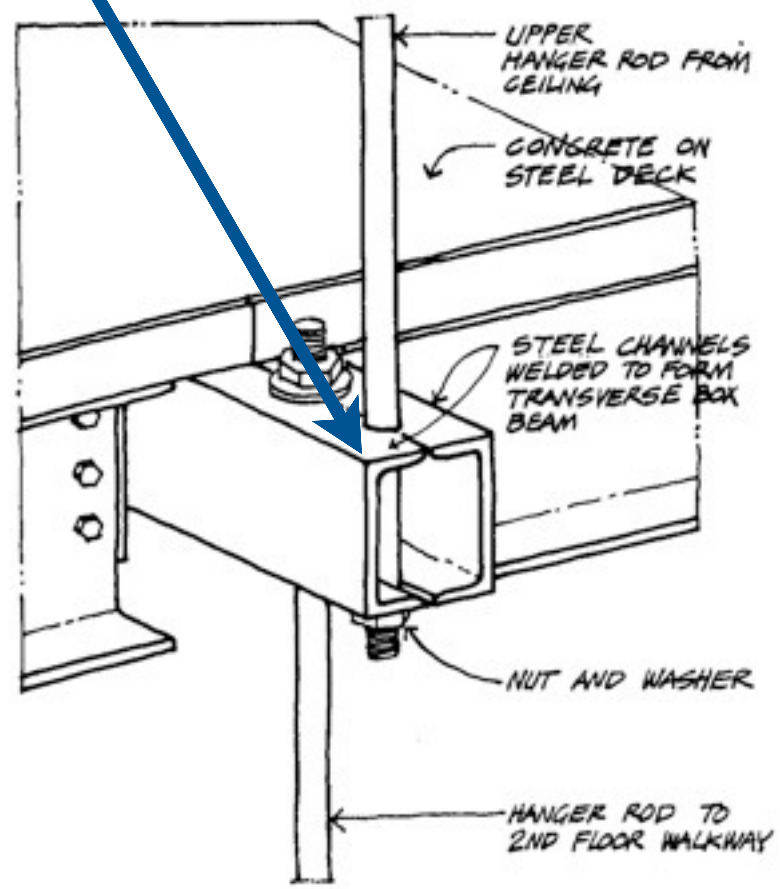
illustrations from Matthys Levy and Mario Salvadori, Why Buildings Fall Down

how it failed

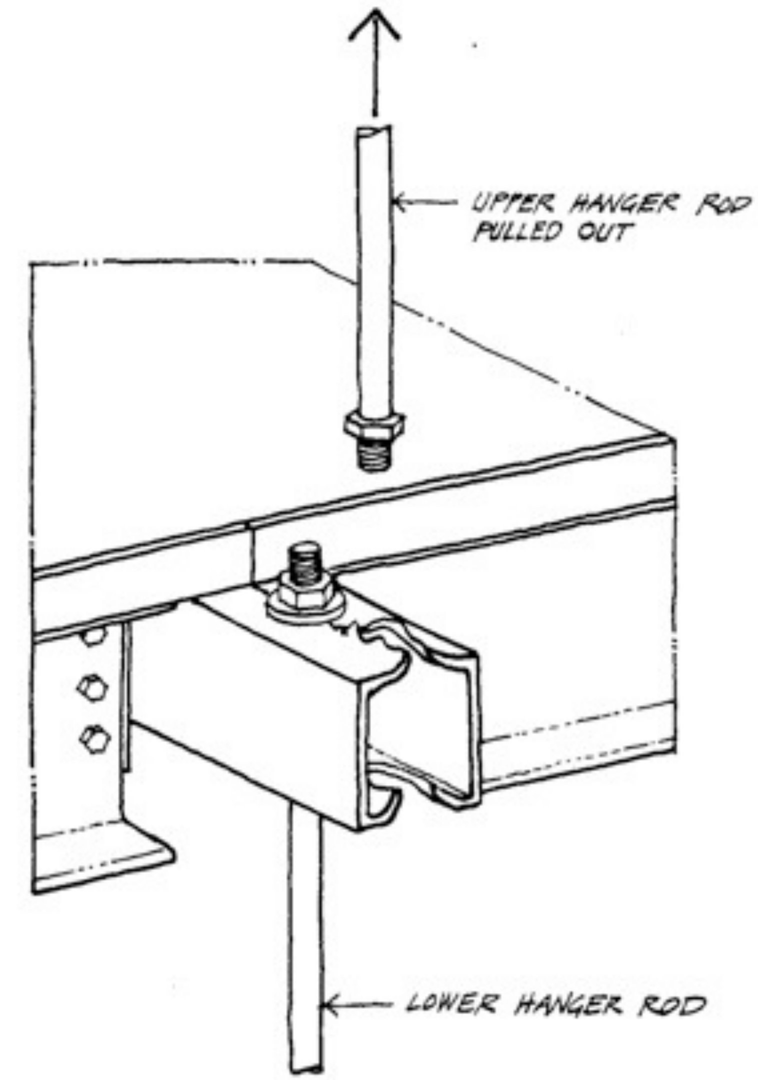
beam supports
two walkways



as designed



as implemented



what happened

therac 25

no argument for success

- AECL fault tree (1983) did not include software
- P(computer selects wrong energy) = 10^{-11}

hard to extract any lessons

- Leveson & Turner (1993): so many flaws, nothing clear

so doomed to fail again

- 17 deaths from similar machine in Panama (2001)
- 621 target/dose/patient errors (2001-9, NY state)

[2001-2009, New York Times, January 22, 2010]

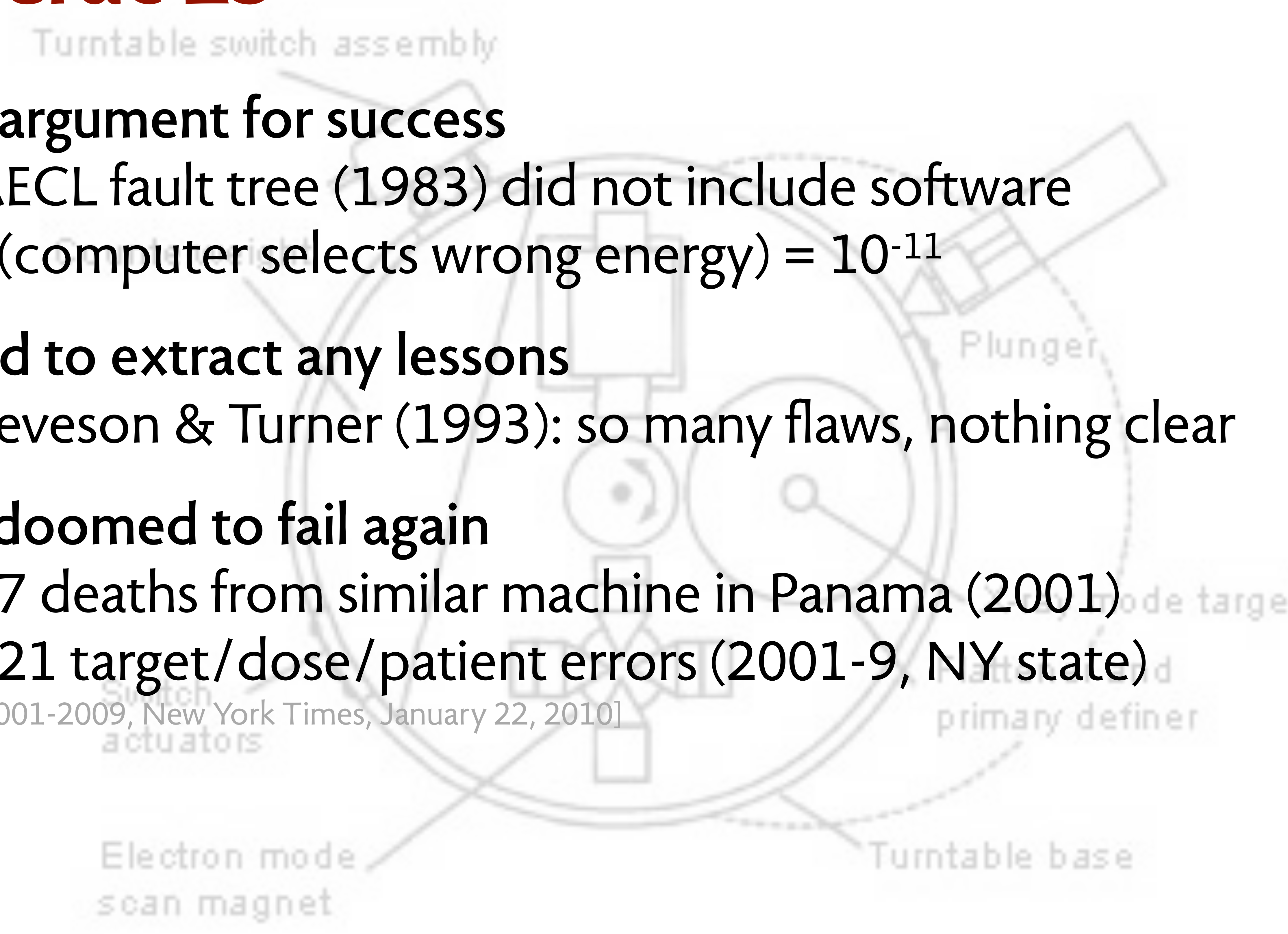


Figure B. Upper turntable assembly

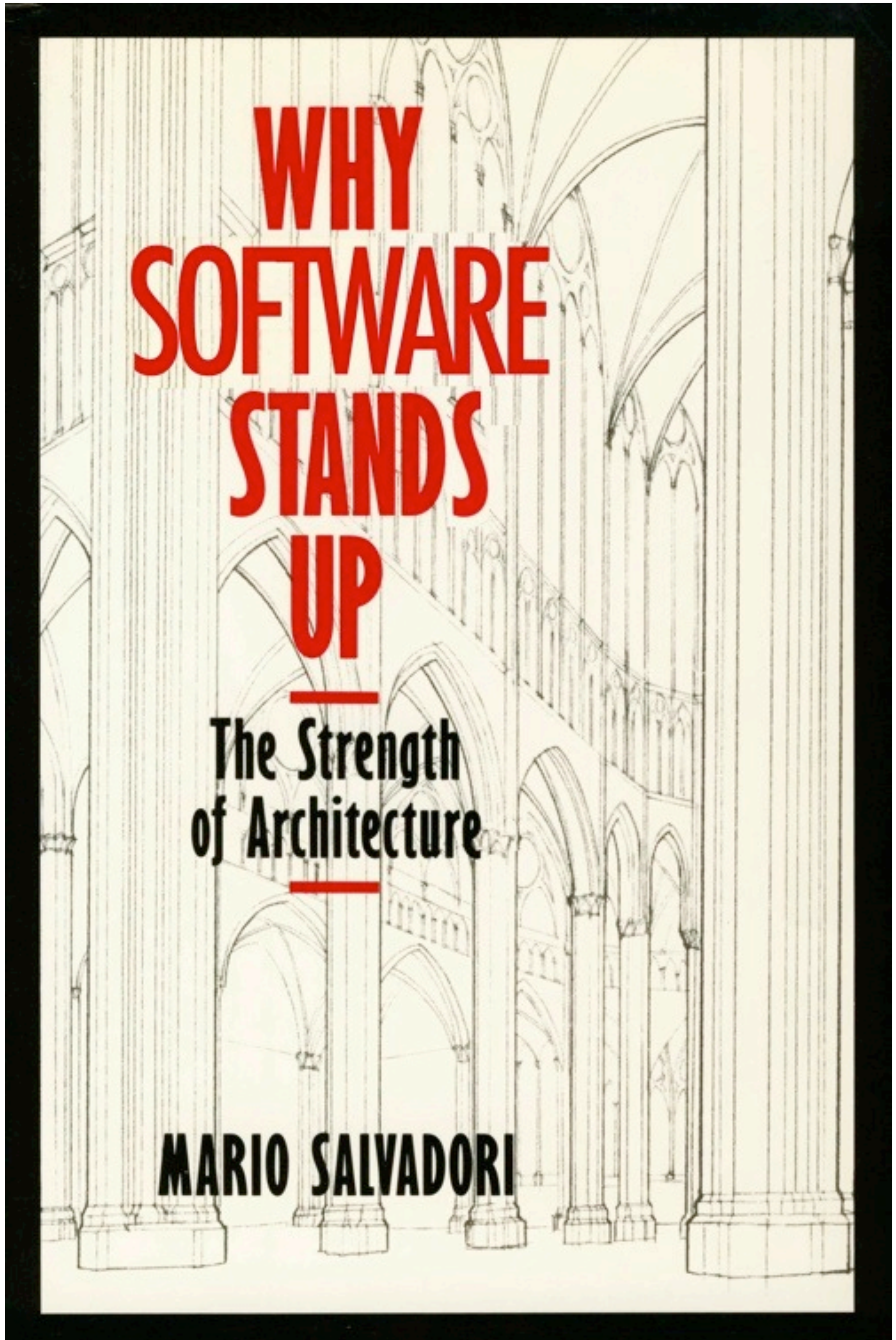
my conclusions

civil engineers

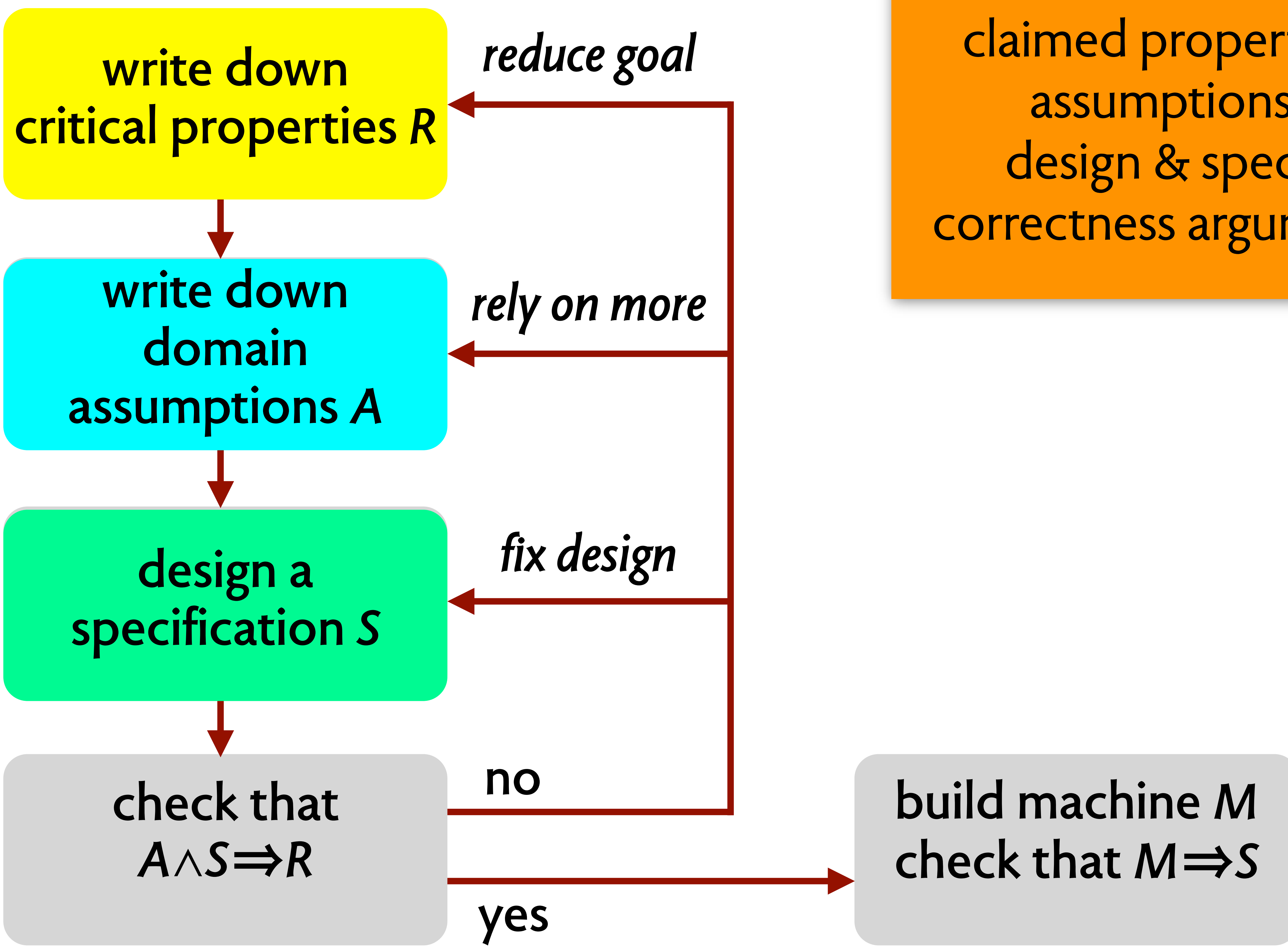
- argue why structure should stand
- failure occurs when argument is flawed

software engineers

- build and hope for the best
- when failure occurs, no story
- can't assign blame or learn for future


WHY SOFTWARE STANDS UP
The Strength of Architecture
MARIO SALVADORI

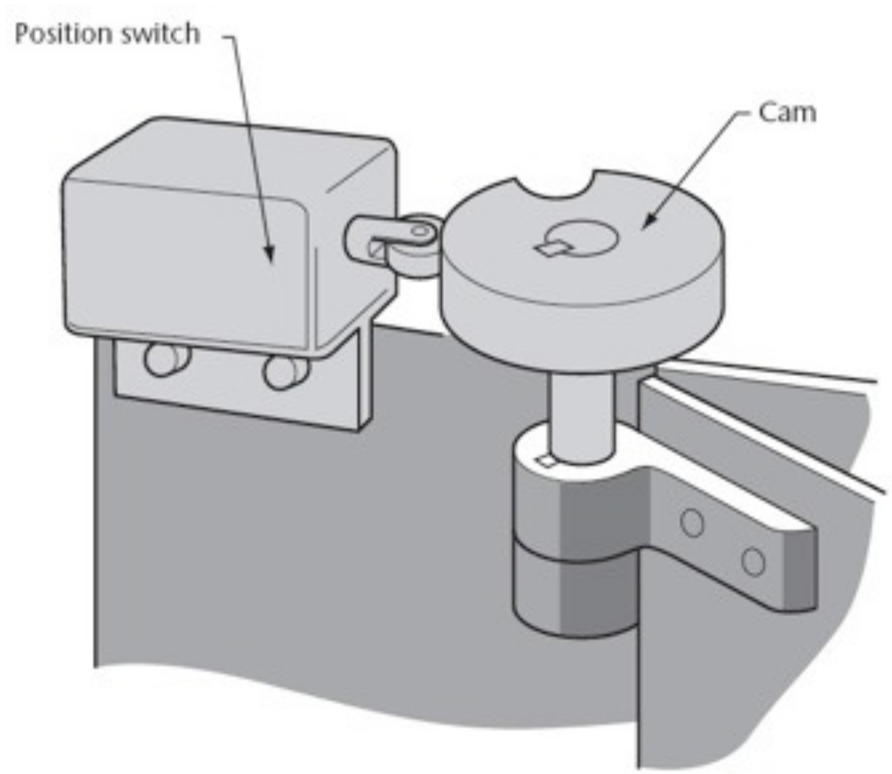
a new approach

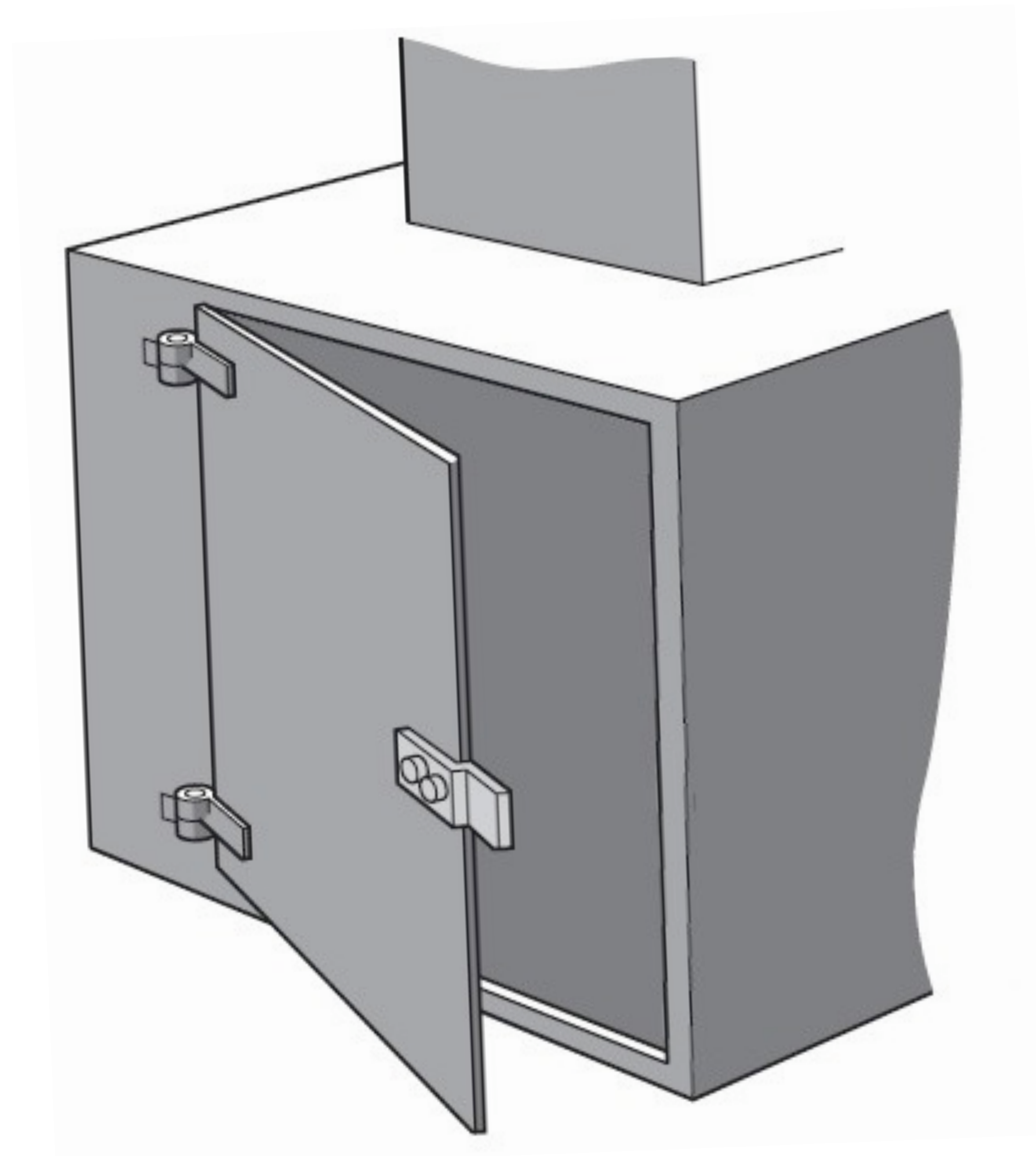


DEPENDABILITY CASE:
claimed properties
assumptions
design & specs
correctness argument

the door interlock problem

problem: design an interlock





a textbook problem

› see, eg, *Engineering a Safer World* [Leveson, 2010]

actually, a real problem

The Worlds First Microwave Test Oven

Here's a picture of the world's first commercial microwave during its first field test. I am on the left, my brother on the right. We used to defeat the door interlock and point it at the end of the countertop where we left a plate of eggs. They exploded like little hand grenades. Drove my mom nuts!

http://www.thescubalady.com/Keith%20Lamb%20History.htm

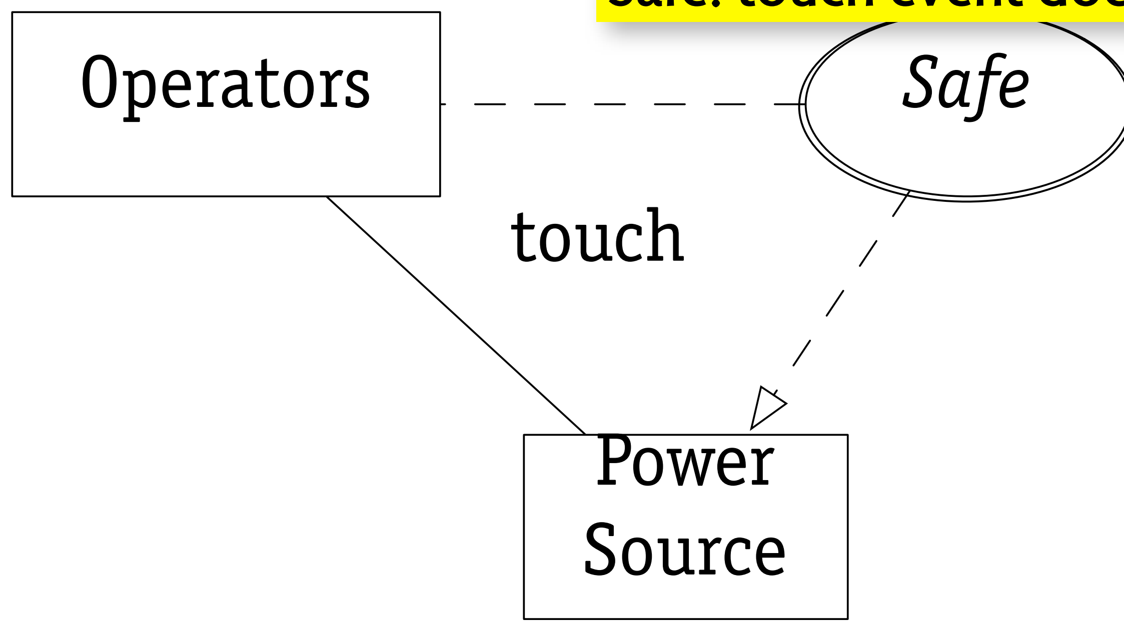
Statistics indicate that five to ten arc-flash accidents that involve a fatality or serious injury to an employee occur every day in the United States.

http://www.iaei.org/magazine/?p=1163

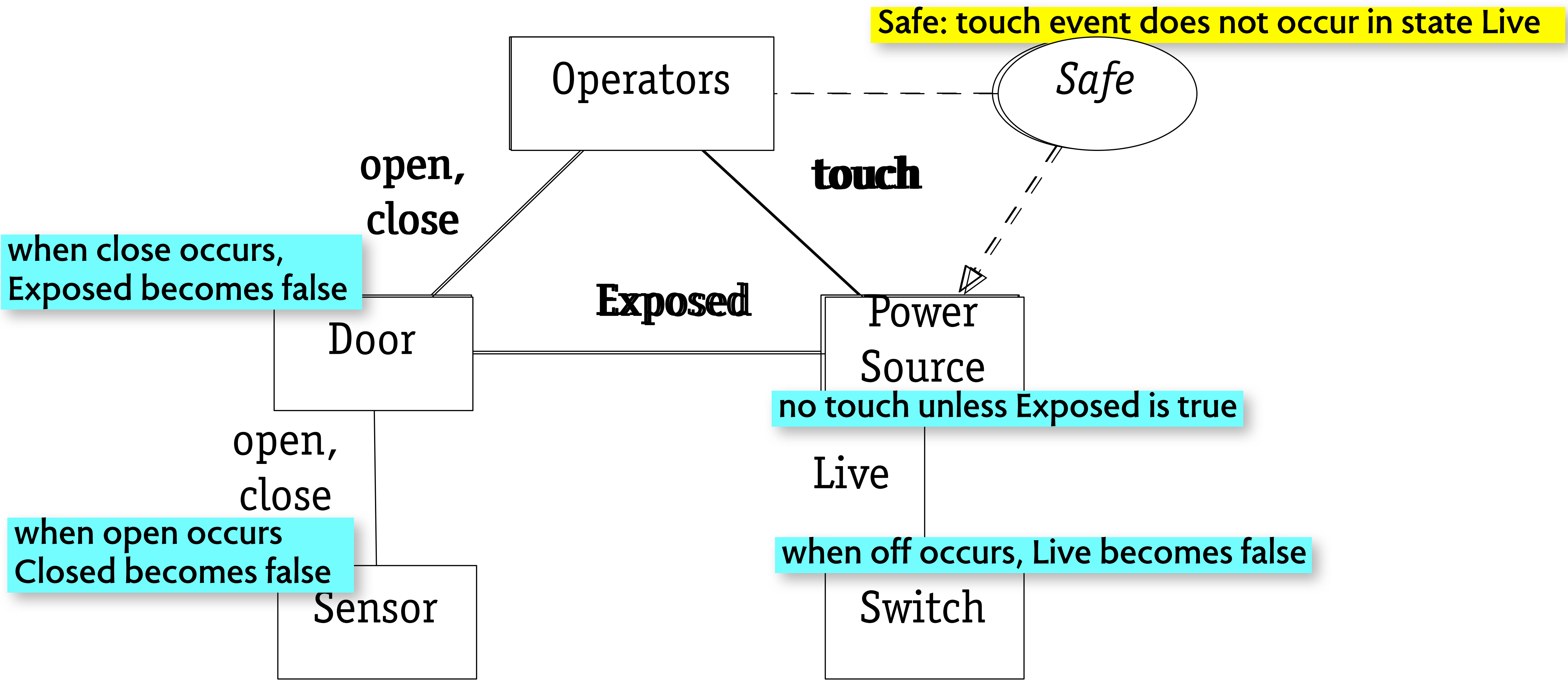
step 1: requirement

no touching live power source

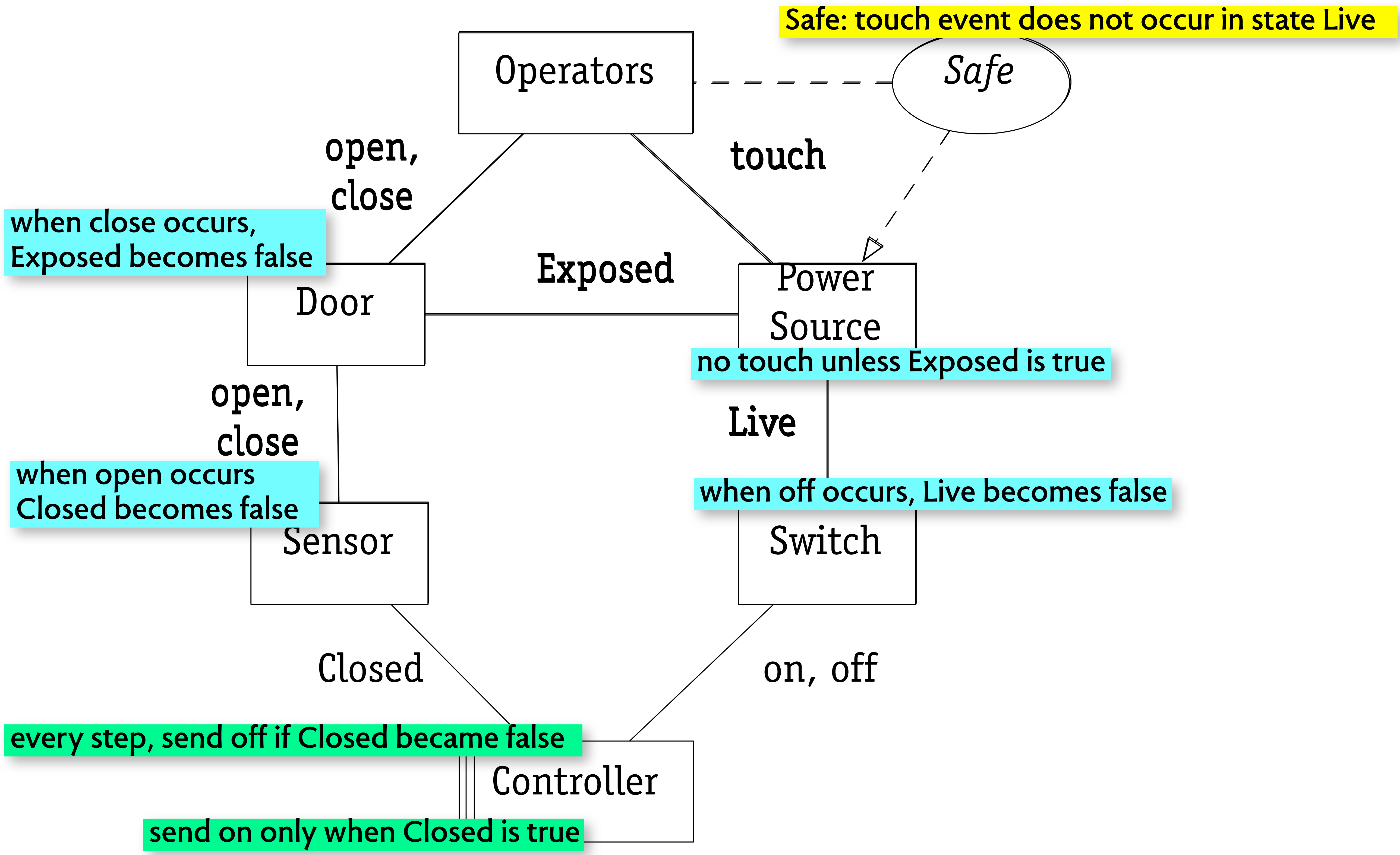
Safe: touch event does not occur in state Live



step 2: domain assumptions



step 3: machine specification



step 4: checking the system argument

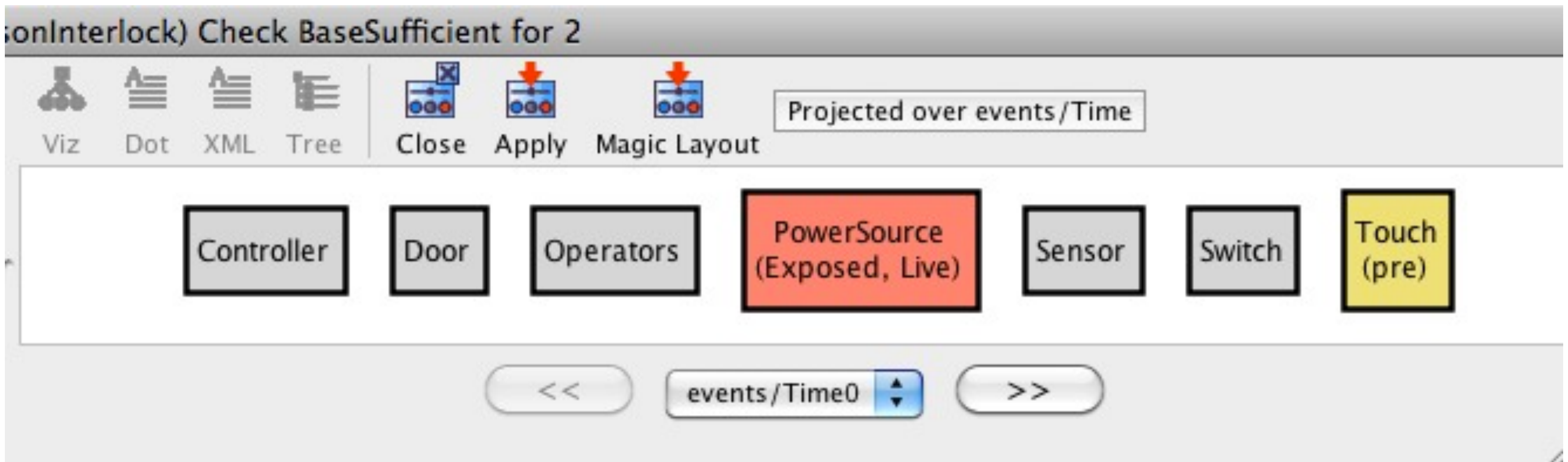
domain assumptions ∧ machine spec ⇒ requirement

```
one sig Sensor extends Domain {
    Closed: set Time
    }

one sig PowerSource extends Domain {
    Exposed, Live: set Time
    }

sig Open extends Event { } {
    not Sensor.Closed.after
    }
```

```
one sig Controller extends Domain { } {
    all t: Time - (first + last) |
        not Sensor.Closed.at [t]
        and Sensor.Closed.at [t.prev]
            implies Off.happensAt [t]
    }
```

```
one sig Safe extends Requirement {} {
    all t: Touch |
        not PowerSource.Live.before [t]
    }
```

counterexample!

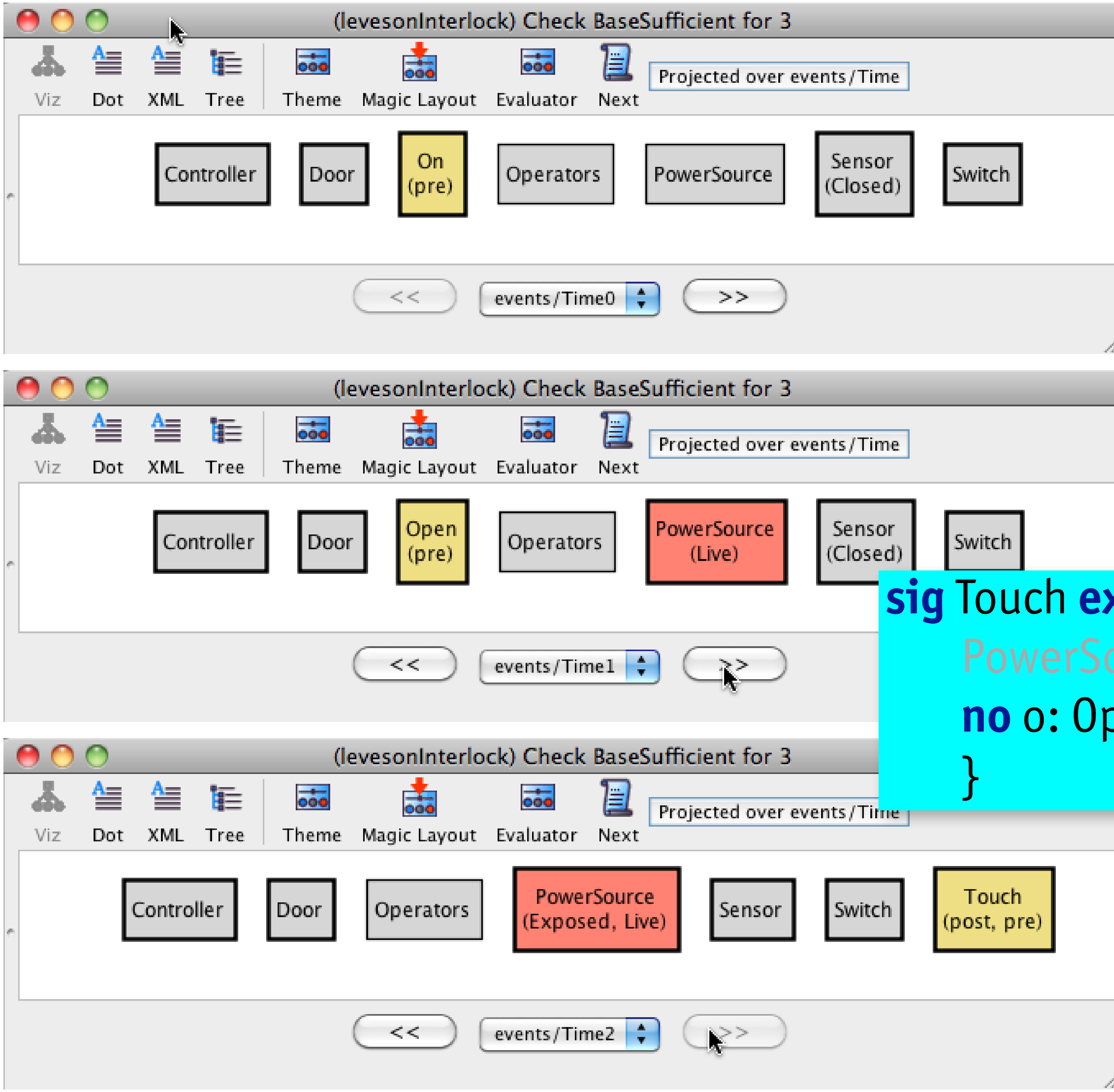



```
one sig PowerSource extends Domain {
    Exposed, Live: set Time }
    {
    not Live.initially
    not Exposed.initially
    }
```

problem:
forgot initial conditions

solution:
record them

counterexample again!



problem: controller turns power off too late

solution: new domain assumption

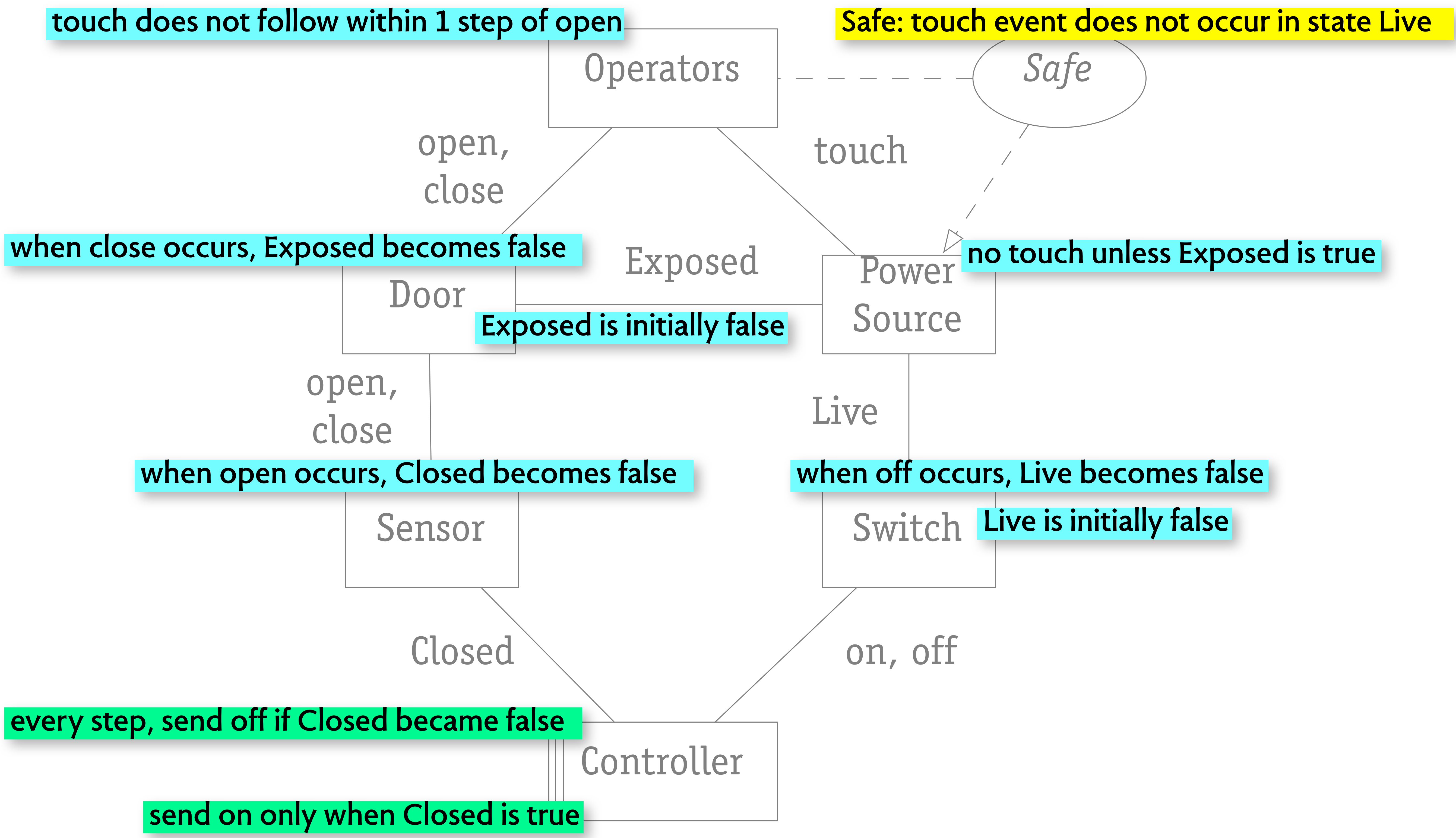
```
sig Touch extends Event { } {
    PowerSource.Exposed.before
    no o: Open | this.follows [o]
    }
```

no more counterexamples

```
Executing "Check BaseSufficient for 8"
   Solver=minisatprover(jni) Bitwidth=4 MaxSeq=7 SkolemDepth=4 Symmetry=20
   7453 vars. 366 primary vars. 14874 clauses. 427ms.
   No counterexample found. Assertion may be valid. 933ms.
   Core reduced from 25 to 11 top-level formulas. 2460ms.
```

Alloy's analysis is
- fully automatic
- large bounded space
- here, analyzed 2^{366} cases

summary


touch does not follow within 1 step of open
Safe: touch event does not occur in state Live
Operators
Safe
open,
close
touch
when close occurs, Exposed becomes false
Exposed
Power
Source
no touch unless Exposed is true
Door
Exposed is initially false
open,
close
Live
when open occurs, Closed becomes false
when off occurs, Live becomes false
Sensor
Switch
Live is initially false
Closed
on, off
every step, send off if Closed became false
Controller
send on only when Closed is true

dependability cases we've worked on

Burr Proton Therapy Center

- correct dose [Robert Seater]
- emergency stop [with Andrew Rae]
- treatment door interlock [Eunsuk Kang, Joe Near, Aleks Millicevic]

Voting systems

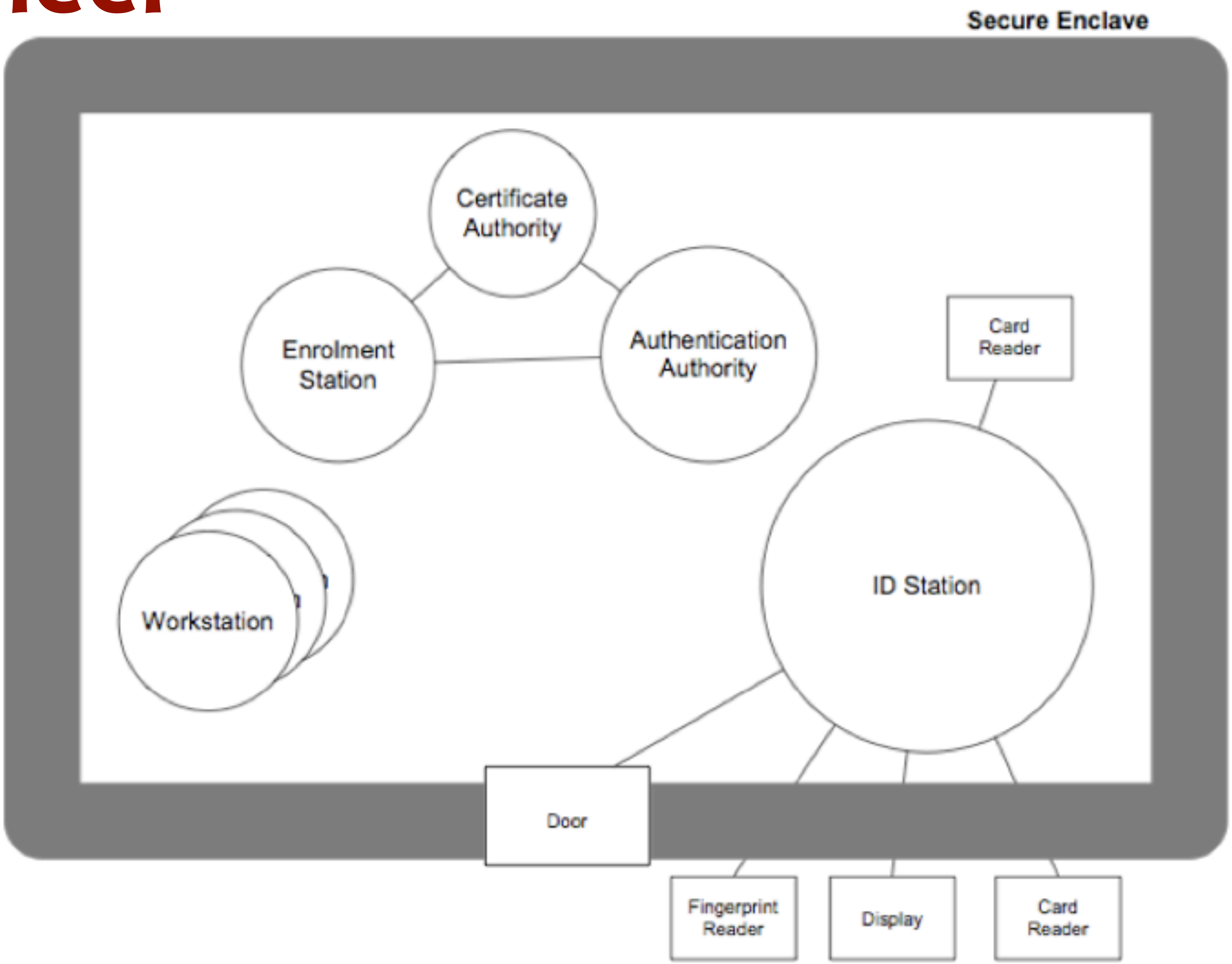
- Pret a Voter [Robert Seater]
- Scantegrity [Eunsuk Kang]

Tokeneer

- ongoing analysis [Eunsuk Kang]

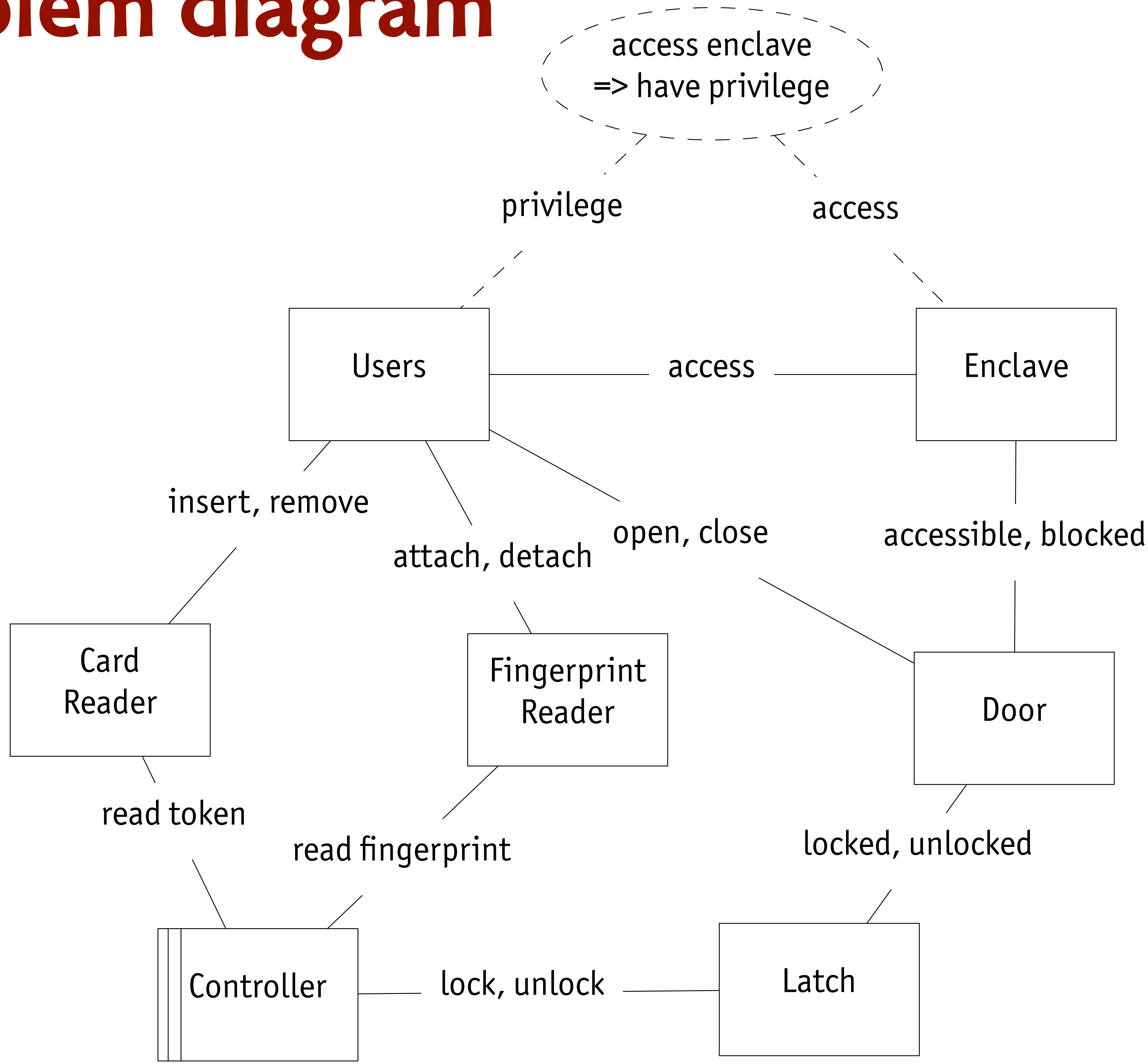
tokeneer

tokeneer



- commissioned by NSA as exemplar
- built by Praxis using Z and SPARK-Ada
- not just open source!

problem diagram



analyzing the design

what Praxis did

- formal spec in Z (about 120 pages); informal reasoning
- code verification with SPARK-Ada

defects found to date

- 5 code-level defects
- requirements issues (using Alloy for test case generation) [Aydal & Woodcock 2009]
- no defects yet found in design

what we're doing

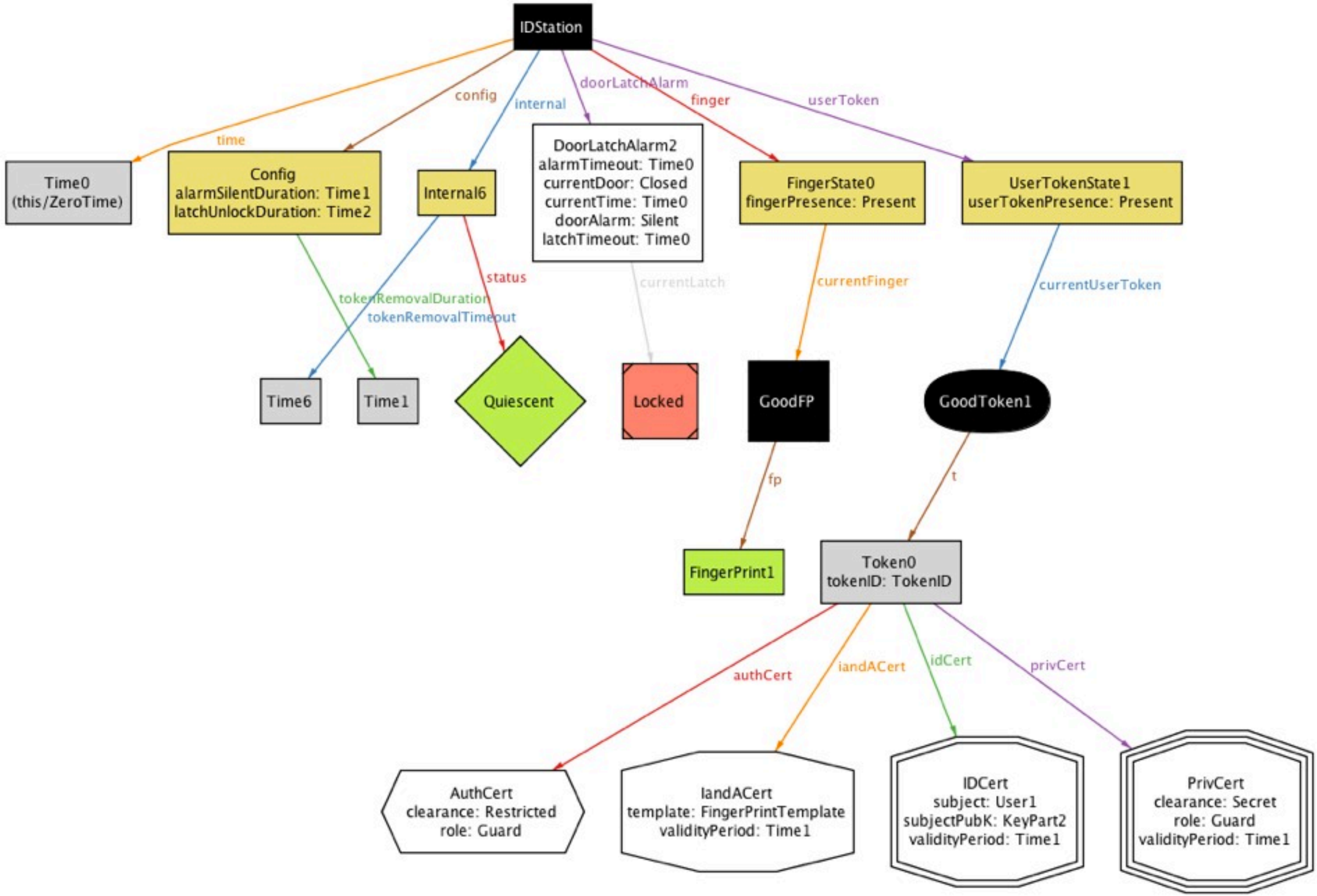
- translating design to Alloy (about 1000 lines so far)
- automatic analysis: design $\wedge$ assumptions $\Rightarrow$ security

sample argument fragments

```
sig DoorLatchAlarm {
   currentTime : Time,
   currentDoor : Door,
   currentLatch : Latch,
   doorAlarm : Alarm,
   latchTimeout : Time,
   alarmTimeout : Time
}{
   -- latch is locked when timed out
   currentLatch = Locked iff gte[currentTime, latchTimeout]

   -- door alarm goes off when the door is open but the latch is locked
   doorAlarm = Alarming iff
      (currentDoor = Open and
         currentLatch = Locked and
         gte[currentTime, alarmTimeout])
}
```

```
-- property 2 : unlock at allowed time (pg. 10, Doc 40_4)
assert UnlockAtAllowedTime {
   all s : Step |
      let s' = s.next,
         ut = IDStation.userToken.s,
         config = IDStation.config.s,
         curr = IDStation.time.s |
            -- if the latch is unlocked, then
            (some w, w' : ControlledWorld | latchUnlocked[w, w', s']) implies {
               -- the user must have a token that has "recently" been validated for an entry
               let token = ut.currentUserToken.t {
                  validToken[token]
                  some recentTime : timesRecentTo[curr, config.tokenRemovalDuration] |
                     recentTime in
                        config.entryPeriod[token.privCert.role][token.privCert.clearance]
               }
            }
}
```

sample screenshot



results so far

bug in security property

› if door is opened, user must hold token with recently validated fingerprint or valid authorization certificate

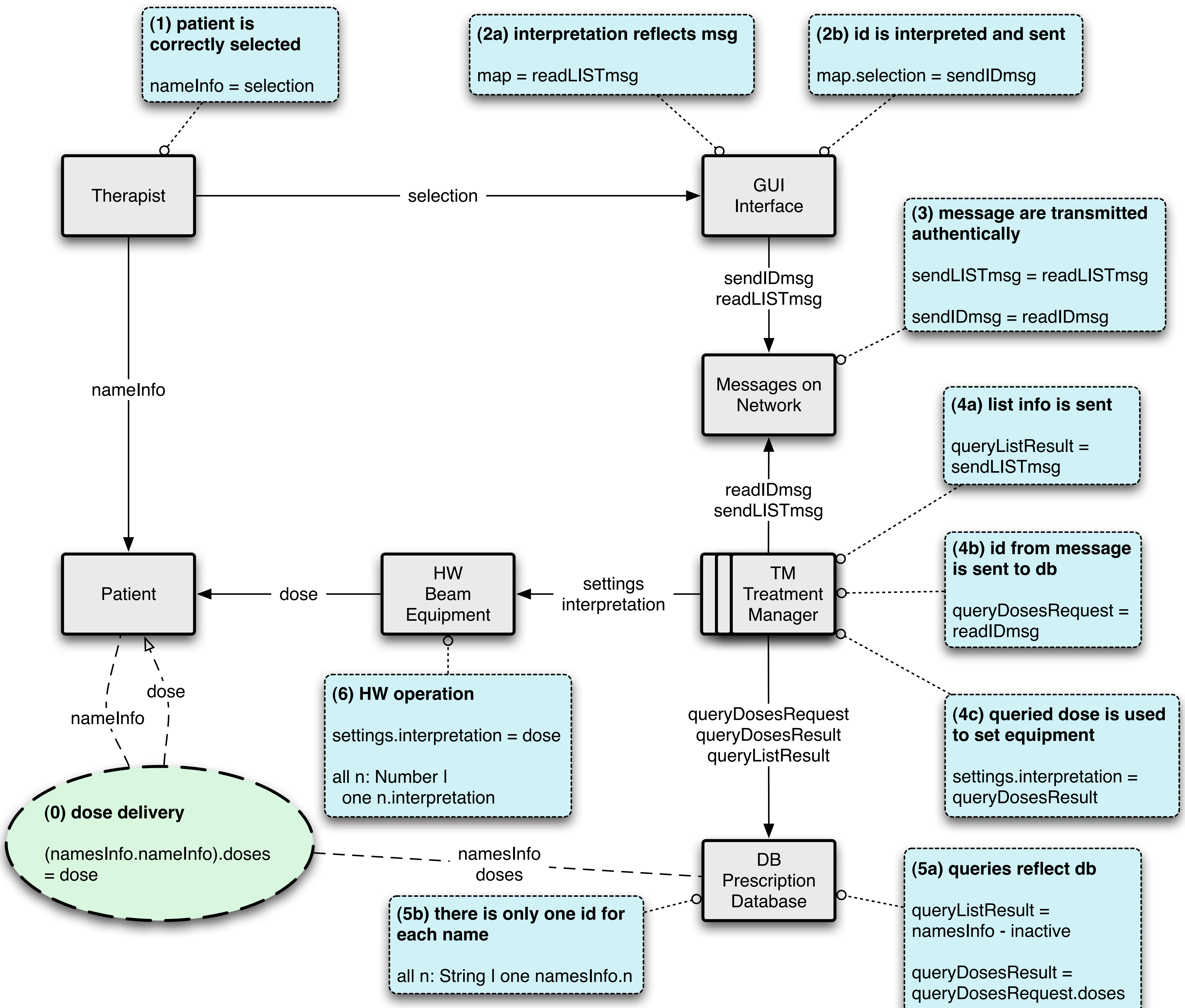
bug in spec for UnlockDoor

› timer not checked if token withdrawn after timeout

proton therapy

proton therapy treatment room

correct dose requirement



correct dose case

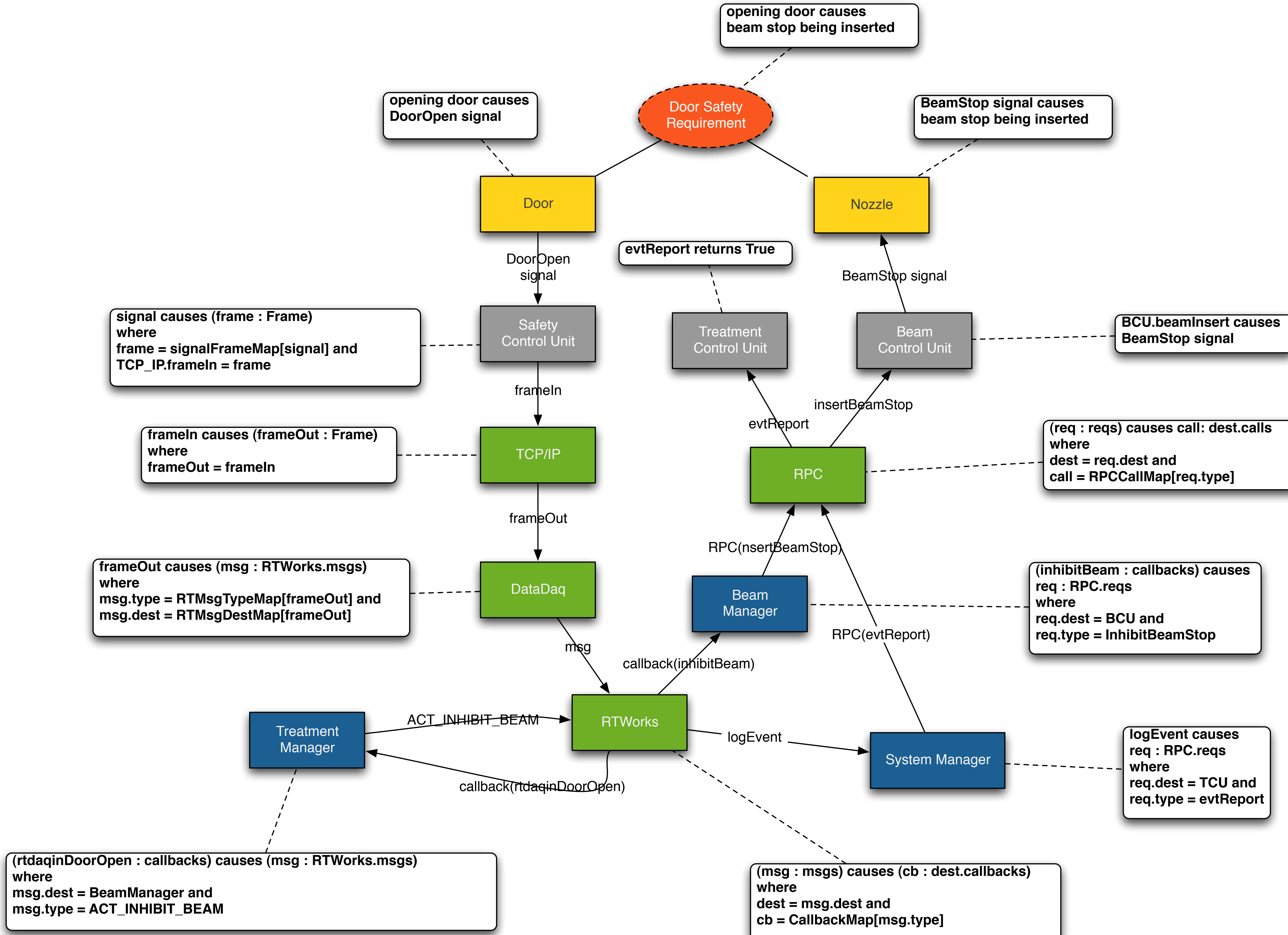
extraction of models

› Alloy models of messaging infrastructure
› C code translated to Java, then to Alloy using Forge

resulting insights

› very long message delay might cause bad dose
› patient identification relies on distinct patient names
› SQL injection attack vulnerability

door interlock requirement



door interlock case

high level analysis in Alloy

- by modelling each component
- simple chain of events

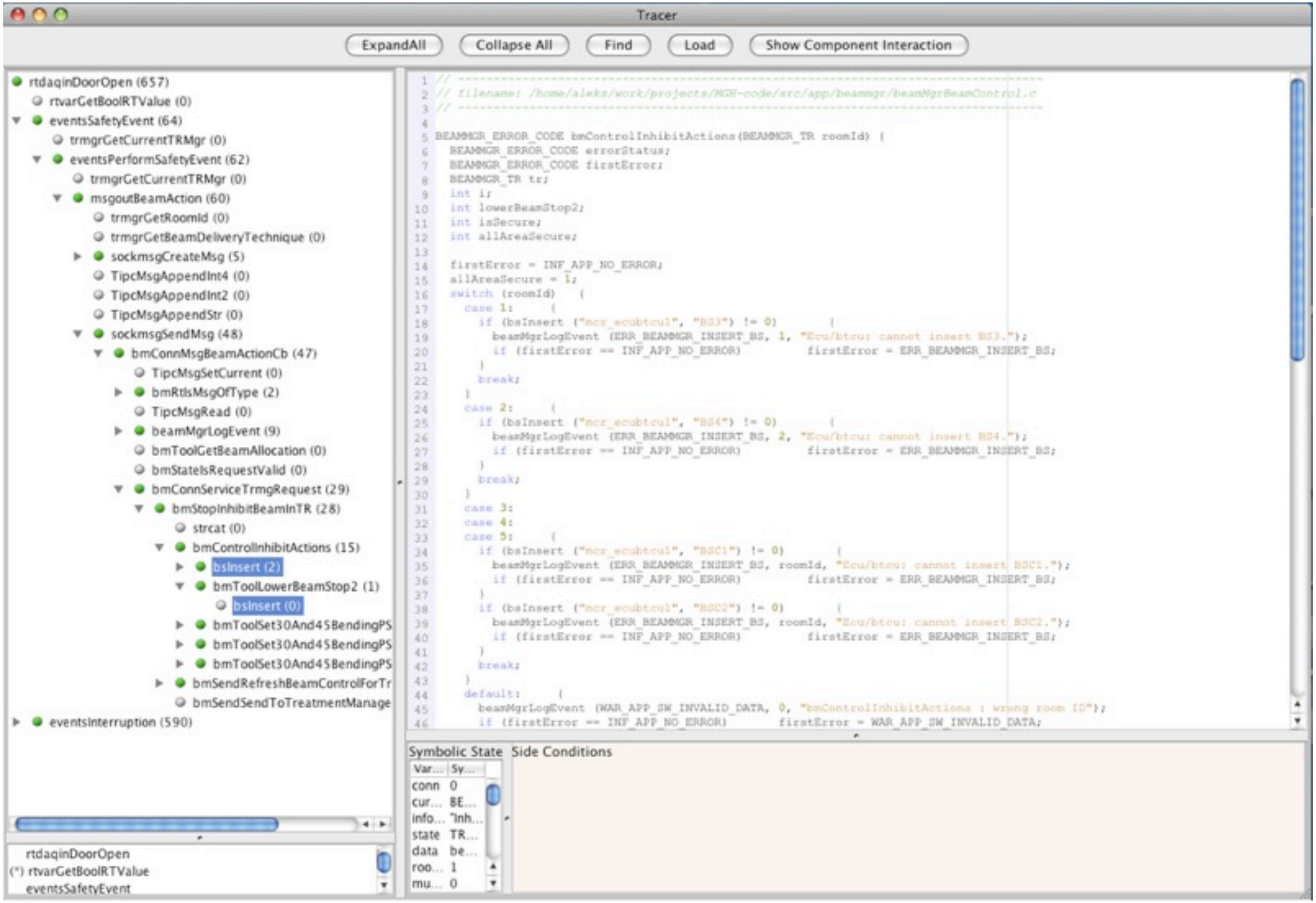
code analysis

- to identify side conditions
- to extract control paths
- but hard due to missing code

approach

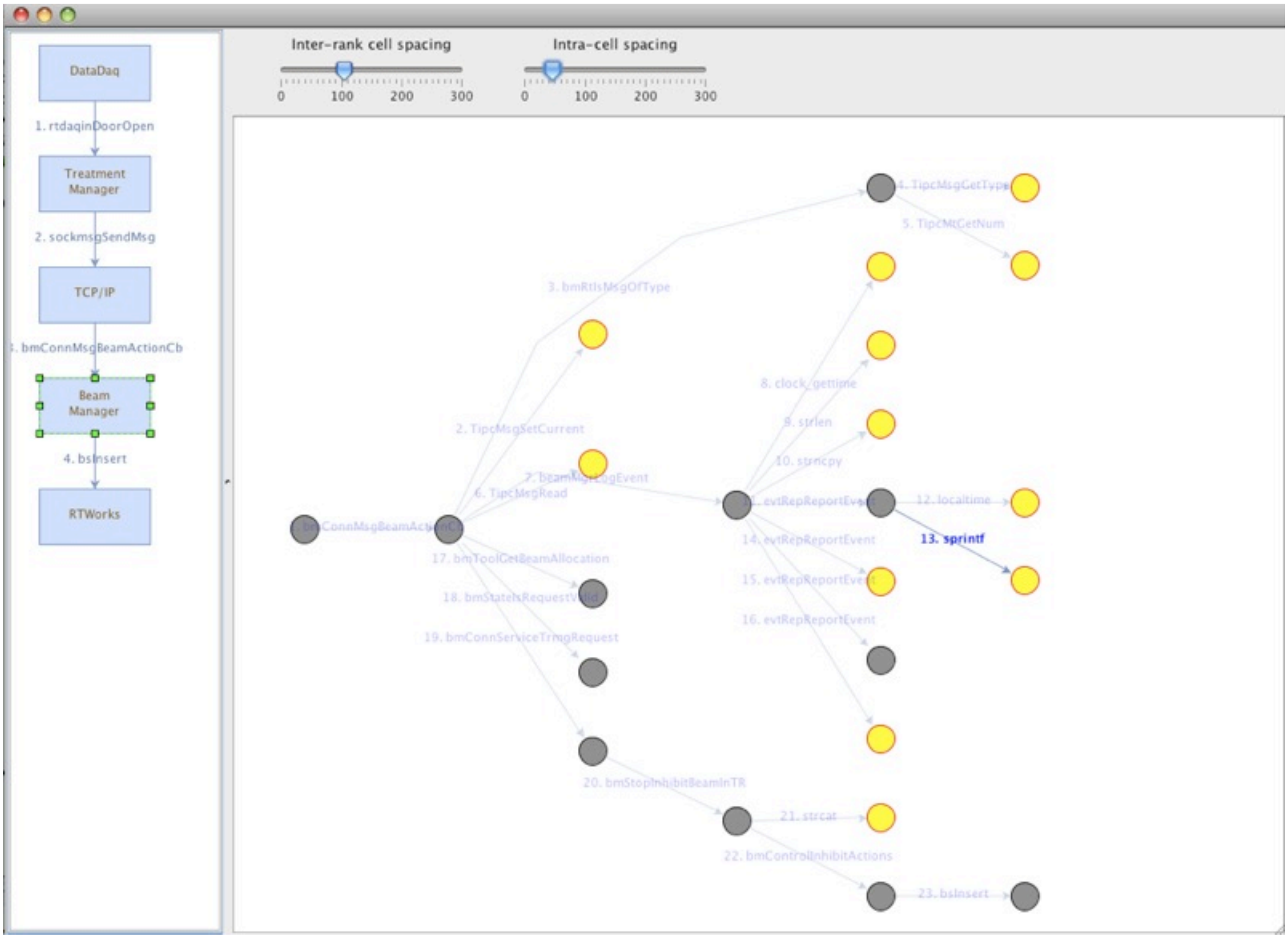
- lightweight extraction of control flow
- abstract interpretation of state
- user provides specs for library calls

tracing call paths



tool and analysis by Aleks Millicevic

tracing calls within a component



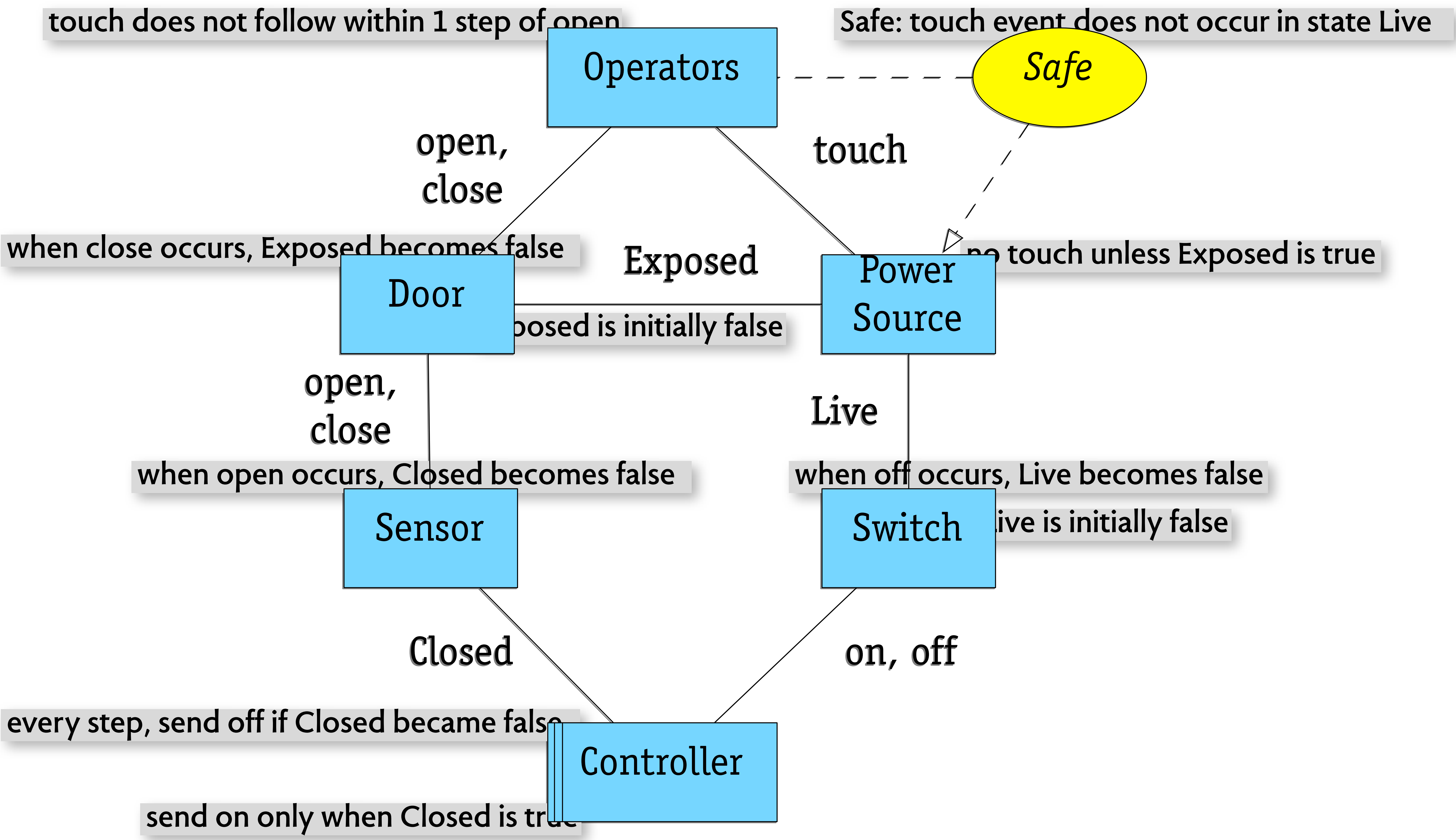
results so far

entanglement

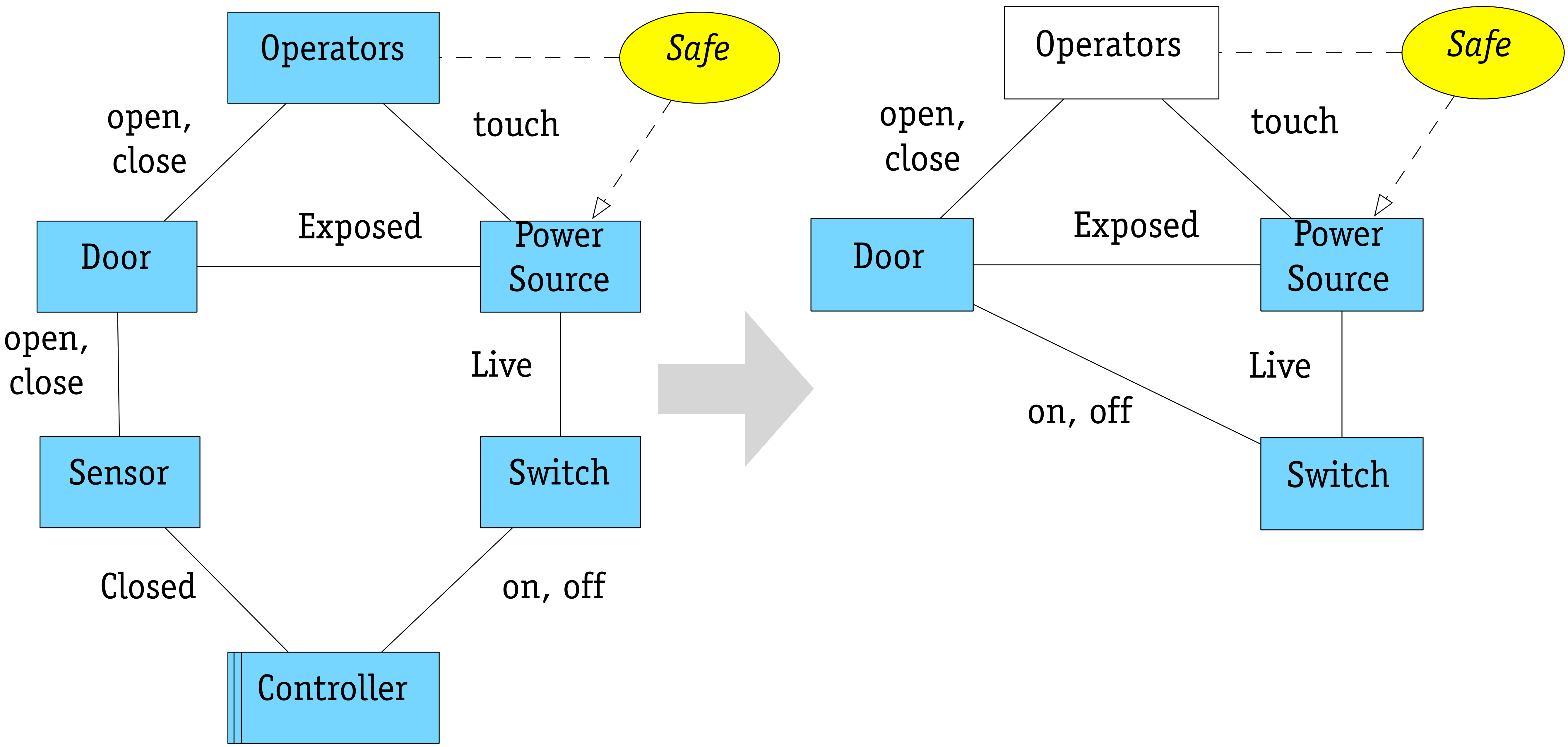
- door safety entangled with logging
- if logging fails, safety action is aborted
- (but hardware safety system…)

how to cheat

identifying the trusted base



reducing the trusted base



simpler design ⇒ simpler argument

analysis with trusted bases

```
one sig Sensor extends Domain {
    Closed: set Time
    }
sig Open extends Event { } {
    Sensor in OK implies
        not Sensor.Closed.after
    }
```

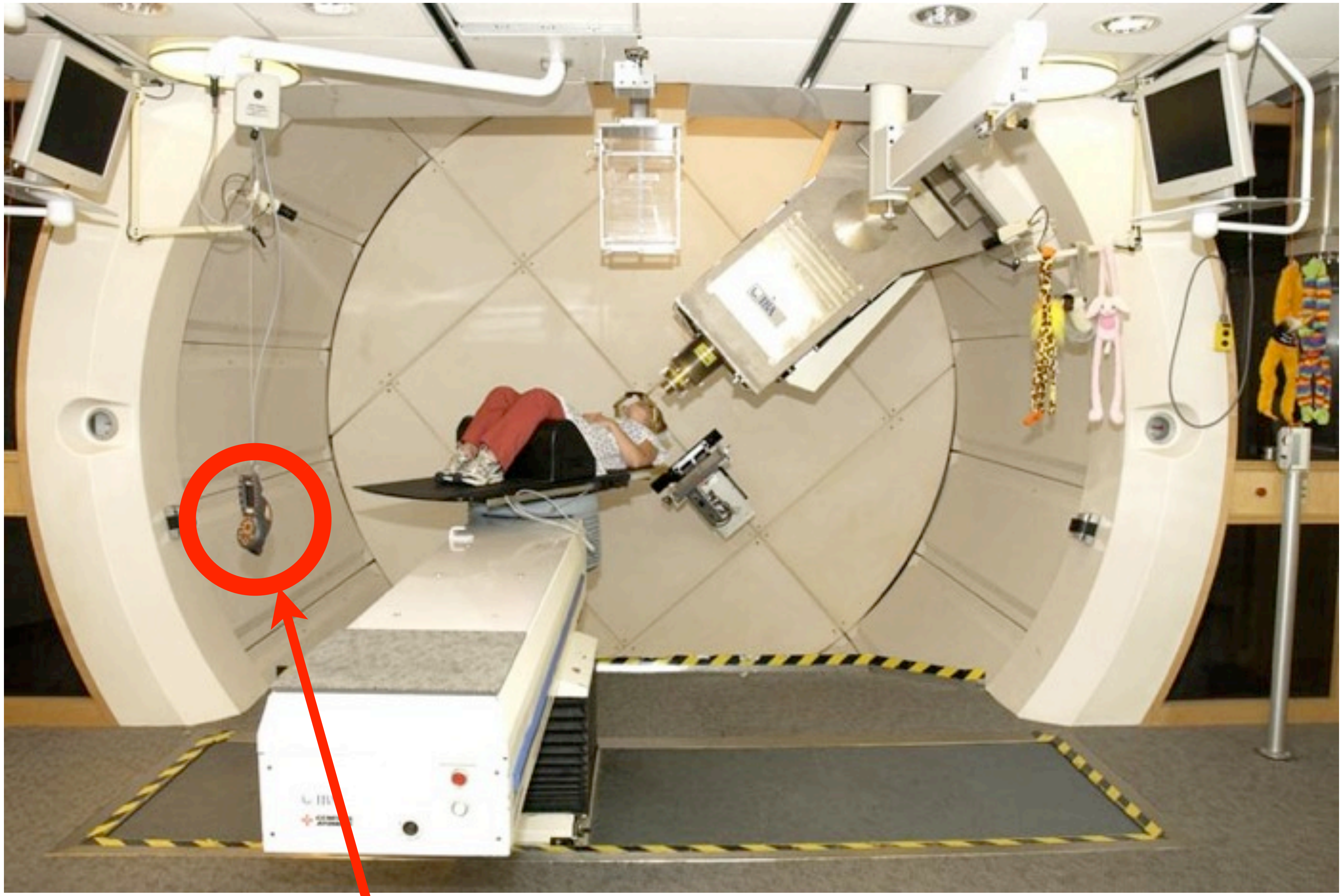
```
one sig Controller extends Domain { } {
    this in OK implies
        all t: Time - (first + last) |
            not Sensor.Closed.at [t]
            and Sensor.Closed.at [t.prev]
                implies Off.happensAt [t]
    }
```

```
one sig Safe extends Requirement {} {
    this in OK iff
        all t: Touch | not PowerSource.Live.before [t]
    trustedBase = Switch + Controller + Sensor + Door + Operators
    }
```

```
assert BaseSufficient {
    all r: Requirement | r.trustedBase in OK implies r in OK
    }
```

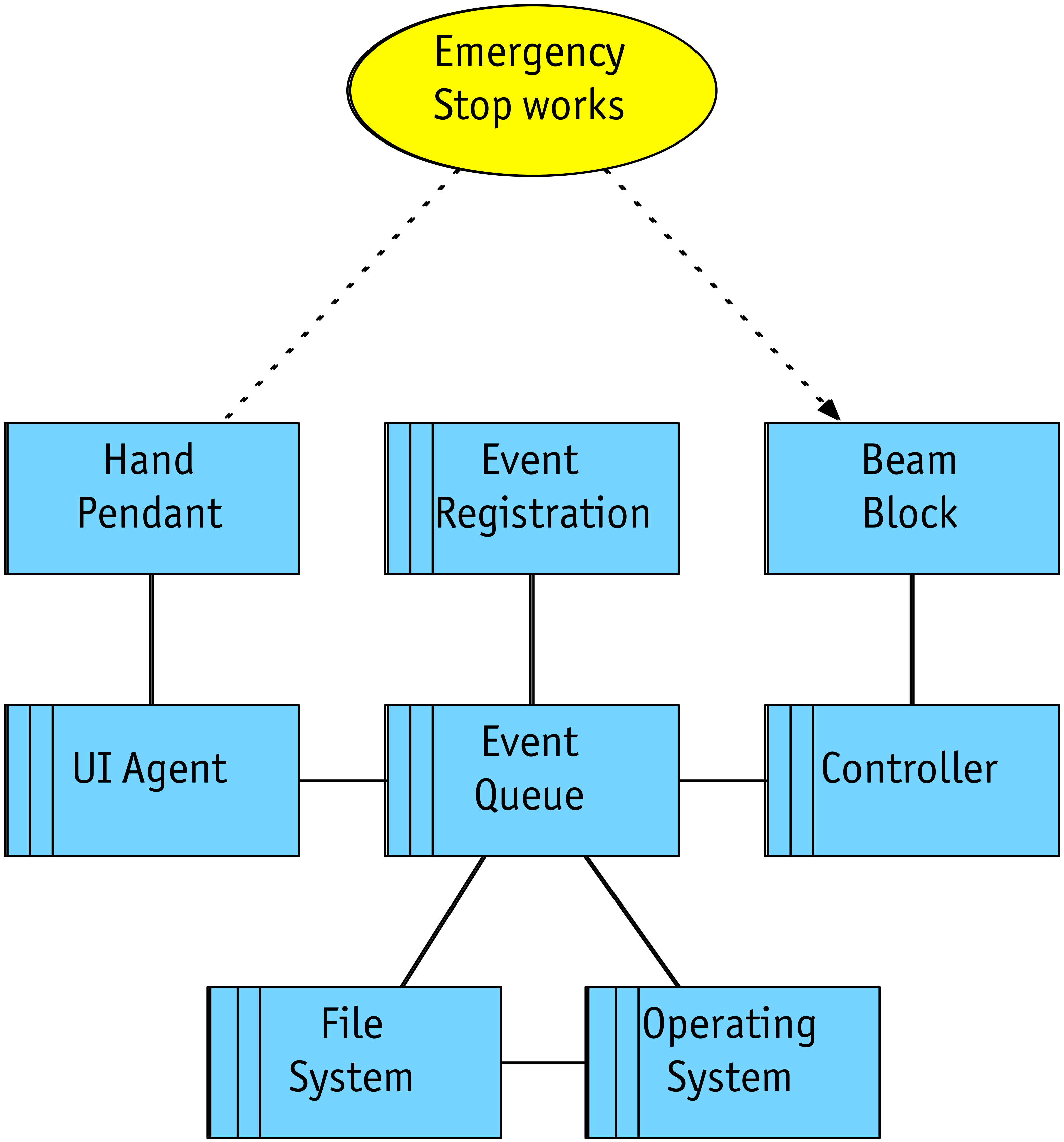
reducing the trusted base: examples

designing emergency stop

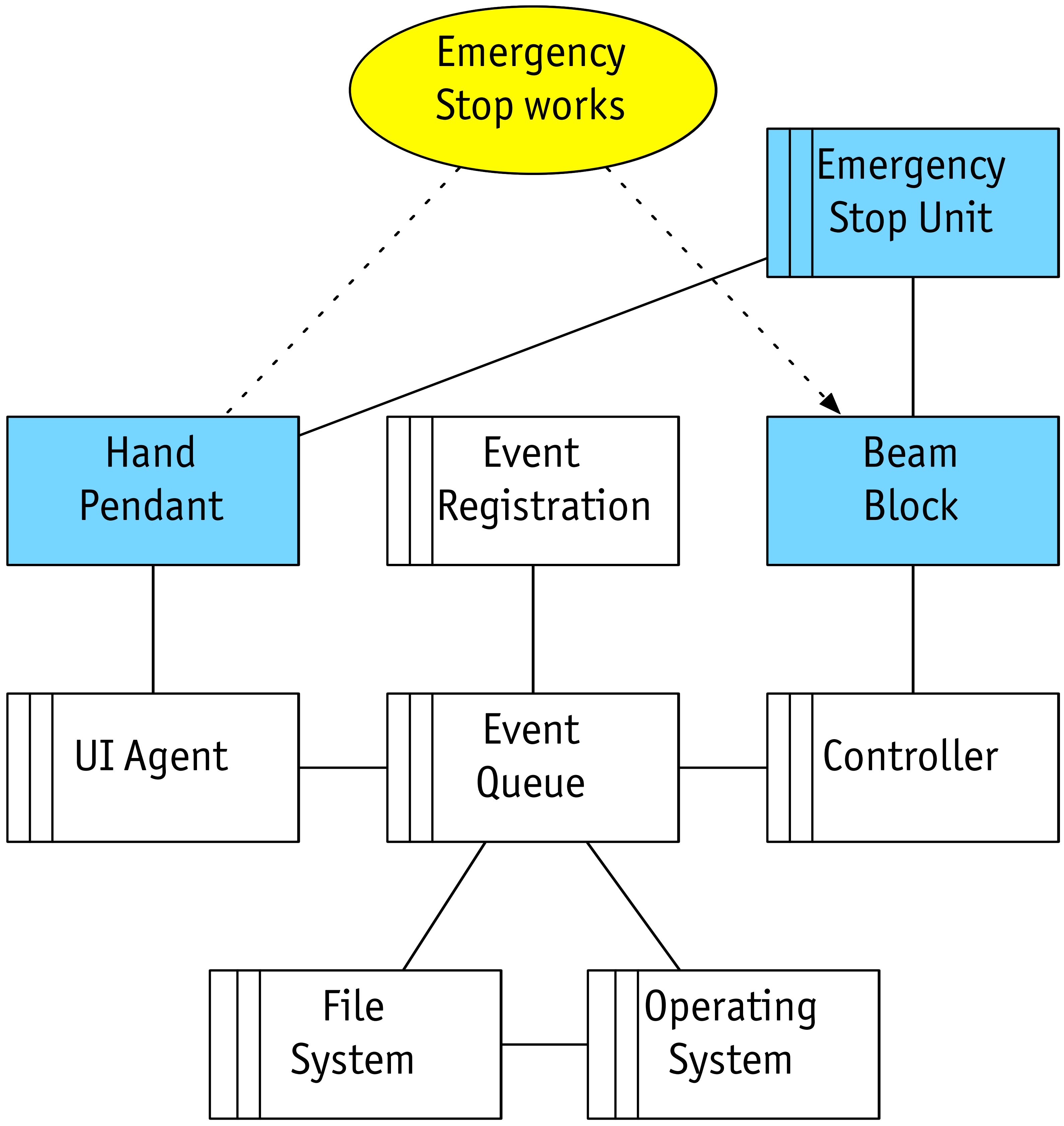


pendant with emergency stop button

existing design

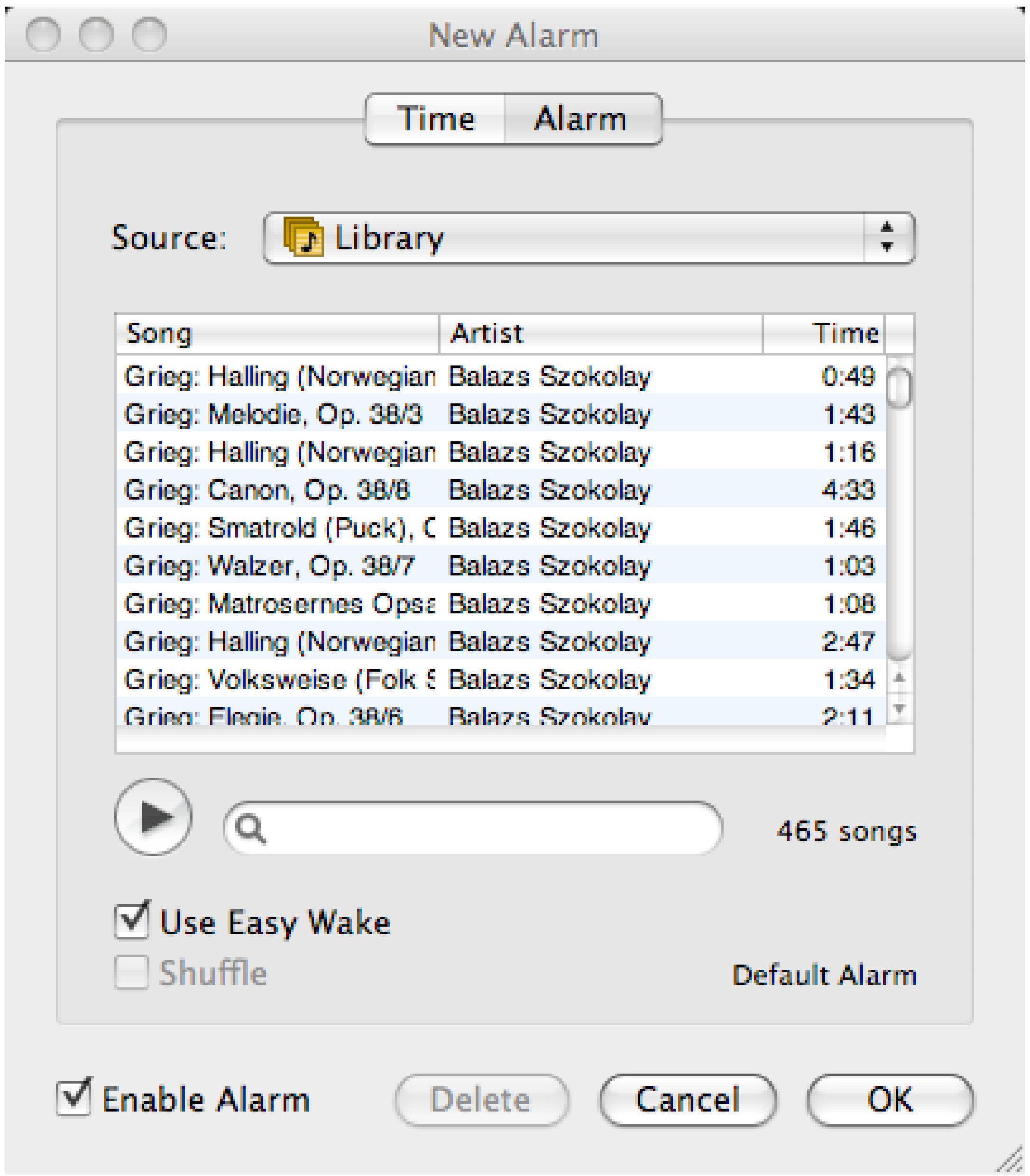


redesign



Emergency Stop works
Emergency Stop Unit
Hand Pendant
Event Registration
Beam Block
UI Agent
Event Queue
Controller
File System
Operating System

alarm clock



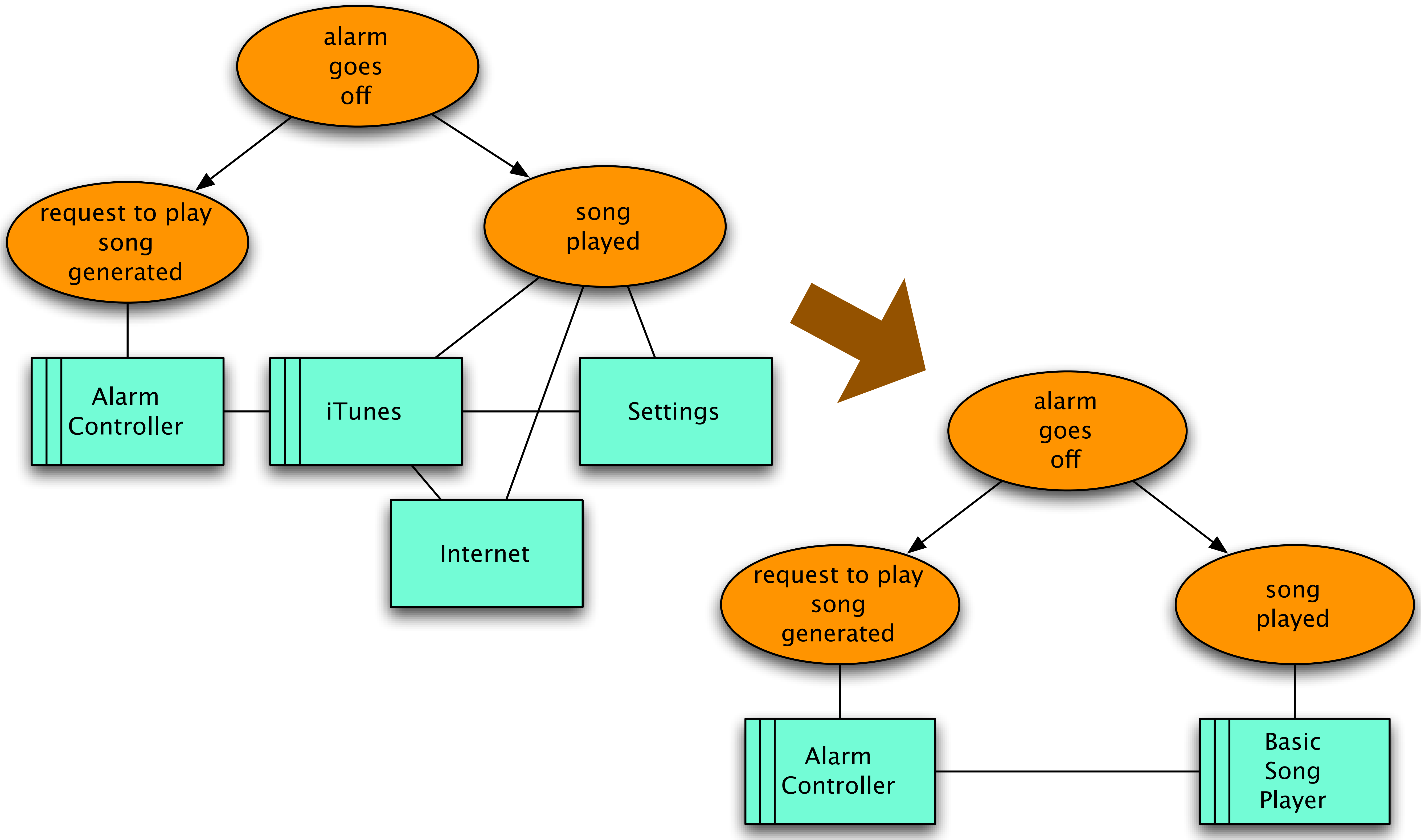
... It's only job is to wake you up in the morning, and I believe you'll find that it does it's job perfectly.

Most other alarm clock applications choose to play the alarms/music via iTunes (via AppleScript). I deliberately decided against this... Consider...

- The alarm is set to play a specific song, but the song was deleted.
- The alarm is set to play a specific playlist, but you renamed the playlist, or deleted it.
- The alarm is set to play a radio station, but the internet is down.
- iTunes was recently upgraded, and requires you to reagree to the license next time you launch it. The alarm application launches it for the alarm...
- You had iTunes set to play to your airTunes speakers, but you left your airport card turned off.
- You had the iTunes preference panel open. (Which prevents AppleScript from working)
- You had a "Get Info" panel open. (Which also prevents AppleScript from working)

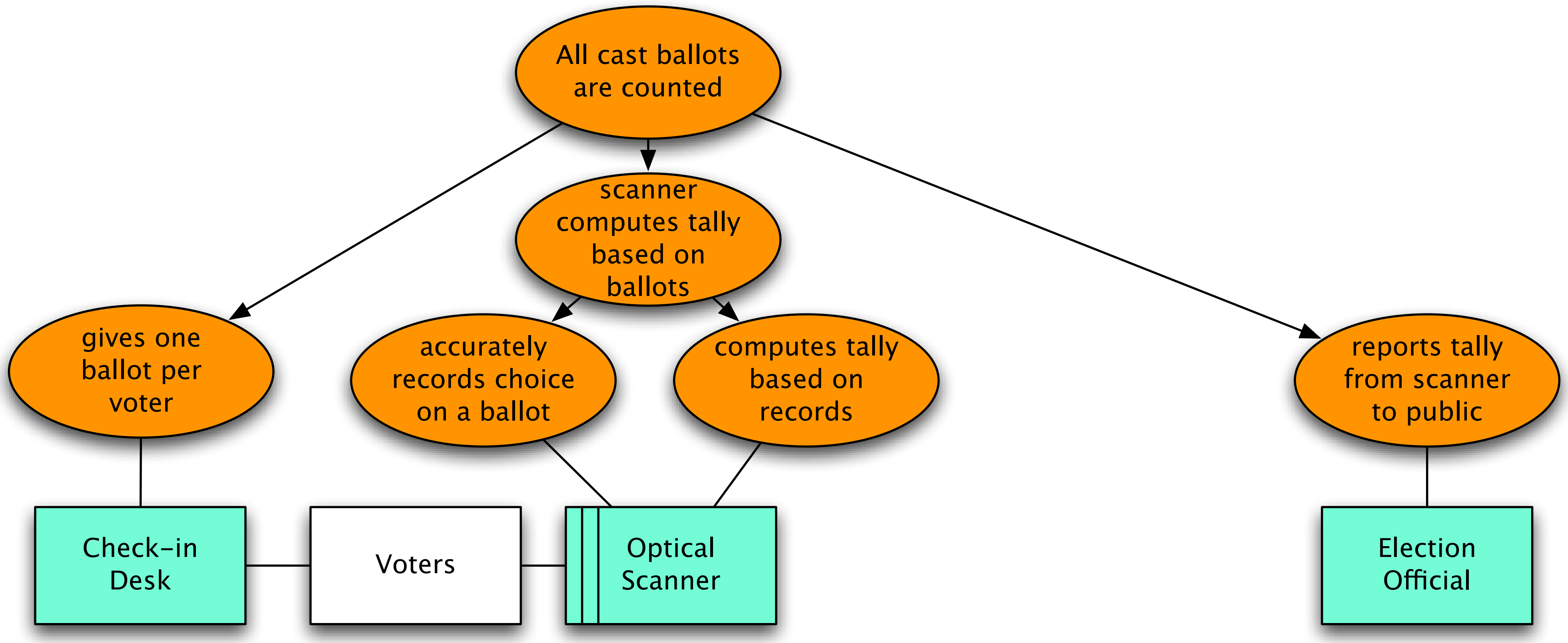
From Alarm Clock, http://www.robbiehanson.com/alarmclock/faq.html

alarm clock

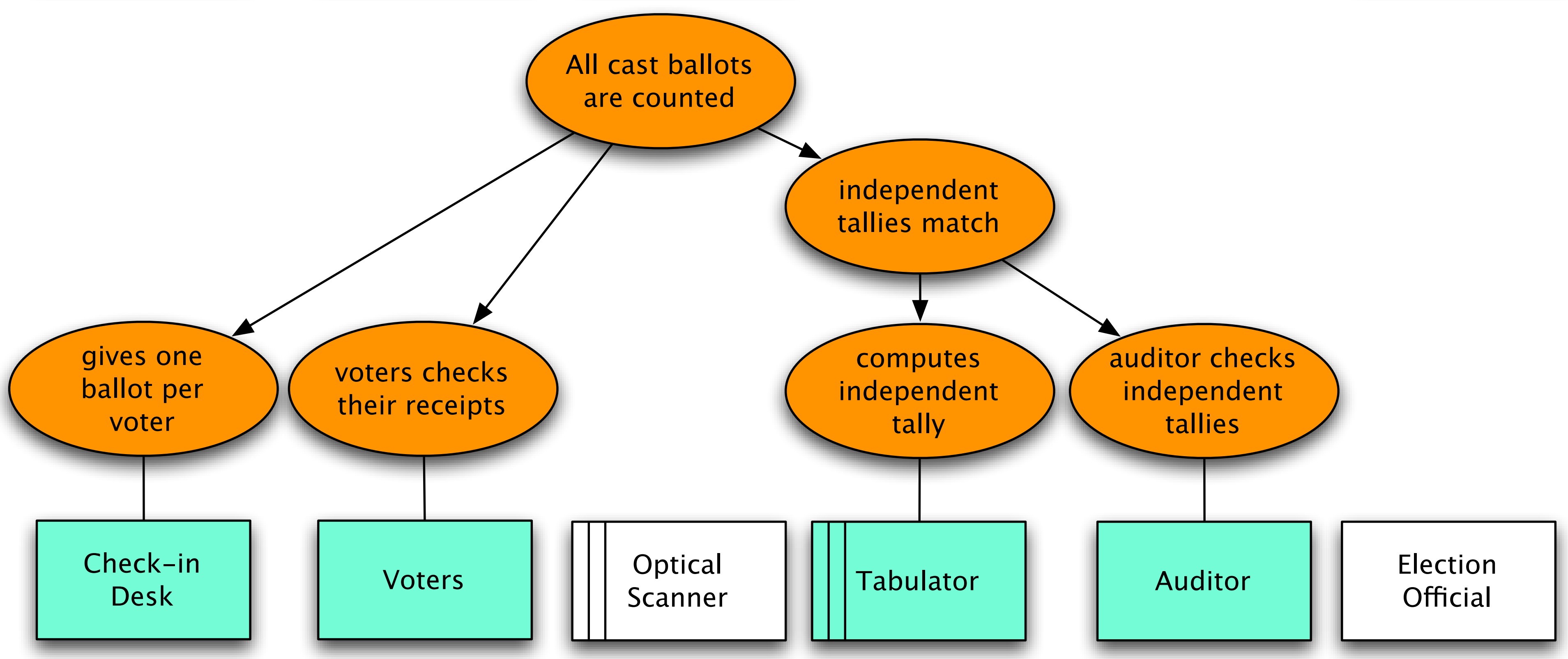


From Alarm Clock, http://www.robbiehanson.com/alarmclock/faq.html

example: voting



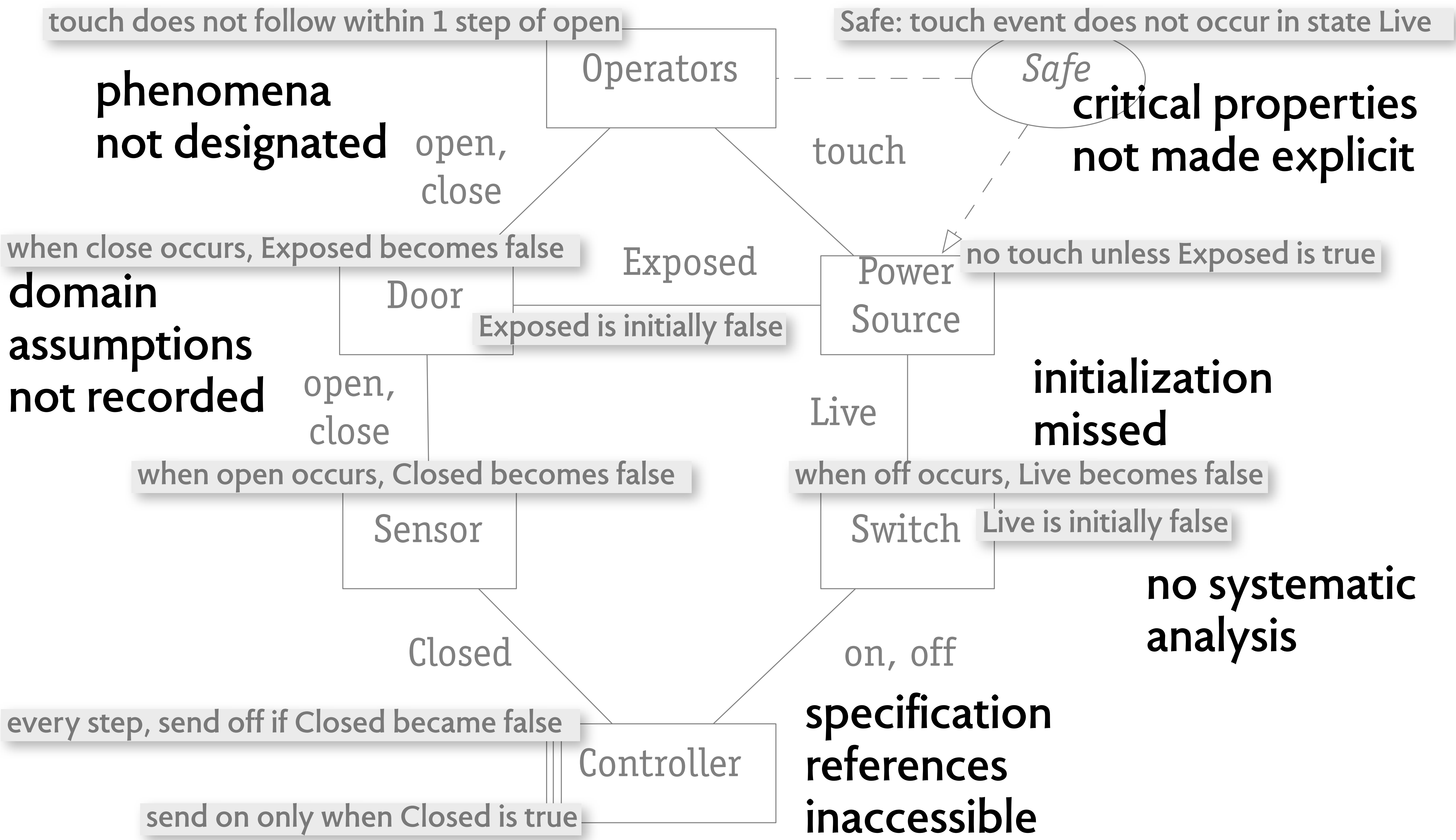
standard design, relying on scanner



Scantegrity design, relying on voters and 3rd party tabulators

conclusions

what's typically (not) done



phenomena not designated

critical properties not made explicit

domain assumptions not recorded

initialization missed

no systematic analysis

specification references inaccessible phenomena

observations

on dependability cases

› if you can’t say why it works, it probably doesn’t

on design

› a principle: design for simple argument

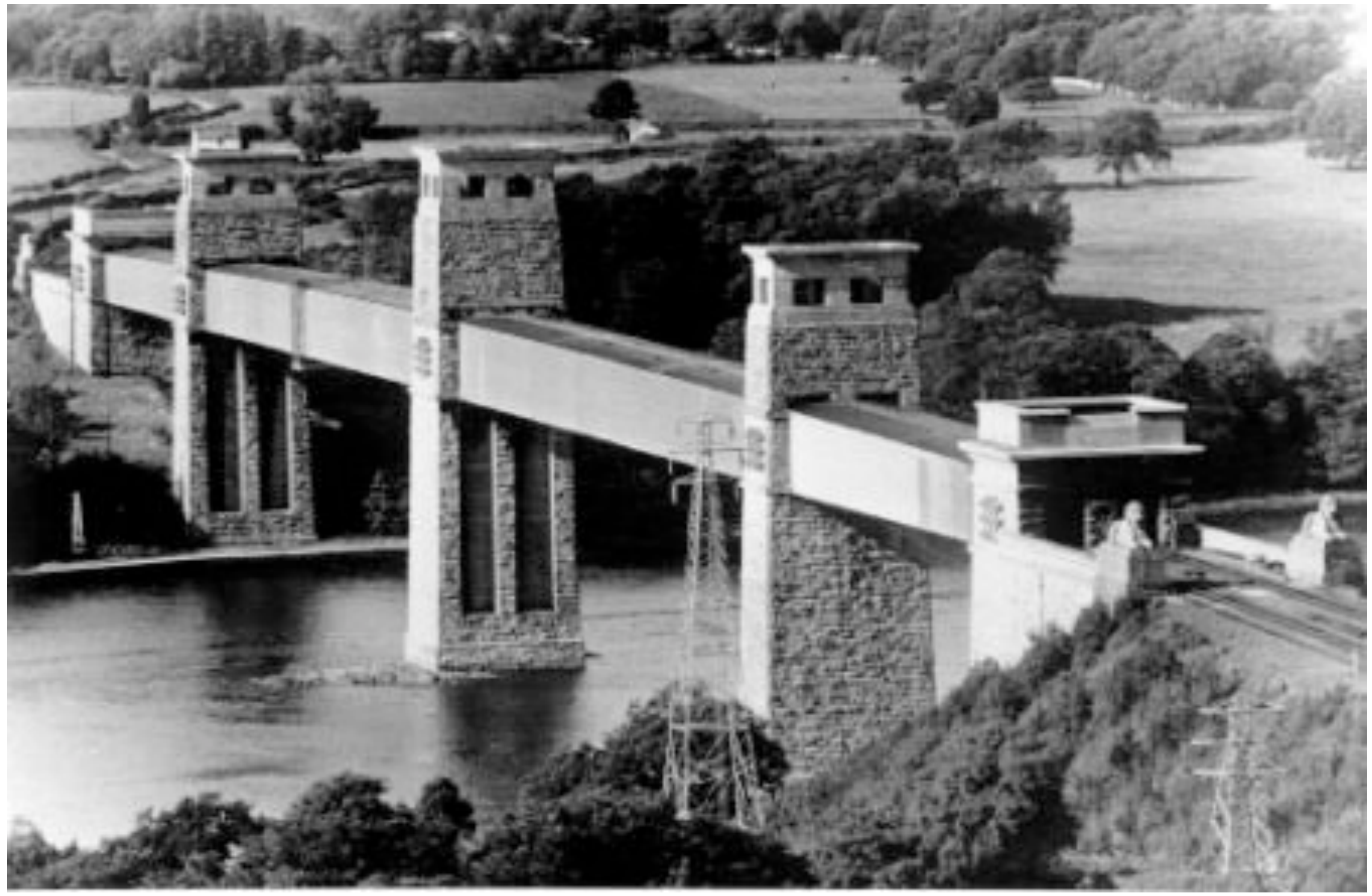
on formal methods

› two benefits: clarity of requirements, mechanical checks

on cost

› key to low cost is upfront investment, non-uniformity

too hard to argue, unsafe to build

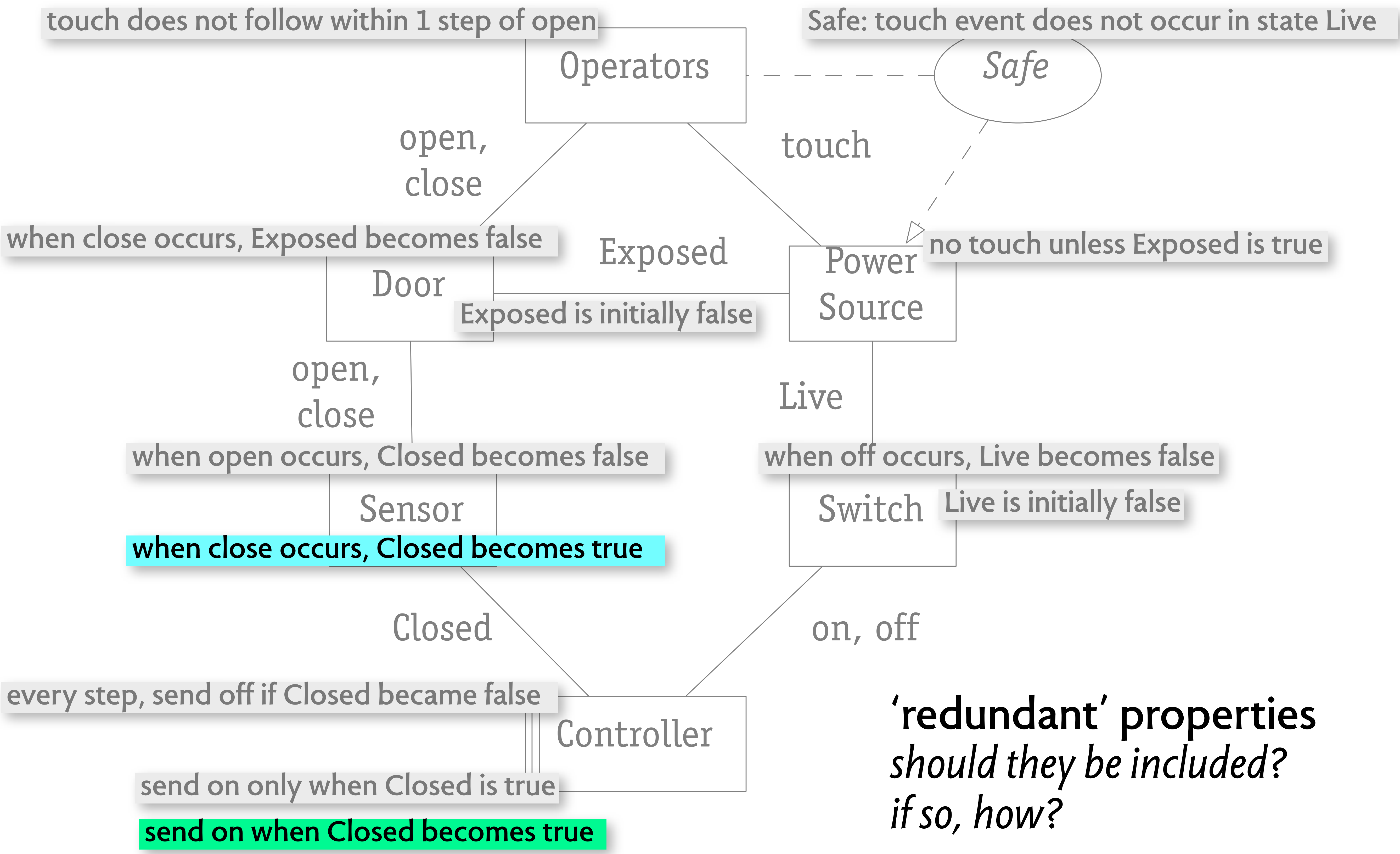


Brittania Bridge (Robert Stephenson, 1850)

The direction and amount of the complicated strains throughout the trussing [would] become **incalculable** as far as all practical purposes are concerned...

Stephenson, explaining why he rejected a suspension design

a research question



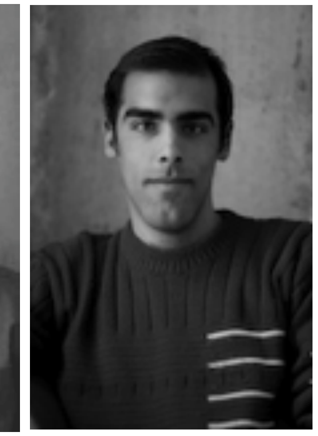
'redundant' properties

should they be included?

if so, how?

acknowledgments





joint work with my students

- Eunsuk Kang, Joe Near, Aleks Millicevic

phenomenology

- Michael Jackson, *Problem Frames* (2001)

dependability cases study

- ‘Sufficient Evidence’ (NAS, 2007)

related work by many

- van Lamsweerde, Kelly, etc (goal structuring)
- Rushby, Knight, Bloomfield (assurance cases)
- ...

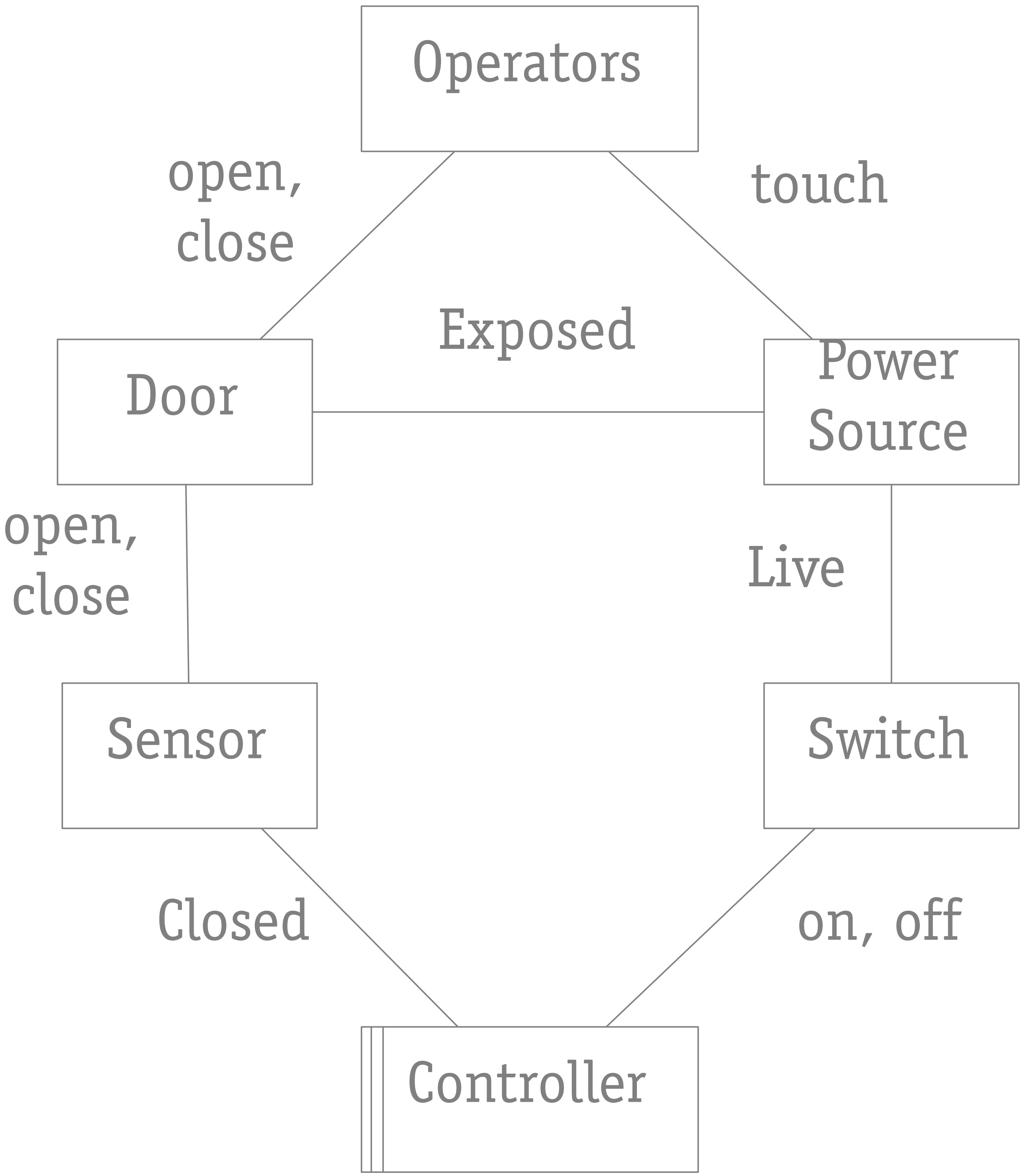
support from NSF, Northrop Grumman, Mass General

a paper about this approach

A Direct Path to Dependable Software, CACM, March 2009
wordle thanks to Jonathan Feinberg, IBM Research, Cambridge

backup slides

designations



events

open: operator opens door fully or partially

close: operator closes door fully

touch: operator touches power

on: controller issues command to switch to turn on

off: controller issues command to switch to turn off

states

Exposed: power source is exposed

Live: power in live state

Closed: sensor is in state that reports door closed

what if analysis finds no flaws?

domain assumptions $\wedge$ machine spec $\Rightarrow$ requirement

informal problems

- wrong domain assumption
- missing phenomena or interactions
- wrong or badly expressed requirement

formal problems

- scope not large enough
- inconsistent axiomatization
- analysis tool is broken
- … *or system is actually safe*

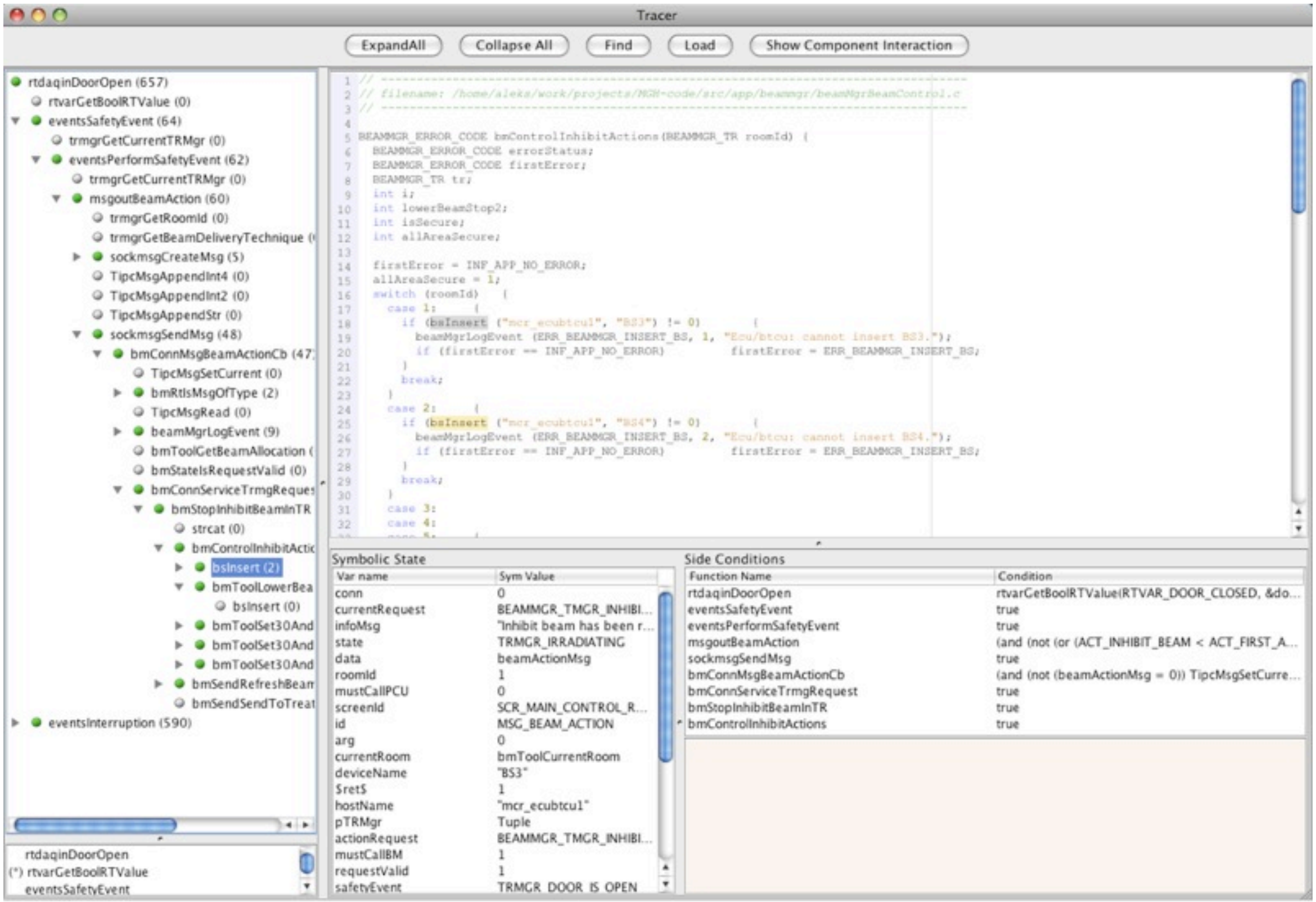
generic modules: domains

```
module domains

abstract sig Domain {}

abstract sig Property {}

abstract sig Requirement extends Property {
    trustedBase: set Domain
    }

sig OK in Domain + Property {}

assert BaseSufficient {
    all r: Requirement | r.trustedBase in OK implies r in OK
    }
```

generic modules: events

```
module events

open util/ordering[Time] as time

sig Time {}

abstract sig Event {
	pre, post: Time
	}

fact Traces {
	all t: Time - last | some e: Event | e.pre = t and e.post = t.next
	all t: Time - last | lone e: Event | e.pre = t
	}
```

examining side conditions



on software risks

"We have become dangerously dependent on large software systems whose behavior is not well understood and which often fail in unpredicted ways."

President's Information Technology Advisory Committee, 1999

"The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in. We're computer professionals. We cause accidents."

Nathaniel Borenstein, Programming as if People Mattered, Princeton University Press, 1991

on accidents

"Accidents are signals sent from
deep within the system
about the vulnerability and
potential for disaster that lie within"

Richard Cook and Michael O'Connor
Thinking About Accidents And Systems (2005)

on design

"There probably isn't a best way to build the system, or even any major part of it; much more important is to avoid choosing a terrible way, and to have a clear division of responsibilities among the parts."

Butler Lampson
Hints for computer system design (1983)