A Systematic Process-Model-Based Approach for Synthesizing Attacks and Evaluating Them

UMASS AMHERST | UCDAVIS UNIVERSITY OF CALIFORNIA

EVT/WOTE 2012
Bellevue, WA
August 7, 2012

Huong Phan
George Avrunin
Lori Clarke
Lee Osterweil
University of Massachusetts Amherst
http://laser.cs.umass.edu

hphan@cs.umass.edu

Matt Bishop
University of California Davis
http://nob.cs.ucdavis.edu/bishop/

Motivation

- Improving election security is important
- Elections are complex processes composed of many steps performed by humans and computer systems
- Past election security work focused largely on:
  - Computer systems
  - Cryptography protocols used in election processes

Our approach

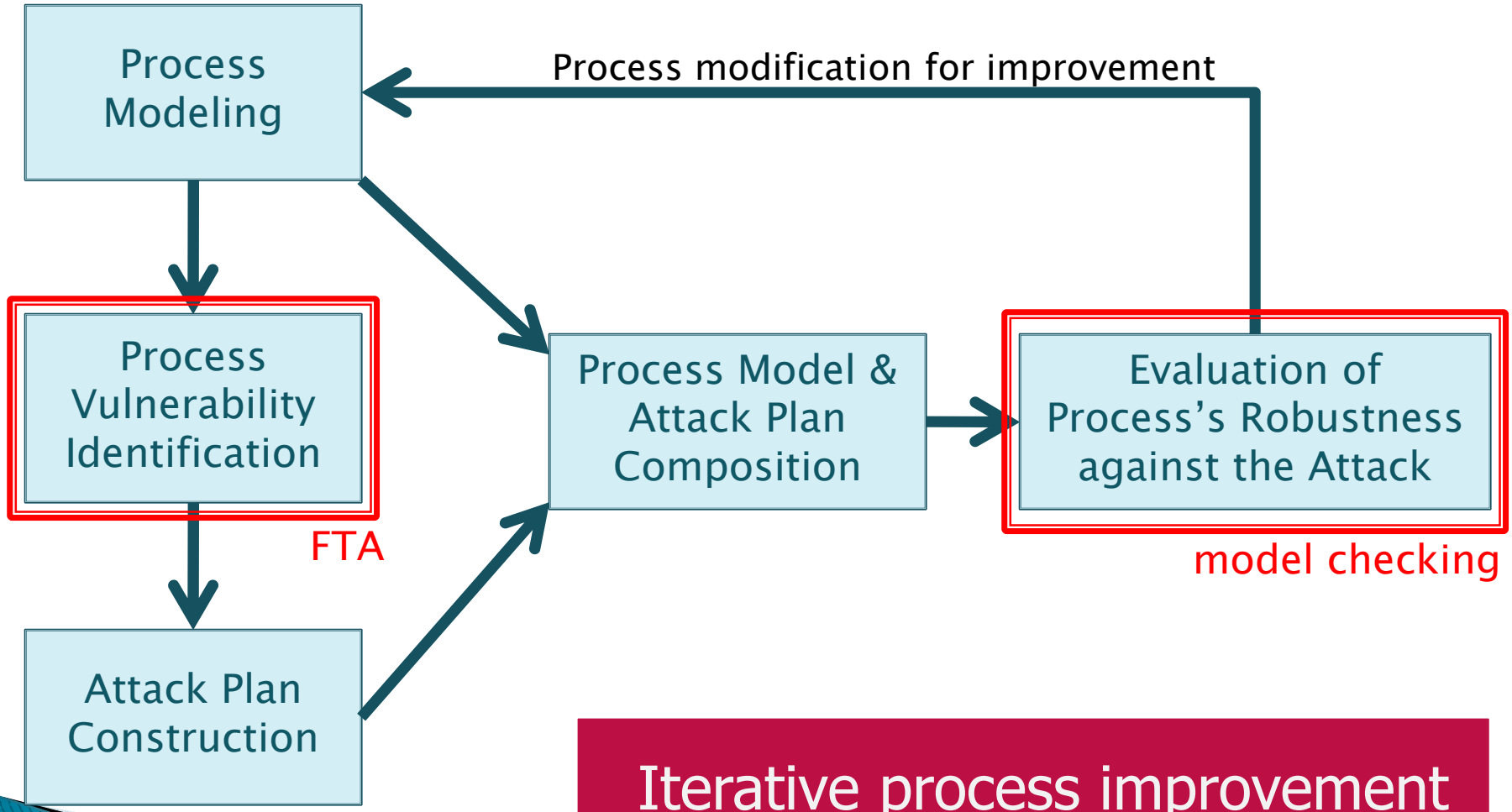
Examine the overall election process by

- Developing a rigorously defined model of the process
  - Derived from and validated by domain experts
- Applying formal analyses
  - To detect vulnerabilities
    - Using Fault Tree Analysis (FTA)
  - To evaluate the process's resistance to attacks based on those vulnerabilities
    - Using model checking

Election officials may use the results of this work to

- Identify and remove vulnerabilities before attacks occur
- Evaluate how well proposed changes will thwart known attacks

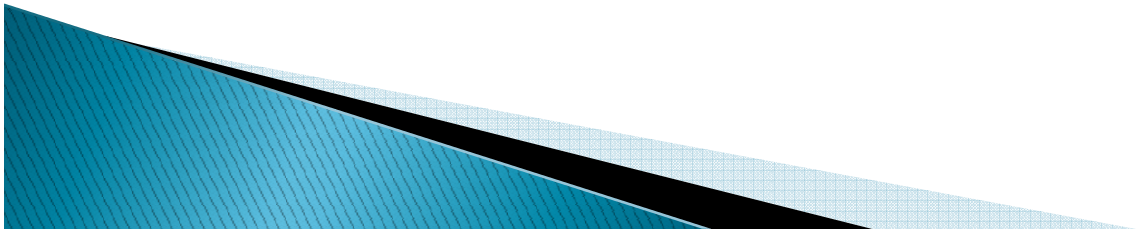
Our approach



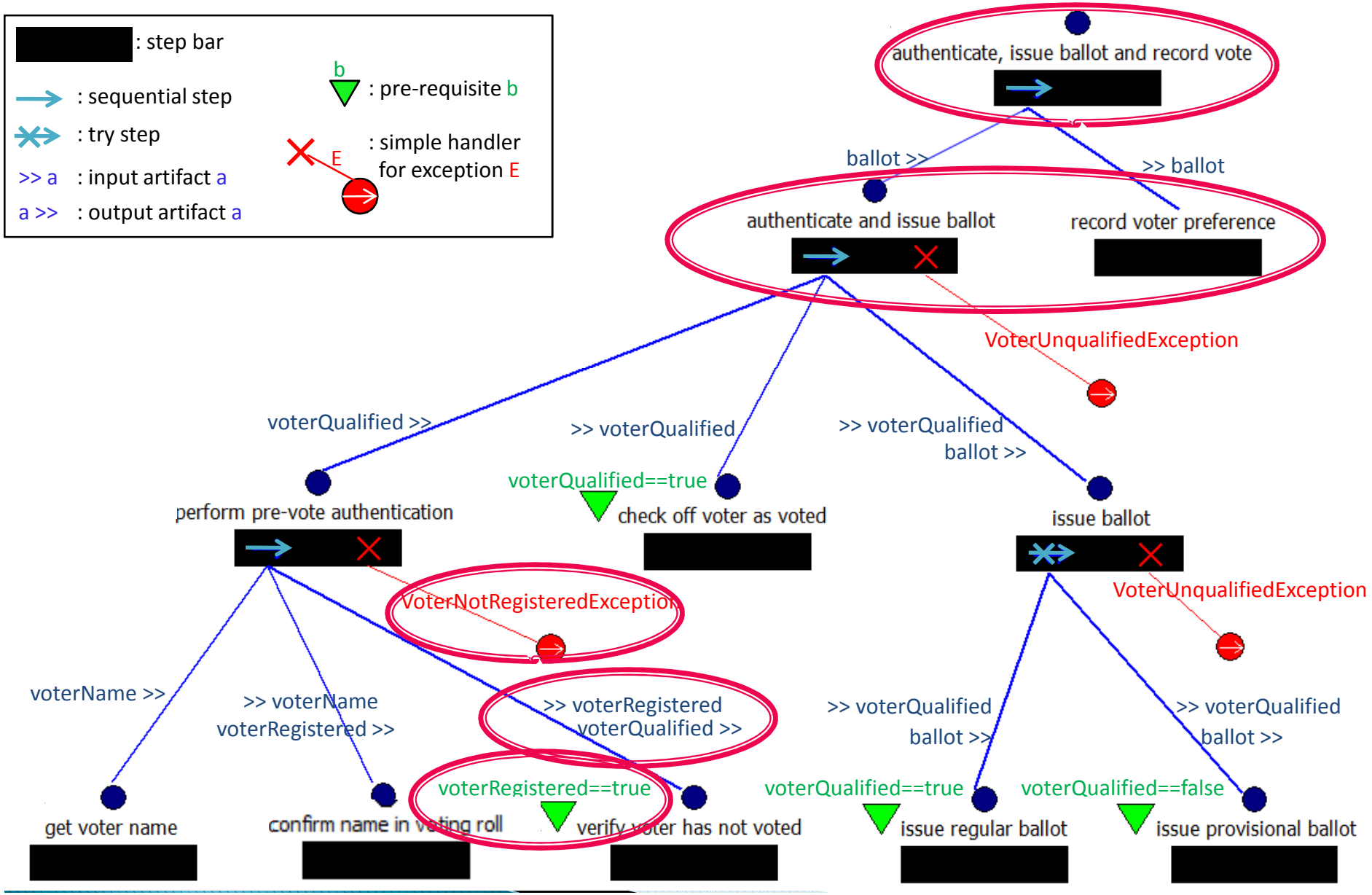
Iterative process improvement

Process modeling with Little-JIL

- Expressive to closely capture real-world processes
- Having well-defined semantics for rigorous analyses
- Accessible to domain experts

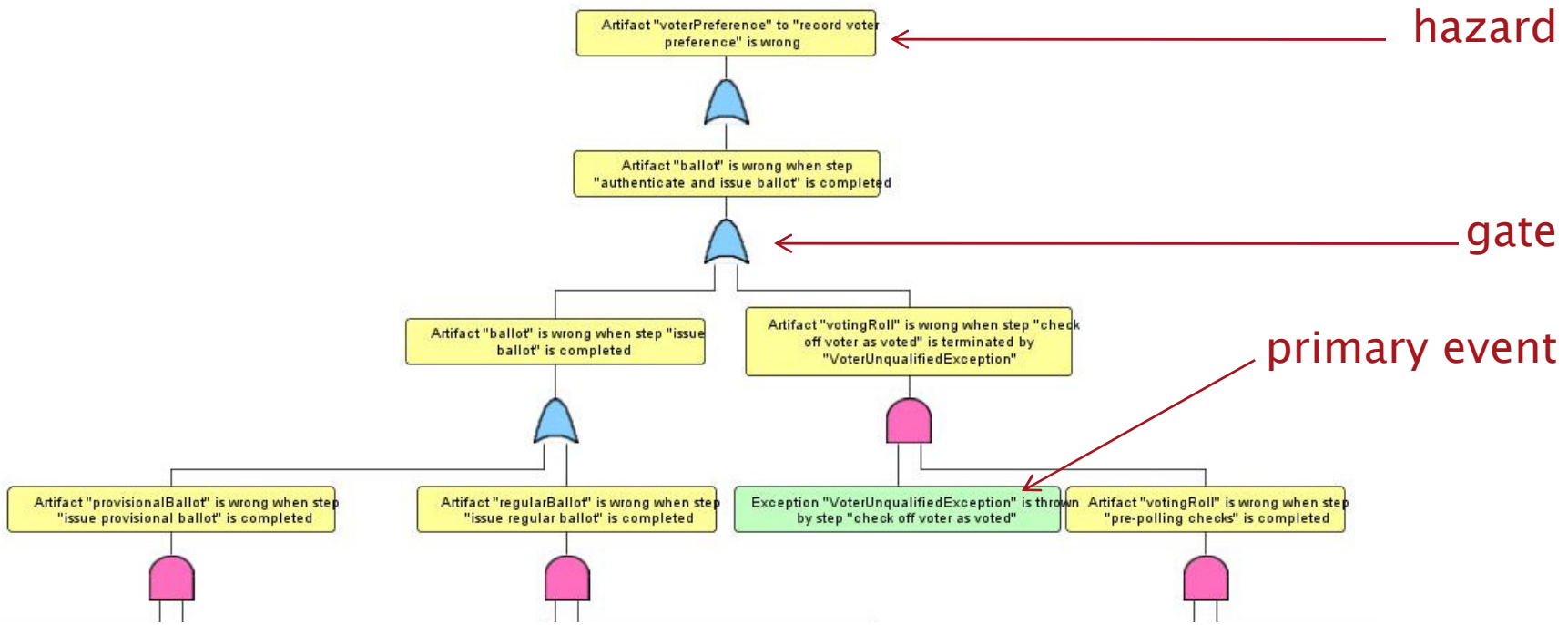


Process modeling with Little-JIL



Fault Tree Analysis (FTA)

- FTA is a deductive, top-down analysis to find out which events in a system could lead to a given ***hazard***
- A ***fault tree*** is a graphical model of the various combinations of events producing the hazard



Fault Tree Analysis (FTA)

- FTA is a deductive, top-down analysis to find out which events in a system could lead to a given ***hazard***
- A ***fault tree*** is a graphical model of the various combinations of events producing the hazard

- A ***minimal cut set*** (***MCS***) is a minimal set of primary events all of whose occurrence ensures that the hazard event occurs
- A MCS indicates a system ***vulnerability*** that an adversary may be able to exploit to create the hazard

Example: can an unqualified voter get to vote with a regular ballot?

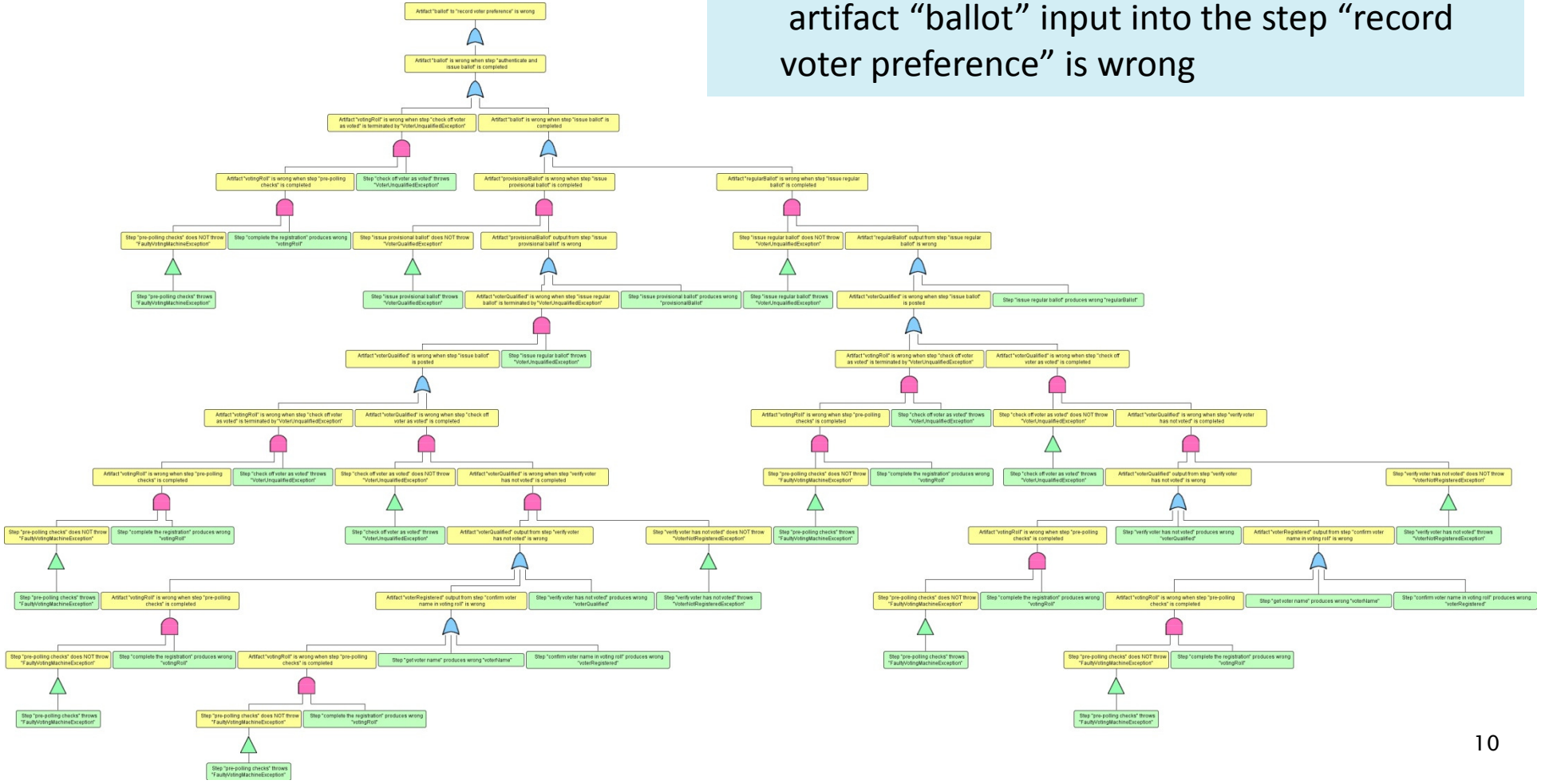
- Applying FTA to the Yolo County election process model

The derived fault tree

Hazard: an unqualified voter gets to vote with a regular ballot

Hazard specification using the FTA tool

artifact “ballot” input into the step “record voter preference” is wrong



The resulting MCSs

- There are 11 MCSs in the fault tree
- Example:

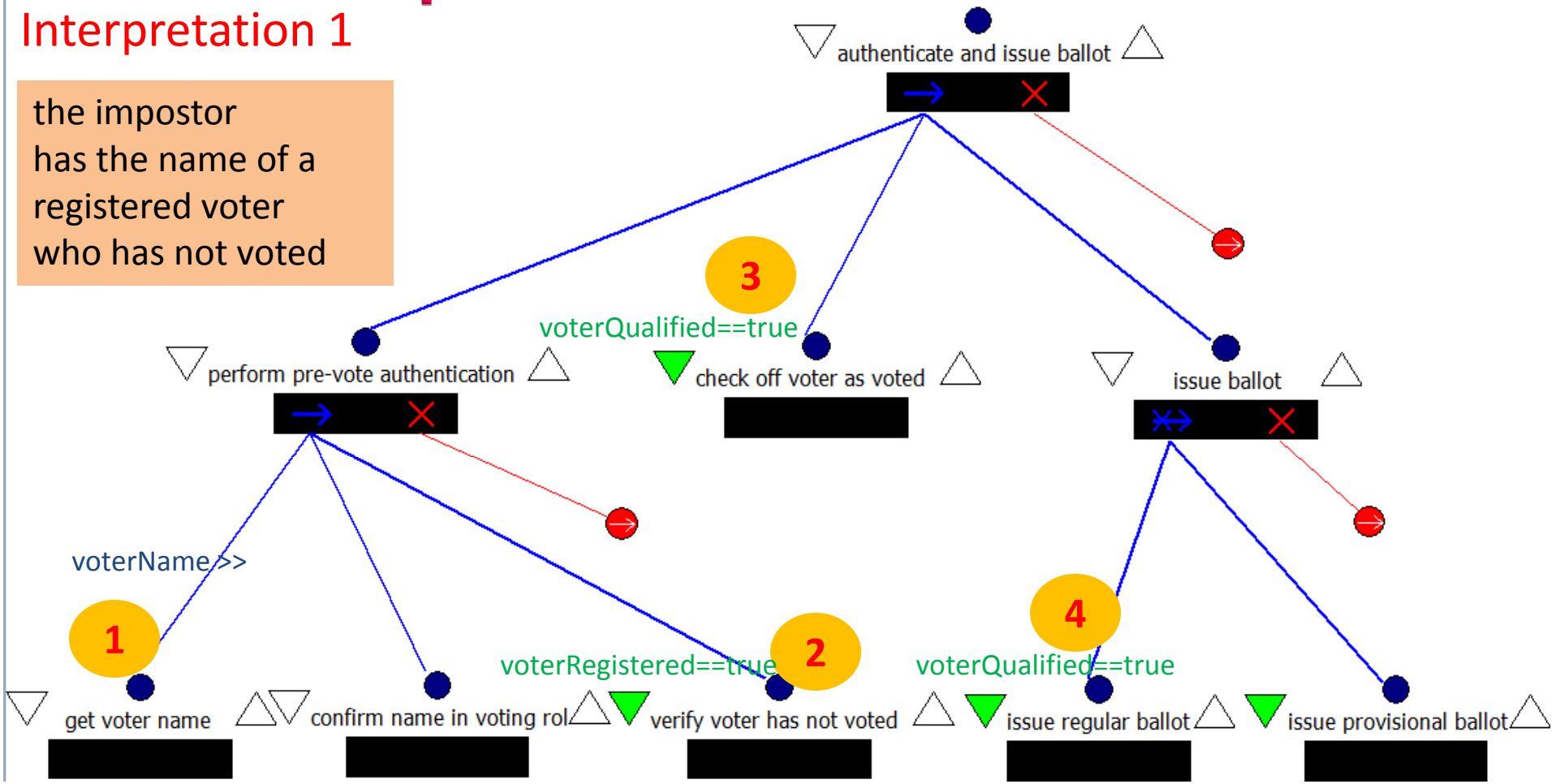
1. Step **get voter name** produces wrong **voterName**
2. Step **verify voter has not voted** does not throw **VoterUnregistered** exception (while checking prerequisite)
3. Step **check off voter as voted** does not throw **VoterUnqualified** exception (while checking prerequisite)
4. Step **issue regular ballot** does not throw **VoterUnqualified** exception (while checking prerequisite)

Interpreting MCSs is challenging

MCS example

Interpretation 1

the impostor
has the name of a
registered voter
who has not voted

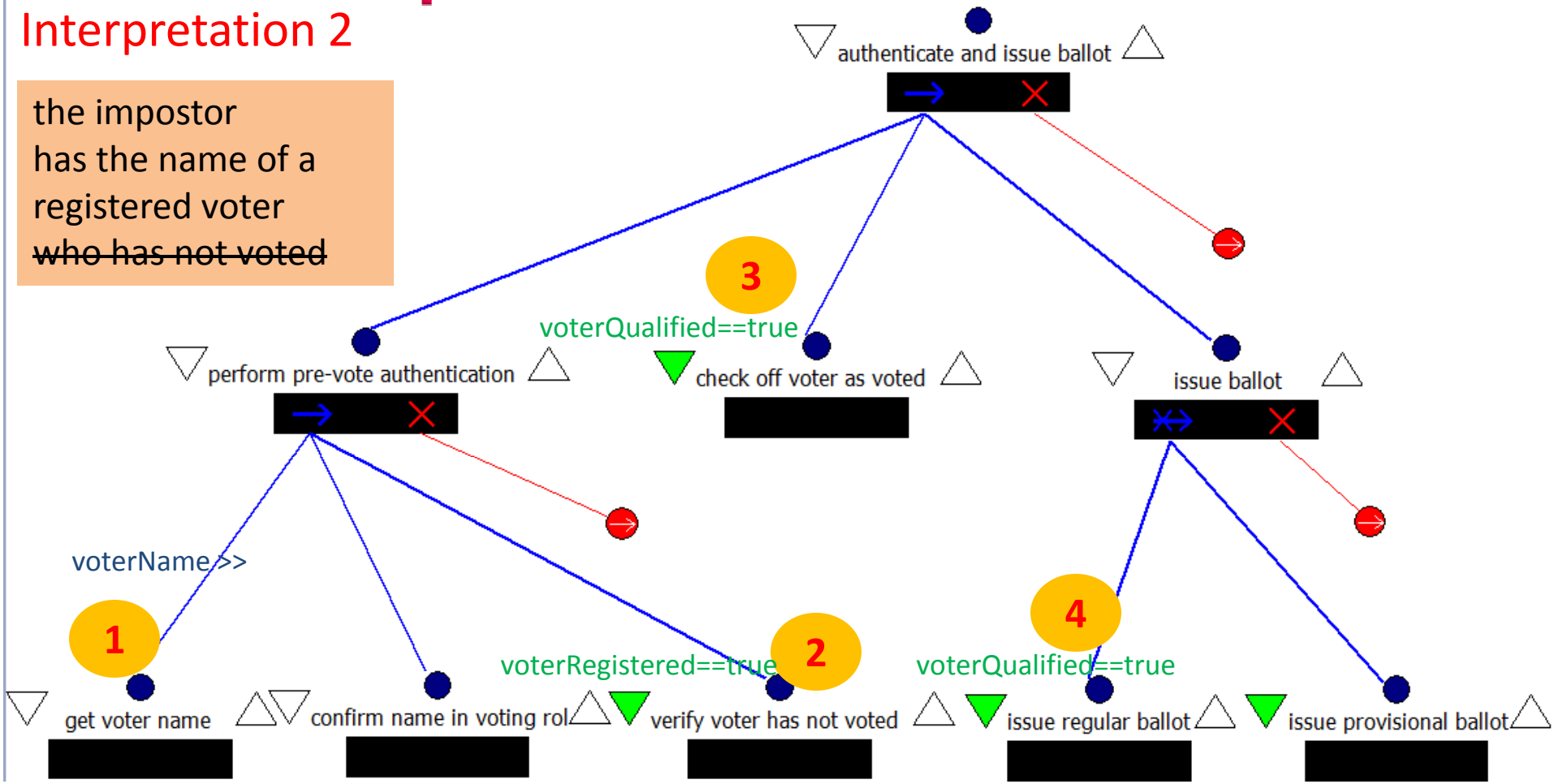


1. Step **get voter name** produces wrong **voterName**
2. Step **verify voter has not voted** does not throw **VoterUnregistered** exception (while checking prerequisite)
3. Step **check off voter as voted** does not throw **VoterUnqualified** exception (while checking prerequisite)
4. Step **issue regular ballot** does not throw **VoterUnqualified** exception (while checking prerequisite)

MCS example

Interpretation 2

the impostor
has the name of a
registered voter
~~who has not voted~~

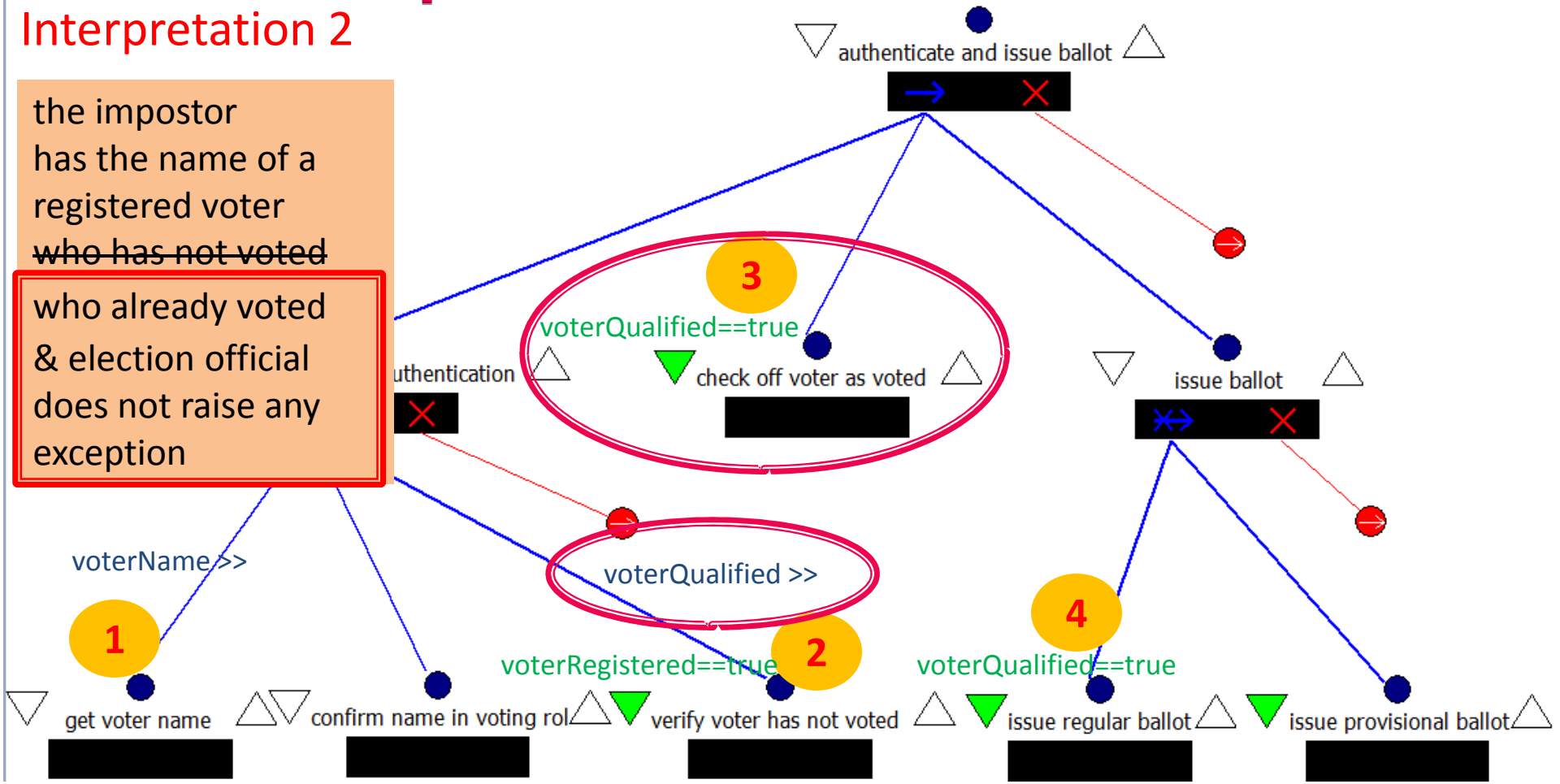


1. Step **get voter name** produces wrong **voterName**
2. Step **verify voter has not voted** does not throw **VoterUnregistered** exception (while checking prerequisite)
3. Step **check off voter as voted** does not throw **VoterUnqualified** exception (while checking prerequisite)
4. Step **issue regular ballot** does not throw **VoterUnqualified** exception (while checking prerequisite)

MCS example

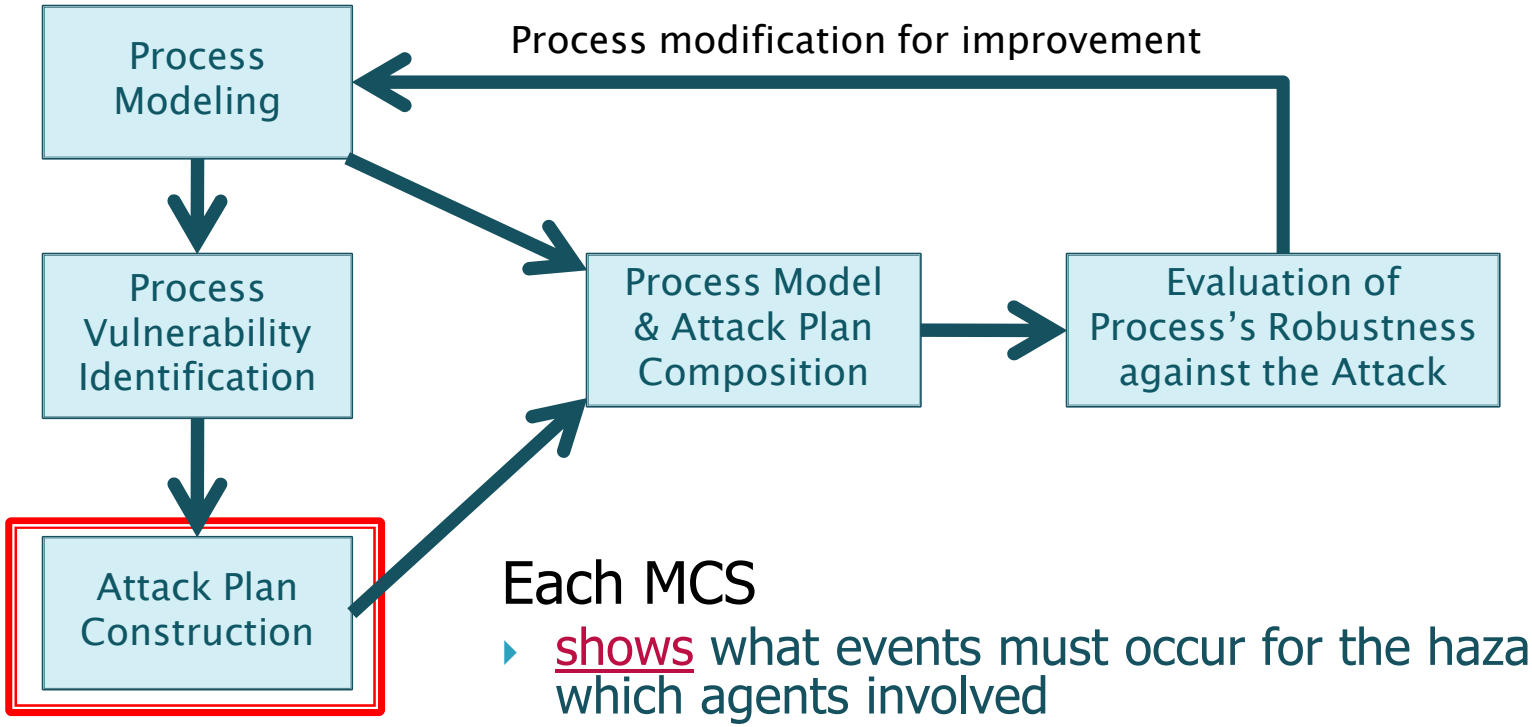
Interpretation 2

the impostor
has the name of a
registered voter
~~who has not voted~~
who already voted
& election official
does not raise any
exception



1. Step **get voter name** produces wrong **voterName**
2. Step **verify voter has not voted** does not throw **VoterUnregistered** exception (while checking prerequisite)
3. Step **check off voter as voted** does not throw **VoterUnqualified** exception (while checking prerequisite)
4. Step **issue regular ballot** does not throw **VoterUnqualified** exception (while checking prerequisite)

From an MCS to attack plans



Each MCS

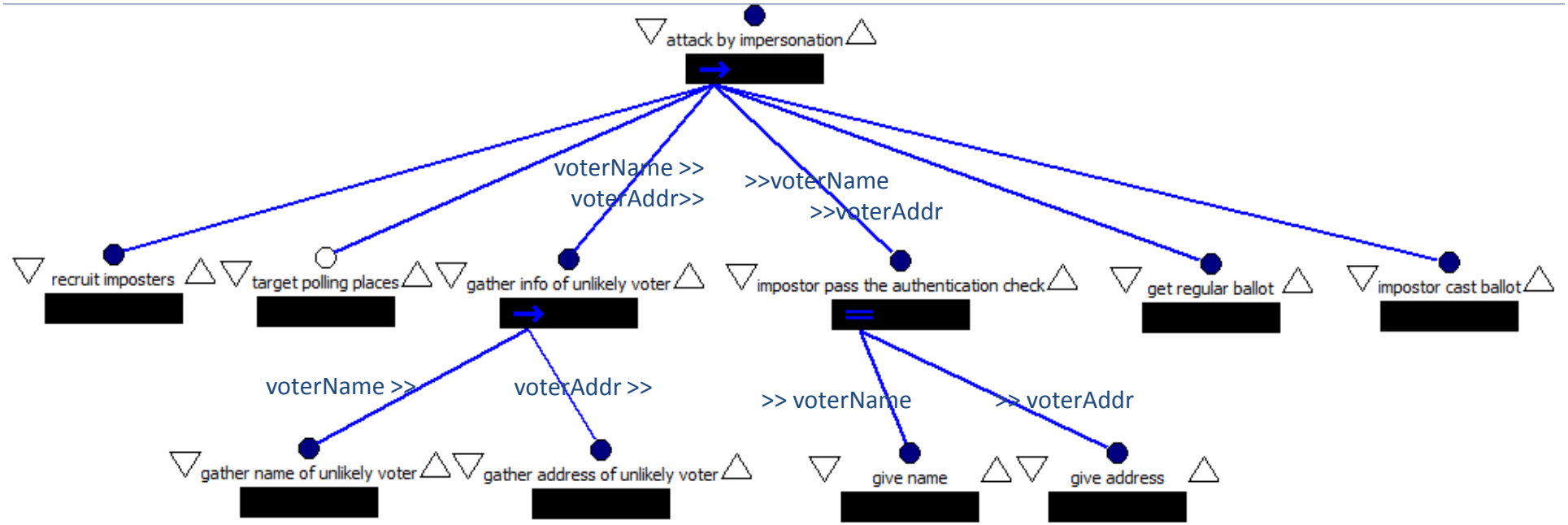
- shows what events must occur for the hazard to occur, which agents involved
- does not show specific actions, practicality, associated costs

Constructing an attack plan

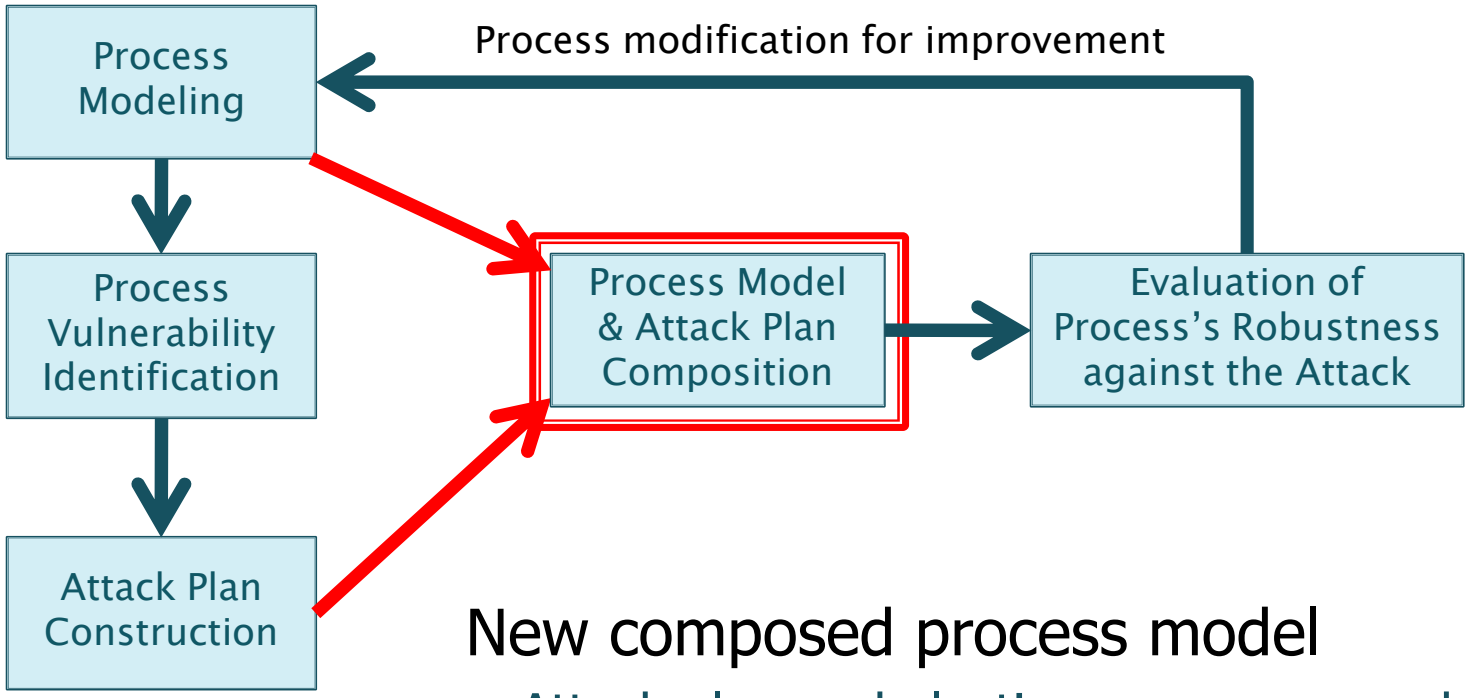
- requires domain expert knowledge
- needs formal definition for further analyses

Modeling the attack plan in Little-JIL

- Matching level of details of the attack with level of details in the model of the attacked process
- Specifying artifact flows and agents



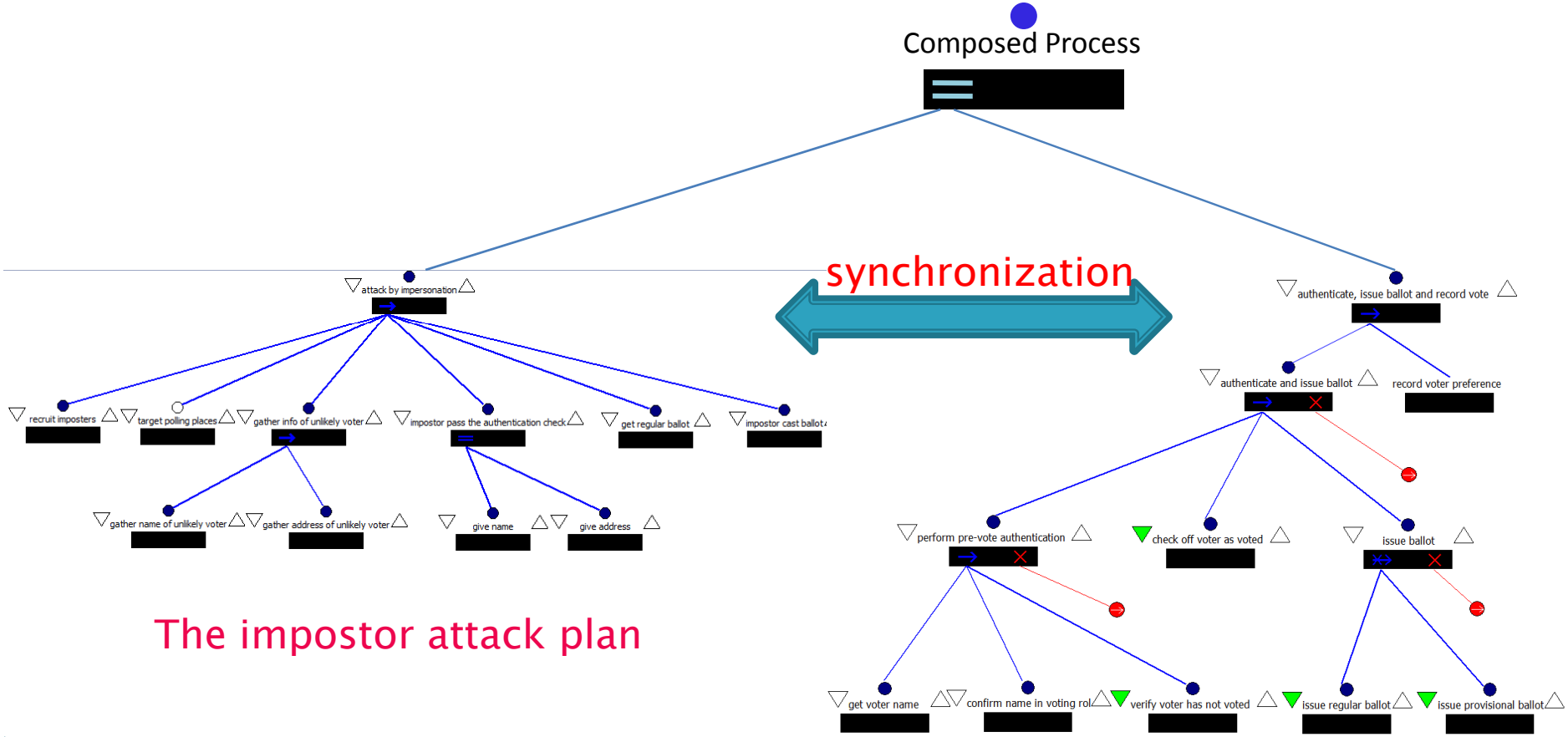
Composing the attack plan and the process model



New composed process model

- Attack plan and election process are sub-processes executing in parallel
- Appropriate synchronization

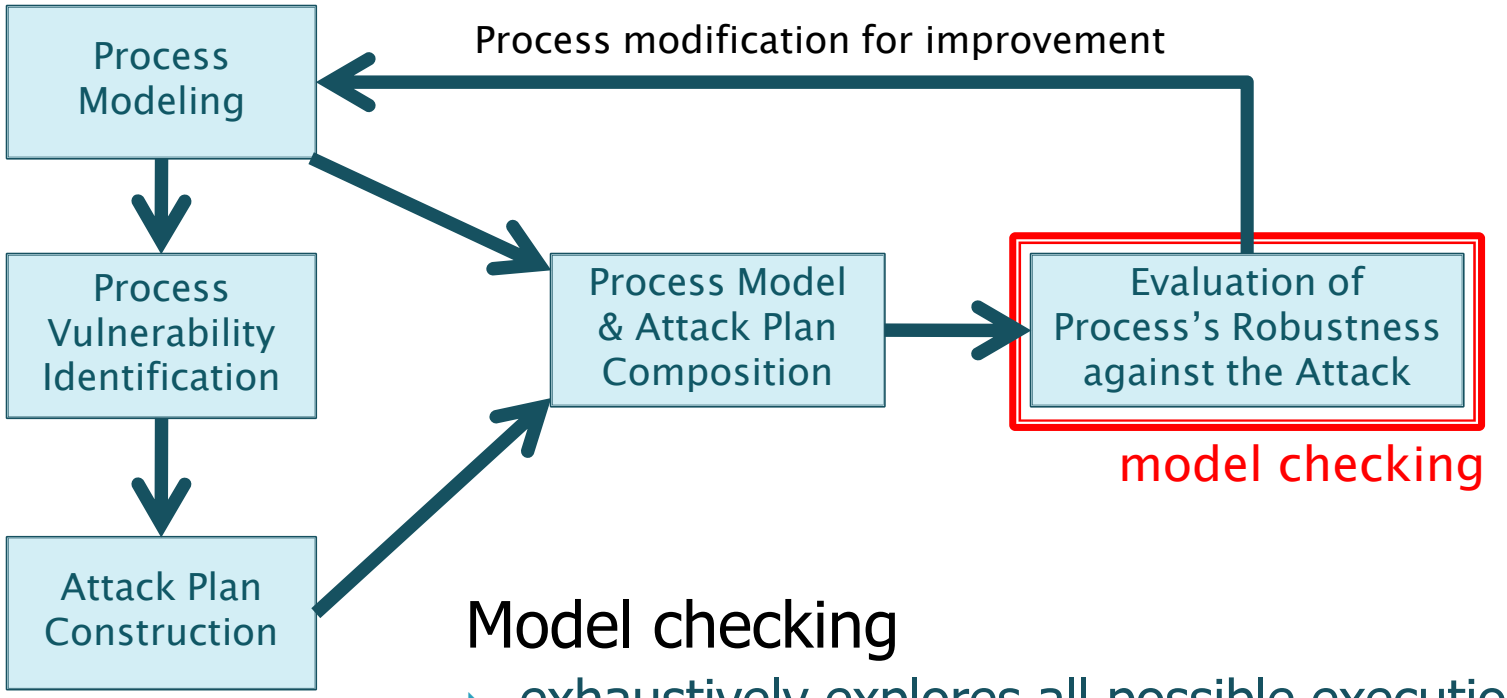
Composing the attack plan and the process model



The impostor attack plan

The election process model

Applying model checking to the composed process model



Model checking

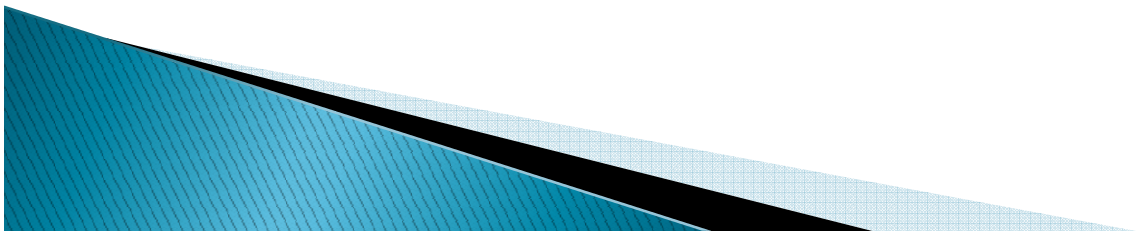
- exhaustively explores all possible execution paths in a finite model of a process,
- determines whether a particular property holds in the model,
- produces a counterexample if the property does not hold.

Applying model checking to the composed process model

- Successful hazard defense is represented by the property: the attack cannot complete successfully
- For our example the property is violated
  - i.e. the attack might complete successfully
  - automatically-generated counterexample shows an attack path (the impostor provides name of registered voter who has not voted)
- Domain expert proposes process modifications
  - one way is to add more authentication
- Re-verify the property with the modified process model
  - the property holds

Related work

- Attack modeling
  - Attack trees, Petri nets, require/provide model, declarative textual specification
  - Usage: documentation, intrusion detection
- Formal reasoning in security
  - Security protocol verification, attack generation
- Process-based security in elections
  - Weldermariam et al



Future work

- More automation
  - Construction of attack plans from MCSs
  - Composition of attack plan and process model
- Agent analyses
  - Consider the number of agents performing steps involved in one MCS
  - Model agent behaviors
    - Explore insider attacks
- MCS and attack plan analyses in terms of probability, cost, impact
- More extensive evaluation

THANK YOU

This work is partially funded by NSF