SAND2011-1375P

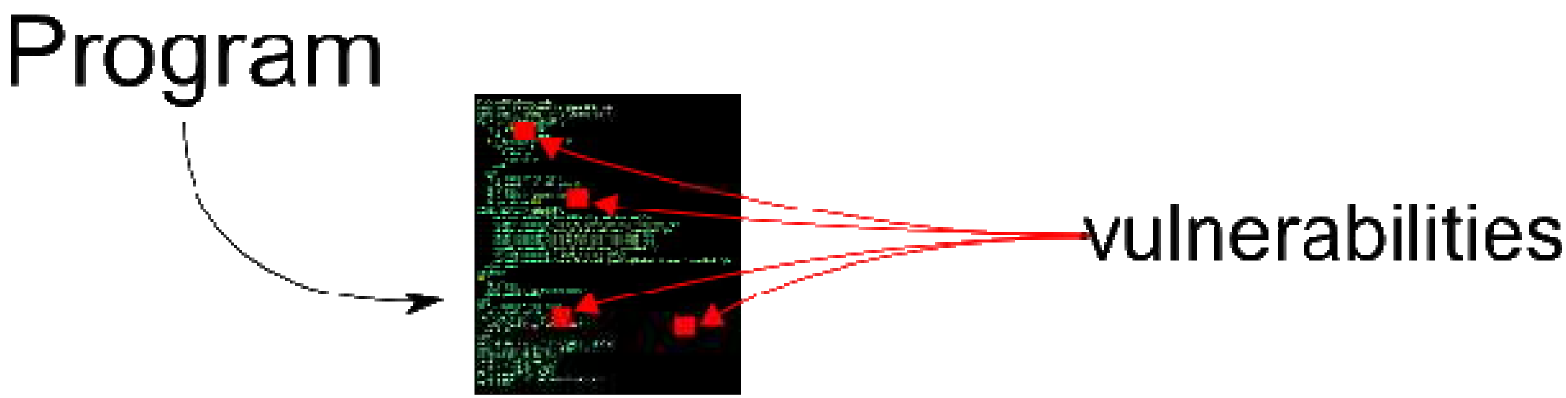
Complex System Modeling and Science-Based Cybersecurity

Computer Sciences and Information Systems
Sandia National Laboratories, Livermore, California

Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Securing an arbitrary code is not just hard; it's impossible

- **Restated: Generic code has vulnerabilities that are unprovable and unknowable**
  - *Not* statistical, even in principle
  - Turing completeness demands that a generic code is undecidable



- **So now what?**

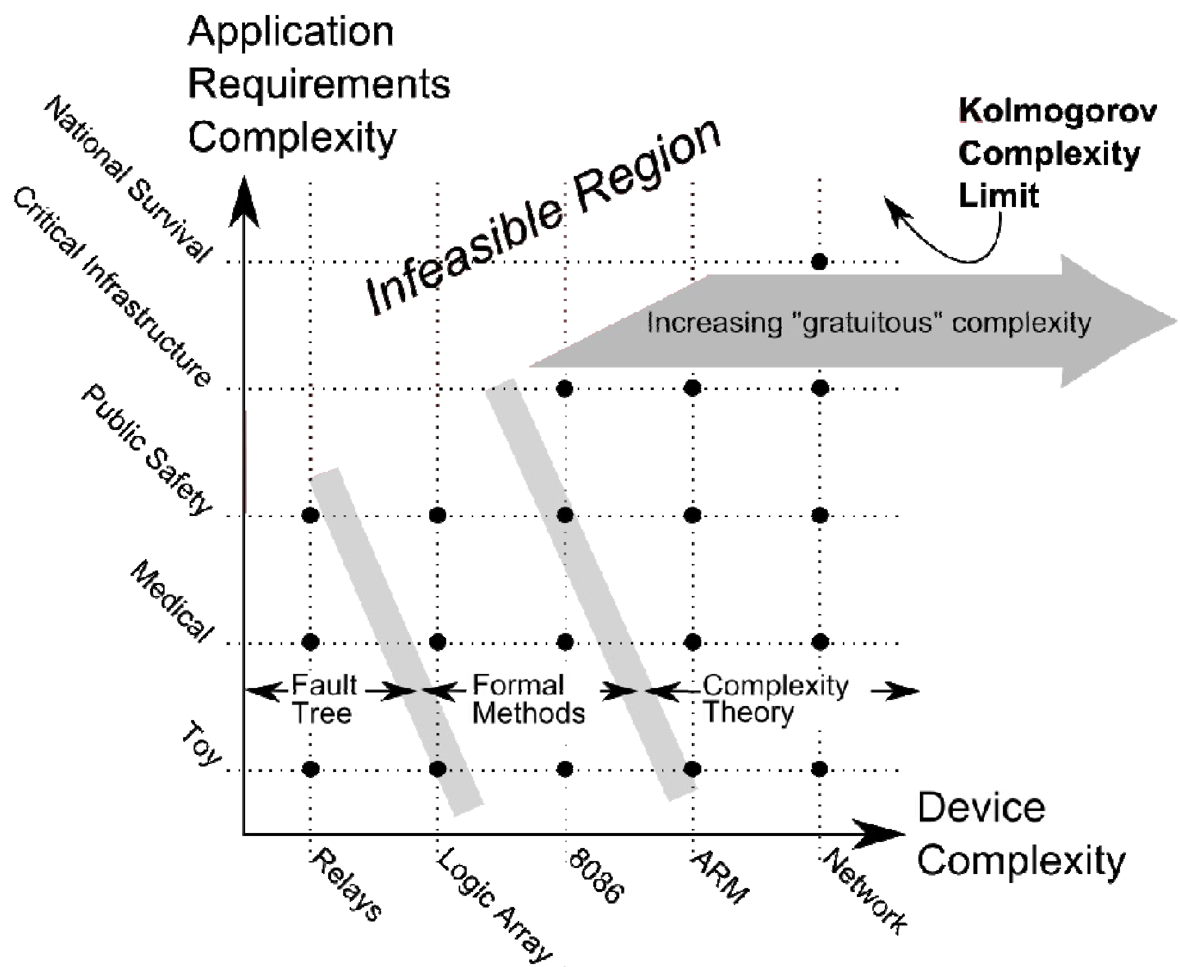
Complexity makes cyber threats *asymmetric*

- **Developer, user, *and attacker* all don't know where the vulnerabilities are (*undecidable*)**
- **Worse, attacker may have planted a vulnerability**
- **Asymmetry: One vulnerability compromises the whole code**
  - Developer has to find all of them (impossible in general)
- **No one can guarantee "this code is clean" or even quantify improvement**

What is complexity?

- **Complex systems are characterized by large numbers of interacting entities where even a few entities can strongly affect system behavior**
- **Complex systems are irreducible; their behavior is emergent and not evident *a priori*, but is accessible via observation and simulation**
- **Examples are ubiquitous**
  - Living things and ecosystems
  - Human societies, economies, and institutions
  - Highly engineered artifacts – e.g., airplanes, NWs
  - Large-scale infrastructure – e.g., power grids
  - Computer software, hardware, and networks

Complexity space illustrates tradeoffs in device engineering and analysis



Sandia National Laboratories

How formal methods work

- **Feed a formal methods tool with a mathematical model of the system (formal specification) and the claimed properties of the system**
  - Automated theorem proving
    - **Uses logical truths and inference to prove the properties of the system**
  - Model checking
    - **Partial order reduction, symbolic manipulation, etc. to reduce the state space**
    - **Exhaustive checking of the reduced states**
- **If a property can be proved false, a counter-example is also provided**

Formal methods are a bridge to complexity, filling an important gap

- **Formal methods use computer analysis to verify digital systems rigorously and exhaustively**
  - Applicable to less complex systems that are still beyond the reach of manual analysis
  - Widely used in high-consequence industrial applications such as aviation and medical devices
- **Verification of components does not generally translate to verification of whole system**
- **Irreducible complexity enters when exploring entire state space is infeasible**
  - Reliability and security assertions become probabilistic
- **Both formal verification and complexity science are vital for gaining confidence in digital systems**

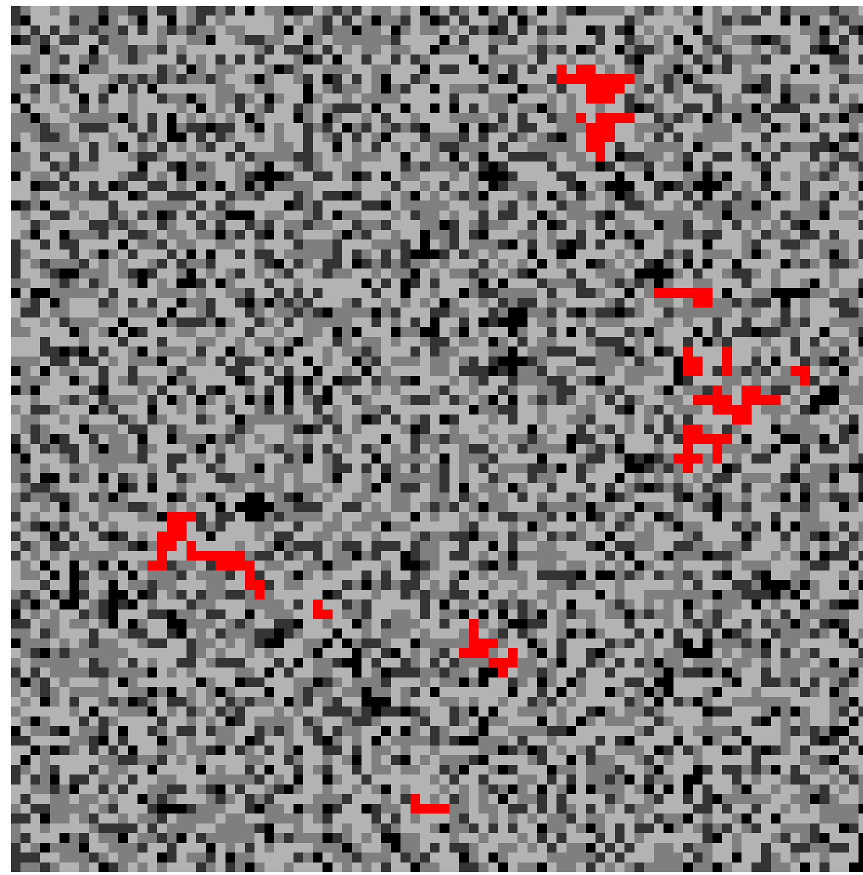
Complexity science offers a new perspective on modeling and design

- **Most real-world systems are too intricate to analyze directly; they are irreducible**
- **Reductionism requires "bottom-up" understanding**
  - Use expert knowledge to model component entities
  - Validate system model vs. observations
  - Make each component entity as reliable as possible
  - Formal methods are the pinnacle of this approach
- **Complexity science provides "top-down" insight relating system structure to emergent behavior**
  - New modeling paradigm: Identify entities by abstraction from idealized models with known emergent behavior
  - New design paradigm: Build real systems based on models with desired emergent behavior

Sandia National Laboratories

Self-organized criticality is a simple example of emergent behavior

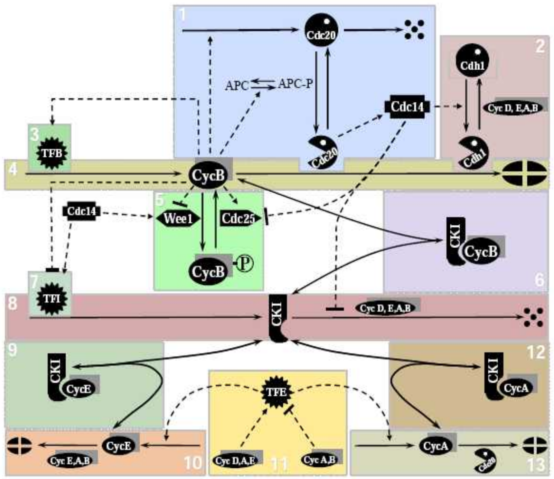
- **“Sandbot”: cyber model of coordinated malware**
- **SOC is *spontaneous* development of multi-scale phenomena with power-law distributions**
  - Similar to thermodynamic criticality but without tuning
- **Illustrated by sandpile model: physics-like CA**
  - Sand is sprinkled randomly
  - Avalanches occur at all scales



Sandia National Laboratories

Complexity is a fact of "life"

- **Biological phenomena are a prototype and inspiration for many complex domains**
  - Life involves a large chemical regulatory network



Eukaryotic cell-cycle regulation

  - "Game of Life" model is based on population dynamics
  - Bio concepts pervade computing (viruses, mutations)
- **Biology typifies complex couplings of manmade systems – economy, energy, cybersecurity**

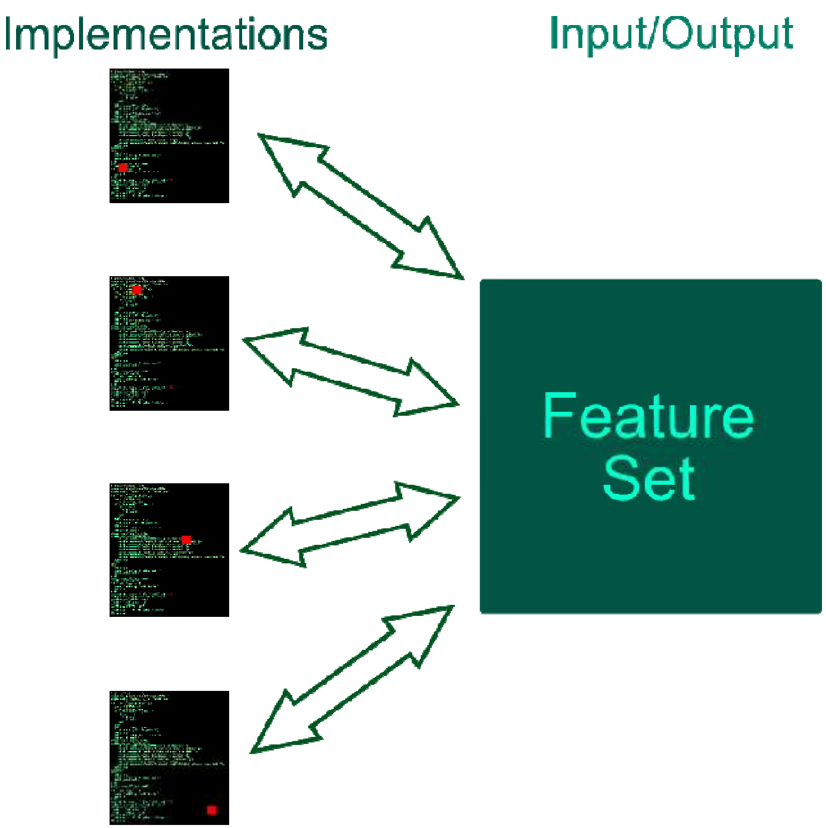
Sandia National Laboratories

Robustness is key to understanding real-world systems with "organic" behavior

- **Highly optimized tolerance (HOT): Systems *designed* or *selected* to perform well despite perturbations**
- **Robustness is necessary for biological evolution and for effective engineering**
- **HOT systems exhibit power-law distributions like SOC but have organic structure (not self-similar)**
- **Adapted robustness to one set of perturbations induces extra fragility to different perturbations**
- **Indeed, rare but catastrophic failures are seen in highly engineered/evolved systems**
  - Electrical blackouts, cyber shutdown of Estonia, financial panics, hacker penetration of bank database, etc.

Sandia National Laboratories

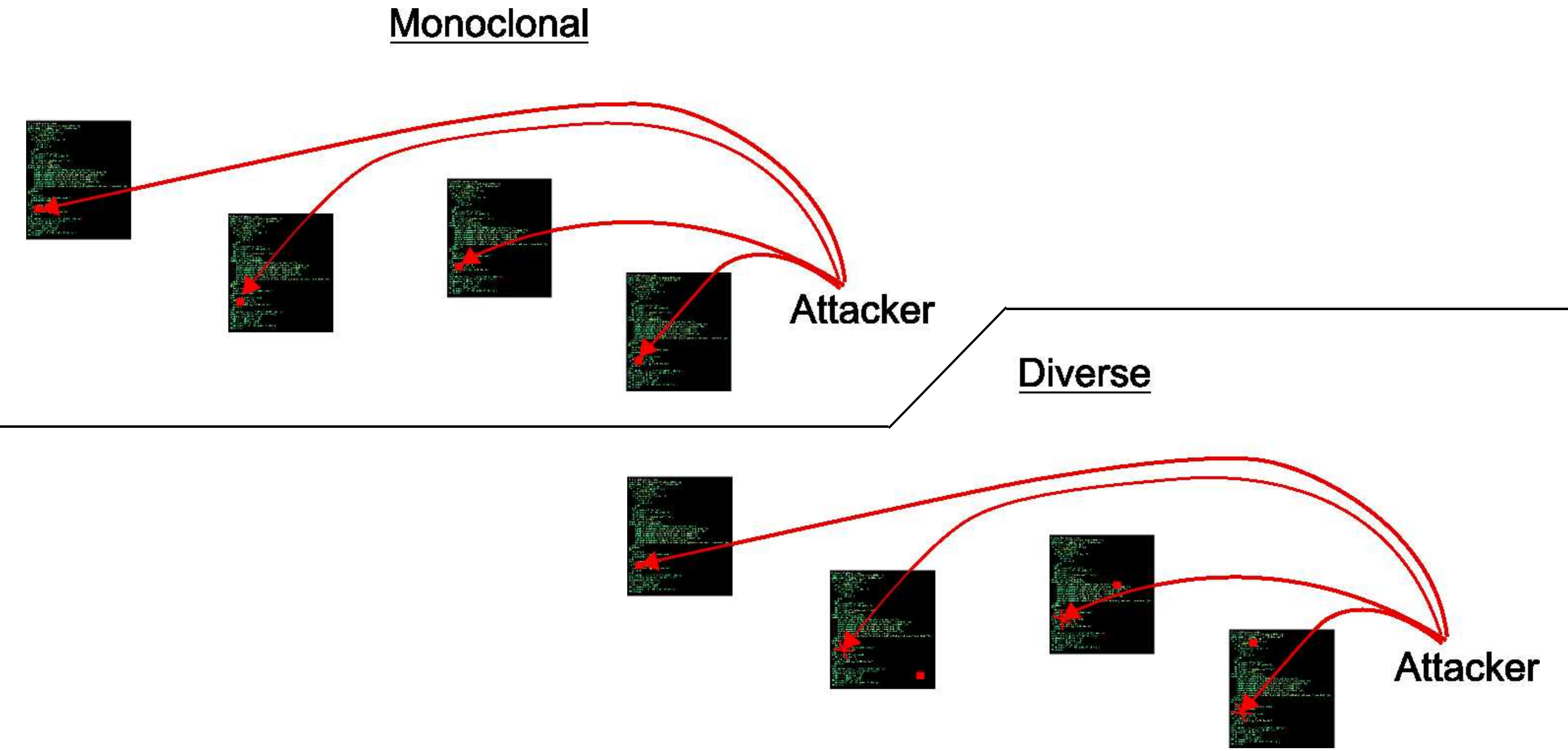
Observation #1: A program's feature set has many implementations



- **Feature set is defined by a test suite**
- **Test suite verifies that an implementation conforms to desired functionality**
- **Test suite is a sample; cannot realistically cover all possible input/outputs**
- **Vulnerabilities arise from untested input/outputs**
- **Any feature set has infinitely many implementations**
  - Finite large number if size is bounded

Observation #2: Ensemble of instances permits the formulation of statistics

- **Assume: Multiple implementations randomize security holes**
- **Ensemble of multiple-version, "randomized" undecidable codes allows formation of security *improvement* statistics**



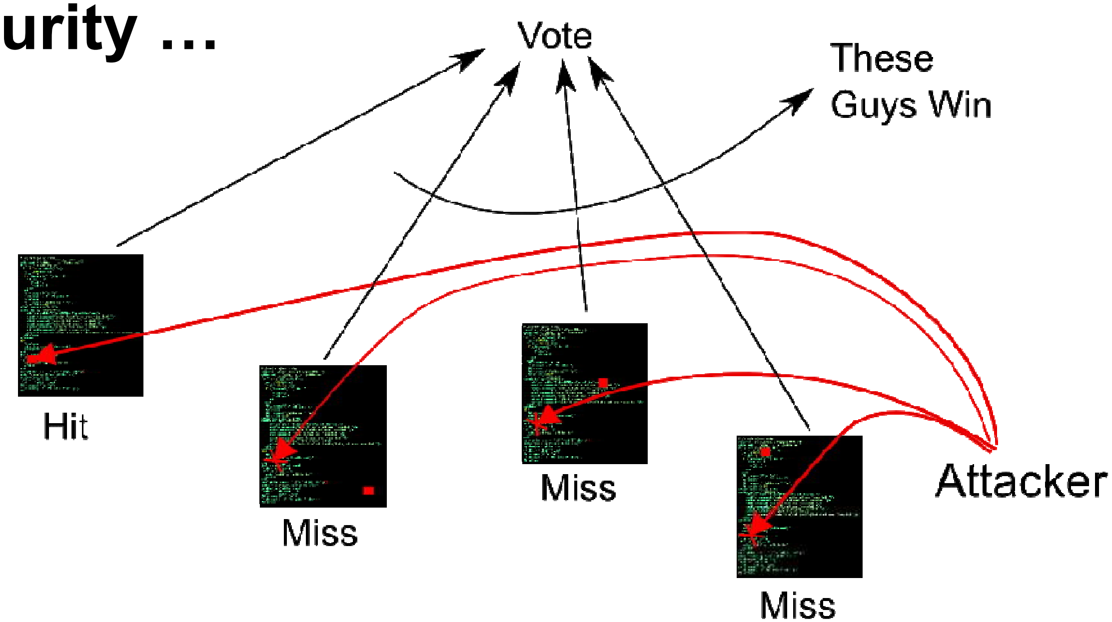
High-reliability systems can be constructed from "*N*-version software"

- **Space Shuttle: 4 computers, identical software, different hardware, same design**
  - Focus is on hardware faults
- **Similarly, software redundancy used mostly for control systems up to now**
  - *N*-version software: Multiple versions implemented to the same feature set by different developers
- **Models of *N*-version software view the control system as a stochastic process that walks the code graph of the software**
  - Control system takes the place of a "fuzzer"

Sandia National Laboratories

Similarly, *N*-version software can quantifiably improve cybersecurity

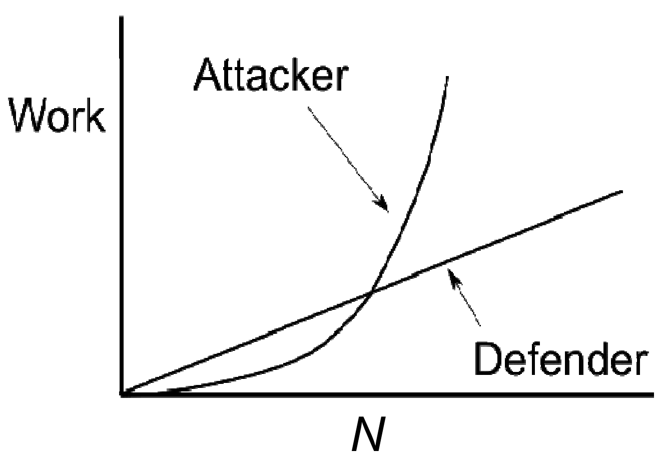
- **Clear generalization of *N*-version reliability to cybersecurity …**



- **… but there are important differences requiring enabling technology**
  - Compromised versions must be removed and replaced
  - Hand-made new versions are time-consuming and expensive
    - **May repeat previous mistakes**

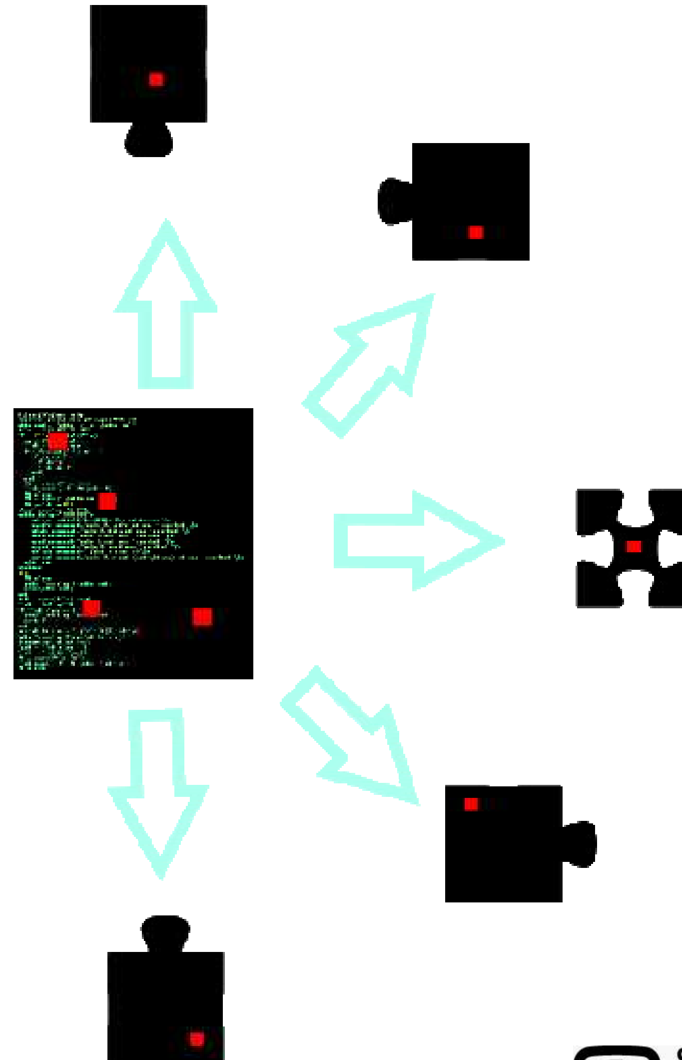
Simple statistics arise from an ensemble of undecidable programs

- **On a specific feature set *F* there is a probability P_v that a particular sample from the set of implementations of *F* will be susceptible to vulnerability *v*. For a voting ensemble of size *N*:**
  - The probability of success for the attacker is $(P_v)^{N/2}$
  - The attacker "work" is the expected number of tries: $(P_v)^{-N/2}$
  - The work for defender is the cost of producing *N* implementations: ~ *N*



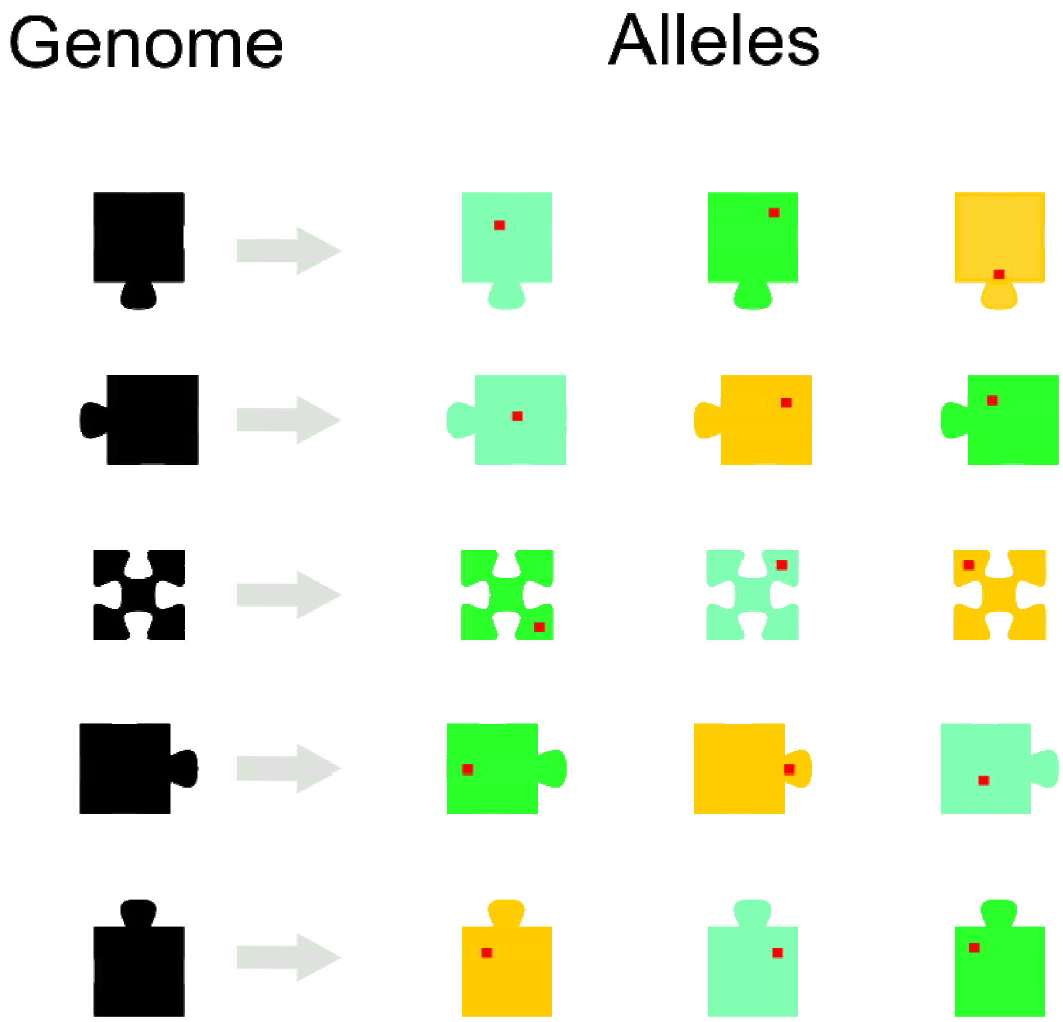
A simple example: Diverse software can be constructed from components

- **Component-based codes automatically conform to a feature set if the constituent components conform to their individual feature sets (semantic interfaces)**
  - Multiple implementations of the code amount to multiple versions of components
  - Components can be mixed and matched to form a combinatorial number of code implementations



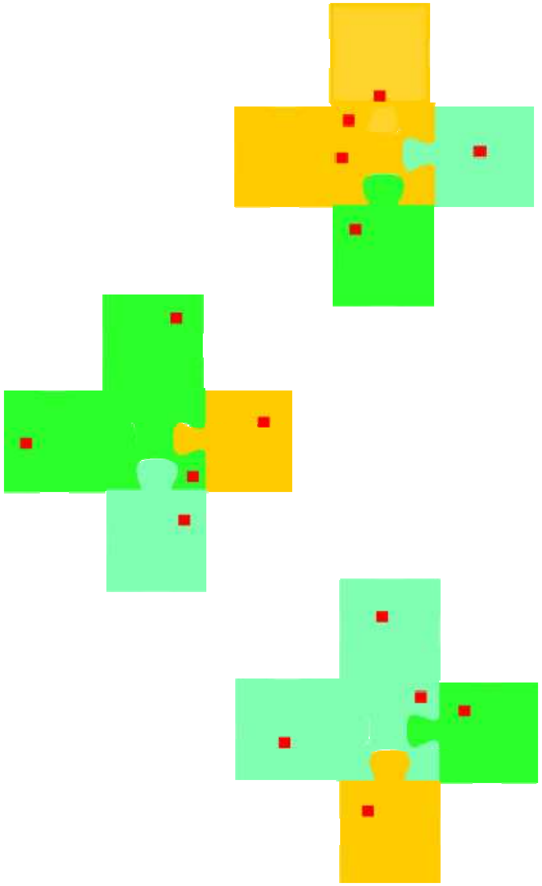
Sandia National Laboratories

Living systems adapt to cope with unknowable attacks



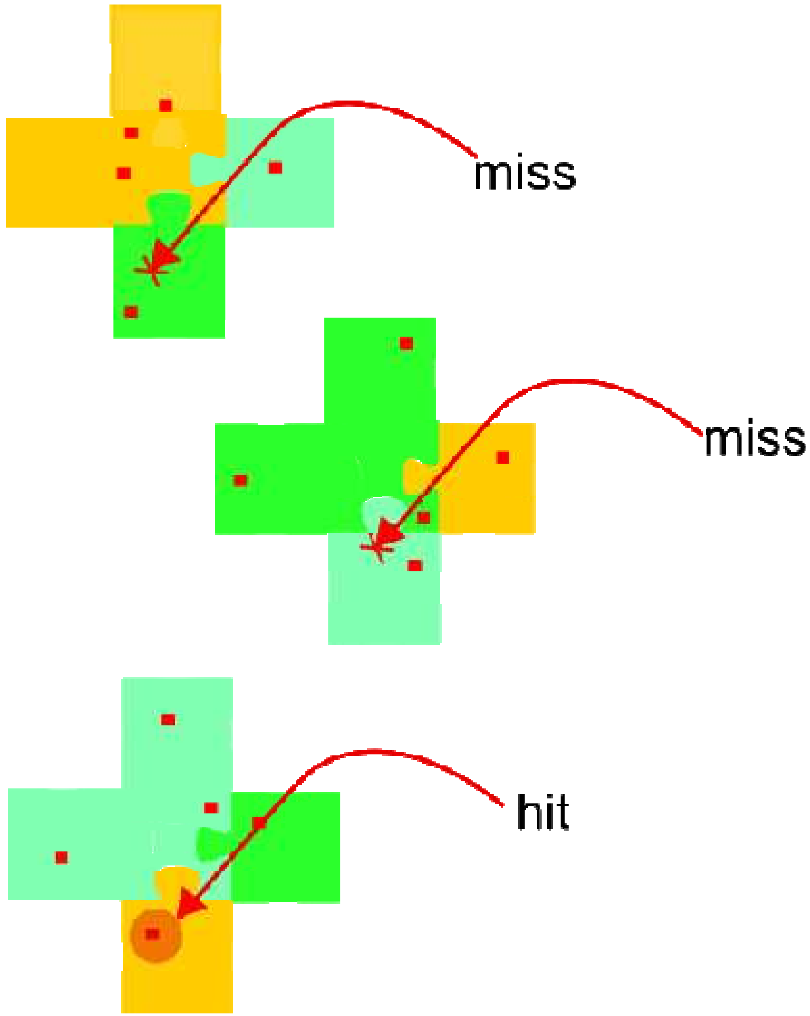
- A component type is similar to a gene; component implementations are similar to alleles of a gene

Reassemble alleles into individuals



- Different alleles can be assembled into new individuals that have “randomized” security holes
- New individuals are differently vulnerable and potentially adaptive
- Excess functionality and planted vulnerabilities can be “annealed” away

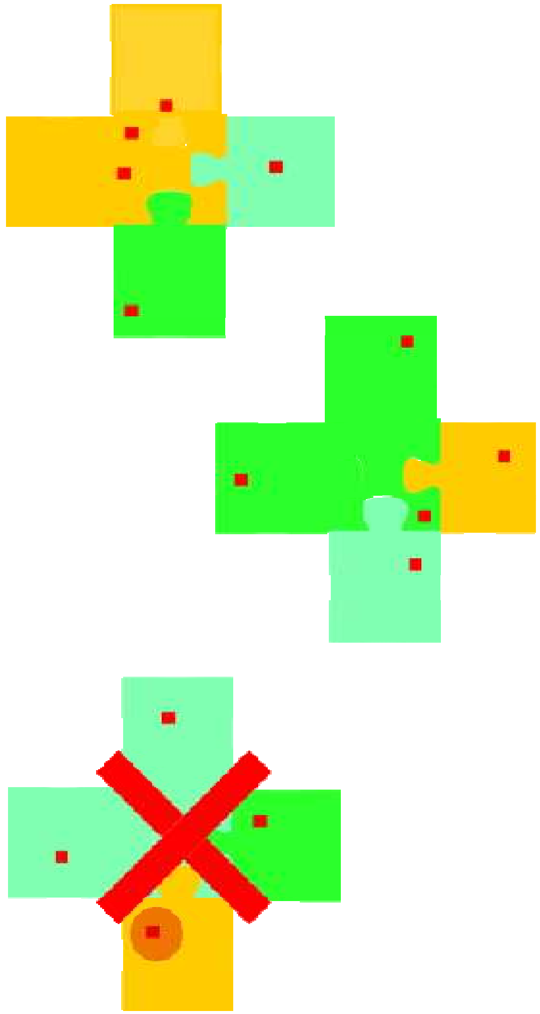
Compare responses from individuals



- **Now different individuals will produce the same feature set but react differently to attacks**

Sandia National Laboratories

Evolve new and more robust individuals

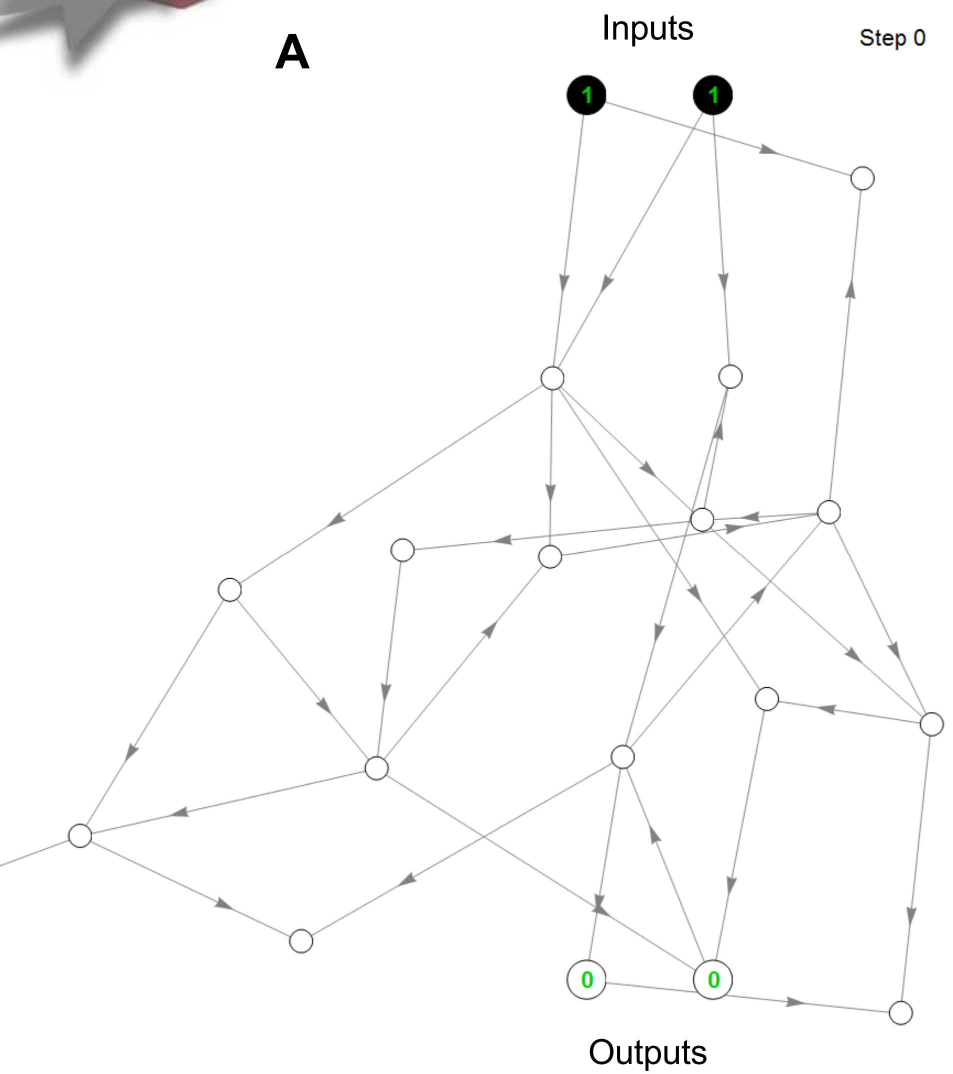


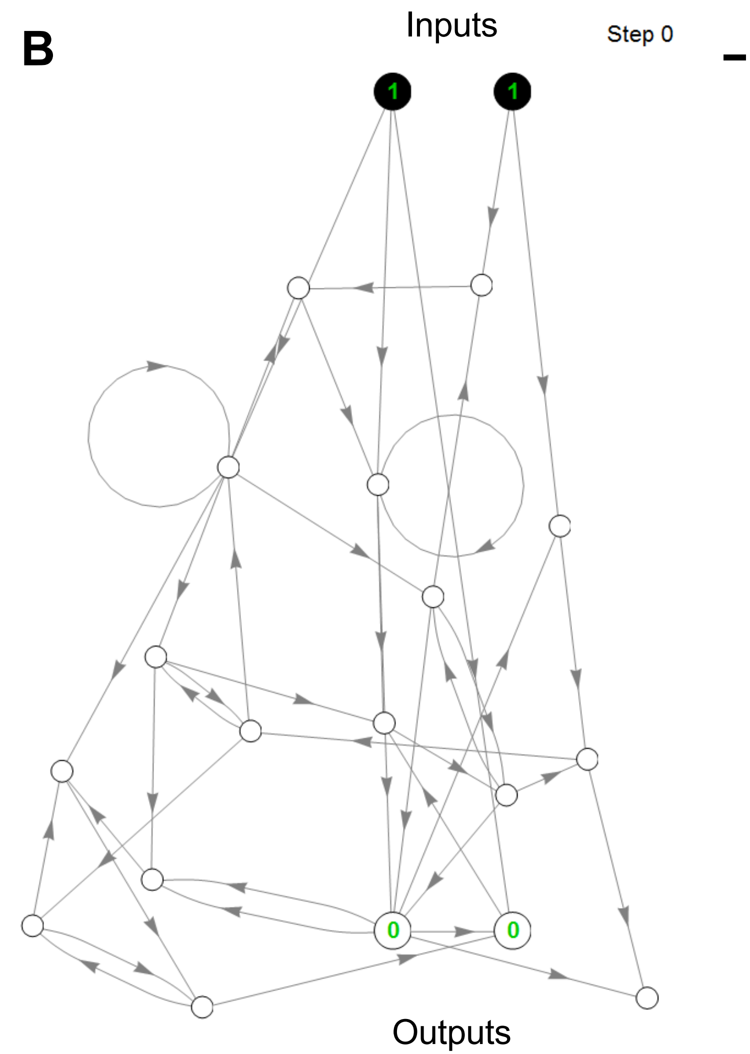
- **Eliminate the one with the differentiated response**

Complexity can address "whole system" robustness and stability

- **Consider designing a digital circuit to add two 1-bit numbers (a "half adder")**
  - This is among the most basic functions appearing in microelectronics
- **There are many ways of composing logic gates to implement this functionality**
- **The next slide shows two such circuits; each performs as a half adder when run for twenty steps**
  - Shown correctly adding 1 + 1 to get the binary result 10
  - They also give correct answers for the other possible inputs

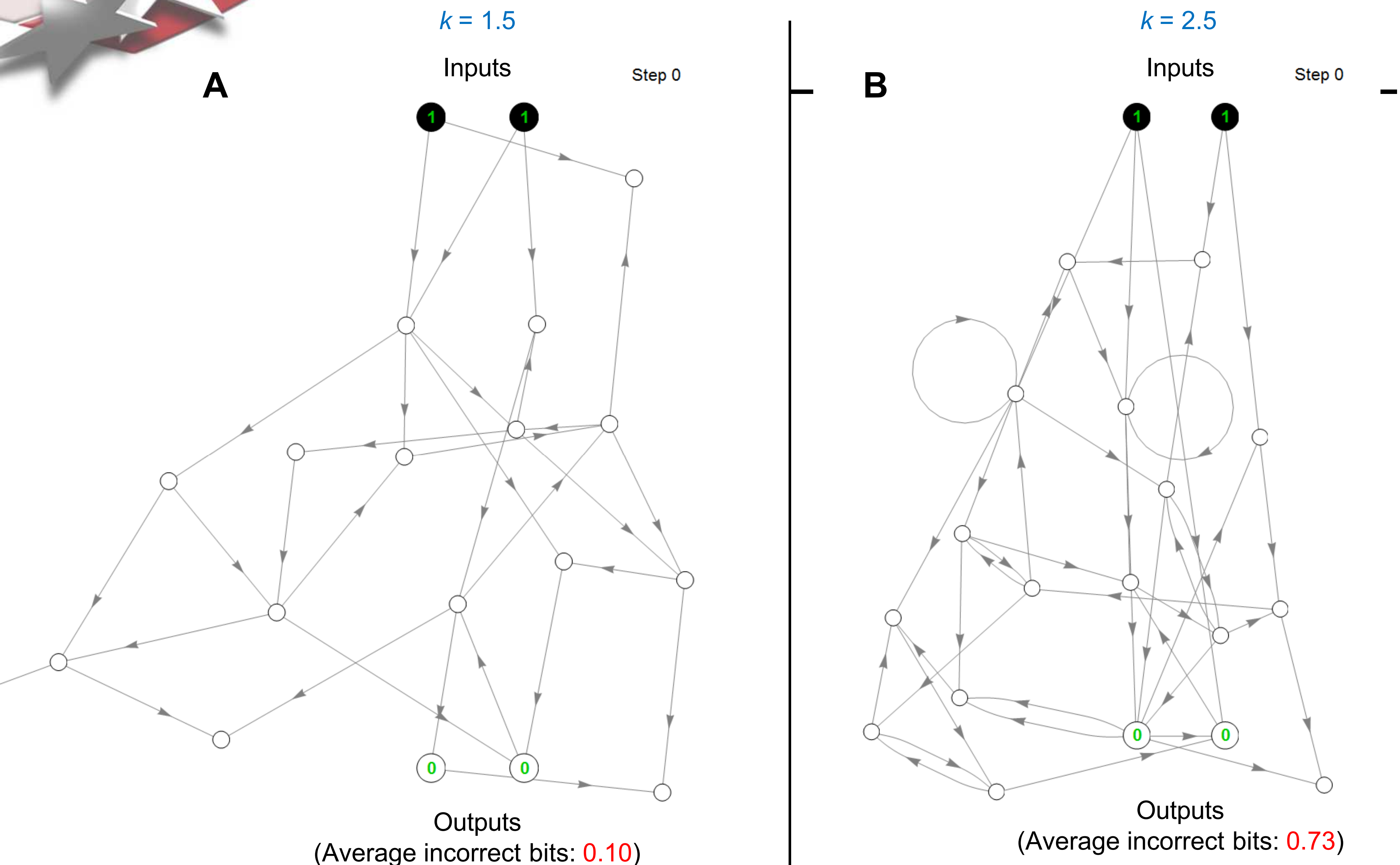
Sandia National Laboratories


A
Inputs
Step 0
1
1
0
0
Outputs


B
Inputs
Step 0
1
1
0
0
Outputs

What distinguishes the two implementations? *Resilience*

- **For this very simple functionality, both circuits can be verified by exhaustive testing**
- **More realistic circuits cannot be tested exhaustively, so we need to understand the effect of untested states**
- **In this example, we introduce occasional gate errors to represent unanticipated behavior**
- **The next slide shows a typical run of each circuit with a 1% error rate per gate update**
  - States that deviate from the ideal run are outlined in red
- **Circuit A has much less error in the final output (greater resilience) than circuit B – why?**
  - In this case, average inputs per node (*k*) makes the difference



k = 1.5
Inputs
Step 0
A
1
1
0
0
Outputs
(Average incorrect bits: 0.10)
k = 2.5
Inputs
Step 0
B
1
1
0
0
Outputs
(Average incorrect bits: 0.73)