Software Understanding for National Security

NIST

Dr. Douglas Ghormley
Dr. Christopher Harrison
Sandia National Laboratories
July 9, 2025

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2025-08422PE

https://suns.sandia.gov/
suns@sandia.gov

Introductions

Dr. Douglas Ghormley, Sandia National Laboratories

Senior Scientist

Dr. Christopher Harrison, Sandia National Laboratories

Distinguished Member of the Technical Staff

Mission Challenges from Software

The widespread use of software that cannot be adequately characterized places our society at immeasurable risk and degrades our integrated deterrence.

Unintentional Supply Chain Scenario

Example: CrowdStrike Outage

Software Challenge: A new configuration file triggered an existing, undiscovered parsing bug in a widely deployed component.

Impact: The bug caused the system to crash, resulting in major disruption across multiple sectors including financial, health care, emergency services, airlines, and government.

Intentional Supply Chain Scenario

Example: SolarWinds Attack

Software Challenge: Malicious code was inserted in a software update of a popular IT administration platform.

Impact: the malicious update was distributed to over 18,000 customers across the globe, infecting key industry (e.g., Microsoft) and USG entities.

National Security Scenario

Example: DOD's F22 Crossing the Dateline

Software Challenge: Unexpected software behavior caused in-flight failure of navigation, fuel, and communications systems.

Impact: F22's aborted the mission and followed fully other functioning aircraft back to base.

Critical Infrastructure Scenario

Example: Salt Typhoon

Software Challenge: Gains initial access to its victim networks by targeting external-facing assets using known vulnerabilities.

Impact: Affecting major telecom companies and resulting in the theft of sensitive correspondence data, including metadata and call details.

Mission Challenges from Software

The widespread use of software that cannot be adequately characterized places our society at immeasurable risk and degrades our integrated deterrence.

Unintentional Supply Chain Scenario

Example: CrowdStrike Outage

Software Challenge: A new configuration file triggered an existing, undiscovered parsing bug in a widely deployed component.

Impact: The bug caused the system to crash, resulting in major disruption across multiple sectors including financial, health care, emergency services, airlines, and government.

Intentional Supply Chain Scenario

Example: SolarWinds Attack

Software Challenge: Malicious code was inserted in a software update of a popular IT administration platform.

Impact: the malicious update was distributed to over 18,000 customers across the globe, infecting key industry (e.g., Microsoft) and USG entities.

National Security Scenario

Example: DOD's F22 Crossing the Dateline

Software Challenge: Unexpected software behavior caused in-flight failure of navigation, fuel, and communications systems.

Impact: F22's aborted the mission and followed fully other functioning aircraft back to base.

Critical Infrastructure Scenario

Example: Salt Typhoon

Software Challenge: Gains initial access to its victim networks by targeting external-facing assets using known vulnerabilities.

Impact: Affecting major telecom companies and resulting in the theft of sensitive correspondence data, including metadata and call details.

Two radically different approaches: addressing each problem individually vs. contemplating the big picture.

Full Scope of the Problem

Examples are entirely notional, for illustration purposes only.

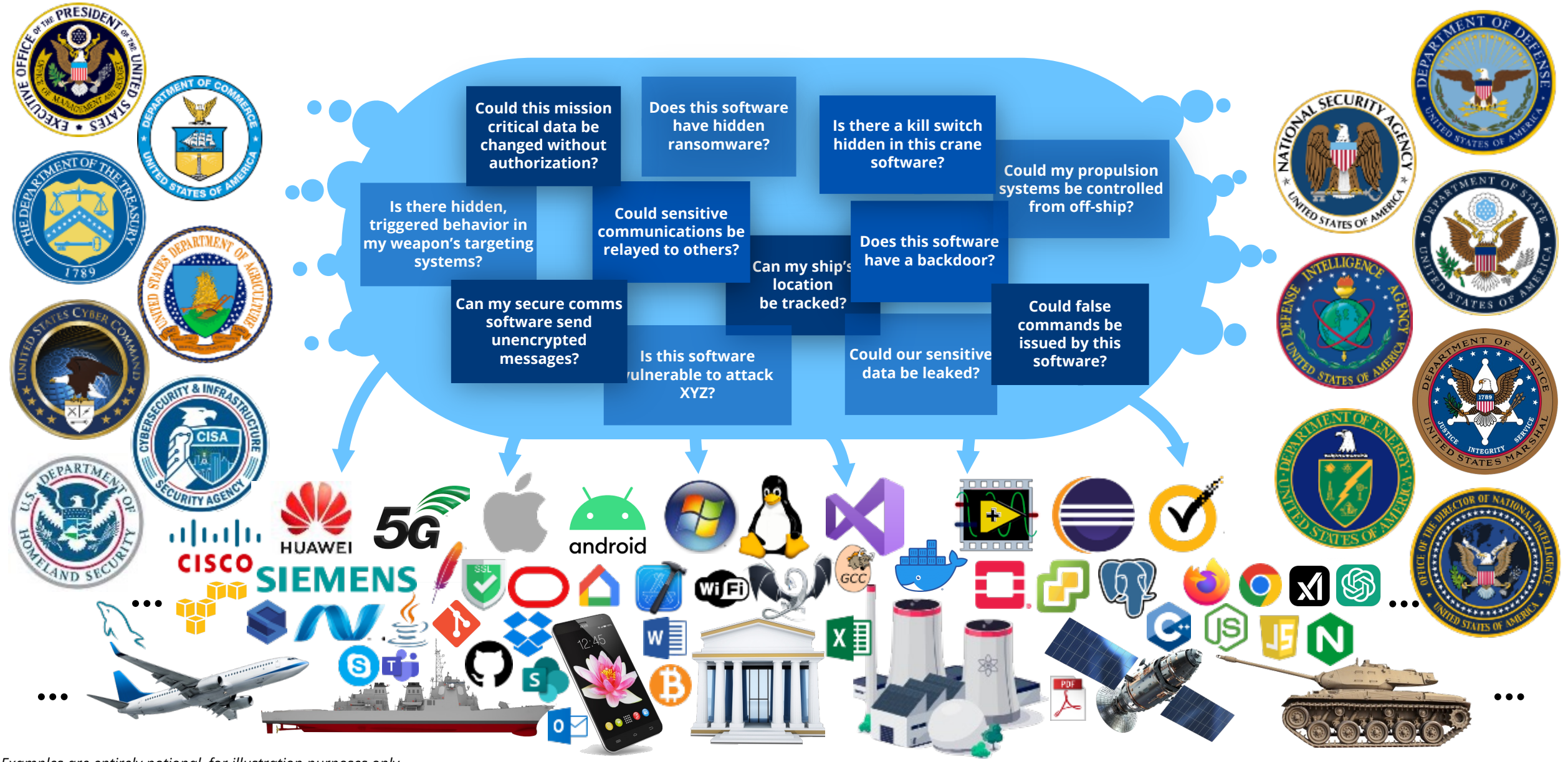
Full Scope of the Problem

Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem



Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

SUNS

Ideally, mission owners would be able to routinely analyze any mission critical software to answer any mission question.

Could this mission critical data be changed without authorization?

Does this software have hidden ransomware?

Is there a kill switch hidden in this crane software?

Could my propulsion systems be controlled from off-ship?

Is there hidden, triggered behavior in my weapon's targeting systems?

Could sensitive communications be relayed to others?

Does this software have a backdoor?

Can my ship's location be tracked?

Can my secure comms software send unencrypted messages?

Is this software vulnerable to attack XYZ?

Could our sensitive data be leaked?

Could false commands be issued by this software?

Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

SUNS

Ideally, mission owners would be able to routinely analyze any mission critical software to answer any mission question.

Could this mission critical data be changed without authorization?

Does this software have hidden ransomware?

Is there a kill switch hidden in this crane software?

Could my propulsion systems be controlled

Observations

1. **Capabilities to routinely answer mission critical questions about software do not exist today.**

What's needed is an understanding of the software's possible behavior.

Examples are entirely notional, for illustration purposes only.

Software Understanding: *Definition*

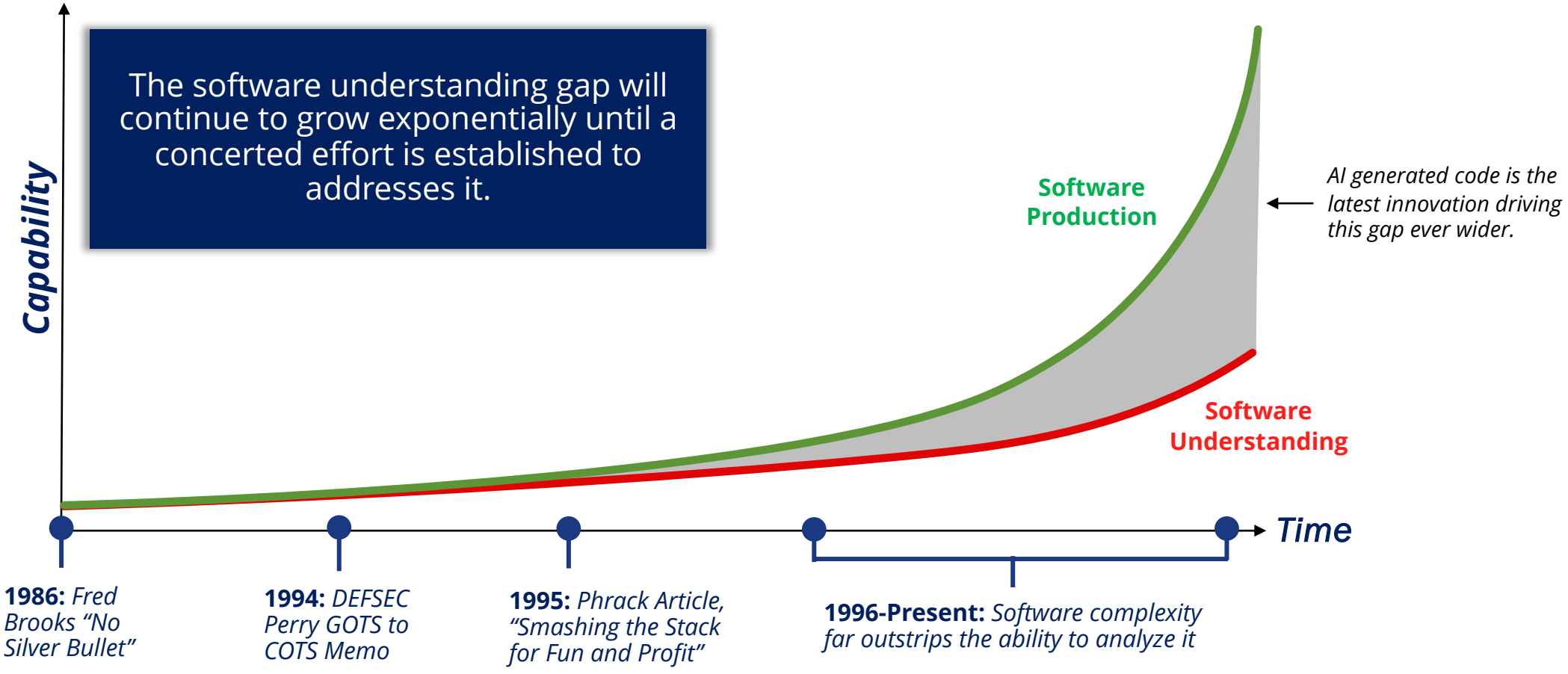
SUNS

"Software Understanding"

The practice of constructing or assessing software-controlled systems to verify or characterize their behaviors across all conditions – normal, abnormal, and hostile.

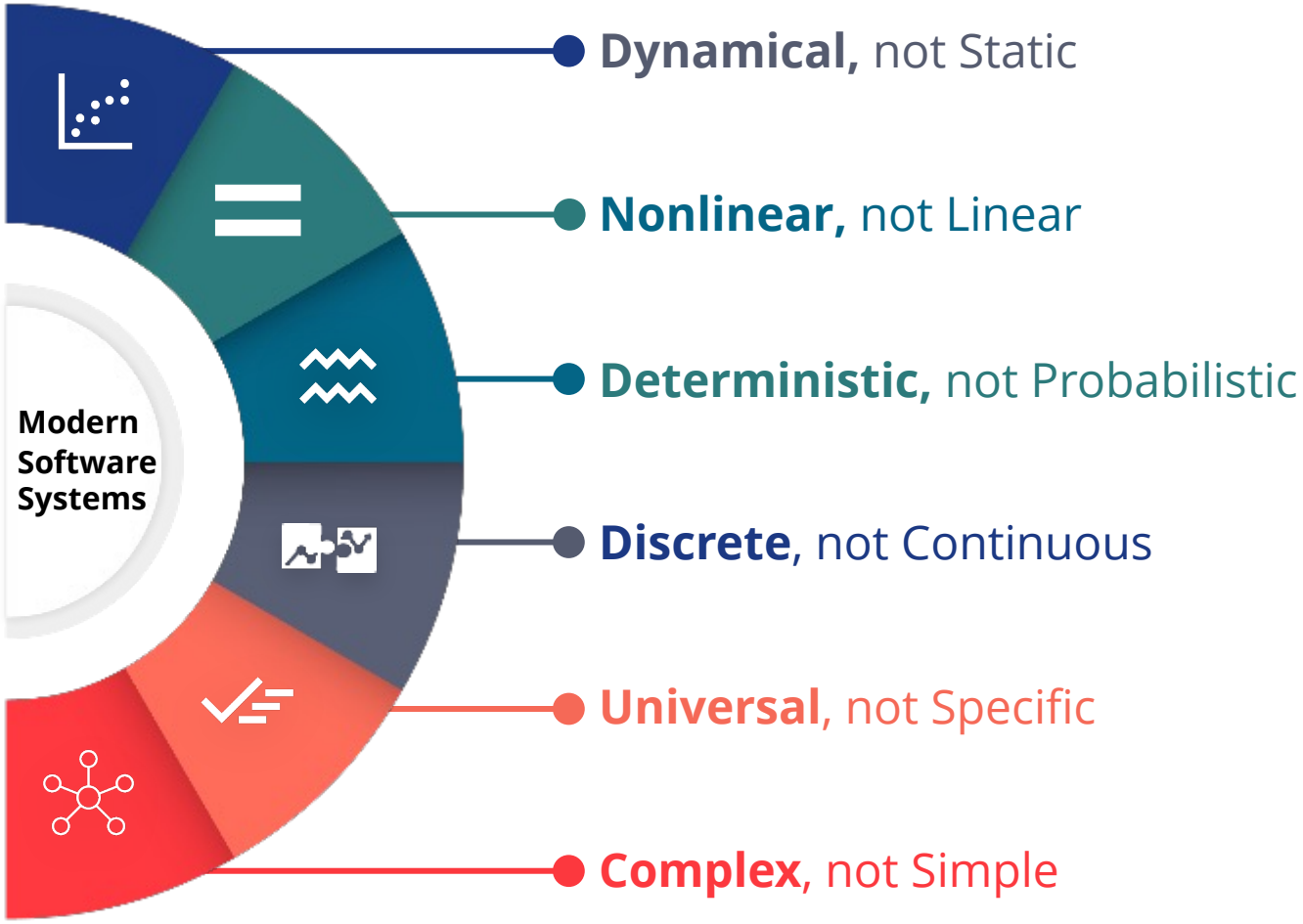
The Software Understanding Gap

Society's ability to produce software has far outstripped our ability to understand it – this gap drives the inscrutability of software behavior that imperils our missions.



This history is traced in more detail in "The National Need for Software Understanding," available at htts://suns.sandia.gov/

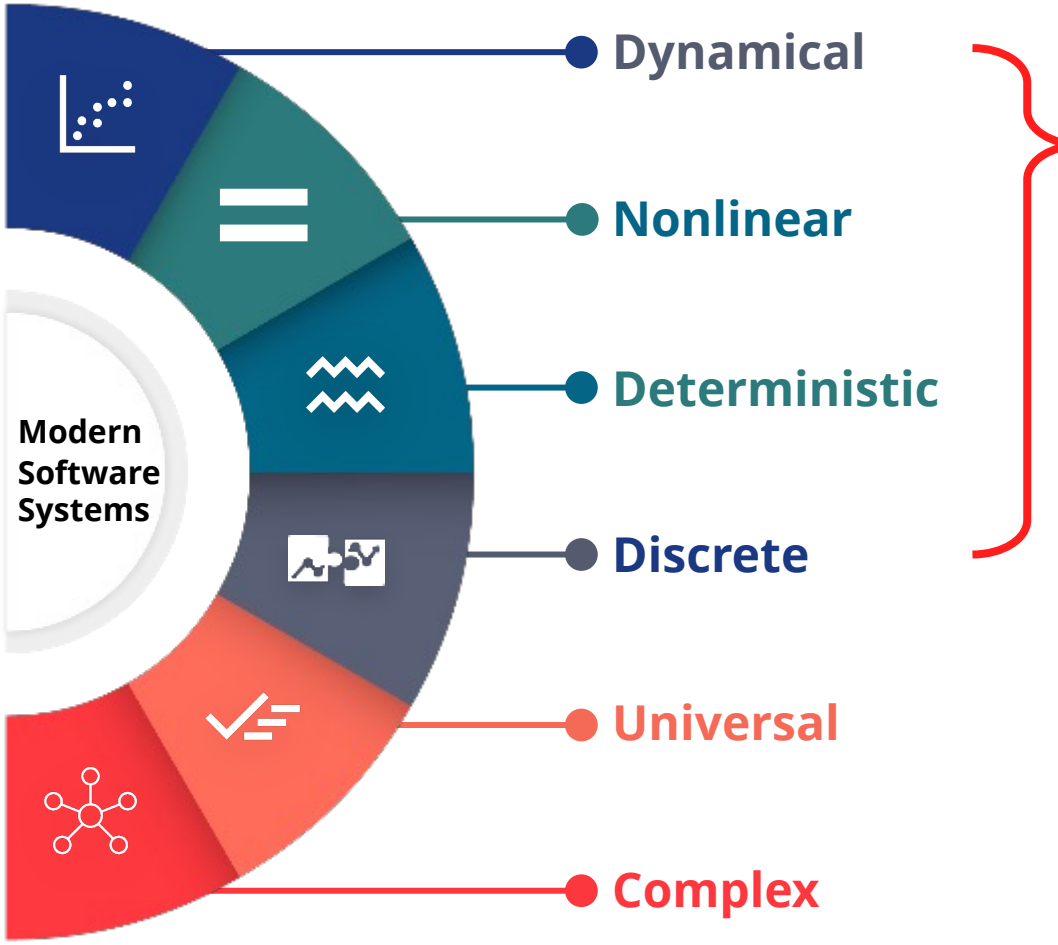
Characteristics of Modern Software Systems



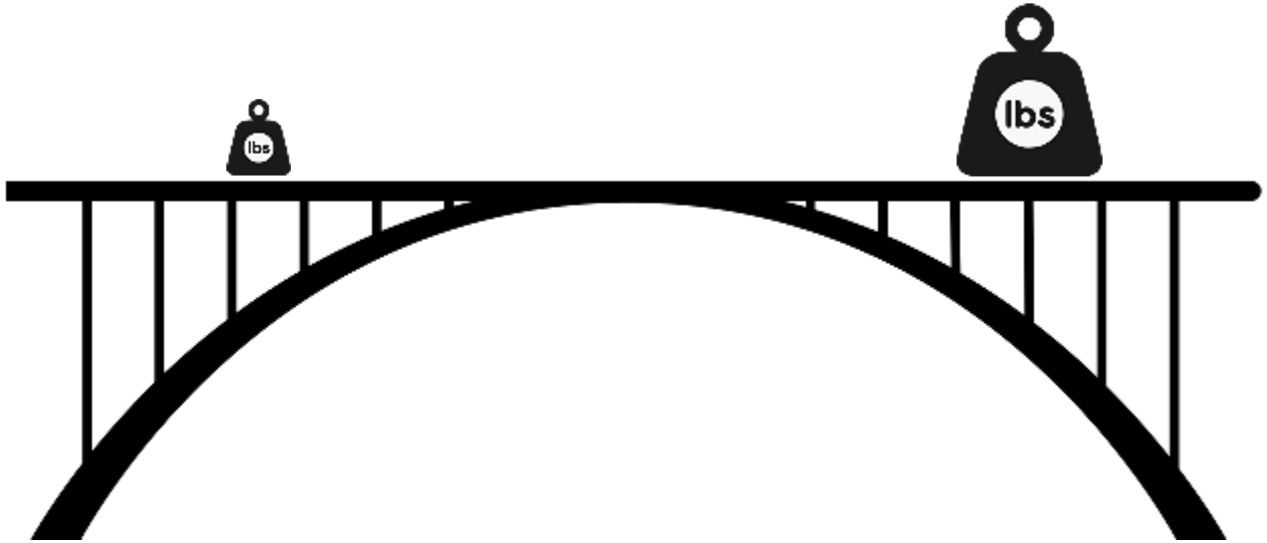
This combination of characteristics is the hardest of the set of options for analysis.

Also, these same characteristics are what makes software so effective in meeting functional requirements.

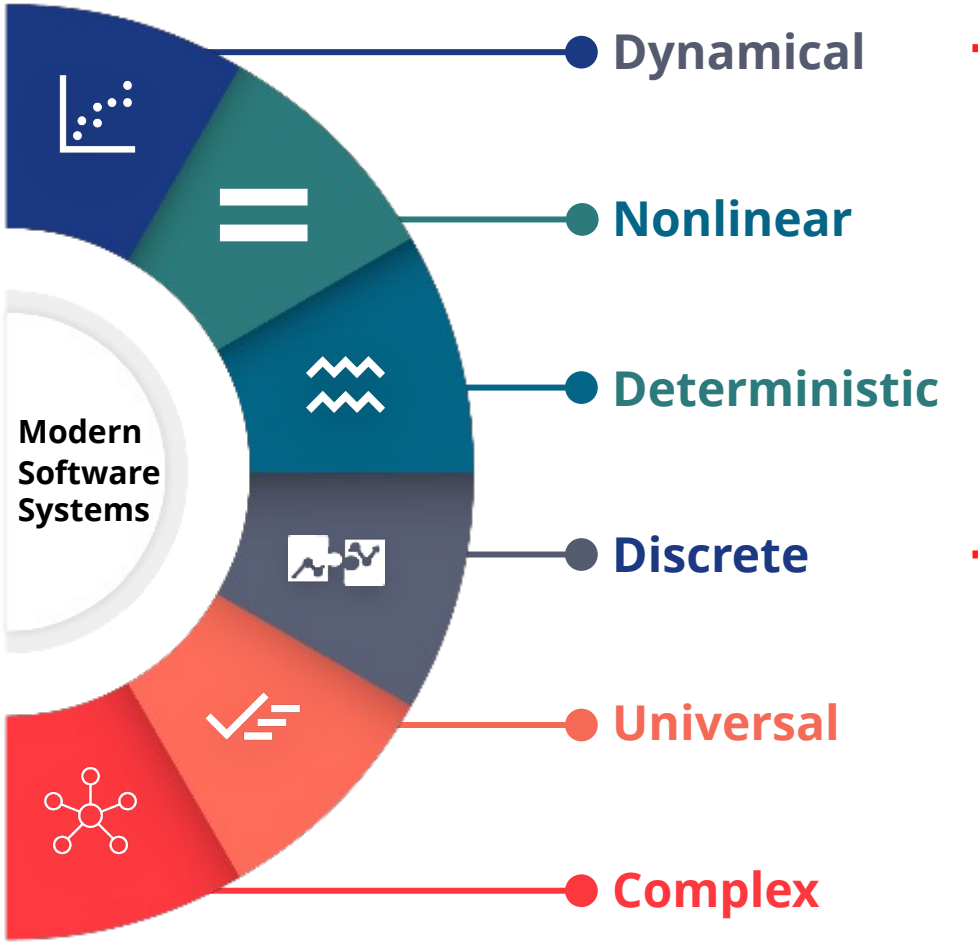
These Systems are Inscrutable

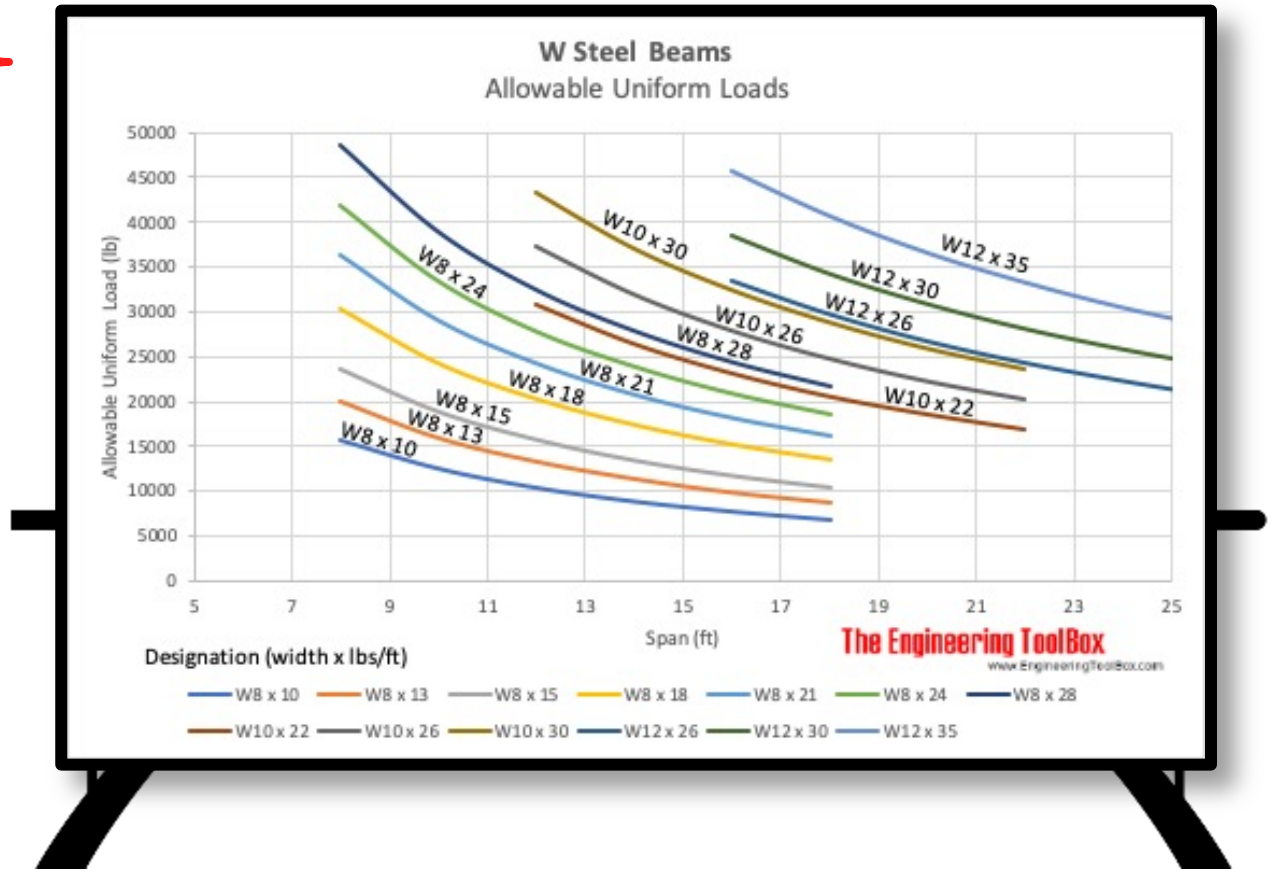


Our lives are steeped in continuous systems.

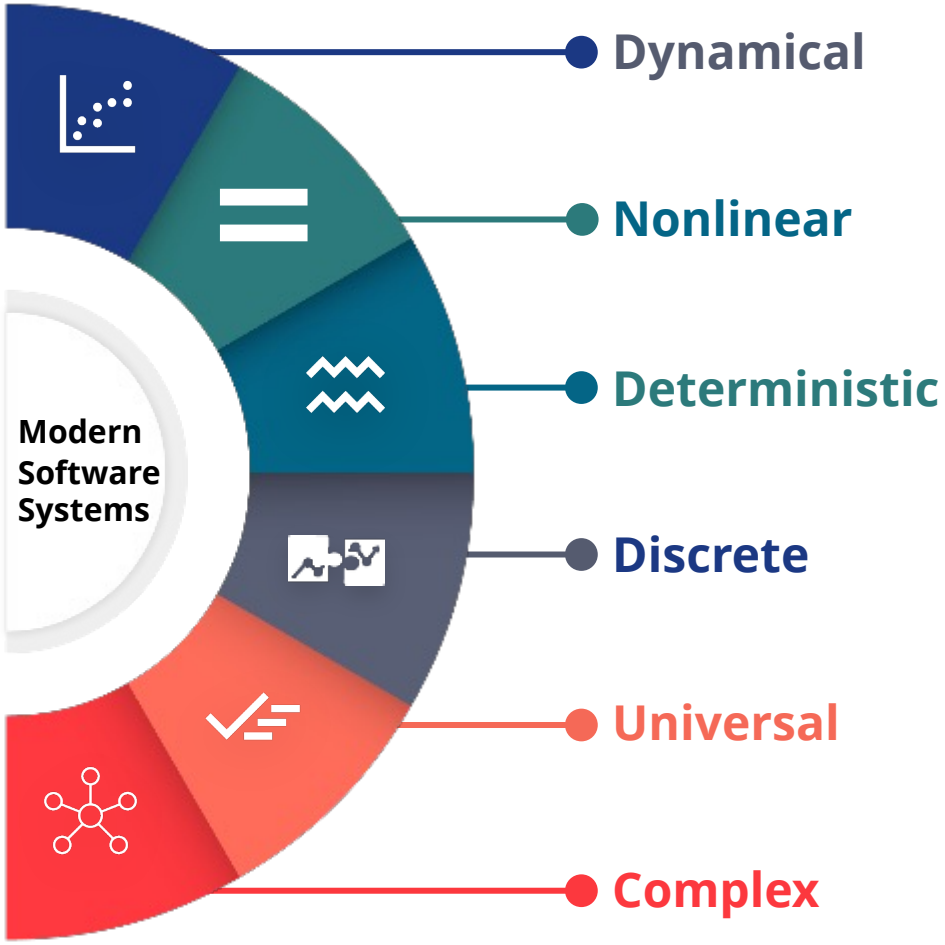


These Systems are Inscrutable

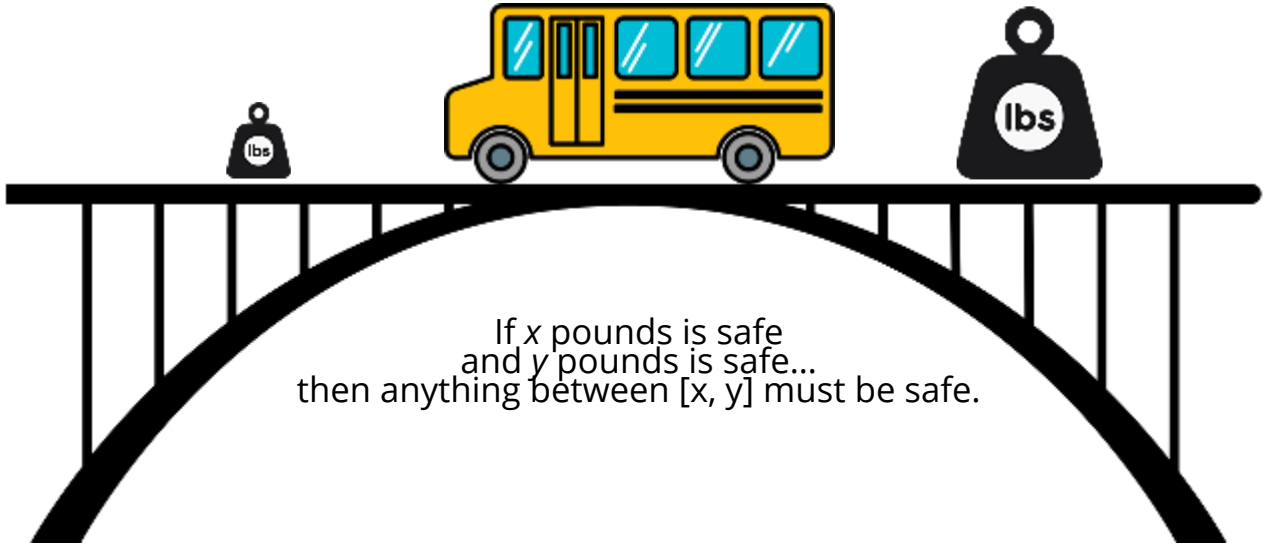




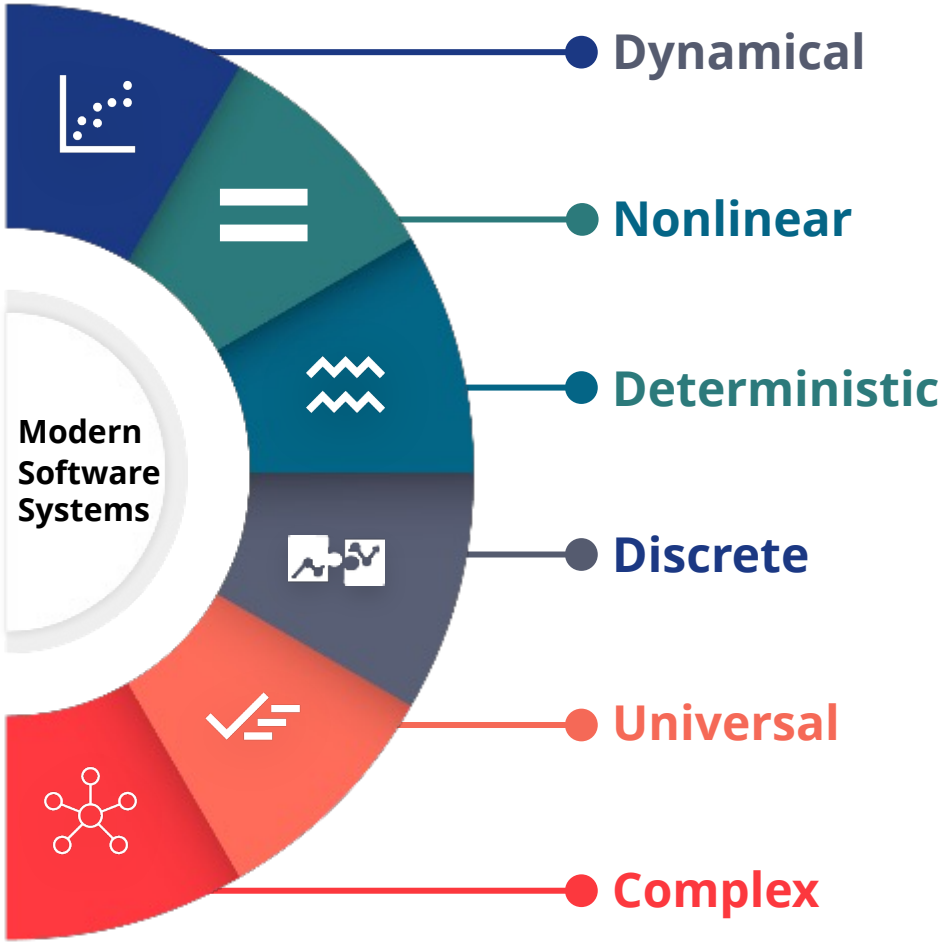
These Systems are Inscrutable



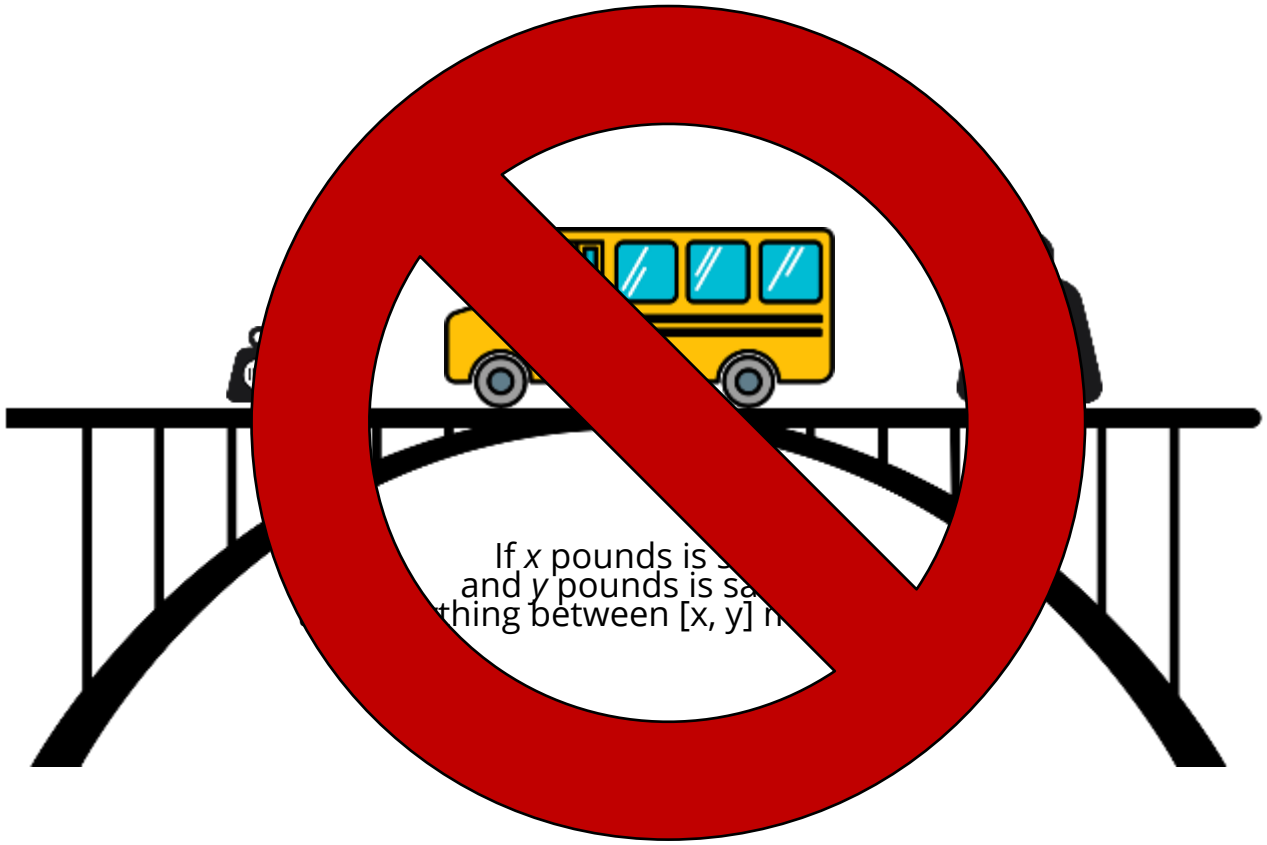
Our lives are steeped in continuous systems.



These Systems are Inscrutable



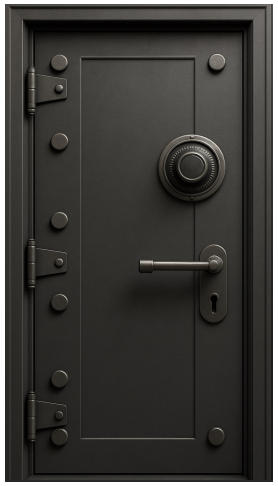
Software is Discrete, not continuous.



Example: Emergent Behavior

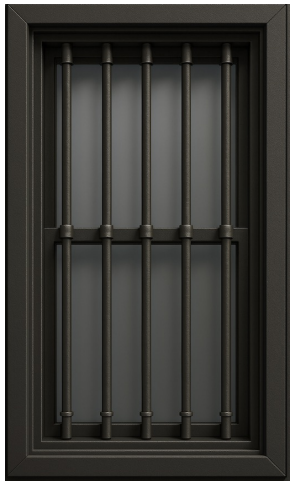
Interactions between even simple components create emergent behaviors. The more complex the components or complicated the interactions, the more behavior may degrade safety or security.

Component 1 is secure



The producer makes a solid steel door that is impossible to pick.

Component 2 is secure



The producer makes barred windows that do not allow direct human entry.

System is Insecure

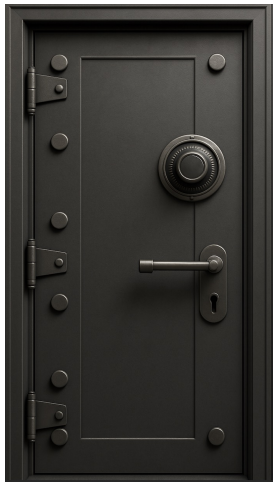
The producer builds windows beside the door creating emergent behavior.

Each component is completely independently secure…

Example: Emergent Behavior

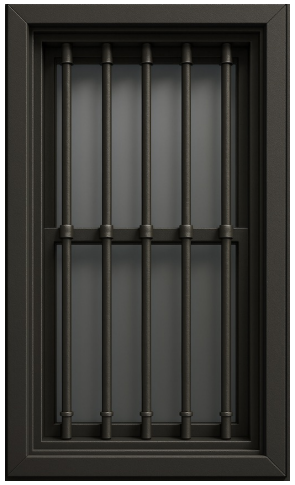
Interactions between even simple components create emergent behaviors. The more complex the components or complicated the interactions, the more behavior may degrade safety or security.

Component 1 is secure



The producer makes a solid steel door that is impossible to pick.

Component 2 is secure



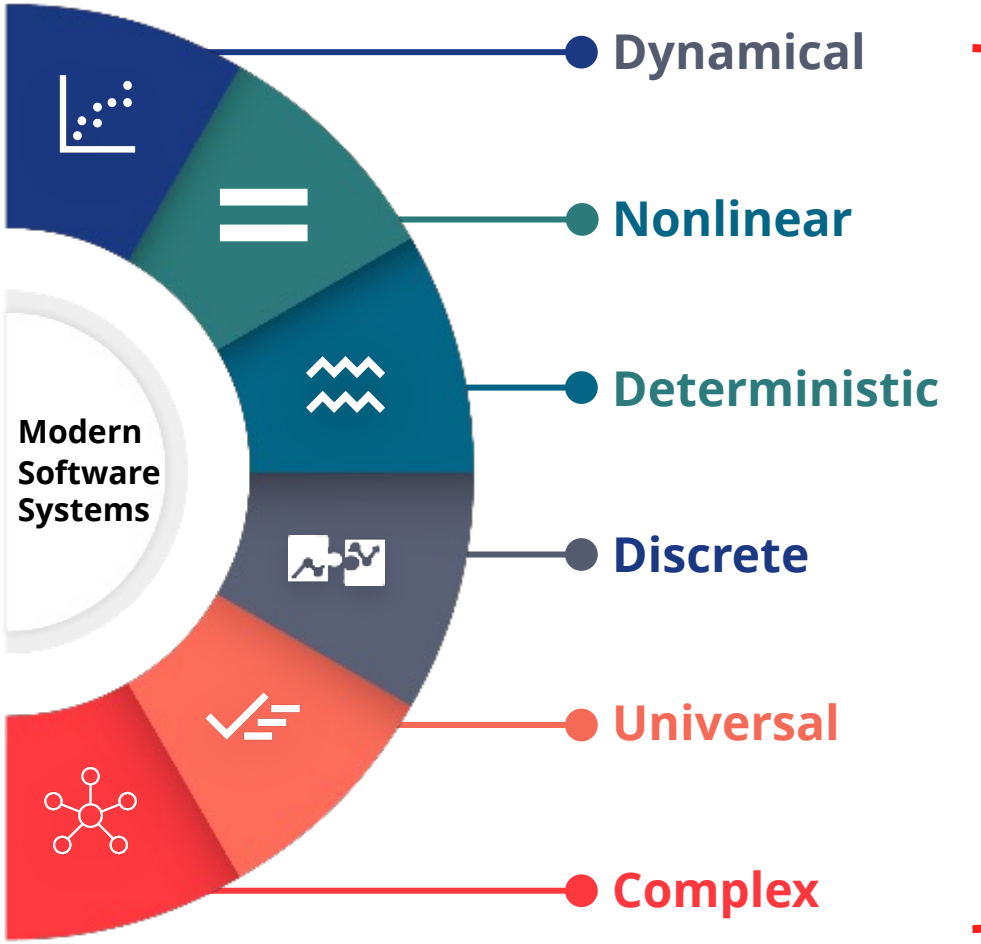
The producer makes barred windows that do not allow direct human entry.

System is Insecure

Adversary exploits behavior to break window and open door from inside.

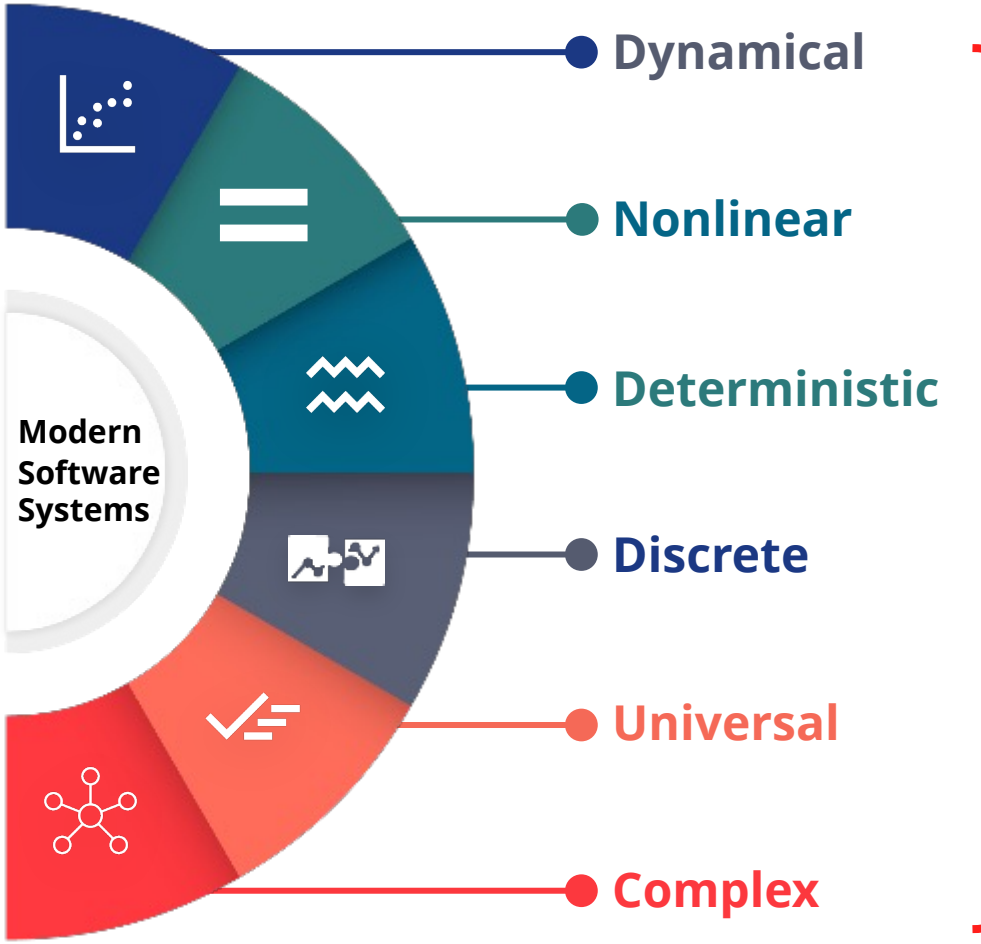
Each component is completely independently secure…but they interact in way that creates an insecure system. System properties can put each other in tension.

These Systems are Inscrutable

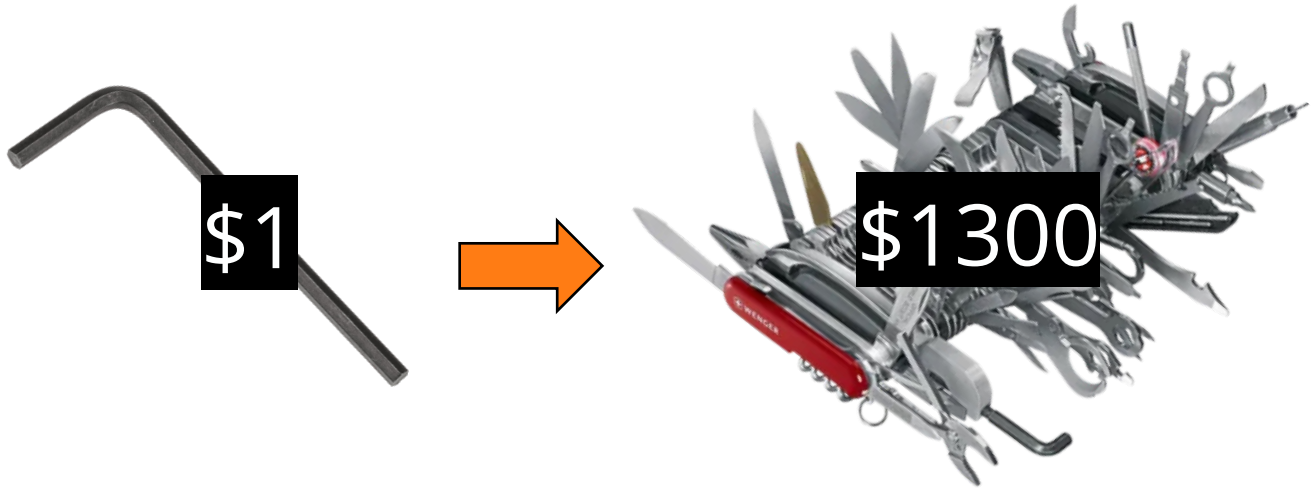


We simulate simplicity with cheap complexity.

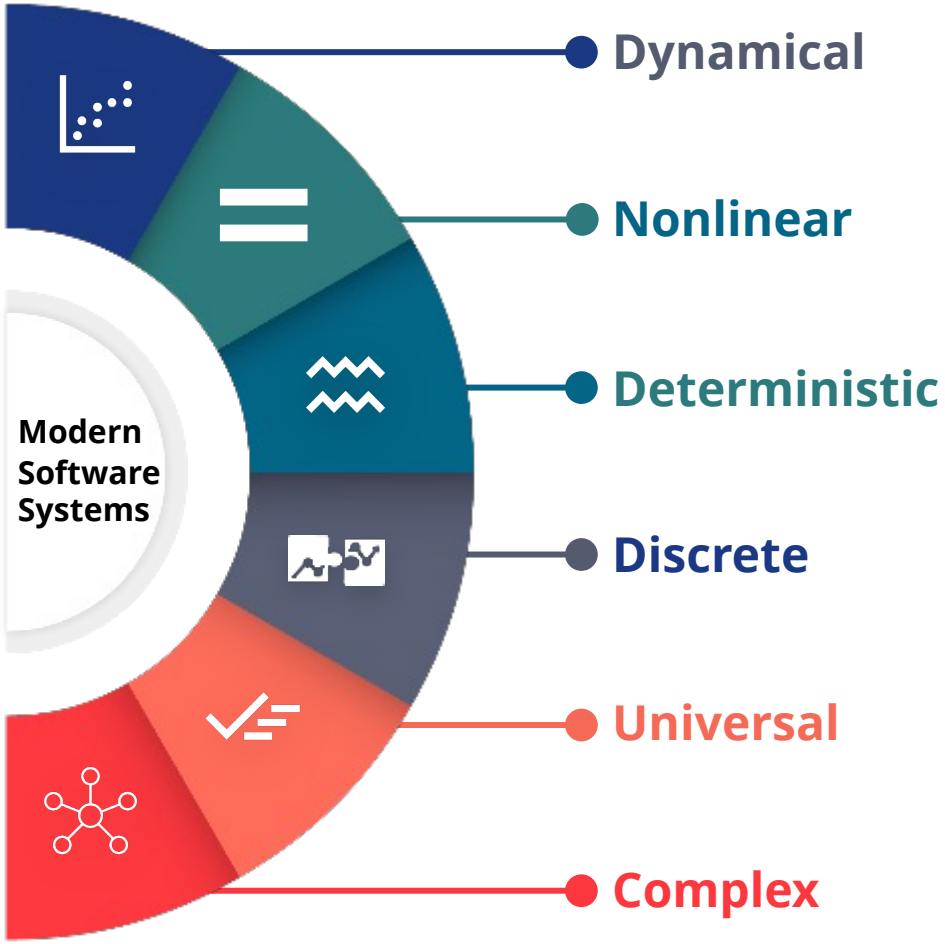
These Systems are Inscrutable



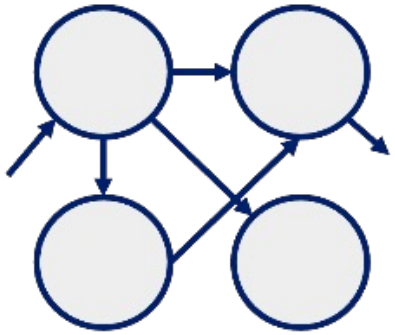
We simulate simplicity with cheap complexity.



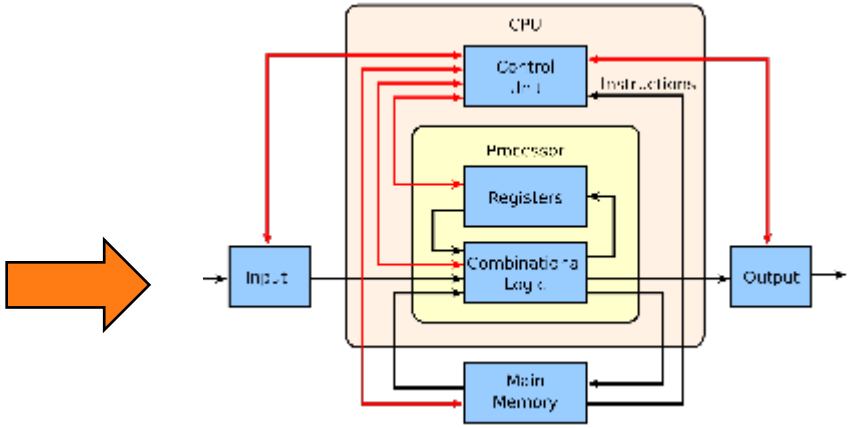
These Systems are Inscrutable



We simulate simplicity with cheap complexity.

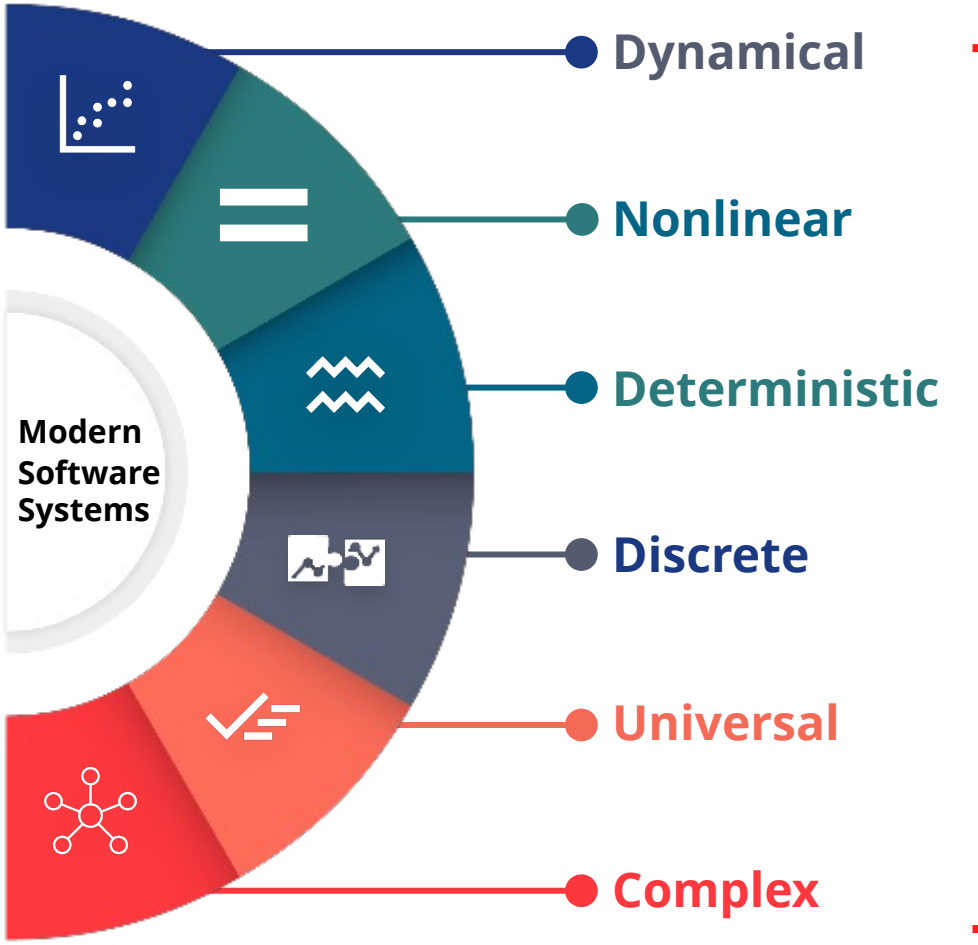


What we need:
simple state machine

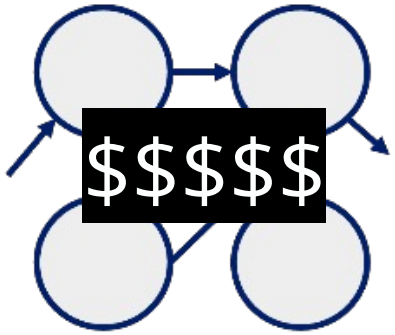


What we have:
universal, general-purpose hardware and software

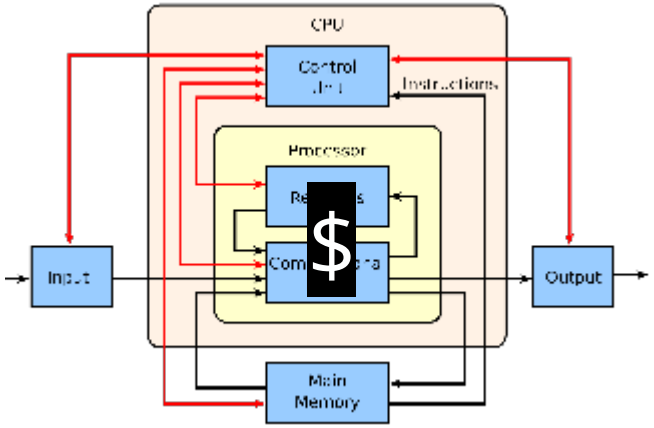
These Systems are Inscrutable



We simulate simplicity with cheap complexity.

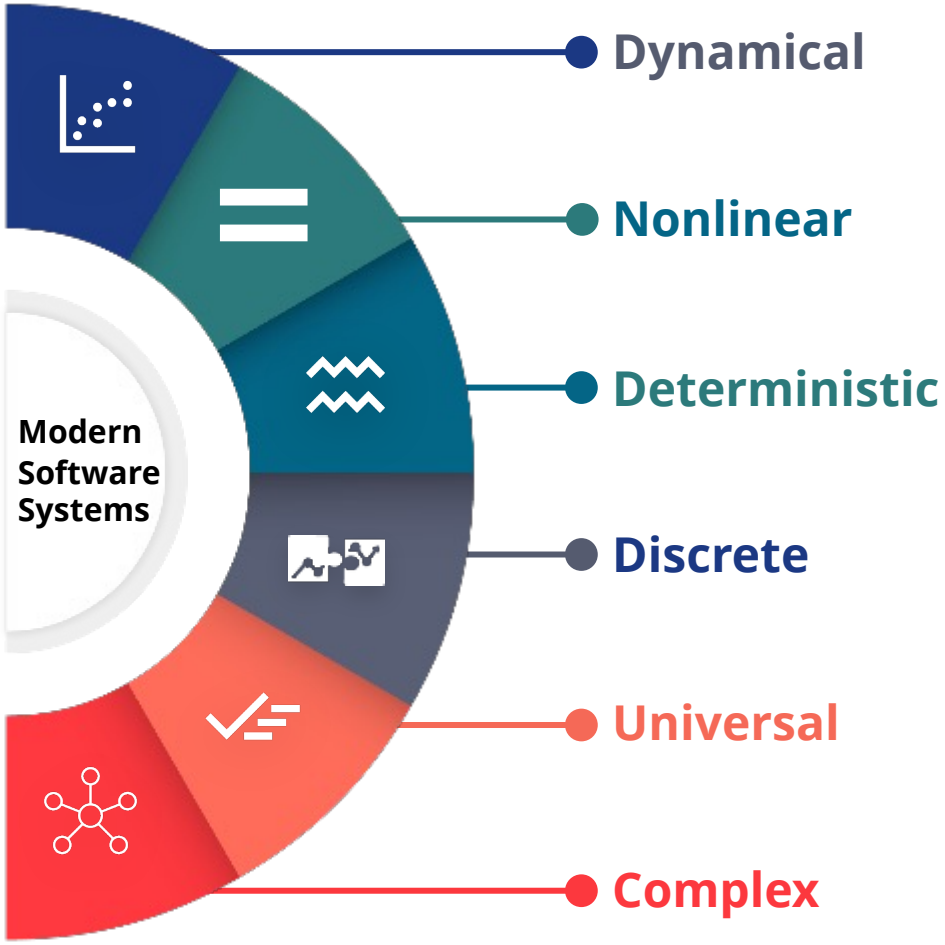


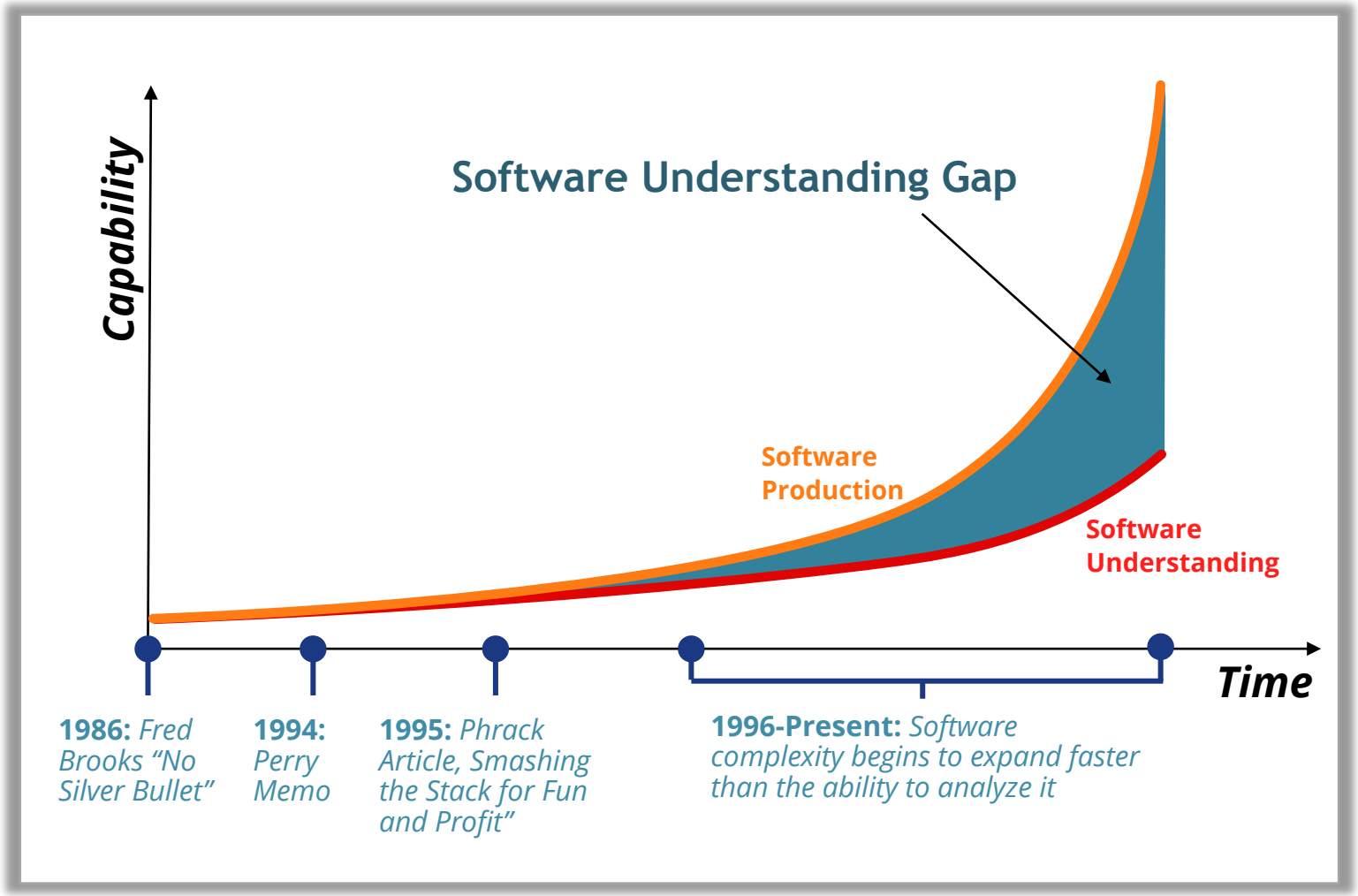
What we need:
simple state machine



What we have:
universal, general-purpose hardware and software

These Systems are Inscrutable





Full Scope of the Problem

We replace mission questions with easily assessed proxies.

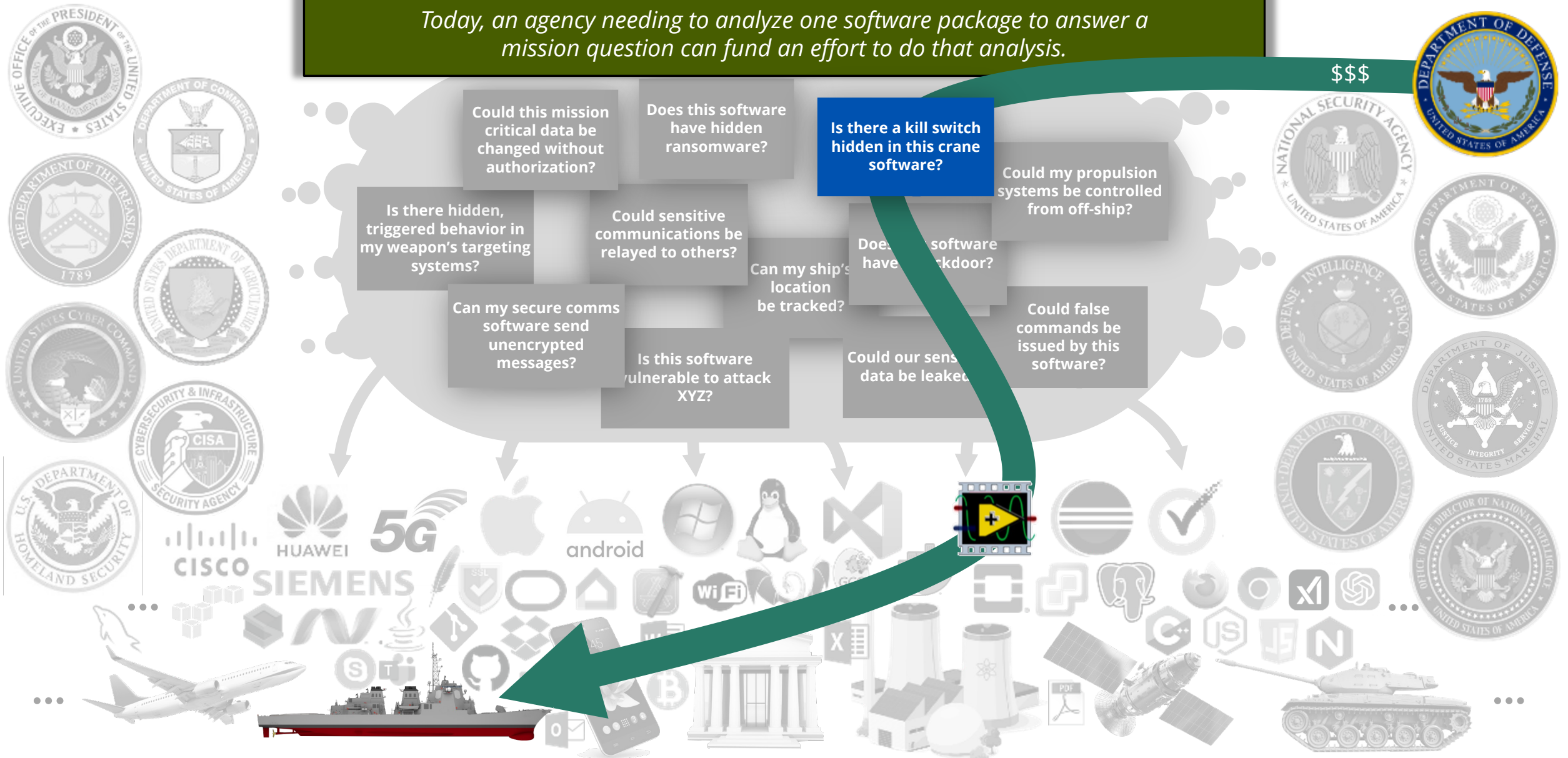
- Do we run tests on the software before use? (Testing)
- Does the supplier certify that they use secure development practices? (Attestation)
- Does the software have patterns of code known to be malicious? (Signatures)
- What software components does the supplier attest to? (Software Bill of Materials)
- Do we trust the supplier of the software? (Provenance)
- Do we use software that observes the software under scrutiny? (Monitoring)

These proxies have positive utility but are *insufficient* for the assurance needs of national security and critical infrastructure systems.

Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis.



Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

SUNS

Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis.

$$$

$$$

Could this mission critical data be changed without authorization?

Does this software have hidden ransomware?

Is there a kill switch hidden in this crane software?

Could my propulsion systems be controlled from off-ship?

Is there hidden, triggered behavior in my weapon's targeting systems?

Could sensitive communications be relayed to others?

Can my sh… locatio… be track…

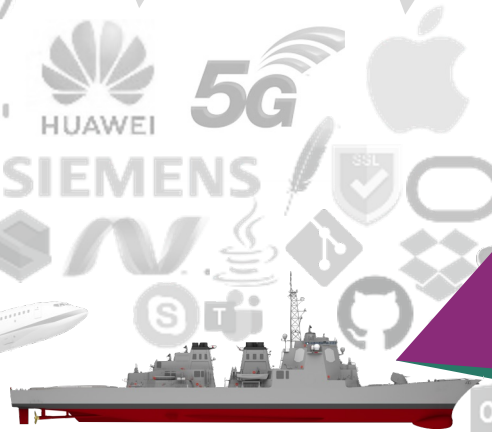
…oes… …have… …software …ckdoor?

Can my secure comms software send unencrypted messages?

Is this software …ulnerable to attack XYZ?

Could our sens… data be leake…

Could false commands be issued by this software?



Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis

$$$
$$$
$$$

Could this mission critical data be changed without authorization?

Does this software have hidden ransomware?

Is there a kill switch hidden in this crane software?

Could my propulsion systems be controlled from off-ship?

Is there hidden, triggered behavior in my weapon's targeting systems?

Could sensitiv… communicati… relayed t…s?

Can my sh… locatio… be track…

…oes… …software have …ckdoor?

Can my secure com… software se… unencry… mes…

Is this software …ulnerable to attack XYZ?

Could our sens… data be leaked

Could false commands be issued by this software?

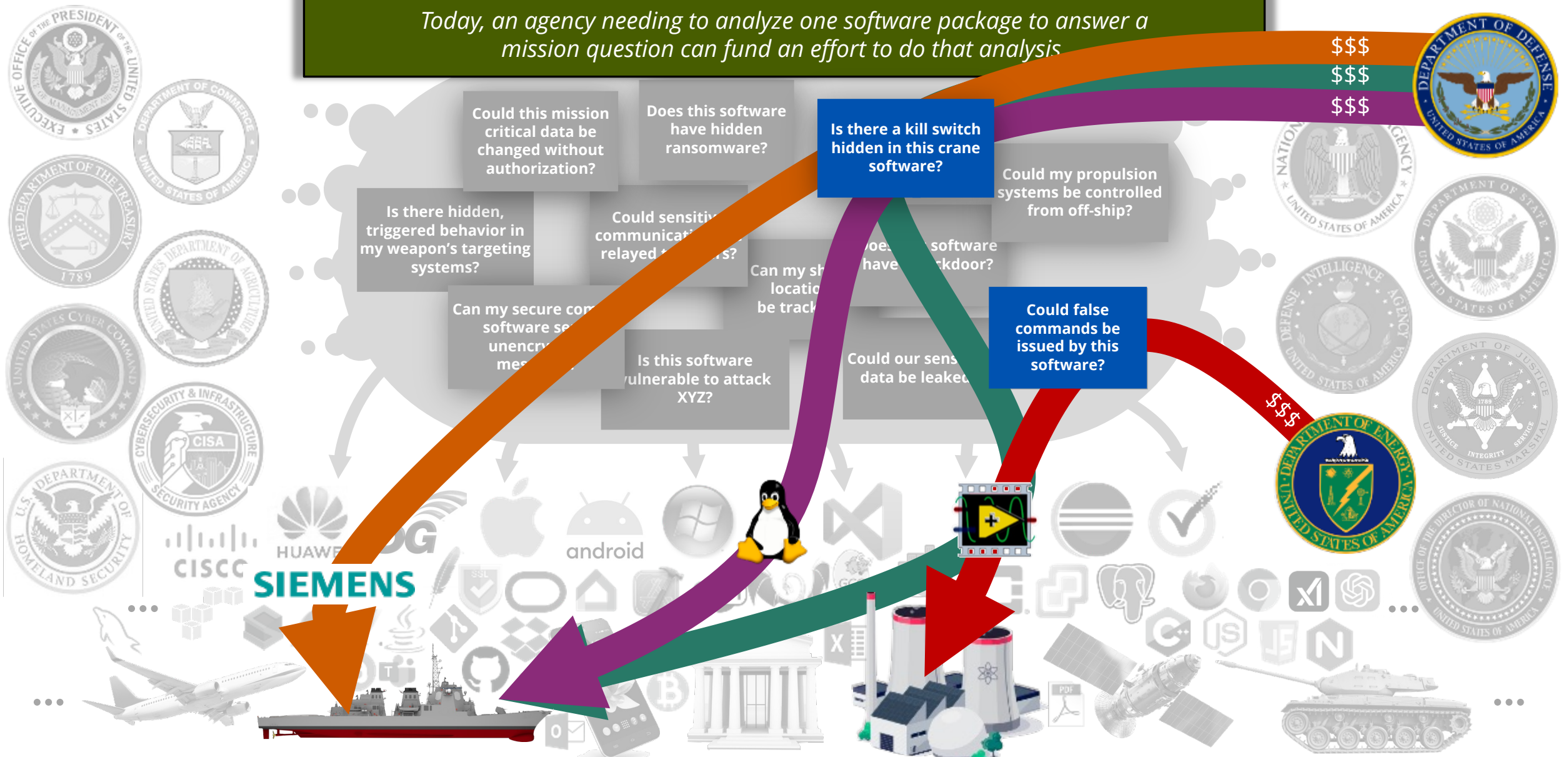
SIEMENS

Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem

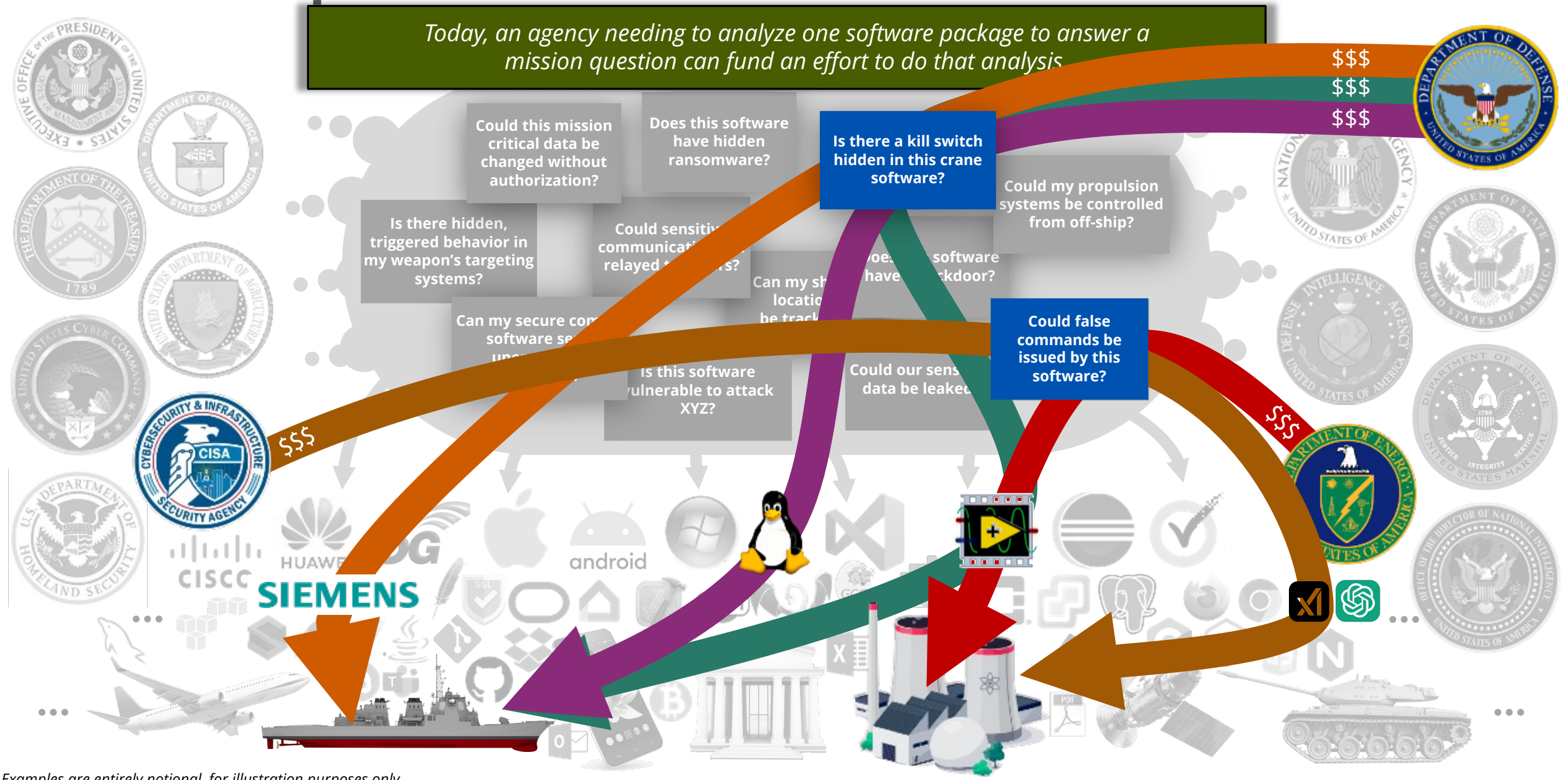
SUNS

Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis



Examples are entirely notional, for illustration purposes only.

Full Scope of the Problem



Examples are entirely notional, for illustration purposes only.

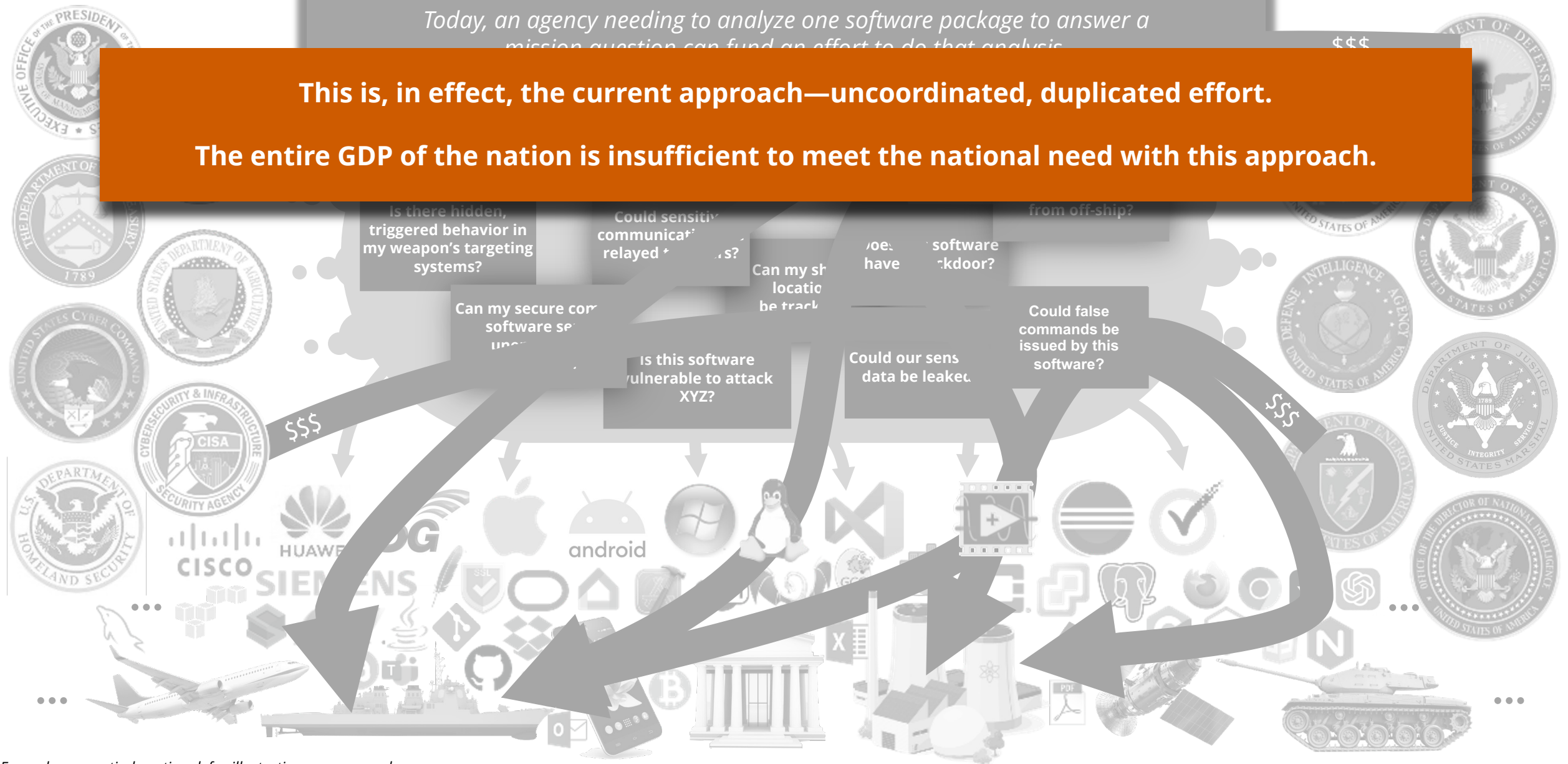
Technical Opportunity

SUNS

Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis

This is, in effect, the current approach—uncoordinated, duplicated effort.

The entire GDP of the nation is insufficient to meet the national need with this approach.



Examples are entirely notional, for illustration purposes only.

Technical Opportunity

SUNS

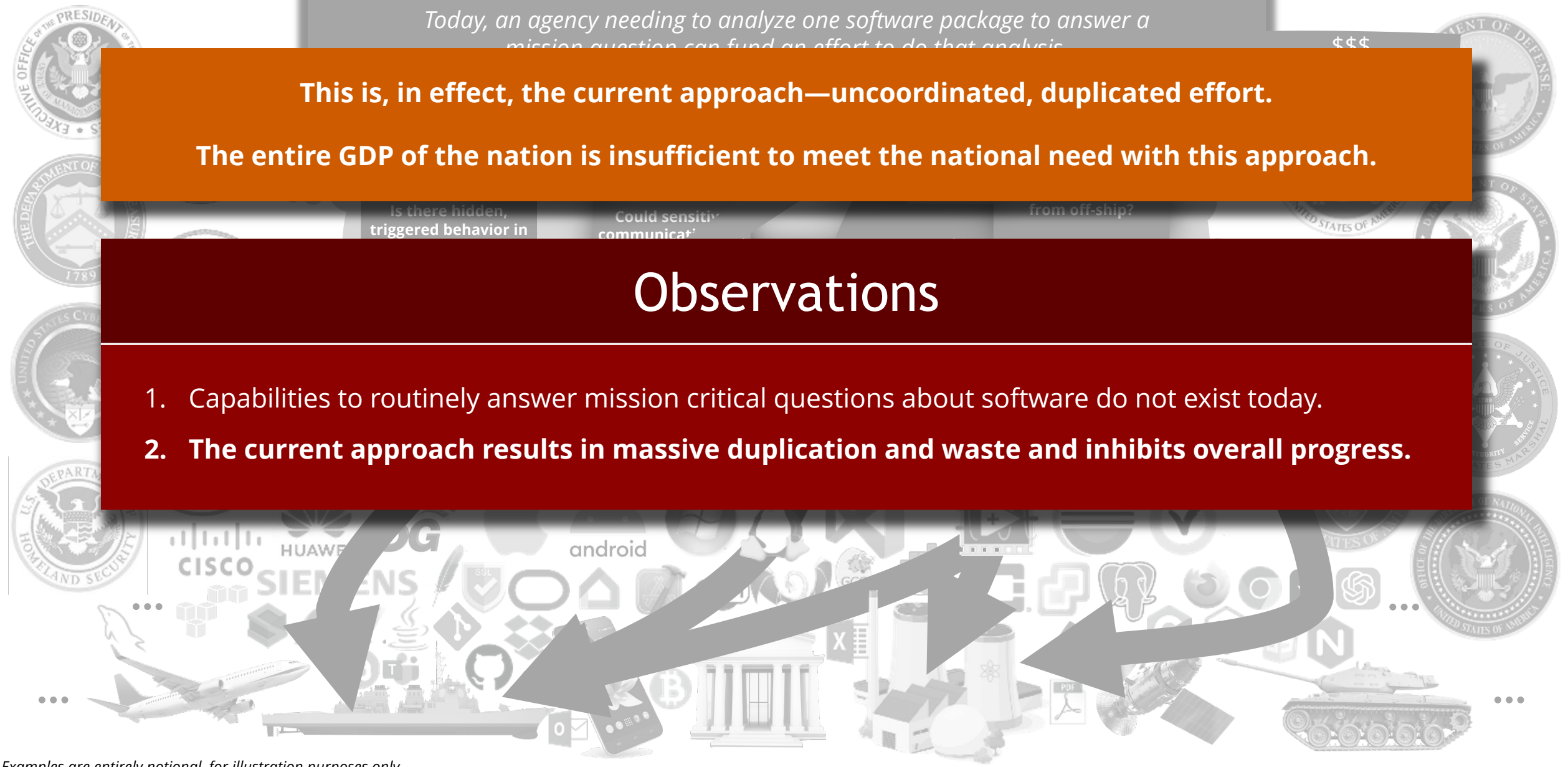
Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis

This is, in effect, the current approach—uncoordinated, duplicated effort.

The entire GDP of the nation is insufficient to meet the national need with this approach.

Observations

1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. **The current approach results in massive duplication and waste and inhibits overall progress.**



Examples are entirely notional, for illustration purposes only.

Technical Opportunity

SUNS

Today, an agency needing to analyze one software package to answer a mission question can fund an effort to do that analysis

This is, in effect, the current approach—uncoordinated, duplicated effort.

The entire GDP of the nation is insufficient to meet the national need with this approach.

Observations

1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. **The current approach results in massive duplication and waste and inhibits overall progress.**

However, there is considerable potential commonality in the technical foundations across these examples.

A coordinated, collaborative strategy could create radically improved capabilities with a positive return on investment.

Examples are entirely notional, for illustration purposes only.

Why Can’t We Just Ask This Mission Question?

SUNS

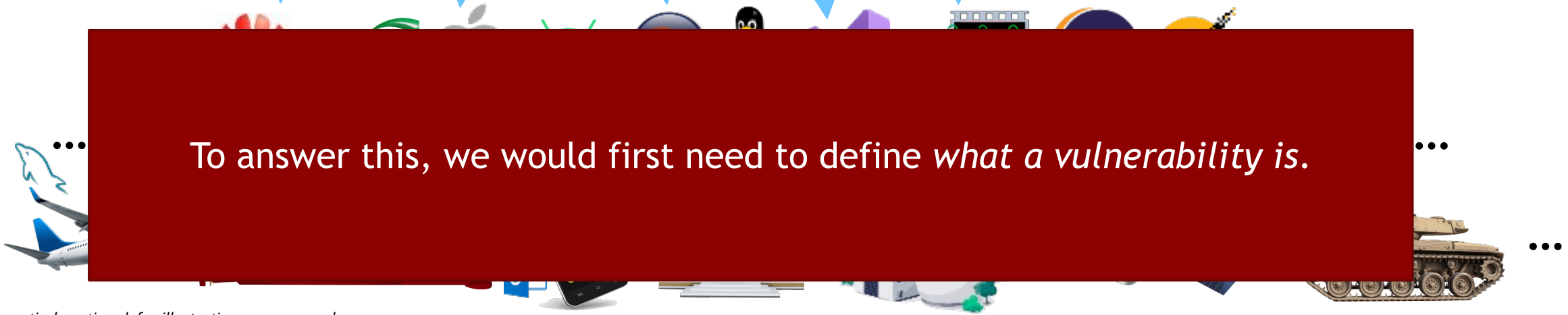
Does my system have a vulnerability in it?

Examples are entirely notional, for illustration purposes only.

Why Can't We Just Ask This Mission Question?

SUNS

Does my system have a vulnerability in it?



To answer this, we would first need to define *what a vulnerability is*.

Examples are entirely notional, for illustration purposes only.

Where Do Vulnerabilities Come From?

SUNS

Vulnerabilities arise when actual system behavior differs from expected system behavior.

Where Do Vulnerabilities Come From?

SUNS

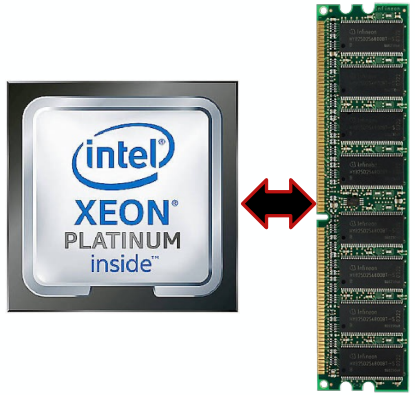
Math

$\sum$

Rule: Don’t divide by zero.

Where Do Vulnerabilities Come From?

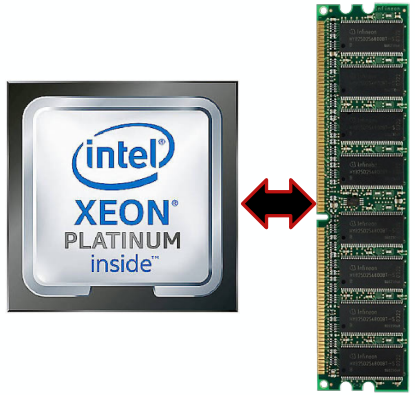
SUNS

Math	CPU & Memory
$\sum$	

Rule: Don’t use uninitialized memory.
Rule: Don’t dereference an invalid pointer.

Where Do Vulnerabilities Come From?

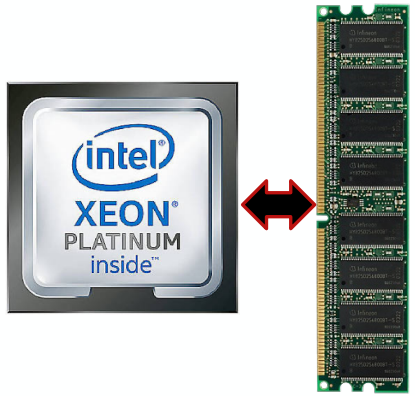
SUNS

Math	CPU & Memory	Operating Systems
$\sum$		

Rule: The login process should not have a hardcoded password.
Rule: User A should not be able to modify files belonging to User B unless User A granted permission.

Where Do Vulnerabilities Come From?

SUNS

Math	CPU & Memory	Operating Systems	Applications
$\sum$			

Rule: Input from one user must be sanitized before displaying to another user.
Rule: Do not serialize executable content in cookie data.

Example: Log4J – Feature or Vulnerability?

CVE-2021-44228
Log4J Library: Critical vulnerability in JNDI resource lookup feature

CRITICAL ALERT!!! 10/10

Systems / products affected:
Apache, Amazon EC2, Atlassian, BMC, Cisco, F-Secure, Fortinet, Okta RADIUS, Red Hat, Splunk, VMWare VCenter Server & more...

Example: Log4J – Feature or Vulnerability?

Log4j 2 / LOG4J2-313

JNDI Lookup plugin support

Details

Type:	New Feature	Status:	CLOSED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	2.0-beta9
Component/s:	None		
Labels:	None		

Description

Currently, Lookup plugins [1] don't support JNDI resources.
It would be really convenient to support JNDI resource lookup in the configuration.

One use case with JNDI lookup plugin is as follows:
I'd like to use RoutingAppender [2] to put all the logs from the same web application context in a log file (a log file per web application context).
And, I want to use JNDI resources look up to determine the target route (similarly to JNDI context selector of logback [3]).

All | Comments | Work Log | History | Activity | Transitions

Woonsan Ko added a comment - 17/Jul/13 20:56 - edited

Hi,

I've just attached a patch (jndi-lookup-plugin.patch) to provide built-in JNDI resource Lookup plugin with unit tests.
Please take a review.

Cheers,

Woonsan

Less than 24 hours!

Ralph Goers added a comment - 18/Jul/13 19:48

Your patch was committed in revision 1504620. Please verify and close.

Woonsan Ko added a comment - 22/Jul/13 02:40

Thank you so much, Ralph!
I appreciate for the site documentation as well.

Cheers,

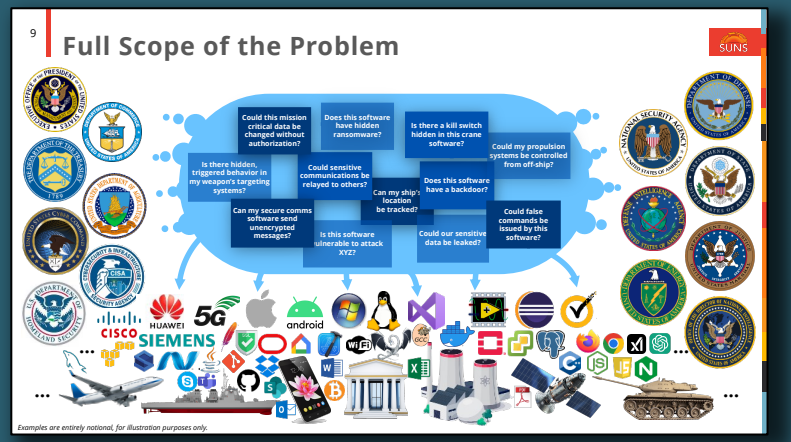
Woonsan

Source: https://issues.apache.org/jira/browse/LOG4J2-313

Example: Log4J – Feature or Vulnerability?

Jul 2013 — Dec 2021

This disagreement is addressed by empowering Mission Owners to routinely analyze any mission critical software to answer any mission question.



Log4j JND

Details

Type:

Priority:

Affects Vers

Component

Labels:

Description

Currently, Lookup plugins [1] don't support JNDI resources.

It would be really convenient to support JNDI resource lookup in the configuration.

One use case with JNDI lookup plugin is as follows:
I'd like to use RoutingAppender [2] to put all the logs from the same web application context in a log file (a log file per web application context).
And, I want to use JNDI resources look up to determine the target route (similarly to JNDI context selector of logback [3]).

Woonsan Ko added a comment - 22/Jul/13 02:40

Thank you so much, Ralph!
I appreciate for the site documentation as well.

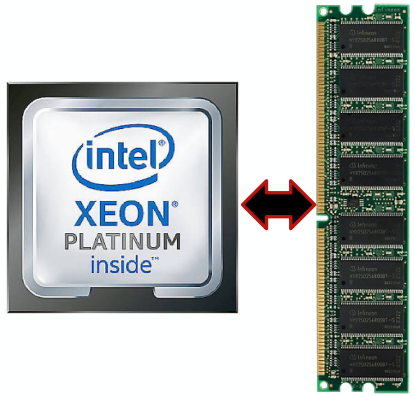
Cheers,

Woonsan

Source: https://issues.apache.org/jira/browse/LOG4J2-313

Where Do Vulnerabilities Come From?

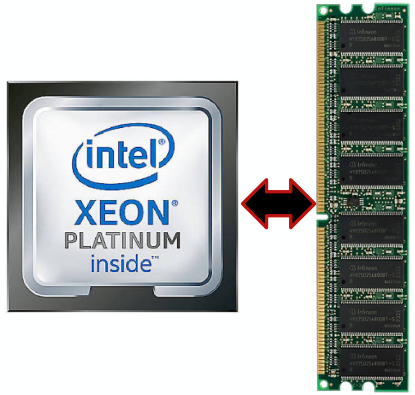
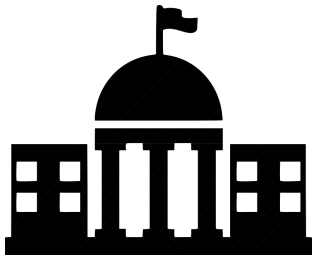
SUNS

Math	CPU & Memory	Operating Systems	Applications	Users
$\sum$				

Rule: My personal information must not be shared beyond the application.
Rule: My camera must not be one when the indicator light is not on.

Where Do Vulnerabilities Come From?

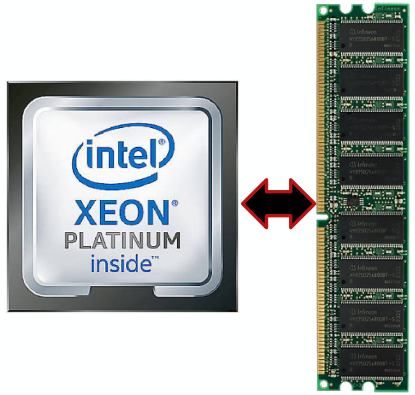
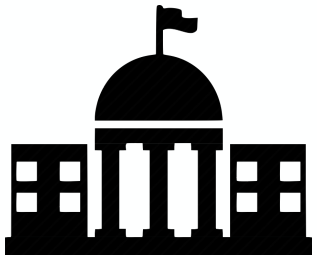
SUNS

Math	CPU & Memory	Operating Systems	Applications	Users	Organization
$\sum$					

Rule: The cameras must always stay on.
Rule: The cameras must never come on.

Where Do Vulnerabilities Come From?

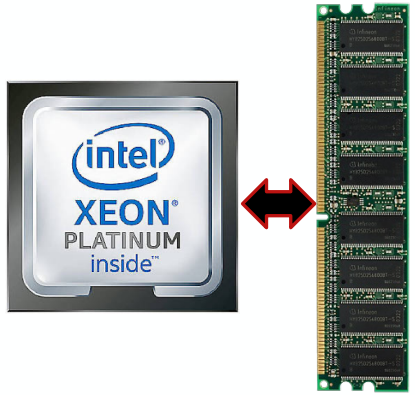
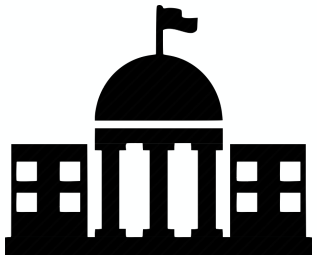
SUNS

Math	CPU & Memory	Operating Systems	Applications	Users	Organization
Σ					

Universal

Where Do Vulnerabilities Come From?

SUNS

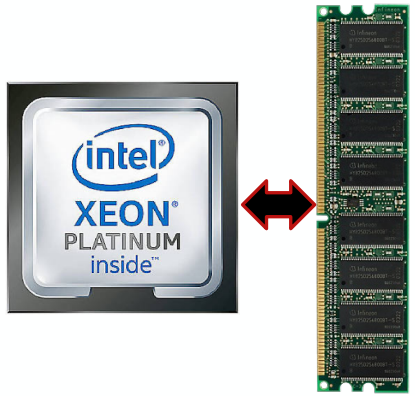
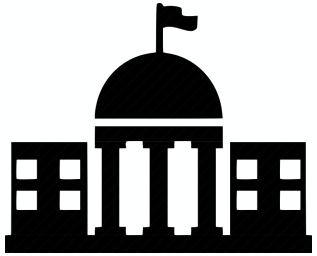
Math	CPU & Memory	Operating Systems	Applications	Users	Organization
Σ					

Universal

Bespoke

Where Do Vulnerabilities Come From?

SUNS

Math	CPU & Memory	Operating Systems	Applications	Users	Organization
Σ					

Universal

Mission Questions reveal bespoke defects that cannot be captured with abstract properties alone.

Bespoke

Where Do Vulnerabilities Come From?

SUNS

Observations

1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. The current approach results in massive duplication and waste and inhibits overall progress.
3. **Consumer confidence cannot follow strictly from producer guarantees, no matter how strong.**

Universal

You cannot make a tool that can check for all vulnerabilities.

Bespoke

Where Do Vulnerabilities Come From?

SUNS

Math | CPU & Memory | Operating Systems | Applications | Users | Organization

Observations

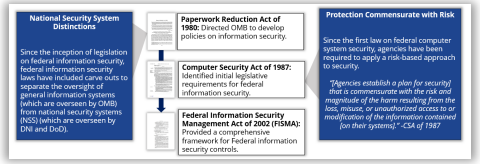
1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. The current approach results in massive duplication and waste and inhibits overall progress.
3. **Consumer confidence cannot follow strictly from producer guarantees, no matter how strong.**

Given these challenges, how should we think about risk?

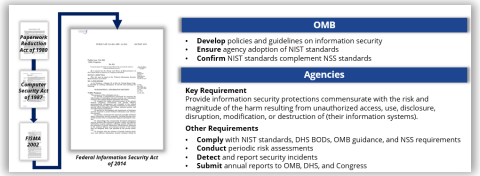
Gap: *Adequate Security Policy Execution*

USG information security policy is based on providing *adequate security* – this concept is integrated into legislation, executive actions, and regulations for federal and CI systems.

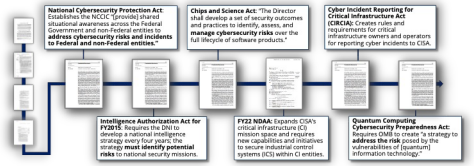
1 History of Information Security Legislation



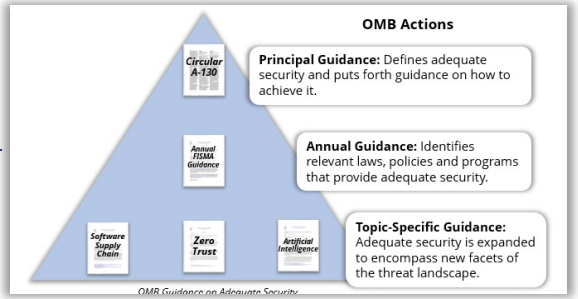
2 FISMA Overview



3 Policy Post-FISMA



4 White House Role



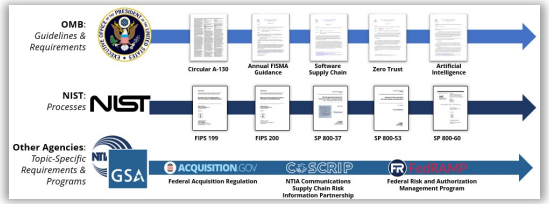
Executive Actions

Obama
- Promoting Private Sector Cybersecurity Information Sharing (EO 13691)
- United States Cyber Incident Coordination (PPD 41)

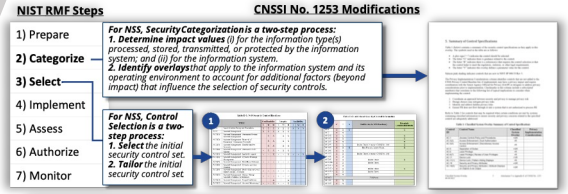
Trump
- Improving Critical Infrastructure Cybersecurity (EO 13636)
- Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure (EO 13800)

Biden
- Executive Order on Improving the Nation's Cybersecurity (EO 14028)
- Improving the Cybersecurity of National Security, DoD, and IC Systems (NSM-8)

5 FCEB Systems



6 National Security Systems



7 Critical Infrastructure

Energy **Finance** **Healthcare**

8 Adequate Security Policies

With software understanding, adequate security will be enhanced by enabling risk identification and mitigation, thus reducing residual risk to acceptable levels.

Inherent Risk	Inherent risk of software behavior can be far better assessed.
=	
Risk Mitigated by Security Controls	Mitigation controls can be targeted to identified software behavior and individual controls can be assessed for effectiveness.
=	
Residual Risk	The USG will understand more fully the residual risk before making risk acceptance decisions.

We lack capabilities today to meet "adequate" security

Gap: *Assessing Risk from Software*

Residual risk is the risk that remains after inherent risk has been partially mitigated.

Understanding inherent risk is the first step in the risk assessment process.

Gap: *Assessing Risk from Software*

SUNS

But how do we rigorously assess risk for software?

Inherent Risk

The risk to an entity in the absence of any direct or focused actions to alter its severity.

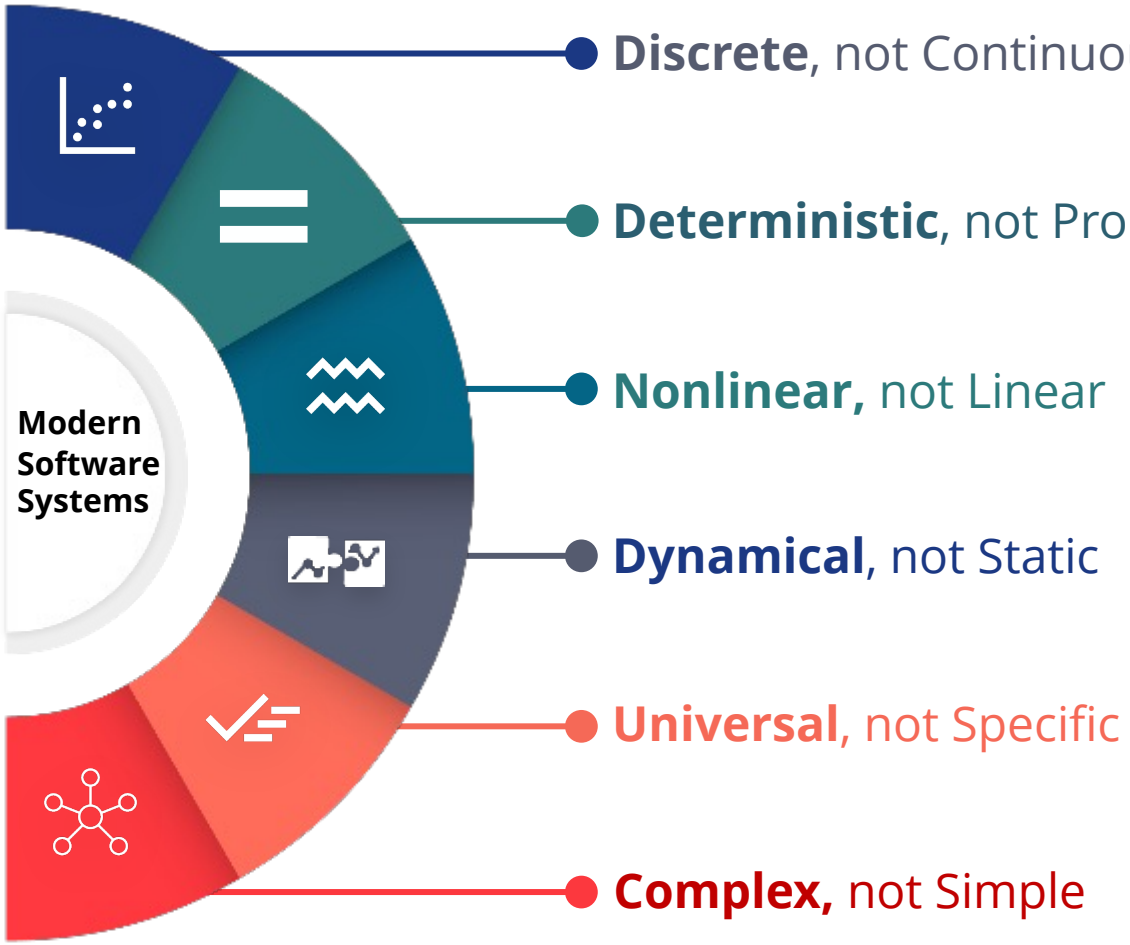
Aleatory Risk: associated with phenomena that are fundamentally random

<scenario, likelihood, consequence>

Statistics is the appropriate branch of math for aleatory risk.

Gap: *Assessing Risk from Software*

Modern software behavior is inscrutable because of its mathematical properties.



Modern software systems are not aleatory:

- Software systems are designed to be deterministic.
- Distributed systems are often called nondeterministic because controlling the input of time is difficult.
- But behavior is either present or not, based on inputs.

Gap: *Assessing Risk from Software*

SUNS

Assessing risk from software requires knowing whether behaviors are present.

Inherent Risk

The risk to an entity in the absence of any direct or focused actions to alter its severity.

Aleatory Risk: associated with phenomena that are fundamentally random

<scenario, likelihood, consequence>

Epistemic Risk: stems from gaps in knowledge, assumptions, or incomplete information

<scenario, ~~likelihood~~, consequence>

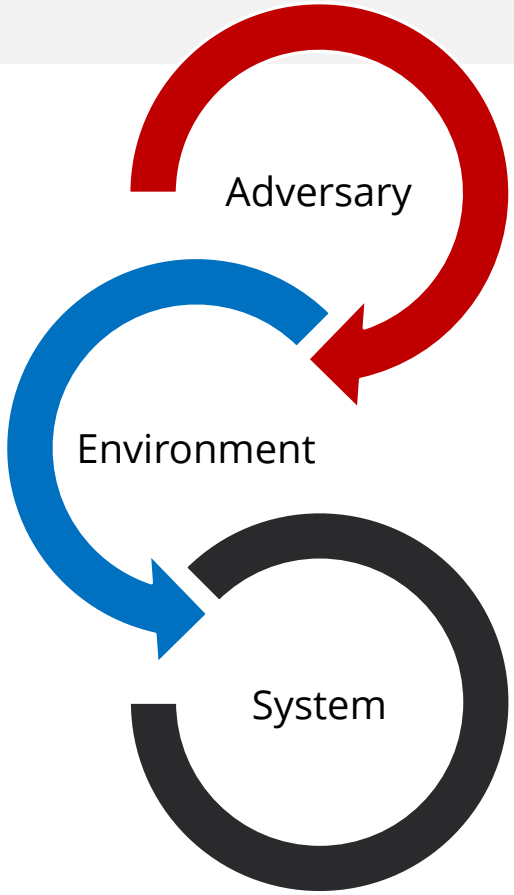
software behavior

Risks: *Epistemic Dominates Mission Risk*

> ***Epistemic risk* is driven by erroneous decisions due to *lack of knowledge* about the system being investigated.**

Overview

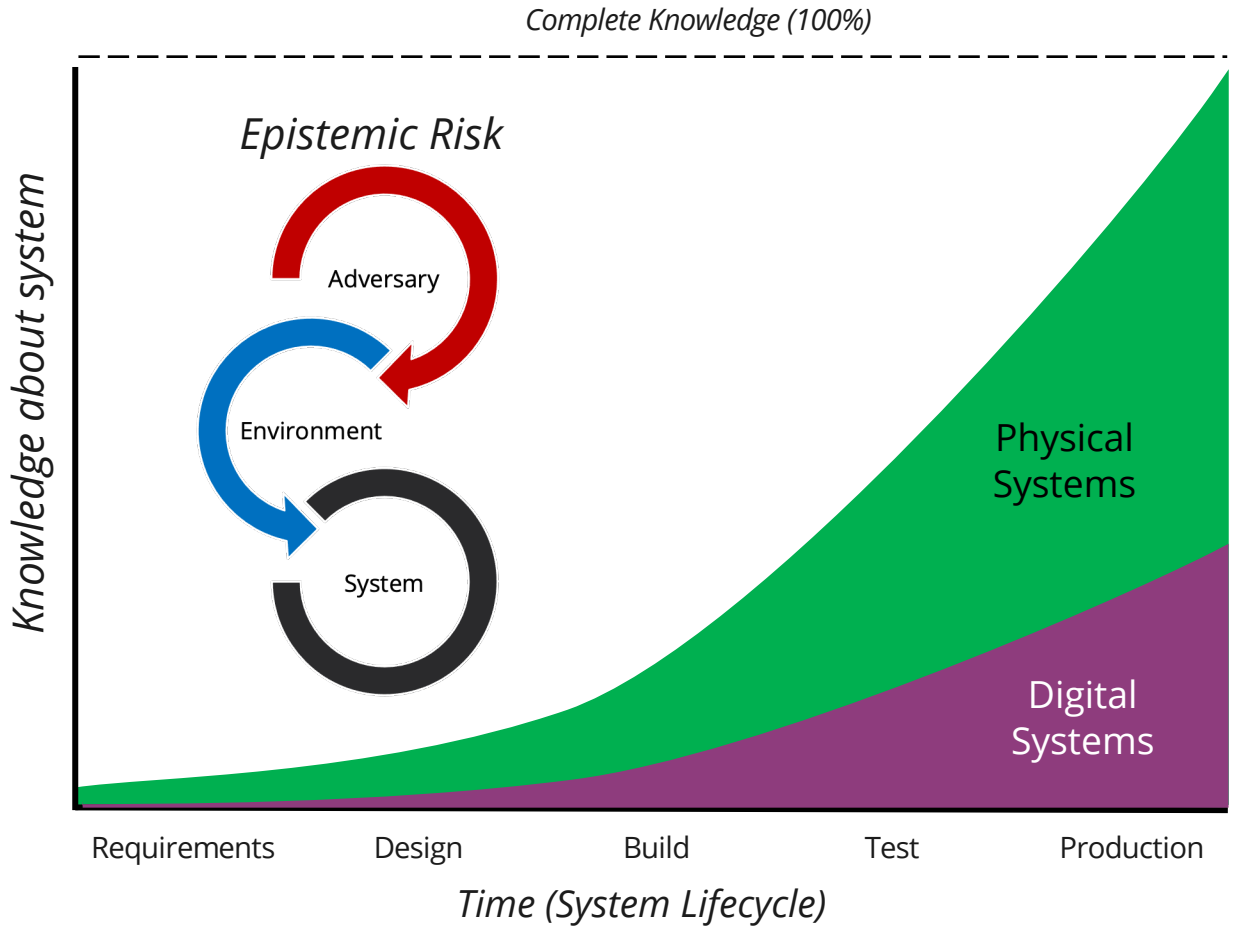
- Epistemic risk is driven by uncertainty about:
  - the **adversary,** their intent and capabilities.
  - the **environment**, the normal, non-normal, and hostile conditions that can occur.
  - the **system**, its full scope of behaviors, intended and unintended.
- This risk arises from knowledge gaps, flawed assumptions, cognitive biases, and poor methodologies/processes.
- The risks are dynamic and evolve throughout the system engineering lifecycle, from initial concept to disposition.



Epistemic Risk:
the core of the mission challenge

Risks: *Epistemic Lifecycle Knowledge*

When developing a system, knowledge of that specific system is gained throughout the lifecycle.



As the system development lifecycle progresses, knowledge is gained

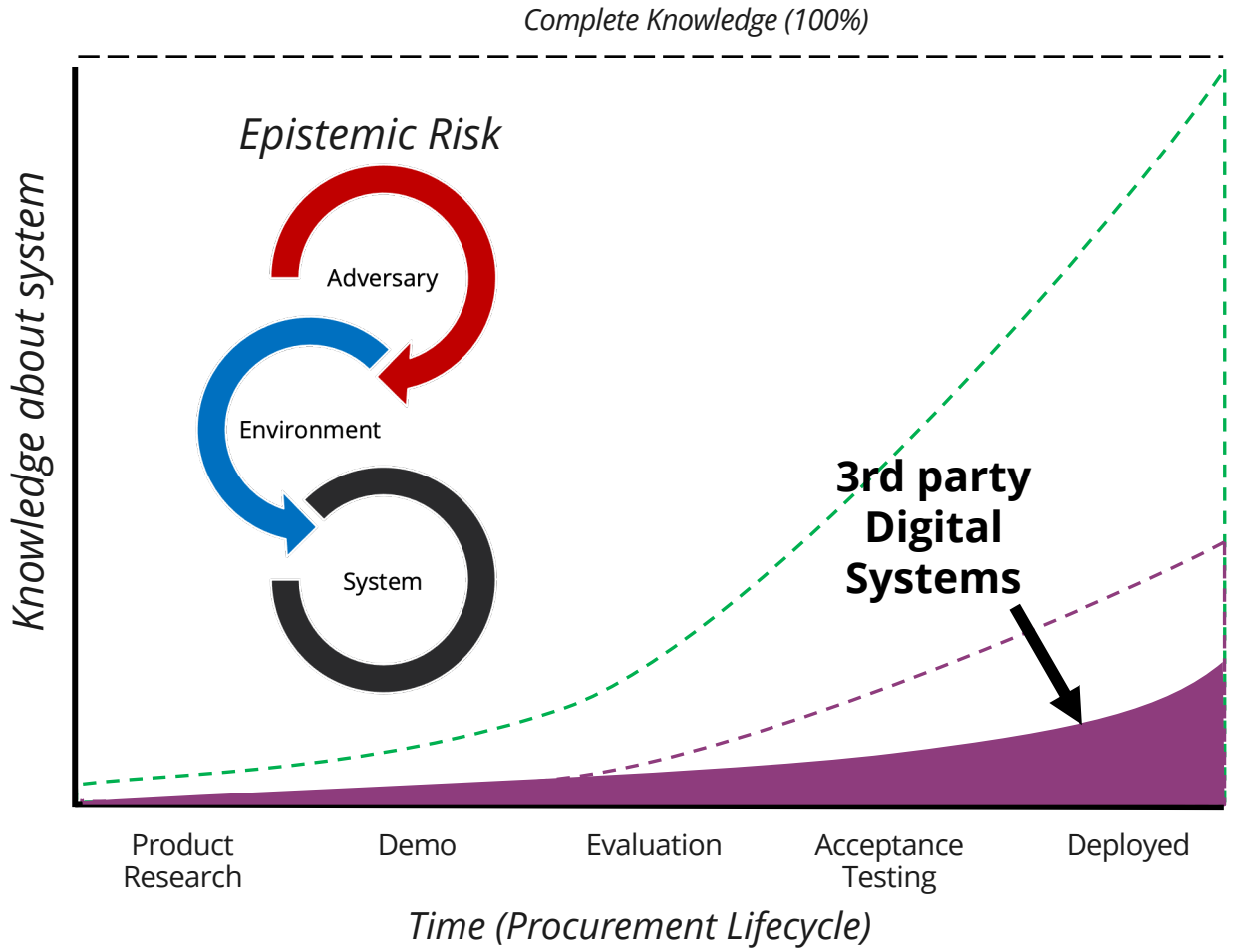
- The ability to address epistemic risk has greatly diminished as systems become digitized.
- We need new ways to buy down risk earlier in the lifecycle.
  - Understand software better, earlier
- Also benefits cost and schedule
- Minimizing risk requires new approaches to maximize understanding.

Forward Understanding

The practice of *constructing* of software-controlled systems to *verify* its behaviors.

Risks: *Epistemic Knowledge of Supply Chain Software*

Knowledge about possible behavior of third-party and legacy software must be acquired differently.



Systems that were not designed to facilitate analysis have even more epistemic risk

- The ability to analyze software depends on how it is designed and built
  - Most software is not designed to support *post hoc* analysis.
  - Includes 3rd-party and legacy systems
- Testing *cannot* provide high levels of assurance.
- We need new ways to gain understanding earlier in the lifecycle, before deploying software.

Reverse Understanding

The practice of *assessing* of software-controlled systems to *characterize* its behaviors.

Gap: *Assessing Risk from Software*

SUNS

Adequate security permits acceptable residual risk, but inadequate technical capabilities to understand mission risk from software undermines the risk acceptance paradigm.

Without software understanding, the process used to assess adequate security has significant technical gaps, leaving unbounded and unacceptable residual risk.

The USG cannot adequately assess the risk to mission posed by unknown software behavior.	**Inherent Risk**
	=
The USG cannot develop mitigations to address unknown risk or assess risk reduction.	**Risk Mitigated by Security Controls**
	=
The USG issues ATOs based on inadequate understanding of risk.	**Residual Risk**

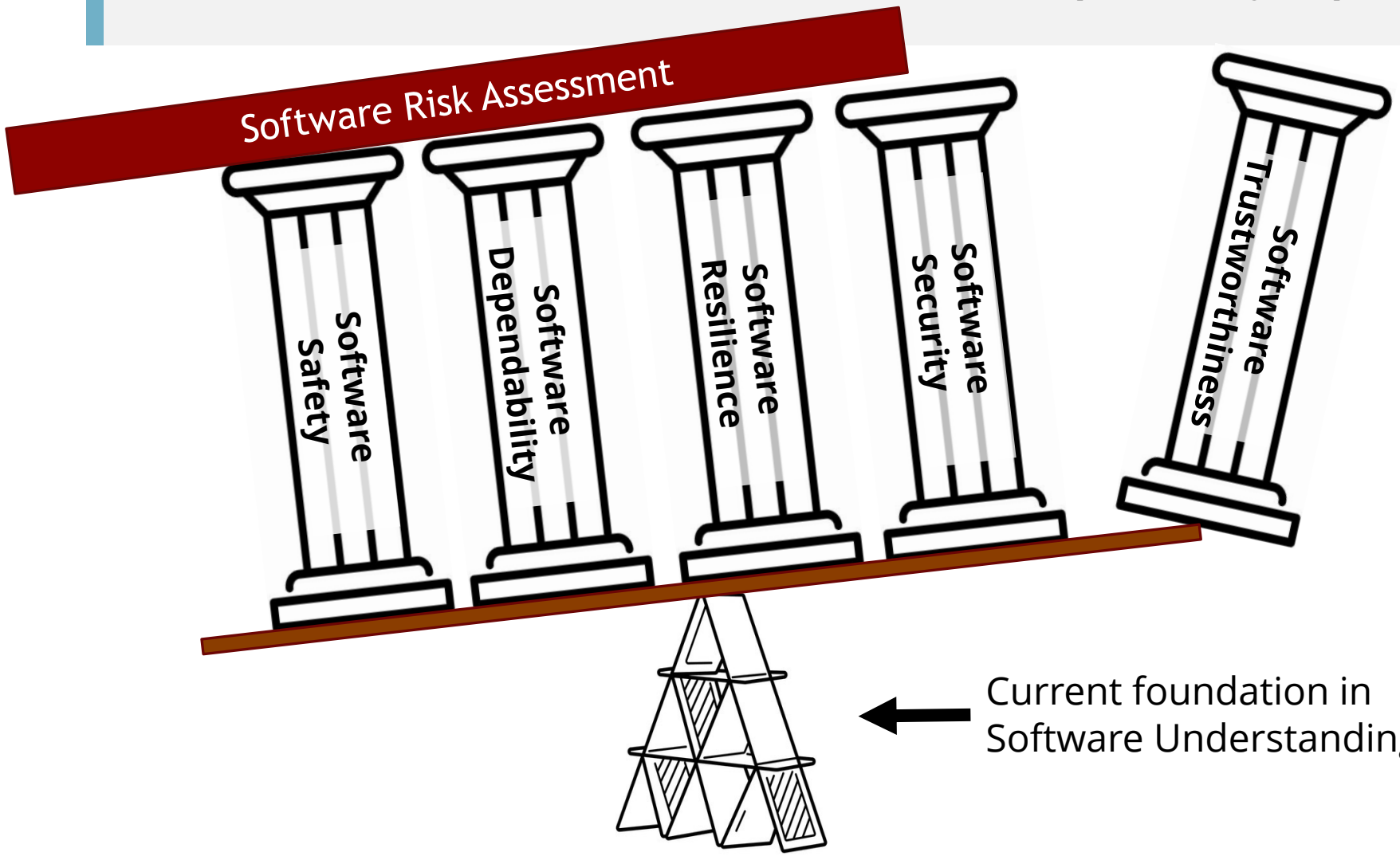
With software understanding, adequate security will be enhanced by enabling risk identification and mitigation, thus reducing residual risk to acceptable levels.

Inherent Risk	Inherent risk of software behavior can be far better assessed.
=	
Risk Mitigated by Security Controls	Mitigation controls can be targeted to identified software behavior and individual controls can be assessed for effectiveness.
=	
Residual Risk	The USG will understand more fully the residual risk before making risk acceptance decisions.

Without clear insight, inherent and residual risk—even potentially calamitous risk—is unknown.

Gap: *Missing Foundation for Reasoning About Software*

The need to understand software behavior underpins many important activities.



Malware detection
Ransomware prevention
Software forensics
Vulnerability research
Safety assessments
Software maintenance
Malicious indicator extraction
…

Gap: *Assessing Risk from Software*

SUNS

Adequate security permits acceptable residual risk, but inadequate technical capabilities to understand mission risk from software undermines the risk acceptance paradigm.

Observations

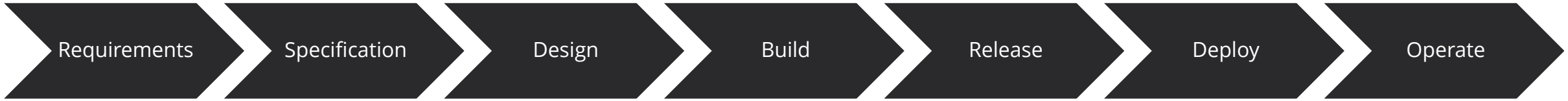
1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. The current approach results in massive duplication and waste and inhibits overall progress.
3. Consumer confidence cannot follow strictly from producer guarantees, no matter how strong.
4. **Mission owners have little choice but to operate critical missions blind to the risks.**

What kind of technical software understanding capabilities do we need?

Solution: Full Lifecycle Software Understanding

SUNS

An Idealized Software Lifecycle



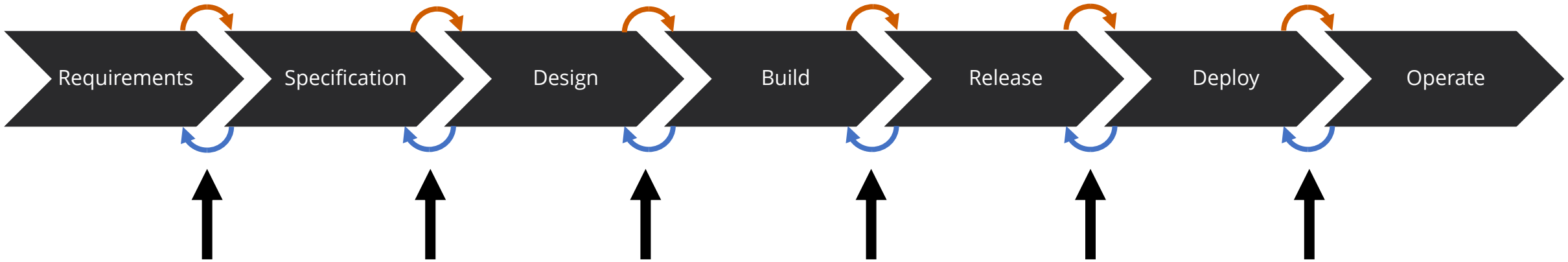
Forward Understanding

The practice of *constructing* of software-controlled systems to *verify* its behaviors.

Reverse Understanding

The practice of *assessing* of software-controlled systems to *characterize* its behaviors.

Solution: Full Lifecycle Software Understanding



- These are not 1:1 transformations.
- Abstract intentions are refined into increasingly concrete detail.
- Different implementation options have different characteristics.
- In many cases, components designed and developed to different standards are integrated.

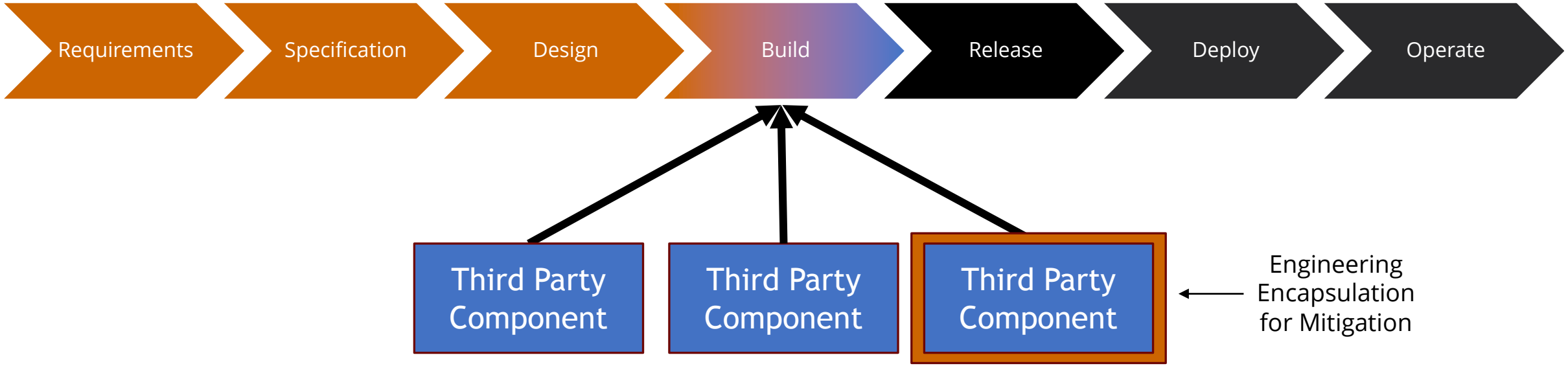
- Each step involves a forward understanding and a reverse understanding challenge.

Forward Understanding	Reverse Understanding
The practice of *constructing* of software-controlled systems to *verify* its behaviors.	The practice of *assessing* of software-controlled systems to *characterize* its behaviors.

Gap: *Full Software Lifecycle Perspective*

SUNS

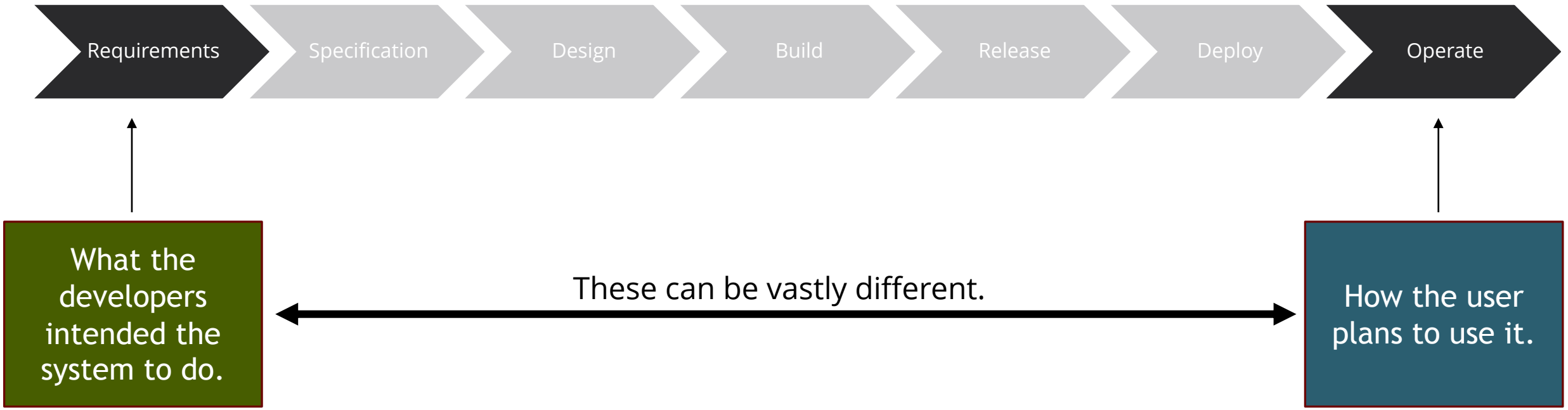
A More Realistic Software Lifecycle



- Mission owners need overall *system* guarantees that cannot be deduced strictly from components
- Component behavior must be characterized and selectively mitigated relative to mission needs
- This will typically require a mix of *forward* and *reverse* understanding

Forward Understanding	Reverse Understanding
The practice of *constructing* of software-controlled systems to *verify* its behaviors.	The practice of *assessing* of software-controlled systems to *characterize* its behaviors.

Gap: *Full Software Lifecycle Perspective*



- Software is vastly easier to analyze when it is designed to facilitate analysis.
  - Build artifacts can simply the reverse understanding process.
- Ideally, the developers would provide technical evidence of what they built the software to do…

  …and mission owners would pose and answer questions about what their mission needs it to do.
- This requires a close partnership between these historically disparate communities.

Software Understanding: *Decomposed*

SUNS

Observations

1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. The current approach results in massive duplication and waste and inhibits overall progress.
3. Consumer confidence cannot follow strictly from producer guarantees, no matter how strong.
4. Mission owners have little choice but to operate critical missions blind to the risks.
5. **Neither the forward nor reverse understanding communities can solve the problem alone.**

- So
  - Build artifacts can simply the reverse understanding process.
- Ideally, the developers would provide technical evidence of what they built the software to do…

  …and mission owners would pose and answer questions about what their mission needs it to do.
- This requires a close partnership between these historically disparate communities.

Software Understanding: *Decomposed*

SUNS

Observations

1. Capabilities to routinely answer mission critical questions about software do not exist today.
2. The current approach results in massive duplication and waste and inhibits overall progress.
3. Consumer confidence cannot follow strictly from producer guarantees, no matter how strong.
4. Mission owners have little choice but to operate critical missions blind to the risks.
5. **Neither the forward nor reverse understanding communities can solve the problem alone.**

What kind of organizational effort is needed to address the software understanding gap?

SUNS History: *Overview*

The USG has been wrestling with software understanding challenges for decades. Recently, efforts have focused on defining challenges, needs, and opportunities.

SUNS Workshop
(March 2023)

SUNS Technical Exchange Meeting
(March 2024)

SUNSEC Founders Meeting
(July 2024)

SUNS RD&E Roadmap
(December 2024)

Presents a technical research, development, and engineering roadmap to enable the U.S. government to achieve greater software understanding.

Closing the Software Understanding Gap
(January 2025)

Defines a call to action for the U.S. government to take decisive and coordinated action to close the software understanding gap.

Software Understanding for National Security – Partnership Forum
(March 2025)

The forum served as the "launch event" for the "Closing the Software Understanding Gap" whitepaper.

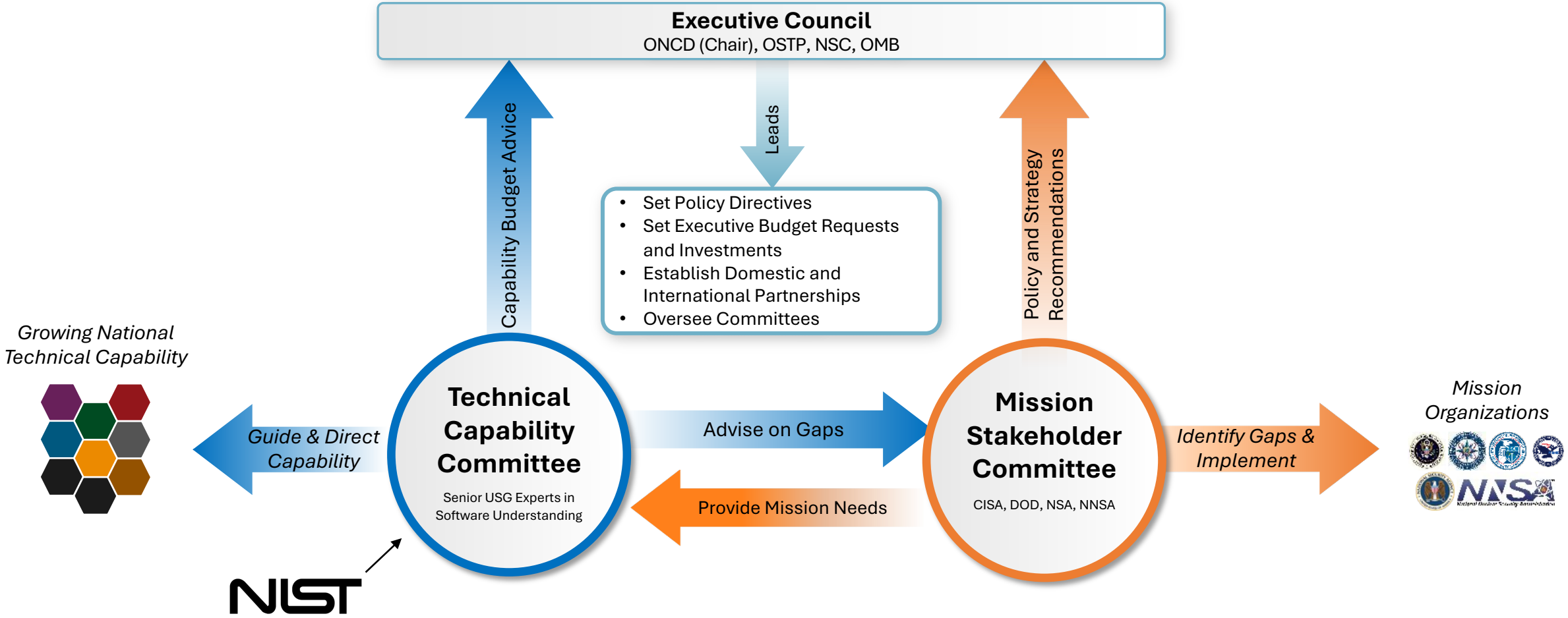
The National Need for Software Understanding
(March 2025)

Outlines the challenges of software understanding for NS&CI missions, discusses the shortcomings of traditional investment approaches, documents the outcomes of the SUNS 2023 Workshop and concludes with recommendations.

These documents are available at https://suns.sandia.gov/

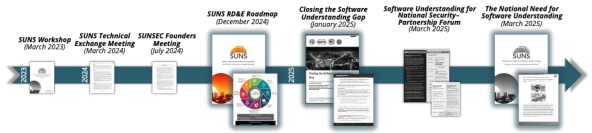
SUNS Executive Council

A notional draft of an organizational construct to achieve the necessary outcomes.



DRAFT SUNSEC Organizational Chart

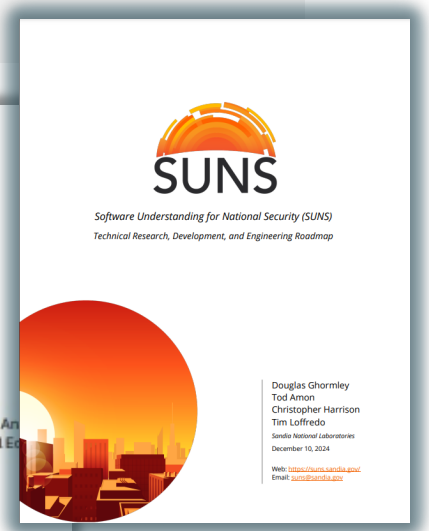
Technical RD&E Roadmap: *Overview*



The RD&E roadmap outlines technical exploration options toward achieve a greater reverse understanding of software within NS&CI mission spaces.

Areas of Research in the Roadmap

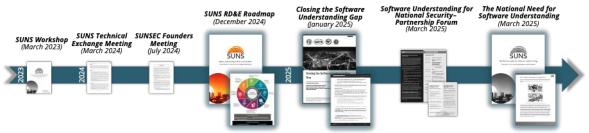
1. Formal Foundations for Software Reasoning
2. Analysis Architectures and Automated Tool Synthesis
3. Software Execution Modeling
4. Model Generation Techniques
5. Analysis Tool Ecosystem
6. Semantic Knowledge Interference
7. Hierarchical Question Decomposition and Evidence Composition
8. Datasets, Benchmarks, and Ground Truth



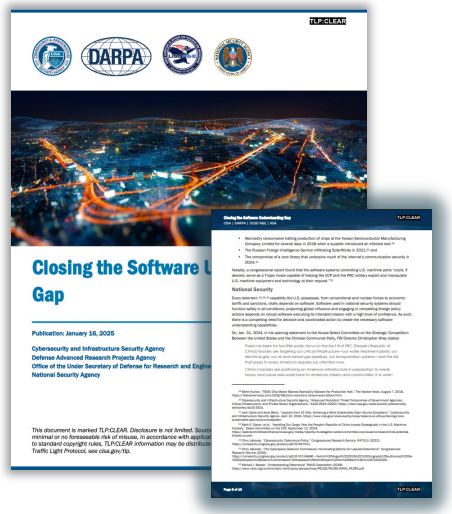
- 8 Major Research Challenges
- 29 Technical Research Sub-Thrusts
- >100 specific RD&E needs

This document is available at https://suns.sandia.gov/

Closing the Software Understanding Gap



This report is a call to action for the US Government to take decisive and coordinated action to close the software understanding gap.

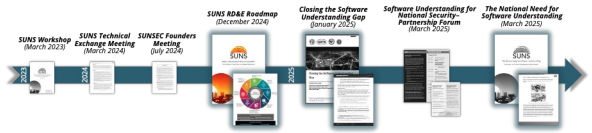


Closing the Software Understanding Gap

Call to Action

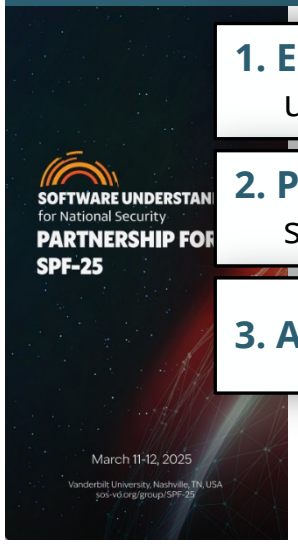
1. **Policy Action:** Reconsider policy to accelerate the development and adoption of software understanding capabilities and cultivate software understanding as a critical national resource.
2. **Technology Procurement:** Reimagine acquisition of software to drive risk lower by empowering the U.S. government to foster and incentivize the widespread adoption of ever-advancing capabilities.
3. **Technical Solutions:** Establish coordinated foundational and applied R&D efforts to invest in common solutions that advance national capabilities more broadly and cost-effectively.

2025 SUNS Partnership Forum (SPF)



The SUNS Partnership Forum 2025 (SPF-25) served as a launch event for the "Closing the Software Understanding Gap" report.

SPF-25 Goals



1. **Engagement:** Engage academia, industry, and government on the software understanding problem.
2. **Perspective:** Gather perspectives on the problem, challenges, and potential solutions.
3. **Action:** Identify actions that the SUNS partners can each take.

The event brought together the communities below to foster engagement, explore solutions, and promote collaboration in closing the software understanding gap.

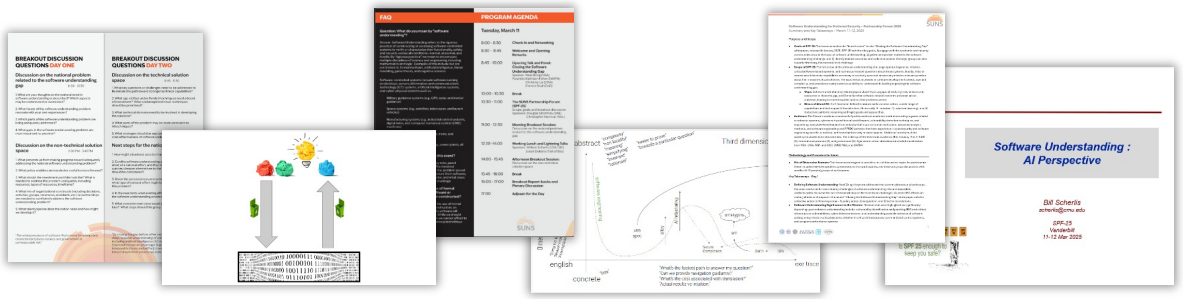
VANDERBILT UNIVERSITY · DARPA · LOCKHEED MARTIN · galois · aws · Collins Aerospace · Qualcomm · Atalanta Therapeutics · ASU · Sandia National Laboratories · INL Idaho National Laboratory · leidos · twosix TECHNOLOGIES · Microsoft · SRI International · National Cyber Security Centre · NSF

2025 SPF: *Outcomes and Key Takeaways*

SPF outlined the significance of the software understanding challenge from academic, industry, and government viewpoint – while highlighting the needed next steps.

Key Takeaways

- **Agreement Across Academia and Industry:** Broad agreement on the national-level challenge and scope.
- **Software Understanding to Drive Solutions:** Broad agreement on Software Understanding as a powerful concept in elucidating the opportunity cost of the current approach and the commonality that could drive solutions.
- **Lack of National Level Efforts:** There was no alternative identified to a national level effort in software understanding.
- **Government Has A Key Role in Discussions:** The absence of the government during discussions was notably impactful, particularly in certain policy areas, such as acquisition.



Proposed Next Steps

1. Engagement with DOD (OUSD R&E, A&S, DARPA) – in particular, multiple participants favored a new DARPA program focused on Software Understanding.
2. Producing and providing a Software Understanding technical packet to the Congressional Research Service.
3. Engagement with NITRD, the National Academies, CAE Symposium, HCSS, and other venues.
4. Engagement with the administration (ONCD, OSTP, OMB, NSC, etc.).

SUNS History: *Overview*

The USG has been wrestling with software understanding challenges for decades. Recently, efforts have focused on defining challenges, needs, and opportunities.

SUNS Workshop
(March 2023)

SUNS Technical Exchange Meeting
(March 2024)

SUNSEC Founders Meeting
(July 2024)

SUNS RD&E Roadmap
(December 2024)

Presents a technical research, development, and engineering roadmap to enable the U.S. government to achieve greater software understanding.

Closing the Software Understanding Gap
(January 2025)

Defines a call to action for the U.S. government to take decisive and coordinated action to close the software understanding gap.

Software Understanding for National Security – Partnership Forum
(March 2025)

The forum served as the "launch event" for the "Closing the Software Understanding Gap" whitepaper.

The National Need for Software Understanding
(March 2025)

Outlines the challenges of software understanding for NS&CI missions, discusses the shortcomings of traditional investment approaches, documents the outcomes of the SUNS 2023 Workshop and concludes with recommendations.

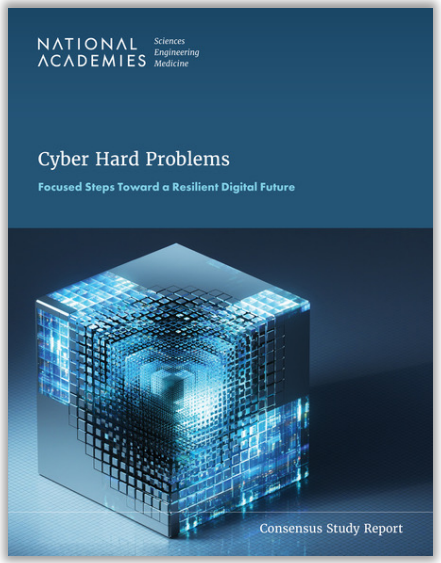
These documents are available at https://suns.sandia.gov/

National Academies: *Cyber Hard Problems (CHP)*

Most of the Cyber Hard Problems cannot be solved independently of closing the Software Understanding Gap.

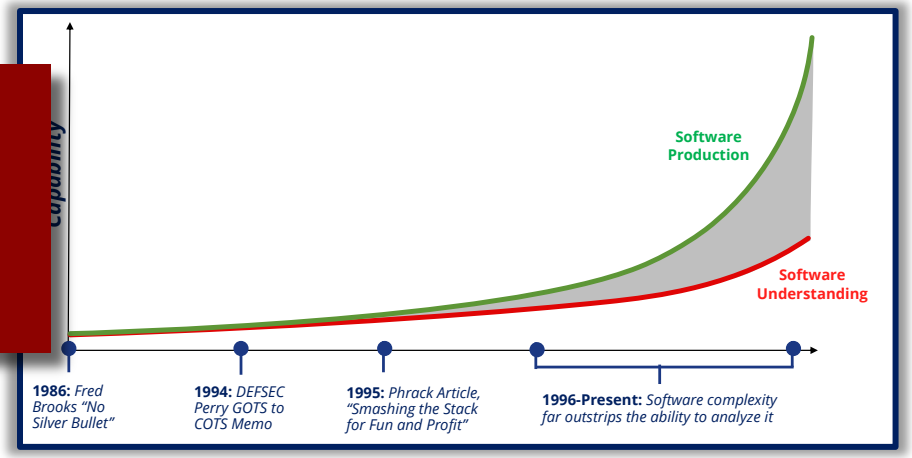
Cyber Hard Problems

1. Risk Assessment and Trust
2. Secure Development
3. System Composition
4. Supply Chain
5. Economic Factors
6. Human-System Interaction
7. Information Provenance
8. Cyber-Physical Systems
9. Artificial Intelligence
10. Operational Security



National Academies Cyber Hard Problems Consensus Study Report 2025

- Many CHPs are intimately related to understanding possible software behavior
- Progress will be severely limited until the software understanding gap is closed



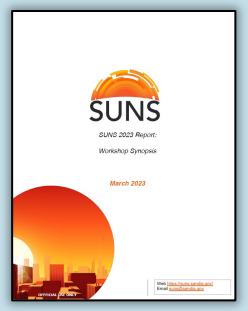
An Invitation

The 2023 SUNS workshop, hosted by DHS S&T, validated the need for and the viability of creating national capabilities to close the software understanding gap.

QUESTIONS PRESENTED

1. What are technical impediments to a national software understanding capability?
2. How could the USG support the expansion of software understanding?
3. What are near-term R&D funding priorities?

Top 5 Gaps:



1. Unified Vision
2. Strong Community
3. Ability to Collaborate
4. Sufficient & Sustained Funding
5. **Robust Metrics and Benchmarks**

Sandia is working on a suite of microbenchmarks as one initial step in being able to measure progress as capabilities improve. We would welcome NIST engagement!

If you're interested in engaging, please email suns@sandia.gov


SUNS