V&V for Autonomous Systems in a Lifecycle Approach to Securing Trust

Dr. Paul Nielsen
Director and CEO

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Why increase autonomy?

Speed

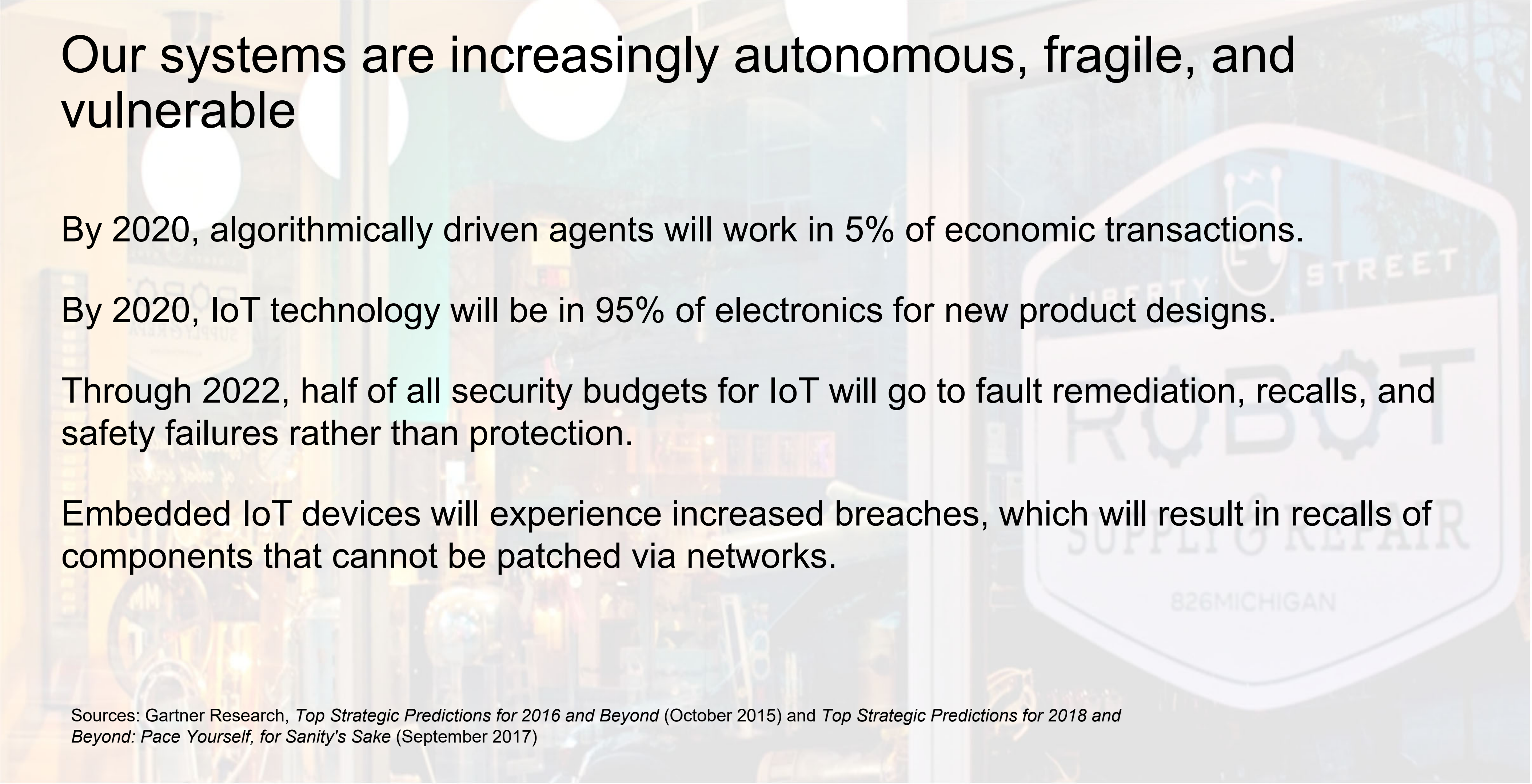
Volume

Danger

Persistence

Communication

Our systems are increasingly autonomous, fragile, and vulnerable



By 2020, algorithmically driven agents will work in 5% of economic transactions.

By 2020, IoT technology will be in 95% of electronics for new product designs.

Through 2022, half of all security budgets for IoT will go to fault remediation, recalls, and safety failures rather than protection.

Embedded IoT devices will experience increased breaches, which will result in recalls of components that cannot be patched via networks.

Sources: Gartner Research, *Top Strategic Predictions for 2016 and Beyond* (October 2015) and *Top Strategic Predictions for 2018 and Beyond: Pace Yourself, for Sanity's Sake* (September 2017)

Autonomous systems are the result of decades of R&D

R&D areas include

- Digitization of sensors
- Adaptive algorithms
- Natural user interfaces
- Machine learning
- Machine vision

. . . and improved software practices

Virtual integration (integrate-then-build)

- Relies on architectural model repository
- Reduces risk, cost, and development time

DevOps

Continuous delivery

Architecture-model-based engineering

Auto code generation

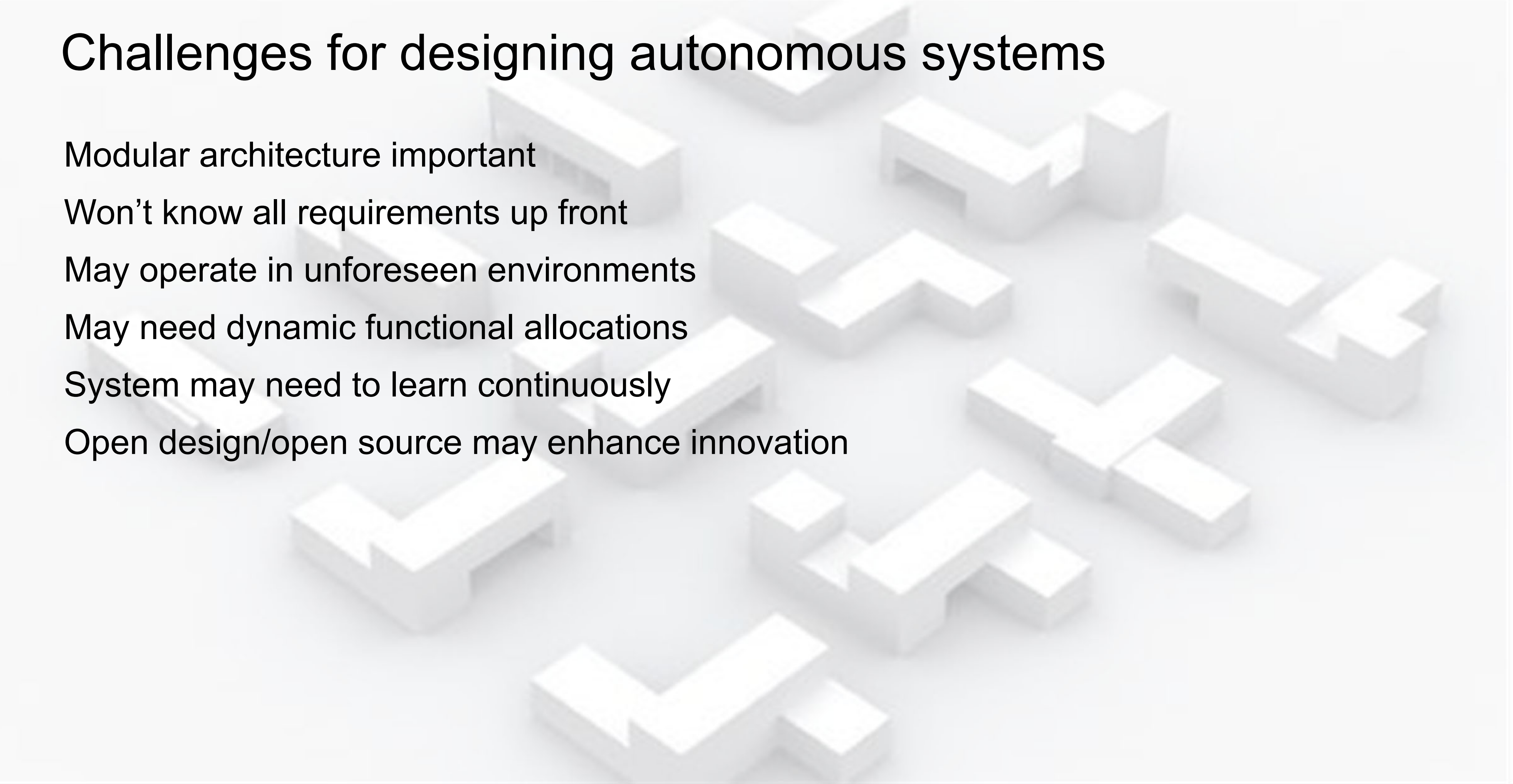
. . . as well as the convergence of software capabilities

2007: DARPA Urban Challenge

> “This car is the holy grail of autonomous driving.”
>
> Prof. Raj Rajkumar, co-director, CMU-General Motors
> Autonomous Driving Collaborative Research Lab

2014: Autonomous Cadillac SRX

Challenges for designing autonomous systems



Modular architecture important

Won’t know all requirements up front

May operate in unforeseen environments

May need dynamic functional allocations

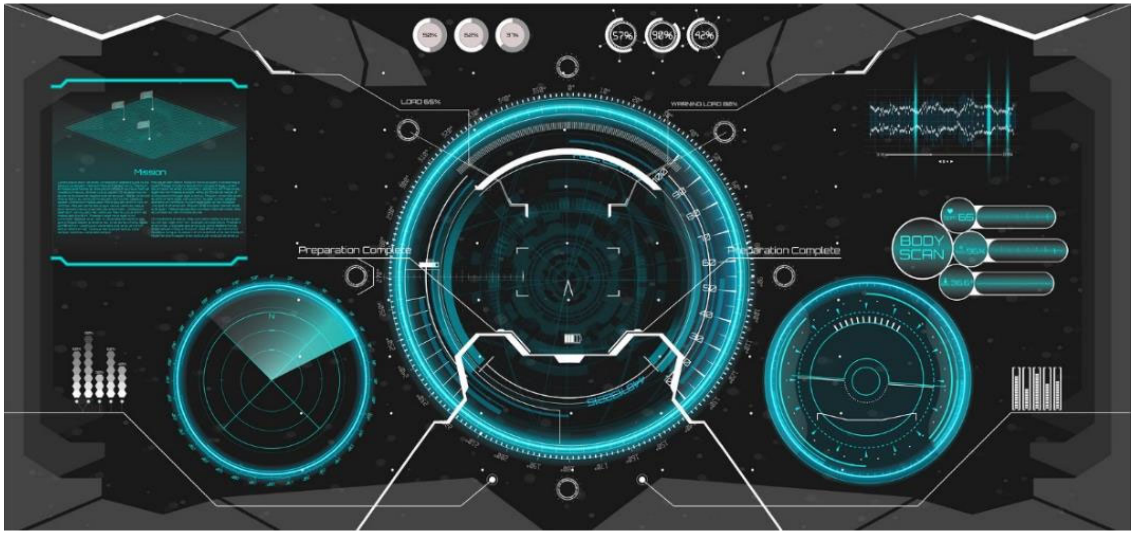
System may need to learn continuously

Open design/open source may enhance innovation

Challenges for building autonomous systems

- Complexity
- Connectedness
- Functional allocations
- Trust

Impact of complexity



Emergent behavior

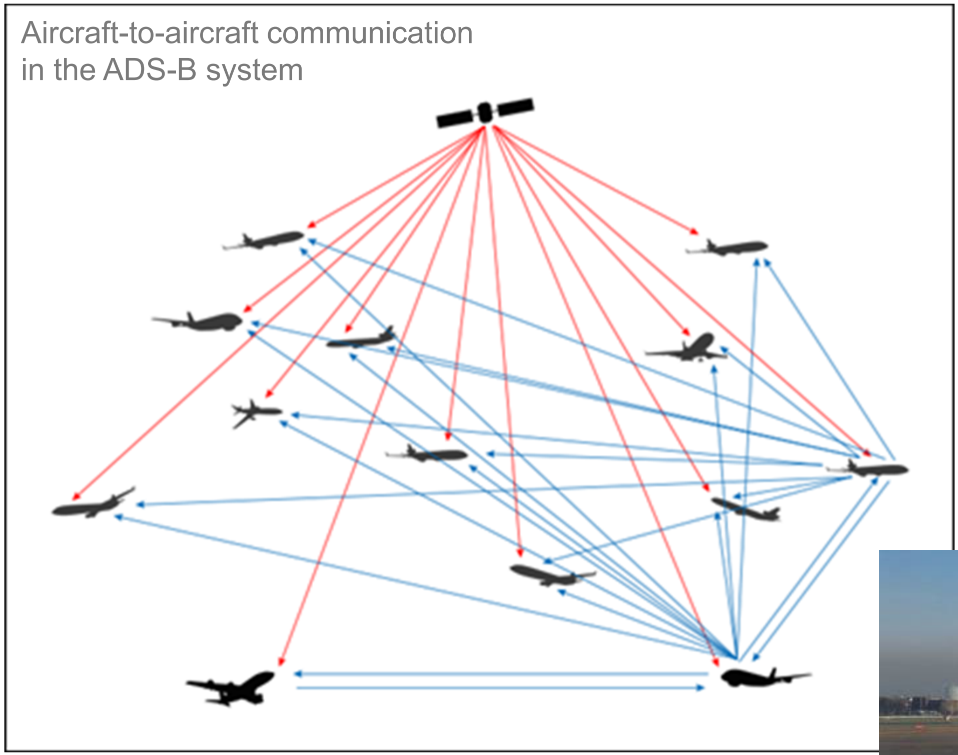
Continuous and asynchronous delivery

System will continuously change

System boundary may be hard to define

Human/machine interface issues

Impact of connectedness



System boundary ever-changing

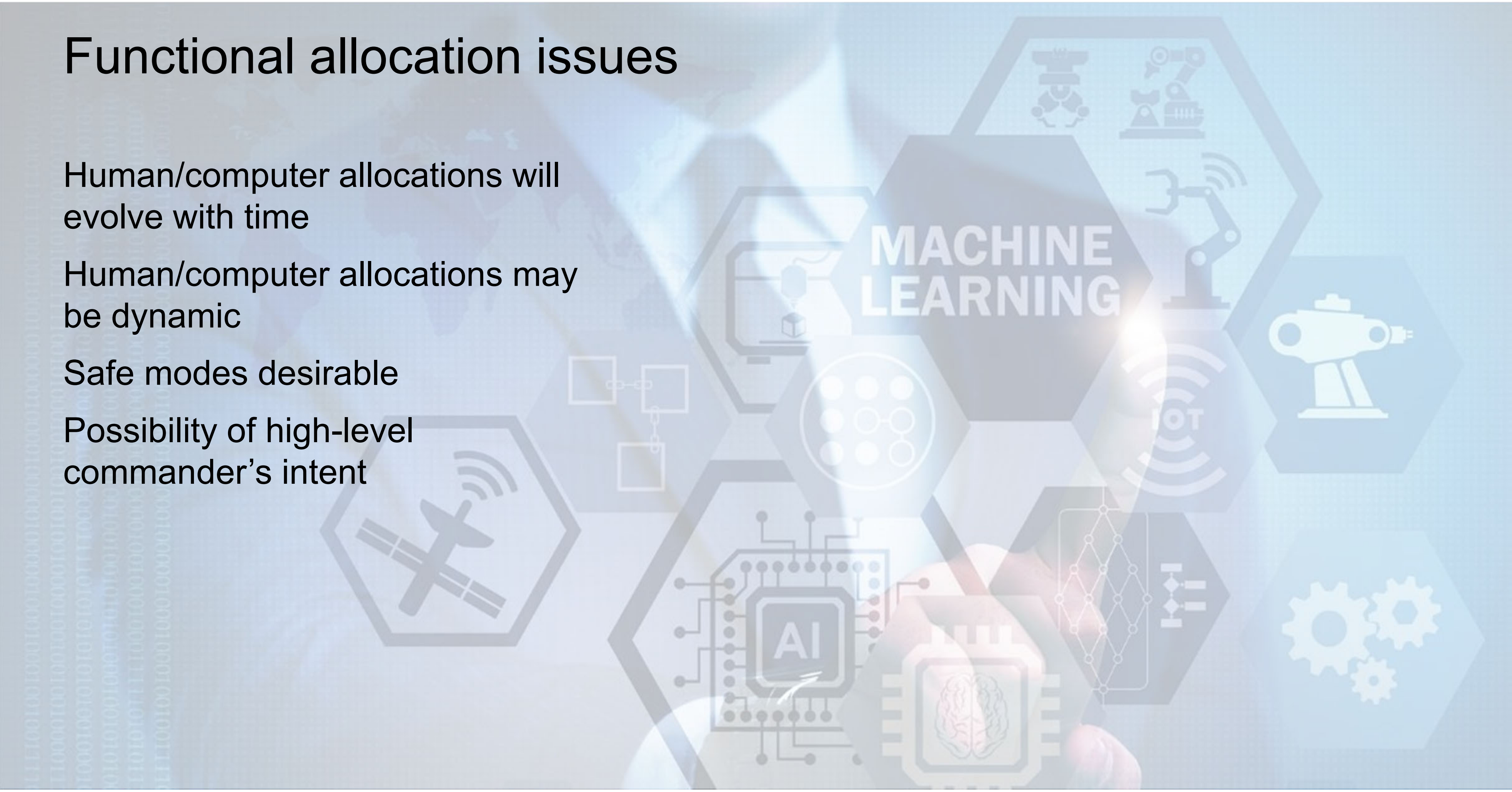
New interfaces the norm rather than exception

Large attack surface for vulnerabilities

Coupling issues

Information overload and interface to human team members

Functional allocation issues



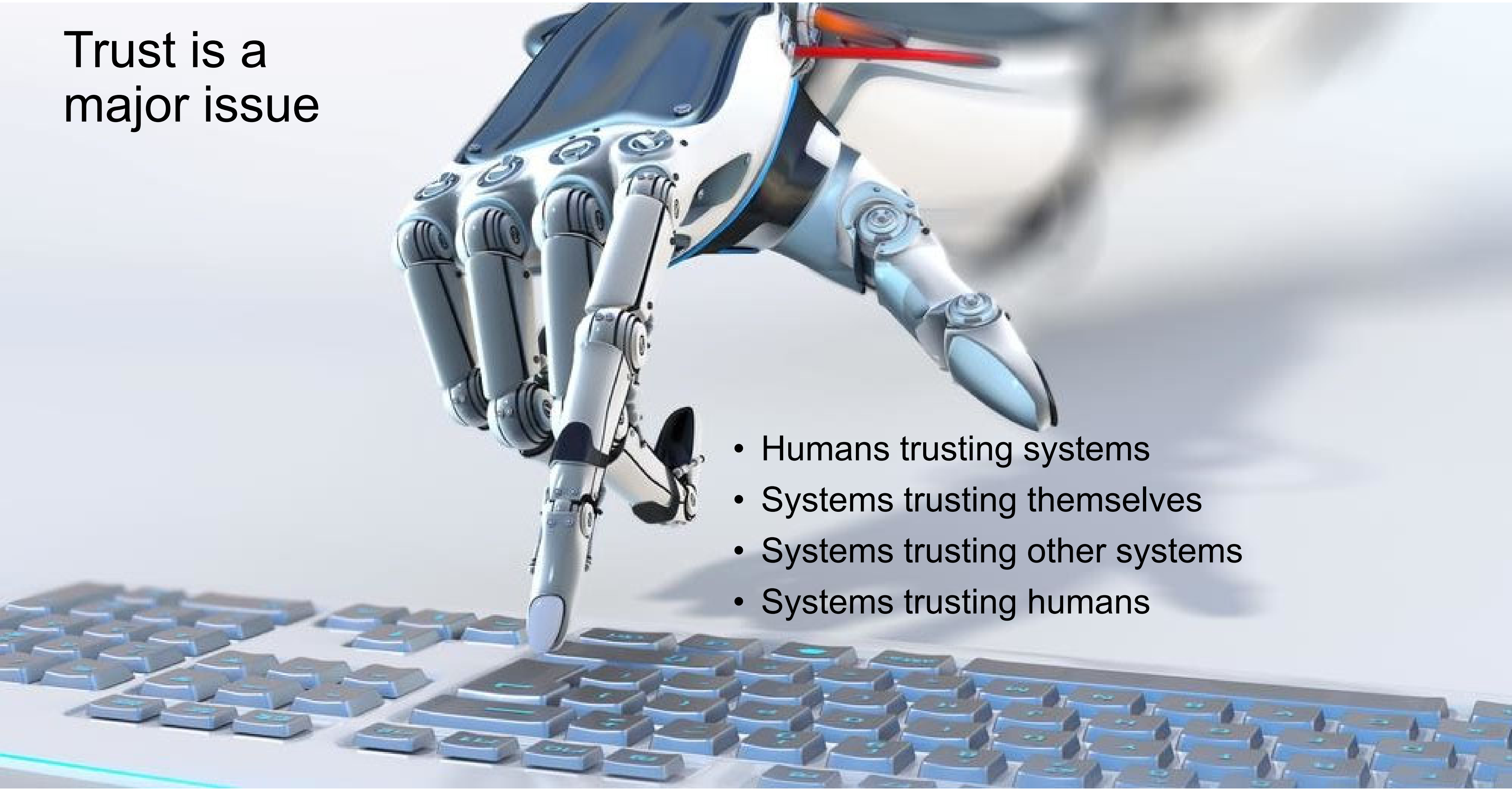
Human/computer allocations will evolve with time

Human/computer allocations may be dynamic

Safe modes desirable

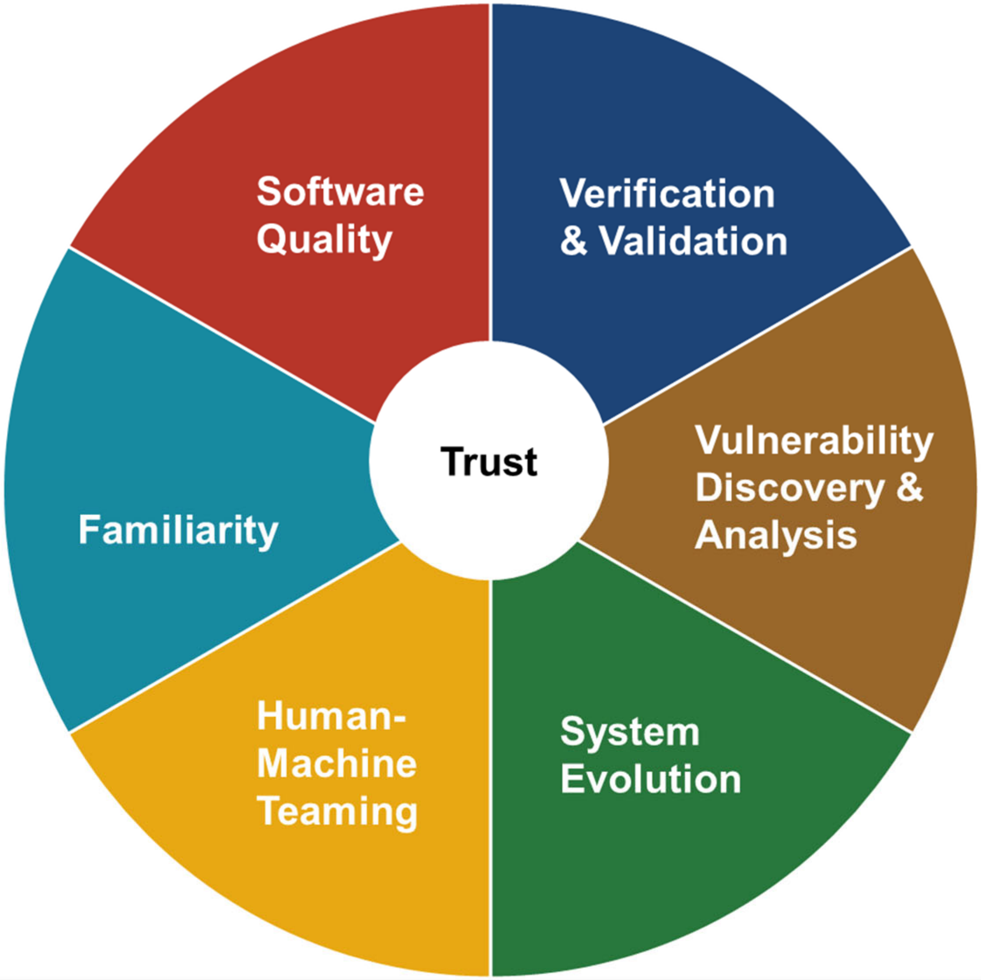
Possibility of high-level commander’s intent

Trust is a major issue



- Humans trusting systems
- Systems trusting themselves
- Systems trusting other systems
- Systems trusting humans

Trust in autonomous systems requires a lifecycle approach



Essential: Reliable datasets

Emphasize data provenance and quality

“Instrument” business and mission processes to produce effective data for ML applications

Create a mechanism to cultivate, label, and *share* data

Protect the data, but not at the expense of maximal sharing to properly vetted researchers and implementers

The more contributors, users, and labelers of data, the better the whole ecosystem will be

Blend development and operational testing

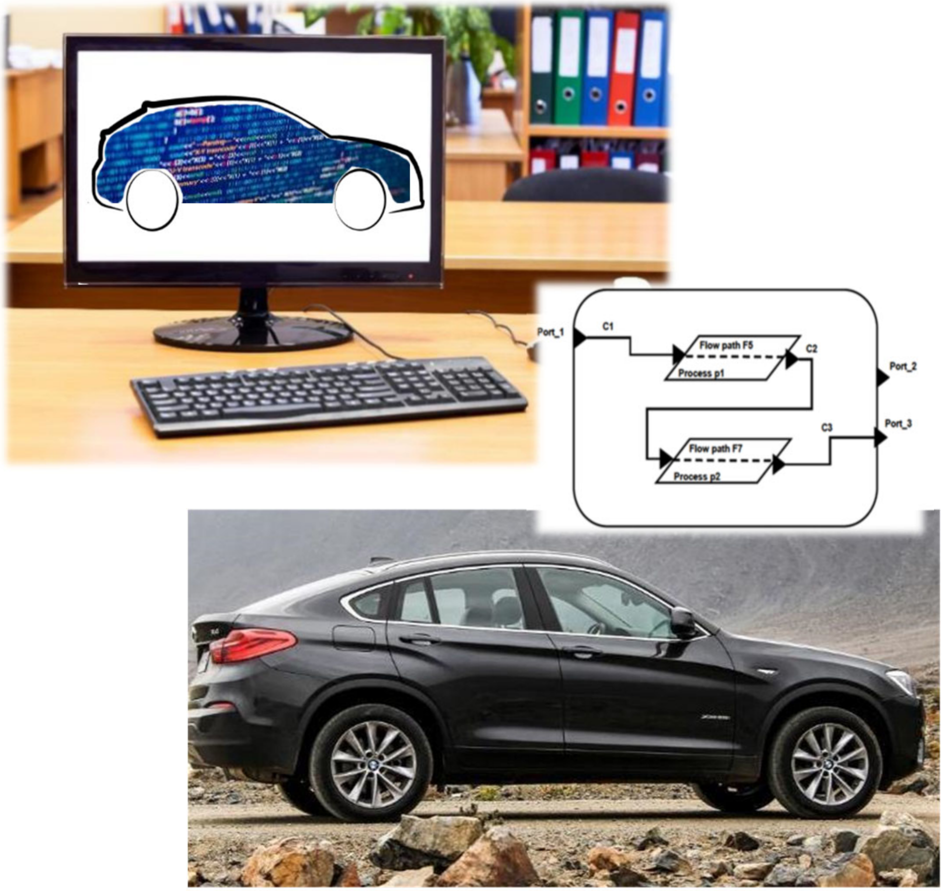
Emphasize communication, collaboration, and integration between developers and operators

Add operations personnel in the development team to transfer knowledge of deployment and maintenance

Manage multiple dimensions

- Culture
- Automation and Measurement
- Process and Practices
- System and Architecture

Adopt M&S in overall T&E program



Transition T&E

- develop and use predictive models of system behavior (e.g., trustworthiness)
- orient toward mission goals

Take a comprehensive approach to M&S in

- V&V
- system design and architecture development

Continue to collect data past deployment

Use the MIT/Lincoln Lab sensor sidecar approach

Adjunct processors that support development and demonstration of advanced software functions

Access a sensor's data in real time while not interfering with the operation of previously verified sensor processors and software

Use formal methods when practical

Runtime assurance

- deploys enforcers to check system during runtime against acceptable limits
- makes correction

Limitations

- often rely on unverified enforcers
- require system-wide re-verification when an enforcer is changed, added, or removed

Recognize importance of cybersecurity

Increased autonomy may help cybersecurity

- Volume, speed, persistence

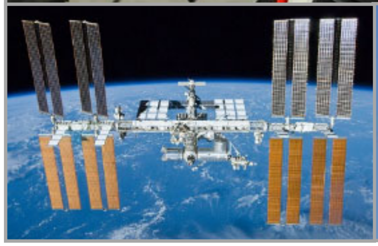
But autonomous systems themselves will be vulnerable

- Normal software and system vulnerabilities
- Mis-training
- Spoofing
- Hidden modes

Vulnerabilities in autonomous control of cyber-physical systems can have more dire consequences

- Need continuous red-teaming

Plan for Software Maintenance and Evolution

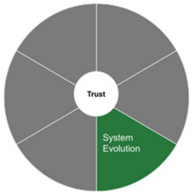


Challenges include

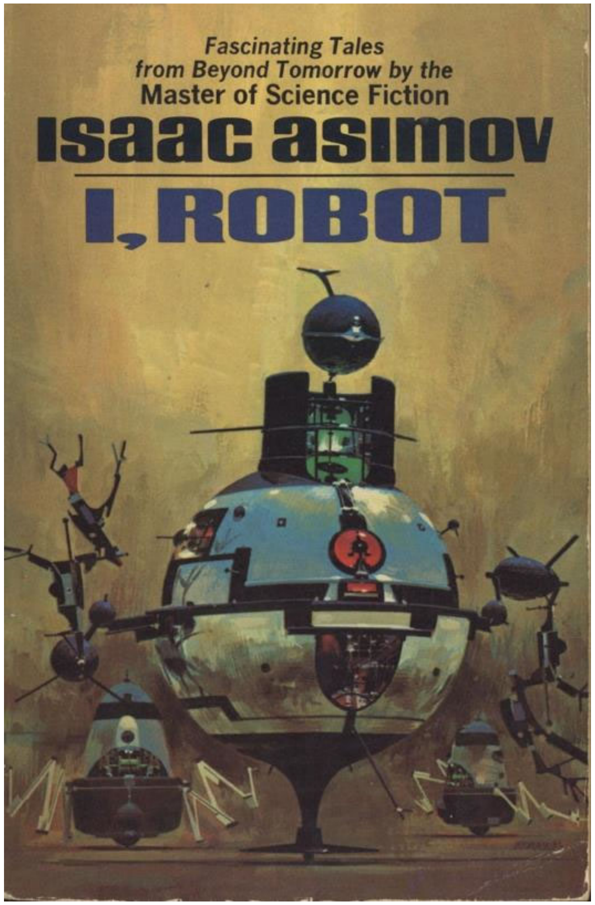
- rising costs
- dynamic operating environments
- legacy environments
- recertification

No break point where software is handed off for sustainment

Involves coordinating processes, procedures, people, and information



Need: Define context for human-machine teaming



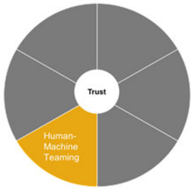
In the real world, autonomy is usually granted within some context—explicit or implicit

- Parents and children
- Soldiers, sailors, marines, and airmen

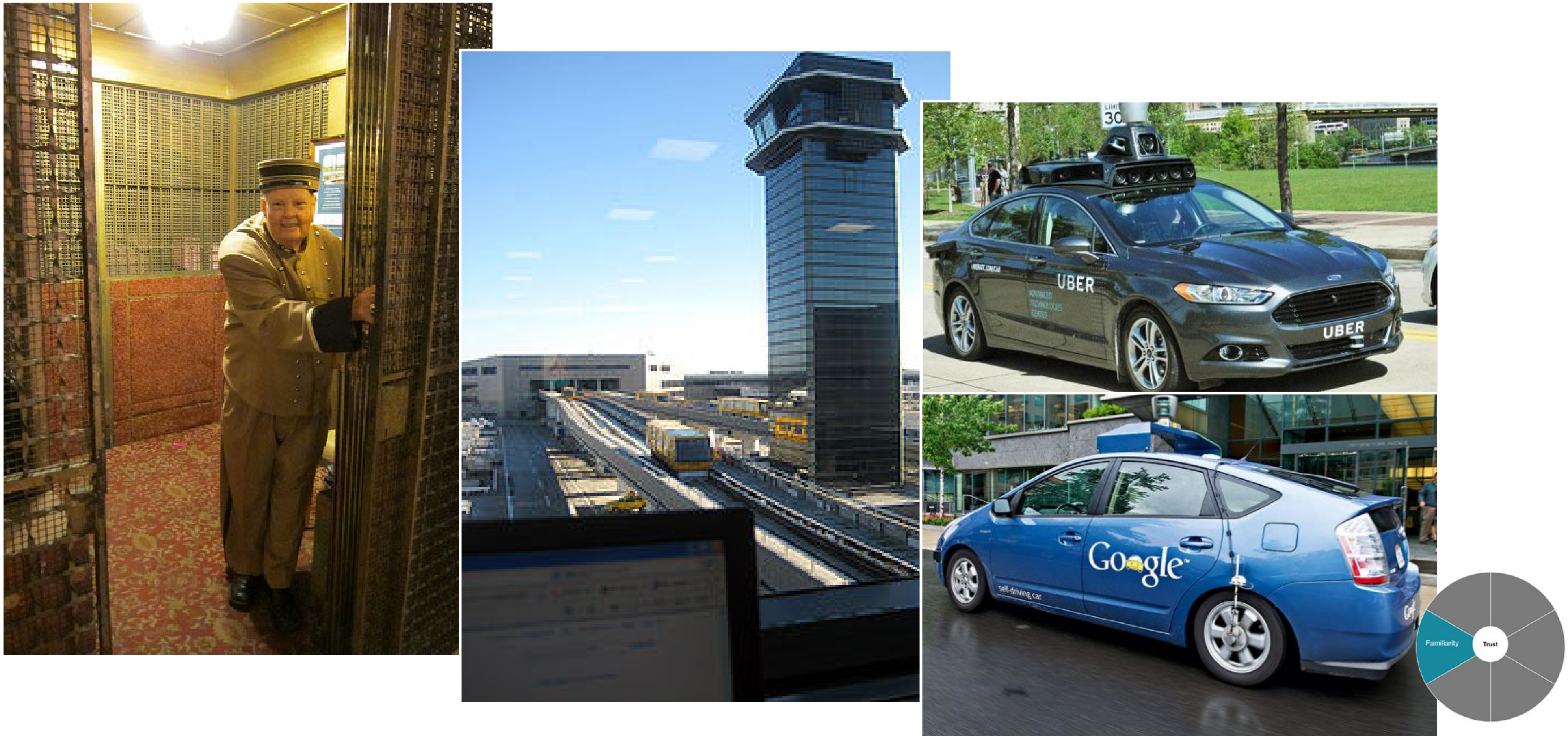
How do we do this for machines?

- Explicit may be easy, but implicit is hard for machines
- Asimov's three laws
- Commander's intent
- Mission orders

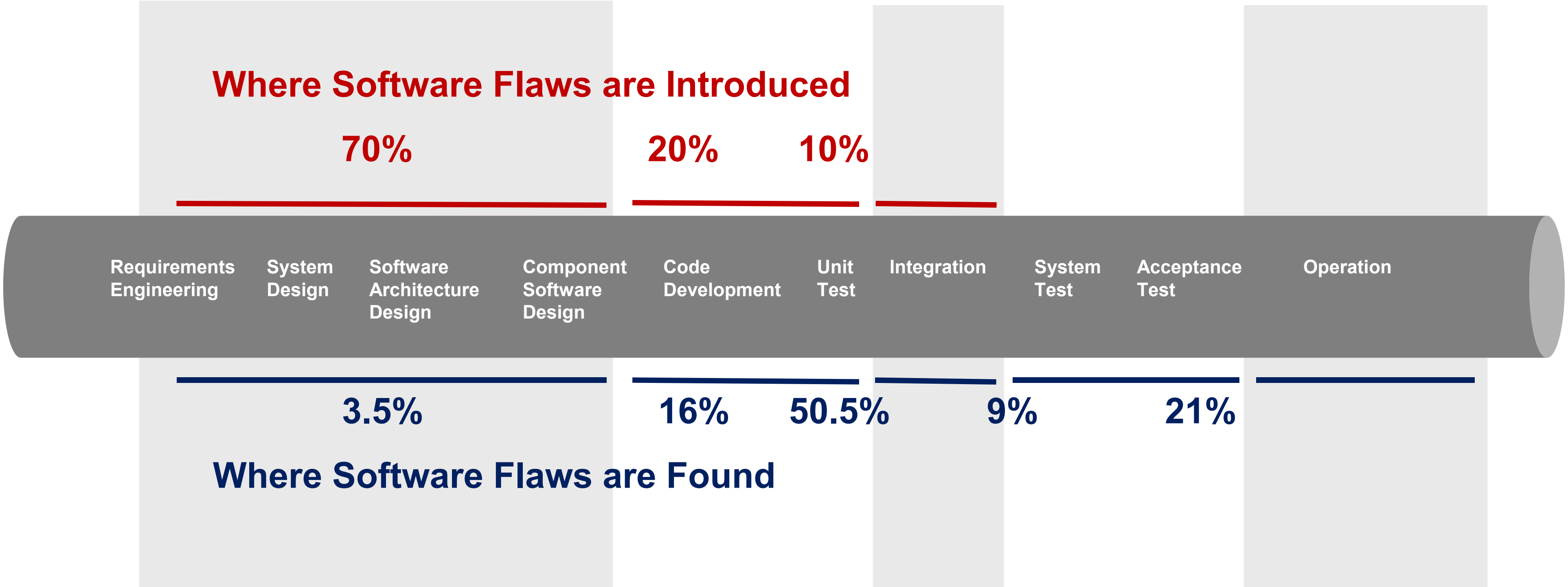
Related to need for explainability and predictability



Gain familiarity



Recognize that software quality is more crucial than ever



Where Software Flaws are Introduced

70% **20%** **10%**

Requirements Engineering | System Design | Software Architecture Design | Component Software Design | Code Development | Unit Test | Integration | System Test | Acceptance Test | Operation

3.5% **16%** **50.5%** **9%** **21%**

Where Software Flaws are Found

Use tooling to integrate the deployment pipeline

Sources: *Critical Code*; NIST, NASA, INCOSE, and Aircraft Industry Studies

Summary

Increase acceptance of non-deterministic performance in T&E

Solid system engineering will be even more important

Current tools and processes may not be sufficient

Transitioning will depend on establishing and building trust

- Complicated by non-deterministic techniques
- Complicated by systems that continue to learn
- Complicated by human-machine teaming

Solid system engineering will determine if we are creating C3PO and Johnny 5

AF archive / Alamy Stock Photo

…or The Borg

AF archive / Alamy Stock Photo

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM18-0976