Adaptable APIs and TAK Integration

Generating Sharable Interoperability from OV-1 through Runtime

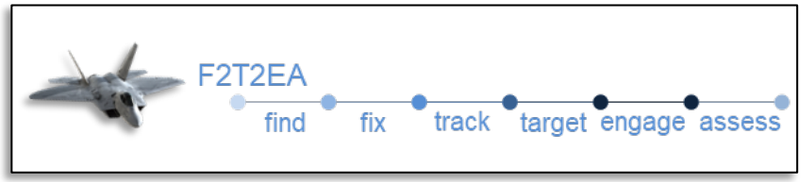
Jimmy “Reverend” Jones, PhD

STITCHES Warfighter Application Team Lead, SAF/AQLV

30 Aug 2023

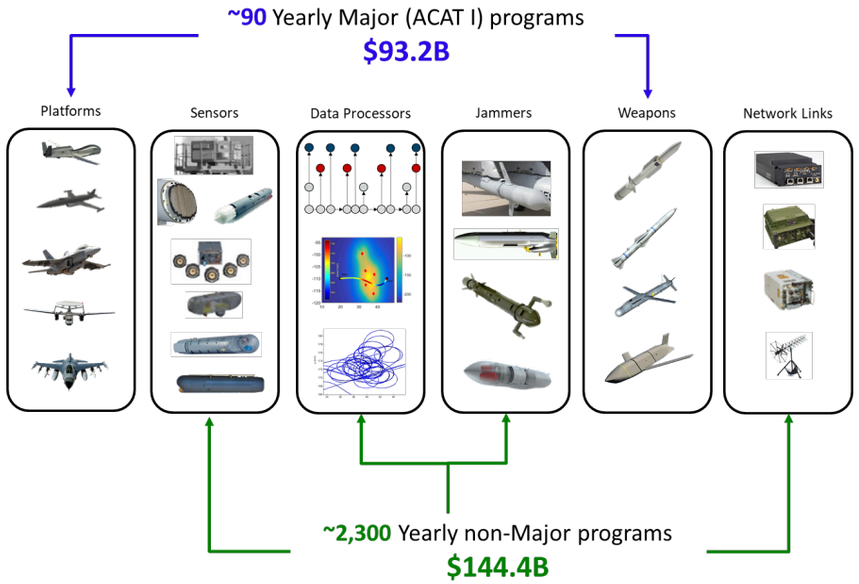
System of systems Technology Integration Tool Chain for Heterogeneous Electronic Systems

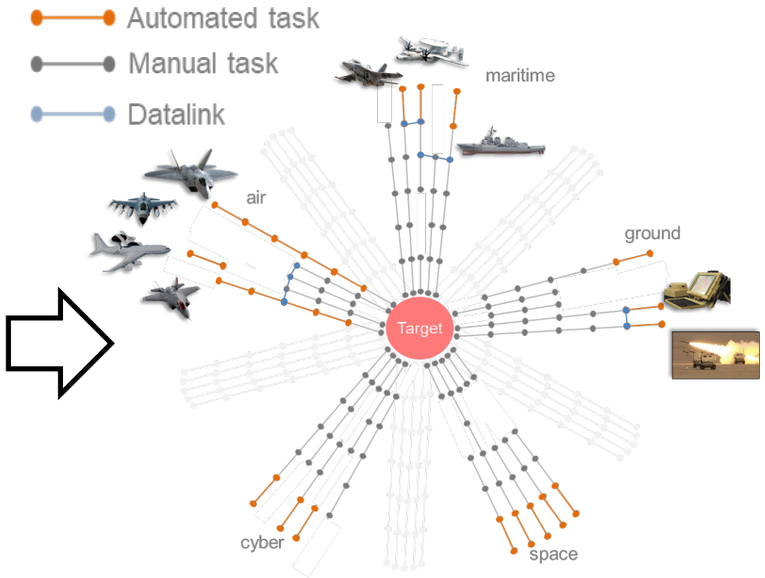
DoD creates “MWS-locked” capabilities



DoD delivers capability over years of process

Individual capabilities are stove-piped within major weapon systems

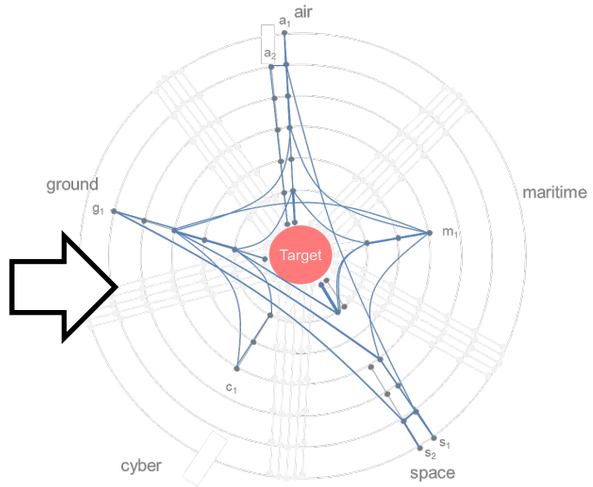




New capabilities from existing systems cannot be created faster than the threat adapts

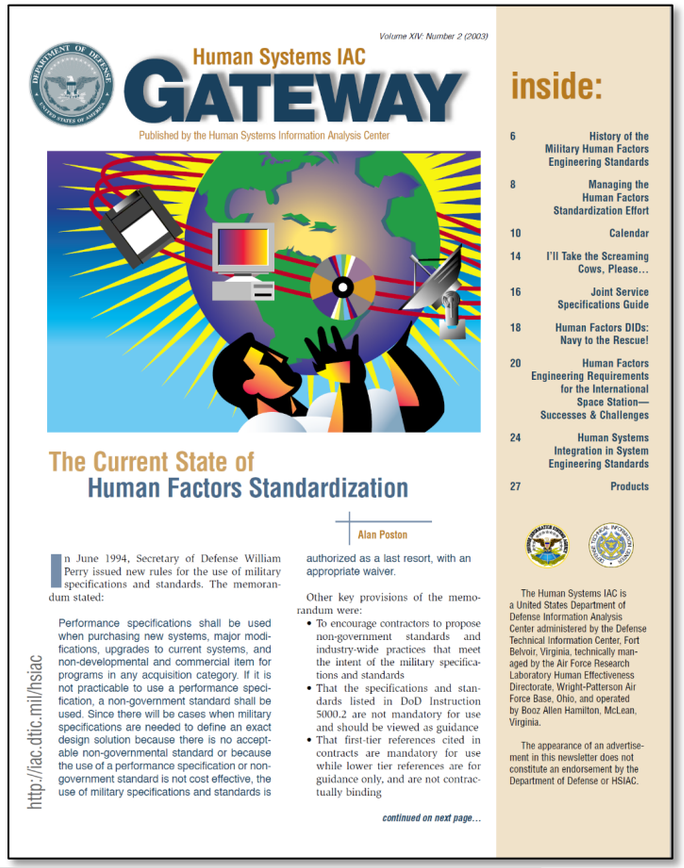
Warfighters want adaptable machine-to-machine capabilities they can connect at mission planning time

Apollo 13 scene of NASA engineers identifying components on board the Apollo 13 capsule and lunar lander to repurpose in order to make a filter CO_2 from the air. HOWARD, R. et al. (2005). *Apollo 13*, Universal City, CA, Universal.



A word on military standards/specifications

- MIL SPEC standardization started post WWII for:
  - "the ability to swap out vehicle subsystems in the highly likely event that they were to be damaged in combat"
- 1994: **45,500 mil specs existed**
- SecDef Perry mandated MIL SPEC reform
  - Reduced MIL SPECS to **~28,300 by 2003**
  - Encouraged non-govt standards and industry-wide practices
  - Specifications and standards listed in DoD Instruction 5000.2 are not mandatory for use and should be viewed as guidance



Volume XIV: Number 2 (2003)

Human Systems IAC

GATEWAY

Published by the Human Systems Information Analysis Center

inside:

The Current State of Human Factors Standardization

Alan Poston

In June 1994, Secretary of Defense William Perry issued new rules for the use of military specifications and standards. The memorandum stated:

Performance specifications shall be used when purchasing new systems, major modifications, upgrades to current systems, and non-developmental and commercial item for programs in any acquisition category. If it is not practicable to use a performance specification, a non-government standard shall be used. Since there will be cases when military specifications are needed to define an exact design solution because there is no acceptable non-governmental standard or because the use of a performance specification or non-government standard is not cost effective, the use of military specifications and standards is authorized as a last resort, with an appropriate waiver.

Other key provisions of the memorandum were:

- To encourage contractors to propose non-government standards and industry-wide practices that meet the intent of the military specifications and standards
- That the specifications and standards listed in DoD Instruction 5000.2 are not mandatory for use and should be viewed as guidance
- That first-tier references cited in contracts are mandatory for use while lower tier references are for guidance only, and are not contractually binding

continued on next page…

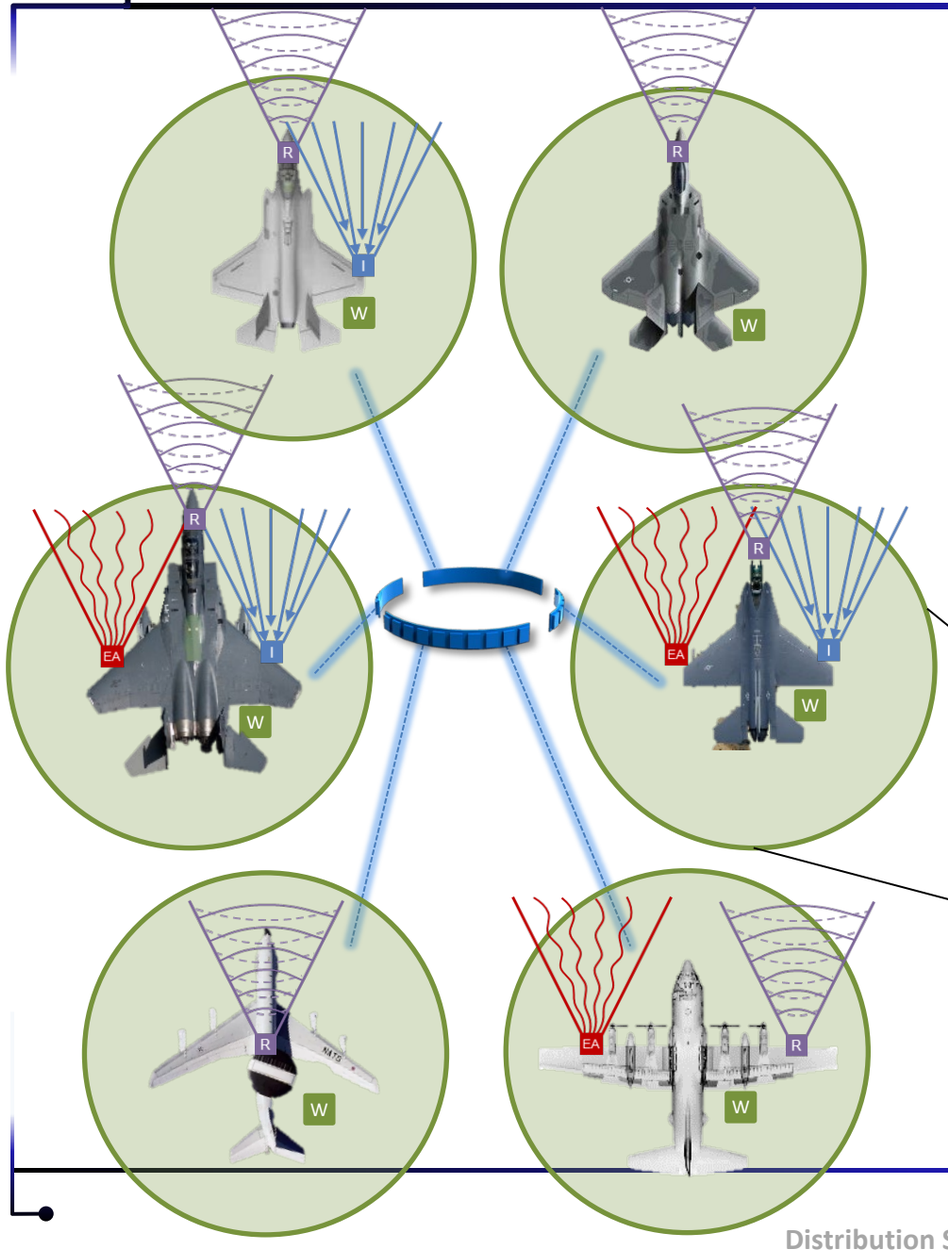
http://iac.dtic.mil/hsiac

The Human Systems IAC is a United States Department of Defense Information Analysis Center administered by the Defense Technical Information Center, Fort Belvoir, Virginia, technically managed by the Air Force Research Laboratory Human Effectiveness Directorate, Wright-Patterson Air Force Base, Ohio, and operated by Booz Allen Hamilton, McLean, Virginia.

The appearance of an advertisement in this newsletter does not constitute an endorsement by the Department of Defense or HSIAC.

Challenge Acquisition Requirements: Program Managers and acquisition decision makers at all levels shall challenge requirements because **the problem of unique military systems does not begin with the standards**. The problem is rooted in the **requirements determination phase** of the acquisition cycle. - William J. Perry, 29 June 1994

Interoperability Issues Abound in the DoD

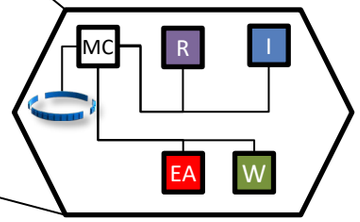


Adding interoperability between aircraft takes years

- Aircraft are upgraded independently creating unique messages
- Machine-to-machine interfaces in an aircraft are highly coupled
- Even coupling between versions of the same spec (e.g., Link-16) is problematic

This interoperability problem occurs within the aircraft as well

- Components are tightly coupled together using a local message standard that doesn't match the documentation
- Adding new technology (e.g., sensor) requires upgrades to multiple components, increasing expense and time

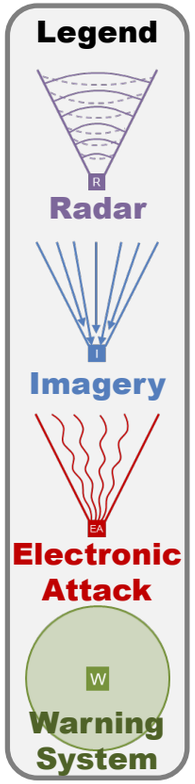


Each aircraft is collection of independent subsystems designed at different eras requiring interoperability

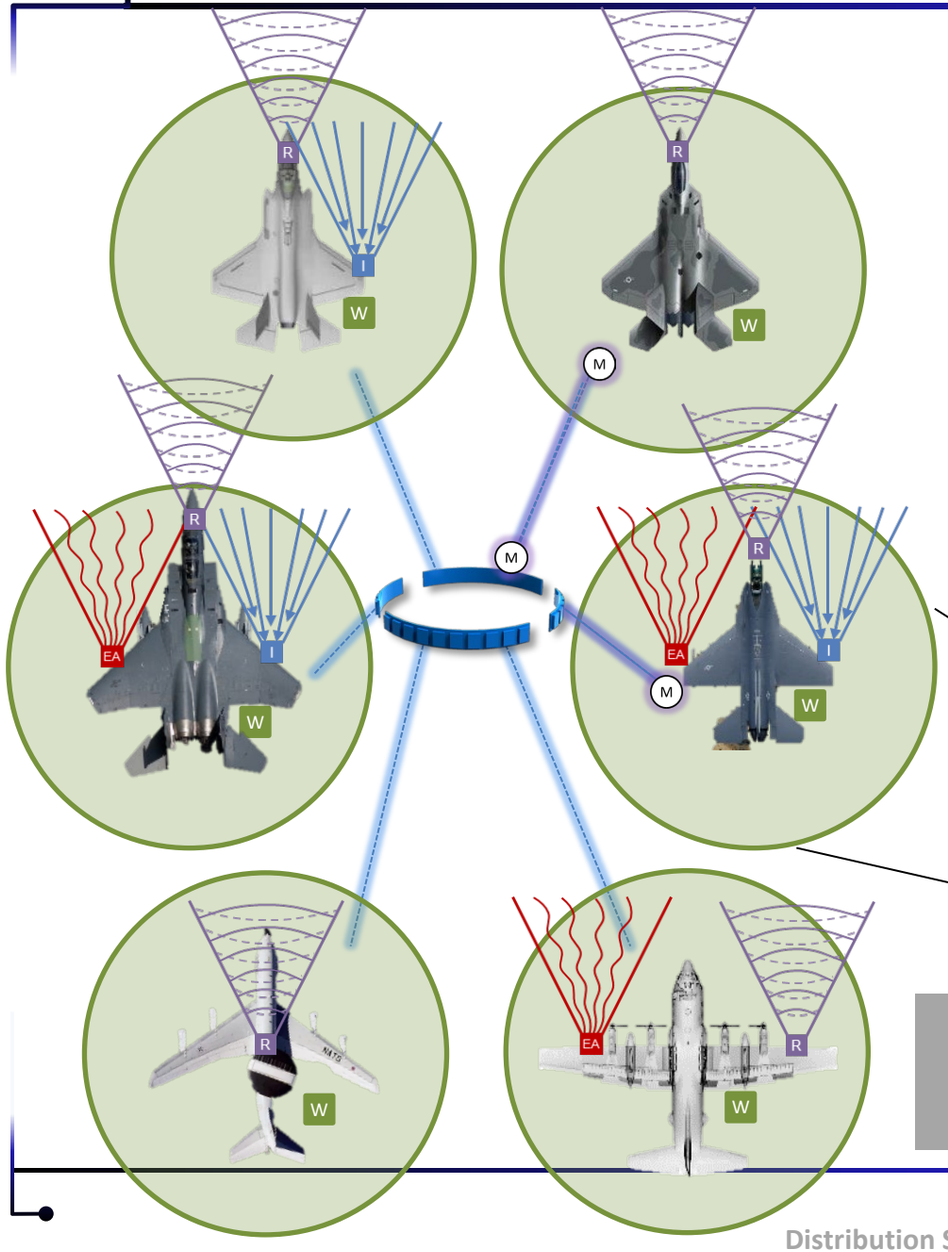
Interoperability is programmed into each system System upgrades compete in 2-3 year block cycles

Global standards have similar issues

- No interoperability between generations of the standard
- Uncertain interoperability between implementations



STITCHES is a toolchain to generate interoperability

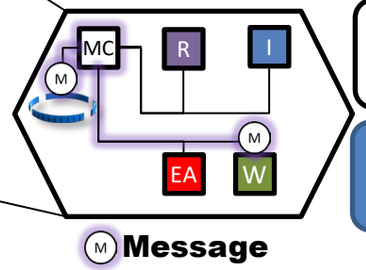


STITCHES efficiently generates message translators

- System software is modified once for STITCHES compatibility
- Generated code is uploaded like mission data files (MDFs)
- Allows for fast and easy reconfiguration of mission capabilities

Auto-generate efficient glue code to

- Interoperate between new subsystems and existing components; old and new generations of the “same” open specification
- Transform platform data to comm’s message formats or encode any message inside fixed format standards like Link-16



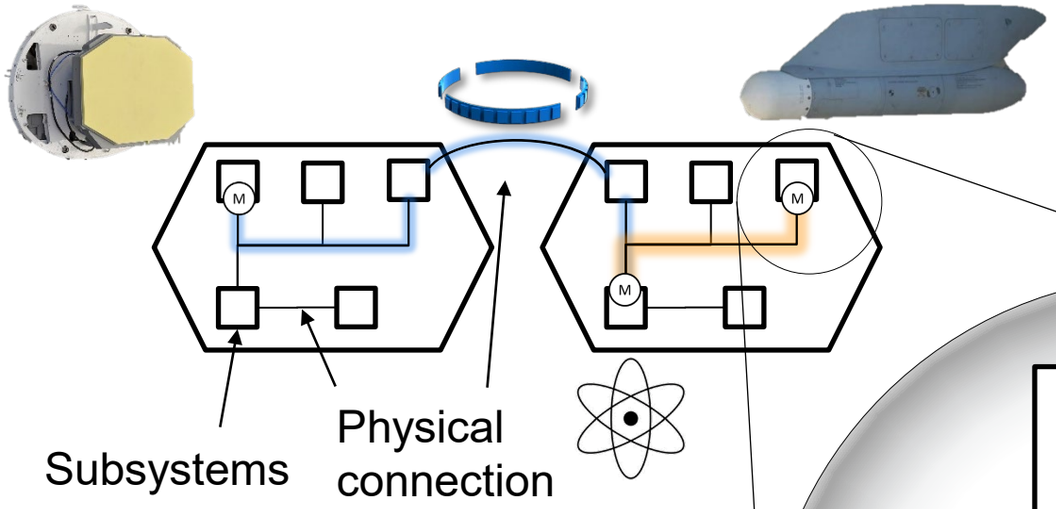
A mission computer’s (MC) software is modified to allow STITCHES MDFs to translate and control data flow

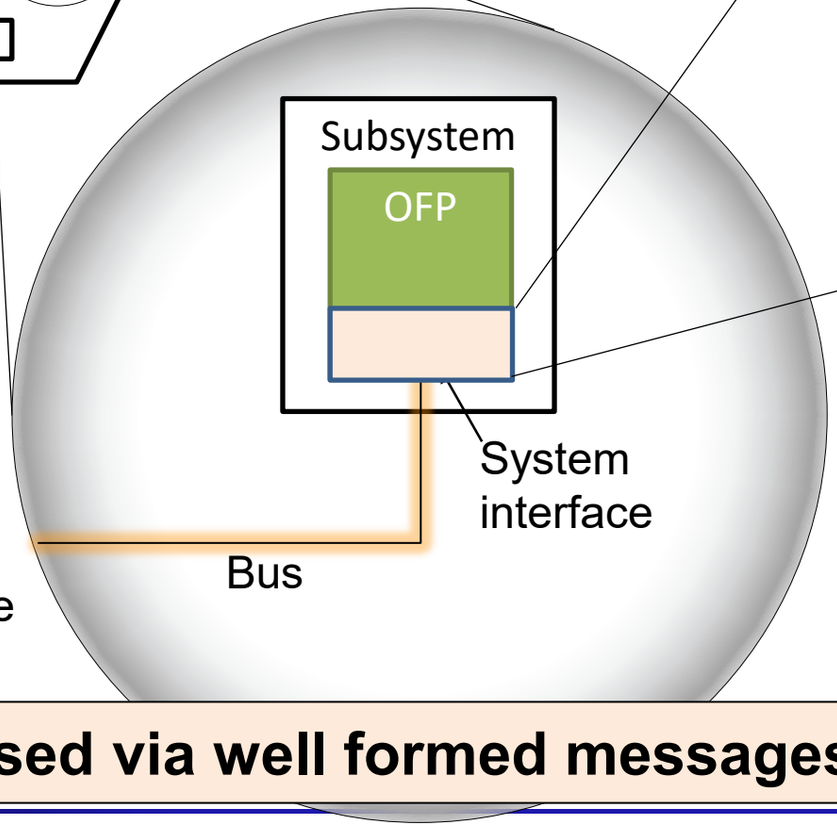
Any data on buses connected to STITCHES MDFs may be translated and rerouted inside or outside an aircraft

STITCHES MDFs **efficiently translate data** and **control data flow** in order to create interoperability between and within connected platforms

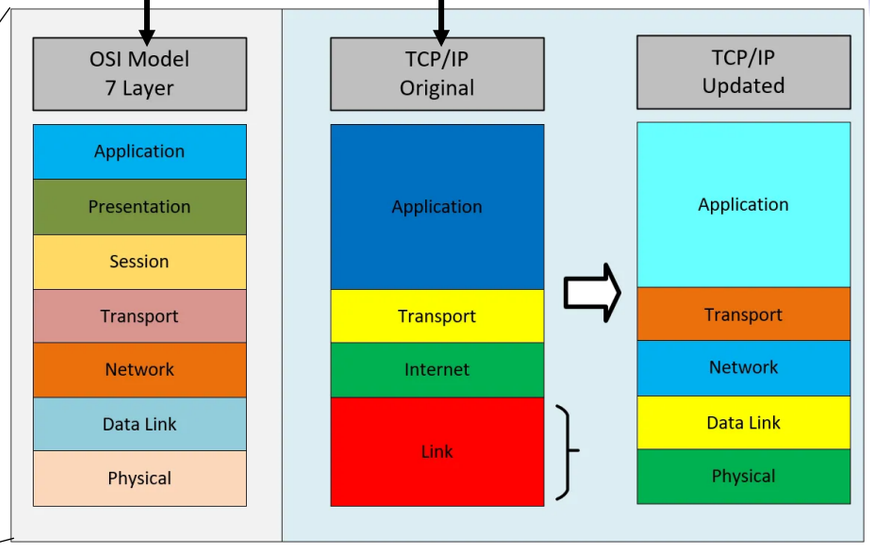
What's an interface and why it's contextual

A System of System where many systems customize interfaces





Failed in 80's "ARPA" internet



Both commercial and DoD systems blend system layers

Layers are a human construct to make it easier to explain or parse work units out

Every system has an interface based on their community:
- MIL-STD 1553
- MIL-STD 1773
- MIL-STD 1760
- ARINC 429
- Fiberchannel
- IEEE 1394
- SOAP, XML, Ethernet...any API

None of these standardize data content

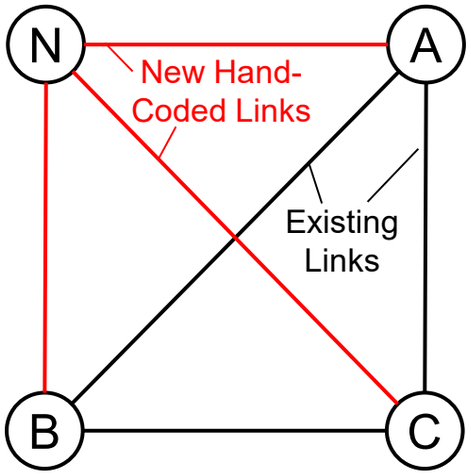
Each 'layer' is accessed via well formed messages

OFP: Operational Flight Program

Interface Translator Technologies

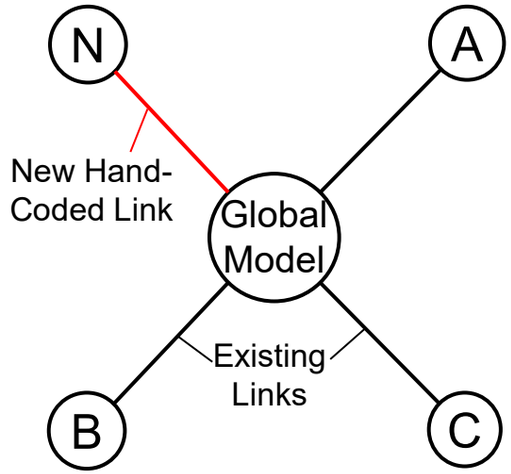
(A) (B) (C) Existing Interfaces (N) New Interface

Link-by-Link Translation



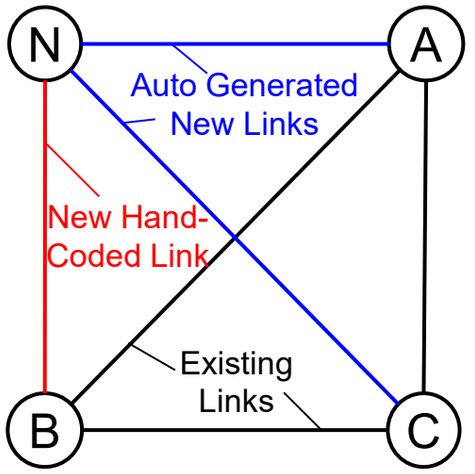
- × Inefficient integration
  - L hand-coded translations (some may already exist)
- ✓ Efficient translation
  - 1 step between each interface
- Many examples

Common Model Translation



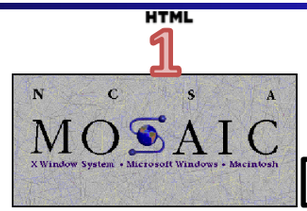
- ✓ Efficient integration
  - 1 hand-coded translation
  - Possible model update
- × Inefficient translation
  - 2 steps between non-global-model interfaces
- E.G. Gateways

Field Transform Graph



- ✓ Efficient integration
  - 1 hand-coded translation (L chances it already exists)
- ✓ Efficient translation
  - 1 step between each interface
- STITCHES

“I want it to work like the internet”



HTML 1 — 1993 — 2008

HTML 2 — 1993-1997

HTML 3 HTML 4 — 1996 — 2020

Secure Sockets Layer (SSL) 1995

Transport Layer Security (TLS) 1.0/1.1 — 1999 — 2020

Image from: https://www.bankmycell.com/blog/iphone-evolution-timeline-chart

1975 1980 1986 1997 2001 2010 2022

“I want it to work like the internet”

HTML 1 HTML 2 HTML 3 HTML 4

N C S A MOSAIC
X Window System • Microsoft Windows • Macintosh

1993 2008

1993-1997

1996 2020

HTML 5

Secure Sockets Layer (SSL)
1995

Transport Layer Security (TLS) 1.0/1.1
1999 2020

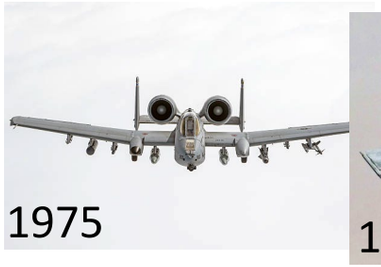
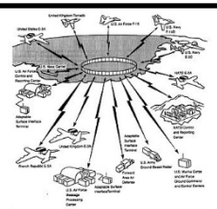


Image from: https://www.bankmycell.com/blog/iphone-evolution-timeline-chart

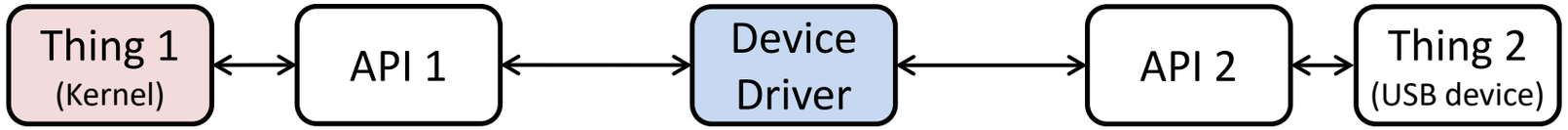
1975 1980 1986 1997 2001 2010 2022



Formal Verification from APIs through the Device Driver

20 April, **1998**: Infamous Blue Screen of Death (BSOD) appeared while unveiling Windows 98 USB plug-and-play features

Issue was a bug in a device driver that implemented an API to the kernel incorrectly

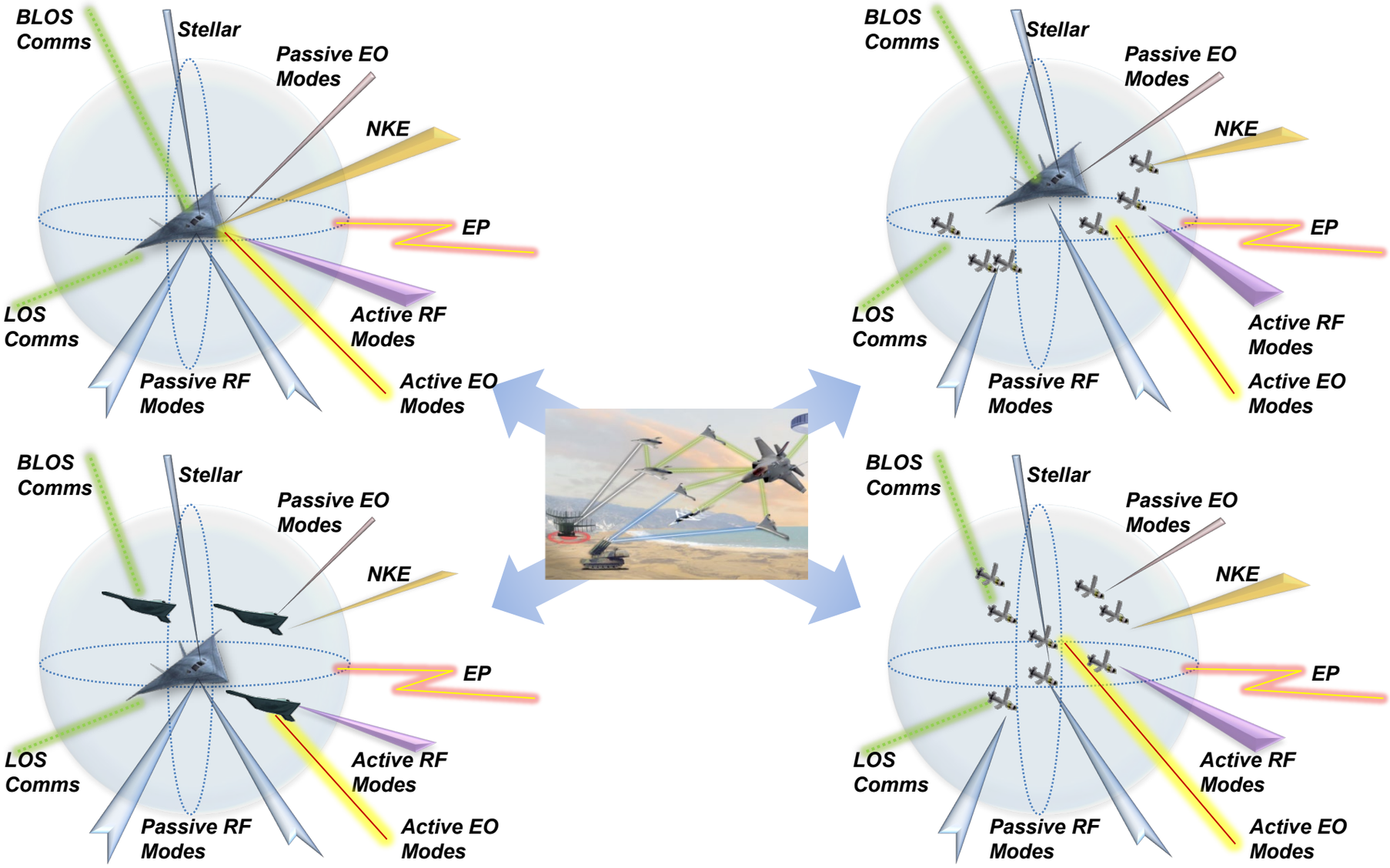


Microsoft launched 3 efforts to solve/minimize implementation conflicts
Solution: **SLAM**. *THE* most successful and longest use of **Formal Verification**

> *"The goal of the SLAM project is to check whether or not a program obeys "API usage rules" that specify what it means to be a good client of an API."* **T. Ball and S. K. Rajaman, Jan 1 2002**

> *"Things like even **software verification**, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do **actual proof about the software and how it works** in order to **guarantee the reliability**."* **Bill Gates, April 18, 2002.**

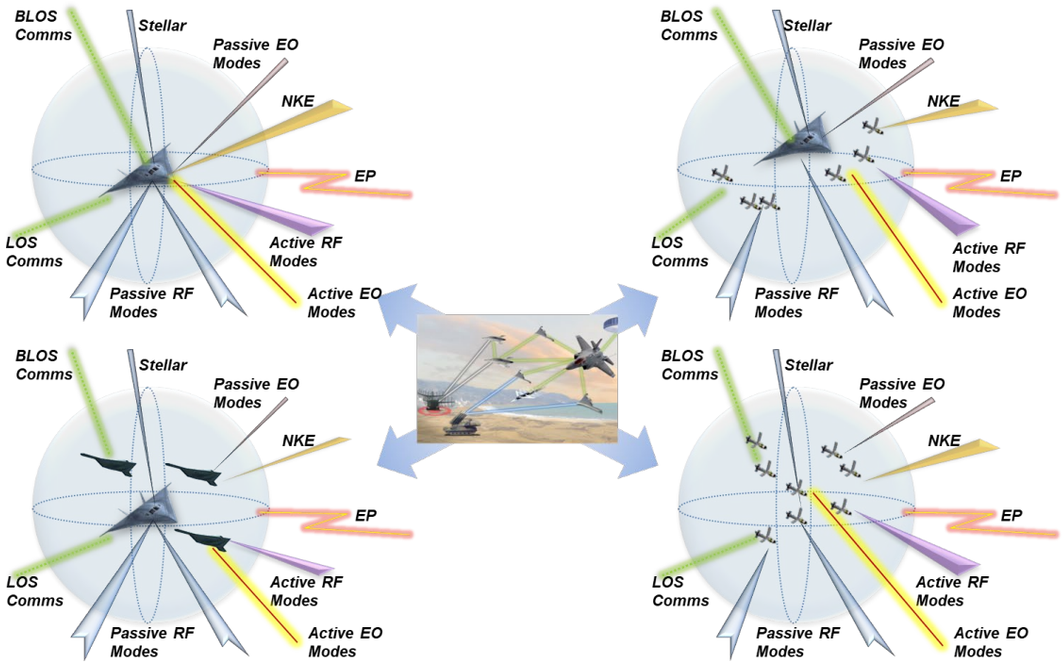
Opportunity: Composable Architectures to Avoid Obsolescence



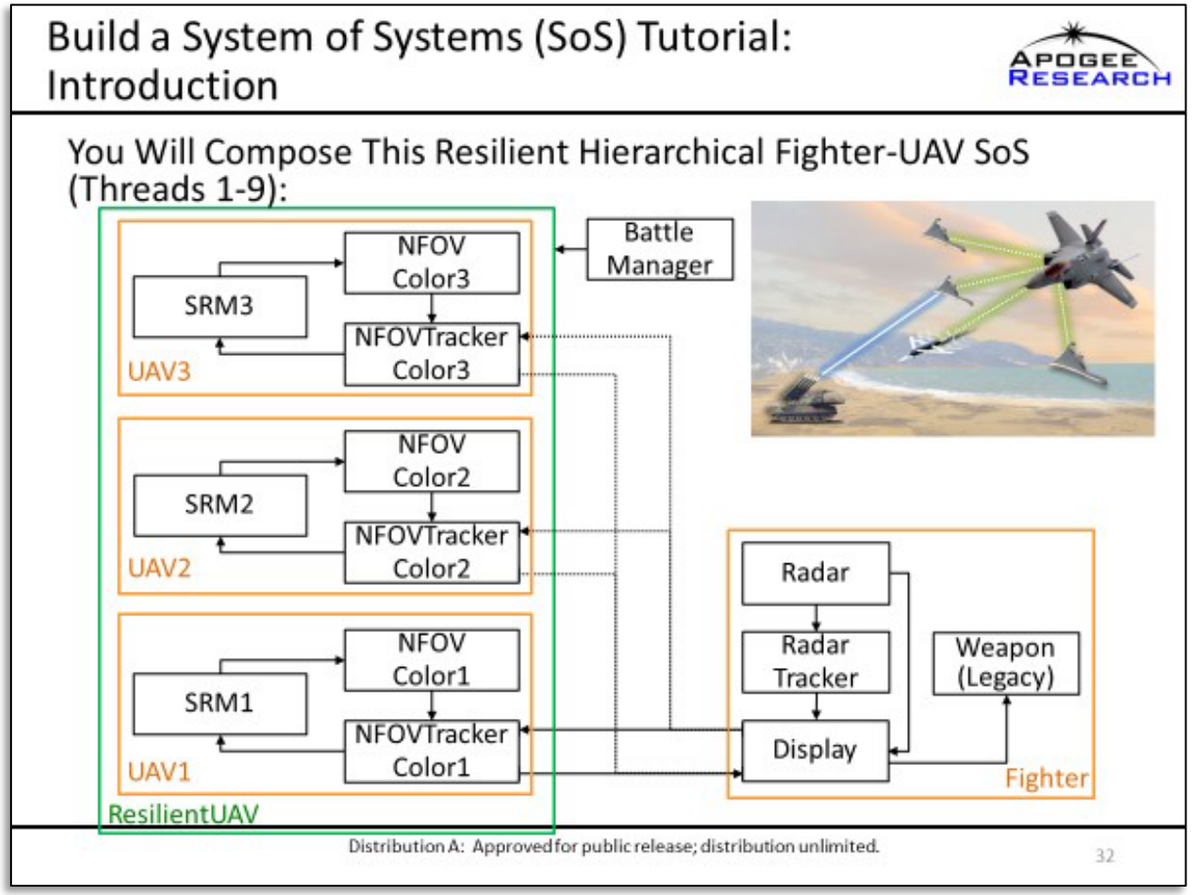
Courtesy of Dr. Joshua Bernstein, Northrop Grumman Electronic Systems

PED: Processing, Exploitation, Dissemination

Portable, Scalable, & Reusable since 2014



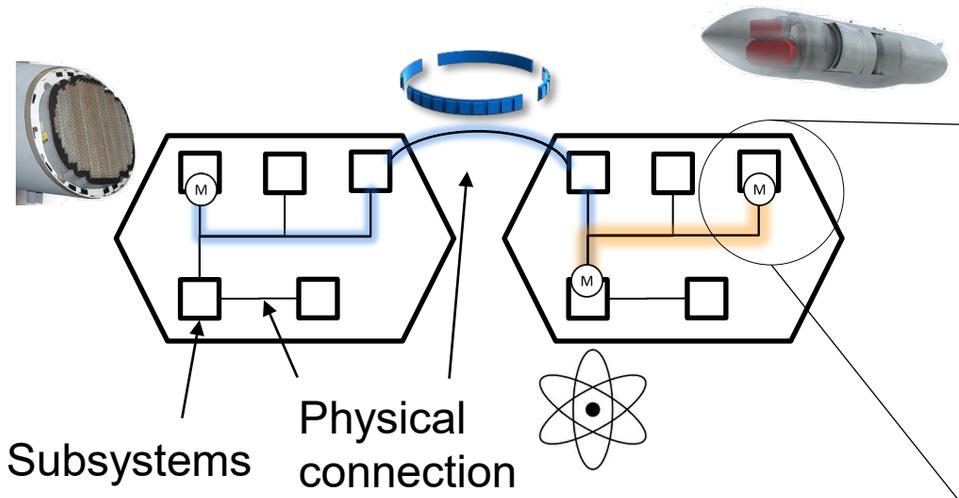
Concept

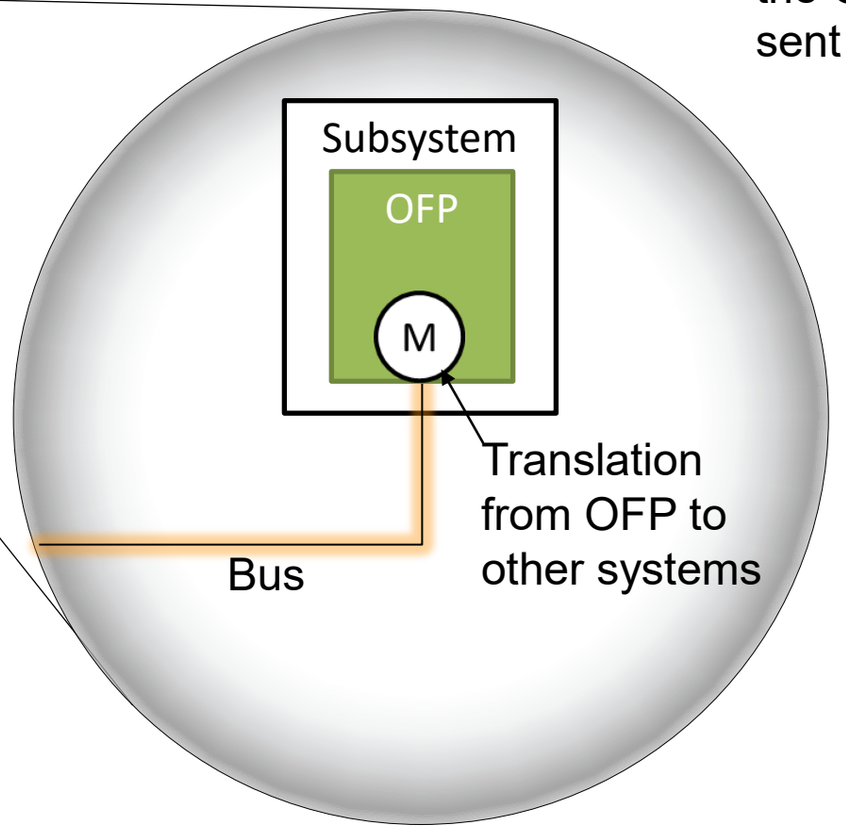


STITCHES' Training Material

Current integration requires updates to several systems

A System of System is created by a user





Any translation happens inside the OFP before a message is sent out on the bus

Often datalinks (Link-16) must update their standard to pass new messages

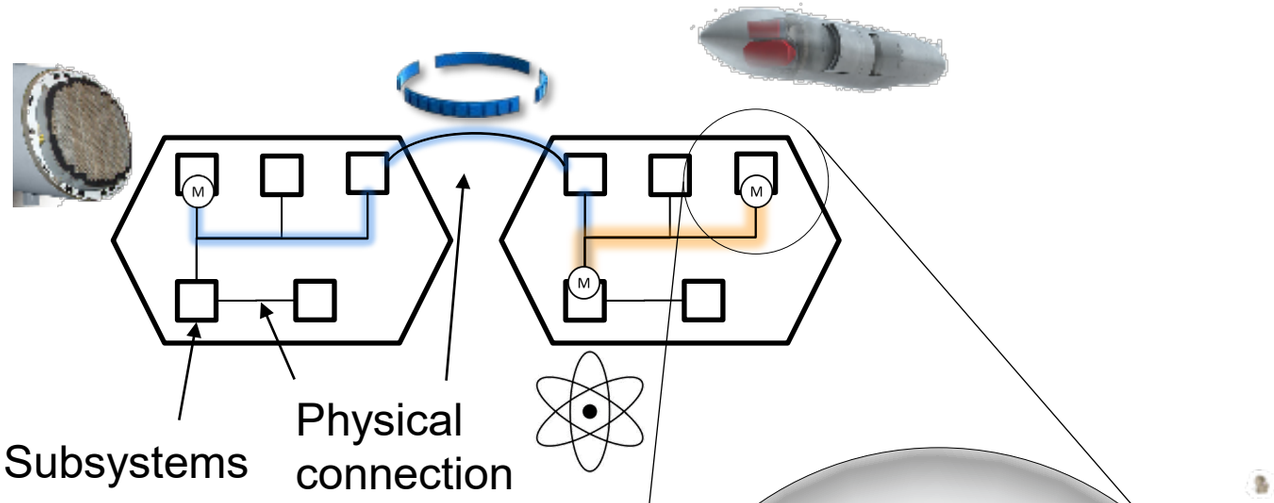
3 generalized types of integration

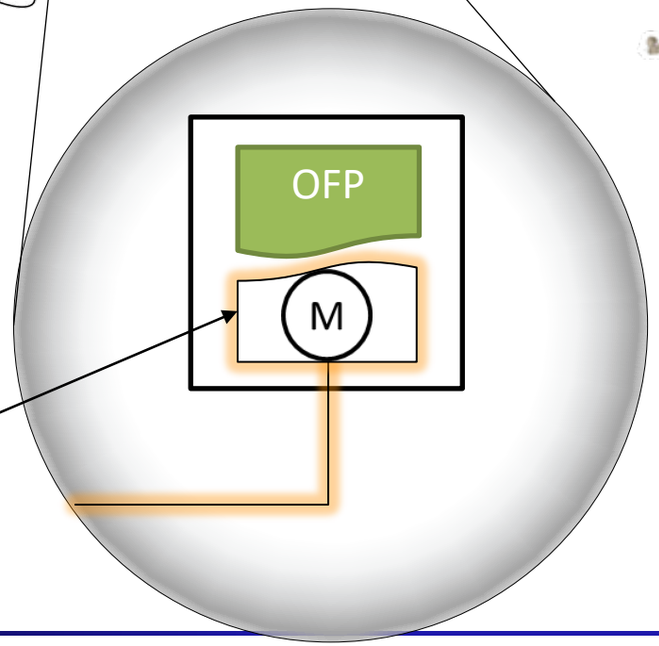
Intra-subsystem (new algorithm)
Intra-aircraft (new subsystem)
Inter-aircraft (new external connection)

A translator is created for every new connection in the System of System

DARPA created a new toolchain called STITCHES

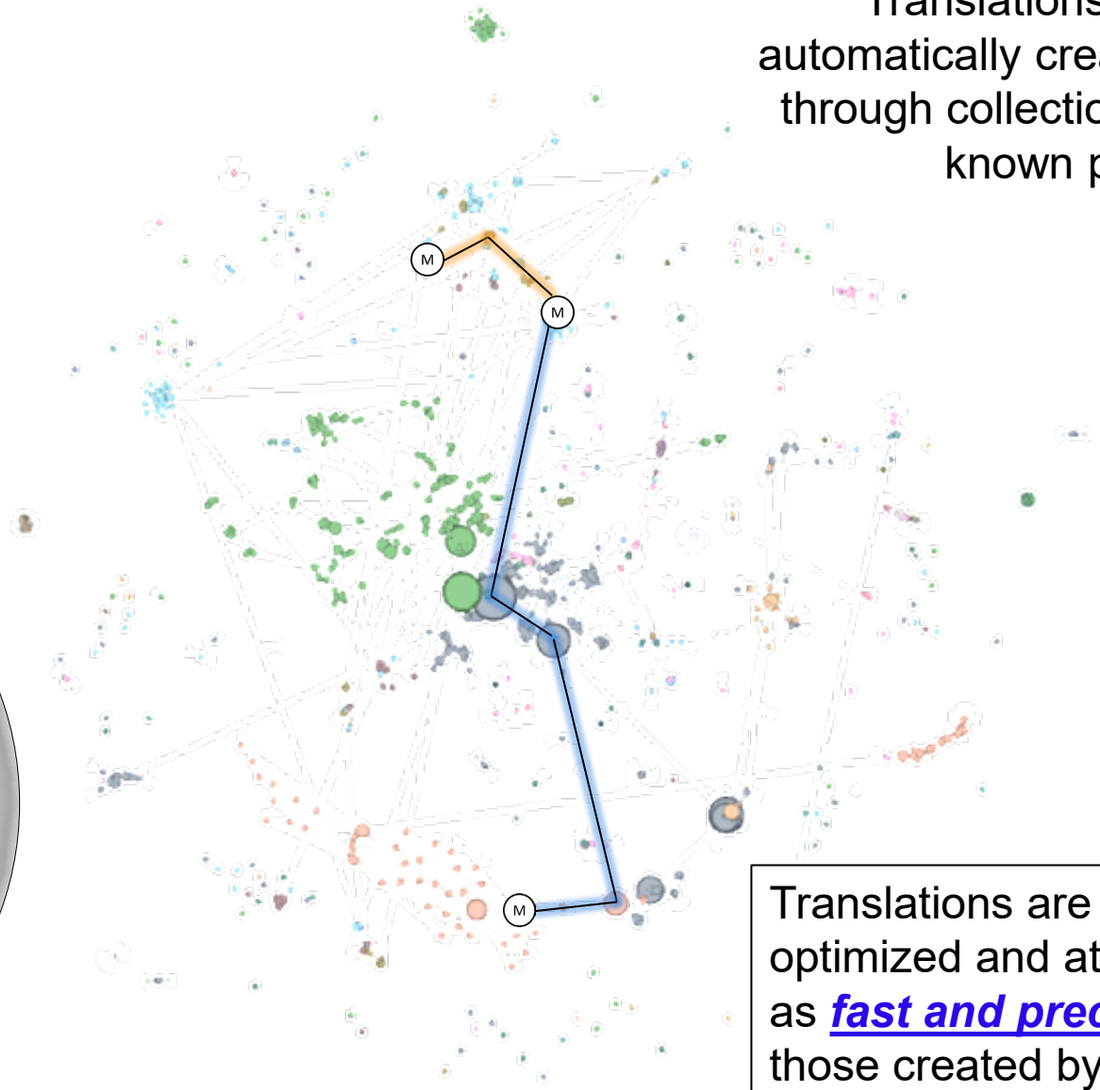
A System of System is created by a user





Automatically created translations are separated from the OFP

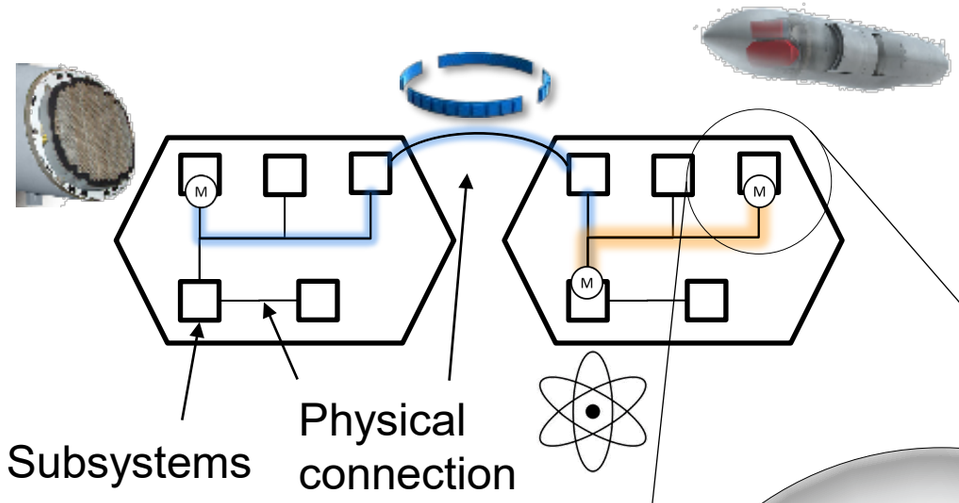
Translations are automatically created through collection of known pairs

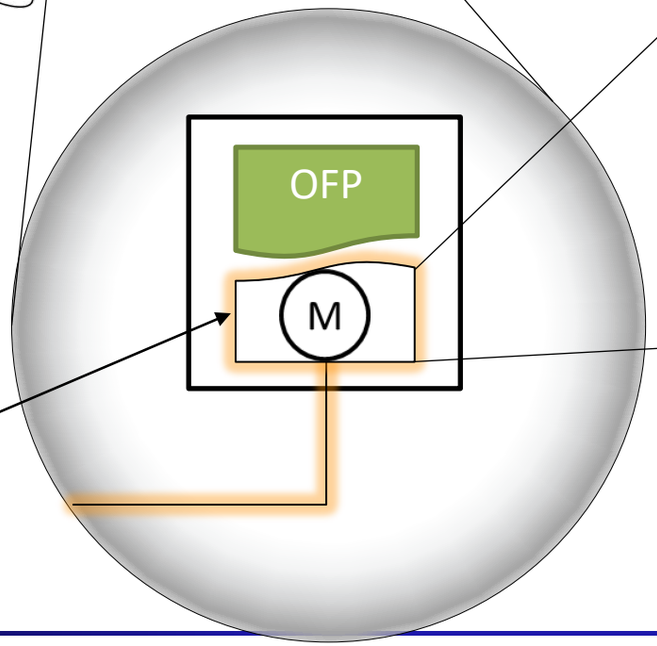


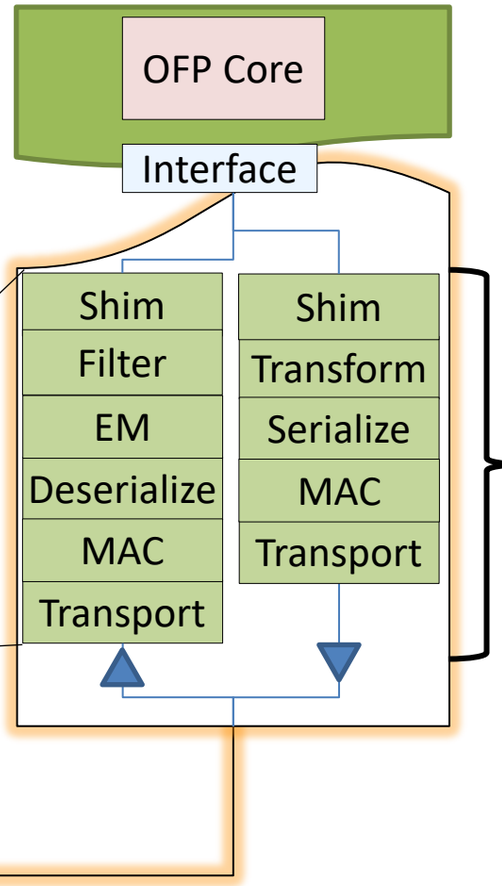
Translations are optimized and at ***least*** as ***fast and precise*** as those created by hand

DARPA created a new toolchain called STITCHES

A System of System is created by a user







Interfaces automatically created producing STITCHES MDFs

Includes

Cross domain rulesets
Translation
Authentication
Bus monitors
Data filters

Automatically created translations are separated from the OFP

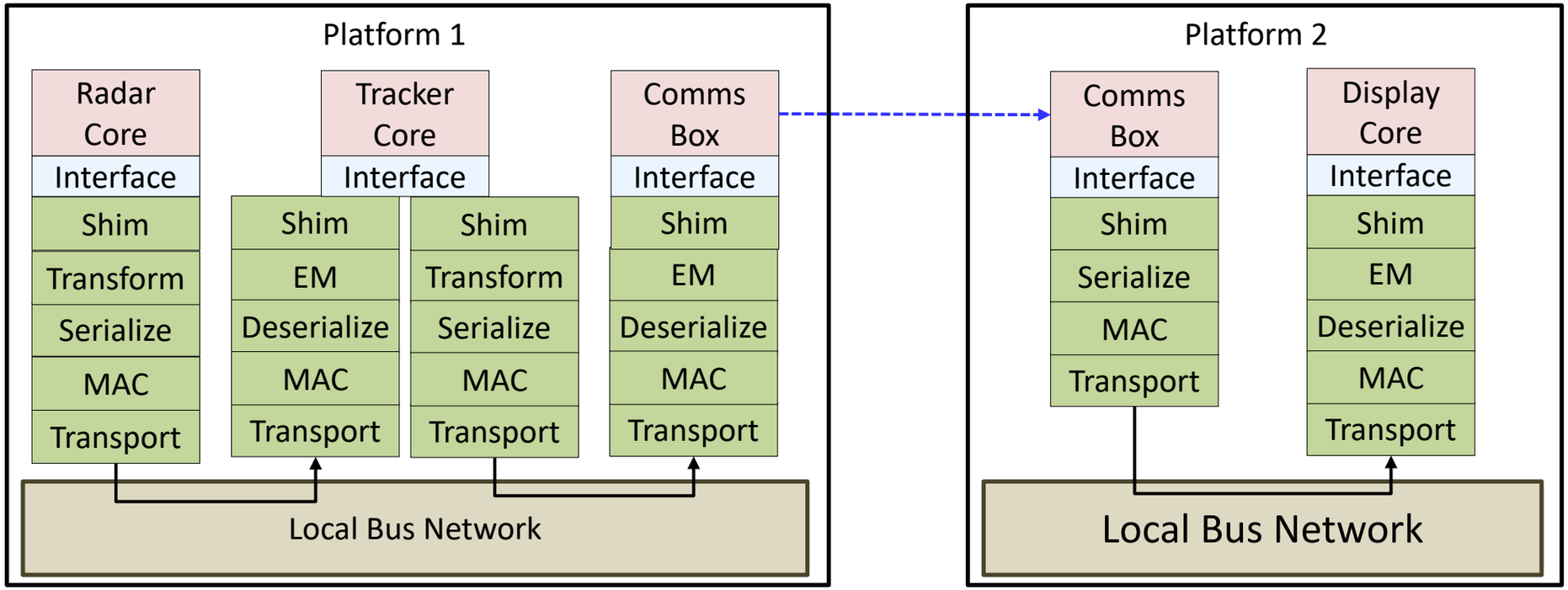
EM: Execution Monitor
MAC: Machine Authenticated Code

What Does STITCHES Produce?

Subsystem Core — Developed by Subsystem Engineers

Interface — Developed by Subsystem Engineer with STITCHES Autogenerated Libraries
Developed once per Core Version, Works for all SoS Configurations

HCAL — Autogenerated by STITCHES; Tailored to Each SoS Configuration

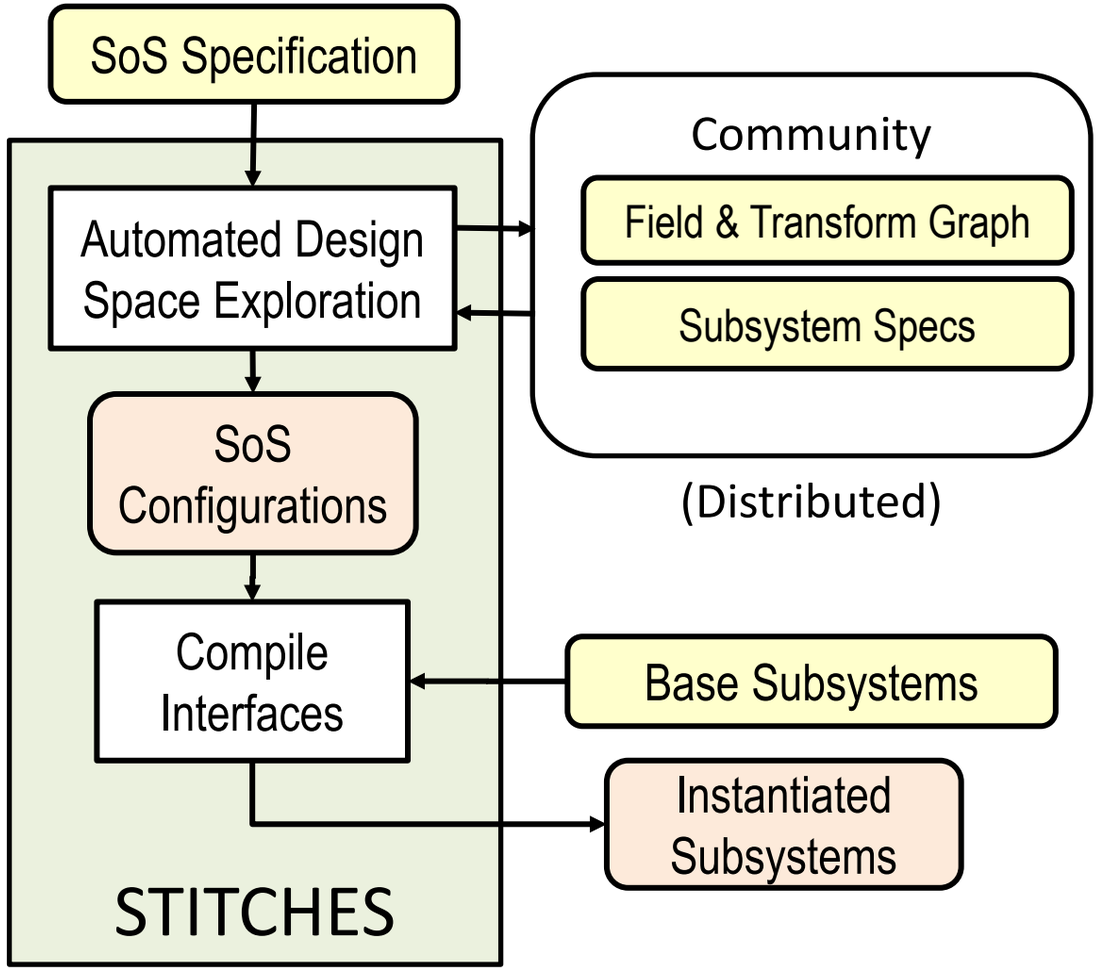


MAC: Message Authentication Code
EM: Execution Monitor

HCAL: Heterogeneous Critical Abstraction Layer

STITCHES is Focused on Implementing a Scalable Approach to Building SoS Capabilities

- Design Space Exploration
  - Process FTG to construct transformation chains
  - Specify interface stack by forming & solving optimization problems
- Compiler
  - Construct interface stack structure
  - Optimize transforms for this instance of the interface
  - Provide cyber security through whitelist property enforcement
  - Generate C++/C90/Java code & compile into binaries



Result: High performance interfaces optimized for each application

SoS: System of systems

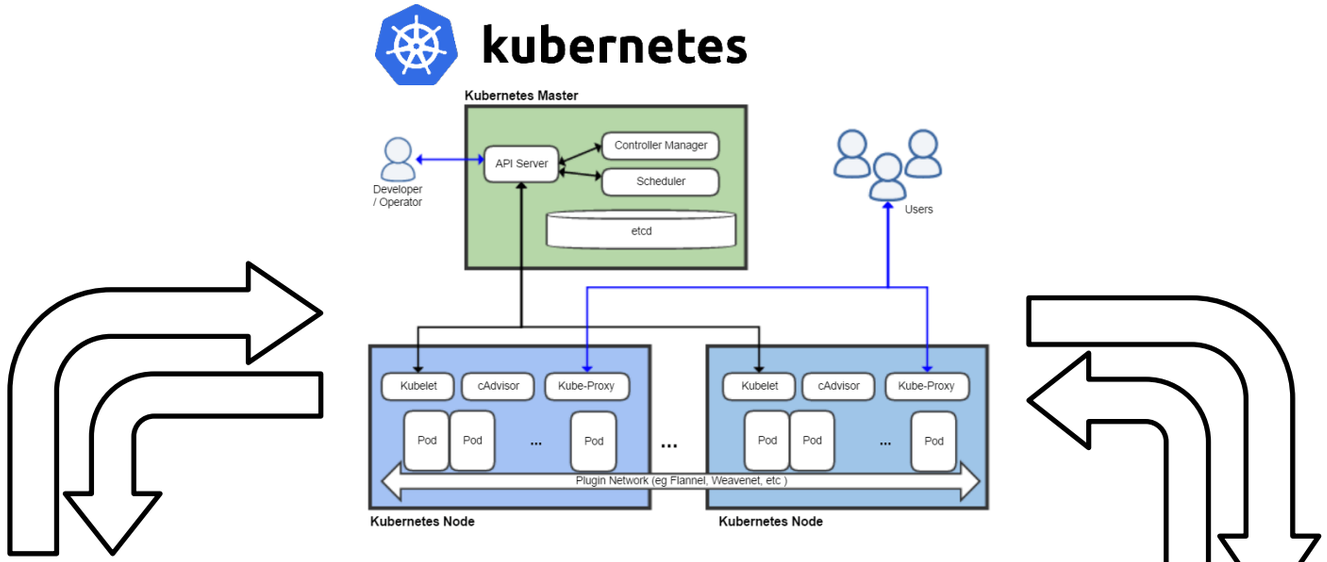
Synergy with Commercial composition and containers

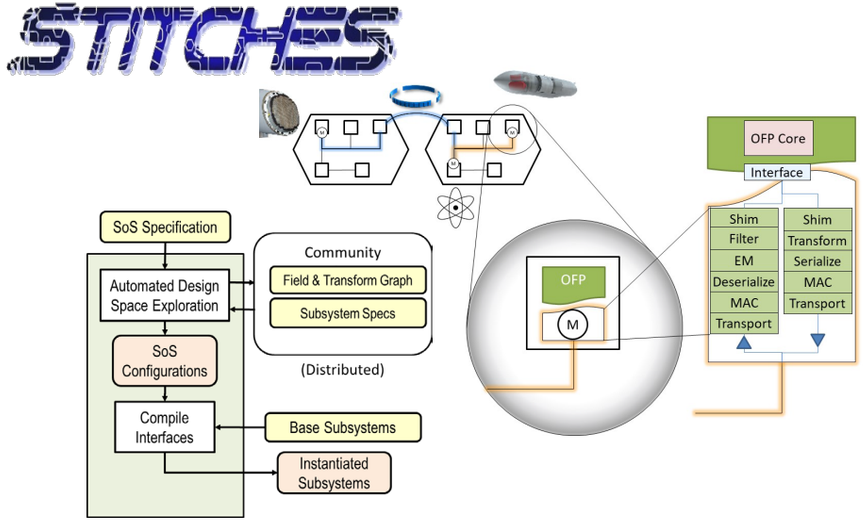
Each can call the other's available APIs and configure, deploy, start, stop, etc

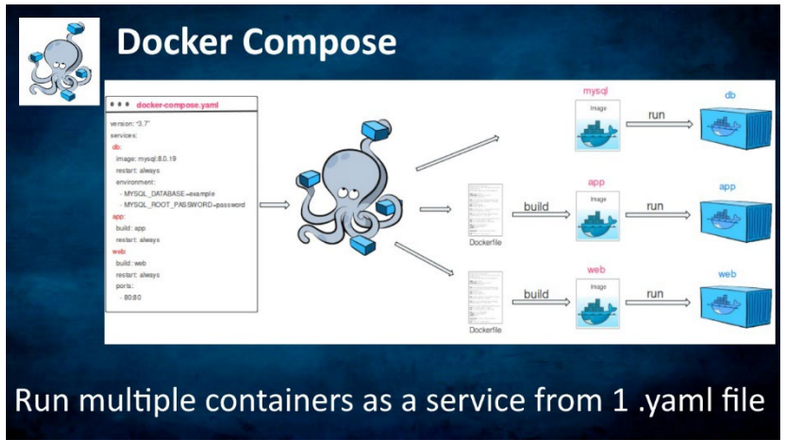
High level specifications define the deployment of software

Sharable inputs and APIs feed the deployment engines

Each is user configurable





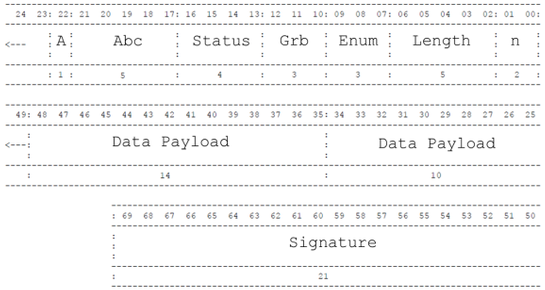


Interoperability via Legacy Comms, What if desired messages aren't supported?

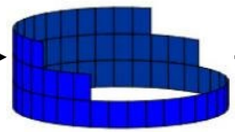
Pod Tracks

?

Legacy Comm Message



Legacy Comm Message

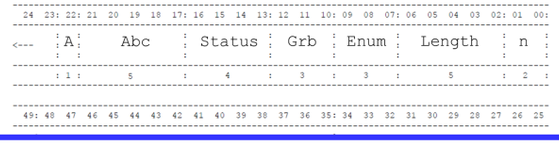


Display Tracks

?

Legacy Comm Message

Auto-generate Transcoder to “Encode” Messages into Legacy Communication Messages



~20 hours to Study Spec and Capture Each Message in FTG (1 time Cost)

Legacy Msg (560 Bits)

Transcoder Generator

15 Seconds on Laptop

Transcoder

Free Bits (483)

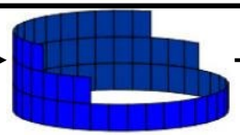
Fixed Bits (77)

Pod Tracks

Serialize

Transcode

Legacy Comm Messages



Manually Defined

Automatically Created

Display Tracks

Transform

Pod Tracks

Deserialize

Transcode

Legacy Comm Messages

Capability Flown in Live Flight since January 2018 with Legacy Comm’s System

STITCHES Solves a Number of Critical Issues

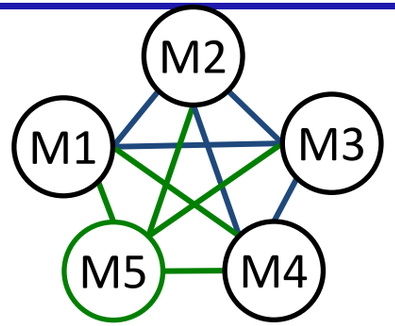
- STITCHES is Open Sourced!
- Global interoperability without global consensus on interface specification
  - Stateless interactions (message transformations)
  - Stateful interactions (multiple source messages required to form destination message)
- Efficient reuse in and evolution of the architecture
- Near real-time construction of the SoS from specification
- Allow legacy subsystems and existing Open Architectures to interoperate with each other and new subsystems on different standards
- Optimized implementation of interfaces that are small and fast
  - Support for high speed packed representations
- Support for tunneling data through legacy communication systems (e.g., Link16)
- Cyber defenses via autogenerated run-time execution monitors that filter out messages due that don’t meet the data and SoS configuration specifications


STITCHES

Key Insight that Makes STITCHES Different

Local Message Standards

- Flexible – You can add new messages easily because no global coordination
- Inefficient - Require N^2 Transforms (all pairs) for Interoperability

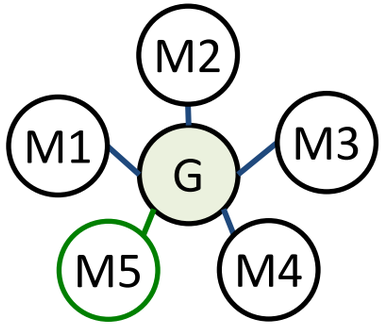


Transform M2 <- M1:
M2 = T21(M1)
Transform M5 <- M1:
M5 = T51(M1)

M# = Message #

Global Open Standards

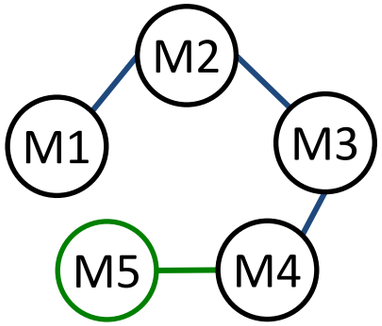
- Efficient – N Transforms to/from the Global Standard
- Not Flexible – Can't change to accommodate new technology without tremendous effort



Transform M2 <- M1:
M2 = T2G(TG1(M1))
Transform M5 <- M1:
M5 = T5G(TG1(M1))

Incremental Standards (STITCHES)

- Efficient – ~N Transforms for Interoperability
- Flexible – You can add new messages easily without significant coordination
- Allows multiple generations of the architecture to function at once



Transform M2 <- M1:
M2 = T21(M1))
Transform M5 <- M1:
M5 = T54(T43(T32(T21(M1))))

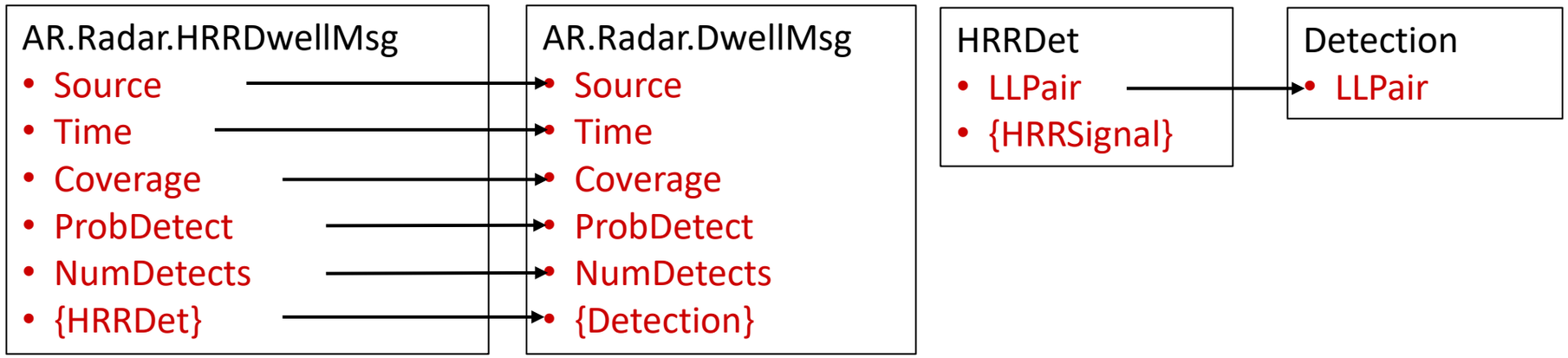
Store the set of these relationships in a Fields and Transform Graph (FTG)

Walking through the Steps…

- Create a STITCHES enabled subsystem (Time: ~Days; Freq: Once per SS version)
  - Start with an existing subsystem (SS) core
  - SS engineer models the interface in the FTG (creates nodes)
  - STITCHES auto-generates SS Interface (SSI) skeleton; SS engineer completes SSI
  - SS Engineer adds FTG links to other nodes to connect to a community
  - Check in the FTG (includes annotations and verification)
- Create a new SoS (Time: ~Hours; Frequency: Once per SoS)
  - SoS engineer defines the SoS Specification (SSes and their connections)
  - If needed, SoS engineer must add any missing required links in the FTG
  - Check in New FTG including incremental verification (with tools such as AGREE)
  - SoS engineer runs STITCHES on the SoS Specification to Build the SoS (generate code)
- FTG provides re-use without common interfaces
  - An FTG Node is re-used in many messages
  - Messages are re-used in many subsystem interfaces
  - Subsystems are re-used in many SoSes

Evolution of the Architecture: Backwards Compatibility

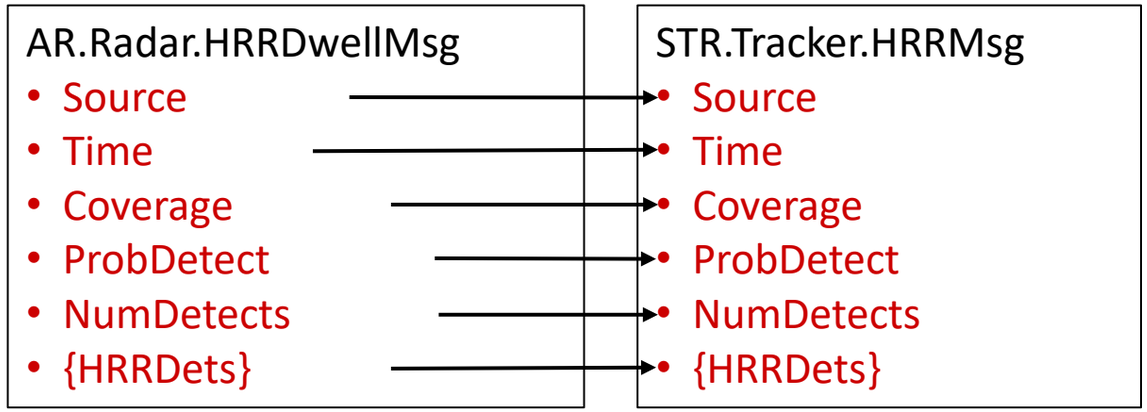
- Let's remember our simple 3 Subsystem ISR SoS
  - Radar: Produces AR.Radar.DwellMsg
  - Tracker: Consumes: STR.Tracker.SensorMsg; Produces: STR.Tracker.TrackMsg
  - Display: Consumes: G.Display.SensorMsg, G.Display.TrackMsg
  - Connections: Radar -> Tracker; Radar->Display; Tracker->Display
- Now let's add in an upgraded radar (Includes an HRR signature)
  - Construct transform that assigns each subfield from the new to the old message



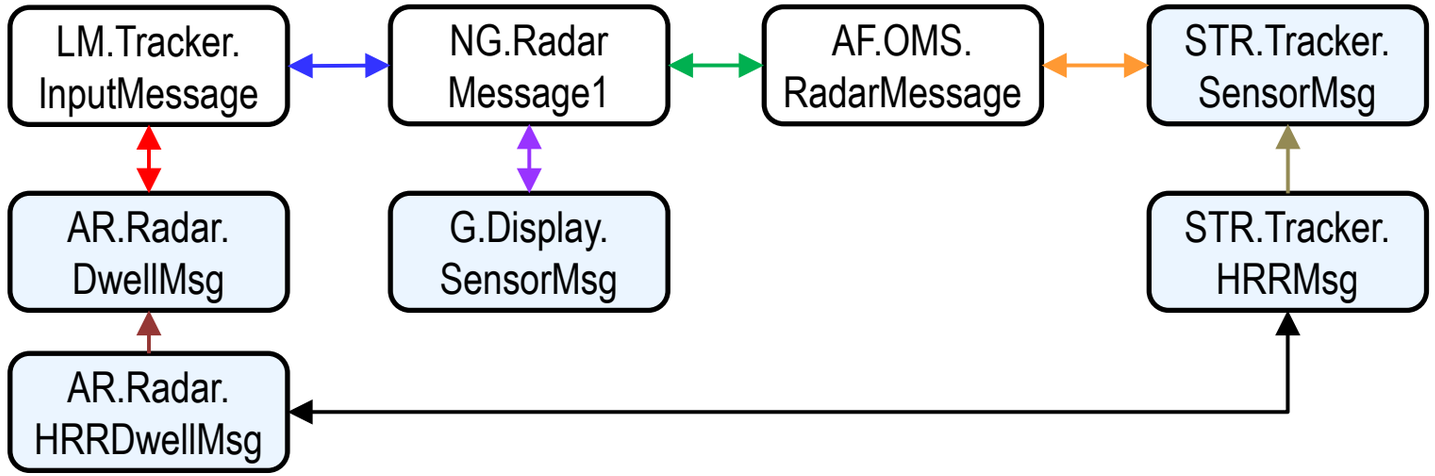
- Now can use the new radar anywhere that you can use the old radar
  (but won't get access to the HRR signal)

Evolution of the Architecture: Forwards Compatibility

- Now Let's Add in an Upgraded Tracker (That can use HRR Signatures)
  - Construct Transform That Assigns Each Subfield from the HRR Source to HRR Destination

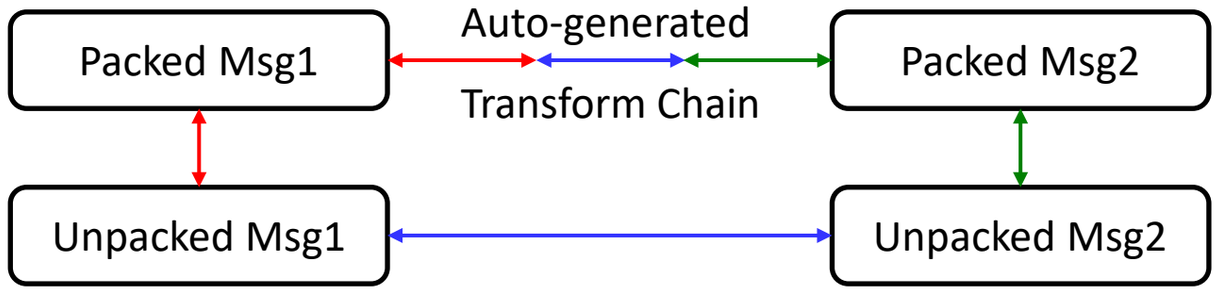


  - Now the New Tracker can use the HRR Signals from the new Radar



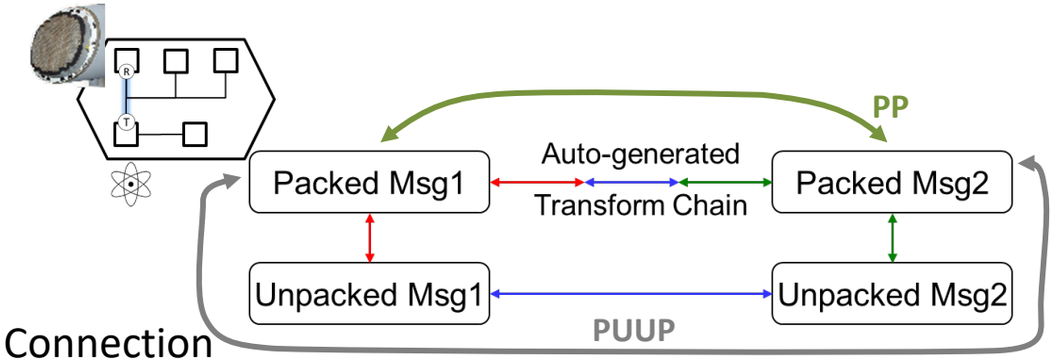
Handling Packed Representations

- Many real systems mix their interface definition with their implementation
  - Result is a serialized (Packed) form of the interface that can represent multiple different interface messages (e.g., STANAG 4607) with descriptor words for run time resolution
  - Packed messages are often used for run-time efficiency - they tend to be the big / high rate messages in the system. So don't want to unpack if not necessary
- Mirrored Unpacked Nodes Provide an Effective and Efficiency Solution
  - Create a Second Unpacked Node that Contains a Structured Version of the Interface
  - Create Transforms between the Unpacked and Packed Nodes
  - Interact with other Interfaces via their Unpacked Representations
  - Auto-generate the Desired (high performance) Packed-to-Packed Transforms



STITCHES Version 6R8.6 Optimized Performance:

[Packed → Unpacked → Unpacked → Packed] vs. [Packed → Packed]



Connection	PUUP vs PP	Speed (Mbps)			Latency (ms)		
		Java8	C++11	C90	Java8	C++11	C90
R1 -> T1 (No Transform)	PUUP	9899	13372	14705	0.6	0.3	0.2
R1 -> T1 (No Transform)	PP	9856	13186	14679	0.5	0.3	0.2
R1 -> T2 (Only Change Time)	PUUP	5555	13397	14481	0.8	0.3	0.2
R1 -> T2 (Only Change Time)	PP	5661	13308	14364	0.8	0.3	0.2
R1 -> T3 (Switch Order Lat, Lon)	PUUP	3216	5431	7128	1.0	0.4	0.3
R1 -> T3 (Switch Order Lat, Lon)	PP	3238	5519	6736	1.0	0.4	0.3
R1 -> T4 (Change All Fields)	PUUP	1595	3346	4137	1.7	0.5	0.4
R1 -> T4 (Change All Fields)	PP	2315	3202	4281	1.0	0.5	0.4

R: Radar Message
T#: Tracker interface #