



**Army Open System Demonstration (AMCOM)
Open Systems Project Engineering Conference (OSPEC)
FY 98 Status Review
29 April - 1 May 1998**

Bruce Lewis
US Army Aviation and Missile Command
Steve Vestal
Honeywell, Principal Investigator

Agenda



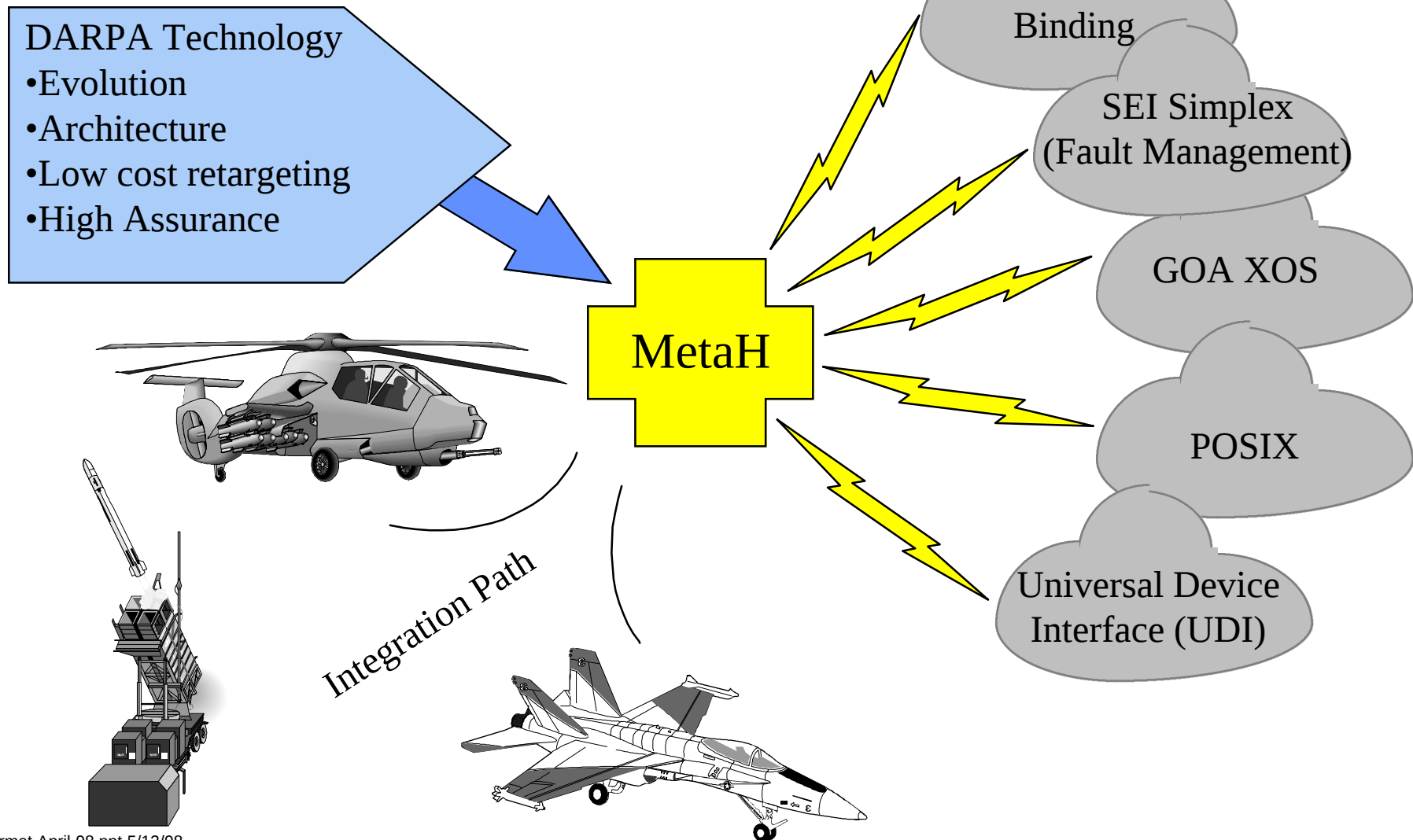
- **Project Description**
- **Goals**
- **Requirements**
- **Tasks/Technical Approach**
- **Results/Recommendations**
- **FY 98 Tasks/Schedule**

What Changes Are Needed to POSIX to Satisfy Missile and Aviation System Requirements? How Can We Know?

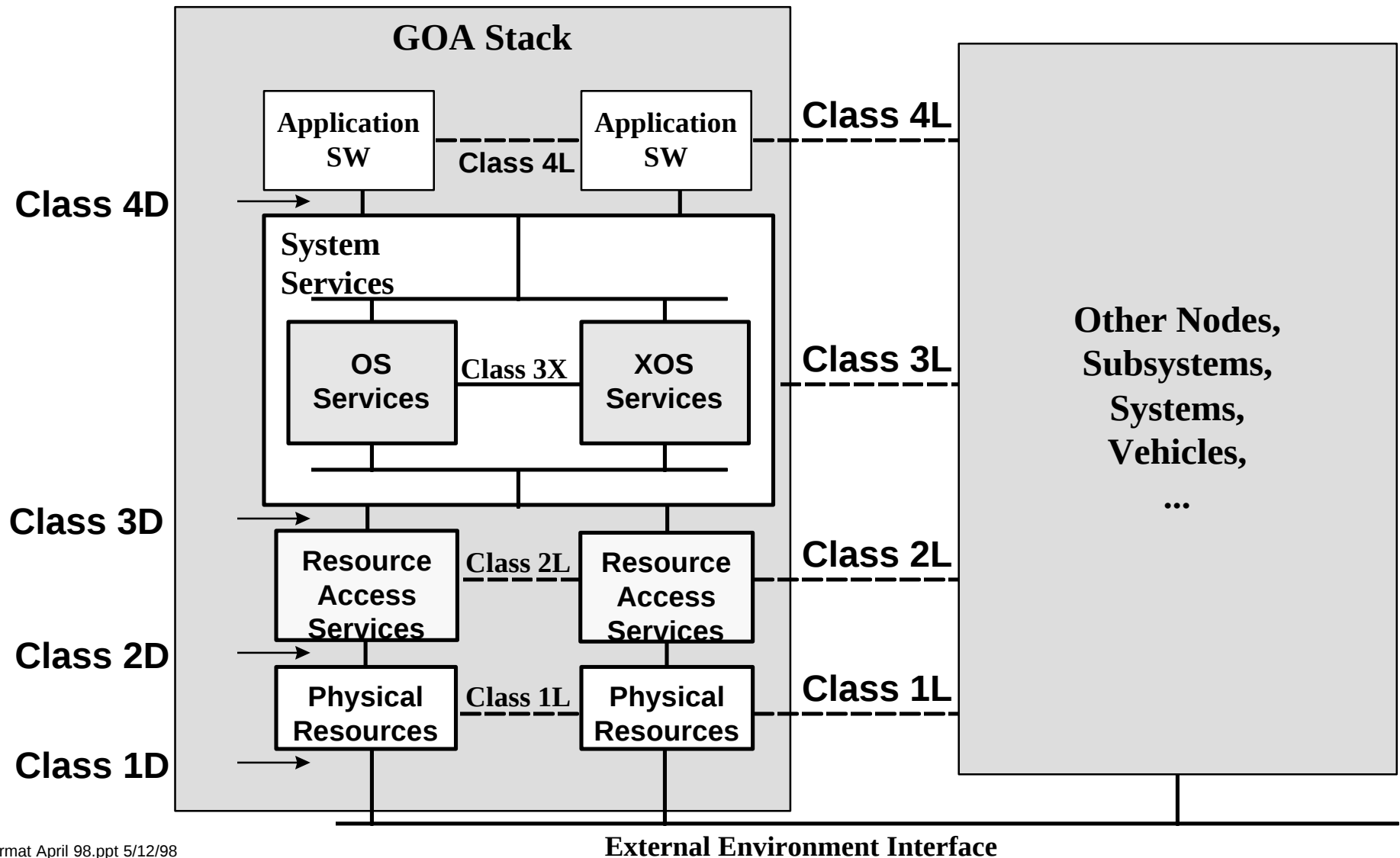


- **Test Many Different Systems by Implementing Them on Top of POSIX.**
 - Expensive!
 - Takes too long! or
- **Implement MetaH on Top of POSIX.**
 - MetaH satisfies a broad range of current and anticipated missile and aviation systems.
 - Cost is reasonable.
 - Will provide quantitative results for
 - performance.
 - adequacy of current POSIX features with recommendations for enhancements and/or new features.
 - Leverages POSIX into DARPA rapid development environment.

Synergism and Integration Path

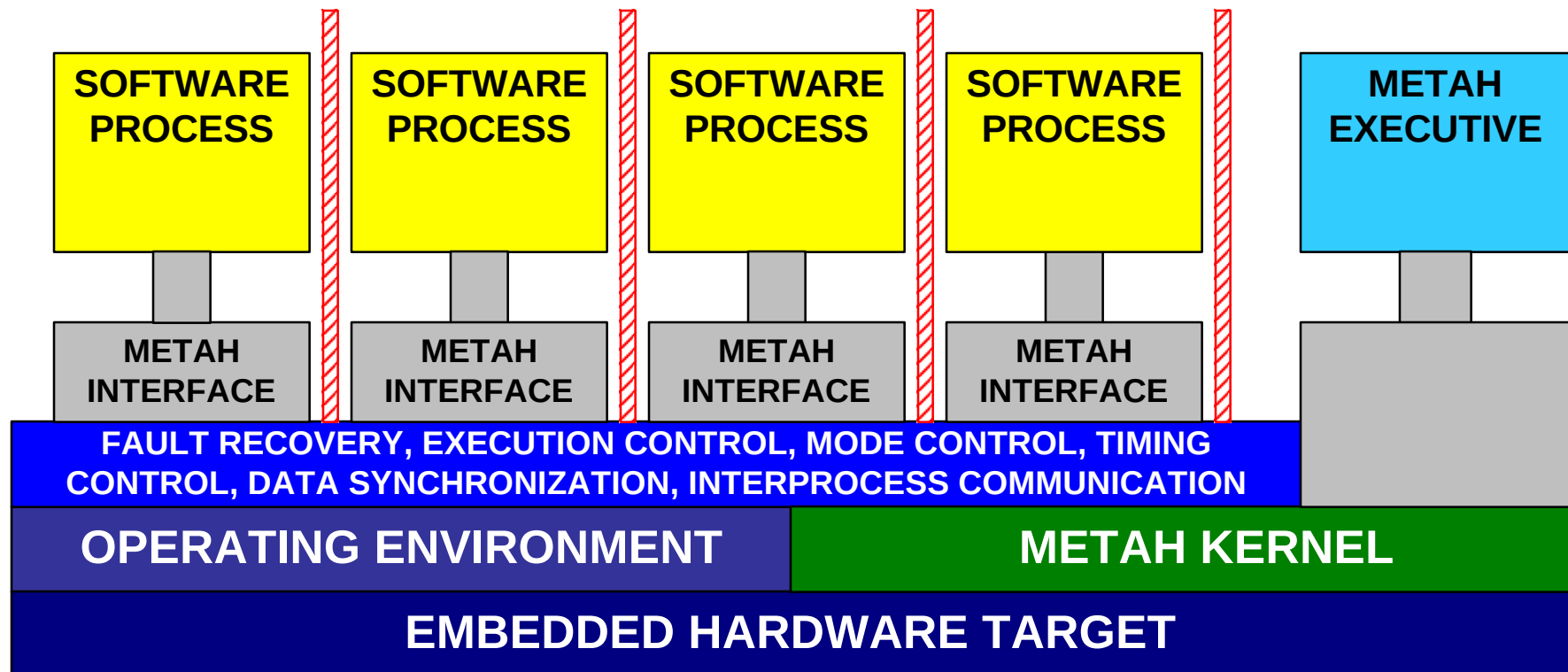


Society of Automotive Engineers Generic Open Architecture Stack





- Strong Partitioning - Not Just Memory Protection But Also ...
 - Timing Control
 - Process Control
 - Interface Control





- **Discover missing required capabilities for Avionics**
- **Reduce adoption barriers for MetaH and SAE/POSIX by working with programs, vendors to meet requirements**
- **Reflect advanced Avionics requirements into SAE/POSIX standards**
- **Demonstrate usability**
- **Leverage POSIX and related Standards into advanced DARPA Developments**
- **Start standardization of MetaH Architecture Description Language for Avionics**



Objectives

- Assess benefits of achievable portability using real-time POSIX
- Assess suitability of real-time POSIX for MetaH-produced missile and helicopter avionics software
- Identify possible POSIX and MetaH enhancements

Methodology / Tasks

- Survey POSIX documents and vendors
- Survey missile and helicopter program requirements
- Prototype a MetaH retarget to a real-time POSIX implementation

Missile and Helicopter Requirements



- **Functional**
- **Processor time and space**
- **Development process**

Missile and Helicopter Functional Requirements



- **Some functional requirements are outside POSIX and MetaH scope**
 - Initial hardware self-test
 - Memory management, e.g. ROM-to-RAM code copies
 - Low-level error management, e.g. lock-step pair restart
 - Hardware device interfaces
- **Missile and helicopter functional requirements were largely inferred from MetaH requirements**
 - MetaH is emerging technology
 - Incorporates methods widely-used in practice
 - Incorporates methods that anticipate future systems

Missile and Helicopter Time and Space Requirements



Some data obtained from

- OH-58D helicopter (Kiowa Warrior)
- Patriot Anti-Cruise missile (PACM)
- Theatre High-Altitude Air Defense (THAAD)
- Army Tactical Missile System (Army TACMS)
- Inertial Flight Measurement Unit (Honeywell IFMU)

Missile Profile

Multi-processor

Some Heterogeneous DSP+GPP

1MB memory/processor

500Hz high rate

Helicopter Profile

Multi-processor

Heterogeneous DSP/GPP

10MB memory/processor

50 Hz high rate

Some security

Missile and Helicopter Development Process Requirements

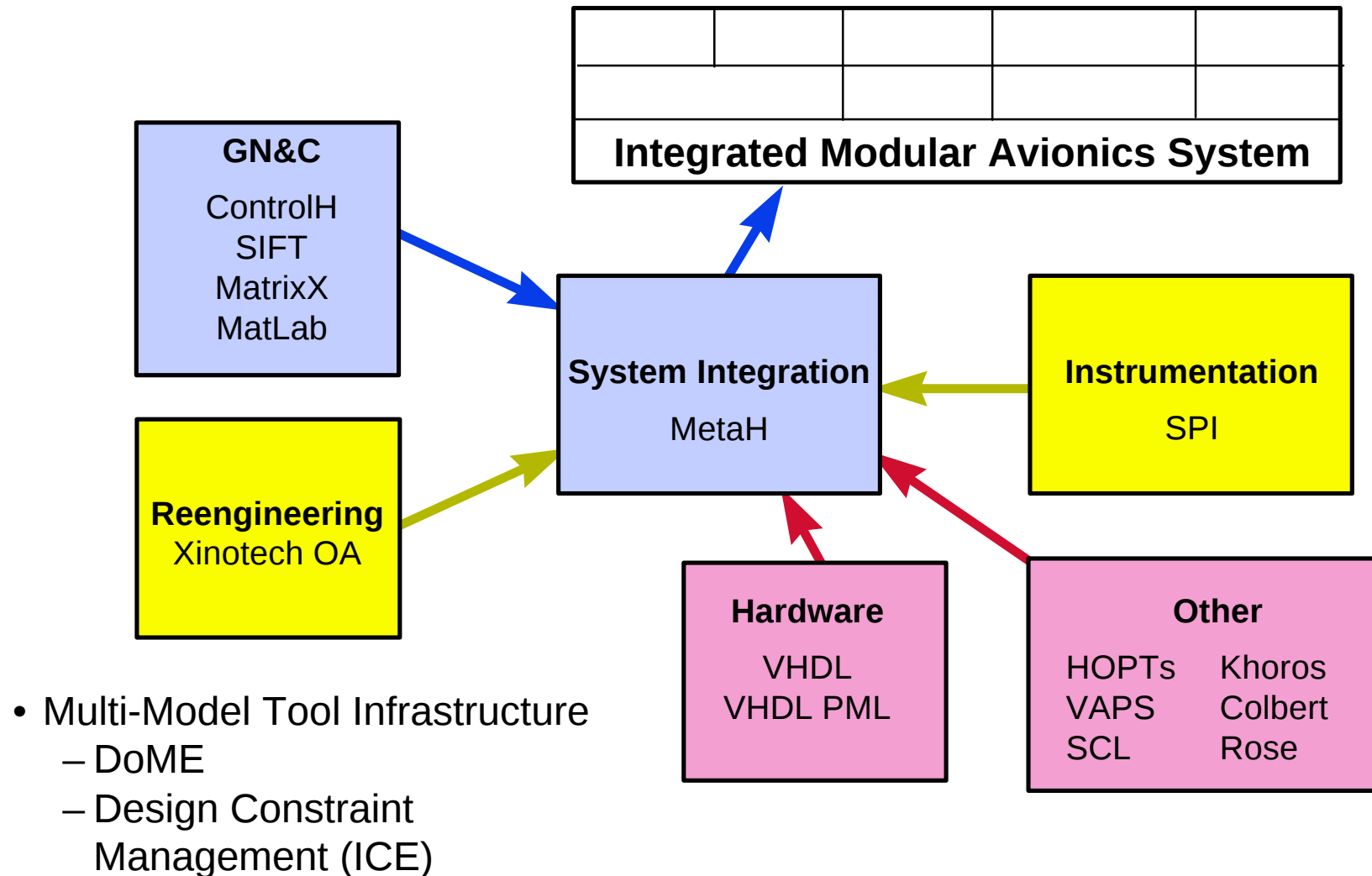


- **Verification**

- RTOS must be verified, too
- Partitioning

- **Multi-Platform Development**

- Workstation testbed flight system





Not a traditional CASE tool

specialized for hard real-time, fault-tolerant, safely/securely partitioned, scalable multi-processor systems

integration toolset with open interfaces to domain-specific generators, re-engineering tools, module libraries, etc.

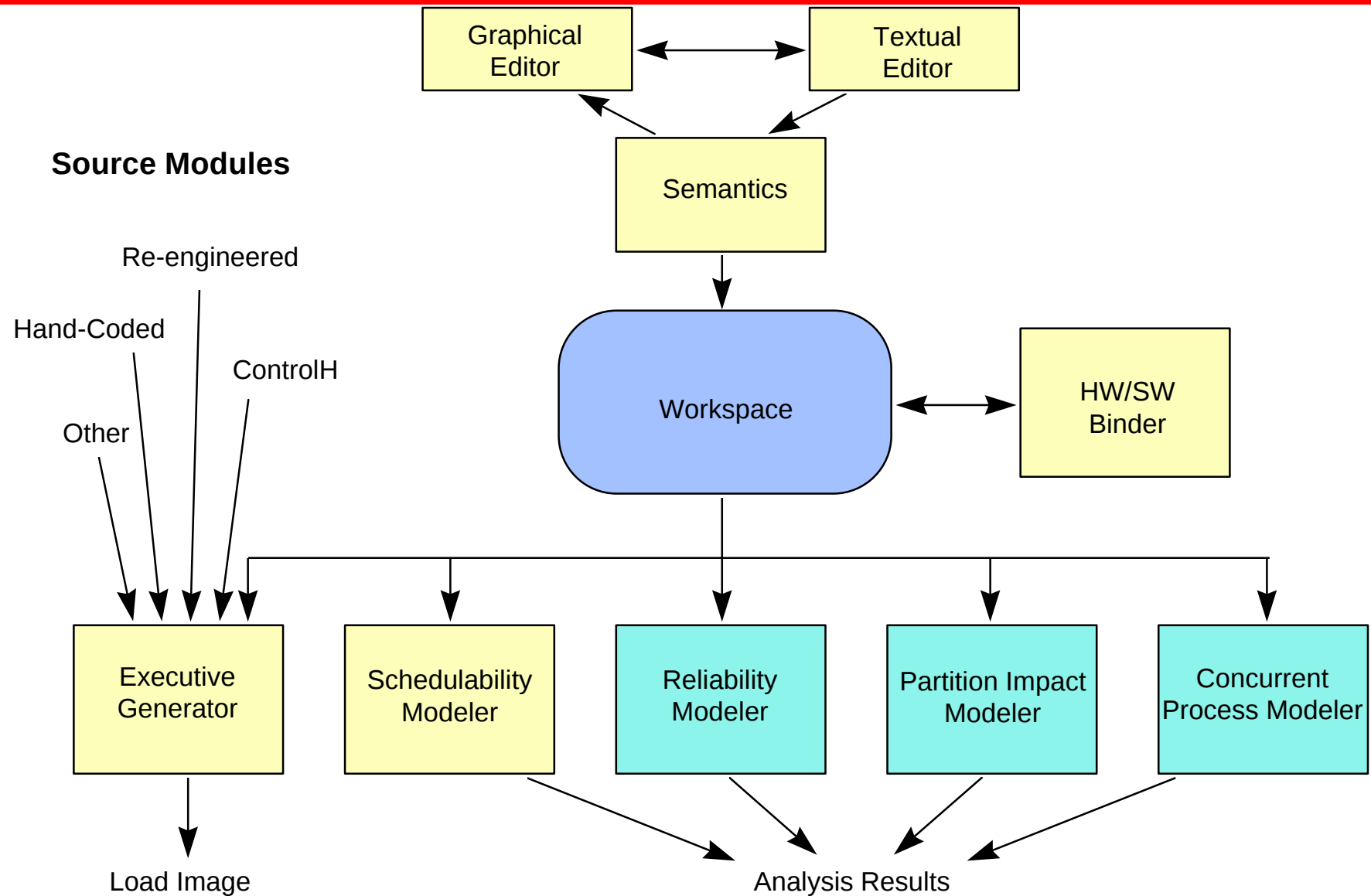
closely couples formal modeling and analysis with design and implementation

attentive to software/hardware interface, multi-processor systems, software/hardware configurability and protability

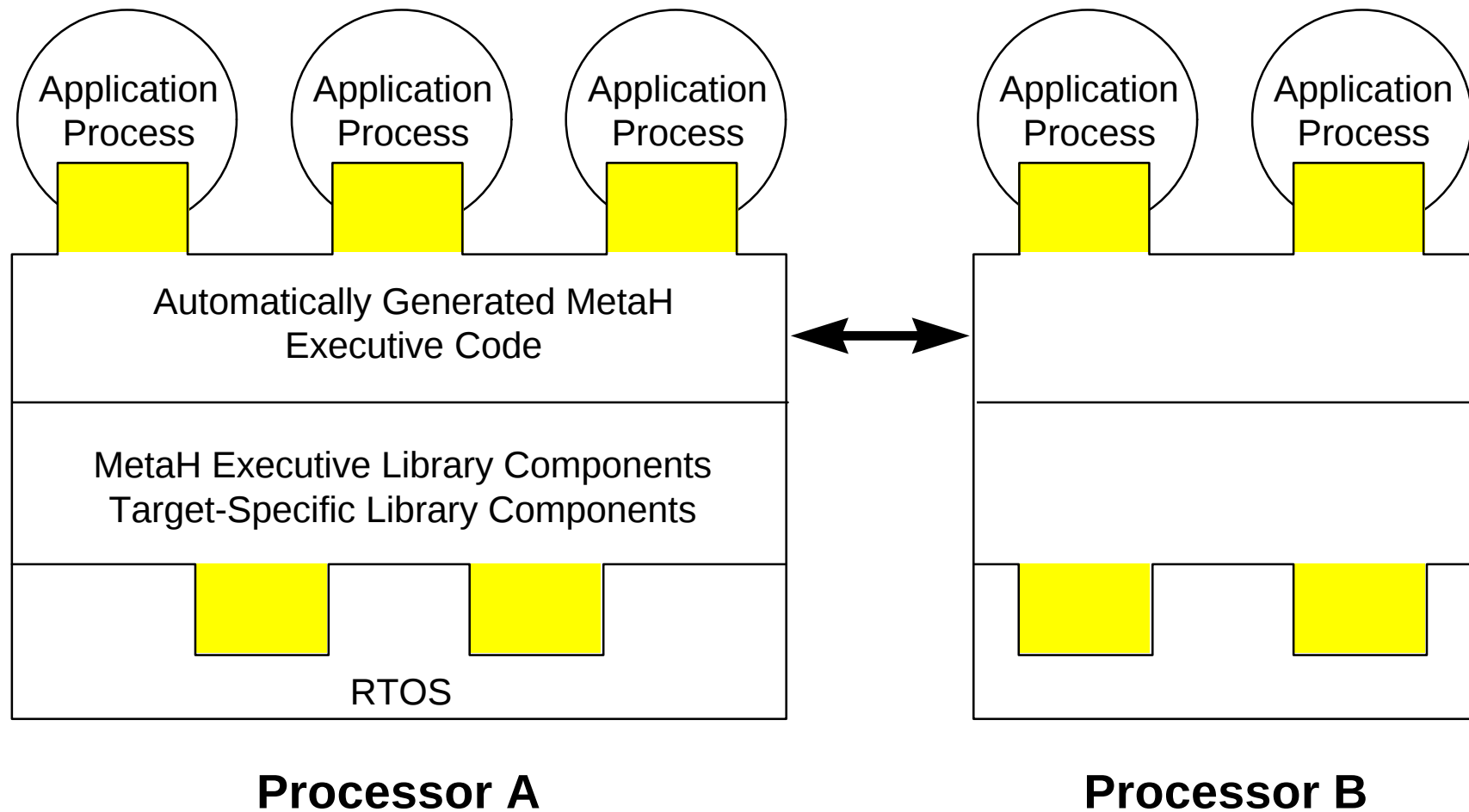
Not a traditional Real-time Operating System

designed to be retargeted to desired execution environment, including existing real-time run-times/kernels/operating systems

application-specific executive is tailored for each application, off-line configuration enables faster and smaller executives



MetaH Executive “Glue” Code





Prototype Technical Approach

- **Design Decisions**
 - Where does MetaH glue code/function go?
 - How do processes communicate with MetaH executive?
 - How is the MetaH process life cycle managed?
 - How are MetaH shared objects implemented?
 - How is MetaH message passing implemented?
 - How is synchronization supported?

- **Alternatives and Selections**
 - Try to support full MetaH
 - Try to use simplest, most direct choices

Summary of Effort



- **POSIX documents and tools reviewed, tools selected**
- **Prototype based on POSIX was partially functional**
- **Much of the prototyping effort was spent filling POSIX gaps and bugs (we filled in vendor's gaps using FSU Florist)**
- **About 70 objects from 11 POSIX packages referenced**
- **Recommendations presented to SAE and POSIX**



Portability Conclusions

- **Portability is limited by incomplete implementations**
- **There are situations where significant benefits are possible anyway**
 - **Proven architecture concepts**
 - **Some source portability**
 - **Portability from workstation to testbed to flight hardware**
 - **Ease processor upgrades**
- **Allocate your own resources to insure an appropriate implementation**



- **Neither current POSIX standards or commercially available implementations are likely to support fully partitioned MetaH off-the-shelf**
- **Partially partitioned MetaH on a reduced vendor-specific POSIX profile is possible and would meet requirements of many helicopter systems**
- **Unpartitioned MetaH with restricted aperiodic scheduling on a reduced vendor-specific POSIX profile or on Ada95 is possible and would meet requirements of many missile systems**



Desired POSIX enhancements

- Support for adaptive real-time scheduling
- Support for partitioning
- Support for extended executives

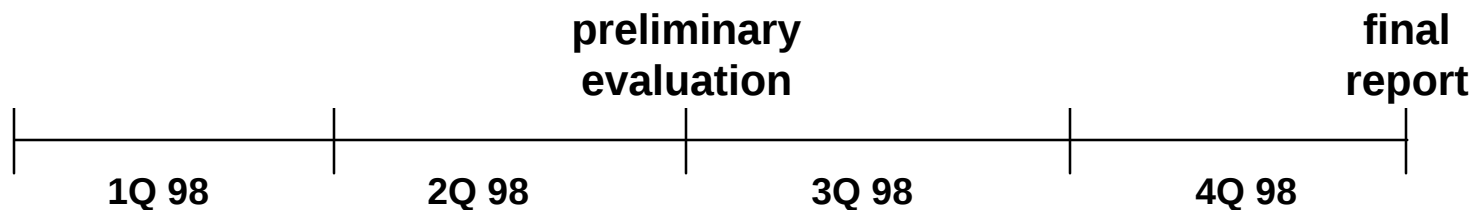
Desired MetaH enhancements

- Subsets for minimal RTOS configurations
- Improved event processing
- Standard IO and device interfaces



1998 Plans

- **Begin Metah standardization process**
- **Preliminary evaluation of the following activities, followed by selection and focus on one**
 - **Extend work on POSIX, MetaH and LynxOS to support and demo efficient fully partitioned and adaptively scheduled multi-processor systems**
 - **Prototype POSIX, MetaH and pAOS to support and demo efficient fully partitioned and adaptively scheduled multi-processor systems**
 - **Incorporate a UDI interface capability in MetaH and experiment with available devices and drivers.**





Additional Information

MetaH Scheduling and Allocation Features



- Periodic and aperiodic processes
- Preperiod deadlines and communication
- Process criticalities
- Multiple user-selectable real-time semaphore protocols
- Hard real-time multi-processor port-to-port communication
- Dynamic reconfiguration of processes and connections
- Processor and channel real-time schedulability analysis
- *Constraint specifications for software/hardware binding*
- *Process chaining for undelayed messages and ordering*
- *Slack scheduling of aperiodic, incremental, queue server processes*
- *Multiple subtasks within multiple threads*



MetaH Partitioning Features

- Processes are allocated individual protected address spaces
- Execution time limits can be specified and enforced
 - Elaboration time
 - Initialization time
 - Compute time
 - Semaphore locking time
- Process scheduling criticalities can be specified
- Process only has the run-time capabilities granted in the specification
- *Communication, data access and scheduling interference checked against specified safety/certification level and data rights attributes*



MetaH Fault-Tolerance Features

- Default behaviors for unhandled application process exceptions
- Process time limits and criticalities to handle timing faults
- Communication and semaphore protocols detect and report faults
- Plug-replaceable inter-processor executive consensus protocol
- *Error models and fault attributes allow specification and reliability modeling of redundancy management architectures*
- *Dynamic reconfiguration (mode changes) with processor restart*
- *Event consensus expressions for fault-tolerant mode changes*



- Retargetable to selected language toolset and RTOS
- Software/hardware interface features: hardware ports, hardware monitors, hardware events
- Processor and device specifications identify driver and interface source modules
- Channel specification used to connect processors in arbitrary topologies
- *Decrease retargeting effort through standard interfaces: Ada95, POSIX, UDI, . . .*
- *Extend distributed scheduling to handle low-bandwidth, high-latency channels, e.g. 1553, CAN, ARINC 659, . . .*

Where does MetaH Glue Code/Function Go?

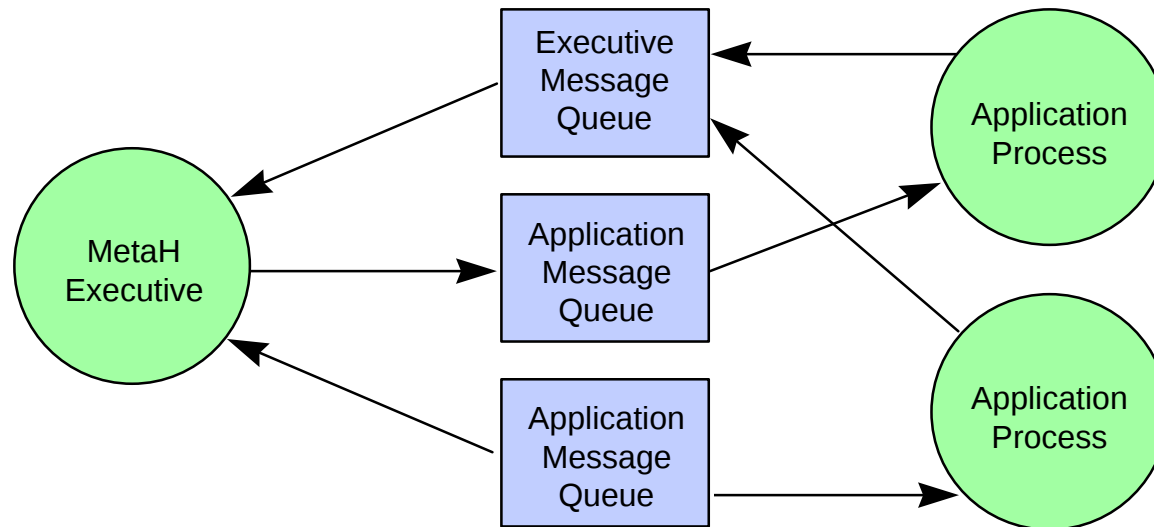


- Map all MetaH services to POSIX services
- Include executive code in application processes
- Add executive code to RTOS kernel
- **Executive code in a separate POSIX process**

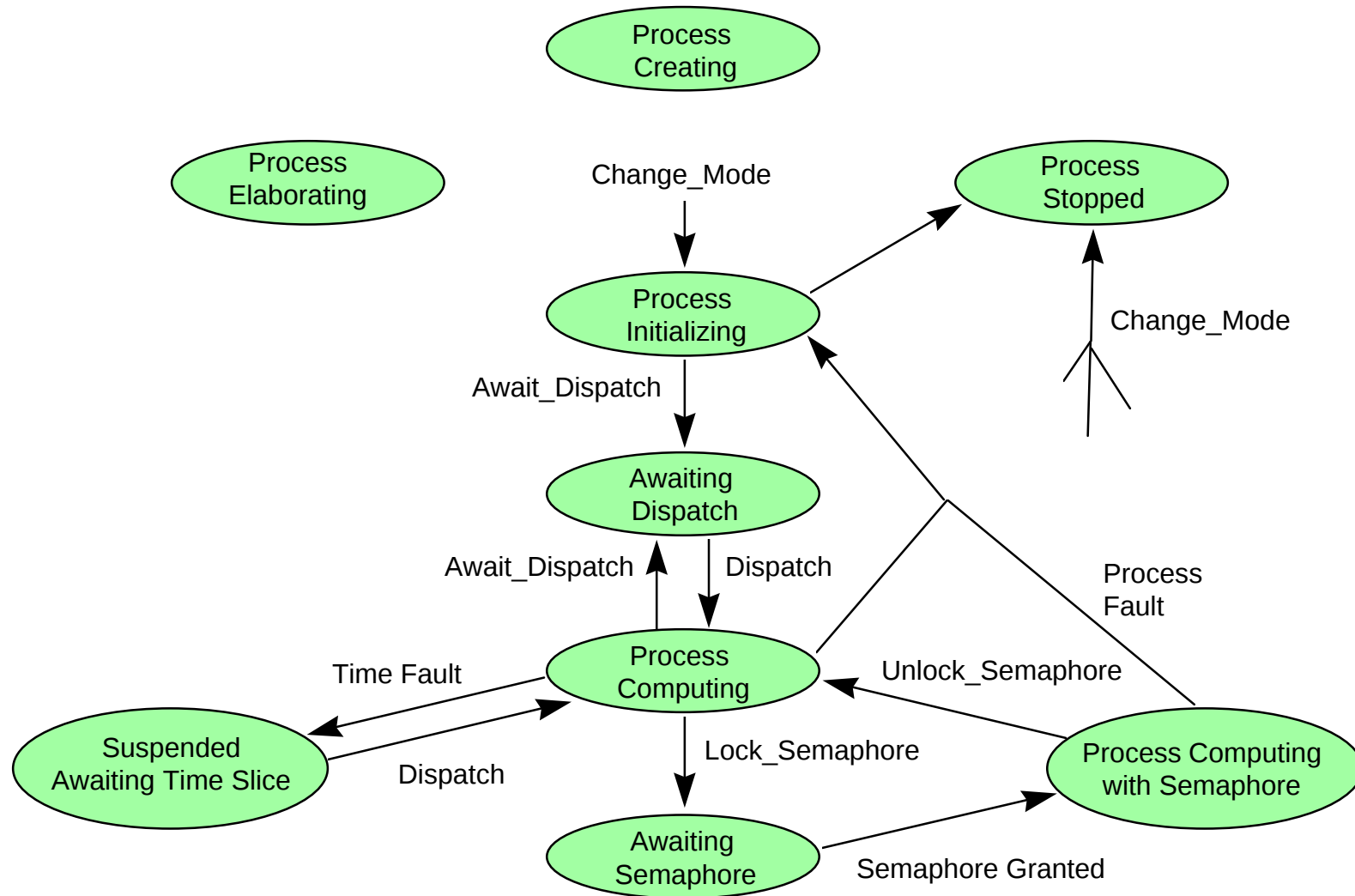
How do Processes Communicate with the MetaH Executive?



- Traps, inter-address-space procedure calls
- Shared memory for parameters and results
- **Messages for parameters and results**
- Semaphores for call/return synchronization
- **Signals for call/return synchronization**



MetaH Process Life Cycle



How is the MetaH Process Life Cycle Managed?



- Process restart service
- **Process start and terminate service**
- Generate “wrapper” for each process

How are MetaH Shared Objects Implemented?

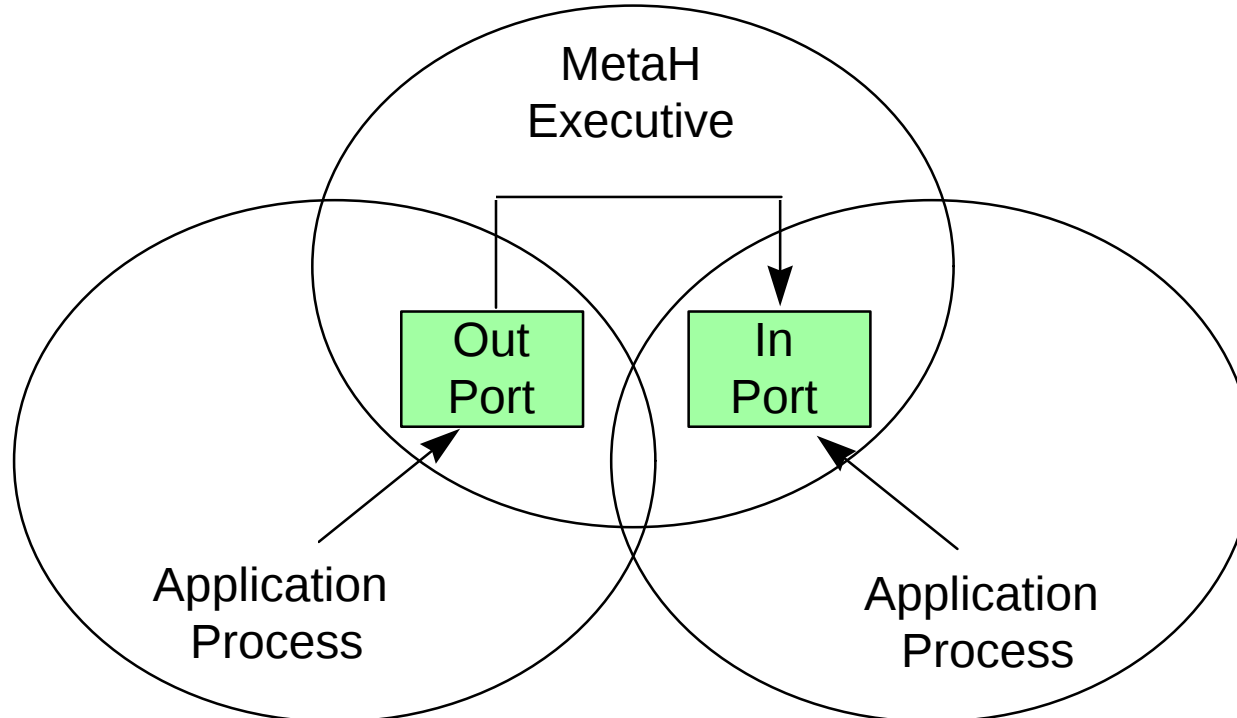


- Linker overlays
- Ada95 shared passive packages
- Shared memory objects and generated address clauses

How are MetaH Messages Implemented?



- POSIX message services
- **Assignments through shared memory objects**



How is Synchronization Supported?



- Applications directly use POSIX services
- **MetaH semaphores built using POSIX services**



Summary of Effort

- Total evaluation (including surveys and paperwork) about 4MM
- Partially functional prototype produced
 - Preprocessing, compile, link by hand
 - Ran some simple examples
 - Several features unimplemented
- Code referenced about 70 objects from 11 POSIX packages

POSIX

POSIX_IO

POSIX_MEMORY_LOCKING

POSIX_MEMORY_MAPPING

POSIX_MESSAGE_QUEUES

POSIX_PERMISSIONS

POSIX_PROCESS_IDENTIFICATION

POSIX_PROCESS_PRIMITIVES

POSIX_PROCESS_SCHEDULING

POSIX_SHARED_MEMORY_OBJECTS

POSIX_SIGNALS



Our Portability Experiences

- **Commercial products offer a subset of what we wanted**
- **We filled in gaps using Florist from FSU**
- **We spent a lot of time on POSIX interface implementation**



Portability Benefits

- **Common and proven architecture and design concepts**
- **Some source code portability**
- **Portability is always limited by application dependence**
- **Process portability: workstation testbed flight system**
- **Ease processor upgrades to reduce recurring hardware cost**



Possible POSIX Enhancements

- Control child priority at start
- Control child process thread priorities
- Child process restart (running and terminated)
- Execution time monitoring, limiting, stop/continue
- Restrict child calls that impact scheduling, memory allocation
- Semaphore semantics to handle fault while locked
- Rapid parent/child service request (e.g. passive parent)



Possible MetaH Enhancements

- **Additional Semaphore Capabilities**
 - Conditional variables
 - Mutexes
- **Unpartitioned MetaH**
 - Identical defect-free semantics
 - Identical RTOS interface, or at least identical MetaH- generated code
- **Simplified aperiodic protocols**
- **Additional event selection & queueing features**



Other Documents and Standards

- **Proposed POSIX additional realtime extensions**
- **Proposed POSIX application environment profiles**
- **Proposed realtime distributed communications**
- **SAE Generic Open Architecture (GOA) framework**
- **Uniform Driver Interface (UDI)**
- **Intelligent Input/Output (I₂O)**

Draft POSIX Additional Realtime Extensions



- **MetaH designed to easily add selectable semaphore protocols, can pass reader/writer and spin-lock capabilities on to application processes. Multi-processor protocols in particular offer complex trade-offs**
- **Interface to shared storage pools across multiple processors should be as compatible as possible with shared memory object interface, e.g. shared memory objects come from local storage pool**
- **MetaH needs system-wide periodic signal driveable from synchronized system clock**

Draft POSIX Application Environment Profiles



- **Minimal Realtime System Profile is a reasonable baseline for flight systems**
- **Desire process interfaces for upward compatibility, with suitably limited semantics**
- **Full processes with protected address spaces and time partitioning would be suitable for IMA systems**



- **MetaH would create endpoints and connections between executives on different processors at start-up**
- **Would like to send different object types (or at least subtypes) over connections**
- **Directory services shouldn't be essential for this**
- **Buffer management, configuration, and heterogeneous systems support useful (heterogeneous languages, too)**
- **MetaH requires a communications schedulability model**
- **MetaH manages end-to-end system scheduling and analysis, interface to a communications link is by timing message release and requiring a delivery deadline**
- **Priority is not directly meaningful for many types of communications link hardware**

SAE Generic Open Architecture Framework



- **MetaH executive is an eXtended Operating System (XOS)**

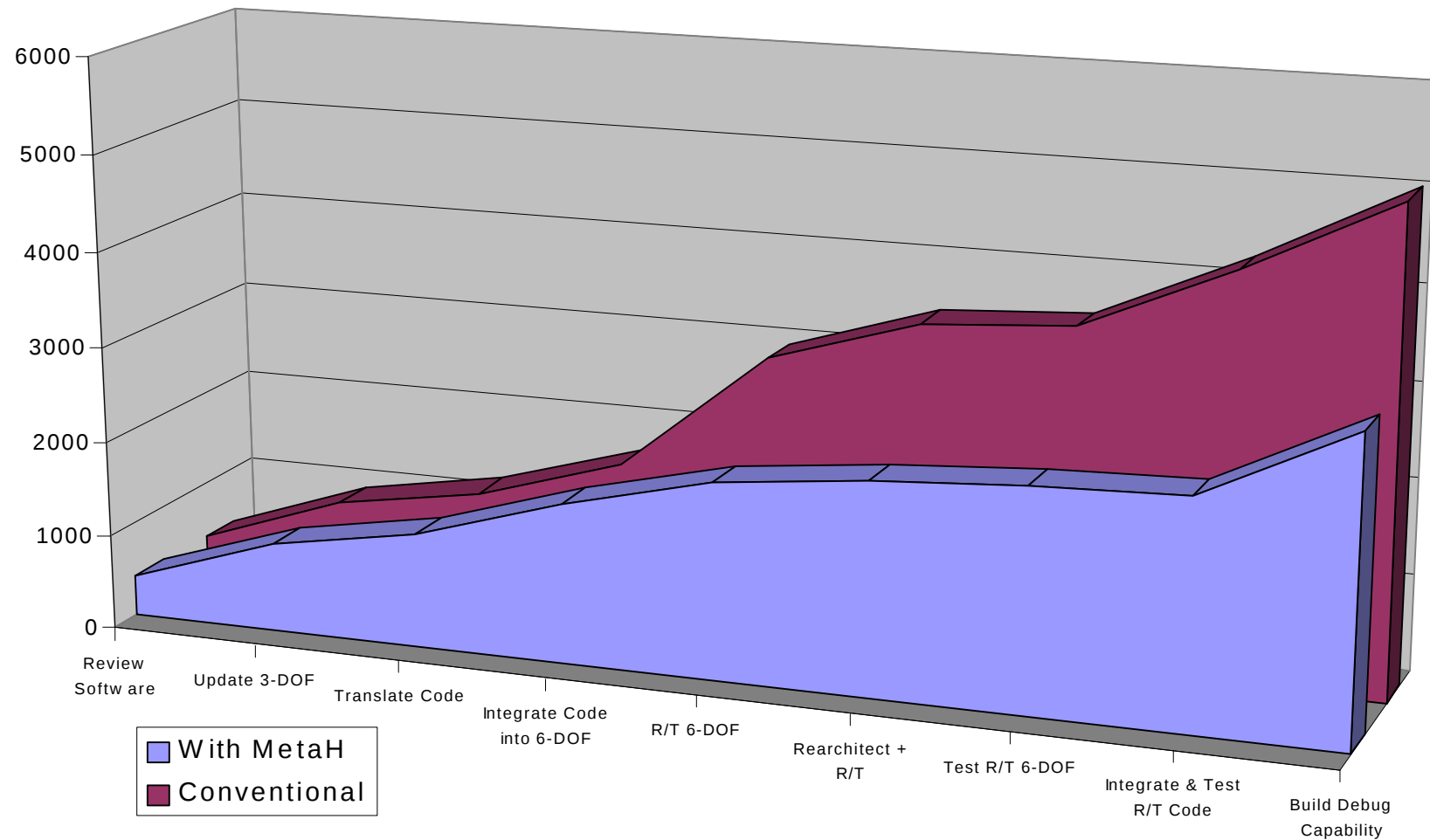
Uniform Driver Interface Intelligent Input/Output



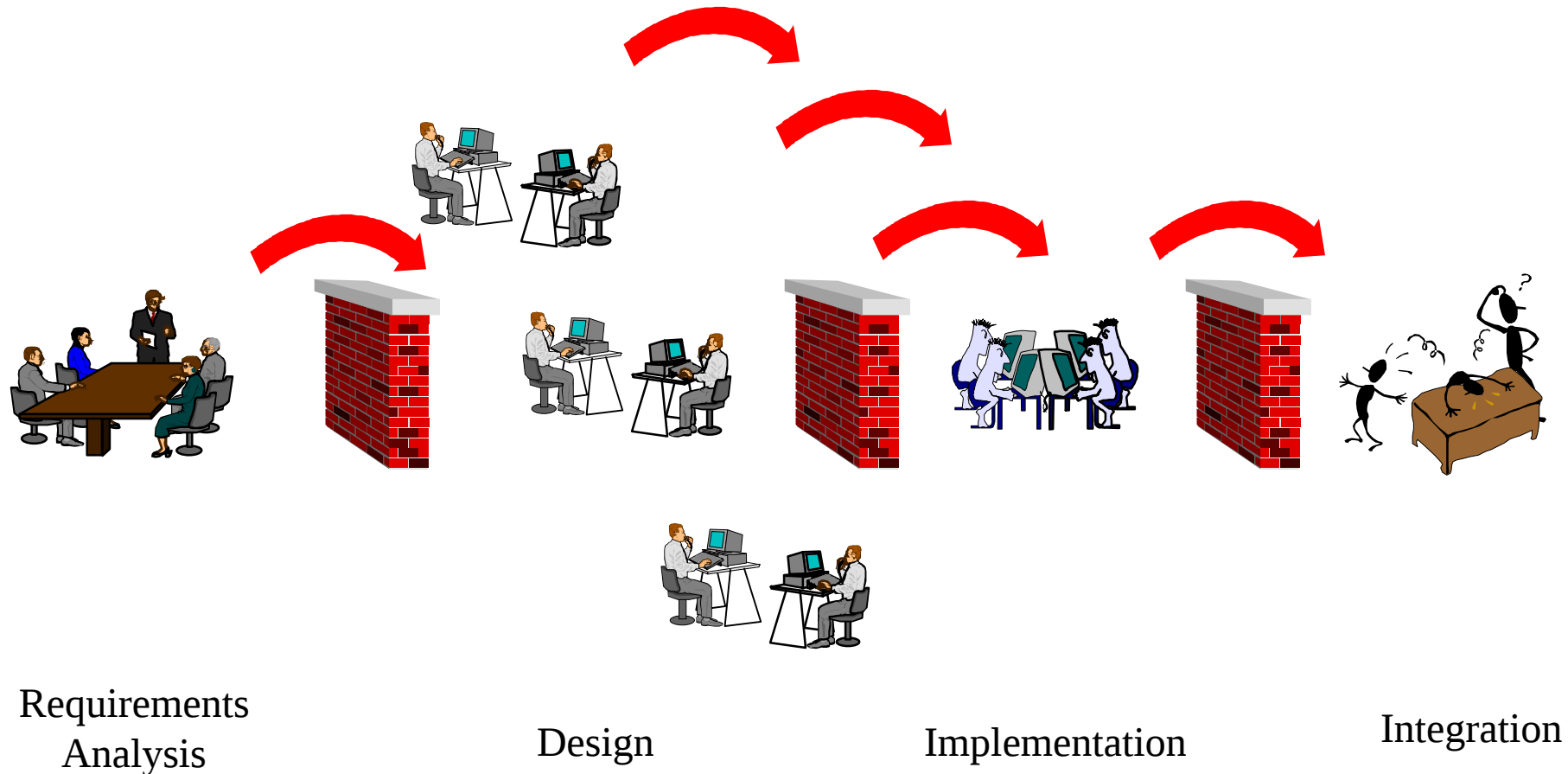
- **POSIX provides interfaces only for some classes of devices, e.g. mass storage, terminals**
- **MetaH would treat UDI modules like others that are being selected and composed into an application, but would need to support a UDI environment for them**
- **Some avionics system components are best treated as MetaH devices (IOPs) rather than MetaH processors, e.g. smart sensors, smart-head displays. Integration with an I₂O toolset/environment might be appropriate**

Effort Saved Using MetaH

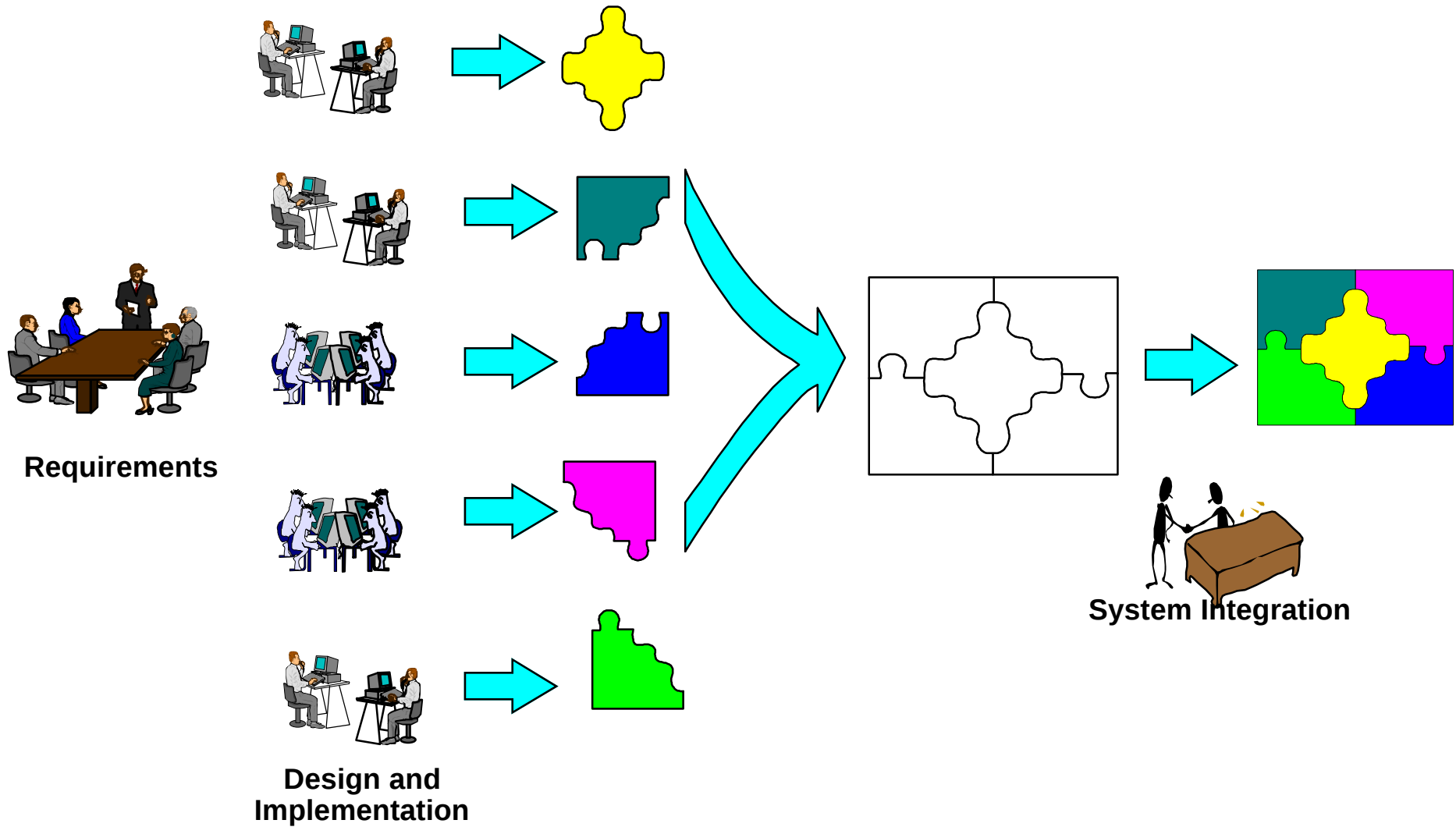
Estimated 37% reduction in Total Effort



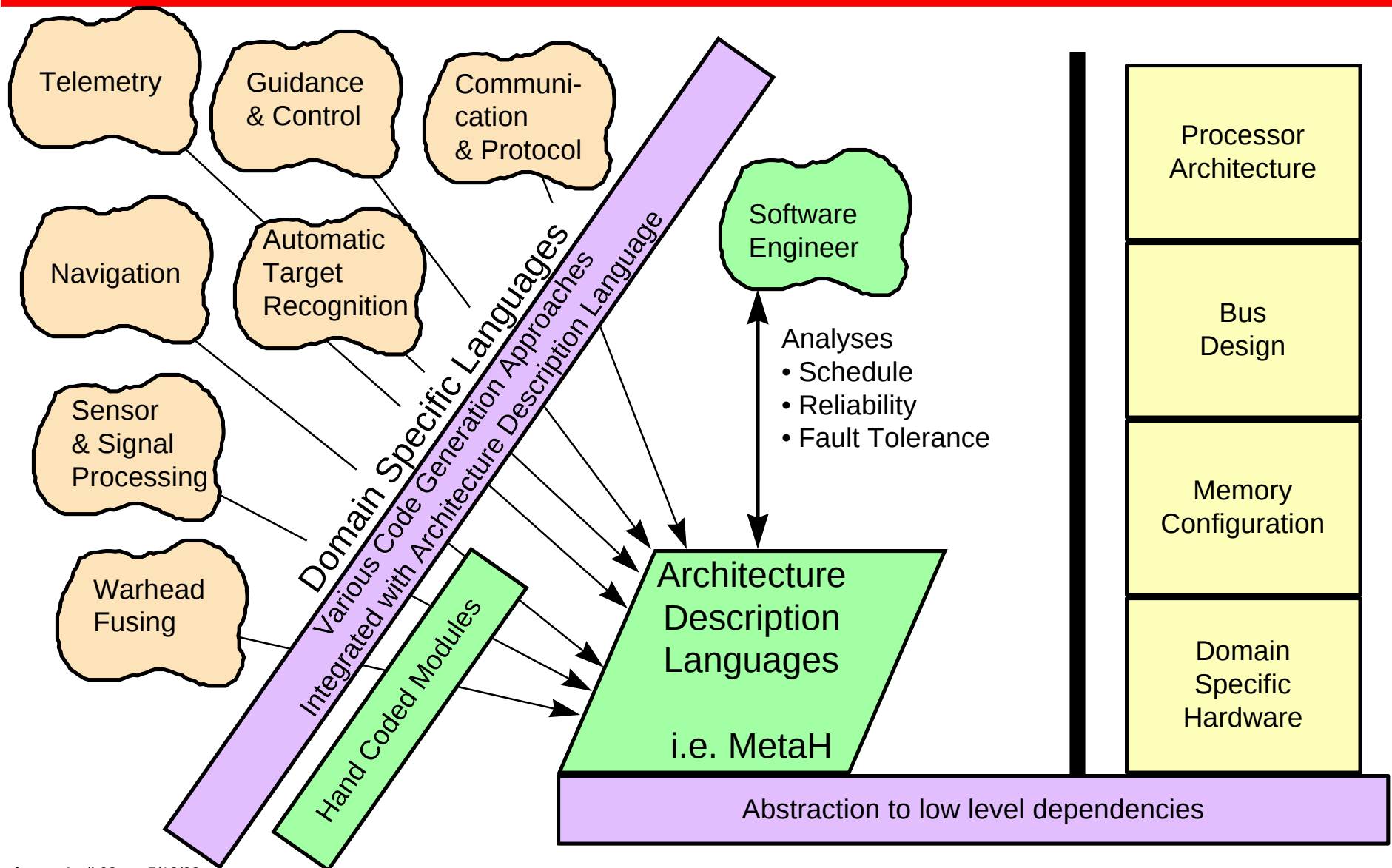
Complex Systems Engineering: A Multi-Disciplinary Engineering Process!



Architecture: The Missing Link in Engineering!



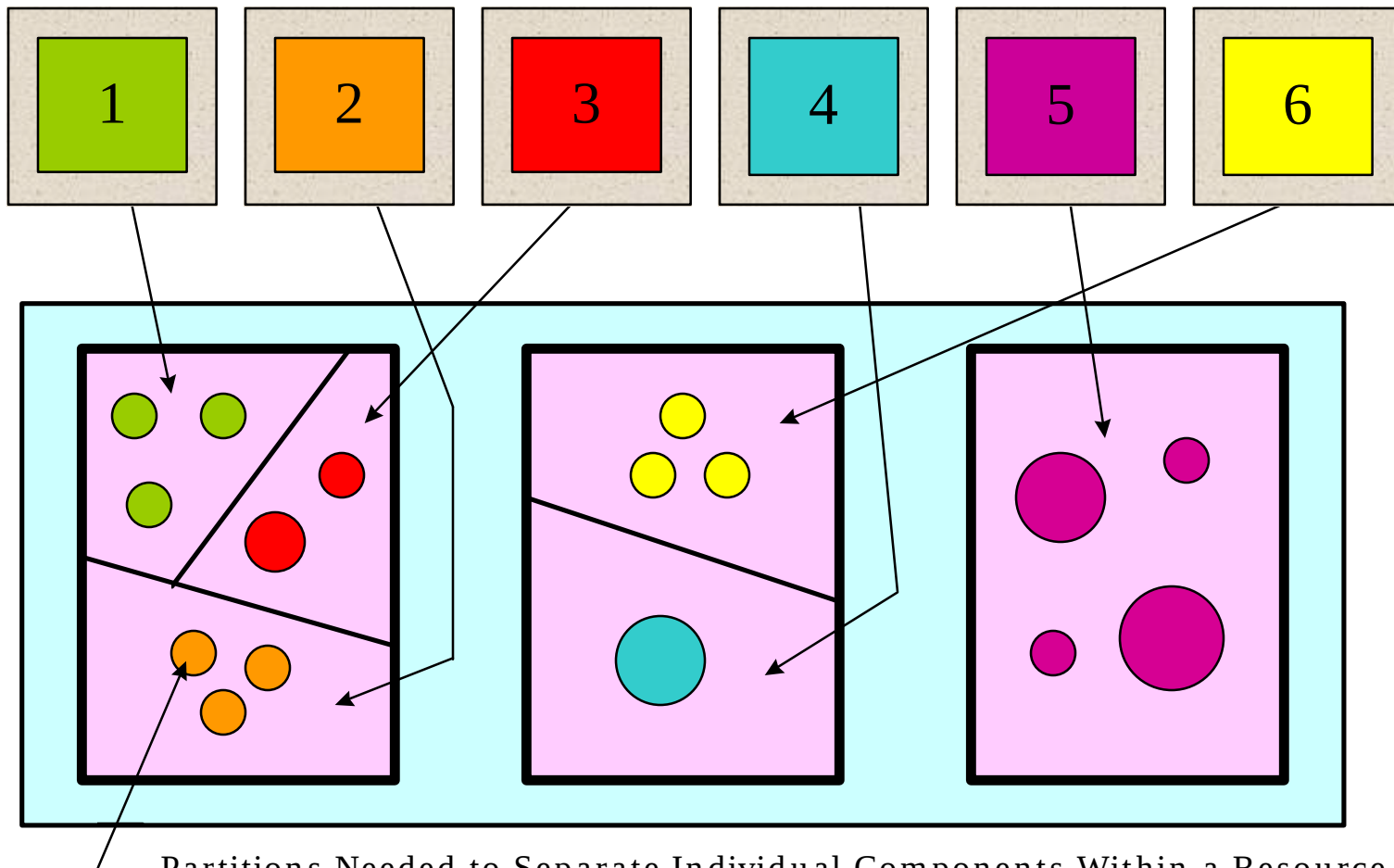
Paradigm for Architecture Based Software Development



Mapping to a Modular System



- Individual Units Map to Different Areas of Modular System's Resources



Partitions Needed to Separate Individual Components Within a Resource