Integrated Analysis of Temporal Behavior of Component-based Distributed Real-time Embedded Systems

Pranav Srinivas Kumar and Gabor Karsai
Vanderbilt University/ISIS

Supported by USAF/AFRL under Cooperative Agreement # FA8750-13-2-0050
Supported by DARPA under contract # NNA11AB14C

Introduction

- The need for design-time schedulability analysis and verification:
  - Hard Real-time Systems must meet operational deadlines, that constrain the amount of time permitted to elapse between a stimulus provided to the system and a response generated by the system
  - Delayed responses and missed deadlines can cause catastrophic effects on the function of the system
- Goal: Complete model-based development toolchain: modeling, analysis, synthesis, operations, maintenance

Problem Statement

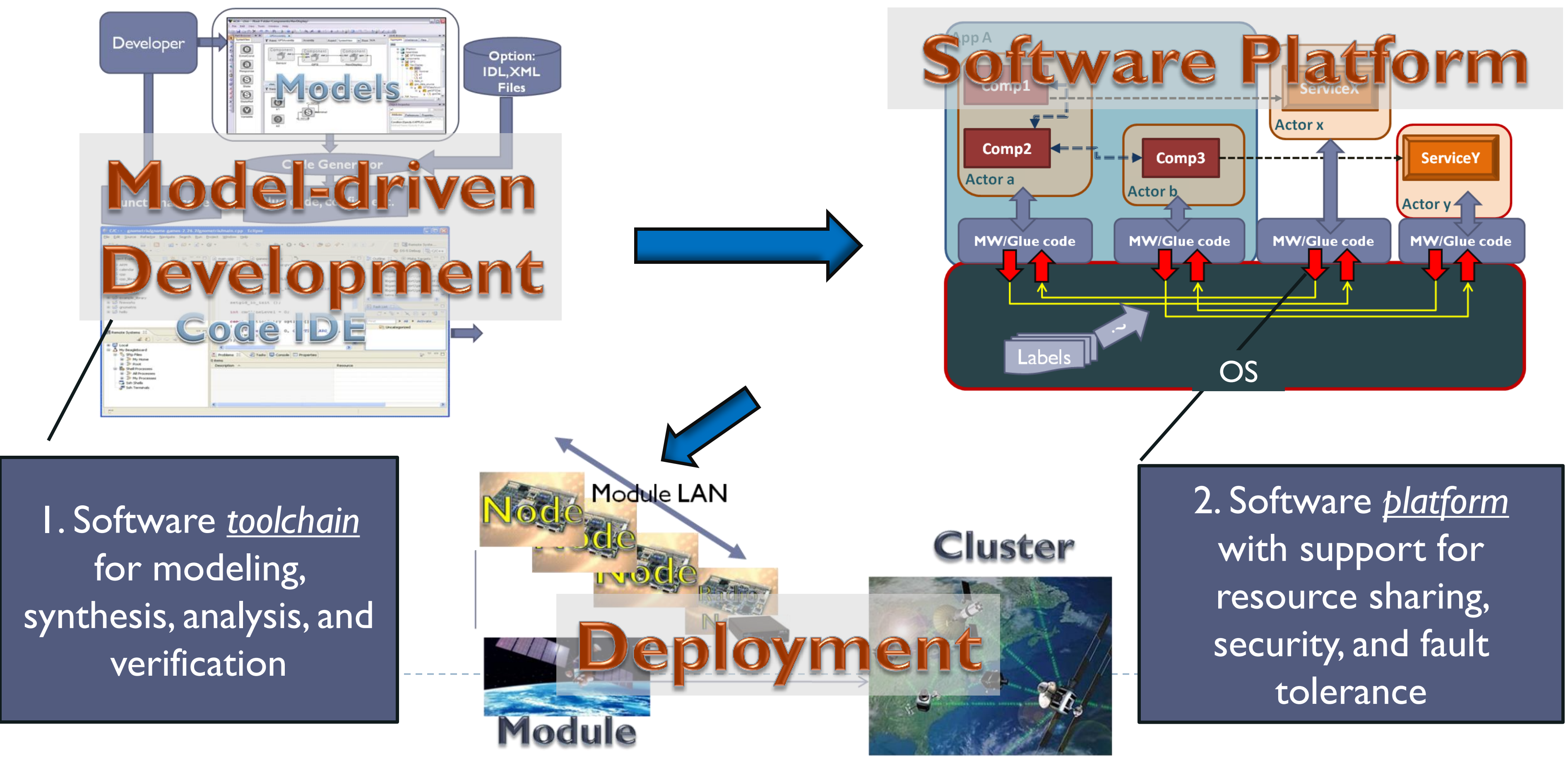
- Many of the existing schedulability analysis tools are not directly applicable to all system designs
  - Domain-specific properties such as arbitrary component interaction patterns, distributed deployment, and time-varying networks make this problematic
- The classic thread-based concurrency model is too low-level and too generic
  - Hard to analyze and use
- Restrictive, yet useful concurrency and component models are needed for which dedicated analysis tools *can* be developed

Problem Statement

- Mixed-criticality Component-based applications
- Distributed Deployment
- Each component exposes a set of interfaces to other components and to the underlying framework
- Different Component Interaction Patterns
- Hierarchical Scheduling
- Can we verify that no component operation misses its deadline?
- Can we verify that all timing requirements are always met?

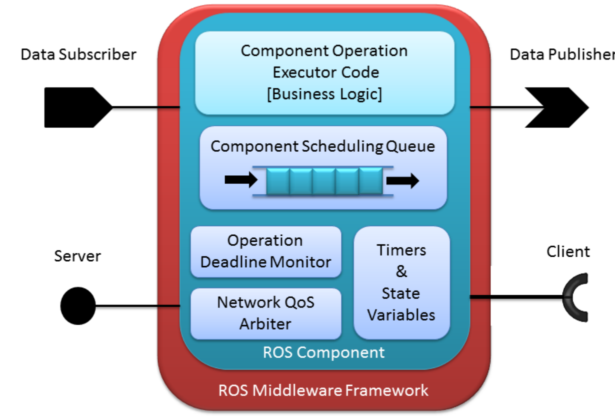
Target Architecture: DREMS

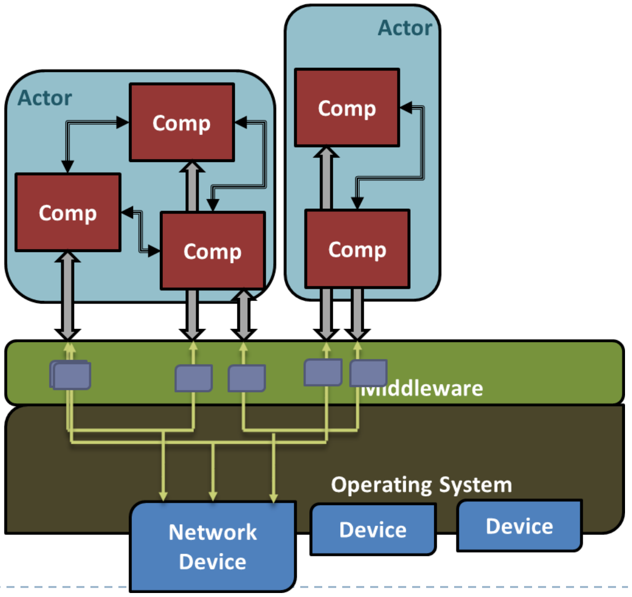
- **D**istributed **RE**altime **M**anaged **S**ystems [DREMS] consists of:
  1. A software *platform*, consisting of an OS and middleware
  2. A software *toolchain,* for modeling applications



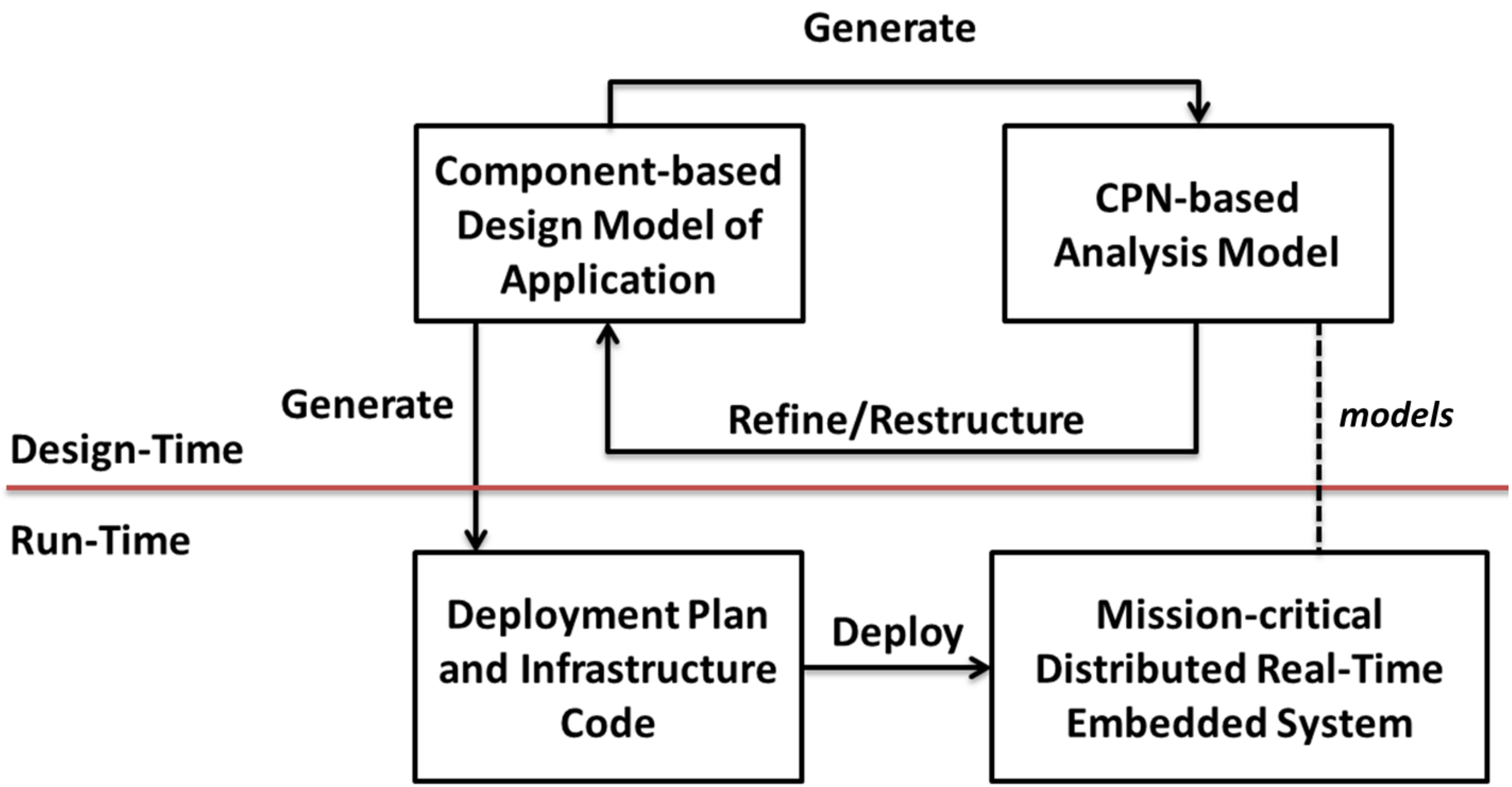
Target Architecture: DREMS

- Developed for a class of distributed real-time embedded systems
  - Remotely managed
  - Satisfy timing requirements
- Software Infrastructure
  - Design-time: Modeling and Analysis Tools
  - Run-time: Well-defined Component Model
  - Rapid Prototyping and Code Generation Features





Verification-driven Workflow



Contributions

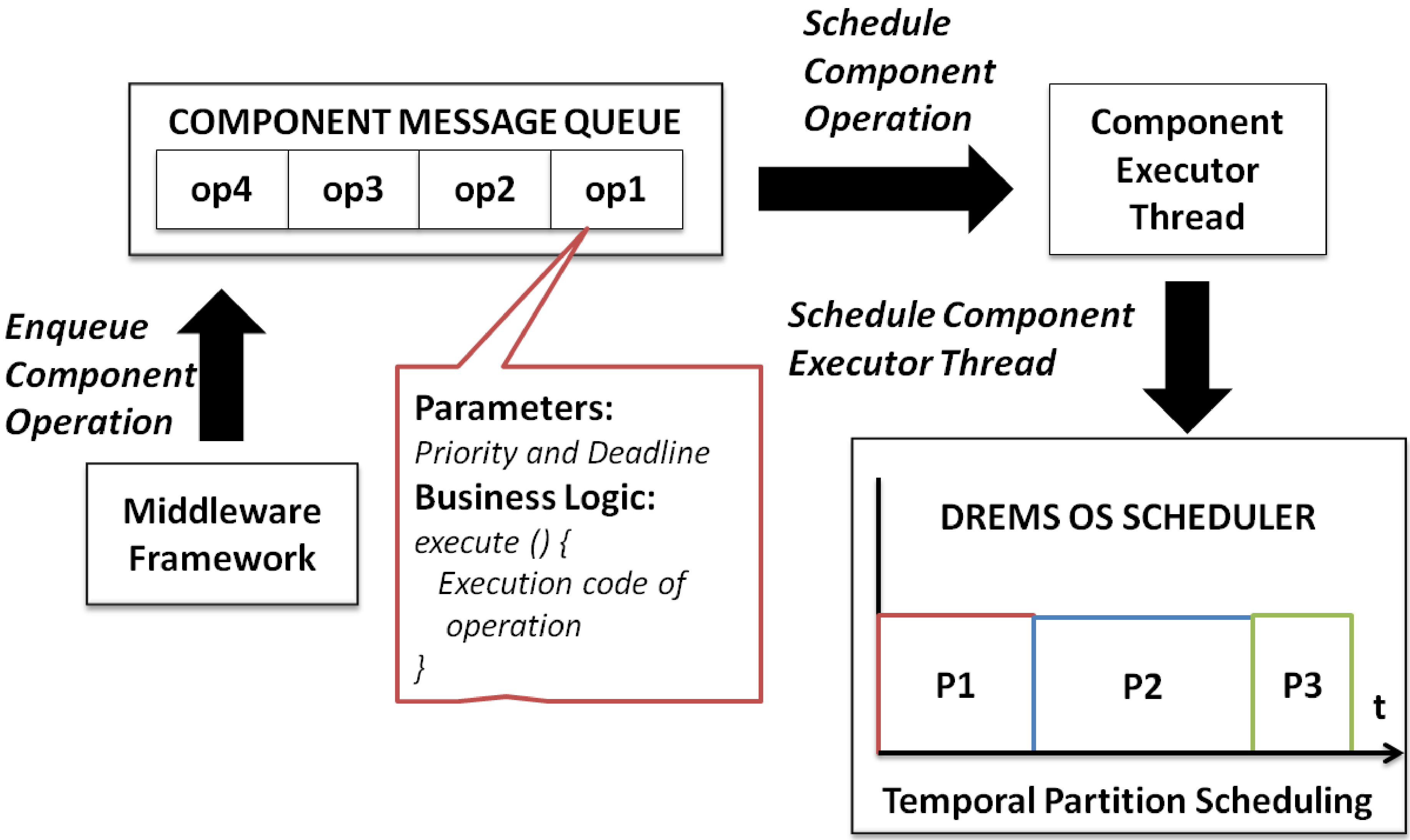
[Based on previous work: DREMS component model]

- A Colored Petri net-based approach to modeling and analyzing the structural behavioral properties of Component-based DRE Systems, such as DREMS
- An approach for modeling the operational behavior of each component in an application
  - The model uses a sequence of timed steps that are executed by the operations
- Improvements to a CPN-based modeling approach enabling better analysis performance and scalability
  - Relies on heuristics that manage time variables and state space data structures efficiently
- Advanced state space analysis techniques applied to reduce analysis time on medium-to-large systems

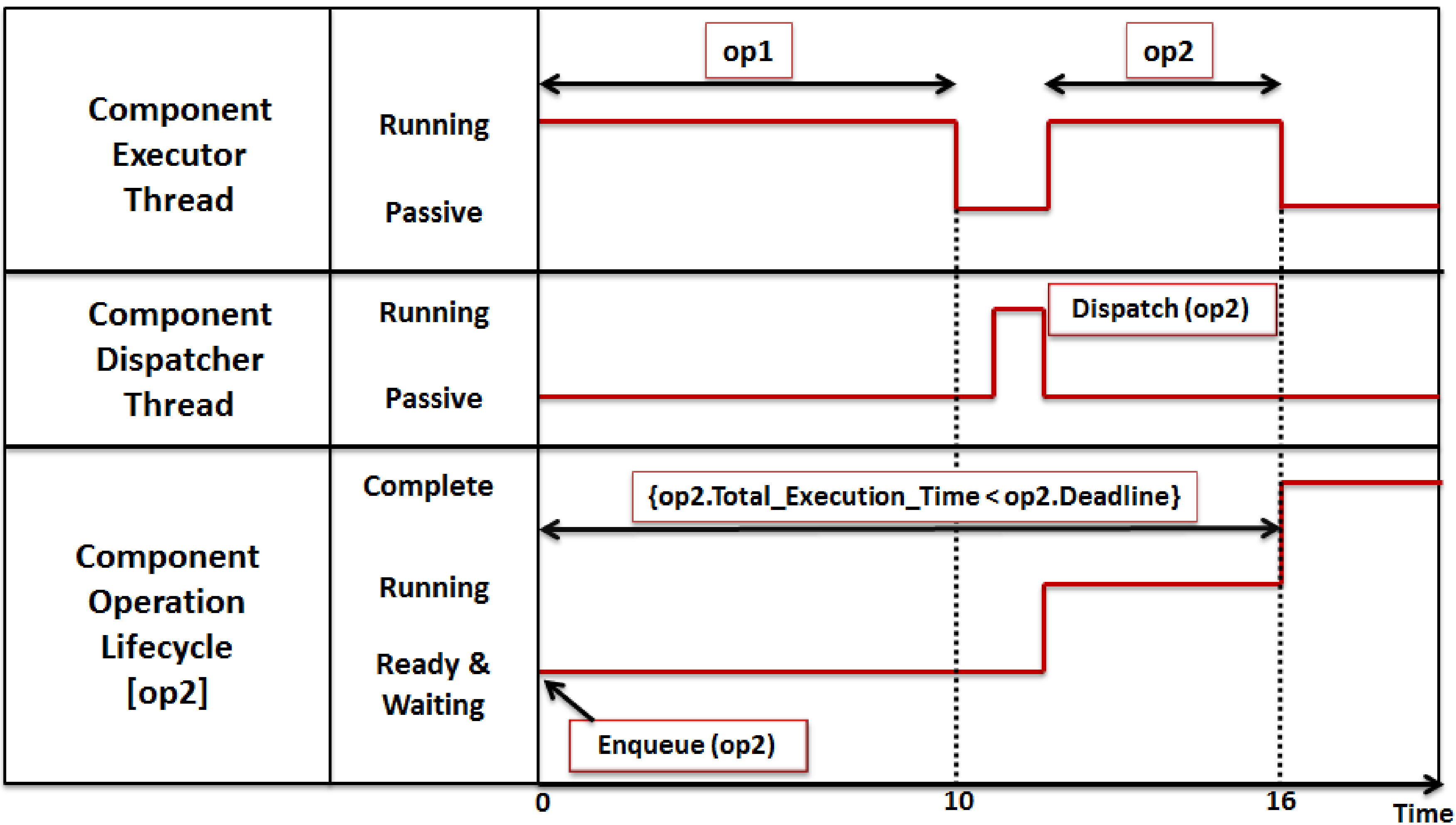
DREMS Background - Hierarchical Scheduling

- Component level
  - DREMS Applications are built by assembling *Components*
  - Component interfaces expose *operations* that can be invoked
  - Components interact via RMI/AMI and pub/sub interactions
  - Operation requests are handled using a per-component *message queue*
  - Scheduling Policy for requests: Non-preemptive Priority FIFO
    - EDF and FIFO policies are planned
  - Single executor thread per component handles requests
- Operating System level
  - Components are grouped into processes
  - Processes are assigned to ARINC-653 style temporal partitions: fixed length, periodic intervals of the CPU's time

Component Operations



Component Operations

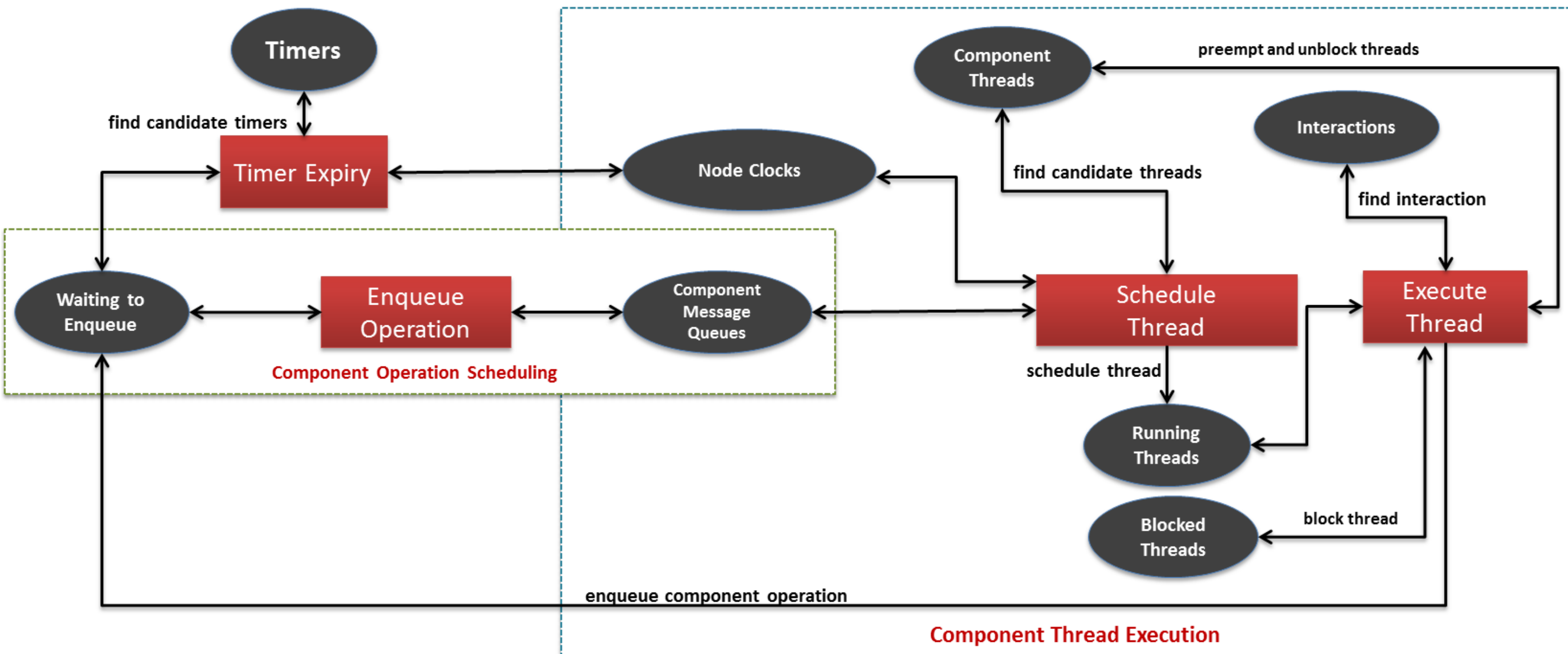


Colored Petri Nets

- Graphical Modeling/Analysis Tool [cpntools.org]
- Tokens contain values of data types called *colors*
- Powerful modeling concepts are facilitated by token colors
  - Heterogeneous data structures such as records with arbitrary number of fields
  - Tokens can be inspected, modified and manipulated by transitions and arc bindings
  - Component properties such as thread priority, port connections and real-time requirements are encoded into a single color token

CPN Analysis Model

- A completed generated Colored Petri Net timing analysis model for a Domain-specific modeling language



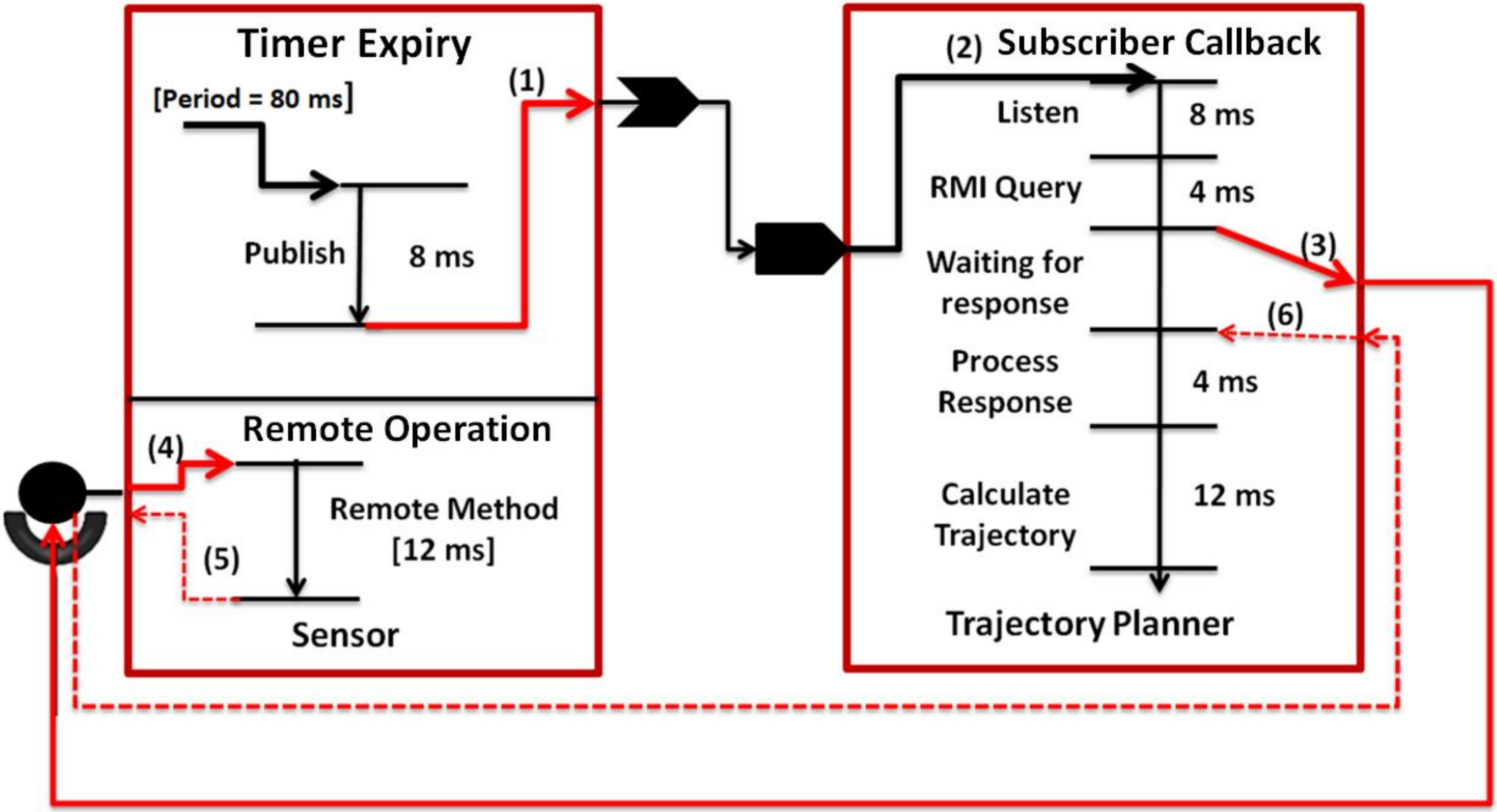
Modeling Temporal Behavior

(* **Business Logic syntax in Extended Backus-Naur Form** *)

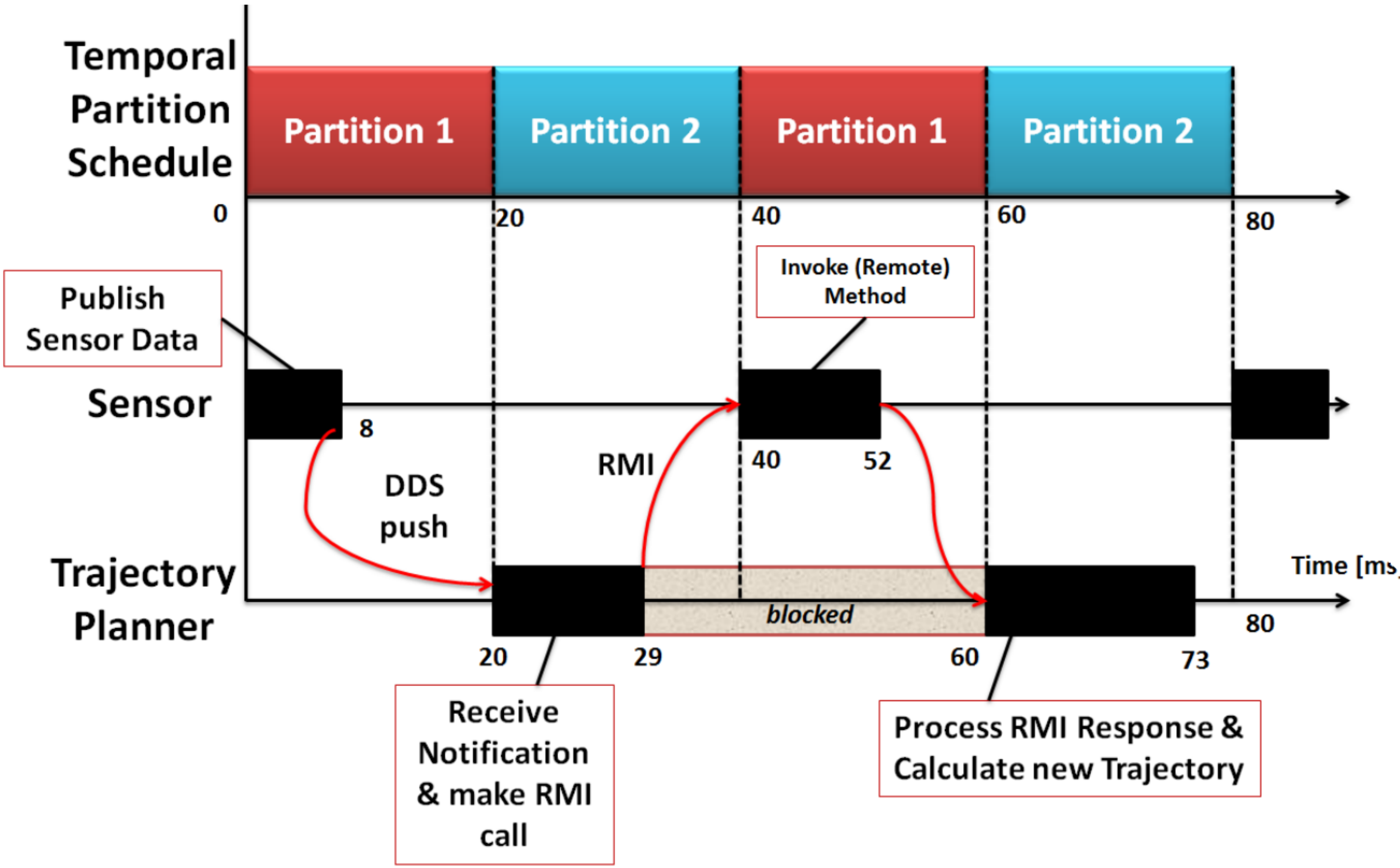
business_logic	=	‘**Do**’, ws, operation_name, ws ‘**[**’, operation_priority, operation_deadline, ‘**]**’, ‘**{**‘, { functional_step }, ‘**};**’ ;
operation_name	=	ID ;
operation_priority	=	INT ;
operation_deadline	=	INT ;
functional_step	=	{sequential_code_block | rmi_call | ami_call | dds_publish | dds_pull_subscribe | dds_push_subscribe | loop} ;
sequential_code_block	=	INT, ‘**;**’ ;
rmi call	=	‘**RMI**’, ws, receptacle_port, ‘**.**’, remote_operation, ‘**[**’ query_time, processing_time ‘**];**’ ;
ami call	=	‘**AMI**’, ws, receptacle_port, ‘**.**’, remote_operation, ‘**[**’ query_time, processing_time ‘**];**’ ;
dds publish	=	‘**DDS_Publish**’, ws, dds_port, ‘**.**’, topic, ‘**[**’, publish_time, ‘**];**’ ;
dds pull subscribe	=	‘**DDS_Pull_Subscribe**’, ws, dds_port, '**.**‘, topic, '**[**‘, processing_time, '**];**’ ;
dds_push_subscribe	=	‘**DDS_Push_Subscribe**’, ws, dds_port, ‘**.**’, topic, ‘**[**’ processing_time, ‘**];**’ ;
loop	=	‘**LOOP**’, ws, ‘**[**’, count, ‘**]**’, ws, ‘**{**‘, {functional_step}, ‘**};**’ ;
receptacle_port	=	ID ;
remote_operation	=	ID ;
dds_port	=	ID ;
topic	=	ID;
query_time	=	INT ;
processing_time	=	INT ;
publish_time	=	INT ;
count	=	INT;

A simple textual language for representing the temporal behavior of a component operation. Fixed, worst-case behavior (no data dependency)

Trajectory Planner (1/2)



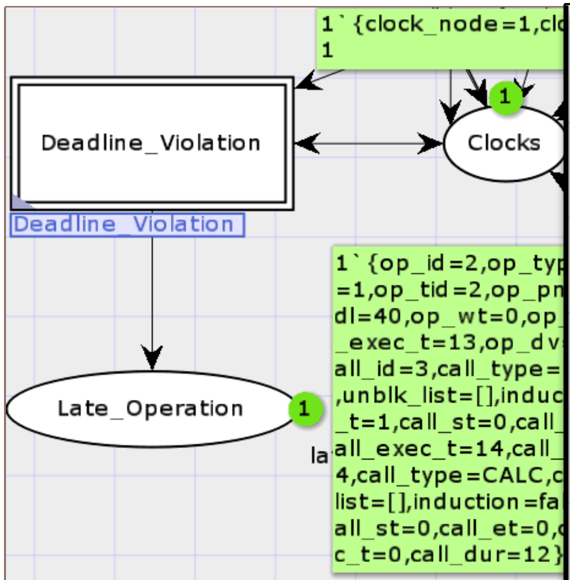
Trajectory Planner (2/2)



State Space Analysis

- Generate a bounded state space; a tree of possible executions from the initial state in the CPN model
- The analyzable states of this system are observed in the markings of the various CPN places in the model
- Using both standard and user-defined queries, the state space is searched to check for system properties
  - Deadline Violations in component operations
  - System-wide deadlocks
  - Worst-case trigger to response times for a known trigger and response operation
  - Partial thread execution order generation based on timing requirements

State Space Analysis – Deadline Violation



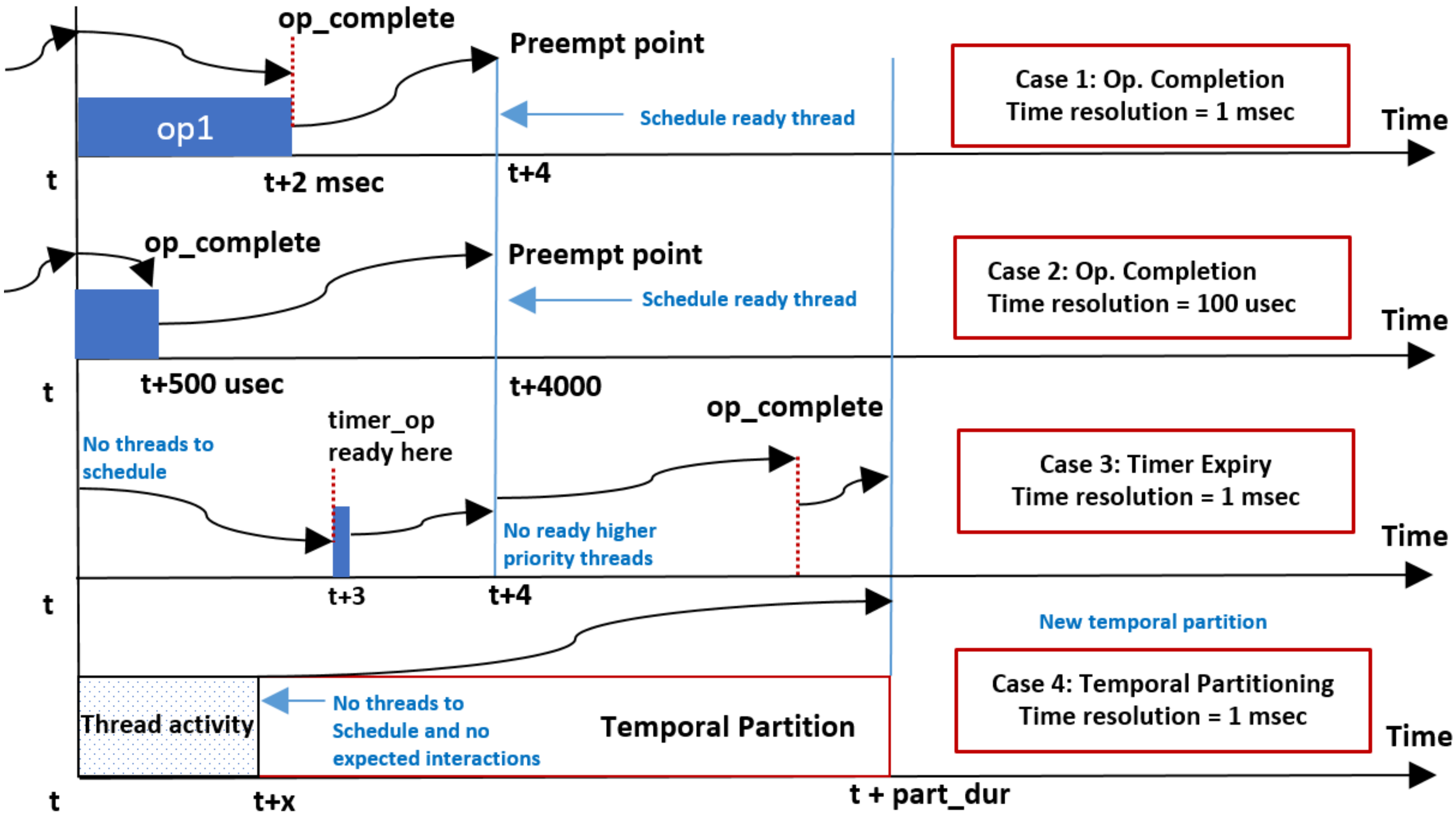

```
val DeadlineViolation = fn : Node -> bool
val Get_Violation_List = fn : Node list -> late_opn ms list
val LateOperation_nodes =
  [99,98,97,96,95,94,93,92,91,90,89,88,87,86,85,84,83,82,81,80,79,78,77,76,75,
   74,73,72,71,70,69,68,67,66,65,64,63,62,61,60,59,58,101,100] : Node list
val sortedLateOperation_nodes =
  [58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,
   83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101]
  : CPN'ColorSets.IntCS.cs list
val FirstDeadlineViolation =
  [{op_calls=[{call_blkt=0,call_dur=14,call_et=0,call_exec_t=14,call_id=3,
            call_op_id=2,call_st=0,call_type=RMI_c,induction=true,pr_t=1,
            q_t=1,unblk_list=[]},
           {call_blkt=0,call_dur=12,call_et=0,call_exec_t=0,call_id=4,
            call_op_id=2,call_st=0,call_type=CALC,induction=false,pr_t=0,
            q_t=0,unblk_list=[]}],op_dl=40,op_dv=true,op_et=0,
    op_exec_t=13,op_id=2,op_nid=1,op_pn=2,op_prio=5,op_st=20,op_tid=2,
    op_type=DDS_OP,op_wt=0}] : late_opn ms
```

```
fun DeadlineViolation n = (Mark.New_Page'Late_Operation 1 n <> []);

fun Get_Violation_List [] = []
  | Get_Violation_List (first_node::rest) =
   (Mark.New_Page'Late_Operation 1 first_node)::(Get_Violation_List rest);

val LateOperation_nodes = SearchNodes (
       EntireGraph,
        fn n => (DeadlineViolation n),
        NoLimit,
        fn n => n,
        [],
        op ::);

val sortedLateOperation_nodes = sort INT.lt LateOperation_nodes;
val FirstDeadlineViolation = (hd (remdupl(Get_Violation_List sortedLateOperation_nodes)));
```

Analysis Improvements: Handling Time

- Explicit modeling of time as an integer-valued *clock* color token in CPN
- Modeling the OS scheduler clock this way allows for easy extensions to its data structure to support
  - Intermediate time stamps and internal state variables
  - Adding temporal partitioning and other prioritization schemes
- Reduces the total number of colors required by the complete model
- Chosen a time quantum is 1 msec (per typical 1KHz scheduler in Linux)

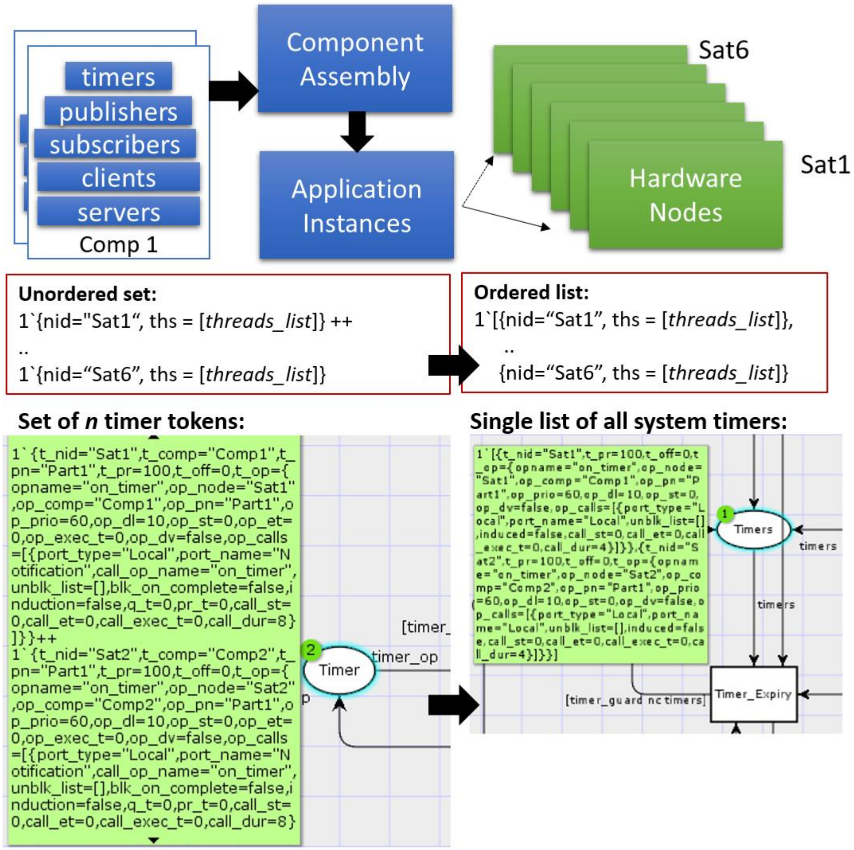
Smarter Handling of Time Progression



Fast-forward time up to the next relevant event (DEVS approach)

Analysis Improvements: Distributed Deployment

- Early models of distributed deployments included a unique token per CPN place for each hardware *node* in the scenario
  - Lead to non-determinism in transition bindings
- Employ structural reduction
  - Merge hardware *node* tokens into a single *list* of tokens instead of an unassociated grouping of node tokens
  - Simultaneous events happen in all nodes at the same time
- *Reduces the resultant state space and dramatically improves scalability*



Advanced State Space Methods

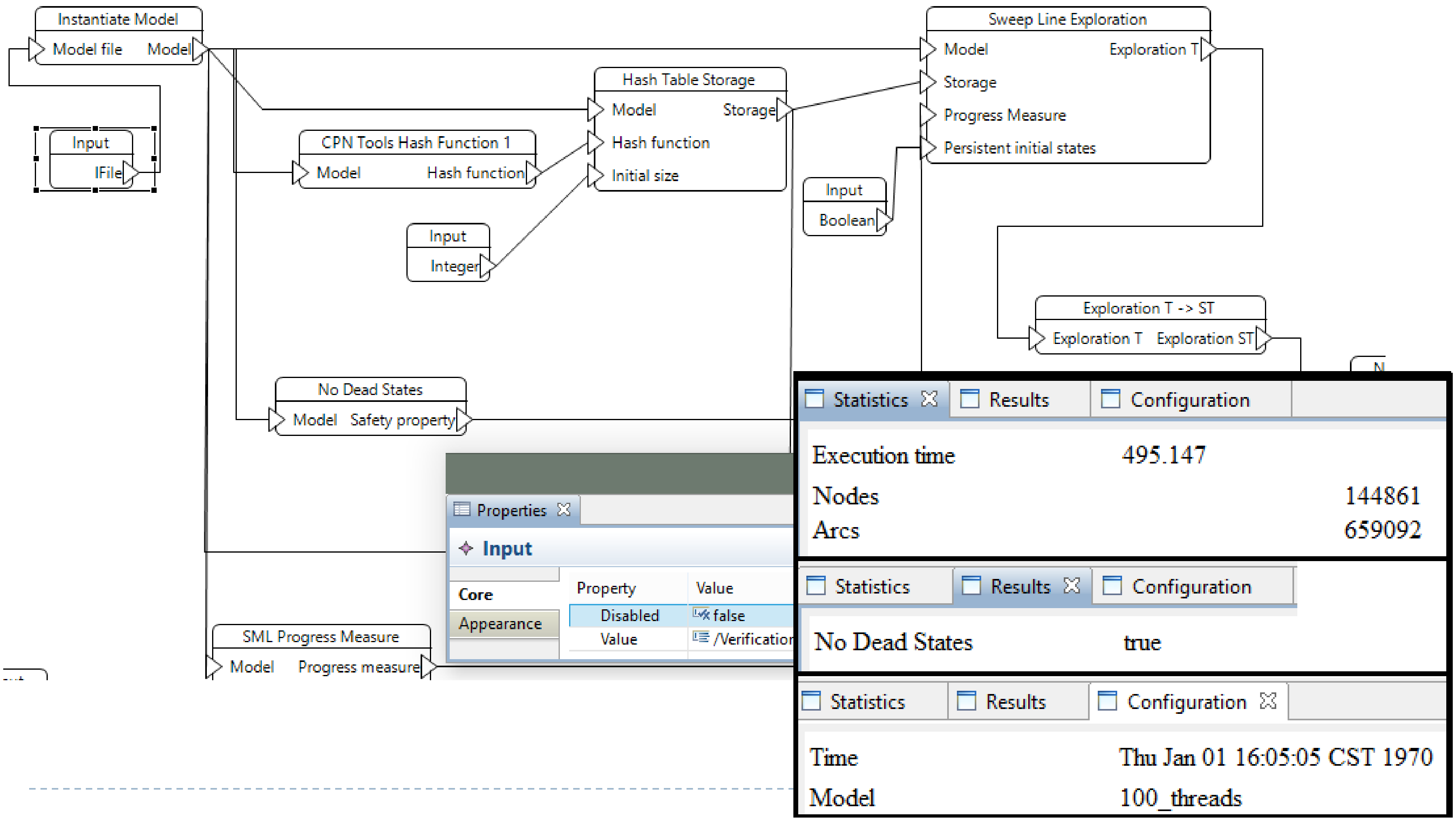
- Compute all reachable states of the modeled system
- Derive a directed graph representing the *state space:* the tree of possible executions that the system can take from an initial state
- Usage:
  - Verify behavioral properties such as lack of queue overflows, deadline violations and deadlocks
  - Derive counter examples if a property is violated
- Potential for state space explosion
  - Needs advanced memory management methods to make state space analysis efficient

Advanced State Space Methods

- In order to easily apply advanced state space reduction techniques, we use a tool called ASAP
- Sweep line method
  - Discard generated states on-the-fly by performing verification checks during generation time
  - Any state that does not violate system properties is deleted
- 100 interacting components in 10 computing nodes:
  - Using CPN Tools built in state space generation:
    - 20 hyperperiods of activity → 36 minutes on a typical laptop
  - Using ASAP and on-the-fly verification
    - Less than 10 minutes to perform deadlock checks on the deployed system

Advanced State Space Methods

ASAP screenshot:



Future Work

- DREMS component communication is facilitated by a time-varying network
  - Bandwidth provided by the system predictably fluctuates between a minimum and maximum (e.g. due to orbital period)
  - Currently we assume worst-case network delay
  - Work in progress: capturing the *network profile* (network performance overt time) of a deployment
- Investigating the utility of this approach on fault-tolerant and self-adaptive systems
  - Integration with a run-time resilience engine
  - Checking for timing anomalies before settling on a reconfiguration strategy

Conclusions

- DREs running time-critical applications must satisfy strict timing requirements to operating safely
- To reduce design and integration complexity, component-based design models are increasingly being used
- Appropriate analysis models are required to study the structural and behavioral complexity of such designs
- Model-based development tools integrated with analysis tools and code generation offer an integrated solution: what is analyzed is the same what runs in the executing system
- Access:
  - https://drems.isis.vanderbilt.edu – complete system
  - Github: rosmod project – next generation based on ROS

Thank You

Overview and Outline

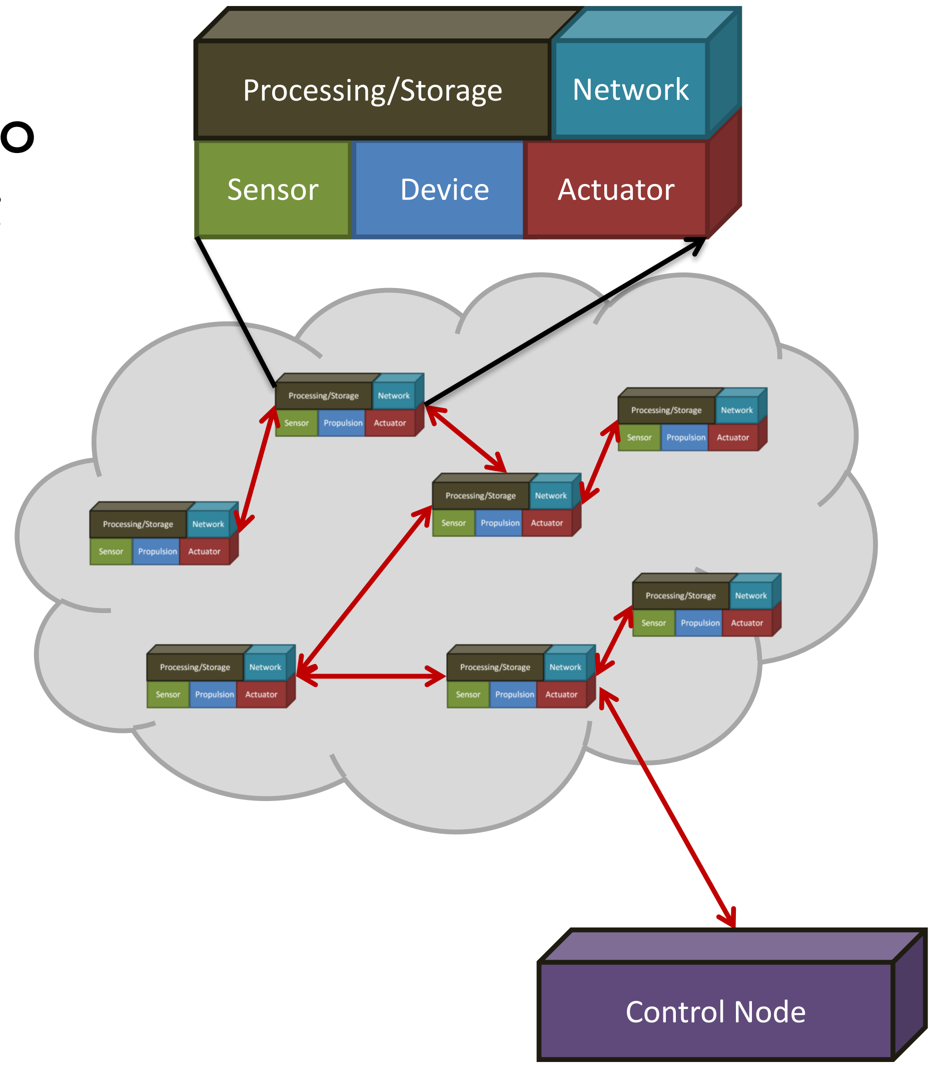
- Distributed Real-time Managed Cyber-Physical Platforms
  - Distributed Applications
  - Challenges
- Layered architecture for building a distributed software platform for these systems
- Overview of the DREMS platform, a prototype implementation.
- Example

Distributed Real-time Managed Cyber-Physical Platforms 1/2

- Built not as a single use, single function network, but as networked (wireless) platforms that can be used by many, possibly concurrent users
- Physical configuration/topology affects the available computational resources.
- Physics imposes timing constraints on the computational and communication activities.
- Critical system software is required to verifiably meet the design requirements.

Distributed Real-time Managed Cyber-Physical Platforms 2/2

- Applications span multiple nodes, for reasons related to the availability of resources:
  - some nodes may have sensors,
  - some may have actuators,
  - some may have computing,
  - some may have storage resources.
- Applications must be architected to rely on loosely interacting components.

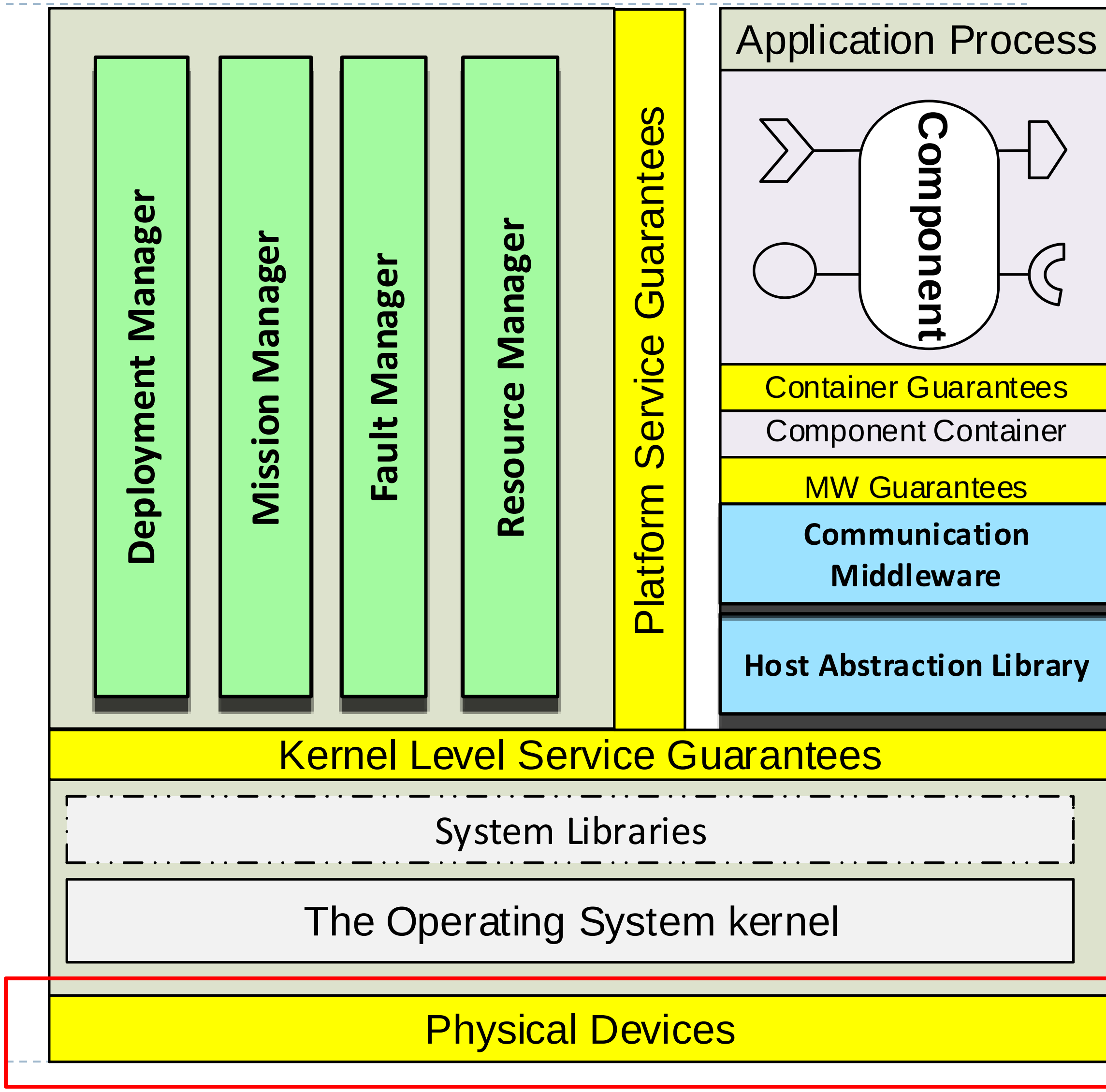


Challenges

- Remote deployment and configuration
- Resilient operation requirements
- Share applications from different vendors and users with different privileges.
- Information sharing/ leakage between applications must be controlled under an overall system security policy
- Performance isolation is critical
  - One application should not be able to affect the functionality or performance of another application
  - The absence of strong performance isolations will also result in security problems

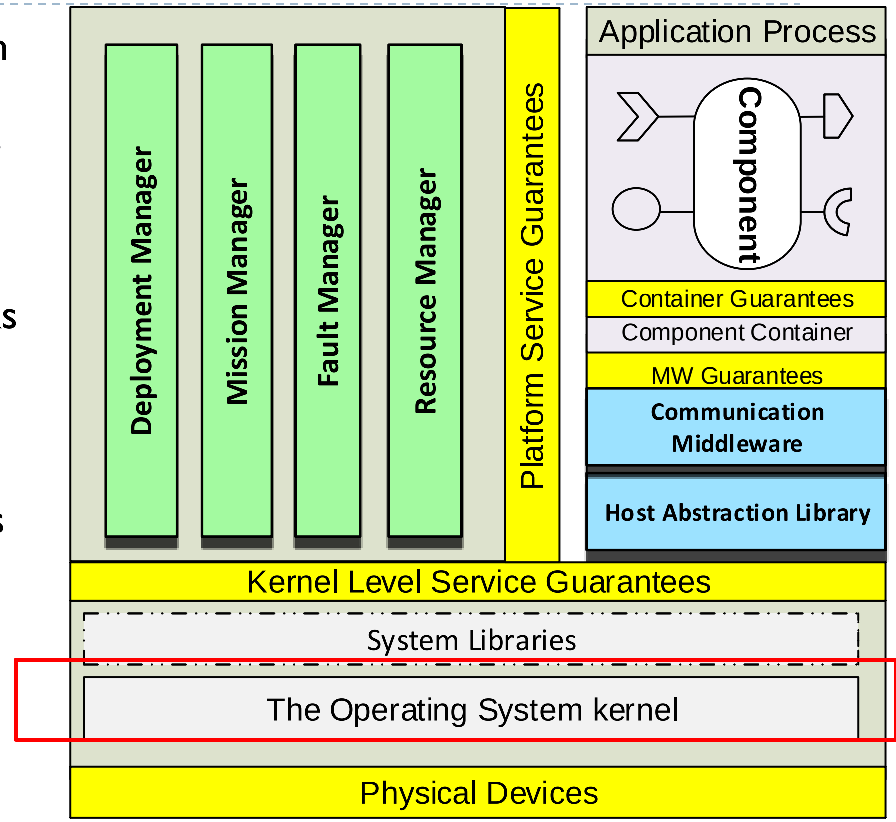
Layered architecture approach

- Watchdog timers reset the platform upon deadlock or critical failures.
- Robust Networking
  - support time-constrained and real-time communications with guarantees
  - Provides updates on channel bandwidth and expected latency map.
  - support multiple traffic classes natively
  - Provide protection against external interference
  - Support link level encryption



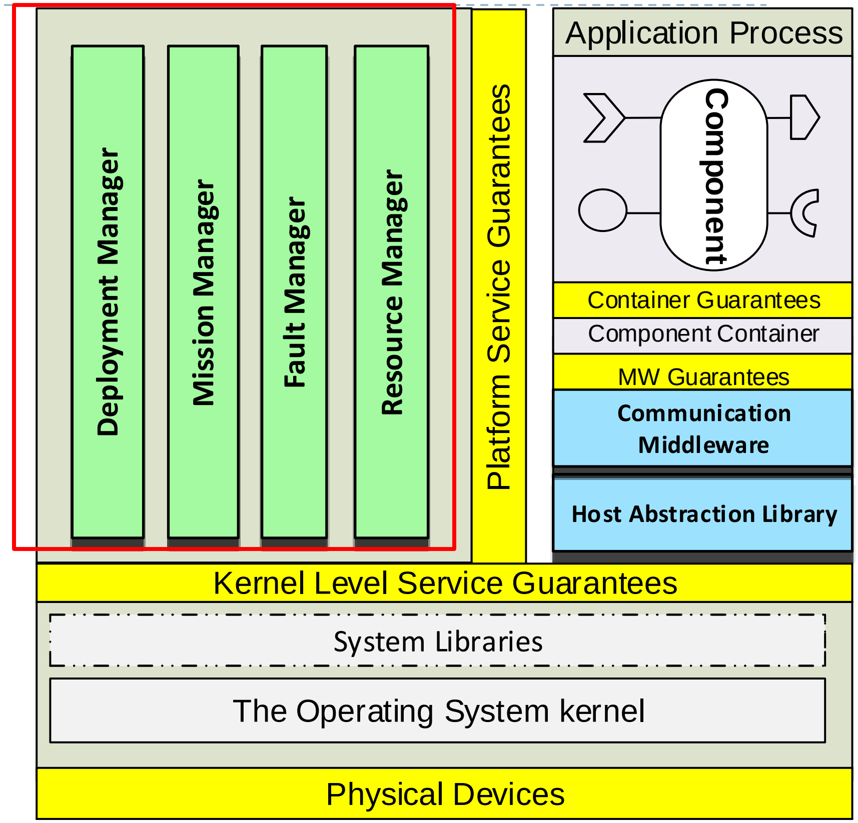
Trusted Computing Base: Operating System

- Support for multiple critical levels of computation tasks
  - Scheduling policies that provide verifiable guarantees for the timeliness of task execution.
  - Strict isolation between applications of different criticality levels.
- Multiple critical levels of communication networks
  - Different traffic classes.
  - Reduction in covert channels of communication.
  - Real-time support for communication.
- Fault tolerant clock synchronization across nodes over wireless network is required
- Confidentiality, Integrity and Authenticity guarantees for communication (require cryptography devices)
  - Mandatory access control and multi-level security may be required.
- Rich and extensible capability model
  - Control what platform services can be used by an application
  - Support for easy migration of application processes between nodes without affecting other applications



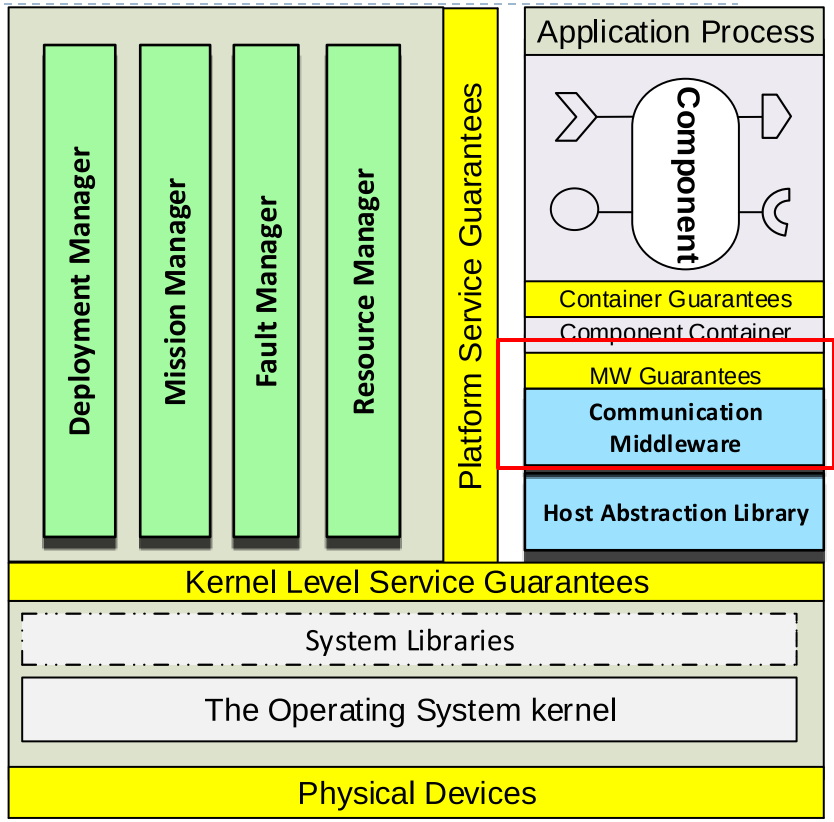
Trusted Computing Base: Platform Services

- Privileged runtime software actors are required for system level tasks
- Deployment Management
  - Fault tolerant application deployment and configuration.
- Mission management
  - Activate/Deactivate applications based on events/passage of time.
- Fault Management
  - Restore mission/system functionality without external intervention
- Resource Management
  - implement a dynamic resource allocation policy, where applications can dynamically request and release resources, and the service honors or rejects these requests while maximizing system utility



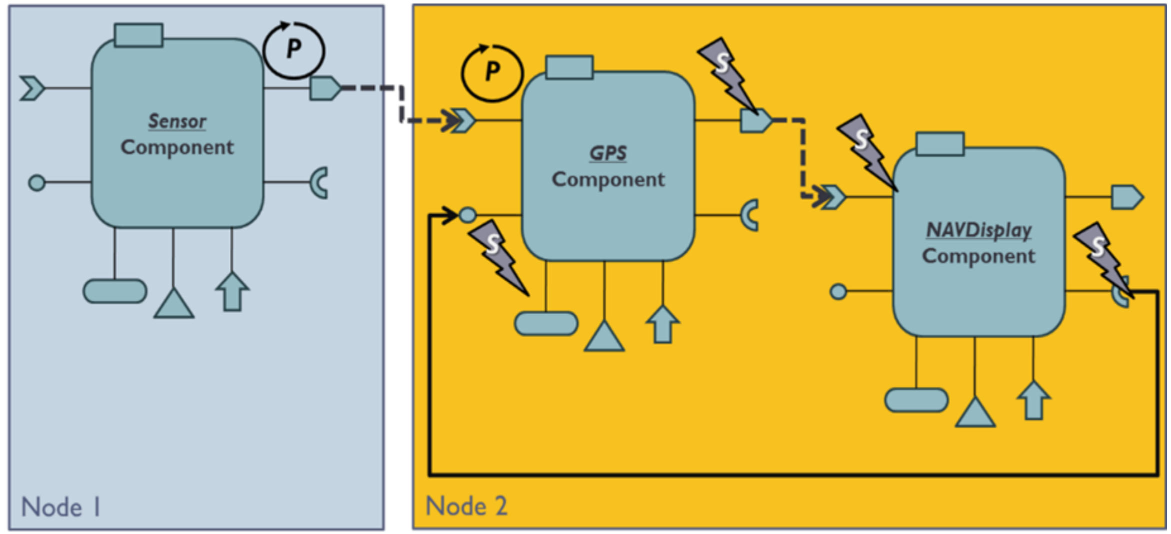
Middleware

- Well defined interaction patterns, e.g.
  - Point to point
  - Pub/Sub
- Provide abstractions to configure and use the different traffic classes and the quality of service information provided by the networking layer
- Support for store and forward for non-real-time communication flows.
- Support for multiple levels of security at the middleware level.



Robust Software Component Model and Development Tools

- Clearly delineate computational aspect from communication aspect.
- Precise scheduling model for the operations.

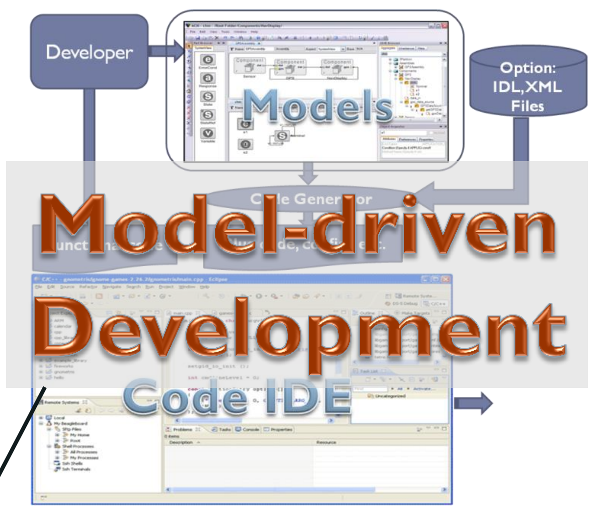


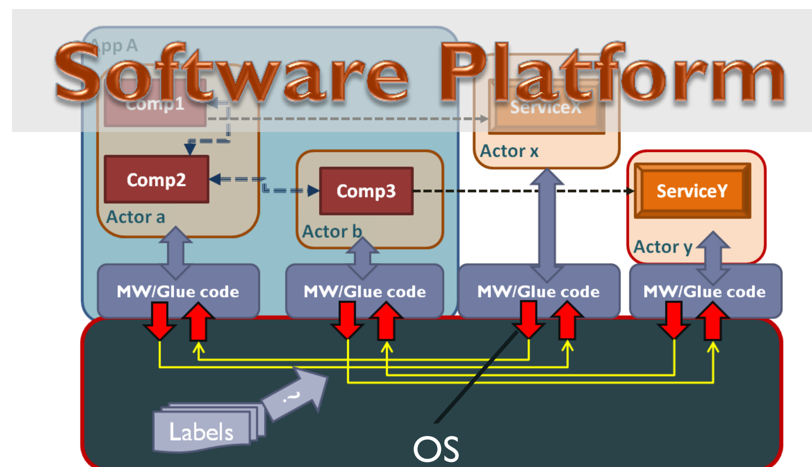
- Support for security policies. Should also provide fault management, including anomaly detection, diagnosis and fault mitigation.
- Prevent technology lock-in:
  - Support for different programming languages
  - Should not require any particular middleware implementation.
- Support for integrated development tools that reduce the time to develop and integrate new applications into the system.

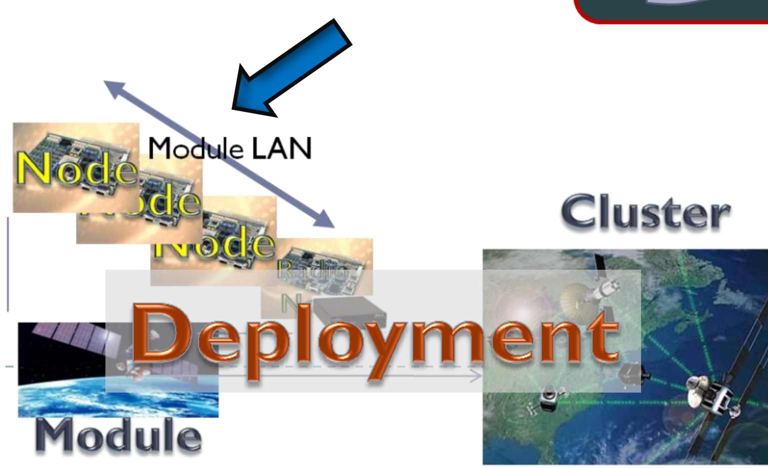
Preliminary Results: DREMS toolchain and platform

DREMS

- Distributed REaltime Managed System. DREMS consists of:
  1. A software *platform*, consisting of an OS and middleware
  2. A software *toolchain,* for modeling applications





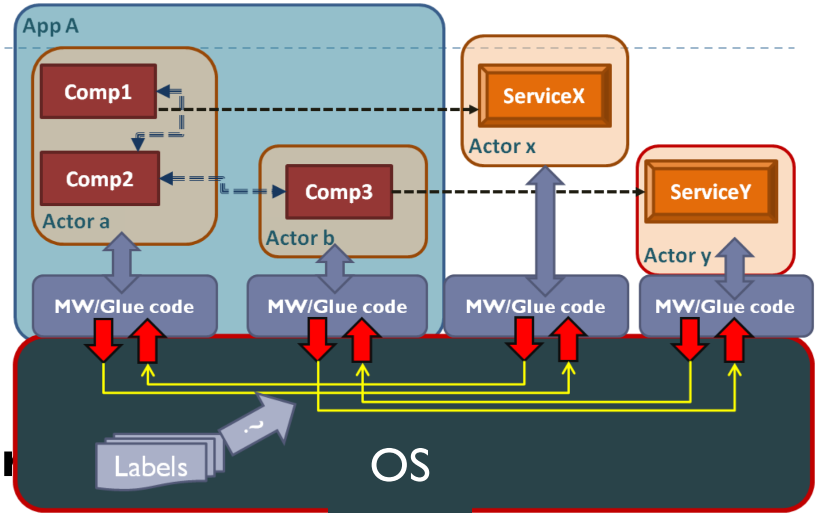


1. Software <u>*toolchain*</u> for modeling, synthesis, analysis, and verification

2. Software <u>*platform*</u> with support for resource sharing, security, and fault tolerance

Operating System

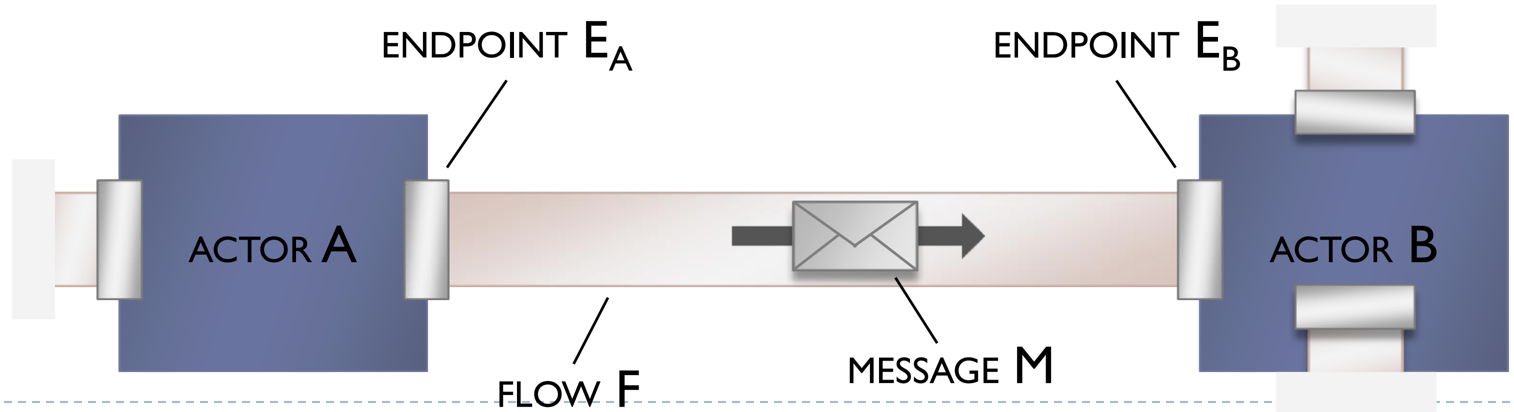
- Supports actor concept.
- Resource sharing is strictly monitored and controlled.
- Fine-grained capability model that controls access to different system services.
- All interactions among actors are via ‘**secur** **transport**’.
- Part of the Trusted Computing Base that enforces the MLS/MAC security policies.



- Real-time scheduling model that supports mixed-criticality systems.
  - System (platform Actors – uses simple priority scheduling)
  - Application (uses temporal partitioning)
  - Best effort (uses CFS)
- Temporal partitioning. Each partition is temporally separated from others.

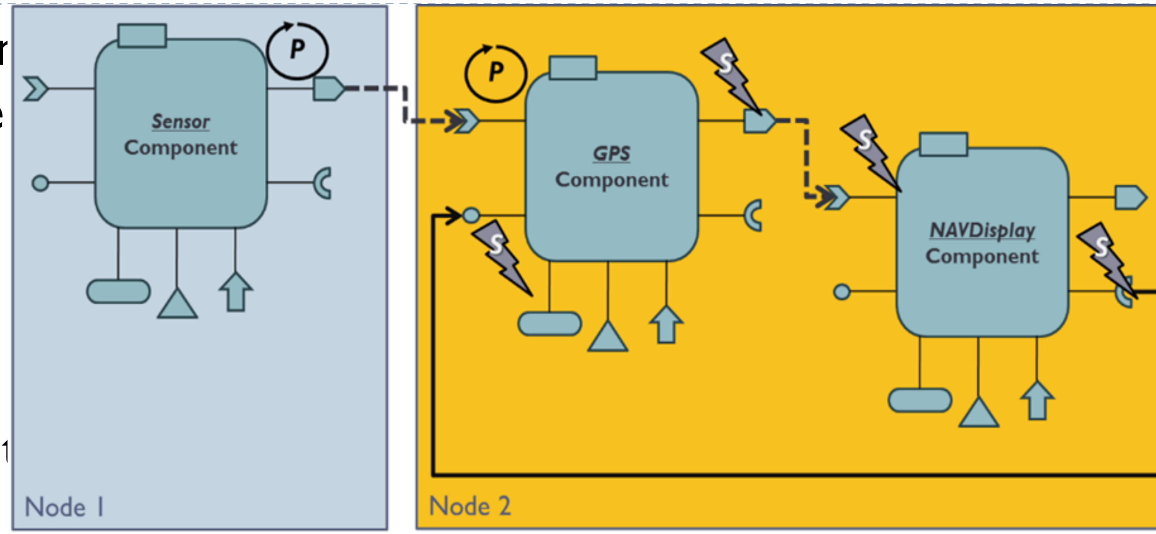
Secure Transport

- The secure transport mechanism enforces the following:
  - Actors write messages only to **endpoints.**
  - Endpoints and **flows** are configured only by trusted platform Actors; used by regular Actors.
  - Enforces mandatory access control for the messages.
  - Supports both UDP and SCTP protocols.
  - All messages must have an label assigned *by the sending actor.*
  - The messages can go to a destination if and only if the destination has a label that **dominates** the label of destination.



Component-Based Distributed Applications

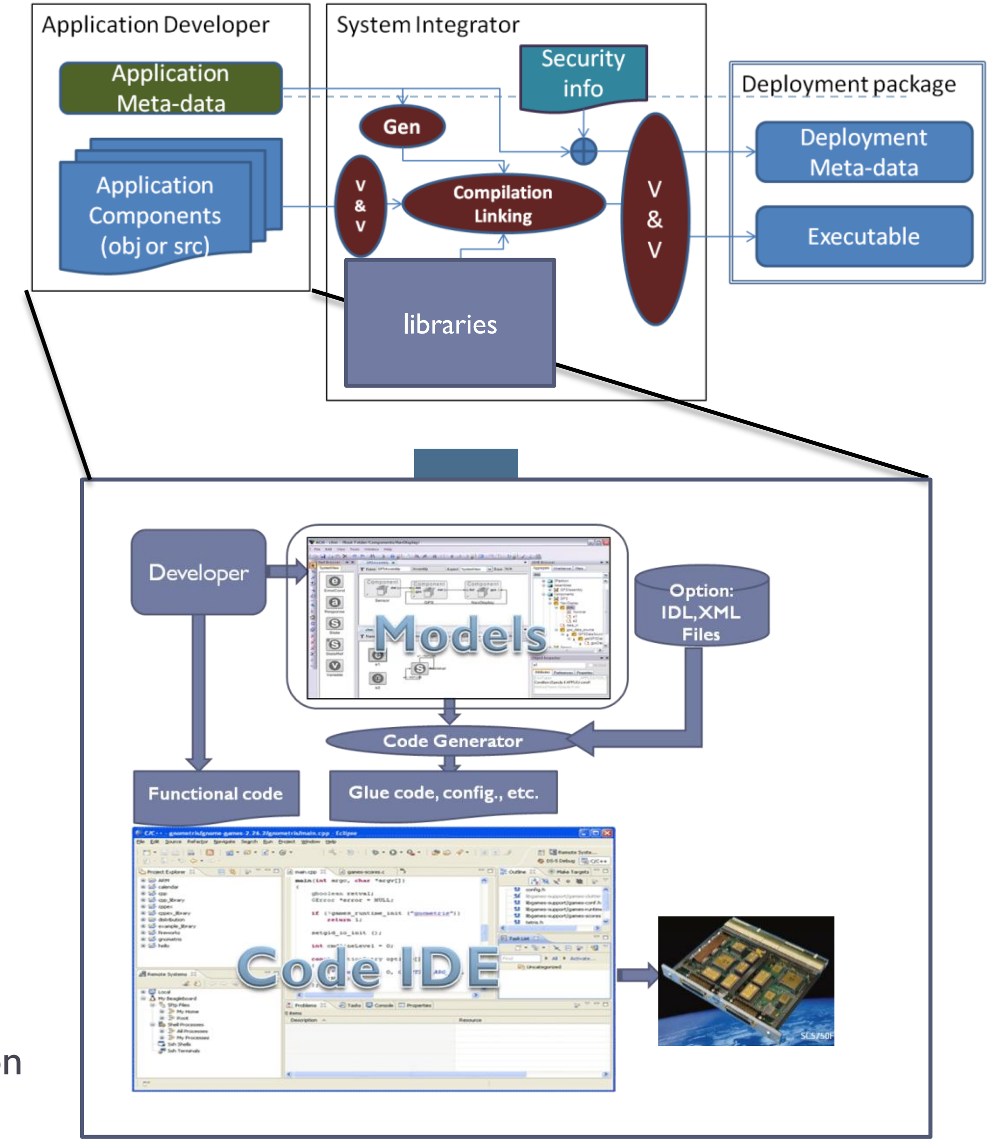
- Apps are architected as or or more actors that share information via secure messaging.
- Actors encapsulate components, with well-defined interaction semant



- Clearly delineate computational aspect from communication aspect.
- Utilizes connectors for adapting to different middleware implementations.
- Component work is divided into operations which are scheduled one at a time.
- The business logic code written by developers is free from any synchronization code, which is one of the common mistakes made by developers.
- Provides different scheduling policies for operations: First-In First-Out, Priority, Earliest Deadline First.

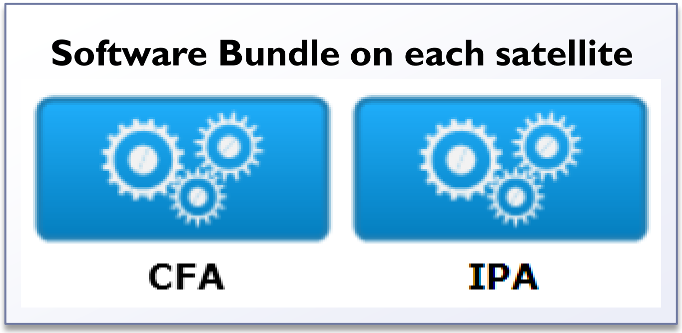
Development and Integration Environment

- Designer creates architectural models for actors and components, and their interactions
- Models are annotated with resource needs, names for security labels, pre/post-conditions, invariants, etc.
- Software generator tools produce skeleton/glue code for the application modules
- Conventional development environment is used to supply algorithmic parts
  - Option: import code generated by code generators, like Matlab/Real-time Workshop
- System integrator verifies and assembles complete software suite
  - Check information flows for policy violations
  - Perform admittance tests by checking application communication requirements against expected network resource profile

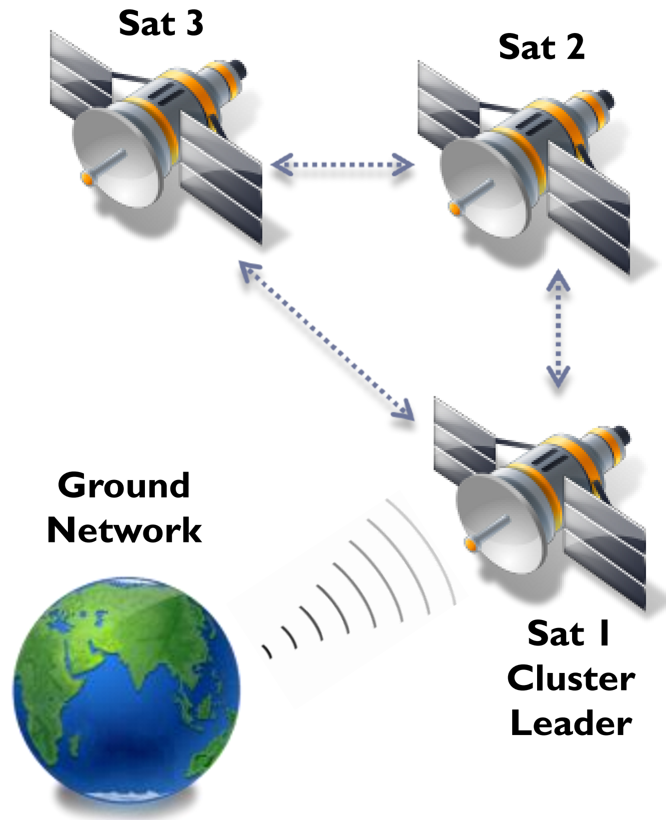


Example

Example: DREMS Orbital Satellite Platform 1/2



Application	Software Components	Partition
Cluster Flight Application (CFA)	Orbital Maintenance	Temporal Partition 1
	Trajectory Planning	System
	Module Proxy	System
	Command Proxy	System
Image Processing Application (IPA)	Image Processing Component (4 on each)	Temporal Partitions 2 & 3



- **Platform management actors** on each satellite provide the capability to **configure**, **deploy** and **manage** the applications and their actors.
- These actors seamlessly share the CPU with the deployed applications.

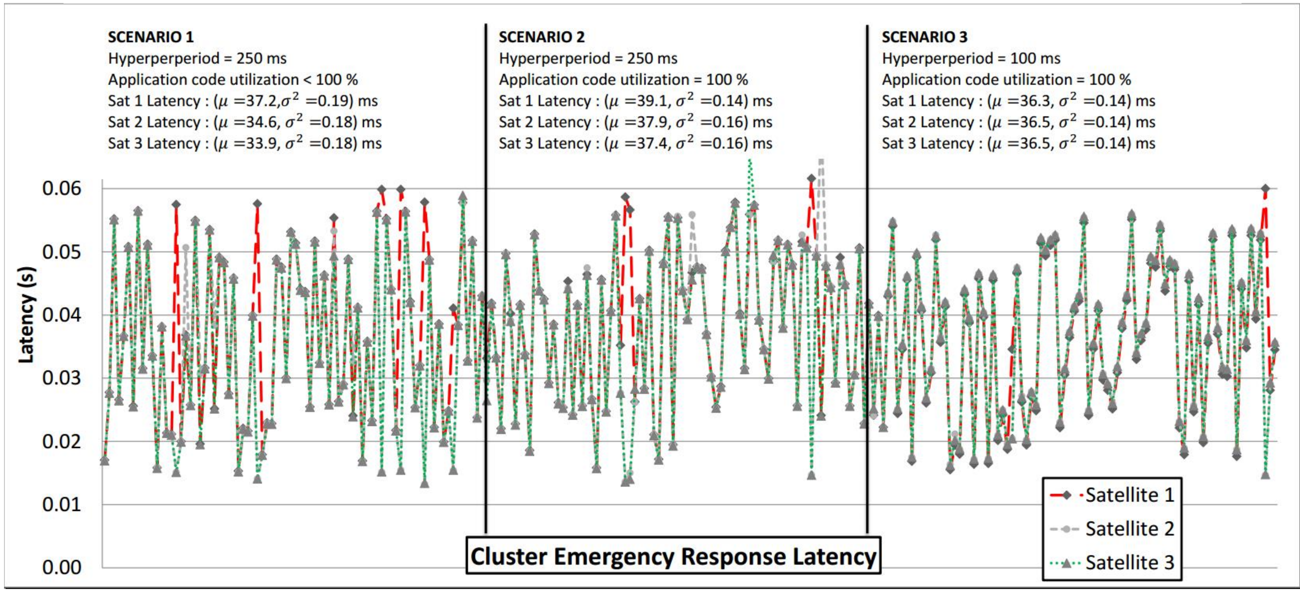
DESIGN-TIME MODELING → **RUN-TIME DEPLOYMENT/MANAGEMENT**

Example: DREMS Orbital Satellite Platform 2/2

- The applications were deployed on the platform.
- Demo integrates the physics model.
- It shows how a critical flight application can be assembled from components using the development tools.
- It shows that the image processing application does not affect the CFA response time.
- The demo shows that the critical flight application is not affected image processing application

BEFORE ENGINE THRUST ACTIVATION at time t

AFTER ENGINE THRUST ACTIVATION at time t + 5 (min)



Summary

- Distributed and managed cyber-physical systems pose requirements that go further than traditional DRE systems.
  - Security, performance isolation, and loose coupling among distributed application are key requirements.
  - The physics of the platform affect the availability of both computation and communication resources
  - A layered architecture that builds upon the guarantees provided by the layer below was described.
- A prototype platform called DREMS was also discussed.
- Please visit: http://www.isis.vanderbilt.edu/DREMS for more details.