Provisioning Dynamic Reconfiguration and Redeployment Capabilities for Enterprise DRE Systems

Gan Deng
Douglas C. Schmidt
Aniruddha Gokhale

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee

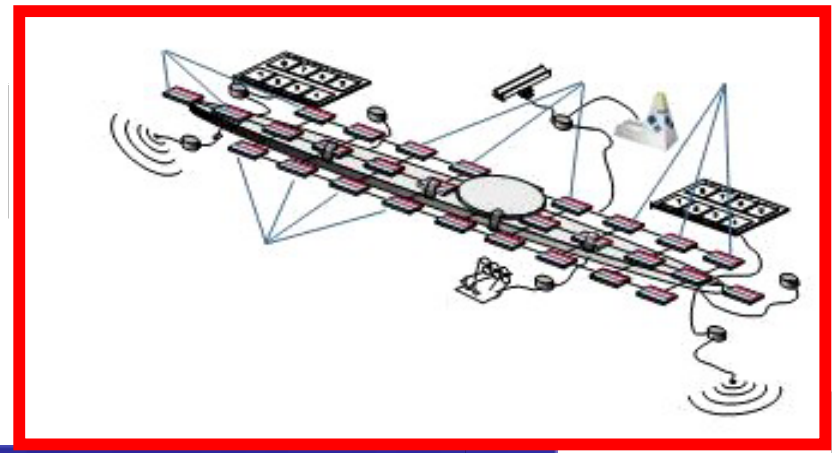
R&D Motivation: Enterprise DRE Systems

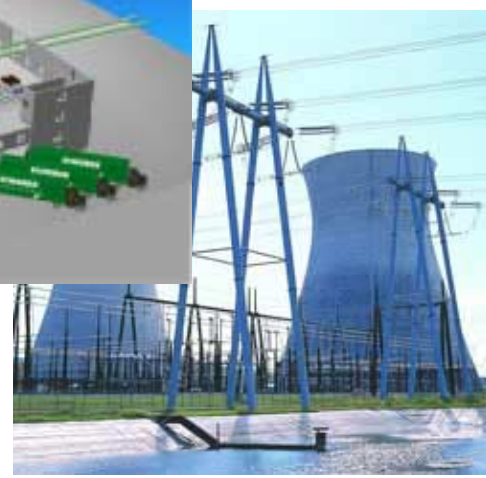
Key Characteristics

- Large-scale, network-centric, dynamic, "systems of systems"
- Simultaneous QoS demands with resource constraints
  - –e.g., loss of resources

Highly Diverse Domains

- Mission-critical systems for critical infrastructure
  - e.g., power grid control, real-time warehouse management & inventory tracking
- ***Total Ship Computing Environment (TSCE)***
  - http://peoships.crane.navy.mil/ddx/



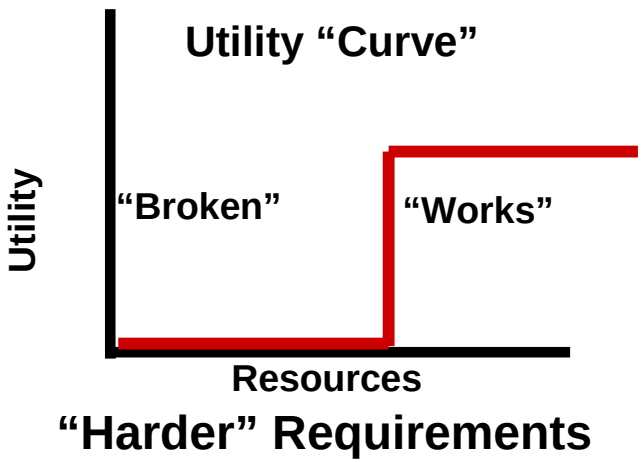


Demands of Traditional DRE Systems

Key Characteristics

- Stringent timing requirements, e.g., deadline, latency & jitter
- System resources & workloads are known in advance
- System resources & workloads change infrequently

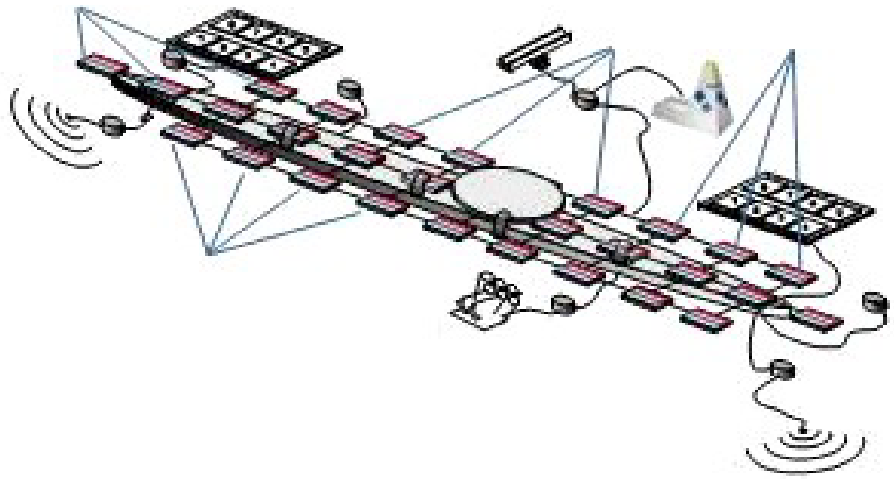
Traditional *closed* DRE Systems



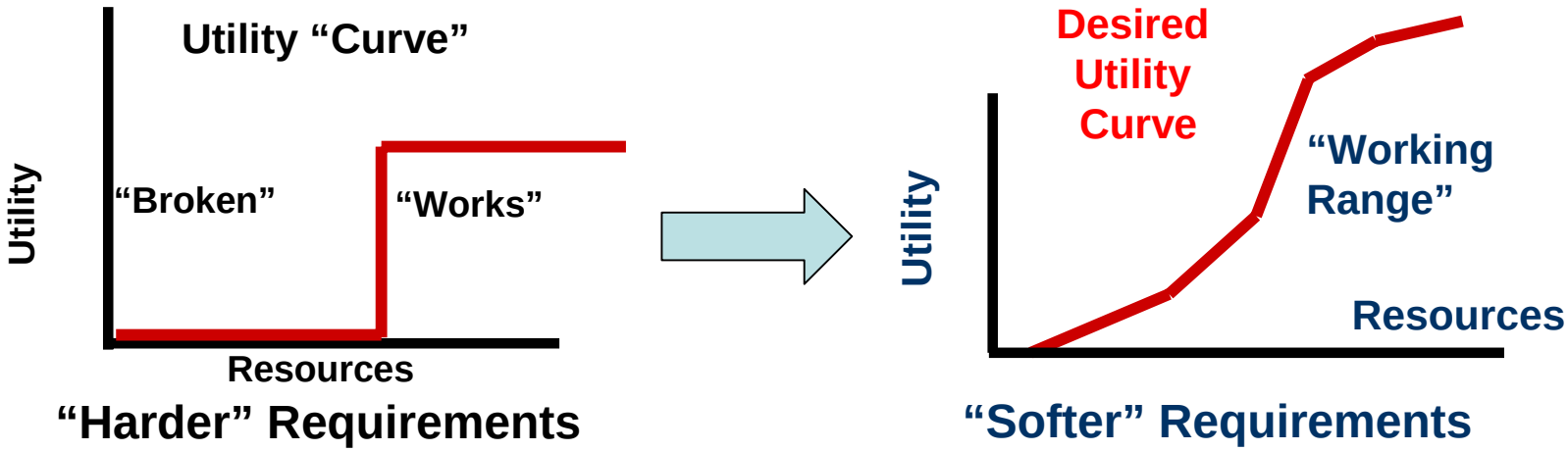
Demands of Enterprise DRE Systems

Key Challenges

- Highly heterogeneous platform, languages & tool environments
- Changing system running environments
- Enormous inherent & accidental complexities

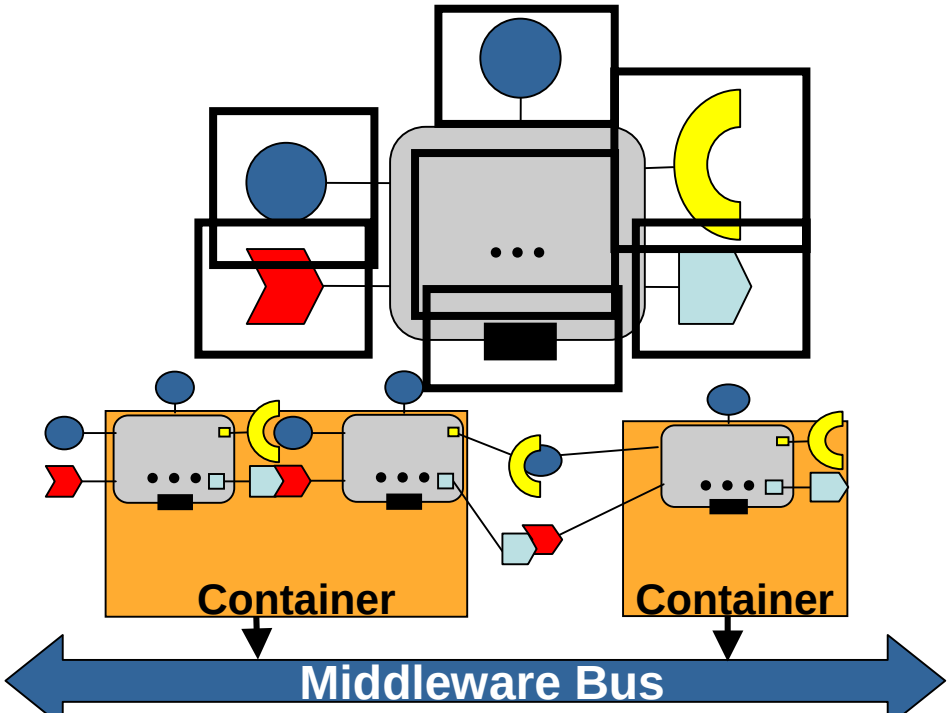


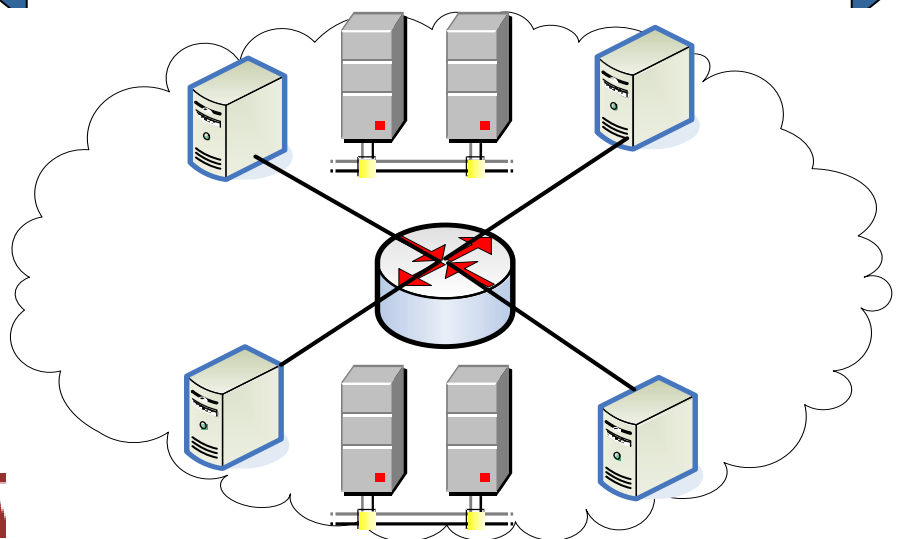
Enterprise DRE Systems



My R&D goal is to ensure end-to-end real-time QoS for enterprise DRE systems

Promising Solution: Component Middleware

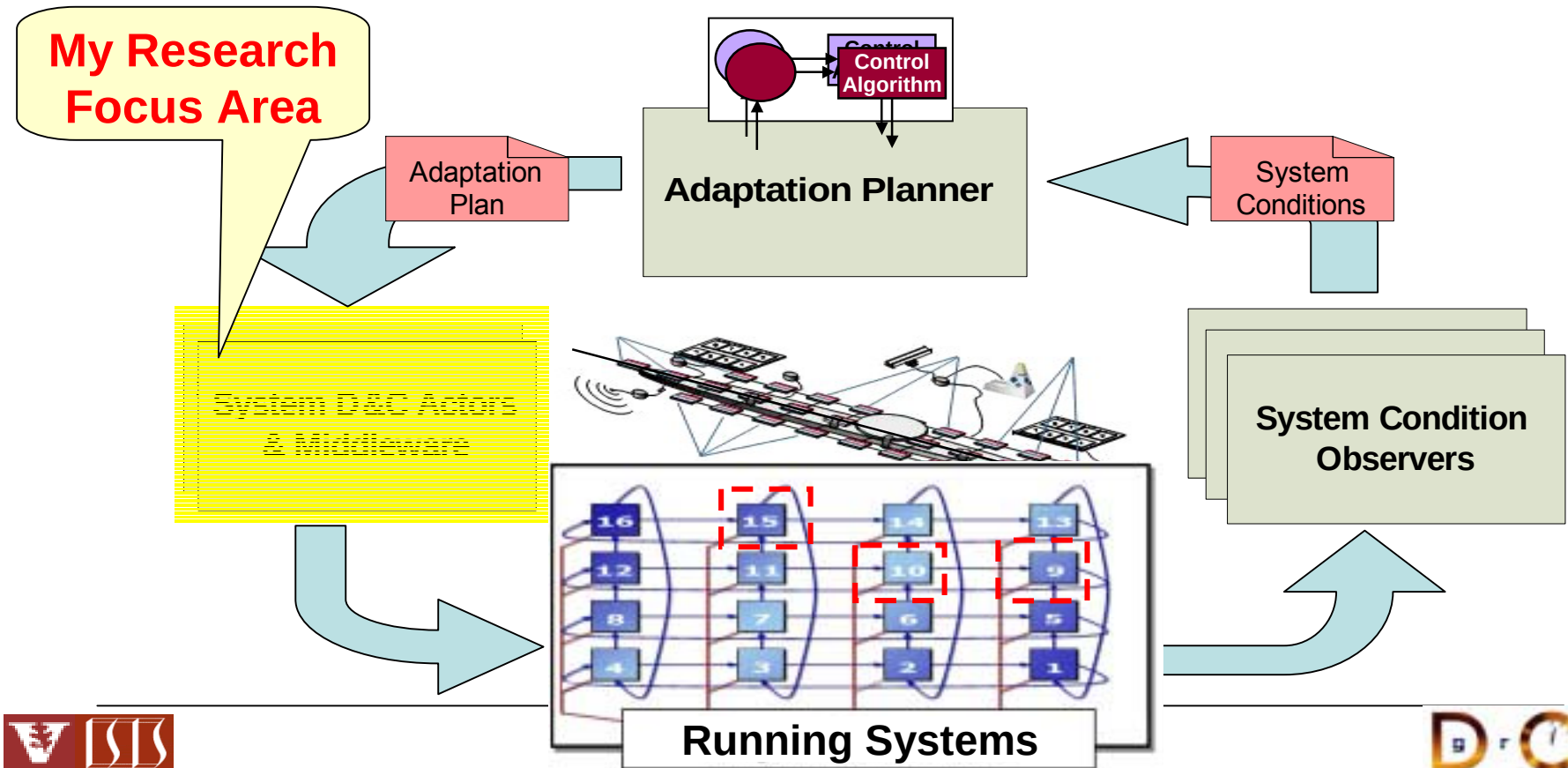




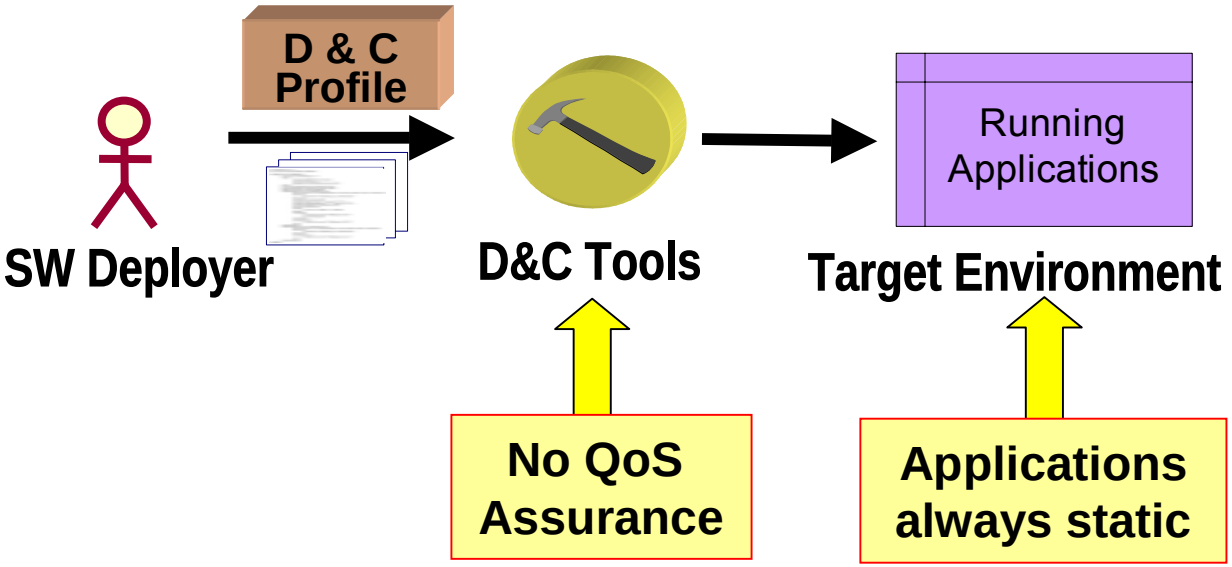
- *Components* encapsulate application "business" logic
- Components interact via *ports*
  - *Provided interfaces*, e.g.,facets
  - *Required connection points*, e.g., receptacles
  - *Event sinks & sources*
  - *Attributes*
- *Containers* provide execution environment for components with common operating requirements
- Components/containers can also
  - Communicate via a *middleware bus* &
  - Reuse *common middleware services*
- All components must be *deployed & configured* (D&C) into the target environment

Runtime QoS Provisioning

- Key Ideas
  - Decouple system adaptation policy from system application code & allow them to be changed independently from each other
  - Decouple system deployment framework & middleware from core system infrastructure to allow enterprise DRE systems dynamically reconfigurable



Limitations with Existing D&C Model



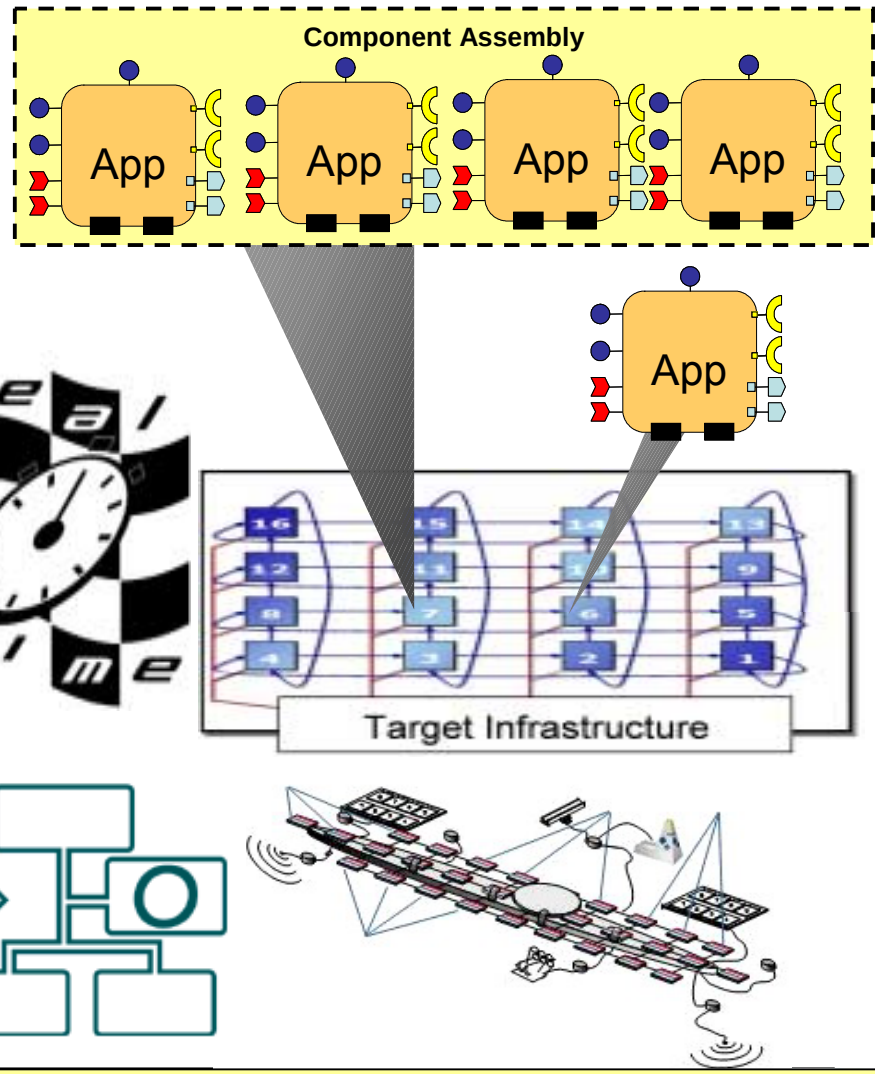
Hypothesis
Existing CCM components & applications could be enhanced with dynamic capabilities w/out breaking standard component programming model & D&C model

- The existing D&C model cannot change the configuration once an application is deployed
  - Must shutdown the entire application & redeploy, which is not feasible for enterprise DRE systems
- The existing D&C model cannot ensure real-time QoS when performing initial D&C & reconfiguration
  - Enterprise DRE systems have stringent QoS requirements for dynamic redeployment & reconfiguration

Baseline: DAnCE→ Deployment And Configuration Engine (D&C Spec)

Proposed Solutions

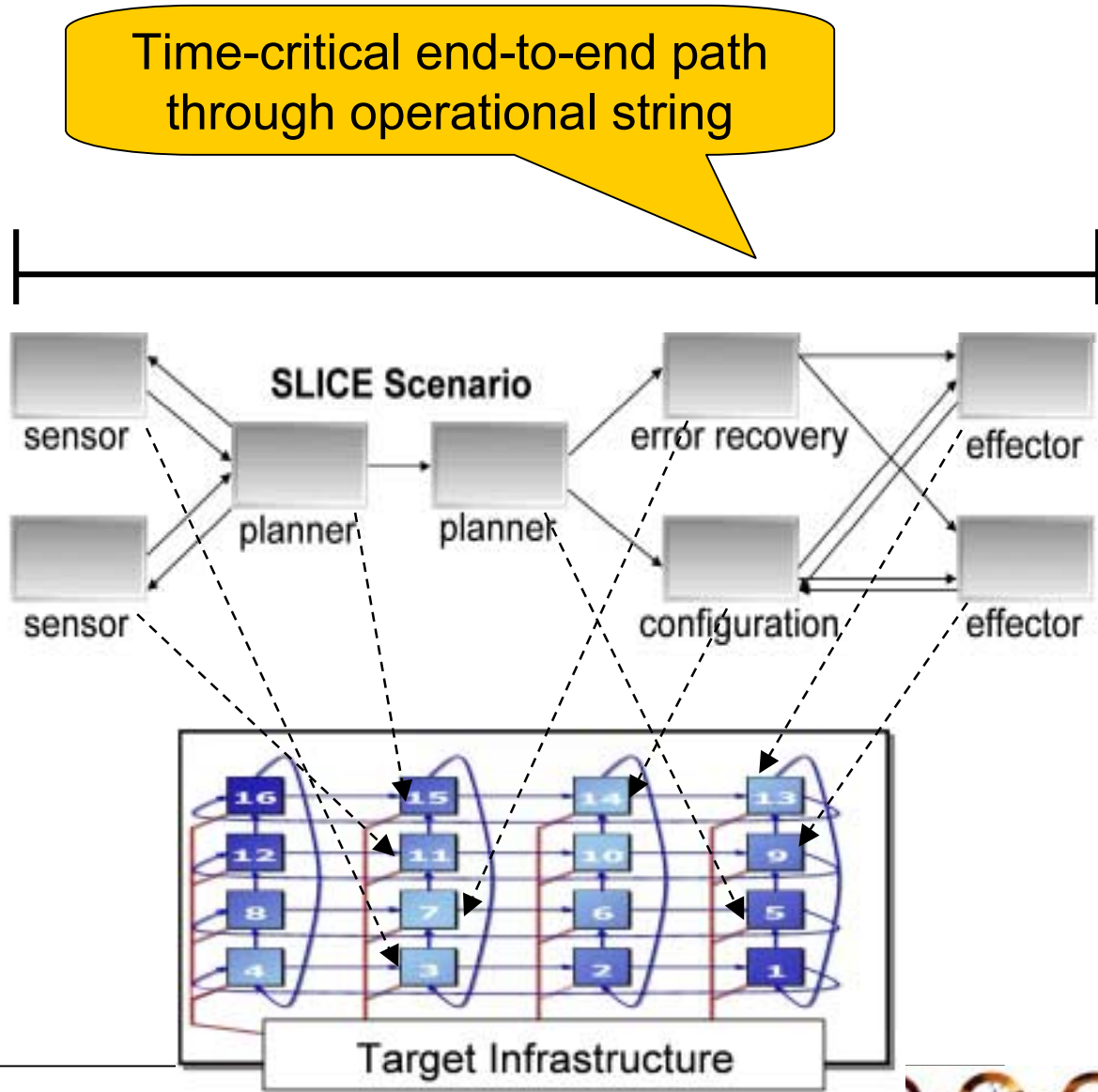
- **Computational model** – Develop dynamic reconfiguration techniques for enterprise DRE systems
- **Execution platform** – Map the dynamic reconfiguration techniques to a more *predictable & time-bounded* execution platform
- **Programming model** – Develop a domain-specific modeling language to simplify dynamic reconfiguration workflow



Constraint **is to support standards-based component & D&C model**

Challenge 1: Reconfigure Enterprise DRE Systems from End-User Perspective

- **Context**
  - Enterprise DRE systems may have thousands of components distributed across hundreds of nodes
  - Components are often grouped together by system architect in the form of *operational strings*
- **Problem**
  - How to make enterprise DRE systems manageable from *end-users* (e.g., software architect, software engineer) perspective?

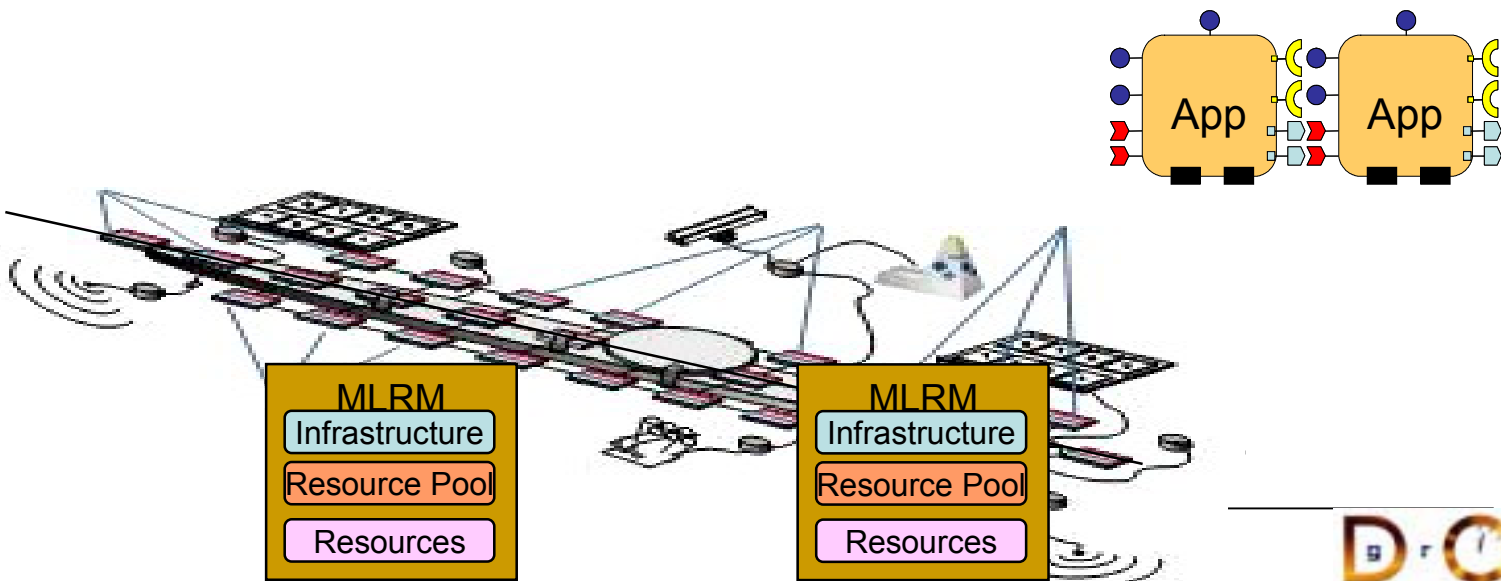


Solution → ReDaC Computational Model

- Develop a ReDaC **computational model**:
  - Introduce operational strings as first-class entities
  - Provide a rich set of services to manage enterprise DRE systems & system resources at different levels of granularity
    - Components

Real-time policy set

Add_Instance <Plan ID> <Node ID> <Component Type> <Policy Set>
Remove_Instance <Plan ID> <Component ID>
Bind <Plan ID> <Source Component ID : Port Name> <Dest Component ID : Port Name>
Remove_Binding <Plan ID> <Source Component ID : Port Name> <Dest Component ID : Port Name>

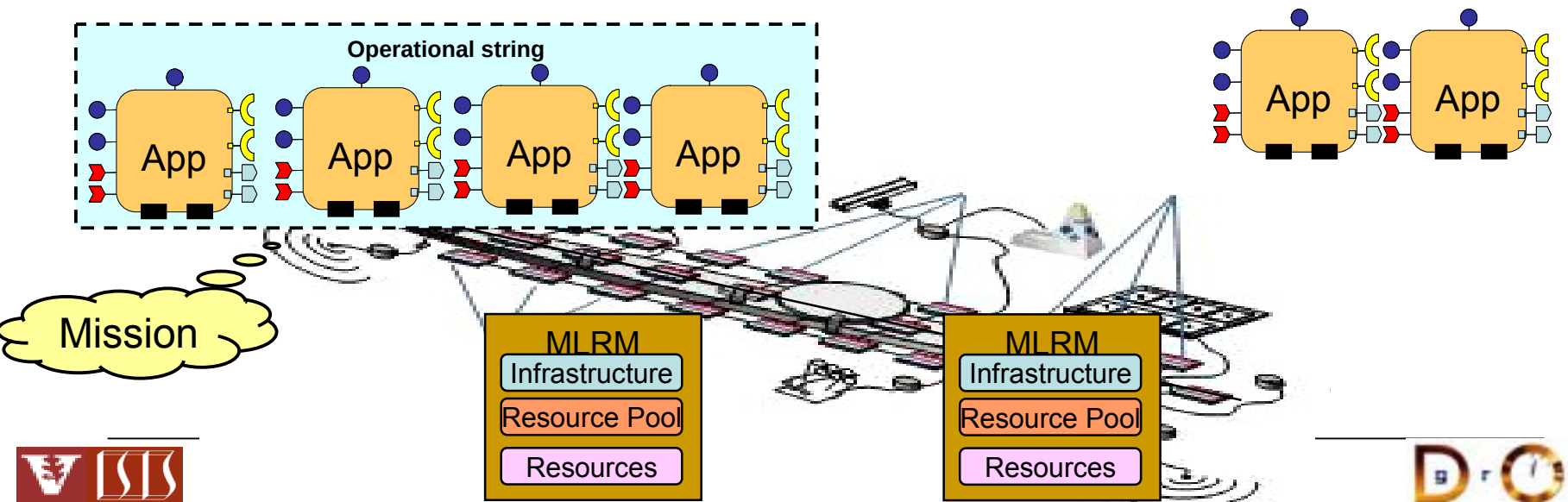


Solution → ReDaC Computational Model

- Develop a ReDaC **computational model**:
  - Introduce operational strings as first-class entities
  - Provide a rich set of services to manage enterprise DRE systems & system resources at different levels of granularity
    - Components
    - Operational strings

Real-time policy sets

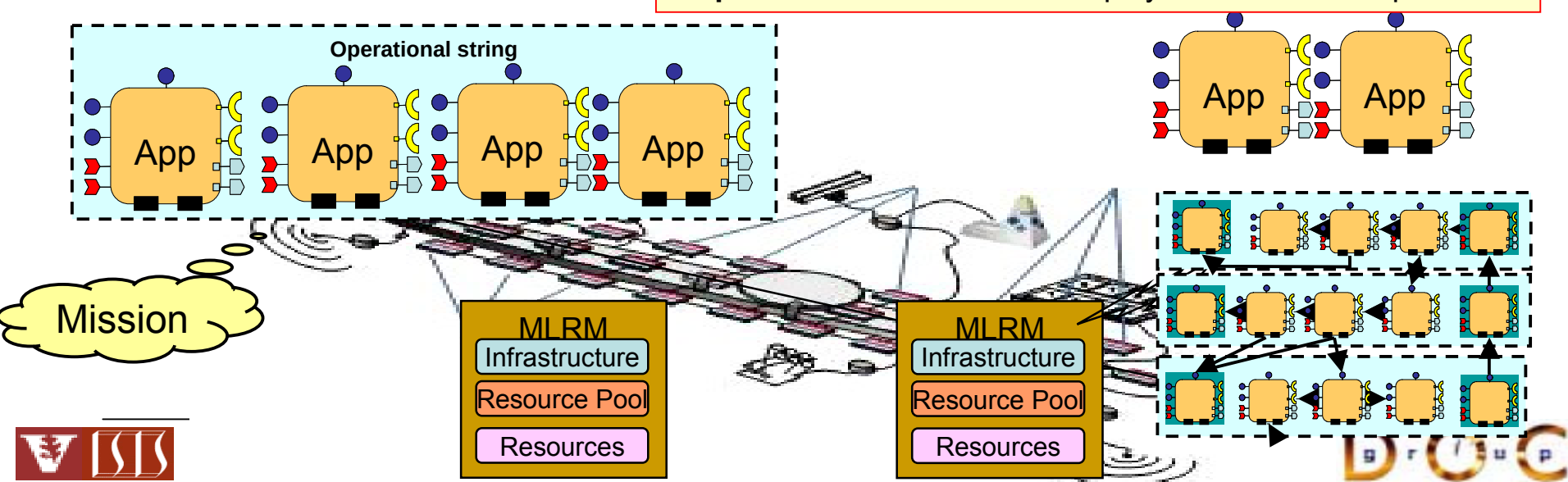
Add_OpString <Plan ID> <OpString Descriptor>
Remove_OpString <Plan ID> <OpString ID>
Update_OpString <Plan ID> <OpString Descriptor>
Bind_OpString_OpString <Source OpString ID : Port ID> <Dest OpString ID : Port ID>



Solution → ReDaC Computational Model

- Develop a ReDaC **computational model** :
  - Introduce operational strings as first-class entities
  - Provide a rich set of services to manage enterprise DRE systems & system resources at different levels of granularity
    - Components
    - Operational strings
    - Entire system deployment plan

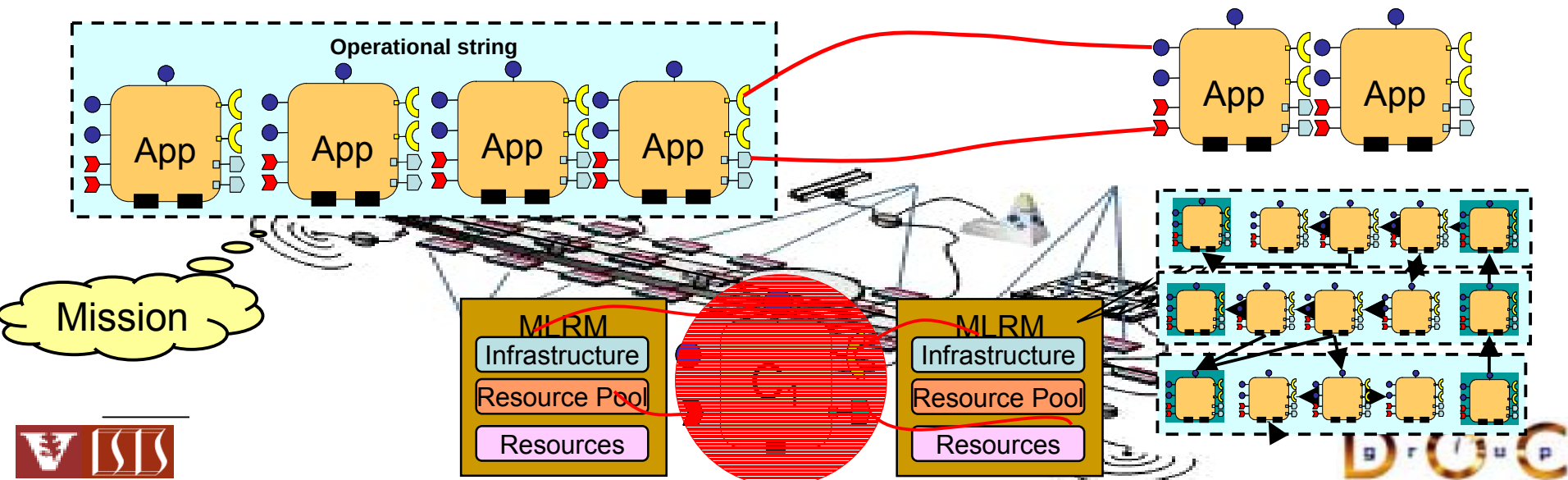
Deploy_Plan < Deployment Plan Descriptor>
Teardown_Plan < Deployment Plan ID>
Update_Plan < Plan ID> <Deployment Plan Descriptor>



Solution → ReDaC Computational Model

- Develop a ReDaC **computational model** :
  - Introduce operational strings as first-class entities
  - Provide a rich set of services to manage enterprise DRE systems & system resources at different levels of granularity
    - Components
    - Operational strings
    - Entire system assemblies
    - Interactions among these entities through well-defined interfaces

Bind_OpString_Component <Plan ID> <OpString ID : Port ID> <Dest Component ID : Port ID>



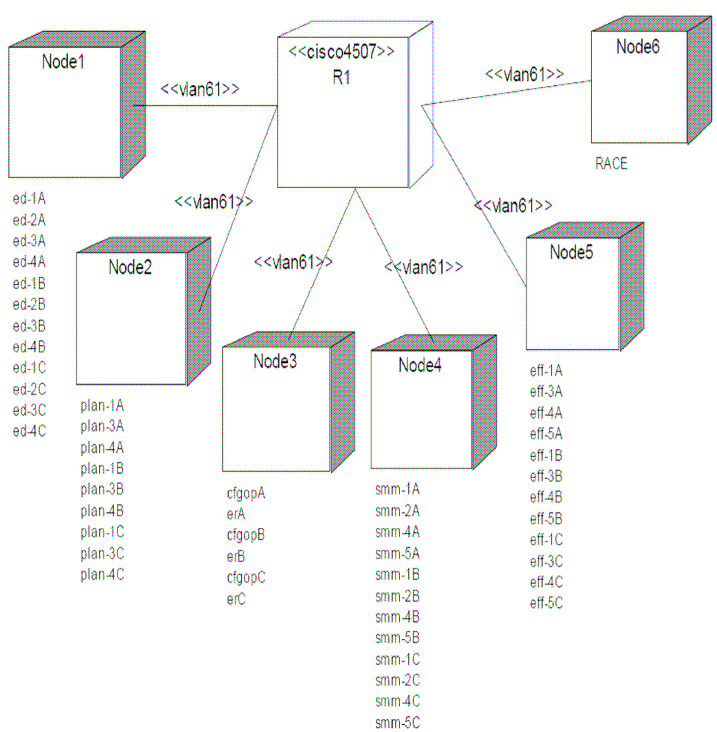
Criteria for Evaluation

Performance Criteria For Reconfiguration (Hypothesis)

- Baseline – Existing DARPA ARMS GT4 Testbed, 3 operational strings, with *mission effectiveness values* (MEV) of 3, 2, 1, redeploy strings.
- Metric M1 – *Average mission effective value loss of all operational strings*

$$M1 = \sum_i \left(\frac{DownTime(S_i)}{TotalTime} \times MEV(S_i) \right)$$

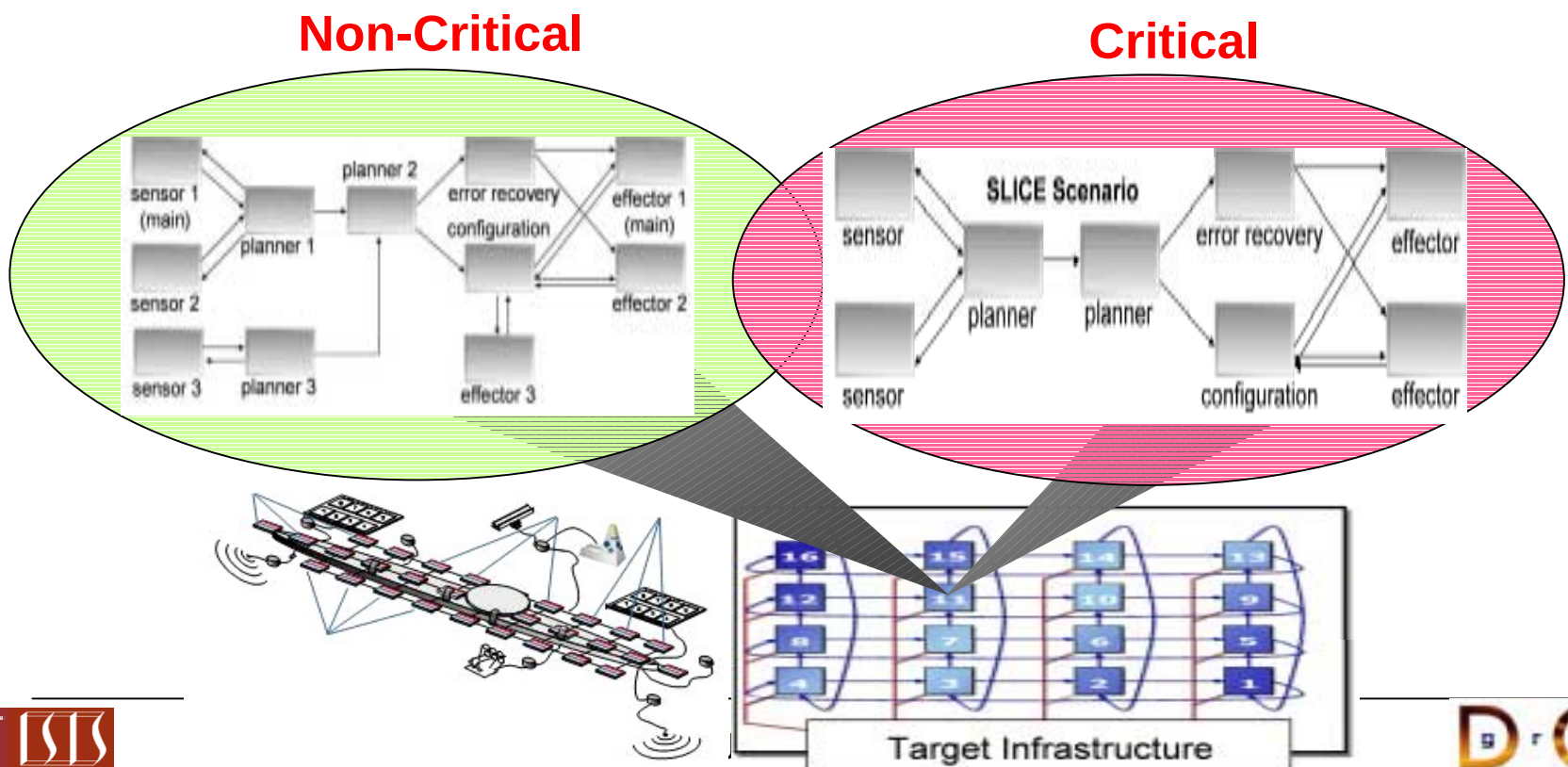
1. **Divide the total down time of each operational string by the total experiment operational time.**
2. **Multiply it by the MEV of that operational string to produce the average MEV loss for that string.**
3. **Sum up all average MEV loss.**



Goal: Reduce M1 value by 20-30%

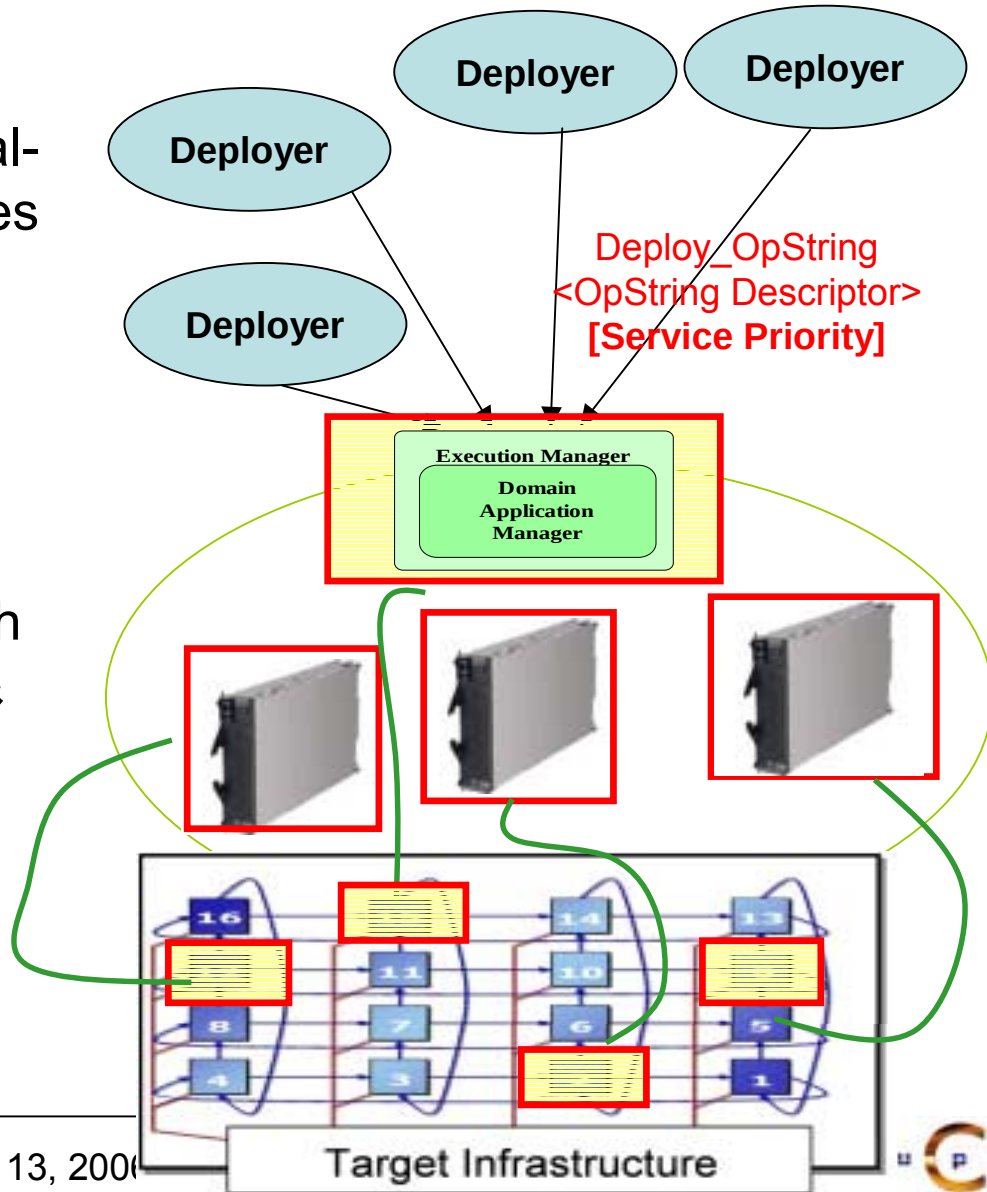
Challenge 2: Ensure Reconfiguration QoS

- **Context**
  - Multiple reconfiguration service requests might arrive simultaneously
- **Problems**
  - How to differentiate services from different requests based on their priorities?



Proposed Solution → ReDaC Execution Platform

- **ReDaC Execution Platform for Service Execution**
  - A novel approach to integrate Real-time CORBA (RT CORBA) features to build a predictable execution platform
  - All standards-based deployment agents (e.g., ExecutionManager, NodeApplicationManager, & NodeManager) are configured with *appropriate RT CORBA* policies & run atop RT CORBA middleware
  - When external clients request setting up & modifying services, the priority level could also be specified as part of the request

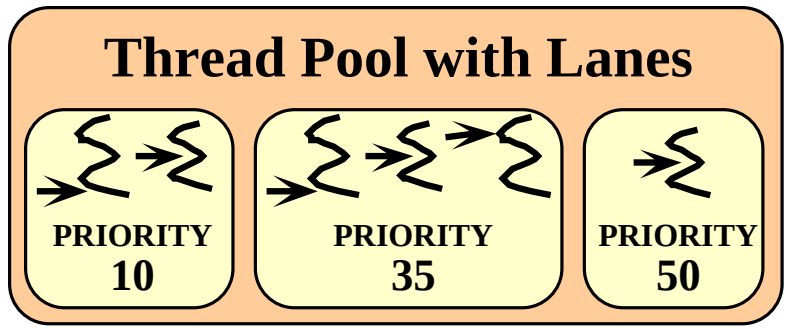


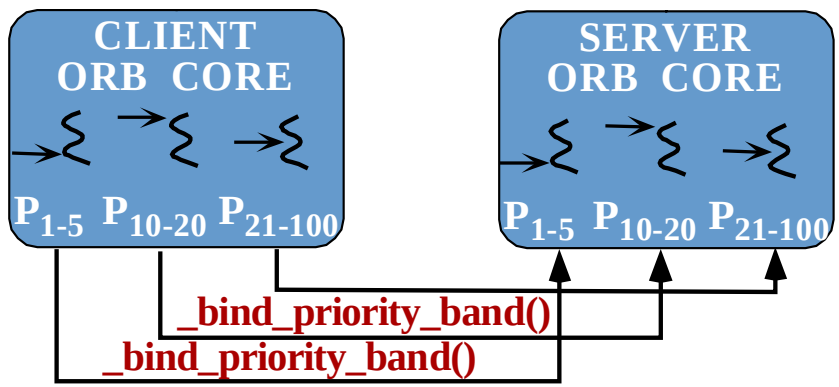
Proposed Solution → ReDaC Execution Platform

- **RT-CORBA Features Leveraged by ReDaC Execution Platform**
  - *Priority Model*
    - Use "ClientPropagated" policy to propagate client request priorities
  - *ThreadPool Model*
    - Use "ThreadPoolWithLanes" policy serve client requests concurrently
    - The "lane" feature offers priority partitioning to avoid priority inversion
  - *Connection Model*
    - Use PriorityBanded policy to differentiate service request based on priority

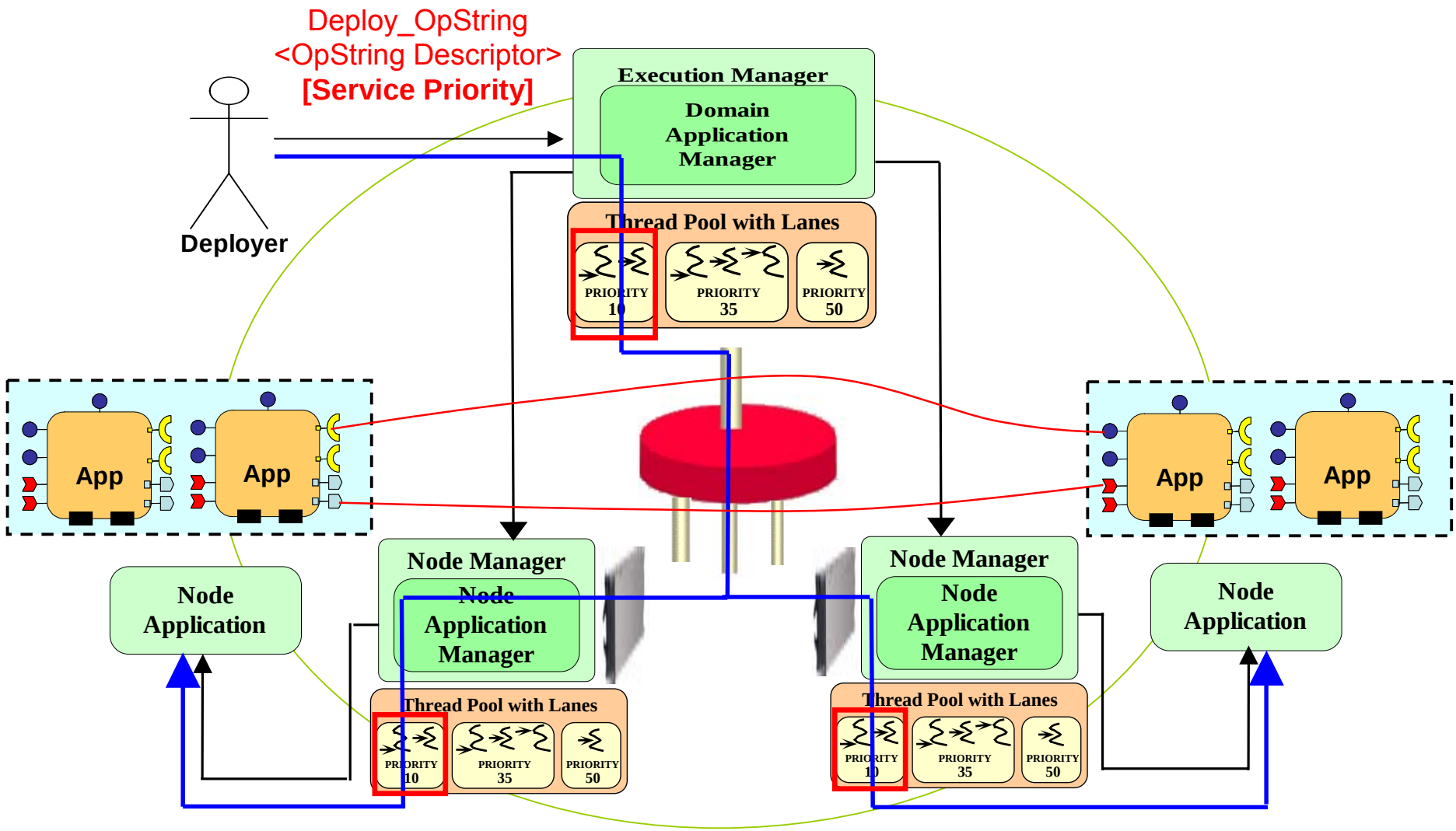
Solaris Priority 136

LynxOS Priority 128





ReDaC Execution Platform



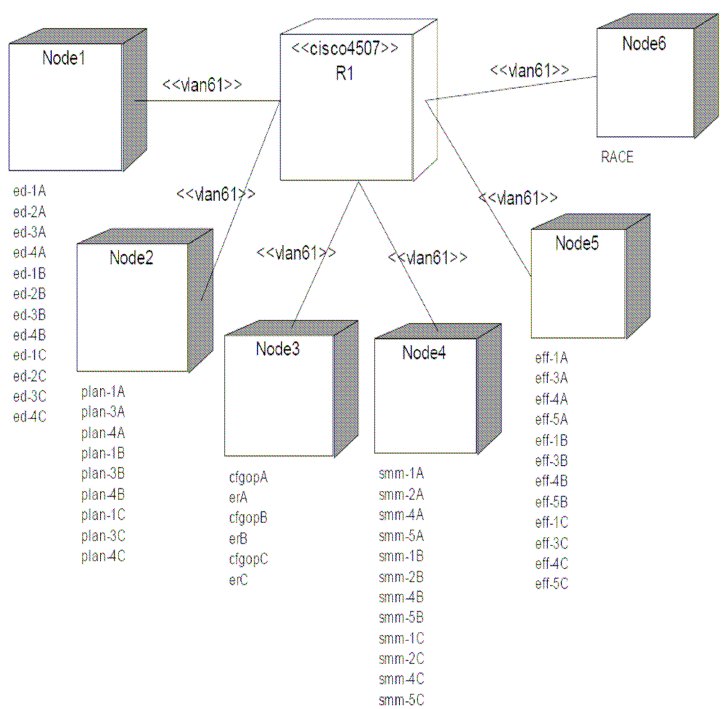
Criteria for Evaluation

Performance Criteria For Reconfiguration (Hypothesis)

- Baseline – Current DARPA ARMS GT4 Testbed (6 operational strings, with priority values of 3, 2, 2, 2, 1, 1, & system workloads are high)
- Metric M2 – *Highest priority operational strings mission effective loss*

$$M2 = \frac{\sum DownTime(S_{max})}{TotalTime}$$

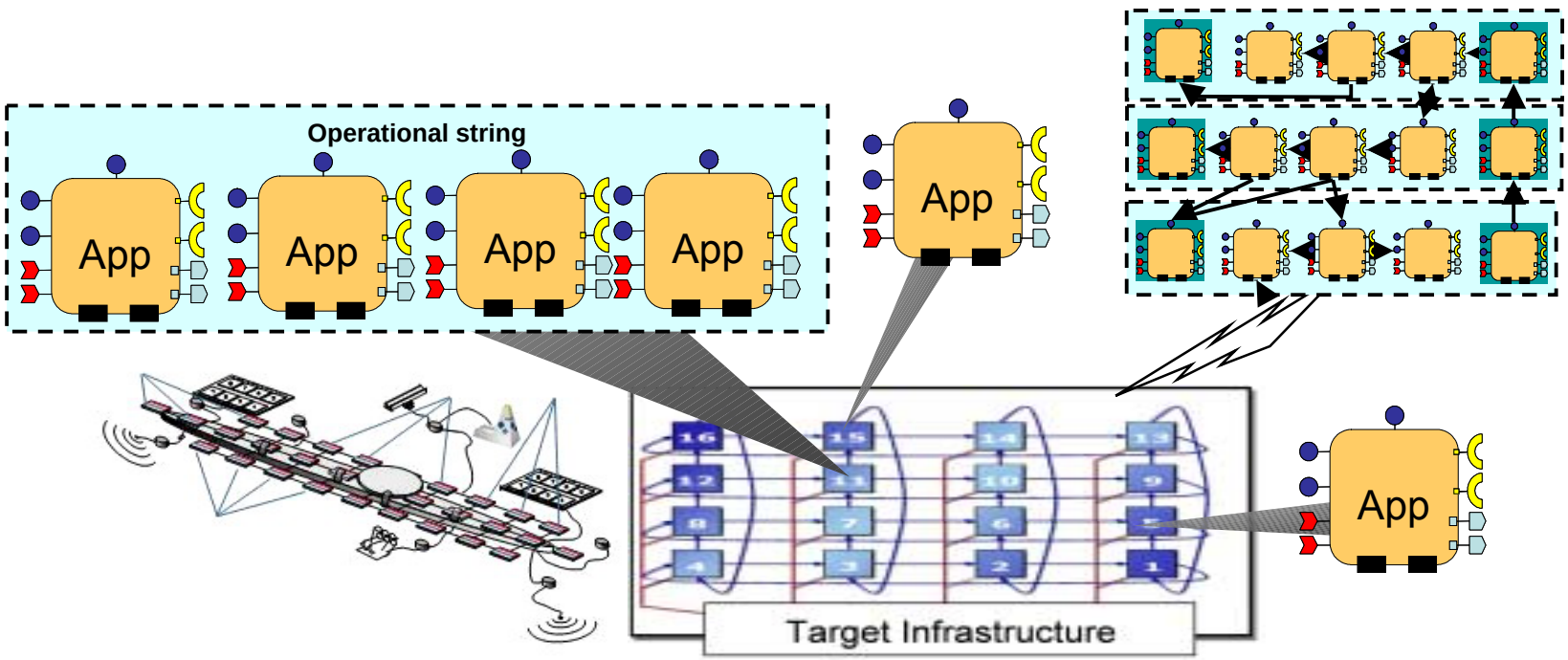
1. Determine when the first operational string with the highest priority goes down and obtain that timestamp.
2. Subtract that from the timestamp when all operational strings with the highest priority were first up.
3. Repeat if another operational string with the highest priority goes down.
4. Sum up all highest priority operational down time.
5. Divide it by the total experimental measurement time



Goal: Reduce M2 value by 20-30%

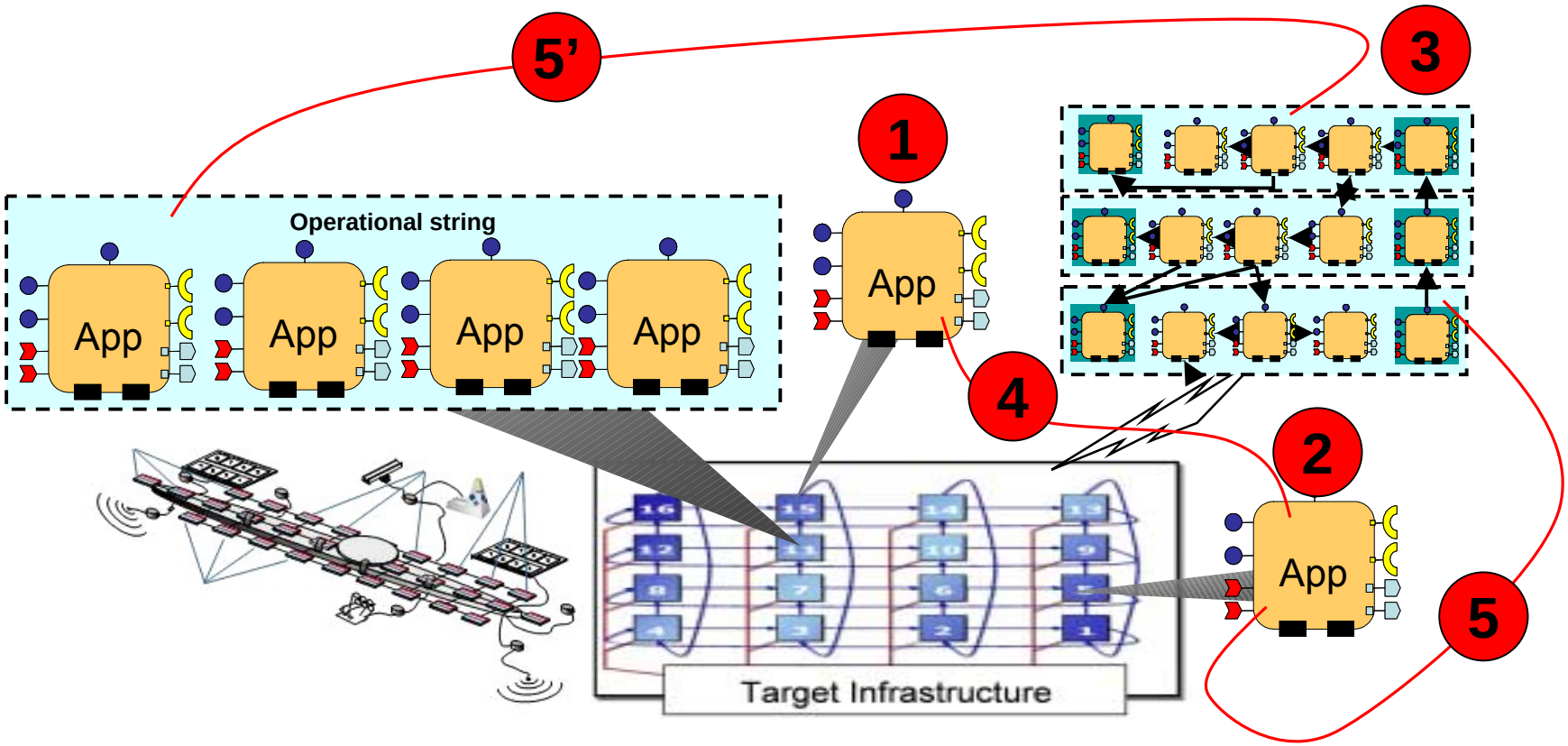
Challenge 3: Ad Hoc Techniques for Planning the System Reconfiguration Process

- **Context**:
  - Enterprise DRE system reconfiguration process is usually composed of a number of subtasks



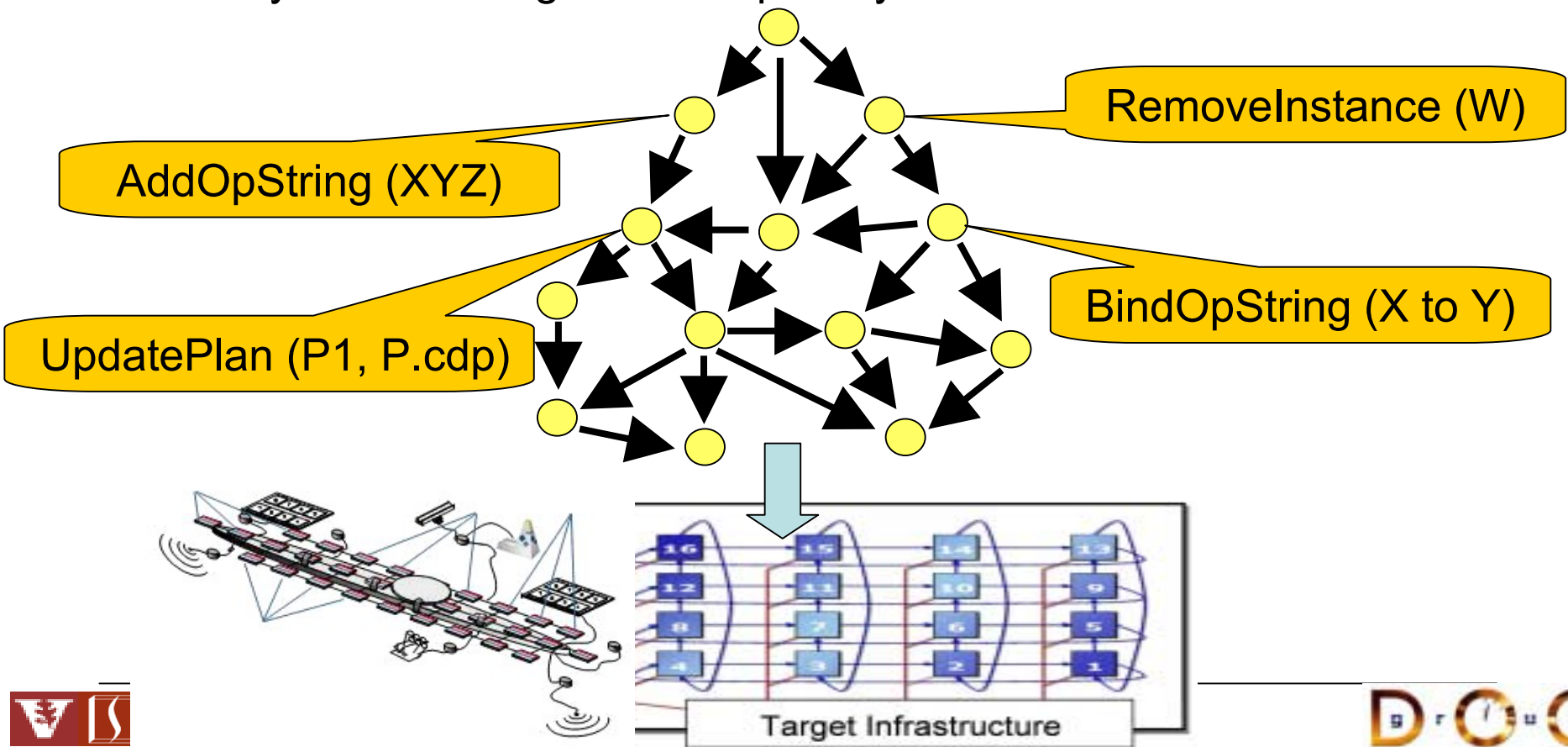
Ad Hoc Techniques for Planning the Reconfiguration Process

- **Problem**:
  - How to allow the reconfiguration process to managed from end-users' perspective?
  - How to specify the causal relationship among various subtasks?



Solution → Reconfiguration Modeling Language (ReML)

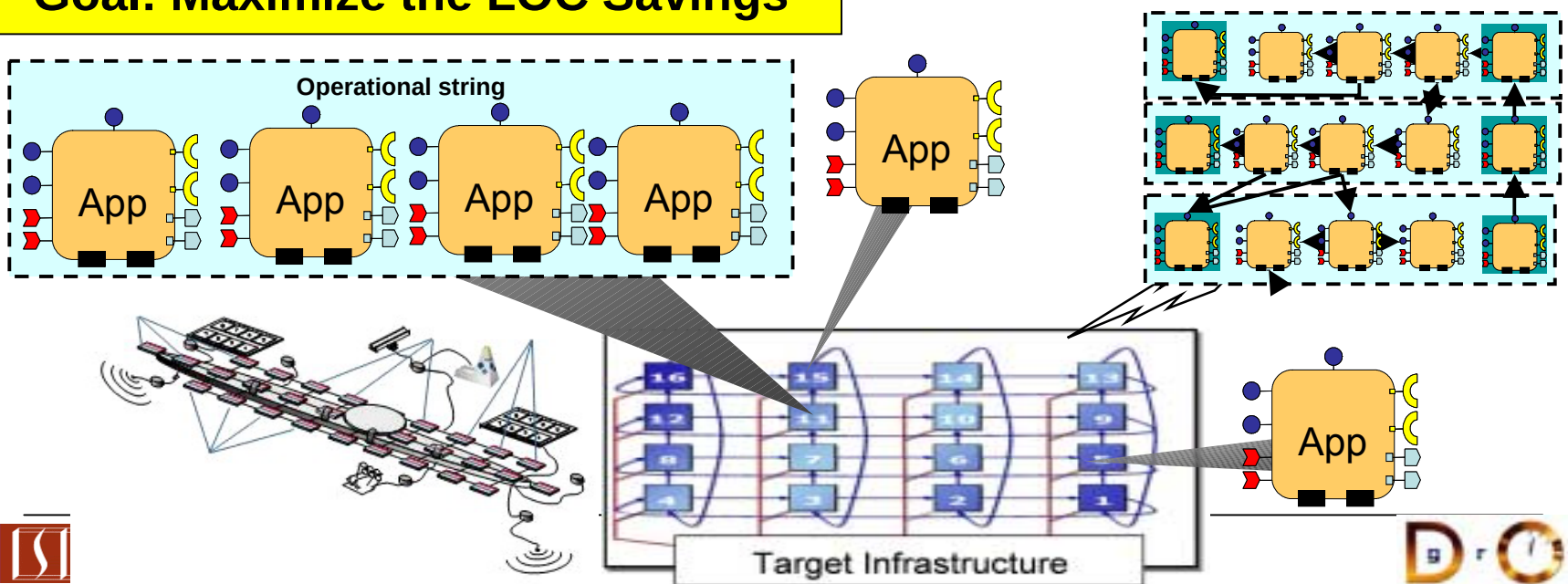
- **Develop a ReDaC Programming Model called ReML:**
  - Allow end-users to declaratively specify the workflow of the reconfiguration process through ReML, a visual DSML
  - Combine the ReML with ReDaC to provision full-fledged enterprise DRE system reconfiguration capability



Criteria for Evaluation

- The **Integration** of ReDaC Programming Model & Computational Model could reduce enterprise DRE systems reconfiguration effort **(Hypothesis)**
  - Metrics → Lines of Code (LOCs) reduced
  - M3 = LOC Saved Per Instance (Estimated ~200 LOC)
  - M4 = LOC Saved Per Connection (Estimated ~200 LOC)
  - M5 = LOC Saved Per Operational String (Estimated ~5,000 LOC for a size of 10 components & 15 connections)

Goal: Maximize the LOC Savings



Questions & Comments