Architectural and Optimization Techniques for Scalable, Real-time and Robust Deployment and Configuration of DRE Systems

Gan Deng
Douglas C. Schmidt
Aniruddha Gokhale

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee

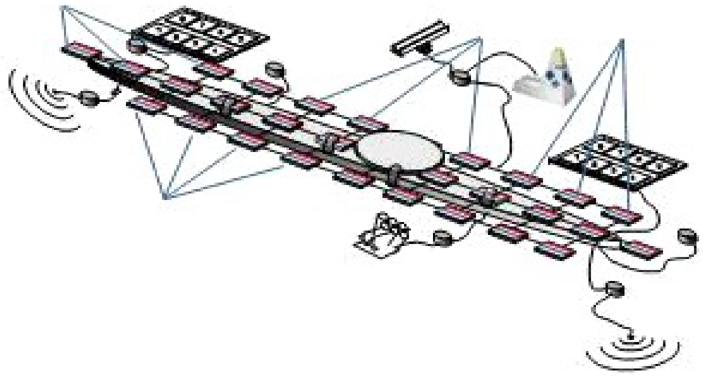
Background: Enterprise DRE Systems

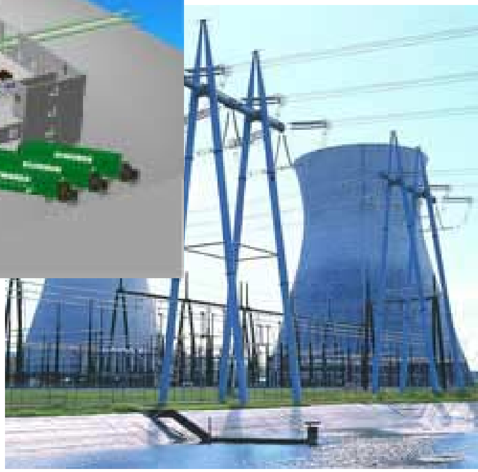
Key Characteristics

- 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/

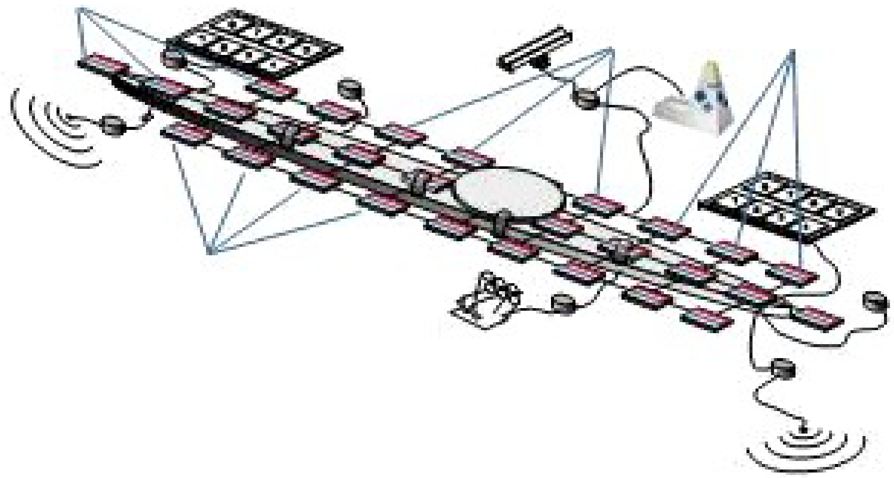




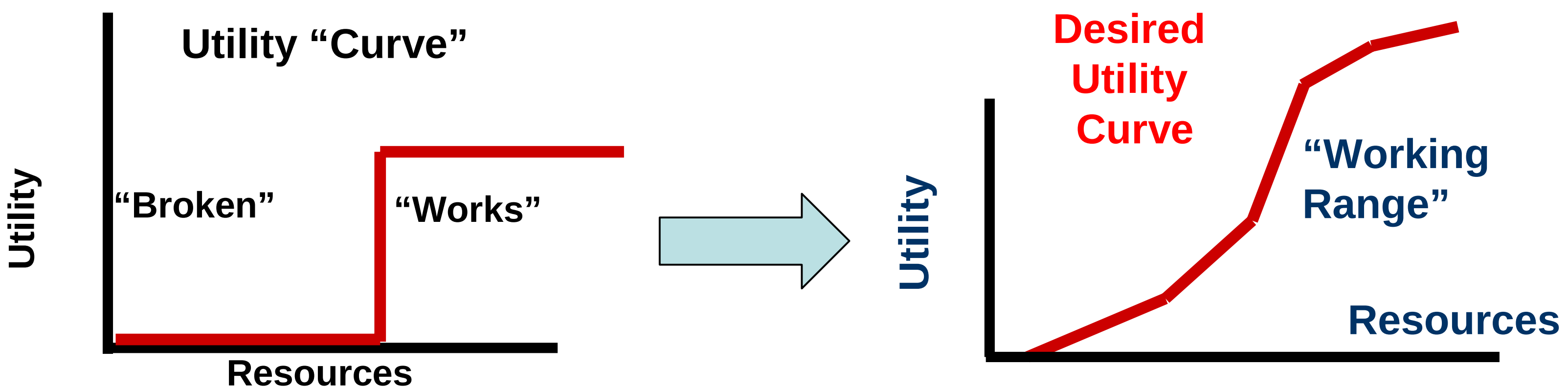
Demands of Enterprise DRE Systems

Key Challenges

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



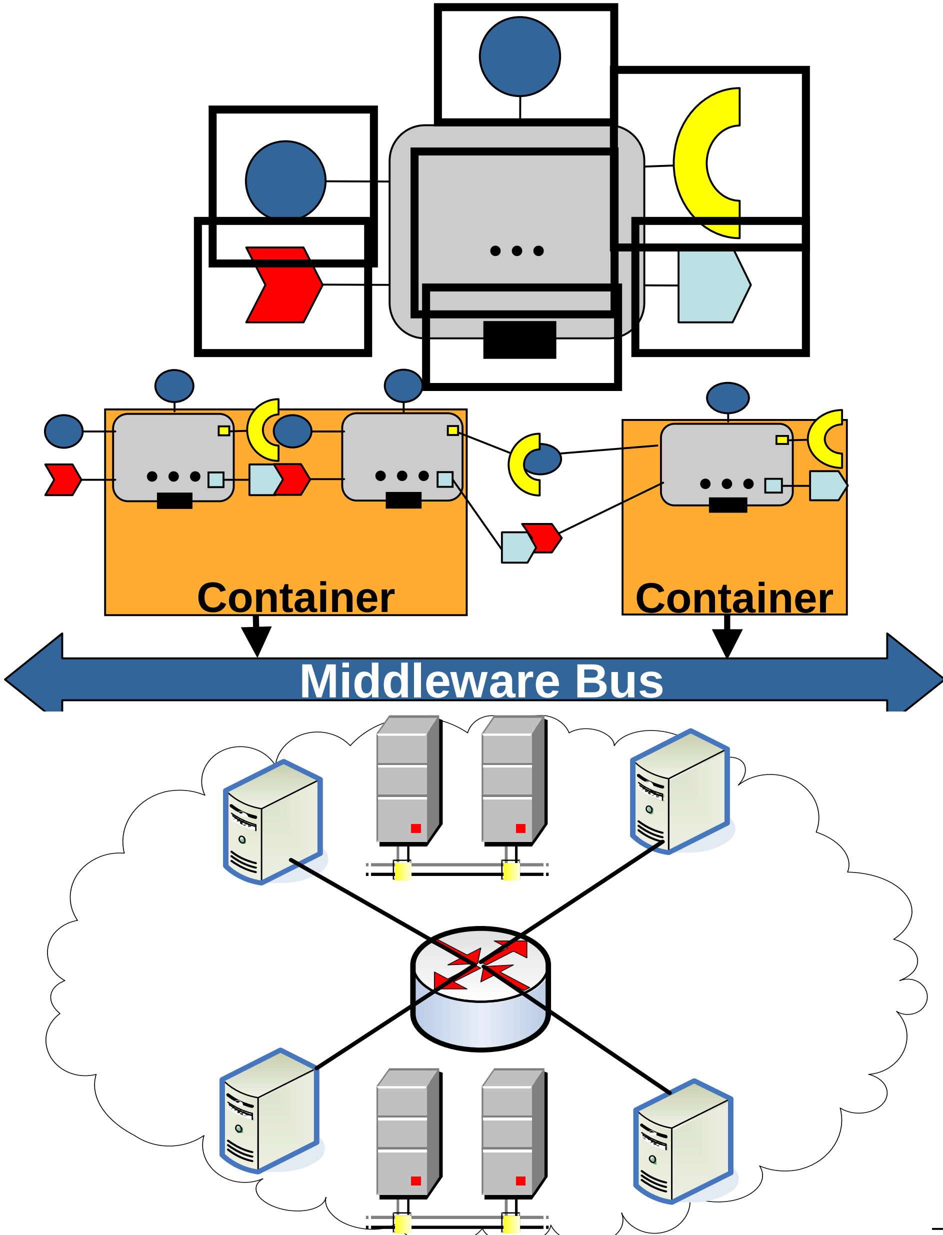
Enterprise DRE Systems



"Harder" Requirements

"Softer" Requirements

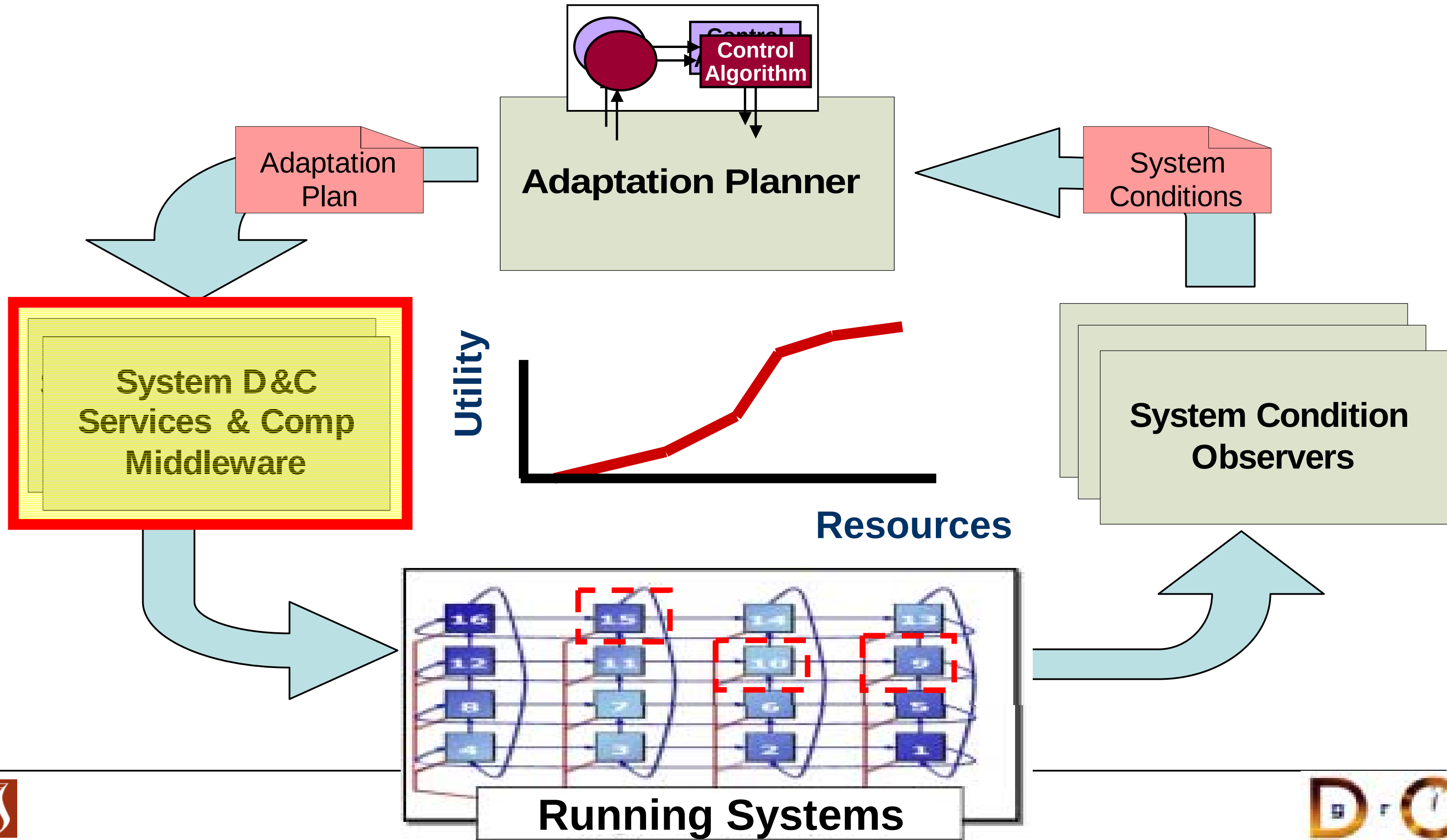
Promising Solution: Component Middleware



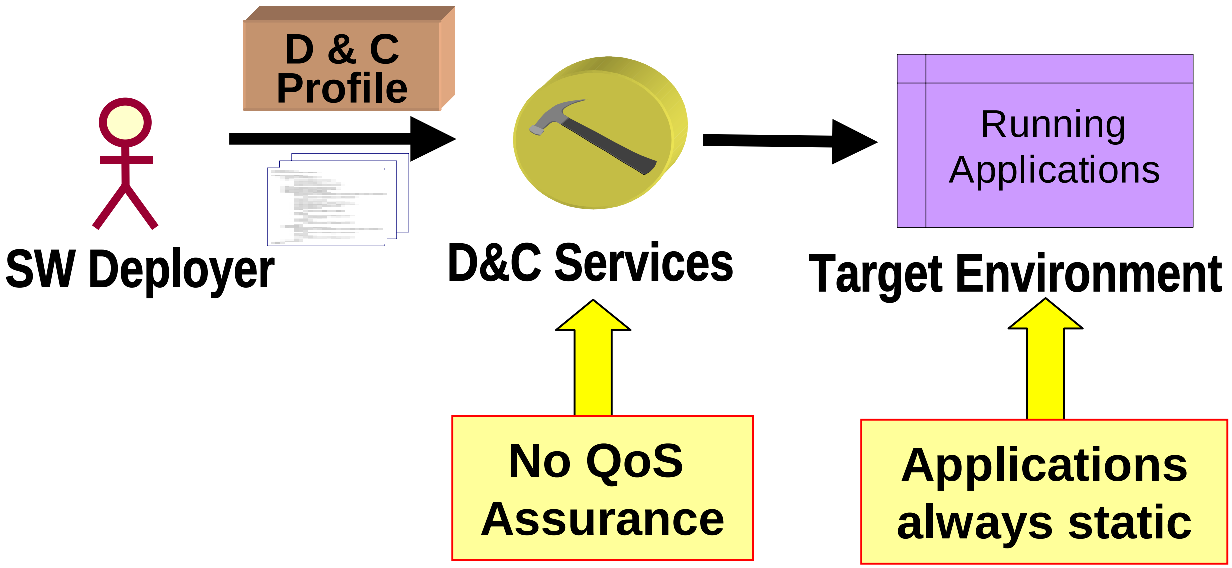
- *Components* encapsulate application “business” logic
- Components interact via *ports*
  - *Provided interfaces*, e.g.,facets
  - *Required connection points*, e.g., receptacles
  - *Event sinks & sources*
  - *Attributes*
- *Containers* provide execution environment for components with common operating 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

Dynamic 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

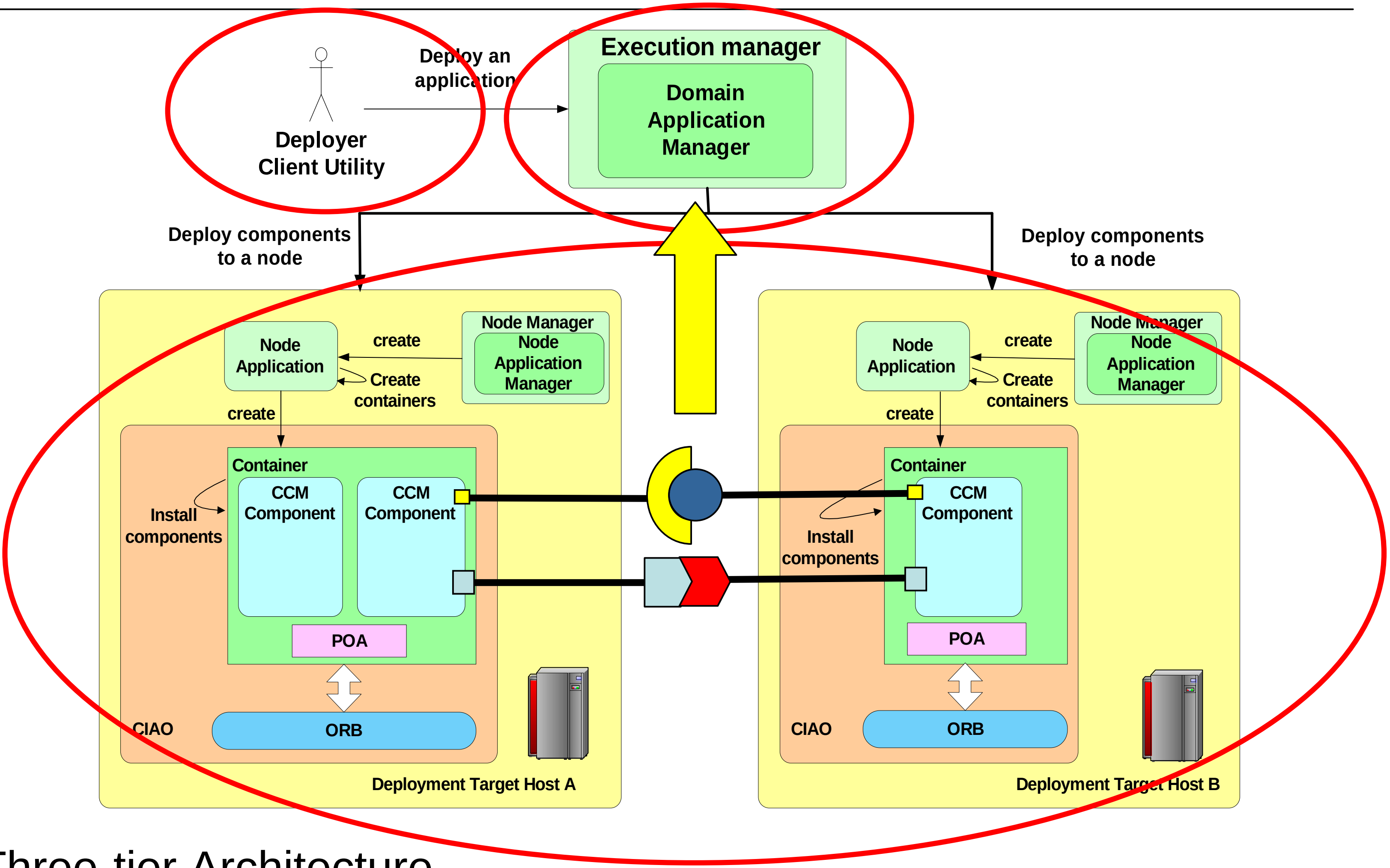


Limitations with Existing D&C Model



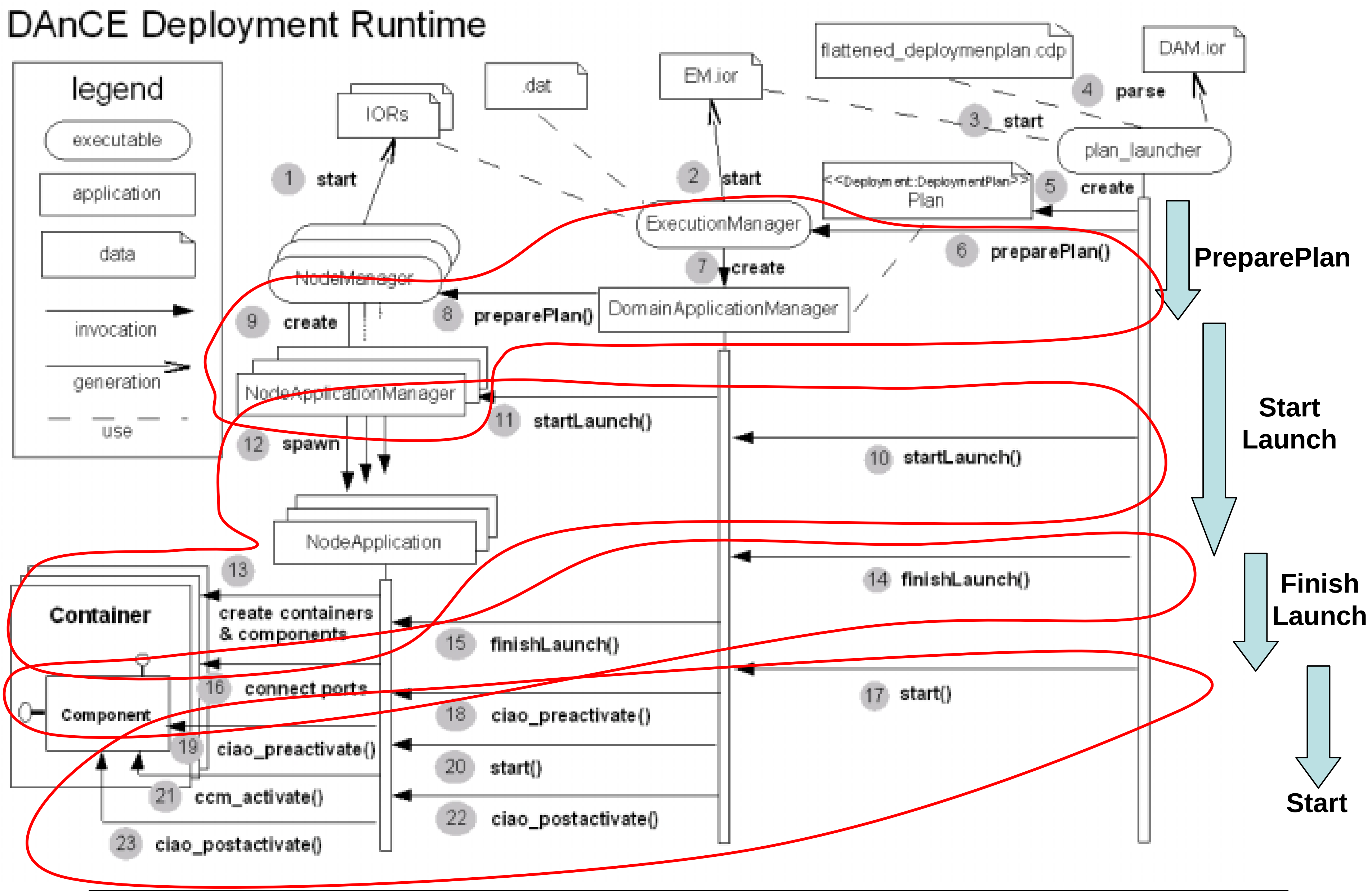
- The existing OMG D&C model cannot change the configuration once an application is deployed
  - Must shutdown the entire application & redeploy, which is not feasible for enterprise DRE systems
- The existing OMG D&C model doesn't address the QoS issues when performing (re)deployment & (re)configuration
  - Scalability (e.g., large # of nodes)
  - Predictability (e.g., differentiate priorities)
  - Robustness (e.g., mask partial failure)

Overview of D&C Architecture Model



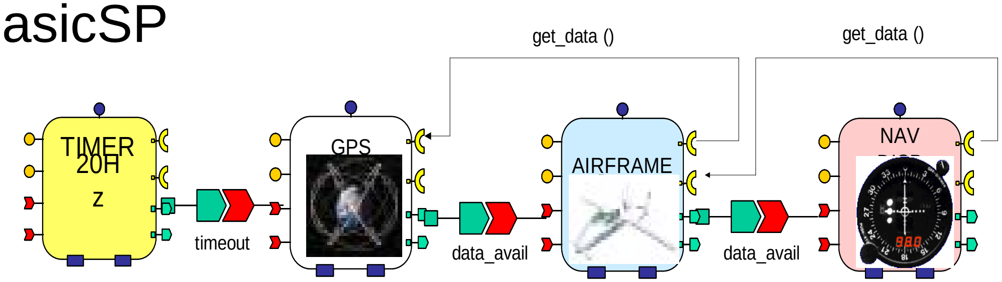
- Three-tier Architecture
- Different nodes need to coordinate with each other
- Central controller coordinates different objects distributed across many different nodes

Overview of D&C Service Runtime

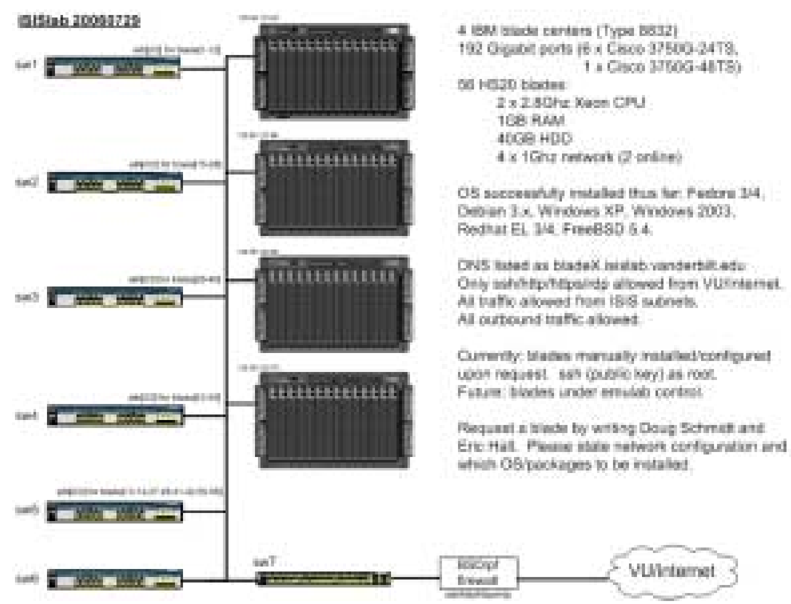


Experiments with D&C Implementation

- Run D&C Experiments on two different CCM-based applications
  - Small Application – Boeing BasicSP Scenario (4 components)
  - Large Application – up to 1000 components in total

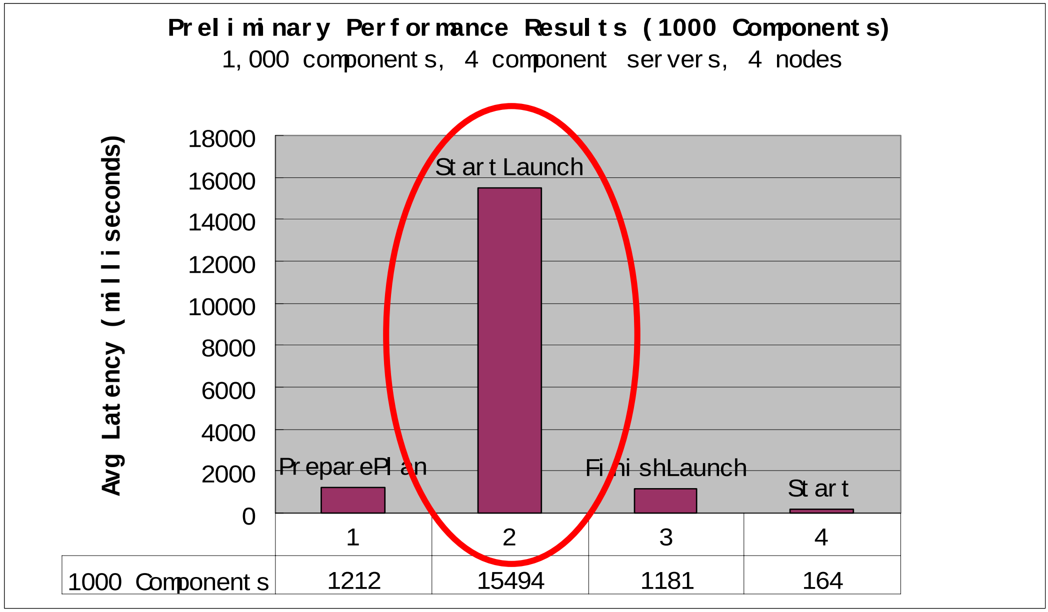


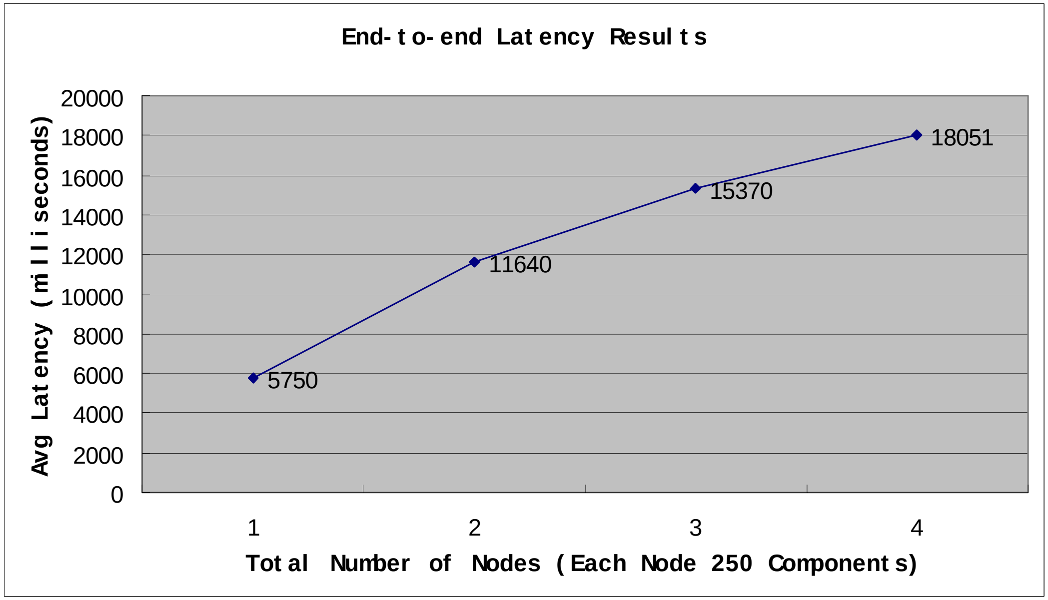
- Experiments performed in the ISIS Lab
  - Dual 2.8 GHz Xeon CPUs, 1GB of ram, 40GB HDD, and gigabit ethernet cards
  - Real-time Fedora Core 4 Linux kernel version 2.6.20-rt8-UNI
  - 5 nodes were used with 4 of them a deployment target for the applicatio and one as central controller runnin ExecutionManager (middle tier server)



Performance Benchmarking Results

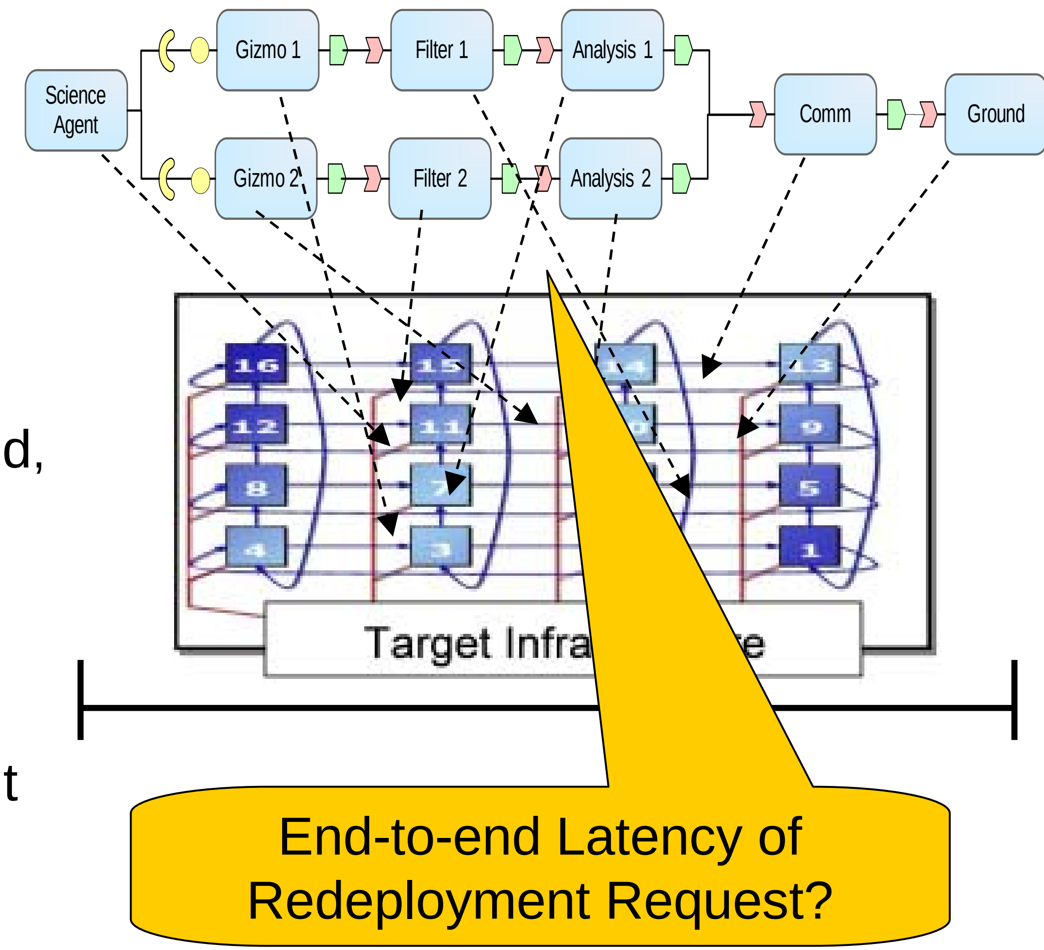
- The majority of latency is incurred during the StartLaunch phase (~85% of total)
  - NAM spawns NA component servers dynamically
  - NA creates containers and set up container policies accordingly
  - Container loads component DLLs dynamically
  - Container creates Homes, Instances & activates port objects, i.e., facets and event consumers
- When the application is scaled up, the end-to-end latency performance is linear to the total number of nodes and total number of components



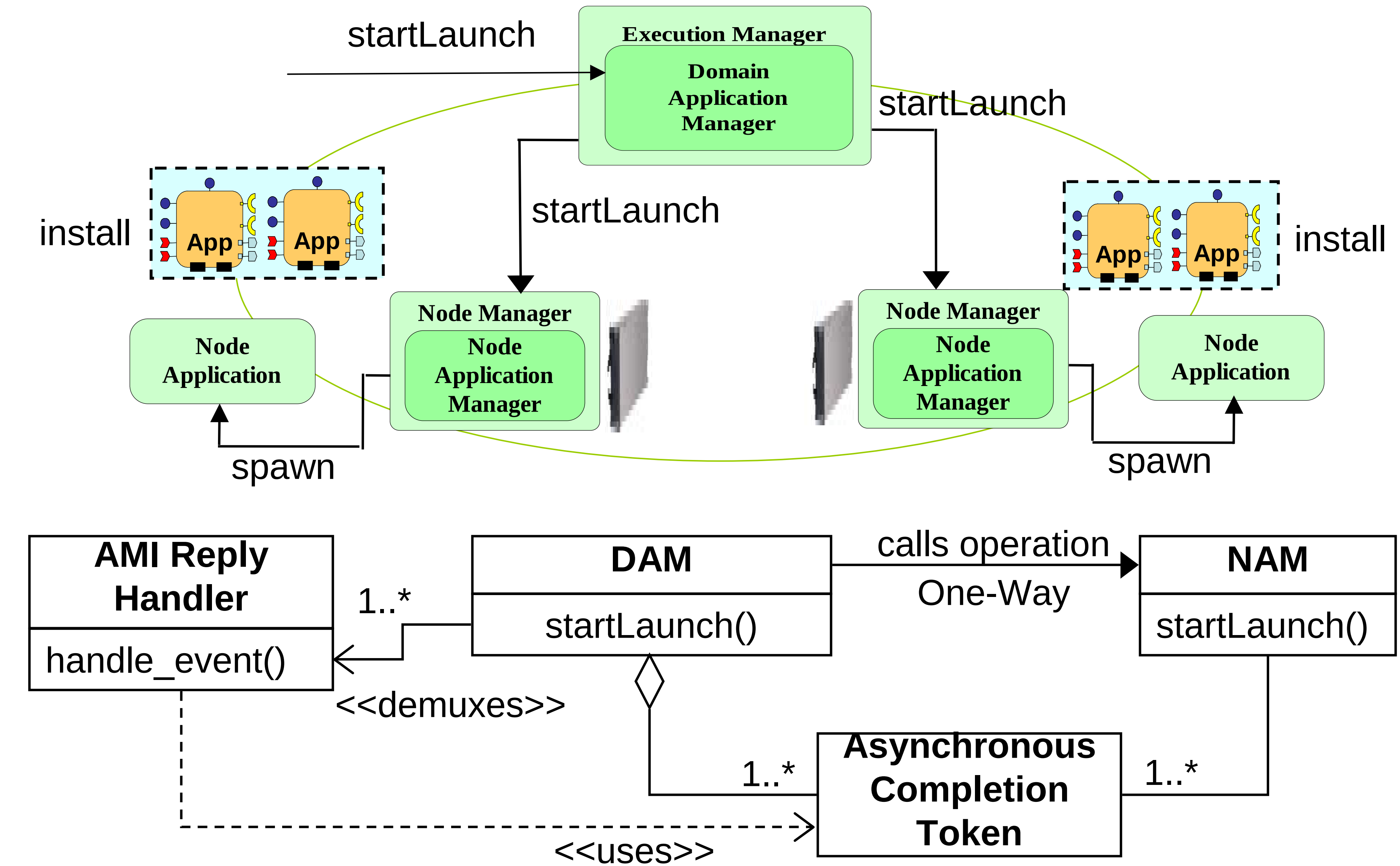


Challenge 1: How to Ensure Scalability?

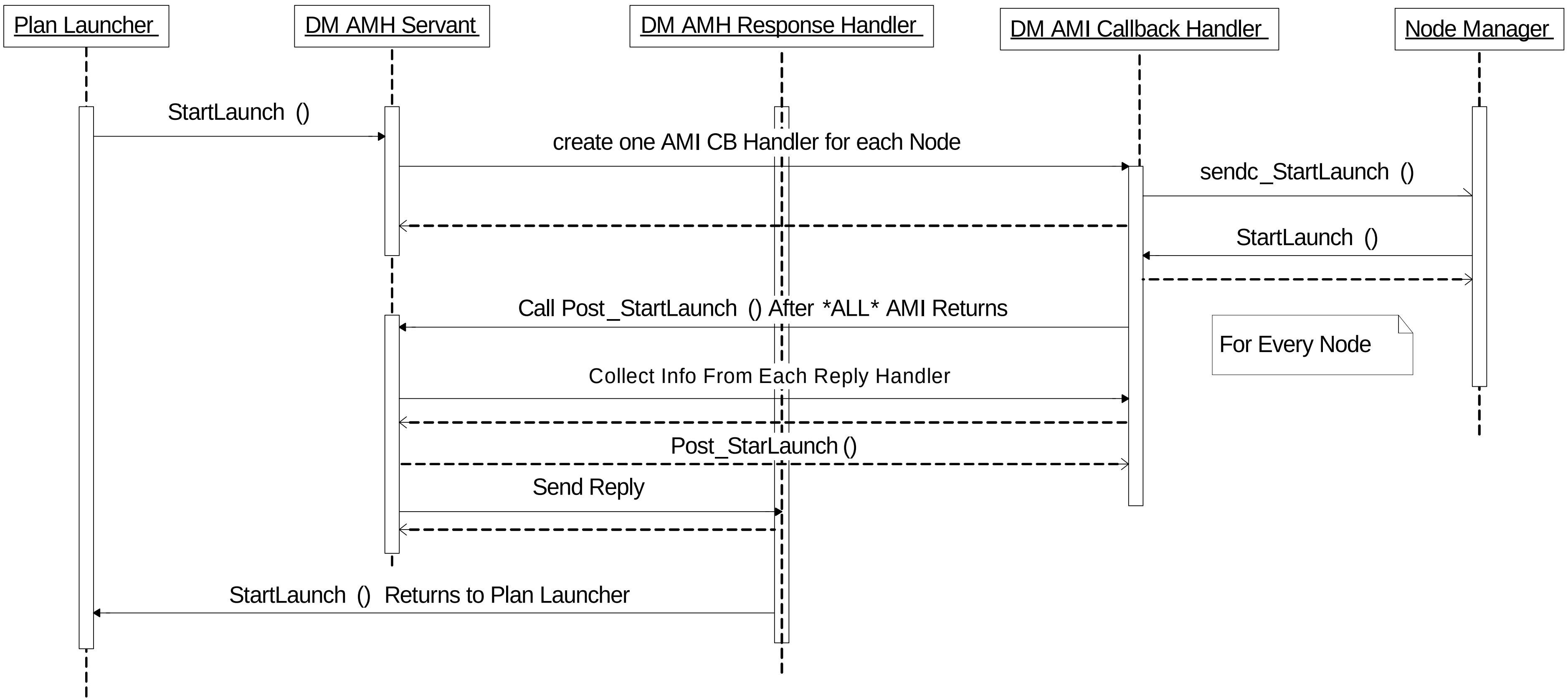
- **Context**
  - Enterprise DRE systems may have hundreds or even thousands of components
  - Component deployment is a complex task:
    - D&C service objects across many (hundreds of) nodes
    - Many tasks must be serialized, e.g., preparePlan, startLaunch, finishLaunch, start, etc.
- **Problem**
  - How to ensure the scalability of handling a single (re)deployment w/ many components & nodes
  - How to ensure the scalability of handling many concurrent (re)deployments



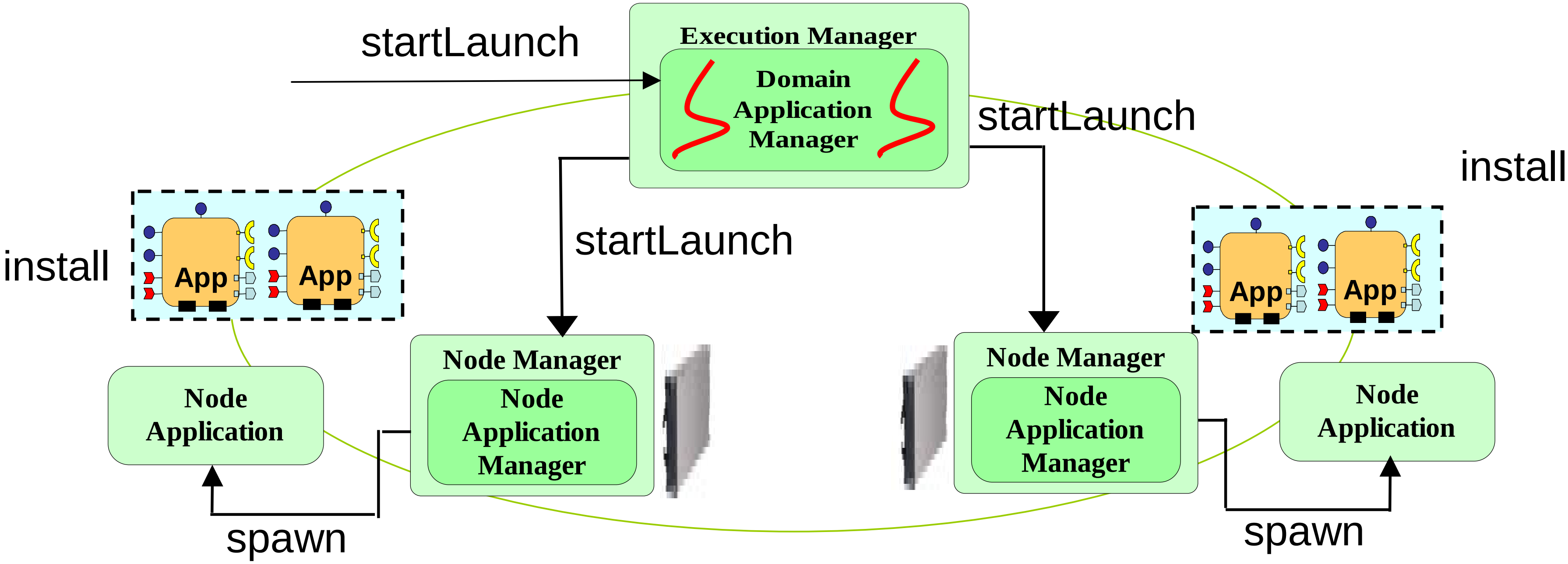
Addressing Scalability Requirement for Single (Re)Deployment → Parallel Processing via AMI/AMH



Addressing Scalability Requirement for Single (Re)Deployment → Parallel Processing via AMI/AMH

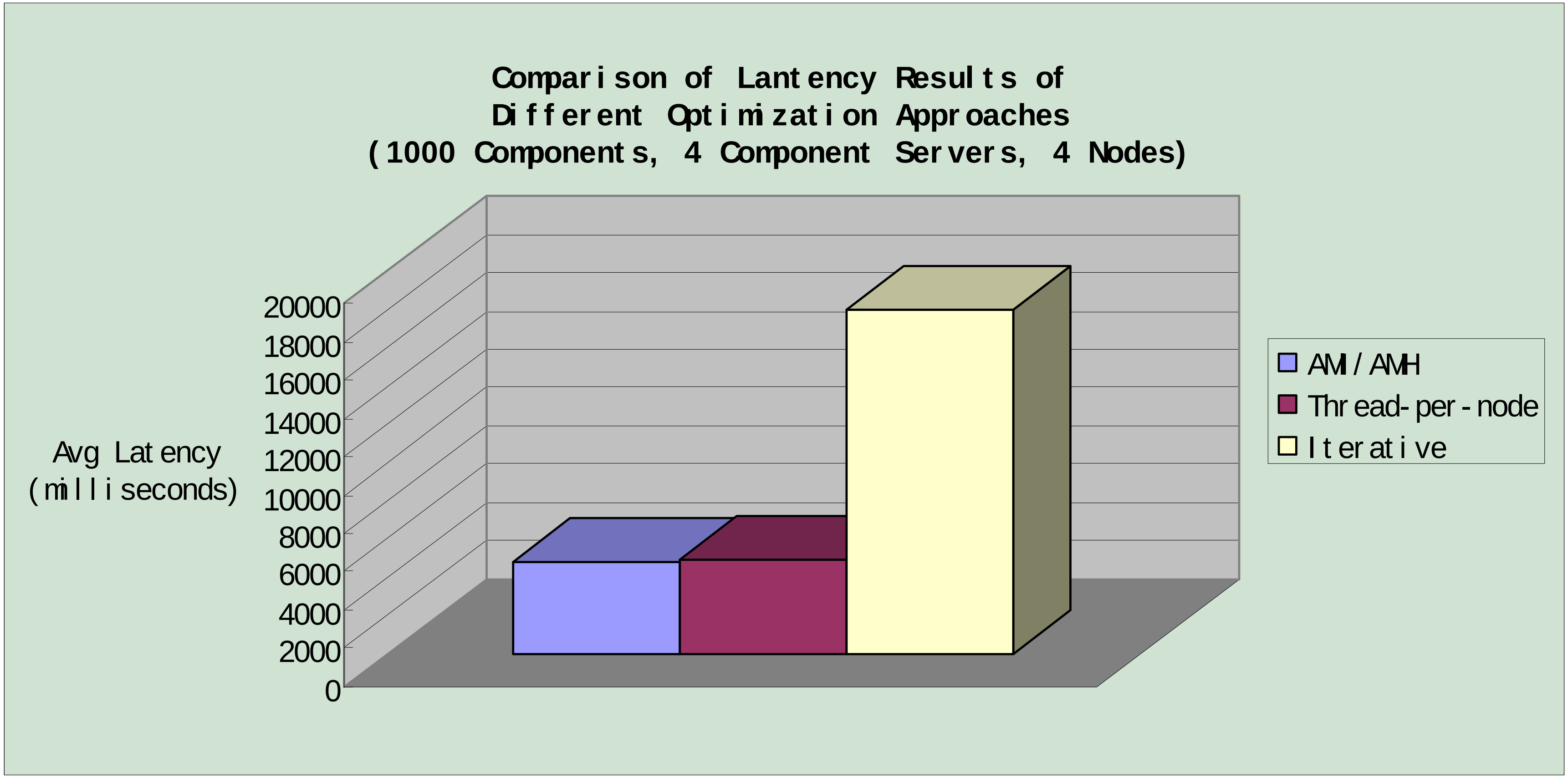


Addressing Scalability Requirement for Single (Re)Deployment → Thread-per-Node Model



- Context-aware threading model
  - ExecutionManager knows which nodes are involved for the particular (re)deployment request by analyzing the deployment plan input
  - DomainApplicationManager spawns a number of threads dynamically with one thread corresponding to one node
  - Each thread has its own execution context, i.e., component installation information on that particular node
  - Each thread makes a two-way synchronous call to install components

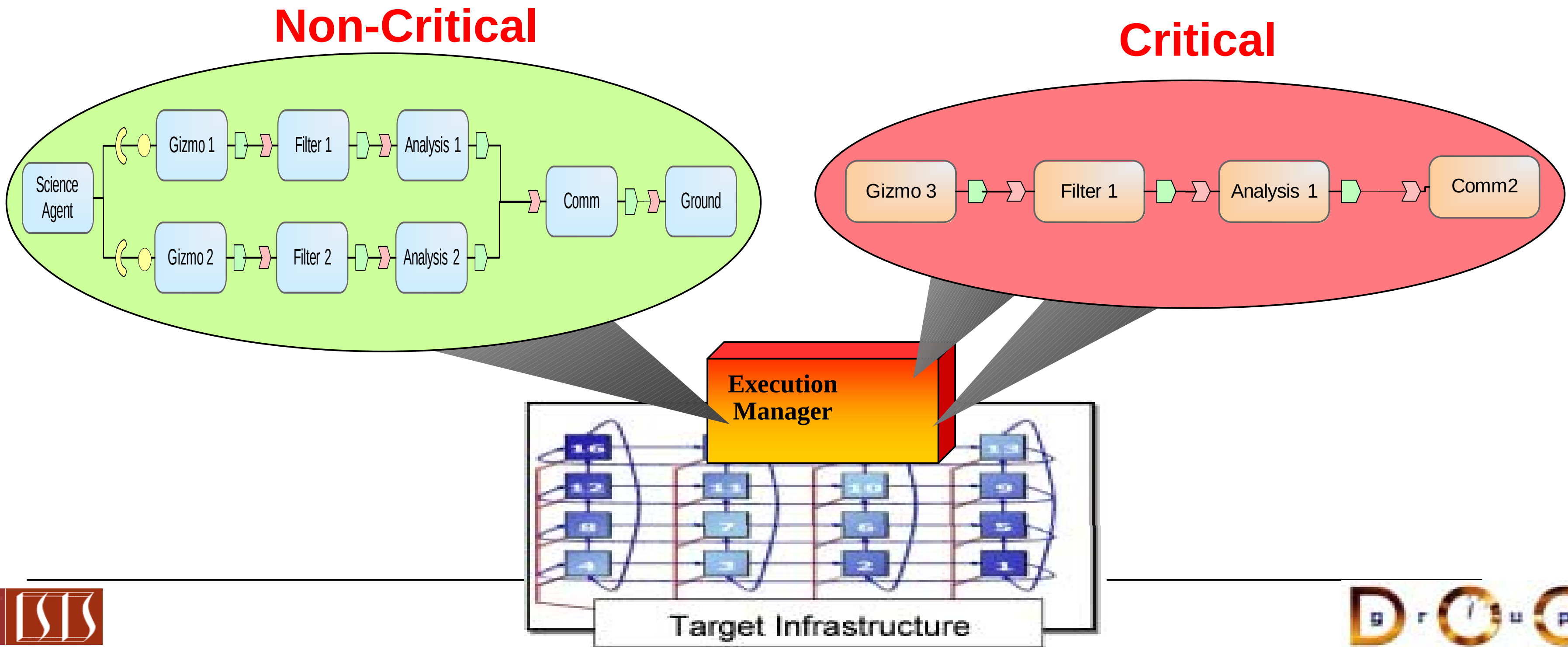
Performance Results



- Both AMI/AMH & Thread-per-node reduces the end-to-end latency by about 73% percent of the iterative approach
- This is because both optimization approaches can take advantage of the parallel processing capabilities of all the 4 different nodes

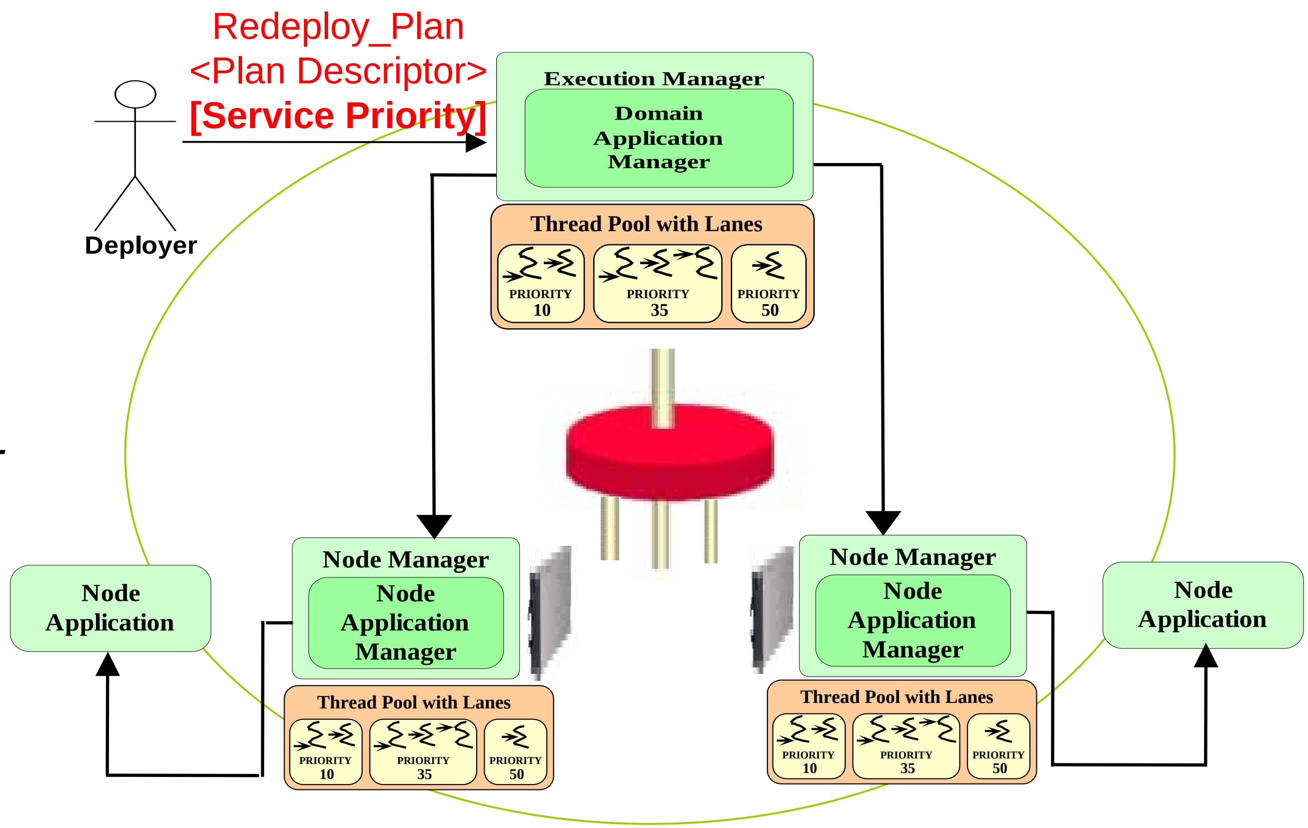
Challenge 2: How to Ensure Predictability?

- **Context**
  - Multiple redeployment requests can be invoked on ExecutionManager simultaneously from different clients
  - Some of the requests are targeted on different deployment plans
  - Some of the requests are targeted on the same deployment plan
- **Problems**
  - How to maximize concurrent processing while also differentiate services from different requests based on their priorities?



Solution → Applying RT-CORBA Features

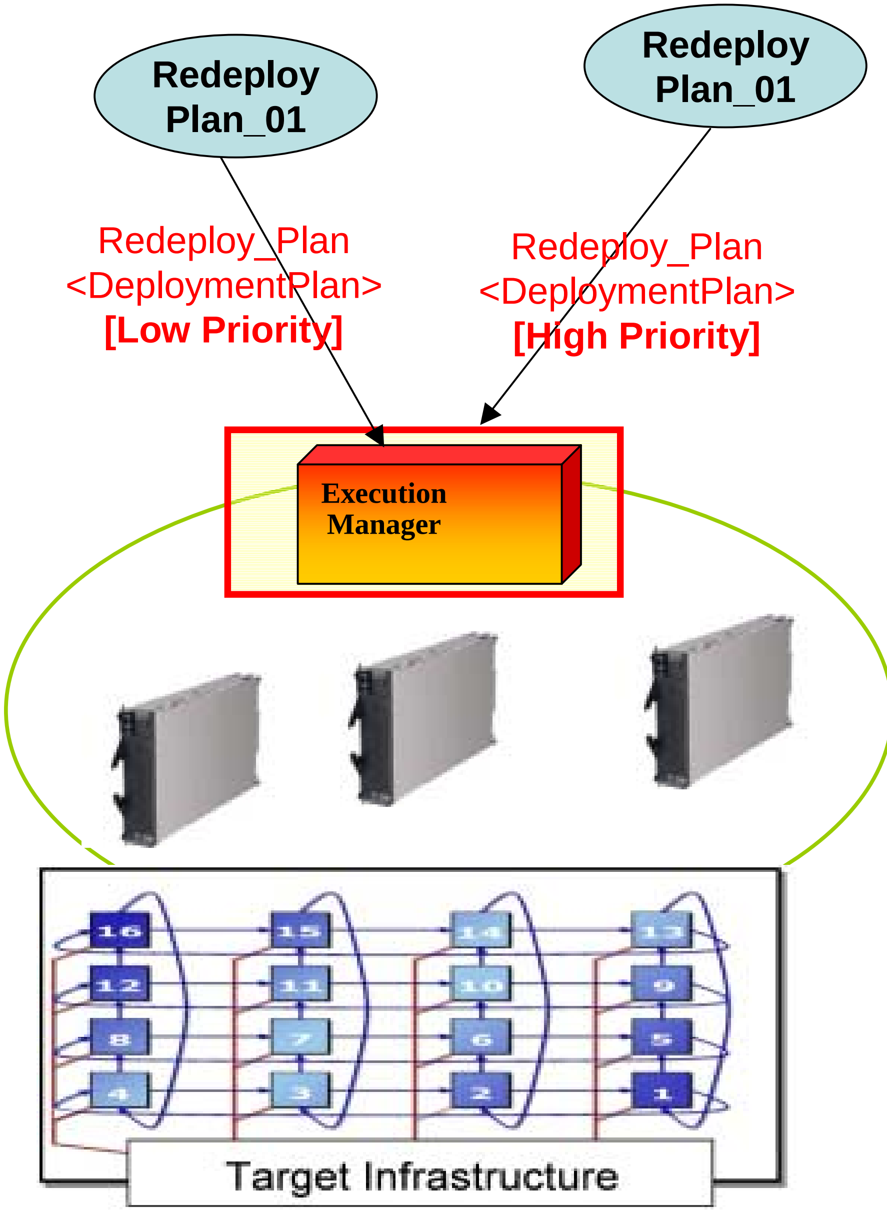
- Apply RT-CORBA features to differentiate services based on priority
  - All D&C objects (e.g., EM, DAM, NM, NAM) are configured with *RT CORBA thread-pool-with-lanes* concurrency model & Client_Propagated priority model



  - The number of lanes, number of static threads, number of dynamic threads & lane priorities are configurable when D&C services are initialized
  - When external clients request setting up & modifying services, the priority level could also be specified as part of the request

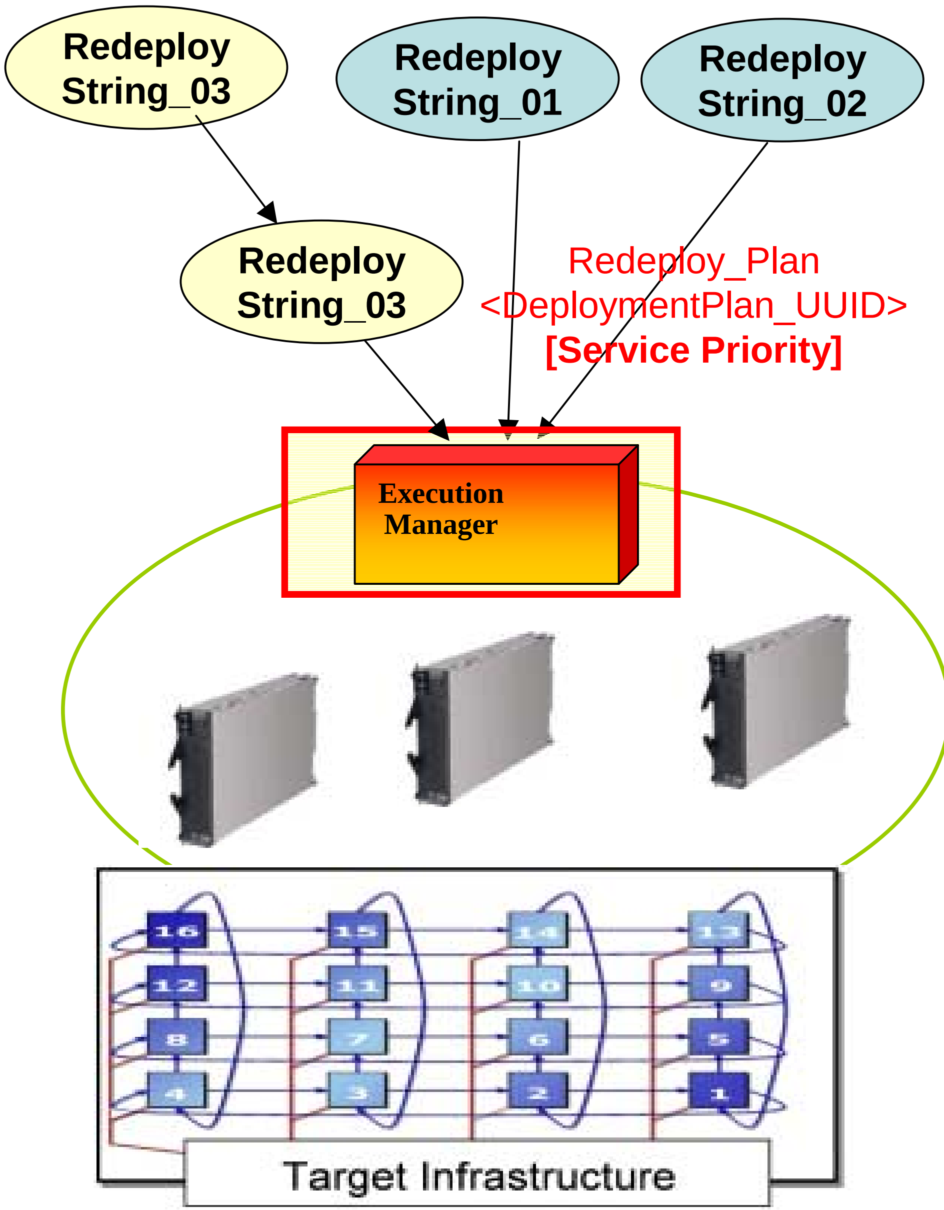
Solution → Admission Control + Differentiate Services

- Applying RT-CORBA concurrency model naively may render application into invalid state
  - Each redeployment request comes with a self-contained (new) deployment plan
  - Later redeployment request can preempt earlier redeployment request if later request has higher priority
  - When the earlier redeployment request is resumed, it will cause the application into invalid state

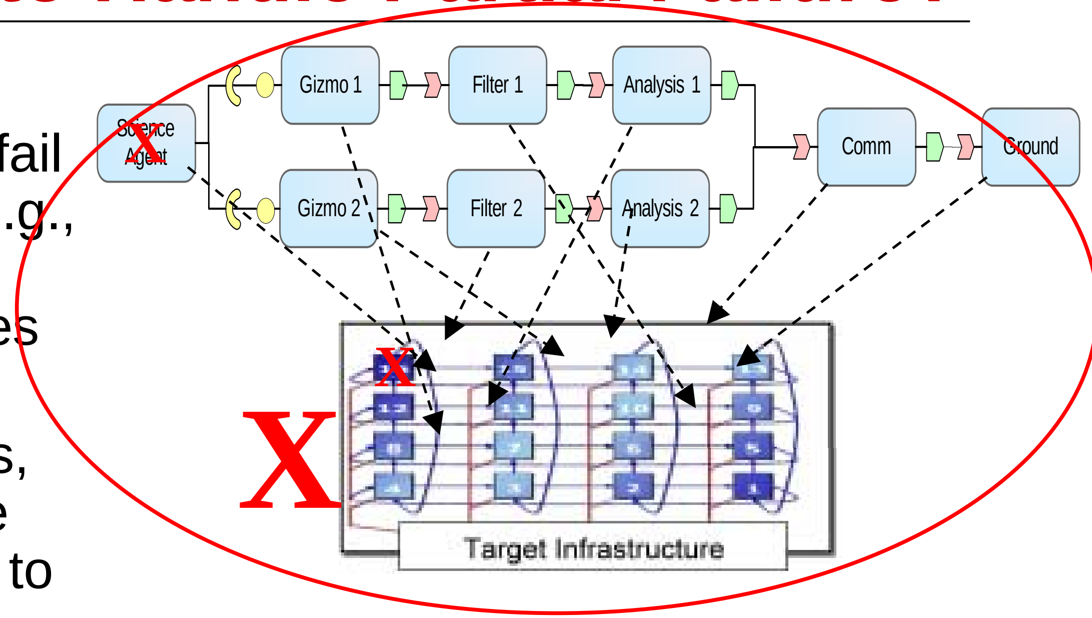


Solution → Admission Control + Differentiate Services

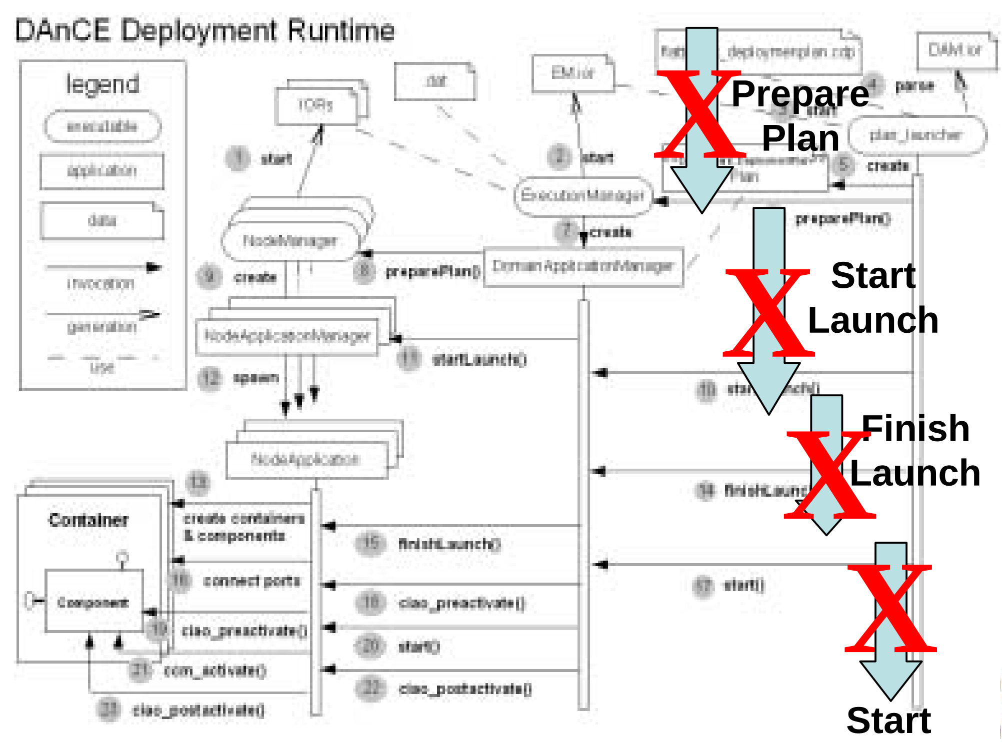
- Build an Admission Control Mechanism to Ensure Concurrent Processing on ExecutionManager
  - Concurrent processing is allowed for redeployment request on different deployment plan
  - Serialized processing must be ensured for redeployment request on the same deployment plan



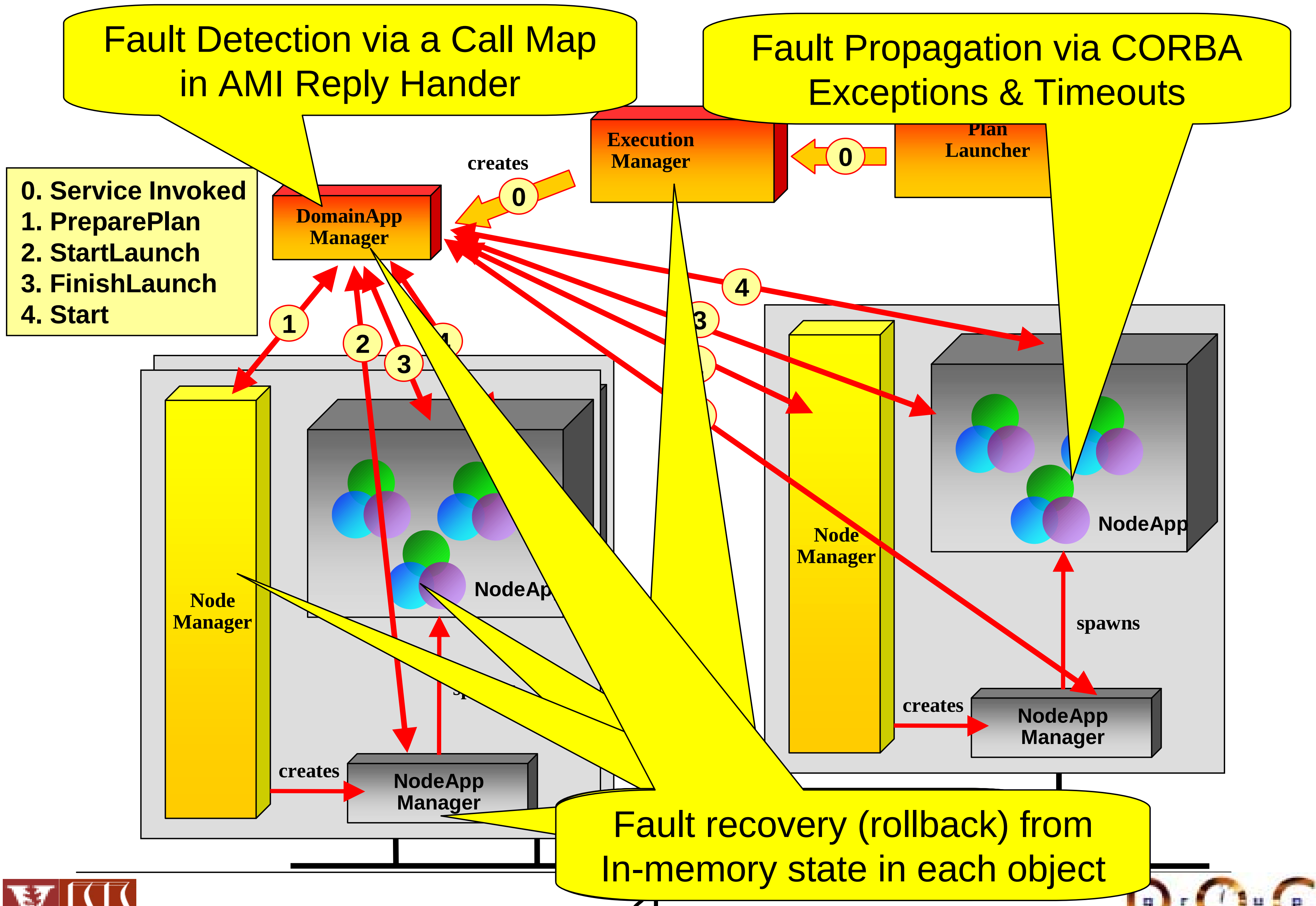
Challenge 3: How to Handle Partial Failure?



- **Context**
  - DRE system deployment can fail due to a number of reasons, e.g., lacking resources, lacking component DLLS, node failures
  - If some deployment failure happens in one or more nodes, the entire deployment must be rolled back to its original state to make the system remain in consistent state
- **Problem**
  - To recover from failure, we must address complexities arose from both *space*-dimension and *time*-dimension
  - Space-dimension → Failure on one node will affect other nodes
  - Time-dimension → Failure in later phase will affect previous phase



Solution → Robust Atomic Deployment



Concluding Remarks

- Dynamic redeployment & reconfiguration is essential to ensure the success of component middleware for enterprise DRE systems
- Existing D&C mechanisms lack support to ensure QoS of dynamic redeployment & reconfiguration
  - Scalability
  - Predictability
  - Robustness
- Architectural optimization techniques described in this presentation establish an effective guidance to address such limitations

Source code is available for download
www.dre.vanderbilt.edu/CIAO

Questions & Comments