



# **IQ-Services: Resource-Aware Middleware for Heterogeneous Applications**

**Karsten Schwan, Matt Wolf**

**CERCS**

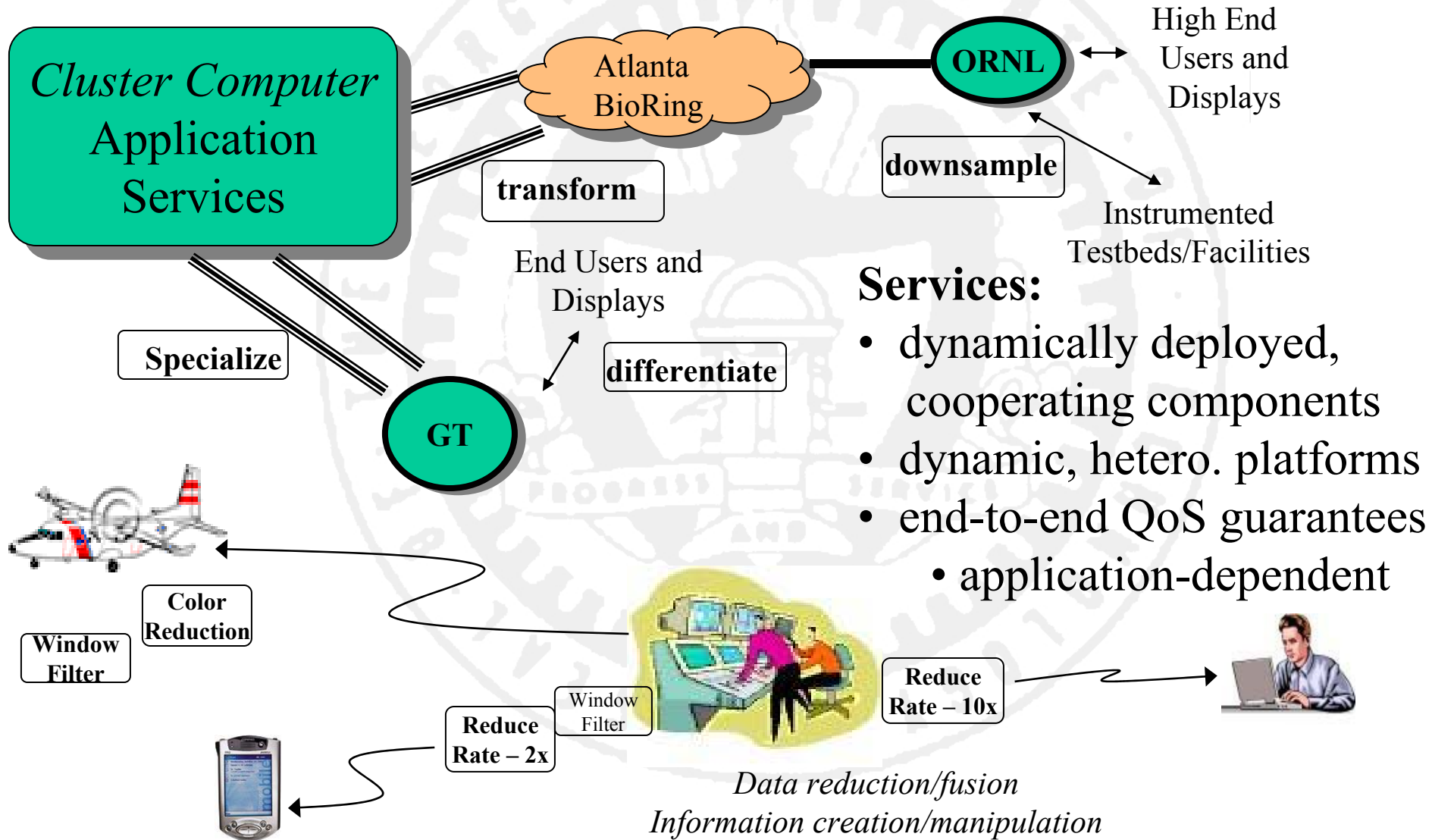
**Georgia Institute of Technology**

**Collaborators:**

**Tucker Balch, Constantinos Dovrolis, Greg Eisenhauer,  
Hsien-Hsin Lee, Santosh Pande, Calton Pu, Christian  
Poellabauer and other Ph.D. students**

**Nagi Rao, Qishi Wu - ORNL**

# Problem Statement



## Services:

- dynamically deployed, cooperating components
- dynamic, hetero. platforms
- end-to-end QoS guarantees
- application-dependent

*Data reduction/fusion*  
*Information creation/manipulation*



# Necessary Elements of Solution Approaches

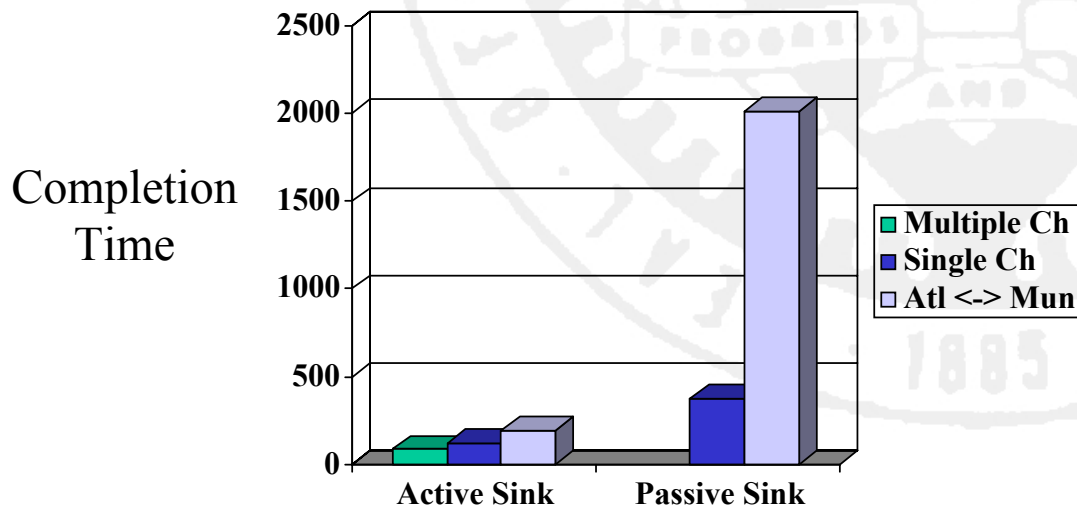
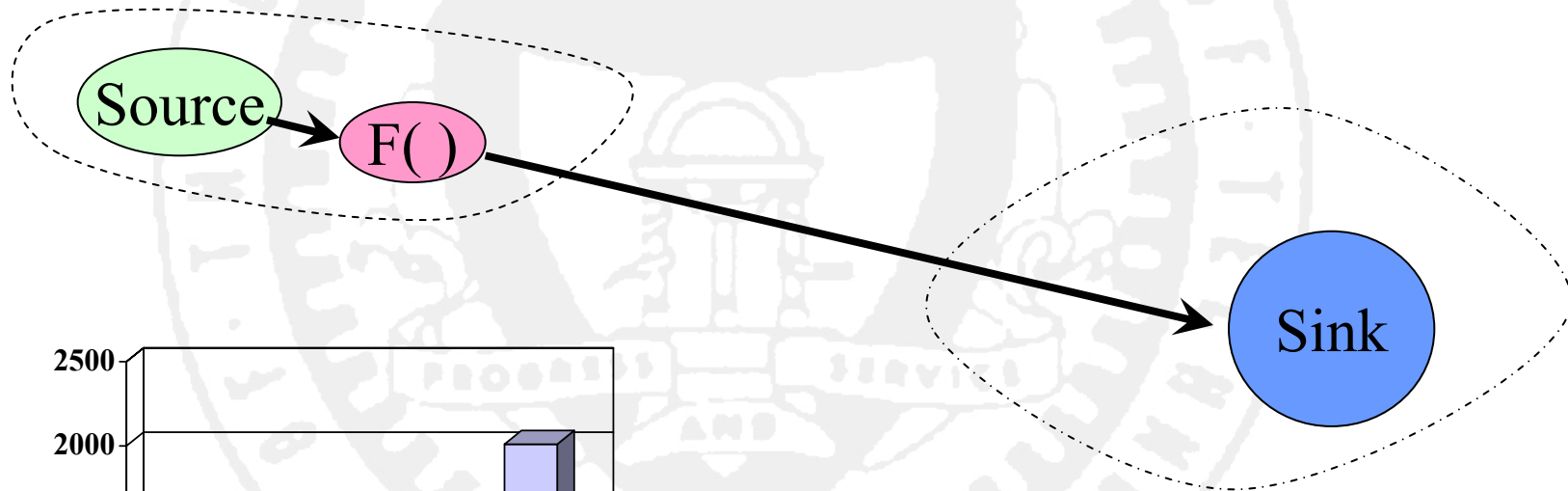
- **I. Middleware/Application Agility:**
  - dynamic service and code deployment (**DCG, dynamic compilation**)
  - code modification and adaptation, dynamic data conversion
  - dynamic overlays, ...
- **II. Resource- and Needs-Awareness:**
  - diverse metrics: bandwidth, power, trust, ...
  - changing end user needs, application behaviors
  - **dynamic monitoring:** system-level support
  - runtime management: multi-dimensional optimization vs. isolation
- **III. Open Infrastructures:**
  - ‘black box’ operating systems: any help from **VM technology**?
  - ‘closed’ networks: application-level services through programmability?



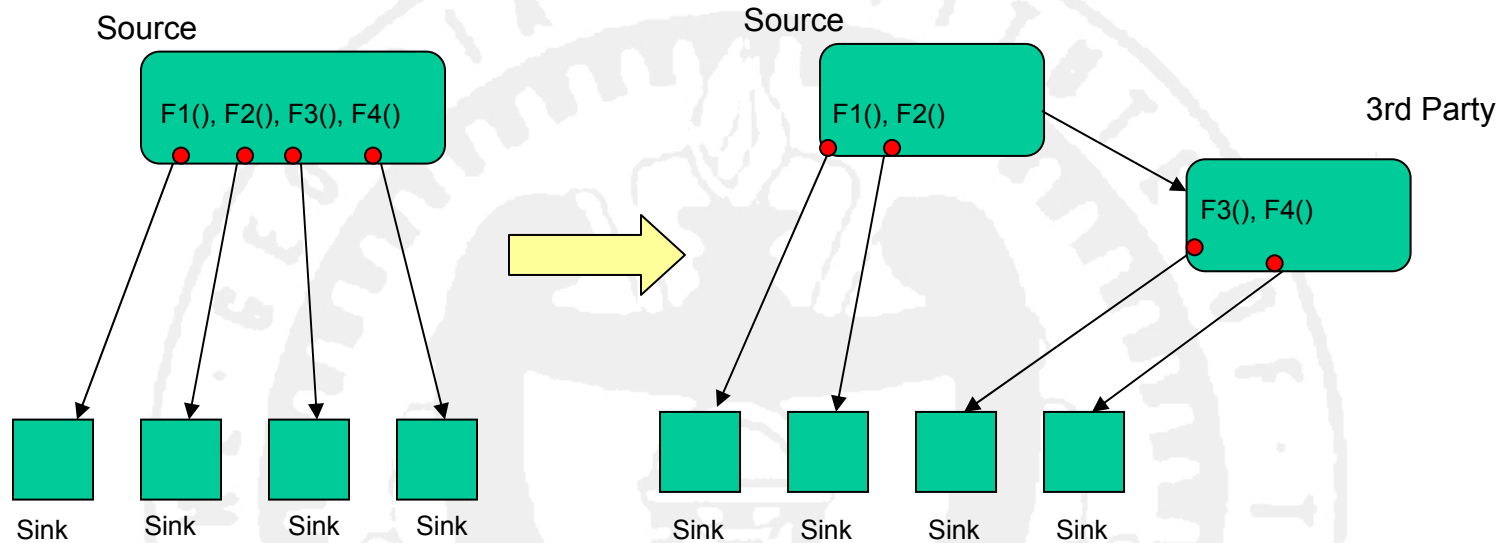
# Solution Approach I: Agile Information Flows

- **Services** composed via *agile* information flows, separated from application components
- **Agile flows** represented with **middleware**
  - Example: pub/sub (events, channels, handlers) with peer-to-peer communications (ECho)
  - Other examples: RMI or SOAP with quality handlers
- **Agility:**
  - dynamic handler deployment (DCG, ...)
  - code modification and adaptation
  - dynamic overlays

# I.a. Agile Flows: Source-side Filter Deployment: Remote Visualization



# I.b. Agility: Dynamic Overlays



## Runtime Overlay Creation and Management

### Assumptions:

- interoperable, typed events (binary -> XML)

### Implementation:

- DCG
- Query and custom (Fi()) operators



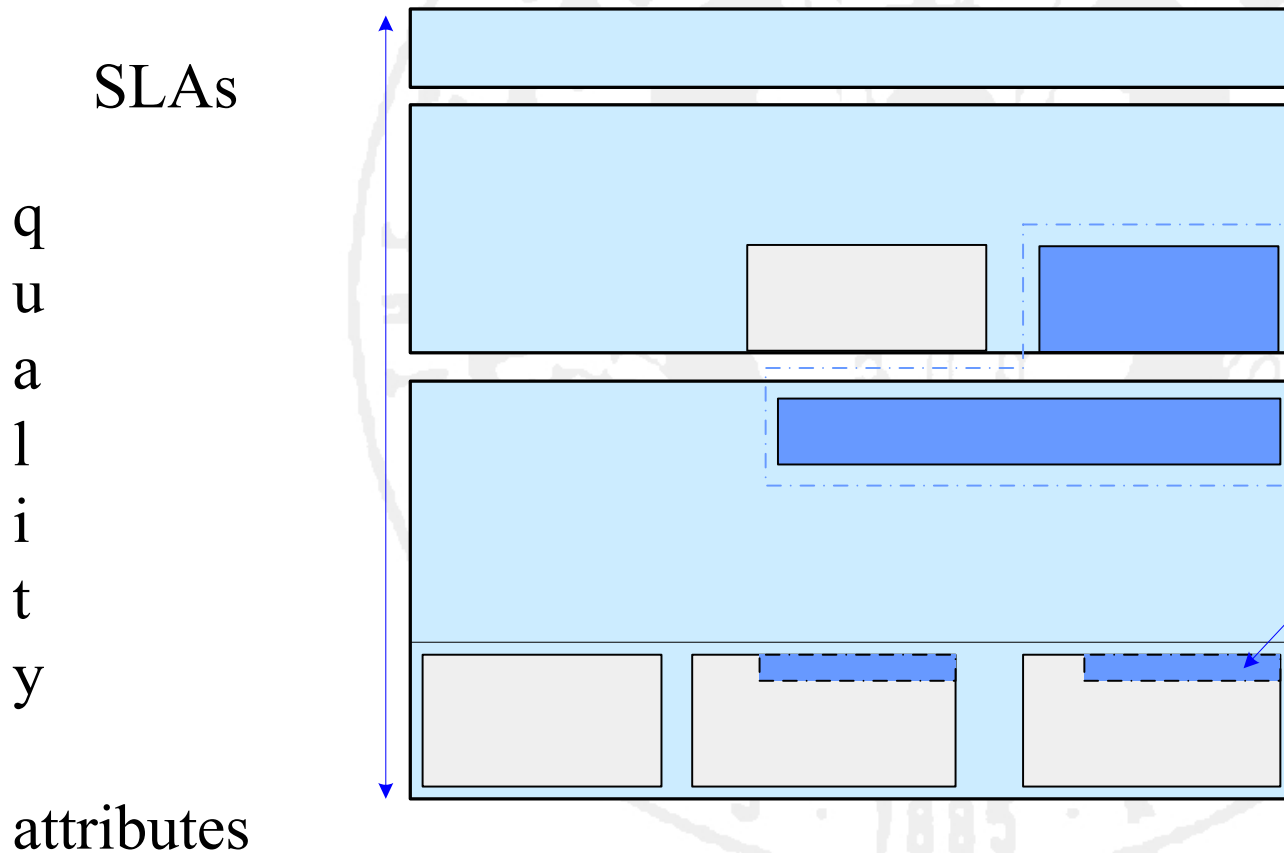
## II. Resource-Awareness in Middleware

- Services (via handlers) interact with end user applications **and** with underlying platforms
  - dynamic end user needs (SLAs, e.g., min. resolution)
  - dynamic platform resources (e.g., available bandwidth)
- Continuous monitoring and service/transport adaptation via **quality attributes**



# II. Resource-Aware Middleware: Software Architecture

Example: 'Netreact' Pub-Sub Middleware: Network Awareness





## II. Netreact Project Components

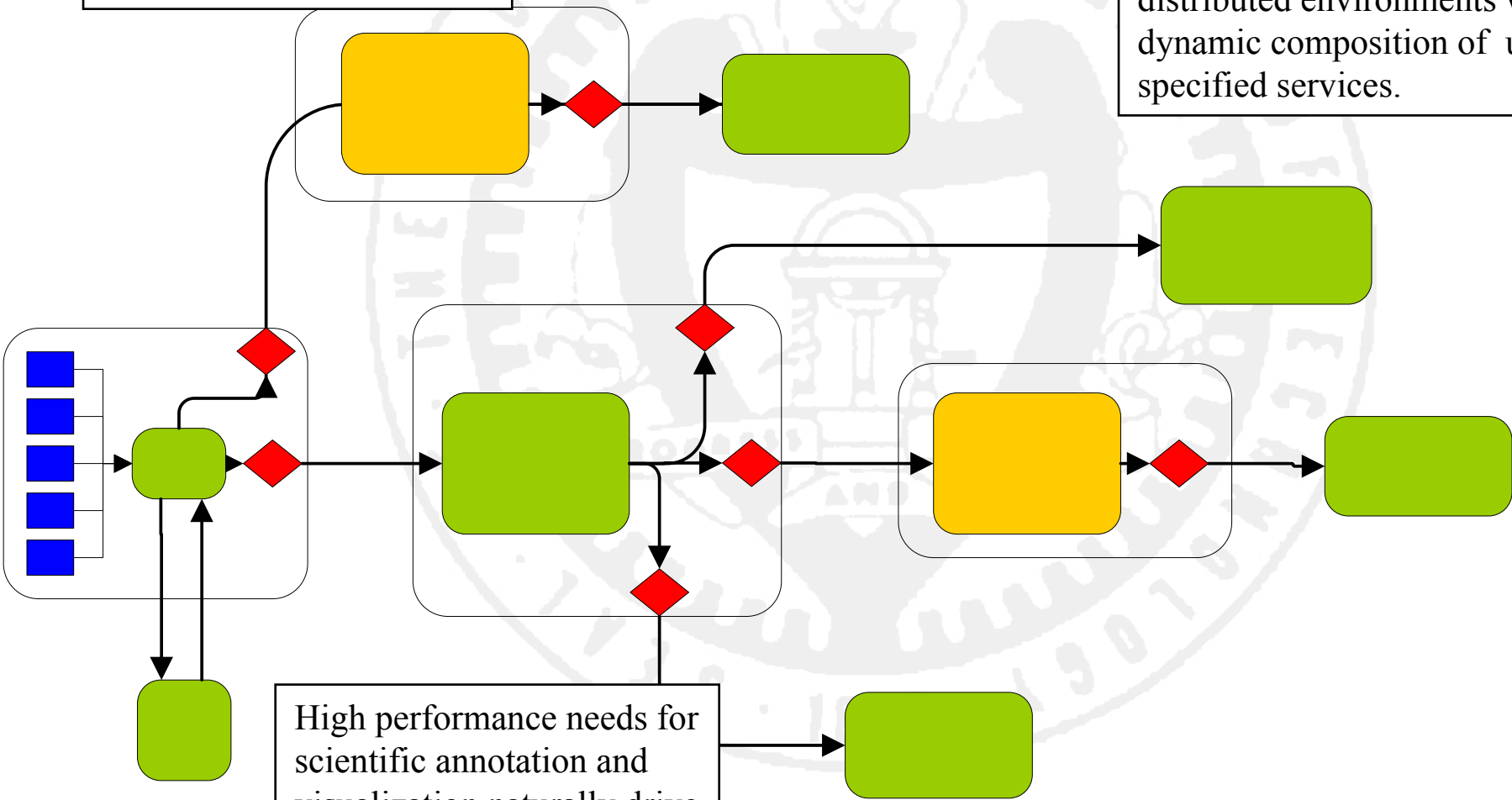
- Capturing end user needs (Pu, Wolf)
- Middleware architecture (Eisenhauer)
- Adaptable/tunable transports (Schwan, Dovrolis)
- Online resource monitoring:
  - Available network bandwidth (Dovrolis)
  - TCP bandwidth (Rao)
  - Distributed resource monitoring (dProc)
- **Evaluation** with real-time collaboration (Wolf, Rao, Wu)



# 'Smartpointer' Collaboration Infrastructure

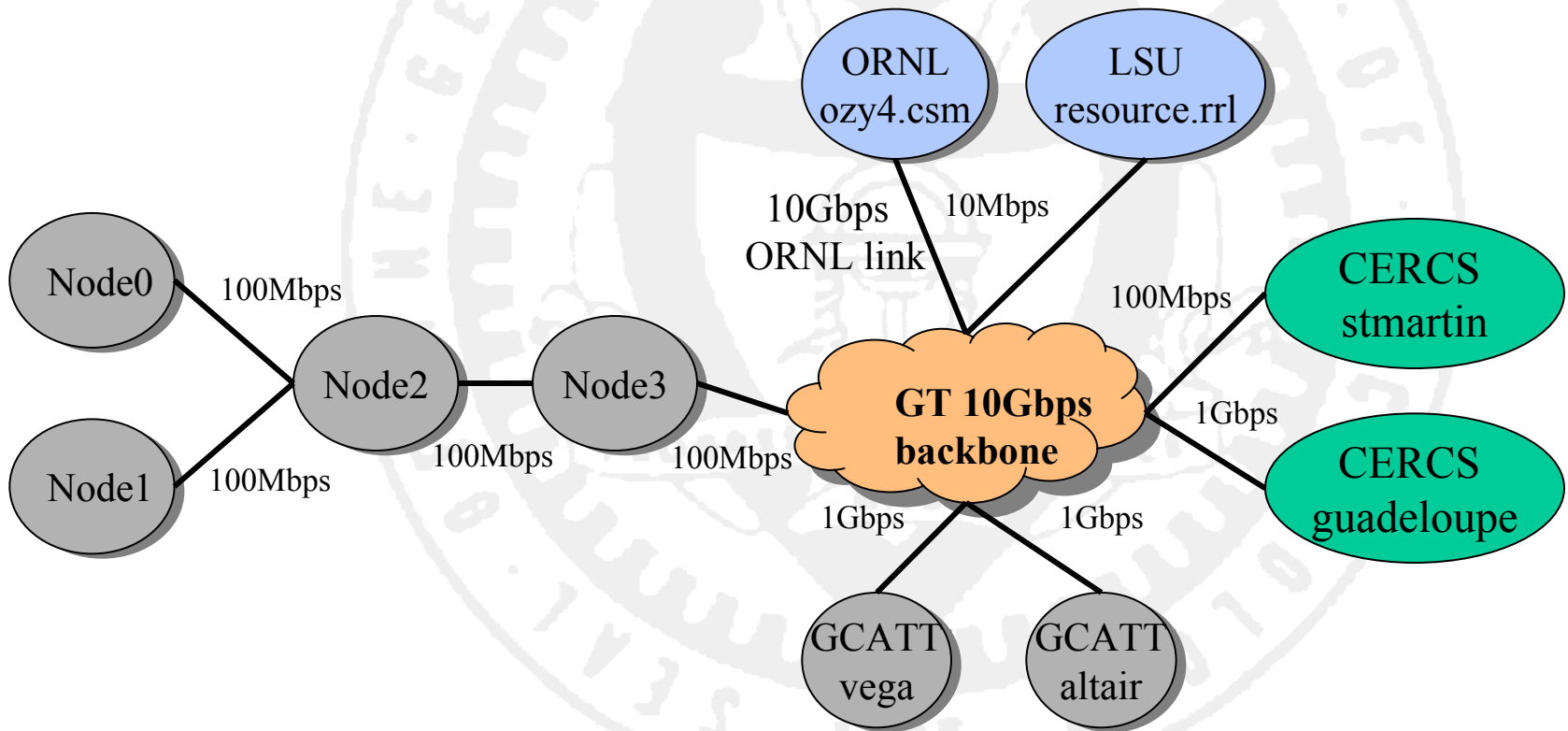
Utilizing resource-aware middleware to enable data-intensive scientific collaboration

Enable scientific collaboration in distributed environments with dynamic composition of user-specified services.



High performance needs for scientific annotation and visualization naturally drive towards Itanium2.

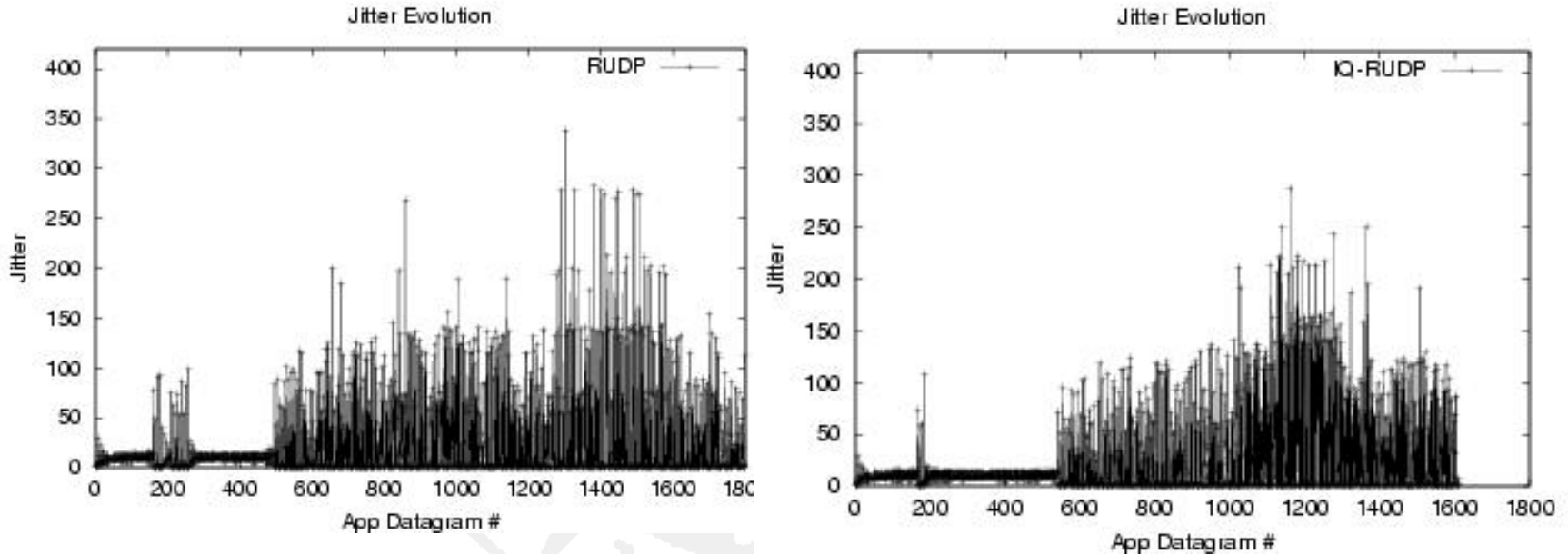
## II. Testbed





## II. Concept Demonstration: Coordinated Adaptation with Adaptive Packet Loss

traffic delay/jitter (on EmuLab)

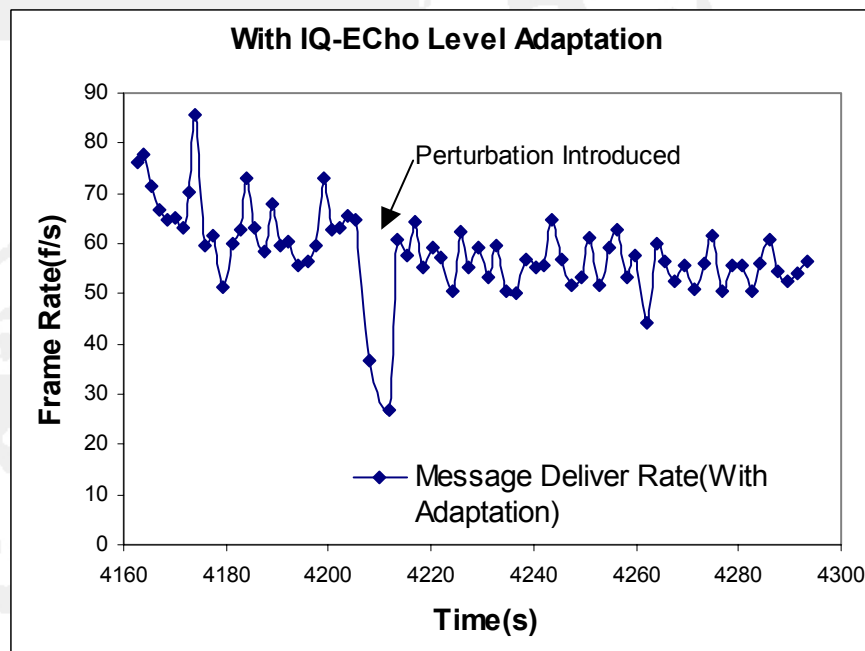
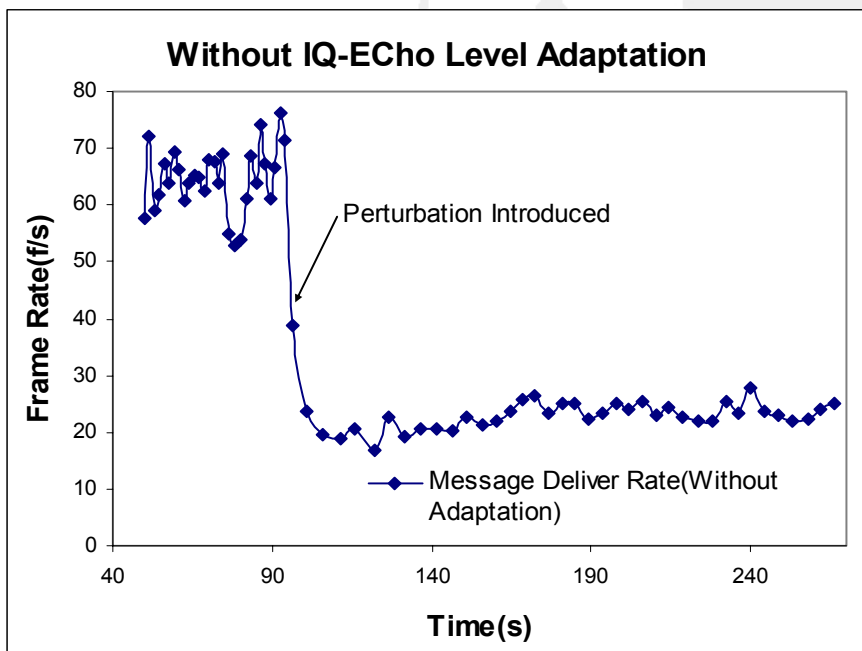


**without vs. with service/protocol coordination**

# II. Measurements:

## Client-aware Filtering Using Cutting Planes

- Adaptive Services (on right) achieve substantially higher frame rate





## II. Netreact: Ongoing/Next

- **New techniques for runtime service deployment, modification:**
  - using meta-data to automatically generate data filters
  - dynamic, resource-aware ‘overlays’
- **Multi-protocol, multi-stream management techniques:**
  - Integrated available bandwidth measurement
  - Standard protocols (e.g., TCP, TCP auto-tuning with Application-layer Delay Based Congestion Avoidance)
- **Use with grid/web middleware and tools:**
  - SOAP/RMI, Grid-FTP, m-by-n data exchanges
  - Exploit emerging grid ‘utility’ standards (e.g., autonomic-IBM, SmartFrog-HP)
  - High end testbeds (optical, 10GB link ORNL)
- **System support for continuous, distributed quality management:**
  - from dproc monitoring infrastructure and q-fabric to ‘Distributed Engines’



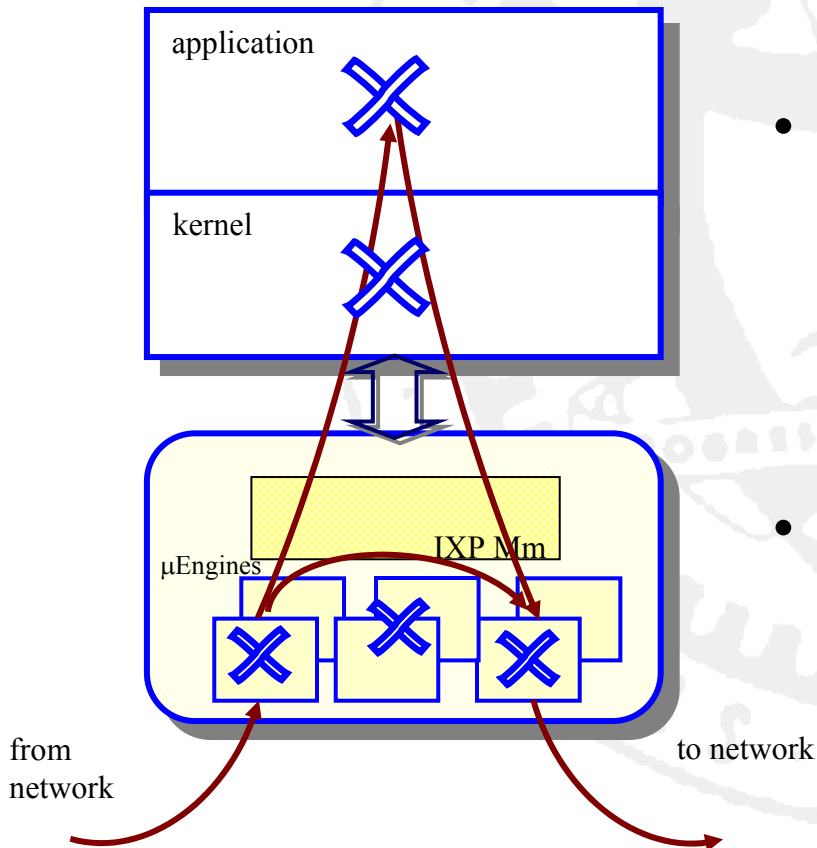
# III. From 'Black Box' OS to Open 'Distributed Engines'

**'Closed' considered harmful**

- 'Open' Platform = Programmable 'Engines'
  - Host-level (Kernel)
  - NP-level (programmable network processors)
- 'Engines' = 'Execution Containers':
  - strong properties (time and space isolation, control-capable inter-engine interfaces, ...)
  - dynamic extension (Multi-level: network, OS, middleware, applications, 'vertical code/data movement
  - integrated operation
- **Management: Continuous, Implicit Quality Management:**
  - implicit mechanisms, application-specific operation

# III. 'Open' Platform:

## From 'Kernel Plugins' to Programmable NPs

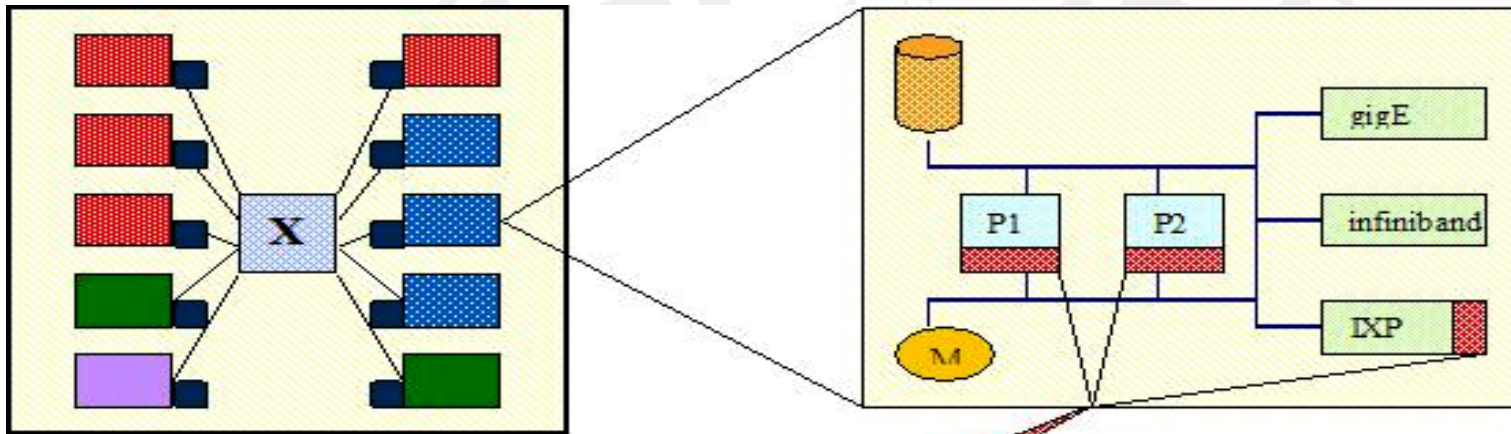


- 'Handling' Information Streams:
  - Host-side:
    - Kernel Plugins and KStreams
  - NP-side:
    - through dynamic stream handlers
- Programming Model?
  - 'Vertical' programming

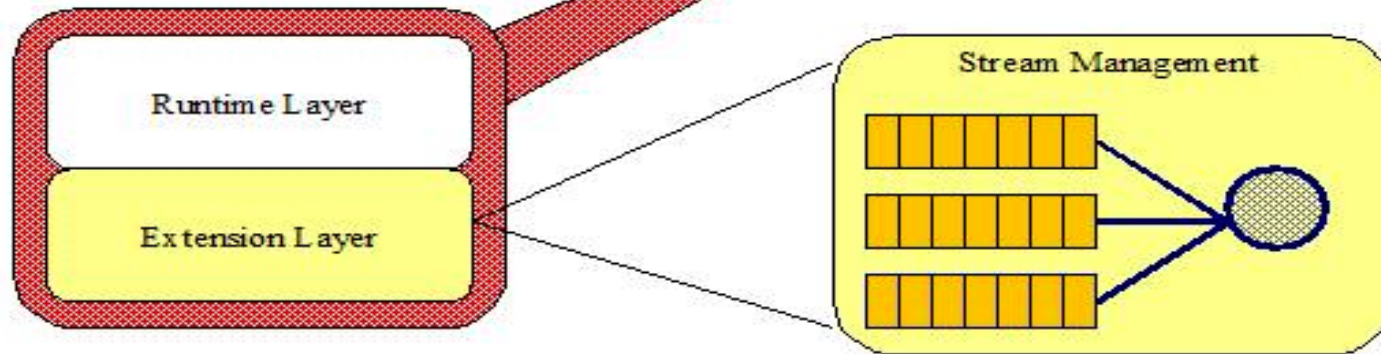
**Will Virtual Machine Technologies Help?**



# GT Nanomodeling Engine: 'Open', Itanium-based Machine?

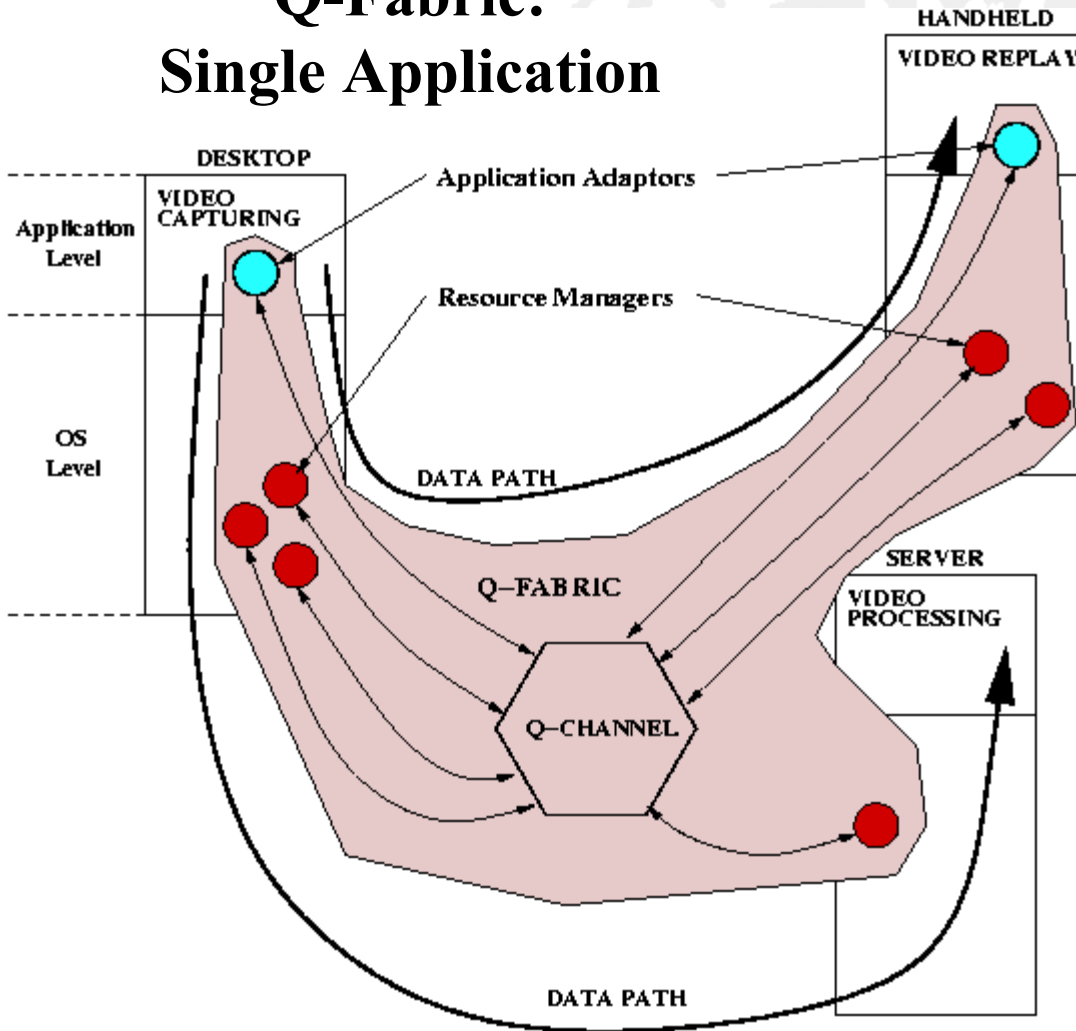


Nanodata Engine

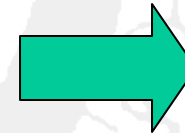


# III. Management: Q-Fabric and dProc

## Q-Fabric: Single Application

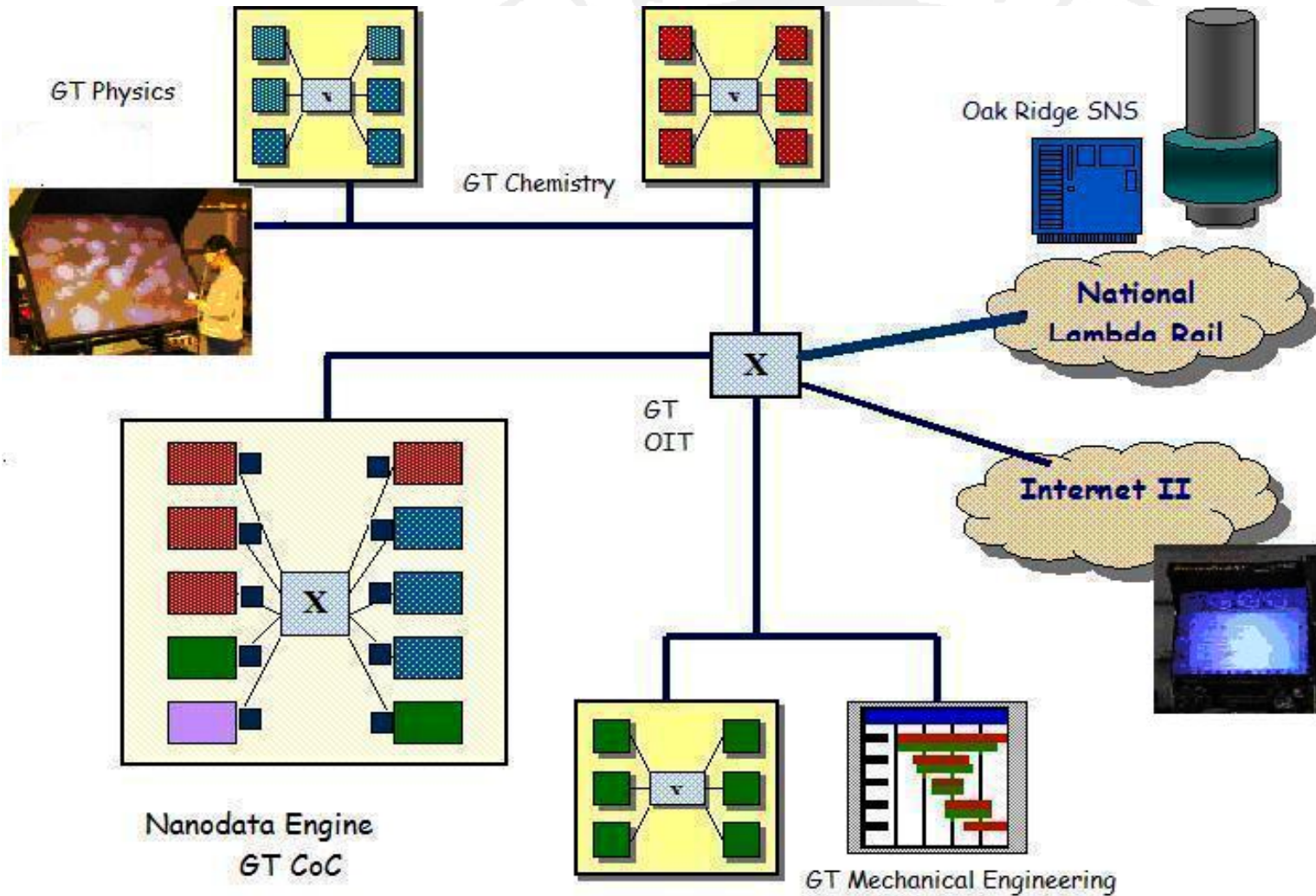


**Distributed Engines:  
Multi-application**



**Distributed Engines:  
Multi-application**

# Vision: GT Science Venue



Dynamically Managed Science Venues



# Conclusions and Future Work

- Problems addressed
  - Heterogeneous, high performance systems, applications, and end user needs
  - Dynamic behaviors
- Middleware
  - Ability to `negotiate' between applications and platforms
- Resource Management
  - Importance of cross-level interactions
    - Application/Middleware, Middleware/System/Network
  - Global vs. local views (e.g., `engine' paradigm)
  - Application-centric QoS scheduling
- `Open' Platforms
  - Programmable infrastructure, Virtual Machine Technology