Middleware for Embedded Adaptive Dependability (MEAD)

Real-Time Fault-Tolerant Middleware Support

Priya Narasimhan
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Collaborators
Thomas D Bracewell, Raytheon Integrated Defense Systems (co-PI, DARPA PCES-II)
Philip J Koopman, Carnegie Mellon University (co-PI, General Motors & NASA HDCP)

Electrical & Computer ENGINEERING

Background

- **Assistant Professor of ECE and CS at Carnegie Mellon University**
  - Research and teaching in the area of dependable distributed middleware
- **MEAD: Real-time fault-tolerant middleware**
  - Primary focus of my talk today
- **Starfish: Secure partition-tolerant scalable middleware**
- **Cyclopes: Robustness evaluation (and benchmarking) of middleware**

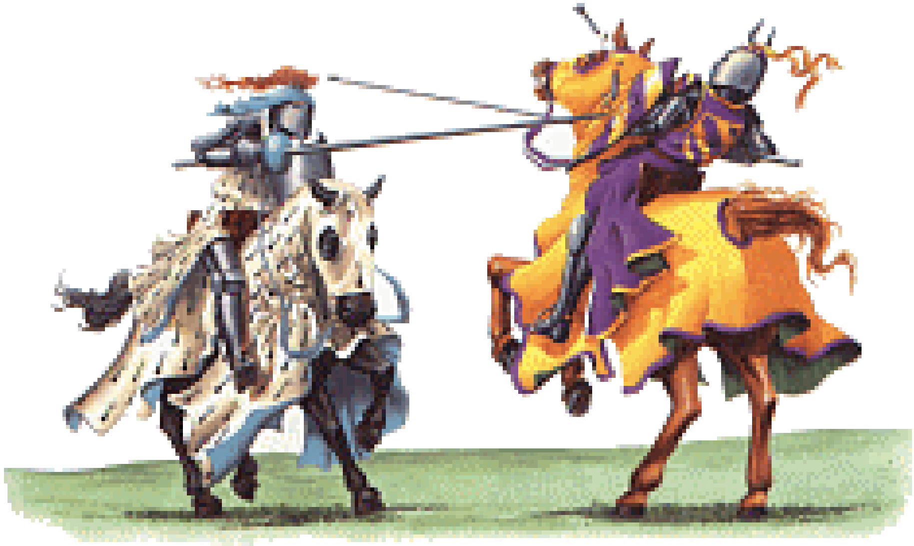
Motivation for MEAD

- **CORBA is increasingly used for applications, where dependability and quality of service are important**
  - The Real-Time CORBA (RT-CORBA) standard
  - The Fault-Tolerant CORBA (FT-CORBA) standard
- **But ……**
  - Neither of the two standards addresses its interaction with the other
  - Either real-time support or fault-tolerant support, but not both
  - Applications that need both RT and FT are left out in the cold
- **Focus of MEAD**
  - Why real-time and fault tolerance do not make a good "marriage"
  - Overcoming these issues to build support for CORBA applications that require **both** real-time **and** fault tolerance

Quality of Service for CORBA Applications

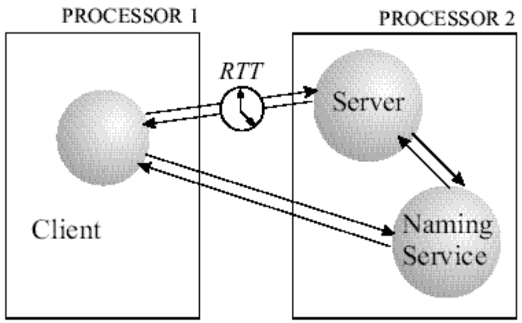
- **The Real-time CORBA (RT-CORBA) standard**
  - Scheduling of entities (threads)
  - Assignment of priorities of tasks
  - Management of process, storage and communication resources
  - End-to-end predictability
- **The Fault tolerant CORBA (FT-CORBA) standard**
  - Replication of entities (CORBA objects or processes)
  - Management and distribution of replicas
  - Logging of messages, checkpointing and recovery
  - Strong replica consistency

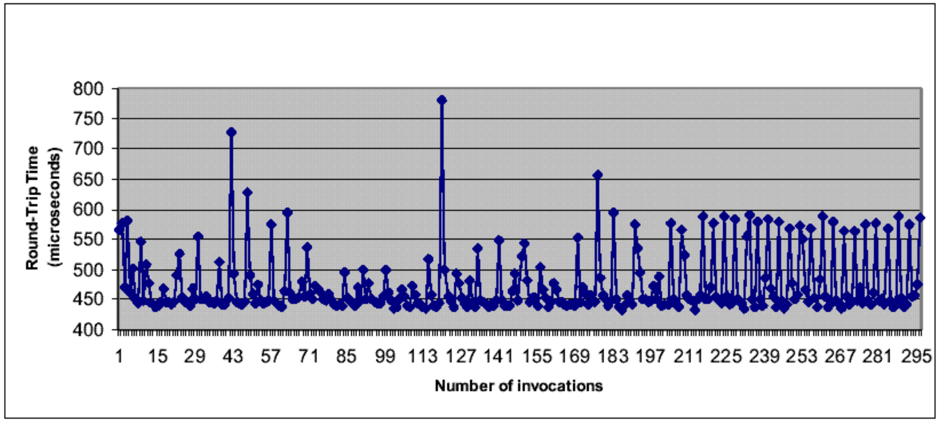
Real-Time Systems	Fault-Tolerant Systems
Requires *a priori* knowledge of events	No advance knowledge of when faults might occur
Operations ordered to meet task deadlines	Operations ordered to preserve data consistency (across replicas)
RT-Determinism ⇨ Bounded predictable temporal behavior	FT-Determinism ⇨ Coherent state across replicas for every input
Multithreading for concurrency and efficient task scheduling	FT-Determinism prohibits the use of multithreading
Use of timeouts and timer-based mechanisms	FT-Determinism prohibits the use of local processor time



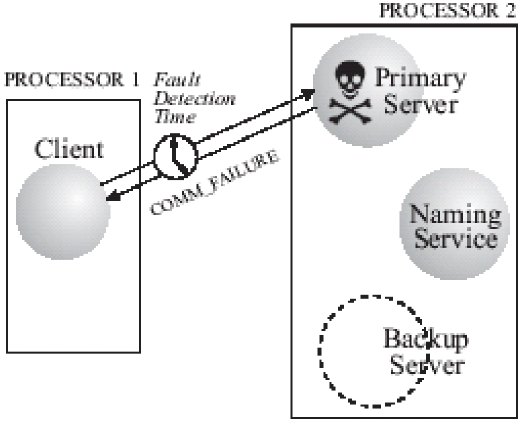
Observations – I

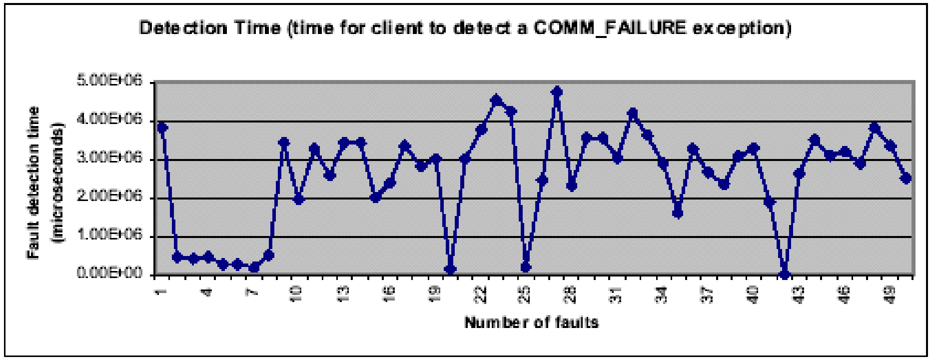
Fault-Free Performance for Simple Real-Time CORBA Applications





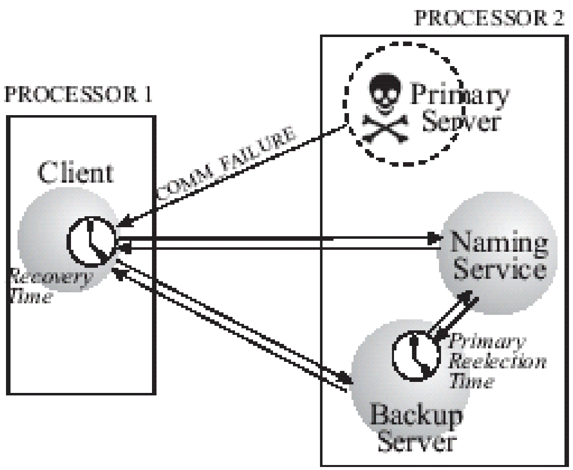
Fault-Detection Time for Simple Real-Time CORBA Applications

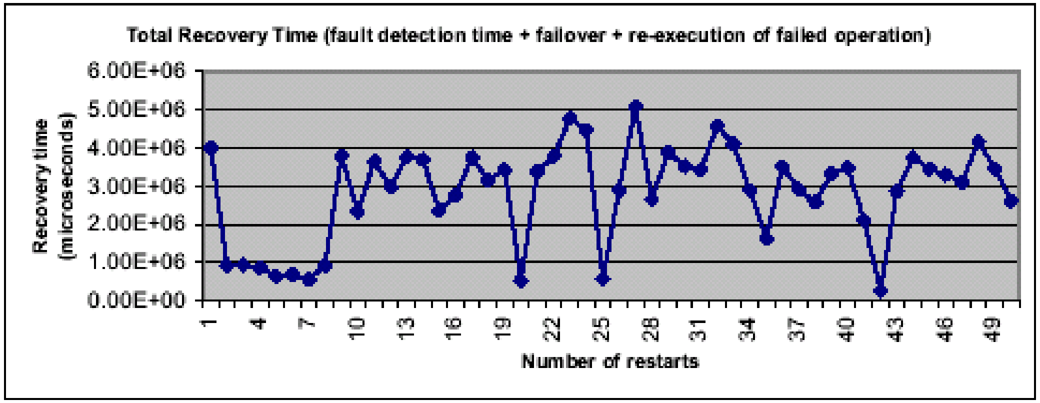




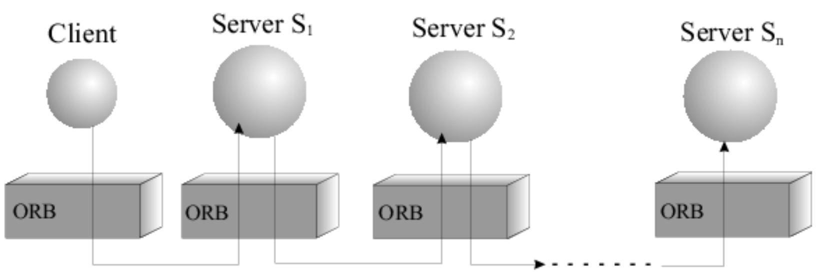
Observations – II

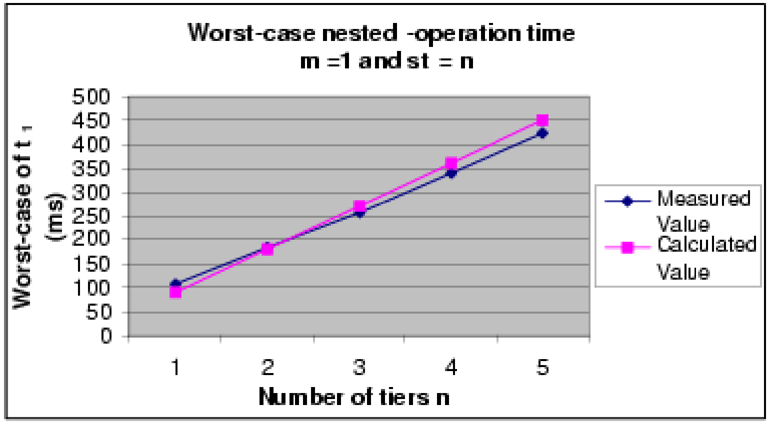
Recovery Time for Simple Real-Time CORBA Applications





Recovery Time for Multi-Tiered "Nested" Real-Time CORBA Applications





Combining Real-Time and Fault-Tolerance

- **Trade-offs between RT and FT for specific scenarios**
  - Effective ordering of operations to meet both RT and FT requirements
  - Resolution of non-deterministic conflicts (e.g., timers, multithreading)
- **Impact of fault-tolerance and real-time on each other**
  - Impact of faults/restarts on real-time behavior
  - Replication of scheduling/resource management components
  - Scheduling (and bounding) recovery to avoid missing deadlines
- **For large-scale systems**
  - Scalable fault detection and recovery
  - Considering nested (multi-tiered) middleware applications
  - Tolerance to partitioning faults

Architectural Overview

- **Use replication to protect**
  - Application objects
  - Scheduler and global resource manager
- **Special RT-FT scheduler**
  - Real-time resource-aware scheduling service
  - Fault-tolerant-aware to decide when to initiate recovery
- **Hierarchical resource management framework**
  - Local resource managers feed into a replicated global resource manager
  - Global resource manager coordinates with RT-FT scheduler
- **Ordering of operations**
  - Keeps replicas consistent in state despite faults, missed deadlines, recovery and non-determinism in the system

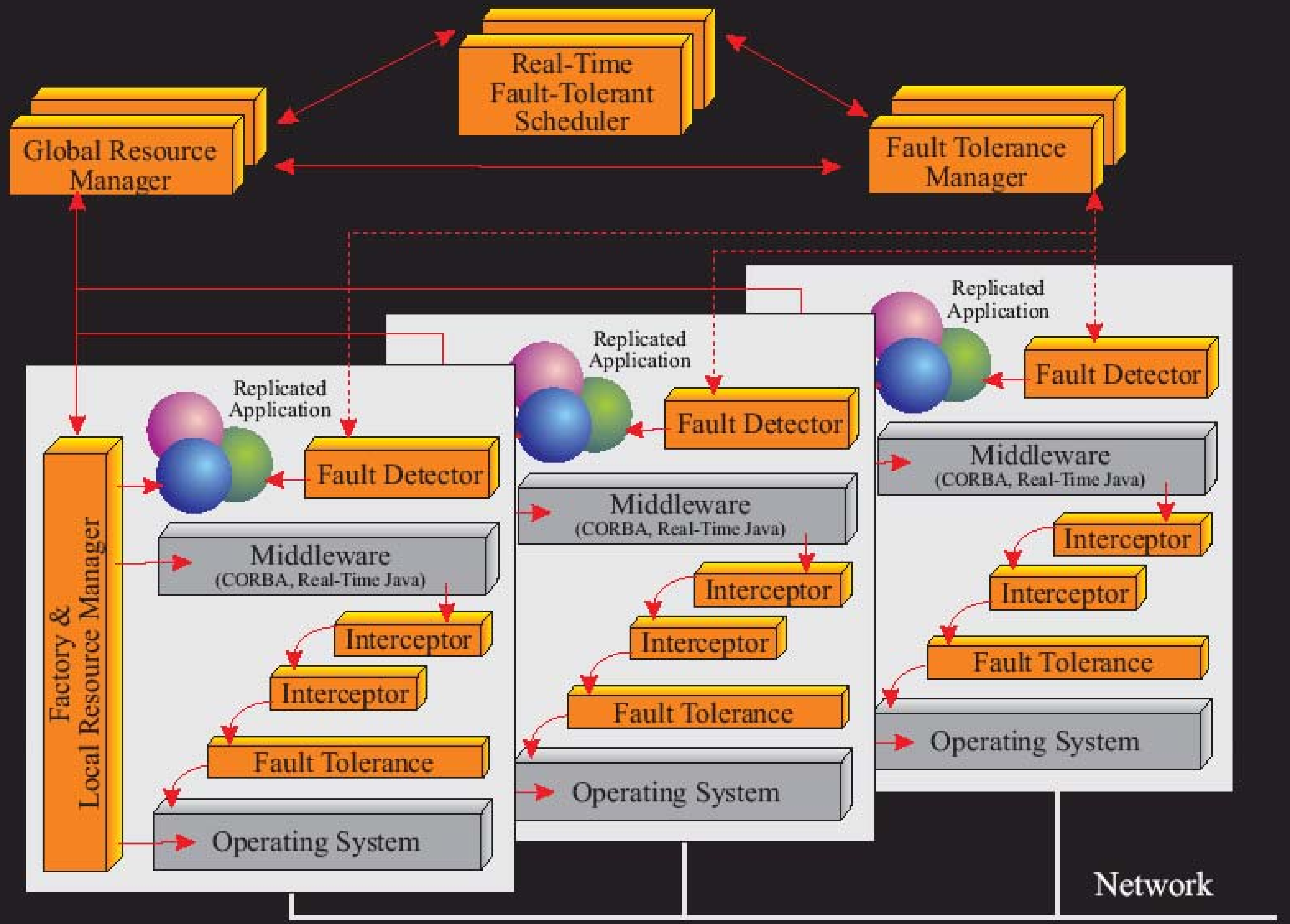
So, What Do We Want To Tolerate?

- **Crash faults**
  - ✓ Hardware and/or OS crashes in isolation
  - ✓ Process and/or Object crashes
- **Communication faults**
  - ✓ Message loss and message corruption
  - ✓ Network partitioning
- **Malicious faults (commission/Byzantine)**
  - ✗ Processor/process/object maliciously subverted
- **Omission faults**
  - ✓ Missed deadline in a real-time system
- **Design faults**
  - ✗ Correlated software/programming/design errors

Fault Model
Kinds of faults that MEAD is designed to tolerate

MEAD (Middleware for Embedded Adaptive Dependability)

- **Our RT-FT Architecture**
- **Why MEAD?**
- **Legendary ambrosia of the Vikings**
- **Believed to endow its imbibers with**
  - Immortality (⇨*dependability*)
  - Reproductive capabilities (⇨*replication*)
  - Wisdom for weaving poetry (⇨*cross-cutting aspects of real-time and fault tolerance*)
  - Happy and long married life (⇨*partition-tolerance*)


Real-Time Fault-Tolerant Scheduler
Global Resource Manager
Fault Tolerance Manager
Replicated Application
Fault Detector
Factory & Local Resource Manager
Middleware
(CORBA, Real-Time Java)
Interceptor
Interceptor
Fault Tolerance
Operating System
Network

Resource-Aware RT-FT Scheduling

- **Requires ability to predict and to control resource usage**
  - Example: Virtual memory is too unpredictable/unstable for real-time usage
  - RT-FT applications that use virtual memory need better support
- **Needs input from the local and global resource managers**
  - Resources of interest: load, memory, network bandwidth
  - Parameters: resource limits, current resource usage, usage history profile
- **Uses resource usage input for**
  - Proactive action
    - Predict and perform new resource allocations
    - Migrate resource-hogging objects to idle machines before they start executing
  - Reactive action
    - Respond to overload conditions and transients
    - Migrate replicas of offending objects to idle machines even as they are executing invocations

Proactive Dependability

- **What if we knew, with some confidence, when a fault was to occur?**
- **Needs input from a fault-predictor (error-log analysis)**
  - To determine when, and what kinds of, faults can occur
  - To schedule fault detection time based on prediction
- **Needs input from a recovery-predictor**
  - **Offline predictor**: Source code analysis for worst-case recovery time
    - Look at each object’s data structures
    - Looks at the object’s containing process and ORB interactions
    - Not comprehensive: unable to predict dynamic memory allocations
  - **Runtime predictor**: Object execution and memory allocation profile
    - Intercepts and observes runtime memory allocations (e.g., object instantiation, library loading), connection establishment, etc.
    - Prepares for the worst-case replica recovery time

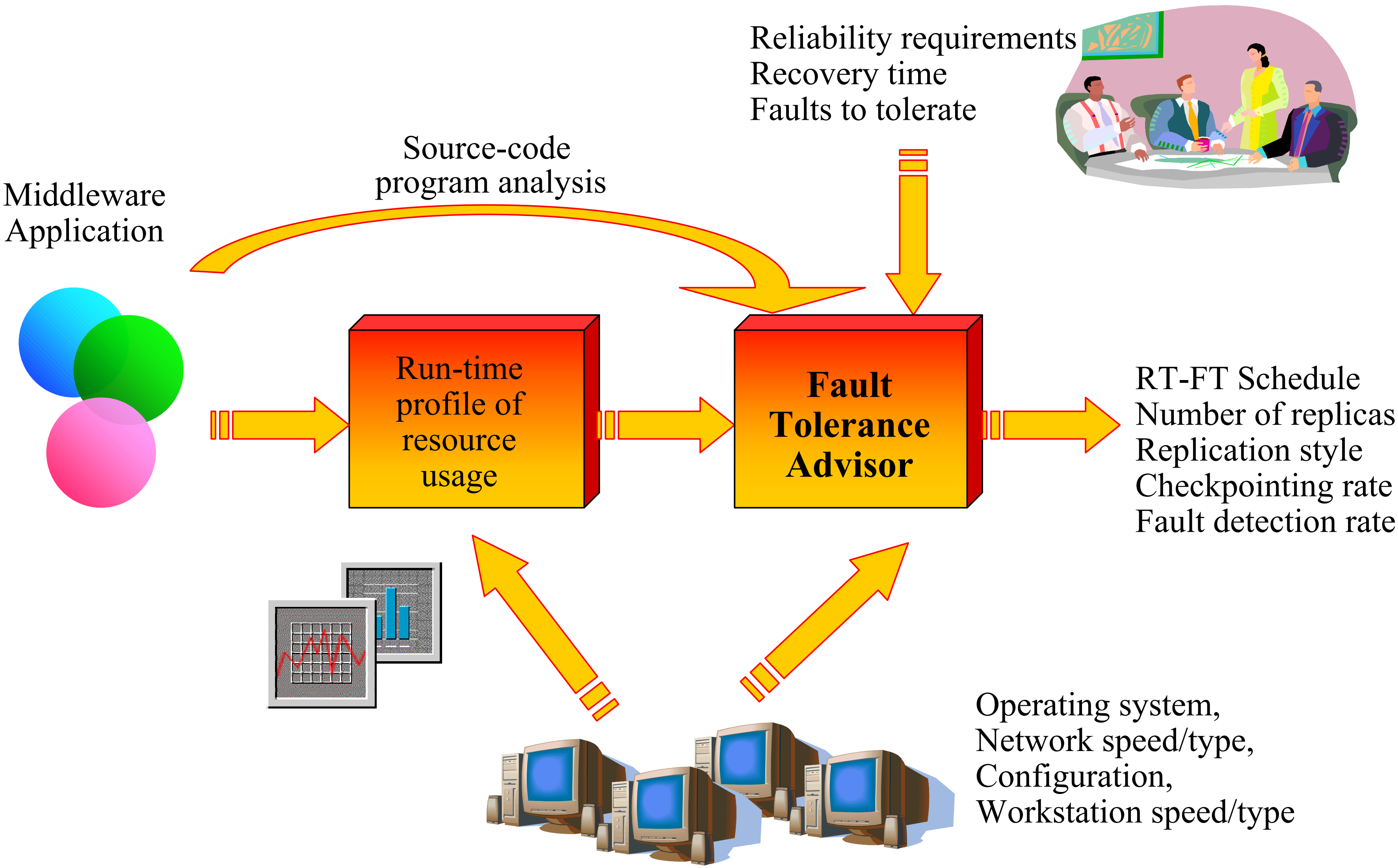
Offline Program Analysis

- **Application may contain RT vs. FT conflicts**
- **Application may be non-deterministic**
- **MEAD sifts interactively through application source-code**
  - To pinpoint sources of conflict between real-time and fault-tolerance
  - To determine size of state, and to estimate recovery time
  - To determine the appropriate points in the application for the incremental checkpointing of the application
  - To highlight, and to compensate for, sources of non-determinism
    - Multi-threading
    - Direct access to I/O devices
    - Local timers
- **Output of program analysis (recovery-time estimates) fed to the Fault-Tolerance Advisor**

Fault-Tolerance Advisor

- **Configuring fault tolerance today is mostly ad-hoc**
- **To eliminate the guesswork, we deployment/run-time advice on**
  - Number of replicas
  - Checkpointing frequency
  - Fault-detection frequency, etc.
- **Input to the Fault-Tolerance Advisor**
  - Application characteristics (using output from program analysis)
  - System reliability characteristics
  - System's and application's resource usage
- **Fault-Tolerance Advisor works with other MEAD components to**
  - Enforce the reliability advice
  - Sustain the reliability of the system, in the presence of faults

Fault-Tolerance Advisor



Summary

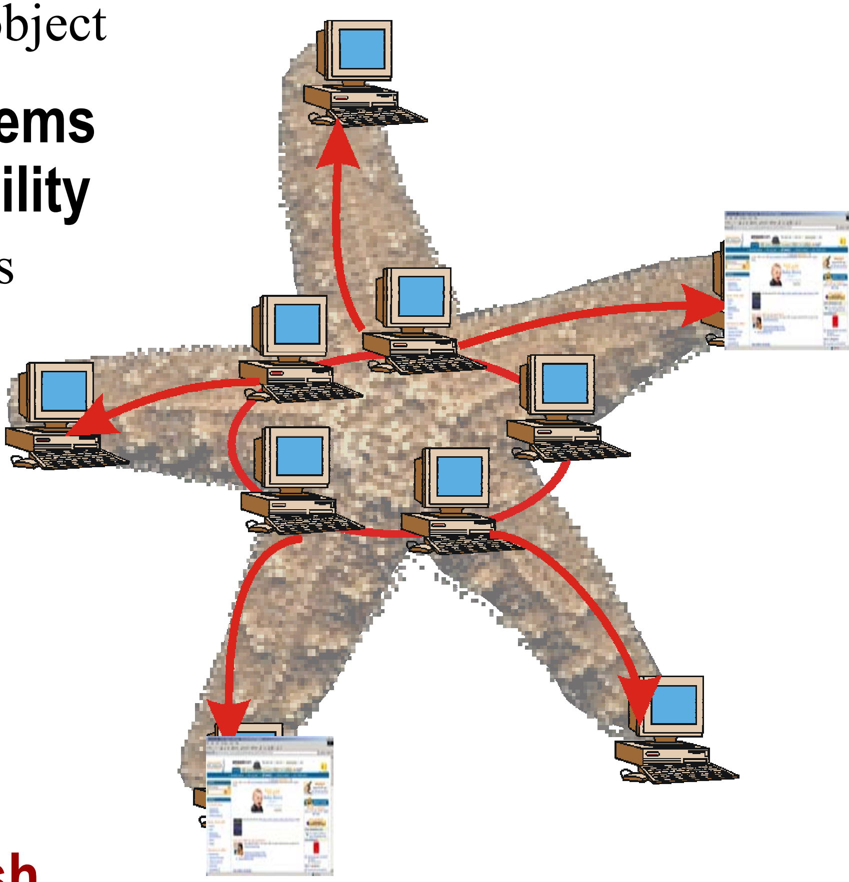
- **Resolving trade-offs between real-time and fault tolerance**
  - Ordering of tasks to meet replica consistency and task deadlines
  - Bounding fault detection and recovery times in asynchronous environment
  - Estimating worst-case performance in fault-free, faulty and recovery cases
- **MEAD's RT-FT middleware support**
  - Tolerance to crash, communication, timing and partitioning faults
  - Resource-aware RT-FT scheduler to schedule recovery actions
  - Proactive dependability framework
  - Fault-tolerance advisor to take the guesswork out of configuring reliability
  - Offline program analysis to detect, and to compensate for, RT-FT conflicts
- **Ongoing research and development with RT-CORBA and Real-Time Java**
- **Intention to participate in the standardization efforts of the OMG**
- **Sponsors: DARPA PCES-II, General Motors, National Science Foundation**

Looking Ahead to RT-FT Standardization

- **Consider (and seek means to reconcile) the fundamental conflicts/tensions between real-time and fault-tolerance**
  - To apply the solution to a wider class of middleware applications
  - To avoid point solutions that might work well, but only for well-understood applications, and only under certain constraints
  - To allow for systems that are subject to dynamic conditions, e.g., changing constraints, new environments, overloads, faults, ……
- **Expose interfaces that support the**
  - Capture of the application's fault-tolerance and real-time needs
  - Tuning of the application's fault-tolerance and real-time configurations
  - Query of the provided "level" of fault-tolerance and real-time
  - Scheduling of both real-time and fault-tolerance (fault-detection, fault-recovery and fault-forecasting) activities

Related Projects: Starfish

- ***System-wide* Intrusion Tolerance**
  - Looks at which parts of the system may have been tainted by faulty processor/object
- **Supports *multi-tiered* wide-area systems with varying guarantees for survivability**
  - Extends the survivability to both clients and servers
  - Proactive containment of malice
- **More comprehensive fault model**
  - Crash faults
  - Communication faults
  - Byzantine/arbitrary faults
  - System/Network partitioning



http://www.ece.cmu.edu/~starfish

Related Projects: Cyclopes

- **Part of the NASA High Dependability Computing Program (HDCP) – joint work with Prof. Philip J. Koopman of Carnegie Mellon University**
- **How do you know if a dependable system is really dependable?**
  - Cyclopes – ensuring robust middleware systems
  - Probing middleware interfaces to see how they respond to anomalies
  - Wrappers to contain detected system vulnerabilities
- **Quantifying dependability**
  - How do you put a number on dependability?
  - Metrics and benchmarks for objective evaluation
  - Need to evaluate "-ilities" in isolation and in composition
- **Evaluation of Java middleware**
  - Generic Baseline (Red Hat Linux/ SUN VM): 4.7 % Robustness Failure Rate
  - Timesys Real-Time Java: Similar rate, but less robust
    - Segmentation faults encountered

For More Information on MEAD

http://www.ece.cmu.edu/~mead

Priya Narasimhan
Assistant Professor of ECE and CS
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Tel: +1-412-268-8801
priya@cs.cmu.edu

Raytheon
Integrated Defense Systems