MEAD
Middleware for Embedded Adaptive Dependability

Priya Narasimhan
Carnegie Mellon University
Pittsburgh, PA
priya@cs.cmu.edu

My Background

- Prior research on dependable enterprise systems
  - Developed systems that provide "out-of-the-box" reliability to CORBA/Java applications
    - No need to change application or ORB code
  - Eternal: Fault-tolerant CORBA/Java support
  - Immune: Secure CORBA/Java support
- Helped to establish Fault-Tolerant CORBA standard and founded company to sell fault-tolerant products based on my PhD research
- Lessons learned [IEEE TOCS 2004]
  - It's hard for users to (re)configure the fault-tolerance of their systems to suit the applications' needs
  - There needs to be a way of mapping high-level user requirements to low-level implementation mechanisms

Motivation for MEAD

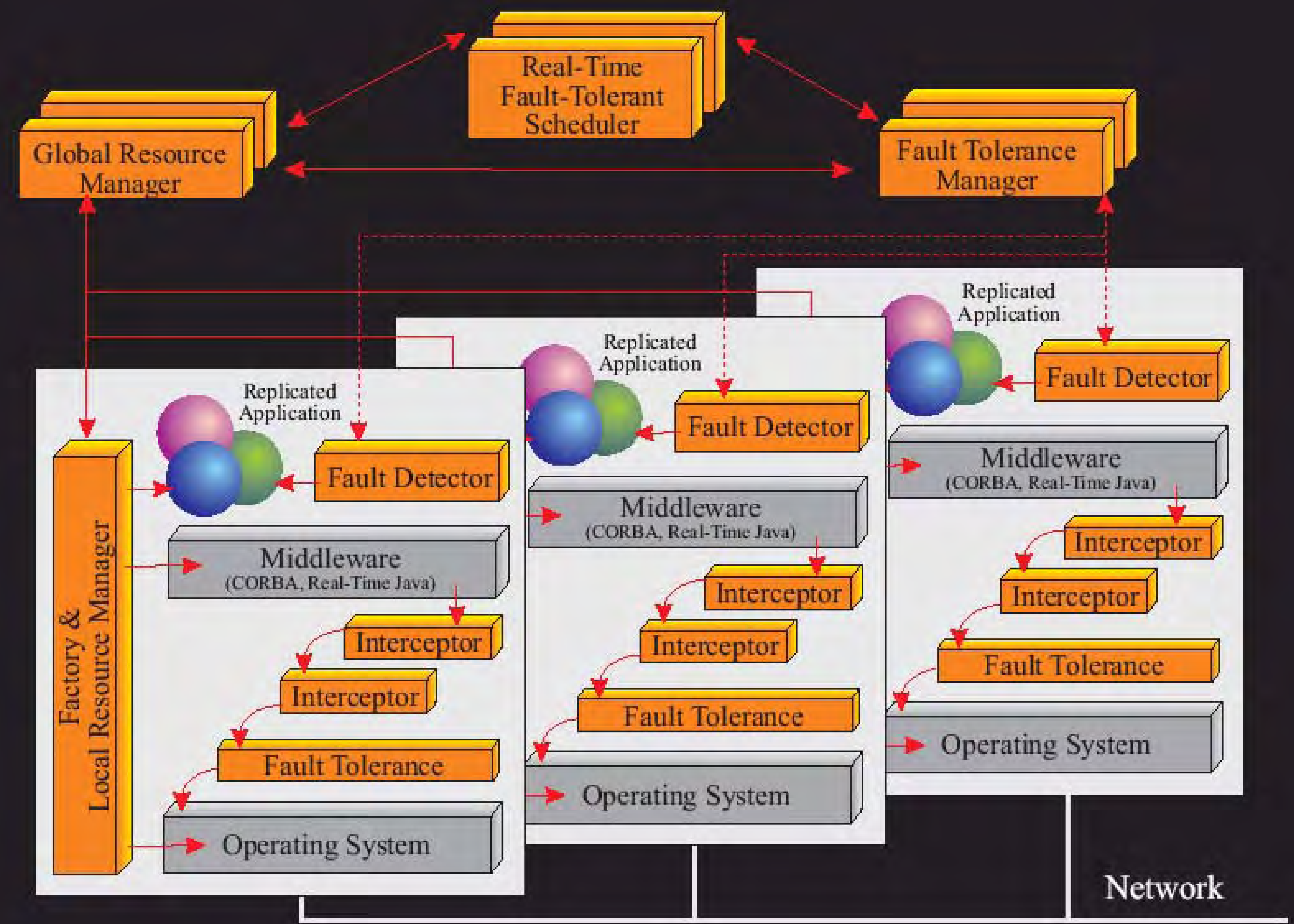
- Middleware is increasingly used for applications, where dependability and quality of service are important
  - Fault-Tolerant CORBA and Fault-Tolerant Java standards
- But ……
  - These standards provide a laundry list of “fault-tolerance properties”
  - No insight into how these properties ought to be set
  - No insight into how fault-tolerance and fault-recovery can be configured to meet an application’s performance or reliability requirements
- One focus of MEAD
  - Providing advice on configuring fault-tolerance for distributed applications
  - Being able to determine this configuration at deployment-time
  - Being able to re-determine and enforce configurations at runtime
  - Being able to perform (re)configuration proactively, where possible
  - Middleware merely a vehicle for exploring proactively configurable fault-tolerance

Research Focus

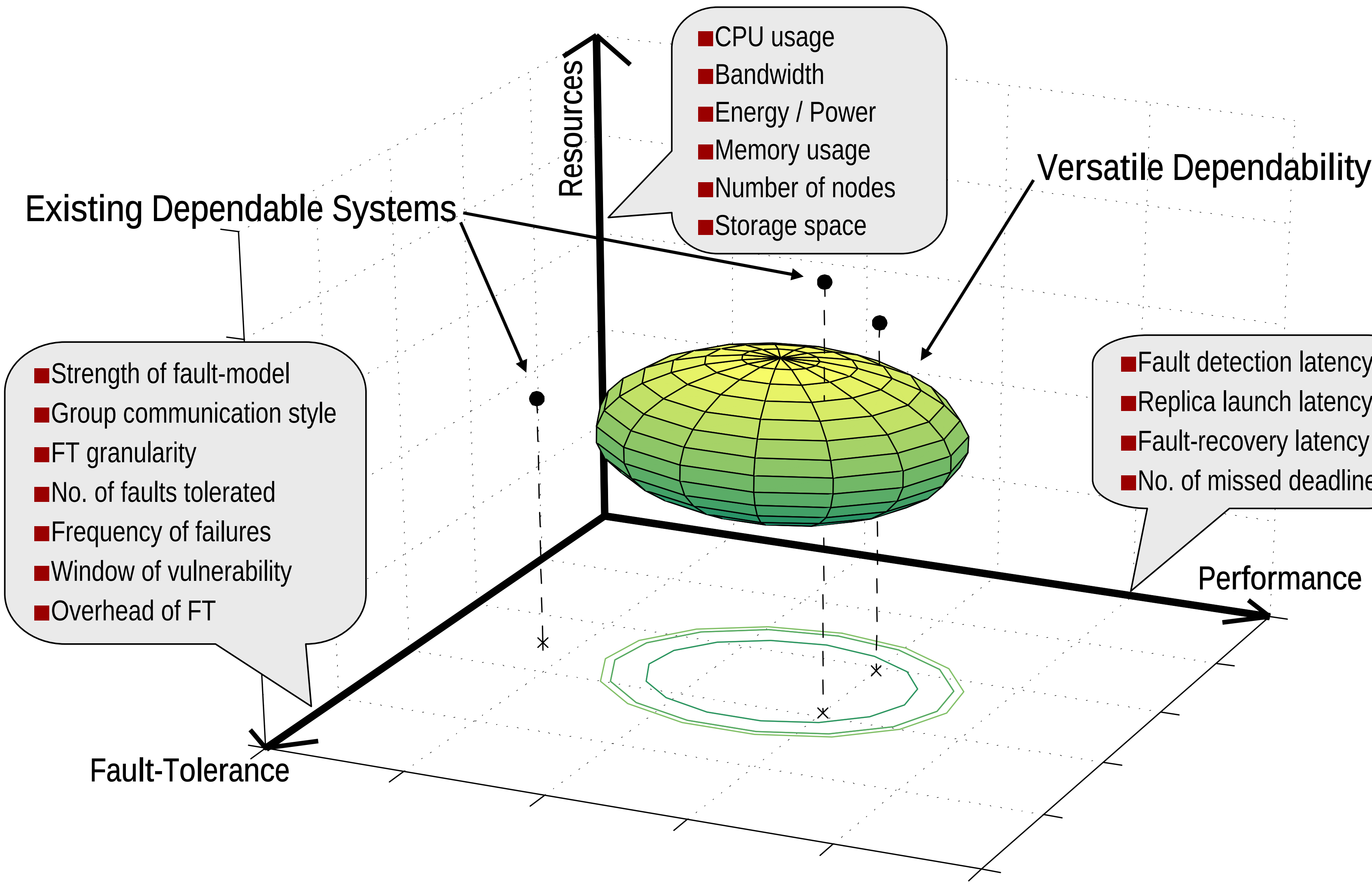
- Overall objectives of the MEAD system
  - Automated, adaptive (re)configuration of fault-tolerance [WADS 2004]
  - Proactive fault-recovery for distributed applications [DSN 2004]
    - Exploiting system information for faster recovery
  - Static analysis of application and middleware code to extract application-level insights and communicate them to the MEAD runtime [SRDS 2004]
  - Zero-downtime, live upgrades of the application
    - Dependency tracking at runtime and development-time
    - Staggered quiescence of different parts of the system
- Target applications
  - Embedded printing applications (HP Labs)
  - Unmanned aerial vehicles (BBN & Boeing)
  - Shipboard computing platforms (Raytheon & Lockheed Martin)
  - Automotive telematics systems (General Motors)

And Now For Something Completely Different ....

- 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 performance and fault tolerance*)


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

Versatile Dependability



"Knobs" of the MEAD System

- Application
  - Frequency of requests
  - Size of requests/responses
  - Size of State
  - Application Resources

High Level Knobs

- External Properties
  - Scalability
  - Availability
  - Real-Time Guarantees

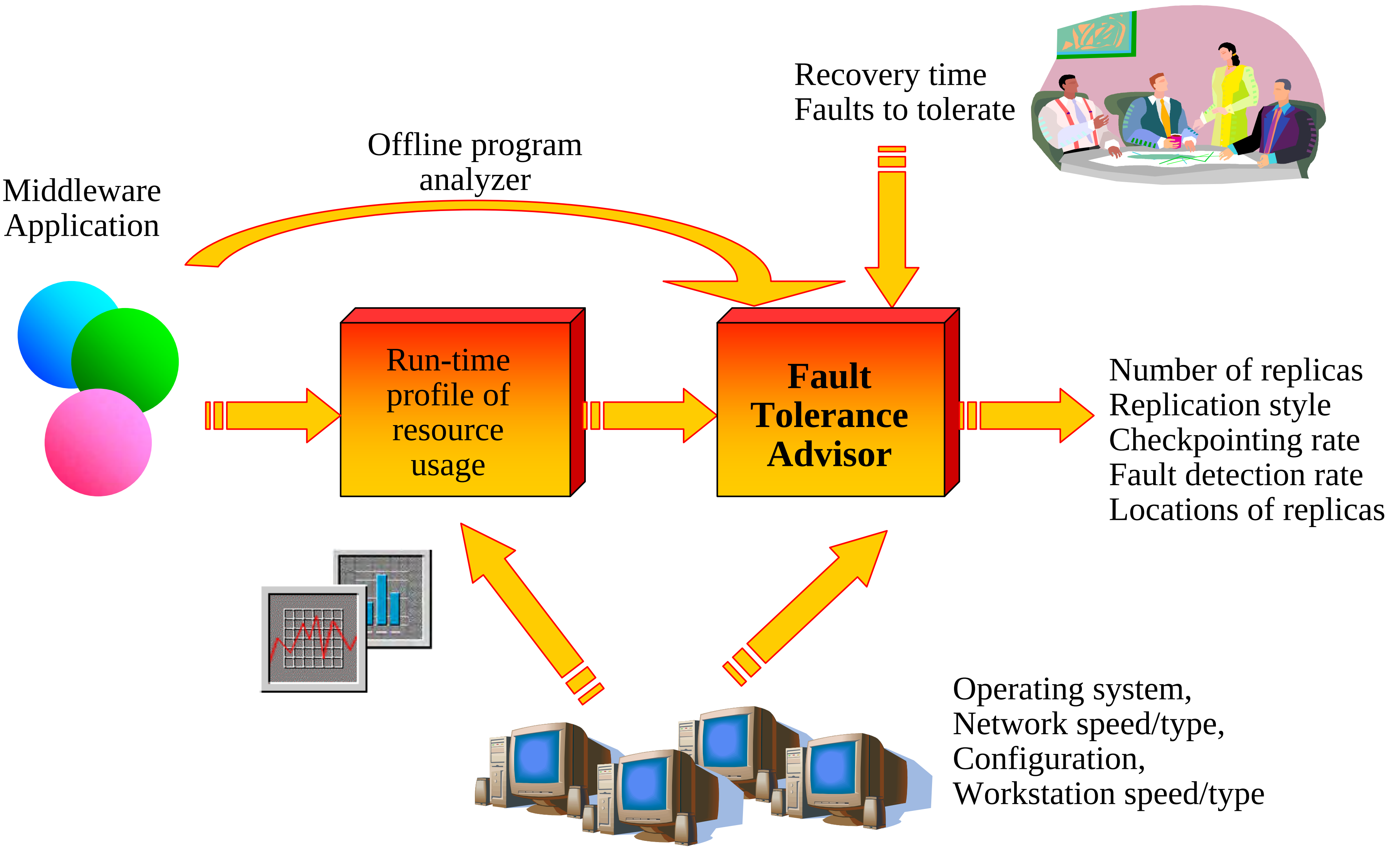
Low Level Knobs

- Fault-tolerance infrastructure (MEAD)
  - Replication (Active, Passive)
  - Number of Replicas
  - Checkpointing Frequency

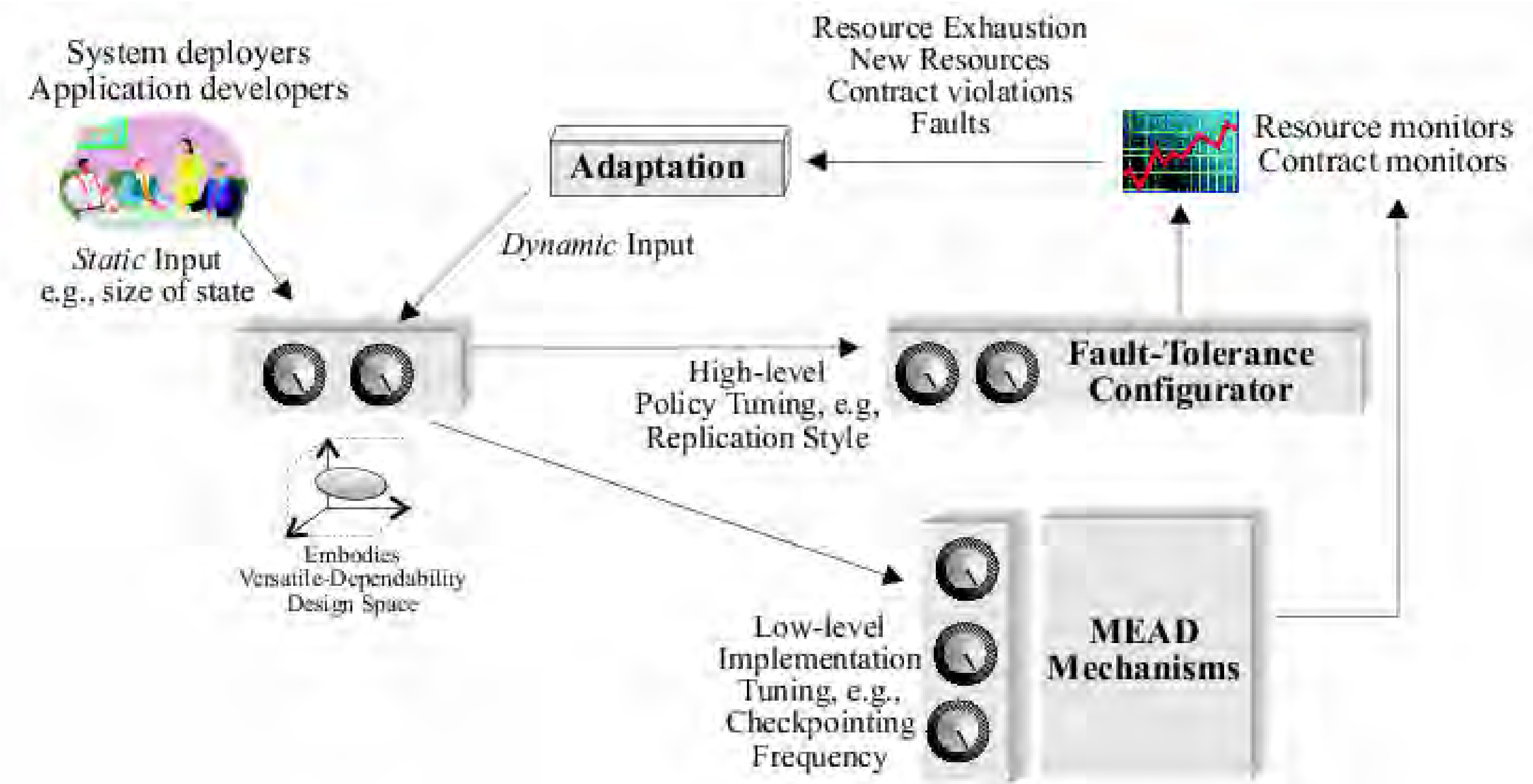
Fault-Tolerance Advisor

- 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 (through 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



Run-Time Adaptation



Mode-Driven Fault-Tolerance Adaptation

- Most applications have multiple modes of operation
  - Example: the unmanned aerial vehicle (UAV) application exhibits
    - Surveillance mode
    - Target recognition mode
- Each mode might require different fault-tolerance mechanisms
  - The critical elements in the path might differ
  - The resource usage might differ, e.g., more bandwidth used in some modes
  - The notion of distributed system "state" might be different
- MEAD aims to provide the "right mode-specific fault-tolerance"
  - Based on the Fault-Tolerance Advisor's inputs
  - In response to (omens heralding) mode changes

Proactive Fault-Tolerance

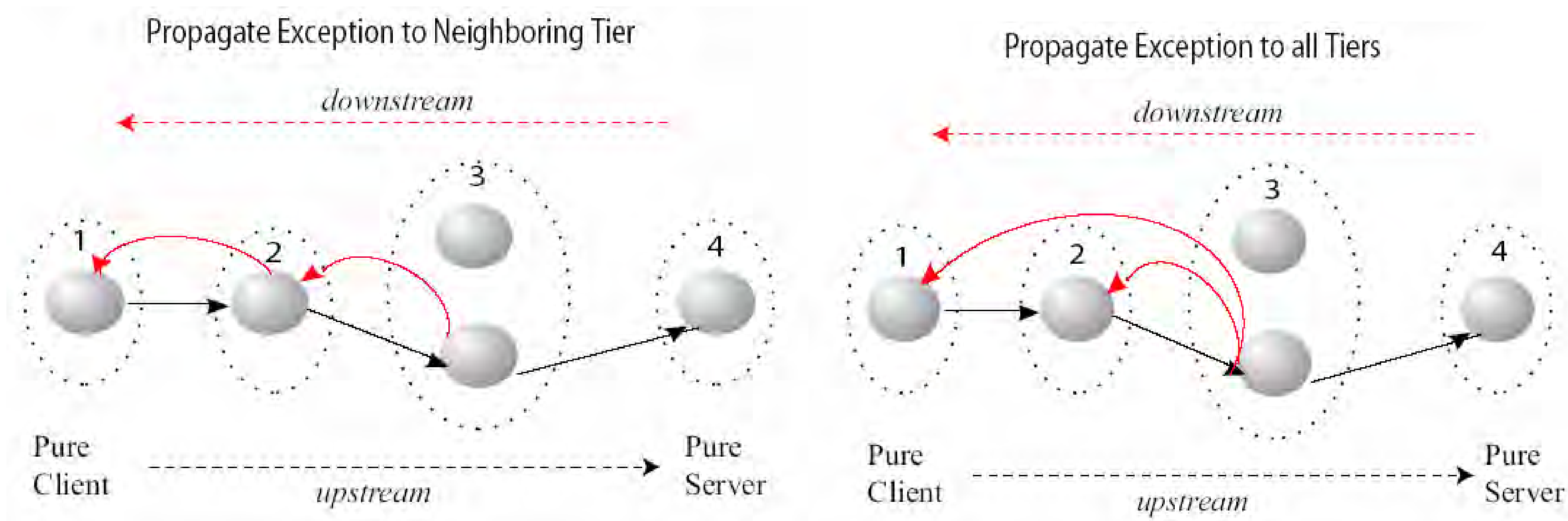
- Involves predicting, with some confidence, when a failure might occur, and compensating for the failure even before it occurs
  - For instance, if we knew that a processor had an 80% chance of failing within the next 5 minutes, we could perform process-migration
- Our goal in MEAD is to
  - Lower the impact faults have on real time schedules
  - Implement proactive dependability in a transparent manner
- Proactive dependability has two aspects:
  - Fault prediction: Reducing the unpredictable nature of faults
  - Proactive recovery: Reducing fail-over times and number of failures experienced at the application-level (primary focus in MEAD)
- <u>Complements, but does not replace,</u> the classical reactive fault-tolerance schemes since we cannot predict every fault

Benefits

- Provides a framework for proactive recovery that is transparent to the client application
- Proactive recovery can
  - Significantly reduce failover times, lowering the impact of a failure on real-time schedules
  - Reduce the number of failures experienced at the application level
  - Exploit knowledge of system topology to provide advance warning of failures to other servers "further down the line" (multi-tiered applications)
  - Request the recovery manager to launch new replicas so that a consistent number of replicas are retained in the group (useful for active replication where a certain number of servers are required to reach agreement)
- Caveat
  - Not applicable to every kind of fault, of course

Ongoing: Topology-Awareness

- Curbing the spread of propagating faults or invoking faster recovery based on
  - System topology,
  - Application’s interconnections,
  - Application’s normal fault-free behavior
- Could also help sequence recovery actions across nodes



Ongoing: Live Software Upgrades

- Live software upgrades
  - Software upgrades currently involve downtime (“scheduled maintenance”)
  - Also, can cause a cascade of upgrades rippling through the system
- Development-time preparation for live upgrades
  - Exploiting program analysis
  - Identify the state before and after the upgrade, and the transition path
  - Prepare the application for upgrades
  - Identify potential points for scheduling upgrades
  - Building component-based applications to be born upgradeable
- Runtime handling of live upgrades
  - Determining quiescence
  - Run-time dependency tracking in a distributed system
  - Staggering out upgrades without incurring downtime

Looking Ahead …….

- OMG (CORBA standards body) in the process of drafting an RFP for RT-FT middleware
- Consider performance, configurability and fault-tolerance
  - 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 <u>and</u> timing needs
  - Tuning of the application's fault-tolerance configurations
  - Query of the provided "level" of fault-tolerance <u>and</u> quality-of-service
  - Scheduling of fault-tolerance activities (fault-recovery)

Current Release of MEAD

- Features
  - Active replication, warm passive replication, resource monitoring
  - Focus on CORBA applications (upcoming – CCM and EJB)
  - Tunable parameters: number of replicas, replication style, checkpointing frequency
- Obtaining MEAD
  - `/groups/pces/uav_oep/mead_cmu/release/` on `users.emulab.net`
- MEAD User Support
  - Manual: http://www.ece.cmu.edu/~mead/release/index.html
  - Problem-reporting
    - http://www.ece.cmu.edu/~mead/release/mead-support-request.html
  - You can also email us at mead-support@lists.andrew.cmu.edu

Teaching Students These Skills

- Mixed class of students – software engineering, electrical engineering, computer science
- Semester-long project – pick a middleware platform (CORBA, J2EE, .NET, …..)
- Baseline
  - Distributed application with reliability, scalability and timing requirements
- Fault-tolerant baseline
  - Evaluate the fault-tolerance (as compared with the non-fault-tolerant version)
- "Real-time" fault-tolerant baseline
  - Make the fault-tolerant baseline application exhibit timing/latency guarantees
- Scalable real-time fault-tolerant final system
  - Make your fault-tolerant real-time baseline application maintain performance, even with 1000 threads, 100 processes, etc.
- Understand the fault-tolerance vs. real-time vs. performance trade-offs
- http://www.ece.cmu.edu/~ece749

Summary

- MEAD’s configurable fault-tolerance
  - Born out of lessons learned in deploying previous fault-tolerant systems
- Advisor to take the guesswork out of configuring fault-tolerance
- “Knobs” for the appropriate expression of a user’s requirements
- Offline program analysis to extract application-level knowledge
- Proactive fault-recovery mechanisms

For More Information

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

Tudor Dumitras, Aaron Paulos, Soila Pertet, Charlie Reverte,
Joe Slember, Deepti Srivastava