

# Real-time Recovery from Distributable Thread Failures

October 2006



<http://real-time.ece.vt.edu>

**MITRE**

<http://www.mitre.org>

**Edward Curley**

*alias@vt.edu*

Master's Student  
Virginia Tech

**Dr. Binoy Ravindran**

*binoy@vt.edu*

Associate Professor  
Virginia Tech

**Jonathan S. Anderson**

*andersoj@andersoj.org*

Ph.D. Student / Lead Engineer  
Virginia Tech / MITRE

**Dr. E. Douglas Jensen**

*jensen@real-time.org*

Consulting Scientist  
The MITRE Corporation

Slides available at <http://andersoj.org/srds06>

# Research Context

- Distributed, real-time systems
  - Timeliness expressed in terms of *TUF/UA* constraints
- Concurrent and sequential control flow
  - Expressed in terms of *distributable threads*
- Application time constraints: seconds to minutes
  - Disaster response command and control
  - Network-centric battle management
- Dynamic network conditions

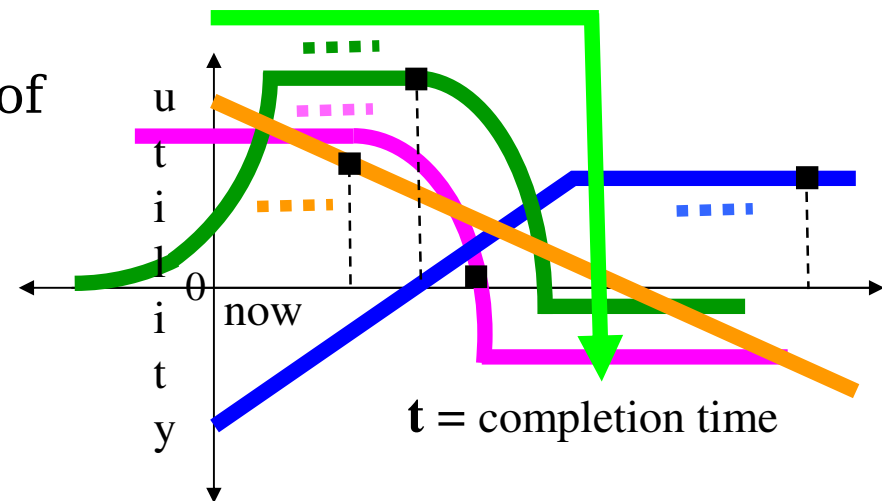
# Agenda

- Application Models
  - TUF/UA Scheduling
  - Distributable Threads and Integrity
- Thread Integrity
  - Thread Polling and TPR
  - Family of Real-Time Thread Integrity Policies
- AUA Scheduling Algorithm
  - Scheduler Performance
- Distributed Real-Time Specification for Java
- Coastal Air Defense Demonstration Application

# Time Utility Functions and Utility Accrual Scheduling

- Time/utility functions (TUF's)
  - express utility to the system of completing an activity as an application- or situation-specific function of when it completes
  - generalize priority and deadline scheduling criteria
- Utility accrual (UA) scheduling algorithms schedule activities according to optimality criteria based on
  - accruing utility – such as maximizing the sum of the utilities
  - satisfying dependencies such as resource constraints, etc.

## Example



### General time/utility functions

- .... Expected or max execution time
- Example scheduled completion times

**Schedule to maximize**  

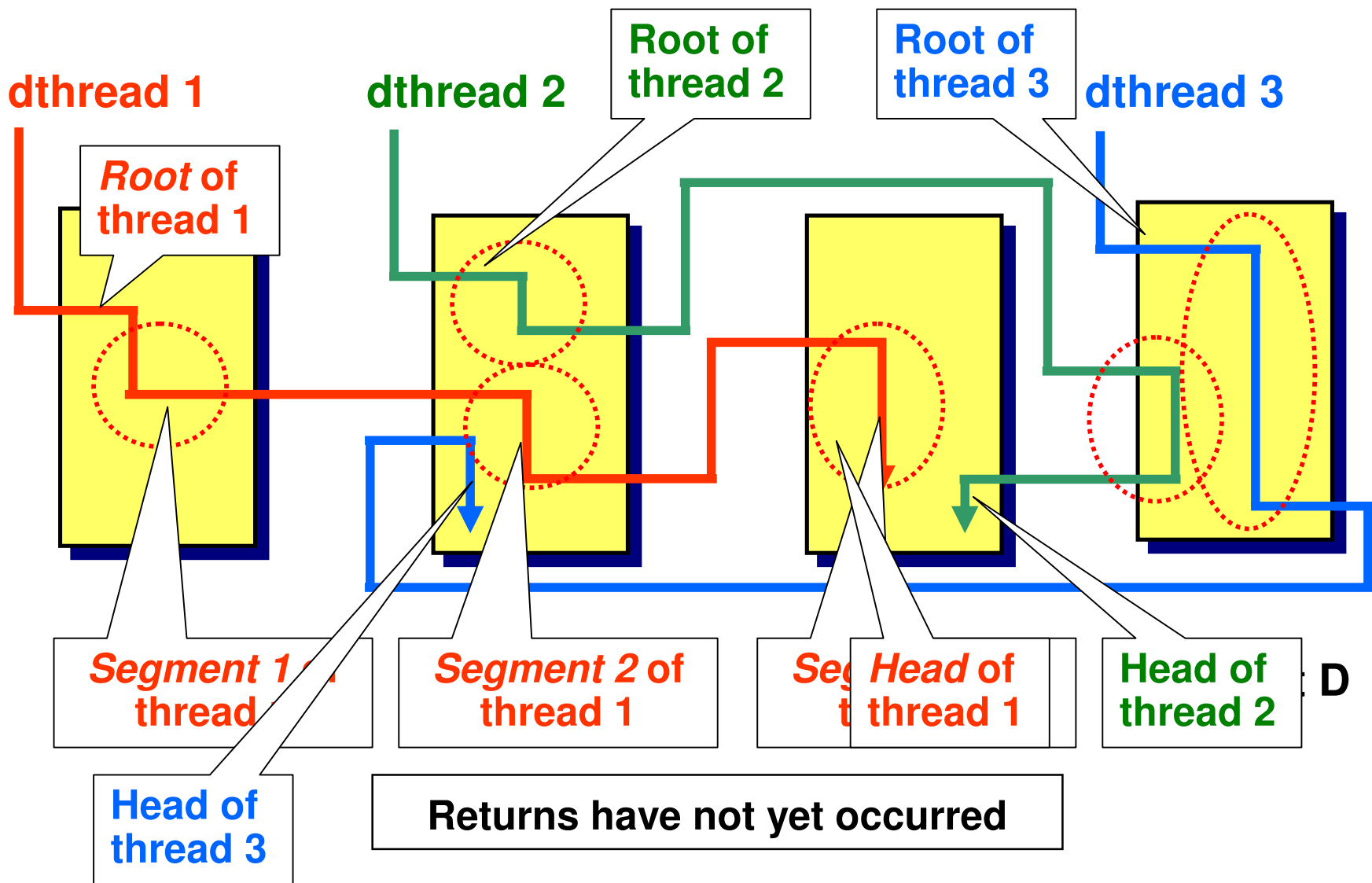
$$U = \sum u_i$$

# Distributable Threads

- Many distributed applications may naturally be structured as
  - sequential (i.e., linear) flows of execution and data...
  - ...within and among known objects (no discovery)...
  - ... moving asynchronously and concurrently
- A ***distributable thread*** is a single logically distinct locus of control flow movement that extends and retracts through (potentially) remote objects
- Distributable Threads have appeared in
  - CMU Alpha research OS (1987)
  - OSF/RI Mach/Mk7.3a (1991)
  - RT-CORBA 2.0 Dynamic Scheduling (2001)
  - DRTSJ (Upcoming)

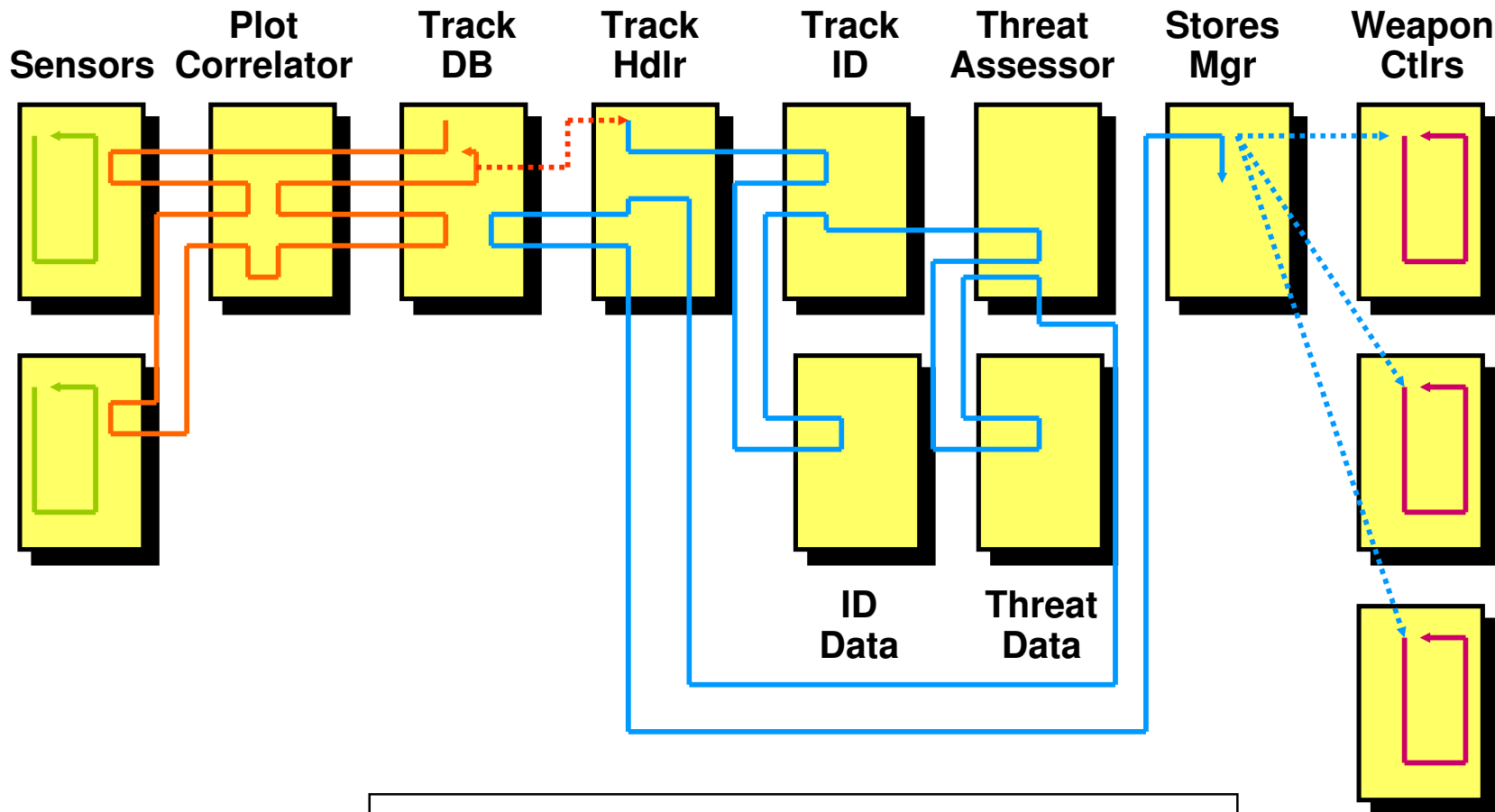
# Distributable Threads:

Extend and retract across nodes



# Distributable Threads: Example Application

## Battle Management Sensor-to-Shooter Chain



*Taken from General Dynamics Battle Management Demonstration*

# Distributable Threads:

## Programming Model

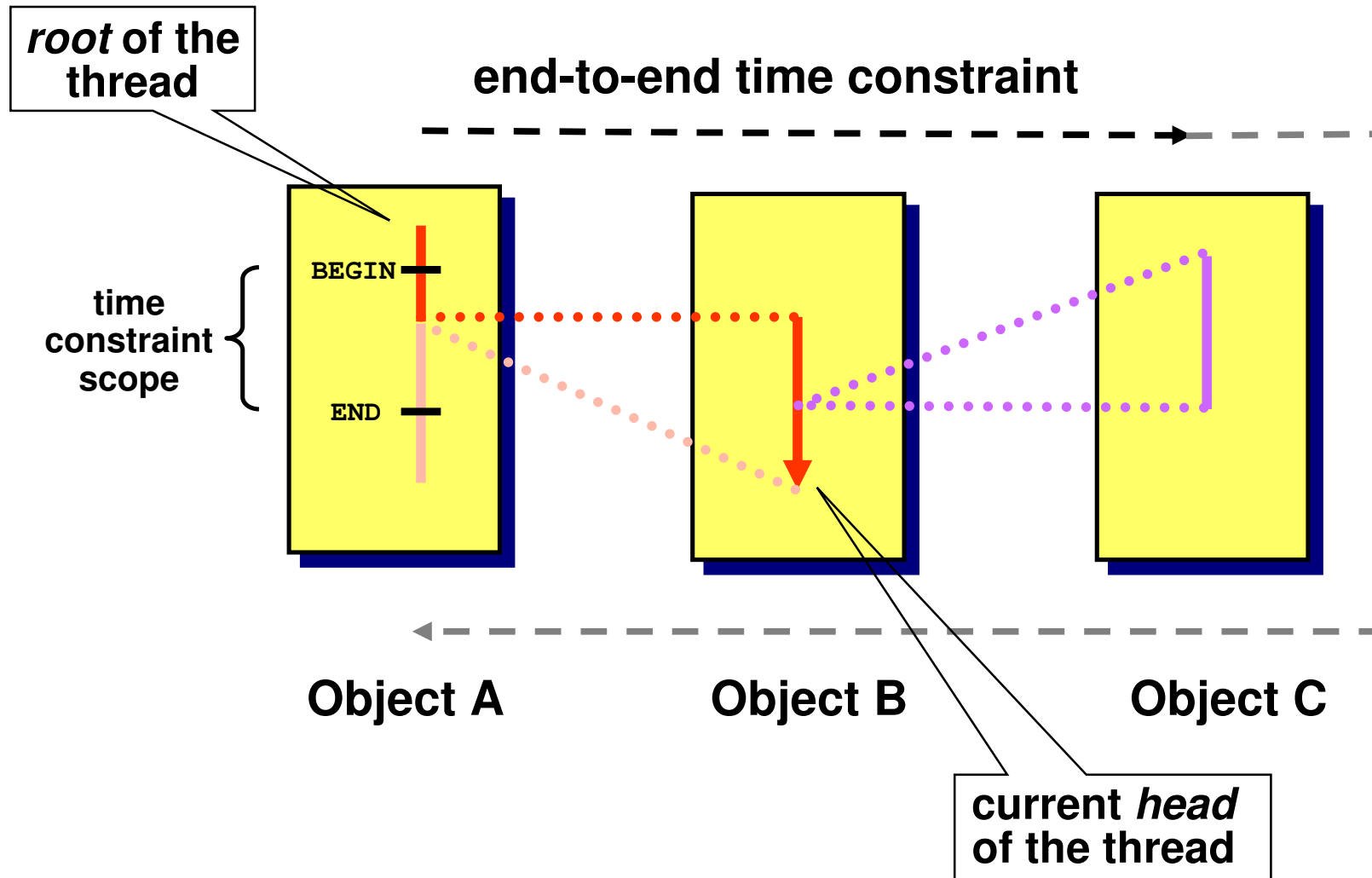
- No constraints on the presence, size, or structure of the data propagated with thread
- Commonly, input parameters propagated with invocations; results (“returns”) propagated back
- Invoked object's ID is known by the invoking object (i.e., the invoked object does not have to be discovered)
- End-to-end properties must be maintained: timeliness, fault management, control, etc...
- If the purpose for a thread is movement of associated data, model can be viewed as a data flow *or* control flow

# Distributable Threads:

## Propagating Context

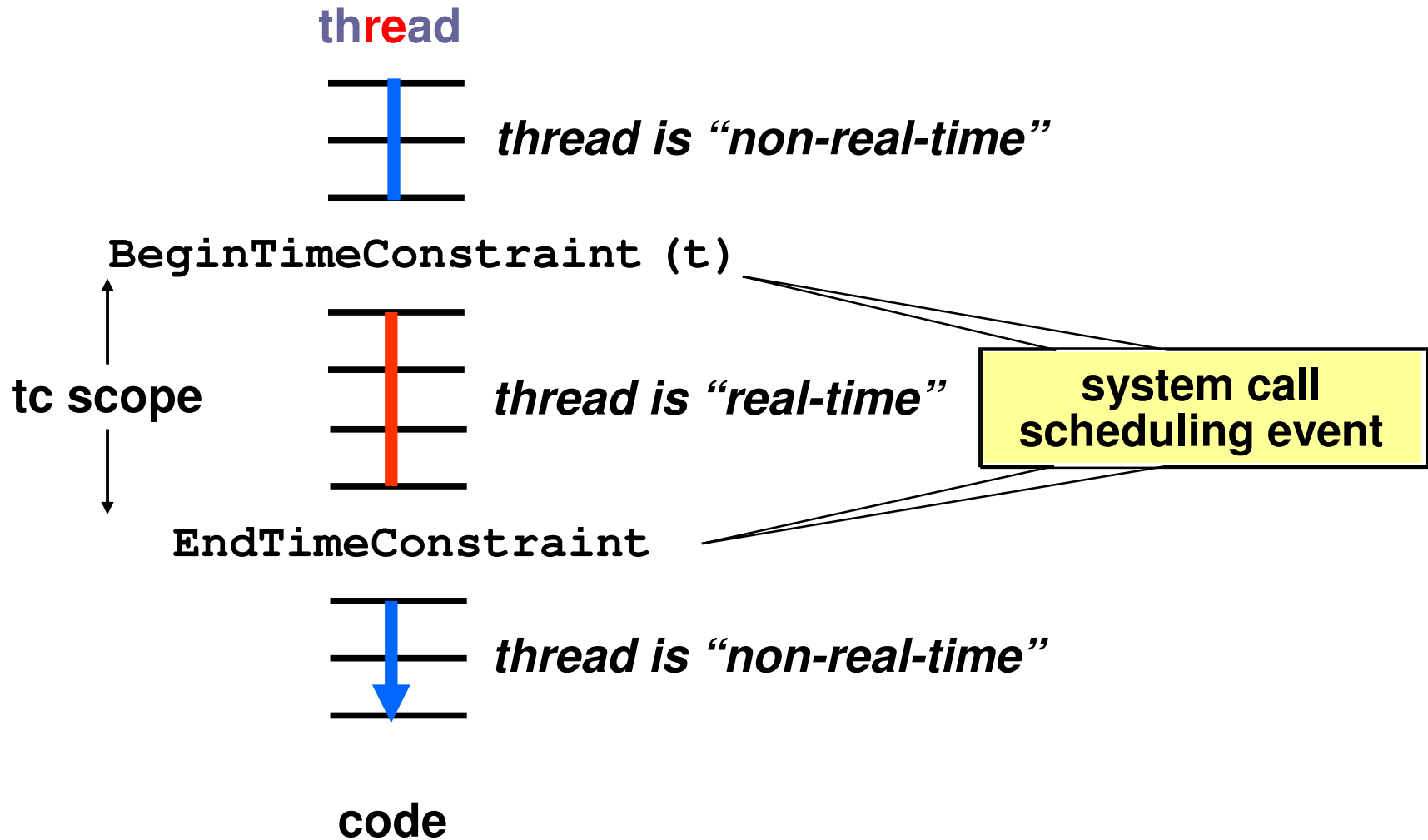
- Threads carry execution context across (object and) node boundaries, including its scheduling parameters (e.g., time constraints, execution time), identity, and security credentials
- The propagated thread context is intended to be used by node schedulers for resolving all node-local resource contention among threads such as that for node's
  - physical (e.g., CPU, I/O, memory, disk)
  - and logical (e.g., locks)resources, according to a discipline that provides acceptably optimal and predictable system-wide timeliness

# Distributable Threads: End-to-end time constraints

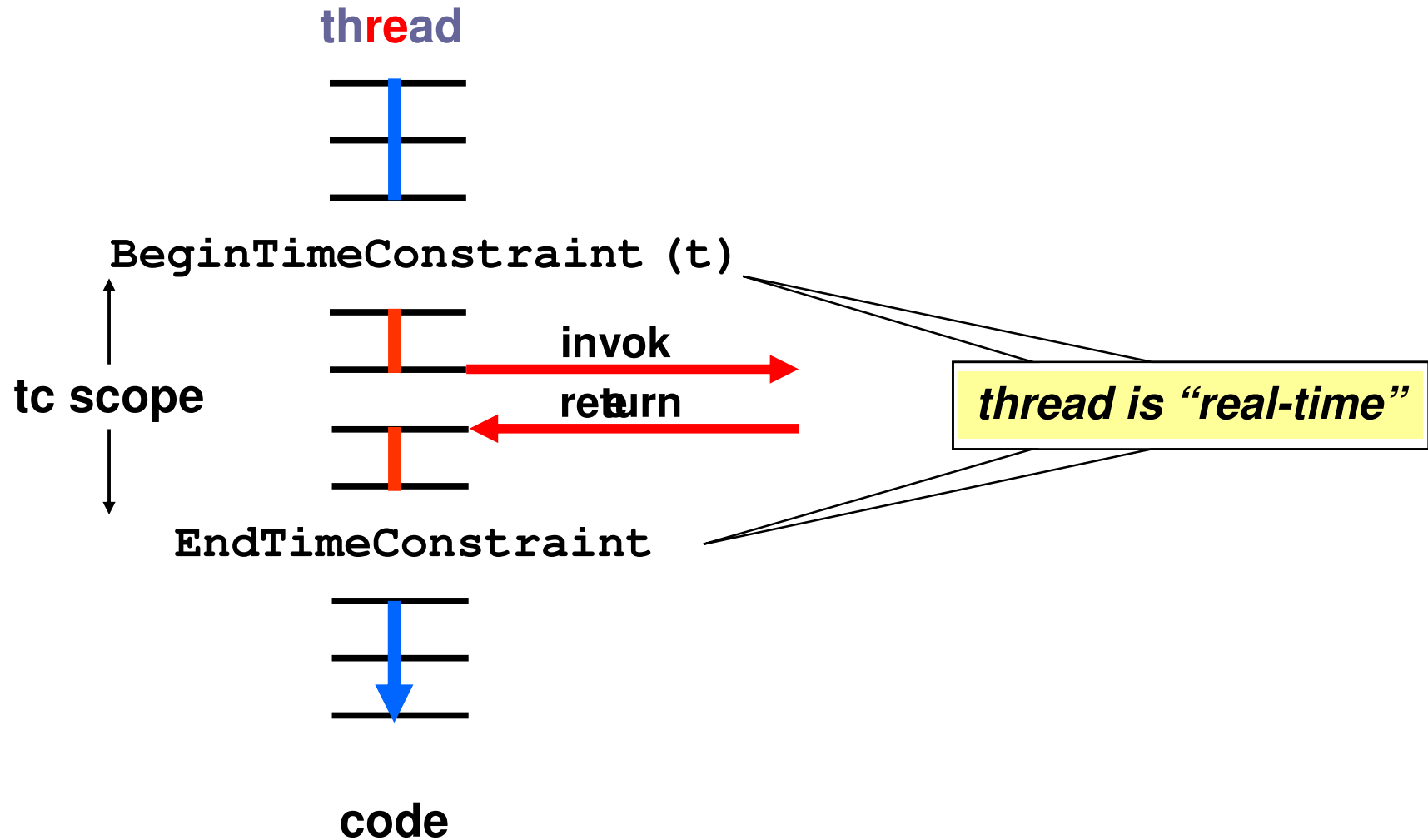


# Distributable Threads:

## End-to-end time constraints



# Distributable Threads: Time Constraint Propagation



# Partial Failures

- Non-trivial dynamic distributed systems should be presumed in a constant state of partial failure
- End-to-end integrity of distributable threads must be maintained at a level appropriate for the application
  - Application-specific policies and tuning
  - Integrity abstraction must be one that programmers and designers can reason about
  - Timely detection of failures endangering distributable thread correctness and timeliness
  - Timely application notification and recovery coordination

# Timely Thread Failure Recovery

## ■ Application Model

- Application consists of dynamic set of distributable threads
- Threads are characterized by TUF/UA time constraints
- Arbitrary node (crash) and communication failures

## ■ Objectives

- Maximize accrued (summed) utility...
- ...in the presence of node and communication failures...
- ...while enforcing thread integrity

## ■ Inability to detect and respond to failures

- Possibly wastes resources on orphaned tasks
- Delays applications' return to safe and productive state
- Both of these appear in the accrued utility metric

# Thread Integrity:

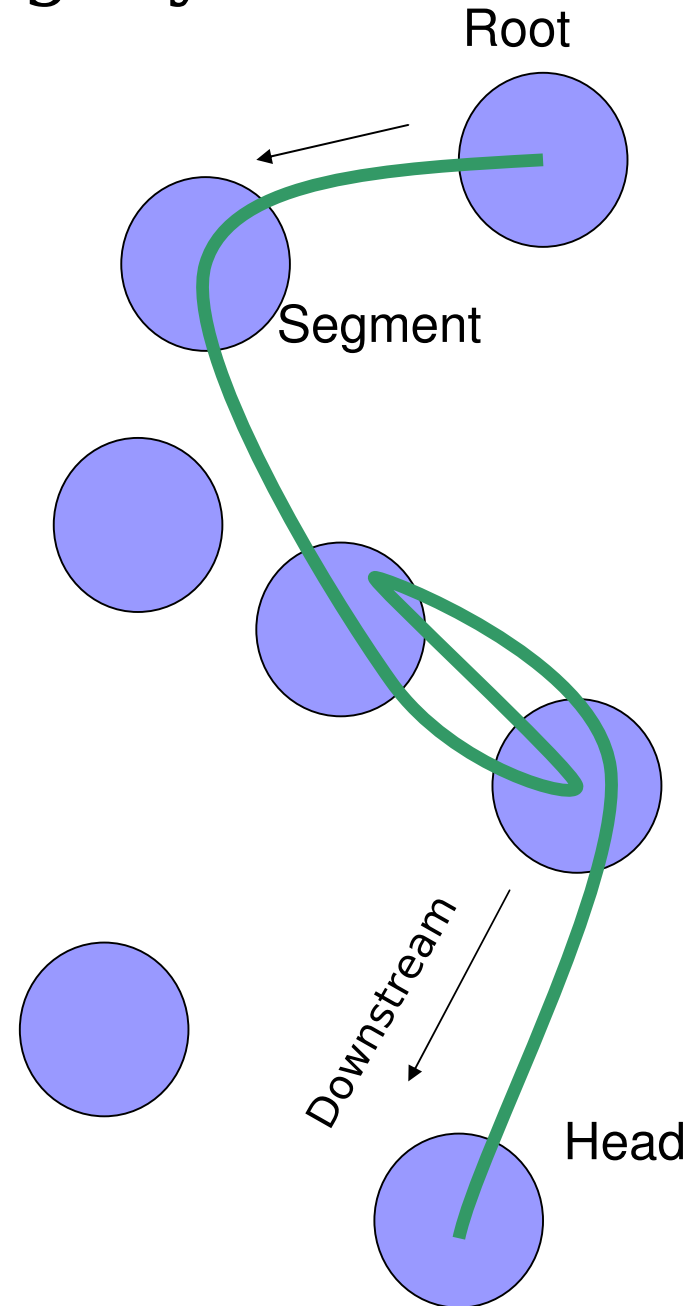
## Assumptions and Properties

- Network model
  - Reliable and/or unreliable communication
  - LAN vs. WAN vs. MANET class of time and connectivity
    - Represent extrema in the static/dynamic network continuum
- Node Failures
  - Node crash failures (fail-stop)
- TMAR Properties
  - End-to-end timeliness (how long to detect/correct a failure)
  - Responsiveness & accuracy
  - Interactions with local scheduler(s)
  - Ordered cleanup (precise, best-effort, no attempt)
  - Assurances about single-head nature of threads

# Distributable Thread Integrity

## Distributable Thread Primer

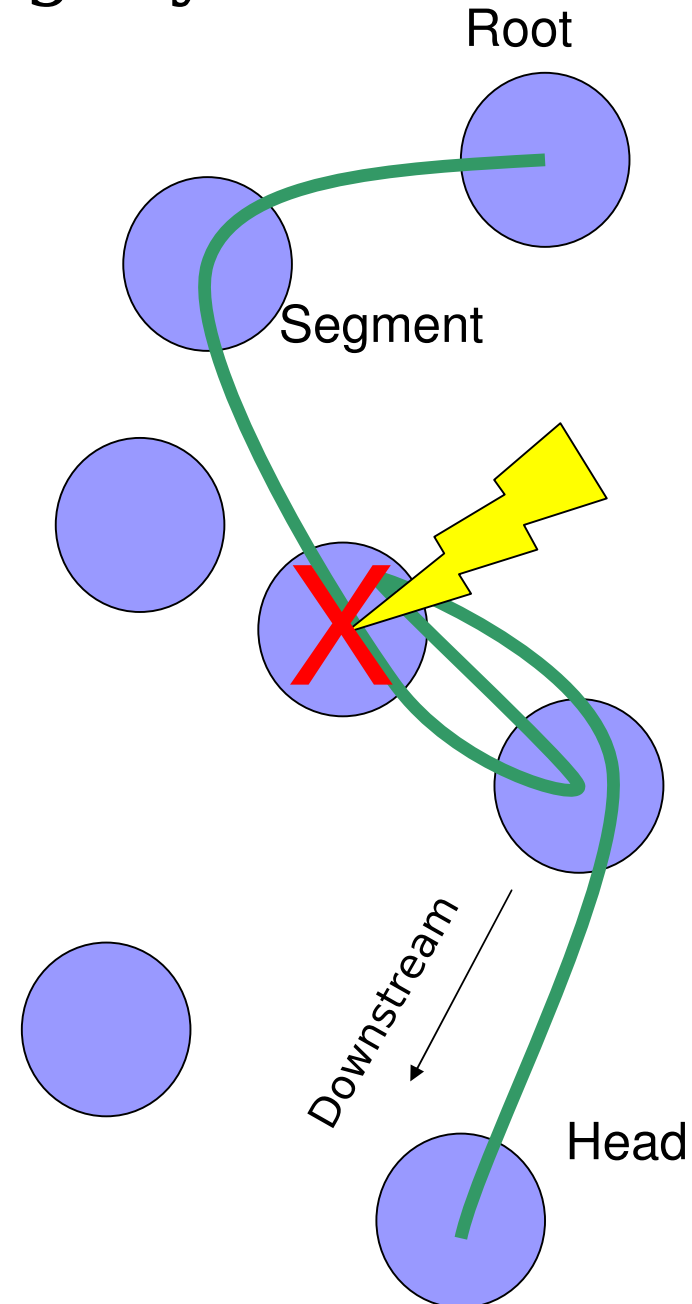
- Thread is instantiated at its root (“root node”)
- Thread may transit any node the network, so that its call-graph is dynamic and difficult to measure
- Goal of enforcing the predicate that only a single, unique point of control (in the application’s view) is active at any instant
- The “distributable thread” is realized by a collection of local threads, called “segments”



# Distributable Thread Integrity

## Thread Polling & TPR

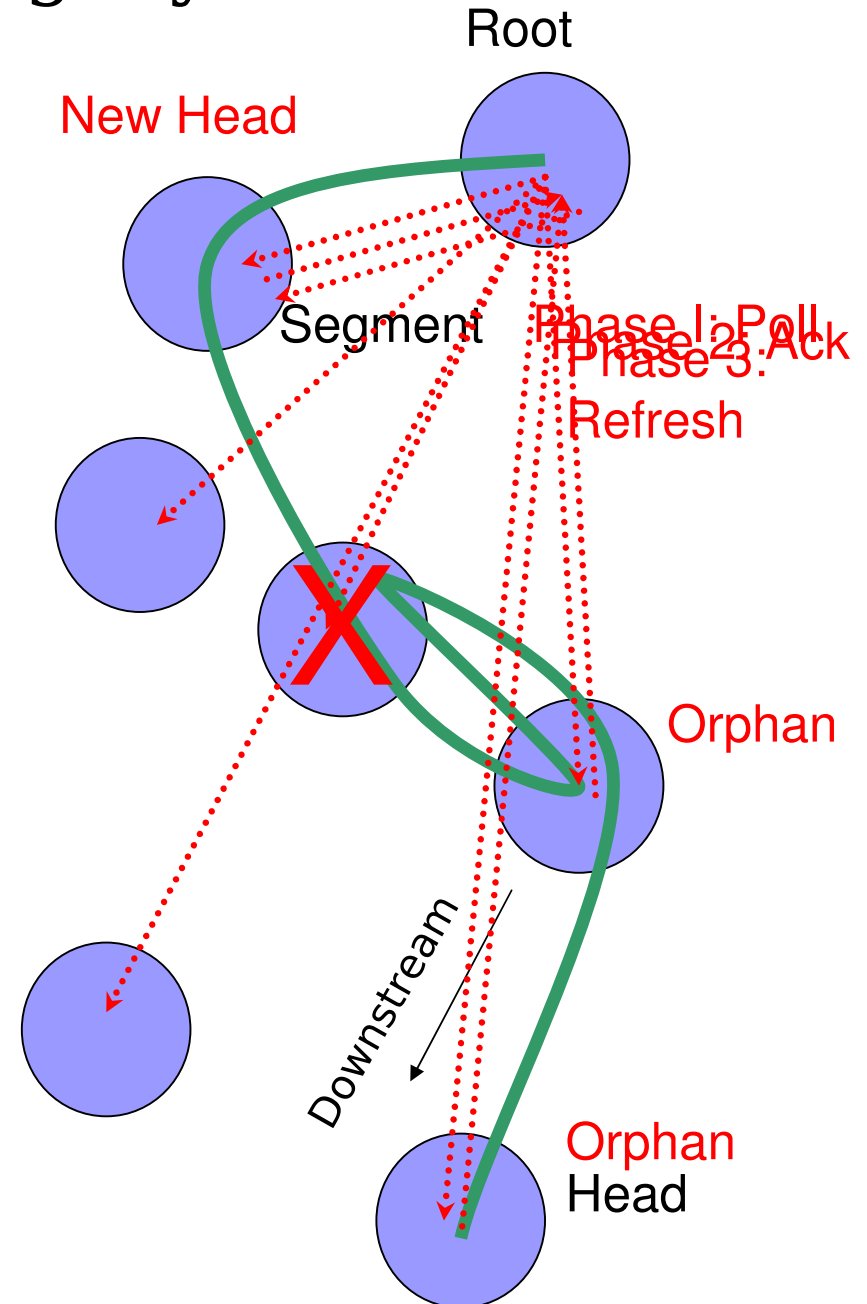
- **Phase I:** Each thread's root node broadcasts a "poll" message to all nodes
- **Phase II:** Every node hosting a segment of the thread responds ("Ack") with information about their segment and knowledge about the location of the head. The root assembles a tentative measure of the call graph.
- **Phase III:** If a persistent "break" or missing head is discovered, the thread is paused, cleanup commences on nodes downstream of the break, and control is returned to a new head.



# Distributable Thread Integrity

## Thread Polling & TPR

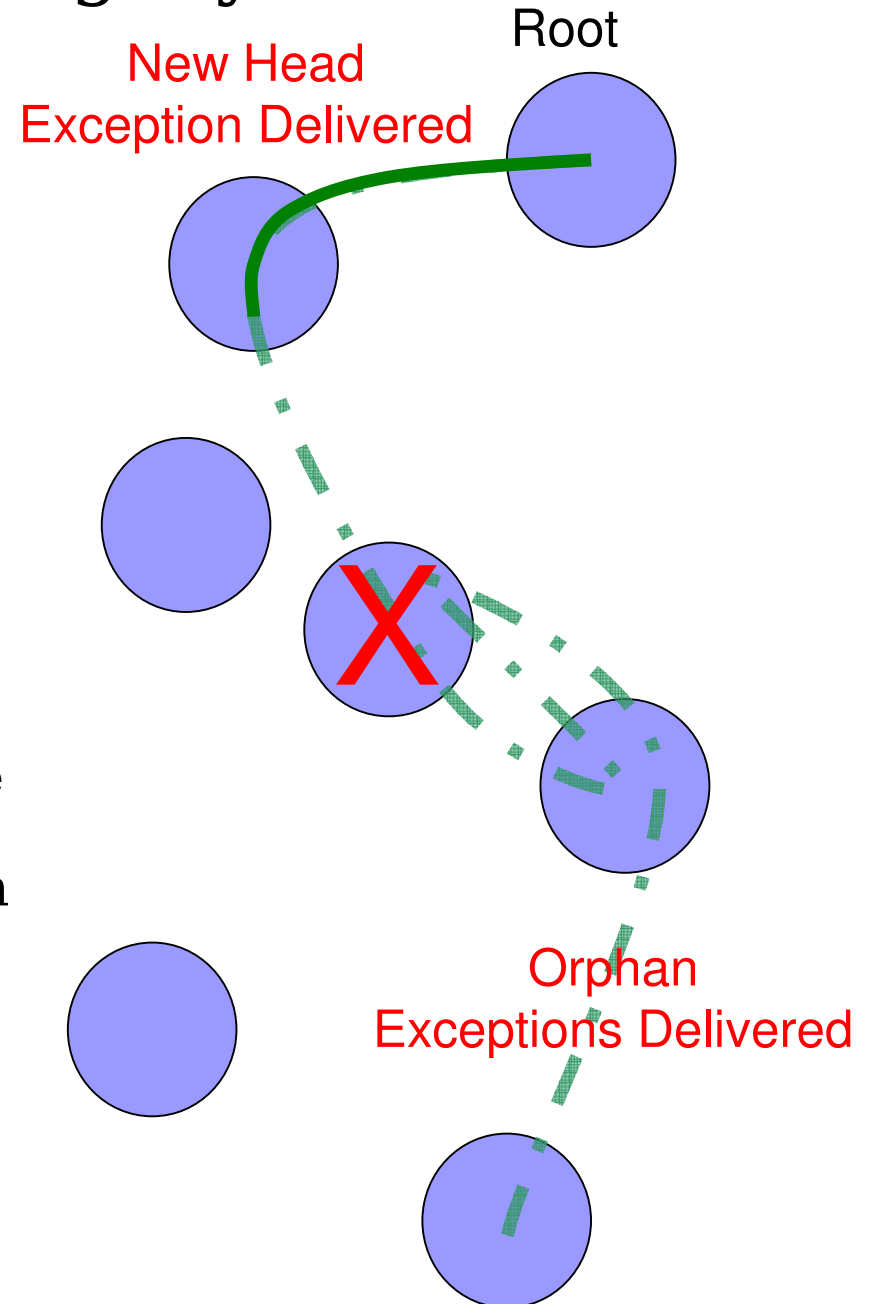
- **Phase I:** Each thread's root node broadcasts a "poll" message to all nodes
- **Phase II:** Every node hosting a segment of the thread responds ("Ack") with information about their segment and knowledge about the location of the head. The root assembles a tentative measure of the call graph.
- **Phase III:** If a persistent "break" or missing head is discovered, the thread is paused, cleanup commences on nodes downstream of the break, and control is returned to a new head.



# Distributable Thread Integrity

## Thread Polling & TPR

- **Phase I:** Each thread's root node broadcasts a "poll" message to all nodes
- **Phase II:** Every node hosting a segment of the thread responds ("Ack") with information about their segment and knowledge about the location of the head. The root assembles a tentative measure of the call graph.
- **Phase III:** If a persistent "break" or missing head is discovered, the thread is paused, cleanup commences on nodes downstream of the break, and control is returned to a new head.



# Distributable Thread Integrity

## D-TPR: History and Motivation

- TPR protocols derived from the Thread Polling protocol present in the Alpha Operating System
- Thread Polling was chosen due to its flexibility and relatively low overhead
- D-TPR Design Goals
  - Operate without broadcast capability
  - Handle intermittent wireless communication errors
  - Provide modifiable parameters to deal with differing network conditions
  - Provide Last-In-First-Out (LIFO) cleanup of orphans
  - Provide Bounded End-to-End recovery time from Distributed Thread breaks

# Distributable Thread Integrity

## D-TPR: Protocol

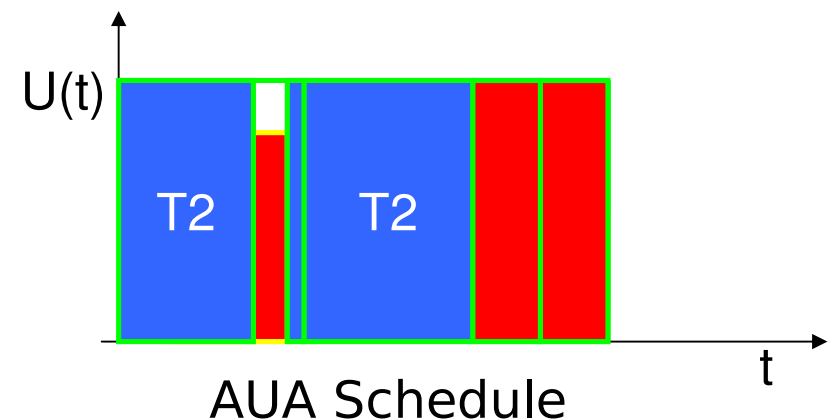
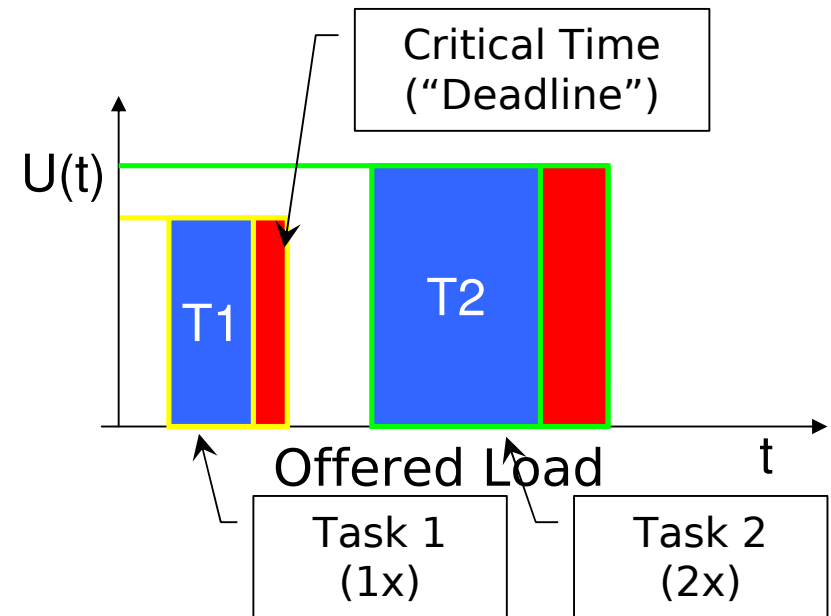
- Polling is maintained between two adjacent nodes that share a common distributable thread
- If poll-messages are not received after an application-derived time constraint, the link between the two nodes is considered severed and the DT is considered to be broken
- Recovering from a break
  - Segments downstream of the break marked as orphans
  - The segment directly upstream of the break becomes the new head of the DT, unless it is already marked as an orphan (multiple breaks)
  - Orphans cleaned up in Last-In-First-Out order (head to root)
  - Control is passed to the application to perform application-specific recovery actions

# Thread Integrity Protocol Family

- Prior Art Alpha OS and SRI Work
  - *Thread-Polling* — Root-centric, refresh-driven approach
  - *Node-Alive* — Conservative, head movement is synchronous with respect to all participating nodes, group communication protocol
  - “LAN” style assumptions; represent distinct points in the engineering trade space
- Our contributions
  - *TPR* — Thread polling, modified to provide end-to-end real-time assurances, again under LAN assumptions (SRDS’06)
  - *D-TPR* — Decentralized form of thread-polling (Curley Thesis)
  - *W-TPR* — Decentralized, MANET, and application-driven transient failure tolerance (Curley Thesis)
  - Designed to function in dynamic networks
  - Coupled with AUA scheduling algorithm

# AUA Scheduling Algorithm

- Derived from Clark's DASA algorithm
- Deterministic Feasibility Check
- Unit downward step TUFs
- Introduces deterministic guarantees for timely execution of abort code
- Optimal (EDF equivalent) in underload
- Each thread (or segment) characterized by
  - Maximum value
  - Critical time
  - Normal Execution
    - Logic
    - Execution time
  - Abort
    - Logic
    - Execution time



Blue: normal execution time  
Red: abort execution time

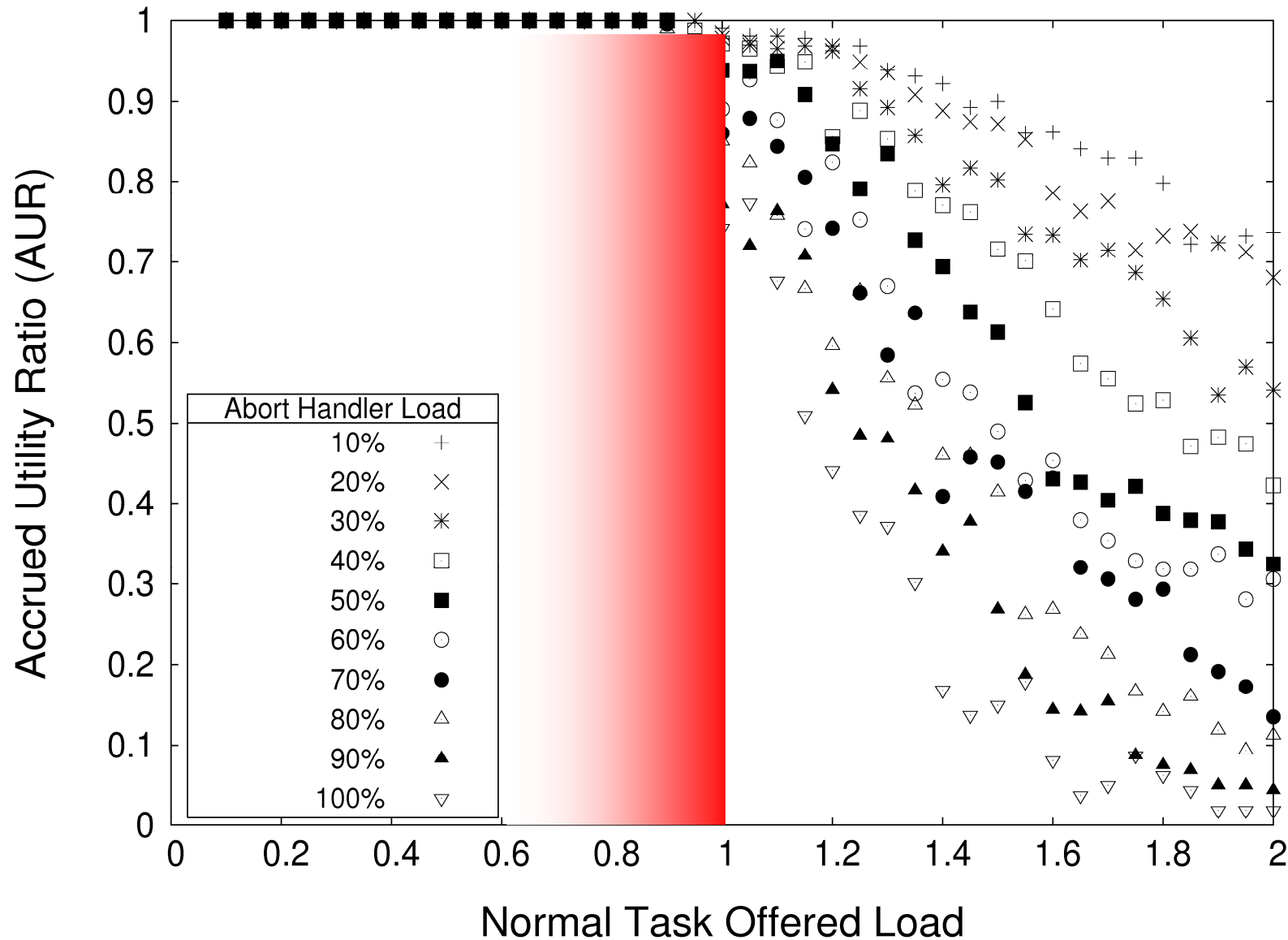
# TUF/UA Scheduling Performance

- Accrued Utility Ratio (AUR)
  - What fraction of the available utility is earned by a resulting schedule?
- Deadline Satisfaction Ratio (DSR)
  - What fraction of the offered tasks are completed by their deadline?
- Deadline Miss Load (DML)
  - At what offered load does a scheduler begin missing deadlines? (Ideal scheduler has  $DML == 1.0$ )

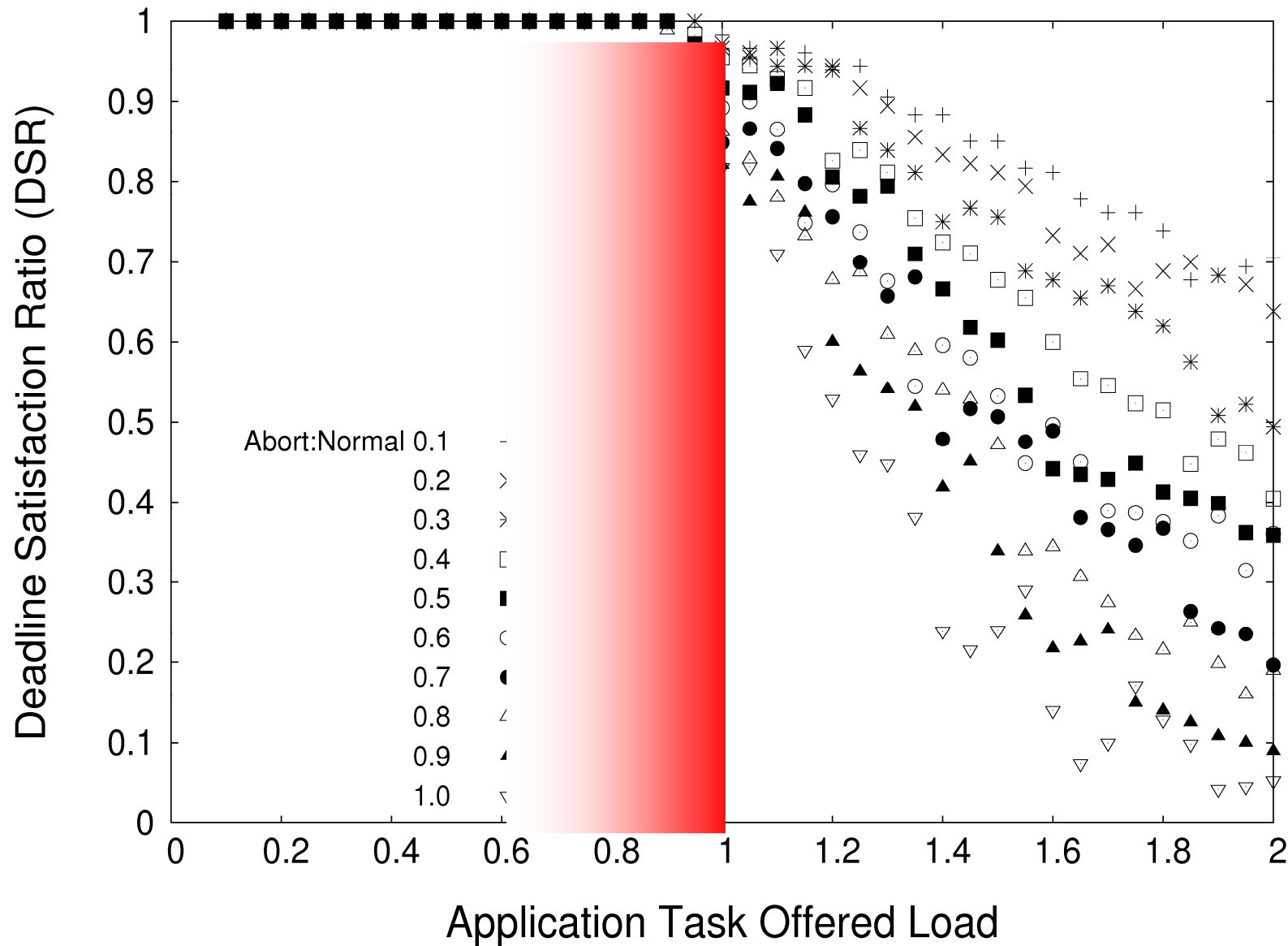
Metrics from following slides collected from an AUA implementation on Apogee's RTSJVM. CPU is 500MHz Intel P-III.

Thread time constraints and execution times are  $o(50ms)$ .

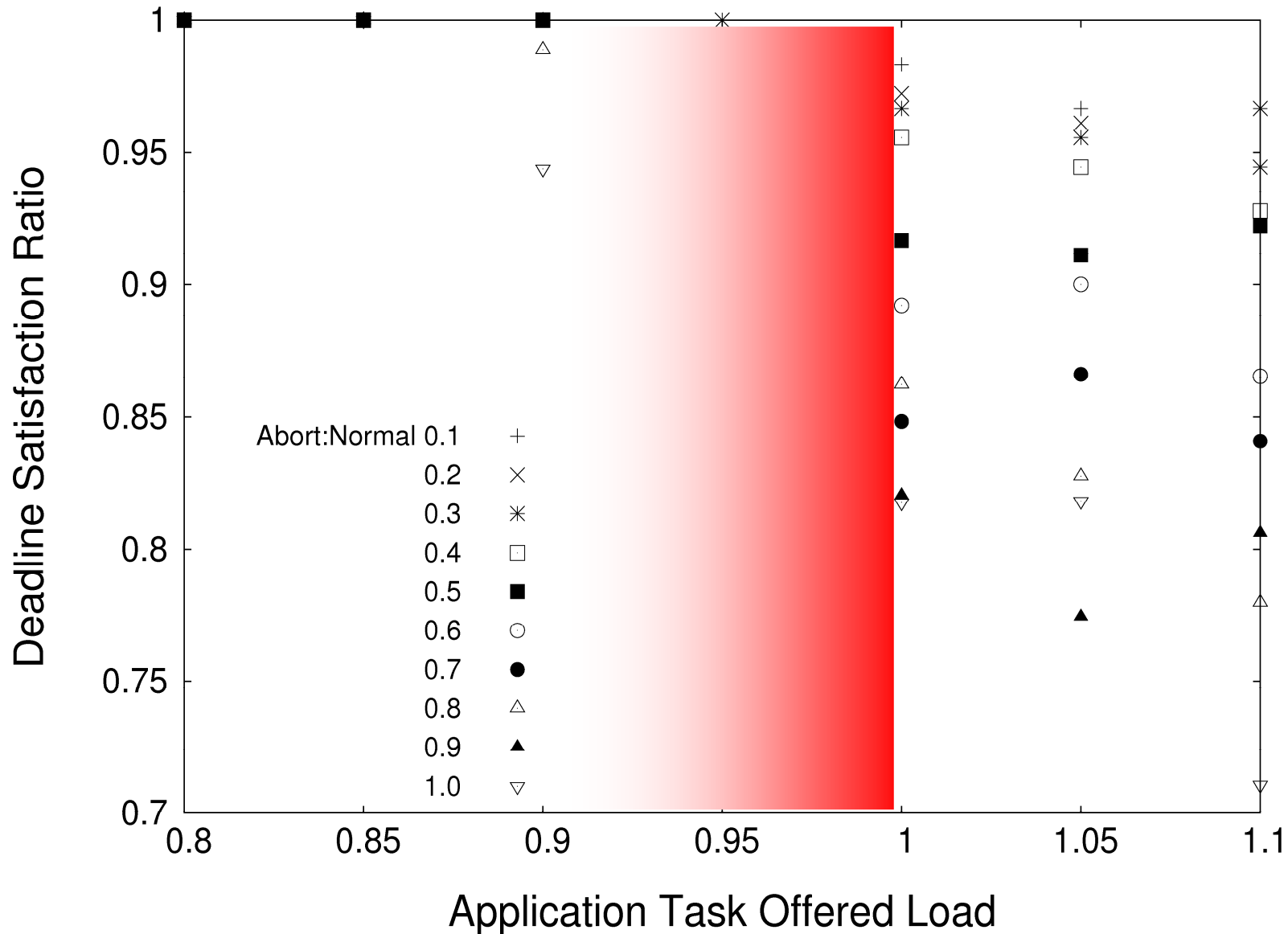
# AUA Scheduling Performance: Accrued Utility



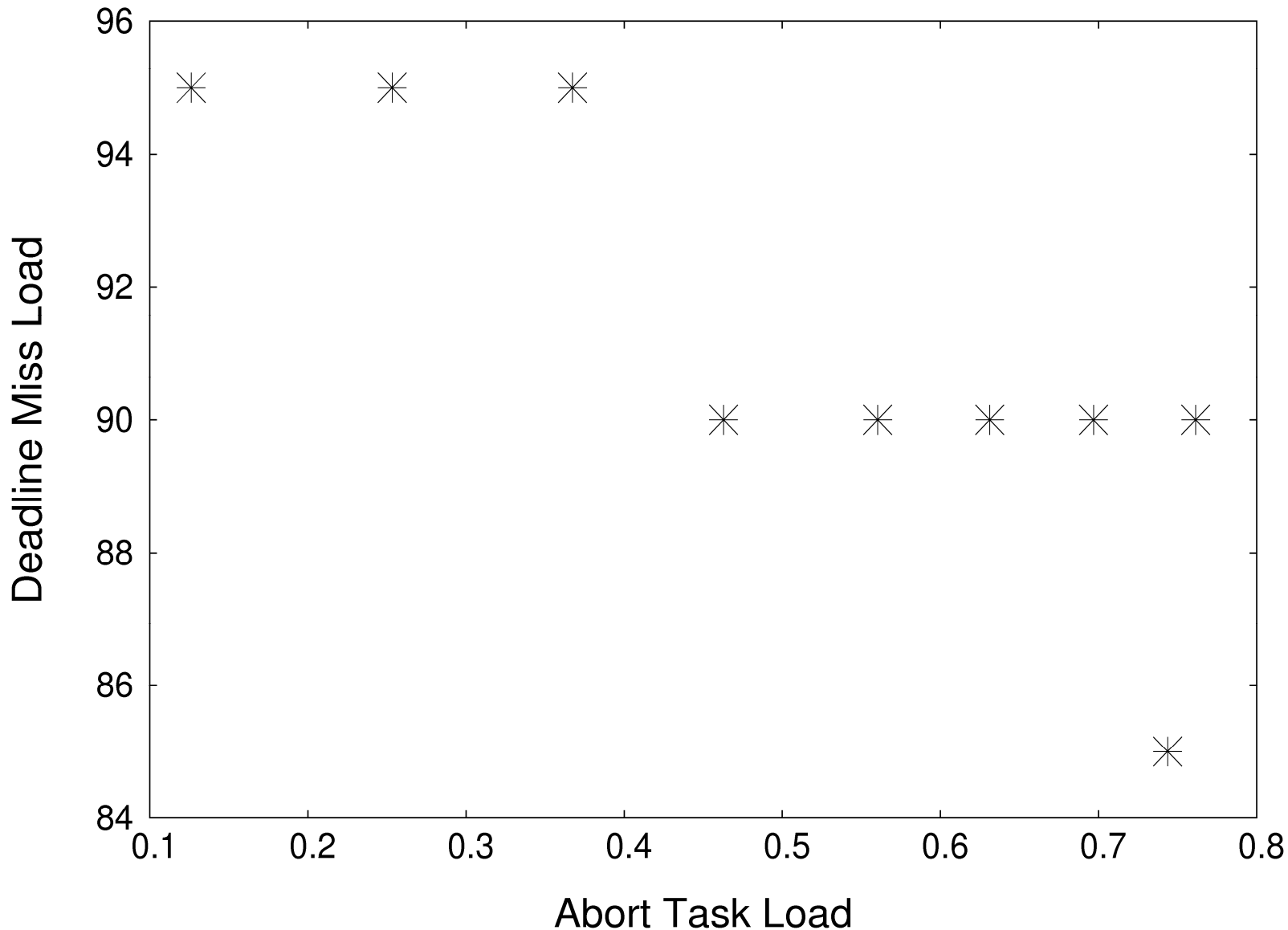
# AUA Scheduling Performance: Deadline Satisfaction



# AUA Scheduling Performance: Deadline Satisfaction



# AUA Scheduling Performance: Deadline Miss Load (Overhead)



# TPR/AUA Family Properties

- Bounded time thread fault recovery...
  - Thread fault detection latency
  - Delivery of application notification delay
  - End-to-end cleanup time
- ...while retaining system timeliness properties
  - Optimal scheduling in underloads, no failures
  - Maximal (heuristic) utility-accrual in overloads
  - Robust to transient failures which do not affect application time constraints

# Distributed Real-Time Specification for Java

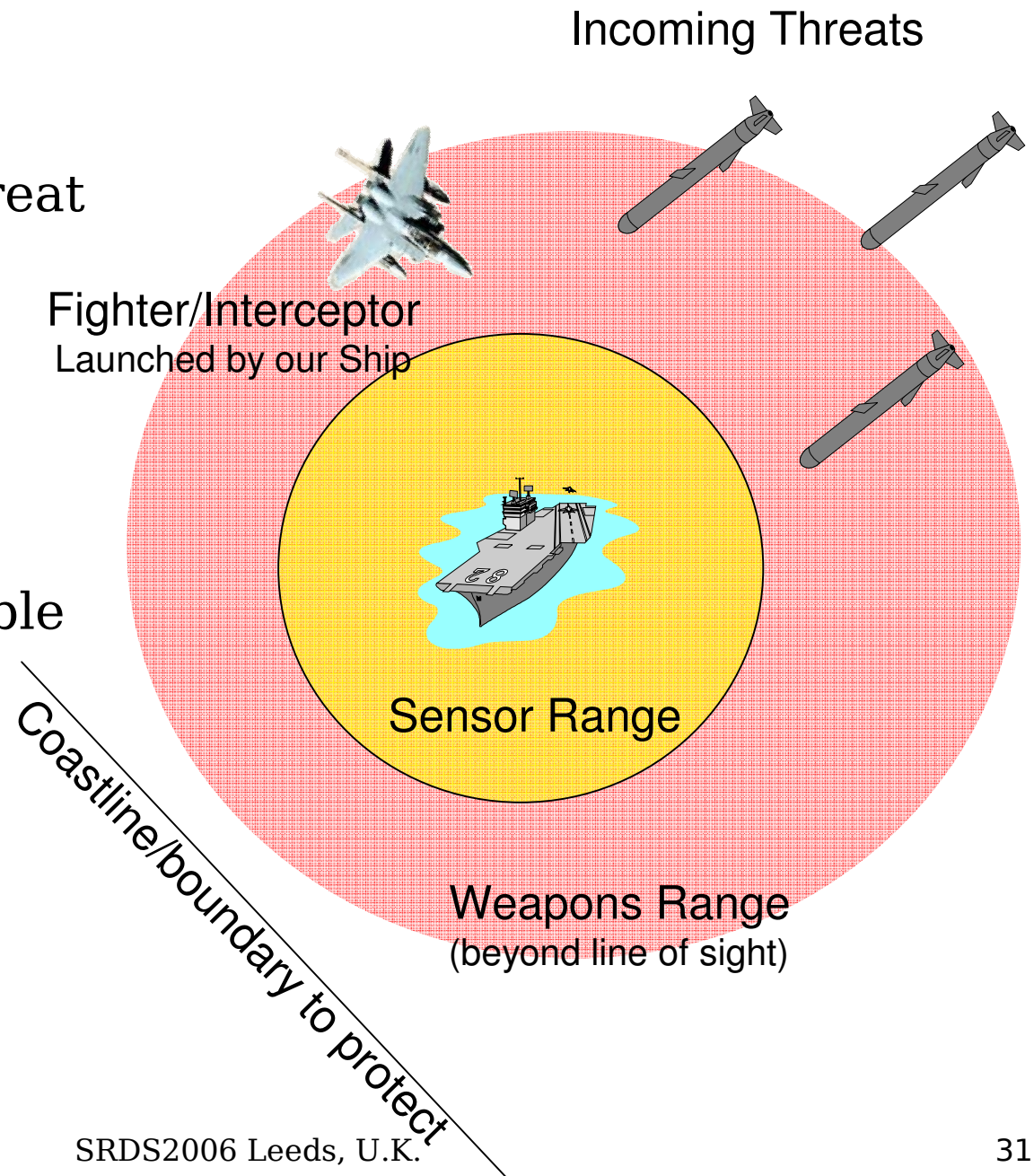
- DRTSJ is the key systems artifact of our work
- JSR-50 Specification Effort:
  - Led by MITRE Corporation and Virginia Tech
  - JVM Support: Apogee Software, Inc.
  - Expert Group Representation: AFRL, OIS, Lockheed-Martin, Boeing, Washington University, Nokia, etc...
- Planned submission of Early Draft Spec and Reference Implementation in November 2006
- DRTSJ Reference Implementation includes
  - Developmental scheduler and thread integrity code derived from ongoing research
  - Stable, standards-track subset targeted for JSR-50

*<http://jcp.org/en/jsr/detail?id=50>*

# Our Motivating Scenario:

## Air and Coastal Defense

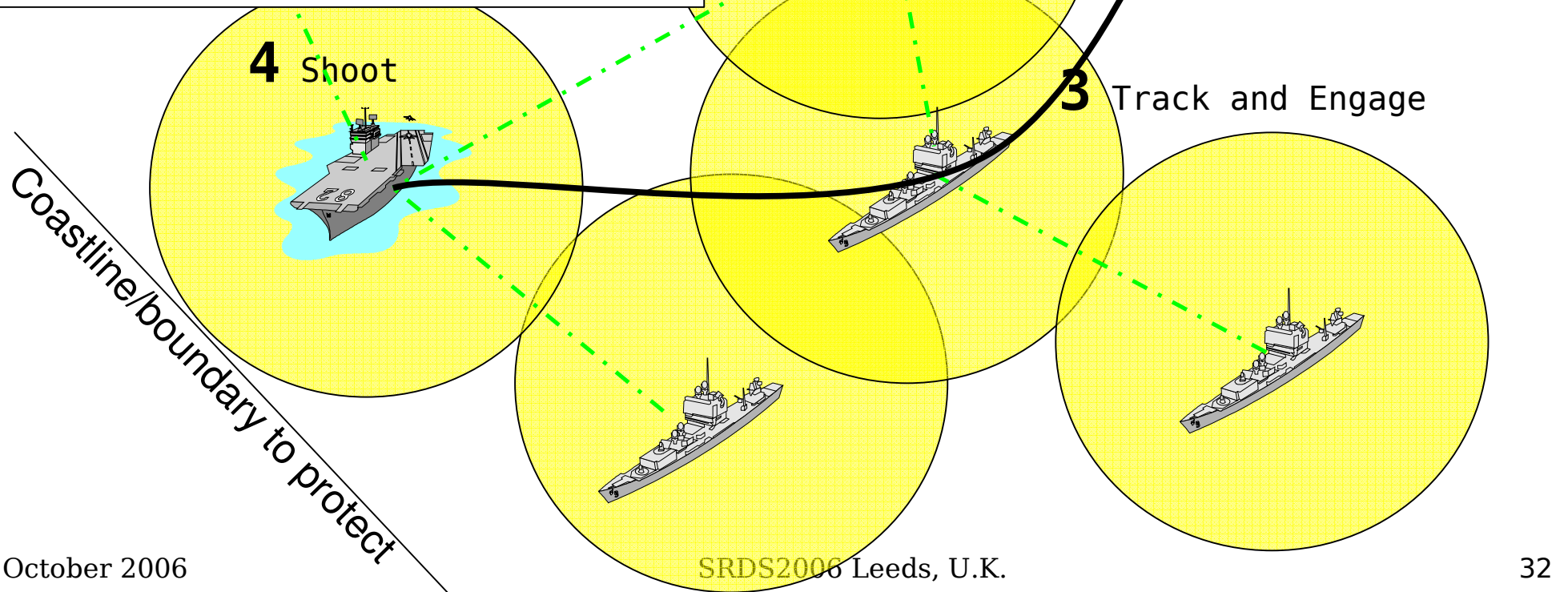
- Single-node coastal defense example
- Prosecuting missile threat is an inherently **sequential** mission:
  - 1) Detect
  - 2) Identify
  - 3) Track/Engage
  - 4) Shoot
- ... for which distributable threads are an ideal technical solution



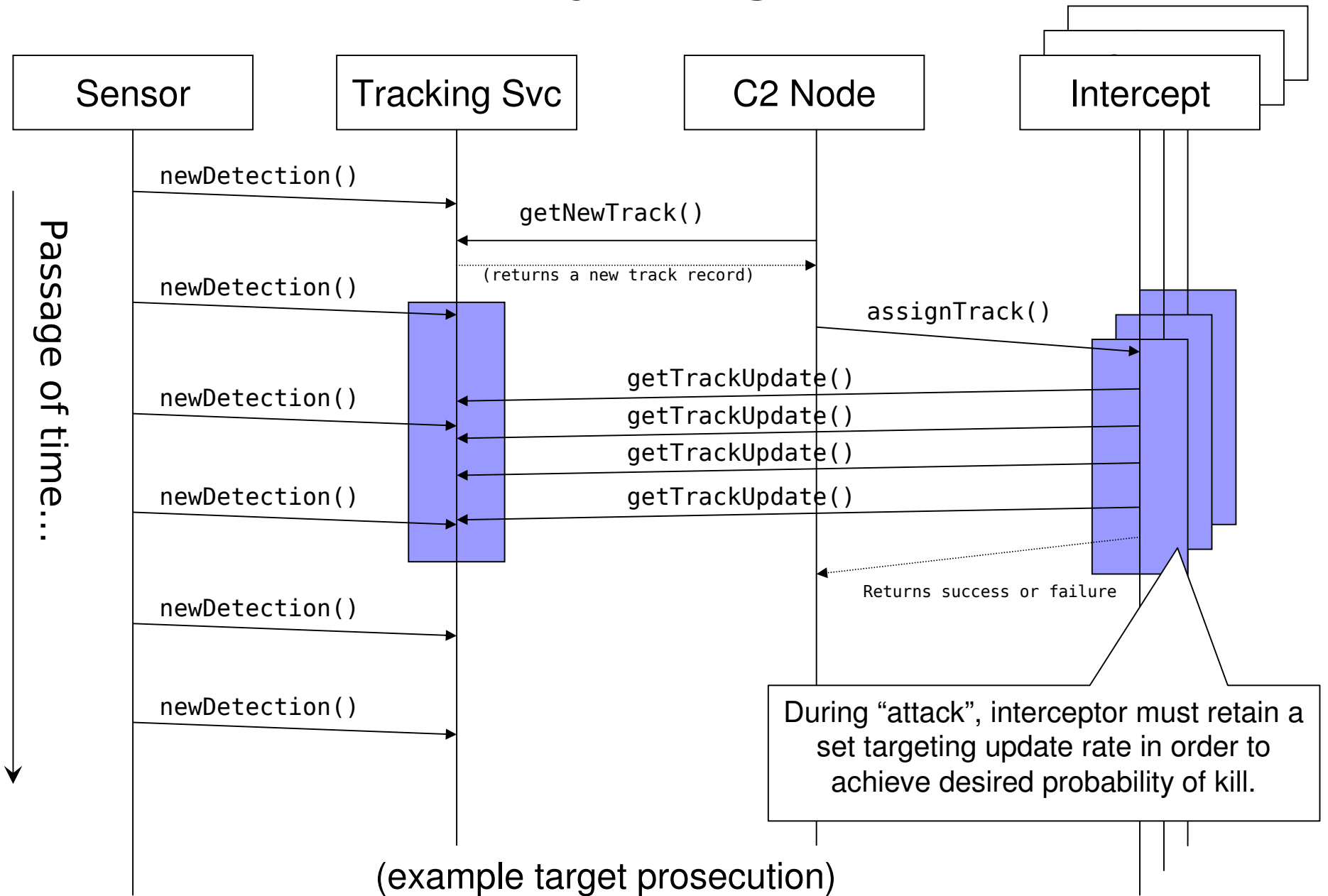
# Our Motivating Scenario: Air and Coastal Defense

**Challenge:** How to detect and respond to partial (communications) failure in the midst of a sequential activity execution?

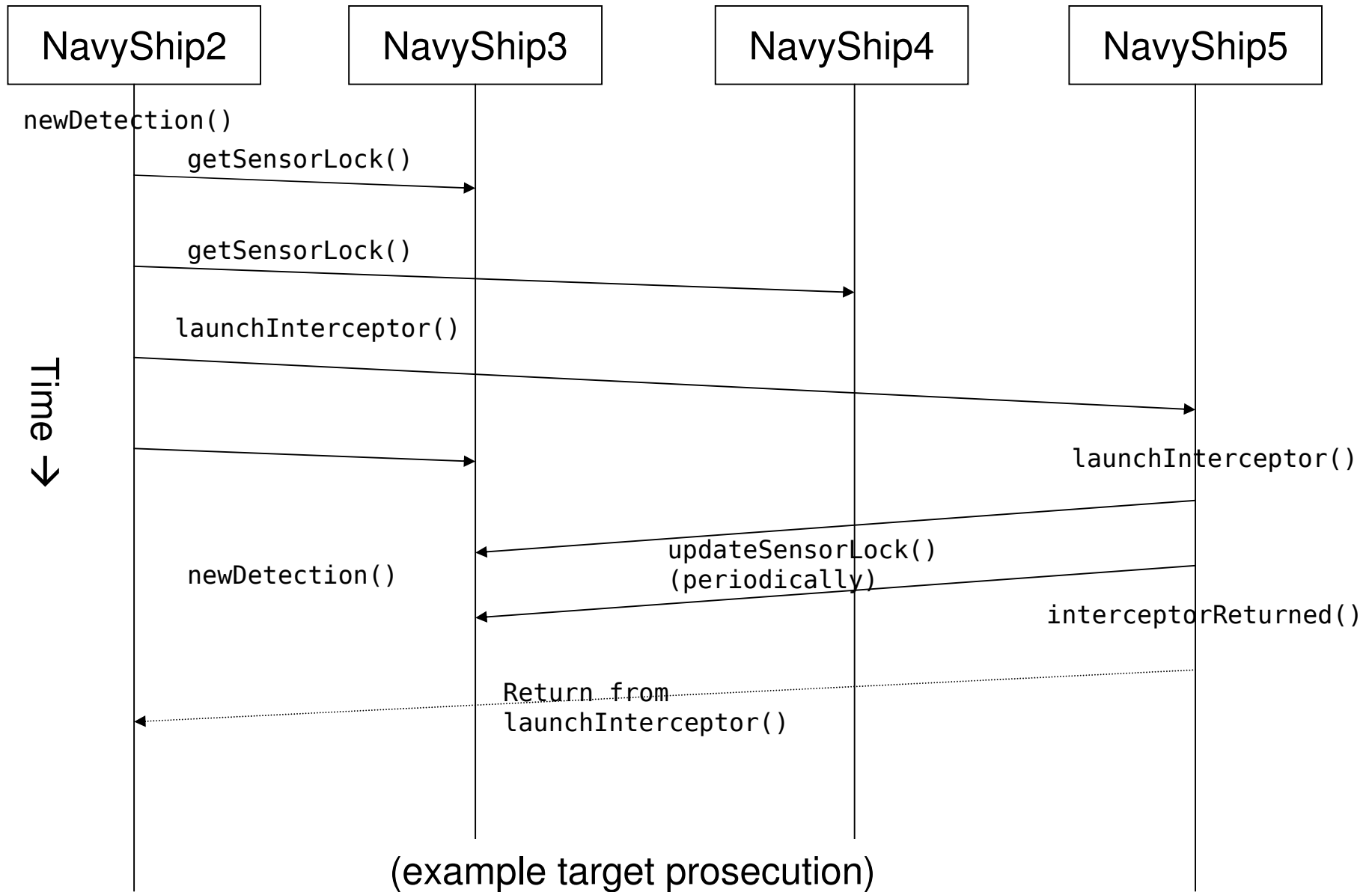
**Approach:** Implement mission in terms of distributable threads with thread integrity protocols



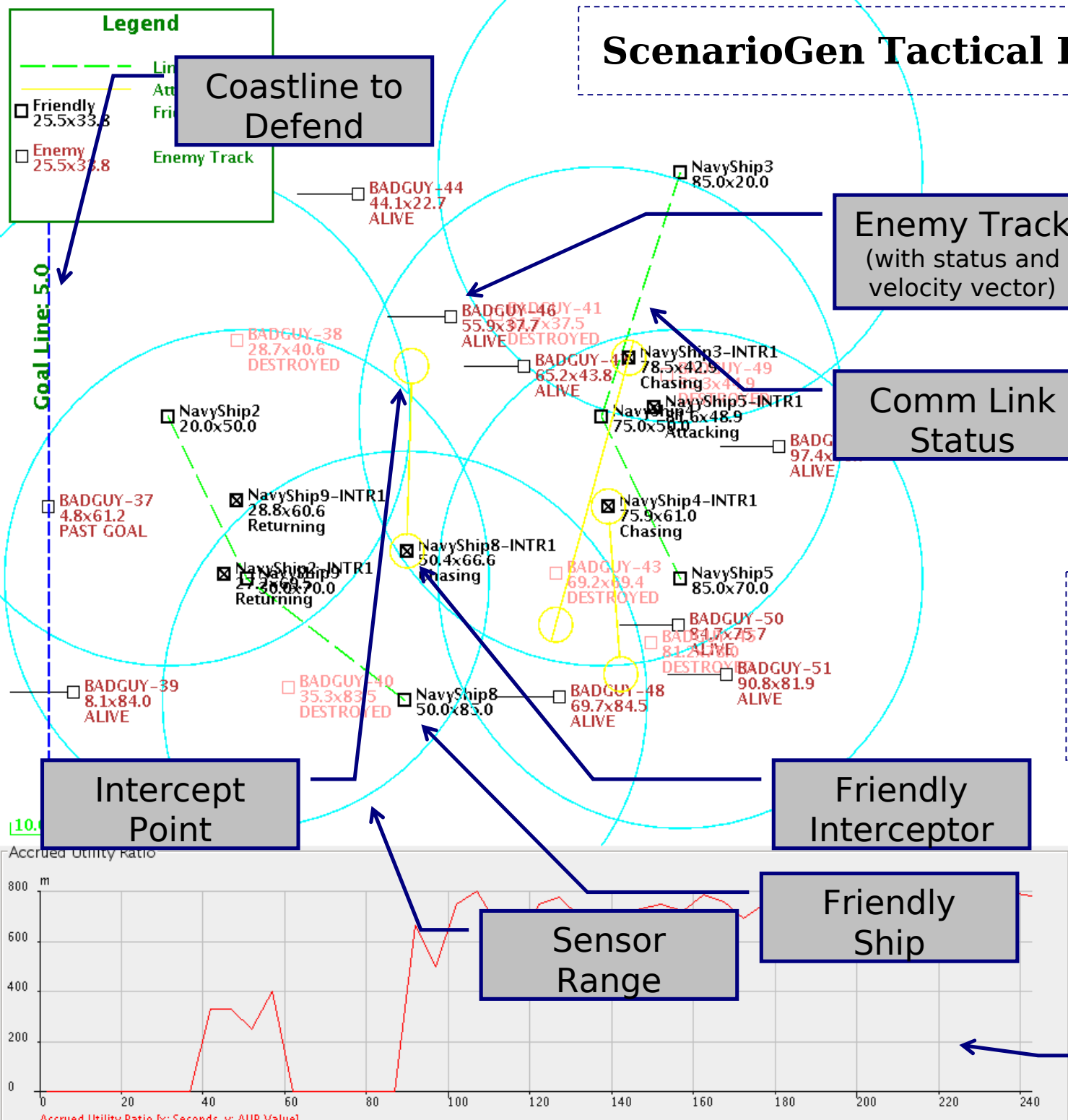
# Scenario Activity Diagram



# Scenario Activity Diagram (MANET)



# ScenarioGen Tactical Display (Example)



## Control Panel

Scenario Control

Start Scenario	Stop Scenario
Pause Scenario	Unpause Scenario
Reset Scenario	

NavyShip2-NavyShip4

NavyShip3-N

NavyShip4-M

Integrity Policy and Comm Settings

On Auto Off

On Auto Off

DRTSJ Thread Integrity Mode

Simple Timeout-Driven TMAR-W

Communications Mode

Static Comms Dynamic Comms

Statistics

Points: -180

Accrued Utility Ratio 0.782608695652174

Targets Destroyed 36

Past Goal Line 3

Scenario Status

Status: RUNNING

Mission Time: 00:03:57.446

NavyShip2	UNKNOWN	Inactive
NavyShip3	UNKNOWN	Inactive
NavyShip4	UNKNOWN	Inactive
NavyShip5	UNKNOWN	Inactive

Accrued Utility

# Results and Contributions

- Abort-Assured Utility Accrual Scheduling Algorithm (AUA)
  - Maximizes accrued utility
  - Optimal in underloads
  - Deterministic guarantee of timely failure recovery
- Family of timely thread integrity protocols
  - Formal assurances for thread integrity
  - End-to-End timely thread scheduling
  - End-to-End timely thread failure recovery
- Implementations in the DRTSJ RI
- Demonstration application
  - Experimental evidence that thread integrity improves application figures of merit

# Ongoing and Future Work

- Extensions to support resource dependencies
- Probabilistic models for communication delays
- “Automatic” application tuning to cope with transient failures
  
- Completion of the DRTSJ Draft Review specification and reference implementation

# Acknowledgements

Research supported by...



**US Navy Office of  
Naval Research (ONR)**  
*<http://www.onr.navy.mil/>*

Advanced Wireless Integrated  
Navy Network (AWINN)  
*<http://awinn.ece.vt.edu/>*

# MITRE

**The MITRE Corporation**

*<http://www.mitre.org>*

MITRE Sponsored Research:  
Predictable End-to-End Timeliness in  
Network-Centric Warfare Systems

# Real-time Recovery from Distributable Thread Failures

October 2006



<http://real-time.ece.vt.edu>

**MITRE**

<http://www.mitre.org>

**Edward Curley**

*alias@vt.edu*

Master's Student  
Virginia Tech

**Dr. Binoy Ravindran**

*binoy@vt.edu*

Associate Professor  
Virginia Tech

**Jonathan S. Anderson**

*andersoj@andersoj.org*

Ph.D. Student / Lead Engineer  
Virginia Tech / MITRE

**Dr. E. Douglas Jensen**

*jensen@real-time.org*

Consulting Scientist  
The MITRE Corporation

Slides available at <http://andersoj.org/srds06>

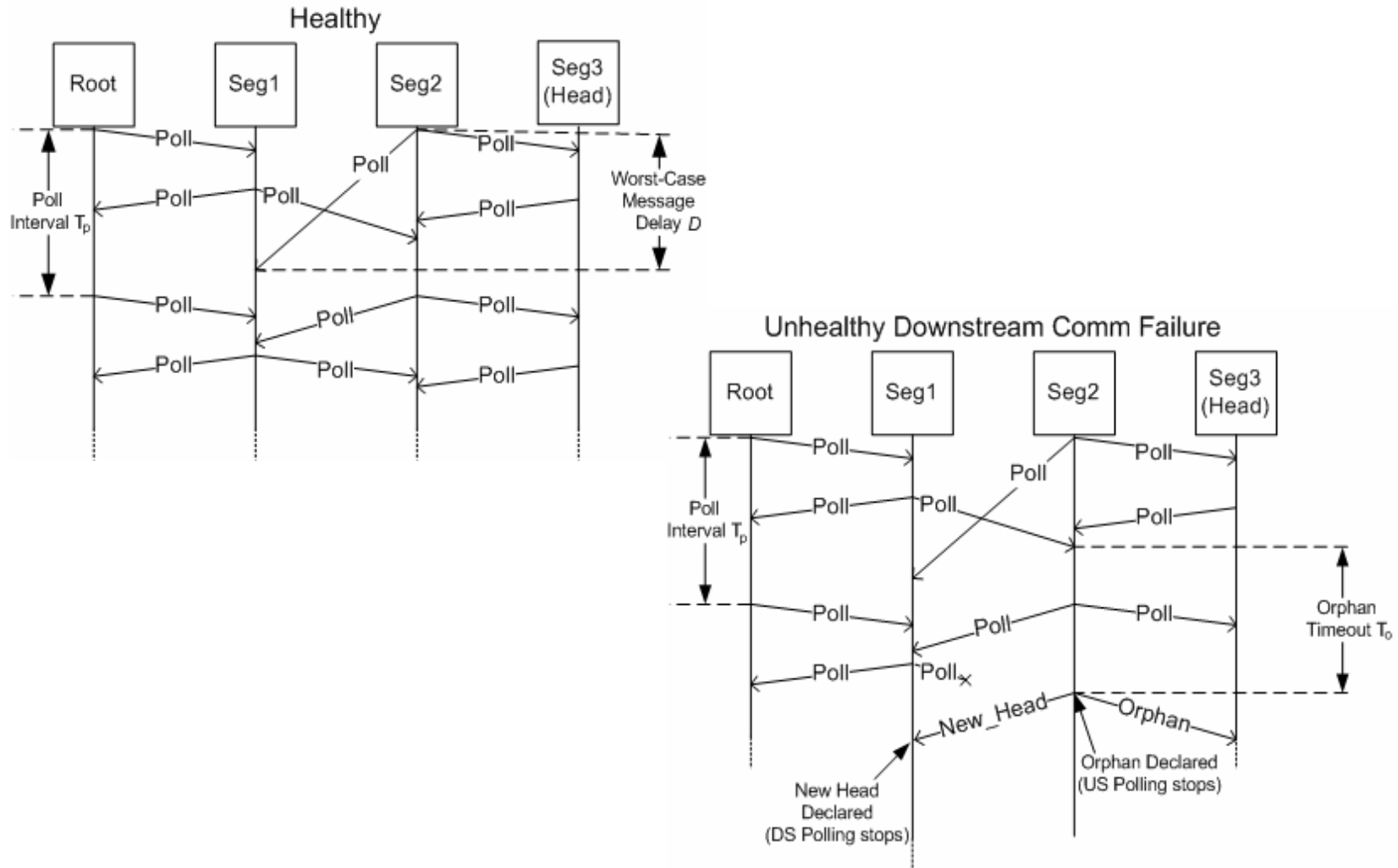


# Questions and Demonstration/Movie

Slides available at  
*<http://andersoj.org/srds06.ppt>*

# Distributable Thread Integrity

## D-TPR: Behavior Diagram



# Distributed Thread Integrity

## W-TPR: Protocol

- W-TPR is a distributed, real-time algorithm
- No active polling is maintained between adjacent nodes
  - Active failure detection during head movement
  - Application time-constraint driven recovery elsewhere
- Recovering from a break
  - All segments downstream of the break marked orphans
  - The segment directly upstream of the break becomes the new head of the DT, unless it is already marked as an orphan (multiple breaks)
  - Orphans are cleaned up in Last-In-First-Out order (head to root)
  - Control is passed to the application to decide how to continue