

AWINN Task 2.2 Demonstration:

Coastal Air Defense Scenario



Binoy Ravindran

binoy@vt.edu



Jonathan Anderson

andersoj@vt.edu

Task 2.2 Demo Agenda Recap

- Task overview
 - Binoy Ravindran
- Demonstration
 - Jonathan Anderson
- Wrap-up
 - Jonathan Anderson/Binoy Ravindran

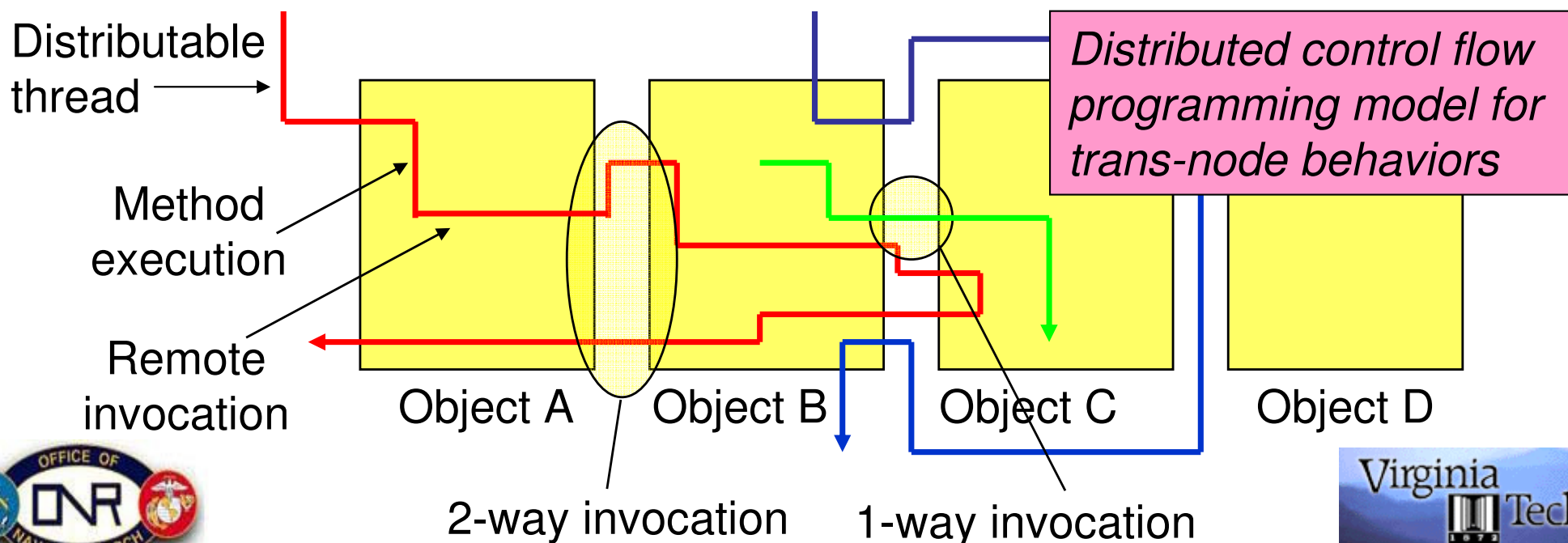


Task 2.2 Objectives

- Develop adaptive RT algorithms
- Complete Sun's Distributed RT Java standard (DRTSJ)
- Develop algorithms for scheduling and synchronizing distributable threads (as in RT-CORBA 1.2) ...

...with assured
timeliness...

in the presence of overloads, node/link
failures, MANET-induced message losses





DRTSJ Status

- DRTSJ is the key systems artifact of Task 2.2 research
- JSR-50 Specification Effort:
 - Spec Lead: E. Douglas Jensen (MITRE)
 - Implementation Team:
Jonathan Anderson (VT, Lead), Doug Wells (former Open Group), Ray Clark (MITRE), Sunil Srivastava (Apogee)
 - Expert Group Representation: AFRL, OIS, Lockheed Martin, Boeing, Washington University, Nokia, etc...
 - Reference Implementation on Apogee's Aphelion
- Planned submission of Spec and Reference Implementation
- Applications and Pluggable Protocols provided by Virginia Tech
 - Scheduling policies: Chewoo Na
 - TMAR policies: Ed Curley
 - Application: Joseph Hickman, Michael Nachmias

Work presented in demo today





DRTSJ Recent Events

- Keynote briefing to the Open Architecture Java Real-Time Workshop, NAVSEA/Dahlgren (1 August)
- Real-Time Java (RTSJ) is a critical component in developmental DoD (and USN) systems
- DRTSJ represented alongside key vendors and stakeholders in this area:
 - AICAS, Aonix, PrismTech, IBM
 - Lockheed-Martin (Aegis), Raytheon (DDG1000), NSWCDD
- Significant feedback on Navy end-user needs from front line developers

More Details at:
<http://drtsj.org>



Why Task 2.2 algorithms matter?

- Distributable threads are a natural way for expressing concurrent sequential activities in the physical world
 - Many distributed RT applications (e.g., NCW) contain such activities
- Many such activities operate under run-time uncertainties
 - Application- and MANET-induced uncertainties
 - Overloads, node failures, (transient/permanent) link failures, message losses
 - Deliberate disconnects/reconnect operations due to participant arrival, change of state, etc...
- Real-time algorithms (e.g., scheduling, synchronization) that can provide end-to-end timing assurances for distributable threads, despite those uncertainties, is an open research problem space
 - Traditionally, ad-hoc “black magic” approach
 - Task 2.2’s research contribution



Task 2.2 Research Contributions

Focus
of
demo

- Class of probabilistically reliable, distributed real-time *scheduling algorithms* (K. Han, S. Fahmy, J. Anderson)
 - Computable probability for satisfying thread time constraints
- Class of *thread integrity protocols* with bounded times for recovering from thread failures (J. Anderson, E. Curley)
 - Bounded time from failure detection till new head notification
- Class of TUF *scheduling algorithms and synchronization mechanisms* for single and multiprocessors (H. Cho)
 - Optimal multiprocessor scheduling synchronization
 - Tradeoffs between different synchronization mechanisms



Composing NCW Systems

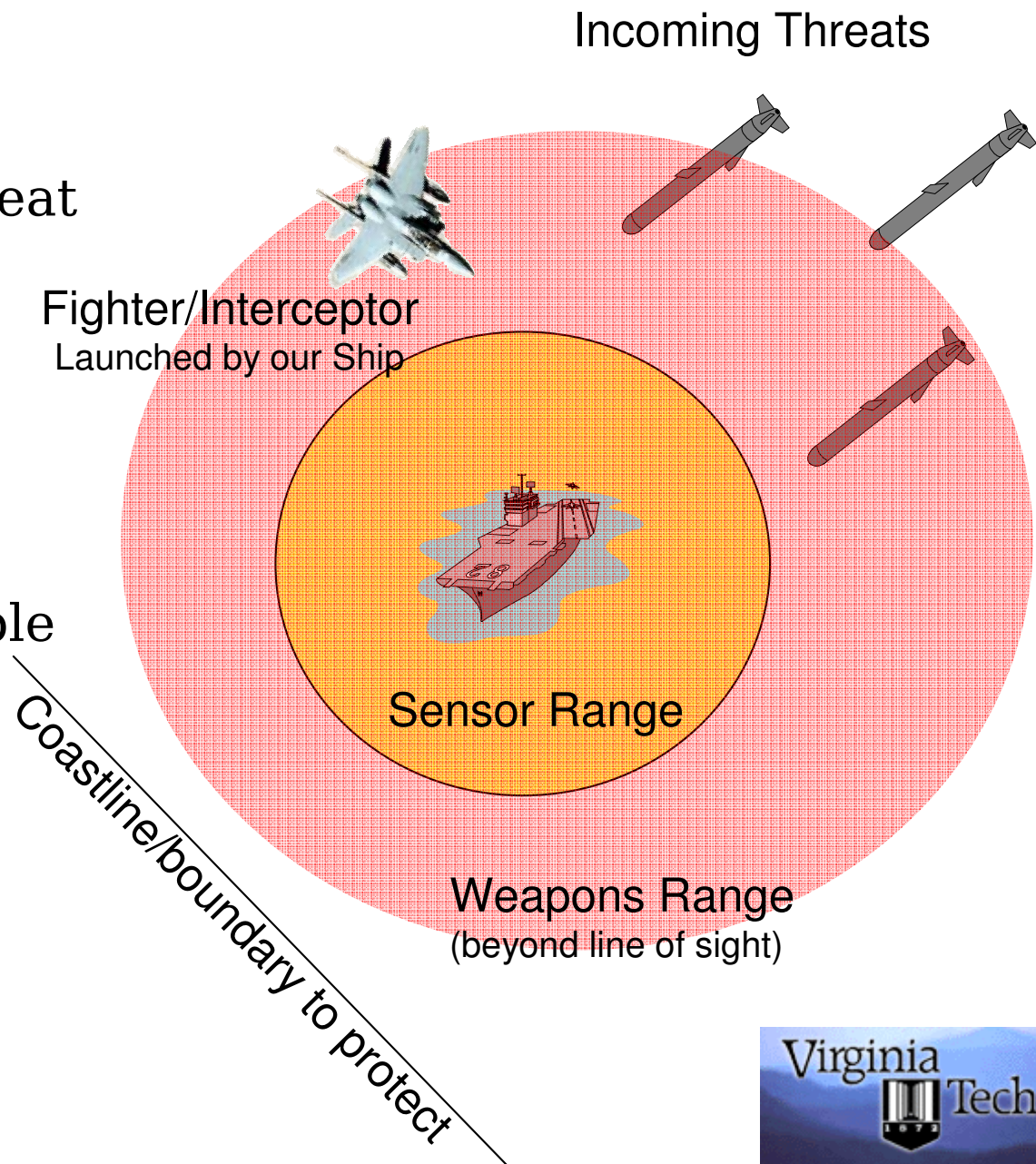
- Network Centric Warfare (NCW) in MANET environments presents new, fundamental challenges...
 - Partial Failures (communications and node) are the common case, not exceptional
 - Communication behaviors may be unpredictable (latency, bandwidth)
 - Yet missions require guarantees about timeliness and ordering of activities
- Today's demo presents real-time algorithms and mechanisms for detecting and recovering from communication failures due to MANET dynamics



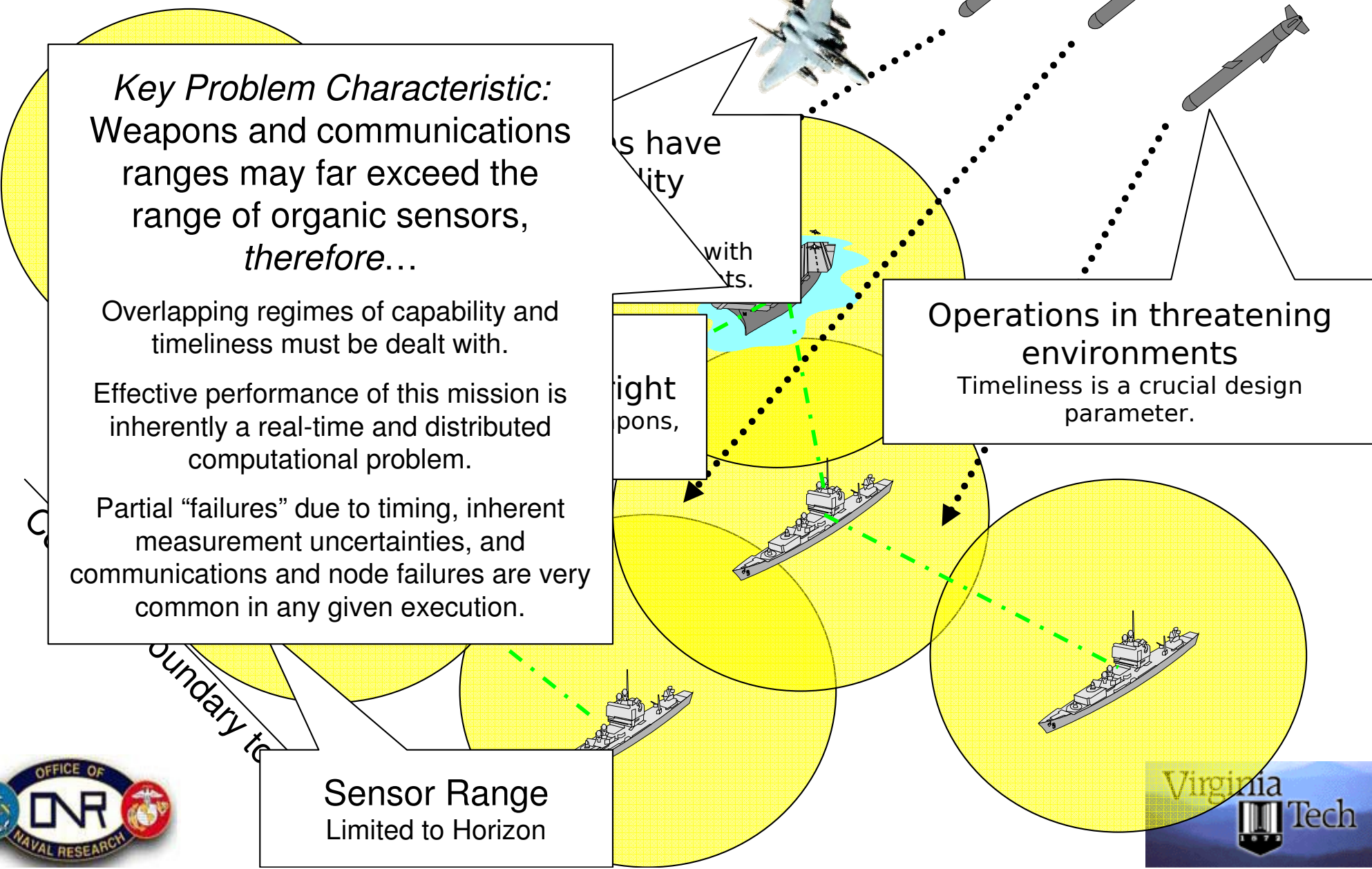
Our Motivating Scenario:

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



Key Problem Characteristic:
Weapons and communications ranges may far exceed the range of organic sensors, *therefore...*

Overlapping regimes of capability and timeliness must be dealt with.

Effective performance of this mission is inherently a real-time and distributed computational problem.

Partial “failures” due to timing, inherent measurement uncertainties, and communications and node failures are very common in any given execution.

s have
lity

ight
ons,

Operations in threatening environments
Timeliness is a crucial design parameter.

Sensor Range
Limited to Horizon



Our Motivating Scenario: Air and Coastal Defense

Key Problem Characteristic:
Weapons and communications
ranges may far exceed the
range of organic sensors.

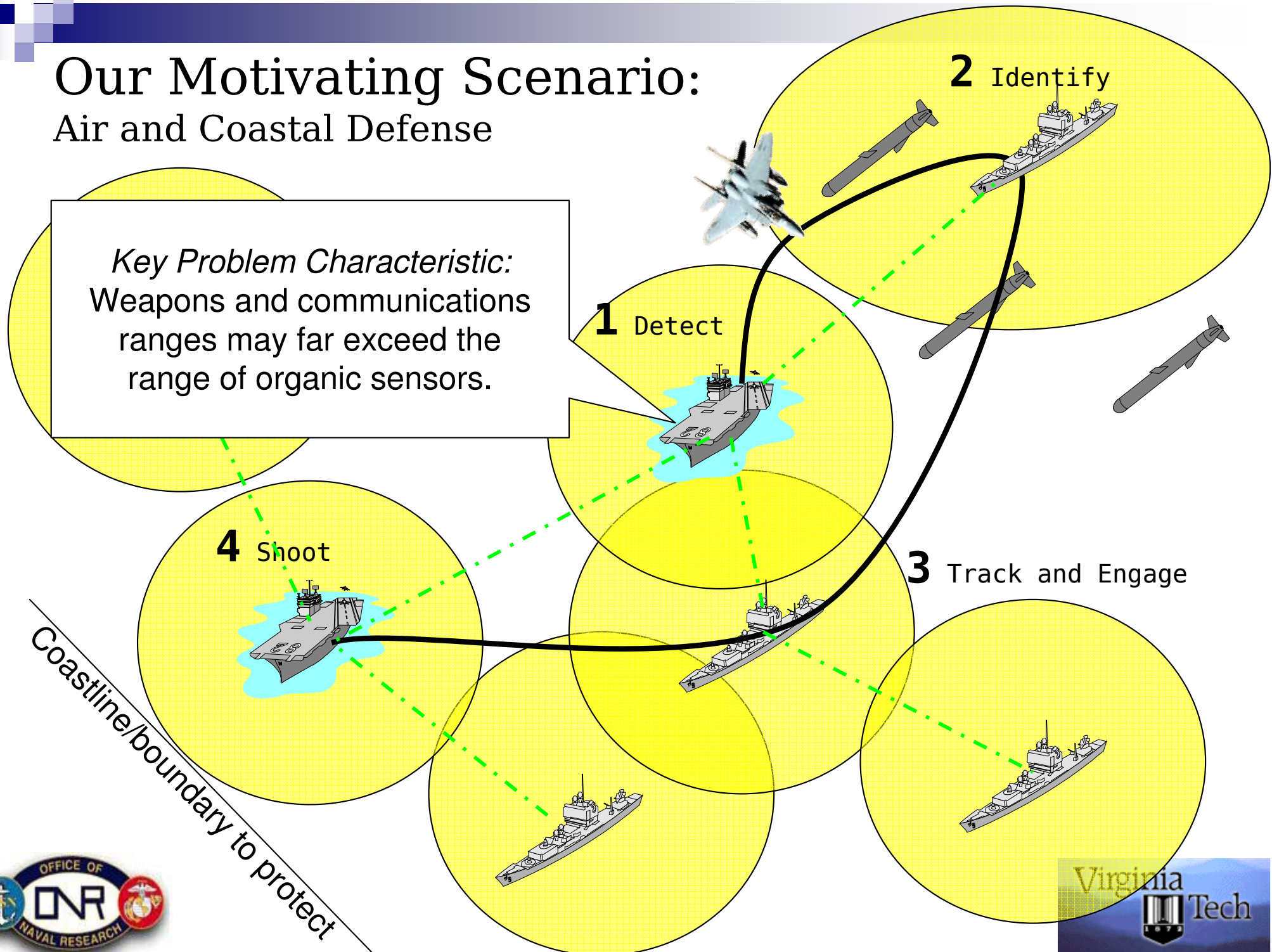
1 Detect

2 Identify

3 Track and Engage

4 Shoot

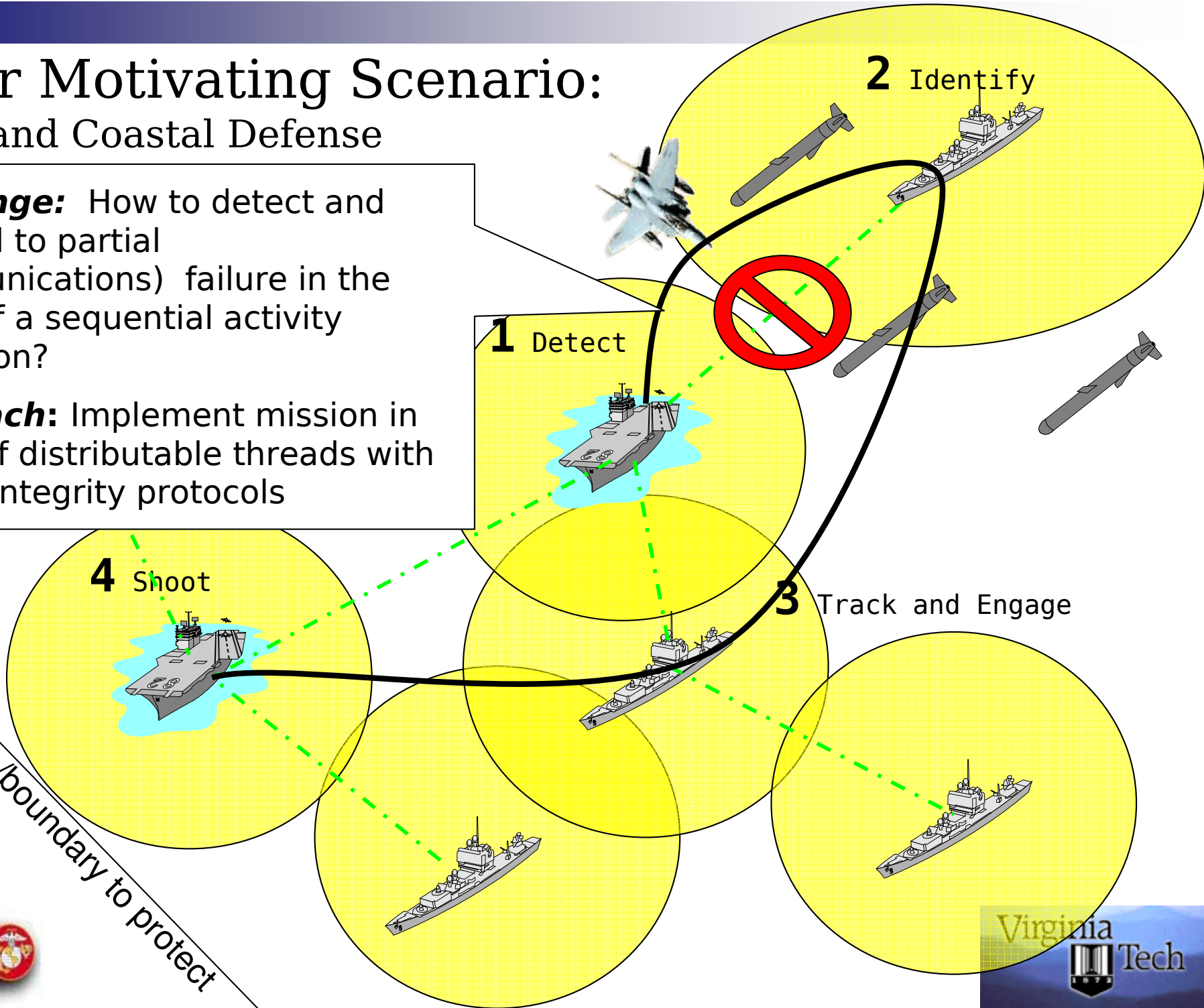
Coastline/boundary to protect



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



ScenarioGen Tactical Display (Example)

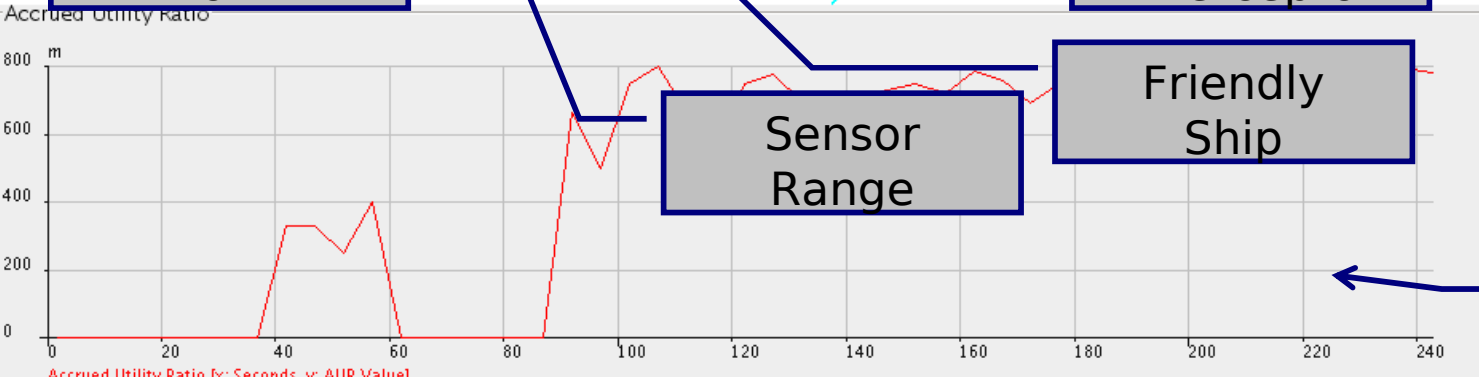
Legend

- Green --- Line
- Yellow --- Att
- Red --- Fri
- Friendly 25.5x33.8
- Enemy 25.5x33.8

Coastline to Defend

Goal Line: 5.0

10.0



Enemy Track (with status and velocity vector)

Comm Link Status

Intercept Point

Friendly Interceptor

Sensor Range

Friendly Ship

Control Panel

Scenario Control

Start Scenario	Stop Scenario
Pause Scenario	Unpause Scenario
Reset Scenario	

NavyShip2-NavyShip4

NavyShip3-N

NavyShip4-N

On
 Auto
 Off

NavyShip2-NavyShip9

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

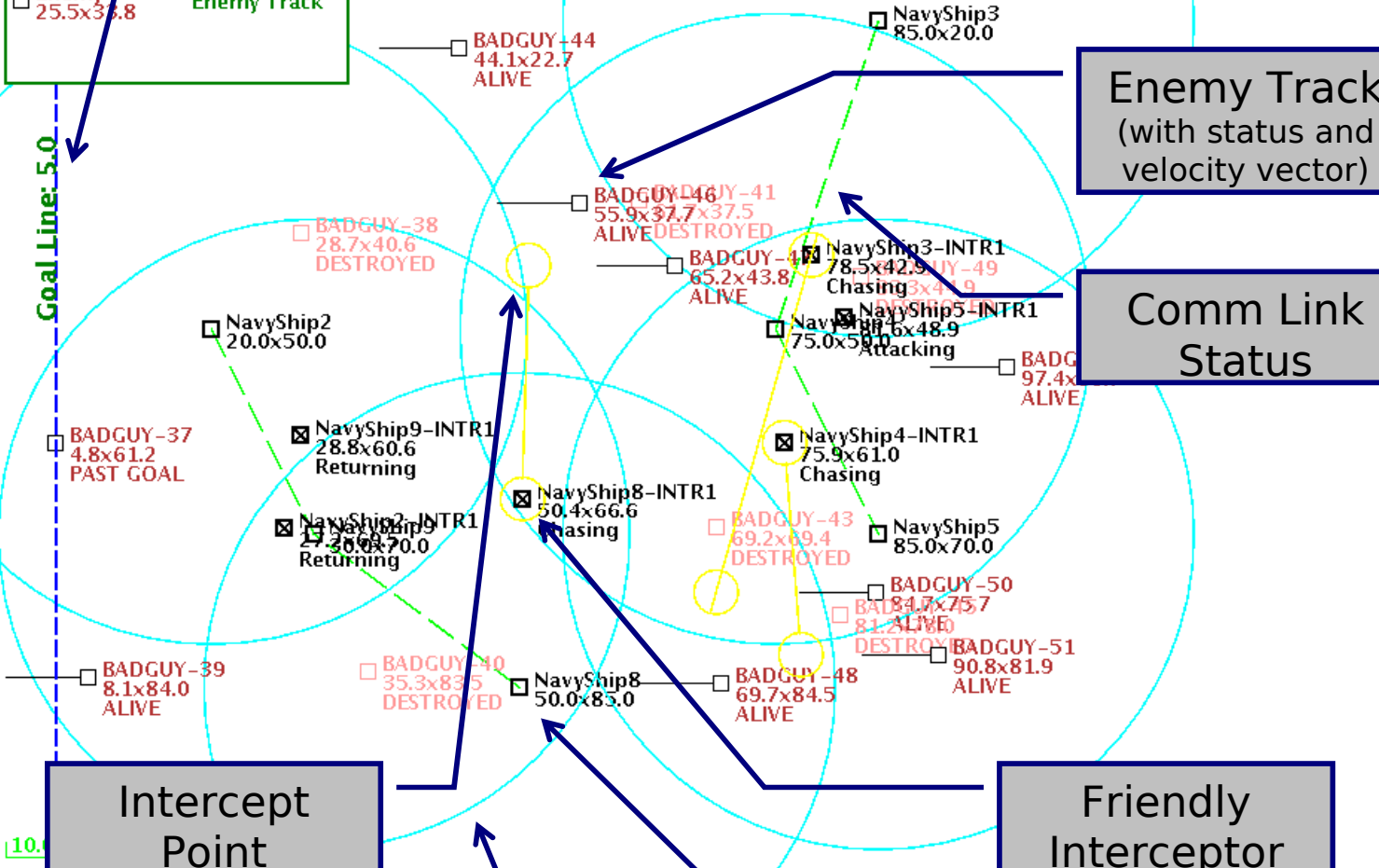
Status: **RUNNING**

Mission Time: 00:03:57.446

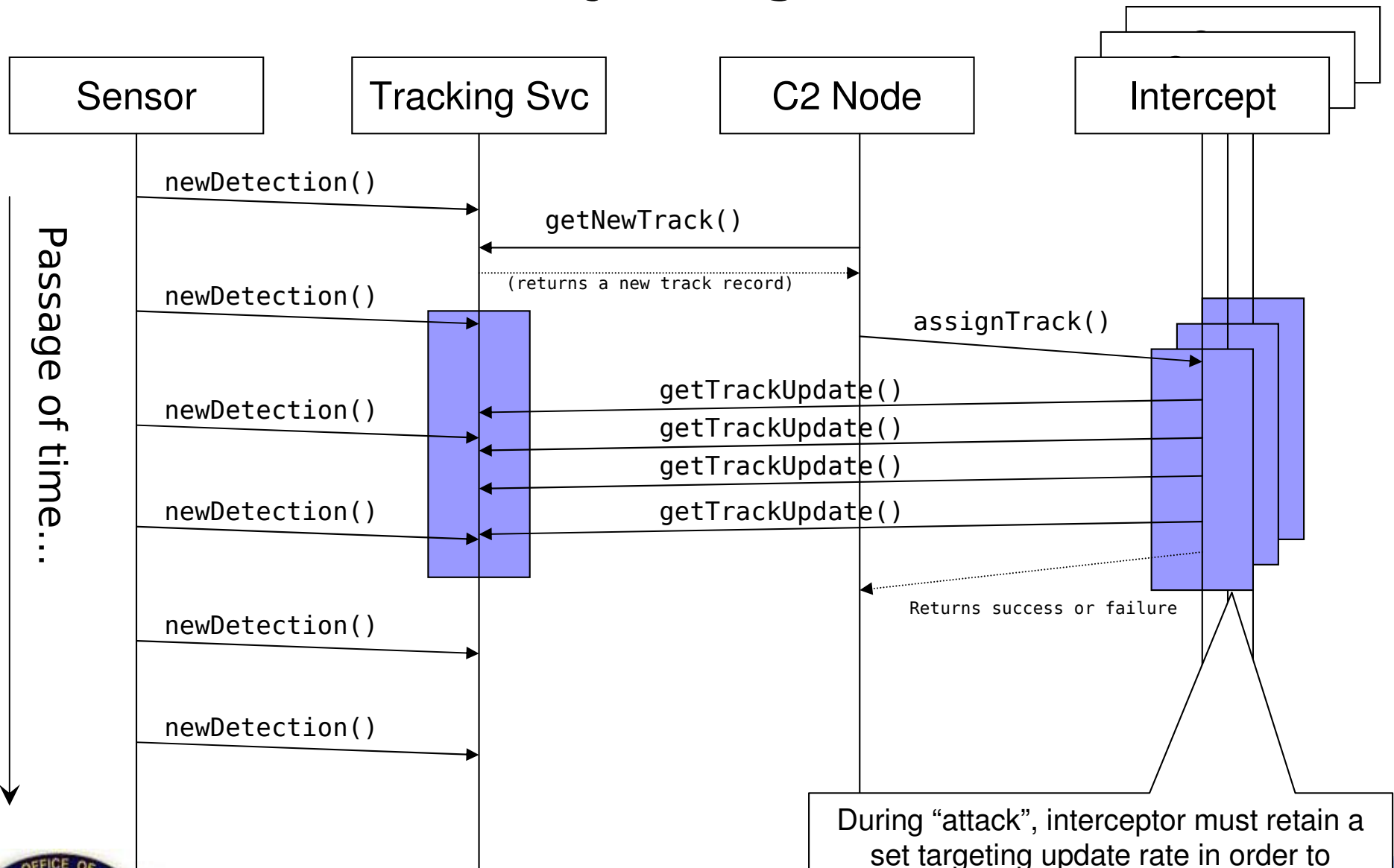
NavyShip2	UNKNOWN	Inactive
NavyShip3	UNKNOWN	Inactive
NavyShip4	UNKNOWN	Inactive
NavyShip5	UNKNOWN	Inactive

Integrity Policy and Comm Settings

Accrued Utility



Scenario Activity Diagram



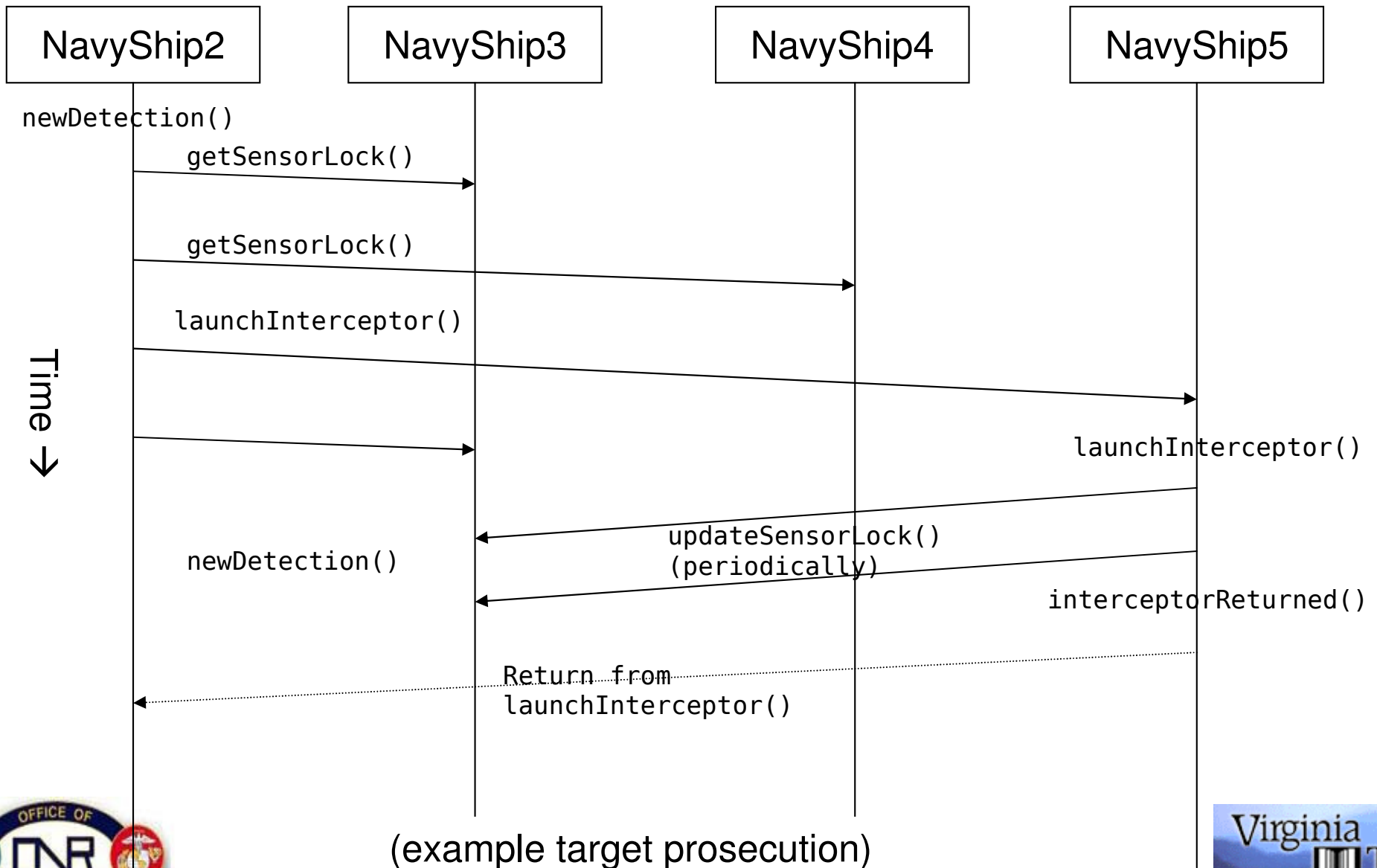
Passage of time...

(example target prosecution)

During "attack", interceptor must retain a set targeting update rate in order to achieve desired probability of kill.



Scenario Activity Diagram (MANET)



(example target prosecution)



Demonstration Features

- Three different integrity policies:
 - Baseline/Simple: (no integrity policy)
 - Fast-Failure-Detector: Aggressive, low-latency policy, ideal for LAN conditions [Trial-and-error tuning]
 - TMAR-W: Application timeliness driven policy for MANET [See “Recovering from ...”, Curley, Anderson, Ravindran, Jensen SRDS’06]
- Effectiveness Measure: AUR
 - Each threat (mission) is assigned a “value” of shooting it down which decreases as the threat nears the “goal line” (coastline)
 - **Accrued Utility Ratio** measures the ratio: earned / total available utility



Legend

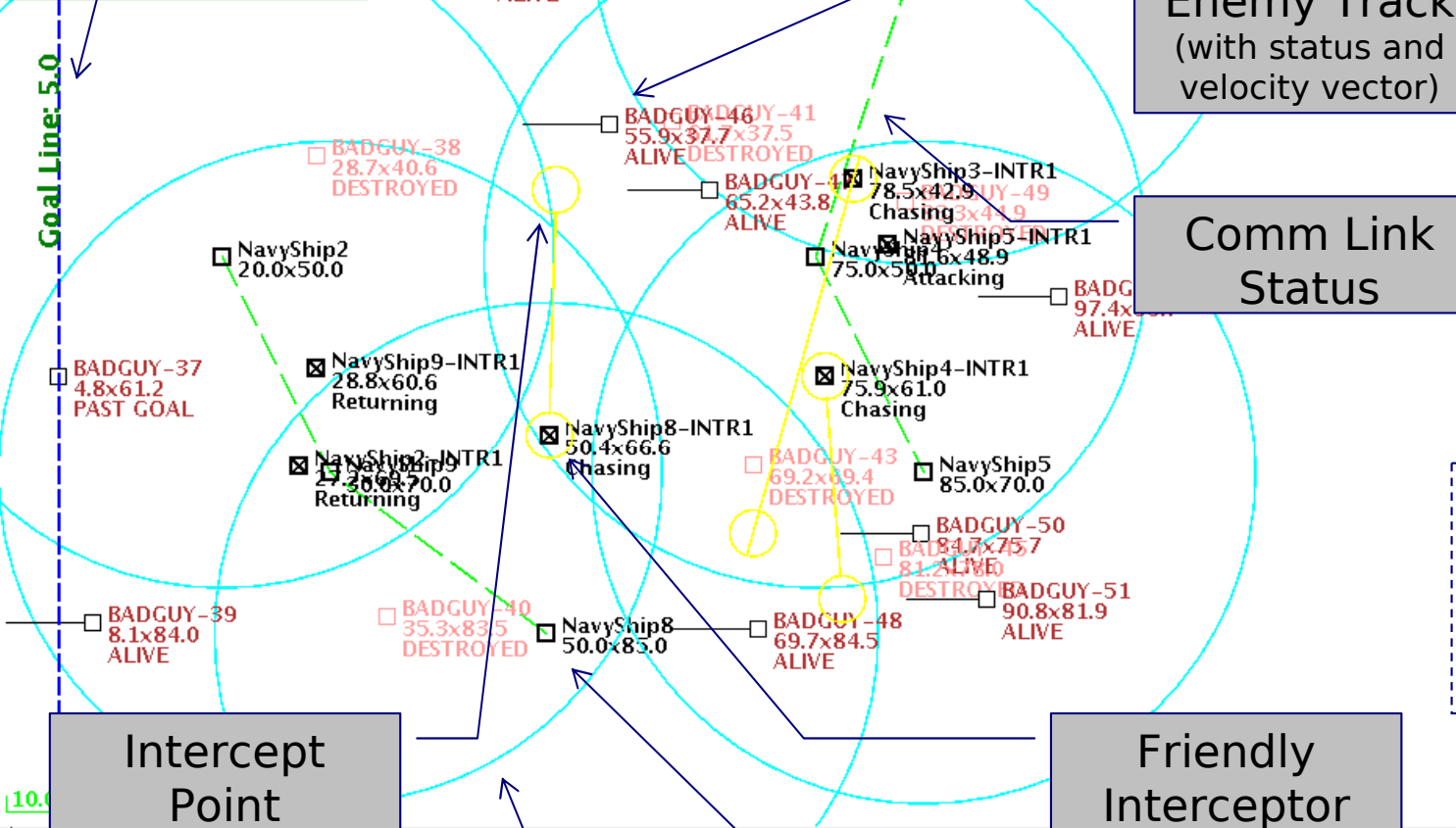
- Lin
- Att
- Fri
- Friendly 25.5x33.8
- Enemy 25.5x33.8
- Enemy Track

Coastline to Defend

Enemy Track (with status and velocity vector)

Comm Link Status

Integrity Policy and Comm Settings



Control Panel

Scenario Control

Start Scenario	Stop Scenario
Pause Scenario	Unpause Scenario
Reset Scenario	

NavyShip2-NavyShip4

NavyShip3-N

NavyShip4-N

On Auto Off

NavyShip2-NavyShip9

On Auto Off

DRTSJ Thread Integrity Mode

Simple Timeout-Driven TMAR-W

Communications Mode

Static Comms Dynamic Comms

Statistics

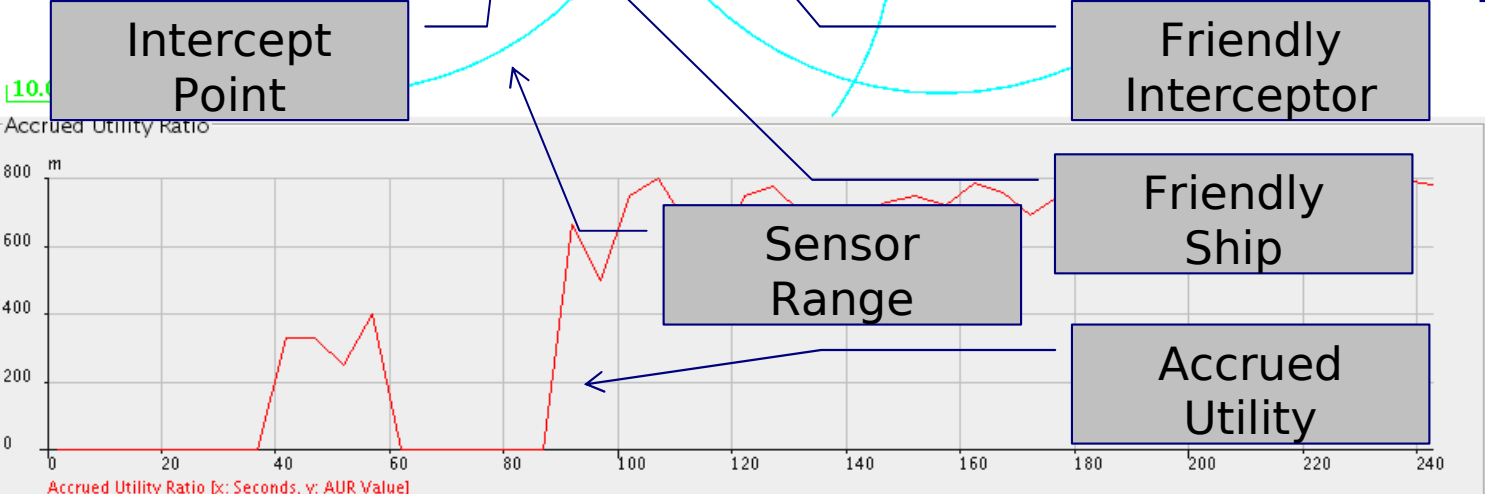
Points:	-180
Accrued Utility Ratio	0.782608695652174
Targets Destroyed	36
Past Goal Line	3

Scenario Status

Status: RUNNING

Mission Time: 00:03:57.446

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





Demonstration

Demonstration Conclusions

- A non-trivial C2 demonstration application...
 - DRTSJ Middleware
 - 4 application component types (~3k SLOC)
 - Implemented across six MANET sites
 - With 3 Integrity Policies
 - Under dynamic network conditions
- Existing programming methodologies are **brittle** in the face of MANET-specific challenges
- **Distributable threads** provide a **coherent**, timely, and robust programming model for DRT MANET systems
- Advanced MANET-aware **thread integrity mechanisms** yield measurable gains in **mission effectiveness**
- DRTSJ provides a programming environment with pluggable scheduling, distributable threads and integrity, built on commercially-available real-time Java platforms



Task 2.2 Accomplishments and Status

- DRTSJ nearing completion
 - Submission to DRTSJ Experts Group in September 2006
 - High visibility tech transition for Task 2.2 work
 - Coastal Air Defense Demonstration Application
- Several publications at highly selective (< 15-20% acceptance ratio) venues
 - 21 papers accepted (7 IEEE/ACM Transactions)
 - 8 under review (4 IEEE/ACM Transactions)
- DoD technology transition (e.g., DDG1000) is being actively sought
 - Demonstration at MITRE/McClean planned in Sept. 06





End

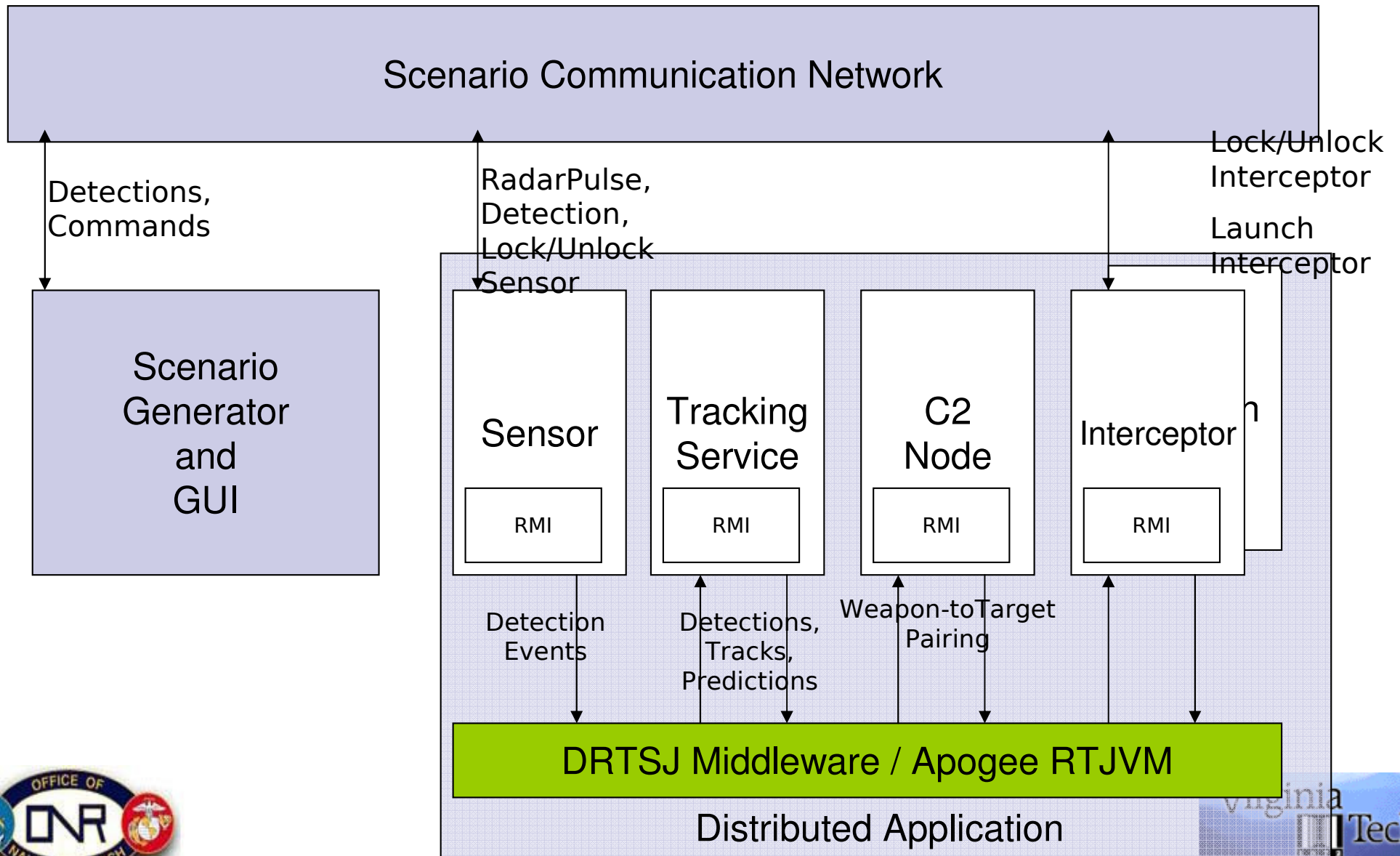
(backup slides follow)

Task 2.2 Demo Agenda

- Task status (B. Ravindran)
- Demo overview (J. Anderson)
- Demo
 - Led by J. Anderson
 - Contributors (2.2 personnel):
H. Cho, E. Curley, J. Hickman,
C. Na, M. Nachmias

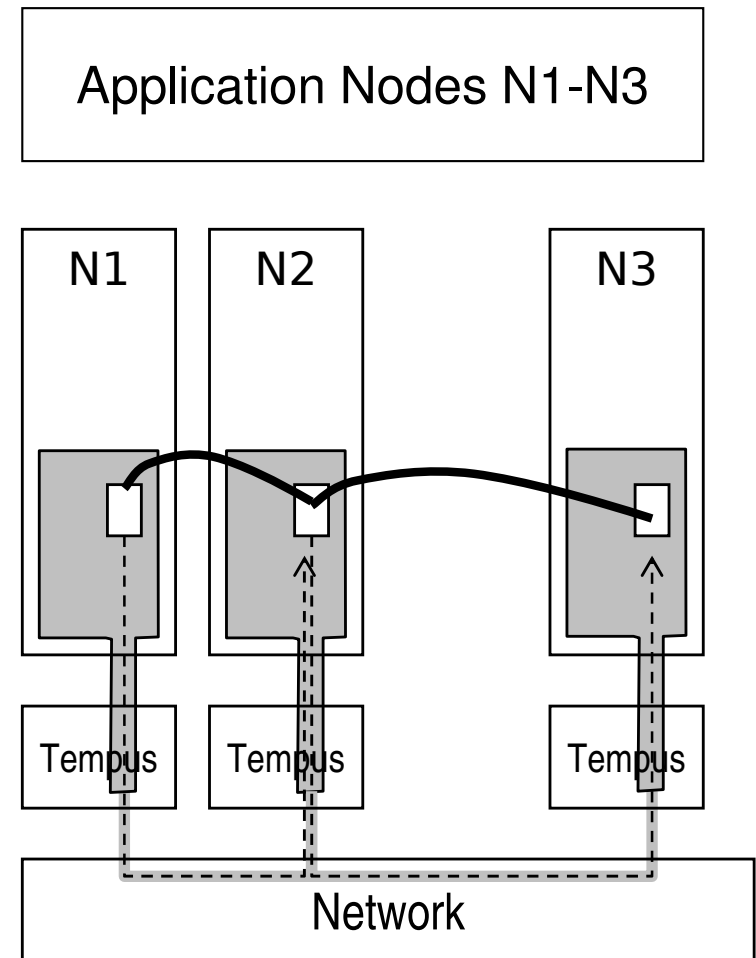


Scenario Deployment Diagram



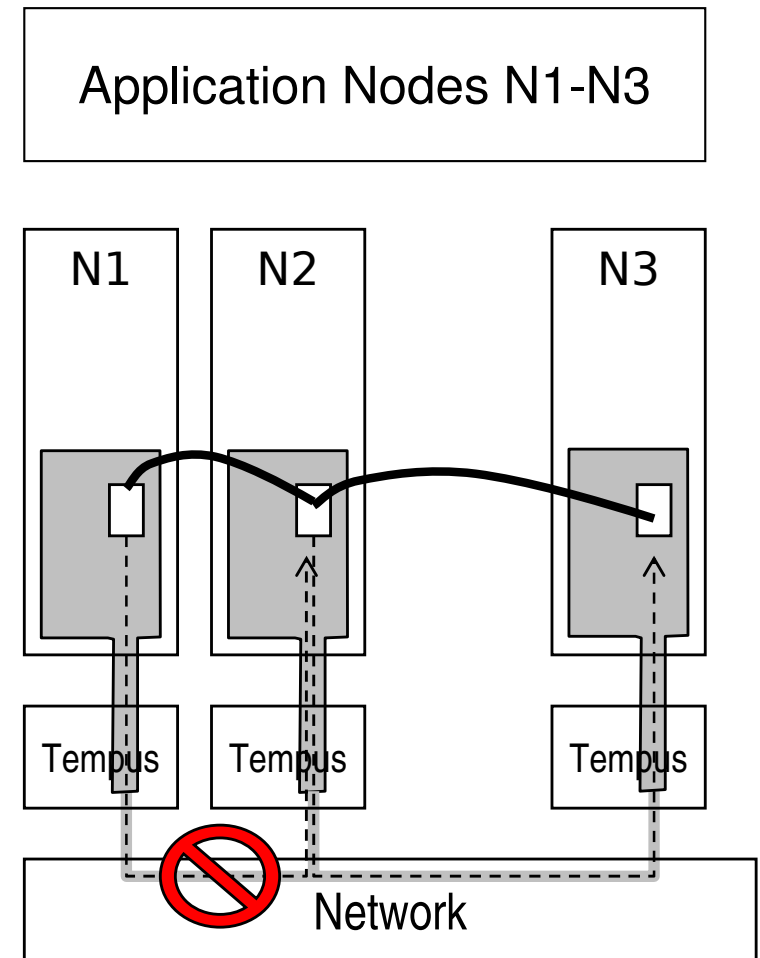
Distributed Threads

- Control-flow abstraction for Distributed Systems
- **Extends the traditional notion of an OS thread to allow method invocations between nodes (across a network)**
- Propagates execution, scheduling, and possibly other state with each invocation
- Enforces the *thread* property that there is a single point of control eligible for execution at any time
- Presents mechanisms to allow application-designers to implement failure strategies
- Provided by our Tempus middleware
- Powerful practical design abstraction because of familiarity with standard threads.



Distributed Thread Integrity

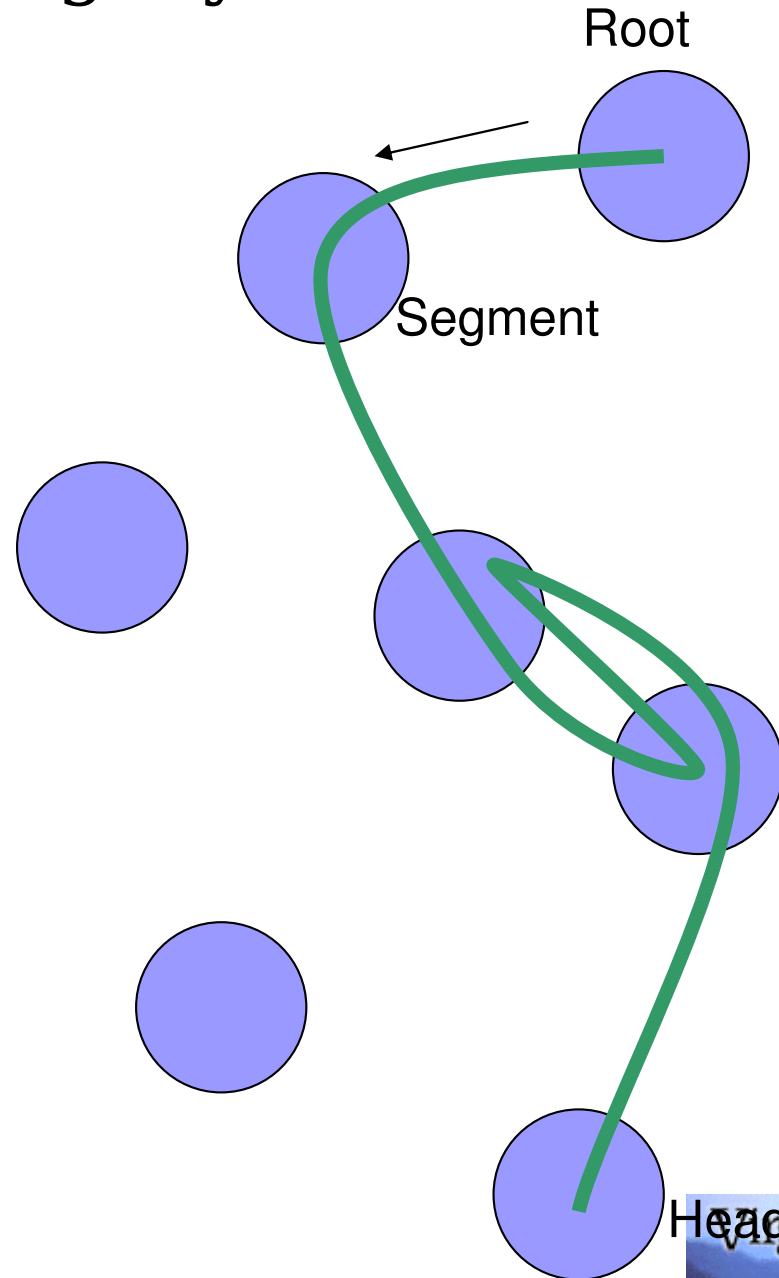
- Maintain thread properties...
 - Single head
 - Coherent call stack
 - Exception handling
- ...even in the presence of failures
- Algorithms
 - Thread-Polling – Poll/lease based scheme to detect thread breaks
 - Node-Alive -- Group-membership algorithm



Distributed Thread Integrity

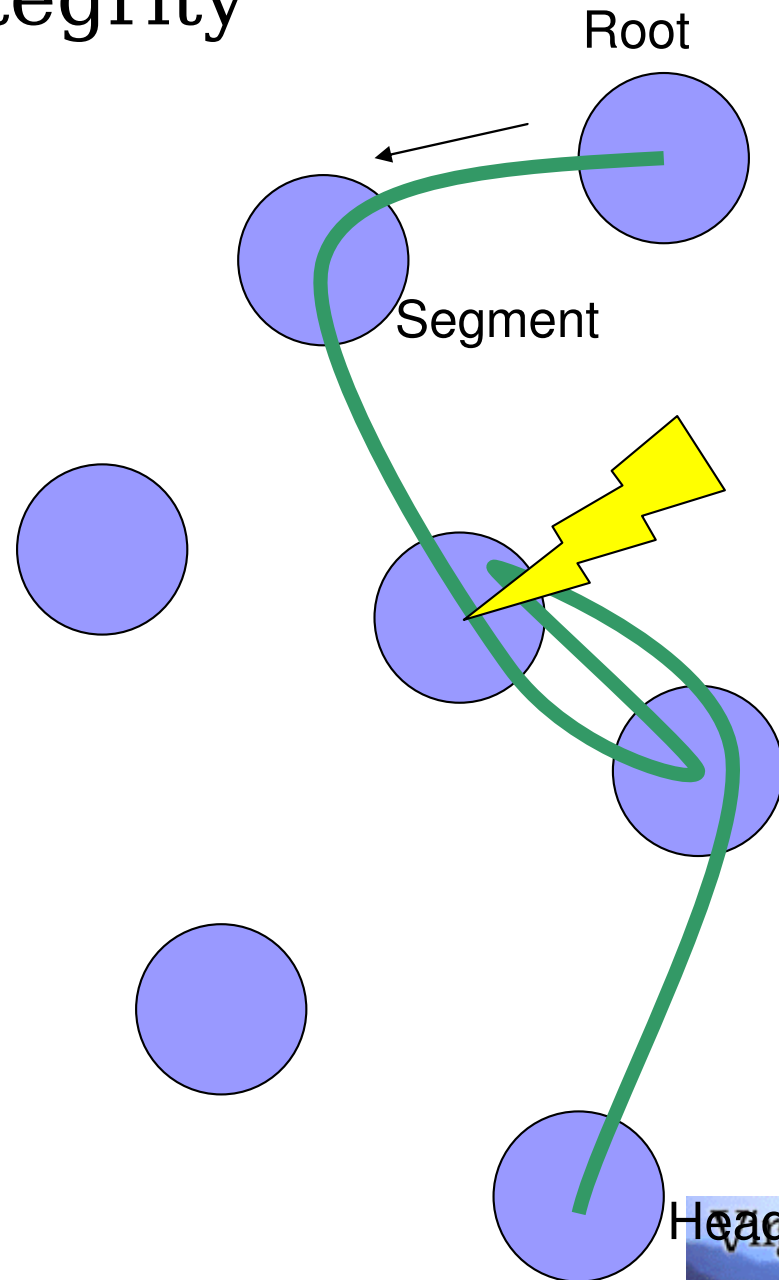
Thread Polling

- 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 “distributed thread” is realized by a collection of local threads, called “segments”



Distributed Thread Integrity

- 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 “distributed thread” is realized by a collection of local threads, called “segments”



History

- The TPW protocol is derived from the Thread Polling (TP) TMAR protocol present in the Alpha Operating System.
- TP was chosen due to its flexibility and relatively low overhead



Design Constraints

- TPW has to:
 - Operate without broadcast capability
 - Handle intermittent wireless communication errors
 - Provide modifiable parameters to deal with differing network conditions
 - Provide Last-In-First-Out cleanup of orphans
 - Provide Bounded End-to-End recovery time from Distributed Thread breaks



Protocol – Detecting a break

- TPW is a distributed algorithm
- Polling is maintained between two adjacent nodes that share a common Distributed Thread (DT)
- When poll-messages are not received after a certain time, the link between the two nodes is considered severed and the DT is considered to be broken

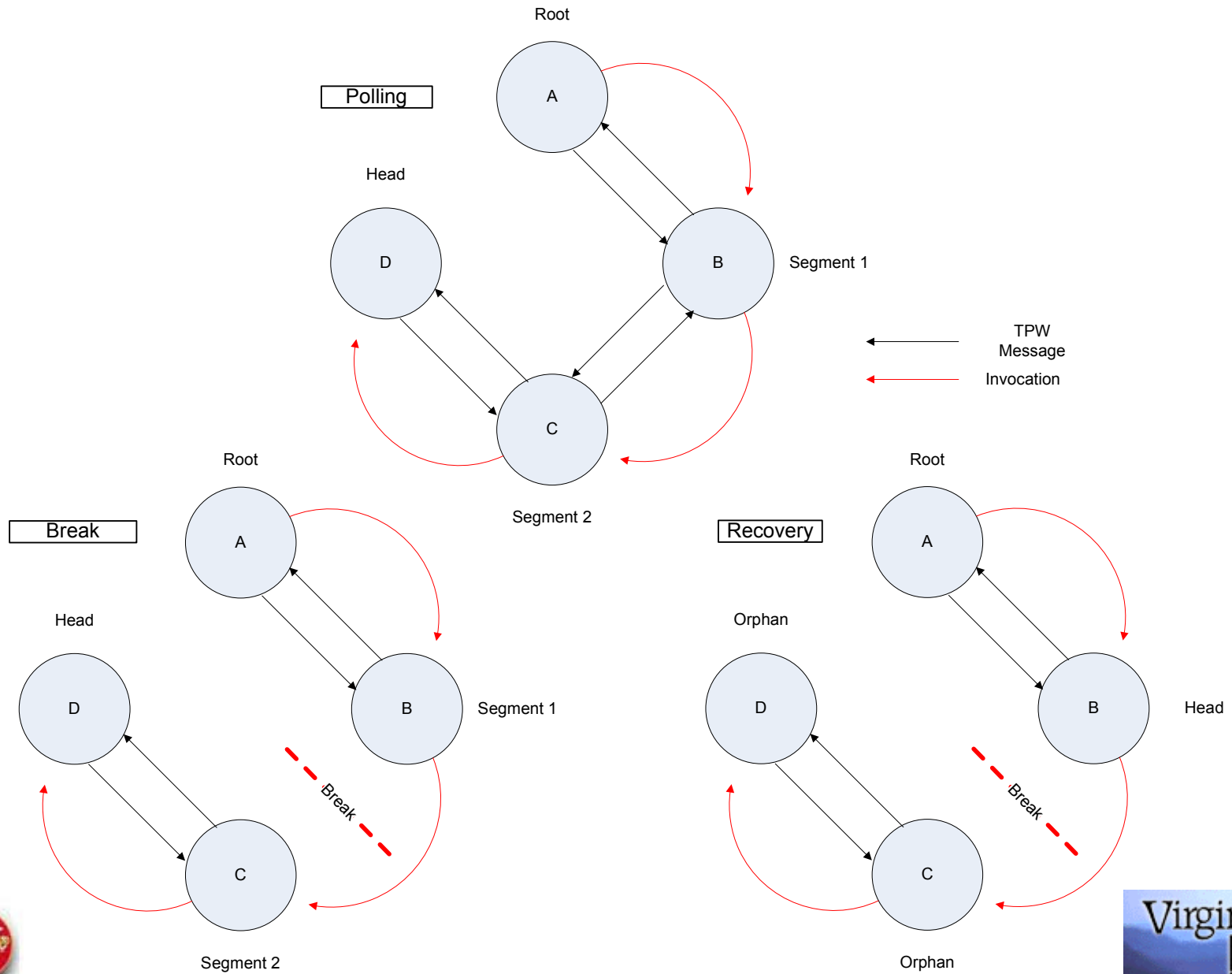


Protocol – Recovering from break

- All segments downstream of the break are 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 are cleaned up in Last-In-First-Out order (head to root)
- Control is passed to the application to decide how to continue



TPW Diagram



Next Steps

■ Short-term Goals

- Extending thread integrity algorithms in support of distributed, consensus-based scheduling
- Distributed, real-time garbage collection
- Advanced lock-free synchronization

■ Longer-term Goals

- Real-Time MANET routing approach
- Complete integrated test-bed and demonstration



Distributed Thread Integrity

Thread Polling

- **Phase I:** Each dthread's root node broadcasts a "refresh" message to all nodes
- **Phase II:** Every node hosting a segment of the thread responds 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.

