Providing Fault-Tolerant Resource Management in a CCM DRE Environment

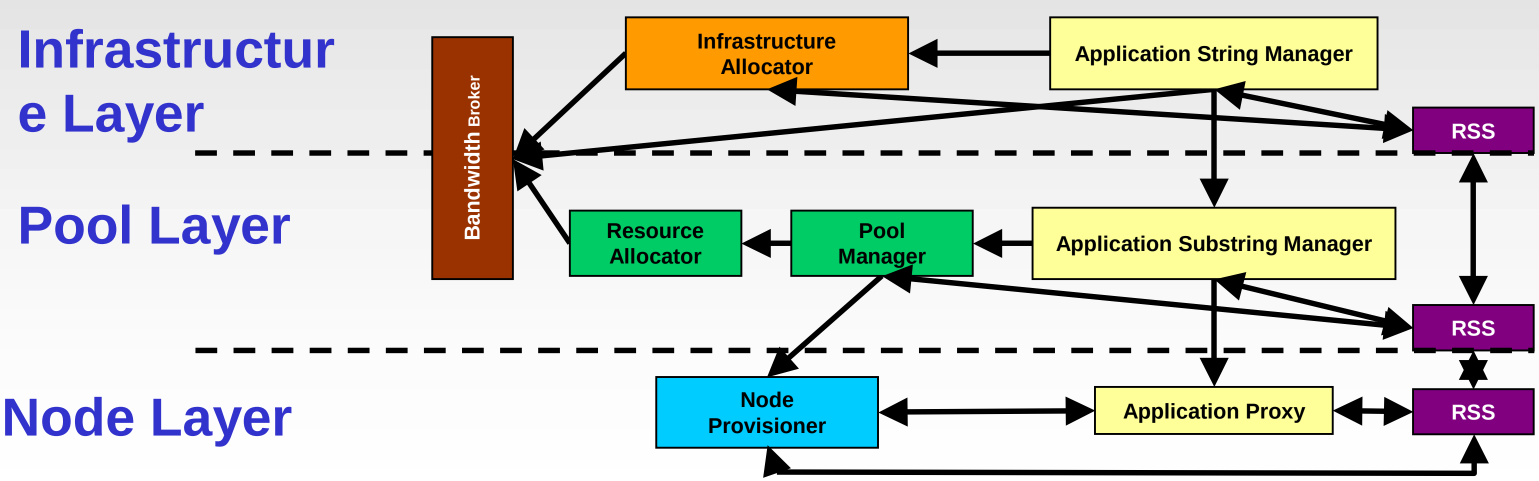
Paul Rubel, Joseph Loyall, Matthew Gillen
(prubel@bbn.com)
BBN Technologies
Aniruddha Gokhale, Jaiganesh Balasubramanian
Vanderbilt University
Priya Narasimhan, Aaron M Paulos
Carnegie Mellon University

Outline

- Integrating Fault-Tolerance into a Resource Management Subsystem for Multi-Layer DRE Systems
- Providing Rapid Recovery from Faults
- Supporting Component Middleware
- Efficiently Supporting Fault-Tolerance in Large-Scale Systems

Multi-Layered dynamic Resource Management (MLRM)

- Manages the allocation and reallocation of computation and network resources to critical sets of applications
- Survives failures of lower-layer elements
  - Pool and node failures are dealt with using redundancy
  - Infrastructure failures require replication



Challenges to building a fault-tolerant MLRM

- MLRM Recovery needed to be very fast so application recovery could begin
  - Support continuous operation
- Existing FT solutions made unreasonable assumptions
  - Only client-server interactions were supported
    - Tiered interactions were necessary
    - Peer-to-Peer interactions were also needed
  - The replication infrastructure permeated the entire system
    - Resources were wasted unnecessarily
- We wanted a general rather than custom solution

Outline

- Integrating Fault-Tolerance into a Resource Management Subsystem for Multi-Layer DRE Systems
- Providing Rapid Recovery from Faults
- Supporting Component Middleware
- Efficiently Supporting Fault-Tolerance in Large-Scale Systems

MLRM Recovery Needs to be *Fast!*

- When a failure occurs critical applications may need to be reallocated to active nodes to continue to provide service
- MLRM needs to recover before this can happen
- Constituent elements of recovery
  - Fault Detection
    - Speed is mechanism- and fault-dependent
  - Recovery from the failure
    - Fault model and fault-tolerance scheme dependent
      - Options are constrained by application characteristics OR
      - Applications are designed around required FT characteristics
    - MLRM applications were already mature
      - We worked within the application constraints using available techniques to make recovery as fast as possible

Fault Detection

- Process failures with active network connections are detected quickly via the OS
  - Closed socket notification is very fast
  - This is not the worst case
- Detecting network failures with TCP timeouts can take minutes

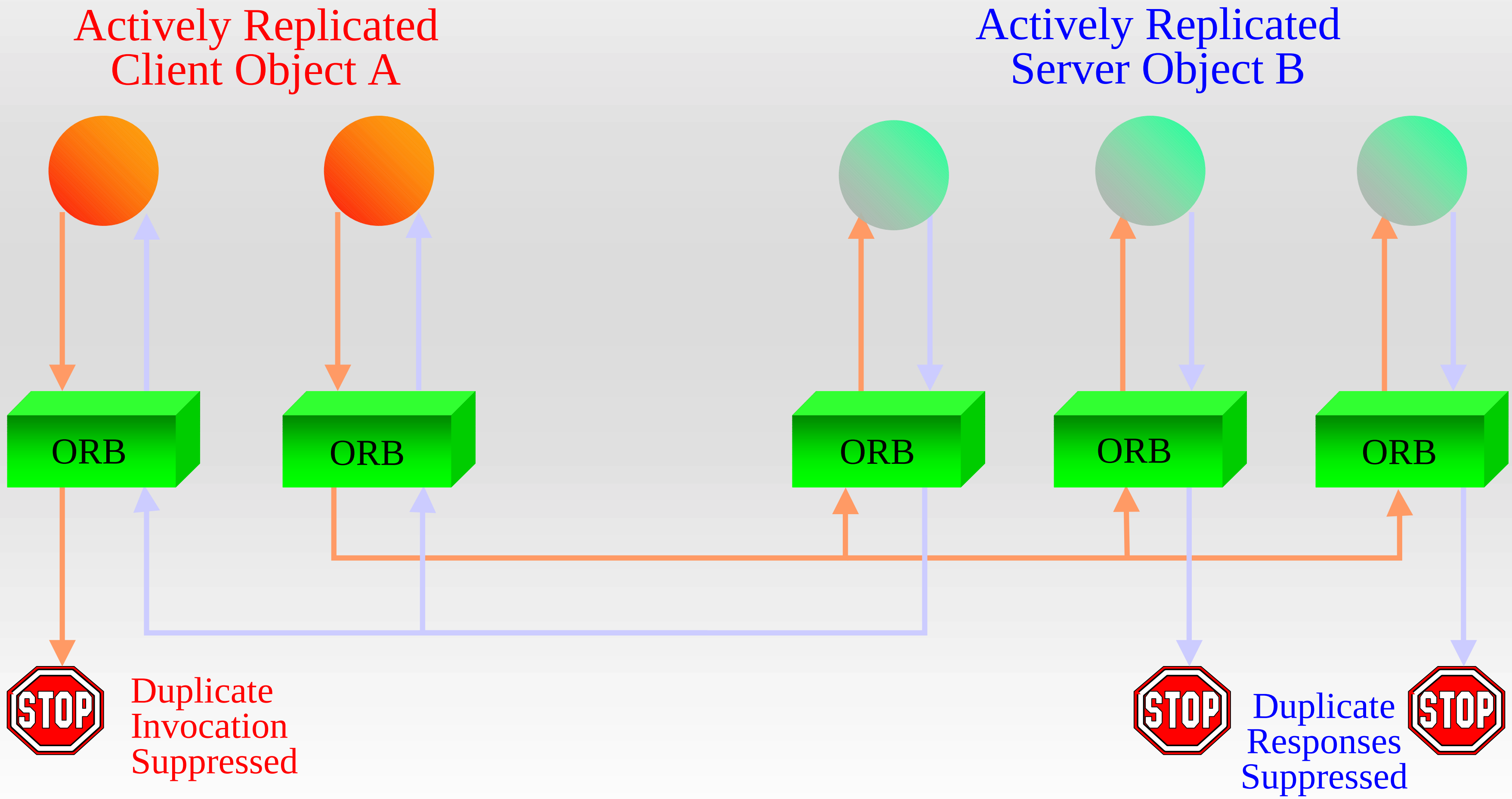
- Active probing is necessary
  - Group Communication System (GCS) provides membership services to detect failures via messages between active members, if you use GCS
  - Custom Node Failure Detection (NFD) logic can also be used for detection of failures of components not in a group or in large scale deployments

Fault Recovery and System Design

- In a distributed system, like MLRM, where pieces can individually fail, FT needs to be distributed
- Replication is well suited for this fault model
- Recovery requirements, application characteristics, and replication style are balanced in a deployed system
  - Active/State Machine provides fast recovery for deterministic apps
  - Passive/Primary-Backup supports non-determinism but depends on getting and setting state
    - Warm - online spare(s)
    - Cold – offline spare(s)
  - Semi-Active, a hybrid approach
  - Semi-Passive, another hybrid

Active Replication

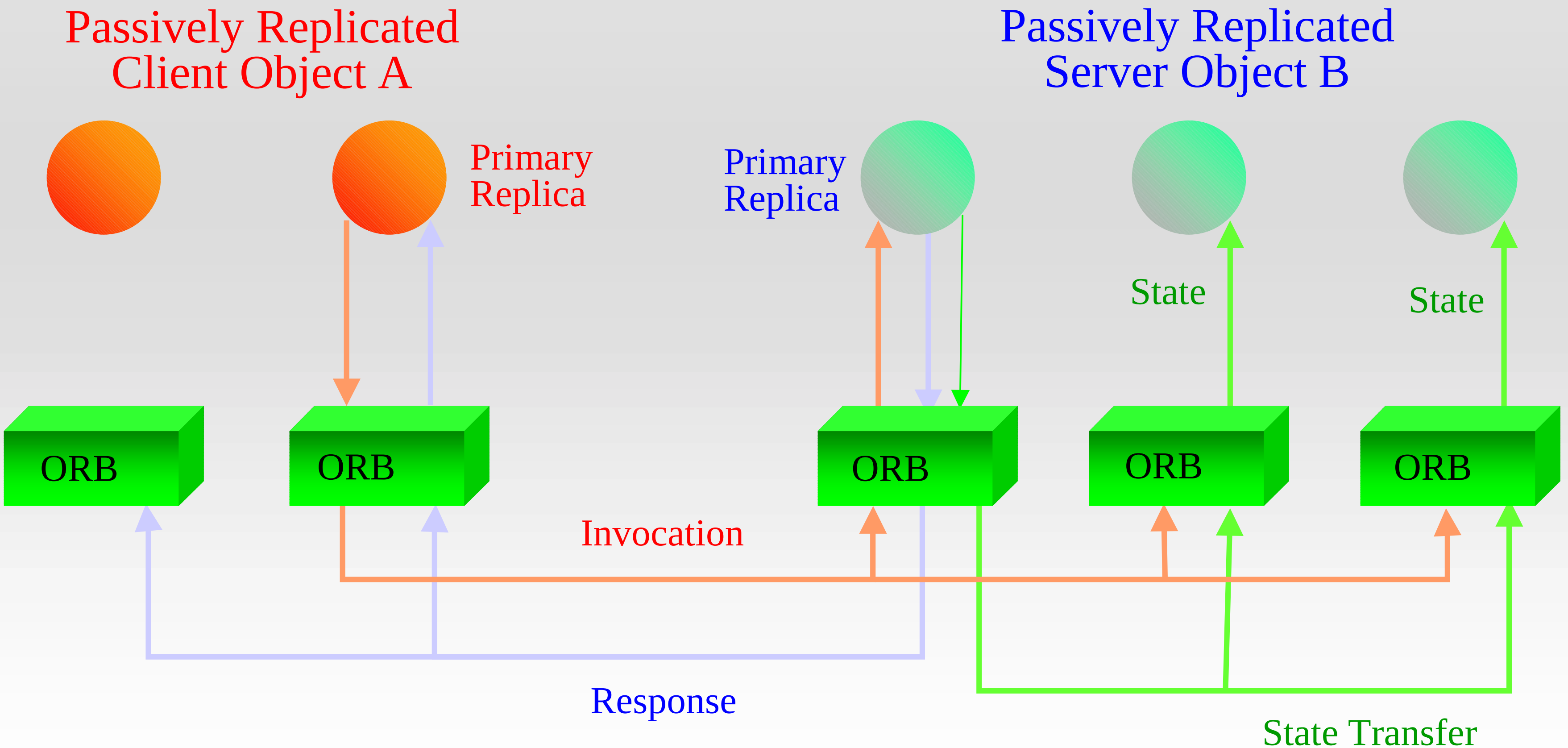
- Every copy receives and processes every message



Passive Replication

- Only one (primary) copy processes all the messages
- Other (backup) copies receive state updates from the primary

- *Cold Passive* – State is transferred only when a backup is promoted to primary
- *Warm Passive* – State is transferred periodically into designated backups



Integrating FT into the MLRM

- Active replication was possible for 1 of 3 MLRM applications due to:
  - The state-machine like nature of interactions with the application
- Passive replication was necessary for the other two components due to:
  - Timers, threading, interactions with network routers, non-determinism
- Active and passive replicas needed to work together to make the MLRM fault-tolerant

Outline

- Integrating Fault-Tolerance into a Resource Management Subsystem for Multi-Layer DRE Systems
- Providing Rapid Recovery from Faults
- Supporting Component Middleware
- Efficiently Supporting Fault-Tolerance in Large-Scale Systems

Changes to Support for Components and F-T

- Changes to the Component Engine (DAnCE)

1. Supporting state exchange for hosted components is now necessary in addition to previous work on application and ORB state

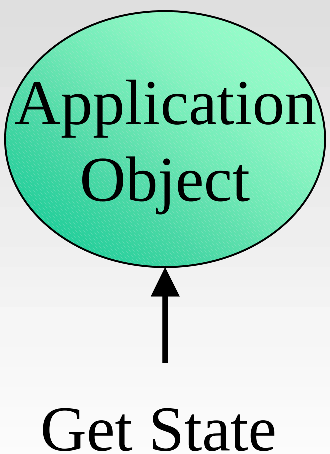
- Changes to the F-T Infrastructure

2. State transfer must wait until the component engine is ready rather than just the application and ORB
3. Support for CCM deployment
4. Support for Peer-to-Peer interactions

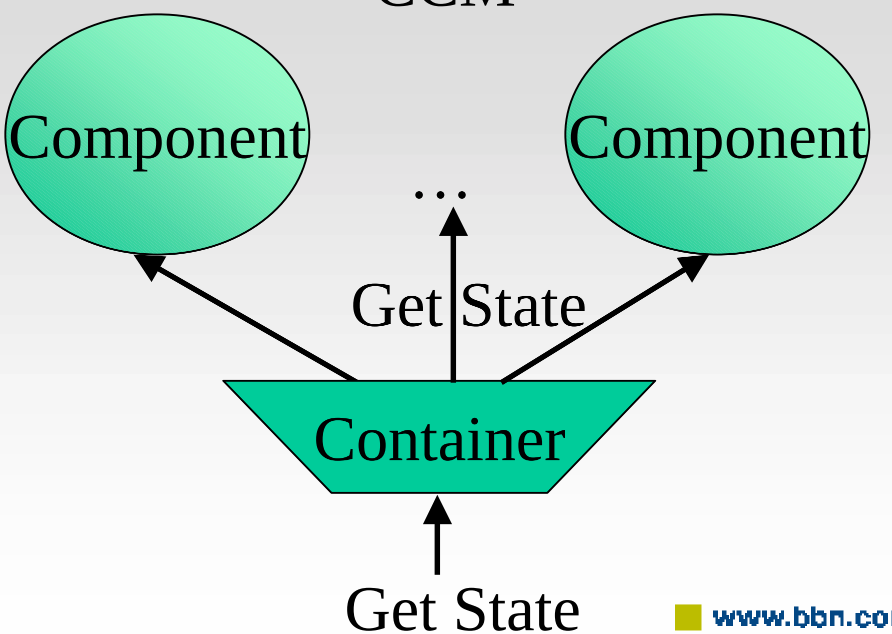
1 - Getting State

- When adding a new replica or using passive replication state is gathered from existing replicas
- CCM requires help from the container to gather state from all components in the process

CORBA 2



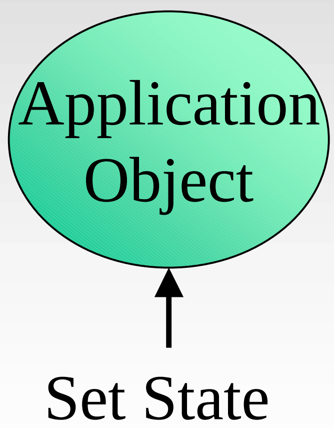
CCM



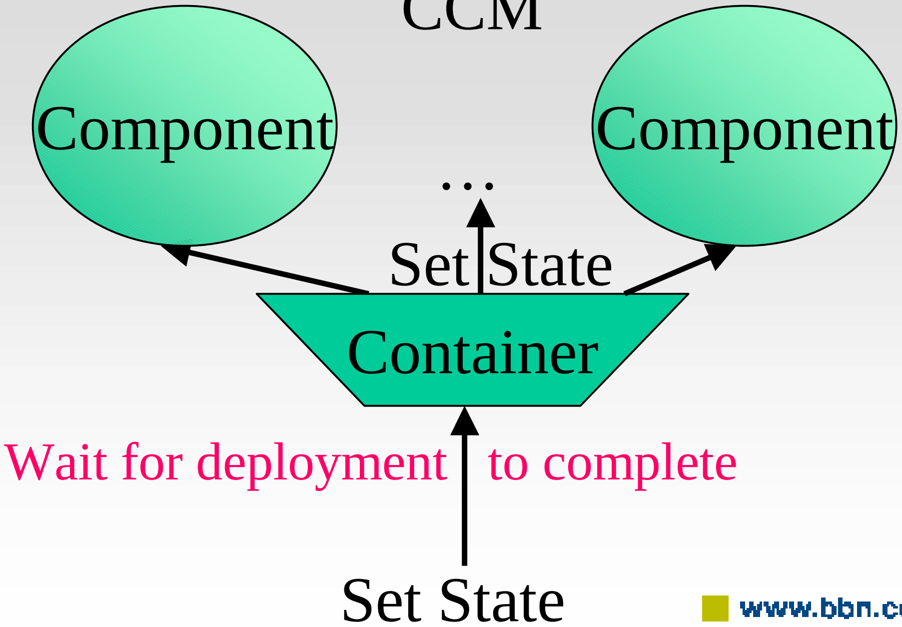
2 - Setting State

- When adding a new replica or using passive replication setting state is used to keep replicas consistent
- CCM requires help from the container to pass state to all components in the process
- We must also wait until deployment is complete so components exist when a call is made on them

CORBA 2

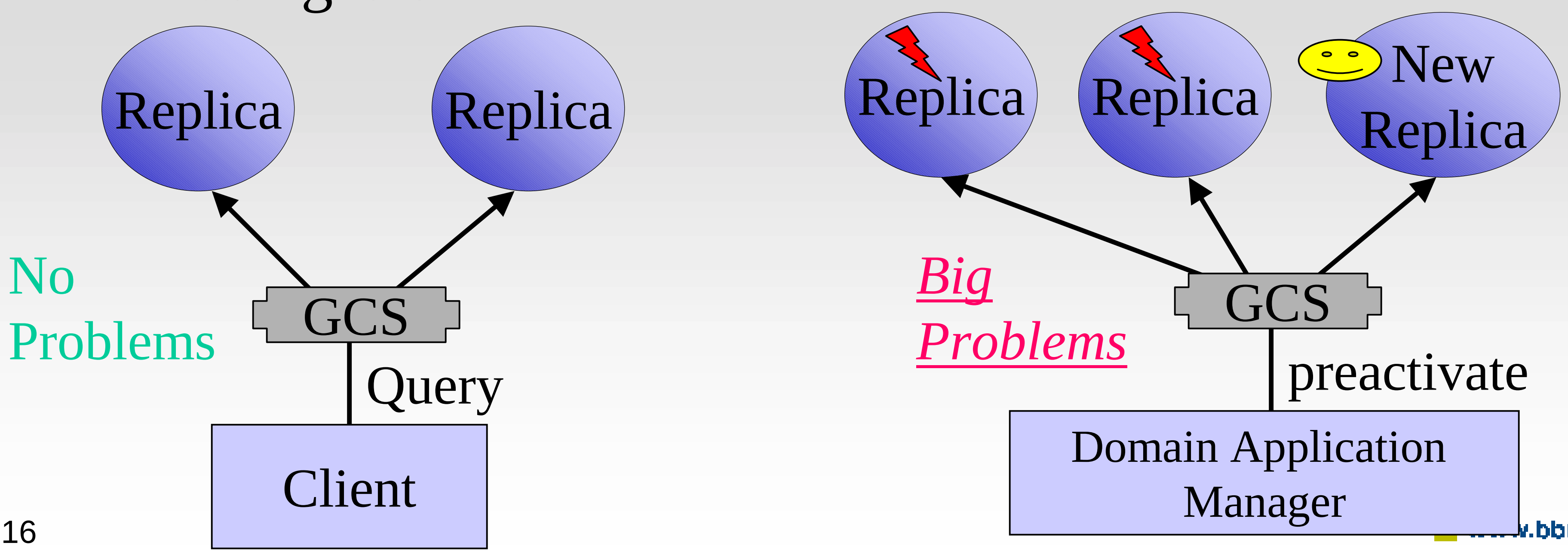


CCM



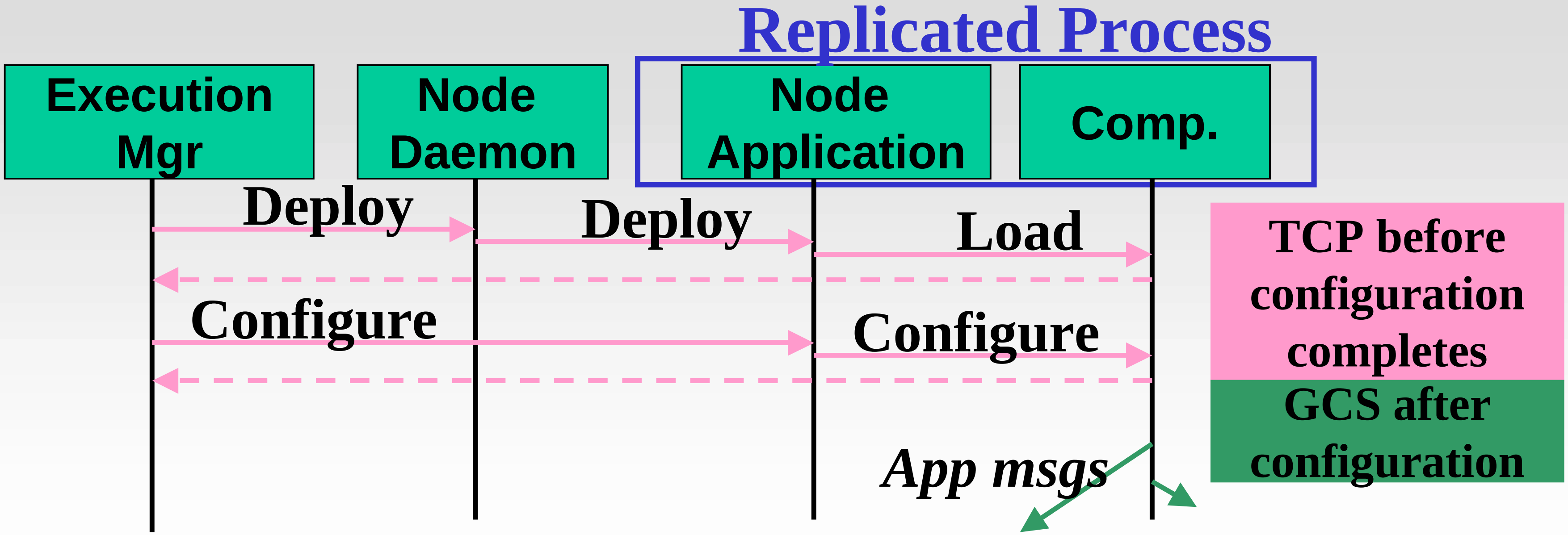
3 - CCM Deployment
The Problem

- Group Communication (GC) allows every replica to send and receive the same messages
- Some messages during deployment cannot be sent using GC



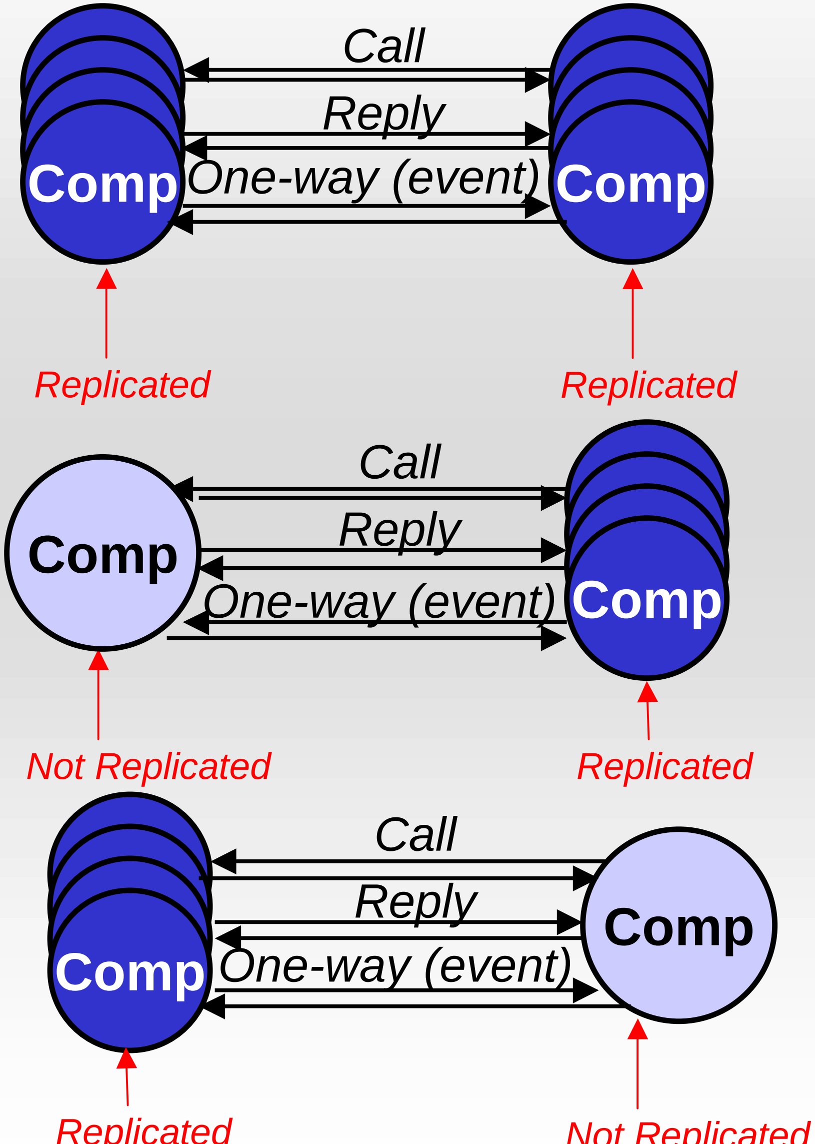
3 - CCM Deployment
The Solution

- Split deployment into epochs
  - Pre-deployment: only non-GCS
  - Post-deployment: only GCS



4 - Support for Peer-to-Peer Interactions

- CCM makes us of P2P interactions
  - Any component can be a client or server
  - Often at the same time
- Messages must be multicast if needed
- Duplicate calls/responses must be suppressed when needed



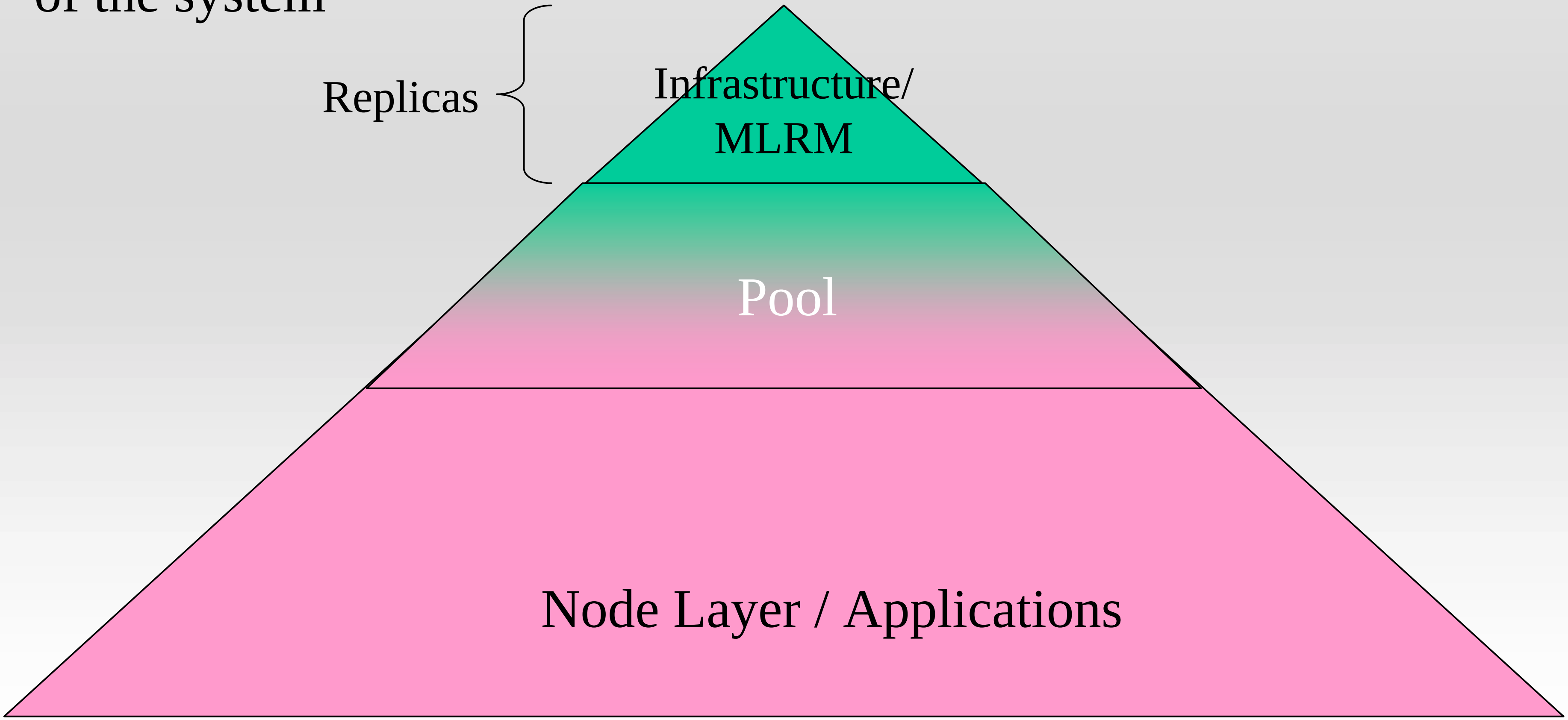
Outline

- Integrating Fault-Tolerance into a Resource Management Subsystem for Multi-Layer DRE Systems
- Providing Rapid Recovery from Faults
- Supporting Component Middleware
- Efficiently Supporting Fault-Tolerance in Large-Scale Systems

Applications

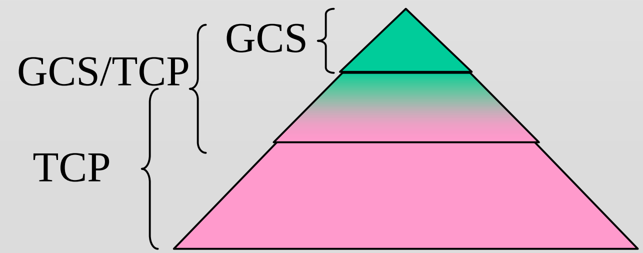
•Replicas are a small fraction of the entire system

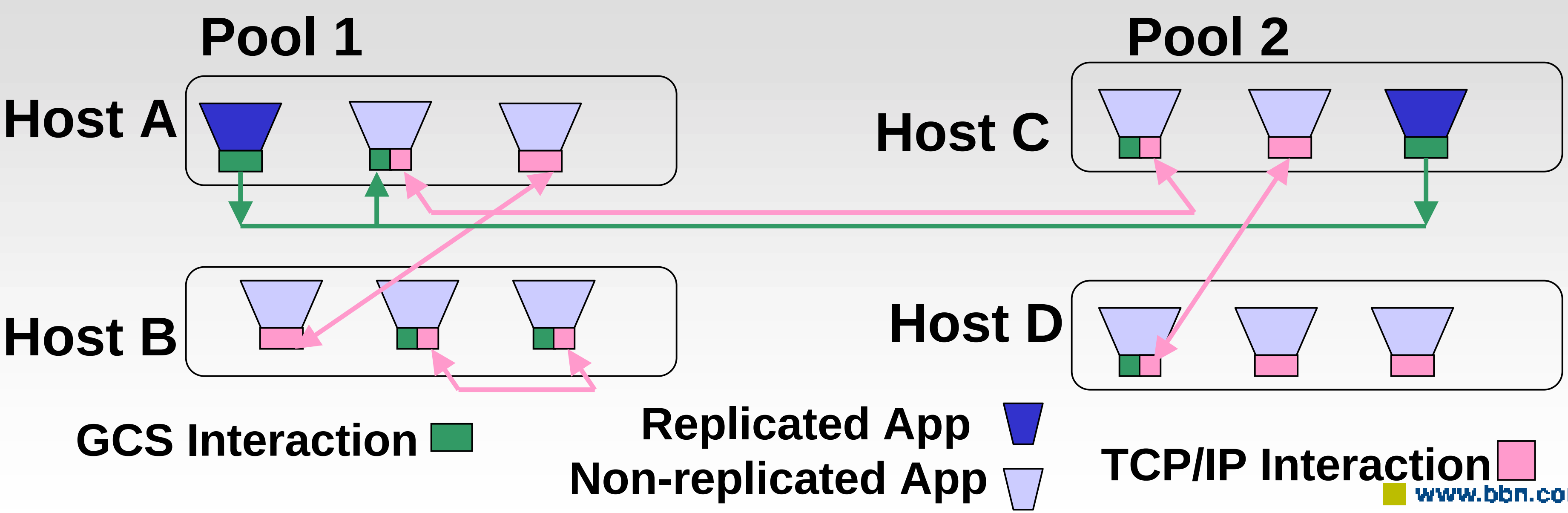
•Constraints due to replication should similarly affect a small fraction of the system



Coexistence of Group Communication between MLRM Layers

- Replicated elements need to use GC to maintain the ordering and consistency necessary for FT
  - Many elements do not need to use GCS
- We have mixed mode communication
  - Replicated elements speak only GCS
  - The immediate neighbors of replicated elements speak both GCS and non-GCS (e.g., TCP)
  - Other elements speak only non-GCS





Onward From the MLRM

- This MLRM FT solution is just one instance of a more general solution
  - One that has been implemented, tested and has recovered from multiple and cascading failures
- The FT techniques and solutions presented have wider applicability than MLRM
- Future work focuses on ease of use, expansion of, and composability of our FT solution

Questions?

- Integrating Fault-Tolerance into a Resource Management Subsystem for Multi-Layer DRE Systems
- Providing Rapid Recovery from Faults
- Supporting Component Middleware
- Efficiently Supporting Fault-Tolerance in Real Systems
- Questions?