Towards Access Control for Isolated Applications

SECRYPT 2016, Lisbon, Portugal

Kirill Belyaev and Indrakshi Ray
Computer Science Department
Colorado State University
Fort Collins, CO, USA

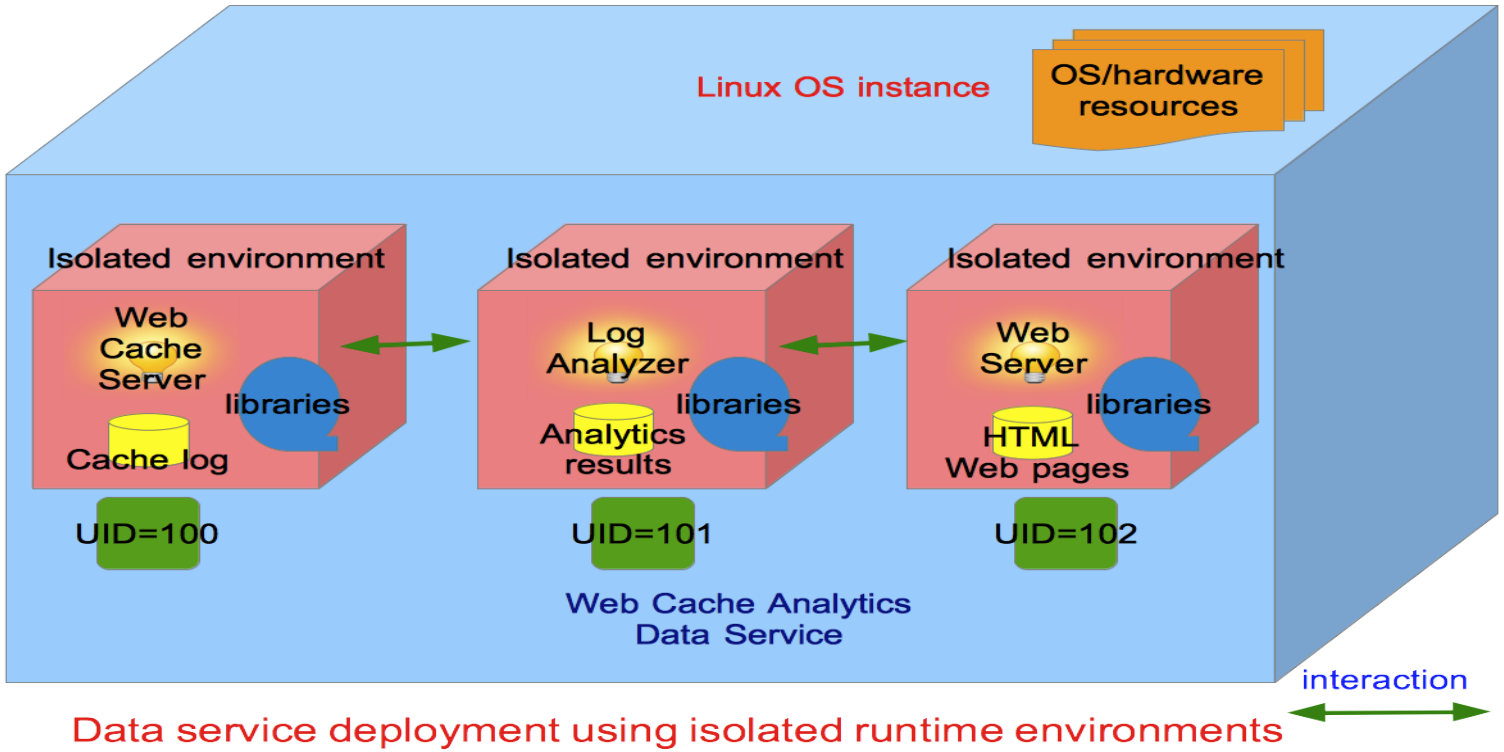
Introduction

Growing trends

- Modern single machine instance can deploy large number of application and data services
- UNIX based OS is a good candidate
- Services can be deployed within Isolated Runtime Environments (IRE)
- IRE can have all the necessary libraries for service deployment

Introduction
Sample deployment (1)

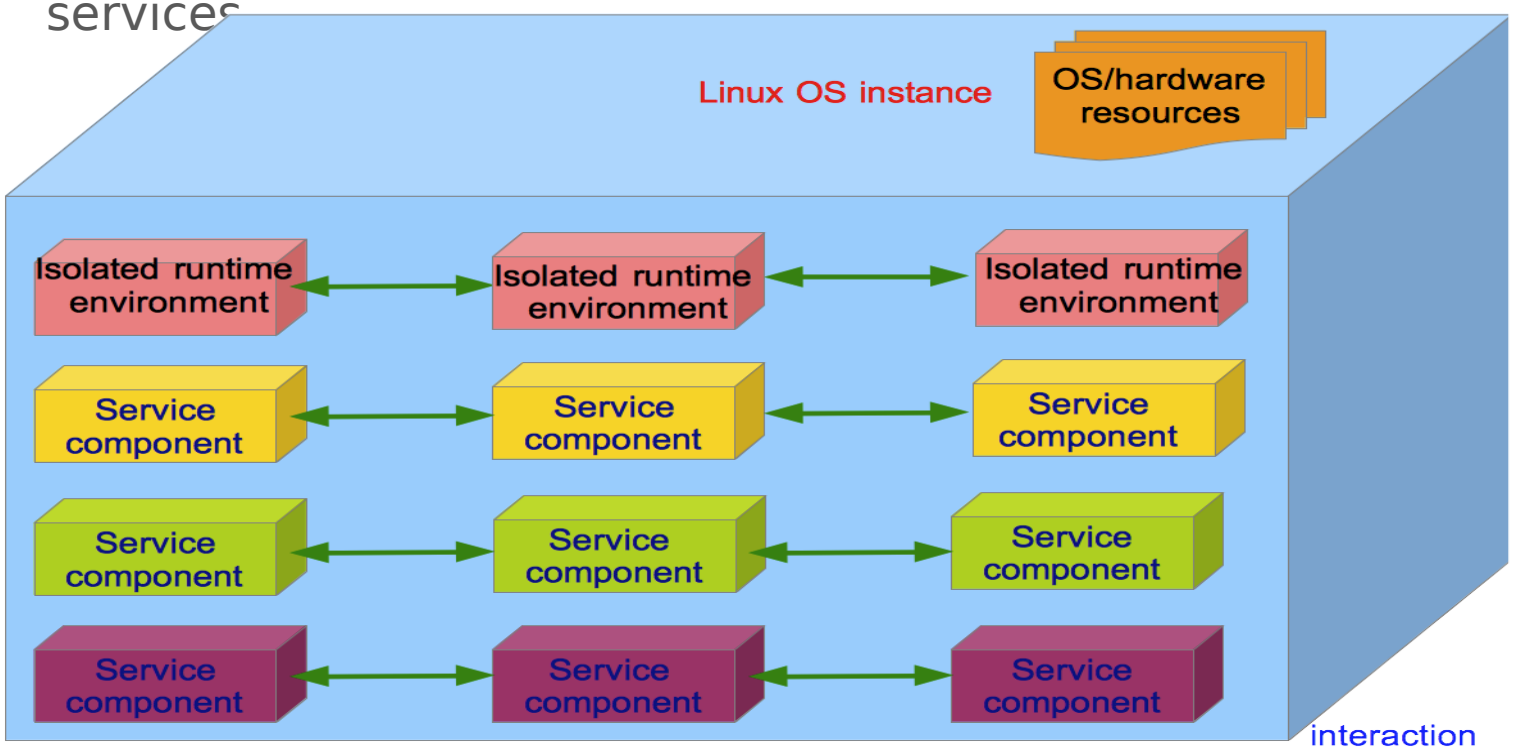
- Single data service can be partitioned into several components



Data service deployment using isolated runtime environments

Introduction
Sample deployment (2)

- Single OS node can host multiple multi-component data services

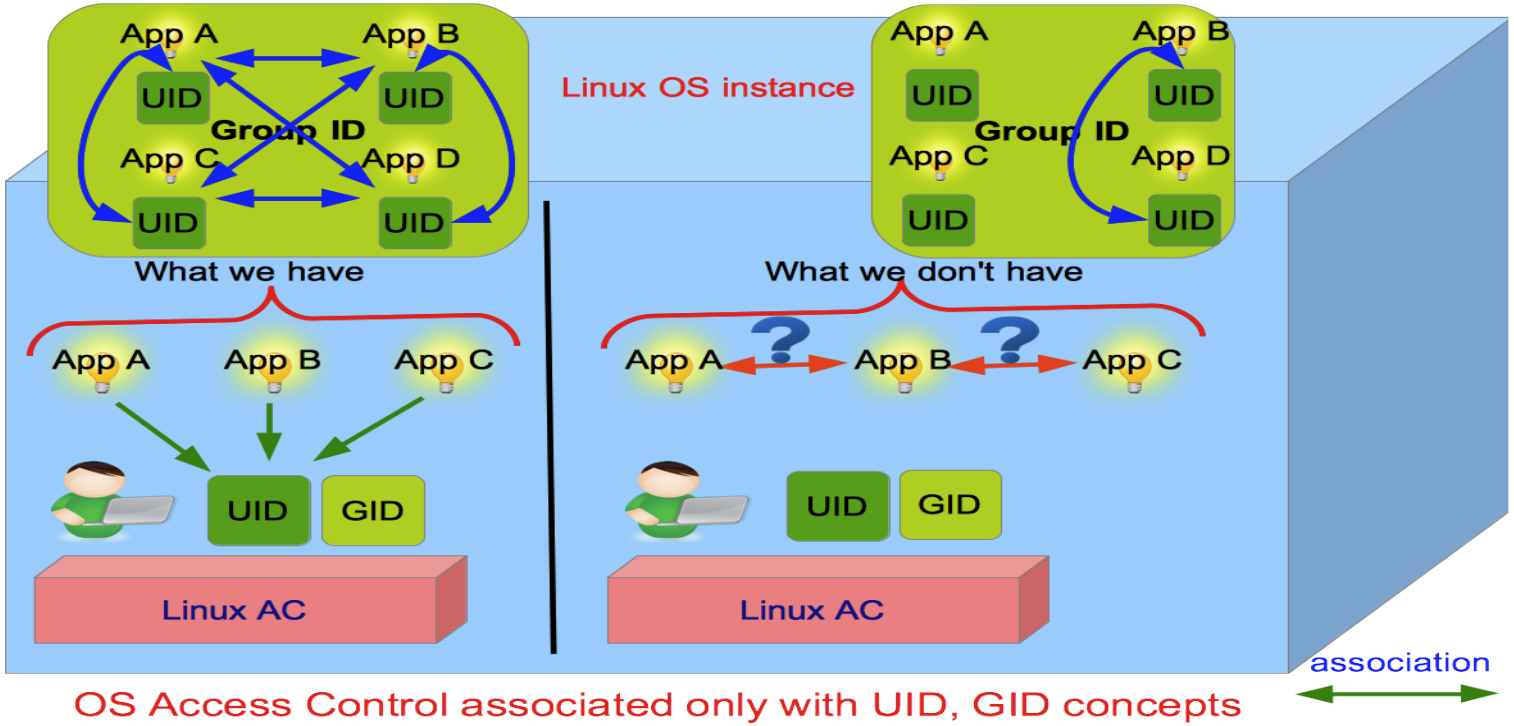


Data services deployment using isolated runtime environments

Problem

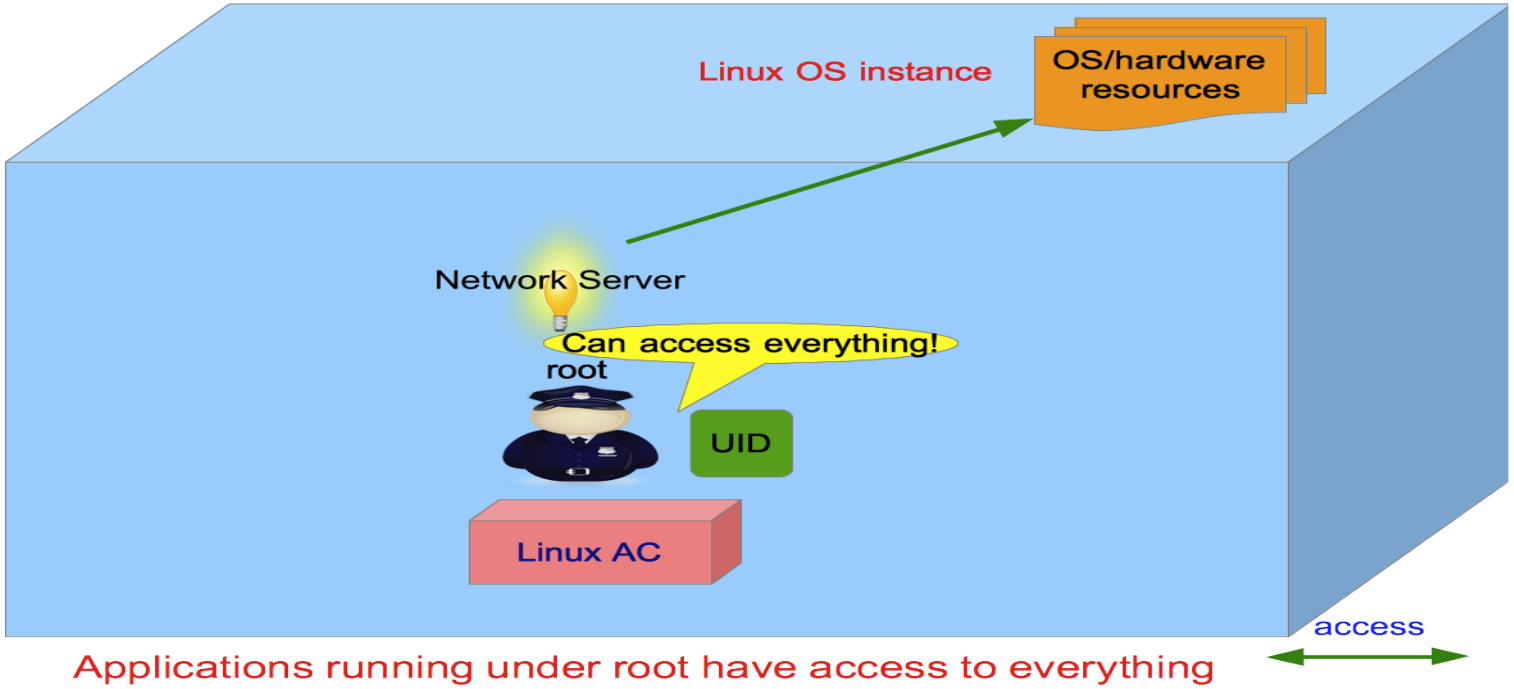
Limitations of OS Access Control

- Linux OS has no concept of application-level Access Control



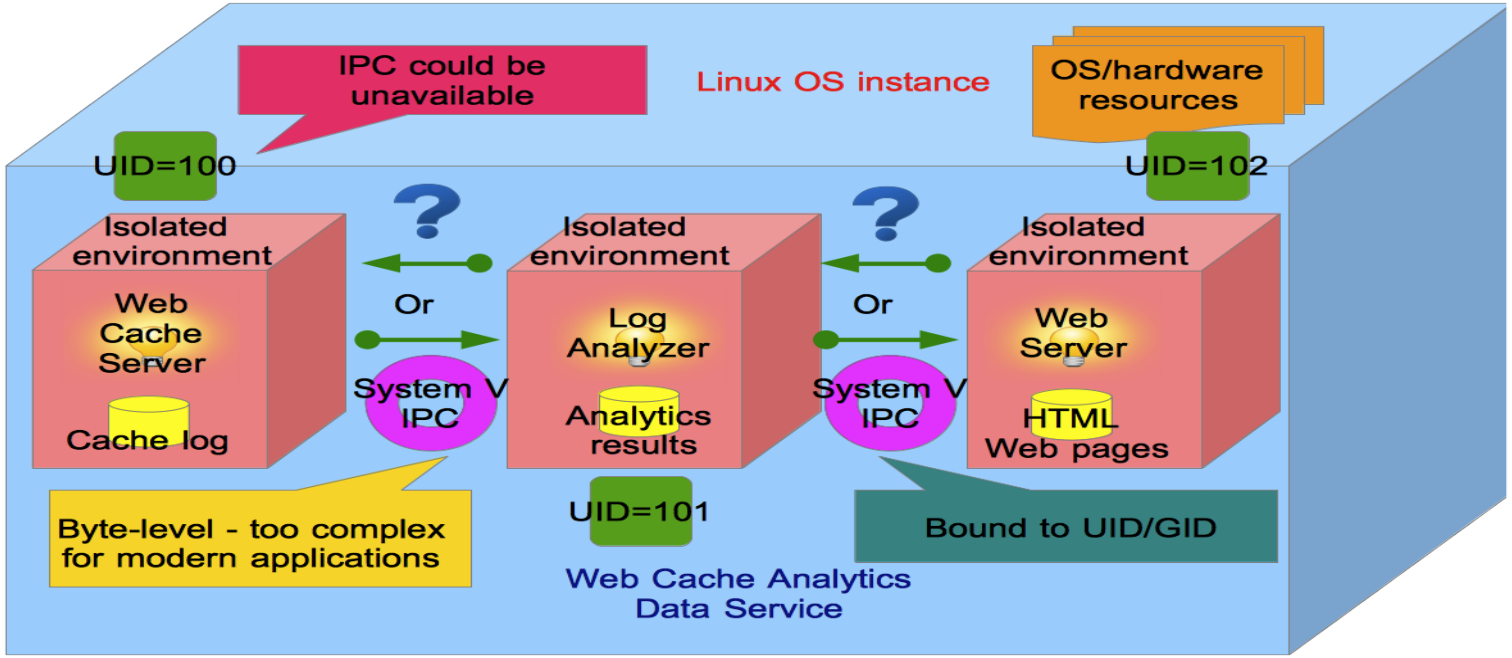
Problem
Privilege escalation

- Applications deployed with super-user privileges make them not bound to normal Access Control rules



Problem
Limitations of UNIX IPC

- UNIX System V Inter-Process Communication (IPC) does not offer regulated way of inter-application interaction



Controlled Sharing in isolated runtime environments using System V IPC

Motivation

- How to provide Access Control (AC) at the granularity of individual applications?
- How to confine applications with minimum privileges without running them with super user privileges?
- How to enable controlled application interaction across isolated runtime environments (IREs) without reliance on conventional UNIX IPC mechanisms?
- How to provide manageable user-space AC interface for a large number of data services?

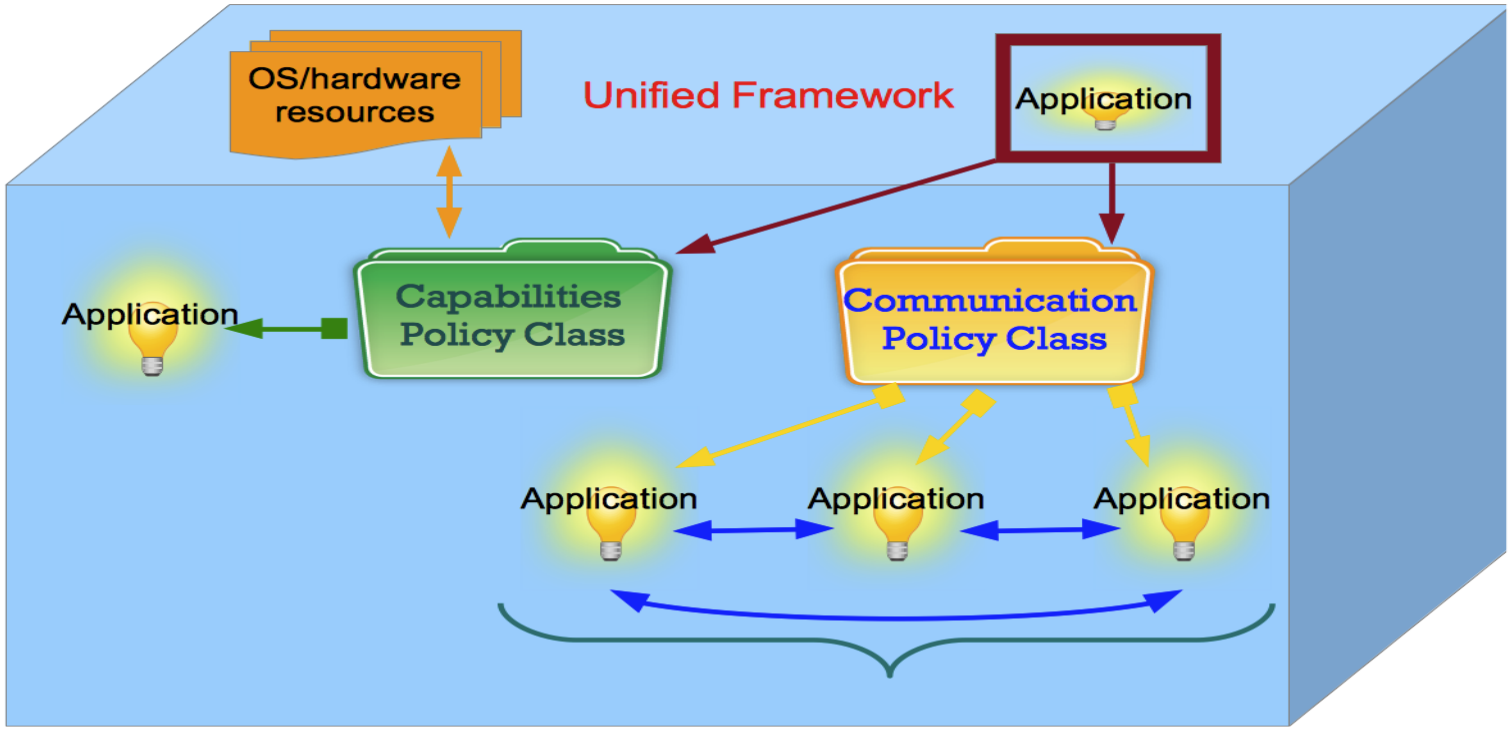
Proposed Contributions

- Develop a novel object-oriented framework for application-level AC in Linux:
  - Capabilities Policy Class Model – enables management and enforcement of application-level resource control
  - Communication Policy Class Model – enables management and enforcement of controlled inter-application interaction across isolated runtime environments
  - Unified framework – combines both models in a unified abstraction

Proposed Contributions

Unified Framework

- 2 types of policy classes to provide application-level Access Control



Policy classes in the unified framework

Outline

- Capabilities Policy Class Model
- Communication Policy Class Model
- Architecture
- Proposed Evaluation

Capabilities Policy Class Model

Capabilities Policy Class Model

What we have

- Access to OS/hardware resources is managed through kernel-space Linux (POSIX) capabilities
- Capabilities split up super user privileges into independent fragments
- Capabilities can manage resource control on the principle of least privilege
- No user-friendly management layer to manage capabilities

Capabilities Policy Class Model

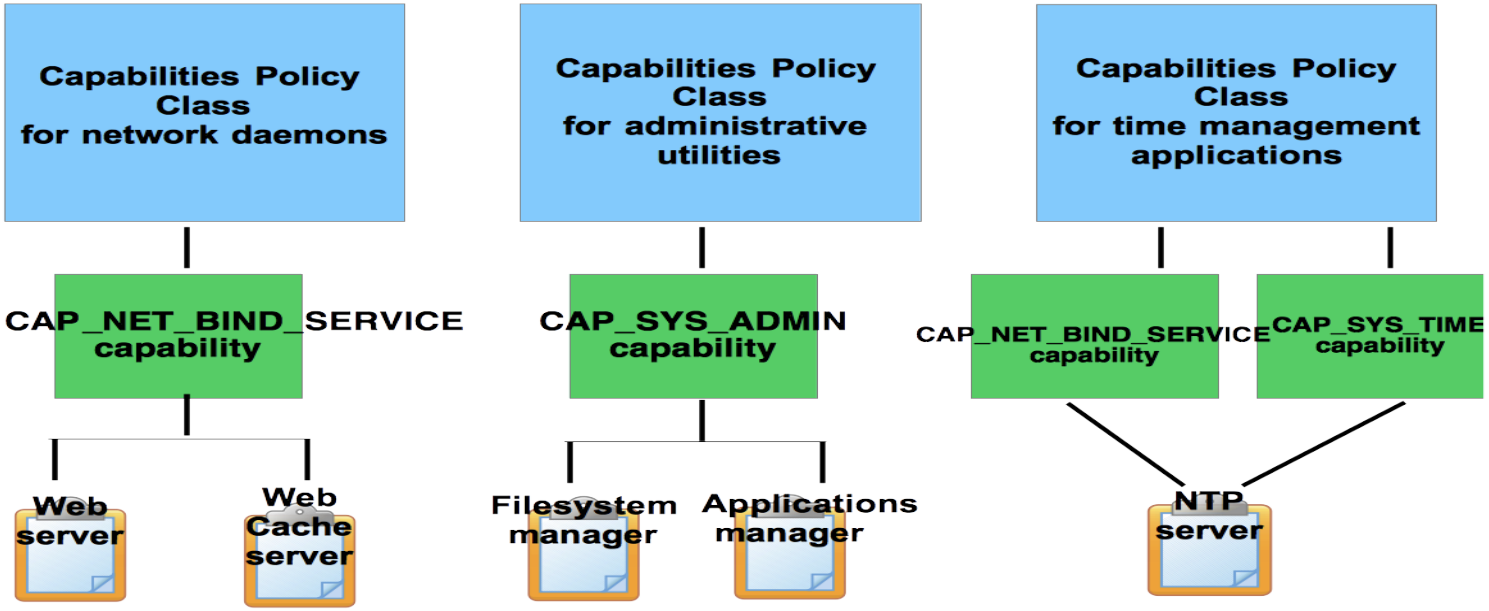
Sample capabilities

- Linux capabilities for instance include:
  - CAP_AUDIT_CONTROL - Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules
  - CAP_DAC_OVERRIDE - Bypass file read, write, and execute permission checks
  - CAP_IPC_LOCK - Lock memory (mlock(2), mlockall(2), mmap(2), shmctl(2))
  - CAP_NET_ADMIN - Perform various network-related operations such as: interface configuration; administration of IP firewall, masquerading, and accounting; modify routing tables; bind to any address for transparent proxying; set type-of-service (TOS); clear driver statistics; set promiscuous mode; enabling multicasting; use setsockopt(2) to set the advanced socket options
  - CAP_SYS_BOOT - Use reboot(2) and kexec_load(2) system calls

Capabilities Policy Class Model

What we need

- Capabilities could be grouped into various policy classes



Linux Capabilities in Capabilities Policy Classes Model

Capabilities Policy Class Model

Operations

- The following high-level operations are proposed:
  - create a capabilities policy class
  - add/remove capabilities to/from a policy class
  - show capabilities in a policy class
  - add/remove applications to/from a policy class
  - show/count apps in a policy class

Capabilities Policy Class Model

Sample usage

- Here is how the model is used in practice:

  - *SHOW_CAPABILITIES;*
  - *SHOW_POLICY_CLASSES;*
  - *CREATE_POLICY_CLASS 1 general_applications_policy_class;*
  - *ADD_POLICY_CLASS_POLICY 1 CAP_KILL;*
  - *ADD_POLICY_CLASS_POLICY 1 CAP_CHOWN;*
  - *MOVE_APP_TO_POLICY_CLASS /containers/A/apps/app-A 1;*
  - *REMOVE_POLICY_CLASS_POLICY 1 CAP_CHOWN;*
  - *SHOW_POLICY_CLASS_POLICIES 1;*
  - *SHOW_POLICY_CLASS_APPS 1;*

Communication Policy Class Model

Communication Policy Class Model

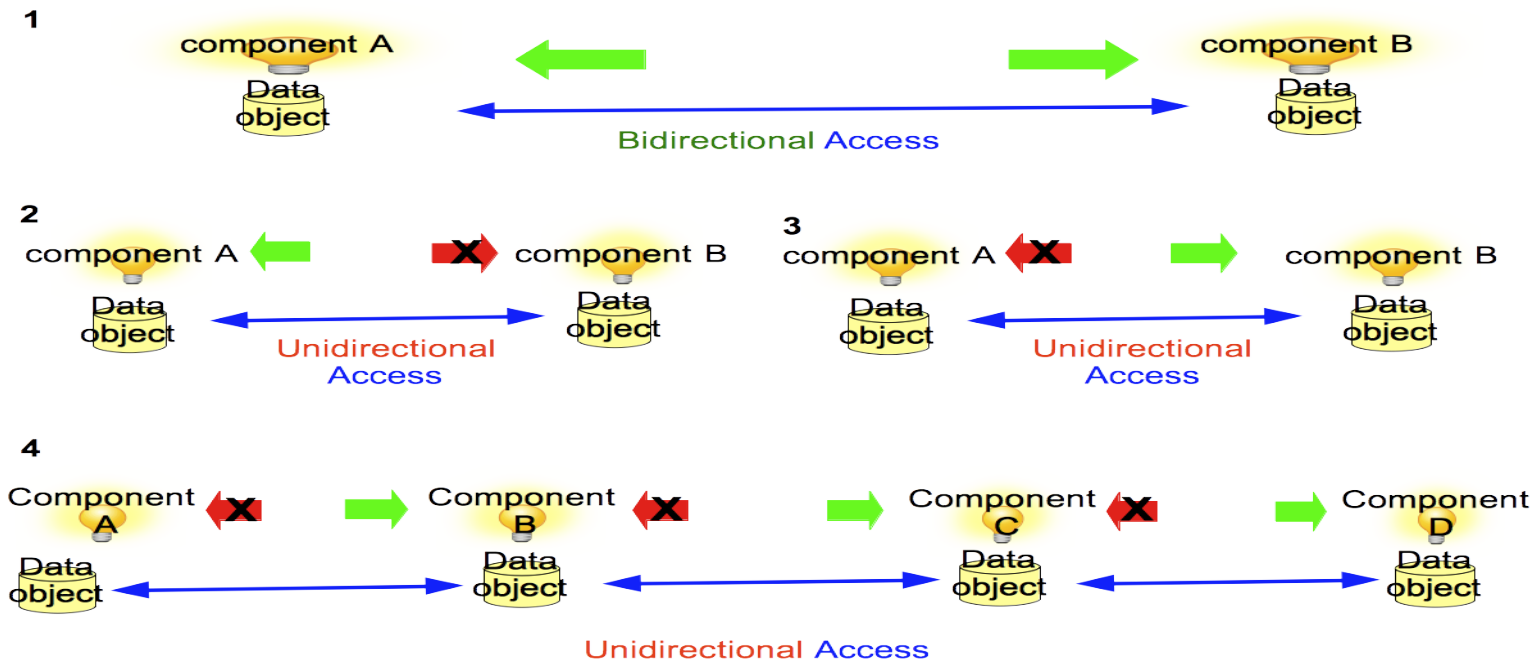
What we have

- A group of applications (service components) may provide a single data service
- A single application can be partitioned into a set of compartments
- Applications in separate isolated runtime environments (IREs) need to communicate
- Communication often involves:
  - sharing data objects
  - Exchanging control objects

Communication Policy Class Model

What we need

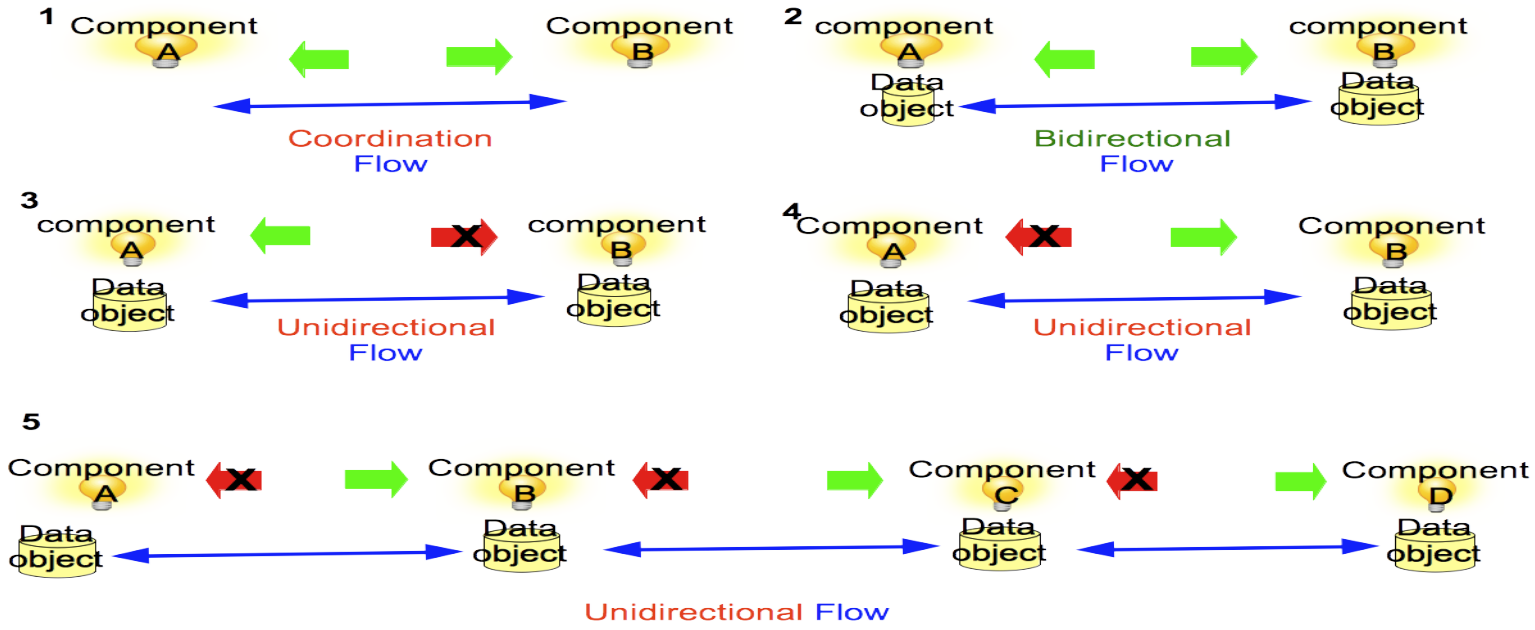
- Provide *bidirectional* replication of data objects
- Provide only *unidirectional* replication of data objects



Communication Policy Class Model

Types of Communication

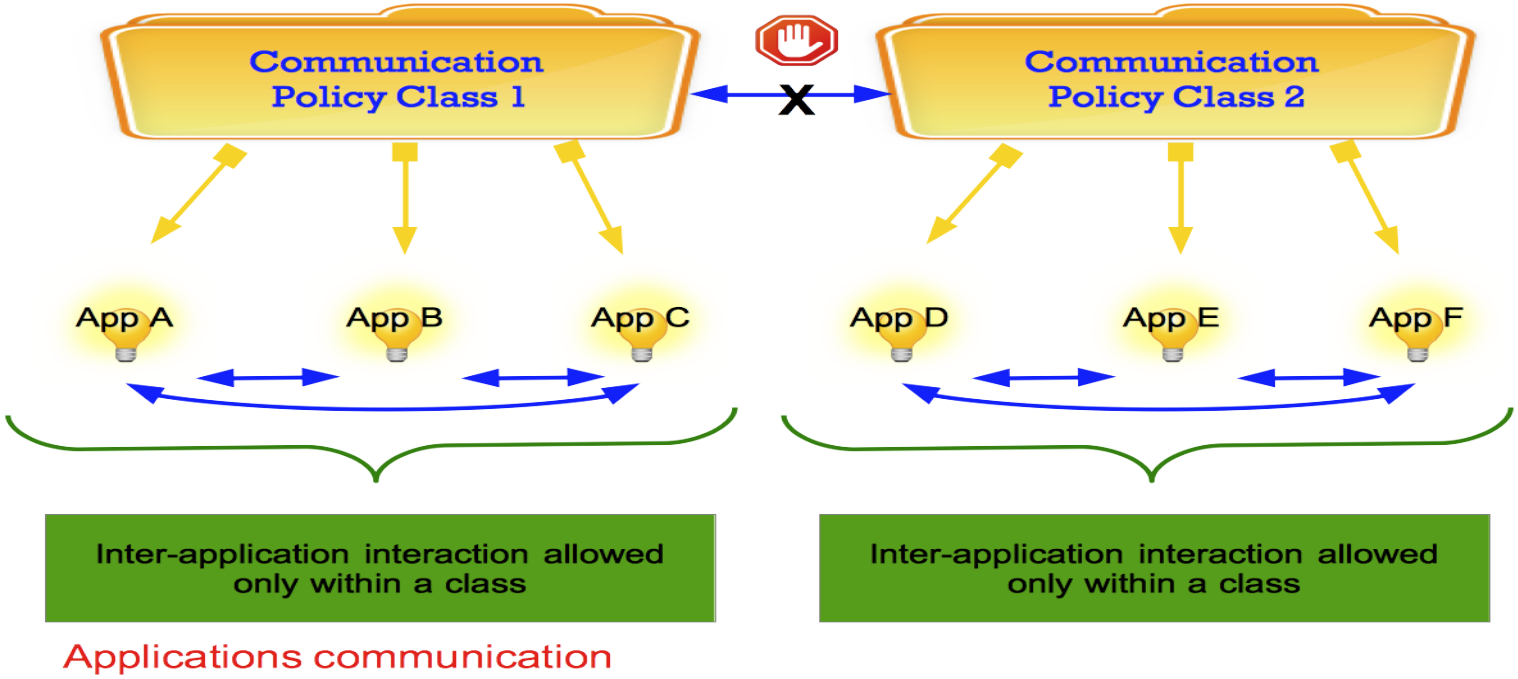
- Group of applications (service components) may:
  - Coordinate - exchange of coordination messages
  - Collaborate - share mutual data objects via replication



Communication Policy Class Model

Properties

- Group of applications can communicate only within the **same** class



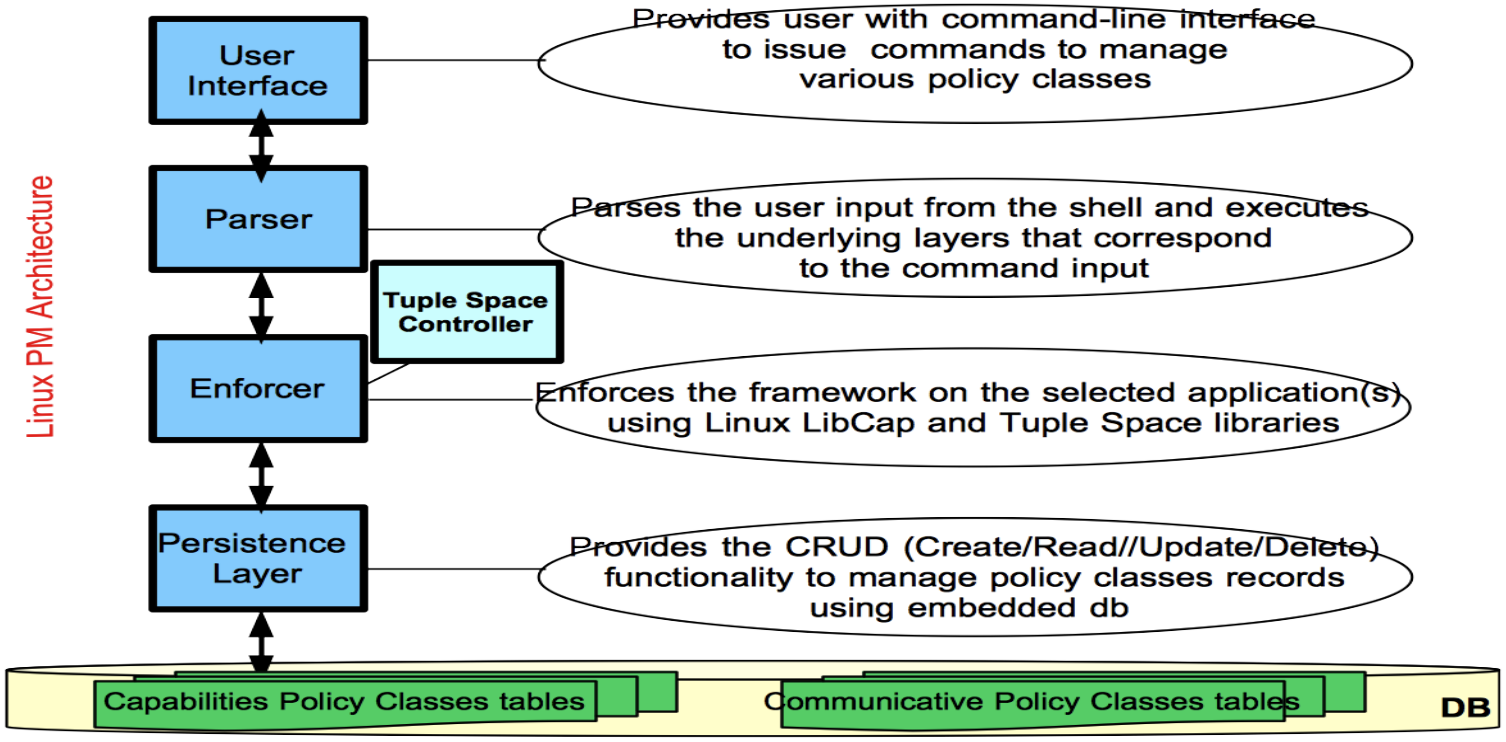
Communication Policy Class Model
Operations

- The following high-level operations are proposed:
  - create a communication policy class
  - add/remove applications to/from a policy class
  - show/count apps in a policy class
  - add/remove associations of an app to request a replica of a data object(s) to/from a policy class
  - enable/disable application coordination with other application(s) in a policy class

Architecture

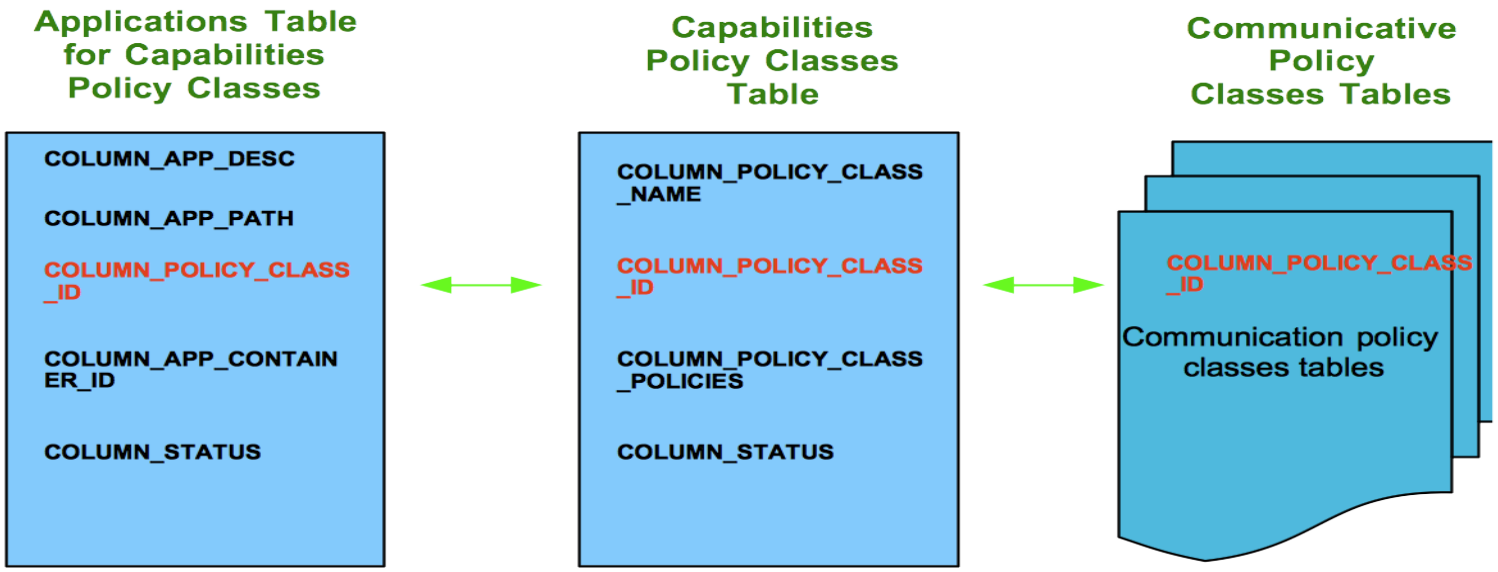
Architecture
System design (1)

- Unified framework proposed in the form of Linux Policy Machine



Architecture
System design (2)

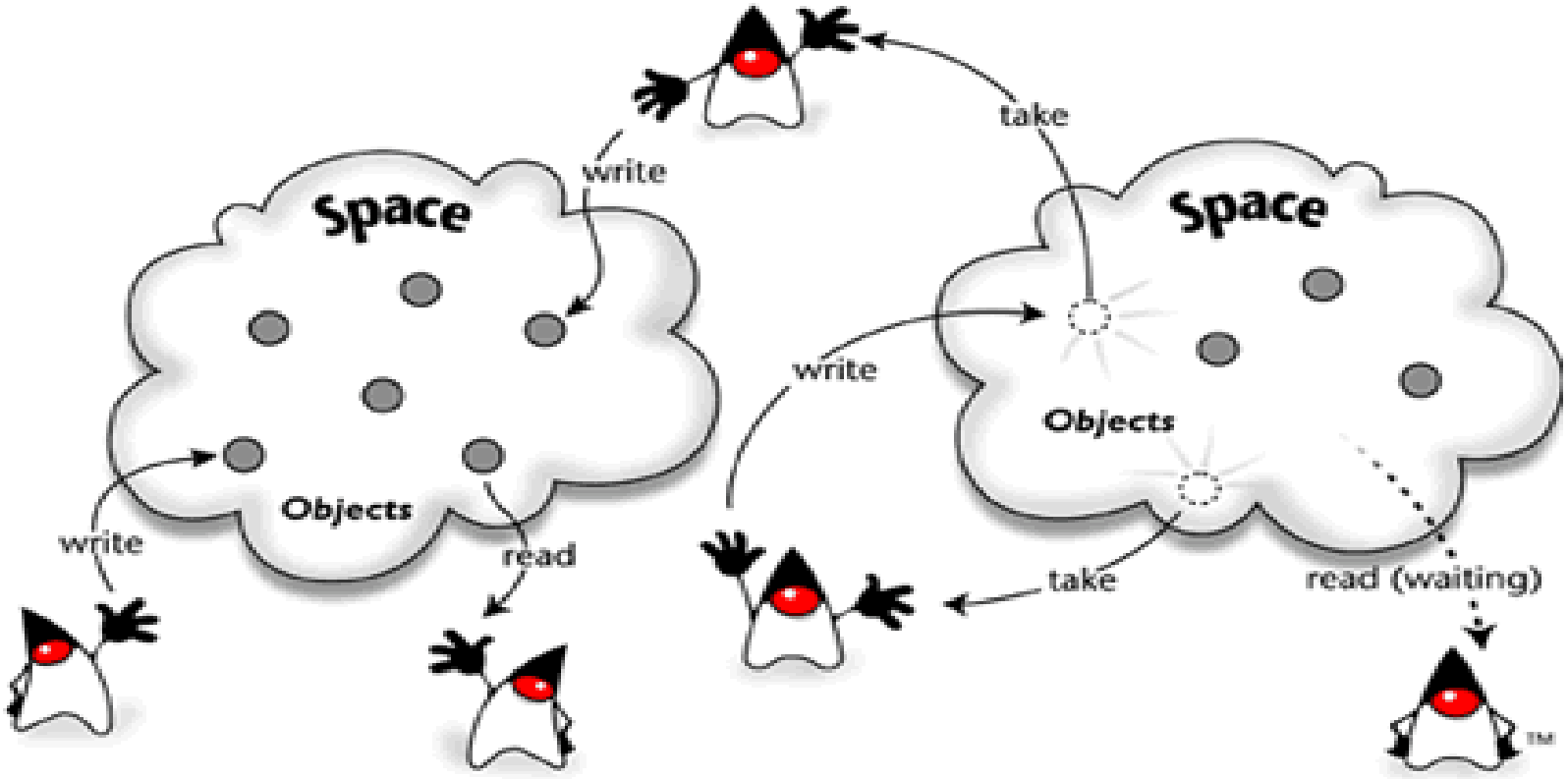
- Unified framework stored in the embedded Sqlite database



Schema of Persistence Layer

Architecture
Communication design (1)

- Adapt indirect communication paradigm - tuple space to enforce decoupled content-based inter-application



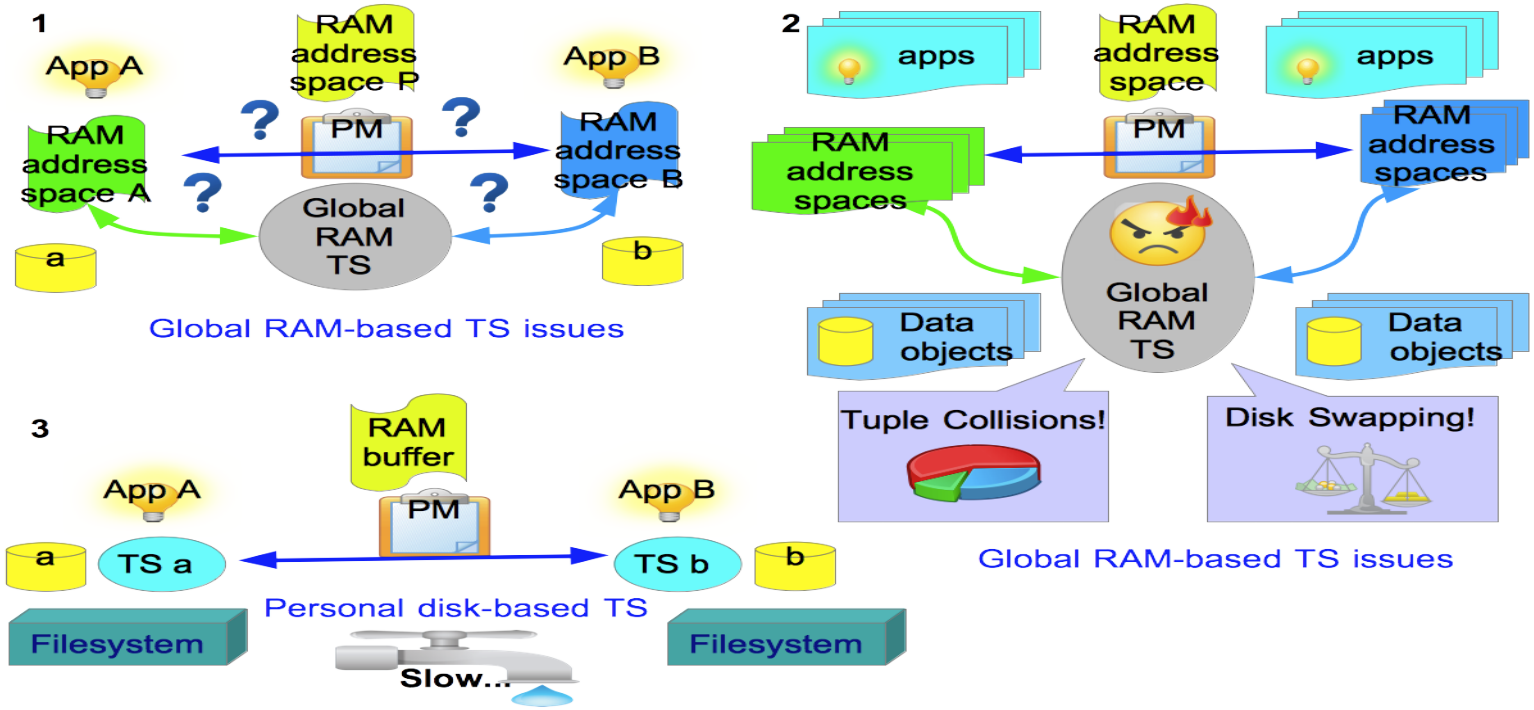
Architecture
Communication design (2) - Limitations

- The basic model relies on a global shared RAM tuple space
- Has number of issues:
  - Tuple collisions could happen
  - Wide array of possible security attacks
  - Overheads of memory utilization
  - Could be inaccessible due to access control policies
  - Suitable mainly for a single application with multiple threads

Architecture

Communication design (3) - Adaptation

- Personal tuple space per application
- Disk/flash based implementation



Architecture

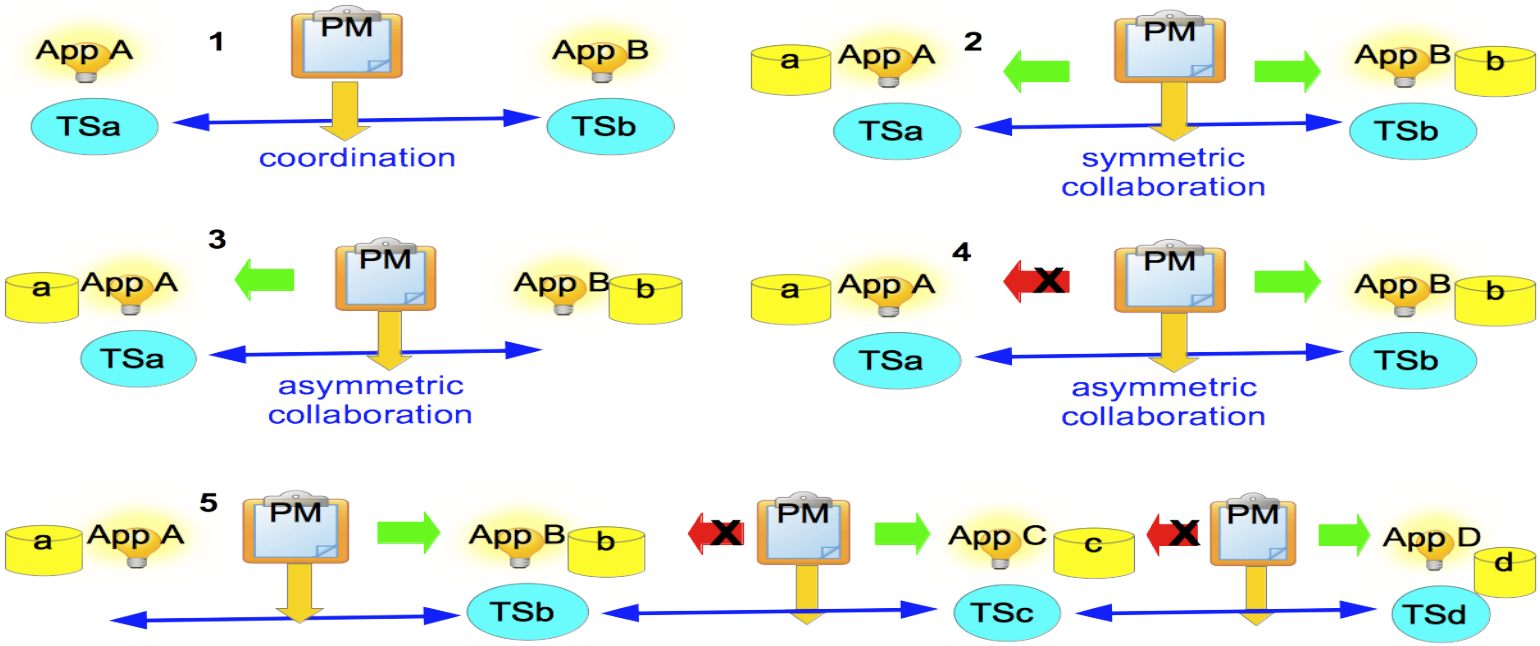
Communication design (4) - Operations

- Propose a set of tentative operations - tuple space *calculus*:
  - *create tuple space*
  - *delete tuple space*
  - *read* operation - returns the value of individual tuple without affecting the contents of a tuple space
  - *append* operation - adds a tuple without affecting existing tuples in a tuple space
  - *take* operation - returns a tuple while removing it from a tuple space

Architecture

Communication design (5) - Patterns

- Application allowed to perform all calculus operations
- Policy Machine restricted to read and append operations



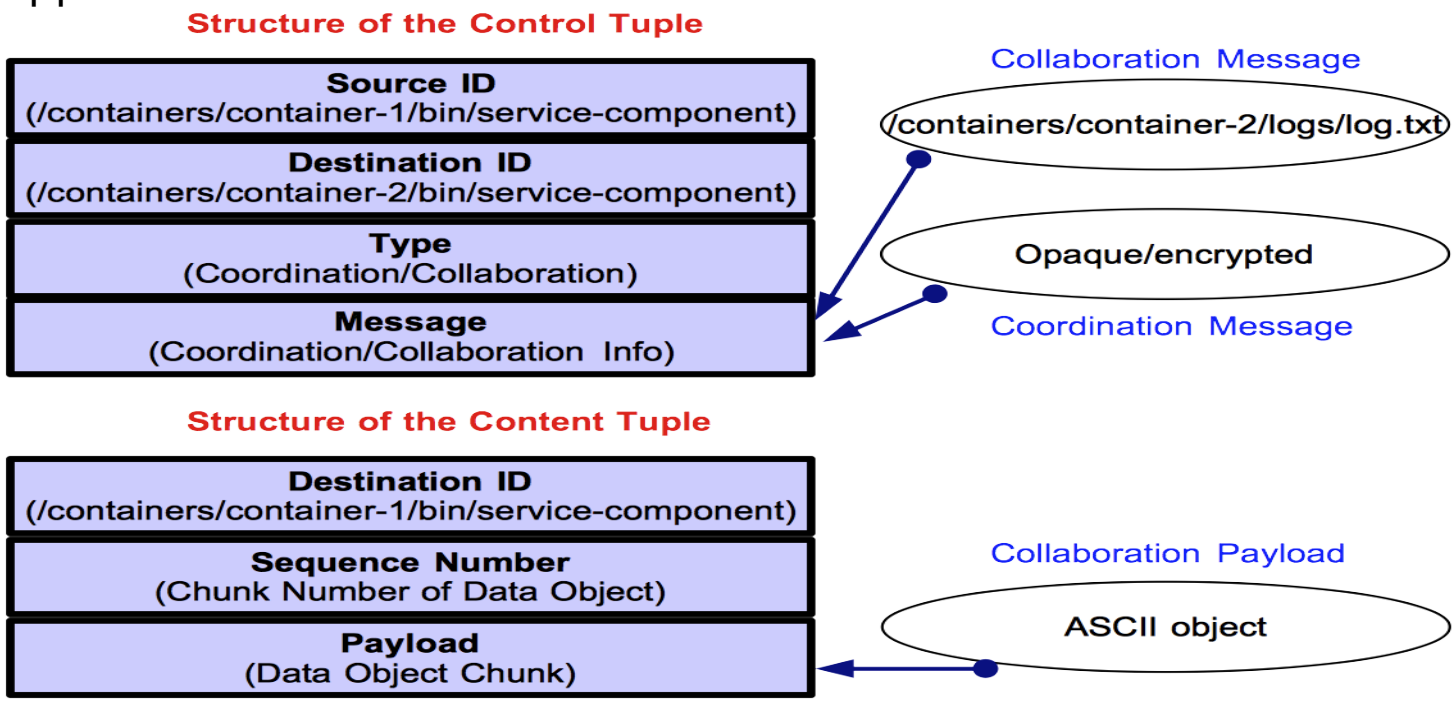
Asymmetric collaboration using a pipeline mechanism

Applications group communication patterns using tuple spaces

Architecture

Communication design (6) – Tuple Types

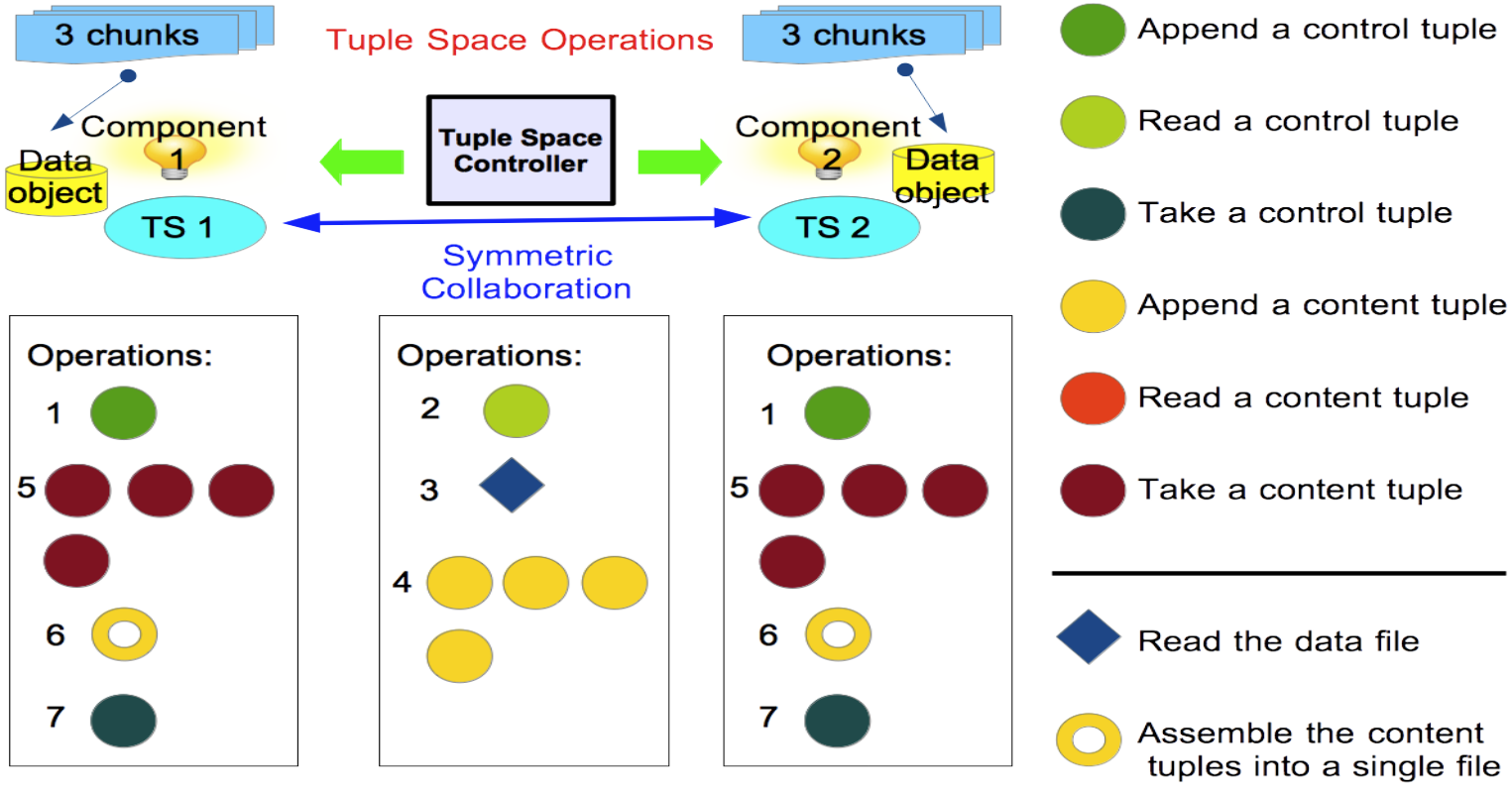
- Control tuples - provide the instructions about coordination or sharing
- Content tuples - mechanism by which data gets shared across applications



Architecture

Communication design (7) – Collaborative Transaction

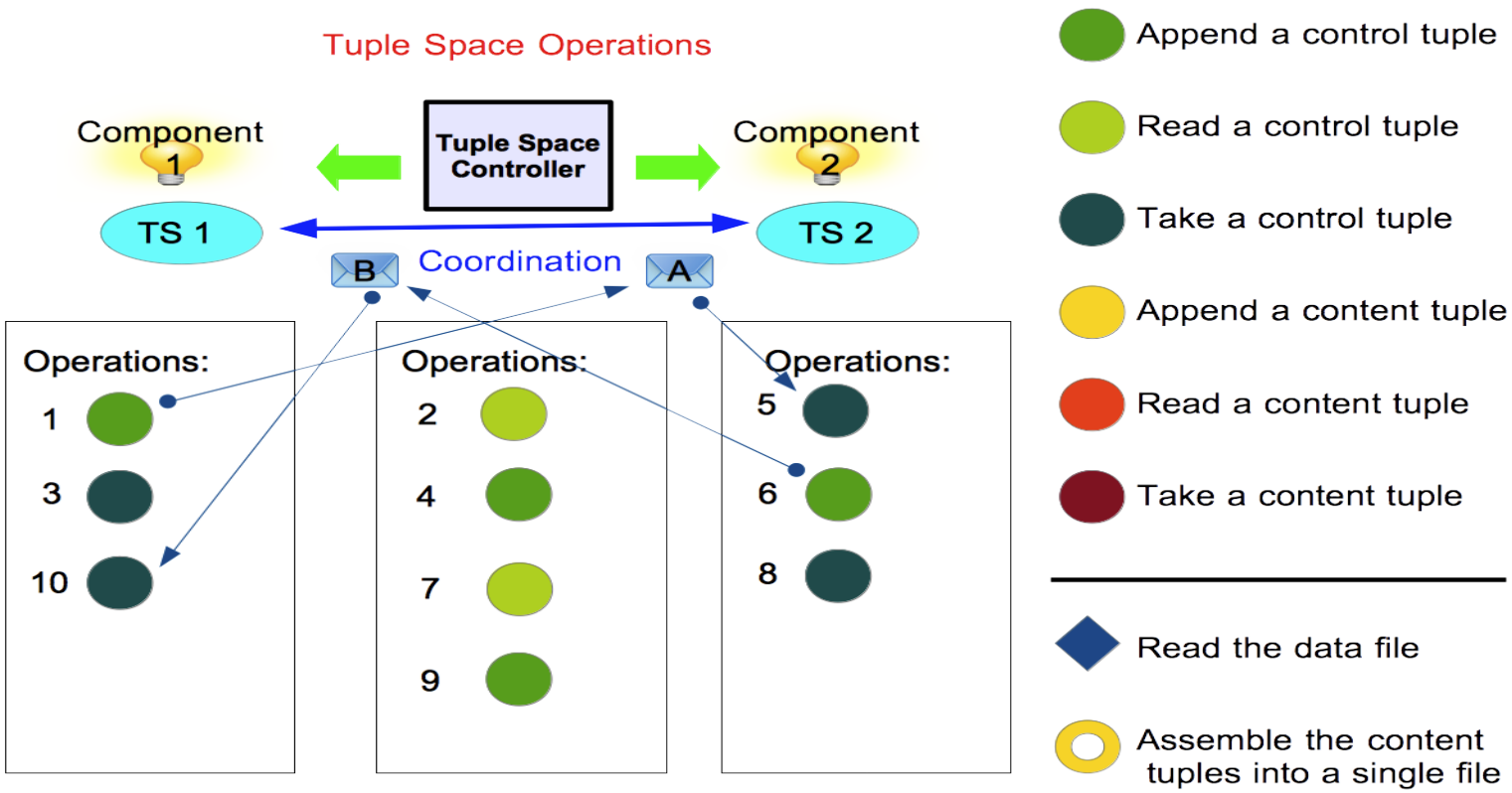
- Policy machine periodically checks for the presence of control tuples



Architecture

Communication design (8) – Coordinative Transaction

- Policy machine periodically checks for the presence of control tuples



Proposed Evaluation

- Capabilities policy classes do not incur performance overhead
  - no extra disk I/O aside from the I/O load of the base system
  - no additional memory utilization
- Communication policy classes are resource intensive
- Evaluate performance with integrated tuple space controller:

Sizes of mediated data objects	**Number of communicating applications**
Disk I/O utilization	Disk I/O utilization
CPU utilization	CPU utilization
RAM utilization	**RAM utilization**

Summary

- Unified Framework consists of:
  - model of *Capabilities Policy Classes* – regulates Linux capabilities on a per-application granularity
  - model of *Communication Policy Classes* - regulates inter-application interaction
- Enforcement of the framework is done via:
  - Capabilities model – *Linux LibCap library*
  - Communication model – *Tuple Space Library/Controller (to be developed)*
- Targets deployment of decoupled multi-component data services on multi-core server instance

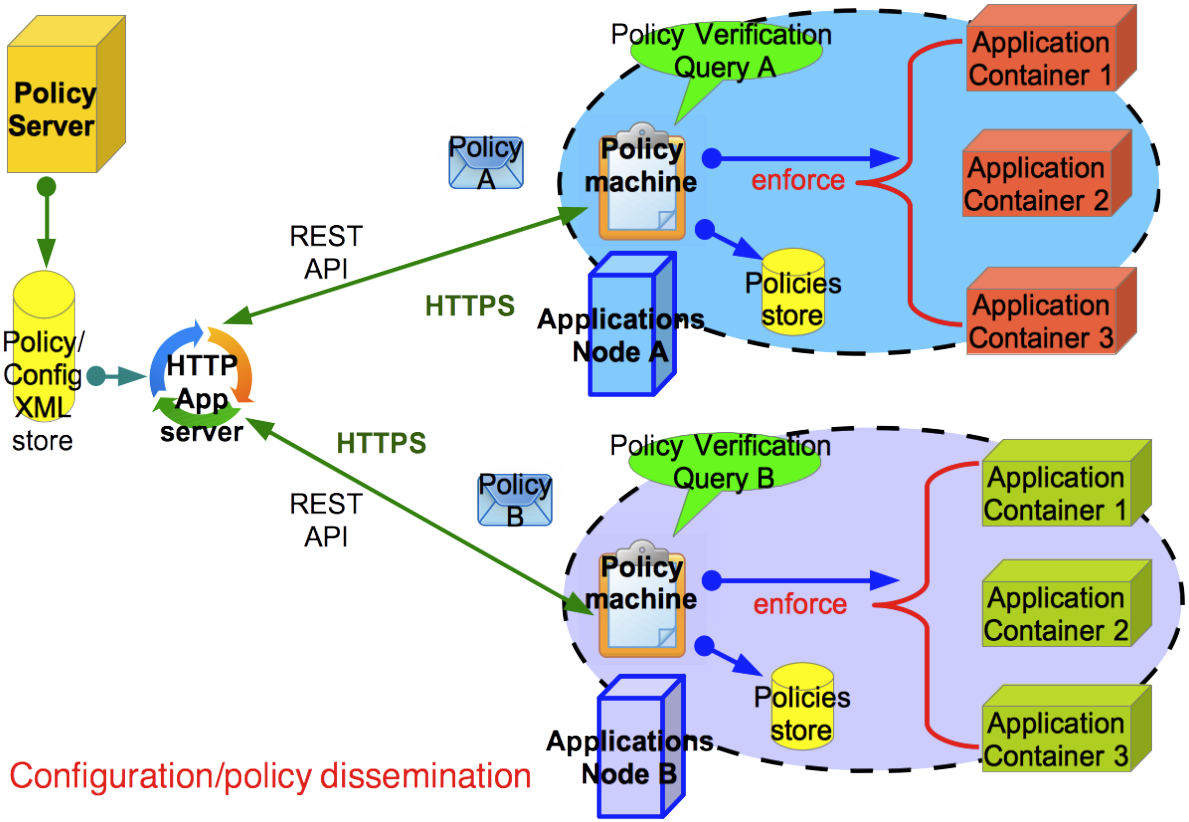
Future Work

- Address the distributed deployment of unified framework
- Address the problems of policy representation and validation
- Investigate additional security for tuple spaces on fast persistent storage
- Investigate scalability issues
- Investigate the applicability for Dockers

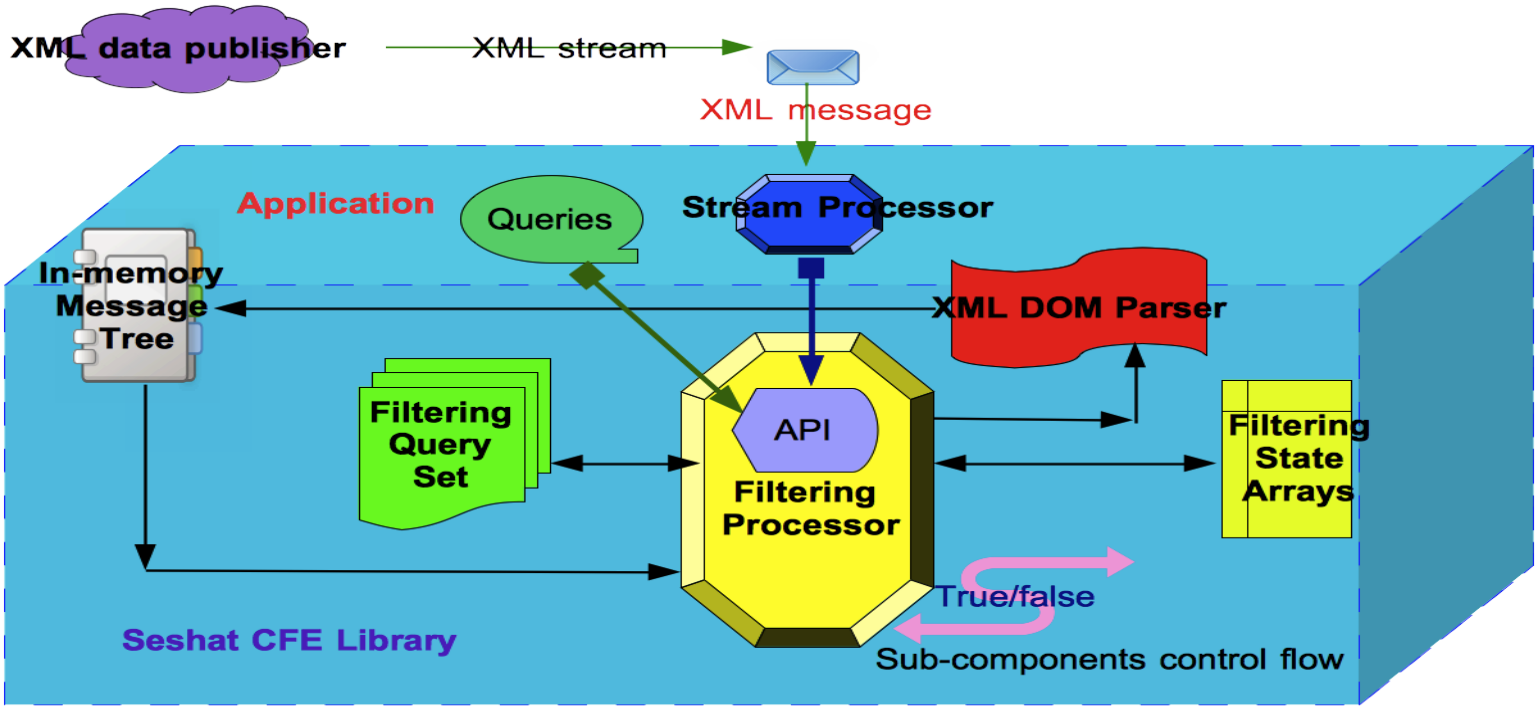
Sample Service Policy in XML

- **<?xml version="1.0" encoding="UTF-8"?> <AutomationPolicy productID="“EEZALAdapter“ version="3.2.2" xmlns="http://www.ibm.com/TSA/Policy.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.ibm.com/TSA/Policy.xsd EEZALPolicy.xsd "> <PolicyInformation> <PolicyName>**Agentless adapter sample policy**</PolicyName> <AutomationDomainName>**AgentlessDomain**</AutomationDomainName> <PolicyToken>**1.0.5**</PolicyToken> <PolicyAuthor>**LPM**</PolicyAuthor> <PolicyDescription>** Agentless adapter sample policy. ---------- **</PolicyDescription> </PolicyInformation> ... </AutomationPolicy>**

Policy orchestration mechanisms



Policy validation with XML content filtering engine



Seshat Content Filtering Engine

Thank you for your attention!

Questions?