RINA Java Prototype demo and development plans

PhD Course on Future Network Architectures and Experimentation

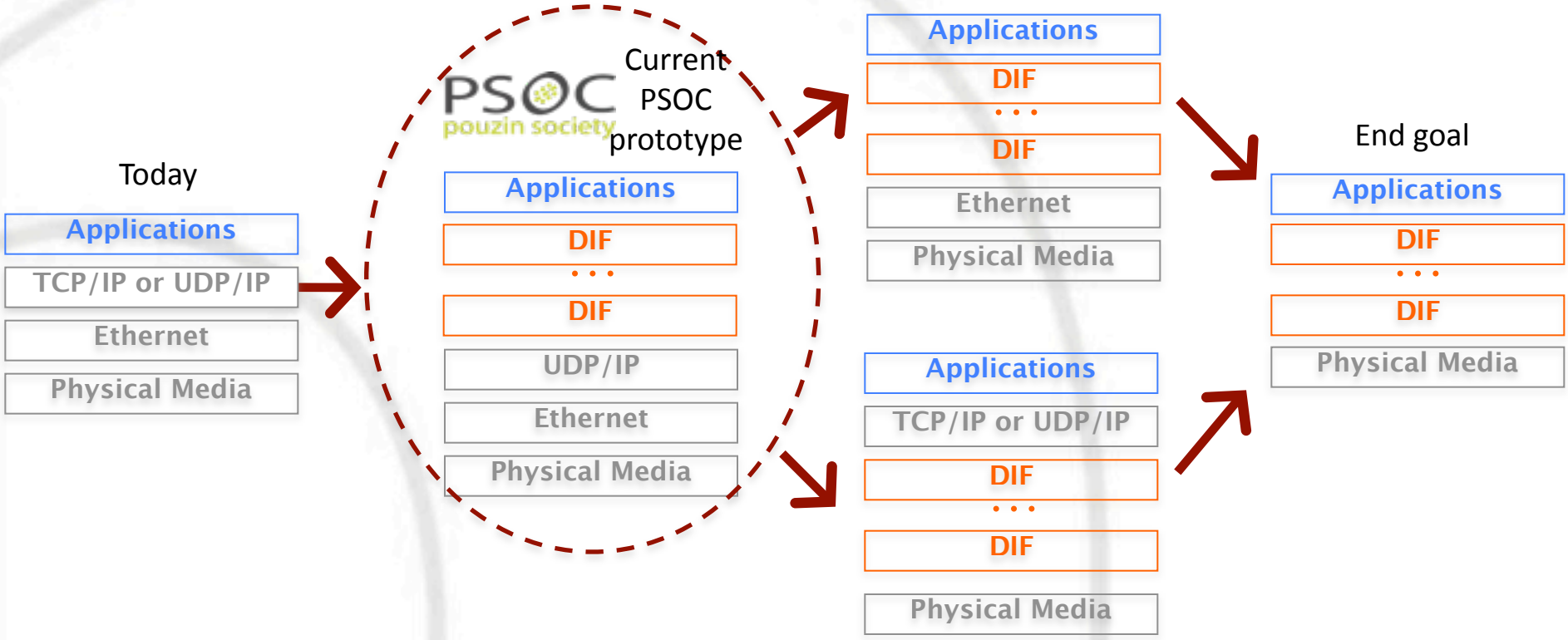
University of Kaiserslautern, March 7th, 2012

Miguel Ponce de Leon, TSSG
John Day, Boston University
Eduard Grasa, Fundació i2CAT

Outline

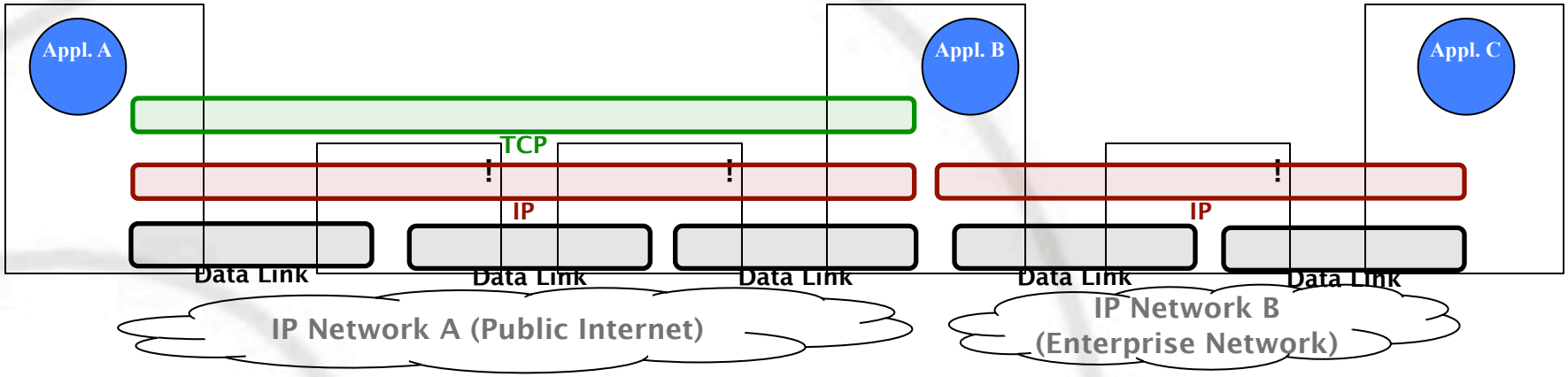
- **RINA Adoption strategy and current prototype rationale**
- Prototype implementation phases
- Prototype description
- Demo

RINA Adoption strategy

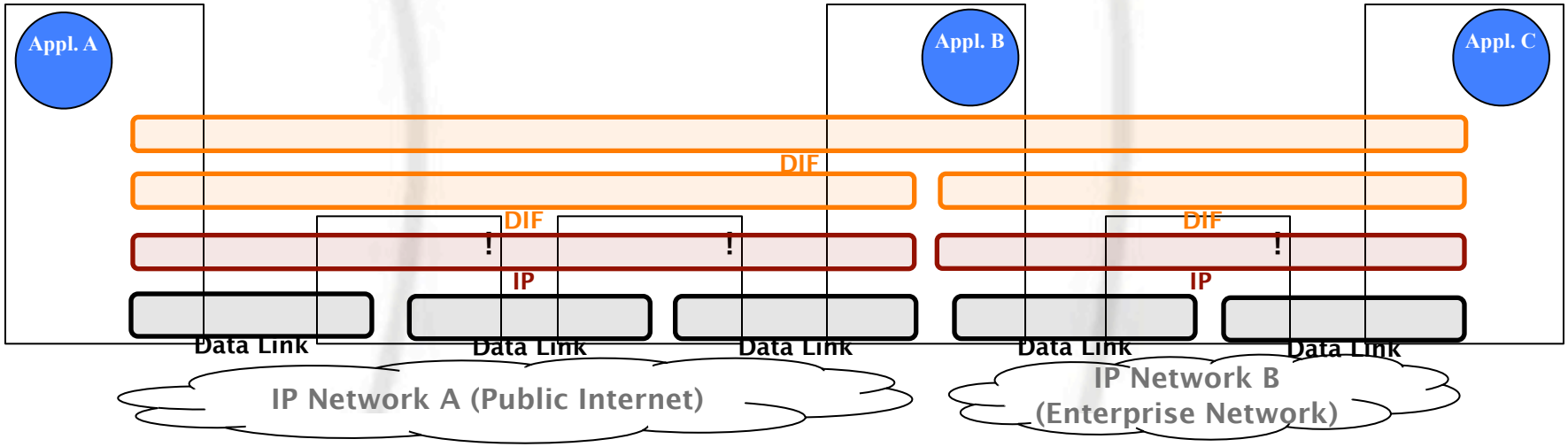


- Start as an overlay to IP, validate technology, work on initial concepts, develop DIF machinery.
  - Useful by itself: internetwork layer(s), decouple application from infrastructure, improved application API, support for multi-homing and mobility.

RINA over IP benefits: Internetwork layer(s)

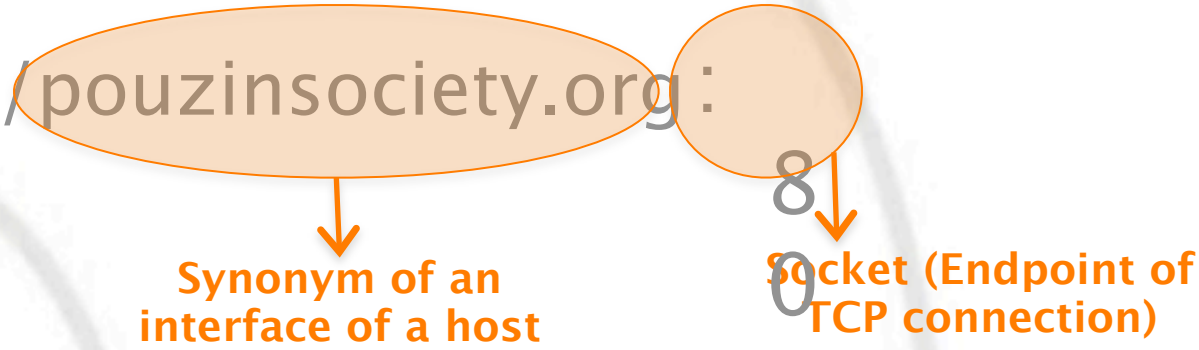


- What if application A wants to communicate with Application C?
  - It cannot do it, unless you start deploying middleboxes like NATs, application-layer gateways, … The architecture doesn't accommodate internetworking!



RINA over IP benefits: Separate applications from infrastructure

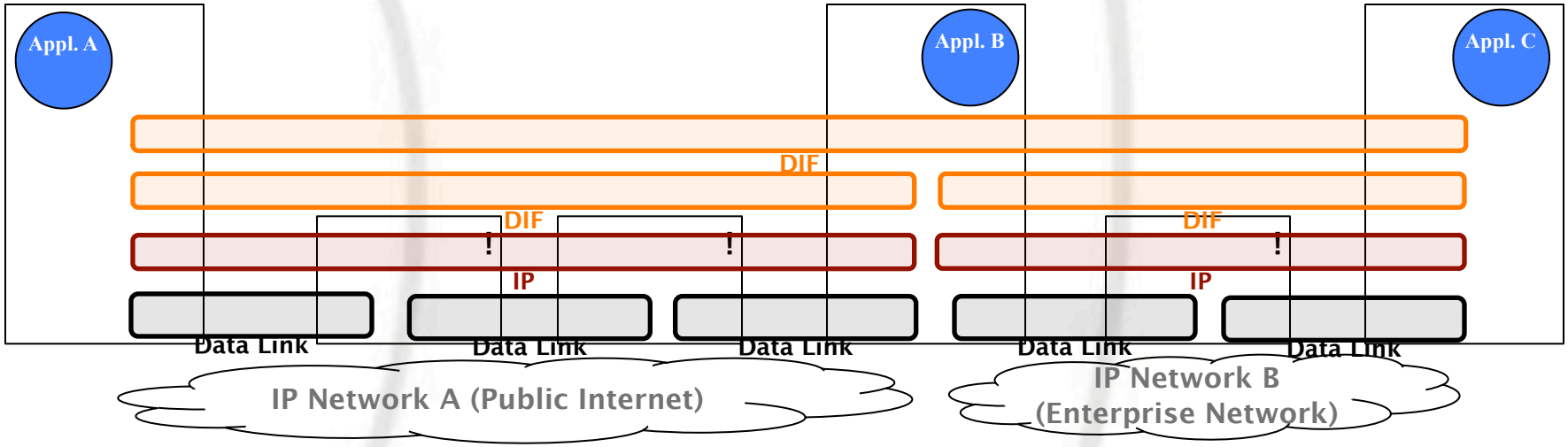
- The current application namespace is tied to IP addressing and TCP/UDP port numbers:
- http://pouzinsociety.org:80



- This makes mobility hard to achieve
- In RINA applications have names that are independent of the layers below (DIFs)
  - Application names can be structured in a way that makes sense for the application
  - The application name doesn't contain the semantics of where the application is in the network (i.e. what is its point of attachment to the layer below)

RINA over IP benefits: Next generation VPN

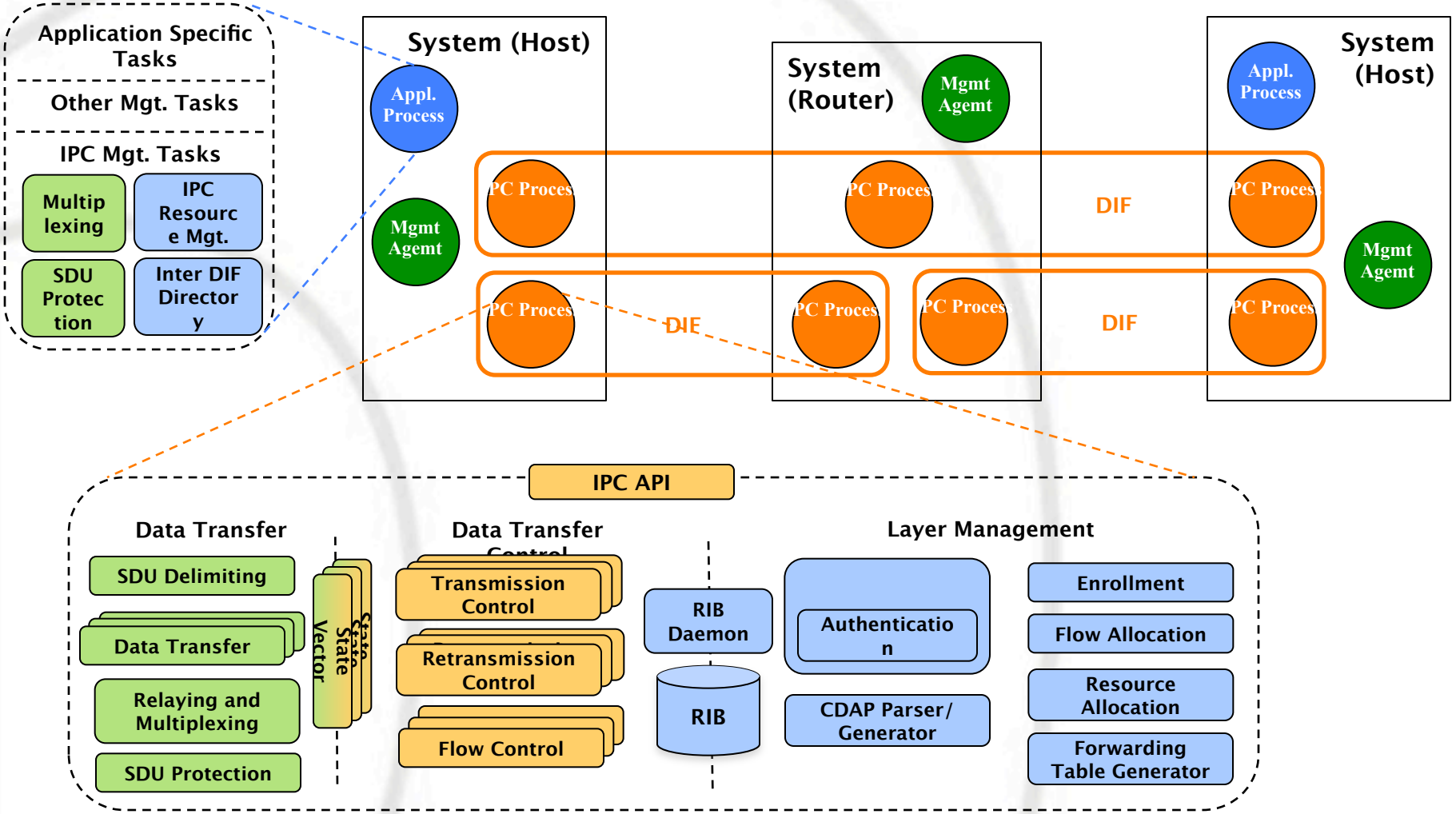
- DIFs are customizable VPNs that can span multiple IP networks.
  - Each DIF has its own addressing scheme, security mechanisms (authentication, authorization), routing strategy, resource allocation strategy (support for different levels of QoS), flow control strategy, data transfer/data transfer control, …
  - Processes (and not systems) are members of the DIFs (different processes can access different DIFs in each system). Processes may not have access to the whole range of DIFs available on their system
  - DIFs open the door to VPNs optimized for certain applications



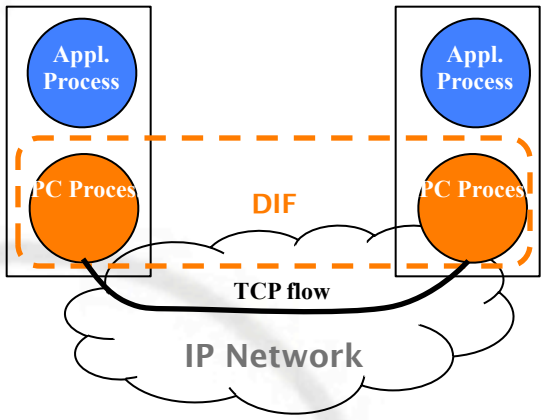
Outline

- RINA Adoption strategy and current prototype rationale
- **Prototype implementation phases**
- Prototype description
- Demo

Architectural model

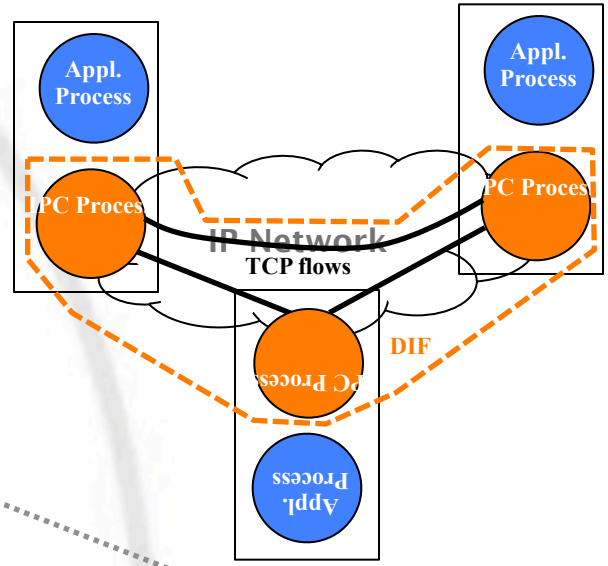


Implementation phases

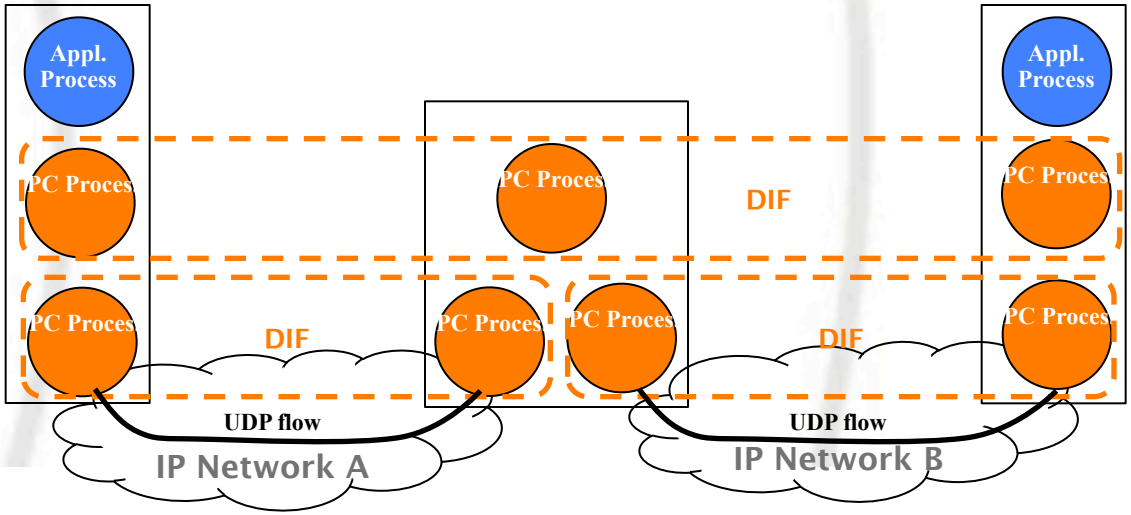


A) Two systems directly connected through a single IP Network (demo)

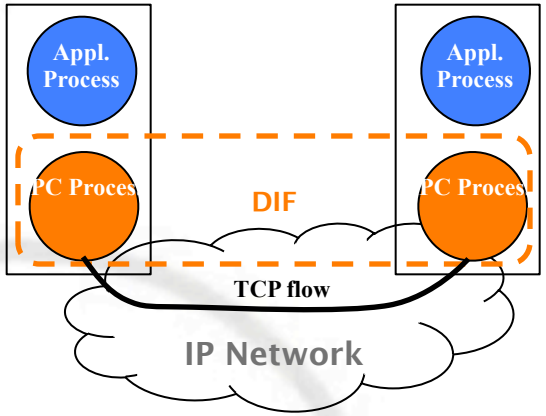
B) N systems directly connected systems through a single IP Network (in 1 month)



C) N systems connected through multiple IP networks (end of 3Q)

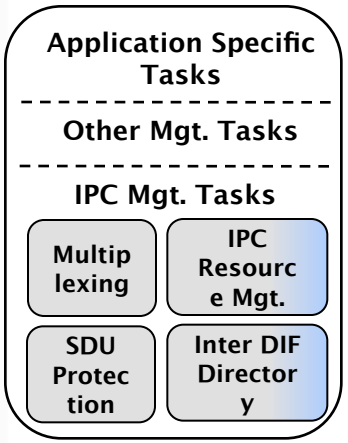


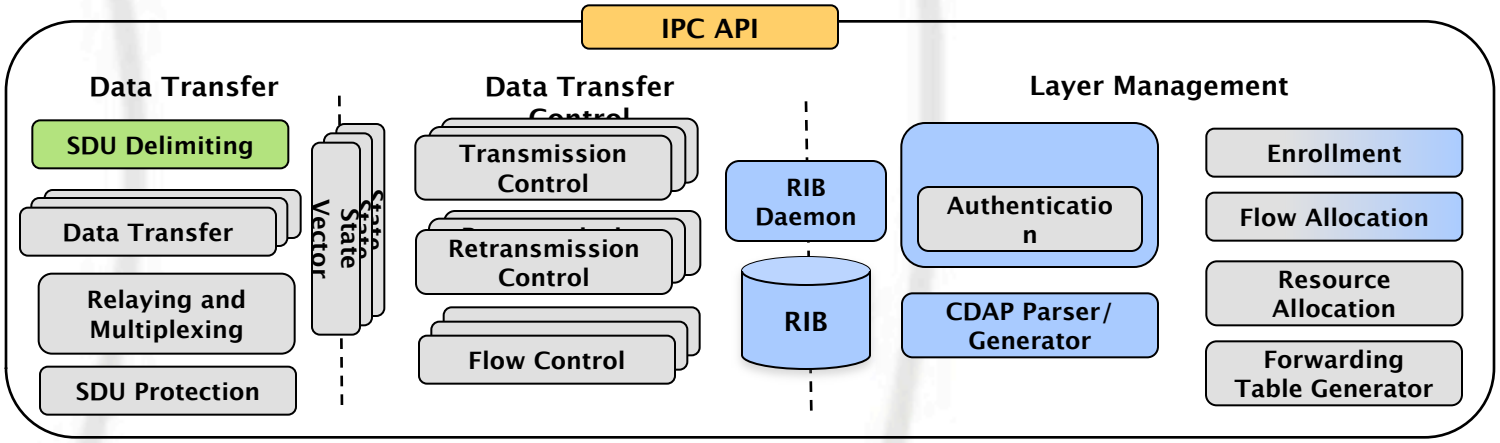
Phase I: 2 systems direct



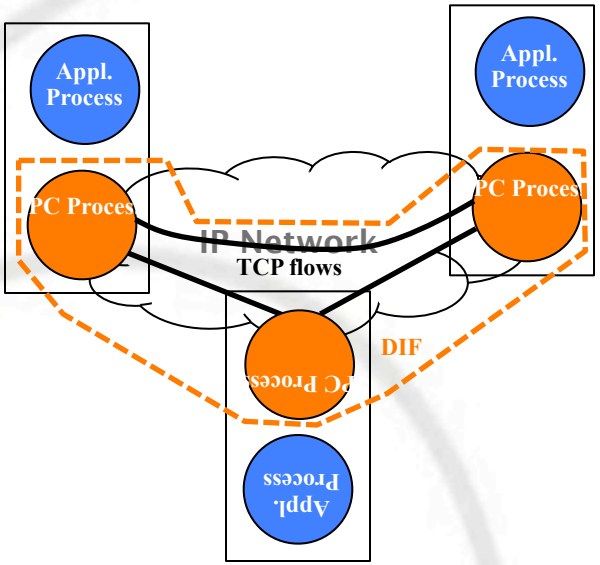
- The following IPC Process components have to be developed:
  - SDU Delimiting, RIB, RIB Daemon, CDAP, CACE, Enrollment, Flow Allocation, IPC API
  - Enrollment and Flow Allocation implemented

- The following IPC Management (OS) components have to be developed
  - Trivial IRM (create, destroy, list IPC Processes), Static IDD (config file)

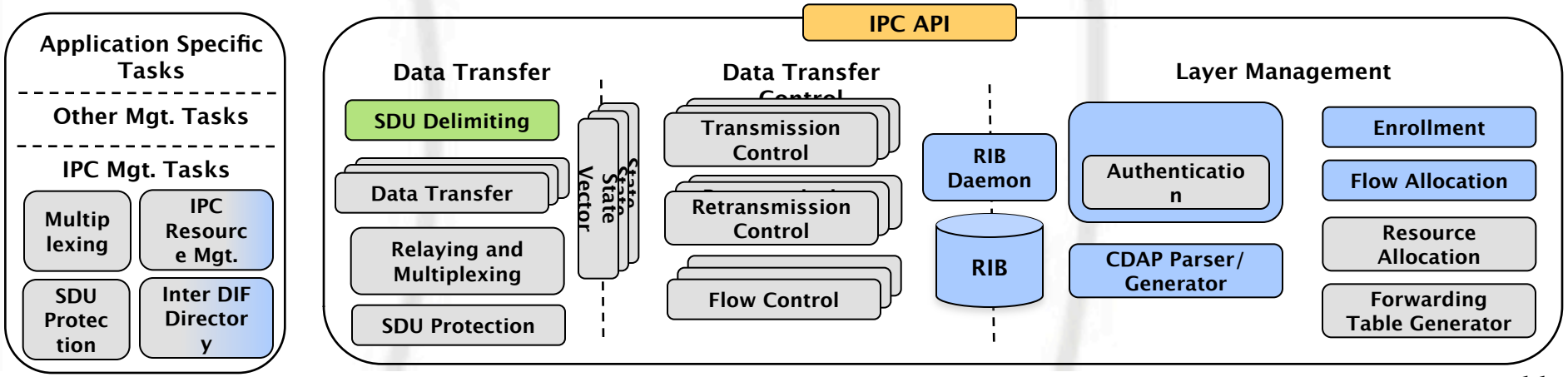




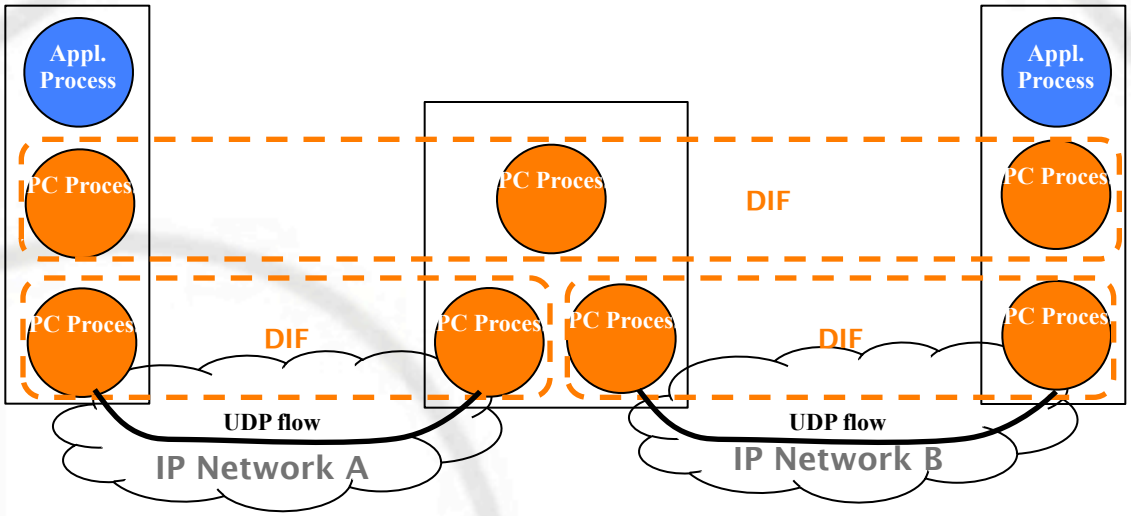
Phase 2: N Systems direct



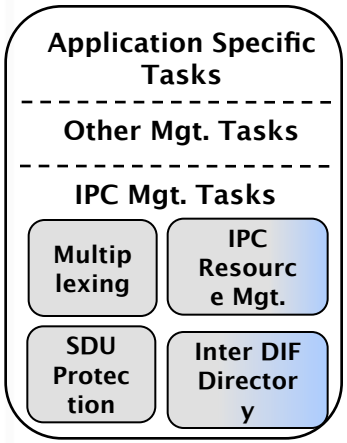
- Changes with respect to phase I:
  - Complete implementation of enrollment and flow allocation.
  - A dynamic, but very simple IDD (just suitable for small groups of DIFs)

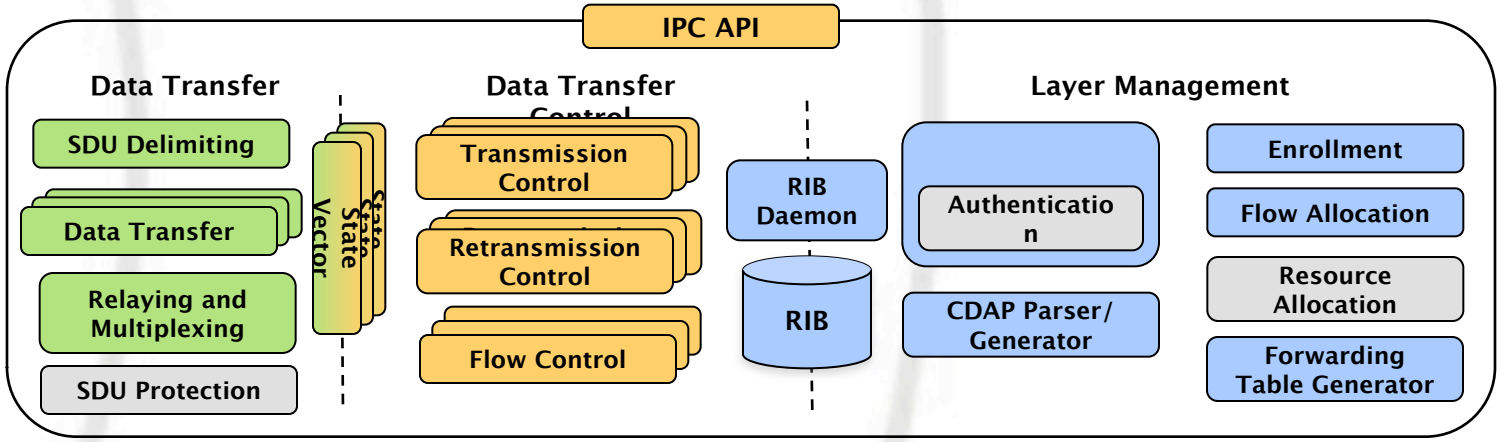


Phase 3: N systems not directly connected



- Changes with respect to phase II:
  - Implementation of EFCP (DTP and DTCP), the RMT and the routing component
  - Authentication, Resource allocation and SDU protection may be out of the scope of the initial prot.





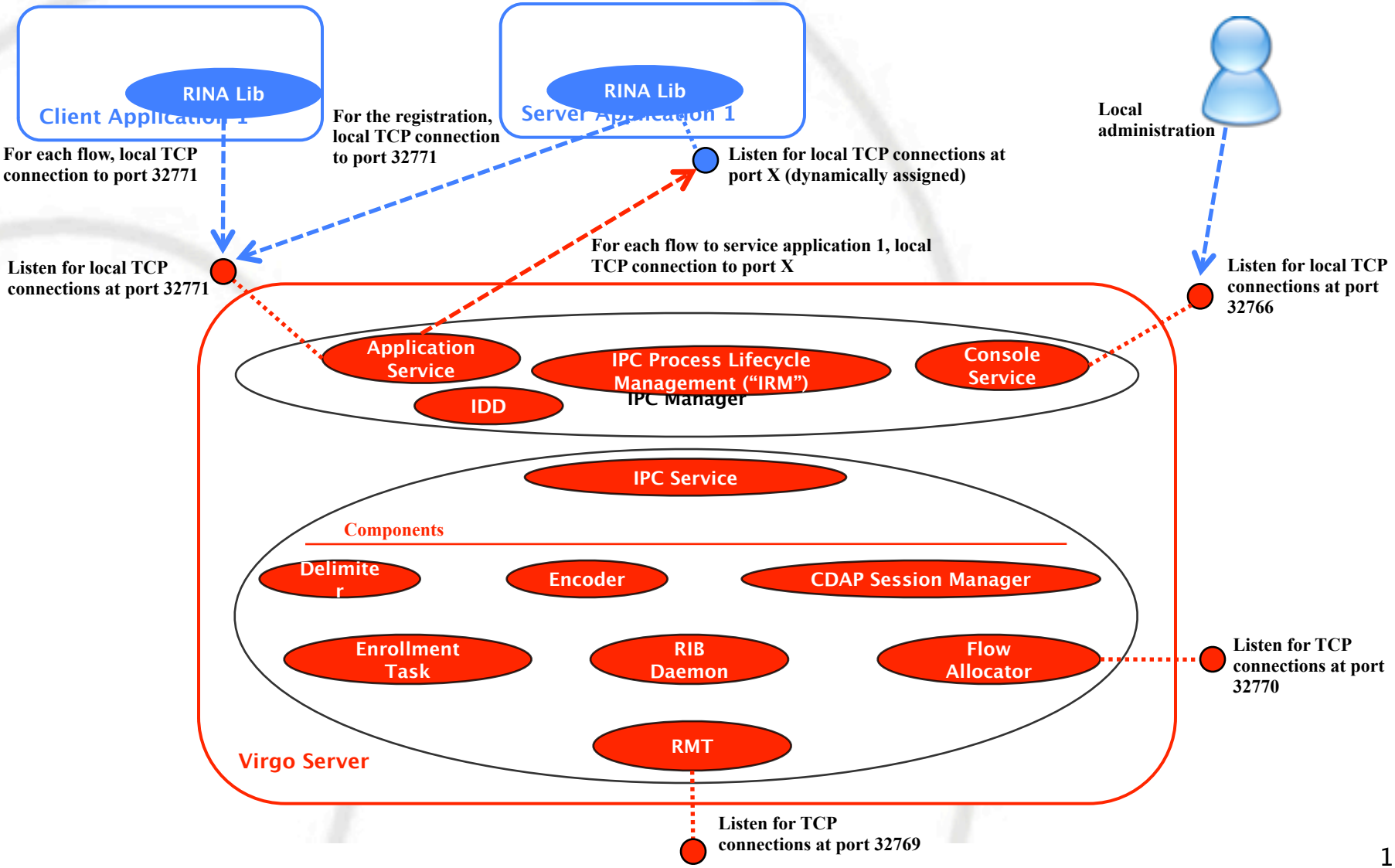
Outline

- RINA Adoption strategy and current prototype rationale
- Prototype implementation phases
- **Prototype description**
- Demo

Prototype description

- Implemented as part of the TINOS framework (a network protocol experimentation framework)
  - https://github.com/PouzinSociety/tinos
- Implemented in Java, using the OSGi technology (OSGi container provided by the Eclipse Virgo container)
  - OSGi is a component model that facilitates building modular Java applications
- Tested on Mac OS X and Linux Debian, but should be multi-platform (support all the platforms that Eclipse Virgo supports)
- Not yet fully integrated with TINOS (once it is, it will be possible to instantiate several "systems" within a single Java process, using XMPP as the underlying "physical substrate")

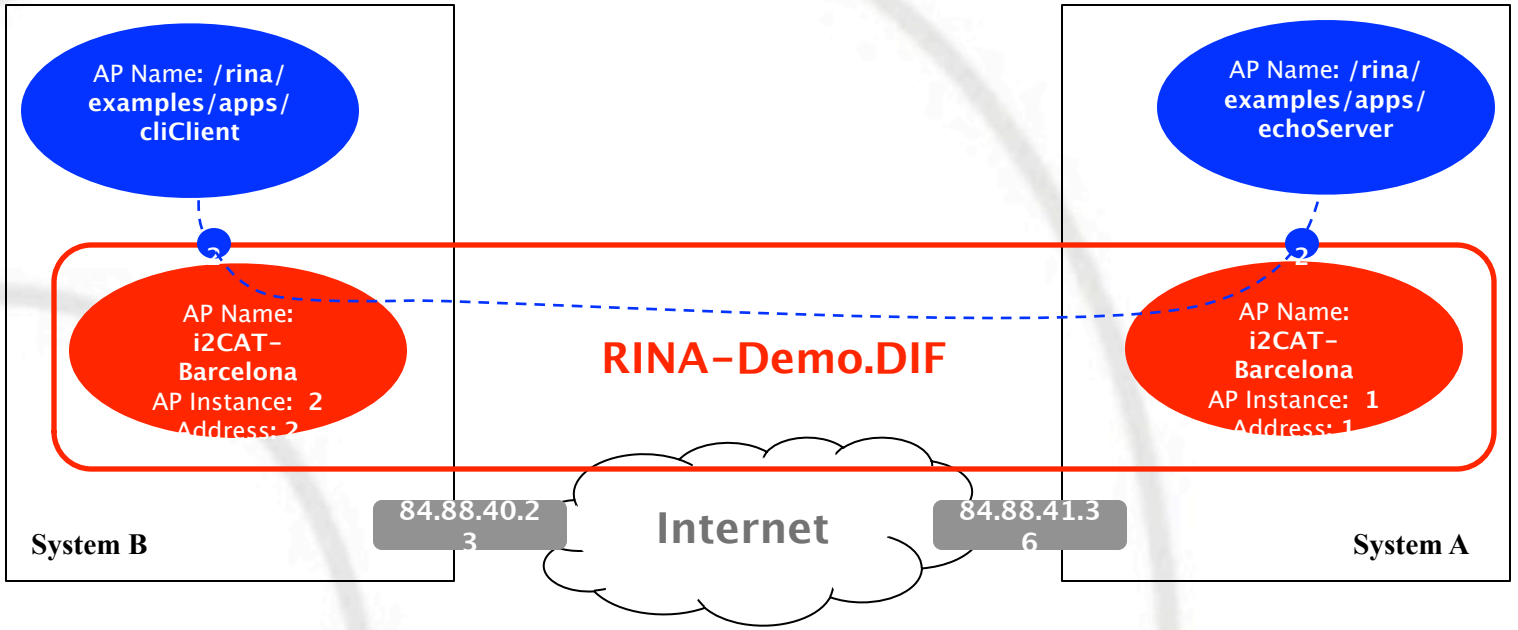
Current structure



Outline

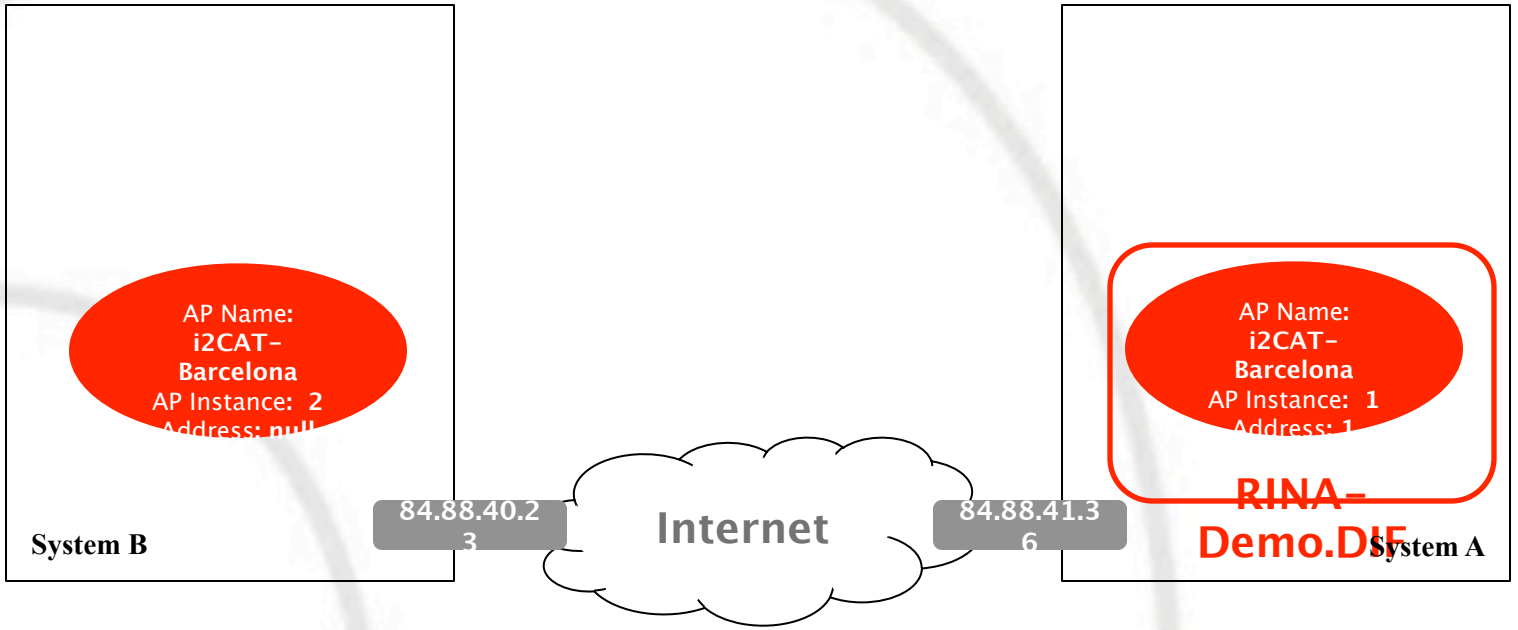
- RINA Adoption strategy and current prototype rationale
- Prototype implementation phases
- Prototype description
- **Demo**

Demo scenario



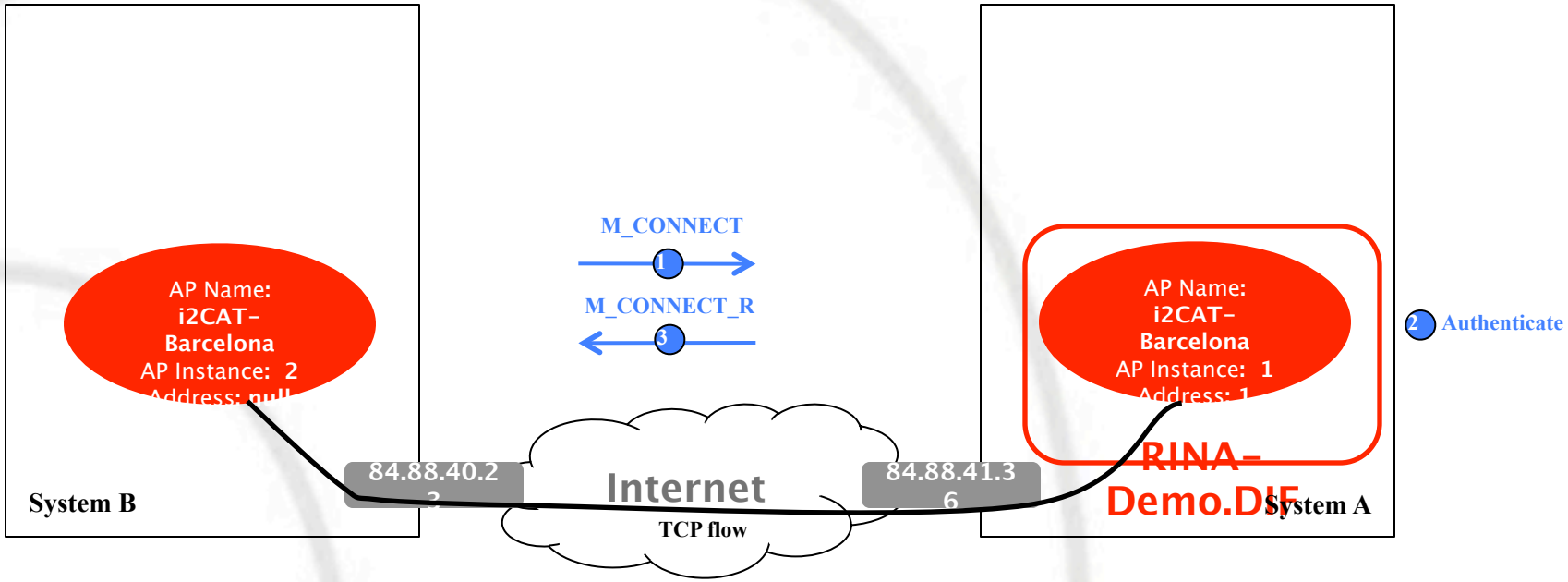
- Two systems, with one IPC process each (red circles).
- One system (left) is running the client application, the other system (right) an echo server application (echoes strings back).
- What works: IPC process creation, enrollment, flow allocation, application interfacing through native RINA API
  - Now have to verify interop with Steve and Peter's prototype, as well as with BU's prototype.

1) Instantiation of the IPC processes



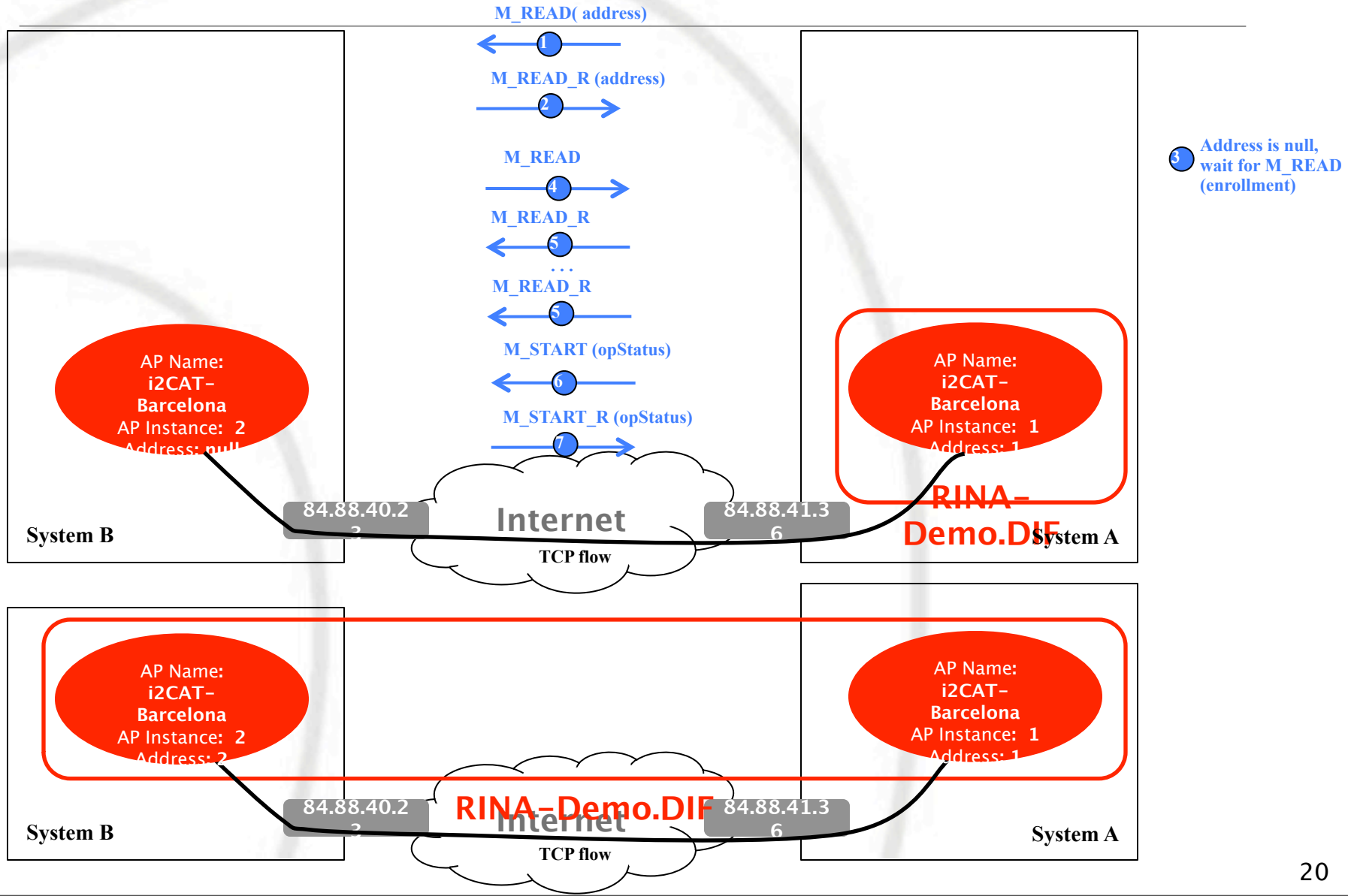
- SSH into system A, start the Virgo container, and connect to the management console. Then create IPC Process i2CAT-Barcelona 1 as the single member of the DIF RINA-Demo.DIF
- SSH into system B, start the Virgo container, and connect to the management console. Then create IPC Process i2CAT-Barcelona 2; not belonging to any DIF

2) Common application Connection Establishment

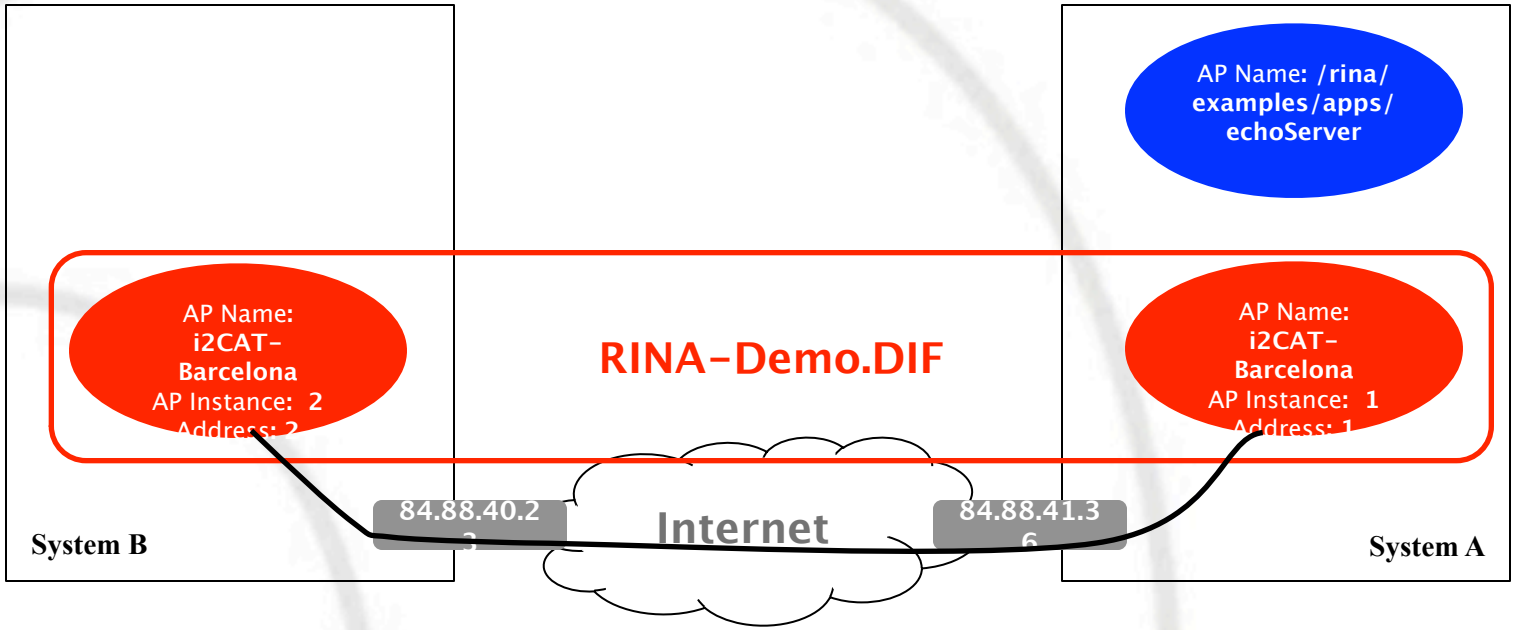


- SSH into system B, connect to the management console, and tell IPC Process i2CAT-Barcelona 2 to enroll to IPC Process i2CAT-Barcelona 1
  - First it will allocate a TCP flow to i2CAT-Barcelona 1, then it will go through CACE (Common application connection establishment) and finally through the enrollment program

3) Enrollment

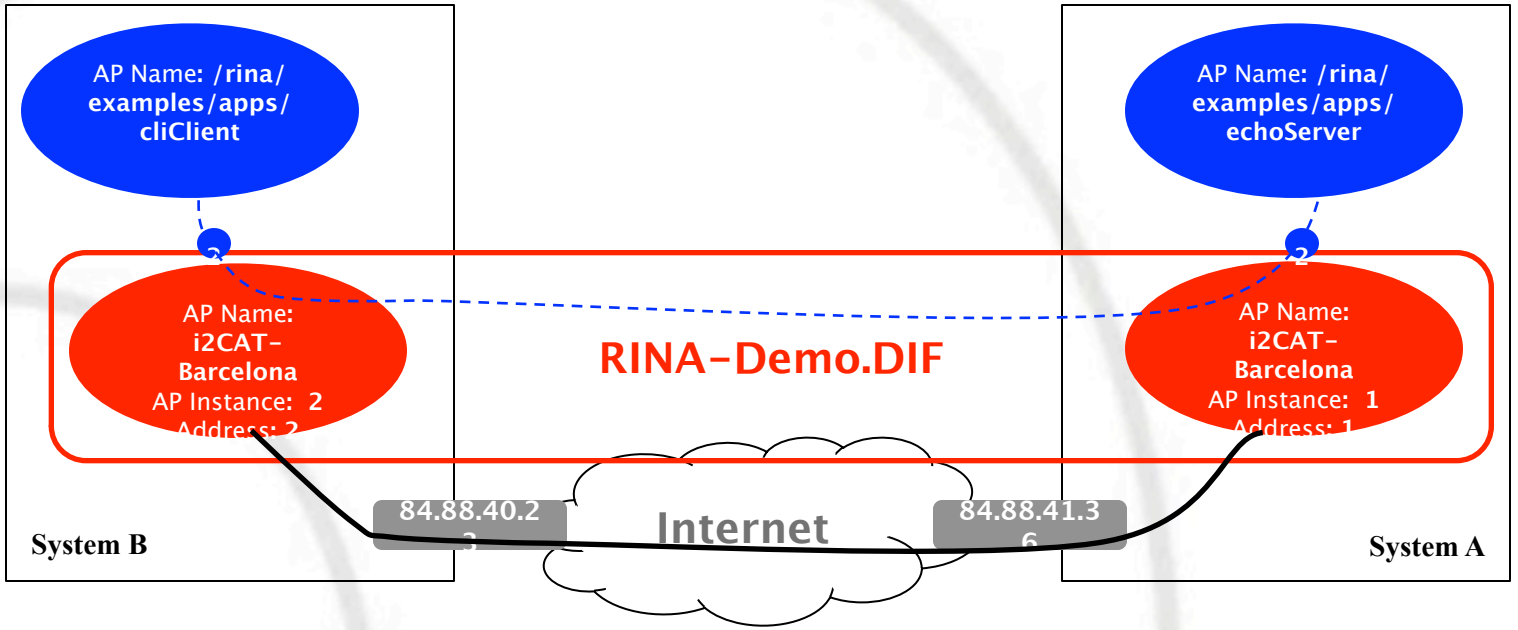


4) Instantiation of the Server application



- SSH into system A, start the Java echo server application. It will register to the RINA-Demo.DIF, making itself available through this DIF.

5) Instantiation of the client application, and allocation of flow



- SSH into system B, start the Java echo client application and tell it to connect to the echo server application. It requests a flow to the echo server application by name (/rina/examples/apps/ echoServer)

Thanks for your

More information about the prototype at

https://github.com/PouzinSociety/tinos/wiki/RINA-Prototype