

# Developing a RINA Prototype over IP using the TINOS framework

7<sup>th</sup> International Conference on Future Internet Technologies  
September 12<sup>th</sup>, 2012

**PSOC**  
pouzin society

**i2cat**  
FUNDACIÓ

**D**istributed  
**A**pplications  
**N**etworks  
**A**rea

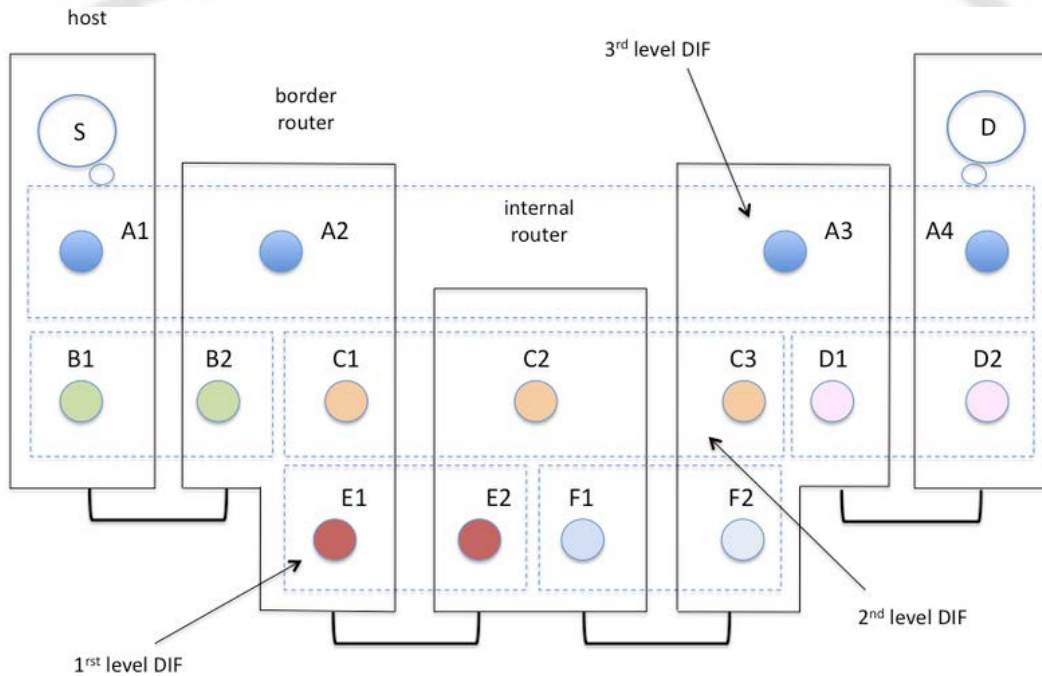
Eduard Grasa  
Research Manager @ DANA  
Fundació i2CAT

# Outline

---

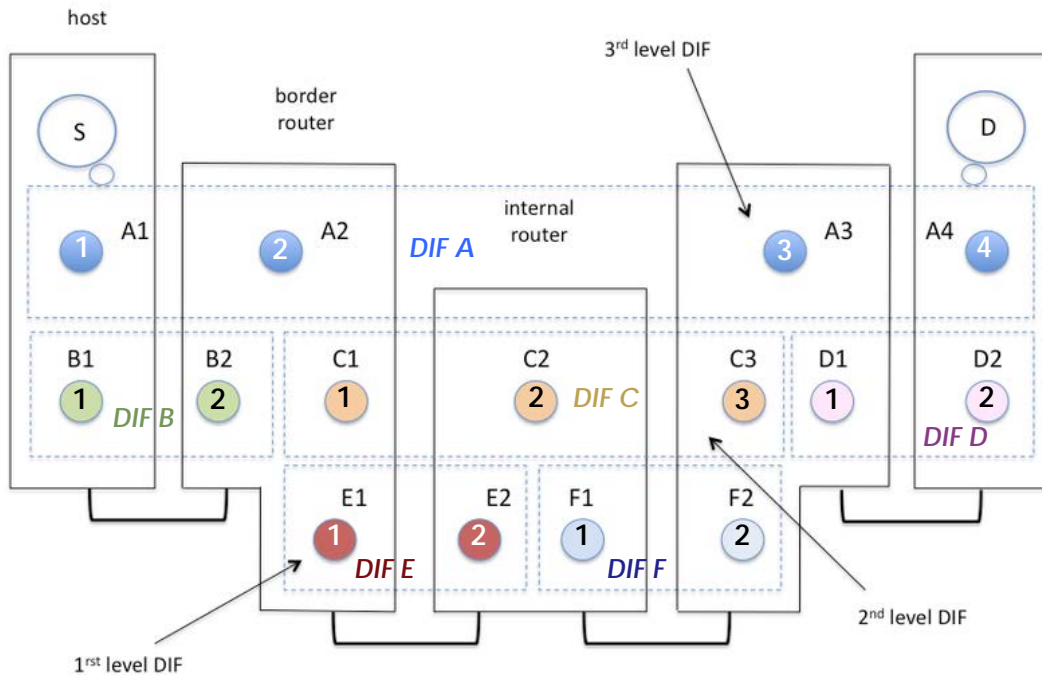
- **Quick introduction to RINA**
- RINA Adoption strategy and current prototype rationale
- Prototype components
- Implementation platform
- Next prototyping steps: FP7 IRATI

# RINA Architecture



- A structure made of recursive layers that provide IPC (**Inter Process Communication**) services to applications on top
- There's a single type of layer that repeats as many times as required by the network designer
- Separation of mechanism from policy
- All layers have the same functions, with different scope and range.
  - Not all instances of layers may need all functions, but don't need more.
- A Layer is a Distributed Application that performs and manages IPC.
  - A Distributed IPC Facility (DIF)
- This yields a theory and an architecture that scales indefinitely,
  - i.e. any bounds imposed are not a property of the architecture itself.

# Naming and addressing in RINA



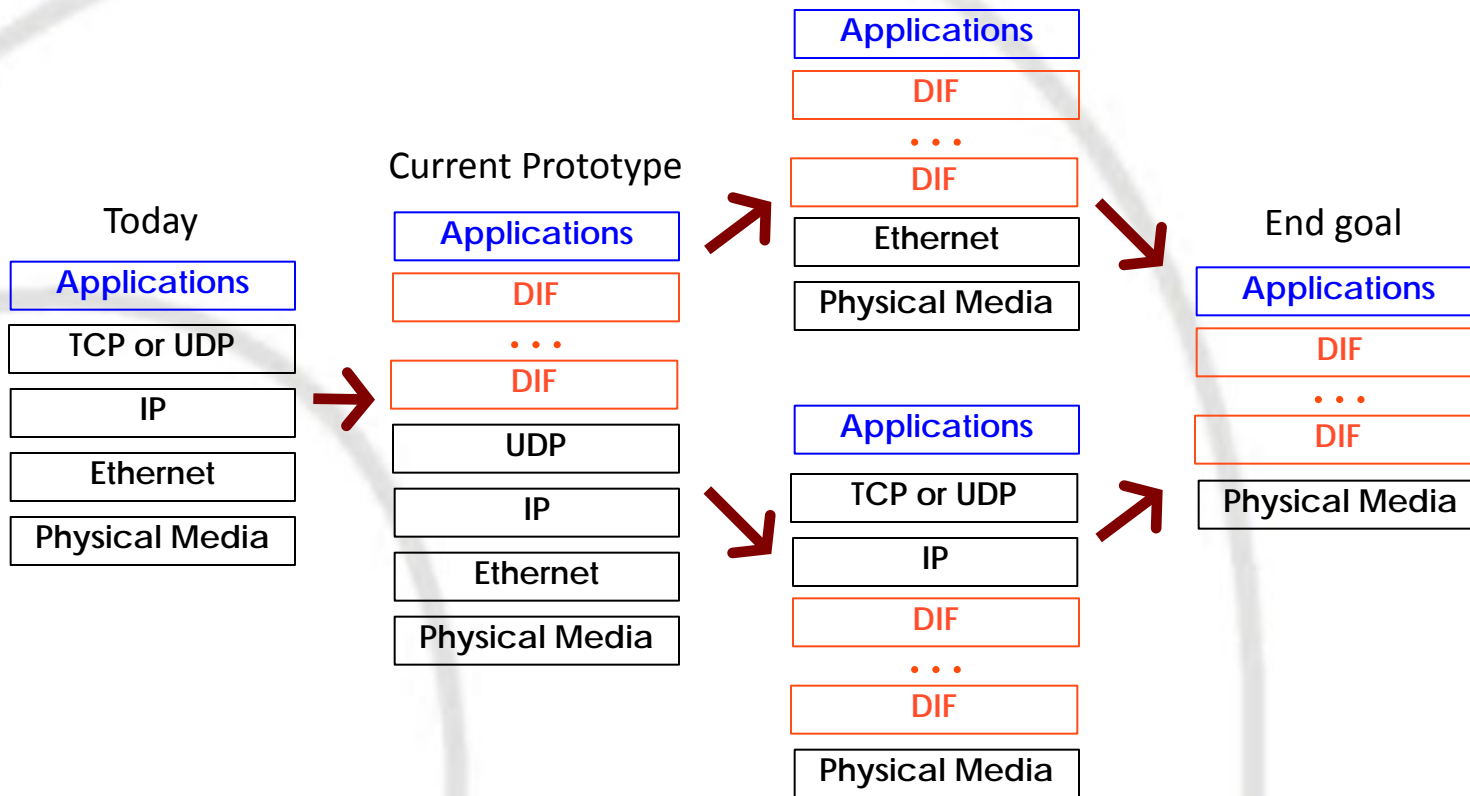
- All application processes (including IPC processes) have a name that uniquely identifies them within the application process namespace.
  - In order to facilitate its operation within a DIF, each IPC process within a DIF gets a synonym that may have topological significance within the DIF (i.e. an address).
- The scope of an address is the DIF, addresses are not visible outside of the DIF.
  - Each DIF has a directory that maps destination Application process names to DIF IPC Process addresses.
  - Because the architecture is recursive, applications, nodes and PoAs are relative
    - For a given DIF of rank N, the process at the layer N+1 is an application and the process at the layer N-1 is a Point of Attachment.

# Outline

---

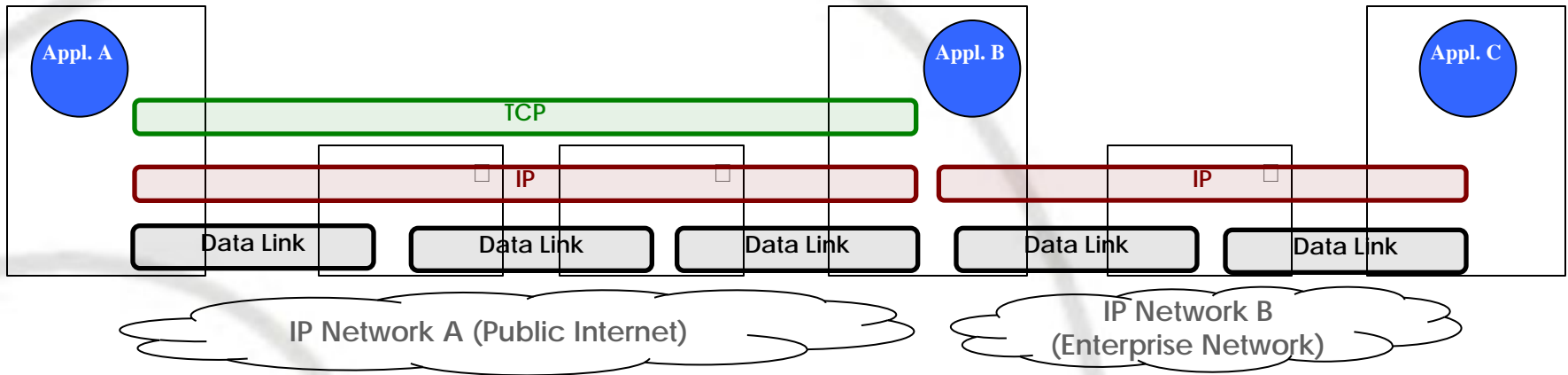
- Quick introduction to RINA
- **RINA Adoption strategy and current prototype rationale**
- Prototype components
- Implementation platform
- Next prototyping steps: FP7 IRATI

# RINA Adoption strategy

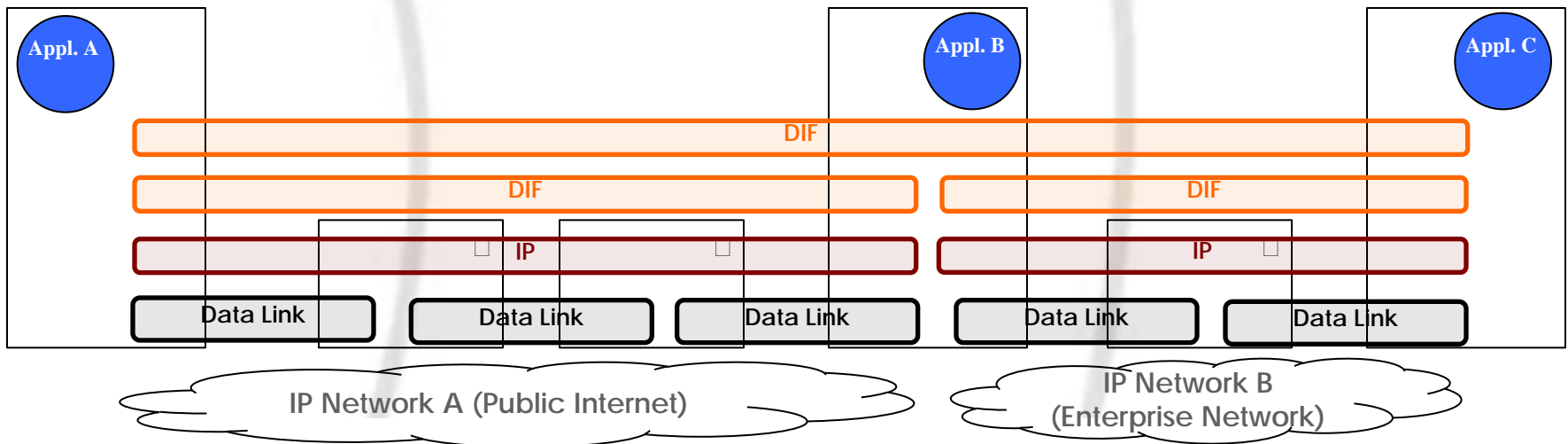


- Start as an overlay to IP, validate technology, work on initial concepts, develop DIF machinery.
  - Useful by itself: internetwork layer(s), decouple application from infrastructure, improved application API, support for multi-homing and mobility.

# RINA over IP benefits: Internetwork layer(s)

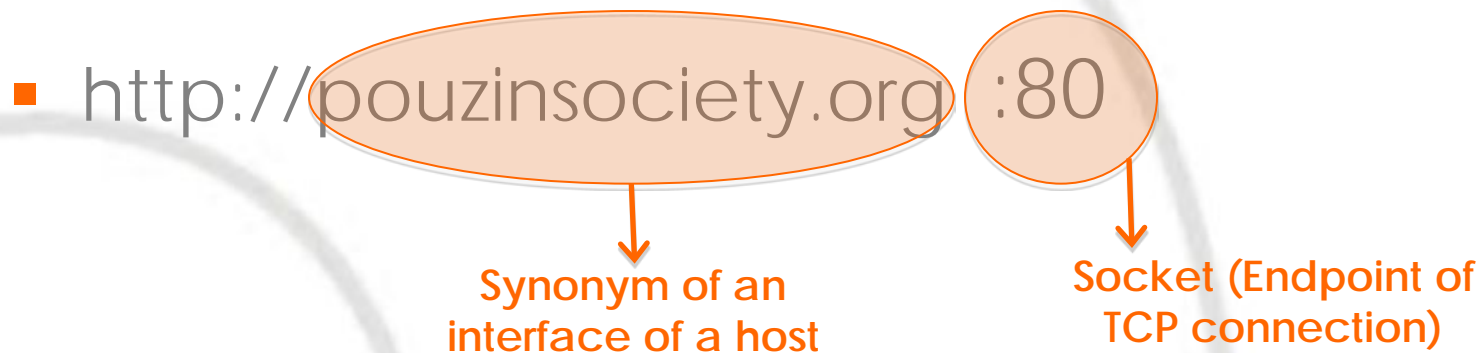


- What if application A wants to communicate with Application C?
  - It cannot do it, unless you start deploying middleboxes like NATs, application-layer gateways, ... The architecture doesn't accommodate internetworking!



# RINA over IP benefits: Separate applications from infrastructure

- The current application namespace is tied to IP addressing and TCP/UDP port numbers:

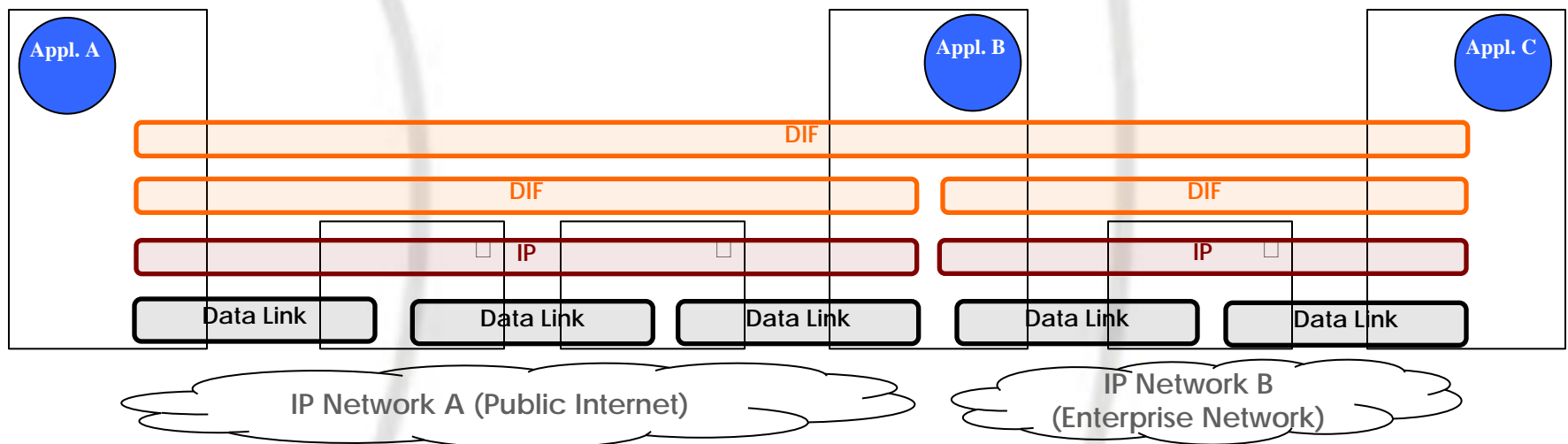


- This makes mobility hard to achieve
- In RINA applications have names that are independent of the layers below (DIFs)
  - Application names can be structured in a way that makes sense for the application
  - The application name doesn't contain the semantics of where the application is in the network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

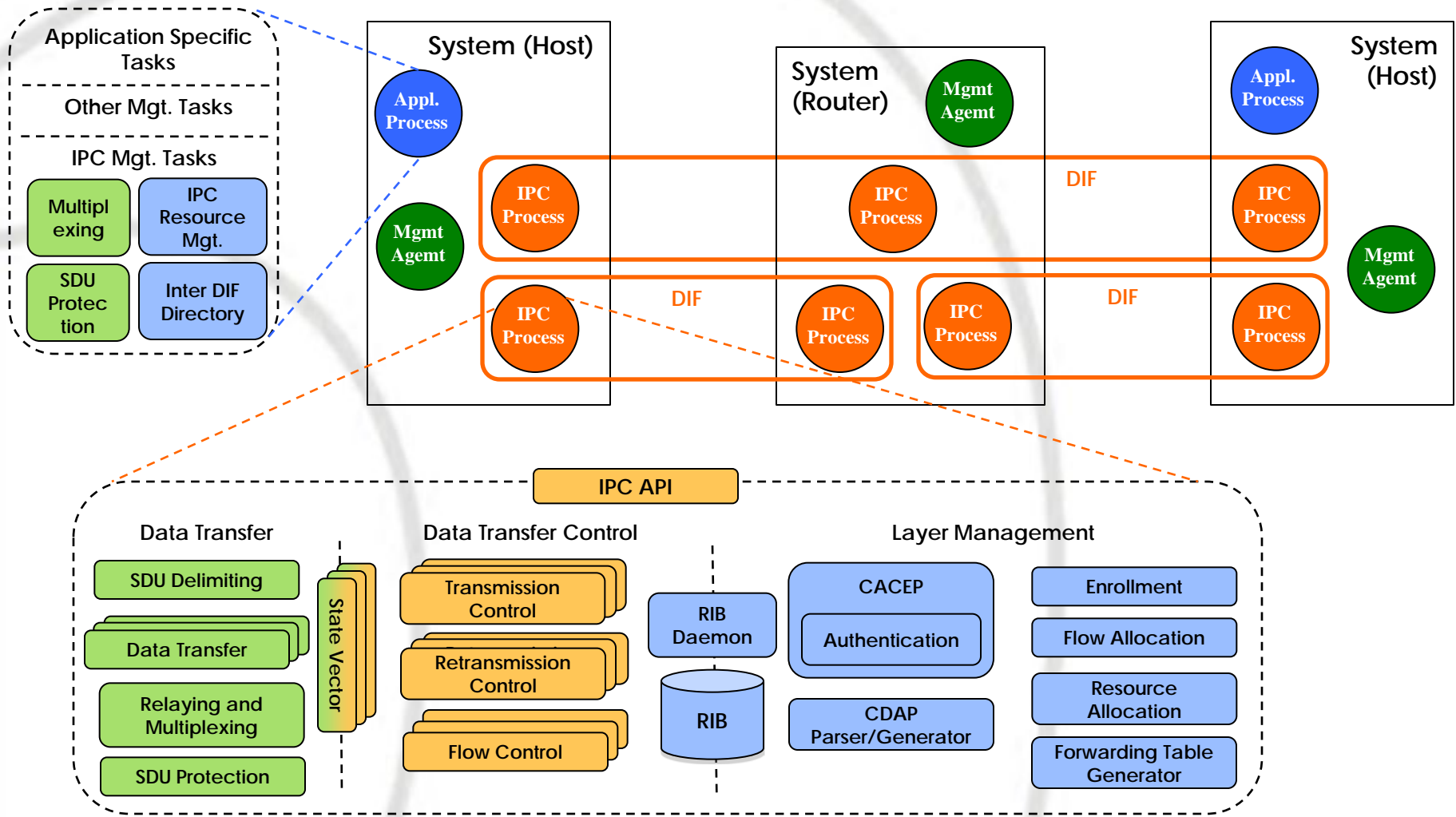


# Outline

---

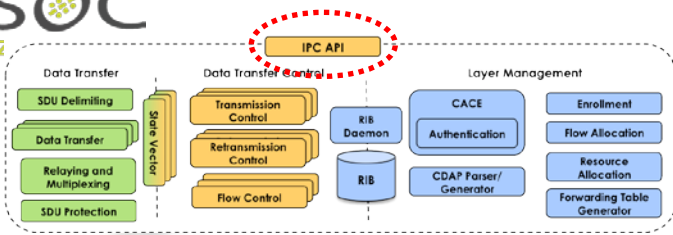
- Quick introduction to RINA and PSOC
- RINA Adoption strategy and current prototype rationale
- **Prototype components**
- Implementation platform
- Next prototyping steps: FP7 IRATI

# Architectural model

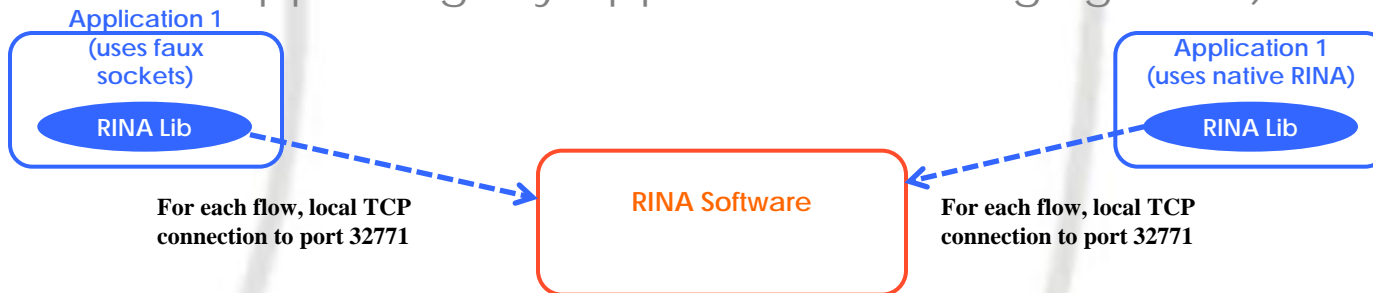


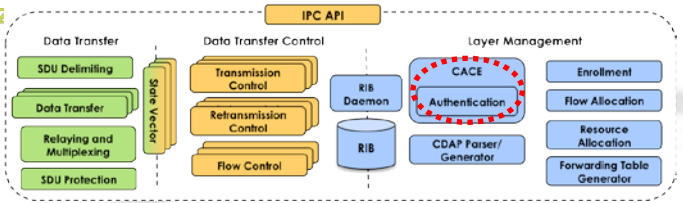
Increasing timescale (functions performed less often) and complexity →

# The IPC API



- Presents the service provided by a DIF: *a communication flow between applications, with certain quality attributes.*
- 6 operations:
  - `portId_allocateFlow(destAppName, List<QoSParams>)`
  - `void _write(portId, sdu)`
  - `sdu_read(portId)`
  - `void _deallocate(portId)`
  - `void _registerApp(appName, List<difName>)`
  - `void _unregisterApp(appName, List<difName>)`
- Implementation: Native RINA API and 'Faux Sockets' API for Java apps, (the latter to support legacy apps without changing them)

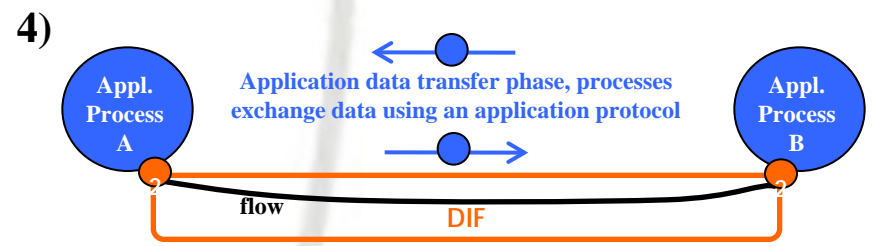
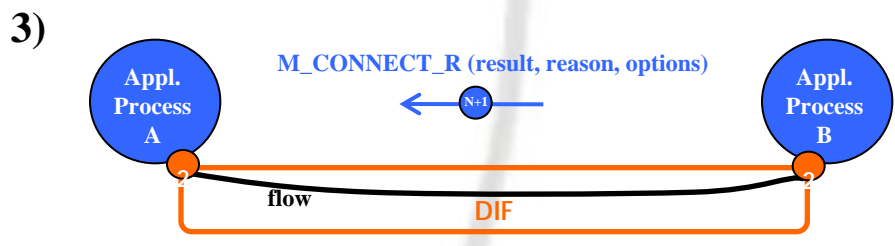
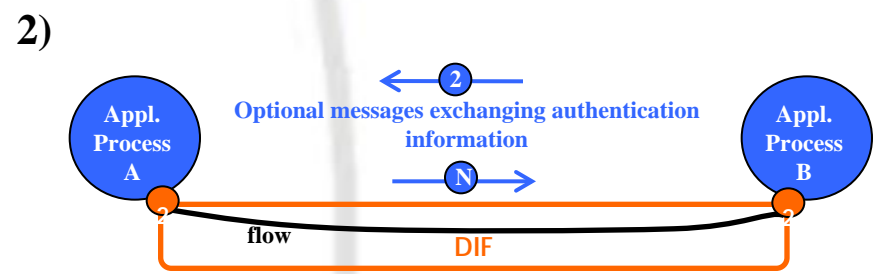
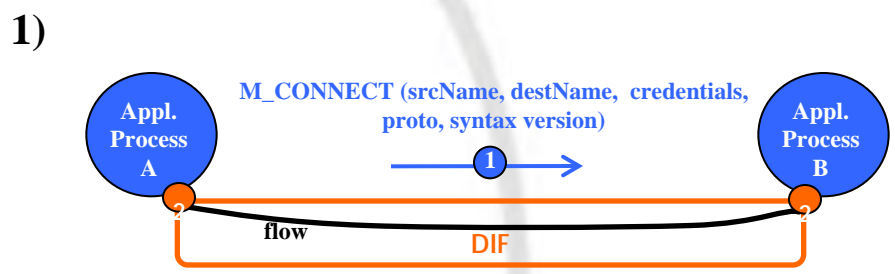


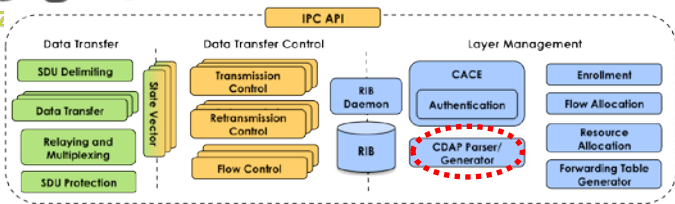


# CACEP

Common Application Connection Establishment Phase

- IPC Processes are just application processes. Once they have a communication flow between them, they have to set up **an application connection** before being able to exchange any further information.
- The application connection allows the two communicating apps to:
  - Exchange naming information with its apposite, optionally **authenticating** it
  - Agree on an application protocol and/or syntax version for the application data exchange phase

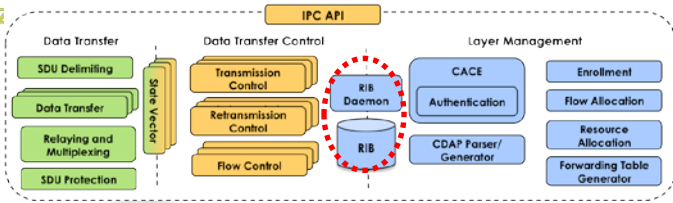




# CDAP

## Common Distributed Application Protocol

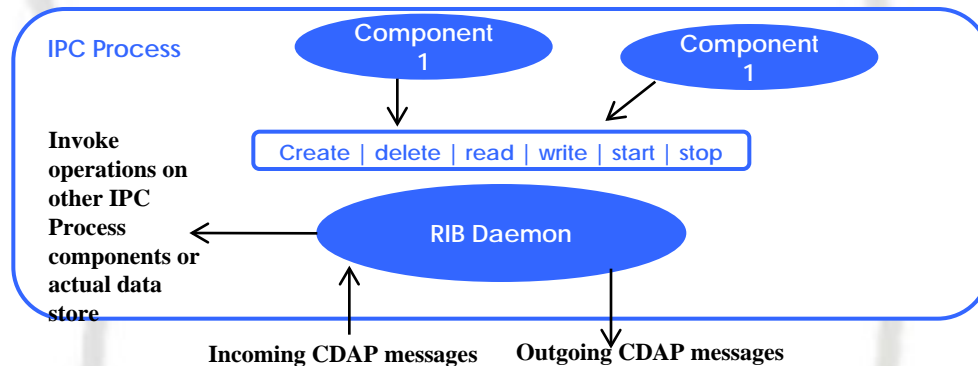
- CDAP is RINA's application protocol, modeled after ACSE and CMIP.
  - CDAP includes CACEP, so it is used both for the application connection establishment and the application data transfer phases.
- CDAP is object oriented, providing 6 primitives to perform operations on objects: create/delete, read/write, start/stop.
  - All the objects have the following attributes: *class*, *name*, *instance* and *value*.
  - The *scope/filter* options allow CDAP messages to be applied to multiple objects at once.
- Implementation:
  - CDAP abstract syntax defines the contents of the messages that can be used by the protocol: connect, release (CACEP) create, delete, read, write, start, stop and its associated response messages.
  - There can be many concrete encodings for a given CDAP abstract syntax, right now we're using **Google Protocol Buffers (GPB)** – many others are possible (JSON, ASN.1, XML, ...). Reasons for GPB:
    - Provides efficient encoding (bitsize and time to parse/generate)
    - Being used in production by Google in massive scaled distributed systems
    - Free, open source tools for developers in many programming languages



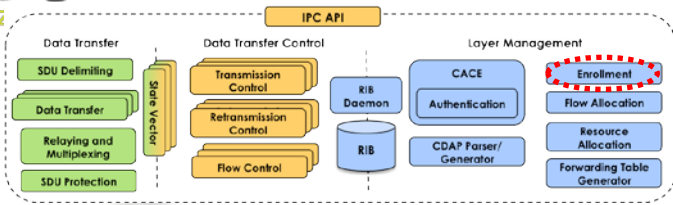
# RIB and RIB Daemon

## Resource Information Base

- The RIB is a **logical representation** of the information known by an application process.
  - Doesn't need to be a database, the information may be stored in the different application Process components.
- The RIB Daemon provides **an API to perform operations on the RIB** (both objects in the local RIB and objects in remote application processes' RIBs).
  - **Transition from an IPC model to a programming model** for the IPC Process Components



- Implementation:
  - The current RIB schema is a tree of the objectNames. The RIB is implemented as a hashtable indexed by objectName
    - As a simplification objectName is assumed to be unique

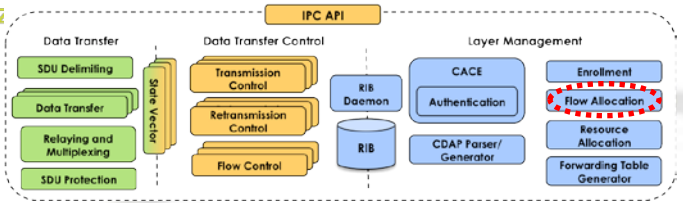


# Enrollment

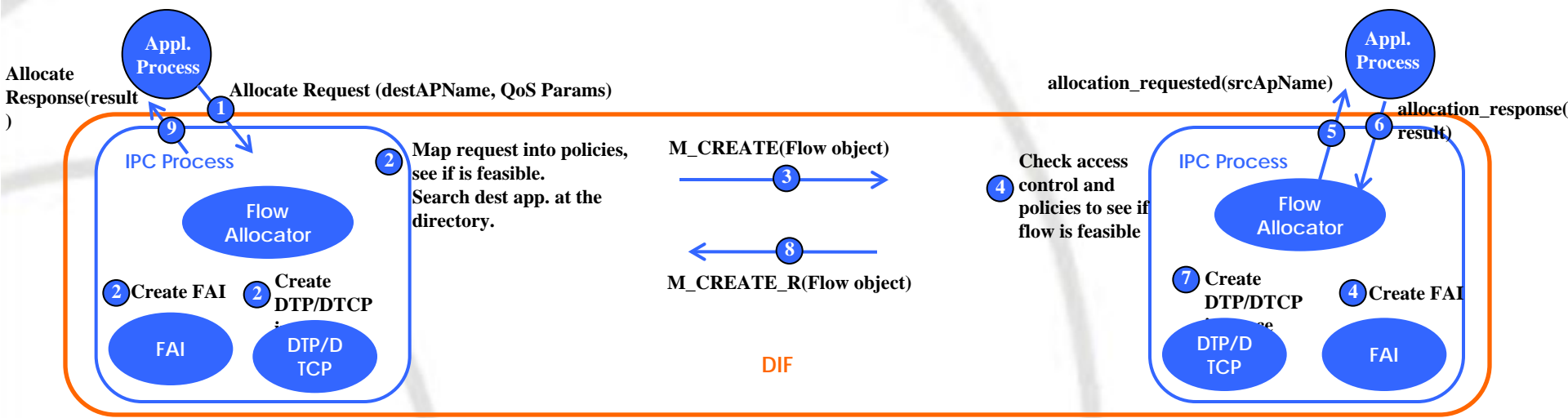
- Enrollment allows an IPC Process to *get enough state information to become an operating member of the DIF*.
  - It occurs when an IPC Process that wants to join a DIF has established an application connection to an IPC Process that is a member of the DIF.
- Implementation:
  - The current enrollment program defines the exchange of objects required to initialize a new IPC Process as a member of the DIF, including:
    - Detecting if an address is stale (or null) and assigning a valid one to the new member (valid addresses are currently preconfigured, will be automated in the future)
    - Provide information about the QoS classes supported by the DIF
    - Information on the policy sets supported by the DIF
    - Information on the DIF data transfer constants, such as max PDU size or the DIF's MPL



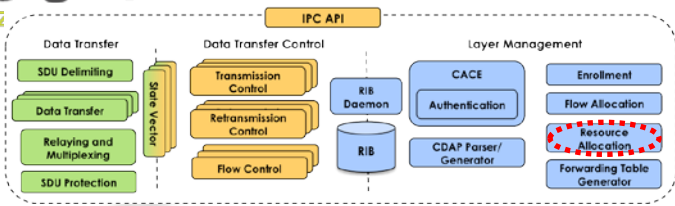
# Flow Allocation



- Manages a flow's lifecycle: *allocation*, *monitoring* and *deallocation*.



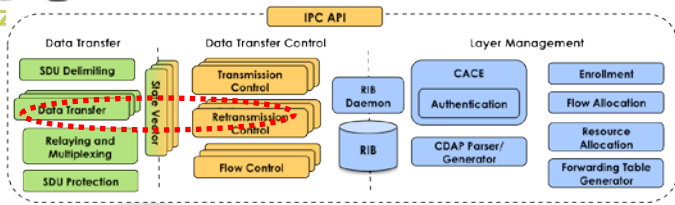
- Note that Flow allocation is separated from data transfer (there can be more than one DTP/DTCP instance associated to the same flow at the same time) – implication of Watson's results
- Implementation:
  - Simple distributed directory that implements a broadcast strategy to disseminate directory updates
    - Ok for initial experimentation on small networks, use more sophisticated approaches in the near future



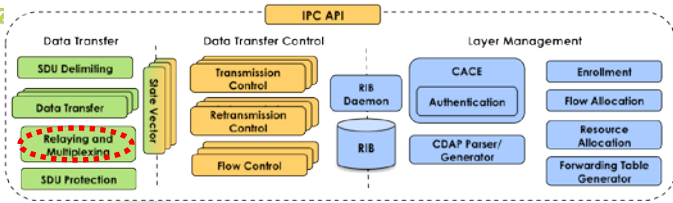
# Resource Allocation

- The component that *decides how the resources in the IPC Process are allocated.*
  - Dimensioning of the queues, creation/suspension/deletion of RMT queues, creation/deletion of N-1 flows, and others
- Implementation:
  - Only the component that manages the creation/deletion of N-1 flows is implemented.
    - Management is in charge of imposing the policies of when to create/delete N-1 flows; management being an external DMS (NMS), the OS where the IPC Process is executing or an autonomic management function in the IPC Process.
    - Currently the number of N-1 flows to be created is a static policy specified in a configuration file.

# Data Transfer and Data Transfer Control



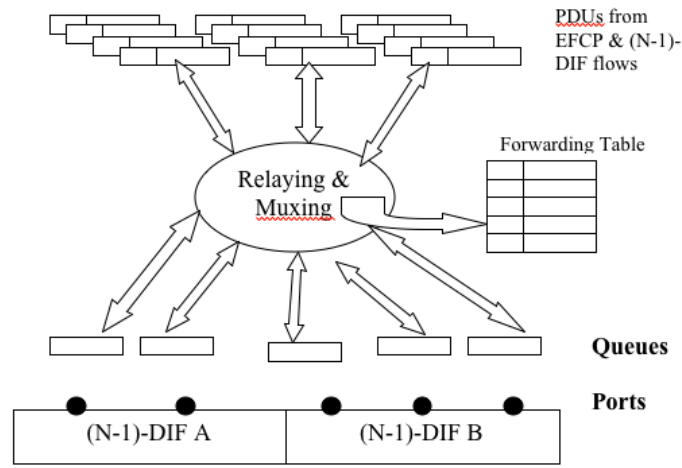
- The RINA data transfer protocol (EFCP) is based on Watson's Delta-T
  - Showed that the necessary and sufficient conditions for reliable synchronization are to bound 3 timers: MPL, Maximum time to ACK and Maximum Time to Retransmit.
    - In other words, explicit connection establishment or teardown, such as in TCP, is not required.
- EFCP has 2 parts: Data Transfer (DTP) and Data Transfer Control (DTCP), loosely coupled through a state vector
  - DTP performs the mechanisms that are tightly coupled to the transported SDU: fragmentation, reassembly, sequencing, addressing, concatenation, separation.
  - DTCP performs the mechanisms that are loosely coupled: transmission control, retransmission control and flow control.
- Implementation:
  - An initial version of DTP that performs sequencing and addressing has been implemented.



# RMT

## Relaying and Multiplexing Task

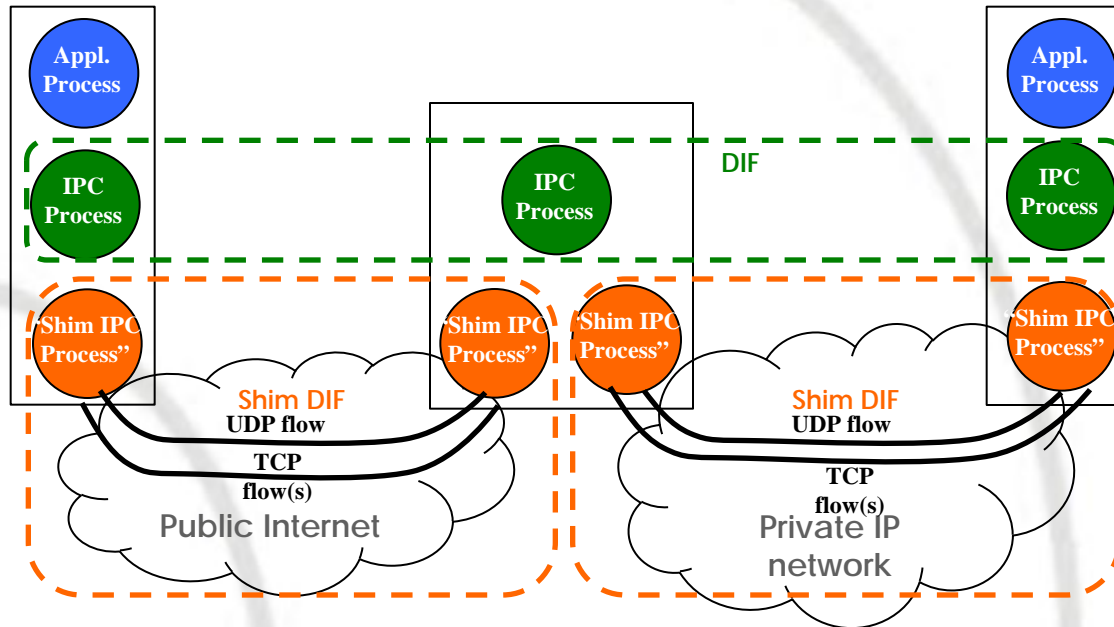
- The RMT multiplexes the PDUs coming from the N-flows originated at the IPC Process into one or more N-1 flows (in its multiplexing role) and forwards the incoming PDUs to another IPC Process through the adequate N-1 flow based on the PDU's destination address (in its relaying role).



© John Day, All Rights Reserved, 2011

- Implementation:
  - A simple RMT has been implemented, serving N FIFO queues in the order that the PDUs arrive at each different queue
    - Lots of room for improvement with different scheduling algorithms

# The Shim DIF



- The “shim IPC Process” for IP networks is not a “real IPC Process”. It just presents an IP network as if it was a regular DIF.
  - Wraps the IP network with the DIF interface.
  - Maps the names of the IPC Processes of the layer above to IP addresses in the IP network.
  - Creates TCP and/or UDP flows based on the QoS requested by an “allocate request”.

# Outline

---

- Quick introduction to RINA and PSOC
- RINA Adoption strategy and current prototype rationale
- Prototype components
- **Implementation platform**
- Next prototyping steps: FP7 IRATI

# Implementation platform

---

- 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”)

# Outline

---

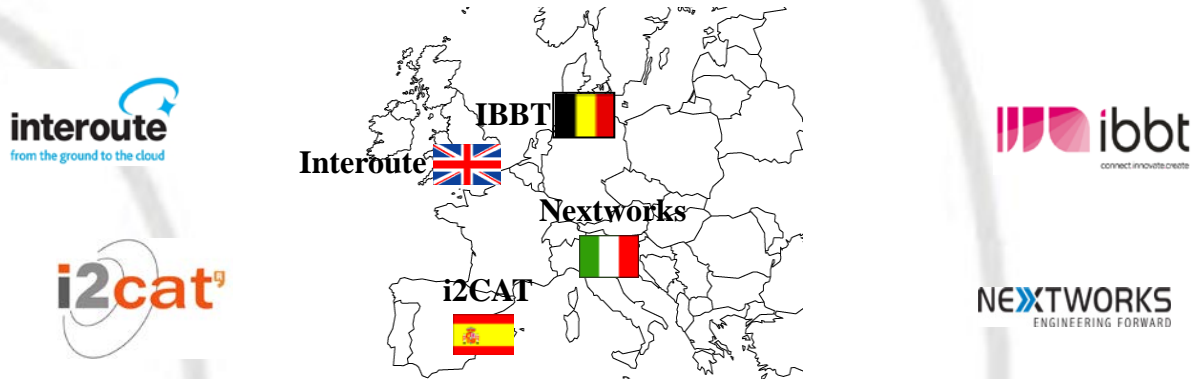
- Quick introduction to RINA and PSOC
- RINA Adoption strategy and current prototype rationale
- Prototype components
- Implementation platform
- **Next prototyping steps: FP7 IRATI**








- What? Main goal

- To **advance** the **state of the art of RINA** towards an **architecture reference model** and **specifications** that are **closer** to enable implementations deployable in **production scenarios**. The **design and implementation** of a **RINA prototype on top of Ethernet** will enable the **experimentation** and **evaluation of RINA in comparison to TCP/IP**.

- Who? 4 partners



- How? Requested 870.000 € funding to the EC to perform 5 activities

- WP1: Project management 
- WP2: Architecture, Use cases and Requirements 
- WP3: Software Design and Implementation 
- WP4: Deployment into OFELIA testbed, Experimentation and Validation 
- WP5: Dissemination, Standardisation and Exploitation 

# Thanks for your attention!



You can contact me at  
[eduard.grasa@i2cat.net](mailto:eduard.grasa@i2cat.net)

More information about RINA at <http://rina.tssg.org>,  
<http://pouzinsociety.org>, <http://csr.bu.edu/rina>

More information about the prototype at  
<https://github.com/PouzinSociety/tinos/wiki/RINA-Prototype>

More information about IRATI at  
<http://irati.eu>