The Pouzin Society

Welcome to the RINAissance!

An Introduction to the RINA Architecture Part I

IRATI RINA Workshop

John Day

Dublin 2014

In a network of devices why would
we route between processes?
- Toni Stoey, RRG 2009

© John Day, 2014
All Rights Reserved

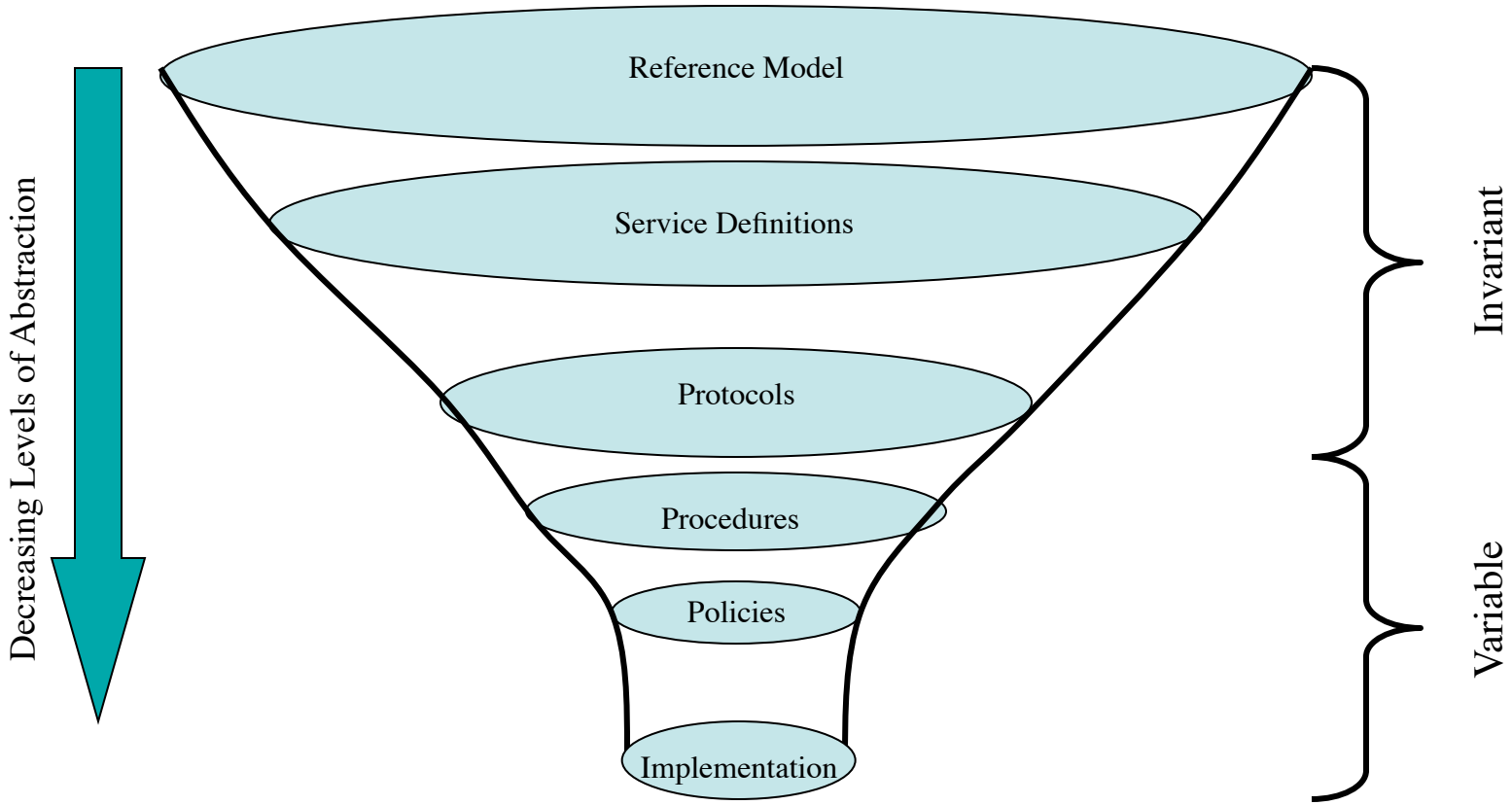
IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

Levels of Abstraction

(Abstraction is Invariance)



- Maximize the Invariant and minimize the discontinuties.
- Today there are essentially no levels of abstraction.

© John Day, 2014
All Rights Reserved

Unlike the Internet

- In RINA, most layers are private. This means that there can be much greater variability in them.
  - Here, there is much greater freedom and independence, but with the same code base.
- The Reference Model, Service Definition, and Protocol Specifications can be combined with off-the-shelf or custom procedures and policies to specify a DIF.
- The concept of a public network is meaningless in RINA.
  - There *may* be a public *layer*, but a public network is a *non-sequitor*.
- Groups may agree on common DIF definitions and policy sets.
  - The focus is not on designing protocols, but on defining policies and configuring DIFs.
- Maximizing commonality reduces operating and equipment costs and improves management.
- Our goal is to make networks simpler, more predictable, more manageable, in other words, more understanding.

© John Day, 2014
All Rights Reserved

What Do We Mean by Invariance?

- An Easy Example: Separating Mechanism and Policy
  - Concept first proposed by Bill Wulf [1975] for operating systems.
    - Separating what is common across solutions from what is uncommon.
    - Mechanics of memory management are the same, allocation scheme or page replacement policy is what varies.
- In protocols, there are a few mechanisms. Virtually all of the variation is in the policies.
  - Acknowledgement is mechanism; when to Ack is policy.
  - This has major implication for the structure of protocols.
- We can apply this across the board to simplify and ensure that cooperating layers behave in a compatible manner.
- Leverage is in the commonality, but letting what must vary vary.
  - Never take it to extremes. It is subtle task.

© John Day, 2014
All Rights Reserved

The Structure of Protocols

- If we separate mechanism and policy, we find there are
- Two kinds of mechanisms:
  - Tightly-Bound: Those that must be associated with the Transfer PDU
    - policy is imposed by the sender.
  - Loosely-Bound: Those that don't have to be.
    - policy is imposed by the receiver.
  - Furthermore, the two are only loosely coupled through a state vector.
    - Implies a very different structure for protocols and their implementations
- Noting that syntactic differences are minimal, we can conclude that
- There is one data transfer protocol with a small number of encodings.

© John Day, 2014
All Rights Reserved

Implications: Protocols I

- Data Transfer Protocols modify state *internal* to the Protocol. Application Protocols modify state *external* to the protocol.
- There are only two protocols (full stop):
  - A data transfer protocol, based on Watson's results,
  - An Application protocol that can perform 6 operations on objects:
    - Read/Write – Create/Delete – Start/Stop
  - There is no distinct protocol like IP.
    - Was just a common header fragment anyway.
- A Layer provides IPC to either another layer or to a Distributed Application with a programming language model. The application protocol is the "assembly language" for distributed computing.
  - As we shall see, we have now made network architecture independent of protocols.

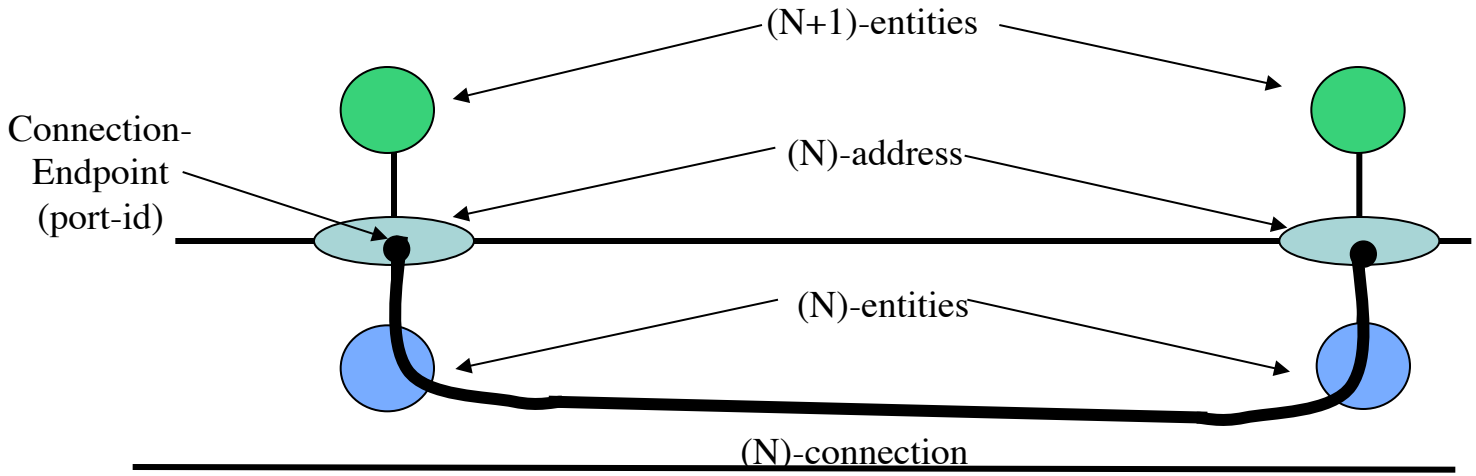
© John Day, 2014
All Rights Reserved

Delta-t Results (1978)

- Richard Watson proves that the *necessary and sufficient* conditions for distributed synchronization requires only that 3 timers are bounded:
  - Maximum Packet Lifetime
  - Maximum number of Retries
  - Maximum time before Ack
- Develops delta-t to demonstrate the result, which has some unique implications:
  - Assumes all connections exist all the time.
  - TCBs are simply caches of state on ones with recent activity
- 1981 paper, Watson shows that TCP has all three timers and more.
  - Matta et al. (2009) shows that delta-t is more robust under harsh conditions than TCP. Boddapati et al. (2012) shows that it is more secure.

© John Day, 2014
All Rights Reserved

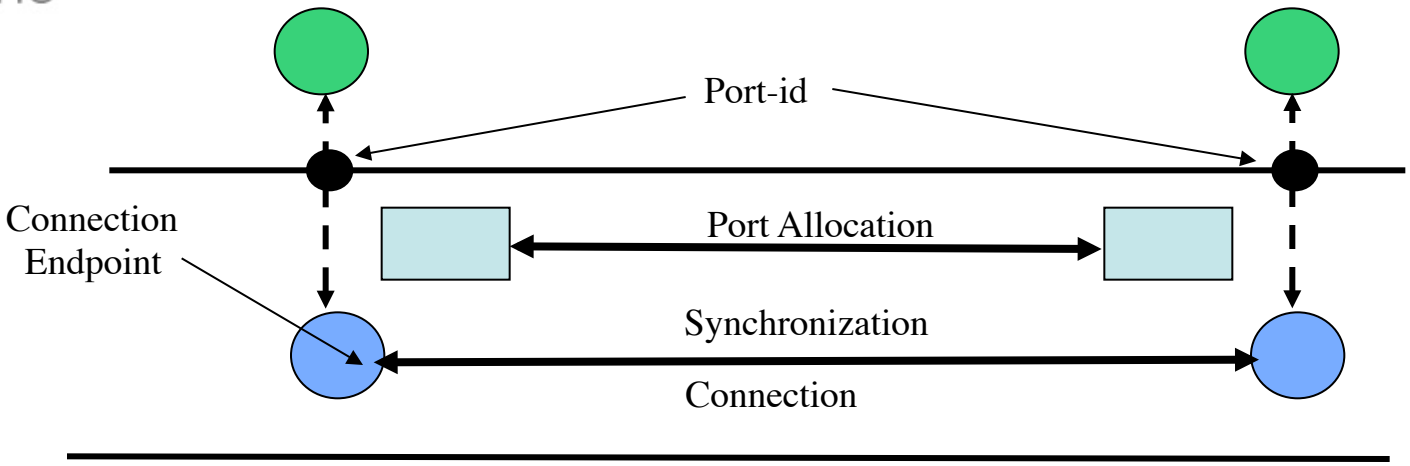
Flaw in the Concept of Connection
(in the Internet and OSI)



- In both, a connection starts in the (N+1)-entity and goes to the peer (N+1)-entity.
  - This is a beads-on-a-string view.
    - (In OSI, while the PTTs insisted on it in the model, it was ignored in practice)
- This combines port allocation and synchronization
- What Watson is saying when he assumes:
  - all connections exist all the time.
- Is that Port allocation and Synchronization are distinct.

© John Day, 2014
All Rights Reserved

Concept of Connection



- Instead delta-t works differently:
  - Ports are allocated, then protocol machines create synchronization (shared state)
  - And connection-end points are bound to the port-ids.
  - We use the term "connection" within the DIF; "flow," for the service provided
- Not conflating the two has significant implications.
  - No traffic for 2MPL, connection disappears, but ports remain allocated
  - Bindings are local. We will later see other implications of this.

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

Implications of Watson's Results

- No explicit state synchronization, i.e. hard state, is necessary.
  - SYNs, FINs are unnecessary
- All *properly* designed data transfer protocols are soft-state.
  - Including protocols like HDLC

- And One Other Thing:
- Watson's result also defines the bounds of networking or IPC:
  - It is IPC if and only if Maximum Packet Lifetime can be bounded.
  - If MPL can't be bounded, it is remote storage.

© John Day, 2014
All Rights Reserved

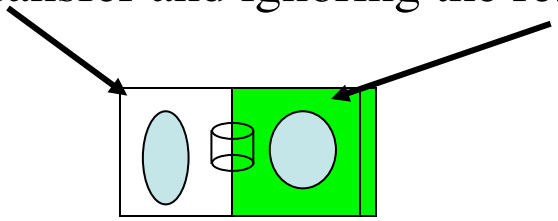
The Connection Connectionless War

- The technical side of what was really an economic war.
  - The Layered Model invalidated both the PTT and IBM business models.
  - Connectionless removed the security blanket of determinism.
  - The war created a bunker mentality that made understanding hard.
    - All or nothing.
- For years, we saw it as the amount of shared state.
  - Connections had lots of shared state; connectionless very little.
- A function of the layer, not a service. Not related to reliability
  - Not visible over the layer boundary.
  - Ports must be allocated even for a connectionless flow.
- Later it became clear that
  - As traffic becomes more deterministic, connections are preferred
    - Down in the layers and in toward the backbone
  - As traffic becomes more stochastic, connectionless is preferred
    - As one moves up and toward the periphery

© John Day, 2014
All Rights Reserved

Resolving the CO/CL Problem

- Lets look at this very carefully
- What makes connection-oriented so brittle to failure?
  - When a failure occurs, no one knows what to do.
  - Have to go back to the edge to find out how to recover.
- What makes connectionless so resilient to failure?
  - Everyone knows how to route everything!
- Just a minute! That means!
  - Yes, connectionless isn't minimal state, but maximal state.
    - The dumb network ain't so dumb.
  - Where did we go wrong?
- We were focusing on the data transfer and ignoring the rest:



© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

So What Do We Know About CO/CL

The Pouzin Society

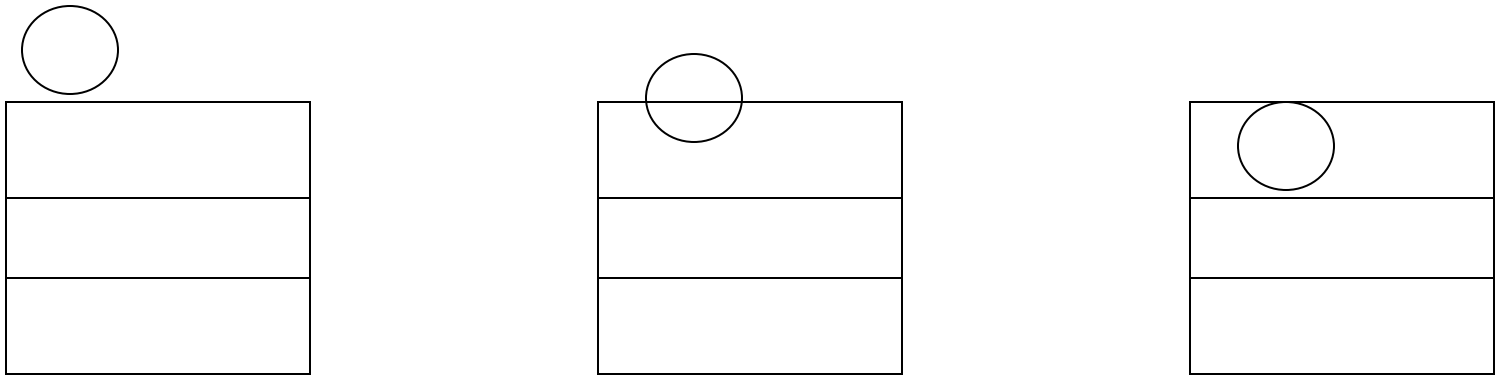
- It is a function of the layer. Should not be visible to applications.
  - Another mistake the Internet makes
- Connectionless is characterized by the maximal dissemination of state information and dynamic resource allocation.
- Connection-oriented mechanisms attempt to limit dissemination of state information and tends toward static resource allocation.
- Applications request the allocation of comm resources.
  - The layer determines what mechanisms and policies to use.
  - Tends toward CO when traffic density is high and deterministic.
  - CL when traffic density is low and stochastic.

© John Day, 2014
All Rights Reserved

Applications and Communication: I

Is the Application in or out of the IPC environment?

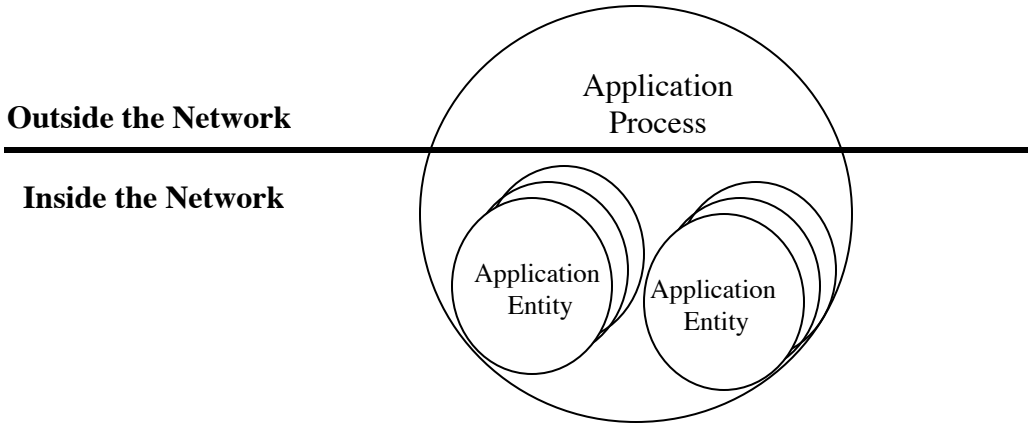
- The early ARPANet/Internet didn't worry too much about it. They didn't need to. They weren't doing any new application development.
- By 1985, OSI had tackled the problem, partly due to turf. Was the Application process inside or outside OSI?



- It wasn't until the web came along that we had an example that in general an application protocol might be part of many applications and an application might have many application protocols.
- The only known example of a political turf battle leading to a major insight!

© John Day, 2014
All Rights Reserved

Applications and Communication: II



- The Application-Entity (AE) is that part of the application concerned with communication, i.e. shared state with its peer.
  - This will turn out to be the "tip of the iceberg," part of a larger structure.
- The rest of the Application Process is concerned with the reason for the application in the first place.
- An Application Process may have multiple AEs, they assumed, for different application protocols.

© John Day, 2014
All Rights Reserved

BUT

As state earlier, there is only one application protocol

- That performs the following operations:
  - Read/Write – Create/Delete – Start/Stop
  - On various objects. Everything is just an object outside the protocol.
    - Application protocols modify state outside the protocol.
- In that case, why do we need all this application-entity stuff?
  - A particular collection of objects are required for an activity.
  - May be shared with others, but has its own access control.
  - One protocol, potentially shared objects, different state machines
  - Hence, all application protocols are stateless, the state is in the application.

  - And, we will see that this was the tip of the iceberg.

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

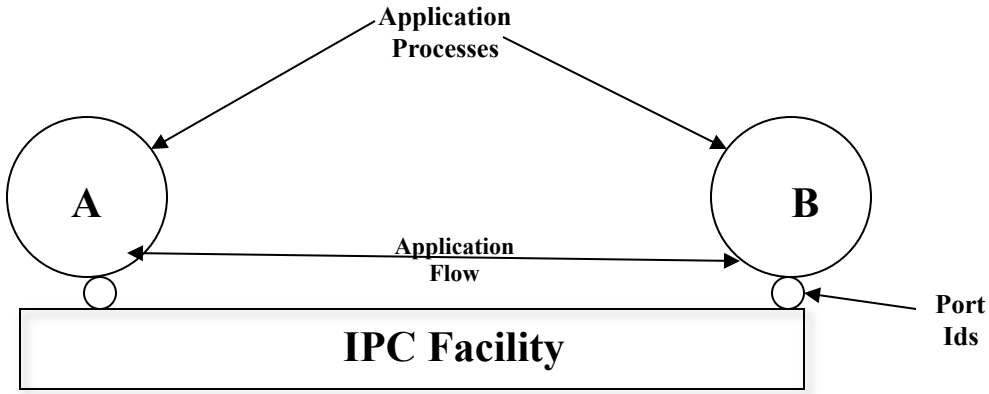
Pristine

The Pouzin Society

A Quick Review

1: Start with the Basics

Two applications communicating in the same system.

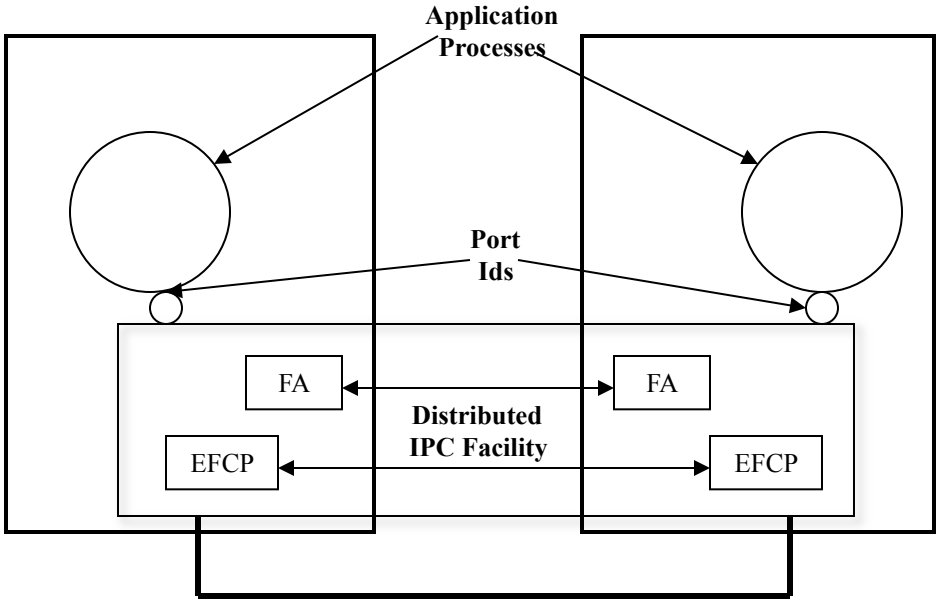


Communication within
a Single Processing System

This is establishes the API. The Application should not be able to distinguish a slow correspondent from operating over the Network.

© John Day, 2014
All Rights Reserved

How Does It Work Now?



- Turns out that Management is the first capability needed to find the other application. Then of course to do that one needs,
- Some sort of error and flow control protocol to transfer information between the two systems.

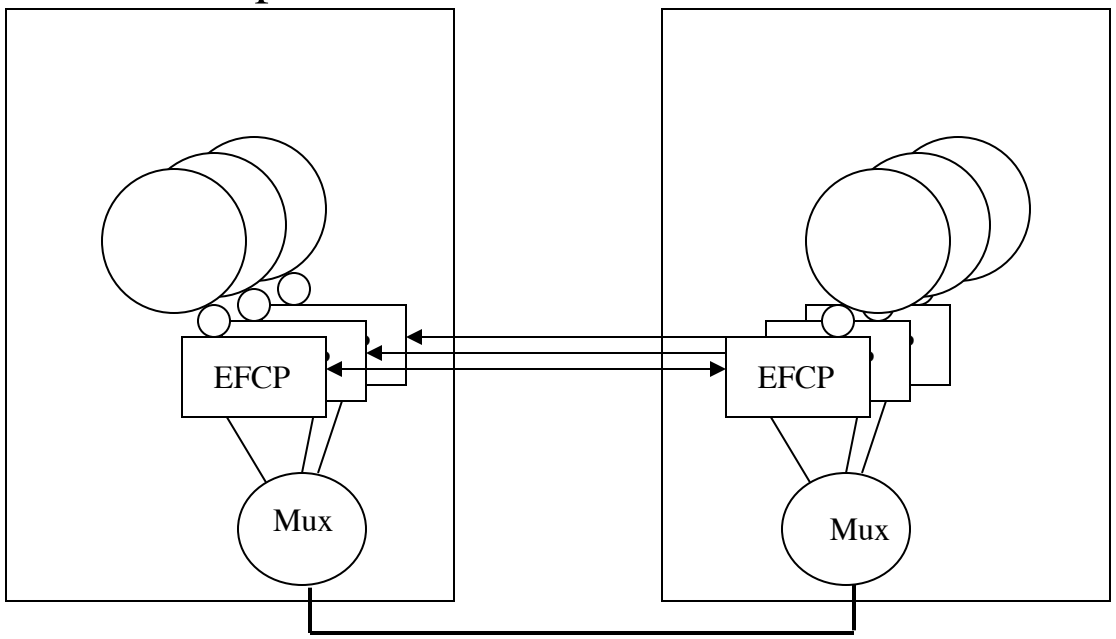
Op	Seq #	CRC	Data

© John Day, 2014
All Rights Reserved

Simultaneous Communication Between Two Systems

i.e. multiple applications at the same time

- Requires two new capabilities



- First, Will have to add the ability in EFCP to distinguish one flow from another.
  - Typically use the port-ids of the source and destination.

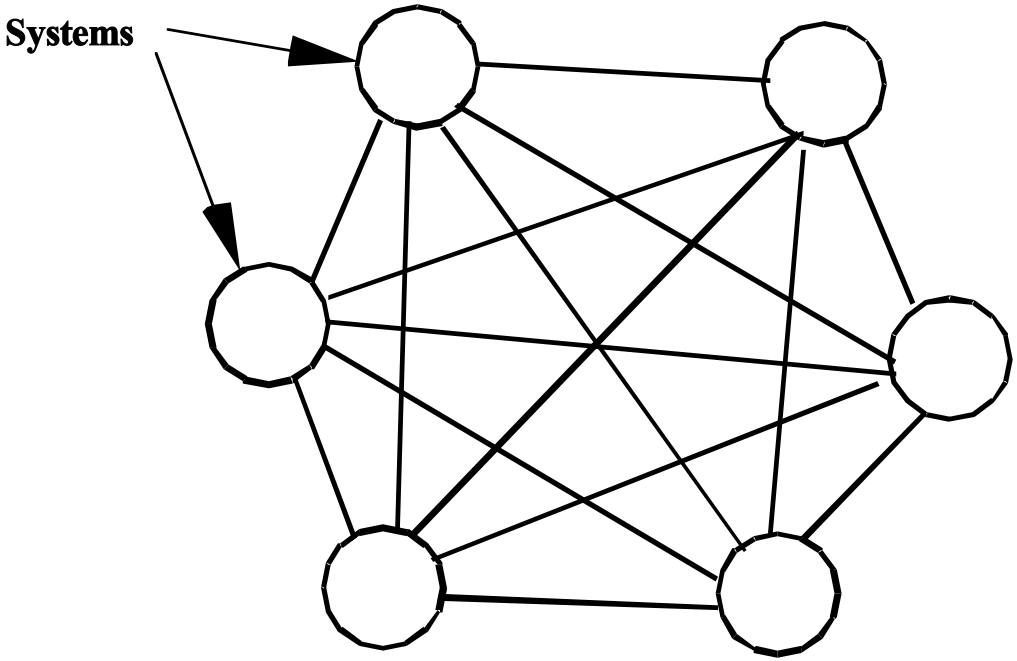
Connection-id

Dest-port	Src-port	Op	Seq #	CRC	Data

- Will also need an application to manage multiple users of a single resource.

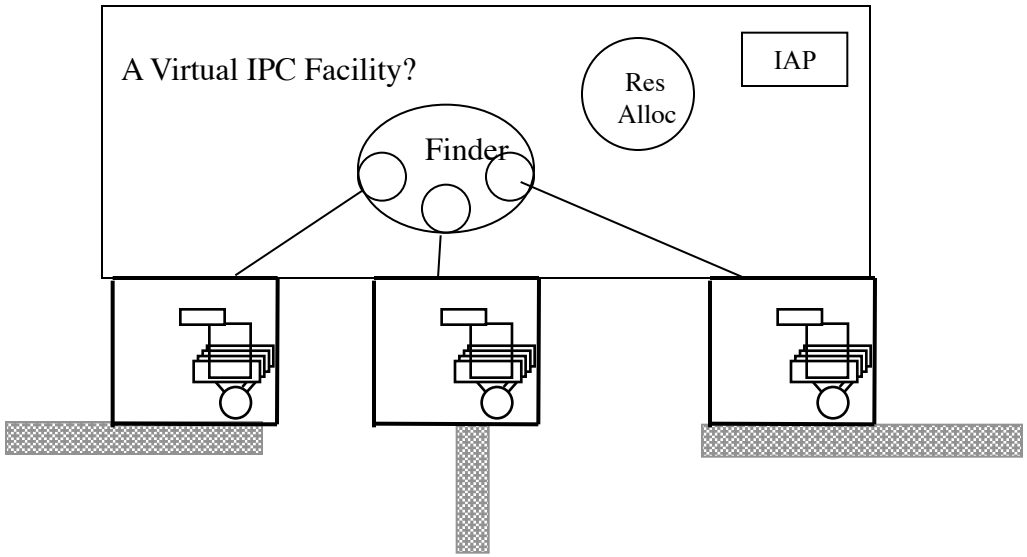
© John Day, 2014
All Rights Reserved

4: Communication with N Systems



© John Day, 2014
All Rights Reserved

A Little Re-organizing



So we have a Distributed IPC Facility for each Interface, then to maintain the API we need an application over all of them to manage their use.

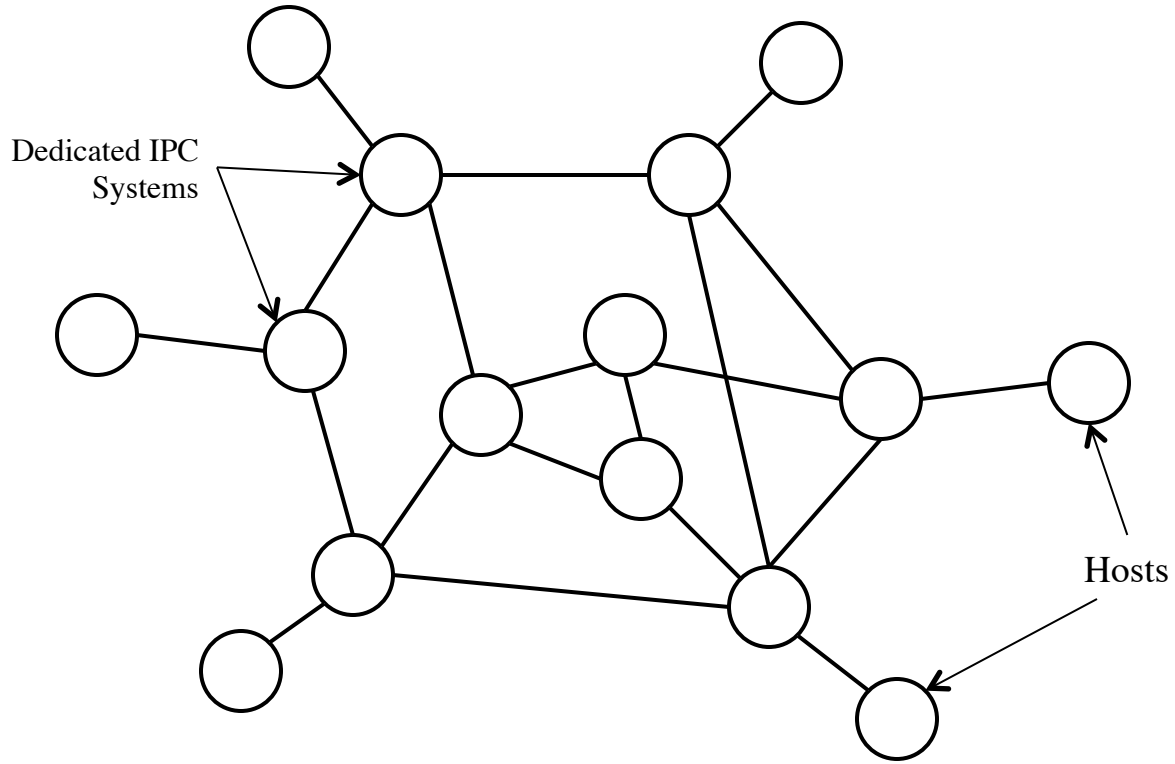
© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

5: Communicating with N Systems (On the Cheap)

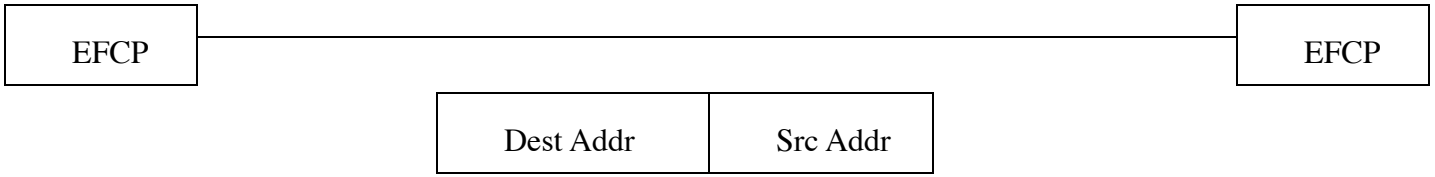


By dedicating systems to IPC, reduce the number of lines required and even out usage by recognizing that not everyone talks to everyone else the same amount.

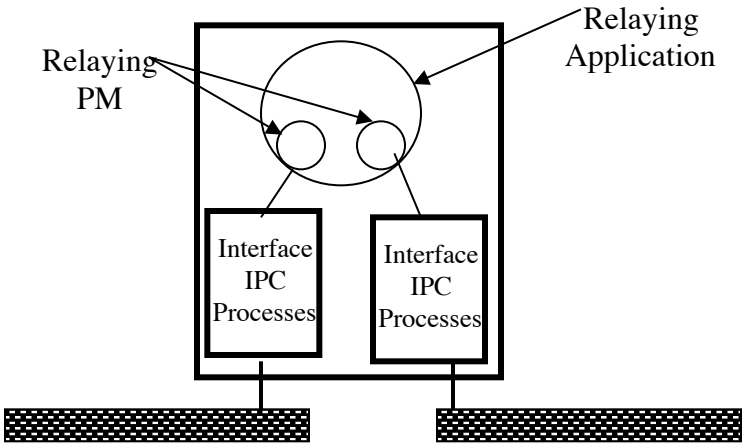
© John Day, 2014
All Rights Reserved

Communications on the Cheap

- But relaying systems over a wider scope requires carrying addresses
- And creates problems too.
  - Can't avoid transient congestion and bit errors in their memories.
- Will have to have an EFCP operating over the relays to ensure the requested QoS reliability parameters



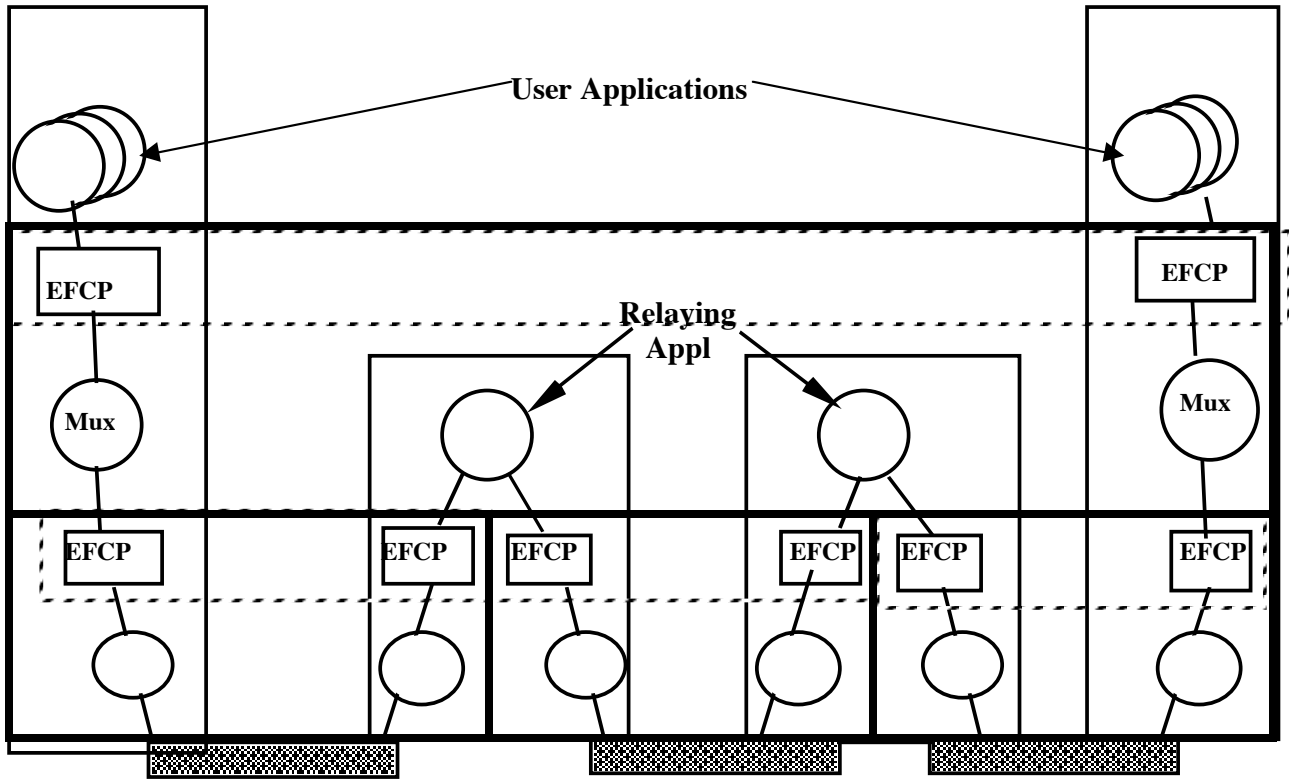
Common Relaying and Multiplexing Application Header



© John Day, 2014
All Rights Reserved

The IPC Model

(A Purely CS View)



Distributed IPC Facilities

© John Day, 2014
All Rights Reserved

The Implications

- Networking is IPC and only IPC.
  - We had been concentrating on the differences, rather than the similarities.
  - Of course, there are layers! What else do you call cooperating shared state that is treated as a black box? A waffle!?
    - Notice the model never required separate protocols for relaying and error and flow control, i.e. separating TCP and IP. Always do what the model says.*
- 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.
  - As we will see, strong layering is better than soft layering.
- 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.
- But this also begs the question:
  - If a layer is a Distributed Application that does IPC, then what is a Distributed Application?

© John Day, 2014
All Rights Reserved

*Always Do What the Problem Says

This requires a bit more consideration.

- Those who did IP would tell you they were following the problem!
  - It was clear that there could be different kinds of transport protocol and they were allowing for that by splitting off IP.
- The problem is that the Problem is a bit coy, she just hints where to go and we have to figure out what she means. ;-)
- In this case, splitting IP from TCP creates problems with fragmentation/reassembly. It breaks the rules.
  - Must not be the direction to take. There must be another way to solve the same problem. We have to look for it.
    - Those who say "simplifying here will merely produce complexity elsewhere" lack imagination.
- If there is a devil in the details, then there is something wrong with the design. A wrong invariance has been chosen.
- The problem is always an angel! ;-)
  - We have to avoid the devils.

© John Day, 2014
All Rights Reserved

The Pouzin Society

That Wasn't the Only Question

- Well Over a Decade Ago
- We Figured Out that Layers Repeated
- That Created a Potential Problem:

© John Day, 2014
All Rights Reserved

Dykstra says Layers are all Different

- Once a function is done in a layer, it isn't repeated.
- If you are going to contradict Dykstra,
  - you better have a good argument, so . . . Better see what he said.
- The Layers of Dykstra's THE Operating System
    - (THE - Technische Hogeschool Eindhoven)
    - "The Structure of the THE-Multiprogramming System," CACM 11(5) 1968.
  - Layer 0 - Multiprogramming
  - Layer 1 - Memory Management
  - Layer 2 - Console communication
  - Layer 3 - I/O
  - Layer 4 - User Programs
  - Layer 5 - Users ("not implemented by us" - EWD)

How Do You Do This . . .

Without This?

- Seem a bit dated, don't they?

© John Day, 2014
All Rights Reserved

A Few Days Later

- Musing about Dykstra's layers:
  - A bit arcane, but 1968 was a long time ago. Machines were far smaller and much more resource constrained.
- We wouldn't do those layers in an OS today.
- What would we do . . . . hmmmmmm
  - kernel, supervisor, user . . .
  - and they all do the same thing!
    - scheduling, memory management and IPC
  - OMG! OSs are recursive as well!
    - Of course this is what the virtualization fad has hacked into without seeing it.
- That lead to

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

Returned to OSs After A 30 Year Hiatus

- Knowing that the Days Were Over When H/W was a Major Constraint and Distorted Possible Solutions!
  - Now things could done the way they should be!
- But what I found was,
  - Nothing had Changed. Still same old timesharing systems!
- No one Seems to have Noticed.
- What had Everyone Been Doing!?

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

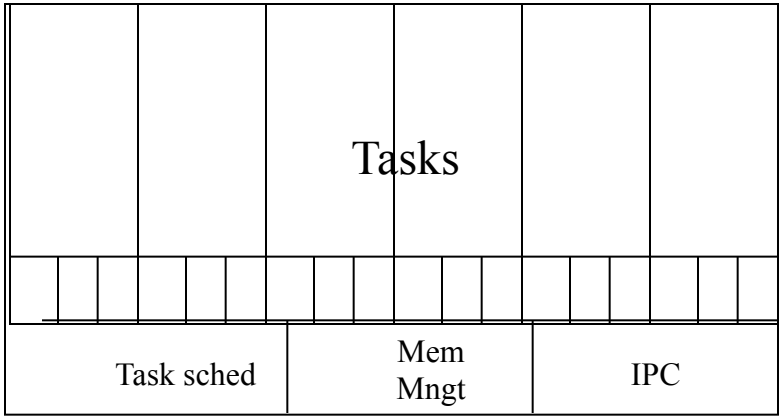
What I Saw

- Threads and Heap management were kludgey with all sorts of warts.
- They never answered the question, if a thread is the locus of processing, then what does that make the "process"?
- Answer: The process is the OS for the threads. The OS is recursive.
    - Hence,
        - <system> ::= <h/w isolation><process>
        - <process> ::= <scheduling> <memory mngt> <IPC><thread>*
        - <thread> ::= <process>
- All Peripherals have their own processor and hence their own specialized OS.
    - There is no I/O, only IPC.
- For most OSs, IPC wasn't there or something cumbersome.

- There were other implications and more to question . . .

© John Day, 2014
All Rights Reserved

The Structure of a Process



- A Process has a recursive structure consisting of task scheduling, memory management, and inteprocess communication to manage its tasks, which are, in turn, processes.

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

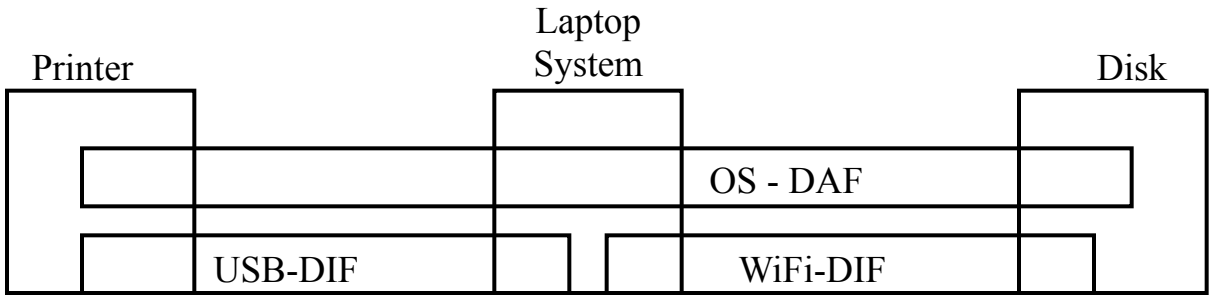
The Pouzin Society

What Does a Modern OS Look Like?

- It certainly isn't a 70s timesharing system

© John Day, 2014
All Rights Reserved

The OS is actually a Distributed Management System



- A traditional OS is a heterogeneous DAF that includes the peripherals.
  - The traditional device drivers are members of the DAF.
  - In the case of the disk, it might have several members: one, looks like a file system, one that looks like a database, and one that yields track and sector access.

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

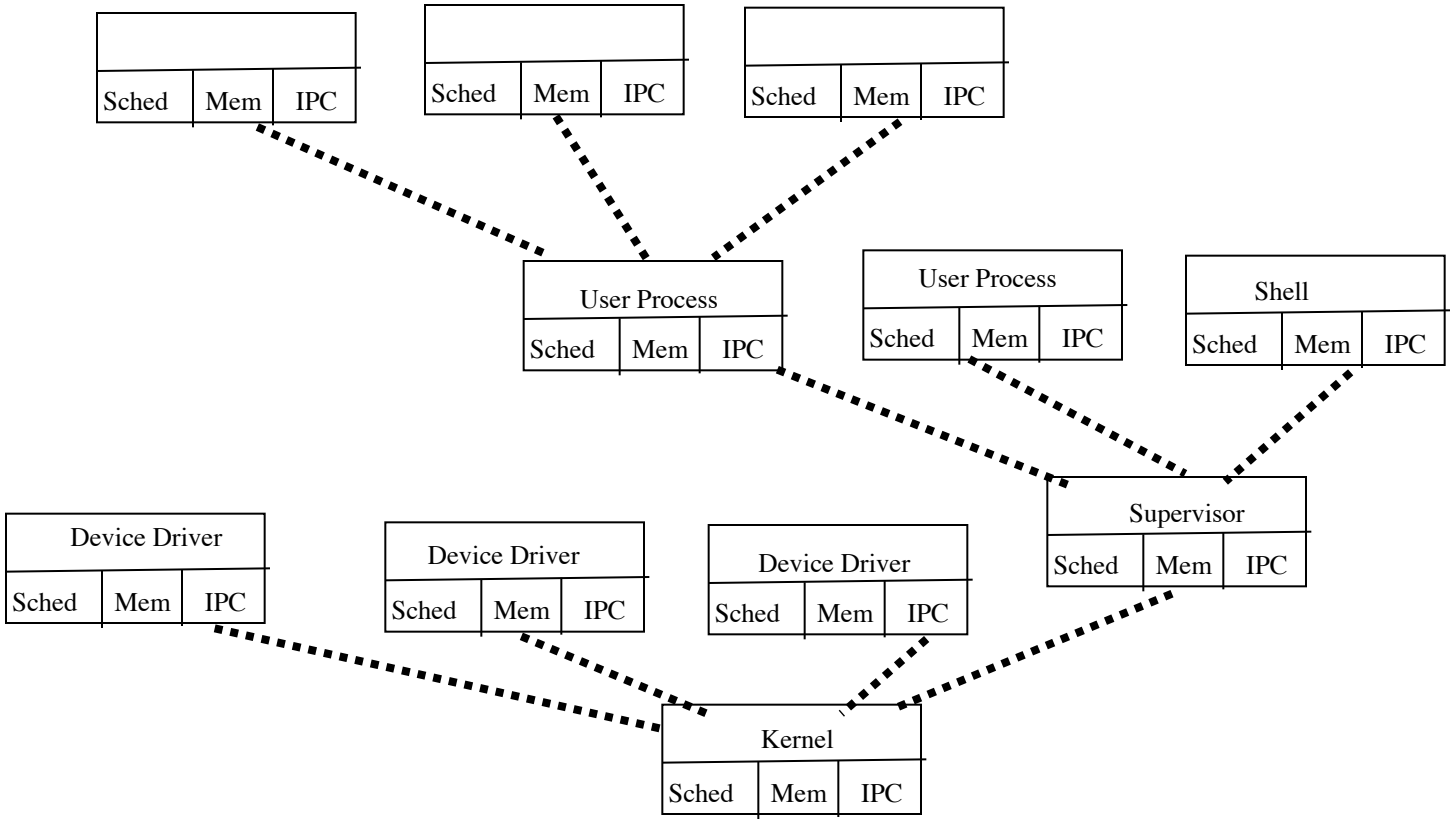
The Pouzin Society

The Basic Model

- A Processing System is everything in the scope of a "test and set" instruction.
- A Computing System is a collection of processing systems wholly or partially under the same management domain.
  - The concept of "my computer" or a computing domain
- The *Operating System* then is really only the *kernel*
- What has been traditionally called the OS is necessarily a distributed management system, you wouldn't know it by what is said.
- Above that may be complex applications that manage their own resources, i.e. with their own OS.

© John Day, 2014
All Rights Reserved

Recursive Operating Systems



Each Process has an OS at its core to manage the "Tasks" of that process. However, "Tasks" are processes to their "tasks" and so on. "Tasks" may be allocated to their own processor.

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

Gee, that is nice, BUT. . .

- We are solving Networking problems!
- Stay focused. One thing at a time!

- Except that the problem keeps dragging us into answering what is a distributed application?

- Well, okay! . . . Then lets just use what we have . . .

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

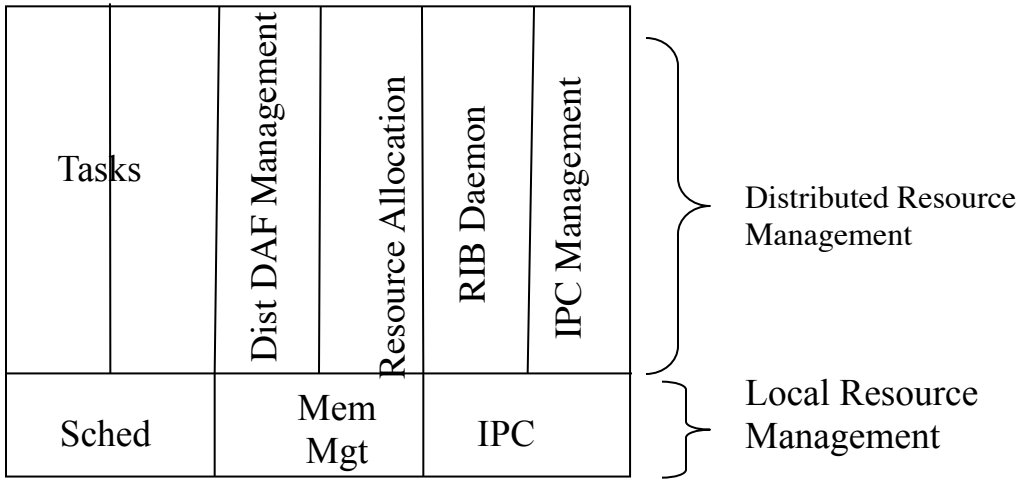
This Has Implications for DAFs

- If a DIF is a set of IPC Processes,
  - really Distributed IPC Processes (DIPs?)
- Then a DAF must be a set of Distributed Application Processes
  - What else but, DAPs!

- What Does a DAP look like?

© John Day, 2014
All Rights Reserved

Structure of DAP



- Local Resource Management is the basic Process structure.
  - Have to manage local resources.
- Then a task for each of these to coordinate with other DAPs in the DAF.

© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

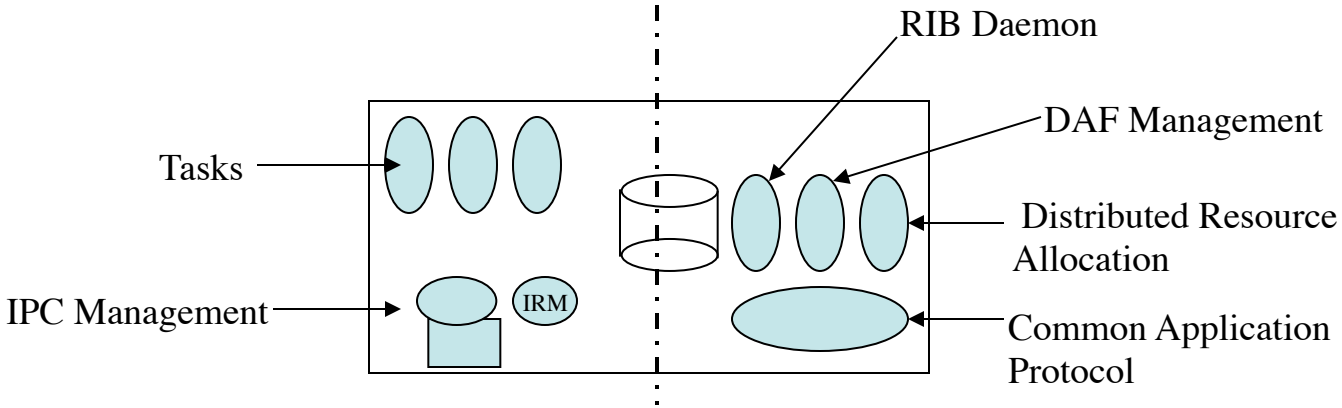
The Pouzin Society

What Do They Do?

- DAF Management - is the local agent for the overall management of the DAF.
- Resource Allocation - corresponds to task scheduling and manages the distributed aspects of the load generated by the tasks.
- RIB Daemon - ensures that the information necessary for the DAP to perform its function is available, both for tasks and the DAF components.
- IPC Management - manages the DAP's use of supporting DIFs.

© John Day, 2014
All Rights Reserved

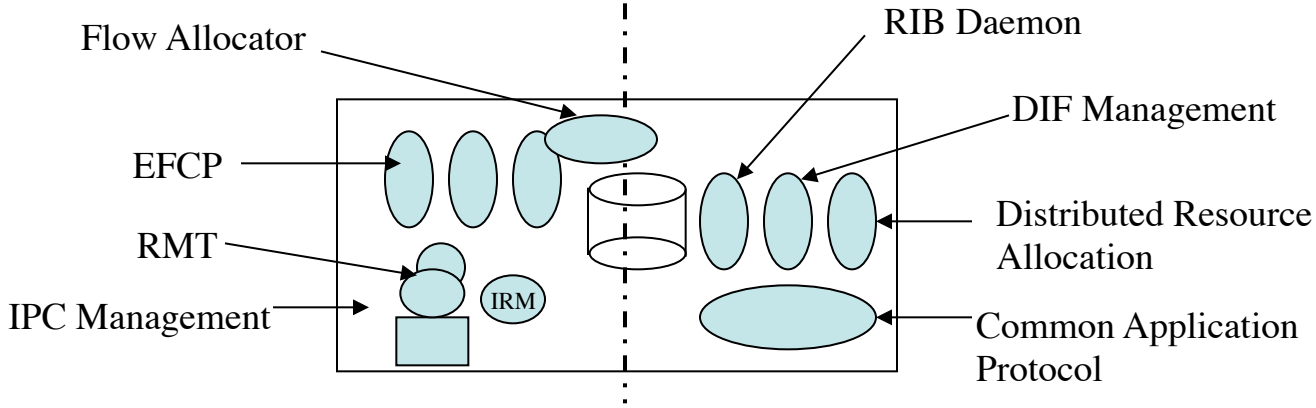
Distributed Application Facility (DAF)



- A DAF consists of cooperating DAPs. Each one has
- The Basic Infrastructure:
  - Resource Management - Cooperates with its peers to manage load, generate forwarding table, etc.
  - RIB Daemon - ensures that information for tasks is available on whatever period or event require, a schema pager.
  - IPC Management - manages the supporting DIFs, multiplexing and SDU Protection.
  - Tasks - the real work specific to why the DAF exists.
- DAPs might be assigned synonyms with scope within the DAF and structured to facilitate use within the DAF. (!)
- Therefore, a DIF is . . . .

© John Day, 2014
All Rights Reserved

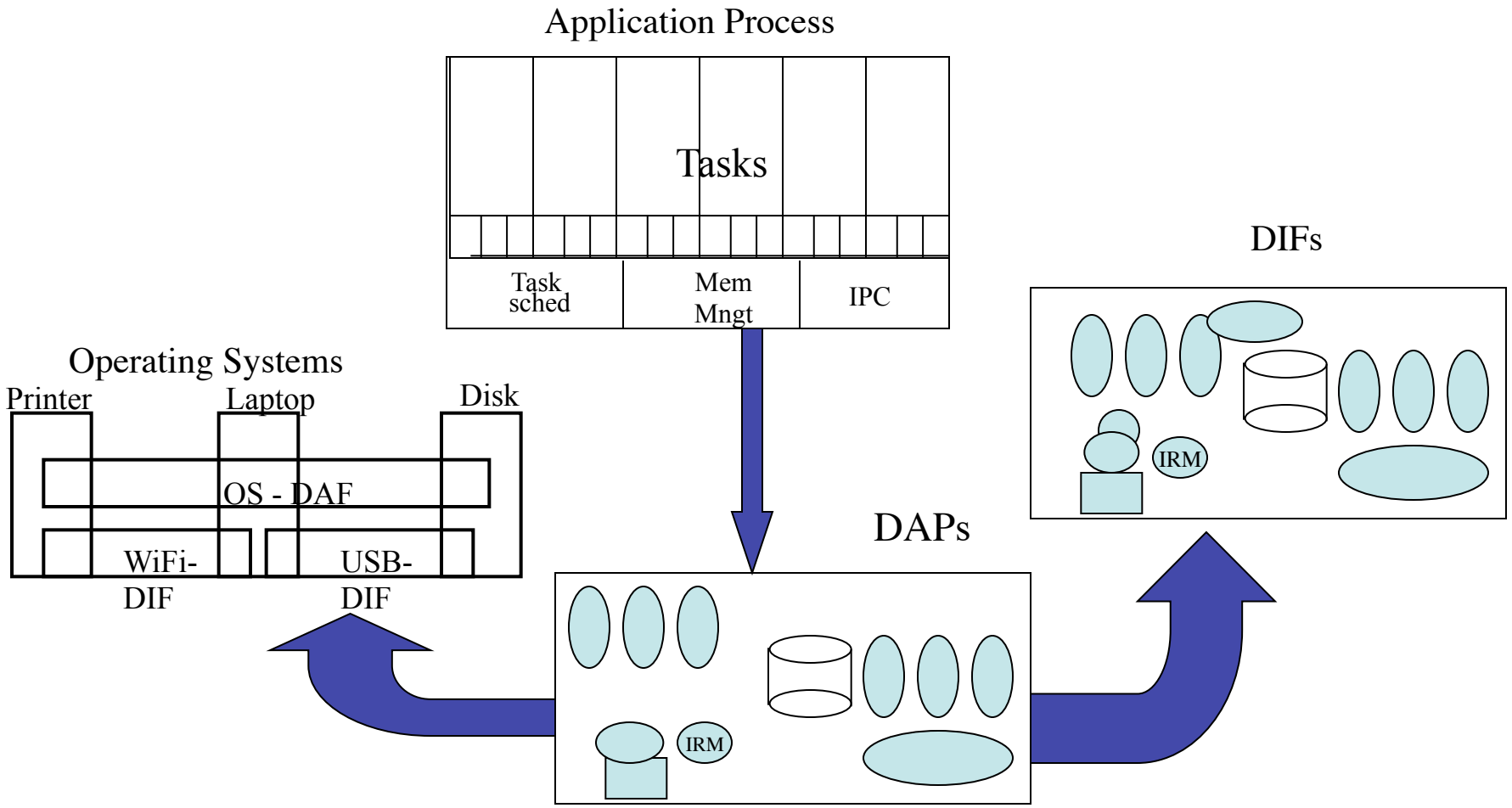
Distributed IPC Facility (DIF)



- A DAF consists of cooperating DAPs. Each one has.
- The Basic Infrastructure:
  - Resource Management - Cooperates with its peers to manage load.
  - RIB Daemon - ensures that information for tasks is available on whatever period or event the tasks (and the DAF components) required, routing update, other event notifications
  - IPC Management - manages the supporting DIFs, multiplexing and SDU Protection.
  - Tasks - Flow Allocator, EFCP, and relaying.
- IPC Processes are assigned synonyms with scope within the DIF and structured to facilitate routing.

© John Day, 2014
All Rights Reserved

A Single Unified Model that Encompasses Operating Systems, Distributed Applications and Networking



© John Day, 2014
All Rights Reserved

IRATI
INVESTIGATING RINA

Pristine

The Pouzin Society

The Organization of
The RINA Reference Model

- Part 1: Basic Concepts of Distributed Systems
- Part 2: Distributed Applications
  - Chapter 1: Basic Concepts of Distributed Applications
  - Chapter 2: Introduction to Distributed Management Systems
- Part 3: Distributed InterProcess Communication
  - Chapter 1: Fundamental Structure
  - Chapter 2: DIF Operations

- New Chapters and Extensions are expected as we learn more.

© John Day, 2014
All Rights Reserved

There's More to Come

- DIFs and How They Work
- Then Naming and Addressing
  - It turns out to be quite straightforward and simple.
- Then a Look at the Internal Operation of a DIF and the Implementations.
- But for Now. . .

© John Day, 2014
All Rights Reserved

The Pouzin Society

Questions?

© John Day, 2014
All Rights Reserved