Networking is IPC and only IPC

John Day

2021

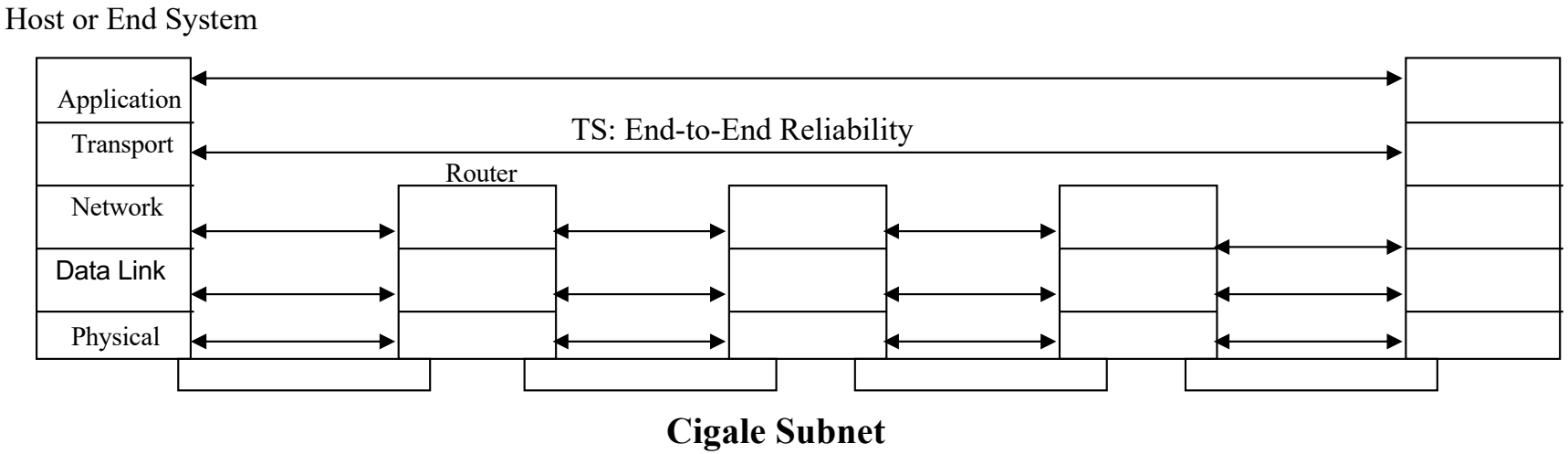
"There is something fascinating about science. One gets such wholesale returns on conjecture out of such a trifling investment of fact."

- Mark Twain

"Life on the Mississippi"

The Cyclades Architecture

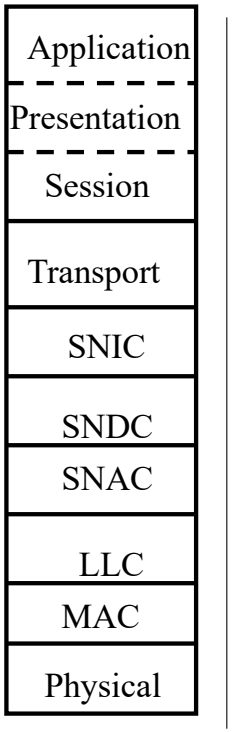
(1972)



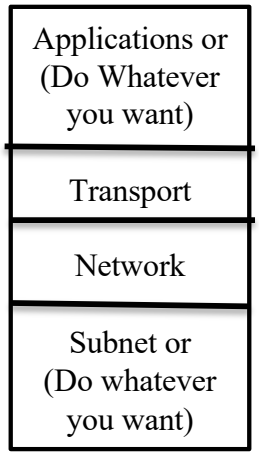
- The Layers as developed by CYCLADES were:
  - Transport Layer provides end-to-end reliability, primarily recovering from loss due to congestion.
  - The Network Layer Does Relaying and Multiplexing of datagrams with the primary loss due to congestion and rare memory errors.
  - The Data Link Layer detects corrupted packets and may do flow control.
    - Hence, it must keep the loss rate much less than the loss due to congestion by the Network Layer.
  - The Physical Layer is the Wire.

At First There Were the 7 Layers

But As Things Developed Things got more complex



While The Internet Stayed With



Regardless, "Something was rotten in Denmark"

That Reflected What We Were Seeing

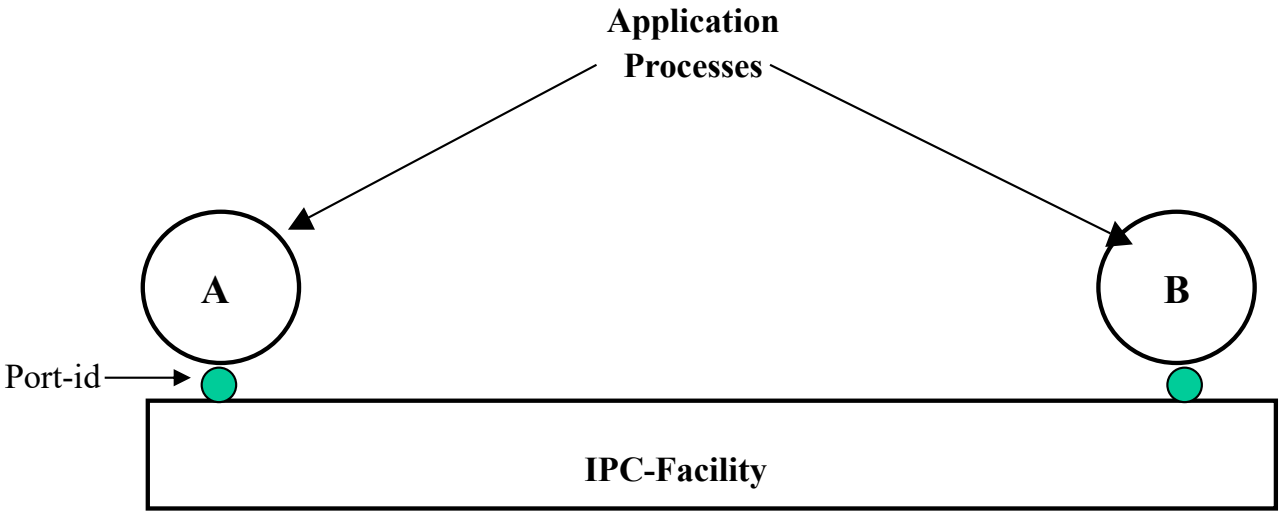
But It Was Clearly a Hodge-Podge

Getting Started

- To understand why networks are the way they are, we must do what the problem says!
- Let's start with basics and build it up.
  - This is going to look *very* simple.
    - Like everything in CS, but it gets more complicated
    - As we do simple things over and over!
  - We are going to "throw away a ladder" on occasion. We will introduce a concept but then modify it or even eliminate it when we gain a better understanding.
- What do we need for communication?
  - Some minimal shared understanding
    - The applications have to understand each other's messages.
  - A medium to carry messages between the two parties.
    - In operating systems, the mechanism to do this is referred as Inter-Process Communication (IPC).

1: Start with the Basics

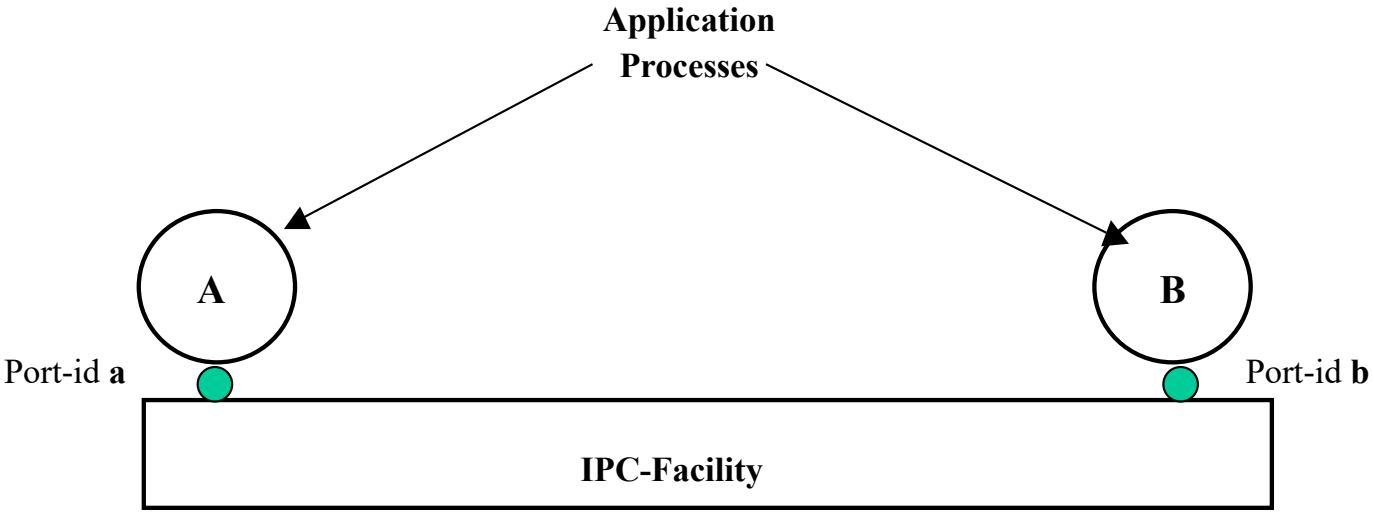
Two applications communicating in the same system.



InterProcess Communication in a Single System

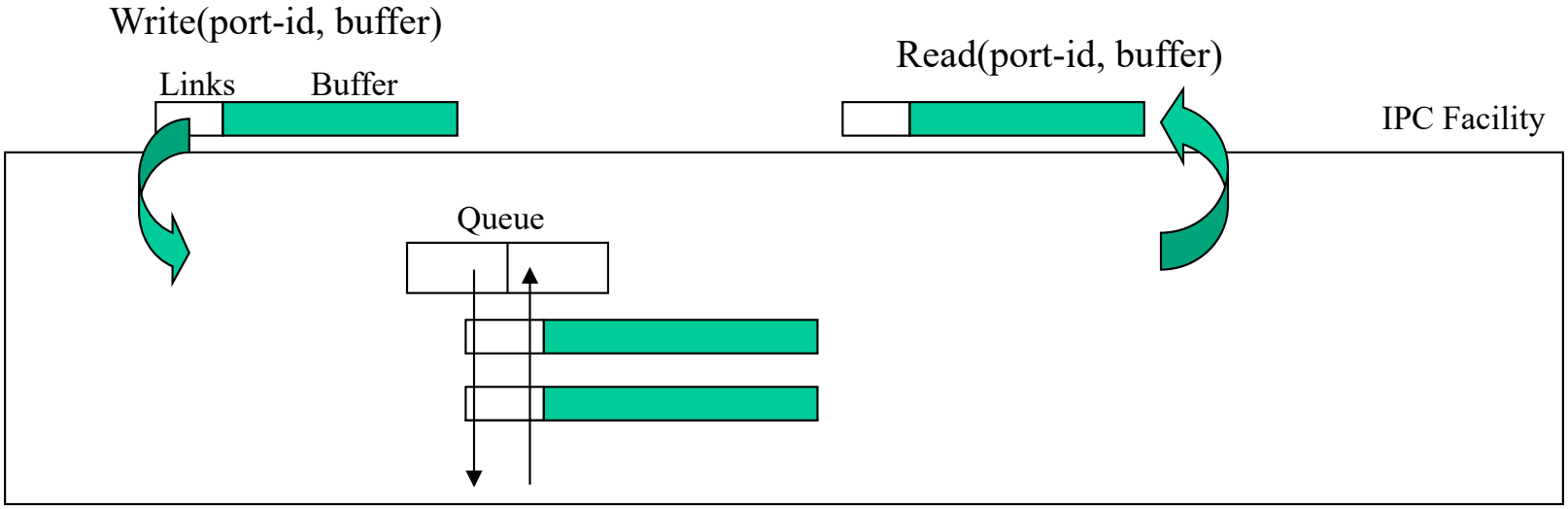
- The Application invokes System Calls to talk to another Application
  - port-id := Allocate (dest-appl-name, IPC characteristics)
  - Read (port-id, buffer-ptr)
  - Write (port-id, buffer-ptr)
  - Deallocate (port-id, reason)
- IPC creates a channel between the two applications, identified by "port-ids" of local scope.
  - Port-ids are a short identifier for the end of the channel and allow an application to have multiple IPC channels at the same time.
    - Serves the same purpose as a file descriptor.
- IPC is generally constructed with a shared state mechanism.
  - Allocates buffers, if sender tries to send too much it is blocked.
  - A characteristic of IPC is it requires instructions to create a critical section to control concurrency, to keep everyone out of each other's way.

How Does It Work?



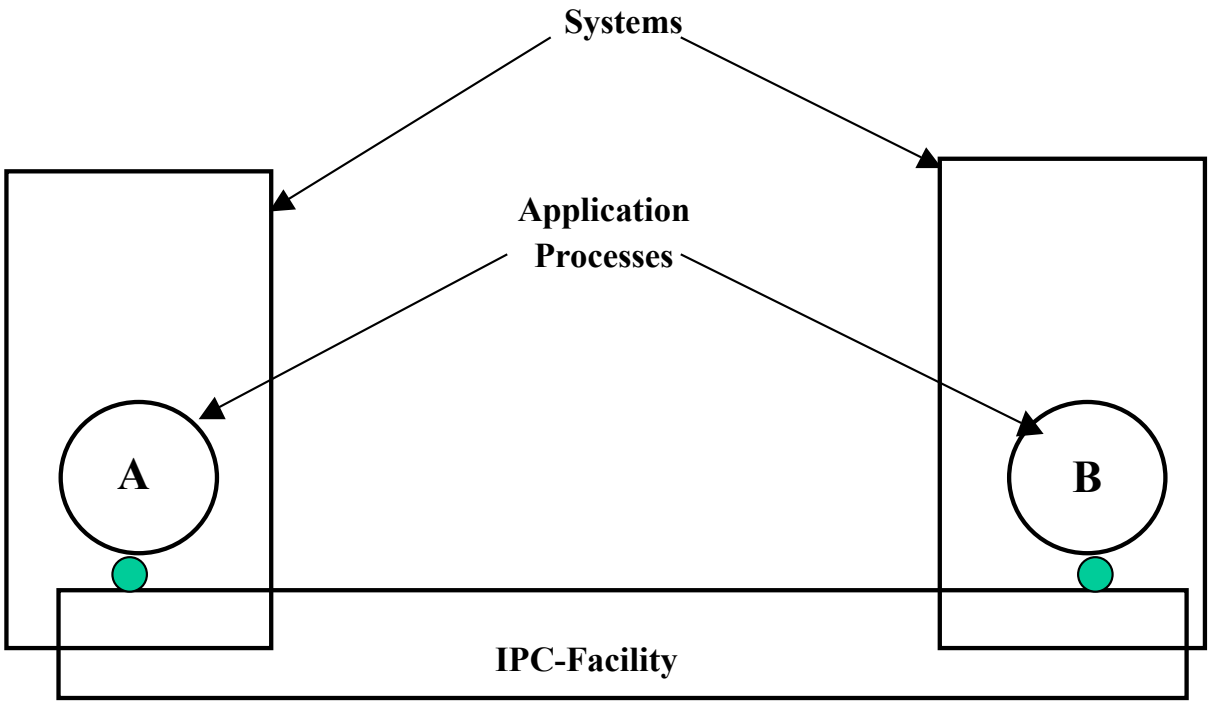
1) **A** invokes the IPC Facility to allocate a channel to **B**; **a** <- Allocate(B, x);
2) IPC determines whether it has the resources to honor the request.
3) If so, IPC uses "search rules" to find **B** and determine whether A has access to it.
4) IPC may cause **B** to be instantiated. **B** is notified of the IPC request from **A** and given a port-id, **b**.
5) If **B** responds positively, and IPC notifies **A** using port-id, **a**.
6) Thru n) Then using system calls **A** may send PDUs to **B** by calling **Write(a, buf),** which B receives by invoking **Read(b, read_buffer)**
7) When they are done one or both invoke Deallocate with the appropriate parameters

What Happens Inside

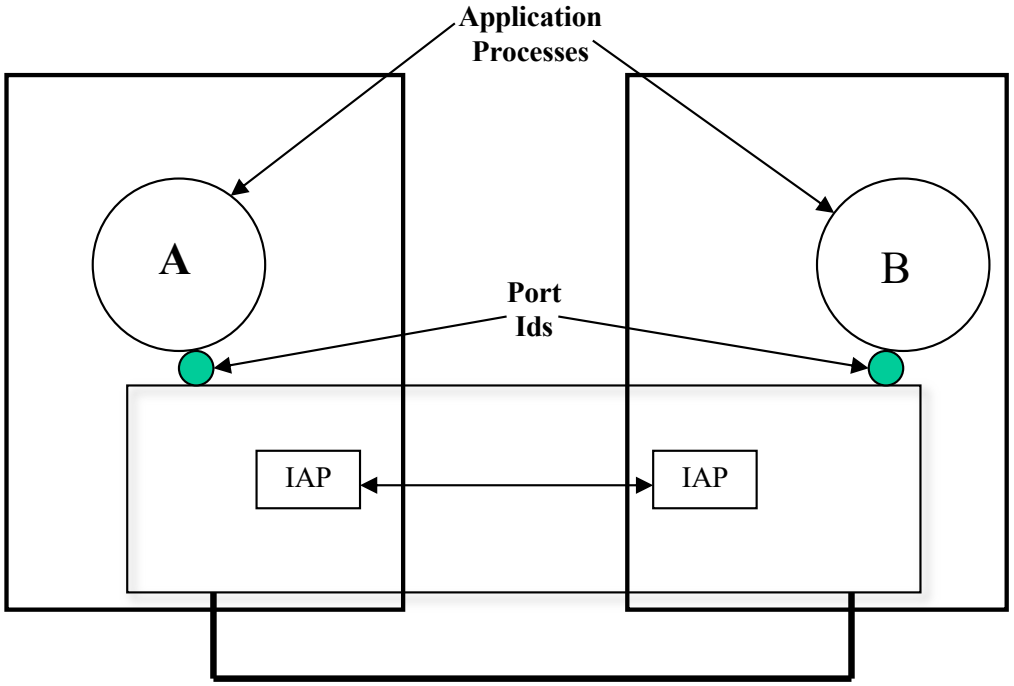


- A FIFO queue head is allocated for each port-id pair.
  - Memory allocation keeps links (user can't see) so buffers can be passed from queue to queue
  - Lots of ways to Implement this.
  - But manipulating the queue requires a critical section, i.e., synchronization.
- Each Write causes a buffer to be linked to the end of the queue.
- Each Read causes buffer at head of the queue to be dequeued and passed to the application.

2: Two Application Communicating in Distinct Systems



How Does It Work Now?



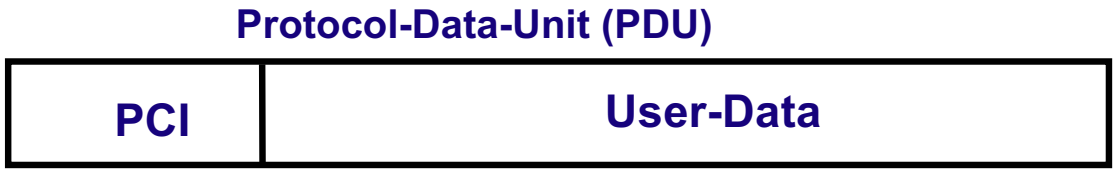
1) A invokes the IPC Facility to allocate a channel to **B**; a <- Allocate (B, x);
2) IPC determines whether it has the resources to honor the request.
3) If so, IPC uses "search rules" to find **B.**
   - Management of name space is no longer under the control of a single system.
     - Each system no longer knows all available applications.
   - Need to tell the other side what to do with the bits being sent to it!
   - Local Access Control can no longer be relied on to provide adequate authorization and authentication.

- Tell what it wants to talk to. Thus, need a protocol to carry application names and access control information.
  – Let's call it the IPC Access Protocol

What Does "IAP" Do?

- Find out if the other system has an application with this name.
- Tell the other system something about the nature of the communication being proposed between the two applications.
  - Sometimes called Quality of Service.
- Send the other system access control information to determine whether the requesting application is allowed to access what it is asking for.
- How do we know when to use it?
  - If the application isn't here, it must be over there!

The Important Thing about PDUs



- This Message is a bit different. Must tell Protocol Machine what to do, and carry User-Data that isn't understood.
- PDUs are called packets, frames, messages, cells, segments, etc. They are all the same thing just in different places.
- PCI or Protocol-Control-*Information* is also called the header.
- I prefer PCI because it emphasizes that:
  - *Information* is what the protocol understands,
  - *User-Data* is what it doesn't understand.
    - The first User-Data will be an IAP-PDU.
- At each layer, the protocol only processes the PCI.

What Does IAP Look Like?

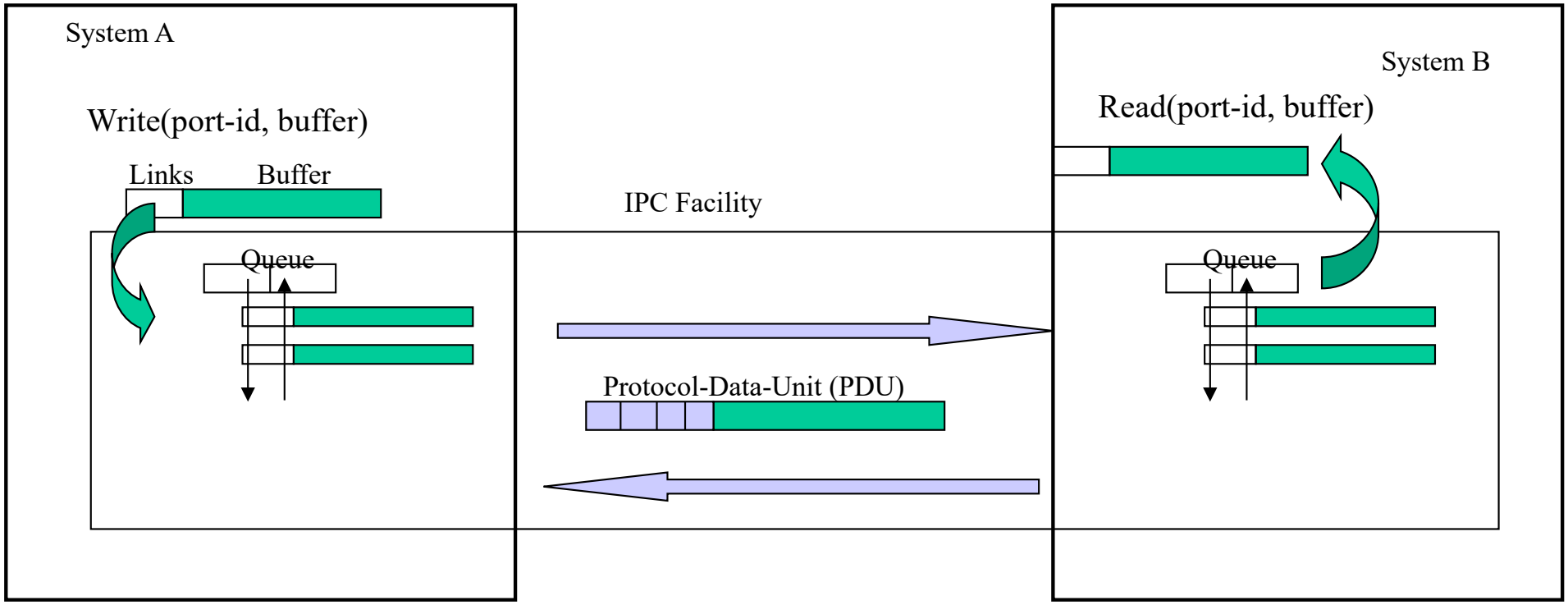
- Simple Request/Response Application Protocol
  - IAP-Req(Dest-Appl-name, message-id, Src-Appl-name, QoS params, Src-authorization)
  - IAP-Resp(Dest-Appl-name, Src-Appl-name, message-id, QoS params, result)

  - A little explanation
    - Include the QoS parameters so that the destination can determine whether it has the resources and the policies to fulfill the request.
    - Notice anything odd about this PDU format?
      - No User Data (The Buck Stops Here)
- But we have a more basic problem:
- How do we get it there?

Getting from A to B

- We knew this was going to be a problem sooner or later.
- We need to be able to send information from A to B.
- And we know:
  - Bad things can happen to messages in transit. Protection against lost or corrupted messages
  - Receiver needs to be able to tell sender, if PDU was received (Ack)
  - Receiver must be able to tell sender, it is going too fast. We need Flow Control. These are feedback mechanisms and
  - Feedback requires loose synchronization and we have lost our means of synchronization:
    - No shared memory. Need new means of synchronization.
    - Must create shared state between two systems. An explicit synchronization mechanism is required.
- We need some kind of Protocol to do Error Recovery and Flow Control.
  - We will call it an "Error and Flow Control Protocol" (EFCP)

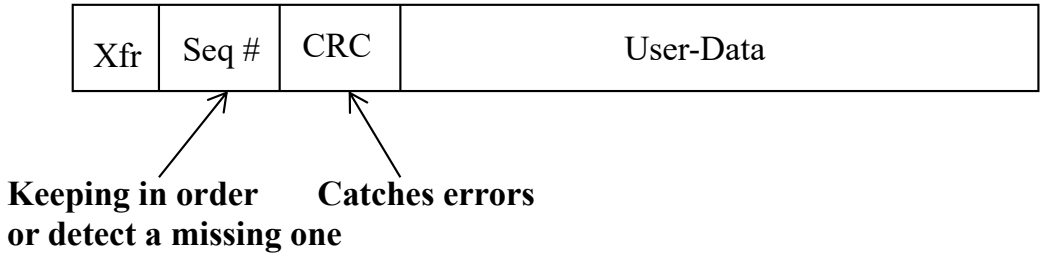
What Happens Inside Now



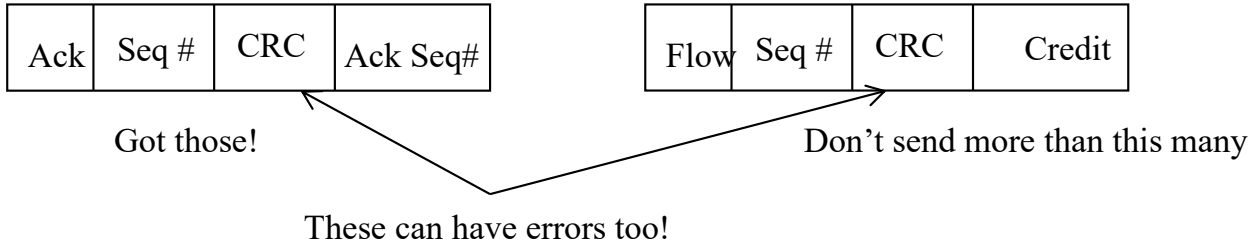
- Now we have real problems! We need a queue at each end *and*
- What needs to be in the PDU?
  - Detect lost or corrupt PDUs, delimit and pack SDUs as User Data, and
- We need synchronization locally for the queues, but we also need synchronization between the systems for flow and retransmission control
  - Tell the sender how much it can send, and what needs to be retransmitted.

What Does an EFCP Look Like?

- Symmetric Protocol to provide error and flow control
  - Transfer PDU

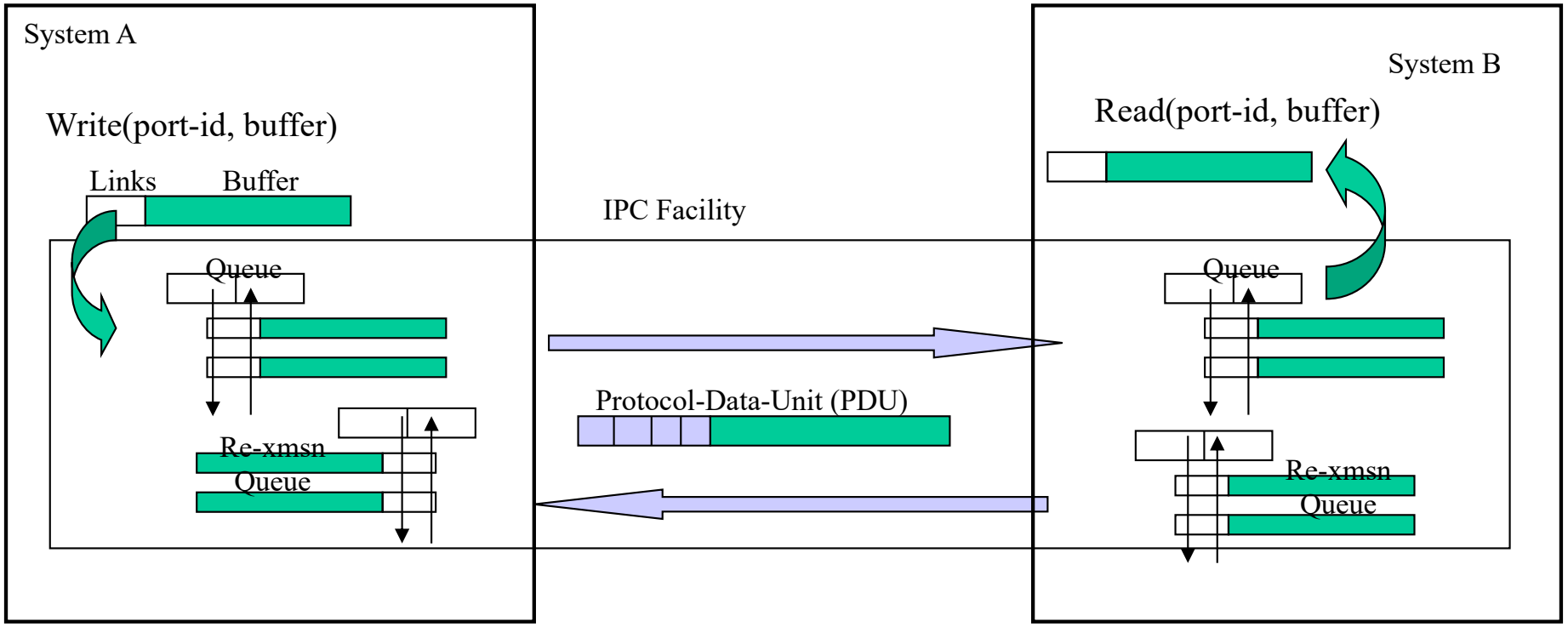


  - Ack and Flow Control PDU types



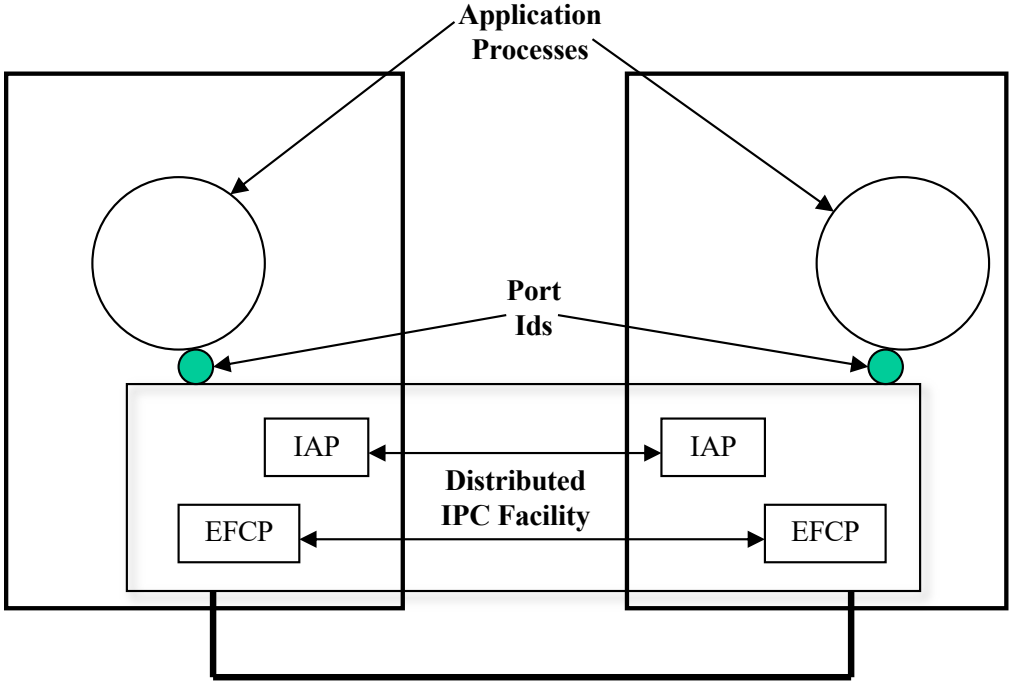
Might combine these
Curious, no user data here either.

What Happens Inside Now



- We need a queue at each end for both data to send and for retransmissions, if data sent is not ack'ed.
  - Actually, a pair of queues, one for each direction (but only so much room)
- The PCI tells the other end what to do with the data and to maintain the synchronization, i.e., the state
- We will go into greater detail later, on exactly how this works.

How Does It Work Now?



1) **A** invokes the IPC facility to allocate a channel to **B** by calling an *Allocate*.

2) IPC determines whether it has the resources to honor that request. If so, IPC uses its *search rules* to find **B.** Not finding it locally, the IAP function constructs an IAP PDU and passes it to the EFCP function, which constructs an EFCP PDU with the IAP PDU as the User-Data and sends and sends the EFCP PDU to **B** creating an EFCP connection to use for sending the IAP request. The EFCP function in the other system will process the EFCP PDU to determine the User-Data is good, strips off the EFCP PCI and passes it to the IAP function.

3) The IAP function processes its PDU and looks for B. When it finds **B,** it determines whether **A** has access to **B**. IPC may cause **B** to be instantiated.

4) **B** is notified of the IPC Request from A is and given port ID **b.**

5) If **B** responds positively, IPC reverses the process in 3) and sends an IAP Response, IPC notifies **A** using port-id **a.**

6) to N, **A** and **B** do their *reads* and *writes.* A and B pass PDUs as SDUs to the IPC Facility, which encapsulates them in EFCP PDUs and sends them, the PDUs are received and processed by the EFCP function in the other system, the PCI is stripped off and delivered to the other application. A and B sending PDUs back and forth with data for a while and when they're done.

7) One or both of them do a *Deallocate* on the port-id to release the resources.

Just Like Before . . . .More or Less

Summarizing the assumptions that no longer hold: I

- Management of name space is no longer under the control of a single system.
  - Each system no longer knows all available applications.
- Must tell the other guy what to do with the stuff sent to it!
- Local Access Control can no longer be relied on to provide adequate authorization and authentication. IPC and the Applications need to be explicitly told about the source.
- All resources are no longer under the control of a single system

Summarizing the assumptions that no longer hold: II

- Bad things can happen to messages in transit. Must detect lost or corrupted messages. Tell sender what has arrived intact.
- Receiver must be able to tell sender, it is going too fast.
  - We need Flow Control.
- These require Synchronization; No Shared Memory, or test and set.
  - Must create distributed shared state between two systems.
    - Explicit synchronization required
- Otherwise, not much has changed!
  - At least not for the applications. (This is important)

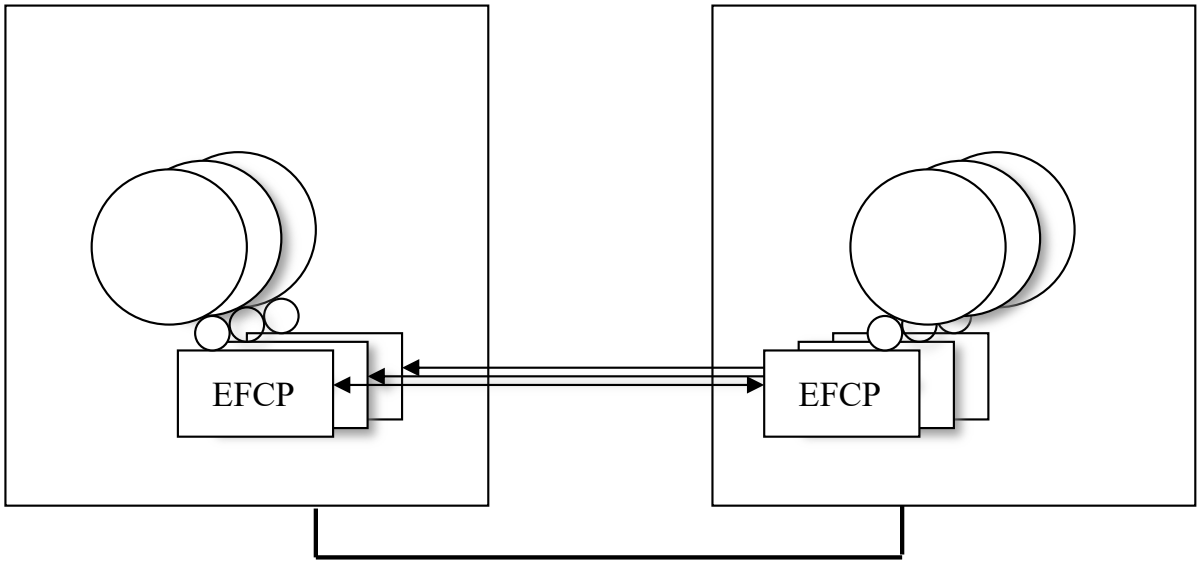
Invalidating these Assumptions Requires Three New Concepts

- An Application Name Space that spans both systems. (not really new)
  - The easy thing to do would be to prefix the application-name with a host name.
- A Protocol to carry Application Names and access control info
  - Applications need to know with whom they are talking
  - IPC must know what Application is being requested to be able to find it.
    - For now, if the requested Application isn't local, it must in the other system.
- A Protocol that provides the IPC Mechanism and does Error and Flow Control.
  - To maintain shared state about the communication, i.e. synchronization
  - To detect errors and ensure order
  - To provide flow control
- Resource allocation can be handled for now by either end refusing service.

3: Simultaneous Communication Between Two Systems

i.e. multiple applications at the same time

- To support this, we have multiple instances of the EFCP.



Will have to add the ability in EFCP to distinguish one flow from another.

Connection-id

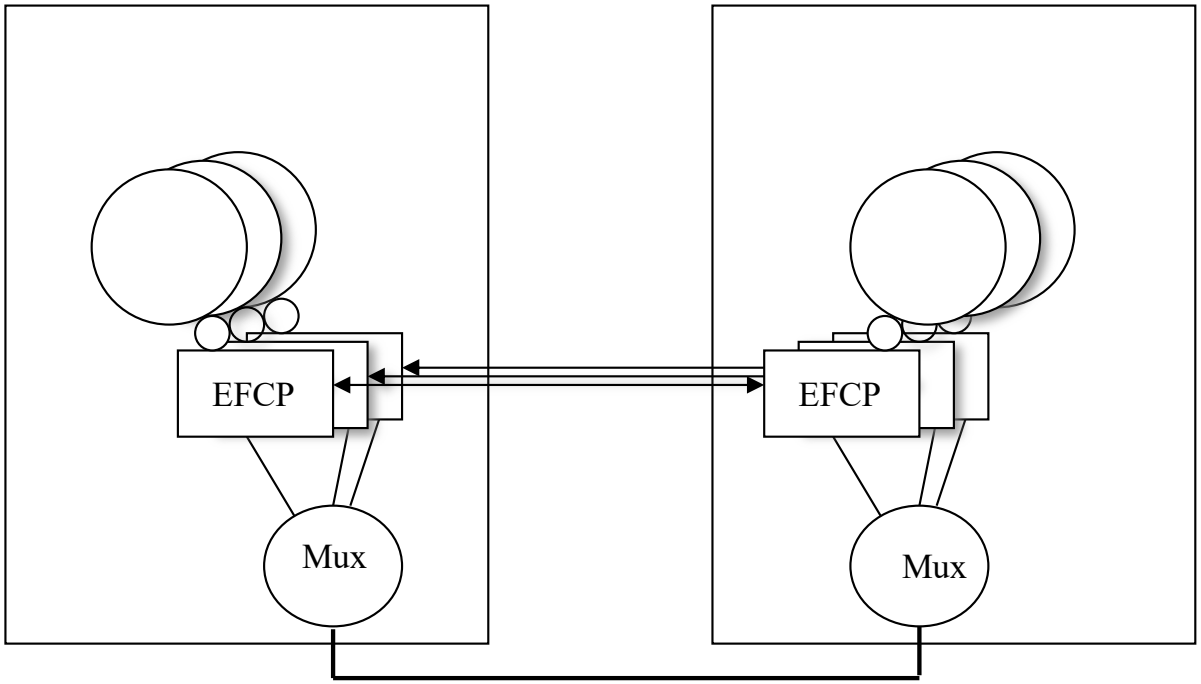
Dest-port	Src-port	Op	Seq #	CRC	Data

Typically use the port-ids of the source and destination.
Also include the port-ids in the information sent in IAP to be used in EFCP synchronization (establishment).

Simultaneous Communication Between Two Systems

i.e. multiple applications at the same time

- In addition to multiple instances of the EFCP.

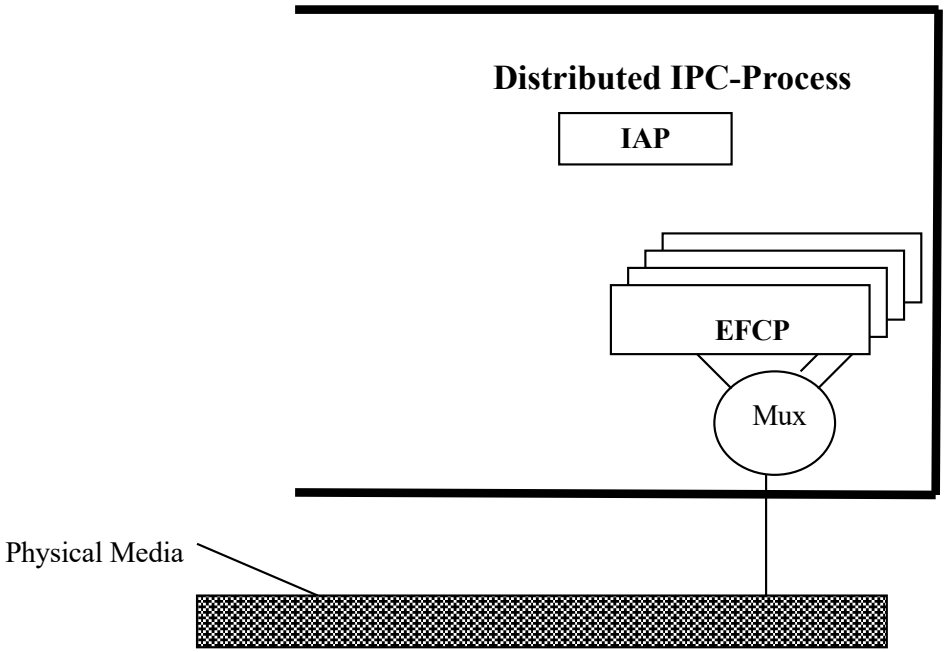


Will also need an application to manage multiple users of a single resource.
(interesting: the definition of an operating system problem)

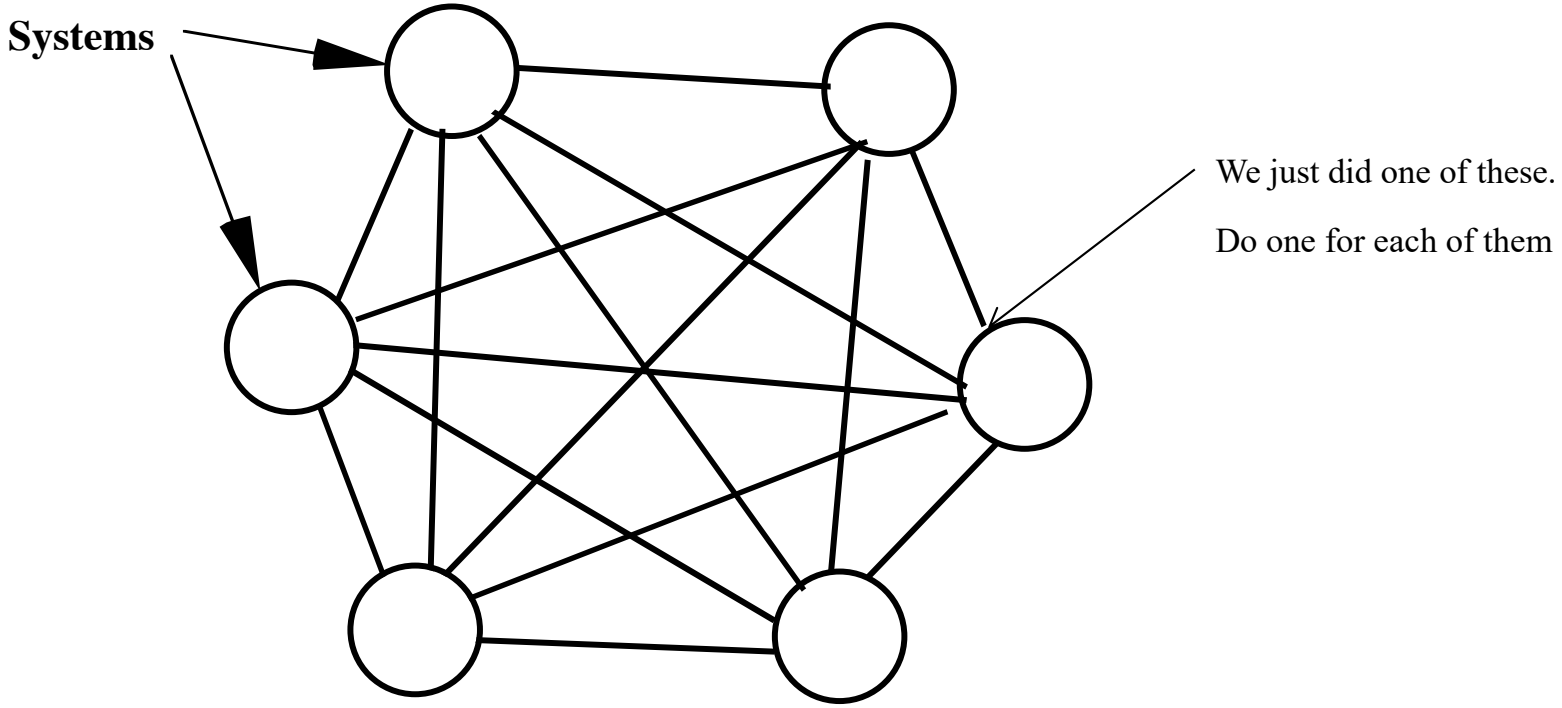
Multiple Instances of IPC

- New Concept: When more than one connection between two systems, a connection-id must be present in the PDU to distinguish what data belongs to which connections.
  - Commonly formed by concatenating local port-ids from each end of the connection to form an unambiguous connection-id within the scope of the two systems.
- New Concept: a multiplexing application to manage the single resource, the physical media.
  - The multiplexing application will need to be fast, its functionality should be minimized, i.e., just the scheduling of messages to send.
  - To provide different QoS, we can assign different classes of traffic to queues with different priorities.
- Application naming gets a bit more complicated than just multiple application-names.
  - Must allow multiple instances of the same process

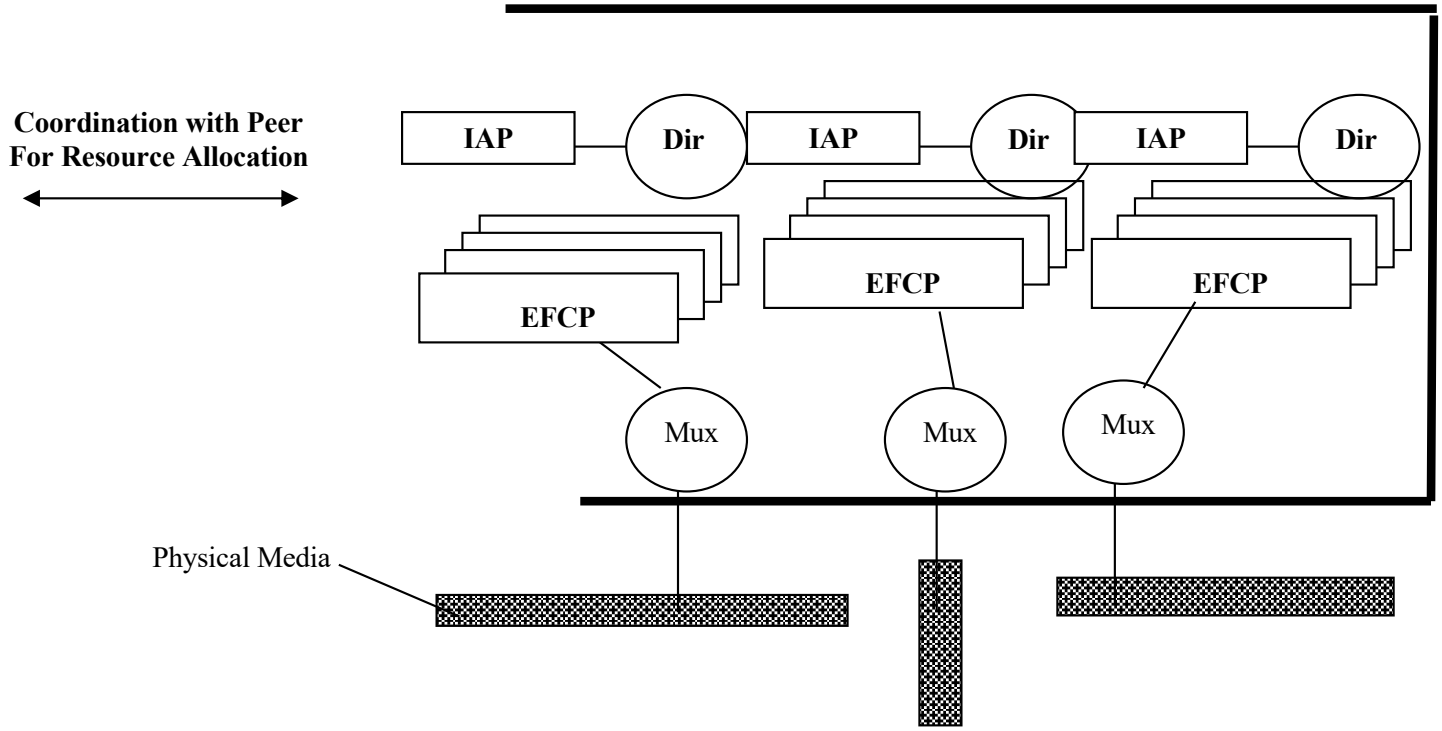
IPC to Support Simultaneous Communication between Two Systems



4: Communication with N Systems

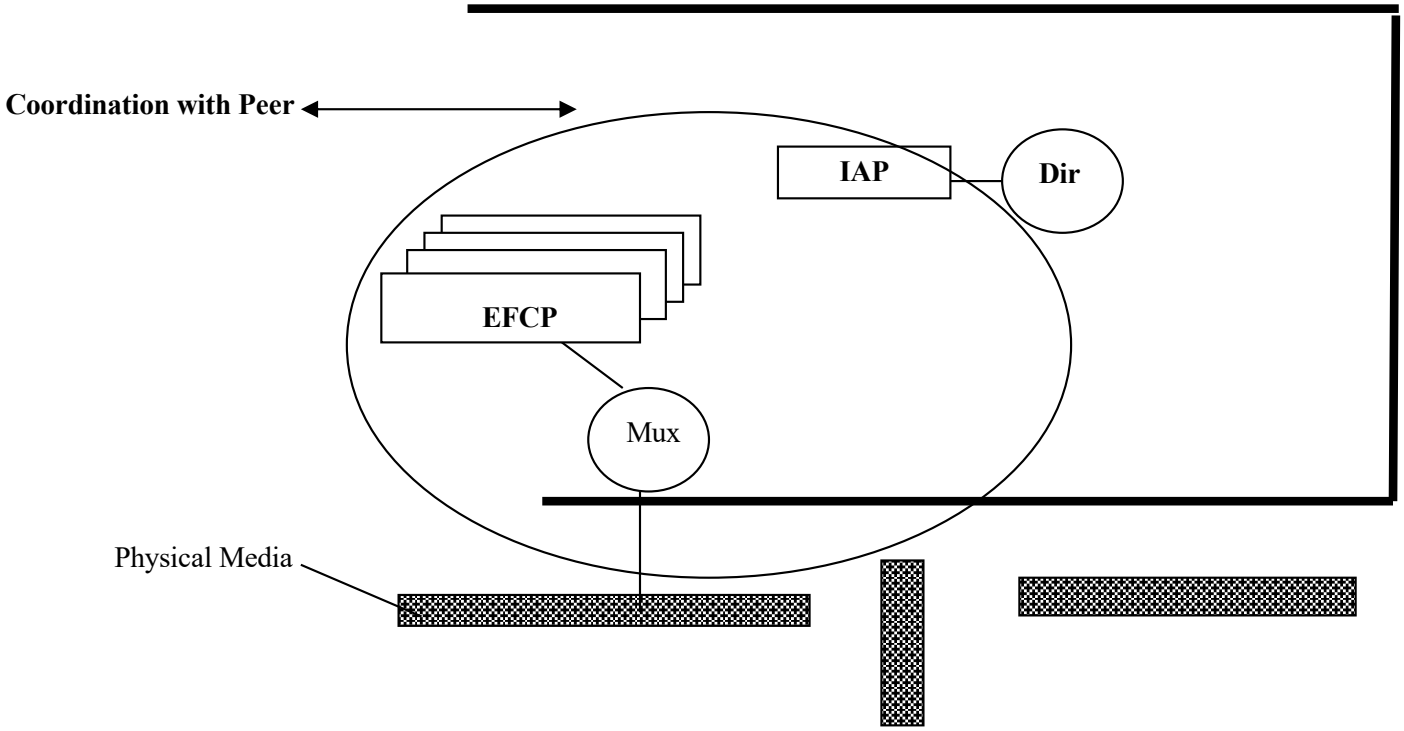


We will Need a Separate Multiplexing Application for each physical media interface.



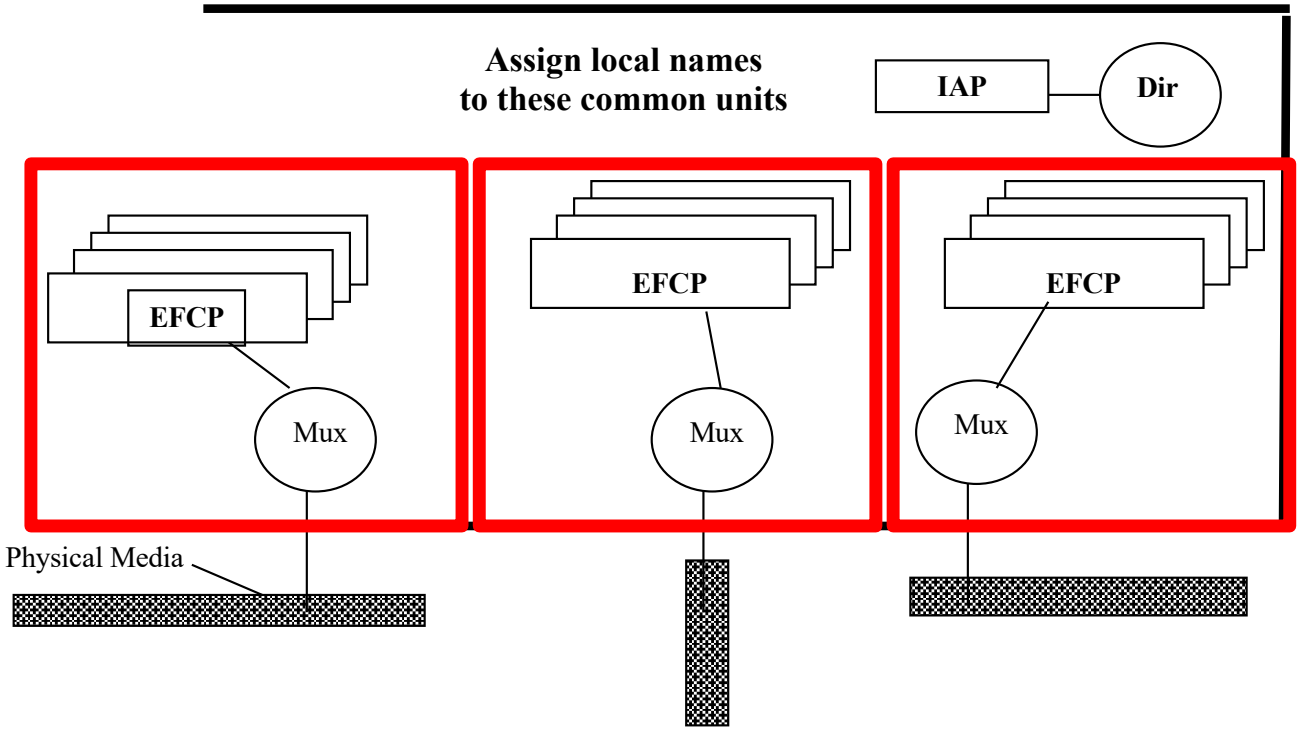
- There is a common structure here that can be factored out.
- And we need to use IAP to find out what applications are at the other end of the wire.
- And with more systems need better coordination of resources

There is Some Common Structure



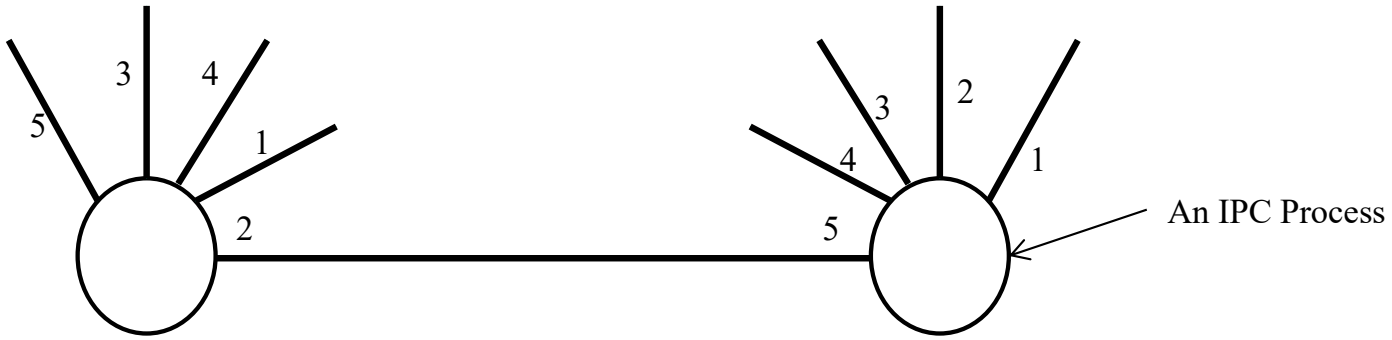
- We can organize interface IPC into modules of similar elements.
- Each one constitutes a Distributed IPC Facility of its own.
  - As required, consists of IAP, EFCP, Mux Application, Directory, Per-Interface Flow Manager
- Then we just need an application to manage their use and moderate user requests.

We need to keep track of what application is at the end of which wire



- Need a local name for each wire.
  - Wires are point-to-point we just need to know what is at the other end.
- Need to keep a table and expand the use of IAP to find them.

Keeping Track of Who is Where

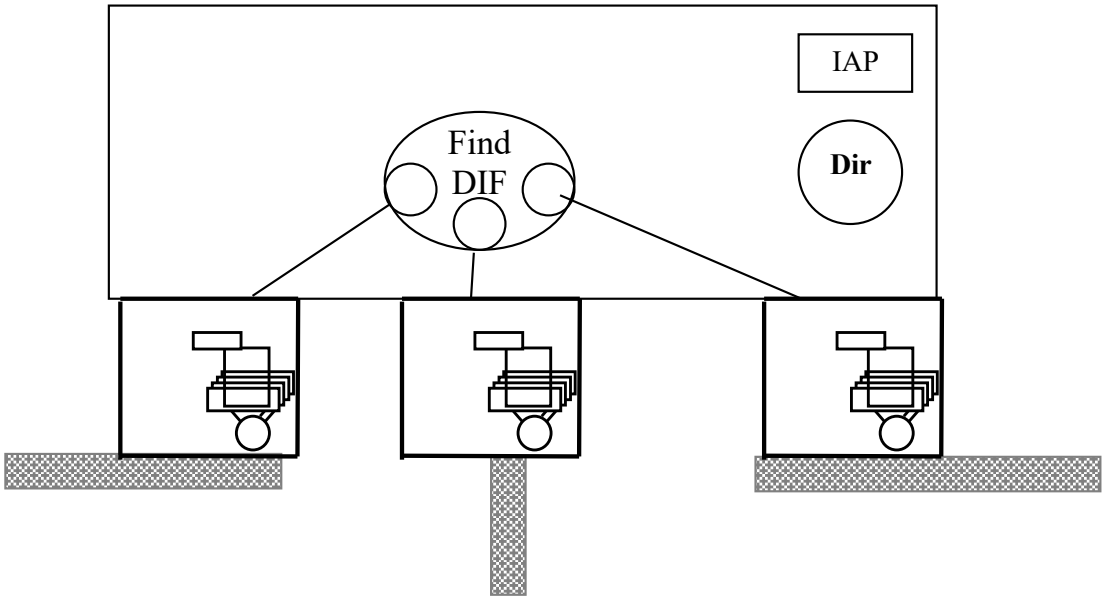


- Local names of multiplexing applications managing interfaces suffice to label interfaces. (With point-to-point wire, PDU goes in one end, it *has to come out* the other end!)
  - Each one can number them however they want. They are only used locally.
- A Local Table, keeps track of which applications are reachable by which interface.
- Same local names can be used to keep track of which EFCP-instances (port-ids) are bound to which Multiplexing Application..
- *But the important thing is so far only local identifiers are needed.*

Replication Entails More Management

- Application naming becomes more elaborate:
  - We need to support multiple instances of the same application.
  - The application names will want to be location-independent as in operating systems.
- IPC can find the destination by choosing the appropriate interface.
- But if enough of them, create a 'database,' i.e., a directory, to remember what is where, i.e., what application names are at the other end of which interfaces.
  - Likely to use a combination of cache and query. Use IAP and remember the results. (As it grows, may require further organization to be efficient.)
- With N-1 destinations, may need to coordinate resource allocation information within our distributed IPC Facility. So, some sort of Flow Manager would be a good idea.

But Applications Shouldn't Have to Know Which Wires to Use!



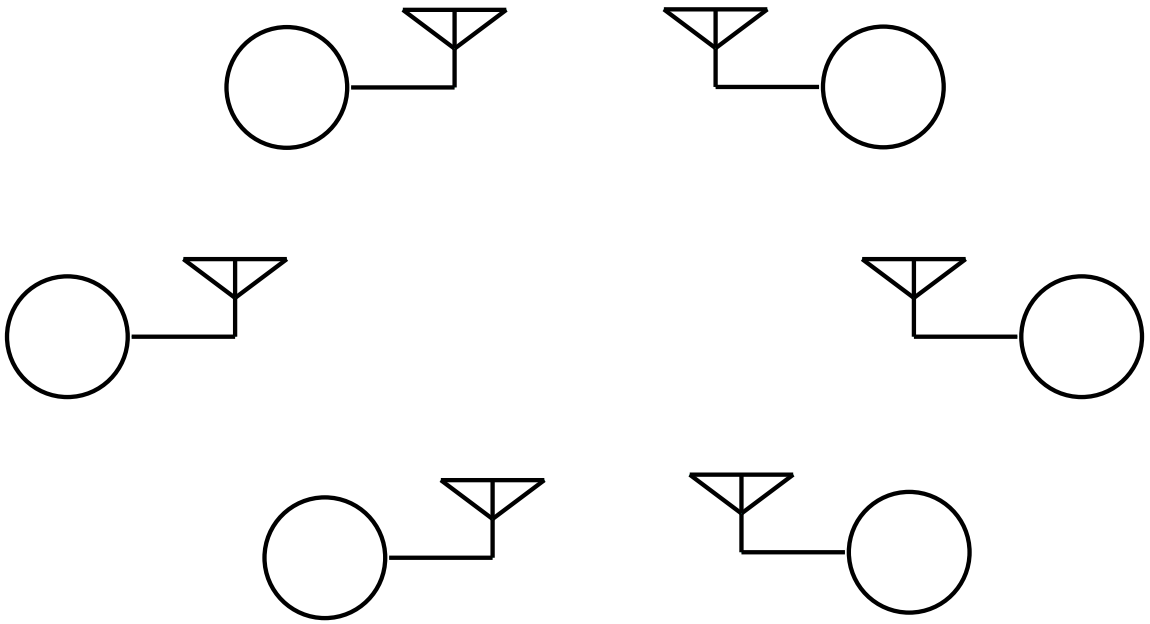
So, with one Distributed IPC Facilities (DIF) for each Interface (wire), we need a function to figure out which wire the requested application is on. This preserves the API the application sees to make it location-independent and look as much like normal file operations as possible.

BUT

- This fully connected graph approach isn't going to scale very well and is going to get very expensive.
  - And not everyone needs to talk to everyone else all the time.
- Need to do something better.

- Hey Day, it's the 21st Century!
- What about Wireless? Everything is wireless.
  - (Yea, okay. But wireless has its own constraints.)

N Systems Using Wireless



This is certainly cheaper. Instead of each node having (N-1) interfaces, each one has 1 interface. However, (why is there always a however?)

Wireless Introduces New Considerations: I

- Wireless does provide a marked decrease in the complexity.
  - Before each node had (N-1) layers of 2 members each,
  - Now each node has one layer of N members.
    - To the user of this layer, all nodes are one hop away.
- Whenever there is a layer with more than 2 members either with shared media or relaying, we need to label the PDU as to who it is for.
  - Every PDU must carry an identifier to indicate the destination of the PDU. (sometimes called an address)
    - If the layer is a shared media, every node should see every PDU sent, each node need only *recognize* which PDUs are for it.
    - If the layer relays, the identifier indicates where its destination is and when it has reached it.
- We will need to add source and destination "addresses" to all PDUs.

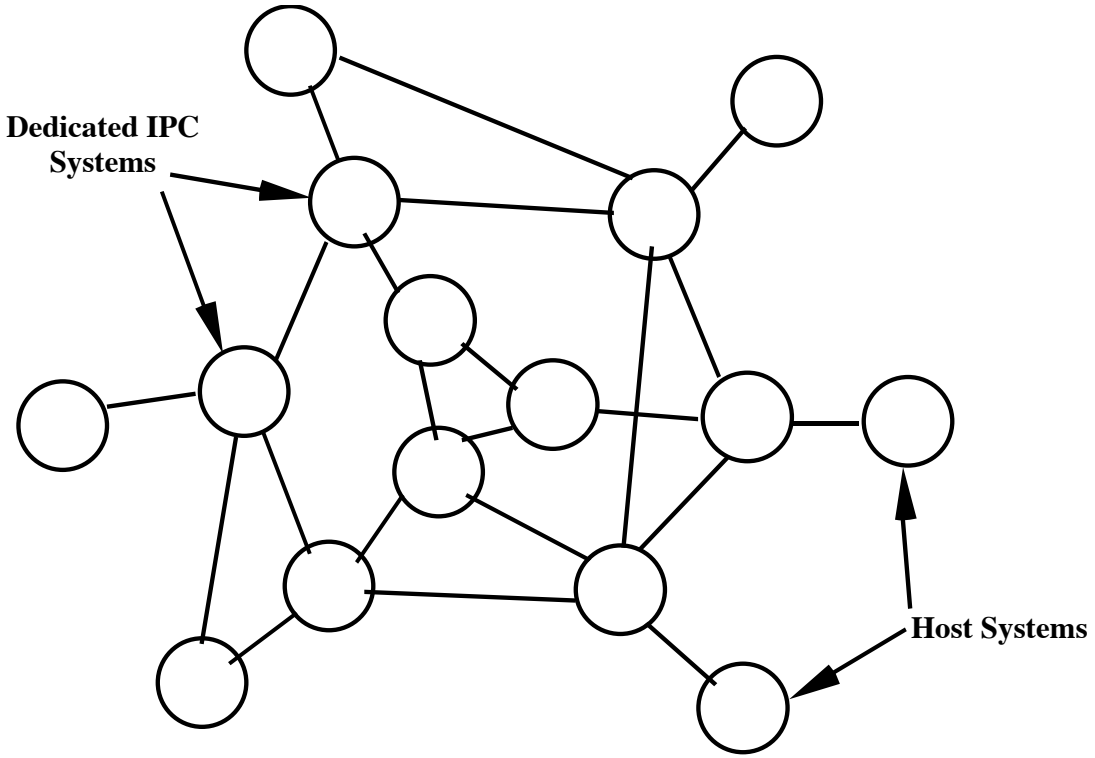
Dest Addr	Src Addr	Dest-port	Src-port	Op	Seq #	CRC	Data

The PDU must be expanded one more time.

Wireless Introduces New Considerations: II

- However, wireless comes with a cost, almost all of which derive from it being a shared medium:
  - It will have limited range.
  - Wireless is a notoriously hostile environment for corrupting PDUs.
  - There is contention for sending PDUs.
    - There are methods that get utilization beyond the basic 36%, but contention is still an issue and greatly reduces the capacity of the network.
- Wireless isn't going to scale well to large networks.
- Consequently, wireless tends to be used at the periphery of a wired network that relays.
  - But we would like to keep the property that makes all members of a layer appear directly connected (one hop away).
- So, what about N Systems Connected with Relays?

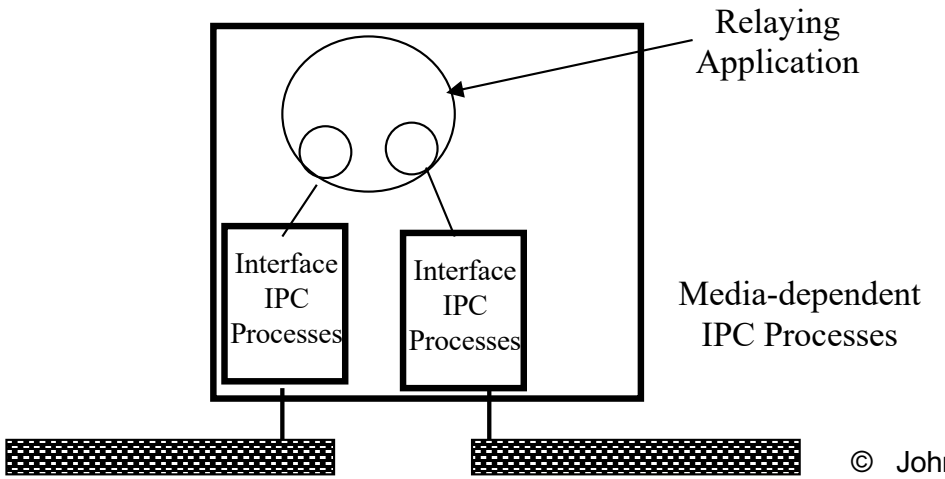
5: N Systems with Relays
(Communicating On the Cheap)



By dedicating systems to IPC, reduce the number of lines required (this requires 19 (shorter) lines, previous case needed 30) and recognize that not everyone talks to everyone else the same amount and there is still good resiliency to failure.

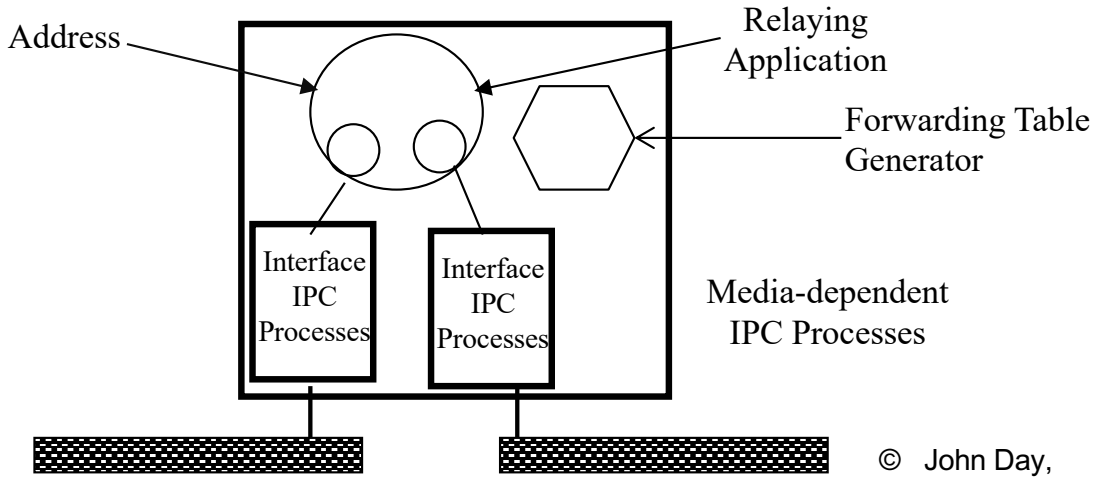
N Systems with Relays: I

- We will need systems dedicated to relaying and multiplexing.
- A relay will consist of:
  - One of our layers for each interface ("wire") that is connected to it.
    - The media may be a variety of technologies with different characteristics
    - Each Interface Layer will use a protocol suitable for that media.
    - If a wire, addresses won't be necessary; local ids will suffice as before.
    - If wireless, addresses will be needed.
    - Should keep errors from the media as low as practical.
    - Want to limit media errors to the smallest scope reasonable.
  - A Relaying application to forward the PDUs.
- Yes, but how does it know which way to send the PDU?



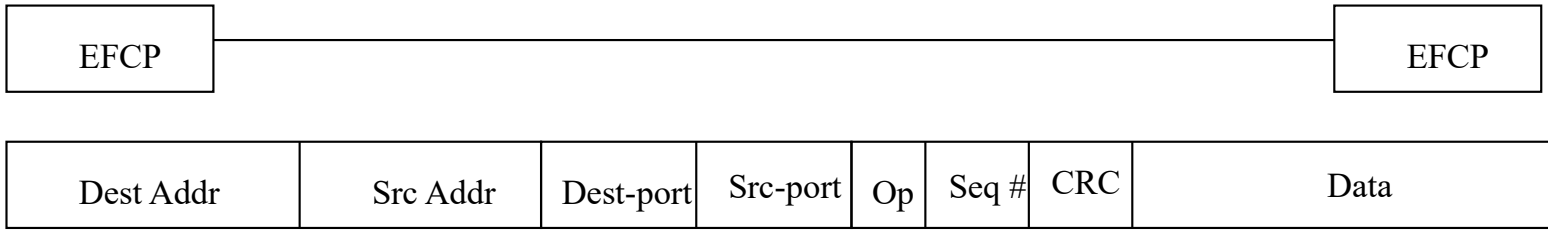
N Systems with Relays: II

- How Does it Know Which Way to Send the PDU?
  - A wire from a host no longer goes to just one destination but is on the path to many destinations.
  - We are creating a layer with more than 2 members.
  - Must assign addresses to all End Systems and Relays.
  - The Protocol for this Layer will carry the addresses.
  - The Relaying Application will use the addresses to determine:
    - is this PDU's destination here or
    - does it have to be forwarded and if so, where?
  - Need to generate a forwarding table for the Relaying Application to use.
    - Traditionally this has been called routing, which exchanges information on connectivity.
- But Relays create new problems

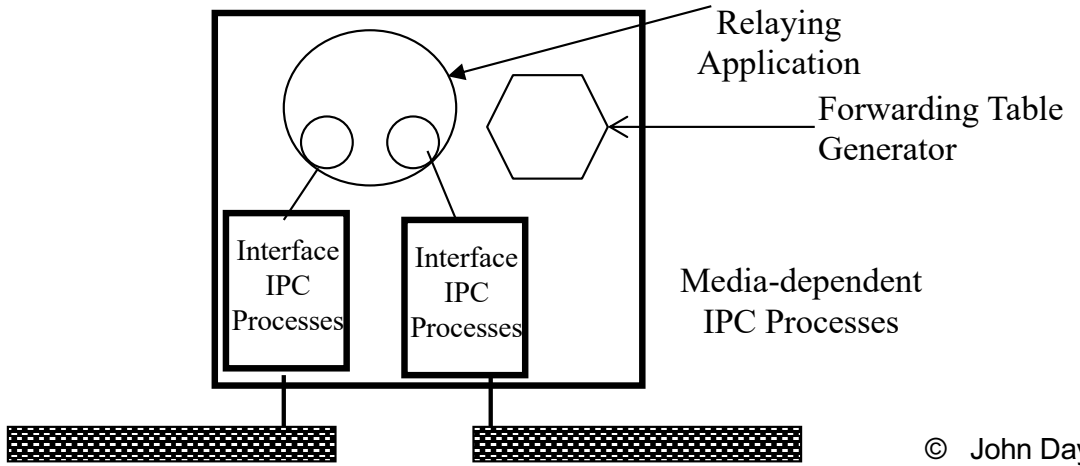


N Systems with Relays: III

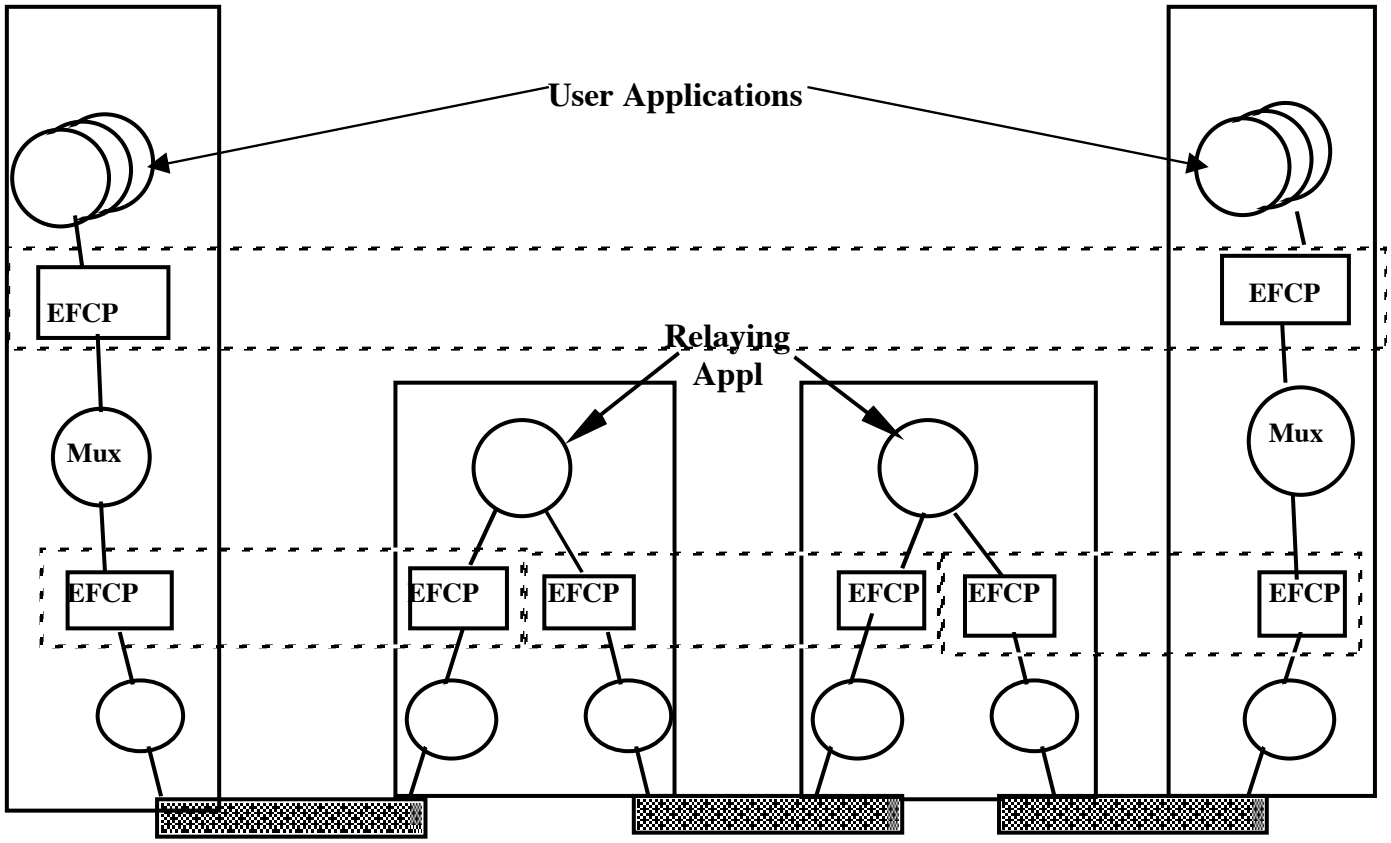
- But relaying systems create problems too.
  - Can't avoid transient congestion from time-to-time.
  - Rare but Annoying bit errors can occur in their memories.
- Will have to have an EFCP operating over the relays to ensure required QoS reliability parameters.
  - We have constructed a layer of greater scope that to the users of the layer makes every node appear directly connected.



Typical PCI for an Error and Flow Control Protocol with Relays



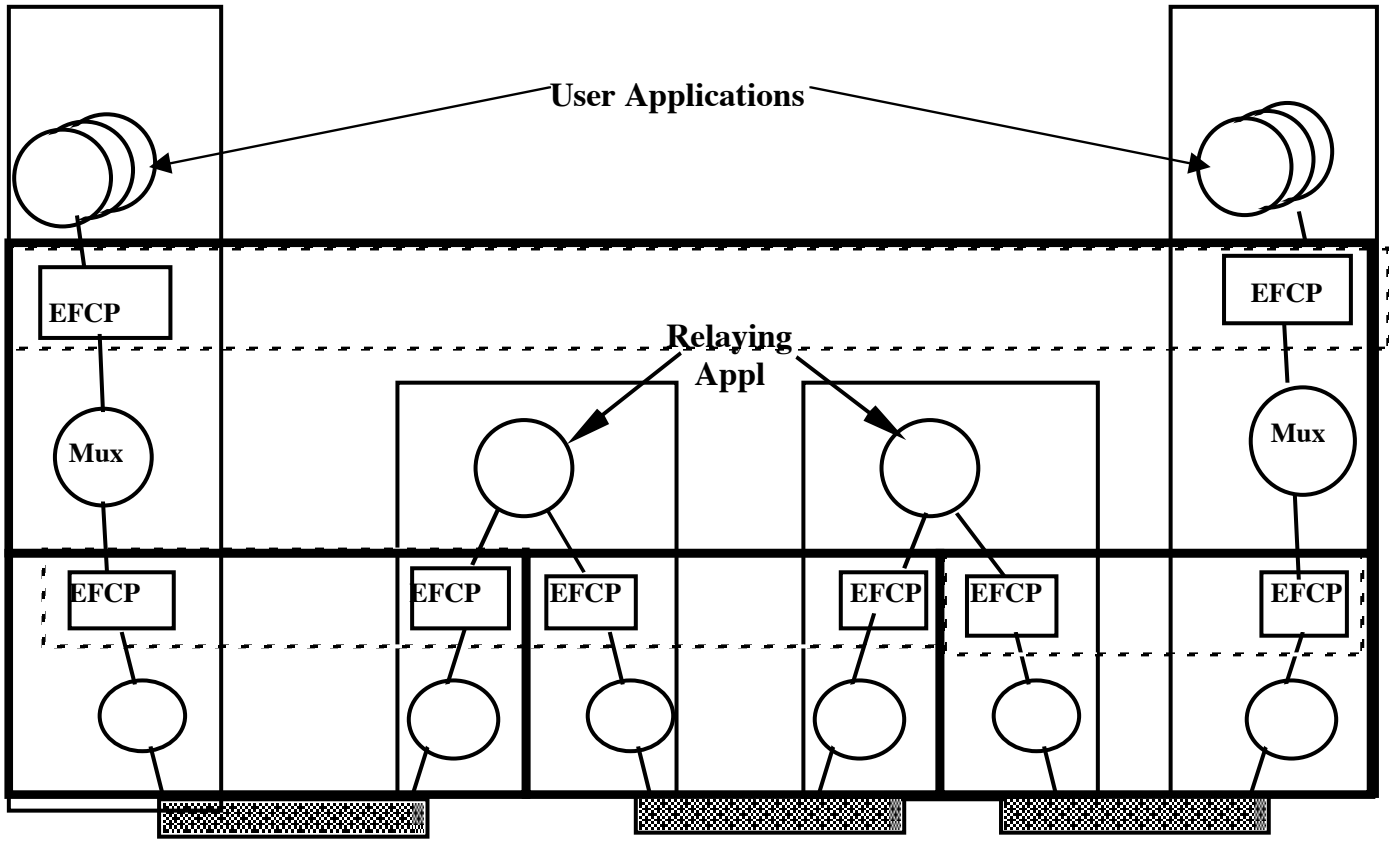
The Big Picture



This should look familiar.

The IPC Model

(A Purely CS View)



Conclusions

- Networking is IPC and only IPC.
- Layers are not divisions of functionality, but divisions of allocation.
  - All Layers have the same functions, they just operate on a different range of the problem.
- A Distributed IPC Facility (a layer) consists of muxing apps, EFCP, and layer management, e.g., their routing and resource allocation.
- Application-names and port-ids are the only externally visible identifiers.
  - Port-ids have only local significance.
  - The only information an application must and should know to establish communication is the destination application-name and its port-id.
  - Port-ids were combined to form a connection-id that is unique within the source/destination address pair.
- We have uncovered two Principles:
  - When there can be more than one flow between two systems, a connection-id is required on the PDUs.
  - When an IPC Facility (a layer) has more than two members, PDUs require name for the IPC Process is needed, sometimes called an address.

The (really) Important Thing about Layers

Remember This ;-)

- A Layer is a locus of distributed shared state

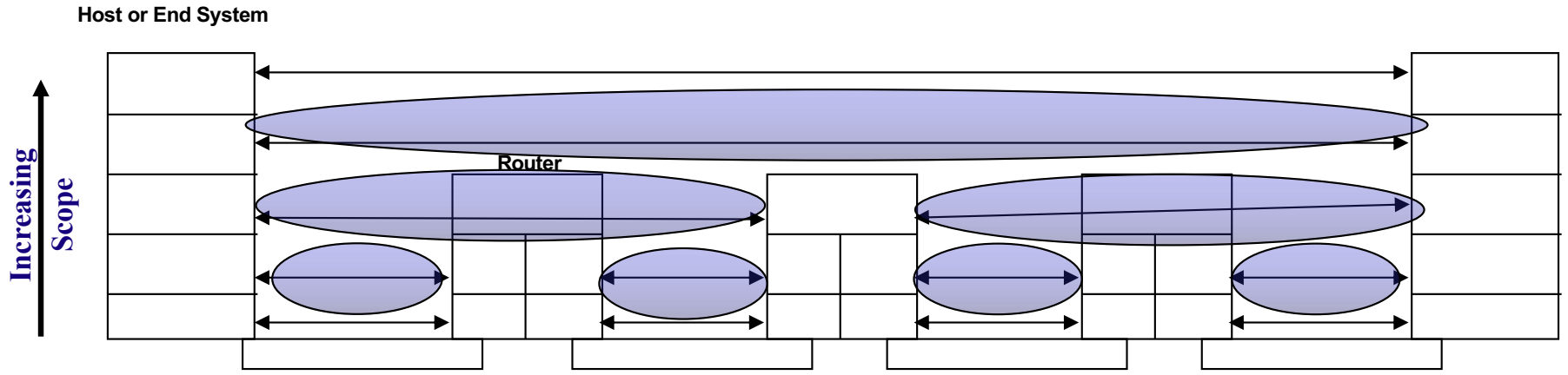
  In OSs, Layering was a convenience, here it is a necessity.

- Different Layers should have different scope

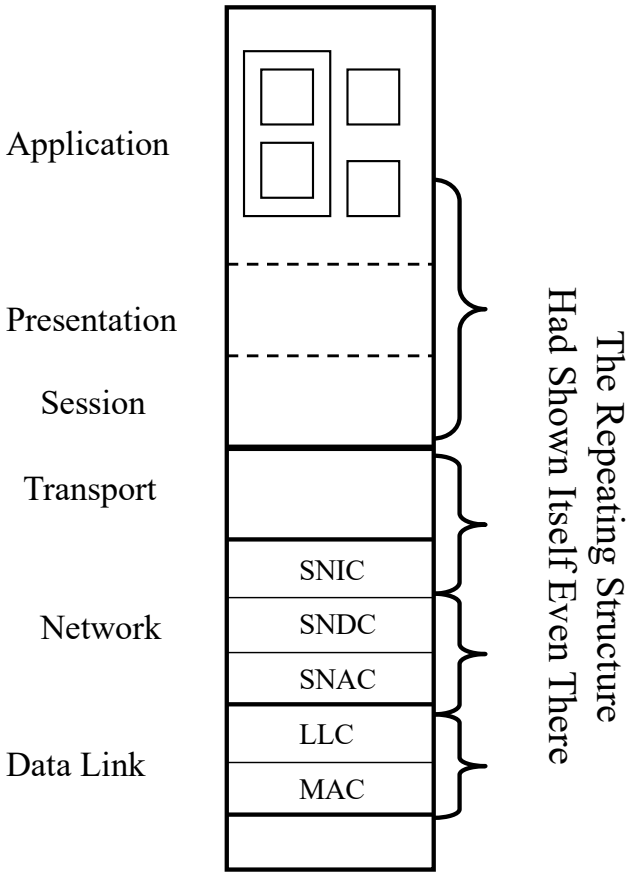
  Either in terms of number of elements or range of operation

  There are multiple layers at the same rank.

- Notice something:
  - As a stack, some functions repeat in every layer. Looks redundant
  - As layers, functions are being performed over different scopes.
    - They don't repeat at all. Haven't done those functions for that scope.



We Should Have Seen It Sooner



The Rules for Layering

- A locus of shared state of a given scope constitutes the necessary and sufficient conditions for a layer.
- There may be multiple layers within the same scope.
  - The functions within layers of the same scope must be independent.
- Functions do not repeat in the same scope.

The IPC Model is the Framework

- from which we will derive our principles. We will return to it again and again to see what it tells us.
- Everything we need follows from this: security, mobility, enrollment, multihoming, quality of service, multicast, etc.
  - And it follows naturally, with nothing else needed.
- The model and what we learn from it will serve as a basis for analyzing existing solutions.

- But there are three other results we need to complete the model:
  - The Nature of Applications and their Relation to IPC.
  - The Separation of Mechanism and Policy, and
  - Watson's results on the synchronization in protocols.

  - Something to do next week.

Not Bad for a Subject We All Knew

A Nice Little Return on Investment

- The Moral of the Story
  We should Practice What We Preach! ;-)

- Advice to every freshman engineer/scientist.
  - Never skip steps. You make mistakes when you skip steps.
    - Like dropping signs or not noticing common structures
  - Always reduce the algebra before doing the arithmetic.
    - It will reveal the fundamental structure of the problem.
- This was a bit tedious, but we learned some things by taking more time.
- Often when an exercise like this is suggested, the response is,
  - 'We already know all that' or 'We don't have time for that'
- No, we only *think* you know all of it.
- There isn't time not to do it!