Tables & Data Structures Management

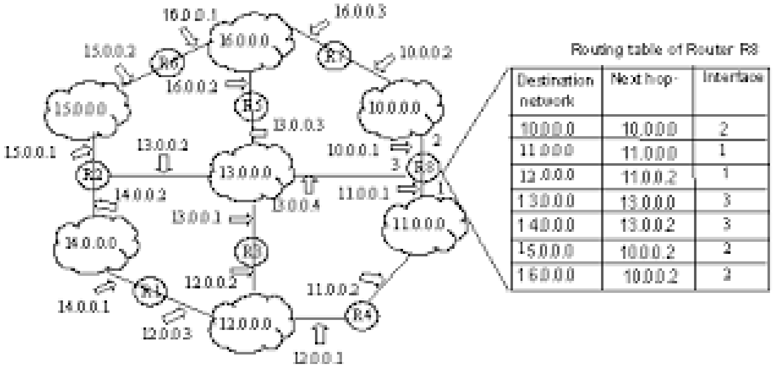
CPE 545

Chapter Summary

- This chapter details some of the tables and other data structures typically used in communications systems and discusses the related design aspects.

Tables and Data Structures

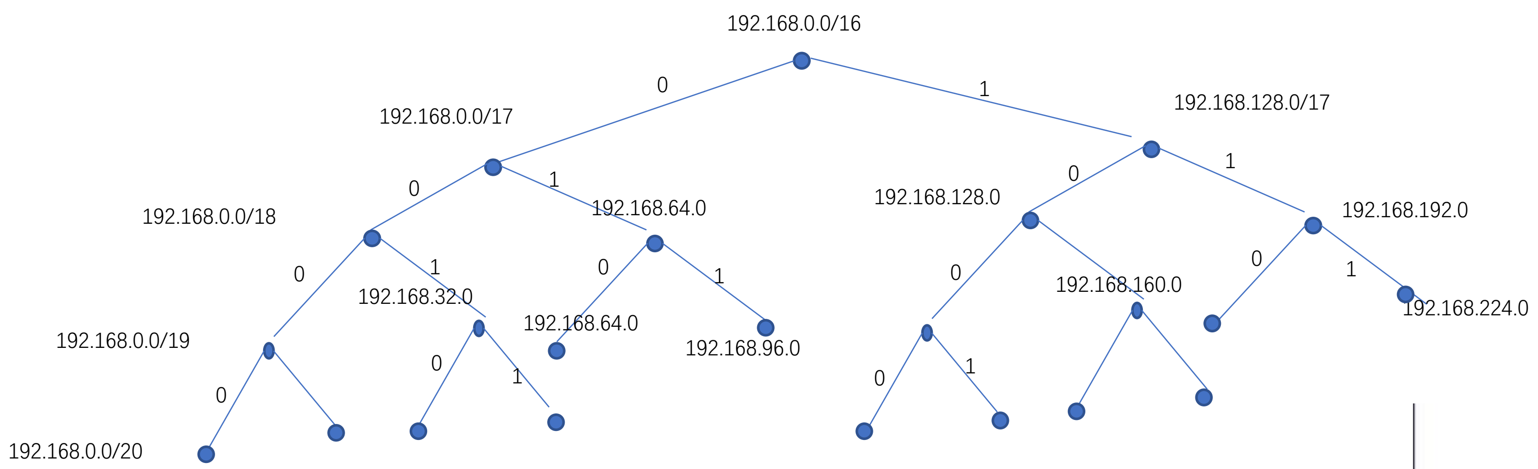
- Issues with tables
  - Are referenced for reading and writing frequently and hence storage and access methods have to be optimized
  - Can be stored in different parts of memory -> DRAM, SRAM,…
  - Can be organized according to the access method
    - Layer 3: Radix tree for variable length prefix
    - IP routing lookups must find the routing entry with the longest matching prefix
    - Layer 2: hashing for fixed-length ind

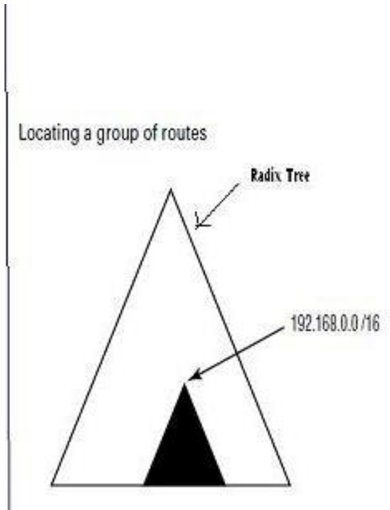


Routing table of Router R8

Destination network	Next hop	Interface
10.0.0.0	10.0.0.0	2
11.0.0.0	11.0.0.0	1
12.0.0.0	11.0.0.2	1
13.0.0.0	13.0.0.0	3
14.0.0.0	13.0.0.2	3
15.0.0.0	10.0.0.2	2
16.0.0.0	10.0.0.2	2

Radix tree


192.168.0.0/16
0
1
192.168.0.0/17
192.168.128.0/17
0
1
0
1
192.168.0.0/18
192.168.64.0
192.168.128.0
192.168.192.0
0
1
0
1
0
0
1
192.168.32.0
192.168.160.0
192.168.224.0
192.168.0.0/19
192.168.64.0
192.168.96.0
0
0
1
0
1
192.168.0.0/20


Locating a group of routes
Radix Tree
192.168.0.0/16

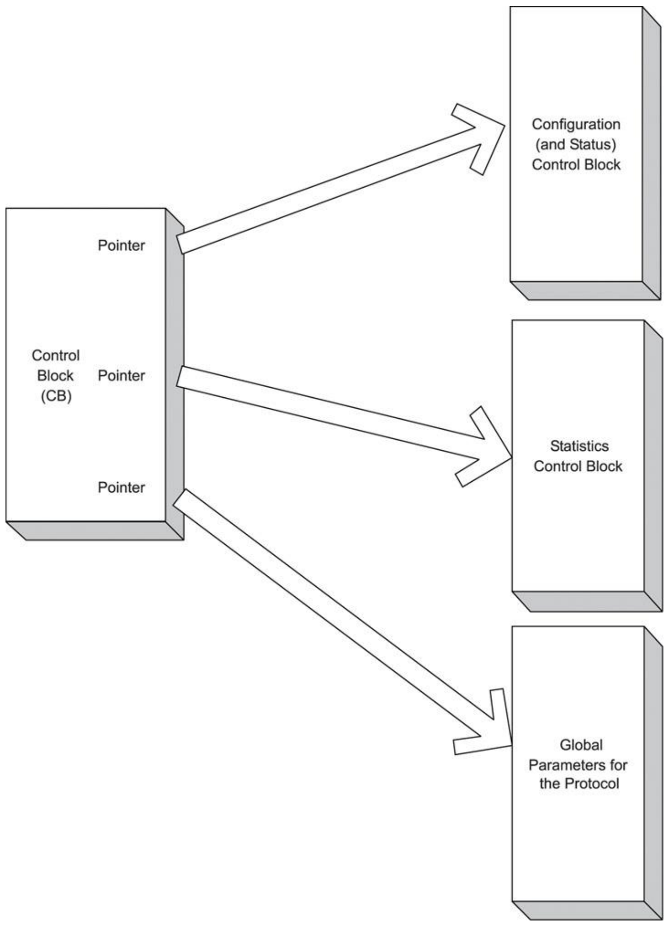
Tables and Data Structures

- Tables for management are used for:
  - Configuration -> info used to set parameters
  - Control -> read/write info to change behavior of communication
  - Status -> details about current state of operation
  - Statistics -> read only monitored counters (e.g. received packets)

Partitioning Tables - Design Approach

- Information in tables can be global or per port
- Root data structure of a protocol or system module is called a control block (CB)
- CB serves as anchor block to access other blocks like config block
  - CB points to config, control, status and statistic variables
  - This scheme allows for more flexibility in memory partitioning

Partitioning Tables - Design Approach


Pointer
Control Block (CB)
Pointer
Pointer
Configuration (and Status) Control Block
Statistics Control Block
Global Parameters for the Protocol

Protocol control block and related blocks for IP.

```
typedef struct ControlBlock {
   BOOLEAN  IPInitialized;
   BOOLEAN  IPBufferInterfaceInitialized;
   BOOLEAN  IPTimerInterfaceIntialized;
   IPConfigBlock *pConfig;
   IPStatsBlock  *pStats;
}IPControlBlock;

typedef struct _IPConfigBlock {
   BOOLEAN ipForwardingEnabled;

   /* to insert in IP packet */
   UINT2   u2TTLValue;

   /* which ICMP messages to respond to*/
   UINT2   icmpMask;
   .........
          .........
  } IPConfigBlock;
```

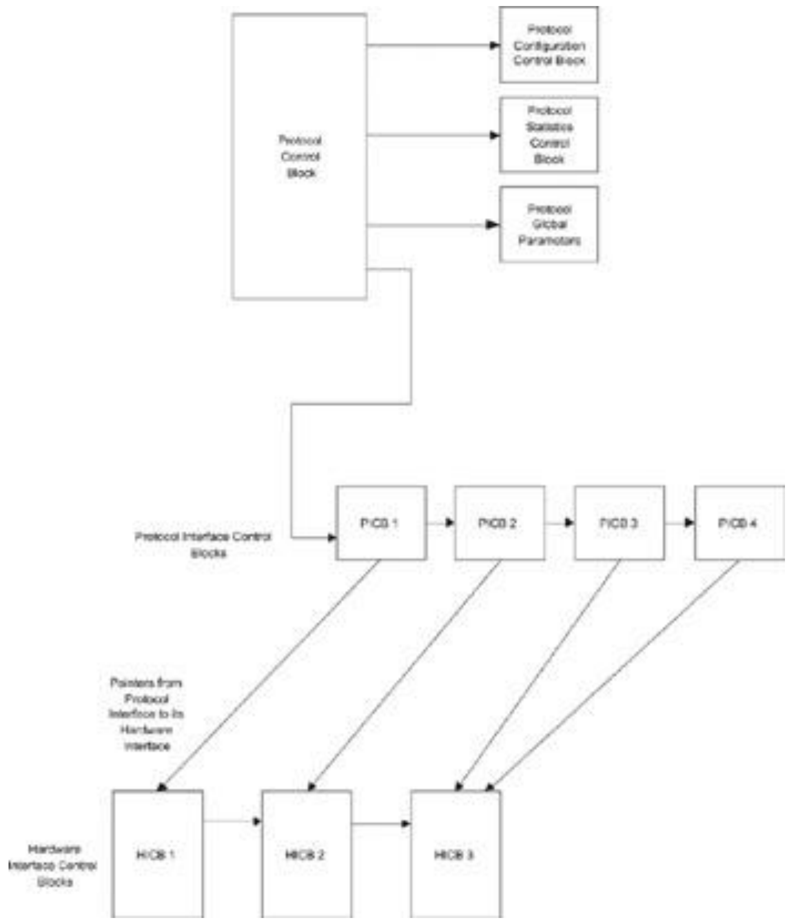
```
typedef struct _IPStatsBlock {
        UINT4        ipInReceives;
        UINT4        ipInHdrErrors;
        UINT4        ipInAddrErrors;
           .........
           .........
        UINT4        ipOutDiscards;
        UINT4        ipOutNoRoutes;
           .........
           .........
           .........
           .........
} IPStatsBlock;
```

Partitioning Tables - Design Approach

- Logical Interfaces
  - Protocols need interface specific configuration and Interface Control Block (ICB) handles this information – (e.g. number of OSPF packets recv'ed)
  - Two types of ICB's
    - H/W interface control block (HICB) - Physical Interface
    - Protocol interface control block (PICB) – Logical Interface
  - HICB represents control, status, of a hardware port while PICB represents control, status, statistics of a protocol on the specific interface
  - Two types of ICB provide for flexibility since:
    - More than one protocol can be enabled on a hardware port
    - More than one logical interface can be specified on a physical interface

Partitioning Tables - Design Approach

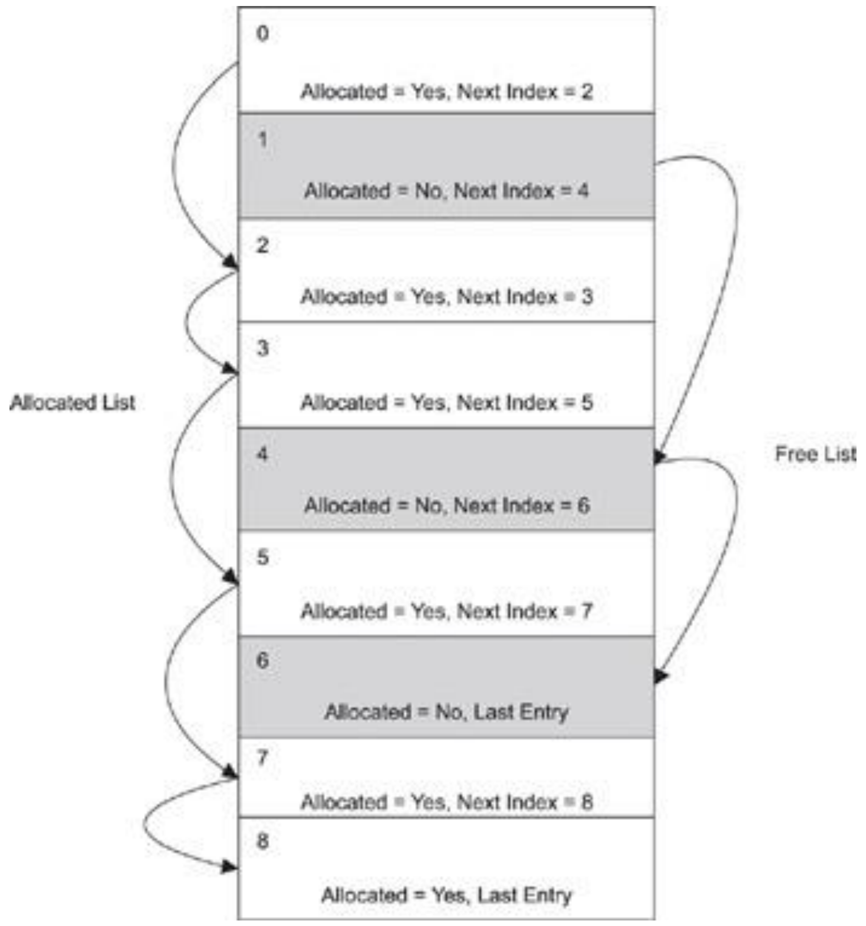
- *There is one HICB per physical port*
- *There is one PICB per logical interface for the protocol.*
- Each PICB has a pointer to its related HICB and is also linked to the next PICB for the same protocol


Protocol Control Block
Protocol Configuration Control Block
Protocol Statistics Control Block
Protocol Global Parameters
Protocol Interface Control Blocks
PICB 1
PICB 2
PICB 3
PICB 4
Pointers from Protocol Interface to its Hardware Interface
Hardware Interface Control Blocks
HICB 1
HICB 2
HICB 3

CB allocation and initialization

- Protocol software allocates a PICB, links it to the Ethernet interface's HICB,
- PICB contains basic parameters for protocol
  - gets allocated when the SNMP manager configures parameters for the protocol (e.g. IP address)
- If and only if all basic parameters in PICB are set, is the protocol ready and enabled

ICB Implementation - Array

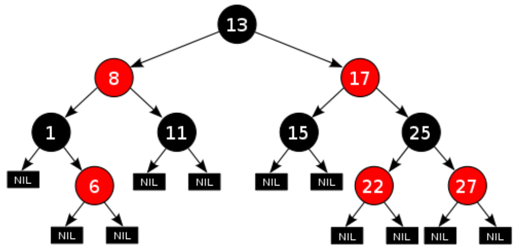

0
Allocated = Yes, Next Index = 2
1
Allocated = No, Next Index = 4
2
Allocated = Yes, Next Index = 3
3
Allocated = Yes, Next Index = 5
Allocated List
4
Allocated = No, Next Index = 6
Free List
5
Allocated = Yes, Next Index = 7
6
Allocated = No, Last Entry
7
Allocated = Yes, Next Index = 8
8
Allocated = Yes, Last Entry

Speeding up Access

- Three methods for speed up
  - Optimized access mechanisms
  - Hardware support
  - Caching

Speeding up Access

- Optimized Access Methods
  - Tree structure is more efficient for access than a linked list structure
    - Allows access to routing table entries stored in leaf nodes in much shorter hops than traversing the entire linked list (O(log *n*))
    - Red and black tree (Self balancing binary search tree)
      - A node is either red or black.
      - Every red node must have two black child nodes and vise versa.
      - Every path from a given node to any of its descendant leaves contains the same number of black nodes.
  - Hashing tables also allow for efficient storage



- Hardware support
  - Ethernet controller can contain hardware hashing mechanism to match destination MAC address
  - Content addressable memory (CAM) provides for parallel searches of a table or other data structure which leads to faster access

Homework

- How do you handle the situation when a hash algorithm does not create a unique hash for all the keys?

Speeding up Access

- Caching
  - Caching provides for storing a subset of large data structure in a high speed memory module for fast access
  - Subset determination algorithm is critical to success of caching for faster access
    - Cache most recently seen destination addresses
    - An entry is accessed more than once in last few transactions
    - Static configuration (fixed tables)
  - Different techniques exist for replacing cache entries
    - Timing out entries - if a cache entry is not used for a period of time, it is replaced
    - Removal of entries can be active or on demand
    - Priority of entries (set priority for removal from cache at the time of caching)
    - LRU (Least Recently Used) scheme can be used when cache is full

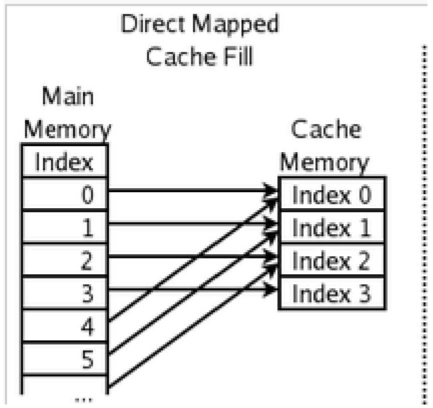
Caching

- Copy a data block (cache line) of 64 bytes from memory into cache.
- We need 6bits to specify each of the 64 bytes.
- Suppose you want to access address $\mathbf{A}_{\mathbf{31-0}}$ `0x3F80260A`
  - We copy the data in addresses $\mathbf{A}_{\mathbf{31-6}}$ `0x3F802600`: $\mathbf{A}_{\mathbf{31-6}}$ `0x3F80263F` from RAM to the cache :
- The upper 26 bits is called *tag*. The lower 6 bits considered the *offset* into the cache line.
- Given the tag and the offset, we know what address the data at a given offset is.

Caching

- If we have a cache memory of size 128KB, where each cache line is 64 bytes, we will have a total of 128k/64 = 2048 cache lines
- To index 2048 cache lines we need 11 bits

tag	index	offset



- 11 bits used for index (identifies the cache line within the cache)
- The upper 15 bits used for *tag (matches the decoded address in RAM if data is in cache)*.
- The lower 6 bits considered the *offset* into the cache line.
- Every cache block has associated with a tag address

Caching management

- **For each cache line there is a:**
  - **V**alid bit: Indicating whether the slot holds valid data. If **V = 1**, then the data is valid. If **V = 0**, the data is not valid. Initially, it's invalid. Once data is placed into the slot, it's valid.
  - **D**irty bit: This bit only has meaning if **V = 1**. This indicates that the data in the slot has been modified (written to) or not. If **D = 1**, the data has been modified since being in the cache.
  - Every cache line has associated with it at least the Dirty and Valid bits, and a tag address

Caching management

- **Cache Hits and Cache Misses**
  - When you look for data at a given address, and find it stored in cache, then you have a *cache hit*. If you don't find the data, then it's a *cache miss*.
  - You want to maximize cache hits, because then you have much improved performance (speed).
- **What Happens in a Cache Miss**
  - If the address (and its contents) are NOT in the cache, then you must access it from main memory. This means that you must copy the data from memory to the cache.
  - Since the cache is a small subset of main memory, there may be data that you need to remove from the cache

Caching types

- Suppose you need to modify data in the cache. There are two ways to do this:
  - **write-back** This says that you update main memory only when the cache line is evicted. Otherwise, only update the cache.
  - **write-through** This says that you update main memory at the same time you update cache.

Caching types

- **write-back**
  - This approach creates an inconsistency between cache and memory. Cache has one view of memory (which is accurate), while main memory does not have the latest updates.
  - Problematic if you had more than one CPU and cached data is in shared memory.
  - Write-back is convenient because you don't have to access main memory, which can be slow.
- **write-through**
  - The disadvantage occurs when there are many writes to memory. This can cause a backlog waiting for the writes to occur.
  - The key is to avoid waiting for the write to memory to complete by writing to memory in the background.
  - The advantage of write-through is that you don't have to worry about memory consistency, since cache and memory should have the same values.

Homework

- In the cache example above, how many bits of the address line is required to map the data in cache line to the memory blocks in the RAM?
- How many 32 bit registers you would need in the cache controller to manage this 128KB cache. i.e:
  - Determine if it is a cache hit or miss, valid or invalid, dirty or clean

Table Resizing

- Dynamic resizing not recommended because it often requires re-allocation of new array and copying the data for contiguity:
  - Reference modification
    - Refers to change in memory pointers during execution
  - Peak memory requirements
    - Refers to memory needed to move data from old to new table
- Recommend approach -> Table size is a boot parameter and a change in size takes effect only when system is re-booted

Table Access Routines

- Not recommended that tables be accessed globally by all modules
- Parameters within a table should not be accessed directly and instead use routines to encapsulate the data
- For this reason, the access routines should protect critical sections using mechanisms like semaphores
- Another benefit of these access routines is producing reentrant code
  - Code is reentrant if it can be called by multiple callers and yet maintain memory consistency across each individual context
  - Reentrant code is extremely useful in designing embedded systems

Router Management

- SNMP Management
  - Agent's front end is used for PDU processing while back-end performs validation and determines actions to be performed
- CLI based Management
  - Inputs acquired from user and verification performed by the agent
- HTTP based management
  - Uses an embedded web server to receive and send HTML data that contains management information
- All three methods have common back-end processes but different at the front-end

CLI

- CLI or Command Line Interface
  - Command line Interface is either text string based or character key based.
  - Can be accessed via Serial port or Telnet session
  - Provides direct access to the directory structure of the file management system and hence used by power users

SNMP

- SNMP or Simple Network Management Protocol
  - Most Dominant protocol in network management
  - Most routers and switches implement CLI and SNMP agent
  - Model assumes the existence of Managers and Agents
  - Agent is a program which communicates with the Manager on one side and with Device or Application on the other side.
  - The SNMP manager monitors and controls the managed device via the access provided by agent .

SNMP

- SNMP facilitates communication between a managed device (a device with an SNMP agent) and an SNMP Manager .
- Communication is via SNMP Protocol Data Units (PDUs) and are encapsulated in UDP packets
- A management information exchange can be initiated by the manager (via polling) or by the agent (via a trap)
- Agent listens for requests and replies to them over UDP port 161 and reports asynchronous traps on UDP port 162

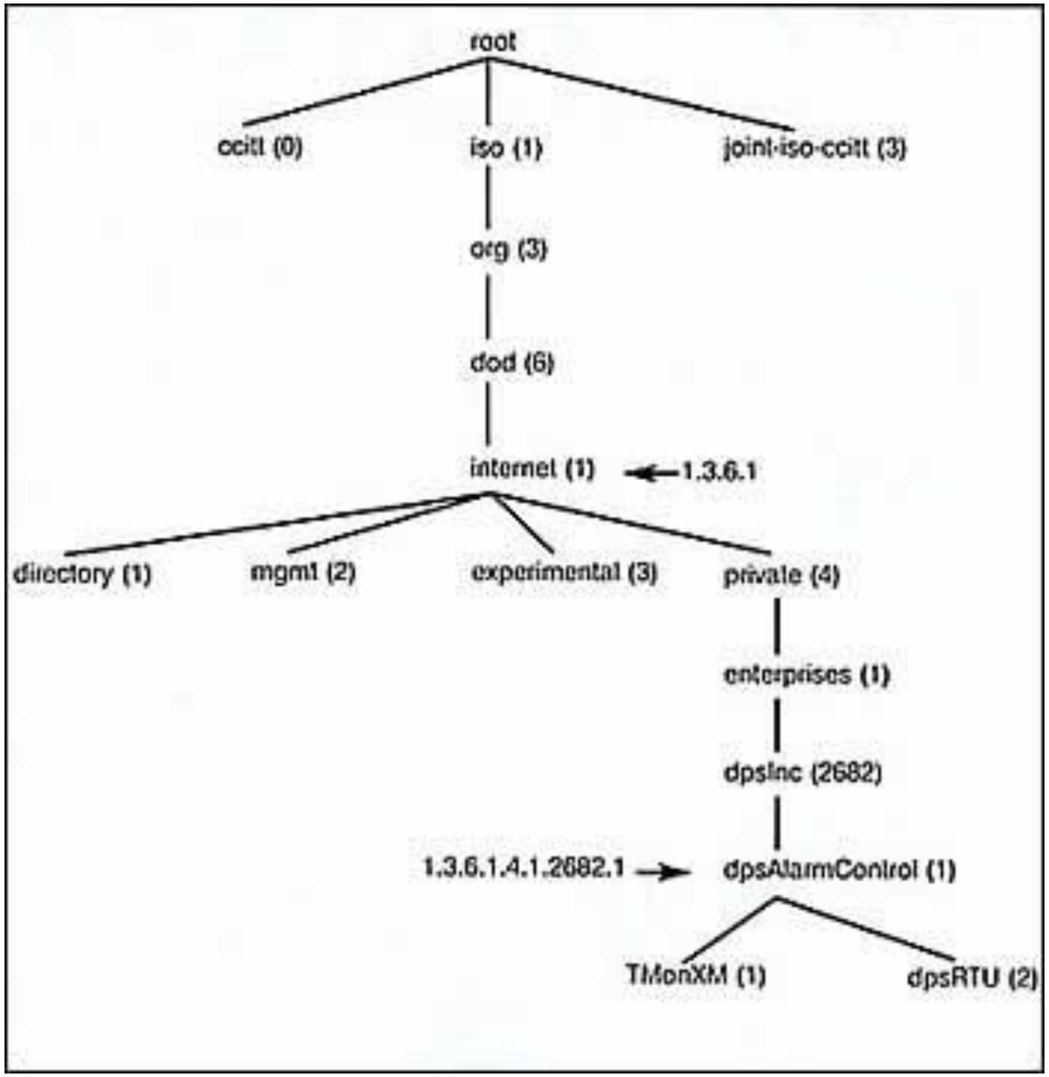
SNMP

- Operations permitted between managers and managed device:
  - GET: Operation performed by manager to obtain information from the agent about an attribute of a managed object.
  - GET-NEXT: : Operation performed by manager to obtain information on the next object in the tree of objects on the managed device.
  - GET-BULK: : Operation performed by manager to obtain information about a group of data from the agent. This is not possible in the case of SNMP V1.
  - SET: : Operation performed by manager to set (or write) the value of an attribute of a managed object.
  - TRAP: An asynchronous notification sent by the agent to the manager; telling it about some event on the managed device.

SNMP

- What is MIB (Management Information Base)
  - MIB is a document about the device (or application).
  - Allows for a remote management of a device (or application) and defines a set of variables that should be published outside (to the Manager).
- Each variable is assigned a unique identifier in SNMP that is called an object identifier (OID).
  - The uniqueness is maintained all over the world.
- The format of OID is a sequence of numbers with dots in between.

SNMP



The branch of the MIB object identifier tree that represents managed elements used by DPS Telecom equipment

MIB

- Management Information Base
  - Specified by bodies like IETF and define management variables
  - RFC 1213 is an MIB for IP forwarding and also called MIB-II and can be managed by an external SNMP manager
  - Configuration variables can be stand-alone (scalar) or part of a table
  - Always use SNMP MIB table as a checklist when implementing data structures

MIB Example

- Management information base (e.g. ipAddr table)

```
ipAddrEntry OBJECT-TYPE
    SYNTAX  IpAddrEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
         "The addressing information for one of this
         entity's IP addresses."
    INDEX   { ipAdEntAddr }
    ::= { ipAddrTable 1 }

 IpAddrEntry ::=
    SEQUENCE {
       ipAdEntAddr
          IpAddress,     // IP address of interface
       ipAdEntIfIndex
          INTEGER,       //Interface Identifier
       ipAdEntNetMask
          IpAddress,     //Net mask for interface
       ipAdEntBcastAddr
          INTEGER,       //Broadcast address used on interface
       ipAdEntReasmMaxSize
          INTEGER (0..65535) //Max Size of reassembled IP
                        //packet
    }
```

Management Table Example

- **Address table (Interface table).**

Addr		IfIndex	NetMask	ReasmMaxSize
10.0.0.1	1	255.0.0.0	10.255.255.255	15000
20.0.0.1	2	255.0.0.0	20.255.255.255	18000
30.0.0.1	3	255.0.0.0	30.255.255.255	15000

- The Addr field is the IP address for the specific interface (identified by an ifIndex).
- The NetMask, BcastAddr and ReasmMaxSize are configurable fields in a table entry.
- These variables are also known as "Read–Write" variables as opposed to "Read Only" variables, which cannot be configured by the manager.

Agent to Protocol Interface

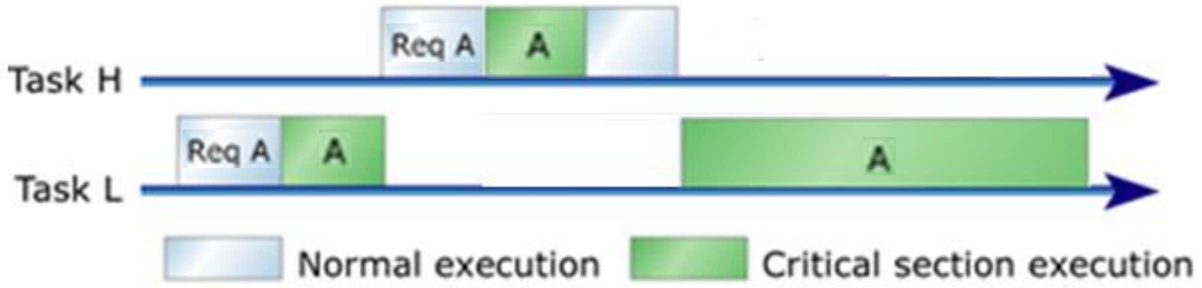
- Low level routines provide access to data structures maintained by the protocol task
- But executed under the context of agent task which leads to a couple of related issues:
  - Priority of the agent task is the lowest within the protocol implementation
    - Because management operations are considered secondary to the main function of the system which is switching and routing
    - Therefore the low level routines execute at a priority lower than the protocol task.
    - Therefore simultaneous access of data structures by protocol task and low level routine creates conflicts.

Agent to Protocol Interface

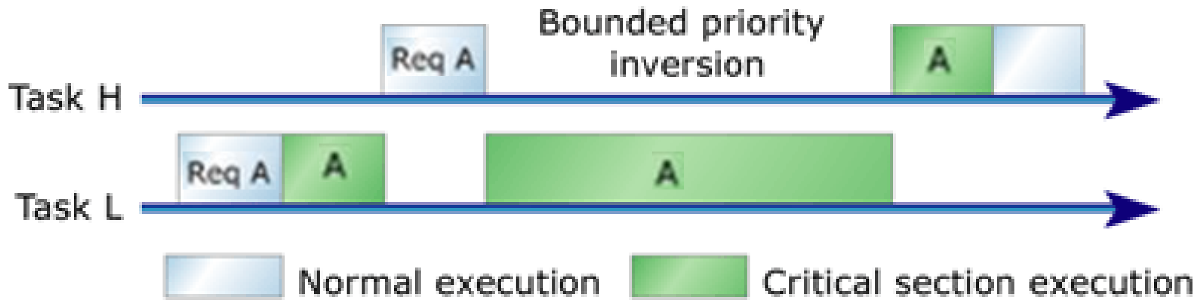
- Solution 1: Mutual exclusion using semaphores
  - Requires significant modifications to protocol code
  - Susceptible to bounded and unbounded priority inversion and possible deadlock
- Solution 2: Lock agent task during execution, so it is not preempted by protocol task
  - Can also be realized by temporarily raising priority of agent task during access
  - Priority inheritance

Bounded Priority Inversion

The task makes a request for Resource A.

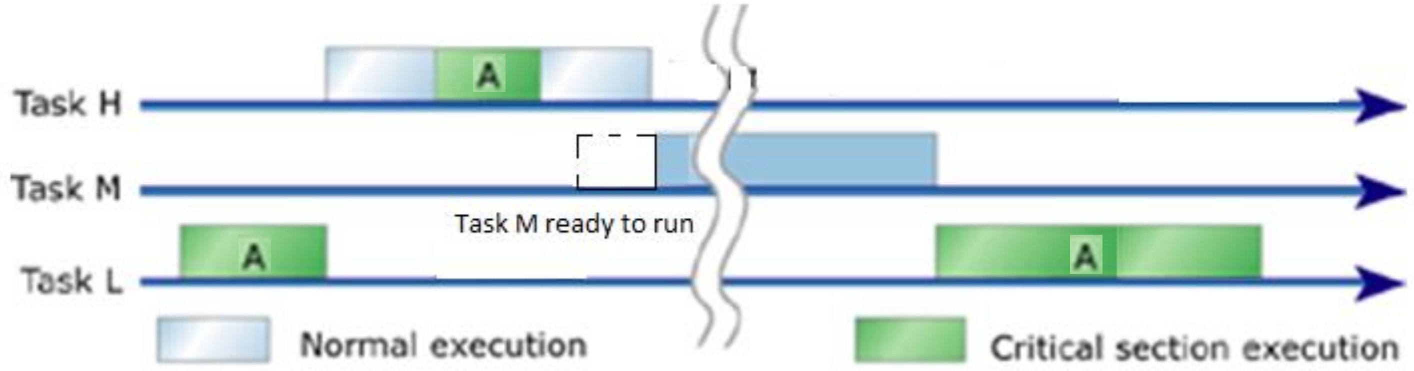


Ideally

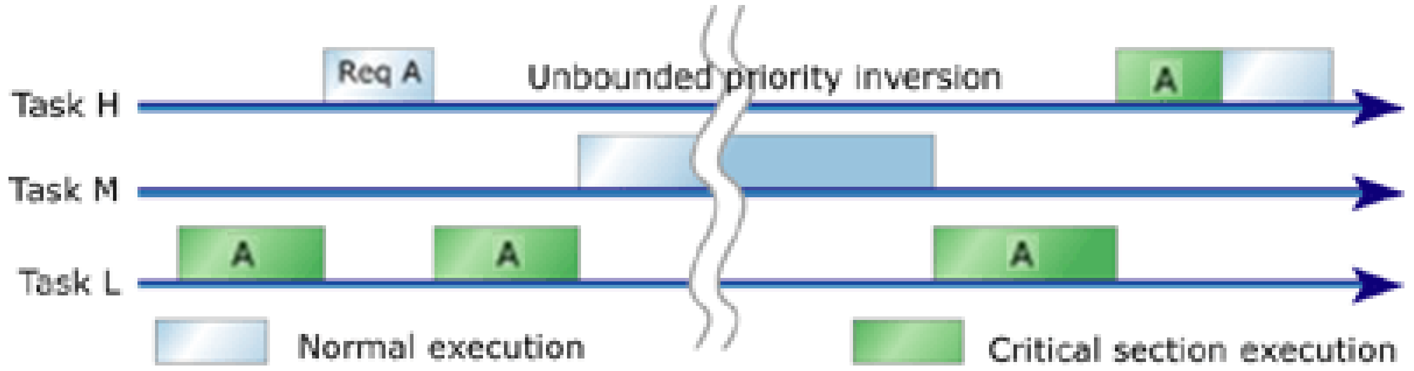


In Reality

Unbounded Priority Inversion



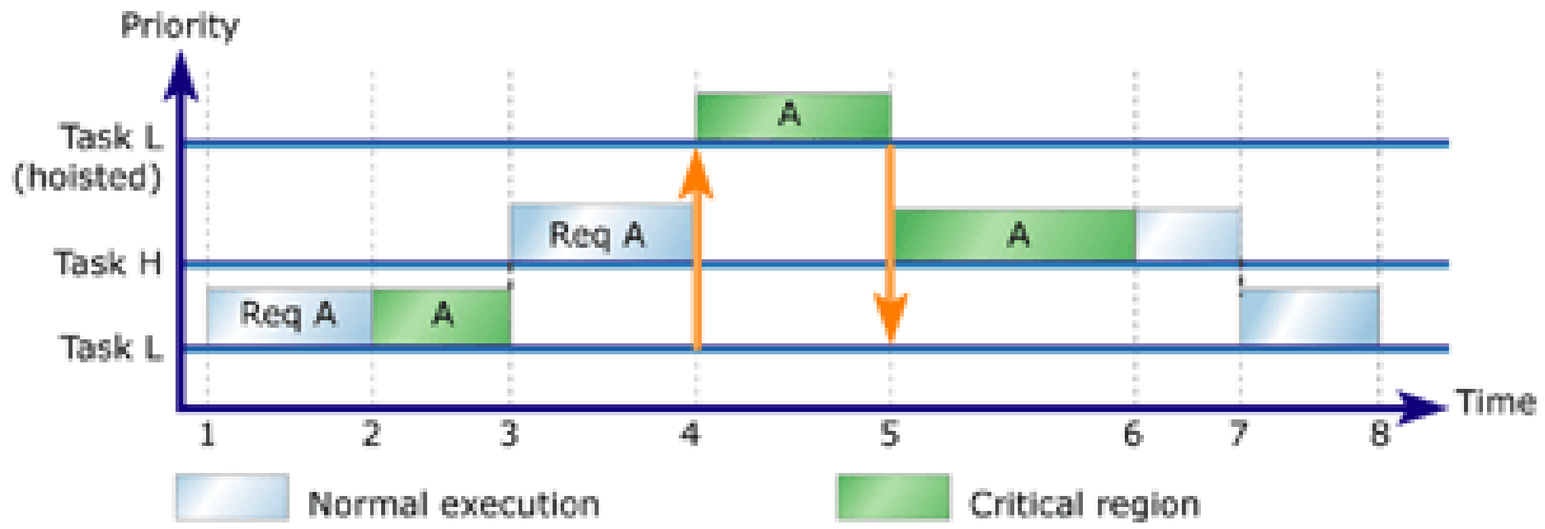
Ideally



In Reality

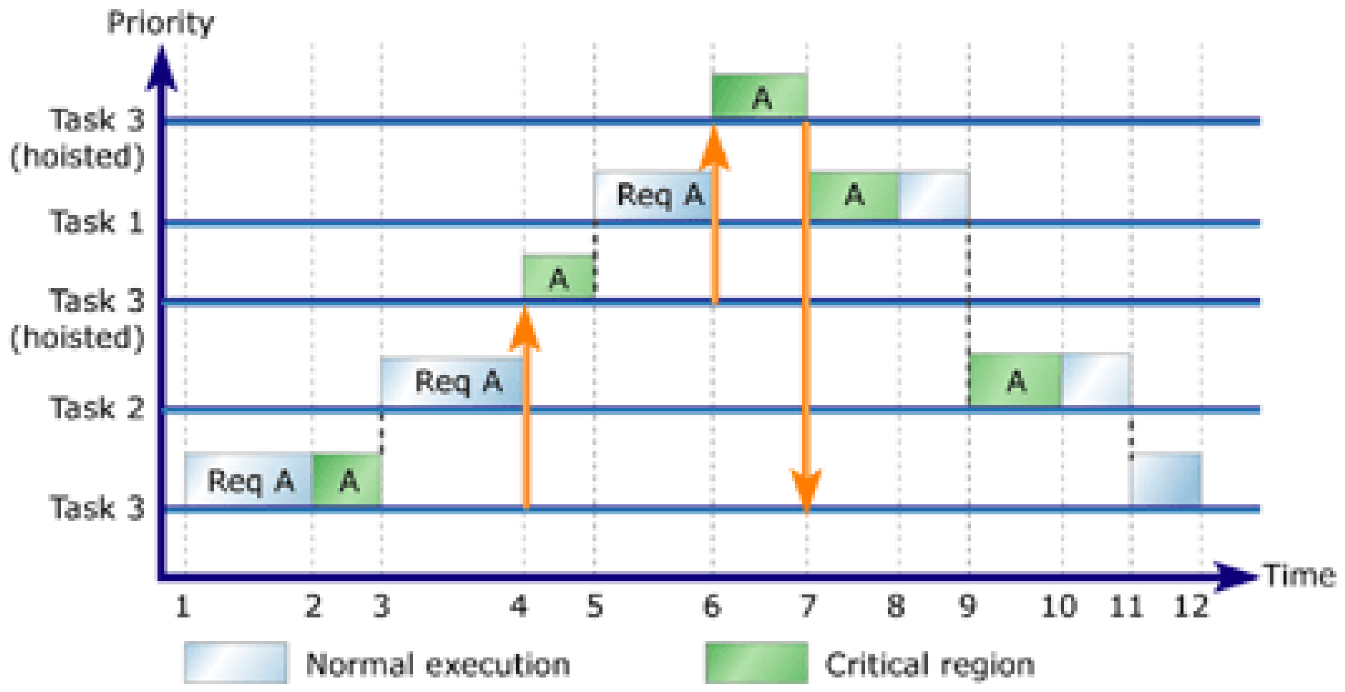
Priority Inversion

Less significant when there are only two tasks fighting for a resource



Simple priority inheritance

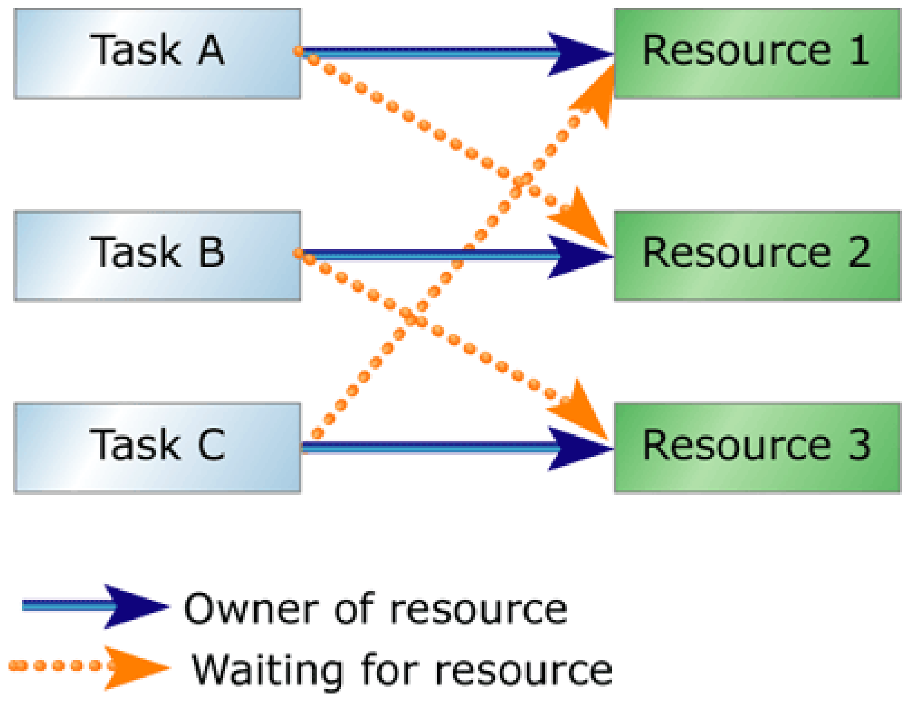
Priority Inversion



Three-task, one-resource priority inheritance

Advantage is more visible when there are three or more tasks are fighting to get the same resource

Dead lock


Task A
Task B
Task C
Resource 1
Resource 2
Resource 3
Owner of resource
Waiting for resource

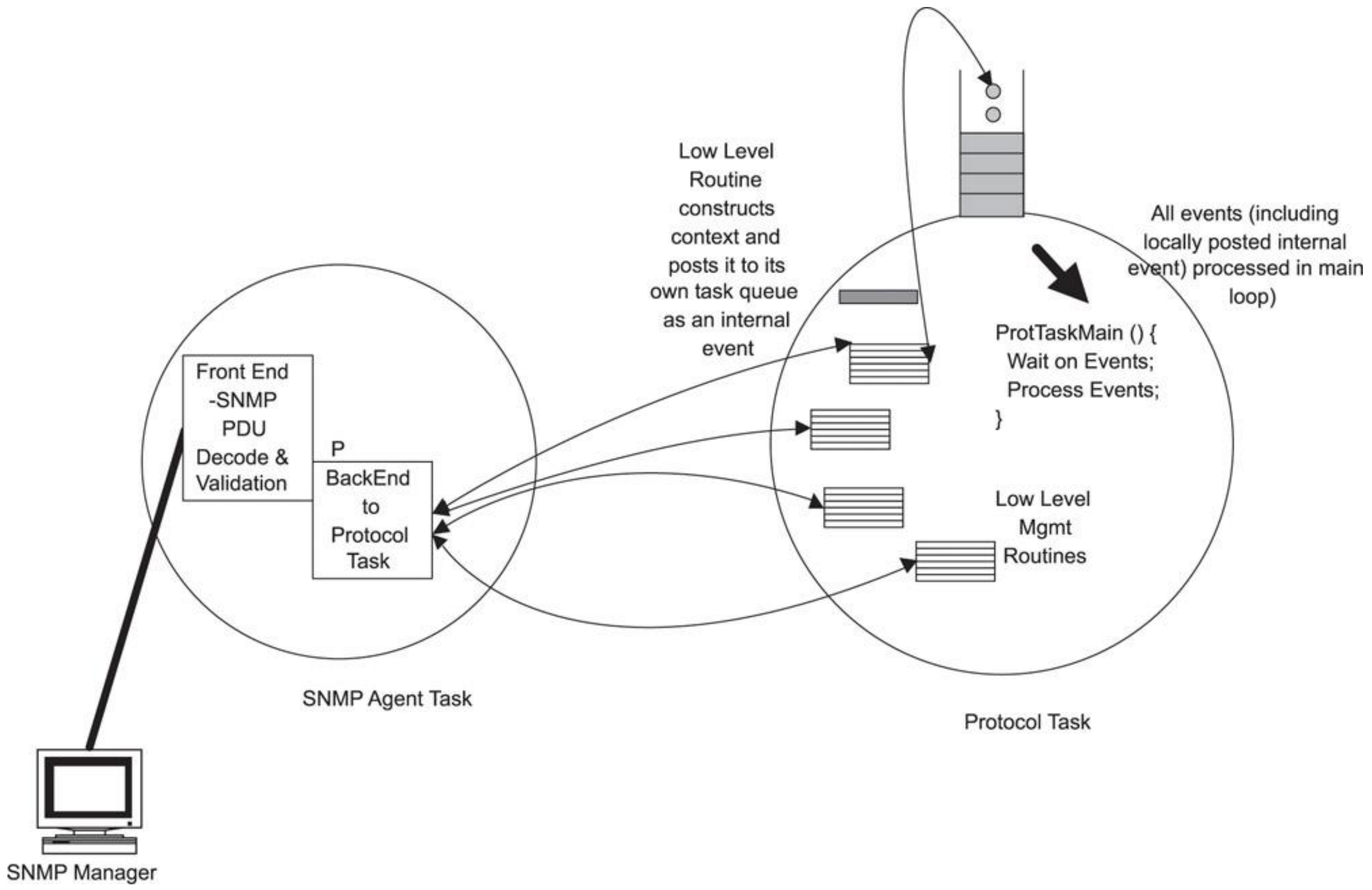
Agent-Protocol Interface contd…

- Ideal Solution: Provide for agent task operation within the main processing loop of the protocol task
  - Prevents lateral access to data structures
  - Only one thread of execution eliminates need for mutual exclusion
- Example of implementation
  - Invoke low level routine within protocol task from the Agent task
  - Low level routine identifies type of operation and sends an EVENT to the protocol task
  - This event is an internal event specific to a task queue
  - Since message is queued, protocol task can process it within the main loop without worrying about access conflicts to the data structure

Agent-Protocol Interface contd…

- Memory separation between Agent and Protocol
  - If Agent task and Protocol task live in separate memory spaces, we HAVE TO USE a message-based interface
    - WHY? Because the agent task will NOT HAVE ACCESS to the low level routines provided by the protocol task
  - If Agent and Protocol tasks live in the same memory space, we could use a message-based or procedure-based interface

SNMP


Low Level Routine constructs context and posts it to its own task queue as an internal event
All events (including locally posted internal event) processed in main loop)
Front End -SNMP PDU Decode & Validation
P
BackEnd to Protocol Task
ProtTaskMain () {
Wait on Events;
Process Events;
}
Low Level Mgmt Routines
SNMP Agent Task
Protocol Task
SNMP Manager

Trap: Device->Manager Communication

- This communication channel is used to capture information that affect the operation of the embedded device
  - Example: Traffic on a port exceeds the threshold, a port is abruptly shut down. Manager must be Alerted.
- This is an important part of management model and requires different mechanisms for capture
  - SNMP uses a mechanism called “trap”
  - In CLI, the alerts are displayed on the user console

Trap:
Device->Manager Communication

- Issues to watch for:
  - Protocol task uses a shared queue for both alerts and normal response to GET requests to the agent which in turn determines when to forward them
  - Too many traps or alerts can become unmanageable
  - Categorizing traps/alerts into critical and non-critical at the protocol level and sending critical alerts before it’s too late will help the manager shut down the system gracefully

System Setup and Configuration

- Boot parameter configuration
  - Typical boot parameters include: Ethernet IP address, boot file name, user name and password, flags, etc.
  - Support for additional parameters will require reprogramming the boot ROM
  - Actual configuration is performed using a command line interface
  - SNMP or HTTP protocols CANNOT be used for the configuration because device will not have an IP address until it is boot configured
- Post boot configuration
  - After boot ROM image has been downloaded, the boot parameters can be changed using SNMP or other protocols
  - After a change, the changed parameters can be saved in EEPROM or flash and the device on re-boot picks up the new parameter values

Saving & Restoring Configuration

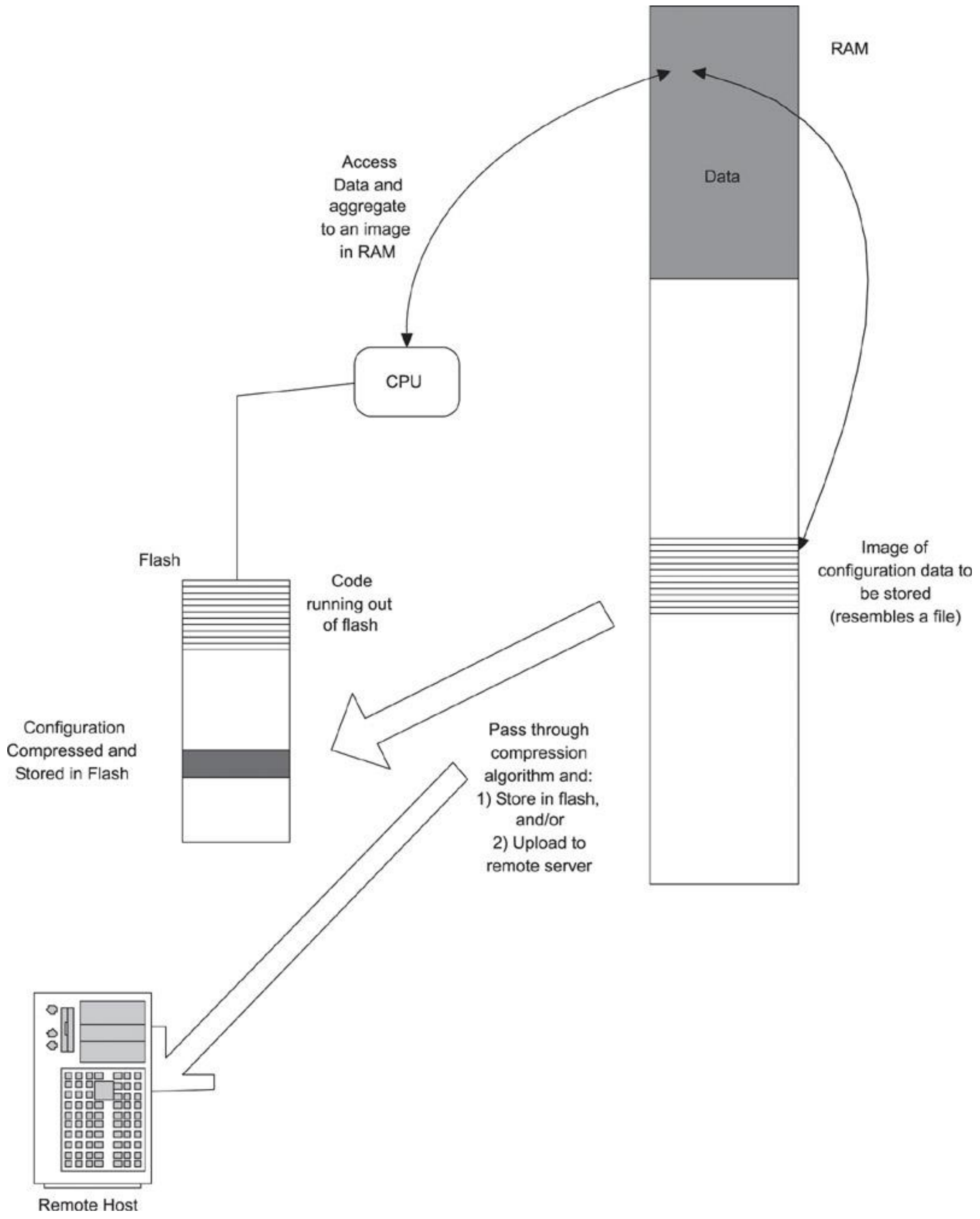
- Manager based approach
  - SNMP (or any protocol) manager uses GET operation to obtain a record of device configuration variables and stores it
  - On reboot, a configuration restoration request is made to manager, which then uses SET operations to restore configuration
  - Provides a centralized approach where no work is required of the device
  - No concern about running out of storage space
- Device based approach
  - Manager instructs device to save current configuration
  - The device determines how to categorize and store basic and non-basic configuration parameters efficiently
  - Relies on the master source of data kept locally on the target in non-volatile memory

Operation & Saving process

- Restoring the configuration
  - Restoration happens at boot time
  - Configuration file is downloaded from a local or remote location and decompressed
  - Simple changing the value of a variable is not sufficient. it is essential that, when restoring configuration, we mimic the effects of the steps to initialize the modules.
  - System is not fully initialized until the restore process has completed

Operation & Saving process

- Saving the Configuration
  - Basic and non-basic parameters are stored at a temp location in the RAM
    - TLV encoding is commonly used-> contains type, length and actual value of parameter
  - The RAM entries are then compressed and a checksum added at end for error detection
  - This compressed entry which resembles a binary file can then be stored in the local flash or a remote location
    - Good design principle is to provide for both local and remote storing since the size may exceed the local flash capacity


RAM
Data
Access Data and aggregate to an image in RAM
CPU
Flash
Code running out of flash
Image of configuration data to be stored (resembles a file)
Configuration Compressed and Stored in Flash
Pass through compression algorithm and:
1) Store in flash, and/or
2) Upload to remote server
Remote Host