CPW – Commercial Processing Workload

- CPW: Based on TPC-C
  - iSeries performance, because it is not a uni-processor, is not in direct relation to MHz. Rather, it is measured with a relative, commercial benchmark
- Users use four distinct interactive applications
- Complex Transactions
- Reads, updates, inserts, deletes, block inserts, index changes
- Journaling and commitment control
- Includes daytime batch

Filesystem, Objectstore, ASP, Singellevel Storage, Userprofiles

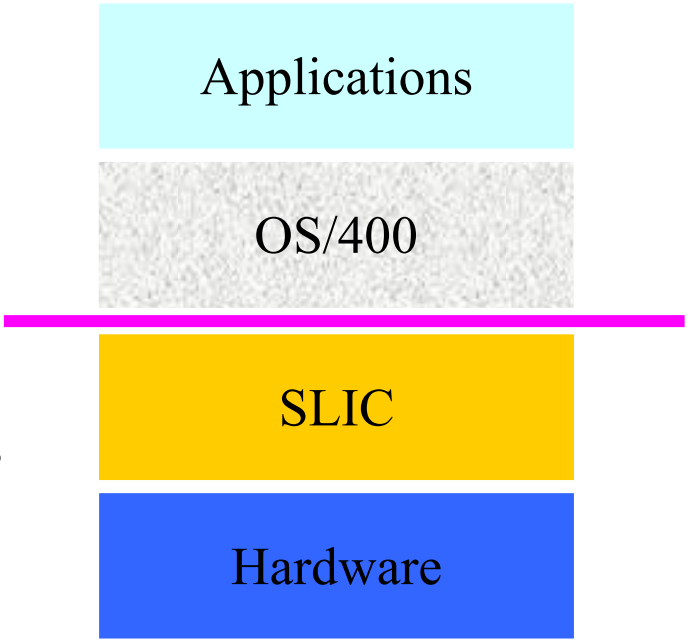
iSeries Architecture – Part 2

Use of Objects

OS/400 Objects

based on

MI System Objects



- Objects are used for complex data structures

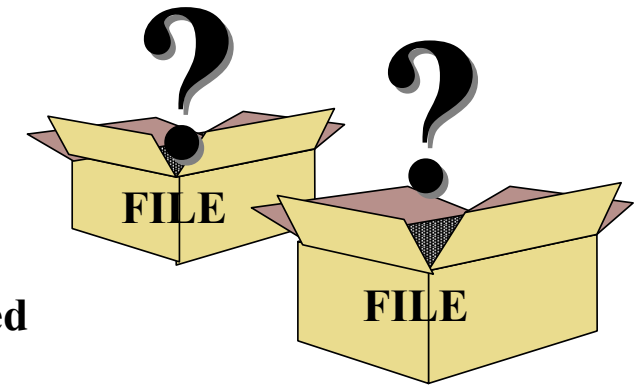
- Examples: Database file, User profile, Program, …

3.1 Object-based Design

Storing Informations

Conventional Systems

- A string of bytes can be allmost anything
- Anything in permanent store is a file

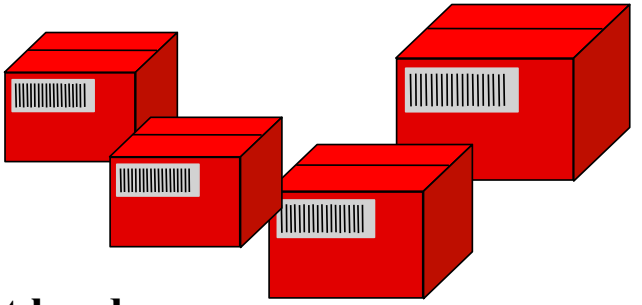


Unlabeled

Program?
Data File?
Control File
Batch File?

iSeries

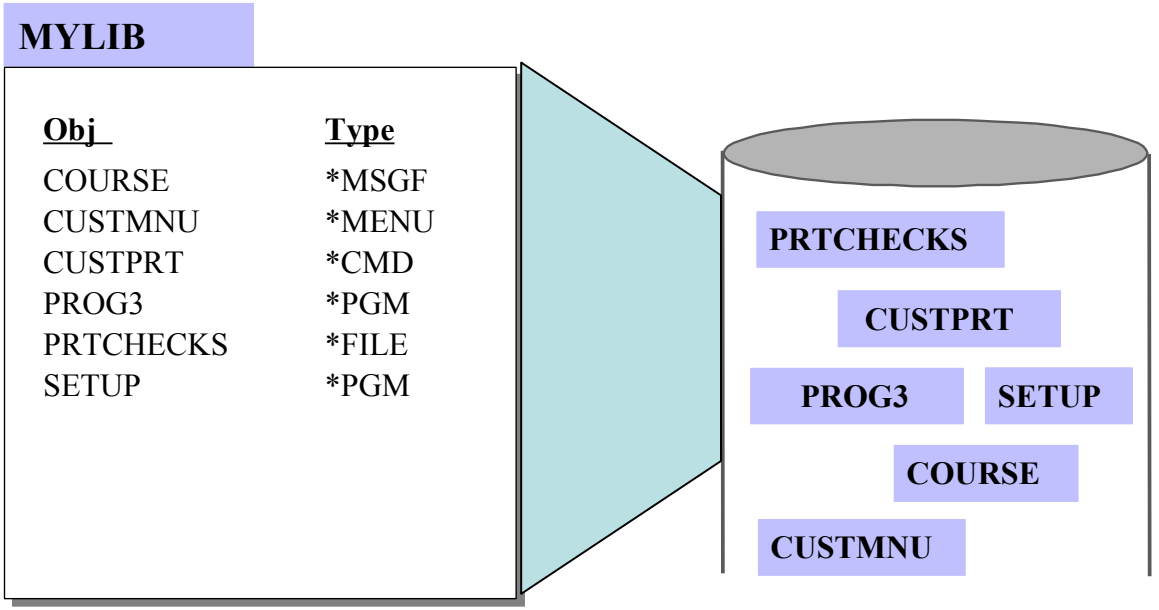
-Object Based Architecture
-Informations are encapsulated



Object header

Type = File
Type = Program
Type = System Object
Type = User Profile
Type = Command

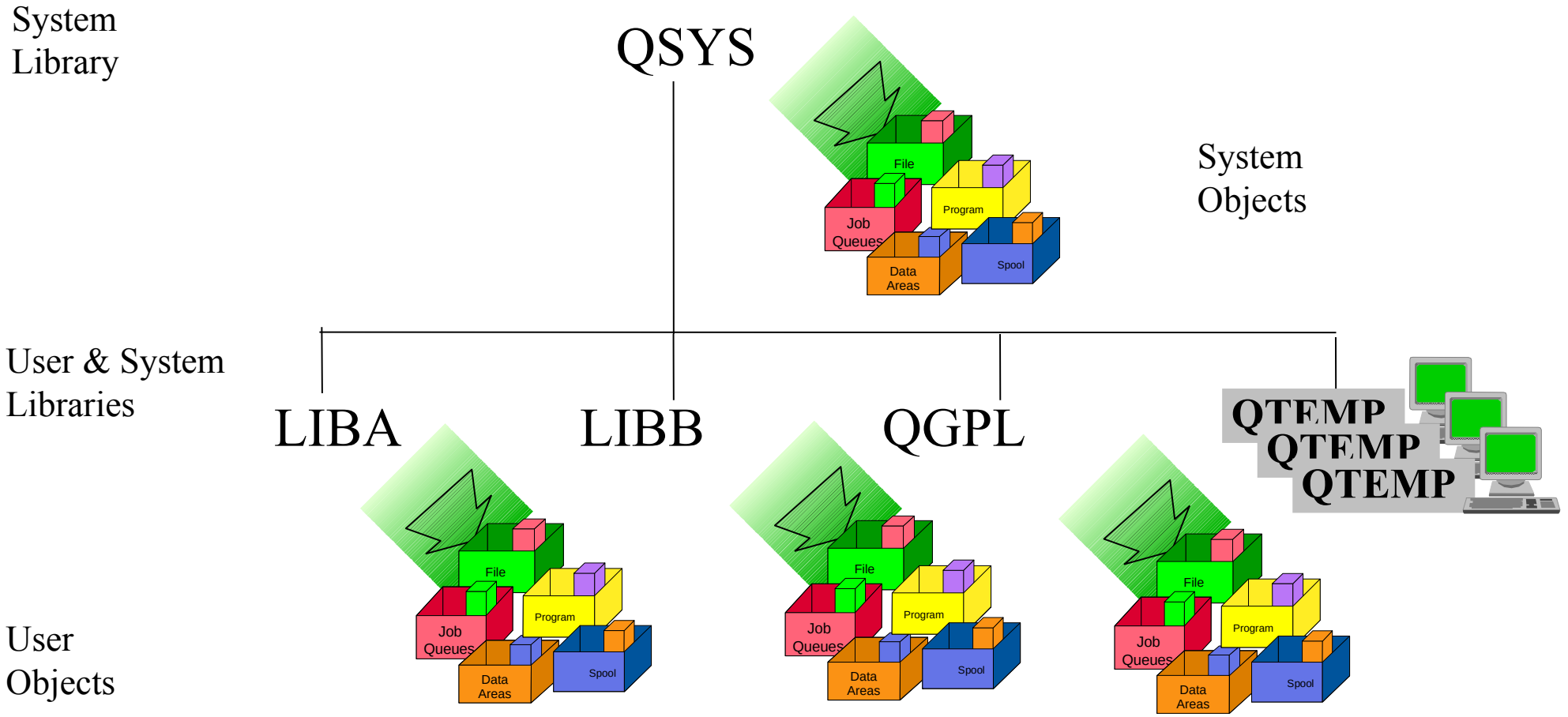
What Is a Library?



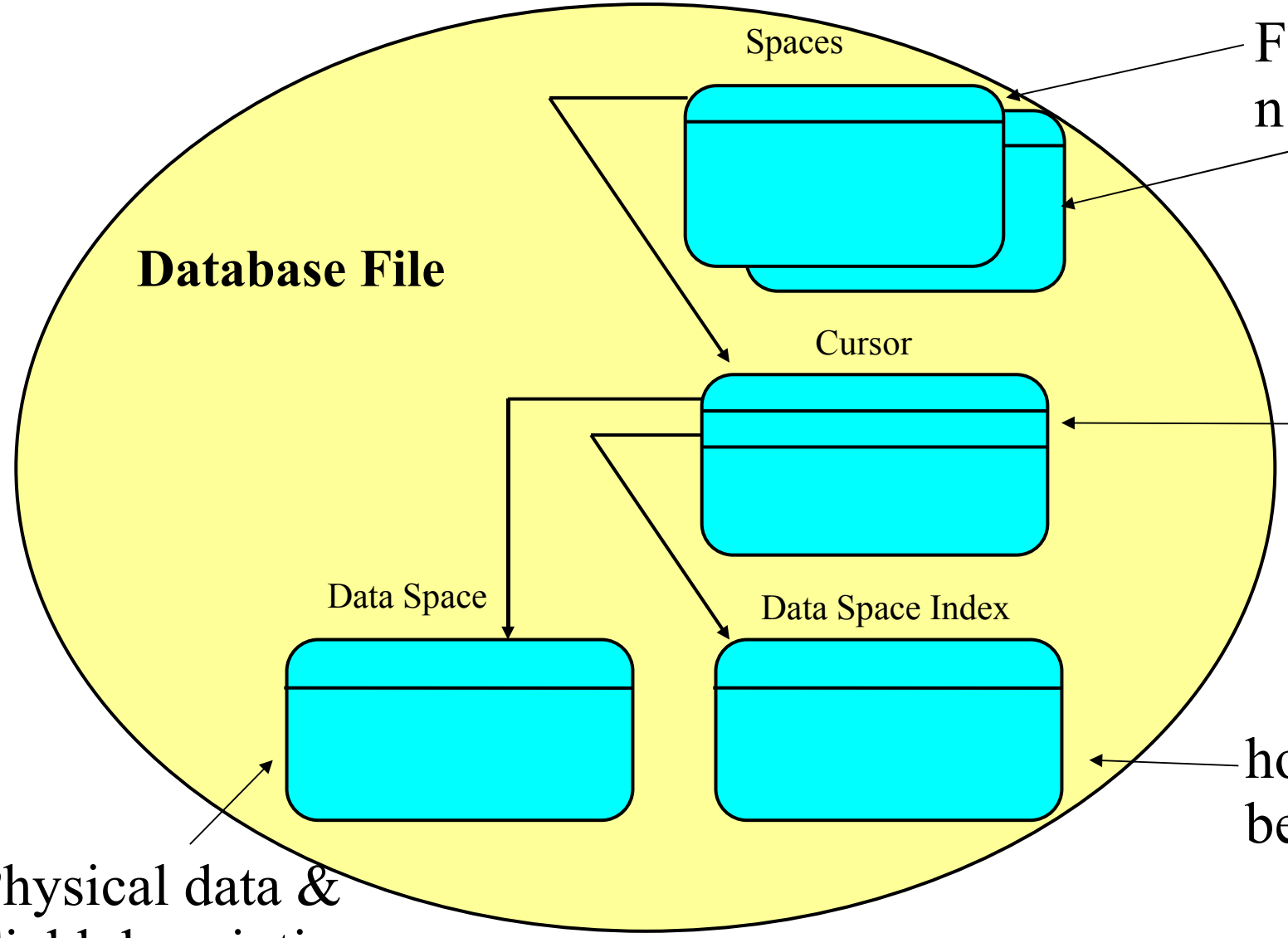
Libraries are used to organize objects

- For security reasons
- For backup reasons
- By application
- By owner
- By object type: program versus files
- By use: production versus test

OS/400 Object File System



OS/400 Object Composition Example



Filedescription

Results of database operations

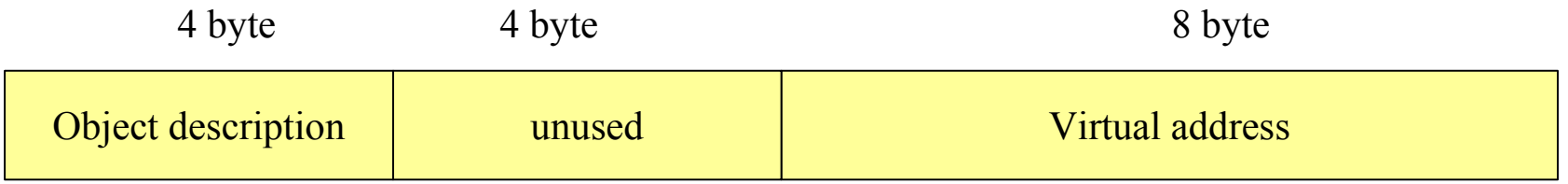
ODP (open datapath)

how data should be viewed

Physical data & Field description

MI Pointer

- 16 Byte long

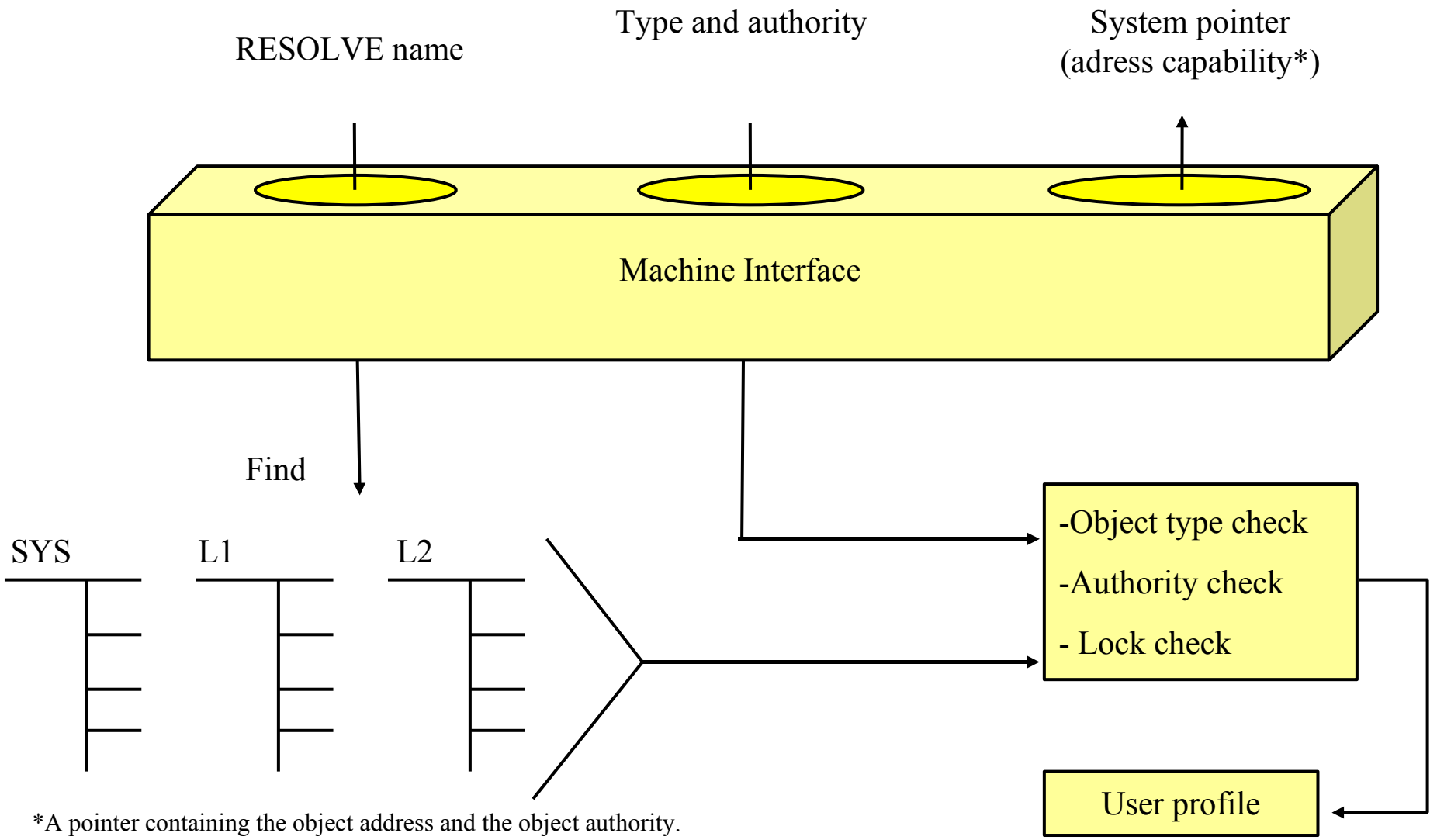


Status bits

- type of pointer
  - System pointer
  - Space pointer
  - Data pointer
  - …
- Informations about the object
  - System pointer -> object type
  - Data pointer -> type of date
  - …
- Authorities
  - Could only be changed by OS in system state

Remarks:
- unused bits could be used to expand to 96-bit addresses without effecting any program above the MI
- type and object informations could also moved out of the pointer to go beyond 96 bits

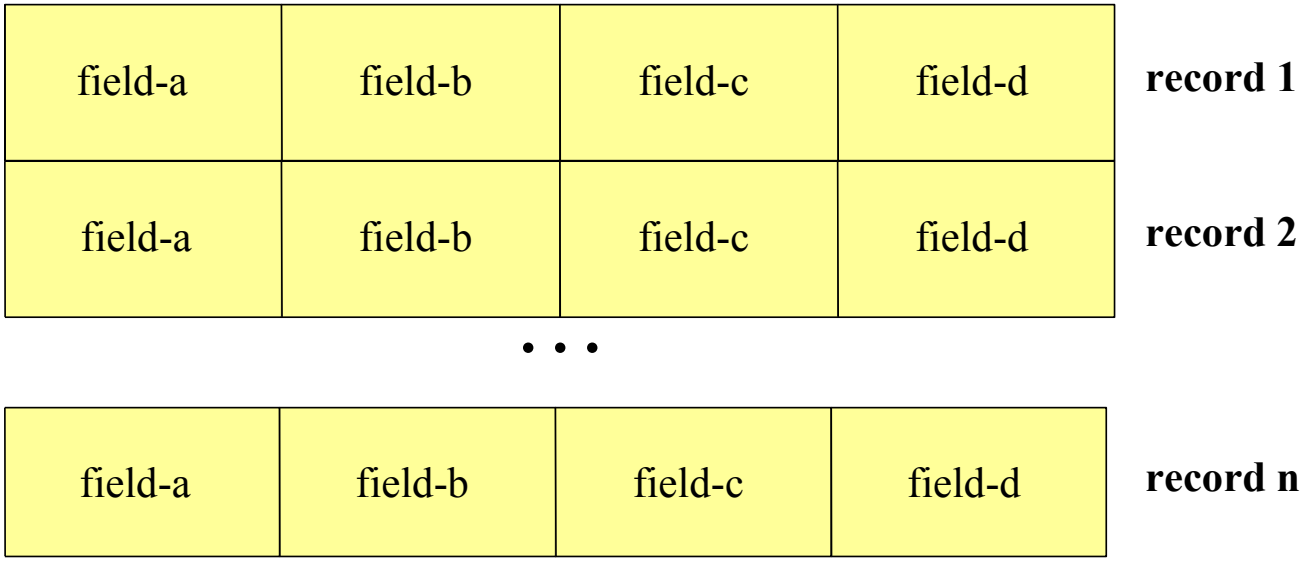
Object Identification: Library, Name, Type



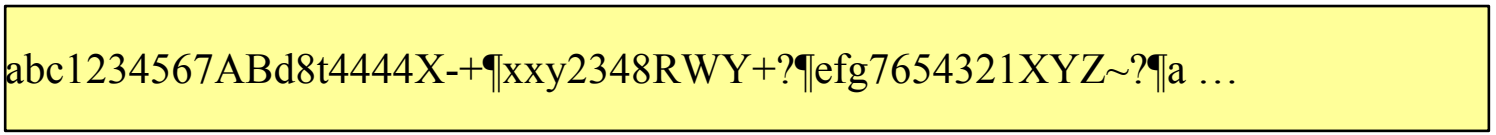
*A pointer containing the object address and the object authority.

Stream Files vs. Database Files

Database files



Stream files



Finding an Object

Simple name: CALL PAY02

Qualified name: CALL PAYTSTLIB/PAY02

QSYS

QCWW	QCXXX
QCZZZ	QCYYY

INQLIB

PAY77	AP60
PAY99	AP55

PAYTSTLIB

AP55	PAY02
PAY01	AP05

PAYLIB

PAY01	PAY04
PAY02	PAY05

Job's library list

- . . .
- QSYS
- QSYS2
- QHLPSYS
- QUSRSYS

System libraries

- QRPG
- QCBL

Product libraries

- PAYLIB

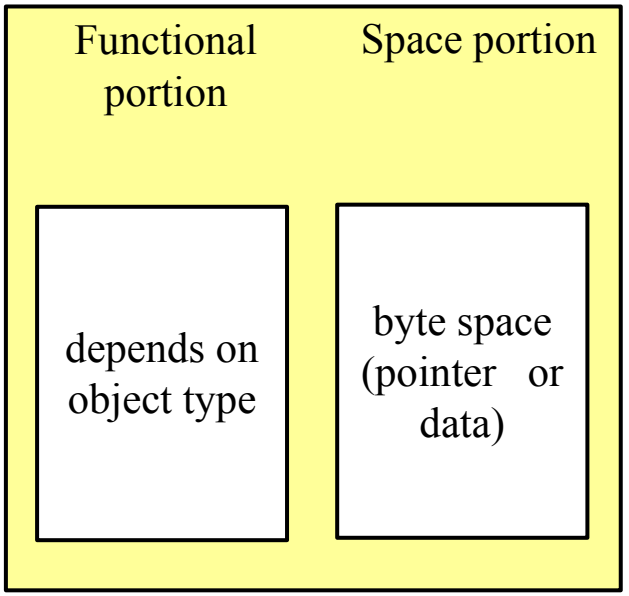
Current library

- . . .
- QGPL
- QTEMP
- PAYTSTLIB

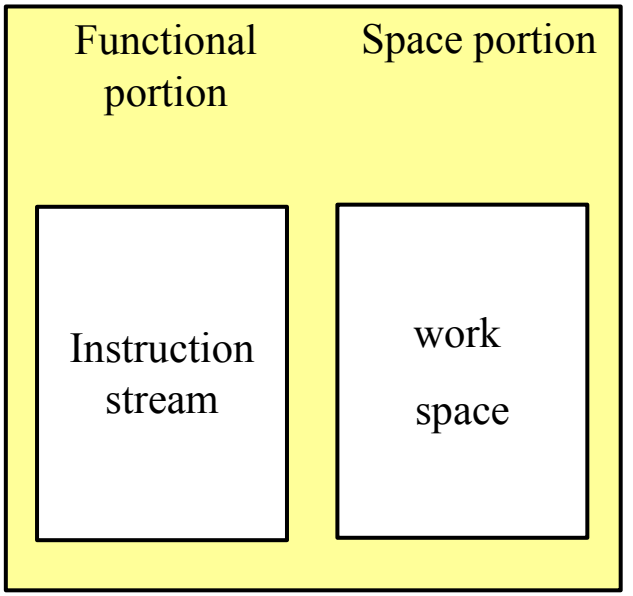
User libraries

- INQLIB

Internal Structure of System Objects



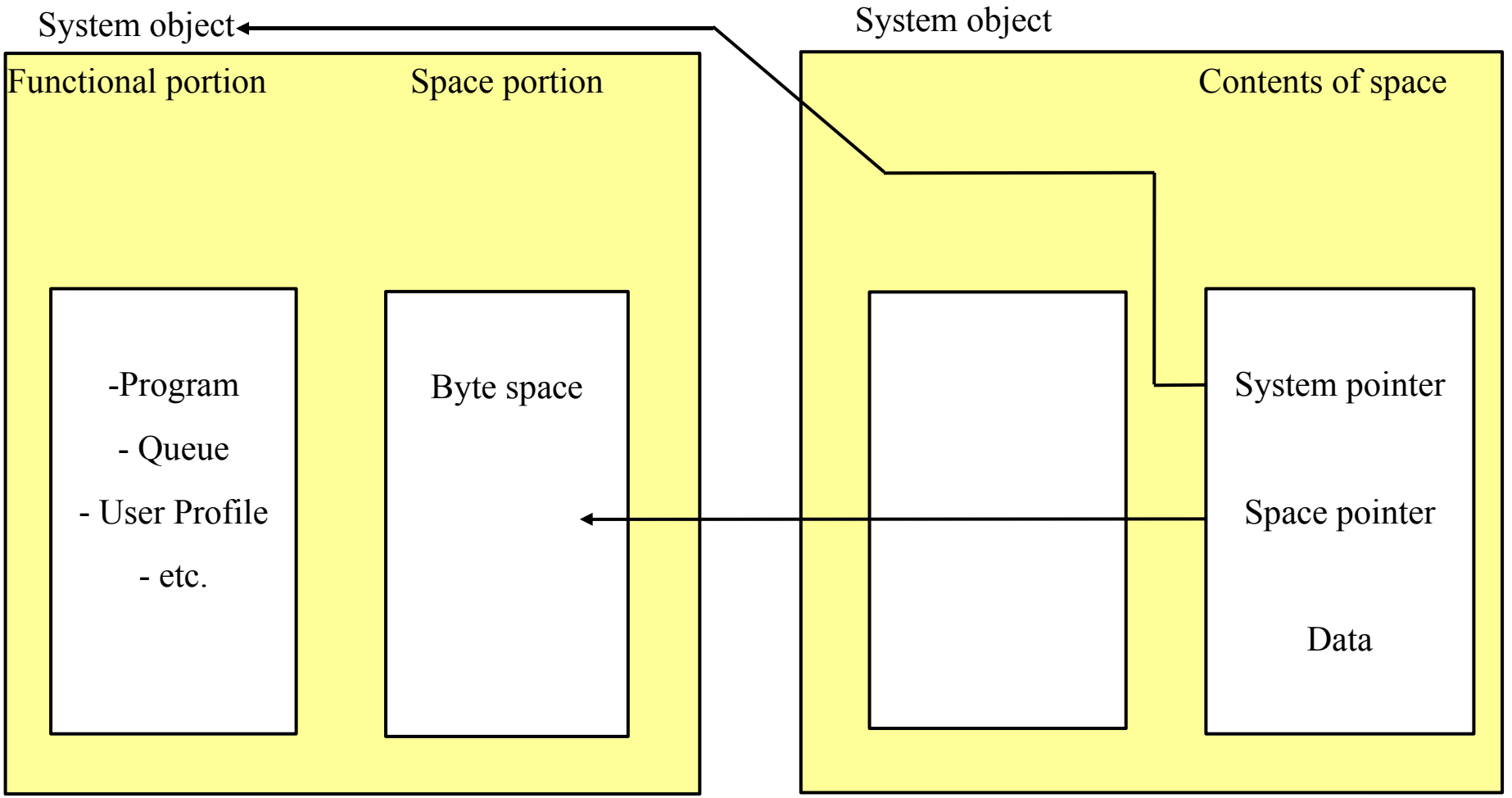
All system objects except spaces



Example: Program

Spaces have no functional portion!

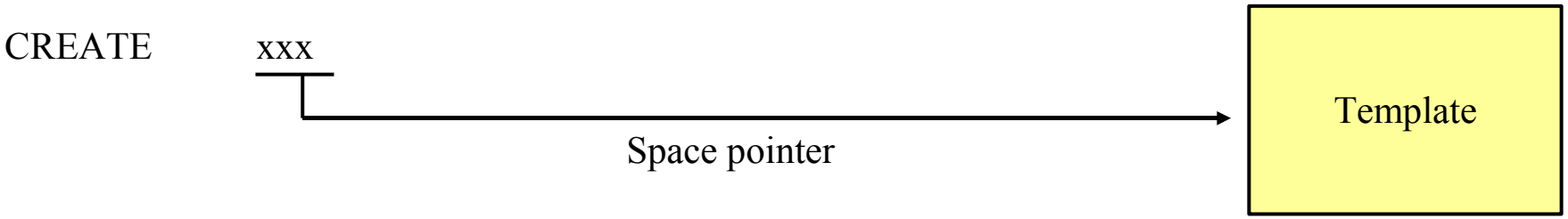
Manipulating Data Inside an Object



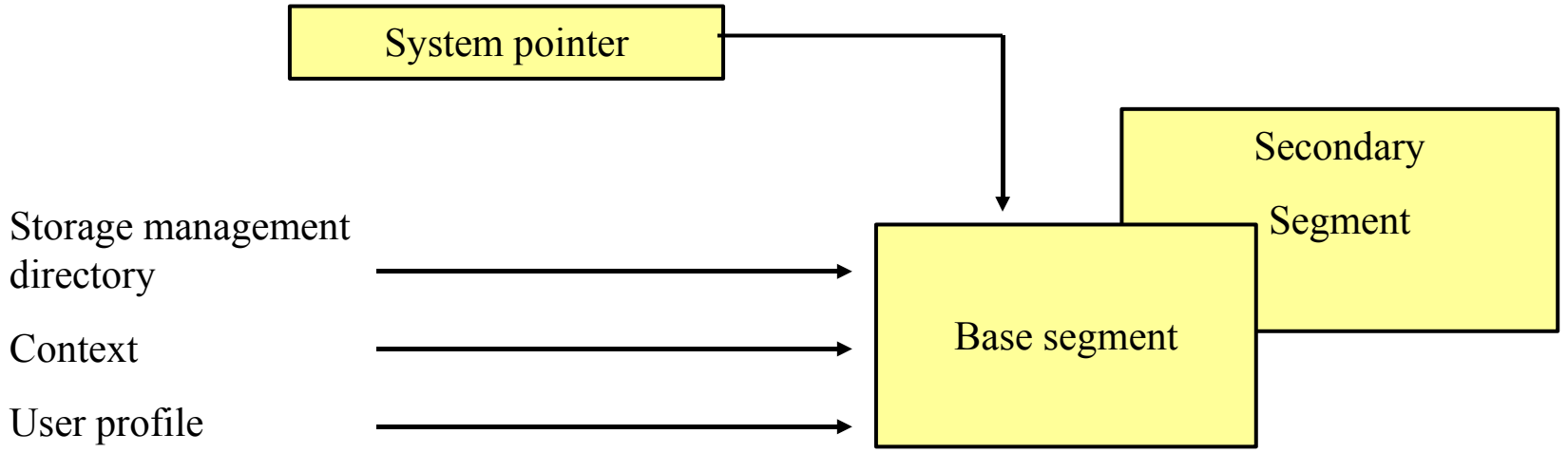
- System pointer can point only to the beginning of an object
- Space pointer points to a byte in the space portion
  - use space pointer to access and manipulate bytes in a space
- Space pointer can be modified by an MI program, system pointer cannot

Object Creation

INPUT



OUTPUT



- System objects must be explicitly created with an CREATE instruction at MI
- A CREATE intructions references to an user-supplied template contined in a space object
- System pointers provides addressability of system objects

Object Persistence

Objects continues to exist in system memory
forever,
unless it‘s explicitly destroyed!

Sharing data between user means in conventional systems requires to store informations in a filesystem.

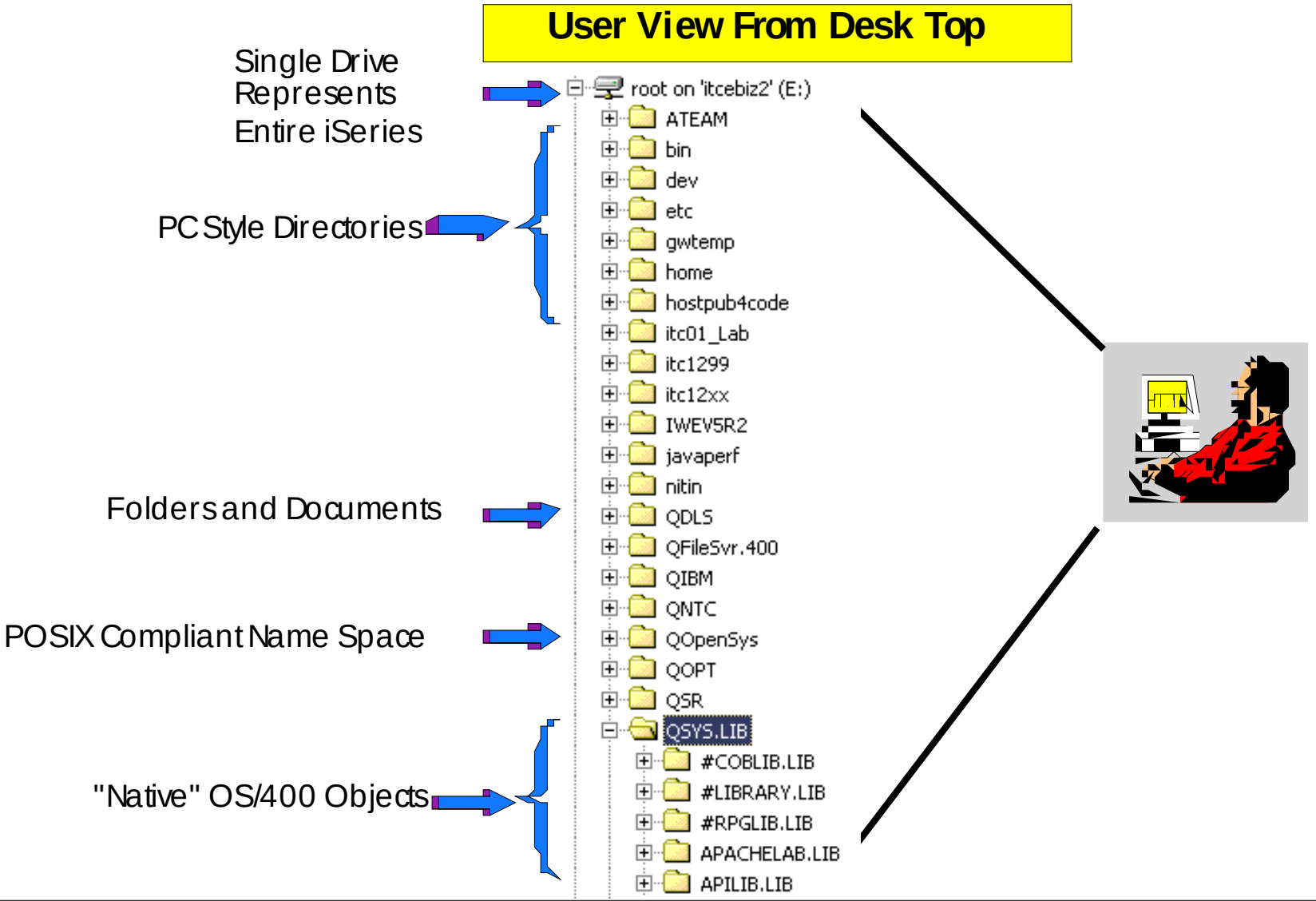
iSeries have a single-level store!

Security?

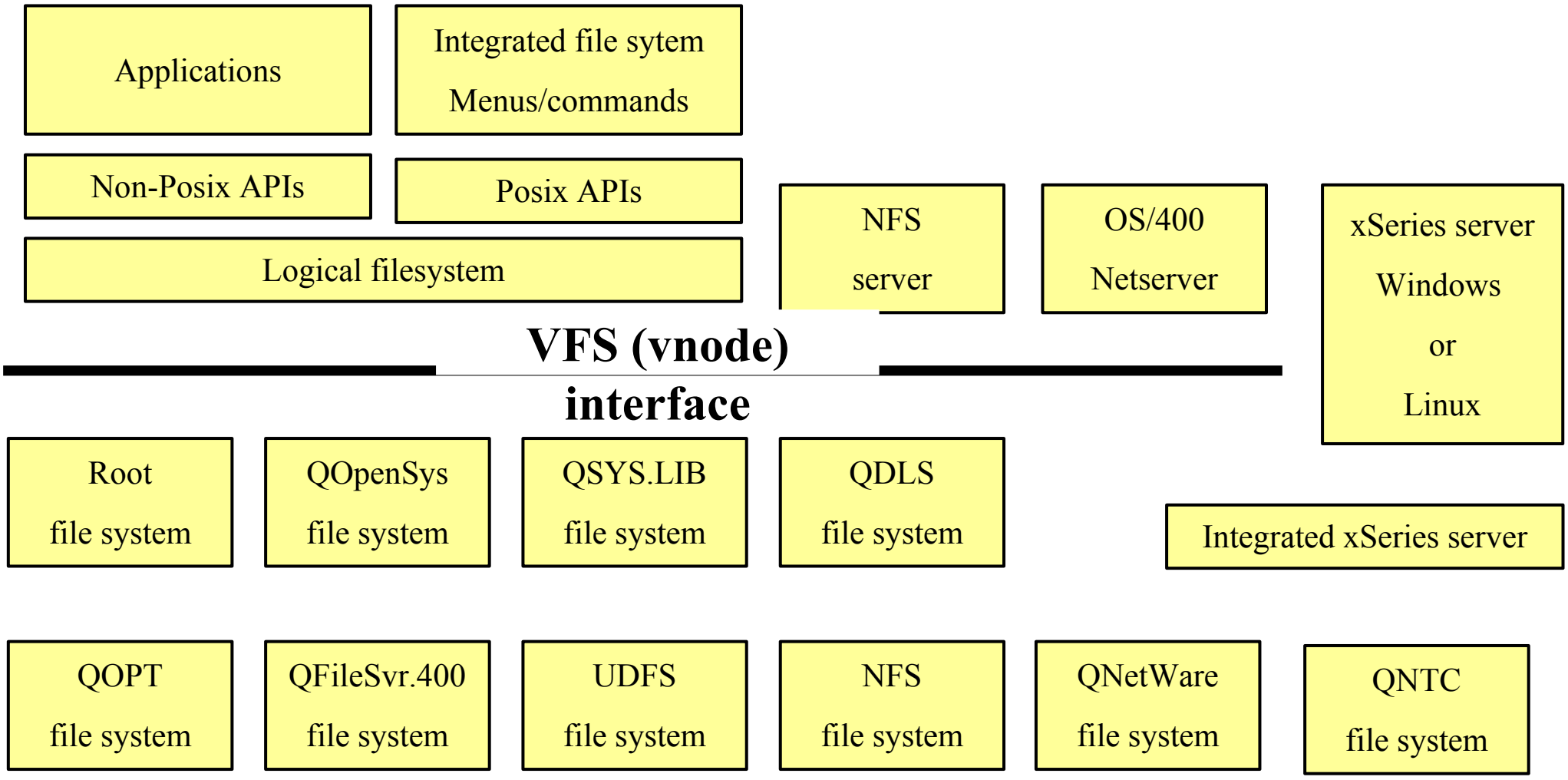
No filesystem?

3.2 File Systems

Integrated File System



Integrated File System Structure

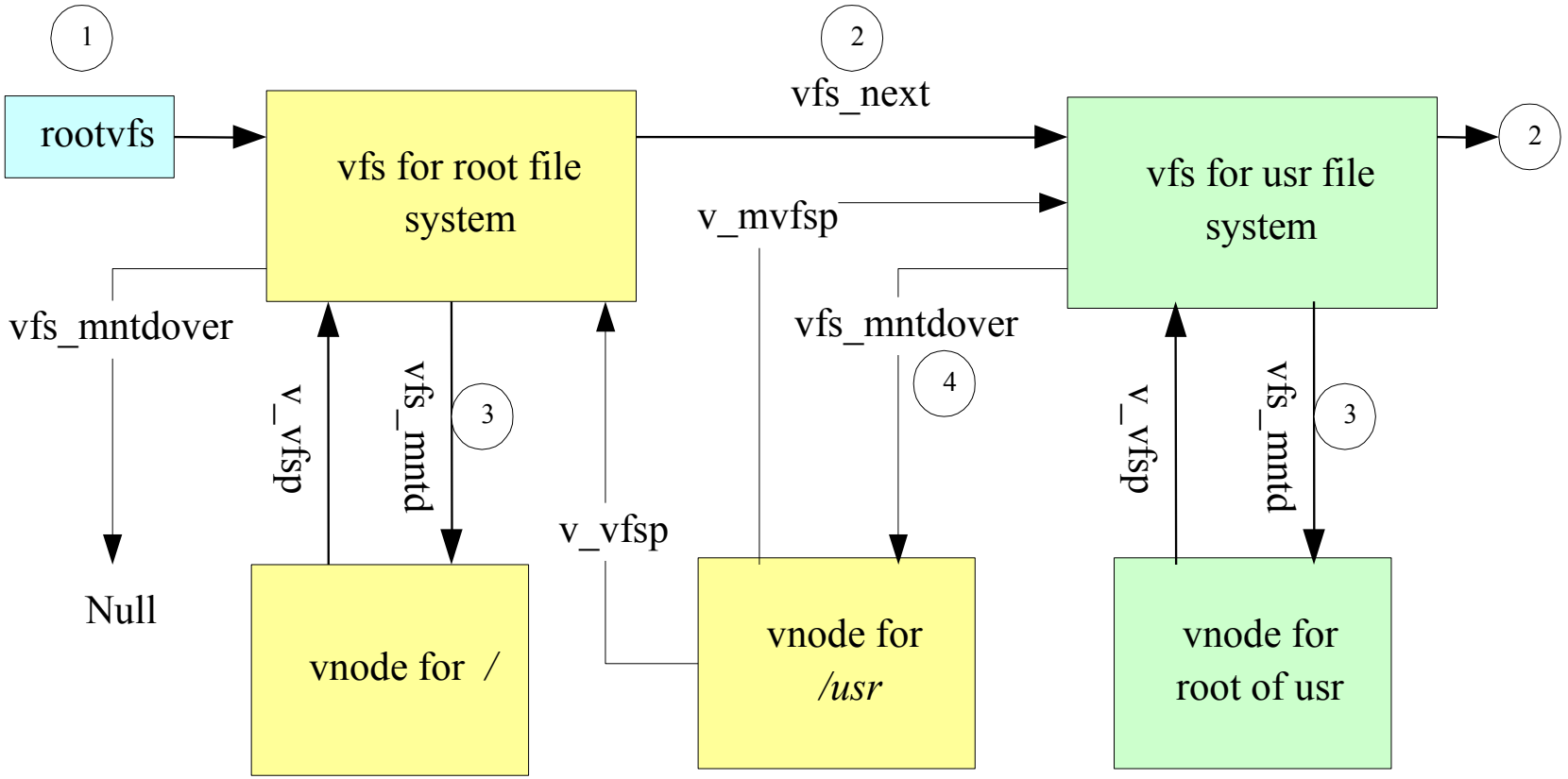


Virtual File System (VFS)

- **Virtual file system** (VFS) is a standard and abstract software layer that allows the operating system kernel (e.g. the Linux kernel) to call file system functions without having to know the type of file system being used.
- A **vnode** is an object in kernel memory that speaks the UNIX file interface (open, read, write, close, readdir, etc.). Vnodes can represent files, directories, FIFOs, domain sockets, block devices, character devices.

- Objectives:
  1. **Both different types of Unix file systems and non-Unix file systems should be supported at the same time**. Different disk partitions may contain different types of file systems, but they should be mountable on each other to form a single directory tree.
  2. Files belonging to different file systems should be **easily sharable** over a network.

Relationship between the *vfs* and *vnode* structures



1. The rootvfs points to the vfs root file system
2. The vfs_next pointers create a linked list of mounted filesystems
3. The vfs_mntd points to the vnode representing the root in the filesystem
4. The vfs_mntdover points to the vnode of the directory the file system is mount over.

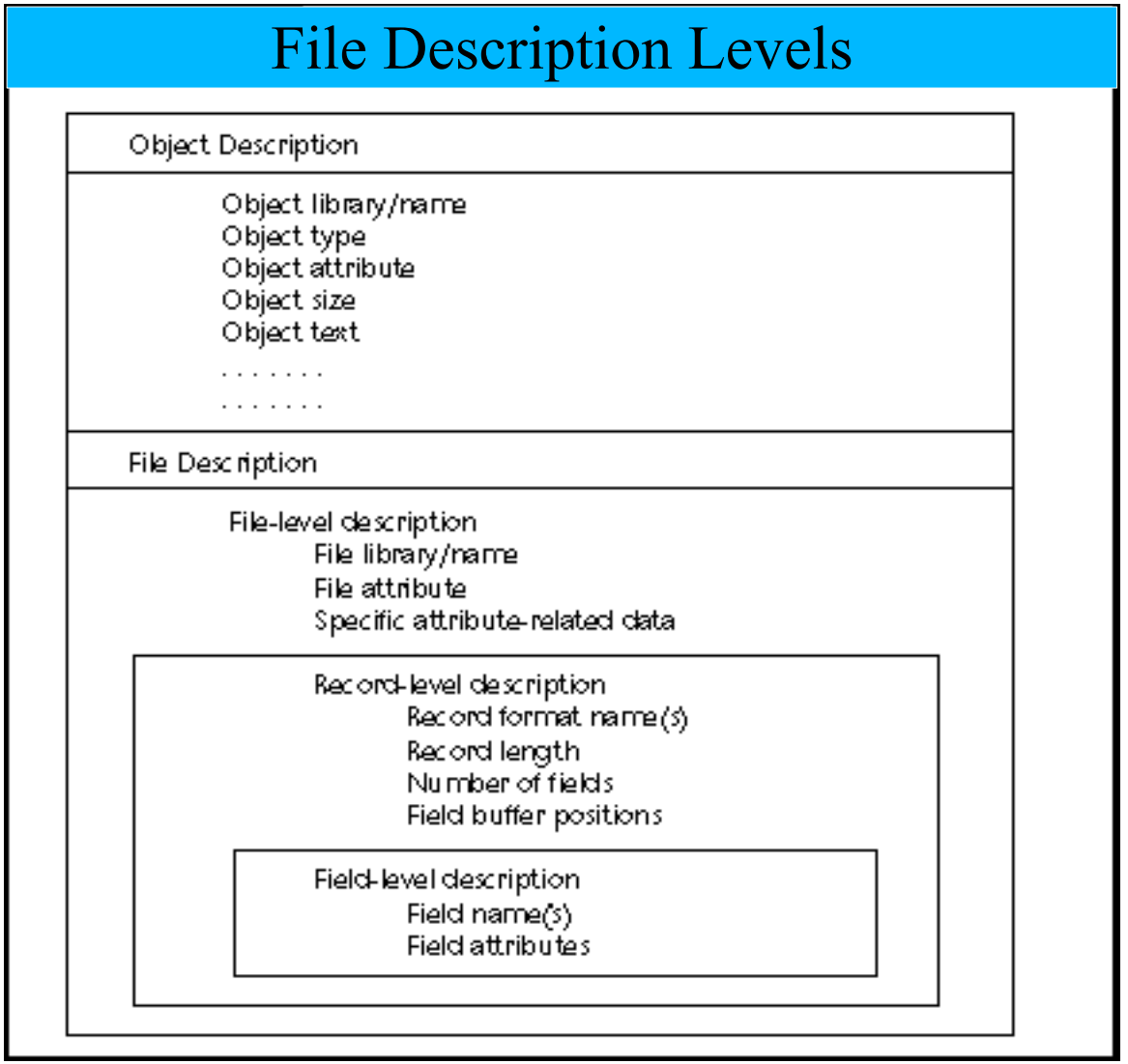
Files? Directories?

OS/400 Files

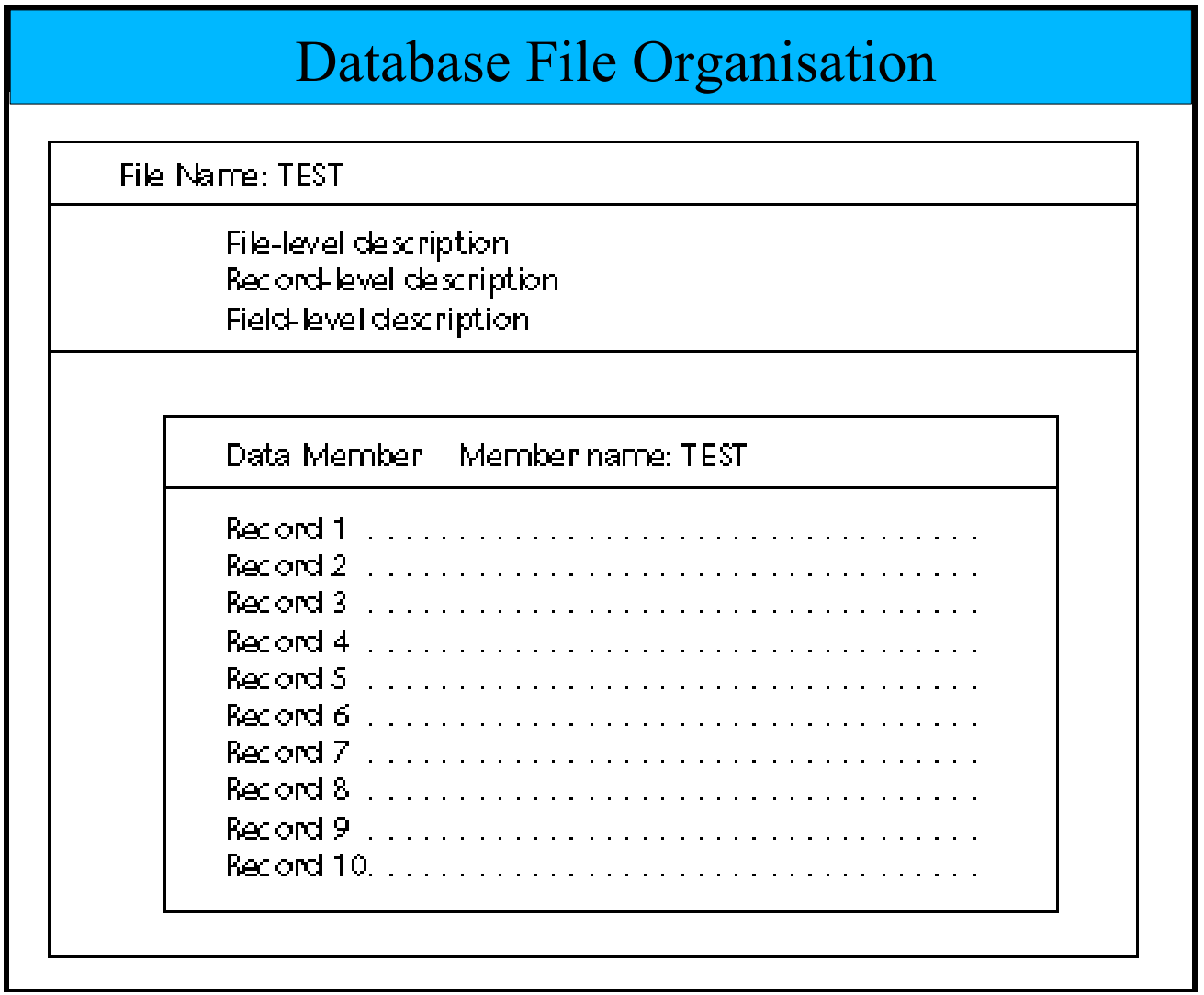
Filetypes, Subtypes, Createcommand

File Type	Subtype	File Description	Create Command
Database File	PF	Physical File	CRTPF
	LF	Logical File	CRTLF
Source File	PF	Physical Source File	CRTSRCPF
Device File	DSPF	Workstation Display File	CRTDSPF
	PRTF	Printer File	CRTPRTF
	TAPF	Tape File	CRTTAPF
	DKTF	Diskette File	CRTDKTF
	ICFF	Intersystem Communications Function File	CRTICFF
DDM File	DDMF	Distributed Data Management File	CRTDDMF
Save File	SAVF	Save File	CRTSAVF

OS/400 Files

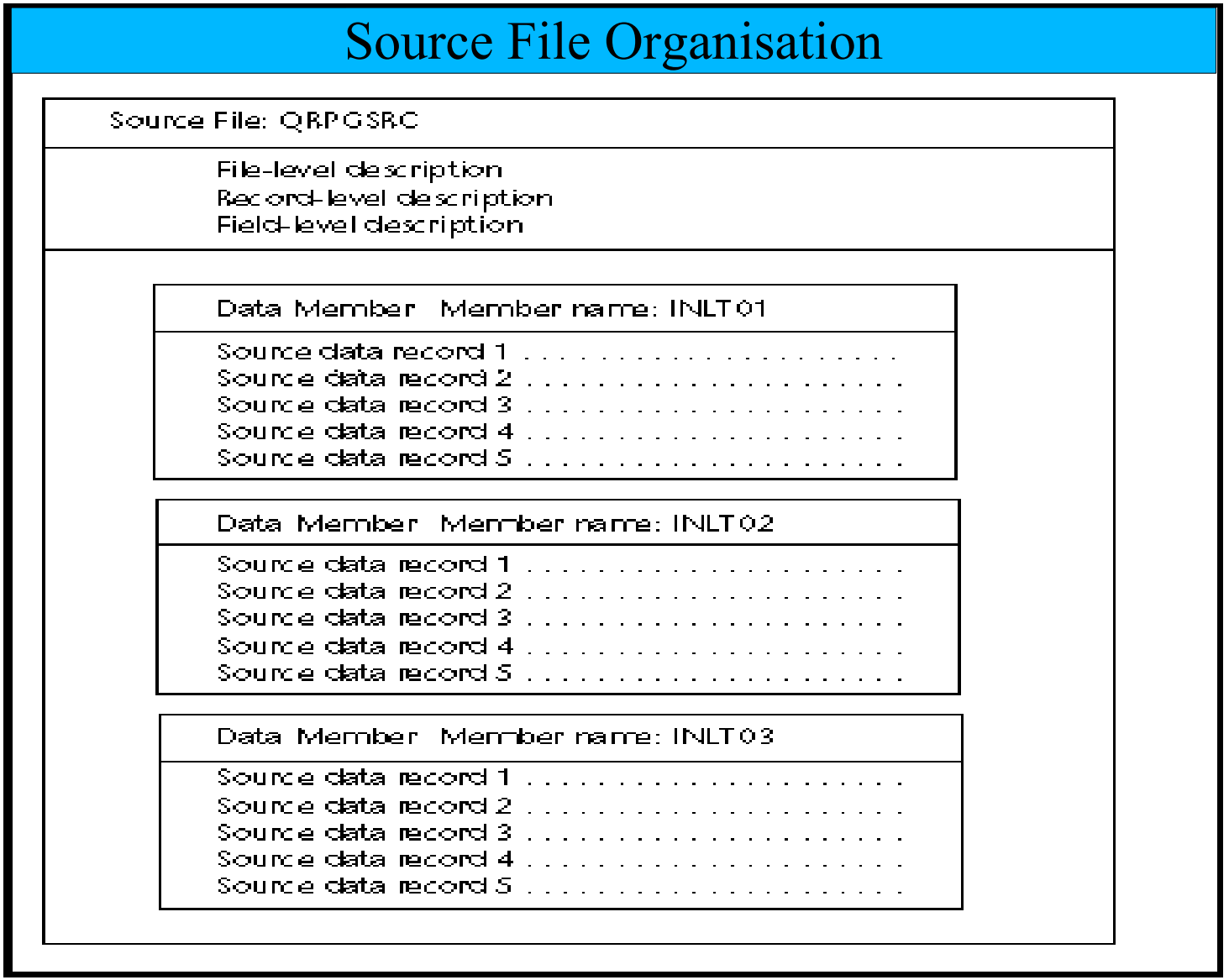


OS/400 Files

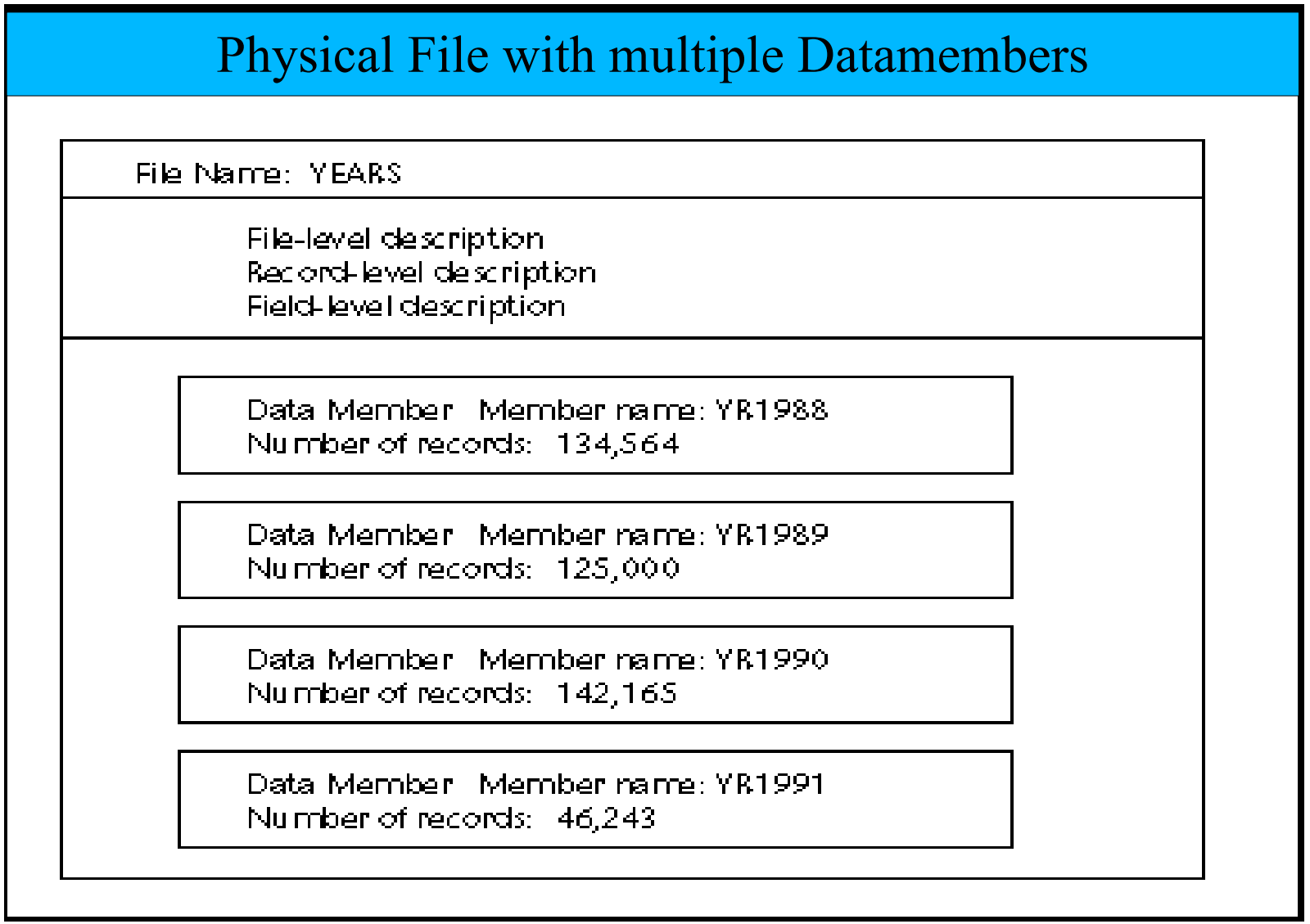


OS/400 Files

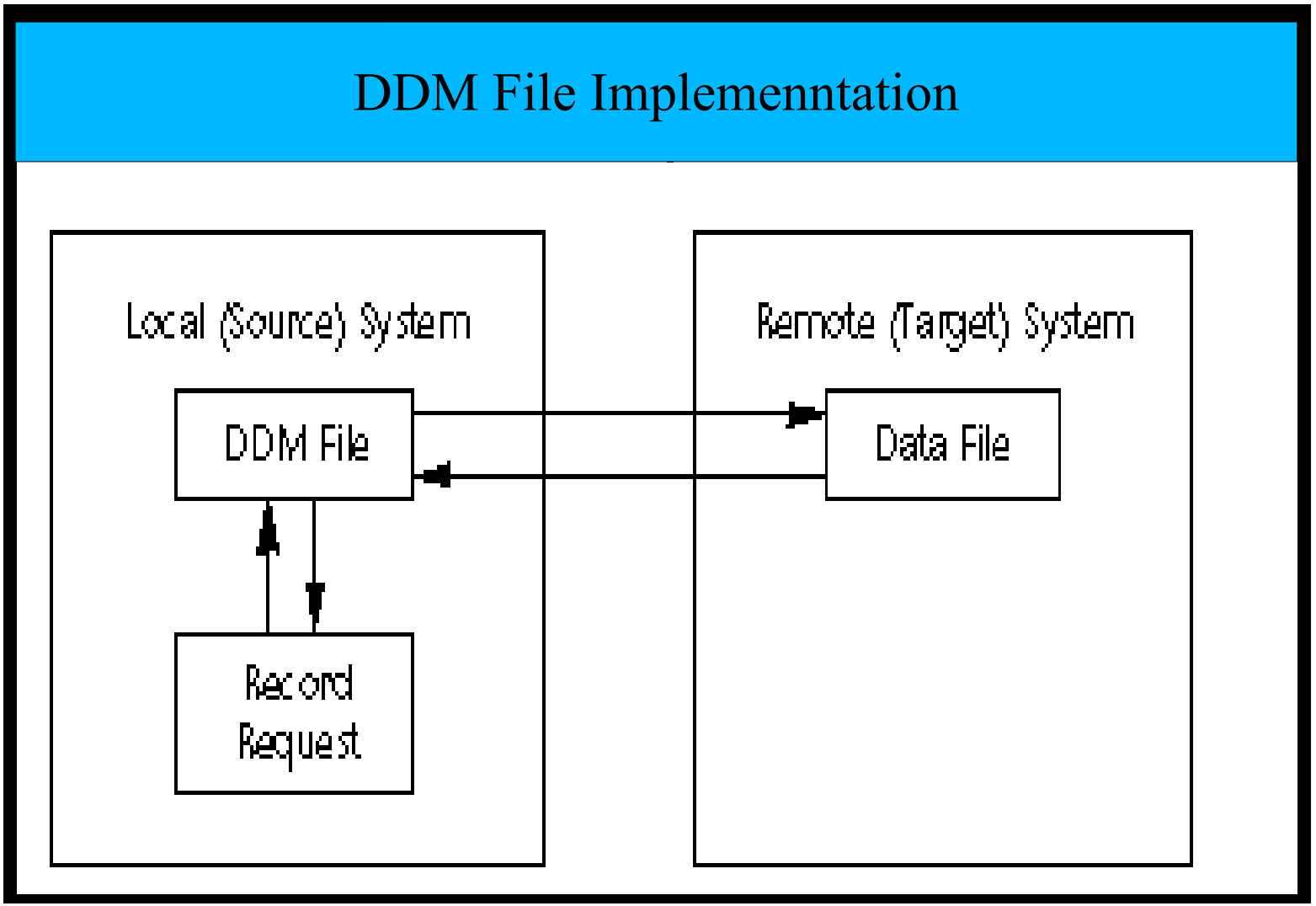
Source File Organisation



OS/400 Files



OS/400 Files

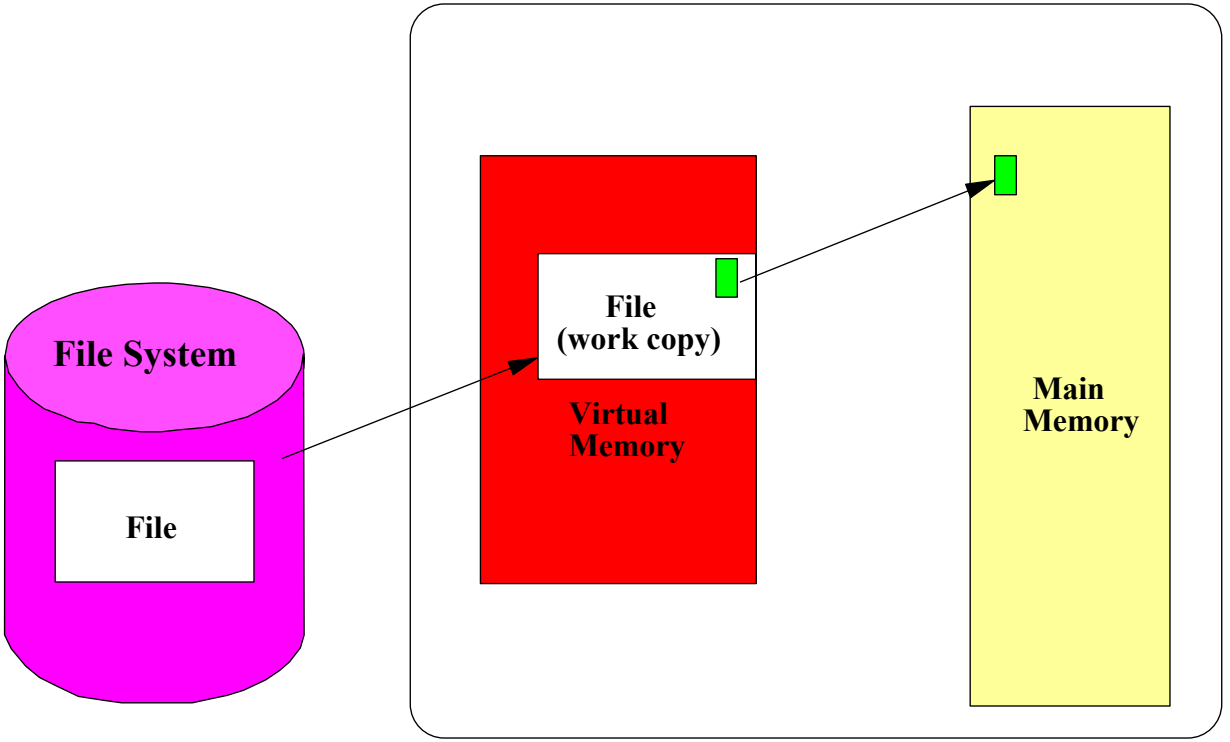


3.3 Single-Level Store

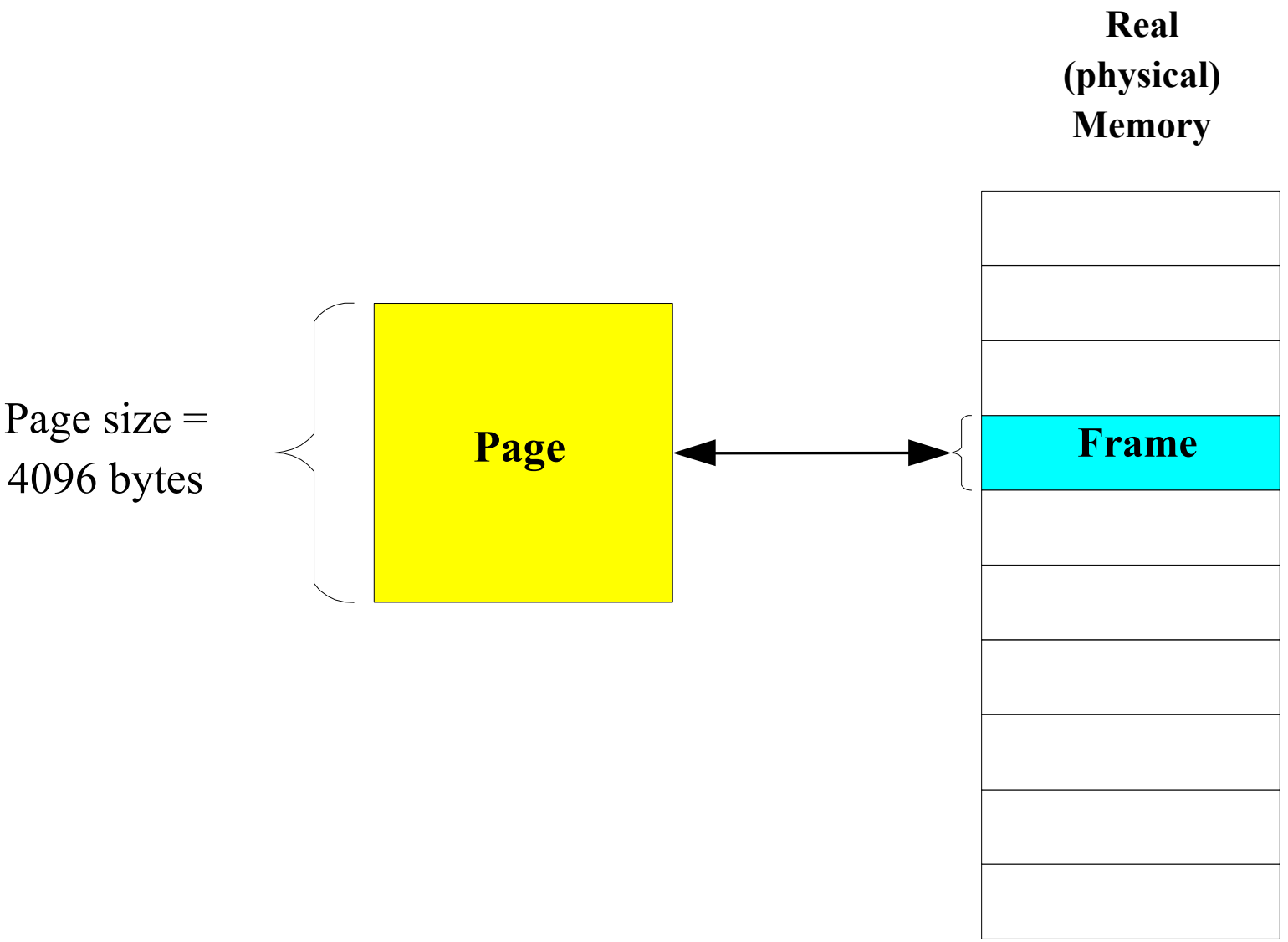
… is not about large address space;
it‘s about sharing.

Conventional Approach

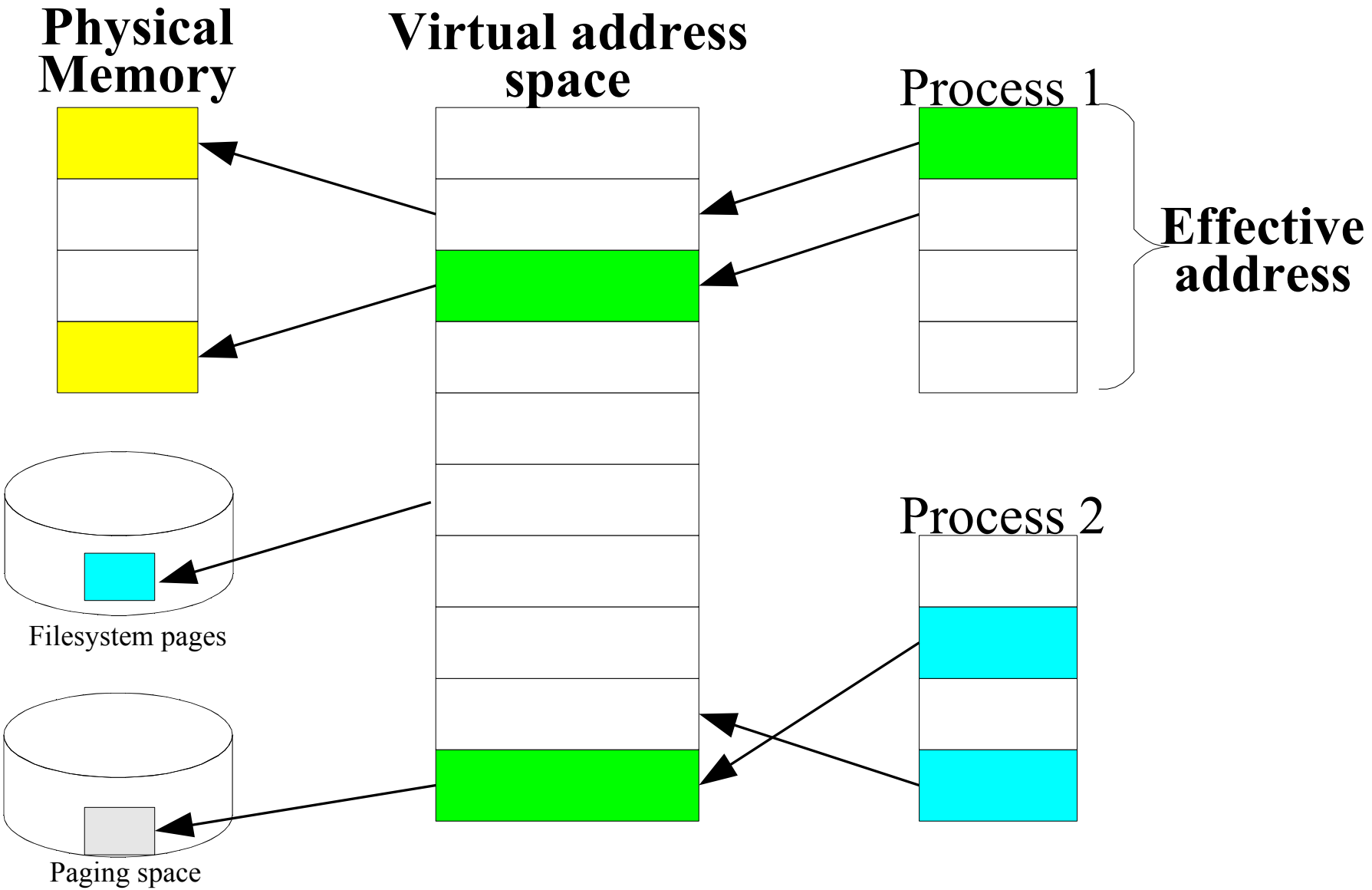
- Non-shared address space model
  - Each user gets a separate address space
- a file system outside of virtual memory (two-level store)
  - Anything have to be moved into virtual memory before it can be used or changed



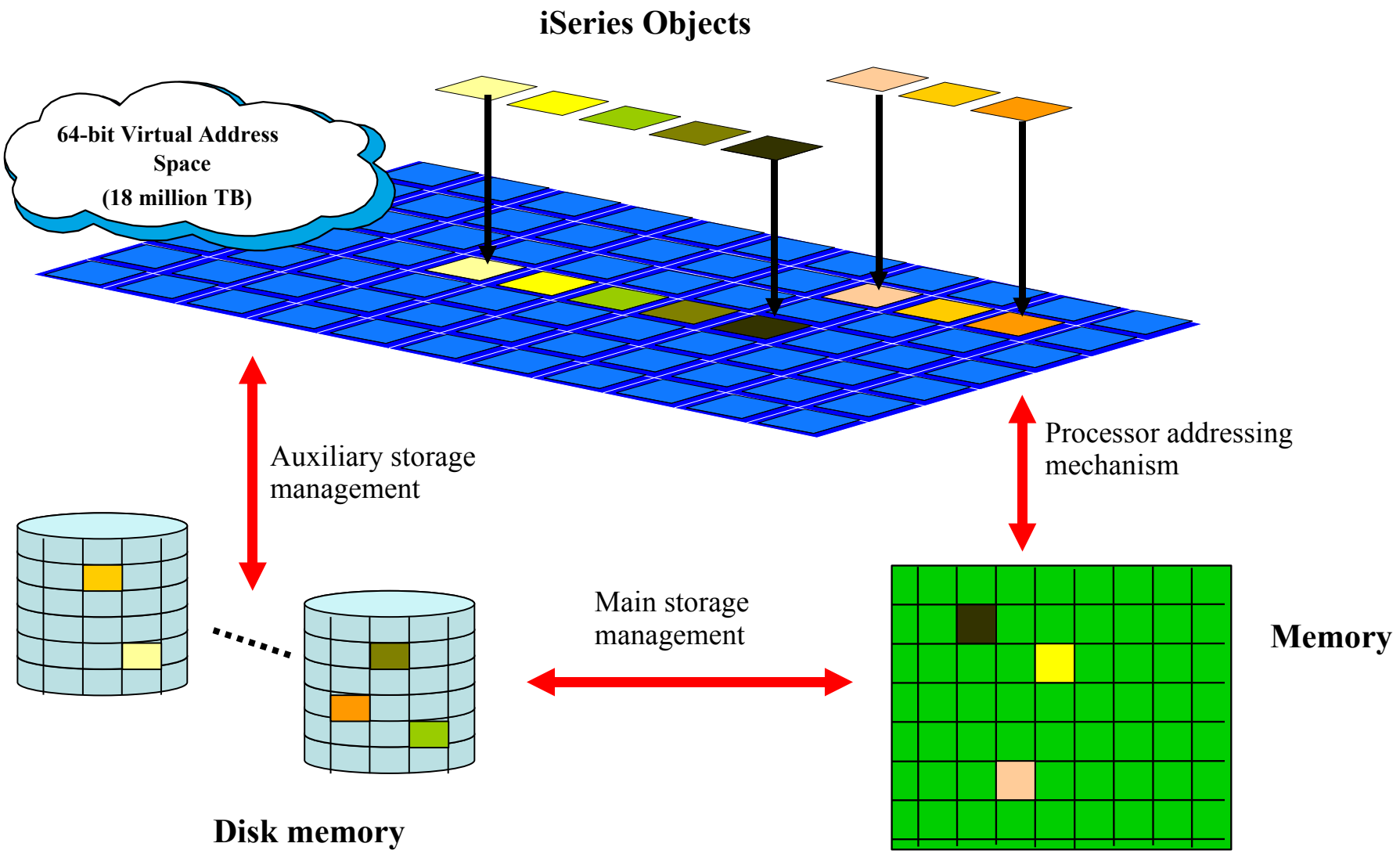
Pages and Frames



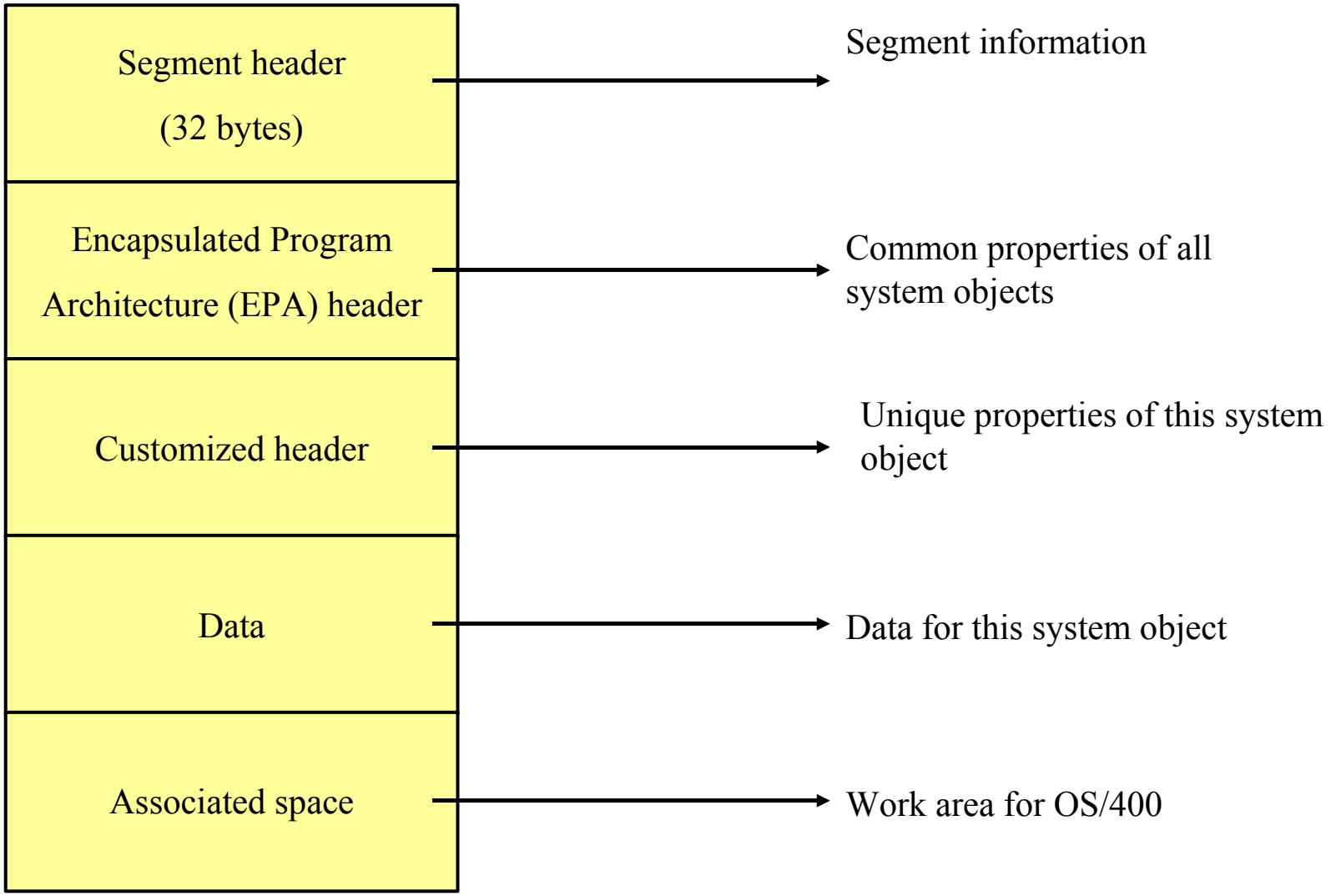
Address Space of Conventional Systems



Objects in the Single Level Store



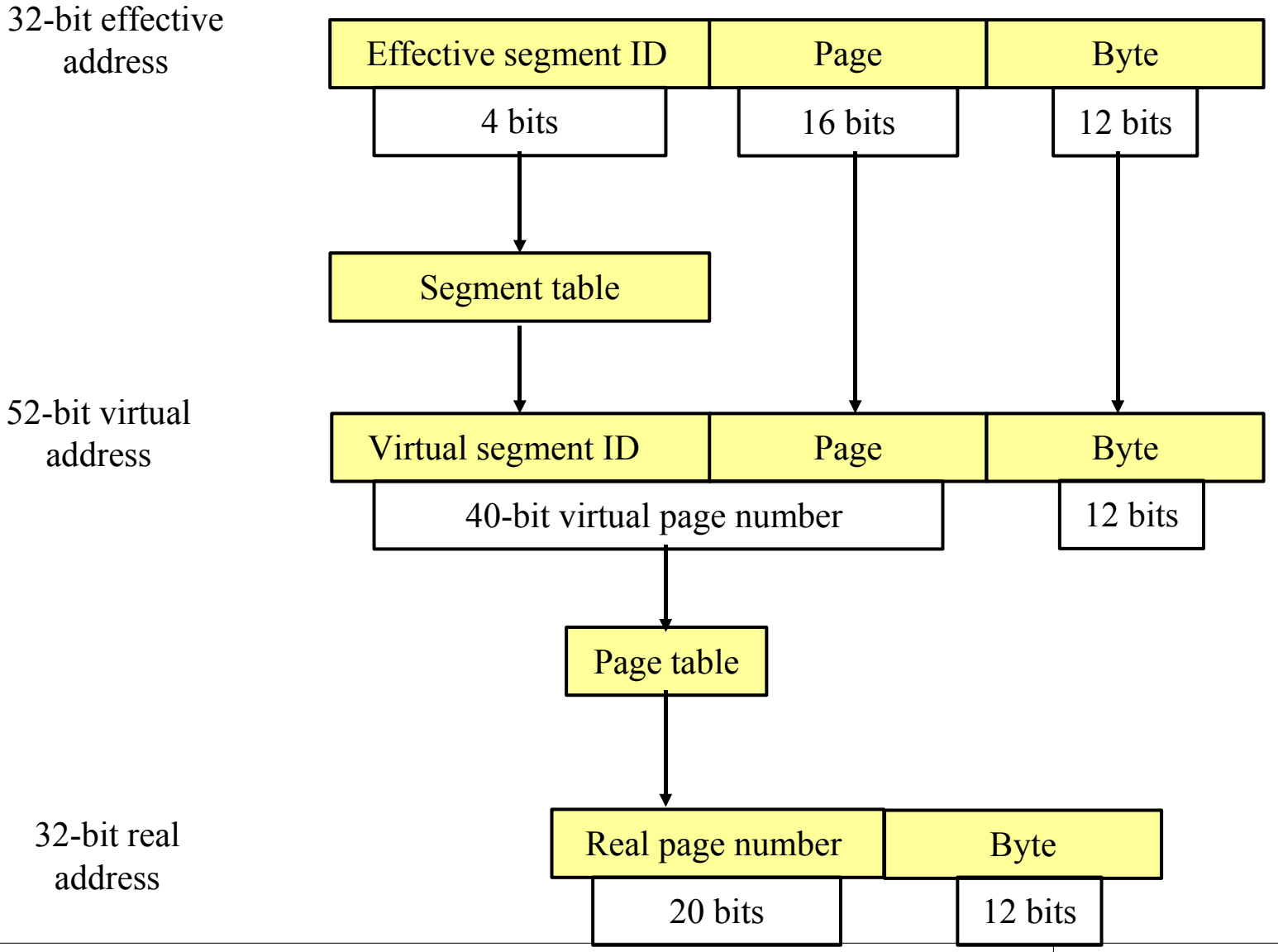
System Object Structure



Translating Addresses

Step	Action
1	An effective address is referenced by a process or by the kernel.
2	The hardware translates the address into a system wide virtual address.
3	The page containing the virtual address is located in physical memory or on disk.
4	If the page is currently located on disk, a free frame is found in physical memory, and the page is loaded into this frame.
5	The memory operation requested by the process or kernel is completed on the physical memory.

Segment-Relative Addressing

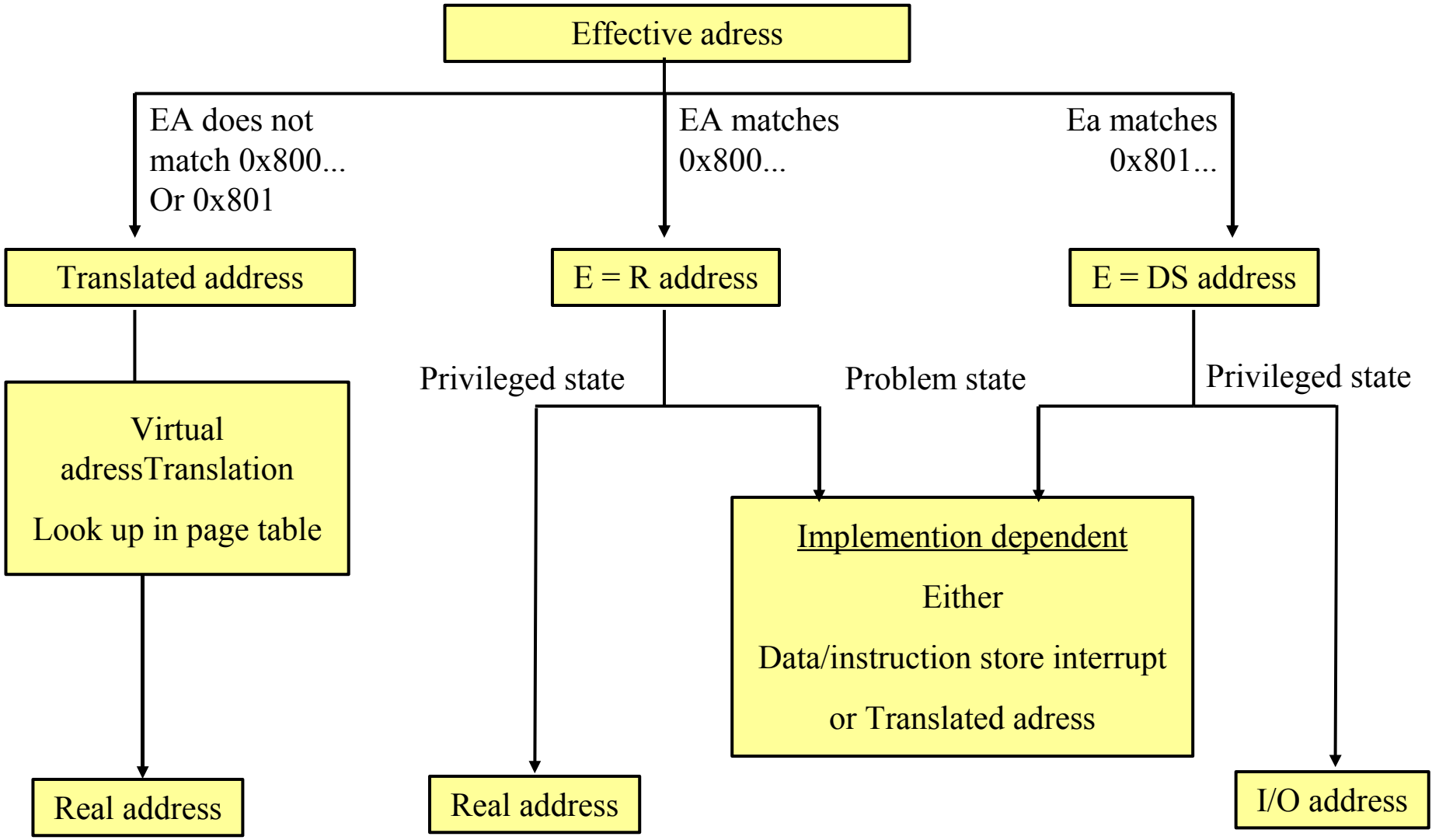


iSeries Memory Model Characteristic

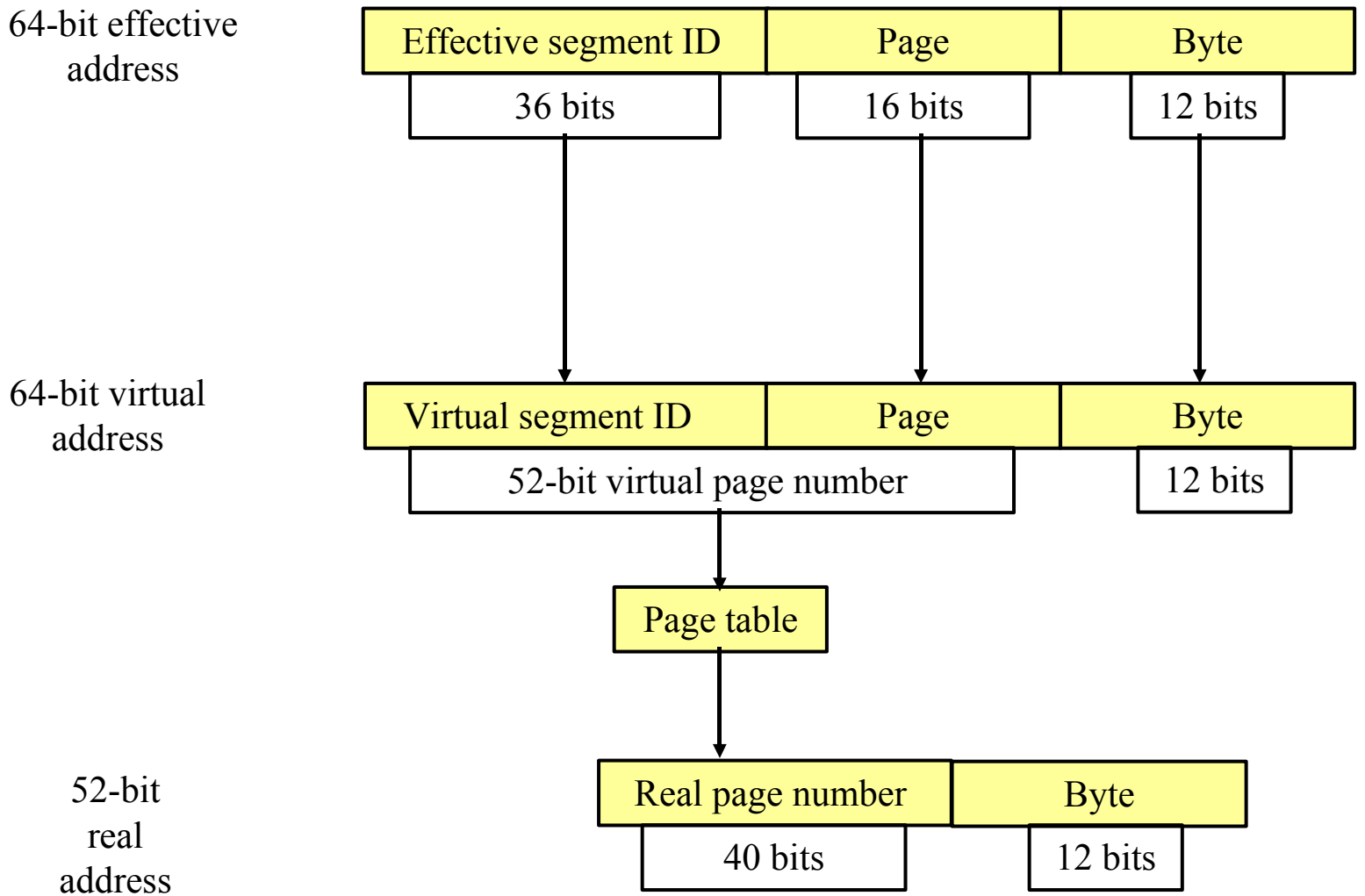
- The page size is 2^{12} bytes (4 KB)
- The effective address range is 2^{64} bytes
  - The effective segment size is 2^{24} bytes (16 MB)
  - The number of effective segments is 2^{40}
- There are two special types of effective addresses
  - 0x800 Effective = Real addresses map all real memory
    (-> The real address range is 2^{52} bytes)
  - 0x801 Effective = Direct-Store addresses map I/O space
- The virtual address range is 2^{64} bytes
  - The number of virtual segments is 2^{40}-2^{29}
  - The size of virtual segments is 2^{24} bytes (16 MB)

No need for effective-to-virtual translation!

iSeries Address Translation



Steps Involved in Address Translation



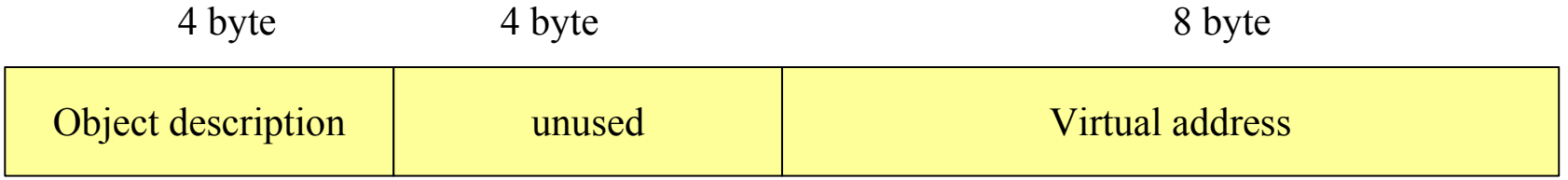
3.3.3 Pointer Protection

Aside from performance, the biggest advantage of a single-level store is that everything can be shared; it‘s biggest disadvantage is that everything can be shared.

Tag-bit

- Pointers should not be modified by MI programs
  - MI programs can use object names and should not modify resolved pointers containing a virtual address of an object
- Usage of a memory protection bit (tag-bit)
  - only privileged instructions can modify the contents of a pointer
  - Tag-bits are used to detect modifications

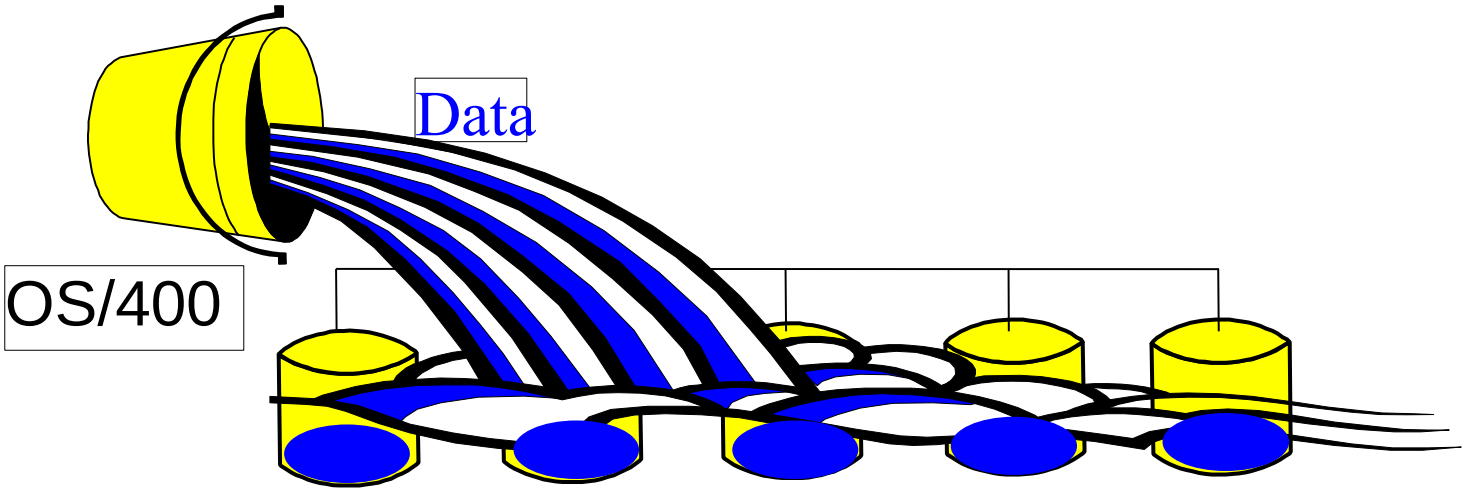
MI Pointer in Memory



- Representation in memory:
  - 2 64-bit words
  - 8 ECC*-bits/word
  - one tag-bit/word
- Every write to memory means also
  - create and store the ECC bits
  - turns off the tag-bits
- Only privileged instructions can set the tag-bits
  - MI transalator does not generate tag instructions

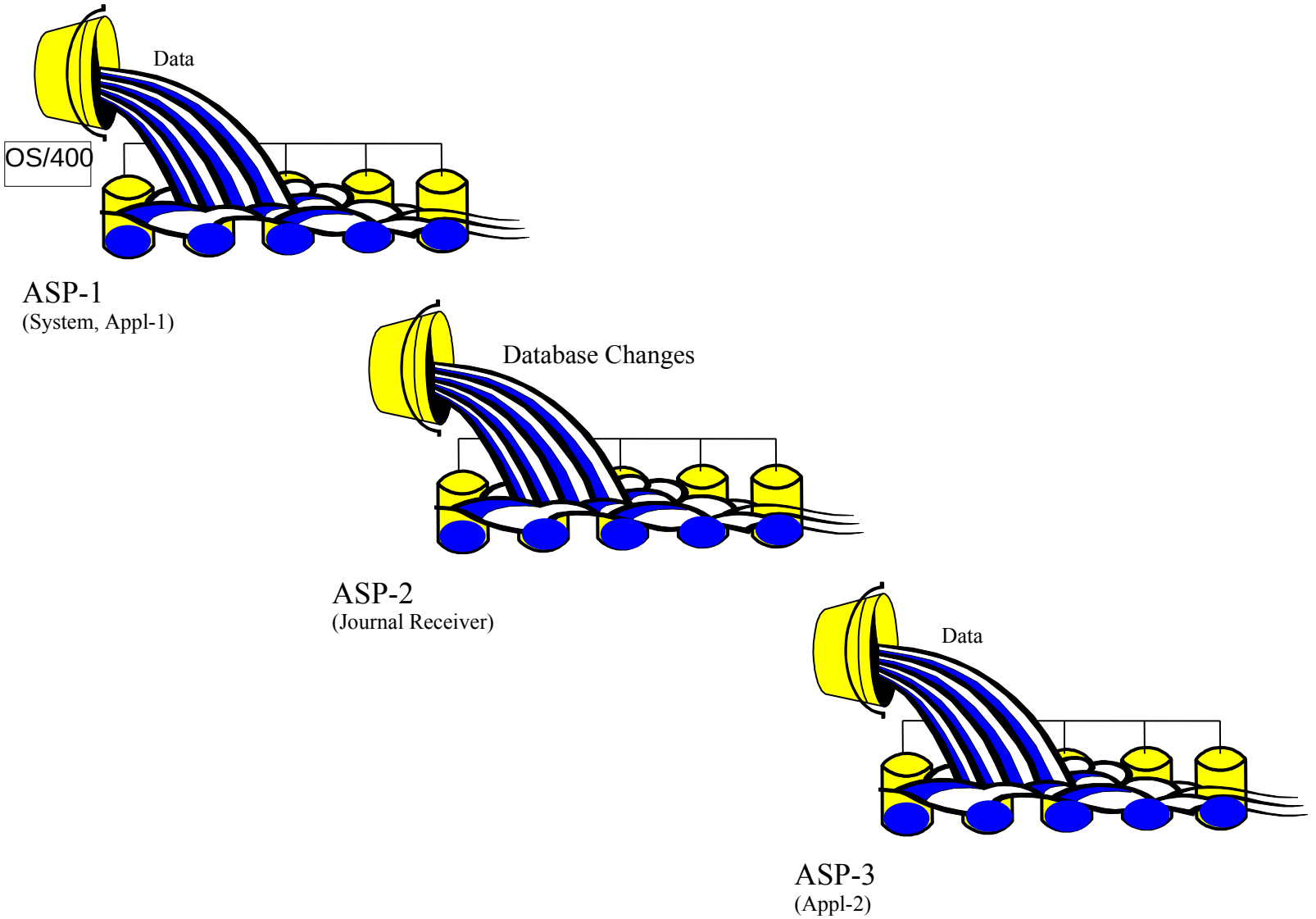
* Error Correction Code

Single-level Store and Disk Management



- All data is spread across available disk arms
  - Optimum performance - automatically
- Not all information is necessarily contiguous (1MB)
  - Improved performance (Balanced disk arm utilization)
- Optional rebalancing
  - space/arm utilization
- Minimal Database Administration
  - Information accessed by name not hardware address

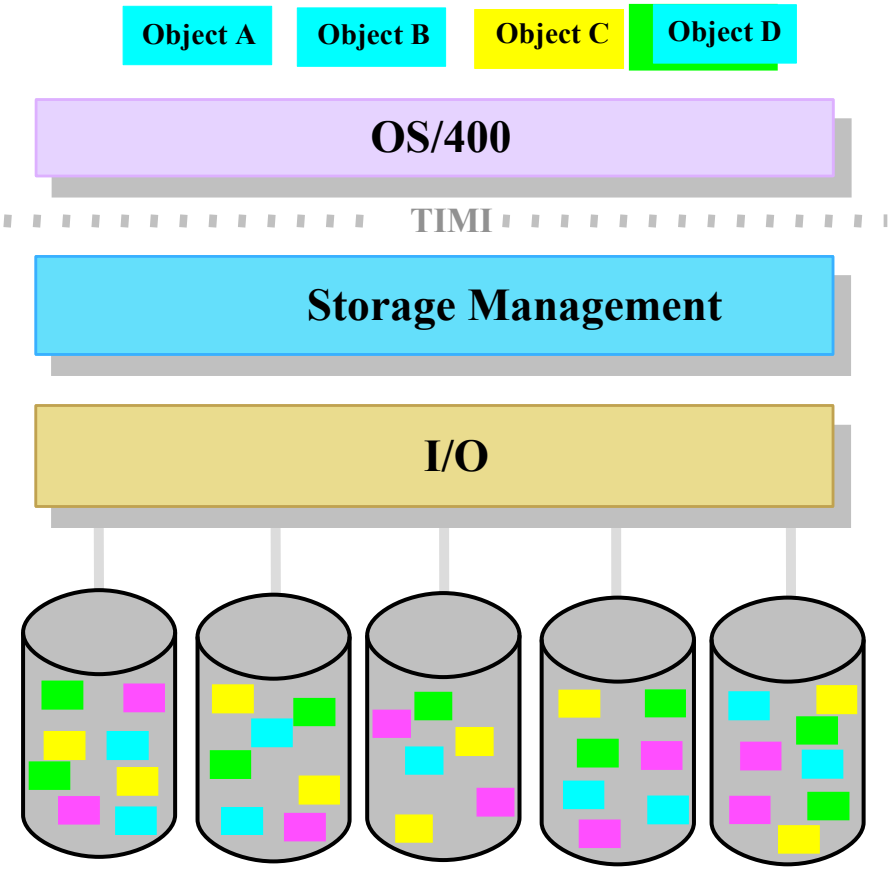
Auxilary Storage Pool (ASP)



iSeries Storage Architecture

- Data is scattered across all disks in a disk pool
- Good performance due to Parallel I/O
- Disks fill evenly
  - No manual data placement
  - No individual "disk full" conditions to handle
- Newly added disk capacity is utilized automatically
- No continuous disk performance monitoring
- Automatic disk operations eliminating DBA needs

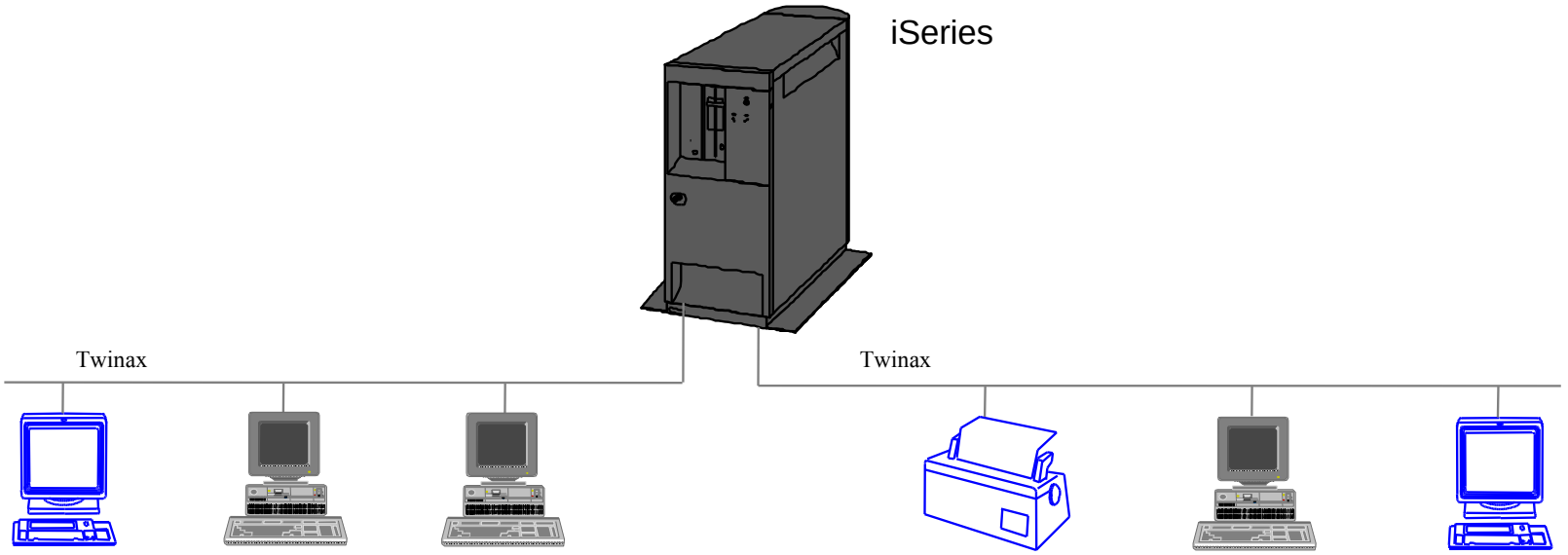
Single Level Store



3.4 Hardware Integration

3.4.1 Hierachy of Micro Processors

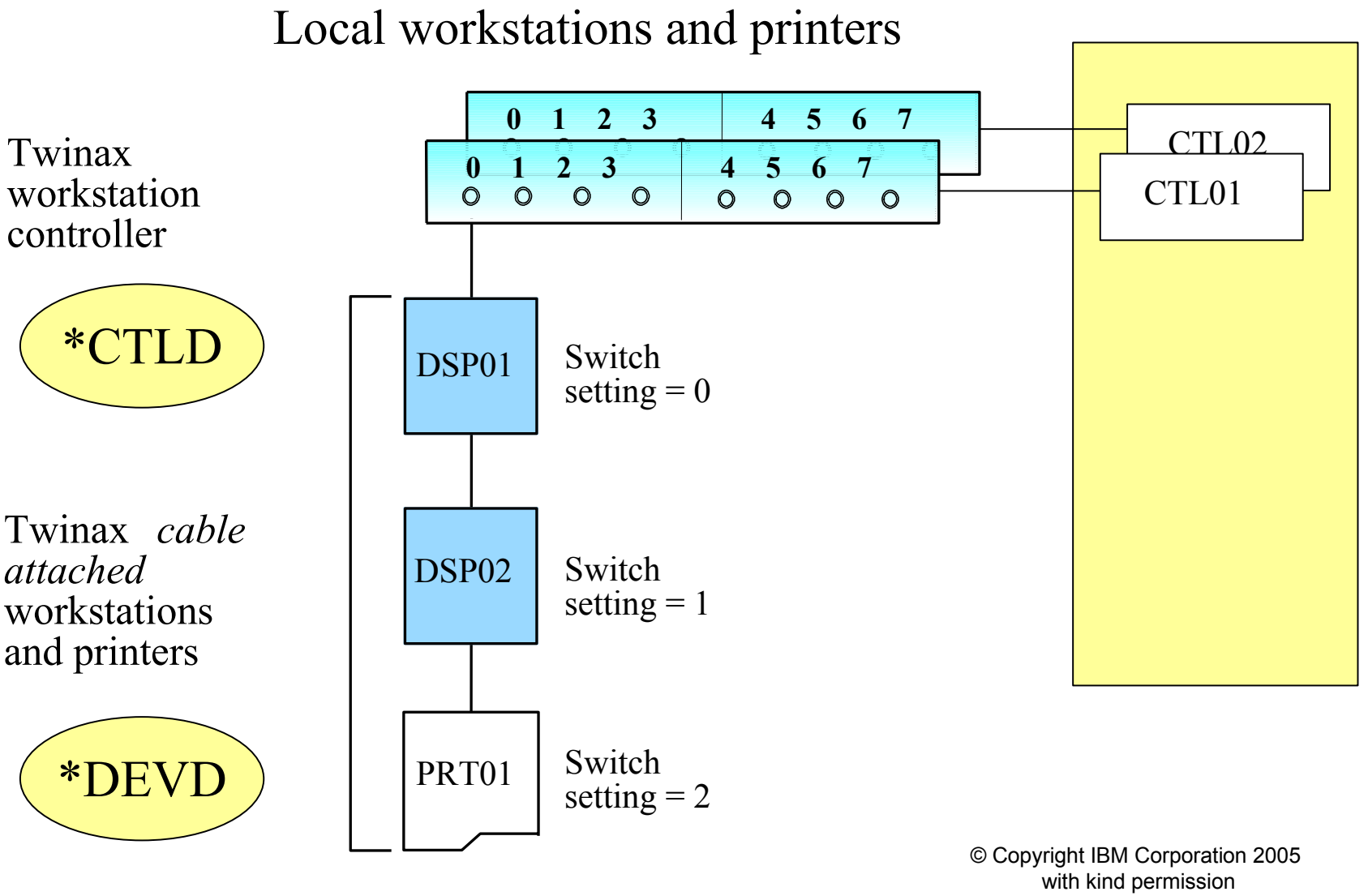
Local Attached Devices



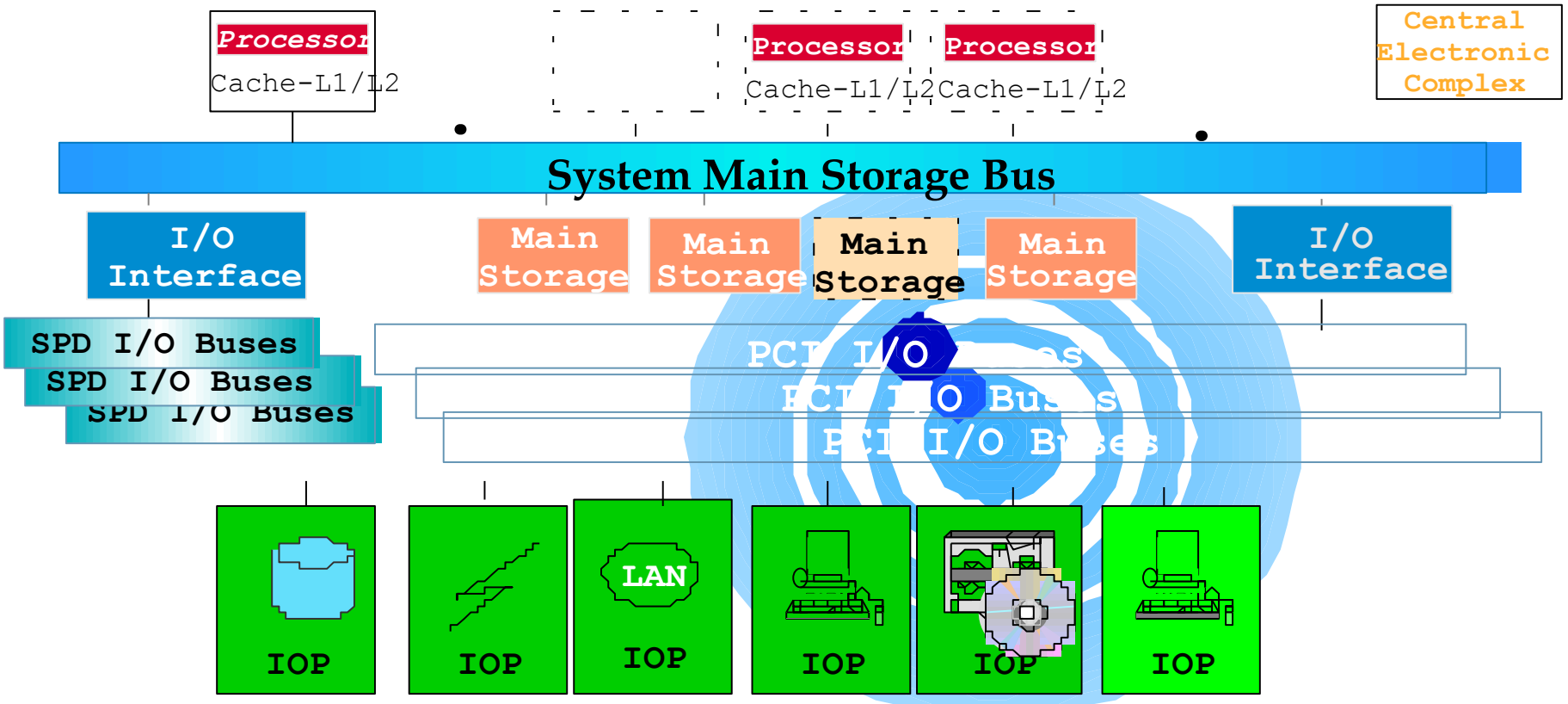
iSeries Terminals are connected via Twinax cabling topology to a central system

- Display and printer devices
- PCs with Twinax adapter and emulation software
- Seven Twinax station addresses per iSeries 5250 workstation controller port
- Twinax cable, up to 1500 m, UTP cable shorter distance

Locally Attached: Workstation Controller (Twinaxial)



Hierachy of Micro Processors

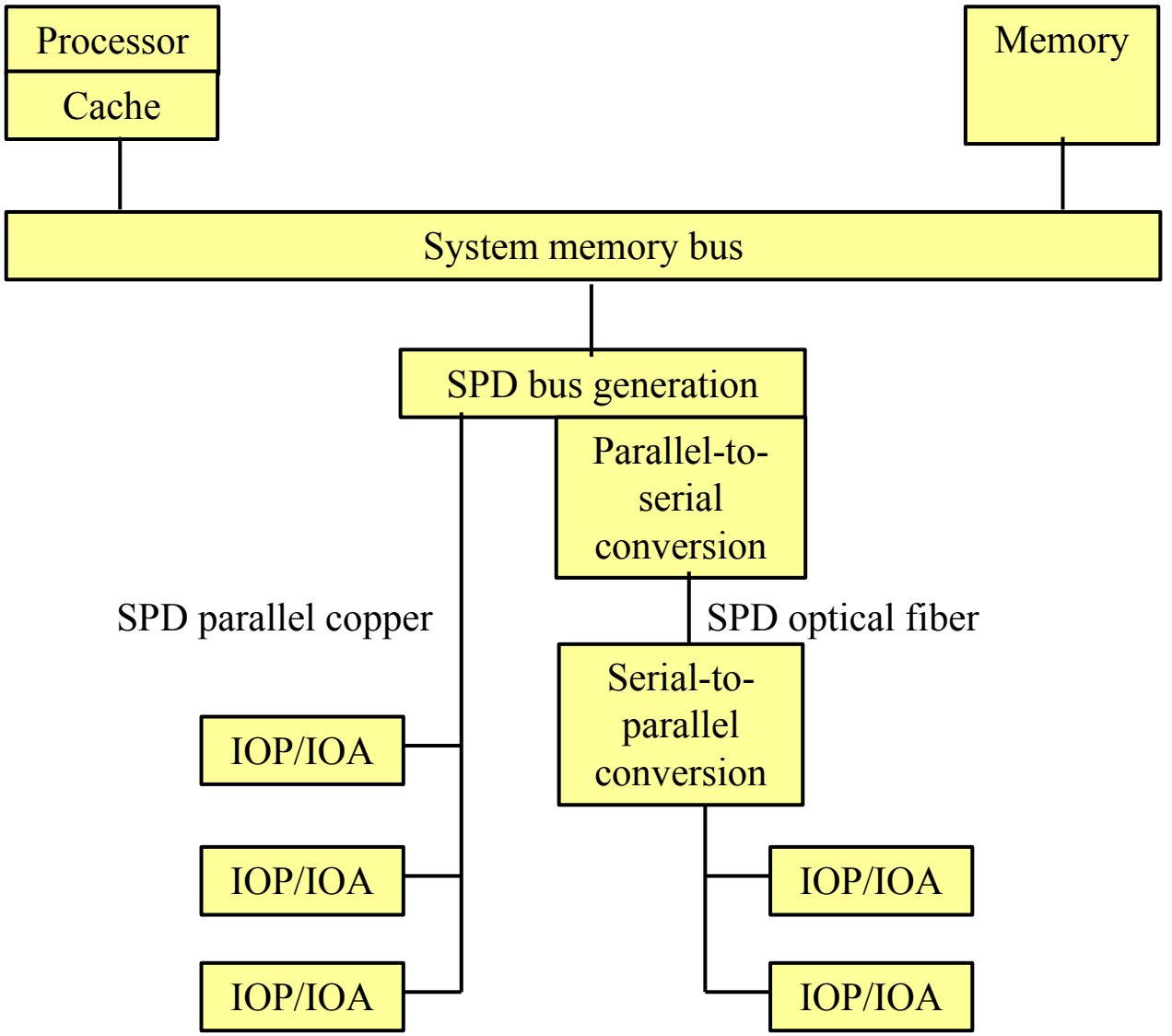


- SPD-System Product Division
- PCI-Peripheral Component Interconnect

3.4.2 iSeries I/O Structure

Remember the Secret of Success …

Original SPD I/O Structures

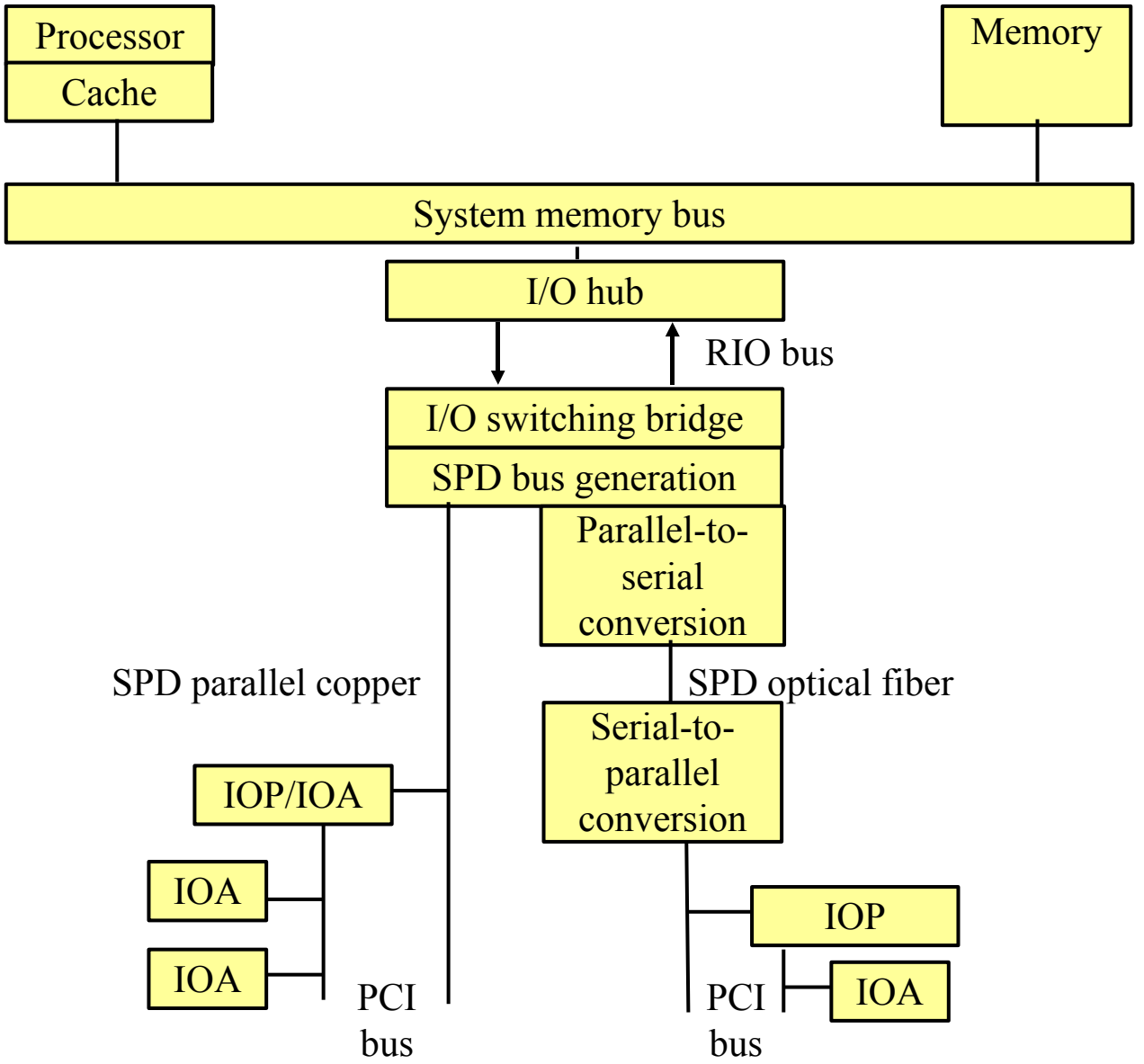


SPD-PCI Transition

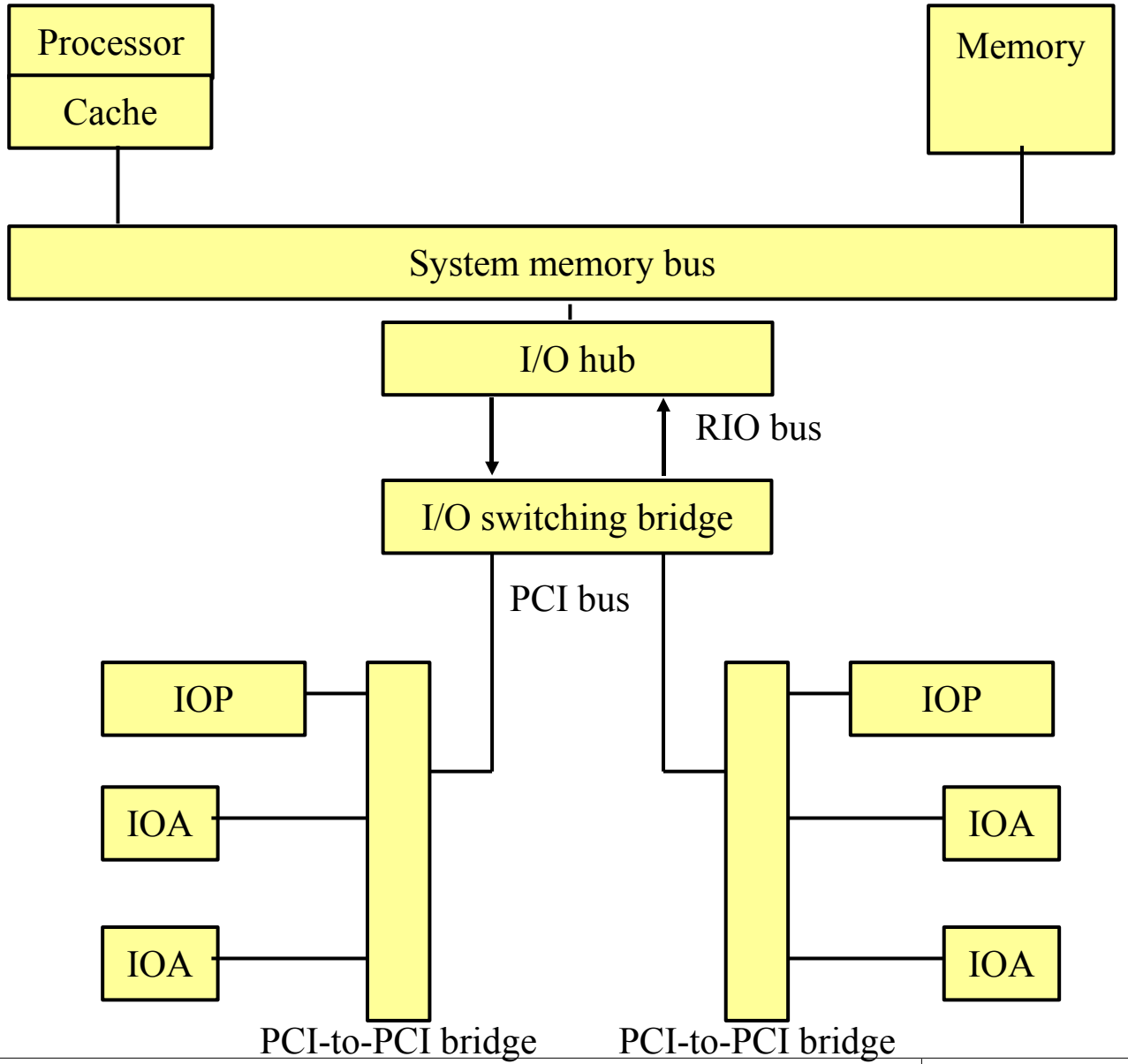
- SPD – System Product Devision

- RIO – Remote I/O
- PCI – Peripheral Component Interface

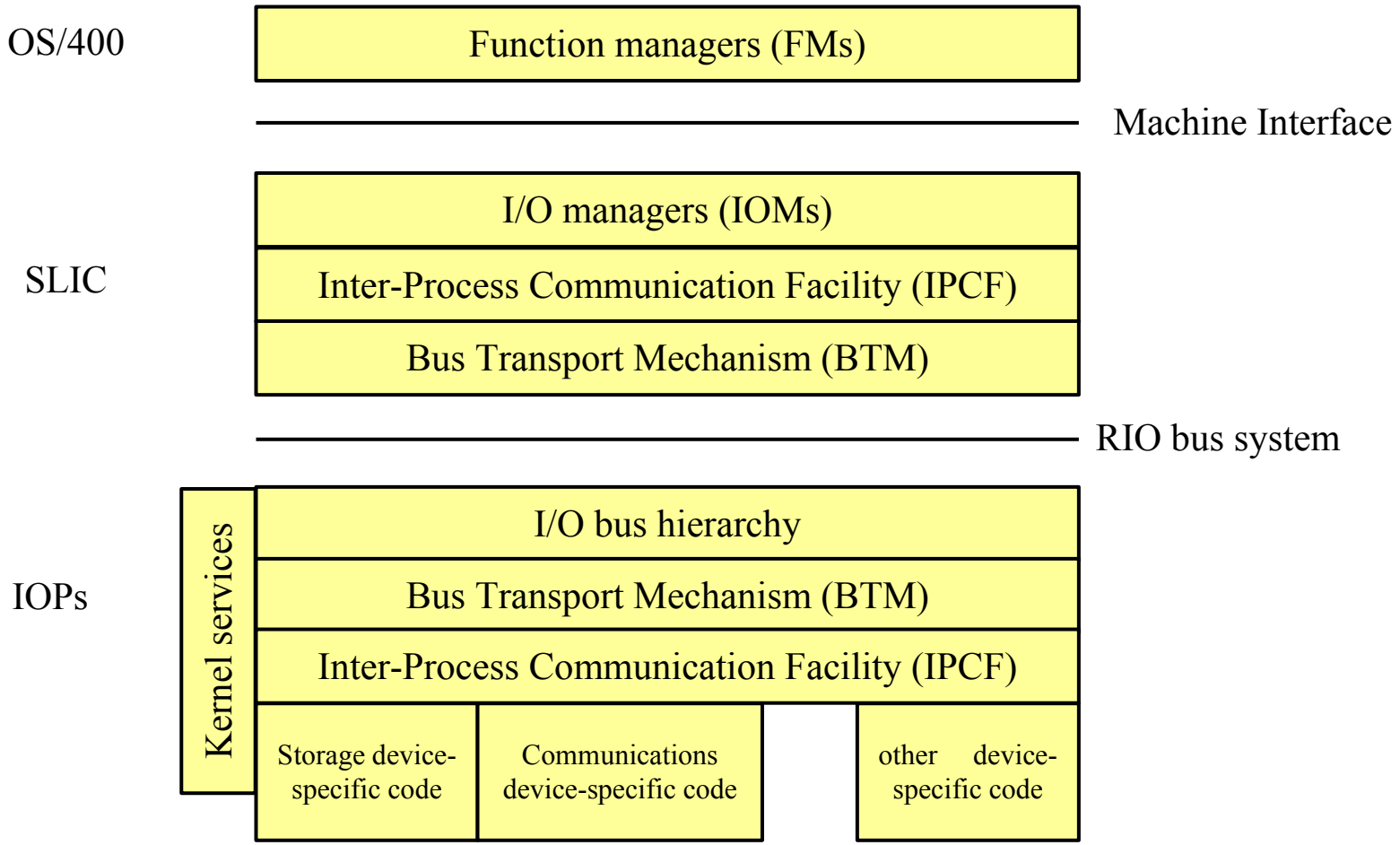
Transition I/O Structure



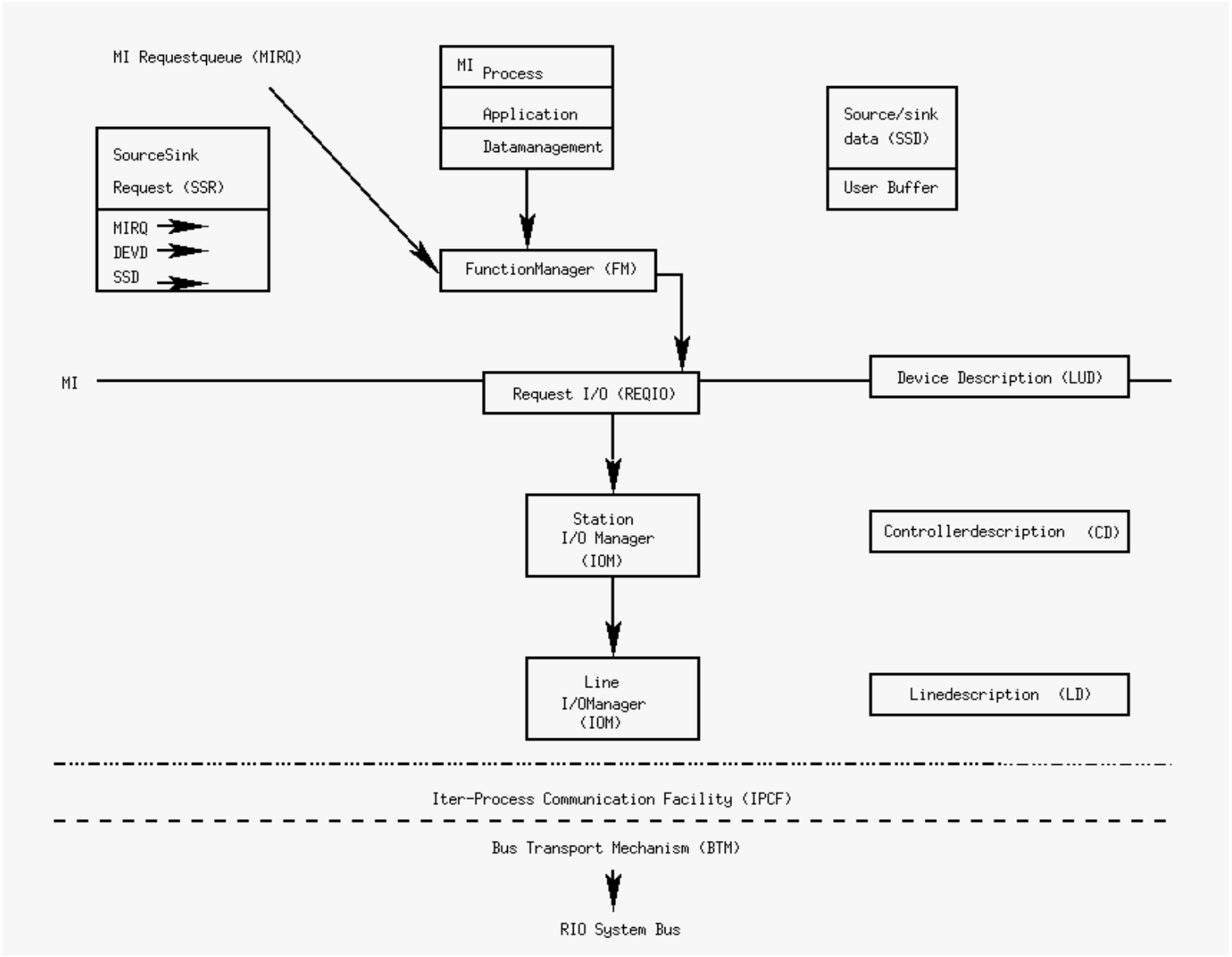
RIO/PCI I/O Structure



I/O Software Structure



iSeries I/O SW-Modules



OS/400 Commands

Eine Übersicht der Actions, bzw. Verbs:

Verb	Funktion
ADD	Add
CHG	Change
CPY	Copy
CRT	Create
DLT	Delete
DSP	Display
END	Beenden eines per *STR* gestarteten Programms
GRT	Grant
MOV	Move
RMV	Remove
RST	Restore
RTV	Retrieve
RVK	Revoke
SET	Set
SND	Send
STR	Start eines Programms
WRK	Work

OS/400 Commands

Die Subjects setzen sich aus Abkürzungen zusammen, wie etwa in o.g. Beispiel der Line Description, LIND. Einige Beispiele sind:

Subject	Beschreibung
LIND	Line Description
PDM	PDM
LIBLE	Library List Entry
SRCPF	Source Physical File
MSG	Message
ACTJOB	Active Job
OBJPDM	Objekt, PDM

Die Wortteile für sich ergeben kein gültiges Kommande. Erst in sinnvoller Kombination werden daraus Kommandos der so genannten Commandline Language, CL genannt. Beispiele seien:

Kommando	Beschreibung	Funktion
WRKACTJOB	Work Active Job	
ADDLIBLE	Add Library List Entry	
DSPMSG	Display Message	Anzeige von Messages
DSPJOB	Display Job	Anzeige der laufenden Jobs
DSPJOBLOG	Display Job Log	Anzeige des Joblogs
WRKLIND	Work Line Description	Konfiguarion der Leitungsbeschreibungen

OS/400 Commands

```
                          Send Message (SNDMSG)
Type choices, press Enter.
Message text .................MSG          Hello World
                                          ------------------------
                                          ------------------------
                                          ------------------------
                                          ------------------------
                                          ------------------------
-------------To user profile ..............  TOUSR    someone
                      Additional Parameters
To message queue ..............  TOMSQ       ----------
   Library ...................                *LIBL____
                      * for more values      ----------
                                             *LIBL____
Message type ..................  MSGTYPE     *INFO_____
Message queue to get reply ...  RPYMSGQ      *voyager__
   Library ...................                ----------
```

SNDMSG MSG('Hello World') TOUSER(someone) MSGTYPE(*INFO) RPYMSG(*voyager)

OS/400 Userprofile

USRPRF Beschreibt den Profilenamen, gewissermaßen den Usernamen

PASSWORD Das Passwort soll nur dem Benutzer bekannt sein. Es kann gesetzt werden auf

**NONE* kein Passwort, kein Login. Bentuzt besipielsweise für Gruppenprofile etc.

**USRPRF* Username ist Passwort, inititales Passwort

Das Passwort selbst

Die Qualität des zu verwendenden Passworts ist enforcierbar durch folgende System Values:

QPWDMINLEN Einstellung der minimalen Passwortlaenge.

QPWDRQDDGT Mindestens ein Digit ist gefordert.

QPWDLMTAJC Benachbarte Nummern werden verhindert.

QPWDLMTREP Ein Alpha-Character darf nur einfach auftreten.

OS/400 Userprofile

PWDEXP Setzt ein Passwort auf expiered. Bei Neueinrichtung oftmals verwendet, um das Neusetzen eines Passwortes zu erzwingen.

STATUS Status des Benutzers: enabled oder disabled. Wird bei entsprechender Einstellung bei wiederholten Fehlversuchen automatisch auf disabled gesetzt.

USRCLS User Class. Es wird unterschieden in:

****PGMR*** Programmer

****SECOFR*** Security Officer, (root-User)

****SECADM*** Security Administrator

****SYSOPR*** System Operator

****USER*** User

OS/400 Userprofile

SPCAUT Special Authority. Die Defaultrechte werden über User Classes definiert. Ein Setzen von Special Authorities modifiziert diese. Es wird unterschieden in:

****ALLOBJ***

****SECADM***

****SAVSYS***

****SERVICE***

Sign On Optionen Die Optionen für den initialen Sign On sind setzbar im Userprofil. Es sind dies:

CURLIB Current Library

INLPGM Initial Program

INLMNU Initial Menu

LMTCPB Limit Capabilities

Printer/Output Handling Die Ausgabe- und Printdevices sind mit den Parametern *PRTDEV*, Print Device und *OUTQ*, Output Queue, definierbar. Dabei sind

OUTQ Ausgabedevice, per Default **WRKSTN*

PRTDEV Ausgabedevice für Druckjobs