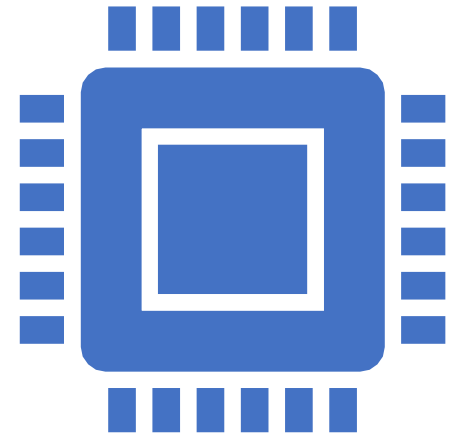


# SOEN 6431

## Software Comprehension and Maintenance

Software and Data Migration



# Legacy systems

Software systems that are developed specially for an organisation have a long lifetime

Many software systems that are still in use were developed many years ago using technologies that are now obsolete

These systems are still business critical that is, they are essential for the normal functioning of the business

They have been given the name legacy systems

# Legacy system change

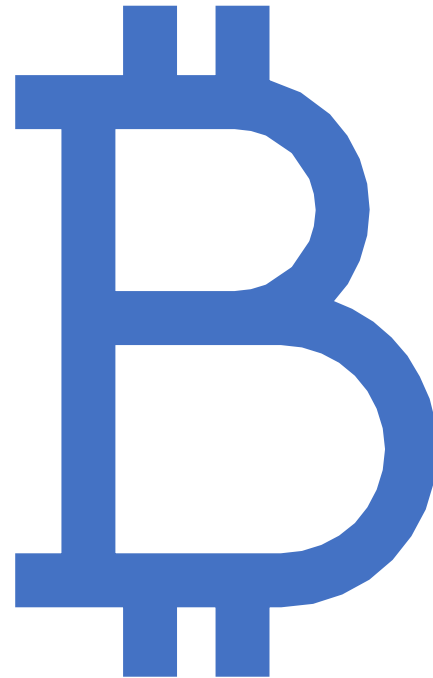
- Systems must change in order to remain useful
- However, changing legacy systems is often expensive
  - Different parts implemented by different teams so no consistent programming style
  - The system may use an obsolete programming language
  - The system documentation is often out-of-date
  - The system structure may be corrupted by many years of maintenance
  - Techniques to save space or increase speed at the expense of understandability may have been used
  - File structures used may be incompatible

# Legacy system replacement

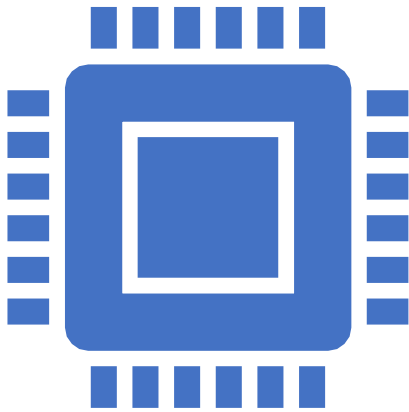
- There is a significant business risk in simply scrapping a legacy system and replacing it with a system that has been developed using modern technology
  - Legacy systems rarely have a complete specification. During their lifetime they have undergone major changes which may not have been documented
  - Business processes are reliant on the legacy system
  - The system may embed business rules that are not formally documented elsewhere
  - New software development is risky and may not be successful

# The legacy dilemma

- It is expensive and risky to replace the legacy system
- It is expensive to maintain the legacy system
- Businesses must weigh up the costs and risks and may choose to extend the system lifetime using techniques such as re-engineering.

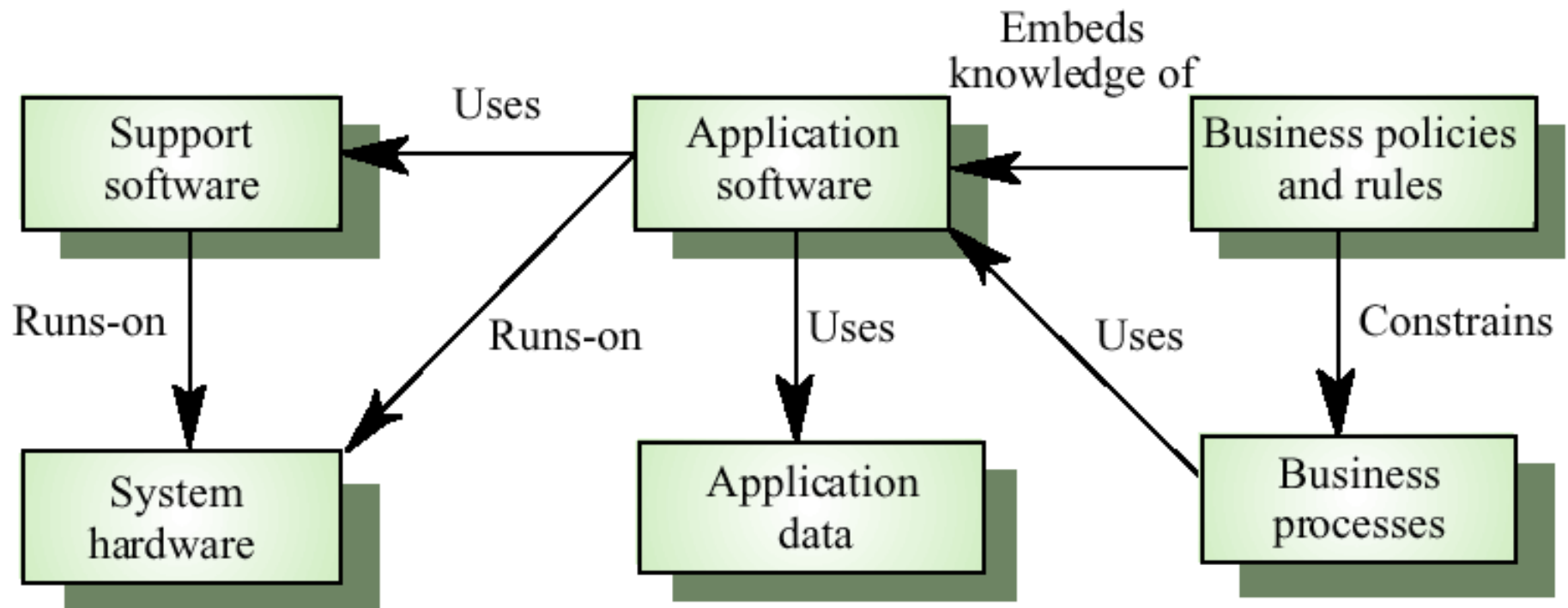


# Legacy system structures



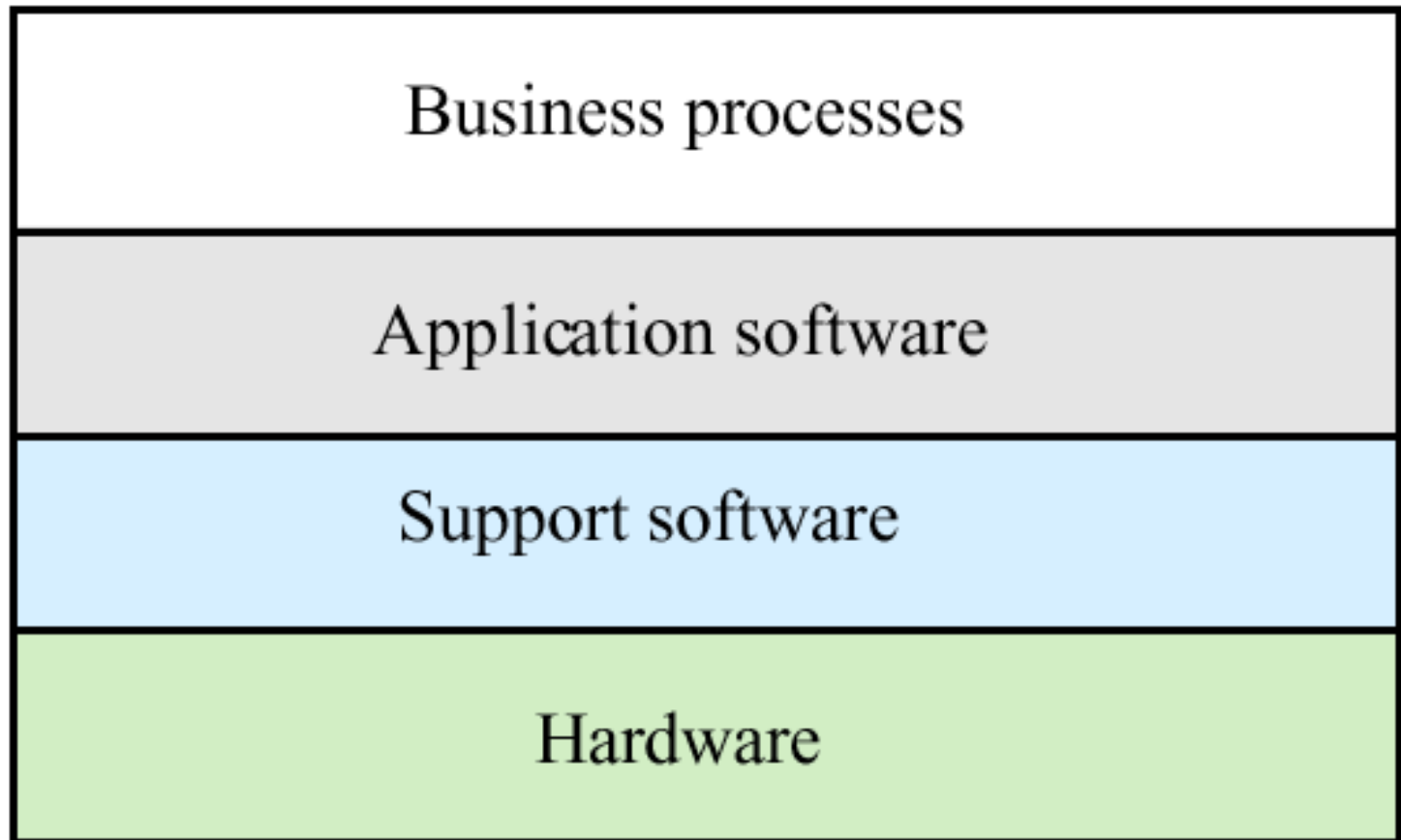
- Legacy systems can be considered to be socio-technical systems and not simply software systems
  - System hardware - may be mainframe hardware
  - Support software - operating systems and utilities
  - Application software - several different programs
  - Application data - data used by these programs that is often critical business information
  - Business processes - the processes that support a business objective and which rely on the legacy software and hardware
  - Business policies and rules - constraints on business operations

# Legacy system components



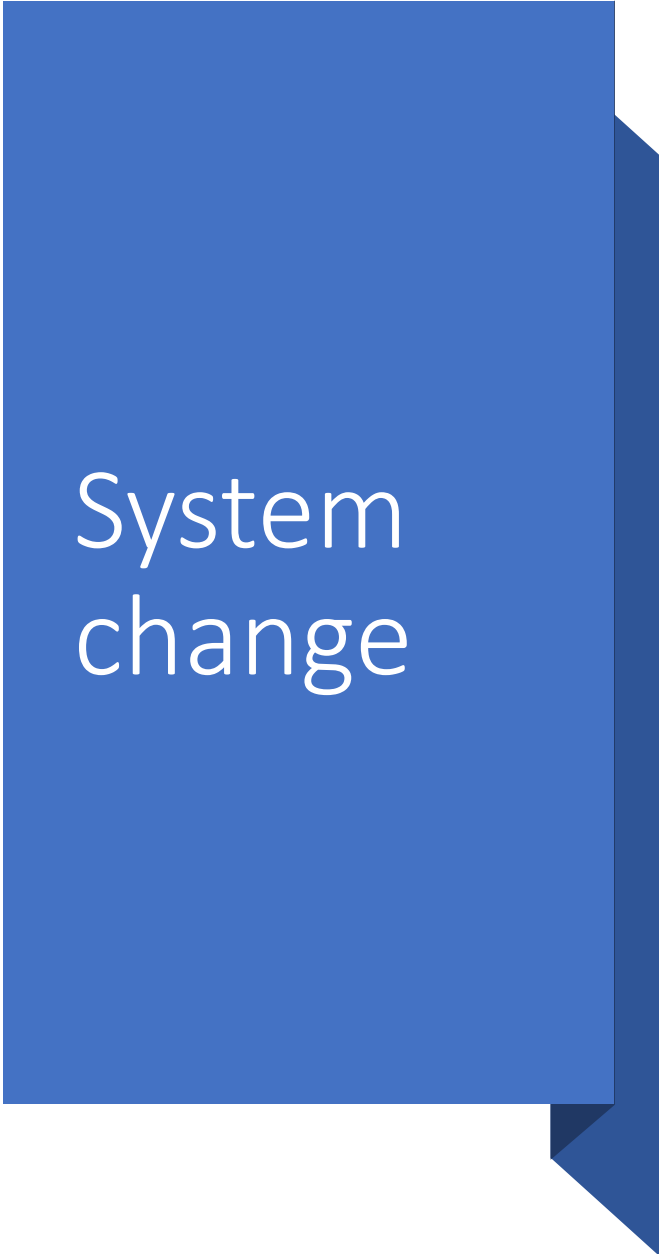
# Layered model

## **Socio-technical system**



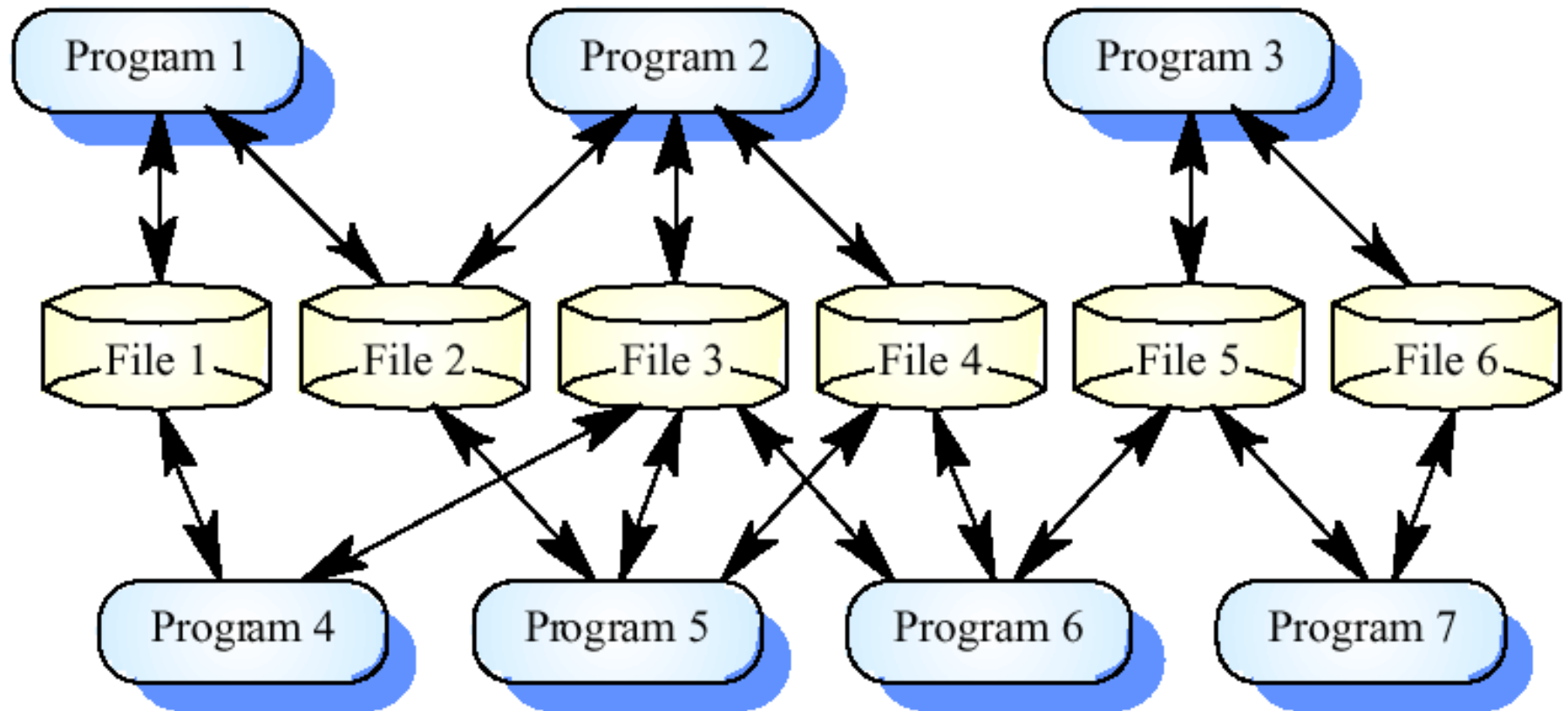


- In principle, it should be possible to replace a layer in the system leaving the other layers unchanged
- In practice, this is usually impossible
  - Changing one layer introduces new facilities and higher level layers must then change to make use of these
  - Changing the software may slow it down so hardware changes are then required
  - It is often impossible to maintain hardware interfaces because of the wide gap between mainframes and client-server systems

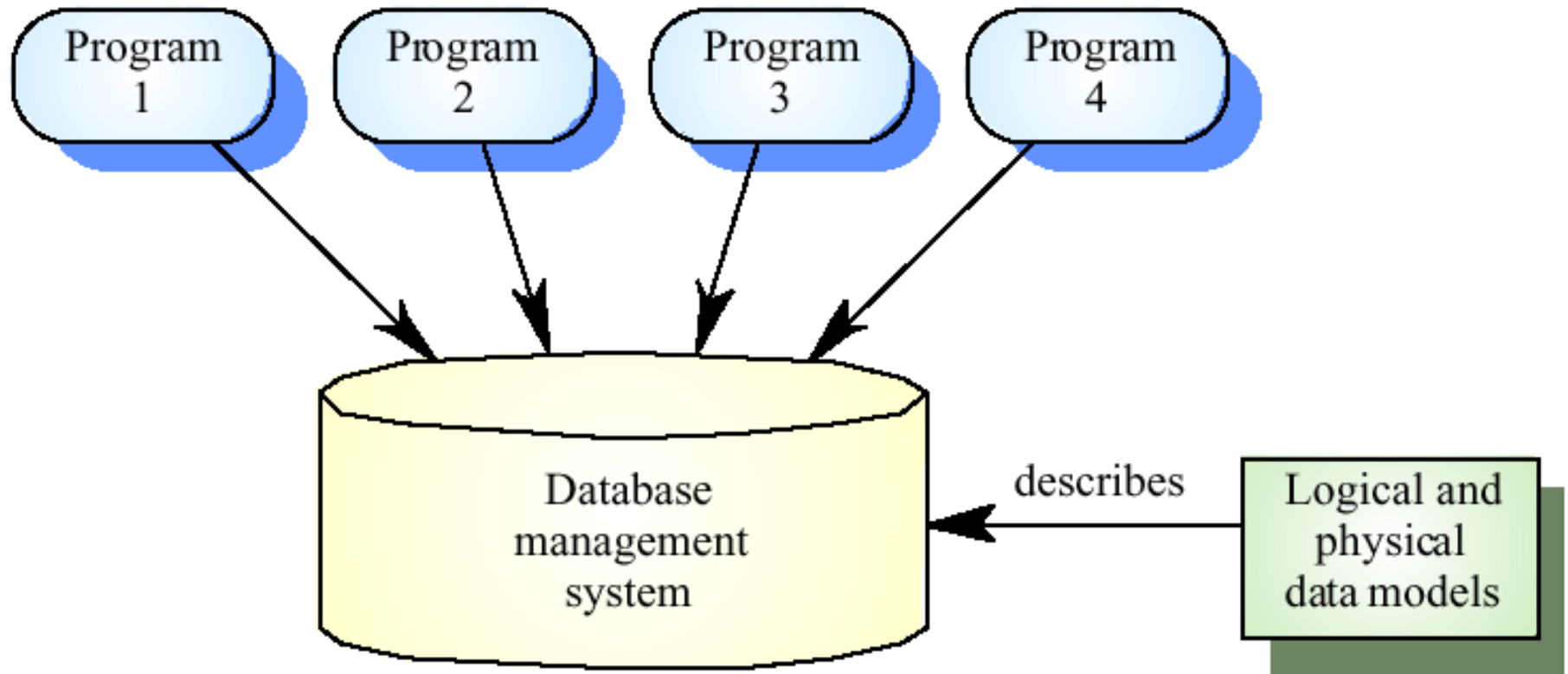


System  
change

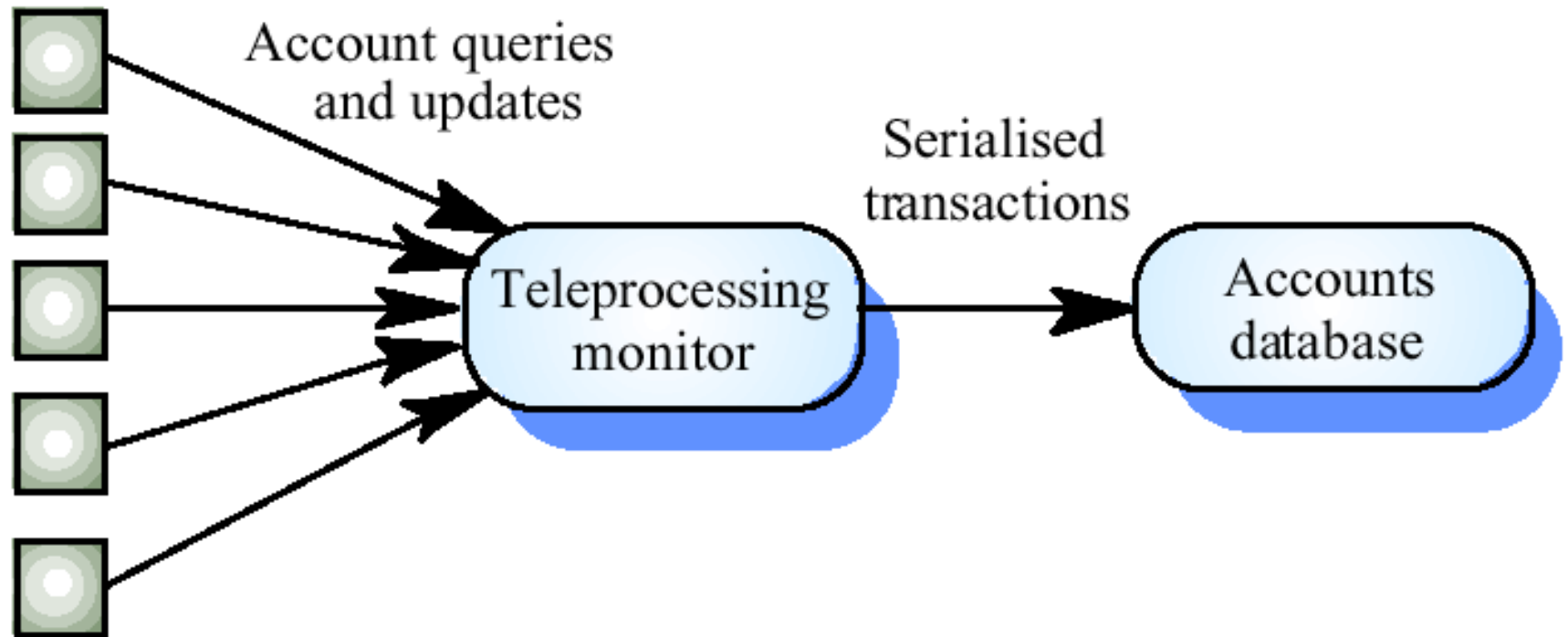
# Legacy application system



# Database-centred system



# Transaction processing



ATMs and terminals



# Legacy data

- The system may be file-based with incompatible files. The change required may be to move to a database-management system
- In legacy systems that use a DBMS the database management system may be obsolete and incompatible with other DBMSs used by the business
- The teleprocessing monitor may be designed for a particular DB and mainframe. Changing to a new DB may require a new TP monitor

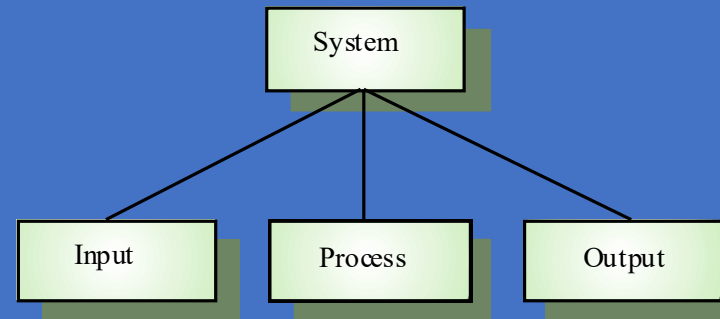
# Legacy system design

- Most legacy systems were designed before object-oriented development was used
- Rather than being organised as a set of interacting objects, these systems have been designed using a function-oriented design strategy
- Several methods and CASE tools are available to support function-oriented design and the approach is still used for many business applications


# Functional design process

- Data-flow design
  - Model the data processing in the system using data-flow diagrams
- Structural decomposition
  - Model how functions are decomposed to sub-functions using graphical structure charts

# Input- process- output model







# Input- process- output

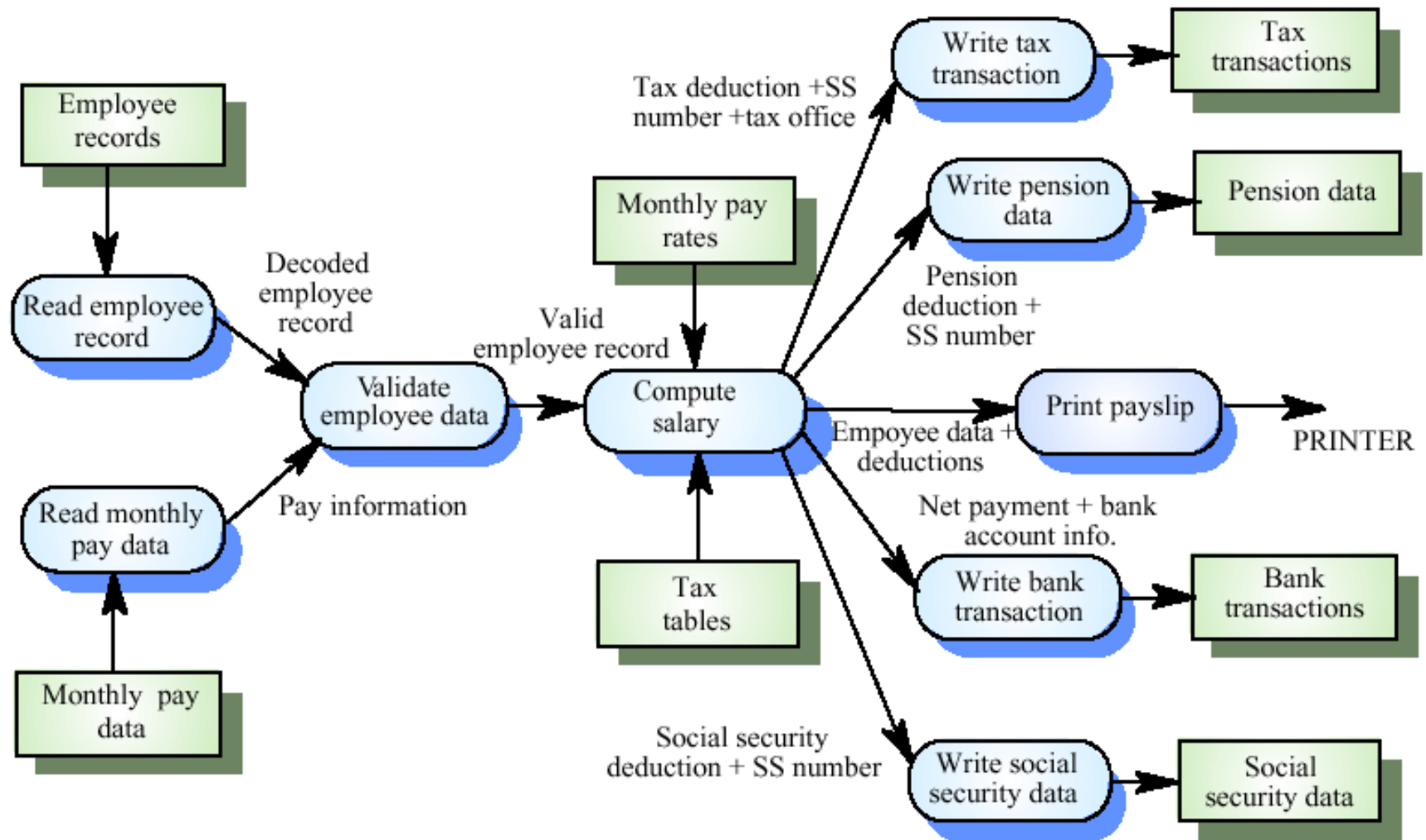
- Input components read and validate data from a terminal or file
- Processing components carry out some transformations on that data
- Output components format and print the results of the computation
- Input, process and output can all be represented as functions with data 'flowing' between them



# Data flow diagrams

- Show how an input data item is functionally transformed by a system into an output data item
- Are an integral part of many design methods and are supported by many CASE systems

# Payroll system DFD





# Payroll batch processing

- The functions on the left of the DFD are input functions
  - Read employee record, Read monthly pay data, Validate employee data
- The central function - Compute salary - carries out the processing
- The functions to the right are output functions
  - Write tax transaction, Write pension data, Print payslip, Write bank transaction, Write social security data

# Transaction processing

- A bank ATM system is an example of a transaction processing system
- Transactions are stateless in that they do not rely on the result of previous transactions. Therefore, a functional approach is a natural way to implement transaction processing

## INPUT

```
loop
  repeat
    Print_input_message (" Welcome - Please enter your card") ;
  until Card_input ;

  Account_number := Read_card ;
  Get_account_details (PIN, Account_balance, Cash_available) ;
```

## PROCESS

```
if Invalid_card (PIN) then
  Retain_card ;
  Print ("Card retained - please contact your bank") ;
else
  repeat
    Print_operation_select_message ;
    Button := Get_button ;
    case Get_button is
      when Cash_only =>
        Dispense_cash (Cash_available, Amount_dispensed) ;
      when Print_balance =>
        Print_customer_balance (Account_balance) ;
      when Statement =>
        Order_statement (Account_number) ;
      when Check_book =>
        Order_checkbook (Account_number) ;
    end case ;
    Print ("Press CONTINUE for more services or STOP to finish");
    Button := Get_button ;
  until Button = STOP ;
```

## OUTPUT

```
Eject_card ;
Print ("Please take your card ) ;
Update_account_information (Account_number, Amount_dispensed) ;

end loop ;
```

Design description of  
an ATM



# Using function- oriented design

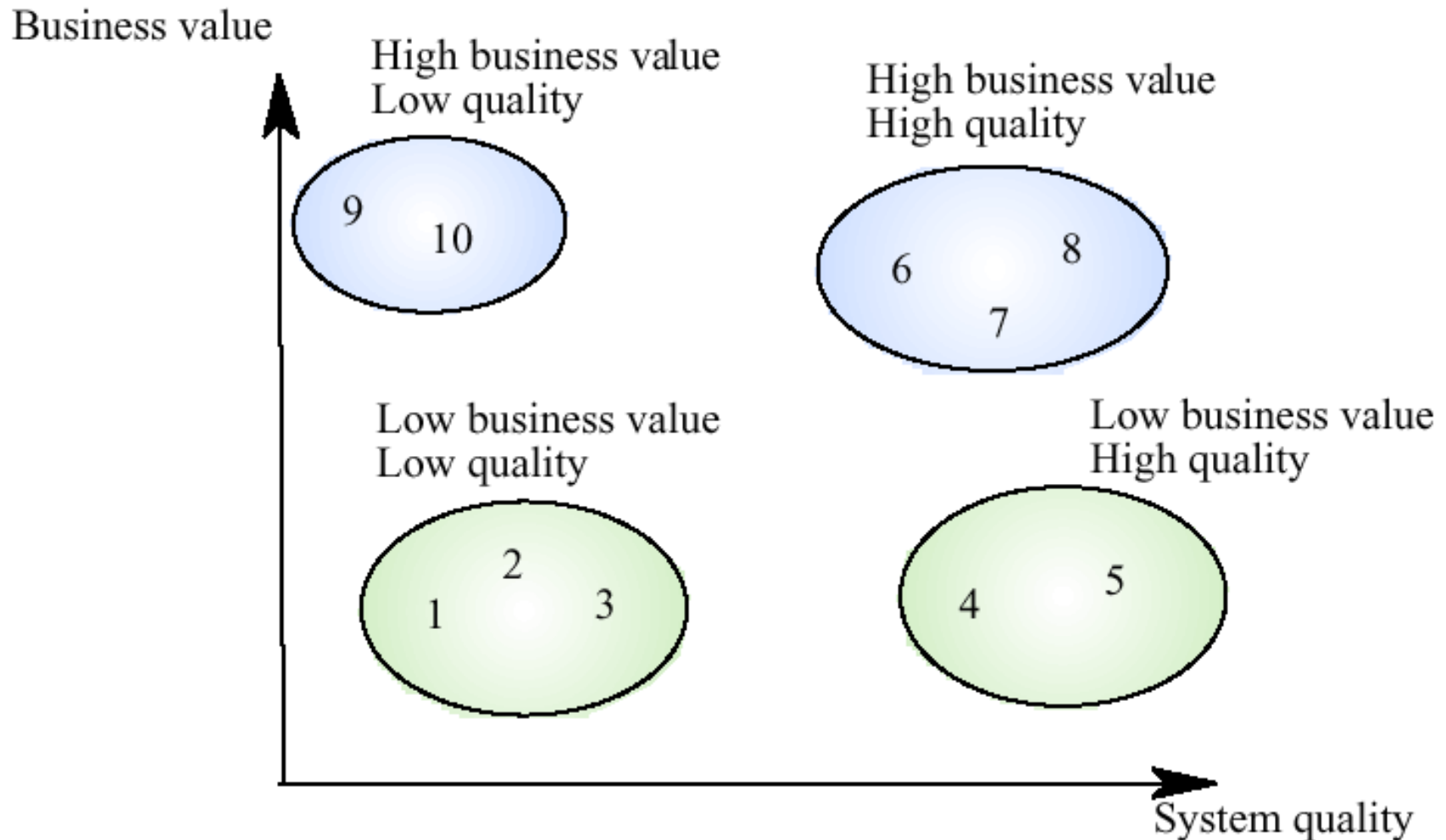
- For some classes of system, such as some transaction processing systems, a function-oriented approach may be a better approach to design than an object-oriented approach
- Companies may have invested in CASE tools and methods for function-oriented design and may not wish to incur the costs and risks of moving to an object-oriented approach

# Legacy system assessment

- Organisations that rely on legacy systems must choose a strategy for evolving these systems
  - Scrap the system completely and modify business processes so that it is no longer required
  - Continue maintaining the system
  - Transform the system by re-engineering to improve its maintainability
  - Replace the system with a new system
- The strategy chosen should depend on the system quality and its business value

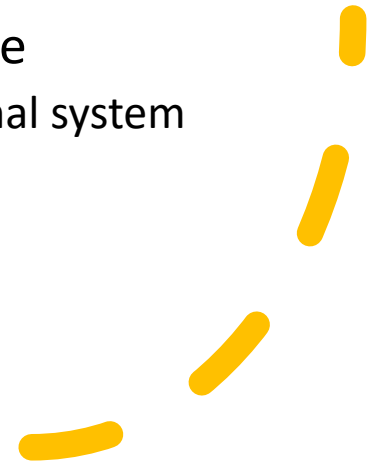


# System quality and business value



# Legacy system categories

- Low quality, low business value
  - These systems should be scrapped
- Low-quality, high-business value
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available
- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain
- High-quality, high business value
  - Continue in operation using normal system maintenance



# Business value assessment

- Assessment should take different viewpoints into account
  - System end-users
  - Business customers
  - Line managers
  - IT managers
  - Senior managers
- Interview different stakeholders and collate results



# System quality assessment

- Business process assessment
  - How well does the business process support the current goals of the business?
- Environment assessment
  - How effective is the system's environment and how expensive is it to maintain
- Application assessment
  - What is the quality of the application software system



# Business process assessment

- Use a viewpoint-oriented approach and seek answers from system stakeholders
  - Is there a defined process model and is it followed?
  - Do different parts of the organisation use different processes for the same function?
  - How has the process been adapted?
  - What are the relationships with other business processes and are these necessary?
  - Is the process effectively supported by the legacy application software?

# Environment assessment

<b>Factor</b>	<b>Questions</b>
Supplier stability	Is the supplier is still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, are the systems maintained by someone else?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to more modern systems.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers etc. be used with current versions of the operating system? Is hardware emulation required?

# Application assessment

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures which are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent and up-to-date?
Data	Is there an explicit data model for the system? To what extent is data duplicated in different files? Is the data used by the system up-to-date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there only a limited number of people who understand the system?

## System measurement

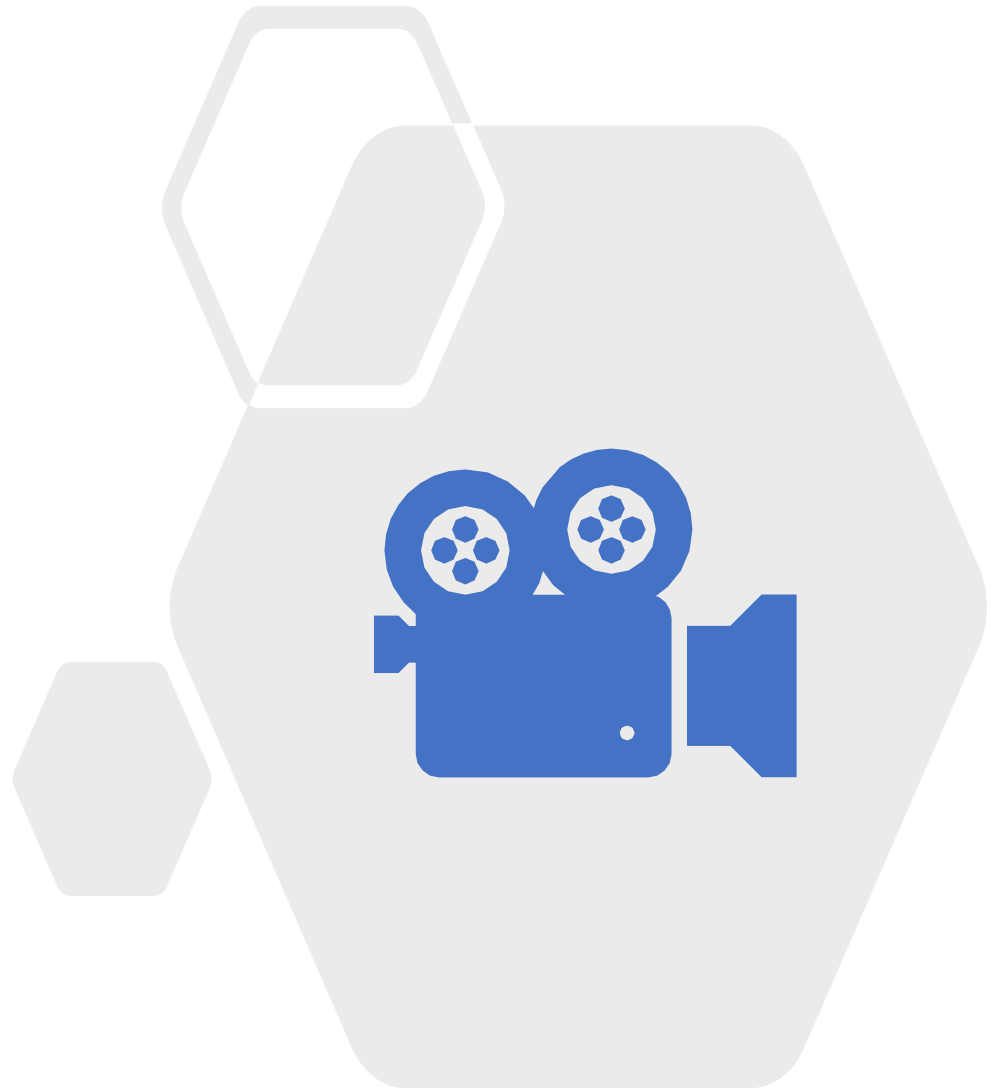
- You may collect quantitative data to make an assessment of the quality of the application system
  - The number of system change requests
  - The number of different user interfaces used by the system
  - The volume of data used by the system



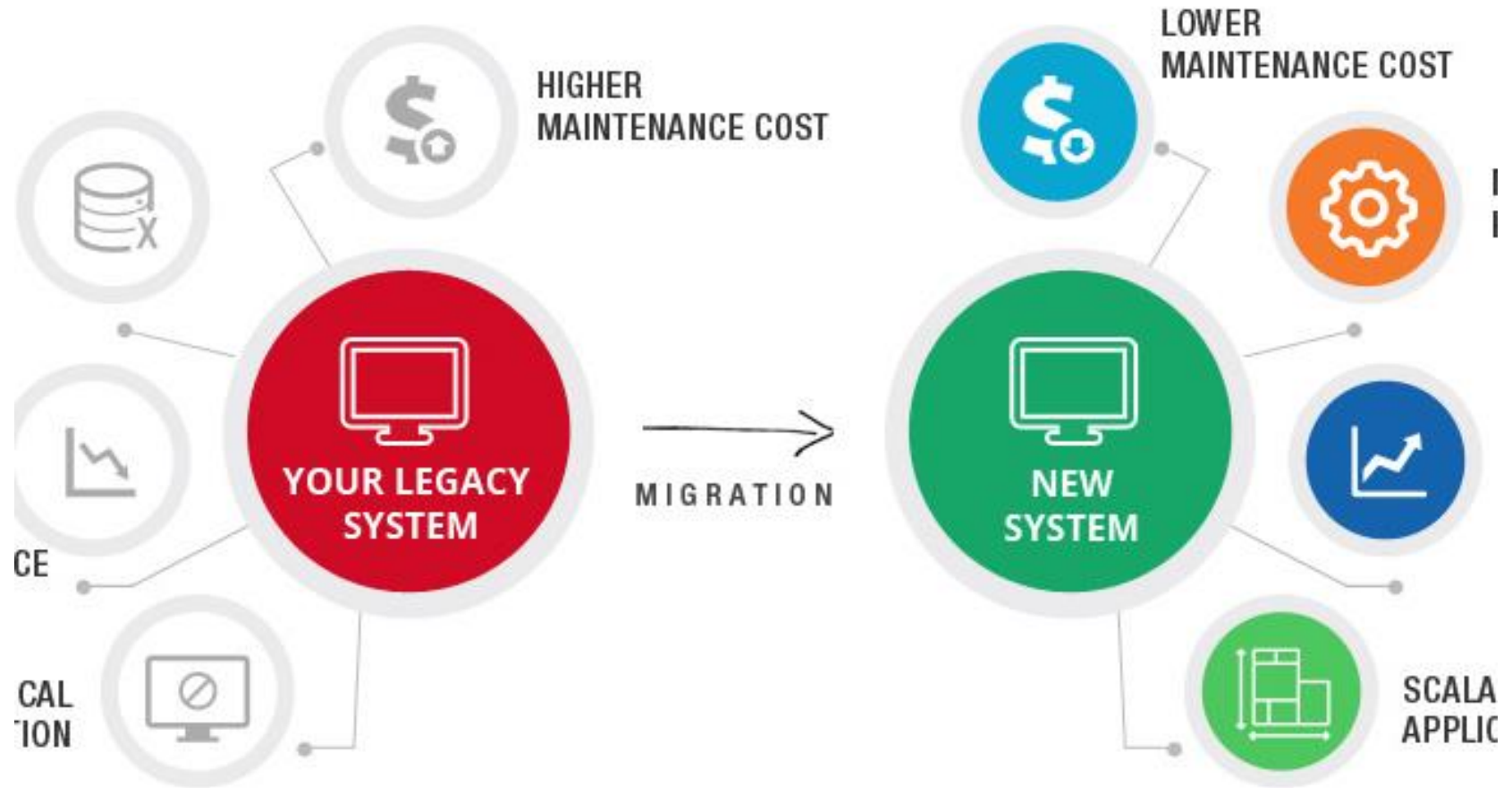
# Key points

- A legacy system is an old system that still provides essential business services
- Legacy systems are not just application software but also include business processes, support software and hardware
- Most legacy systems are made up of several different programs and shared data
- A function-oriented approach has been used in the design of most legacy systems

# Video



# Application Migration Solution



# Why Migration of Legacy Systems is required?

- Very **High Cost of Maintenance**
- Legacy systems are **old** and **complex** to understand
- Limited **availability of SMEs** and skilled resources
- Legacy systems are **non-user friendly** and provide **limited opportunities** to improve the business value
- Integration or compatibility with newer systems or technologies is **very difficult and time consuming**

# Reasons and objectives for System Migration (1/3)

## General Migration Reasons:

Adapt	Adapt to e-commerce platform
Adapt	Adapt to web technology
Reduce	Reduce time to market
Support	Support new business rules
Allow	Allow customizable billing
Adapt	Adapt to evolving tax laws
Reengineer	Reengineer business processes

# Reasons and objectives for System Migration (2/3)

## Software Migration Reasons:

Higher	Higher productivity
Lower	Lower maintenance costs
Move	Move to object-oriented platforms
Inject	Inject component technology
Adapt	Adapt to modern data exchange technology
Leverage	Leverage modern methods and tools

# Reasons and objectives for System Migration (3/3)

## SW Architecture Migration Reasons:

Move	Move to network-centric platforms
Integrate	Integrate cooperative information systems
Leverage	Leverage centralized repositories
Move	Move from hierarchical to relational db
Take	Take advantage of web user interfaces
Provide	Provide interoperability via buses and gateways among applications
Move	Move to client-server architectures

# Migration

Transformation of an Legacy Systems is complex, challenging and critical IT initiative

Transformation / migration of legacy system tend to involve high level of risk

Need to understand various challenges faced during testing of migrated system and how these challenges are overcome



# Common Requirements For Migration

- Ensure continuous, safe, reliable, robust, ready access to mission-critical functions and information
  - Migrate in place
- Minimize migration risk
  - Reduce migration complexity
  - Make as few changes as possible in both code & data
  - Alter the legacy code to facilitate and ease migration
  - Concentrate on the most important current and future requirements
- Minimize impact on
  - users
  - applications
  - databases
  - operation
- Maximize benefits of modern technology
  - user interfaces, dbs, middleware, COTS
  - automation, tools



# Data Intensive Systems: Merger of Two Banks



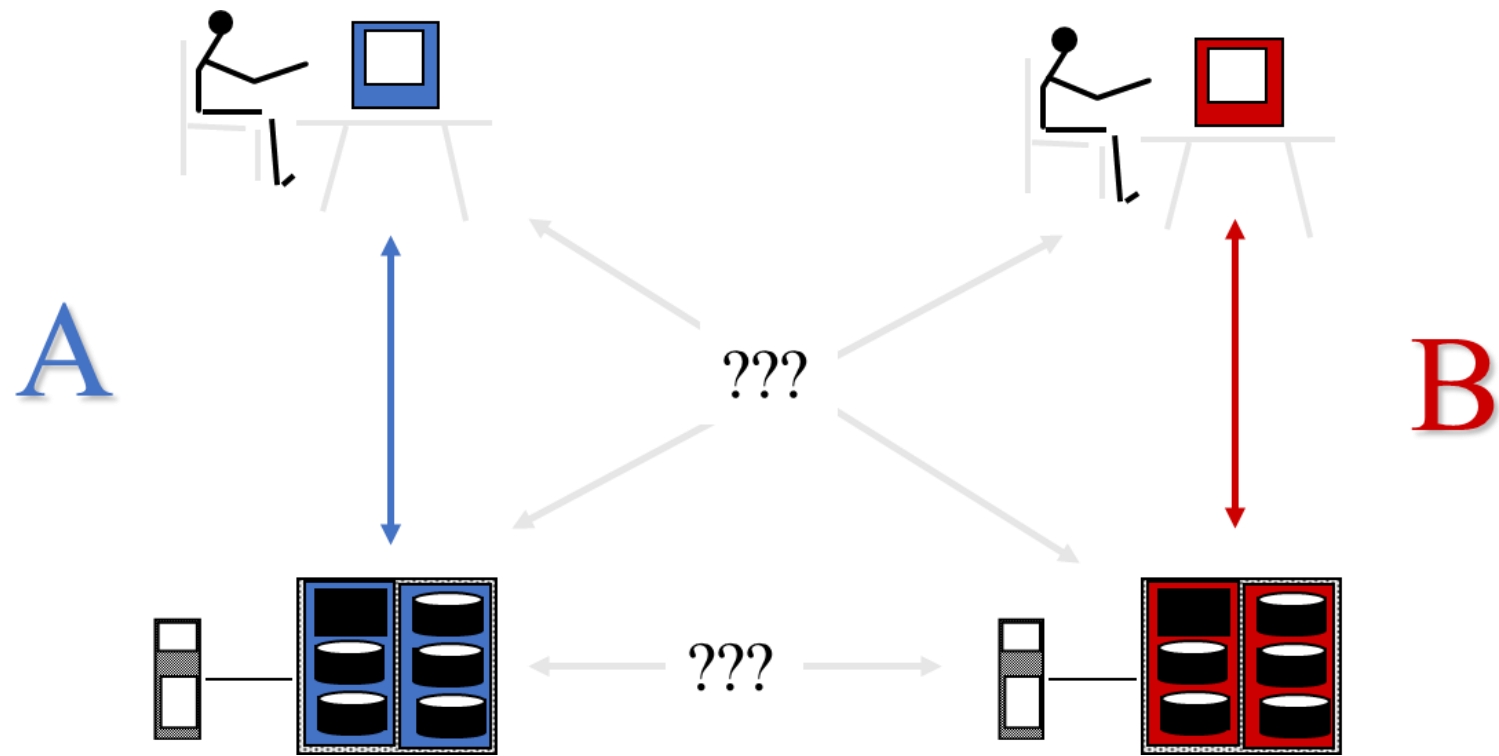
Each bank has a **database** with its customer accounts. The databases are used by staff at many branches and for back-office processing. These systems are examples of Repository Architectural Style.



The requirement is to **integrate the two banks** so that they appear to the customers to be a **single organization** and to provide integrated service from all branches.

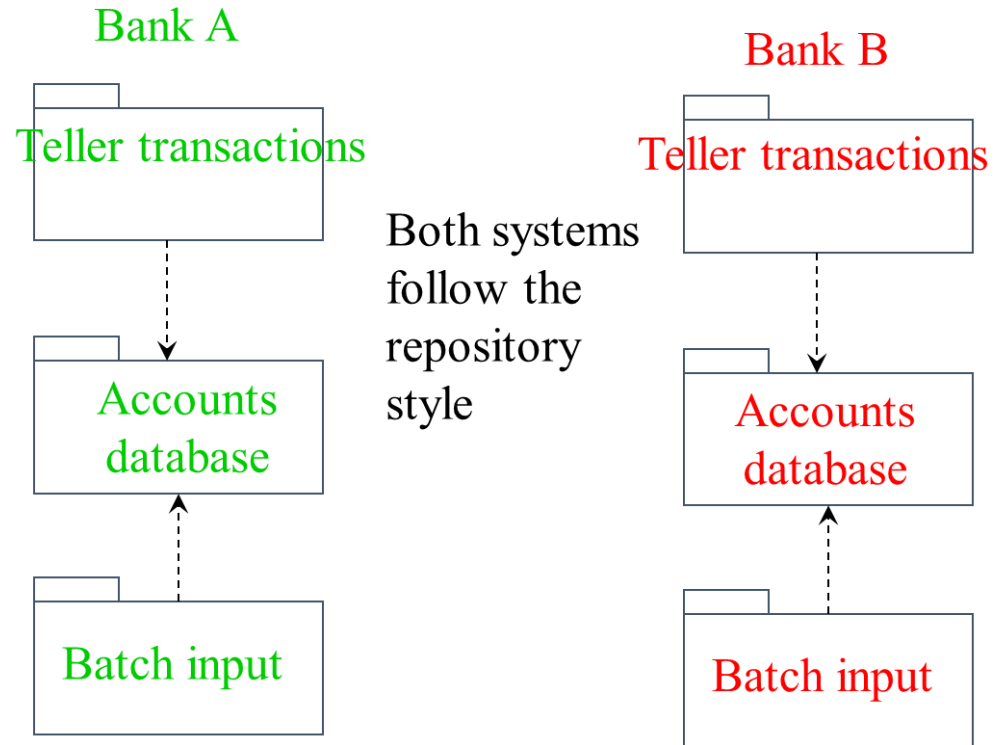


*This is an example of working with legacy systems.*



Merger of Two Banks: Options

# Merger of Two Banks: Given is the follow setting



# Merger of Two Banks: Architectural Options

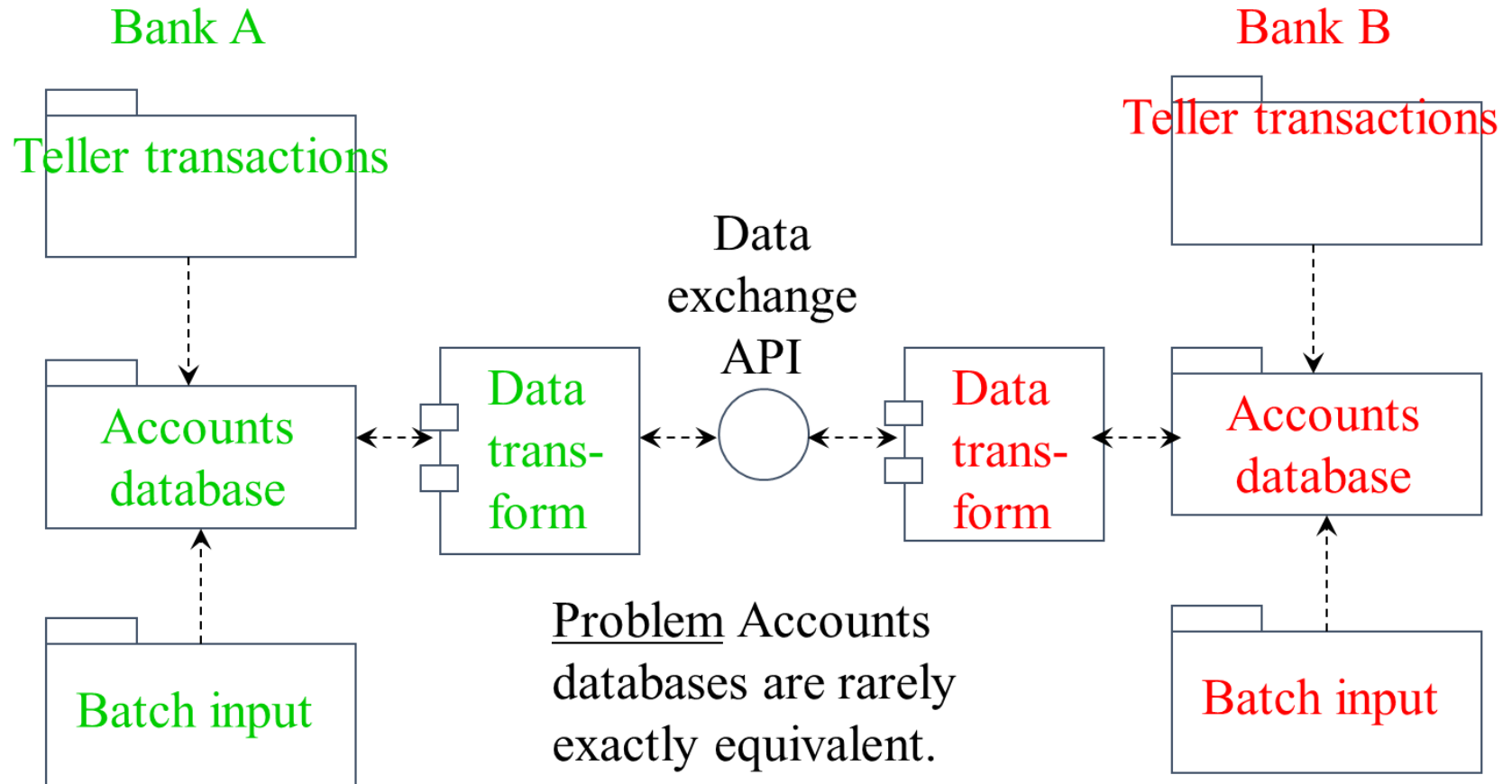
## **Convert everything to System A**

- convert databases
- retrain staff
- enhance System A (software and hardware)
- discard System B

## **Build an interface between the databases in System A and System B**

## **Extend client software so that it can interact with either System A or System B database**

# Merger of Two Banks: Interface between the Databases



# Common Requirements For Migration

## Ensure

Ensure continuous, safe, reliable, robust, ready access to mission-critical functions and information

- Migrate in place

## Minimize

Minimize migration risk

- Reduce migration complexity
- Make as few changes as possible in both code & data
- Alter the legacy code to facilitate and ease migration
- Concentrate on the most important current and future requirements

## Minimize

Minimize impact on

- users
- applications
- databases
- operation

## Maximize

Maximize benefits of modern technology

- user interfaces, dbs, middleware, COTS
- automation, tools

# Classification of Legacy Information System Architectures

- **Decomposable**

- Separation of concerns
- Interfaces, applications, db services are distinct components
- Functional decomposition
- Ideal for migration





# Classification of Legacy Information System Architectures



- **Semidecomposable**
  - Applications and db services are not readily separable
  - System is not easily decomposable

## An abstract composition of various geometric shapes on a white background. In the top left, a green right-angled triangle is partially visible. To its right is a blue semi-circle. Below the green triangle is a blue circle. In the center is a large orange semi-circle. To the right of the orange semi-circle are two vertical yellow bars. In the bottom left is a large orange circle. Above it are two yellow curved bars. To the right of the large orange circle are three yellow curved bars of different orientations. In the bottom right is a green square frame.

- No functional components are separable
- Users directly interact with individual modules

There is nothing more difficult to arrange, more doubtful of success, and more dangerous to carry through than initiating changes.

—N. Machiavelli

# What are some of the Migration Strategies

- Ignore
  - retire, phase out, let fail
- Replace with COTS applications
- Cold turkey
  - rewrite from scratch
  - high risk
- Integrate and access in place
  - integrate future apps into legacy apps without modifying legacy apps

# Gradual Migration or “Chicken Little”

## **Migration Objective**

- Re-architect and transition the applications incrementally
- Replace LIS with target application
- Language migration
- Schema and data migration
- User interface migration

# Gradual Migration or “Chicken Little”

## **Intend:**

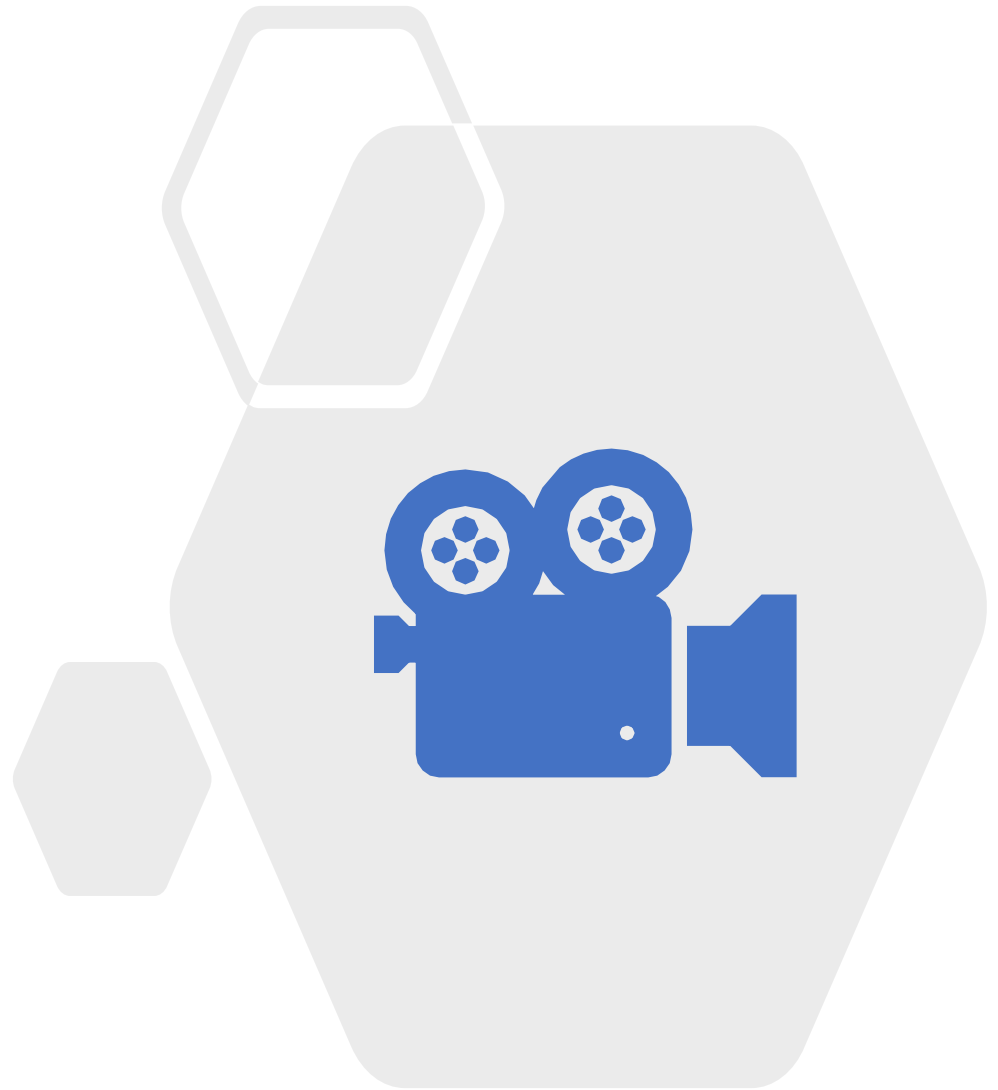
- The intent **is to phase out legacy applications over time**
- In place access is not economical in the long run
- More effective, less risky than cold turkey
- Allows for independent user interface and database evolution
- Incremental

# Gradual Migration or “Chicken Little”

## However:

- Legacy and target **applications must coexist during migration**
- A gateway to isolate the migration steps so that the end users do not know if the info needed is being retrieved from the legacy or target system
- Development of gateways is difficult and costly

# Video



# Types of Migration Testing

- ☐ Functional Equivalence Testing
- ☐ As-Is Migration Testing for User Interface
- ☐ Web Service Migration Testing
- ☐ Authorization and Authentication Testing
- ☐ System Integration Testing
- ☐ Performance Testing



# Types of Migration Testing

- ❑ Functional Equivalence Testing
- ❑ As-Is Migration Testing for User Interface
- ❑ Web Service Migration Testing
- ❑ Authorization and Authentication
- ❑ System Integration Testing
- ❑ Performance Testing

*Migration testing is primarily the functional equivalence testing along with the necessary non-functional testing to validate the non-functional requirements*

# Challenges faced during Migration Testing

- Very **limited** and **poor knowledge** of legacy systems
- **Lack of functional, technical documentation and training material**
- **Non-availability** of functional experts, Subject Matter Experts (SMEs) and Business Users
- **Legacy systems** are not user-friendly and they are difficult to understand

## Challenges faced during Migration Testing (continued)

- **Non-availability of complete test scripts** including regression scenarios
- Development **migration compatibility** issues of legacy system results increase **development time** and effort which in-turn reduce testing time frame
- Delay during development phase due to unforeseen **technical issues** and **dependencies** on various external tools and software used

# Strategy to Overcome Migration Testing Challenges

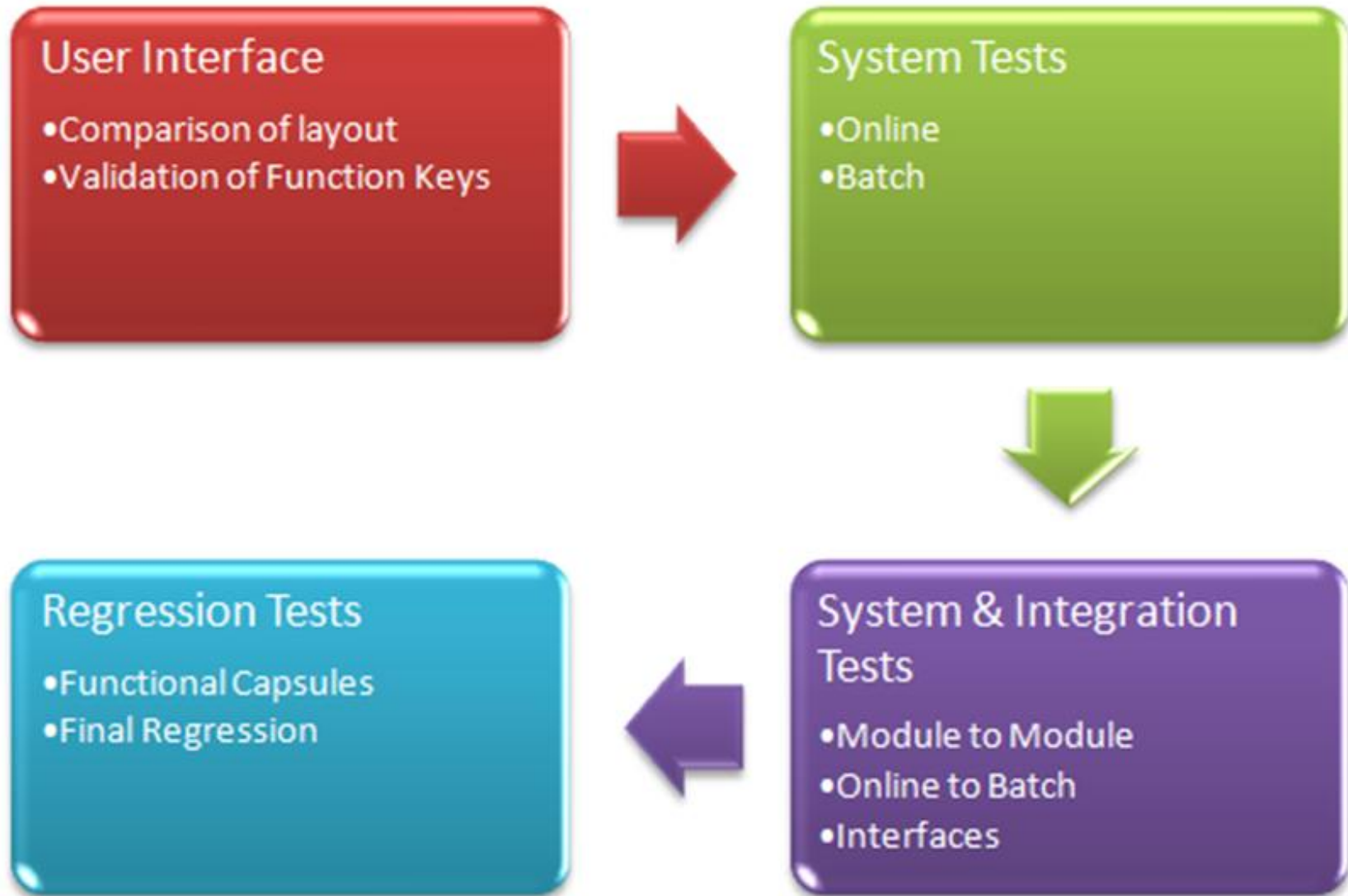
- ❑ **Analysis of existing regression packs** to understand core functionality of the legacy system
- ❑ **Code analysis** by technical experts
- ❑ Creation of technical as well as **functional documentation** based on the analysis performed
- ❑ Using **available test cases of recent projects / enhancements** as a reference for migration test scripts

# Strategy to Overcome Migration Testing Challenges

Continued...

- ❑ Perform **testing in phased manner** (i.e. in smaller chunks) to reduce waiting time
- ❑ Identify **cross functional dependencies** within the system so that testing of multiple functionalities can be performed simultaneously

# Functional Equivalence Testing - Workflow



# Value Additions of Application Migration Testing

- ✓ Functional and **business processes continuity** maintained
- ✓ **Performance improvement** of around 5 to 7 percent
- ✓ Existing **defects in Legacy System** were identified and fixed
- ✓ **Functional documentations**, technical documentations, test packs and test results are now available
- ✓ Risk of **non-availability of skilled test resources** is eliminated

# Data Migration



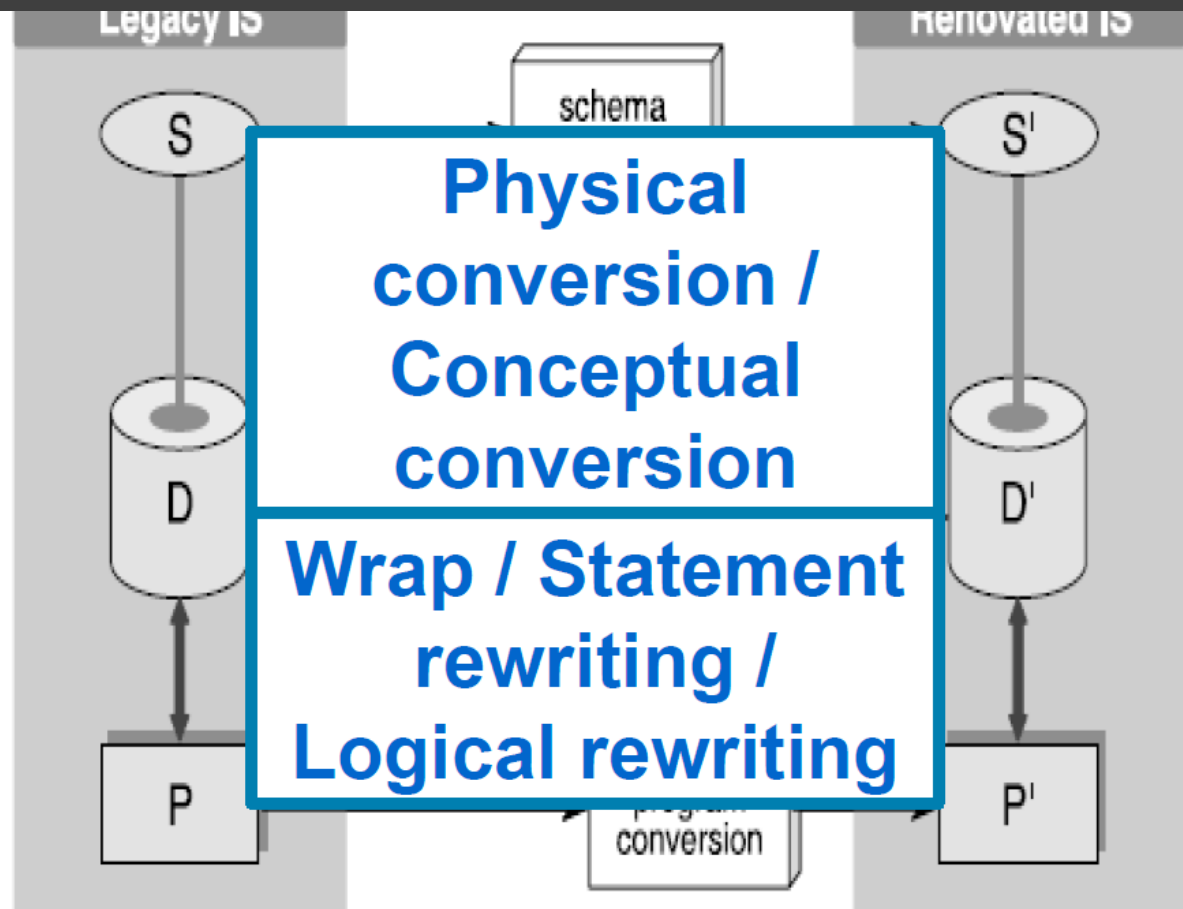


# Databases

- **Central for information systems**
- **Contain major company assets: data**
- **Often developed using outdated technology**
- **Migration should**
  - **Preserve the data**
  - **Improve the technology used**
    - **Flexibility**
    - **Availability of skills**

# Database migration

- **S – DB schema**
- **D – DB data**
- **P – data manipulation programs**



# Physical Migration

## SQL

## COBOL

DATA DIVISION.

FILE SECTION.

FD PERSON-FILE

DATA RECORD IS **PERSON-ITEM**.

01 PERSON-ITEM.

02 PERSON-KEY.

03 PERSON-ID PICTURE X(4).

02 PERSON-NAME PICTURE X(20).

02 PERSON-ADDRESS PICTURE X(20).

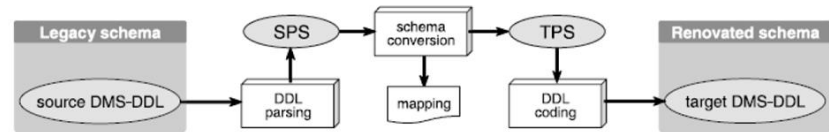
02 PERSON-CITY PICTURE X(18).

```
CREATE TABLE PERSON-ITEM  
(PERSON-ID varchar(4) PRIMARY KEY,  
PERSON-NAME varchar(20),  
PERSON-ADDRESS varchar(20),  
PERSON-CITY varchar(18))
```

**Advantages and  
disadvantages of  
physical conversion?**

# Physical migration: Translation vs. Improvement

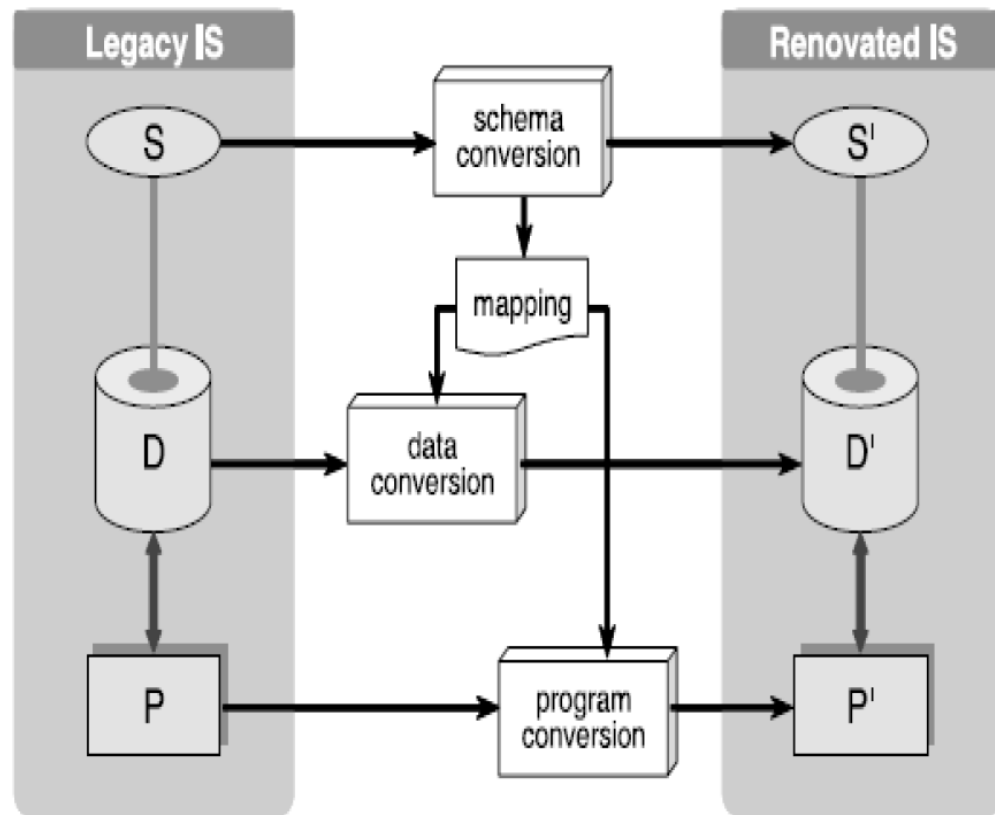
7/14/2021



- **Easy to automate**
  - Existing work: COBOL  $\Rightarrow$  relational, hierarchical  $\Rightarrow$  relational, relational  $\Rightarrow$  OO
- **“Migration as translation” vs “migration as improvement”**
- **Semantics is ignored**
  - Limitations of COBOL  $\Rightarrow$  Design decisions in the legacy system  $\Rightarrow$  Automatic conversion  $\Rightarrow$  the same design decisions in the new system
    - Risk: compromised flexibility

# Recall...

- So far we have considered DB schemas only
- Next step: data migration



- **Strategy depends on the schema migration strategy**
- **Physical conversion: straightforward**
  - **Data format conversion**
- **Conceptual conversion**
  - **Data may violate implicit constraints**
  - **Hence, data cleaning is required as preprocessing**
  - **Once the data has been cleaned up: akin to physical conversion**

## **Analyse:**

- **Define inconsistencies and detect them**

**Define individual transformations and the workflow**

**Verify correctness and effectiveness**

- **Sample/copy of the data**

**Transform**

**Backflow if needed**

- **If the “old” data still will be used, it can benefit from the improvements.**

How to clean up data

- **If inconsistency has been detected, the offending instances**
  - **Are removed**
  - **Are modified so the offending data becomes NULL**
  - **Are modified by following user-defined preferences**
    - **One table might be more reliable than the other**
    - **One attribute may be more reliable than the other**
  - **Are modified to reduce the (total) number of modifications required to restore consistency**

Inconsistency  
Management