Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

Scenario

List of author names goes here

Abstract

Need some instructions here.

APM.1064.00.01 **Draft** 20 October 1993

Briefing Note

Distribution:
Supersedes:
Superseded by:

Managing the Applications

E. Jane Cameron

Director, Network Systems Specification Research
Bellcore

Today's Business Climate

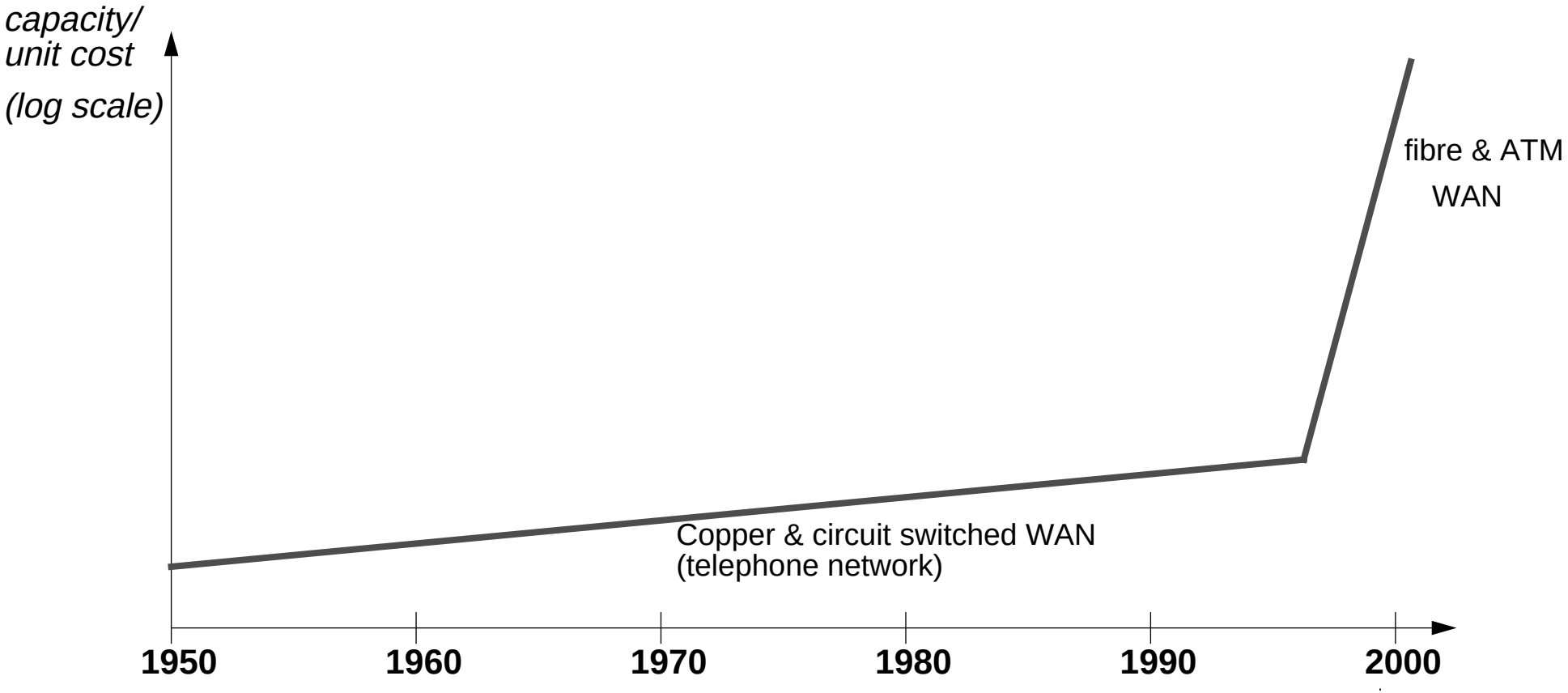
Every Business faces:

- Customers who expect high quality customized/ personalized services and products
- Short time-to-market, Short market-window
- Heavy competition

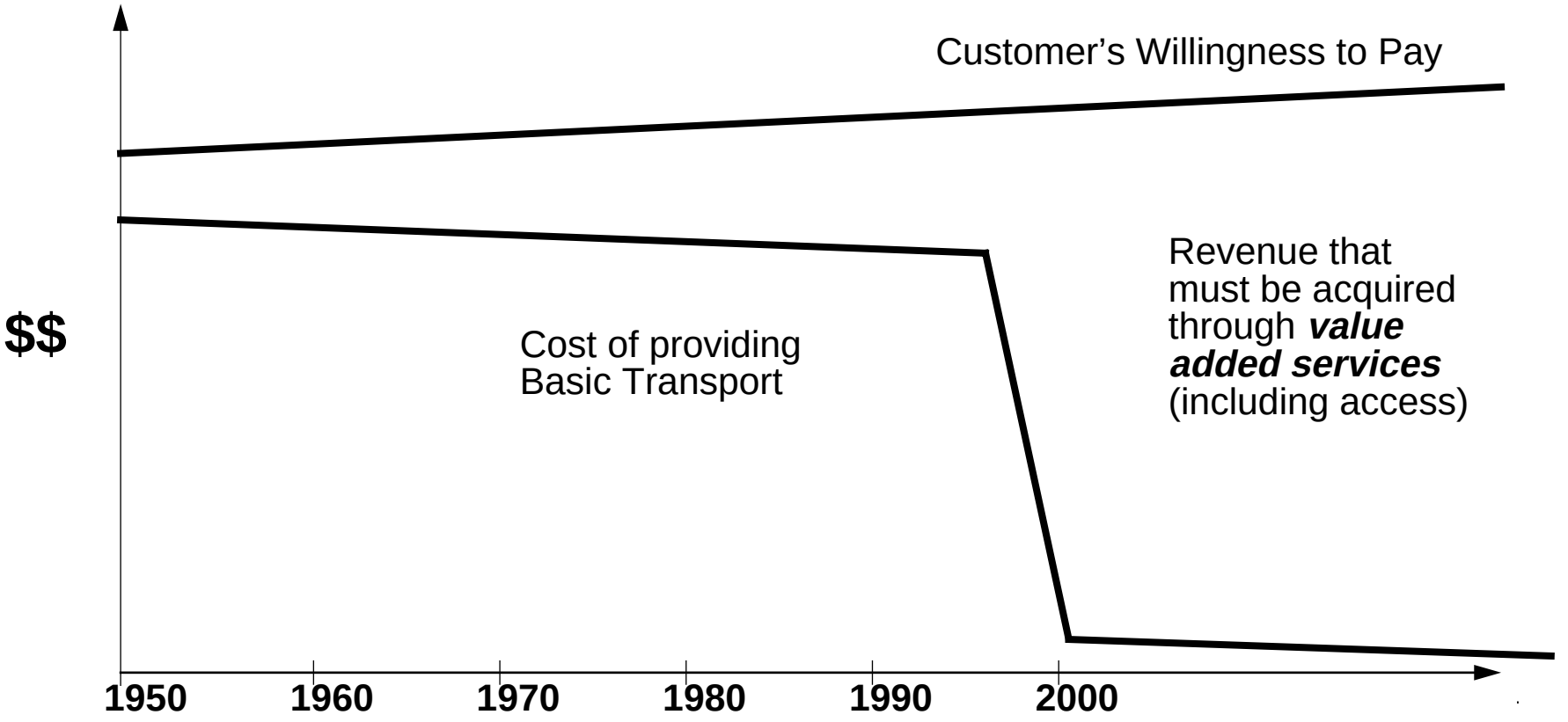
Every Business is increasing

- Its reliance on its computer/network infrastructure (including telecommunications)

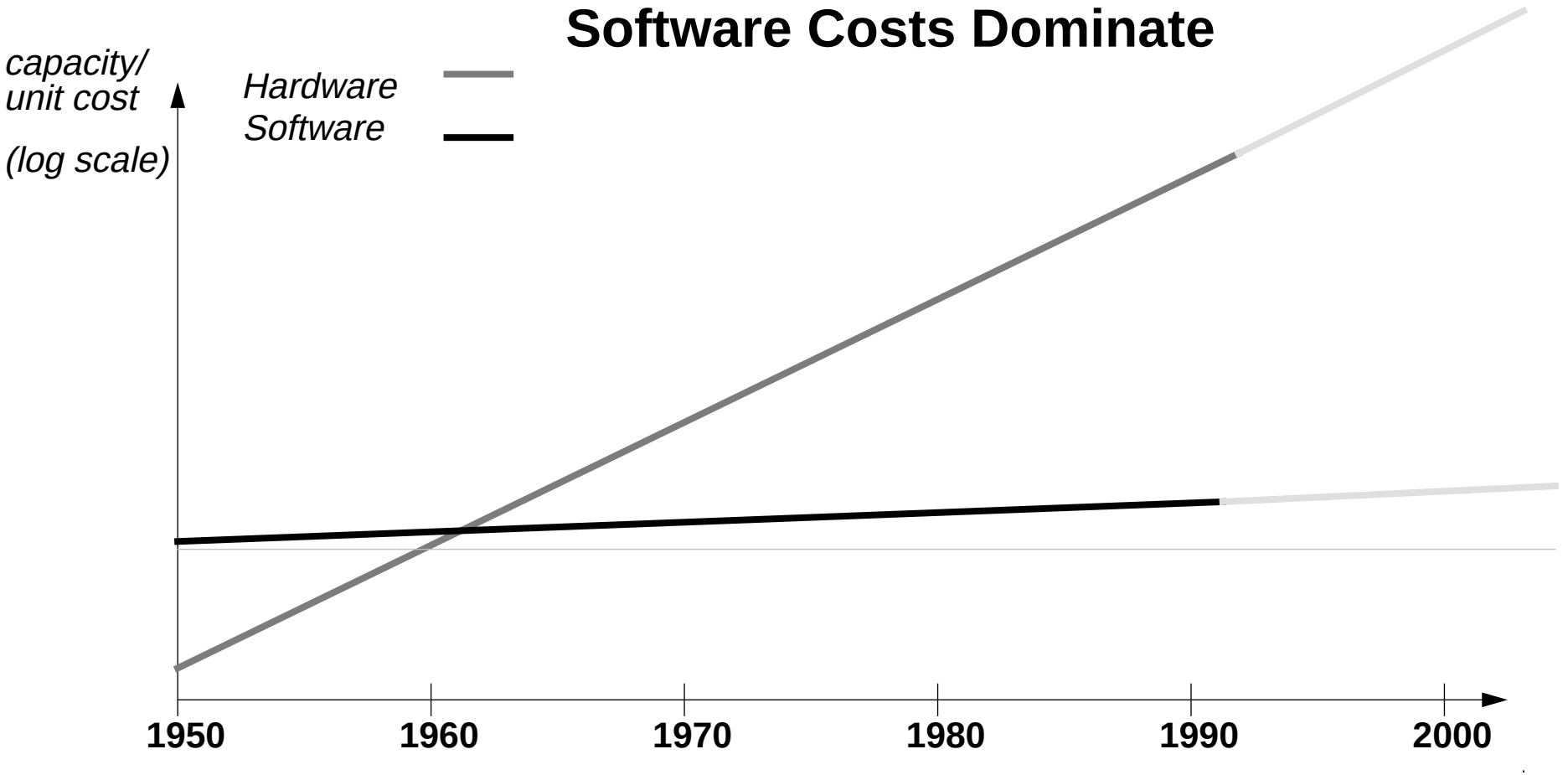
The Communications Revolution



The Telecoms Business Challenge



Software Costs Dominate



“Building distributed systems is like giving a enema to a tiger in a phone box -- it’s messy and you are likely to end up dead!”

---Jim Horning

Reduce the Software Burden

Improve time-to-market

- Rapid design
- Rapid code and test
- Rapid installation

Avoid re-implementation

- Build on existing systems
- Build a better legacy

Increase interoperability

- Freedom to choose technology
- Freedom to mix new and existing technologies
- Freedom to couple technologies

Value Added Service Creation at Bellcore

Programme of Network Systems Research

Value Added Services:

Multi-media Services -- Desktop Conferencing
Mobile Users

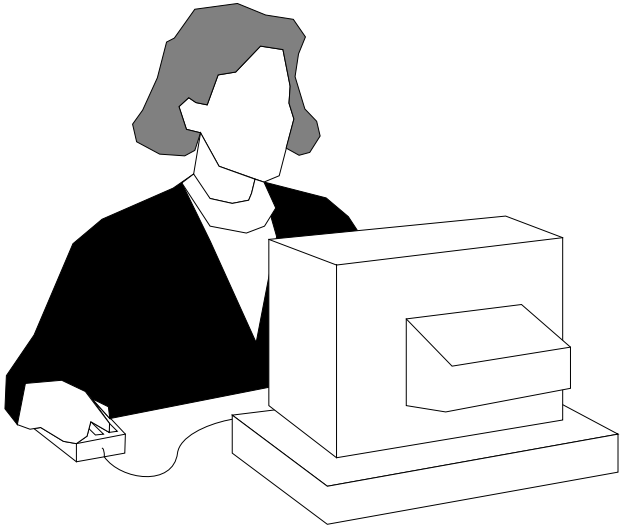
Service Creation and Management:

Feature Interaction Management
Data-Cycle Architecture

In each case we found technology was only a small part of the problem...

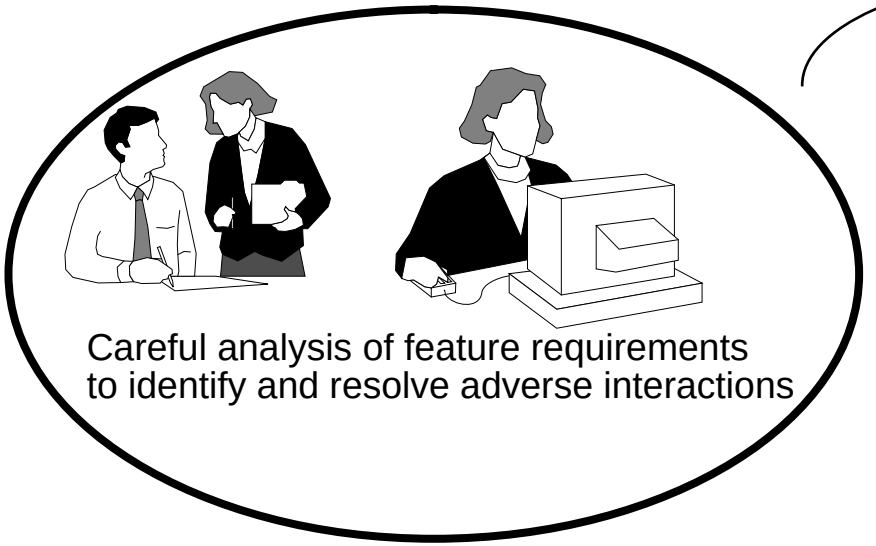
ANSA

Feature Interactions: Requirements

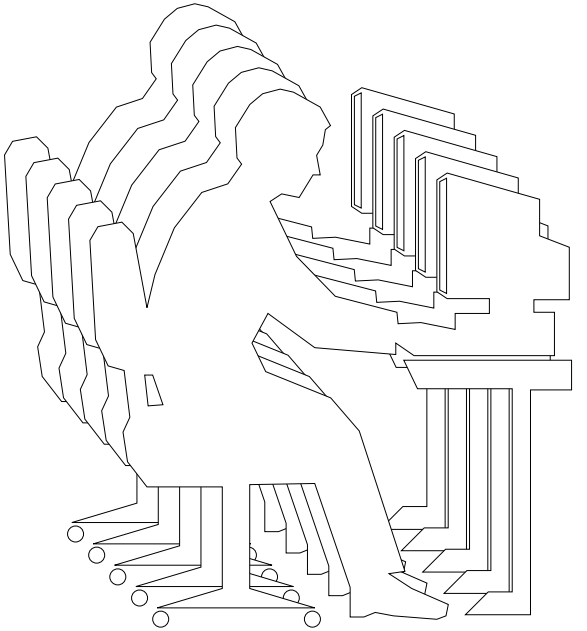


Careful analysis of feature requirements
to identify and resolve adverse interactions

Feature Interactions: Implementation

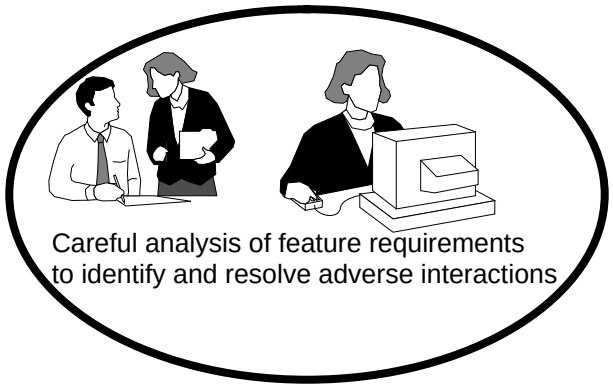


Careful analysis of feature requirements
to identify and resolve adverse interactions

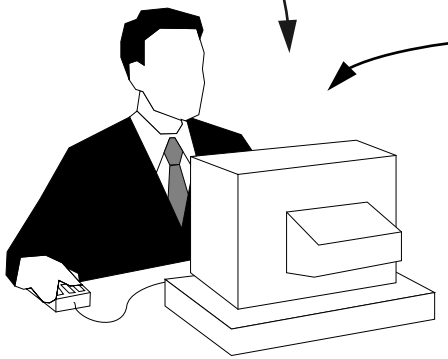


Adding the feature functionality to either a switching system or an IN component. (Some reprogramming of the switching system is required).

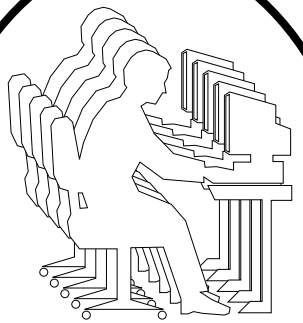
Feature Interactions: Testing



Careful analysis of feature requirements to identify and resolve adverse interactions

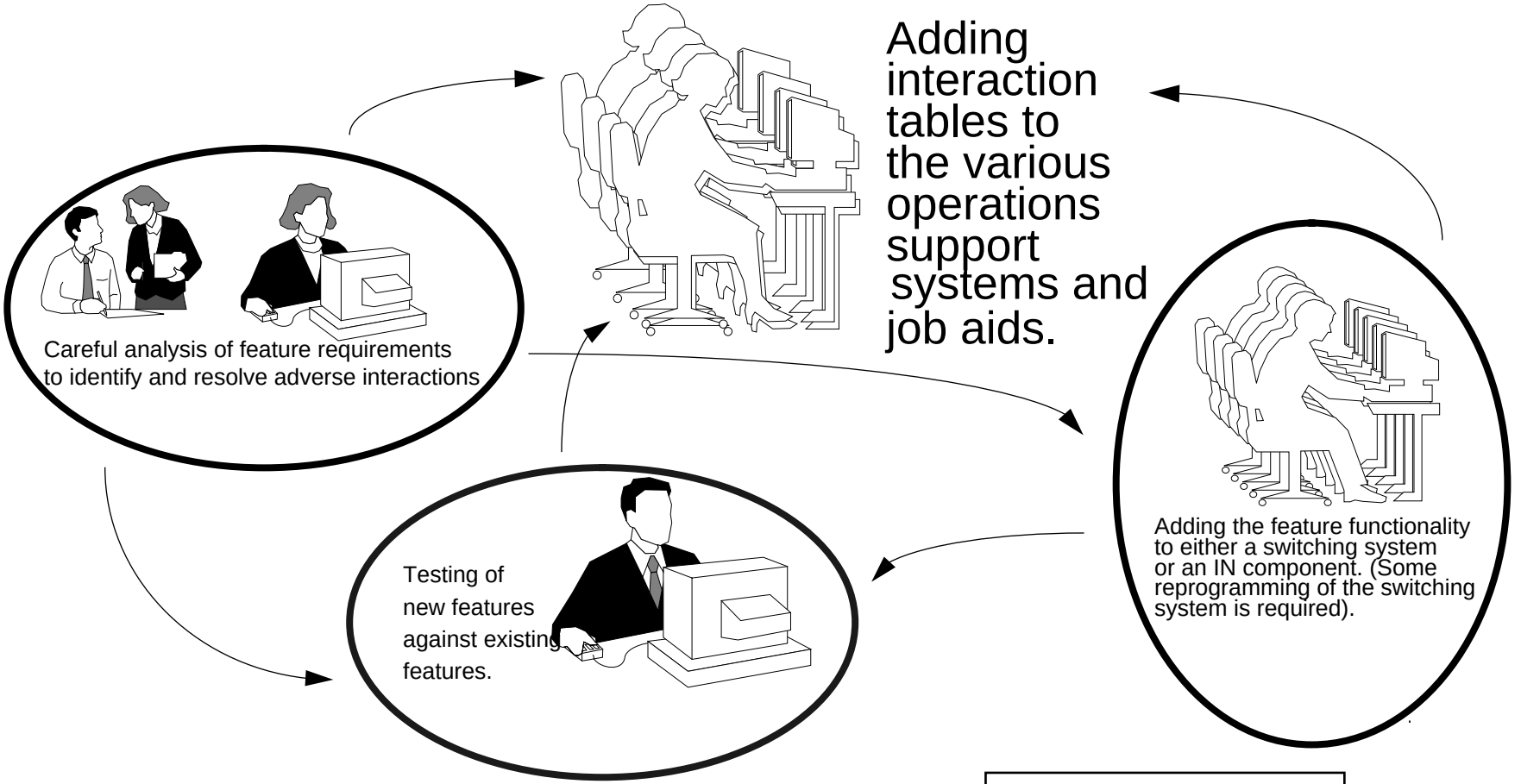


Testing of new features against existing features.

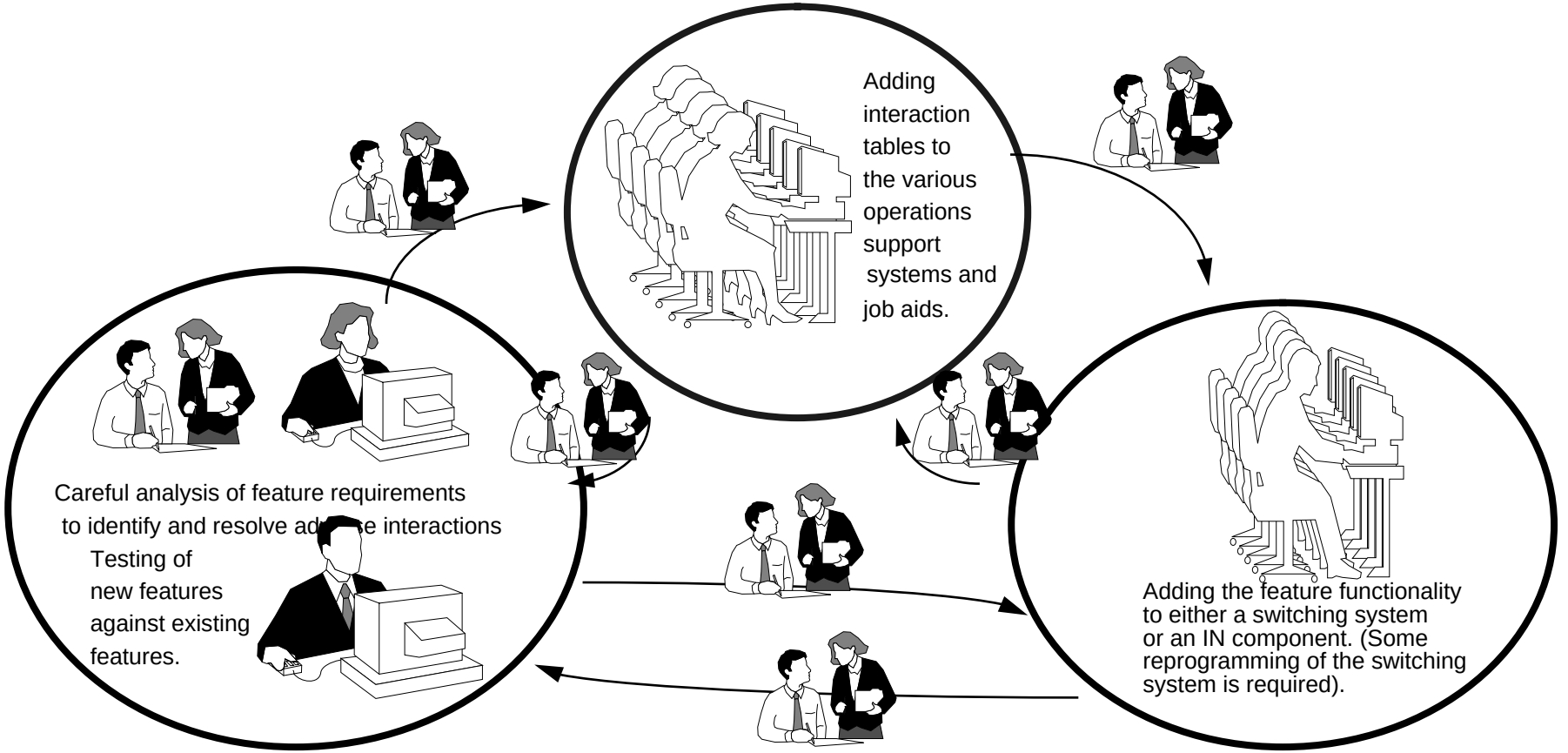


Adding the feature functionality to either a switching system or an IN component. (Some reprogramming of the switching system is required).

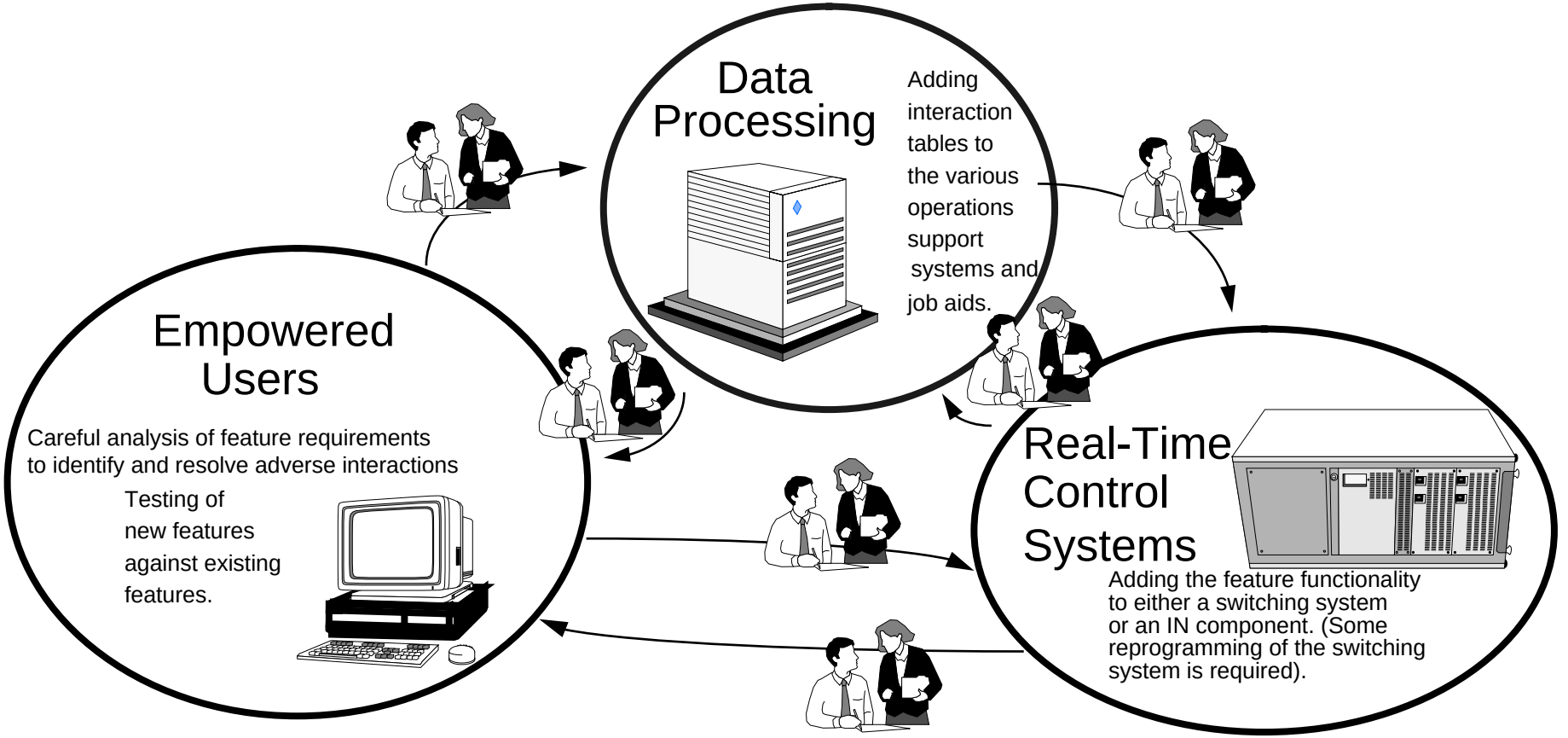
Feature Interactions: Deployment



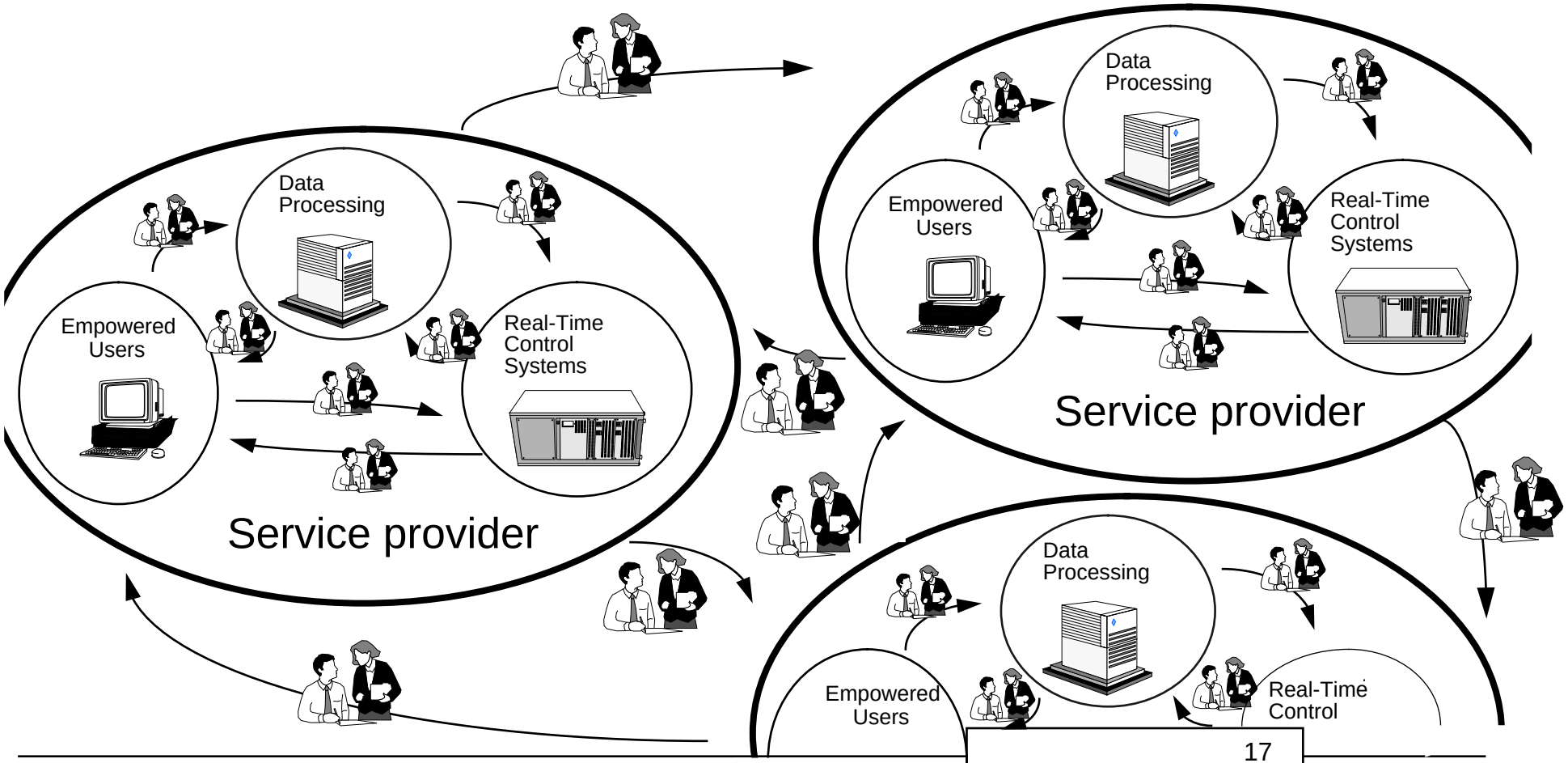
Feature Interactions: the Process



Feature Interactions: the Technology



Feature Interactions: Multiple Domains



Managing Feature Interactions

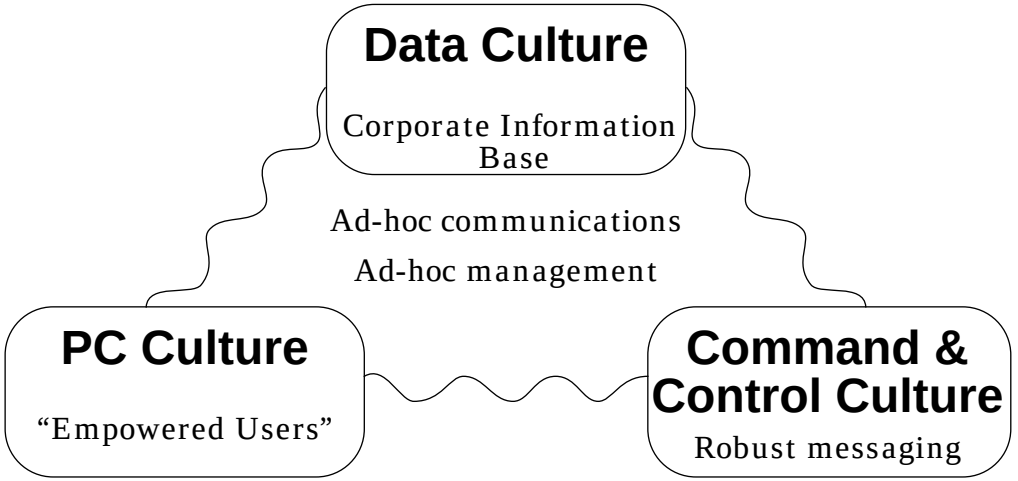
A piece of a much larger problem ---

SERVICE MANAGEMENT

across federated, networked islands!

Distributed Computing Cultures

- **Three important and quite separate distributed computing cultures exist**



- **Each has its own techniques for distribution and application integration**
- **They meet different needs - no single one will absorb the others**
- **Legacy of existing technology choices and applications will persist**
- **Current focus on Data <-> PC axis is too narrow**

Testimonials

Provisioning and managing services across federated networked islands is a common challenge.

Siemens

ANSA based management overlay for OSI MAP protocol based manufacturing system.

NASA

ANSA based data retrieval and remote processing overlay for the Internet.

GESI

ANSA based document transfer overlay for PC-based hospital laboratory system.

A Software Architecture for Service Management

The various software systems used by an organization, collectively function as a system, whether or not they were designed as such or are managed as such.

Ideally a Software Architecture should facilitate the coordination and evolution of software systems, which support business processes.

Software Architecture Assessment

In the *Which?* Report (Consumer Reports) on Software Architectures:

What would the main categories be?

What would be the check list for each category?

What would the criteria for evaluation be?

What would be the rating for current approaches? (OSF DCE, OMG CORBA, INA, etc.)

Software is about Organization and Structure

A Sketchpad drawing is entirely different from the trail of carbon left on a piece of paper. Information about how the drawing is tied together is stored in the computer as well the information which gives the drawing its particular appearance.

------Ivan Sutherland

Ivan Sutherland, SketchPad: A machine graphical communication system. In *Proceedings of the Spring Joint Computer Conference*, (1963) pp 329 - 346.

Sketchpad Innovations

Software inventions: cursor, window (zooming, not scrolling), clipping rubber band drawing, constraint based drawing, gesture recognition

Software structures: Object-Oriented structures including: master (class definition), instances, inheritance, modifications to instances, instances as masters, etc.

Open Software Architectures

To build effective useful software to do something, we must store information about:

- **how to perform the required functions**
- **how the various parts relate to one another**

Open Software Architectures

Open systems must carry their own model so they can evolve, therefore:

Specifications of components must be visible from the system.

Implementations of components must be available to the system.

How did we get in the mess we're in?

Here are the steps...

Abstraction

Programming Language (individual programs)

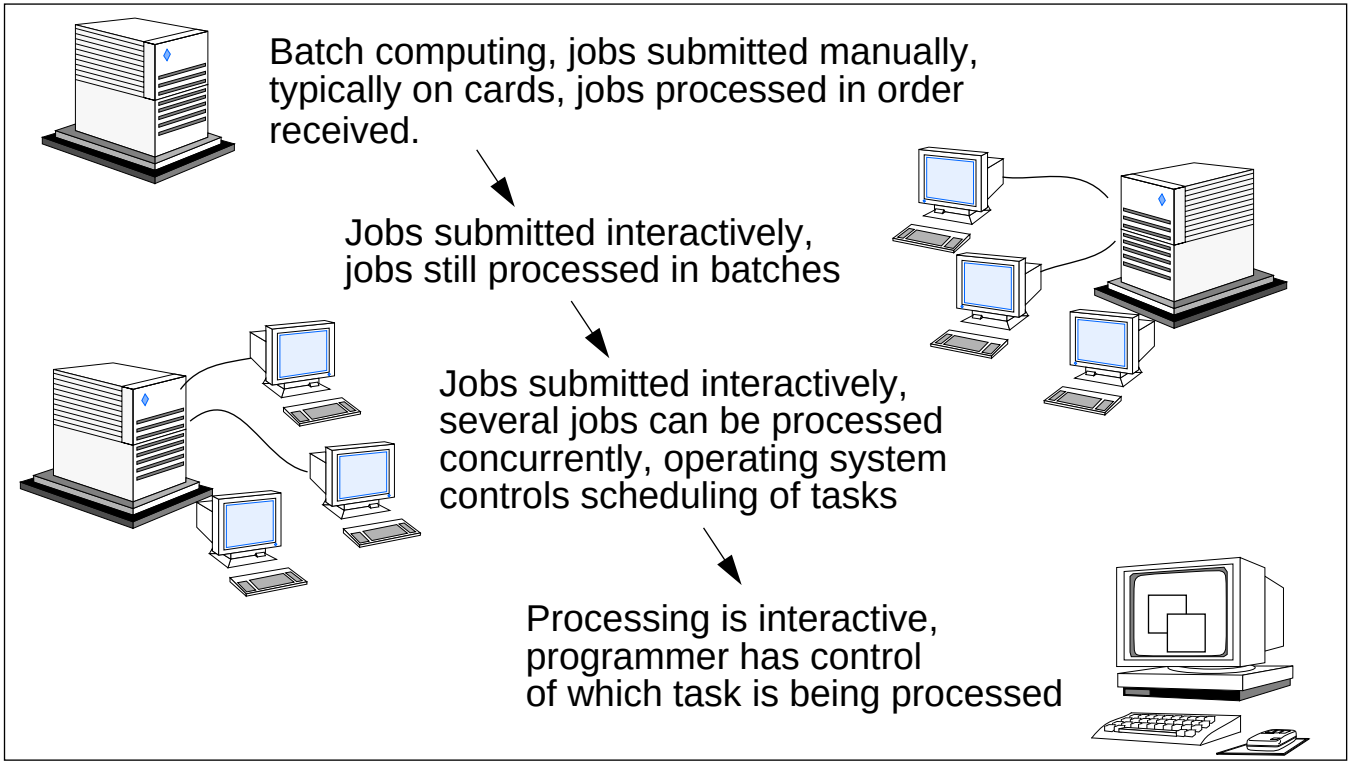
System Architecture (relationship among programs)

assembly languages

block structured languages

higher-level programming languages

object-oriented languages



batch

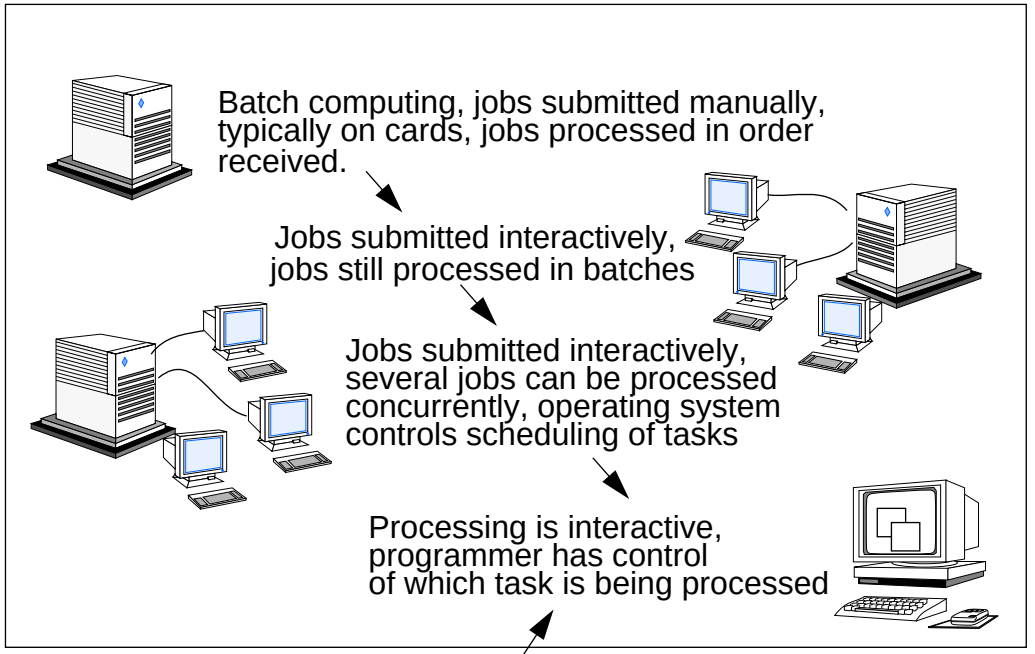
multi-access

interactive

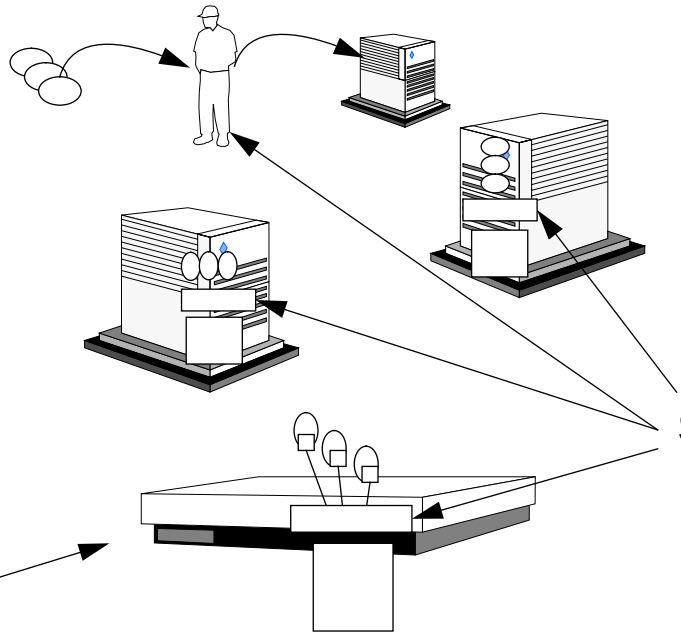
direct manipulation

The operating system and language do more for the application.

Automation



Software Architecture

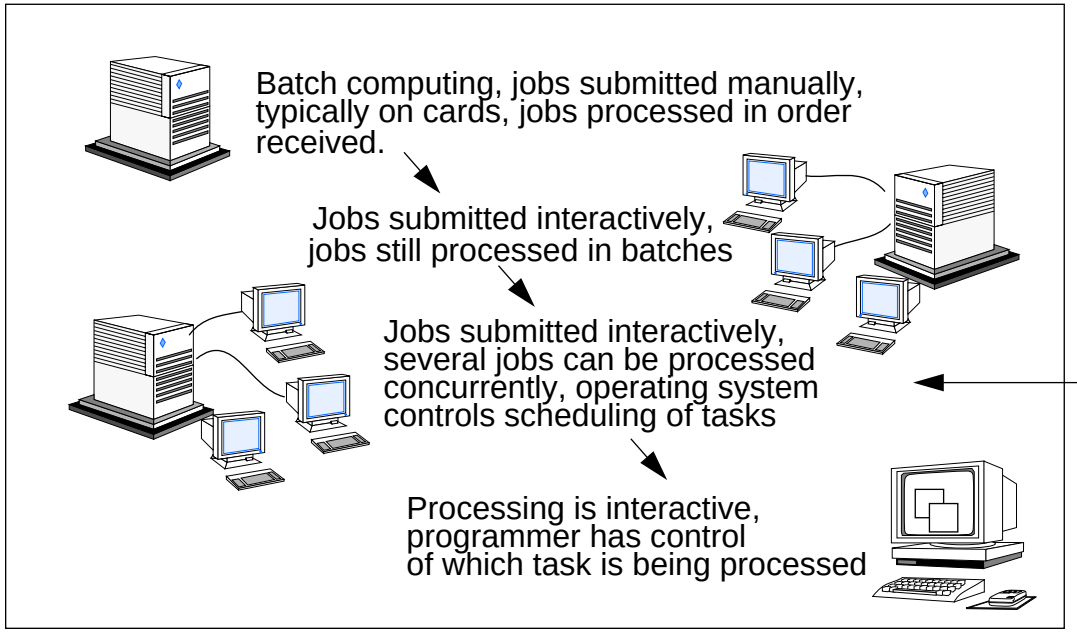


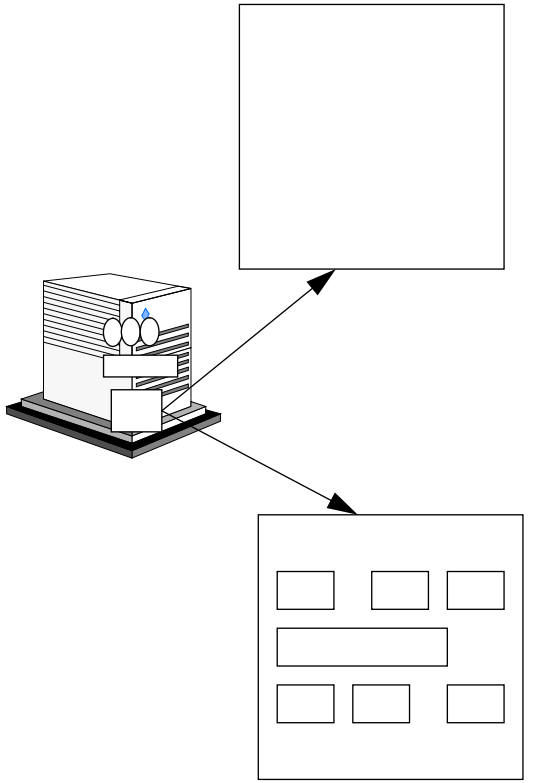
Each application carries its own bespoke operating system

system software

application software

Divide and Conquer



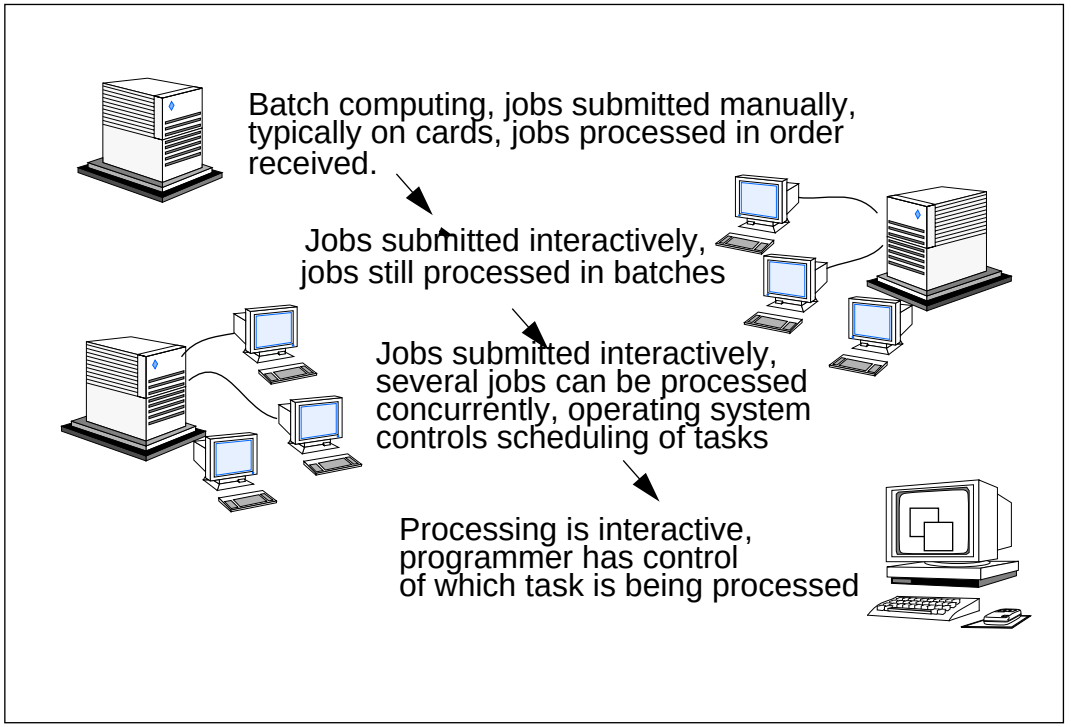


monolithic operating system

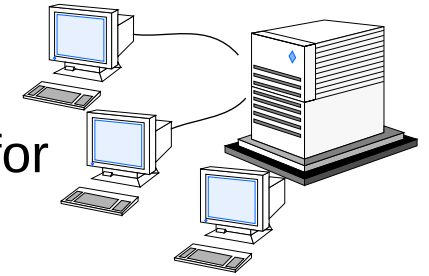
light-weight kernel, with various functions separated out.

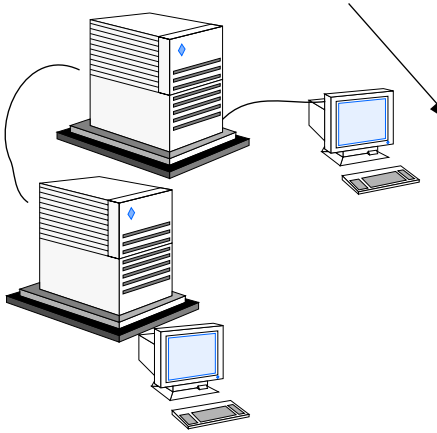
Traditional operating system is now complex and monolithic. Unix system created the notion of a small operating system kernel with system functions provided as services.(1975)

Network Computing



Early data communications
Remote access for terminals.



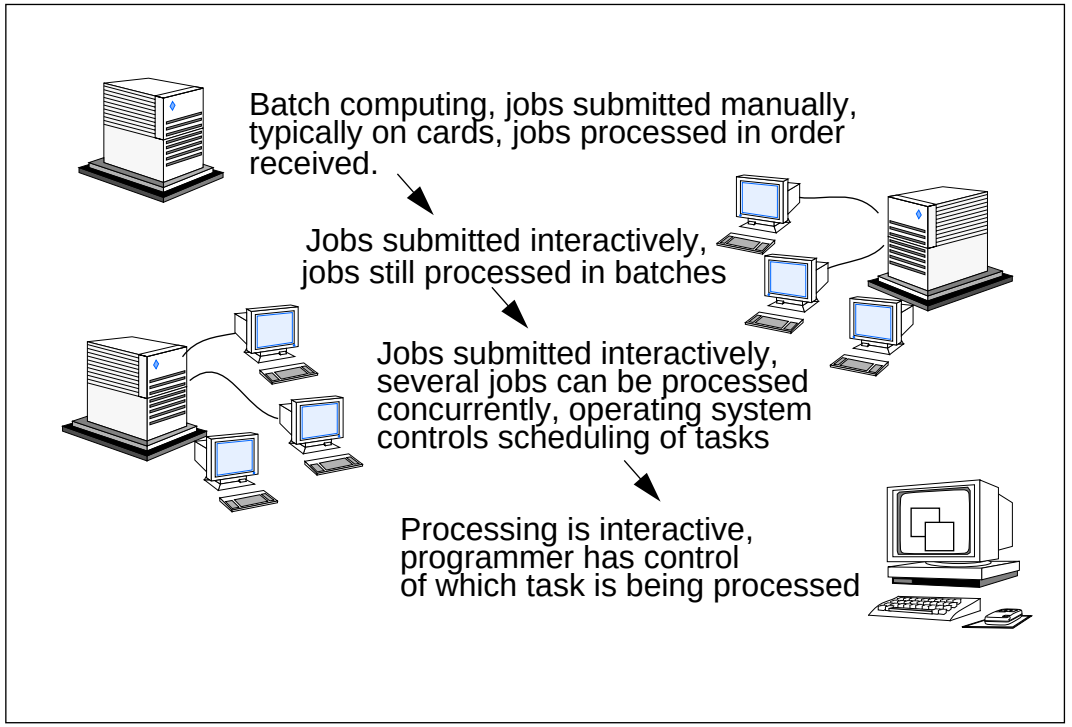


File transfer protocol is used to move files between machines

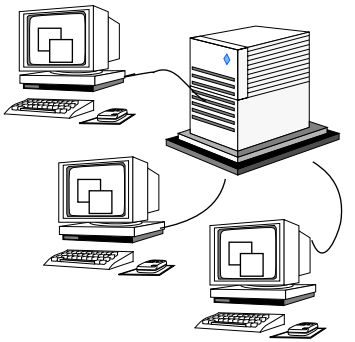
Interprocess Communication (IPC) is based on shared memory.

Explicit Communications is Added

Client-Server Computing



Client-Server (Workstations)



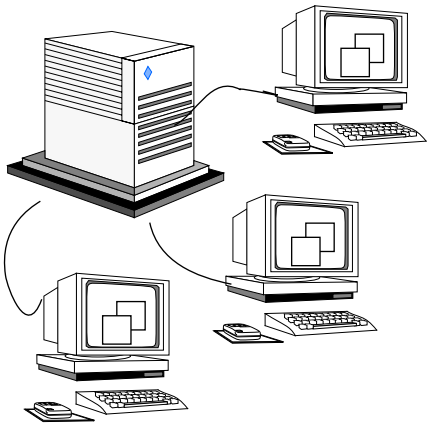
Each workstation (client) has its own operating system, which supports both multi-tasking and direct-manipulation. Server also has its own operating system. Clients and Servers co-operate.

Interprocess Communication (IPC) is based on shared memory.

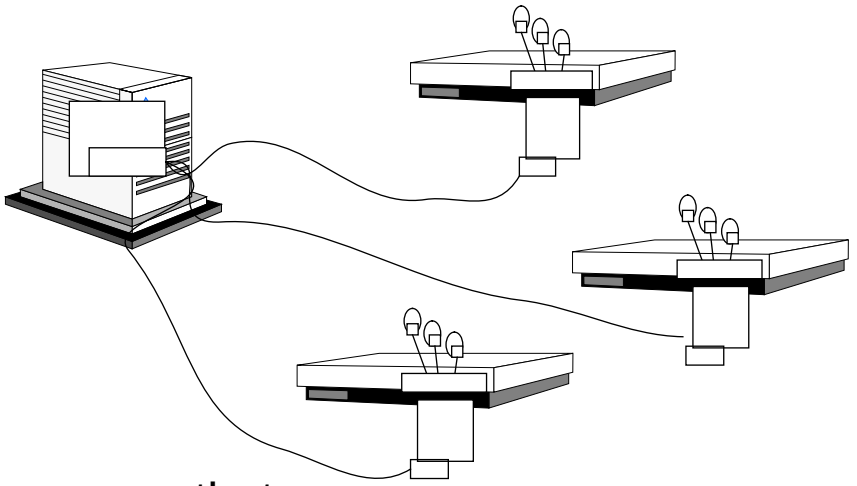
IPC between Clients and Servers is based on a communications protocol.

Client-Server Computing

Physical Architecture



Software Architecture

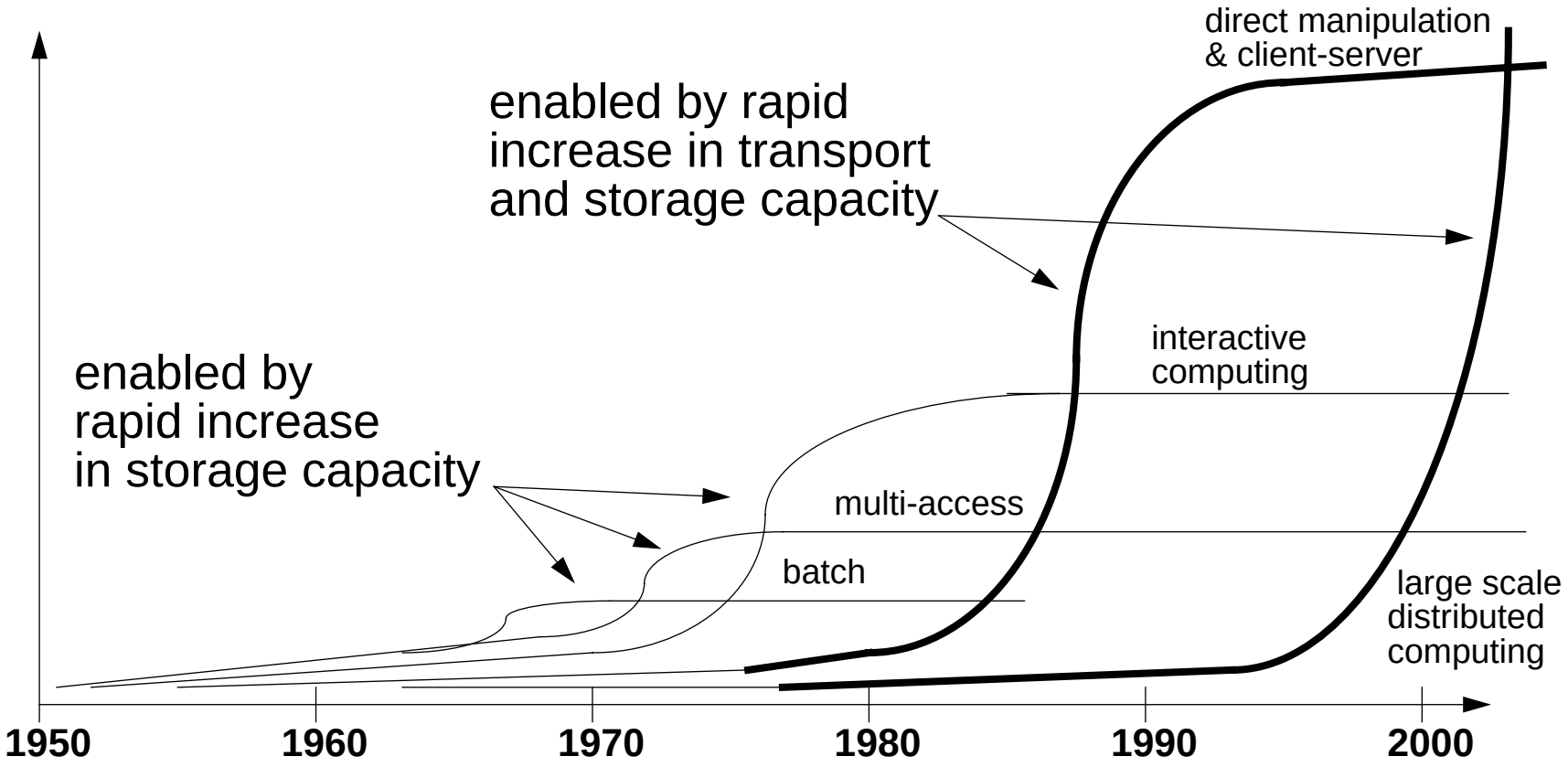


To create an application program that can communicate with applications that reside in other locations, the programmer must know their names, locations,....

application software

system software

System Software Generations



How does ANSA help us out of this mess?

Different Applications, Different Needs

There isn't going to be a single response to the wish list, because we are in a world of some very fundamental trade-offs, including.....

- **ABSTRACTION versus SPECIALIZATION**

The more you hide, the less chance there is to tinker

- **CONSISTENCY versus AVAILABILITY**

Availability means copies, increases risk of inconsistency

- **AUTONOMY versus UNIFORMITY**

Autonomy gives more freedom, but leads to differences which increases complexity

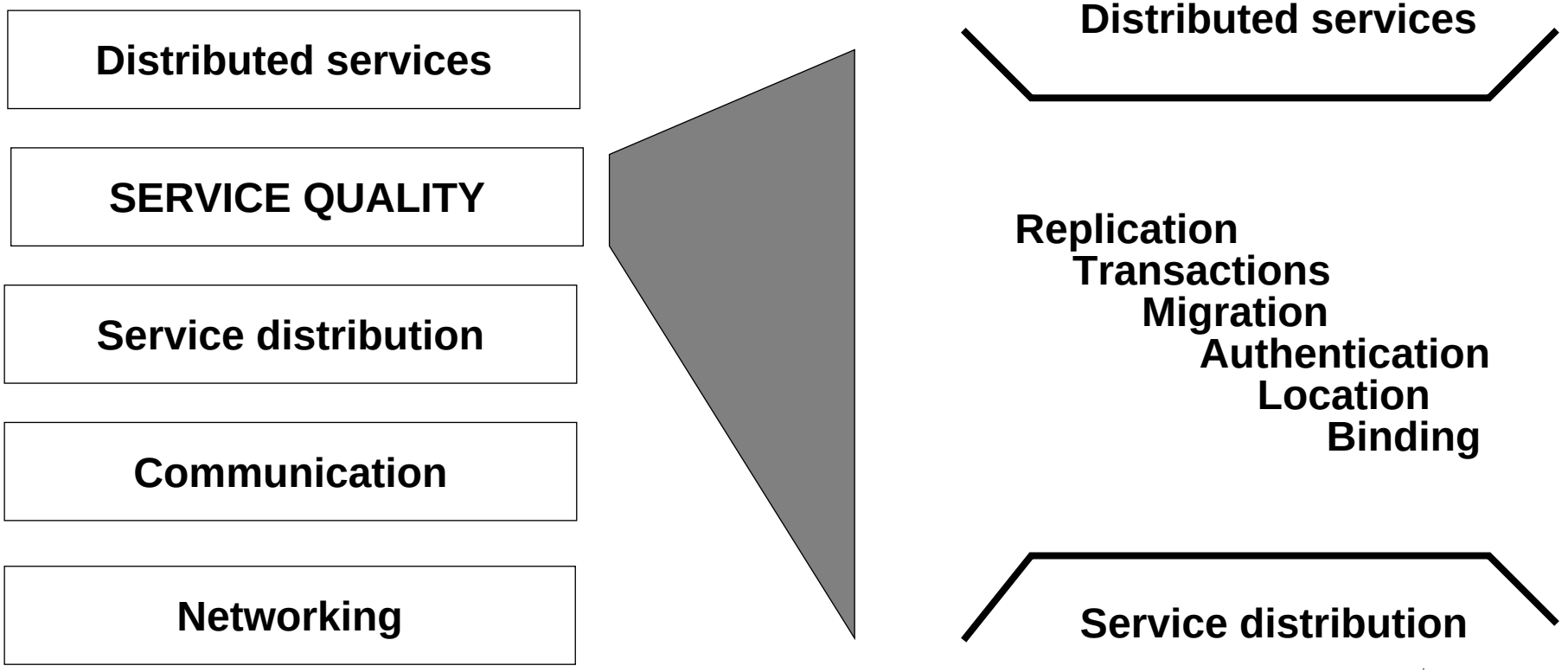
- **SECURITY versus CONVENIENCE**

Security makes things harder to do

- **any many other trade-offs**

Means that one size does not fit all.

Structure of a Distributed System



Architecture Checklist

Divide & Conquer (clear separation of concern)

Abstract & Automate (enable the use of software tools)

Minimize Software Bureaucracy (bookkeeping & overhead)

Behave predictability (performance & reliability)

Be extensible and scalable

Incorporate existing software effectively

Obey standards whenever beneficial

ANSA Principles

Define SERVICES

quality, location, function

Reverse CONVENTIONAL assumptions

to get distribution and federation

Separate CONCERNS

using the projections to gain clarity

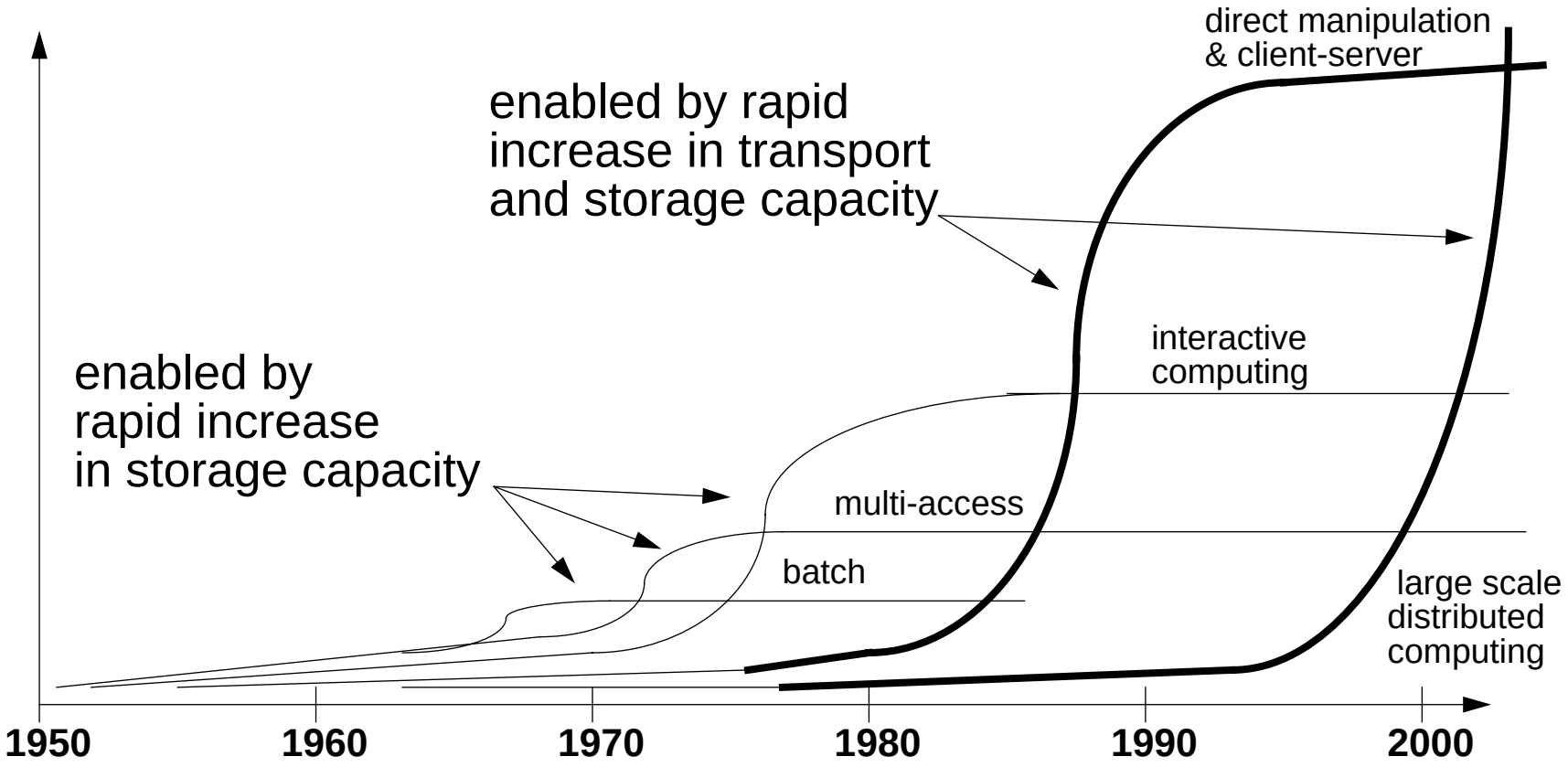
HIDE irrelevant detail

using transparencies

What's Missing?

Managing the Application Objects.

System Software Generations



Distributed Computing Straddles Two Generations

Current generation is *technology driven*:

DCE, CORBA, etc. are building on C++, Unix, etc.

Upcoming generation is *service driven*:

Must manage services needed for distributed computing in large scale systems.

Scenario for ANSA phase III

Phase I laid foundations and theory

Phase II developed and tested basic technology

Phase III adds dependability, performance, automation, management of federations

Phase III results will enable creation of *Management Engines* for federated, networked islands.

Approach to Scenario

Many different examples = individual scenarios

Generic Scenario = *Management Engine*

Map back onto selected individual scenarios = Proof of Concept

The “Management Engine”