Unit 15:
Experimental Microkernel Systems

15.2. Chorus: Microkernel and User-space Actors

CHORUS: a new Look at Microkernel-based Operating Systems

- OS structuring:
  - modular set of system servers which sit on top of a minimal microkernel, rather than using the traditional monolithic structure.
- Microkernel provides generic services:
  - such as processor scheduling and
  - memory management, independent of a specific operating system.
- The microkernel also provides:
  - a simple **inter-process communication (IPC)** facility
  - that allows system servers to call each other and exchange data
  - regardless of where they are executed,
  - in a multiprocessor, multicomputer, or network configuration.

CHORUS: Principles

- Primitive services form a standard base
  - support the implementation of operating system-specific functions
- System-specific functions can be configured
  - into system servers managing the other physical and logical resources of a computer system,
  - such as files, devices and high-level communication services.
- A set of system servers is referred to as a **subsystem**.
- Real-time systems tend to be built along similar lines
  - simple, generic executive
  - supporting application-specific real-time tasks.

UNIX and Microkernels

- UNIX introduced the concept of a standard, hardware-independent operating system
  - portability allowed platform builders to reduce their time to market.
  - However, today's versions become increasingly more complex.
  - For example, UNIX is being extended with facilities for real-time applications and on-line transaction processing.
- Even more fundamental is the move toward distributed systems.
  - It is desirable in today's computing environments that new HW/SW resources, be integrated into a single system, distributed over a network.
  - The range of communication media includes shared memory, buses, high-speed networks, local-area networks, and wide-area networks.
- This trend to integrate new hardware and software components will become fundamental as collective computing environments emerge.

Problems

- The attempt to reorganize UNIX to work within a microkernel framework poses problems.
- A primary concern is efficiency:
  - a microkernel-based modular operating system must provide performance comparable to that of a monolithic kernel.
  - The elegance and flexibility of the client-server model may exact a cost in message-handling and context-switching overhead.
  - If this penalty is too great, commercial acceptance will be limited.
- Another pragmatic concern is compatibility:
  - compatible interfaces are needed not only at the level of application programs,
  - but also for device drivers,
  - streams modules, and other components.
  - In many cases binary as well as source compatibility is required.
- These concerns affect the structure of the operating system.

Some CHORUS History

- First implementation of a UNIX-compatible microkernel-based system was developed 1984-1986 as a research project at INRIA.
  - explore the feasibility of shifting as much function as possible out of the kernel
  - demonstrate that UNIX could be implemented as a set of modules that did not share memory.
- In late 1986, an effort to create a new version was launched
  - based on an entirely rewritten CHORUS nucleus; at Chorus syste‘mes.
- The current version some new goals,
  - including real-time support and - not incidentally - commercial viability.
- UNIX subsystem
  - compatible with System V Release 3.2 is currently available (1992),
  - with System V Release 4.0 and 4.4BSD systems under development.
  - implementation performs comparably with well-established monolithic-kernels
- The system has been adopted for use in commercial products
  - ranging from X terminals and telecommunication systems to
  - mainframe UNIX machines.

CHORUS V2 Overview

- The CHORUS project, while at INRIA, began researching distributed operating systems with CHORUS V0 and V1.
  - These versions proved the viability of a modular, message-based distributed operating system, examined its potential performance, and explored its impact on distributed applications programming.
- CHORUS V2 represented the first intrusion of UNIX into the peaceful CHORUS landscape. The goals were:
  1. To add UNIX emulation to the distributed system technology of CHORUS V1;
  2. Demonstrate the feasibility of a UNIX implementation with a minimal kernel and semi-autonomous servers;
  3. To explore the distribution of UNIX services;
  4. And to integrate support for a distributed environment into the UNIX interface.
- CHORUS architecture has always consists of a modular set of servers running on top of a microkernel (the nucleus) which included all of the necessary support for distribution.

CHORUS v2: Execution model

- The basic execution entities supported by the v2 nucleus were mono-threaded actors running in user mode and isolated in protected address spaces.
  - Execution of actors consisted of a sequence of ``processing-steps'' which mimicked atomic transactions:
  - ports represented operations to be performed; messages would trigger their invocation and provide arguments.
  - The execution of remote operations were synchronized at explicit ``commit'' points.
  - An ever present concern in the design of CHORUS was that fault-tolerance and distribution are tightly coupled; hardware redundancy both increases the probability of faults and gives a better chance to recover from these faults.

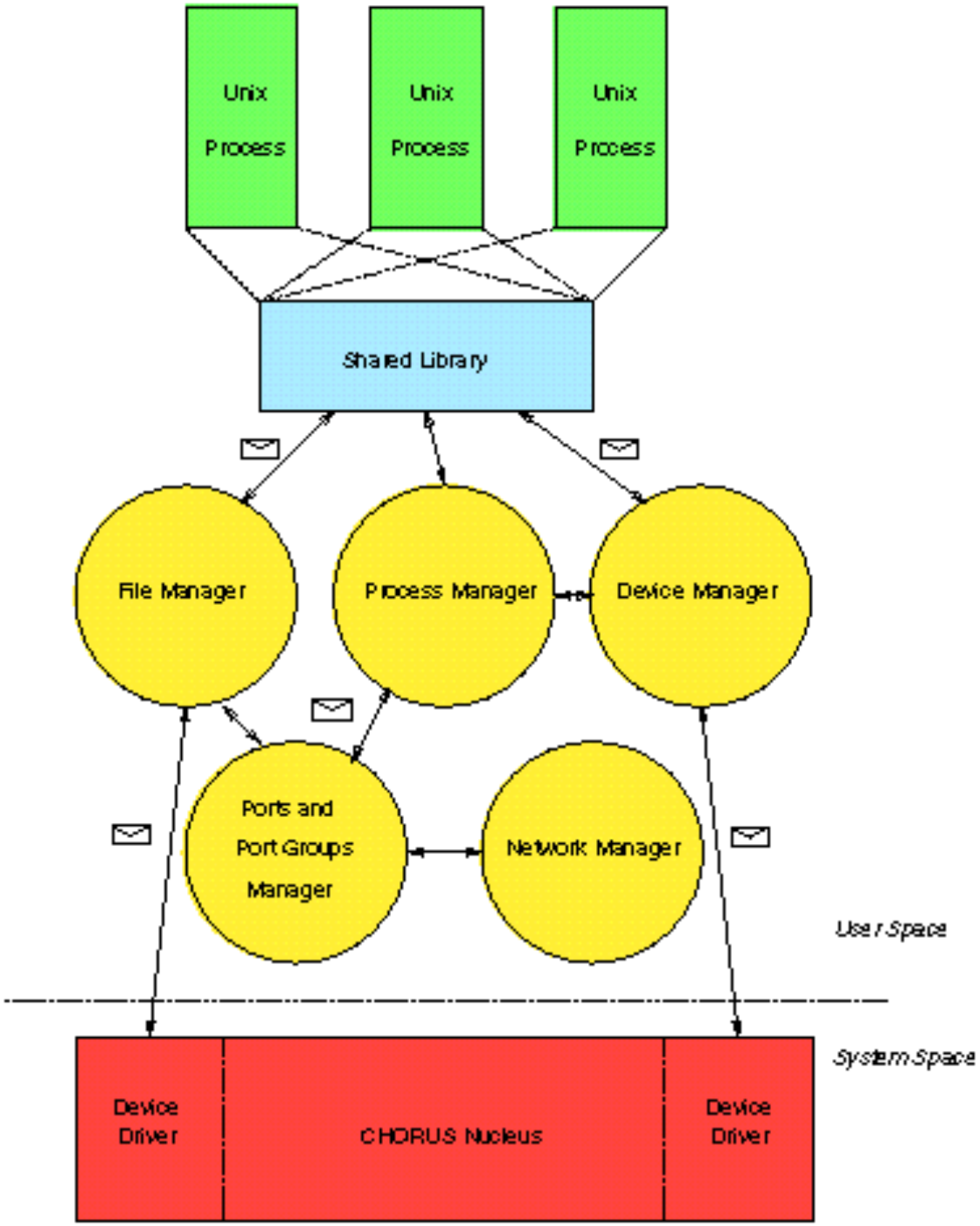
CHORUS v2 Communication

- Communication in CHORUS V2 was, as in many current systems, based upon the exchange of messages through ports.
  - Ports were attached to actors, and had the ability to migrate from one actor to another.
  - Furthermore, ports could be gathered into port groups, which allowed message broadcasting as well as functional addressing.
  - The port group mechanism provided a flexible set of client-server mapping semantics including dynamic reconfiguration of servers.
- Ports, port groups, and actors were given global unique names, constructed in a distributed fashion by each nucleus for use only by the nucleus and system servers.
  - Private, context-dependent names were exported to user actors. These port descriptors were inherited in the same fashion as UNIX file descriptors.

UNIX on top of CHORUS v2

- A full UNIX System V was built on top of CHORUS v2.
- UNIX was split into three servers:
  - a process manager, dedicated to process management,
  - a file manager for block device and file system management, and
  - a device manager for character device management.
- In addition, the nucleus was complemented with two servers, one which managed ports and port groups, and another which managed remote communications
- UNIX network facilities (sockets) were not implemented at this time.

UNIX on CHORUS v2

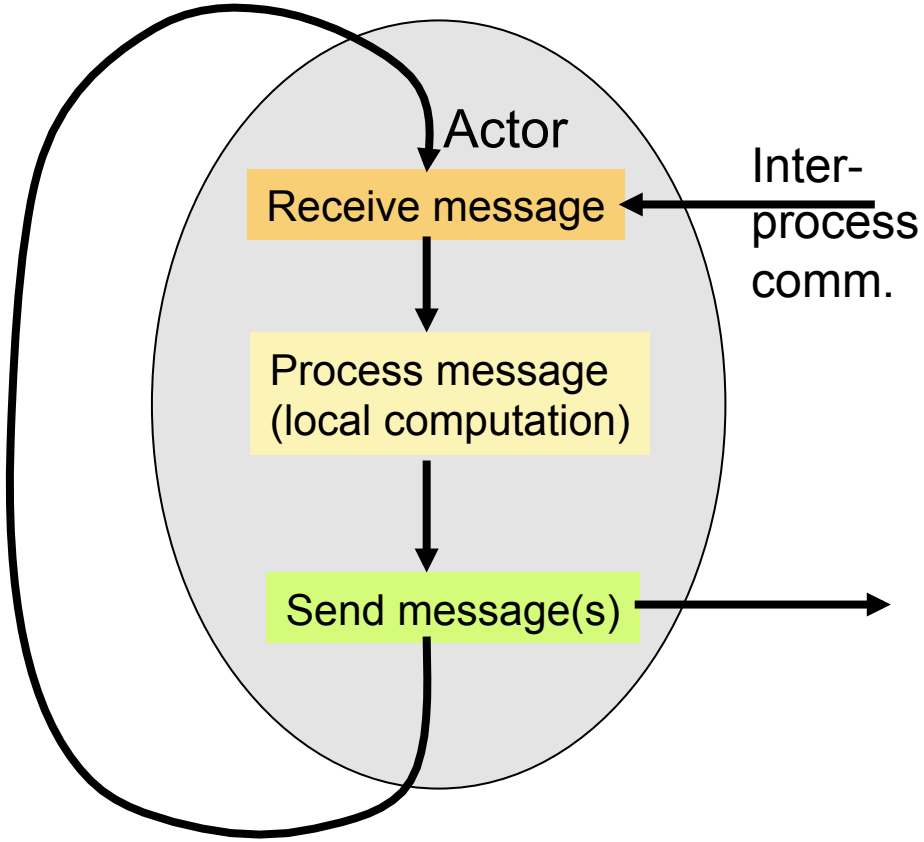


Actors

- A UNIX process was implemented as a CHORUS actor.
  - All interactions of the process with its environment, i.e. all system calls, were performed as exchanges of messages between the process and system servers.
  - Signals were also implemented as messages.
- This “modularization“ impacted UNIX in the following ways:
  1. UNIX data structures were split between the nucleus and several servers.
  2. Most UNIX objects, files in particular, were designated by network-wide capabilities which could be exchanged freely between subsystem servers and sites.
- As many UNIX system calls as possible were implemented by a process-level library.
- The process context was stored in process-specific library data at a fixed, read-only location within the process address space.
  - The library invoked the servers, when necessary via RPC.

The Actor Paradigm

- Sequential execution; single-threaded
- Synchronous
- Message-based communication



CHORUS v2 behavior

- For example:
  - the process manager was invoked to handle a fork(2) system call and
  - the file manager for a read(2) system call on a file.
- Source-level compatibility with UNIX only:
  - The library resided at a predefined user virtual address in a write-protected area.
  - Library data holding the process context information was not completely secure from malicious or unintentional modification by the user.
  - Errant programs could experience new, unexpected error behavior.
  - Programs that depended upon the standard UNIX address space layout could cease to function because of the additional address space contents.

Extended UNIX Services

CHORUS V2 extended UNIX services in two ways:

- by allowing their distribution while retaining their original interface (e.g. remote process creation and remote file access).
- by providing access to new services without breaking existing UNIX semantics (e.g. CHORUS IPC).

Basic Abstractions

Unique Identifier (UI)	global name
Actor	unit of resource allocation
Thread	unit of sequential execution
Message	unit of communication
Port, Port Groups	unit of addressing and (re)configuration basis
Region	unit of structuring of an Actor address space

Segment	unit of data encapsulation
Capability	unit of data access control
Protection Identifier	unit of authentication

CHORUS v3 Goals

- The design of CHORUS v3 system has been strongly influenced by a new major goal:
  to design a microkernel technology suitable for the implementation of commercial operating systems.
- CHORUS v2 was a UNIX-compatible distributed operating system.
- The CHORUS v3 microkernel is able to support operating system standards while meeting the new needs of commercial systems builders.

Chorus v3 Design Guidelines

- Portability:
  - CHORUS v3 microkernel must be portable to many machine architectures.
  - Architecture-independent memory management system, replacing the hardware-specific CHORUS v2 memory management.
- Generality:
  - the CHORUS v3 microkernel must provide a set of functions that are sufficiently generic to allow the implementation of many different sets of OS semantics; some UNIX-related features had to be removed from nucleus.
  - The nucleus must maintain its simplicity and efficiency for users or subsystems which do not require high level services.
- Compatibility:
  - UNIX source compatibility in CHORUS v2 had to be extended to binary compatibility in v3, both for user applications and device drivers.
  - In particular, the CHORUS v3 nucleus had to provide tools to allow subsystems to build binary compatible interfaces.

CORUS v3 Design Guidelines (contd.)

- Real-time:
  - process control and telecommunication systems comprise important targets for distributed systems.
  - In these areas, the responsiveness of the system is of prime importance.
  - The CHORUS V3 nucleus is, first and foremost, a distributed real-time executive.
  - The real-time features may be used by any subsystem, allowing for example, a UNIX subsystem to be naturally extended to be suitable for real-time applications needs.
- Performance:
  - for commercial viability, good performance is essential in an operating system.
  - While offering the base for building modular, well-structured operating systems, the nucleus interface must allow these operating systems to reach at least the same performance as conventional, monolithic, implementations.

New Solutions

- A CHORUS v3 actor is a resource container
  - address space in which multiple threads may execute.
  - Threads are scheduled as independent entities, allowing real parallelism on a multiprocessor architecture.
  - Multiple threads allow the simplification of the control structure of server-based applications.
  - New nucleus services, such as thread execution control and synchronization have been introduced.

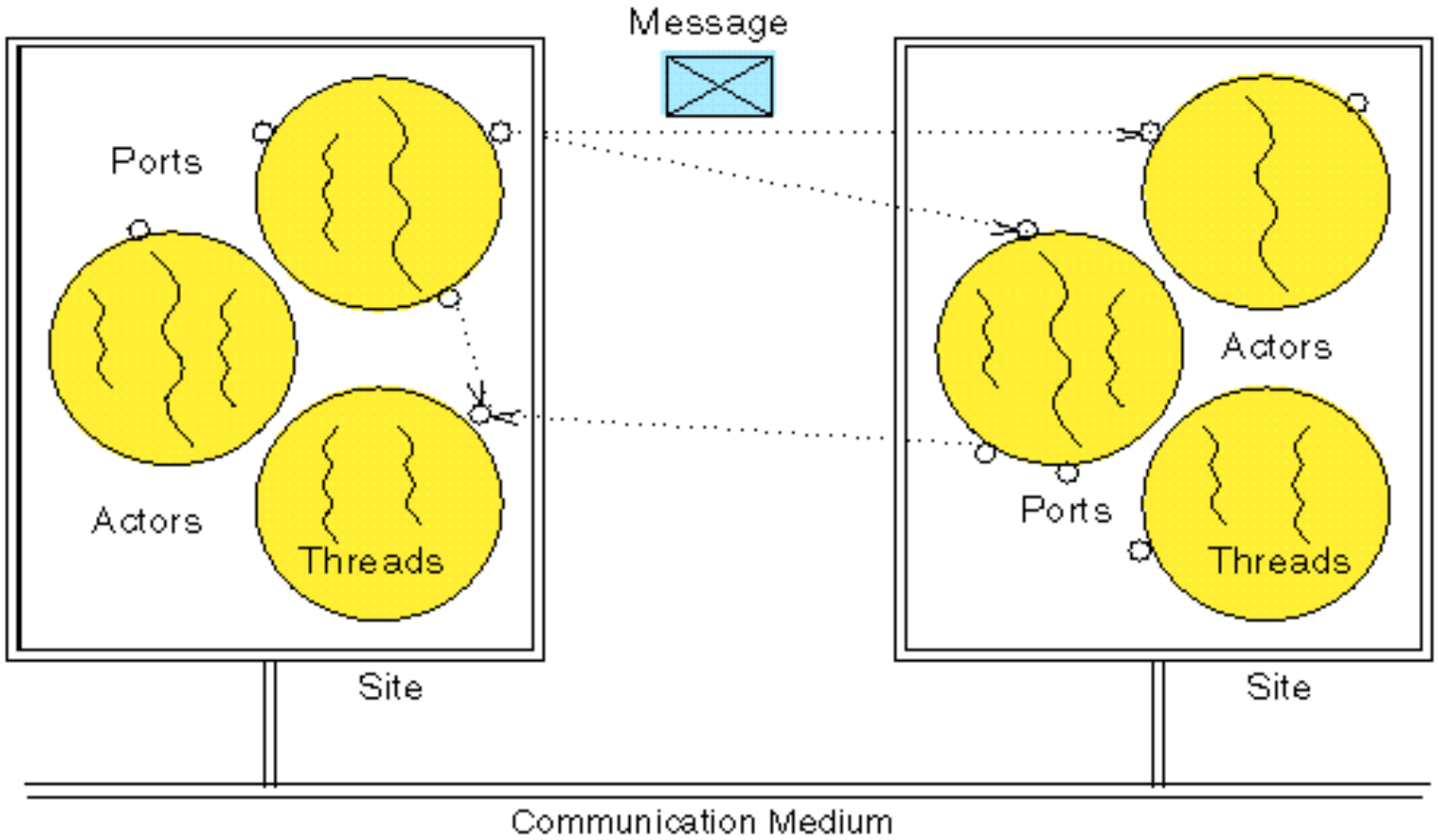
CHORUS Inter-Process Communication

- The inter-process communication model also evolved:
  - atomic transactions of v2 have been replaced, in v3, by RPC paradigm
  - has since evolved into an extremely efficient lightweight RPC protocol.
- IPC messages remain untyped.
- The CHORUS IPC mechanism is simple and efficient when communicating among homogeneous sites.
  - When communicating between heterogeneous sites, higher-level protocols are used, as needed.
  - allow the construction of simple and efficient applications without forcing them to pay a penalty for sophisticated mechanisms which were required only by specific classes of programs.

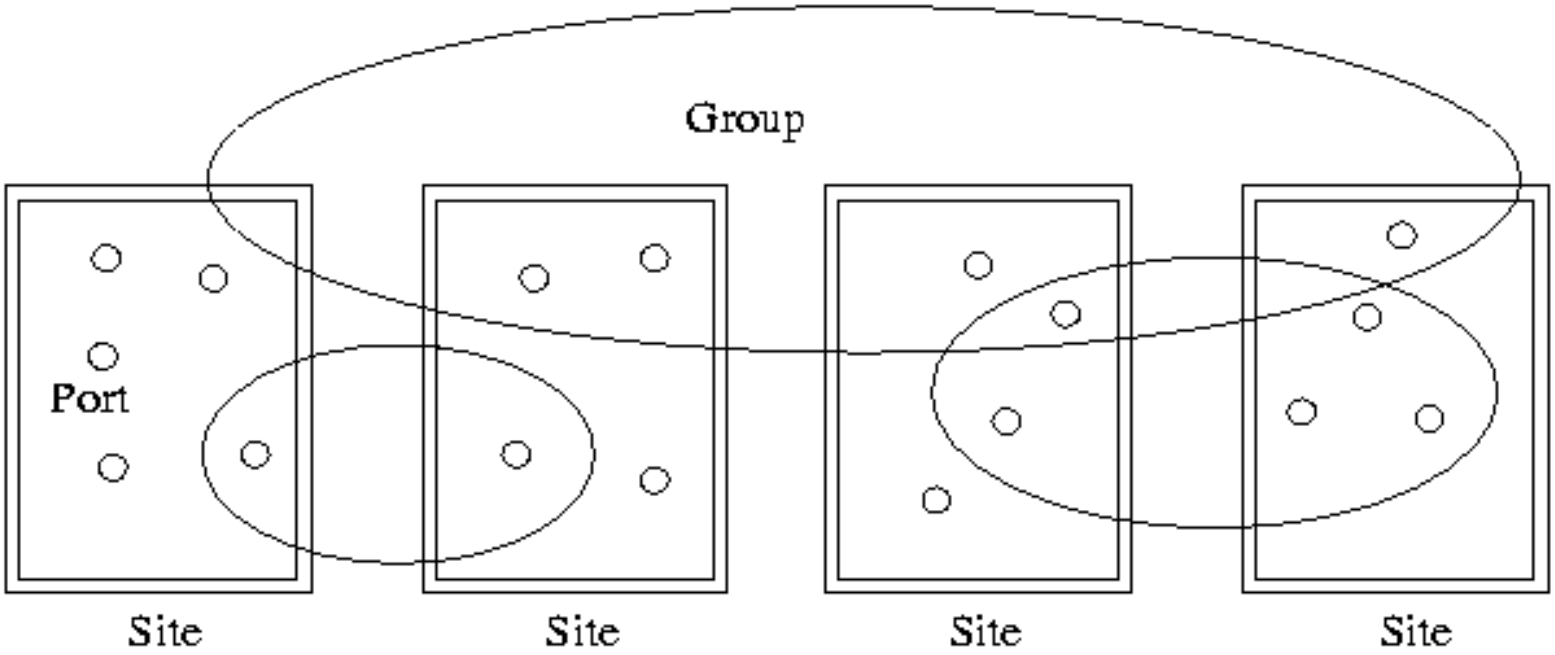
CHORUS Ports

- CHORUS v3 makes global names of ports and port groups (unique identifiers) visible to the user, discarding the UNIX-like CHORUS v2 contextual naming scheme.
- Contextual identifiers turned out not to be an effective paradigm.
- The first consequence of using unique identifiers is simplicity:
  - port and port group names may be freely exchanged by nucleus users,
  - avoiding the need for the nucleus to maintain complex actor context.
- The second consequence is a lower level of protection:
  - the CHORUS V3 philosophy is to provide subsystems with the means for implementing their own level and style of protection rather than enforcing protection directly in the microkernel.
  - For example, if the security of v2 context-dependent names is desired, a subsystem can easily and efficiently export a protected name-space server.

Actors, Ports, Threads, Messages

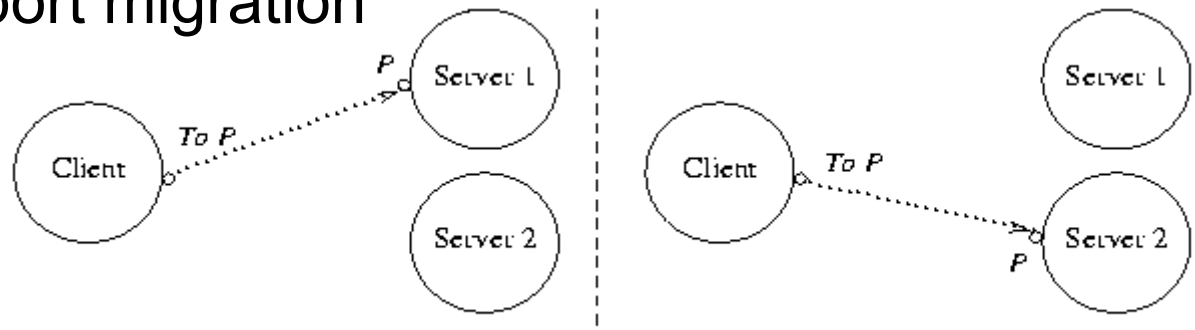


Chorus Port Groups



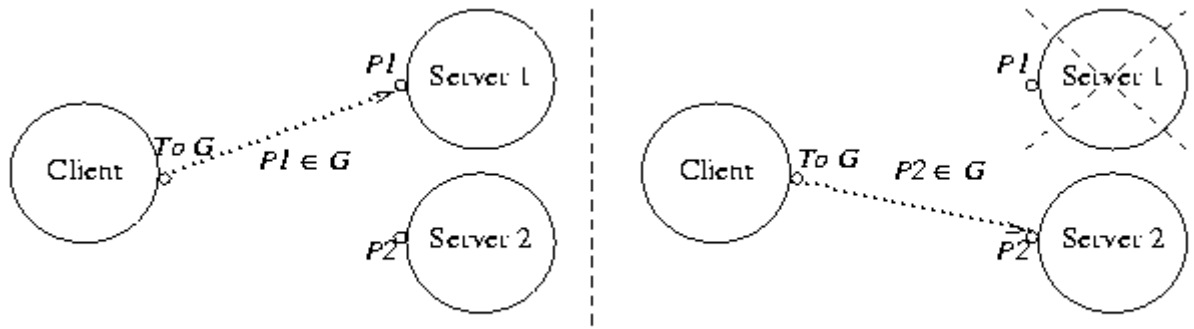
Reconfiguration

...using port migration



Port P migrates from Server 1 to Server 2

...using port groups



P2 has replaced P1 in group G

Lesson: UNIX Port Extensions

- CHORUS IPC interface maintained normal UNIX-style semantics.
  - Employing the same form as the UNIX file descriptor for port descriptors was intended to provide uniformity of model.
  - The semantics of ports were sufficiently different from the semantics of files to negate this advantage.
  - In operations such as fork, for example, it did not make sense to share port descriptors in the same fashion as file descriptors. Attempting to force ports into the UNIX model resulted in confusion.
- Lesson: A user-level IPC interface was important, but giving it UNIX semantics was cumbersome and unnecessary.
  - This lesson is an example of a larger principle; the nucleus abstractions should be primitive and generally applicable - they should not be coerced into the framework of a specific operating system.

Virtual Memory

- The machine dependent virtual memory system of CHORUS V2 has been replaced, in V3, by a highly portable VM system.
  - The VM abstractions presented by the V3 nucleus include ``segments'' and ``regions.''
  - Segments encapsulate data within a CHORUS system and typically represent some form of backing store, such as a swap area on a disk.
  - A region is a contiguous range of virtual addresses within an actor that map a portion of a segment into its address space.
  - Requests to read or to modify data within a region are converted by the virtual memory system into read or modify requests within the segment.
  - ``External Mappers'' interact with the virtual memory system using a nucleus-to-Mapper protocol to manage data represented by segments. Mappers also provide the synchronization needed to implement distributed shared memory.

Actor Context

- CHORUS V2 was built around a ``pure'' message-passing model,
  - strict protection was incorporated at the lowest level; all servers were implemented in protected user address spaces.
- This distinct separation enforced a clean, modular design of a subsystem. However, it also led to several problems:
  1. A UNIX subsystem based on CHORUS V2 required the use of user-level system call stubs and altered the memory layout of a process and, therefore, could never provide 100% binary compatibility;
  2. All device drivers were required to reside within the nucleus;
  3. Context switching expense was prohibitively high.
- The most fundamental enhancement made between CHORUS V2 and V3 was the introduction of the supervisor actor.

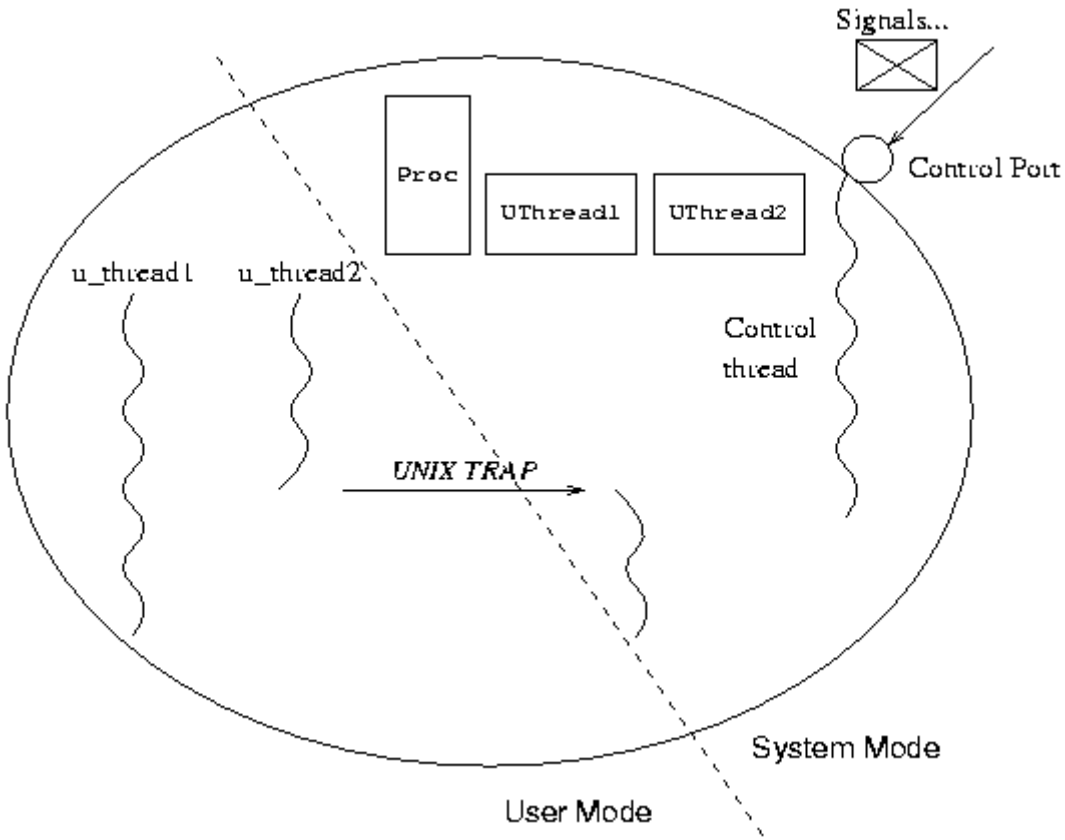
Supervisor Actors

- Supervisor actors share the supervisor address space and their threads execute in a privileged machine state.
  - Although they reside within the supervisor address space, supervisor actors are truly separate entities;
  - they are compiled, link edited, and loaded independently of the nucleus and of each other.
- The introduction of supervisor actors creates several opportunities for system enhancement in the areas of compatibility and performance.

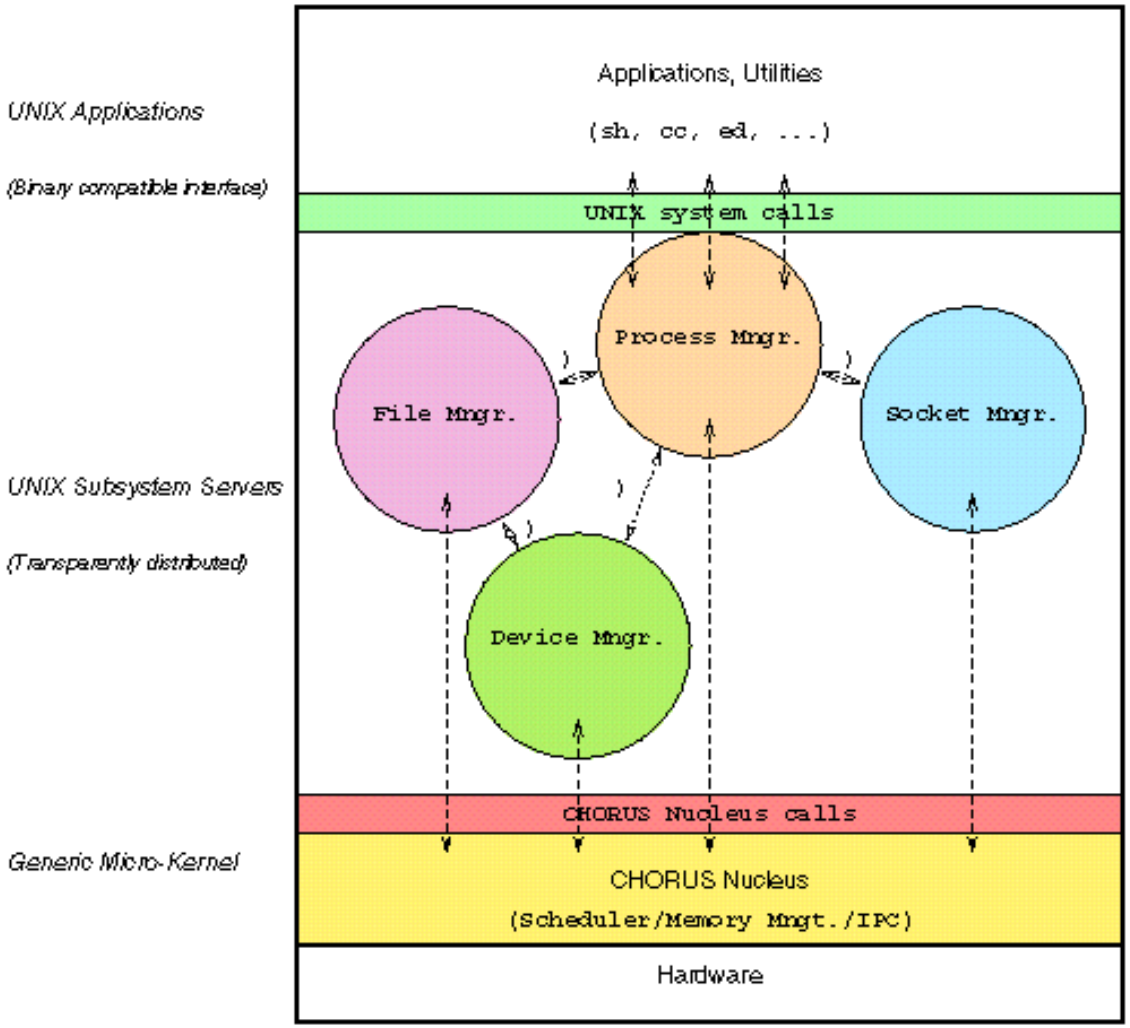
UNIX Subsystem

- As a consequence of these nucleus evolutions, the UNIX subsystem implementation has also evolved.
  - In particular, full UNIX binary compatibility has been achieved. Internally, the UNIX subsystem makes use of new nucleus services, such as multi-threading and supervisor actors.
  - The CHORUS v2 user-level UNIX system-call library has been moved inside the process manager and is now invoked directly by system-call traps.
- Modularization is difficult.
  - Care must be taken to decompose the data structures and function along meaningful boundaries.
  - Performing this decomposition is an iterative process.
  - The system is first decomposed along broad functional lines. The data structures are then split accordingly, possibly impacting the functional decomposition.

UNIX Process as a CHORUS Actor



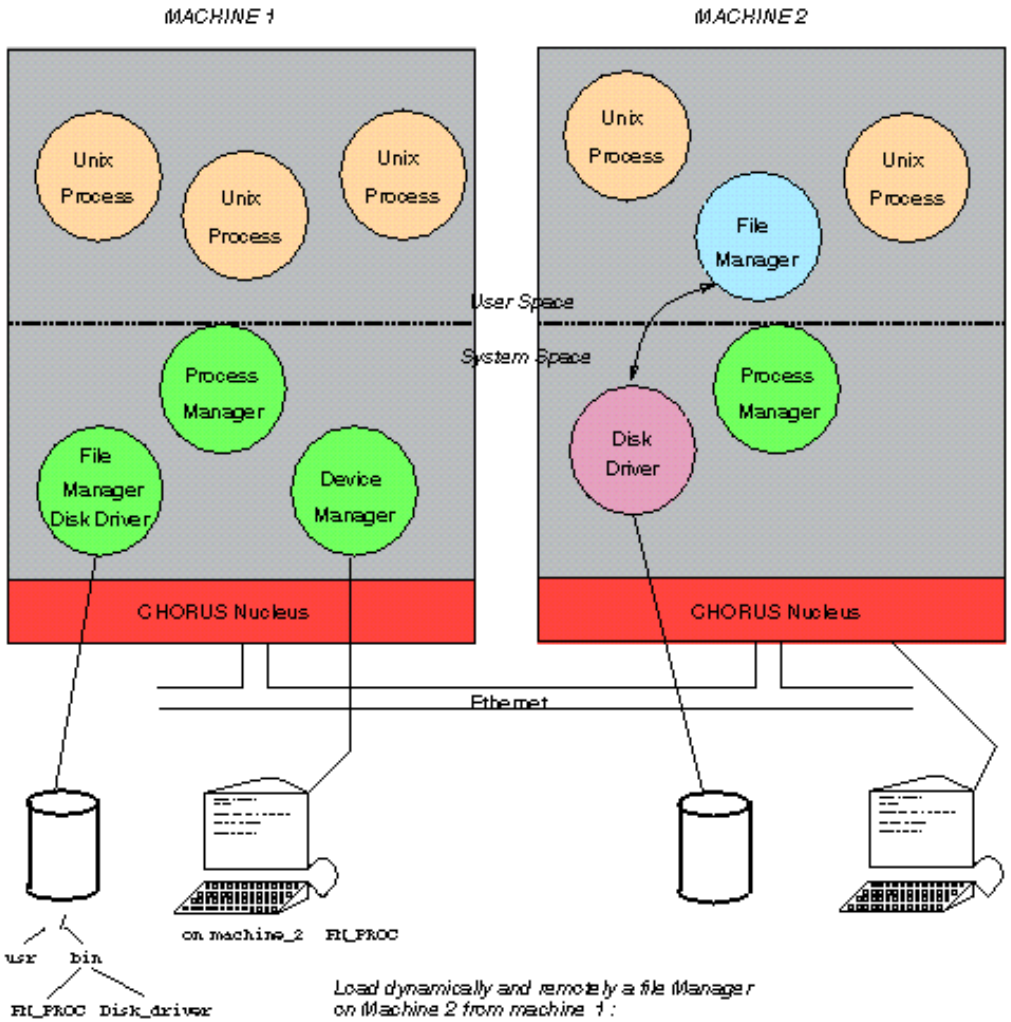
CHORUS/MiX-V3 Architecture



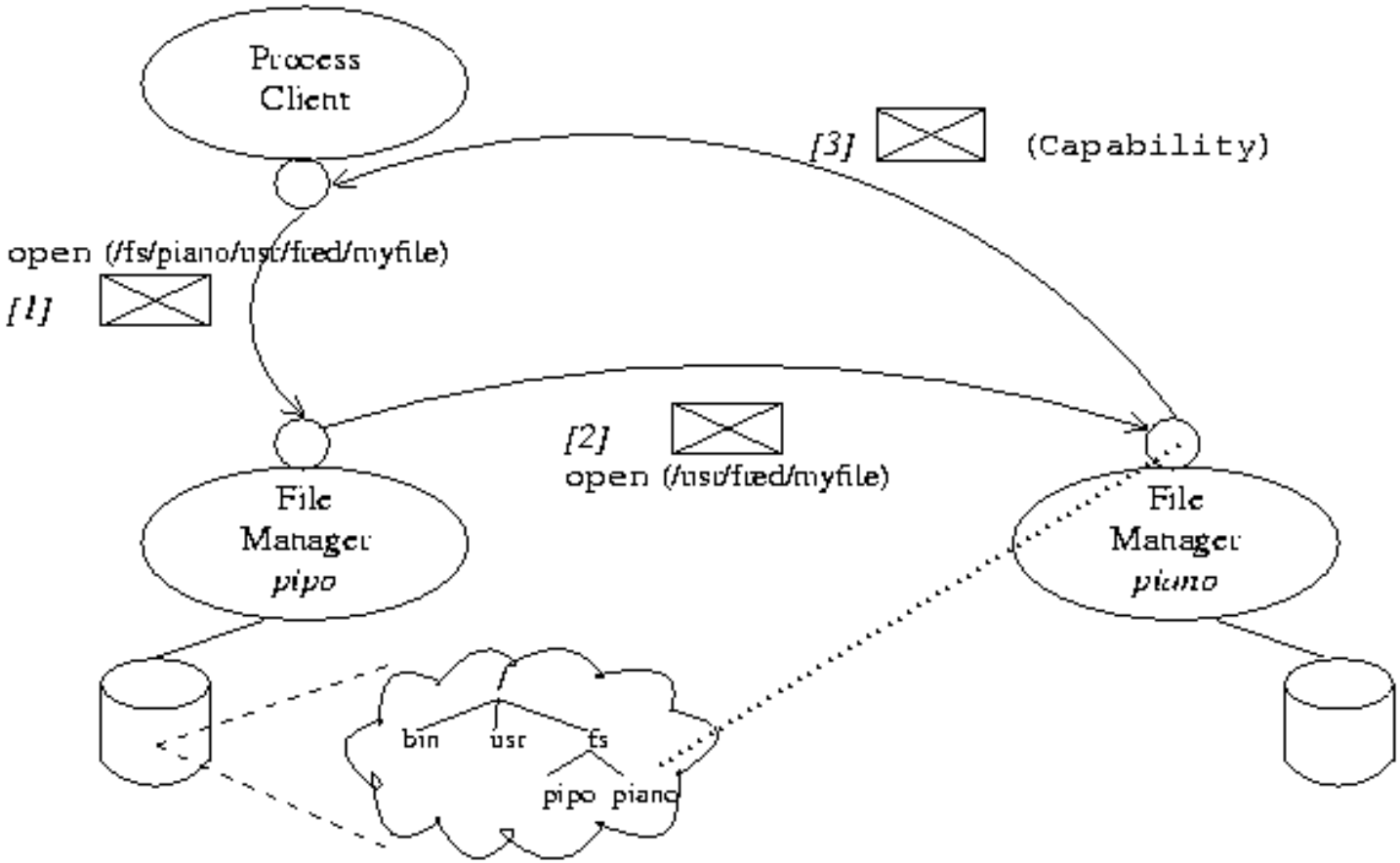
Supervisor Actors (contd.)

- Supervisor actors, alone, are granted direct access to the hardware event facilities.
  - any supervisor actor may dynamically establish a handler for any particular hardware interrupt, system call trap, or program exception.
  - A connected handler executes as an ordinary subroutine, called directly from the corresponding low-level handler in the nucleus.
  - Several arguments are passed to it, including the interrupt/trap/ exception number and the processor context of the executing thread.
  - The handler routine may take various actions, such as processing an event and/or awakening a regular thread in the actor.
  - The handler routine then returns to the nucleus.

CHORUS/MiX File Manager as a UNIX Process in User Space



Example: File Access



Performance Benefits

- Performance benefits of supervisor actors come in several areas.
  - Memory and processor context switches are minimized through use of connected handlers rather than messages, and in general through address-space sharing of actors of a common subsystem which happen to be running on a single site.
  - Trap expense can be avoided for nucleus system calls executed by supervisor actors.
  - Finally, supervisor actors allow a new level of RPC efficiency. The ``lightweight RPC'' mechanism of optimizes pure RPC for the case where client and server reside on the same site.
  - This ``featherweight'' RPC is substantially lower in overhead, while still mediated by the nucleus and still using an interface similar to that of pure RPC.

Performance Lessons

- Lesson: Implementing part of an operating system in user-level servers, while elegant, imposes prohibitive message passing and context switching overheads not present in a monolithic implementation of the system.
- To allow microkernel technology to compete in the marketplace, it was necessary to provide a solution to these problems.
- Supervisor actors provide the advantages of a modular system while minimally sacrificing performance.

Naming and Protection

- Basic protection for port/actor names is threefold:

1. All messages are stamped by the nucleus with the sending port's unique identifier as well as its protection identifier. Protection identifiers allow the source of a message to be reliably identified as they may be modified only by trusted actors.
2. Global names are randomly generated in a large, sparse name space; knowing a valid global name does not help much in finding other valid names.
3. Objects within CHORUS may be named using capabilities which consist of a <name, key> tuple. Capabilities are constructed using whatever techniques are deemed appropriate by the server that provides them, and may incorporate protection schemes.

Protection (contd.)

- Port groups, as implemented by the nucleus, have keys related to the group name by means of a non-invertible function.
  - Knowledge of the group name conveys the right to send messages to the group, but knowledge of the key is required to insert or delete members from the group.
- Higher degrees of port and/or message security can be implemented by individual subsystems, as required.
  - Subsystems may act as intermediaries in message communications to provide protection, or may choose to completely exclude CHORUS IPC from the set of abstractions they export to user tasks.

Conclusions

- The challenge in designing CHORUS V3 was to make this technology suitable for commercial systems requirements;
  - to provide performance comparable to similar monolithic systems and to provide full compatibility with these systems.
  - The second-generation microkernel design was driven by these requirements and we were forced to reconsider the role of the microkernel.
- Instead of strictly enforcing a single, rigid, system architecture, the microkernel is now comprised of a set of basic, flexible, and versatile tools.
  - Experience with CHORUS V2 taught that some functions, such as IPC management, belong within the microkernel.
  - Device drivers and support for heterogeneity, on the other hand, are best handled by separate servers and protocols.
  - Supervisor actors are crucial to both performance and binary compatibility with existing systems. A global name space is necessary to simplify the interactions between system

Conclusions (contd.)

- Decisions, such as the choice between high security and high performance, are not be enforced a priori by the microkernel.
- The CHORUS v3 microkernel has met its requirements: the CHORUS/MiX microkernel-based UNIX system provides the level of performance of real-time executives,
  - is compatible with UNIX at the binary level, and is truly modular and fully distributed. It has been adopted by a number of manufacturers for real-time and distributed commercial UNIX systems.
- Further work will concentrate on exploiting this technology to provide advanced operating system features, such as a distributed UNIX with a single system image and fault tolerance.

Literature

- A. Bricker, M. Gien, M. Guillemont, J. Lipkis, D. Orr and M. Rozier *"A new look at micro-kernel-based UNIX operating systems: Lessons in performance and compatibility"* CS-TR-91-7, In: Proc. of the EurOpen Spring'91 Conference, Tromsoe, Norway,20-24 May 1991.
- M. Gien, L. Grob. *"Micro-kernel Based Operating Systems: Moving UNIX onto Modern System Architectures"* CS-TR-91-81, In: Proc. UniForum'92 Conference, San Francisco, CA, 22-24 January 1992.
- N. Batlivala, B. Gleeson, J. Hamrick, S. Lurndal, D. Price, J. Soddy, V. Abrossimov *"Experience with SVR4 Over CHORUS",* CS-TR-92-50, in: Proc. of the Usenix Workshop on Micro-kernels and Other Kernel Architectures, Seattle, WA, April 27-28, 1992