Design philosophy of operating systems (II)

Hung-Wei Tseng

What the OS kernel should do?

Outline

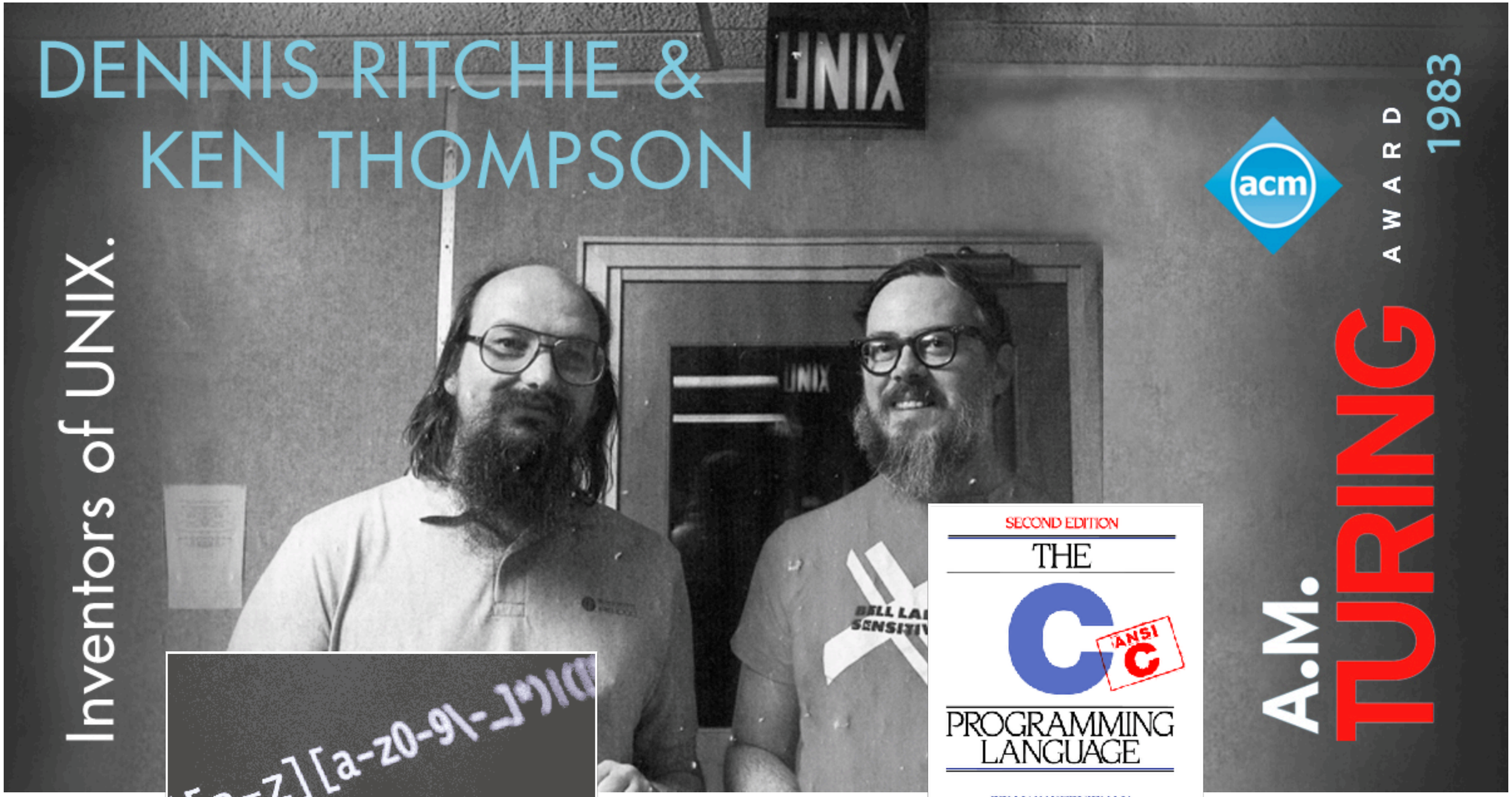
- The UNIX time-sharing operating system
- Mach: A New Kernel Foundation For UNIX Development

The UNIX Time-Sharing System

Dennis M. Ritchie and Ken Thompson
Bell Laboratories

DENNIS RITCHIE & KEN THOMPSON

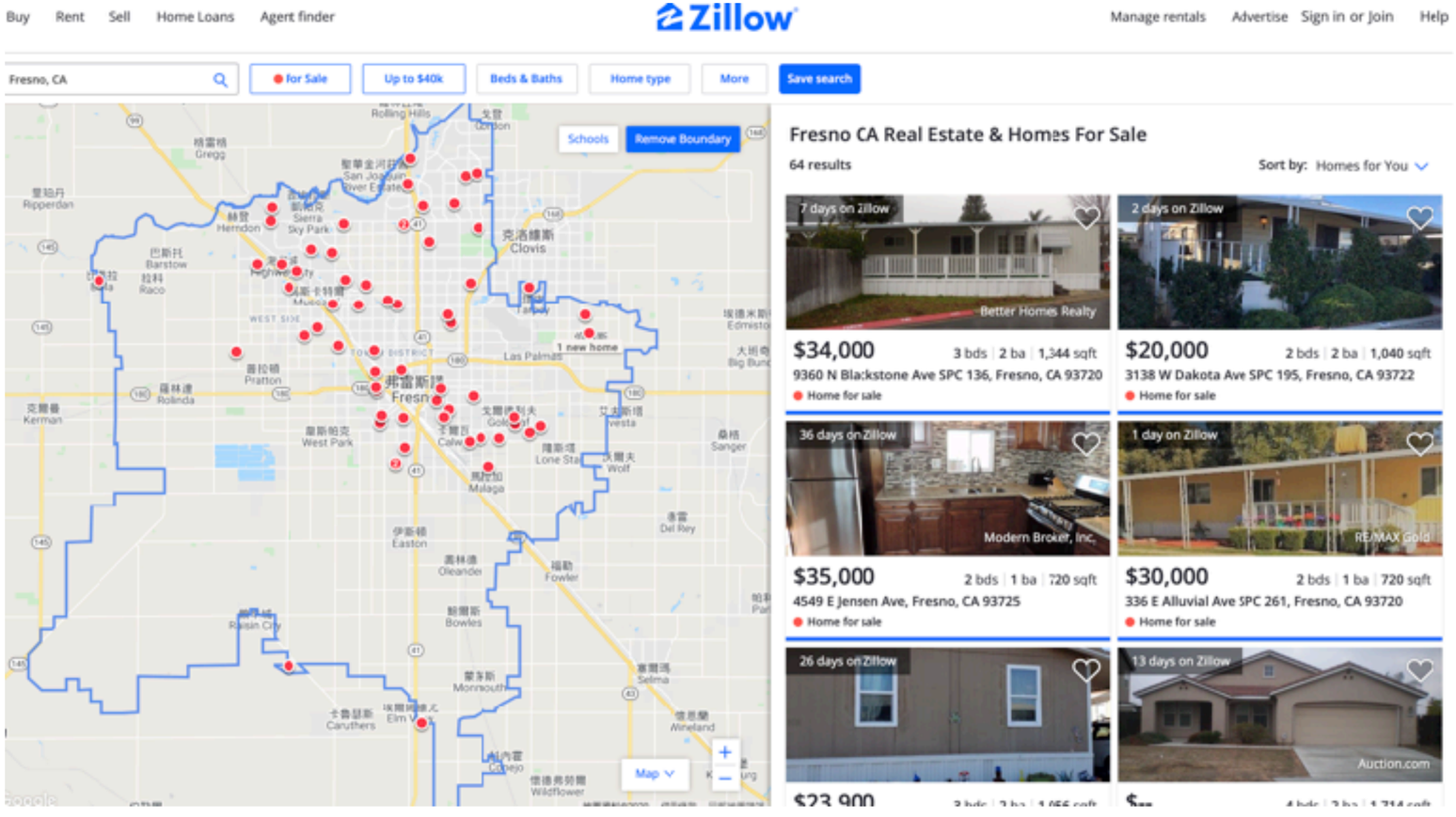
Inventors of UNIX.



A.M. TURING AWARD 1983

Why should we care about “UNIX”

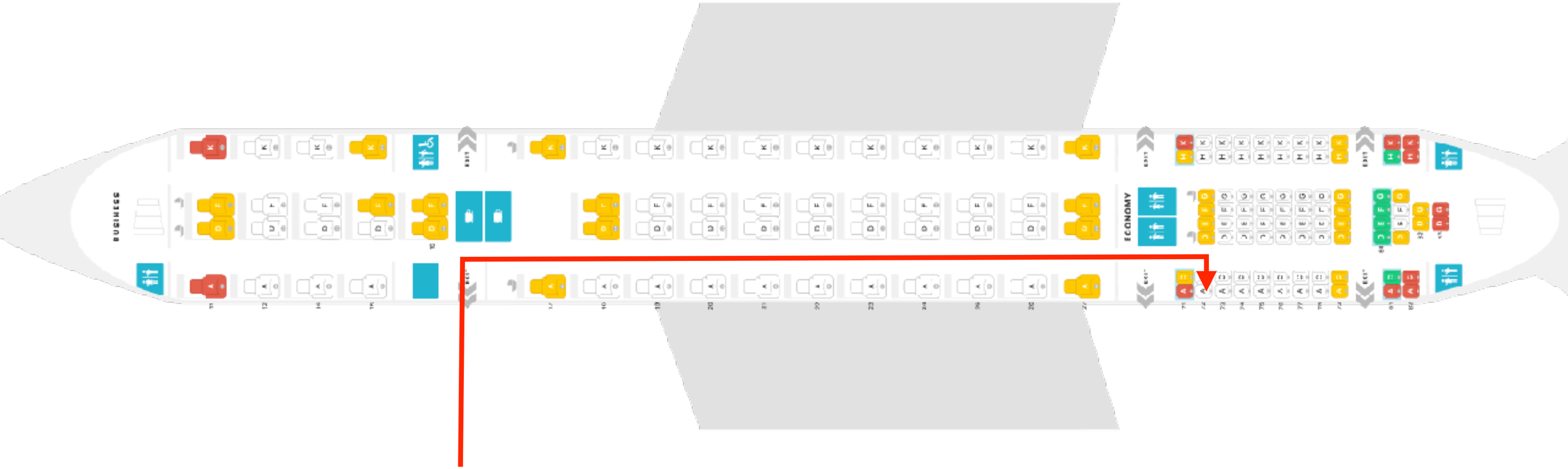
- A powerful operating system on “inexpensive” hardware (still costs USD $40,000)
- An operating system promotes simplicity, elegance, and ease of use
- They made it



What UNIX proposed

- Providing a file system
- File as the unifying abstraction in UNIX
- Remind what we mentioned before

Right amplification



Demo: setuid

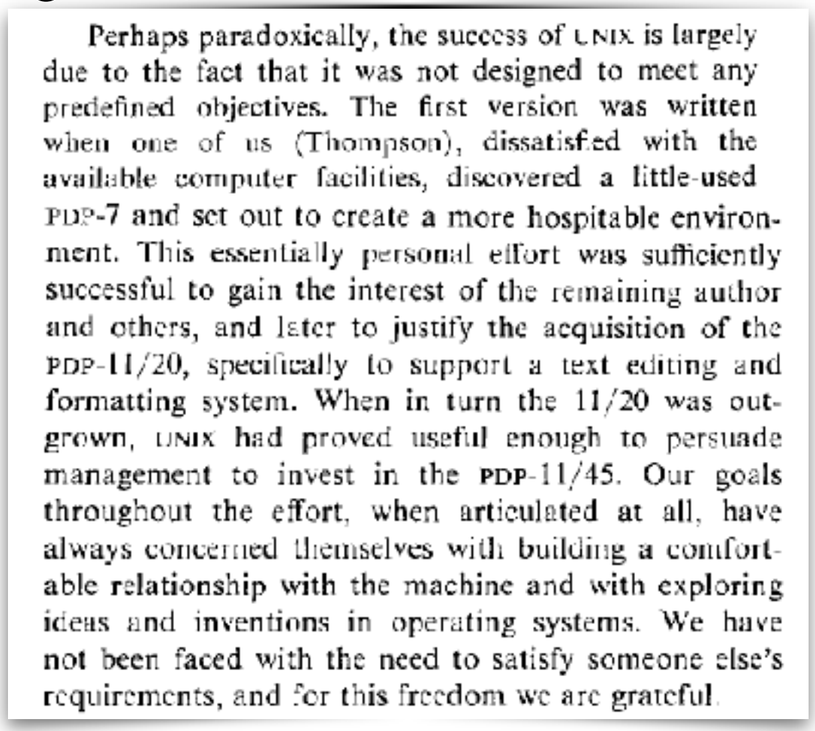
- chmod u+s allows “others” to execute the program as the creator
- There exists a file “others” cannot read
- Another program can dump the content
- Without setuid, others still cannot read the content
- With setuid, others can read that!

Shell

- A user program provides an interactive UI
- Interprets user command into OS functions
- Basic semantics:
  command argument_1 argument_2 …
- Advanced semantics
  - Redirection
    - >
    - <
  - Pipe
    - |
  - Multitasking
    - &

The impact of UNIX

- Clean abstraction
- File system — will discuss in detail after midterm
- Portable OS
  - Written in high-level C programming language
  - The unshakable position of C programming language
- We are still using it!



Perhaps paradoxically, the success of UNIX is largely due to the fact that it was not designed to meet any predefined objectives. The first version was written when one of us (Thompson), dissatisfied with the available computer facilities, discovered a little-used PDP-7 and set out to create a more hospitable environment. This essentially personal effort was sufficiently successful to gain the interest of the remaining author and others, and later to justify the acquisition of the PDP-11/20, specifically to support a text editing and formatting system. When in turn the 11/20 was outgrown, UNIX had proved useful enough to persuade management to invest in the PDP-11/45. Our goals throughout the effort, when articulated at all, have always concerned themselves with building a comfortable relationship with the machine and with exploring ideas and inventions in operating systems. We have not been faced with the need to satisfy someone else's requirements, and for this freedom we are grateful.

Mach: A New Kernel Foundation For UNIX Development

Mike Accetta , Robert Baron , William Bolosky , David Golub , Richard Rashid , Avadis Tevanian , Michael Young
Computer Science Department, Carnegie Mellon University

Why "Mach"?

- The hardware is changing
  - Multiprocessors
  - Networked computing
- The software
  - The demand of extending an OS easily
  - Repetitive but confusing mechanisms for similar stuffs

> be built and future development of UNIX-like systems for new architectures can continue. The computing environment for which Mach is targeted spans a wide class of systems, providing basic support for large, general purpose multiprocessors, smaller multiprocessor networks and individual workstations (see

> As the complexity of distributed environments and multiprocessor architectures increases, it becomes increasingly important to return to the original UNIX model of consistent interfaces to system facilities. Moreover, there is a clear need to allow the underlying system to be transparently extended to allow user-state processes to provide services which in the past could only be fully integrated into UNIX by adding code to the operating system kernel.

Make UNIX great again!

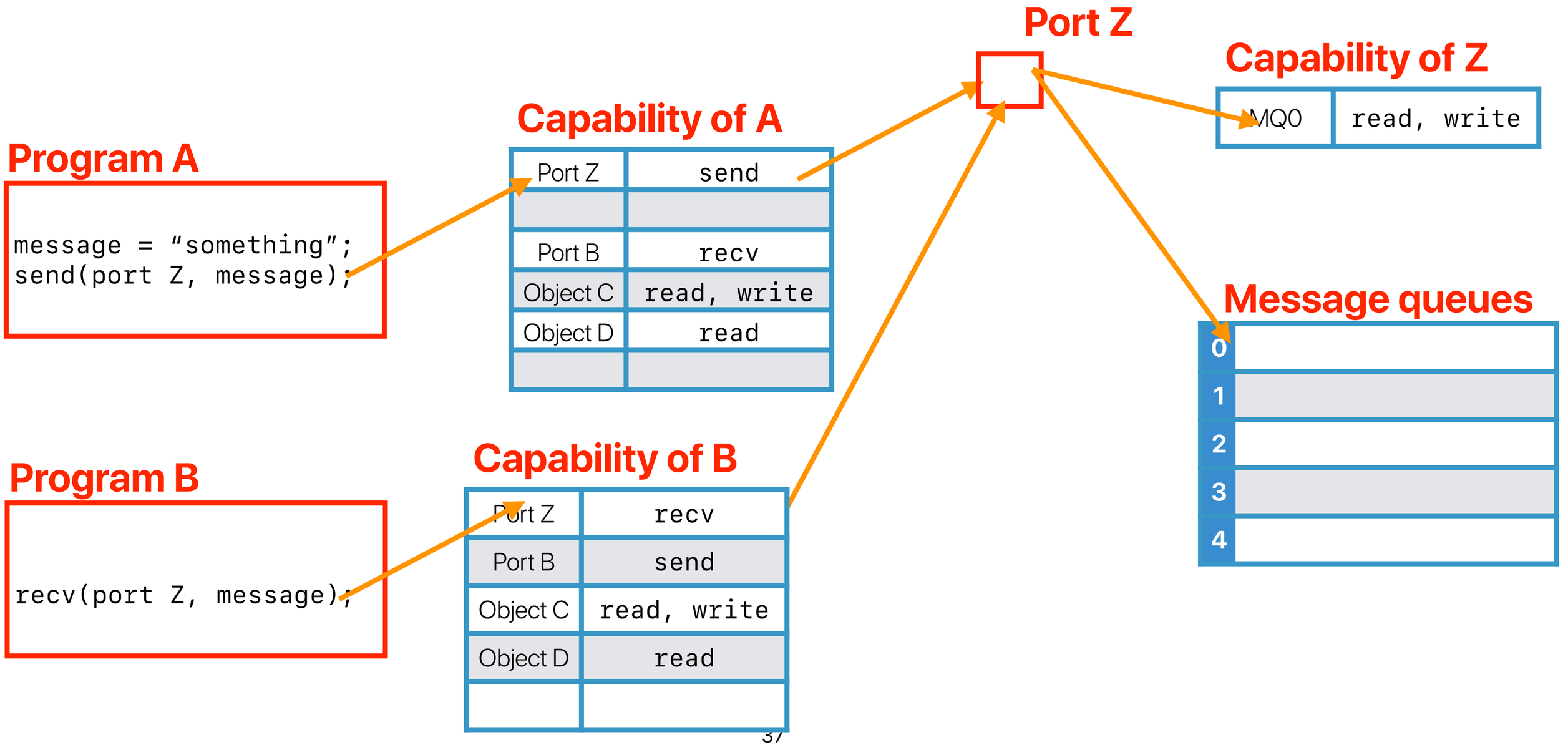
Interprocess communication

- UNIX provides a variety of mechanisms
  - Pipes
  - Pty's
  - Signals
  - Sockets
- No protection
- No consistency
- Location dependent

Ports/Messages

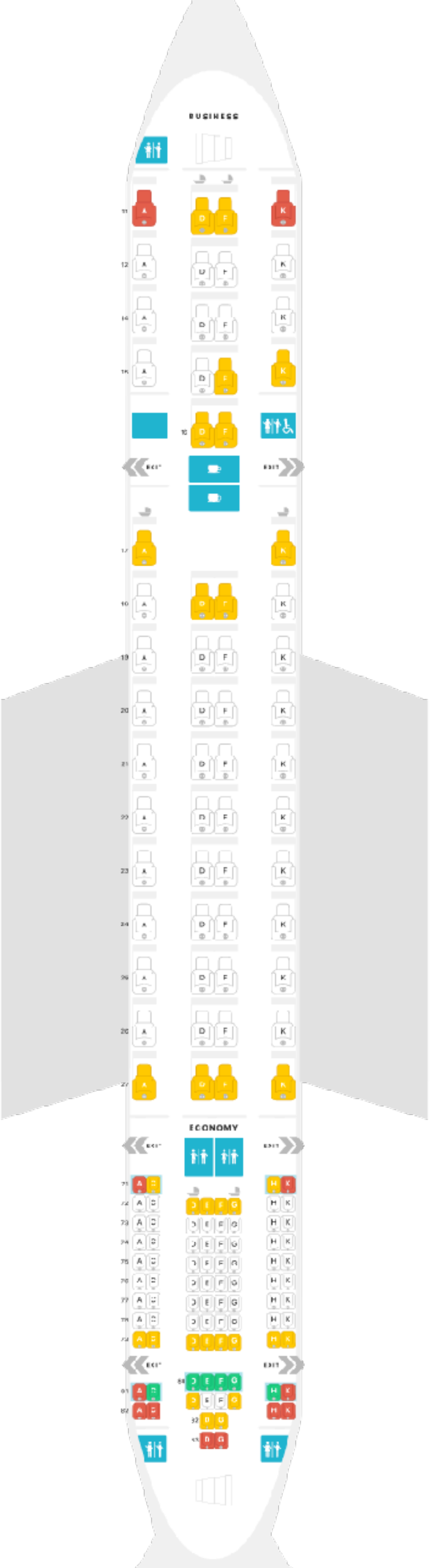
- Port is an abstraction of:
  - Message queues
  - Capability
- What do ports/messages promote?
  - Location independence — everything is communicating with ports/messages, no matter where it is

Ports/Messages



What is capability? — Hydra

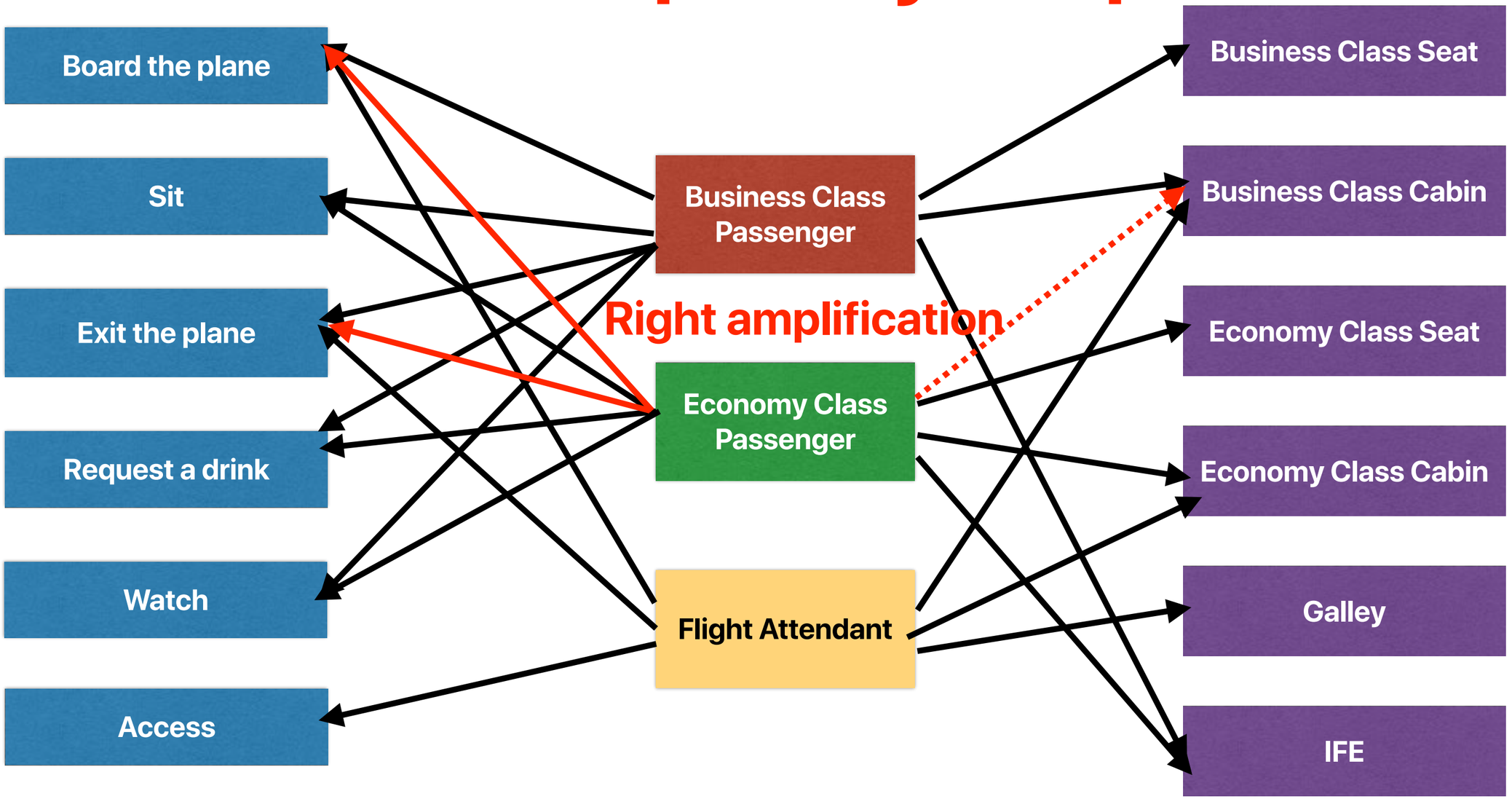
- An access control list associated with an object
- Contains the following:
  - A reference to an object
  - A list of access rights
- Whenever an operation is attempted:
  - The requester supplies a capability of referencing the requesting object — like presenting the boarding pass
  - The OS kernel examines the access rights
    - Type-independant rights
    - Type-dependent rights

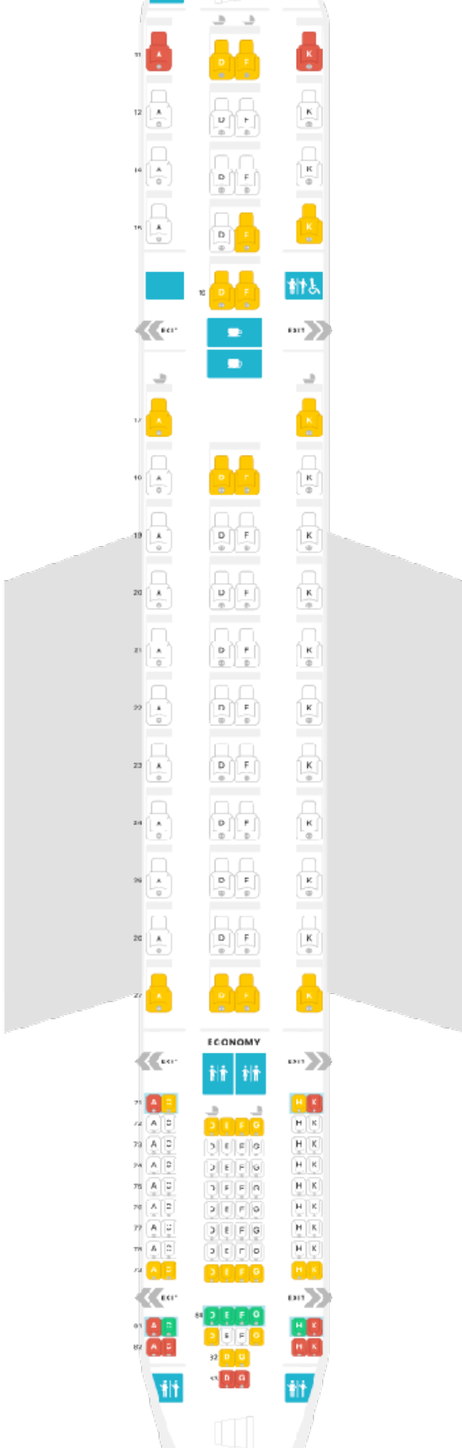

BUSINESS
ECONOMY

Capability v.s. boarding pass

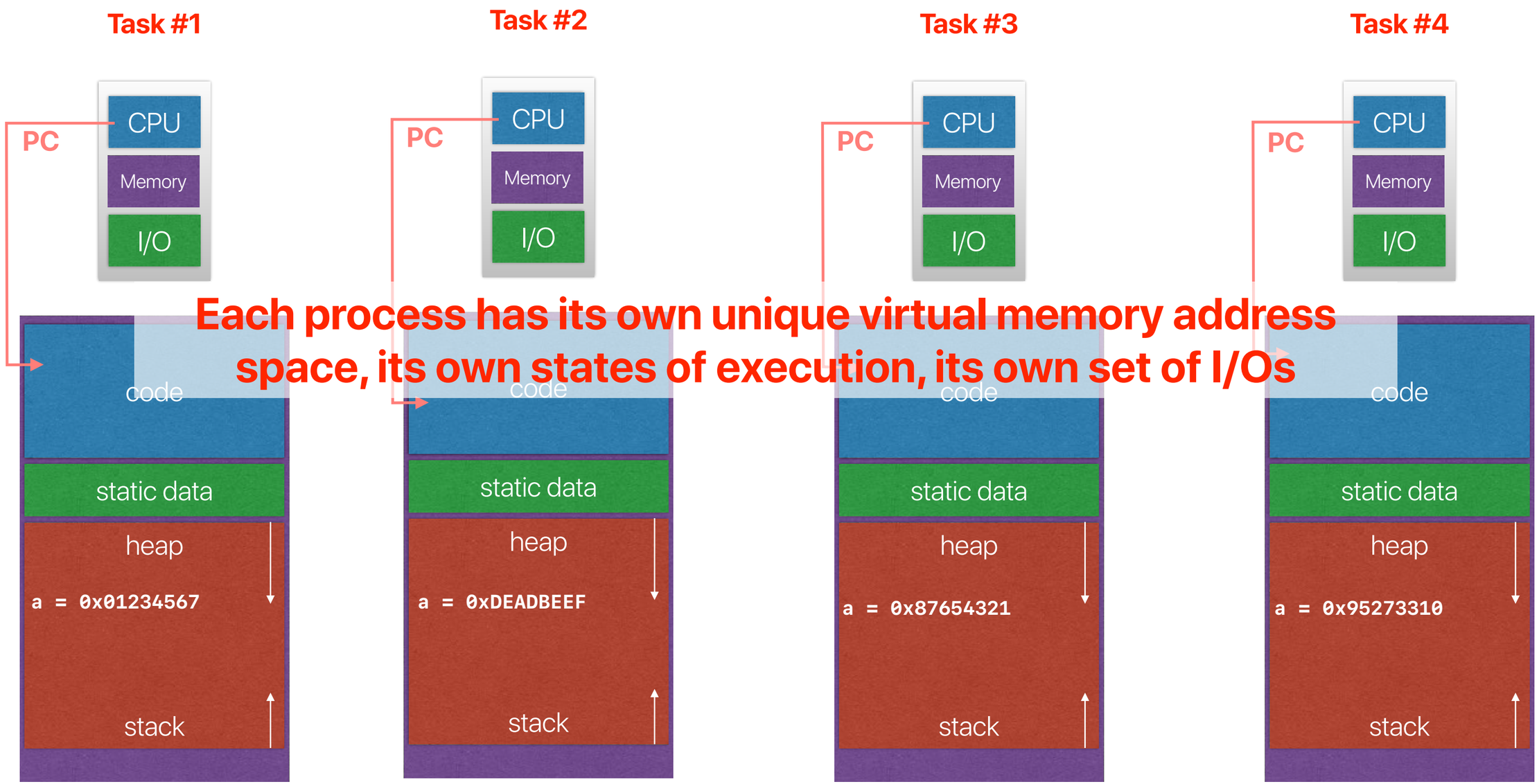
- You can only enjoy the ground services (objects) that your booking class provides
- You can only access the facilities (objects) on the airplane according to the booking class

Capability in a plane



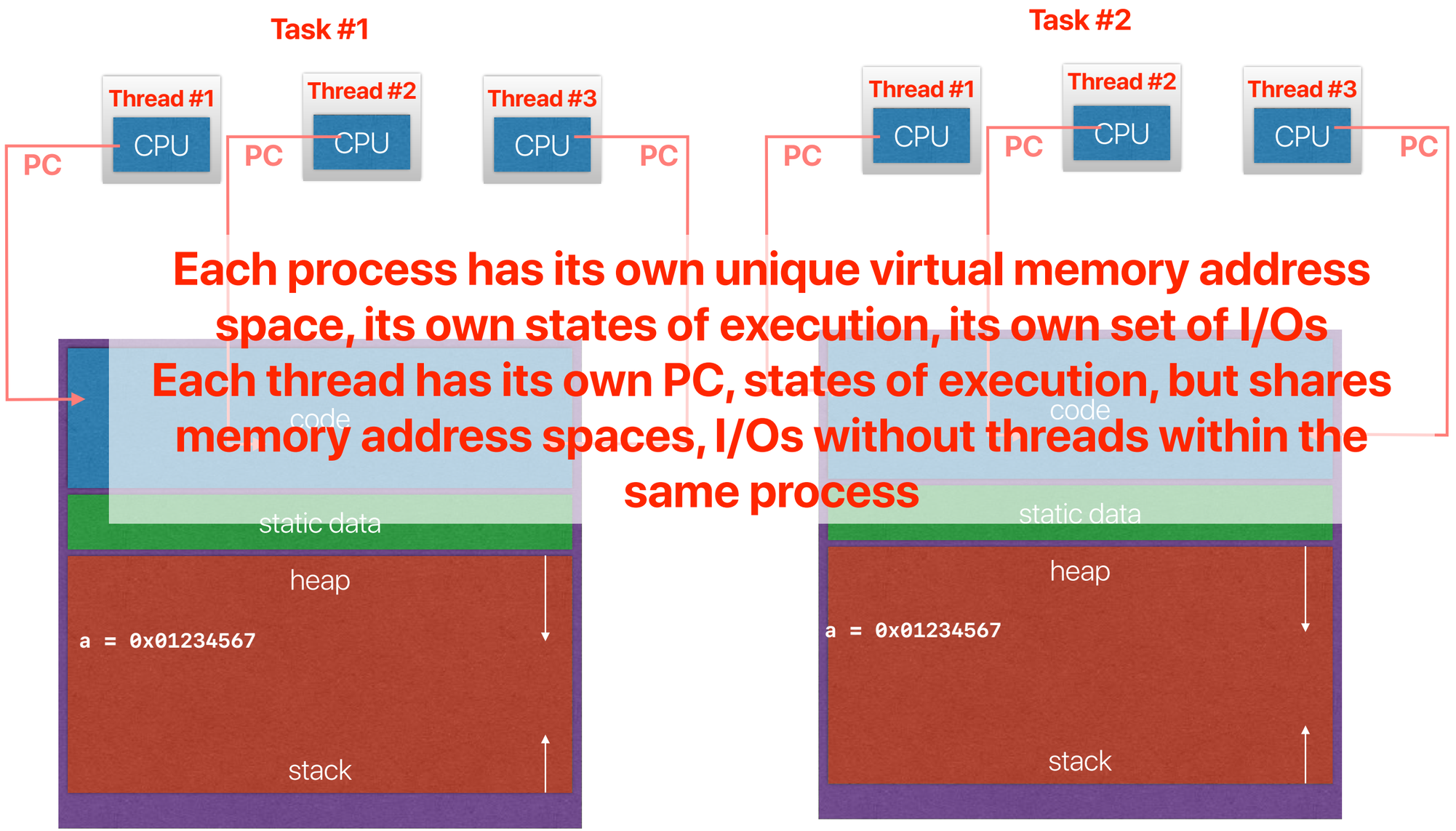


Tasks/processes



Each process has its own unique virtual memory address space, its own states of execution, its own set of I/Os

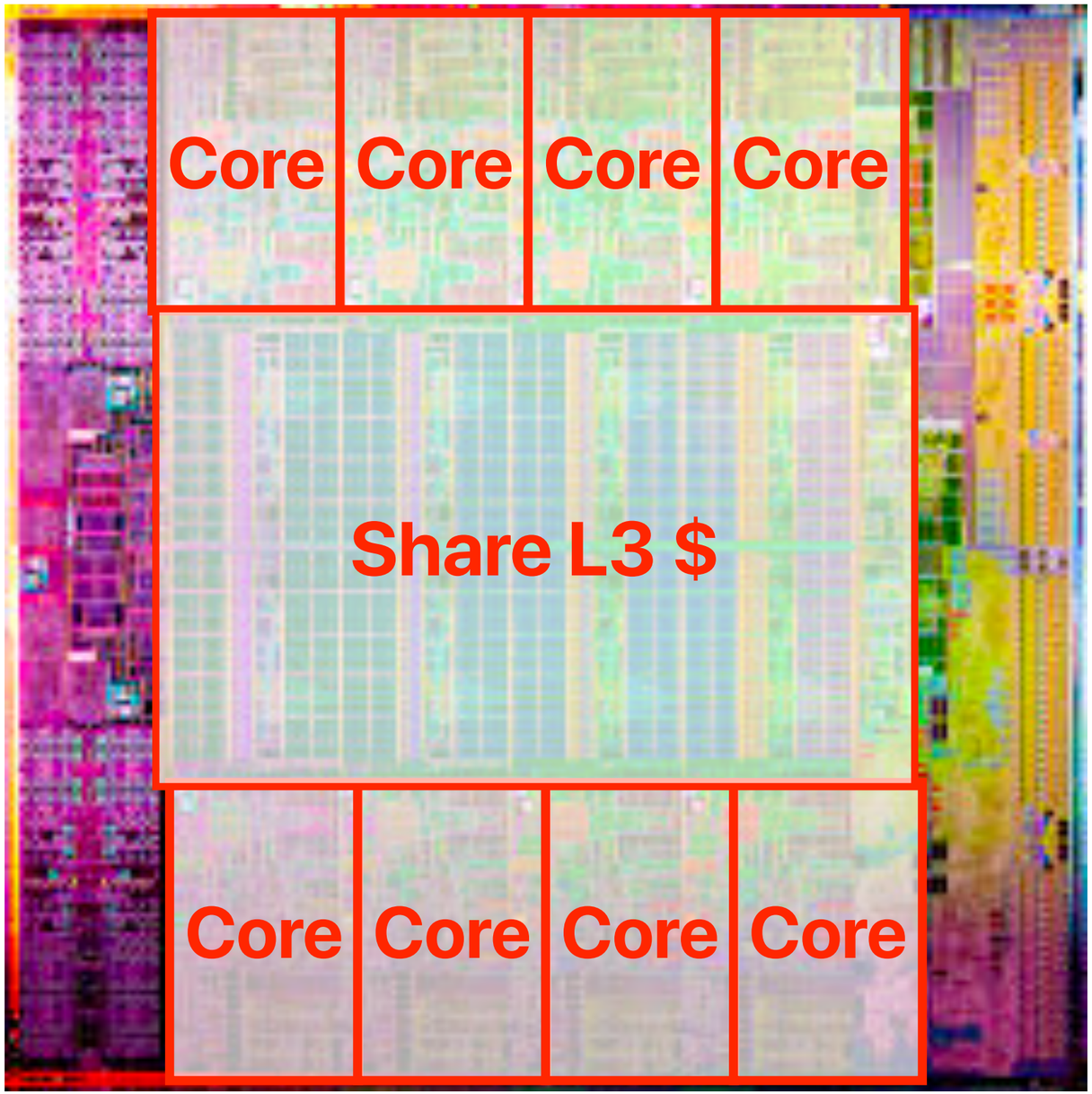
Threads



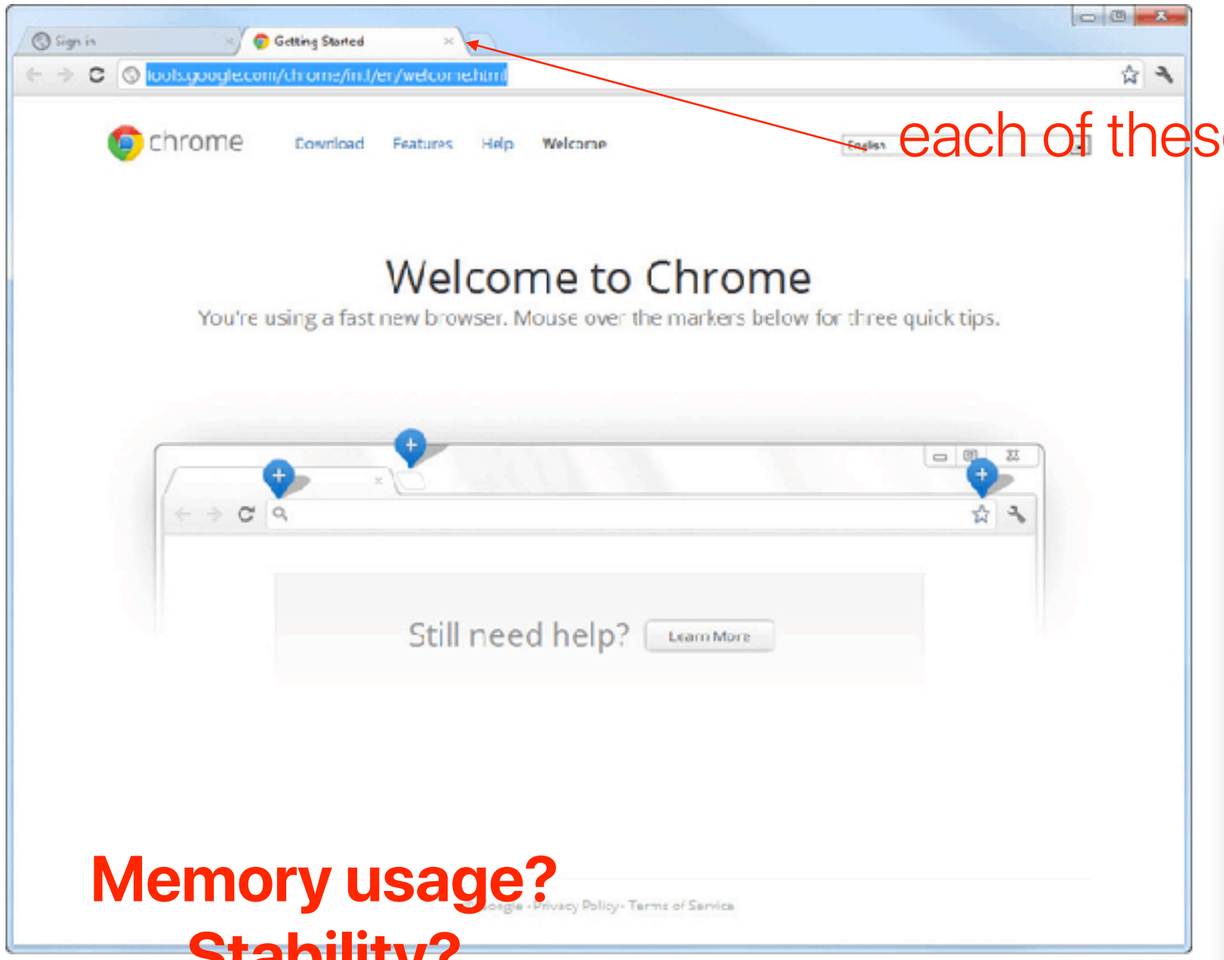
Each process has its own unique virtual memory address space, its own states of execution, its own set of I/Os

Each thread has its own PC, states of execution, but shares memory address spaces, I/Os without threads within the same process

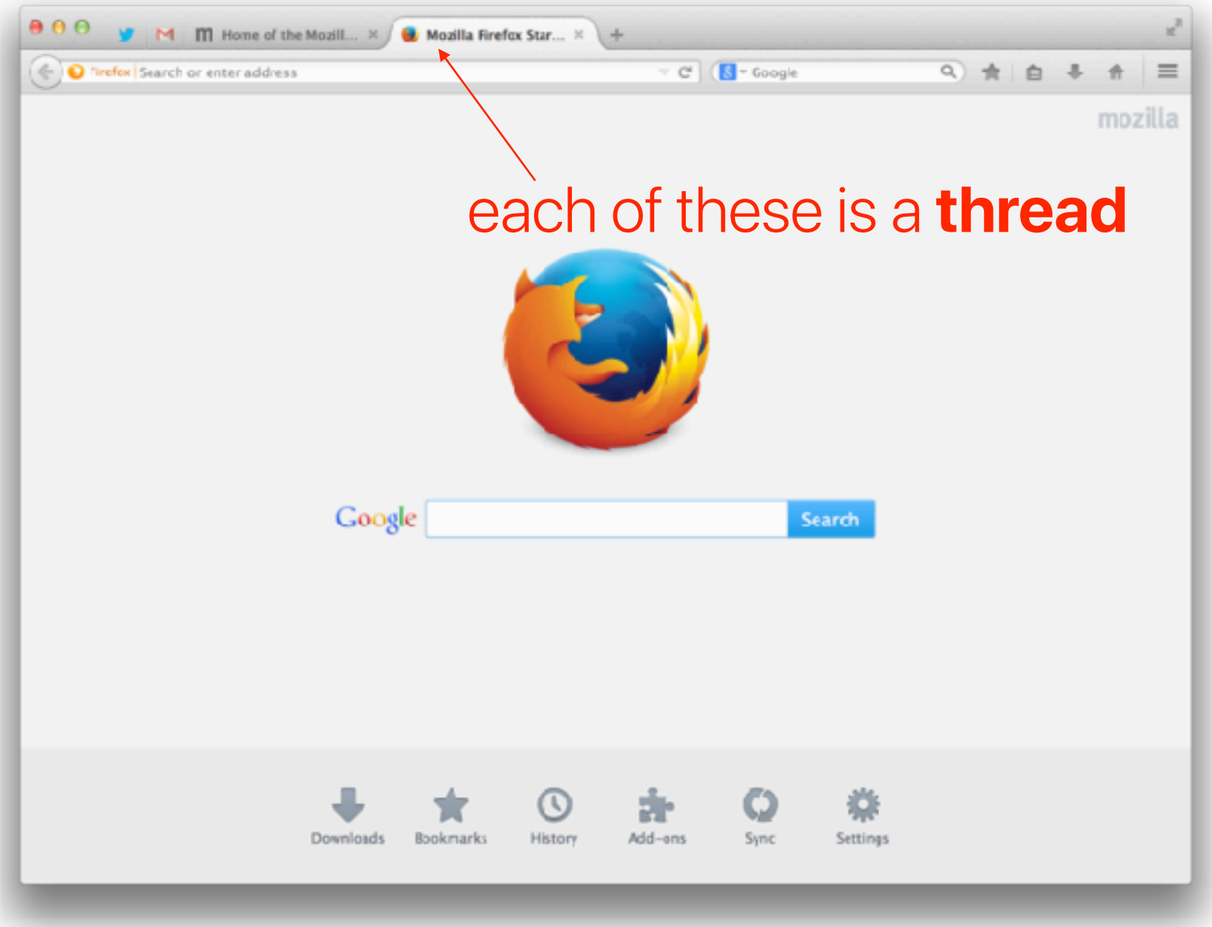
Intel Sandy Bridge



Case study: Chrome v.s. Firefox



each of these is a **process**



each of these is a **thread**

Memory usage?

Stability?

Security?

Latency?

The cost of creating processes

- Measure process creation overhead using lmbench http://www.bitmover.com/lmbench/

The cost of creating processes

- Measure process creation overhead using lmbench http://www.bitmover.com/lmbench/
- On a 3.2GHz intel Core i5-6500 Processor
  - Process fork+exit: 53.5437 microseconds
  - More than 16K cycles

Whys v.s. whats

- How many pairs of the “why” and the “what” in Mach are correct?

	Why	What
(1)	Support for multiprocessors	**Threads**
(2)	Networked computing	**Messages/Ports**
(3)	OS Extensibility	**Microkernel/Object-oriented design**
(4)	Repetitive but confusing mechanisms	**Messages/Ports**

A. 0
B. 1
C. 2
D. 3
E. 4

The impact of Mach

- Threads
- Extensible operating system kernel design
- Strongly influenced modern operating systems
  - Windows NT/2000/XP/7/8/10
  - MacOS

Mach Overview

The fundamental services and primitives of the OS X kernel are based on Mach 3.0. Apple has modified and extended Mach to better meet OS X functional and p

Mach 3.0 was originally conceived as a simple, extensible, communications microkernel. It is capable of running as a stand-alone kernel, with other traditional o networking stacks running as user-mode servers.

However, in OS X, Mach is linked with other kernel components into a single kernel address space. This is primarily for performance; it is much faster to make a messages or do remote procedure calls (*RPC*) between separate tasks. This modular structure results in a more robust and extensible system than a monolithic k microkernel.

Thus in OS X, Mach is not primarily a communication hub between clients and servers. Instead, its value consists of its abstractions, its extensibility, and its flex

- object-based APIs with communication channels (for example, ports) as object references
- highly parallel execution, including preemptively scheduled threads and support for *SMP*
- a flexible scheduling framework, with support for real-time usage
- a complete set of *IPC* primitives, including messaging, *RPC*, synchronization, and notification
- support for large virtual address spaces, shared memory regions, and memory objects backed by persistent store
- proven extensibility and portability, for example across instruction set architectures and in distributed environments
- security and resource management as a fundamental principle of design; all resources are virtualized

Mach Kernel Abstractions

Mach provides a small set of abstractions that have been designed to be both simple and powerful. These are the main kernel abstractions:

- *Tasks*. The units of resource ownership; each task consists of a virtual address space, a *port right namespace*, and one or more *threads*. (Similar to a process.
- *Threads*. The units of CPU execution within a task.
- *Address space*. In conjunction with memory managers, Mach implements the notion of a sparse virtual address space and shared memory.
- *Memory objects*. The internal units of memory management. Memory objects include named entries and regions; they are representations of potentially persi
- *Ports*. Secure, simplex communication channels, accessible only via send and receive capabilities (known as port rights).
- *IPC*. Message queues, remote procedure calls, notifications, semaphores, and lock sets.
- *Time*. Clocks, timers, and waiting.