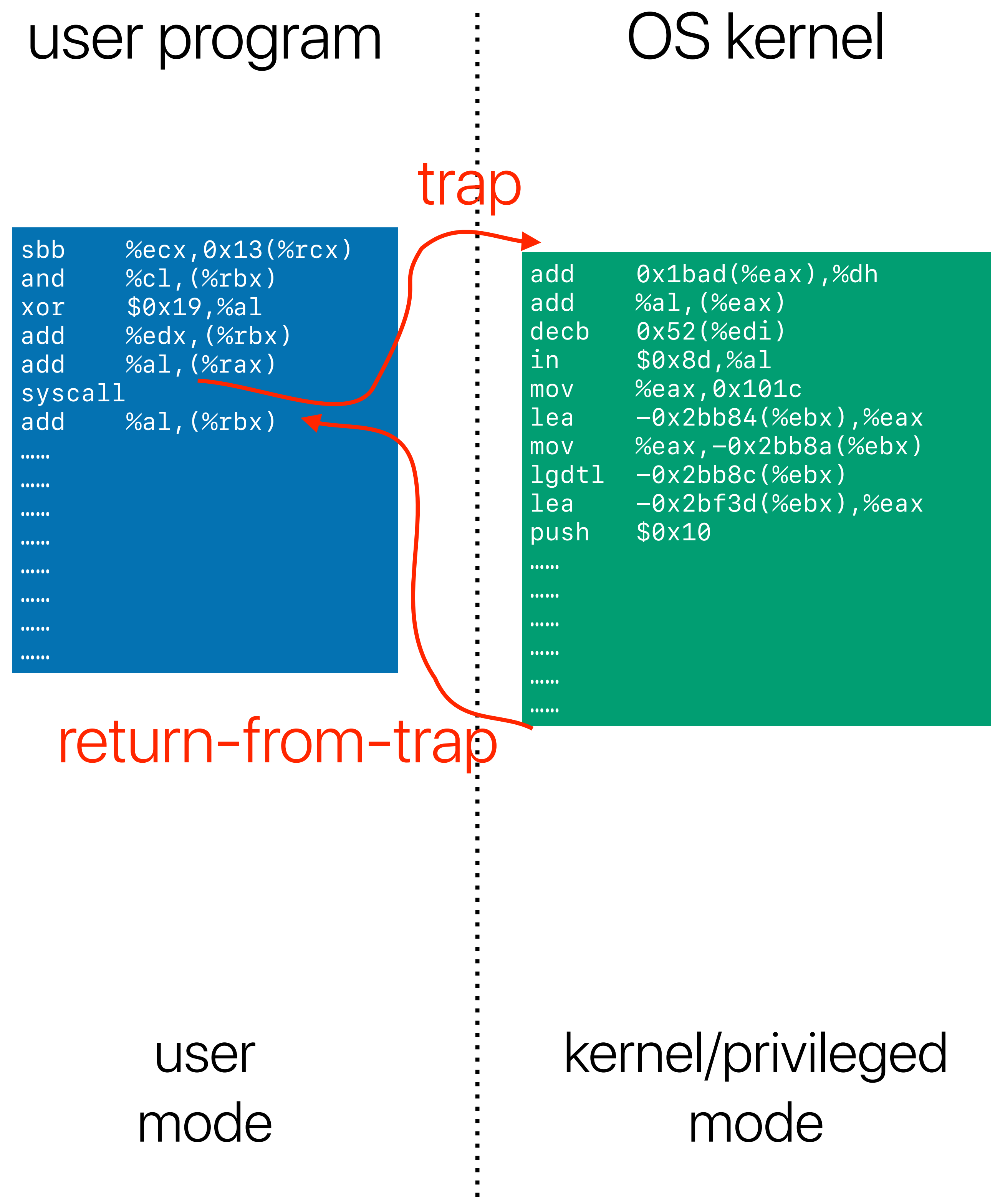
Design philosophy of operating systems (II)

Hung-Wei Tseng

Recap: How applications can use privileged operations?

- Implemented in "trap" instructions
  - Raise an exception in the processor
  - The processor saves the exception PC and jumps to the corresponding exception handler in the OS kernel
- The OS kernel only get involved when necessary
  - System calls
  - Hardware interrupts
  - Exceptions
- The OS kernel works **on behave of** the requesting process — not a process
  - Somehow like a function call to a dynamic linking library
  - As a result — overhead of copying registers, allocating local variables for kernel code and etc…



```
sbb     %ecx,0x13(%rcx)
and     %cl,(%rbx)
xor     $0x19,%al
add     %edx,(%rbx)
add     %al,(%rax)
syscall
add     %al,(%rbx)
……
……
……
……
……
……
……
……
```

```
add     0x1bad(%eax),%dh
add     %al,(%eax)
decb    0x52(%edi)
in      $0x8d,%al
mov     %eax,0x101c
lea     -0x2bb84(%ebx),%eax
mov     %eax,-0x2bb8a(%ebx)
lgdtl   -0x2bb8c(%ebx)
lea     -0x2bf3d(%ebx),%eax
push    $0x10
……
……
……
……
……
……
```

Recap: THE

- Why should people care about this paper in 1968?
  - Turn-around time of **short** programs
  - Economic use of peripherals
  - Automatic control of backing storage
  - Economic use of the machine

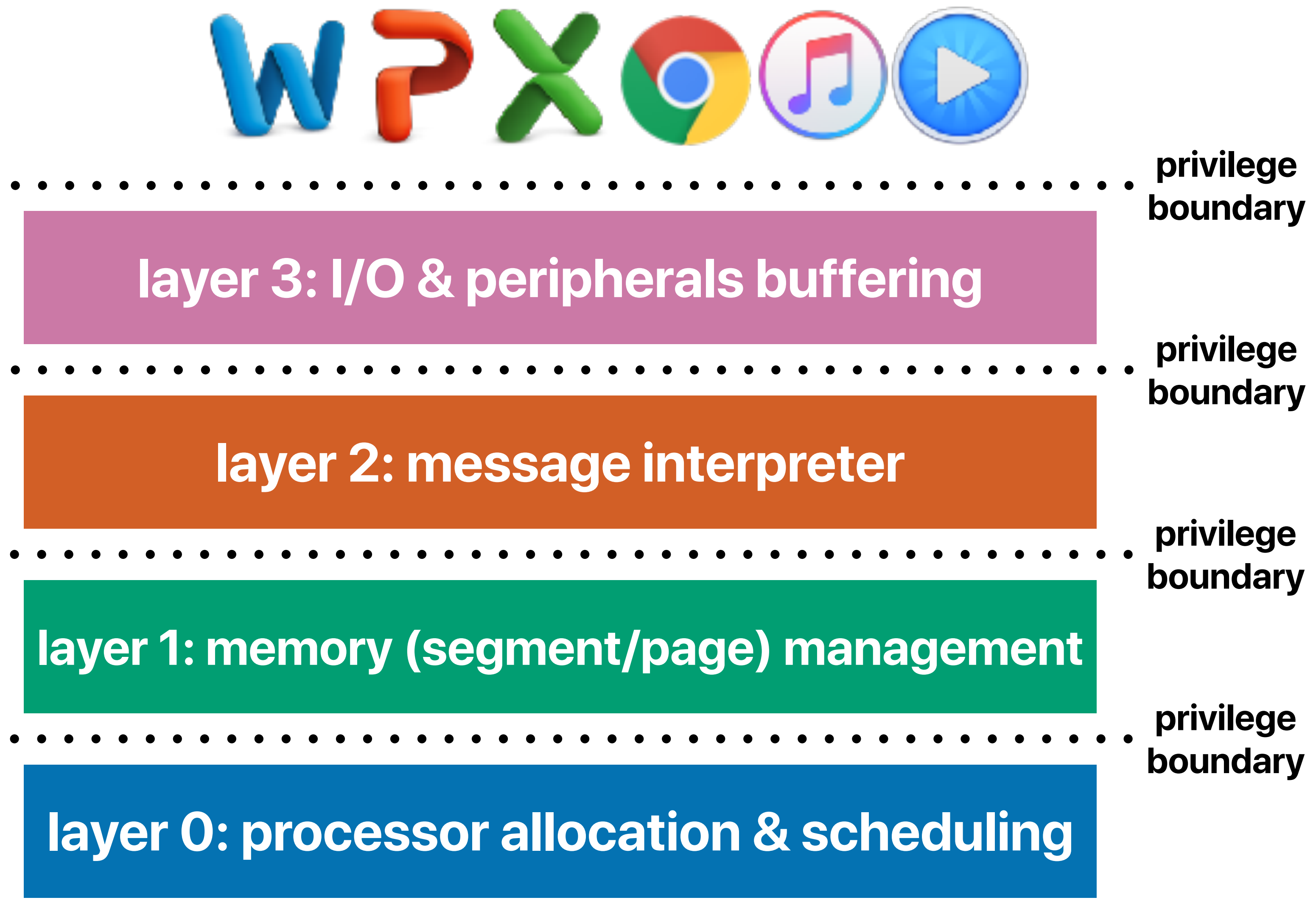
Process Abstraction
Virtual memory
Mutex

- Designing a system is difficult in 1968
  - Difficult to verify soundness
  - Difficult to prove correctness
  - Difficult to deal with the complexities

Layered Design

Recap: THE

THE


privilege boundary
layer 3: I/O & peripherals buffering
privilege boundary
layer 2: message interpreter
privilege boundary
layer 1: memory (segment/page) management
privilege boundary
layer 0: processor allocation & scheduling

The overhead of kernel switches/system calls

- On a 3.7GHz intel Core i5-9600K Processor, please make a guess of the overhead of switching from user-mode to kernel mode.
  A. a single digit of nanoseconds
  B. tens of nanoseconds
  [✓ C. hundreds of nanoseconds]
  D. a single digit of microseconds
  E. tens of microseconds

Operations	Latency (ns)
L1 cache reference	1 ns
Branch mispredict	3 ns
L2 cache reference	4 ns
Mutex lock/unlock	17 ns
Send 2K bytes over network	44 ns
Main memory reference	100 ns
Read 1 MB sequentially from memory	3,000 ns
Compress 1K bytes with Zippy	2,000 ns
Read 4K randomly from SSD*	16,000 ns
Read 1 MB sequentially from SSD*	49,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially from disk	825,000 ns
Disk seek	2,000,000 ns
Send packet CA-Netherlands-CA	150,000,000 ns

Recap: Why layered/hierarchical design?

- How many the following is/are true regarding the proposed hierarchical design
  1. The hierarchical design facilitates debugging
  2. The hierarchical design makes verification of system components easier
  3. ✗ The hierarchical design reduces the overhead of running a single process — function calls/syscalls, memory copying, and etc…
  4. ✗ The hierarchical design allows flexible resource allocation — what a potential problem is this?

  A. 0

  B. 1

  [✓ C. 2]

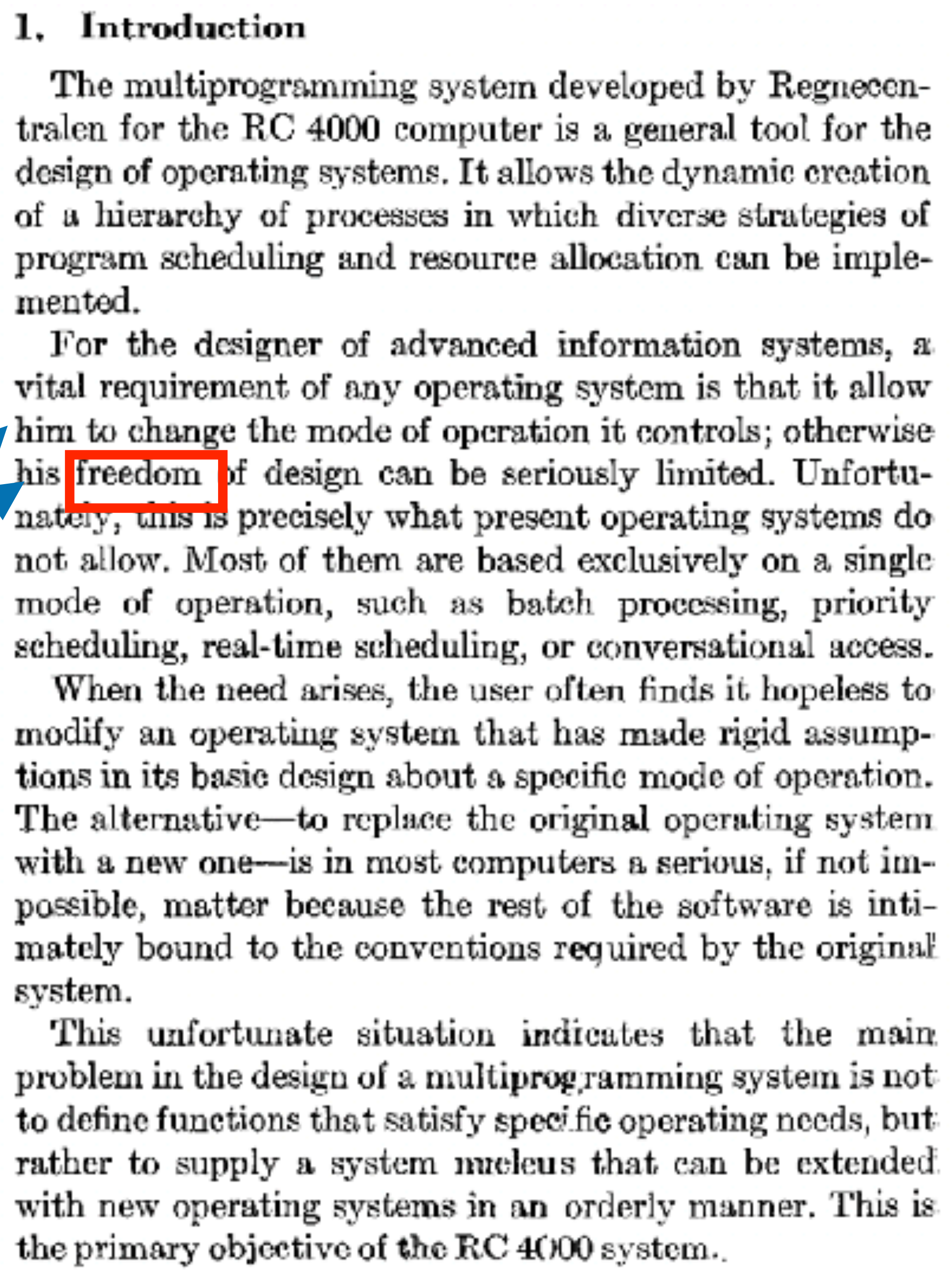
  D. 3

  E. 4

Recap: Why "Nucleus"

- Which of the following words best described the why of "The Nucleus of a Multiprogramming System"
  - A. Feasibility
  - B. Performance
  - [✓ C. Freedom]
  - D. Hierarchy
  - E. Robustness

avoid this kind of pronoun



1. Introduction

The multiprogramming system developed by Regnecentralen for the RC 4000 computer is a general tool for the design of operating systems. It allows the dynamic creation of a hierarchy of processes in which diverse strategies of program scheduling and resource allocation can be implemented.

For the designer of advanced information systems, a vital requirement of any operating system is that it allow him to change the mode of operation it controls; otherwise his freedom of design can be seriously limited. Unfortunately, this is precisely what present operating systems do not allow. Most of them are based exclusively on a single mode of operation, such as batch processing, priority scheduling, real-time scheduling, or conversational access.

When the need arises, the user often finds it hopeless to modify an operating system that has made rigid assumptions in its basic design about a specific mode of operation. The alternative—to replace the original operating system with a new one—is in most computers a serious, if not impossible, matter because the rest of the software is intimately bound to the conventions required by the original system.

This unfortunate situation indicates that the main problem in the design of a multiprogramming system is not to define functions that satisfy specific operating needs, but rather to supply a system nucleus that can be extended with new operating systems in an orderly manner. This is the primary objective of the RC 4000 system.

Outline

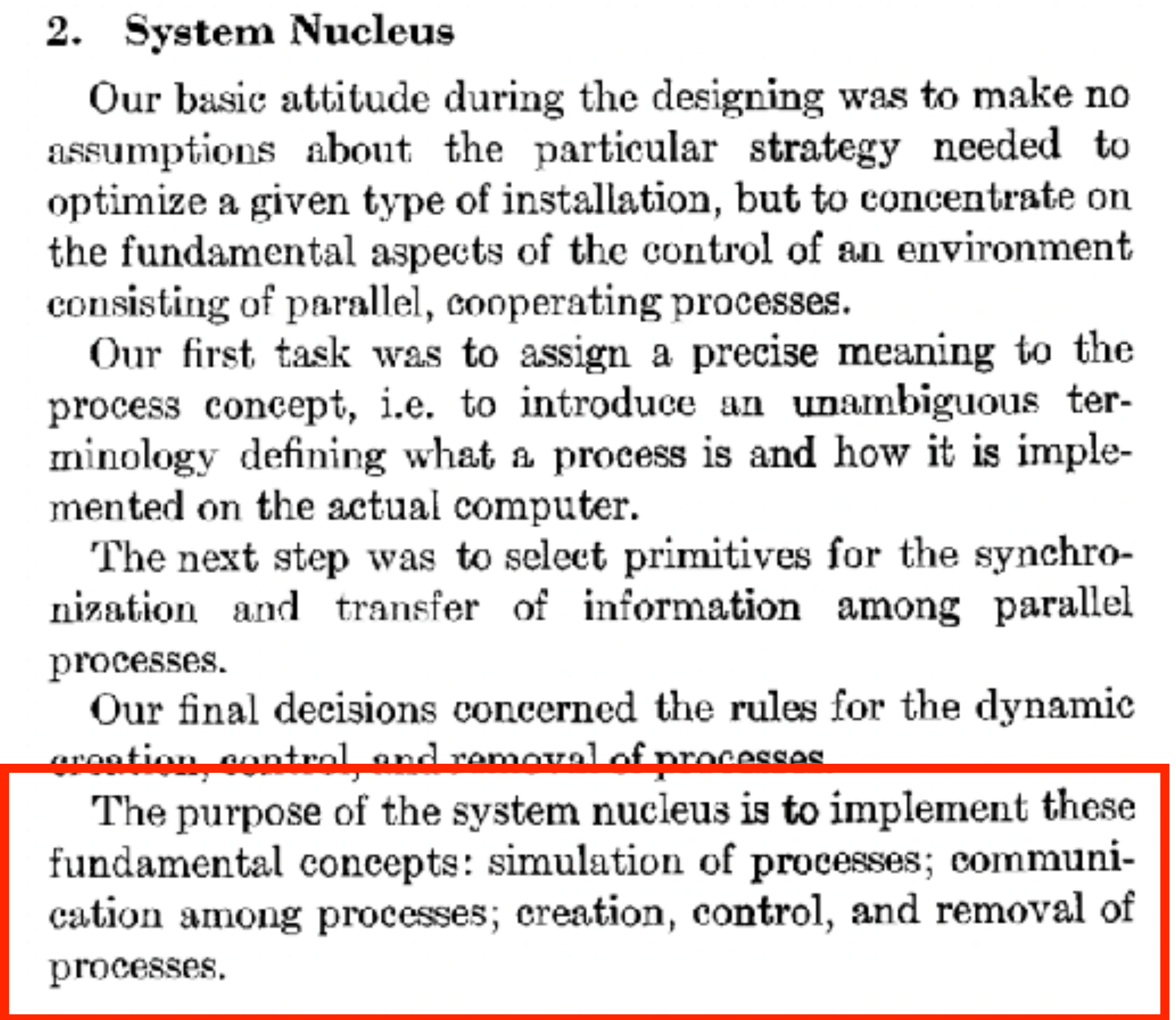
- Nucleus (cont.)
- The UNIX time-sharing operating system
- Mach: A New Kernel Foundation For UNIX Development

Poll close in 1:30

What is “system nucleus”

- Regarding “system nucleus”, how many of the following statements are correct?
  ① The system nucleus is a process
  ② The system nucleus allows multiple operating systems to execute concurrently
  ③ The system nucleus provides primitives to load and swap programs
  ④ Operating systems are user-level processes in the system nucleus architecture
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

System Nucleus ...



2. System Nucleus

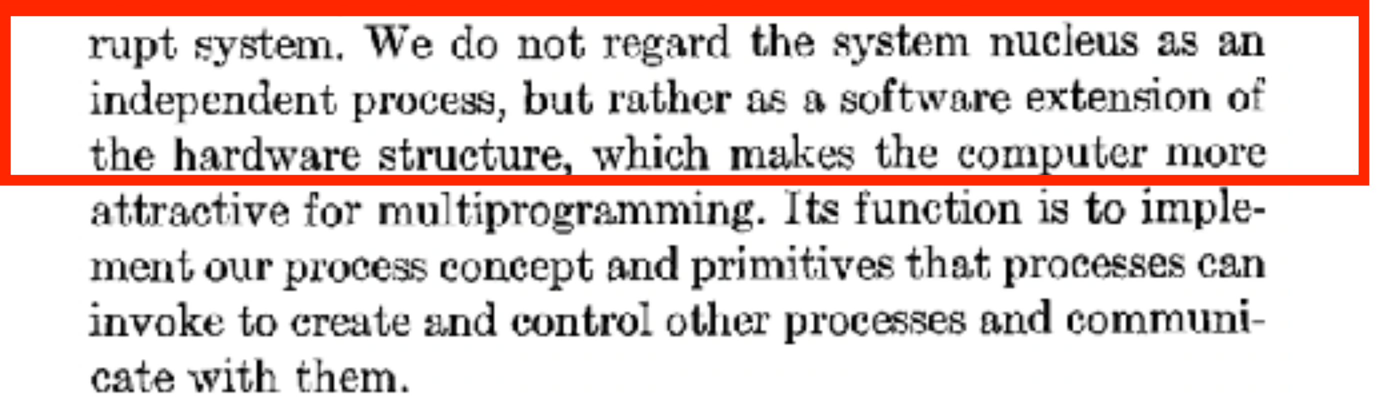
Our basic attitude during the designing was to make no assumptions about the particular strategy needed to optimize a given type of installation, but to concentrate on the fundamental aspects of the control of an environment consisting of parallel, cooperating processes.

Our first task was to assign a precise meaning to the process concept, i.e. to introduce an unambiguous terminology defining what a process is and how it is implemented on the actual computer.

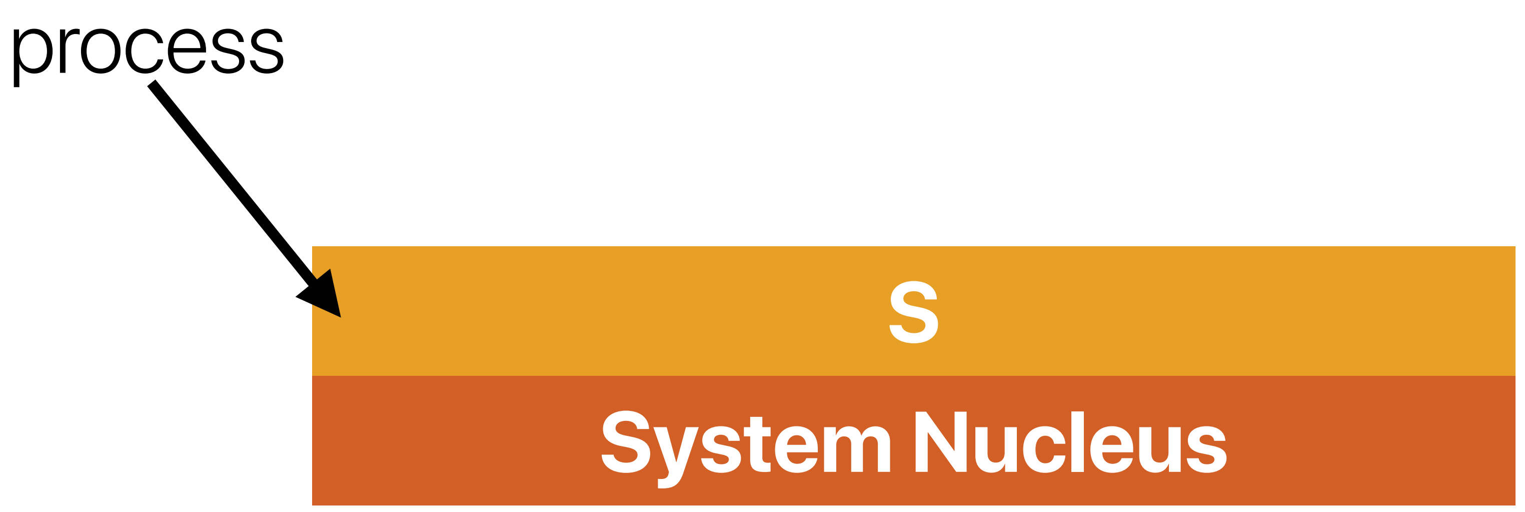
The next step was to select primitives for the synchronization and transfer of information among parallel processes.

Our final decisions concerned the rules for the dynamic creation, control, and removal of processes.

The purpose of the system nucleus is to implement these fundamental concepts: simulation of processes; communication among processes; creation, control, and removal of processes.



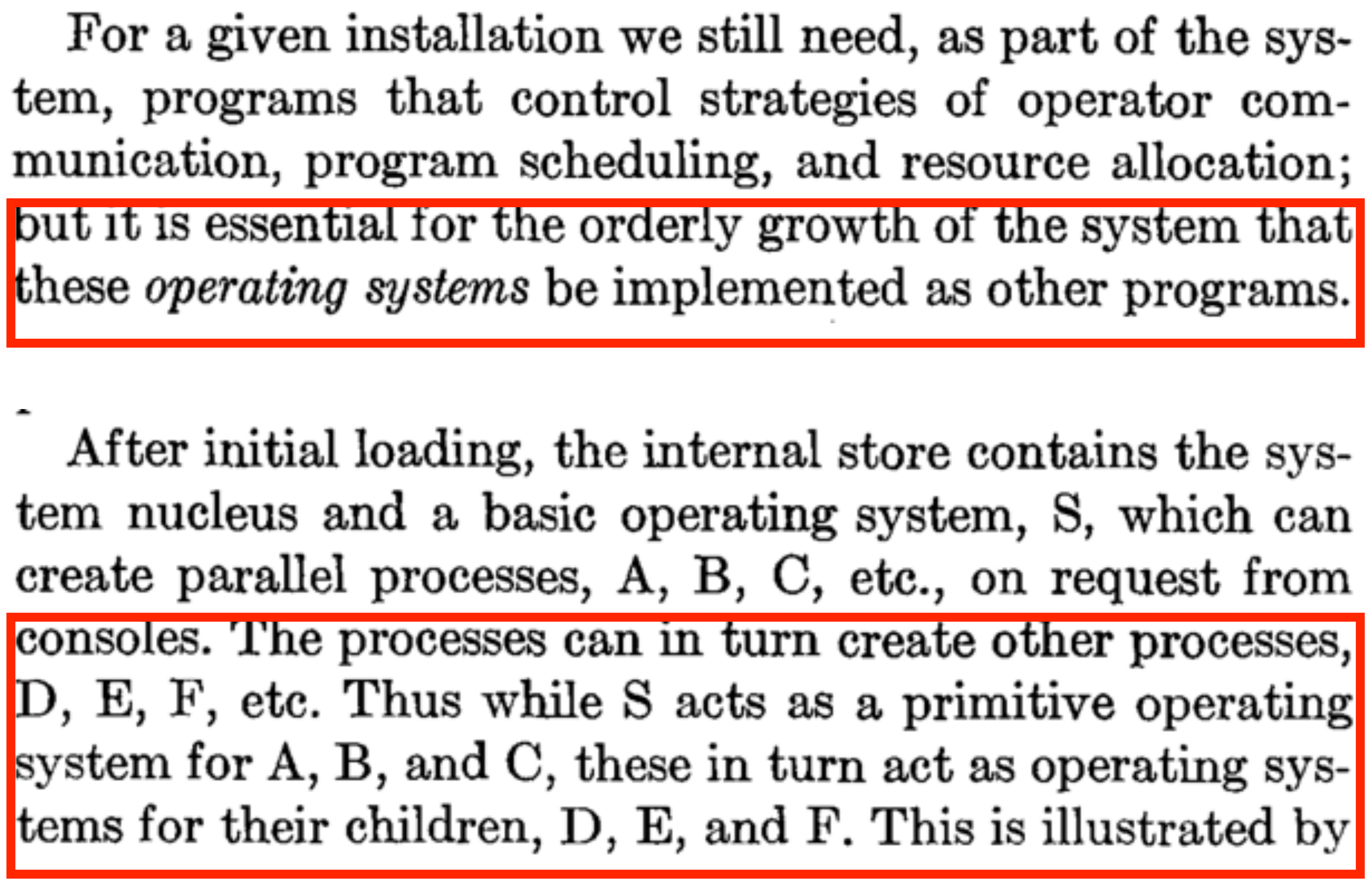
rupt system. We do not regard the system nucleus as an independent process, but rather as a software extension of the hardware structure, which makes the computer more attractive for multiprogramming. Its function is to implement our process concept and primitives that processes can invoke to create and control other processes and communicate with them.



What is “system nucleus”

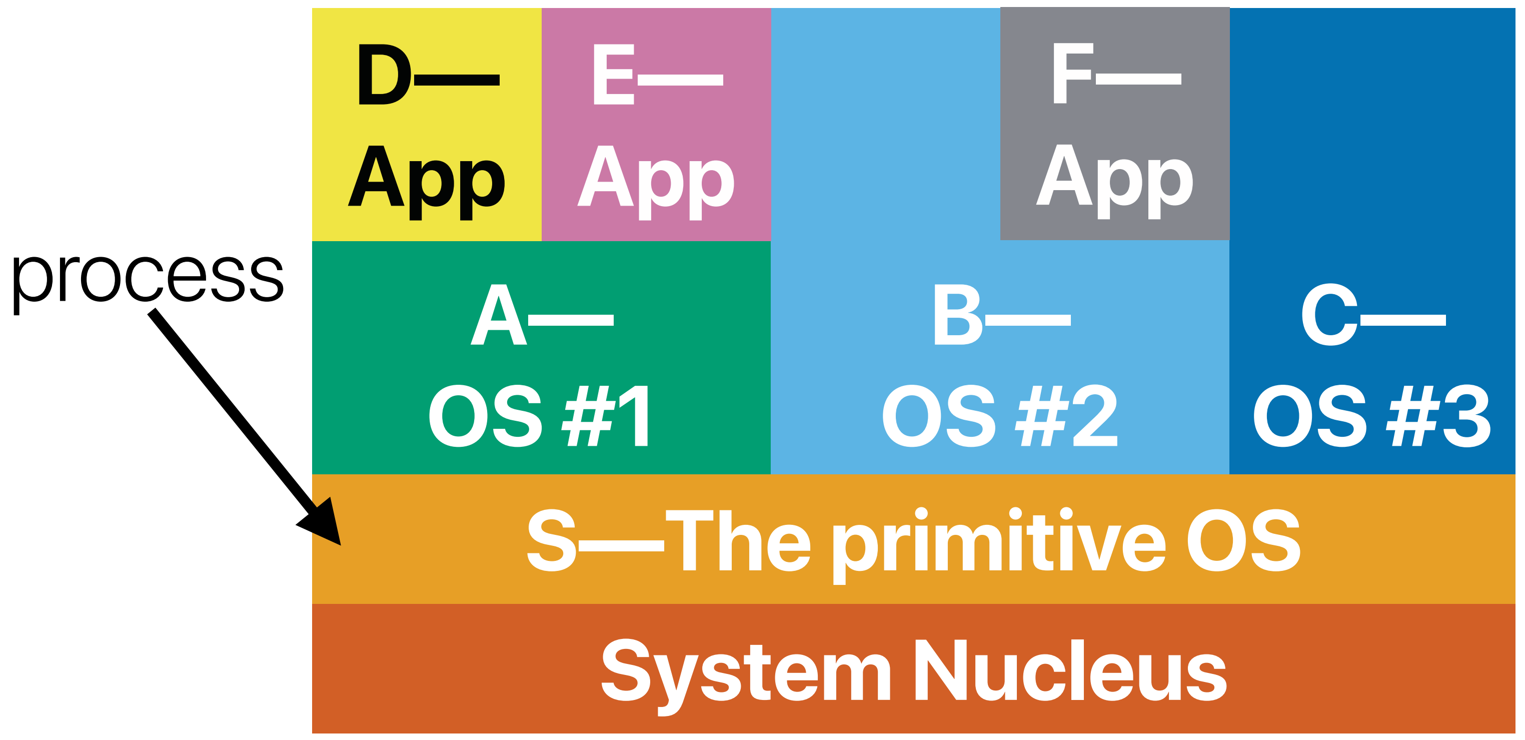
- Regarding “system nucleus”, how many of the following statements are correct?
  - ✗ The system nucleus is a process
  - ② The system nucleus allows multiple operating systems to execute concurrently
  - ③ The system nucleus provides primitives to load and swap programs
  - ④ Operating systems are user-level processes in the system nucleus architecture
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Can multiple OSs running concurrently?

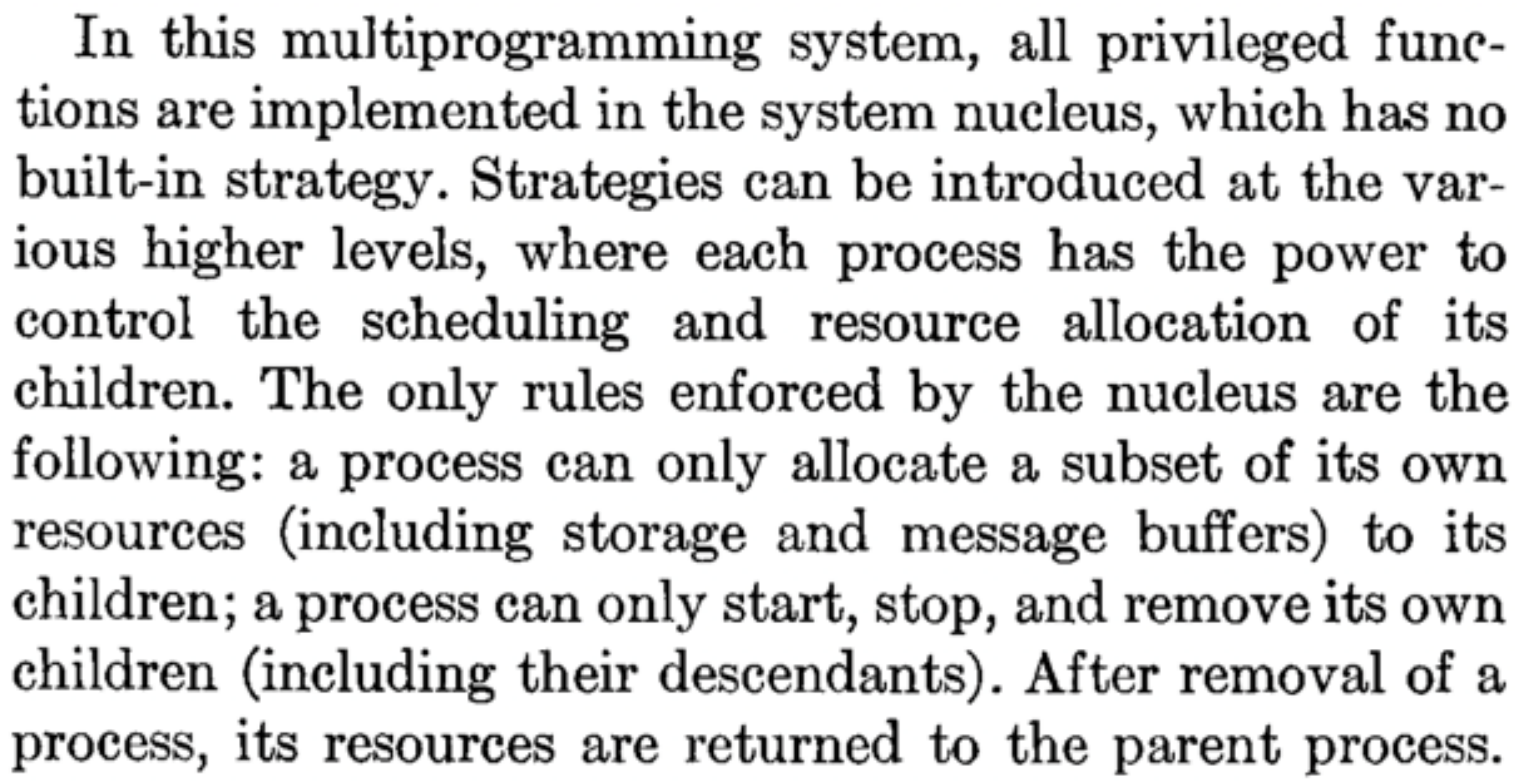


For a given installation we still need, as part of the system, programs that control strategies of operator communication, program scheduling, and resource allocation; but it is essential for the orderly growth of the system that these *operating systems* be implemented as other programs.

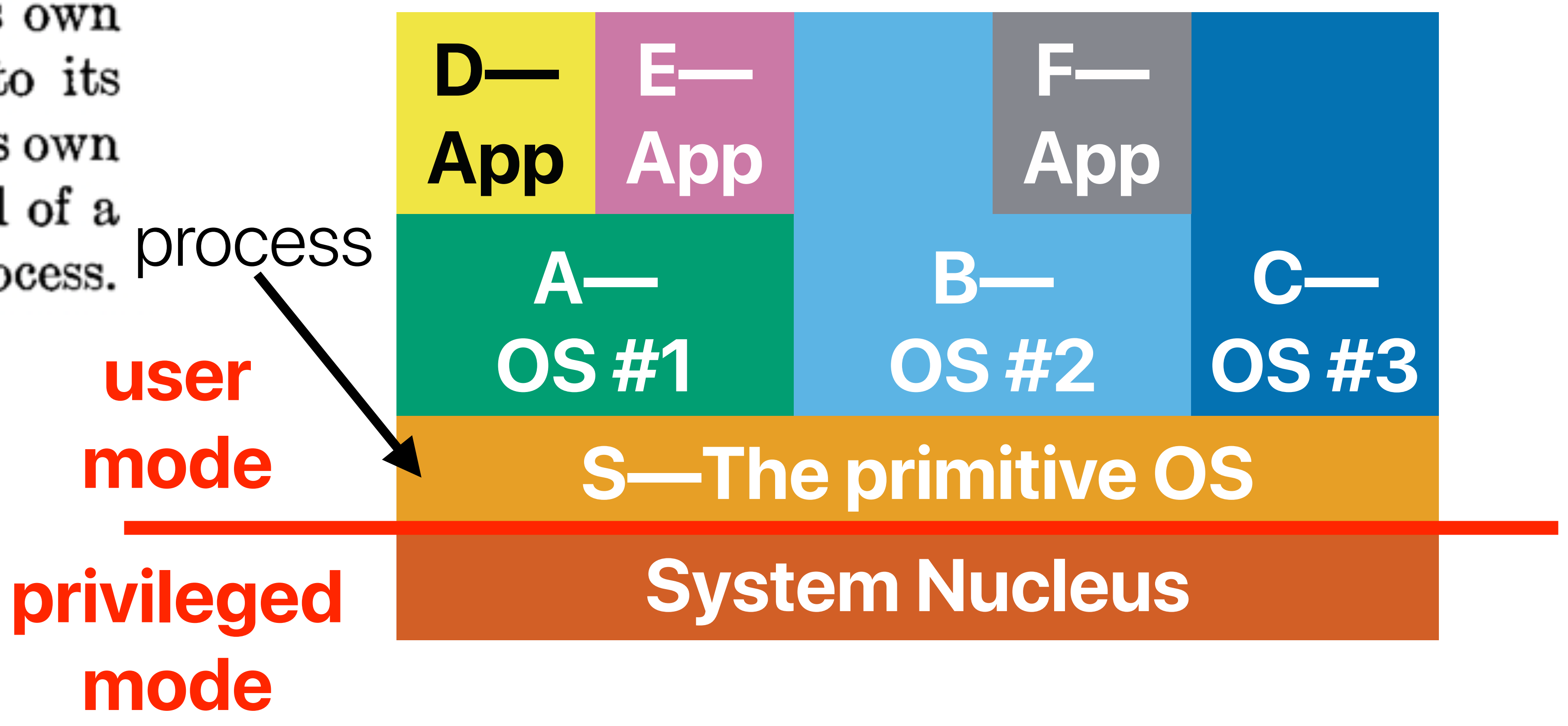
After initial loading, the internal store contains the system nucleus and a basic operating system, S, which can create parallel processes, A, B, C, etc., on request from consoles. The processes can in turn create other processes, D, E, F, etc. Thus while S acts as a primitive operating system for A, B, and C, these in turn act as operating systems for their children, D, E, and F. This is illustrated by



How many layers?



In this multiprogramming system, all privileged functions are implemented in the system nucleus, which has no built-in strategy. Strategies can be introduced at the various higher levels, where each process has the power to control the scheduling and resource allocation of its children. The only rules enforced by the nucleus are the following: a process can only allocate a subset of its own resources (including storage and message buffers) to its children; a process can only start, stop, and remove its own children (including their descendants). After removal of a process, its resources are returned to the parent process.



What is "system nucleus"

- Regarding "system nucleus", how many of the following statements are correct?
  - ✗ ① The system nucleus is a process
  - ② The system nucleus allows multiple operating systems to execute concurrently
  - ③ The system nucleus provides primitives to load and swap programs
  - ④ Operating systems are user-level processes in the system nucleus architecture
  - A. 0
  - B. 1
  - C. 2
  - [✓ D. 3]
  - E. 4

Hierarchical design v.s. flat structure

- Hierarchical
  - Ease of debugging/verification/testing
  - Lack of flexibility — you can only interact with neighbor layers
  - Overhead in each layer — not so great for performance
- Flat
  - Flexibility/Freedom

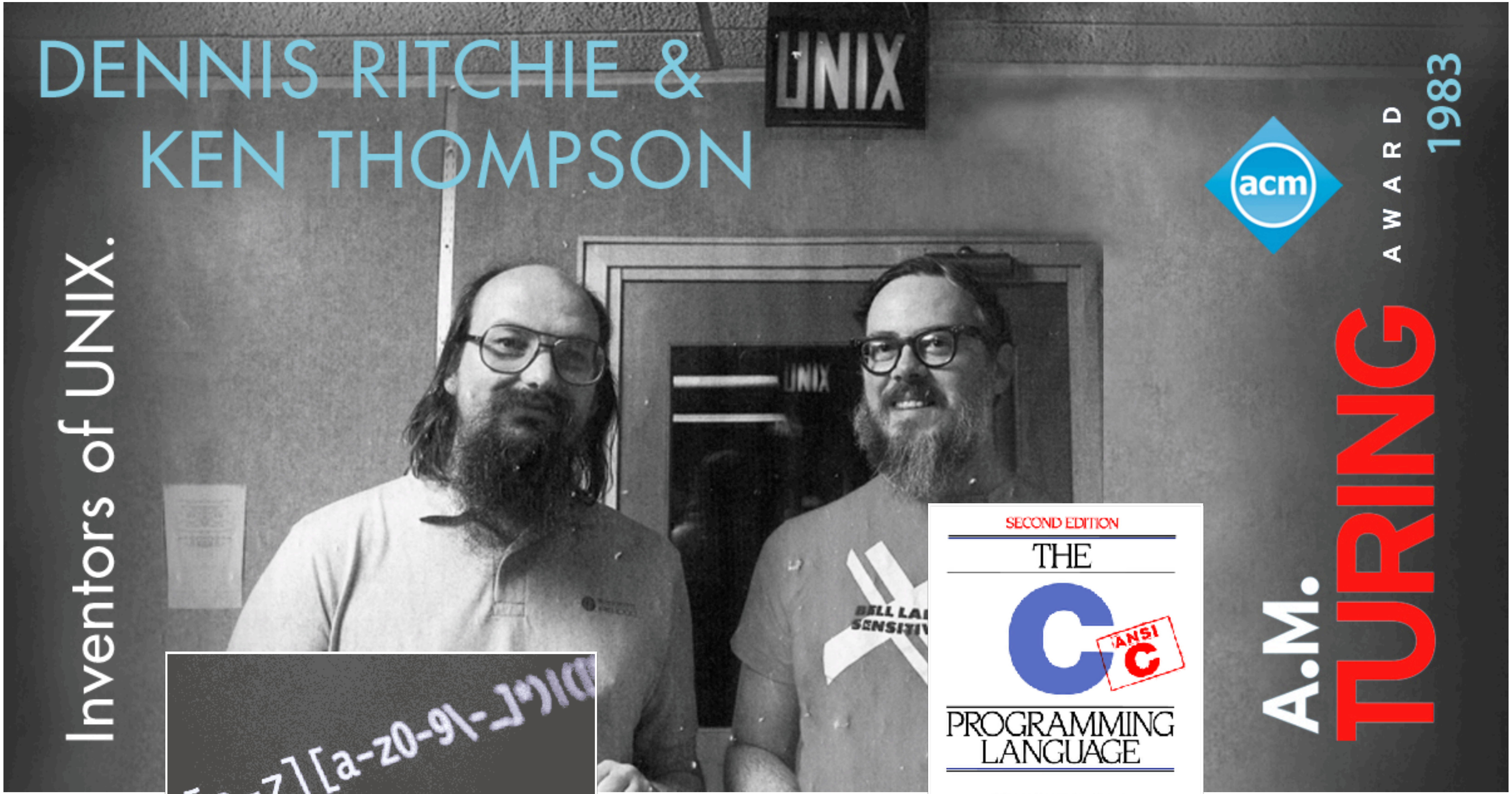
What the OS kernel should do?

The UNIX Time-Sharing System

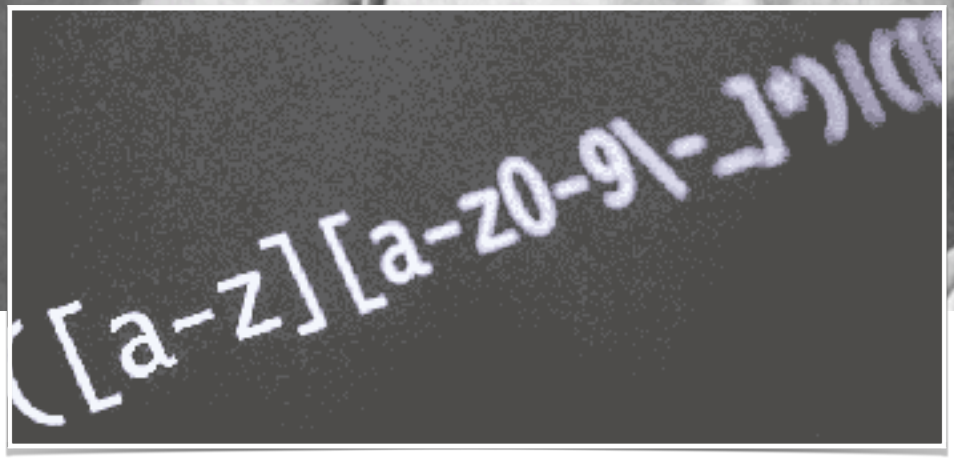
Dennis M. Ritchie and Ken Thompson
Bell Laboratories

DENNIS RITCHIE & KEN THOMPSON

Inventors of UNIX.



A.M. TURING AWARD 1983



```
([a-z][a-z0-9\-_]*)(@
```

SECOND EDITION

THE C PROGRAMMING LANGUAGE

ANSI C

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES

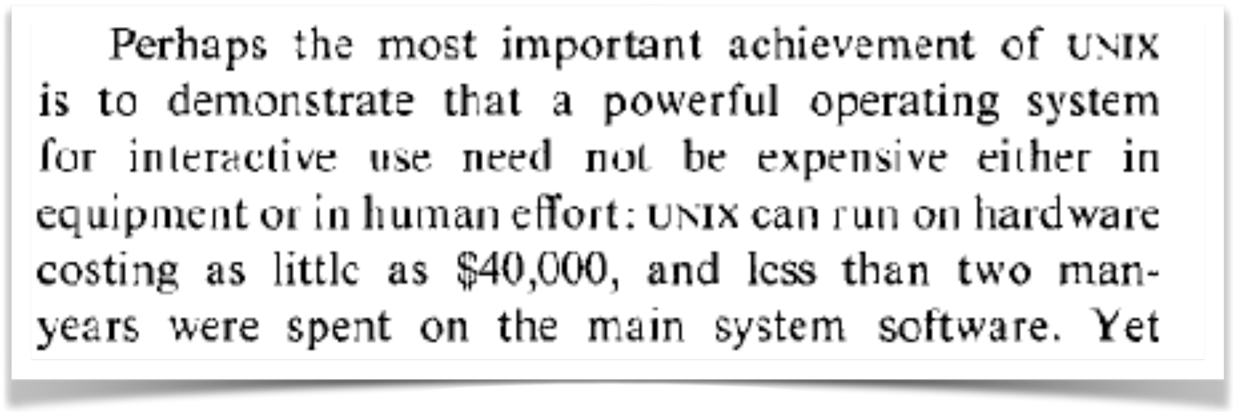
Poll close in 1:30

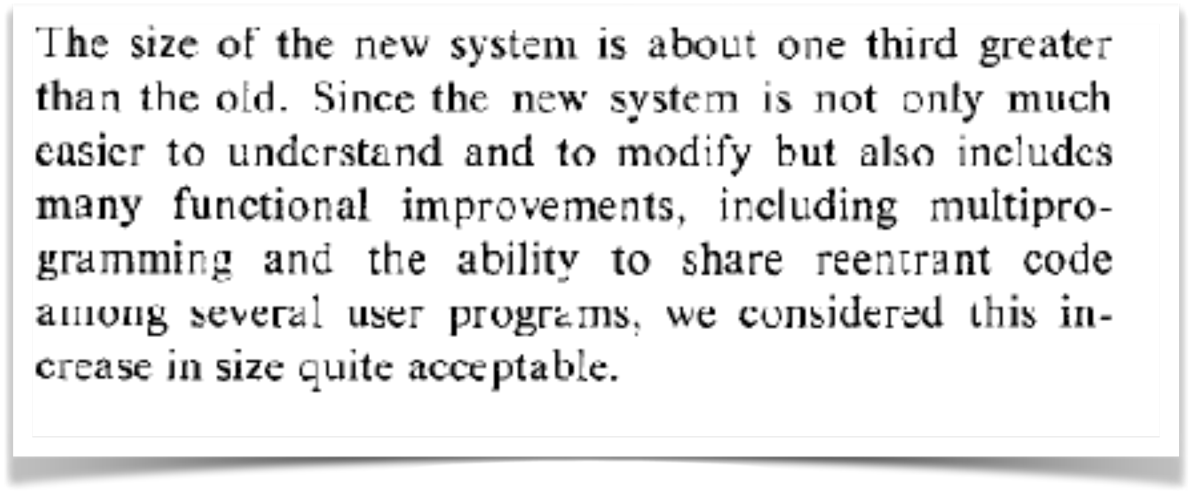
Why they built "UNIX"

- How many of following statements is/are the motivations of building UNIX?
  ① Reducing the cost of building machines with powerful OSes
  ② Reducing the burden of maintaining the OS code
  ③ Reducing the size of the OS code
  ④ Supporting networks and multiprocessors
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Why they built "UNIX"

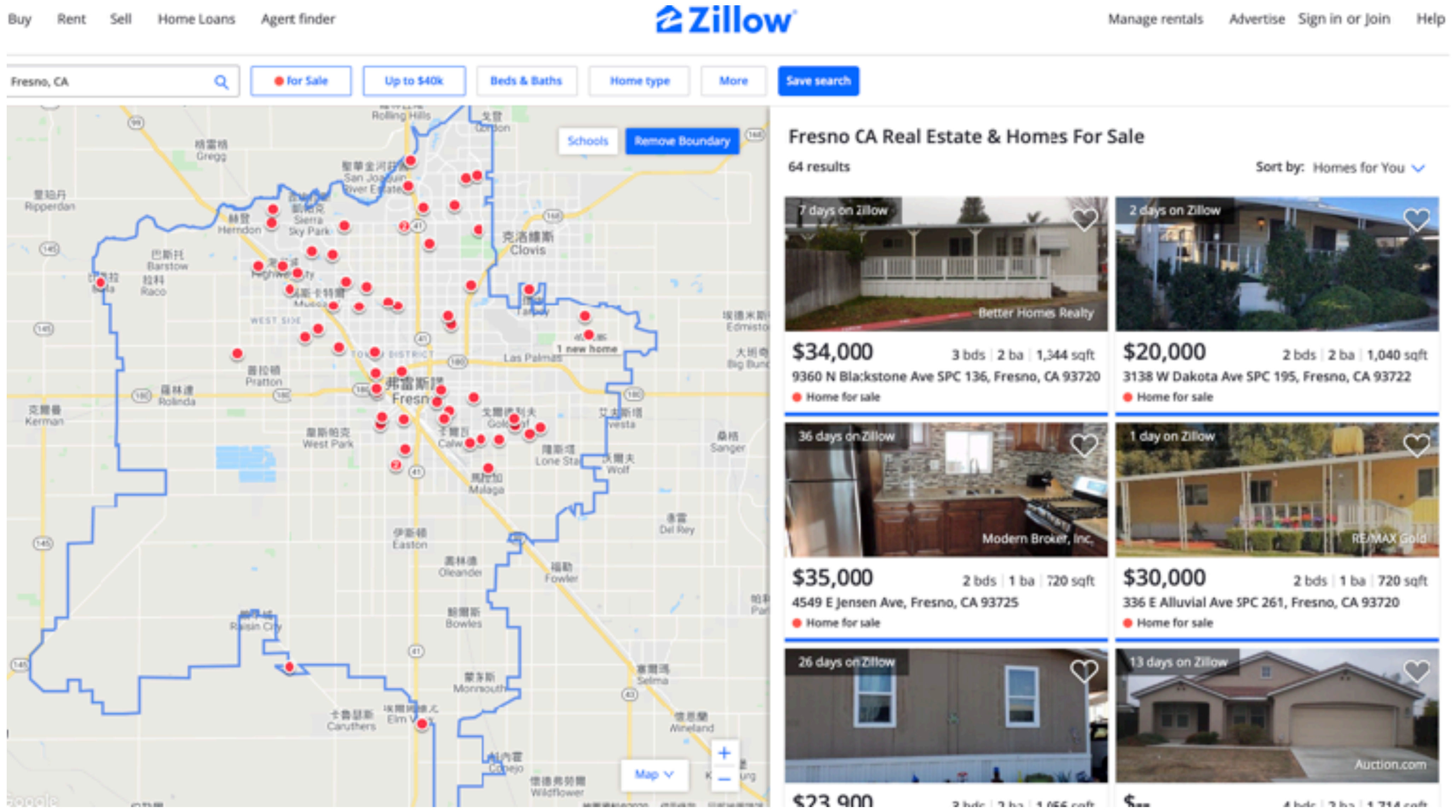
- How many of following statements is/are the motivations of building UNIX?
  ① Reducing the cost of building machines with powerful OSes
  ② Reducing the burden of maintaining the OS code
  ③ Reducing the size of the OS code
  ④ Supporting networks and multiprocessors
  A. 0
  B. 1
  [✓ C. 2]
  D. 3
  E. 4


Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive either in equipment or in human effort: UNIX can run on hardware costing as little as $40,000, and less than two man-years were spent on the main system software. Yet


The size of the new system is about one third greater than the old. Since the new system is not only much easier to understand and to modify but also includes many functional improvements, including multiprogramming and the ability to share reentrant code among several user programs, we considered this increase in size quite acceptable.

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



Fresno CA Real Estate & Homes For Sale

64 results

Sort by: Homes for You

7 days on Zillow — Better Homes Realty
$34,000 — 3 bds | 2 ba | 1,344 sqft
9360 N Blackstone Ave SPC 136, Fresno, CA 93720
Home for sale

2 days on Zillow
$20,000 — 2 bds | 2 ba | 1,040 sqft
3138 W Dakota Ave SPC 195, Fresno, CA 93722
Home for sale

36 days on Zillow — Modern Broker, Inc.
$35,000 — 2 bds | 1 ba | 720 sqft
4549 E Jensen Ave, Fresno, CA 93725
Home for sale

1 day on Zillow — RE/MAX Gold
$30,000 — 2 bds | 1 ba | 720 sqft
336 E Alluvial Ave SPC 261, Fresno, CA 93720
Home for sale

26 days on Zillow

13 days on Zillow — Auction.com

What UNIX proposed

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

Poll close in 1:30

The file abstraction

- How many of the following statements about UNIX is/are correct?
  - ① The semantics of accessing a device and accessing a text file is the same
  - ② For the file name `/alpha/beta/gamma`, alpha, beta, gamma are all files.
  - ③ Altering the content of directory requires privileged operations
  - ④ The programmer needs to treat random and sequential file accesses differently
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

The file abstraction

- How many of the following statements about UNIX is/are correct?
  ① The semantics of accessing a device and accessing a text file is the same
  ② For the file name `/alpha/beta/gamma`, alpha, beta, gamma are all files.
  ③ Altering the content of directory requires privileged operations
  ④ The programmer needs to treat random and sequential file accesses differently **doesn't**

  A. 0
  B. 1
  C. 2
  [✓ D. 3]
  E. 4

Poll close in 1:30

Protection

- Regarding the protection in the assigned UNIX paper, how many of the followings is/are correct?
  ① The same file may have different permissions for different user-id
  ② The owner of the file may not have the permission of writing a file
  ③ If the user does not have a permission to access a device, set-user-id will guarantee that the user will not be able to access that device
  ④ In the UNIX system described in this paper, if the file owner is “foo”, then the user “bar” will have the same permission as another user (e.g. “xyz”).
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

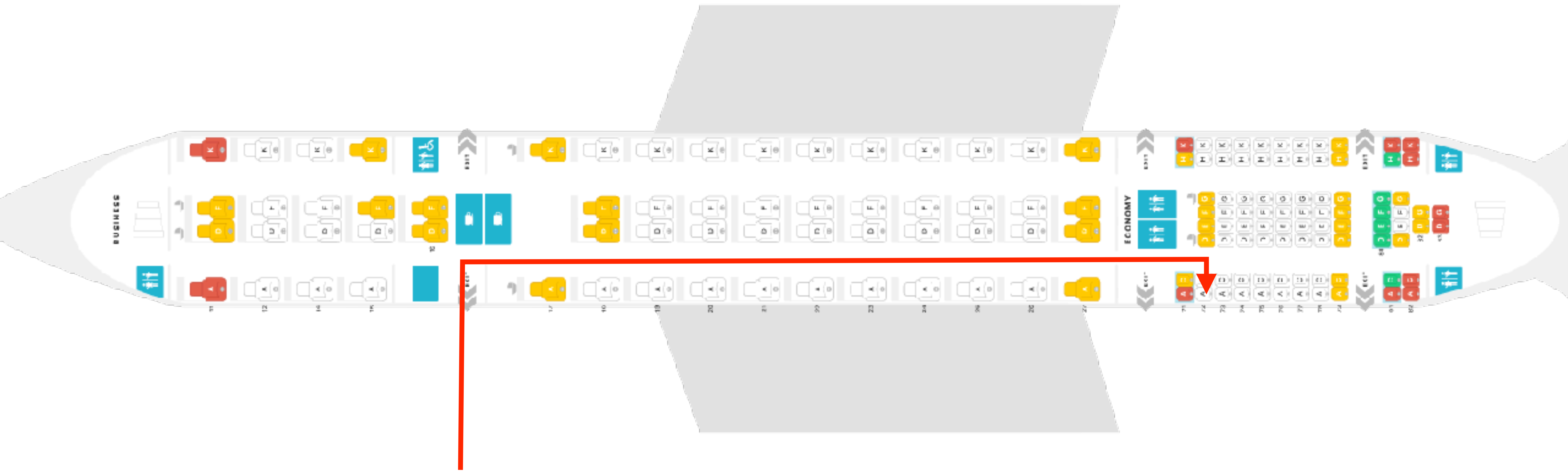
Protection

- Regarding the protection in the assigned UNIX paper, how many of the followings is/are correct?
  1. The same file may have different permissions for different user-id
  2. The owner of the file may not have the permission of writing a file
  3. If the user does not have a permission to access a device, set-user-id will ~~guarantee that the user will not be able to access that~~ device
     allow the user to have the same permission as the creator of the
  4. In the UNIX system described in this paper, if the file owner is "foo", then the user "bar" will have the same permission as another user (e.g. "xyz").

  The UNIX system at that time doesn't have "group" — everyone other than the owner is "others"

  A. 0
  B. 1
  C. 2
  [✓ D. 3]
  E. 4

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!

UNIX’s interface of managing processes

The basic process API of UNIX

- `fork`
- `wait`
- `exec`
- `exit`

`fork()`

- `pid_t fork();`
- `fork` used to create processes (UNIX)
- What does `fork()` do?
  - Creates a **new** address space (for child)
  - **Copies** parent's address space to child's
  - Points kernel resources to the parent's resources (e.g. open files)
  - Inserts child process into ready queue
- `fork()` returns twice
  - Returns the child's PID to the parent
  - Returns "0" to the child

Poll close in 1:30

What will happen?

- What happens if we execute the following code?

```
int main() {
    int pid;
        if ((pid = fork()) == 0) {
            printf ("My pid is %d\n", getpid());
        }
        printf ("Child pid is %d\n", pid);
         return 0;
}
```

Assume the parent's PID is 2; child's PID is 7.

	# of times "my pid" is printed	my pid values printed	# of times "child pid" is printed	child pid values printed
A	1	7	2	7, 0
B	1	2	2	7, 0
C	2	7, 2	1	7
D	1	0	2	7, 2
E	1	7	1	7

Poll close in 1:30

What will happen?

- What happens if we execute the following code?

```
int main() {
    int pid;
        if ((pid = fork()) == 0) {
            printf ("My pid is %d\n", getpid());
        }
        printf ("Child pid is %d\n", pid);
         return 0;
}
```

Assume the parent's PID is 2; child's PID is 7.

	# of times "my pid" is printed	my pid values printed	# of times "child pid" is printed	child pid values printed
A	1	7	2	7, 0
B	1	2	2	7, 0
C	2	7, 2	1	7
D	1	0	2	7, 2
E	1	7	1	7

What will happen?

- What happens if we execute the following code?

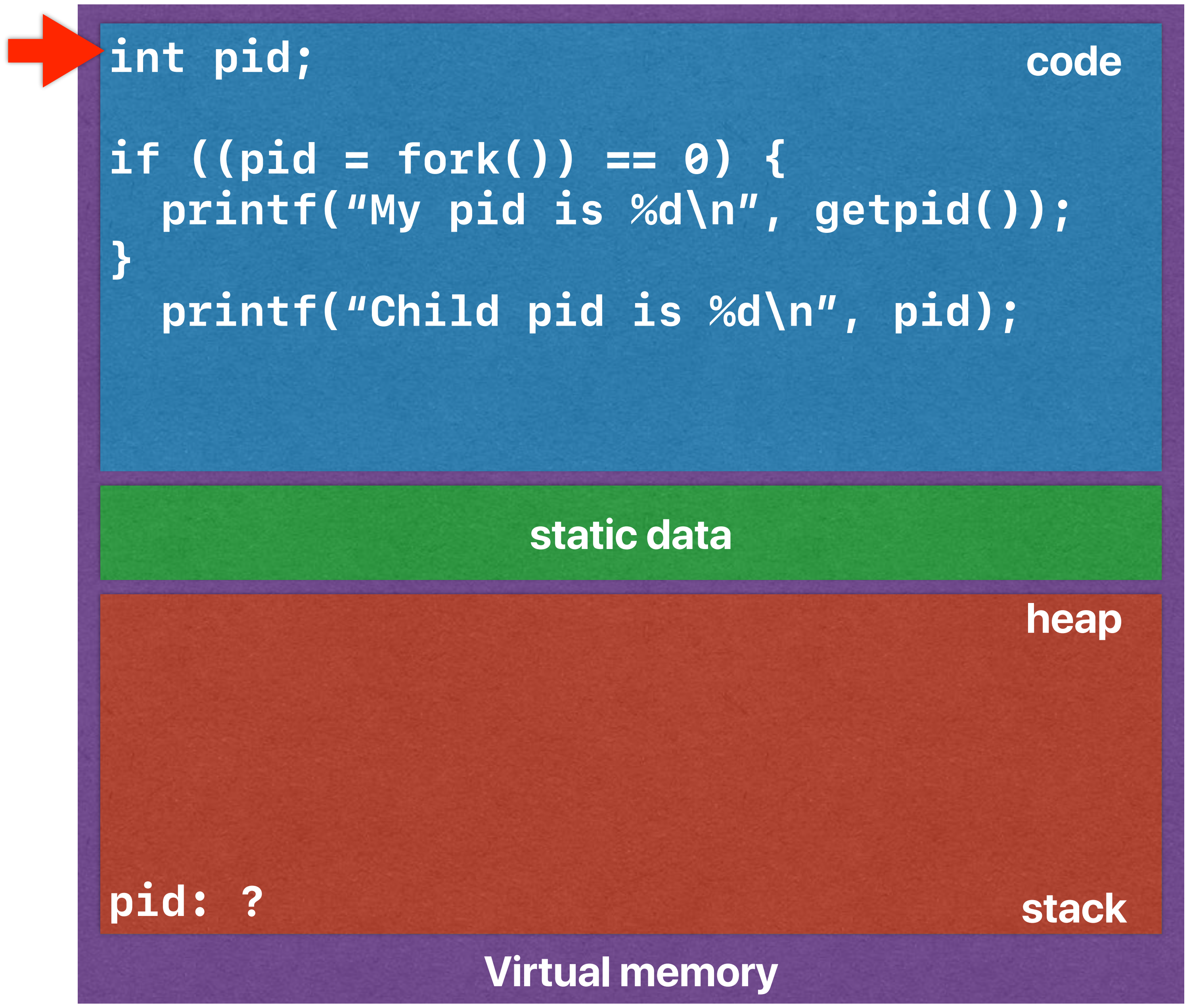
```
int main() {
    int pid;
        if ((pid = fork()) == 0) {
            printf (”My pid is %d\n", getpid());
        }
        printf (”Child pid is %d\n", pid);
         return 0;
}
```

Assume the parent’s PID is 2; child’s PID is 7.

	# of times “my pid” is printed	my pid values printed	# of times “child pid” is printed	child pid values printed
A	1	7	2	7, 0
B	1	2	2	7, 0
C	2	7, 2	1	7
D	1	0	2	7, 2
E	1	7	1	7

Assume
the parent's PID is 2;
child's PID is 7.

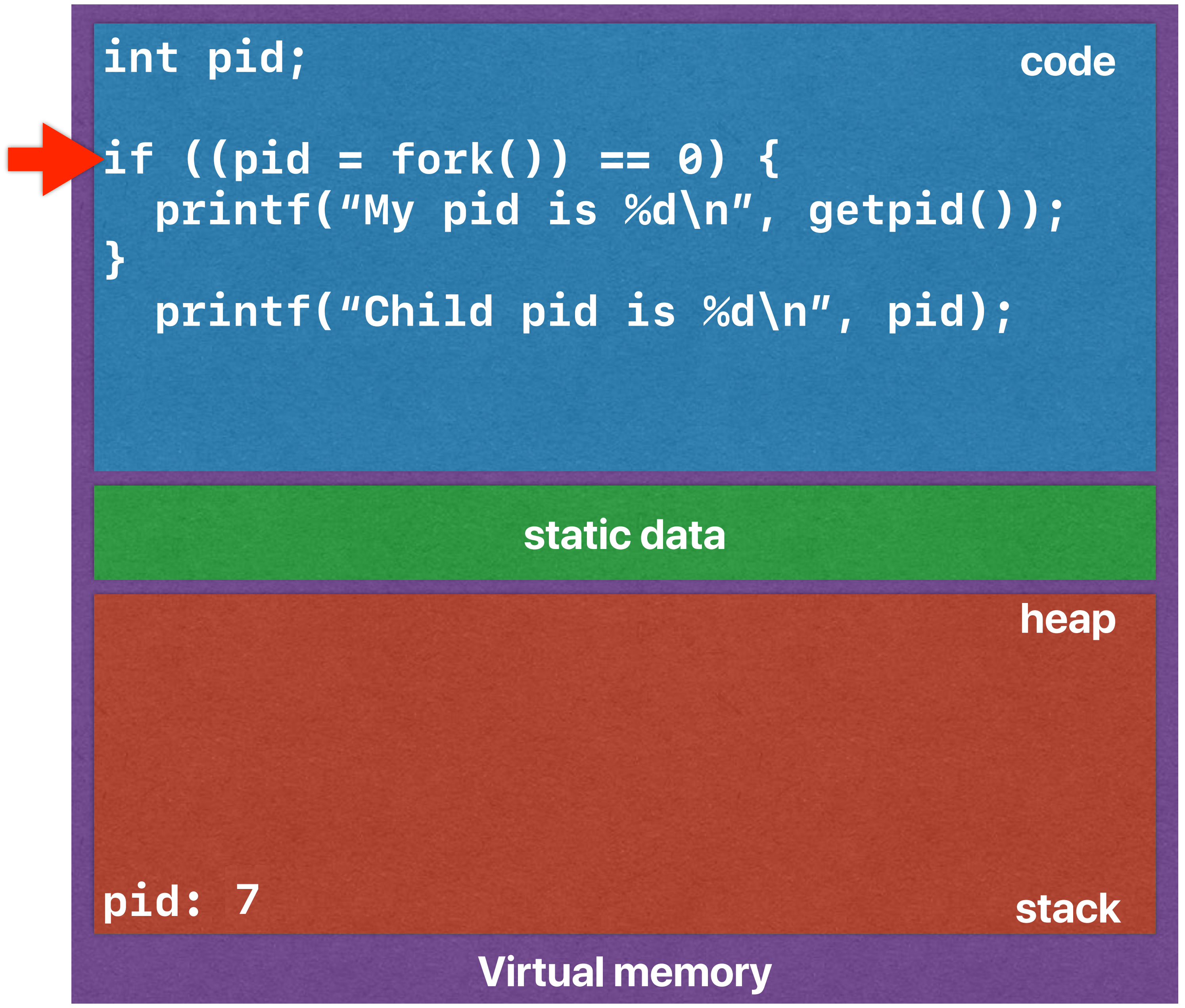
fork()



```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

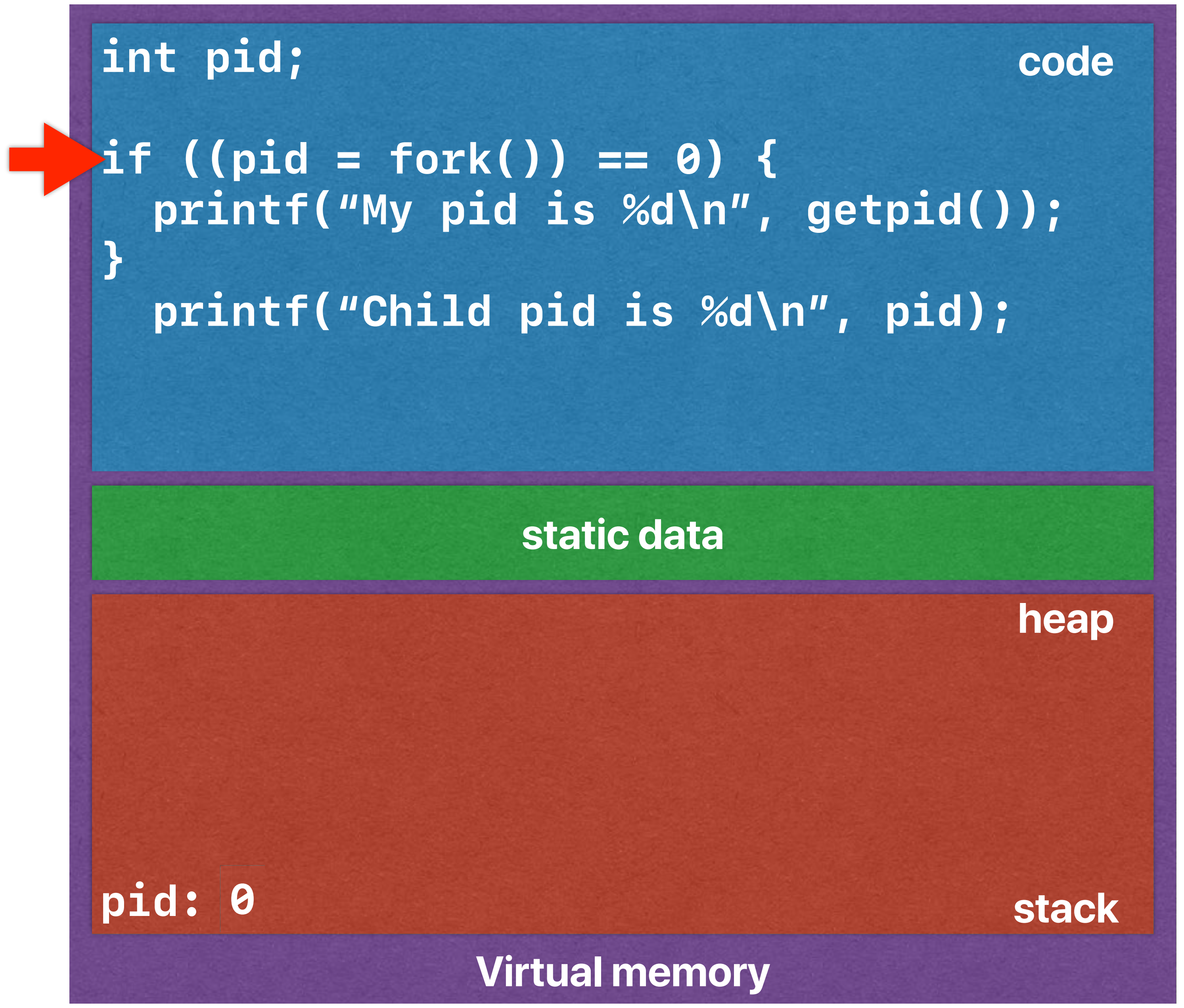
Assume
the parent's PID is 2;
child's PID is 7.

fork()



```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

```
pid: 7
```



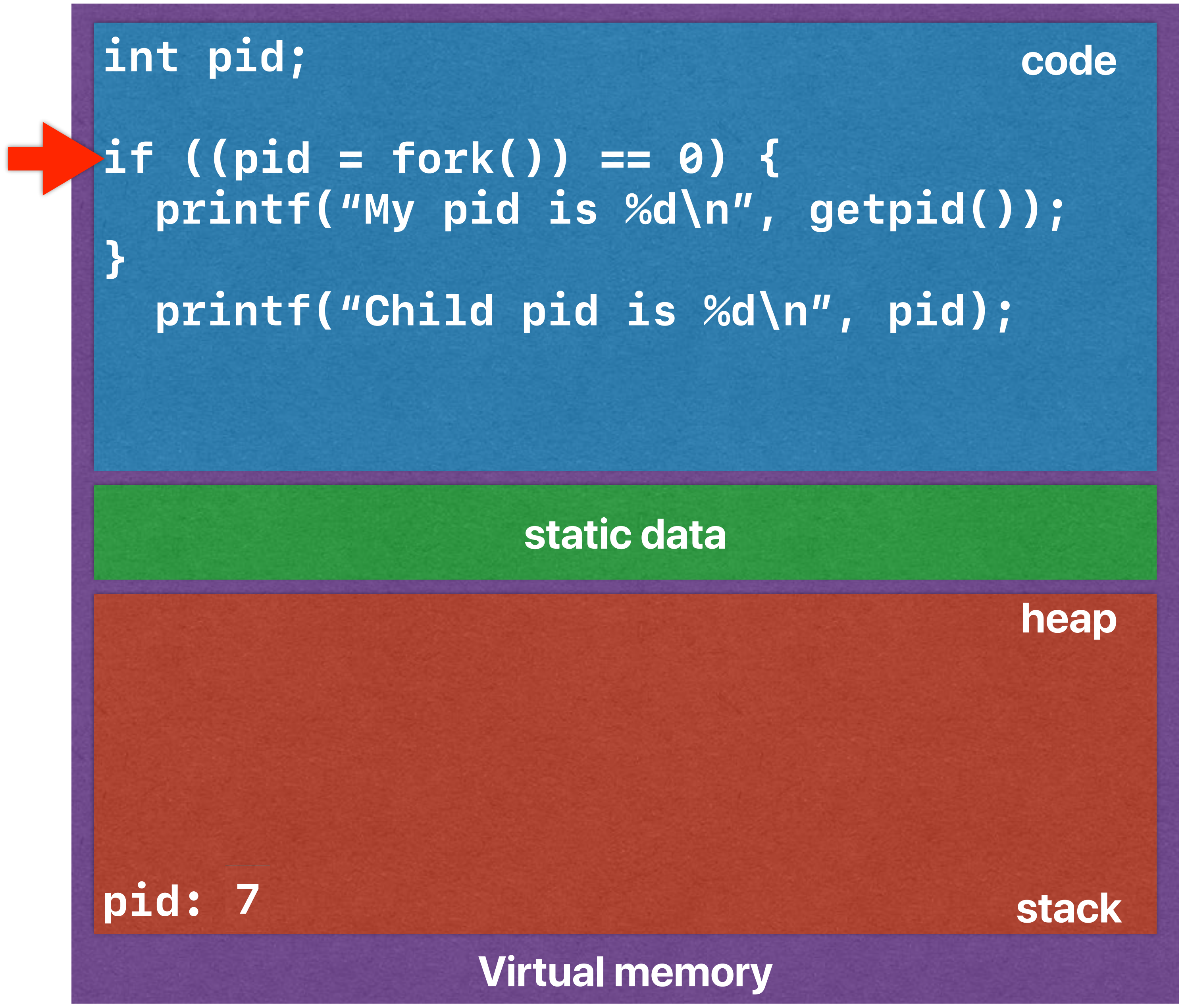
```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

```
pid: 0
```

Assume the parent's PID is 2; child's PID is 7.

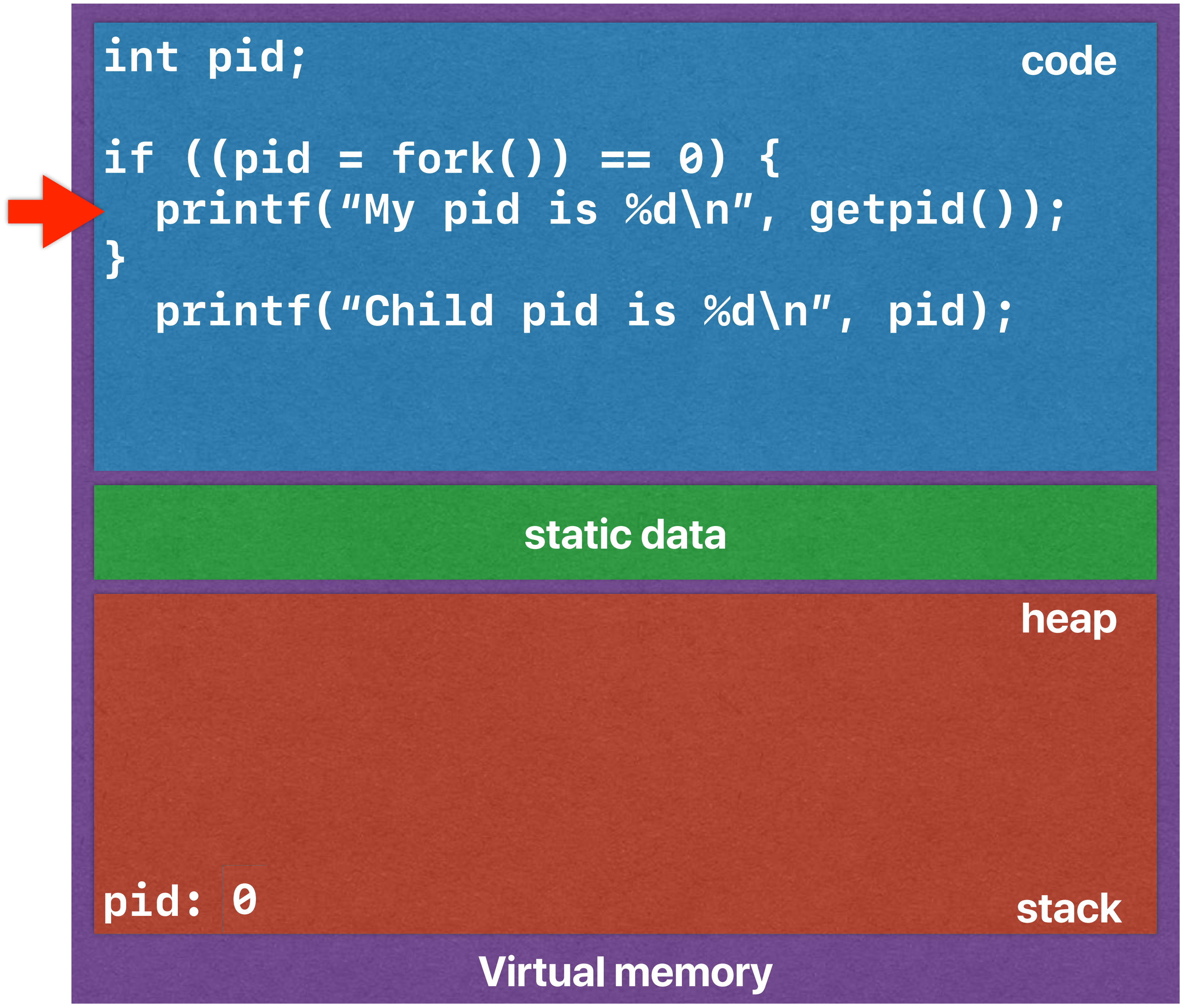
`fork()`

Output:
My pid is 7



```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

```
pid: 7
```



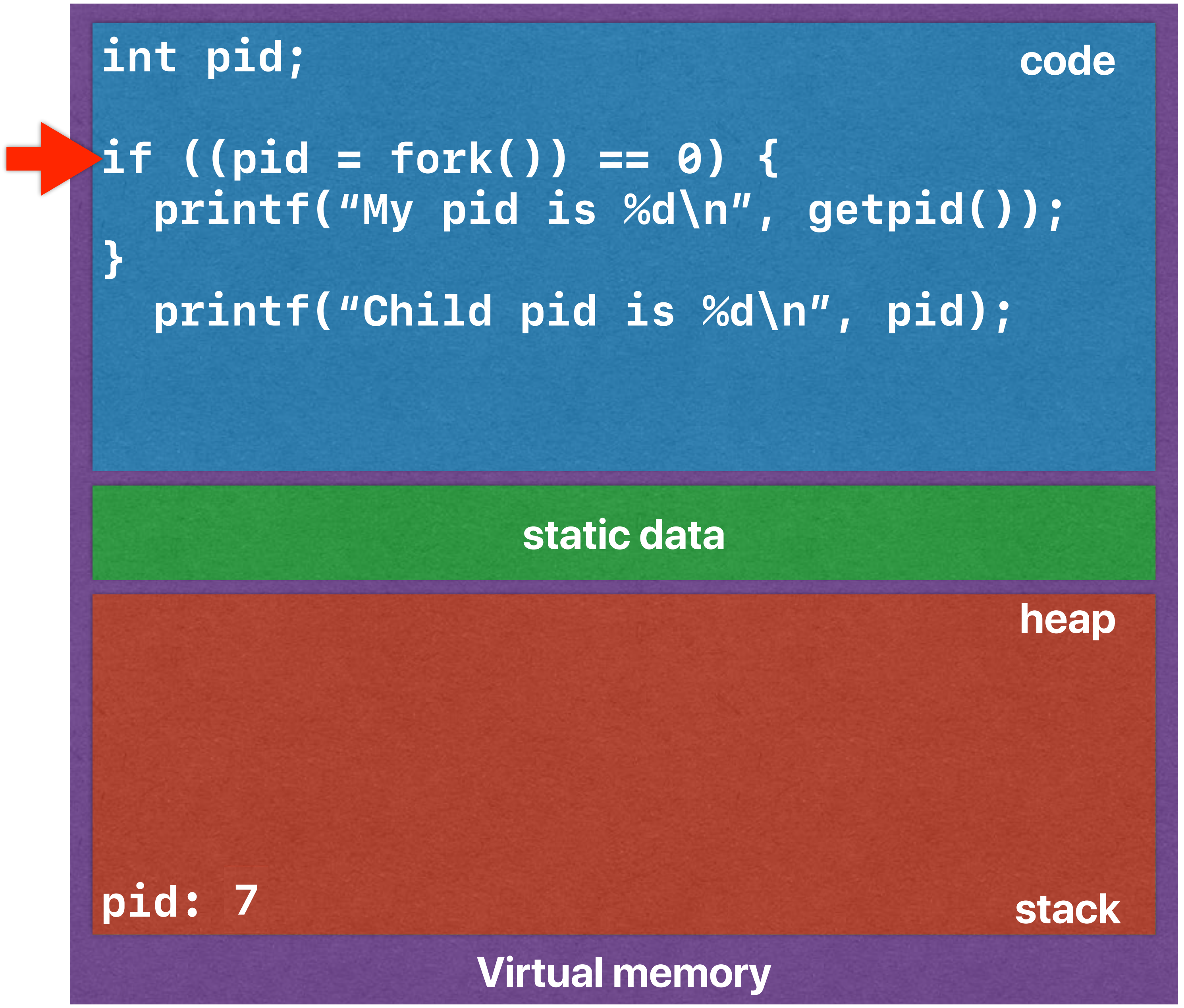
```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

```
pid: 0
```

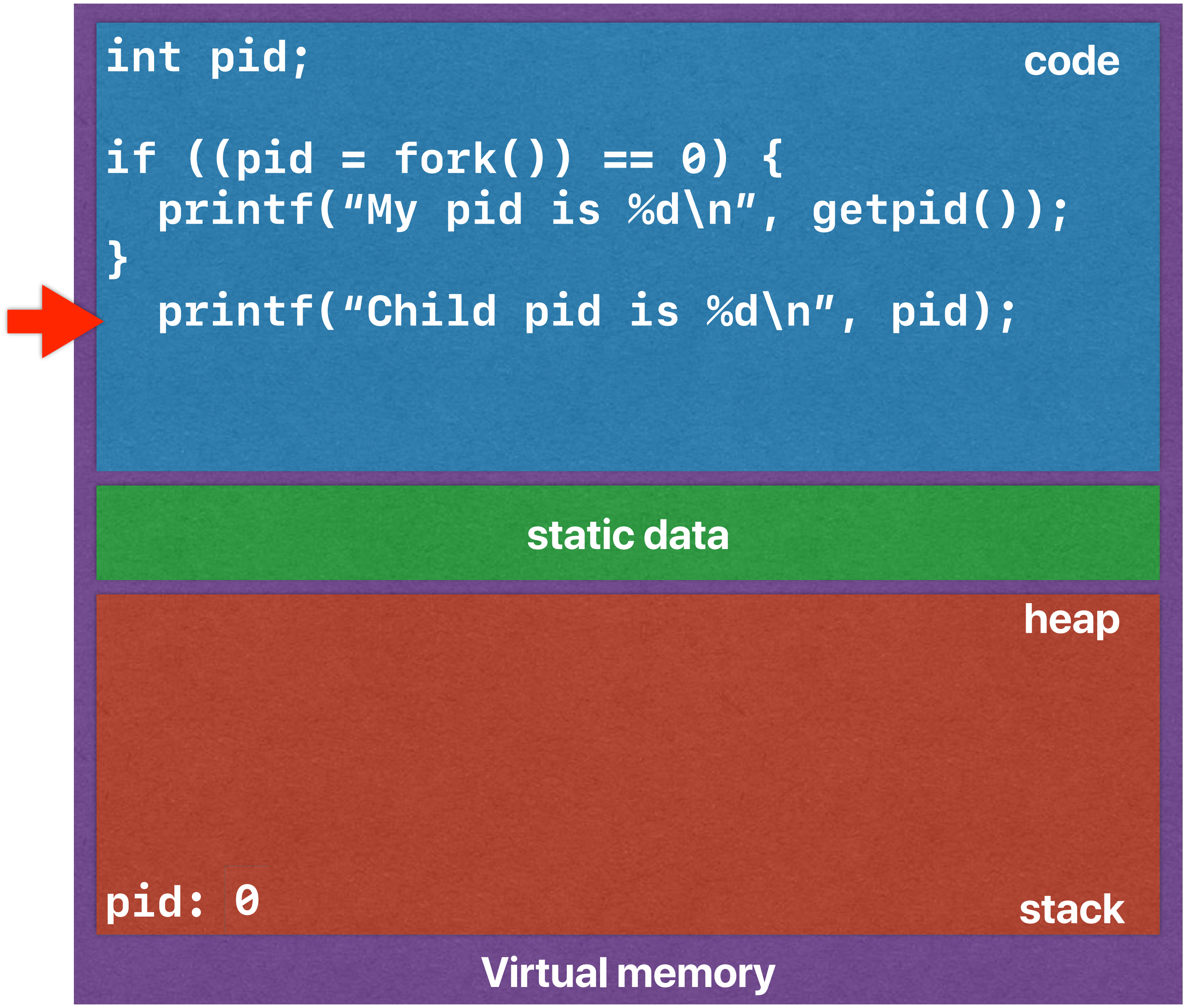
Assume
the parent's PID is 2;
child's PID is 7.

fork()

Output:
My pid is 7
Child pid is 0



```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

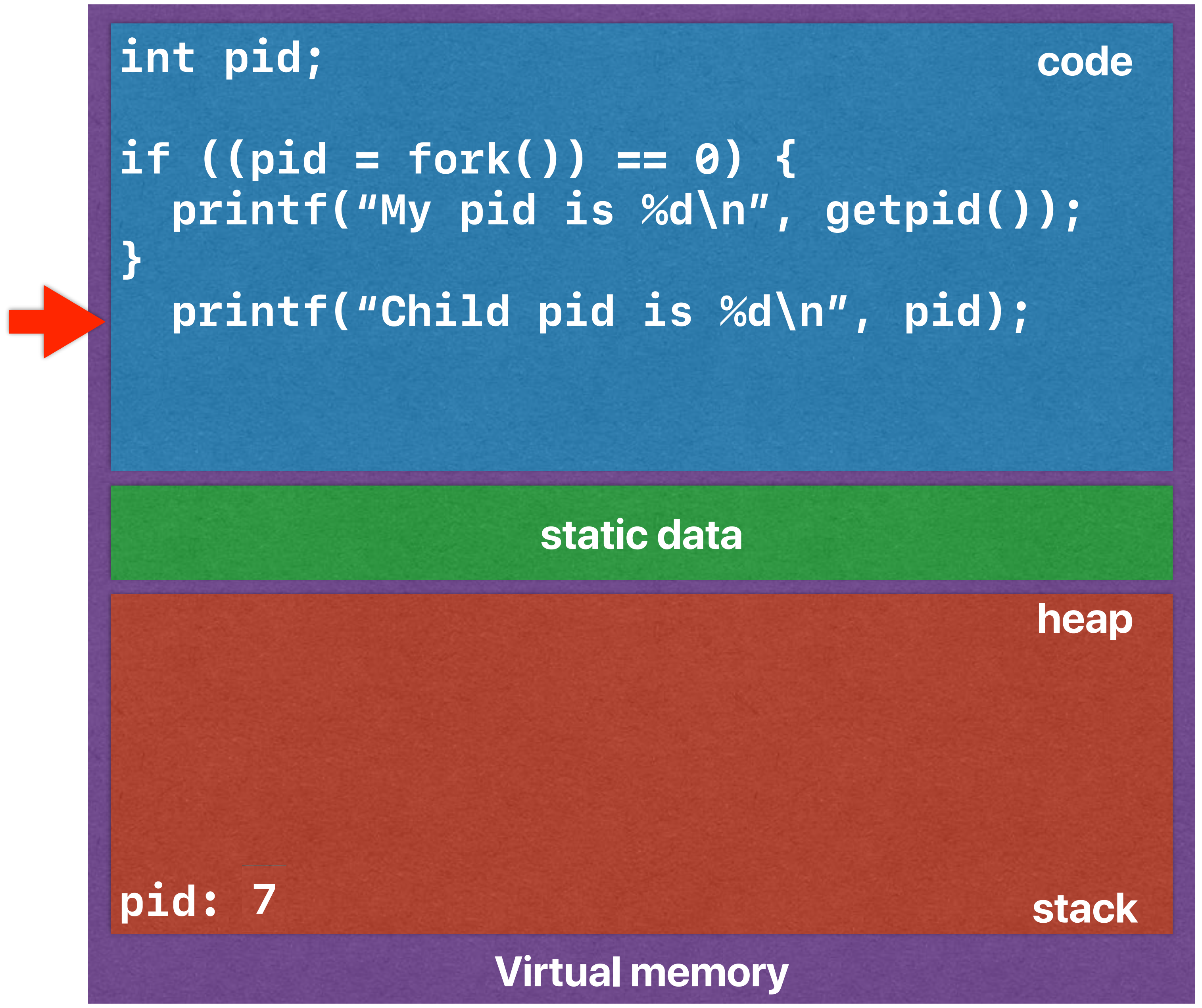


```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

Assume
the parent's PID is 2;
child's PID is 7.

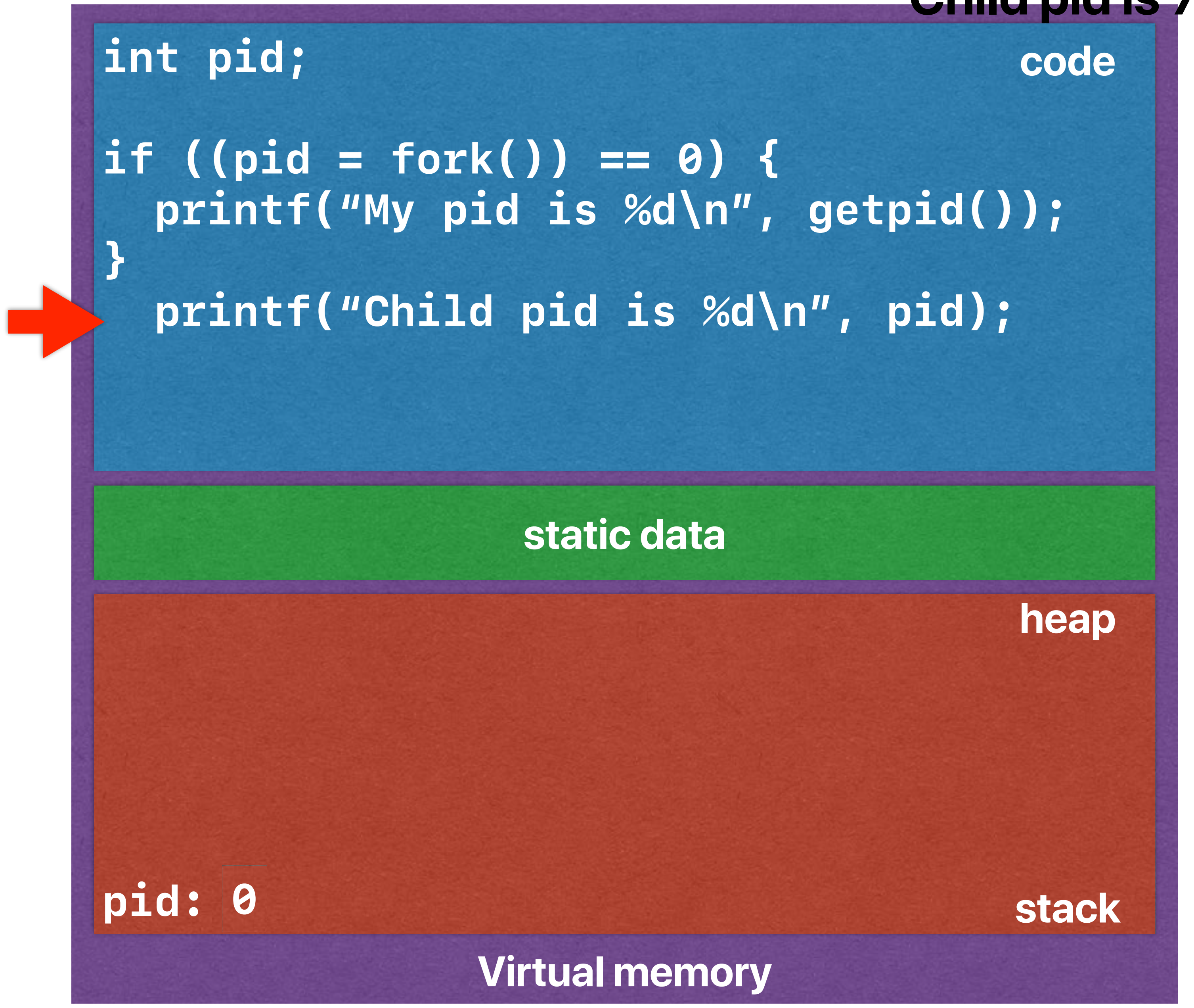
fork()

Output:
My pid is 7
Child pid is 0
Child pid is 7



```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

```
pid: 7
```



```
int pid;

if ((pid = fork()) == 0) {
  printf("My pid is %d\n", getpid());
}
  printf("Child pid is %d\n", pid);
```

```
pid: 0
```

What will happen?

- What happens if we execute the following code?

```
int main() {
    int pid;
        if ((pid = fork()) == 0) {
            printf ("My pid is %d\n", getpid());
        }
        printf ("Child pid is %d\n", pid);
        return 0;
}
```

Assume the parent's PID is 2; child's PID is 7.

	# of times "my pid" is printed	my pid values printed	# of times "child pid" is printed	child pid values printed
A	1	7	2	7,0
B	1	2	2	7,0
C	2	7,2	1	7
D	1	0	2	7,2
E	1	7	1	7

Announcement

- Reading quizzes due next Tuesday
  - Welcome new friends! — will drop a total of 6 reading quizzes for the quarter
  - Attendance count as 4 reading quizzes
  - We plan to have a total of 11 reading quizzes
- Office Hour links are inside Google Calendar events
  - https://calendar.google.com/calendar/u/0/r?cid=ucr.edu_b8u6dvkretn6kq6igunlc6bldg@group.calendar.google.com
  - Different links from lecture ones
  - We cannot share through any public channels so that we can better avoid Zoom bombing
- We will make both midterm and final exams online this quarter
  - Avoid the uncertainty of COVID-19
  - Avoid high-density in the classroom (only sits 60 and we have 59 for now) during examines

Computer
Science &
Engineering

202

つづく