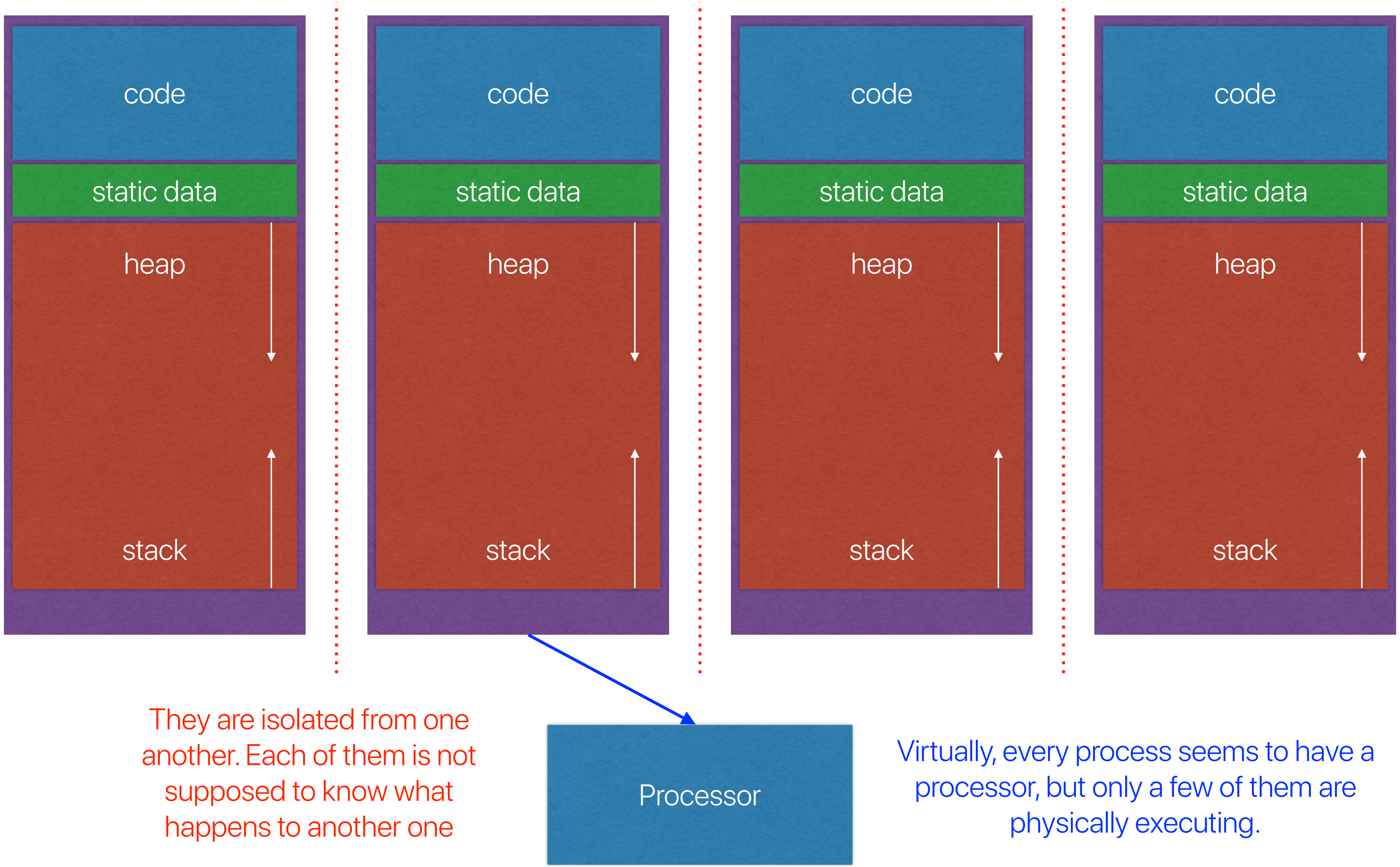
Design philosophy of operating systems (IV)

Hung-Wei Tseng

Recap: Each process has a separate virtual memory space



They are isolated from one another. Each of them is not supposed to know what happens to another one

Virtually, every process seems to have a processor, but only a few of them are physically executing.

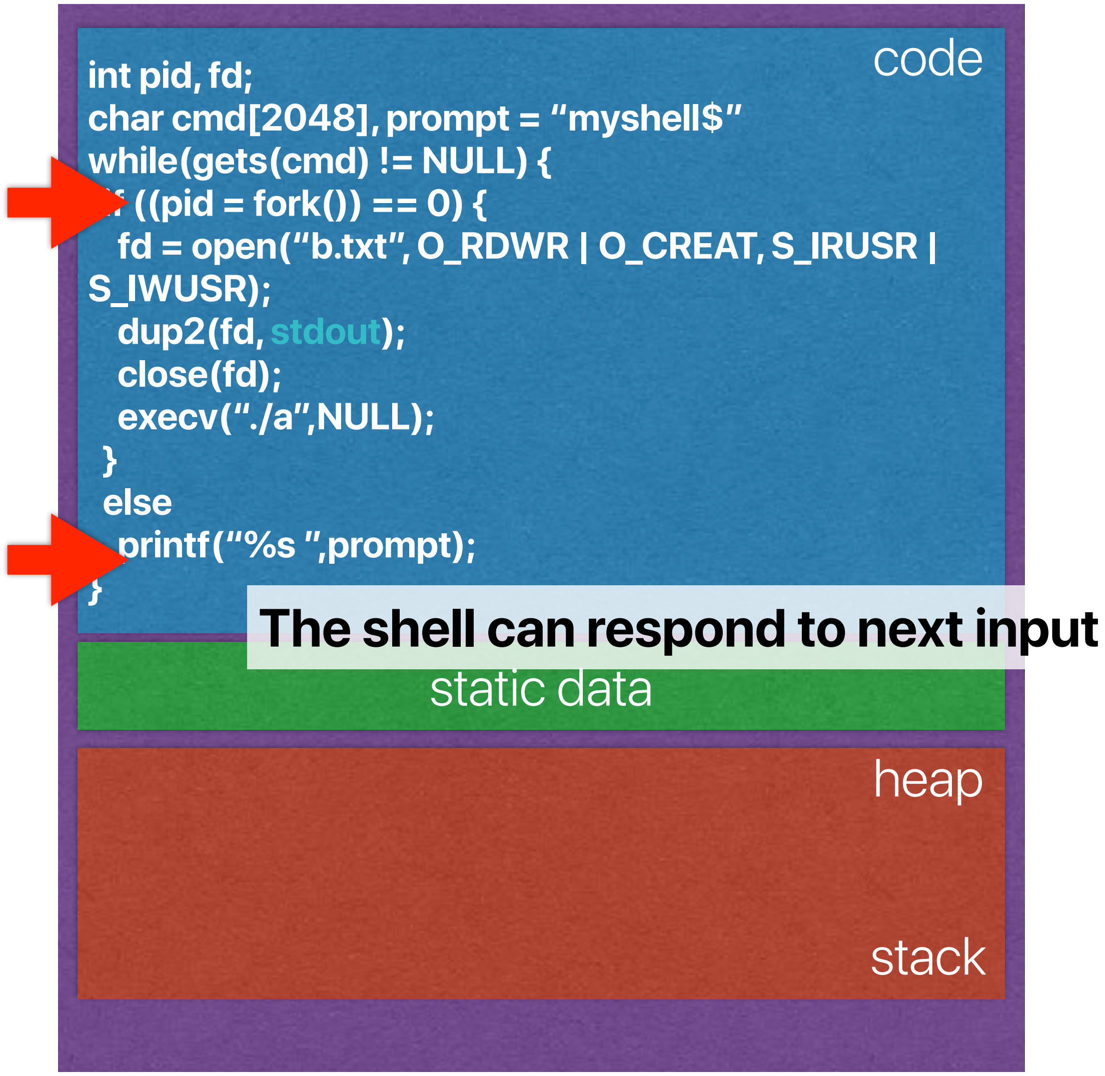
Recap: The basic process API of UNIX

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

Recap: How to implement redirection in shell

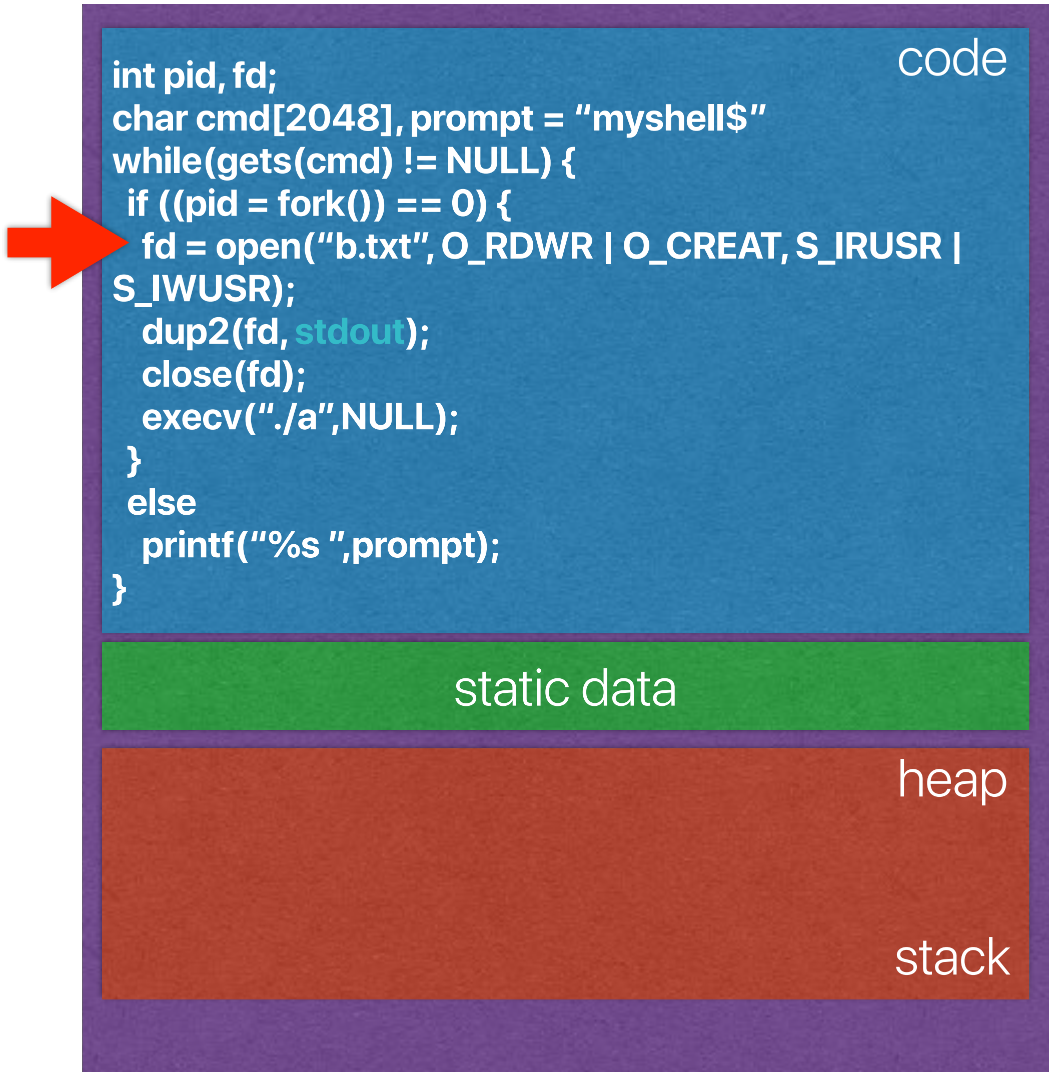
- Say, we want to do ./a > b.txt
- fork
- The forked code opens b.txt
- The forked code dup the file descriptor to stdin/stdout
- The forked code closes b.txt
- exec("./a", NULL)

Homework for you:
Think about the case when
your `fork` is equivalent to `fork+exec`()



```
int pid, fd;
char cmd[2048], prompt = "myshell$"
while(gets(cmd) != NULL) {
 if ((pid = fork()) == 0) {
   fd = open("b.txt", O_RDWR | O_CREAT, S_IRUSR |
S_IWUSR);
   dup2(fd, stdout);
   close(fd);
   execv("./a",NULL);
 }
 else
   printf("%s ",prompt);
}
```

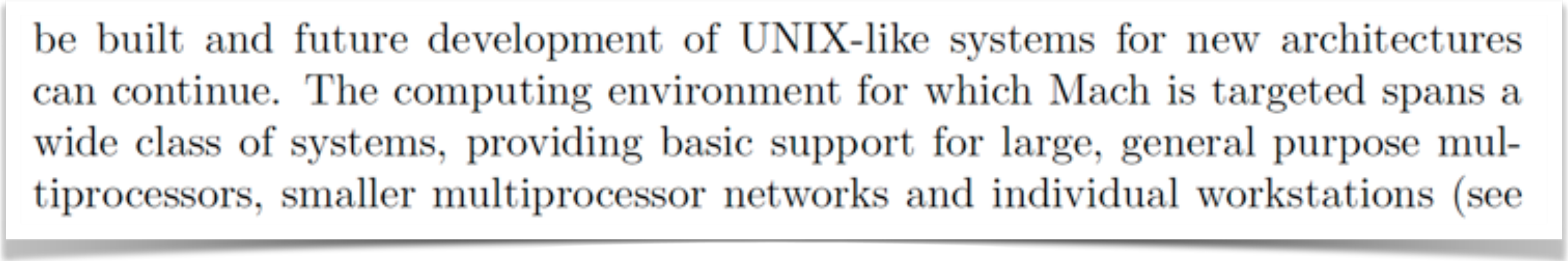
The shell can respond to next input



```
int pid, fd;
char cmd[2048], prompt = "myshell$"
while(gets(cmd) != NULL) {
 if ((pid = fork()) == 0) {
   fd = open("b.txt", O_RDWR | O_CREAT, S_IRUSR |
S_IWUSR);
   dup2(fd, stdout);
   close(fd);
   execv("./a",NULL);
 }
 else
   printf("%s ",prompt);
}
```

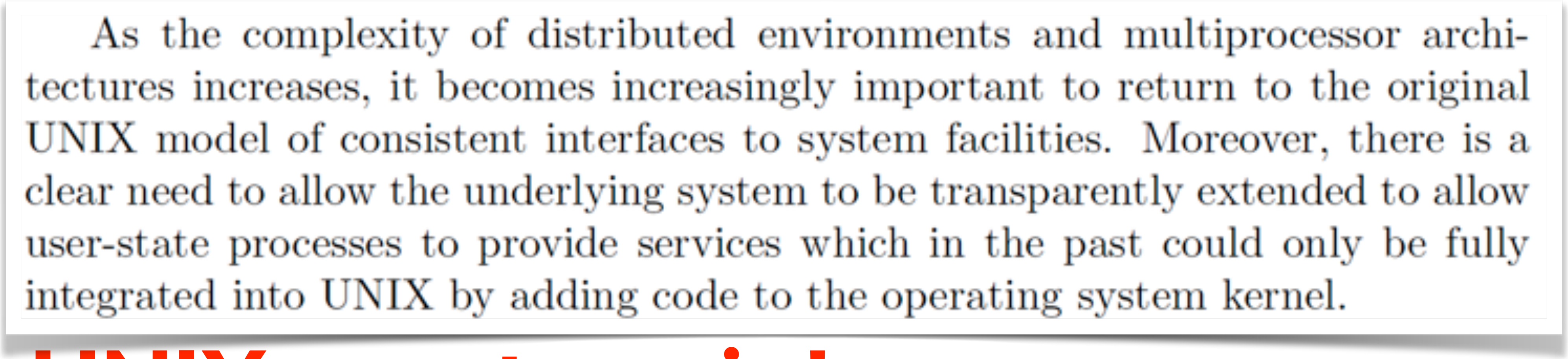
Why "Mach"?

- The hardware is changing
  - Multiprocessors
  - Networked computing



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

- The software
  - The demand of extending an OS easily
  - Repetitive but confusing mechanisms for similar stuffs



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!

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

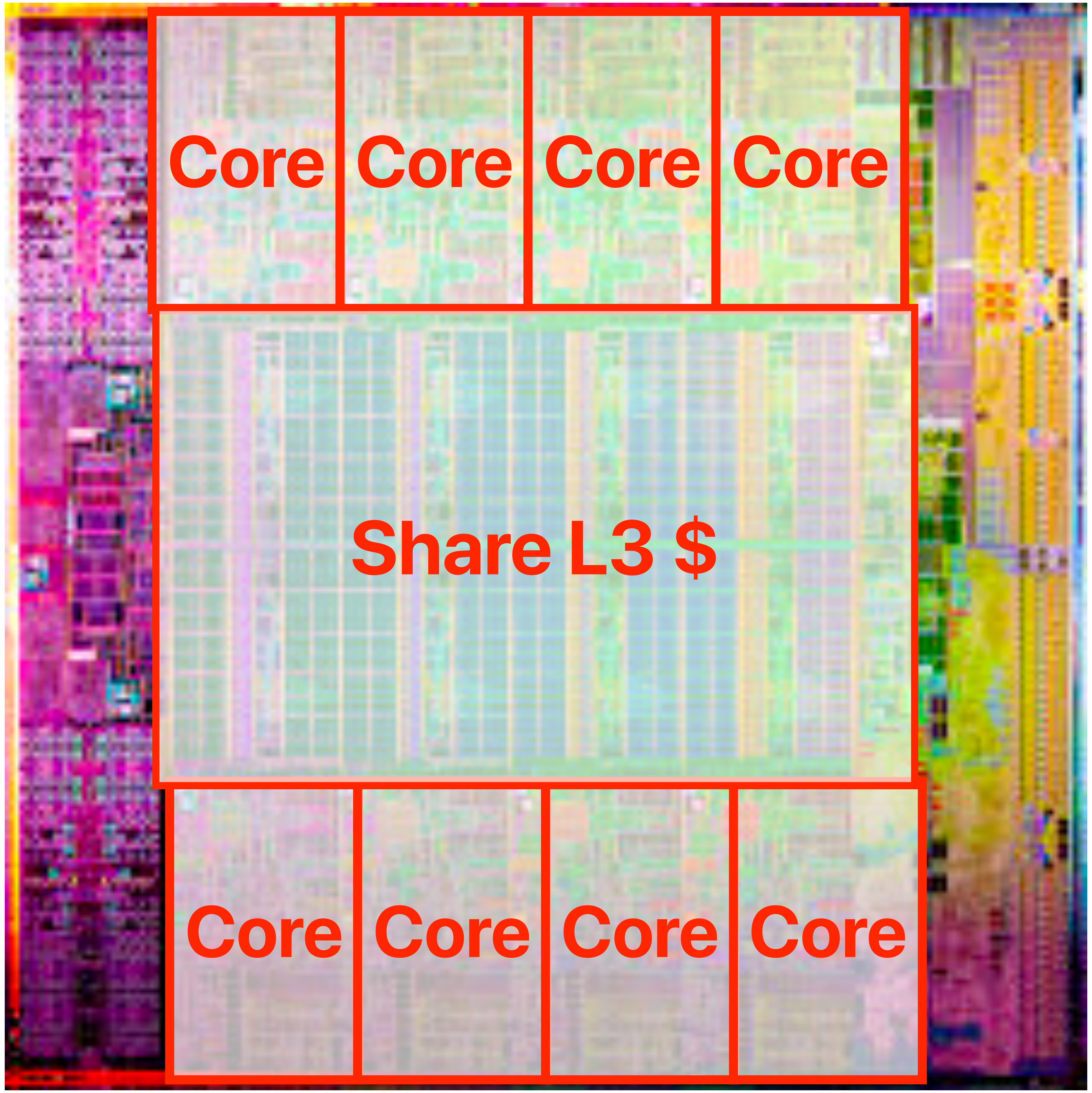
B. 1

C. 2

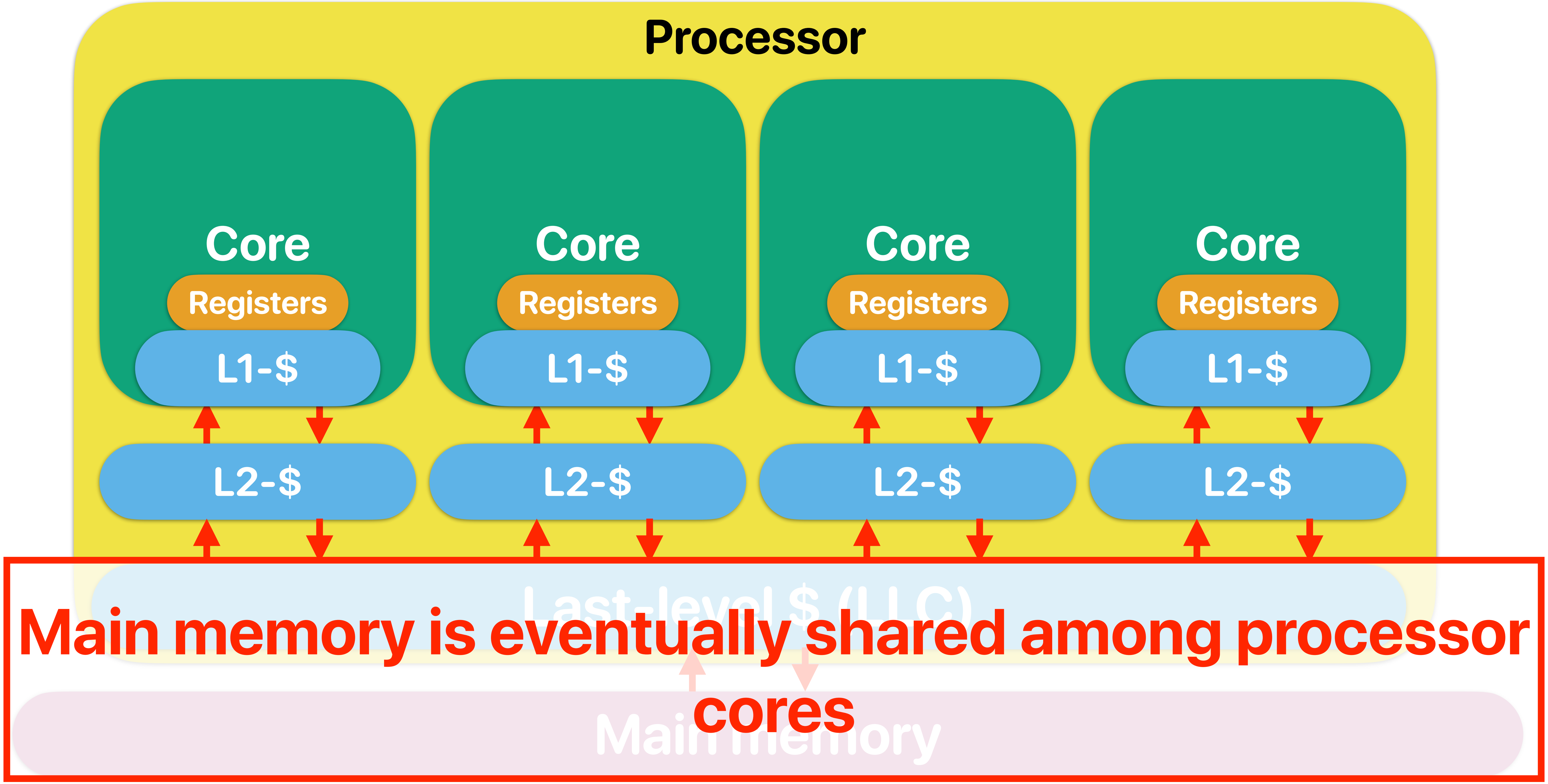
[✓ D. 3]

E. 4

Intel Sandy Bridge



Concept of chip multiprocessors



Main memory is eventually shared among processor cores

Current scoreboard

Red	Blue
4	5

Outline

- Mach: A New Kernel Foundation For UNIX Development (cont.)
- Taxonomy of Kernels
- Threads

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

Poll close in 1:30

Tasks/Processes and threads

- How many of the following regarding the comparison of parallelizing computation tasks using processes and threads is/are correct?

  ① The context switch and creation overhead of processes is higher

  ② The overhead of exchanging data among different computing tasks for the same applications is higher in process model

  ③ The demand of memory usage is higher when using processes

  ④ The security and isolation guarantees are better achieved using processes

  A. 0

  B. 1

  C. 2

  D. 3

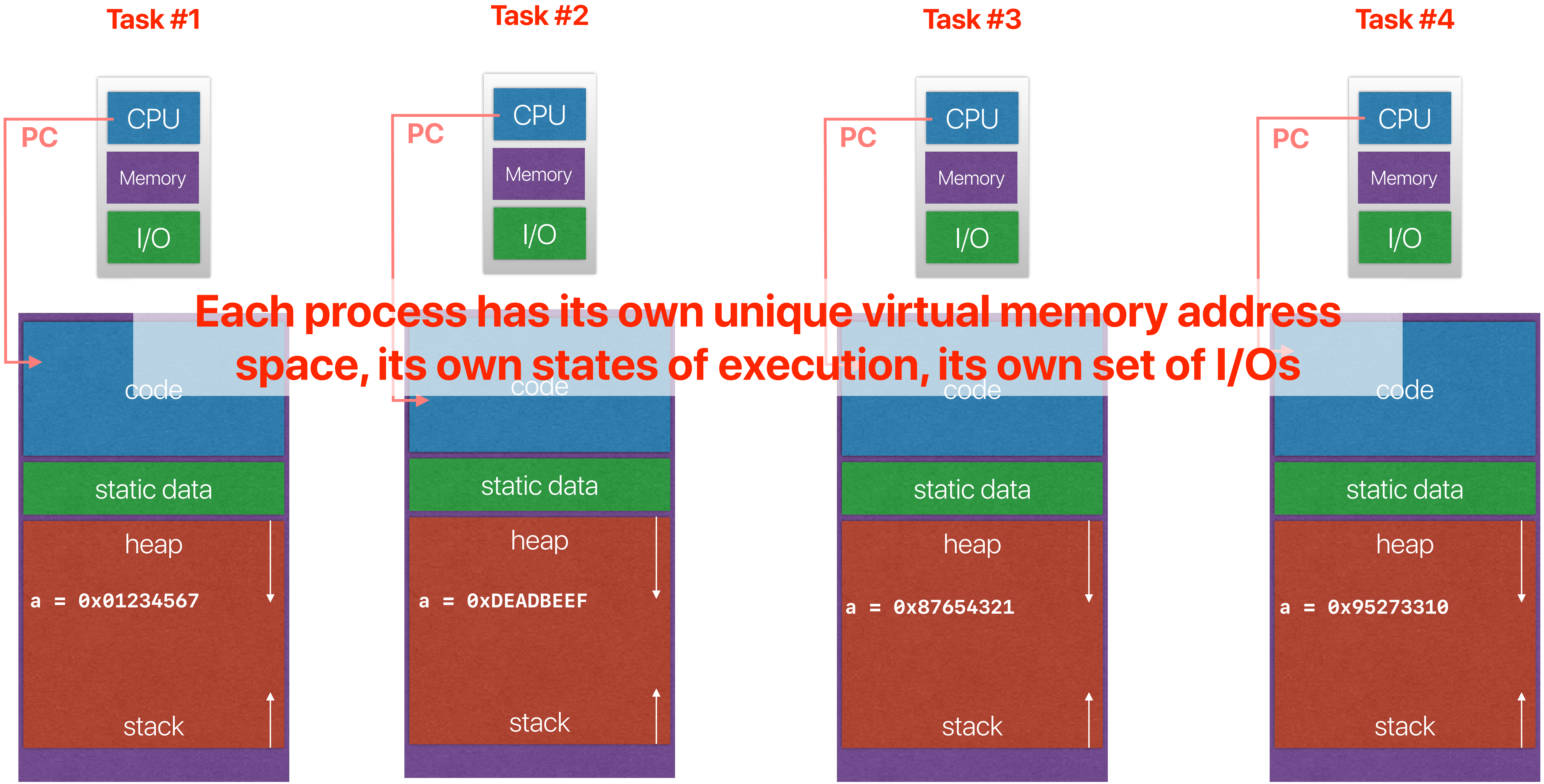
  E. 4

Poll close in 1:30

Tasks/Processes and threads

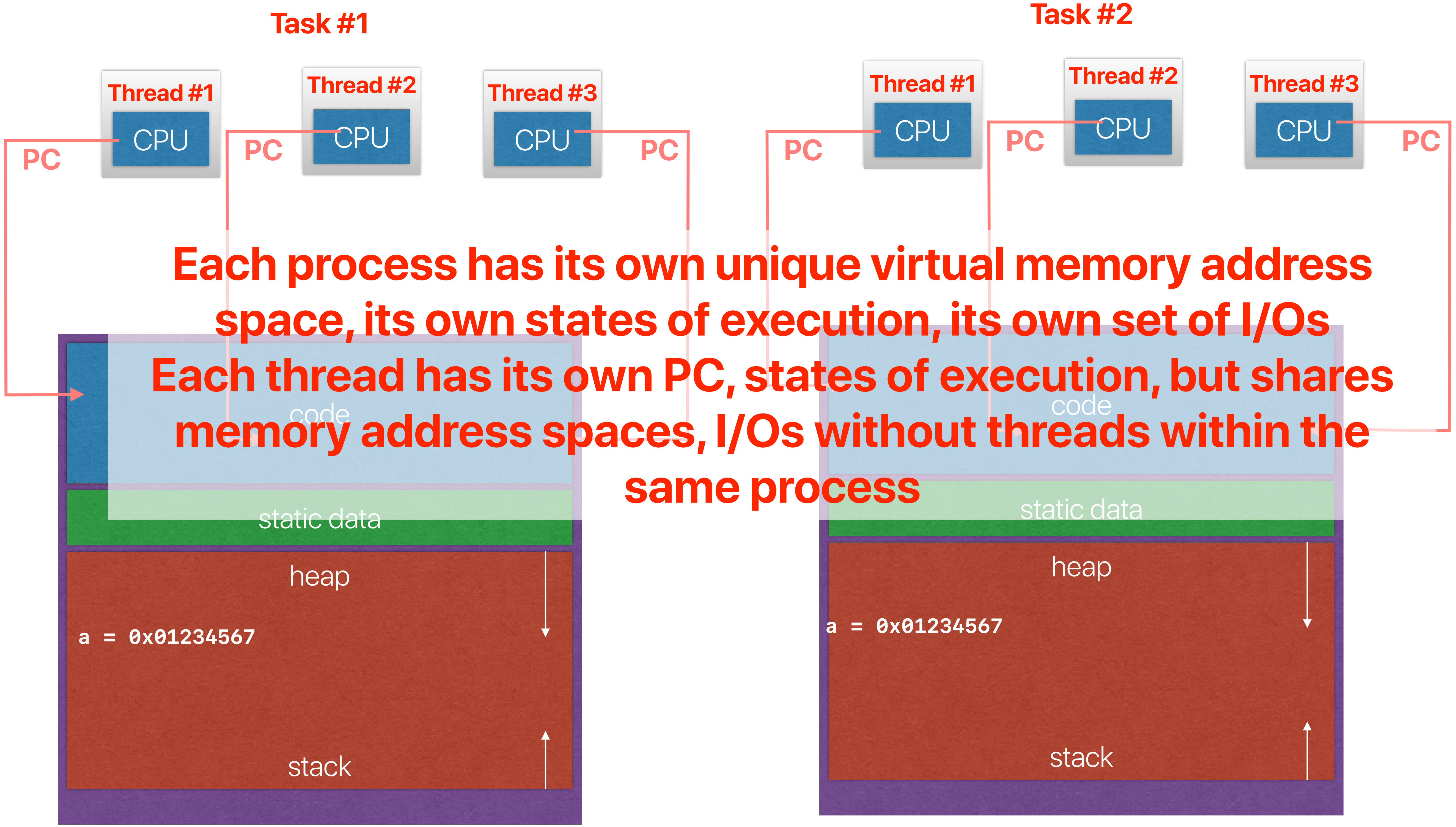
- How many of the following regarding the comparison of parallelizing computation tasks using processes and threads is/are correct?
  ① The context switch and creation overhead of processes is higher
  ② The overhead of exchanging data among different computing tasks for the same applications is higher in process model
  ③ The demand of memory usage is higher when using processes
  ④ The security and isolation guarantees are better achieved using processes
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

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

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 3.7GHz intel Core i5-9600K Processor
  - Process fork+exit ~ 57 microseconds
  - More than 16K cycles

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

Tasks/Processes and threads

- How many of the following regarding the comparison of parallelizing computation tasks using processes and threads is/are correct?

① The context switch and creation overhead of processes is higher

— you have to change page tables, warm up TLBs, warm up caches, create a new memory space …

② The overhead of exchanging data among different computing tasks for the same applications is higher in process model

— you cannot directly share data without leveraging other mechanisms

③ The demand of memory usage is higher when using processes

— each process needs its own address space even if most data are potentially identical

④ The security and isolation guarantees are better achieved using processes

— separate address, it's not easy to access data from another process

A. 0

B. 1

C. 2

D. 3

[✓ E. 4]

Why Threads?

- Process is an abstraction of a computer
  - When you create a process, you duplicate everything
  - However, you only need to duplicate CPU abstraction to parallelize computation tasks
- Threads as lightweight processes
  - Thread is an abstraction of a CPU in a computer
  - Maintain separate execution context
  - Share other resources (e.g. memory)

Poll close in 1:30

What should threads share?

- How many of the following memory elements should be shared by two threads in the same process?
  ① Stack section
  ② Data section
  ③ Text/code section
  ④ Page table
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Poll close in 1:30

What should threads share?

- How many of the following memory elements should be shared by two threads in the same process?

  ① Stack section

  ② Data section

  ③ Text/code section

  ④ Page table

  A. 0

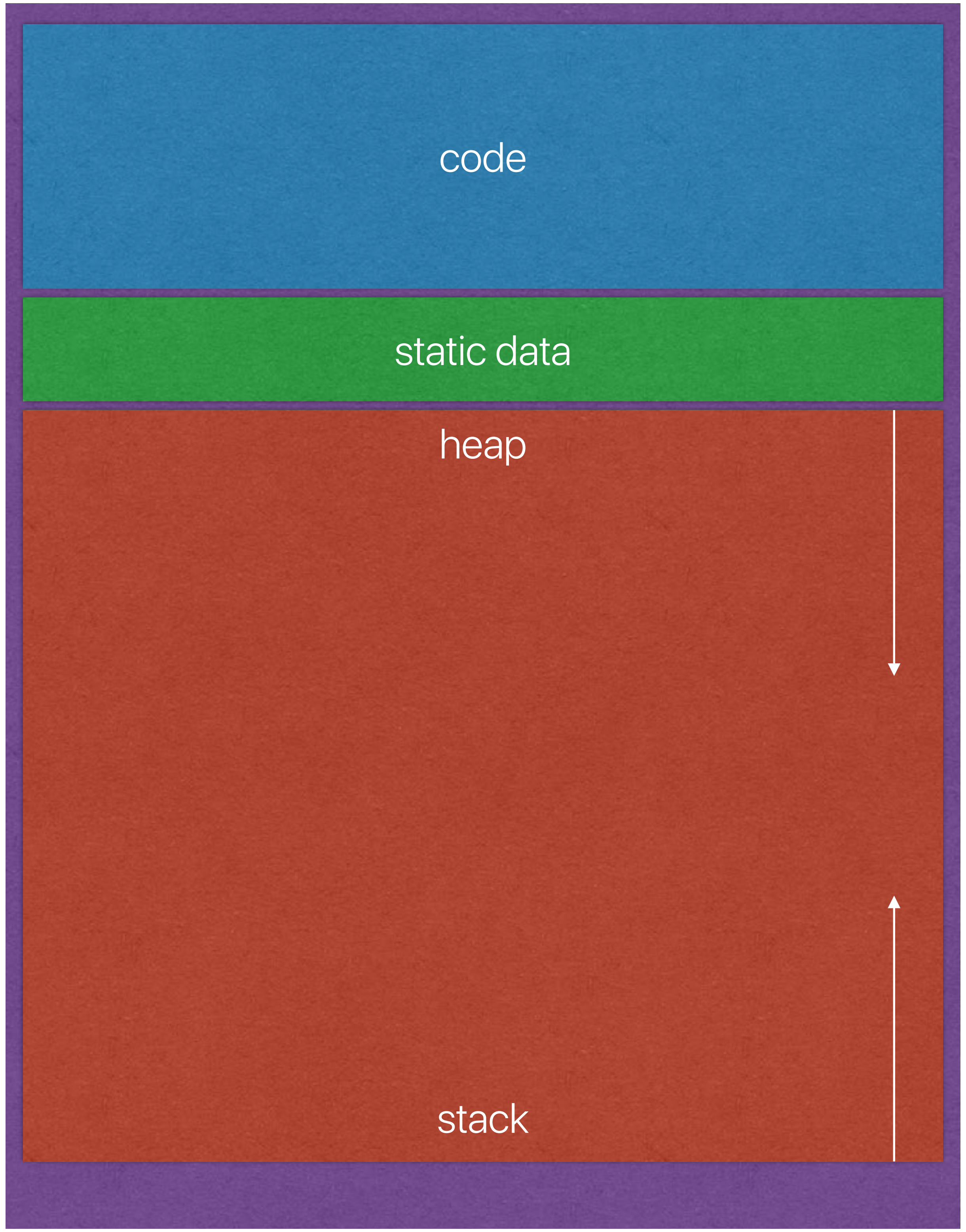
  B. 1

  C. 2

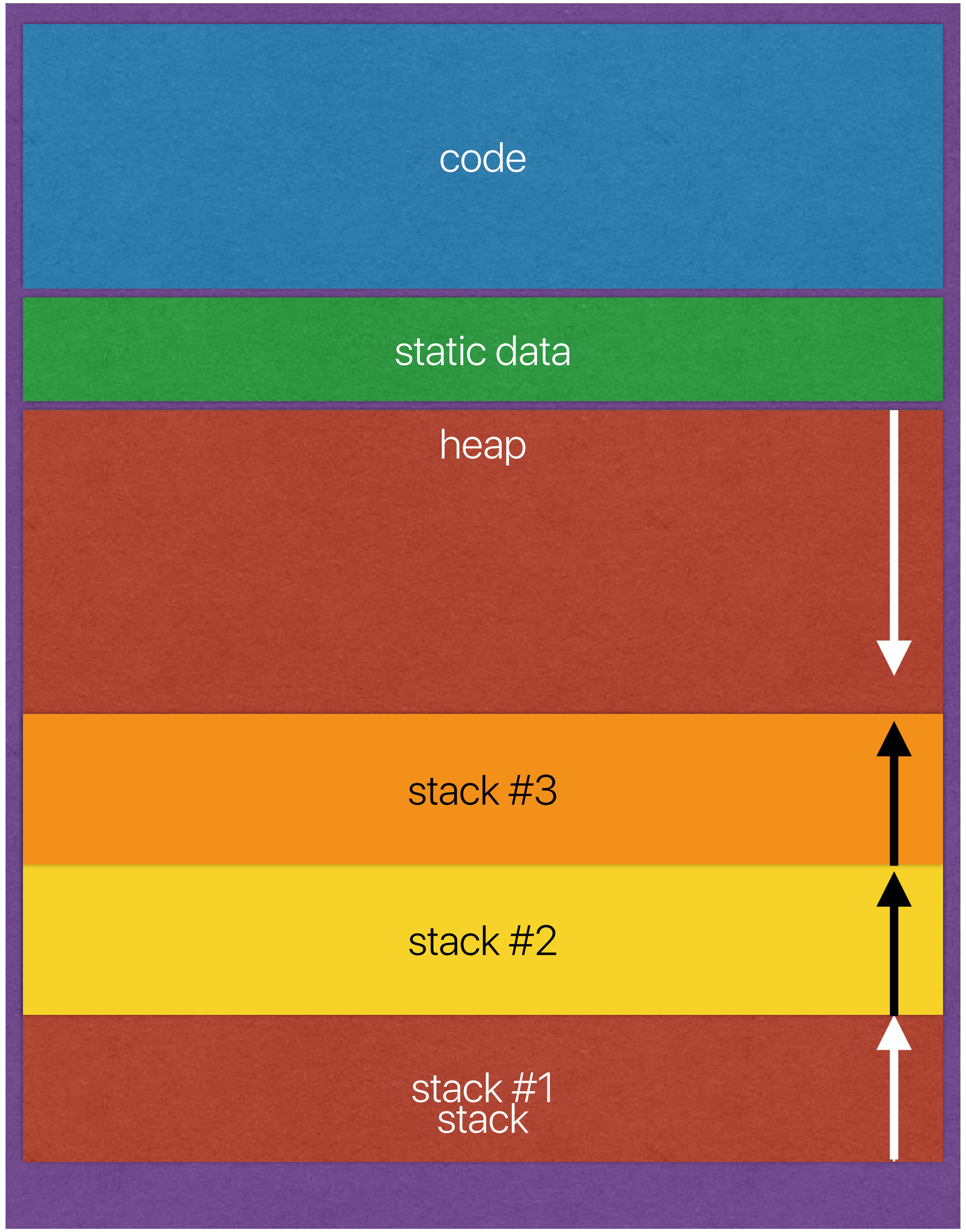
  D. 3

  E. 4

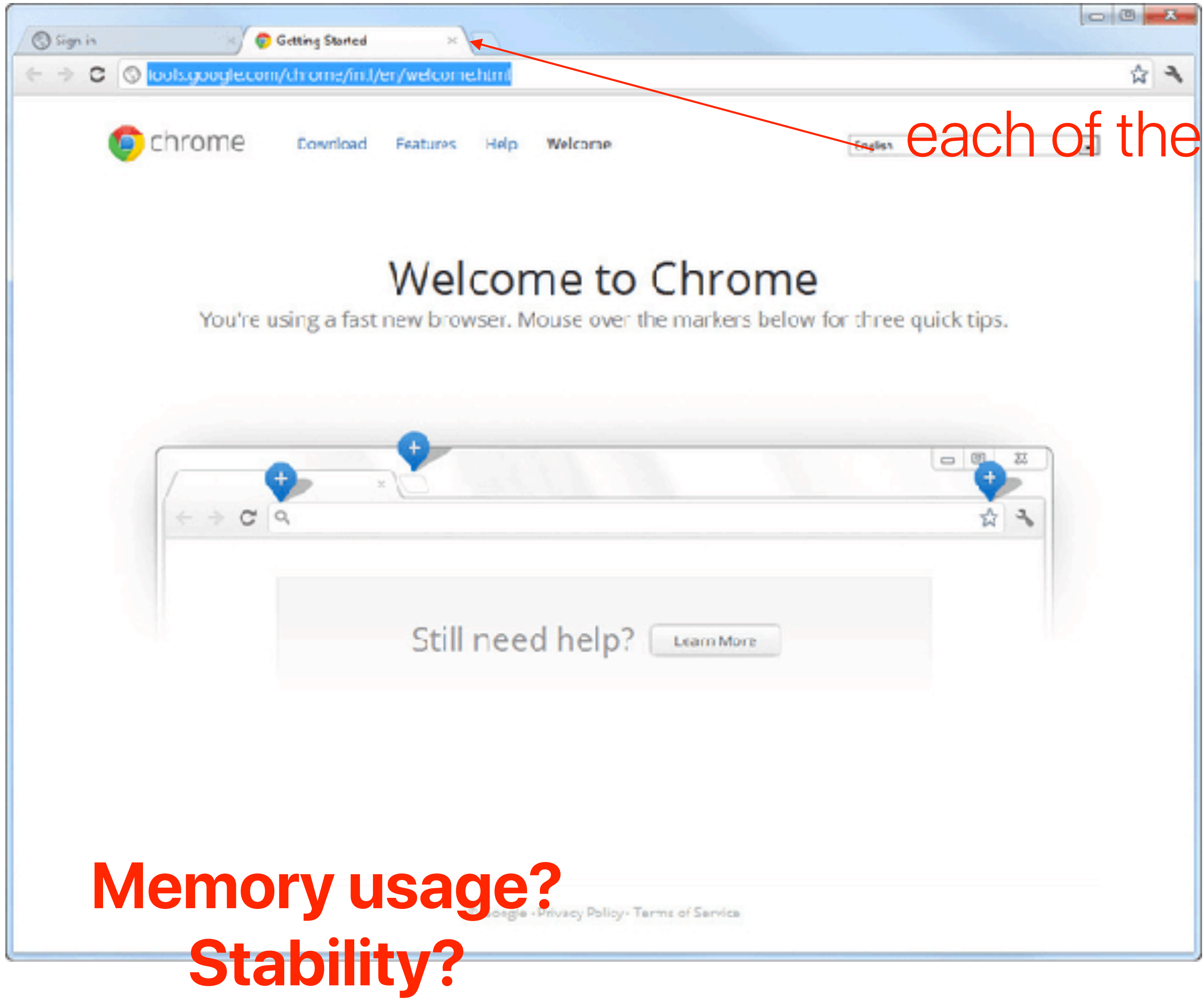
The virtual memory of single-threaded applications



The virtual memory of multithreaded applications



Case study: Chrome v.s. Firefox


Sign in
Getting Started
tools.google.com/chrome/intl/en/welcome.html
chrome
Download
Features
Help
Welcome
English

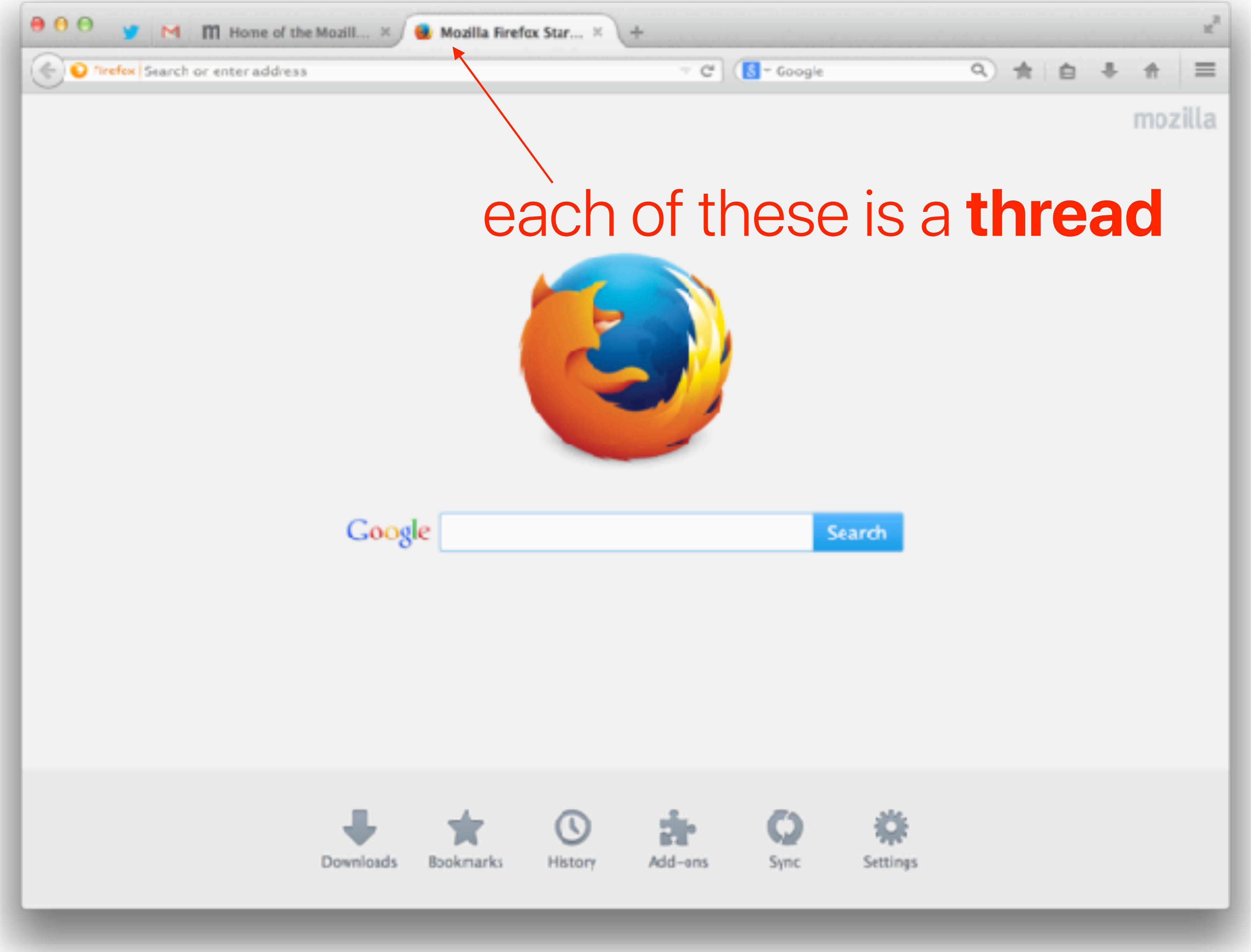
Welcome to Chrome

You're using a fast new browser. Mouse over the markers below for three quick tips.

Still need help?

Learn More
Google - Privacy Policy - Terms of Service

each of these is a **process**



each of these is a **thread**

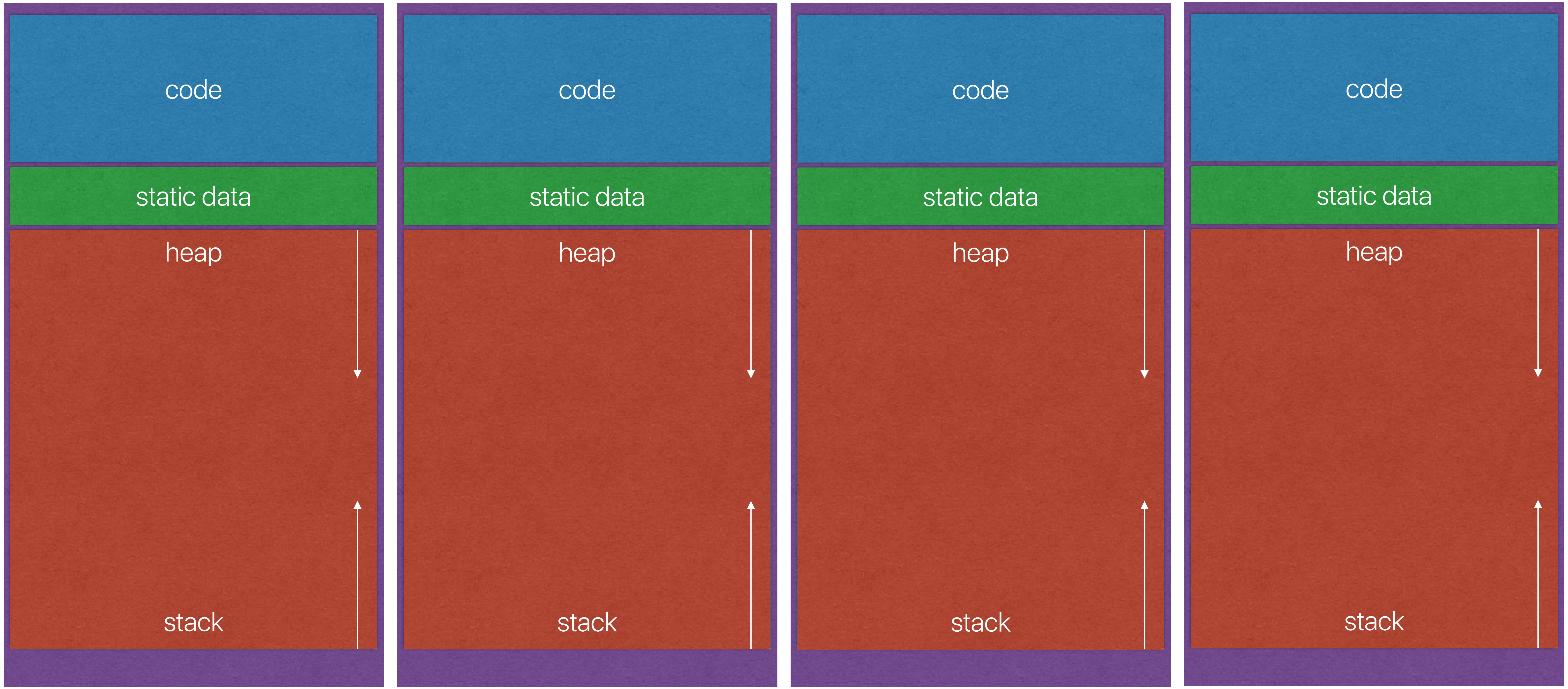
Memory usage?

Stability?

Security?

Latency?

Chrome



Tab #1

code

static data

heap

stack

Tab #2

code

static data

heap

stack

Tab #3

code

static data

heap

stack

Tab #4

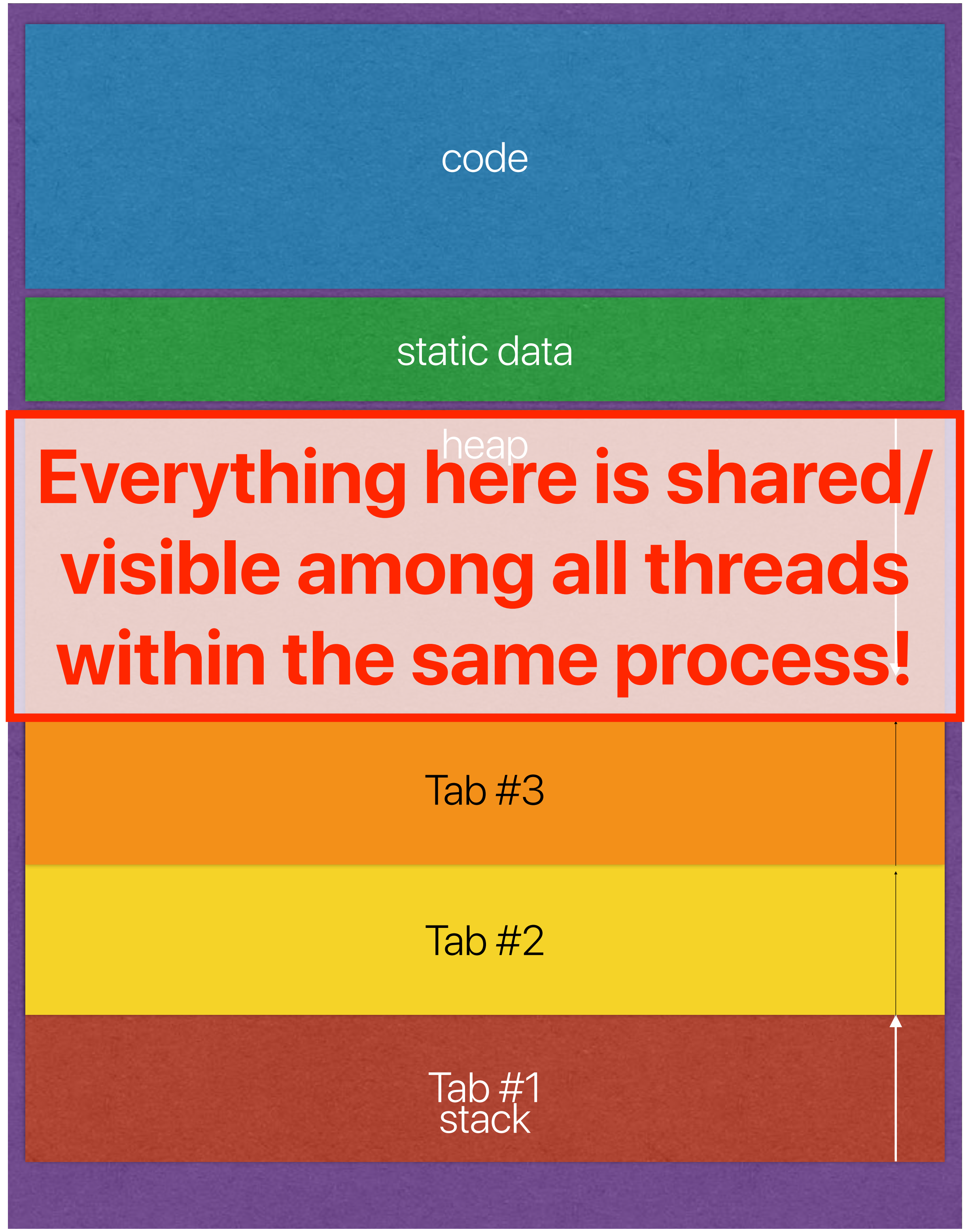
code

static data

heap

stack

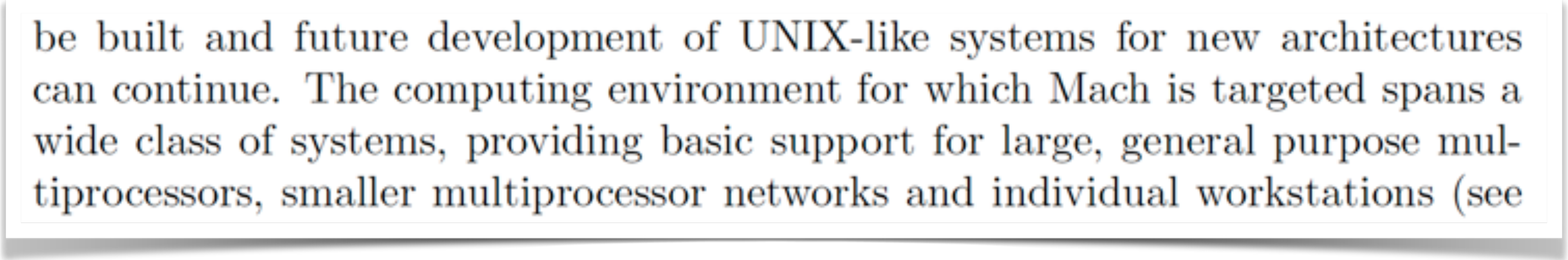
Firefox



Everything here is shared/ visible among all threads within the same process!

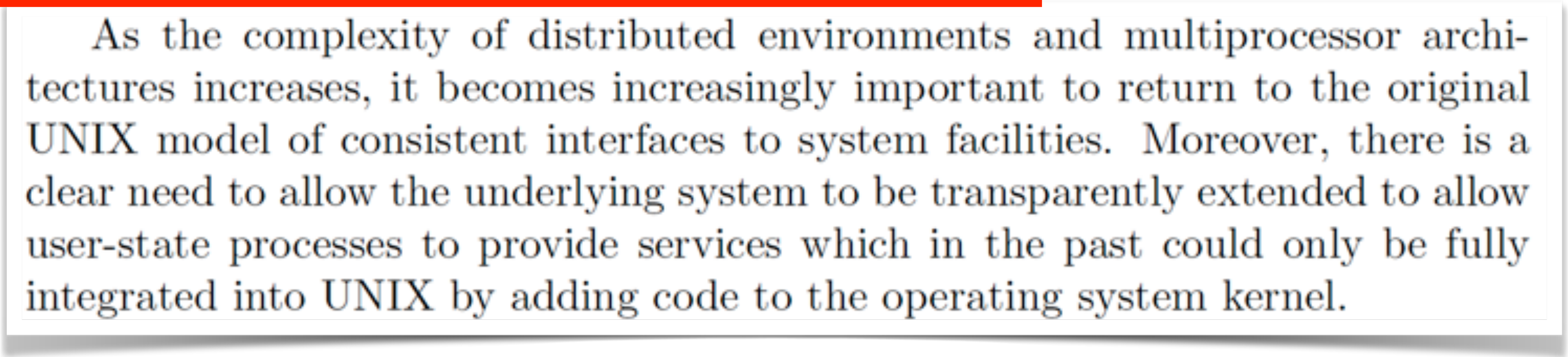
Why "Mach"?

- The hardware is changing
  - Multiprocessors
  - Networked computing



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

- The software
  - The demand of extending an OS easily
  - Repetitive but confusing mechanisms for similar stuffs



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.

Interprocess communication

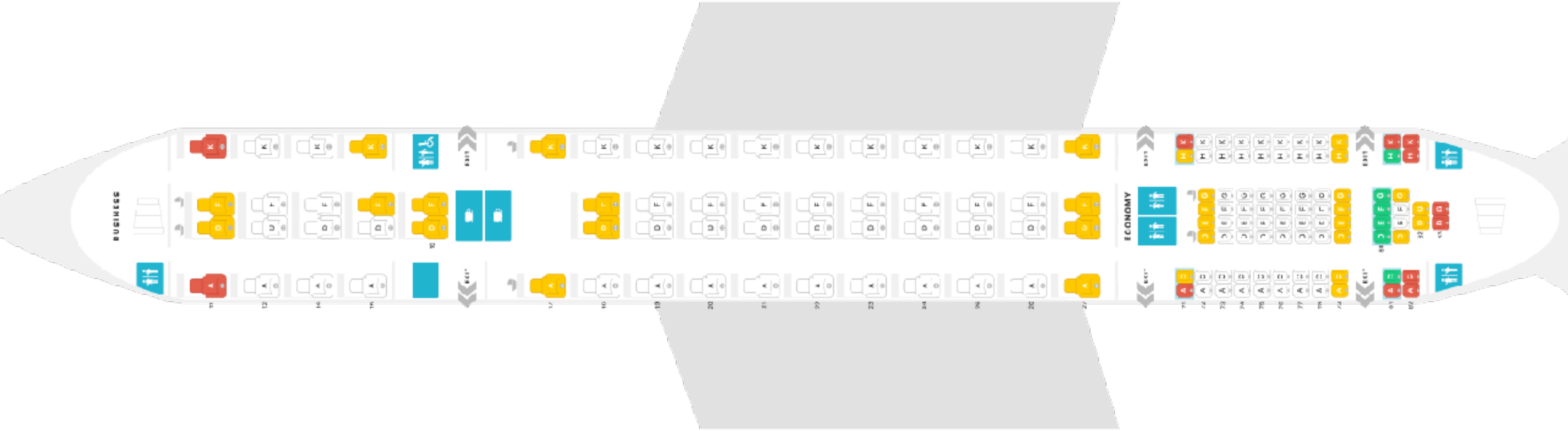
- UNIX provides a variety of mechanisms
  - Pipes
  - Pty’s
  - Signals
  - Sockets
- No protection
- No consistency
- Location dependent

Ports/Messages

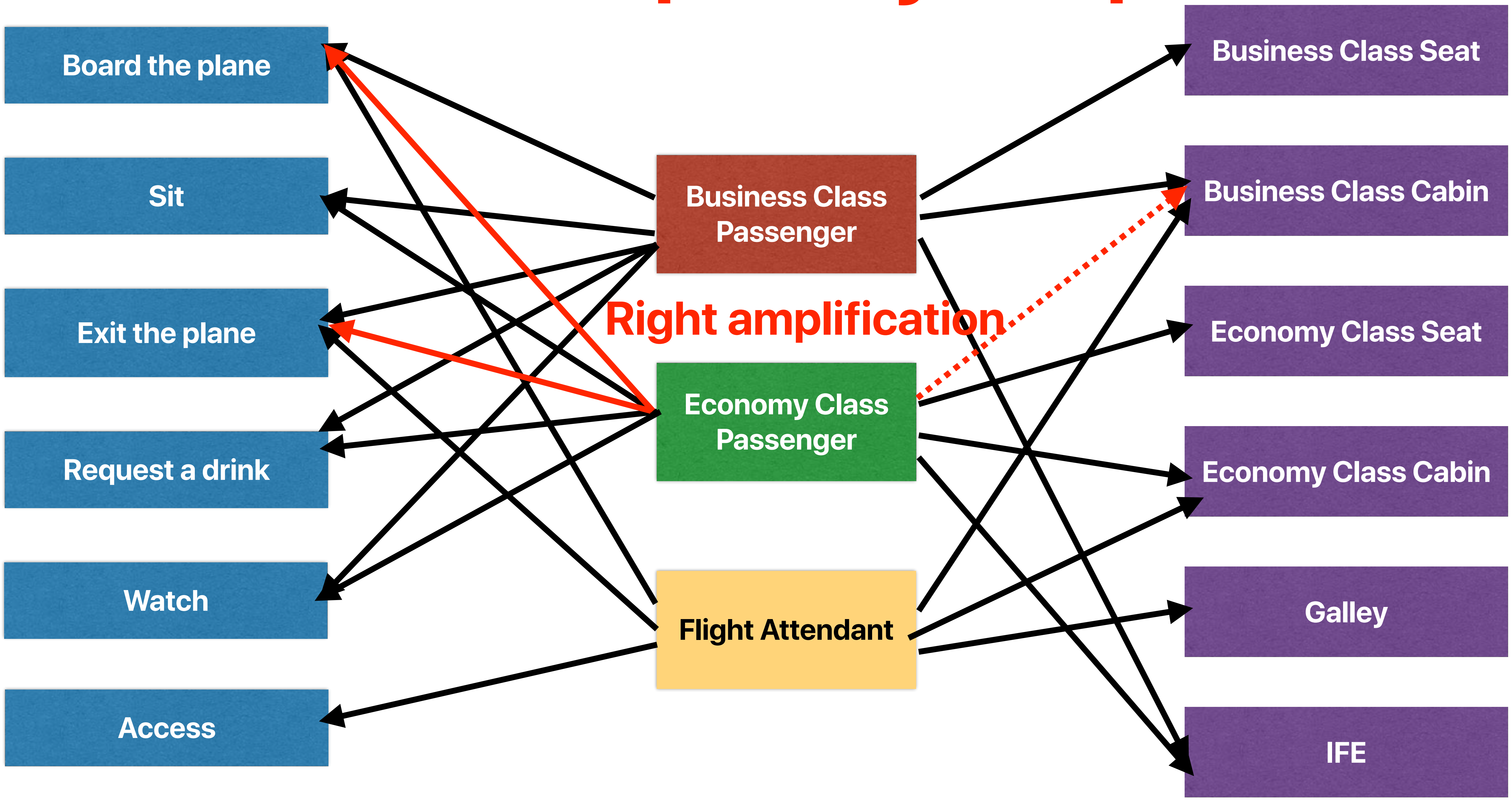
- Port is an abstraction of:
  - Message queues
  - Capability
- What do ports/messages promote?
  - Location independence — everything is communicating with ports/messages, no matter where it is

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


BUSINESS
ECONOMY
EXIT
EXIT
EXIT
EXIT
EXIT
EXIT

Capability in a plane



Board the plane

Sit

Exit the plane

Request a drink

Watch

Access

Business Class Passenger

Right amplification

Economy Class Passenger

Flight Attendant

Business Class Seat

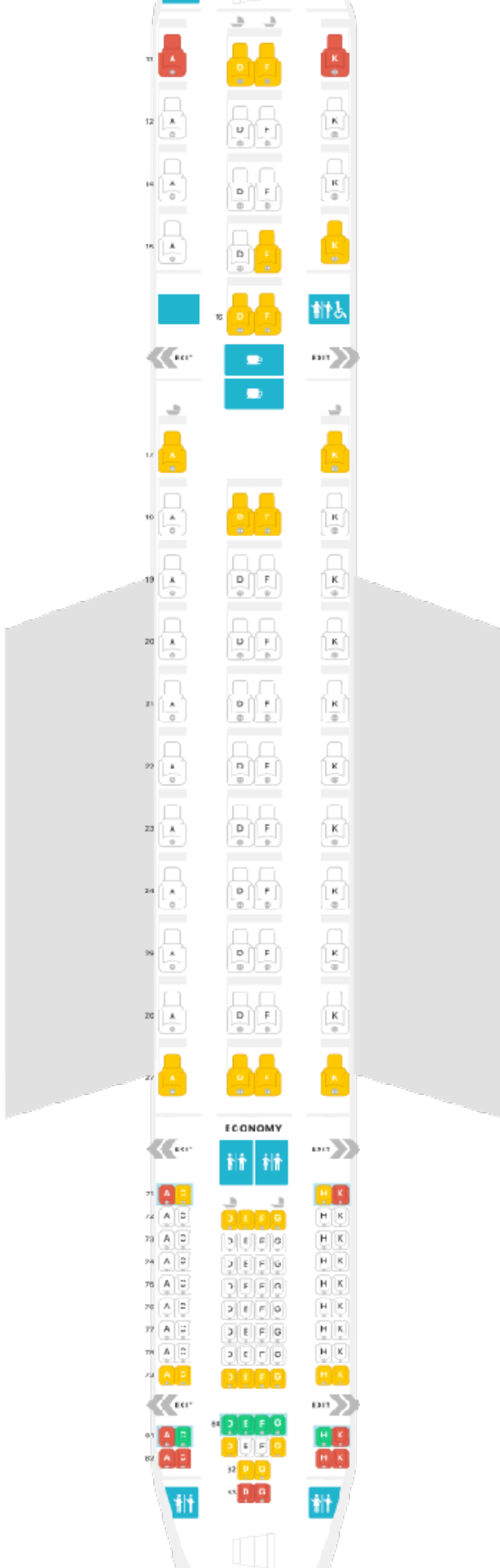
Business Class Cabin

Economy Class Seat

Economy Class Cabin

Galley

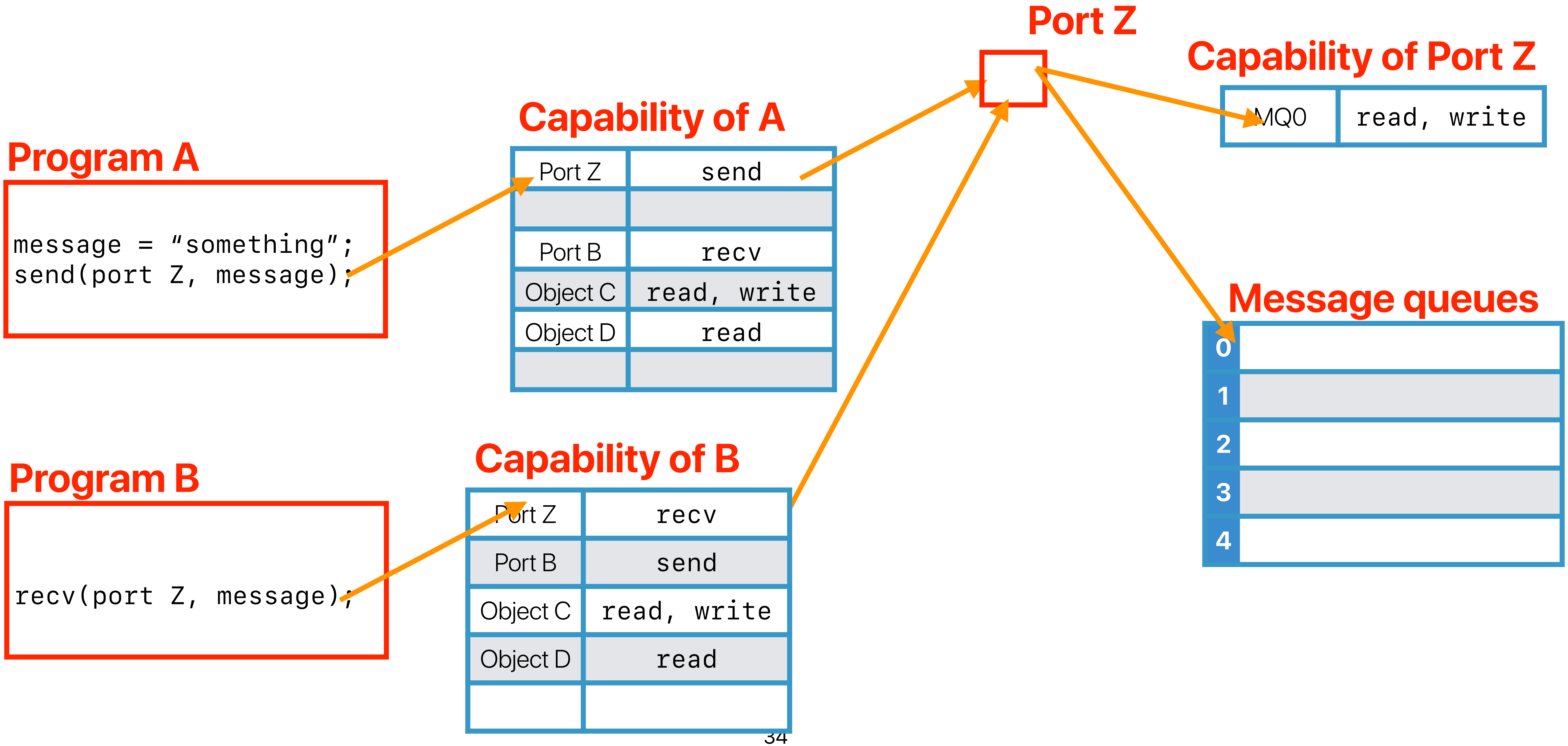
IFE



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

Ports/Messages



Program A

```
message = “something”;
send(port Z, message);
```

Program B

```
recv(port Z, message);
```

Capability of A

Port Z	send
Port B	recv
Object C	read, write
Object D	read

Capability of B

Port Z	recv
Port B	send
Object C	read, write
Object D	read

Port Z

Capability of Port Z

MQ0	read, write

Message queues

0	
1	
2	
3	
4	

```java
class JBT {

   int variable = 5;

   public static void main(String args[]) {
      JBT obj = new JBT();

      obj.method(20);
      obj.method();
   }

   void method(int variable) {
      variable = 10;
      System.out.println("Value of Instance variable :" + this.variable);
      System.out.println("Value of Local variable :" + variable);
   }

   void method() {
      int variable = 40;
      System.out.println("Value of Instance variable :" + this.variable);
      System.out.println("Value of Local variable :" + variable);
   }
}
```

Poll close in 1:30

What's in the kernel?

- How many of the following Mach features/functions are implemented in the kernel?
  ① I/O device drivers
  ② File system
  ③ Shell
  ④ Virtual memory management
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

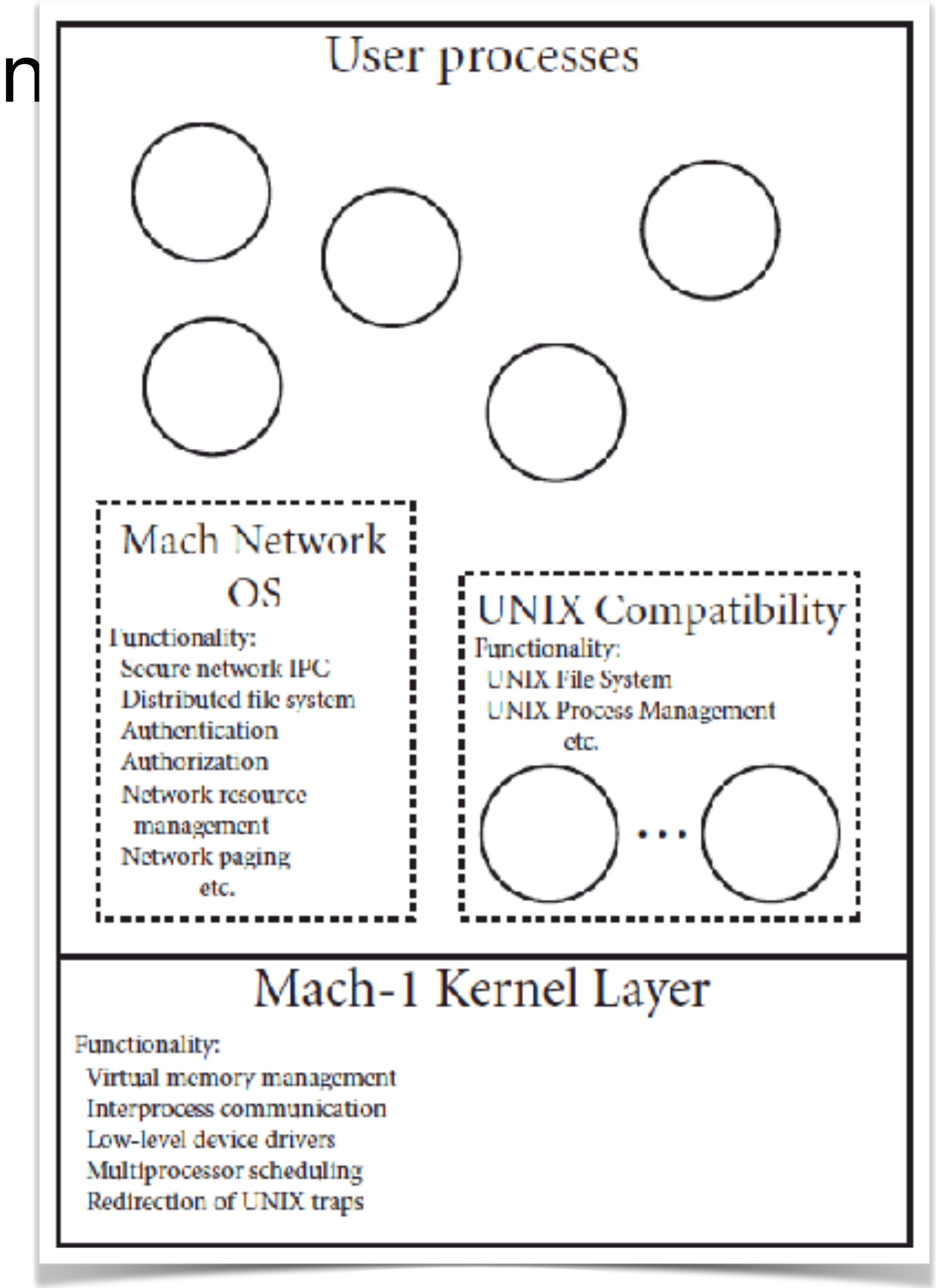
Poll close in 1:30

What's in the kernel?

- How many of the following Mach features/functions are implemented in the kernel?
  ① I/O device drivers
  ② File system
  ③ Shell
  ④ Virtual memory management
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

What's in the kernel?

- How many of the following Mach features/fun implemented in the kernel?
  ① I/O device drivers
  ② File system
  ③ Shell
  ④ Virtual memory management
  A. 0
  B. 1
  [✓ C. 2]
  D. 3
  E. 4



User processes

Mach Network OS
Functionality:
Secure network IPC
Distributed file system
Authentication
Authorization
Network resource management
Network paging
etc.

UNIX Compatibility
Functionality:
UNIX File System
UNIX Process Management
etc.

...

Mach-1 Kernel Layer
Functionality:
Virtual memory management
Interprocess communication
Low-level device drivers
Multiprocessor scheduling
Redirection of UNIX traps

Poll close in 1:30

Types of kernels

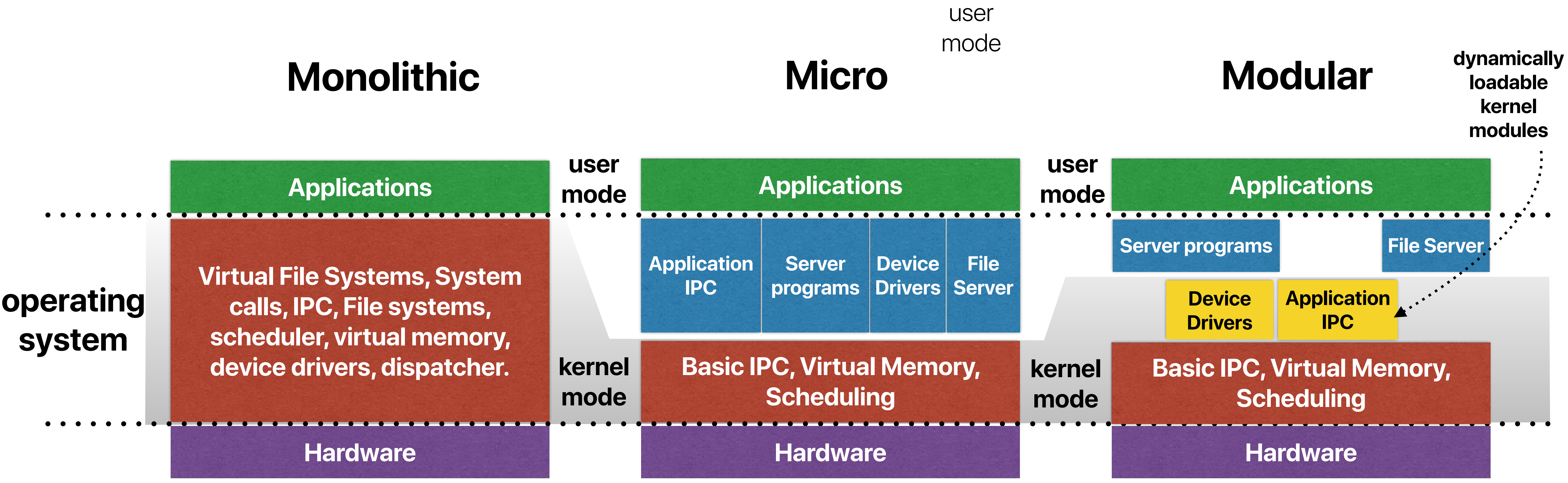
- What type of kernels does the UNIX described in Dennis M. Ritchie's paper belong to?
  A. Microkernel — the kernel only provides a minimal set of services including memory management, multitasking and inter-process communication
  B. Monolithic — the kernel implements every function that cannot be in a user-space library: device drivers, scheduler, memory handling, file systems, network stacks
  C. Modular — the kernel provides a basic set of functions like microkernels, but allows load/unload kernel modules if necessary
  D. Layered kernel — the kernel follows strict layered design that lower-order module cannot interact with higher-order modules

Poll close in 1:30

Types of kernels

- What type of kernels does the UNIX described in Dennis M. Ritchie's paper belong to?
  A. Microkernel — the kernel only provides a minimal set of services including memory management, multitasking and inter-process communication
  B. Monolithic — the kernel implements every function that cannot be in a user-space library: device drivers, scheduler, memory handling, file systems, network stacks
  C. Modular — the kernel provides a basic set of functions like microkernels, but allows load/unload kernel modules if necessary
  D. Layered kernel — the kernel follows strict layered design that lower-order module cannot interact with higher-order modules

Types of Kernels



Original UNIX

Hydra, Mach

Linux, Windows, MacOS

Types of kernels

- What type of kernels does the UNIX described in Dennis M. Ritchie's paper belong to?

  A. Microkernel — the kernel only provides a minimal set of services including memory management, multitasking and inter-process communication **Hydra, Mach**

  B. Monolithic — the kernel implements every function that cannot be in a user-space library: device drivers, scheduler, memory handling, file systems, network stacks **Old UNIX**

  C. Modular — the kernel provides a basic set of functions like microkernels, but allows load/unload kernel modules if necessary **Linux, Windows, MacOS, FreeBSD**

  D. Layered kernel — the kernel follows strict layered design that lower-order module cannot interact with higher-order modules **THE**

Poll close in 1:30

Why not microkernels?

- Although Mach's design strongly influenced modern operating systems, why most modern operating systems do not adopt the design of microkernels?
  A. Microkernels are more difficult to extend than monolithic kernels
  B. Microkernels are more difficult to maintain than monolithic kernels
  C. Microkernels are less stable than monolithic kernels
  D. Microkernels are not as competitive as monolithic kernels in terms of application performance
  E. Microkernels are less flexible than monolithic kernels

Poll close in 1:30

Why not microkernels?

- Although Mach's design strongly influenced modern operating systems, why most modern operating systems do not adopt the design of microkernels?
  A. Microkernels are more difficult to extend than monolithic kernels
  B. Microkernels are more difficult to maintain than monolithic kernels
  C. Microkernels are less stable than monolithic kernels
  D. Microkernels are not as competitive as monolithic kernels in terms of application performance
  E. Microkernels are less flexible than monolithic kernels

Why not microkernels?

- Although Mach’s design strongly influenced modern operating systems, why most modern operating systems do not adopt the design of microkernels?

A. Microkernels are more difficult to extend than monolithic kernels

B. Microkernels are more difficult to maintain than monolithic kernels

C. Microkernels are less stable than monolithic kernels

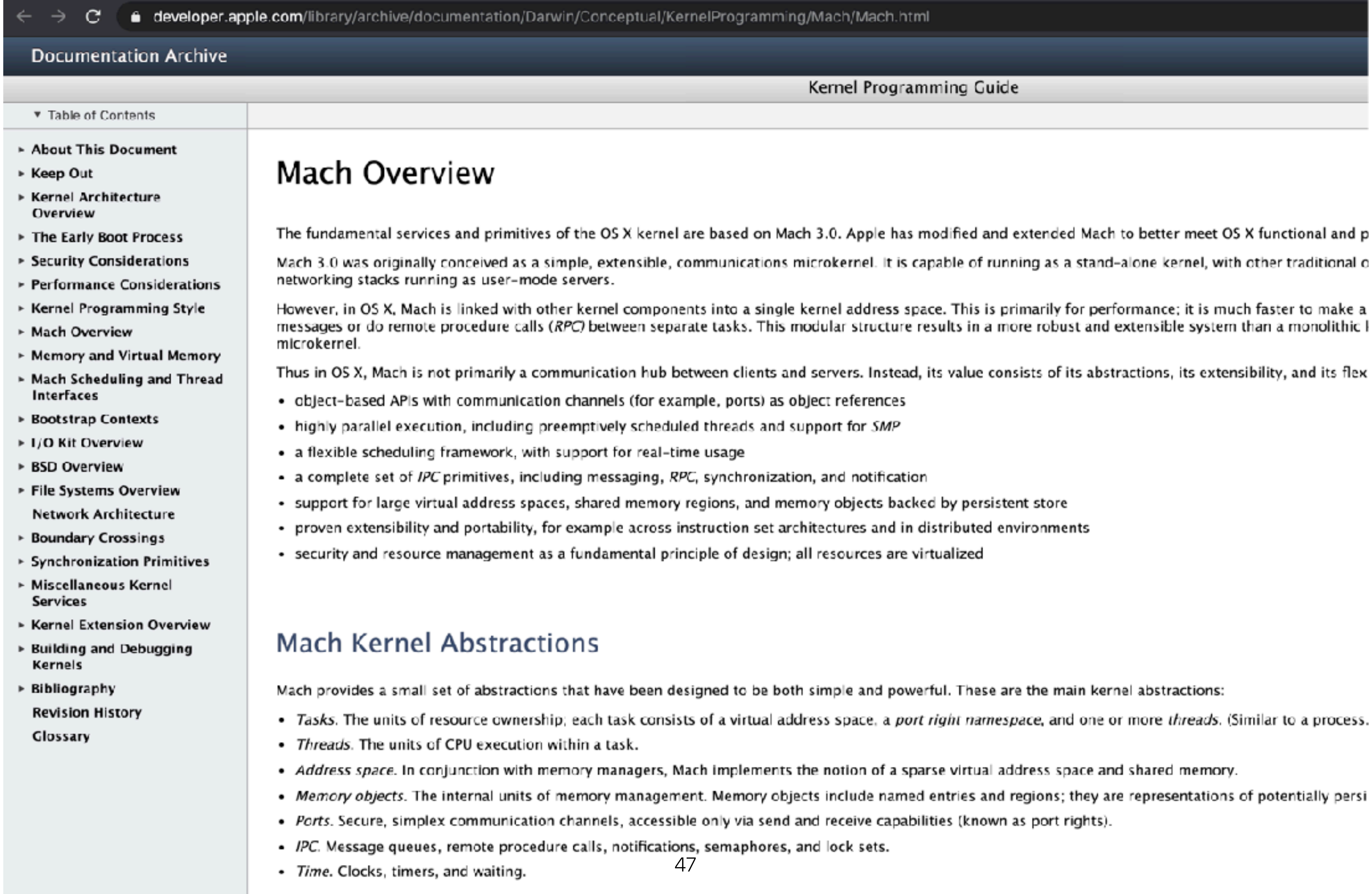
[✓ D. Microkernels are not as competitive as monolithic kernels in terms of application performance]

Context switches!

E. Microkernels are less flexible than monolithic kernels

The impact of Mach

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


developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/Mach/Mach.html
Documentation Archive
Kernel Programming Guide
Table of Contents

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 l 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

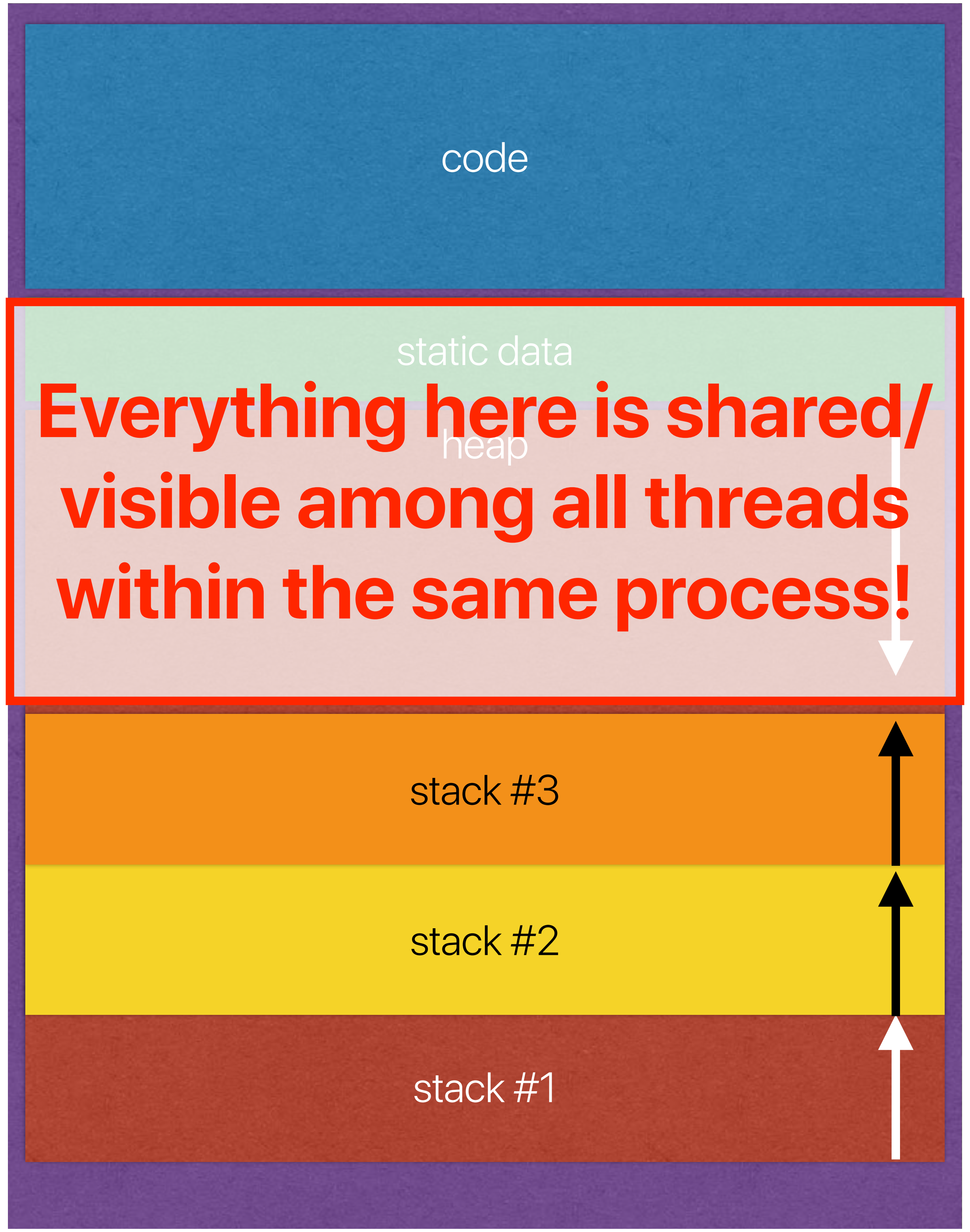
Mach Kernel Abstractions

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.

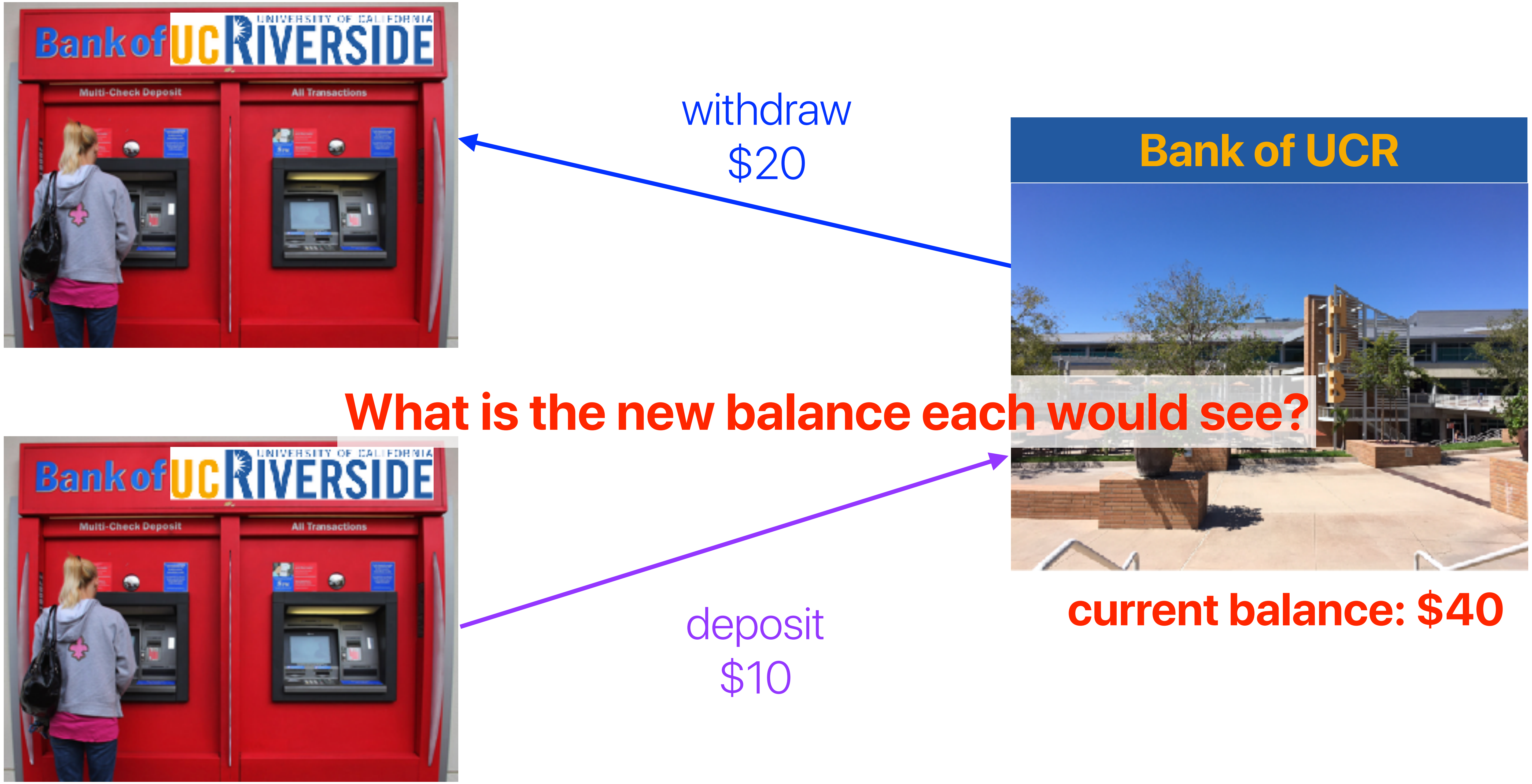
Thread programming & synchronization

The virtual memory of multithreaded applications



Everything here is shared/ visible among all threads within the same process!

Joint Banking



withdraw
$20

Bank of UCR

What is the new balance each would see?

deposit
$10

current balance: $40

Poll close in 1:30

Joint Banking

- If the shared variable, balance, initially has the value of 40, what value(s) might it hold after threads A and B finish after we call `deposit(10)` and `withdraw(20)`?

A. 30

B. 20 or 30

C. 20, 30, or 50

D. 10, 20, or 30

Thread A

```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Thread B

```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Poll close in 1:30

Joint Banking

- If the shared variable, balance, initially has the value of 40, what value(s) might it hold after threads A and B finish after we call `deposit(10)` and `withdraw(20)`?

Thread A

```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Thread B

```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

A. 30
B. 20 or 30
C. 20, 30, or 50
D. 10, 20, or 30

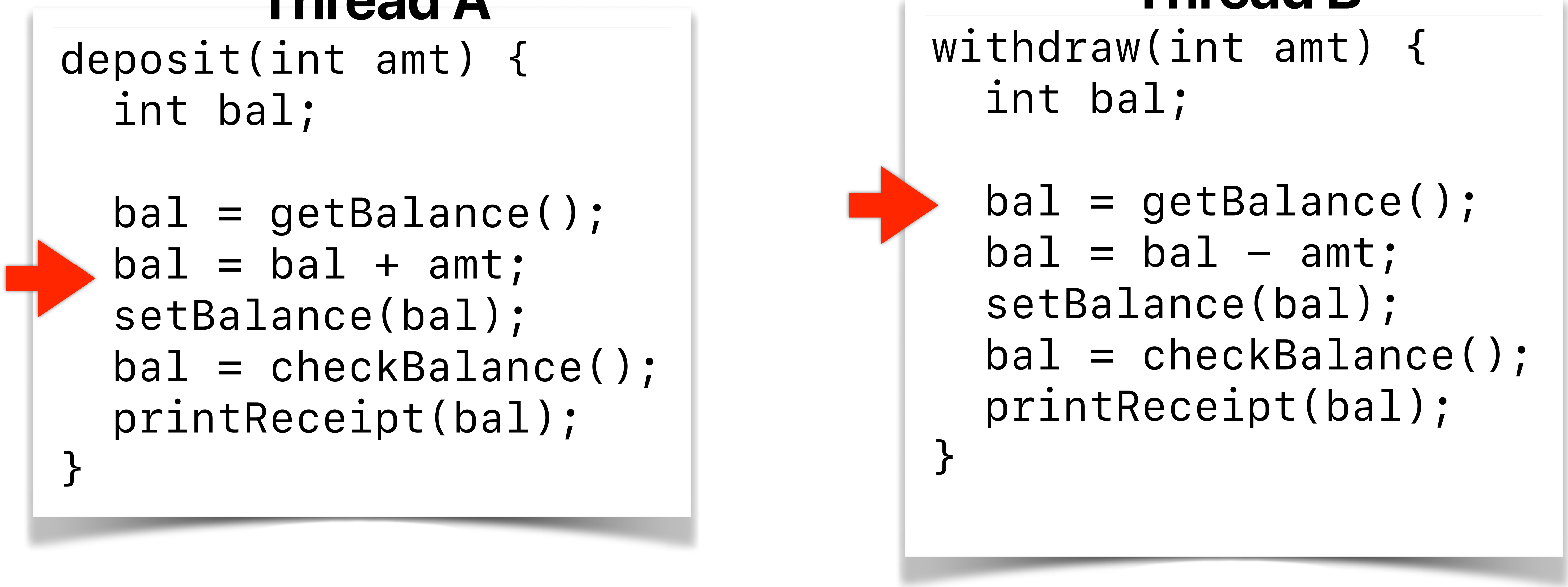
Bank Account Race Condition

current balance: $50

Thread A

current balance: $40

Thread B



```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Step 1: T_A runs and calls `getBalance` followed by adding `amt` (10) to `bal`

Bank Account Race Condition

current balance: $50

Thread A

```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

current balance: $20

Thread B

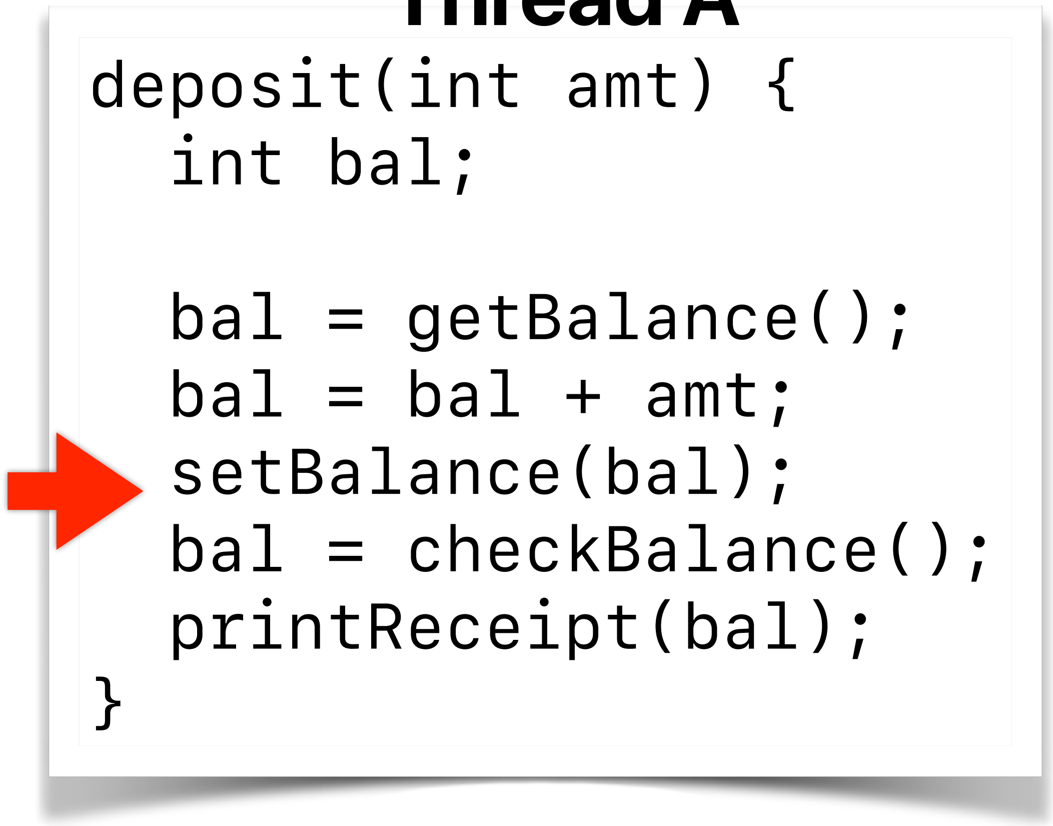
```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Step 2: Switch to T_B which calls `getBalance`, followed by subtracting `amt` (20) from `bal`

Bank Account Race Condition

current balance: $50

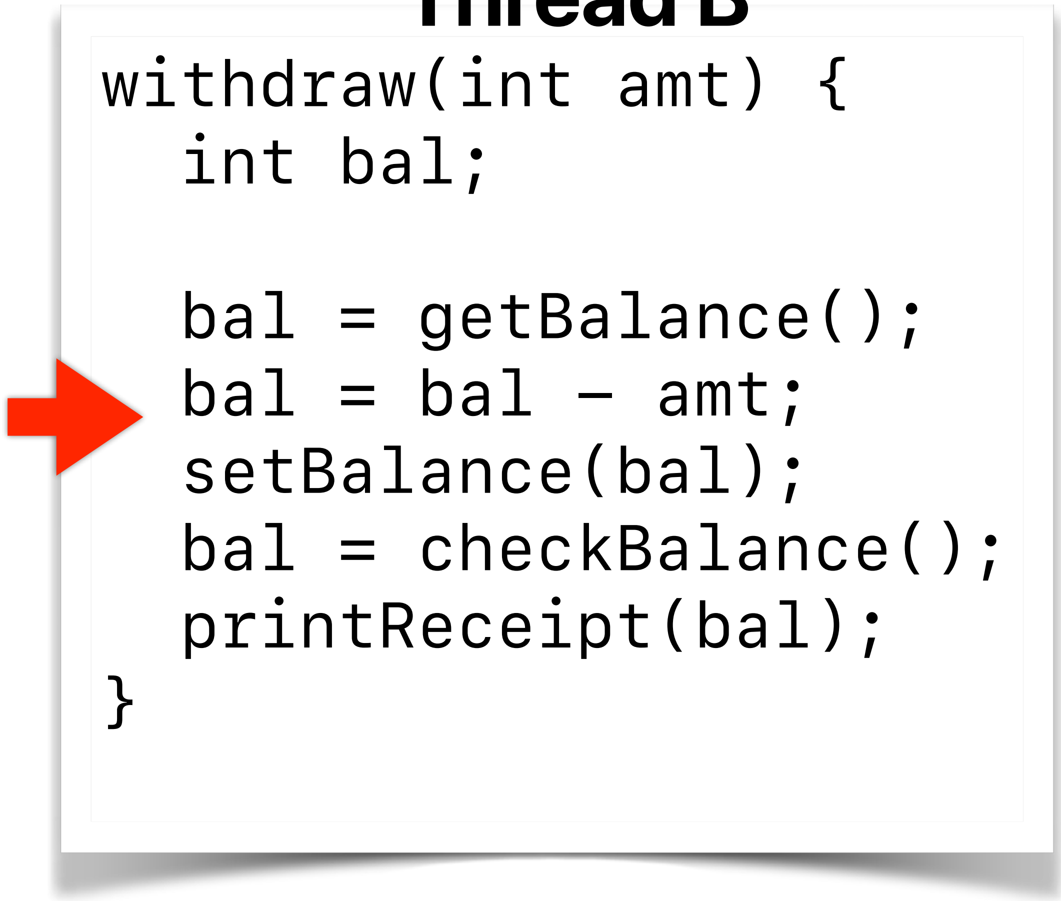
Thread A



```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

current balance: $20

Thread B



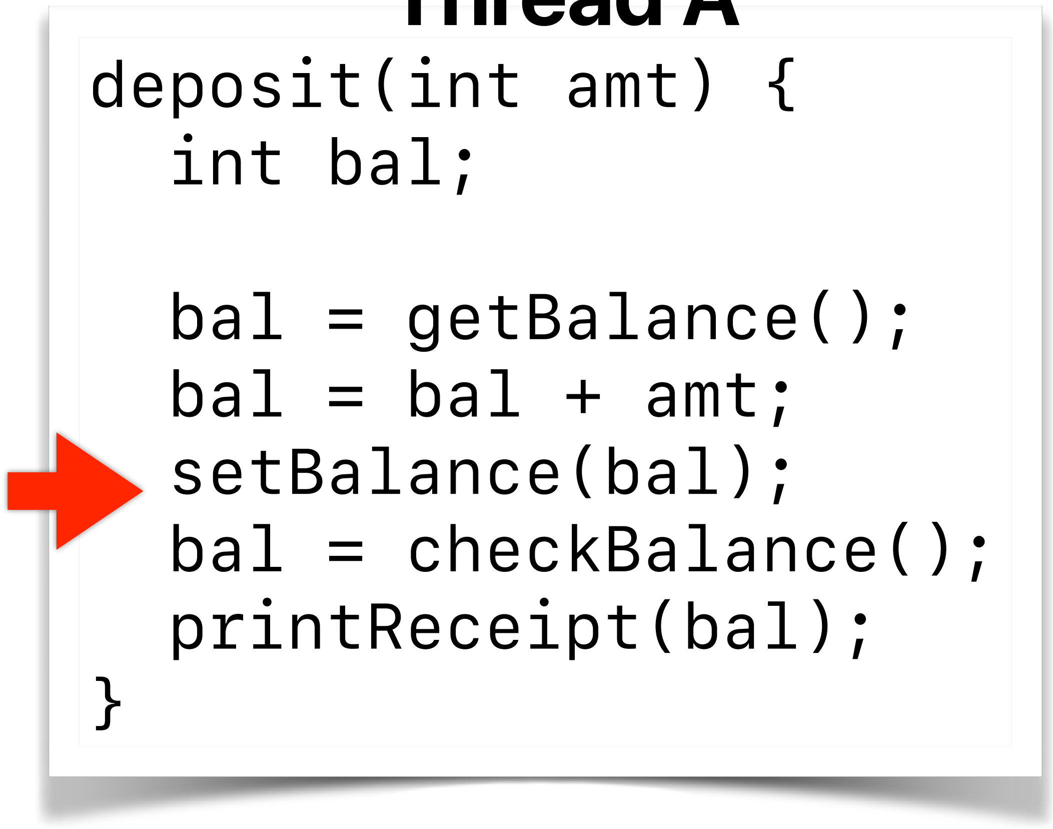
```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Step 3: Switch back to T_A which calls `setBalance`

Bank Account Race Condition

current balance: $50

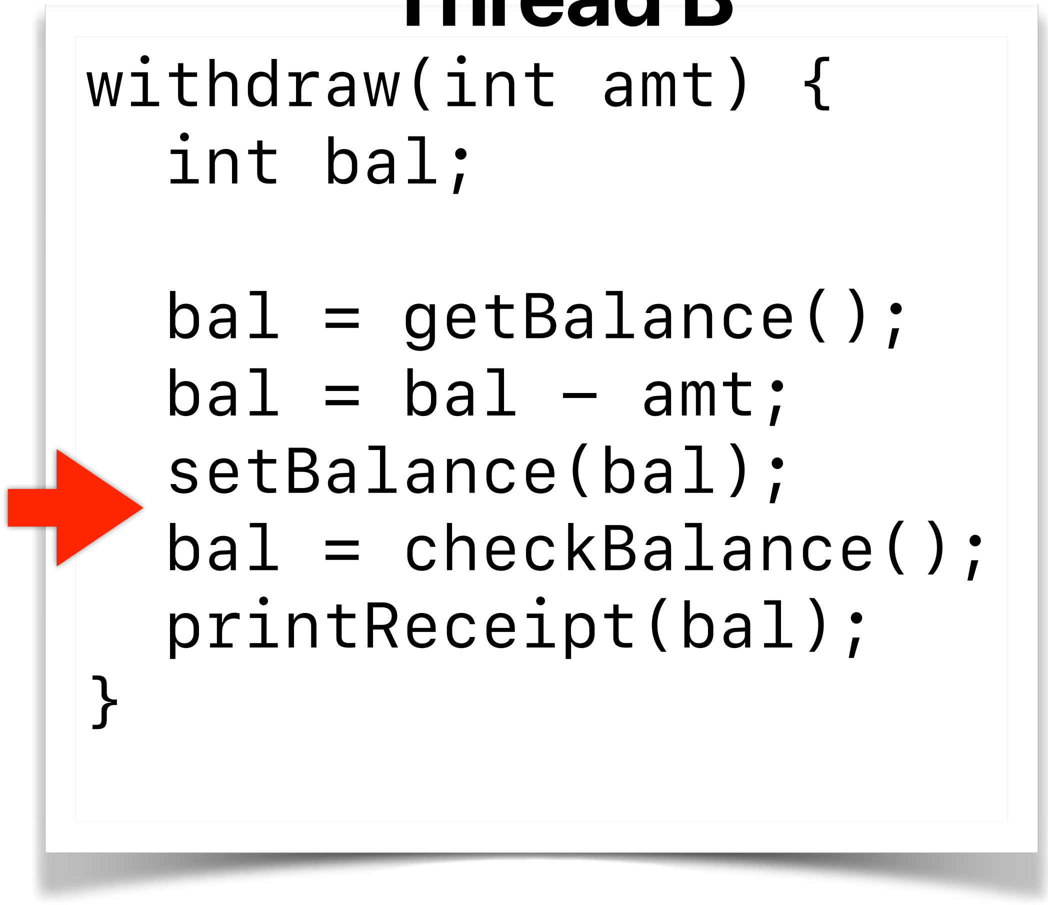
Thread A



```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

current balance: $20

Thread B



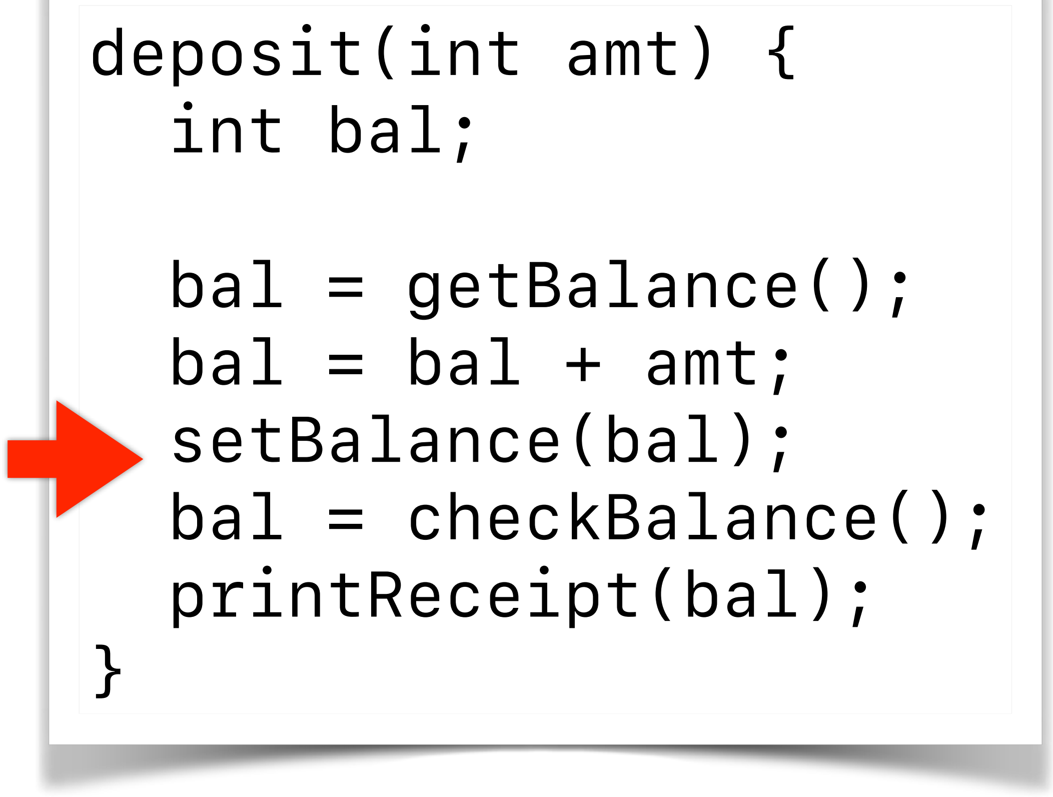
```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Step 4: Switch back to and finish T_B by calling `setBalance`, followed by `printReceipt`

Bank Account Race Condition

current balance: $20

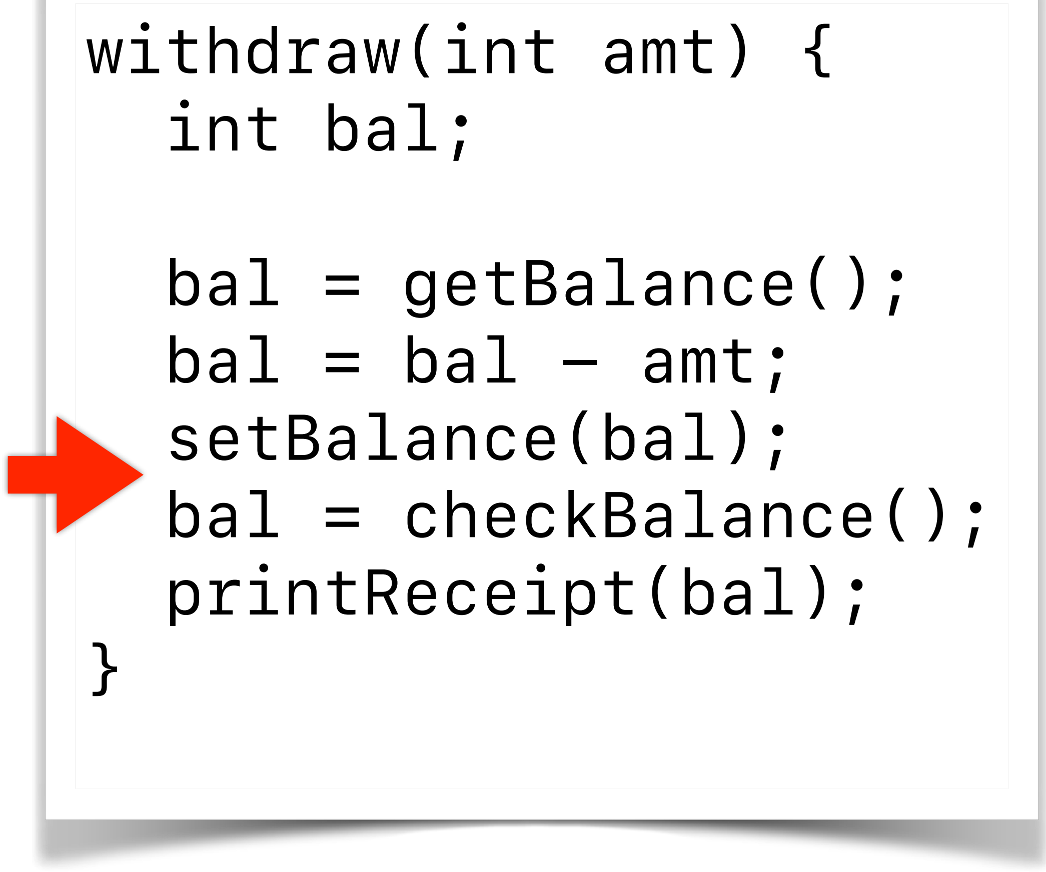
Thread A



```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

current balance: $20

Thread B



```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Step 5: Finish T_A by calling `checkBalance`, followed by `printReceipt`

Honey, we need to chat

Joint Banking

- If the shared variable, balance, initially has the value of 40, what value(s) might it hold after threads A and B finish after we call `deposit(10)` and `withdraw(20)`?

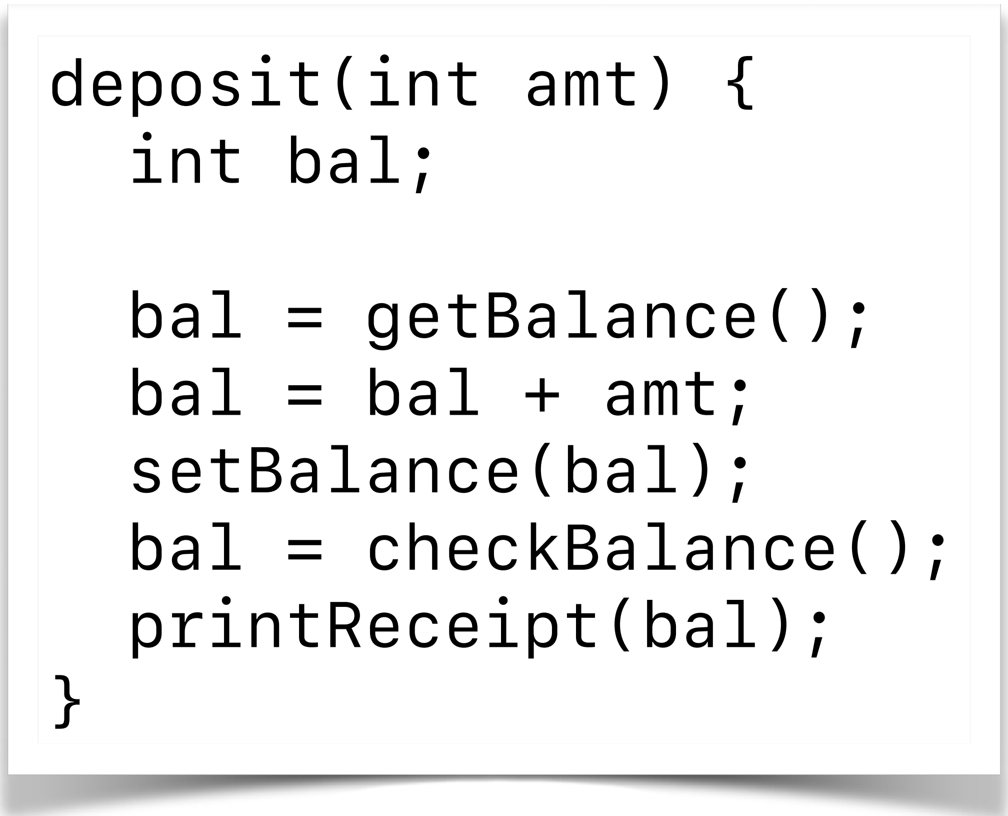
A. 30

B. 20 or 30

[✓ C. 20, 30, or 50]

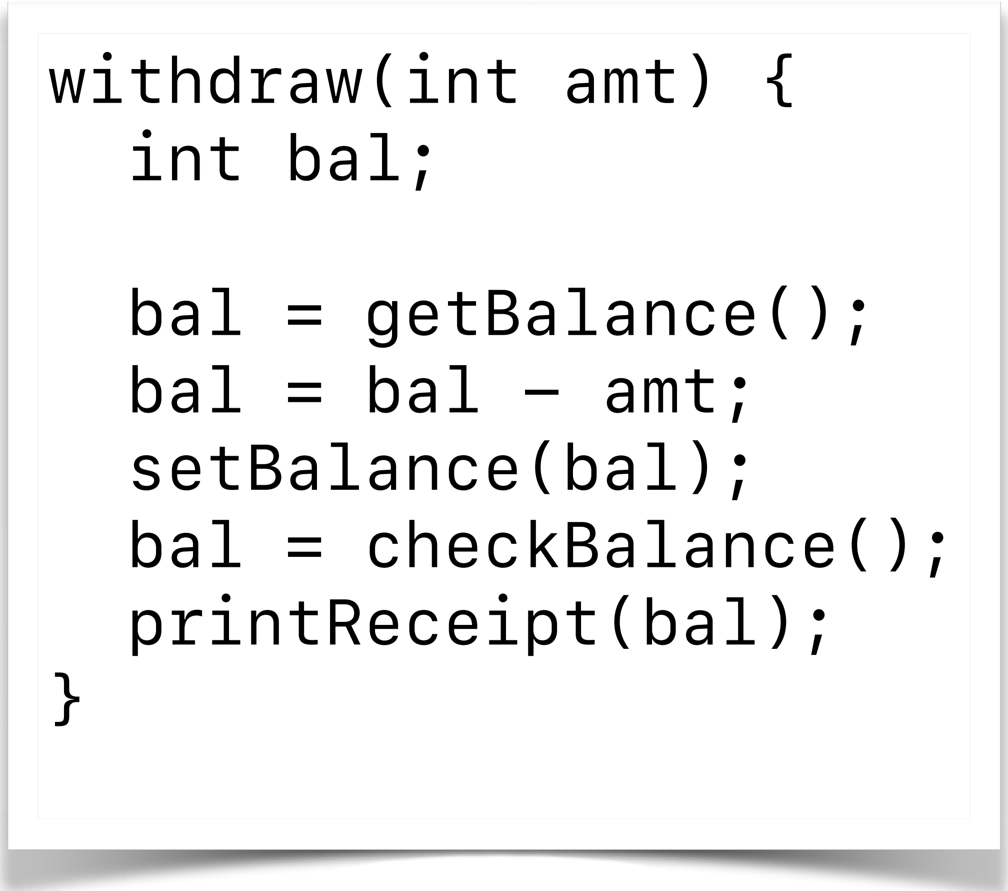
D. 10, 20, or 30

Thread A



```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Thread B



```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Processors/threads are not-deterministic

- Processor/compiler may reorder your memory operations/instructions
- Each processor core may not run at the same speed (cache misses, branch mis-prediction, I/O, voltage scaling and etc..)
- Threads may not be executed/scheduled right after it's spawned

Synchronization

- Concurrency leads to multiple active processes/threads that share one or more resources
- Synchronization involves the orderly sharing of resources
- All threads must be on the same page
- Need to avoid race conditions

Critical sections

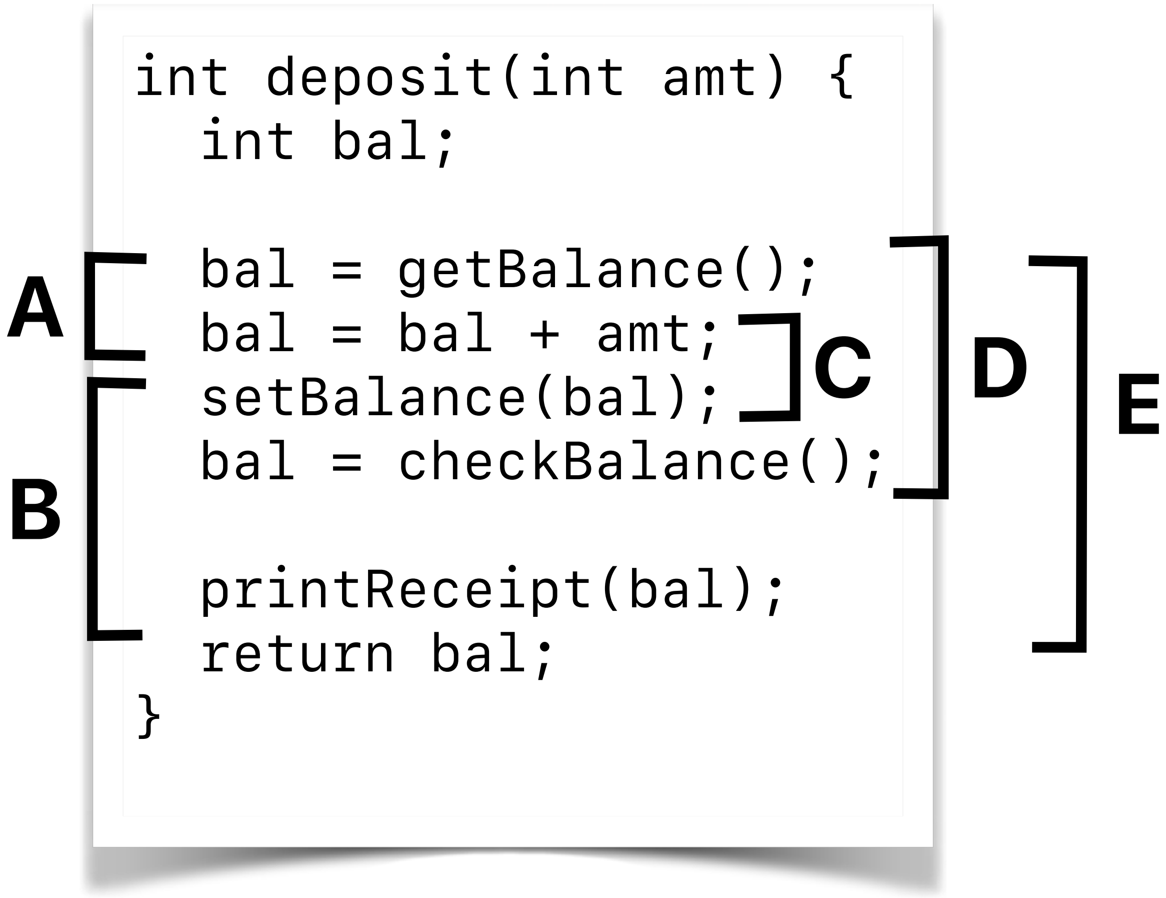
- Protect some pieces of code that access shared resources (memory, device, etc.)
- For safety, critical sections should:
  - Enforce mutual exclusion (i.e. only one thread at a time)
  - Execute atomically (all-or-nothing) before allowing another thread

Poll close in 1:30

Identifying Critical Sections

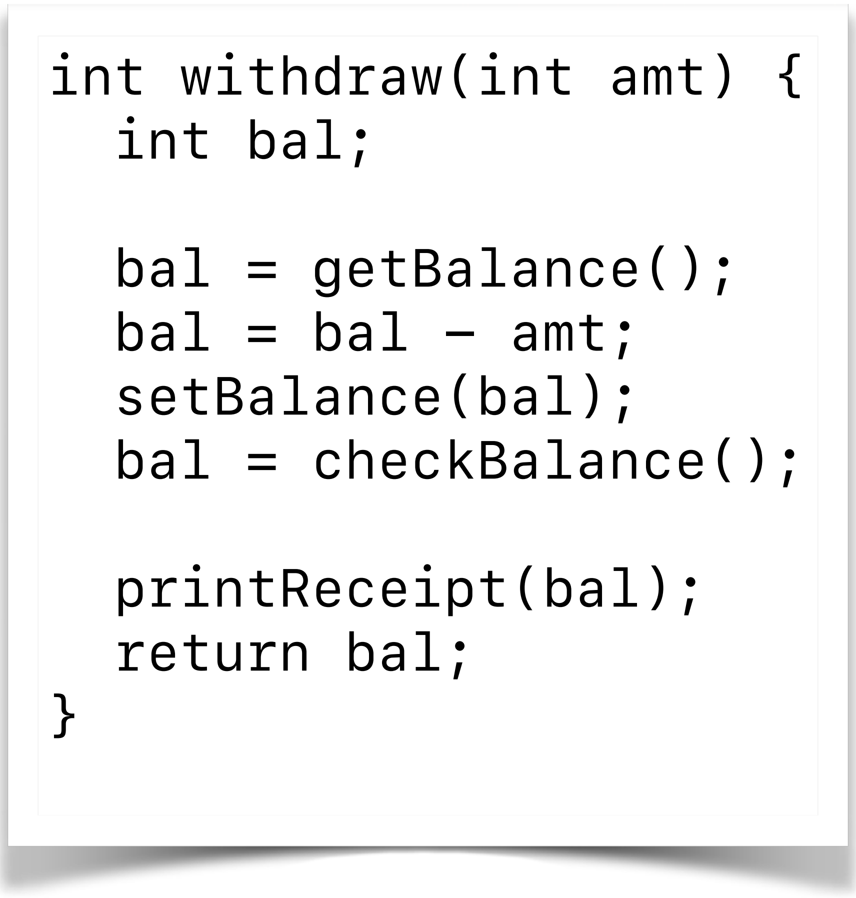
- Which is the smallest code region in Thread A that we can make as a critical section to guarantee the outcome as 30?

Thread A



```
int deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();

  printReceipt(bal);
  return bal;
}
```

Thread B

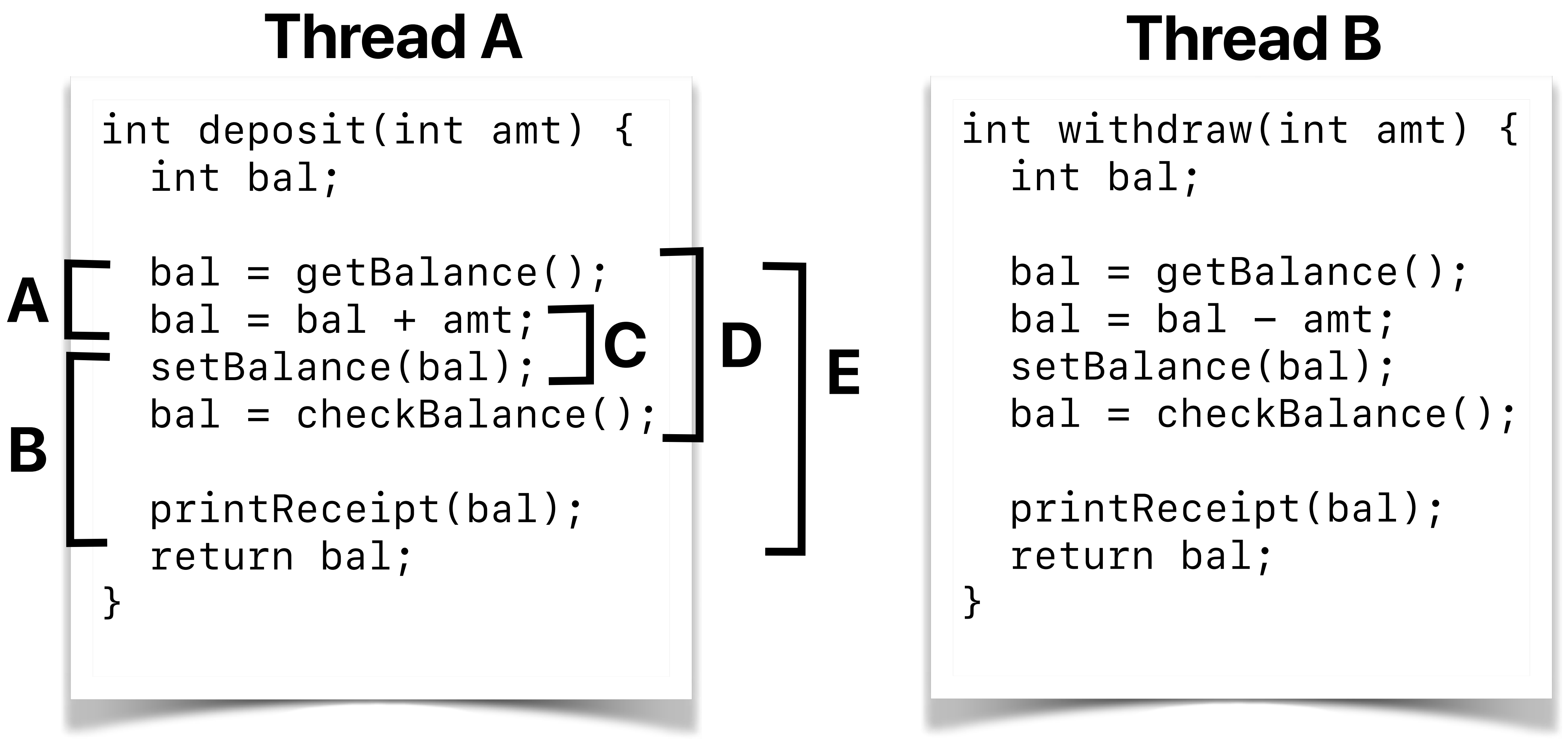


```
int withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();

  printReceipt(bal);
  return bal;
}
```

Poll close in 1:30

Identifying Critical Sections

- Which is the smallest code region in Thread A that we can make as a critical section to guarantee the outcome as 30?



Thread A

```
int deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();

  printReceipt(bal);
  return bal;
}
```

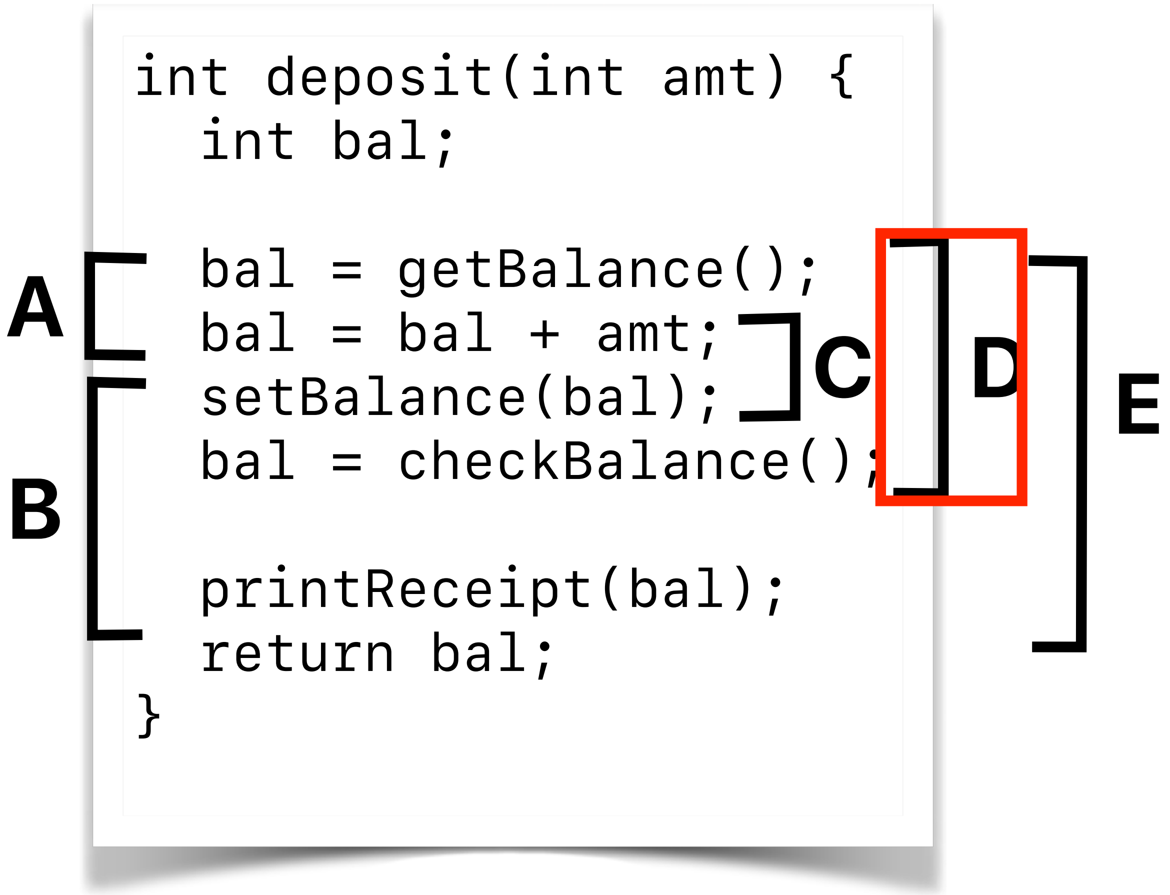
Thread B

```
int withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();

  printReceipt(bal);
  return bal;
}
```

Identifying Critical Sections

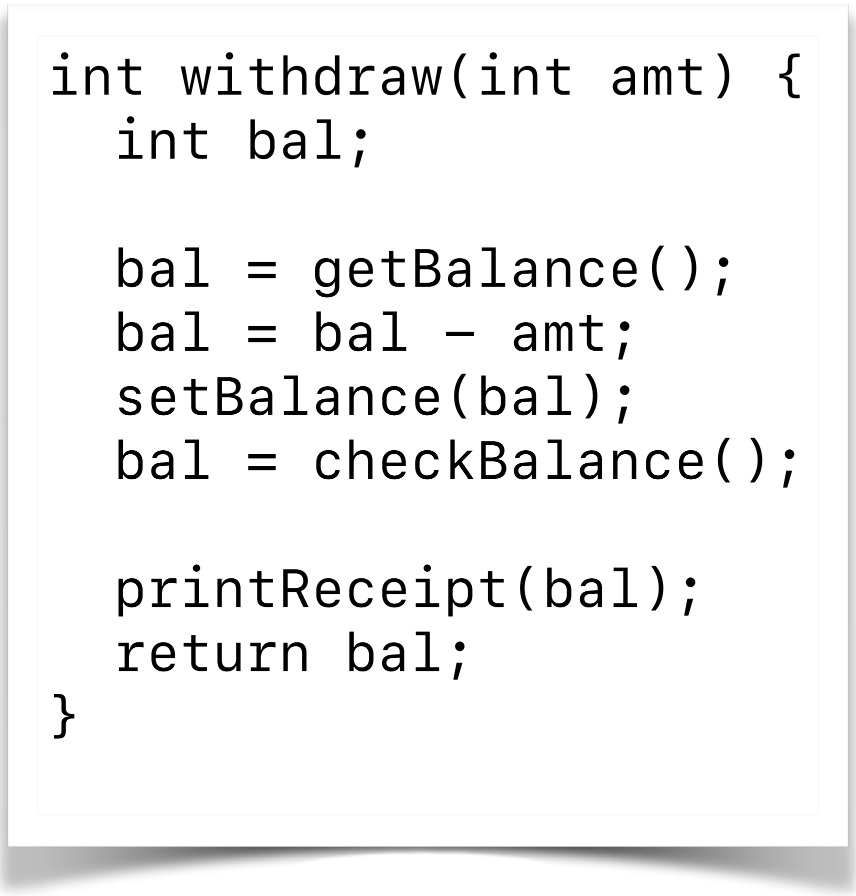
- Which is the smallest code region in Thread A that we can make as a critical section to guarantee the outcome as 30?

Thread A



```
int deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();

  printReceipt(bal);
  return bal;
}
```

Thread B



```
int withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();

  printReceipt(bal);
  return bal;
}
```

Critical sections

Thread A

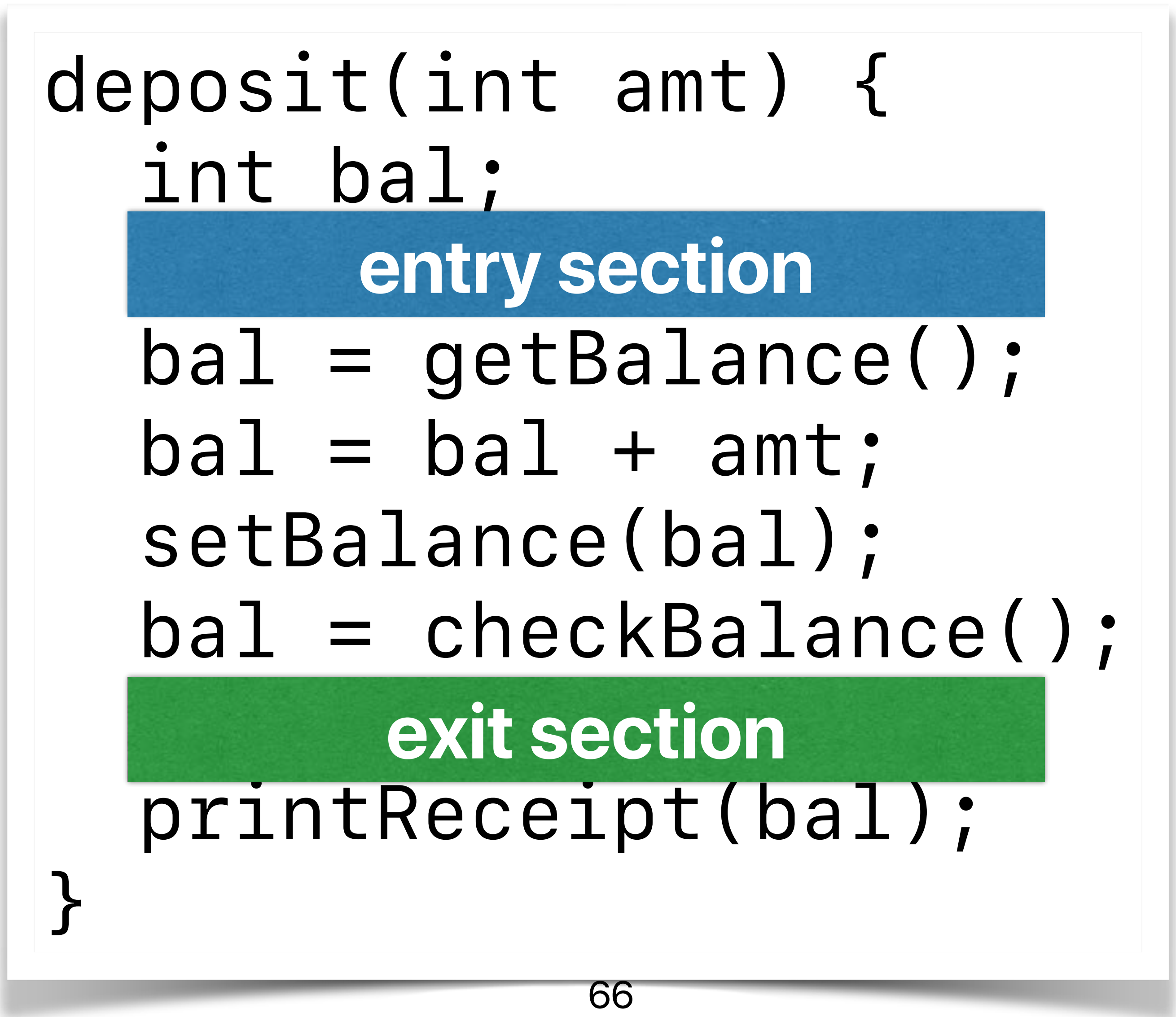
```
deposit(int amt) {
  int bal;

  bal = getBalance();
  bal = bal + amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Thread B

```
withdraw(int amt) {
  int bal;

  bal = getBalance();
  bal = bal - amt;
  setBalance(bal);
  bal = checkBalance();
  printReceipt(bal);
}
```

Critical section

- Entry section acts as barrier, only allowing a single thread in at a time
- Exit section should remove barrier for other threads' entry



```
deposit(int amt) {
   int bal;
      entry section
   bal = getBalance();
   bal = bal + amt;
   setBalance(bal);
   bal = checkBalance();
      exit section
   printReceipt(bal);
}
```

Announcement

- Reading quizzes due next Tuesday
- Project groups in 2
  - Will release the project by the end of the week
  - You may preview the idea/scope of the project https://github.com/hungweitseng/CS202-ResourceContainer
  - **Install an Ubuntu Linux 16.04 using VirtualBOX or VMWare as soon as you can!**

Computer
Science &
Engineering

202

つづく