COMP-SCI-431
Intro Operating Systems

Lecture 2 – Processes, Threads, and Resources

Adu Baffour, PhD

University of Missouri-Kansas City
Division of Computing, Analytics, and Mathematics
School of Science and Engineering
aabnxq@umkc.edu

Lecture Objectives

- To understand the concept of processes in operating systems.
- To explore the reasons behind the use of processes in computing.
- To examine the role and significance of the process control block in managing processes.
- To gain knowledge about the various operations that can be performed on processes.
- To explore the concept of resources in the context of operating system processes.
- To understand the fundamentals of threads and their relevance in executing processes.

Outline

Outline

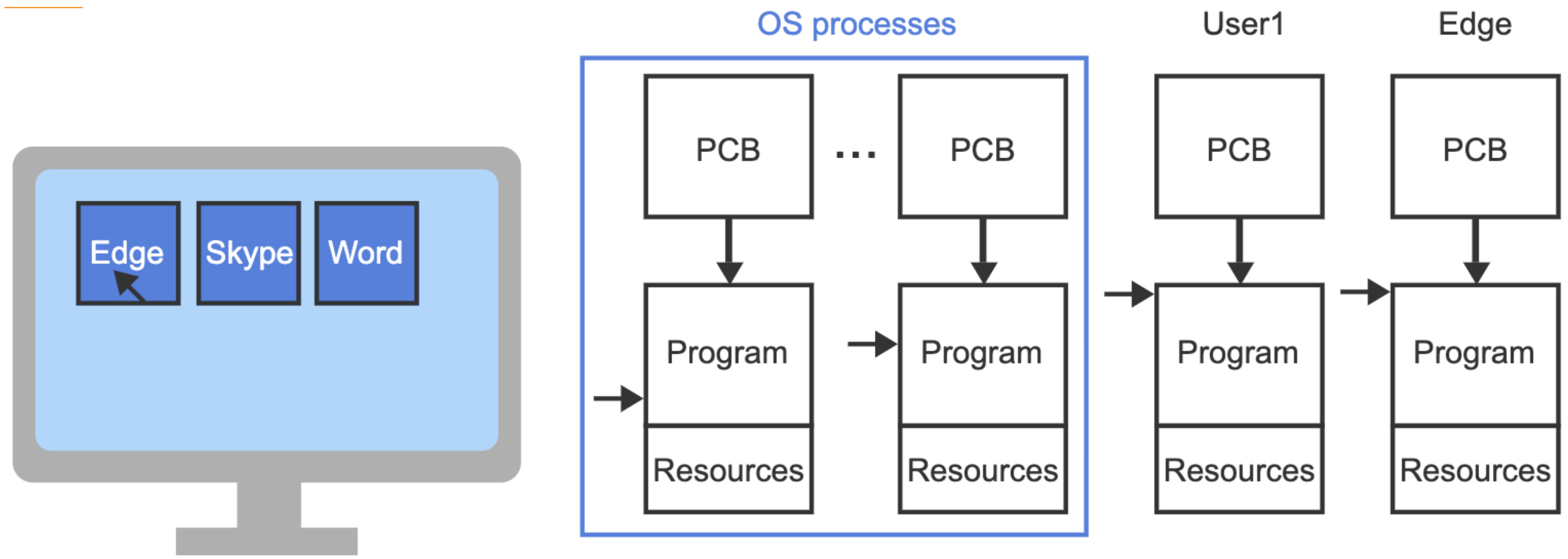
2.1 The process concept

The process concept

- A ***process*** is an instance of a program being executed by an OS.
- Ex: When a user opens a new application like a web browser or text editor, the OS creates a new process.
- The OS keeps track of each process using a ***process control block*** (***PCB***).
- PCB is a data structure that holds information for a process, including the current instruction address, the execution stack, the set of resources used by the process, and the program being executed.
- The PCB is the concrete representation of a process.

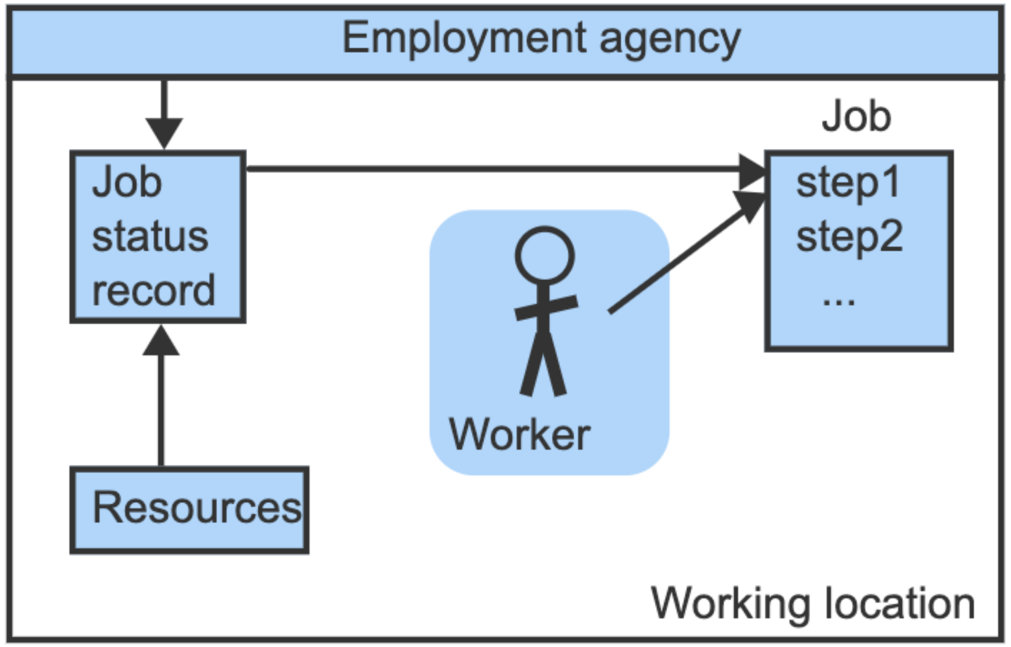
2.1 The process concept

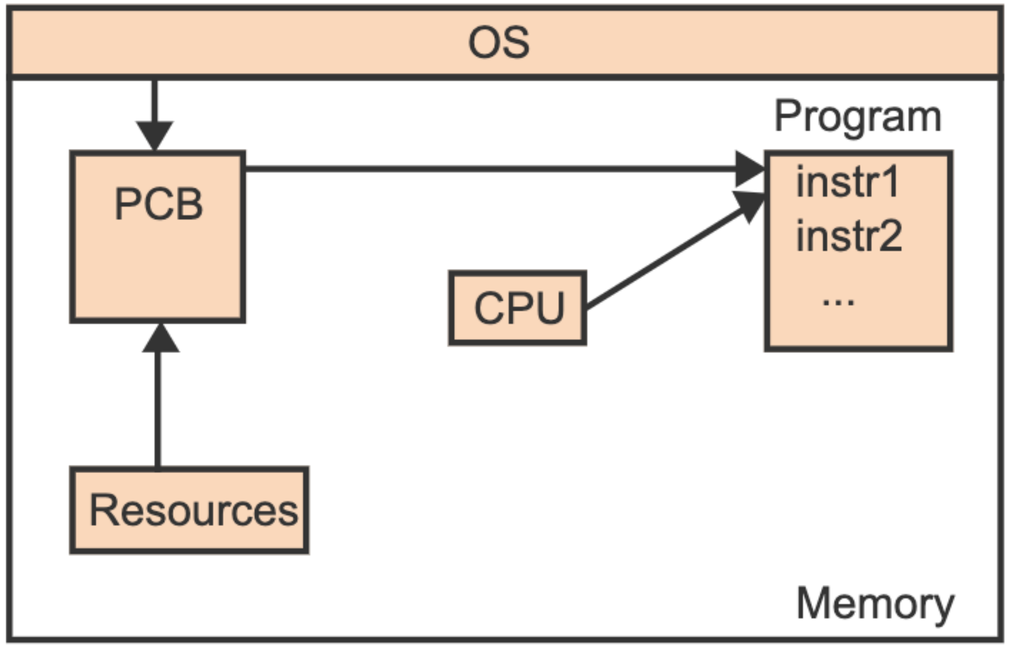
OS creates a new PCB for every new process



2.1 The process concept

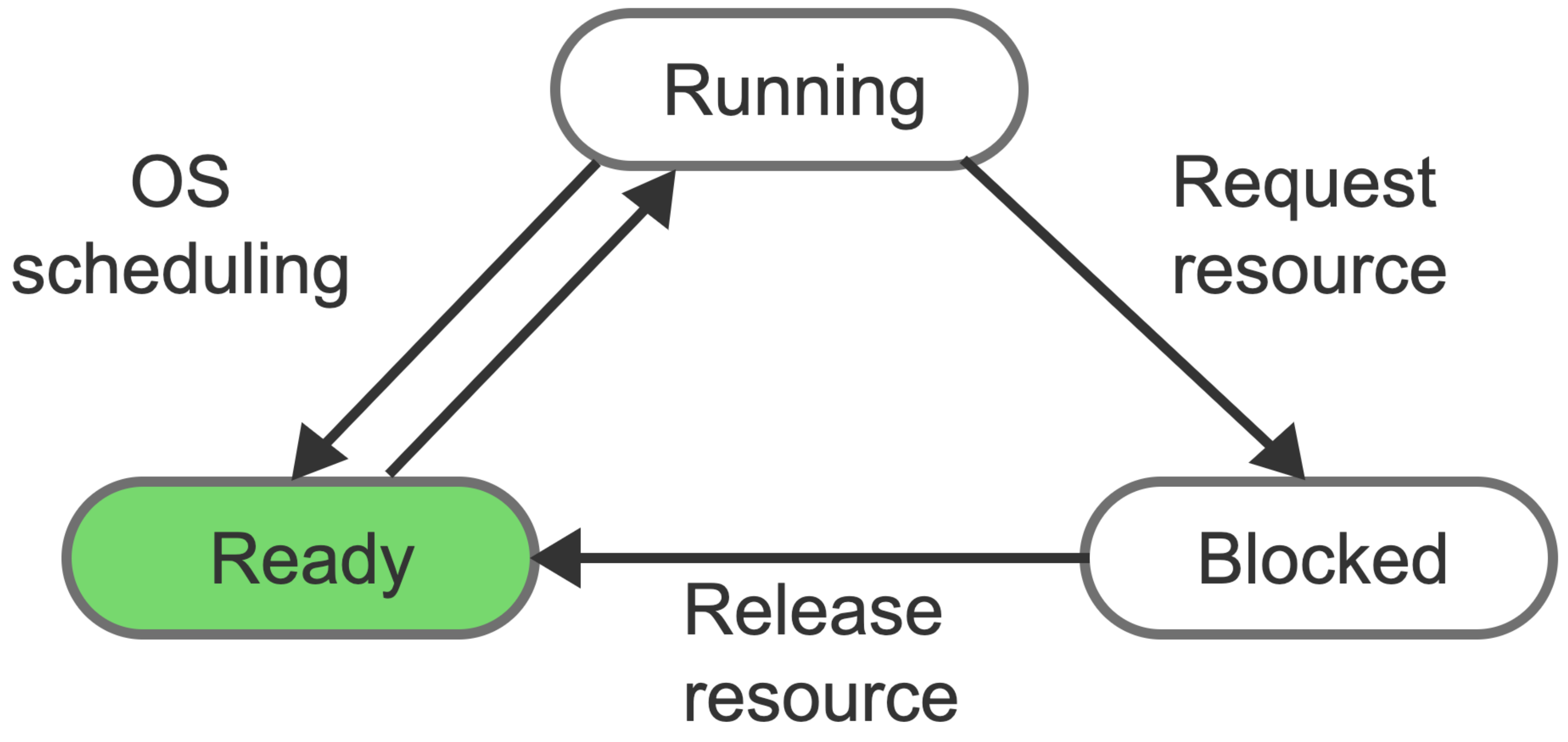
The process concept by analogy





2.1 The process concept

Process states and transitions



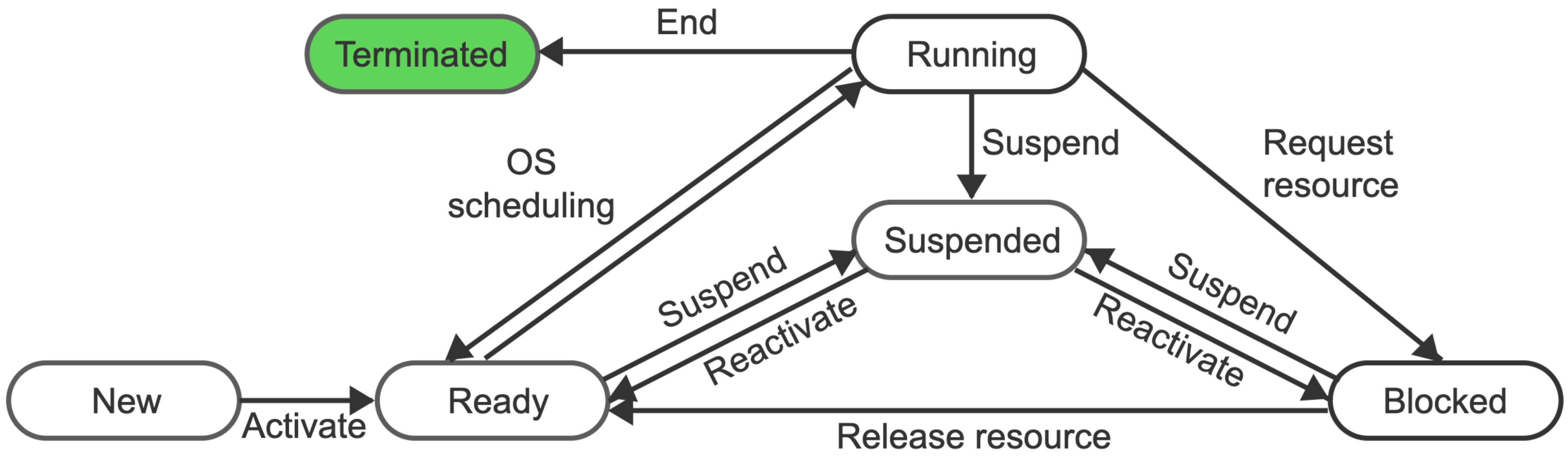
2.1 The process concept

Additional process states and transitions

- A newly created process is placed into the ***new state*** before the process is allowed to compete for the CPU.
- A process is placed into the ***terminated state*** when execution can no longer continue but before the PCB is deleted.
- A process may be placed into the ***suspended state*** even though the CPU and all resources are available.

2.1 The process concept

Additional state transitions



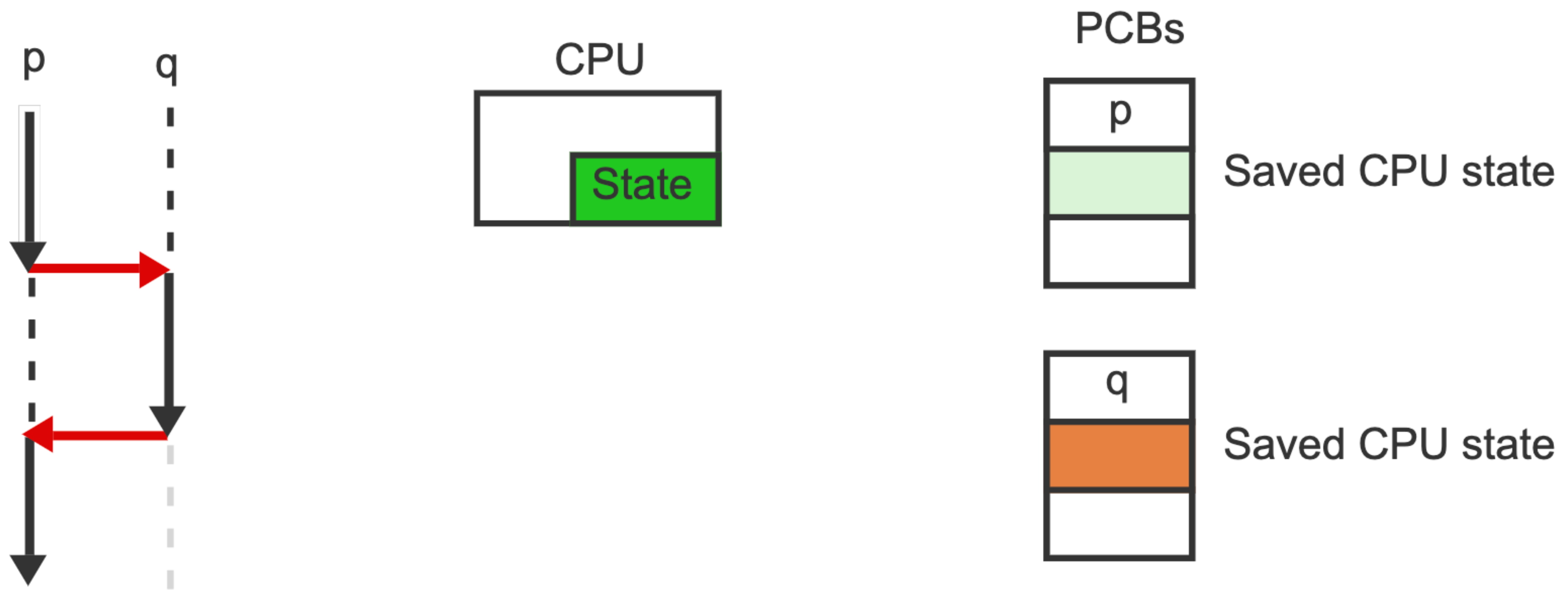
2.1 The process concept

The context switch

- A ***context switch*** is the transfer of control from one process to another.
- OS must save all information about the stopped process. This information is restored when the process again gets a chance to run.
- The ***CPU state*** consists of all intermediate values in any CPU registers and hardware flags during the interruption.
- One of the registers is the program counter, which determines the next instruction to execute after restoring the process.

2.1 The process concept

Context switches between processes p and q



Outline

2.2 Why processes

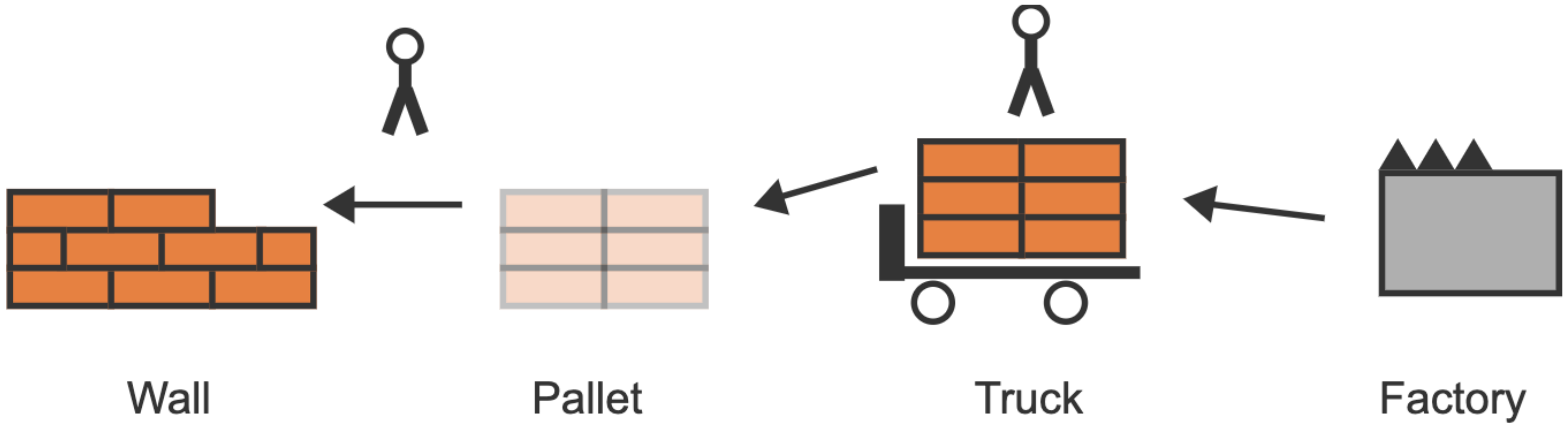
A modular structuring concept

- An application can generally be divided into multiple tasks, each implemented as a process.
- Advantages:

1. The interfaces between the processes are simple and easy to understand.
2. Each process can be designed and studied in isolation.
3. The implementation reduces idle time by overlapping the execution of multiple processes.
4. Different processes can utilize separate CPUs, if available, thus speeding up the execution.

2.2 Why processes

Two cooperating processes yield a better design than one: A wall-building analogy



Outline

2.3 The process control block

Contents of the PCB

- The PCB is the instantiation of a process.
- Upon creation, the OS assigns every process a unique identifier.
- This identifier, p, could be a pointer to the PCB structure or an index into an array of PCBs.
- A PCB's specific implementation and contents vary between different OSs, but the following is a generic structure representative of most modern OSs.

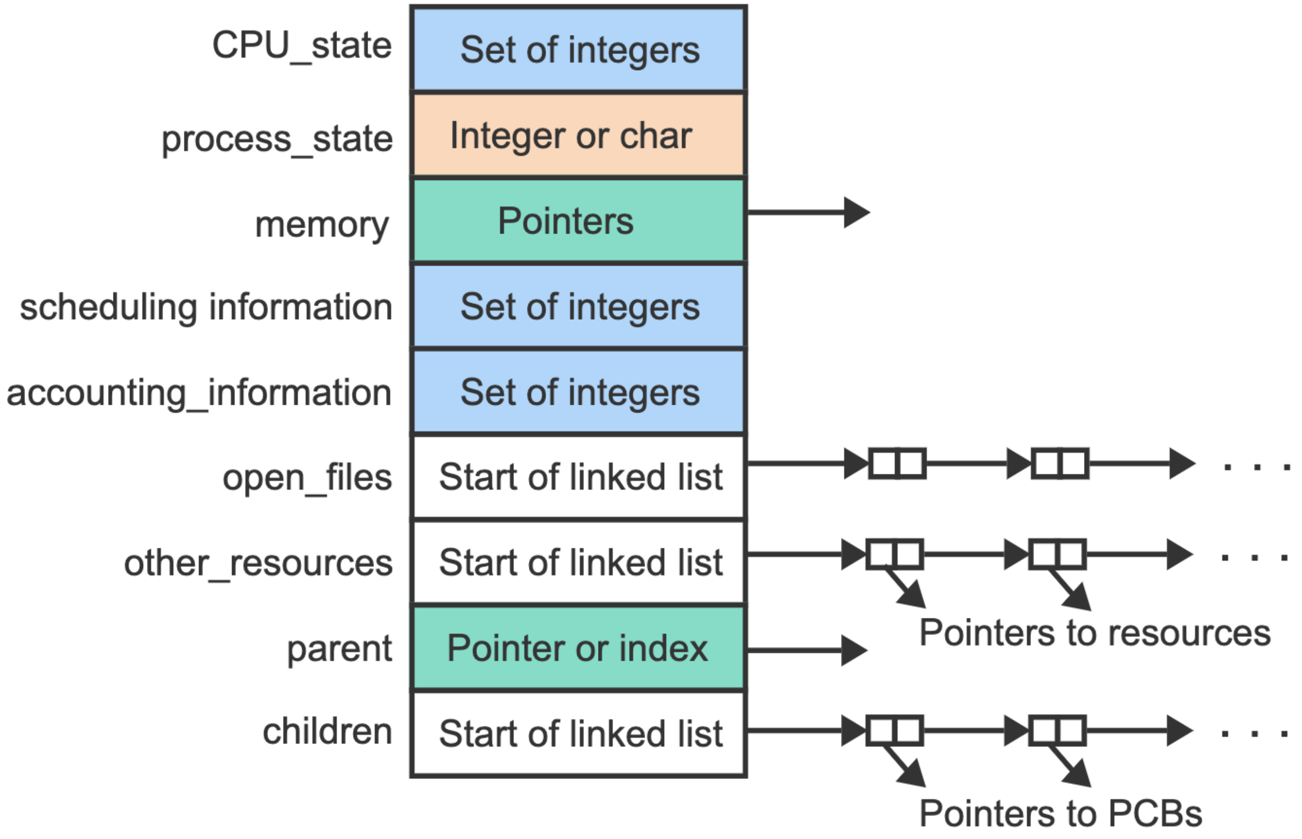
2.3 The process control block

PCB field	Explanation
CPU_state	When p is stopped, the current state of the CPU, consisting of various hardware registers and flags, is saved in this field. The save information is copied back to the CPU when p resumes execution.
process_state	Stores p's current state. Ex: Running, ready, or blocked.
memory	Describes the area of memory assigned to p. In the simplest case the field would point to a contiguous area of main memory. In systems using virtual memory (discussed elsewhere) the field could point to a hierarchy of memory pages or segments.
scheduling_information	Contains information used by the scheduler to decide when p should run. The information typically records p's CPU time, the real time in the system, the priority, and any possible deadlines.
accounting_information	Keeps track of information necessary for accounting and billing purposes. Ex: The amount of CPU time or memory used.
open_files	Keeps track of the files currently open by p.
other_resources	Keeps track of any resources, such as printers, that p has requested and successfully acquired.
parent	Every process is created by some other running process. The ***parent process*** of a process p is the process that created p. The parent field records the identity of p's parent.
children	A ***child process*** c of process p is a process created by p. Process p is c's parent. The identity of every child process c of p is recorded in the children field.

2.3 The process control block

The PCB data structure

- The entries of a PCB are composed of different data types, including integers, characters, or pointers.
- Consequently, each PCB is implemented as a heterogeneous data structure. Ex: "struct" in the C language.



2.3 The process control block

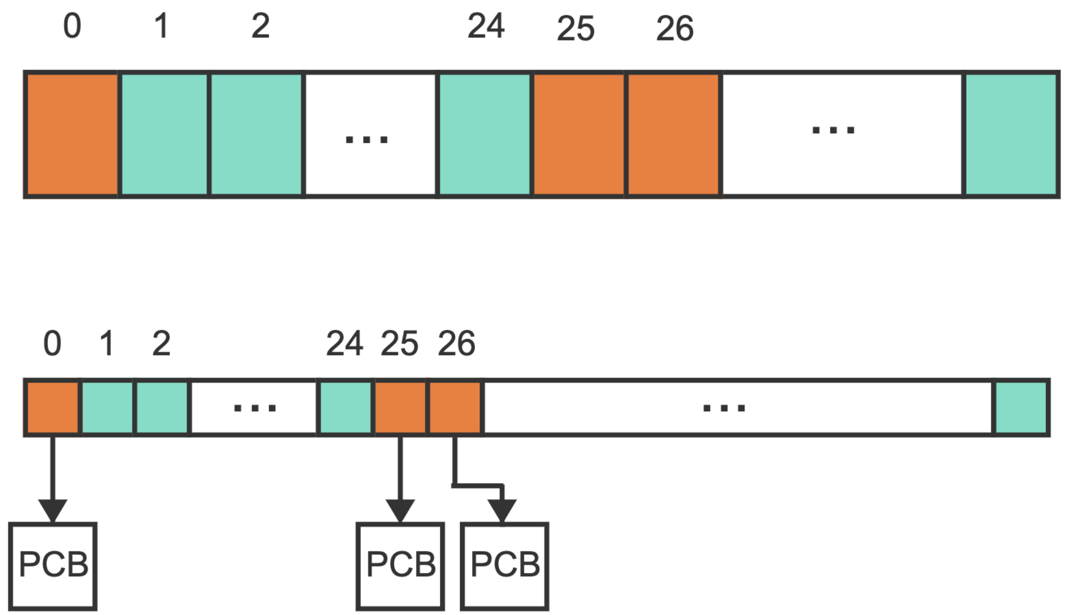
Organizing PCBs

The OS must allocate and deallocate PCBs efficiently as processes are created and destroyed. Two ways exist to organize all PCBs:

- An array of structures.
  - The PCBs are marked as free or allocated, eliminating the need for dynamic memory management.
  - The main drawback is wasting memory space to maintain sufficient PCB slots.
- An array of pointers to dynamically allocated PCBs.
  - The pointer array wastes little space and can be much larger than the array of structures.
  - The drawback is the overhead of dynamic memory management to allocate each new PCB and to free the memory when the process terminates.

2.3 The process control block

Allocating and deallocating PCBs



Linked lists require dynamic memory management, which is costly.

The Linux OS has pioneered an approach that eliminates this overhead.

2.3 The process control block

Managing PCBs

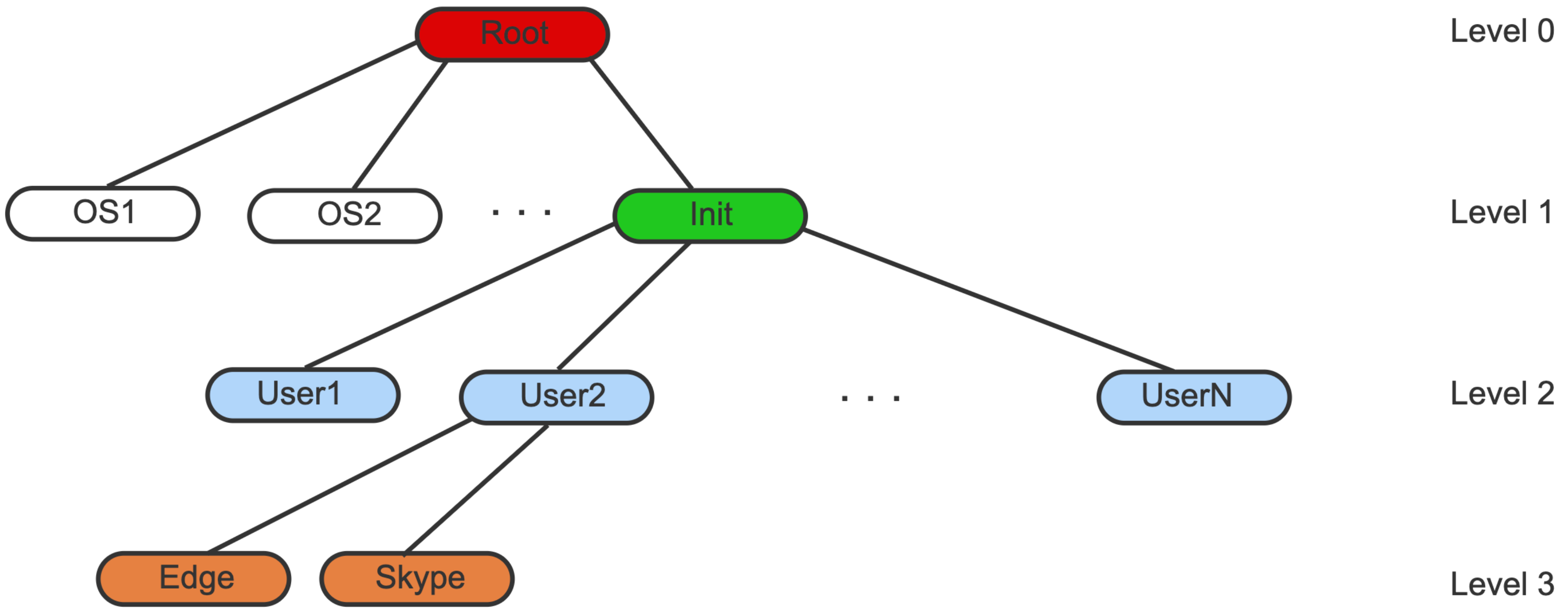
- The OS maintains all PCBs organized on various lists.
- A ***waiting list*** is associated with every resource and contains all processes blocked on that resource because the resource is not available.
- Another important list is the ***ready list*** (***RL***): A list containing all processes that are in the ready state and thus can run on the CPU.
- The RL maintains all processes sorted by importance, expressed by an integer value called the priority.
- The RL can be a simple linked list where the priority of a process is the current position in the list.

Outline

2.4 Operations on processes

The process creation hierarchy

- A ***process creation hierarchy*** is a graphical representation of the dynamically changing parent-child relationships among all processes.



2.4 Operations on processes

Process creation

The ***create process function*** allocates a new PCB, fills the PCB entries with initial values, and links the PCB to other data structures in the system.

The create process function

```
p = create(state0, mem0, sched0, acc0) {
      p = allocate new PCB
      p.cpu_state = state0
      p.memory = mem0
      p.scheduling_information = sched0
      p.accounting_information = acc0
      p.process_state = ready
      p.parent = self
      insert p into self.children
      p.children = NULL
      p.open_files = NULL
      p.other_resources = NULL
      insert p into RL
      scheduler()
      return p
   }
```

2.4 Operations on processes

Process destruction

- The ***destroy process function*** destroys a process by freeing the PCB data structure and removing any references to the PCB from the system.
- Depending on the OS, the destroy function may also destroy all the process's descendants to prevent having "orphan" processes in the system.
- The destroy() function performs the following steps:
    1. Remove p from either the RL (when p is ready) or from the waiting list of a resource (when p is blocked).
    2. Remove p from the list of children of the calling process.
    3. Release all memory and other resources, close all open files, and deallocate the PCB.
    4. Call the scheduler to select the next process to run.

2.4 Operations on processes

Process destruction

The destroy process function

```
destroy(p) {
   for all c in p.children destroy(c)
   remove p from RL or waiting list
   remove p from parent's list of children
   release p.memory
   release p.other_resources
   close all p.open_files
   deallocate PCB
}
scheduler()
```

Outline

2.5 Resources

Representing resources

- ***A resource control block (RCB)*** is a data structure that represents a resource.
- A generic RCB of resource r

RCB field	Explanation
resource_description	Contains the description of r's properties and capabilities.
state	Shows the current availability of r. If r is a single-unit resource then this field indicates whether r is currently free or allocated to some process. If r has multiple identical units, such as a pool of identical buffers, then this field keeps track of how many units are currently free and how many are allocated.
waiting_list	When a process requests r and r is currently unavailable then the process's PCB is removed from the ready list and added to r's waiting list. The process is moved back to the ready list when r becomes available and is allocated to the process.

2.5 Resources

Requesting a resource

- A resource is ***allocated*** to a process if the process has access to and can utilize the resource.
- A resource is ***free*** if the resource may be allocated to a requesting process.
- The ***request resource function*** allocates a resource r to a process p or blocks p if r is currently allocated to another process.

```
request(r) {
   if (r.state == free) {
      r.state = allocated
      Insert r into self.other_resources
   }
   else {
      self.state = blocked
      Move self from RL to r.waiting_list
      scheduler()
   }
}
```

2.5 Resources

Releasing a resource

- The ***release resource function*** allocates the resource r to the next process on the r's waiting list.
- If the waiting list is empty, r is marked as free.

```
release(r) {
   Remove r from self.other_resources
   if (r.waiting_list == empty)
      r.state = free
   else
      Remove process q from the head of r.waiting_list
      Insert r into q.other_resources
      q.process_state = ready
      Move q from r.waiting_list to RL
      scheduler()
}
```

2.5 Resources

The scheduler

- The ***scheduler function*** determines which process should run next and starts the process.
- Assuming the RL is implemented as a priority list, the scheduler() function performs the following tasks:

1. Find the highest priority process q on the RL.
2. Perform a context switch from p to q if either of the following conditions is met:
    - The priority of the running process p is less than that of q.
    - The state of the running process p is blocked.

2.5 Resources

The ***scheduler function***

```
scheduler() {
   Find highest priority process q on RL
   if (p.priority < q.priority) or (p.process_state == blocked)
      perform context switch from p to q
}
```

Outline

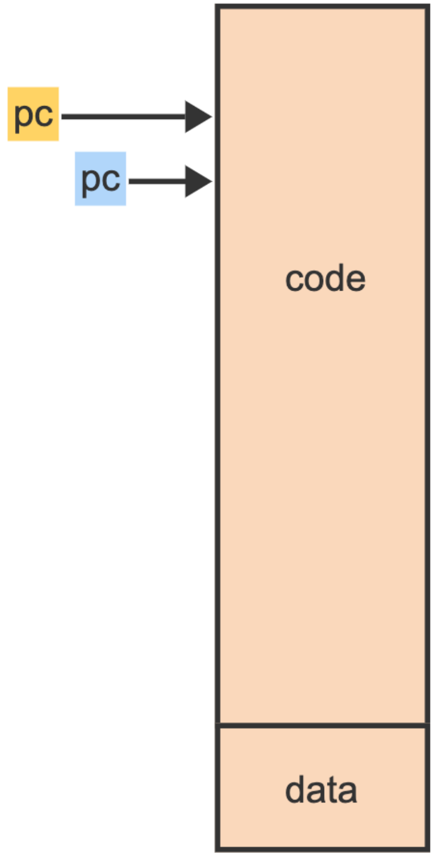
2.6 Threads

The thread concept

- A ***thread*** is an instance of executing a portion of a program within a process without incurring the overhead of creating and managing separate PCBs.

For example:

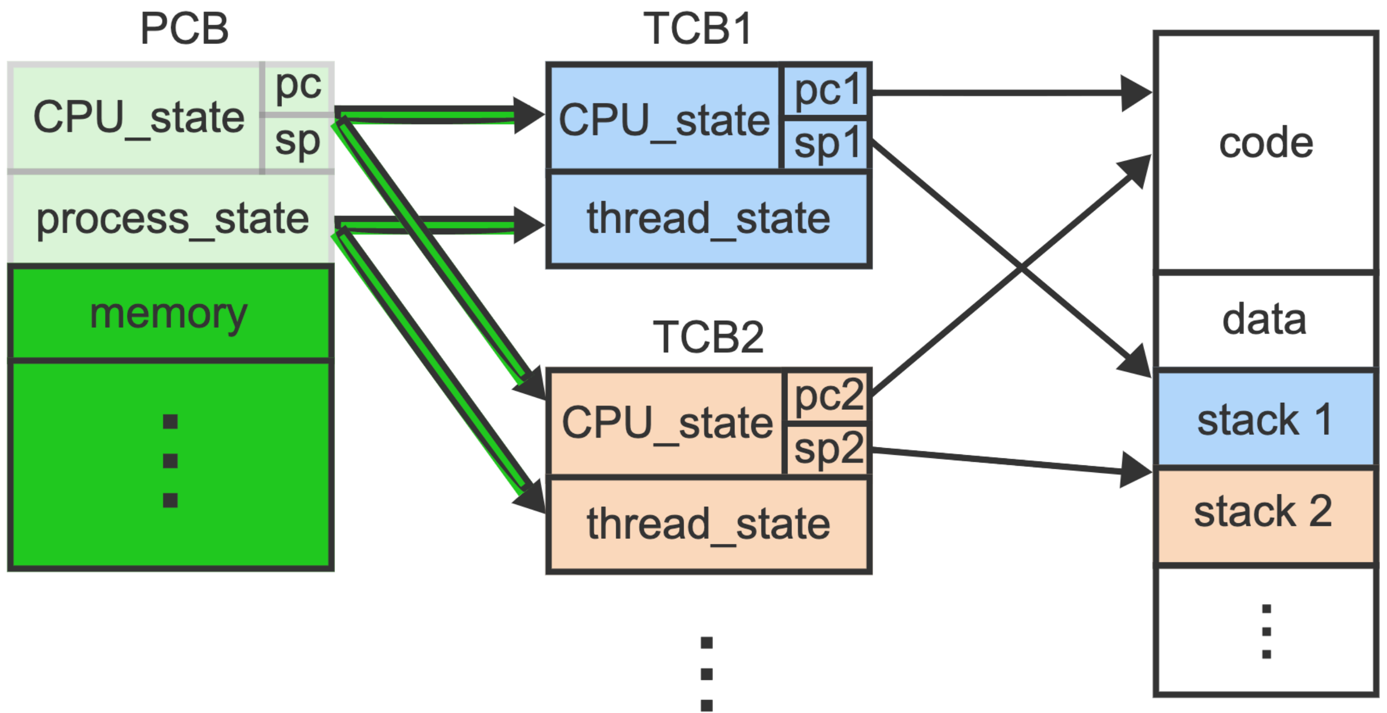
- The Chrome browser can run the Dinosaur game while simultaneously waiting for data from the Internet.
- A word processor can run a spell checker at the same time as the user is typing.
- An Internet server, such as an email or web page server, receives many requests from different users simultaneously.



2.6 Threads

The thread control block

- A ***thread control block (TCB)*** is a data structure that holds a separate copy of the dynamically changing information necessary for a thread to execute independently.
- The replication of only the bare minimum of information in each TCB while sharing the same code, global data, resources, and open files is what makes threads much more efficient to manage than processes.



2.6 Threads

User-level vs kernel-level threads

User-level

- Threads can be implemented completely within a user application.
- A thread library provides functions to create, destroy, schedule, and coordinate threads.
- All thread management occurs within the user process. The kernel is not aware of the multiple threads.

kernel-level

- The alternative is to implement threads within the OS kernel.
- The TCBs are not directly accessible to the application, which uses kernel calls to create, destroy, and otherwise manipulate the threads.

2.6 Threads

Combining user-level and kernel-level threads

Advantages of user-level threads (ULTs) over kernel-level threads (KLTs):

- ULTs are much faster to manage; thus, many more can be created than KLTs.
- Applications using ULTs are portable between different OSs without modifications.

Main disadvantages of ULTs over KLTs:

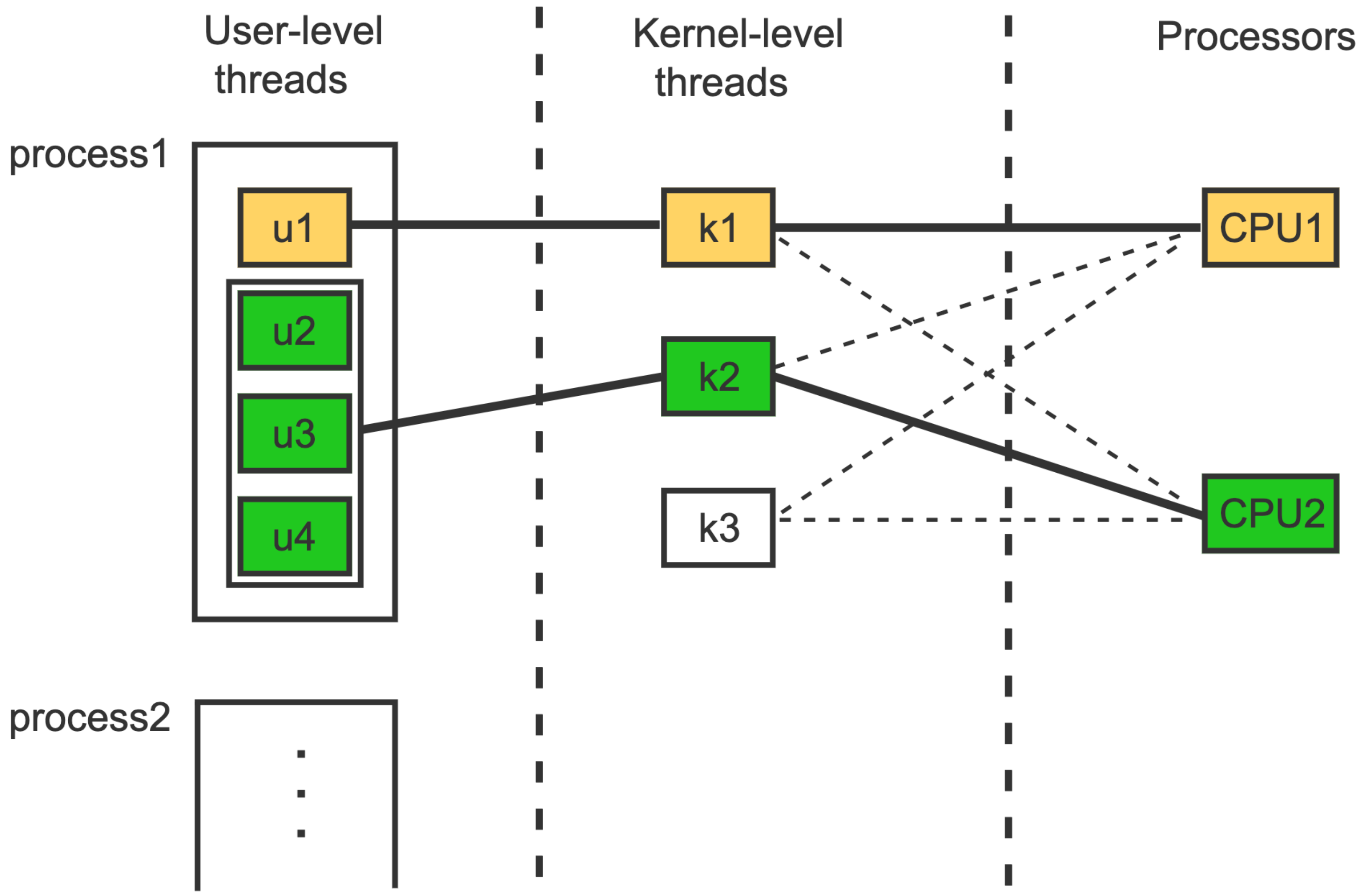
- ULTs are not visible to the kernel. When one ULT blocks, the entire process blocks, which decreases concurrency.
- ULTs cannot take advantage of multiple CPUs because the kernel perceives the process as a single thread of execution.

Many modern OSs take a combined approach. Each application can implement any number of ULTs, which are then mapped on the KLTs based on the ULTs' needs and the available resources.

2.6 Threads

Mapping user-level threads on kernel-level threads

1. The kernel maintains a small number of KLTs. The kernel also schedules the KLTs on the available CPUs.
2. Each process can create any number of ULTs using a thread library.
3. The programmer decides which ULTs or groups of ULTs are mapped on separate KLTs.
4. When u1 blocks, k1 blocks as well. But as long as at least one of u2, u3, or u4 runs, the process continues in k2.
5. When u1 runs again, and one of u2, u3, or u4 continues running, the process may continue in parallel on both CPUs.



End of Lecture

Thank you

Any questions?