Section 5
Concurrent Computing

1. Concurrent systems
2. Process management
3. Inter-process communication
4. Threads

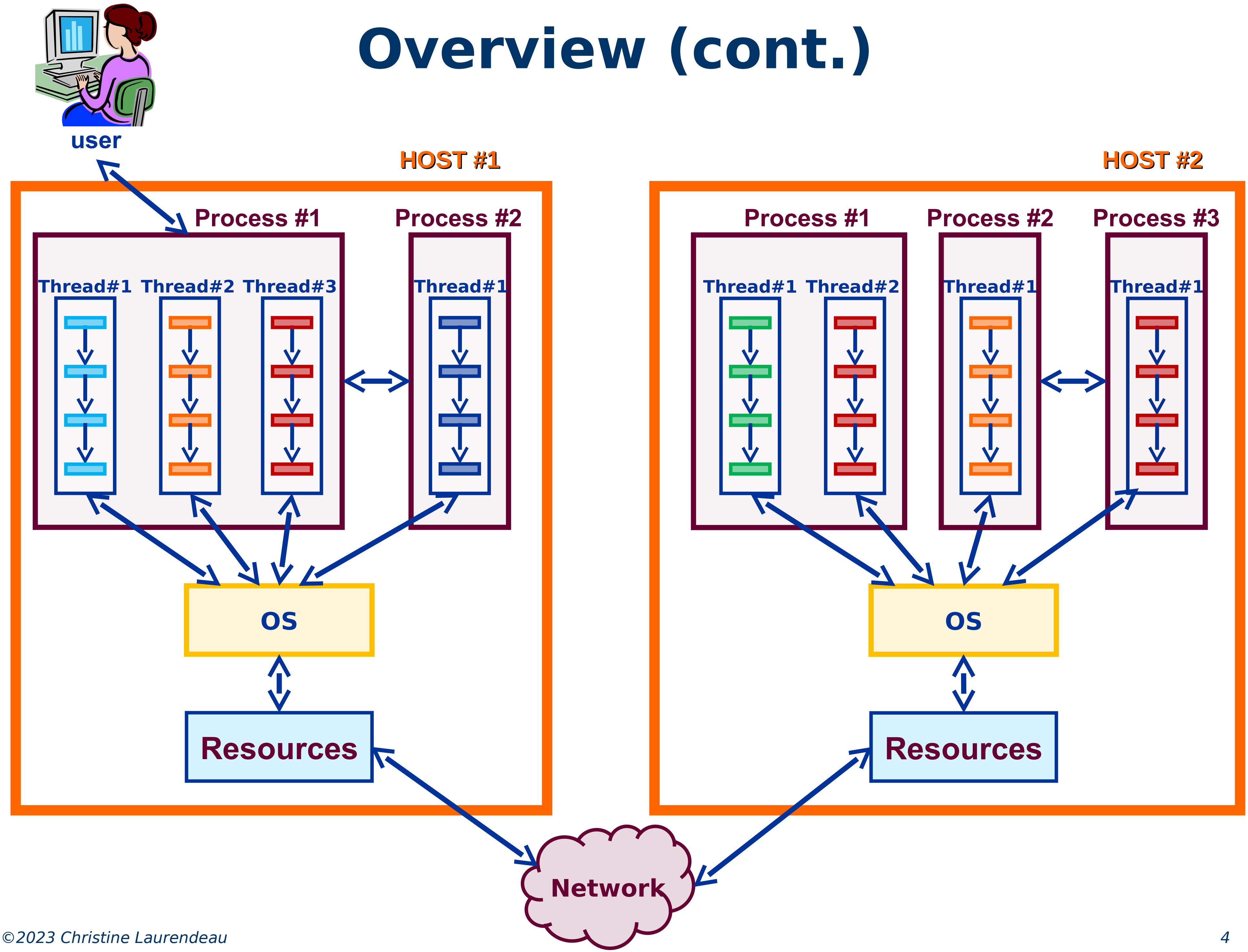
Section 5.1
Concurrent Systems

1. Overview
2. Types of concurrent systems
3. Issues in concurrency

5.1.1 Overview

- What is *concurrency*?
  - in general, it means doing more than one thing at the same time
- What is *concurrent computing*?
  - it's when a program has more than one control flow
  - in software engineering, a *system* is a large program or application
  - a system can be:
    - distributed, and/or
    - multi-process, and/or
    - multi-threaded

Overview (cont.)



user

HOST #1

Process #1

Thread#1 Thread#2 Thread#3

Process #2

Thread#1

OS

Resources

HOST #2

Process #1

Thread#1 Thread#2

Process #2

Thread#1

Process #3

Thread#1

OS

Resources

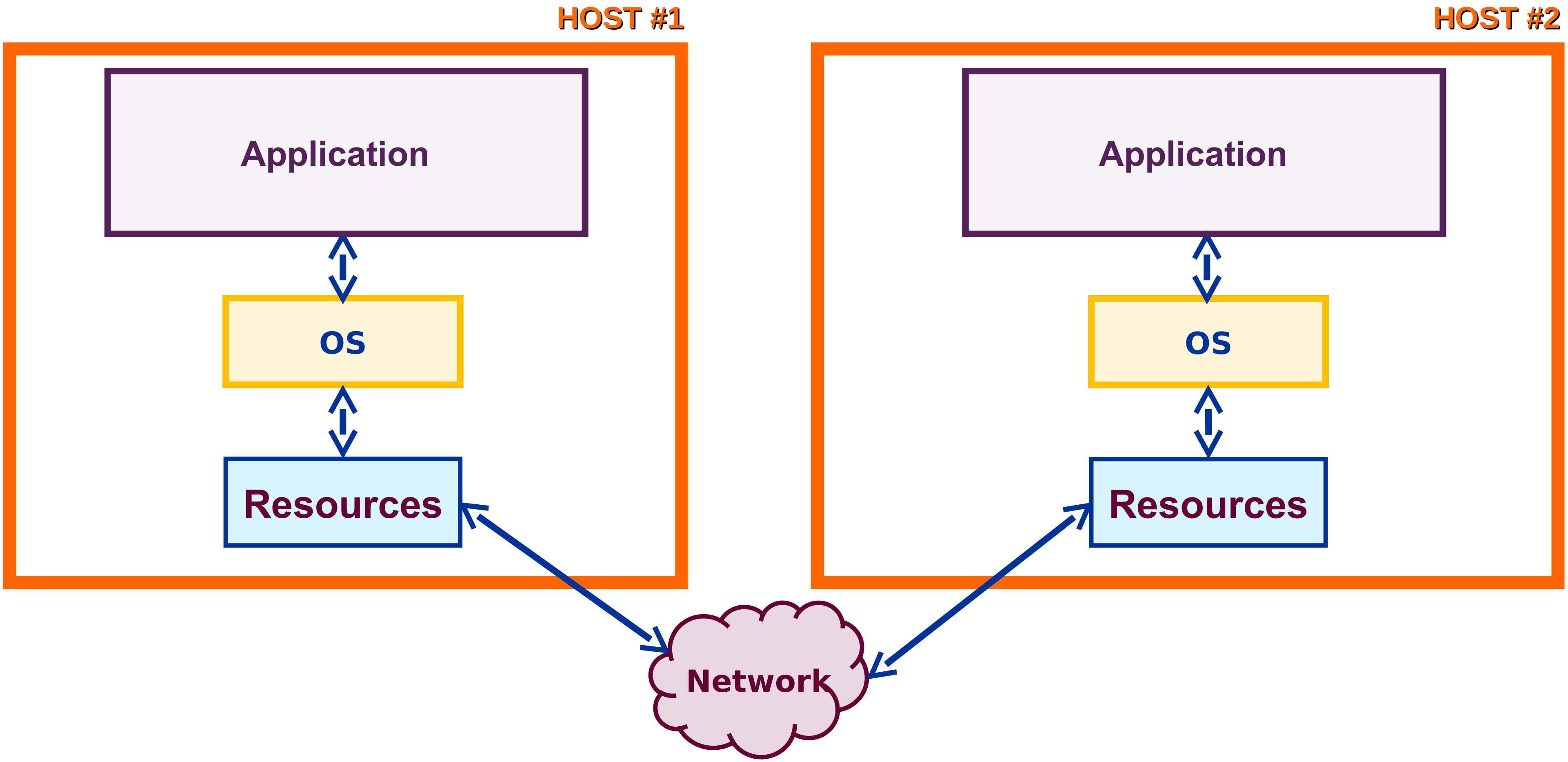
Network

5.1.2 Types of Concurrent Systems

- Distributed systems
- Multi-process systems
- Multi-threaded systems

Distributed Systems

- What is a *distributed system*?
  - it's a software system that executes over multiple physical hosts
    - typically in different locations, cities, or countries



©2023 Christine Laurendeau

Distributed Systems (cont.)

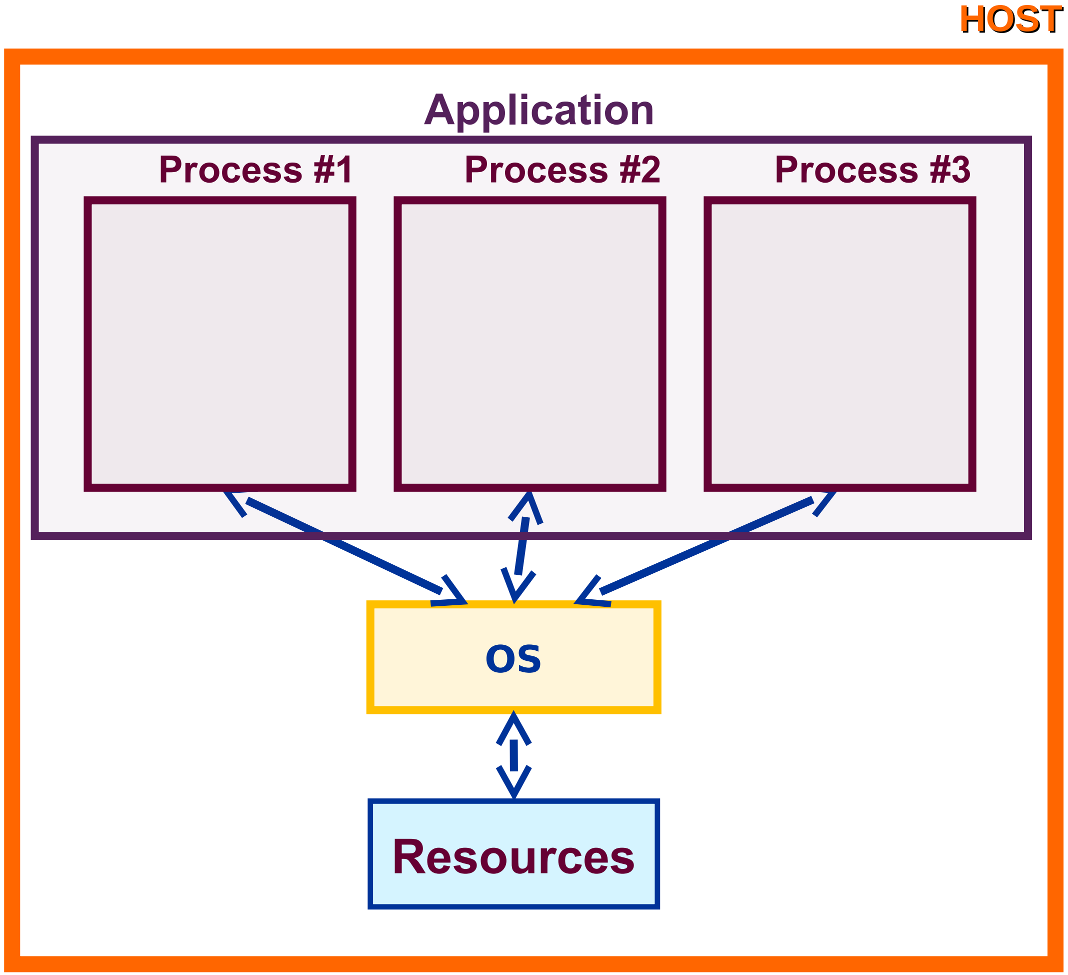
- Characteristics of a distributed system
  - each host has different resources
    - different file systems
    - different CPUs, processing capabilities
    - … everything …
  - hosts must be networked together in order to communicate
    - **intranet**: network internal to an organization
    - **internet**: network external to all organizations (public network)

Distributed Systems (cont.)

- Why a distributed system?
  - users can be in different physical locations
  - server hosts can be in different physical locations
  - a single host may have insufficient processing power
  - example:
    - server computers store the data
    - client computers access the centralized data

Multi-Process Systems

- What is a *multi-process system*?
  - it's a system made up of multiple processes (running executables)



Multi-Process Systems (cont.)

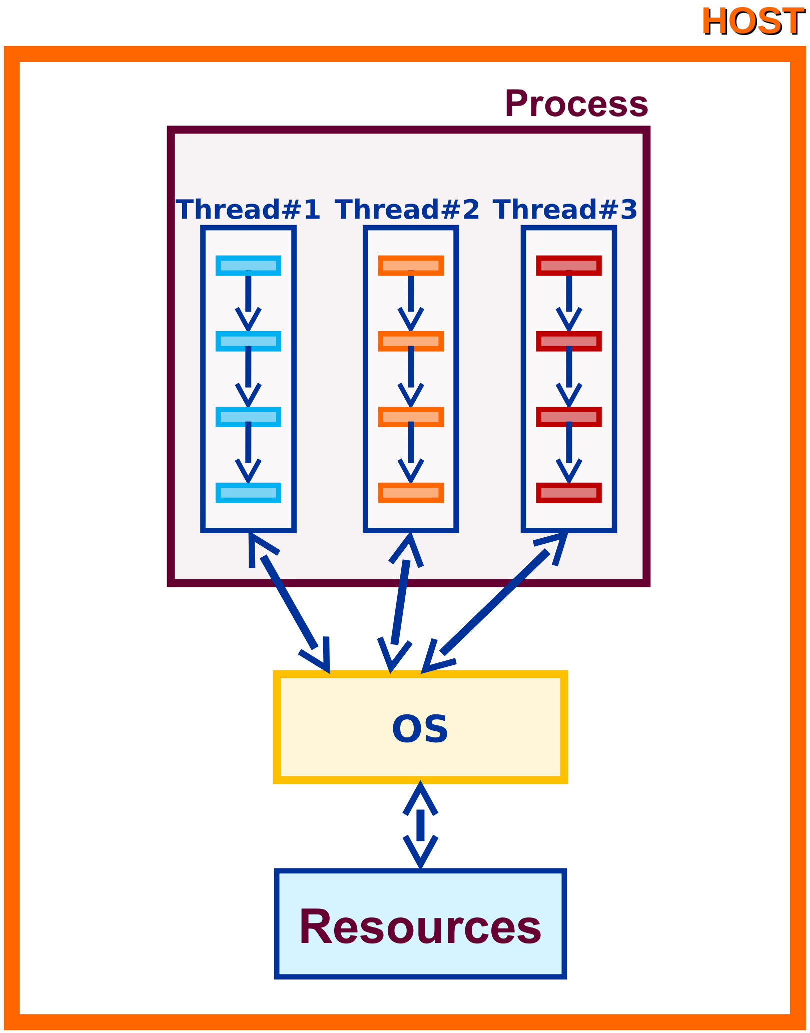
- Characteristics of a multi-process system
  - the multiple processes can be:
    - *different* executables
    - multiple instances of the *same* executable
  - each process has:
    - its own independent control flow(s)
    - its own virtual memory
  - processes typically need to communicate with each other
    - they must use inter-process communication (IPC) techniques

Multi-Process Systems (cont.)

- Why a multi-process system?
  - the system may have very different tasks to perform
  - the tasks may be completely independent from each other
  - the tasks may use different resources than each other
  - example:
    - one client process may communicate with the user
    - one server process may handle user requests
    - one process may regulate access to the database

Multi-Threaded Systems

- What is a *multi-threaded system*?
  - it’s where a process has multiple control flows, called *threads*



Multi-Threaded Systems (cont.)

- Characteristics of a multi-threaded system
  - all the threads in one process share the same:
    - virtual memory
    - address space
    - resources
  - different threads may need to synchronize with each other
    - to communicate or exchange information
  - this creates possible issues with:
    - race conditions
    - deadlocks

Multi-Threaded Systems (cont.)

- Why a multi-threaded system?
  - a process may have different tasks to perform
  - the tasks may be somewhat dependent on each other
  - example:
    - one thread blocks, waits for user input, dispatches user requests
    - other threads deal with the user requests

5.1.3 Issues in Concurrency

- Shared resources
  - multiple processes or threads may need the same resource
    - example: data in a variable or in a file
  - operations that make changes to resources must be *atomic*
    - atomic operations cannot be *preempted* (interrupted) by the CPU
  - examples of shared resources
    - processes accessing the same file
      - file should be locked, to prevent other processes from accessing it
    - threads accessing a shared variable
      - variable should be locked using a *semaphore* or *mutex*

Issues in Concurrency (cont.)

- Deadlock
  - this happens when multiple threads are blocked, all waiting for a condition that will never occur
  - deadlocks are usually due to:
    - programming error(s)
    - the improper handling of semaphores or mutexes

- Race condition
  - when the correctness of a program depends on one thread reaching a point in the control flow before another thread
    - this order **cannot** be guaranteed