Introduction to Concurrency

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 5828 — Lecture 3 — 01/22/2008

Credit where Credit is Due

- Some text and images for this lecture come from the lecture materials provided by the publisher of the Magee/Kramer textbook. As such, some material is copyright © 2006 John Wiley & Sons, Ltd.

Lecture Goals

- Review material in Chapter 1 of the Magee/Kramer book
  - What do we mean by concurrent programs?
  - What do we mean by model-based software engineering?
  - Examine fundamental approach used in this book:
    - Concepts, Modeling, Practice

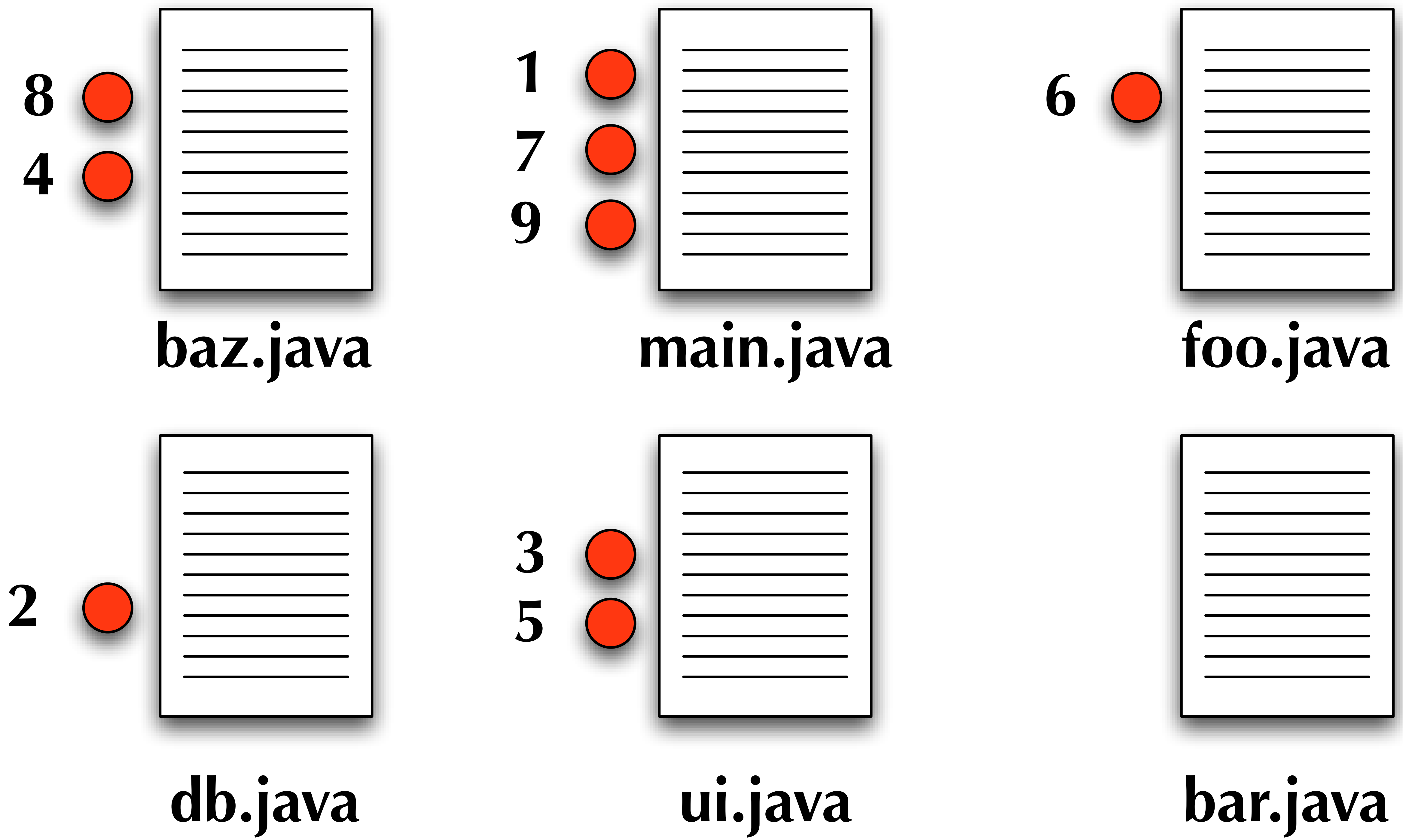
More on the Authors: “The Two Jeffs”

- Jeff Kramer
  - Dean of the Faculty of Engineering and Professor of Distributed Computing at the Department of Computing at Imperial College London
  - ACM Fellow
  - Editor of IEEE’s Transactions on Software Engineering
  - Winner of numerous software engineering awards including best paper and outstanding research awards
- Jeff Magee
  - Professor at the Department of Computing at Imperial College London
  - Long time member of the SE community with more than 70 journal and conference publications!
- This book’s material is based on their SE research into modeling concurrency over the past 20 years

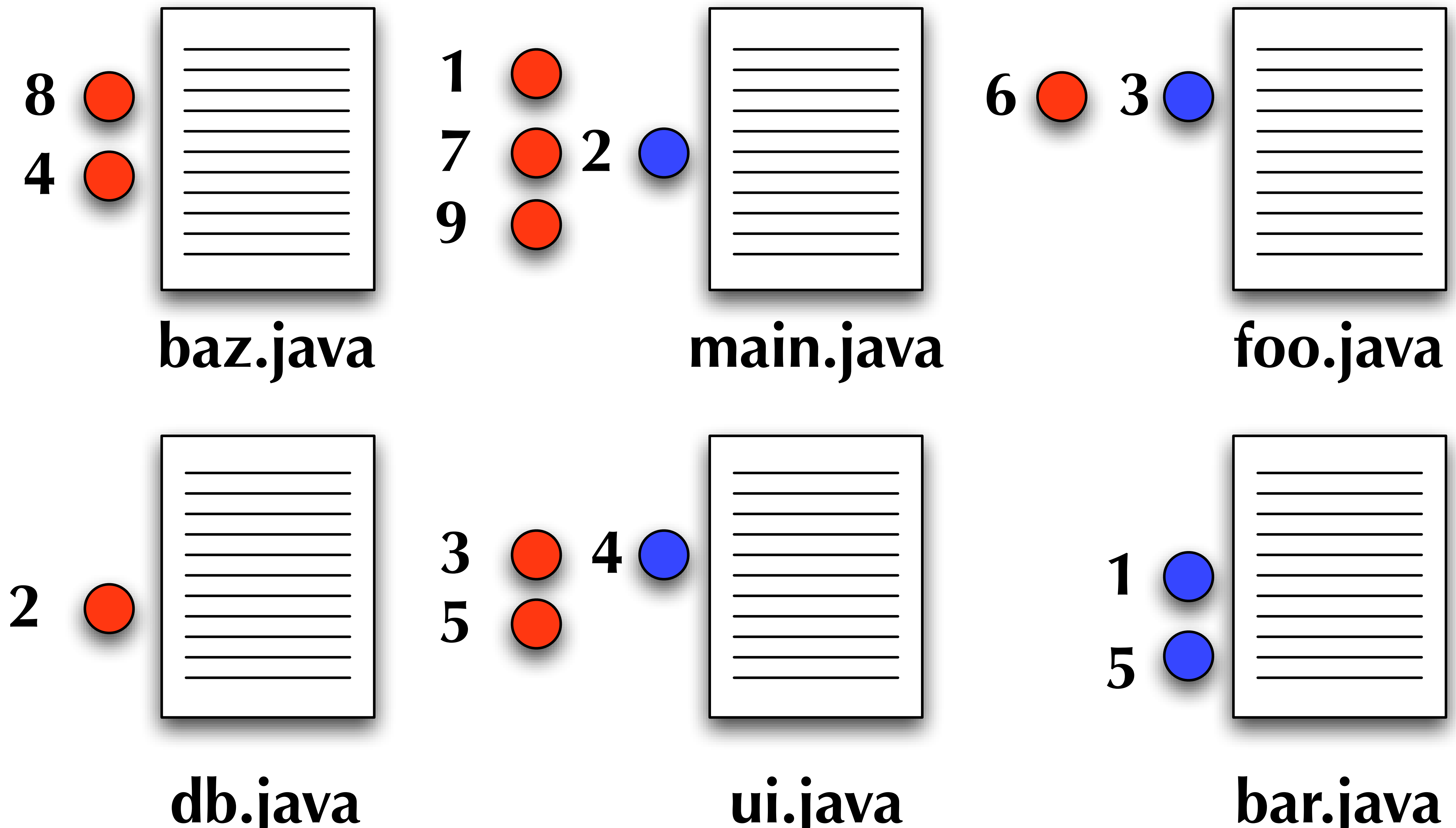
What is a Concurrent Program?

- When we execute a program, we create a **process**
  - A **sequential program** has a **single thread** of control
  - A **concurrent program** has **multiple threads** of control
- A single computer can have multiple processes running at once
  - If that machine, has a single processor, then the illusion of multiple processes running at once is just that: **an illusion**
    - That illusion is maintained by the operating system that coordinates access to the single processor among the various processes
  - If a machine has more than a single processor, then **true parallelism** can occur: you can have N processes running simultaneously on a machine with N processors

Another View: Sequential Program



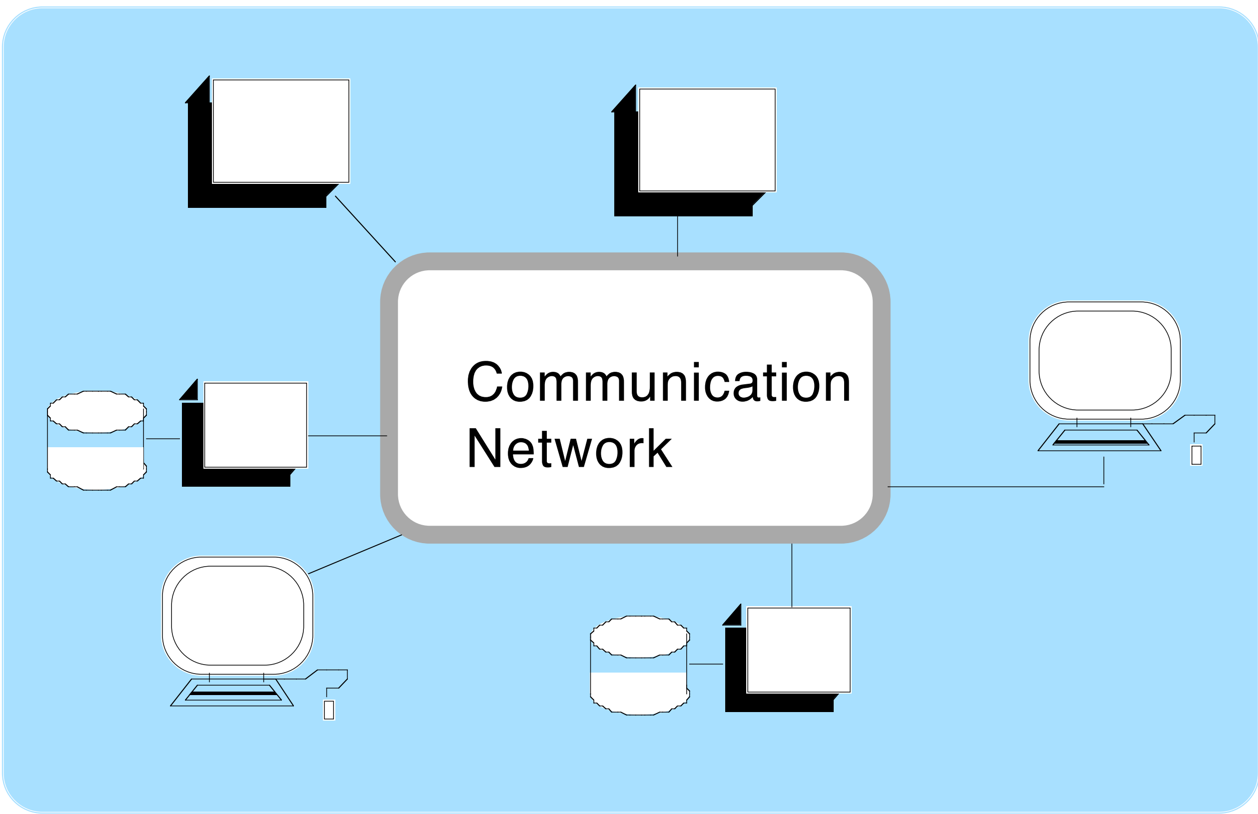
Another View: Concurrent Program



Expanding the Definition

- As we'll see later in the semester, the definition of "program" is expanding
  - In particular, its no longer true that
    - single program = single process
- A single "logical" programs can follow any of these patterns these days
  - single process, single thread, single machine (single core or multi-core)
  - single process, multiple threads, single machine
  - multi-process, single threaded, single or multiple machines
  - multi-process, multi-threaded, single or multiple machines
- Concurrent programs can perform multiple computations in parallel and can control multiple external activities which occur at the same time
  - A key difficulty with dealing with programming concurrent programs is dealing with interactions between threads or processes

Concurrent and Distributed Software?



- Interacting, concurrent software components of a system:
  - single machine → shared memory interactions
    - the value of a shared variable can become meaningless if updated by multiple threads simultaneously (without some form of control)
  - multiple machines → network interactions

Why Concurrent Programming?

- Performance gain from multi-core hardware
  - True parallelism
- Increased application throughput
  - an I/O call need only block one thread
- Increased application responsiveness
  - high priority thread for user requests
- More appropriate structure
  - for programs which interact with the environment, control multiple activities, and handle multiple events
  - by partitioning the application's thread/process structure to match its external conditions (e.g. one thread per activity)

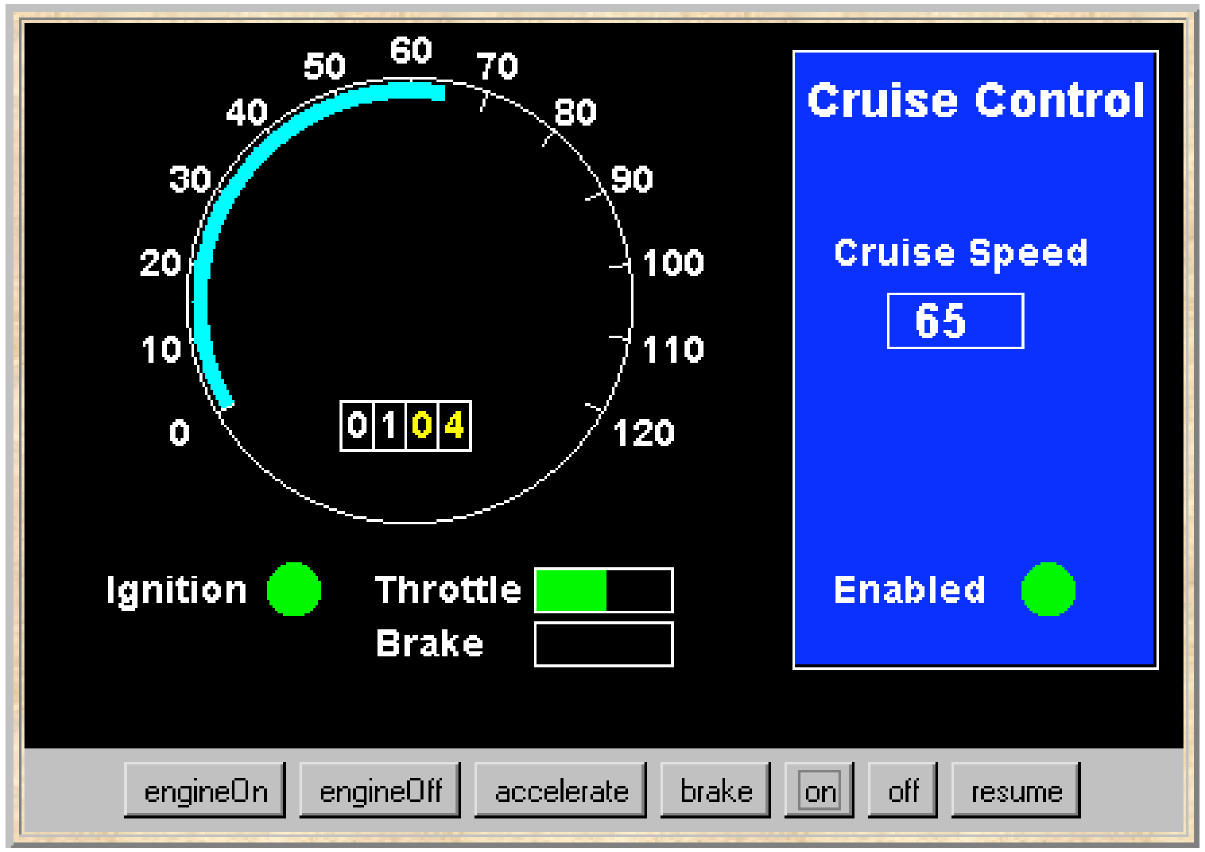
Concurrent Programming: Hard to Ignore

- Multi-core chips are becoming widely deployed
  - Only multi-threaded applications will see the performance benefits that these chips offer
- Programming for the Web often requires concurrent programming
  - AJAX
  - Web browsers are examples of multi-threaded GUI applications
    - without threads the UI would block as information is downloaded
- Lots of other domains in which concurrency is the norm
  - Embedded software systems, robotics, “command-and-control”, …

BUT…

- While concurrency is widespread it is also error prone
    - Programmers trained on single-threaded programs face unfamiliar problems: synchronization, race conditions, deadlocks, etc.
- Example: Therac-25
        - Concurrent programming errors contributed to accidents causing death and serious injury
    - Mars Rover
        - Problems with interaction between concurrent tasks caused periodic software resets reducing availability for exploration

Cruise Control System



Two Threads: Engine and Control

Once implemented:
- Is the system safe?
- Would testing reveal all errors?
- How many paths through system?

- *Requirements*
  - *Controlled by three buttons*
    - *on, off, resume*
  - *When ignition is switched on and on button pressed, current speed is recorded and* ***system maintains the speed of the car at the recorded setting***
  - *Pressing the brake, the accelerator, or the off button disables the system*
  - *Pressing resume re-enables the system*

Models to the Rescue!

- The answer to our questions on the previous slide is to **construct a model** of the concurrent behavior of the system and then **analyze the model**
  - This is one benefit of models, **they focus on one particular aspect of the world** and **ignore all others**
- Consider the model on the front of the Concurrency book
  - The picture shows the image of a real-world train next to its model
  - Depending on the model, you can ask certain questions and get answers that reflect the answers you would get if you asked “the real system”
  - For the train model, you might be able to ask
    - What color is the train? How long is it? How many cars does it have?
  - But not
    - What’s the train’s maximum speed?
    - How does it behave when a car derails?

Models, continued

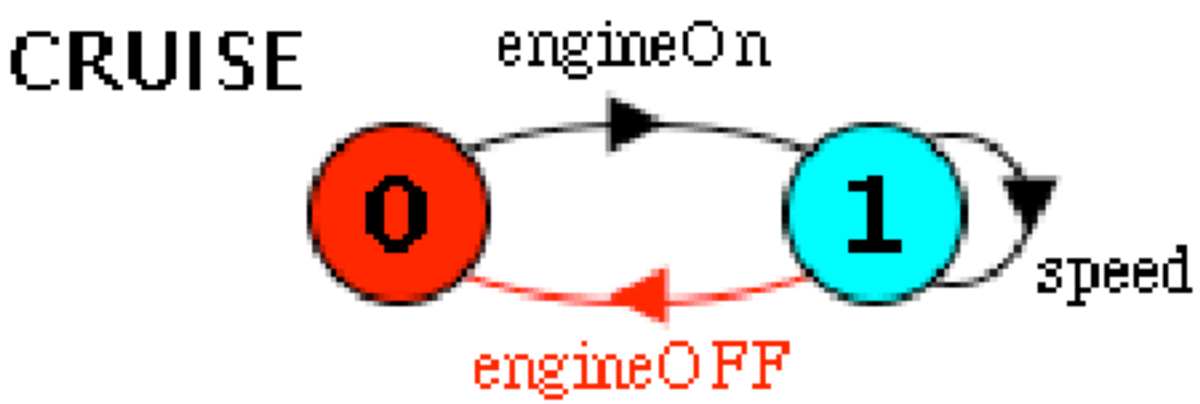
- In summary, a model is a simplified representation of the real world
  - A model airplane, e.g., used in wind tunnels models only the external shape of the airplane
  - The reduction in scale and complexity achieved by modeling allows engineers to analyze properties of the model
  - The earliest models were physical (like our model train)
    - modern models tend to be mathematical and are analyzed by computers
- Engineers use models to gain confidence in the adequacy and validity of a proposed design
  - focus on an aspect of interest — concurrency
  - can animate model to visualize a behavior
  - can analyze model to verify properties

Models for Concurrency

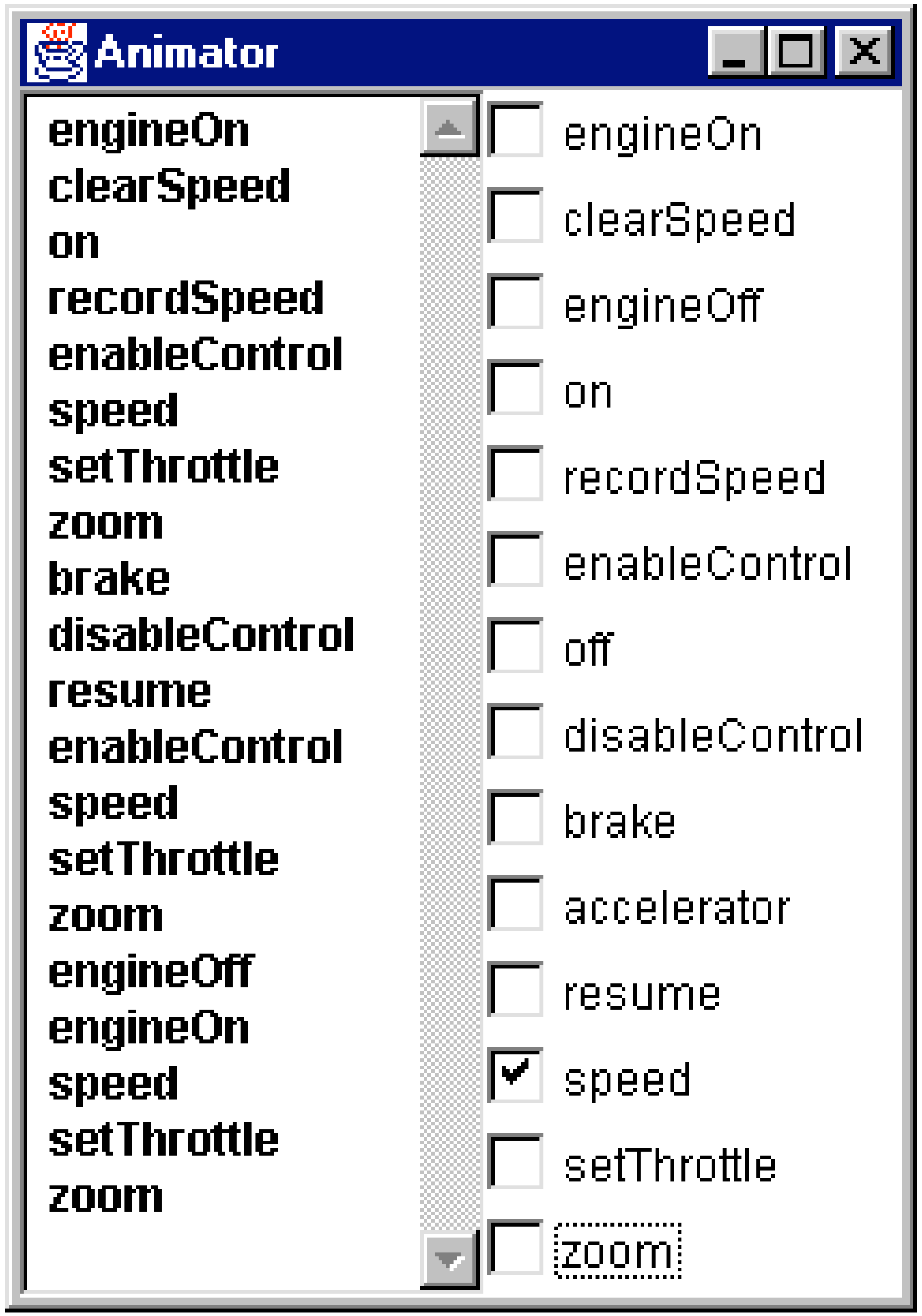
- When modeling concurrency, we make use of a type of finite state machine known as a **labeled transition system** or **LTS**
  - These machines are described textually with a specification language called **finite state processes** (**FSP**)
- These machines can be displayed and analyzed by an analysis tool called **LTSA**
  - Note: LTSA requires a Java 2 run time system, version 1.5.0 or later
  - On Windows and Mac OS systems, you can run the LTSA tool by double clicking on its jar file
  - Note: Its not the most intuitive piece of software, but once you “grok it”, it provides all of the advertised functionality

Modeling the Cruise Control System

- We won’t model the entire system, lets look at a simplified example
- Given the following specification
  - CRUISE = (engineOn -> RUNNING),
  - RUNNING = (speed -> RUNNING | engineOFF -> CRUISE).
- We can generate a finite state machine that looks like this



LTSA



- LTSA allows us to enter specifications and generate state machines like the ones on the previous slide
- It can also be used to “animate” or step through the state machine
- Lets see a demo
- Note: animation at left shows the problem we encountered before with the cruise control system

LTSA, continued

- Using a modeling tool, like LTSA, allows us to understand the concurrent behaviors of systems, like the cruise control system, BEFORE they are implemented
  - This can save a lot of time and money, as it is typically easier to test and evolve a model's behavior than it is to implement the system in a programming language

Applying Concepts/Models via Programming

- Textbook makes use of Java to enable practice of these concepts
- Java is
  - widely available, generally accepted, and portable
  - provides sound set of concurrency features
- Java is used for all examples, demo programs, and homework exercises in textbook
  - This is not to say that Java is the ONLY language that supports concurrency; many languages have concurrency feature built-in or available via third-party libraries
- The book makes use of “toy programs” as they can focus quickly on a particular class of concurrent behavior

Wrapping Up

- Concepts
  - We adopt a model-based approach for the design and construction of concurrent programs
- Models
  - We use finite state machines to represent concurrent behavior
- Practice
  - We use Java for constructing concurrent programs
- We will be presenting numerous examples to illustrate concepts, models and demonstration programs

Coming Up Next

- Lecture 4: Processes and Threads
  - Chapter 2 of Magee and Kramer
- Lecture 5: Modeling the Process and Life Cycle
  - Chapter 2 of Pfleeger and Atlee