Introduction to Concurrency

Kenneth M. Anderson
University of Colorado, Boulder

CSCI 5828 — Lecture 4 — 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 is based on their SE research into modeling concurrency over the past 20 years

Why worry?

- “Concurrency is hard and I’ve only ever needed single-threaded programs: Why should I care about it?”
- Answer: multi-core computers, increasing use of clusters
  - Growth rates for chip speed are flattening
    - “lets wait a year and our system will run faster!”: No longer!
  - Instead, chips are becoming “wider”
    - more cores, wider bus (more data at a time), more memory
- As chips are not getting faster (the same way they used to), a single-threaded, single process application is not going to see any significant performance gains from new hardware

New Model

- Instead, the way in which **software** will see performance gains with new hardware is if **they are designed to get faster the more processors they have available**
  - **This is not easy**: the computations that an application performs has to be amenable to parallelization
- Such an application will see noticeable speed improvements when run on machines with more processors
  - Laptops currently have 2-cores, will soon have 4-cores and Intel has an 80-core beast waiting in the wings
    - A system written for n-cores could potentially see an 80x speed-up when run on such a machine

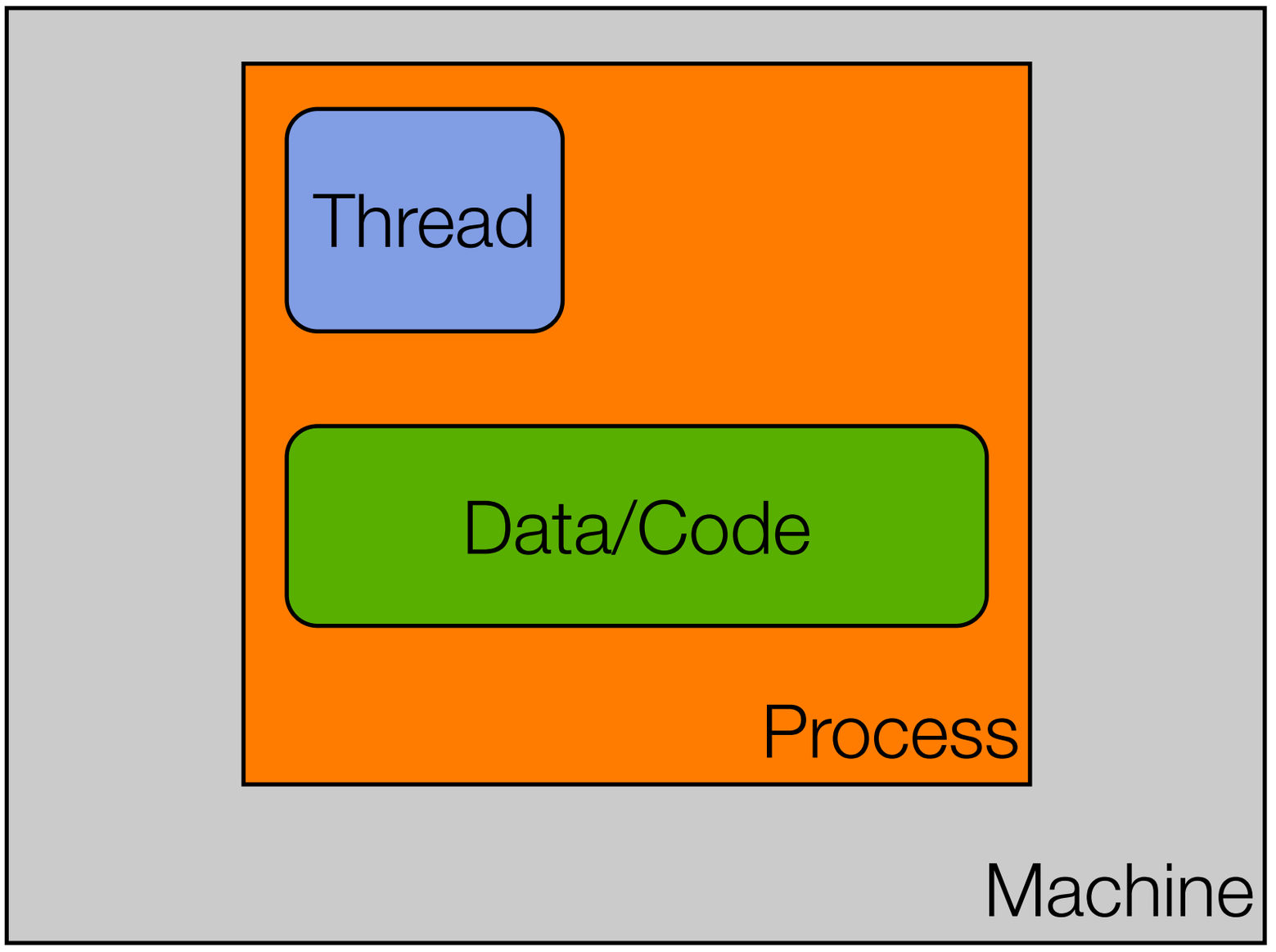
In addition…

- Concurrent programming is becoming hard to ignore
- In addition to the increasing presence of multi-core computers there are lots of other domains in which concurrency is the norm
  - Embedded software systems, robotics, “command-and-control”, high-performance computing (use of clusters), …
- Closer to home: Web programming often requires concurrent programming
  - AJAX
  - Web browsers are examples of multi-threaded GUI applications
    - without threads the UI would block as information is downloaded

BUT…

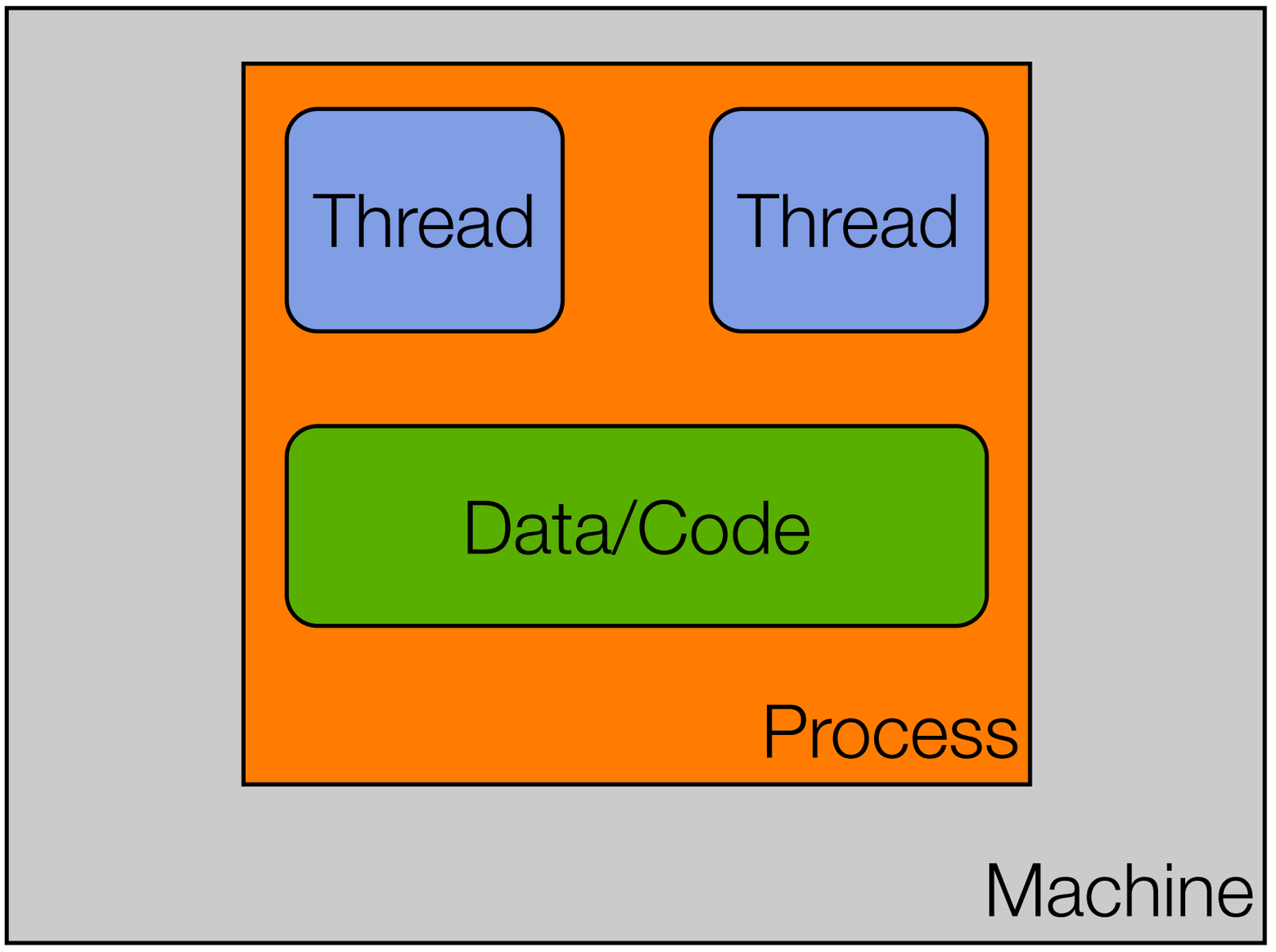
- While concurrency is widespread it is also error prone
  - Programmers trained on single-threaded programs face unfamiliar problems: synchronization, race conditions, deadlocks, etc.
- Example: Therac-25
  - Concurrent programming errors contributed to accidents causing death and serious injury
  -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

Basics: **Single** Thread, **Single** Process, **Single** Machine



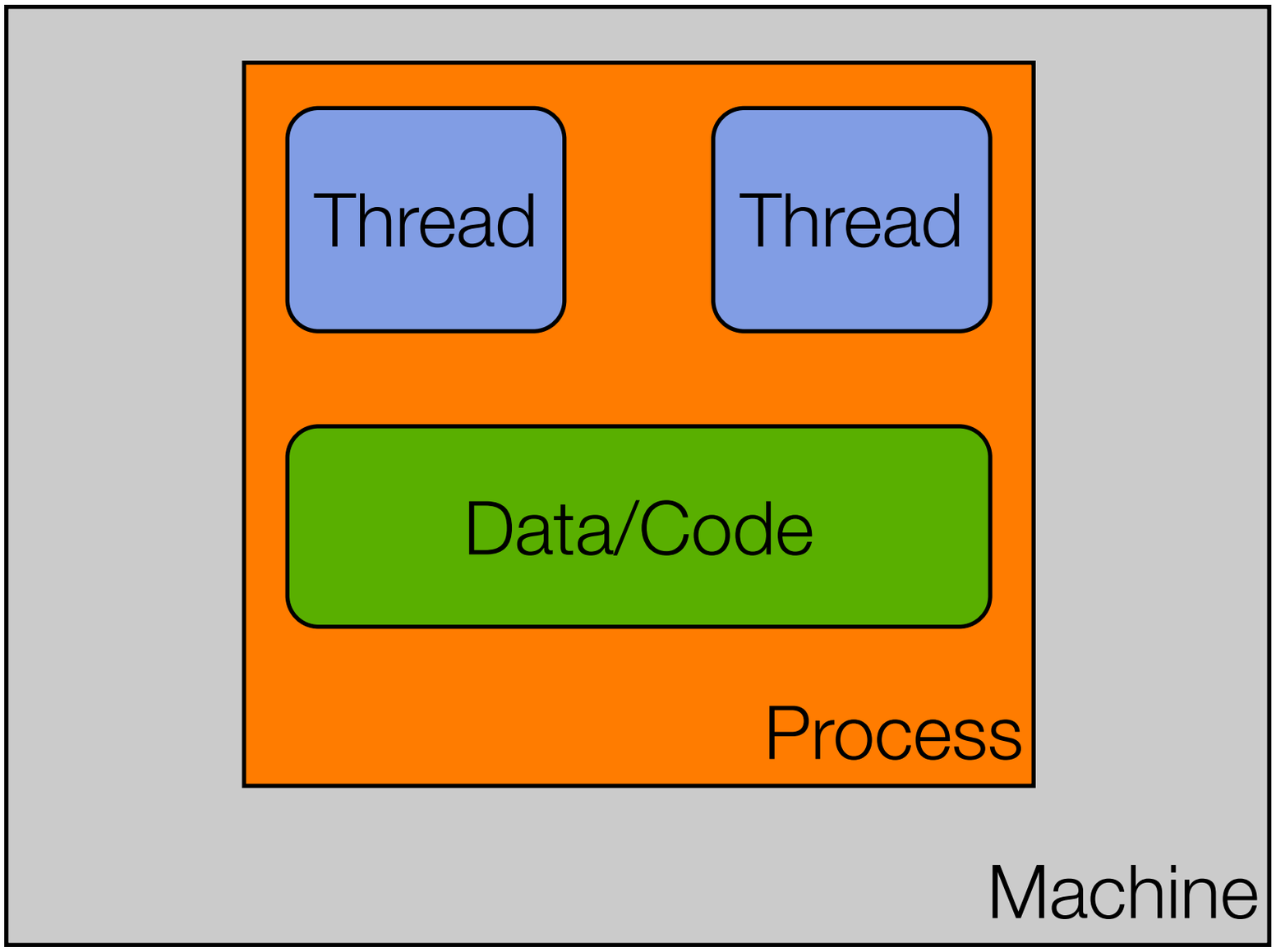
Sequential Program == Single Thread of Control

Basics: **Multiple** Thread, Single Process, Single Machine



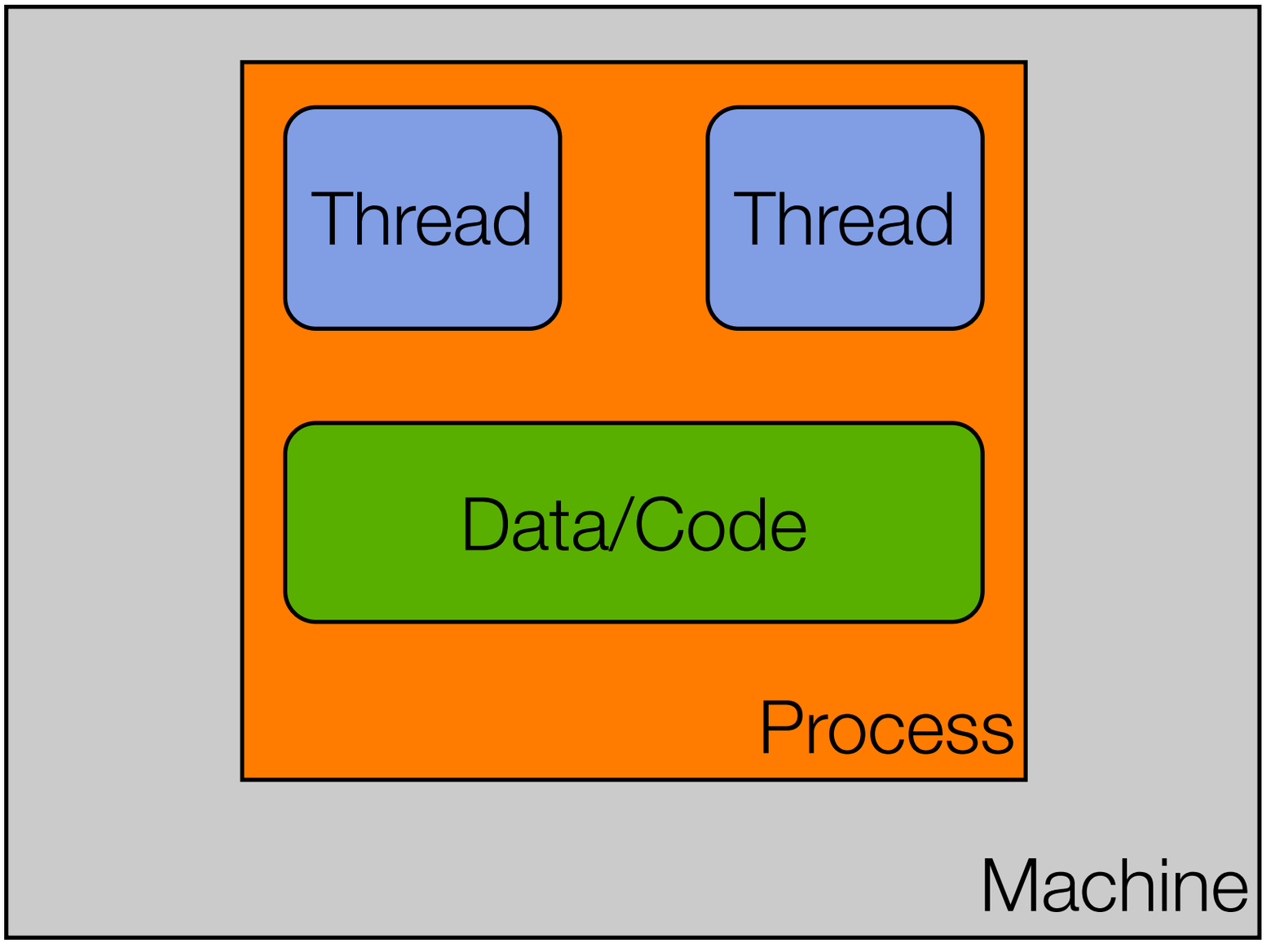
Concurrent Program == Multiple Threads of Control

Multi-Thread: **But is it truly parallel?**



Concurrent Program == Multiple Threads of Control

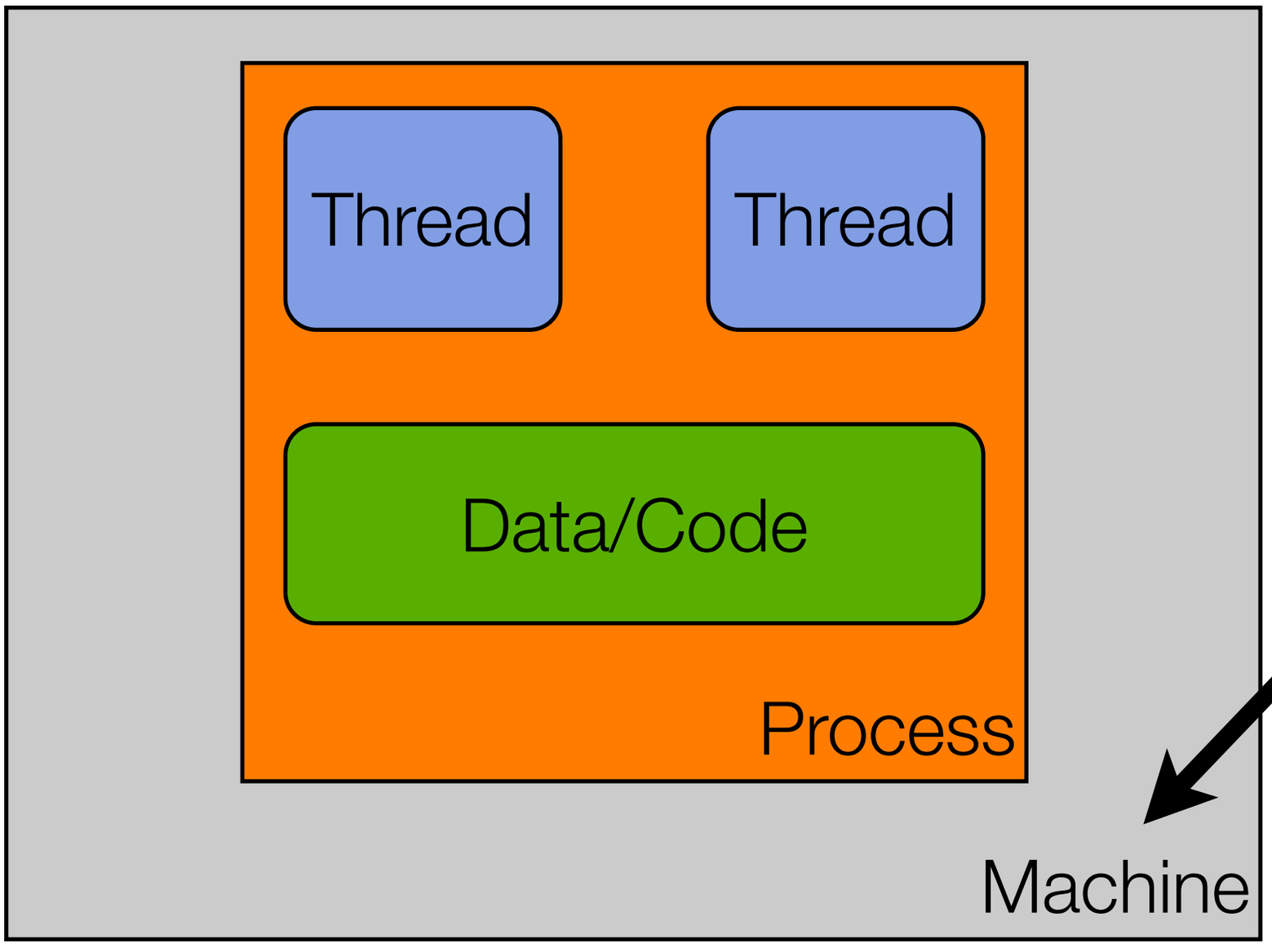
Multi-Thread: **But is it truly parallel?**



We may have multiple threads in this process, but we may not have events truly occurring in parallel. Why not?

Concurrent Program == Multiple Threads of Control

Multi-Thread: **But is it truly parallel?**



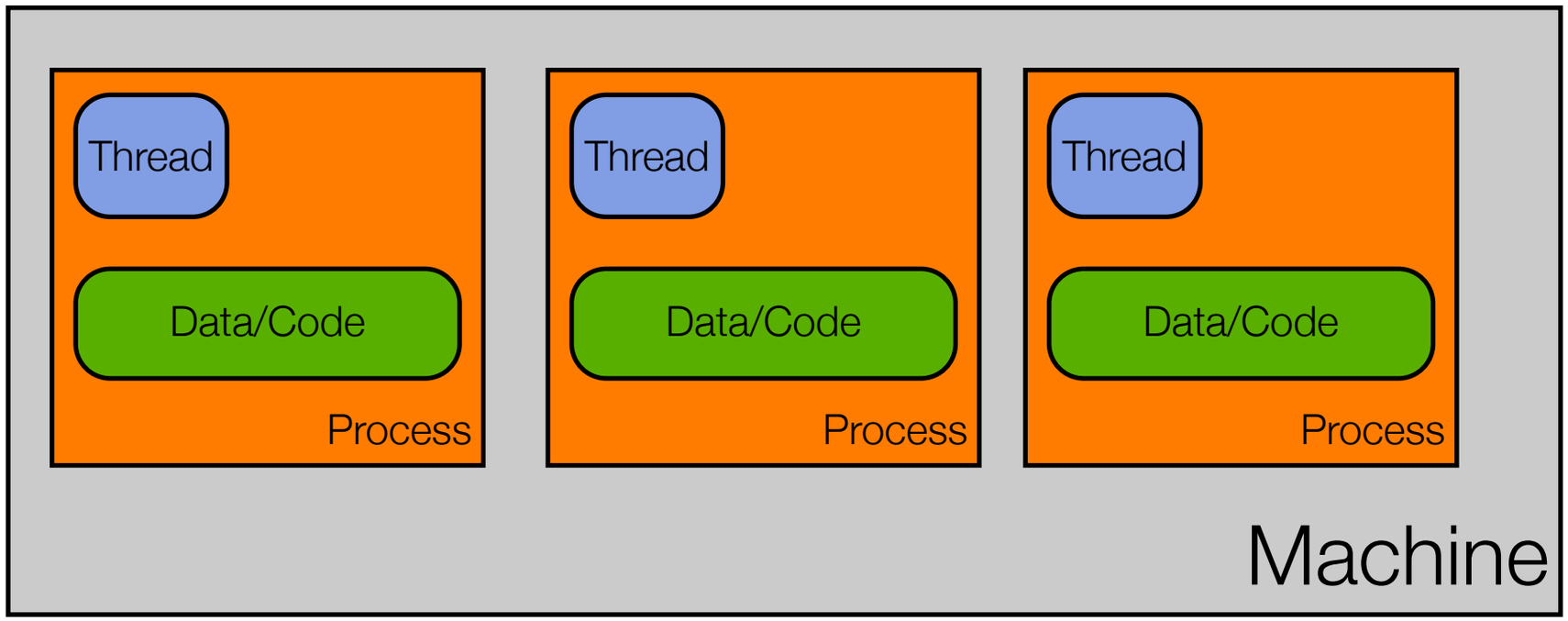
We may have multiple threads in this process, but we may not have events truly occurring in parallel. Why not?

It depends on the machine!

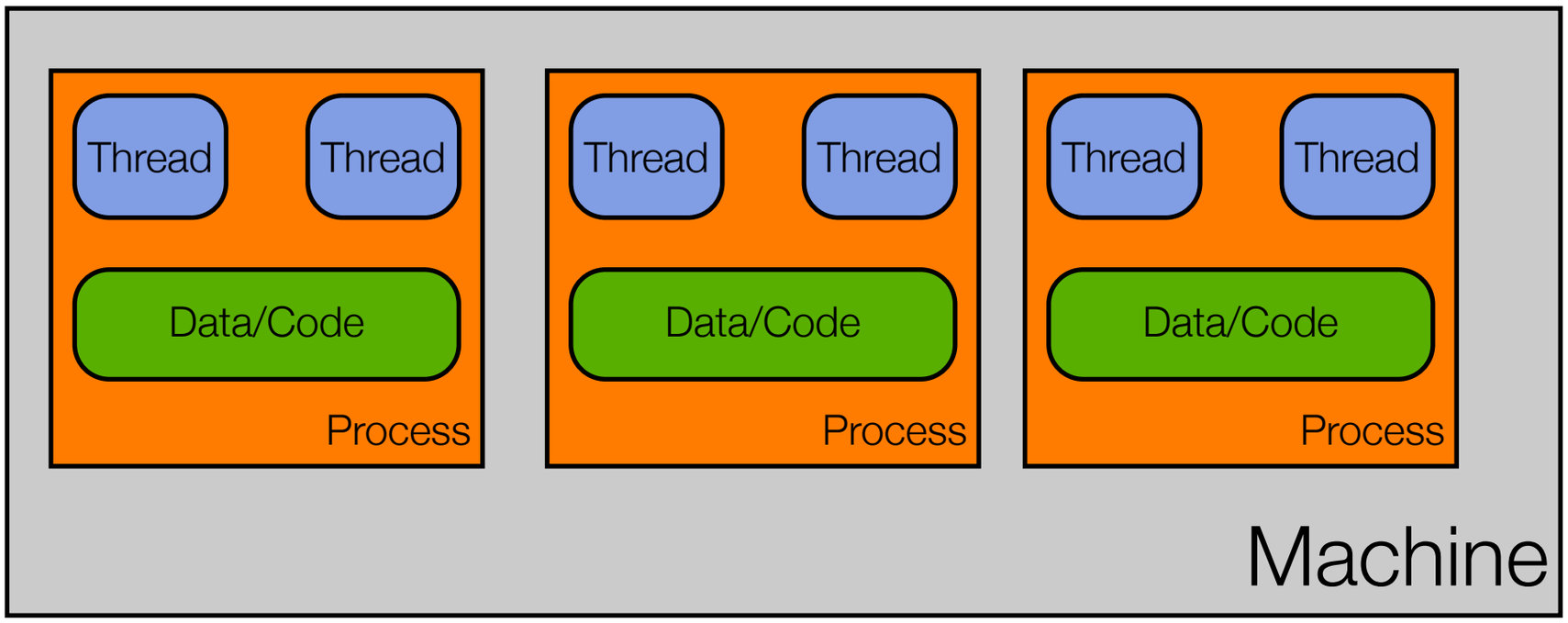
If the machine has multiple processors, then **true parallelism can occur**. Otherwise, parallelism is **simulated**

Concurrent Program == Multiple Threads of Control

Basics: Single Thread, **Multiple** Process, Single Machine

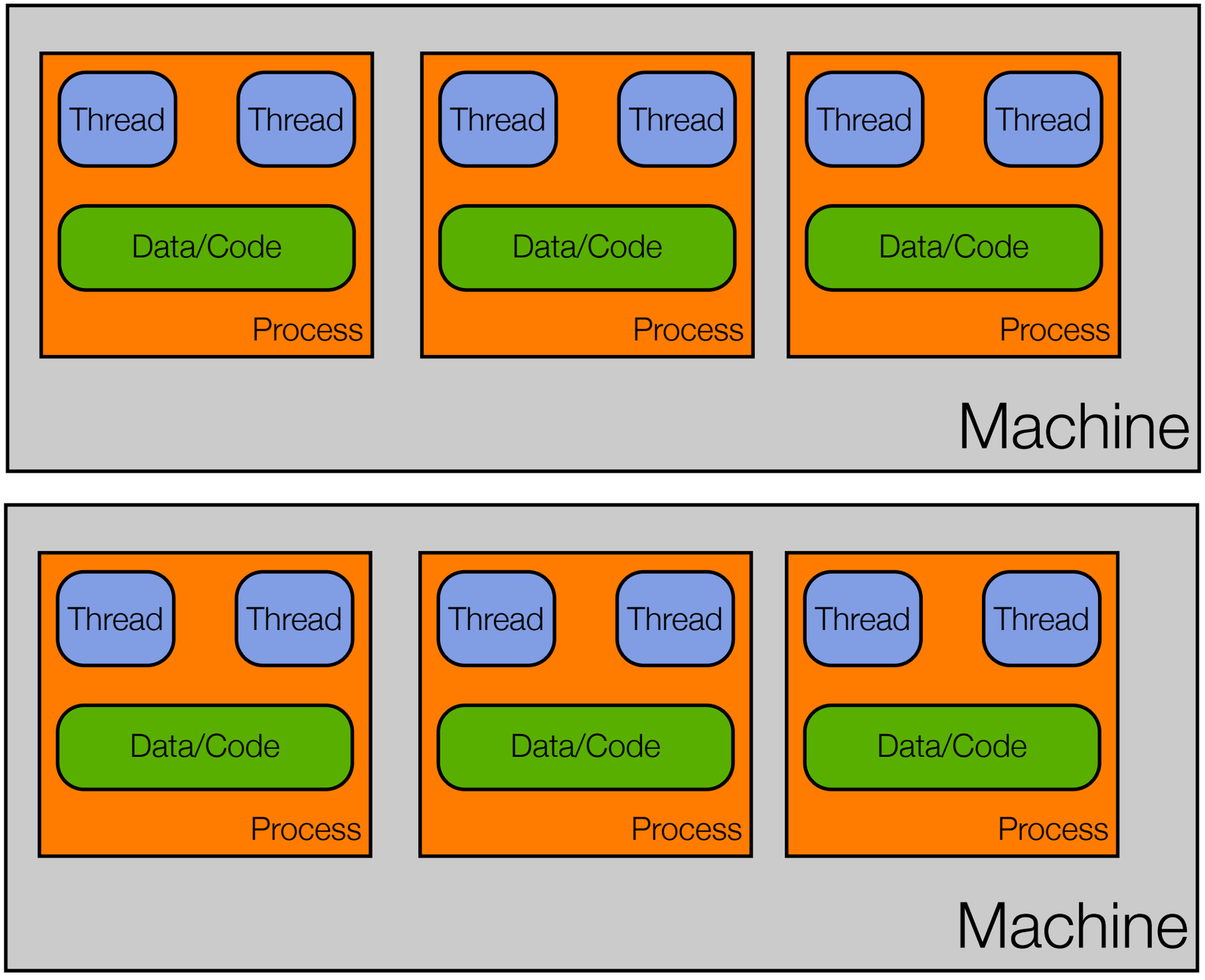


Basics: **Multi**-thread, **Multi**-Process, Single Machine



Note: You can have way more than just two threads per process.

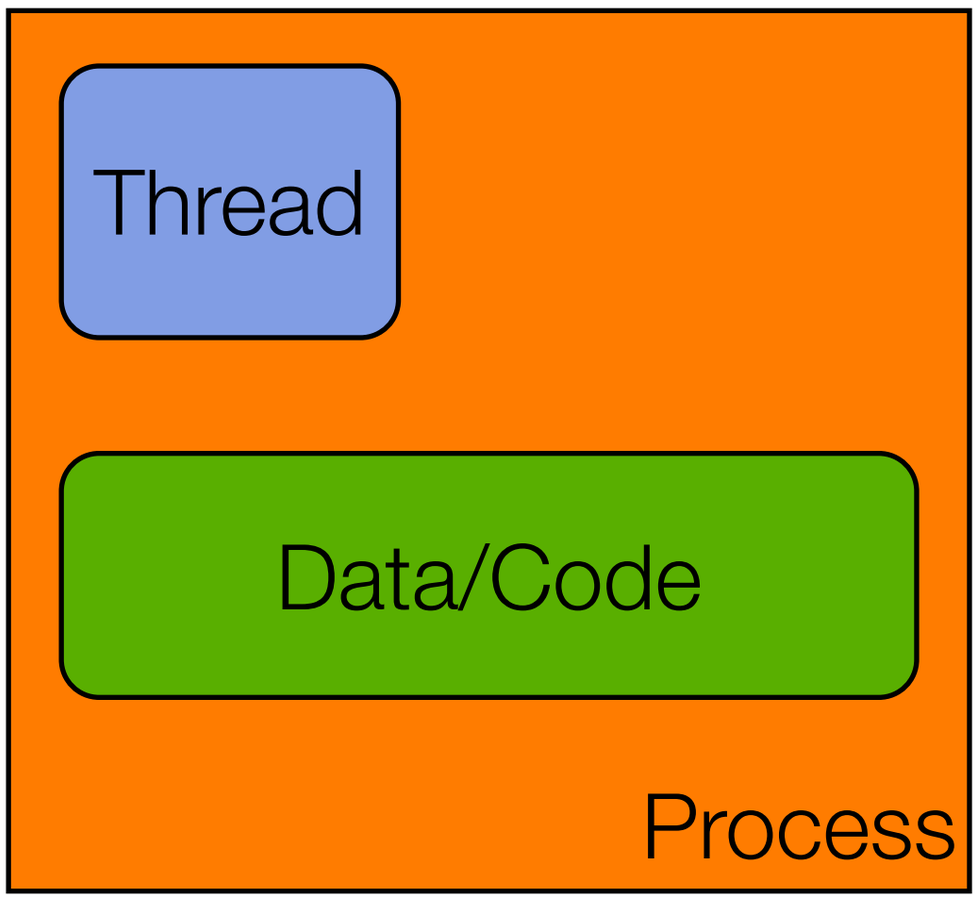
Basics: **Multi-everything**



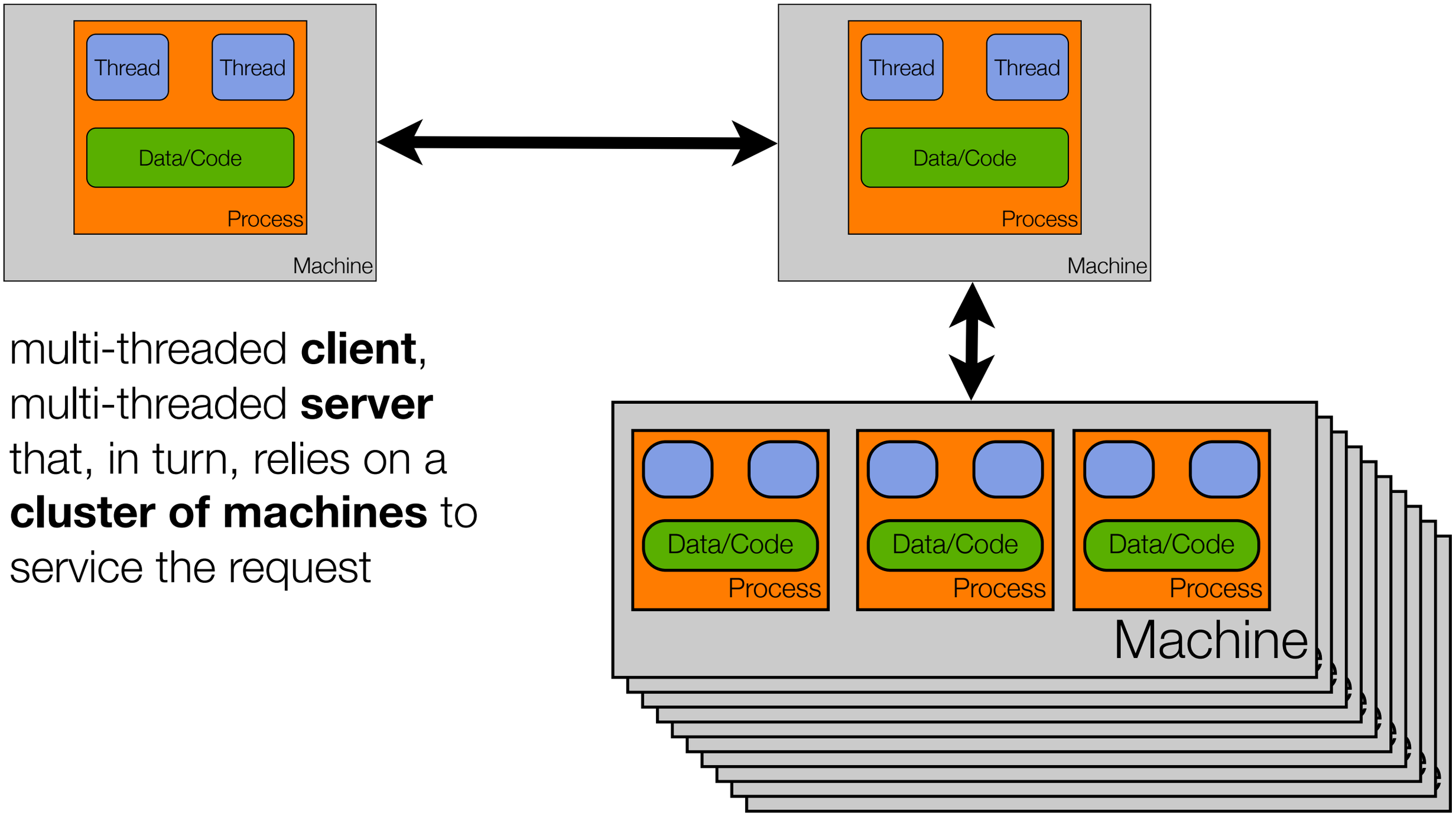
Applications are Dead! Long Live Applications!

Due to the ability to have multiple threads, multiple processes, and multiple machines work together on a single problem, the notion of an application is changing. It used to be that:

Application ==

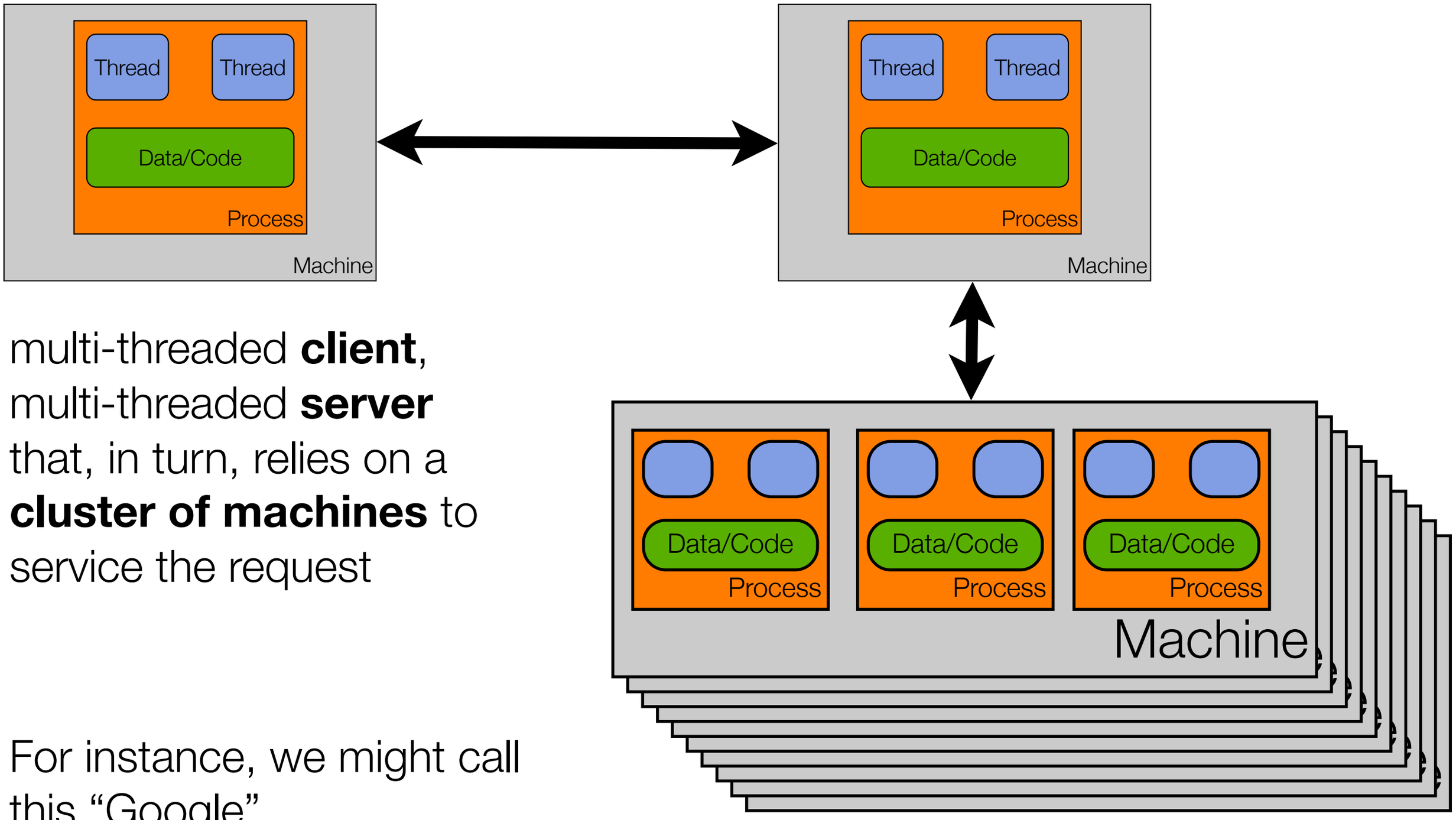


Now… we might refer to this as “an application”



multi-threaded **client**,
multi-threaded **server**
that, in turn, relies on a
cluster of machines to
service the request

Now… we might refer to this as “an application”



multi-threaded **client**,
multi-threaded **server**
that, in turn, relies on a
cluster of machines to
service the request

For instance, we might call this “Google”

Architecture Design Choices

- When designing a modern application, we now have to ask
  - How many machines are involved?
  - What components will be deployed on each machine?
  - For each component:
    - Does it need concurrency?
    - If so, will we achieve concurrency via
      - multiple threads?
      - multiple processes?
      - both?

Consider Chrome (I)

- Google made a splash last year by announcing the creation of a new web browser that is
  - multi-process (one process per tab) and
  - multi-threaded (multiple threads handle loading of content within each process)
- In typical Google style, they documented their engineering choices via a comic book
  - <http://www.google.com/googlebooks/chrome/index.html>

Consider Chrome (II)

- Some of the advantages they cite for this design
  - stability
    - single-process, multi-threaded browsers are vulnerable to having a crash in one tab bring down the entire browser
  - speed
    - multi-process browsers can be more responsive due to OS support
  - security
    - exploits in single-process browsers are easier if malware loaded in one tab can grab information contained in another tab; much harder to grab information across processes

Stainless

- Chrome is not available for my platform
  - But that has not stopped independent developers from creating a multi-process, multi-threaded browser for the Mac
  - See <http://www.stainlessapp.com/> for details
- Demo

Other benefits to multi-process design*

- Lots of existing applications that do useful things
  - Think of all the powerful command line utilities found in Unix-based platforms; You can take advantage of that power in your own application
    - Create a sub-process, execute the desired tool in that process, send it input, make use of its output
- Memory leaks in other programs are not YOUR memory leaks
  - As soon as the other program is done, kill the sub-process and the OS cleans up
- Flexibility: An external process can run as a different user, can run on a different machine, can be written in a different language, …

* Taken from discussion in Cocoa Programming for Mac OS X by Aaron Hillegass

Example: ZIPspector

- ZIPspector is a GUI application that makes use of the Unix command line tool zipinfo to display the contents of a zip archive
  - This example taken from Aaron Hillegass's Cocoa Programming for Mac OS X
- ZIPspector GUI runs in one process; When you select a zip archive, the program
  - creates a subprocess, executes zipinfo in it, captures results, allows subprocess to die, and then displays results in table
- The multi-process aspect of this app happens very quickly!

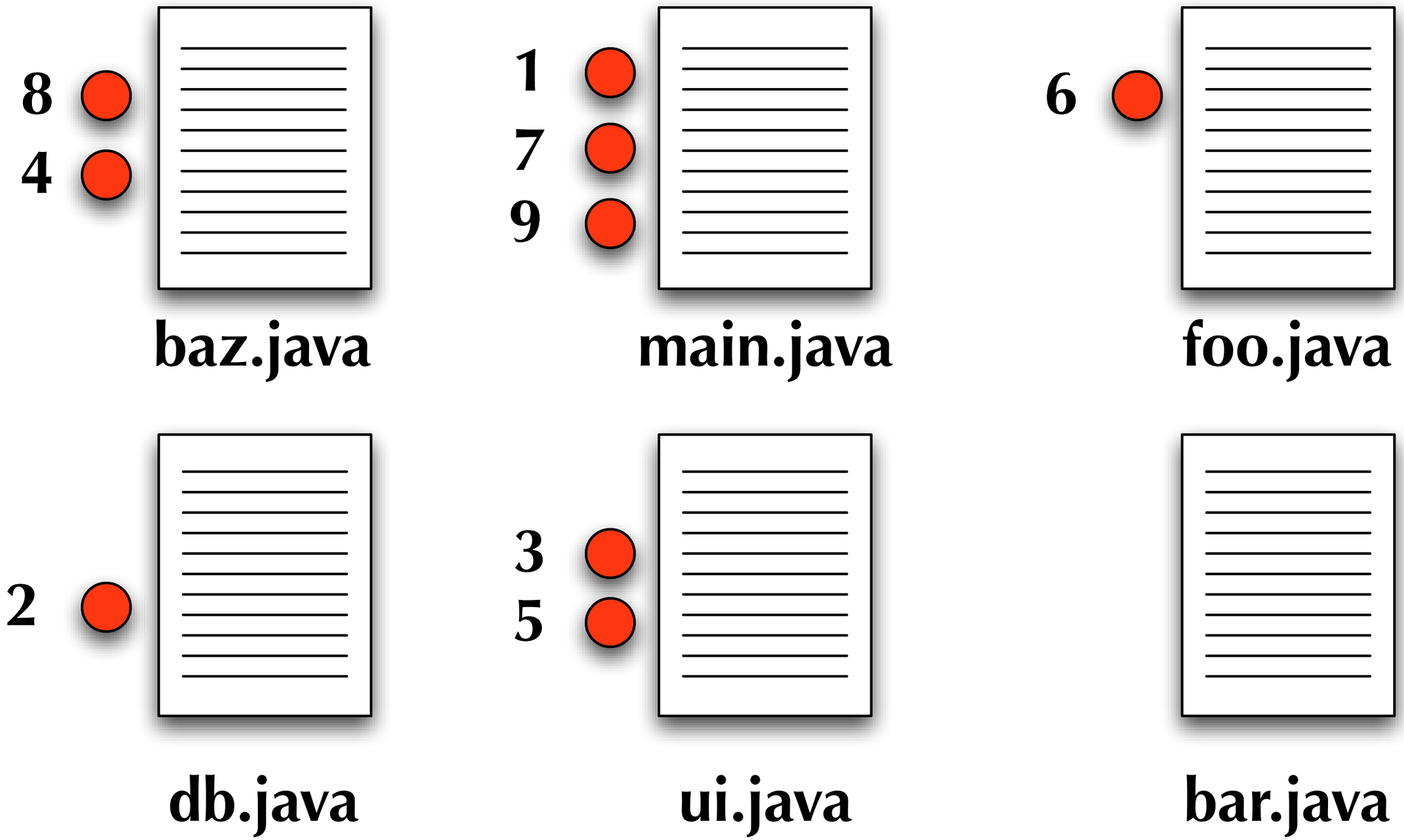
Review

- 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

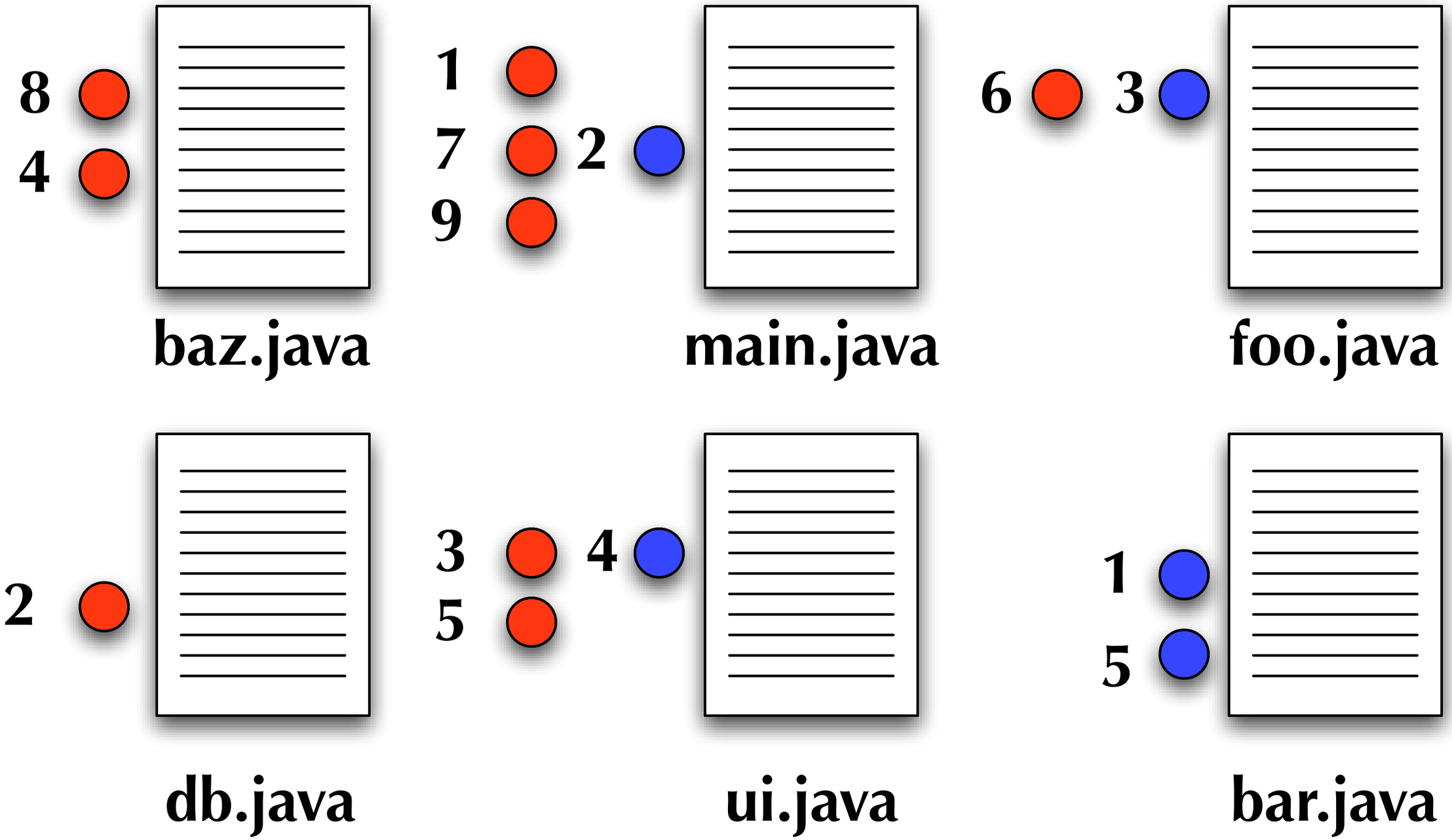
So, what's the problem?

- Concurrent programs can perform multiple computations in parallel and can control multiple external activities which occur at the same time
- Sounds great. So what's the problem?
  - Designing/Implementing/Testing concurrent programs is hard
    - Much harder than testing sequential programs due to
      - **interference**: two threads accessing shared data inappropriately
      - **race conditions**: behaviors that appear in one configuration but don't appear in other configurations
      - **deadlock**: threads block waiting for each other

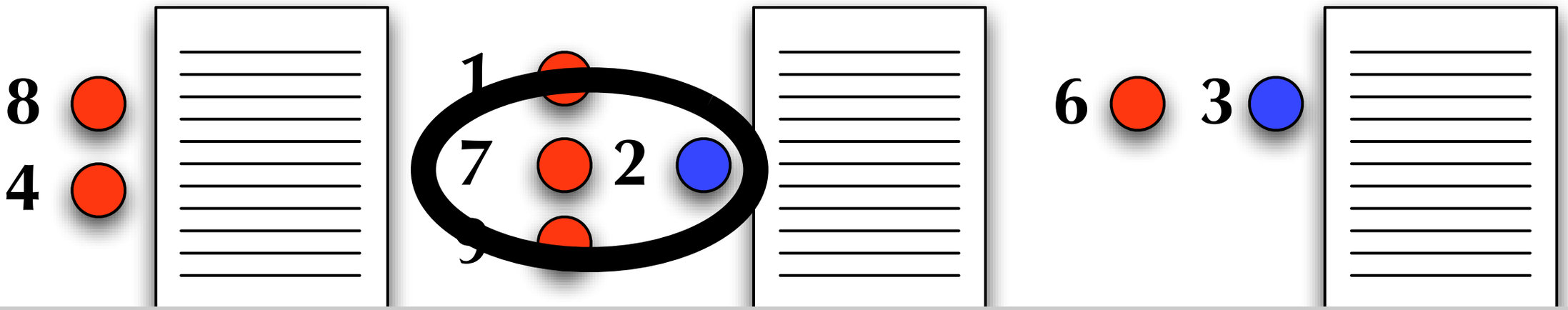
Another View: Sequential Program



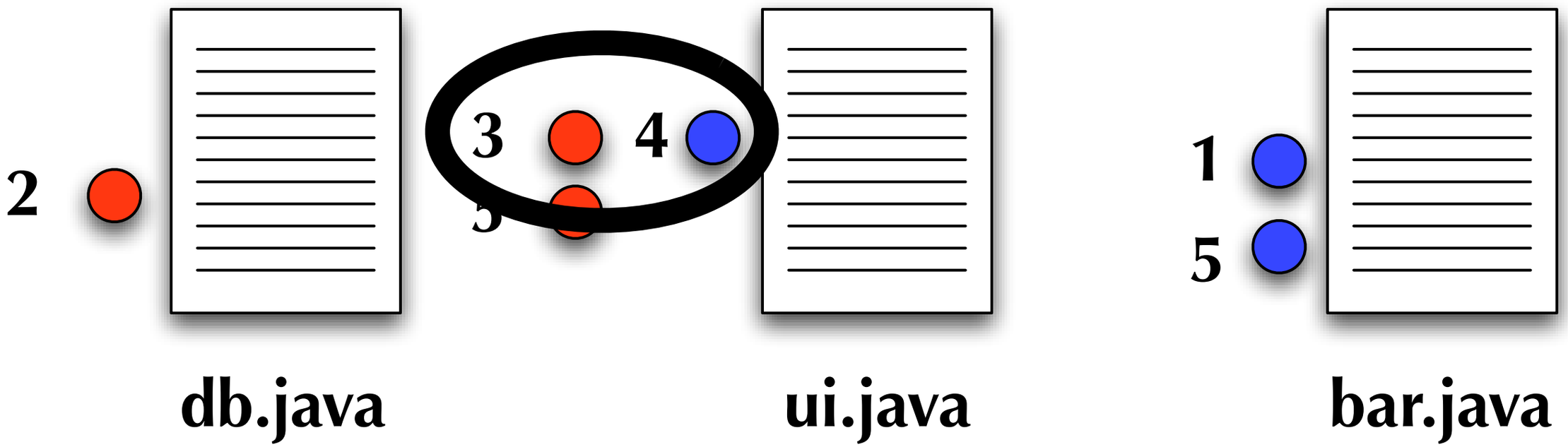
Another View: Concurrent Program



Example of Interference



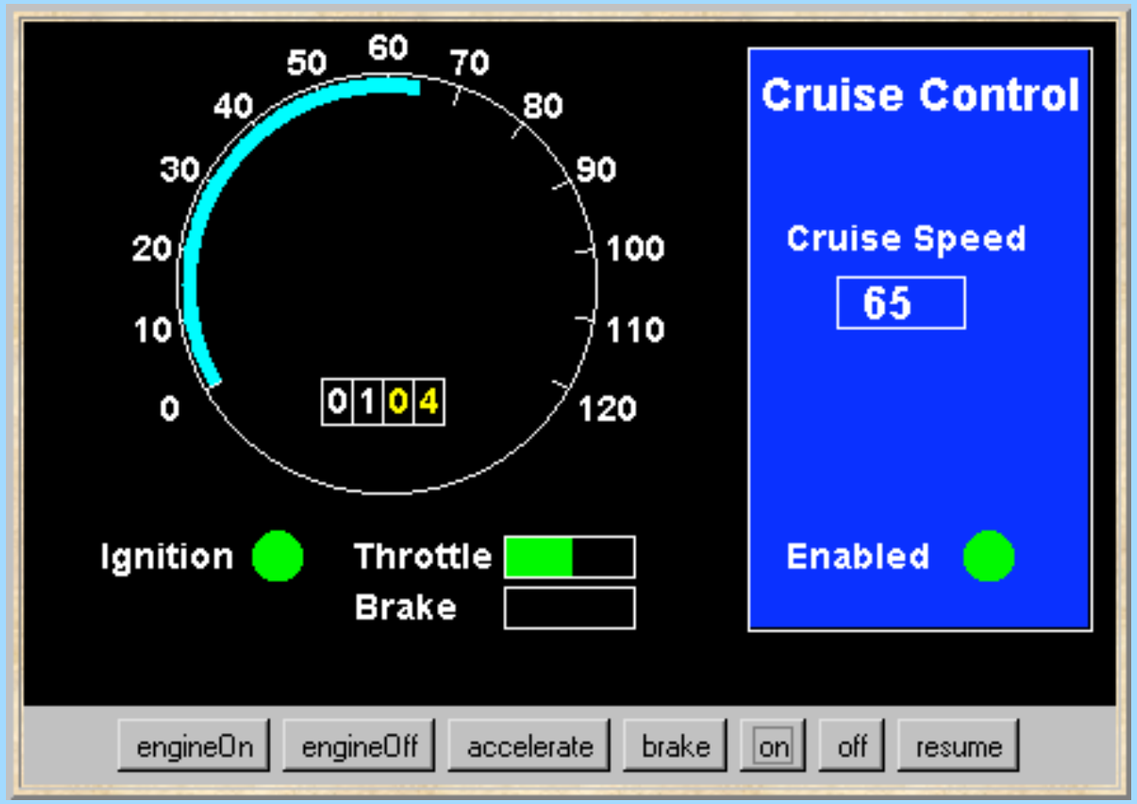
The potential for interactions… two threads hitting the same method at the same time, potentially corrupting a shared data structure



Benefits of 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)

Ex.: Cruise Control System



Two Threads: Engine and Control

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!

- To answer, we need a model of the concurrent behavior of the system and then we need to analyze it
  - 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 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”

Models to the Rescue!

- 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

- 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 analyzed by computers

Models, continued

- 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 support hypothesis testing
  - we make observations and test against our model's predictions
  - if predictions match observations, we gain confidence in the model; otherwise, we update model and try again

Models for Concurrency

- When modeling concurrency
  - our book makes use of a type of finite state machine known as a labeled transition system (LTS)
    - LTS == Model
  - These machines are described textually with a specification language called finite state processes (FSP)
    - FSP == Specification Language
      - Used to generate an instance of an LTS

Models for Concurrency

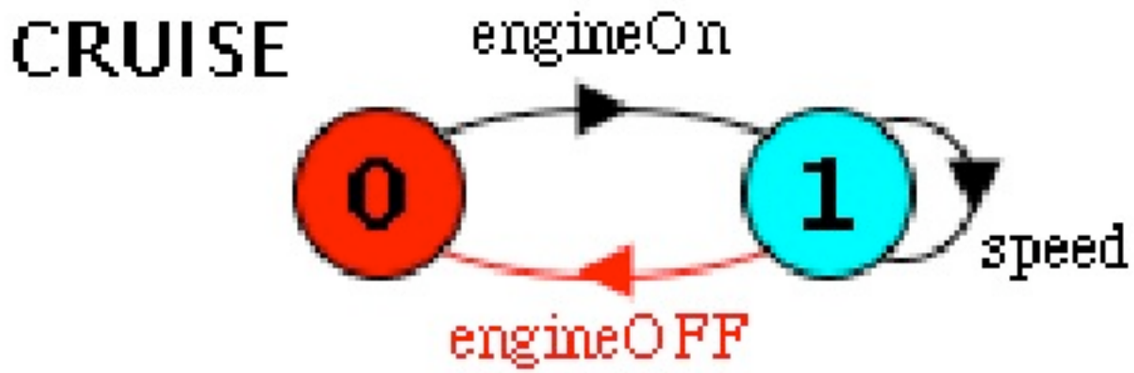
- These machines can be displayed and analyzed by an analysis tool called LTSA
  - Note: LTSA requires a Java 2 run time system, version 1.5.0 or later
  - On Windows and Mac OS systems, you can run the LTSA tool by double clicking on its jar file
  - Note: Its not the most intuitive piece of software, but once you “grok it”, it provides all of the advertised functionality

Modeling the Cruise Control System

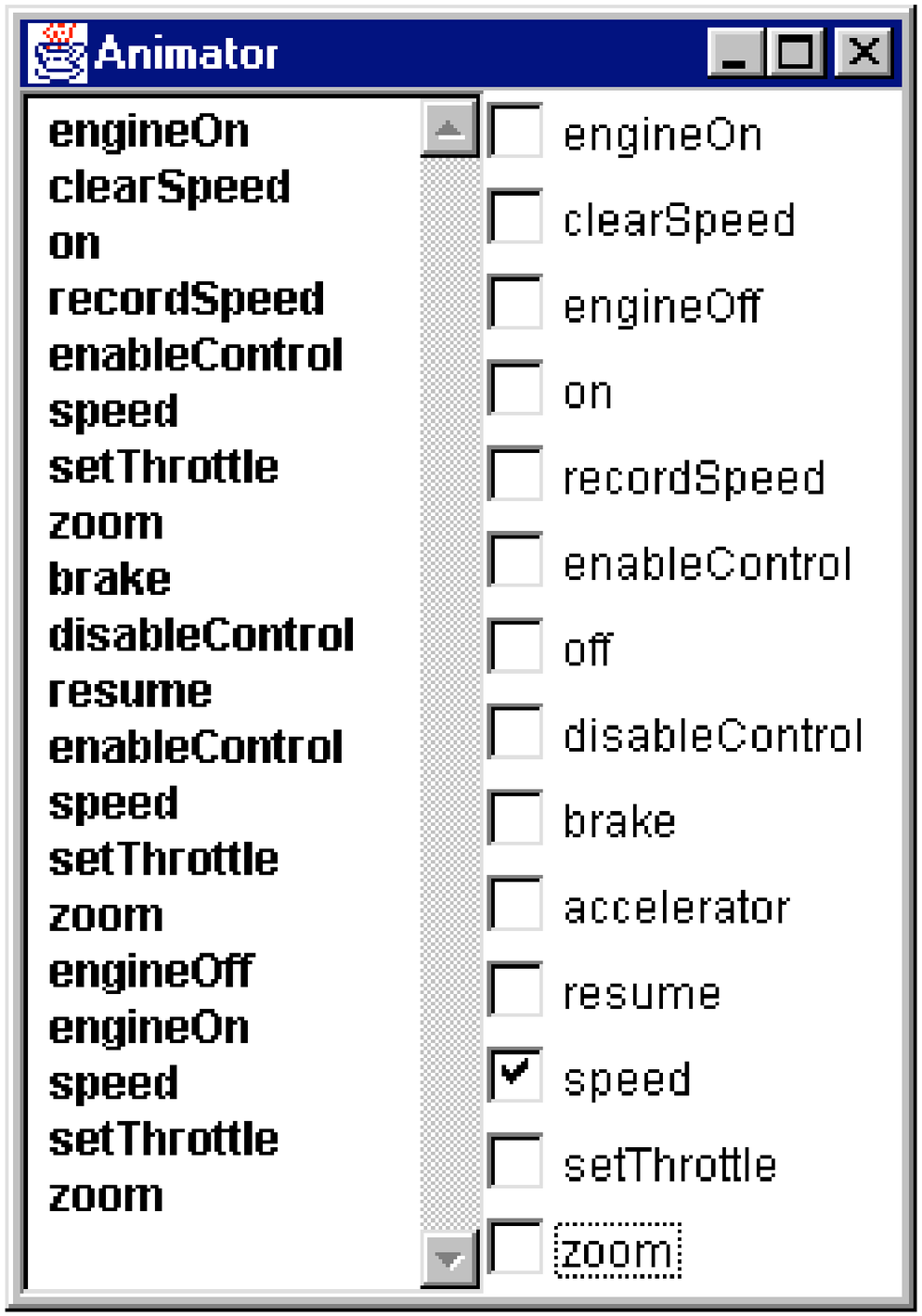
- We won't model the entire system
  - lets look at a simplified example
- Given the following specification

```
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 uses 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

Applying Concepts/ Models via Programming

- This is not to say that Java is the ONLY language that supports concurrency; many languages have concurrency features built-in or available via third-party libraries
  - As a result, I am open to students using other languages, as long as the language has concurrency features similar to Java
- 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
  - finite state machines to represent concurrent behavior
- Practice
  - Book uses Java for constructing concurrent programs
- We will be presenting numerous examples to illustrate concepts, models and demonstration programs

Coming Up Next

- Lecture 5: Gathering Requirements
  - Chapter 2 of Pilone & Miles
- Lecture 6: Processes and Threads
  - Chapter 2 of Magee and Kramer