Kernel Implementations I
So, What’s Inside the Computer?

• Consider any popular video game, e.g., Nintendo Wii bowling
• Are there bowling balls and bowling pins inside the game console’s computer?
So, What’s Inside the Computer?

• Consider any popular video game, e.g., Nintendo Wii bowling
• Are there bowling balls and bowling pins inside the game console’s computer?
  – Of course not!
• What’s really inside the computer, then, that makes bowling-like behavior?
So, What’s Inside the Computer?

• Consider any popular video game, e.g., Nintendo Wii bowling.
• Are there bowling balls and bowling pins inside the game console?
  – Of course not!
• What’s really inside the computer, then, that makes bowling-like behavior?

A thought experiment: What dynamic behavior would you see if you had a magical magnifying glass and could see inside the computer at any level of detail while it’s running?
A Useful Metaphor
A Tower of Abstractions

• Bowling pins?
• Vectors?
• Numbers?
• Bits?
• Voltages?
• Electrons?
• ???
A Tower of Abstractions

- Bowling pins?
- Vectors?
- Numbers?
- Bits?
- Voltages?
- Electrons?
- ???

These are all “just” mathematical models, i.e., abstractions used to explain and predict behavior.
A Tower of Abstractions

• Bowling pins?
• Vectors?
• Numbers?
• Bits?

• Voltages?
• Electrons?
• ???

Domain: **Physics**
These models are supposed to match physical reality, and are discarded if they do not; limited by the physical world.
A Tower of Abstractions

- Bowling pins?
- Vectors?
- Numbers?
- Bits?
- Voltages?
- Electrons?
- ???

Domain: Computing

These models are entirely artificial (need not match physical reality); limited only by the creativity of the software engineer.
A Tower of Abstractions

- Bowling pins
- Vectors
- Numbers
- Bits
- Voltages
- Electrons
- ???
A Tower of Abstractions

- Bowling pins
- Vectors
- Numbers
- Bits
- Voltages
- Electrons
- ???

Numbers may be built on top of bits…
A Tower of Abstractions

- Bowling pins
- Vectors
- Numbers
- Bits
- Voltages
- Electrons
- ???

Bits may be built on top of voltages…
A Tower of Abstractions

• Bowling pins
• Vectors
• Numbers
• Bits
• Voltages
• Electrons
• ???

Voltages may be built on top of (?) electrons…
Interpretation of Representation

• Let’s not take the tower-building metaphor too far!

• A better approach is to think about interpreting a lower-level configuration (a.k.a. a representation) to get a higher-level value
A Tower of Abstractions

• Bowling pins
• Vectors
• Numbers
• Bits
• Voltages
• Electrons
• ???

(Configurations of) bits may be interpreted as numbers...
A Tower of Abstractions

- Bowling pins
- Vectors
- Numbers
- Bits
- Voltages
- Electrons
- ???

(Configurations of) voltages may be interpreted as bits...
A Tower of Abstractions

- Bowling pins
- Vectors
- Numbers
- Bits
- Voltages
- Electrons
- ???

(Configurations of) electrons may be interpreted as voltages…
First Example: QueueKernel

• For QueueKernel, one idea is to represent a Queue variable’s value by using a java.util.List variable

• By convention (of the OSU CSE components), a kernel class that directly represents the new type using a component from the Java libraries that is very similar, has a name ending in “L”
  – In this case, it is called Queue1L
Detailed Example: Queue1L

• What existing components (including built-in types of Java, and the Java libraries) could you build it on top of?
  – In other words, what could you use as a data representation that could be interpreted as a Queue value?
Context of **Queue1L**

Has bodies for the constructor, plus the 7 methods from **Standard** and **QueueKernel**.
Context of **Queue1L**

- **newInstance**
- **clear**
- **transferFrom**

- **constructor**
- **enqueue**
- **dequeue**
- **length**
- **iterator**

- **Standard** extends **QueueKernel**
- **Queue** implements **Iterable**
- **QueueSecondary** extends **Queue1L**
Instance Variables

• Each separate Queue1L object has its own distinct java.util.List variable that represents its object value.

• Note: In the code we will examine for Queue1L, there is a declaration of a private instance variable whose value is the java.util.List that represents one Queue1L object (namely, this).
Instance Variables

- Each separate Queue1L object has its own distinct java.util.List variable that represents its object value.
- Note: In the code we will examine for Queue1L, there is a declaration of a private instance variable whose value is the java.util.List that represents one Queue1L object (namely, this).

The adjective instance means there is a distinct variable (with the same name) for each instance (i.e., each distinct object) of the class in which it is declared.
Let’s Look at Queue1L.java
Other Representations?

- Is there any other way to use a `java.util.List` to represent an object value of type `Queue`?
Other Representations?

• Is there any other way to use a `java.util.List` to represent an object value of type `Queue`?

• Yes!
  – What if you simply thought of the front of the `Queue` as being at the right end of the `java.util.List` rather than the left?
  – How would the code change with this “reversed” interpretation?
Resources

• OSU CSE Components API: Queue
  – http://cse.osu.edu/software/common/doc/