Interactive Programming

CS160: User Interfaces

John Canny

Reminders

Midterm one week from Weds.

Review next Monday.

Previous semesters midterms will be posted today.

Due today

Low fidelity prototype

Pair programming II

Out

Interactive prototype (due 11/05)

Pair programming III (due 10/29)

This time

User Interface components and events

Model-View Controller (MVC) pattern

Multithreading for interactivity – need and risks

Some design patterns for multithreaded programs

User Interface Components

Each component is an object with

- Bounding box
- Paint method for drawing itself
  - Drawn in the component's coordinate system
- Callbacks to process input events
  - Mouse clicks, typed keys

Java:

```
public void paint(Graphics g)
{
   g.fillRect(…); // interior
   g.drawString(…); // label
   g.drawRect(…); // outline
}
```

User Interface Components

In Javascript/HTML5, you don’t normally see the internals of component rendering. But you can draw widgets using either HTML 5 model:

- **HTML 5 canvas** (procedural)
- **SVG** (declarative)

Of these two, SVG is likely to be a better choice. Since you define objects to be rendered in SVG, the browser runtime takes care of repainting, resizing, and “object” identity.

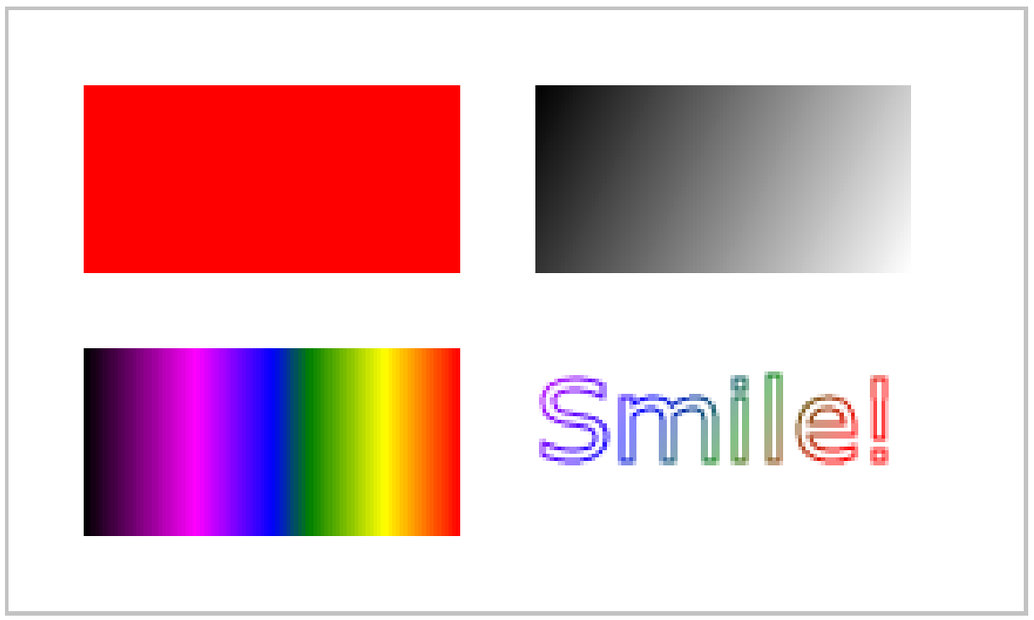
HTML 5

```
<script>
var c=document.getElementById("myCanvas");
{
    var ctx=c.getContext("2d");
    ctx.fillStyle="#FF0000";
    ctx.fillRect(20,20,100,50);

    var grd=ctx.createLinearGradient(140,20,240,70);
    grd.addColorStop(0,"black");
    grd.addColorStop(1,"white");
    ctx.fillStyle=grd;
    ctx.fillRect(140,20,100,50);

    var grd2=ctx.createLinearGradient(20,90,120,90);
    grd2.addColorStop(0,"black");
    grd2.addColorStop("0.3","magenta");
    grd2.addColorStop("0.5","blue");
    grd2.addColorStop("0.6","green");
    grd2.addColorStop("0.8","yellow");
    grd2.addColorStop(1,"red");
    ctx.fillStyle=grd2;
    ctx.fillRect(20,90,100,50);

    ctx.font="30px Verdana";
    var grd3=ctx.createLinearGradient(140,20,240,90)
    grd3.addColorStop(0,"black");
    grd3.addColorStop("0.3","magenta");
    grd3.addColorStop("0.6","blue");
    grd3.addColorStop("0.8","green");
    grd3.addColorStop(1,"red");
    ctx.strokeStyle=grd3;
    ctx.strokeText("Smile!",140,120);
}
</script>
```



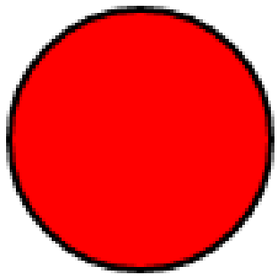
SVG

Inline XML that describes graphic elements

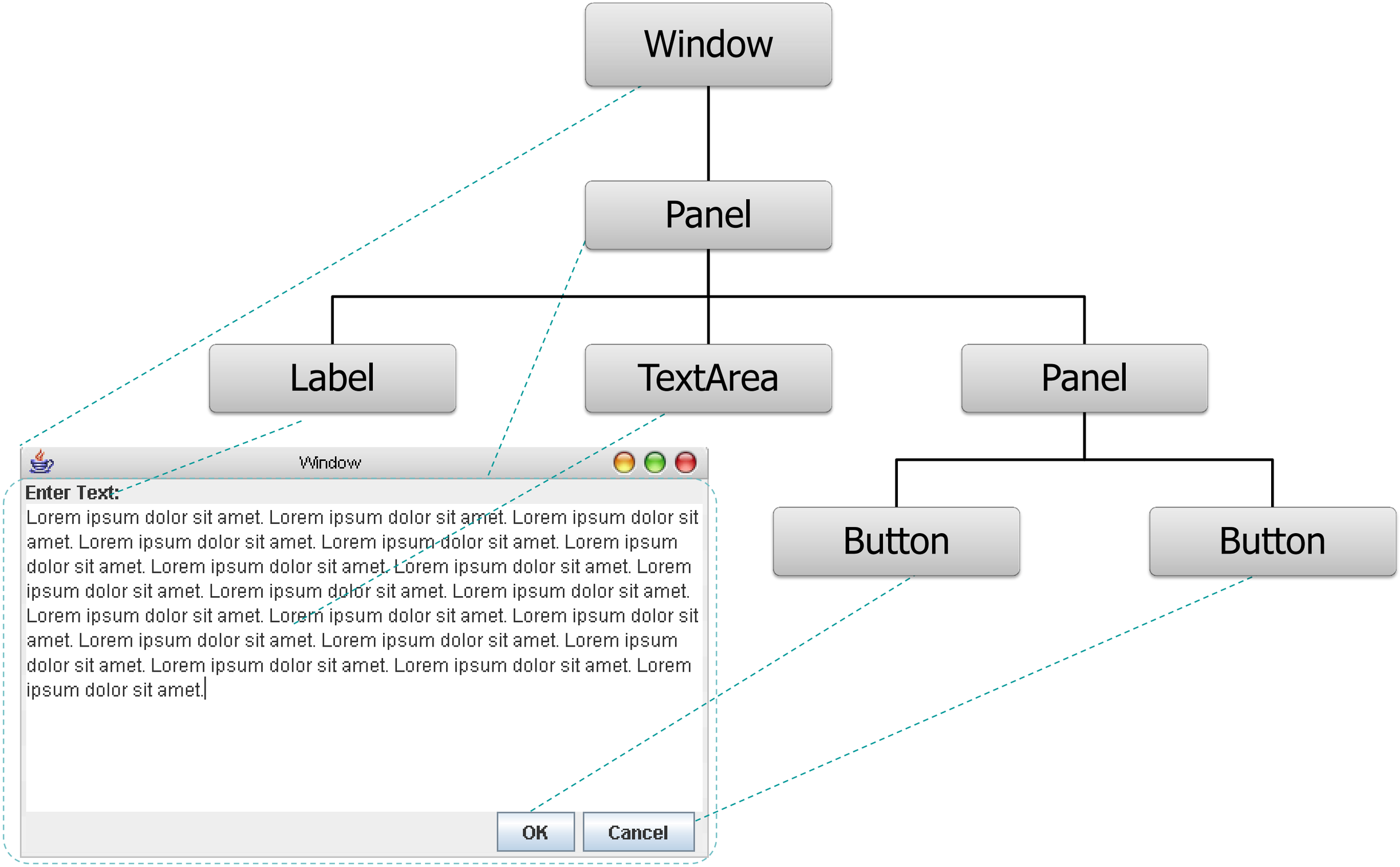
```
<html>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="100" cy="50" r="40" stroke="black"
  stroke-width="2" fill="red" />
</svg>

</body>
</html>
```



Layout: Containment Hierarchy

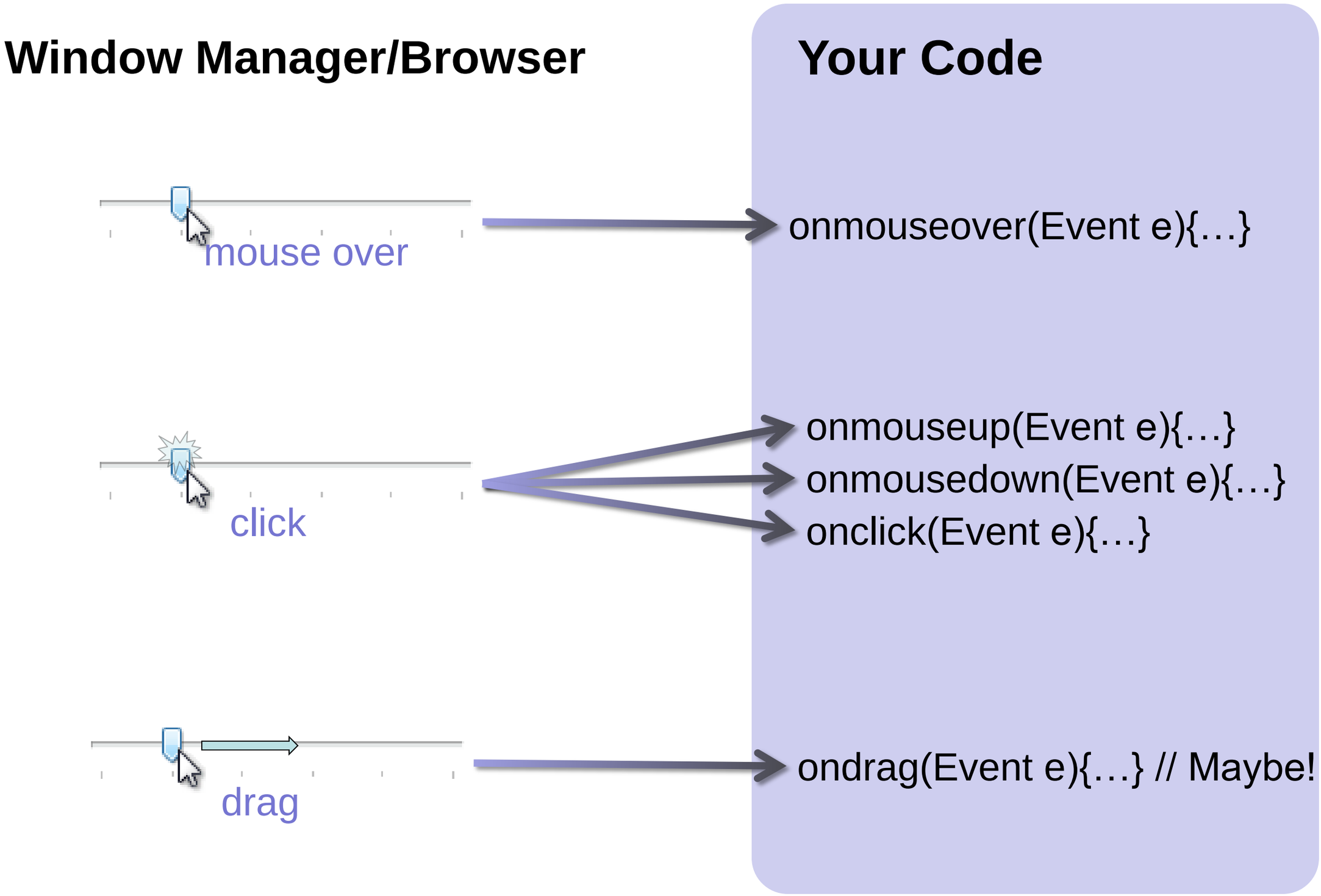


Anatomy of an Event

Encapsulates info needed for handlers to react to input:

- Event Type (mouse moved, key down, accelerometer etc)
- Event Source (the input component)
- Timestamp (when did event occur)
- Modifiers (Ctrl, Shift, Alt, etc)
- Event Content
  - Mouse: x,y coordinates, button pressed, # clicks
  - Keyboard: which key was pressed

Callbacks


Window Manager/Browser
Your Code
mouse over
onmouseover(Event e){…}
click
onmouseup(Event e){…}
onmousedown(Event e){…}
onclick(Event e){…}
drag
ondrag(Event e){…} // Maybe!

Event Registration

Can be either in the element description or in code:

```
<button onclick="copyText()" id="button1">Click Me</button>

mybutton = document.getElementById("button1");
mybutton.onclick(copyText());
```

Tells the runtime you want it to run the specified callback code when the event happens and the object is active.

Event Dispatch Loop

Mouse moved (t_0,x,y)

<u>Event Queue</u>

- Queue of input events

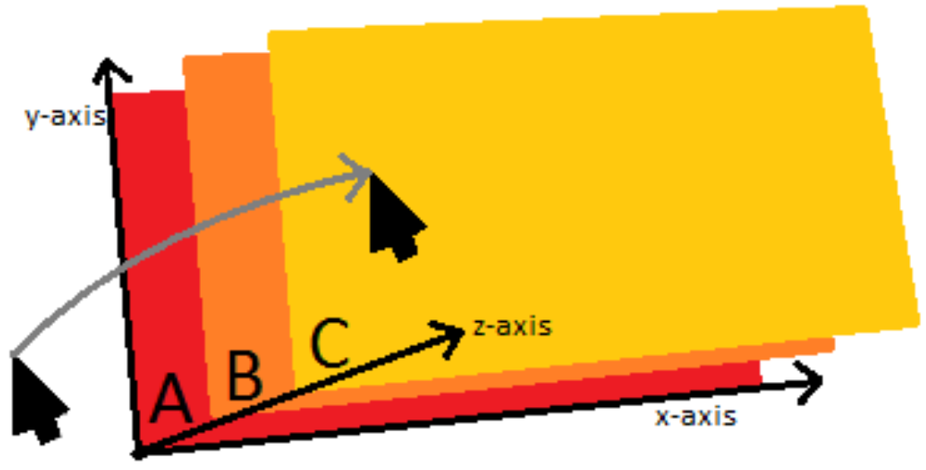
<u>Event Loop</u> (runs in dedicated thread)

- Remove next event from queue
- Determine event type
- Find proper component(s)
- Invoke callbacks on components
- Repeat, or wait until event arrives

<u>Component</u>

- Invoked callback method
- Update application state
- Request repaint, if needed

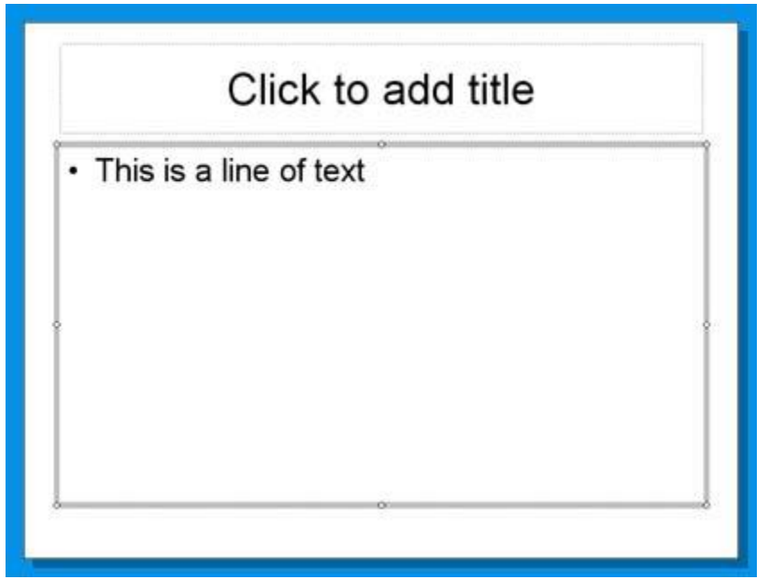
HTML 5 Events



Element stack, C a child of B, B a child of A. All dimensions are the same.

mousemove
(Pointing device is moved into element C; the topmost element in the stack)
mouseover (element *C*)
mouseenter (element *A*)
mouseenter (element *B*)
mouseenter (element *C*)
mousemove (multiple events in element *C*)
(Pointing device is moved out of element C...)
mouseout (element *C*)
mouseleave (element *C*)
mouseleave (element *B*)
mouseleave (element *A*)

HTML 5 Focus



- In addition, a particular element in the document has “focus” at any given time.
- This element receives keyboard events and may have special sensitivity to other events.

Outline

User Interface components and events

Model-View Controller (MVC) pattern

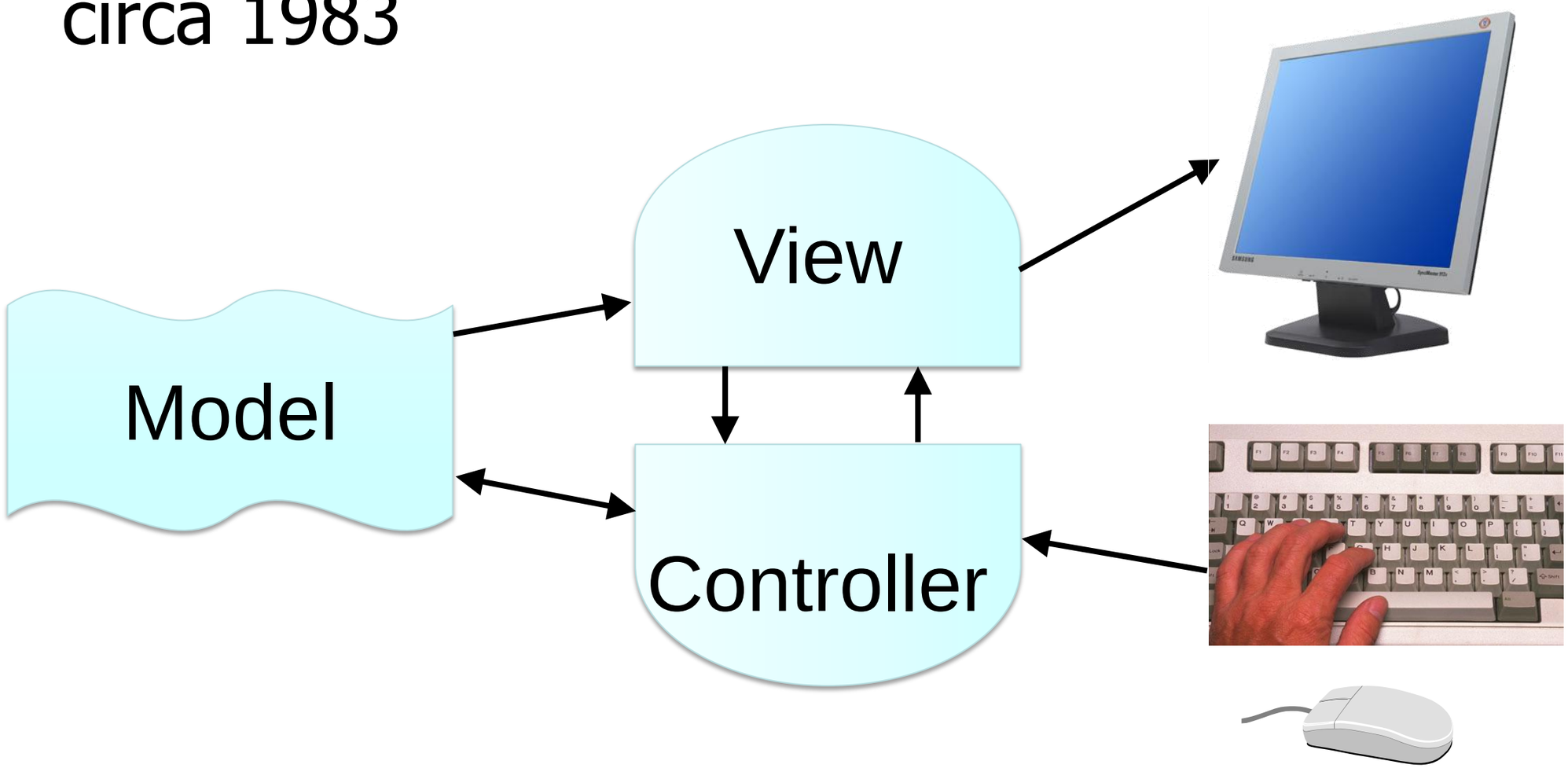
Multithreading for interactivity – need and risks

Some design patterns for multithreaded programs

Model-View-Controller

OO Architecture for interactive applications

- introduced by Smalltalk developers at PARC circa 1983

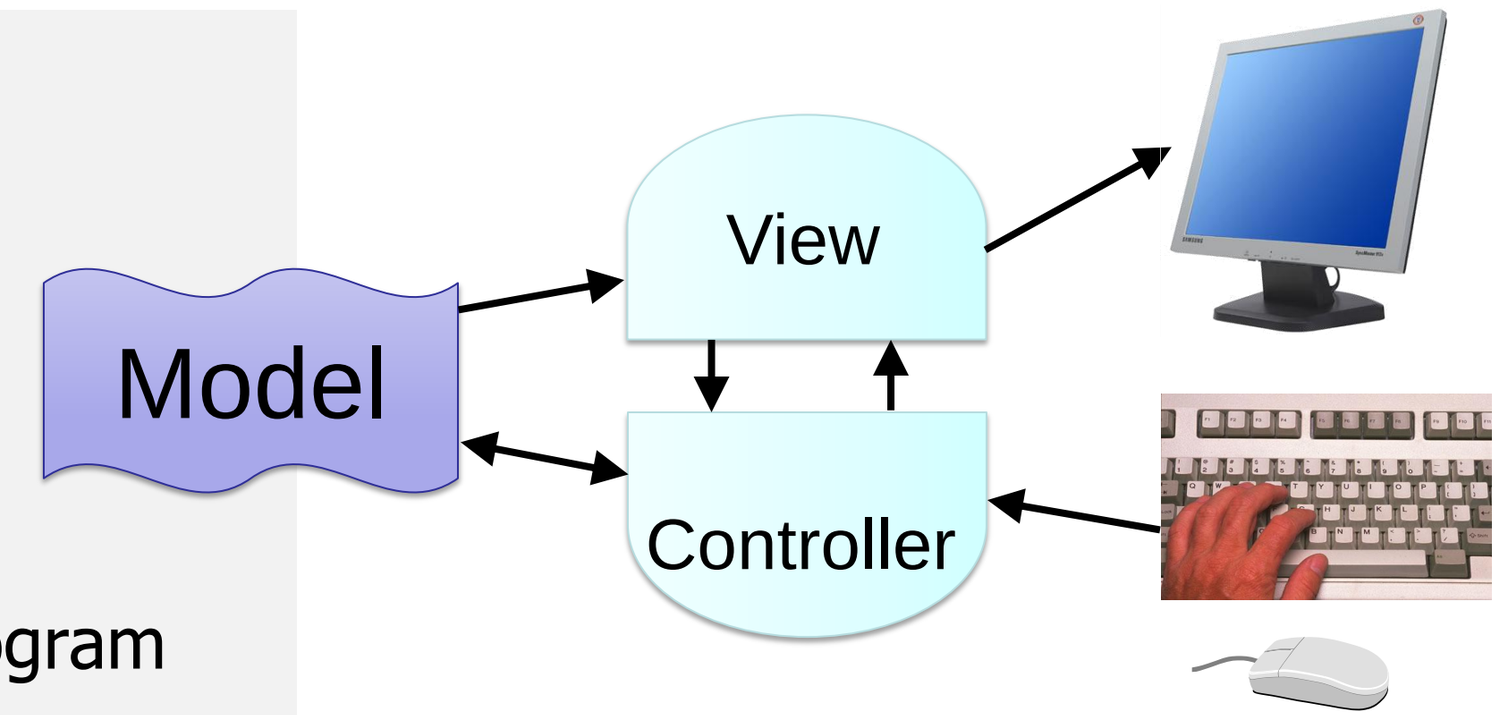


Model

Information the app is manipulating

Representation of real world objects

- circuit for a CAD program
  - logic gates and wires connecting them
- shapes in a drawing program
  - geometry and color
- DOM !

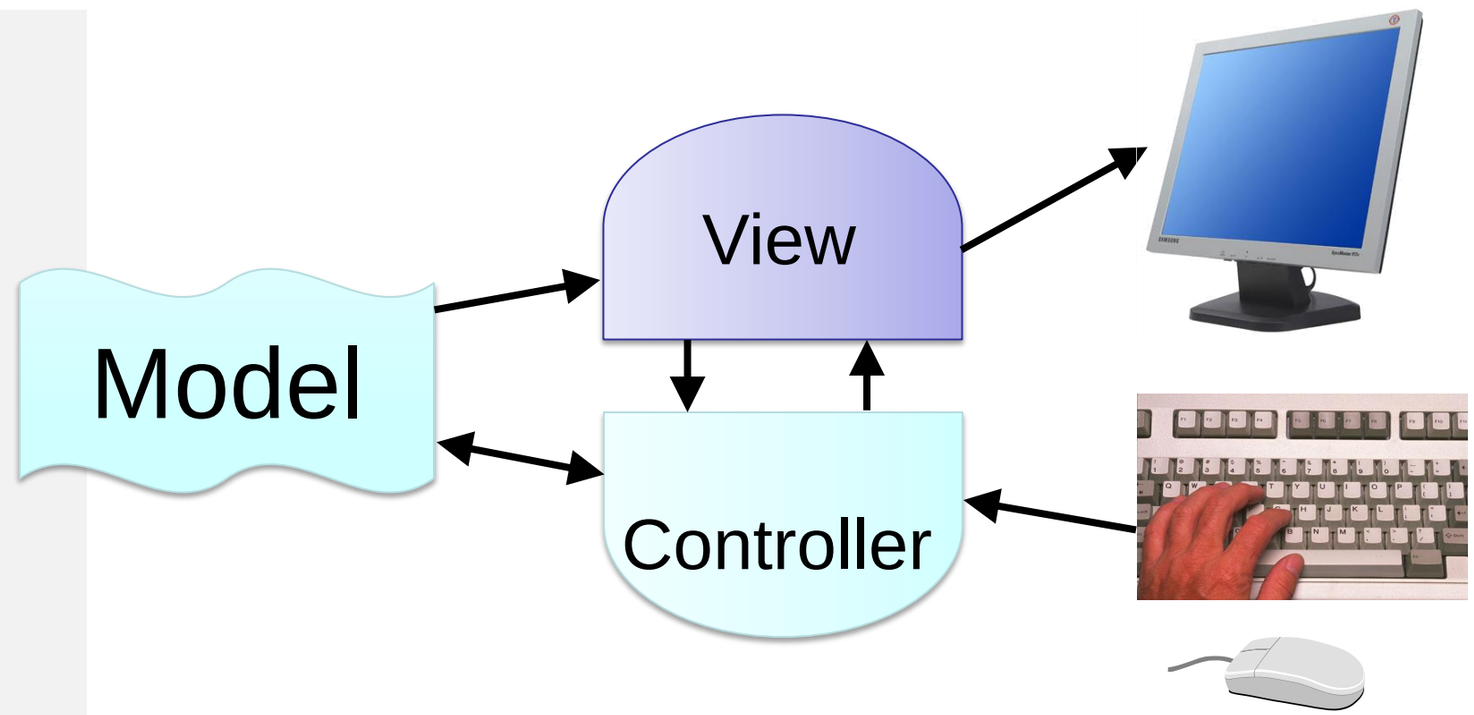


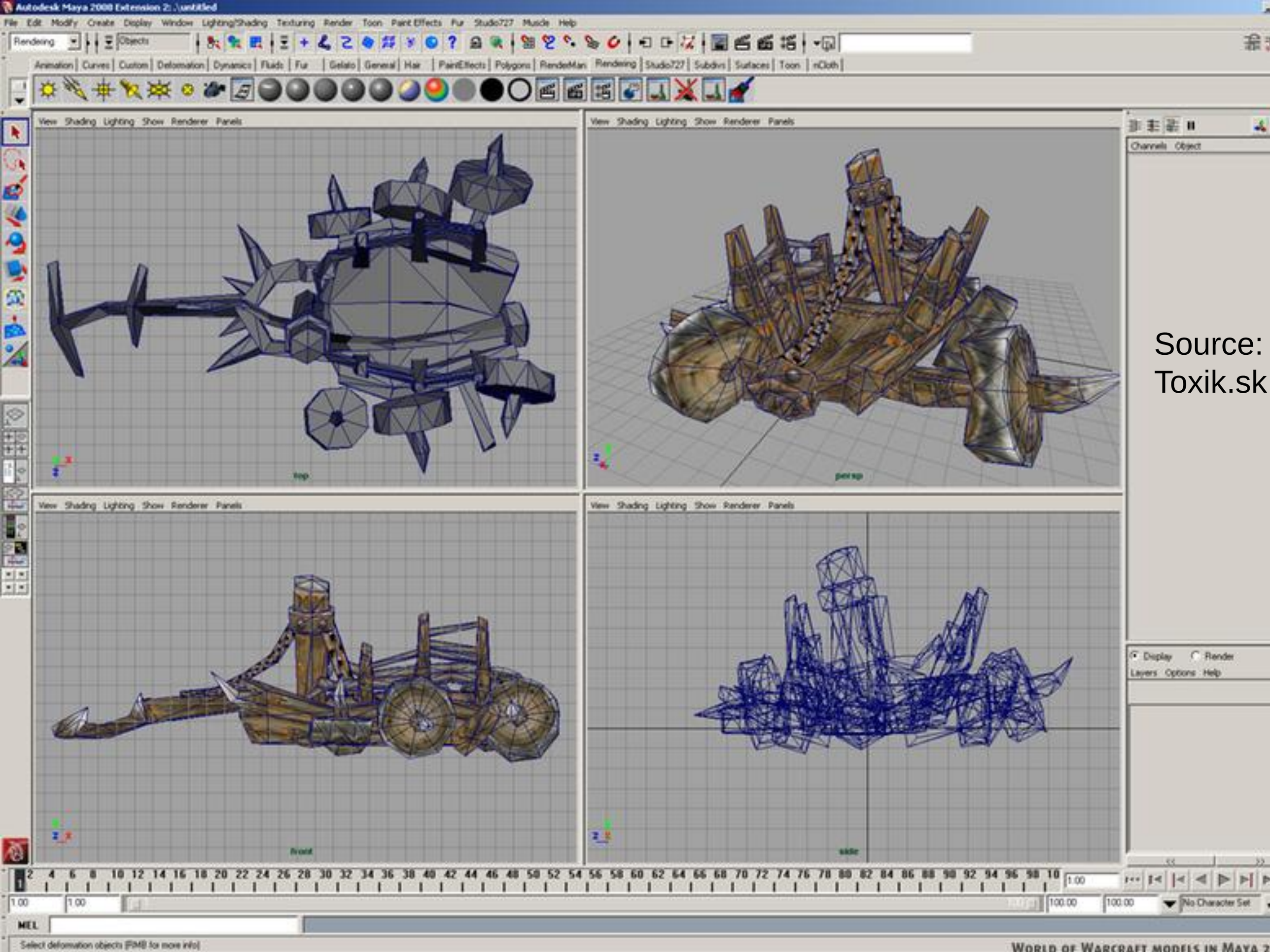
View

Implements a visual display of the model

May have multiple views

- e.g., shape view and numerical view
- Ex: CSS !

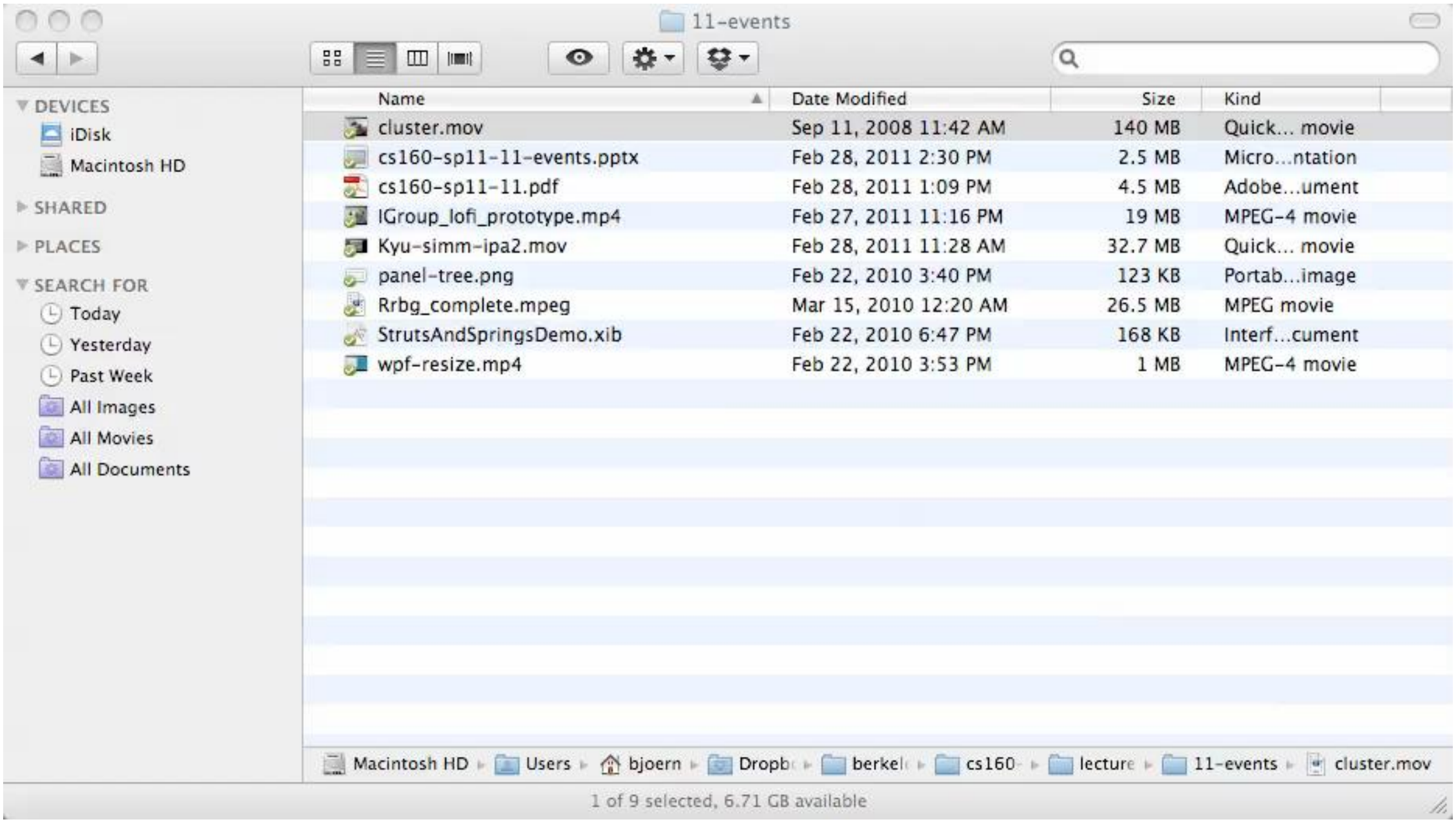




Source: Toxik.sk

World of Warcraft models in Maya 2

Multiple Views



View

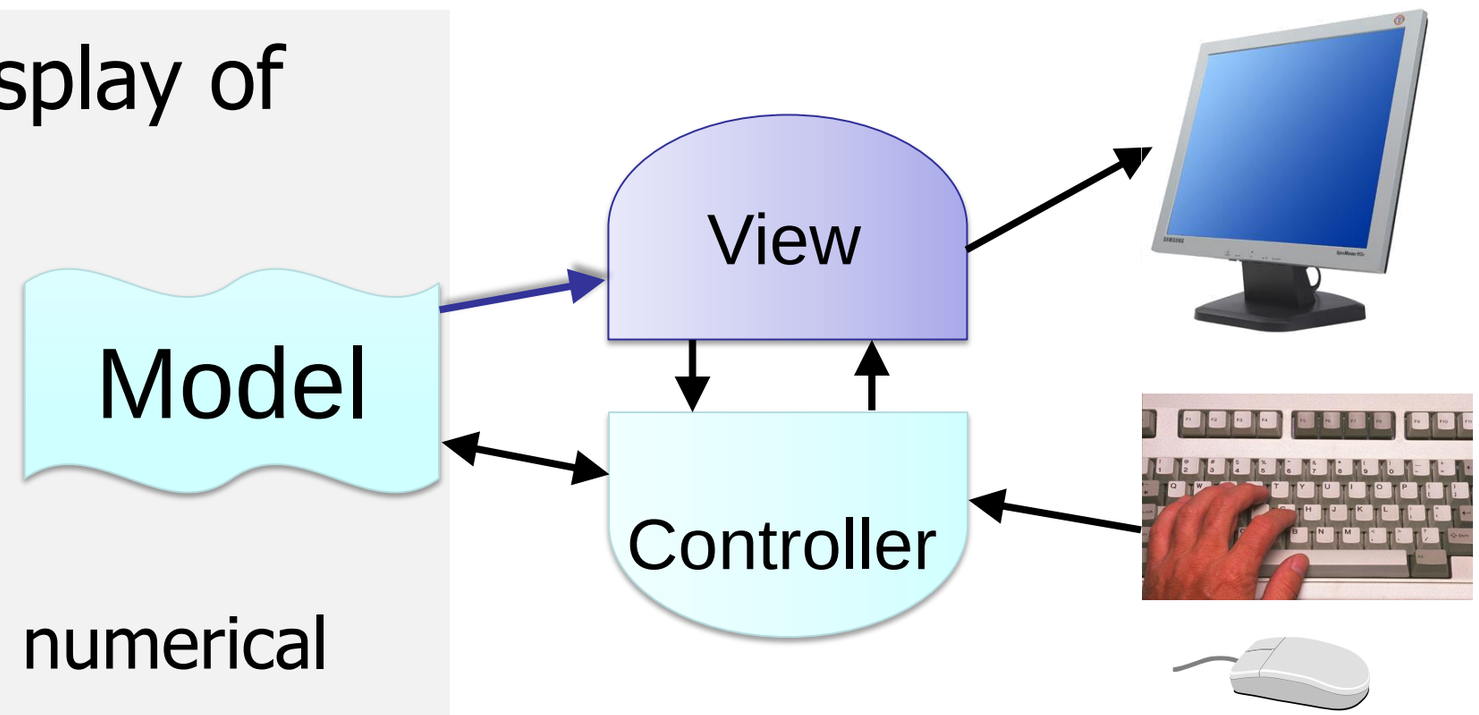
Implements a visual display of the model

May have multiple views

- e.g., shape view and numerical view

Any time model changes each view must be notified so it can update

- e.g., adding a new shape



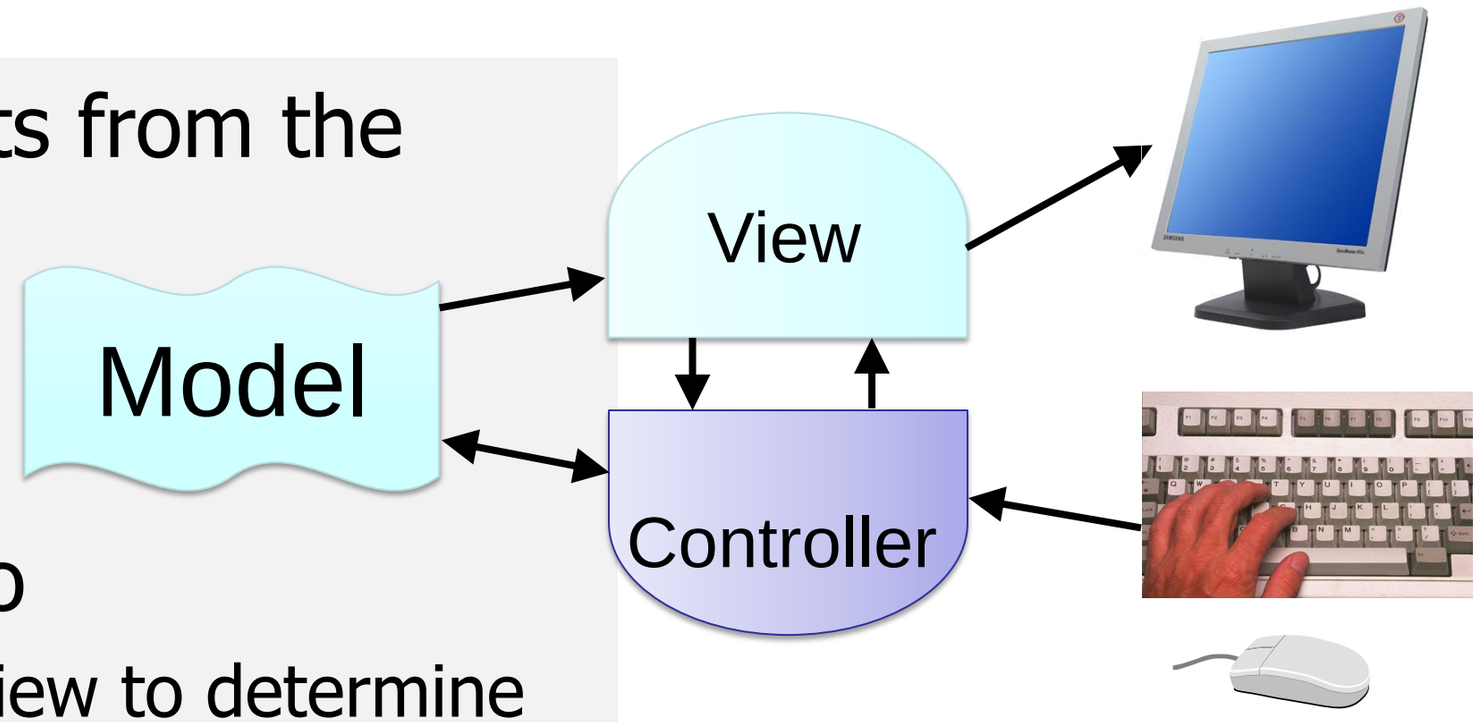
Single Controller Style

Receives all input events from the user

Decides what events mean and what to do

- communicates with view to determine the objects being manipulated (e.g., selection)

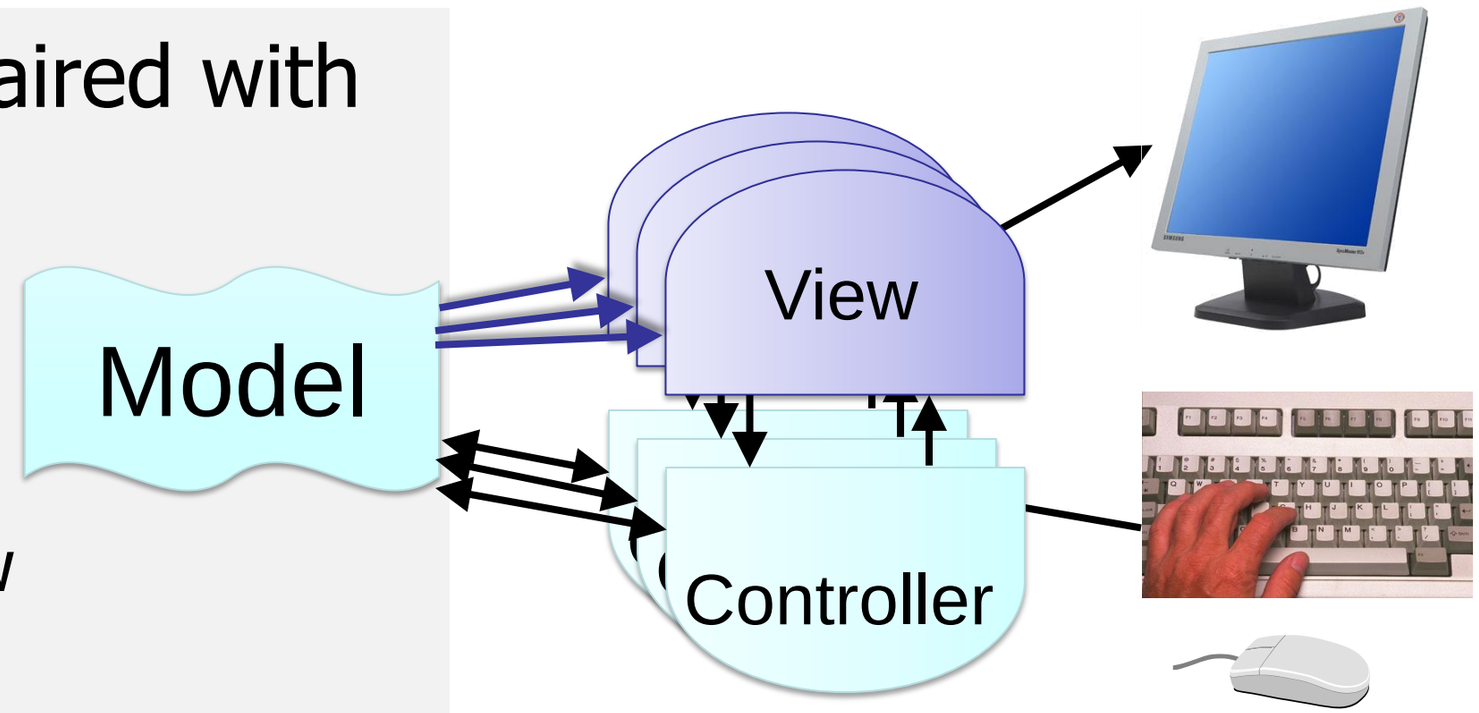
- calls model methods to make changes on objects
  - model makes change and notifies views to update



Multi-Controller Style

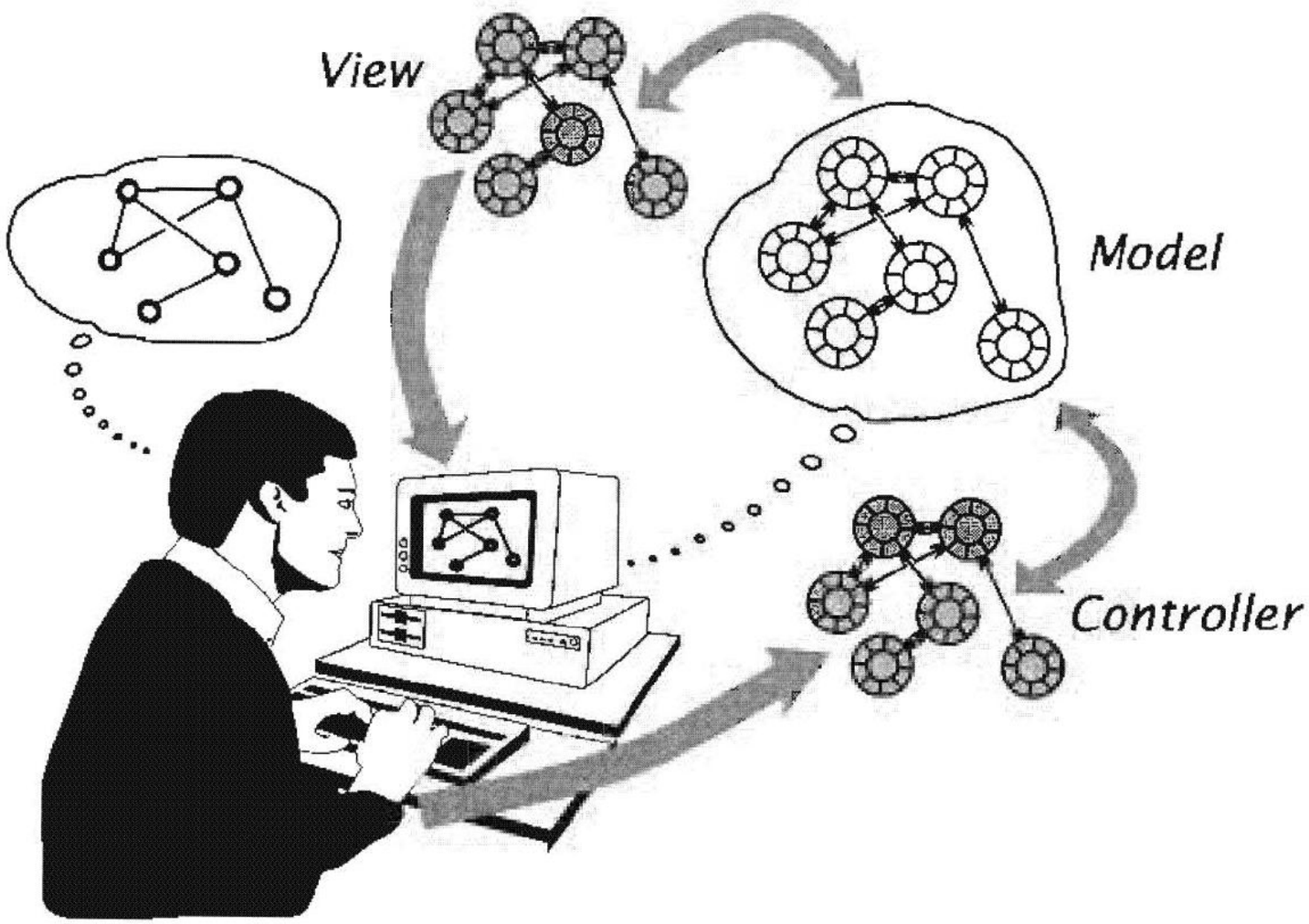
Controllers generally paired with Views

- Good modularity
- But note that events may not always land on the intended View



Why MVC?

Why MVC?



“The user's conceptual model of the system captures the semantics of objects, relationships, and behavior” (Collins)

Why MVC?

Combining MVC into one class will not scale

- Like HTML < 4, massive code changes to improve web page appearance even if content fixed.

Separation eases maintenance and extensibility

- Separates design work: Content creators don’t need to be designers, designers don’t need to be content experts.
- can change a view later, e.g., draw shapes in 3D
- flexibility of changing input handling when using separate controllers

Outline

User Interface components and events

Model-View Controller (MVC) pattern

Multithreading for interactivity – need and risks

Some design patterns for multithreaded programs

Why Multithreading?

Interactive programs need to respond **fast** to user input.

Direct manipulation assumes that objects onscreen move with the user's hand.

Why Multithreading?

But not all code returns from event-handling so fast:

- File access
- Network operations
- Database lookup
- Simulation

Why Multithreading?

We at least need to decouple the code processing screen events from the code that handles them.

But we often need to do more to make sure the code runs robustly, even in the presence of errors.

And for games?:

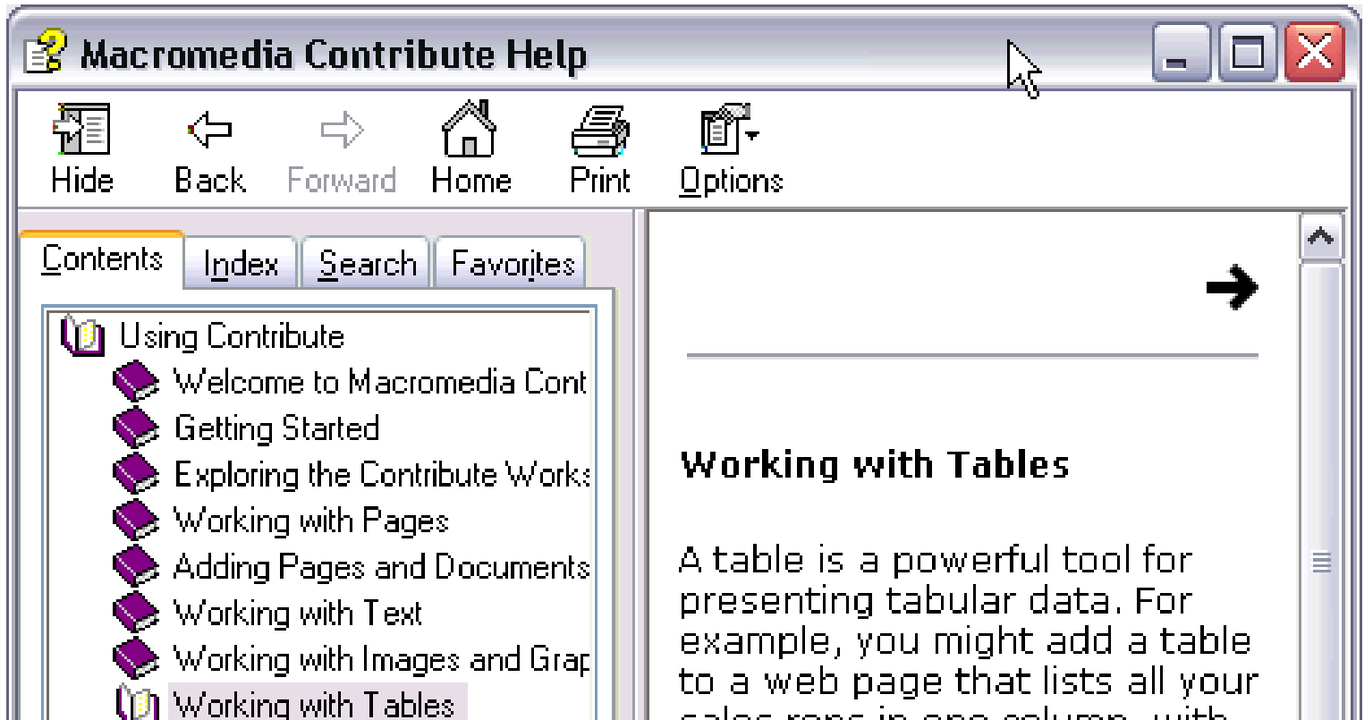
Why Multithreading?

Other processes that need to stay "live"

- Getting help
- Aborting (need to handle the abort operation)
- Doing something else while waiting for a long operation

Running these modules in a single thread gives users many aggravations:

Such as?



Why Multithreading?

More examples:

Multicore processors: use all the CPUs

Dynamic widgets:

- Clocks
- Progress indicators
- Mail inbox…

PS your course projects don’t **have** to be multi-threaded, they are interactive prototypes.

Multithreading in Web Interfaces

Doesn’t exist in general. Event handlers must *always* be fast.

Was available in Google Gears ^;

This makes it hard to build robust web applications. But there are some workarounds:

- Long operations (e.g. media playback) should be startable asynchronously and generate lots of events that reflect status: onload, onready, on paused, onstuck,…
- You can queue timer events for timeouts.

What is a Thread?

A **thread** is a **partial virtual machine**. Each thread has its own stack (and local variables), but shares its heap space and with other threads.

```
for (i=0; i<n; i++) {
    tmp = A[i];
    A[i] = B[i];
    B[i] = tmp;
}
```

VM1 **VM2**

What is a Process?

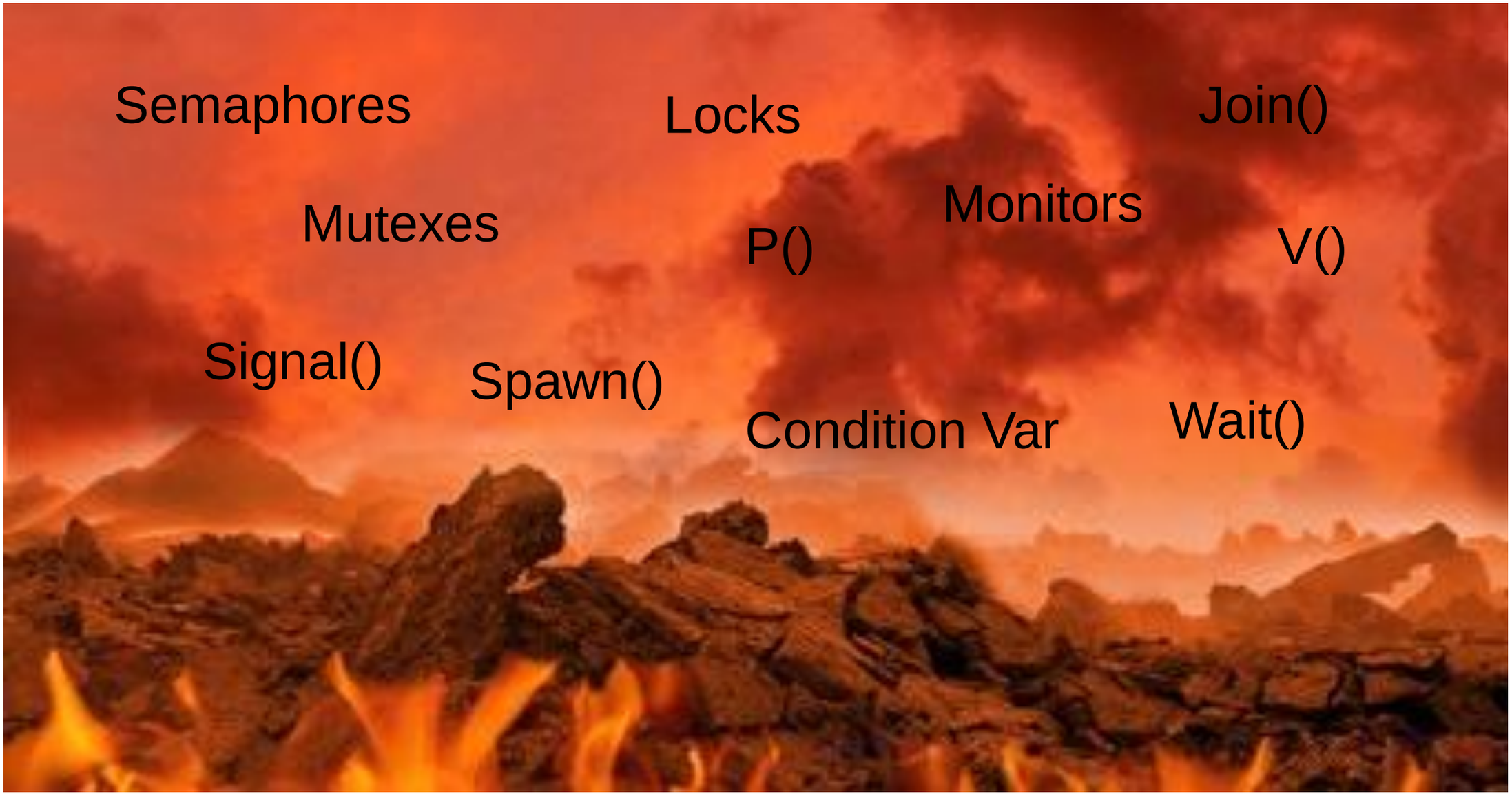
A **process** is a **complete virtual machine** with its own stack and heap.

Since processes don’t share memory with each other, they need other mechanisms (e.g. system message queues) to communicate with each other.

Processes vs. Threads

i.e. processes are like dating but multithreaded programs are like living together.

Threading Hell



After a long and careful analysis the results are clear: 11 out of 10 people can't handle threads.”
— Todd Hoff

Thread Coordination

Data primitives to allow one thread to process data without interference from others.

Include semaphores, mutexes, condition variables, monitors.

These are all low-level constructs and very error-prone.

Java, Objective-C and C# support a “synchronized” block wrapper.

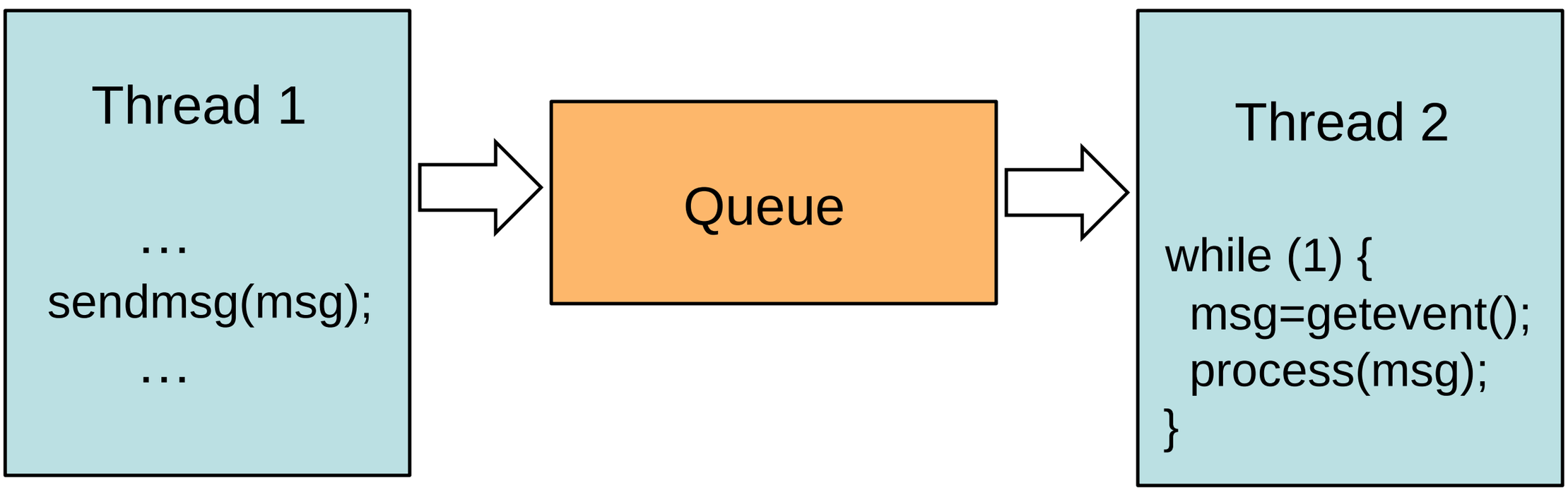
Concurrency Design Patterns

- Message queue
- GUI Thread / Worker thread pool
- Database / Model-View-Controller
- Actor

Message Queues

Two threads (or processes) with limited communication can use a message queue.

This is a simple implementation which minimizes coordination and data-sharing between the two threads.



Message Queues

Examples:

- The event queue in almost every GUI toolkit
- The “Handler” queue for threads in Java
- .NET MessageQueue objects
- Thread-specific message queues in the Windows GUI API
- Java Message Service (JMS)

And similar primitives exist between processes:

- POSIX message queues (in Linux)
- CORBA asynchronous messaging

Message Queues

We can realize a message queue by synchronizing queue and dequeue operations in a standard queue. i.e.

```
public Class myQueue<T> {
   Queue<T> Q;
   public synchronized void enqueue(T v) {
      Q.enqueue(v);
   }
   public synchronized T dequeue() {
      return Q.dequeue();
   }
```

Message Queues

Advantages:

- Code is basically sequential in each thread. Much easier to develop and debug.
- Reusable queue libraries do all the hard work.

Disadvantages:

- Inefficient if too much communication, or if complex data structures are passed.
- Need message loop/dispatcher on receiving end – not very modular.

Runnables

A weakness with a simple message queue approach is that behavior is very limited – the receiver only responds to messages it already knows what to do with.

A much more powerful mechanism is to post **code** (Runnables):

```
public class X implements Runnable {
   int y, z;
   public X(int y0, int z0) {y = y0; z = z0;} // Save y, z on create
   public void run() {
      // do something useful, using y, z at some later time
   }
}
```

Runnables

Runnables are class instances (Objects), and can be pushed into a queue like other messages.

When the message handler in the receiver dequeues a runnable, it recognizes it by type, and calls its run() method.

In this way, the runnable (which is created in an originating thread), gets executed in a different thread.

Design Patterns

- Message queue
- GUI Thread / Worker thread pool
- Database / Model-View-Controller
- Actor

GUI Thread + Worker ThreadPool

The GUI thread can only do one thing. A long operation (e.g. file read/write) has to run in another thread. We typically call those worker threads.

Creating/destroying threads is expensive, we don’t want to do it with each task. So we establish a **thread pool**, which is persistent and reusable.

Tasks (runnables) are assigned to threads by the pool service. You don’t normally need to know what is happening.

Example App

Simulates:

- File read and write
- Network connections
- A live help system

Demo

Design Patterns

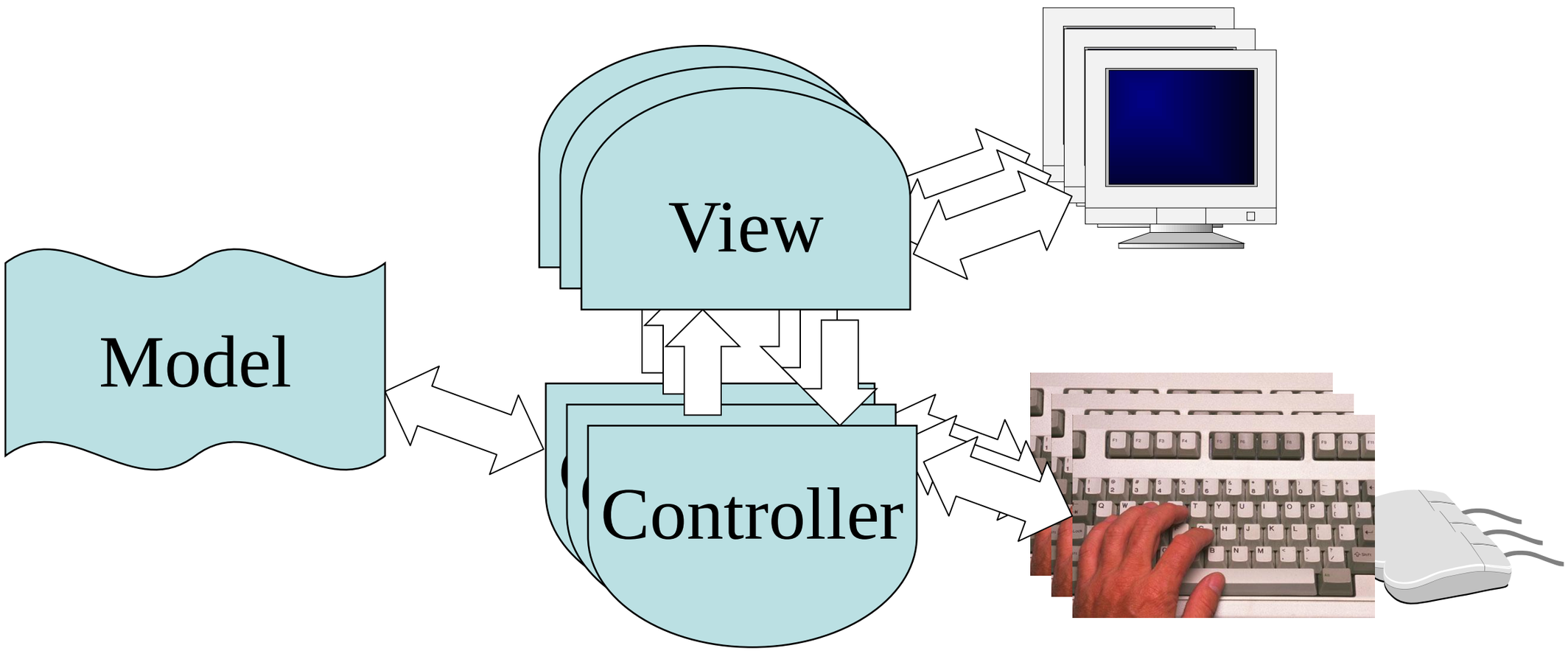
- Message queue
- GUI Thread / Worker thread pool
- Database / Model-View-Controller
- Actor

Model-View-Controller

MVC is an excellent pattern for concurrent programming:

State is centralized in the model, no other communication needed

Controllers+Viewers run independently, and each can have its own thread.



Model-View-Controller

Databases provide an excellent backend for the model:

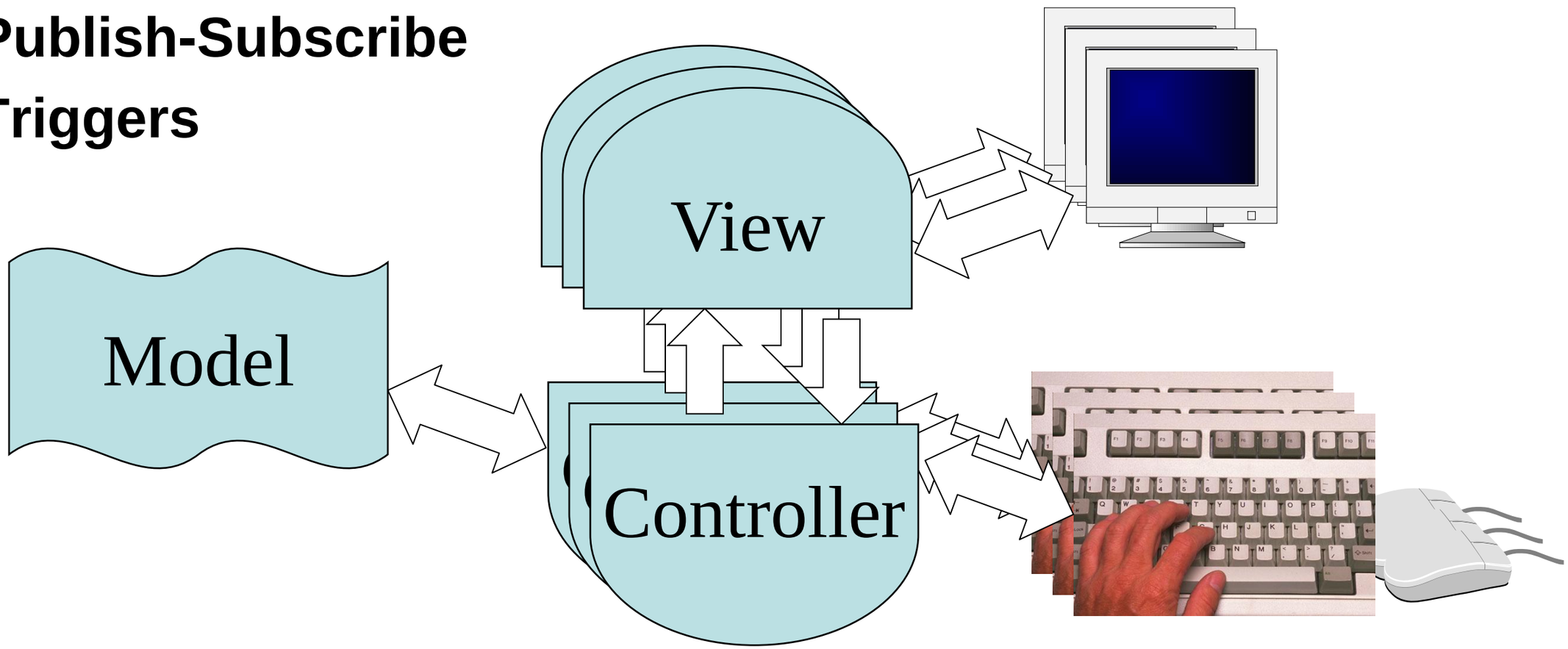
Transactions – complex updates are atomic.

Locking at different scales: an entire table or a row of a table.

Consistency constraints (relations).

Publish-Subscribe

Triggers



MVC for multithreading

Advantages:

- Extensible, modular.
- Easy to develop and debug.
- Save much coding if a database is used.

Disadvantages:

- Heavy use of resources (space, time, memory).
- Discourages quick information flows.
- Can be very slow with many users if locks are too coarse.

Example

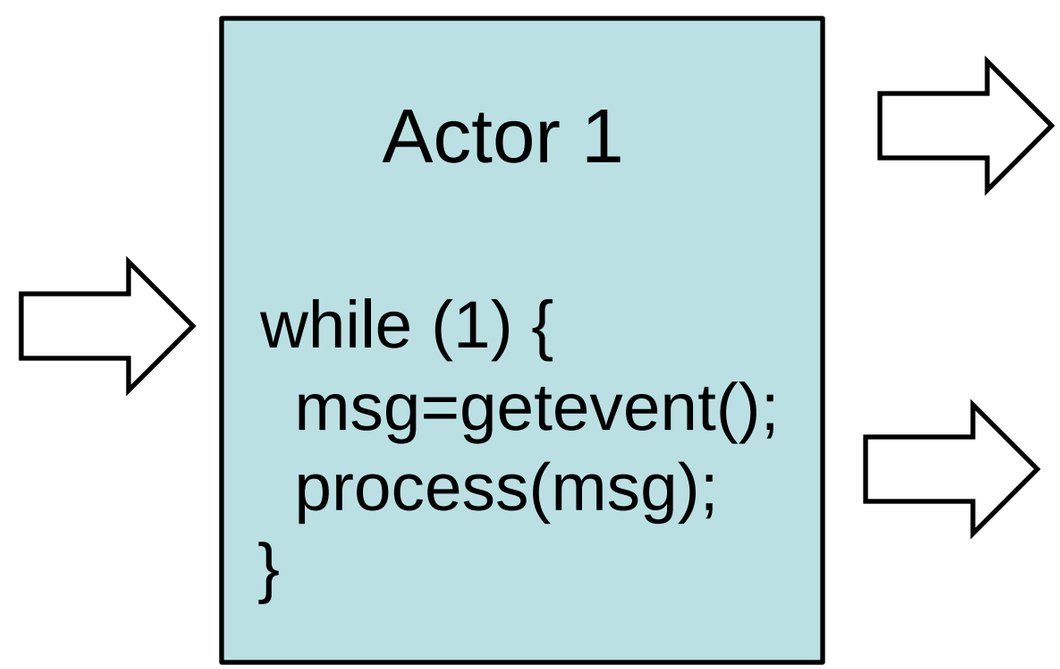
Actor

An actor is a class instance that runs its own thread.

Since data and methods are closely associated in a class, using a single thread to “run” the actor is very modular.

The actor will need an event loop to process incoming events.

Synchronized queues or mailboxes support communication.



Actor

Advantages:

Easy to design – like the sequential version of the class, but with the event loop added.

Good alignment between threads and data, minimizes contention and probability of inconsistency.

Exploits multicore processors.

Disadvantages:

A system of actors can be very complex to model.

Best to use a mixture of actors and “passive” classes.

A large multi-actor system is resource-intensive (memory, time,…)

Debugging

Not too difficult – similar to sequential debugging with a few extra operations:

Attach: attach the debugger to a running process.

List threads: list the running threads in the program.

Select a thread: Pick one to view or step through.

Thread-specific breakpoints: Stop the program when one specific thread reaches a program line.

Debugging

Note: You can also configure Eclipse to pause the VM when a breakpoint is hit in one thread.

From breakpoints view, select “suspend VM”.

Example

Review

Design patterns for multithreaded programs:

- Message queue
- GUI thread/Worker threadPool
- MVC
- Actor

Debugging multithreaded programs

Review

Design patterns for multithreaded programs:

- Message queue
- GUI thread/Worker threadPool
- MVC
- Actor