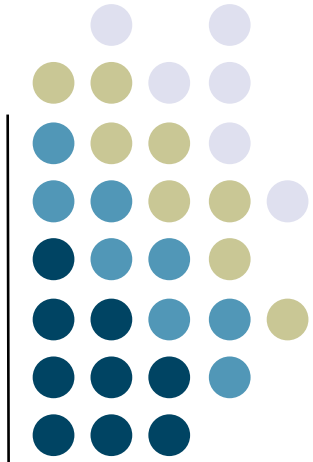
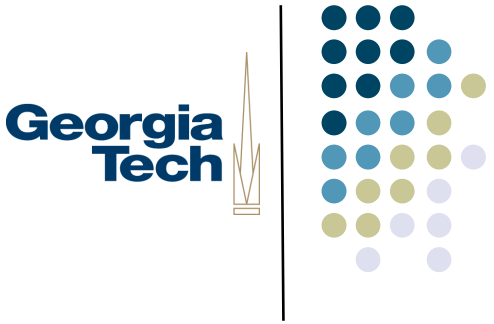
Input
(part 2: input models)



Georgia Tech

Dealing with diversity



- Saw lots of diversity in devices
  - actual details of devices (e.g., device drivers) is a real pain
  - how do we deal with the diversity?
- Need a model (abstraction) for input
  - like file systems abstract disks
  - higher level & device independent

Logical device approach

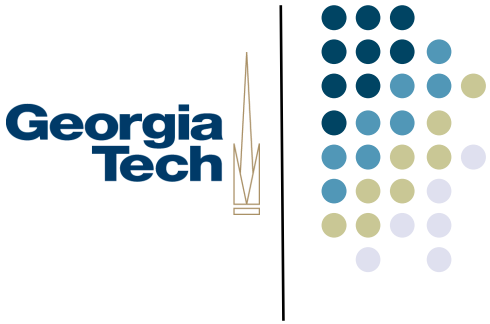
- One approach "logical devices"
  - A logical device is characterized by its software interface (only)
    - the set of values it returns
  - Rest of semantics (how it operates) fixed by category of device or left to the particular device

Logical device approach

- Fixed set of categories
  - old "Core Graphics" standard had 6
    - keyboard, locator, valuator, button
    - pick, stroke
- If actual device is missing, device is simulated in software
  - valuator => simulated slider
  - 3D locator => 3 knobs
- 1st step towards today's interactors

Logical device approach

- Abstraction provided by logical device model is good
- But… abstracts away too many details (some are important)
  - example: mouse vs. pen on palm pilot
    - Both are locators
    - What's the big difference?



Not a success but..

- Still useful to think in terms of "what information is returned"
- Categorization of devices useful
  - Two broad classes emerged
    - Event devices
    - Sampled devices

Categorization of devices

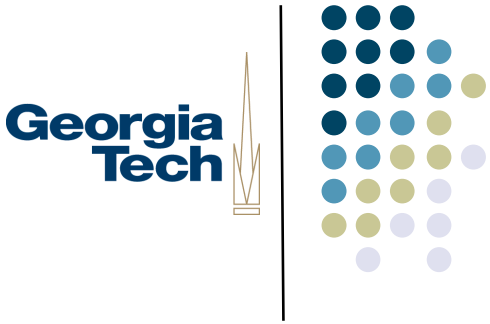
- Event devices
  - Time of input is determined by user
    - Best example: button
    - When activated, creates an "event record" (record of significant action)

Categorization of devices

- Sampled devices
  - Time of input is determined by the program
    - Best example: valuator or locator
    - Value is constantly updated
      - Might best think of as continuous
    - Program asks for current value when it needs it

A unified model

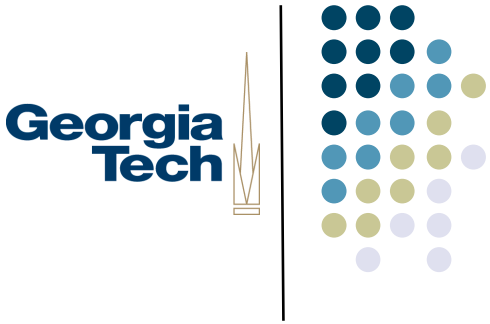
- Anybody see a way to do both major types of devices in one model?



A unified model: the event model

- Model everything as events
  - Sampled devices are handled with "incremental change" events
  - Each measurable change in value produces an event containing the new value
  - Program can keep track of the current value if it wants to sample

Simulating sampling under the event model of input

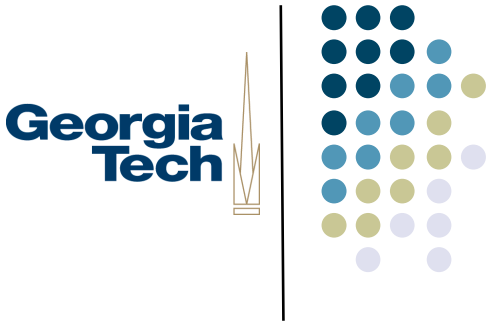


- Can cause problems
  - lots of little events
    - Can fall behind if doing a lot of computation/redraw for every event
      - machines are fast, blah blah blah
      - but can get behind (sampling provided built in throttling)

The event input model

- Almost all systems now use this

- An "event" is an indication that "something potentially significant" has just happened
  - in our case user action on input device
  - but, can be generalized



The event input model

- "Event records" are data structures (or objects) that record relevant facts about an event
  - generally just called "events"
- Event records often passed to an "event handler" routine
  - sometimes just encode relevant facts in parameters instead of event record

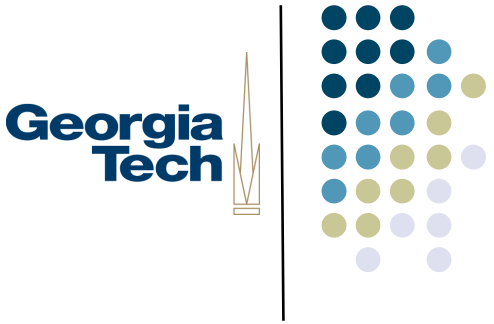
- Terminology redux: Swing calls these event handlers *listeners*; in other systems they are *callbacks*

Relevant facts

- What do we need to know about each event?

Relevant facts

- What
- Where
- When
- Value
- Additional Context

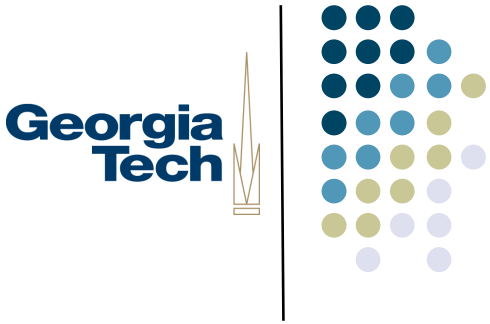


What

- What (exactly) caused the event
  - e.g., left mouse button went down
  - for "method based" systems this may be implicit in what handler gets called

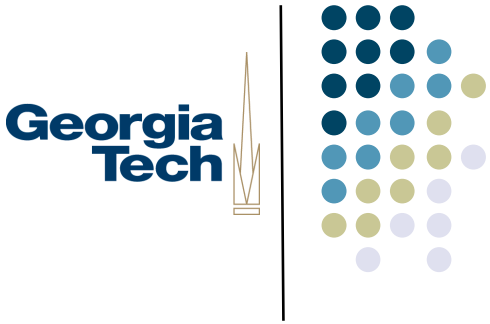
Where

- Where was the primary locator (mouse) when event happened
  - x,y position
  - also, inside what window, object, etc.
  - this is specific to GUIs, but it;s critical
    - e.g., can't tell what mouse button down means without this



When

- When did the event occur
  - Typically are dealing with events from the (hopefully recent) past
    - queued until program can get to them
  - In absolute time or relative to some start point
  - Hopefully at resolution of 10s of ms
    - important for e.g., double-clicks



Value

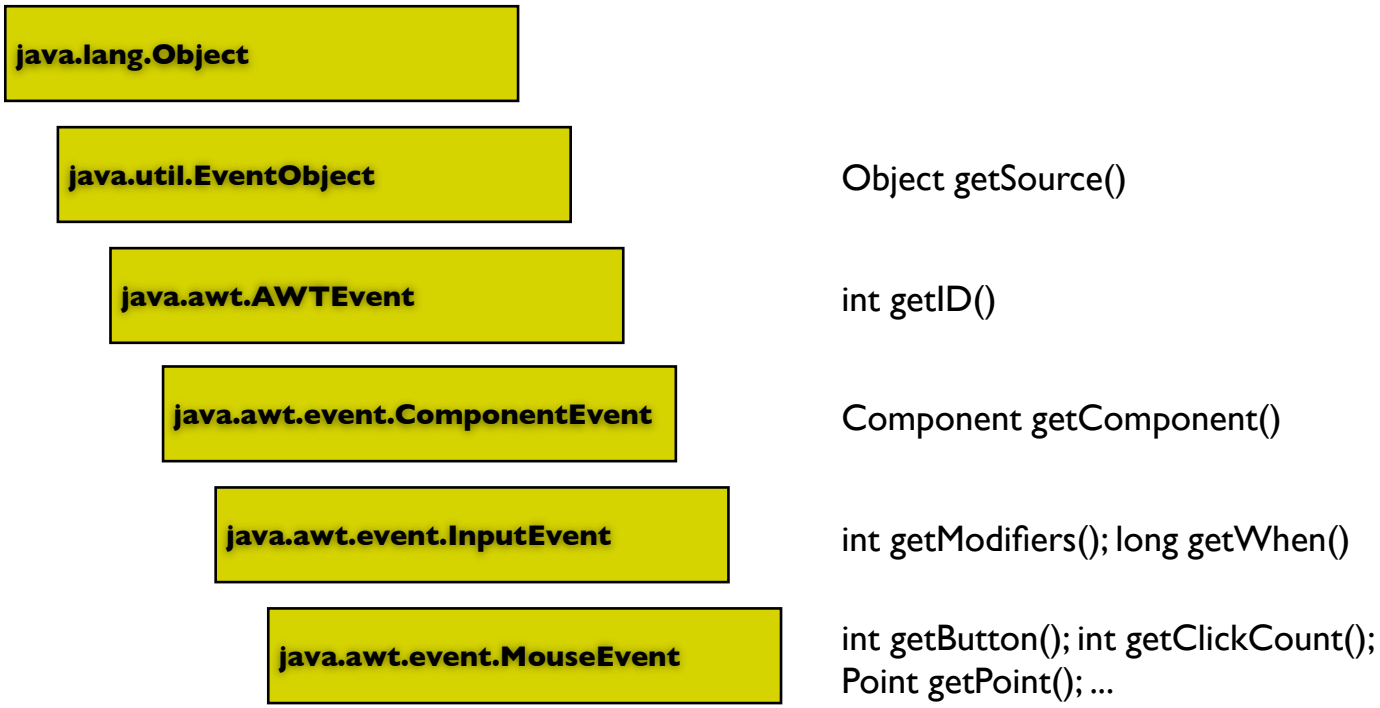
- Input value
  - e.g., ASCII value of key press
  - e.g., value of valuator
  - some inputs don't have a value
    - e.g. button press

Additional context

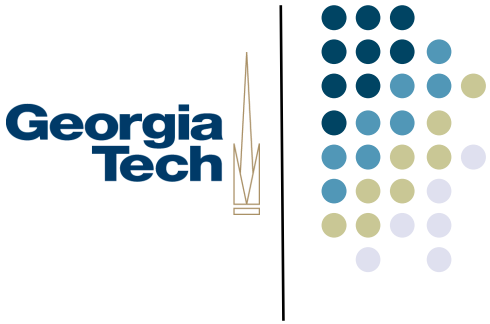
- Status of important buttons
  - shift, control, and other modifiers
  - possibly the mouse buttons

Example: Swing events

- Reuses and borrows heavily from AWT (it has to)
  - A pretty generic / typical event model
- Lots (and lots) of hierarchy. Example:



Common methods in Swing events

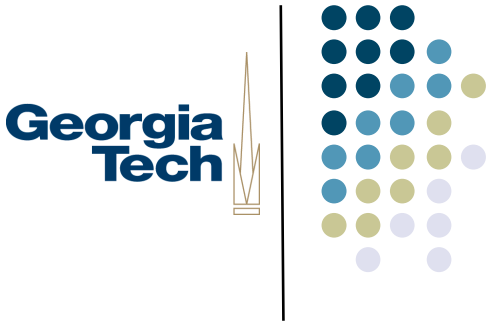


- What
  - getID() -- code for kind of event
  - getClickCount() -- mouse only, indicates double-click, etc.
- Where
  - getSource(), getComponent() -- component that event is "in"
  - getX(), getY(), getPoint() -- location relative to that component
- When
  - getWhen() -- timestamp in milliseconds
- Value
  - getKeyChar(), getKeyCode() -- get information about keypresses (for example)
  - getModifiers() -- were shift, ctrl, meta, ... held down?

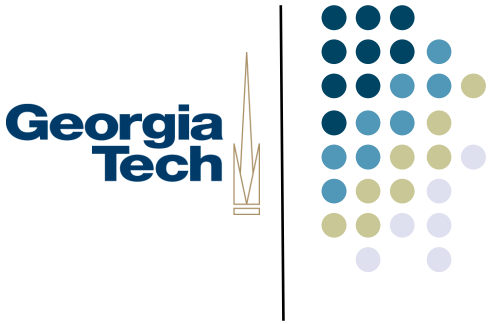
Extending the event model

- Events can extend past simple user inputs
  - Extra processing of raw events to get "higher level" events
    - window / object enter & exit
    - list selection
    - rearrangement of the interactor hierarchy
  - Can extend to other "things of significance"
    - arrival of network traffic

Extending the event model



- Window systems typically introduce a number of events
  - window enter/exit region enter/exit
    - system tracks mouse internally so code acts only at significant points
  - Redraw / damage events
  - Resize & window move events



Synchronization and events

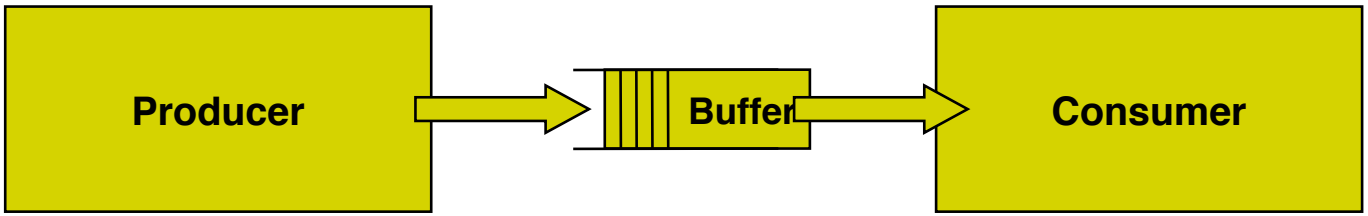
- The user and the system inherently operate in parallel
  - asynchronously
    - Means different programming model for applications (asynchronous callbacks)
    - Means special work for toolkit/window system implementations
- This is a producer consumer problem
  - user produces events
  - system consumes them

Synchronization and events

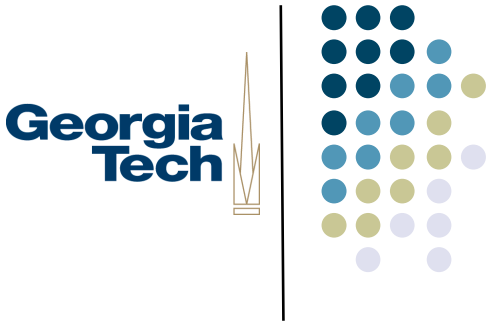
- Need to deal with asynchrony
  - both parties need to operate when they can
  - but can't apply concurrency control techniques to people
- How do we handle this?

Synchronization and events

- Use a queue (buffer) between



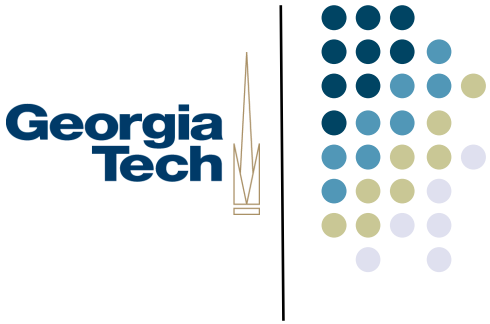
  - As long as buffer doesn't overflow, producer does not need to block
  - Consumer operates on events when it can



Implications of queued events

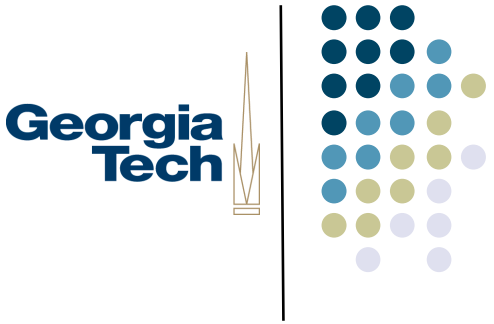
- We are really operating on events from the past
  - hopefully the recent past
- But sampled input is from the present
  - mixing them can cause problems
  - e.g. inaccurate position at end of drag

Using events from an event queue

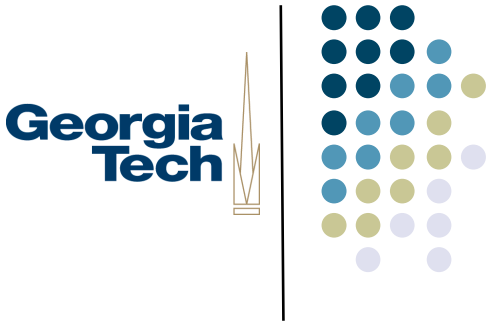


- Basic paradigm of event driven program can be summed up with one prototypical control flow
  - Will see several variations, but all on the same theme

Using events from an event queue




```
Main_event_loop()
   init();
   set_input_interests();
   repeat
      evt = wait_for_event();
      case evt of
         … dispatch evt -- send to some object
      end case;
      redraw_screen();
   until done;
```

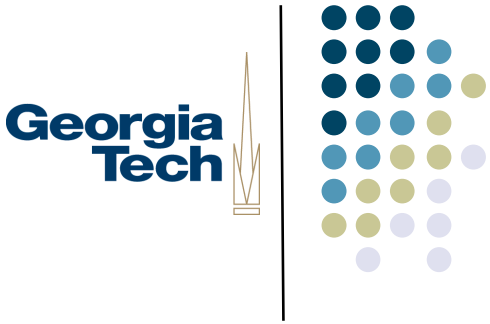


Using events from an event queue

- Very stylized code
  - in fact, generally you don't even get to write it
  - often only provide system with routines/methods to call for "dispatch"

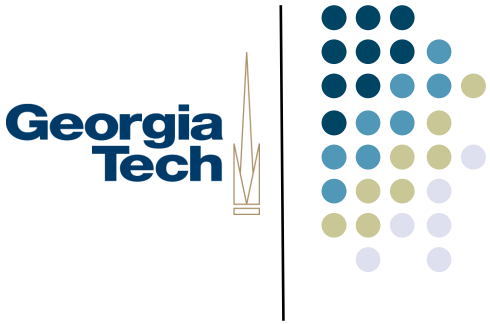
```
repeat
   evt = wait_for_event();
   user_object.handle_event(evt);
   redraw_screen();
until done;
```

Using events from an event queue



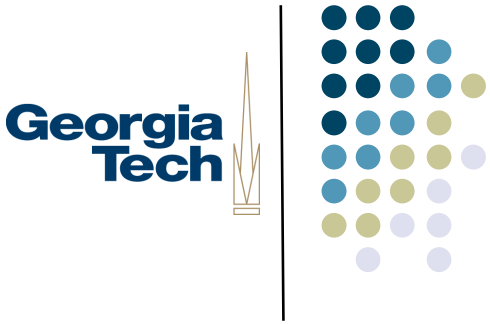
- Two big questions:
  - What object(s) gets the event?
  - What does it do with it?
    - Interpret it based on what the event is, what the object is, and what state the object is in

Dispatch strategies: what object gets the event



- Simple approach
  - lowest object in interactor tree that overlaps the position in event gets it
    - if that object doesn't want it, try its parent, etc.
  - "Bottom first" dispatch strategy

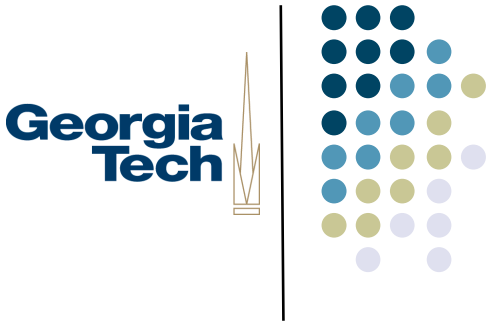
Dispatch strategies: what object gets the event



- Can also do "top-first"
  - root gets it
  - has chance to act on it, or modify it
  - then gives to overlapping child
  - has another chance to act on it if child (and its children) doesn't take it

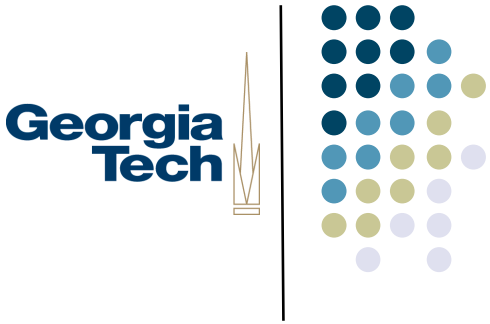
➔ more flexible (get top-first & bottom-first)

But… a problem with fixed dispatch strategies like this



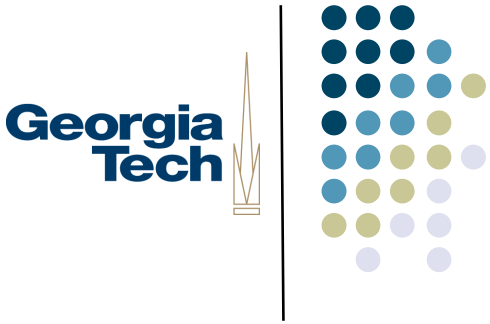
- Does this work for everything?

But… a problem with fixed dispatch strategies like this



- Does this work for everything?
  - What about key strokes?
  - Should these be dispatched based on cursor location?
    - Probably not
    - Probably want them to go to "current text focus"

Two major ways to dispatch events



- Positional dispatch
  - Event goes to an object based on position of the event
- Focus-based dispatch
  - Event goes to a designated object (the current focus) no matter where the mouse is pointing

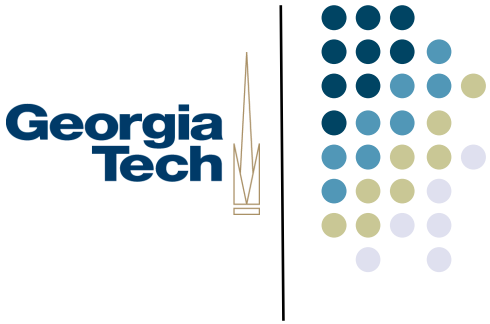
Question

- Would mouse events be done by focus or positional dispatch?

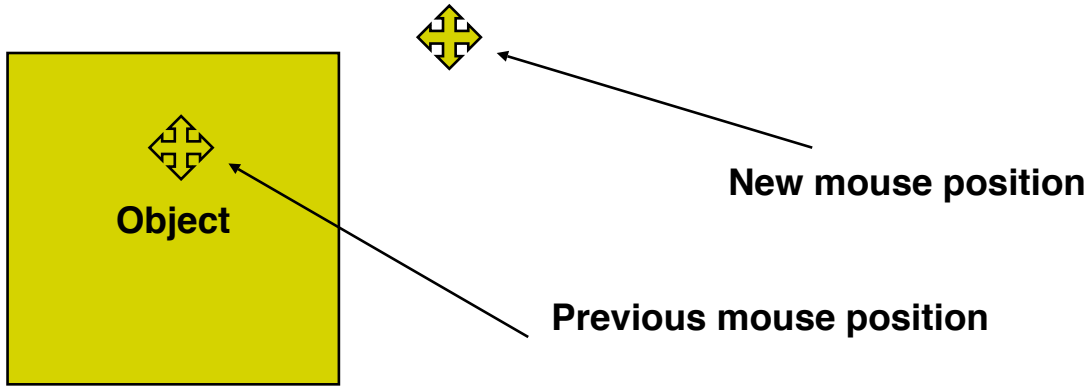
Question & answer

- Would mouse events be done by focus or positional dispatch?
- It depends…
  - painting: use positional
  - dragging an object: need focus (why?)

Dragging an object needs focus dispatch

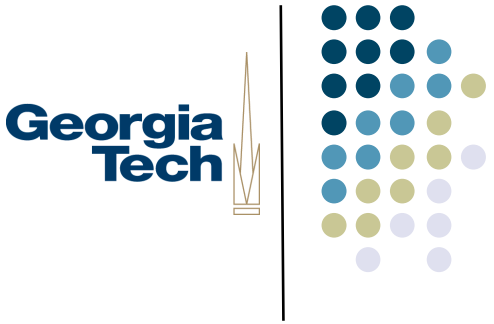


- Why? What if we have a big jump?

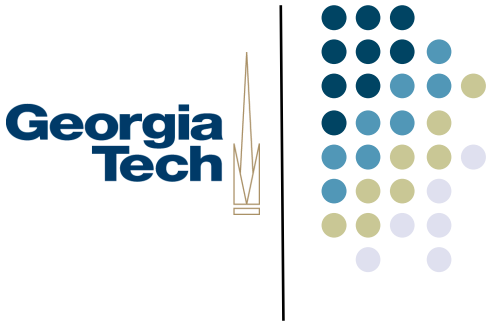


- Cursor now outside the object and it doesn't get the next event!

Positional and focus based dispatch



- Will need both
- Will need a flexible way to decide which one is right
  - will see this again later, for now just remember that sometimes we need one, sometimes another



Positional dispatch

- If we are dispatching positionally, need a way to tell what object(s) are "under" a location
- "Picking"

Picking

- Probably don't want to pick on the basis of a point (single pixel)
  - Why?

Picking

- Probably don't want to pick on the basis of a point (single pixel)
  - Why?
  - Because it requires a lot of accuracy
- Instead may want to pick anything within a small region around the cursor

Implementing pick

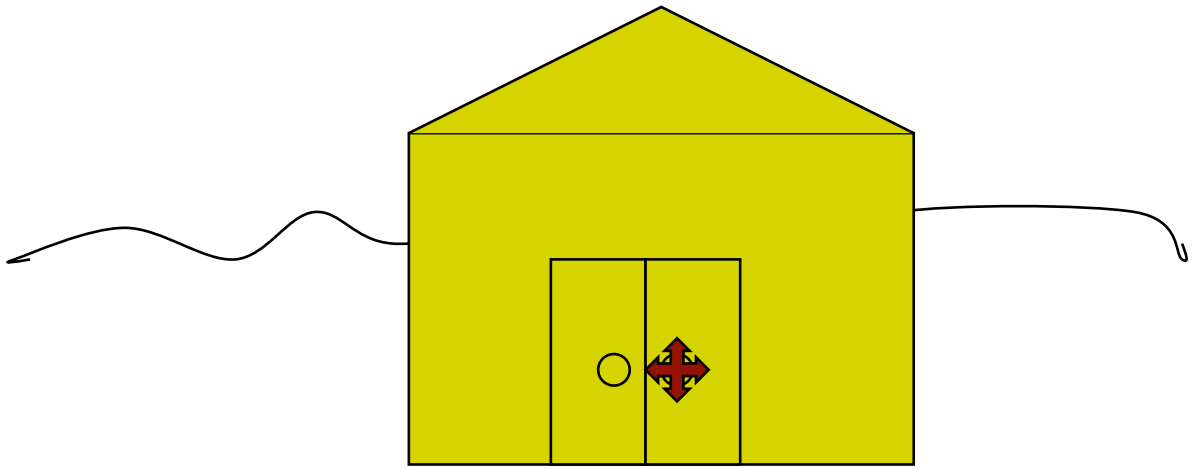
- Possible to apply a clipping algorithm
  - small clip region around cursor
  - pick anything that is not completely clipped away

Implementing pick

- Better is a recursive "pick traversal"
  - Walk down the object tree
  - Each object does a local test customized to its shape, state (enabled or not), and semantics
  - Also tests its children recursively

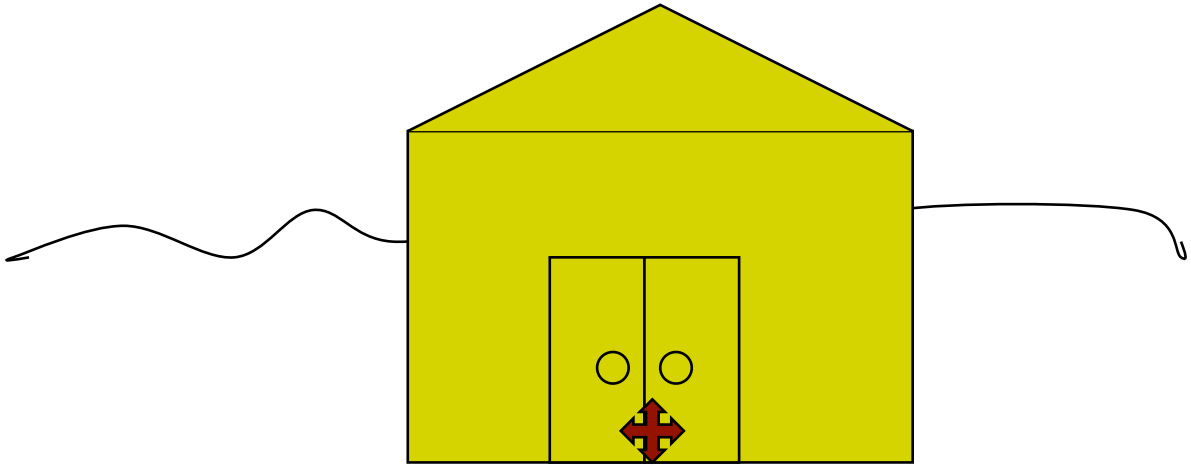
Pick ambiguity

- Classic problem, what if multiple things picked?
  - Two types
  - Hierarchical ambiguity
    - are we picking the door knob, the door, the house, or the neighborhood?



Pick ambiguity

- Spatial ambiguity
  - Which door are we picking?

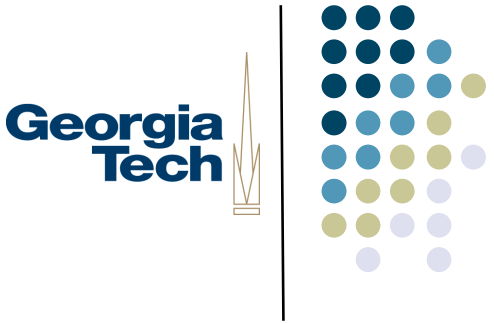


Solutions for pick ambiguity

- No "silver bullet", but two possible solutions
  - "Strong typing" (use dialog state)
    - Not all kinds of objects make sense to pick at a given time
      - Turn off "pickability" for unacceptable objects
        - reject pick during traversal

Solutions for pick ambiguity

- Get the user involved
  - direct choice
    - typically slow and tedious
  - pick one, but let the user reject it and/or easily back out of it
    - often better
    - feedback is critical


Georgia
Tech