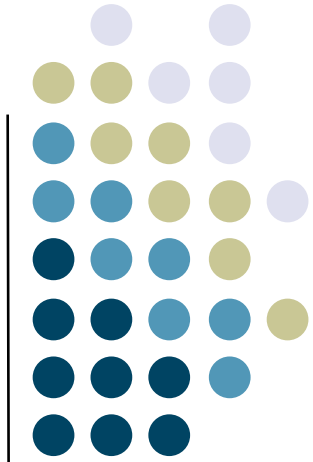
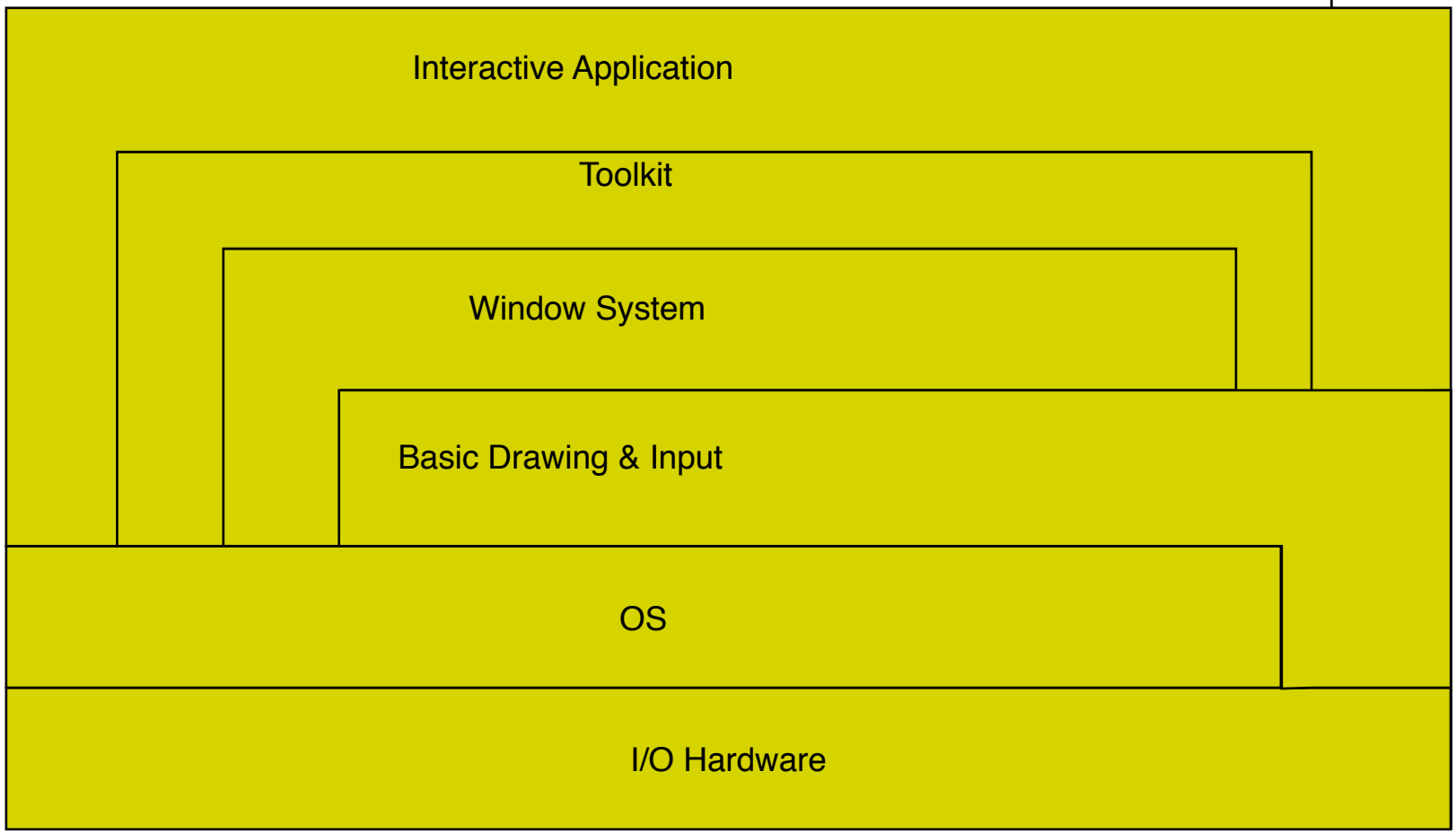
Output in Window Systems and Toolkits

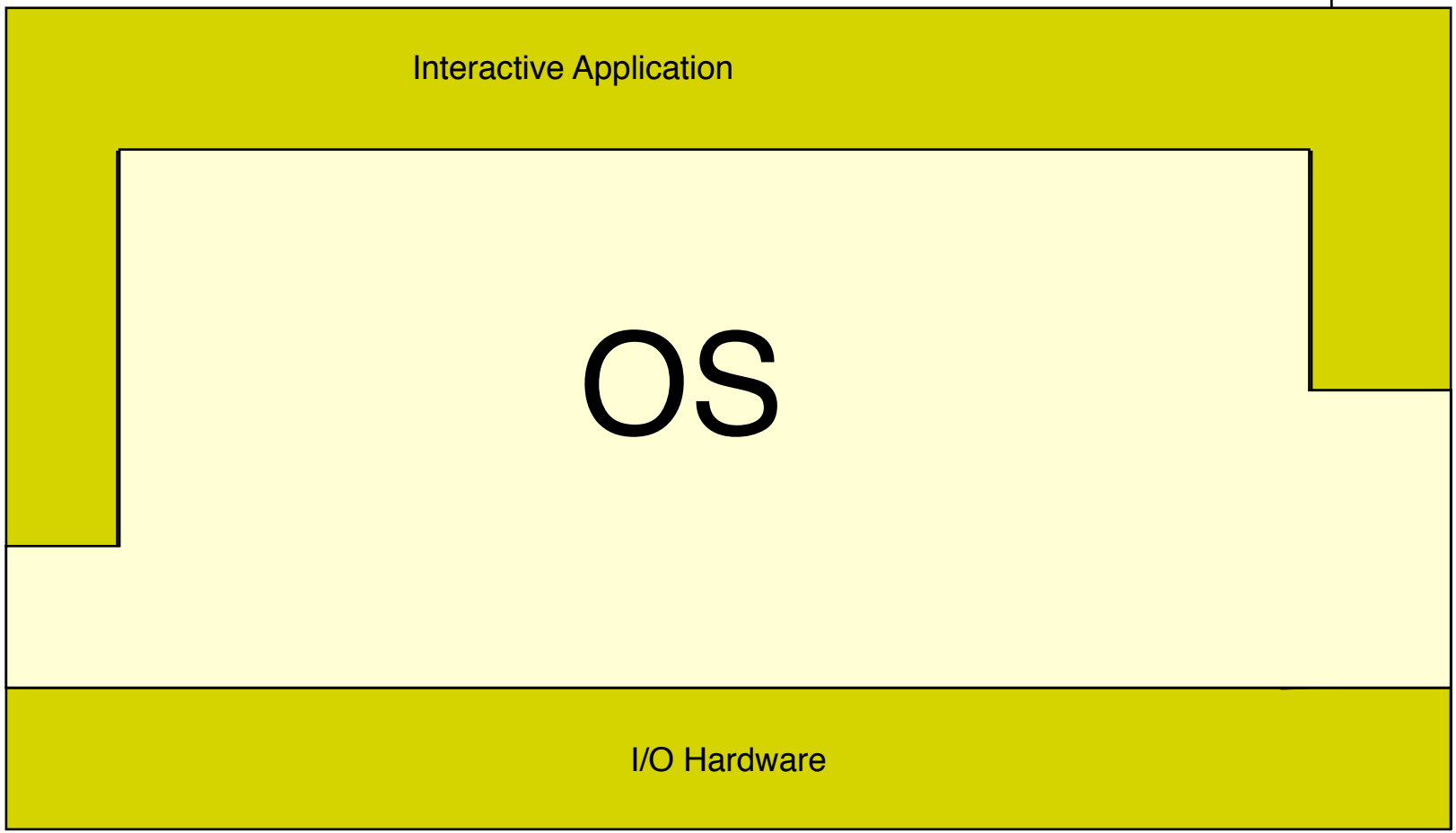


Georgia Tech

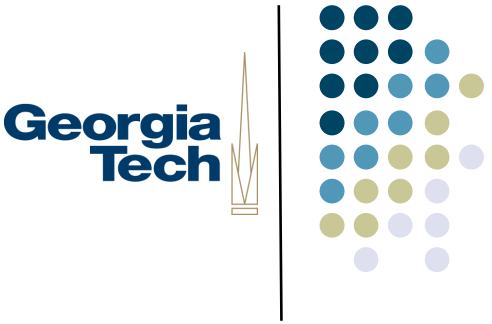
Interactive System Layers



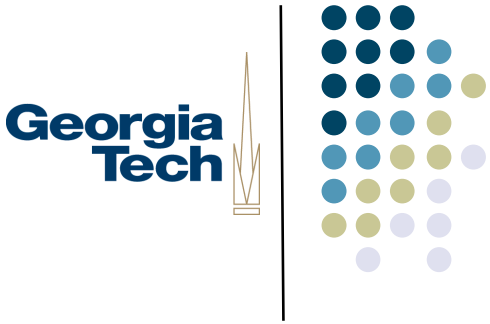
Because of commercial pressure:



Window Systems



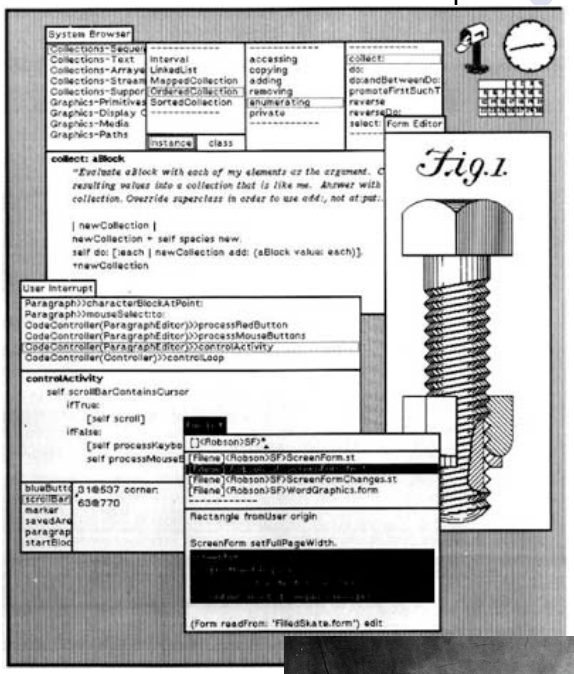
Output (and input) normally done in context of a window system



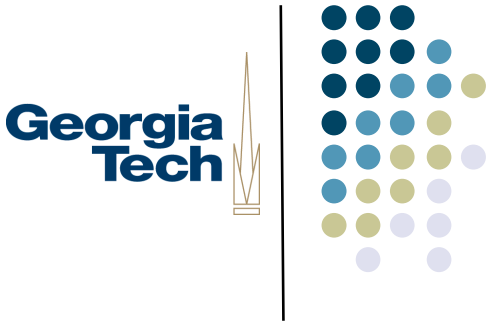
- Should be familiar to all
- Developed to support metaphor of overlapping pieces of paper on a desk (desktop metaphor)
  - Good use of limited space
    - leverages human memory
  - Good/rich conceptual model

A little history...

- The BitBlt algorithm
  - Dan Ingalls, "Bit Block Transfer"
  - (Factoid: Same guy also invented pop-up menus)
- Introduced in Smalltalk 80
- Enabled real-time interaction with windows in the UI



- Why important?
  - Allowed fast transfer of blocks of bits between main memory and display memory
  - Fast transfer required for multiple overlapping windows
  - Xerox Alto had a BitBlt machine instruction

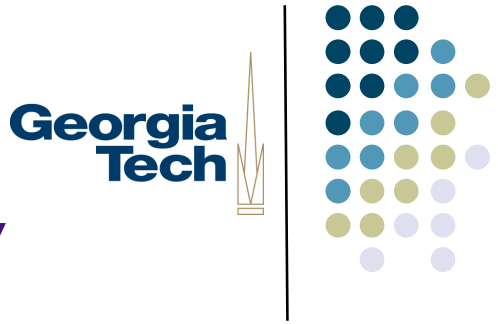


Goals of window systems

- Virtual devices (central goal)
  - virtual display abstraction
    - multiple raster surfaces to draw on
    - implemented on a single raster surface
    - illusion of contiguous non-overlapping surfaces

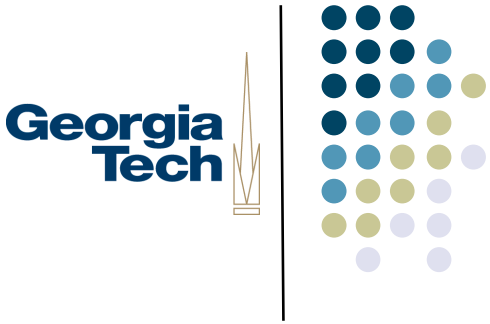
Virtual devices

- Also multiplexing of physical input devices
- May provide simulated or higher level "devices"
- Overall better use of very limited resources (e.g. screen space)
  - strong analogy to operating systems
  - Each application "owns" its own windows
  - Centralized support within the OS (usually)
    - X Windows: client/server running in user space
    - SunTools: window system runs in kernel
    - Windows/Mac: combination of both



Window system goals: Uniformity

- Uniformity of interface
  - two interfaces: UI and API
- Uniformity of UI
  - consistent "face" to the user
  - allows / enforces some uniformity across applications
    - but this is mostly done by toolkit

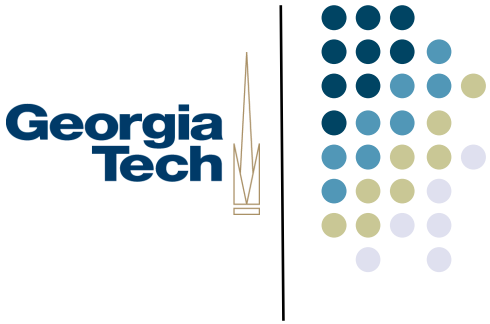


Uniformity

- Uniformity of API
  - provides virtual device abstraction
  - performs low level (e.g., drawing) operations
    - independent of actual devices
  - typically provides ways to integrate applications
    - minimum: cut and paste

Other issues in window systems

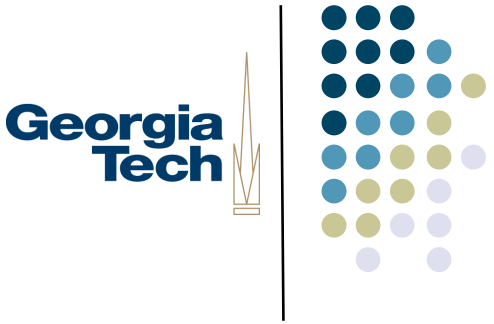
- Hierarchical windows
  - some systems allow windows within windows
    - don't have to stick to analogs of physical display devices
  - child windows normally on top of parent and clipped to it



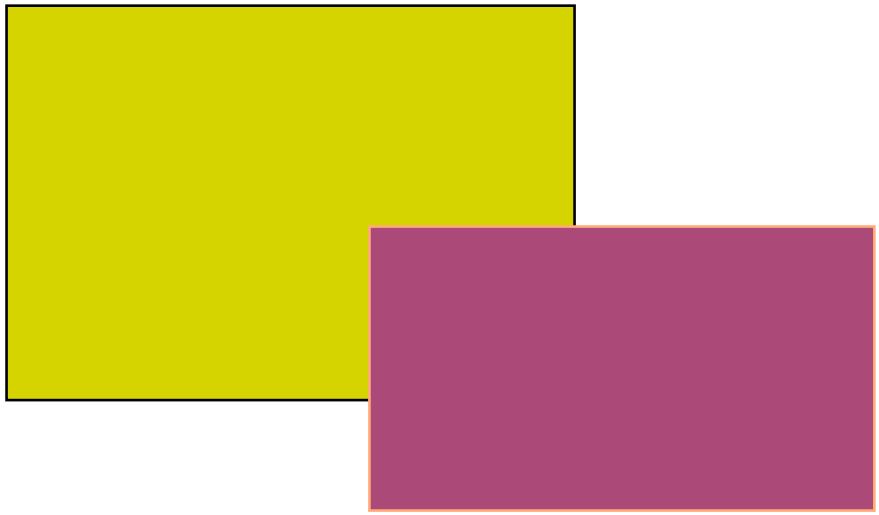
Issue: hierarchical windows

- Need at least 2 level hierarchy
  - Root window and "app" level
- Hierarchy turns out not to be that useful
  - Toolkit containers do the same kind of job (typically better)

Issue: damage / redraw mechanism

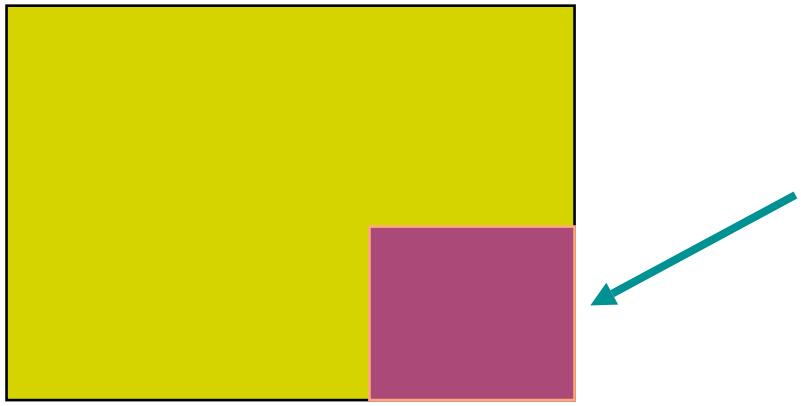


- Windows suffer "damage" when they are obscured then exposed (and when resized)



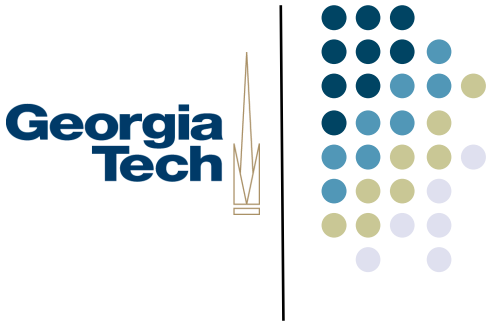
Damage / redraw mechanism

- Windows suffer "damage" when they are obscured then exposed (and when resized)



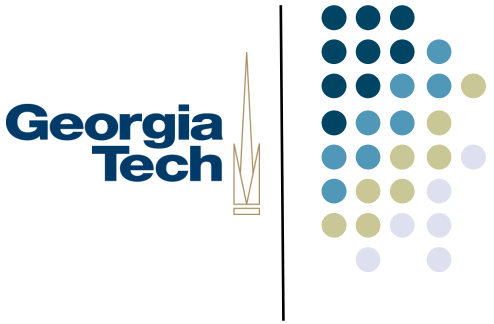
Wrong contents, needs redraw

Damage / redraw, how much is exposed?



- System may or may not maintain (and restore) obscured portions of windows
  - "Retained contents" model
  - For non-retained contents, application has to be asked to recreate / redraw damaged parts

Damage / redraw, how much is exposed?



- Have to be prepared to redraw anyway since larger windows create "new" content area
- But retained contents model is still very convenient (and efficient)
  - AWT doesn't do this, its optional under Swing

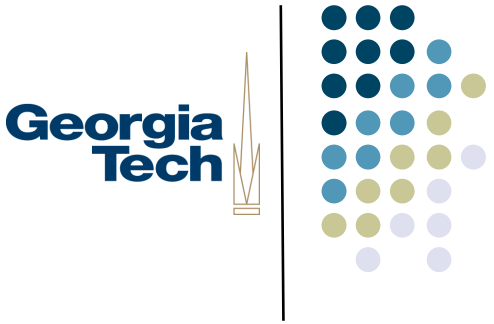
Output in Toolkits

- Output (like most things) is organized around the interactor tree structure
  - Each object knows how to draw (and do other tasks) according to what it is, plus capabilities of children
  - Generic tasks, specialized to specific subclasses

Output Tasks in Toolkits

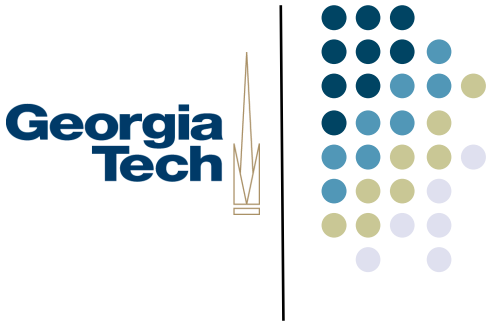
- Recall 3 main tasks
  - Damage management
  - Layout
  - (Re)draw

Damage Management



- Interactors draw on a certain screen area
- When screen image changes, need to schedule a redraw
  - Typically can't "just draw it" because others may overlap or affect image
  - Would like to optimize redraw

Damage Management

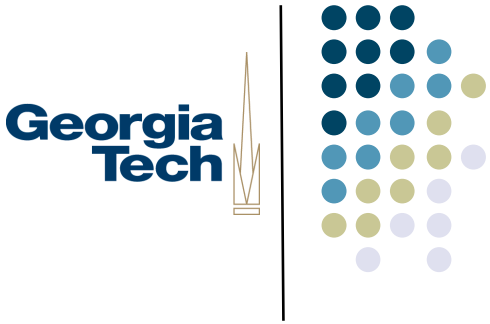


- Typical scheme (e.g., in Swing) is to have each object report its own damage
  - Tells parent, which tells parent, etc.
  - Collect damaged region at top
  - Arrange for redraw of damaged area(s) at the top
    - Typically batched
    - Normally one enclosing rectangle

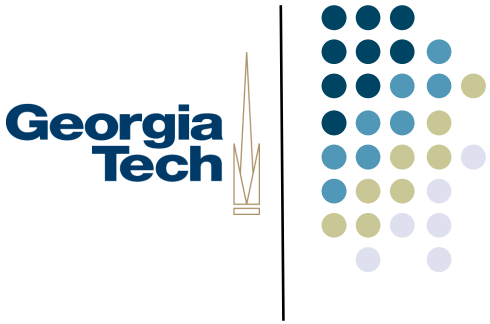
Redraw

- In response to damage, system schedules a redraw
- When redraw done, need to first ensure that everything is in the right place and is the right size
  - ➡ Layout

Can We Just Size and Position as We Draw?



Can We Just Size and Position as We Draw?



- No.
  - Layout of first child might depend on last child's size
    - Arbitrary dependencies
    - May not follow redraw order
- Need to complete layout prior to starting to draw

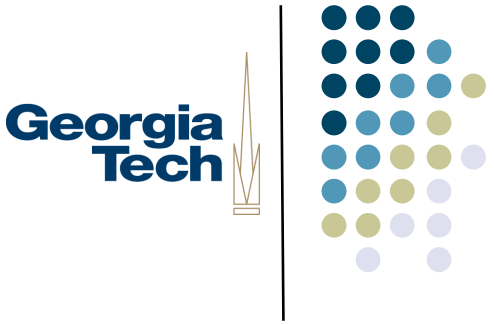
Layout Details

- Later in the course…

- But again, often tree structured
  - E.g., implemented as a traversal

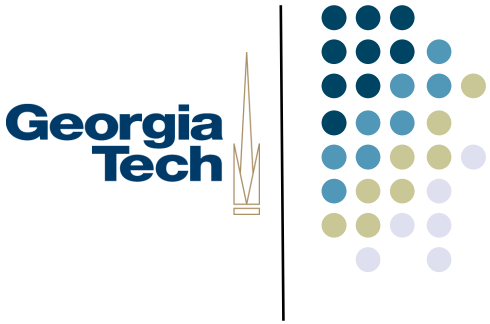
    Local part of layout +

    Ask children to lay themselves out



(Re)draw

- Each object knows how to create its own appearance
  - Local drawing + request children to draw selves (➡ tree traversal)
- Systems vary in details such as coordinate systems & clipping
  - E.g., Swing has parents clip children


Georgia
Tech