Event dispatch and propagation

Made by your 340 TAs Brian, David, Michelle, My, Jay :)

Vocab definitions

- **Event dispatch:** how events are handed out to Views
- **Event propagation:** order of how events are handed out
- **Consuming** an event: the event has been handled and will not be propagated any further
- ***Theoretical*** Positional propagation (know these in theory, don't worry about in practice):
  - **Bottom-up**: The event sent to the "lowest", frontmost interactor in the tree that contains the mouse position. If it's not consumed there it goes up the tree of interactors that contain the mouse position.
  - **Top-down**: Event is sent to the topmost interactor that contains the mouse location, then passed down recursively to children.
  - **Bubble out:** Traversal starts top down, bounding rectangles are hints. Event is sent to bottom most item (drawn last), the event can bubble back (with knowledge of what was hit). See Slide 13 for details.
- **Theoretical** Focus-based propagation: Windowing system determines which interactor gets the event, so there's no specific order

How event propagation happens (theoretical)

1. After the event happens, we choose and filter Views that can possibly get the event using "**picking**"
    a. Step through the view hierarchy in post order to get the list of Views
    b. Filter out Views that do not contain the mouse event
2. Traverse the list of picked views asking each one if they want to consume the event
    a. Traversal order depends on whether you use "**top down**" or "**bottom up**"
    b. If a View does not consume the event, it is propagated to the next one in the traversal order
3. Once the event is consumed, that's it! We will wait for the next event to happen and then repeat these steps again

How event propagation happens (real life)

1. In the real-world, **Android either does capturing or bubbling.**
2. You can decide between capturing versus bubbling depending on your application's needs.

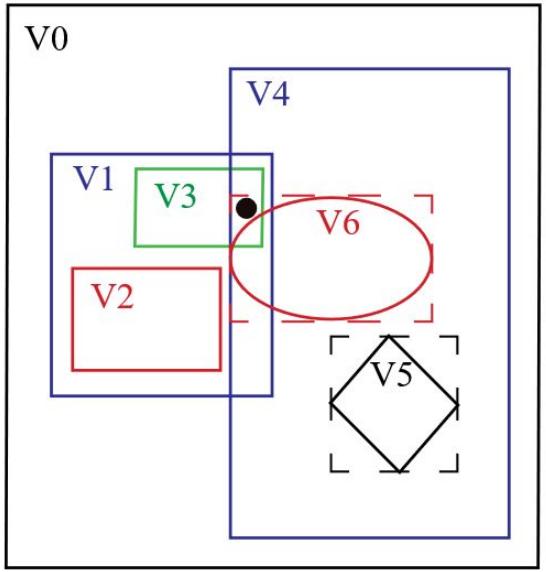
When you would use **bubbling**:

- **Using a button!** The button needs to get the input first so that it can pass on information up to its parent
- For example, when user clicks on the color picker and we want to highlight the entire color picker before showing what color they chose

When you would use **capturing**:

- Whenever the parent view needs the information before its child (perhaps if the parent wants to decide if the child should get the input)

Android event dispatch + propagation example

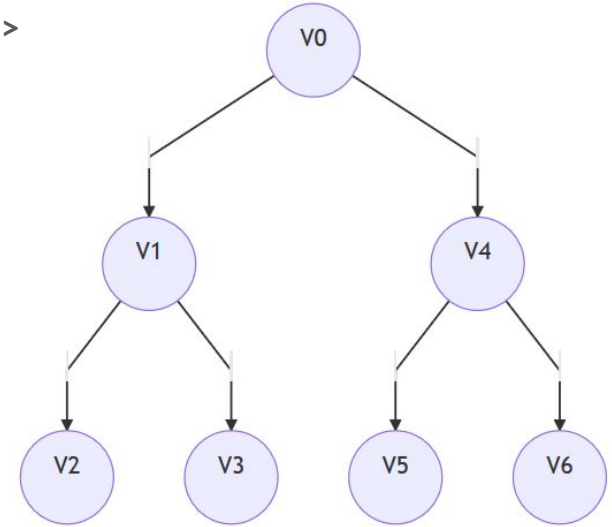


Hierarchy in tree form ->

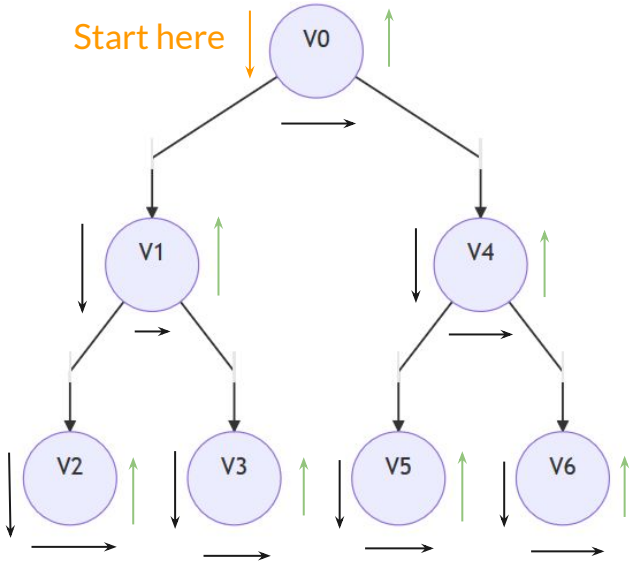
View hierarchy:

- V0
  - V1
    - V2
    - V3
  - V4
    - V5
    - V6

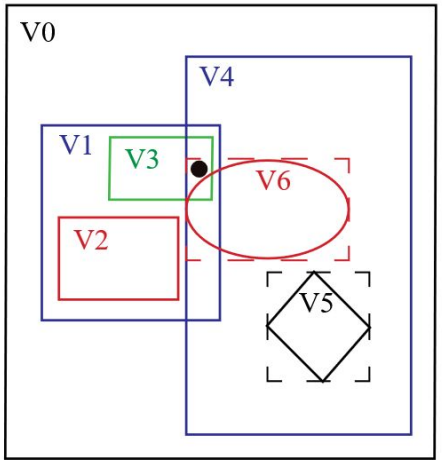
<- Layout wireframe



Example step 1: picking

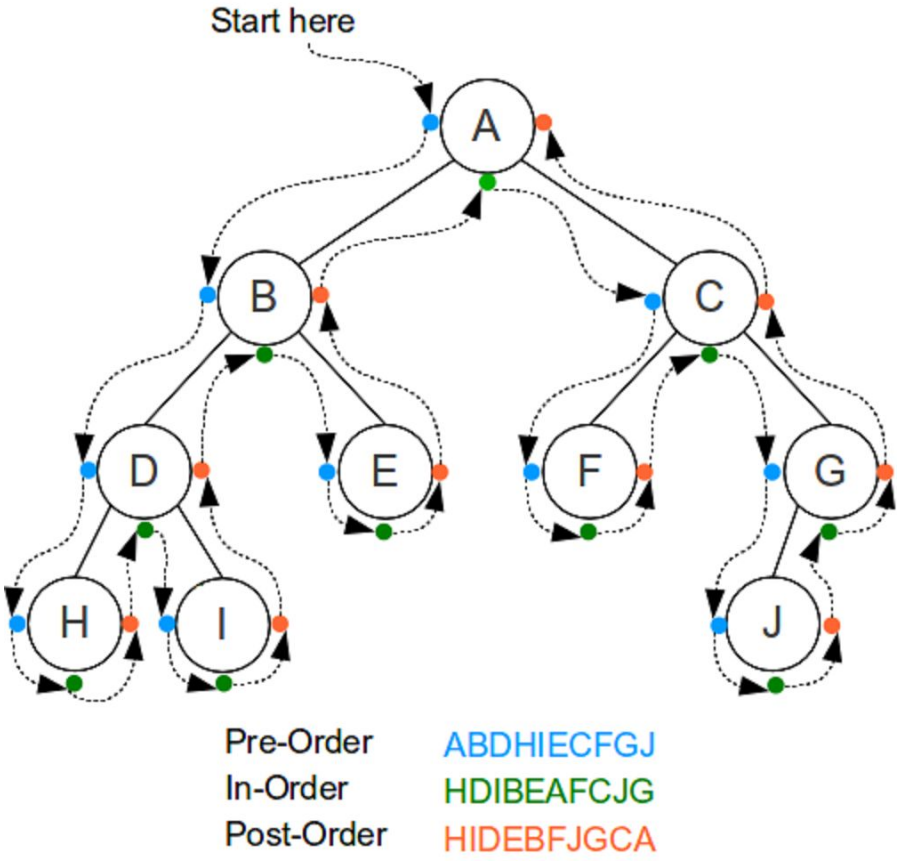


1. Do a **post order traversal** of the View tree
   a. For a review of post order traversal, see the next slide
2. List of Views after post order traversal:
   [V2, V3, V1, V5, V6, V4, V0]
3. Now filter out the Views that don't contain the event
   [~~V2~~, V3, V1, ~~V5~~, V6, V4, ~~V0~~]
4. Final list of picked Views:
   [V3, V1, V6, V4]



Interlude: Tree traversal review

For more in depth review, there are some good resources on the 143 website, GeeksforGeeks, and StackOverflow



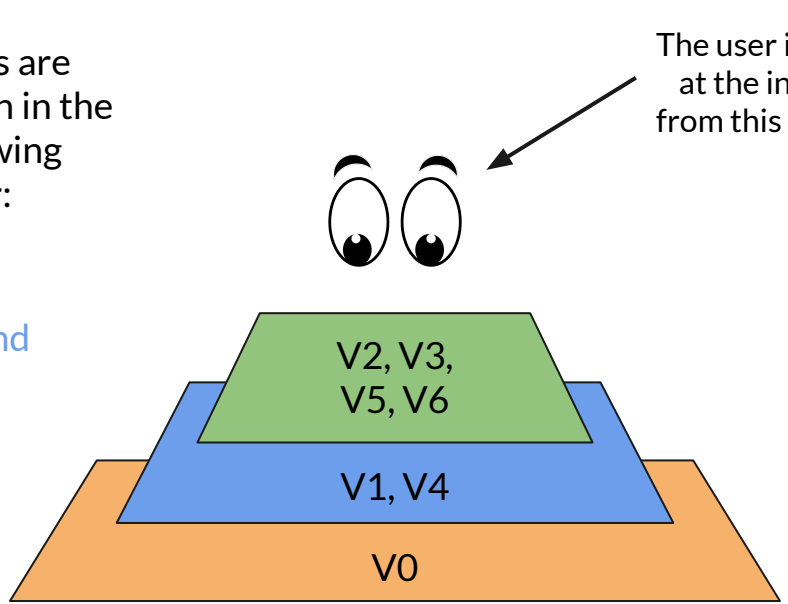
Order of Drawing Views

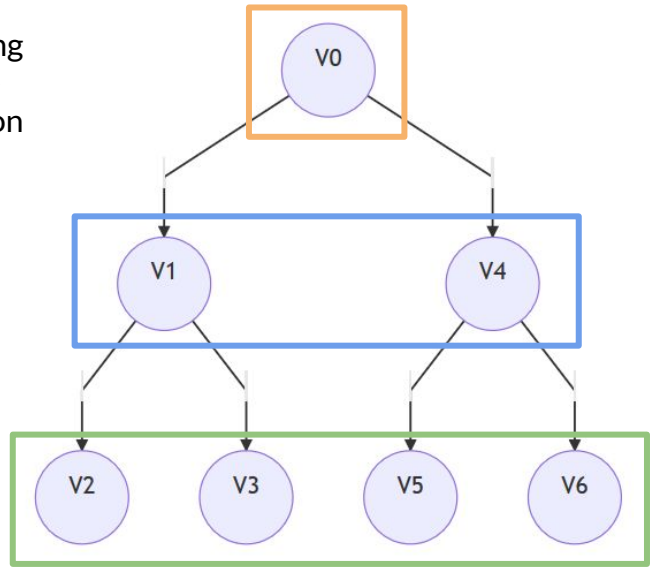
Views are drawn in the following order:

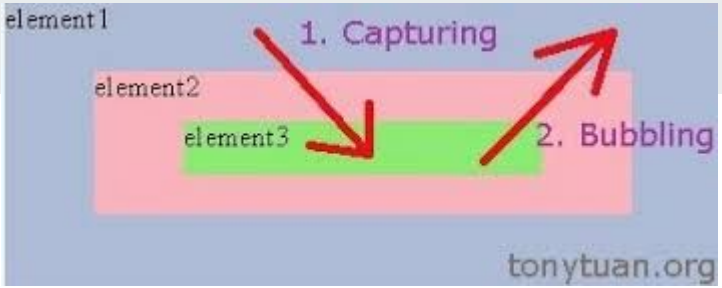
First

Second

Third







Example step 2 [option 1]: capture

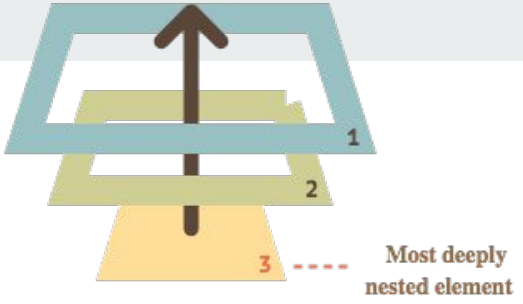
Suppose that V6 is the one consuming the event and our list of Views is [V3, V1, V6, V4]

1. Dispatch starts at V4 (end of the picked View object list)
2. V4 says it does not want to consume the event
3. Dispatch moves on to V6 (next on the list)
4. V6 says it wants to consume the event
5. Event propagation stops because event has been consumed
6. If V6 had not consumed the event, dispatch would ask V1, then ask V3 if V1 also does not consume the event

General capture process:

1. Dispatch starts at the **end** of the picked View object list (the top of the View hierarchy tree but the bottom-most/first thing that has been drawn)
2. Walking down the list, dispatch asks: will you consume this event?
    a. If true: the event is consumed and the event propagation stops
    b. If false: Move to the next element in the View list

Considered an implementation of the **top-down** theoretical dispatch strategy



Example step 2 [option 2]: bubble

Suppose that V6 is the one consuming the event and our list of Views is [V3, V1, V6, V4]

1. Dispatch starts at V3 (front of picked View list)
2. V3 says it does not want to consume the event
3. Dispatch moves on to V1 (next on the list)
4. V1 says it does not want to consume the event
5. Dispatch moves on to V6 (next on the list)
6. V6 says it wants to consume the event
7. Event propagation stops because event has been consumed
8. If V6 didn't consume the event, dispatch would ask V4

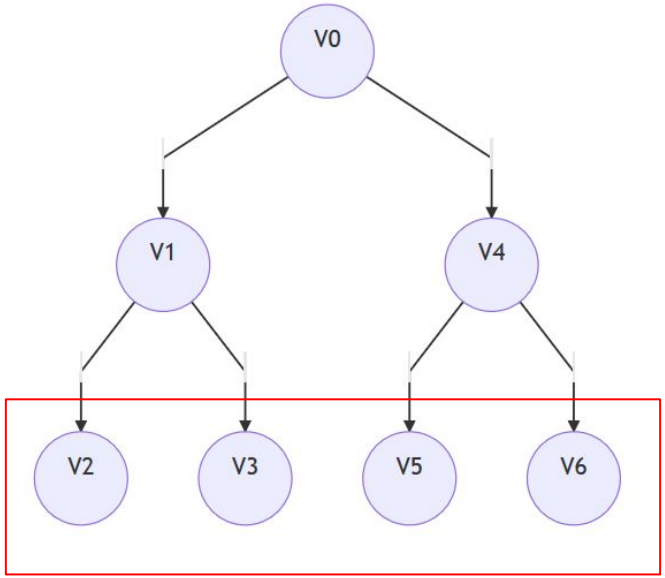
General bubble process:

1. Dispatch starts at the **front** of the picked View object list (the bottom of the View hierarchy tree but the top most/last thing that has been drawn)
2. Walking down the list, dispatch asks: will you consume this event?
    a. If true: the event is consumed and the event propagation stops
    b. If false: Move to the next element in the View list

Considered an implementation of the **bottom-up** theoretical dispatch strategy

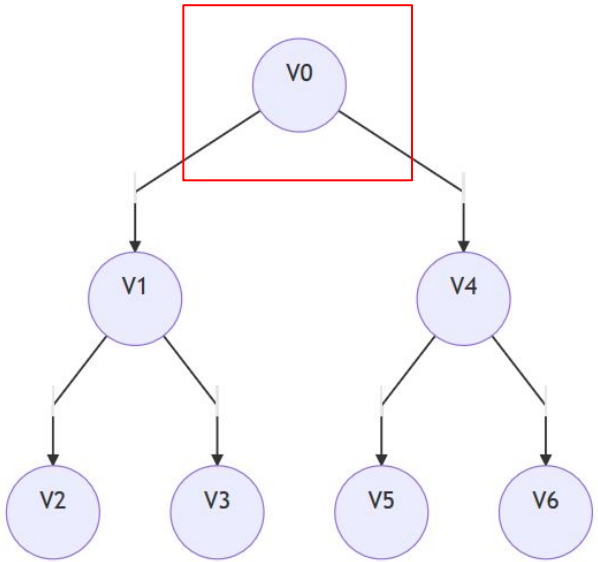
Event Dispatch *Theory* - Bottom Up

1. In this case, we would dispatch the event first to V2, V3, V5, and V6 since they are the “**lowest**” and frontmost Views
2. If the event is not consumed, it would be propagated up to V1 and V4
3. If the event still has not been consumed, it would be propagated up to the parent View V0



Event Dispatch *Theory* - Top Down

1. In this case, we would first dispatch the event to V0 because it is the **topmost** parent View
2. If V0 does not consume the event, it will be propagated down to V1 and V4
3. If the event still has not been consumed, it will be further propagated down to V2, V3, V5, and V6



Event Dispatch *Theory* - Bubble Out (NOT BUBBLING IN EVENT PROPAGATION)

- ***Note: Know this in theory, but don't worry about it in practice!***
- Bubble Out used when there **isn't a clear nesting of different interactors**

Process:

1. The front-most objects are checked first if they want to consume the event
2. If the object decides to consume the event, the object attaches to the event
3. This way, the parent object knows which child object consumed the event
    a. Example: Object 1 and Object 2 has attached to the event, now object 1 (parent) knows that object 2 consumed the event

Object 1 ---------- Event1

Object 2 ----------/

Additional Resources

- Sophie's Ed post: https://us.edstem.org/courses/381/discussion/50520
- Input dispatch process lecture slides: https://courses.cs.washington.edu/courses/cse340/20sp/slides/wk05/pps-geom.html#14
- Bubbling and Capturing: https://javascript.info/bubbling-and-capturing