Modeling Input

Input, Interaction, & Accessibility
Spring 2019

Check-in on HTML/CSS assignment

- How is it going?
- Stuck on anything?

Today

- Modeling input methods
- Describing input devices

Note

- Today is a bit of an infodump
- Want to give you some high level tools to work with
- We'll revisit later
- Focus on the concepts and knowing which tools are available

Goals for today

- Establish a solid foundation for talking about input and interaction methods
- Begin to explore tools for evaluating and comparing these methods
- See an input device and think about how it can be used, what performance might be like

Today

- More on KLM
- Input devices
- State models in HCI
- Pointing

The Keystroke-Level Model (1983)

Big ideas

- Top-down vs. bottom-up thinking in user interfaces
  - **Top-down:** Understanding users' goals, plans, understanding, examining misconceptions
  - **Bottom-up:** Understanding the mechanics of how people perform tasks

KLM: Background

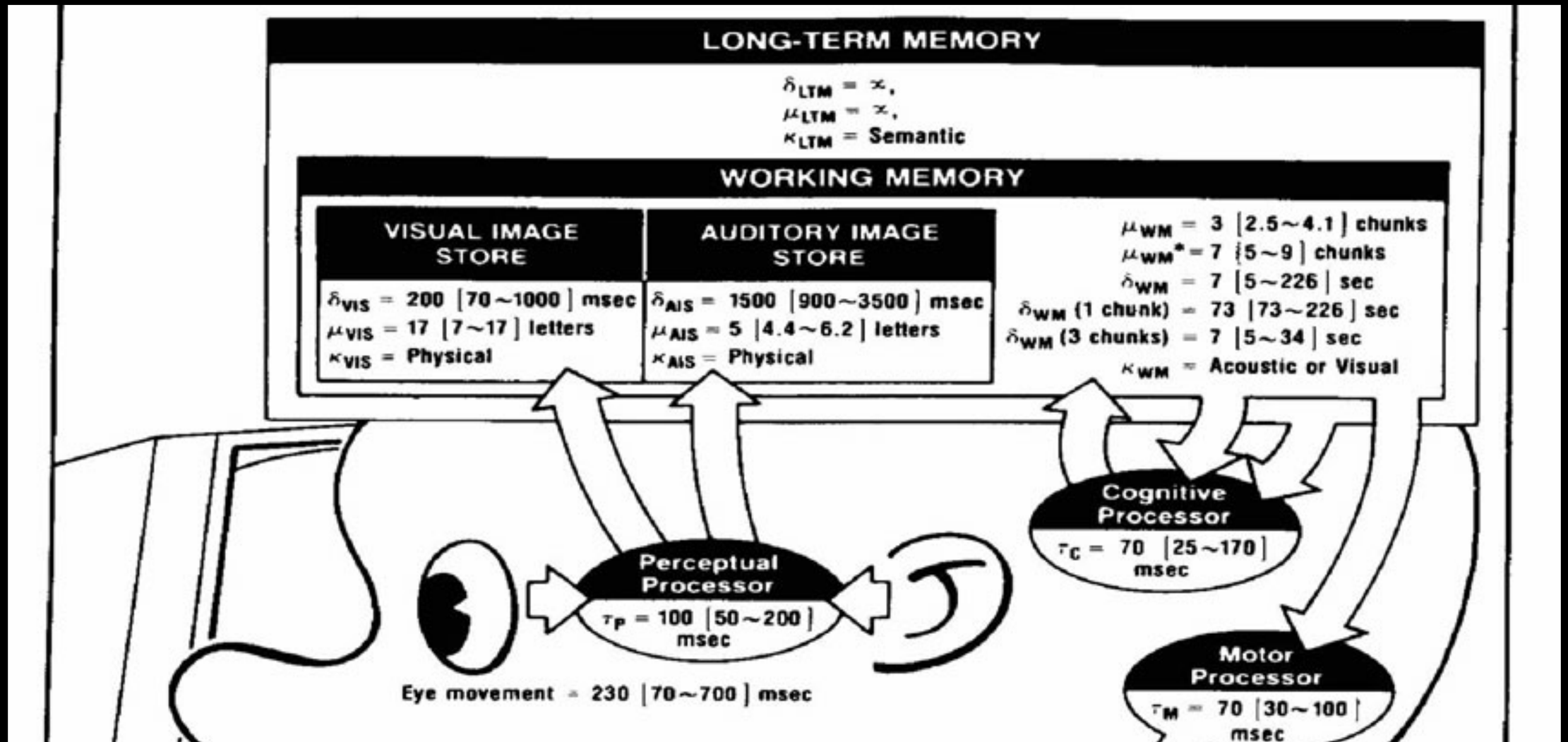
- Developed by
  - Stu Card - psychologist from Xerox PARC
  - Tom Moran - engineer from Xerox PARC
  - Allen Newell - psychologist from CMU
- Followed on from cognitive modeling / information processing theory in psychology
- Goal: estimate task performance by timing each component of an interaction

Big ideas: KLM

- We can think about interactions with our UI in terms of a sequence of low level actions: keystrokes, pointing, homing, drawing, and “mental work”
- Pointing and text entry are the fundamental building blocks of UI interaction
  - What else is there?

Relatives of KLM

Model Human Processor (Card, Moran, Newell 1983) - Model cognitive operations



Relatives of KLM (2)

- **GOMS (Card, Moran, Newell 1983):**
  - Defines tasks in terms of **goals, operators, methods,** and **selection (of methods)**
  - Multiple GOMS models (KLM is the simplest one)

Cognitive models in HCI

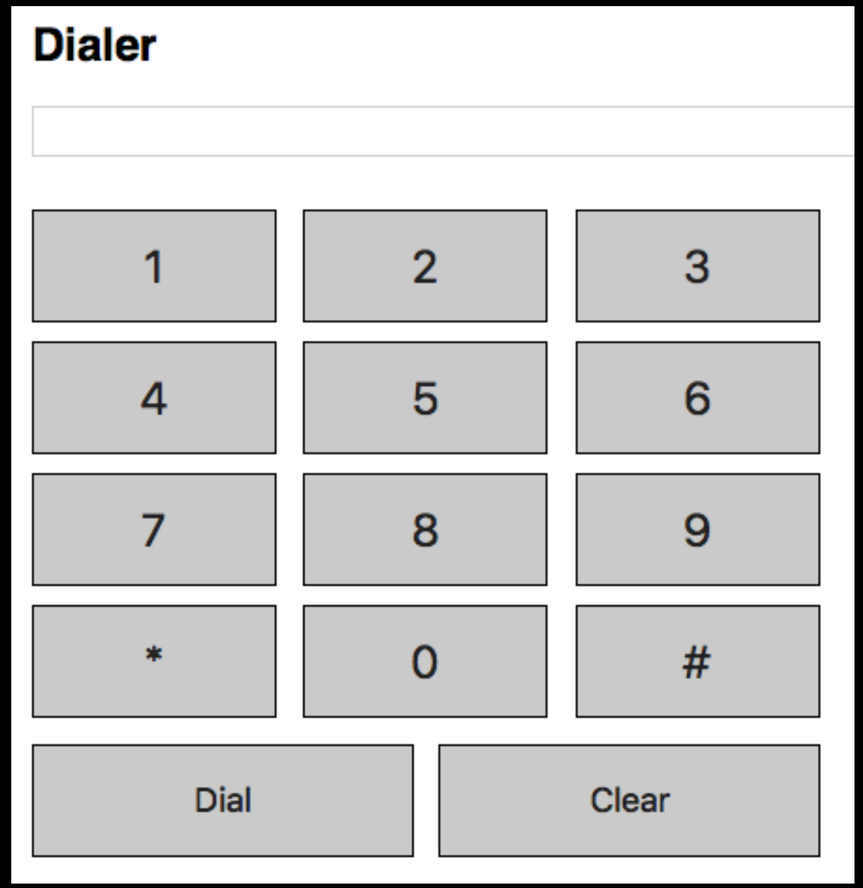
- Extremely popular in the 1980s
- Currently a niche topic - why?
- Still used in some domains
  - Mobile input
  - Screen reader use

Why bother with cognitive models?

- In this class, we'll be examining different types of input devices
- Provides a way of estimating the overall effort to access an interface, and approximating the effects of different input methods

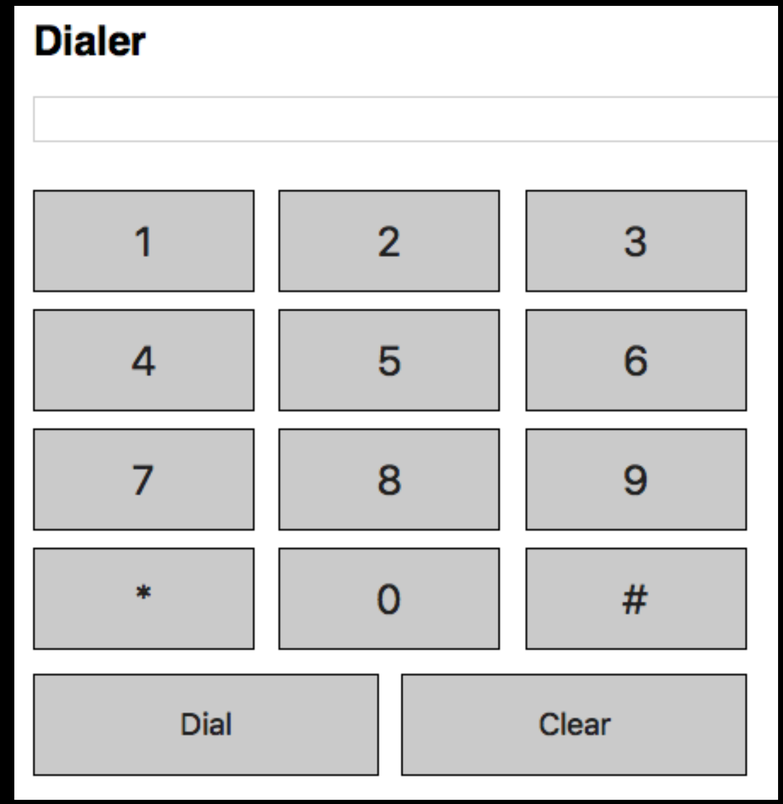
Modeling example

- Consider a numeric keypad used to dial 10-digit numbers
- Using the test string: 001–010-0101
- How do we model this for mouse input? Keyboard input?



Modeling 001–010-0001

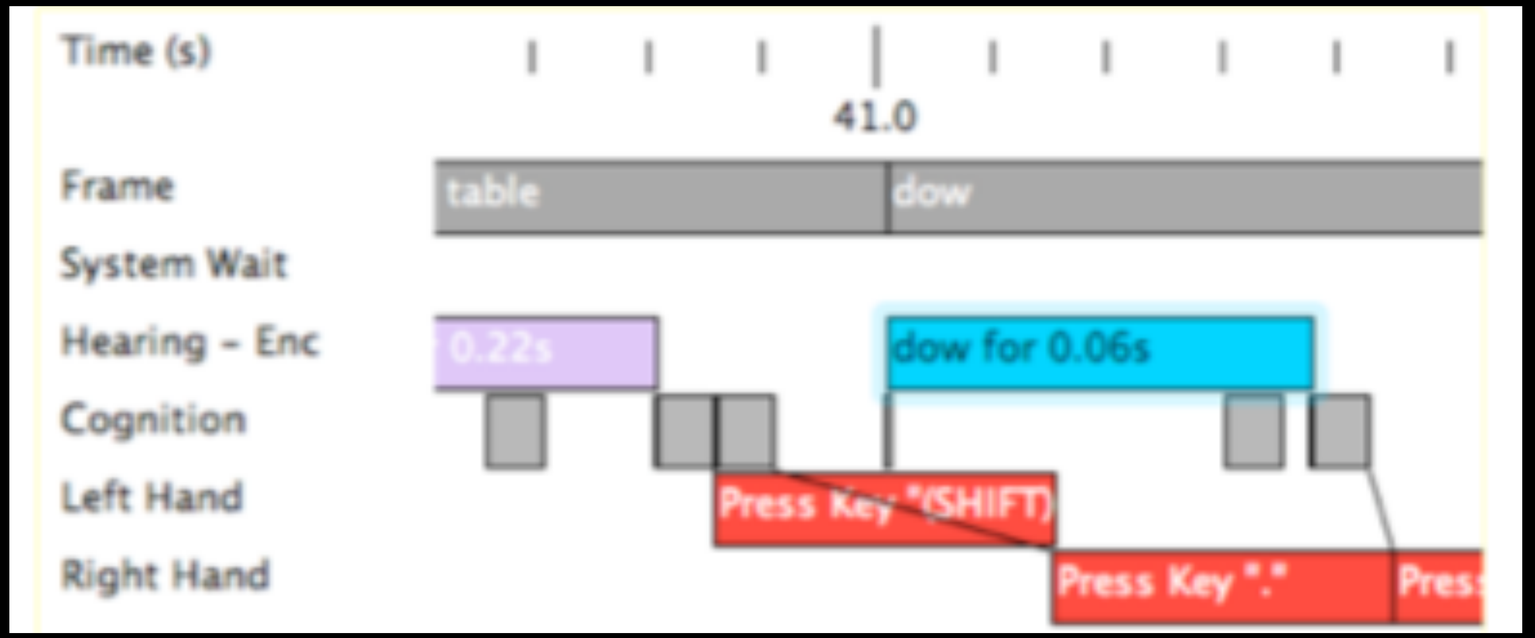
- Mouse input:
  - (H?)(PK)*10
- By default in HTML, the user would navigate between keys using TAB (K) and will select using enter (K):
  - (H?)K(tab)*10+K(enter)+K(enter)+K(shift)+K(tab)*10+K(enter) …
- What if we add keyboard shortcuts? (1-0)
  - K(0)+K(0)+K(1) …



KLM for screen readers

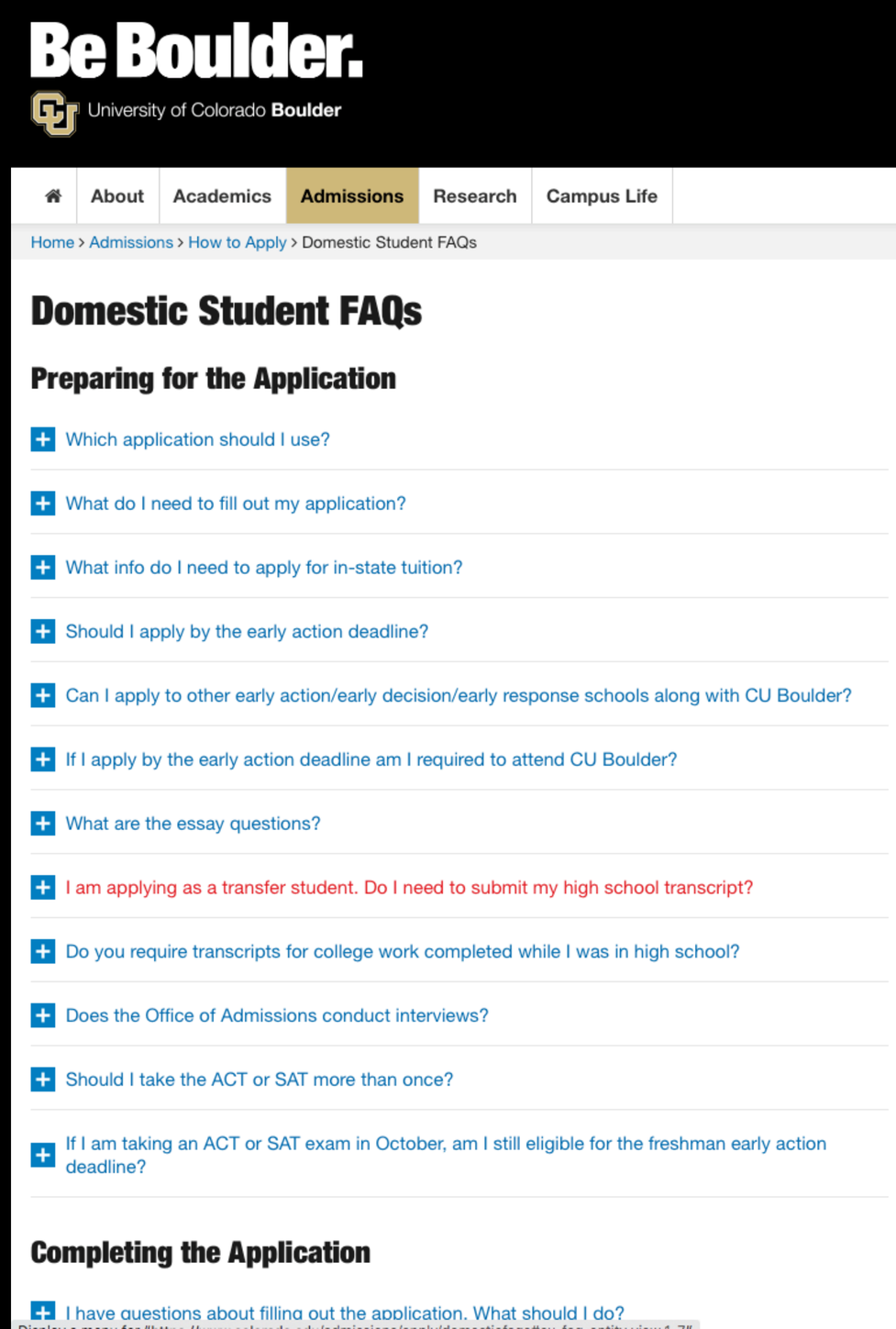
(Trewin et al., ASSETS 2010)

- Operations:
  - Hearing
  - Cognition
  - Left hand
  - Right hand



Example 2

- FAQ has 4 categories (preparing application, completing application, after submitting the application, understanding admissions decisions)
- Assume student wants to get to 5th item of 4th section
- On an improperly structured web page, a screen reader user might need to tab through each item (approx 10 items * 3 categories)
- Properly marking headers will allow a screen reader user to navigate directly through headings, reducing steps by an order of magnitude



Example 3: StickyKeys

- StickyKeys is a system-wide feature to make modifier keys (shift, ctrl, alt) **modal**
  - To get $ -> press Shift and then press 4
- So, to get from 0 to 1:
  - (K(Shift)+K(tab))*10 -> 20 keystrokes

StickyKeys

- Adding a basic feature: press modifier twice to hold it down
- K(shift)+K(shift)+K(tab)*10 = 12 (40% reduction in keystrokes)

Summary: KLM

- Isn’t this just common sense?
- In some ways, yes - estimating this informally is often good enough
- But, there are times where this can be important
  - Mission critical tasks
- Important to compare all possible input methods
  - Sometimes it really is incredibly inefficient for users of alternative UIs
  - Designers usually make sure that whatever method **they** use is efficient

Next: Input devices

Talking about input devices

- What is sensed? Actuating a button or switch, position, movement, force, etc.
- Dimensions of input / degrees of freedom
- Characteristics of the input device: sensor speed, latency, precision
- Direct vs. indirect input

Current trends

- Move from single interaction mode to multiple interaction modes (multimodal interaction)
- New sensing technologies give us even more freedom to define interactions (Nintendo Labo)
- Complementary technologies: voice good for picking a song, maybe less so for laying out a poster

“Natural” and “intuitive”

- Can be misleading
- It’s difficult to separate out what we know naturally from what we’ve learned
- Force yourself to think through what about it seems natural or intuitive

Why is this intuitive?

- Can see all the options (don't need to remember them)
- Can click to select one (don't need to remember a command)
- Icons enable quick visual search

Considering unconventional use

- Often people will use devices in unconventional ways or situations
- This may affect what is possible via that input mode

Deceptive Devices and Illusory Interactions

Anderson, Grossman, Wigdor, Fitzmaurice

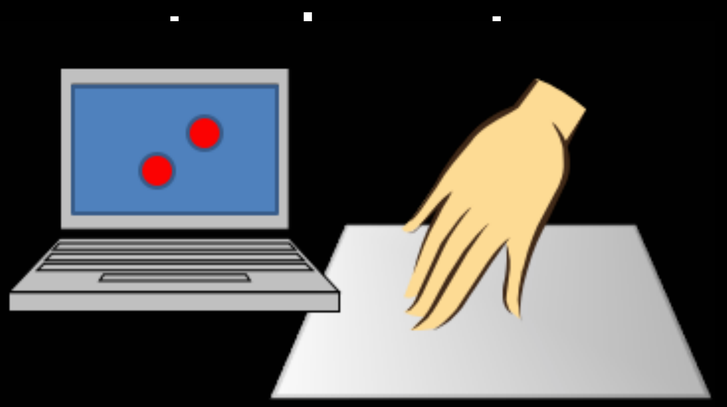
Understanding input

- Direct vs. indirect
- Absolute vs. relative
- Transfer functions
- Clutching
- **Modeling input state**

Direct vs. indirect input

- **Direct:** input and output are colloca

- **Indirect:** input and output are sep
  usually shown with a cursor



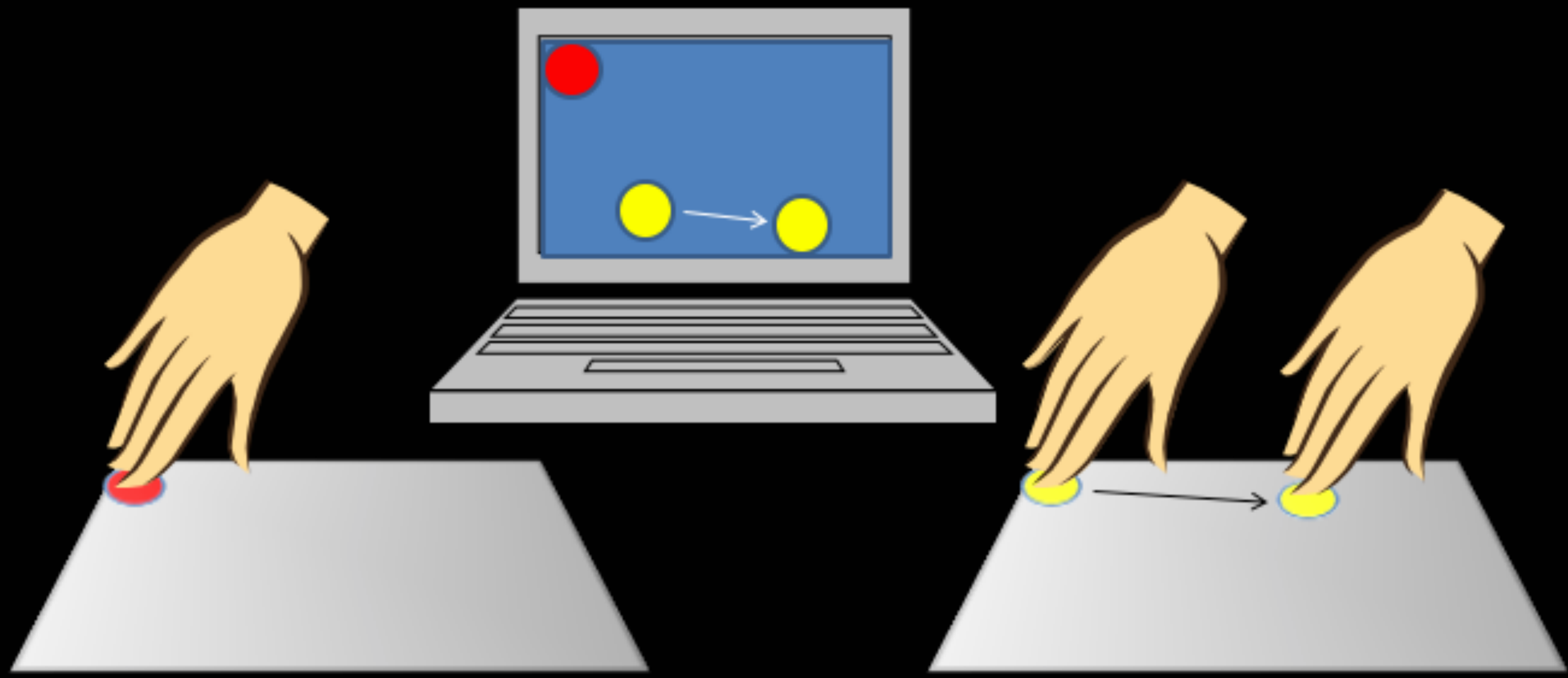
Direct or indirect?

- Mouse
- Touchpad
- Trackball
- Touch screen

Benefits of direct vs. indirect input?

- Benefits of direct input?
  - Less indirection, more self-contained, greater expressivity, easier to learn
- Benefits of indirect input?
  - Precision, “fat finger problem”
  - Efficiency, specialized devices
  - Density of controls which could be faster
  - Occlusion
  - Tactile / proprioceptive sensations

Absolute vs. relative pointing



Absolute: user touches (x,y) and cursor moves to (x,y)

Relative: cursor stays in place, is moved (dx,dy) by user input

Transfer function

- How do we map movement on a relative pointing device to cursor movement?
- Transfer function f(dx,dy) relates to cursor movement
- What if we get it wrong? Too low? Too high?
- **Clutching:** picking up and repositioning an input device, such as a mouse

Input device state

Input modeling

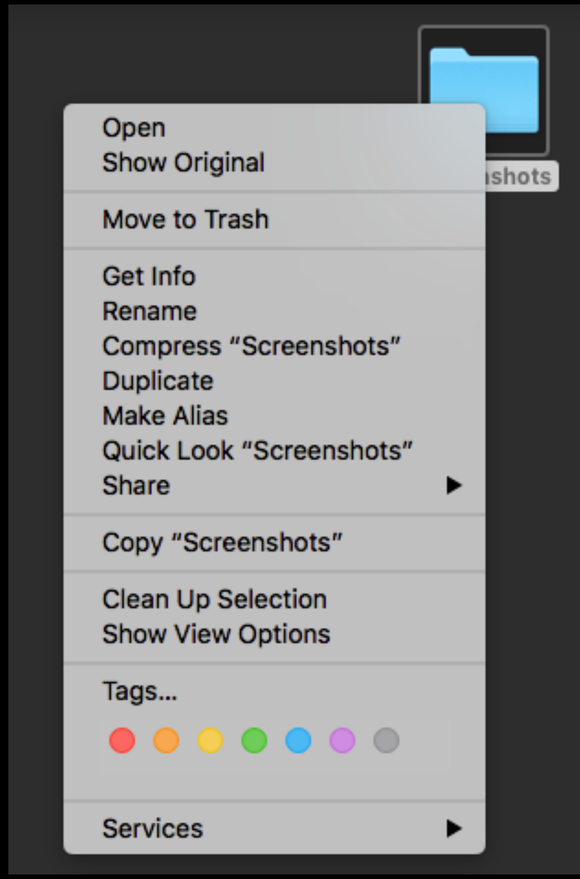
- Understand that interaction takes place in stages
- From the user's perspective: *pointing, clicking*
- From the device's perspective: *mouse button up, mouse button down*
- We can model the states (and transitions) of input devices to understand how they work

Why modeling input state matters

- Comparing two input devices: can one be substituted for another?
- Identifying parts of the user interface that will be tricky on a new type of device
- Finding underutilized input methods and using them to provide more expressive input

Example: right-click

- Most mouse-based user interfaces have the capability of “right-click” or other alternate selection
- Most touch screens do not
- How have designers dealt with that issue?



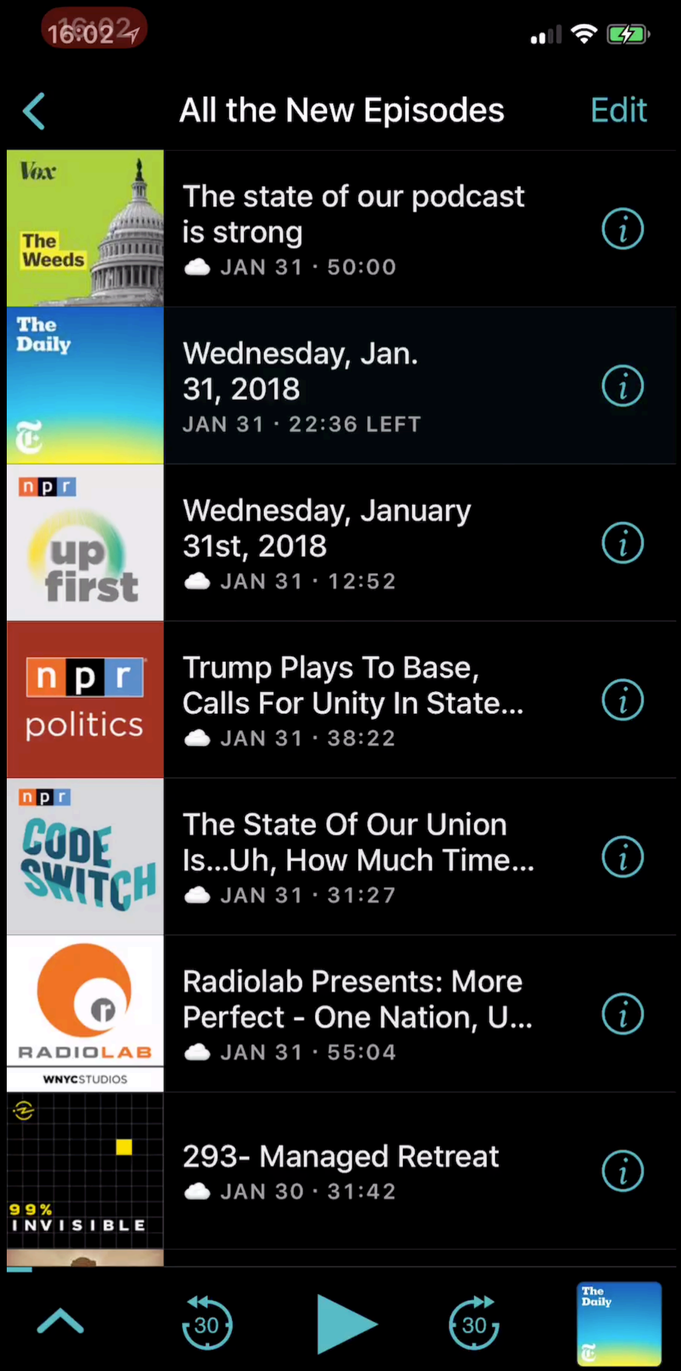
Solving no right click

Solving the no-right-click problem

- Touch and hold (“dwell”) - slows down input
- Add another layer of menus - adds steps
- Use gestures - this can work, since gestures are physically easy to perform on touch screens, but may be difficult to learn or remember
- Pressure sensing - add new sensing capabilities to the input device

Previewing gestures on iOS

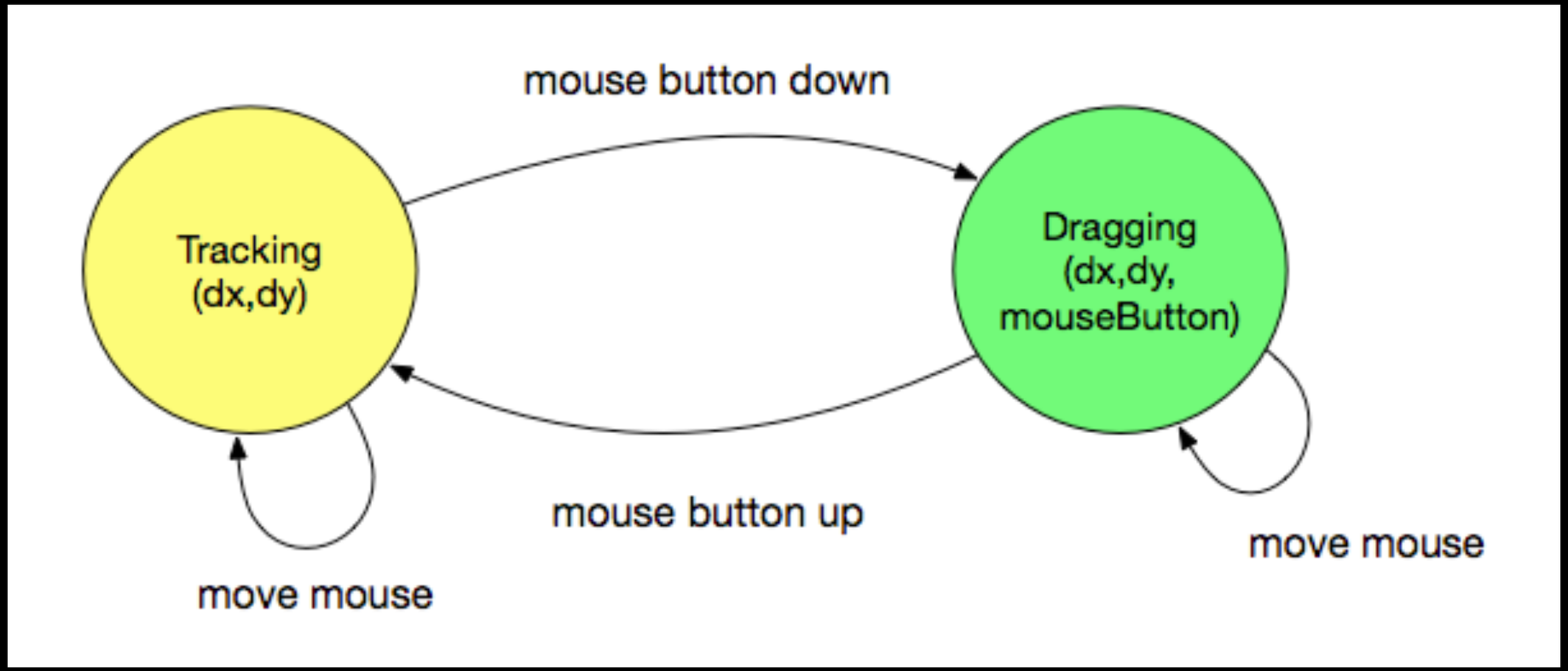
- Drag item slightly to the side
- Options “peek out”



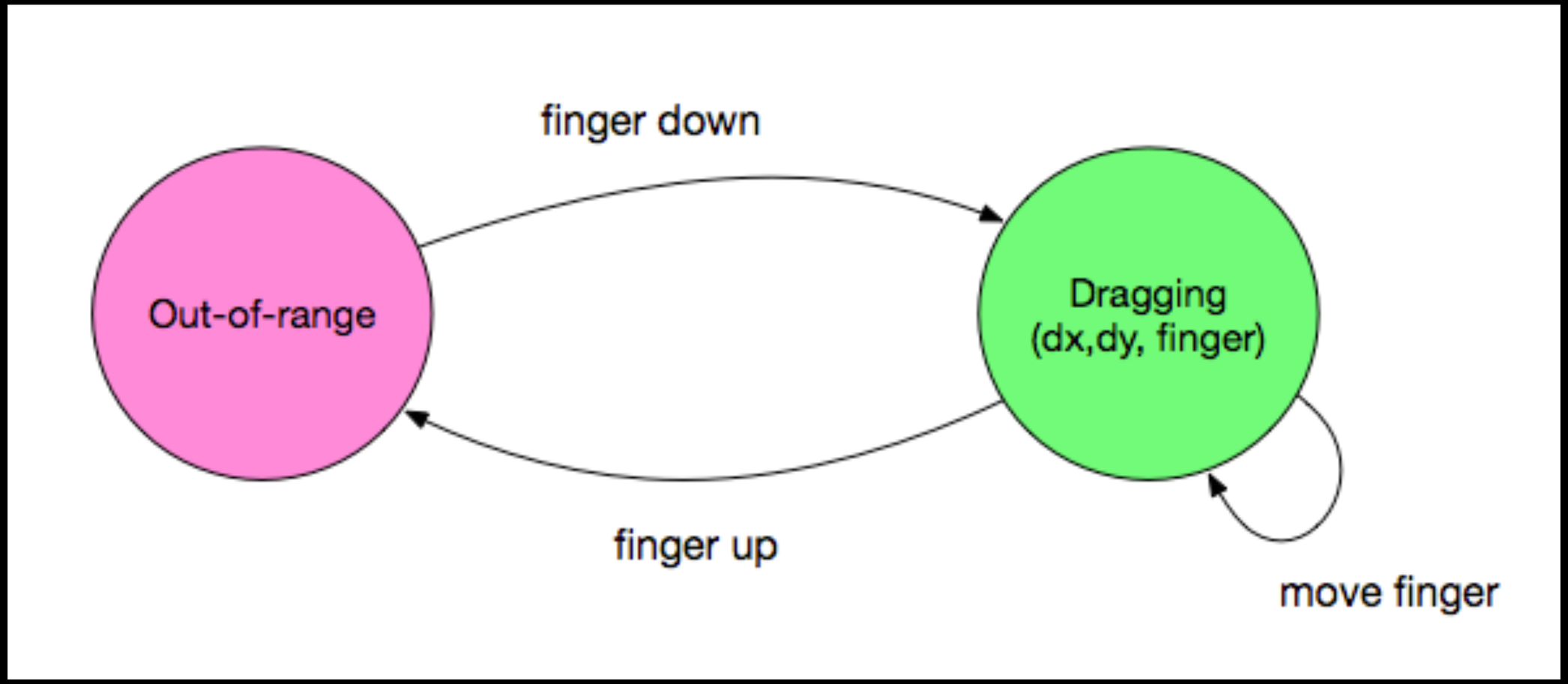
State models

- We can represent the state of an input device using a state-transition diagram
- Draw a circle for each state, and draw arrows showing how to transition between each state
- Track what data is recorded during each state

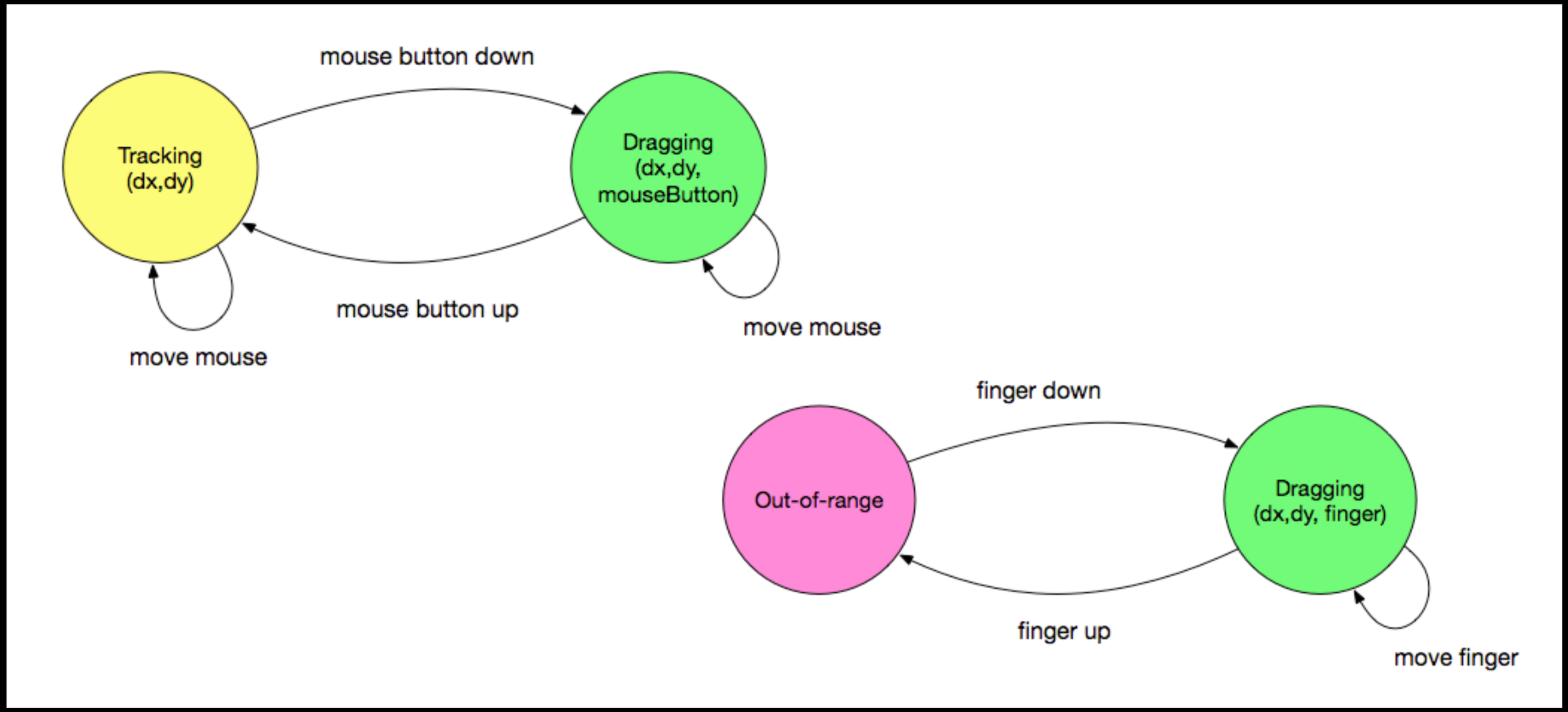
Mouse state-transition diagram



Touch screen state-transitions



Mouse vs. touch screen

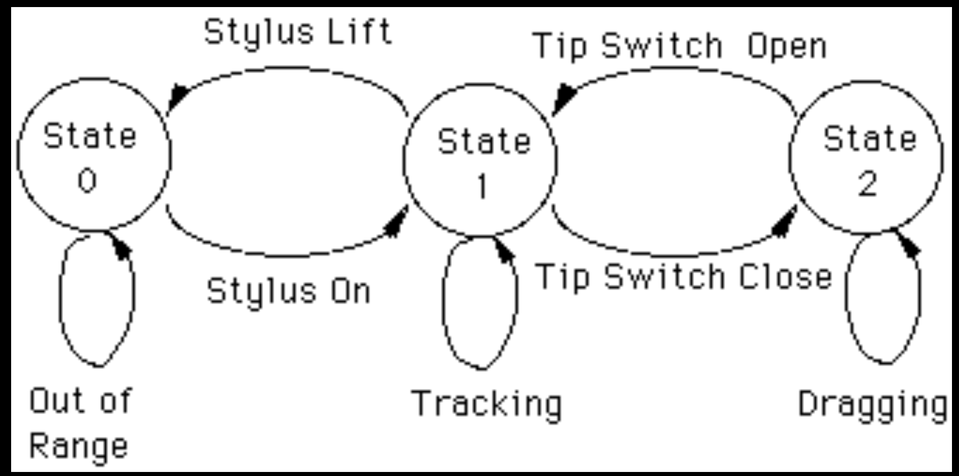


Mouse vs. touch: so what?

- Mouse has “hover” state, touch doesn’t
  - Need alternative ways to preview actions
- No way to “lift” the mouse cursor off of the screen
  - Need to move cursor from A to B
  - Usually this means actions are only taken when the mouse button is down

The three-state model

- Standard model for user input devices, proposed by Bill Buxton in 1990
- **Out-of-range:** system cannot sense the input device
- **Tracking:** system is tracking the position of the device (but no interaction)
- **Dragging:** user moves an item on screen by moving the input device



How programs handle input

- **Input events** - our programming environment has built-in functionality for tracking keyboard, mouse, touch input
  - Example events: mouseDown, mouseUp, mouseMove, keyDown, keyUp
- We can write event handlers that respond to those events

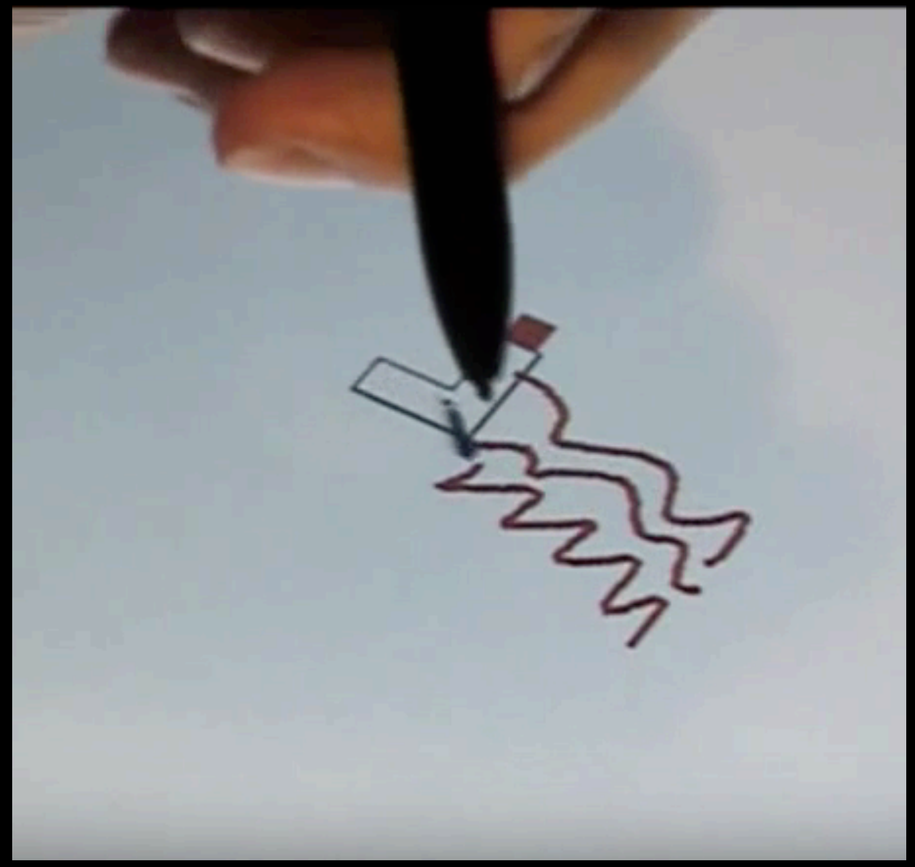
Event handling in JQuery

```
$("#dialer_pad").click(function() {
    // do something
})

$(“#dialer_pad").mousemove(function(event) {
    alert(event.x)
})
```

Repurposing leftover degrees of freedom

- Hover Widgets (Grossman, Hinckley, Baudisch, Agrawala, Balakrishnan)



Let’s model some input devices

- **To my left:** Model a two-button mouse
- **To my right:** Model the pen used in hover widgets

State models

- Two-button mouse
  - …
- Active stylus
  - …

Adding complexity

- So how would you prototype something like Hover Widgets?
- Usually, track low level events, and keep track of the state

HoverWidgets in pseudocode

```
var isHovering = false;

function onMove(direction) {
  if (direction == right) isHovering = true;
  else if (direction == left) isHovering = false;
  else if (direction == down) isHovering = false;
  else if (direction == up) {
    if (isHovering) == true doHoverAction()
    else isHovering = false
}
```

Summary: input devices

- Several categories / dimensions of input devices
- We can model the sensing state of devices and map them back to a task

Pointing

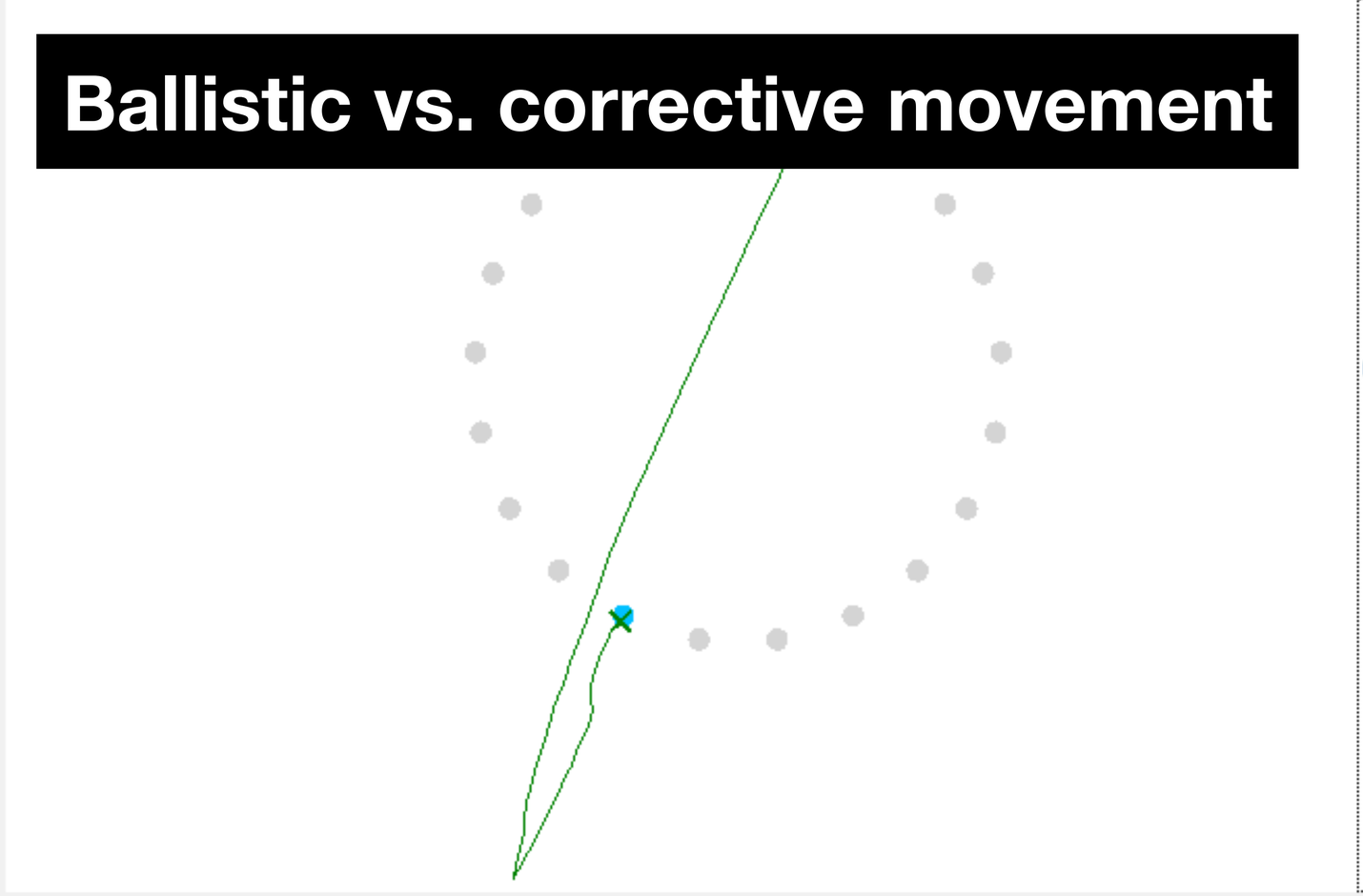
What is pointing, anyway?

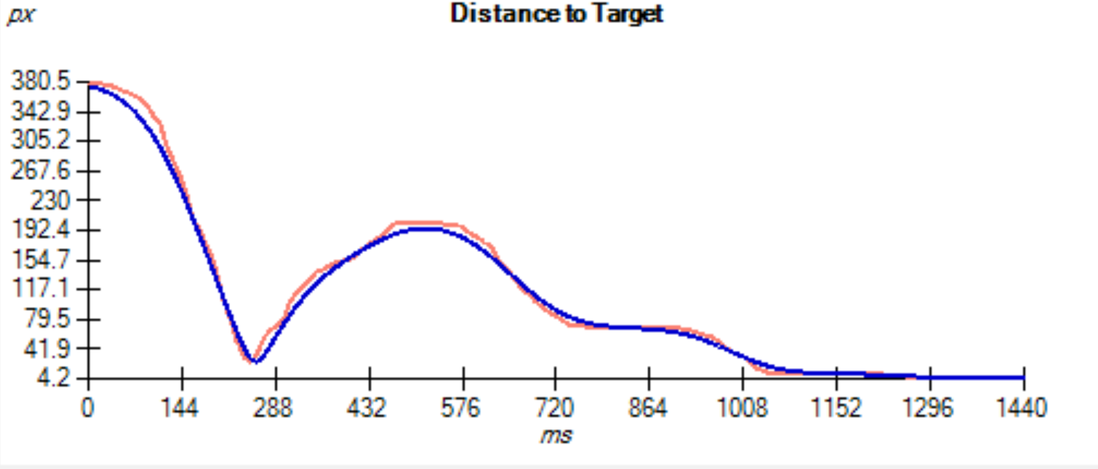
- We might think of it in narrow terms: e.g., using a mouse to move a cursor and click an icon
- More generally, all **aimed movements** are pointing
- Typing == pointing at a series of physical keys, and actuating them

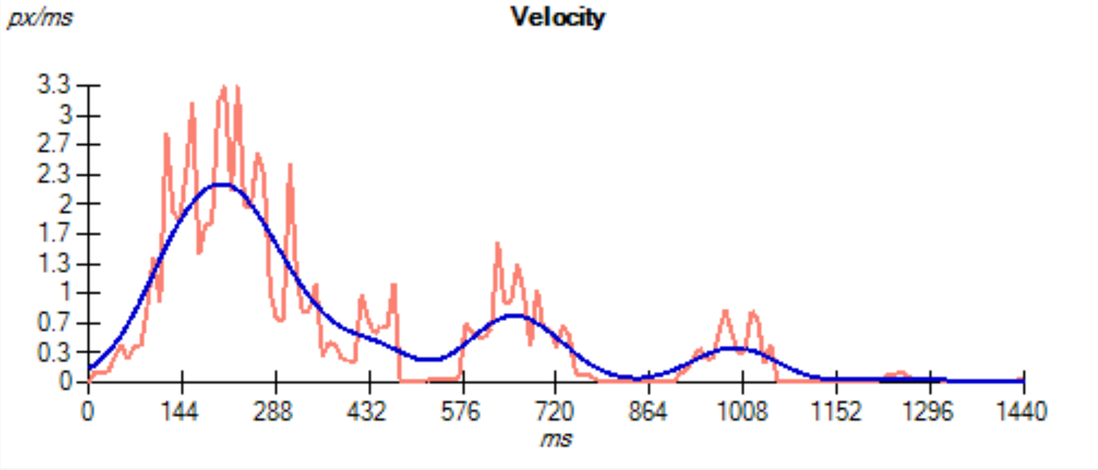
Hand movements in pointing

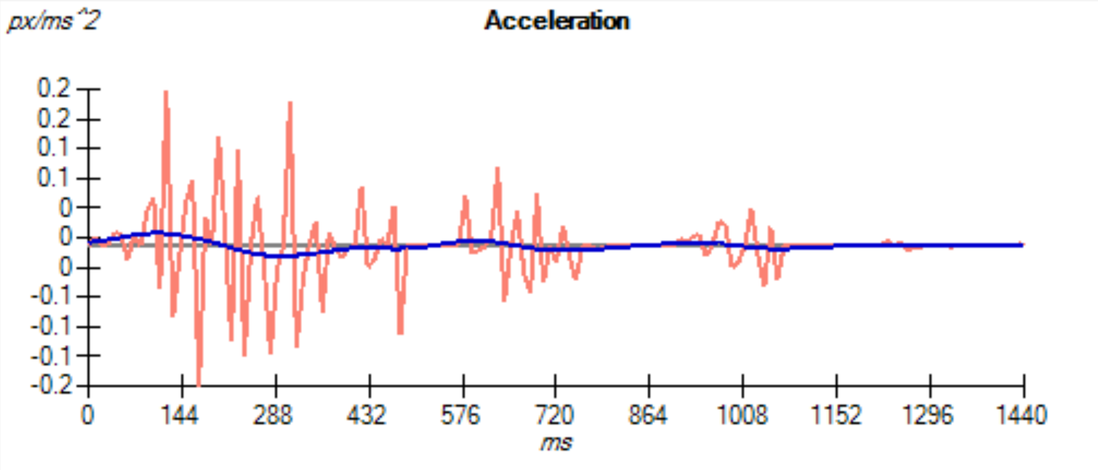
- Pointing is **closed loop movement**
  - User responds to visual and proprioceptive feedback
- First, a fast, inaccurate **ballistic movement**
- Then, a series of small, iterative, precise **corrective movements**

Ballistic vs. corrective movement



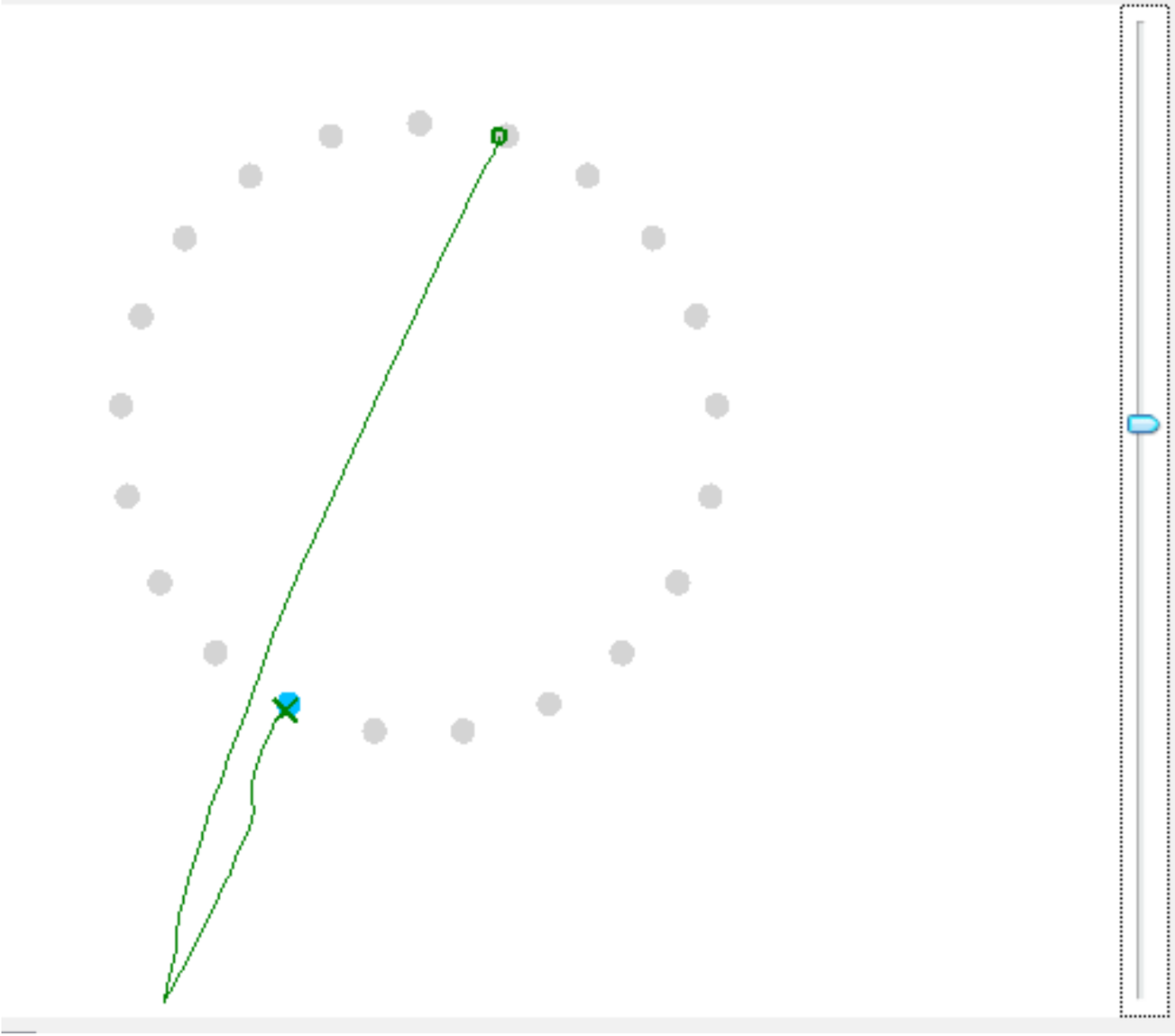


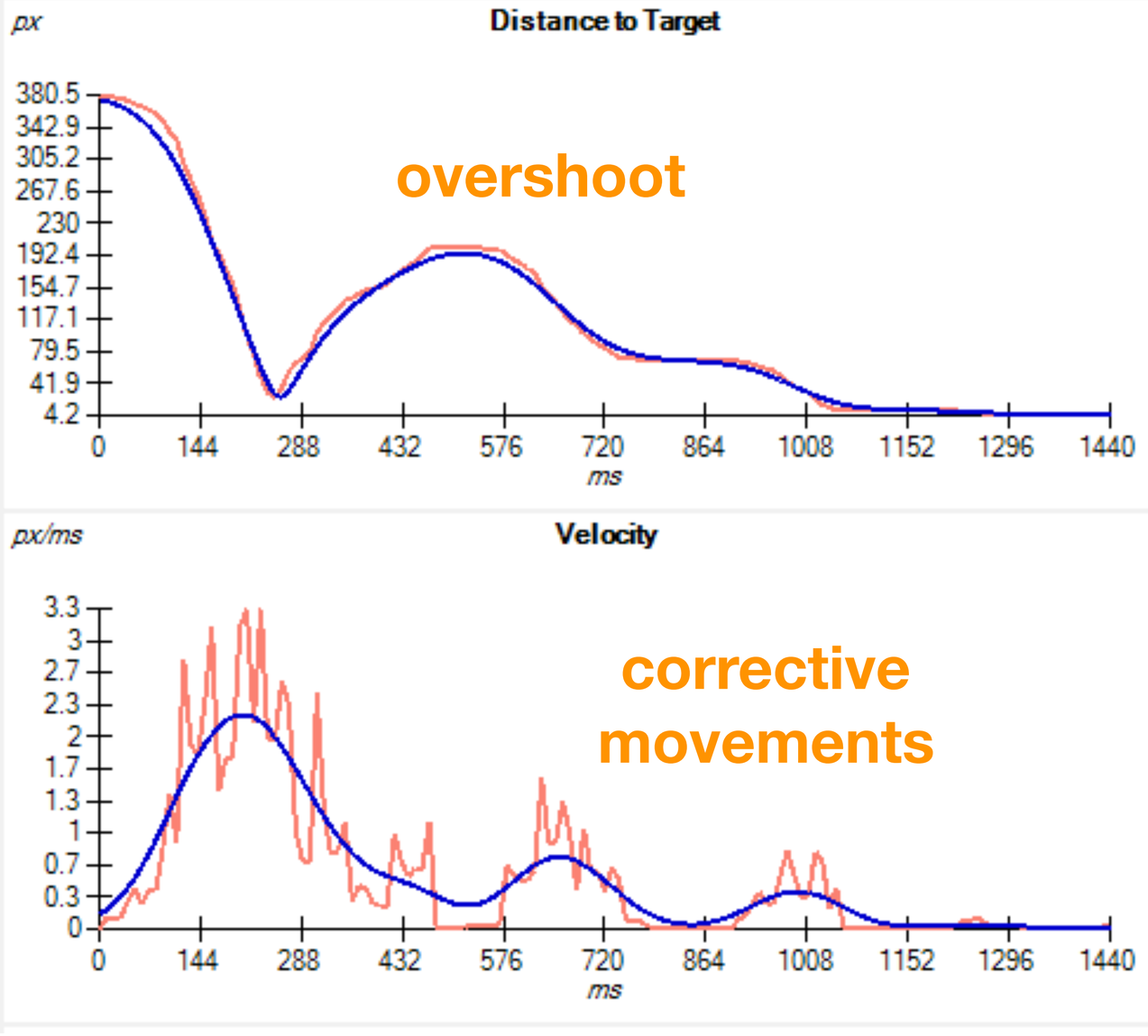




Web | Plain


```
- <Trial number="3" circular="true" metronome="false" completed="true" practice="false"
    lastCircle="{X=776.593,Y=266.53,Radius=8}"
    thisCircle="{X=636.6943,Y=622.986,Radius=8}" isoCenter="{X=720, Y=450}"
    MT="-1" A="385" W="16" axis="111.4286" angle="110.8346" ae_1d="385.1842"
    dx_1d="3.7697" ae_2d="385.187" dx_2d="4.0427" MTe="1527" MTRatio="-1"
    entries="1" overshoots="1" error="false" spatialOutlier="false" temporalOutlier="false"
    start="{X=771,Y=266,Time=0}" end="{X=634,Y=626,Time=1527}">
  - <Movement count="63" travel="792.4244" duration="1446" submovements="4"
      maxVelocity="{X=230, Y=3.343278}" maxAcceleration="{X=120,
      Y=0.1929051}" maxJerk="{X=120, Y=0.02452481}" taskAxisCrossings="2"
      movementDirectionChanges="4" orthogonalDirectionChanges="1"
      movementVariability="7.7" movementError="8.1896" movementOffset="8.1503">
```




px
Distance to Target
380.5
342.9
305.2
267.6
230
192.4
154.7
117.1
79.5
41.9
4.2
0
144
288
432
576
720
864
1008
1152
1296
1440
ms
overshoot
px/ms
Velocity
3.3
3
2.7
2.3
2
1.7
1.3
1
0.7
0.3
0
0
144
288
432
576
720
864
1008
1152
1296
1440
ms
corrective movements

Evaluating pointing?

- How should we evaluate whether our input techniques are good?
- Let’s brainstorm criteria

Evaluating input devices

Question

- Which is better, a mouse or trackball?
- How do we answer a question like this?

An effective pointing device is…

Questions about pointing

- Answer questions like:
  - How does this input device compare to another?
  - Is interface layout A or B more efficient?
  - Does interface C or D work better for User X?

Evaluating pointing

- We can measure movement times (and error rates)
- But these have relatively little explanatory power
  - If pointing is slow, should we always just make the cursor move faster?
  - What knobs should we turn to improve performance?
  - What do our test results tell us about other situations?

Fitts’ Law

Big idea: Fitts' Law

- Formula that provides an accurate model of pointing performance in a given context
  - Allows us to estimate in what-if scenarios
  - Helps explain differences between different pointing scenarios
  - Provides actionable steps for changing pointing performance

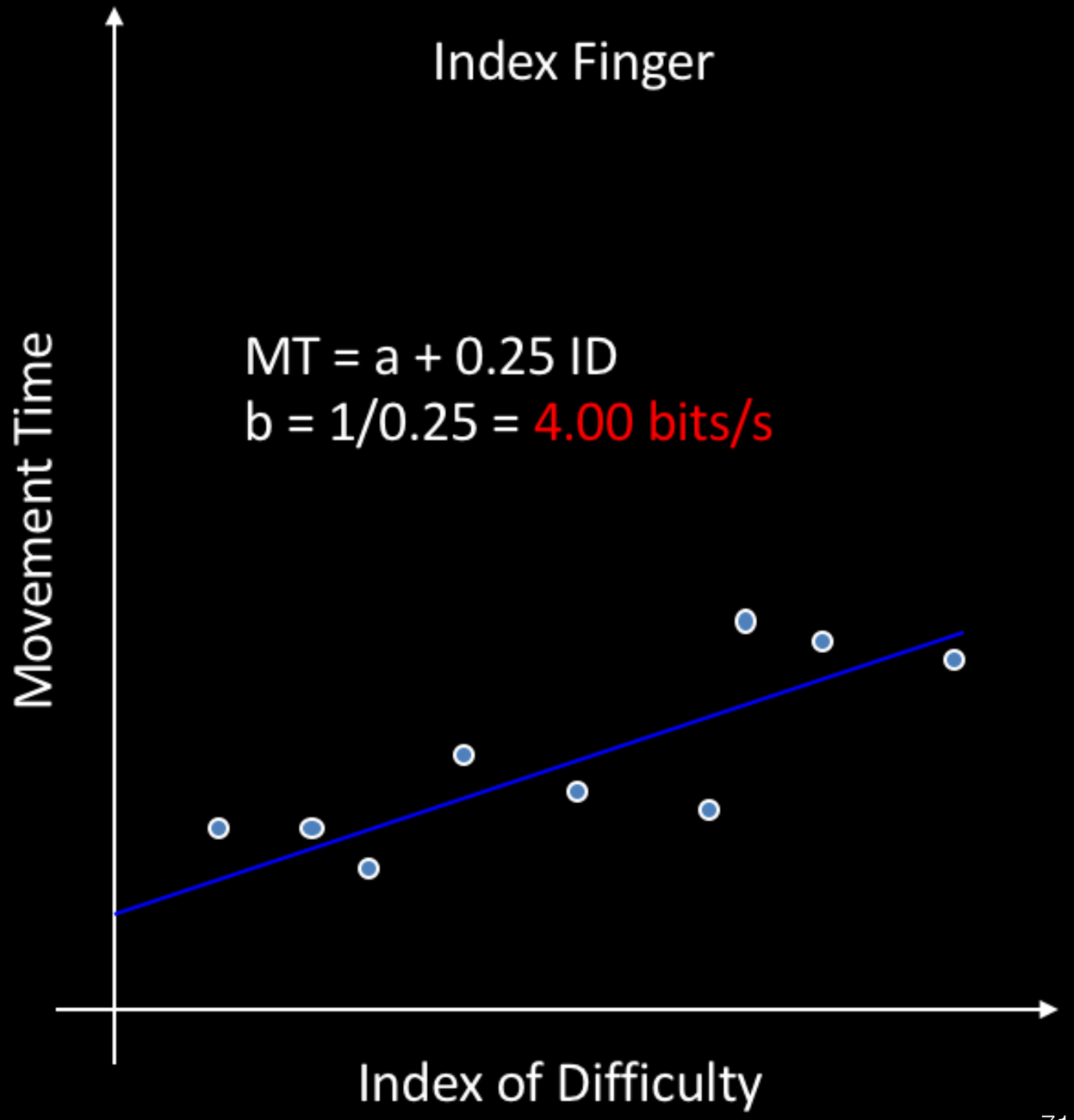
Fitts' Law

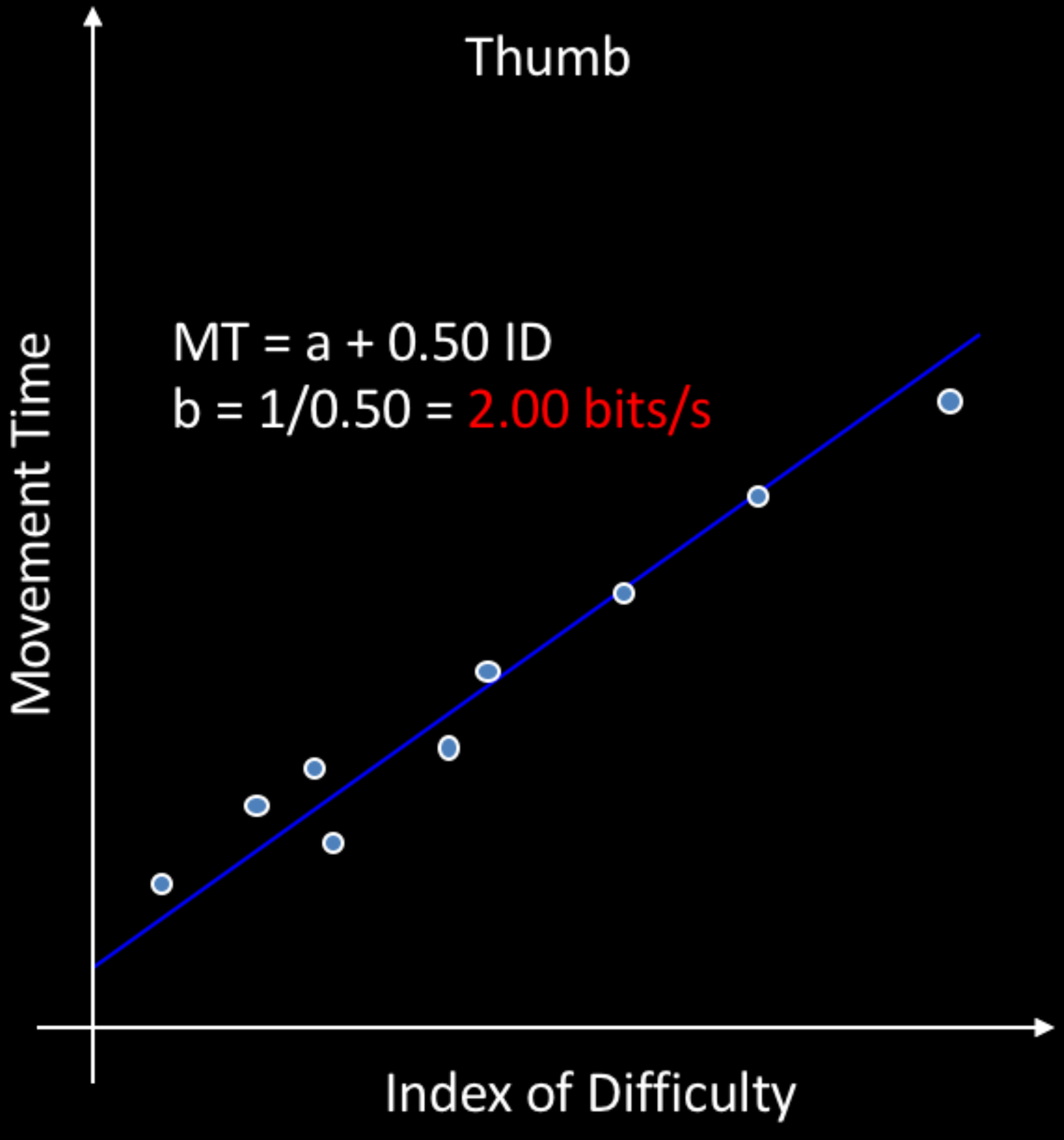
- Movement time = f(target size, target distance, efficiency of the pointing device)
- MT = a + b * log2(A / W + 1)
  - A = amplitude (aka distance of movement)
  - W = width (aka target size)

Index of difficulty

- MT = a + b**x**, where **x = log2(A / W + 1)**
  - **x** is the **Index of Difficulty**
  - It represents the difficulty of the pointing task
- **a** and **b** are constants
  - These represent performance characteristics of the pointing device itself! AKA index of performance

Which finger has better performance?





Fitts' Law: What to remember

- Index of difficulty (A and W) describes a pointing task, and is the same across the devices
- Index of performance (a and b) represents the efficiency of a given pointing device / context
- If y is MT and x is ID, IP is the slope!

So what?

- **A** and **W** give us a way to talk about pointing task difficulty
- **a** and **b** give us a way of comparing different pointing contexts
  - Different pointing devices (mouse vs. trackball)
  - User groups (older adults vs. kids)
  - Contexts (sitting vs. standing vs. walking)
  - **1/b** is often called **Index of Performance** (or throughput)

MacKenzie, I. S., Sellen, A., & Buxton, W. (1991). A comparison of input devices in elemental pointing and dragging tasks. *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems – CHI '91*, pp. 161-166. New York: ACM. [PDF] [software]

A Comparison of Input Devices in Elemental Pointing and Dragging Tasks

I. Scott MacKenzie[1], Abigail Sellen[2] and William Buxton[2]

[1]Ontario Institute for Studies in Education
University of Toronto
Toronto, Ontario, Canada M4S 1V6

[1]Seneca College of Applied Arts and Technology
Toronto, Ontario, Canada M2J 2X5

[2]Dynamic Graphics Project, Computer Systems Research Institute
University of Toronto
Toronto, Ontario Canada M5S 1A4

Abstract
An experiment is described comparing three devices (a mouse, a trackball, and a stylus with tablet) in the performance of pointing and dragging tasks. During pointing, movement times were shorter and error rates were lower than during dragging. It is shown that Fitts' law can model both tasks, and that within devices the index of performance is higher when pointing than when dragging. Device differences also appeared. The stylus displayed a higher rate of information processing than the mouse during pointing but not during dragging. The trackball ranked third for both tasks.

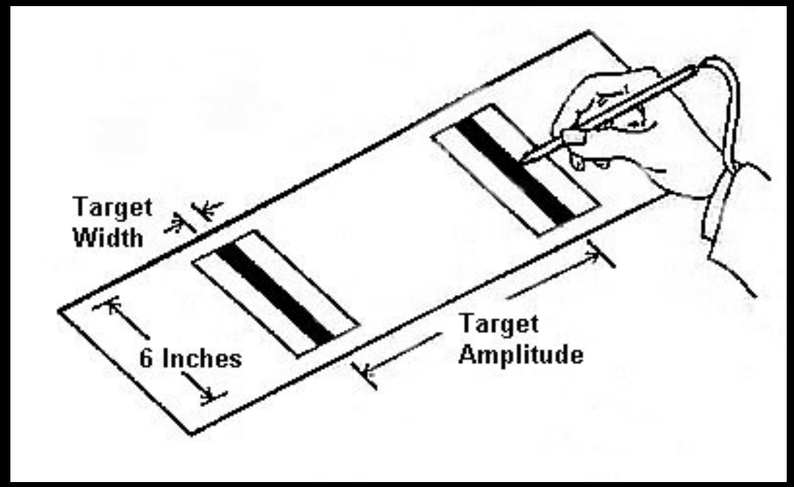
Keywords: Input devices, input tasks, performance modeling.

Using Fitts's Law

- We can predict how pointing performance would change on an easier or more difficult interface
- We can compare objective performance between devices, even if they perform differently in different contexts
  - e.g. Input device A is great at small targets but overall slow; input device B is pretty good at all target sizes - which is better?

Fitts's Law history

- Developed by Paul M. Fitts in 1954
- Designed for a **reciprocal pointing task**
  - Movement in 1 dimension only
  - Used a stylus on touch pads



Does Fitts's Law apply to me?

- It has been tested across a variety of user groups, body parts, input devices, even underwater
- It is a loose approximation that works in many contexts
- Useful as long as we can measure start time, end time, target distance, target size
- Applies to 1-dimensional movement only
  - Models for 2D and 3D movement have been proposed

Using Fitts's Law

1. If we know **a** and **b**
   - Predict movement times for different indices of difficulty (A and W)
2. If we don't know **a** and **b**
   - Run tests to collect movement times; solve for **a** and **b**

Running a Fitts's Law study

- When we want to calculate a and b
  - Use a variety of indices of difficulty
  - Collect movement times
  - Use regression to calculate **a** and **b**

Fitts's Law Software

- FittsStudy by Jake Wobbrock (Windows)
- GoFitts from Scott MacKenzie
- Interactive Fitts Visualization from Simon Weller

Limitations of Fitts's Law

- Does not tell us anything about error rate
  - Expects a fixed 4% error rate
  - Does not consider speed-accuracy tradeoffs
  - Other models focus on errors (e.g. Harada et al)
- Only 1D movement

Fitts’s Law is an approximation

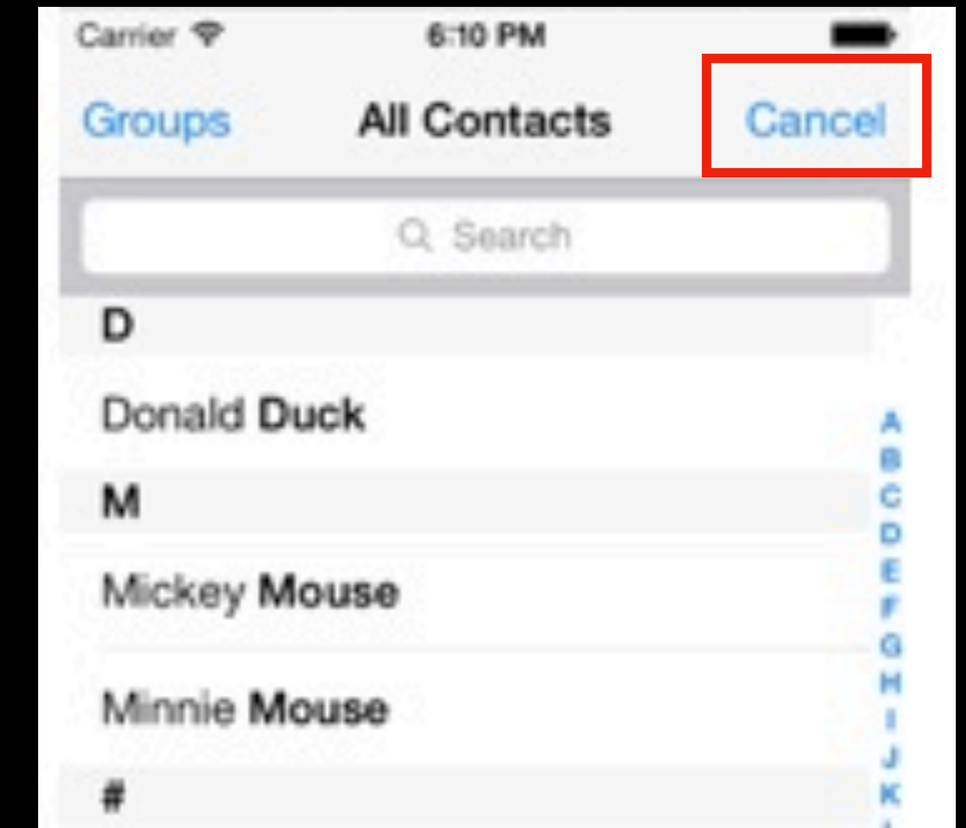
- Doesn’t actually model anything about the user
  - Individual dexterity, practice, fatigue
- Doesn’t model other aspects of pointing
  - Visual search, cognition

Interesting physiological problems

- Fat Finger problem - fingers are too large to select some targets, can occlude touch screen
- Gorilla arms - inability to keep arms raised
- Midas Touch problem - more on this later

Other trade-offs

- What is optimal for pointing may negatively affect readability, aesthetics, etc.
- Larger buttons may mean more screens to click through
- Hit area of target does not need to exactly match visual appearance



Pointing facilitation

Can we beat Fitts's Law?

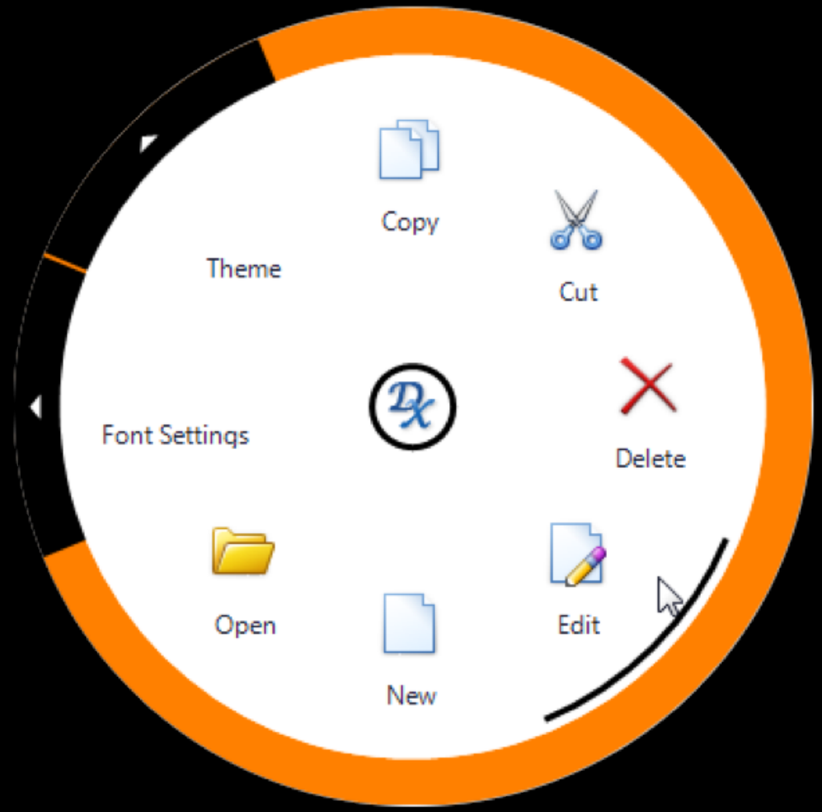
- How does Fitts's Law teach us how to improve performance?
  - Reduce (effective) distance to target
  - Increase (effective) size of target

Improving performance

- Pointing performance will be improved when targets are **nearby** and **large**
- So, design UIs such that important functions are nearby and large

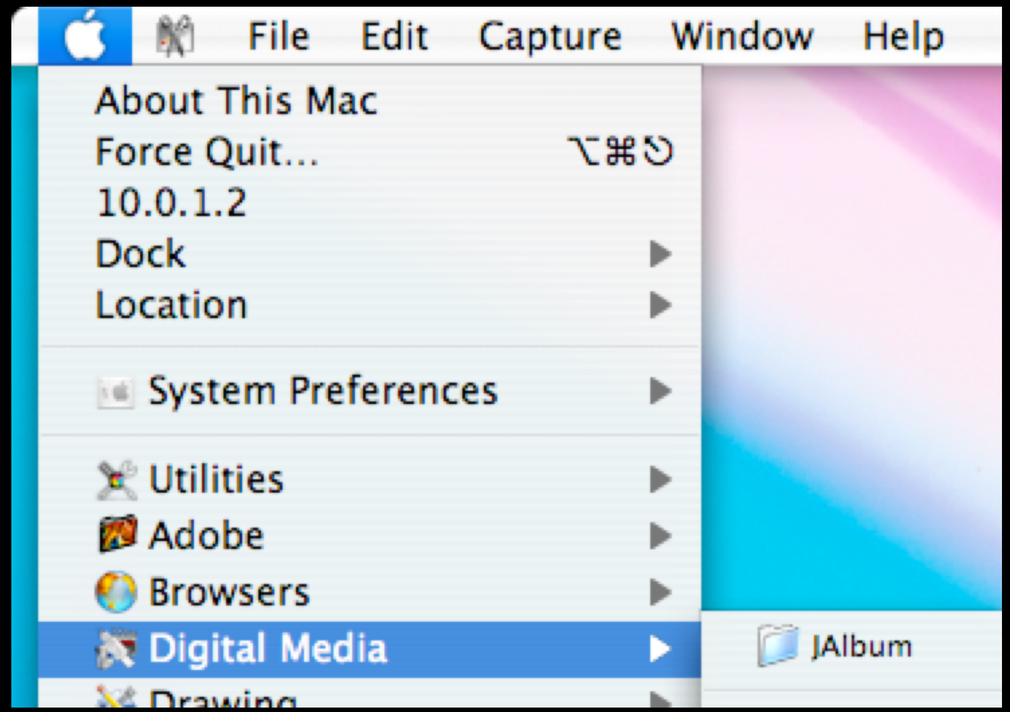
Radial menus

- Place menus around the cursor
- Often used in games and artistic software
- Limitations?



Edges and corners

- Edges and corners typically have infinite size
- No penalty for overshooting the target



Pointing facilitation

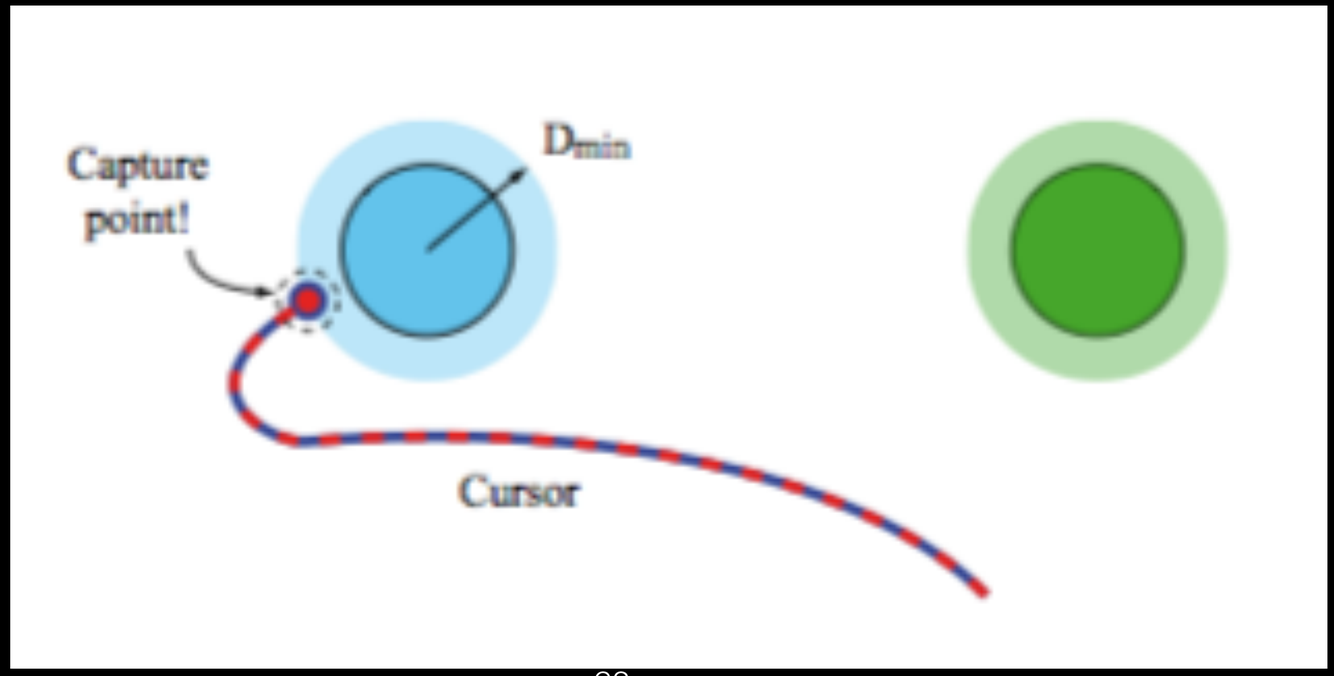
- Two fundamental approaches
- **Target aware** - the system knows where targets are on the screen
- **Target agnostic** - the system does not know where targets are on the screen

Why target agnostic?

- Sometimes the programmer does not have the ability to programmatically identify UI layout (e.g. old UI toolkits)
- Sometimes the number of targets is too large (e.g. I can click on any letter in this sentence)

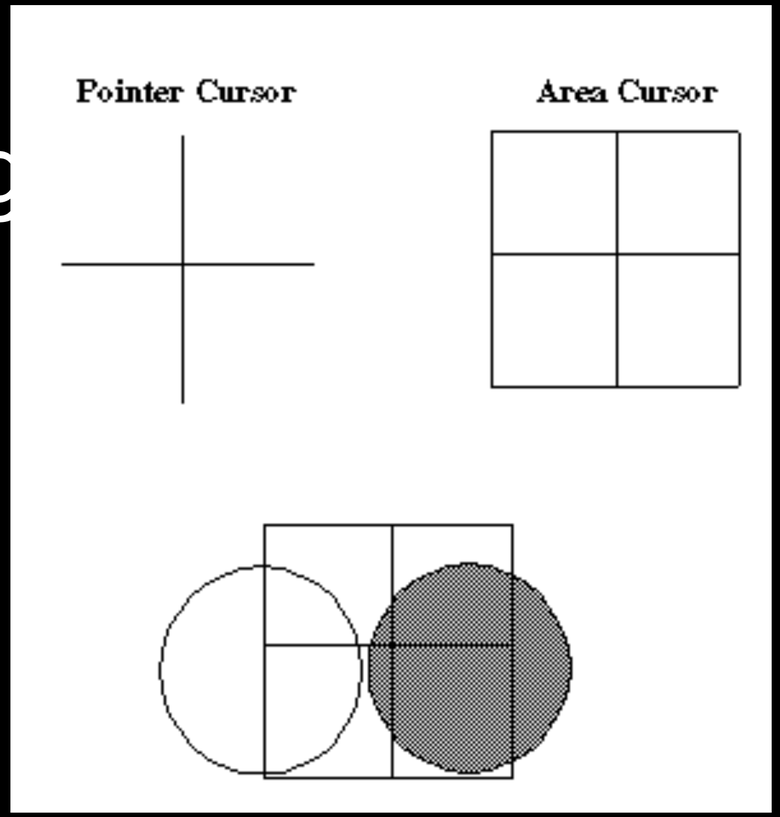
Facilitating movement

- Jump between icons or make “sticky” icons
- “Gravity fields”



Area Cursors

- One way to improve pointing is to
  size of the cursor: easier to hit targ
  bigger cursor
- Downside: this method can break
  targets are close to each other



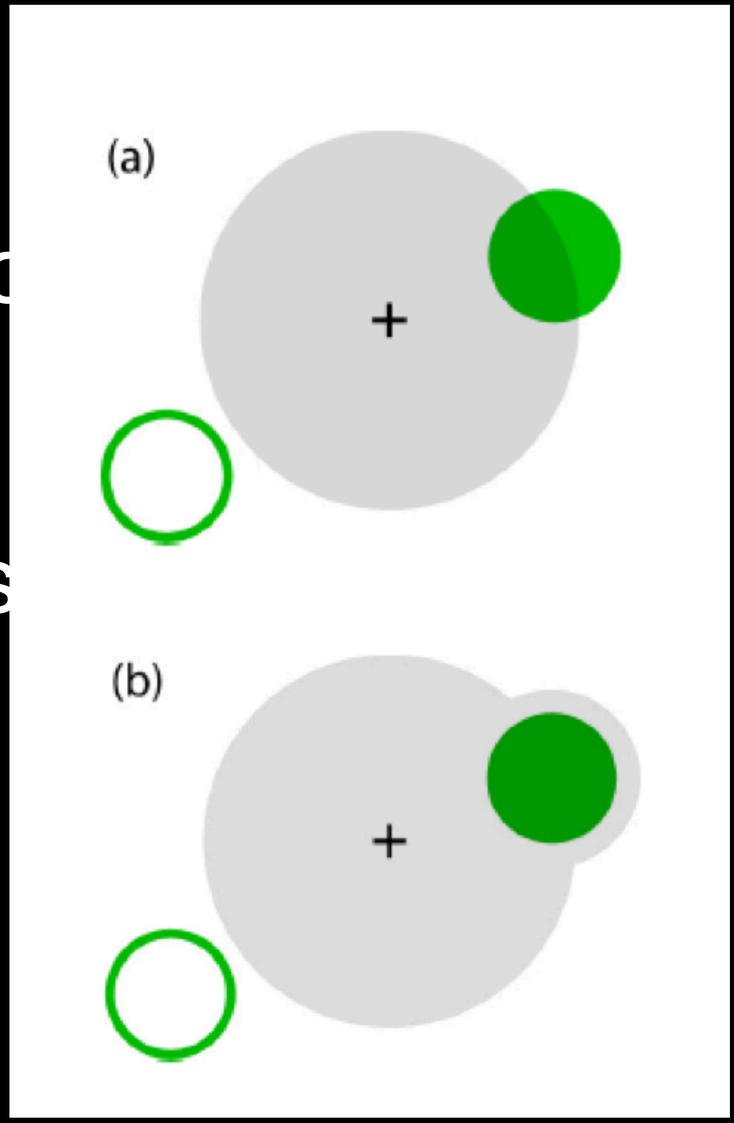
Limitation of area cursor

- What to do with densely packed targets?

Bubble Cursor

(Grossman and Balakrishnan 2005)

- Dynamically resize area cursor to c
  closest target, but not the second
- Considered by many to be the bes
  facilitation method



Target-agnostic methods

- Angle Mouse (Wobbrock et al.)
  - Detect when corrective movements start (based on speed and angular deviation)
  - Slow the cursor during correction phase

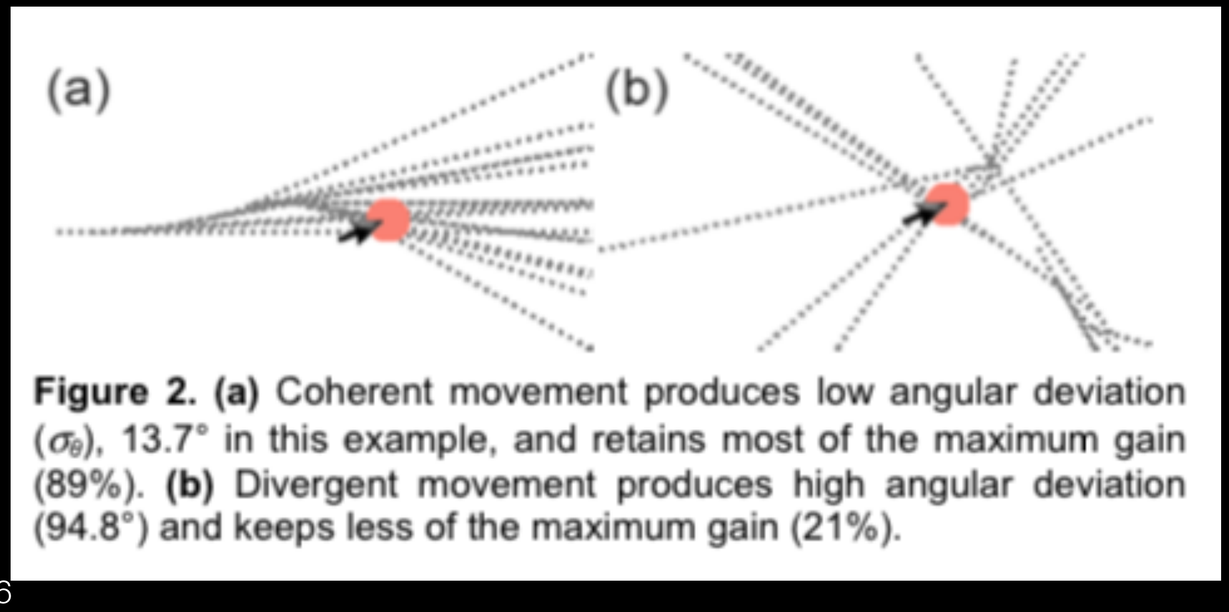


Figure 2. (a) Coherent movement produces low angular deviation (σ_θ), 13.7° in this example, and retains most of the maximum gain (89%). **(b)** Divergent movement produces high angular deviation (94.8°) and keeps less of the maximum gain (21%).

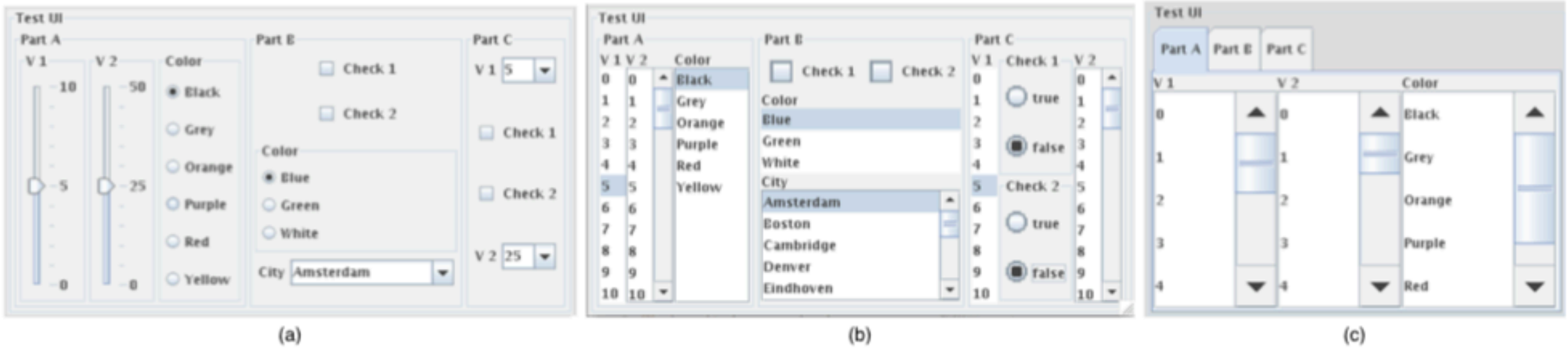
More target-agnostic methods

- Steady Clicks (Trewin et al.)
  - For pen-based interfaces
  - Some users, especially older users, accidentally “slip” the cursor while clicking
  - Detect this movement (based on characteristic shape) and find the target from a few seconds before

User-specific models

- Some individuals may have additional, atypical challenges related to pointing
- Examples
  - Parkinson’s Disease may cause tremor when using a pointing device
  - Muscular Dystrophy may impair large movements, but doesn’t affect precision
- Can we build user-specific models of pointing?

Supple++ (Gajos et al.)



(a) (b) (c)

Figure 5: Three renderings of a synthetic interface used in the preliminary study and automatically generated under the same size constraint: (a) base line preference-optimized GUI; (b) personalized for the mouse user with muscular dystrophy (M03); (c) personalized for the eye tracker user (ET01). GUIs (b) and (c) were generated by SUPPLE++.

Mini-design activity

- Consider our phone dialer user interface
- Sketch out a variant for
  - User with tremor
  - User with hand fatigue
  - User with low vision (but fine motor ability)

For next time

- We’ll test Fitts Law
- Come up with a hypothesis to test today