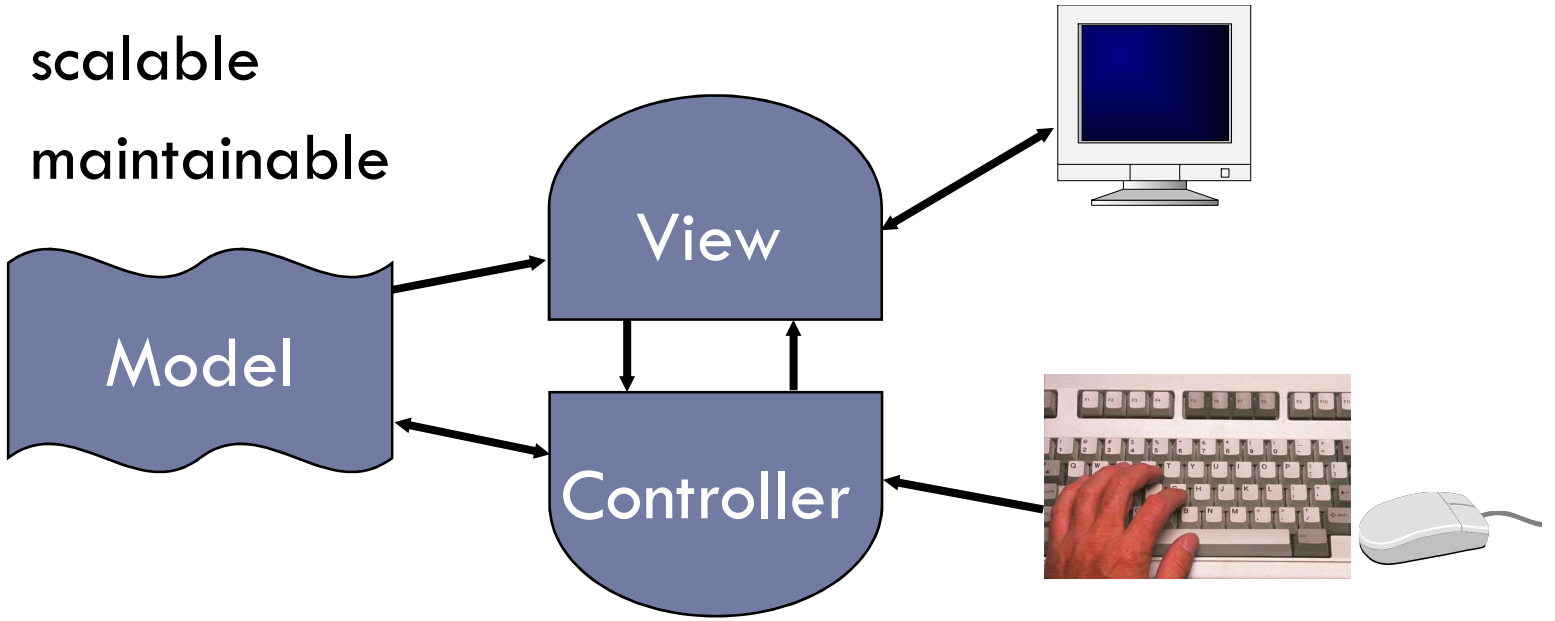
CONTROLLER STATE MACHINES

CMPT 381

Model-View-Controller

- An architecture for interactive applications
  - introduced by Smalltalk developers at PARC
- Partitions application so that it is:
  - scalable
  - maintainable



Overview

Touch interaction is overloaded

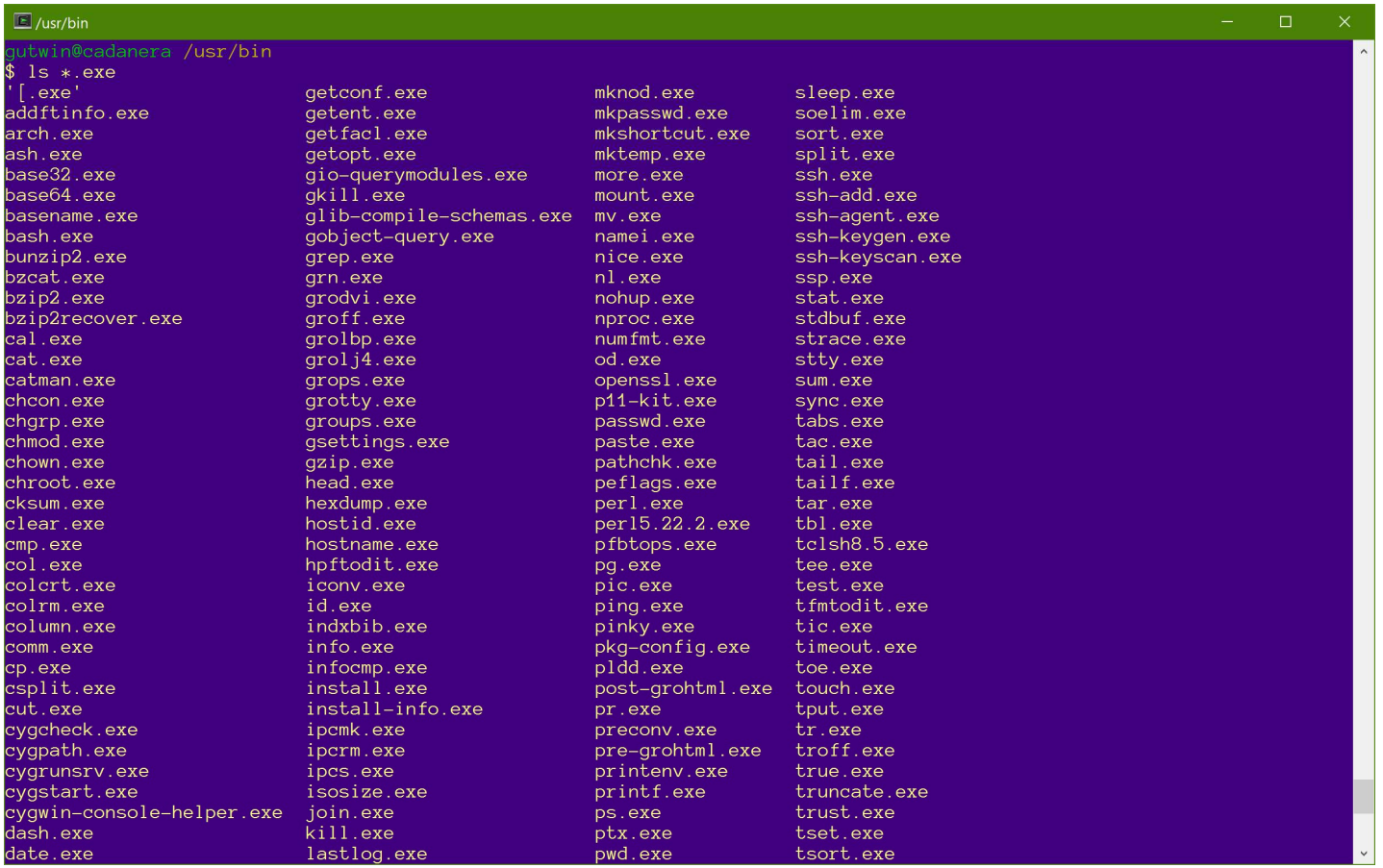
State machines

Implementation: switch statement

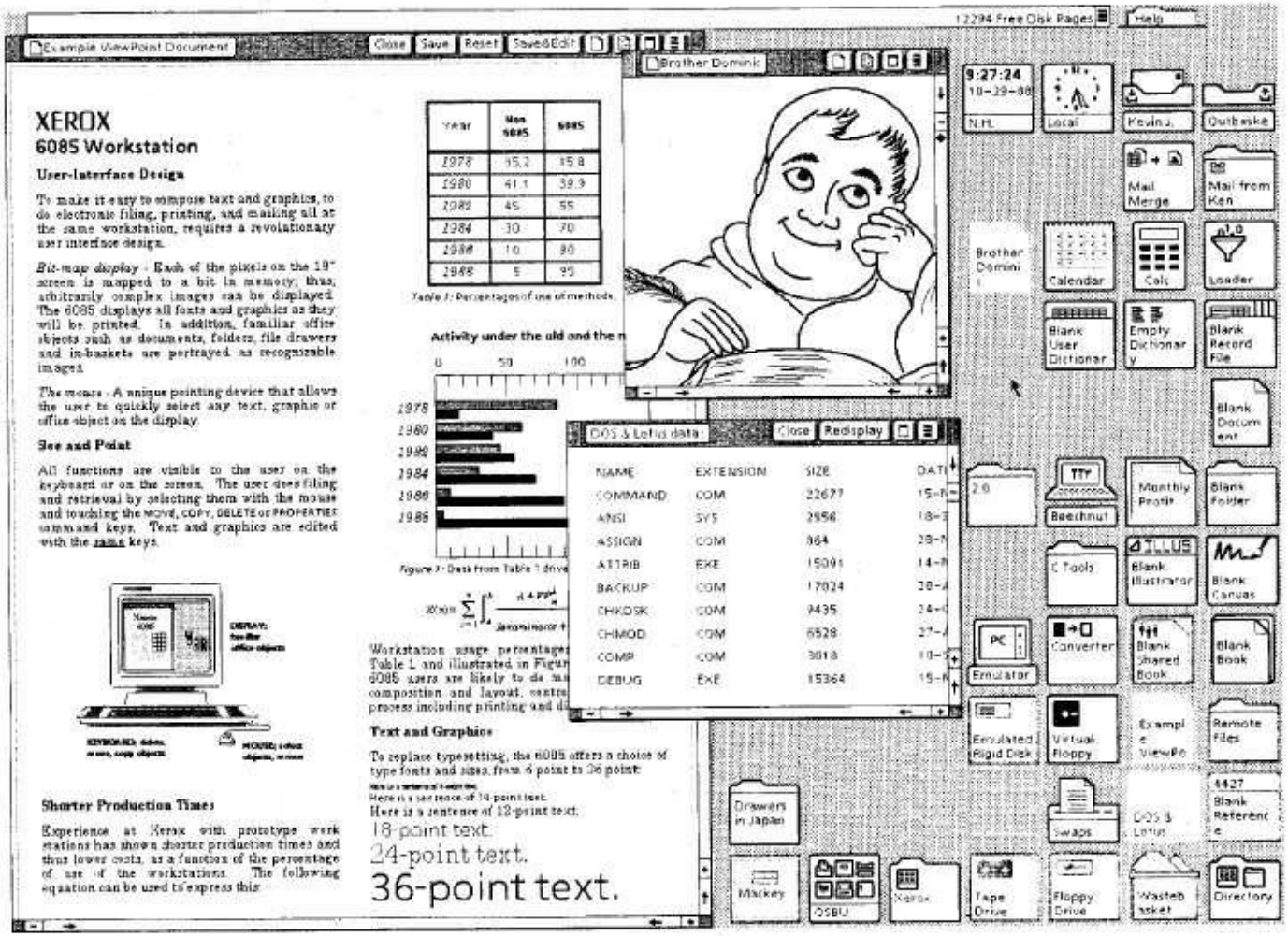
Implementation: State pattern

Interaction: verbs vs. nouns

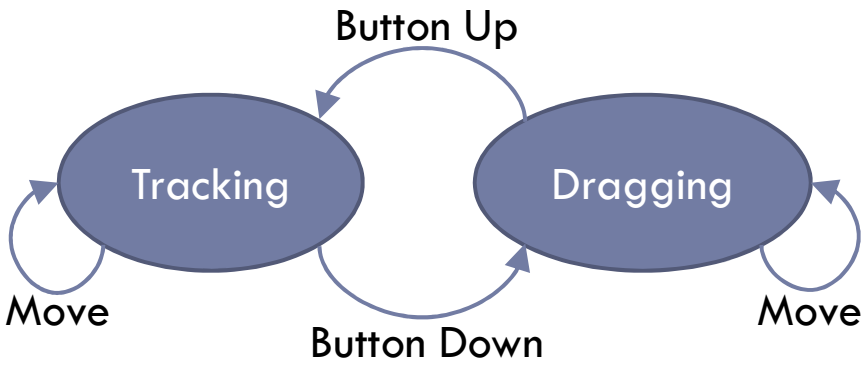
Command languages: verbs



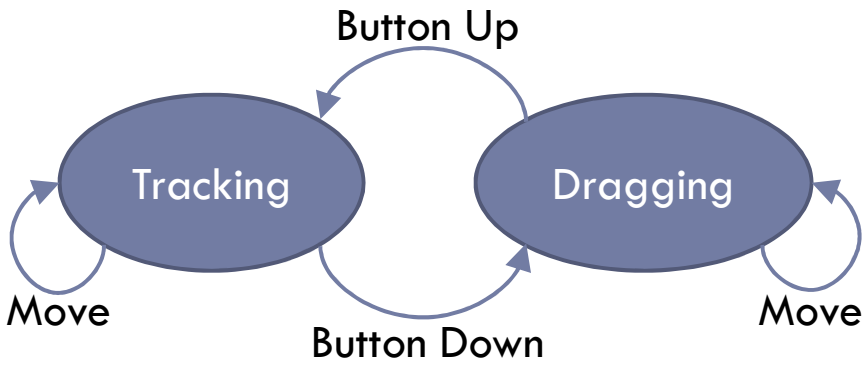
Direct manipulation: nouns



Mouse state model ("mouse verbs")

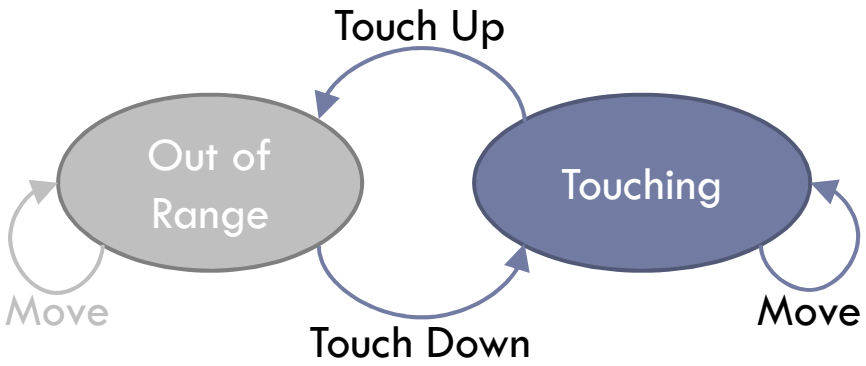


Mouse state model ("mouse verbs")



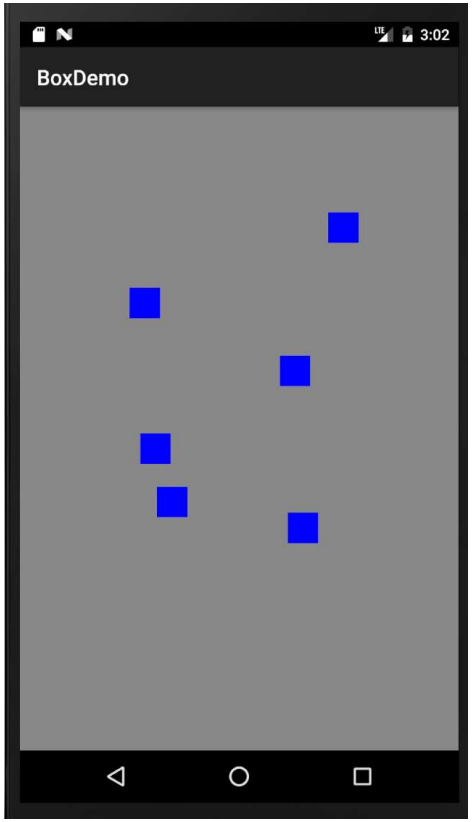
These verbs are heavily *overloaded* in direct-manipulation systems: one action can mean many things, depending on context

Touchscreen state model



BoxDemo touch interaction

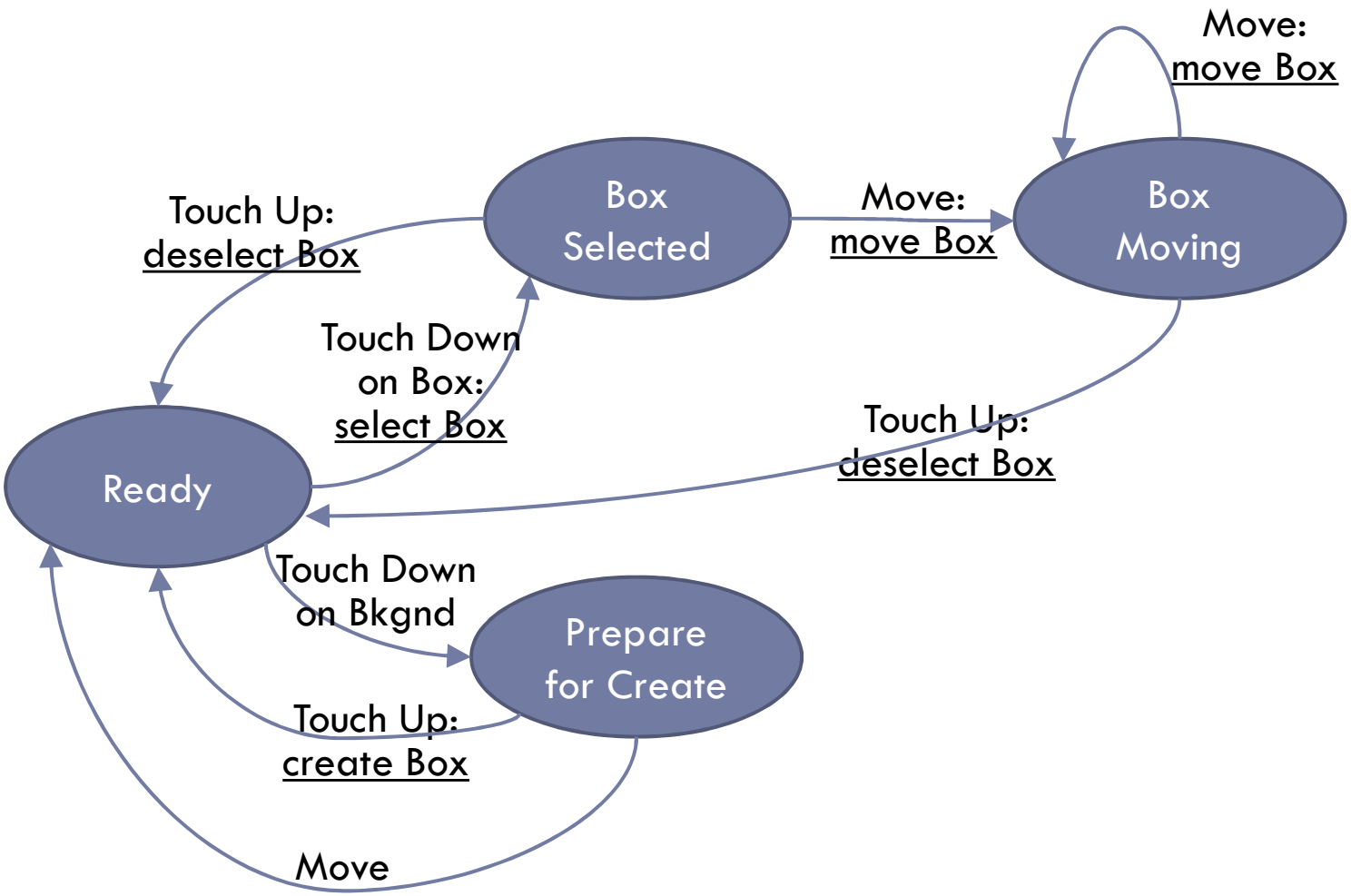
- Touch Down action should:
  - select box OR start create
- Touch Up action should:
  - create box OR finish drag OR nothing
- Move action should:
  - drag box OR cancel create



- How to keep track of all this?

BoxDemo touch interaction

BoxDemo interaction state model

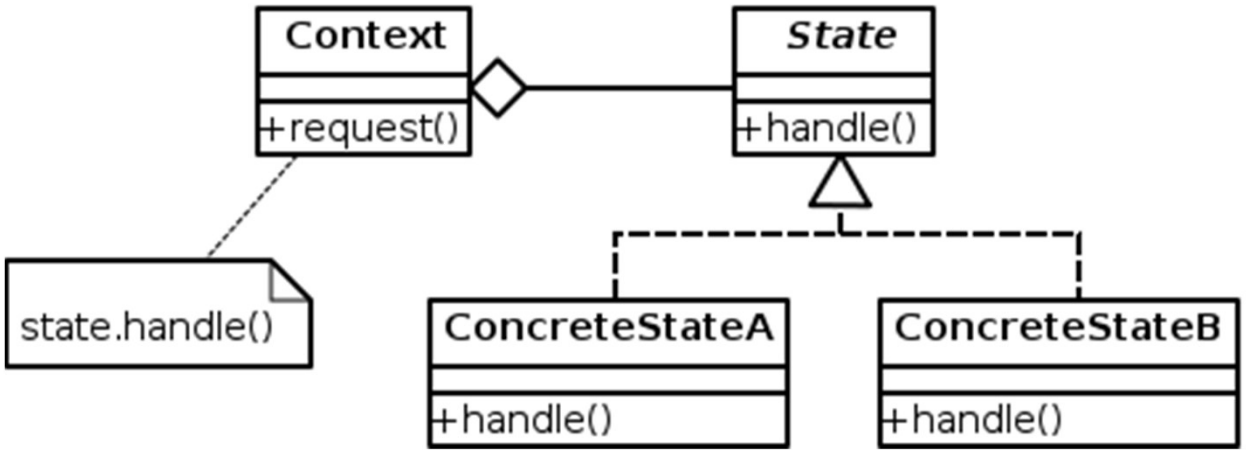


Implementation: switch statement

```
int state;
final int STATE_READY = 0;
final int STATE_PREPARE_CREATE = 1;
final int STATE_SELECTED = 2;
final int STATE_MOVING = 3;

public boolean onTouch(View v, MotionEvent event) {
  switch (state) {
    case STATE_READY:
      switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
          // is the touch on box or background
                        . . .
```

Implementation: State pattern



Implementation: State pattern

```
public class InputReadyState implements InputState{

  public void handleTouch(BoxViewController context, MotionEvent event) {
    switch (event.getAction()) {
      case MotionEvent.ACTION_DOWN:
        // if on a box, select
        if (context.model.contains(event.getX(),event.getY())) {
          context.selected = context.model.findClick(event.getX(),event.getY());
          context.view.invalidate();
          context.bState = new InputSelectedState();
        } else {
          // on background, so prepare for create
          context.bState = new InputPrepareState();
        }
        break;
    }
  }
}
```

Event / Context / Side effects