

# Designing Interactive Systems I

## Interaction Design Notations

Prof. Dr. Jan Borchers  
Media Computing Group  
RWTH Aachen University

Winter Semester '24/'25

<https://hci.rwth-aachen.de/dis>



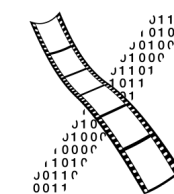
**RWTHAACHEN**  
UNIVERSITY

# Review

- Ten Golden Rules of Interface Design
- Human Deadlines
- Latency & Responsiveness

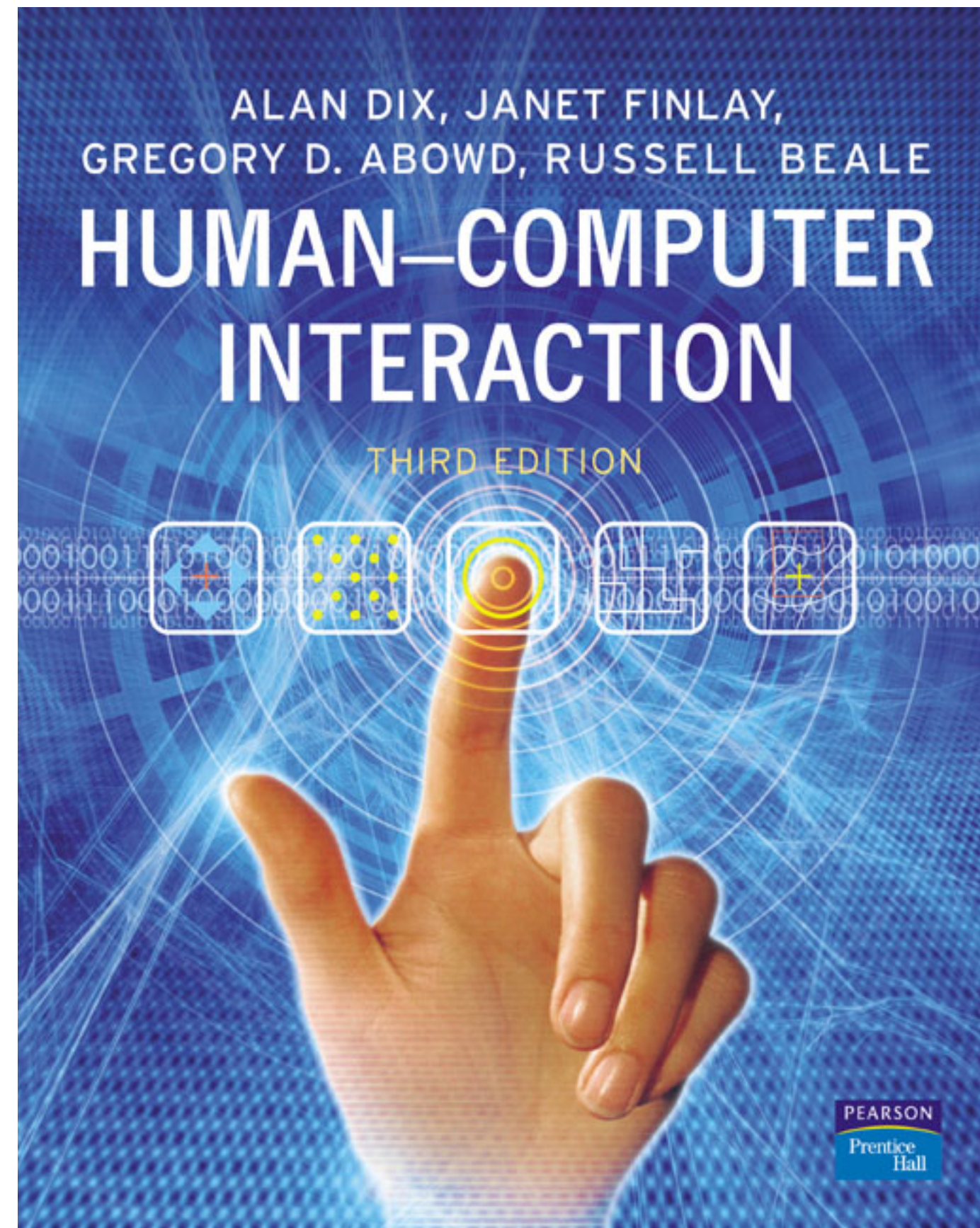


# Interaction Design Notations



# Interaction Design Notations

- Alan Dix et al.: **Human-Computer Interaction**, 3rd ed. (2003), Chapter 16





# Back to BASICS

```
Print "Please enter a number"  
INPUT n  
Print "The square of",n,"is",n*n
```

- What are the problems with using such a notation to specify a dialog?



# Why UI Specification Languages

- In normal programming languages, UI and algorithms are mixed up
- System and user decisions are hard to distinguish
- Error checking on inputs dominates and complicates code
- First step: bundling I/O in classes/procedures
- Second step: Use a more efficient, readable language to specify the dialog
  - A priori to design the dialog
  - As part of the implementation (executable spec.)



# Specifying User Interfaces

- Problem: Describe the proposed design of a user interface
- Approach: natural/semi-formal/formal languages
- Many standard computer science techniques apply
- The more modern the UI, the harder to describe textually, depending on modality and UI style



# Grammars

- Mostly BNF-like

```
expr ::= empty | atom expr | '(' expr ')' expr
```

- E.g., Shneiderman's multiparty grammar

```
<Session>      ::= <U: Opening> <C: Responding>  
<U: Opening>   ::= LOGIN <U: Name>  
<U: Name>      ::= <U: string>  
<C: Responding> ::= HELLO [<U: Name>]
```

- Great for command-line UIs, e.g., banking ATMs, Unix commands
- Less suitable for GUIs



# Grammars

- Regular expressions
  - `select-line click click* double-click`

- E.g., Unix “copy” command synopsis:

```
cp [-R [-H | -L | -P]] [-f | -i | -n] [-pv] source_file target_file
cp [-R [-H | -L | -P]] [-f | -i | -n] [-pv] source_file ... target_dir
```

recursion policies

overwrite policies

- Short and precise, but hard to read, requires additional information about semantics

# Production Rules

- Unordered list of rules: `if condition then action`
  - Condition based on state or pending events
  - Every rule always potentially active
- Good for concurrency
- Bad for sequence



# Event-based Production Rules

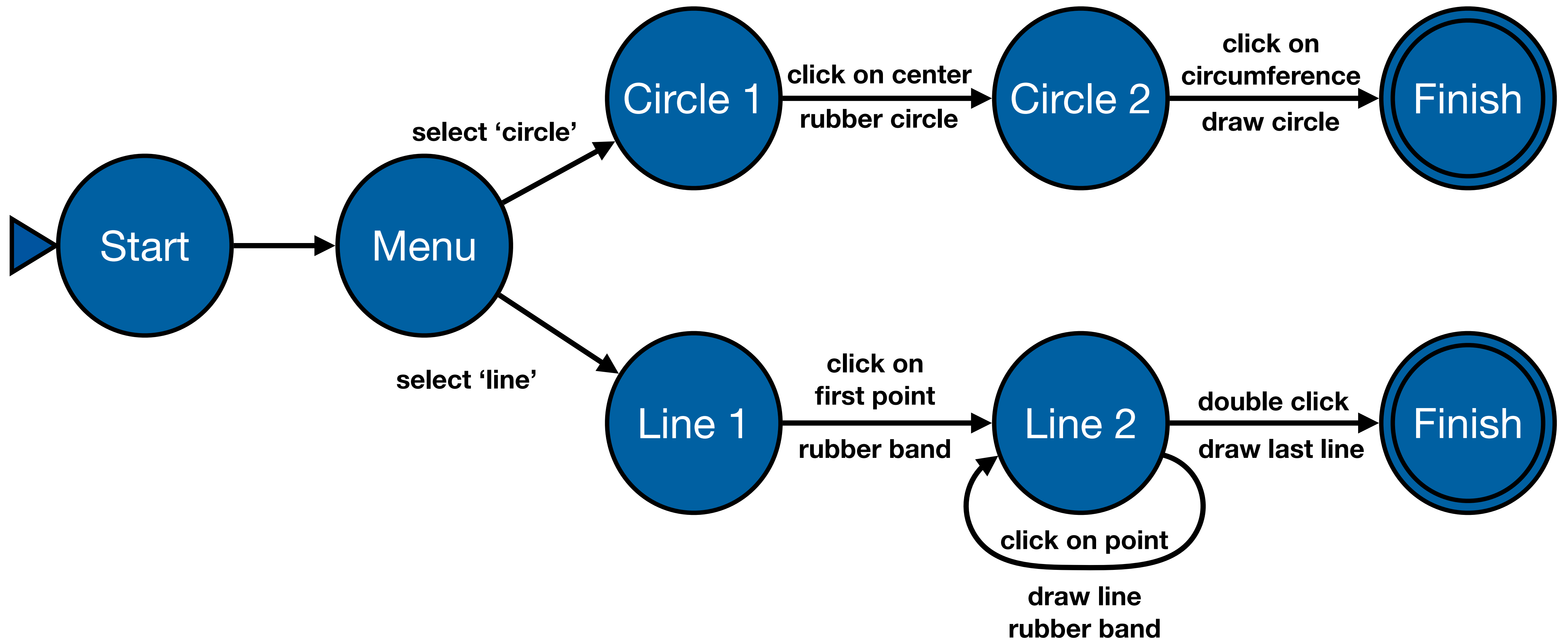
```
select-line → first  
click first → rest  
click rest → rest  
double-click rest → <draw line>
```

- Note:
  - Events added to list of pending events
  - ‘first’ and ‘rest’ are internally generated events
- Bad at state!

# Graph Notations: STNs

- State Transition Networks (STNs)
  - Most common tool to specify dialogs
  - Established format (since 1960s)
- Consisting of:
  - **States** (usually the system waiting for some user action)
  - **Transitions** (which have a **user action** and a **system response** associated with them)
- Describes **sequences** of user actions and system responses

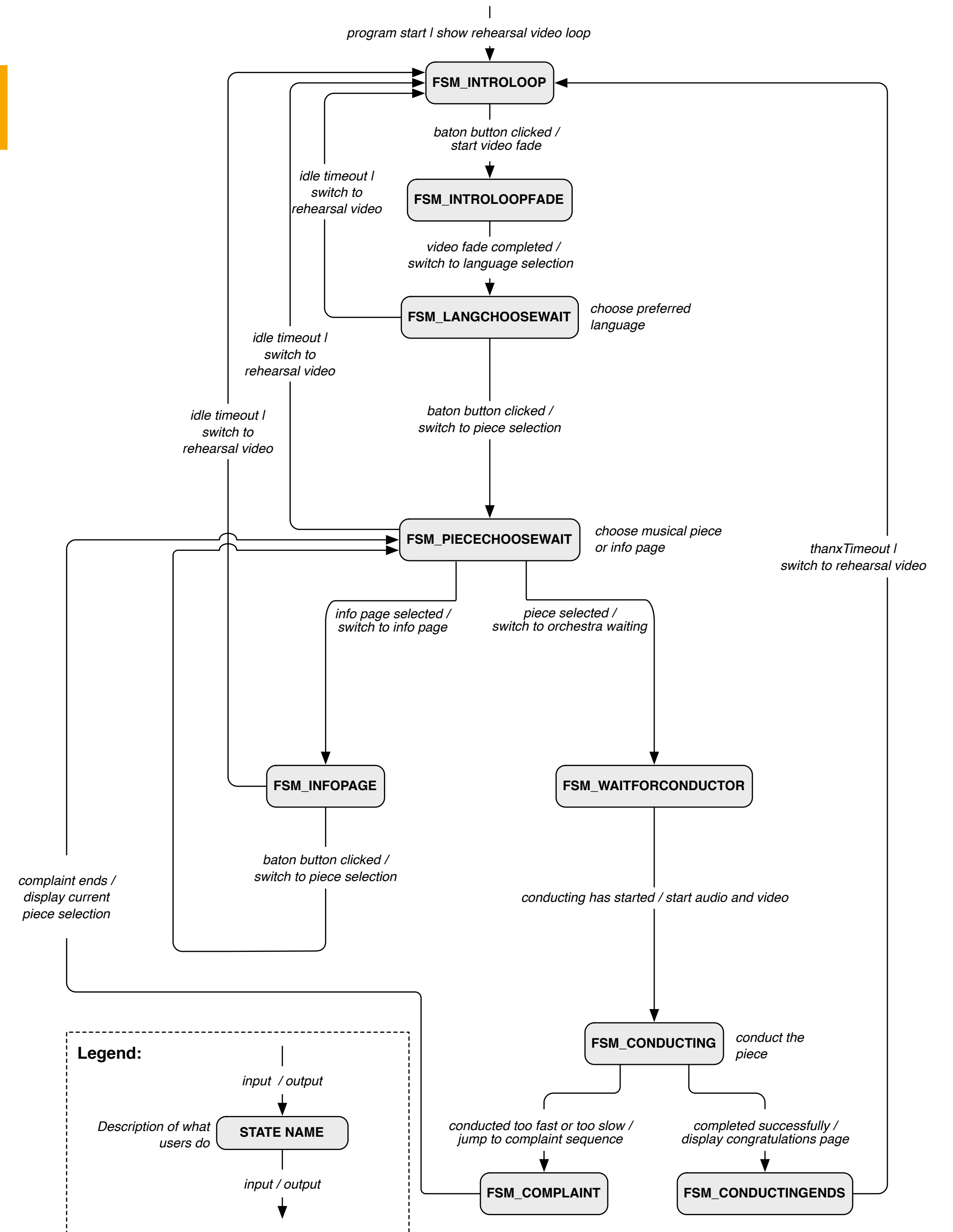
# Graph Notations: STNs

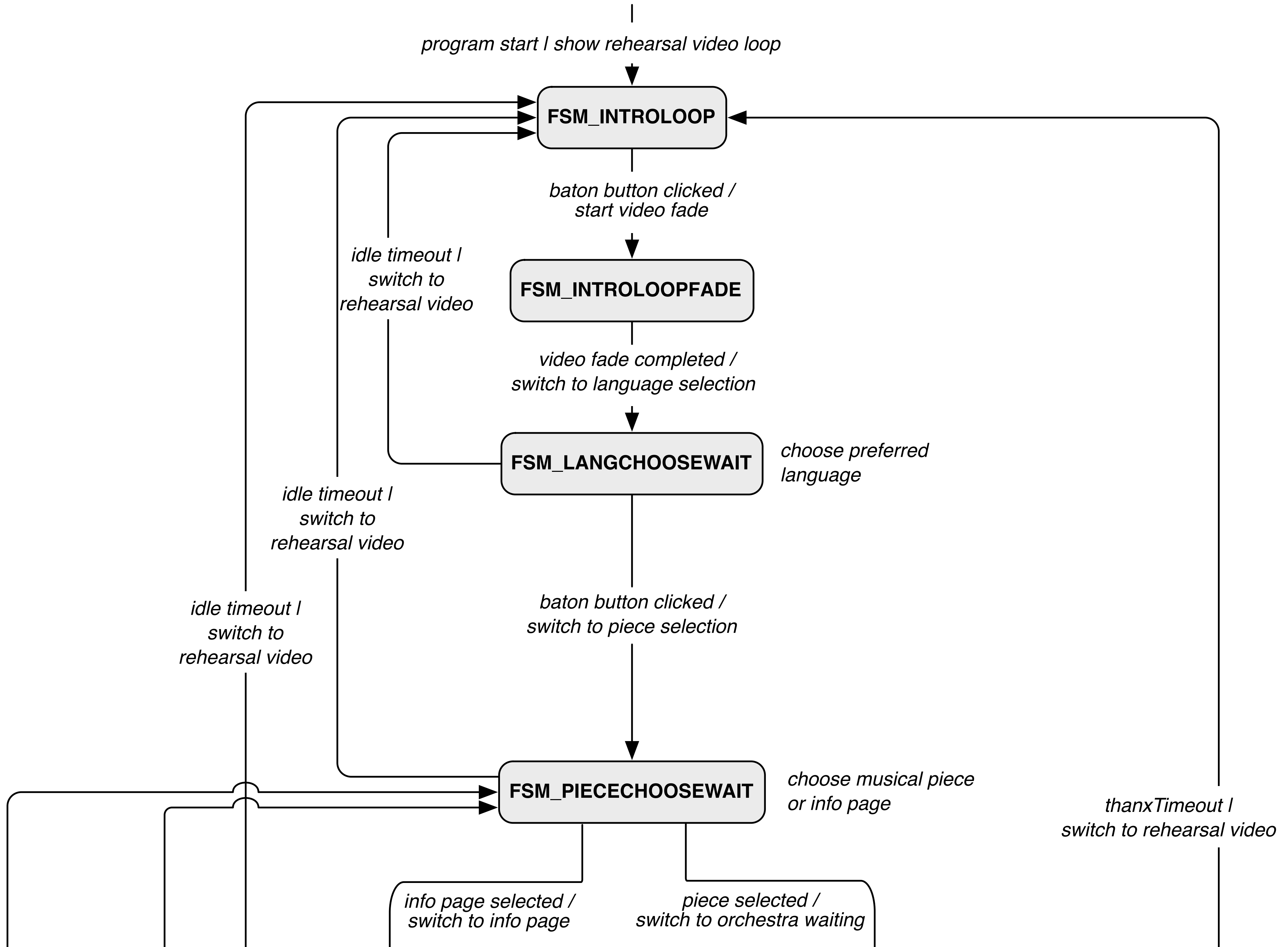






# Example: STN for Personal Orchestra Dialog





program start / show rehearsal video loop

**FSM\_INTROLOOP**

baton button clicked / start video fade

**FSM\_INTROLOOPFADE**

video fade completed / switch to language selection

**FSM\_LANGCHOOSEWAIT**

choose preferred language

baton button clicked / switch to piece selection

**FSM\_PIECECHOOSEWAIT**

choose musical piece or info page

info page selected / switch to info page

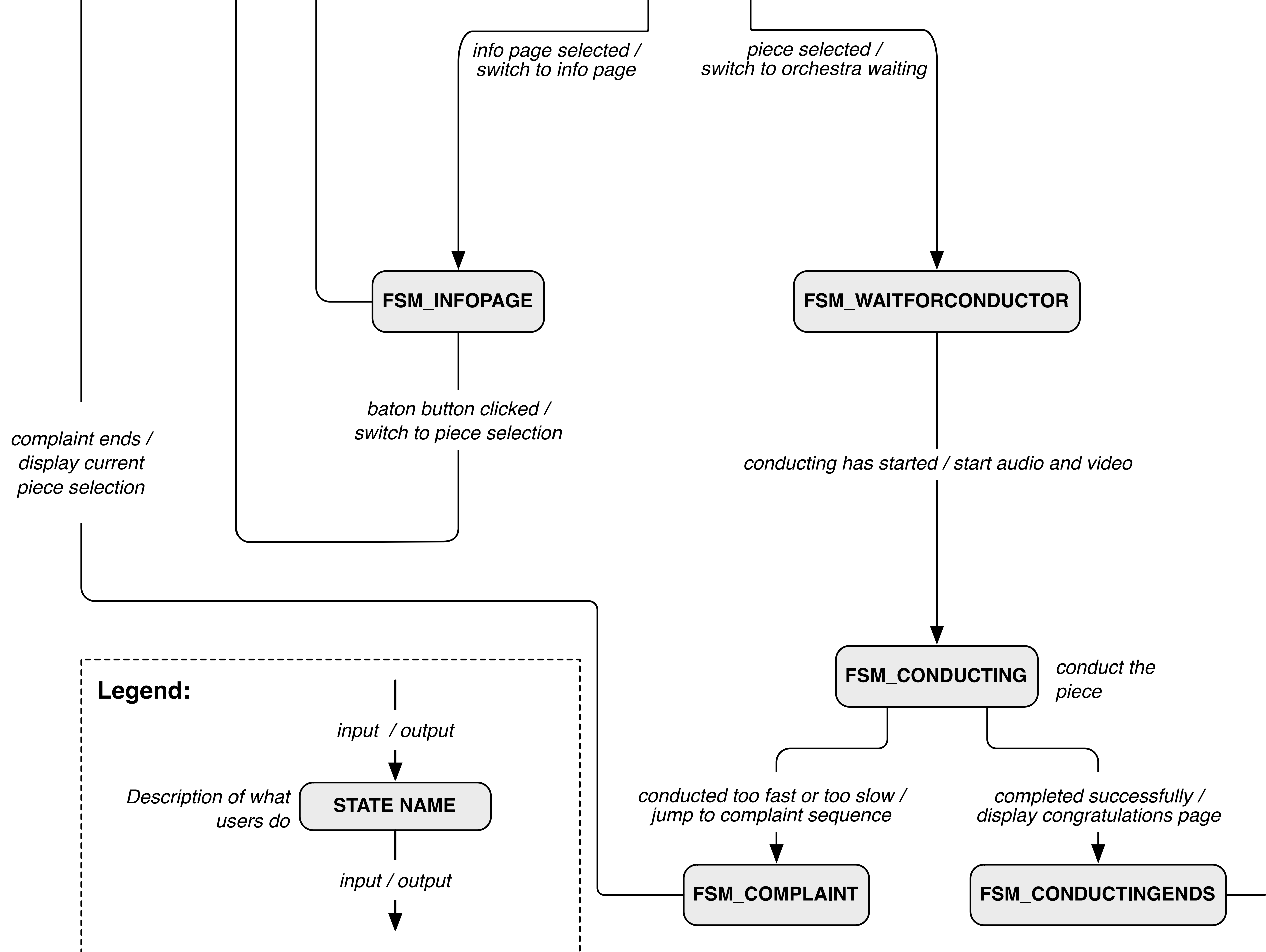
piece selected / switch to orchestra waiting

thanxTimeout / switch to rehearsal video

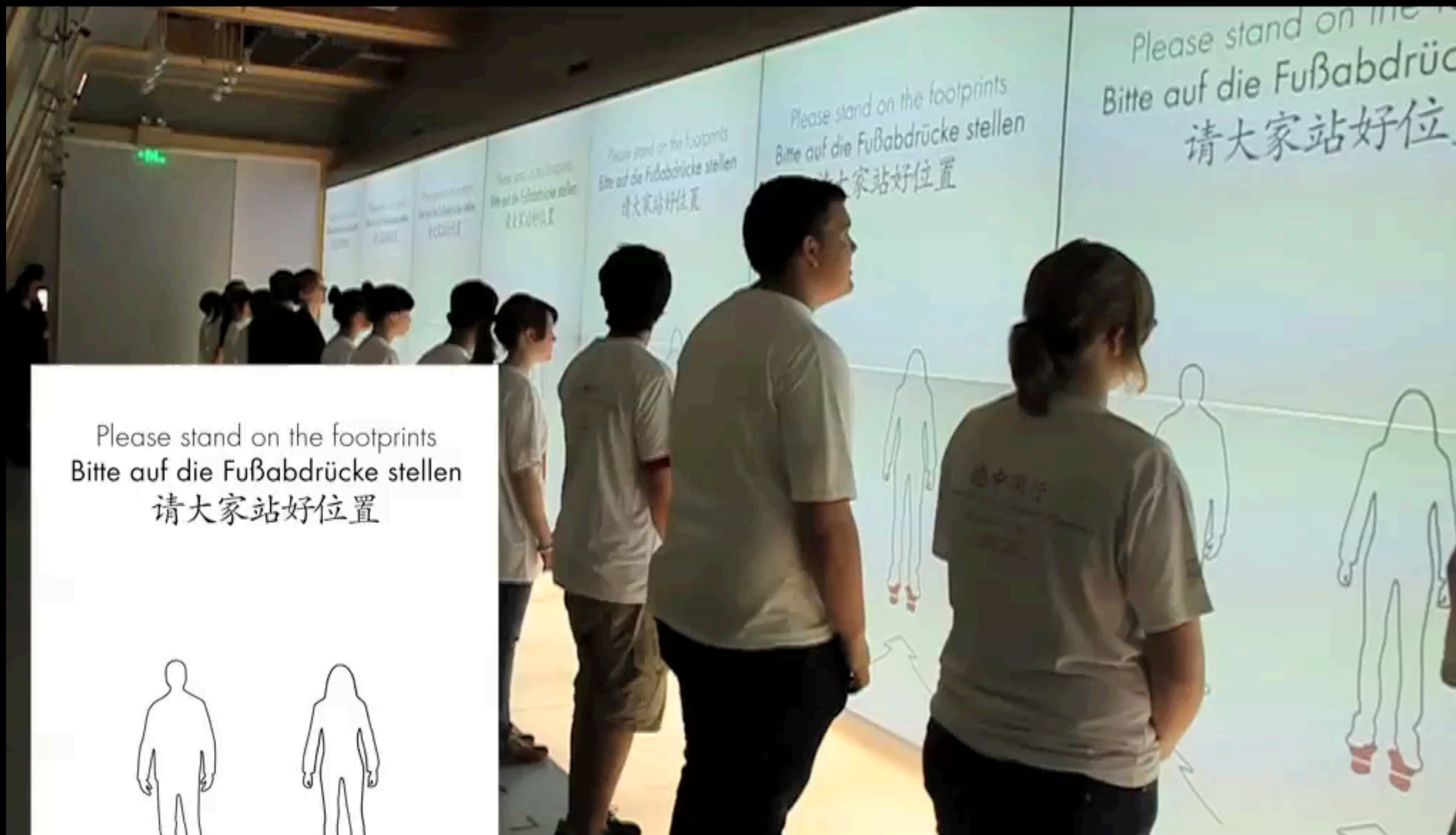
idle timeout / switch to rehearsal video

idle timeout / switch to rehearsal video

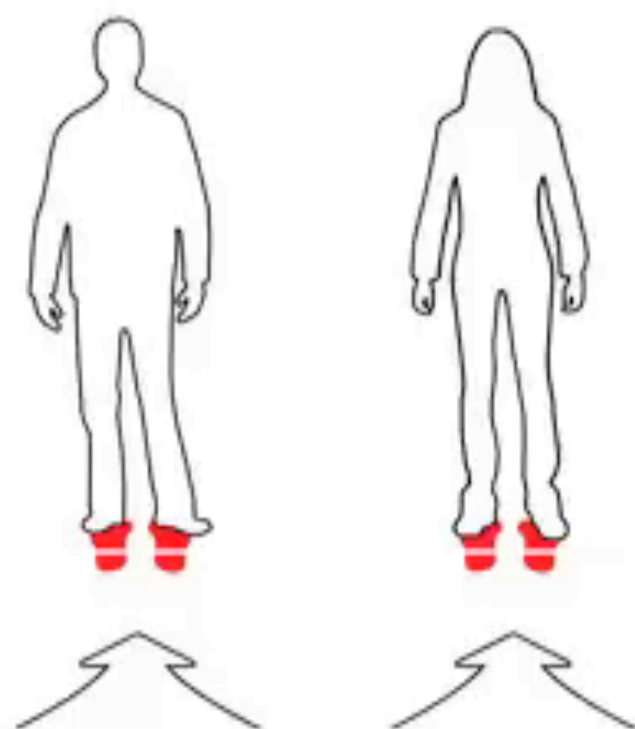
idle timeout / switch to rehearsal video





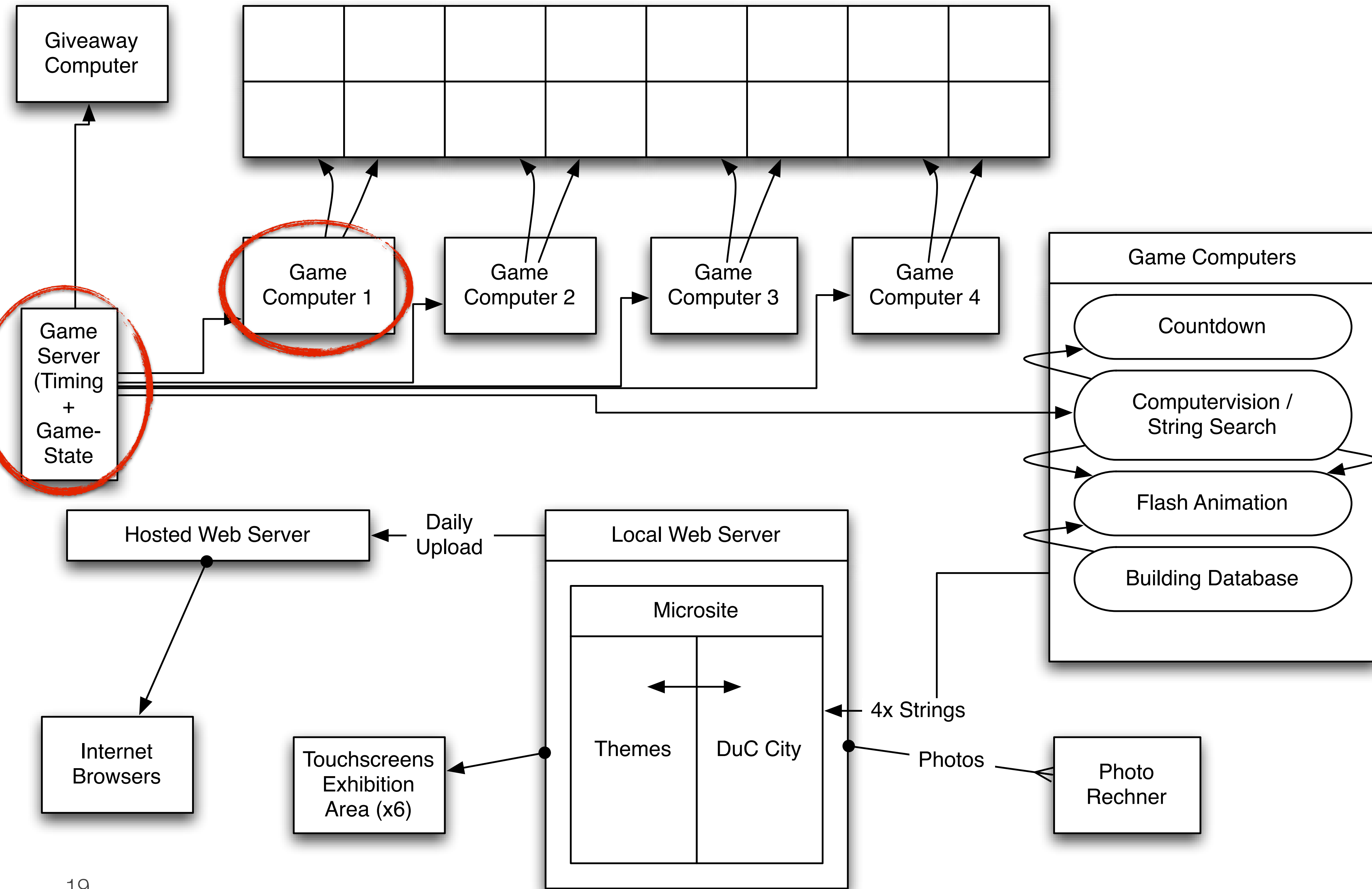


Please stand on the footprints  
Bitte auf die Fußabdrücke stellen  
请大家站好位置





# 16 Screens (Game Area)

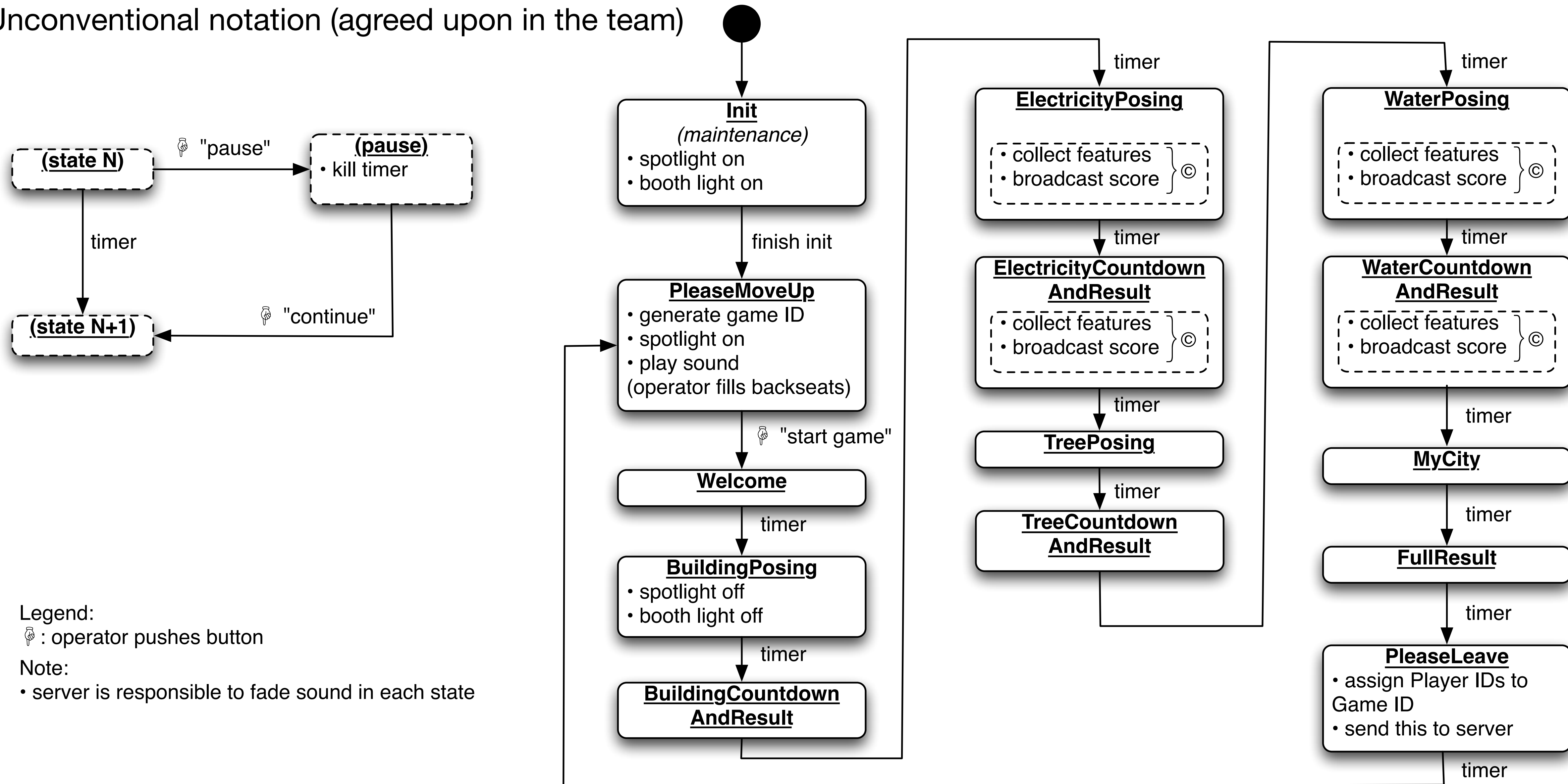


# Silhouettes Components

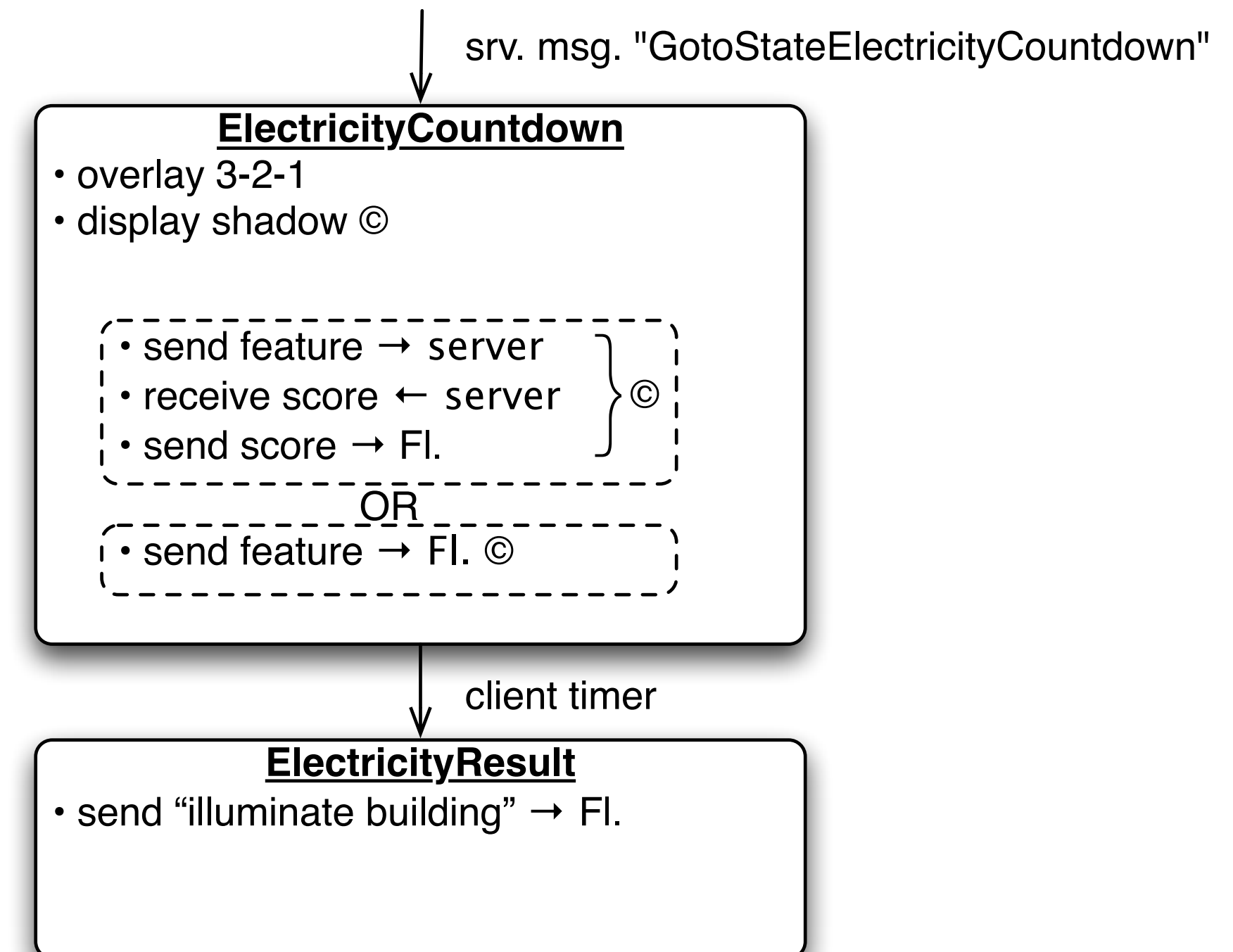
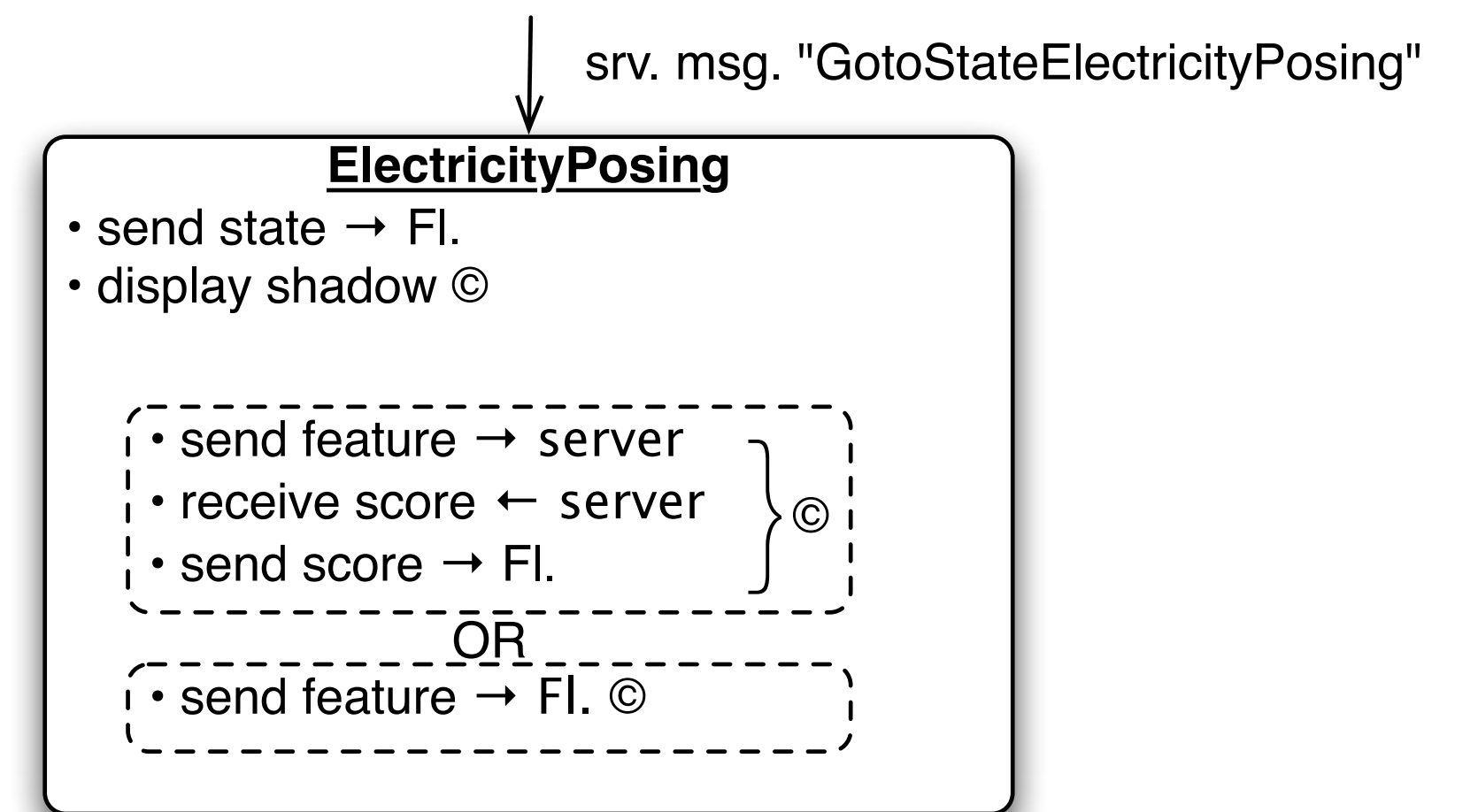
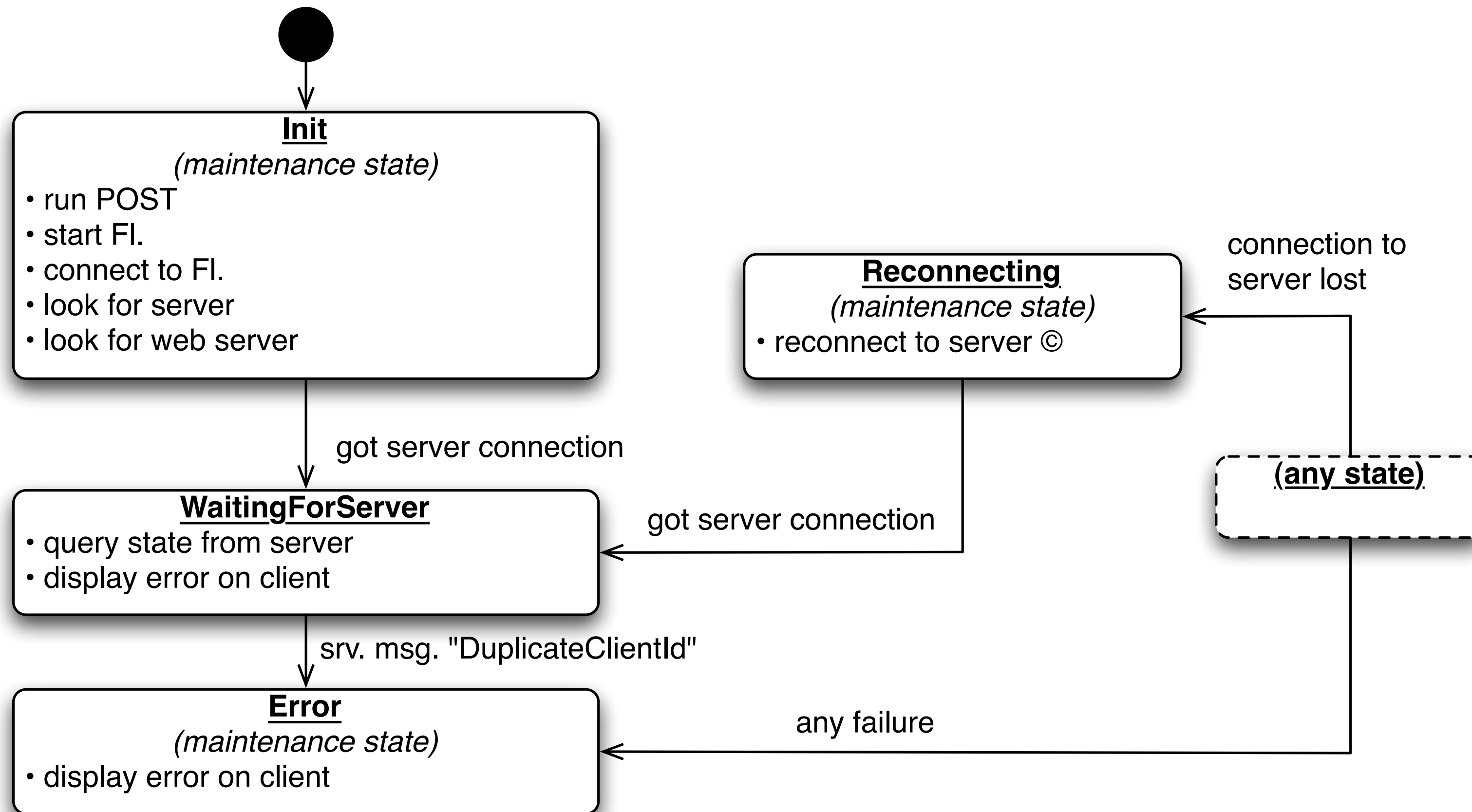
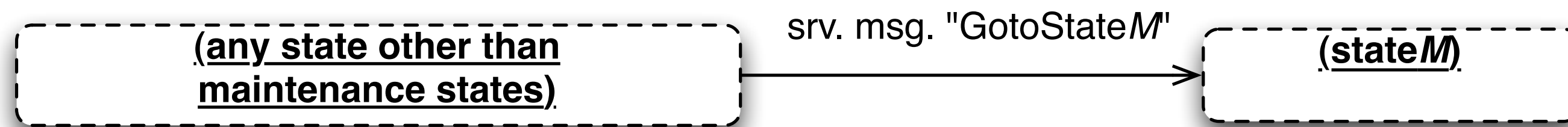


# Silhouettes: Server STN

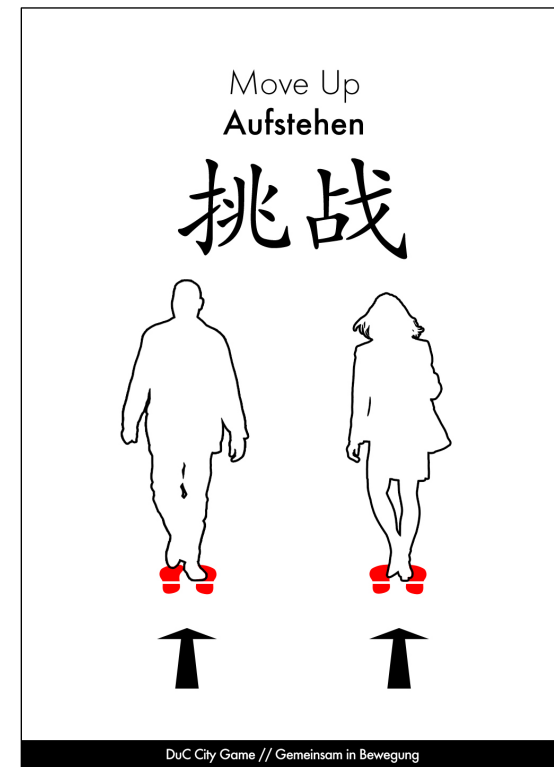
- Unconventional notation (agreed upon in the team)



# Silhouettes: Client STN



Visuals	Description	Server State/Action	Client State/Light
---------	-------------	---------------------	--------------------



Waiting people will be asked to move up to the front and take their positions on the indicated spots.

**PleaseMoveUp**

- generate game ID
- spotlight on
- play sound (operator fill backseats)

DMX msg.  
- fade **in** spots on indicated footprints on the floor

srv. msg.  
"GotoStatePleaseMoveUp"

**PleaseMoveUp**

- send state → Fl.

"start game"  
by button push



DMX msg.  
- fade **out** spots on indicated foot prints on the floor  
- fade **in** light in the waiting booths

**Welcome**

srv. msg.  
"GotoStateWelcome"

**Welcome**

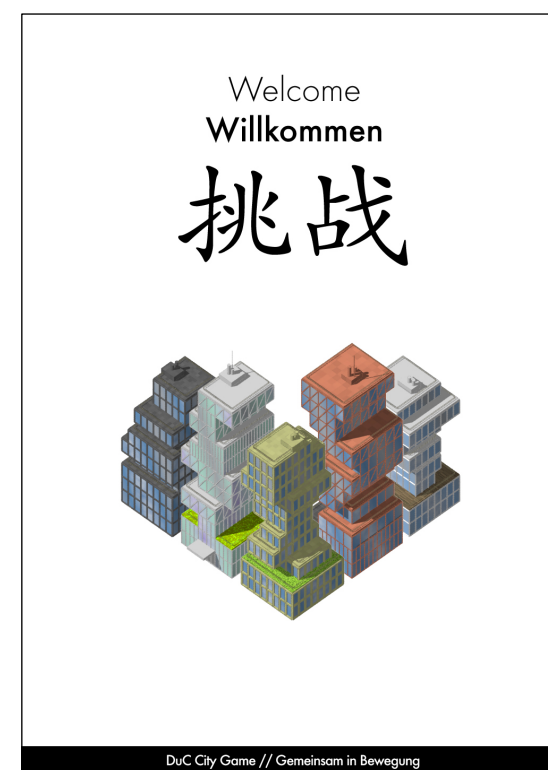
- send state → Fl.
- display mnemonics



DMX msg.  
- fade **out** light in the waiting booths



Playing people will be presented a welcome screen.



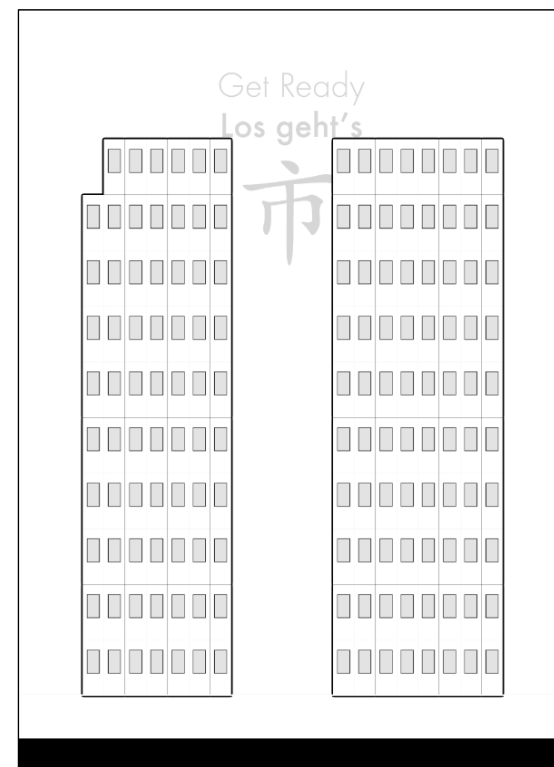
People from the outer waiting queue will take their seats in the inside waiting booths.





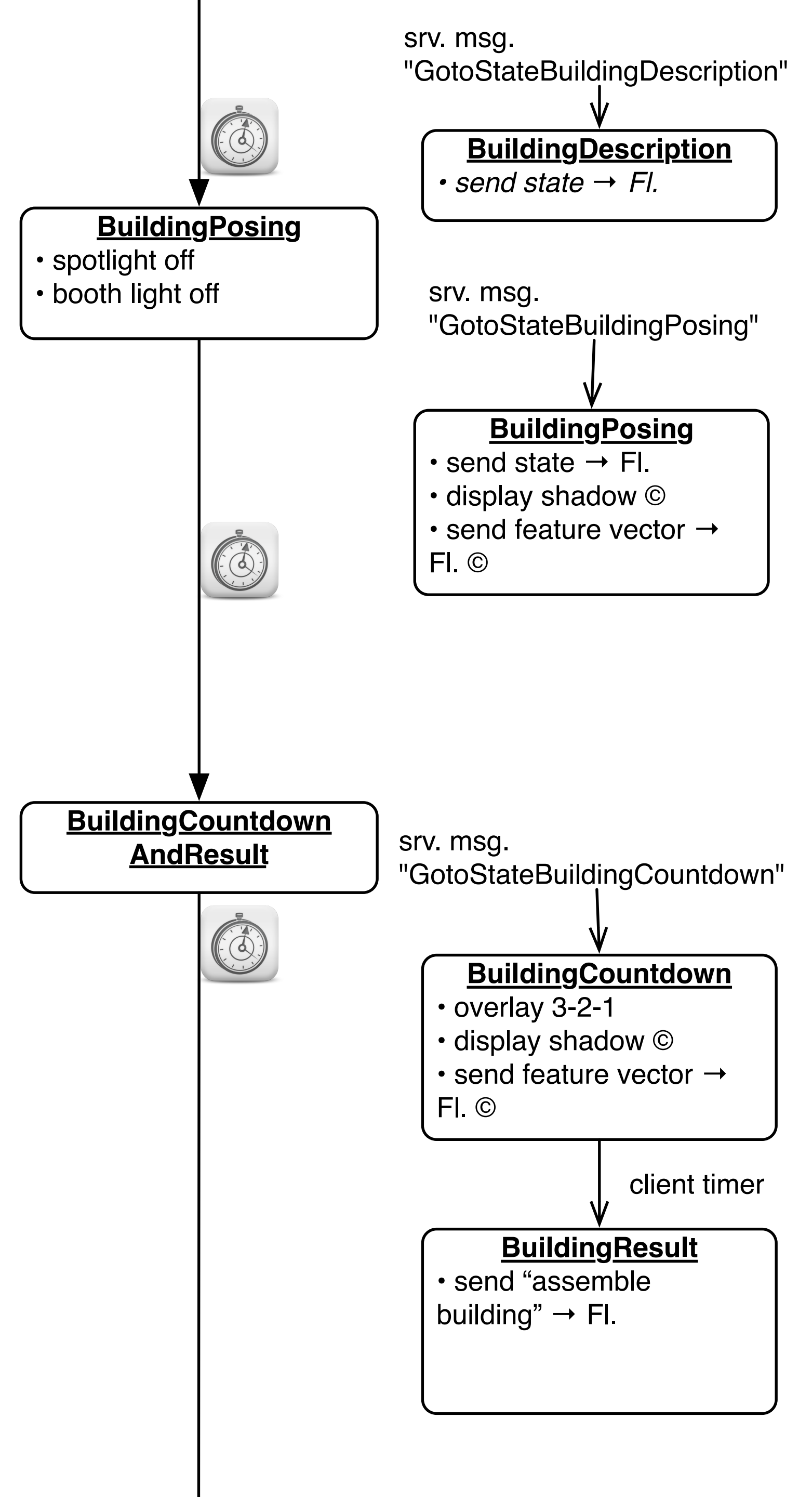
Iconic illustrations will remind the player how to create buildings by their shadows.

People will see their shadows while posing in front of the screens.

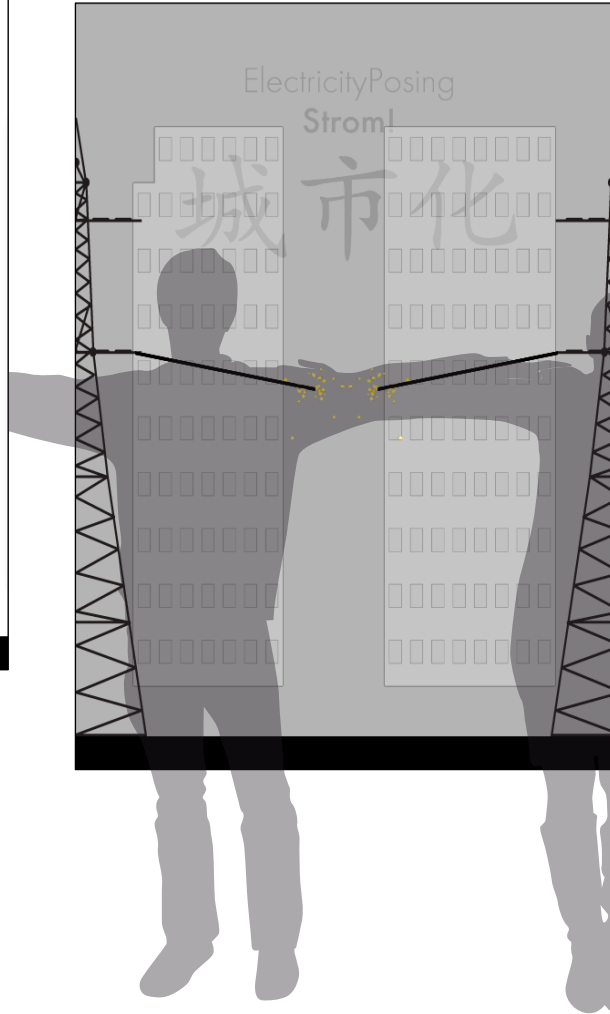
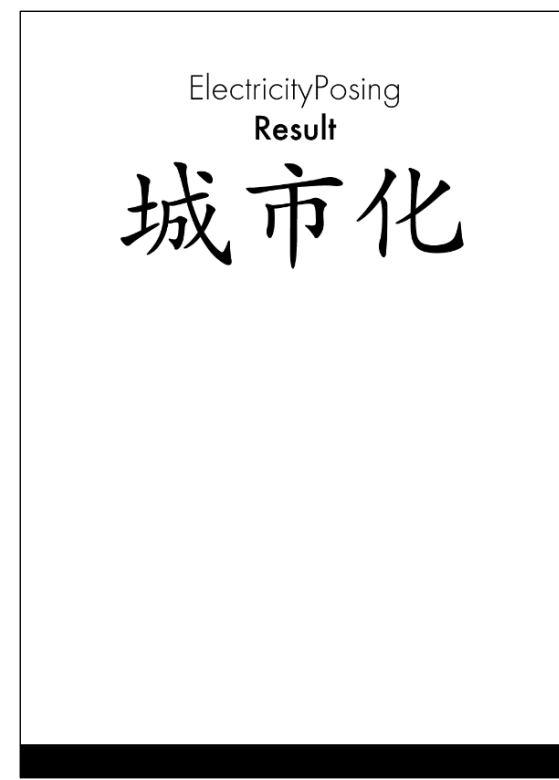


A countdown will indicate that shadows will be "frozen".

People will see the representation of their shadows as buildings.







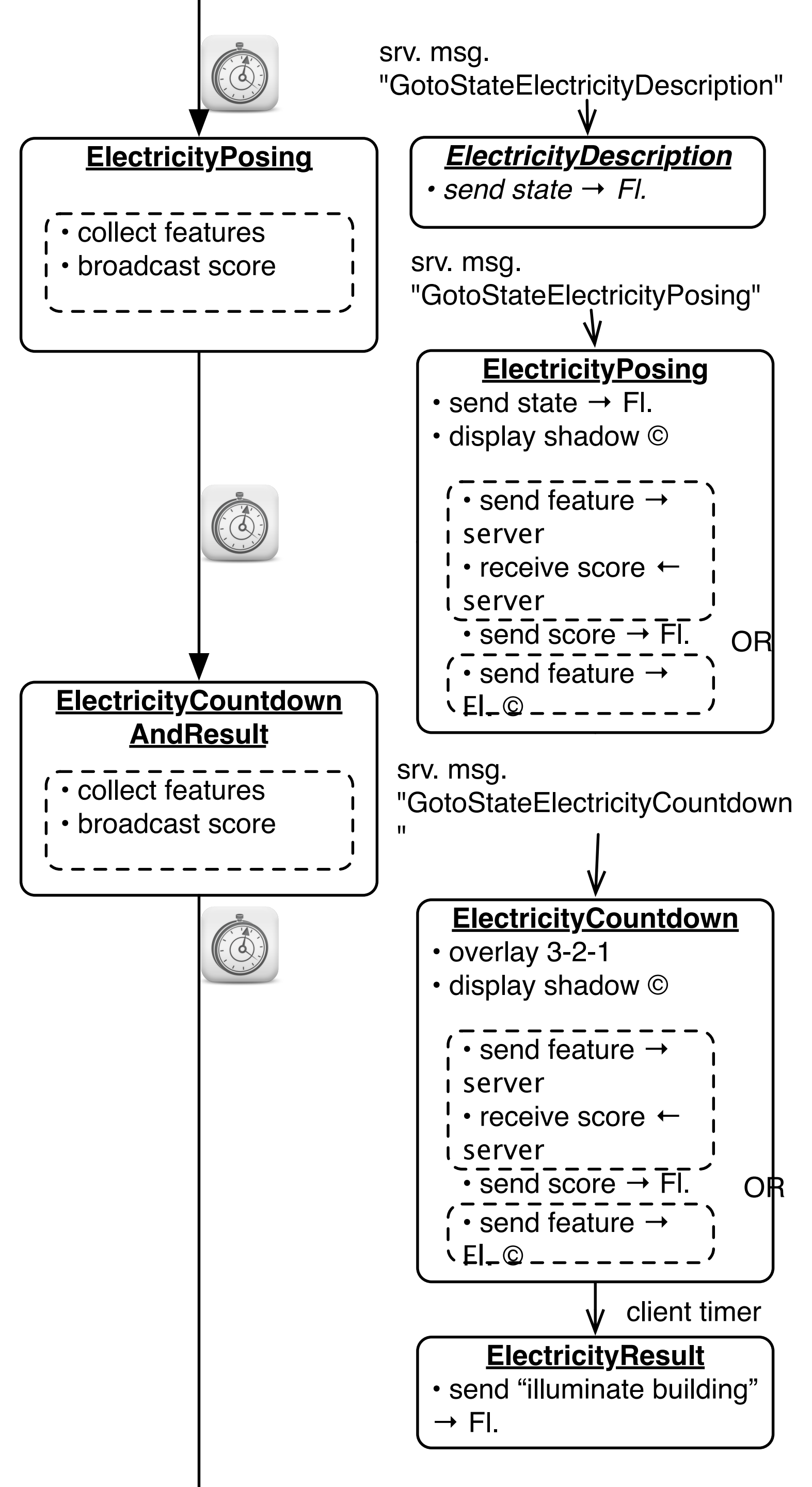
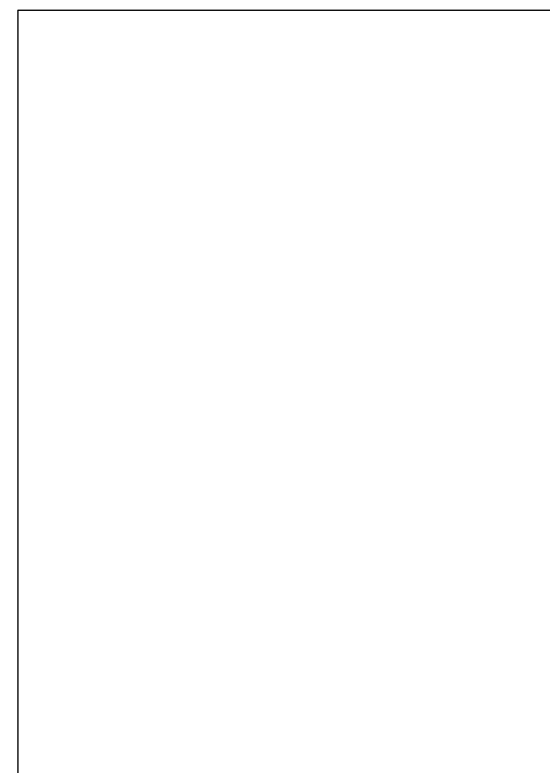
Iconic illustrations will remind the player how to pose in order to have their shadows overlap and thus create a power line.

People will see their shadows while posing in front of the screens.



A countdown will indicate that shadows will be "frozen".

People will see illuminated buildings in 4 different versions??



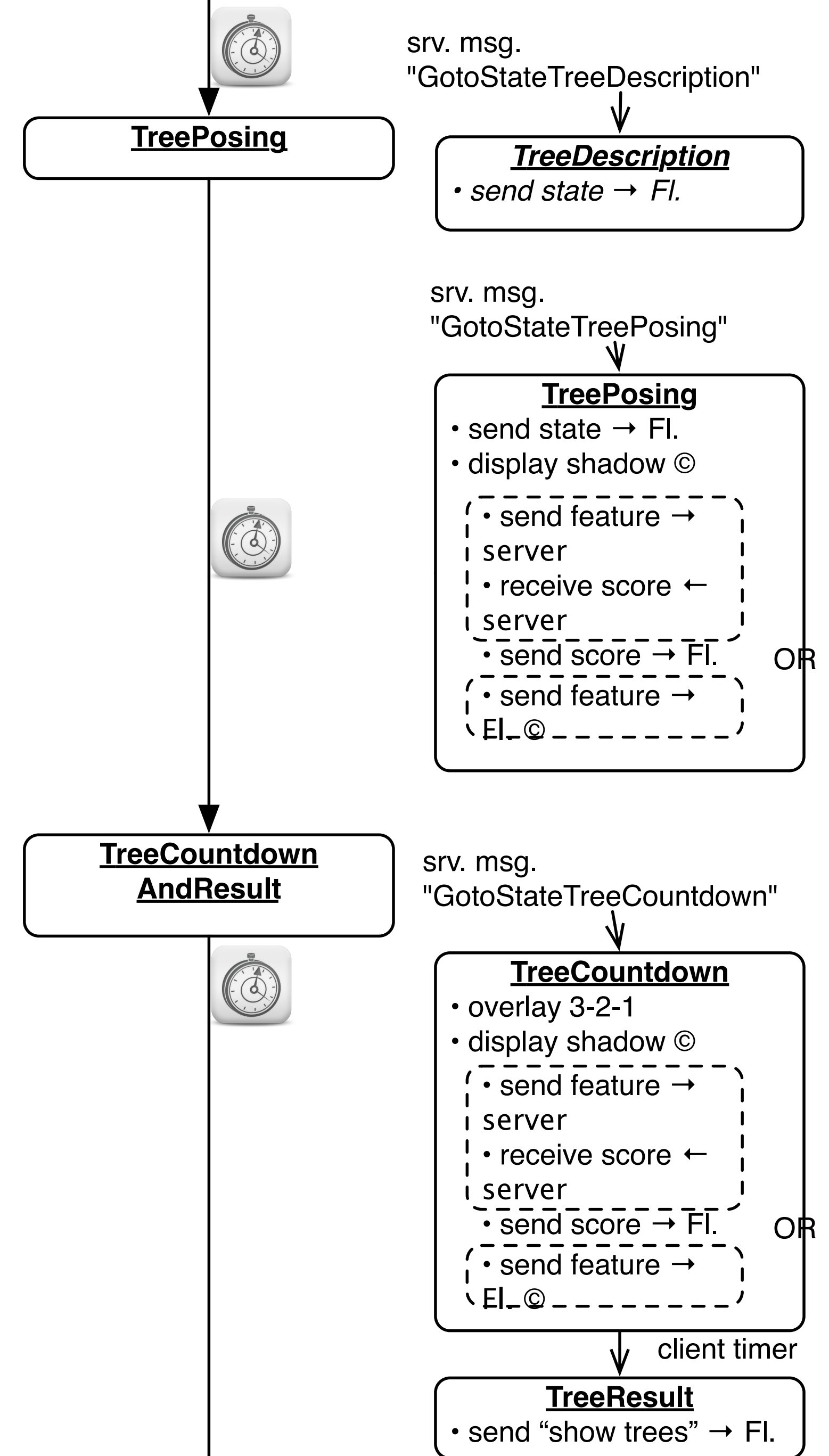


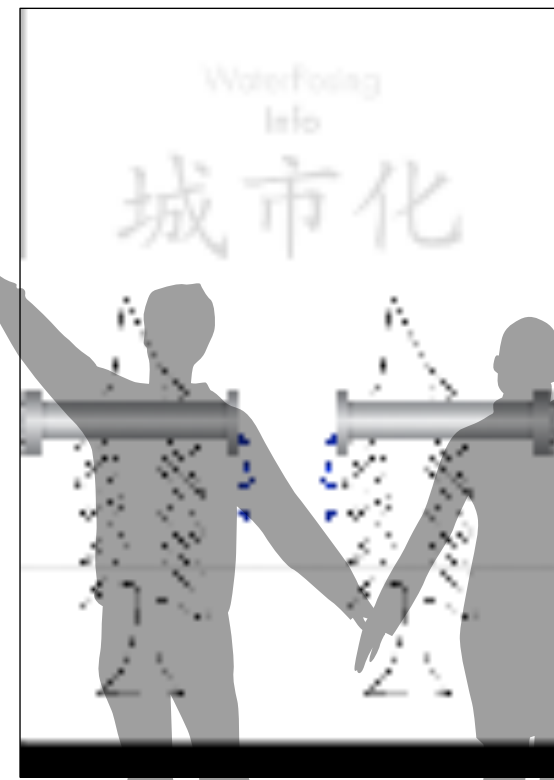
Iconic illustrations will remind the player how to create trees by their shadows.

People will see their shadows while posing in front of the screens.

A countdown will indicate that shadows will be "frozen".

People will see the representation of their shadows as trees.





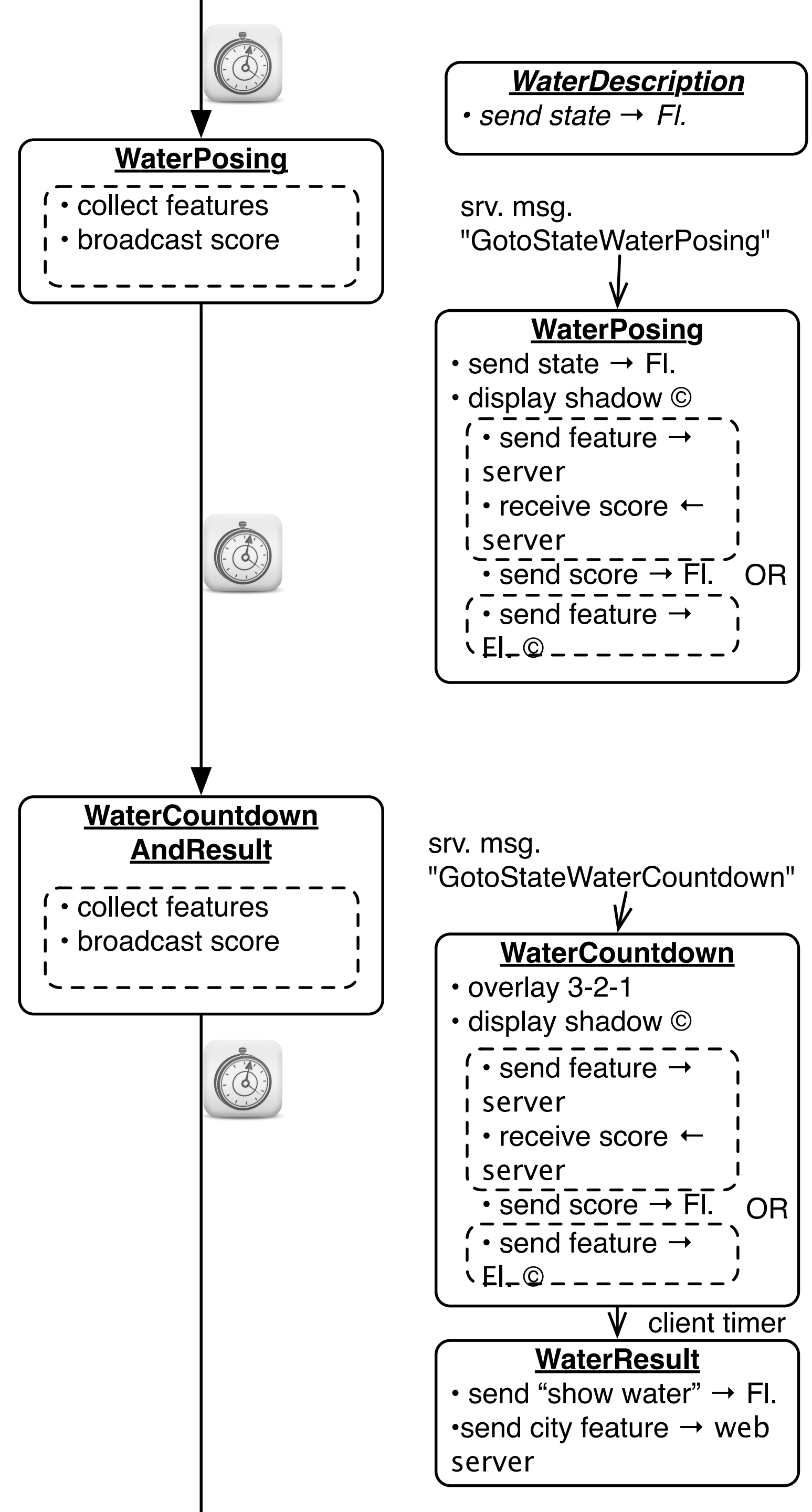
Iconic illustrations will remind the player how to pose in order to have their shadows overlap and thus create a water line.

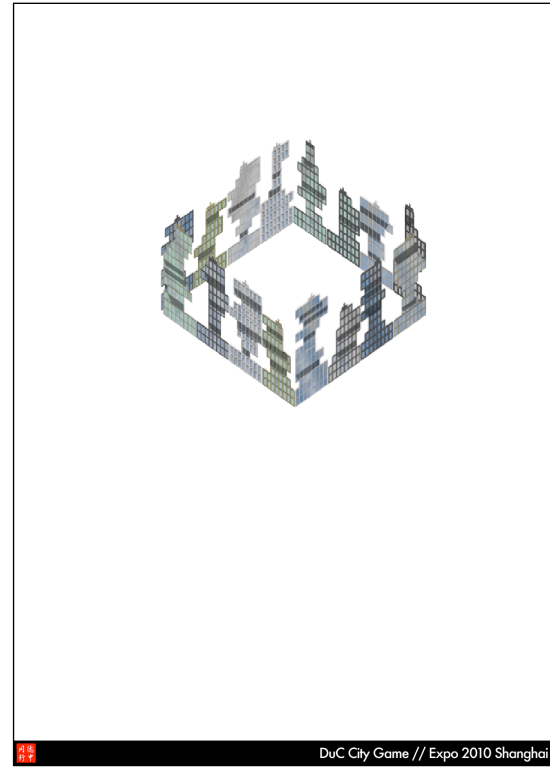
People will see their shadows while posing in front of the screens.



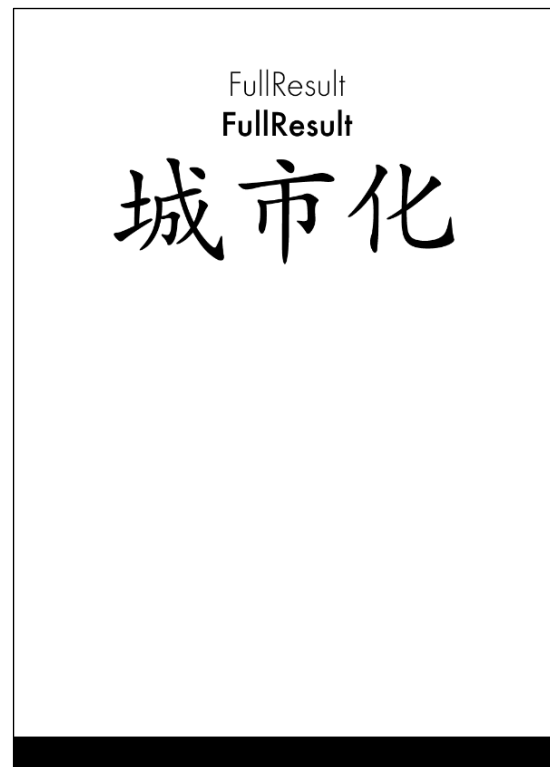
A countdown will indicate that shadows will be "frozen".

People will see the trees that they created with different green colours/trees with leaves?

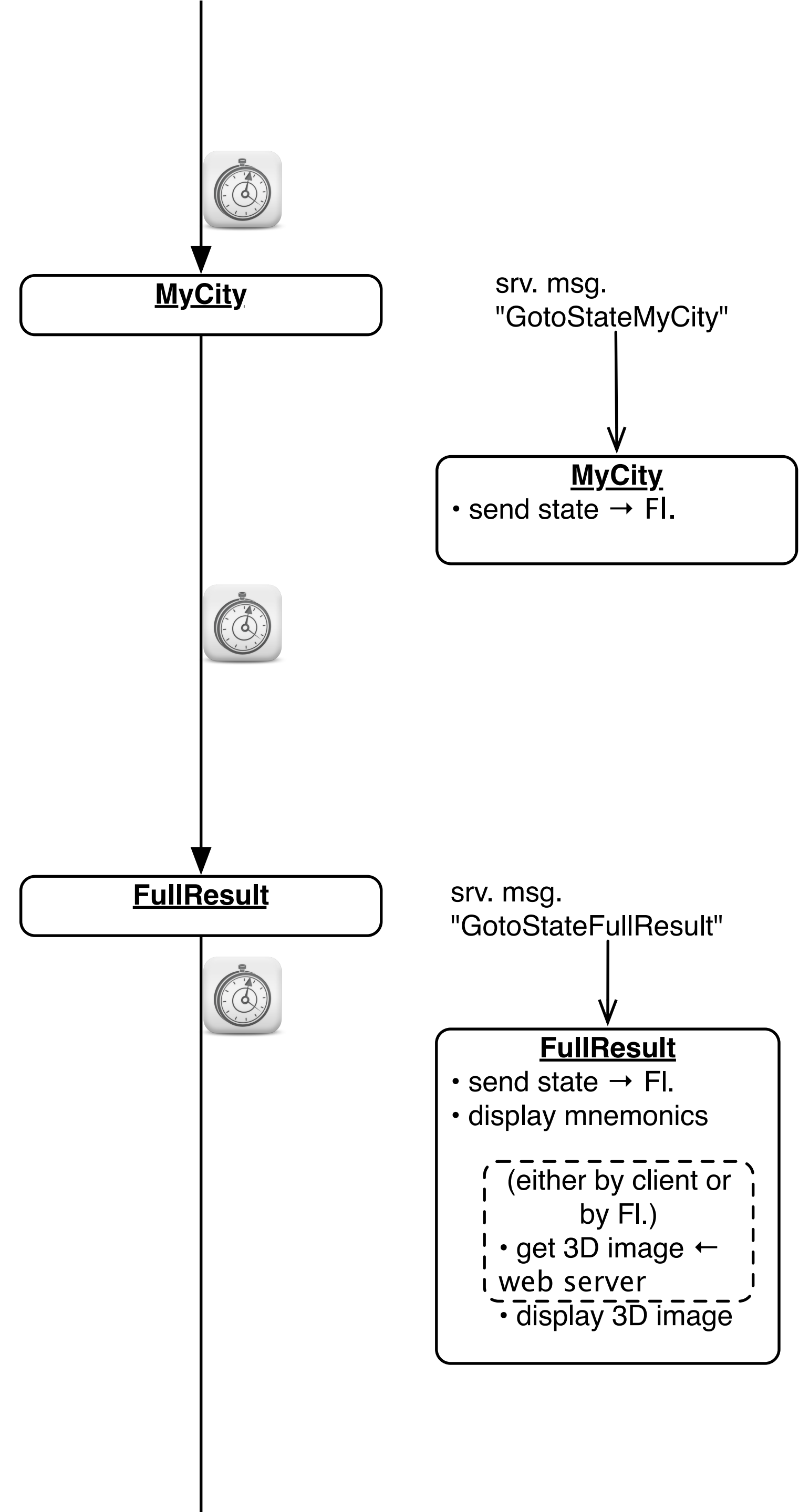




People will see a animation of what will happen with their skyline.



People will see their city block with the block number and the URL of the DuC Website.





People will be asked to move out.

Beyond the playing area, visitors can deepen their understanding of city concepts at individual interactive stations before leaving the pavilion.



**PleaseLeave**

- assign Player IDs to Game ID
- send this to server

srv. msg.  
"GotoStatePleaseLeave"



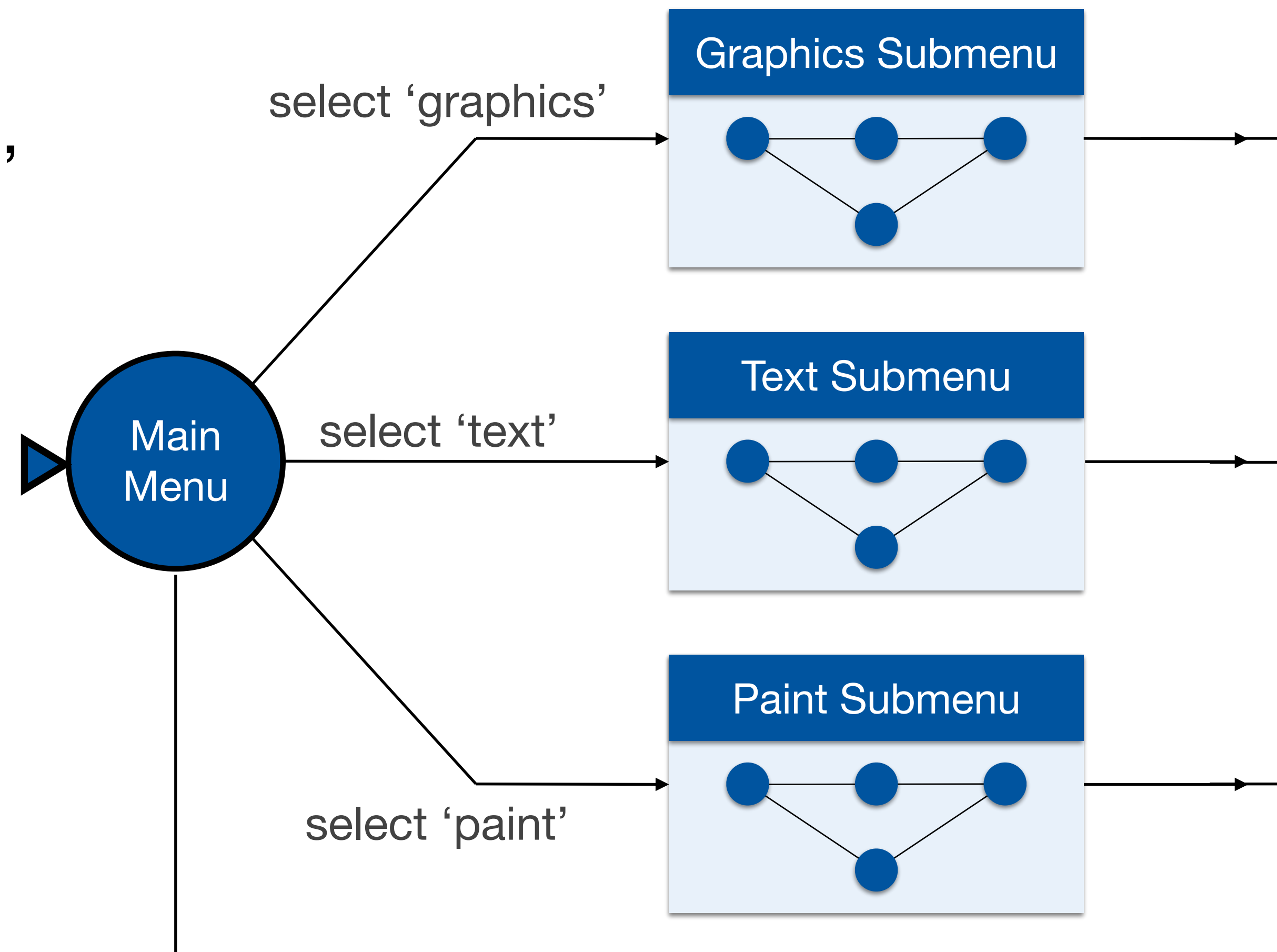
**PleaseLeave**

- send state → Fl.



# Hierarchical STNs

- **Start** and **Finish** states serve to glue an STN for a sub dialog (e.g., a certain menu selection) into a larger dialog (e.g., operating the application in general)
- Same expressive power as STNs, just more convenient
- The dialog structure of an entire system can be specified this way





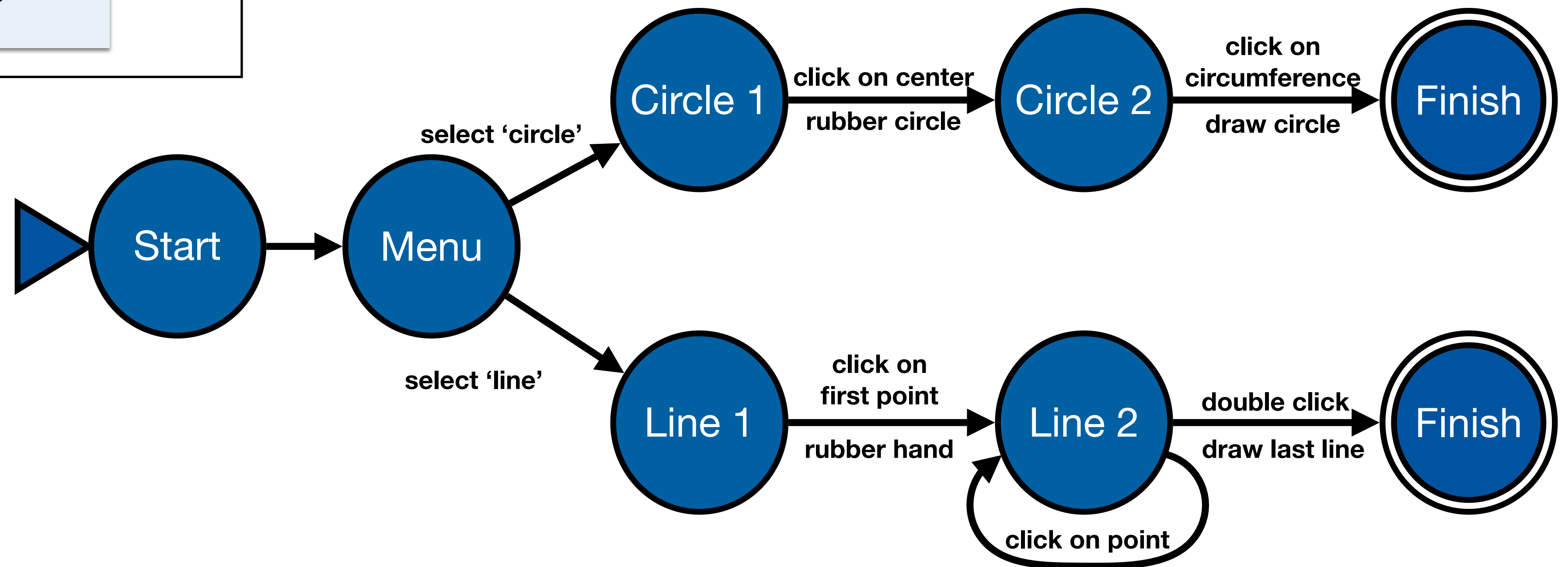
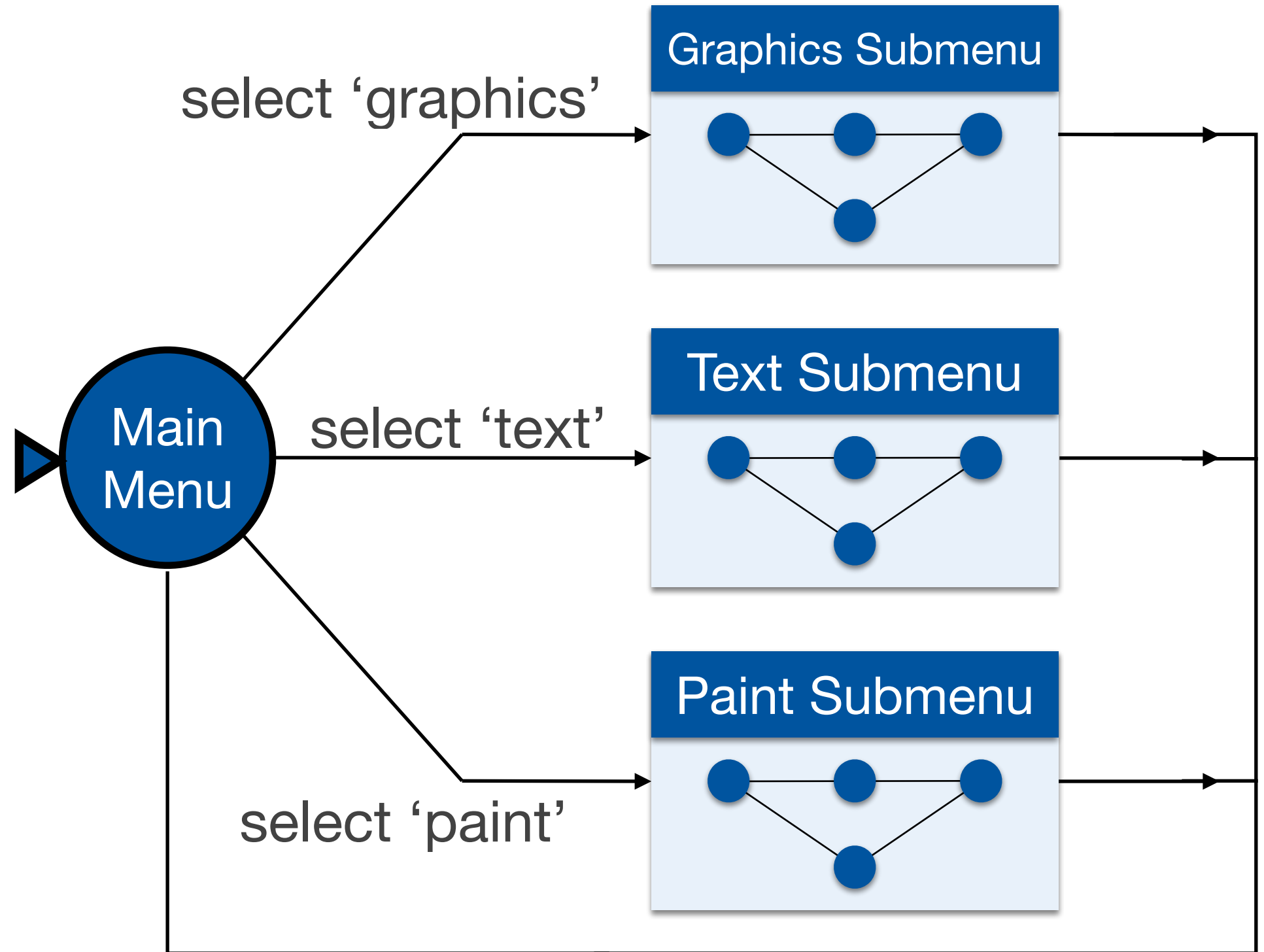
# Using STNs in Prototyping

- Create a simple STN for the dialogs envisioned
- Create one UI snapshot (sketch if paper prototype) per state (label it with the state name)
- Include offscreen area for annotations and to include extra buttons simulating user actions that do not correspond to simple clicks on the current screen
- When walking the user through your paper prototype, consult the STN to find out how to respond to each user action

# Using STNs in Prototyping

- Alternative: Let the computer “execute” the STN to run the prototype
- Use tools such as Keynote, PowerPoint



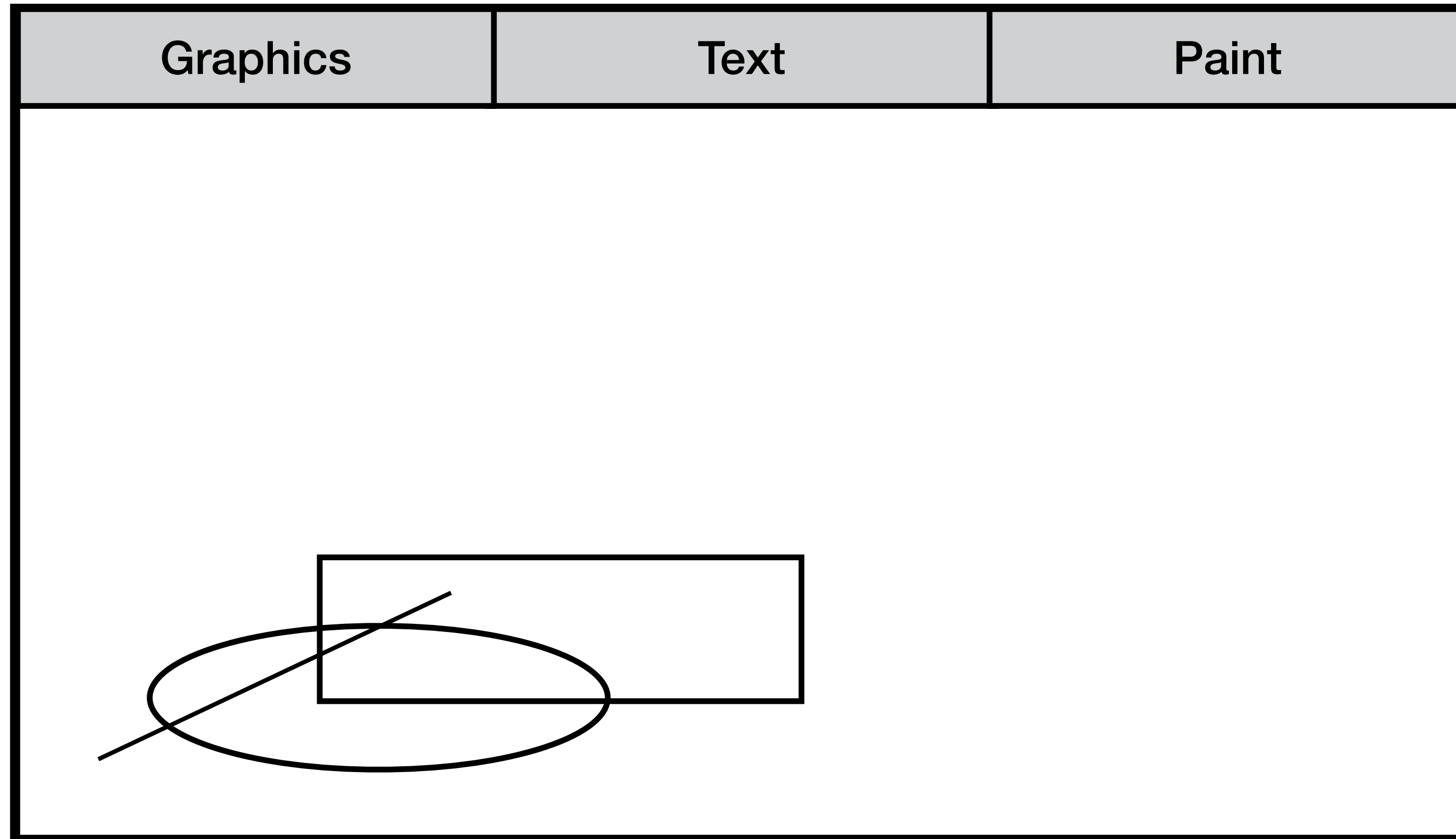


# Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state:

Main Menu



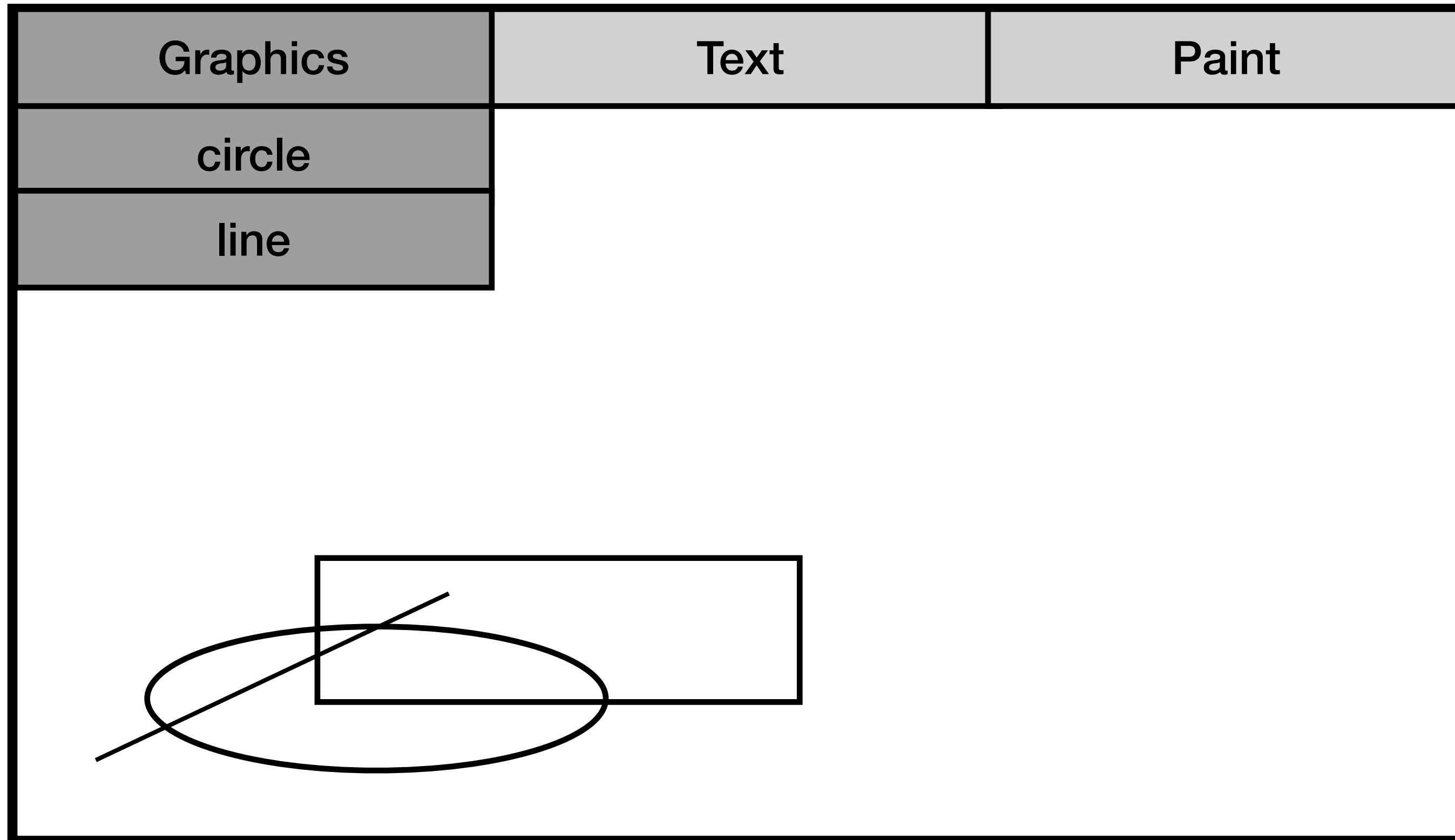


# Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state:

Main Menu

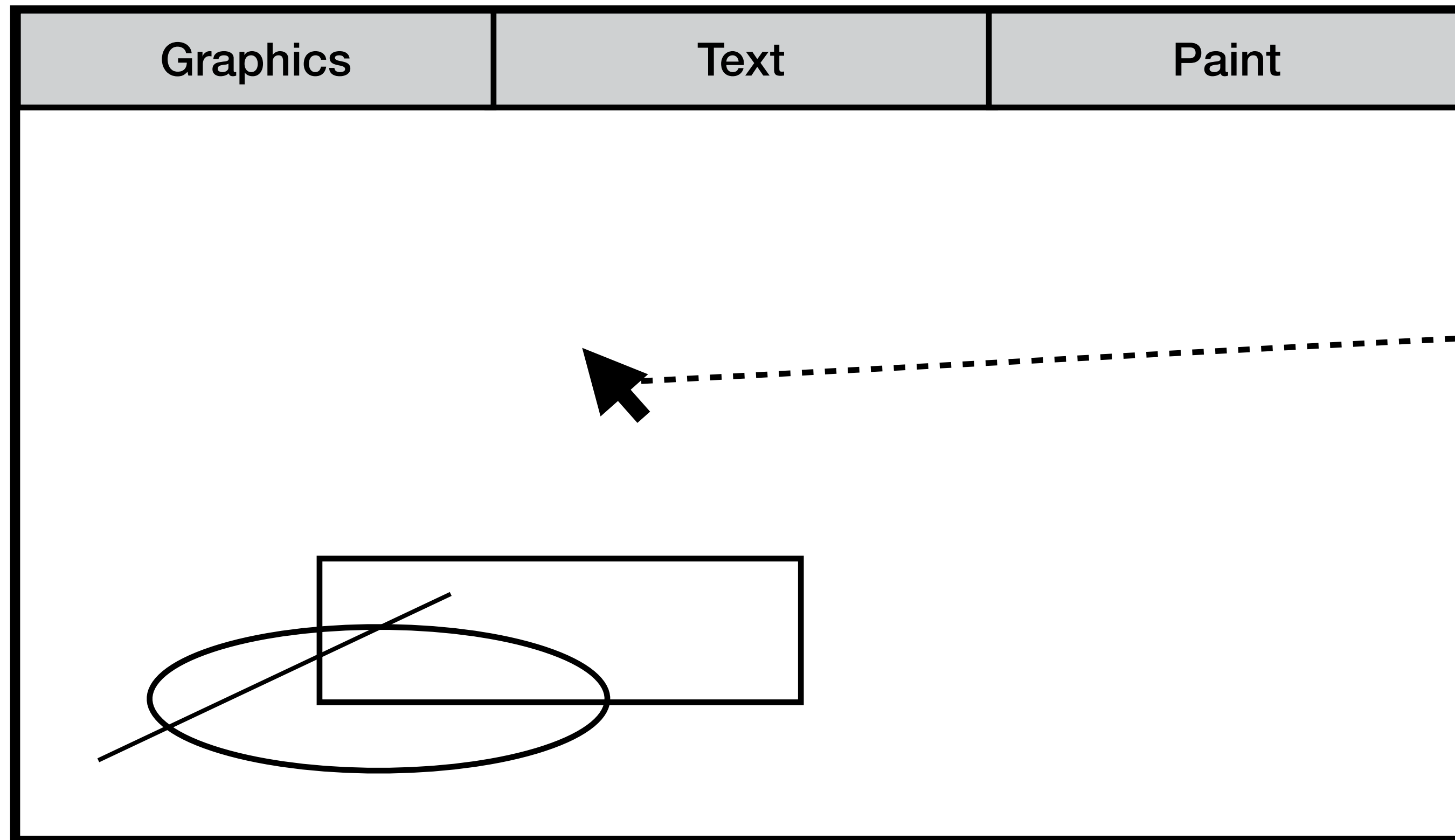


# Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state:

Circle 1



Click this button to simulate a click on the drawing area.

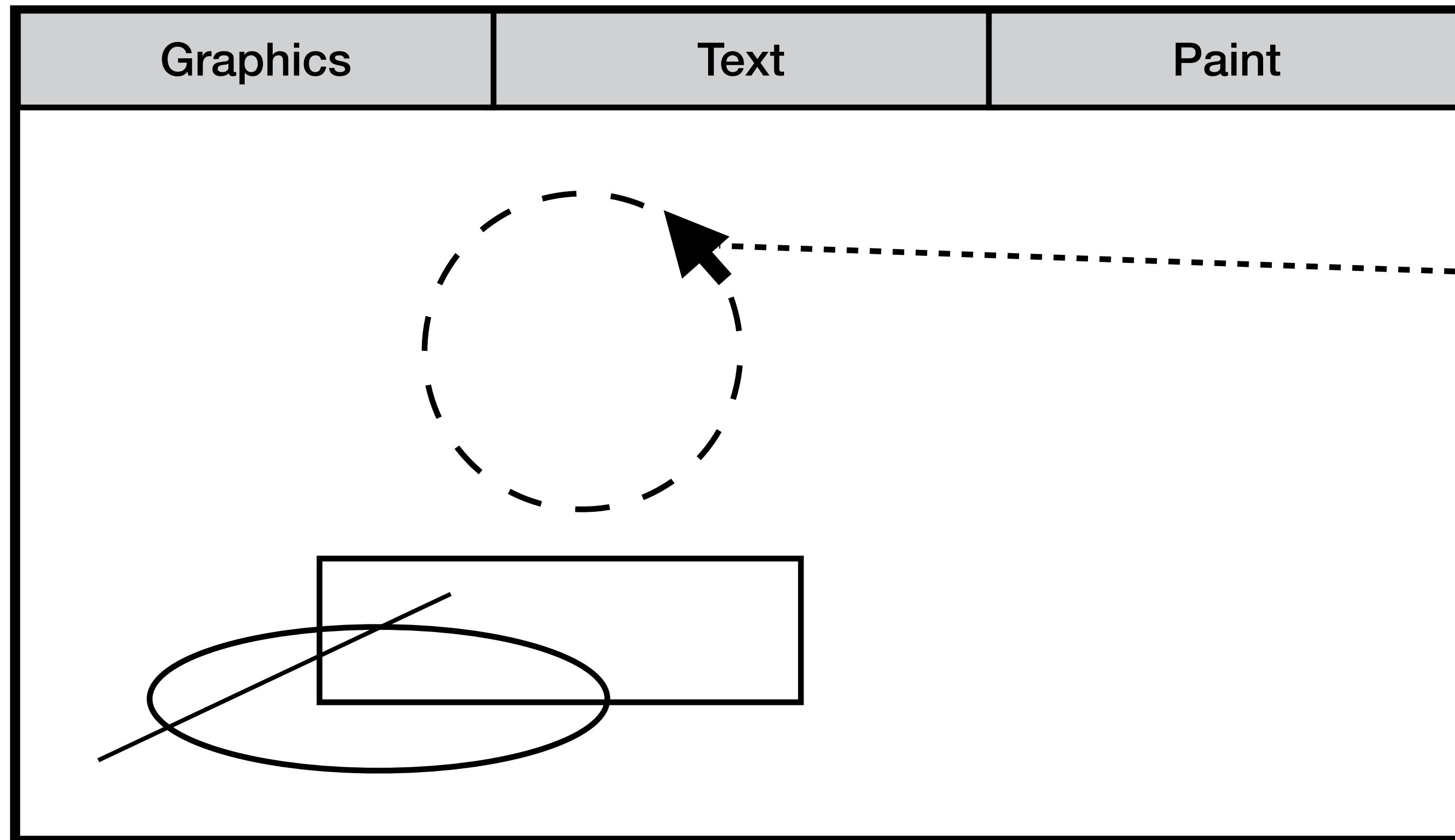
Click on Center

# Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state:

Circle 2



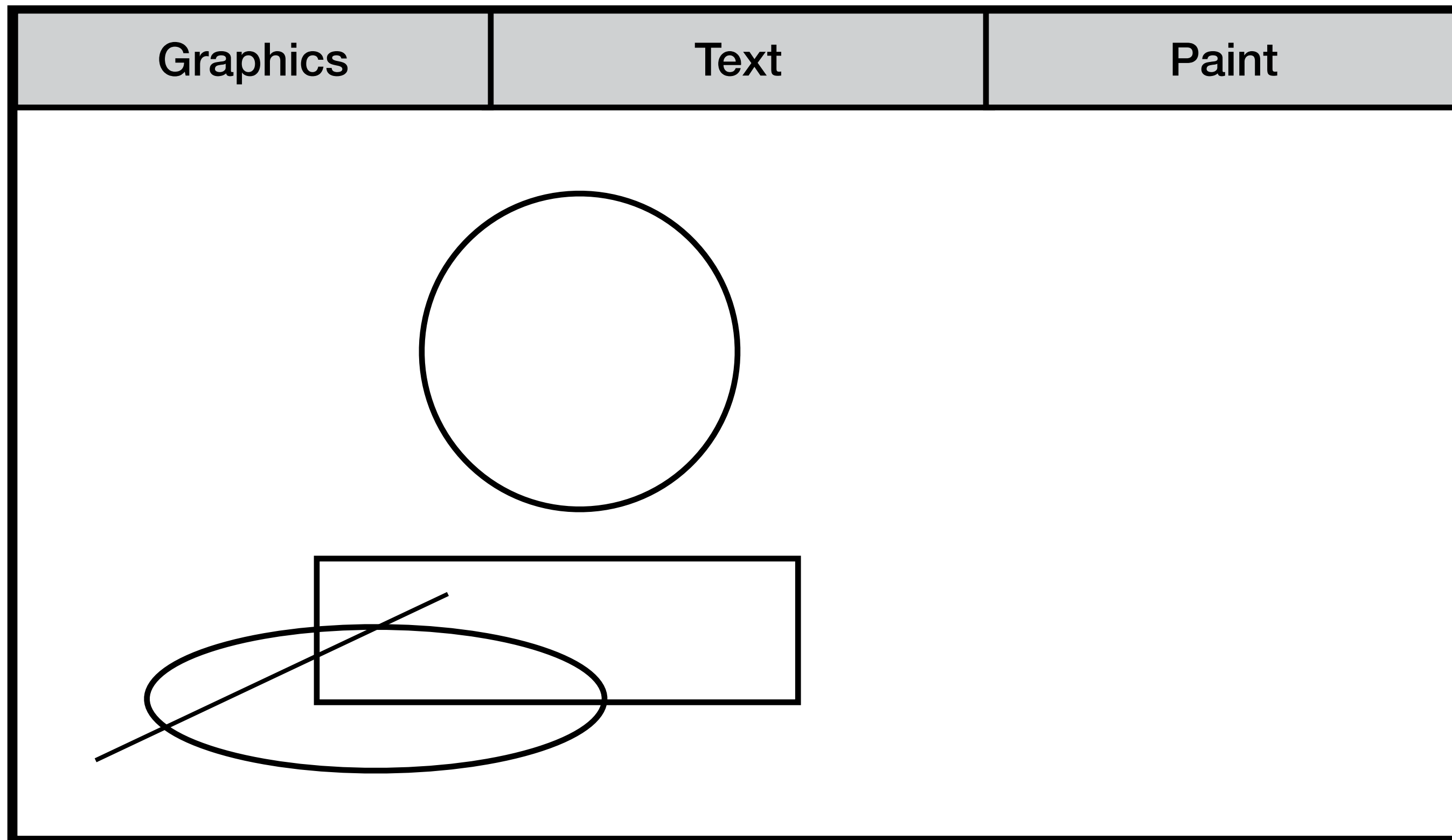
Click this button to simulate a click on the drawing area.

Click on Circumference

# Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state: **End of Drawing**



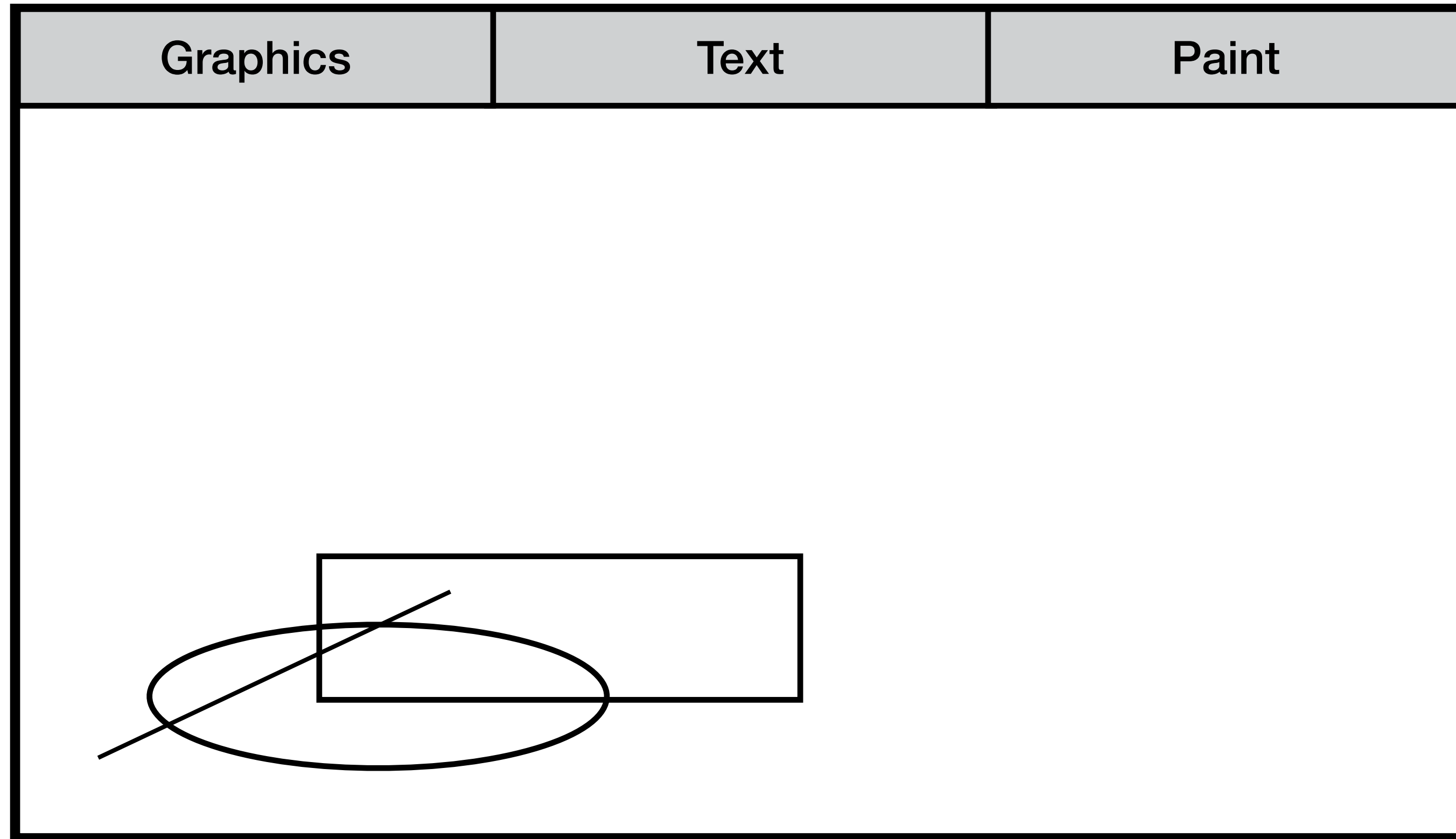
The circle is drawn now.  
Click the button to go back to main menu.

**Back to Main Menu**

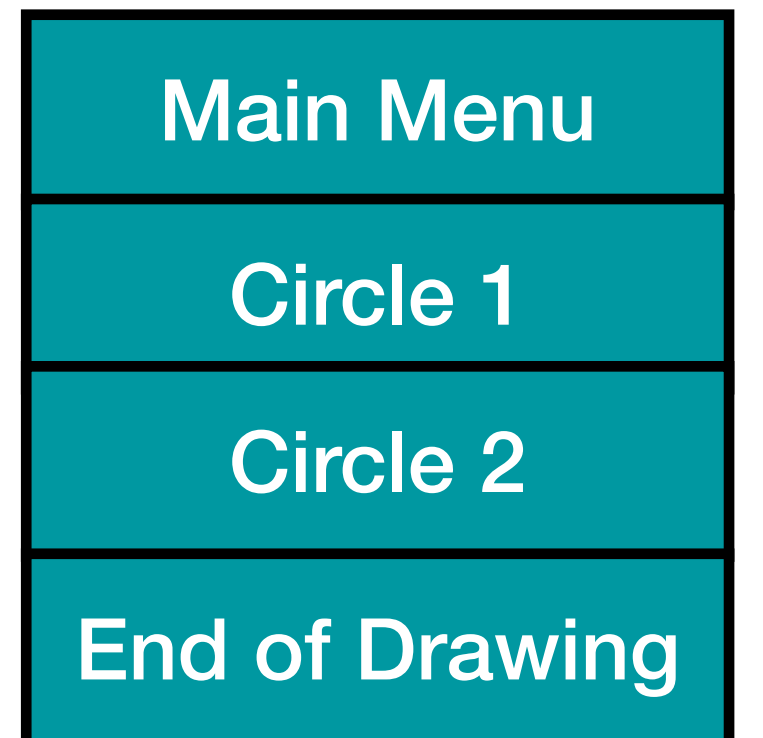


# Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8



Current state:



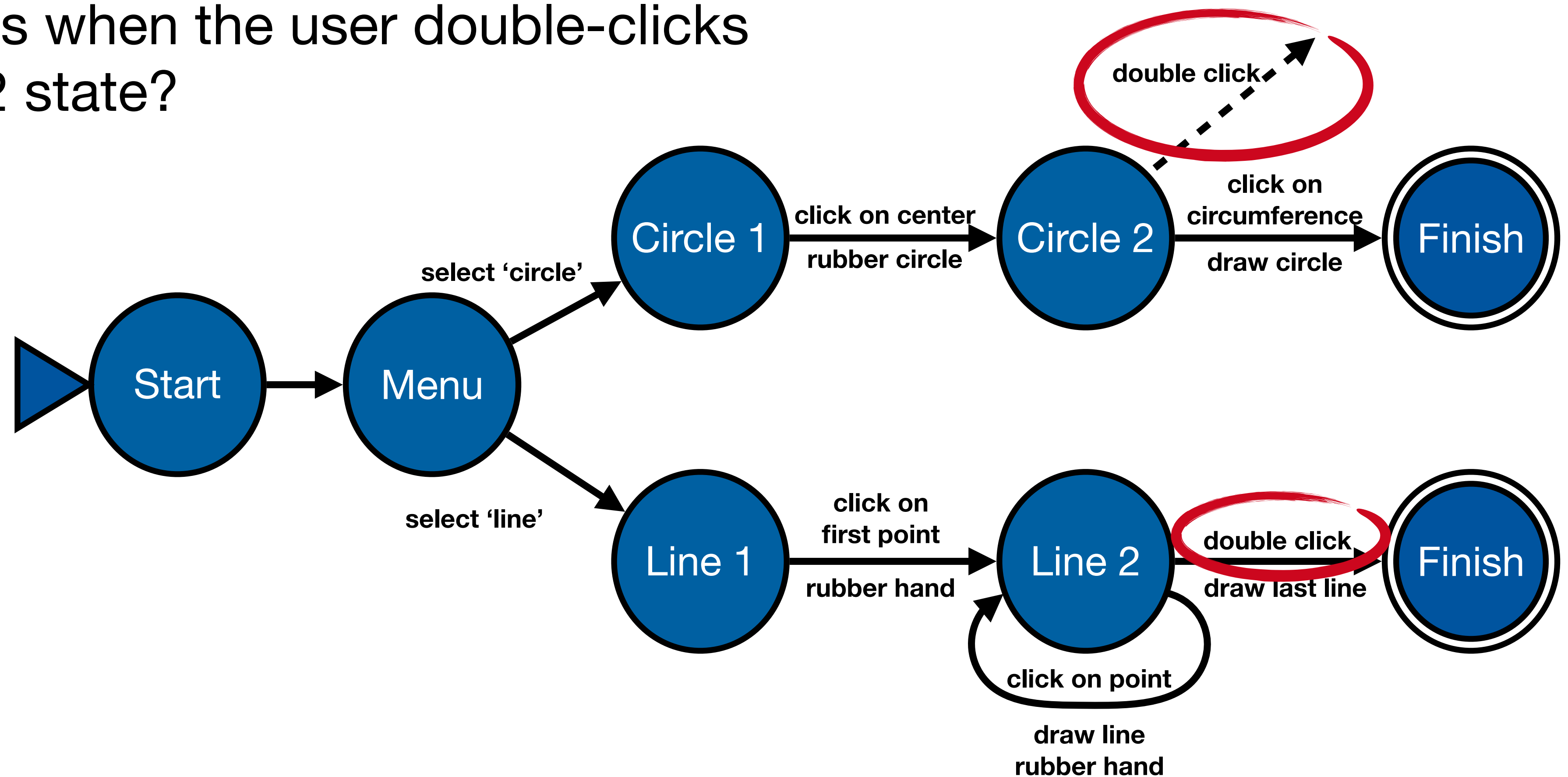
# Checking STN Properties: States

- Completeness
  - Can you get anywhere from anywhere?
  - Are all possible actions covered in every state?
  - How easily?
- Reversibility
  - Can you get to the previous state?
  - But NOT undo
- Dangerous states
  - Some states you don't want to get to



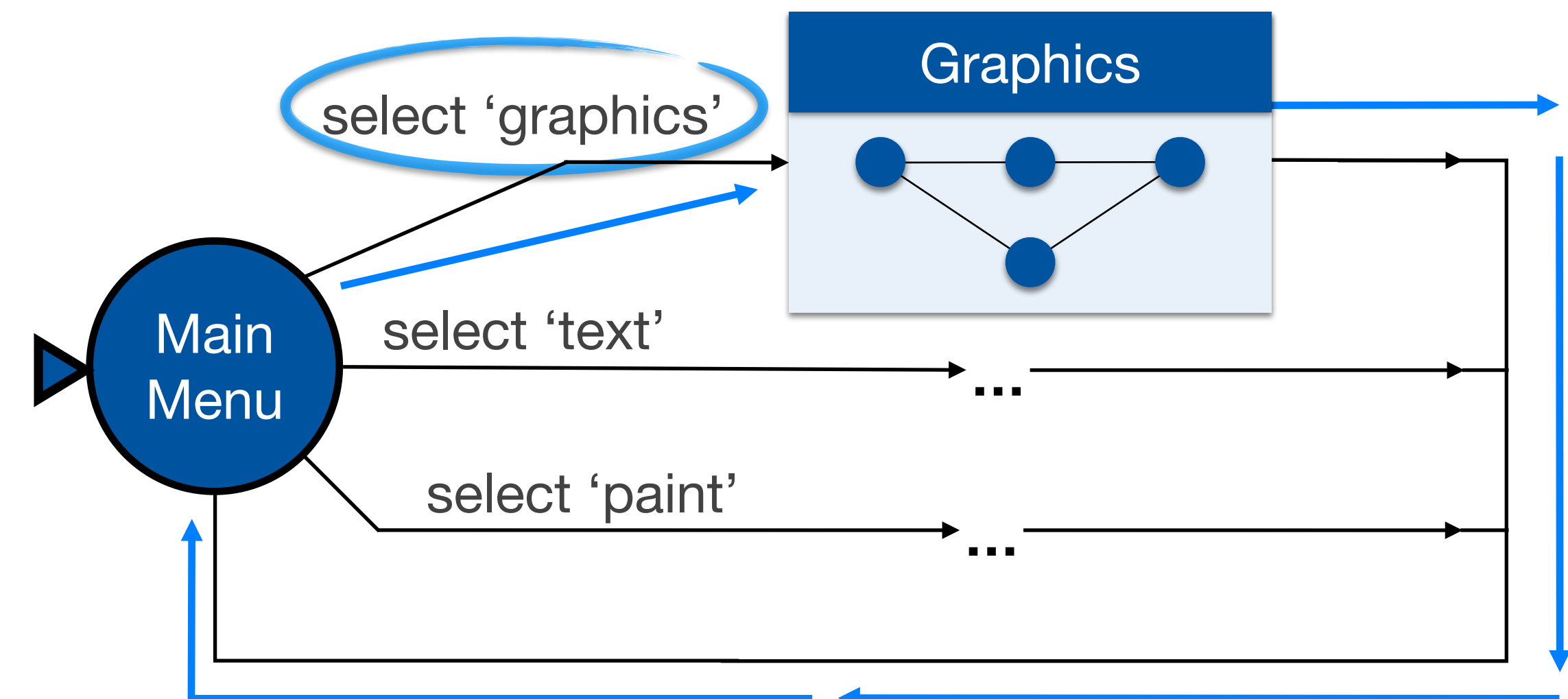
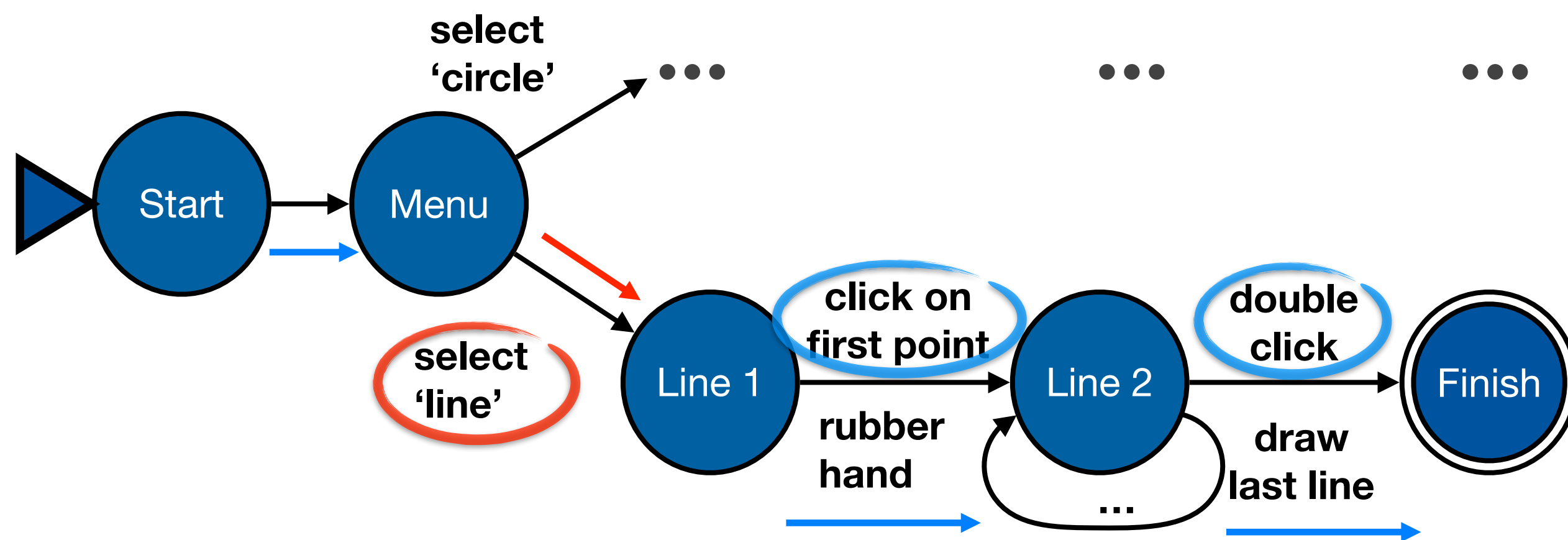
# Checking Transition Properties: Completeness

- Missing arcs indicate unspecified user input
- What happens when the user double-clicks in the Circle 2 state?



# Checking Transition Properties: Reversibility

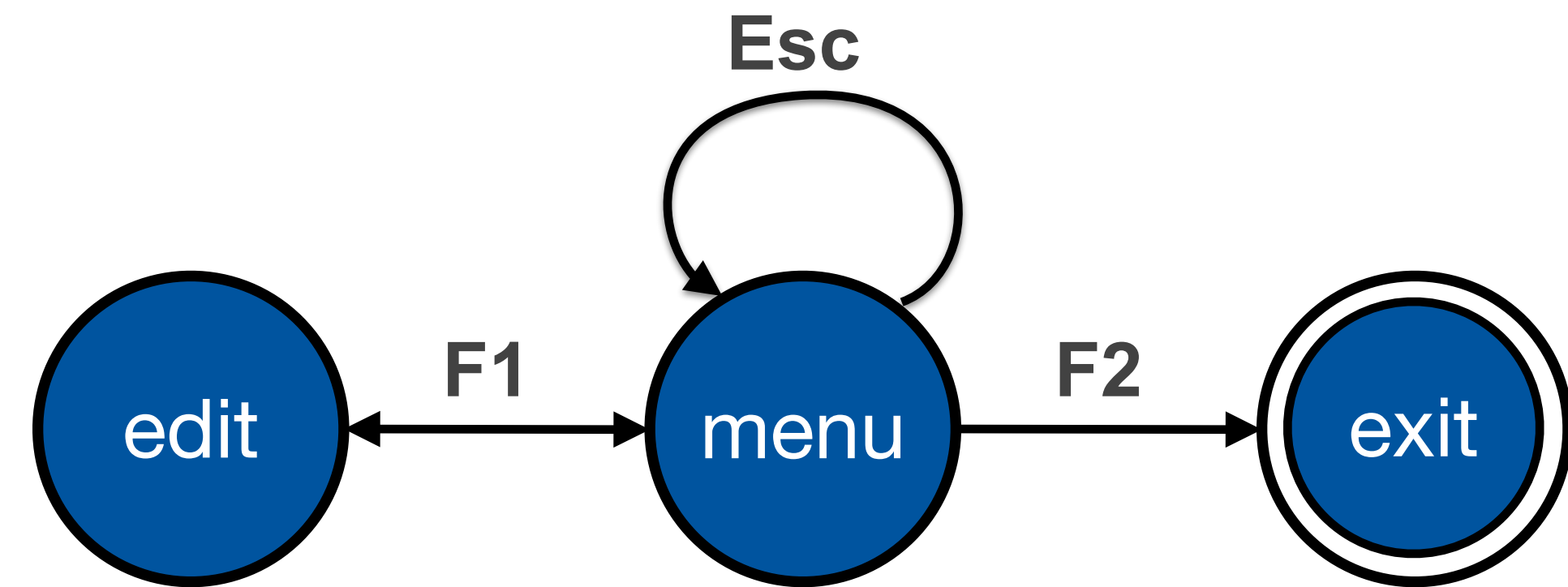
- E.g., reversing **select 'line'** requires *Click - double click - select 'graphics'* (3 actions)
- Note: Reverse means just getting back to a state, **not** to “undo” its effect





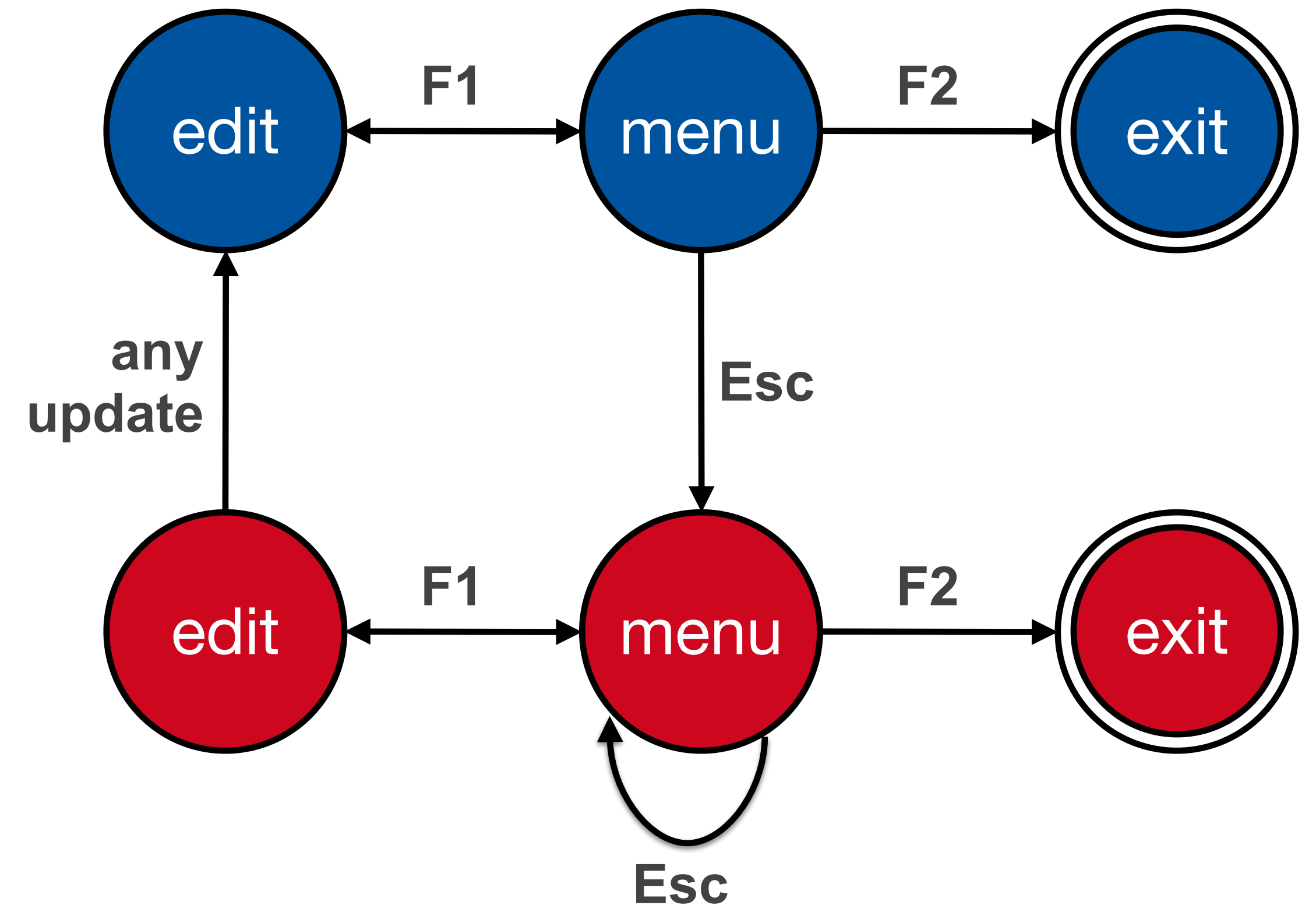
# Dangerous States Example

- Word processor: two modes and exit
  - F1 - changes mode
  - F2 - exit (and save)
  - Esc - no mode change
- But ... Esc resets autosave



# Dangerous States Example

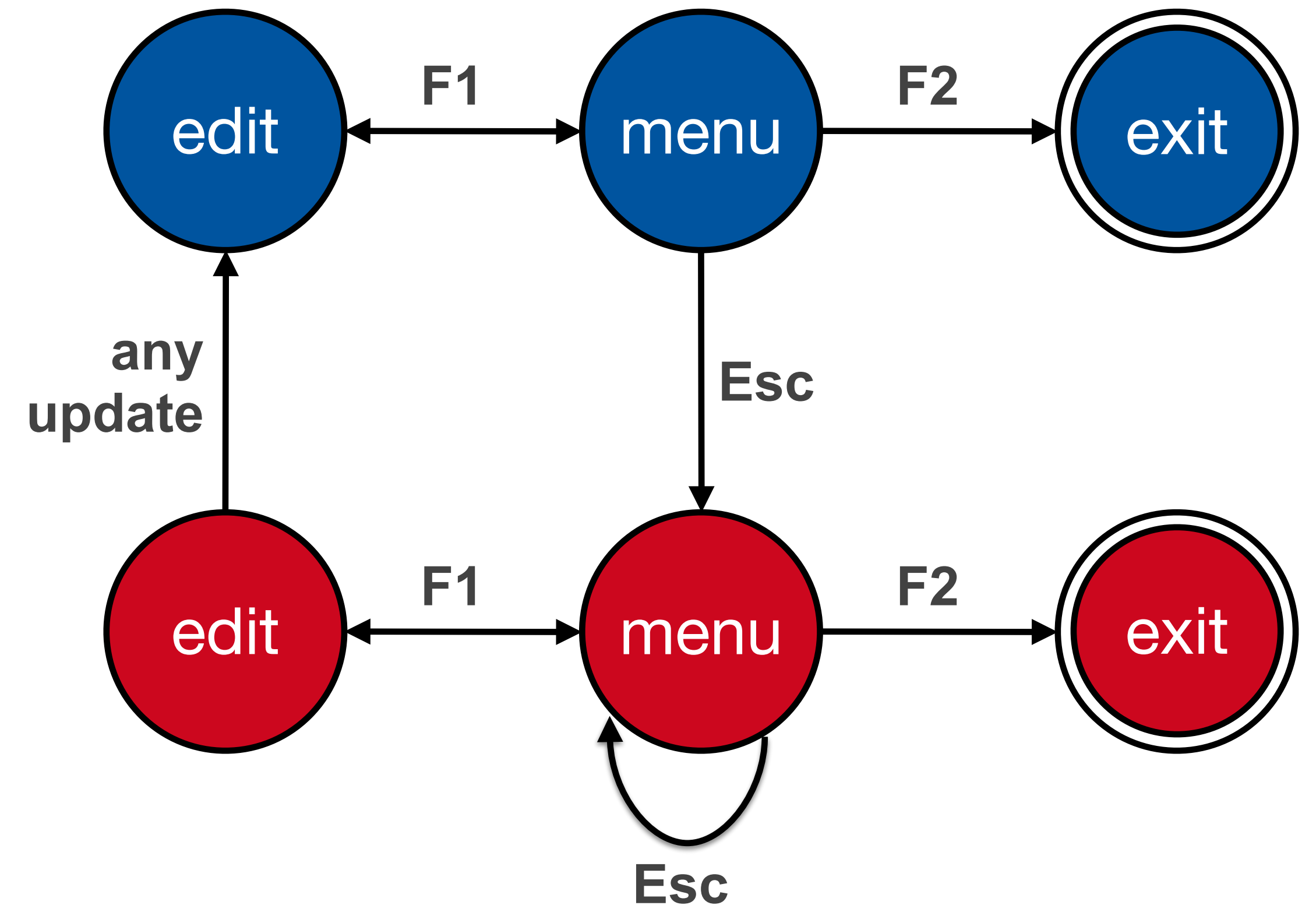
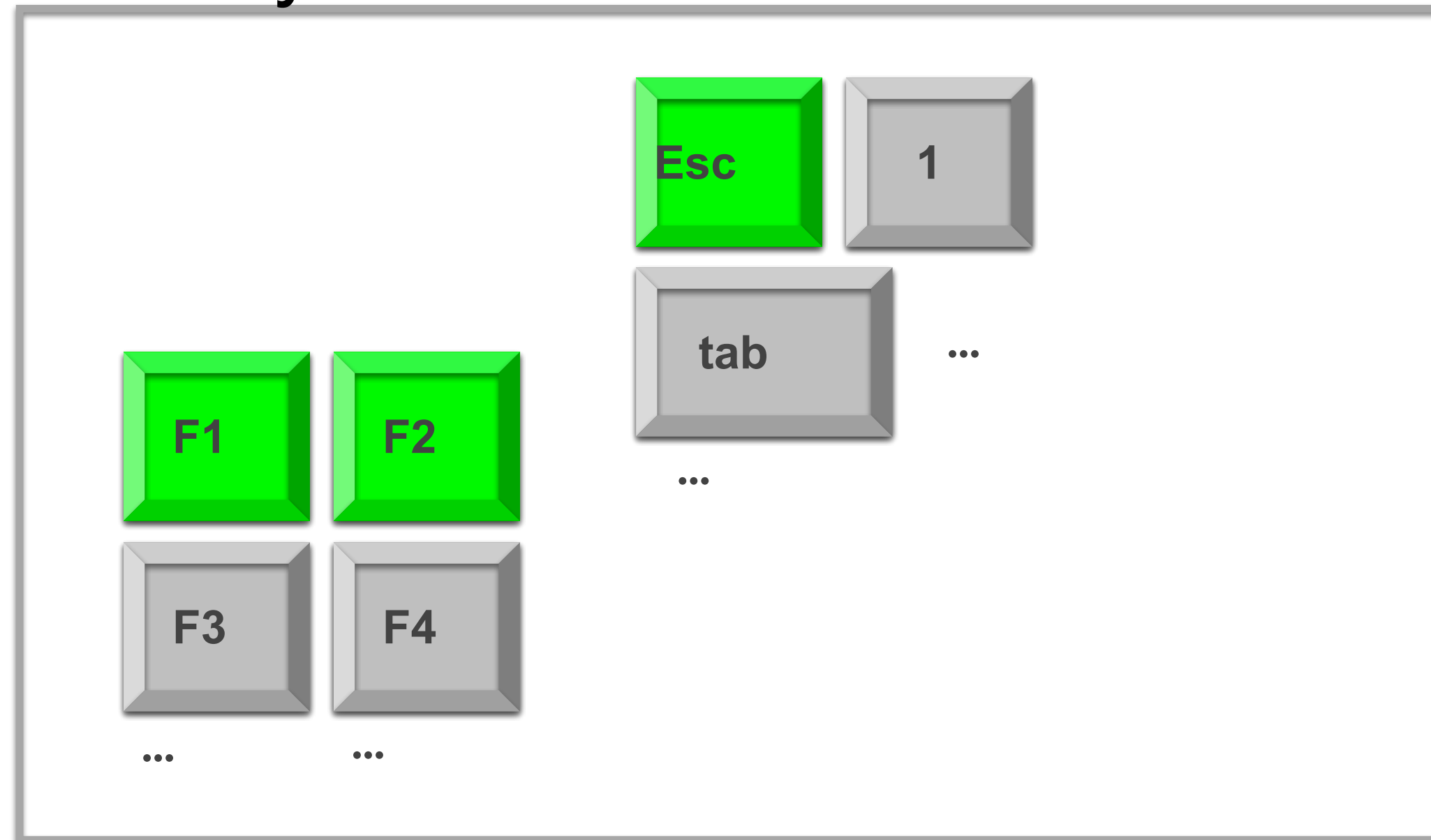
- Exit with/without save  $\Rightarrow$  dangerous states
- Duplicate states - semantic distinction



- F1-F2 - exit with save
- F1-Esc-F2 - exit with no save

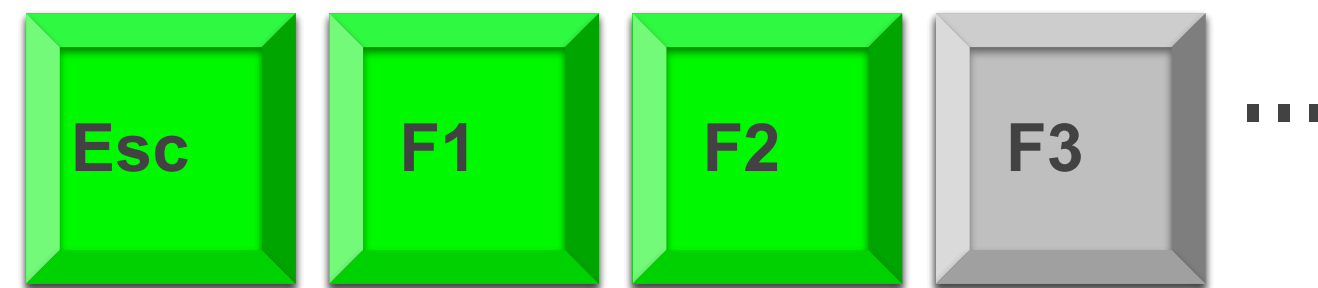
# Dangerous States Example: Layout Matters

Old keyboard—OK



# Dangerous States Example: Layout Matters

new keyboard layout

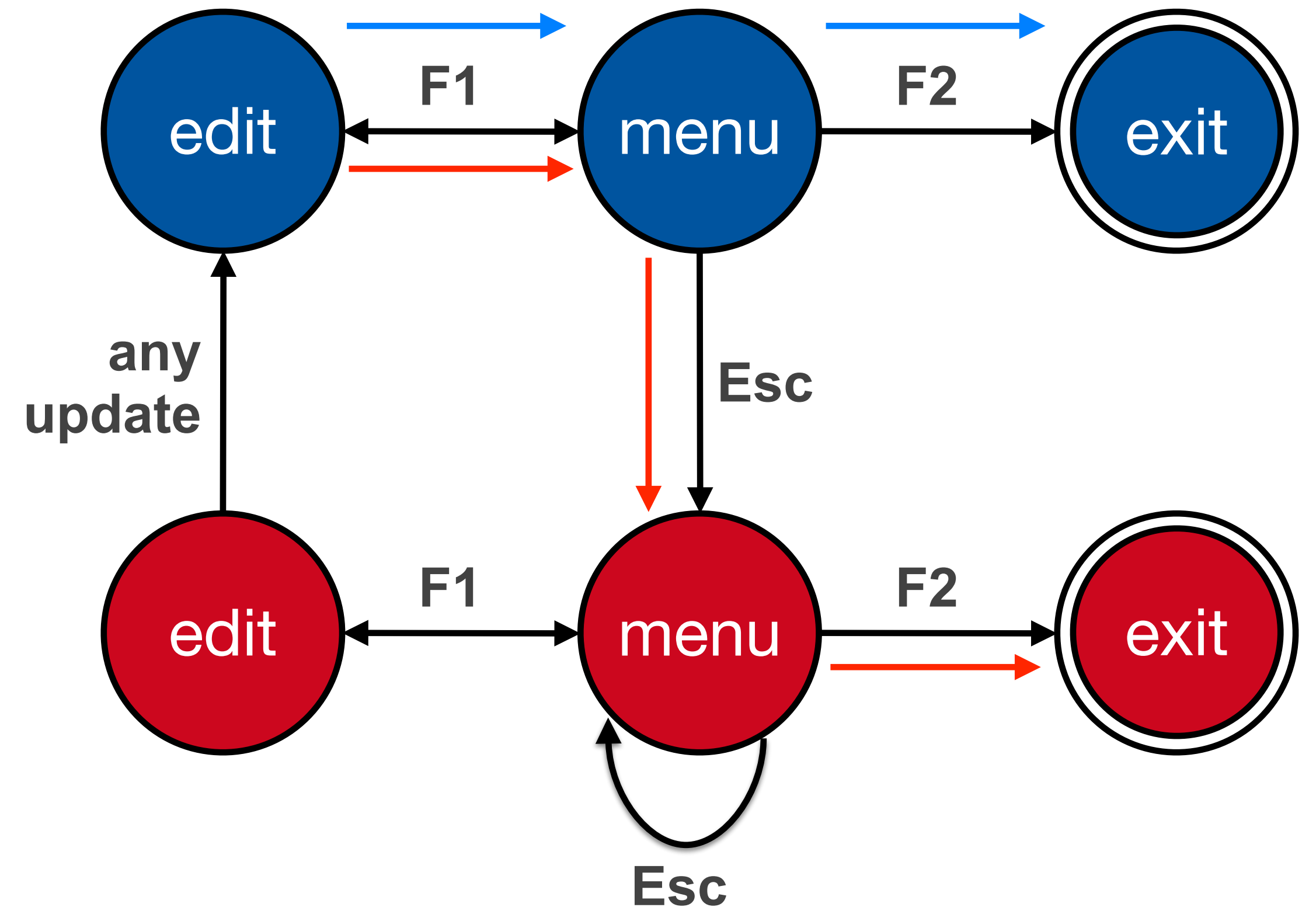


Intend **F1-F2** (save)



Finger catches Esc

**F1-Esc-F2** — disaster!



# Checking STN Properties: Other Transition Properties

- Determinism
  - Several arcs for one action
    - Deliberate: application decides
    - Accidental: production rules
- Nested escapes
- Consistency
  - Same action, same effect?
  - Modes and visibility

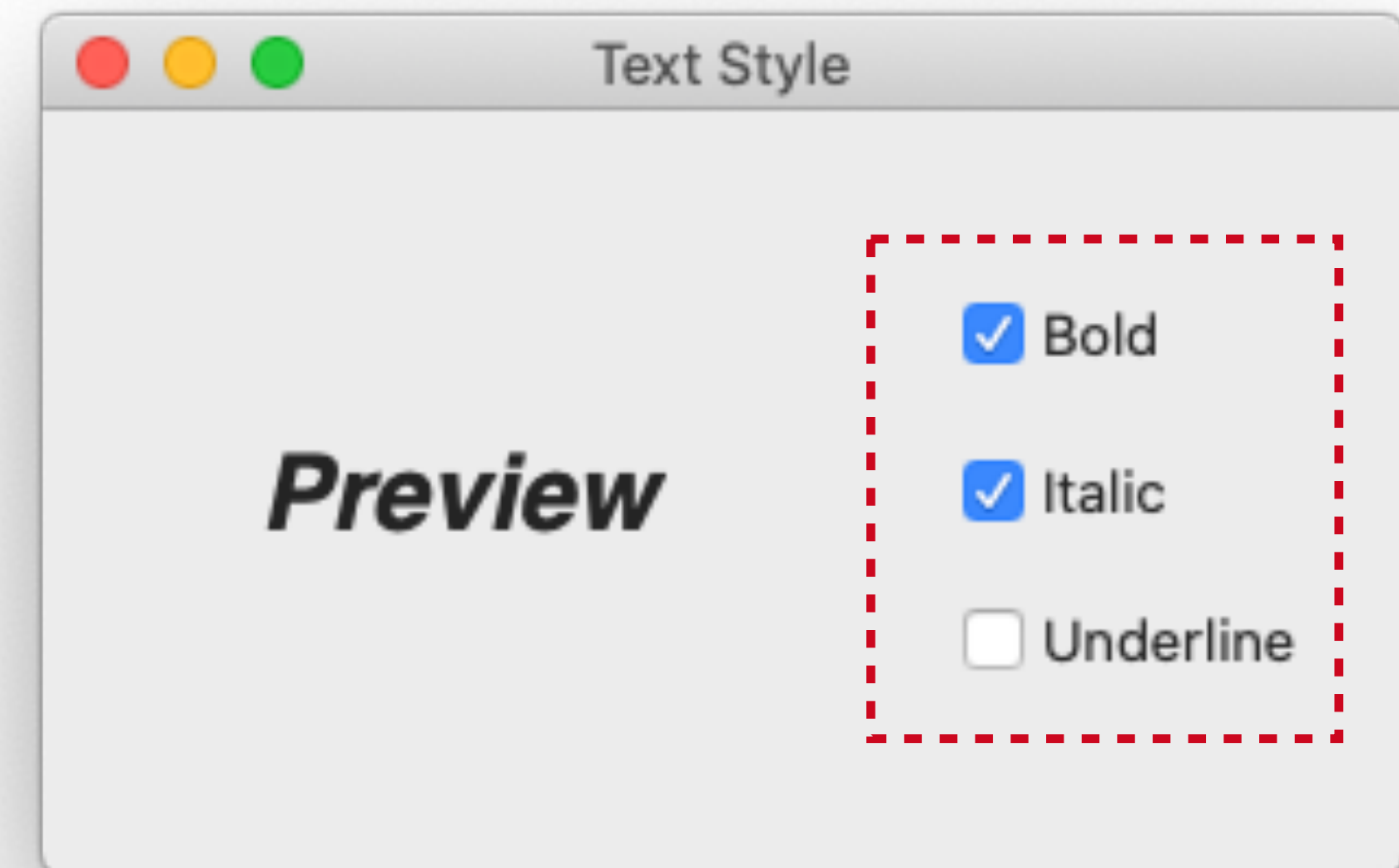




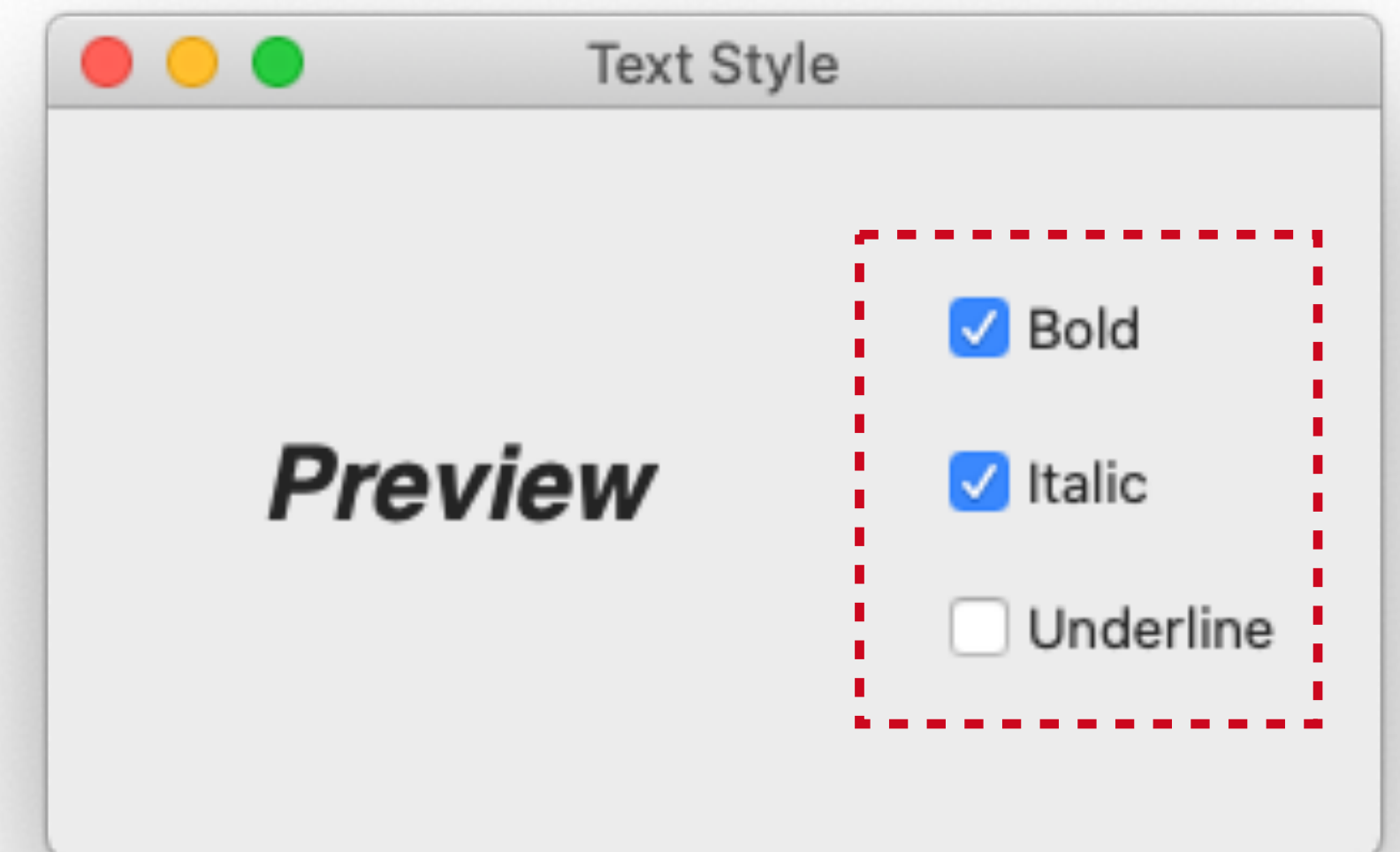
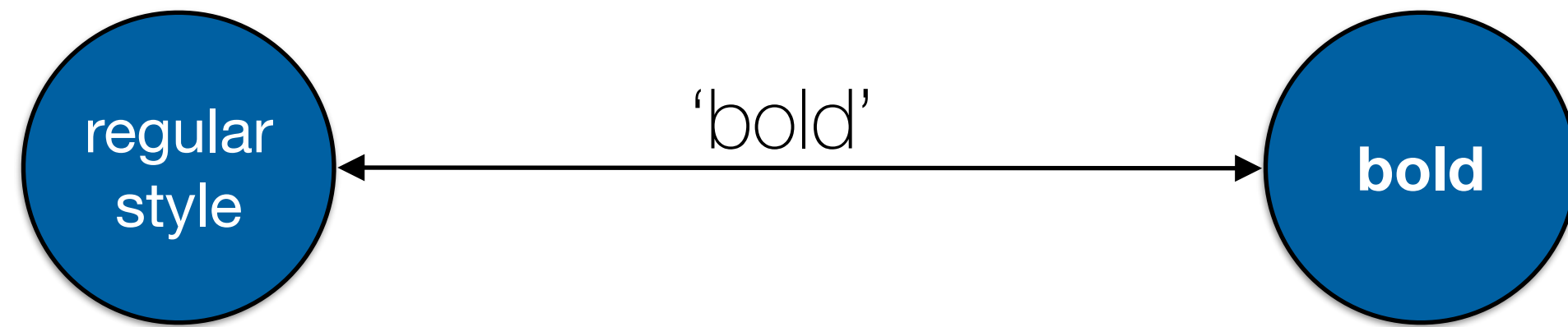
# In-Class Exercise: STN



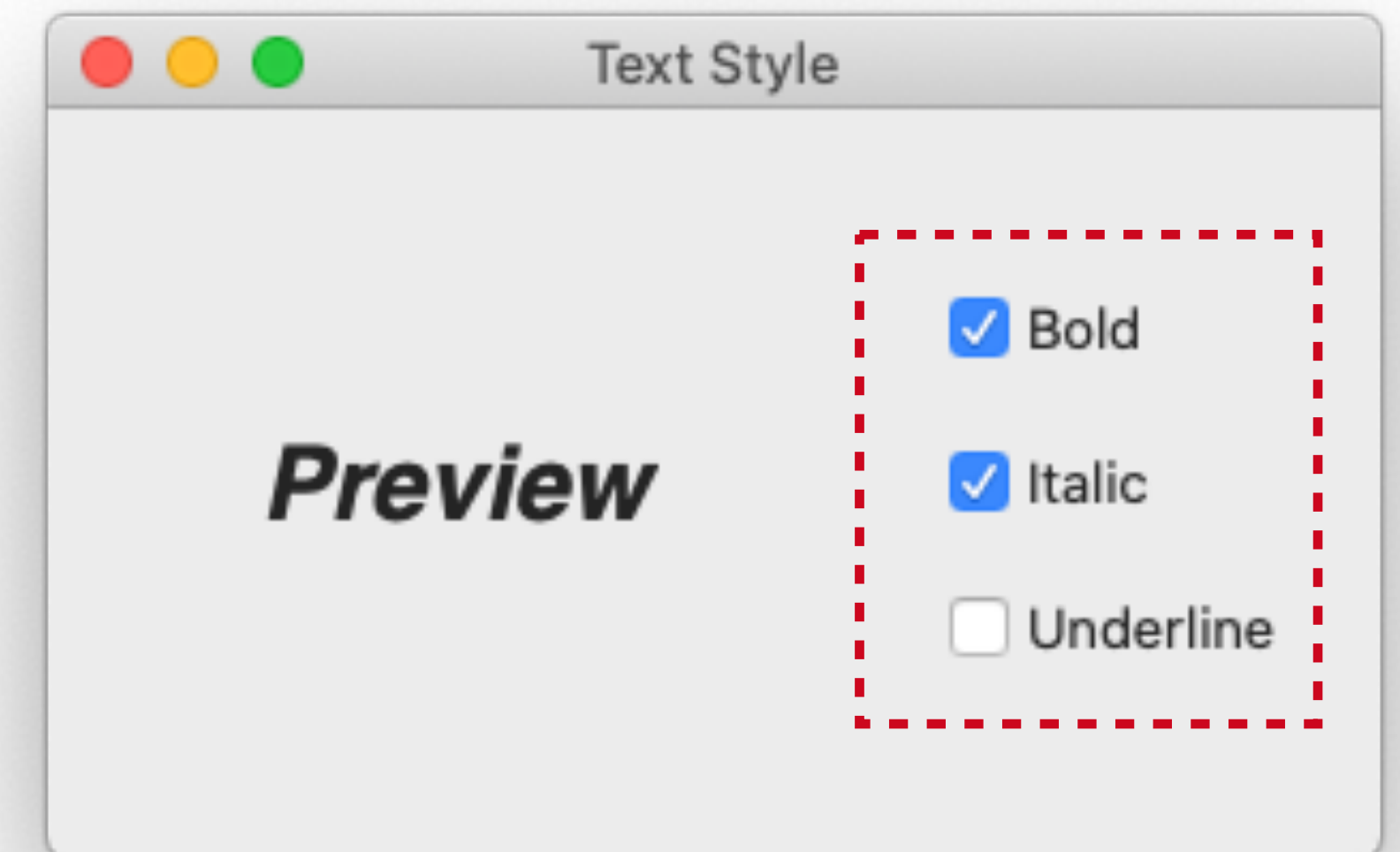
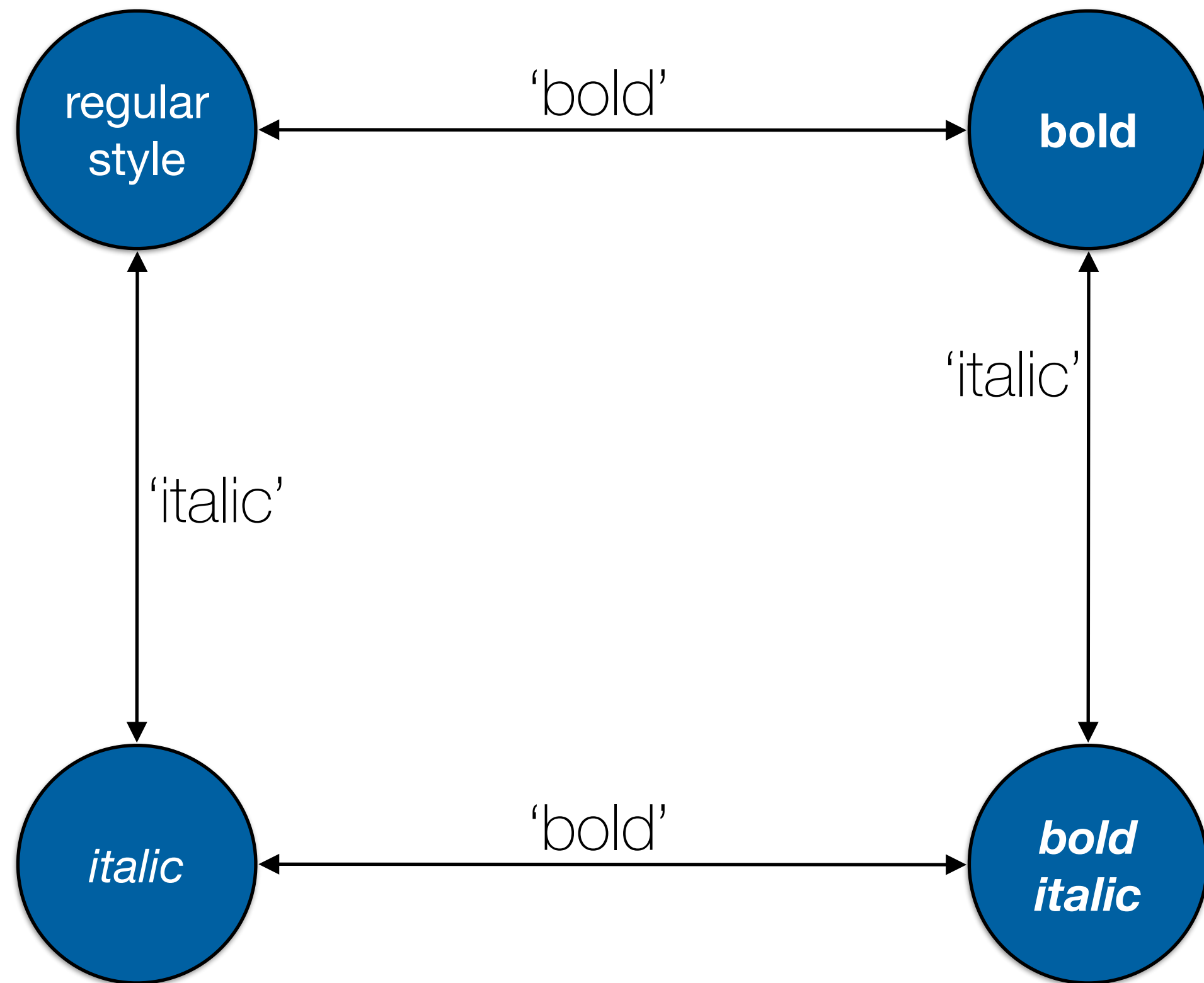
- Simple dialog to select bold, italics, and/or underline
- Draw the state diagram for:
  - Only Bold checkbox
  - Bold and italics checkboxes
  - All three checkboxes



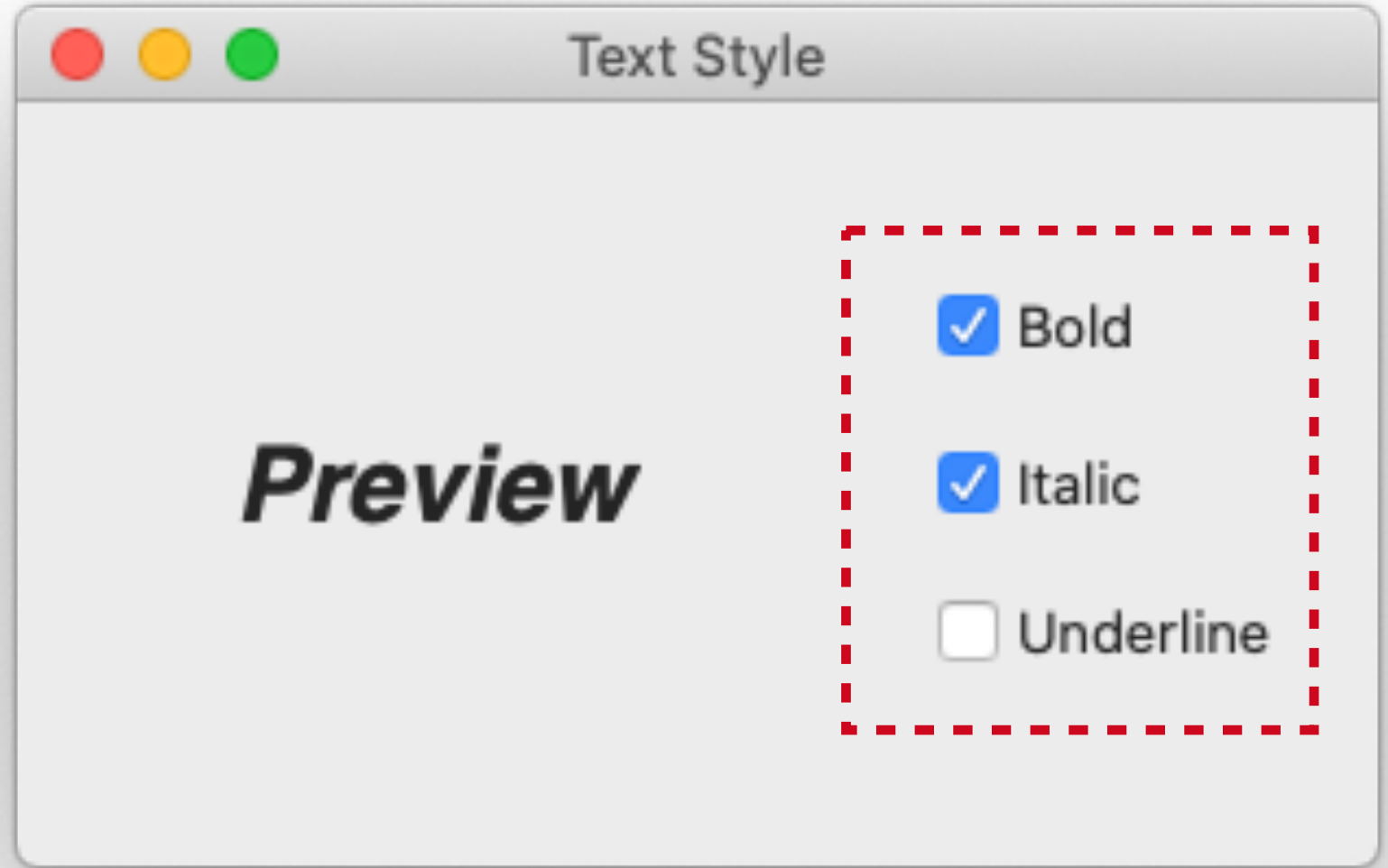
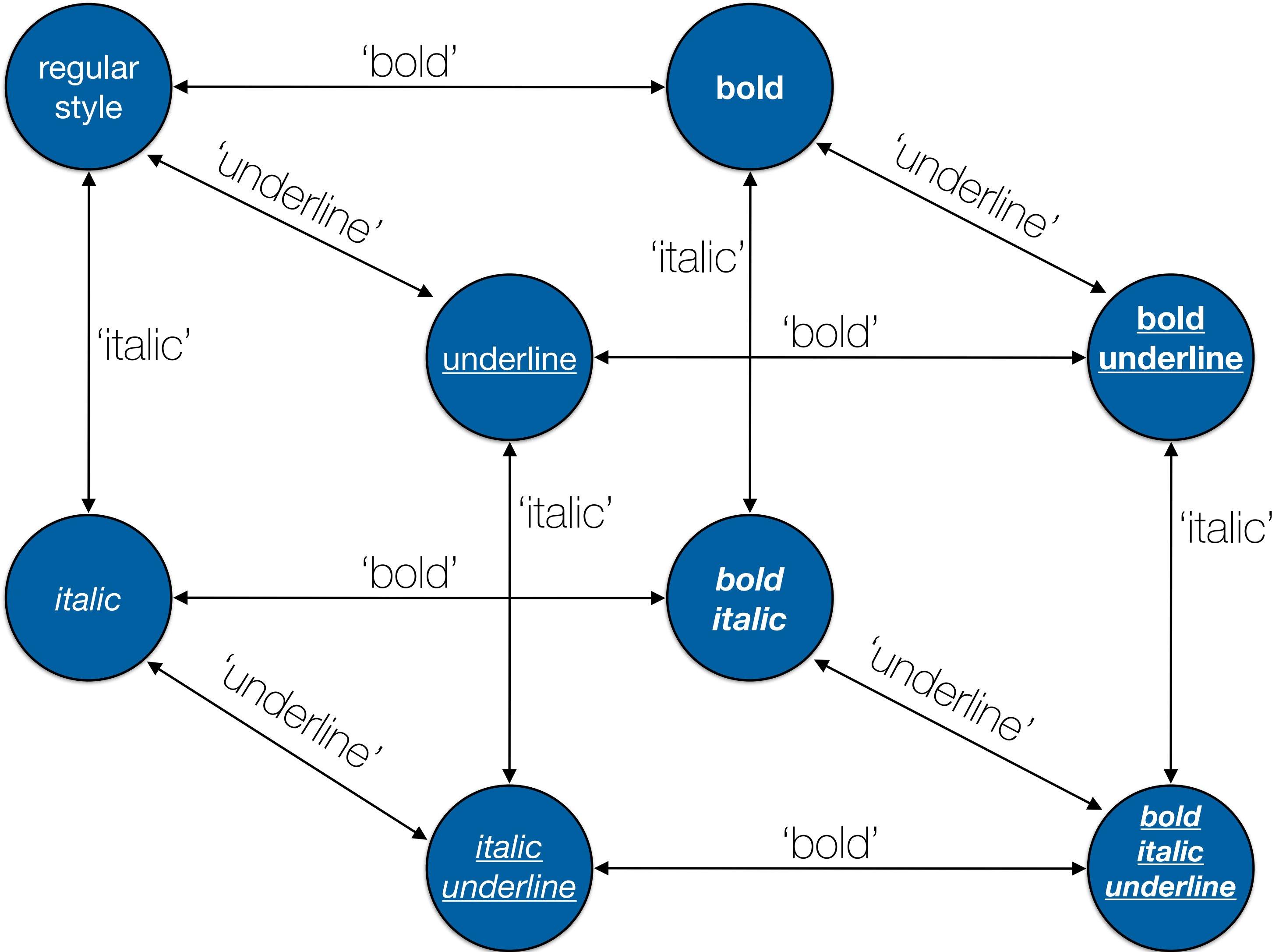
# Bold Checkbox



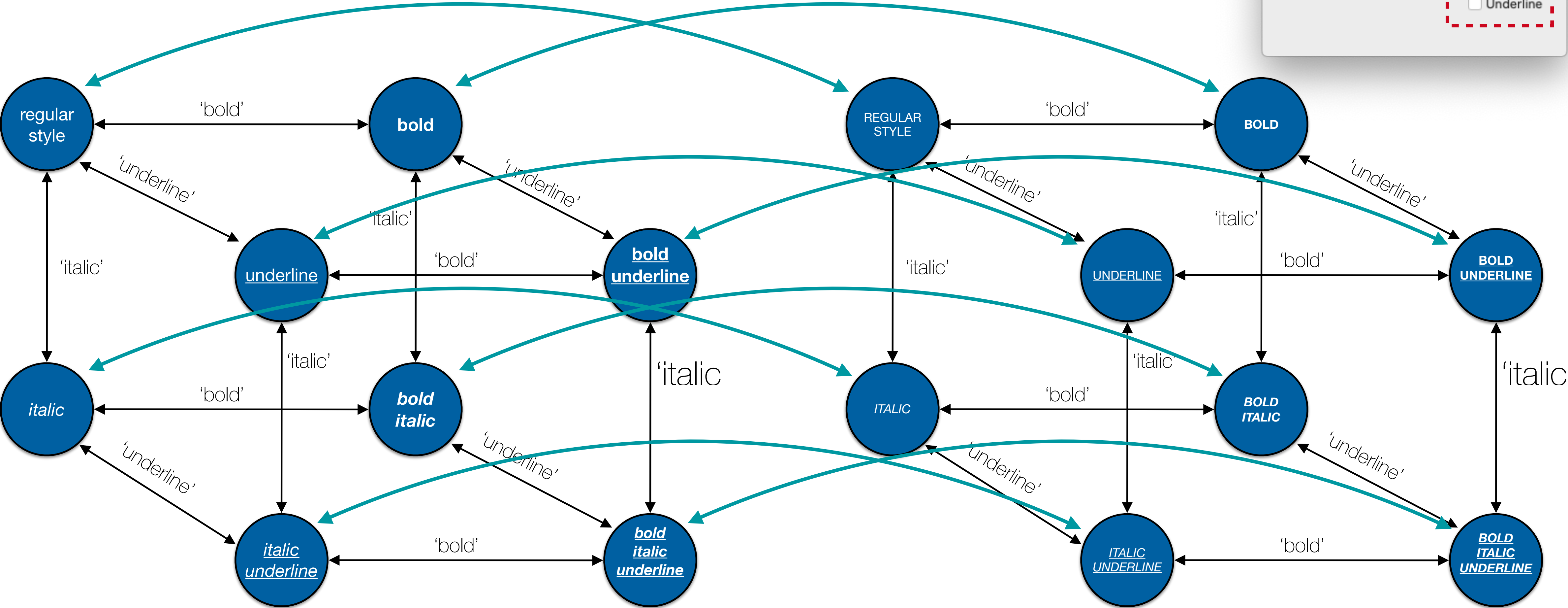
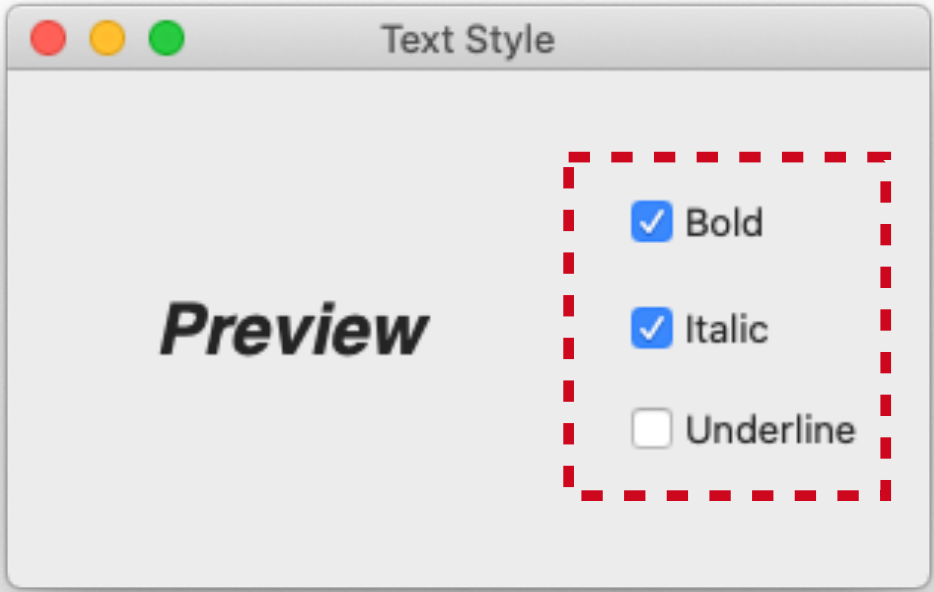
# Bold & Italic Combined



# All Three Options



# Adding Another Option...



Normal case

CAPITALIZED

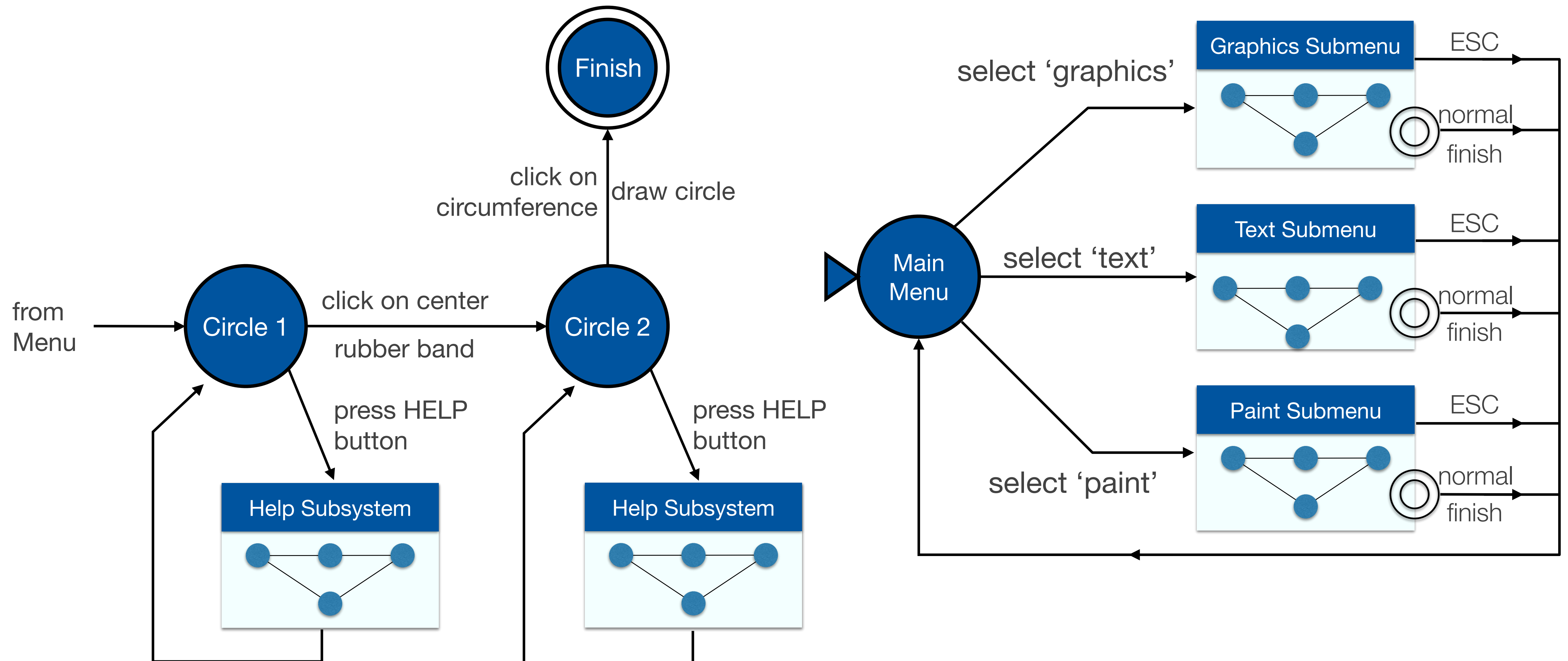


# STNs: State Explosion

- Problem: Combining two concurrent STNs with N and M states leads to new STN with  $N \times M$  states
- STN hides clear structure of the dialog
- Especially problematic with modern GUIs
- Similar problems with “Escape” and “Help” options
  - ESC can be modeled as special second “Finish” exit active throughout subdialog
  - Help can be modeled as little subdialog hanging off every single state in the STN
  - Gets messy



# Example: ESC & Help in STNs



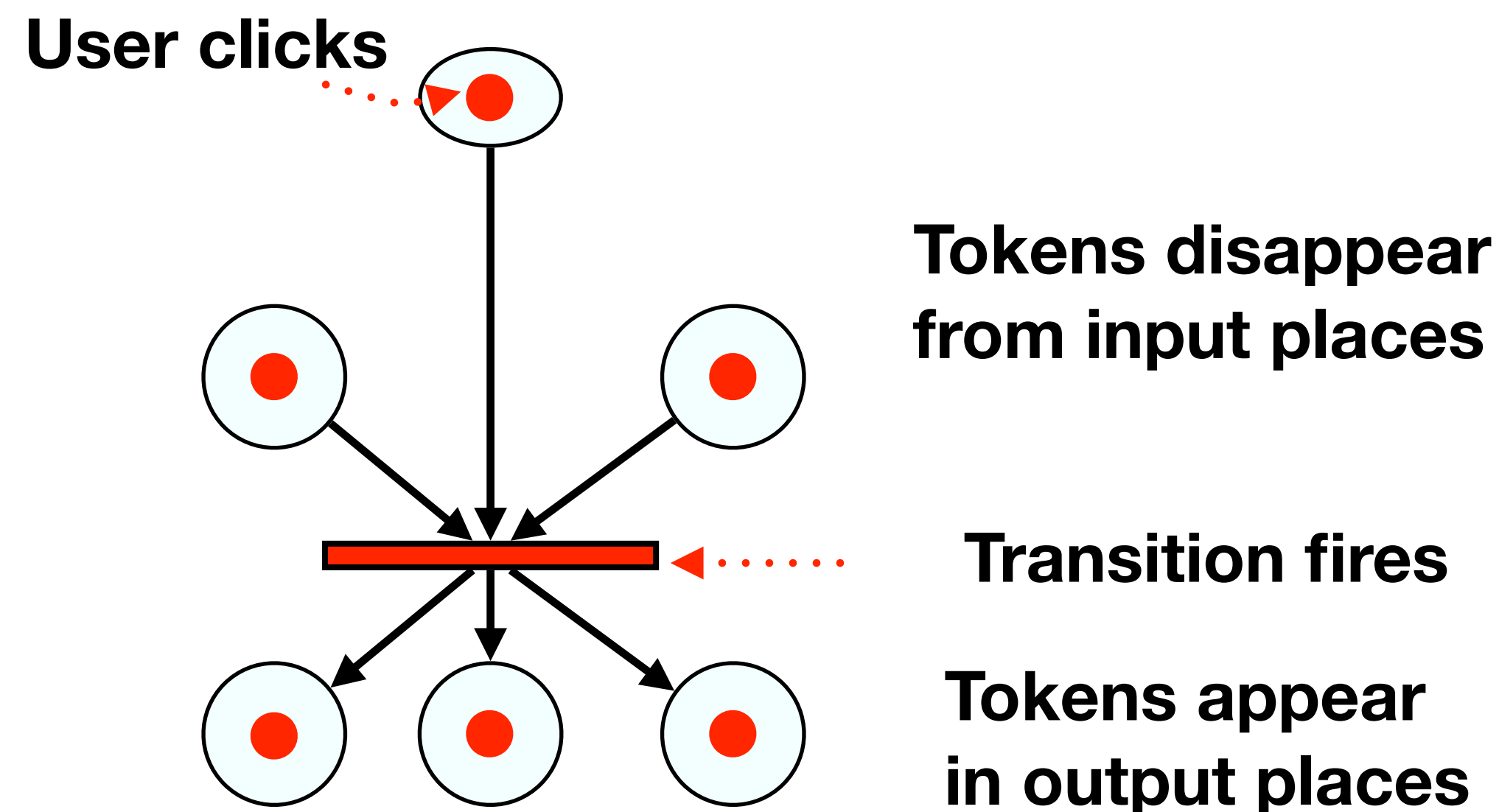
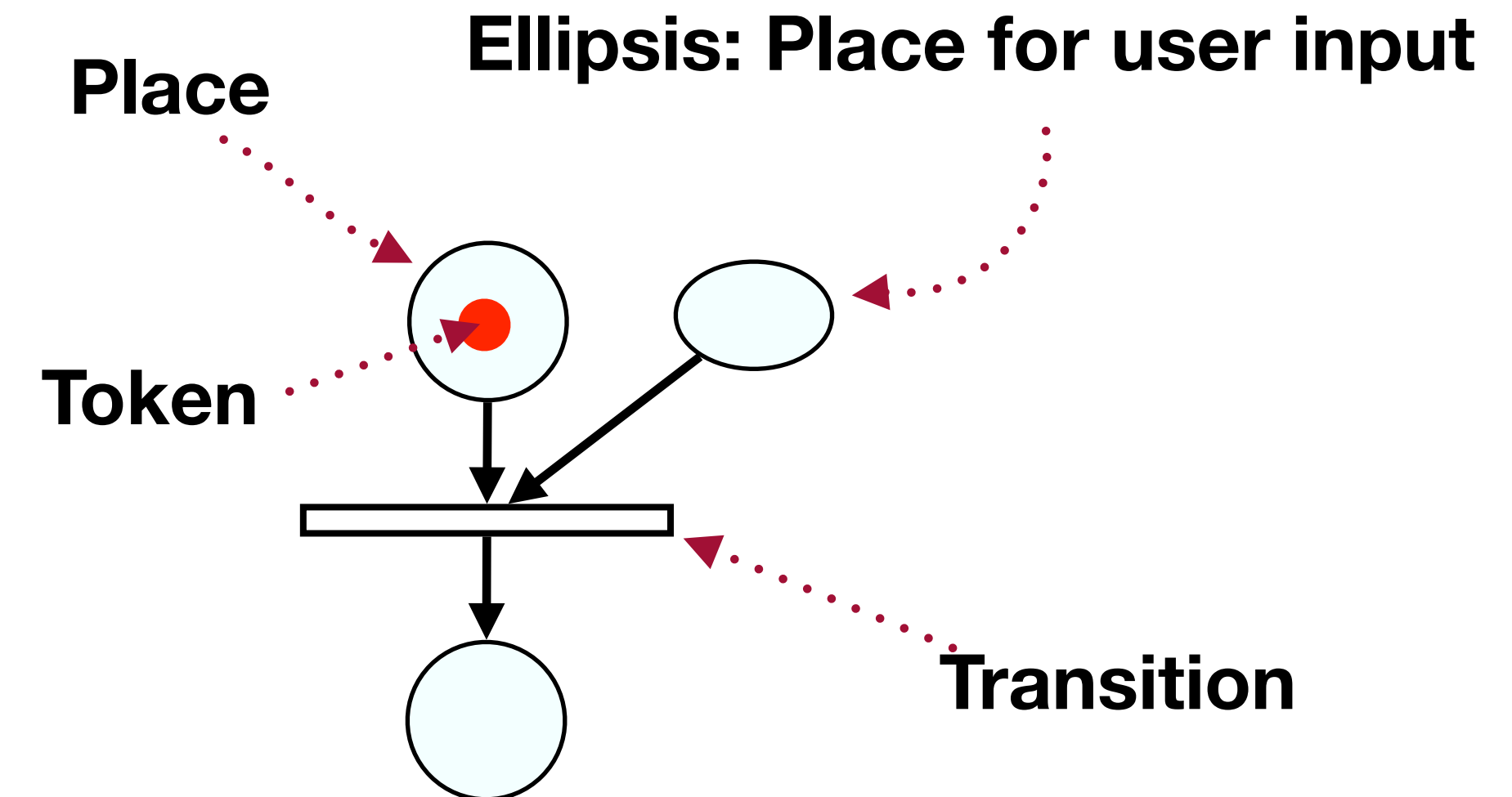
# Petri Nets

- Better approach to dialogs that have several states at once
- But not better for sequential dialogs and mutually exclusive UI elements (radio buttons)
- Relatively old formalism to model concurrency



# Petri Nets

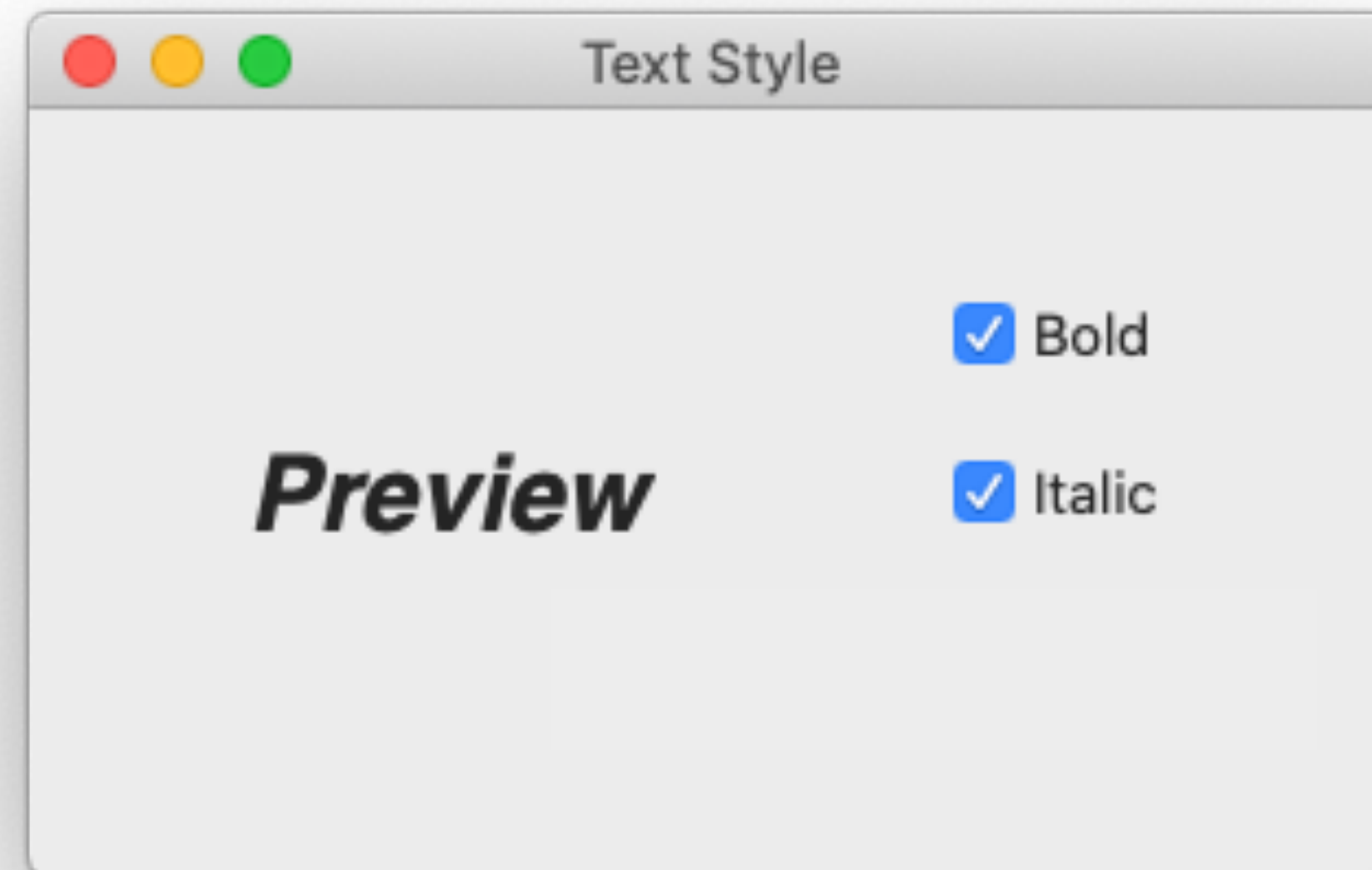
- Transition fires when all input places have one or more token
  - A token is produced in each output place
- Positions of all tokens represent the current state
  - NOTE: This is different from state machines



# In-Class Exercise

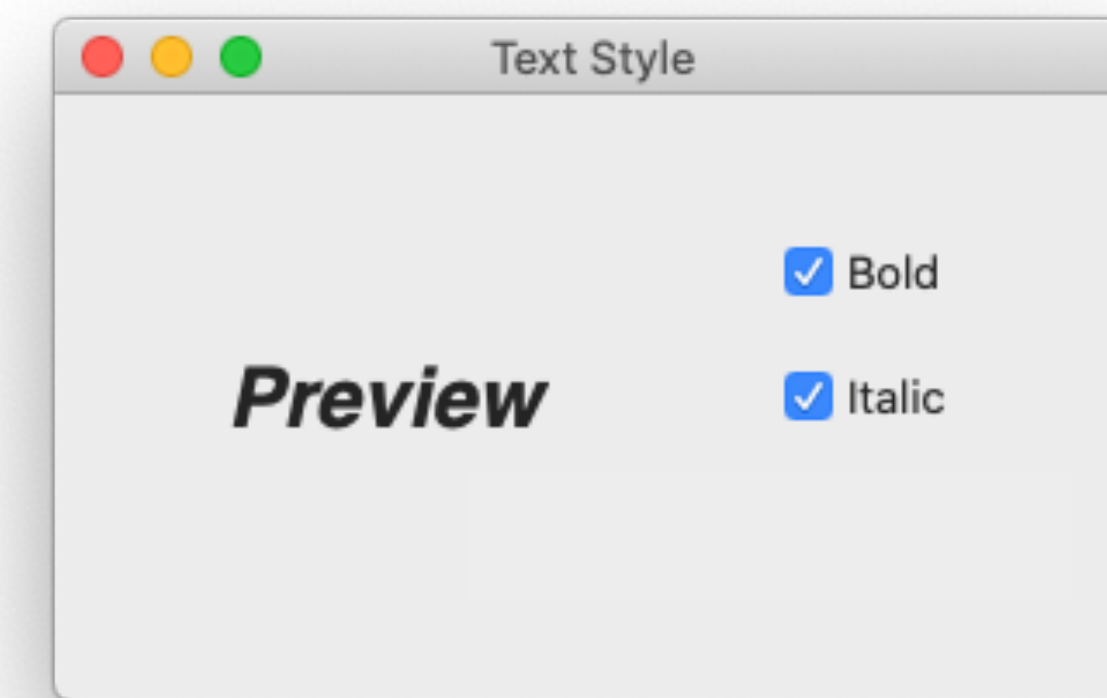


Draw the Petri net for our dialog box with concurrent “Bold” and “Italic” options (ignore “Underline” for now)

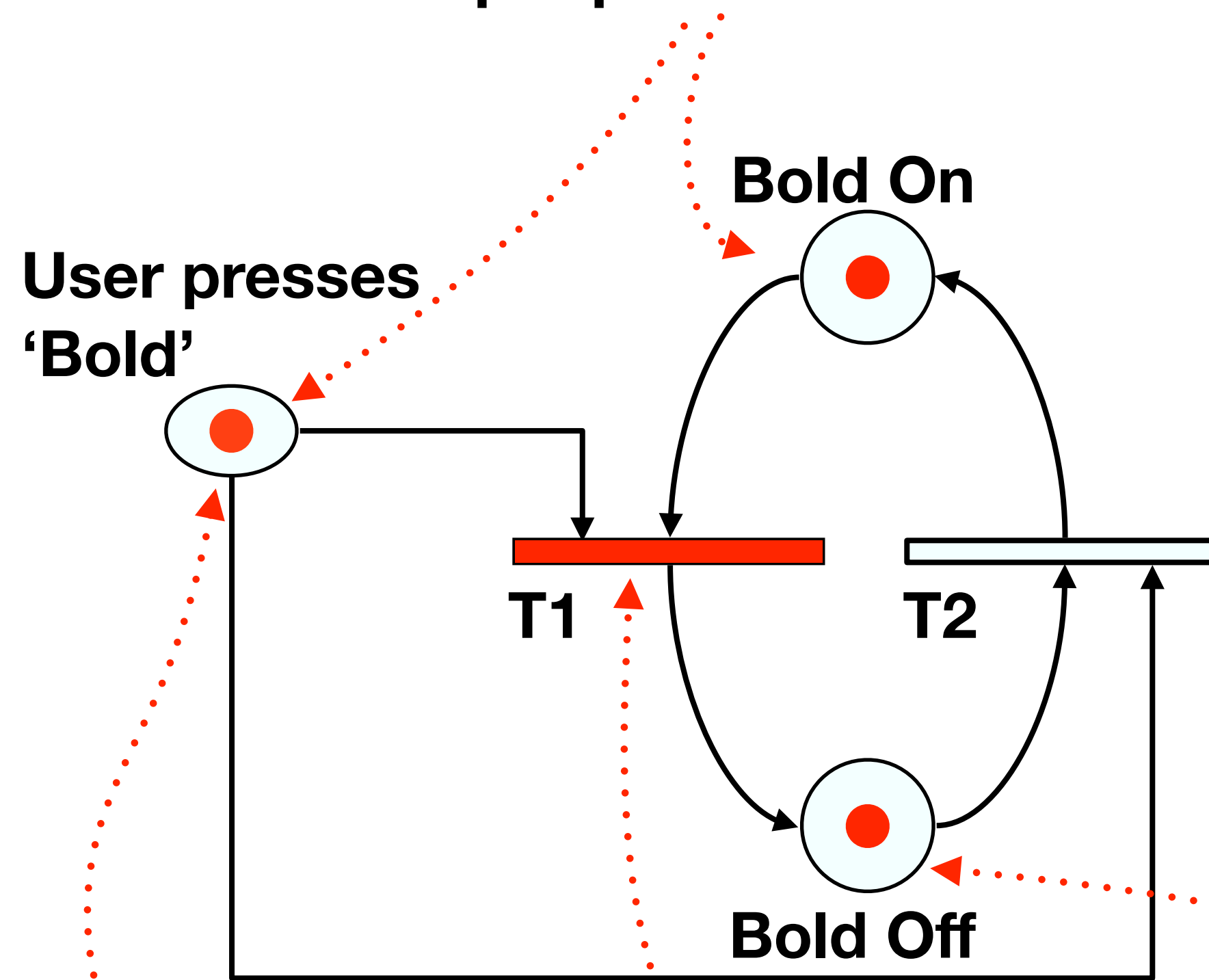




# Petri Net For "Bold & Italic" Dialog

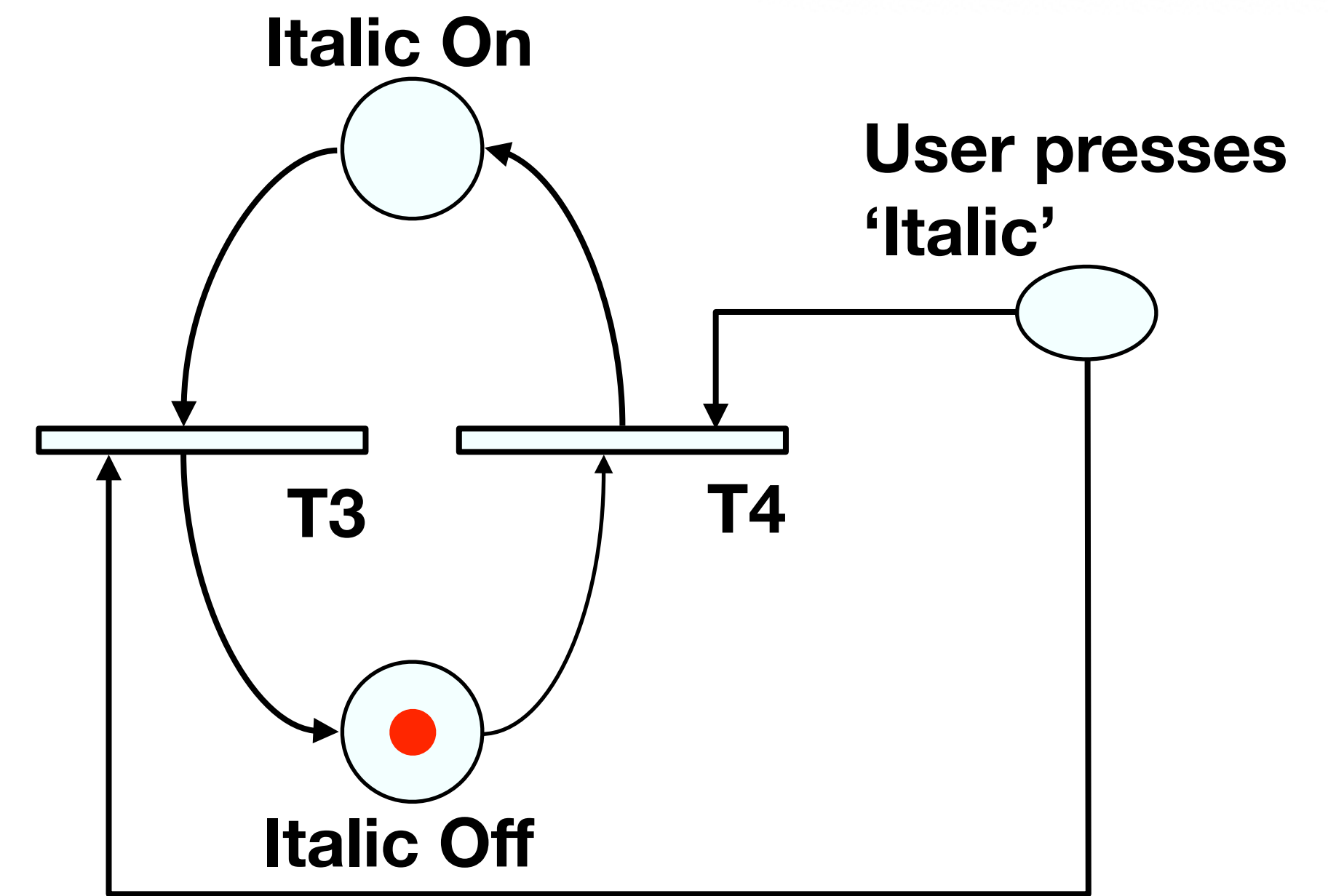


Tokens are consumed from all input places



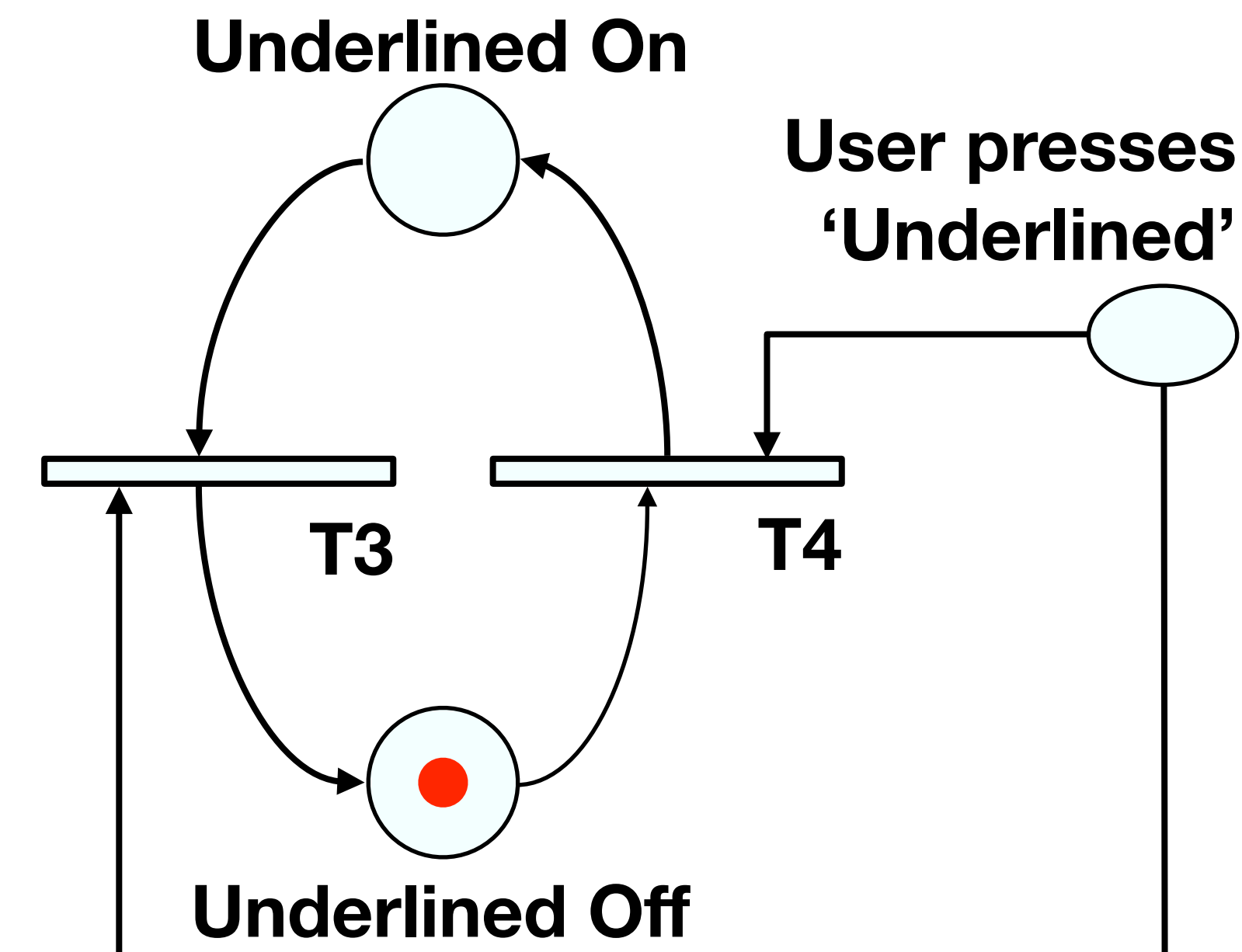
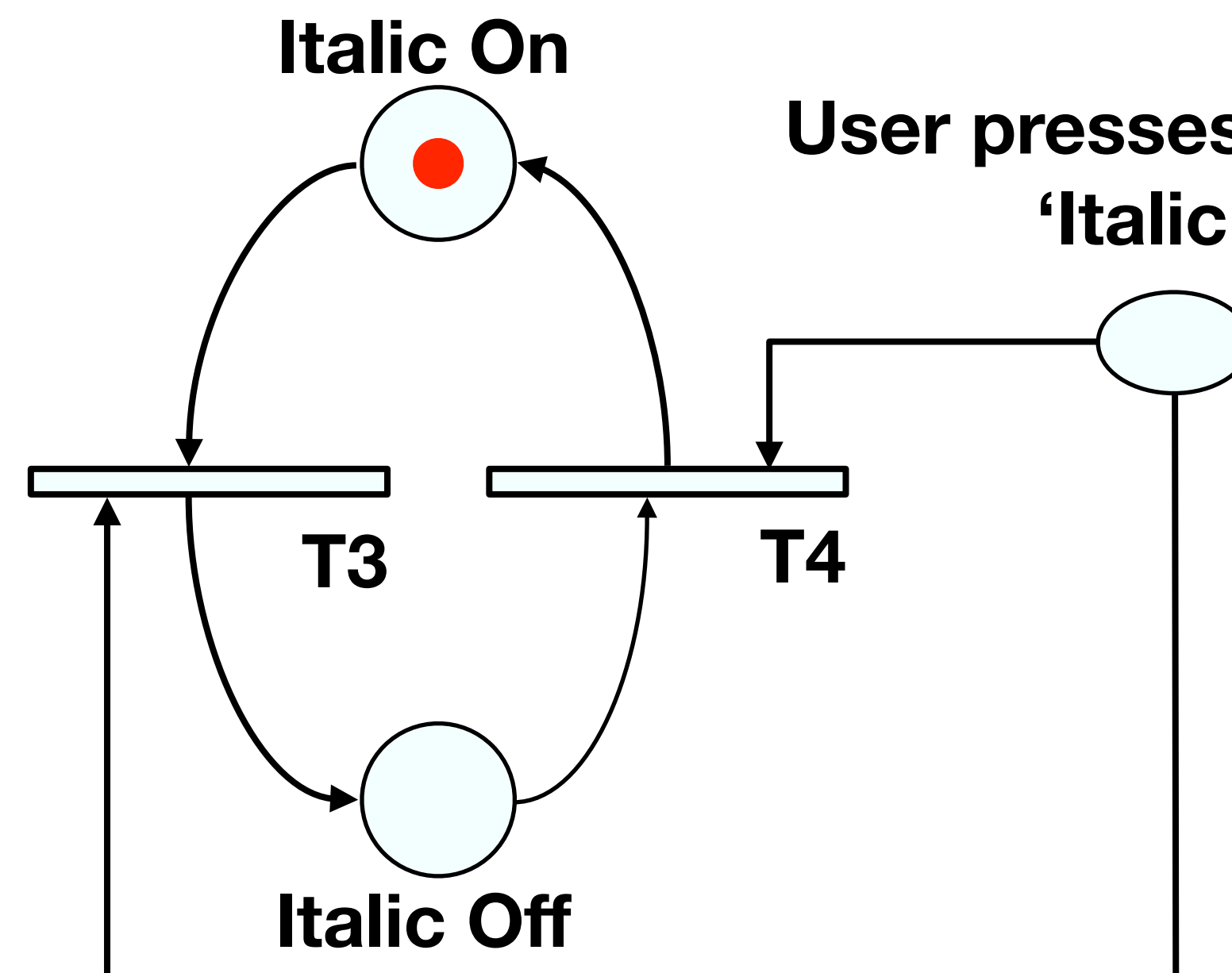
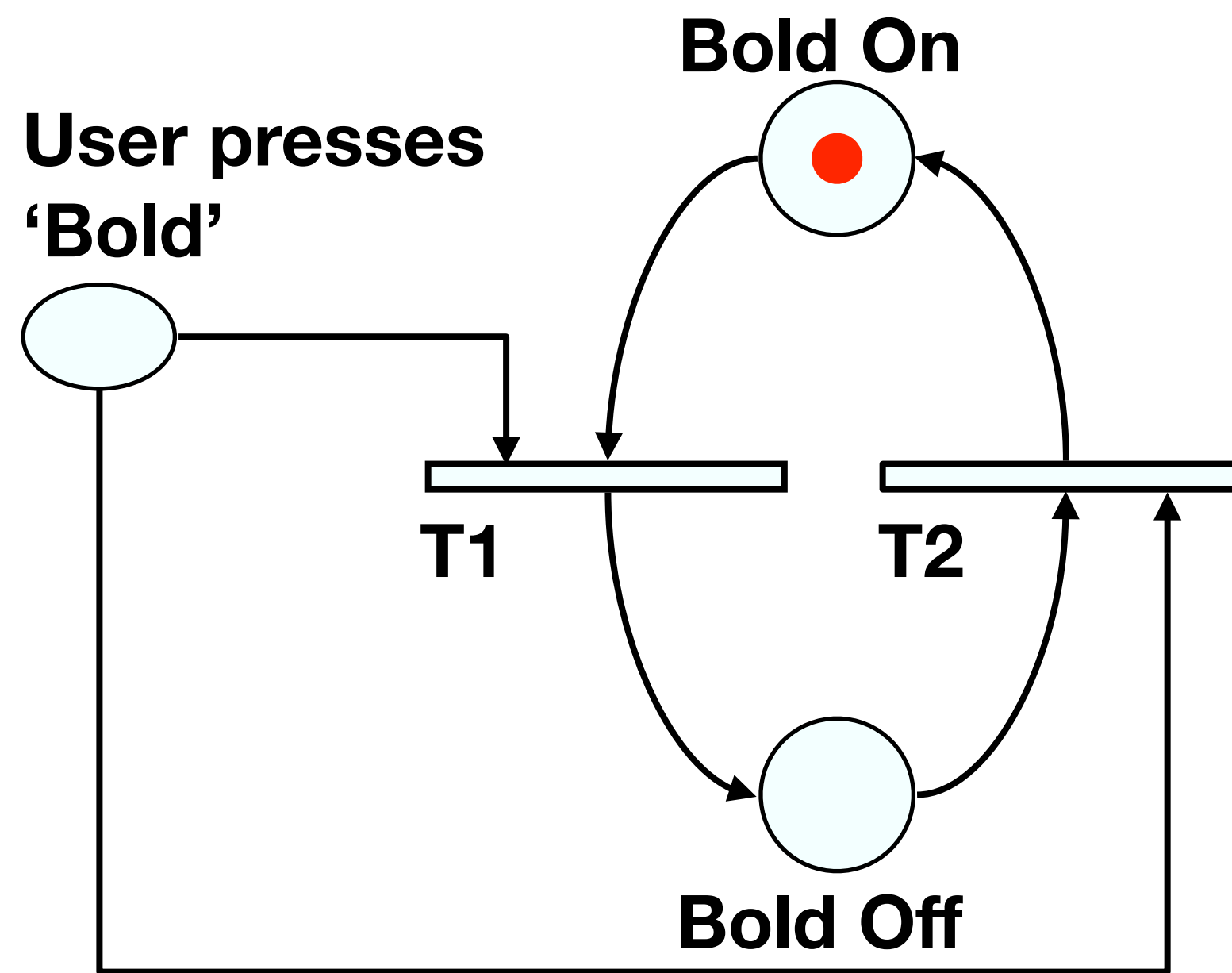
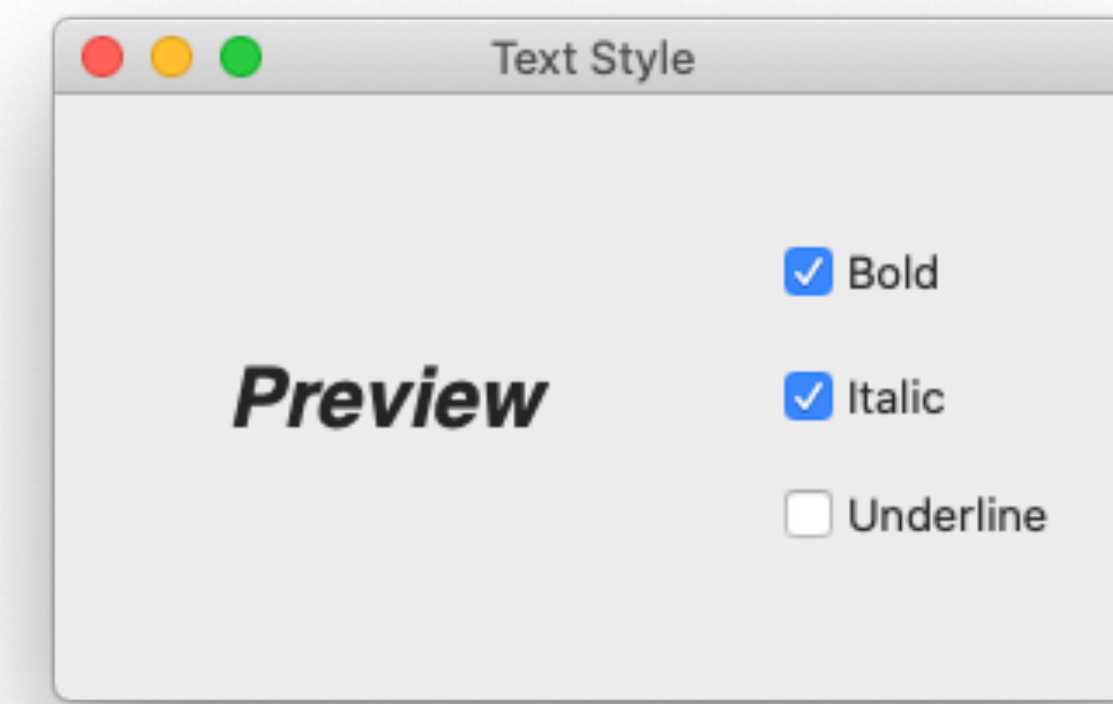
User action represented as a new token

Transition 'fires' when all input places have tokens



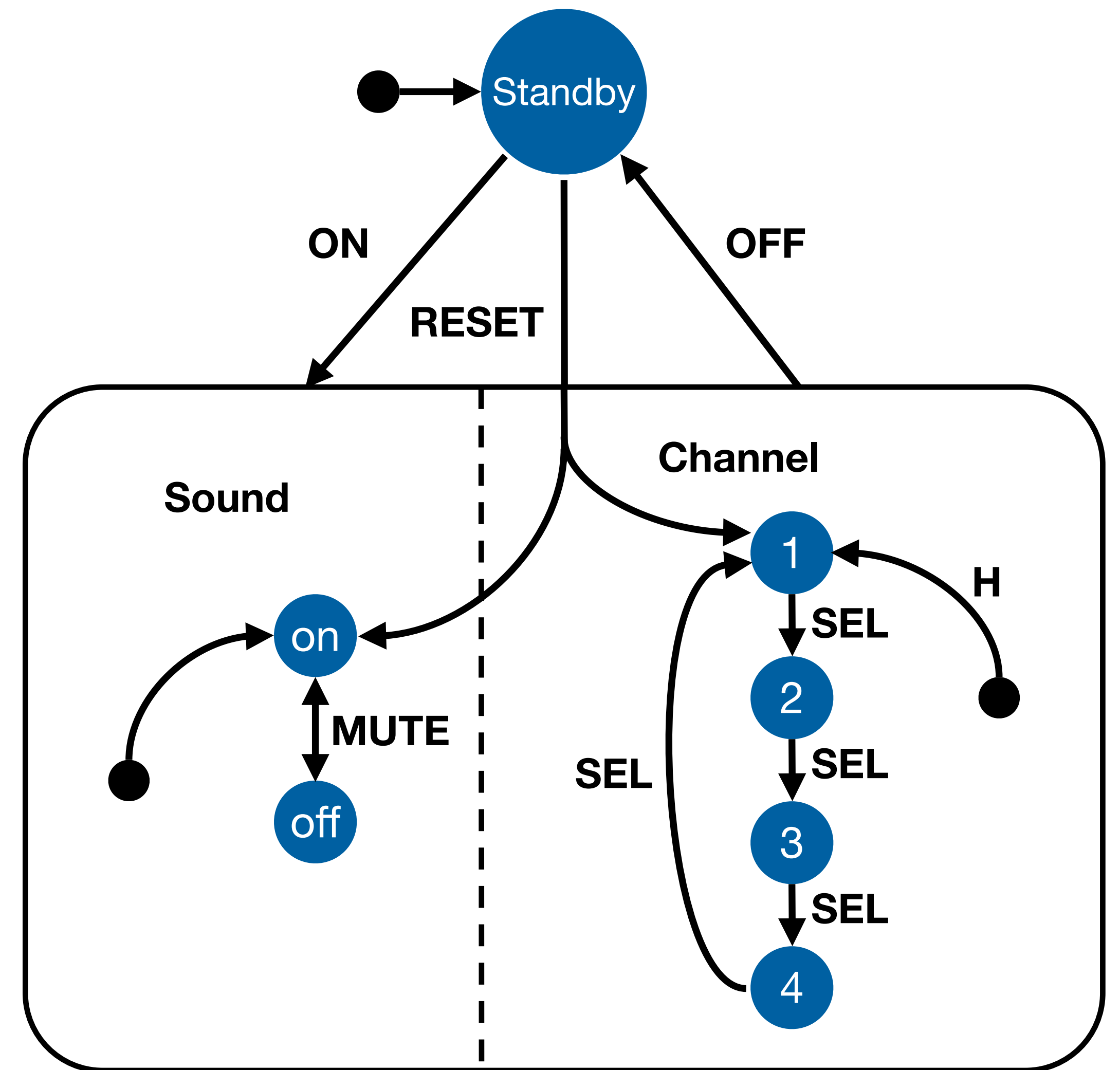
A token is produced in each output place

# Petri Net For “Bold & Italic & Underlined” Dialog



# State Charts

- By Harel; used in UML
- Example: TV Control Panel
- State Charts extend STNs
  - Hierarchy
  - Concurrent sub-nets
    - ON resumes both state machines
  - Escapes
    - OFF always active
  - History
    - Link marked “H” goes back to last state on re-entering subdialog



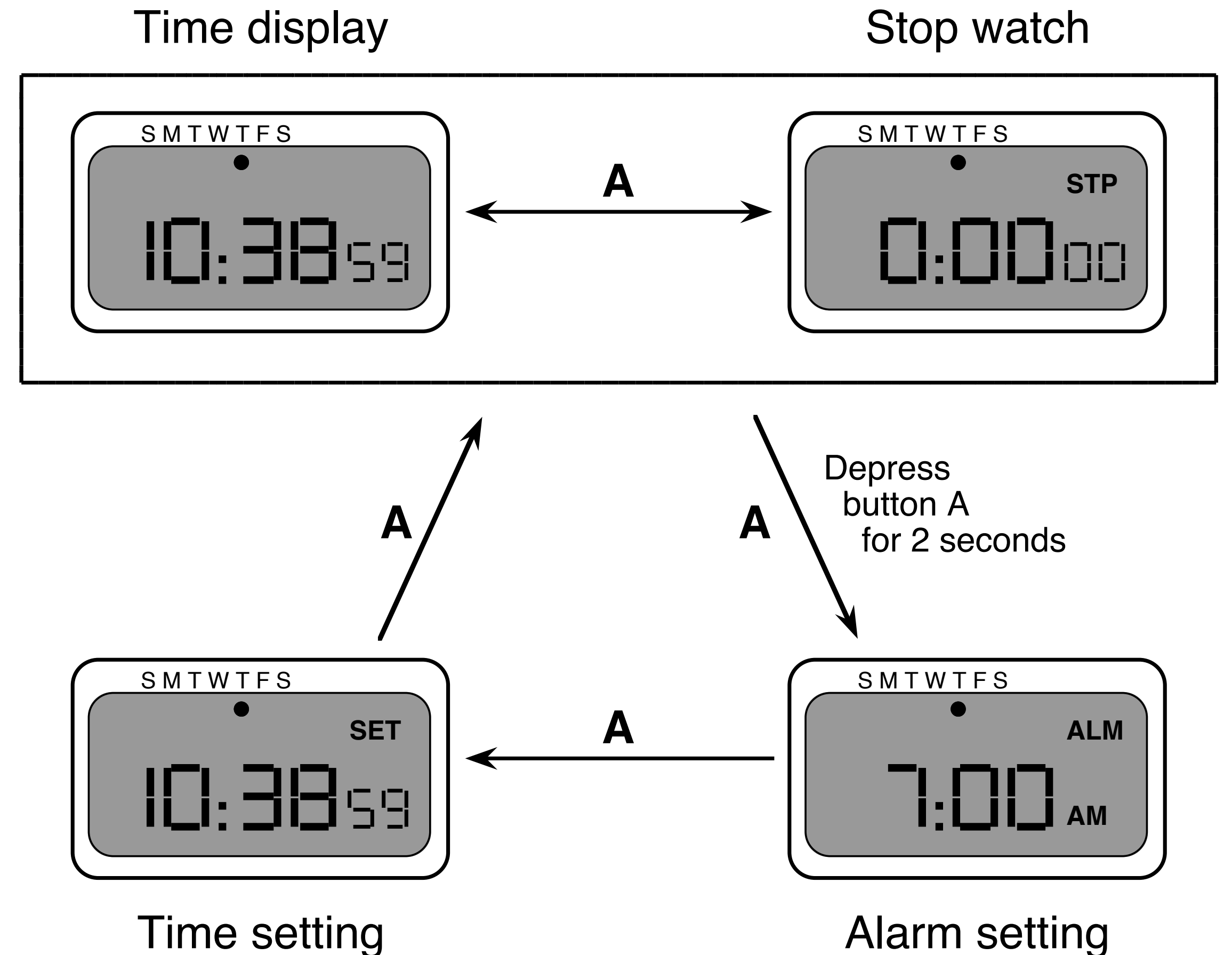
# Diagrams For User Documentation

- Some dialog descriptions are clear enough to serve as user documentation (similar to GOMS)
- Especially if description uses screen shots and is semi-formal



# Digital Watch – User Instructions

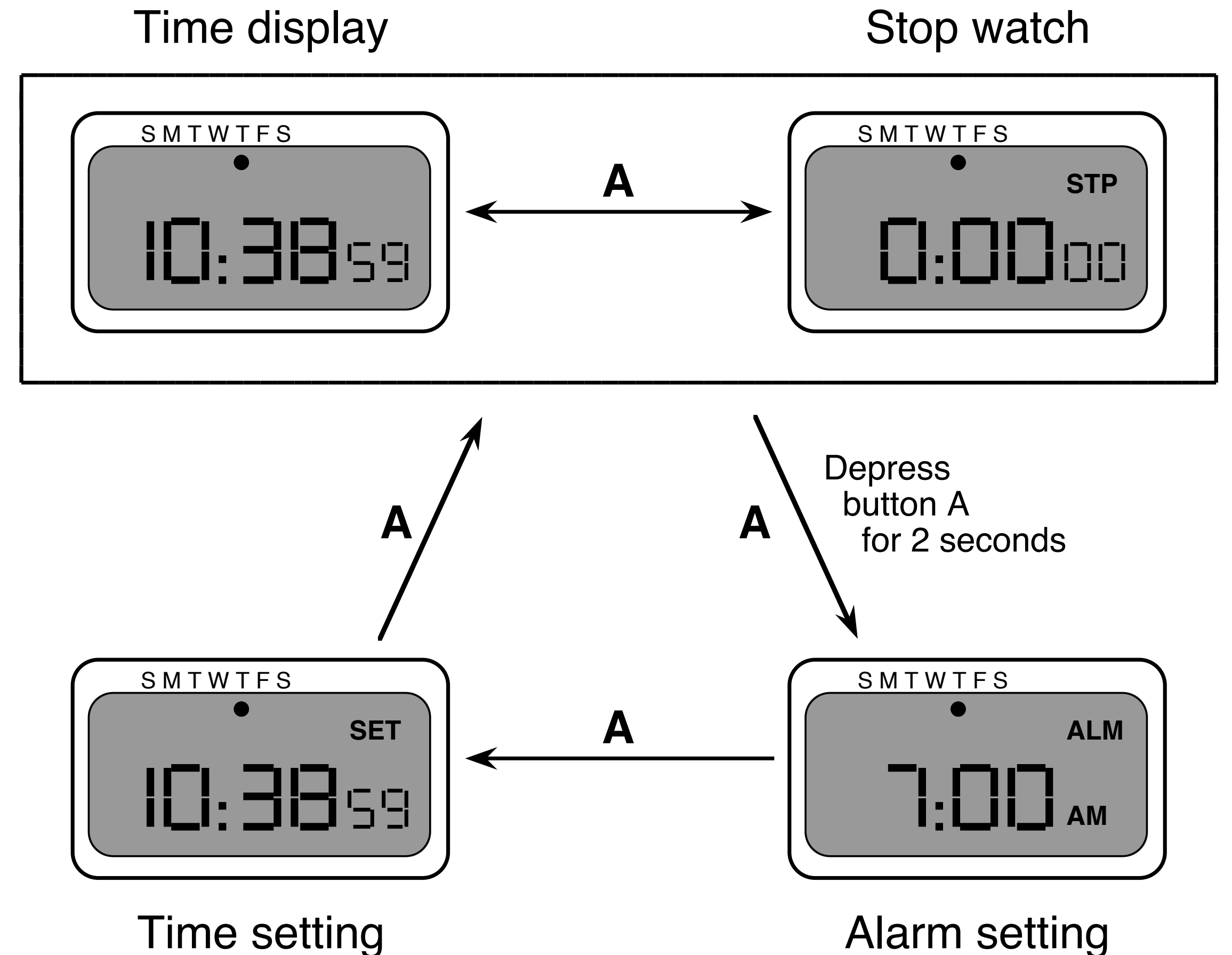
- Two main modes
- Limited interface
  - 3 buttons
- Button A changes mode





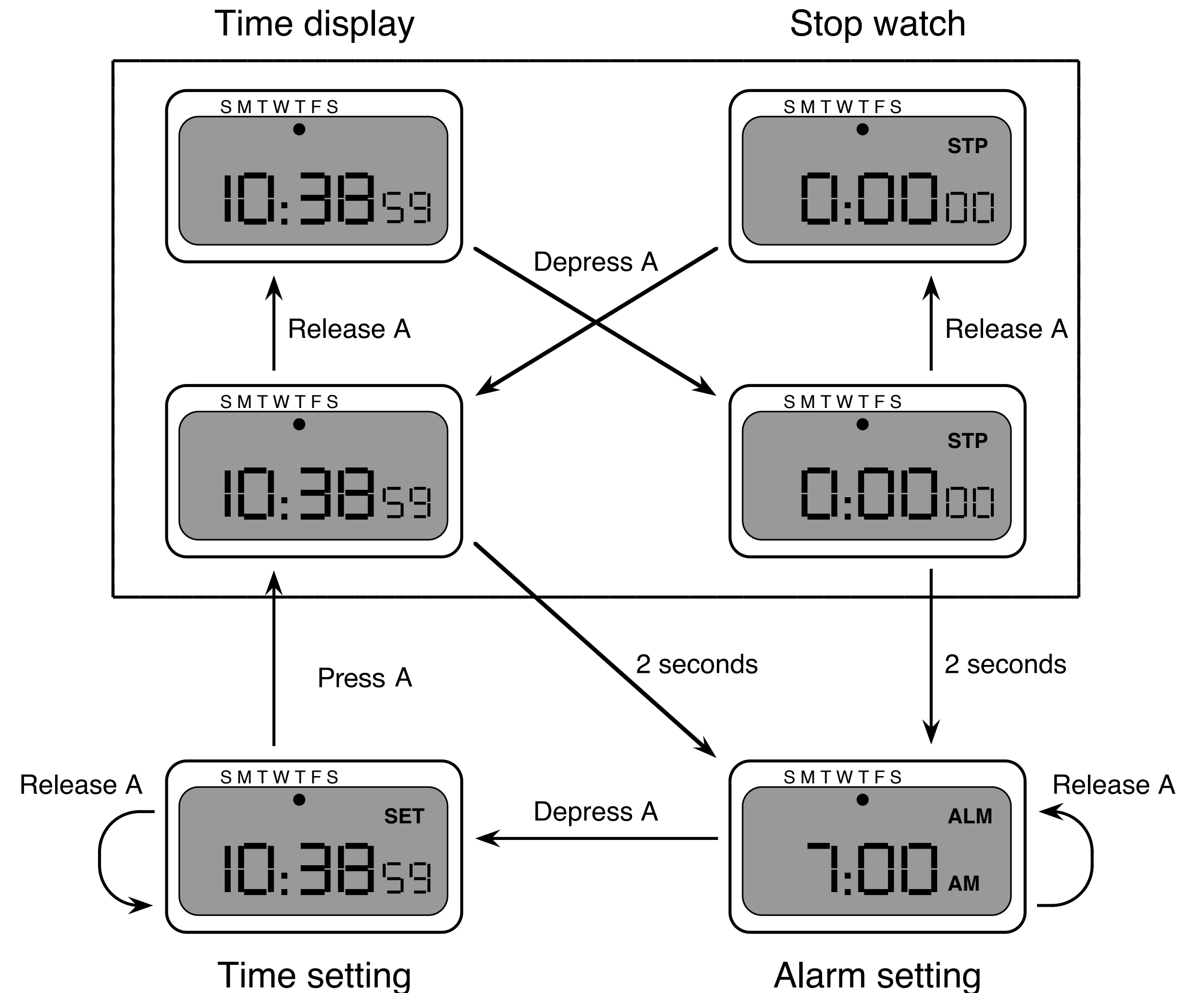
# Digital Watch – User Instructions

- Dangerous states
- Completeness
  - Distinguish depress A and release A
  - What do they do in all modes?



# Digital Watch – User Instructions

and... that's just one button



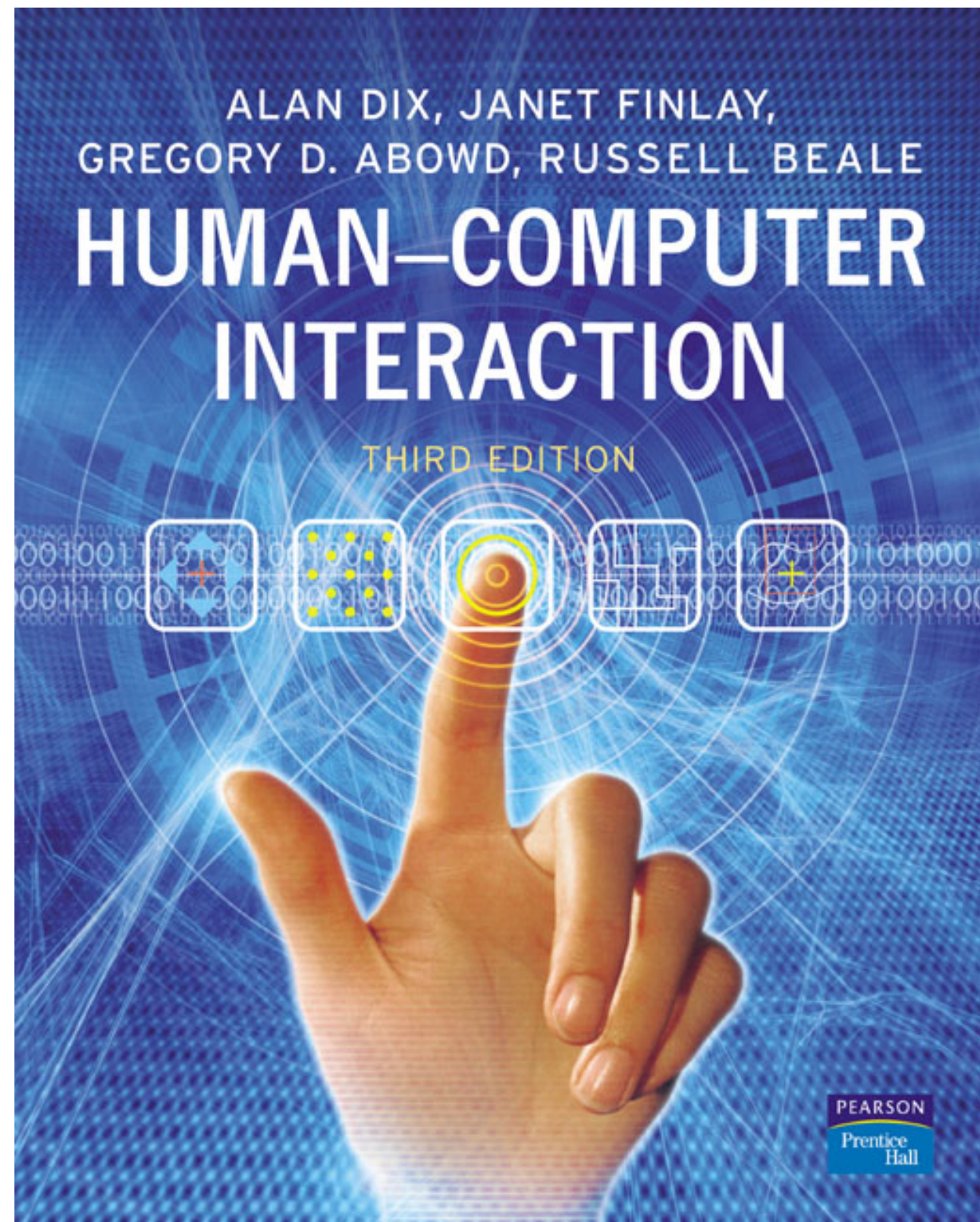
# Semantics - Raw Code

- Event loop for word processor
- Dialogue description: very distributed
- Syntactic/semantic trade-off: terrible!

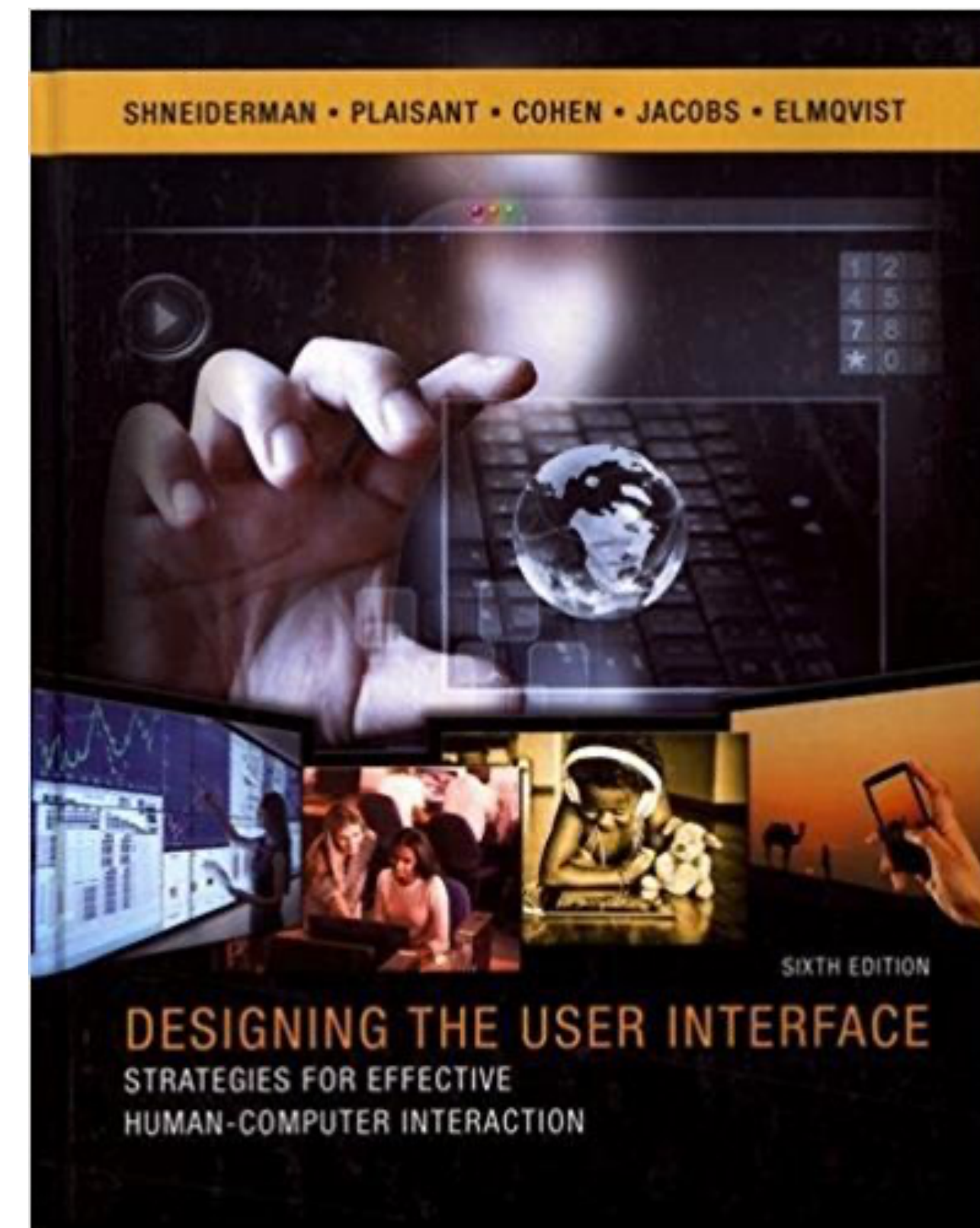
```
switch ( ev.type ) {
  case button_down:
    if ( in_text ( ev.pos ) ) {
      mode = selecting;
      mark_selection_start(ev.pos);
    }
    ...
  case button_up:
    if ( in_text ( ev.pos )
        && mode == selecting ) {
      mode = normal;
      mark_selection_end(ev.pos);
    }
    ...
  case mouse_move:
    if (mode == selecting ) {
      extend_selection(ev.pos);
    }
    ...
} /* end of switch */
```



# Further Reading



Alan Dix et al.:  
**Human-Computer Interaction**,  
3rd ed. (2003), Chapter 16

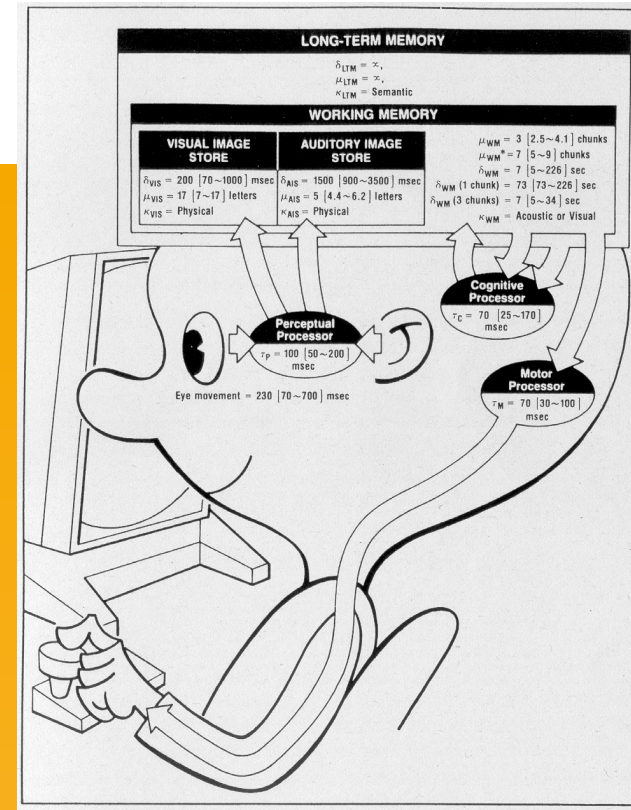


Ben Shneiderman:  
**Designing The User Interface**,  
6th ed. (2017), esp. Chapter 5

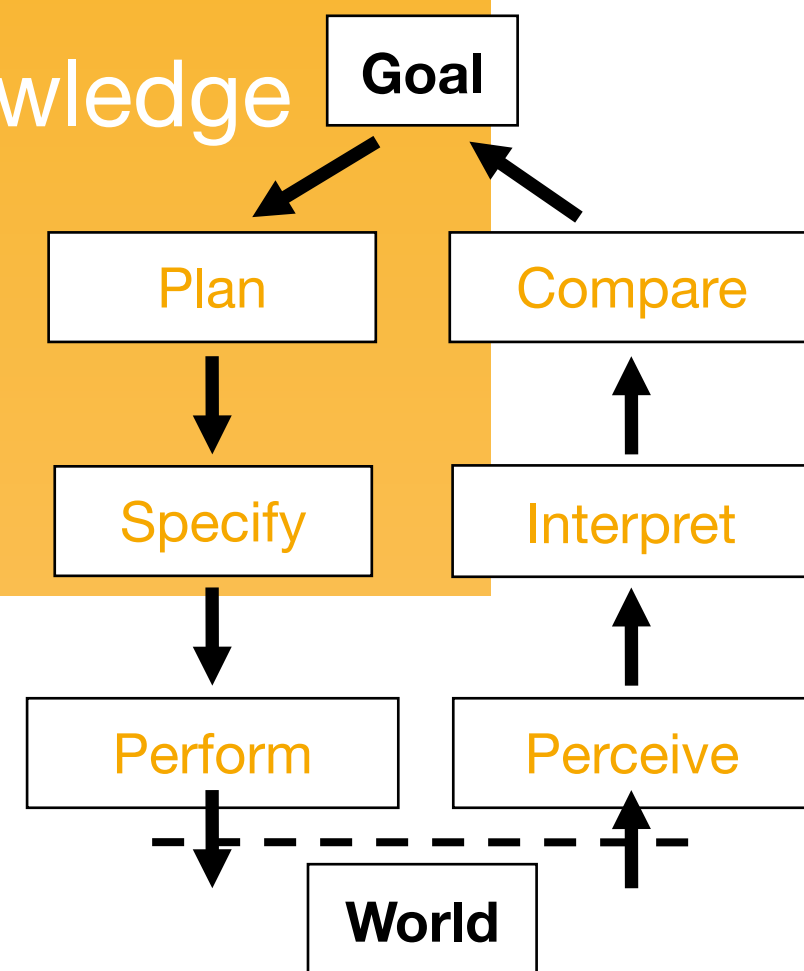


# Roadmap

## Human



- Performance
- Models of interaction
  - Affordances
  - Mappings
  - Constraints
  - Types of knowledge
  - Errors
- Visual Design



## Case Studies

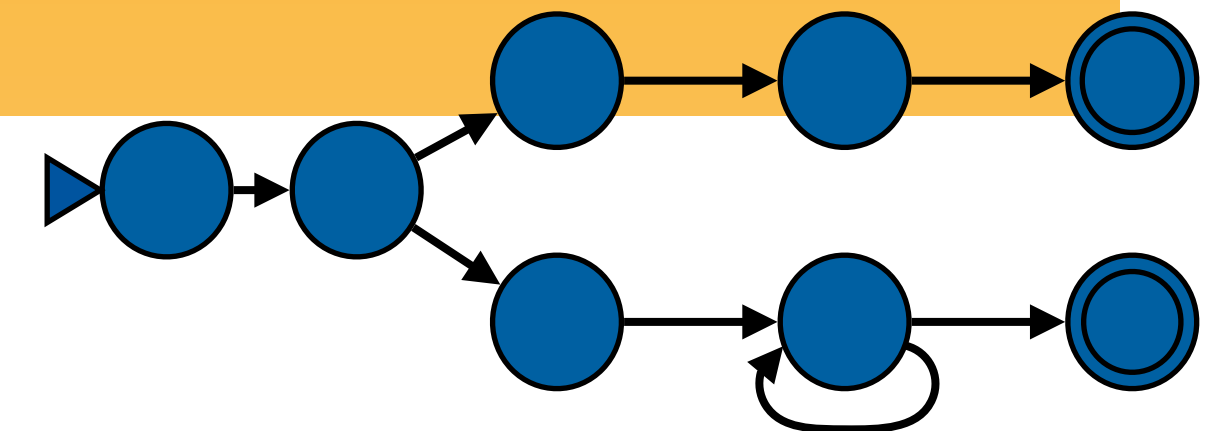
- History of HCI
- Visions
- Technology Phases



## Development Process



- Iterative design
- User observation
- Ideation
- Prototyping
- User studies and evaluation
- Interaction design notation



# What's Next?

- **Designing Interactive Systems 2** (6 ECTS)

<https://hci.rwth-aachen.de/dis2>

- What makes a UI tick?

- Technical concepts, software paradigms and technologies behind HCI and user interface development

- **Current Topics in HCI** (6 ECTS)

<https://hci.rwth-aachen.de/cthci>

- Understand & practice ways to do research in HCI

- Learn about the latest research in HCI from recent conference and journal articles (and meet our Ph.D. students!)

Interested in a HiWi position or  
B.Sc./M.Sc. thesis?  
<https://hci.rwth-aachen.de/jobs>