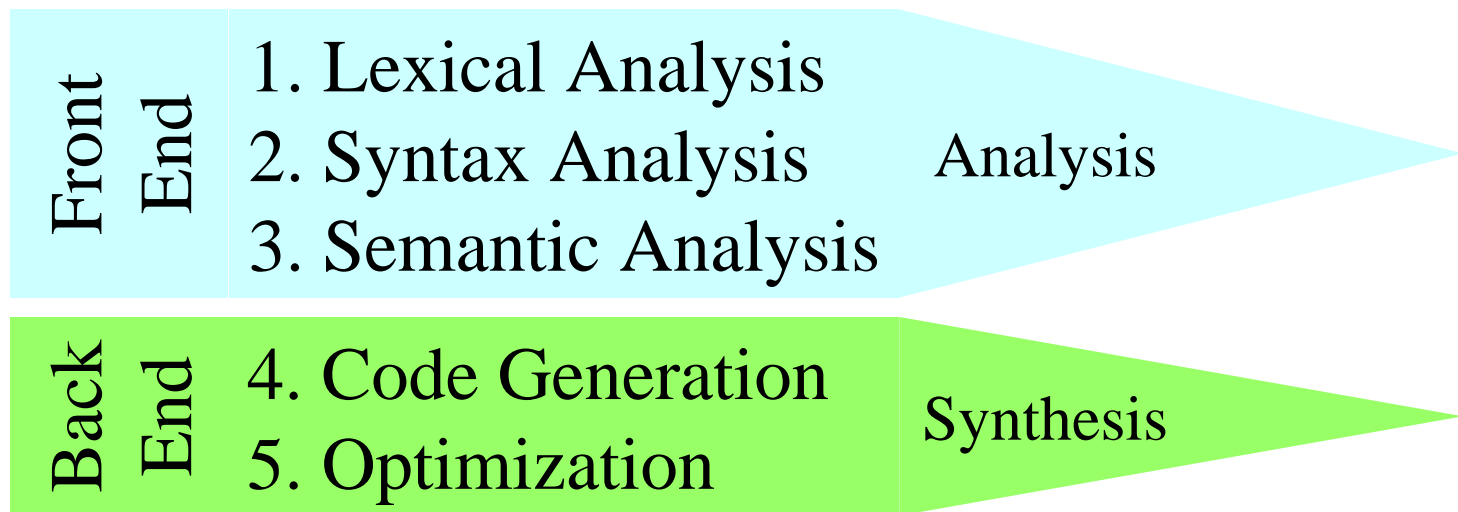


CS 3240

Presentation 3 Regular Expressions (Scanners)

Parts of Compilers



Lexical Analysis

- Read input one character at a time
 - Group characters into tokens
 - Remove whitespace, comments
 - Encode token types
 - Some (minimal) syntax checking
 - Detect errors/Generate error messages
-
- A lexical language tells us which are legal strings in that language
 - Lexical rules specify how alphabet can be combined to form legal strings

Tokens

- Identifiers
 - start with letter or "_" continue with 0 or more letters or numbers or "_"
- Keywords
 - reserved words that are part of language (if, else, return)
 - Note: A language doesn't have to have keywords
- Operators
 - + - * / % & && | ||
- Literals
 - Numeric, string, etc.
- Special symbols
 - ; { } ()

Regular Expressions

- Ways to represent strings of characters which follow certain rules
 - Regular Expression (a formula for matching strings that follow some pattern)
 - State Machine (Finite automata)
 - Computer Program
- Two notations
 - Classical
 - Unix style (recognized by grep/vi/etc.)

Regular Expressions

- Single character
 - Matches itself (e.g. a matches a, b matches b, etc.)
 - a
 - 7
 - z
- May be concatenated
 - dog
- Dot . means match any character (not newline)
 - b.d matches bad, bid, bed, b3d, bcd, b(d, etc.

Regular Expressions

- `[]` means a character class

`b[aeiou]d` matches `bad, bed, bid, bod, bud`

Note: case sensitive

`[Bb][AaEeIiOoUu][Dd]`

`[a-z]` same as `[abcdefghijklmnopqrstuvwxyz]`

`[a-zA-Z]`

`[_a-zA-Z]`

A period: `\.` or `[.]`

How do you describe bracket itself?

Match left hand bracket `[[]`

Match right land bracket `[]]`

Regular Expressions

- Metasymbols
 - Some same as used in BNF Grammar
 - Some have different meaning
 - * [] ^ \$ + /
 - (Must escape \ if using as match character)

Regular Expressions

Some important operators

- +** means one or more
- *** means zero or more (Kleene star)
- ?** means zero or one
- |** either or e.g. $a | b$ means either a or b

Regular Expressions

Some important UNIX style operators (used by grep, awk, vi, perl etc.)

- ^ means match at beginning
- \$ means match at end

Regular Expressions

- `^` inside of a character class (`[]`) means complement of that character class
 - `^a` : Match (word | line) starting with "a"
 - `[^a]` : Match any character except a small a
- `$` means match line ending with whatever precedes `$` sign
 - `grep the myfile` will find all occurrences of "the" in myfile
 - `grep -w the myfile` means match "the" as a word
 - `grep -w ^the myfile` means match "the" as word only at beginning of line
 - `grep -w the$ myfile` means match the "the" as word only at end of line
- `.*` Matches everything (except newline `\n`)!

Regular Expressions

- Write a regular expression to match an identifier
`[a-zA-Z_][a-zA-Z_0-9]*`
- Write a regular expression for an unsigned integer
`[0-9][0-9]*`
- Unsigned floating point
`[0-9]+[.][0-9]+`
- What about
`[0-9]*[.][0-9]*`
- Okay?
 - 1-Yes
 - 2-No

Regular Expressions

- Write a regular expression to match an identifier
`[a-zA-Z_][a-zA-Z_0-9]*`
- Write a regular expression for an unsigned integer
`[0-9][0-9]*`
- Unsigned floating point
`[0-9]+[.][0-9]+`
- What about
`[0-9]*[.][0-9]*`
- Allows `.`
`([0-9]+[.][0-9]* | [0-9]*[.][0-9]+)`

Regular Expressions

- Match any character not a vowel
`[^aeiouAEIOU]`
- Match any uc or lc consonant
`[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z]`
- To include ^ put it somewhere other than the beginning
- Regular Expressions are used in grep, sed, awk, perl, vi, shells, lex, yacc
 - Note: Each of these may have its own slightly different rules

More info on RegEx

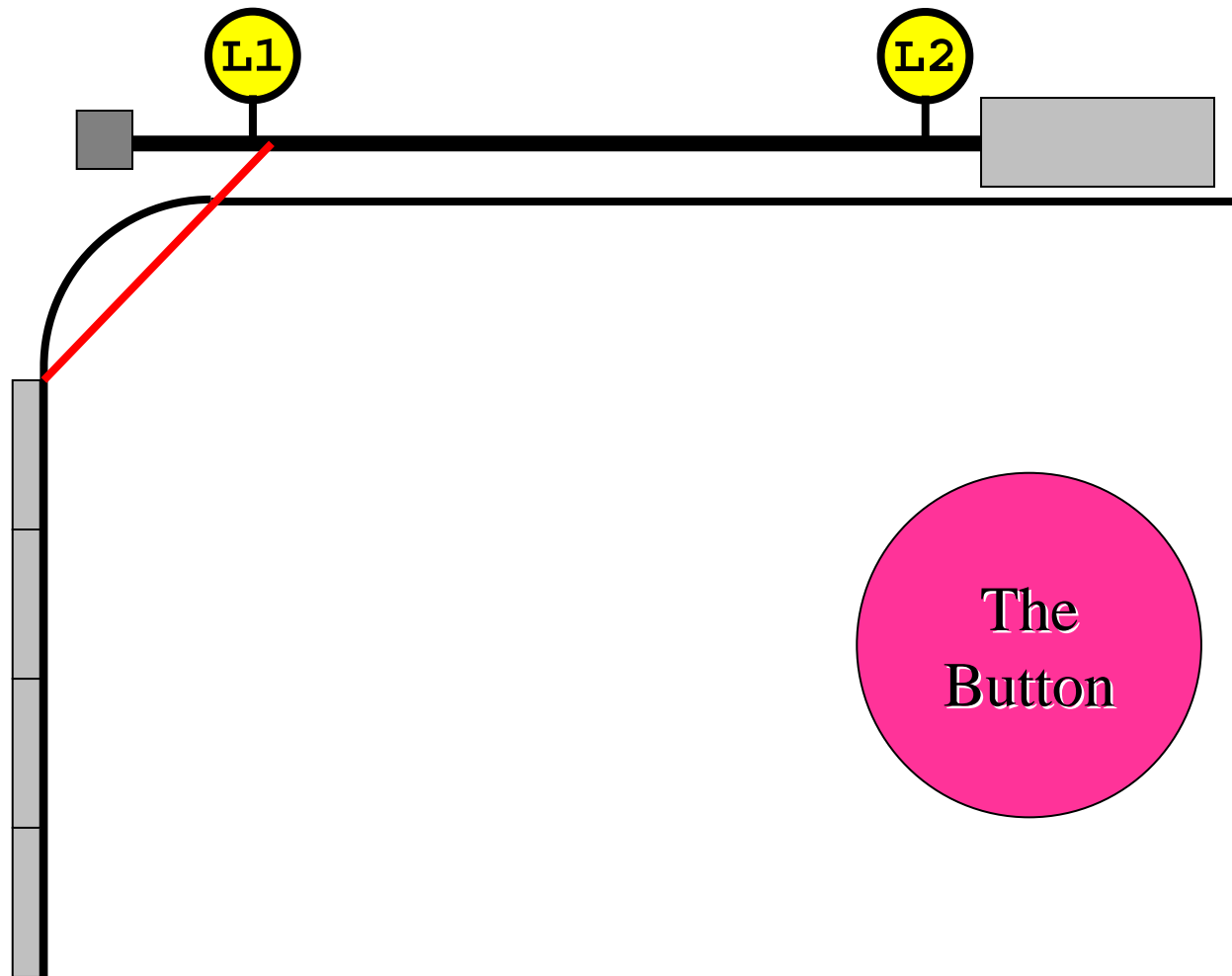
- Loudon Book
- man regex
- Other sources...
 - Various O'Reilly books...
- grep uses standard regular expressions
- egrep uses extended regular expressions
- Nice practice tool:
`http://www.weitz.de/regex-coach/`

Practice Regular Expressions

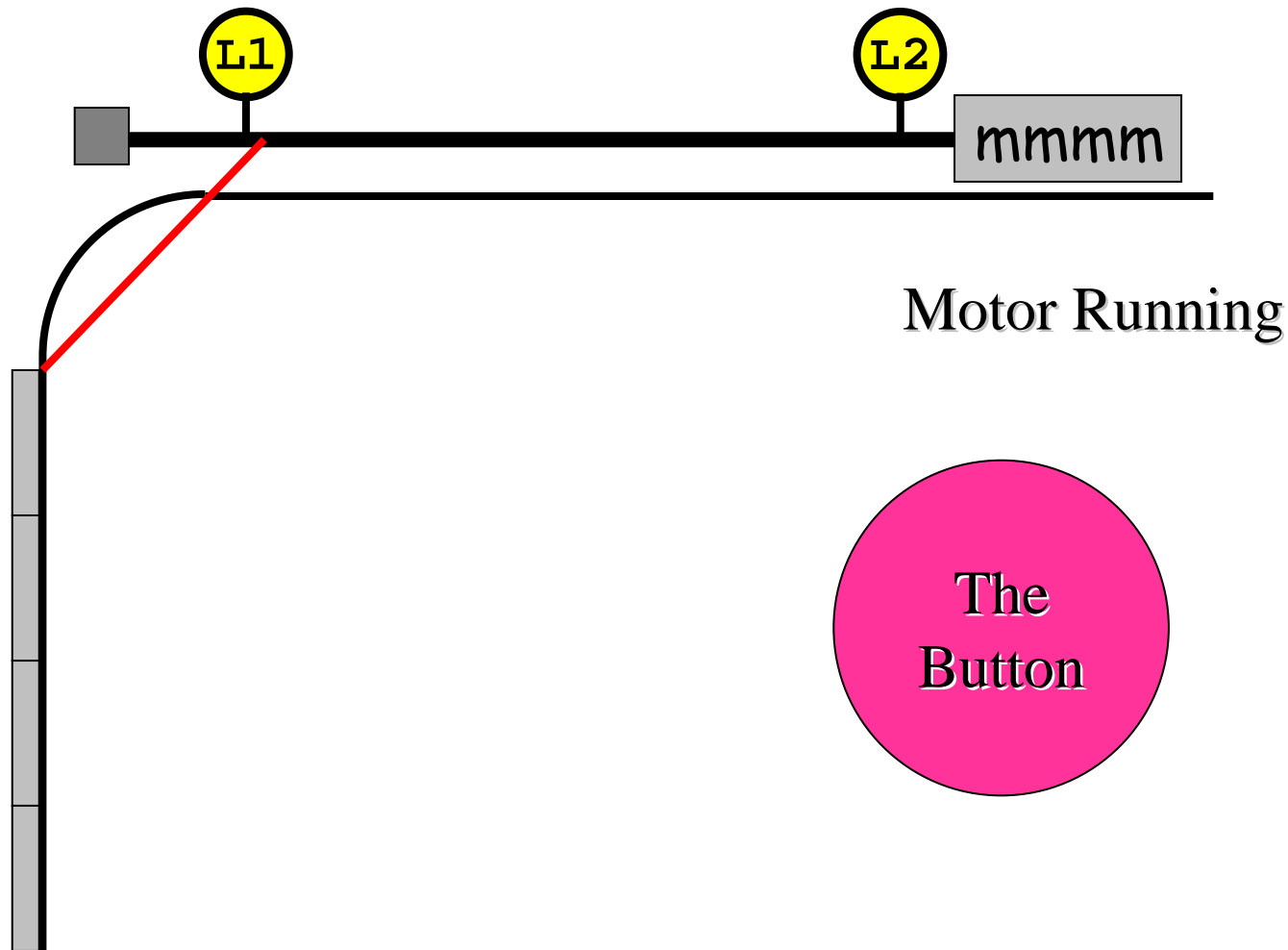
- Write a regular expression consisting of 0's and 1's which may have a 0 but whenever it occurs it must be followed by a 1
- An identifier that starts with a lower case letter or underscore and followed by one or more of lower case letters and digits and underscores. A letter or a digit must follow an underscore.
- A string that consists of at least 3 consecutive 0's

Questions?

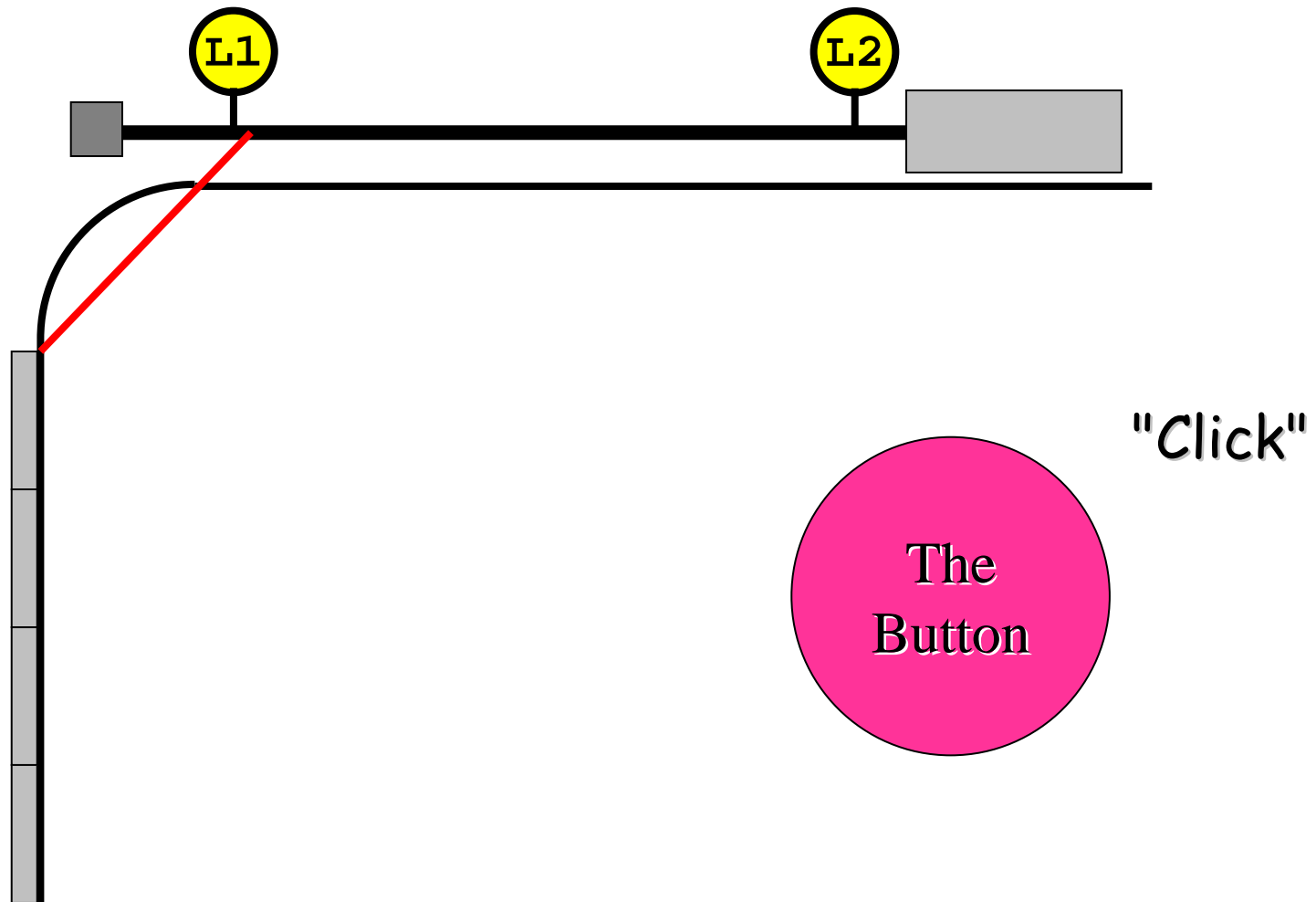
My Garage Door



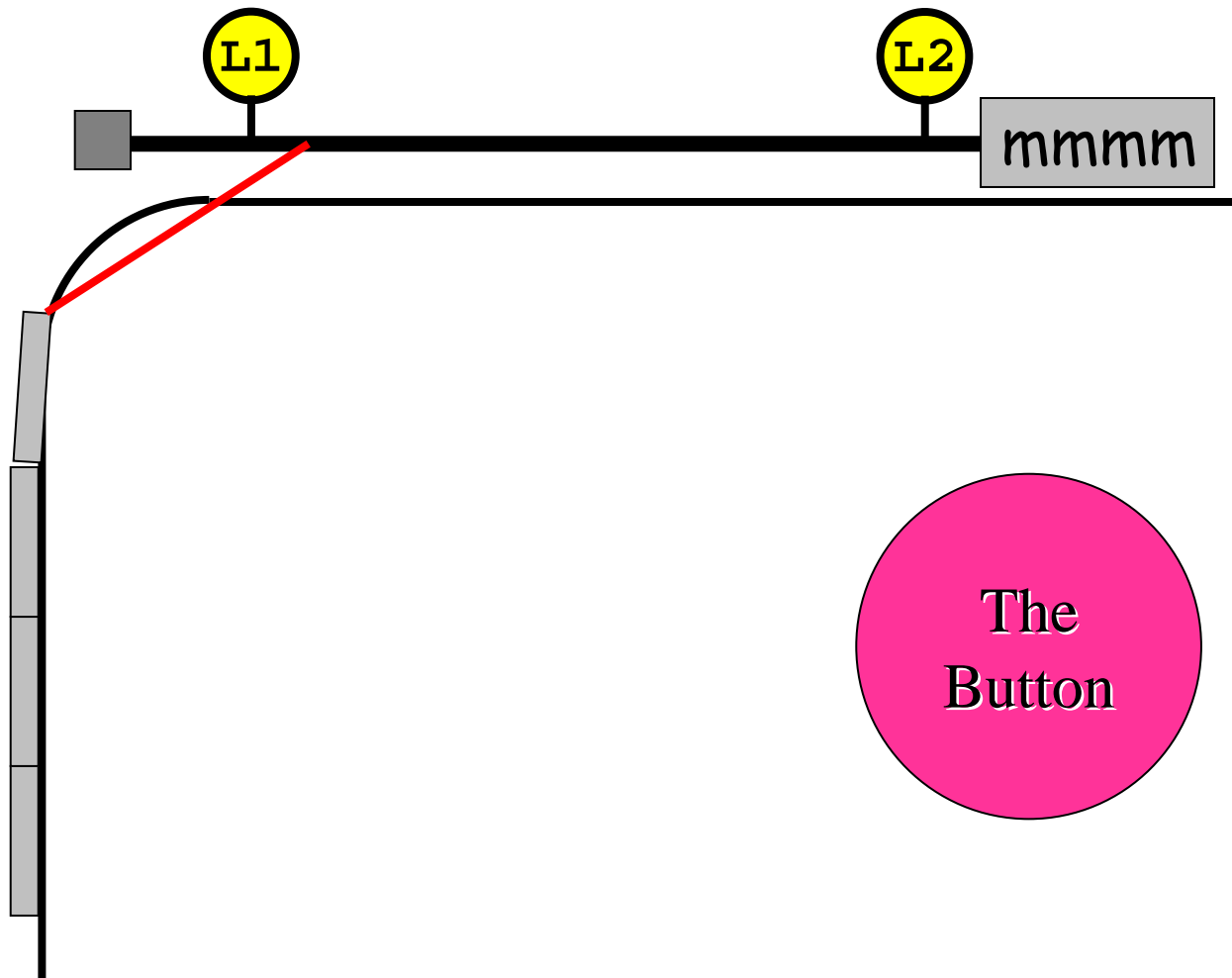
My Garage Door



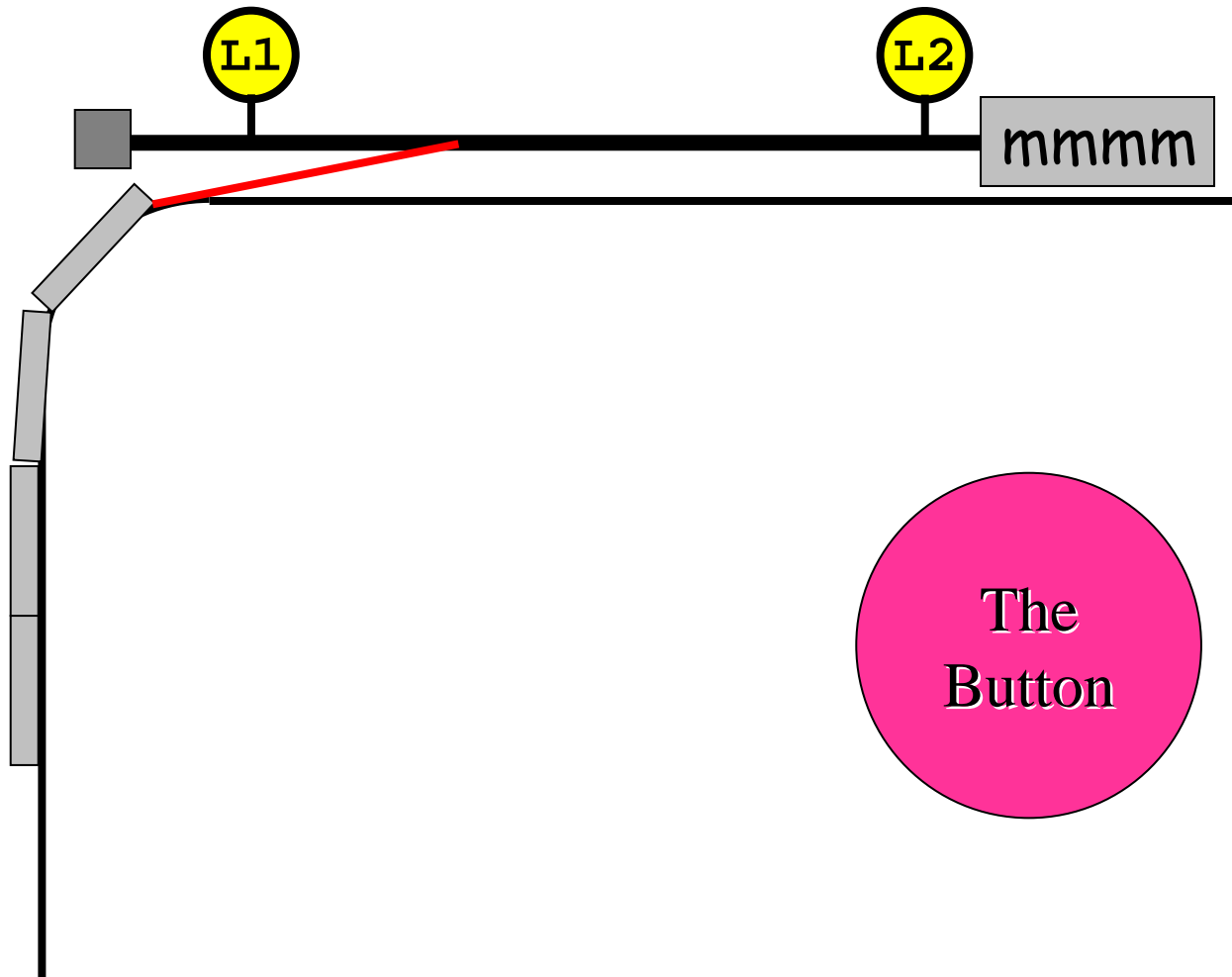
My Garage Door



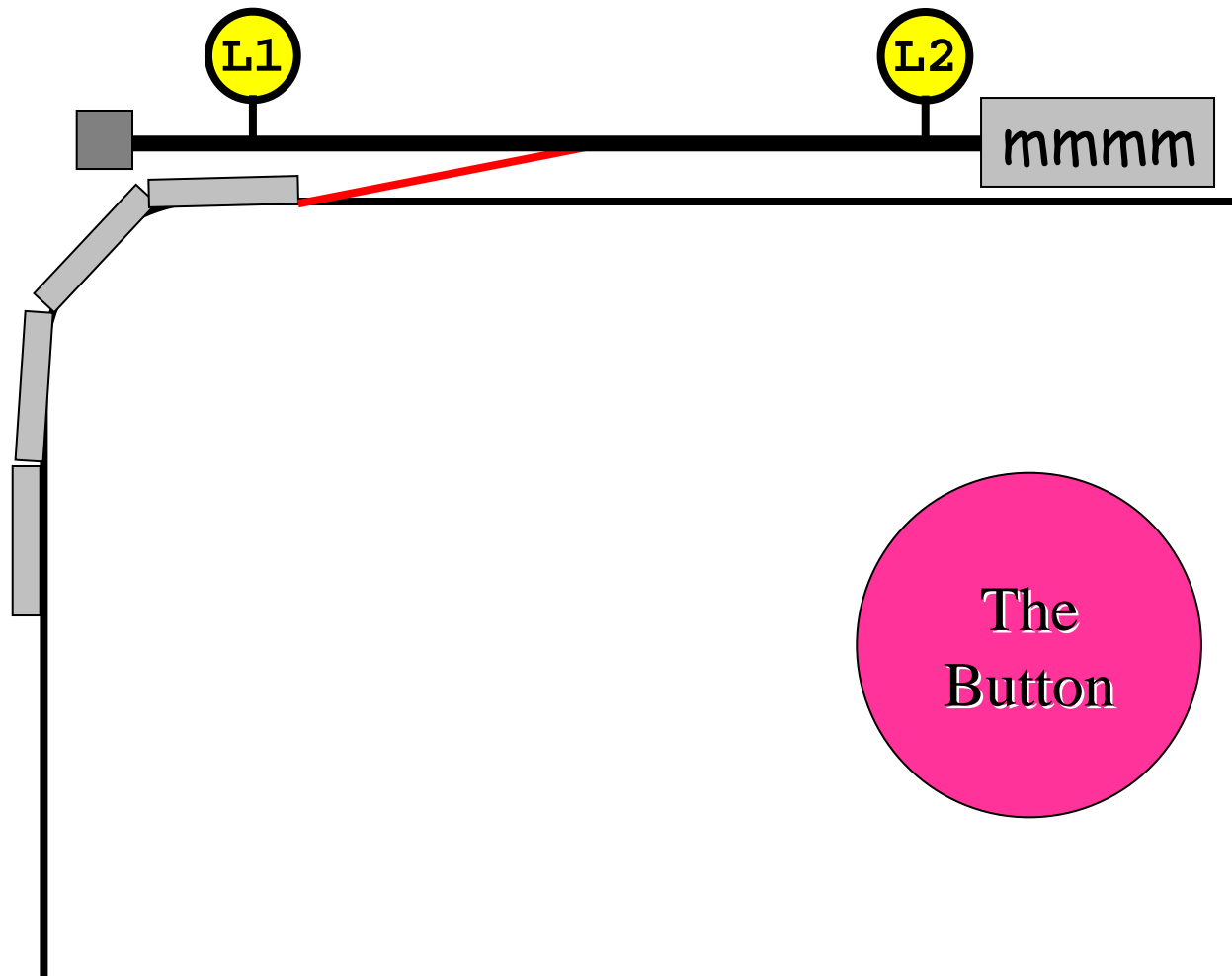
My Garage Door



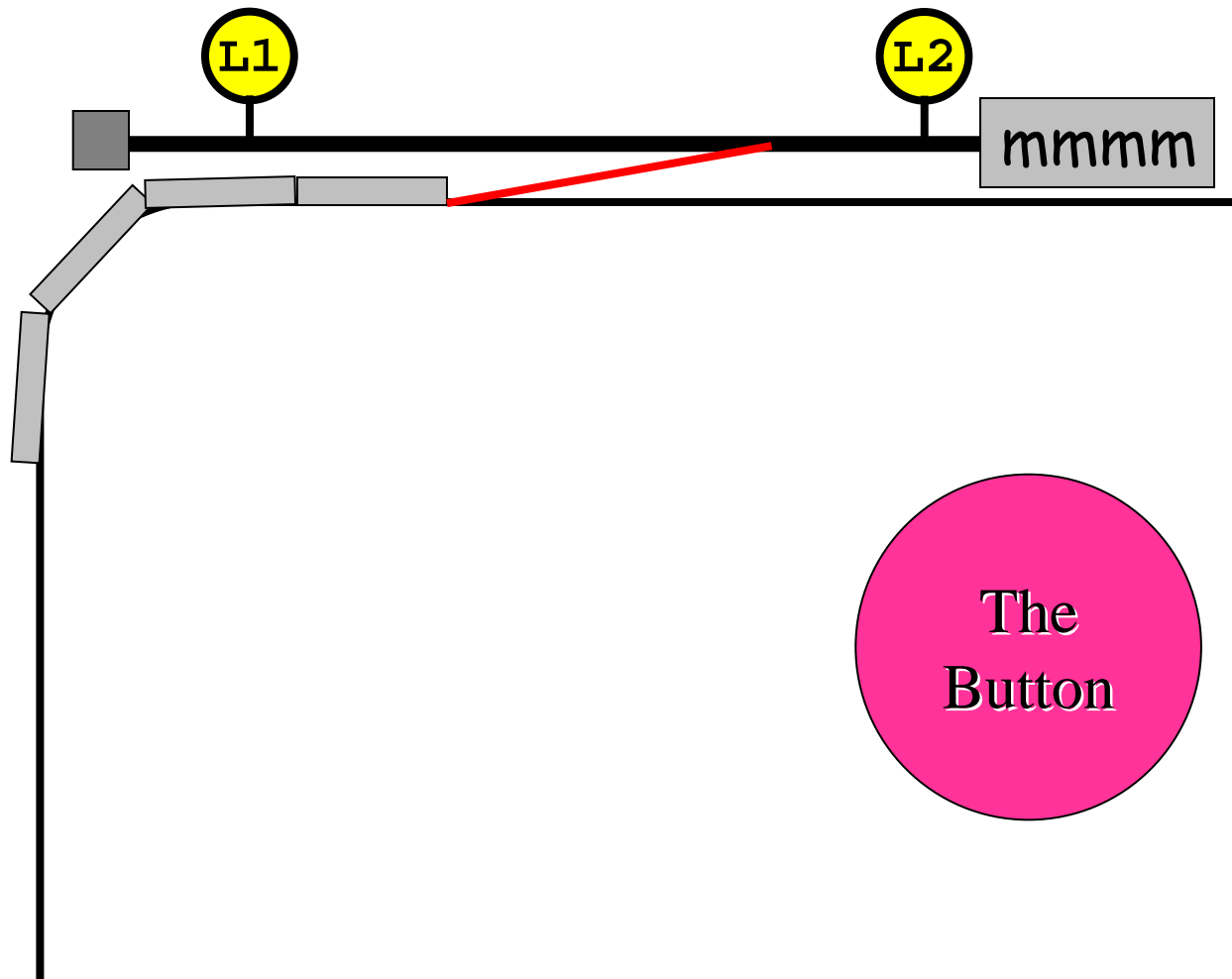
My Garage Door



My Garage Door

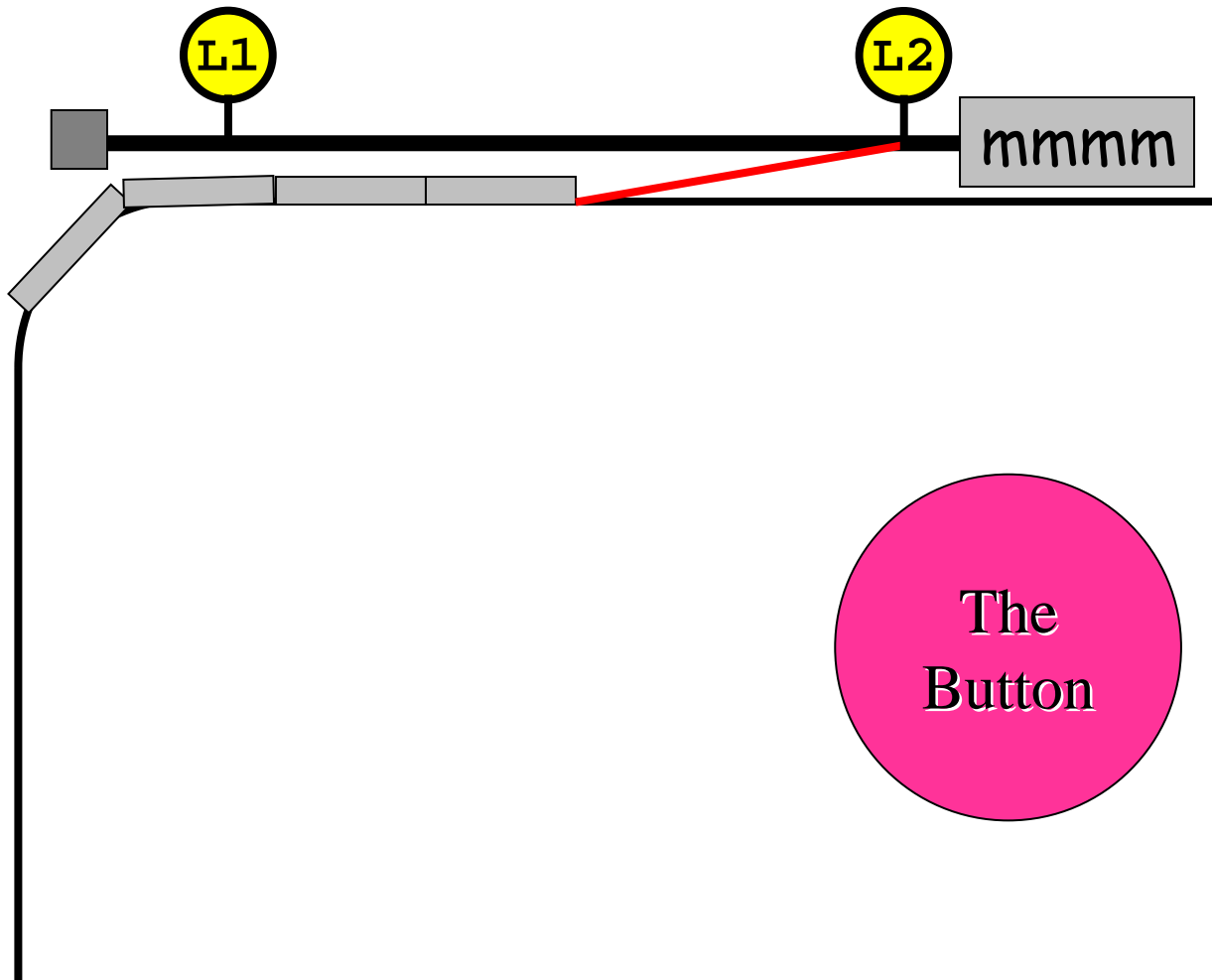


My Garage Door

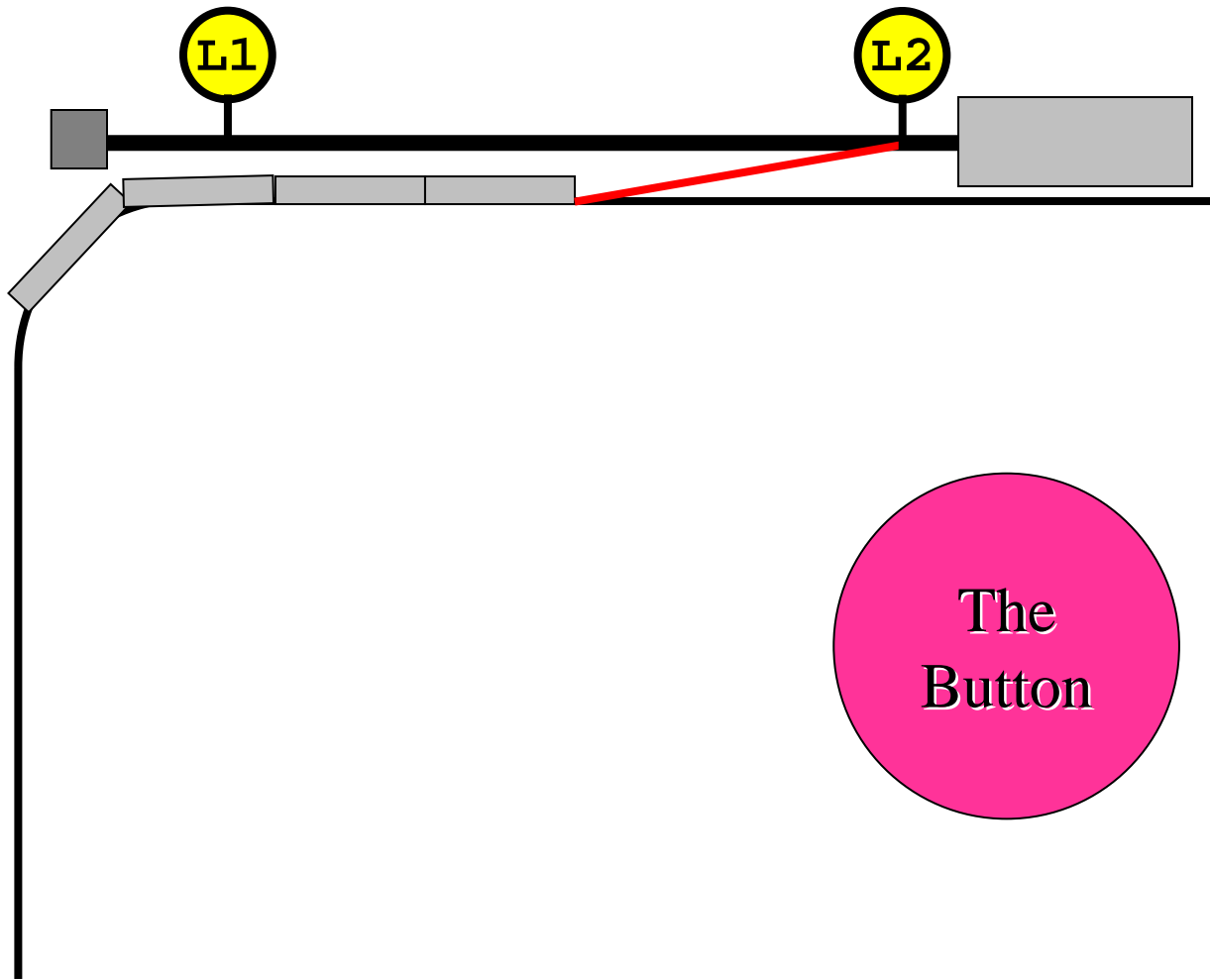


The
Button

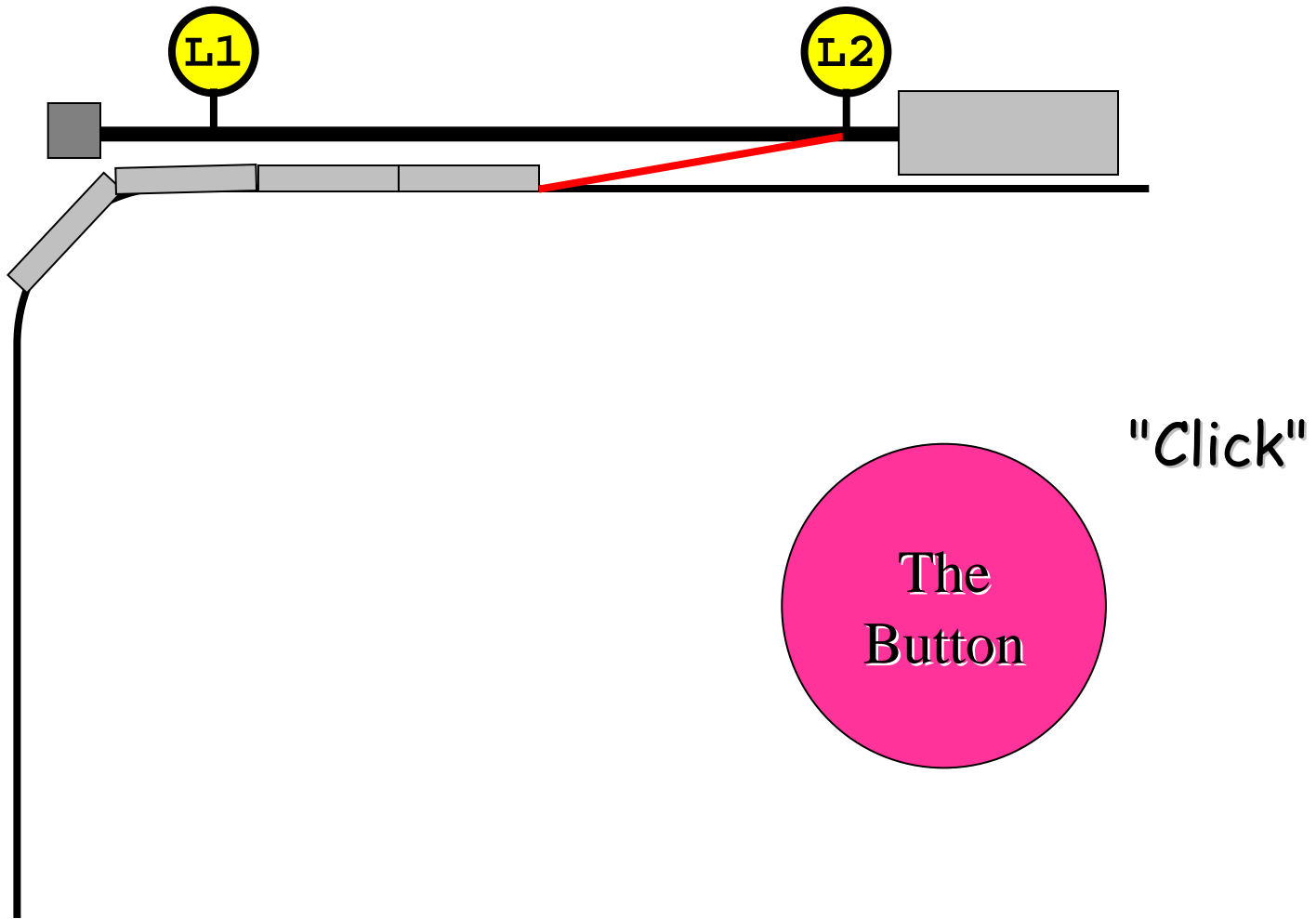
My Garage Door



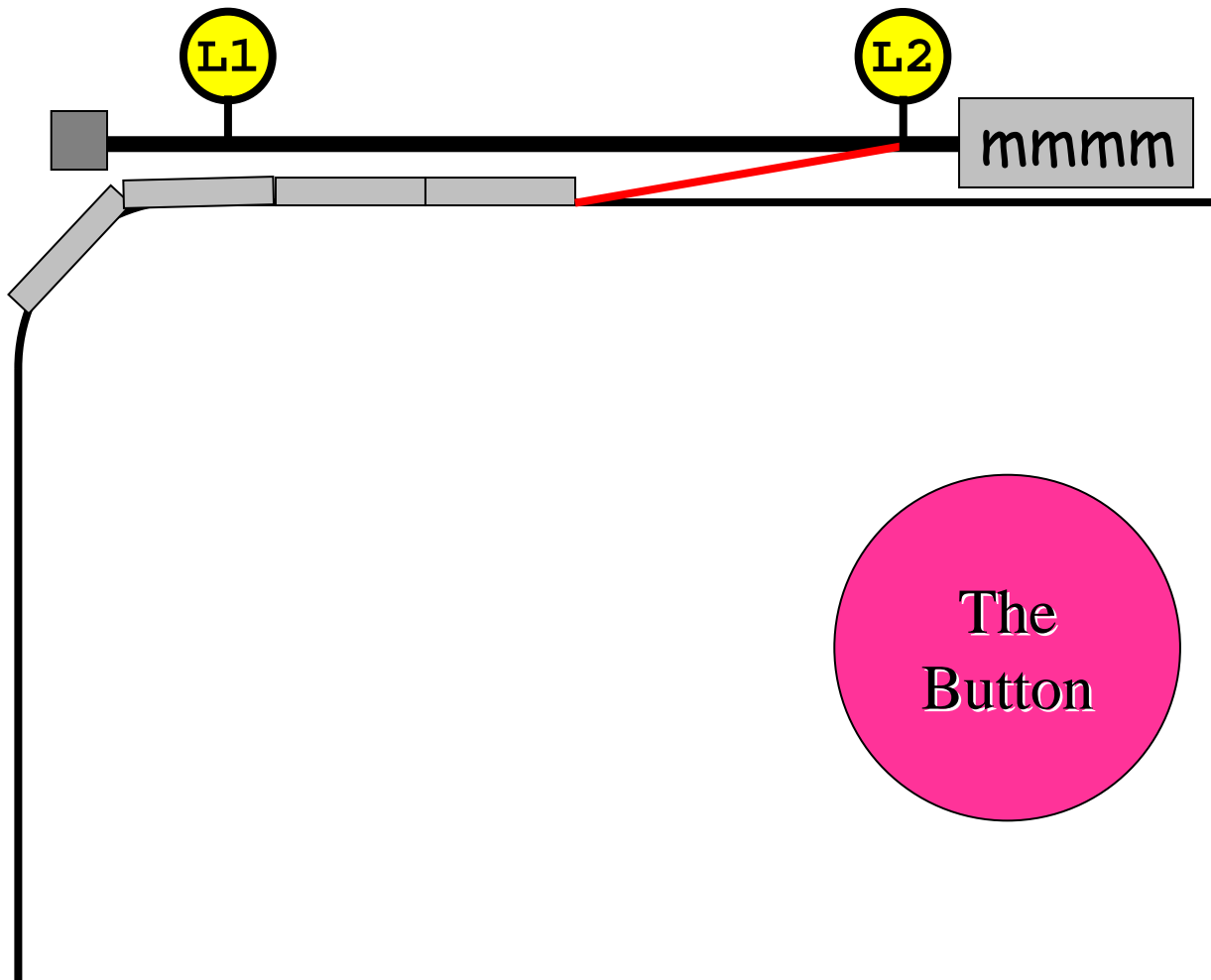
My Garage Door



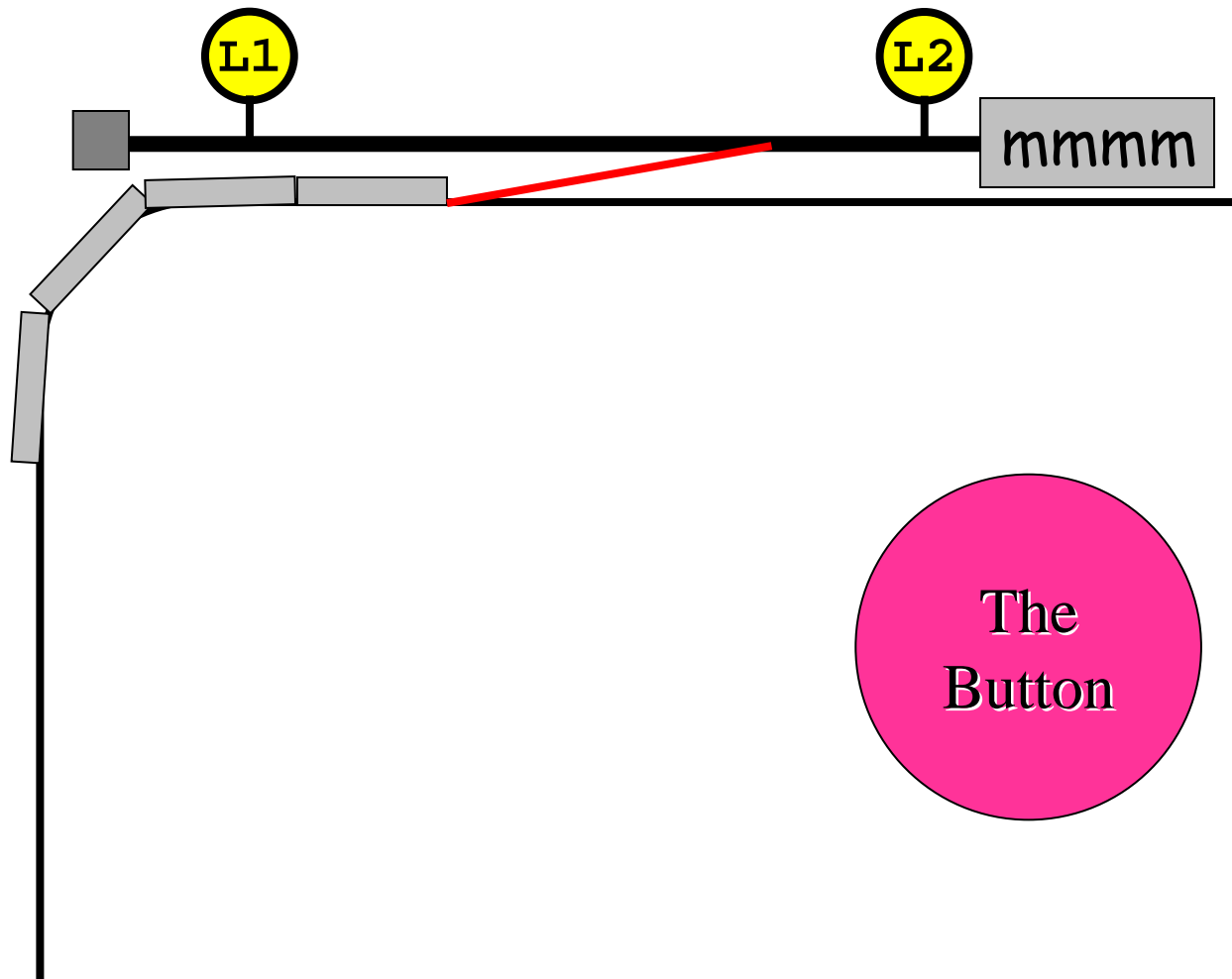
My Garage Door



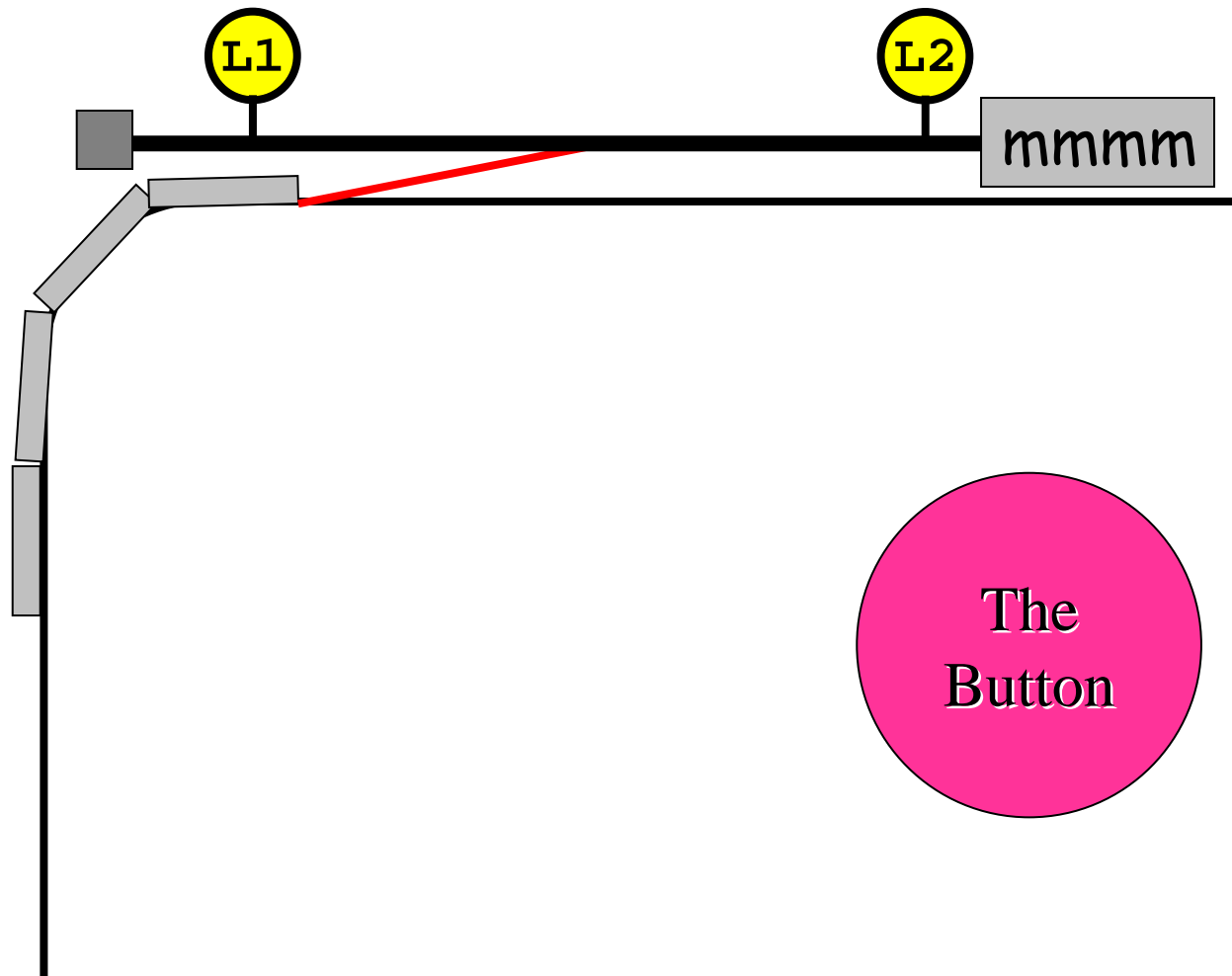
My Garage Door



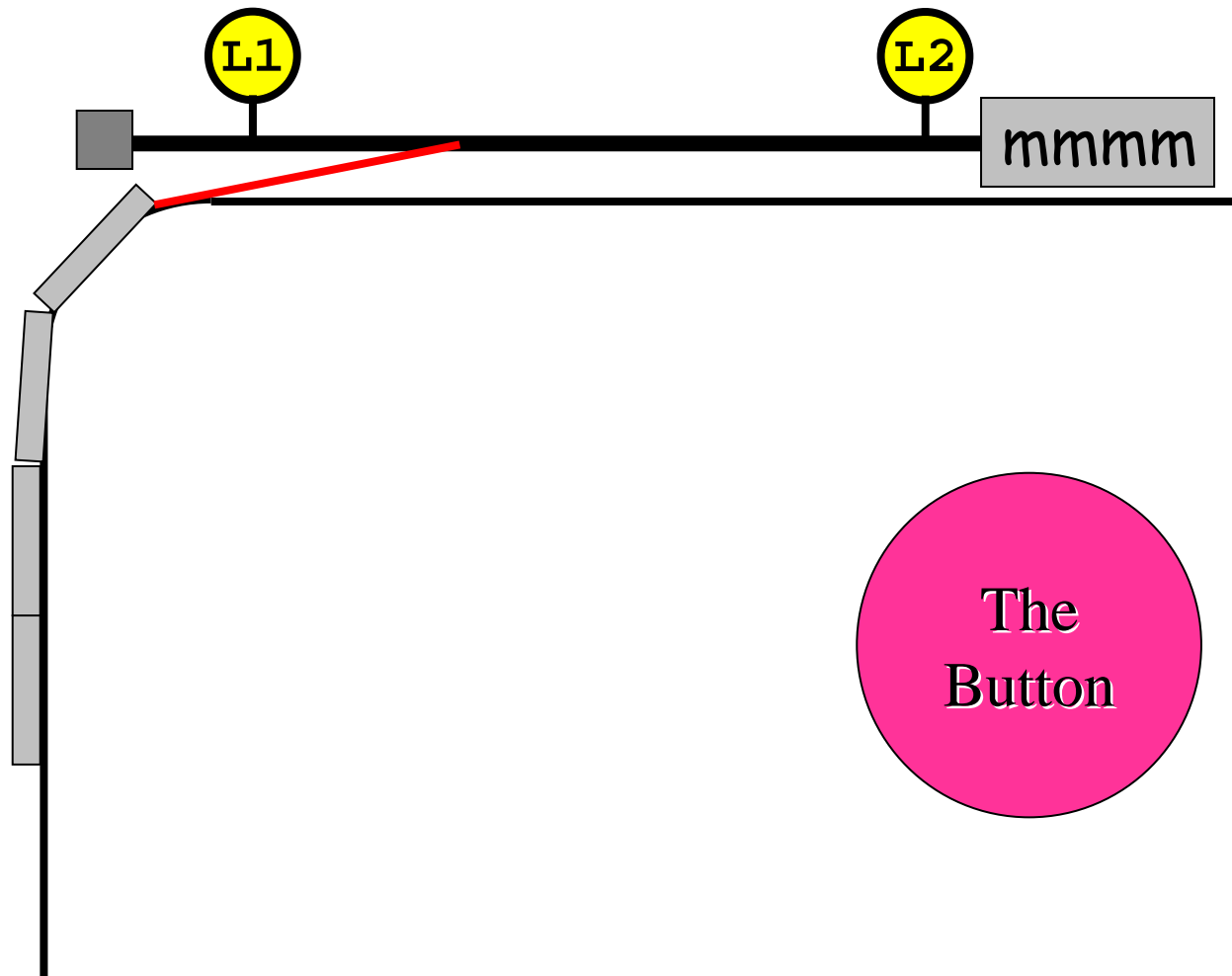
My Garage Door



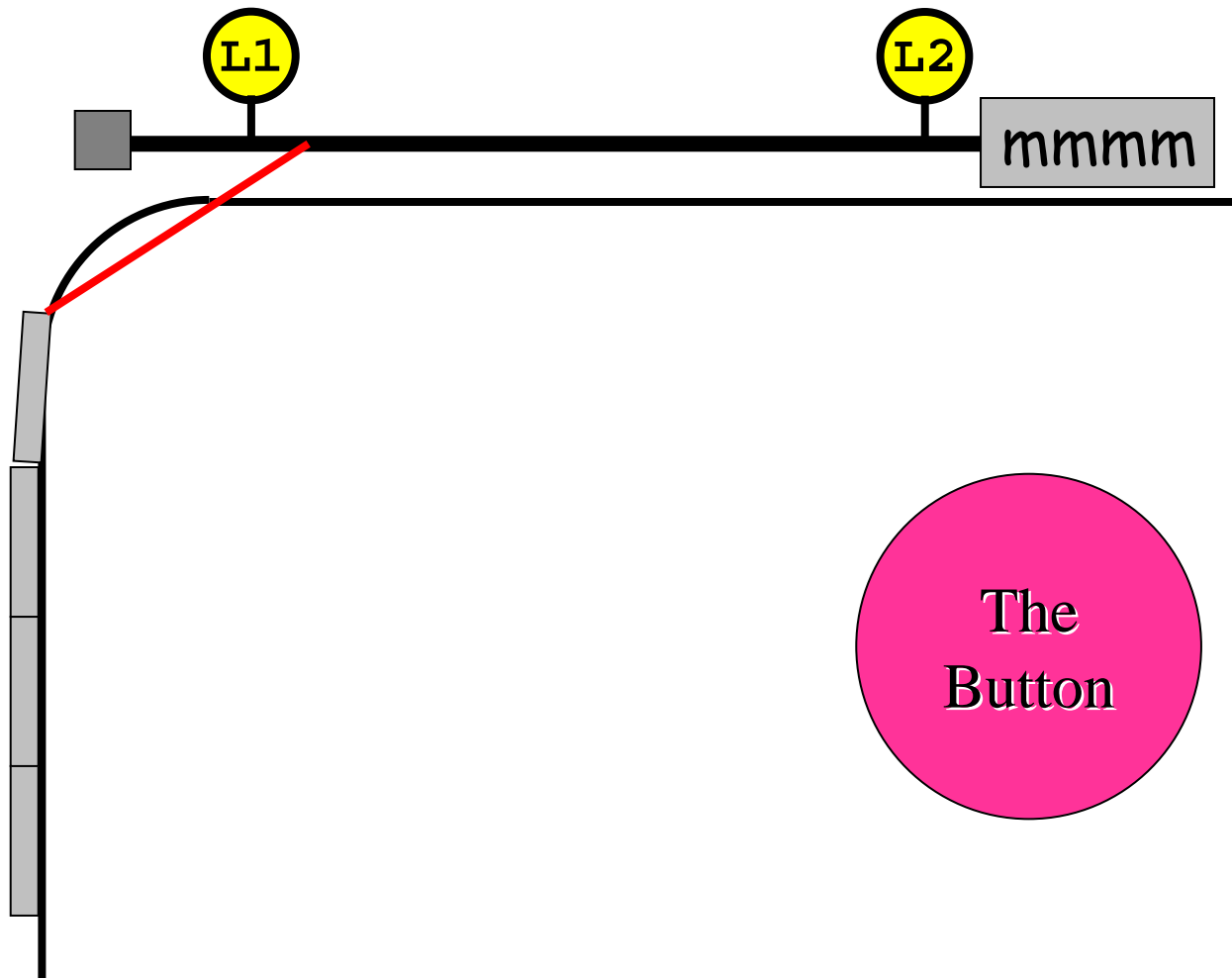
My Garage Door



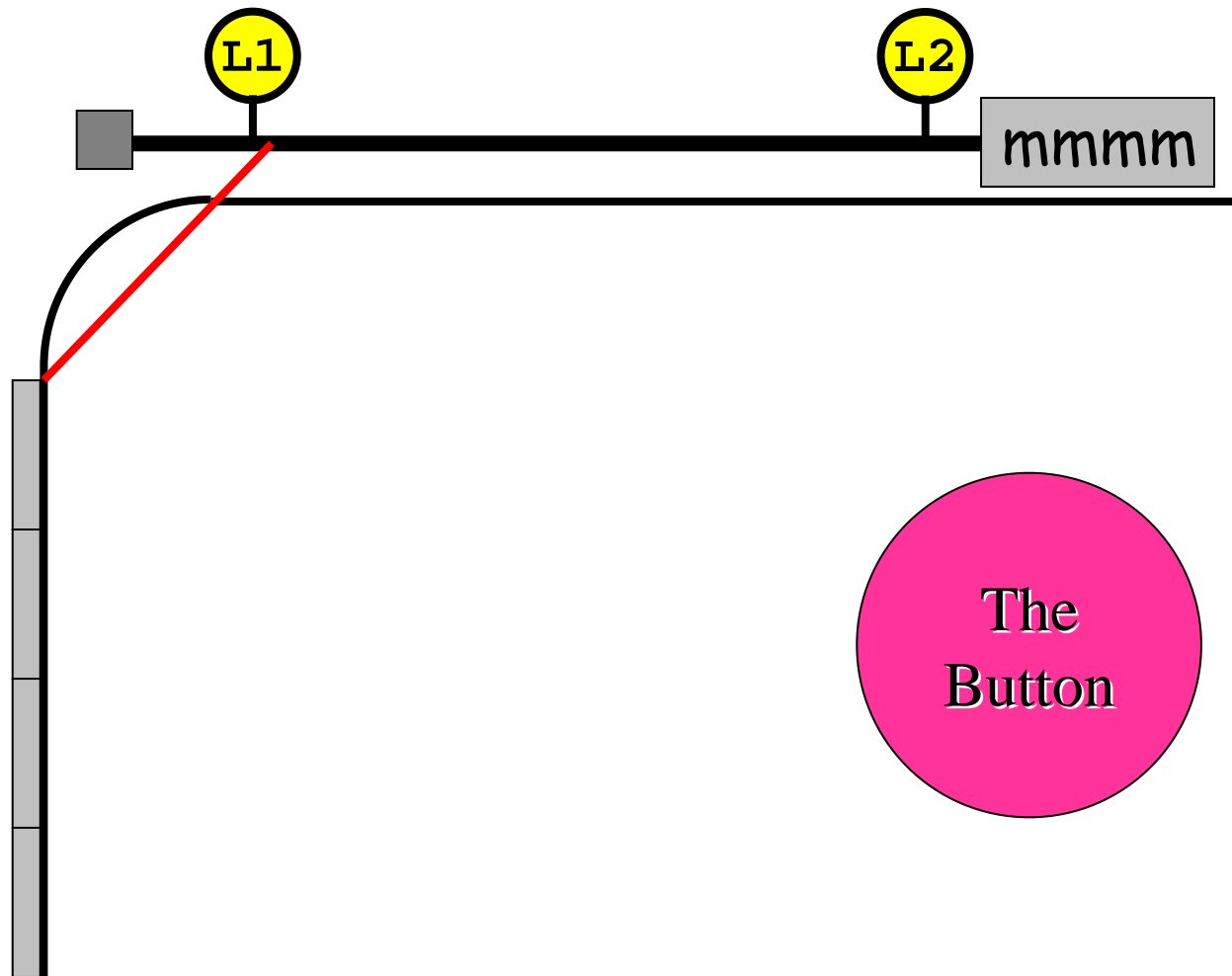
My Garage Door



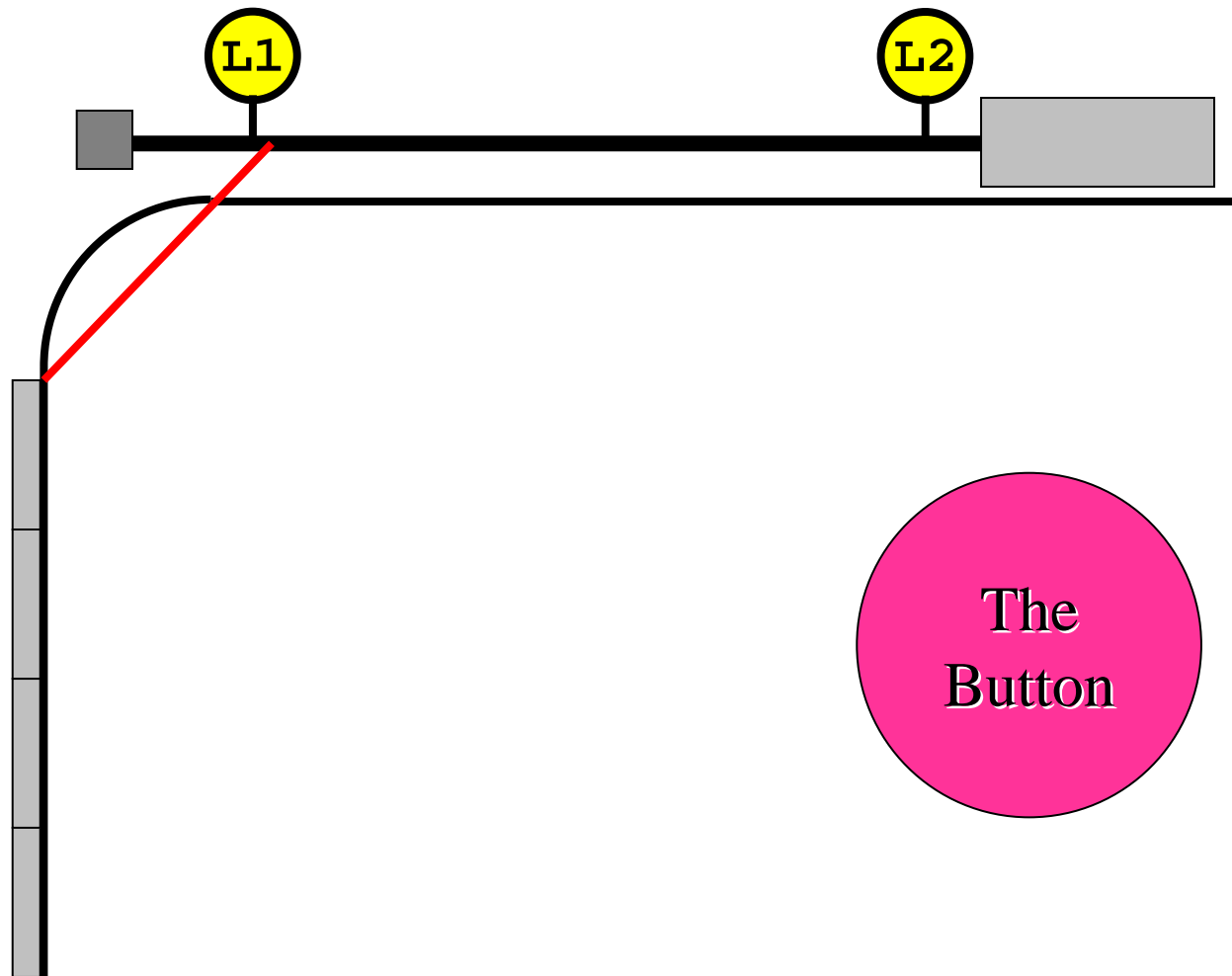
My Garage Door



My Garage Door



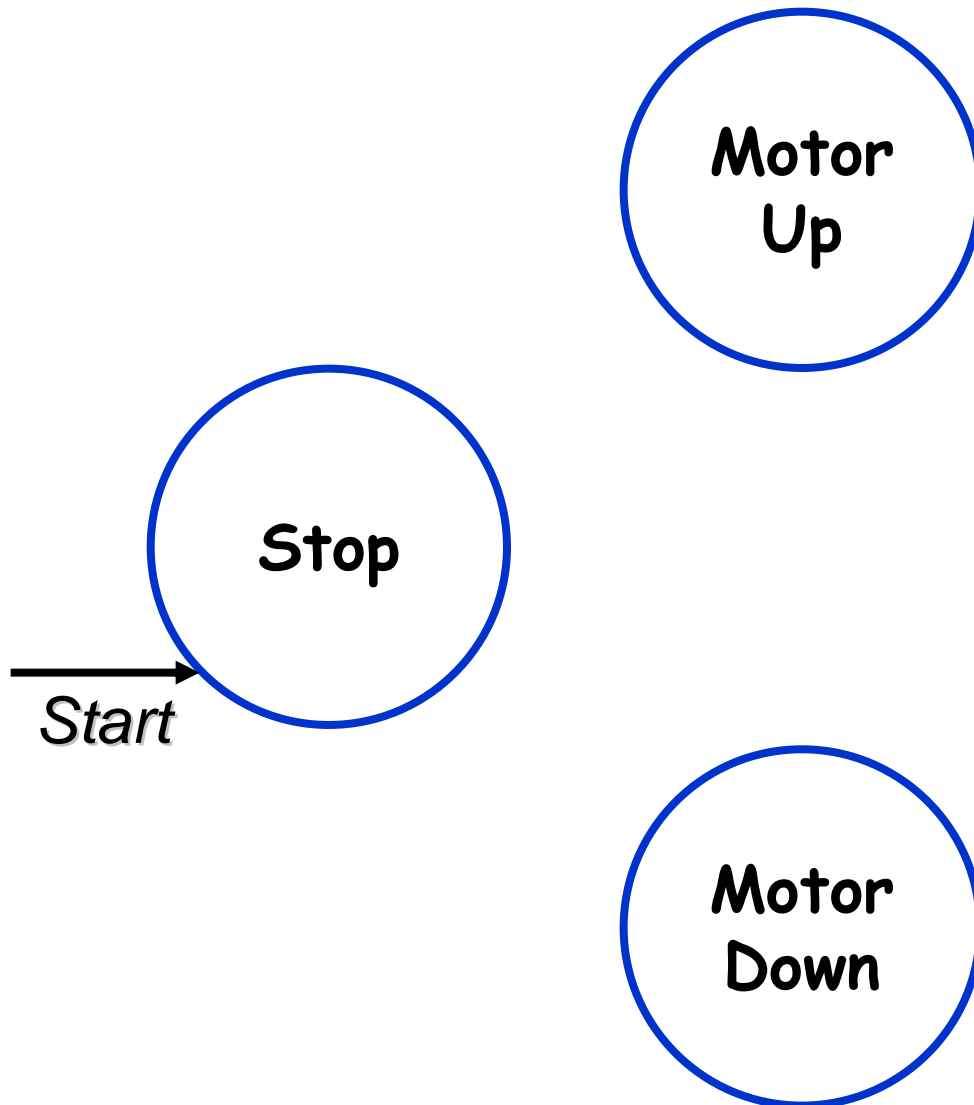
My Garage Door



How can we describe the control logic?

A State Machine!

State Machine

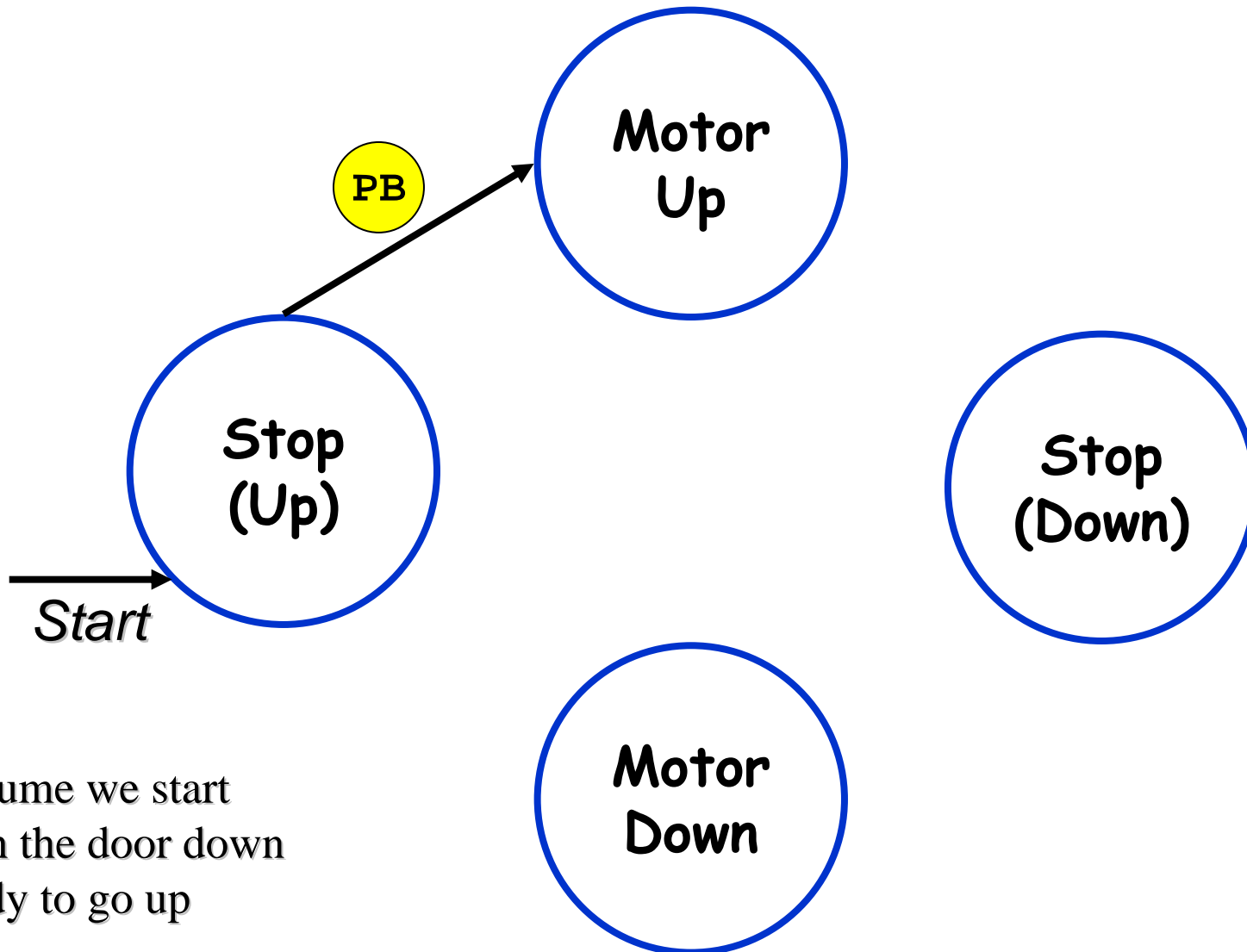


Enough
States?

1-Yes

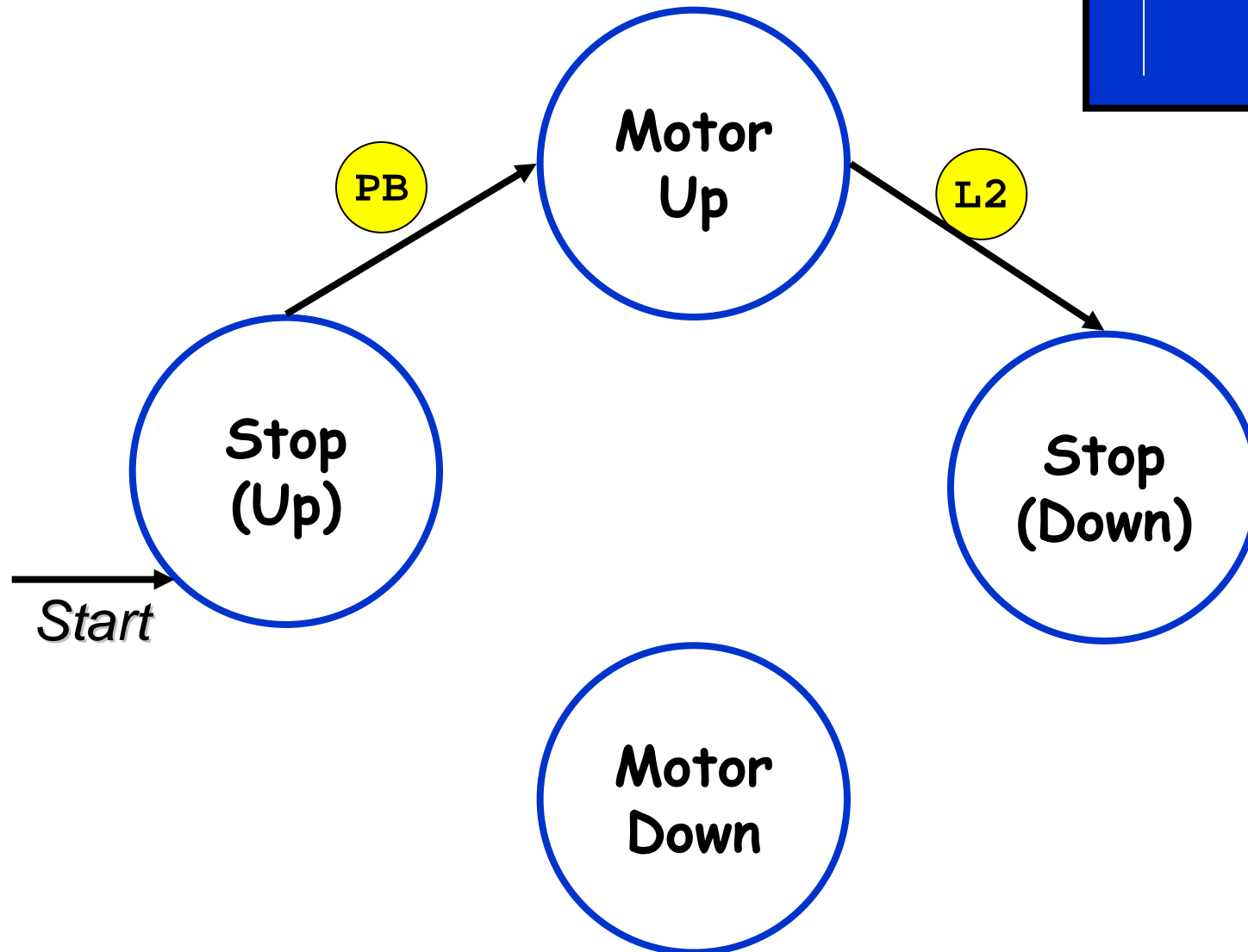
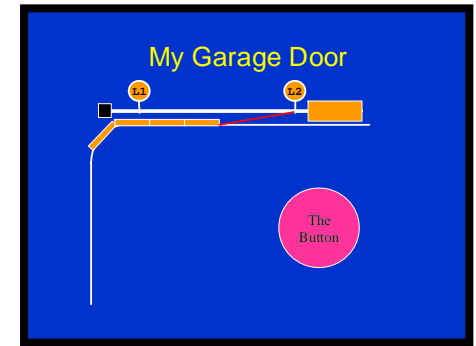
2-No

State Machine

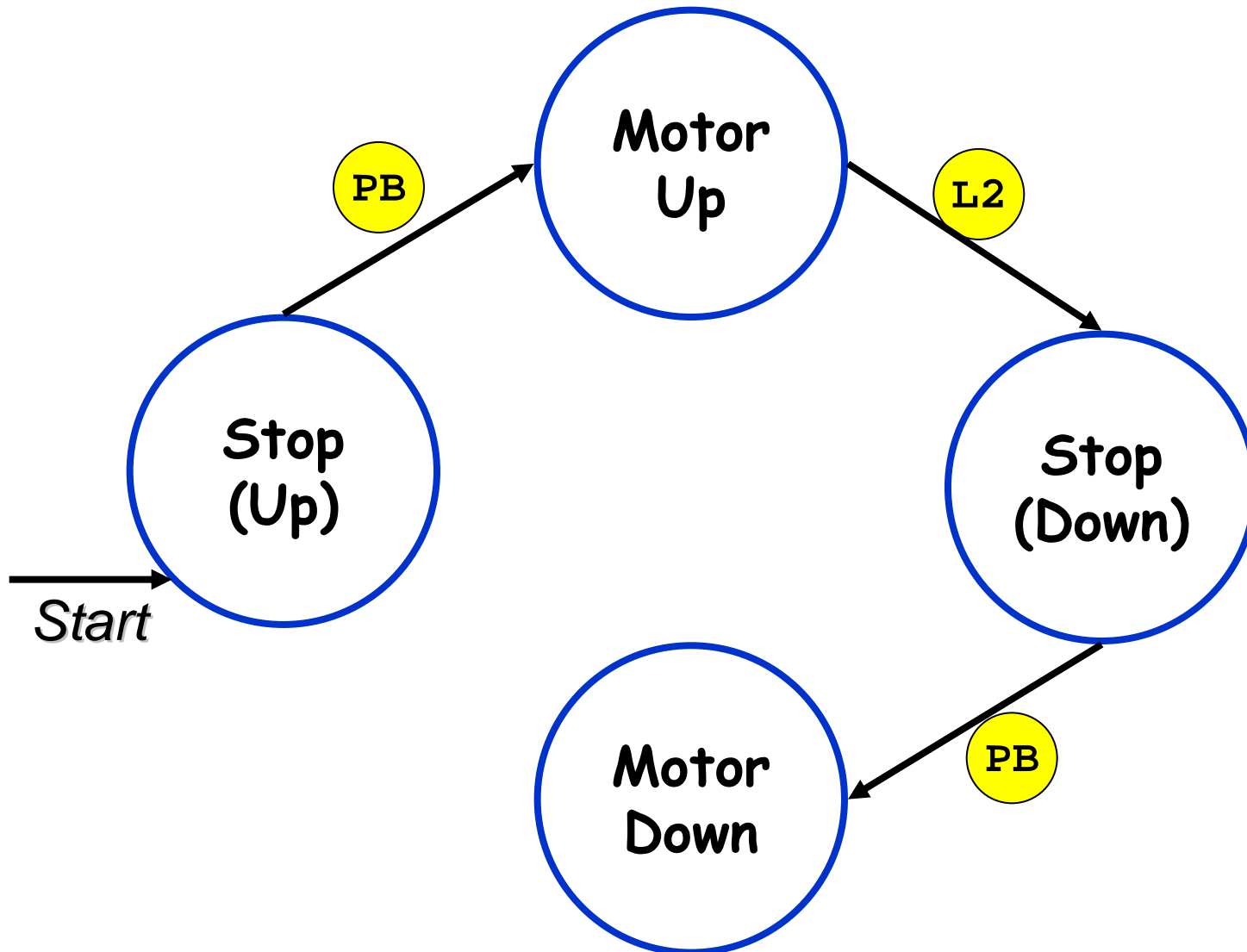


Assume we start
with the door down
ready to go up

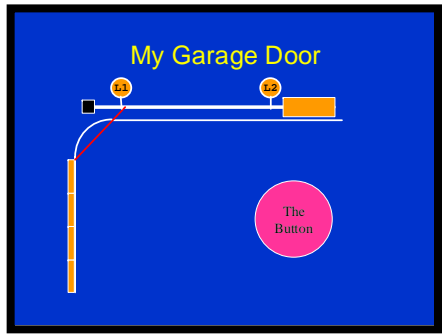
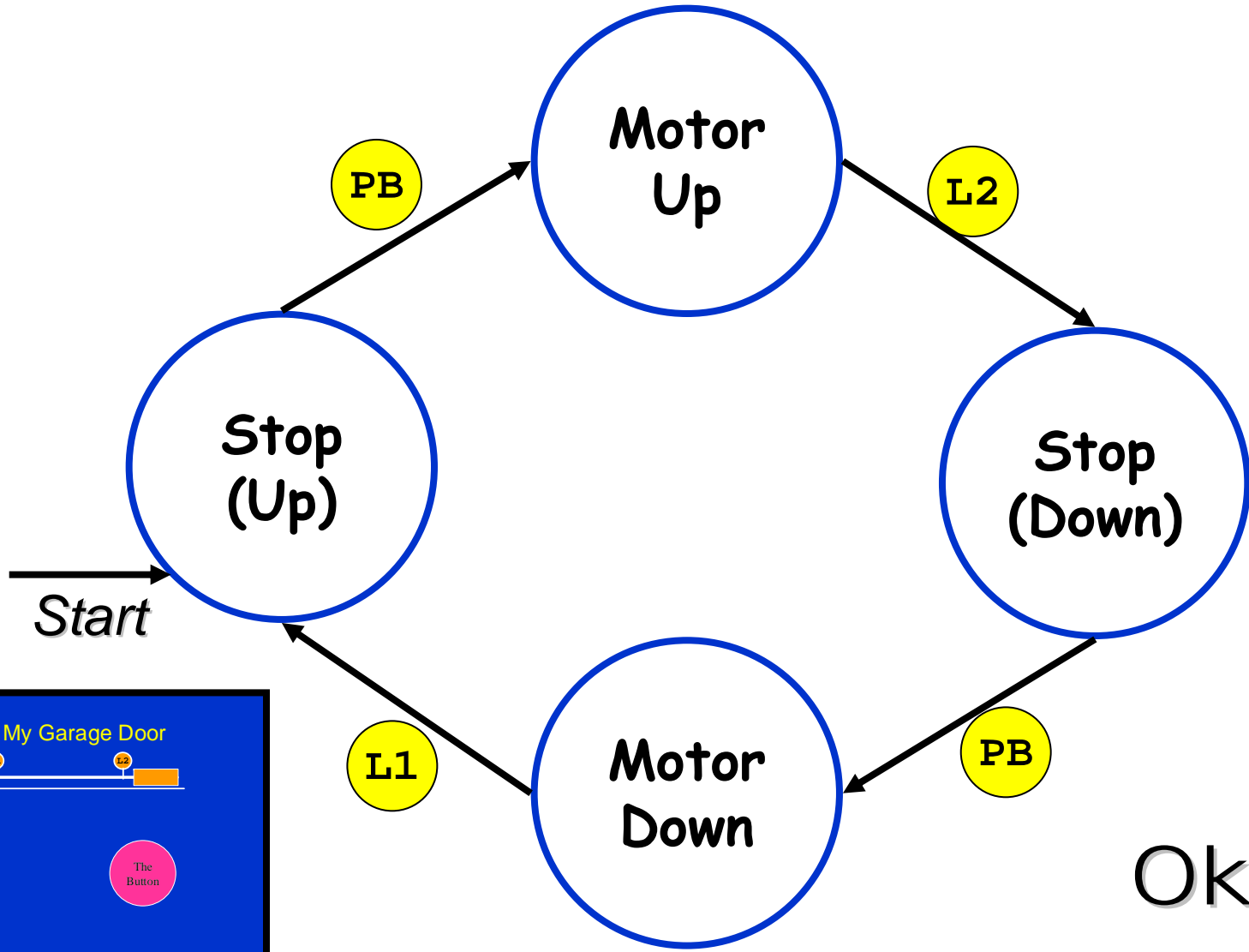
State Machine



State Machine

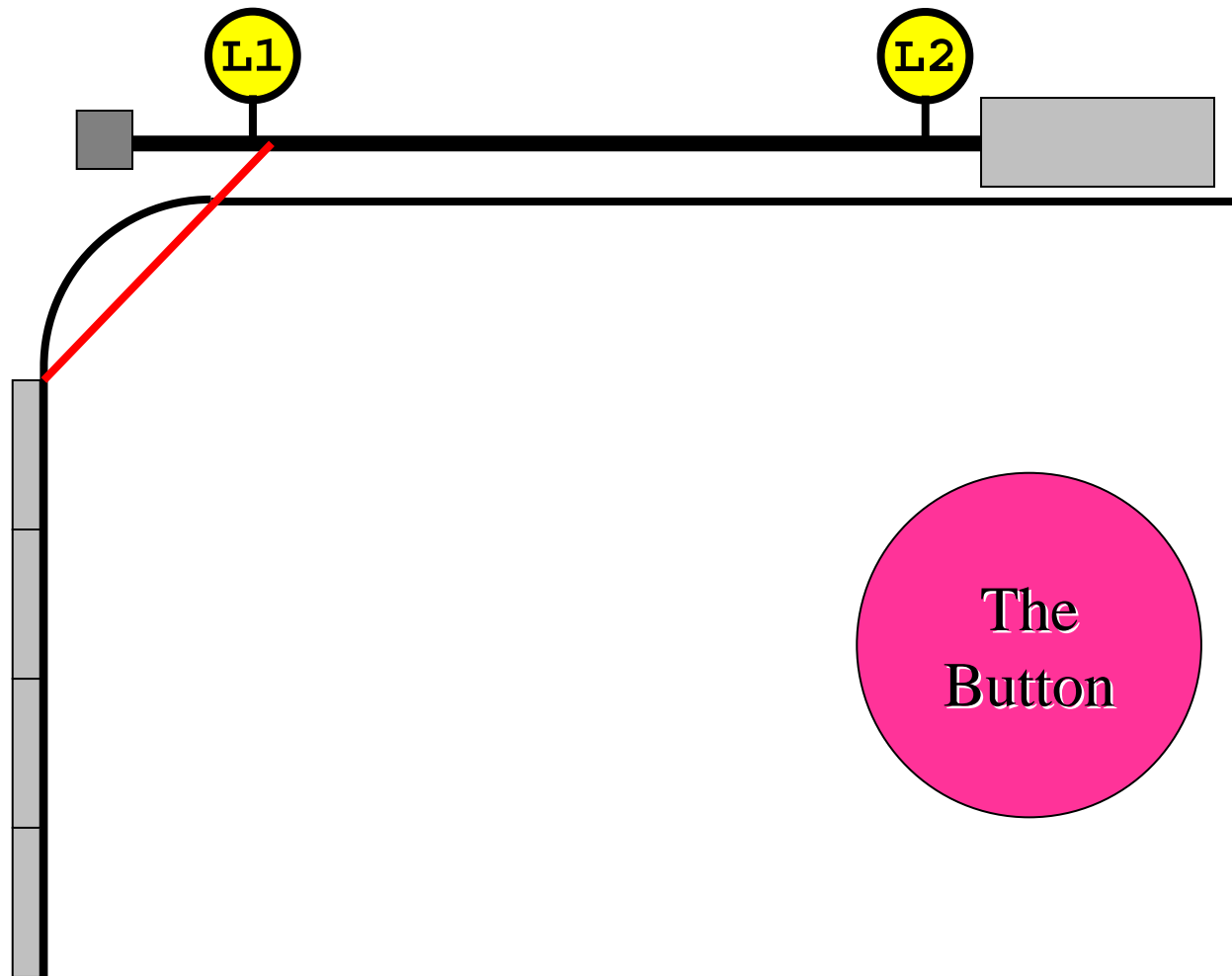


State Machine

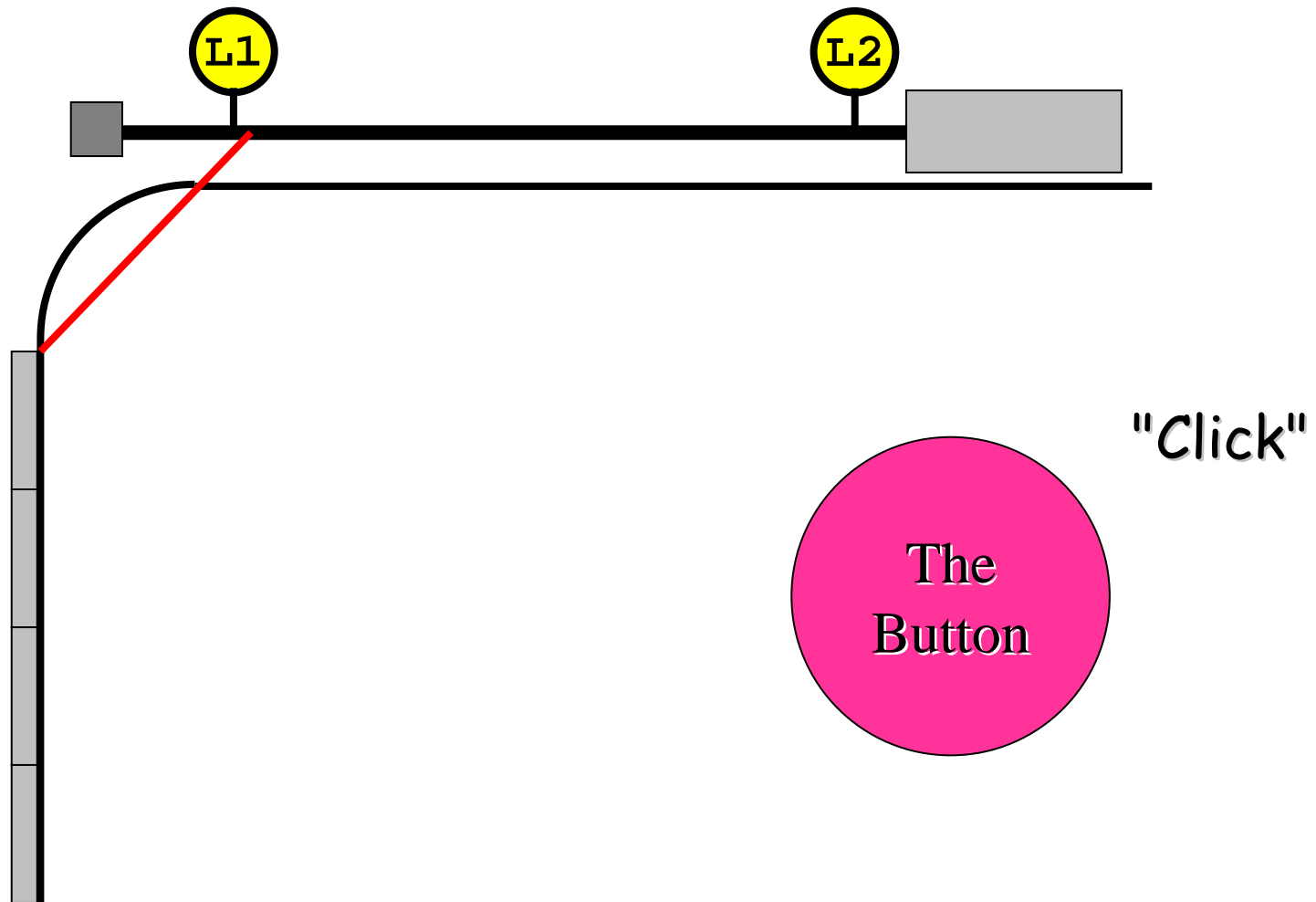


Okay?

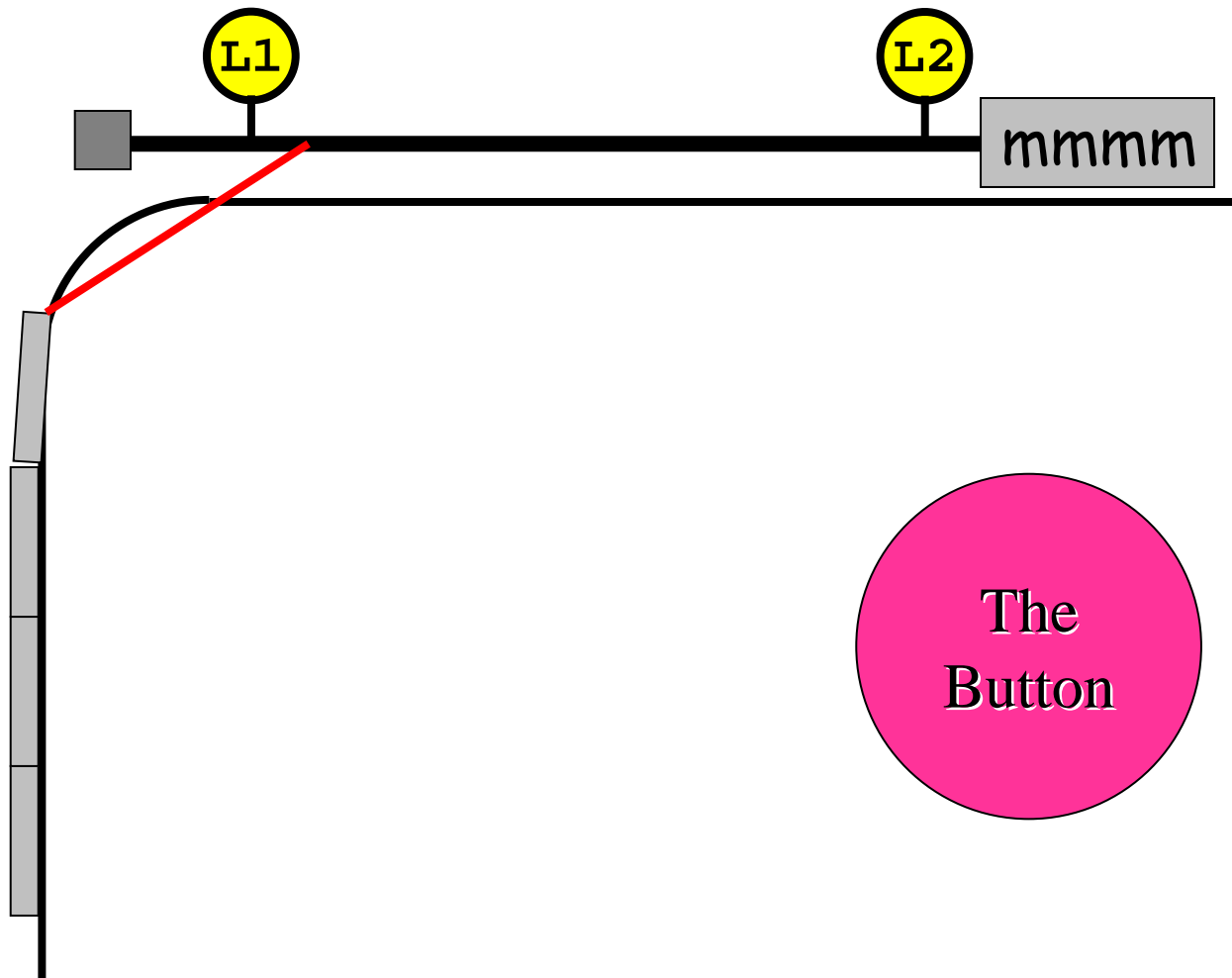
My Garage Door



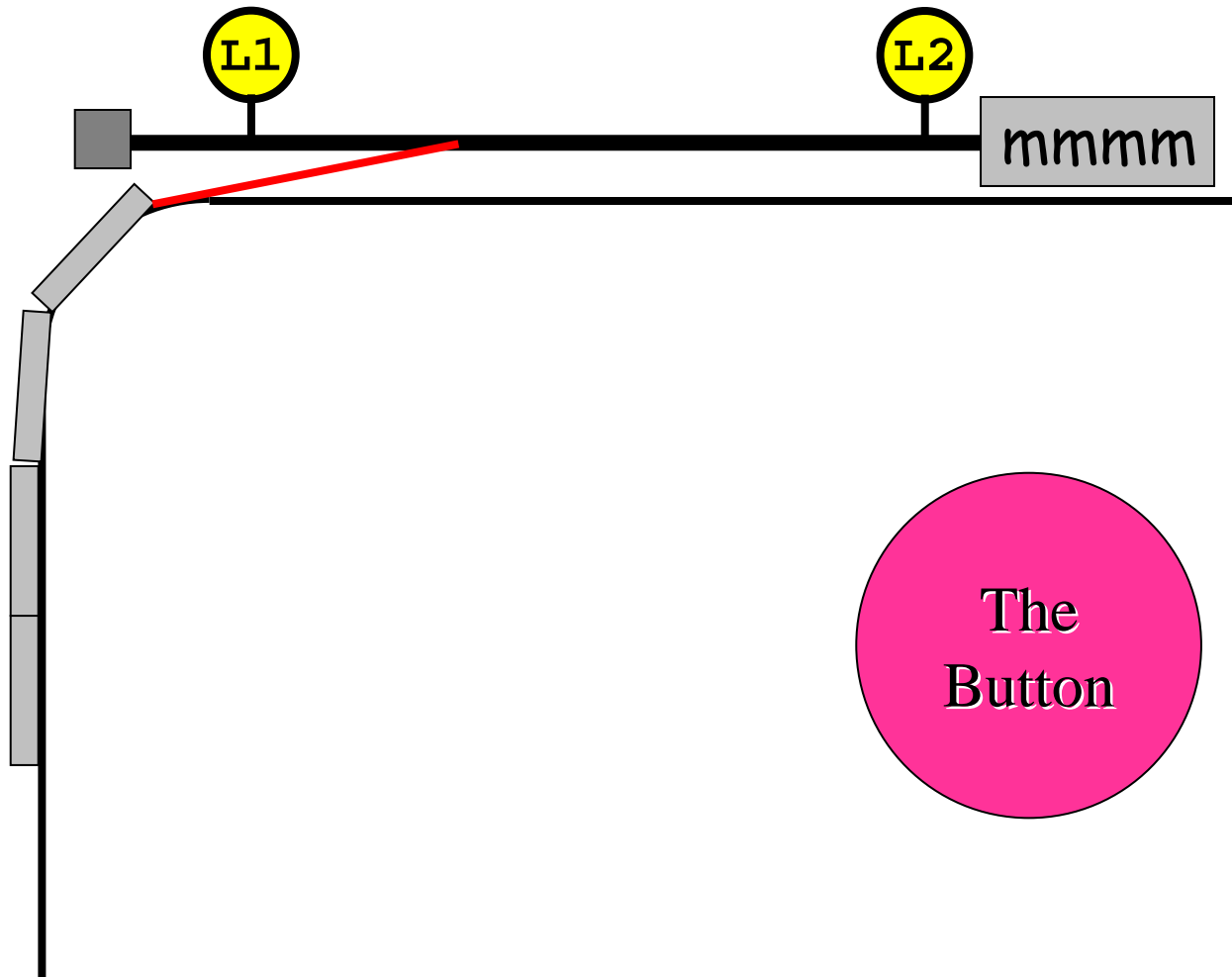
My Garage Door



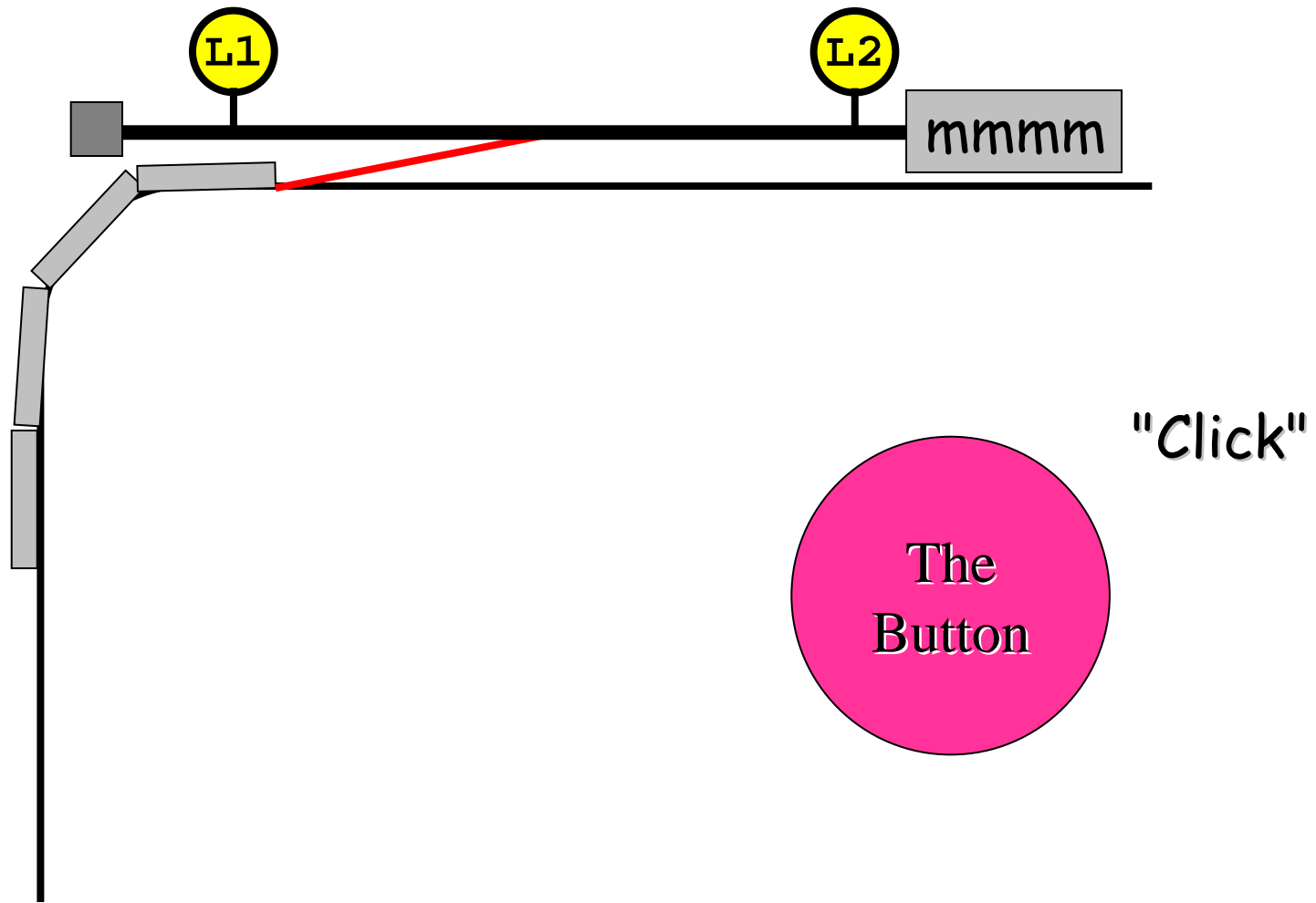
My Garage Door



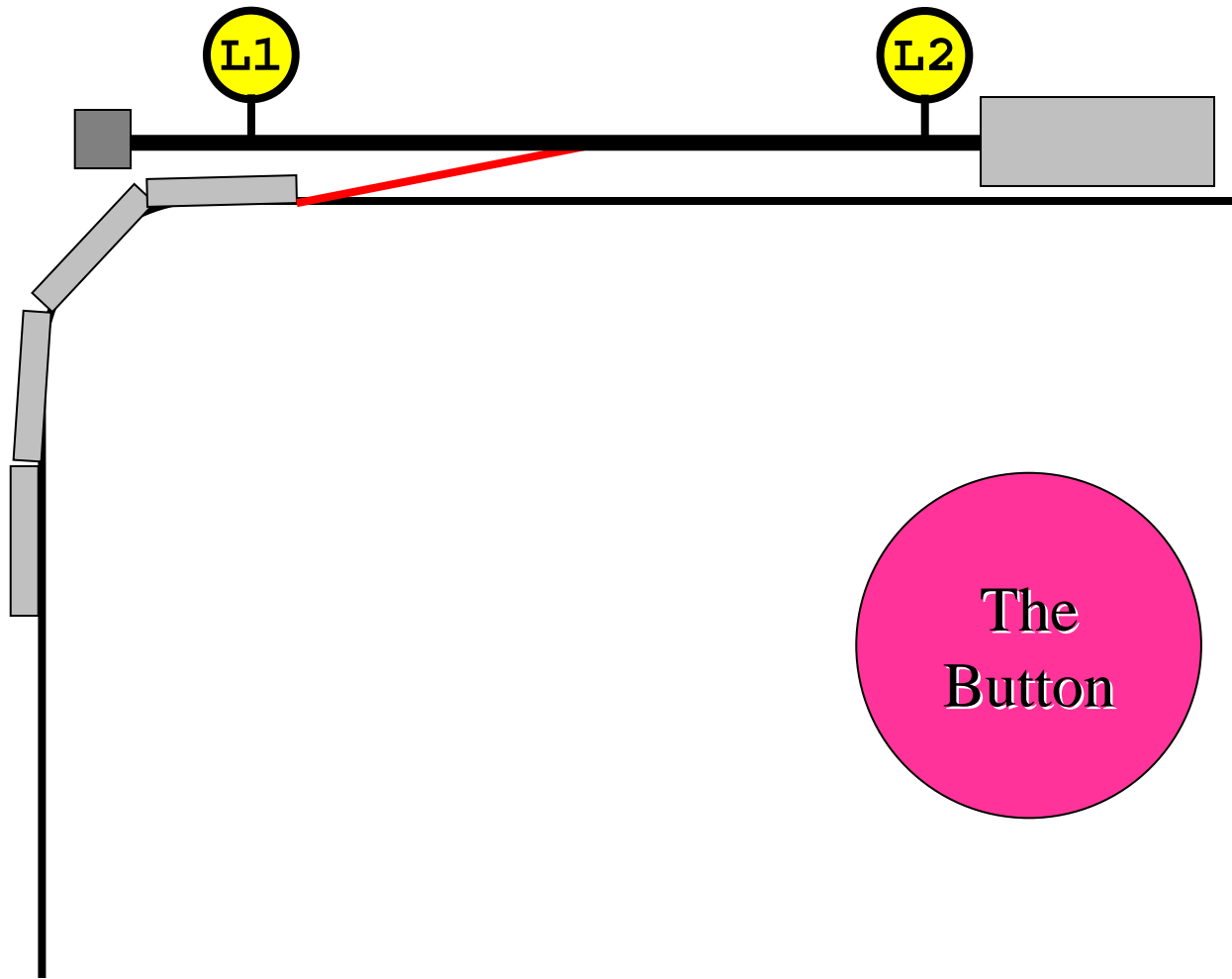
My Garage Door



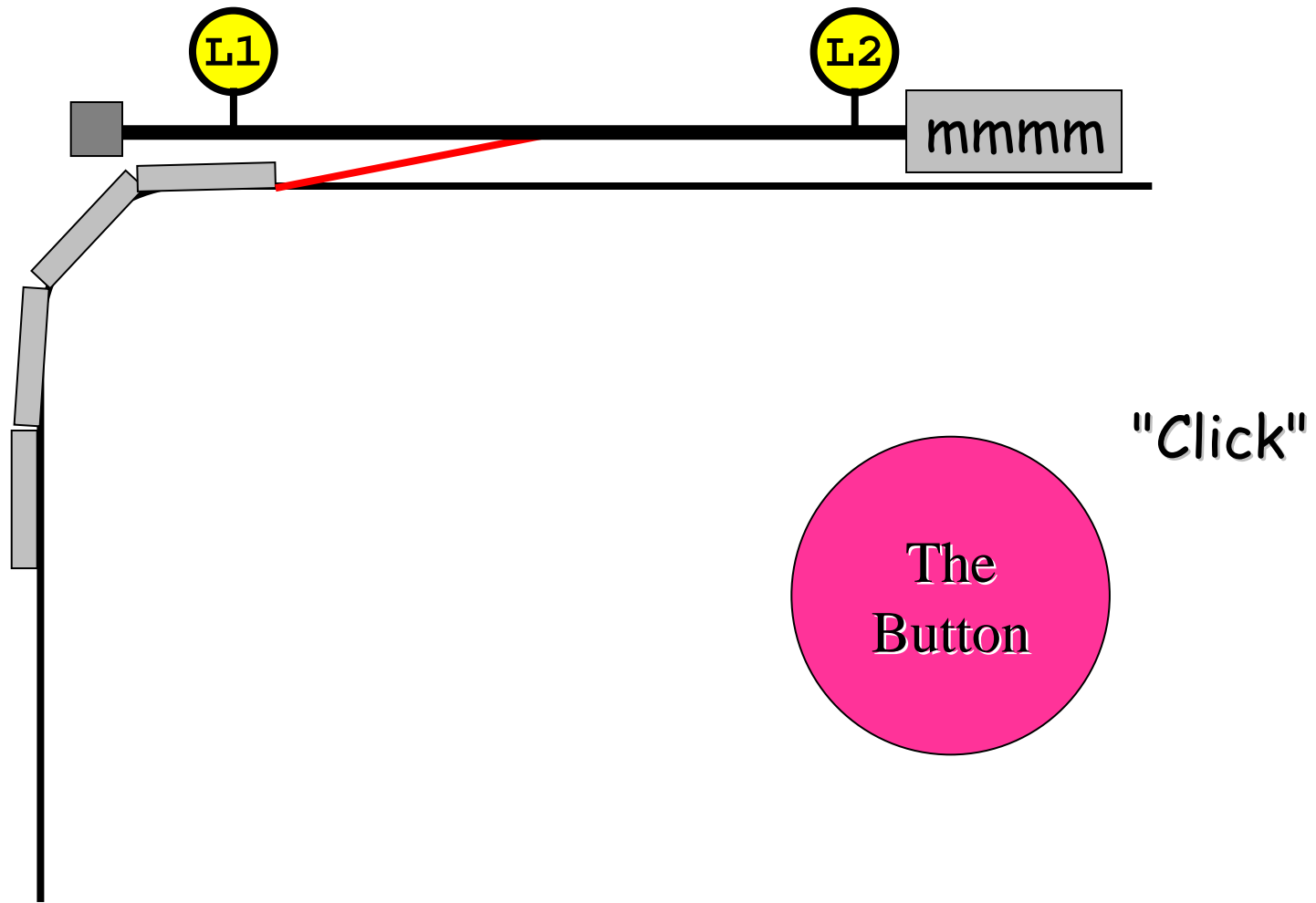
My Garage Door



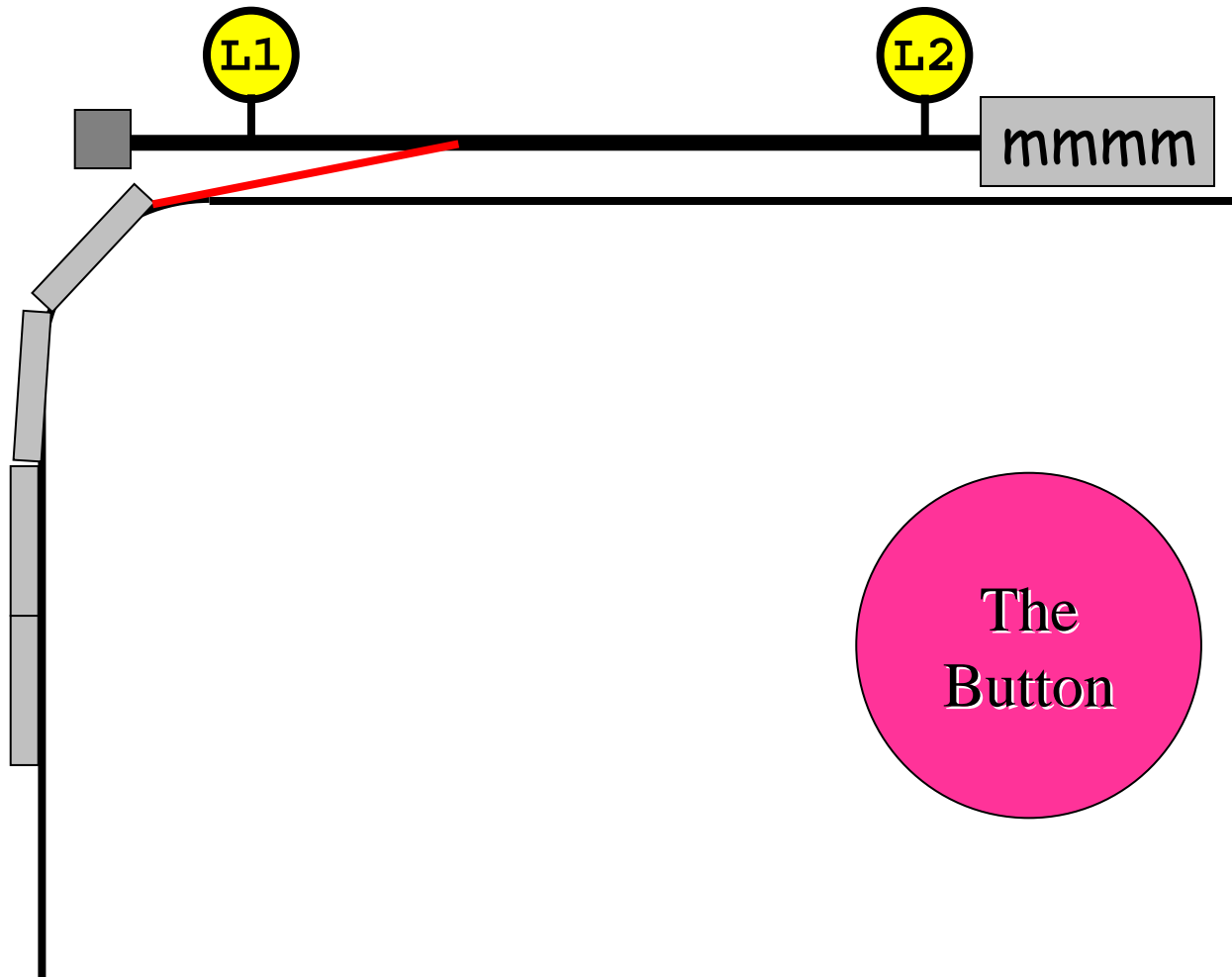
My Garage Door



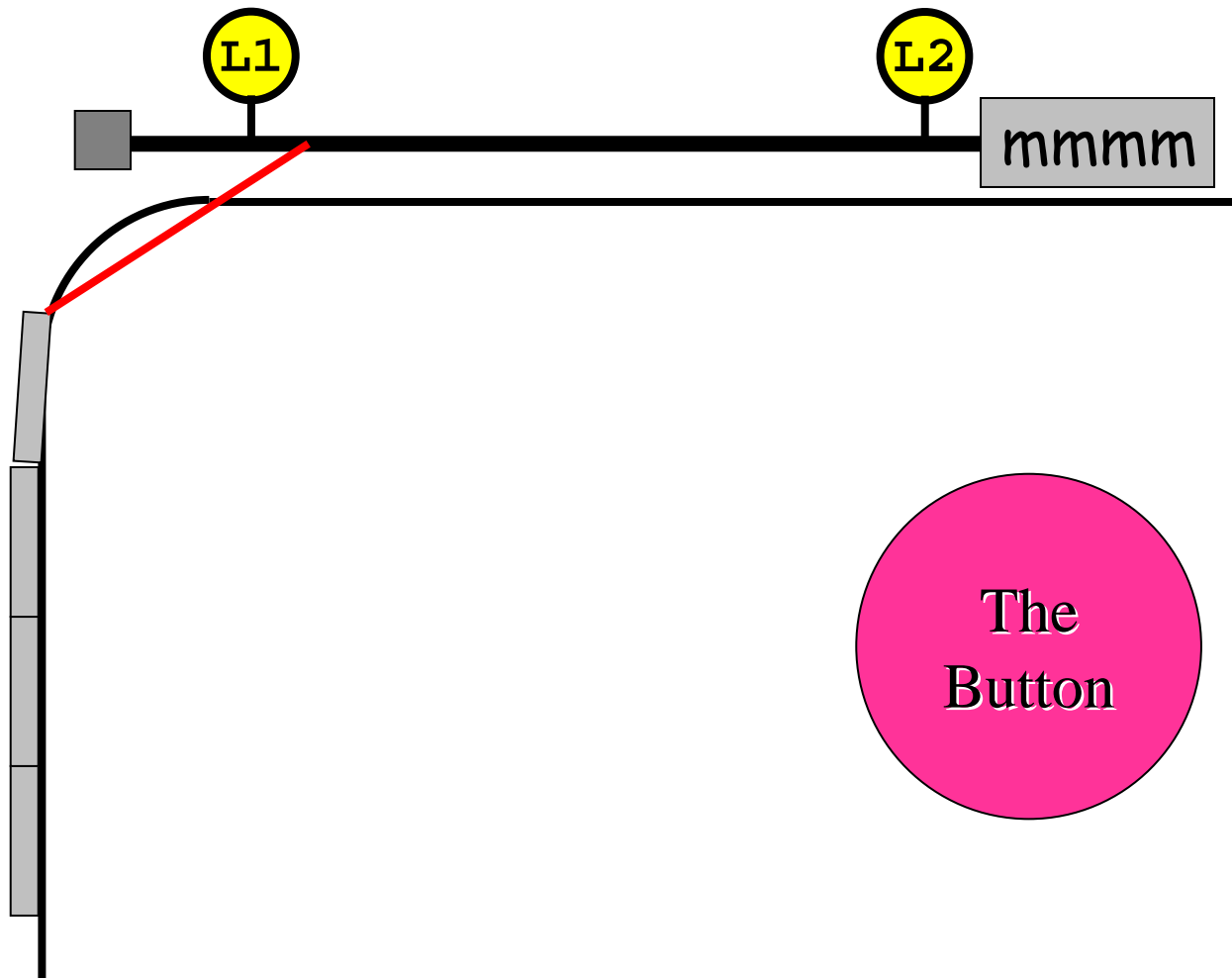
My Garage Door



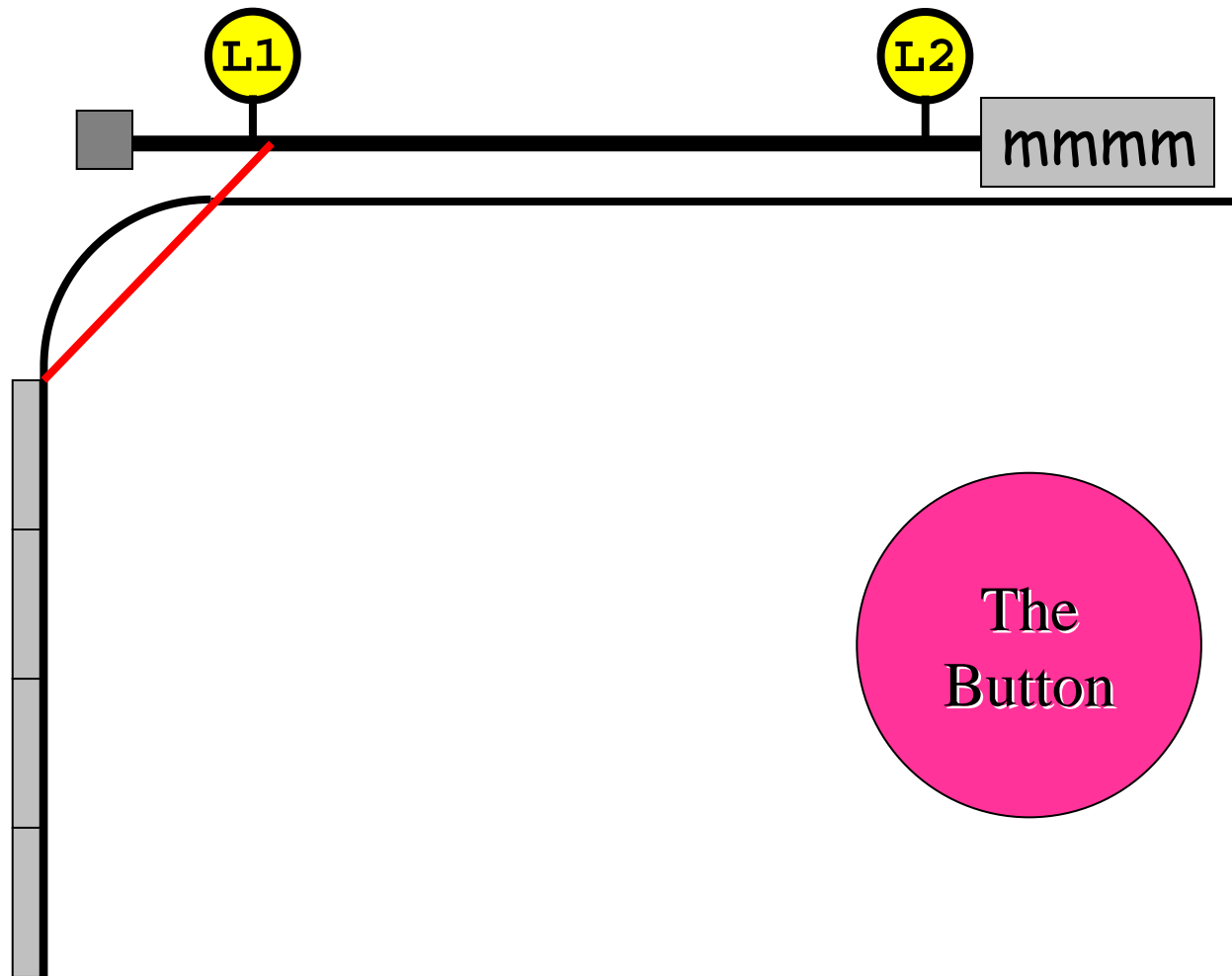
My Garage Door



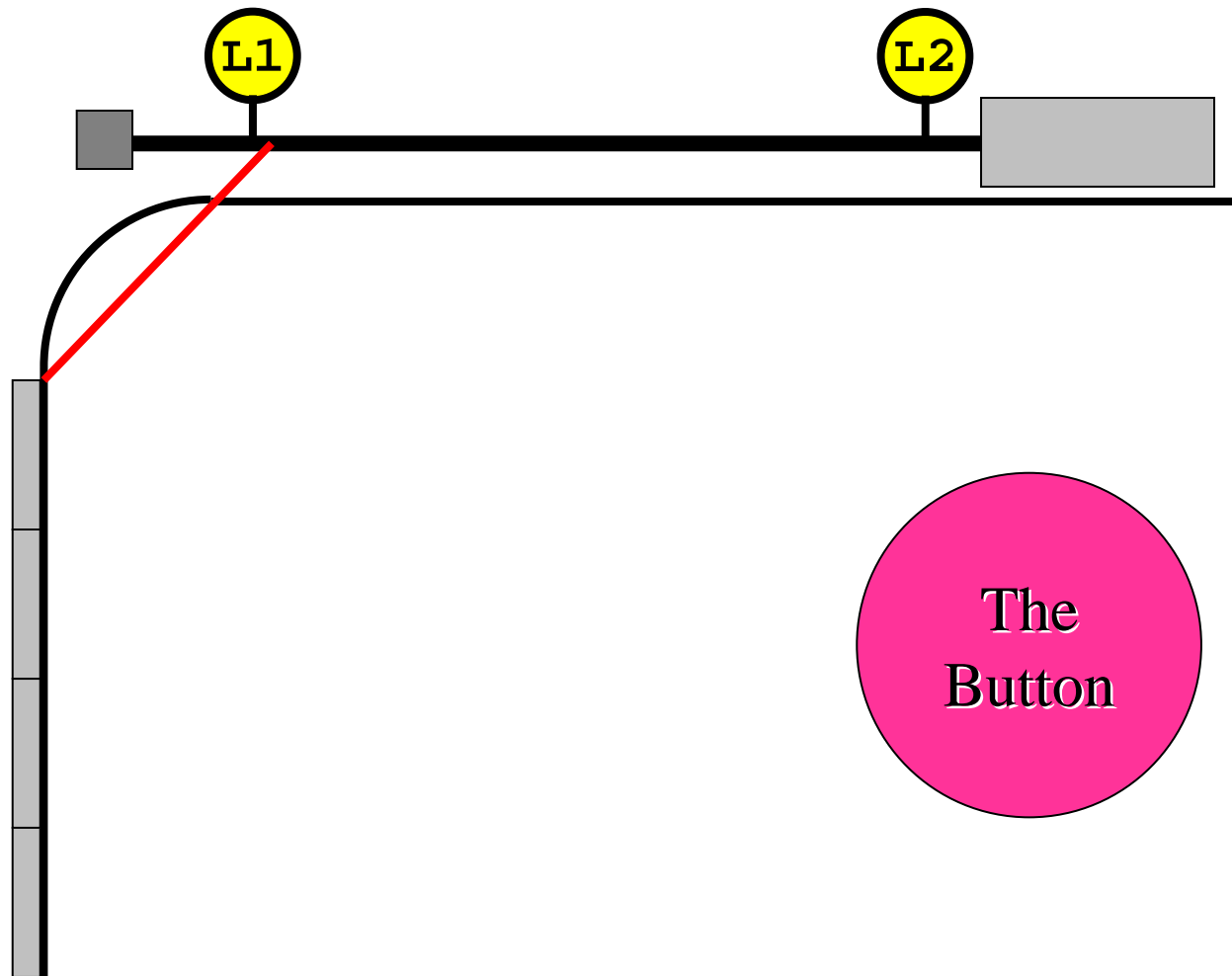
My Garage Door



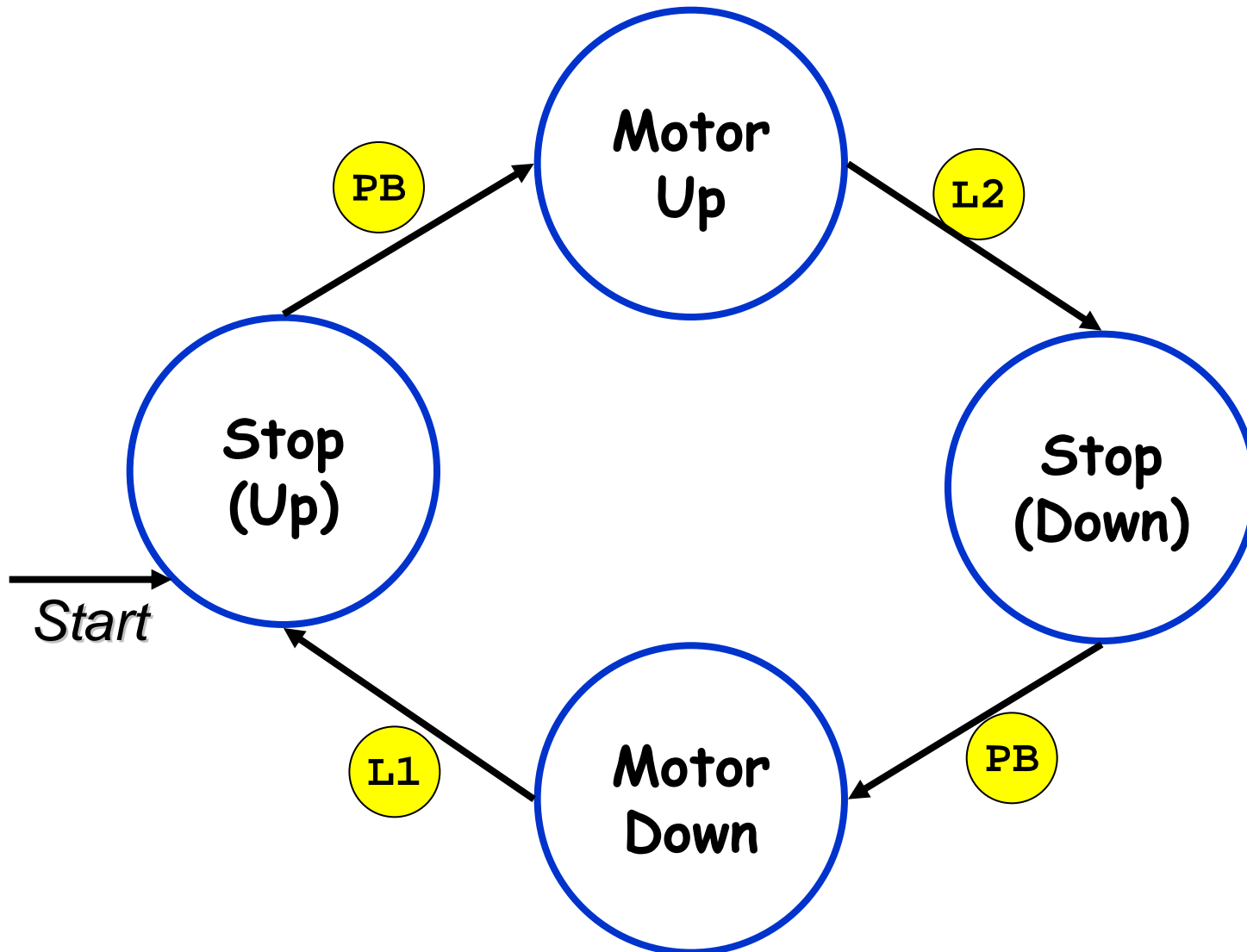
My Garage Door



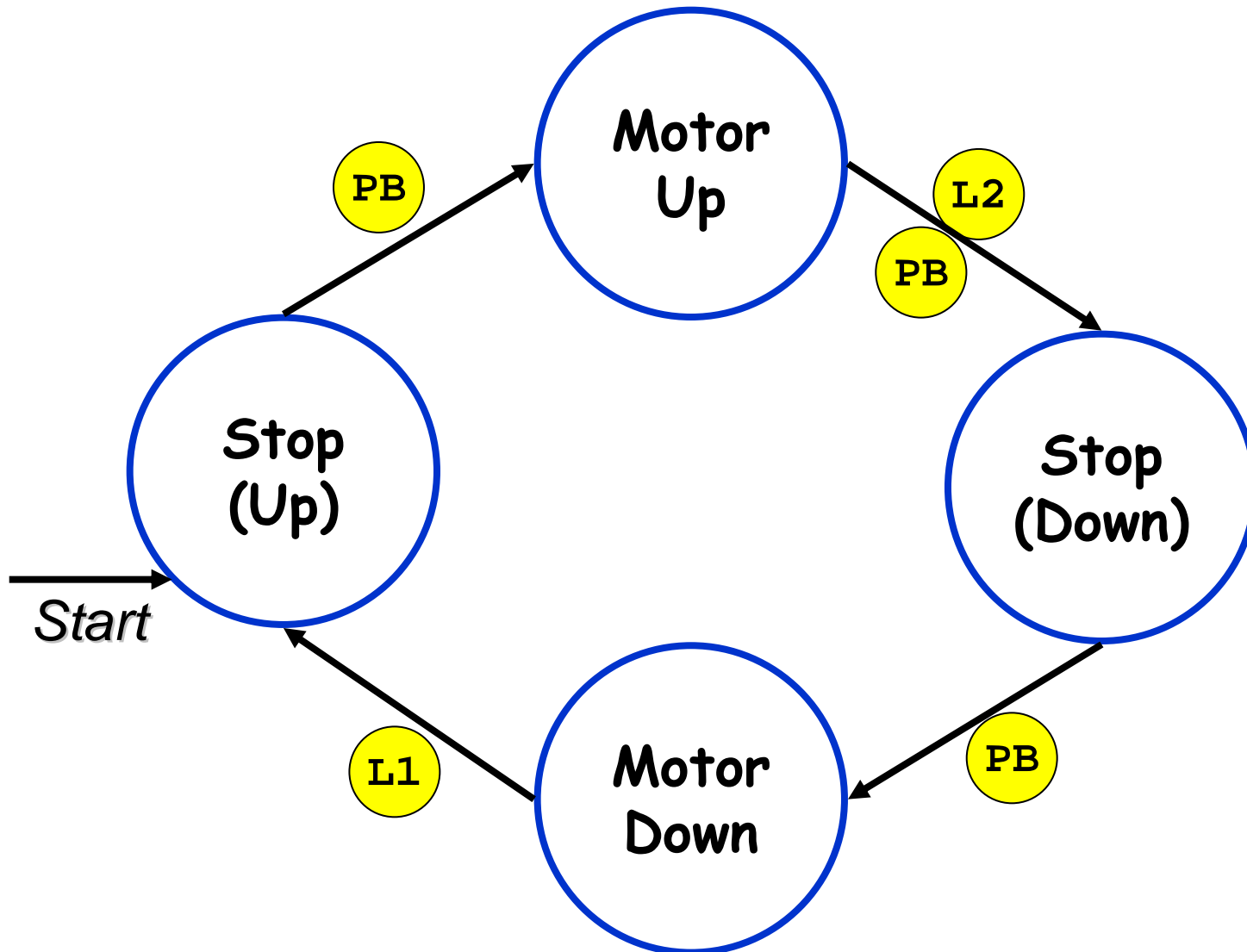
My Garage Door



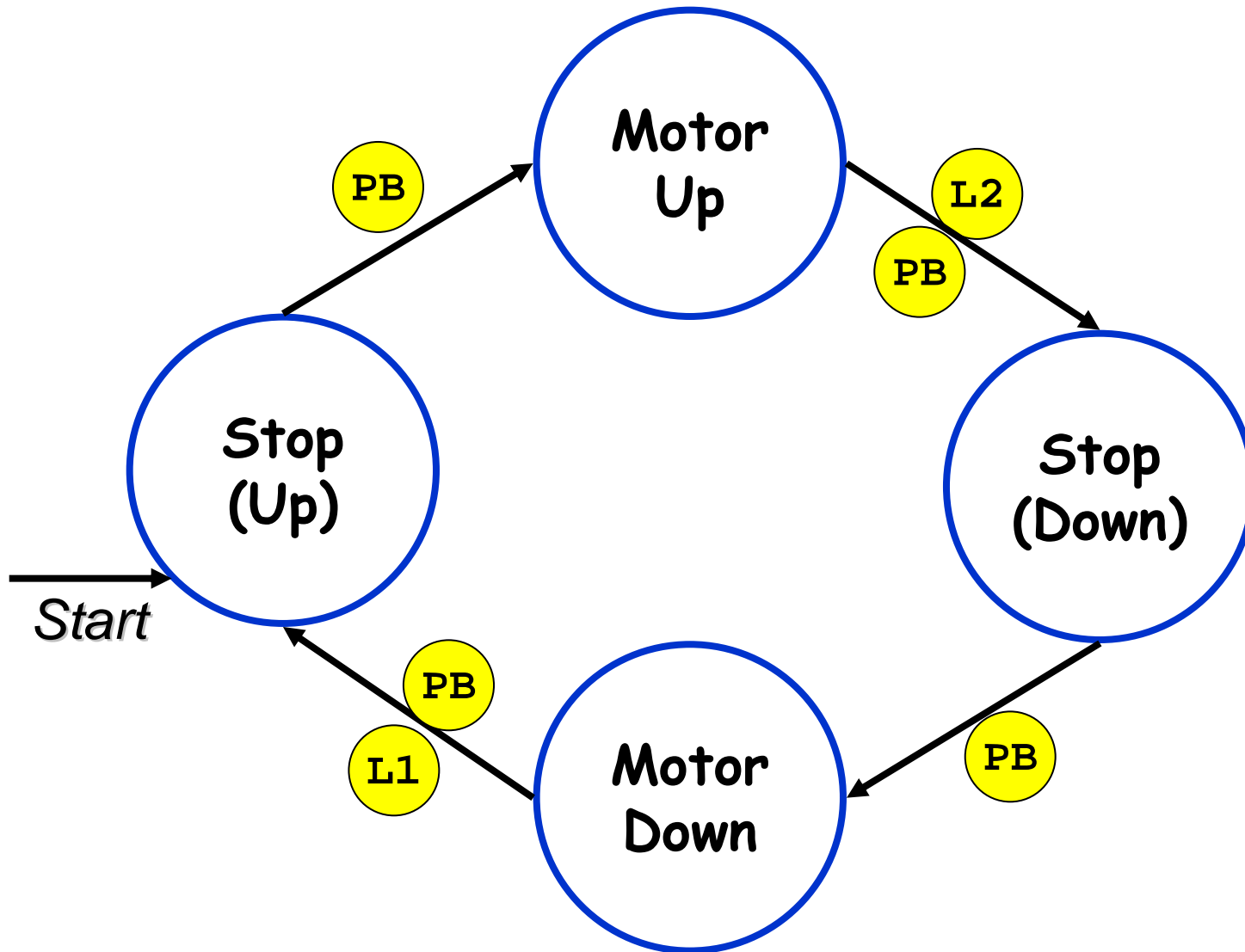
State Machine



State Machine



State Machine

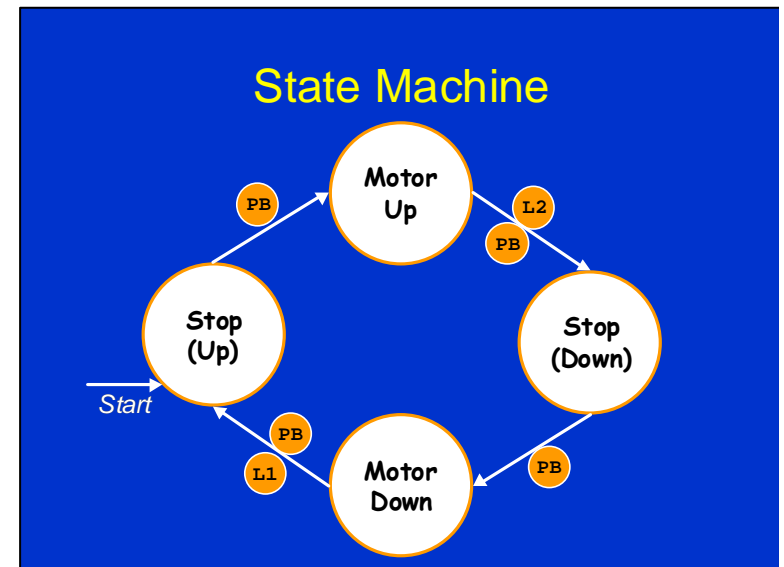


```

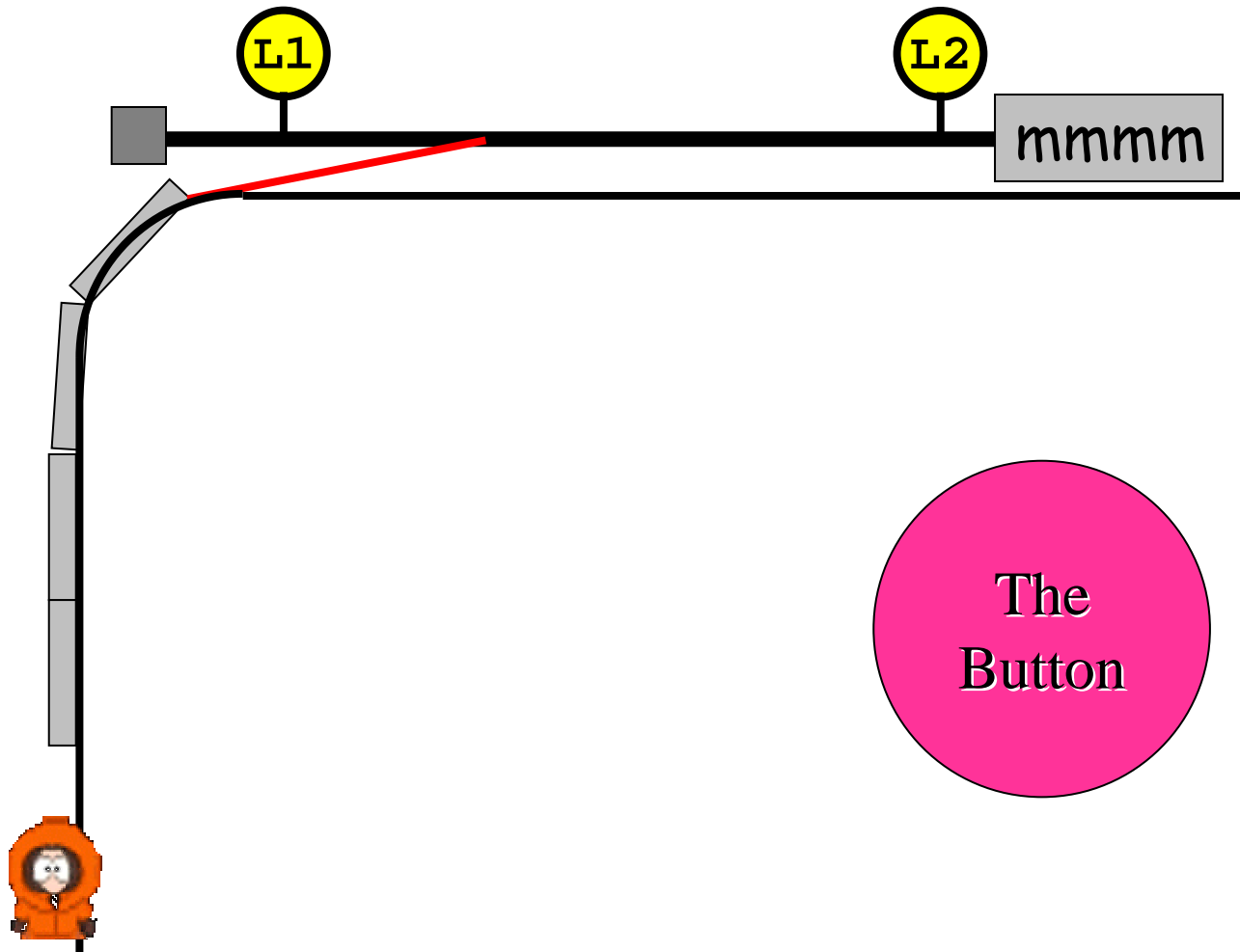
enum {L1, L2, PB, NONE} input;
enum {STOPUP, MOTORUP, STOPDOWN, MOTORDOWN} state;
while(1) {
    /* Get input here */
    switch(state) {
        case STOPUP:
            if(input == PB)
                state = MOTORUP;
            break;
        case MOTORUP:
            if (input == PB || input == L2)
                state = STOPDOWN;
            break;
        case STOPDOWN:
            if(input == PB)
                state = MOTORDOWN;
            break;
        case MOTORDOWN:
            if(input == PB || input == L1)
                state = STOPUP;
            break;
    }
    setMotor(state);
}

```

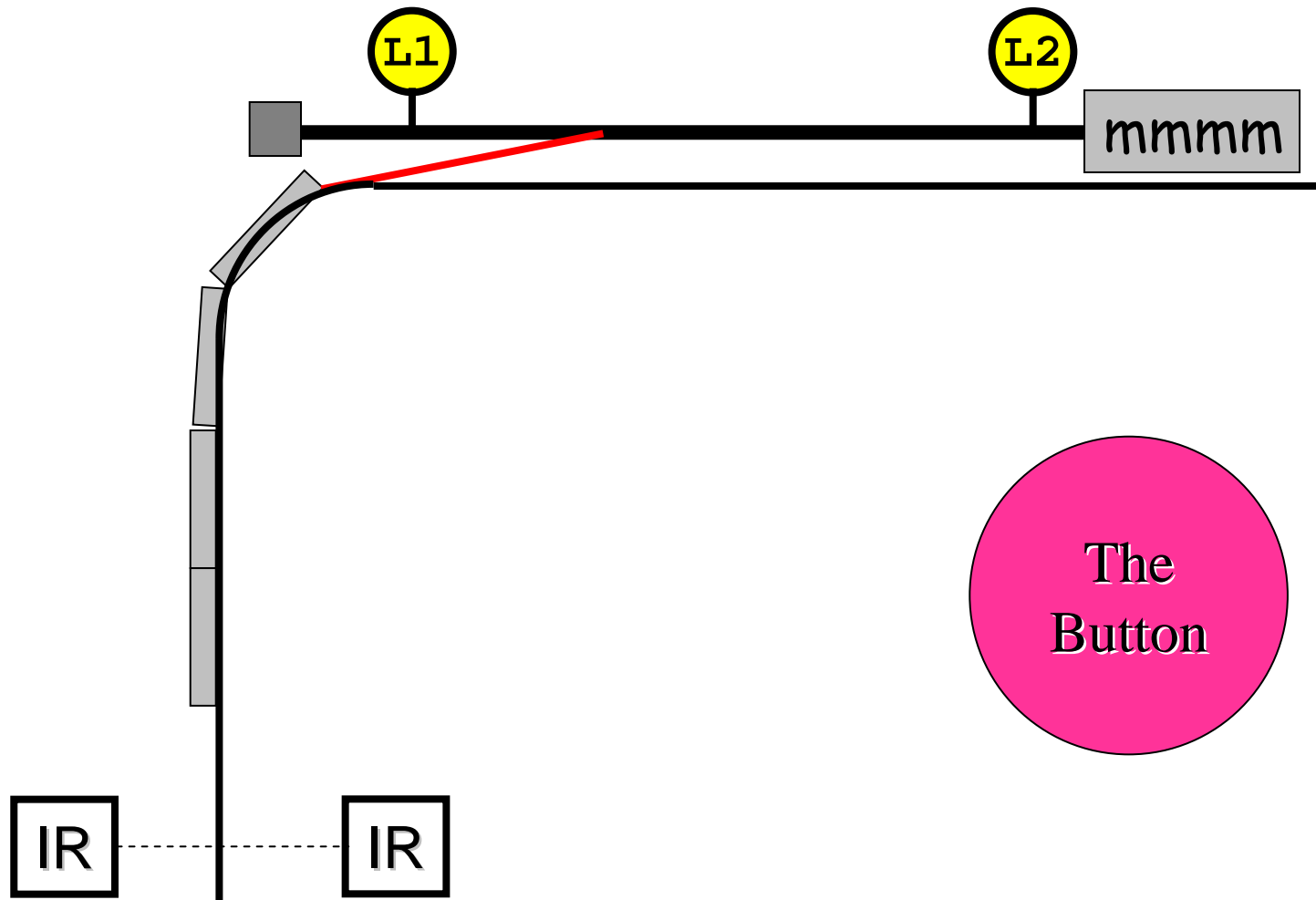
Warning!
Code not bulletproof
Do not use to control
actual apparatus!



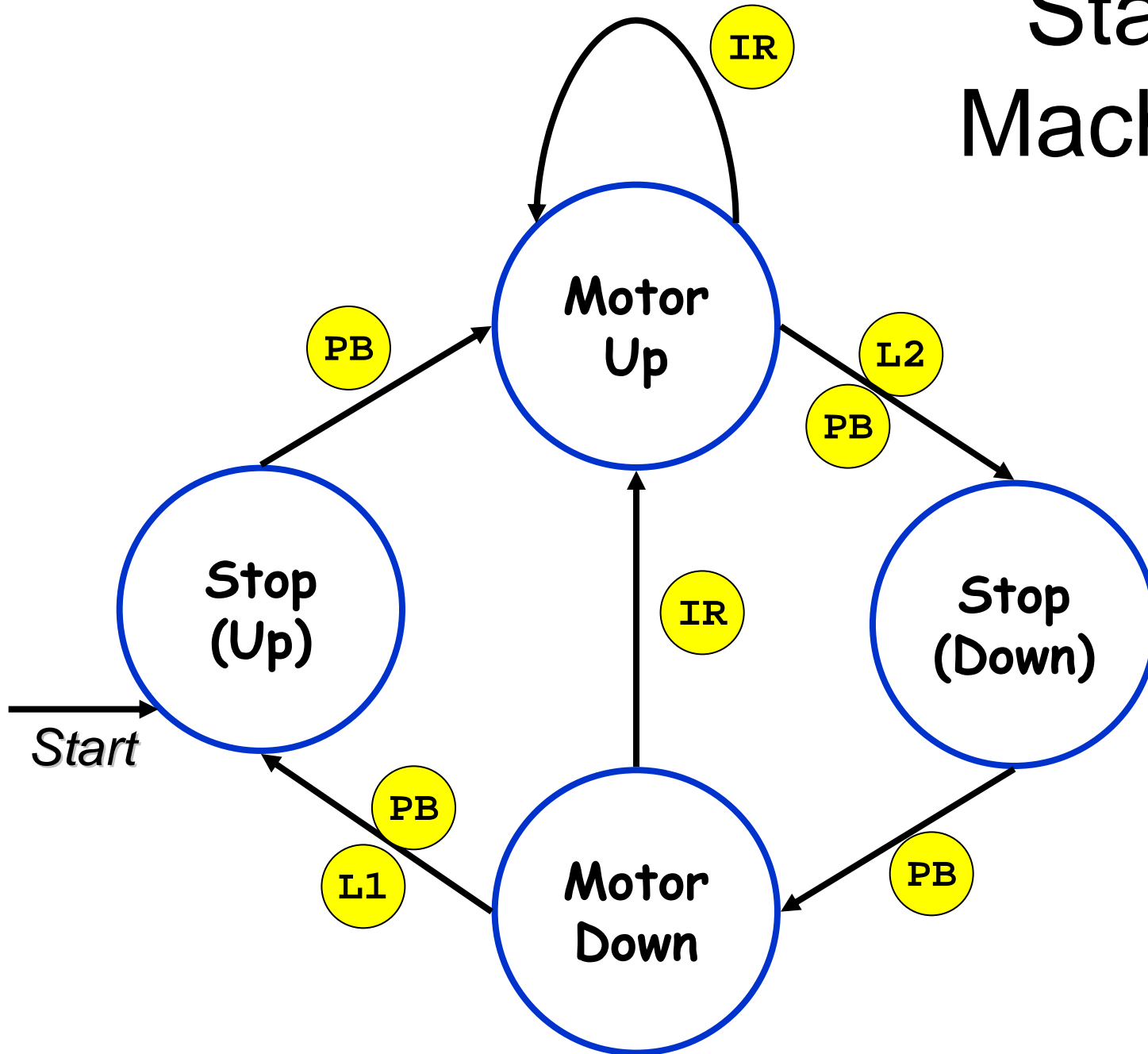
My Garage Door



My Garage Door



State Machine



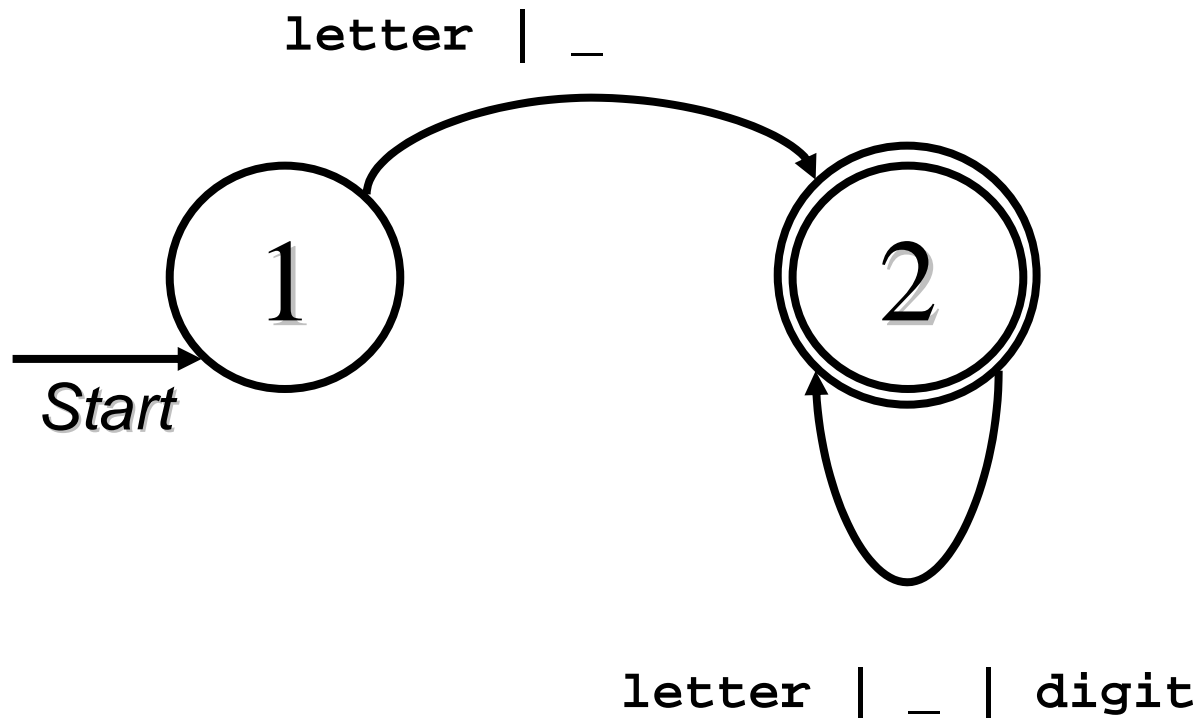
Questions?

So what does this have to do
with regular expressions?

Plenty!

Previews of Coming Attractions

[a-zA-Z_][a-zA-Z_0-9]*



Questions?

