Lecture 11

C pointers and Arrays / Event-based programming

Agenda

01. Pointers/Arrays – continued –
02. Events and Services Framework
03. Finite State Machines

Stuff

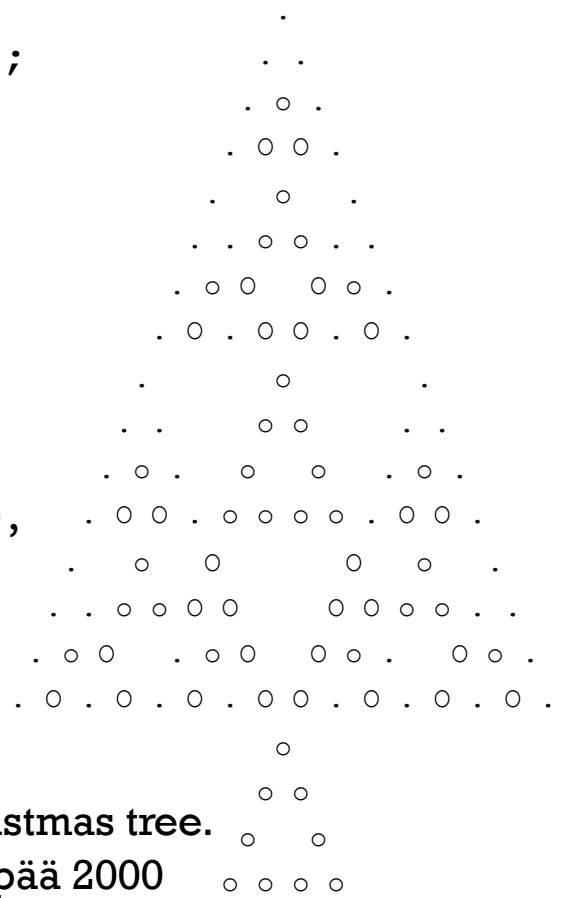
Parts fabrication:

- Parts are being fabricated for you by the TA's:
  - Pros: you don't have to spend hours at the machines to do this
  - Cons: you aren't learning how to use these machines
- There may be limited opportunities for those submitting late or doing a 2nd iteration. Each iteration will count as 1 late day.
- It's easier if you submit one DWG file with multiple parts rather than many separate DWG files.
- 3D printing is slower than expected (not sure about our machine allocation)

C can be an obscure language

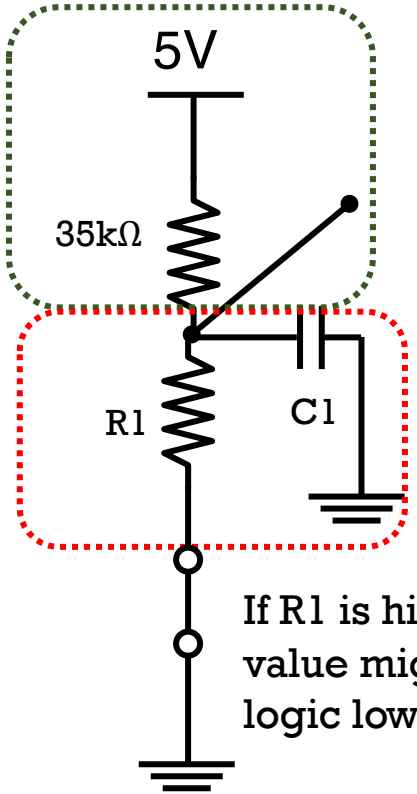
```
int a[4<<9],i;main(){for(a[40]=1;i++<1620;
 printf(i%80?"%c":"\n"," .oO"
 [a[i]&3]),a[i+79]+=a[i],
 a[i+81]+=a[i])a[1304]=a[1336]=0;}
```

```
int a[4 << 9], i;
main() {
  for (a[40] = 1; i++ < 1620;
       printf(i % 80 ? "%c" : "\n", " .oO" [a[i] & 3]),
         a[i + 79] += a[i], a[i + 81] += a[i])
    a[1304] = a[1336] = 0;
}
```



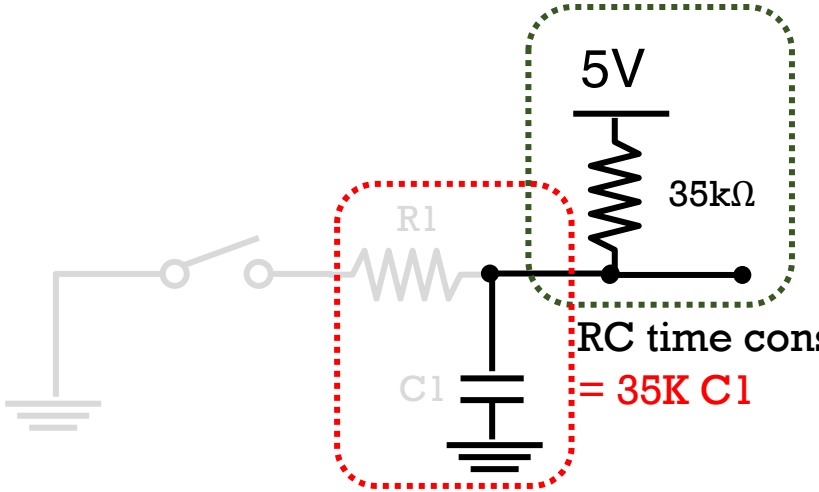
Output: Prints a christmas tree.
by Hannu Kankaanpää 2000

Lab 1 Filters on Switches



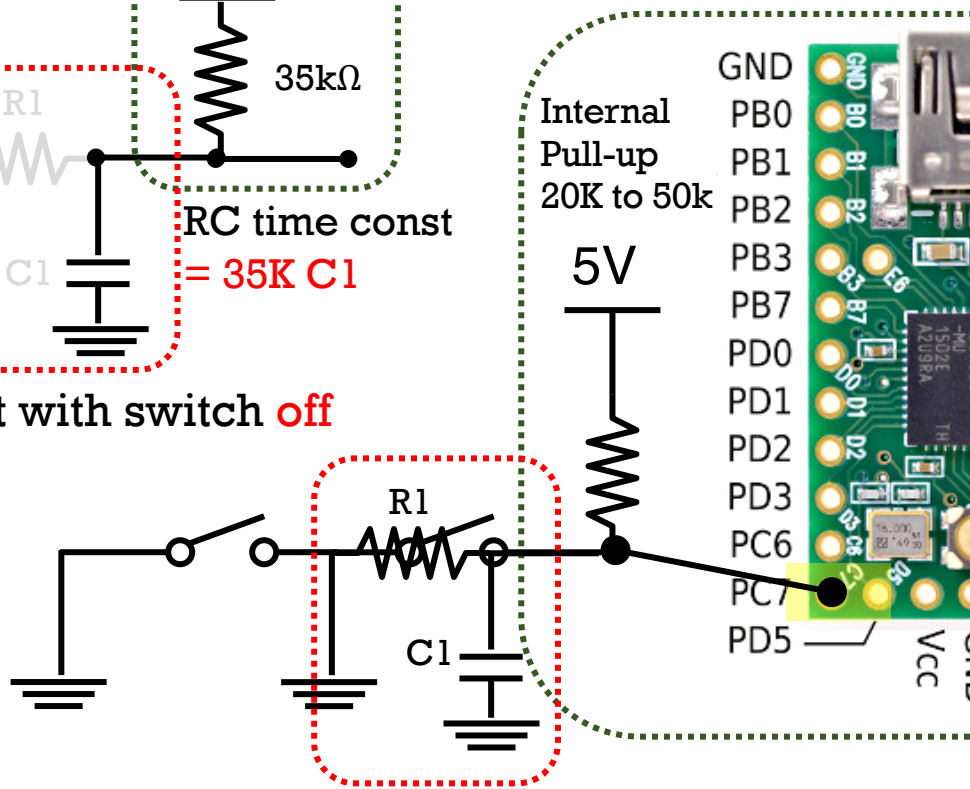
If R1 is higher than ~10K value might not be valid logic low.

Circuit with switch on



RC time const
= 35K C1

Circuit with switch off



01

Pointers and Arrays – part 2

Pointer declarations

- Declarations using * operator combined with a variable type

```
int    *ip;    // pointer to an integer  (2 bytes)
double *dp;    // pointer to a double    (8 bytes)
float  *fp;    // pointer to a float     (4 bytes)
char   *cp;    // pointer to a character (1 byte)
// Note: char* cp; is the same as char * cp; and char *cp;
```

- Size of pointers are all the same (16bit for ATmega)
- Size of the things they are point to may vary (important for arrays)

Using Pointers

```
#include "teensy_general.h"
#include "t_usb.h"
#define NLCR    m_usb_tx_char(10); m_usb_tx_char(13) // print newline


int main () {
  int  var = 20;   // actual variable declaration
  int  *ip;        // pointer variable declaration

  m_usb_init();

  ip = &var;  // store address of var in pointer variable
  m_usb_tx_hex( &var ); NLCR; // print address of var variable
  m_usb_tx_hex( ip ); NLCR;   // address stored in pointer variable
  m_usb_tx_uint( *ip ); NLCR; // access the value using the pointer
  while (1) ;
}
```

0AFA random address location compiler finds

Output:

0AFA

0AFA

20

Using Pointers

- Declaration

```
int     *fooptr;
```

- Assignment

```
fooptr = 42;
```

Will generate a warning when compiling

- Dereferencing

```
*fooptr = 42;
```

Stores 42 into the location at fooptr

```
int bar = *fooptr;
```

Loads the contents in fooptr into bar

- Passing pointers

```
int subroutine1( int *fooptr) {
  *fooptr = 20;   // changes content of pointer (like a global)
  return 1;
}
```

Using Pointers

- Declaration

```
int     *fooptr;
```

- Assignment

```
fooptr = 42;
```

Will generate a warning when compiling

- Dereferencing

```
*fooptr = 42;
```

Stores 42 into the location at fooptr

```
int bar = *fooptr;
```

Loads the contents in fooptr into bar

- Pointer arithmetic (4 operations)
  - ++ Increment address by one word (sizeof variable)
  - -- Decrement address by one word (sizeof variable)
  - + Add to address # of words (sizeof variable)
  - - Subtract from address # of words (sizeof variable)

Pointers and Arrays

- Arrays can be treated as pointers almost always

```
int array[];
array == &array[0] == &array
int *iptr;
*iptr == iptr[0]
```

- Some exceptions

```
int *arrayA, arrayB[8];
arrayA = arrayB;
```

`arrayA[i]` is the same as `arrayB[i]`

`*(++arrayA)` is the same as `arrayB[1];` but

`*(++arrayB)` will give an error. You can't change the value of the address of an array.

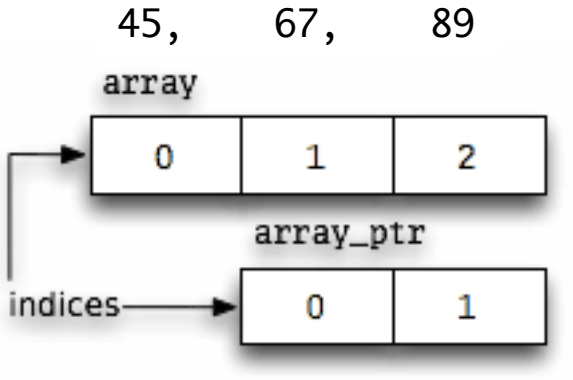
- Strings are arrays of characters that are NULL terminated, adding extra

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
char greeting[] = "Hello";
```

Pointers and Arrays

- Q1: What does the following print?

```
int array[] = { 45, 67, 89 };
int *array_ptr = &array[1];
printf("%i\n", array_ptr[1]);
```



Pre-increment and post-increment

```
void somefunction()
{
  int x = 10, a;
   a = ++x;  // Value of x will change before assignment

  m_usb_tx_uint( a);
  m_usb_tx_uint( x);
}
```

```
x = x+1;
a = x;
```

Output:

11

11

```
void somefunction()
{
  int x = 10, a;
   a = x++;  // Value of x will change after assignment

  m_usb_tx_uint( a);
  m_usb_tx_uint( x);
}
```

```
a = x;
x = x+1;
```

Output:

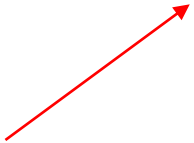
10

11

Q2: Pointer Arithmetic (what 4 numbers will print?)

- Indexing an array with pointer math

```
int array[] = { 45, 67, 89 };
int *array_ptr = array;
somecode() {
  m_usb_tx_uint(*(array_ptr++));
  m_usb_tx_uint(*(array_ptr++));
  m_usb_tx_uint(*array_ptr);
  m_usb_tx_uint(*(array+2));
}
```



*array_ptr;
array_ptr=array_ptr+1;

Q2: Pointer Arithmetic (what 4 numbers will print?)

- Indexing an array with pointer math

```
int array[] = { 45, 67, 89 };
int *array_ptr = array;
somecode() {
  m_usb_tx_uint(*(array_ptr++));
  m_usb_tx_uint(*(array_ptr++));
  m_usb_tx_uint(*array_ptr);
  m_usb_tx_uint(*(array+2));
}
```

*array_ptr;
array_ptr=array_ptr+1;

parentheses has no effect in many post increment cases

Extra stuff about pointers

- In general, you can probably get through this course without using pointers.
- Pointers which point to NULL or 0 are a special case. Usually NULL is an invalid address, often indicating uninitialized pointers or other special cases.
- Multiple indirection is valid (I don't recommend it…)

```
int    a =  3;
int   *b = &a;
int  **c = &b;
int ***d = &c;
```

- Function pointers. You can treat functions as variables by passing pointers to functions. (advanced topic – not necessary in this course). http://boredzo.org/pointers/

02

Events and Services Framework

Programming Embedded Systems

Program Structure:

- often asynchronous
- "simultaneous" inputs & outputs
- sequences unknowable, re-orderable
- no "end" or "exit"

Inputs:

- sensors (switches, light sensors, voltages, etc.)
- timers
- user inputs (keypad, push-buttons)

Outputs:

- update a display
- move something
- switch something on or off
- in general → CHANGE SOMETHING

Q3A Lab 1 Loop Exercise

- Assume you have set up a timer to use OCR1A as a 50 Hz PWM signal and hooked up an LED to the OC1A pin so that writing to OCR1A will set the pulse width of the LED.
  - `OCR1A = 255;` will be 100% on,
  - `OCR1A = 0;` will be 100% off
- Write a loop (or loops) that will cause the LED to take 255 steps to grow in intensity to 100% over and over.

```
while(1) {




}
```

Q3B Loop Exercise

- Write code to have a second LED attached to OCR1B to grow in intensity in sync (50Hz) 255 steps.

```
while(1) {
  for (int i=0; i<255; i++) {
    OCR1A = i;
    _delay_ms(20);
  }
}
```

Q3C Tought exercise

- How can we have the second LED change it's frequency independent of the first?

```
while(1) {
  for (int i=0; i<255; i++) {
    OCR1A = i;
    _delay_ms(20);
  }
  for (int i=0; i<255; i++) {
    OCR1B = i;
    _delay_ms(20);
  }
}
```

```
while(1) {
  for (int i=0; i<255; i++) {
    OCR1A = i;
    OCR1B = i;
    _delay_ms(20);
  }
}
```

Events and Services Framework

- Conceptual framework
- An excellent method for *Event-Driven* programming
- Emphasizes design first

Events and Services Framework

RULE #1:

Recognize that tasks break down ONLY into two fundamental classes:

a. Event Detectors

b. Services

Events and Services Framework

CORROLARY TO RULE #1:

- Keep Event-Detector and Service routines as short as possible.
  - Try NOT to have too many (or long) delays()
- Must implement "Non-Blocking" routines

Blocking code has indefinite waits,
e.g. `while (!something) wait;`

If you think you need this
`while()`, this is a hint
to add another event

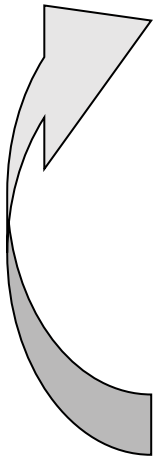
Writing Events and Services Programs

Complete program structure:

- Initialize hardware and software
- Continually test for Events
  - Round-robin scanning
  - "Non-blocking" code

```
If (test1) dosomething()
If (test2) dosomethingelse()
Etc.
```

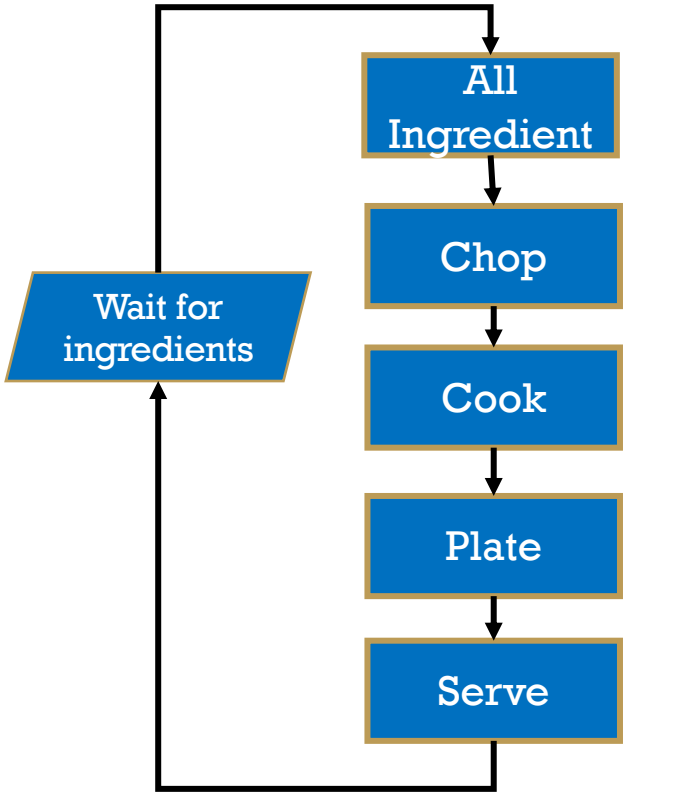
- Perform Service(s) when Event detected
  - "Non-blocking" code
- Repeat



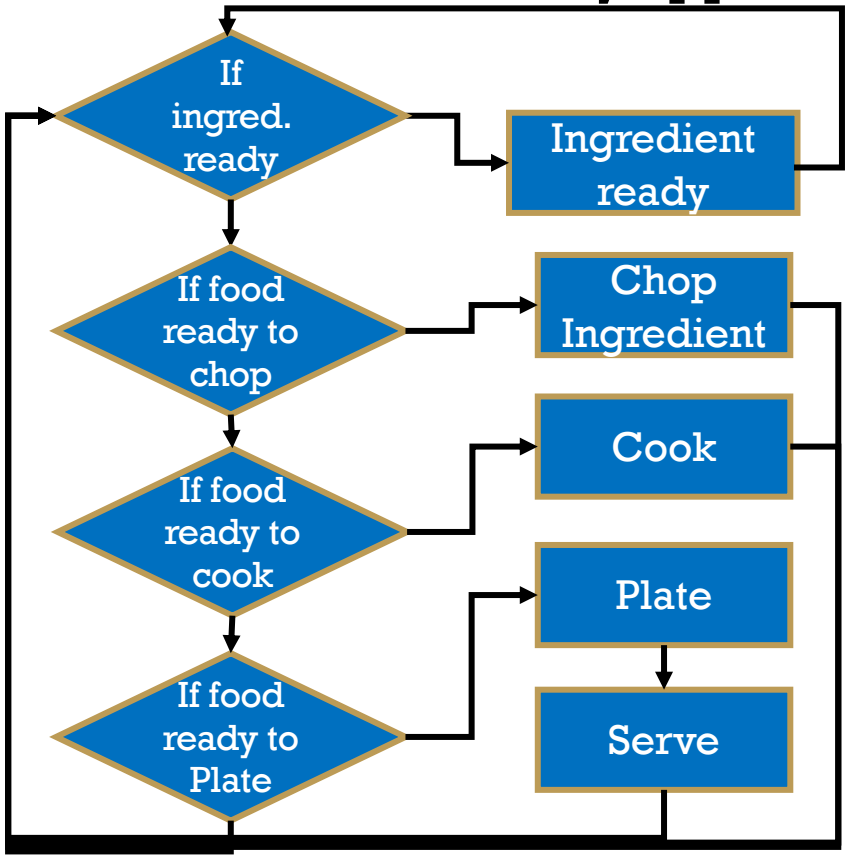
Overcooked video game

Analysis of Server Client Process

- Process one dish at a time.

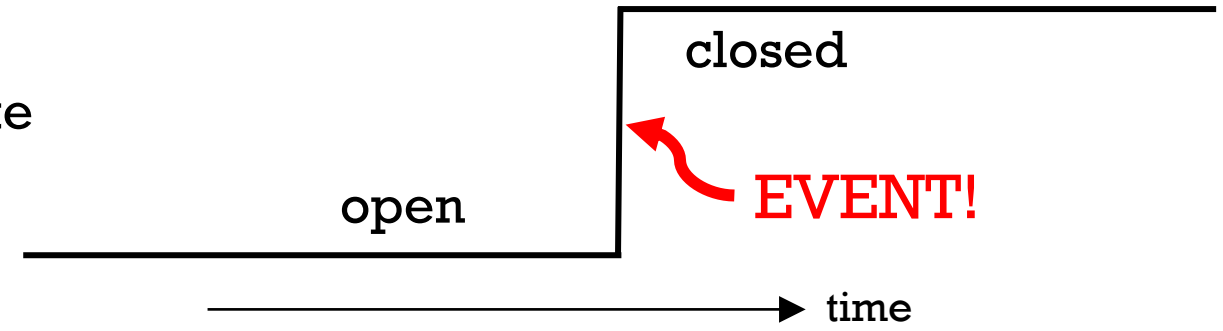


- Process tasks as they appear



So, what is an event?

Example:
Switch state



Instantaneous change
(e.g a change in state)

It is NOT the
value of a state

In subroutines, use `static` local variable
to detect events changed between calls

How to unblock blocking code?

- Break up long routines
- Add events

- Allow for “simultaneous” processes to occur as services triggered by events.

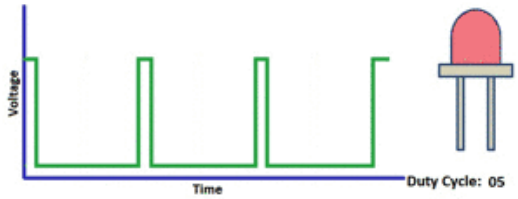
Breaking up for loop

```
while(1) {
  if (i++ < 255) {
        ledPWMservice();
  }
  else i=0;



  _delay_ms(50);
}
```

```
void ledPWMservice() {
   OCR1A = i;
   m_usb_tx_uint(i);
}
```

```
while(1) {
   for (i=0; i<255; i++) {
      OCR1A = i;
      m_usb_tx_uint(i);
      _delay_ms(50);
   }
}
```

Portion of an LED ramp function increasing duty cycle of PWM



What happens when we have two PWM channels

Breaking up for loop

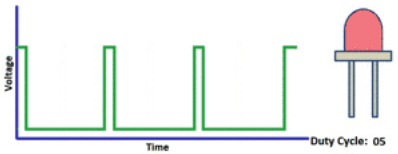
```
while(1) {
  if (i++ < 255) {
        ledPWMservice();
  }
  else i=0;
  if (j++ < 180) {
        ledPWMservice2();
  }
  else j=90;
  _delay_ms(50);   // share same delay(50)
}
return 0;
```

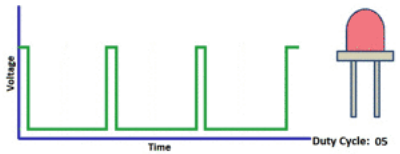
```
int i,j;
while(1) {
  for (i=0; i<255; i++) {
    OCR1A = i;
    m_usb_tx_uint(i);
    _delay_ms(50);
  }
  for (j=90; j<180; j++) {
    OCR1B = j;
    m_usb_tx_uint(j);
    _delay_ms(50);
  }
}
```

Two LEDs controlled by output capture PWM Timer1A and Timer1B

```
void ledPWMservice() {
    OCR1A = i;
    m_usb_tx_uint(i);
}
void ledPWMservice2() {
    OCR1B = j;
    m_usb_tx_uint(j);
}
```

// i and j are globals





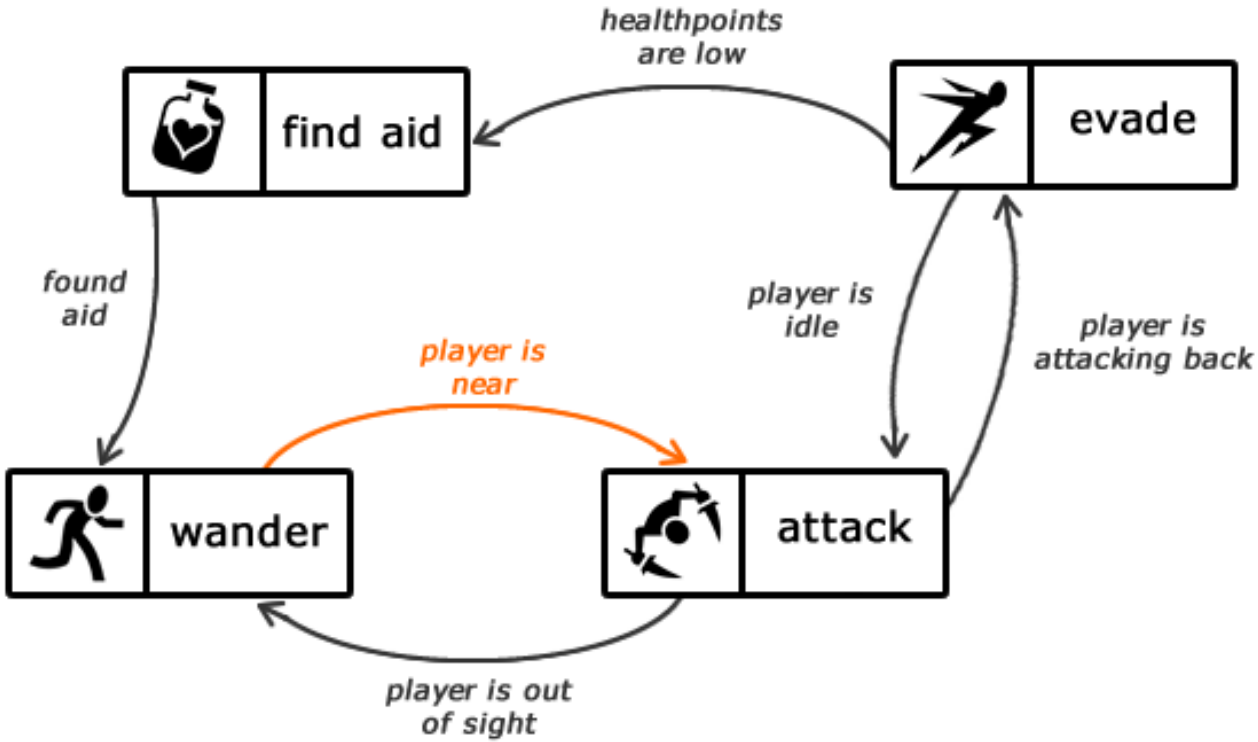
Events and Services Framework

- Breaking into short services emphasizes design first
- Gets you to use structures that:
  1) make it clear how to define the low-level functions
  2) make debugging code simpler (even before coding!)

03

Finite State Machines

Video Game AI using FSM



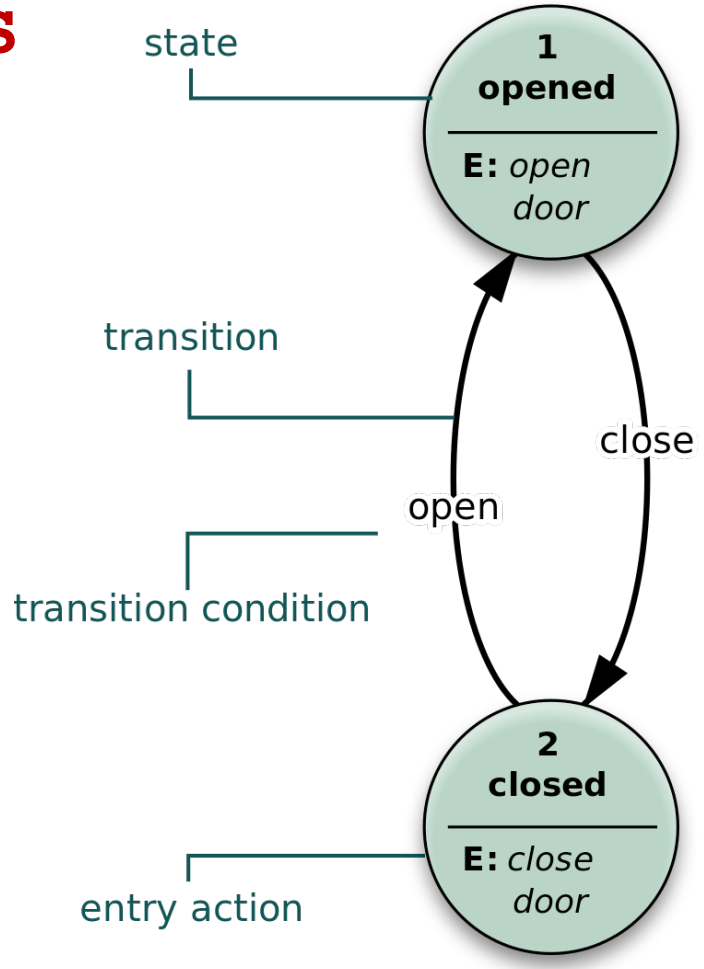
All icons made by Lorc, and available on http://game-icons.net.

https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867

Finite State Diagrams

- Abstract description of system behavior.
- Sometimes well suited for embedded applications

- There are many different representations and implementations.

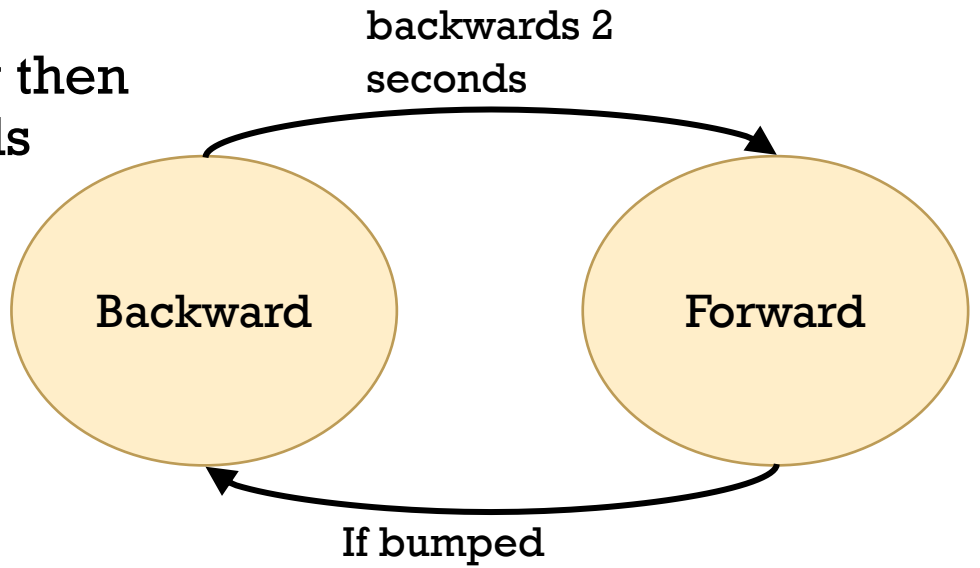


Finite state machine example

Behavior:

Continuously moves forward

But if it bumps into something then move backwards for 2 seconds



Q4: Write pseudo-code for this program

Pseudo Code Example

Entry action

```
Start state of moving forward
        set state FORWARD;
        Turn on motor forward direction
```

While in state

```
Loop Moving forward
        check if bumped to start backward
```

Entry action

```
Start state of moving backward
        set state BACKWARD;
        Turn on motor backward direction
```

While in state

```
Loop Moving backward
        if 2 secs passed start forward
```

Pseudo Code Example (Events and Services)

Events

Services

```
Loop forever
if in FORWARD state,  check if bumper           then start backwards
if in BACKWARD state, check if 2 seconds passed then start forward

if in FORWARD state,                            step forward
if in BACKWARD state,                           step backwards
```

```
Start state of moving FORWARD
        set state FORWARD;
        Turn on motor forward direction
Moving forward
        ;
```

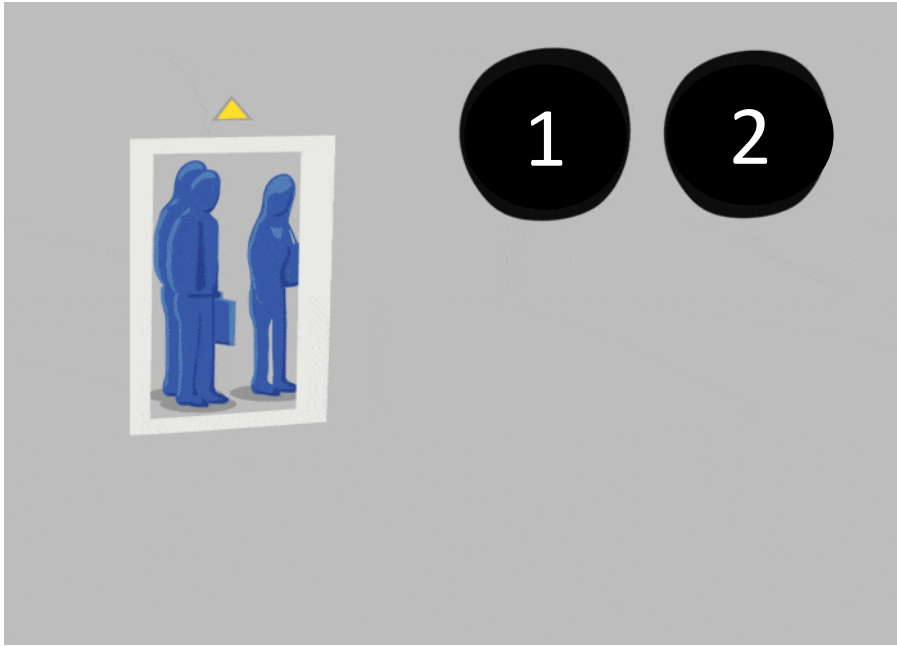
```
Start state of moving backward
        set state backward;
        Turn on motor backward direction
Moving backward
        ;
```

```
void startForward() { // Start state of moving forward
  state = FORWARD;  // set state forward;
  setMotorDirection(1); // Turn on motor forward direction
}

void duringForward() { // Moving forward
  if (bumped()) startBackward(); // check if bumped to move backward
}

void startBackward() {// Start state of moving backward
  state = BACKWARD;  // set state backward;
  setMotorDirection(-1);  //  backward direction
}

void duringBackward() {// Moving backward
  if (twosecondspassed()) startForward();  // if 2 secs passed move forward
}
...

int main() {
 ...
  while (1) {
    if (state == FORWARD) duringForward();
    if (state == BACKWARD) duringBackward();
  }
}
```

```
#define FORWARD 1
#define BACKWARD 2
int state; // global variable
```

Two Floor Elevator FSM

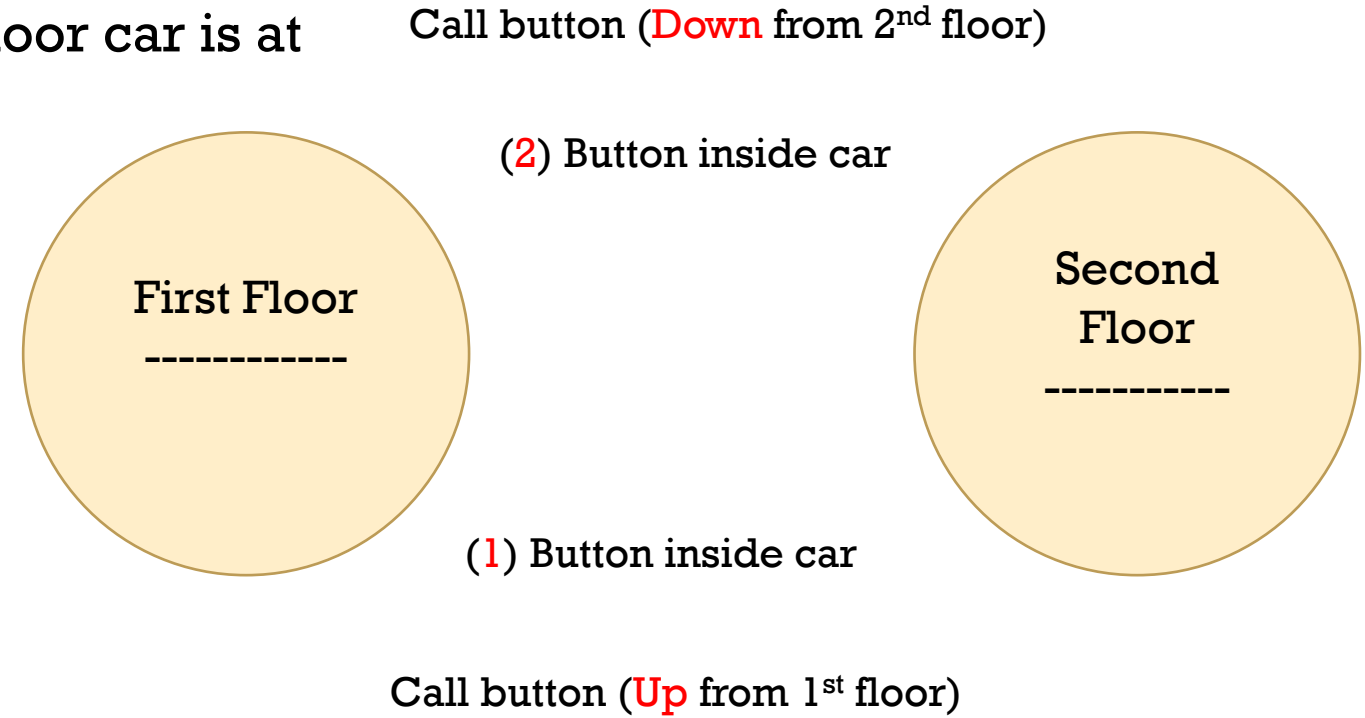
- Car floor [1st, 2nd]
- Door state [open/close]
- Floor 1 button [on / off]
- Floor 2 button [on / off]
- 1stF call (going up) [on / off]
- 2ndF call (going down) [on / off]



Finite state machine for an elevator

- Highest level two states: based on which floor car is at

Q5 Draw (no hold) arrows for the transitions between states?



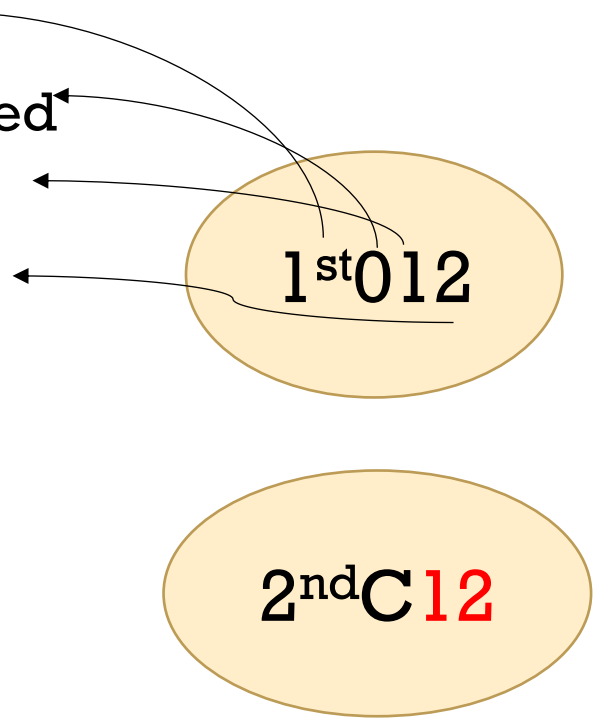
Labeling for Elevator States

More States?

• On which floor	1st	2nd
• Door state	Opened	Closed
• Floor 1 button	**1** on	1 off
• Floor 2 button on	**2** on	2 off

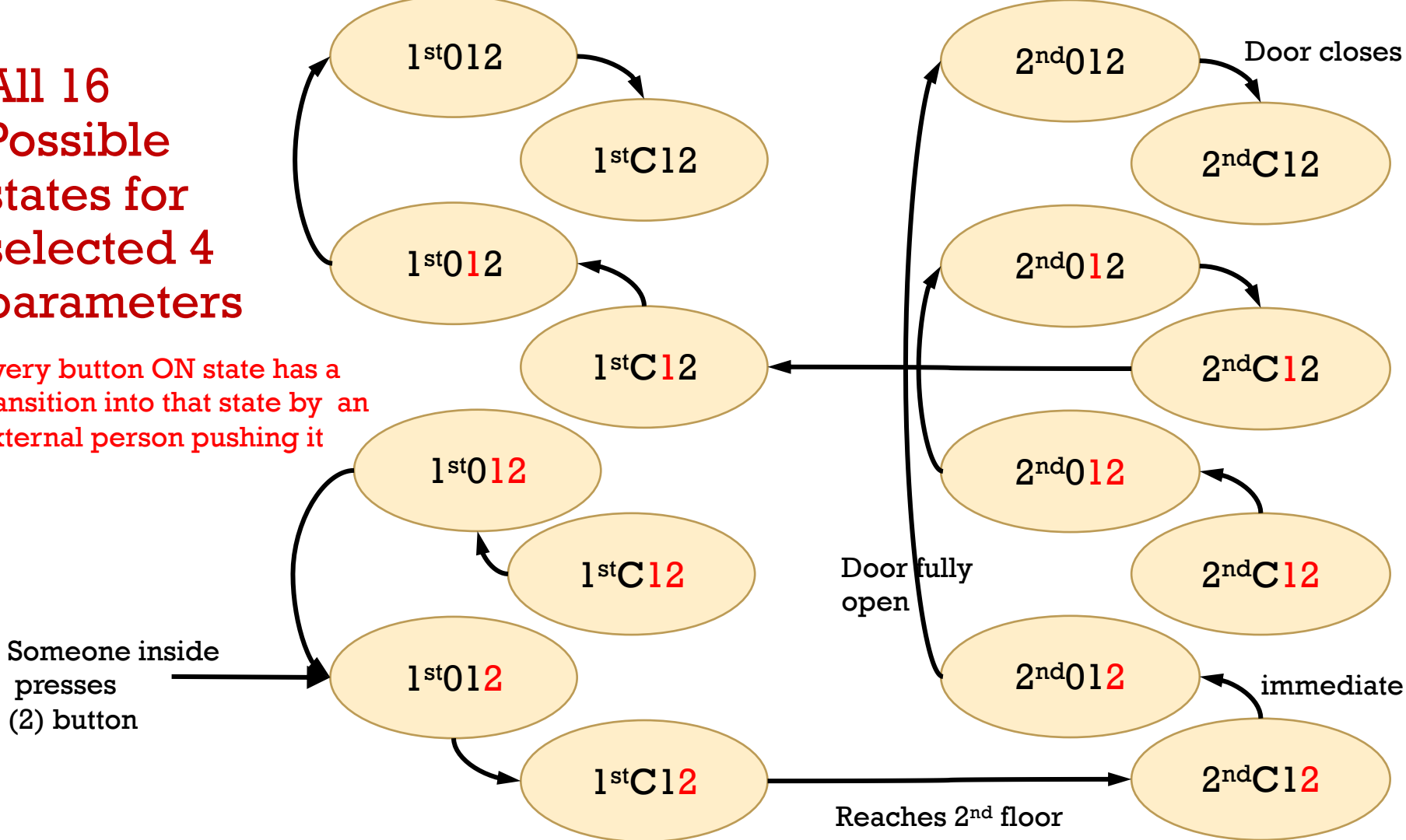
Ignoring for now

• 1stF call button (going up)	**U** on	U off
• 2ndF button (going down)	**D** on	D off



All 16 Possible states for selected 4 parameters

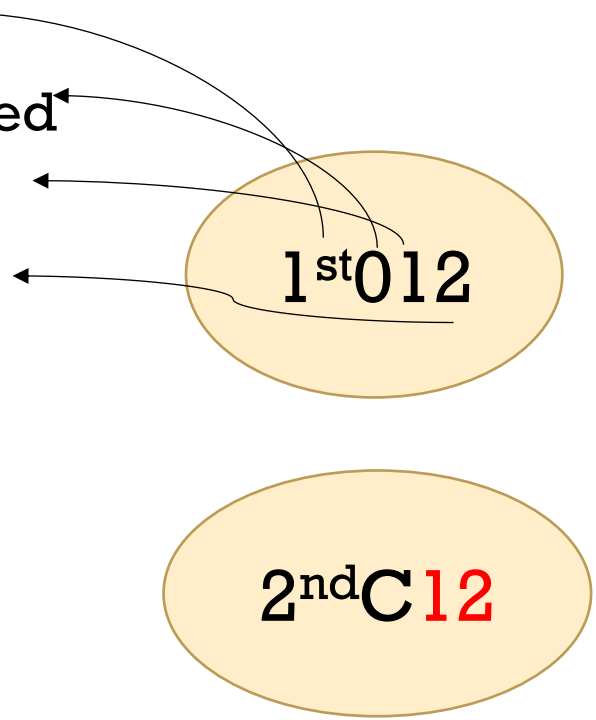
Every button ON state has a transition into that state by an external person pushing it

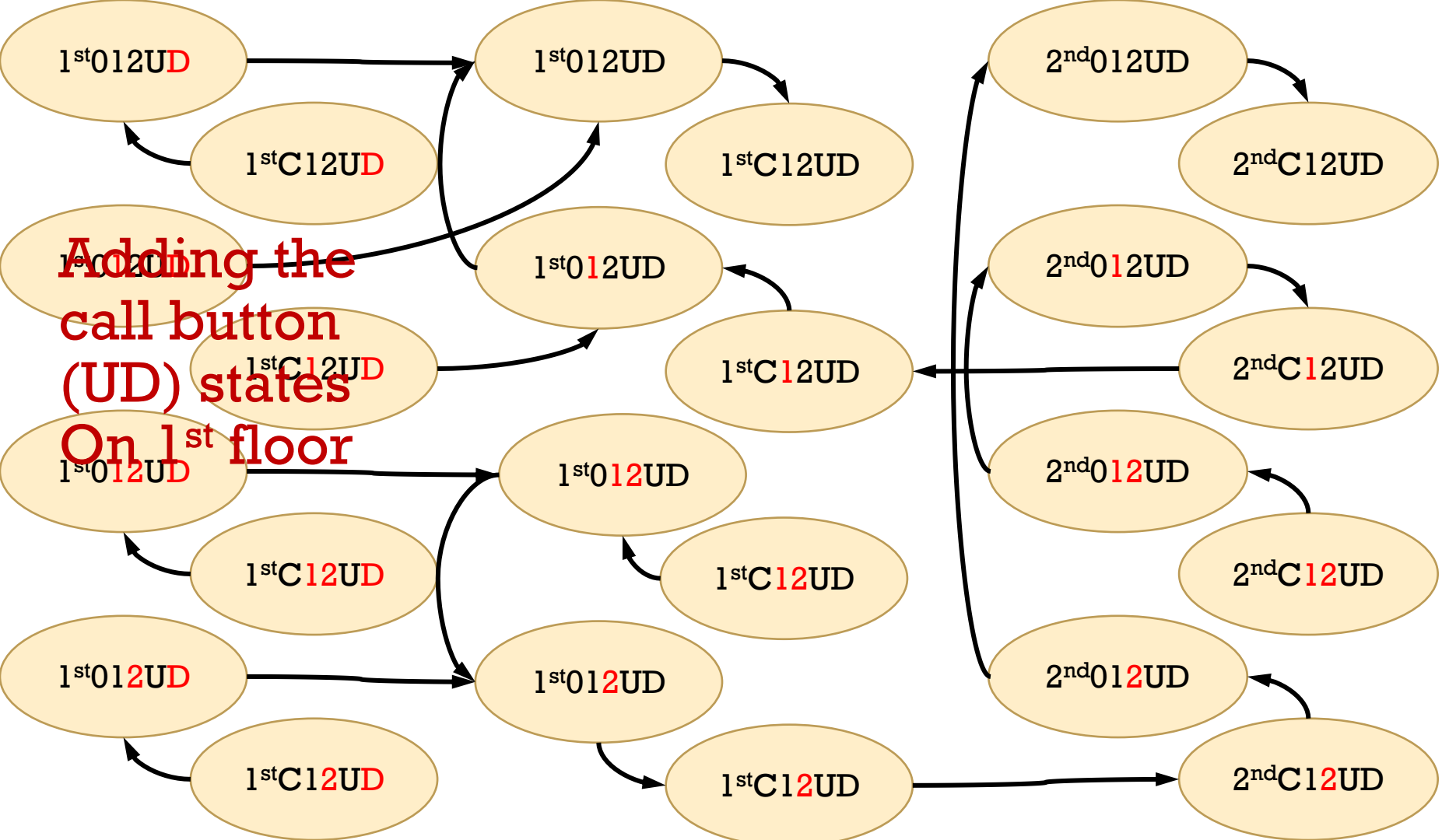


Adding Up Down Elevator States

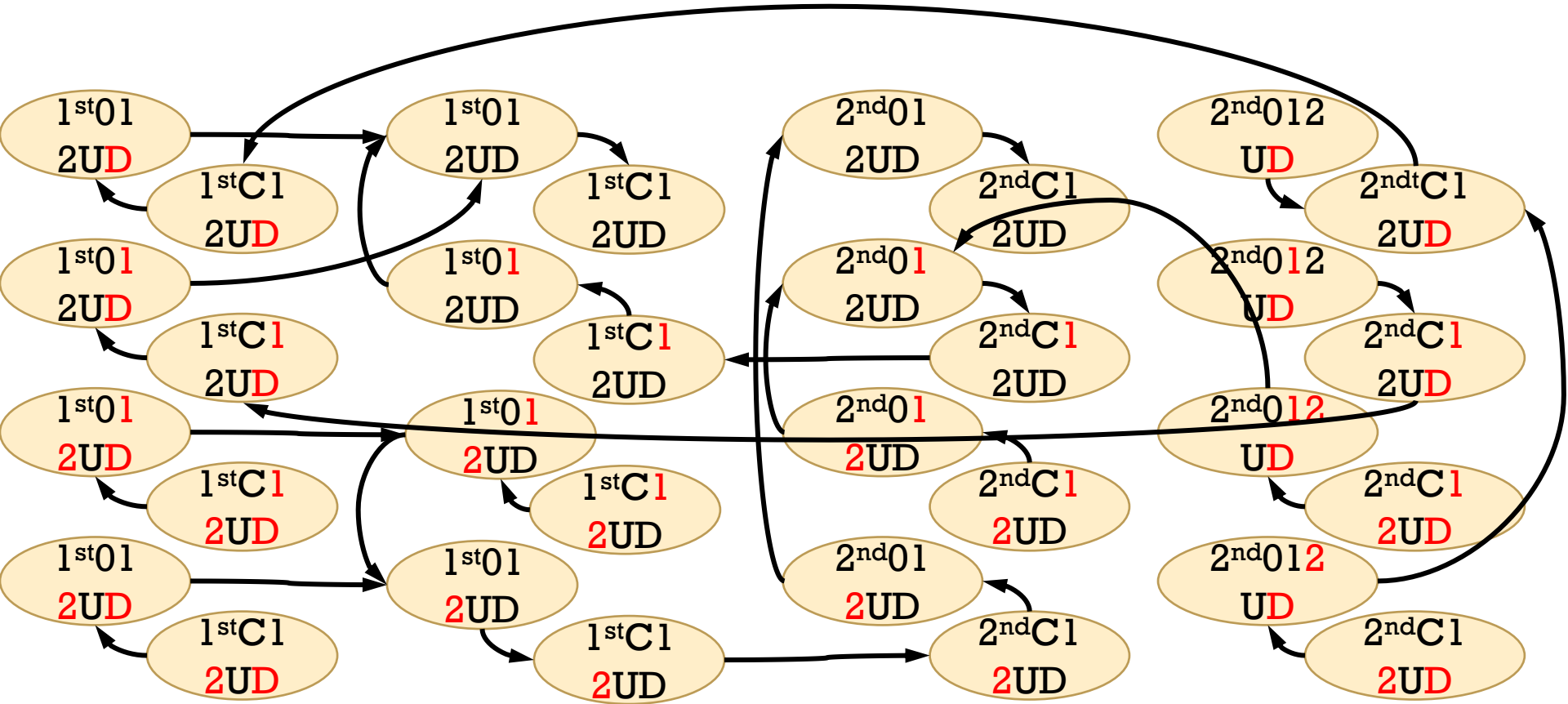
More States?

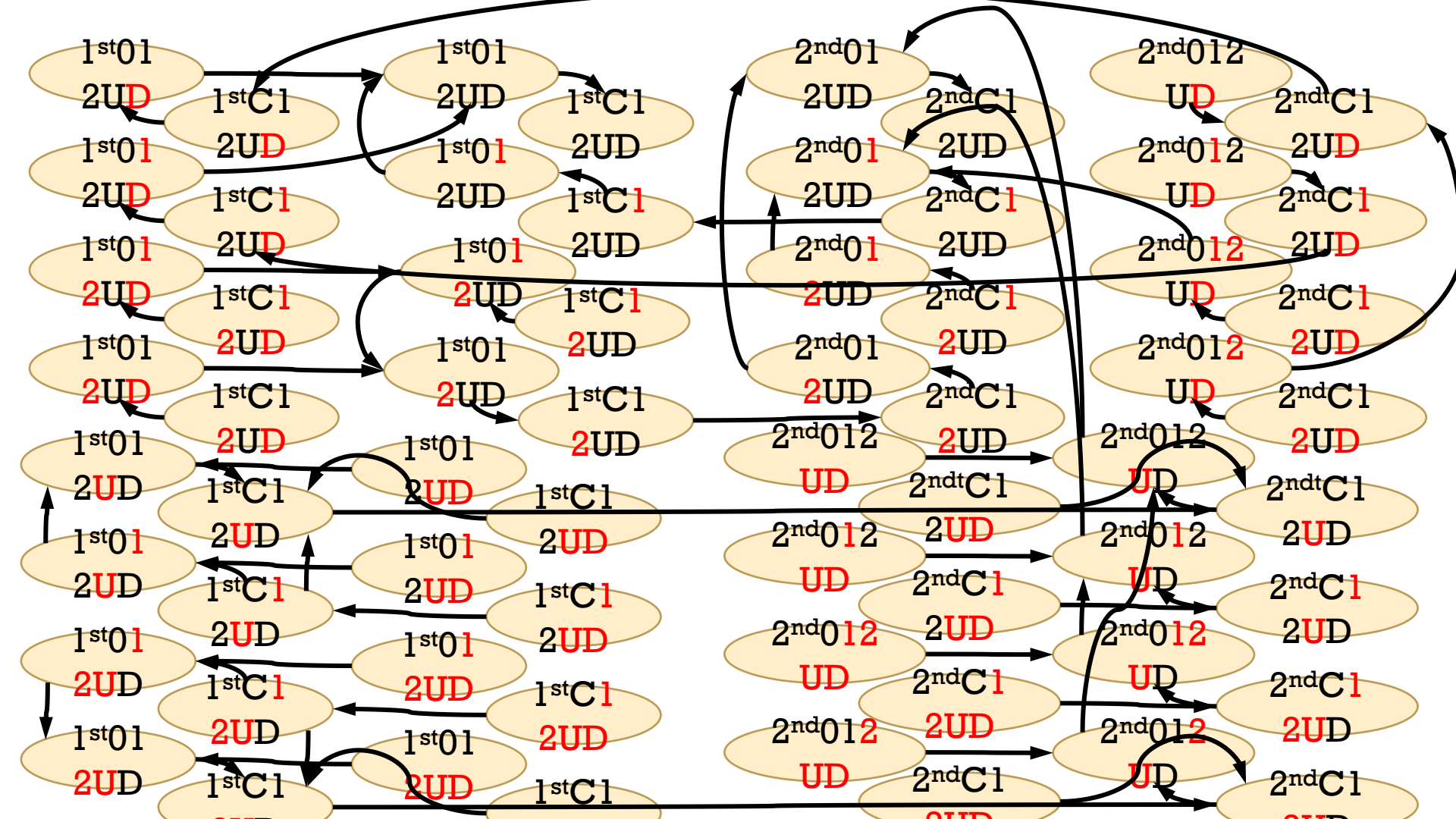
- On which floor — 1st — 2nd
- Door state — Opened — Closed
- Floor 1 button — **1** on — 1 off
- Floor 2 button on — **2** on — 2 off
- 1stF call button (going up) — **U** on — U off
- 2ndF button (going down) — **D** on — D off

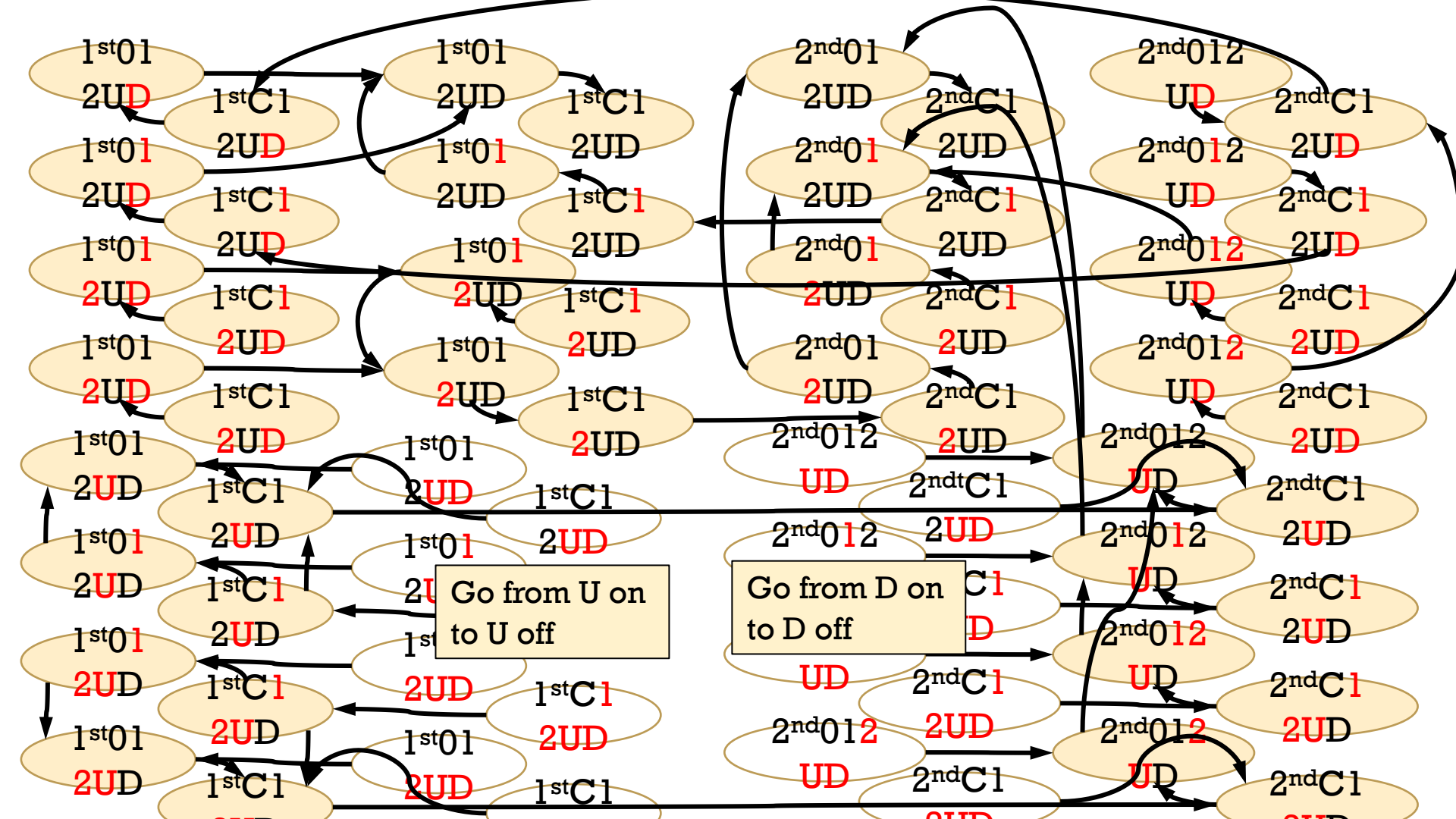


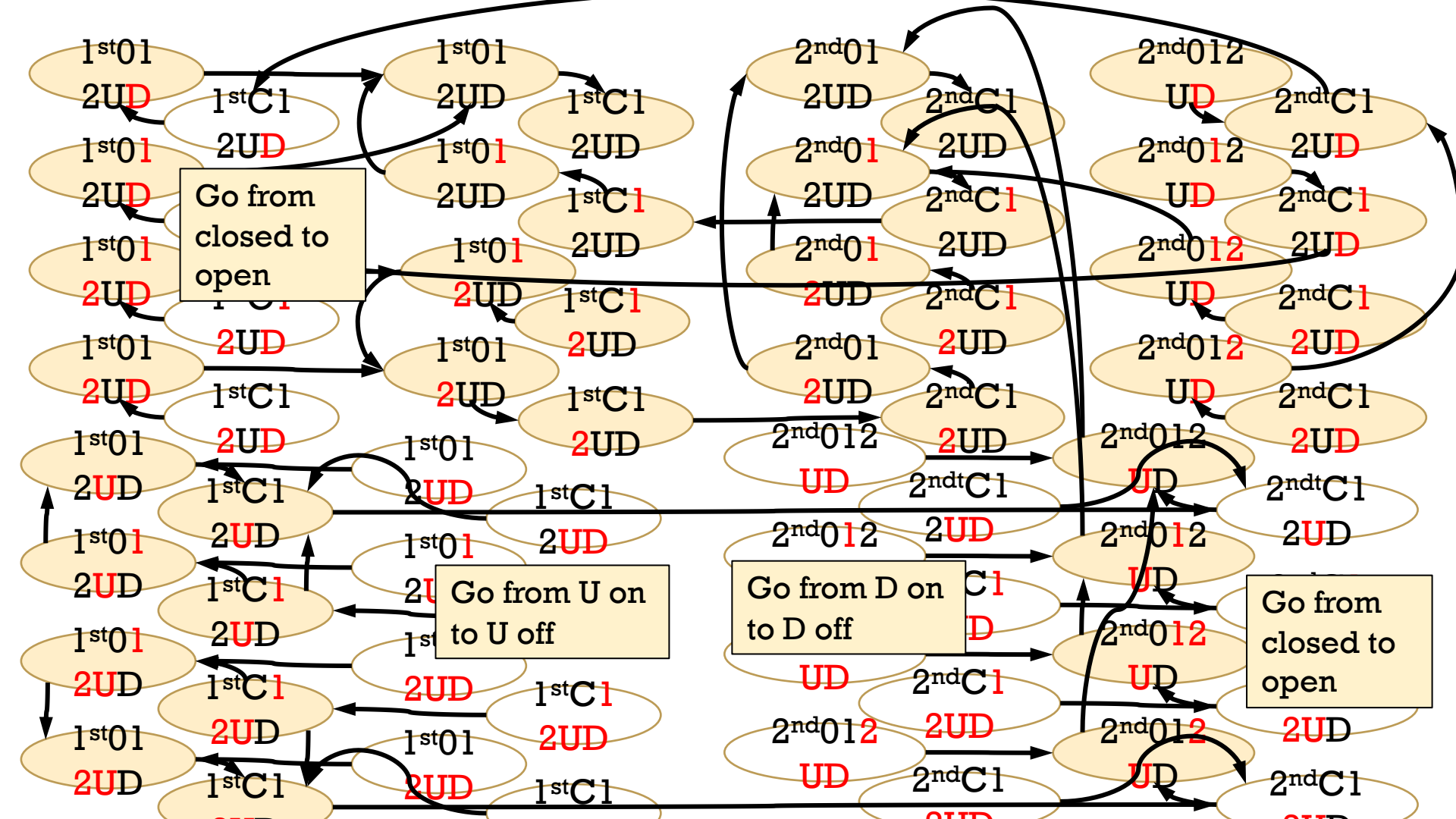

Adding the call button (UD) states On 1st floor
1st012UD
1st012UD
2nd012UD
1stC12UD
1stC12UD
2ndC12UD
1st012UD
1st012UD
2nd012UD
1stC12UD
1stC12UD
2ndC12UD
1st012UD
1st012UD
2nd012UD
1stC12UD
1stC12UD
2ndC12UD
1st012UD
1st012UD
2nd012UD
1stC12UD
1stC12UD
2ndC12UD

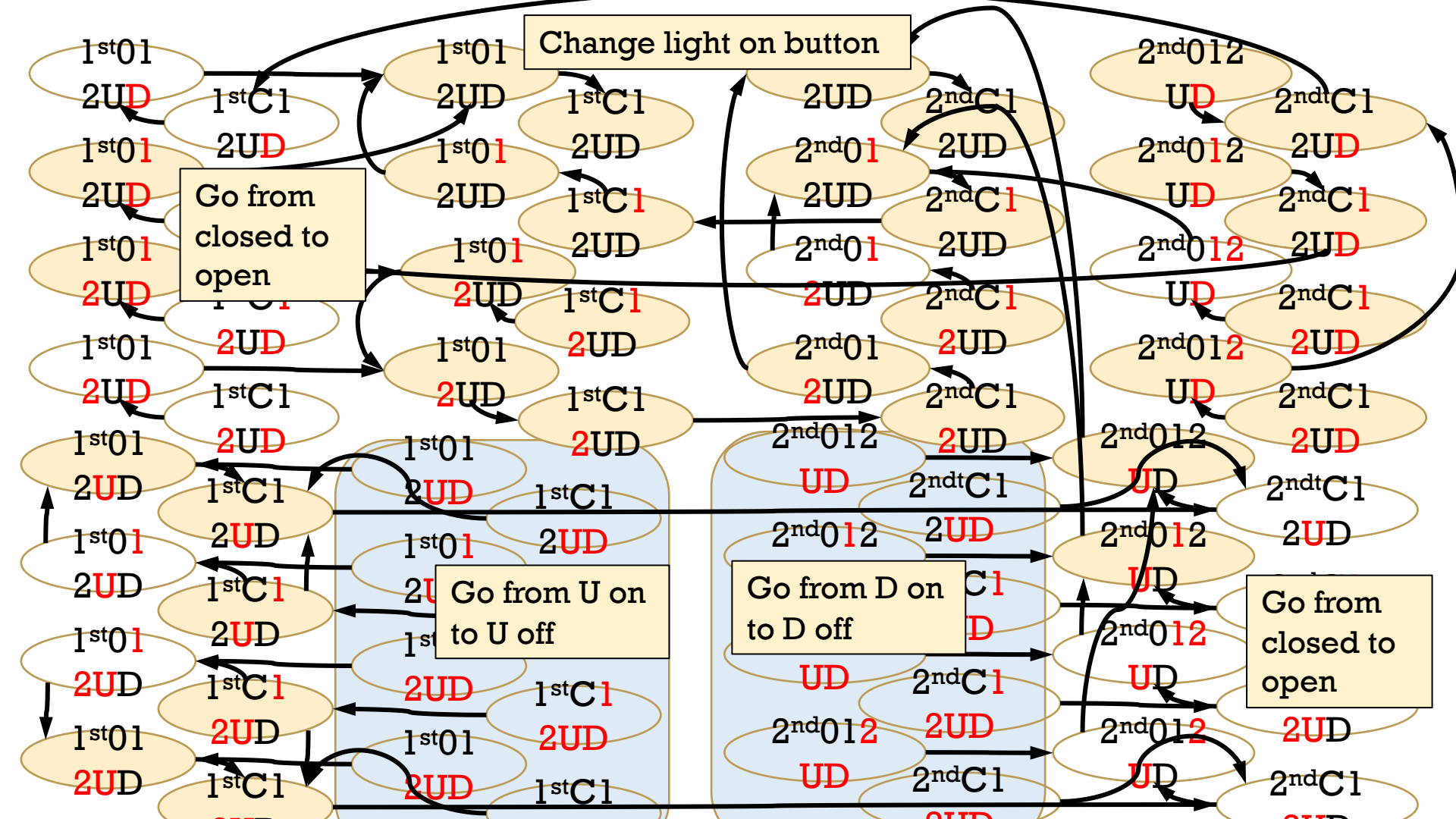
Adding Down Call Button On (adds 16)




1st01 2UD
1stC1 2UD
2nd01 2UD
2ndC1 2UD
2nd012 UD
2ndtC1 2UD


1st01
2UD
1stC1
2UD
2nd01
2UD
2ndC1
2UD
2nd012
UD
2ndtC1
2UD
Go from U on
to U off
Go from D on
to D off


Go from closed to open
Go from U on to U off
Go from D on to D off
Go from closed to open


Change light on button
Go from closed to open
Go from U on to U off
Go from D on to D off
Go from closed to open

Resources

- FSM using C code example (Dr.Dobbs article) on canvas
  - Files->Resources-> FSM-samek DrDobbs.pdf

- Google ”finite state machine”
- Electrical engineering aspects: FSM with output (Mealy, Moore machines) -> creating FSM circuits
- Computer science aspects: properties of computer languages, grammars, etc.
- Automated FSM code generators.

Summary

- Arrays are like pointers (variables that hold addresses)
- Pointers are used to at the lower level to access registers and often for more complex functions. For MEAM510, most functions can be achieved without pointers.
- Events and Services framework provides a structure for asynchronous, concurrent tasks in embedded systems. It makes writing larger complex of systems easier. It is highly recommended for when you write your final project code.
- FSM's can help to visually organize program flow