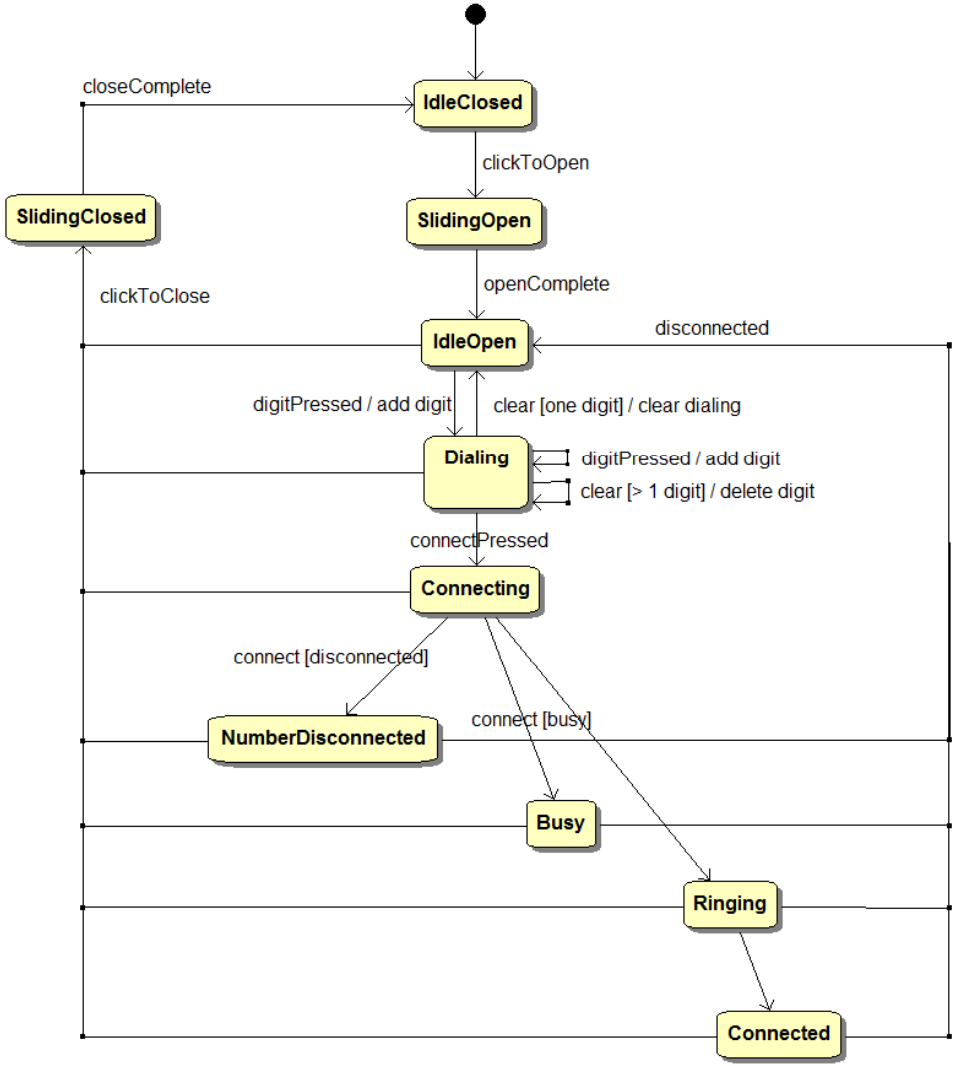
States and State Machines

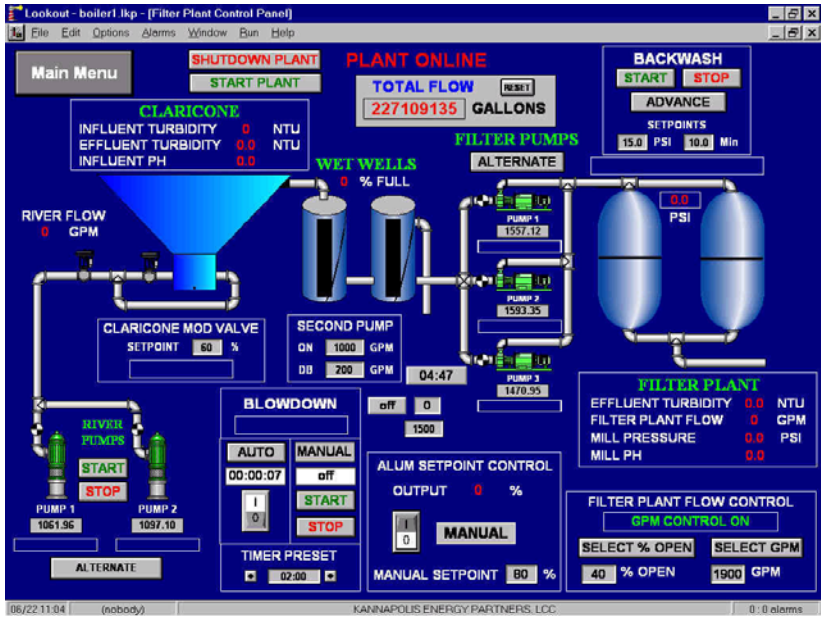


What is this for?

State machines are commonly used in…

Embedded Systems

Factory/Process Controls



State

State – An abstraction of the current status of a system.
States are assigned names.

Waiting for a Keypress
Waiting for Elvis
Raising Firearm to Fire
Cellphone is Dialing
Door Opening

Verbs with “ing”

Paper Jammed
Battery is Below Limit
Power is On
Door Open
Prius Accelerator Stuck

Statement of condition

States in a Garage Door

Are there any more states?

DoorClosed

DoorOpen

More States

DoorOpening

DoorClosing

How we will express this in a program

```
/* Our possible garage door states */
#define DoorClosed 1
#define DoorOpening 2
#define DoorOpen 3
#define DoorClosing 4
```

Above *main* in our program

```
int main()
{
   int state = DoorClosed;
   …
```

In the *main* function

Why do we care? We do different things depending on the current state. What does the button do in each state?

Naming Conventions - States

We will usually name states with camel-case and a capital first letter. We'll use #defines in our programs for state names.

WaitingForKeypress
WaitingForElvis
RaisingFirearm
CellphoneDialing
DoorOpening

Verbs with "ing"

PaperJammed
BatteryBelowLimit
PowerOn
DoorOpen
PriusAccelStuck

Statement of condition

Events

An event is an *occurrence in time*. It is considered atomic and instantaneous.

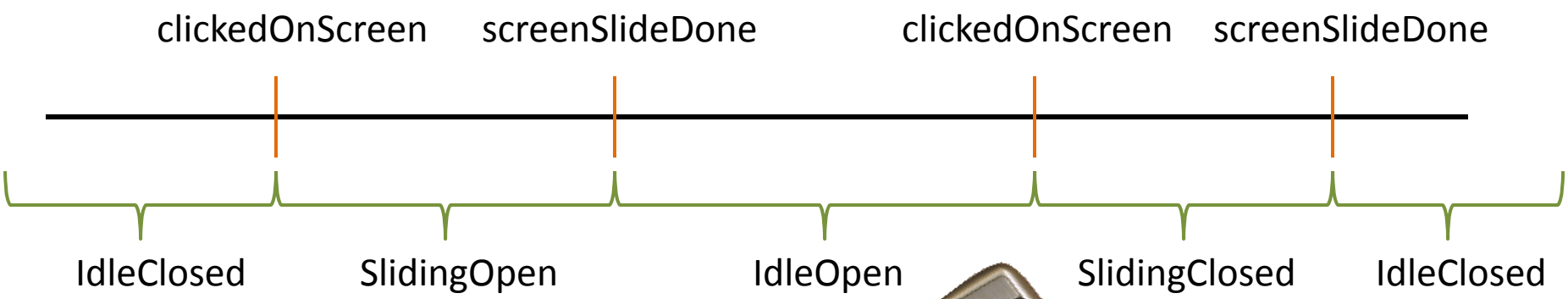
Left mouse button pressed
Key pressed
Elvis has left the building
Flight 529 has landed
Power turned on

Past tense verbs

Paper is jammed
Message has timed out
10 seconds has elapsed
Battery is below limit
Project 2 is due

Onset of condition

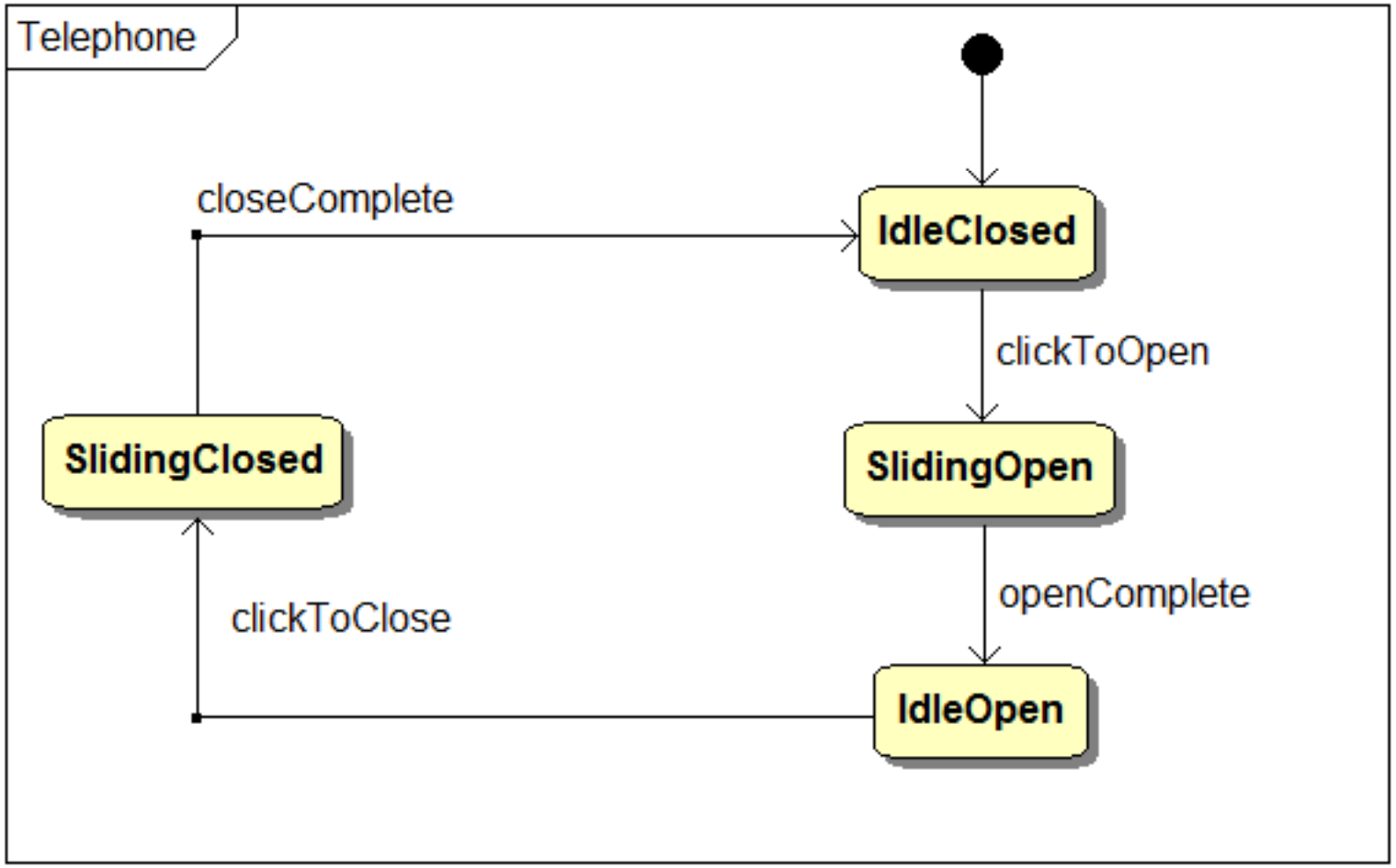
Events vs. State



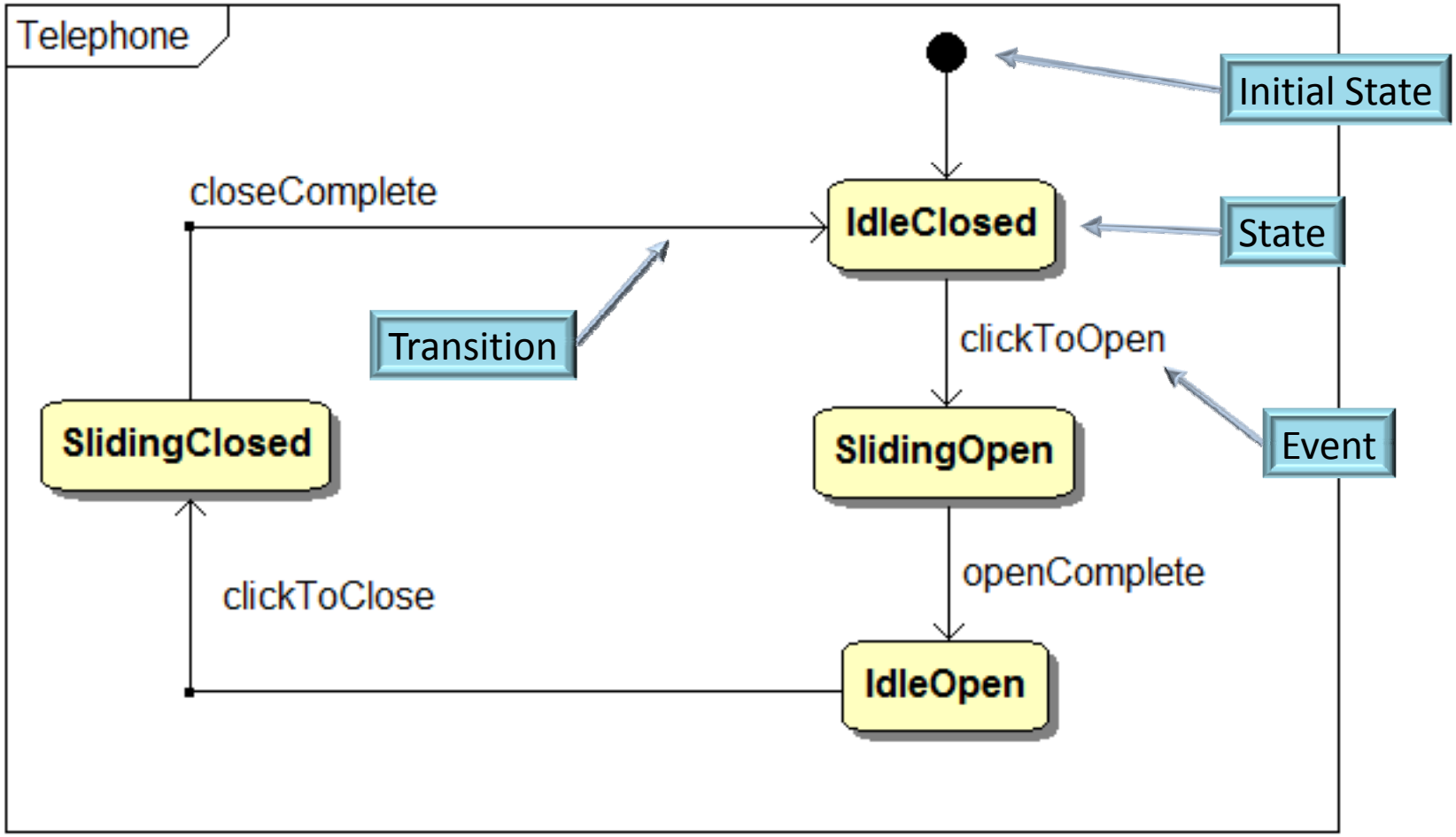
An event is what causes us to change from state to state.

State Diagrams

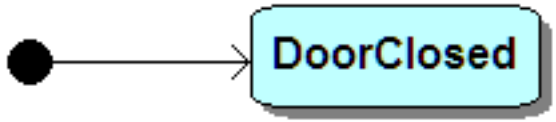
State diagrams describe what we will implement in code as a *state machine*.



State Diagrams



Starting out State Machine




```
int main()
{
   int state = DoorClosed;
   …
```

```
int main()
{
   int state = DoorClosed;

   printf("Garage Startup\n");
   GarageStartup();

   while(IsGarageRunning())
   {

   }

   printf("Garage Shutdown\n");
   GarageShutdown();
   return 0;
}
```

A Control Loop

A continuous loop in a controls application that controls the system. Right now our loop is doing nothing. We'll want to add code to make our garage door work.

Important Idea

We do something different depending on the state we are in. It makes sense to create a *function* for each state.

```
void StateDoorClosed(int *state)
{
}
```

Note the pointer. We pass the state by reference. It is an in/out parameter

What should happen when we are in the DoorClosed state?

DoorClosed state…

If the button is pressed:

    Start the motor

    Go to the DoorOpening state

otherwise:

    Do nothing…

DoorClosed state…

If the button is pressed:
    Start the motor
    Go to the DoorOpening state
otherwise:
    Do nothing…

```
void StateDoorClosed(int *state)
{
   if(WasButtonPressed())
   {
      SetMotorPower(1);
      *state = DoorOpening;
   }
}
```

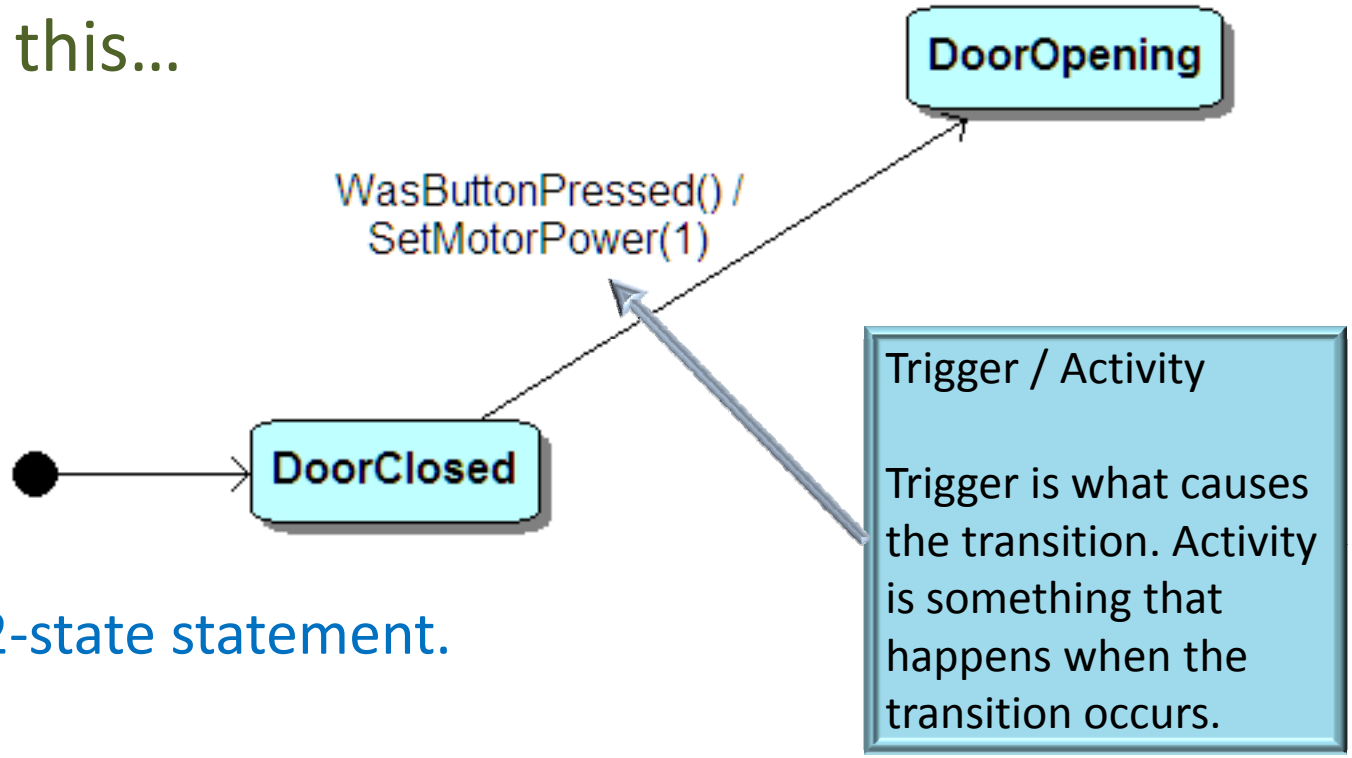
The Control Loop – Handling States

```
while(IsGarageRunning())
{
   switch(state)
   {
   case DoorClosed:
      StateDoorClosed(&state);
      break;
   }

}
```

We will put a switch statement in our control loop to handle the states.

We now have this…



This is a simple 2-state statement.

What happens

```
while(IsGarageRunning())
{
   switch(state)
   {
   case DoorClosed:
      StateDoorClosed(&state);
      break;
   }
}
```

Control Loop

```
void StateDoorClosed(int *state)
{
   if(WasButtonPressed())
   {
      SetMotorPower(1);
      *state = DoorOpening;
   }
}
```

State Function

The control loop runs continuously (1000 times per second in this program). Each time it calls a function for the current state. That state function decides if we need to change the state and does anything else we need to do while in that state.

A Garage Door Opener