Week 1 - Friday

COMP 3400

Last time

- What did we talk about last time?
- Fixed width types
- Systems
- Course themes
- System architectures

Questions?

Assignment 1

Project 1

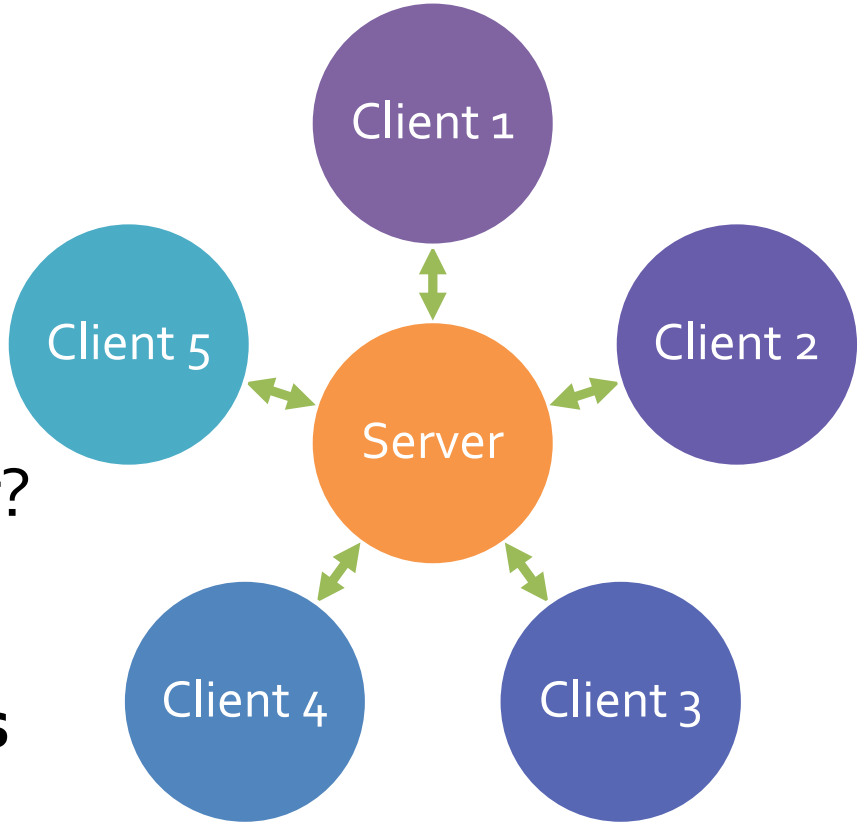
System Architectures

System architectures

- System architectures are models of systems that describe:
  - Relationships between entities in the system
  - Ways the entities communicate
- Different architectural styles have pros and cons
- Using a certain style can have big impacts on system performance
- Common styles:
  - Client/server
  - Peer-to-peer (P2P)
  - Layered
  - Pipe-and-filter
  - Event-driven
  - Hybrid

Client/server architectures

- This book considers **client/server architectures** from the perspective of a many clients connecting to a single server
  - If you recall, the Software Engineering book describes client/server as a system with many servers, each of which offer a single service
- How does a client know how to reach the server?
  - Uniform resource identifier (URI) is a common way: www.goats.net/image.jpg
- Client/server architectures depend on **protocols** to define how clients can request services and understand the response



Client/server advantages and disadvantages

ADVANTAGES

- Updates are simple, because only the server needs to be updated
- Only the server needs to be checked for security problems or data corruption

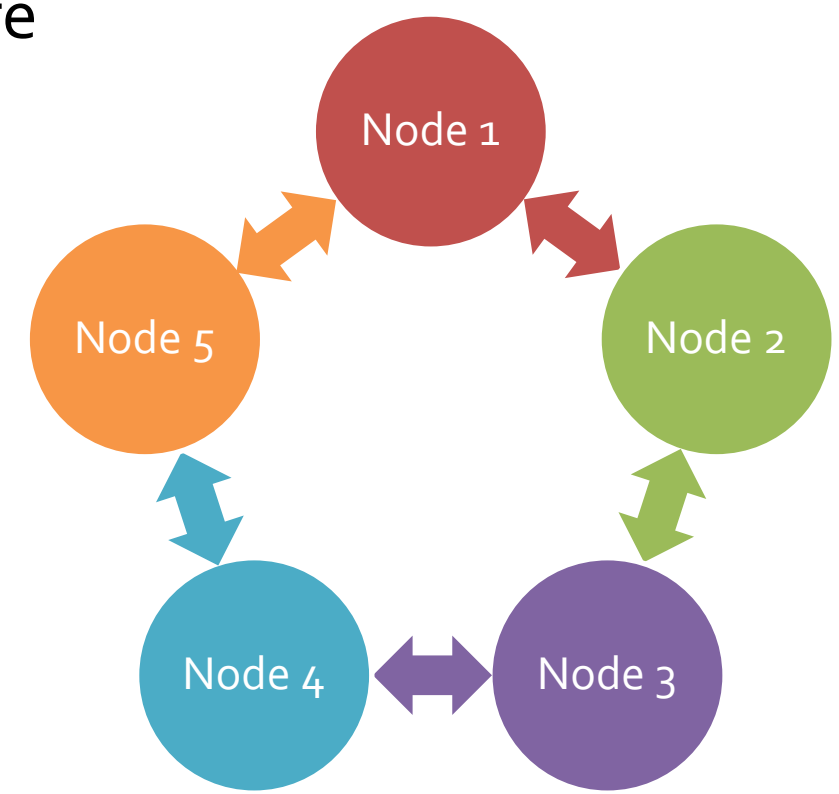
DISADVANTAGES

- Single point of failure

- To reduce the single point of failure problem, it's common to have multiple servers that offer the same services or files
- To work, these servers must coordinate with each other when one is updated

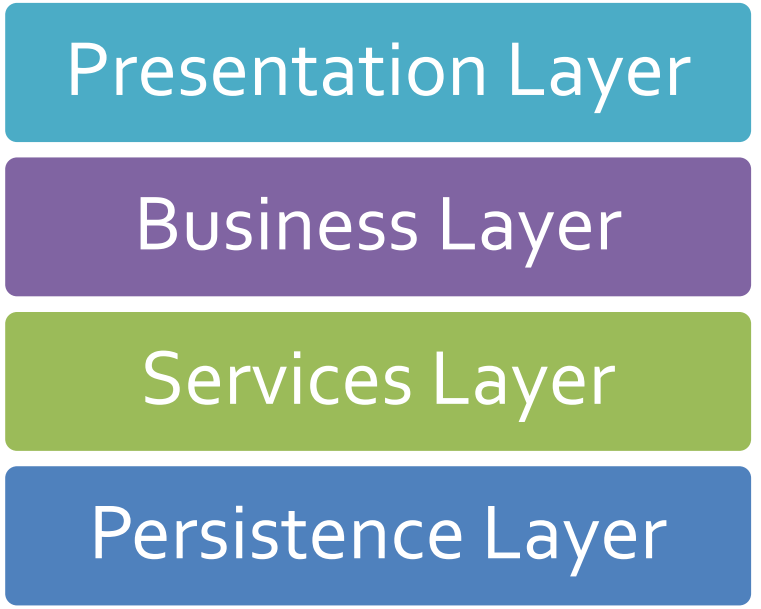
Peer-to-peer (P2P) architectures

- If more and more servers are used, the architecture begins to look like a **P2P architecture**
  - BitTorrent
  - DNS
- In P2P, there is usually no distinction between clients and servers, since most entities act as both
- Advantages:
  - Service scales, staying the same or improving as the number of users goes up
- Disadvantages:
  - Security: A corrupted node can be hard to detect
  - Administration: Propagating changes can be difficult



Layered architectures

- **Layered architectures** divide systems into a strict hierarchy of components
- Each layer can only communicate with the layer above and below it
- Advantages:
  - As long as a new layer knows how to talk to the layer above and below, it can be swapped out with an old layer
  - New layers can be added on top
- Disadvantages:
  - It's hard to divide systems into hierarchical layers
  - It can be inefficient to prevent one layer from talking directly to one much lower or higher
  - Some services at each layer are redundant



Pipe-and-filter architectures

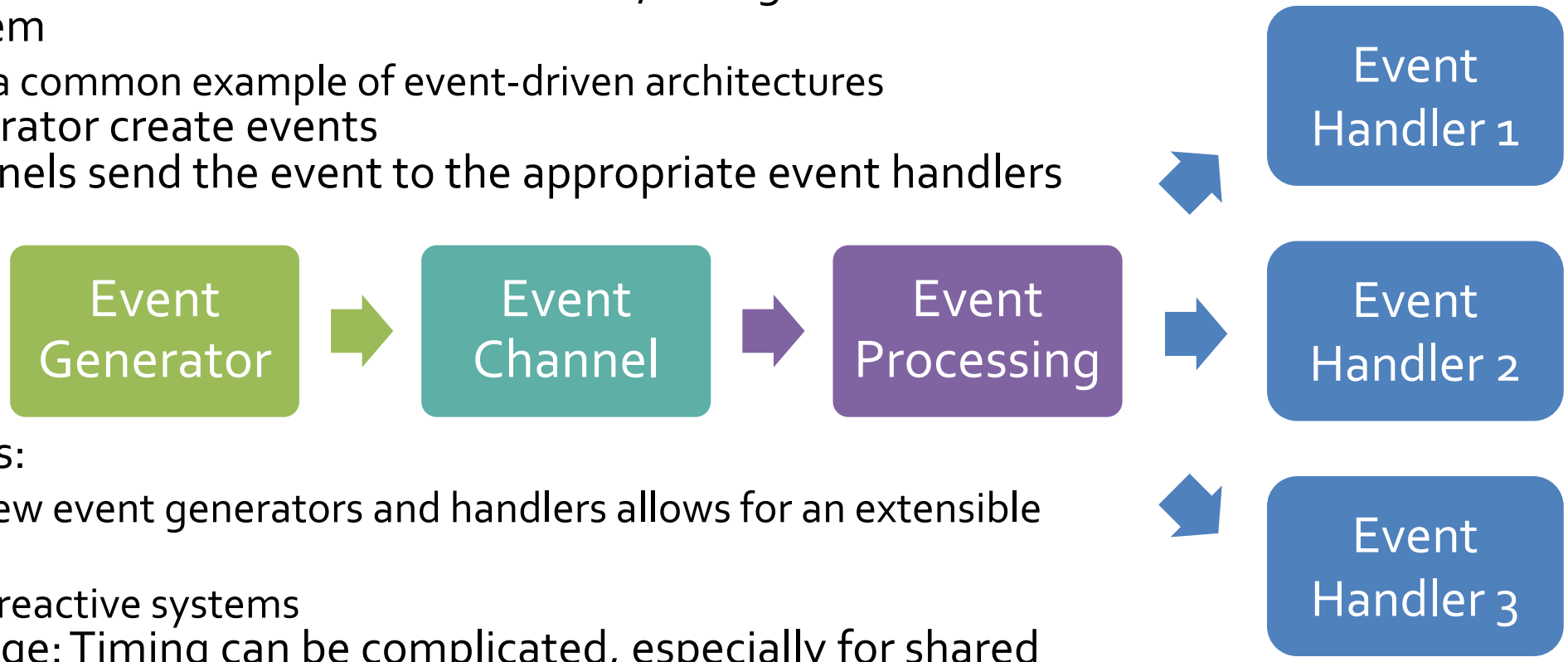
- **Pipe-and-filter architectures** send data in one direction through a series of components
- The output of one stage is the input of the next
- Each stage transforms the data in some way
- Examples:
  - Linux command-line piping

```
sort foo.txt | grep -i error | head -n 10 > out.txt
```

  - Java stream filtering
  - Stages of a compiler
- Advantages:
  - Good for serial data processing
  - Modular components that have the same input and output can be reused in different sequences
- Disadvantage: No error recovery if something breaks in the middle

Event-driven architectures

- **Event-driven architectures** react to events, changes in the state of the system
  - GUIs are a common example of event-driven architectures
- Event generator create events
- Event channels send the event to the appropriate event handlers



- Advantages:
  - Adding new event generators and handlers allows for an extensible system
  - Good for reactive systems
- Disadvantage: Timing can be complicated, especially for shared resources

Hybrid architectures

- We talk about the previous architectures because they're models that have been successful in the past
- Most real systems are a mix of different architectures
  - The whole system could be one architecture, but its components have their own
  - A system could be mostly one architecture but break a couple of rules
  - There can be different ways of looking at the same system
- Example: OS kernel
  - Event-driven because it has interrupt handlers to respond to signals from the hardware
  - Client/server because applications that make system calls are making requests
  - Layered because file systems and networking operate with layers from the generic operation down to the requirements of particular hardware

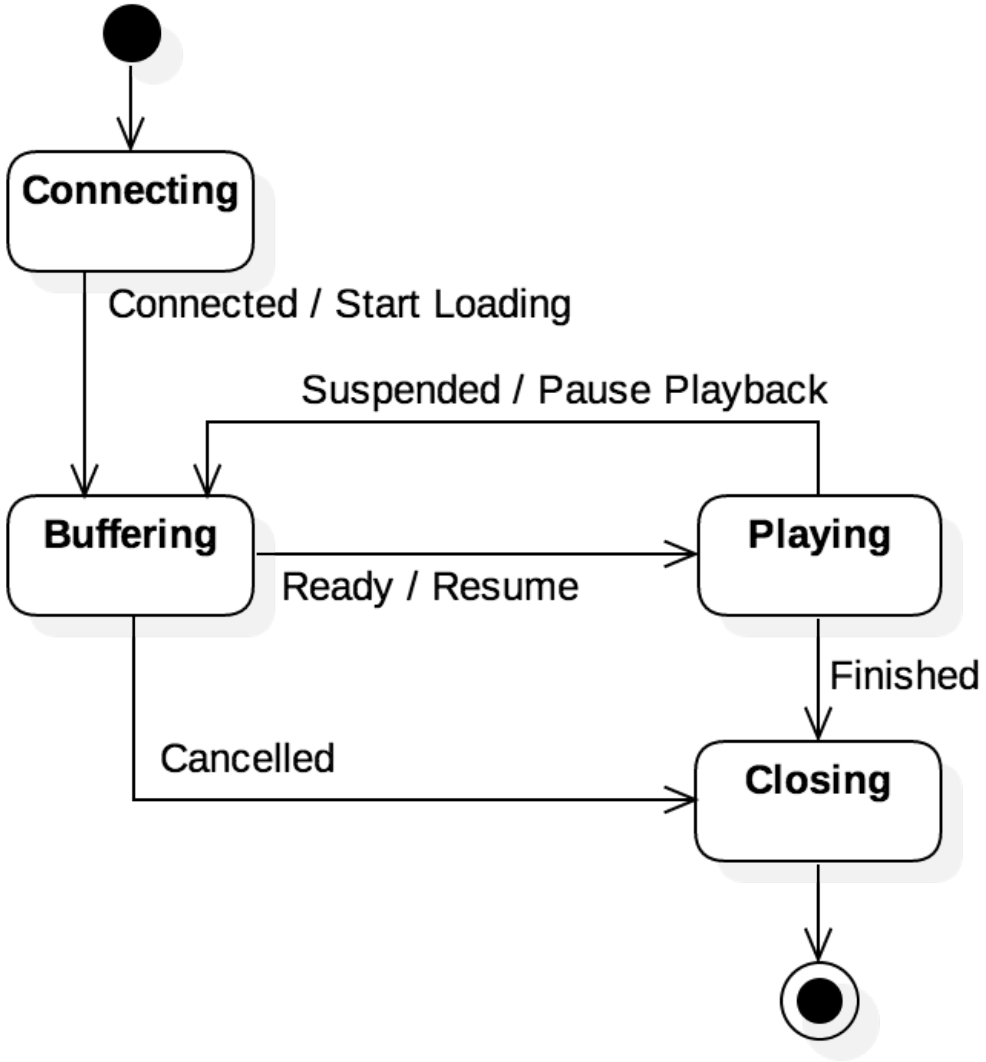
State Machines

States

- **States** are different, meaningful configurations that a system can be in
- They show up all over computer science
  - Deterministic finite automata (DFA) are a simple model of computation equivalent to regular expressions
  - State machines are useful for parsing
  - State machines can express different states that a system (like a microwave or a enemy in a video game) can be in
- **Transitions** are changes from one state to another
- Events trigger transitions which then have **effects**, visible behavior

UML state models

- As discussed in COMP 3100, UML standardizes **state models** as a way to visualize states and transitions
  - States are shown as rounded rectangles
  - A solid circle shows the initial state
  - A solid circle in a circle shows the final state
  - Transitions are shown as labeled arrows
  - Effects (if any) are written after a slash after the transition label

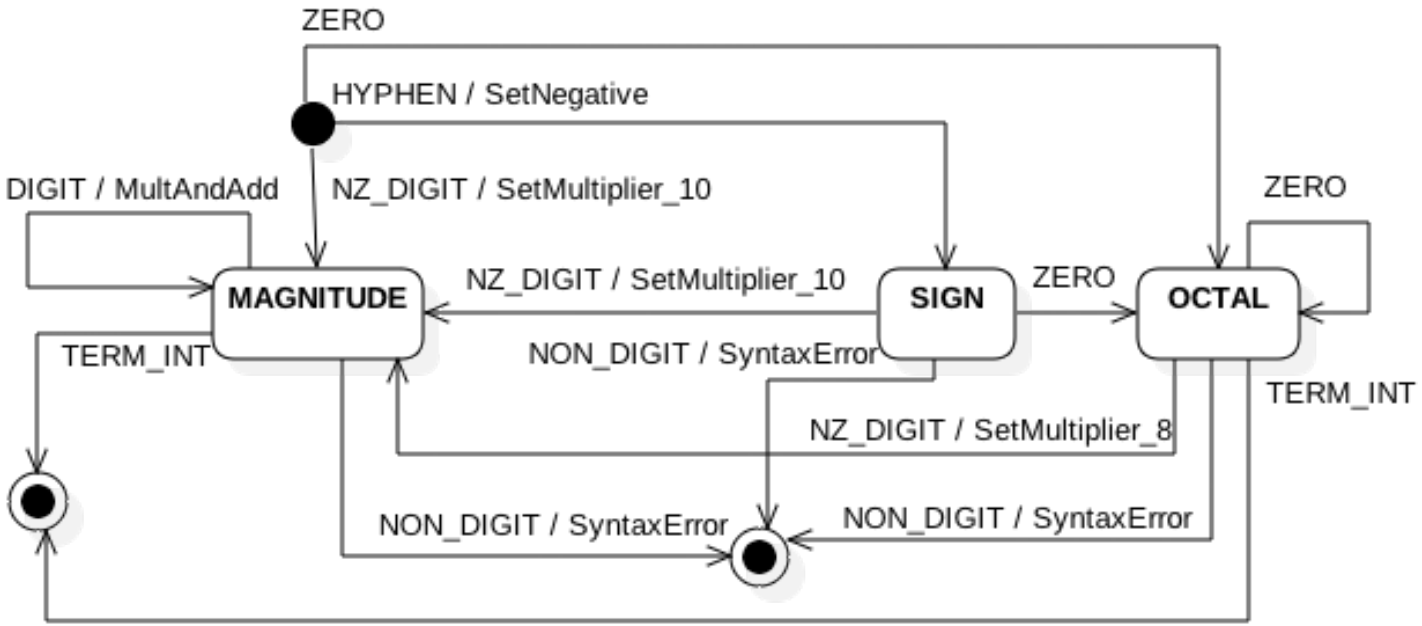


State space explosion

- When constructing state machines from some other representation, it's possible to generate a large number of states
- For example, a different state for every configuration of bits in a 32-bit integer would mean $2^{32} \approx 4$ billion different states
- To be useful as models, there needs to be a reasonable number of states
  - Different states can be grouped together based on what's meaningful for a given problem

State machines as recognizers

- State machines are often used to recognize strings as being legal or illegal
- Consider a state machine from Project 1 designed to recognize integer values (formatted in either decimal or octal)
- In addition to recognizing integers as legal or illegal, the machine builds the integer based on the effects



Regular expressions and state machines

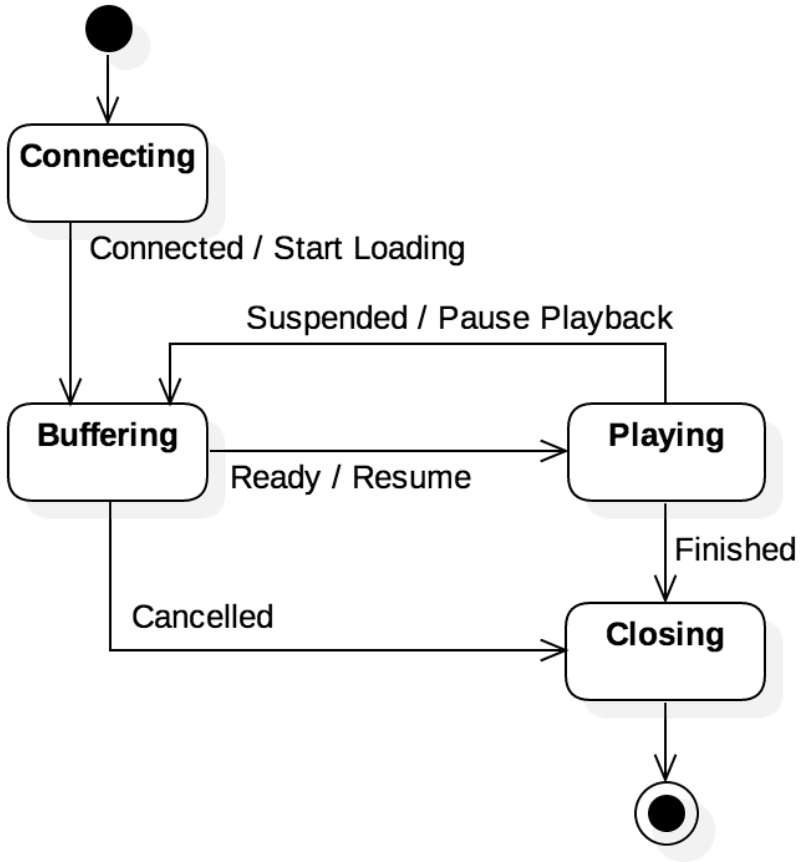
- As COMP 3200 covers, regular expressions and finite state machines are equivalent
  - For every regular expression, there's an equivalent FSM
  - For every FSM, there's an equivalent regular expression
- Example:
  - Regular expression: 1 0 1*
  - FSM:

Implementing state machines

- There are algorithms to convert between regular expressions and state machines
- Most regular expression libraries build a state machine as a way to see if strings match the regular expression
- One way to implement state machines is with a 2D array
  - One row for every state
  - One column for every event, saying which state a given state will transition to
- If there are effects, a second 2D array can show which effects happen on those transitions
- If an action happens whenever a state is entered, a 1D array can hold that information

Example transition table

- The state model on the left has a transition table on the right



	Events				
States	Connect	Suspend	Ready	Finish	Cancel
Connecting	Buffering				
Buffering			Playing		Closing
Playing		Buffering		Closing	
Closing					

Example table in code

- Two **enum**s are used to list the states and the events
- A 2D array stores the transitions

```
typedef enum { CONN, BUFF, PLAY, CLOS, NST } ms_t;
typedef enum { Connect, Suspend, Ready, Finish, Cancel } event_t;
#define NUM_STATES (NST+1)
#define NUM_EVENTS (Cancel+1)
static ms_t const _transition[NUM_STATES][NUM_EVENTS] =
{
  // Connect Suspend Ready   Finish Cancel
  {  BUFF,   NST,    NST,    NST,   NST  }, // Connecting
  {  NST,    NST,    PLAY,   NST,   CLOS }, // Buffering
  {  NST,    BUFF,   NST,    CLOS,  NST  }, // Playing
  {  NST,    NST,    NST,    NST,   NST  }  // Closing
};
```

Effects

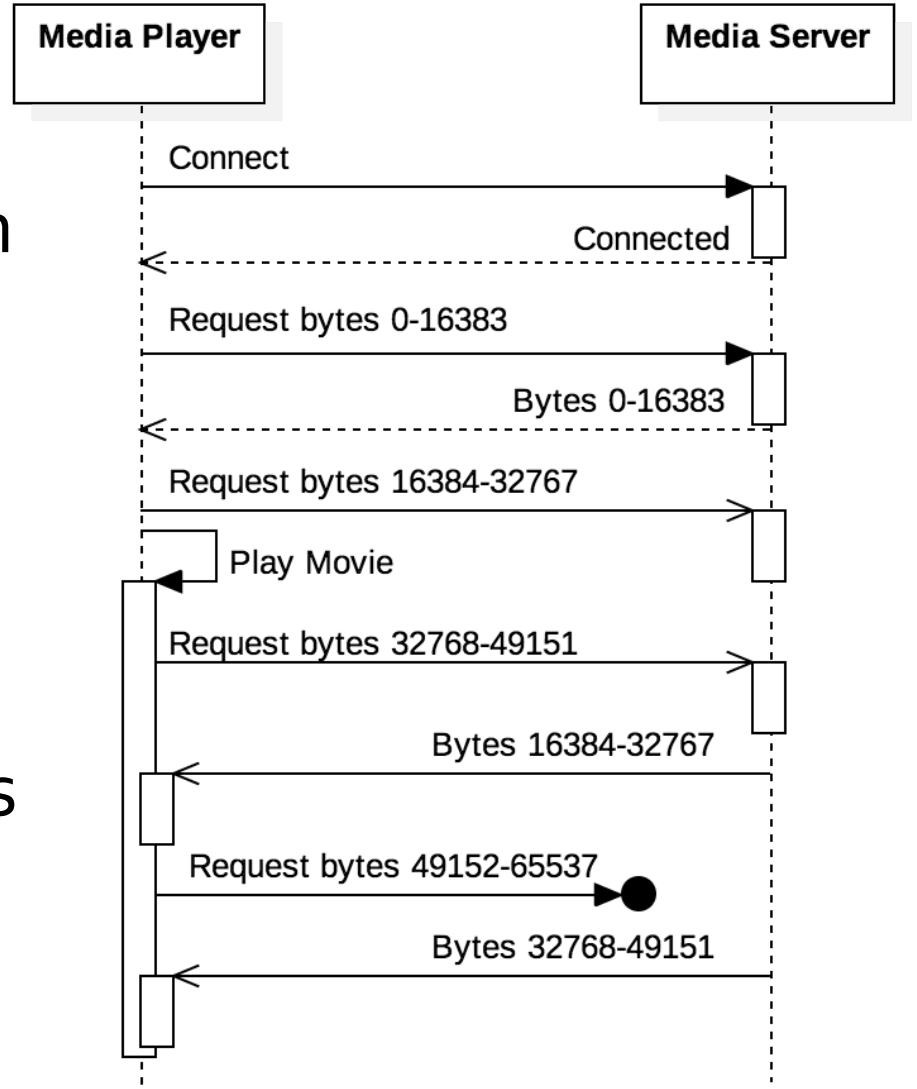
- A table filled with function pointers can be used for effects

```
static action_t const _effect[NUM_STATES][NUM_EVENTS] = {
  // Connect      Suspend      Ready    Finish Cancel
  {  start_load, NULL,        NULL,    NULL,  NULL }, // Connecting
  {  NULL,       NULL,        resume,  NULL,  NULL }, // Buffering
  {  NULL,       pause_play,  NULL,    NULL,  NULL }, // Playing
  {  NULL,       NULL,        NULL,    NULL,  NULL }  // Closing
};
```

Sequence Models

Sequence models

- State models don't have any timing or sequence information
- **Sequence models** show the order in which messages are sent from one entity to another
  - Solid arrows show synchronous messages
  - Open arrows show asynchronous messages
  - Dotted lines show responses
  - Messages that end in circles are lost
- The order of messages in sequence models is logical, not scaled by time



Upcoming

Next time…

- Processes
- Multiprogramming
- Kernel

Reminders

- **No class on Monday**
- **Work on Assignment 1**
  - Due next Friday by midnight!
- Look over Project 1
- Read sections 2.1, 2.2, and 2.3