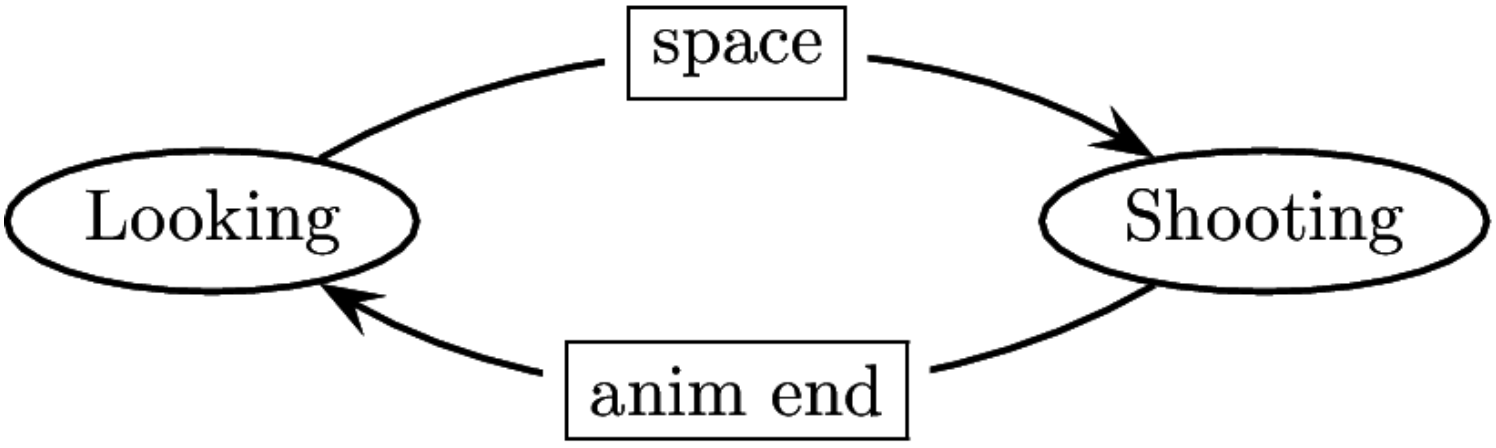
Finite State Machines

CSCI 321

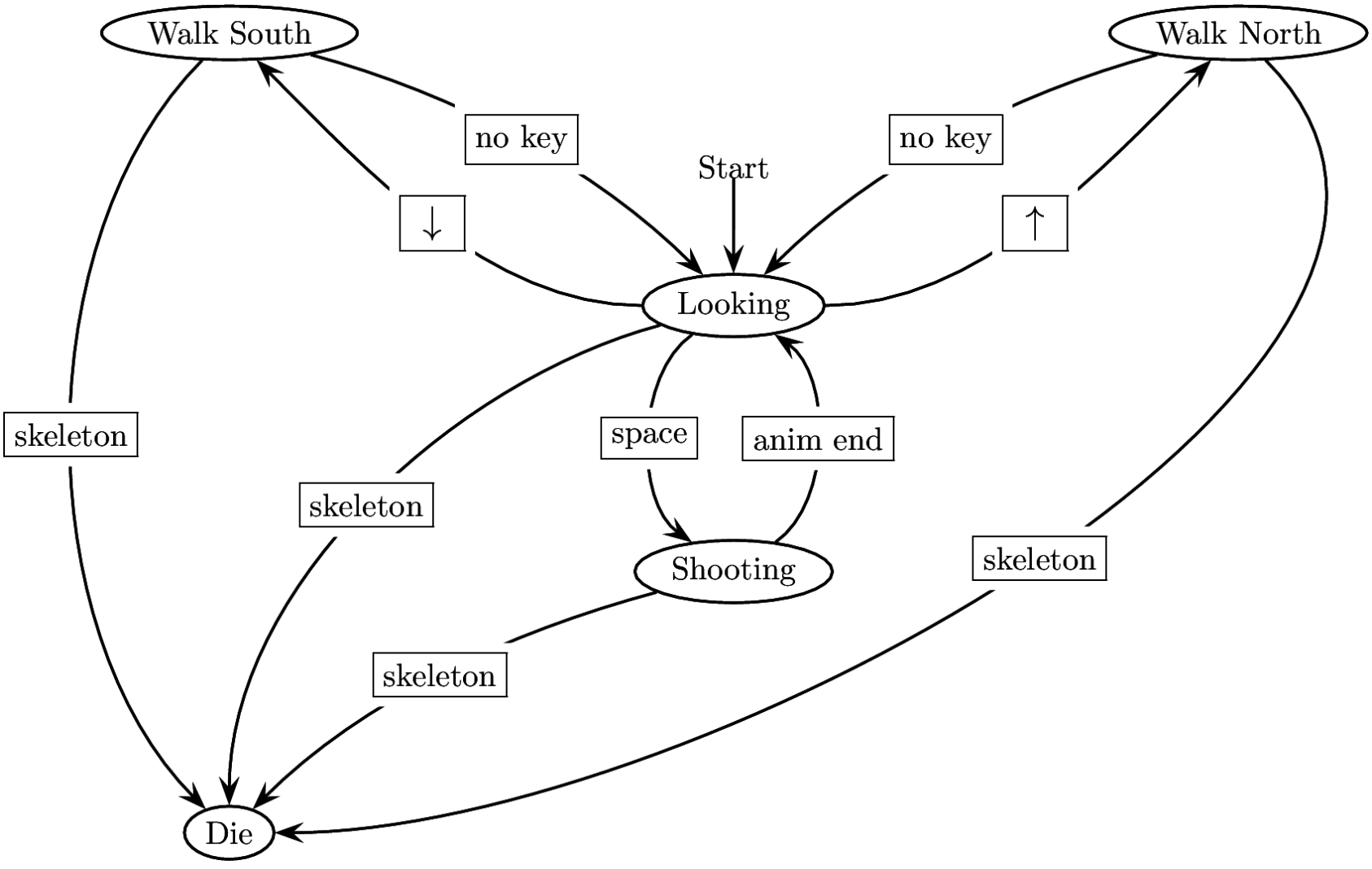
WWU

May 9, 2016

Simple State Machine



More Complex State Machine



Advantages of State Machines

- Quick and simple to code
- Easy to debug
- Little computational overhead
- Intuitive and easy to understand
- Flexible
  - Add new states
  - Add new rules
  - Add sub-machines and super-machines

FSM implementation 1.0: If-then statements

```
case CurrentState:
  EvadeEnemy:
    ...
    if Safe():
      ChangeState(Patrol)
    ...
  Patrol:
    ...
    if Threatened():
      if StrongerThanEnemy():
        ChangeState(Attack)
      else:
        ChangeState(Runaway)
    ...
  Attack:
    ...
    if not(StrongerThanEnemy()):
      ChangeState(Runaway)
    ...
  Runaway:
    ...
```

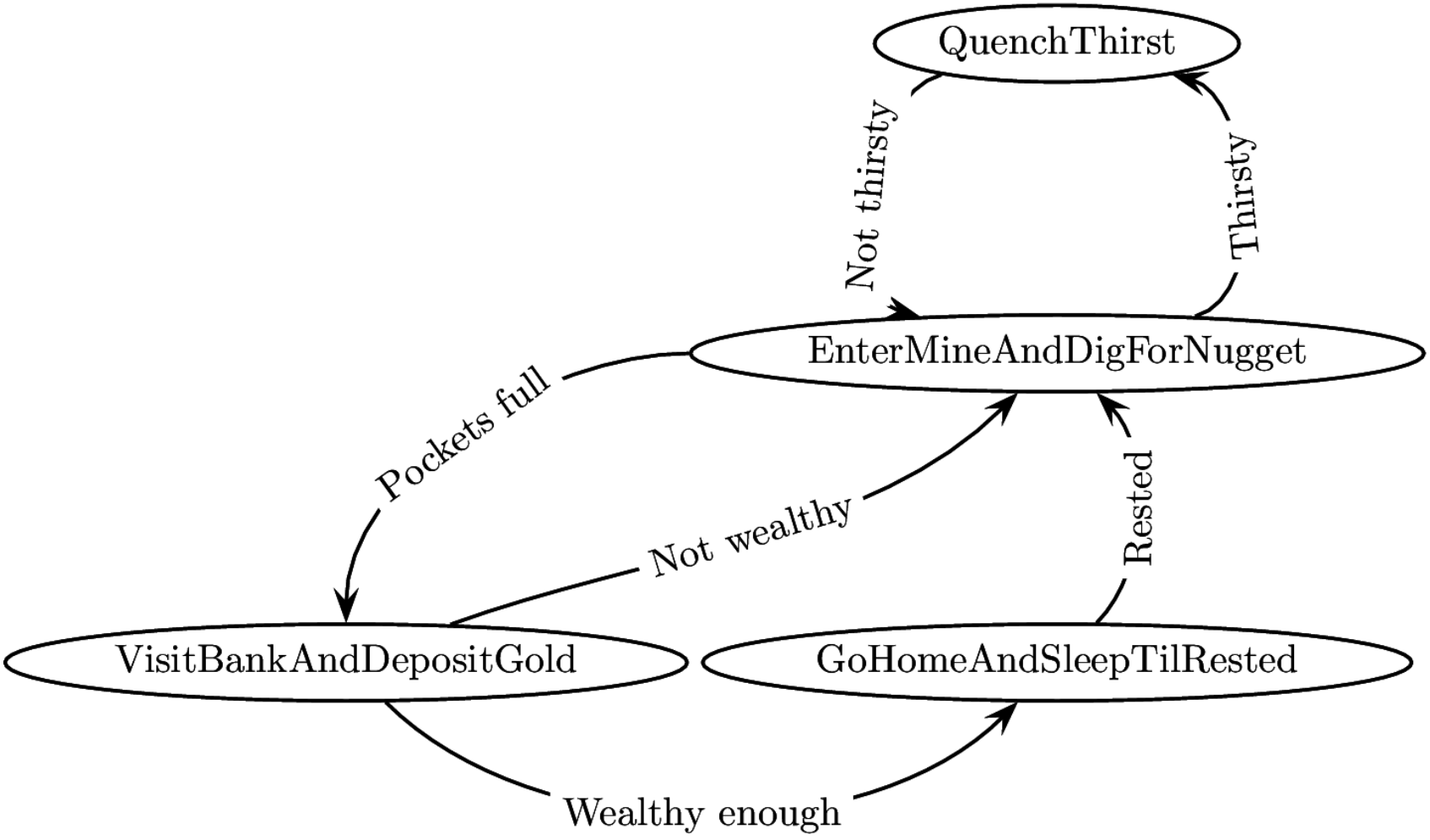
FSM Implementation 2.0: Transition Tables

Current State	**Condition**	**State Transition**
Runaway	Safe	Patrol
Attack	WeakerThanEnemy	Runaway
Patrol	Threatened AND StrongerThanEnemy	Attack
Patrol	Threatened AND WeakerThanEnemy	Runaway

FSM Implementation 3.0: Embedded Rules

- Embed transitions in the states themselves.
- There is no master control or code.
- Easily extensible–only affected states need be modified.
- **State Design Pattern**

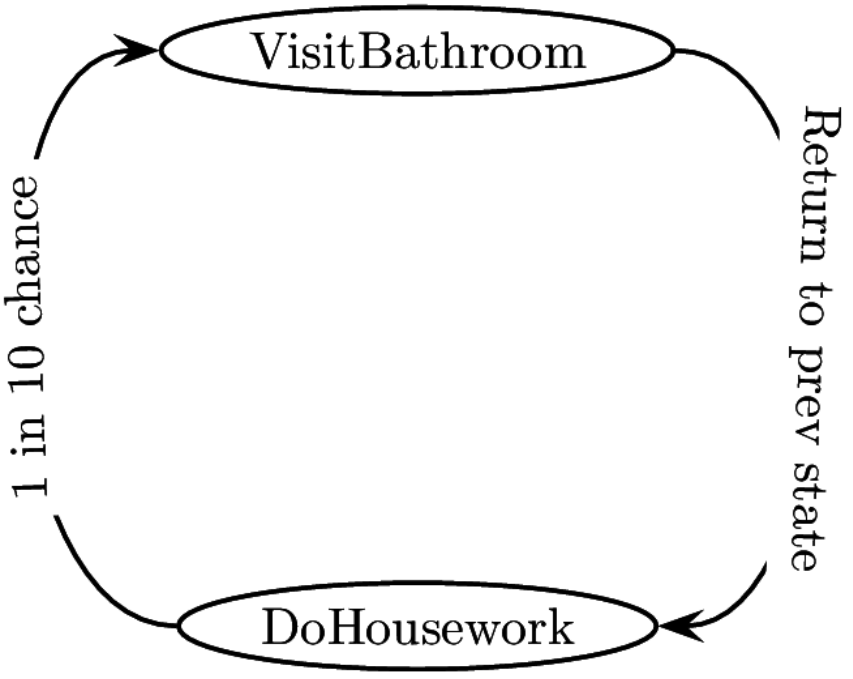
Miner Bob's State Diagram



Global States and State Blips

- Some things may happen at any time, no matter what state you're in.
  - E.g., going to the bathroom.
  - Use a **Global State** for these.
  - Global state updated every clock tick, in addition to the current state.
- Need to remember what you were doing previously.
  - Use a **Blip State** for these.
  - Push current state in to a previous state pointer.
  - Should use a stack, but sometimes not necessary.
- See Figure 2.4

Elsa's State Diagram



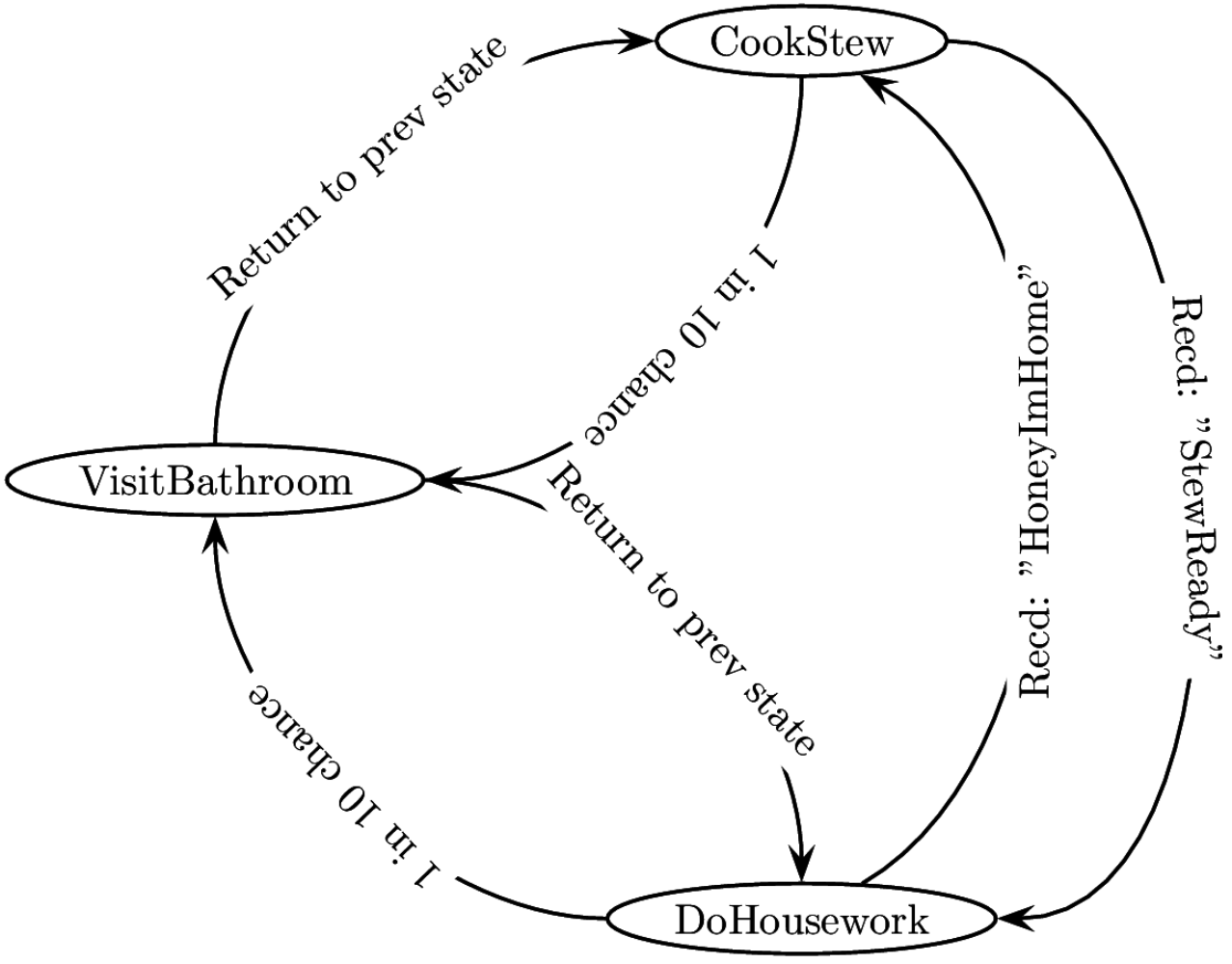
Messaging

- More general communication than collision.
- Better for intelligent agents.
- Bullets can message things they hit.
- Football player makes a pass, messages receiver, who responds accordingly.
- Injured NPC can message its comrades, who can come to give aid.
- Lighting a match in a dark room sends delayed message to burn out in thirty seconds.

Messaging

- Events in one object can trigger events in other objects.
- Intuitive way to design this is a **message** system.
- `Telegram` structure and `MessageDispatcher` class.
- `MessageDispatcher` uses priority queue of messages.
- Also uses an `EntityManager` so that IDs can be mapped to entities.
- Entities have `HandleMessage` added to their Global State.
- Global States have `OnMessage` added.
  - Returns a boolean so messages can be rerouted if necessary.
  - `MessageDispatcher` sends messages when time has arrived.

Elsa's State Diagram with Messaging



Summing Up

- FSMs a good way to manage complex behavior.
- Can use two FSMs in parallel, *e.g.*:
  - One for movement
  - One for weapon selection, aiming, firing
- Can use hierarchical FSMs: one state machine inside the state of another, *e.g.*:
  - High level states: **Explore**, **Combat**, **Patrol**
  - Low level states inside Combat:
    - Dodge
    - ChaseEnemy
    - Shoot