

---

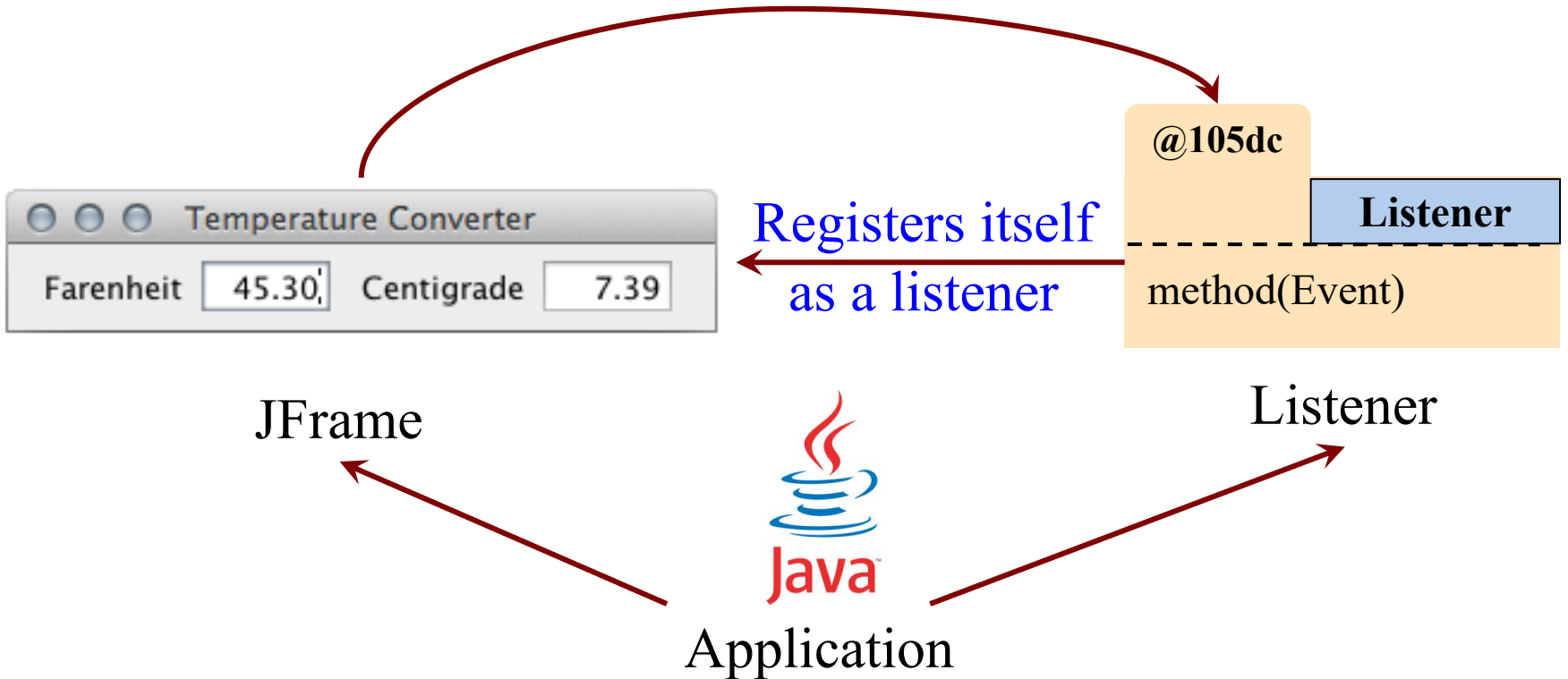
the  
gamedesigninitiative  
at cornell university

---

# Game Loop

# 2110-Level Apps are Event Driven

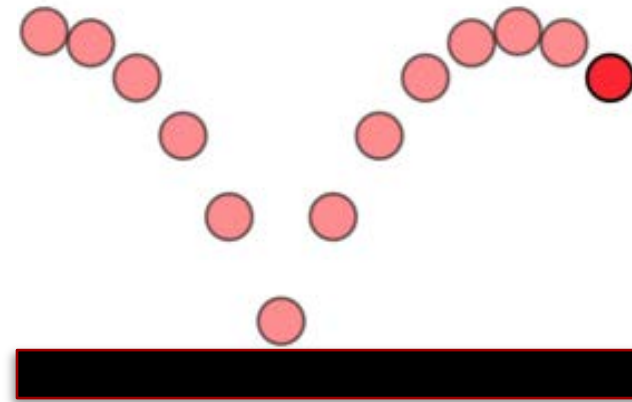
Generates event *e* and then  
calls `method(e)` on listener



# Limitations of the Event Model

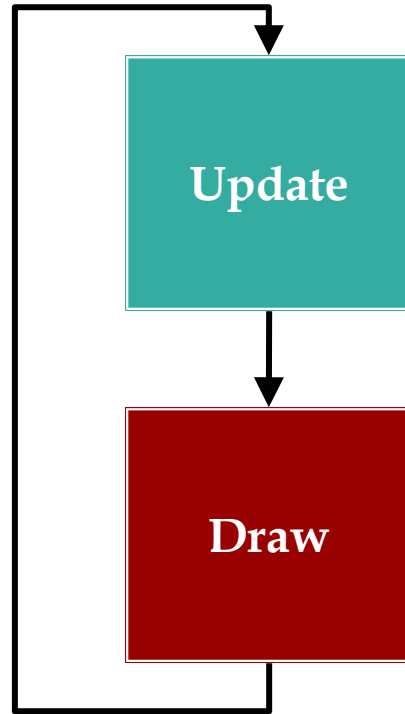
---

- Program only reacts to user input
  - Nothing changes if user does nothing
  - Desired behavior for productivity apps
- Games continue without input
  - Character animation
  - Clock timers
  - Enemy AI
  - Physics Simulations



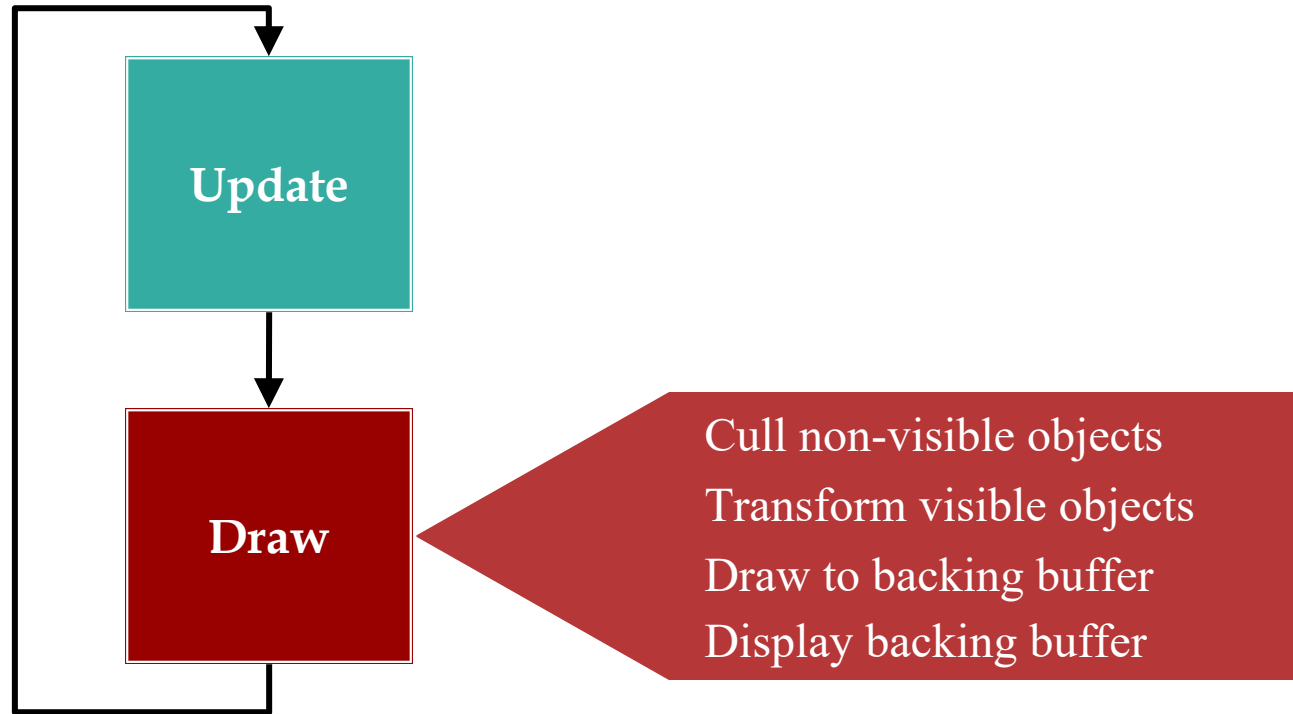
# The Game Loop

---



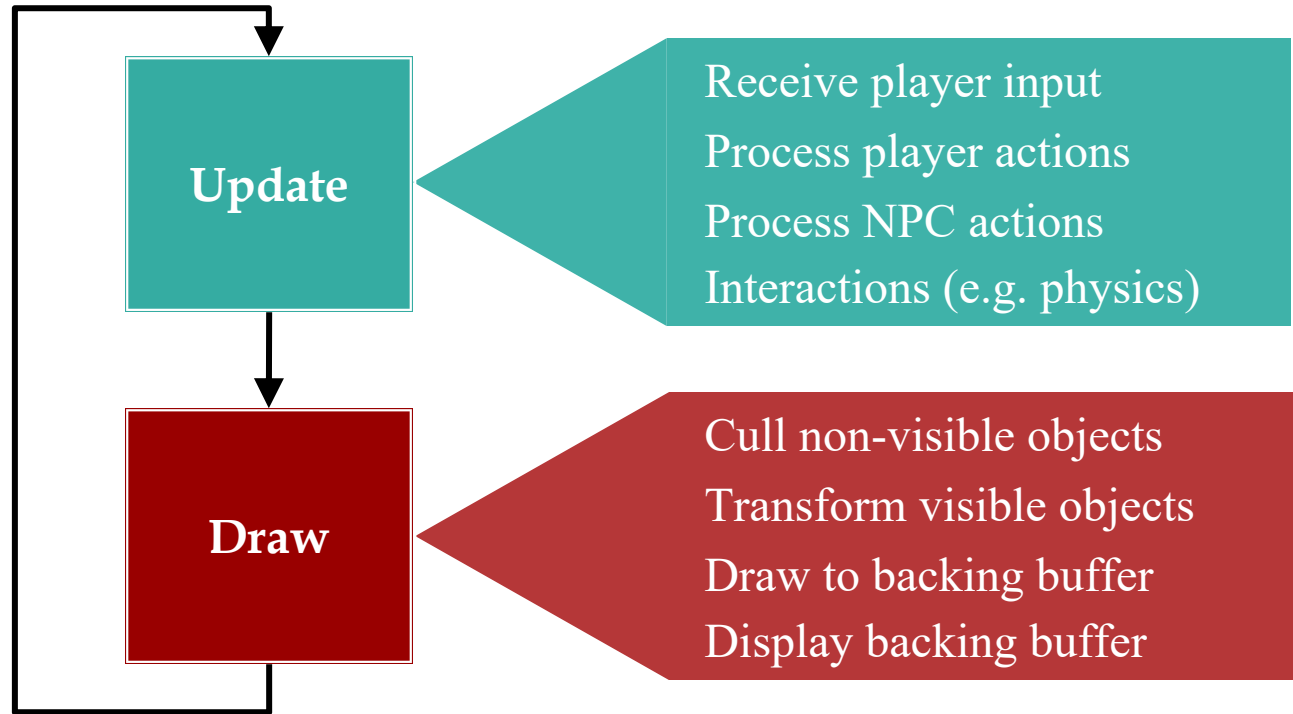
# The Game Loop

---



# The Game Loop

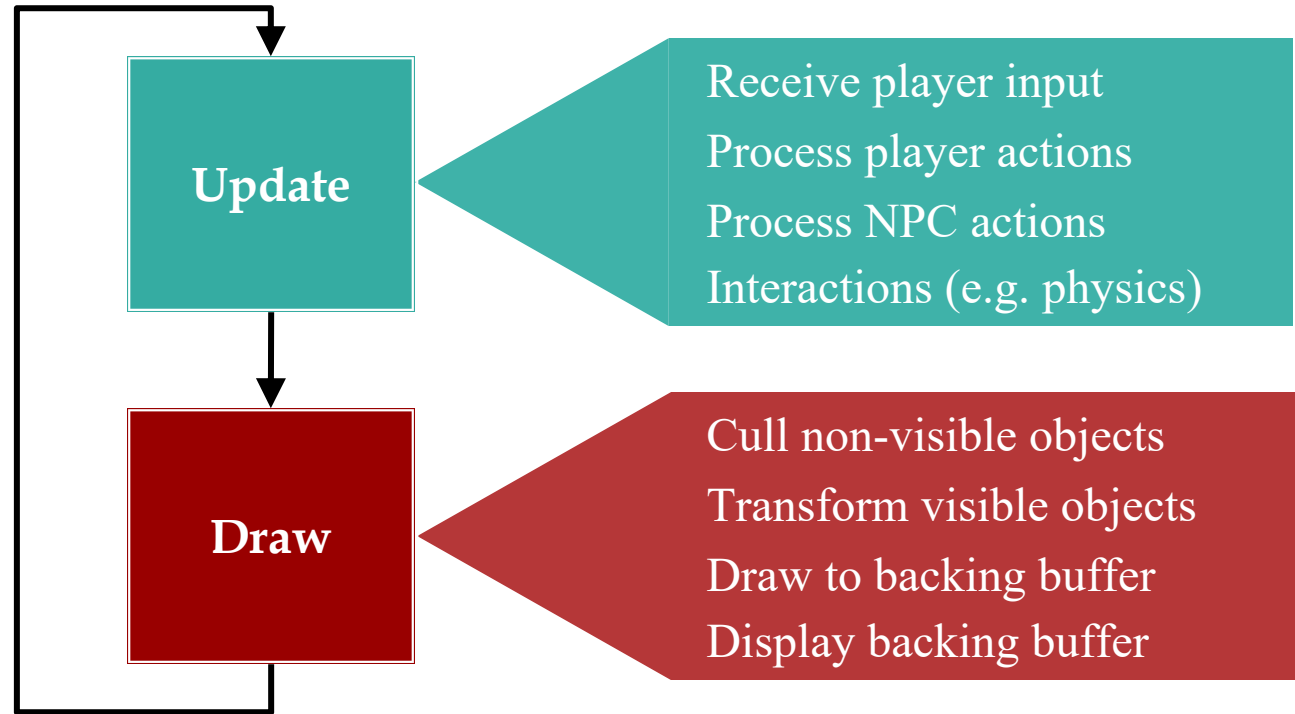
---



# The Game Loop

---

60 times/s  
=  
16.7 ms



# Few Words on Drawing

---

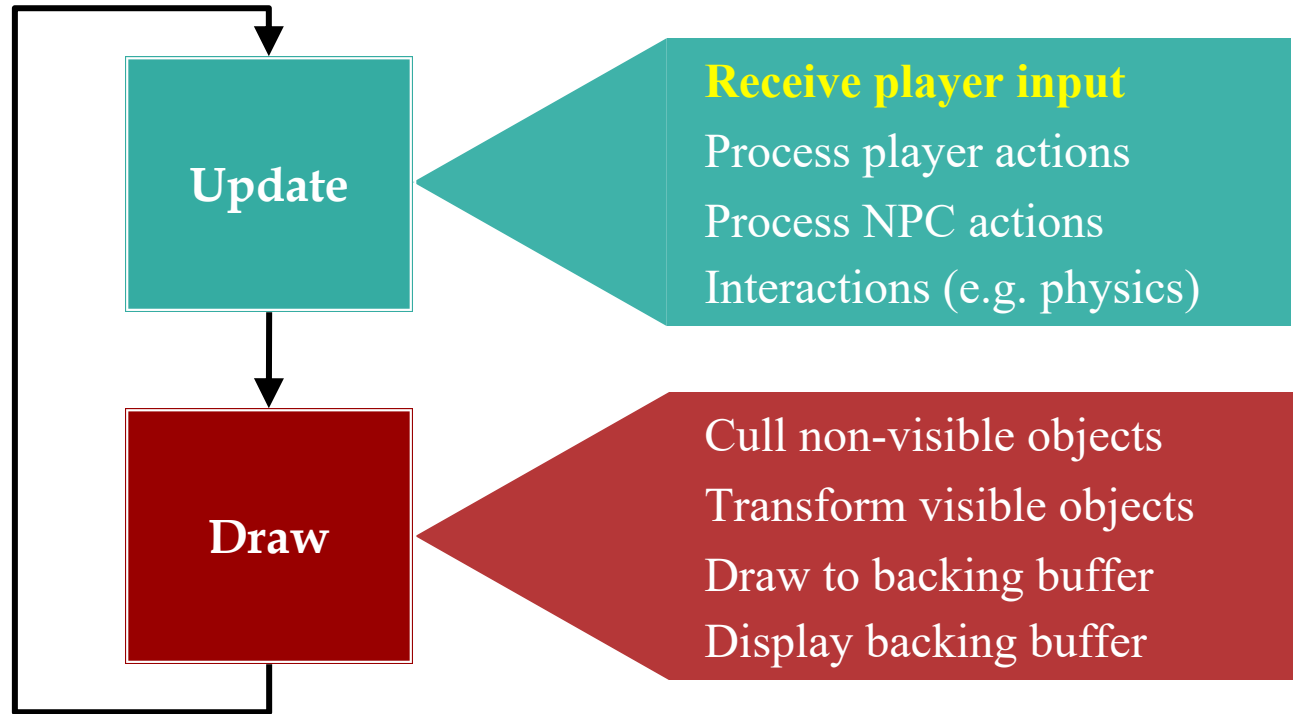
- Drawing needs to be **fast!**
  - Do as little computation as possible
  - But draw as few objects as possible
- Is this a contradiction?
  - Need to compute what to draw
  - So drawing *less* has extra overhead
- **Rule:** do **not** modify game state in draw
  - Any extra computation is local-only





# The Game Loop

---



# Player Input

---

- Traditional input is event-driven
  - Events capture state of controller
  - OS/VM generates events for you
  - Listeners react to events
- Game loop uses **polling** for input
  - Ask for controller state at start of loop
  - **Example**: What is joystick position?
  - If no change, do no actions that loop



# Problem with Polling

---

- Only one event per update loop
  - Multiple events are lost
  - **Example:** Fast typing
- Captures state at beginning
  - Short events are lost
  - **Example:** Fast clicks
- Event-driven mostly avoids these problems
  - Captures **all** events as they **happen**
  - But capture still has a frame-rate **resolution**



# Combining Input Approaches

---

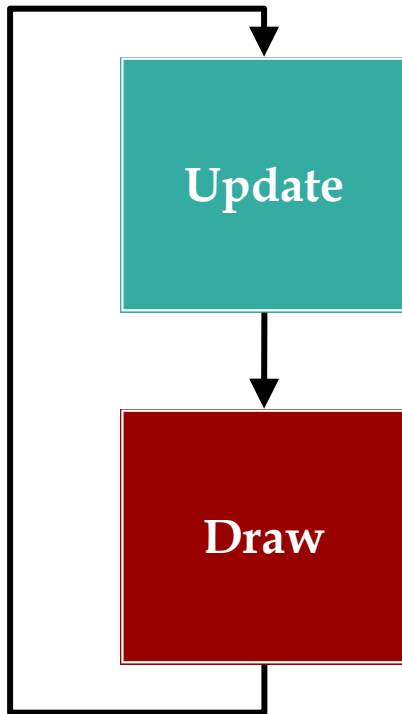
- LibGDX input is extremely flexible
  - Every input type supports events OR polling
- **Polling:** `Input` interface
  - Access it through the static class `GDX.Input`
  - Allows you to read the input state right now
- **Events:** `InputProcessor` interface
  - Register it with the appropriate input device
  - Works exactly like Swing listeners

# Problem: Timing

---

```
public class MyProcessor implements  
    InputProcessor {
```

```
    public void keyTyped(char c) {  
        // Do something with input
```



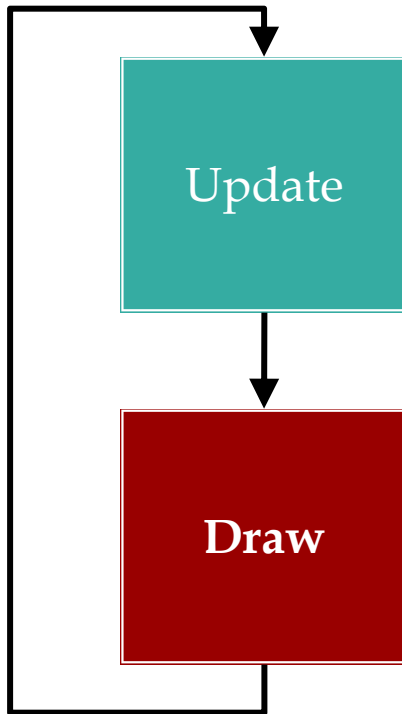
How do these  
fit together?

```
    }  
}
```

# Problem: Timing

```
public class MyProcessor implements  
InputProcessor {
```

```
public void keyTyped(char c) {  
    // Do something with input
```



How do these  
fit together?

```
}  
}
```

Unclear  
exactly when  
it is invoked

# Classic Producer-Consumer Problem

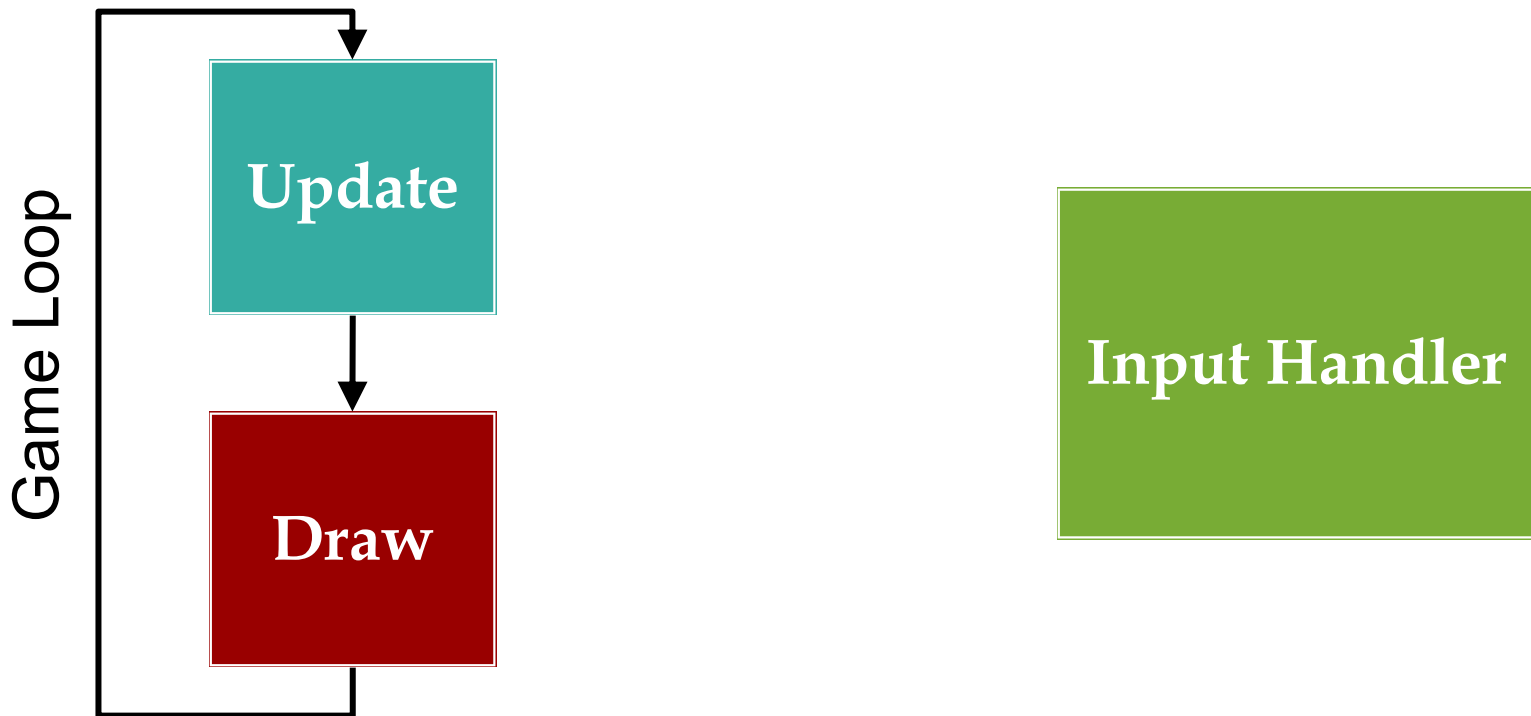
---

**Consumer**

---

**Producer**

---



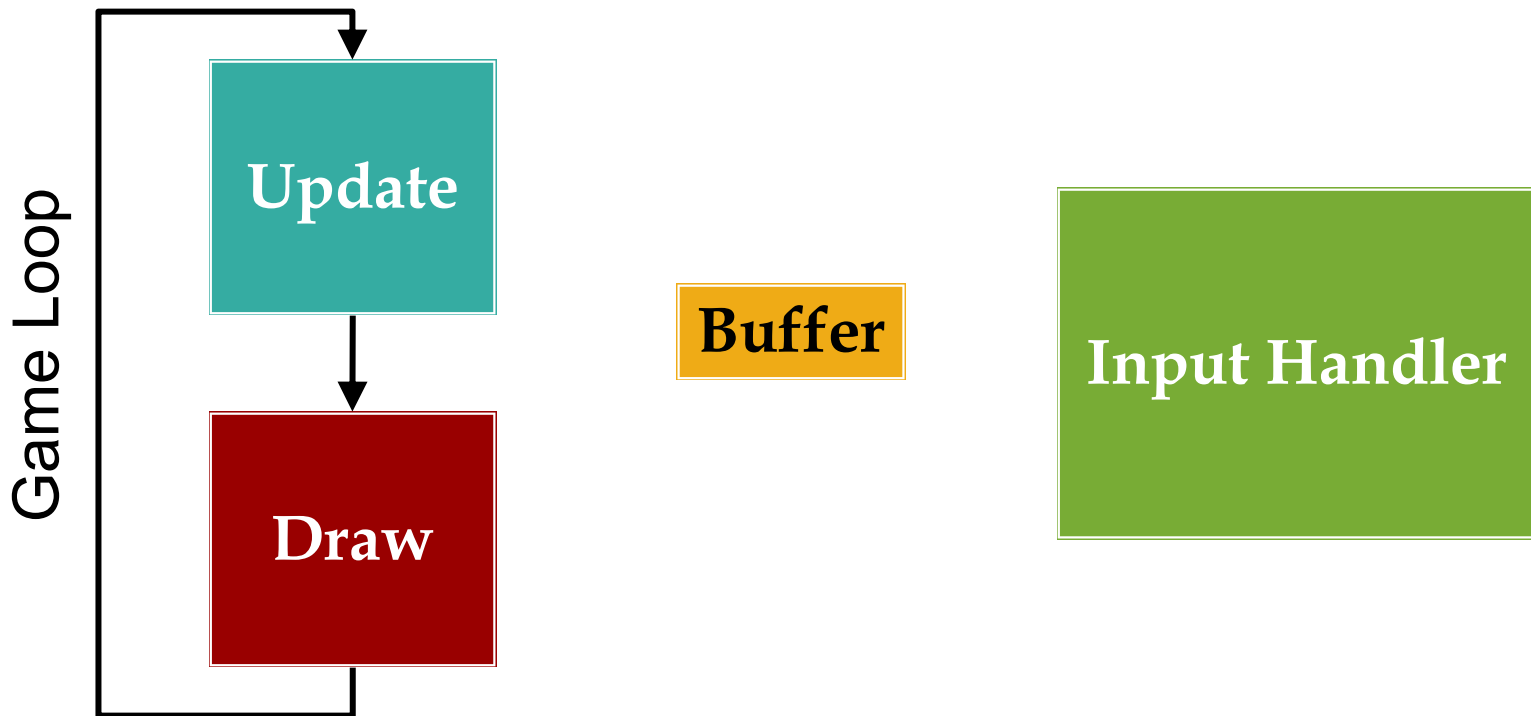
# Classic Producer-Consumer Problem

---

**Consumer**

**Producer**

---

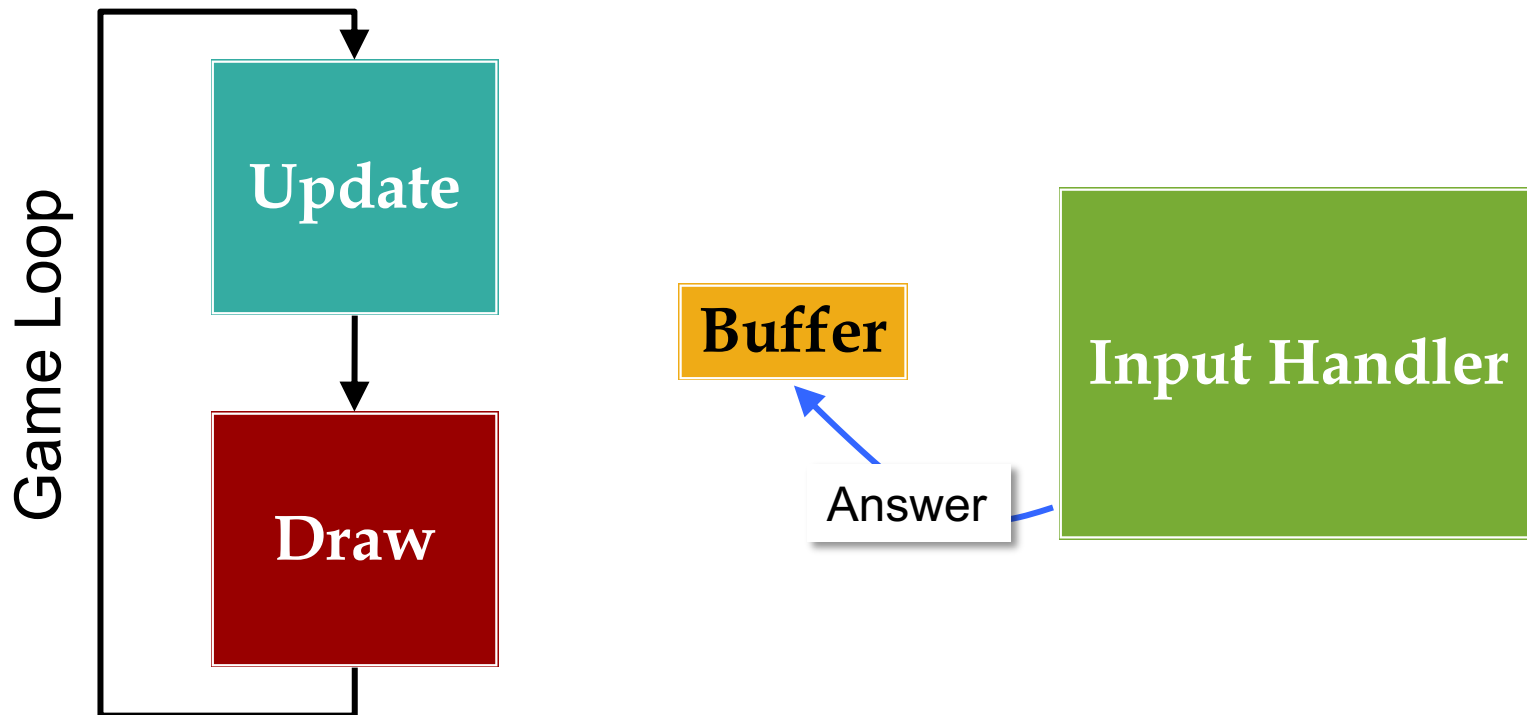




# Classic Producer-Consumer Problem

**Consumer**

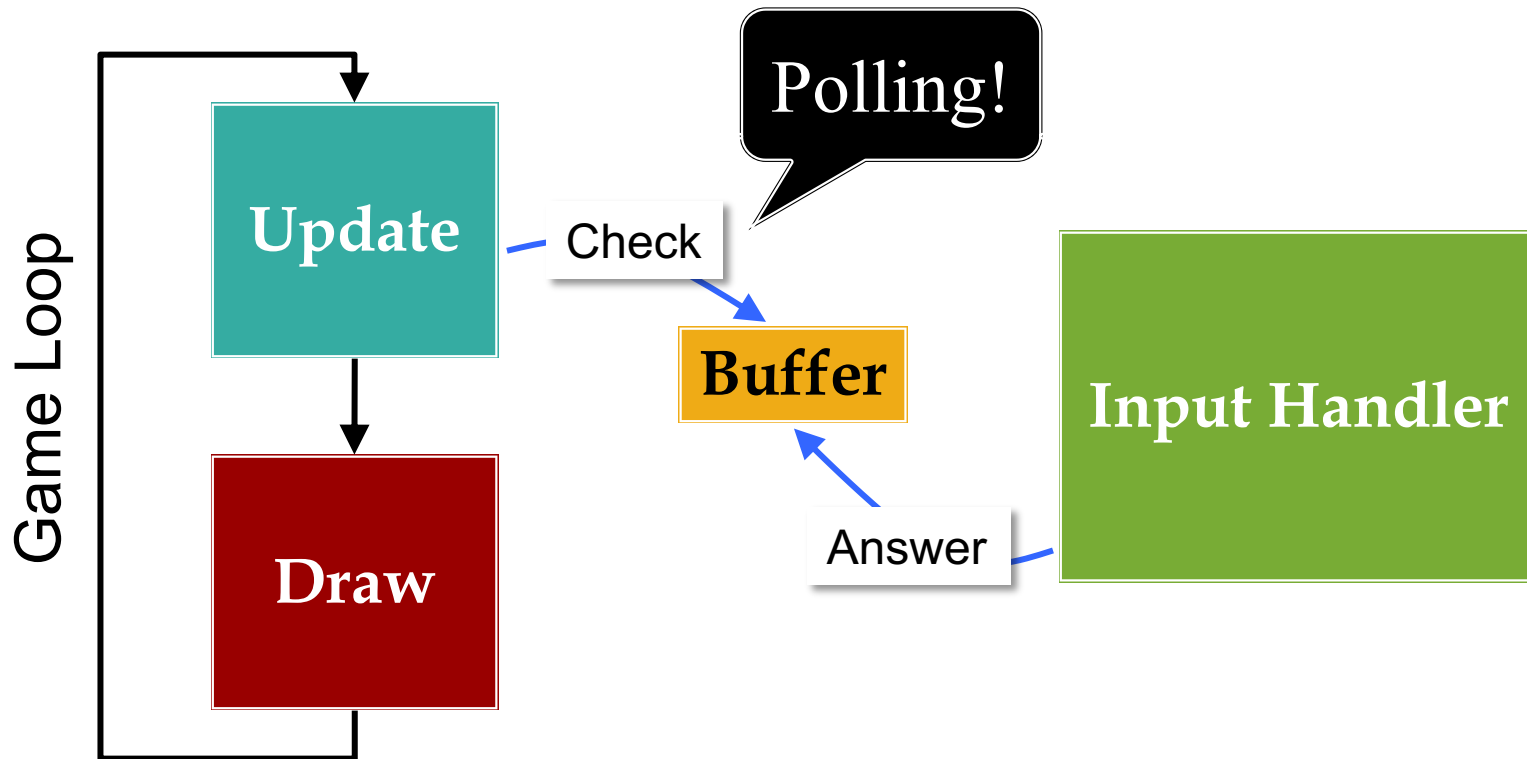
**Producer**



# Classic Producer-Consumer Problem

Consumer

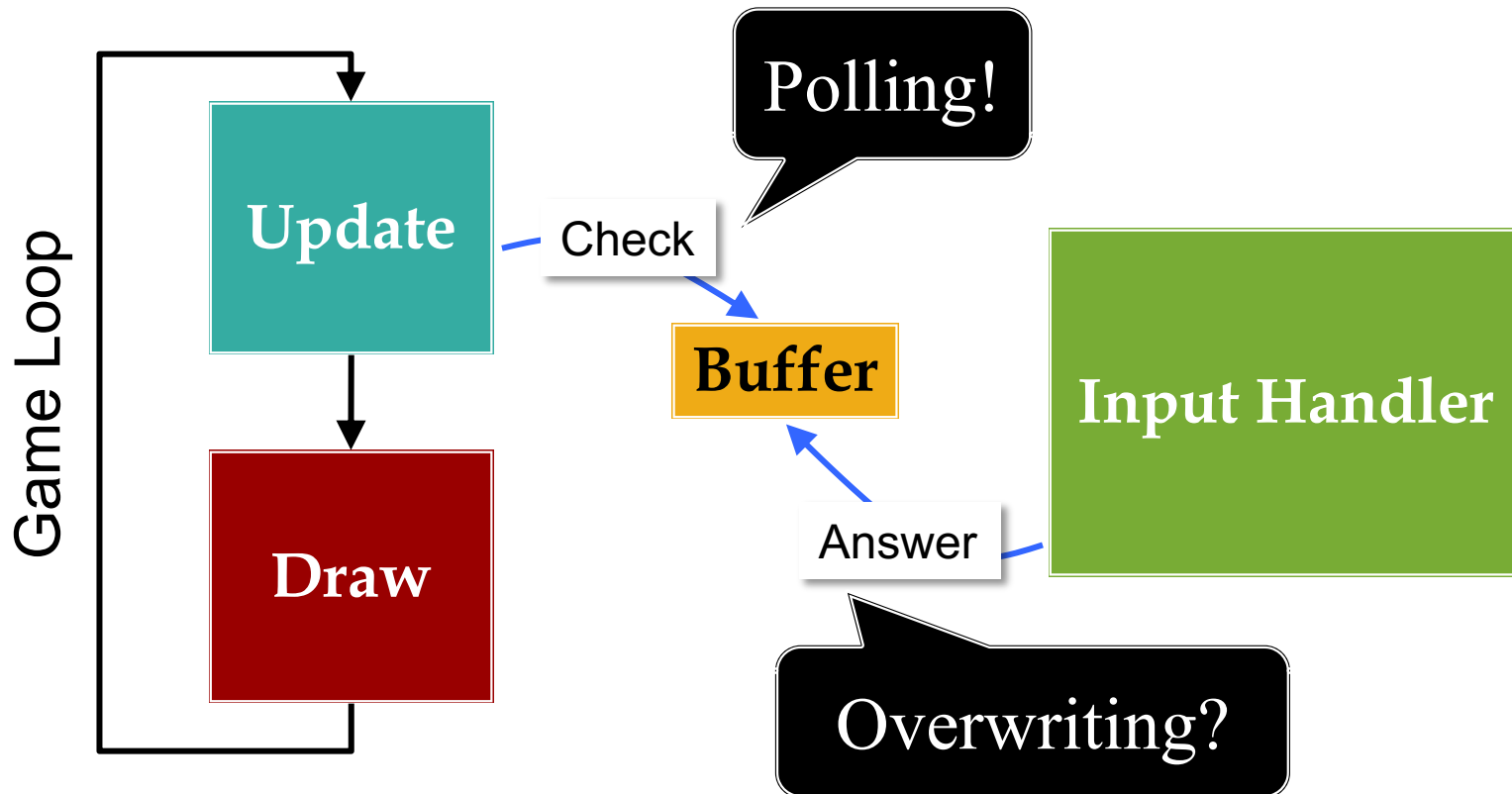
Producer



# Classic Producer-Consumer Problem

Consumer

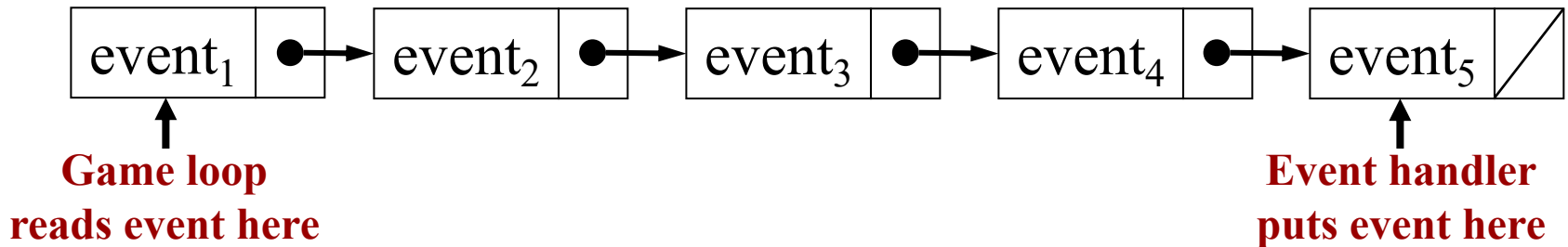
Producer



# Buffering Input

---

- If overwriting an issue, need an **event queue**
  - Input processor writes at end of the queue
  - Game loop reads from the front of queue



- Generally requires multiple **threads**
  - Event handler is (usually) OS/VM provided thread
  - Game loop itself is an additional thread

# Event Handlers: Really Necessary?

---

- Most of the time: **No**
  - Frame rate is short: 16.7 ms
  - Most events are  $> 16.7$  ms
  - Event loss not catastrophic
- Buffering is sometimes undesirable
  - Remembers every action ever done
  - But may take a longer time to process
  - If takes too long, just want to abort



# Picking the Right Input

---

## Polling

---

- When game loop is explicit
  - Actively animating screen
  - Must time input correctly
- **Example:** playing the game



## Event Driven

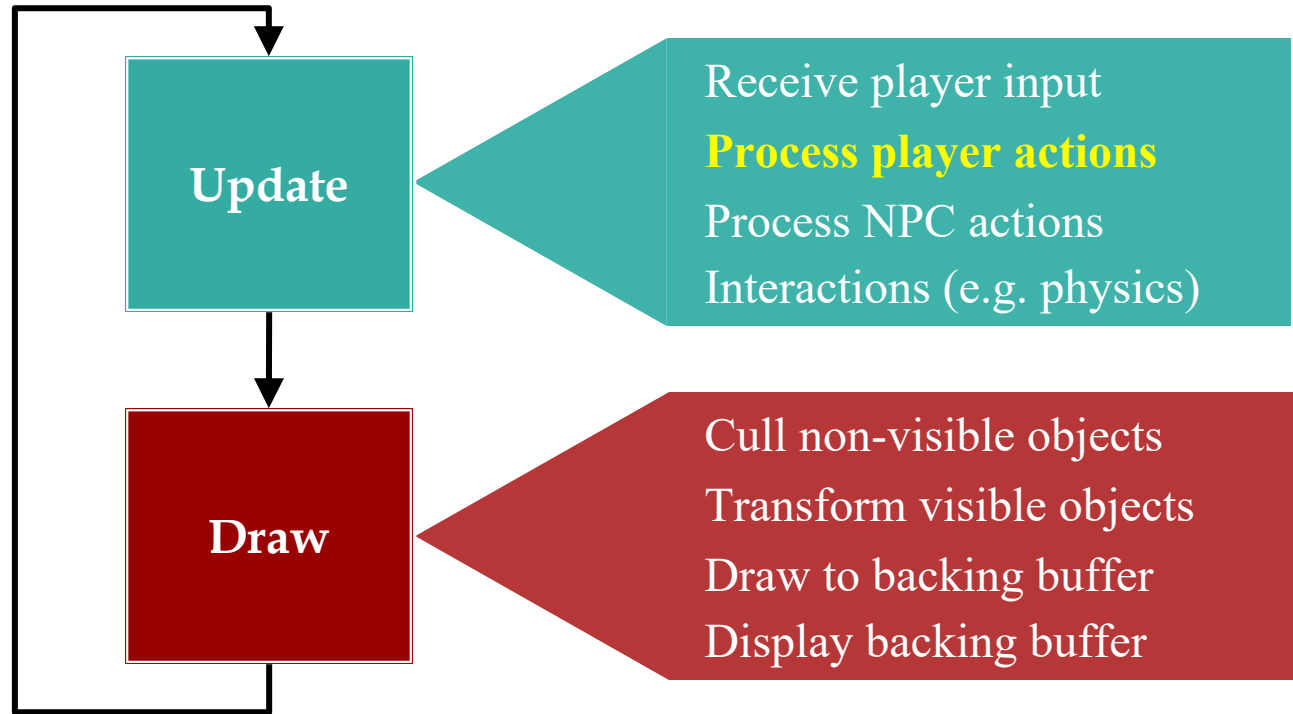
---

- When game loop is implicit
  - Art assets are largely static
  - Nothing to do if no input
- **Example:** a menu screen



# The Game Loop

---



# Player Actions

---

- Actions alter the game state
  - Can alter player state: **movement**
  - Can alter opponent state: **damage**
- Player actions correspond to user input
  - Choice is determined by input controller
  - Else action is performed by computer
- These are your game **verbs!**



# Abstract Actions from Input

---

- **Actions:** functions that modify game state
  - `move(dx,dy)` modifies `x`, `y` by `dx`, `dy`
  - `attack(o)` attacks opponent `o`
- Input controller **maps** input to actions
  - Read input state from controller
  - Pick an action and call that function
- Input handler should never alter state directly!

# Abstract Actions from Input

---

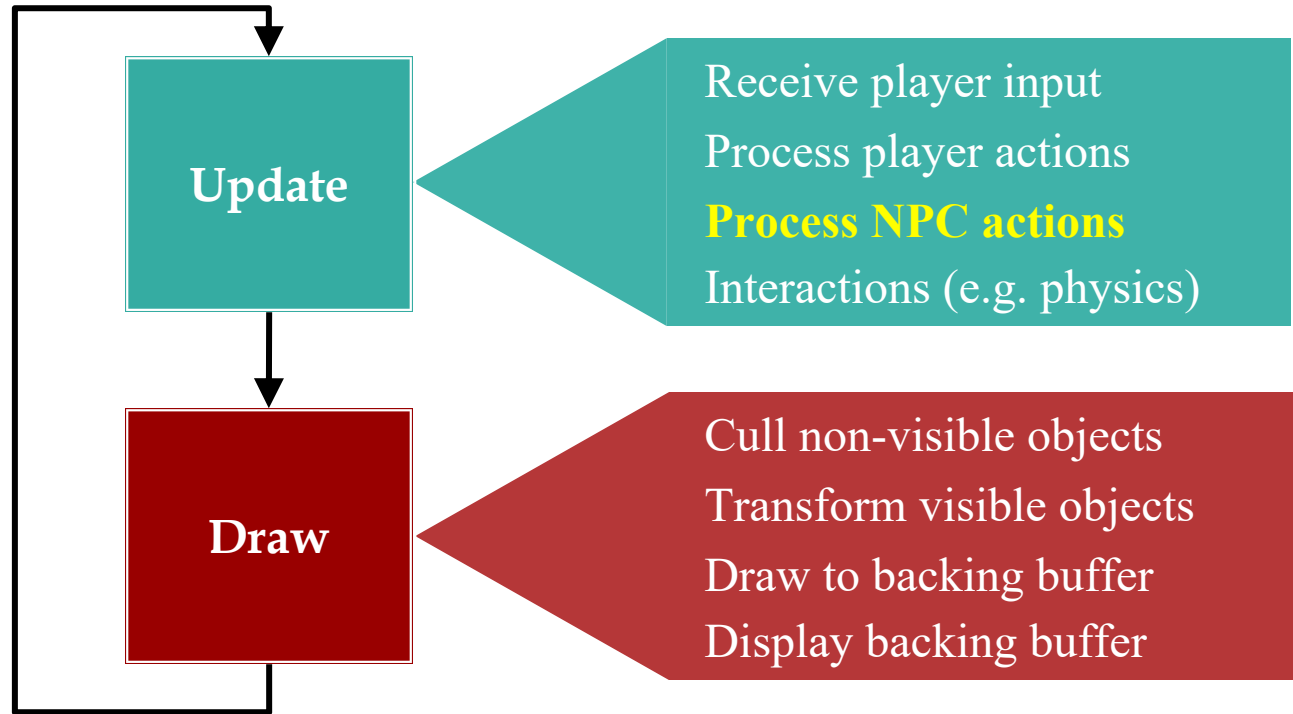
- **Actions:** functions that modify game state
  - `move(dx,dy)` modifies `x`, `y` by `dx`, `dy`
  - `attack(o)` attacks opponent `o`
- Input controller **maps** input to actions
  - Read input state from controller
  - Pick an action and call that function
- Input handler should never alter state directly!



**Design** versus  
**Implementation**

# The Game Loop

---



# NPC: Non-Player Character

---

- NPC is an intelligent computer-controlled entity
  - Unlike a physics object, it can act, not just interact
  - Sometimes called an *agent*
- NPCs have their own actions/verbs
  - But no input controller to choose
- Work on **sense-think-act** cycle
  - **Sense:** perceive the world around it
  - **Think:** choose an action to perform
  - **Act:** update the game state



# Act versus Sense-Think

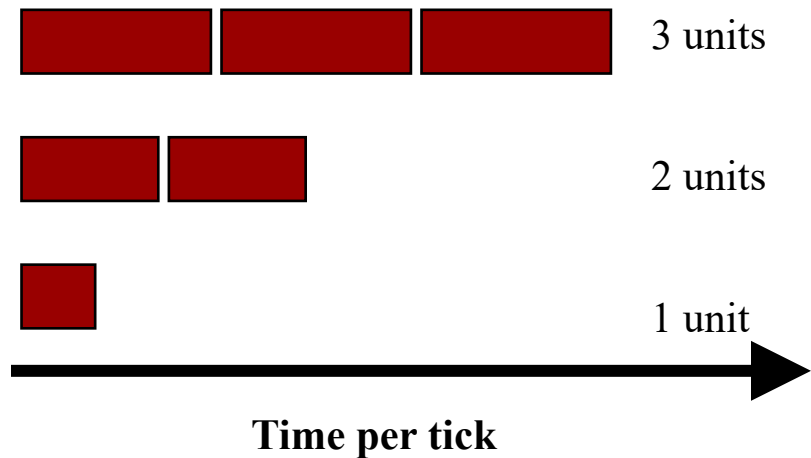
---

- Act should be *very* fast!
  - Function to update state
  - **Example**: apply velocity
  - Exactly like the player
- Sense-think unique to NPC
  - The *hard* computation
  - Focus of AI lectures
- **Multiplayer**: Replace sense-think with human decision



# Problem with Sensing

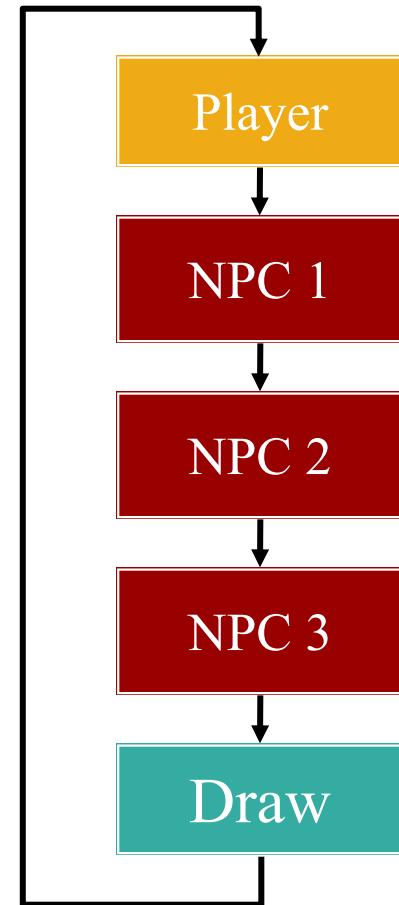
- Sensing may be slow!
  - Consider *all* objects
- Example: morale
  - $n$  knights,  $n$  skeletons
  - Knights fear skeletons
  - Proportional to # seen
- Count skeletons in view
  - $O(n)$  to count skeletons
  - $O(n^2)$  for all units



# Processing NPCs

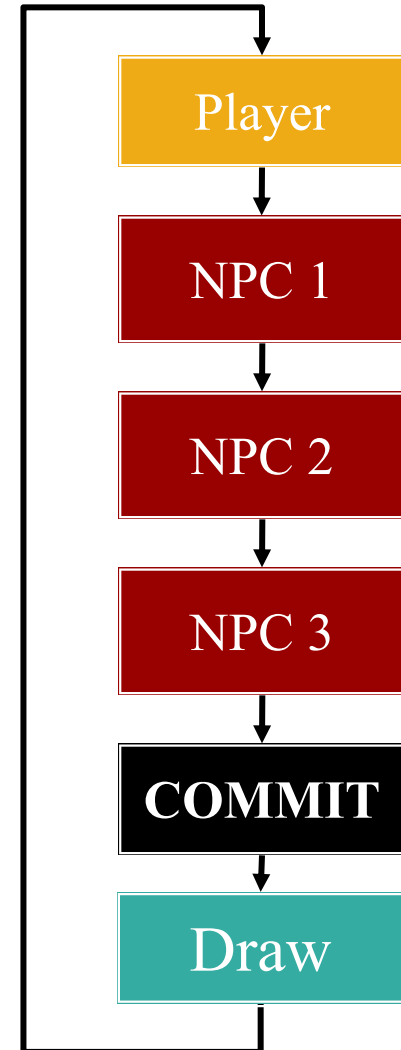
---

- Naïve solution: **sequentially**
- **Problem:** NPCs react too fast!
  - Each reads the actions of previous
  - Even before drawn on screen!



# Processing NPCs

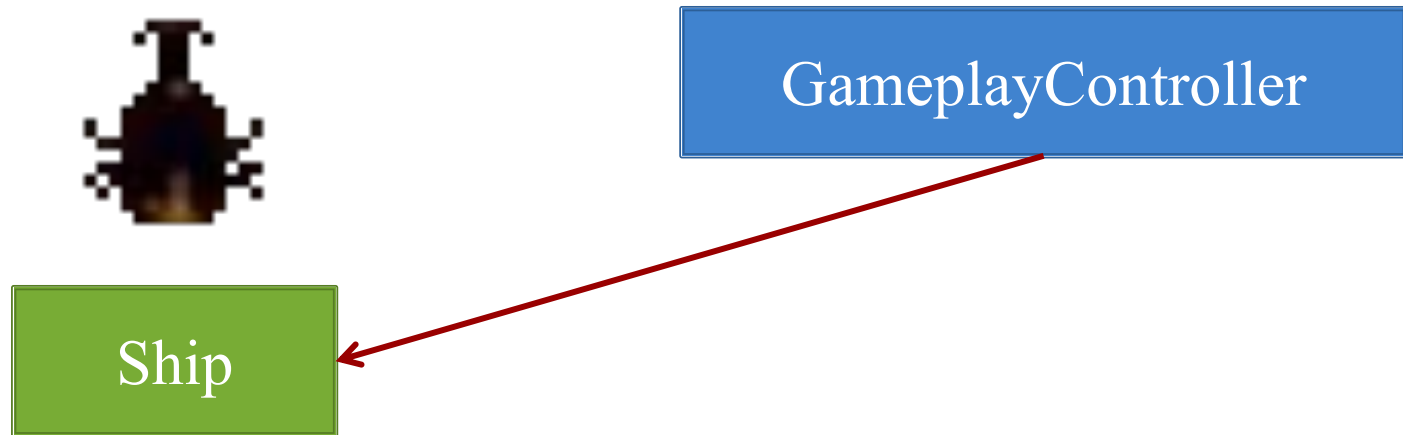
- Naïve solution: *sequentially*
- **Problem:** NPCs react too fast!
  - Each reads the actions of previous
  - Even before drawn on screen!
- **Idea:** only react to what can see
  - *Choose* actions, but don't perform
  - Once all chosen, then perform
  - Another reason to abstract actions





# Processing Actions in Lab 3

---



- Decides whether to shoot
- Stores intent in the object
- But **DOES NOT** shoot
- Waits until objects commit
- Checks intent in Ship object
- Performs action for intent

# Problem: Pathfinding

- Focus of Game Lab 2
  - Crucial if top view
  - Major area of research
- Potentially very slow
  - $n$  NPCs,  $g$  grid squares
  - Dijkstra:  $O(g^2)$
  - For each NPC:  $O/ng^2)$
- **Moving obstacles?**

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
3	2	1	2	3	4	5	6	7		13	14	15	16
4	3	2	3	4	5	6	7	8		12	13	14	15
5	4	3	4	5	6	7	8	9	10	11	12	13	14
6	5	4	5	6	7	8	9	10	11	12	13	14	15

# Problem: Pathfinding

- Focus of Game Lab 2
  - Crucial if top view
  - Major area of research

- Potentially very slow

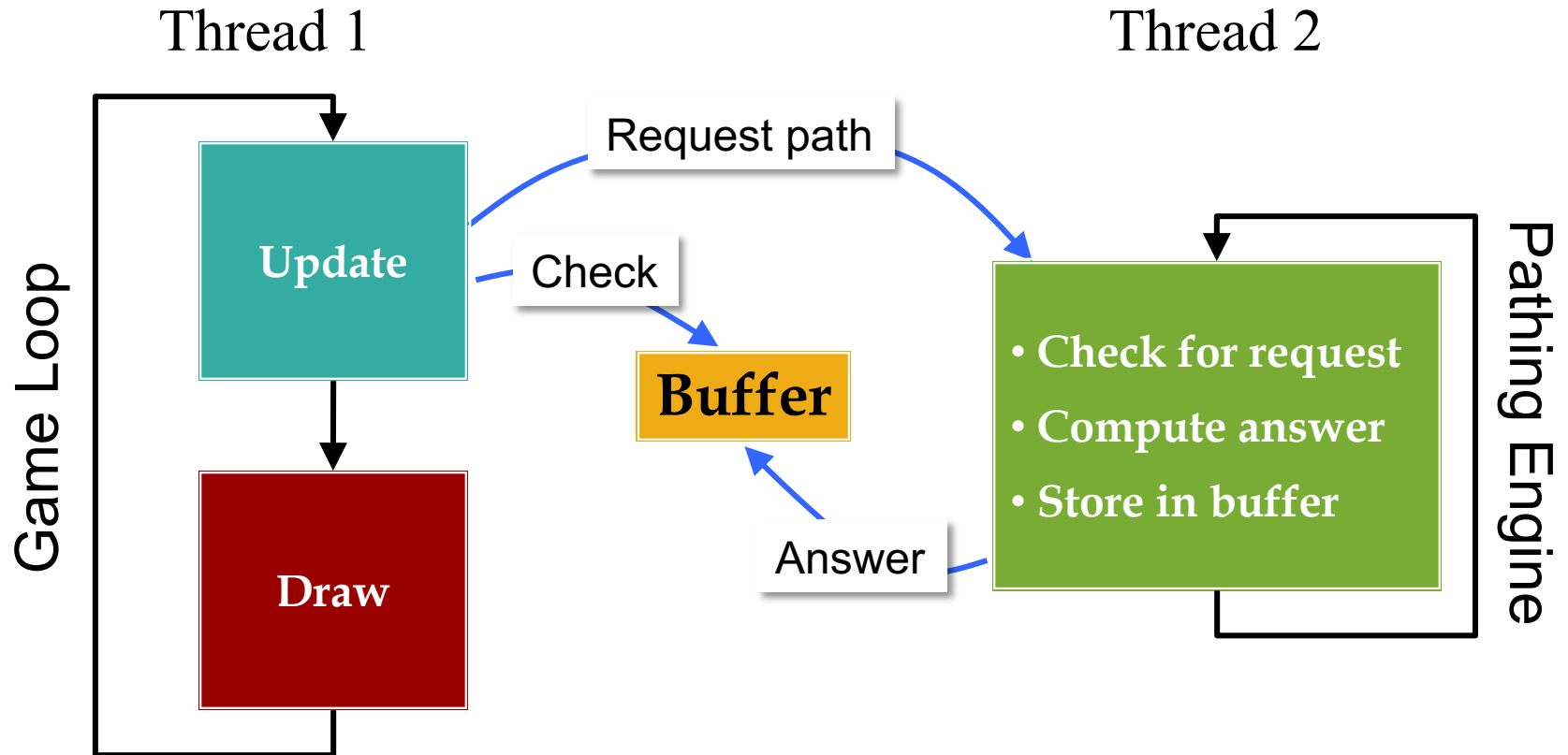
Often more than 16.7ms

- $O(n^2)$
- For each NPC:  $O(n^2)$

- **Moving obstacles?**

7	6	5	6	7	8	9	10	11		19	20	21	22
6	5	4	5	6	7	8	9	10		18	19	20	21
5	4	3	4	5	6	7	8	9		17	18	19	20
4	3	2	3	4	5	6	7	8		16	17	18	19
3	2	1	2	3	4	5	6	7		15	16	17	18
2	1	0	1	2	3	4	5	6		14	15	16	17
1	0	-1	0	1	2	3	4	5		13	14	15	16
0	-1	-2	-1	0	1	2	3	4		12	13	14	15
-1	-2	-3	-2	-1	0	1	2	3		11	12	13	14
-2	-3	-4	-3	-2	-1	0	1	2		10	11	12	13
-3	-4	-5	-4	-3	-2	-1	0	1		9	10	11	12
-4	-5	-6	-5	-4	-3	-2	-1	0		8	9	10	11
-5	-6	-7	-6	-5	-4	-3	-2	-1		7	8	9	10
-6	-7	-8	-7	-6	-5	-4	-3	-2		6	7	8	9
-7	-8	-9	-8	-7	-6	-5	-4	-3		5	6	7	8
-8	-9	-10	-9	-8	-7	-6	-5	-4		4	5	6	7
-9	-10	-11	-10	-9	-8	-7	-6	-5		3	4	5	6
-10	-11	-12	-11	-10	-9	-8	-7	-6		2	3	4	5
-11	-12	-13	-12	-11	-10	-9	-8	-7		1	2	3	4
-12	-13	-14	-13	-12	-11	-10	-9	-8		0	1	2	3
-13	-14	-15	-14	-13	-12	-11	-10	-9		-1	0	1	2
-14	-15	-16	-15	-14	-13	-12	-11	-10		-2	-1	0	1
-15	-16	-17	-16	-15	-14	-13	-12	-11		-3	-2	-1	0
-16	-17	-18	-17	-16	-15	-14	-13	-12		-4	-3	-2	-1
-17	-18	-19	-18	-17	-16	-15	-14	-13		-5	-4	-3	-2
-18	-19	-20	-19	-18	-17	-16	-15	-14		-6	-5	-4	-3
-19	-20	-21	-20	-19	-18	-17	-16	-15		-7	-6	-5	-4
-20	-21	-22	-21	-20	-19	-18	-17	-16		-8	-7	-6	-5
-21	-22	-23	-22	-21	-20	-19	-18	-17		-9	-8	-7	-6
-22	-23	-24	-23	-22	-21	-20	-19	-18		-10	-9	-8	-7
-23	-24	-25	-24	-23	-22	-21	-20	-19		-11	-10	-9	-8
-24	-25	-26	-25	-24	-23	-22	-21	-20		-12	-11	-10	-9
-25	-26	-27	-26	-25	-24	-23	-22	-21		-13	-12	-11	-10
-26	-27	-28	-27	-26	-25	-24	-23	-22		-14	-13	-12	-11
-27	-28	-29	-28	-27	-26	-25	-24	-23		-15	-14	-13	-12
-28	-29	-30	-29	-28	-27	-26	-25	-24		-16	-15	-14	-13
-29	-30	-31	-30	-29	-28	-27	-26	-25		-17	-16	-15	-14
-30	-31	-32	-31	-30	-29	-28	-27	-26		-18	-17	-16	-15
-31	-32	-33	-32	-31	-30	-29	-28	-27		-19	-18	-17	-16
-32	-33	-34	-33	-32	-31	-30	-29	-28		-20	-19	-18	-17
-33	-34	-35	-34	-33	-32	-31	-30	-29		-21	-20	-19	-18
-34	-35	-36	-35	-34	-33	-32	-31	-30		-22	-21	-20	-19
-35	-36	-37	-36	-35	-34	-33	-32	-31		-23	-22	-21	-20
-36	-37	-38	-37	-36	-35	-34	-33	-32		-24	-23	-22	-21
-37	-38	-39	-38	-37	-36	-35	-34	-33		-25	-24	-23	-22
-38	-39	-40	-39	-38	-37	-36	-35	-34		-26	-25	-24	-23
-39	-40	-41	-40	-39	-38	-37	-36	-35		-27	-26	-25	-24
-40	-41	-42	-41	-40	-39	-38	-37	-36		-28	-27	-26	-25
-41	-42	-43	-42	-41	-40	-39	-38	-37		-29	-28	-27	-26
-42	-43	-44	-43	-42	-41	-40	-39	-38		-30	-29	-28	-27
-43	-44	-45	-44	-43	-42	-41	-40	-39		-31	-30	-29	-28
-44	-45	-46	-45	-44	-43	-42	-41	-40		-32	-31	-30	-29
-45	-46	-47	-46	-45	-44	-43	-42	-41		-33	-32	-31	-30
-46	-47	-48	-47	-46	-45	-44	-43	-42		-34	-33	-32	-31
-47	-48	-49	-48	-47	-46	-45	-44	-43		-35	-34	-33	-32
-48	-49	-50	-49	-48	-47	-46	-45	-44		-36	-35	-34	-33
-49	-50	-51	-50	-49	-48	-47	-46	-45		-37	-36	-35	-34
-50	-51	-52	-51	-50	-49	-48	-47	-46		-38	-37	-36	-35
-51	-52	-53	-52	-51	-50	-49	-48	-47		-39	-38	-37	-36
-52	-53	-54	-53	-52	-51	-50	-49	-48		-40	-39	-38	-37
-53	-54	-55	-54	-53	-52	-51	-50	-49		-41	-40	-39	-38
-54	-55	-56	-55	-54	-53	-52	-51	-50		-42	-41	-40	-39
-55	-56	-57	-56	-55	-54	-53	-52	-51		-43	-42	-41	-40
-56	-57	-58	-57	-56	-55	-54	-53	-52		-44	-43	-42	-41
-57	-58	-59	-58	-57	-56	-55	-54	-53		-45	-44	-43	-42
-58	-59	-60	-59	-58	-57	-56	-55	-54		-46	-45	-44	-43
-59	-60	-61	-60	-59	-58	-57	-56	-55		-47	-46	-45	-44
-60	-61	-62	-61	-60	-59	-58	-57	-56		-48	-47	-46	-45
-61	-62	-63	-62	-61	-60	-59	-58	-57		-49	-48	-47	-46
-62	-63	-64	-63	-62	-61	-60	-59	-58		-50	-49	-48	-47
-63	-64	-65	-64	-63	-62	-61	-60	-59		-51	-50	-49	-48
-64	-65	-66	-65	-64	-63	-62	-61	-60		-52	-51	-50	-49
-65	-66	-67	-66	-65	-64	-63	-62	-61		-53	-52	-51	-50
-66	-67	-68	-67	-66	-65	-64	-63	-62		-54	-53	-52	-51
-67	-68	-69	-68	-67	-66	-65	-64	-63		-55	-54	-53	-52
-68	-69	-70	-69	-68	-67	-66	-65	-64		-56	-55	-54	-53
-69	-70	-71	-70	-69	-68	-67	-66	-65		-57	-56	-55	-54
-70	-71	-72	-71	-70	-69	-68	-67	-66		-58	-57	-56	-55
-71	-72	-73	-72	-71	-70	-69	-68	-67		-59	-58	-57	-56
-72	-73	-74	-73	-72	-71	-70	-69	-68		-60	-59	-58	-57
-73	-74	-75	-74	-73	-72	-71	-70	-69		-61	-60	-59	-58
-74	-75	-76	-75	-74	-73	-72	-71	-70		-62	-61	-60	-59
-75	-76	-77	-76	-75	-74	-73	-72	-71		-63	-62	-61	-60
-76	-77	-78	-77	-76	-75	-74	-73	-72		-64	-63	-62	-61
-77	-78	-79	-78	-77	-76	-75	-74	-73		-65	-64	-63	-62
-78	-79	-80	-79	-78	-77	-76	-75	-74		-66	-65	-64	-63
-79	-80	-81	-80	-79	-78	-77	-76	-75		-67	-66	-65	-64
-80	-81	-82	-81	-80	-79	-78	-77	-76		-68	-67	-66	-65
-81	-82	-83	-82	-81	-80	-79	-78	-77		-69	-68	-67	-66
-82	-83	-84	-83	-82	-81	-80	-79	-78		-70	-69	-68	-67
-83	-84	-85	-84	-83	-82	-81	-80	-79		-71	-70	-69	-68
-84	-85	-86	-85	-84	-83	-82	-81	-80		-72	-71	-70	-69
-85	-86	-87	-86	-85	-84	-83	-82	-81		-73	-72	-71	-70
-86	-87	-88	-87	-86	-85	-84	-83	-82		-74	-73	-72	-71
-87	-88	-89	-88	-87	-86	-85	-84	-83		-75	-74	-73	-72
-88	-89	-90	-89	-88	-87	-86	-85	-84		-76	-75	-74	-73
-89	-90	-91	-90	-89	-88	-87	-86	-85		-77	-76	-75	-74
-90	-91	-92	-91	-90	-89	-88	-87	-86		-78	-77	-76	-75
-91	-92	-93	-92	-91	-90	-89	-88	-87		-79	-78	-77	-76
-92	-93	-94	-93	-92	-91	-90	-89	-88		-80	-79	-78	-77
-93	-94	-95	-94	-93	-92	-91	-90	-89		-81	-80	-79	-78
-94	-95	-96	-95	-94	-93	-92	-91	-90		-82	-81	-80	-79
-95	-96	-97	-96	-95	-94	-93	-92	-91		-83	-82	-81	-80
-96	-97	-98	-97	-96	-95	-94	-93	-92		-84	-83	-82	-81
-97	-98	-99	-98	-97	-96	-95	-94	-93		-85	-84	-83	-82
-98	-99	-100	-99	-98	-97	-96	-95	-94		-86	-85	-84	-83
-99	-100	-101	-100	-99	-98	-97	-96	-95		-87	-86	-85	-84
-100	-101	-102	-101	-100	-99	-98	-97	-96		-88	-87	-86	-85

# Asynchronous Pathfinding



**Looks like input buffering!**

# Asynchronous Pathfinding

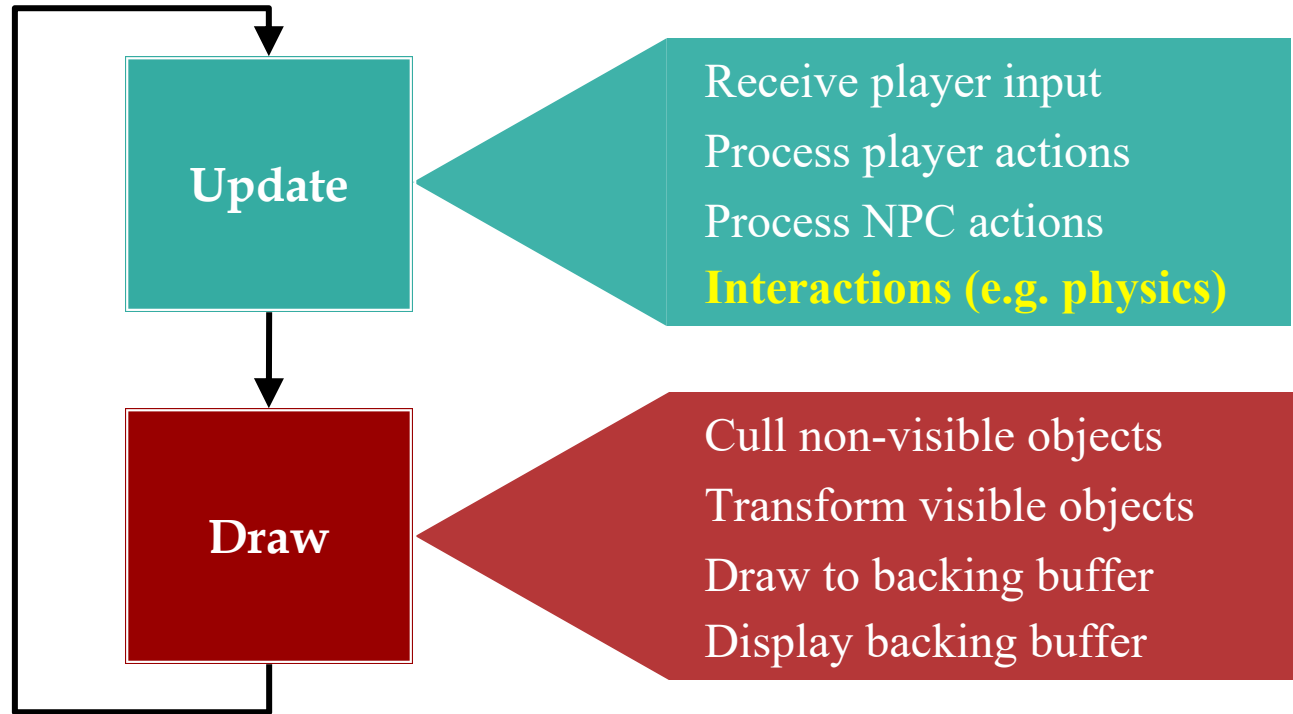
---

- NPCs do not get answer right away
  - Check every loop until answered
  - Remember request; do not ask again
- What to do until then?
  - Act, but don't think!
  - If nothing, **fake** something
  - “Stomping Feet” in RTSs



# The Game Loop

---



# Purpose of a Physics Engine

---

- Moving objects about the screen
  - **Kinematics**: Without regard to external forces
  - **Dynamics**: The effect of forces on the screen
- Collisions between objects
  - **Collision detection**: Did a collision occur?
  - **Collision resolution**: What do we do?
- More on this issue later (~Spring Break)

# Physics Engines: Two Levels

---

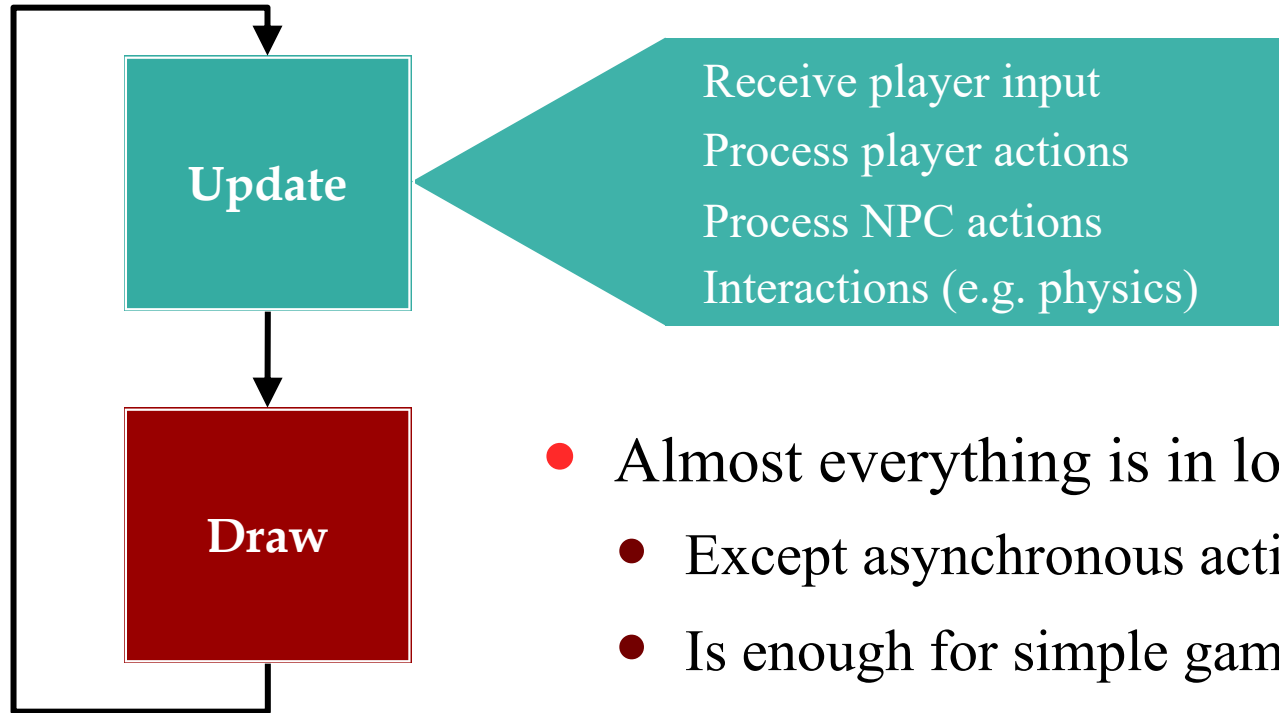
- **White Box:** Engine corrects movement errors
  - Update object state ignoring physics
  - Physics engine nudges object until okay
- **Black Box:** Engine handles everything
  - Do not move objects or update state
  - Give forces, mass, velocities, etc. to engine
  - Engine updates to state that is *close enough*





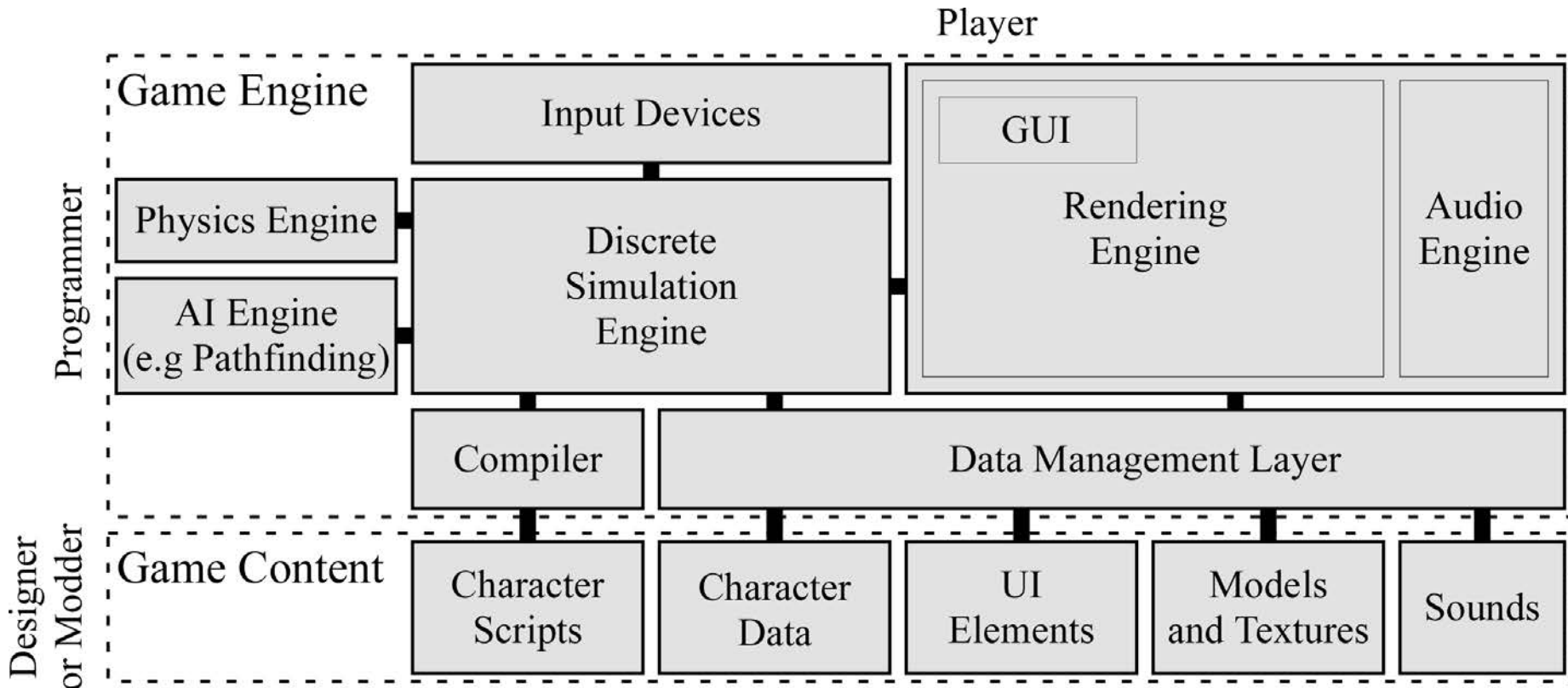
# The Game Loop

---

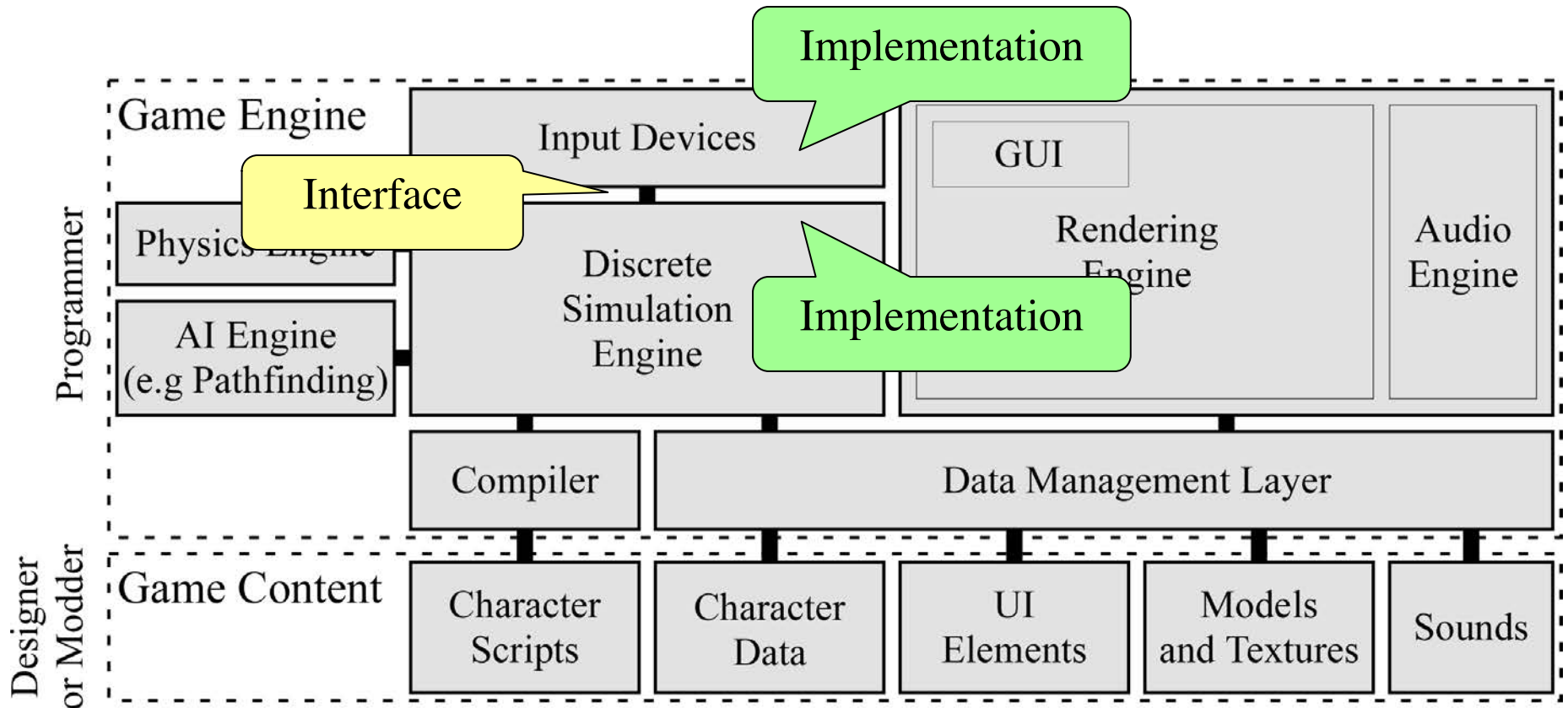


- Almost everything is in loop
  - Except asynchronous actions
  - Is enough for simple games
- How do we organize this loop?
  - Do not want spaghetti code
  - Distribute over programmers

# Architecture: Organizing Your Code



# Architecture: Organizing Your Code



# Where Did This Come From?

