JavaScript Story Task – The Smart Kitchen

🎬 Scene 1: The Kitchen Starts (Event)

Inside a smart kitchen, there is a large touch screen.

The chef walks in and presses a big button:

Start Cooking

Immediately, the kitchen system starts working.

This represents in JavaScript:
An Event occurs → a function is pushed into the Call Stack and starts executing immediately.

Scene 2: Preparing Ingredients (Synchronous)

The kitchen instantly begins:

- Chopping vegetables
- Washing the rice
- Preparing the plates

All of this happens immediately with no waiting.

This represents:
Synchronous tasks executed directly inside the **Call Stack**.

The chef hears:

"Preparing ingredients..."

🍚 Scene 3: Cooking the Rice (Asynchronous)

Now the kitchen puts the rice on the stove.

Rice takes time to cook…
The kitchen will not stop and wait.

It says:

“I will cook the rice, and when it’s done, I’ll let you know.”

This represents:
The task leaves the Call Stack and is placed in the **Event Queue** (Async operation).

Scene 4: Cooking Vegetables (Asynchronous)

The kitchen also starts cooking vegetables.

Vegetables also take time.

The kitchen says again:

“When the vegetables are ready, I’ll notify you.”

Now there are **two async tasks waiting in the Event Queue**.

Scene 5: Call Stack Becomes Empty

The kitchen finishes all the quick tasks.
The **Call Stack is now empty**.

The Event Loop looks at the Event Queue and asks:

“Which task is ready to be executed?”

🍚 Scene 6: Rice is Ready

The rice finishes first.
Its message moves from the Event Queue to the Call Stack:

"Rice cooked"

Scene 7: Vegetables are Ready

Then the vegetables finish.
Their message moves to the Call Stack:

"Vegetables cooked"

🍽 Scene 8: Serve the Dish (Callback)

The kitchen has a rule:

“The dish cannot be served until both rice and vegetables are ready.”

Once both are done, the kitchen announces:

"Dish is ready!"

This represents a **Callback that depends on the completion of two async tasks**.

Scene 9: Making Sauce (Callback Hell)

Suddenly, the chef says:

“Wait! We also need to make sauce!”

The kitchen replies:

“Okay… but the sauce must be prepared after the dish is ready…”

Then another step depends on the sauce, and another step after that…

Things start to get complicated and deeply nested.

This represents:
Callback Hell — multiple dependent async tasks inside each other.

✓ What the Chef Hears (Expected Output)

```
Cooking started!
Preparing ingredients...
Rice cooked
Vegetables cooked
Dish is ready!
Sauce is ready!
```

Hint

Simulate a cooking process where each step represents **sync or async operations**:

1. **User clicks "Start Cooking" button** → triggers an **Event**.
2. **Prepare ingredients** (Synchronous).
3. **Cook rice** (Async, simulate with `setTimeout`) → after done, log `"Rice cooked"`.
4. **Cook vegetables** (Async, simulate with `setTimeout`) → after done, log `"Vegetables cooked"`.
5. **Serve dish** (Callback after rice & vegetables done) → log `"Dish is ready!"`.
6. **Optional:** Add another nested async task (like `"Make sauce"`) to simulate **Callback Hell**.

Bonus how this kitchen process could be improved using **Promises** instead of Callback Hell.