let's walk through this code step-by-step in a **clear, story-like flow** so you can see exactly what's happening and why.

What the code does in plain words

You have:

- A list of items: `["one", "two", "three", "four", "five"]`
- A **limit** (`max = 3`) meaning: *at most 3 items can be "in progress" at the same time*.
- A counter `started` to track how many are currently in progress.
- A `recursive()` function that:
  1. Starts new tasks until it hits the limit or runs out of items.
  2. Each task waits 1 second (`setTimeout`) before printing the item.
  3. When a task finishes, it reduces the `started` count and calls `recursive()` again to start more if possible.

Execution Flow

Initial Call

1. `recursive()` is called for the first time.
2. `while (array.length > 0 && started < max)` → true (5 items, started = 0).
3. Loop runs:
   - **First iteration**:
     - `started` becomes 1.
     - Remove `"one"` from the array.
     - Schedule a `setTimeout` to log `"one"` after 1 second.
   - **Second iteration**:
     - `started` becomes 2.
     - Remove `"two"`.
     - Schedule log `"two"` after 1 second.
   - **Third iteration**:
     - `started` becomes 3.
     - Remove `"three"`.
     - Schedule log `"three"` after 1 second.

4. Now `started = 3` (max reached), so the `while` loop stops.

After 1 second

- The first scheduled timeout fires:
  - Logs `"one"`.
  - `started` becomes 2.
  - Calls `recursive()` again:
    - `array` now has `["four", "five"]`, `started = 2`, so it can start 1 more task.
    - Removes `"four"`, schedules it to log after 1 second, `started = 3` again.
- Almost at the same time, the second timeout fires:
  - Logs `"two"`.
  - `started` becomes 2.
  - Calls `recursive()`:
    - Removes `"five"`, schedules it, `started = 3`.

After 2 seconds

- `"three"` logs, `started` becomes 2, calls `recursive()`:
  - No items left in `array`, so nothing happens.
- `"four"` logs, `started` becomes 2, calls `recursive()`:
  - No items left.
- `"five"` logs, `started` becomes 2, calls `recursive()`:
  - No items left.

Key Points

- **Concurrency limit**: At most `max` items are “in flight” at once.
- **Recursive re-entry**: Each finished task triggers another check to start more work.
- **Order**: Items are processed in the order they’re removed from the array, but because they all have the same delay, the first batch finishes together.

Timeline Summary

Code Copy

```
t=0s: Start "one", "two", "three" (started=3)
t=1s: "one" done → start "four"
      "two" done → start "five"
      "three" done → nothing left
t=2s: "four" done
      "five" done
```