1. Queue - infinitely scalable

2. Event based - more for handling specific events

```javascript
const cuttingQueue = [];
const cookingQueue = [];
const packagingQueue = [];

async function cutBiscuit() {
    while (true) {
        if (cuttingQueue.length > 0) {
            const biscuit = cuttingQueue.shift();
            console.log("Cutting biscuit...");
            // Simulate cutting process
            await new Promise(resolve => setTimeout(resolve, 1000));
            cookingQueue.push(biscuit);
            console.log("Biscuit cut and sent to cooking queue.");
        }
        // Simulate idle time for the function
        await new Promise(resolve => setTimeout(resolve, 100));
    }
}

async function cookBiscuit() {
    while (true) {
        if (cookingQueue.length > 0) {
            const biscuit = cookingQueue.shift();
            console.log("Cooking biscuit...");
            // Simulate cooking process
            await new Promise(resolve => setTimeout(resolve, 1500));
            packagingQueue.push(biscuit);
            console.log("Biscuit cooked and sent to packaging queue.");
        }
        // Simulate idle time for the function
        await new Promise(resolve => setTimeout(resolve, 100));
    }
}

async function packageBiscuit() {
    while (true) {
        if (packagingQueue.length > 0) {
            const biscuit = packagingQueue.shift();
            console.log("Packaging biscuit...");
            // Simulate packaging process
            await new Promise(resolve => setTimeout(resolve, 500));
            console.log("Biscuit packaged.");
            // You can add further steps like storing the packaged biscuit or shipping it out
        }
        // Simulate idle time for the function
        await new Promise(resolve => setTimeout(resolve, 100));
    }
}
```

```
async function makeBiscuits(numRequests) {
    for (let i = 0; i < numRequests; i++) {
        const biscuit = `Biscuit ${i + 1}`;
        cuttingQueue.push(biscuit);
        console.log(`${biscuit} added to cutting queue.`);
        // Simulate delay between requests
        await new Promise(resolve => setTimeout(resolve, 500));
    }
}

async function simulateProduction(numRequests) {
    console.log(`Starting production with ${numRequests} biscuits.`);
    // Start all stages concurrently
    const cutPromise = cutBiscuit();
    const cookPromise = cookBiscuit();
    const packagePromise = packageBiscuit();

    // Make external requests
    await makeBiscuits(numRequests);

    // Wait for all stages to finish
    await Promise.all([cutPromise, cookPromise, packagePromise]);
    console.log("All biscuits processed.");
}

// Run the simulation with 10 biscuits
simulateProduction(10);
```

```
const EventEmitter = require('events');                                    javascript

// Create an event emitter instance
const eventEmitter = new EventEmitter();

// Event names
const CUTTING_FINISHED = 'cuttingFinished';
const COOKING_FINISHED = 'cookingFinished';
const PACKAGING_FINISHED = 'packagingFinished';

async function cutBiscuit(biscuit) {
    console.log("Cutting biscuit...");
    // Simulate cutting process
    await new Promise(resolve => setTimeout(resolve, 1000));
    console.log("Biscuit cut.");
    // Emit cutting finished event
    eventEmitter.emit(CUTTING_FINISHED, biscuit);
}

async function cookBiscuit(biscuit) {
    console.log("Cooking biscuit...");
```

```
    // Simulate cooking process
    await new Promise(resolve => setTimeout(resolve, 1500));
    console.log("Biscuit cooked.");
    // Emit cooking finished event
    eventEmitter.emit(COOKING_FINISHED, biscuit);
}

async function packageBiscuit(biscuit) {
    console.log("Packaging biscuit...");
    // Simulate packaging process
    await new Promise(resolve => setTimeout(resolve, 500));
    console.log("Biscuit packaged.");
    // Emit packaging finished event
    eventEmitter.emit(PACKAGING_FINISHED, biscuit);
}

// Subscribe to events
eventEmitter.on(CUTTING_FINISHED, biscuit => {
    console.log(`Cutting of ${biscuit} finished. Sending to cooking...`);
    cookBiscuit(biscuit);
});

eventEmitter.on(COOKING_FINISHED, biscuit => {
    console.log(`Cooking of ${biscuit} finished. Sending to packaging...`);
    packageBiscuit(biscuit);
});

eventEmitter.on(PACKAGING_FINISHED, biscuit => {
    console.log(`Packaging of ${biscuit} finished.`);
    // You can add further steps like storing the packaged biscuit or shipping it out
});

async function makeBiscuits(numRequests) {
    for (let i = 0; i < numRequests; i++) {
        const biscuit = `Biscuit ${i + 1}`;
        console.log(`${biscuit} added to cutting queue.`);
        // Start cutting process
        cutBiscuit(biscuit);
        // Simulate delay between requests
        await new Promise(resolve => setTimeout(resolve, 500));
    }
}

async function simulateProduction(numRequests) {
    console.log(`Starting production with ${numRequests} biscuits.`);
    // Make external requests
    await makeBiscuits(numRequests);
    console.log("All biscuits processed.");
}

// Run the simulation with 10 biscuits
simulateProduction(10);
```