◇ 1. Promise.all()

☑ Waits for **all promises to resolve**, else **fails fast on first reject**

☑ All success

```
const p1 = new Promise(res => setTimeout(() => res("☑ p1 (3s)"), 3000));
const p2 = new Promise(res => setTimeout(() => res("☑ p2 (1s)"), 1000));
const p3 = new Promise(res => setTimeout(() => res("☑ p3 (2s)"), 2000));

console.time("⏱ all");
Promise.all([p1, p2, p3])
  .then(results => {
    console.timeEnd("⏱ all"); // ~3s
    console.log("☑ All resolved:", results);
  })
  .catch(error => {
    console.timeEnd("⏱ all");
    console.log("❌ Rejected:", error);
  });
```

⏱ **Time taken**: 3s (maximum of all) 🔄 All run in parallel 📌 If all resolve, you get an array of values 👄 If any reject, it **fails immediately**, ignoring later results

---

❌ One fails

```
const p1 = new Promise(res => setTimeout(() => res("☑ p1 (3s)"), 3000));
const p2 = new Promise((_, rej) => setTimeout(() => rej("❌ p2 failed (1s)"),
1000));
const p3 = new Promise(res => setTimeout(() => res("☑ p3 (2s)"), 2000));

console.time("⏱ all (fail)");
Promise.all([p1, p2, p3])
  .then(console.log)
  .catch(error => {
    console.timeEnd("⏱ all (fail)"); // ~1s
    console.log("❌ Caught early reject:", error);
  });
```

⏱ **Time taken**: ~1s (fails fast) ⚠ Other promises **continue**, but their results are ignored

---

◇ 2. Promise.allSettled()

🔘 Waits for **all to settle** (resolved or rejected)

```
const p1 = new Promise(res => setTimeout(() => res("✅ p1 (3s)"), 3000));
const p2 = new Promise((_, rej) => setTimeout(() => rej("❌ p2 (1s)"), 1000));
const p3 = new Promise(res => setTimeout(() => res("✅ p3 (2s)"), 2000));

console.time("⏱ allSettled");
Promise.allSettled([p1, p2, p3])
  .then(results => {
    console.timeEnd("⏱ allSettled"); // ~3s
    console.log("🟢 Settled results:", results);
  });
```

📜 **Output**:

```
[
  { status: 'fulfilled', value: '✅ p1 (3s)' },
  { status: 'rejected', reason: '❌ p2 (1s)' },
  { status: 'fulfilled', value: '✅ p3 (2s)' }
]
```

⏱ **Time taken**: 3s (waits for all) 🔁 All run in parallel ✅ You get full result status — perfect for batch jobs/logging ❌ Never throws

---

◇ 3. `Promise.race()`

🏁 Resolves or rejects with **first settled promise**

✅ First to resolve

```
const p1 = new Promise(res => setTimeout(() => res("✅ p1 (3s)"), 3000));
const p2 = new Promise(res => setTimeout(() => res("✅ p2 (1s)"), 1000));
const p3 = new Promise(res => setTimeout(() => res("✅ p3 (2s)"), 2000));

console.time("⏱ race");
Promise.race([p1, p2, p3])
  .then(result => {
    console.timeEnd("⏱ race"); // ~1s
    console.log("🏁 Winner:", result);
  });
```

⏱ **Time taken**: 1s ✅ First success wins 🔁 Others continue but are ignored

---

❌ First to reject

```
const p1 = new Promise(res => setTimeout(() => res("☑ p1 (3s)"), 3000));
const p2 = new Promise((_, rej) => setTimeout(() => rej("❌ p2 failed (1s)"),
1000));
const p3 = new Promise(res => setTimeout(() => res("☑ p3 (2s)"), 2000));

console.time("⏱ race (fail)");
Promise.race([p1, p2, p3])
  .then(console.log)
  .catch(error => {
    console.timeEnd("⏱ race (fail)"); // ~1s
    console.log("🚫 First rejected:", error);
  });
```

⏱ **Time taken**: 1s ❌ First reject short-circuits race 📌 Useful for implementing timeouts or early exits

---

◇ 4. `Promise.any()`

🏅 Resolves with **first fulfilled**, ignores rejections

☑ First fulfilled

```
const p1 = new Promise((_, rej) => setTimeout(() => rej("❌ p1 failed (3s)"),
3000));
const p2 = new Promise(res => setTimeout(() => res("☑ p2 (1s)"), 1000));
const p3 = new Promise(res => setTimeout(() => res("☑ p3 (2s)"), 2000));

console.time("⏱ any");
Promise.any([p1, p2, p3])
  .then(result => {
    console.timeEnd("⏱ any"); // ~1s
    console.log("🎯 First Fulfilled:", result);
  });
```

⏱ **Time taken**: 1s ☑ First resolved value returned 🔁 Rejections are **ignored** unless all fail

---

❌ All fail (Edge Case)

```
const p1 = new Promise((_, rej) => setTimeout(() => rej("❌ p1 (1s)"), 1000));
const p2 = new Promise((_, rej) => setTimeout(() => rej("❌ p2 (2s)"), 2000));
const p3 = new Promise((_, rej) => setTimeout(() => rej("❌ p3 (3s)"), 3000));

console.time("⏱ any (fail)");
Promise.any([p1, p2, p3])
  .then(console.log)
  .catch(error => {
    console.timeEnd("⏱ any (fail)"); // ~3s
```

```
  console.log("💥 All failed:", error.name); // AggregateError
  console.error(error.errors); // Array of reasons
});
```

⏱️ **Time taken**: 3s (waits till all fail) ❌ Throws `AggregateError` with all reasons 📌 Ideal when **any one** success is enough

---

☑️ Summary Table

Method	Resolves With	Rejects When	Time Taken	Waits for All?	Ignores Failures?
`Promise.all()`	All fulfilled values	❌ If *any* fails	Max time	☑️ Yes	❌ No
`Promise.allSettled()`	Array of `{status, value}`	❌ Never rejects	Max time	☑️ Yes	☑️ Yes
`Promise.race()`	First to settle (any)	❌ If first is rejected	Fastest settle	❌ No	❌ No
`Promise.any()`	First fulfilled	❌ If *all* fail (AggregateError)	Fastest success / All fail	❌ No	☑️ Yes

---

real-world use cases

◇ `Promise.all()`

➡️ **Use when all async tasks are required to succeed** before proceeding.

☑️ Use Cases:

1. **Loading dashboard data**: Fetch user profile, notifications, and analytics *all together*.

```
Promise.all([fetchUser(), fetchNotifications(), fetchAnalytics()])
```

   - Proceed only if **all succeed**.
   - Fail fast if any fail.

2. **Uploading multiple files together**

```
Promise.all(files.map(file => uploadFile(file)))
```

- Show "Upload successful" only if **every file uploads**.

3. **Dependency loading** (e.g., scripts, styles, configs)

```
Promise.all([loadScript(), loadStyles(), loadConfig()])
```

- Render page only after **all assets are ready**.

◇ Promise.allSettled()

➡️ **Use when you want to know results of all promises**, regardless of success/failure.

🟢 Use Cases:

1. **Bulk operations report**:
   - Run multiple API calls and **log all their statuses**, even if some fail.

```
Promise.allSettled(users.map(u => sendEmail(u.email)))
```

2. **Form field validation** (non-blocking):
   - Run validations for fields (sync + async), then **show per-field result**.

```
Promise.allSettled([
  validateUsername(name),
  validateEmail(email),
  validatePhone(phone)
])
```

3. **Non-critical background tasks**:
   - Like analytics, logging, etc. where failure shouldn't block the app.

◇ Promise.race()

➡️ **Use when you care only about the fastest result**, regardless of success or failure.

🏁 Use Cases:

1. **Timeout fallback**:
   - If API call takes too long, fail early.

```
Promise.race([
  fetchData(),
  timeout(5000) // custom promise that rejects in 5s
])
```

2. **First response wins**:
   - Query multiple CDNs or mirrors and use whichever responds first.

```
Promise.race([fetchFromCDN1(), fetchFromCDN2(), fetchFromCDN3()])
```

3. **User interaction race**:
   - Wait for first user action: key press, mouse move, or scroll.

```
Promise.race([
  once(document, 'keydown'),
  once(document, 'mousemove'),
  once(document, 'scroll')
])
```

◇ Promise.any()

➡️ **Use when you only need one to succeed**, and don't care which.

🥇 Use Cases:

1. **CDN redundancy**:
   - Try multiple sources for a script/image — load from first that **succeeds**.

```
Promise.any([
  fetchFromCDN1(),
  fetchFromCDN2(),
  fetchFromCDN3()
])
```

2. **Authentication fallback**:
   - Try different auth strategies (e.g., cookie, token, session).

```
Promise.any([
  checkCookie(),
```

```
  checkLocalStorage(),
  checkSSO()
])
```

3. **Search in parallel services**:
    - Send search query to multiple APIs (e.g., DuckDuckGo, Bing, Google) and use first valid result.

🧠 Summary Table

Combinator	Best When You Want To...	Ignores Errors?	Short-circuits?
`Promise.all()`	Wait for **all to succeed** or fail fast	❌ No	☑ Yes (on reject)
`Promise.allSettled()`	Get results of **all**, succeed or fail	☑ Yes	❌ No
`Promise.race()`	Get **first to settle**, regardless of outcome	❌ No	☑ Yes
`Promise.any()`	Get **first to succeed**, ignore failures	☑ Yes	☑ Yes (on first success)