Got it — you want an **Angular / RxJS** solution where `combineLatest` emits **only when at least one source changes to a new value** and **skips emissions if the combined result is the same as the previous one**.

This is a common pattern to avoid unnecessary API calls or UI updates.

✅ Recommended RxJS Pattern

You can achieve this by combining:

- `combineLatest` — to merge multiple streams.
- `distinctUntilChanged` — to skip emissions when the combined output is the same as the previous one.
- **Custom comparator** — to deeply compare arrays/objects.

Example: Skip When Combined Values Are Equal

```
import { combineLatest } from 'rxjs';
import { distinctUntilChanged } from 'rxjs/operators';
import { isEqual } from 'lodash-es'; // or write your own deep
compare

// Example observables (replace with your selectors or streams)
const obs1$ = store.select(selector1);
const obs2$ = store.select(selector2);
const obs3$ = store.select(selector3);

combineLatest([obs1$, obs2$, obs3$]).pipe(
  // Skip if the combined array is the same as the previous emission
  distinctUntilChanged((prev, curr) => isEqual(prev, curr))
).subscribe(([val1, val2, val3]) => {
  console.log('Values changed:', val1, val2, val3);
  // Trigger API call or update logic here
});
```

🔹 Without Lodash (Lightweight Deep Compare)

If you don't want to import `lodash-es`, you can use:

```
function deepEqual(a: any, b: any): boolean {
  return JSON.stringify(a) === JSON.stringify(b);
}

combineLatest([obs1$, obs2$, obs3$]).pipe(
  distinctUntilChanged((prev, curr) => deepEqual(prev, curr))
).subscribe(values => {
  console.log('Changed:', values);
});
```

🔹 If You Only Care About Primitive Values

If all streams emit **primitives** (numbers, strings, booleans), you can use:

```
combineLatest([obs1$, obs2$, obs3$]).pipe(
  distinctUntilChanged((prev, curr) =>
    prev.every((val, i) => val === curr[i])
  )
).subscribe(values => {
  console.log('Changed:', values);
});
```

📌 Notes

- `combineLatest` emits immediately after **all sources have emitted at least once**.
- `distinctUntilChanged` compares **the entire emitted array** — so if only one value changes, it will still emit.
- For **null/undefined filtering**, you can add `.pipe(filter(values => values.every(v => v != null)))` before `distinctUntilChanged`.

If you want, I can give you a **version that waits until all values are non-null and then emits only once** — perfect for initial API calls.
Do you want me to prepare that?