Angular Change Detection Summary

1. Default Strategy (ChangeDetectionStrategy.Default)

---

- Behavior: Angular automatically checks for changes in the parent component and reflects them in the child.
- Example:
  - AppComponent (Parent) has a data array and passes it to ChildComponent via @Input().
  - When AppComponent updates the data (e.g., via an event or timeout), the ChildComponent automatically re-renders.
- No need for manual change detection.

2. OnPush Strategy (ChangeDetectionStrategy.OnPush)

---

- Behavior: Angular only checks for changes when:
  - @Input() reference changes
  - Events are triggered (e.g., button click)
  - Manually invoking ChangeDetectorRef.detectChanges() or markForCheck()
- Example:
  - AppComponent has an array and uses OnPush.
  - If the array is mutated (e.g., push), UI does not reflect changes.
  - If a new array is assigned (new reference), changes are detected.

3. OnPush Strategy with Manual Change Detection

---

- Scenario: Data is updated via setTimeout or HTTP call.

- To reflect changes in UI:

  - Trigger an Angular event (e.g., button click)

  - Or use: ChangeDetectorRef.markForCheck() / detectChanges()

4. Using BehaviorSubject with OnPush

---

- Behavior: Ideal for reactive patterns.

- Changes to BehaviorSubject emit new values.

- If asyncPipe is used in the template, change detection works automatically.

  (Because asyncPipe internally triggers CD)

- Example:

  - serviceData$ = new BehaviorSubject<Data[]>([]);

  - Update with next(): serviceData$.next([...]);

  - Use *ngFor="let item of serviceData$ | async"

Conclusion:

---

- OnPush gives performance benefits but needs explicit cues.

- Default strategy is easier but less optimized.

- Observables (BehaviorSubject) + asyncPipe + OnPush is a powerful and clean combo.

Tip:

---

Always prefer immutability (e.g., [...array, newItem]) when using OnPush.